

# Lecture 5: Spark (continue)

COSC 526: Introduction to Data Mining



THE UNIVERSITY OF  
TENNESSEE  
KNOXVILLE  
KNOXVILLE

**BIG ORANGE. BIG IDEAS.®**

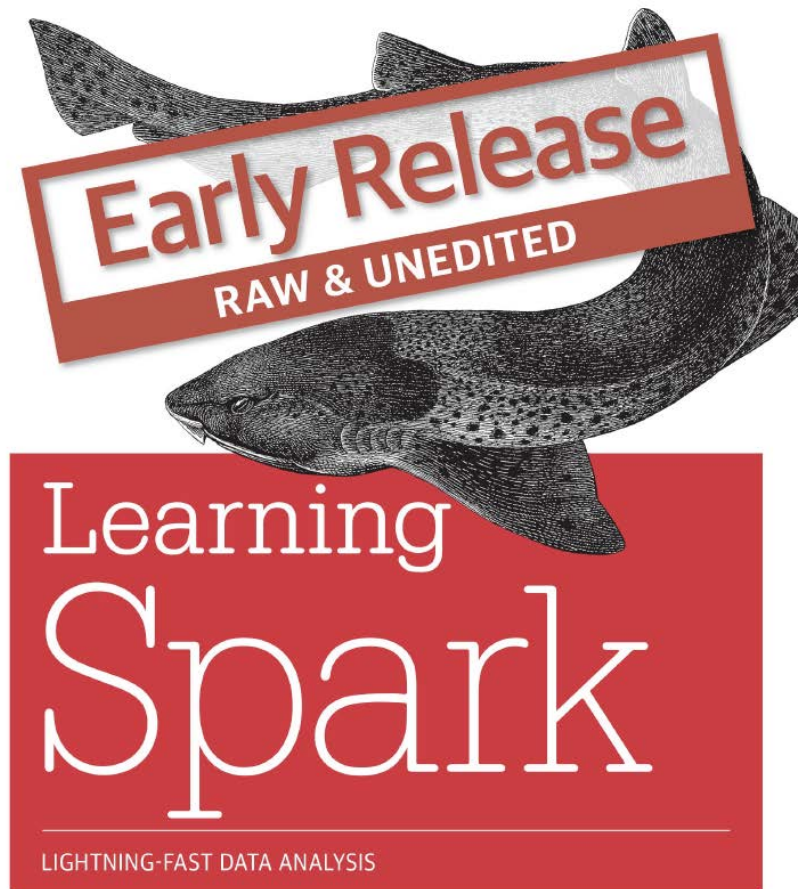
# Today Outline

- Log into your Jetstream account
- Complete discussion on Spark operations
- Zoom into WorkCount on Spark
  - What storage resources do we use and when
- Project
  - Two examples of projects: posters and 2-page abstract
- Assignment 5
- If time left, live chat with video

# Spark Operations

# Spark reference

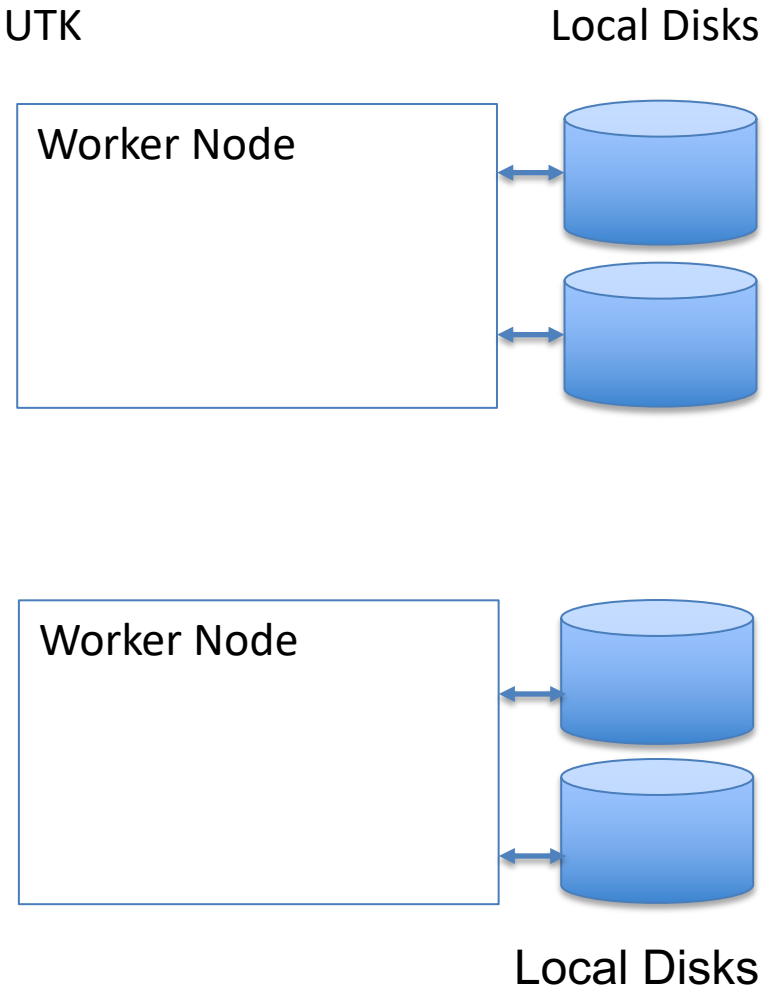
O'REILLY®



Holden Karau,  
Andy Kowinski & Matei Zaharia

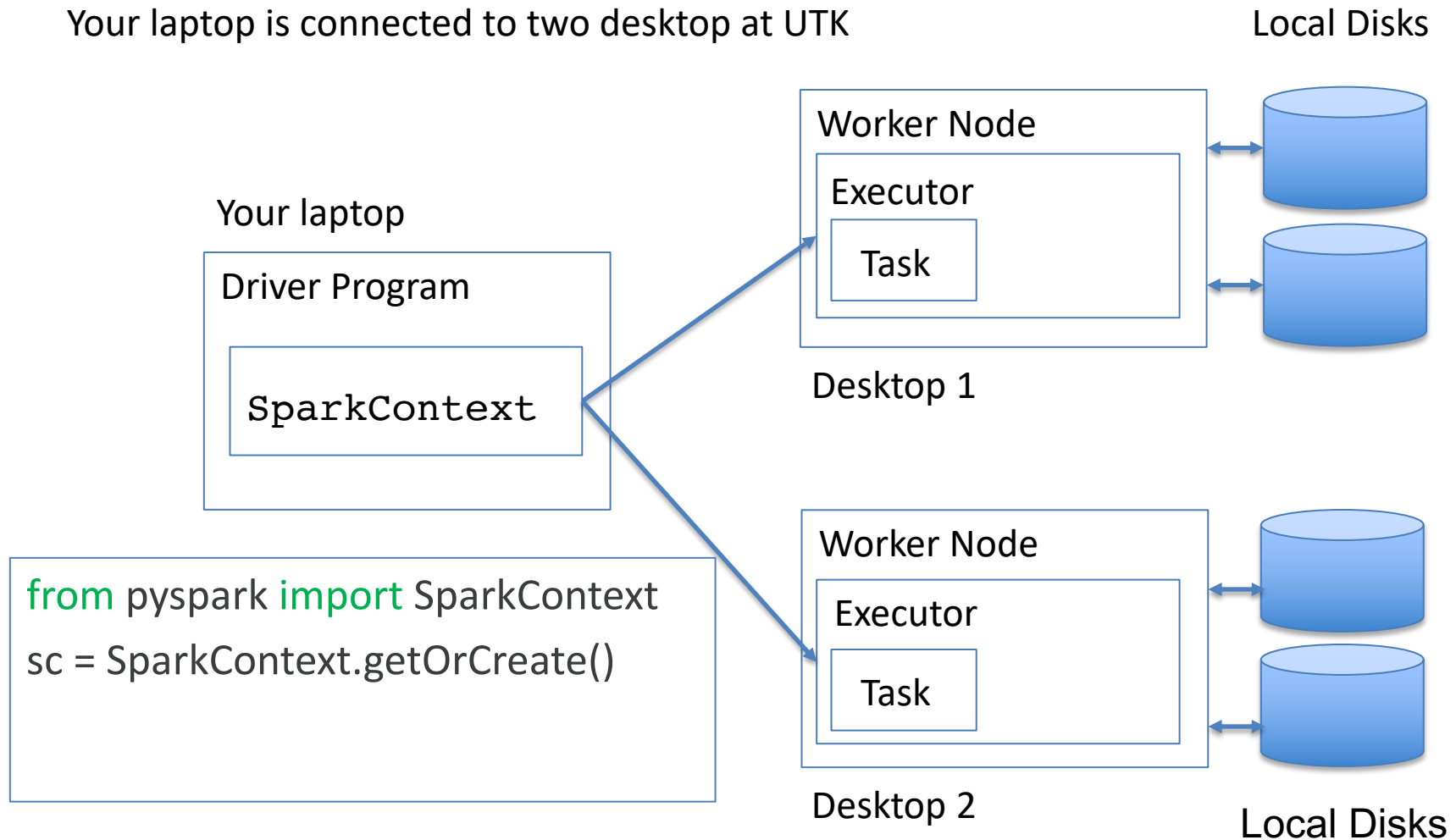
# Hypothetical Scenario

Your laptop is connected to two desktop at UTK



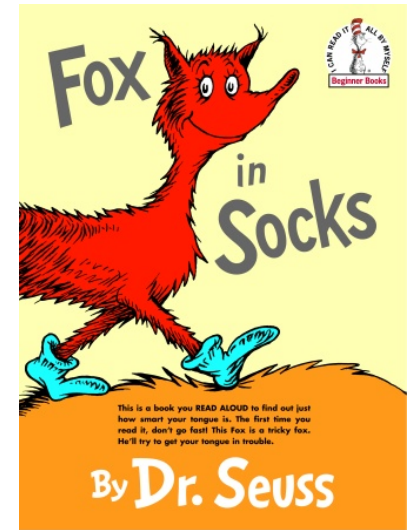
# Hypothetical Scenario

Your laptop is connected to two desktop at UTK



# 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.*



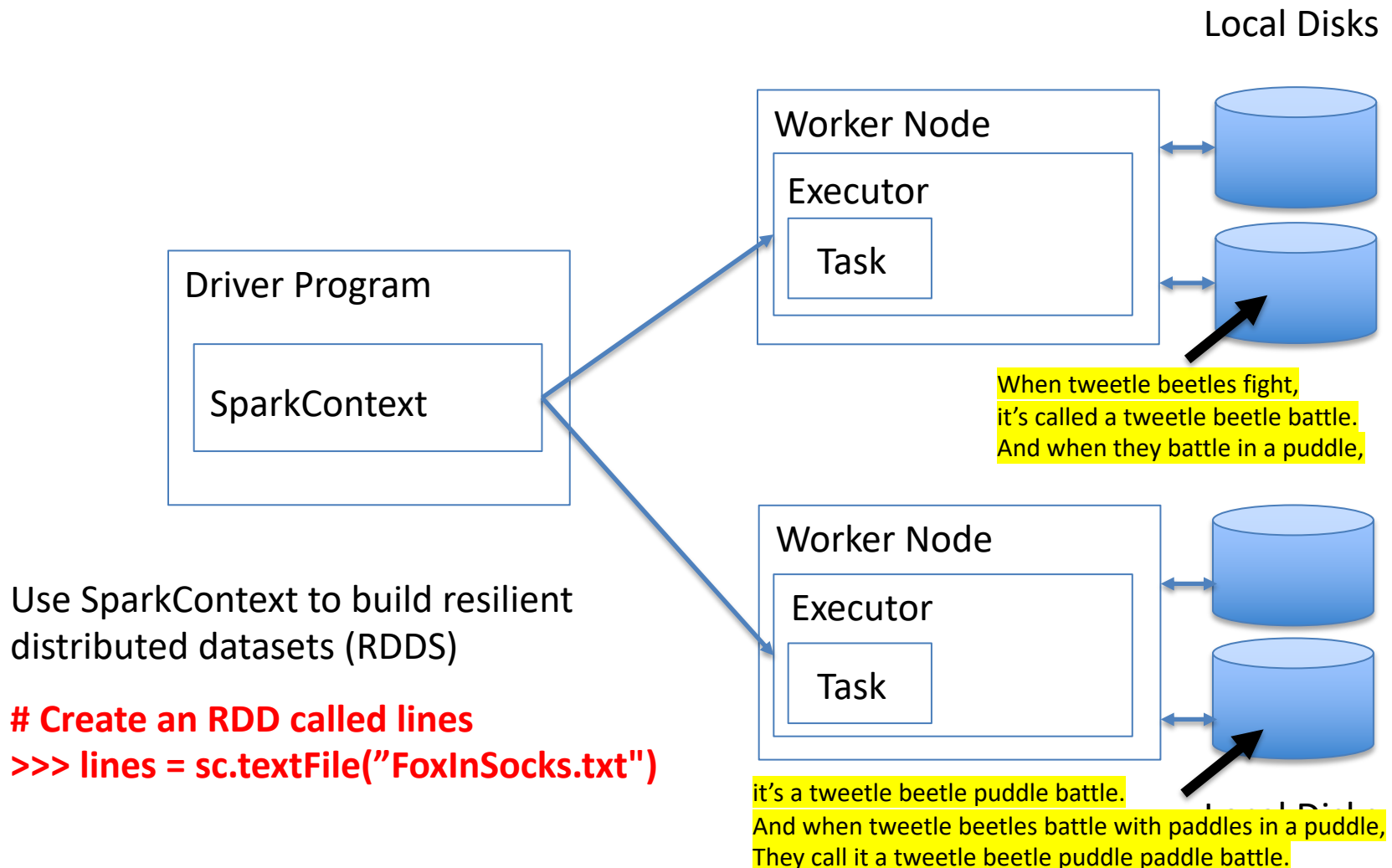
# Create an RDD called lines

```
>>> 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





# 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 (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	<code>rdd.map(x =&gt; x + 1)</code>	<code>{2, 3, 4, 4}</code>
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.	<code>rdd.flatMap(x =&gt; x.to(3))</code>	<code>{1, 2, 3, 2, 3, 3, 3}</code>
filter	Return an RDD consisting of only elements which pass the condition passed to filter	<code>rdd.filter(x =&gt; x != 1)</code>	<code>{2, 3, 3}</code>
distinct	Remove duplicates	<code>rdd.distinct()</code>	<code>{1, 2, 3}</code>
<code>sample(withReplacement, fraction, [seed])</code>	Sample an RDD	<code>rdd.sample(false, 0.5)</code>	non-deterministic

# 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
<code>union</code>	Produce an RDD contain elements from both RDDs	<code>rdd.union(other)</code>	<code>{1, 2, 3, 3, 4, 5}</code>
<code>intersection</code>	RDD containing only elements found in both RDDs	<code>rdd.intersection(other)</code>	<code>{3}</code>
<code>subtract</code>	Remove the contents of one RDD (e.g. remove training data)	<code>rdd.subtract(other)</code>	<code>{1, 2}</code>
<code>cartesian</code>	Cartesian product with the other RDD	<code>rdd.cartesian(other)</code>	<code>{(1, 3), (1, 4), ... (3,5)}</code>

# Actions

From Book in Chap 2


RDD for the examples in the table:

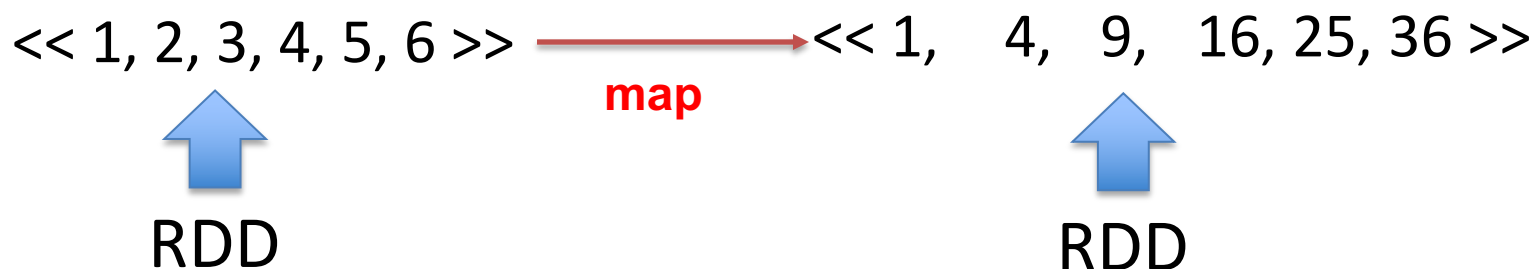
`rdd = {1, 2, 3, 3}`

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

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

# Use *reduce* Operation on Numbers

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)`

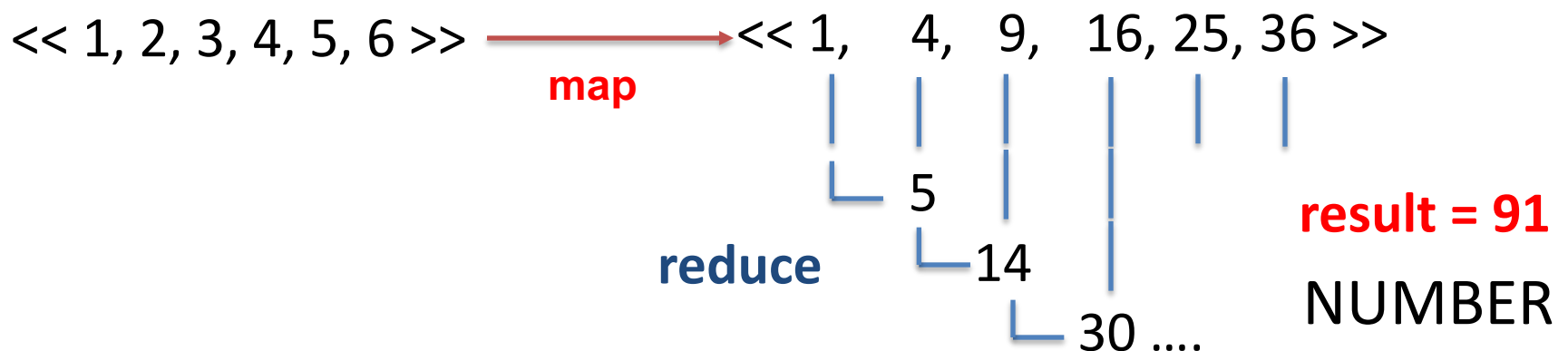


# Use *reduce* Operation on Numbers

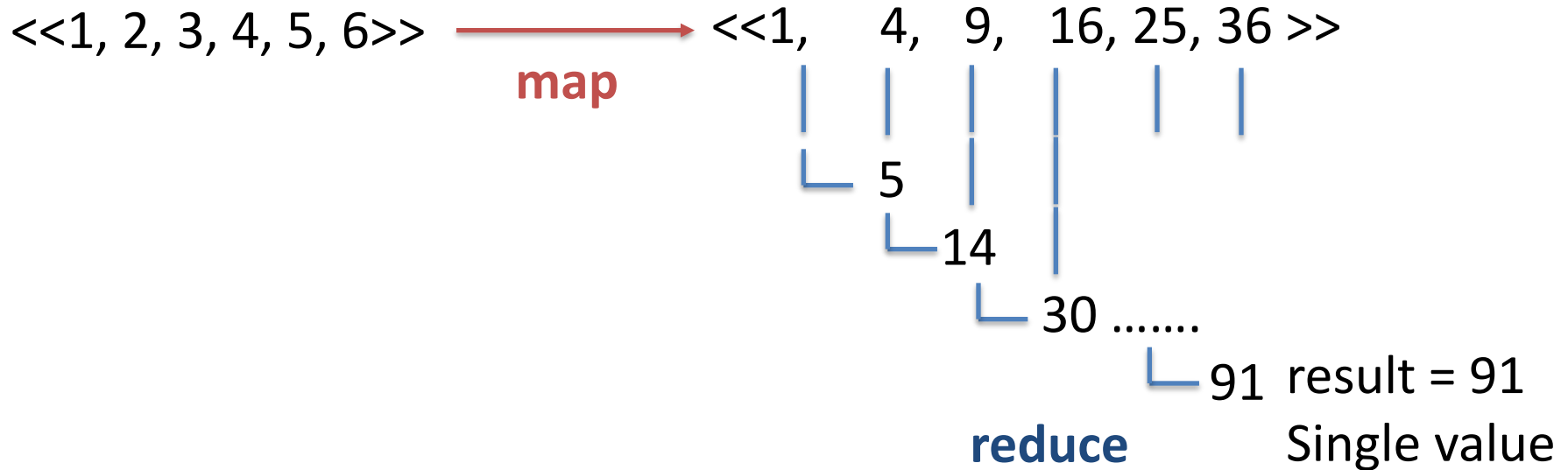
In [1]

```
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(lambda x, y: x+y) # single value
```



# Use *reduce* Operation on Numbers (I)



Note: Spark does **NOT** guarantee the order of operands



# Use *reduce* Operation on Numbers (II)

Node 1

<<1, 2, 3, >>



map

<<1, 4, 9 >>

Node 2

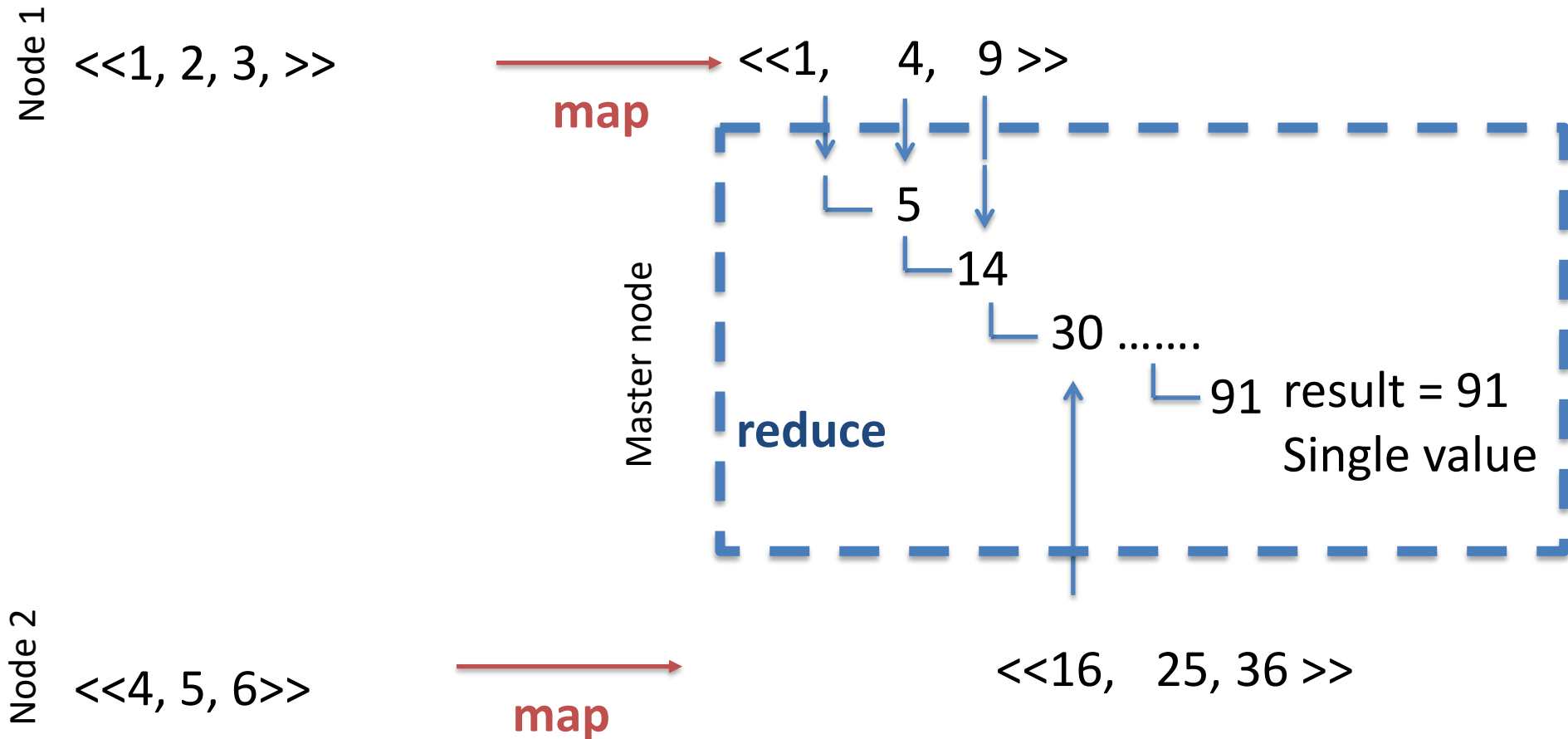
<<4, 5, 6>>



map

<<16, 25, 36 >>

# Use *reduce* Operation on Numbers (II)



# Use *reduce* Operation on Numbers (III)

<<1, 2, 3, >>

map

<<1, 4, 9 >>

Master node

16

41

42

.....

91

result = 91

Single value

reduce

<<4, 5, 6>>

map

<<16, 25, 36 >>

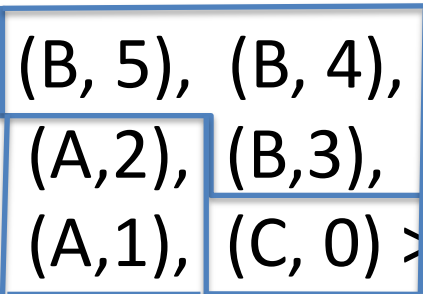
# Use *reduceByKey* Operation on Numbers

This is an RDD

<< (B, 5), (B, 4),  
      (A, 2), (B, 3),  
      (A, 1), (C, 0) >>

→

<< (B, 5), (B, 4),  
      (A, 2), (B, 3),  
      (A, 1), (C, 0) >>



results = lists.**reduceByKey**(lambda x, y: x+y)

<< (B, 12),  
      (A, 3),  
      (C, 0) >>

←

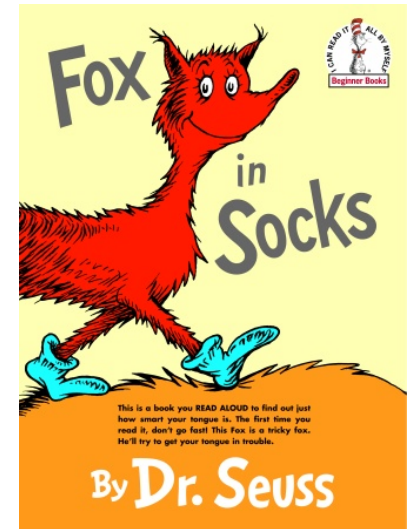
B [5, 4, 3] = 12  
A [2, 1] = 3  
C [0] = 0

This is an **RDD**

**The data is still on the remote nodes**

# 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.

# Create Key-Values in RDDs

In [3]

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

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.>

# 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))
```

Node 1 <When, When tweetle beetles fight,>  
<it's, it's called a tweetle beetle battle. >  
<And, And when they battle in a puddle, >

Node 2 <it's, it's a tweetle beetle puddle battle. >  
<And, And when tweetle beetles battle with paddles in a puddle,>  
<They, They call it a tweetle beetle puddle paddle battle.>

# 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)
```

Node 1 <When, When tweetle beetles fight,>

Node 2



# 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))
```

Node 1 <"When",1> <"tweetle", 1> <"beetles",1> <"fight",1>  
<"it's",1> <"called",1> ....

Node 2 <"it's", 1> <"a",1> <"tweetle",1> <"beetle",1> <"puddle",1>  
<"battle",1><"And", 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)
```

Node 1 <"When",2> <"tweetle", 2> <"beetles",1> <"fight",1>  
<"it's",1> <"called",1> ....

Node 2 <"it's", 1> <"a",3> <"tweetle",3> <"beetle",2> <"puddle",2>  
<"battle",2><"And", 1> ...

# 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!!!!**

# 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)
```

**Your machine**

**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)
```

**Your machine**      **application name**

```
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!
```

**THESE (line, workds, pairs, counts) ARE ALL RDD**

# 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()
```

**Your machine**      **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)
```

**Your machine**      **application name**

```
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()
```

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.

Practical Problems

—

The MapReduce Sequential Implementation

# Assignment 5

- 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 19 - 8AM ET***



THE UNIVERSITY OF  
TENNESSEE  
KNOXVILLE

**BIG ORANGE. BIG IDEAS.®**

