Lecture 4: Introduction to Spark

COSC 526: Introduction to Data Mining

Spring 2020

THE UNIVERSITY OF TENNESSEE KNOXVILLE

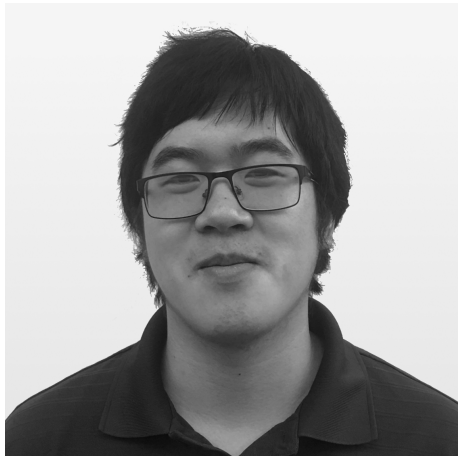
BIG ORANGE. BIG IDEAS.®

Instructors:

Michela Taufer

Danny Rorabaugh

GRA:



Nigel Tan

Today Outline

- Overview of Spark core concepts
- Create SparkContext and Resilient distributed datasets (RDDs)
- Run parallel operations on RDDs
- Implement sequential map and reduce functions
  - Create appreciation for future parallel implementations

Spark Introduction

Spark reference

5

Spark is …

- … a “computational engine that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many **worker** machines, or a computing cluster.”

Running Spark

- Apache Spark has four APIs: Java, Scala, R, and **Python**
- Ways to use the Python API:
  - Interactively, using **pyspark**
  - Non-interactively, using **spark-submit**
  - Within the **Jupyter Notebook**
- Run interactively: open the Python version of the Spark shell
  bin/pyspark
- Run non-interactively: Write Python scripts and run script:
  bin/spark-submit my_script.py
- Run in Jupyter Notebook: Assignment 5

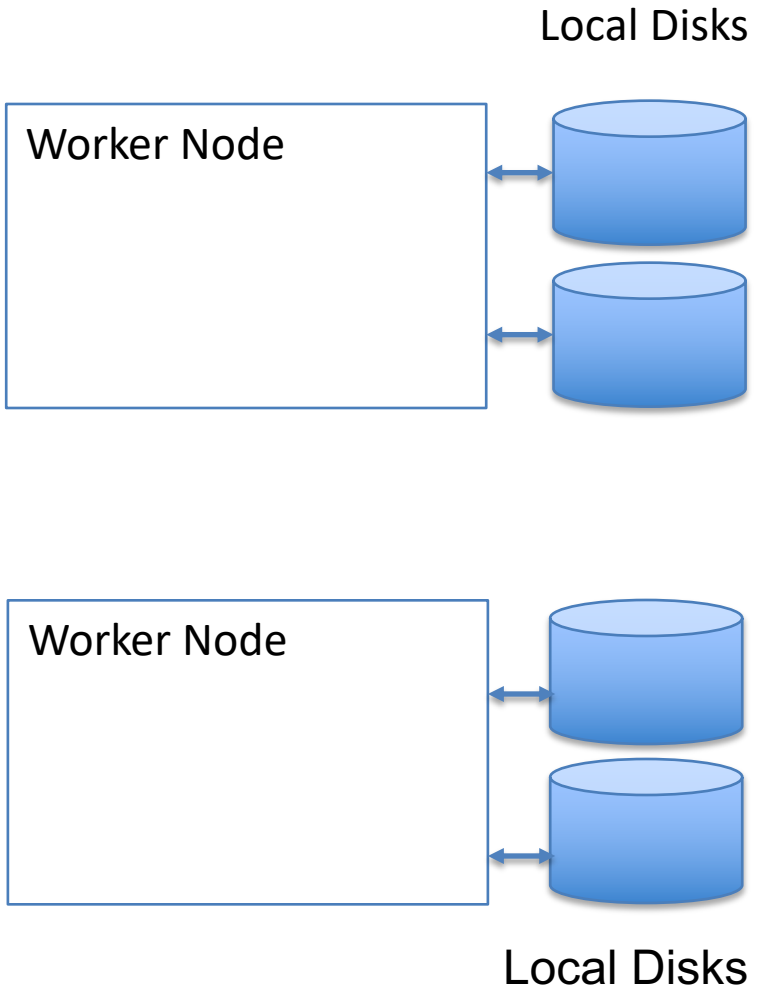
Running Spark

- Apache Spark has four APIs: Java, Scala, R, and **Python**
- Ways to use the Python API:
  - Interactively, using **pyspark**
  - Non-interactively, using **spark-submit**
  - Within the **Jupyter Notebook**
- Run interactively: open the Python version of the Spark shell
  bin/pyspark
- Run non-interactively: Write Python scripts and run script:
  bin/spark-submit my_script.py
- Run in Jupyter Notebook: Assignment 5

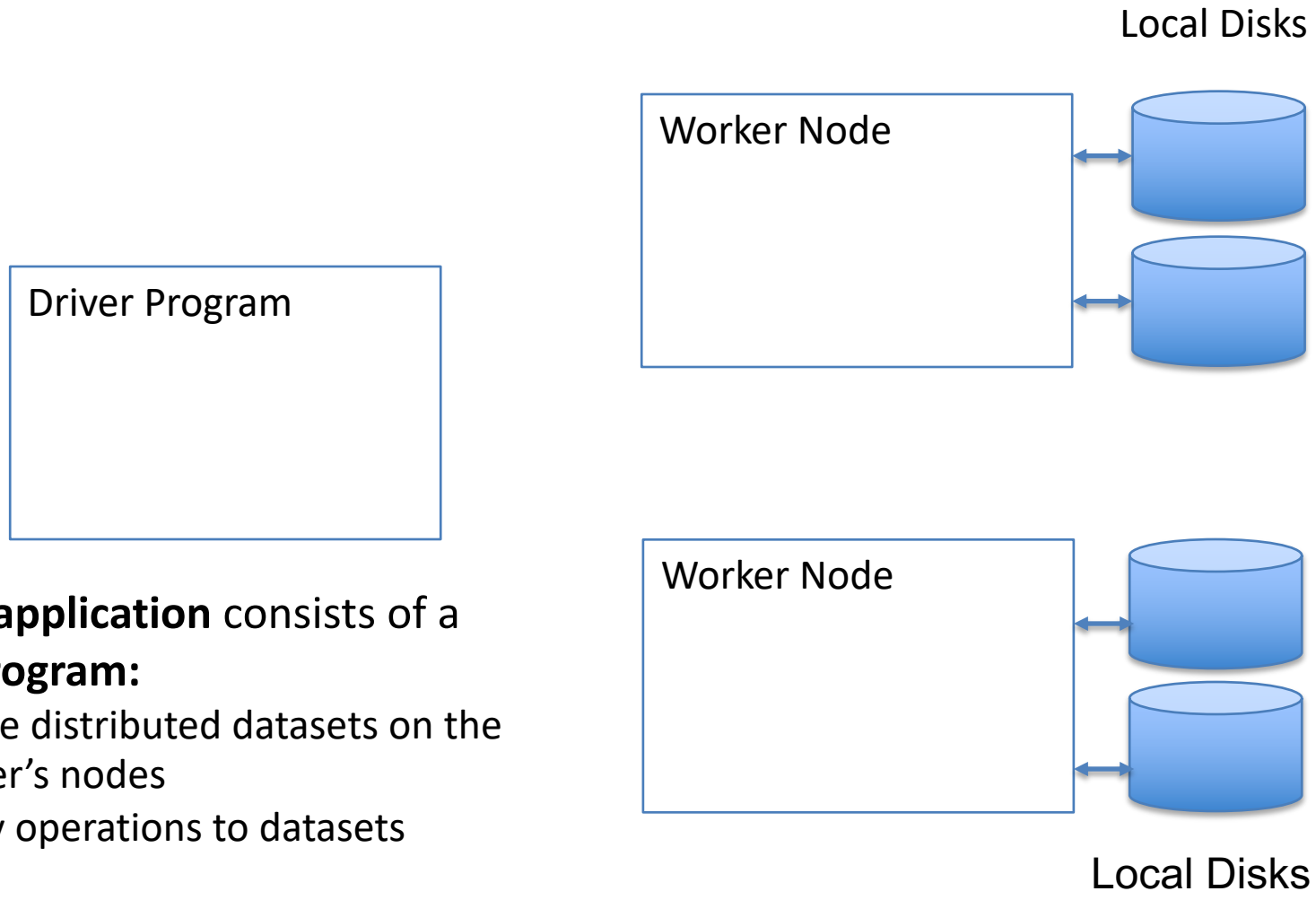
Spark Core Concepts

- A **Spark application** consists of a **driver program**
- A drive program:
  - Defines distributed datasets on the cluster
  - Applies operations to datasets
- A driver program accesses Spark through a SparkContext object
- A **SparkContext** represents a connection to a computing cluster
- Spark uses SparkContext to build resilient **distributed datasets (RDDS)**

Spark Core Concepts



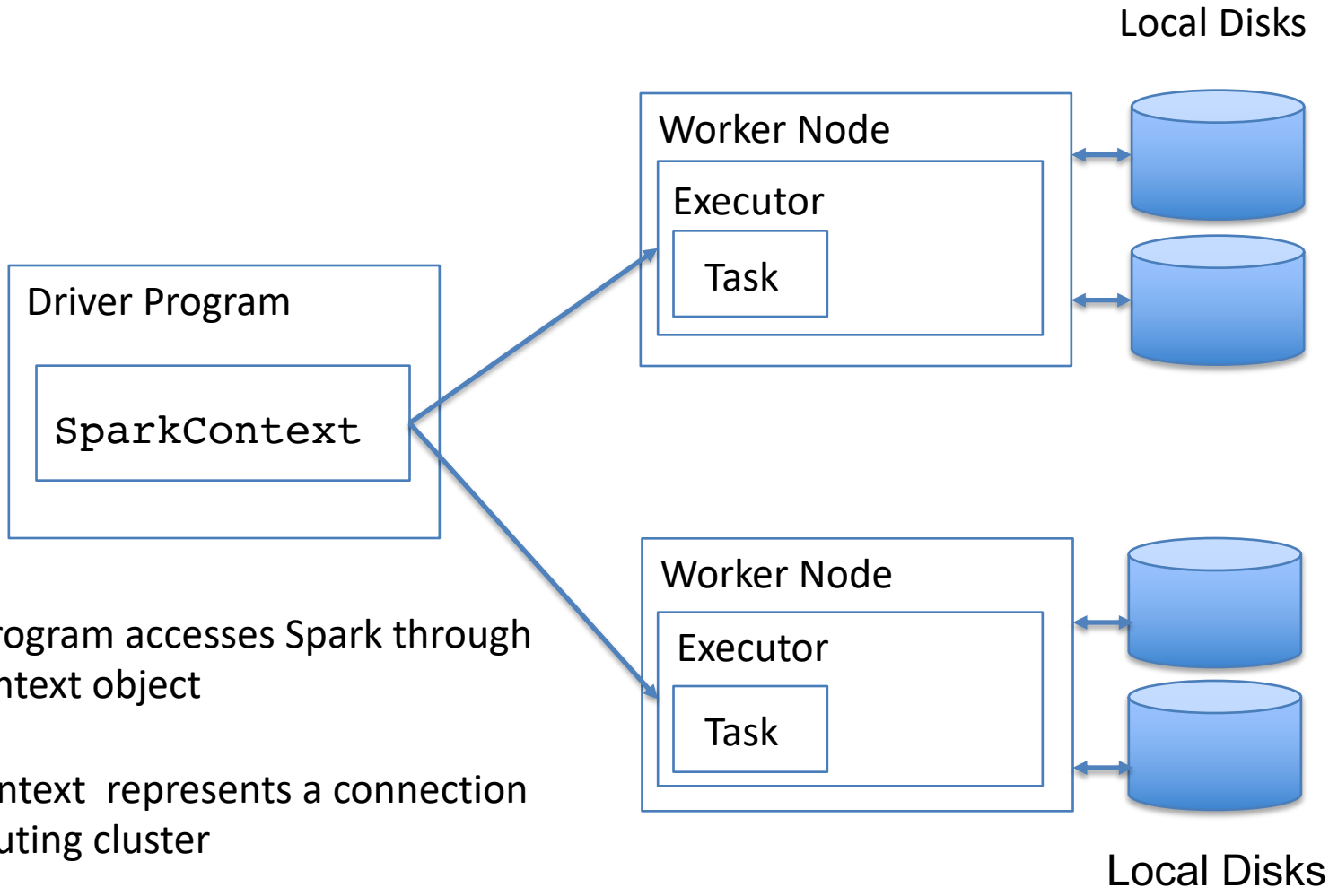
Spark Core Concepts



A **Spark application** consists of a **driver program:**

- Define distributed datasets on the cluster's nodes
- Apply operations to datasets

Spark Core Concepts



A driver program accesses Spark through a SparkContext object

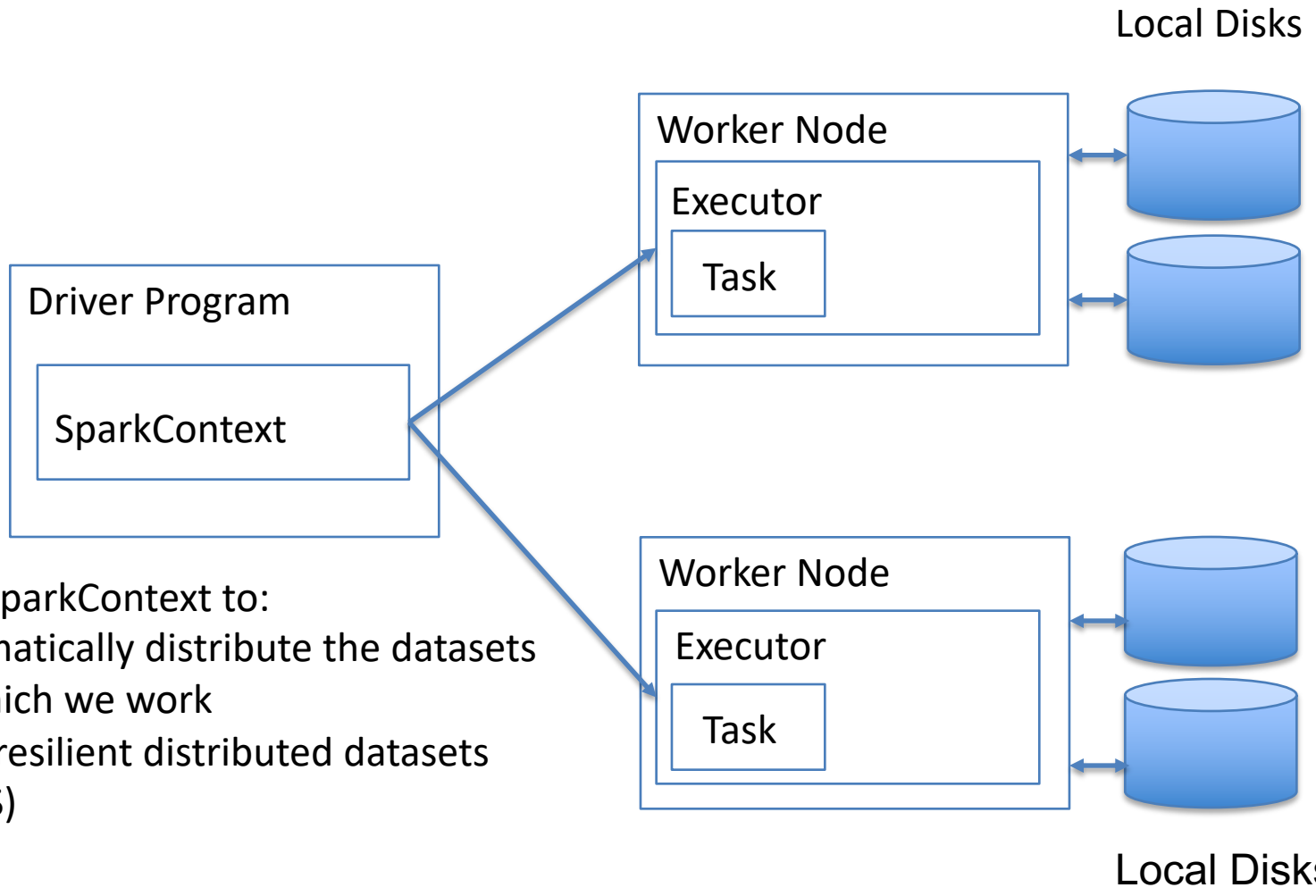
A SparkContext represents a connection to a computing cluster

Create a SparkContex

In [1]

```
from pyspark import SparkContext


sc = SparkContext.getOrCreate()
```

Spark Core Concepts



We use SparkContext to:

- Automatically distribute the datasets on which we work
- Build resilient distributed datasets (RDDS)

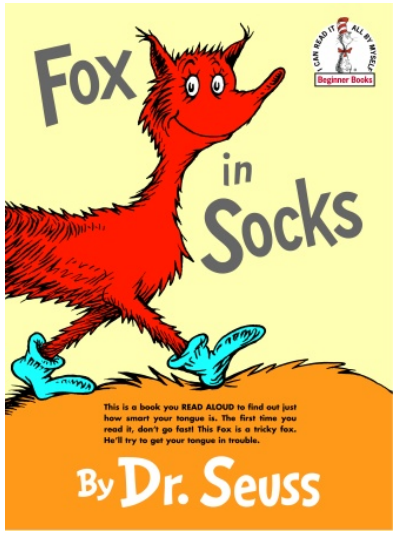
Resilient Distributed Datasets

- Spark operates on a distributed collections of data called Resilient Distributed Datasets (or RDDs)
- We express Spark computation through operations on RDDs
- Datasets are automatically distributed across a cluster
- Operations are automatically parallelized across a cluster

→ RDDs are Spark's fundamental abstraction for distributed data and computation

Given the file “FoxInSocks.txt”

When tweetle beetles fight,
it’s called a tweetle beetle battle.
And when they battle in a puddle,
it’s a tweetle beetle puddle battle.
And when tweetle beetles battle with paddles in a puddle,
They call it a tweetle beetle puddle paddle battle.

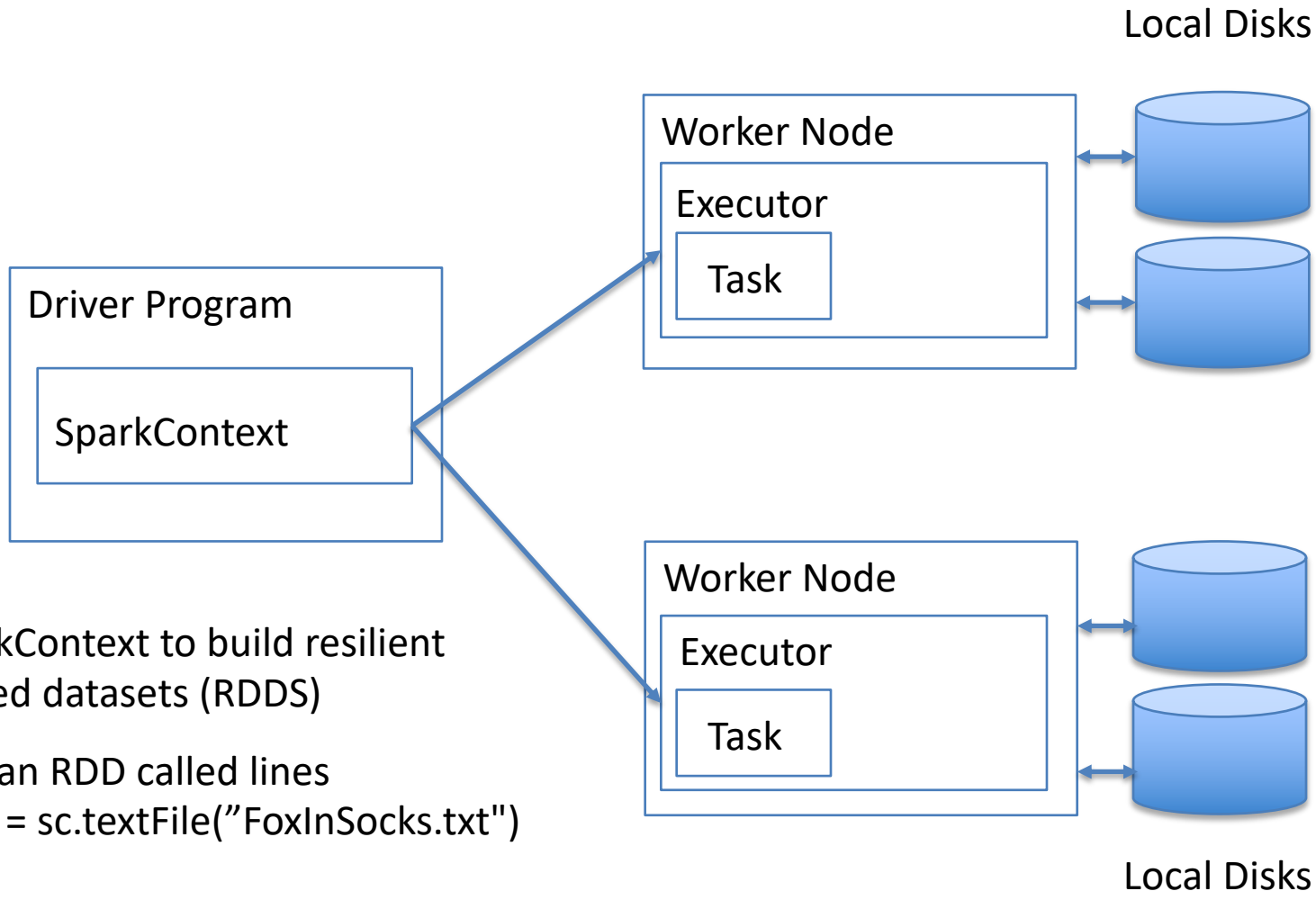


# Create an RDD called lines
>>> lines = sc.textFile(”FoxInSocks.txt")

File lines automatically distributed across nodes of 2-node cluster

Node 1	When tweetle beetles fight, it’s called a tweetle beetle battle. And when they battle in a puddle,
Node 2	it’s a tweetle beetle puddle battle. And when tweetle beetles battle with paddles in a puddle, They call it a tweetle beetle puddle paddle battle.

Spark Core Concepts



Use SparkContext to build resilient distributed datasets (RDDS)

```
# Create an RDD called lines
>>> lines = sc.textFile(”FoxInSocks.txt")
```

Create a SparkContex

In [1]

```
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

lines = sc.textFile(”FoxInSocks.txt")
```

Operations: Count Number of Lines

In [1]

```
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

lines = sc.textFile(”FoxInSocks.txt")

# Count the number of items in this RDD
>>> lines.count()
```

Operations: Print First Line

In [1]

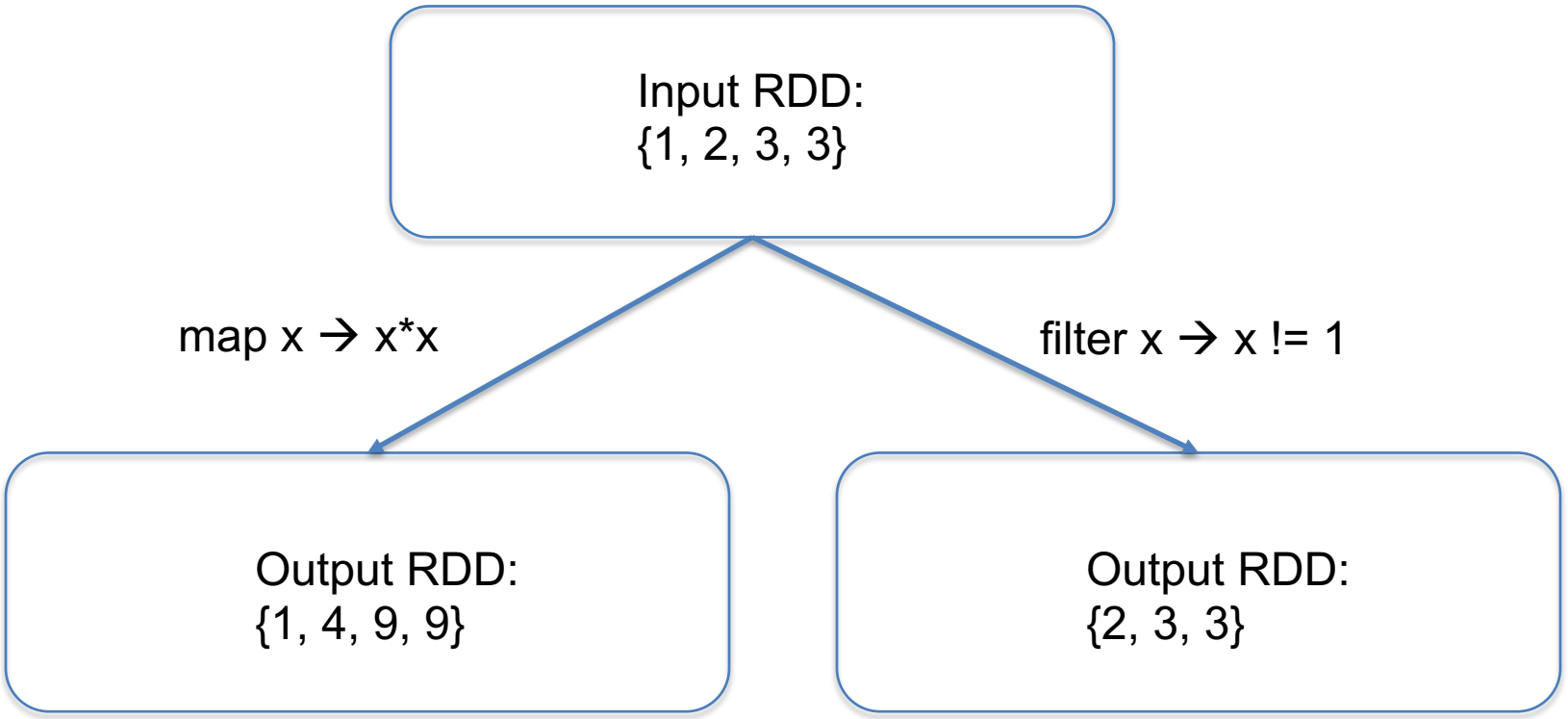
```
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

lines = sc.textFile(”FoxInSocks.txt")

# First item in this RDD, i.e. first line of FoxInSocks.txt
>>> lines.first()
```

Operations

- **Transformations:** lazily evaluated—no immediate computation
  - "Return" new RDDs obtained by transforming an old RDD
  - Input: RDD type → OPERATION → Output: RDD type
- **Actions:** cause all *queued* transformations to be applied
  - Return a list or value to the driver (serial) process
  - Input: RDD → OPERATION → Output: NOT a RDD type (e.g., integer)

Transformations I

- Transformations (lazily evaluated—no immediate computation)



Transformations I (Cont.)

From Book in Chap 2

- Transformations (lazily evaluated—no immediate computation)

Function Name	Purpose	Example	Result
map	Apply a function to each element in the RDD and return an RDD of the result	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
flatMap	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
filter	Return an RDD consisting of only elements which pass the condition passed to filter	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
distinct	Remove duplicates	`rdd.distinct()`	`{1, 2, 3}`
sample(withReplacement, fraction, [seed])	Sample an RDD	`rdd.sample(false, 0.5)`	non-deterministic

Use *map* Operation on Numbers

In [1]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()

numbers = sc.parallelize([1, 2, 3, 3])
squared = numbers.map(lambda x: x * x)
```

Use *flatmap* Operation on Numbers

In [1]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()

lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
```

Work on Text with *filter*

In [2]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

def hasWhen(line):
    return ”when" in line

whenLines = lines.filter(hasWhen)
```

Work on Text with *filter (Cont.)*

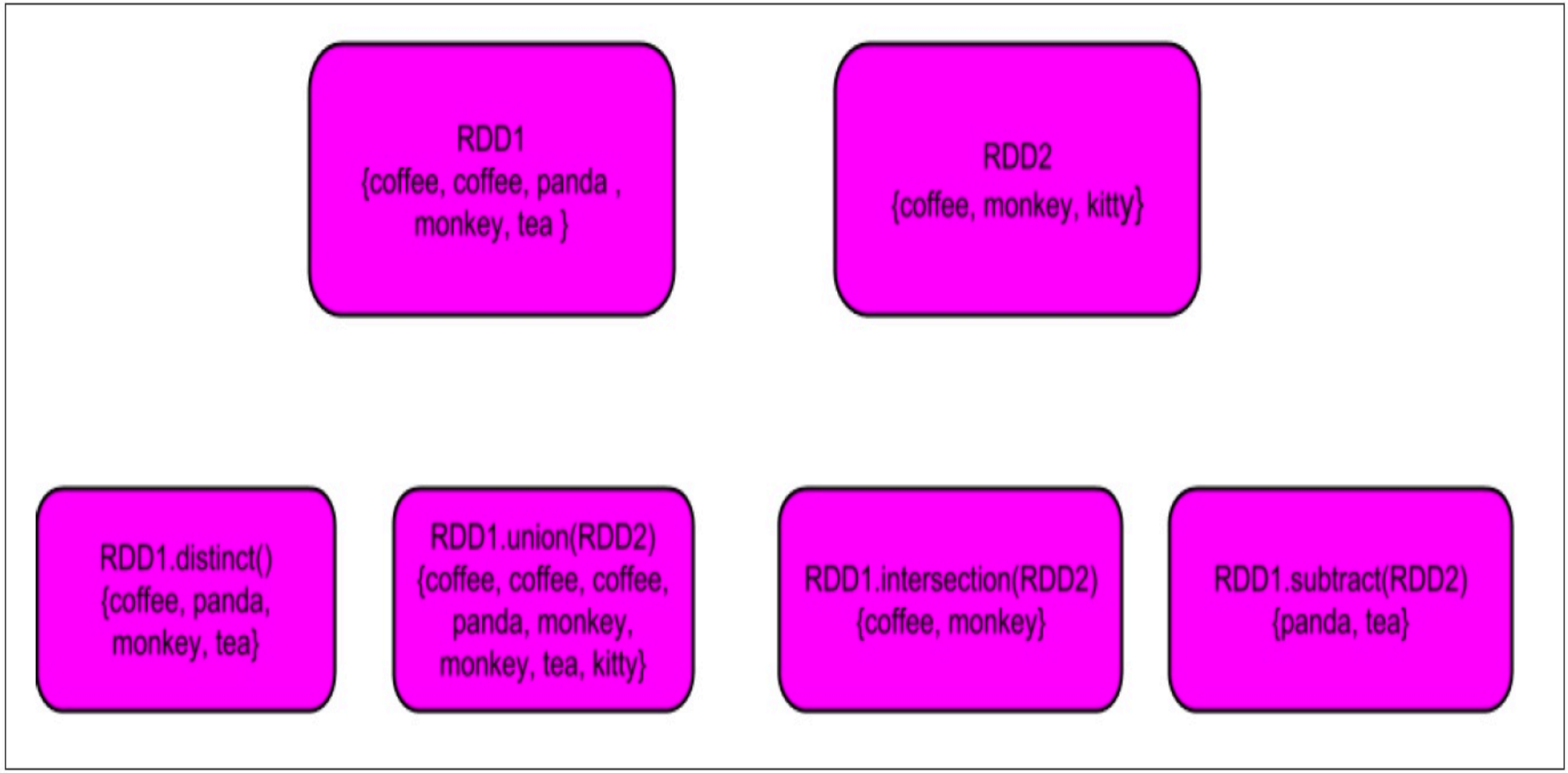
In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

whenLines = lines.filter(lambda line: ”when" in line)
```

Transformations II (Cont.)

From Book in Chap 2

- Transformations (lazily evaluated—no immediate computation)



Transformations II (Cont.)

From Book in Chap 2

- Transformations (lazily evaluated—no immediate computation)

RDDs for the examples in the table:
rdd = {1, 2, 3}
other = {3, 4, 5}

Function Name	Purpose	Example	Result
union	Produce an RDD contain elements from both RDDs	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
intersection	RDD containing only elements found in both RDDs	`rdd.intersection(other)`	`{3}`
subtract	Remove the contents of one RDD (e.g. remove training data)	`rdd.subtract(other)`	`{1, 2}`
cartesian	Cartesian product with the other RDD	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

Actions

From Book in Chap 2

RDD for the examples in the table:
rdd = {1, 2, 3, 3}

Function Name	Purpose	Example (In Scala)	Result
collect()	Return all elements from the RDD	`rdd.collect()`	{1, 2, 3, 3}
count()	Number of elements in the RDD	`rdd.count()`	4
take(num)	Return num elements from the RDD	`rdd.take(2)`	{1, 2}
top(num)	Return the top num elements the RDD	`rdd.top(2)`	{3, 3}
takeOrdered(num)(ordering)	Return num elements based on providing ordering	`rdd.takeOrdered(2)(myOrdering)`	{3, 3}

Function Name	Purpose	Example (In Scala)	Result
takeSample(withReplacement, num, [seed])	Return num elements at random	`rdd.takeSample(false, 1)`	non-deterministic
reduce(func)	Combine the elements of the RDD together in parallel (e.g. sum)	`rdd.reduce((x, y) => x + y)`	9
fold(zero)(func)	Same as reduce but with the provided zero value	`rdd.fold(0)((x, y) => x + y)`	9
aggregate(zeroValue)(seqOp, combOp)	Similar to reduce but used to return a different type	`rdd.aggregate(0, 0)({case (x, y) => (y._1() + x, y._2() + 1)}, {case (x, y) => (y._1() + x._1(), y._2() + x._2())})`	(9, 4)
foreach(func)	Apply the provided function to each element of the RDD	`rdd.foreach(func)`	nothing

From Book in Chap 2

Use *map* and *collect* Operations on Numbers

In [1]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()

numbers = sc.parallelize([1, 2, 3, 4])
squared = numbers.map(lambda x: x * x).collect()
for num in squared:
    print "%i " % (num)
```

Use *flatMap* and *first* Operations on Text

In [1]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()

lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() # returns "hello"
```

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

pairs= lines.map(lambda x: (x.split(" ")[0], x))
```

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

pairs= lines.map(lambda x: (x.split(" ")[0], x))
```

<When, When tweetle beetles fight,>
<it’s, it’s called a tweetle beetle battle.>
<And, And when they battle in a puddle,>
….

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

pairs= lines.map(lambda x: (x.split(" ")[0], x))
results = pairs.filter(lambda x: len(x[1]) < 28)
```

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

pairs= lines.map(lambda x: (x.split(" ")[0], x))
results = pairs.filter(lambda x: len(x[1]) < 28)
```

<When, When tweetle beetles fight,>

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

words = rdd.flatMap(lambda x: x.split(" "))
pairs= words.map(lambda x: (x, 1))
```

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

words = rdd.flatMap(lambda x: x.split(" "))
pairs= words.map(lambda x: (x, 1))
```

<“When”, 1> <“tweetle”, 1> <“beetles”,1> <“fight,”, 1>
…

Create Key-Values in RDDs

In [3]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
lines = sc.textFile(”FoxInSocks.txt")

words = rdd.flatMap(lambda x: x.split(" "))
pairs= words.map(lambda x: (x, 1))
results = pairs.reduceByKey(lambda x, y: x + y)
```

The Special Case of *ReduceByKey*

- Reduce takes a function and use it to combine values
- ReduceByKey takes a function and use it to combine values
- BUT ReduceByKey **DO NOT** implemented as an action
  - Return a new RDD consisting of each key and the reduced value for that key

WHY?

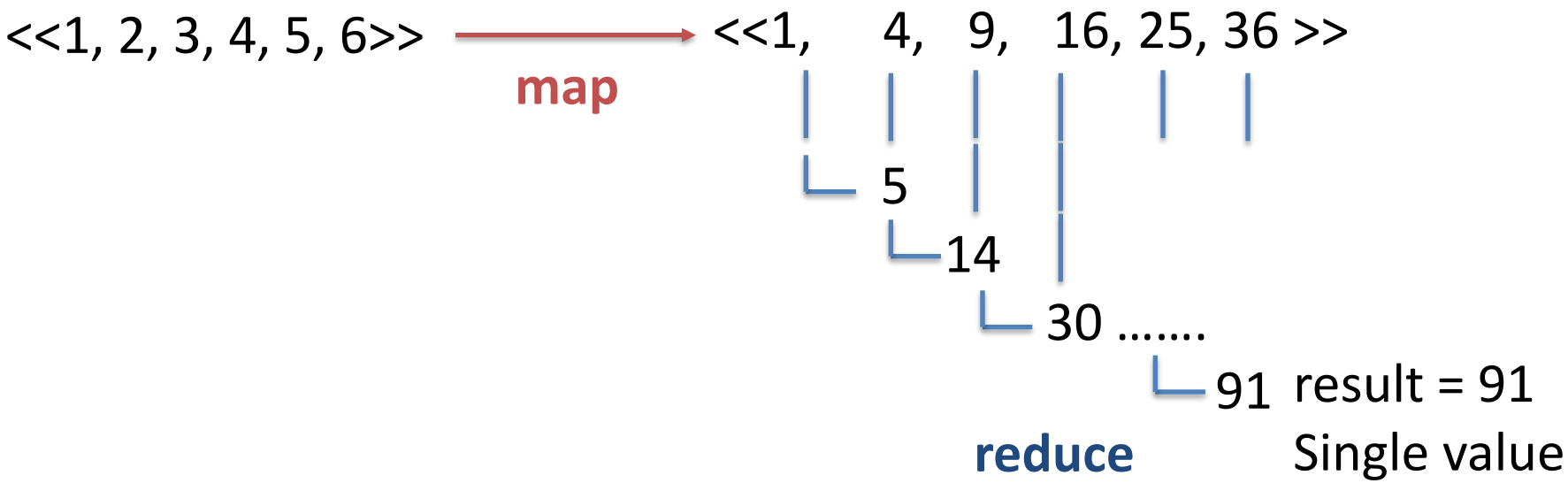
The Special Case of *ReduceByKey*

- Reduce takes a function and use it to combine values
- ReduceByKey takes a function and use it to combine values
- BUT ReduceByKey **DO NOT** implemented as an action
  - Return a new RDD consisting of each key and the reduced value for that key

WHY?

- *reduceByKey* runs several parallel reduce operations:
  - one for each key in the dataset
  - each operation combines values together which have the same key.
- Datasets can have very large numbers of keys!!!!

Use *reduce* Operation on Numbers I

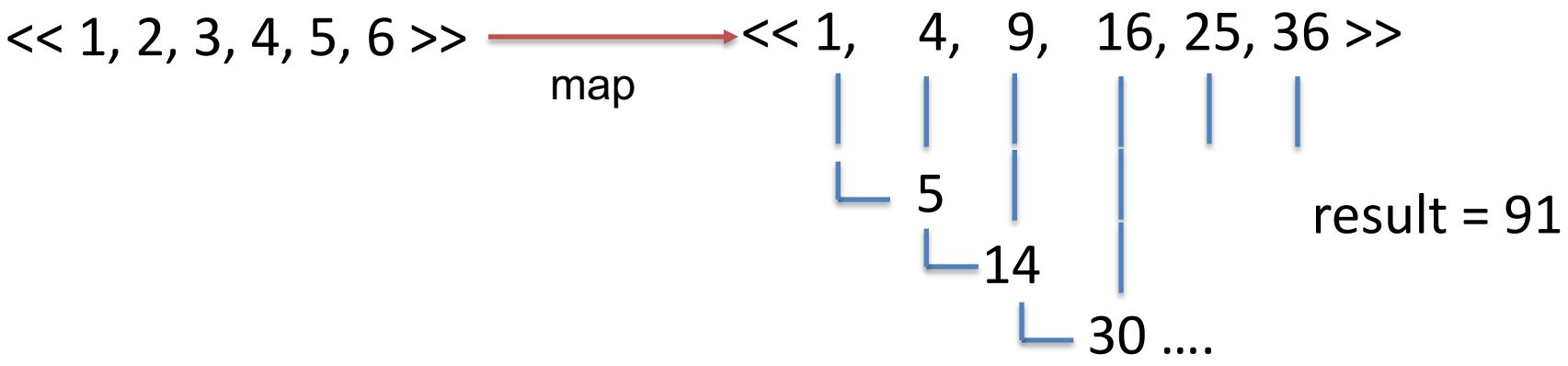


Note: Spark does NOT guarantee the order of operands

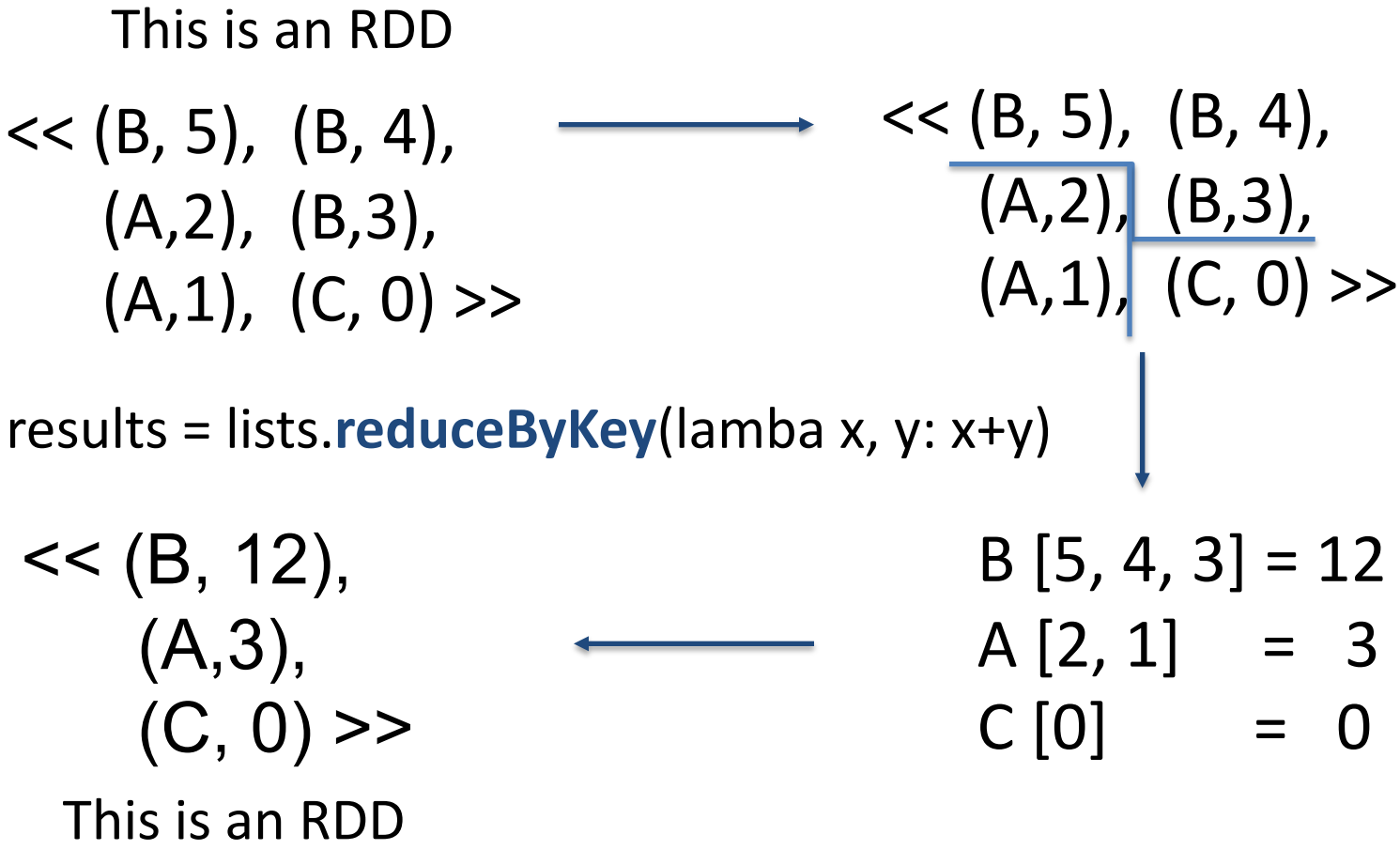
Use *reduce* Operation on Numbers I

In [1]

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()


numbers = sc.parallelize([1, 2, 3, 4, 5, 6])
squared = numbers.map(lambda x: x * x)
result = squared.reduce(lamba x, y: x+y) # single value
```



Use *reduce* Operation on Numbers I



WordCount Benchmark

In [4]

```
# Initializing Spark in Python
from pyspark import SparkContext, SparkConf
conf = SparkConf().setMaster("local").setAppName("WordCount")
sc = SparkContext(conf=conf)
```

WordCount Benchmark

In [4]

```
# Initializing Spark in Python
from pyspark import SparkContext, SparkConf
conf = SparkConf().setMaster("local").setAppName("WordCount")
sc = SparkContext(conf=conf)
```

cluster URL

application name

WordCount Benchmark

In [4]

```
# Initializing Spark in Python
from pyspark import SparkContext, SparkConf
conf = SparkConf().setMaster("local").setAppName("WordCount")
sc = SparkContext(conf=conf)


lines  = sc.textFile("FoxInSocks.txt")
words  = lines.flatMap(lambda line: line.split())
pairs  = words.map(lambda word: (word, 1))
counts = pairs.reduceByKey(lambda a, b: a+b) # counts is an RDD!
```

cluster URL

application name

WordCount Benchmark

In [4]

```
# Initializing Spark in Python
from pyspark import SparkContext, SparkConf
conf = SparkConf().setMaster("local").setAppName("WordCount")
sc = SparkContext(conf=conf)


lines  = sc.textFile("FoxInSocks.txt")
words  = lines.flatMap(lambda line: line.split())
pairs  = words.map(lambda word: (word, 1))
counts = pairs.reduceByKey(lambda a, b: a+b) # counts is an RDD!

results = counts.collect()
```

cluster URL

application name

WordCount Benchmark

In [4]

```
# Initializing Spark in Python
from pyspark import SparkContext, SparkConf
conf = SparkConf().setMaster("local").setAppName("WordCount")
sc = SparkContext(conf=conf)


lines  = sc.textFile("FoxInSocks.txt")
words  = lines.flatMap(lambda line: line.split())
pairs  = words.map(lambda word: (word, 1))
counts = pairs.reduceByKey(lambda a, b: a+b) # counts is an RDD!

results = counts.collect()
```

cluster URL

application name

Invoke the **action collect()** which brings all the elements of the RDD counts to the driver.
The action causes all the queued up transformations to be applied.

Assignment 4

Assignment 4

- Python has map and reduce functions:
  - Do not take advantage of parallel processing (i.e., they are sequential)
- Define three methods:
  (i.e., mapSequential, reduceSequential,
  and reduceByKeySequential)
- Extend Python's map and reduce functions to act like those in *Apache Spark*

Deadline: ***February 7 - 8AM ET***

Reading

Reading

- **(LOW) Apache Spark: A Unified Engine for Big Data Processing.** Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. CACM, 2016.
- ***(MEDIUM) Spark: Cluster Computing with Working Sets.*** Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. University of California, Berkeley.
- **(HIGH) Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.** Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. University of California, Berkeley.

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

BIG ORANGE. BIG IDEAS.®

BIG ORANGE BIG IDEAS