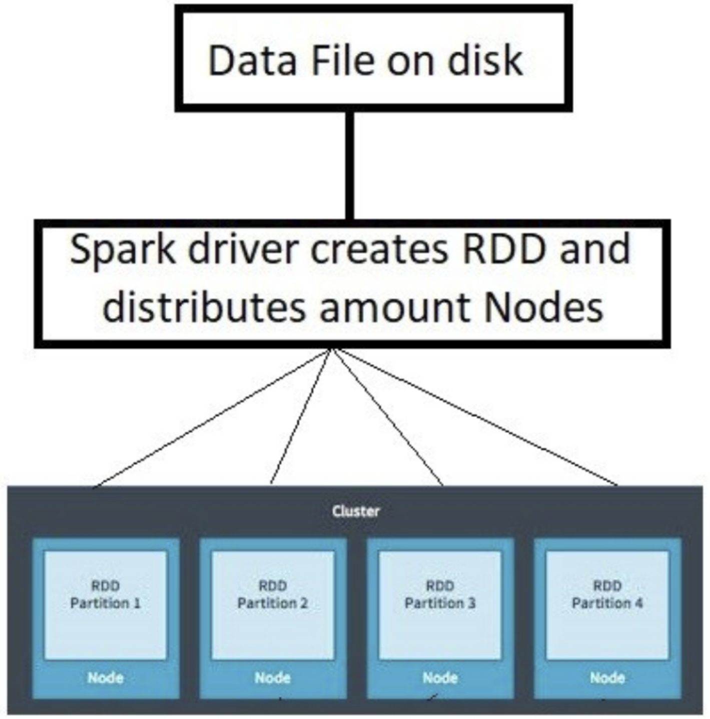
Introduction to PySpark RDD

BIG DATA FUNDAMENTALS VIA PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

What is RDD?

- RDD = **R**esilient **D**istributed **D**atasets



Data File on disk

Spark driver creates RDD and distributes amount Nodes

Decomposing RDDs

- Resilient Distributed Datasets
  - Resilient: Ability to withstand failures
  - Distributed: Spanning across multiple machines
  - Datasets: Collection of partitioned data e.g, Arrays, Tables, Tuples etc.,

Creating RDDs. How to do it?

- Parallelizing an existing collection of objects
- External datasets:
  - Files in HDFS
  - Objects in Amazon S3 bucket
  - lines in a text file
- From existing RDDs

Parallelized collection (parallelizing)

- `parallelize()` for creating RDDs from python lists

```
numRDD = sc.parallelize([1,2,3,4])
```

```
helloRDD = sc.parallelize("Hello world")
```

```
type(helloRDD)
```

```
<class 'pyspark.rdd.PipelinedRDD'>
```

From external datasets

- `textFile()` for creating RDDs from external datasets

```
fileRDD = sc.textFile("README.md")
```

```
type(fileRDD)
```

```
<class 'pyspark.rdd.PipelinedRDD'>
```

Understanding Partitioning in PySpark

- A partition is a logical division of a large distributed data set
- `parallelize()` method

```
numRDD = sc.parallelize(range(10), minPartitions = 6)
```

- `textFile()` method

```
fileRDD = sc.textFile("README.md", minPartitions = 6)
```

- The number of partitions in an RDD can be found by using `getNumPartitions()` method

Let's practice

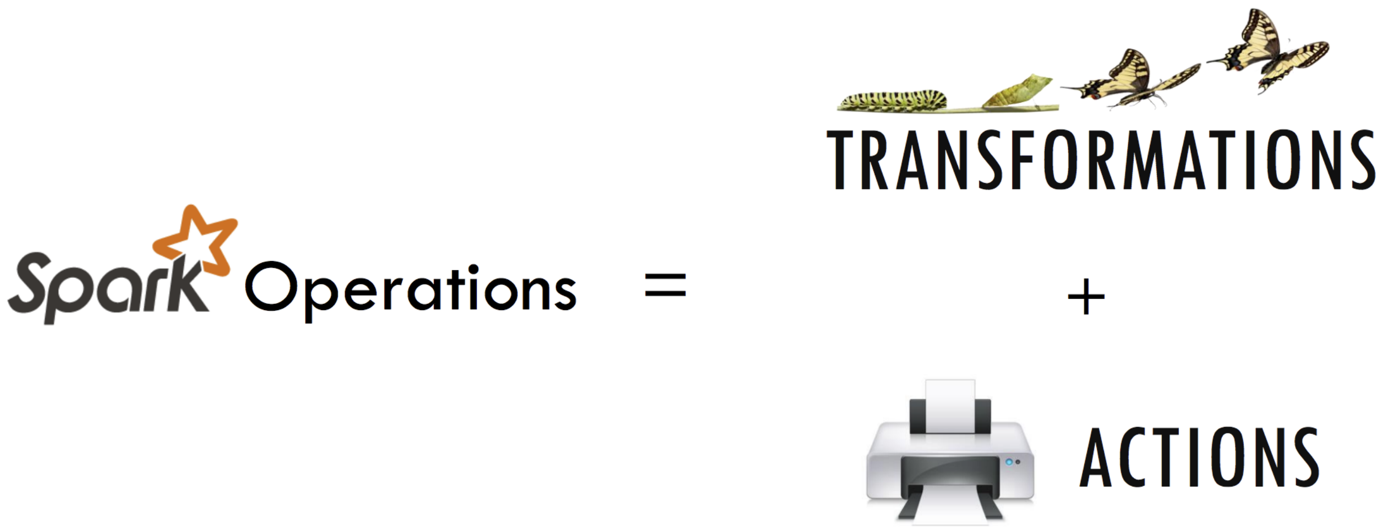
BIG DATA FUNDAMENTALS VIA PYSPARK

RDD operations in PySpark

BIG DATA FUNDAMENTALS VIA PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

Overview of PySpark operations

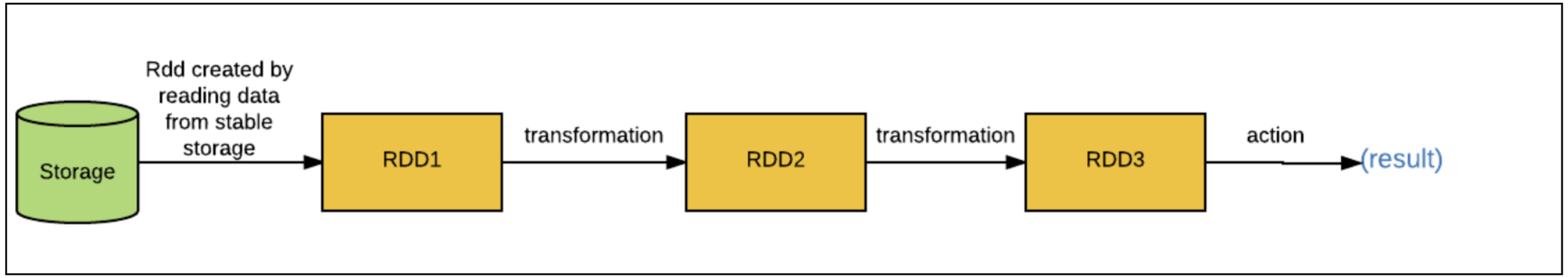


Spark Operations = TRANSFORMATIONS + ACTIONS

- Transformations create new RDDS
- Actions perform computation on the RDDs

RDD Transformations

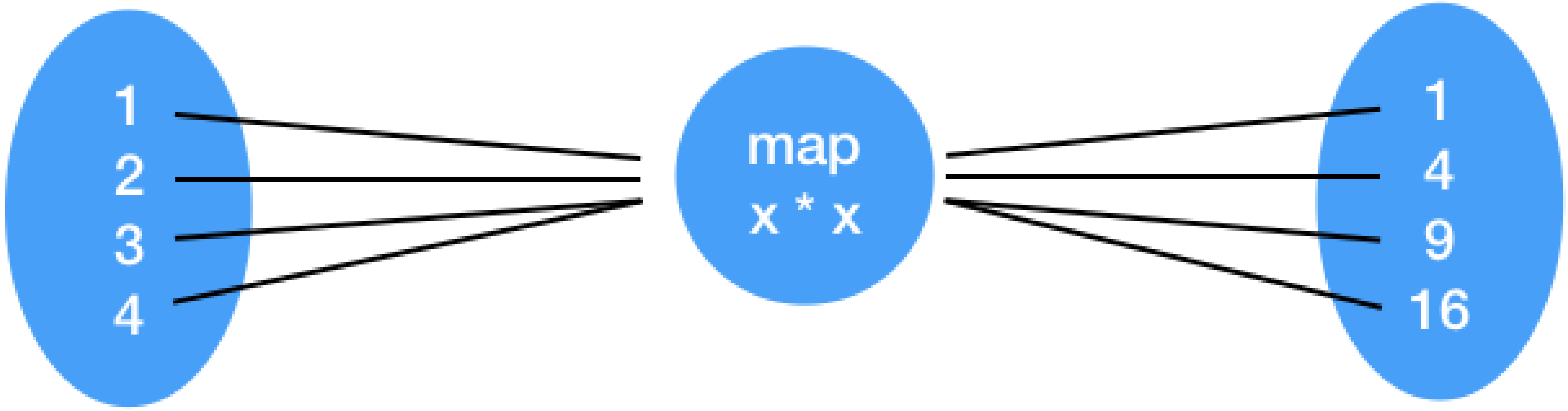
- Transformations follow Lazy evaluation



- Basic RDD Transformations
  - `map()` , `filter()` , `flatMap()` , and `union()`

map() Transformation

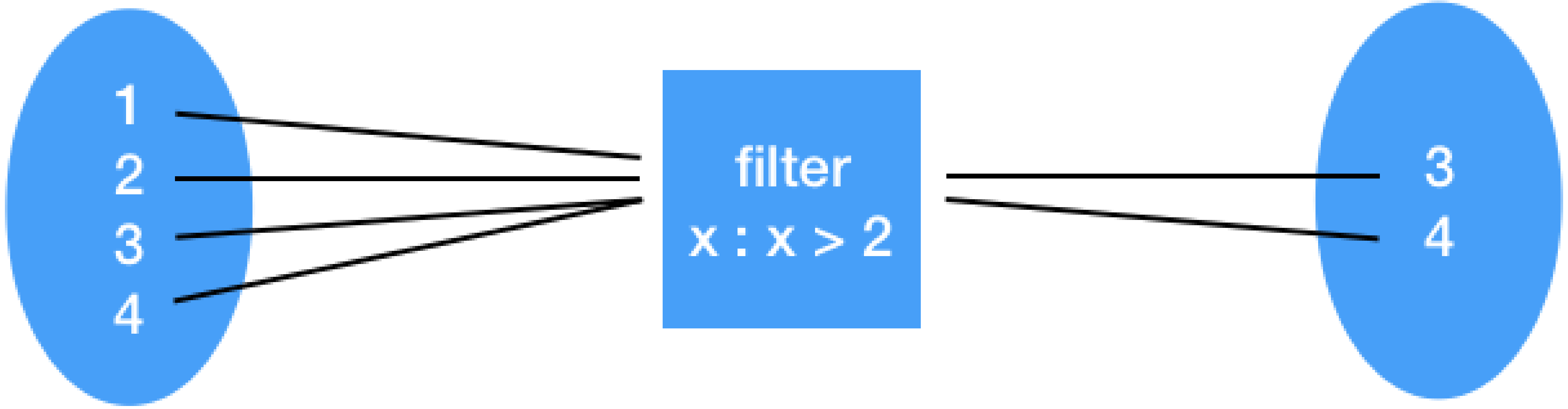
- map() transformation applies a function to all elements in the RDD



```
RDD = sc.parallelize([1,2,3,4])
RDD_map = RDD.map(lambda x: x * x)
```

filter() Transformation

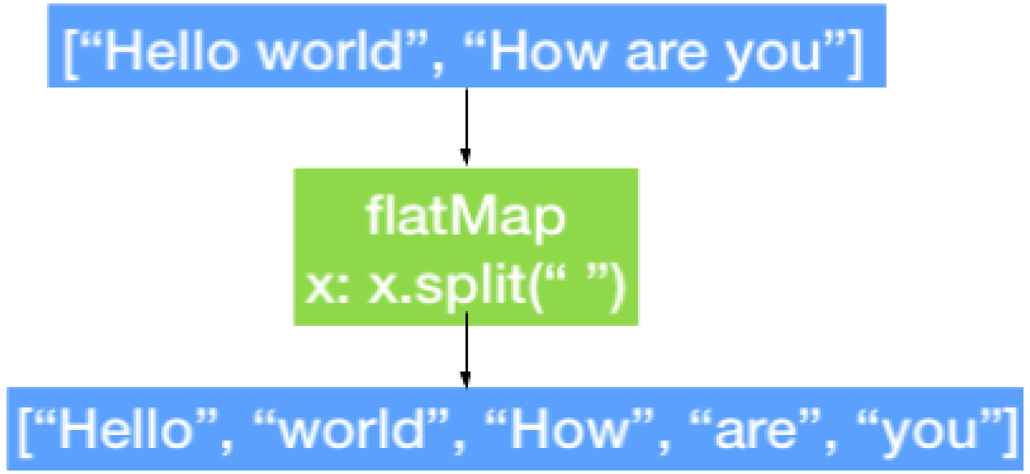
- Filter transformation returns a new RDD with only the elements that pass the condition



```
RDD = sc.parallelize([1,2,3,4])
RDD_filter = RDD.filter(lambda x: x > 2)
```

flatMap() Transformation

- flatMap() transformation returns multiple values for each element in the original RDD



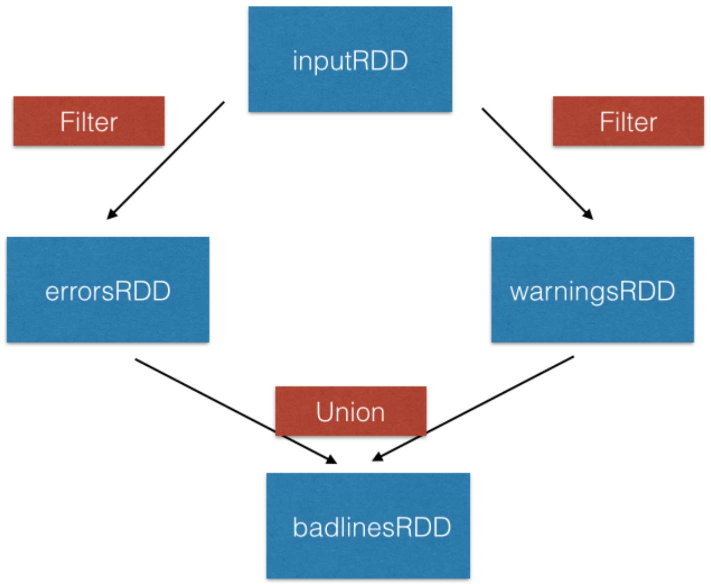
[“Hello world”, “How are you”]

flatMap
x: x.split(“ ”)

[“Hello”, “world”, “How”, “are”, “you”]

```
RDD = sc.parallelize(["hello world", "how are you"])
RDD_flatmap = RDD.flatMap(lambda x: x.split(" "))
```

union() Transformation



```
inputRDD = sc.textFile("logs.txt")
errorRDD = inputRDD.filter(lambda x: "error" in x.split())
warningsRDD = inputRDD.filter(lambda x: "warnings" in x.split())
combinedRDD = errorRDD.union(warningsRDD)
```

RDD Actions

- Operation return a value after running a computation on the RDD
- Basic RDD Actions
  - collect()
  - take(N)
  - first()
  - count()

collect() and take() Actions

- collect() return all the elements of the dataset as an array
- take(N) returns an array with the first N elements of the dataset

```
RDD_map.collect()
```

```
[1, 4, 9, 16]
```

```
RDD_map.take(2)
```

```
[1, 4]
```

first() and count() Actions

- first() prints the first element of the RDD

```
RDD_map.first()
```

```
[1]
```

- count() return the number of elements in the RDD

```
RDD_flatmap.count()
```

```
5
```

Let's practice RDD operations

BIG DATA FUNDAMENTALS VIA PYSPARK

Working with Pair RDDs in PySpark

BIG DATA FUNDAMENTALS VIA PYSPARK

Upendra Devisetty
Science Analyst, CyVerse

Introduction to pair RDDs in PySpark

- Real life datasets are usually key/value pairs
- Each row is a key and maps to one or more values
- Pair RDD is a special data structure to work with this kind of datasets
- Pair RDD: Key is the identifier and value is data

Creating pair RDDs

- Two common ways to create pair RDDs
  - From a list of key-value tuple
  - From a regular RDD
- Get the data into key/value form for paired RDD

```
my_tuple = [('Sam', 23), ('Mary', 34), ('Peter', 25)]
pairRDD_tuple = sc.parallelize(my_tuple)
```

```
my_list = ['Sam 23', 'Mary 34', 'Peter 25']
regularRDD = sc.parallelize(my_list)
pairRDD_RDD = regularRDD.map(lambda s: (s.split(' ')[0], s.
```

Transformations on pair RDDs

- All regular transformations work on pair RDD
- Have to pass functions that operate on key value pairs rather than on individual elements
- Examples of paired RDD Transformations
  - reduceByKey(func): Combine values with the same key
  - groupByKey(): Group values with the same key
  - sortByKey(): Return an RDD sorted by the key
  - join(): Join two pair RDDs based on their key

reduceByKey() transformation

- `reduceByKey()` transformation combines values with the same key
- It runs parallel operations for each key in the dataset
- It is a transformation and not action

```
regularRDD = sc.parallelize([("Messi", 23), ("Ronaldo", 34)
pairRDD_reducebykey = regularRDD.reduceByKey(lambda x,y : x
pairRDD_reducebykey.collect()
```

```
[('Neymar', 22), ('Ronaldo', 34), ('Messi', 47)]
```

sortByKey() transformation

- `sortByKey()` operation orders pair RDD by key
- It returns an RDD sorted by key in ascending or descending order

```
pairRDD_reducebykey_rev = pairRDD_reducebykey.map(lambda x:
pairRDD_reducebykey_rev.sortByKey(ascending=False).collect(
```

```
[(47, 'Messi'), (34, 'Ronaldo'), (22, 'Neymar')]
```

groupByKey() transformation

- `groupbykey()` groups all the values with the same key in the pair RDD

```
airports = [("US", "JFK"),("UK", "LHR"),("FR", "CDG"),("US"
regularRDD = sc.parallelize(airports)
pairRDD_group = regularRDD.groupByKey().collect()
for cont, air in pairRDD_group:
  print(cont, list(air))
```

```
FR ['CDG']
US ['JFK', 'SFO']
UK ['LHR']
```

join() transformation

- `join()` transformation joins the two pair RDDs based on their key

```
RDD1 = sc.parallelize([("Messi", 34),("Ronaldo", 32),("Neym
RDD2 = sc.parallelize([("Ronaldo", 80),("Neymar", 120),("Me
```

```
RDD1.join(RDD2).collect()

[('Neymar', (24, 120)), ('Ronaldo', (32, 80)), ('Messi', (3
```

Let's practice

BIG DATA FUNDAMENTALS VIA PYSPARK

More actions

BIG DATA FUNDAMENTALS VIA PYSPARK

Upendra Devisetty

Science Analyst, CyVerse

reduce() action

- reduce(func) action is used for aggregating the elements of a regular RDD
- The function should be commutative (changing the order of the operands does not change the result) and associative
- An example of `reduce()` action in PySpark

```
x = [1,3,4,6]
RDD = sc.parallelize(x)
RDD.reduce(lambda x, y : x + y)
```

```
14
```

saveAsTextFile() action

- `saveAsTextFile()` action saves RDD into a text file inside a directory with each partition as a separate file

```
RDD.saveAsTextFile("tempFile")
```

  - `coalesce()` method can be used to save RDD as a single text file

```
RDD.coalesce(1).saveAsTextFile("tempFile")
```

Action Operations on pair RDDs

- RDD actions available for PySpark pair RDDs
- Pair RDD actions leverage the key-value data
- Few examples of pair RDD actions include
  - `countByKey()`
  - `collectAsMap()`

countByKey() action

- `countByKey()` only available for type (K, V)
- `countByKey()` action counts the number of elements for each key
- Example of `countByKey()` on a simple list

```
rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])
for kee, val in rdd.countByKey().items():
  print(kee, val)
```

```
('a', 2)
('b', 1)
```

collectAsMap() action

- `collectAsMap()` return the key-value pairs in the RDD as a dictionary
- Example of `collectAsMap()` on a simple tuple

```
sc.parallelize([(1, 2), (3, 4)]).collectAsMap()
```

```
{1: 2, 3: 4}
```

Let's practice

BIG DATA FUNDAMENTALS VIA PYSPARK