Mining Massive Datasets

Lecture 1

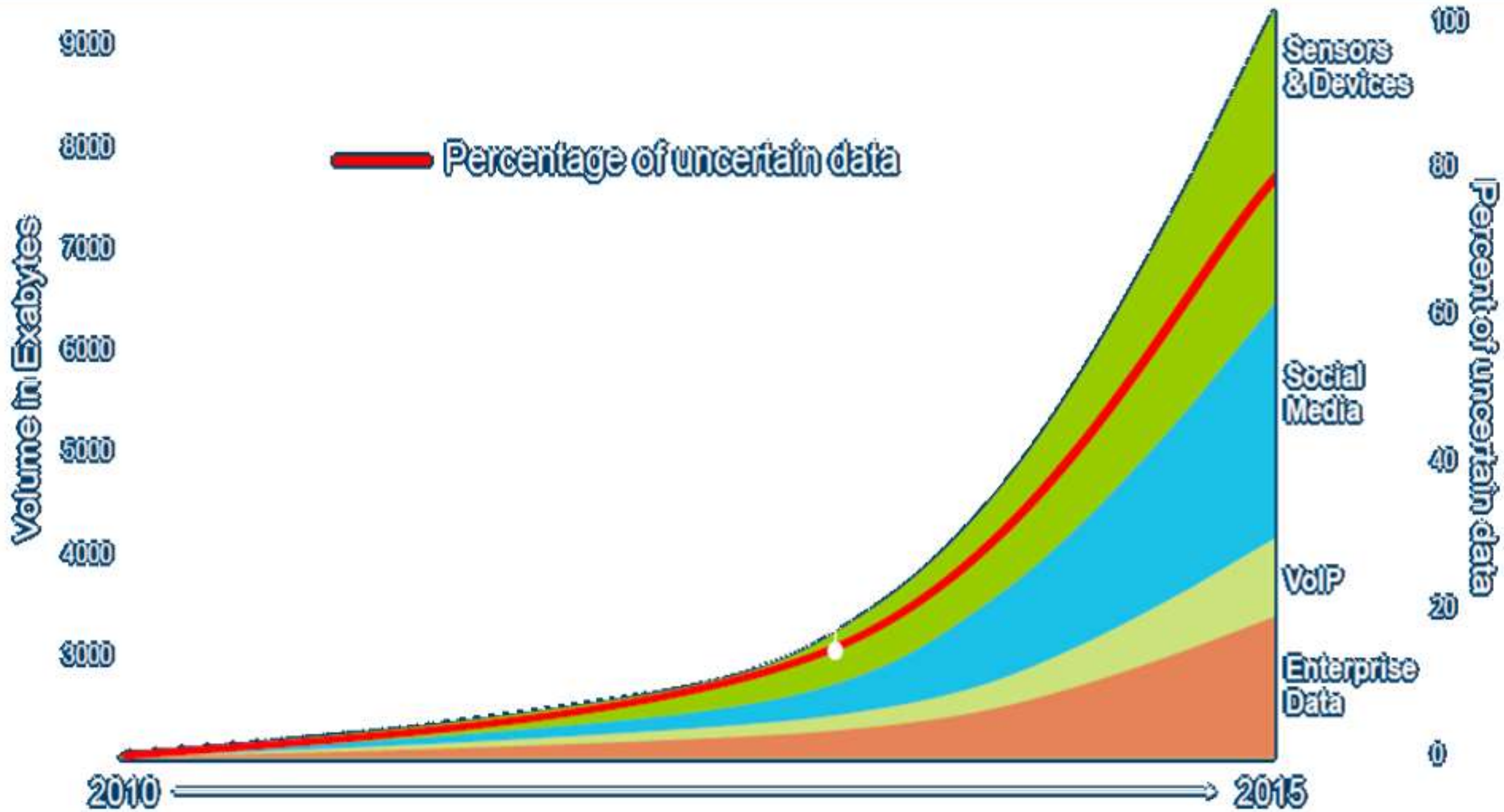
Artur Andrzejak
http://pvs.ifi.uni-heidelberg.de

Note on Slides

A substantial part of these slides come (either verbatim or in a modified form) from the book
Mining of Massive Datasets
by Jure Leskovec, Anand Rajaraman, Jeff Ullman
(Stanford University).
For more information, see the website accompanying the book: http://www.mmds.org.

Motivation and Course Contents

Lot of Data, Lot of Computing Power



Data ≠ Knowledge

- To extract the knowledge from data it needs to be
  - Stored
  - Managed
  - Processed
  - And Analyzed

This course

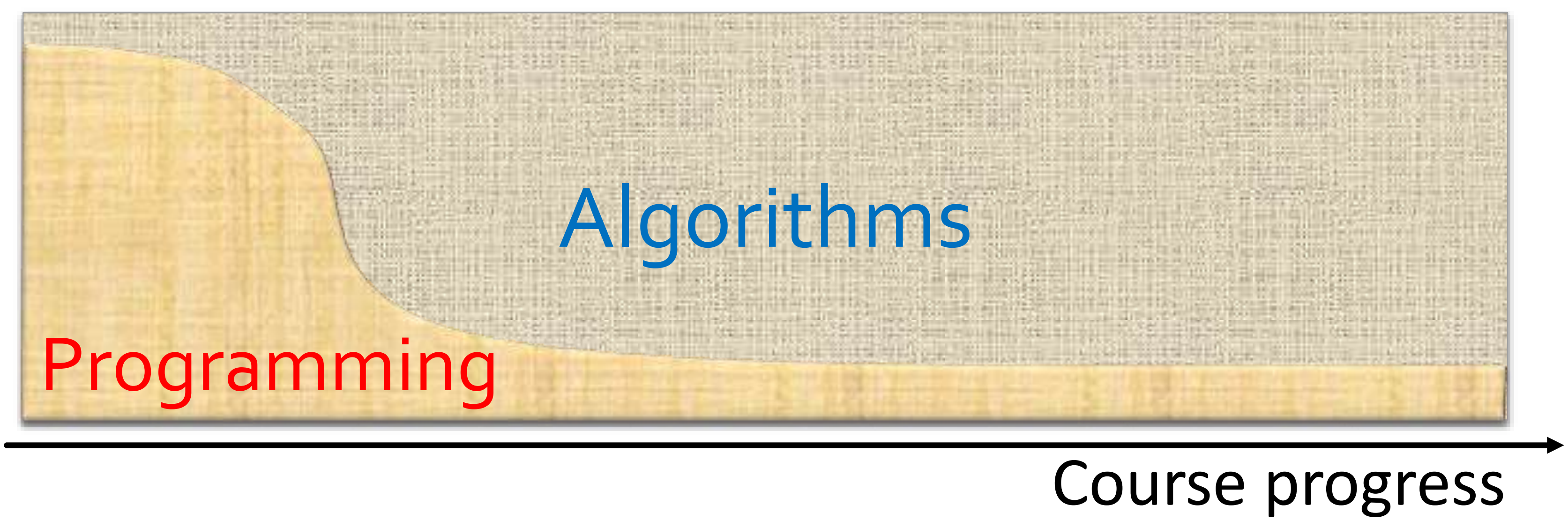
Data Mining ≈ Big Data ≈
Predictive Analytics ≈ Data Science

Data Mining Tasks

- Descriptive methods
  - Find human-interpretable patterns that describe the data
    - **Example:** Clustering
- Predictive methods
  - Use some variables to predict unknown or future values of other variables
    - **Example:** Recommender systems

Programming vs. Algorithms

- You need programming to analyze (esp. large) data
  - E.g. Wikipedia text corpus is stored as XML, to extract text you need to write scripts
- In the first part of this course, we will learn programming to process and analyze large data
- Later we will use this to implement algorithms



What will we learn?

- **We learn paradigms & tools for programming**
  - Spark & related libs (Shark, GraphX, MLlib, …)
  - Hadoop MapReduce & related frameworks (Apache Mahout, Pig, …)
  - … and a bit of Python
- **We learn to mine different types of data**
  - Data is high dimensional
  - Data is a graph
  - Data is infinite/never-ending
  - Data is labeled

What will we learn?

- **We will learn to solve real-world problems**
  - Recommender systems
  - Spam detection
  - Duplicate document detection
- **We will learn various algorithmic “tools”**
  - Linear algebra (e.g. SVD)
  - Optimization (stochastic gradient descent)
  - Dynamic programming
  - Hashing (e.g. locality-sensitive hashing, LSH)

Overview (Superset)

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Programming in Spark & MapReduce

Questionnaire

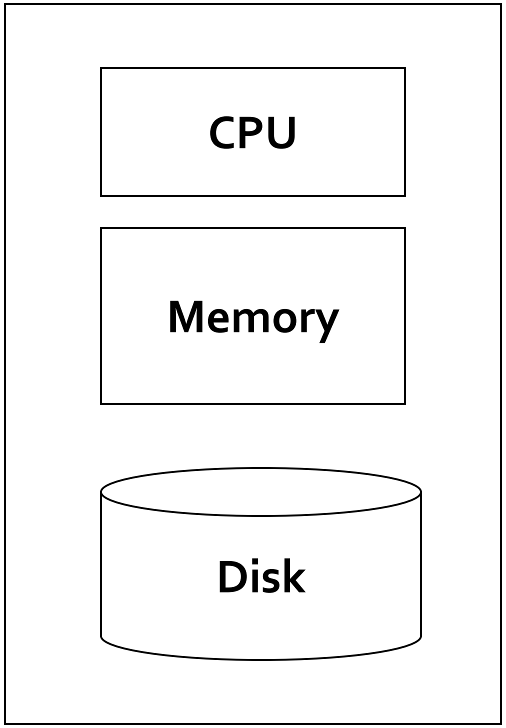
- Who have heard about MapReduce?
- … about Spark? 25 / 7
- … about Hive? 3

- Who have used Apache Hadoop? 6

- Who has programmed in Python? all
- … in Java? 25
- … in Scala?
- … in C++?

Distributed Processing: Motivation

Single Node Architecture



Machine Learning, Statistics

"Classical" Data Mining

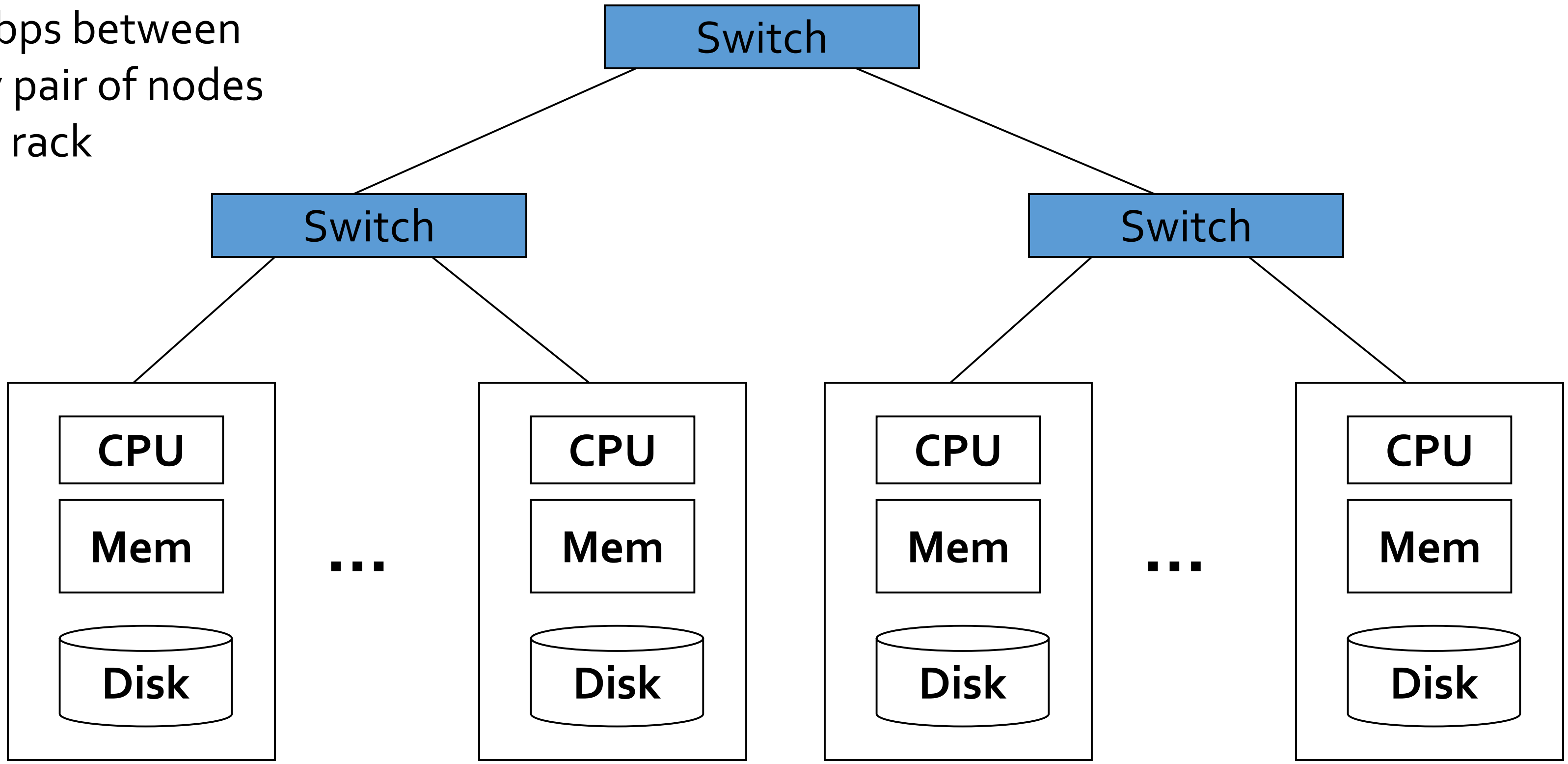
Motivation: Google Example

- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do** something useful with the data!
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between any pair of nodes in a rack



Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware
- **Challenges:**
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Dependability - machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

Storage Infrastructure

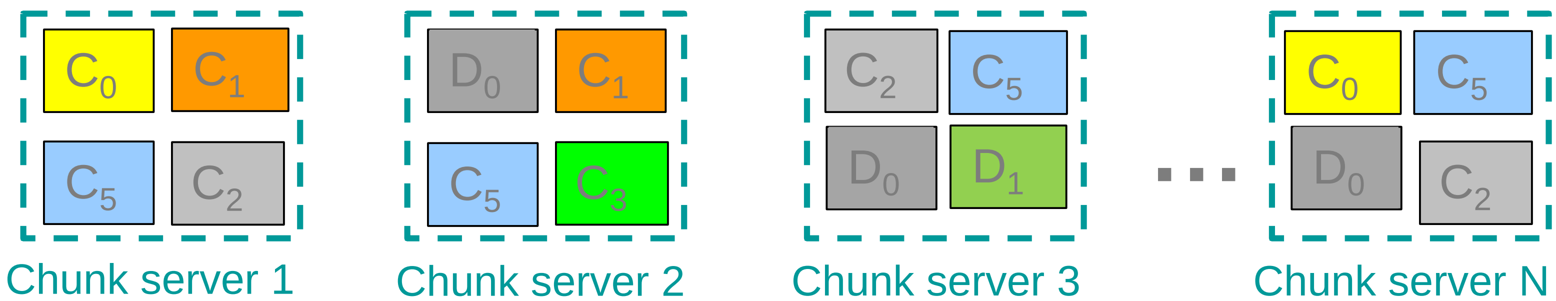
- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
    - Google GFS; Hadoop HDFS;
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

- Chunk servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- Master node
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- Client library for file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines
- Each chunk replicated on different machines
  - Seamless recovery from disk or machine failure



Chunk servers also serve as compute servers

Implementations

- **Google File System** (GFS)
  - Not available outside Google
  - More info: Wikipedia, paper (2003)
- **Hadoop Distributed File System** (HDFS)
  - An open-source implementation in Java
  - A de-facto standard for storing large data
  - Download: http://lucene.apache.org/hadoop/
- Other distributed file systems
  - Lustre (for HPC), Fossil (for HPC), GPFS (IBM), Ceph (Red Hat), XtreemFS (EU Project),…

What Is Apache Hadoop?

- "The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.
- … Modules:
  - **Hadoop Common**
  - **Hadoop Distributed File System (HDFS)**
  - **Hadoop YARN**
  - **Hadoop MapReduce**"
- Other Hadoop-related projects at Apache:
  - **Ambari™**
  - **Avro™**
  - **Cassandra™**
  - **Chukwa™**
  - **HBase™**
  - **Hive™**
  - **Mahout™**
  - **Pig™**
  - **Spark™**
  - **Tez™**
  - **ZooKeeper™**

Using HDFS – Various Ways

- Via shell commands, e.g.
  - bin/hadoop dfs -mkdir <hdfs-dir>
  - bin/hadoop dfs -put <local-dir> <hdfs-dir>
  - bin/hadoop dfs -get <hdfs-dir> <local-dir>
- By mounting HDFS like a local file system
  - http://wiki.apache.org/hadoop/MountableHDFS
- Programmatically: Java, MapReduce, Spark, ….
  - https://developer.yahoo.com/hadoop/tutorial/module2.html
- Watch video "Hadoop Tutorial: Intro to HDFS"
  - https://www.youtube.com/watch?v=ziqx2hJY8Hg

Programming Paradigms

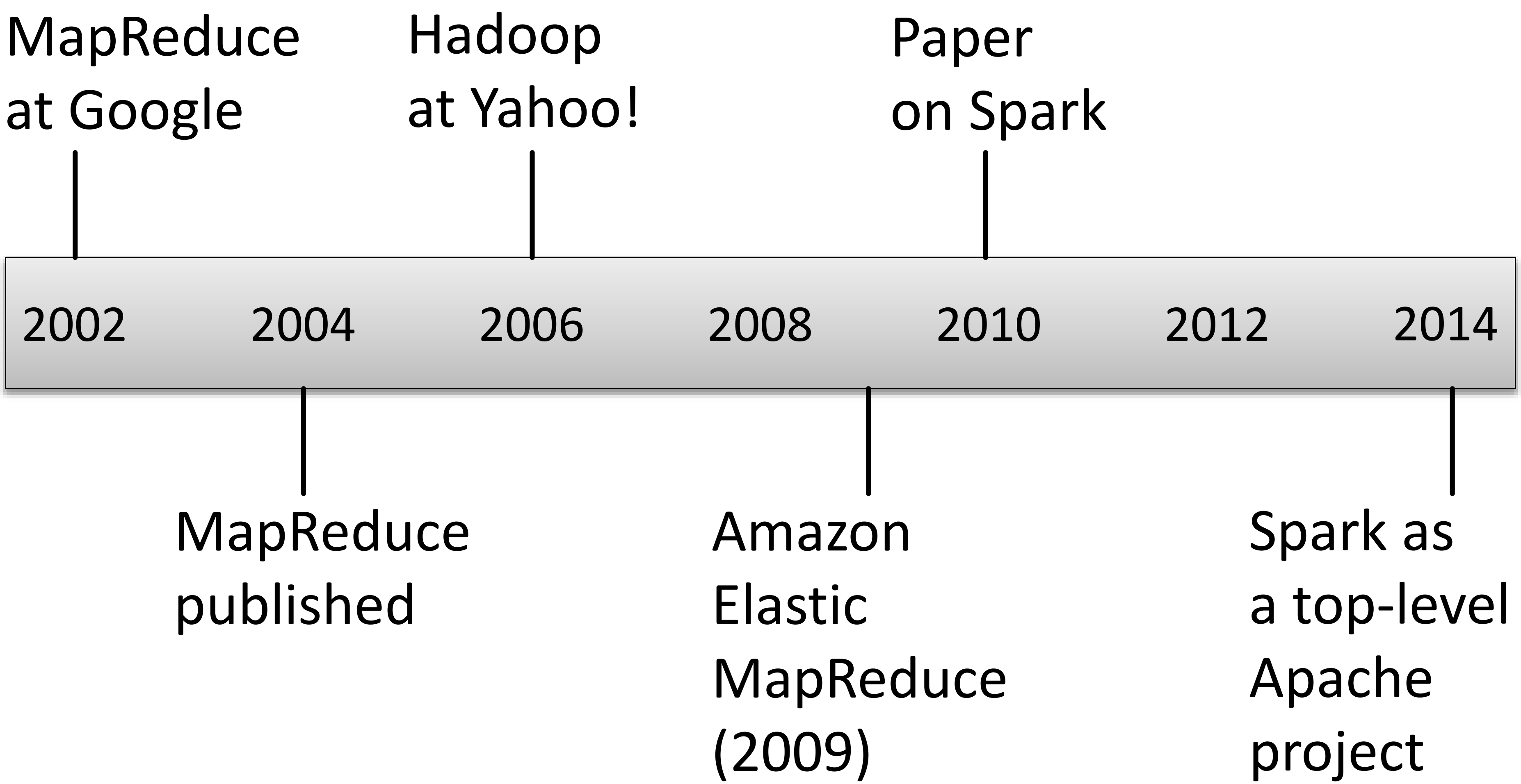
Large-Scale Data Processing

- Our focus: How to perform large scale computing for data processing & analysis?
- **Challenges**:
  - How to distribute computation?
  - How to make programming less difficult?
  - How to deal with high rate of component failures?

Efficient Frameworks Needed

- Issues: Copying data costs time / Nodes fails
- Ideas:
  - Store files multiple times for reliability
  - Bring computation close to the data
- **Solutions:**
  - Fault-tolerant storage infrastructure (file system)
    - **Hadoop Distributed File System** (HDFS)
  - Programming models (selection)
    - **Hadoop Map-Reduce**: older, but still most widely used
    - Apache **Spark**: richer set of operators & faster

A Brief History



MapReduce Problems

- Difficulty of programming
- Performance bottlenecks
  - Esp. for iterative jobs or “multi-MR” jobs
- MapReduce is not an ideal paradigm for large applications (with multiple processing phases)
- => A lot of specialized systems were created as workarounds

MapReduce Problems

MapReduce

Pregel
Giraph

Hive
Drill
Tez

Impala
GraphLab

Storm
S4
…

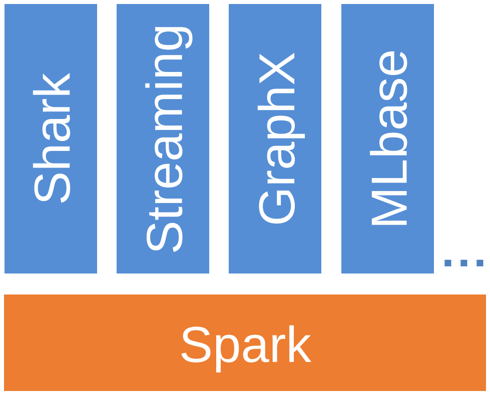
General batch processing

Specialized systems (graph processing, databases, streaming, ...)

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

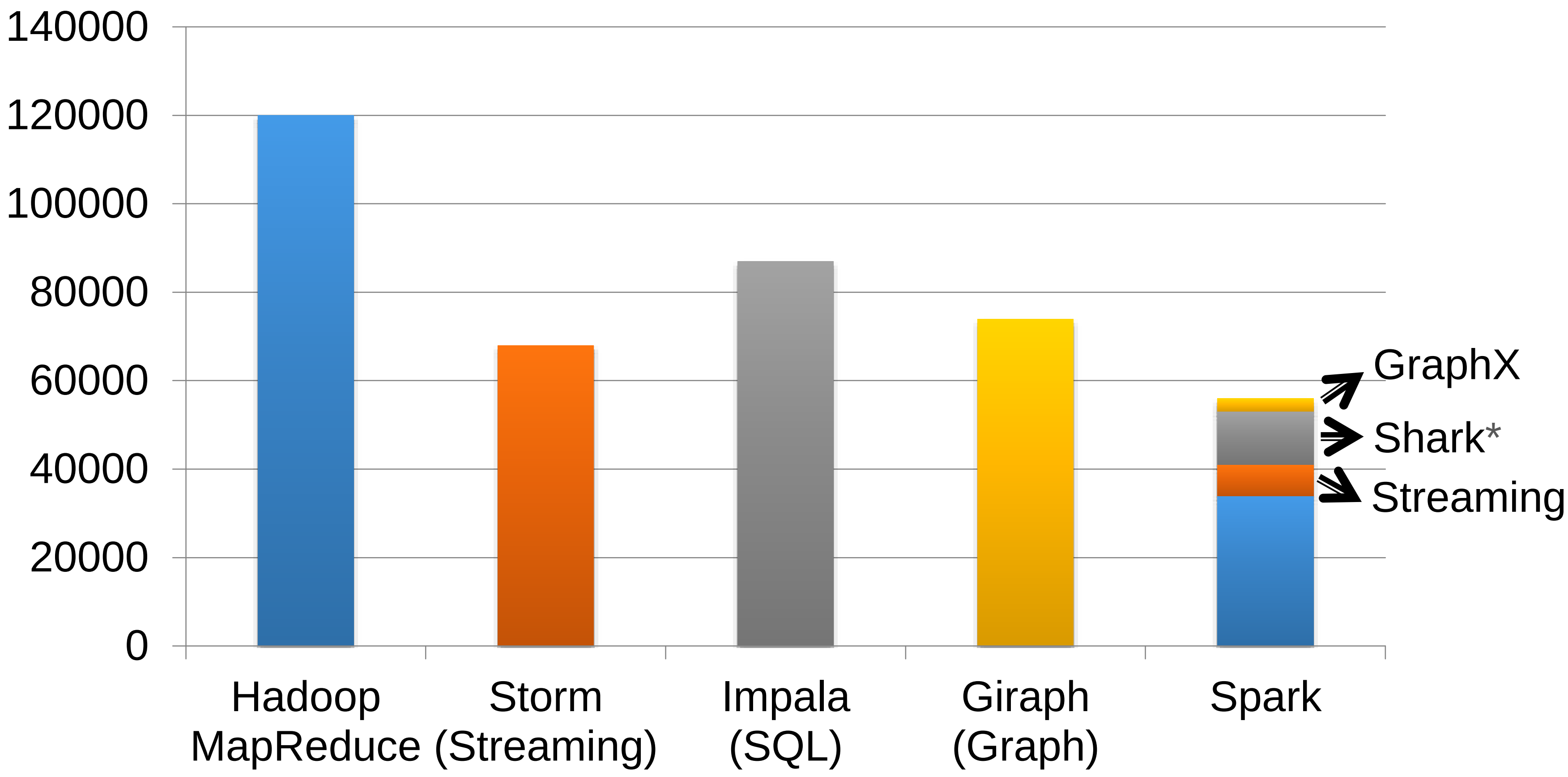
Spark's Approach

- 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

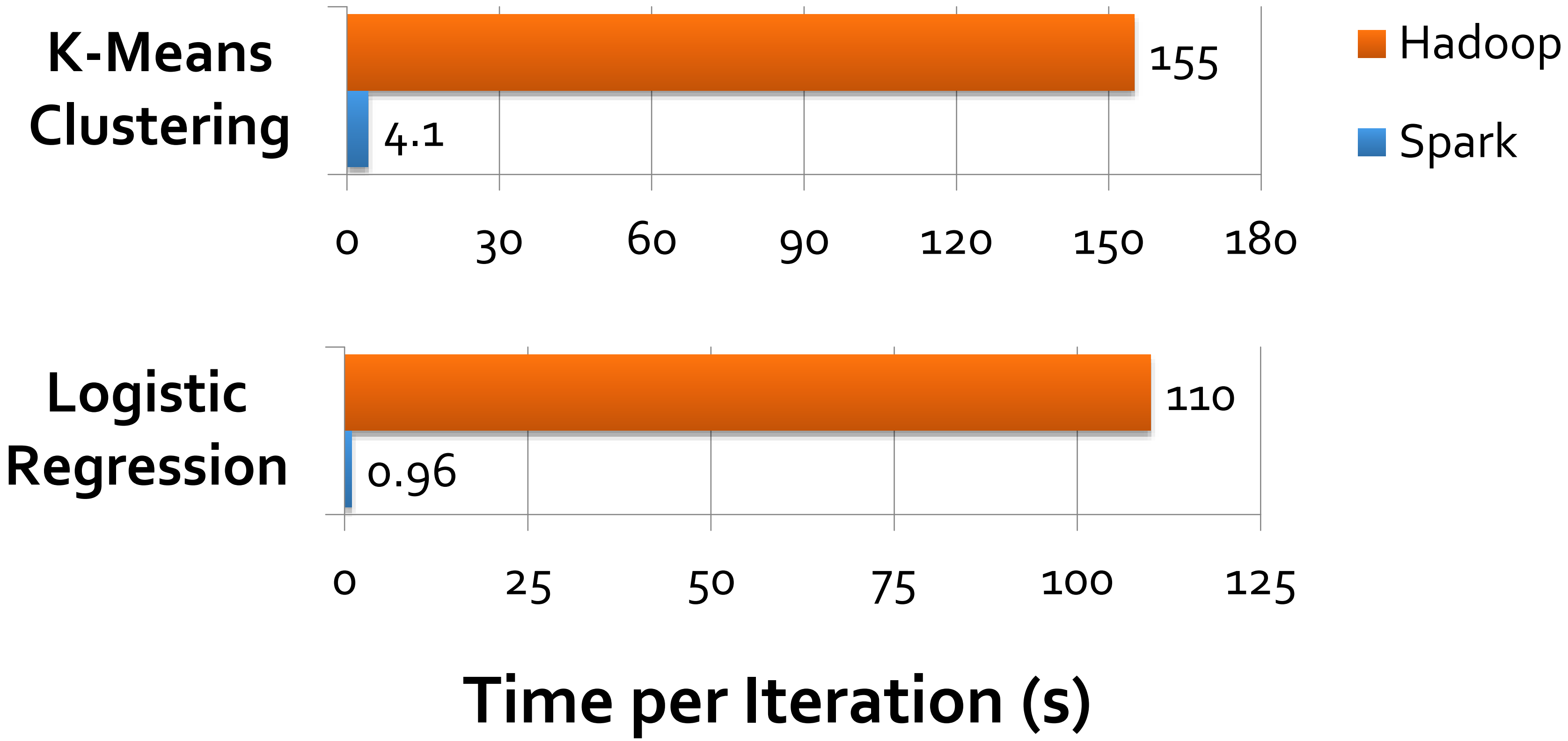


non-test, non-example source lines

* also calls into Hive

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

Performance: Iterative Algorithms



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

MapReduce is Being Replaced

*Website of **Mahout**, a library for large-scale machine learning algorithms (as of 10 Oct 2014)*

25 April 2014 - Goodbye MapReduce

The Mahout community decided to move its codebase onto modern data processing systems that offer a richer programming model and more efficient execution than Hadoop MapReduce. **Mahout will therefore reject new MapReduce algorithm implementations from now on**.

...

We are building our future implementations on top of a DSL for linear algebraic operations which has been developed over the last months. Programs written in this DSL are automatically optimized and executed in parallel on Apache Spark.

Spark Programming: Introduction

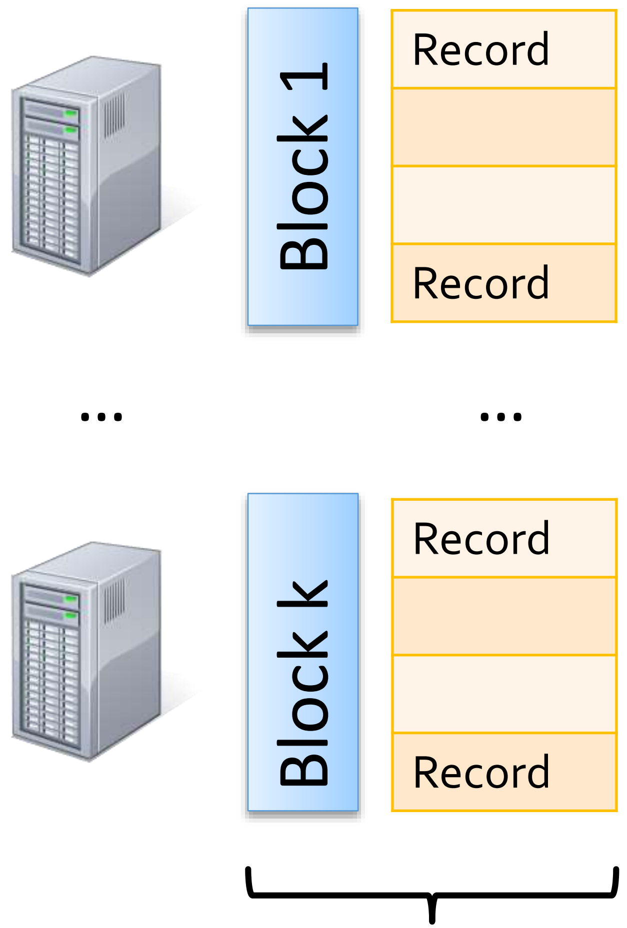
Virtual Machine with Spark

- We have prepared a virtual machine with Spark (under Oracle VirtualBox)
- Available on a USB stick
- 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

Key Idea 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

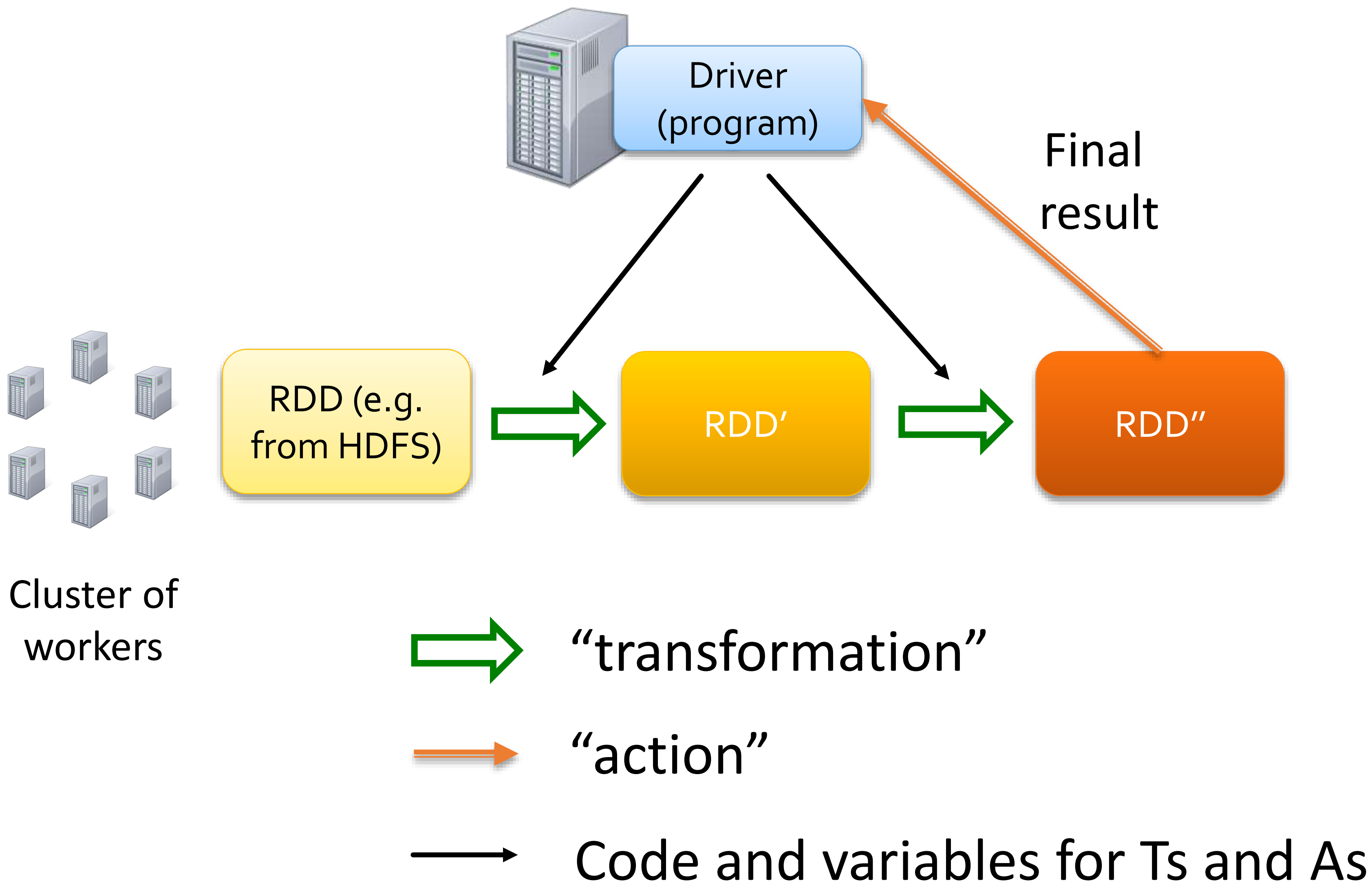
Resilient Distributed Dataset (RDD)



Can be a text line, a sting, a (key, value) pair, or any serializable object

A RDD; accessible in code via a single "variable"

Computations with RDDs



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

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}
```

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

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

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
```

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

Some Key-Value Operations

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

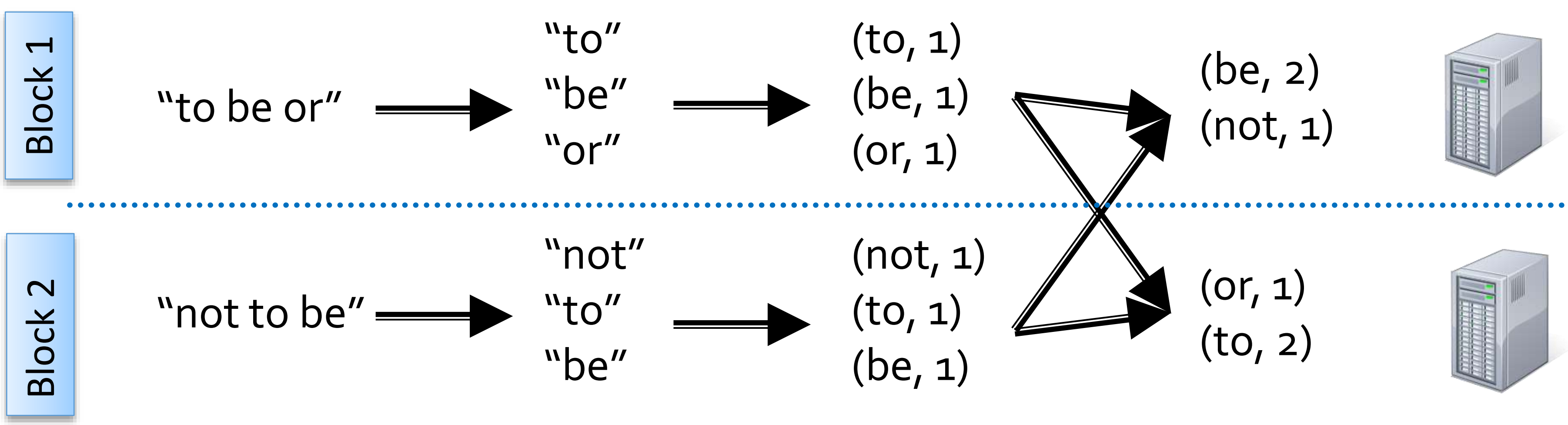
- ```
  pets.reduceByKey(lambda x, y: x + y)
                   # => {(cat, 3), (dog, 1)}
  ```

- ```
  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
  ```

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)`

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

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!)

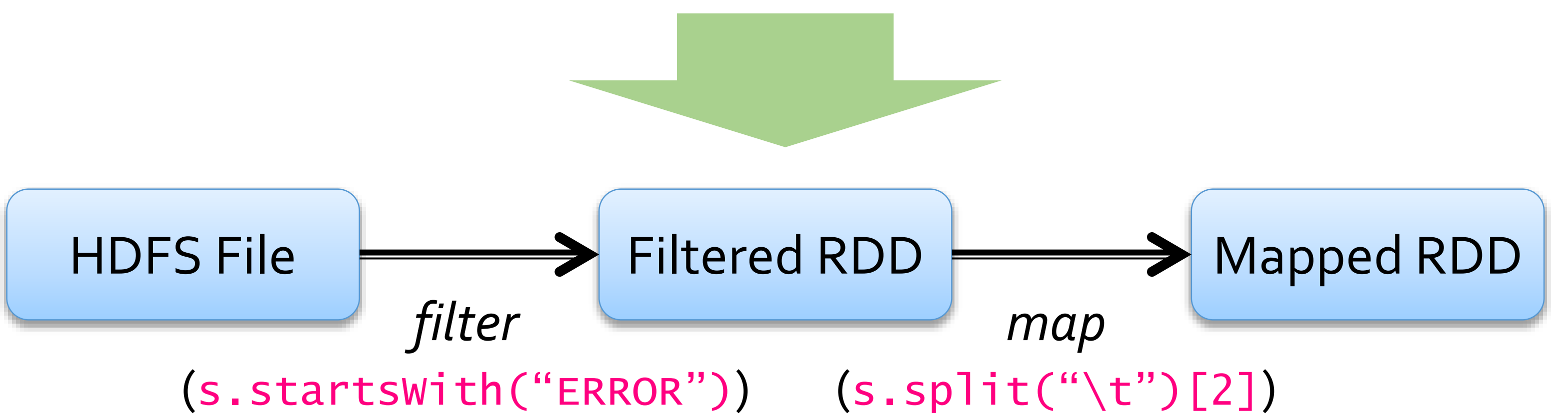
There are also shared variables:
broadcasts, accumulators

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

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

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

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

More Resources

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System
  - http://labs.google.com/papers/gfs.html
- Hadoop Wiki
  - Introduction: http://wiki.apache.org/lucene-hadoop/
  - Getting Started: http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses

Thank you.

Questions?