

Streaming de Dados em Tempo Real: Aula 3

Prof. Felipe Timbó



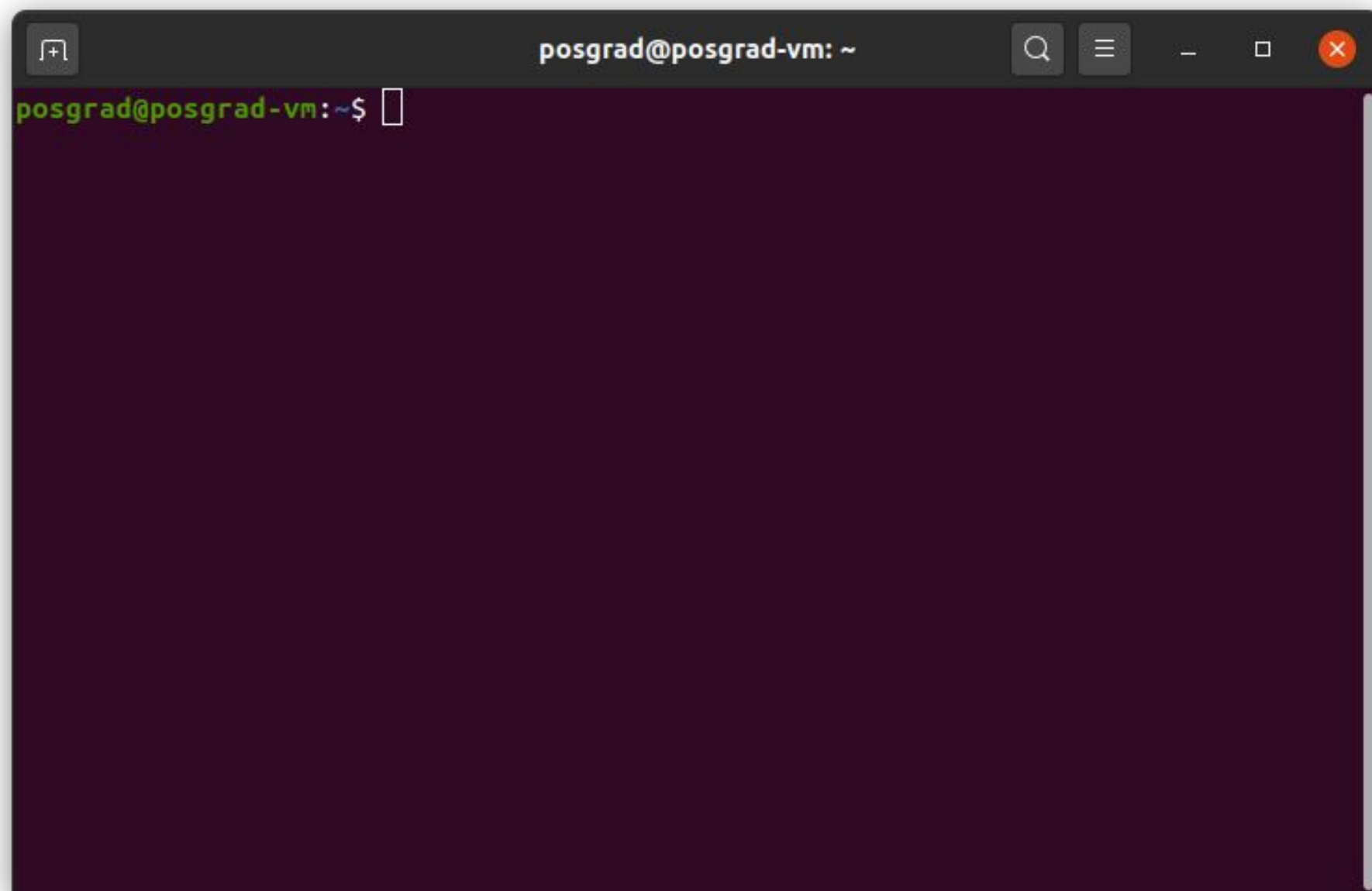
Ementa (dia 3)

- Introdução ao Apache Spark
 - Ações e Transformações
 - RDD's e Dataframes
- Manipulação de dados com Apache Spark e Python

Instalando o Spark

Iniciando a VM

1. Iniciar a VM com Ubuntu configurada nas aulas passadas
2. Logar na VM e abrir um Terminal



Instalando o Spark no Ubuntu

3. Baixar Spark + Hadoop com o comando abaixo:

➤ `wget`

```
https://dlcdn.apache.org/spark/spark-3.2.3/spark-3.2.3-bin-hadoop2.7.tgz
```

Instalando o Spark no Ubuntu

4. Extrair os arquivos:

➤ `tar -xvzf spark-*`

5. Mover os arquivos para o diretório *opt/spark*:

➤ `sudo mv spark-3.2.3-bin-hadoop2.7 /opt/spark`

Configurando o ambiente Spark

6. Configurar as variáveis de ambiente com os comandos:

- `echo "export SPARK_HOME=/opt/spark" >> ~/.profile`
- `echo "export PATH=$PATH:/opt/spark/bin:/opt/spark/sbin" >> ~/.profile`
- `echo "export PYSPARK_PYTHON=/usr/bin/python3" >> ~/.profile`

7. Garantir que as novas variáveis podem ser acessadas:

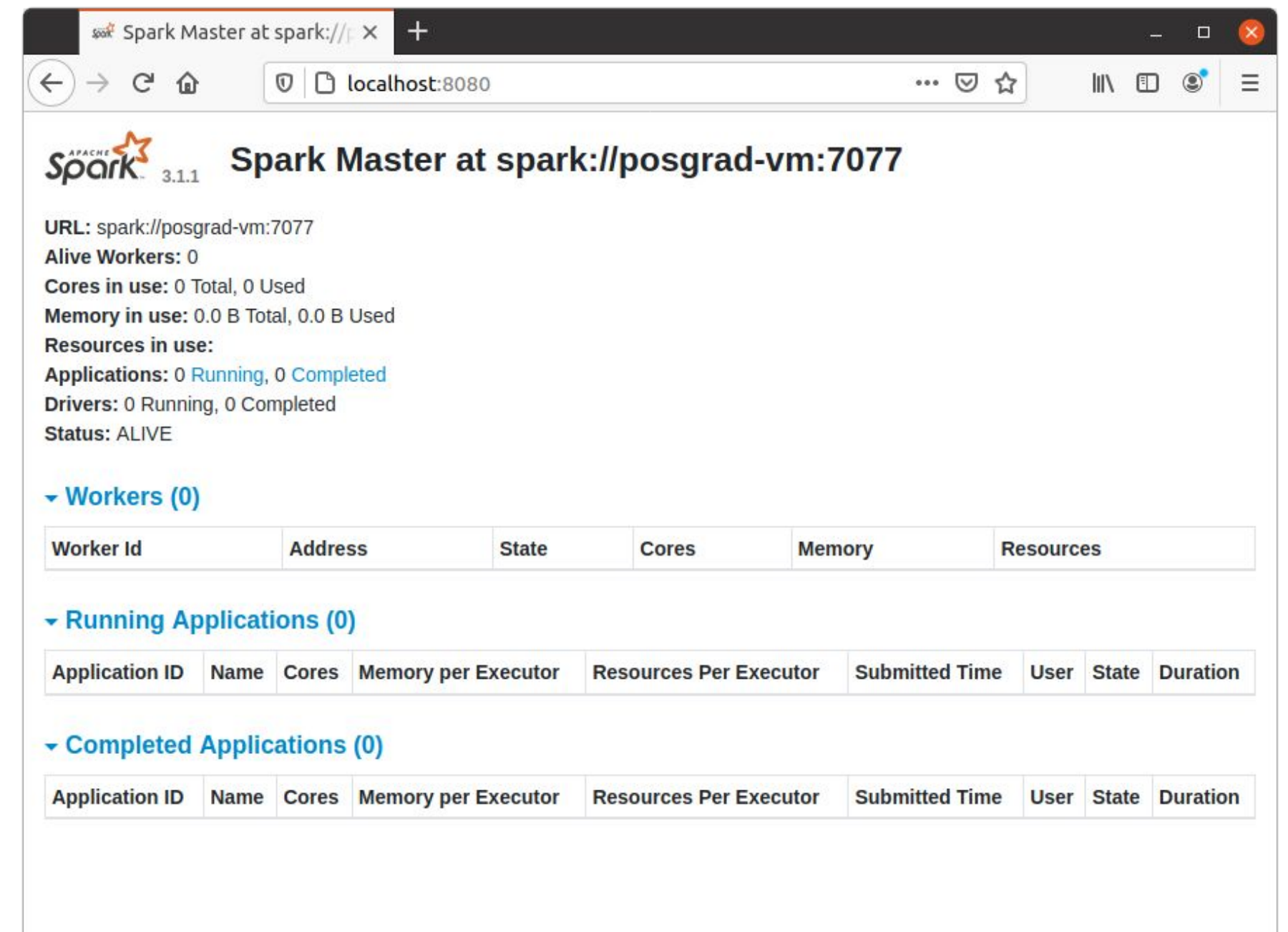
- `source ~/.profile`

Iniciando o Spark

8. Iniciar o spark:

➤ `start-master.sh`

9. Acessar localhost:8080 no navegador e visualizar o Spark rodando:



The screenshot shows the Spark Master web interface in a browser. The address bar shows 'localhost:8080'. The page title is 'Spark Master at spark://posgrad-vm:7077'. The interface displays the following information:

- URL: spark://posgrad-vm:7077
- Alive Workers: 0
- Cores in use: 0 Total, 0 Used
- Memory in use: 0.0 B Total, 0.0 B Used
- Resources in use:
- Applications: 0 Running, 0 Completed
- Drivers: 0 Running, 0 Completed
- Status: ALIVE

Below this information, there are three expandable sections:

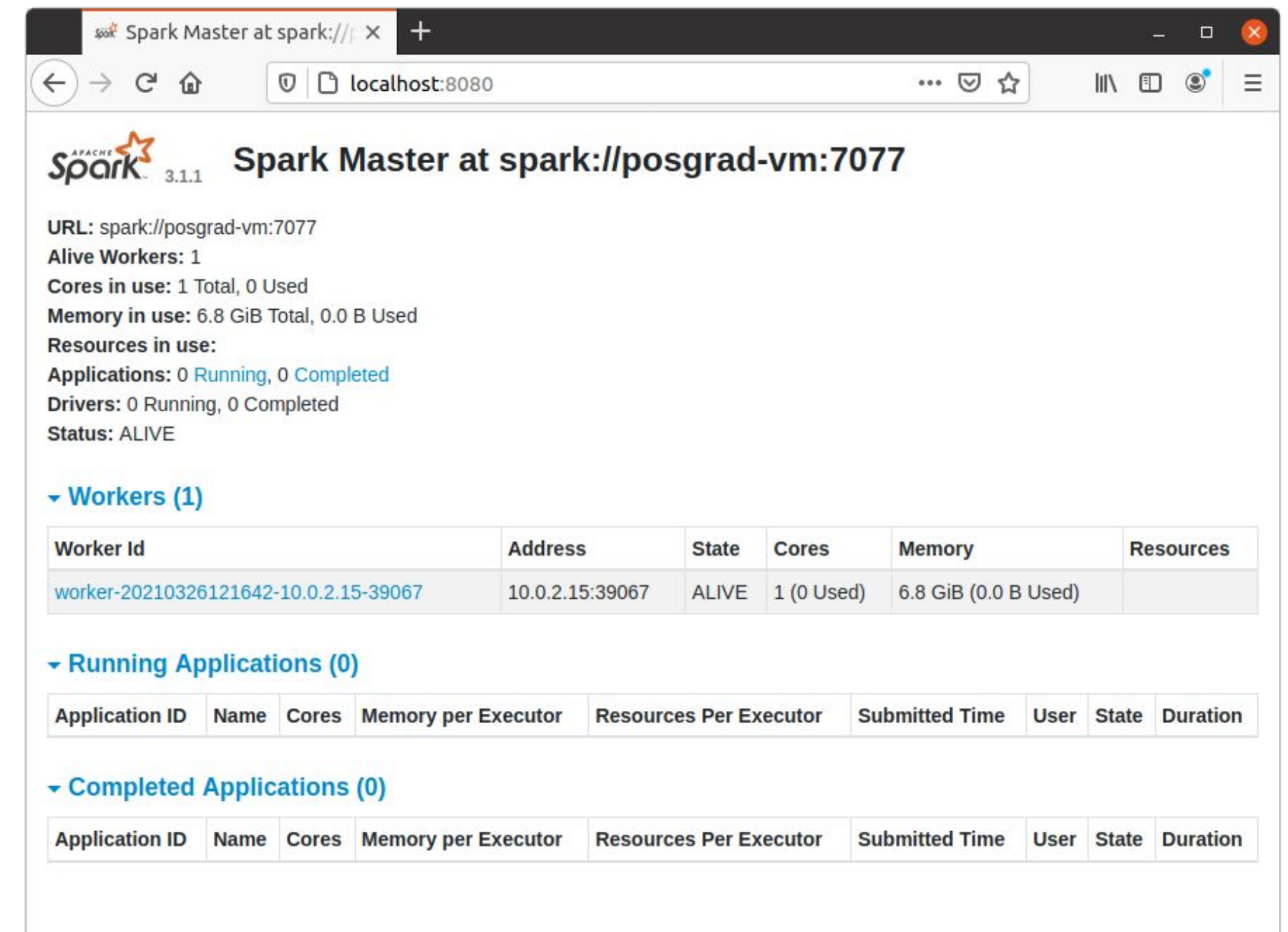
- Workers (0)**: A table with columns: Worker Id, Address, State, Cores, Memory, Resources.
- Running Applications (0)**: A table with columns: Application ID, Name, Cores, Memory per Executor, Resources Per Executor, Submitted Time, User, State, Duration.
- Completed Applications (0)**: A table with columns: Application ID, Name, Cores, Memory per Executor, Resources Per Executor, Submitted Time, User, State, Duration.

Iniciando o Spark

10. Iniciar o nó trabalhador:

➤ `start-worker.sh spark://posgrad-VirtualBox:7077`

11. Acessar localhost:8080 no navegador e visualizar o Worker ativo:



The screenshot shows the Spark Master web interface in a browser window. The title bar indicates the URL is `localhost:8080`. The page header shows the Apache Spark logo and version 3.1.1, along with the title "Spark Master at spark://posgrad-vm:7077".

The main content area displays the following information:

- URL: `spark://posgrad-vm:7077`
- Alive Workers: 1
- Cores in use: 1 Total, 0 Used
- Memory in use: 6.8 GiB Total, 0.0 B Used
- Resources in use:
- Applications: 0 Running, 0 Completed
- Drivers: 0 Running, 0 Completed
- Status: ALIVE

Below this information, there are three expandable sections:

- Workers (1)**: A table showing the status of the worker nodes.
- Running Applications (0)**: A table showing the status of running applications.
- Completed Applications (0)**: A table showing the status of completed applications.

Worker Id	Address	State	Cores	Memory	Resources
worker-20210326121642-10.0.2.15-39067	10.0.2.15:39067	ALIVE	1 (0 Used)	6.8 GiB (0.0 B Used)	

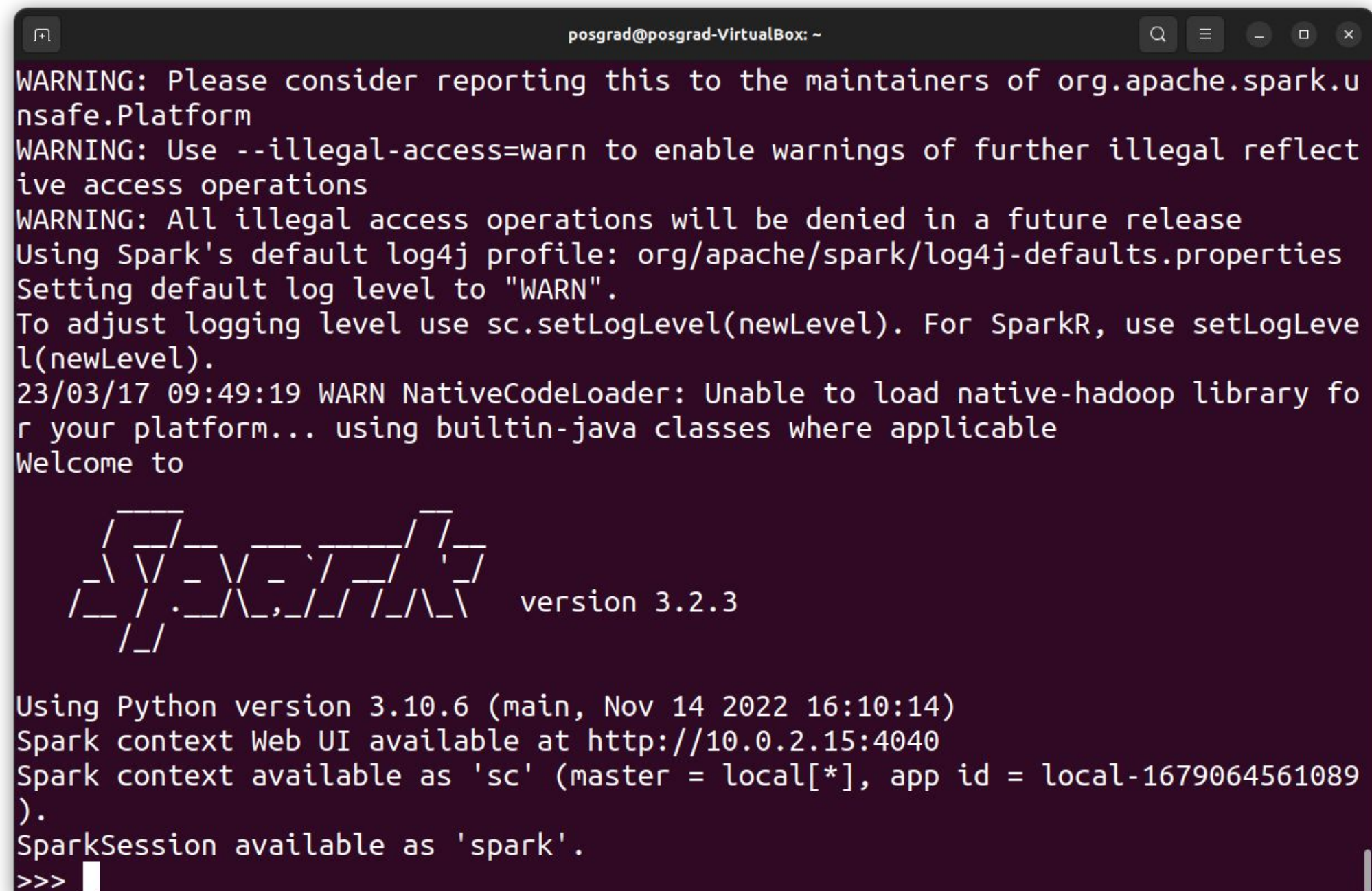
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Rodando o Spark Shell

12. Verificar a instalação do Spark Shell com seguinte comando:

➤ `pyspark`

A terminal window titled 'posgrad@posgrad-VirtualBox: ~' showing the output of the 'pyspark' command. The output includes several warning messages about illegal reflective access operations, the Spark version (3.2.3), and the Python version (3.10.6). It also displays the Spark context Web UI URL and the SparkSession object.

```
posgrad@posgrad-VirtualBox: ~  
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform  
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations  
WARNING: All illegal access operations will be denied in a future release  
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties  
Setting default log level to "WARN".  
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).  
23/03/17 09:49:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
Welcome to  
  
  ____      _ _      _ _  
 / ___|    / _ \    / _ \    _ _  
 \___ \  __/ ___ \  / ___ \  _ _  
  ___) | | |___) | | |___) | | |  
 /____|_| \____|_| \____|_| \____|_|  
                                version 3.2.3  
  
Using Python version 3.10.6 (main, Nov 14 2022 16:10:14)  
Spark context Web UI available at http://10.0.2.15:4040  
Spark context available as 'sc' (master = local[*], app id = local-1679064561089).  
SparkSession available as 'spark'.  
>>>
```

Caso apareça a imagem ao lado, você instalou o Spark com sucesso :)

Spark

Spark

- *“A fast and general engine for large-scale data processing”*
- Manipulações, transformações e análises complexas
 - Aprendizagem de máquina
 - Mineração de dados
 - Análise de grafos
 - Streaming
- Linguagens: **Python**, Java, e Scala

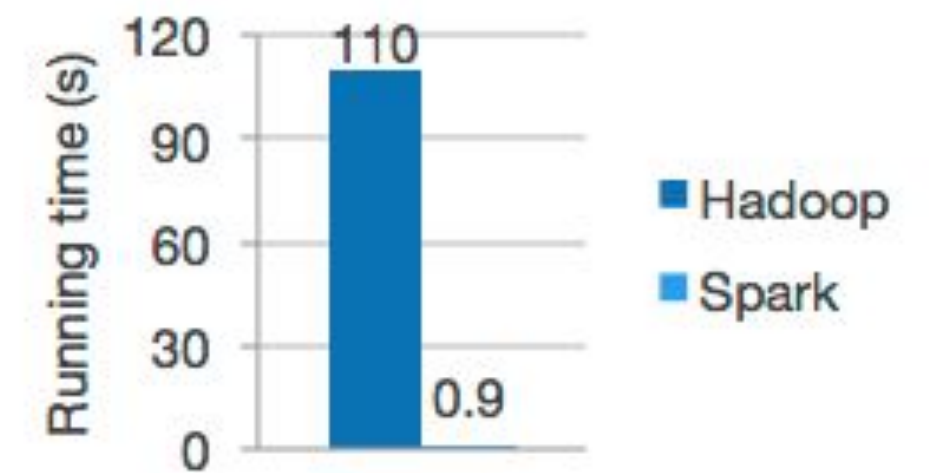
Spark - Quem usa?

- Amazon
- Ebay: análise de log e agregação
- NASA JPL: Deep Space Network
- Groupon
- TripAdvisor
- Etc. <https://spark.apache.org/powered-by.html>



Spark - Desempenho

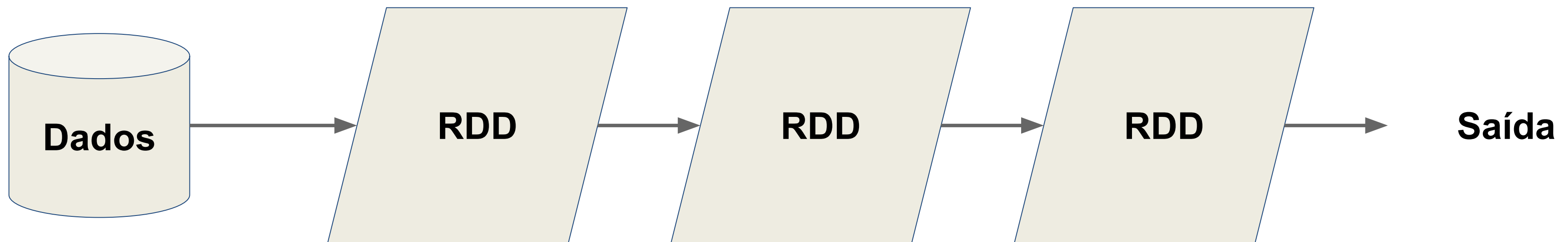
- 100x mais rápido que o Hadoop MapReduce em memória
- 10x mais rápido que o Hadoop MapReduce em disco
- 2x a 5x menos código



Logistic regression in Hadoop and Spark

RDD - Resilient Distributed Dataset

- Abstração fornecida pelo Spark para a manipulação de dados.
- Representação de um dado distribuído pelos nós do cluster que pode ser operado em paralelo.
- Transformações e Ações



RDD - Resilient Distributed Dataset

A solid red rectangular box with a thin black border.

Particionado

Dividido em nós
de um cluster

A solid purple rectangular box with a thin black border.

Imutável

RDDs, uma vez
criados, não podem
ser alterados

A solid blue rectangular box with a thin black border.

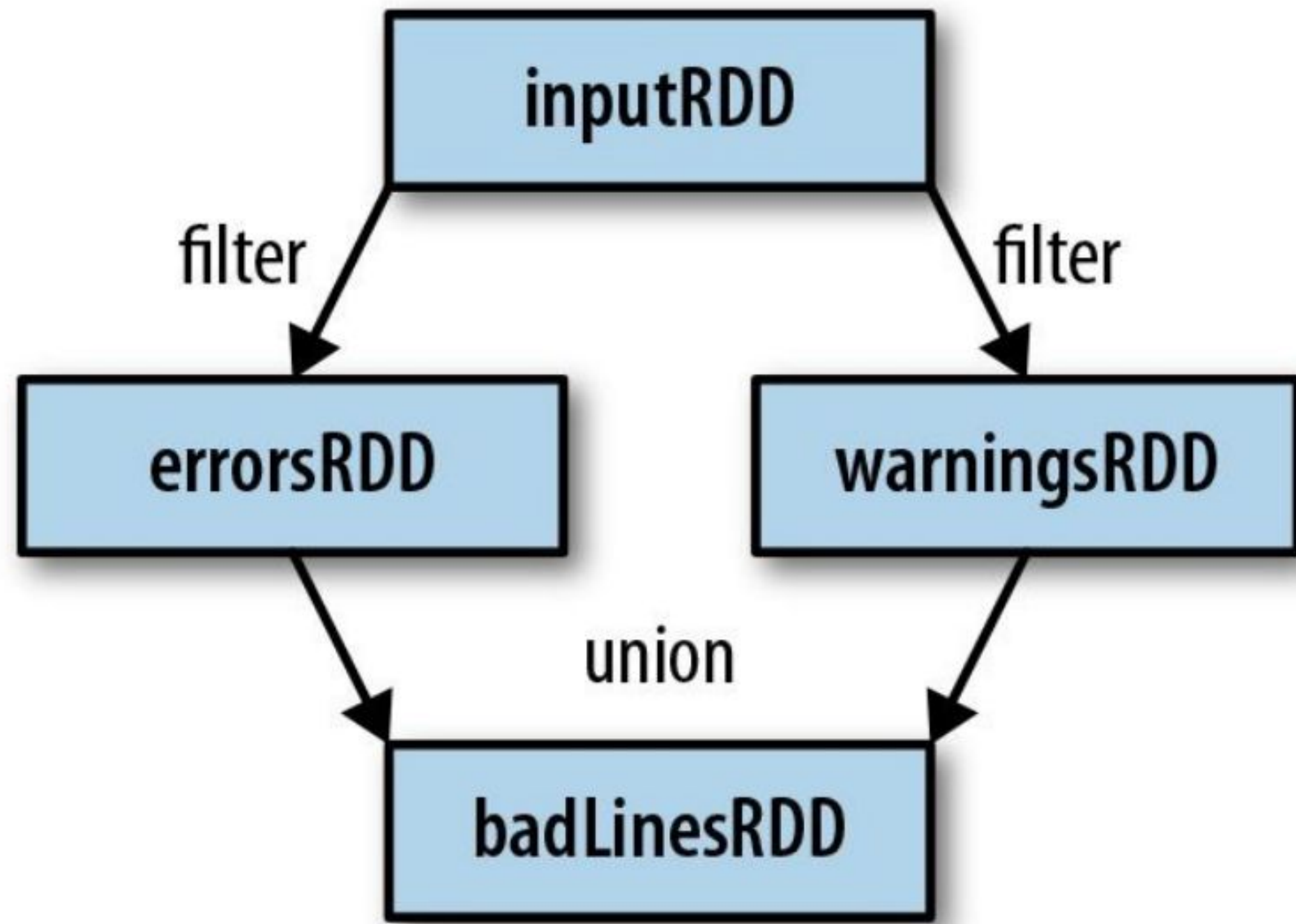
Resiliente

Podem ser
reconstruídos mesmo
se um nó caia

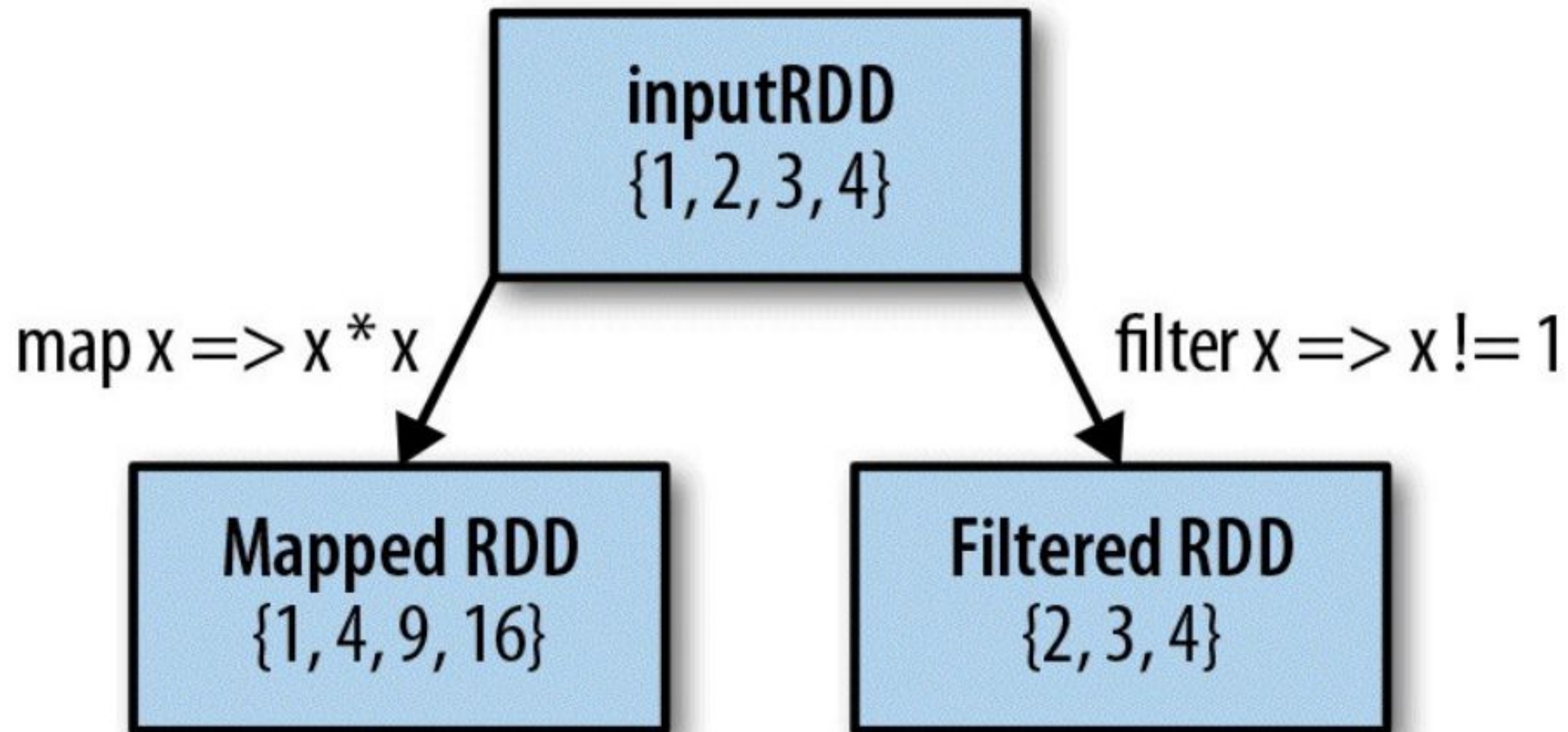
Transformações

- map
- flatmap
- filter
- distinct
- sample
- union, intersection, subtract, cartesian
- etc.

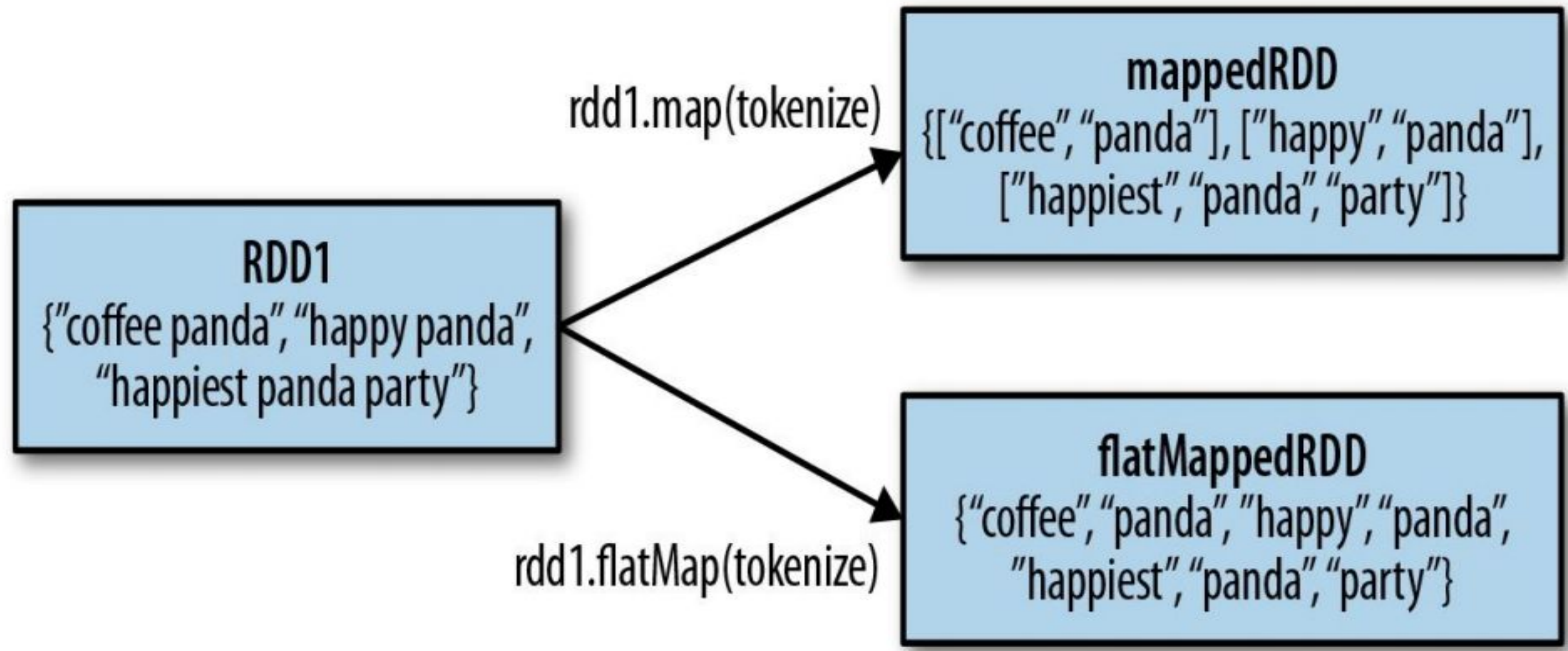
Transformações



Transformações



Transformações



Transformações

RDD1
{coffee, coffee, panda,
monkey, tea}

RDD2
{coffee, money, kitty}

RDD1.distinct()
{coffee, panda,
monkey, tea}

RDD1.union(RDD2)
{coffee, coffee, coffee,
panda, monkey,
monkey, tea, kitty}

RDD1.intersection(RDD2)
{coffee, monkey}

RDD1.subtract(RDD2)
{panda, tea}

Transformações

Transformação	Descrição	Exemplo	Resultado
map (<i>func</i>)	retorna um RDD depois de processar cada elemento com a função <i>func</i>	sc.parallelize([1, 2, 3, 4,5]).map(lambda x: x + 10).collect()	{ 11, 12, 13, 14, 15 }
filter (<i>func</i>)	Retorna um RDD depois de selecionar os enementos onde <i>func</i> é verdadeira	sc.parallelize([1, 2, 3, 4, 5]).filter(lambda x: x == 3).collect()	3
flatMap (<i>func</i>)	Similar ao map, mas cada item pode ser mapeado para 0 ou mais items (<i>func</i> deve retornar uma sequência, ao invés de um item).	sc.parallelize(["this is line 1", "this is the second line"]).flatMap(lambda line: line.split(" ")).collect()	['this', 'is', 'line', '1', 'this', 'is', 'the', 'second', 'line']

Transformações

Transformação	Descrição	Exemplo	Resultado
union (<i>otherDataset</i>)	Retorna uma coleção que contém a união de todos os elementos da fonte e do argumento.	sc.parallelize([1, 2, 3, 4, 5]).union(sc.parallelize([6, 7, 8, 9, 10])).collect()	{1,2,3,4,5,6,7,8,9,10}
intersection (<i>otherDataset</i>)	Retorna um RDD que contém uma interseção dos elementos da fonte e do argumento.	sc.parallelize([1, 2, 3, 4, 5]).intersection(sc.parallelize([4, 5, 6, 7, 8])).collect()	{5,4}
distinct ([<i>numTasks</i>]))	Retorna uma coleção que contém os elementos distintos da fonte.	sc.parallelize([(1,2), (1,2), (1,2), (3,4), (5,6), (7,8), (7,8)]).distinct().collect()	{(1,2), (5,6), (7,8), (3,4)}
reduceByKey (<i>func</i> , [<i>numTasks</i>])	Quando invocado sobre uma coleção de pares (k, v), retorna uma coleção de pares (k, v) onde os valores de cada chave são agregados usando pela função <i>func</i> , que deve ser do tipo (v, v).	sc.parallelize([(1,2), (1,2), (1,2), (3,4), (5,6), (7,8), (7,8)]).reduceByKey(lambda x, y: x + y).collect()	(1,6) (3,4) (7,16) (5,6)

Transformações

Transformação	Descrição	Exemplo	Resultado
sortByKey (<i>[ascending]</i> , <i>[numTasks]</i>)	Quando invocado sobre uma coleção de pares (k, v), onde k implementa Ordered, retorna uma coleção de pares (k, v) ordenados pela chaves.	sc.parallelize([(1,2), (1,2), (1,2), (3,4), (5,6), (7,8), (7,8)]).sortByKey().collect()	(1,2) (1,2) (1,2) (3,4) (5,6) (7,8) (7,8)
join (<i>otherDataset</i> , <i>[numTasks]</i>)	Quando invocado sobre uma coleção de pares do tipo (k, v) e (k, w), retorna uma coleção de pares (k, (v, w)).	sc.parallelize([(1,"um"), (2,"dois"), (3,"tres")]).join(sc.parallelize([(1,"one"), (2, "two"), (3, "three")])).collect()	(1,(um,one)) (3,(tres,three)) (2,(dois,two))

Ações

- collect
- count
- countByValue
- take
- top
- reduce
- etc.

Obs. Lazy evaluation: nada acontece até uma dessas funções serem chamadas!

Ações

Ações	Descrição	Exemplo	Resultado
reduce (<i>func</i>)	Agrega os elementos de uma coleção usando a função <i>func</i> (que precisa de dois argumentos, retornando um). A função deve ser acumulativa ou associativa, para ser computada corretamente em paralelo.	<code>sc.parallelize([1,2,3,4,5]).reduce(lambda x, y: x + y)</code>	15
collect ()	Retorna todos os elementos de uma coleção como um vetor dentro do programa <i>driver</i> .	<code>sc.parallelize([1,2,3,4,5]).collect()</code>	<code>Array(1, 2, 3, 4, 5)</code>
count ()	Retorna o número de elementos dentro da coleção de dados.	<code>sc.parallelize([1,2,3,4,5]).count()</code>	5
first ()	Retorna o primeiro elemento da coleção de dados.	<code>sc.parallelize([1,2,3,4,5]).first()</code>	1
take (<i>n</i>)	Return an array with the first <i>n</i> elements of the dataset.	<code>sc.parallelize([1,2,3,4,5]).take(2)</code>	<code>Array(1, 2)</code>

Mais Ações e Transformações em...

https://spark.apache.org/docs/latest/rdd-programming-guide.html

APACHE

Spark

3.1.1

Overview

Programming Guides

API Docs

Deploying

More

Q

Search the docs

a matching hashCode() method. For full details, see the contract outlined in the [Object.hashCode\(\)](#) documentation.

Transformations

The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc ([Scala](#), [Java](#), [Python](#), [R](#)) and pair RDD functions doc ([Scala](#), [Java](#)) for details.

Transformation	Meaning
<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .
<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).
<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.
<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.
<code>sample(withReplacement, fraction, seed)</code>	Sample a fraction <i>fraction</i> of the data, with or without replacement, using a given random number generator seed.
<code>union(otherDataset)</code>	Return a new dataset that contains the union of the elements in the source dataset and the argument.
<code>intersection(otherDataset)</code>	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
<code>distinct([numPartitions])</code>	Return a new dataset that contains the distinct elements of the source dataset.
<code>groupByKey([numPartitions])</code>	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

APACHE

Spark

3.1.1

Overview

Programming Guides

API Docs

Deploying

More

Q

Search the docs

Actions

The following table lists some of the common actions supported by Spark. Refer to the RDD API doc ([Scala](#), [Java](#), [Python](#), [R](#)) and pair RDD functions doc ([Scala](#), [Java](#)) for details.

Action	Meaning
<code>reduce(func)</code>	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
<code>collect()</code>	Return 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.
<code>count()</code>	Return the number of elements in the dataset.
<code>first()</code>	Return the first element of the dataset (similar to take(1)).
<code>take(n)</code>	Return an array with the first <i>n</i> elements of the dataset.
<code>takeSample(withReplacement, num, [seed])</code>	Return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
<code>takeOrdered(n, [ordering])</code>	Return the first <i>n</i> elements of the RDD using either their natural order or a custom comparator.
<code>saveAsTextFile(path)</code>	Write 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 will call toString on each element to convert it to a line of text in the file.
<code>saveAsSequenceFile(path)</code> (Java and Scala)	Write 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).
<code>saveAsObjectFile(path)</code> (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using <code>SparkContext.objectFile()</code> .

Criando dados para “brincar”

Criar um arquivo de entrada:

- `cd ~`
- `gedit alunos.csv`

Iniciar o spark shell

- `pyspark`

A notepad icon with a light beige background and a folded corner. It contains the following text:

```
Nome, Idade  
John, 23  
Jim, 22  
Emily, 41  
Nina, 50  
Susan, 31
```

Brincando com os dados

- `alunos = sc.textFile("file:///home/posgrad/alunos.csv")`
- `alunos.take(2)`
- `alunosComJ = alunos.filter(lambda row: 'J' in row)`
- `alunosComJ.take(2)`

- `nums = sc.parallelize([1, 2, 3, 4])`
- `numsMaisUm = nums.map(lambda x: x+1)`
- `nums2 = sc.parallelize([4, 5, 6])`
- `nums.union(nums2).take(10)`
- `nums.intersection(nums2).take(10)`
- `exit()`

Spark

Rodando script Python

WordCount em Spark

1. Download dos dados e do script no Terminal

- `cd ~`
- `wget https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/shakespeare.txt`
- `wget https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/wordcount.py`

2. Abrir o script no VSCode

- `code wordcount.py`

3. Rodar no terminal o script utilizando Spark:

- `/opt/spark/bin/spark-submit wordcount.py`

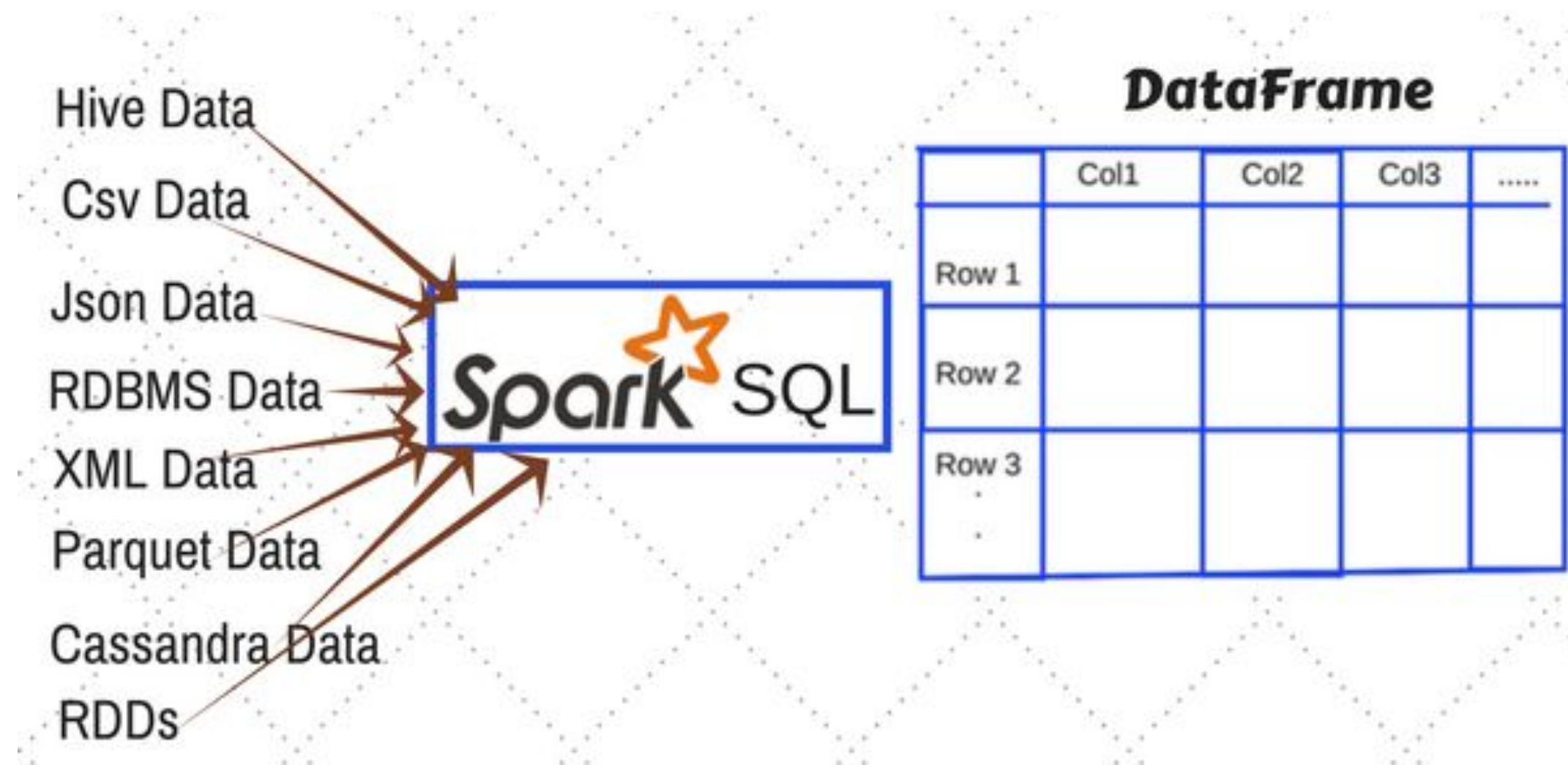
Exercício

Ordenar pelas palavras que mais aparecem no livro de Shakespeare.

Spark - Dataframes

Spark - Dataframes

- Extensão do RDD
- Pode rodar consultas SQL
- Possui um esquema (maneira melhor de lidar)



Dataframes - Exemplo

Em outro terminal, baixar o conjunto de dados de pessoas

➤ `wget`

```
https://raw.githubusercontent.com/felipetimbo/streaming-data-course/main/people.json
```

Dataframes - Exemplo

Criando um dataframe de pessoas no **terminal Pyspark**

```
➤ df = spark.read.json("file:///home/posgrad/people.json")
```

Brincando com Dataframes

- `df.show()`
- `df.printSchema()`
- `df.select(df.age).show()`
- `df.select(df['name'], df['age'] + 1).show()`
- `df.filter(df['age'] > 21).show()`
- `df.groupBy("age").count().show()`
- `df.orderBy("age", ascending = False).take(2)`
- `newrow = spark.createDataFrame([(16, 'Karlos')], df.columns)`
- `newdf = df.union(newrow)`
- `newdf.createOrReplaceTempView("pessoas")`
- `spark.sql("SELECT * FROM pessoas").show()`

Por fim...

Para finalizar o serviço do Spark, basta executar os comandos:

- `stop-worker.sh`
- `stop-master.sh`

Dúvidas