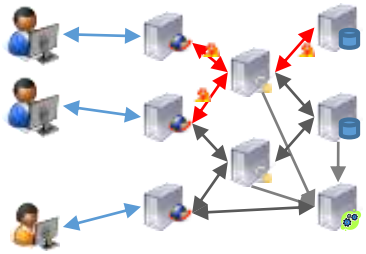
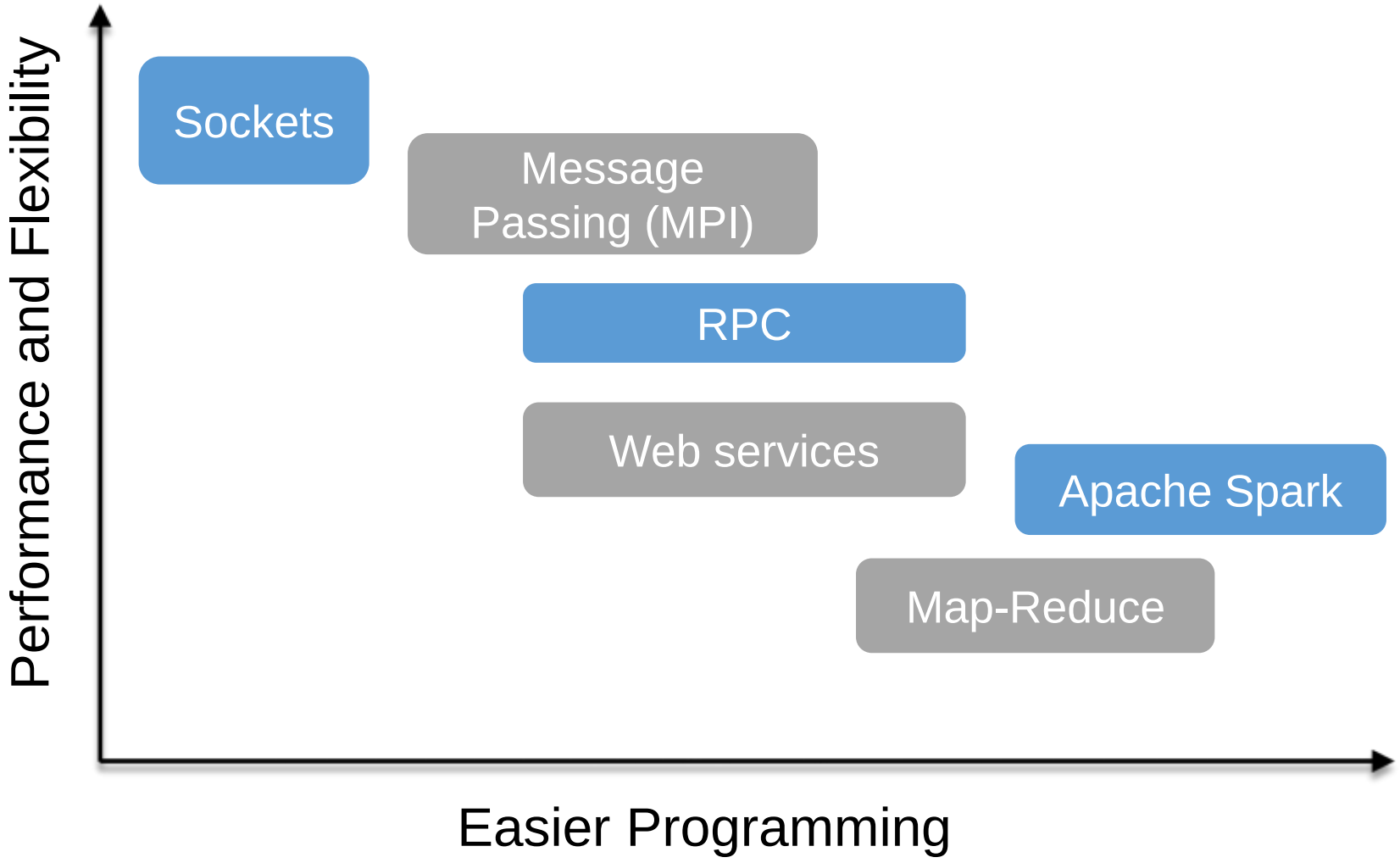


Verteilte Systeme/ Distributed Systems

Artur Andrzejak

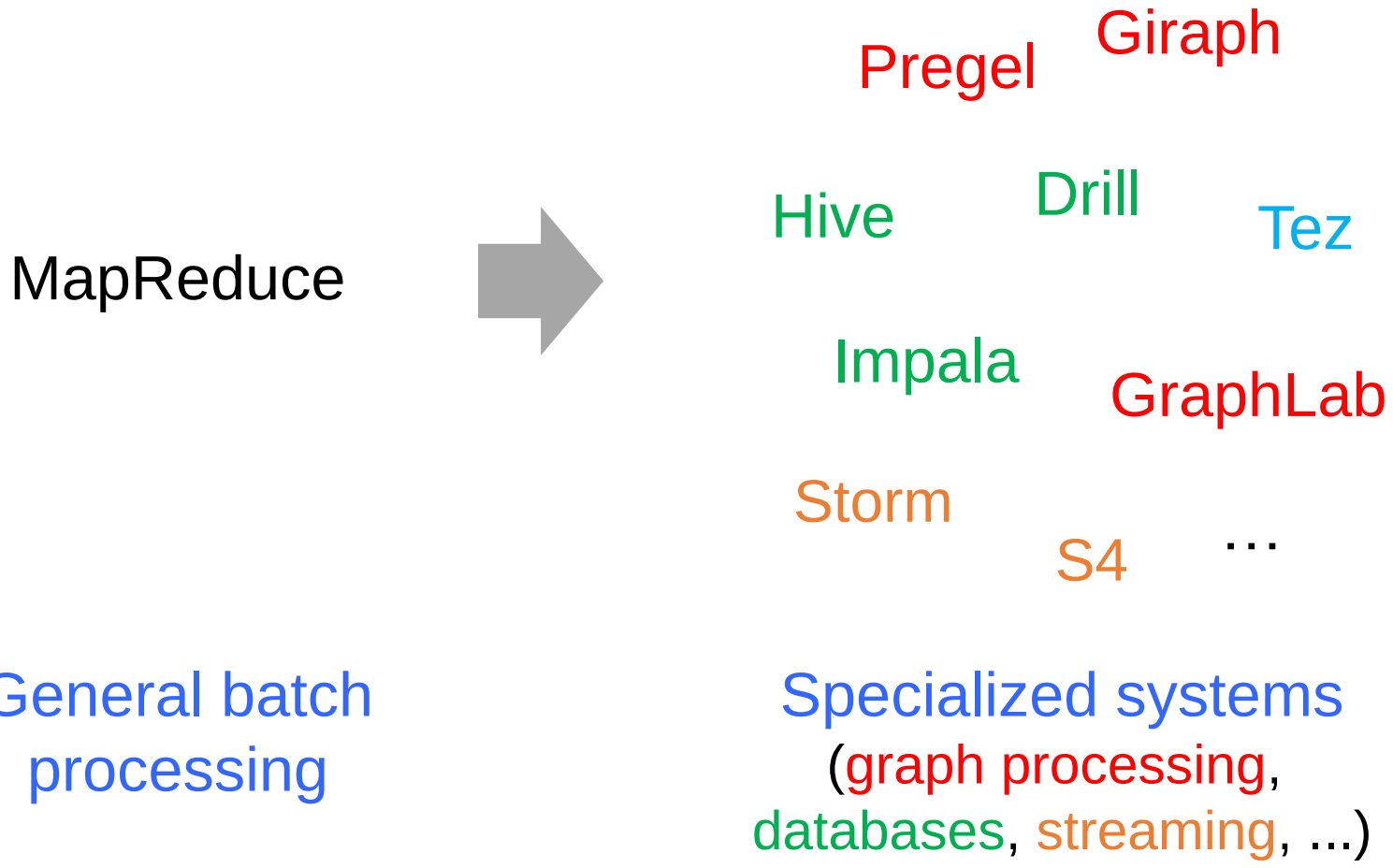
4

Trade-offs Distributed Programming



Apache Spark: Basics

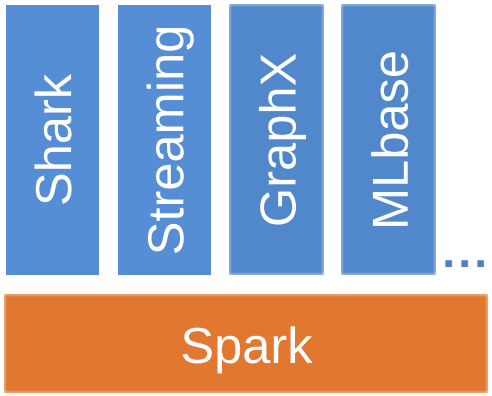
MapReduce Problems



From: The State of Spark, and Where We're Going Next
Matei Zaharia, Spark Summit (2013)

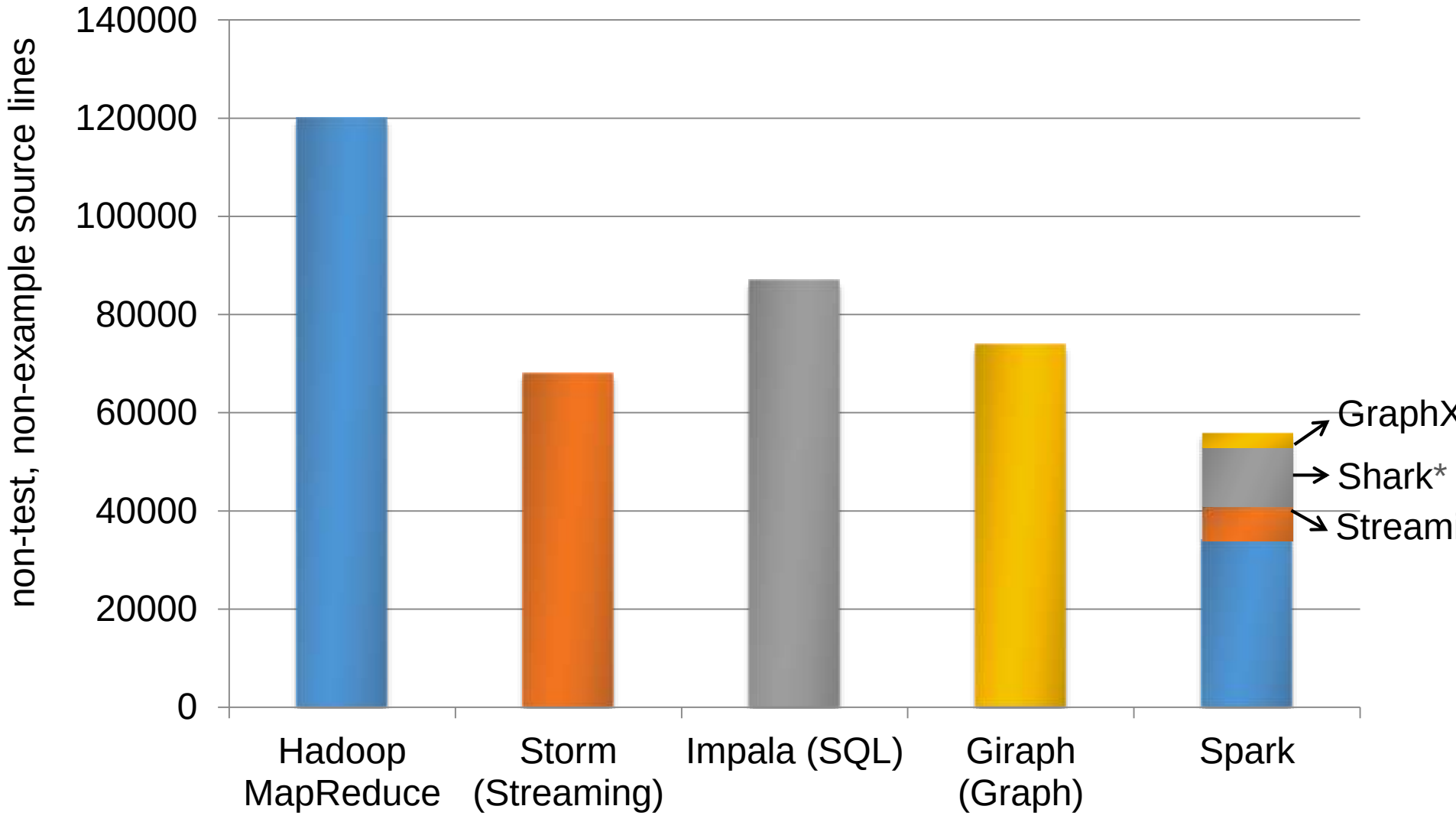
Spark's Approach

- Instead of specializing, *generalize* MapReduce to support new apps in same engine
- Two changes (general task DAG & data sharing) are enough to express previous models!
- Unification has big benefits
  - For the engine
  - For users



From: The State of Spark, and Where We're Going Next
Matei Zaharia, Spark Summit (2013)

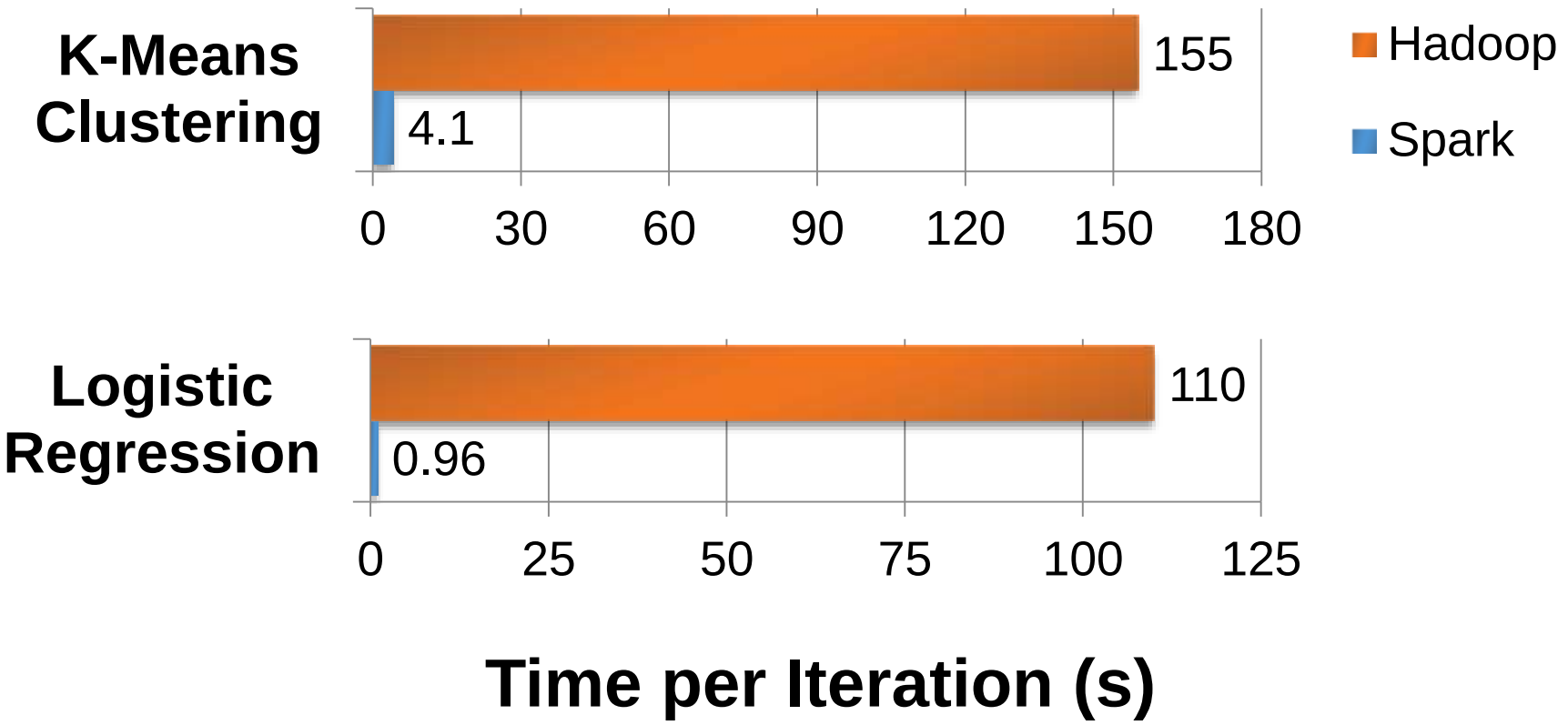
Code Size: Spark vs. Others



From: The State of Spark, and Where We're Going Next
Matei Zaharia, Spark Summit (2013)

* also calls into Hive

Performance: Iterative Algorithms

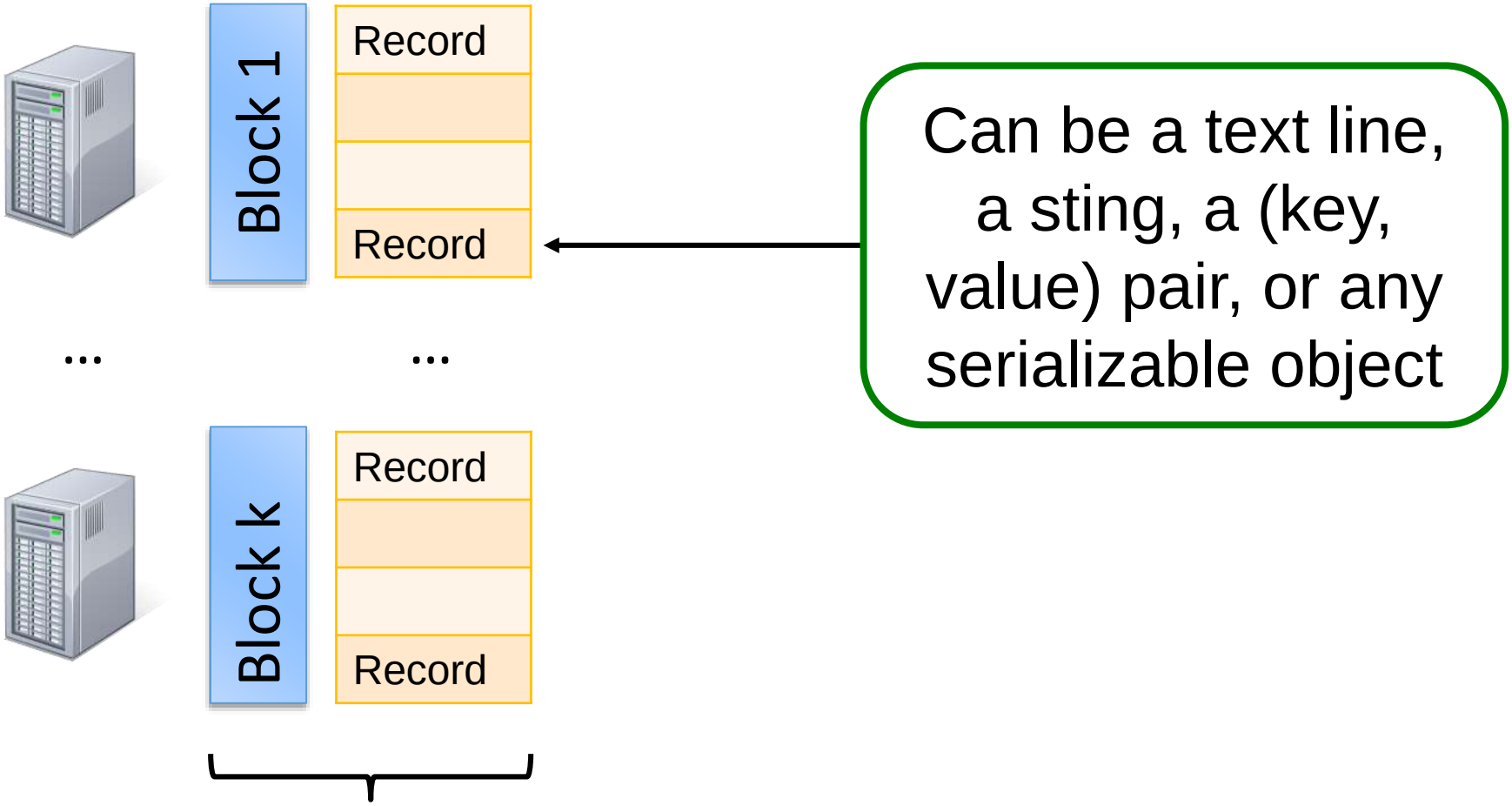


From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

Spark: Key Ideas and Data Structure

- **Write programs in terms of transformations on distributed datasets**
- Main data structure: **resilient distributed dataset** (**RDD**)
  - Collections of records spread across a cluster

Resilient Distributed Dataset (RDD)



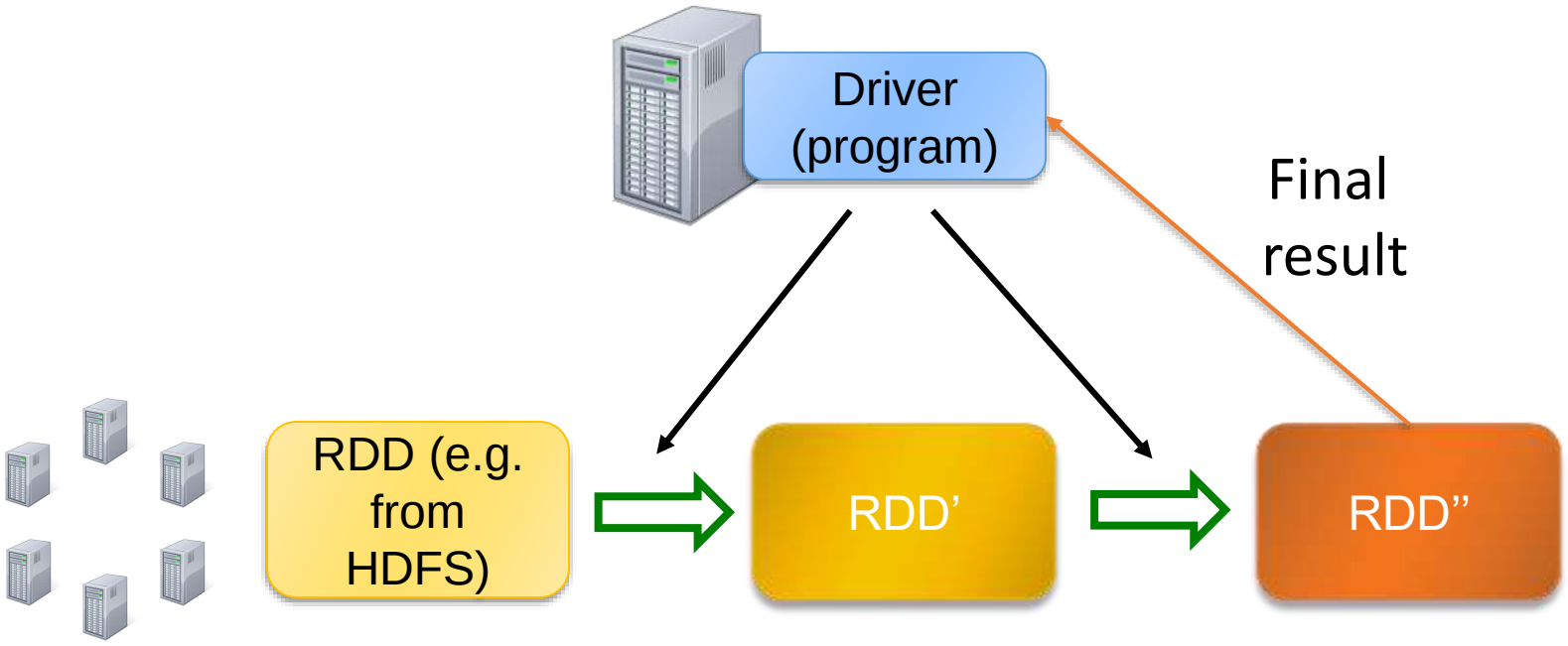
A RDD; accessible in code via a single “variable”

Operations

- **Transformations** (e.g. map, filter, groupBy)
  - Lazy operations to build RDDs from other RDDs
- **Actions** (e.g. count, collect, save)
  - Return a result or write it to storage

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

Computations with RDDs

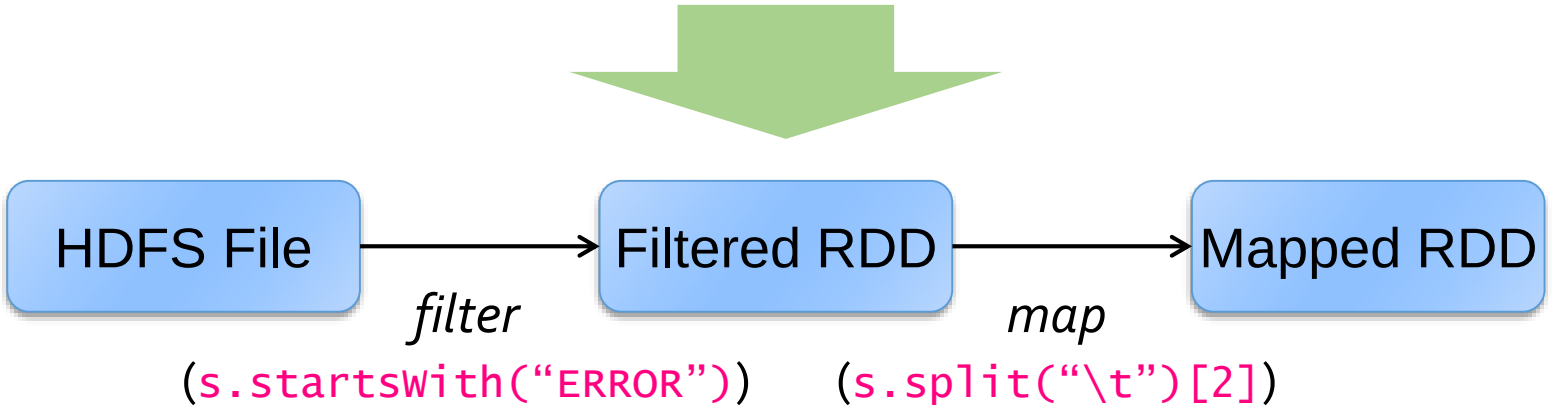


Fault Recovery

RDDs track *lineage* information that can be used to efficiently recompute lost data

Ex:

```
msgs = textFile.filter(lambda s: s.startsWith("ERROR"))
               .map(lambda s: s.split("\t")[2])
```



From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

Creating RDDs (Python)

SparkContext is the "entry point" to Spark functionality, here referenced by variable `sc`

```
# Turn a Python list [1, 2, 3] into an RDD
sc.parallelize([1, 2, 3])

# Load text file from local FS (file/dir) or HDFS
sc.textFile("file.txt")
sc.textFile("directory/*.txt")
sc.textFile("hdfs://namenode:9000/path/file")
```

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

Interlude: Lambdas in Python

- We need to pass code to Ts and As
  - Handle code like variables
- Convenient: **anonymous functions**, or inline functions; in Python: lambda functions
- Syntax:

lambda <params> **:** expression

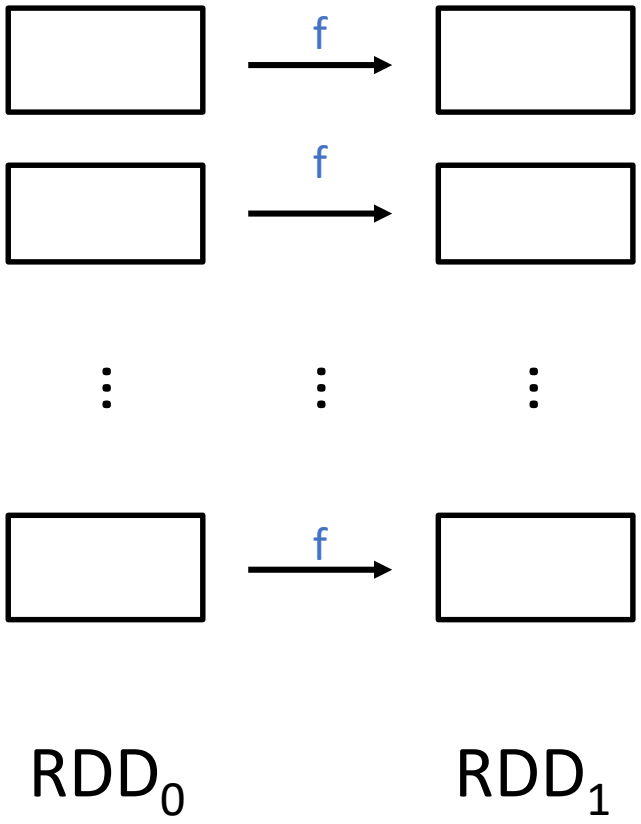
- Example – computing x^2
  - g = **lambda** x**:** x**2; same as def g(x): return x**2
- What is this doing?
  - **lambda** x, y**:** x**2 + y**2 <= 1.0

Basic Transformations

```
nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
even = squares.filter(lambda x: x % 2 == 0) //
{4}

# Map each element to zero or more others
flats = nums.flatMap(lambda x: [x, -x, x*x])
    # => {1, -1, 1, 2, -2, 4, 3, -3, 9}
```

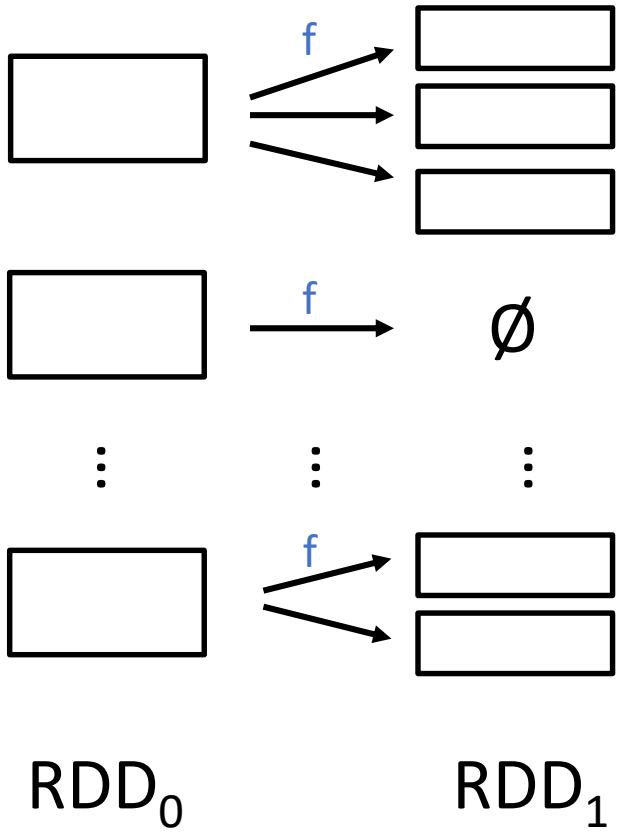

Essential Transformations

map with f:val -> val



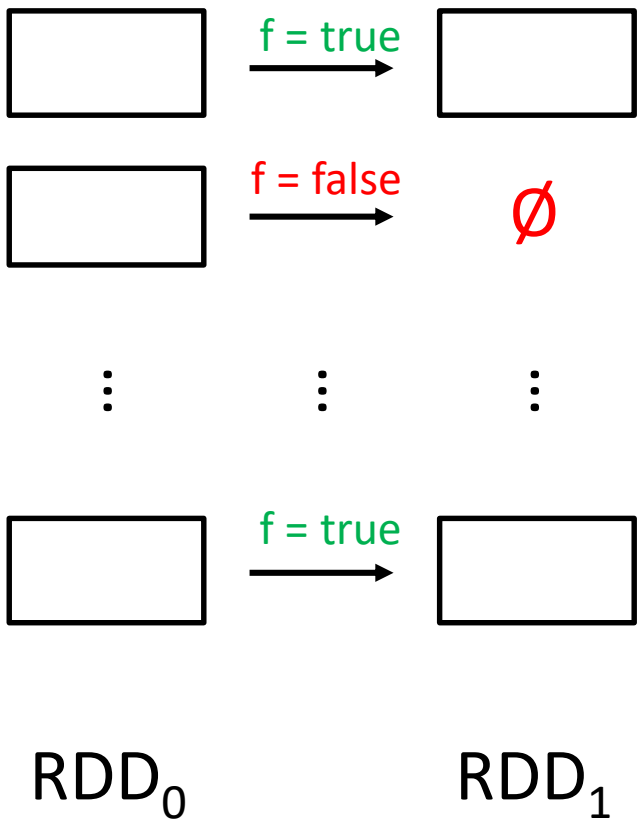
flatMap with

f:val -> [val,…,val]



Essential Transformations

filter with f:val -> boolean



Others (useful):

- union
- distinct
- join
- groupByKey …

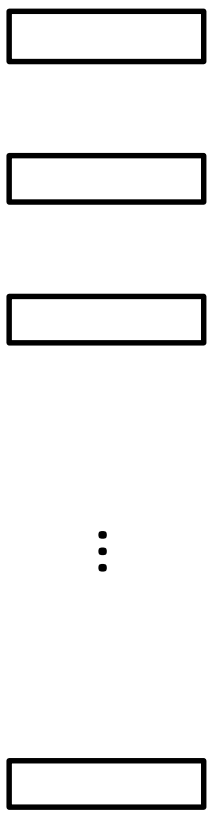
Basic Actions

```
nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
nums.collect() # => [1, 2, 3]

# Return first K elements
nums.take(2)   # => [1, 2]

# Count number of elements
nums.count()   # => 3

# Merge elements with an associative function
nums.reduce(lambda x, y: x + y)  # => 6

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

„Reduce“-Action

reduce with f: val, val -> val



Working with Key-Value Pairs

- Spark's "distributed reduce" transformations operate on RDDs of key-value pairs
- Python:

```
pair = (a, b)
pair[0] # => a
pair[1] # => b
```

- Scala:

```
val pair = (a, b)
pair._1 // => a
pair._2 // => b
```

- Java:

```
Tuple2 pair = new Tuple2(a, b);
pair._1 // => a
pair._2 // => b
```


Some Key-Value Operations

- ```
  pets = sc.parallelize(
    [("cat", 1), ("dog", 1), ("cat", 2)])
  ```

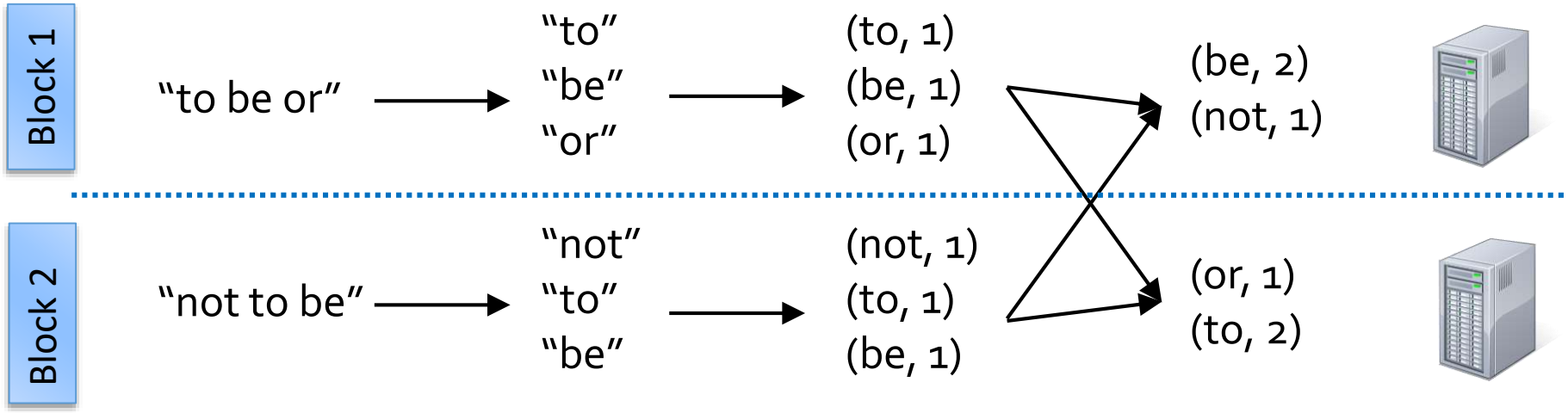
- ```
  pets.reduceByKey(lambda x, y: x + y)
                      # => {(cat, 3), (dog, 1)}
  ```

- ```
  pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
  ```
- ```
  pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog, 1)}
  ```

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

Example: Word Count

- `lines = sc.textFile("hamlet.txt")`
- ```
  counts = lines.flatMap(lambda line: line.split(" "))
                .map(lambda word : (word, 1))
                .reduceByKey(lambda x, y: x + y)
  ```



From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013
  ```

Other Key-Value Operations

```
visits = sc.parallelize([ (“index.html”, “1.2.3.4”),
                          (“about.html”, “3.4.5.6”),
                          (“index.html”, “1.3.3.1”) ])

pageNames = sc.parallelize([ (“index.html”, “Home”),
                             (“about.html”, “About”) ])

visits.join(pageNames)
# (“index.html”, (“1.2.3.4”, “Home”))
# (“index.html”, (“1.3.3.1”, “Home”))
# (“about.html”, (“3.4.5.6”, “About”))

visits.cogroup(pageNames)
# (“index.html”, ([“1.2.3.4”, “1.3.3.1”], [“Home”]))
# (“about.html”, ([“3.4.5.6”], [“About”]))
```


Setting the Level of Parallelism

- All the pair RDD operations take an optional second parameter p for number of tasks

- `words.reduceByKey(lambda x, y: x + y, 5)`
- `words.groupByKey(5)`

Using Local Variables

Essentially, piece of code executed by a transformation or action

- Any external variables you use in a closure will automatically be shipped to the cluster:
- ```
  query = sys.stdin.readline()
  pages.filter(lambda x: query in x).count()
  ```
- Some caveats:
  - Each task gets a new copy (no updates sent back)
  - Variable must be Serializable / Pickle-able
  - Don't use fields of an outer object (ships all of it!)

There are also shared variables: broadcasts, accumulators

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013


  ```

Passing Values to/from Functions

- Functions in transformations/actions are **closures**: they have read-only access to all driver variables visible at closure definition
  - Related: **broadcast** variables
- Transformations
  - NOT possible to return values to driver process
  - NO exchange of values between code executing on different records of a RDD

Other RDD Operators

- map
- filter
- groupBy
- sort
- union
- join
- leftOuterJoin
- rightOuterJoin

- reduce
- count
- fold
- reduceByKey
- groupByKey
- cogroup
- cross
- zip..

- sample
- take
- first
- partitionBy
- mapWith
- pipe
- save

More details: spark-project.org/docs/latest/

From: *Parallel Programming with Spark,* **Matei Zaharia**, AmpCamp 2013

Installing PySpark Natively

- PySpark: Spark with Python bindings: info, APIs
- Linux
  - See e.g. tutorial here
  - Or just do: pip3 install pyspark (see here)
- Windows 10: Windows Subsystem for Linux (WSL)
  - Install Spark under linux as above
  - You can even install/run PyCharm etc. under WSL, with X11 server (see previous slides for details)
- Windows “native”
  - You could try to install Spark directly (tutorial), but …
    - Tell me if this really works! ☺

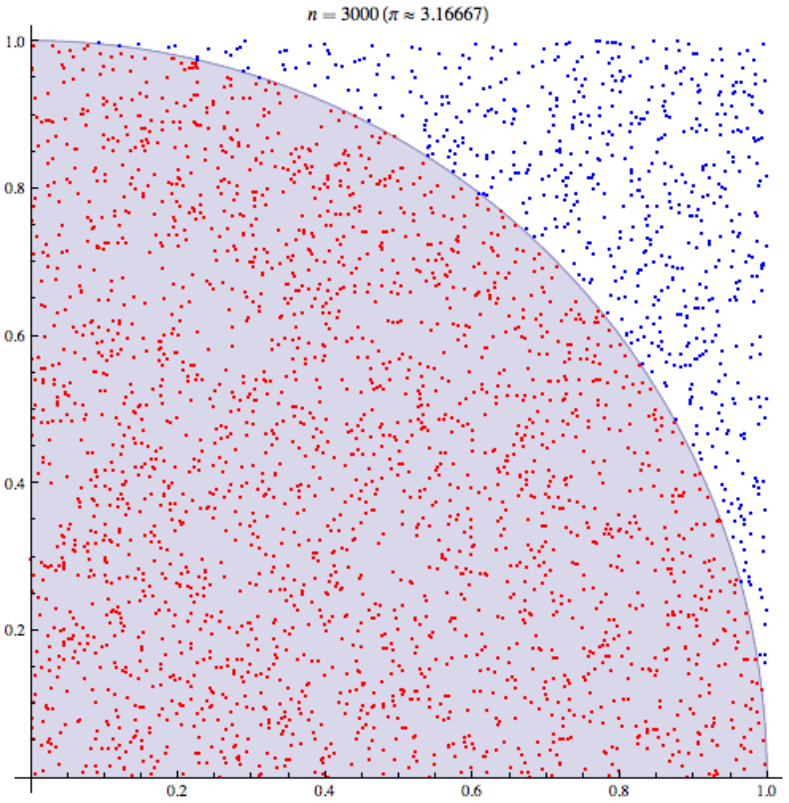
Virtual Machine with Spark

- We have prepared a virtual machine with Spark (under Oracle VirtualBox):
  - https://heibox.uni-heidelberg.de/d/2a001bb696/
- Using Spark:
  - Most convenient: included Intellij IDEA IDE with a starter project in Python
  - For freaks via a Spark shell:
    - Python: pyspark, Scala: spark-shell
- Spark 2.2.0 from July 2017, but ok for exercises

Apache Spark: Example

Estimating π

- We use a Monte Carlo method to estimate the value of Pi (π)
- Idea: Count the share of random points (x, y) whose distance to (0,0) is 1 or less
- Naturally parallelizable



Source: Wikipedia

Estimating Pi/4

```
N = 1000000
def inCircle(p):
    x, y = random(), random()
    return 1  if x*x + y*y < 1 else 0

myplus = lambda a, b: a + b

rawDataRDD = sc.parallelize(xrange(0,N))
inCircleRDD = rawDataRDD.map(inCircle)
count = inCircleRDD.reduce(myplus)

print "Pi is roughly %f" % (4.0 * count / N)
```

Generates two pseudo-random numbers in [0.0, 1.0)

True iff distance of (x,y) to origin is < 1

Generates an iterable ("lazy list") from 0 to N-1

A Stand-Alone Program

```
import sys
from random import random
from operator import add
from pyspark import SparkContext

if __name__ == "__main__":
"""Usage: pi [partitions]"""“

sc = SparkContext (appName="PythonPi")
partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
N = 500000 * partitions

… [code as above, without N = 1000000] …

sc.stop()
```

Execution Example

- Open terminal window
- cd spark/examples/src/main/python
- spark-submit pi.py 1
  - => Pi is roughly 3.139168
- spark-submit pi.py 2
  - => Pi is roughly 3.142220
- spark-submit pi.py 2
  - => Pi is roughly 3.138352
- spark-submit pi.py 4
  - => Pi is roughly 3.142624

Recommended Videos on Spark

- Introduction tutorials on Spark
  - Parallel programming with Spark Presented by Matei Zaharia UC Berkeley AmpLab 2013
  - https://www.youtube.com/watch?v=e-56inQL5hQ&t=30s
  - Parallel Programming with Spark (Part 1 & 2) by Matei Zaharia (2012)
  - https://www.youtube.com/watch?v=7k4yDKBYOcw
- Coursera
  - Big Data Analysis with Scala and Spark
  - https://www.coursera.org/learn/scala-spark-big-data
  - Enroll -> Audit, then for free!

Reading Materials on Spark

- Free Materials:
  - Spark Programming Guide
    - https://spark.apache.org/docs/latest/rdd-programming-guide.html
  - Apache Spark Tutorial: ML with PySpark
    - https://goo.gl/u4RjeB
  - Cheat Sheet PySpark-RDD Basics, https://goo.gl/UF5zVr
  - Jacek Laskowski, Mastering Apache Spark 2, GitBook.com, https://goo.gl/yFYRYm
- Books
  - Matthew Rathbone: 10+ Great Books for Apache Spark
    - https://blog.matthewrathbone.com/2017/01/13/spark-books.html

Thank you.