GKTCS INNOVATIONS

Actions
&
RDD Transformations

Actions

GKTCS INNOVATIONS

Actions, in contrast to transformations, execute the scheduled task on the dataset; once you have finished transforming your data you can execute your transformations.

The .take(...) method

This is most arguably the most useful (and used, such as the .map(...) method). The method is preferred to .collect(...) as it only returns the n top rows from a single data partition in contrast to .collect(...), which returns the whole RDD. This is especially important when you deal with large datasets:

```
data_first = data_from_file_conv.take(1)
```

Actions

GKTCS INNOVATIONS

If you want somewhat randomized records you can use .takeSample(...) instead, which takes three arguments: First whether the sampling should be with replacement, the second specifies the number of records to return, and the third is a seed to the pseudo-random numbers generator:

```
data_take_sampled = data_from_file_conv.takeSample(False, 1, 667)
```

Actions

GKTCS INNOVATIONS

The .collect(...) method

This method returns all the elements of the RDD to the driver. As we have just provided a caution about it, we will not repeat ourselves here.

The .reduce(...) method

The .reduce(...) method reduces the elements of an RDD using a specified method.

You can use it to sum the elements of your RDD:

```
rdd1.map(lambda row: row[1]).reduce(lambda x, y: x + y)
```

This will produce the sum of 15.

Actions

GKTCS INNOVATIONS

We first create a list of all the values of the rdd1 using the .map(...) transformation, and then use the .reduce(...) method to process the results. The reduce(...) method, on each partition, runs the summation method (here expressed as a lambda) and returns the sum to the driver node where the final aggregation takes place.

The .reduceByKey(...) method works in a similar way to the .reduce(...) method, but it performs a reduction on a key-by-key basis:

```
data_key = sc.parallelize(
[('a', 4),('b', 3),('c', 2),('a', 8),('d', 2),('b', 1),
('d', 3)],4)
data_key.reduceByKey(lambda x, y: x + y).collect()
```

The preceding code produces the following:

```
Out[122]: [('b', 4), ('c', 2), ('a', 12), ('d', 5)]
```

Actions

GKTCS INNOVATIONS

The .count(...) method

The .count(...) method counts the number of elements in the RDD. Use the following code:

```
data_reduce.count()
```

This code will produce 6, the exact number of elements in the data_reduce RDD.

The .count(...) method produces the same result as the following method, but it does not require moving the whole dataset to the driver:

```
len(data_reduce.collect()) # WRONG -- DON'T DO THIS!
```

Actions

GKTCS INNOVATIONS

If your dataset is in a key-value form, you can use the .countByKey() method to get the counts of distinct keys. Run the following code:

```
data_key.countByKey().items()
```

This code will produce the following output:

```
Out[132]: dict_items([('a', 2), ('b', 2), ('d', 2), ('c', 1)])
```

Actions

GKTCS INNOVATIONS

The .saveAsTextFile(...) method

As the name suggests, the .saveAsTextFile(...) the RDD and saves it to text files: Each partition to a separate file:

```
data_key.saveAsTextFile(
'/Users/drabast/Documents/PySpark_Data/data_key.txt')
```

Actions

GKTCS INNOVATIONS

To read it back, you need to parse it back as all the rows are treated as strings:

```
def parseInput(row):
    import re
    pattern = re.compile(r'\(\'([a-z])\', ([0-9])\)')
    row_split = pattern.split(row)
    return (row_split[1], int(row_split[2]))
data_key_reread = sc .textFile(
'/Users/drabast/Documents/PySpark_Data/data_key.txt').map(parse
Input)
data_key_reread.collect()
```

The list of keys read matches what we had initially:

```
Out[159]: [('a', 4), ('b', 3), ('c', 2), ('a', 8), ('d', 2), ('b', 1), ('d', 3)]
```

Actions

GKTCS INNOVATIONS

The .foreach(...) method

This is a method that applies the same function to each element of the RDD in an iterative way; in contrast to .map(..), the .foreach(...) method applies a defined function to each record in a one-by-one fashion.

It is useful when you want to save the data to a database that is not natively supported by PySpark.

Here, we'll use it to print (to CLI - not the Jupyter Notebook) all the records that are stored in data_key RDD:

```
def f(x):
print(x)
data_key.foreach(f)
```

If you now navigate to CLI you should see all the records printed out. Note, that every time the order will most likely be different.

Iterative Operations on Spark RDD

GKTCS INNOVATIONS

The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

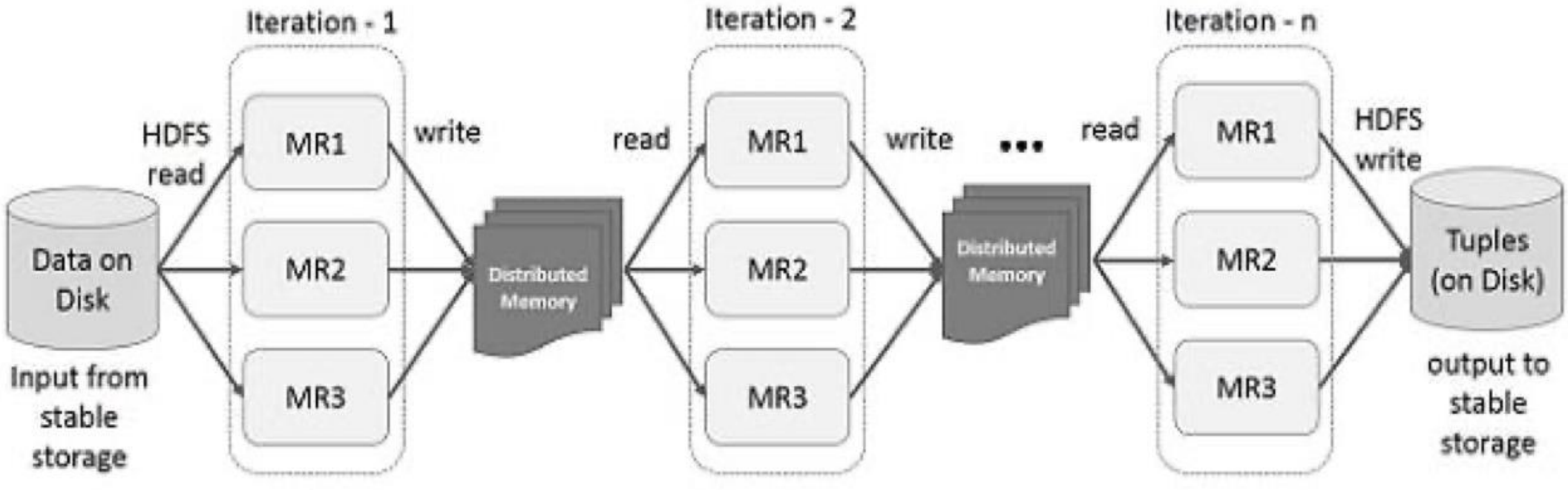
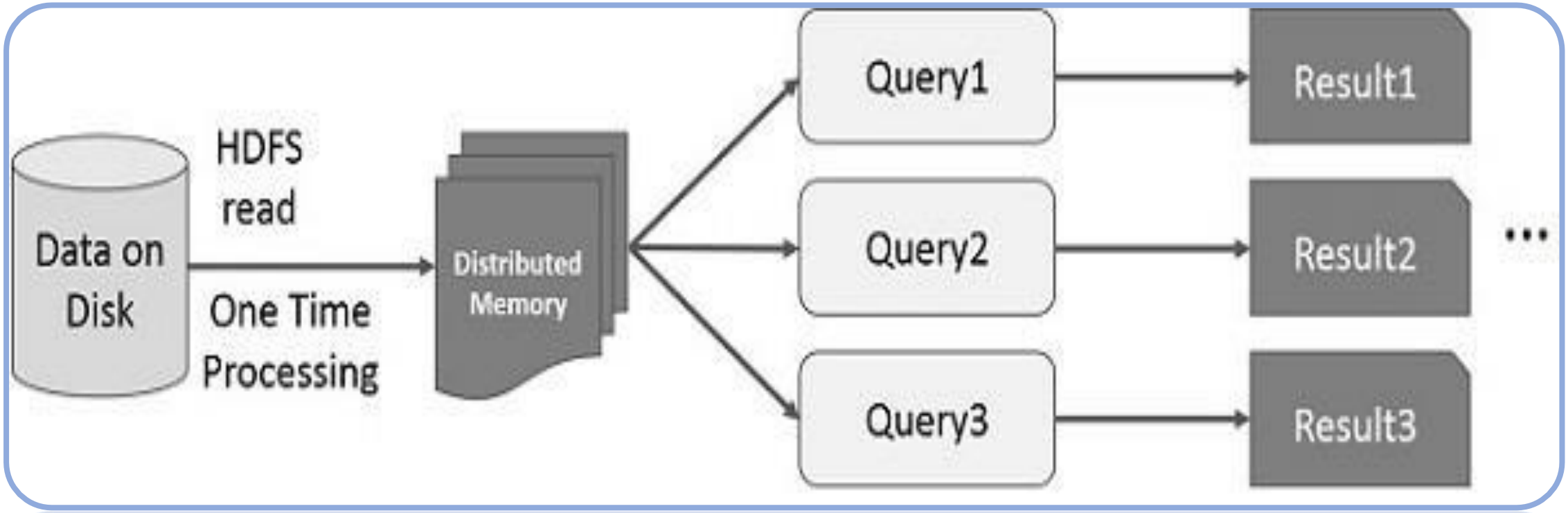


Figure: Iterative operations on Spark RDD

Iterative Operations on Spark RDD

GKTCS INNOVATIONS

This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



Spark Shell

GKTCS INNOVATIONS

Spark provides an interactive shell: a powerful tool to analyze data interactively.

It is available in either Scala or Python language.

Spark's primary abstraction is a distributed collection of items called a Resilient Distributed Dataset (RDD).

RDDs can be created from Hadoop Input Formats (such as HDFS files) or by transforming other RDDs.

Open Spark Shell

The following command is used to open Spark shell.

```
$ spark-shell
```

```
scala> val inputfile = sc.textFile("input.txt")
```

RDD Transformations

GKTCS INNOVATIONS

The Spark RDD API introduces few Transformations and few Actions to manipulate RDD.

RDD Transformations

RDD transformations returns pointer to new RDD and allows you to create dependencies between RDDs.

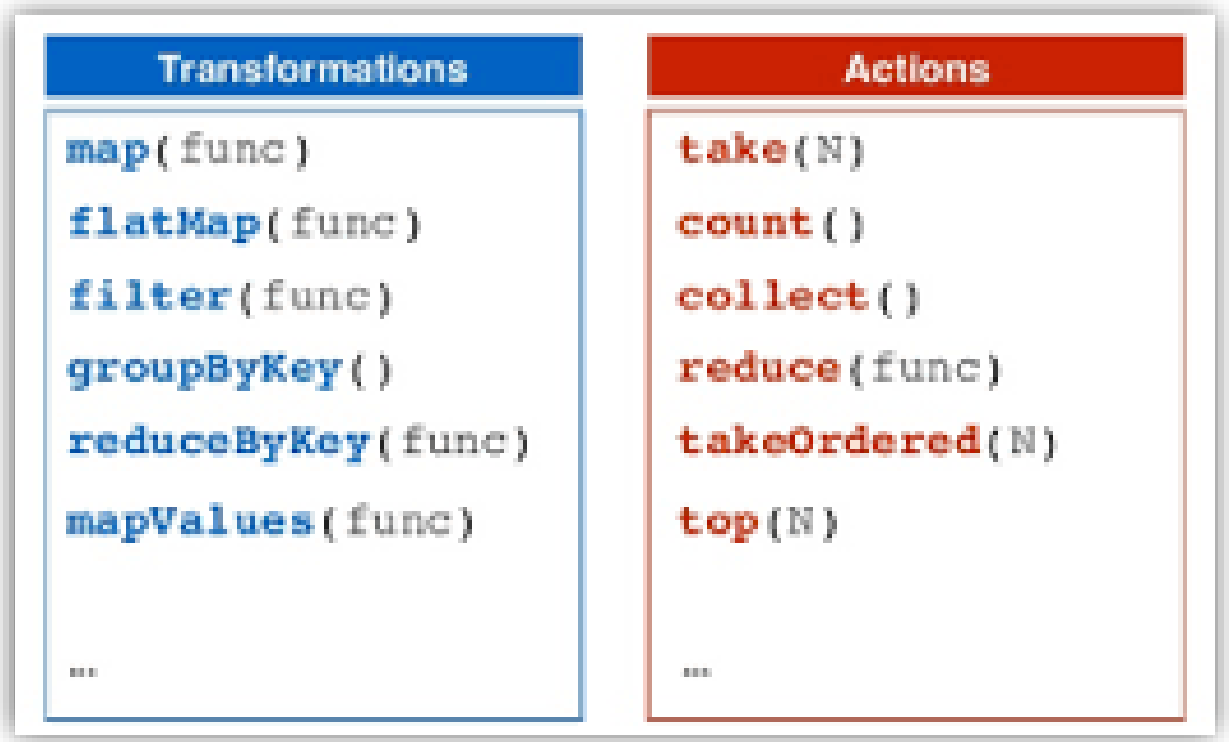
Each RDD in dependency chain (String of Dependencies) has a function for calculating its data and has a pointer (dependency) to its parent RDD.

RDD Transformations

GKTCS INNOVATIONS

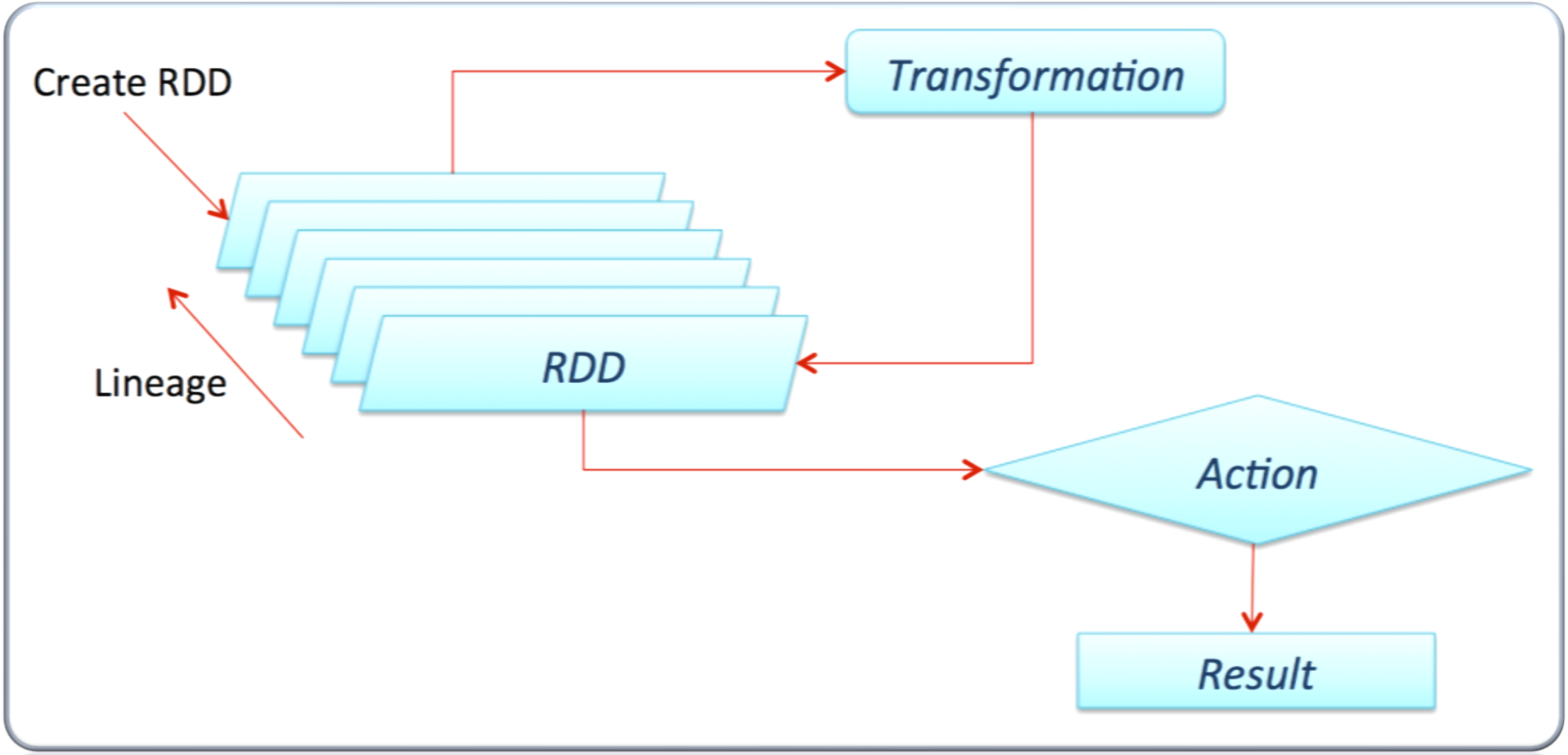
Spark is lazy, so nothing will be executed unless you call some transformation or action that will trigger job creation and execution.

Therefore, RDD transformation is not a set of data but is a step in a program (might be the only step) telling Spark how to get data and what to do with it.



Iterative Operations on Spark RDD

GKTCS INNOVATIONS



RDD Transformations

GKTCS INNOVATIONS

Sr.	Transformation and Meaning
1	**map(func)** Returns a new distributed dataset, formed by passing each element of the source through a function func.
2	**filter(func)** Returns a new dataset formed by selecting those elements of the source on which func returns true.
3	**flatMap(func)** Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
4	**mapPartitions(func)** Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.

RDD Transformations

GKTCS INNOVATIONS

Sr.	Transformation and Meaning
5	**mapPartitionsWithIndex(func)** Similar to map Partitions, but also provides func with an integer value representing the index of the partition, so func must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
6	**sample(withReplacement, fraction, seed)** Sample a fraction of the data, with or without replacement, using a given random number generator seed.
7	**union(otherDataset)** Returns a new dataset that contains the union of the elements in the so dataset and the argument.

RDD Transformations

GKTCS INNOVATIONS

Sr.	Transformation and Meaning
8	**intersection(otherDataset)** Returns a new RDD that contains the intersection of elements in the source dataset and the argument.
9	**distinct([numTasks]))** Returns a new dataset that contains the distinct elements of the source dataset.
10	**groupByKey([numTasks])** When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or aggregateByKey will yield much better performance.

RDD Transformations

GKTCS INNOVATIONS

Sr.	Transformation and Meaning
11	**reduceByKey(func, [numTasks])** When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V, V) => V.
12	**aggregateByKey(zeroValue)(seqOp, combOp, [numTasks])** When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different from the input value type, while avoiding unnecessary allocations.
13	**sortByKey([ascending], [numTasks])** When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the Boolean ascending argument.

RDD Transformations

GKTCS INNOVATIONS

Sr.	Transformation and Meaning
14	**join(otherDataset, [numTasks])** When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.
15	**cogroup(otherDataset, [numTasks])** When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (Iterable<V>, Iterable<W>)) tuples. This operation is also called group With.
16	**cartesian(otherDataset)** When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).

RDD Transformations

GKTCS INNOVATIONS

Sr.	Transformation and Meaning
17	**pipe(command, [envVars])** Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
18	**coalesce(numPartitions)** Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
19	**repartition(numPartitions)** Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
20	**epartitionAndSortWithinPartitions(partitioner)** Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling repartition and then sorting within each partition because it can push the sorting down into the shuffle machinery.

RDD Actions

GKTCS INNOVATIONS

Sr.	Actions and Meaning
1	**reduce(func)** Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
2	**collect()** Returns all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
3	**count()** Returns the number of elements in the dataset.
4	**first()** Returns the first element of the dataset (similar to take (1)).

RDD Actions

GKTCS INNOVATIONS

Sr.	Actions and Meaning
5	**take(n)** Returns an array with the first n elements of the dataset.
6	**takeSample (withReplacement,num, [seed])** Returns an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
7	**takeOrdered(n, [ordering])** Returns the first n elements of the RDD using either their natural order or a custom comparator.
8	**saveAsTextFile(path)** Writes the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark calls toString on each element to convert it to a line of text in the file.

RDD Actions

GKTCS INNOVATIONS

Sr.	Actions and Meaning
9	**saveAsSequenceFile(path) (Java and Scala)** Writes the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
10	**saveAsObjectFile(path) (Java and Scala)** Writes the elements of the dataset in a simple format using Java serialization, which can then be loaded using SparkContext.objectFile().
11	**countByKey()** Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
12	**foreach(func)** Runs a function func on each element of the dataset. This is usually, done for side effects such as updating an Accumulator or interacting with external storage systems.