Spark Basics

Prof. Hyuk-Yoon Kwon

Spark Basics

In this chapter you will learn

- How to start the Spark Shell
- About the SparkContext
- Key Concepts of Resilient Distributed Datasets (RDDs)
  - What are they?
  - How do you create them?
  - What operations can you perform with them?
- How Spark uses the principles of functional programming

What is Apache Spark?

- **Apache Spark is a fast and general engine for large-scale data processing**
- **Written in Scala**
  - Functional programming language that runs in a JVM
- **Spark Shell**
  - Interactive – for learning or data exploration
  - Python or Scala
- **Spark Applications**
  - For large scale data processing
  - Python, Scala, or Java

Spark Shell

- The Spark Shell provides interactive data exploration (REPL)
- Writing Spark applications without the shell will be covered later

Python Shell: **pyspark**

```
$ pyspark
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.3.0
      /_/

Using Python version 2.7.8 (default, Aug 27
2015 05:23:36)
SparkContext available as sc, HiveContext
available as sqlCtx.
>>>
```

Scala Shell: **spark-shell**

```
$ spark-shell
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.3.0
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM)
64-Bit Server VM, Java 1.7.0_67)
Created spark context..
Spark context available as sc.
SQL context available as sqlContext.

scala>
```

REPL: Read/Evaluate/Print Loop

Spark Context

- **Every Spark application requires a Spark Context**
  - The main entry point to the Spark API
- **Spark Shell provides a preconfigured Spark Context called `sc`**

entry point for running Spark API.

Python

```
Using Python version 2.7.8 (default, Aug 27 2015 05:23:36)
SparkContext available as sc, HiveContext available as sqlCtx.

>>> sc.appName
u'PySparkShell'
```

Scala

```
...
Spark context available as sc.
SQL context available as sqlContext.

scala> sc.appName
res0: String = Spark shell
```

Practice – Spark Documentation

1. **Start Firefox in your Virtual Machine and visit the Spark documentation on your local machine, using the provided bookmark or opening the URL**
   - file:/usr/lib/spark/docs/_site/index.html
2. **From the Programming Guides menu, select the Spark Programming Guide. Briefly review the guide. You may wish to bookmark the page for later view or conducing projects**
3. **From the API Docs menu, select either Scaladoc or Python API. Bookmark the API page for use during class.**

RDD (Resilient Distributed Dataset) ⇒ apply functions on RDD to get results.

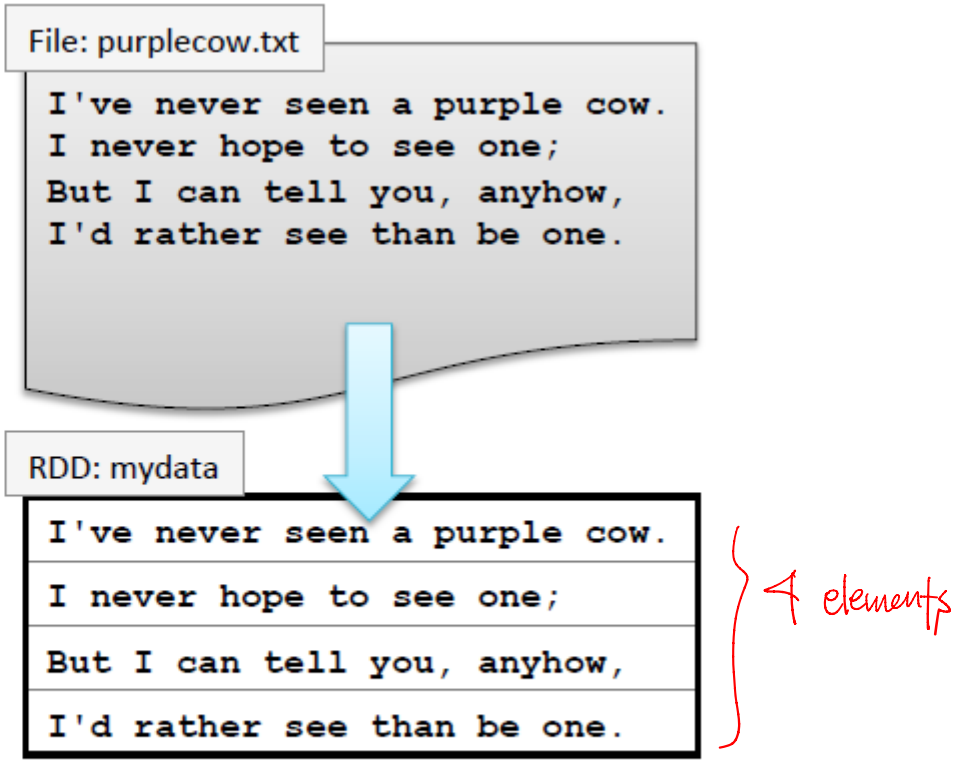
- **RDD (Resilient Distributed Dataset)**
  - Resilient – if data in memory is lost, it can be recreated
  - Distributed – processed across the cluster
  - Dataset – initial data can come from a file or be created programmatically
- **RDDs are the fundamental unit of data in Spark**
- **Most Spark programming consists of performing operations on RDDs**

Creating an RDD

- **Three ways to create an RDD**
  - From a file or set of files
  - From data in memory
  - From another RDD

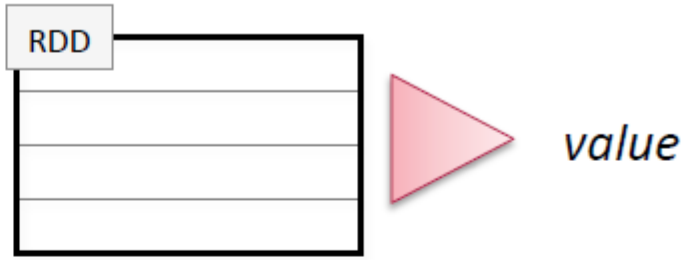
Example: A File-based RDD

```
> val mydata = sc.textFile("purplecow.txt")
…
15/01/29 06:20:37 INFO storage.MemoryStore:
   Block broadcast_0 stored as values to
   memory (estimated size 151.4 KB, free 296.8
   MB)

> mydata.count()

…
15/01/29 06:27:37 INFO spark.SparkContext: Job
   finished: take at <stdin>:1, took
   0.160482078 s
4
```

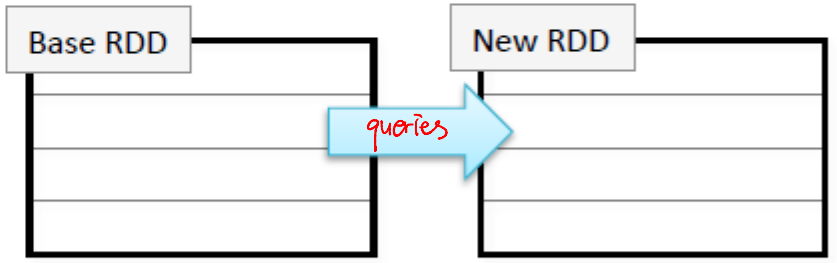


RDD Operations

- **Two types of RDD operations**
  - Actions – return values



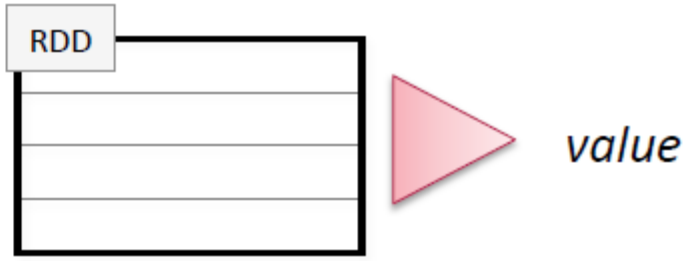
  - Transformations – define a new RDD based on the current one(s)



- **Pop quiz:**
  - Which type of operation is `count()`? actions

RDD Operations: Actions

- **Some common actions**
  - **`count()`** – return the number of elements
  - **`take(`***`n`***`)`** – return an array of the first *n* elements
  - **`collect()`** – return an array of all elements
  - **`saveAsTextFile(`***`file`***`)`** – save to text file(s)

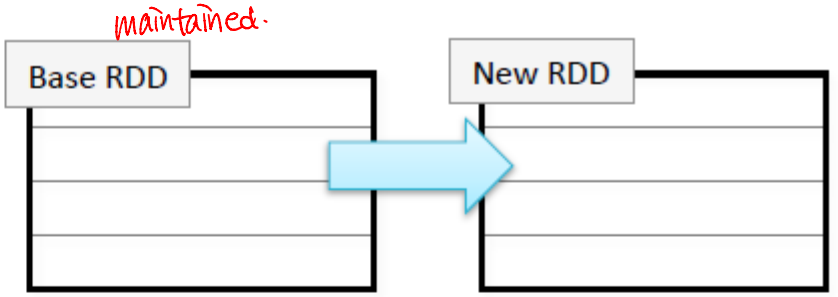



```
> mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for line in mydata.take(2):
    print line
I've never seen a purple cow.
I never hope to see one;
```

```
> val mydata =
  sc.textFile("purplecow.txt")

> mydata.count()
4

> for (line <- mydata.take(2))
    println(line)
I've never seen a purple cow.
I never hope to see one;
```

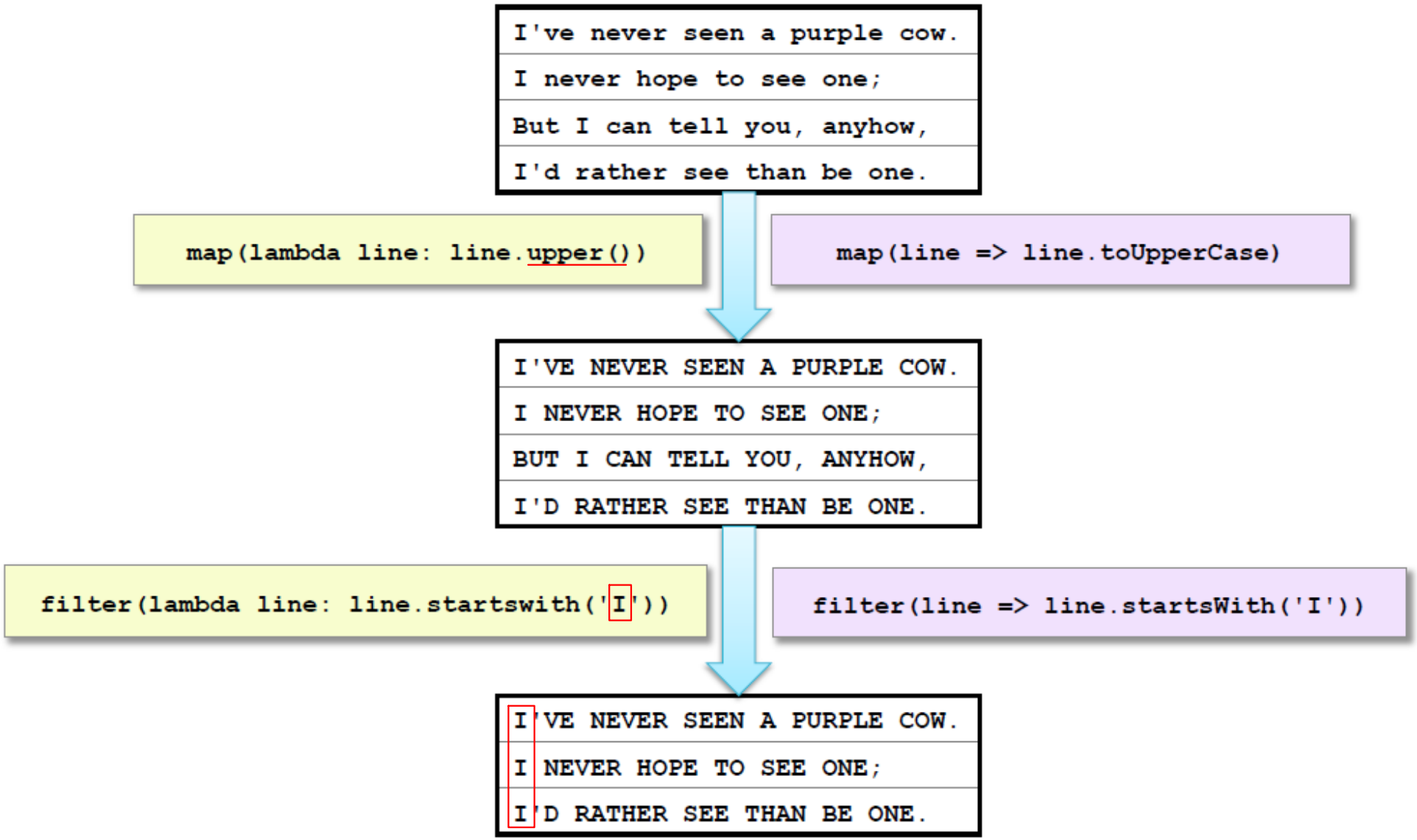
RDD Operations: Transformations

- **Transformations create a new RDD from an existing one**



- **RDDs are immutable**
  - Data in an RDD is never changed
  - Transform in sequence to modify the data as needed
- **Some common transformations**
  - **`map(`***`function`***`)`** – creates a new RDD by performing a function on each record in the base RDD
  - **`filter(`***`function`***`)`** – creates a new RDD by including or excluding each record in the base RDD according to a boolean function

Example: map and filter Transformations



Practice: Explorer RDDs using the Spark Shell

1. Files and data used in this practice

- Exercise directory: $DEV1/exercises/spark-shell
- Data files: $DEV1DATA/frostroad.txt

2. Start the Spark Shell (Python or Scala)

- pyspark (or spark-shell)
- sc

3. Load and view text file (Python or Scala)

- Define an RDD to be created by reading in the test file on the local file system.
- Try counting the number of lines in the dataset
- Try executing the collect operation to display the data in the RDD

```
$ ls $DEV1/exercises/
$ ls $DEV1DATA/
$ cat /home/training/training_materials/data/frostroad.txt
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I--
I took the one less traveled by,
And that has made all the difference.

$ pyspark

> mydata = sc.textFile("file:/home/training/training_materials/data/frostroad.txt")
> mydata.count()
23

> mydata.collect()
[u'Two roads diverged in a yellow wood,',
 u'And sorry I could not travel both',
 u'And be one traveler, long I stood',
 u'And looked down one as far as I could',
 u'To where it bent in the undergrowth;',
 u'',
 u'Then took the other, as just as fair,',
 u'And having perhaps the better claim,',
 u'Because it was grassy and wanted wear;',
 u'Though as for that the passing there',
 u'Had worn them really about the same,',
 u'',
 u'And both that morning equally lay',
 u'In leaves no step had trodden black.',
 u'Oh, I kept the first for another day!',
 u'Yet knowing how way leads on to way,',
 u'I doubted if I should ever come back.',
 u'',
 u'I shall be telling this with a sigh',
 u'Somewhere ages and ages hence:',
 u'Two roads diverged in a wood, and I--',
 u'I took the one less traveled by,',
 u'And that has made all the difference.']
```

Laze Execution (1) for efficiency

- Data in RDDs is not processed until an *action* is performed

File: purplecow.txt

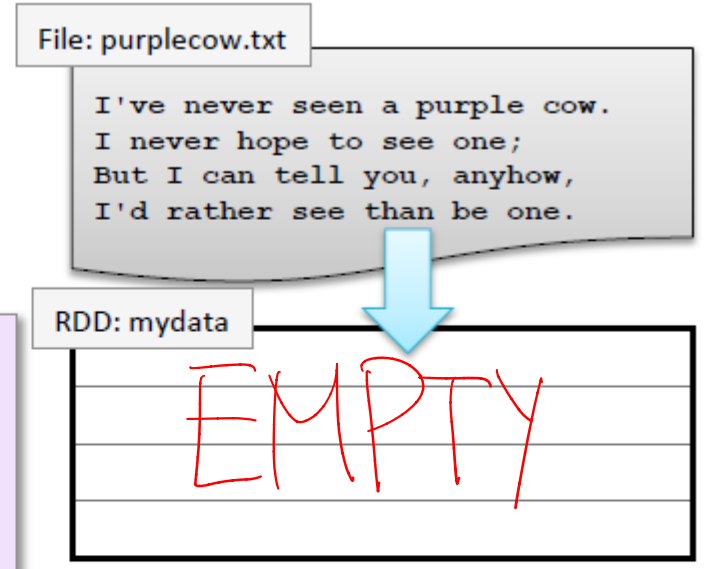
```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

```
>
```

Laze Execution (2)

- Data in RDDs is not processed until an *action* is performed

```
> val mydata = sc.textFile("purplecow.txt")
```



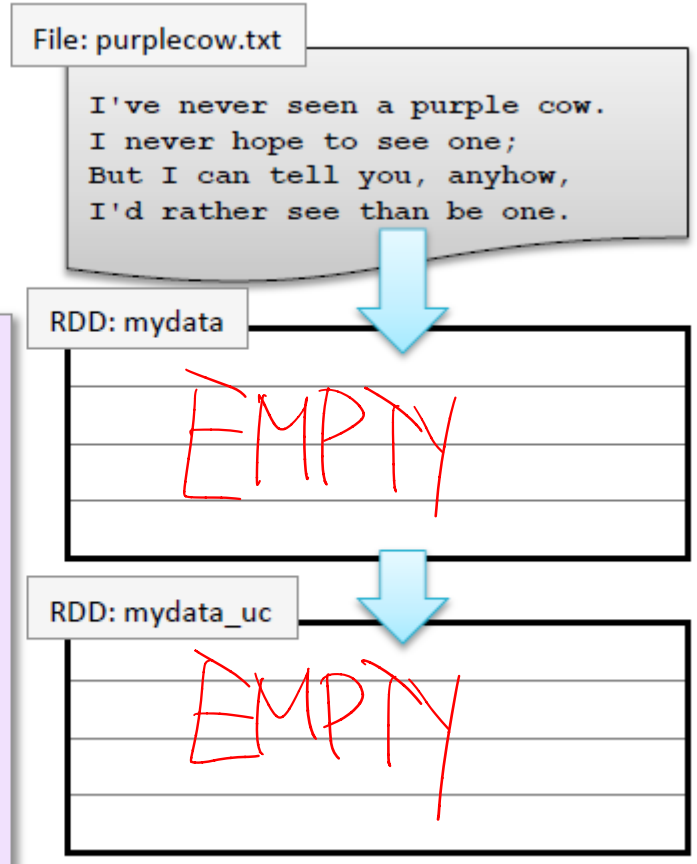
⇒ creating var. linking file.

⇒ reading file (x)

Laze Execution (3)

- Data in RDDs is not processed until an *action* is performed

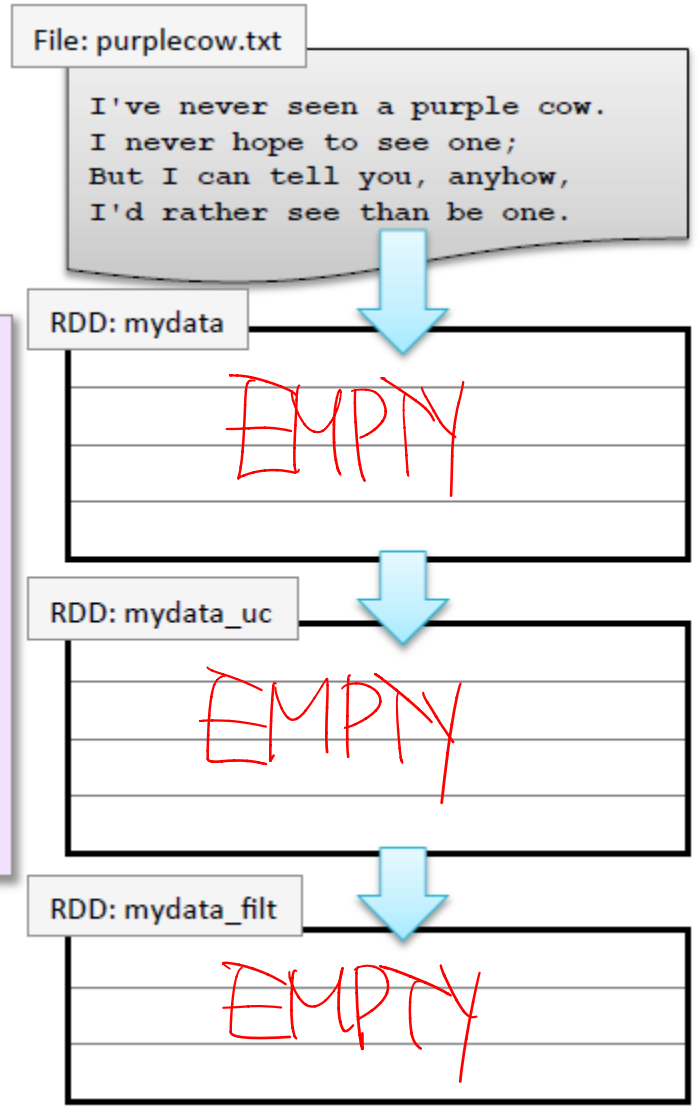
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
```



Laze Execution (4)

- Data in RDDs is not processed until an *action* is performed

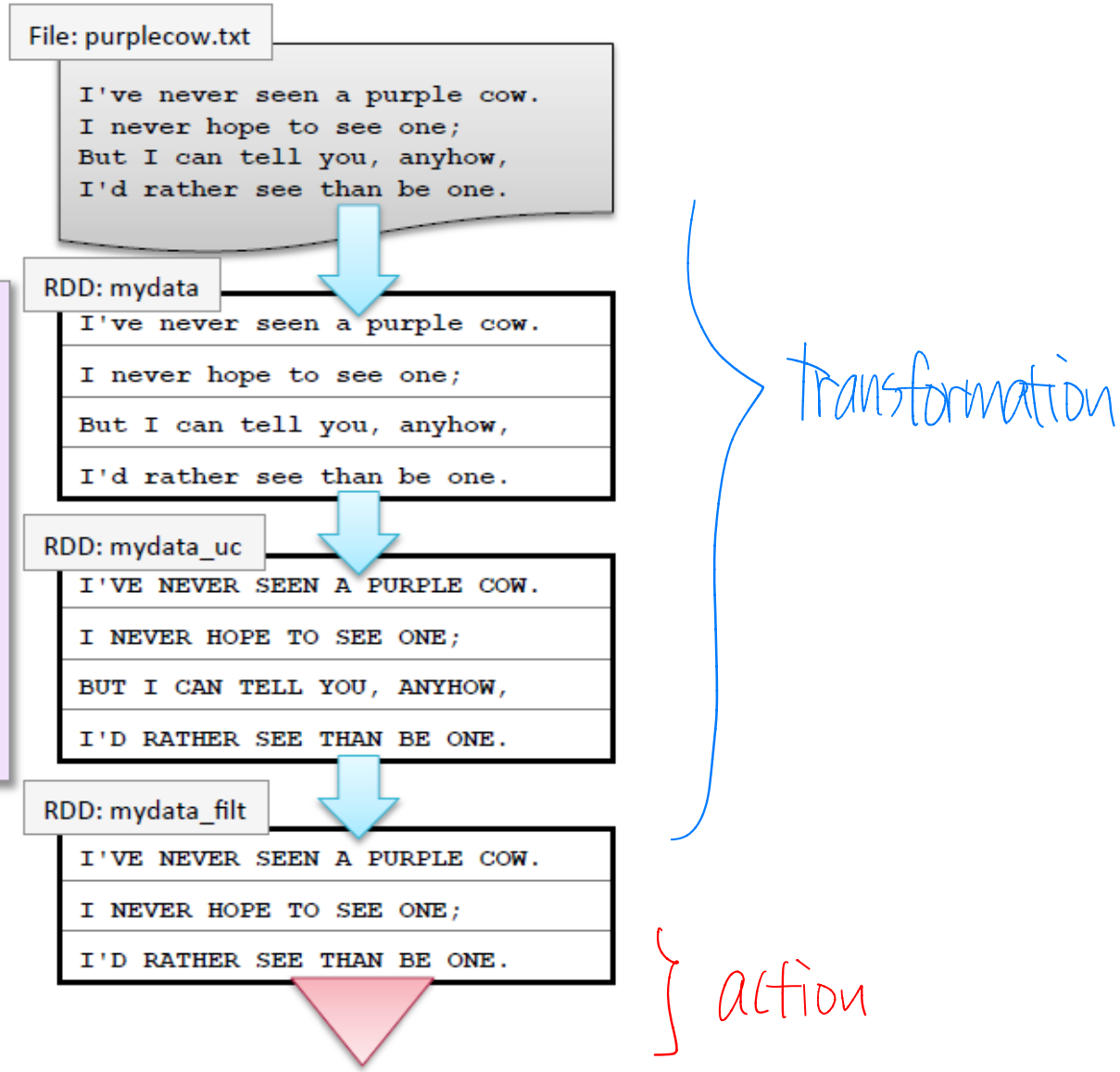
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
```



Laze Execution (5)

- Data in RDDs is not processed until an *action* is performed

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.count()
3
```



Chaining Transformations (Python)

- Same example in Python

```
> mydata = sc.textFile("purplecow.txt")
> mydata_uc = mydata.map(lambda s: s.upper())
> mydata_filt = mydata_uc.filter(lambda s: s.startswith('I'))
> mydata_filt.count()
3
```

is exactly equivalent to

```
> sc.textFile("purplecow.txt").map(lambda line: line.upper()) \
   .filter(lambda line: line.startswith('I')).count()
3
```

chaining

Chaining Transformations (Scala)

- Transformations may be chained together

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line => line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line => line.startsWith("I"))
> mydata_filt.count()
3
```

is exactly equivalent to

```
> sc.textFile("purplecow.txt").map(line => line.toUpperCase()).
   filter(line => line.startsWith("I")).count()
3
```

RDD Lineage and toDebugString (Python)

- `toDebugString` output is not displayed as nicely in Python

```
> mydata_filt.toDebugString()
(1) PythonRDD[8] at RDD at …\n |  purplecow.txt MappedRDD[7] at textFile
at …[]\n |  purplecow.txt HadoopRDD[6] at textFile at …[]
```

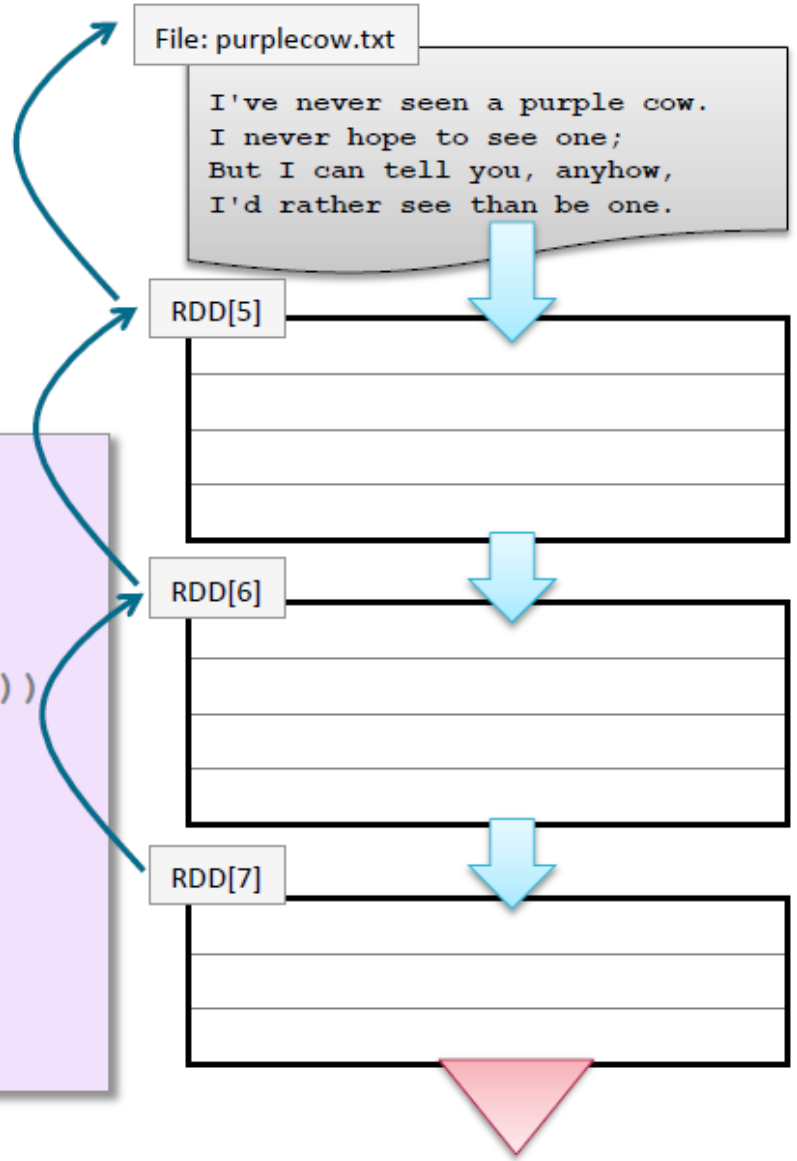
- Use `print` for prettier output

```
> print mydata_filt.toDebugString()
(1) PythonRDD[8] at RDD at …
 |  purplecow.txt MappedRDD[7] at textFile at …
 |  purplecow.txt HadoopRDD[6] at textFile at …
```

RDD Lineage and toDebugString (Scala)

- Spark maintains each RDD's *lineage* – the previous RDDs on which it depends
- Use `toDebugString` to view the lineage of an RDD

```
> val mydata_filt =
    sc.textFile("purplecow.txt").
    map(line => line.toUpperCase()).
    filter(line => line.startsWith("I"))
> mydata_filt.toDebugString

(2) FilteredRDD[7] at filter …
 |  MappedRDD[6] at map …
 |  purplecow.txt MappedRDD[5] …
 |  purplecow.txt HadoopRDD[4] …
```



Practice: Use RDDs to Transform a Dataset (1)

Files and Data used in this practice

- Exercise directory: $DEV1/exercises/spark-transform
- Data files: /loudacre/weblogs/* (HDFS)
  - Original files are located in /home/training/training_materials/data/static_data/weblogs
  - Upload this directory into HDFS

Explorer the Loudacre Web log files

- Using the HDFS command line or Hue File browser, review one of the files in the HDFS /loudacre/weblogs directory, e.g., FlumeData.1423586038966

```
$ ls -al /home/training/training_materials/data/static_data/weblogs
total 160008
drwxrwxr-x 2 training training  20480 Aug 27  2015 .
drwxr-xr-x 7 training training   4096 Sep 18  2015 ..
-rw-rw-r-- 1 training training 527908 Aug 27  2015 FlumeData.1424275921226
-rw-rw-r-- 1 training training 527933 Aug 27  2015 FlumeData.1424275921227
-rw-rw-r-- 1 training training 527874 Aug 27  2015 FlumeData.1424275921228
...
-rw-rw-r-- 1 training training 527883 Aug 27  2015 FlumeData.1424275921534
-rw-rw-r-- 1 training training 527899 Aug 27  2015 FlumeData.1424275921535
-rw-rw-r-- 1 training training  21836 Aug 27  2015 FlumeData.1424275921536

$ cat /home/training/training_materials/data/static_data/weblogs/FlumeData.1424275921536
249.25.86.121 - 57351 [15/Mar/2014:00:31:22 +0100] "GET /theme.css HTTP/1.0" 200 16800 "http://
www.loudacre.com"  "Loudacre Mobile Browser Sorrento F41L"
...
22.252.151.164 - 23251 [15/Mar/2014:00:00:30 +0100] "GET /theme.css HTTP/1.0" 200 10684 "http://
www.loudacre.com"  "Loudacre Mobile Browser Sorrento F11L"

$ hadoop fs -mkdir -p /loudacre/weblogs
$ hadoop fs -put /home/training/training_materials/data/static_data/weblogs/* /loudacre/weblogs/
```

Use RDDS to Transform

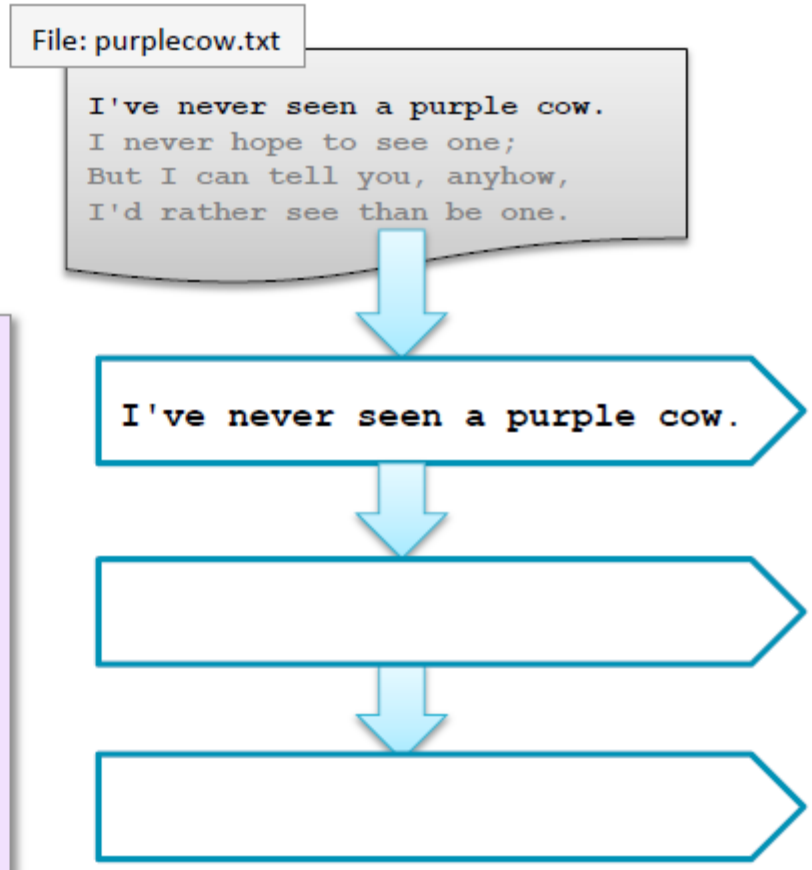
1. Create an RDD from the data file
2. Create an RDD containing only those lines that are requests for JPG files (i.e., the file extension is .jpg)
3. View the first lines of the data
4. Count the number of JPG requests by using a single line command
5. Print the length of each line in the log file for the first five lines
6. Print an array of words of each line in the log file for the first five lines

```
$ pyspark

> logfile="/loudacre/weblogs/FlumeData.*"
> logs=sc.textFile(logfile)
> logs.take(5)
Out[]:
[u'3.94.78.5 - 69827 [15/Sep/2013:23:58:36 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 14417 "http://www.loudacre.com"  "Loudacre Mobile Browser iFruit 1"',
 u'3.94.78.5 - 69827 [15/Sep/2013:23:58:36 +0100] "GET /theme.css HTTP/1.0" 200 3576 "http://www.loudacre.com"  "Loudacre Mobile Browser iFruit 1"',
 u'19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00277.html HTTP/1.0" 200 15517 "http://www.loudacre.com"  "Loudacre Mobile Browser Ronin S1"',
 u'19.38.140.62 - 21475 [15/Sep/2013:23:58:34 +0100] "GET /theme.css HTTP/1.0" 200 13353 "http://www.loudacre.com"  "Loudacre Mobile Browser Ronin S1"',
 u'129.133.56.105 - 2489 [15/Sep/2013:23:58:34 +0100] "GET /KBDOC-00033.html HTTP/1.0" 200 10590 "http://www.loudacre.com"  "Loudacre Mobile Browser Sorrento F00L"']

> jpglogs=logs.filter(lambda x: ".jpg" in x)
> jpglogs.collect()
> sc.textFile(logfile).filter(lambda x: ".jpg" in x).count()
Out[]: 64978

> length = logs.map(lambda s: len(s))
> length.take(5)
Out[]: [151, 143, 154, 147, 160]
```

Pipelining (1)

- When possible, Spark will perform sequences of transformations by row so no data is stored

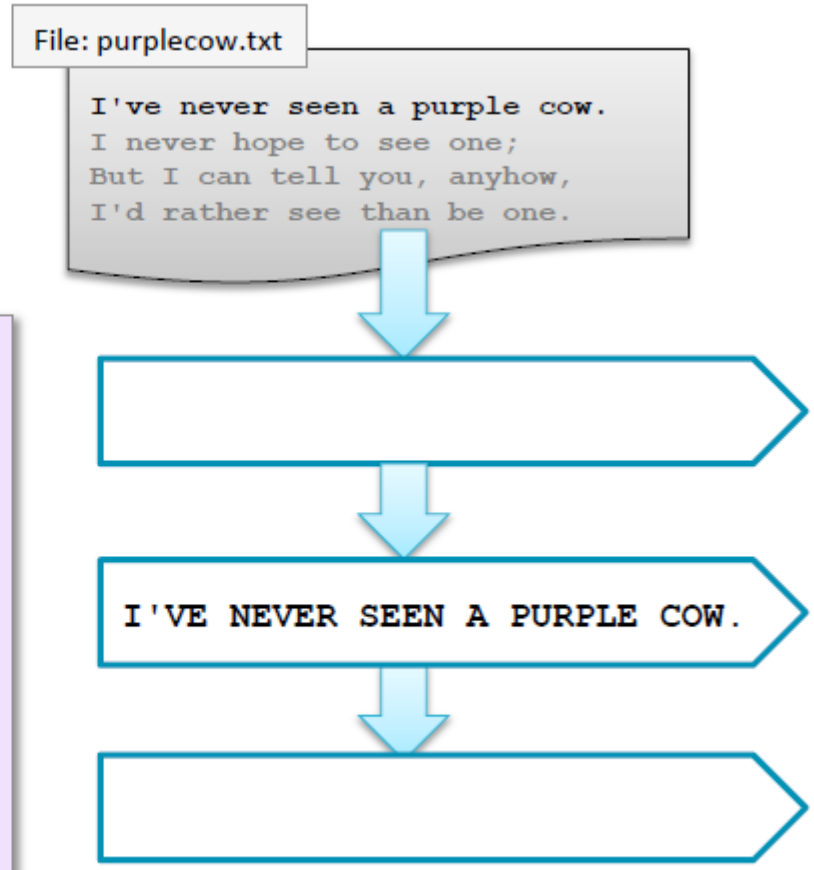
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
```



Pipelining (2)

- When possible, Spark will perform sequences of transformations by row so no data is stored

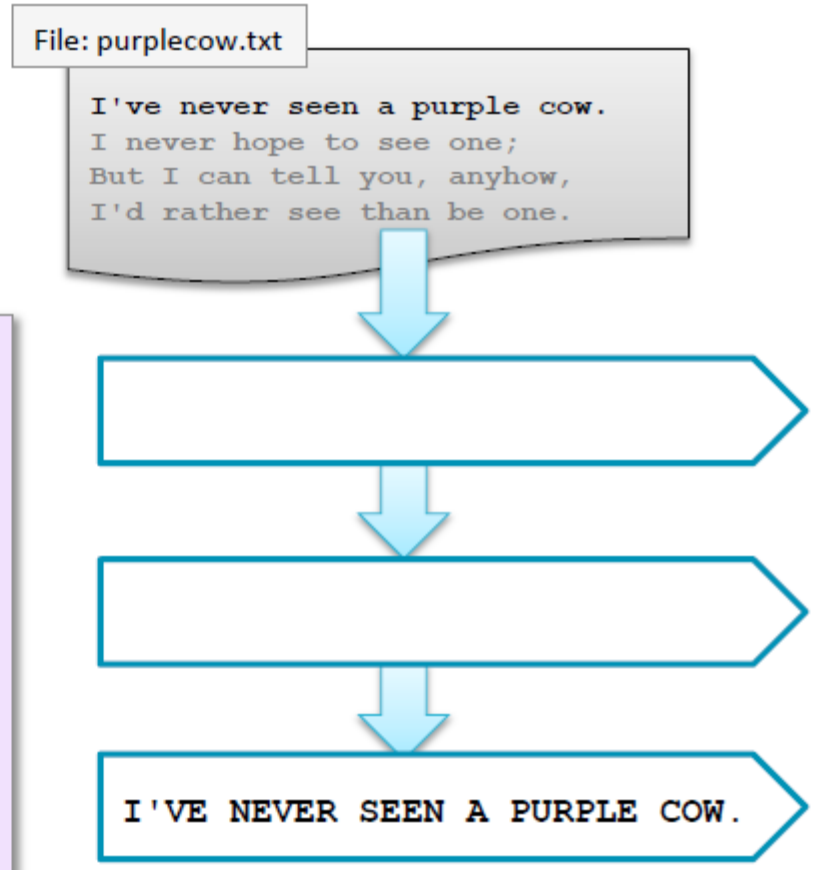
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
```



Pipelining (3)

- When possible, Spark will perform sequences of transformations by row so no data is stored

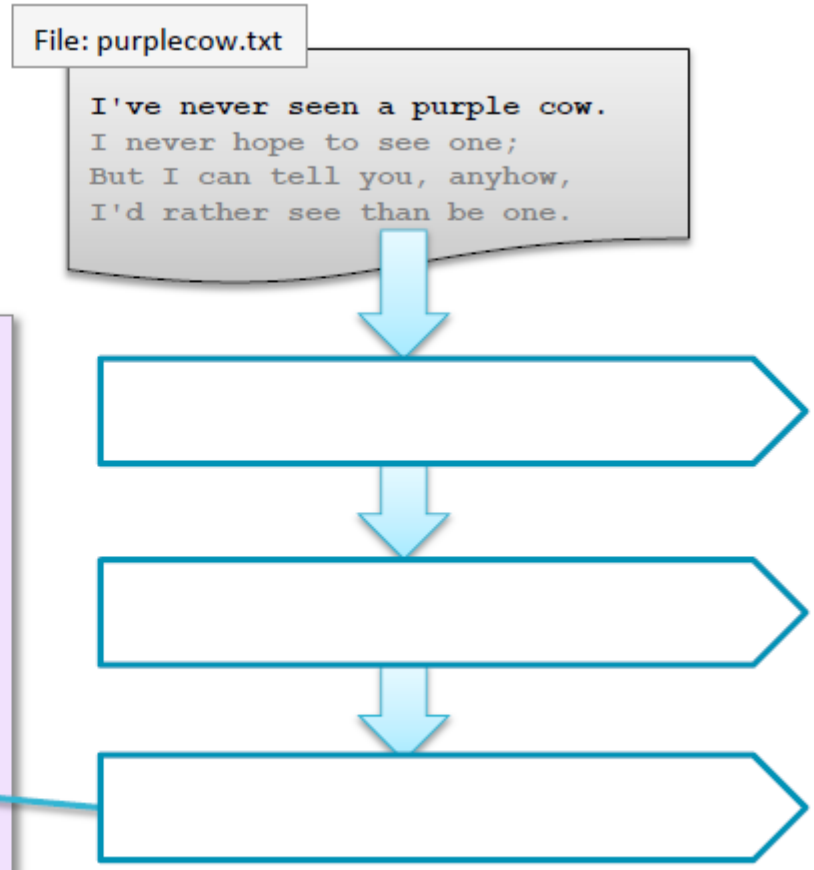
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
```



Pipelining (4)

- When possible, Spark will perform sequences of transformations by row so no data is stored

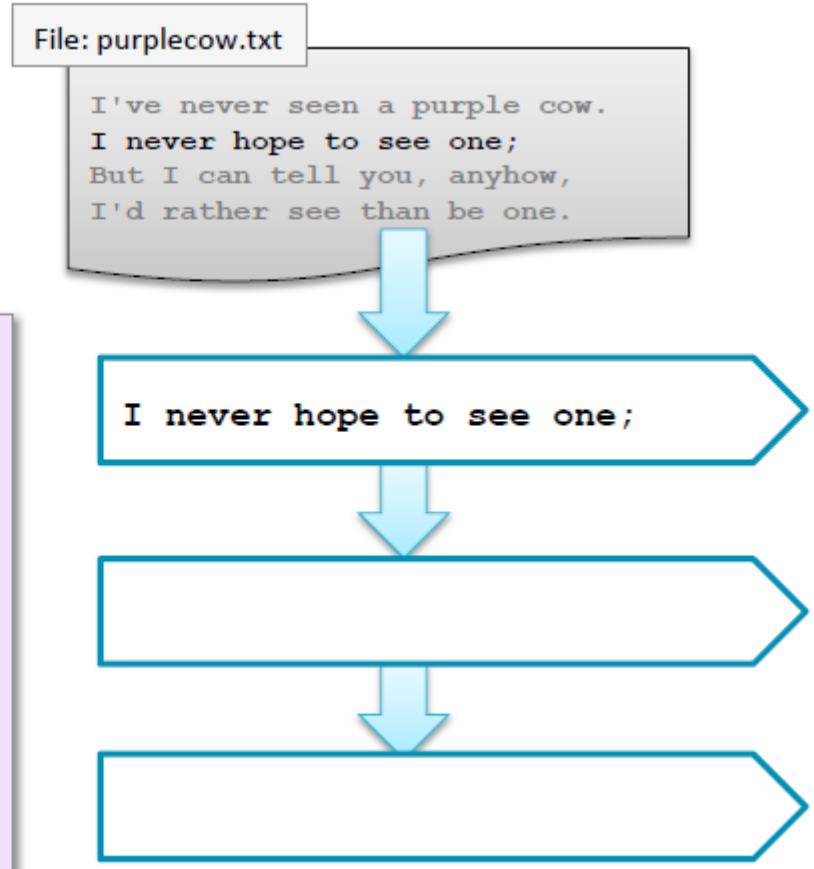
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
I'VE NEVER SEEN A PURPLE COW.
```



Pipelining (5)

- When possible, Spark will perform sequences of transformations by row so no data is stored

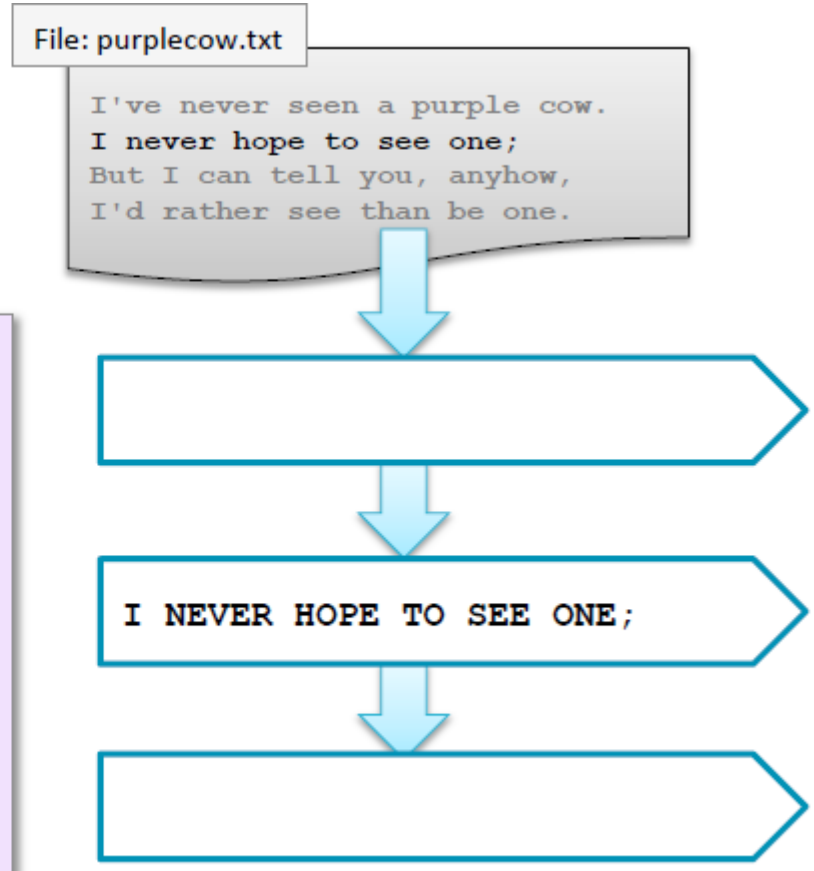
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
I'VE NEVER SEEN A PURPLE COW.
```



Pipelining (6)

- When possible, Spark will perform sequences of transformations by row so no data is stored

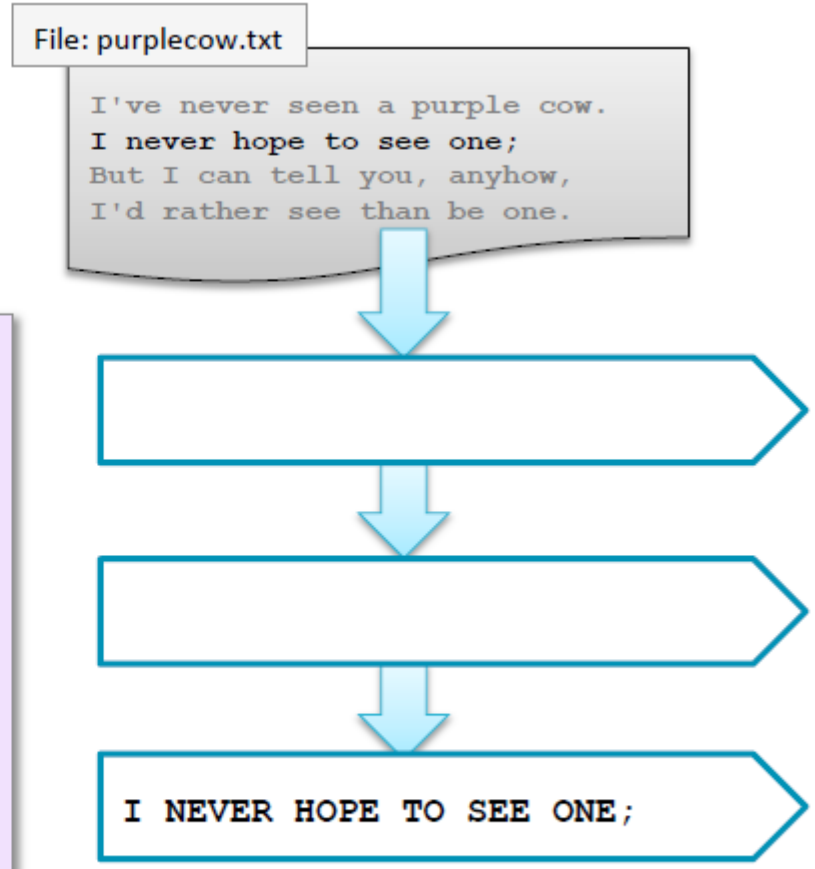
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
I'VE NEVER SEEN A PURPLE COW.
```



Pipelining (7)

- When possible, Spark will perform sequences of transformations by row so no data is stored

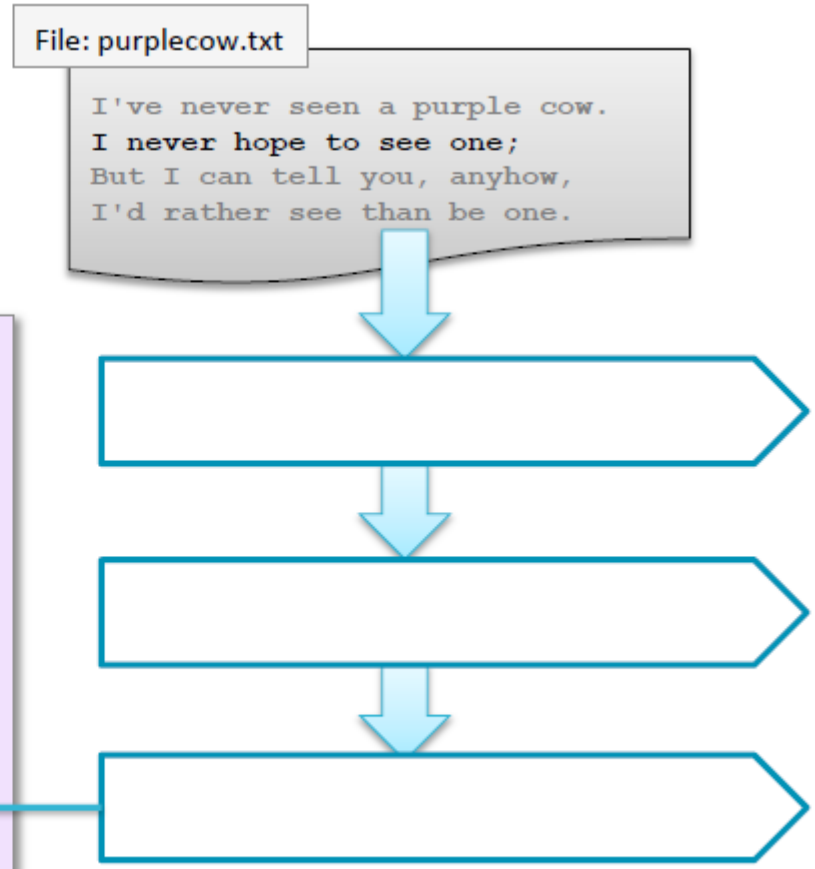
```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
I'VE NEVER SEEN A PURPLE COW.
```



Pipelining (8)

- When possible, Spark will perform sequences of transformations by row so no data is stored

```
> val mydata = sc.textFile("purplecow.txt")
> val mydata_uc = mydata.map(line =>
  line.toUpperCase())
> val mydata_filt = mydata_uc.filter(line
  => line.startsWith("I"))
> mydata_filt.take(2)
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
```



Practice: Use RDDs to Transform a Dataset (2)

1. Define a new RDD, called *ips*, containing just the IP addresses from each line in the log file. The IP address is the first "word" in each line.
2. Use for loop to print each IP in ips (syntax - for a in SetA: print a)
3. Save the list of IP addresses as a text file
4. Define a new RDD to create a dataset consisting of the IP address and corresponding user ID for each request for an HTML file (Disregard requests for other file types). The user ID is the third field in each log file line. Display the data in the form ipaddress/userid, e.g.,:

```
165.32.101.206/8
100.219.90.44/102
182.4.148.56/173
246.241.6.175/45395
175.223.172.207/4115
…
```

```
$ pyspark

> logfile="/loudacre/weblogs/FlumeData.*"
> logs=sc.textFile(logfile)
> ips = logs.map(lambda line: line.split()[0])
> ips.saveAsTextFile("file:/home/training/iplist")

$ cd iplist
$ ls
$ cat part-00000
3.94.78.5
3.94.78.5
19.38.140.62
19.38.140.62
129.133.56.105
...

> htmllogs = logs.filter(lambda s: ".htm" in s).map(lambda s: (s.split()[0], s.split()[2]))
> for x in htmllogs.take(5): print x[0]+"/"+x[1]
3.94.78.5/69827
19.38.140.62/21475
129.133.56.105/2489
217.150.149.167/4712
209.151.12.34/45922
```

Functional Programming in Spark

- **Spark depends heavily on the concepts of *functional programming***
  - Functions are the fundamental unit of programming
  - Functions have input and output only
    - No state or side effects
- **Key concepts**
  - Passing functions as input to other functions
  - Anonymous functions

Passing Functions as Parameters

- Many RDD operations take functions as parameters
- Pseudocode for the RDD map operation
  - Applies function **`fn`** to each record in the RDD

```
RDD {
    map(fn(x)) {
        foreach record in rdd
        emit fn(record)
    }
}
```

Example: Passing Named Functions

- Python

```
> def toUpper(s):
     return s.upper()
> mydata = sc.textFile("purplecow.txt")
> mydata.map(toUpper).take(2)
```

- Scala

```
> def toUpper(s: String): String =
    { s.toUpperCase }
> val mydata = sc.textFile("purplecow.txt")
> mydata.map(toUpper).take(2)
```

Anonymous Functions

- **Functions defined in-line without an identifier**
  - Best for short, one-off functions
- **Supported in many programming languages**
  - Python: `lambda x: ...`
  - Scala: `x => ...`
  - Java 8: `x -> ...`

Example: Passing Anonymous Functions

- **Python:**

```
> mydata.map(lambda line: line.upper()).take(2)
```

- **Scala:**

```
> mydata.map(line => line.toUpperCase()).take(2)
```

OR

```
> mydata.map(_.toUpperCase()).take(2)
```

Scala allows anonymous parameters using underscore (_)

Example: Java

Java 7

```
...
  JavaRDD<String> lines = sc.textFile("file");
  JavaRDD<String> lines_uc = lines.map(
    new MapFunction<String, String>() {
      public String call(String line) {
        return line.toUpperCase();
      }
    });
...
```

Java 8

```
...
 JavaRDD<String> lines = sc.textFile("file");
 JavaRDD<String> lines_uc = lines.map(
    line -> line.toUpperCase());
...
```

Essential Points

- **Spark can be used interactively via the Spark Shell**
  - Python or Scala
  - Writing non-interactive Spark applications will be covered later
- **RDDs (Resilient Distributed Datasets) are a key concept in Spark**
- **RDD Operations**
  - Transformations create a new RDD based on an existing one
  - Actions return a value from an RDD
- **Lazy Execution**
  - Transformations are not executed until required by an action
- **Spark uses functional programming**
  - Passing functions as parameters
  - Anonymous functions in supported languages (Python and Scala)