Spark Basics

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?
  - The principles of functional programming and how Spark uses them

Instructor Note

- Run the Demo setup script

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Review
- References
- Hands-On Exercise

What Is Apache Spark?

- Apache Spark is a fast, general purpose engine for large-scale data processing
- Written in Scala
  - A functional programming language that runs in a JVM
- Spark current version 1.5 (released 9/9/2015)
  - In this course we use version 1.0 (released 5/30/2014)
- Spark Shell
  - Interactive, for learning or data exploration
  - Supports Python or Scala (not Java, though a Java interactive shell is under development)
- Spark Applications
  - For large scale data processing
  - Written in Python, Scala, or Java
  - R support introduced with Spark 1.5

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Summary
- Review
- References
- Hands-On Exercise

The Spark Shell (Hands-On)

- The Spark Shell provides the ability to explore data interactively
  - In a Read-Evaluate-Print loop (REPL)

Python Shell: pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/


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


In [1]:
```

Scala Shell: spark-shell

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/
Using Scala version 2.10.4 (Java HotSpot(TM)
64-Bit Server VM, Java 1.7.0_51)
...
Spark context available as sc.


scala>
```

Basic Syntax (Hands-On)

Python Shell: pyspark

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/


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

In [1]: x = 10
In [2]: print x
10
In [3]: x = x + 1
In [4]: print x
11
# Clear the screen with [Ctrl][L]
exit
```

Spark Shell: spark-shell

```
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/
Using Scala version 2.10.4 (Java HotSpot(TM)
64-Bit Server VM, Java 1.7.0_51)
...
Spark context available as sc.

scala> var x = 10
x: Int = 10
scala> println(x)
10
scala> x = x + 1
x: Int = 11
scala> println(x)
11
scala> val y = 10
y: Int = 10
scala> println(y)
10
scala> y = y + 1
<console>:14: error: reassignment to val
       y = y + 1
// Clear the screen with [Ctrl][L]
exit
```

Additional Shell Features Of Note (1 Of 2)

- Getting help from the command line
  - pyspark --help
  - spark-shell --help
- Launch the Spark shell
  - pyspark (Python)
  - spark-shell (Scala)
  - Read the log messages
- Each Spark shell runs its own web server, by default on port 4040
  - Access from the browser built into the VM
    - http://localhost:4040
  - What happens if you try to access the UI from a browser outside of your VM?
    - It won't work, because this browser does not have access to the server running in the VM
- What happens if you run both shells concurrently?
  - The second shell reports an error during bootstrapping, because the port on which its web server wants to run (4040) has already been claimed by the other shell
    - The second shell falls back automatically to port 4041

Additional Shell Features Of Note (2 Of 2)

- How do you enter multi-line commands?
  - Python
    - Terminate all but the final line of the command with a \
  - Scala
    - Split the line between any token

Python Shell: pyspark

```
In [1]: rdd = sc.textFile \
            ("purplecow.txt")
```

Scala Shell: spark-shell

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

- How do you paste multiple commands into the shell at
  - Python: %cpaste[Enter] Data from Clipboard [Ctrl][D]

Python Shell: pyspark

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

Spark Shell: spark-shell

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

Spark Context

- Every Spark application requires a Spark Context
  - Serves as the main entry point to the Spark API
  - Represents your connection to the Spark cluster, telling Spark how to access the cluster
  - Provides access to other parts of the Spark API
- The Spark Shell provides a preconfigured Spark Context as a global variable named sc
  - [illegible]

Python Shell: pyspark	Scala Shell: spark-shell

`In [1]: sc.appName`
`Out[1]: u'PySparkShell'`
	`scala> sc.appName`
	`res0: String = Spark shell`

-

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Review
- References
- Hands-On Exercise

Resilient Distributed Dataset (RDD)

- The central abstraction, and fundamental unit of data in Spark
- A collection of elements partitioned across the nodes of the cluster that can be operated on in parallel
  - Resilient
    - If data in memory is lost, it can be recreated
  - Distributed
    - Data is stored in memory across the cluster, not limited to a single node
  - Dataset
    - A collection of data
- Today we focus on the DataSet itself
- Tomorrow we explore its Resilient, Distributed characteristics
- Most Spark programming consists of performing operations on RDDs
  - Loading an RDD (Input/Read)
  - Transforming an RDD (Process/Eval)
  - Analyzing or saving the result of processing the RDD (Output/Print)
- http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

Creating An RDD

- There are different ways to create an RDD
  - From a file or set of files
    - Loading the data from the file system
  - Parallelizing a collection (e.g., an array)
    - Dividing a non-distributed collection into a number of slices that will be distributed across multiple nodes
  - Transforming an existing RDD
    - Mapping the contents of one RDD to produce another
- Once created, an RDD is immutable
  - You can create a new RDD from an existing RDD
  - But you cannot alter the contents of an existing RDD

File-Based RDDs (1 Of 2)

- Another way to create an RDD is to load data into it from 1+ files using the Spark Context's textFile() method
  - Accepts a single file, a directory, a "globbed" list of files, or a comma-separated list of files

```
sc.textFile("myfile.txt")
sc.textFile("mydata/*.log")
sc.textFile("myfile1.txt, myfile2.txt")
```

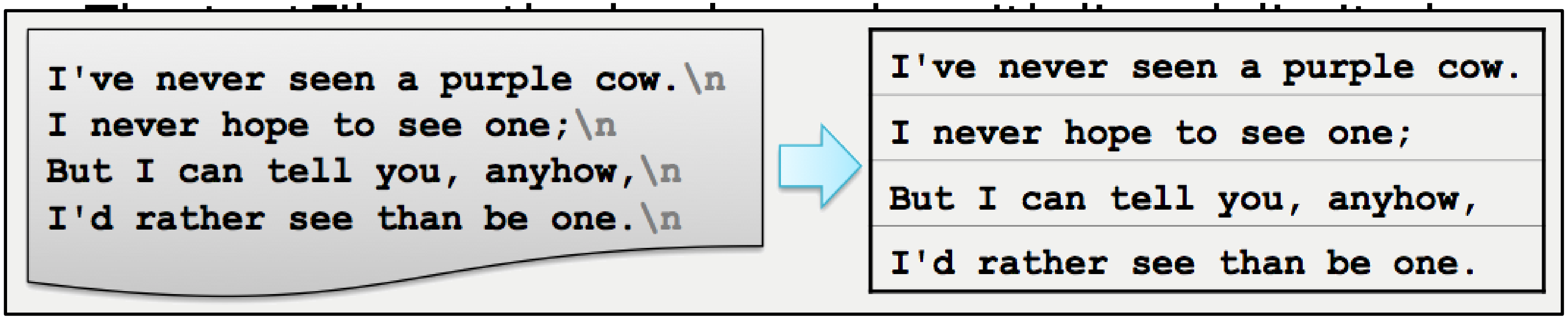
[illegible]

    - But the version of Spark used in this course defaults to HDFS
    - Note that you may include 1 or 3 slashes after the prefix

Absolute URI	prefix:/home/training/myfile.txt
Relative URI	myfile.txt

File-Based RDDs (2 Of 2)

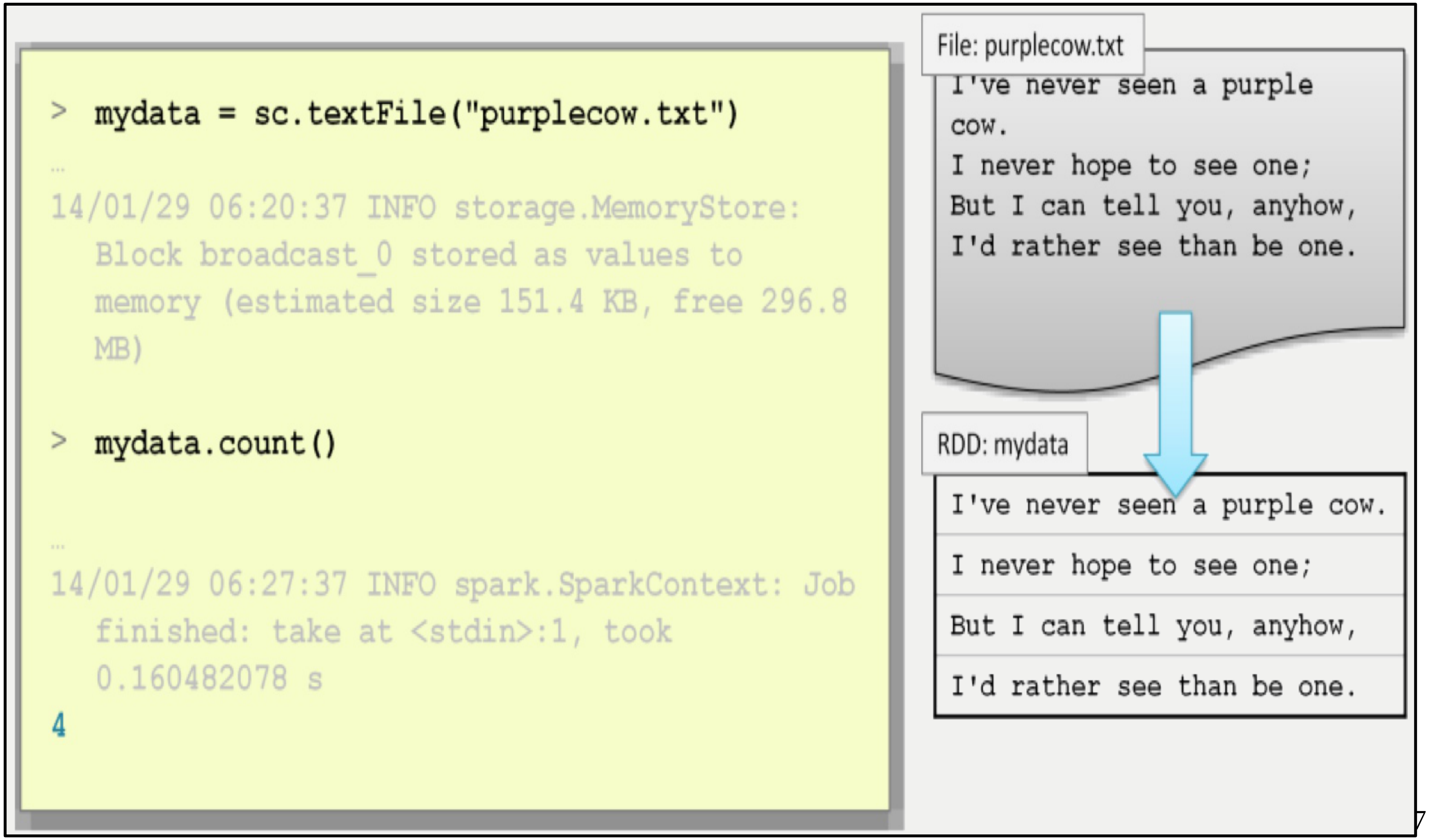
- The textFile method maps each line of an input file into a separate RDD element



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

```
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.
```

Example: A File-Based RDD



```
> mydata = sc.textFile("purplecow.txt")
…
14/01/29 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()

…
14/01/29 06:27:37 INFO spark.SparkContext: Job
   finished: take at <stdin>:1, took
   0.160482078 s
4
```

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.
```

RDD: mydata

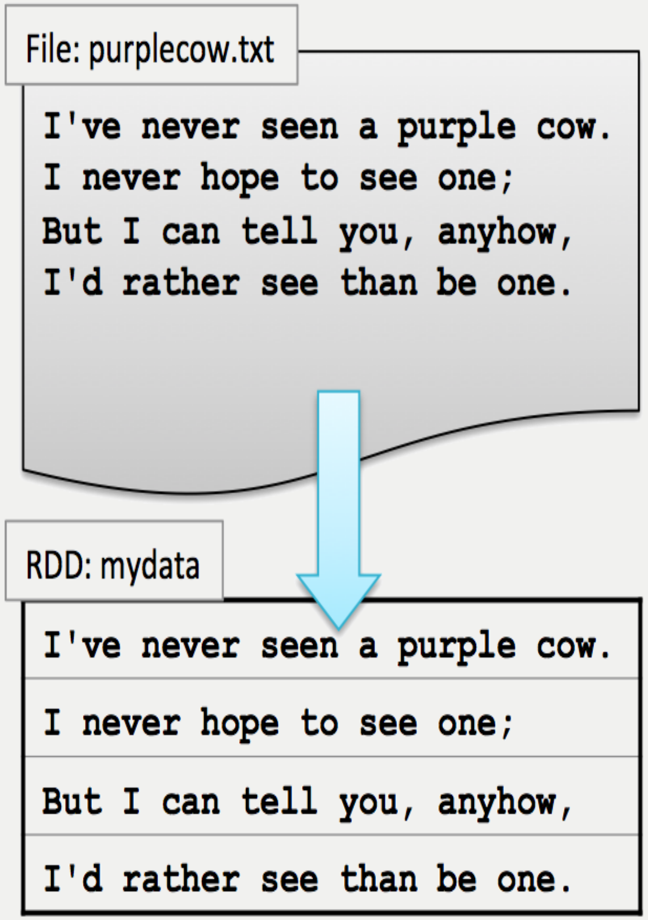
```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

Example: A Scala 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
```



File: purplecow.txt

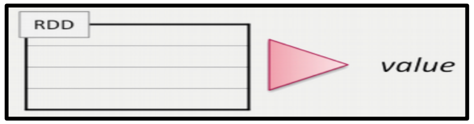
```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

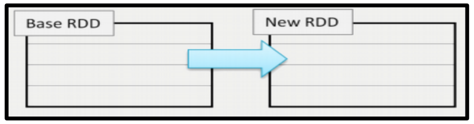
RDD: mydata

```
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.
```

RDD Operations

- There are two types of RDD operations
  - Action
    - Computes values
  - Transformation
    - Defines a new RDD from an existing RDD
- Based on this definition, is count an Action or a Transformation?





RDD Operations: Actions

- Common Actions
  - count()
    - Returns the number of elements in an RDD
  - take(n)
    - Returns an array of the first n elements in an RDD
  - collect()
    - Returns an array of all elements in an RDD
  - saveAsTextFile(file)
    - Saves an RDD to a text file(s)
- See Basic Spark Functions.odp

Python Shell: pyspark

```
> 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;
```

Spark Shell: spark-shell

```
> val mydata = sc.textFile("purplecow.txt")
> mydata.count()
Long = 4
> for (line <- mydata.take(2))
      println(line)
I've never seen a purple cow.
I never hope to see one;
```

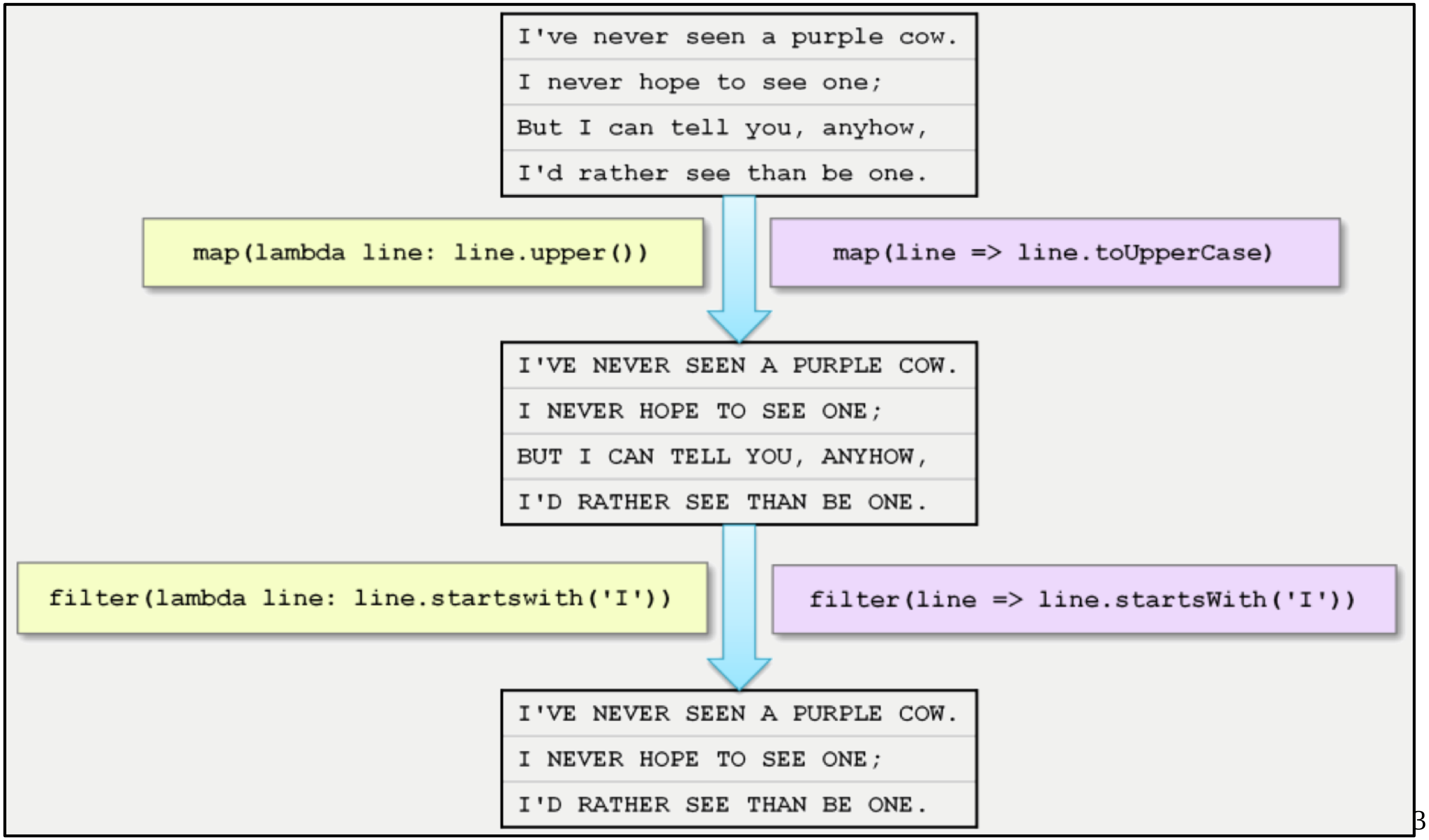
What's The Difference Between An RDD And An Array?

- Internal Representation
  - An RDD is a distributed dataset that resides on multiple machines
  - An array is a monolithic dataset that resides on a single machine
- Implementation
  - Arrays are native language constructs in Java, Scala and Python
  - RDDs are a Spark abstraction, implemented as a class with its own API
- API/Interface
  - RDDs and Arrays use different interfaces
  - The abstractions are the same, but because they are different data types, their interfaces differ

RDD Operations: Transformations

- A Transformation creates a new RDD from an existing RDD
- RDDs are immutable
  - Data in an RDD is never changed
  - Instead, create a new RDD, whose contents are the result of "transforming" (applying some function sequentially) to the elements of an existing RDD
- Common Transformations
  - map(function)
    - Creates a new RDD by applying a function (received as a parameter) to each element in the base RDD
      - e.g. square, pigLatin
    - Note that this function is unrelated to the map function in MapReduce
  - filter(function)
    - Creates a new RDD, including or excluding records from the base RDD by applying a boolean function to each record of the base RDD
      - e.g. isPrime, startsWith
- Spark programs are mostly made up of a series of transformations
- See Basic Spark Functions.odp

Example: Map And Filter Transformations

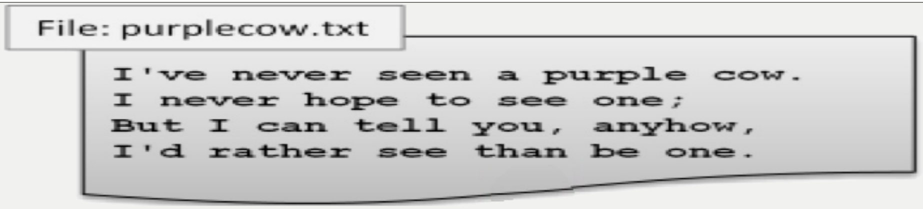

I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
map(lambda line: line.upper())
map(line => line.toUpperCase)
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
BUT I CAN TELL YOU, ANYHOW,
I'D RATHER SEE THAN BE ONE.
filter(lambda line: line.startswith('I'))
filter(line => line.startsWith('I'))
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
I'D RATHER SEE THAN BE ONE.

Lazy Execution

- Data in RDDs is not actually processed until an action is requested
  - Transformations are merely queued up for evaluation pending a request for an action
  - This is the same programming model as used in Apache Pig

Execution Sequence (1 Of 5)

- Start with a text file



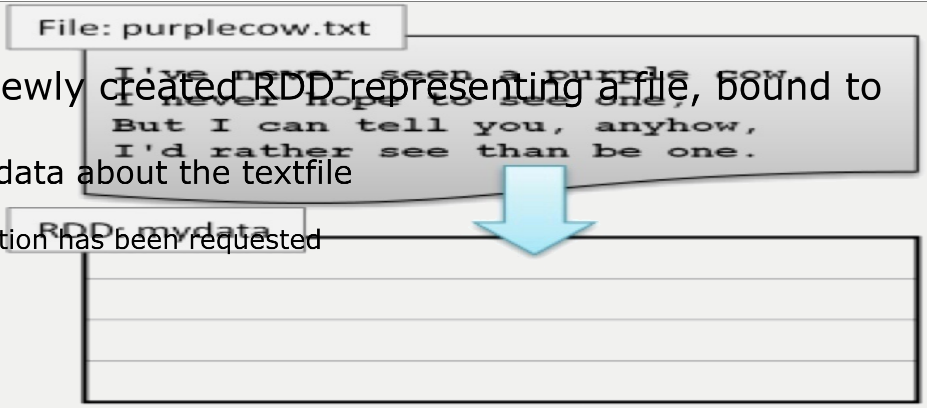
File: purplecow.txt

```
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.
```

Execution Sequence (2 Of 5)

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

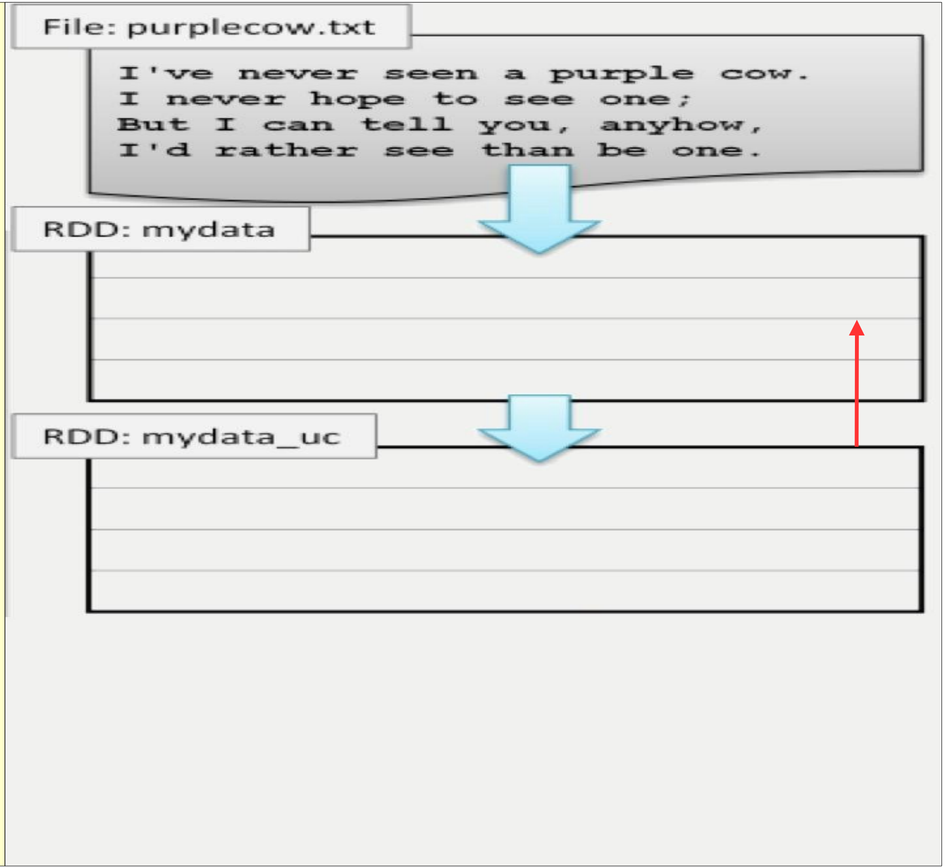
- Execute the textFile() function, returning a newly created RDD representing a file, bound to the name "mydata"
  - Note that mydata is an RDD representing metadata about the textfile
    - It is not a buffer
    - It is “empty” to represent “symbolically” that no action has been requested



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.

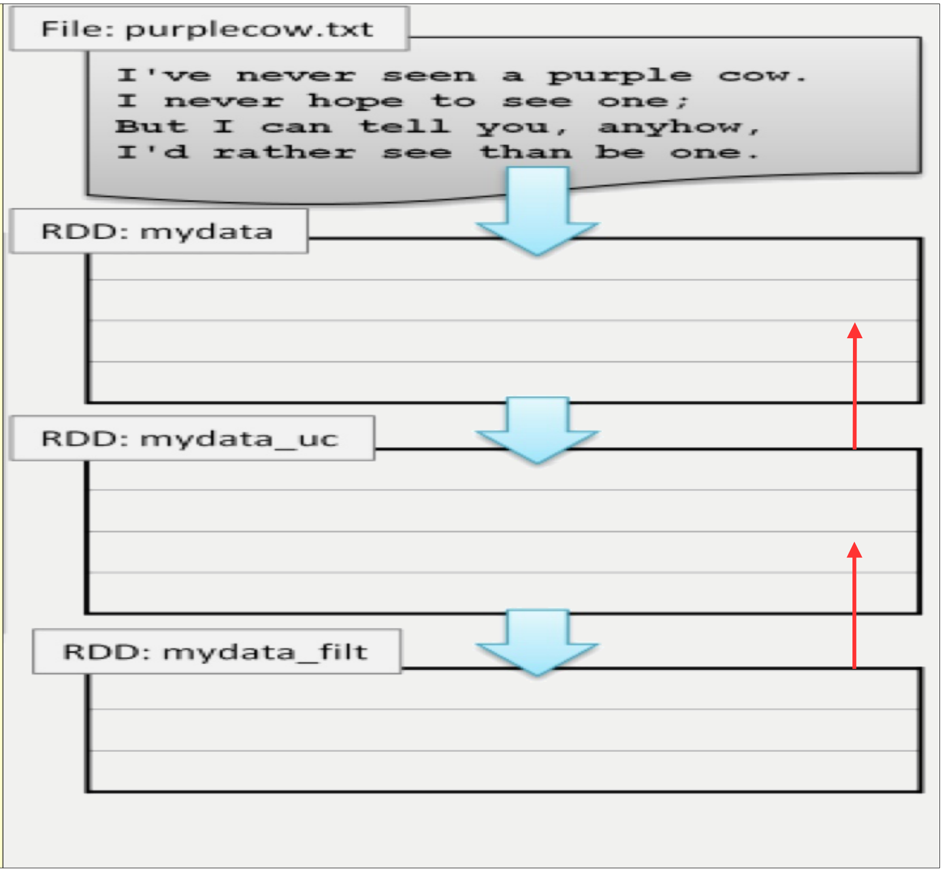
Execution Sequence (3 Of 5)

```
> mydata = sc.textFile("purplecow.txt")
> mydata_uc = mydata.map \
      (lambda line: line.upper())
```


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.
RDD: mydata
RDD: mydata_uc

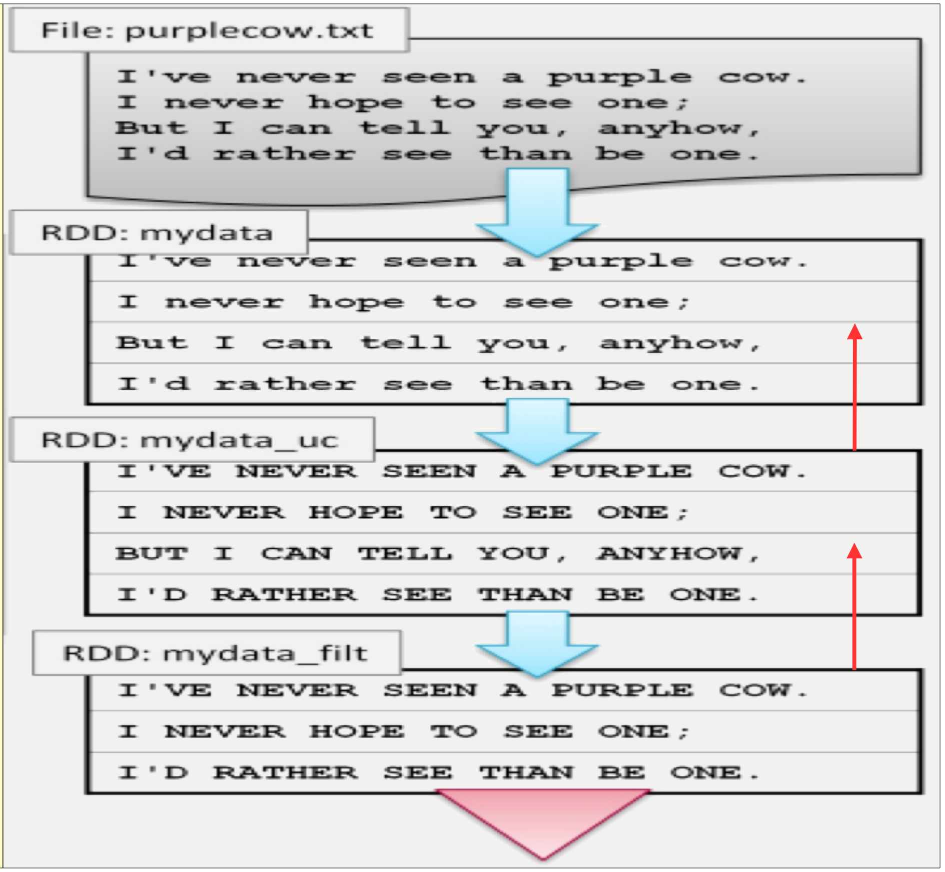
Execution Sequence (4 Of 5)

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



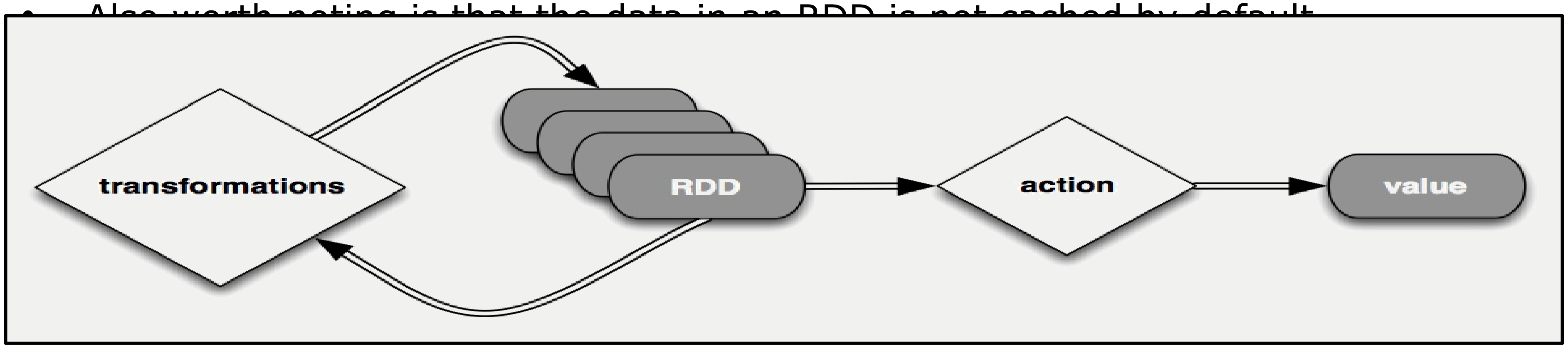
Execution Sequence (5 Of 5)

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


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.
RDD: mydata
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.
RDD: mydata_uc
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.
RDD: mydata_filt
I'VE NEVER SEEN A PURPLE COW.
I NEVER HOPE TO SEE ONE;
I'D RATHER SEE THAN BE ONE.

Spark Workflow

- Data in an RDD is not processed until an Action (in this case, invocation of the count() function) is performed
  - Lazy evaluation
    - e.g. When the textFile() function is invoked, all that happens is that a name is created for the RDD which is linked to the file
      - Thus, if you enter the wrong name for the file, you won't actually receive an error message until later
  - When an action is performed, Spark will work backward to the root transformation, before executing the sequence of transformations on which the action depends
- Also worth noting is that the data in an RDD is not cached by default



Chaining Transformations: Python

- Transformations may be chained together
  - Thus the following code blocks yield the same outcome

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

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

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

Chaining Transformations: Scala

- Transformations may be chained together
  - Thus the following code blocks yield the same outcome
    - Chaining can be a useful technique to avoid polluting the shell namespace with

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

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

```
> sc.textFile("purplecow.txt").map(line => line.upper())
      .count().filter(line => line.startswith('I'))
```

Food For Thought (1 Of 2)

- How would you write the following code in 1 statement?

Python Shell: pyspark

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

Spark Shell: spark-shell

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

Food For Thought (2 Of 2)

- How would you write the following code in 1 statement?

Python Shell: pyspark

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

Spark Shell: spark-shell

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

Python Shell: pyspark

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

Spark Shell: spark-shell

```
> { sc.textFile("purplecow.txt")
        .map(_.toUpperCase())
        .filter(_.startsWith("I"))
        .count()
  }
```

Chapter Topics

Functional Programming In Spark

- Spark depends heavily on the concepts of functional programming
  - Functions are the fundamental programming construct
  - Functions have input and output only (deterministic)
    - No mutable state or side effects
- Why functional programming?
  - Because the absence of state or side effects greatly simplifies the architecture of distributed computing and parallel processing environments
- Key features
  - Functions are first class citizens
    - They can be treated as code or data
  - Functions may be passed as input to other functions
  - Functions may be returned as output from other functions
  - Functions may be anonymous
    - Declared dynamically and discarded to avoid polluting the namespace with short-lived names

Receiving Functions As Parameters

- Many RDD operations receive functions as parameters
  - When this happens the RDD will invoke the function received in its parameter list as part of its operation
- Pseudocode for the RDD map operation

```
RDD {
    map(fn(x)) {
        foreach value in rdd
            emit fn(value)
    }
}
...
rdd.map(square)
```

What's So Great About Passing Functions As Arguments To Other Functions? (1 Of 3)

- It lends itself well to problem decomposition and chaining
  - Breaking a problem down into layers, each of which builds on the work of the next layer
    - Analogous to a pipes and filter architecture
- It can be used to capture similarities in problems that might otherwise appear unrelated
  - Supporting the notion of abstraction
  - Lending itself to code reuse
    - e.g. The Template Method Design Pattern
- For example, do the following 3 problems seem similar or different?
  - Given a list of numbers, generate a second list of numbers whose values are 5% greater than those provided as input
  - Given a sentence generate its Pig-Latin equivalent
  - Given a photograph, generate its negative

What's So Great About Passing Functions As Arguments To Other Functions? (2 Of 3)

- Suppose that I already have written 3 low level functions
  - addFivePercent()
  - wordToPigLatin()
  - pixelToNegative()
- Suppose also that a library provides certain generic elements for representing and processing the members of a dataset
  - emptySet
    - A representation of an empty dataset
  - isEmpty()
    - A function that indicates weather a dataset is empty
  - join()
    - A function that returns a single dataset consisting of all of the members of multiple datasets received in its parameter list
  - firstElementOf()
    - A function that returns the value of the first element of a dataset
  - allButFirstElementOf()
    - A function that returns the values of all but the first element of a dataset

What's So Great About Passing Functions As Arguments To Other Functions? (3 Of 3)

- Having the ability to pass functions as arguments to other functions allows me to write (pseudo) code like this

```
define [templateMethod data function]
    if isEmpty(data)
        return emptySet
        return join(
            function(firstElementOf(data))
            templateMethod(allButFirstElementOf(data) function))
```

Expression	Outcome
templateMethod([10 20 30 40] add5Percent)	[10.5 21 31.5 42]
templateMethod([I Love Spark] wordToPigLatin)	[Iay oveLay arkSpay]
templateMethod([1 0 0 0 1] pixelToNegative)	[0 1 1 1 0]

Example: Passing Named Functions

Python Shell: pyspark

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

Spark Shell: spark-shell

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

string

- Of course, we could have just passed s.upper, but we wanted to show how we could define our own named function and pass that named function as an argument to another function

- In this example, we define a function which takes a single parameter
  - Spark will invoke that function for each item in the RDD, passing each one individually as the argument to the function in a separate function call

One Problem With Named Functions (1 Of 2)

- Functions are powerful tools for abstraction and problem decomposition
  - It is not uncommon in any program to find lots of small, computational functions, each of which might perform a single calculation on a single parameter, returning the result of that calculation as its outcome
    - e.g. y = square(x)
  - This is great when the function in question is reusable
- But many functions are "one-offs"
  - Self-contained pieces of code, designed to address a narrowly defined problem, offering the benefits of modularity and local scope, but without the benefit of reuse
  - Having to name such functions pollutes the application namespace with unnecessary names

One Problem With Named Functions (2 Of 2)

- Perhaps you recall the idea behind chaining transformations
  - If you only reference a variable twice, once to assign it the result of an expression, the other to reference that result by name, perhaps you should just refer to the expression directly in the context of a larger expression (chain), without actually naming it
- This idea can be applied to function usage as well
  - If you only reference a function twice, the first time to define its behavior, the second time to invoke that function, perhaps you should just define the function directly, in the context of a larger expression (as an argument to another function) without actually naming it
  - Which leads us to ...

What Are Anonymous Functions?

- Functions defined in-line without an identifier
  - Best for short, one-off functions
  - Common in chained transformations
- Supported in many programming languages
  - Python
  - Scala
  - Java 8
  - Scheme
  - Lisp

Passing Anonymous Functions As Arguments To Other Functions

- Python

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

- The code in these examples does the same thing as predecessors from earlier slides, but without first having to

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

  - Because the function does not have a name, we call it anonymous

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

- Note that Scala has a shortcut for referencing a value passed to an anonymous function: "_"
  - Replacing the need for the duplicate reference to "line" in this example
  - Be judicious with the use of this reference, as it makes the code more difficult to understand

Java Example

Java 7	Java 8

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

- Java 8 supports closures using a syntax similar to that used in Scala
  - This only works in a Java based Spark application (since there is no Spark interactive shell for Java)

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Review
- References
- Hands-On Exercise

Summary

- Spark can be used interactively via the Spark Shell
  - Currently supports Python and 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 a functional programming model
  - Which supports passing functions as arguments to other functions
  - Anonymous functions are also supported

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Review
- References
- Hands-On Exercise

Review

- What does RDD stand for?

Review Answer

- What does RDD stand for?
  - Resiliant
  - Distributed
  - Dataset

Review

- What are the two types of RDD operations?

Review Answer

- What are the two types of RDD operations?
  - Transformations
  - Actions

Review

- What are the differences between a Transformation and an Action?

Review Answer

- What are the differences between a Transformation and an Action?
  - Transformation
    - Defines a new RDD from an existing RDD
    - Evaluation is deferred until an Action is invoked
  - Action
    - Computes a value

Review

- What do the following common actions do?
  - count()
  - take(n)
  - collect()
  - saveAsTextFile(file)

Review Answer

- What do the following common actions do?
  - count()
    - Return the number of elements in an RDD
  - take(n)
    - Return an array of the first n elements in an RDD
  - collect()
    - Return an array of all elements in an RDD
  - saveAsTextFile(file)
    - Save an RDD to a text file(s)
-

Review

- What do the following common transformations do?
  - map(function)
  - filter(function)

Review Answer

- What do the following common transformations do?
  - map(function)
    - Creates a new RDD by applying a function to each record in the base RDD
      - e.g. square, pigLatin
  - filter(function)
    - Creates a new RDD, including or excluding records from the base RDD by applying a boolean function to each record of the based RDD
      - e.g. isPrime, isPalindrome

Review

- Which of the following are true for functional programming languages? (Choose all that apply)
  - Variables are immutable
  - Functions can be passed as arguments to other functions
  - Functions can be returned as output to other functions
  - Functions can be defined in the act of invoking other functions

Review Answer

- Which of the following are true for functional programming languages? (Choose all that apply)
  - Variables are immutable
  - Functions can be passed as arguments to other functions
  - Functions can be returned as output to other functions
  - Functions can be defined in the act of invoking other functions

Review

- A function defined in the act of invoking another function is called a(n) __________ function

Review Answer

- A function defined in the act of invoking another function is called a(n) anonymous function

Review

- What are the differences between an RDD and an array?

Review Answer

- What are the differences between an RDD and an array?
  - Internal Representation
    - An RDD is a distributed dataset that resides on multiple machines
    - An array is a monolithic dataset that resides on a single machine
  - Abstraction
    - Arrays are native language constructs in Java, Scala and Python
    - RDDs are a Spark abstraction, implemented as a class with its own API

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Review
- References
- Hands-On Exercise

References

- The following offer more information on topics discussed in this chapter
  - Official Spark Documentation
    - https://people.apache.org/~tdas/spark-1.0.0-rc11-docs/index.html
  - Read-Evaluate-Print loop
    - http://en.wikipedia.org/wiki/Read–eval–print_loop
  - Resiliant Distributed DataSet
    - http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
  - Python Course
    - https://developers.google.com/edu/python/
  - Scala Tutorial
    - http://ampcamp.berkeley.edu/4/exercises/introduction-to-the-scala-shell.html
  - Java 8 adds support for Spark
    - http://blog.cloudera.com/blog/2014/04/making-apache-spark-easier-to-use-in-java-with-java-8/

Chapter Topics

- What Is Apache Spark?
- Using The Spark Shell
- RDDs (Resilient Distributed Datasets)
- Functional Programming In Spark
- Summary
- Review
- References
- Hands-On Exercise

Introduction To Exercises: Pick Your Language

- Python or Scala
  - For most exercises in this course, you may choose to work with either Python or Scala
    - Exception: Spark Streaming is currently available only in Scala
  - Most course examples are presented in Python
- Solution and example files
  - .pyspark
    - Files containing sequences of python commands to be entered interactively in the python shell
  - .scalaspark
    - Files containing sequences of scala commands yo be entered interactively in the scala shell
  - .py
    - Complete Python Spark applications executed from the command line
  - .scala
    - Complete Scala Spark applications to be compiled and packaged before being executed from the command line

Introduction To Exercises: Classroom Virtual Machine

- Your virtual machine
  - Log in as user training (password training)
  - Pre-installed and configured with
    - Spark 1.0 and CDH 5 (Cloudera's Distribution, including Apache Hadoop)
    - Various tools including Emacs, IntelliJ, and Maven
- Training materials: ~/training_materials/sparkdev folder in the VM
  - data - Sample datasets used in exercises
  - examples - All the example code in this course
  - solutions - Solutions for Scala Shell and Python exercises
  - stubs - Starter code required in some exercises

Introduction To Exercises: The Data

- Most exercises are based around a hypothetical company: Loudacre Mobile
  - A cellular telephone company
- Loudacre Mobile Customer Support has many sources of data they need to process, transform and analyze
  - Customer account data
  - Web server logs from Loudacre’s customer support website
  - New device activation records
  - Customer support Knowledge Base articles
  - Information about models of supported devices

Hands-On Exercises

- Read the General Notes
- Do the following Hands-On Exercises
  - Setting Up
  - Viewing The Spark Documentation
  - Using The Spark Shell
  - Getting Started With RDDs
- Please refer to the Hands-On Exercise Manual
-