Fast and Expressive Big Data Analytics with Python

Matei Zaharia

UC Berkeley / MIT

spark-project.org

What is Spark?

Fast and expressive cluster computing system interoperable with Apache Hadoop

Improves efficiency through:

» In-memory computing primitives
» General computation graphs

→ Up to 100× faster (2-10× on disk)

Improves usability through:

» Rich APIs in Scala, Java, Python
» Interactive shell

→ Often 5× less code

Project History

Started in 2009, open sourced 2010

17 companies now contributing code

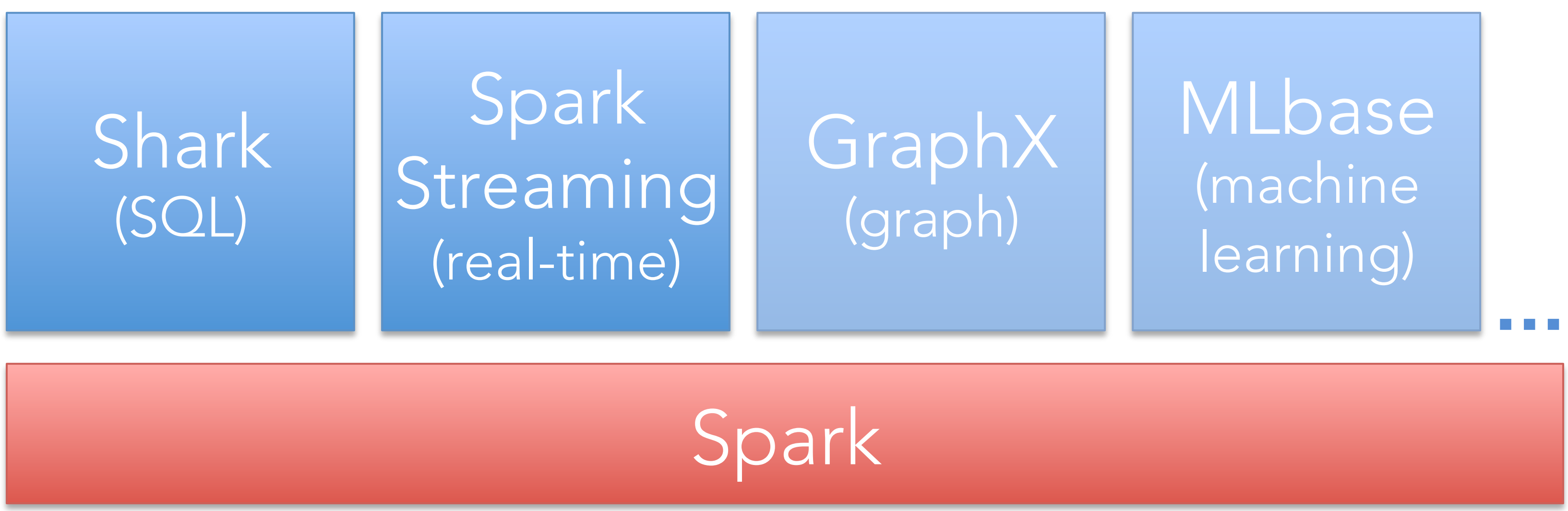
» Yahoo!, Intel, Adobe, Quantifind, Conviva, Bizo, …

Entered Apache incubator in June

Python API added in February

An Expanding Stack

Spark is the basis for a wide set of projects in the Berkeley Data Analytics Stack (BDAS)



More details: amplab.berkeley.edu

This Talk

- Spark programming model
- Examples
- Demo
- Implementation
- Trying it out

Why a New Programming Model?

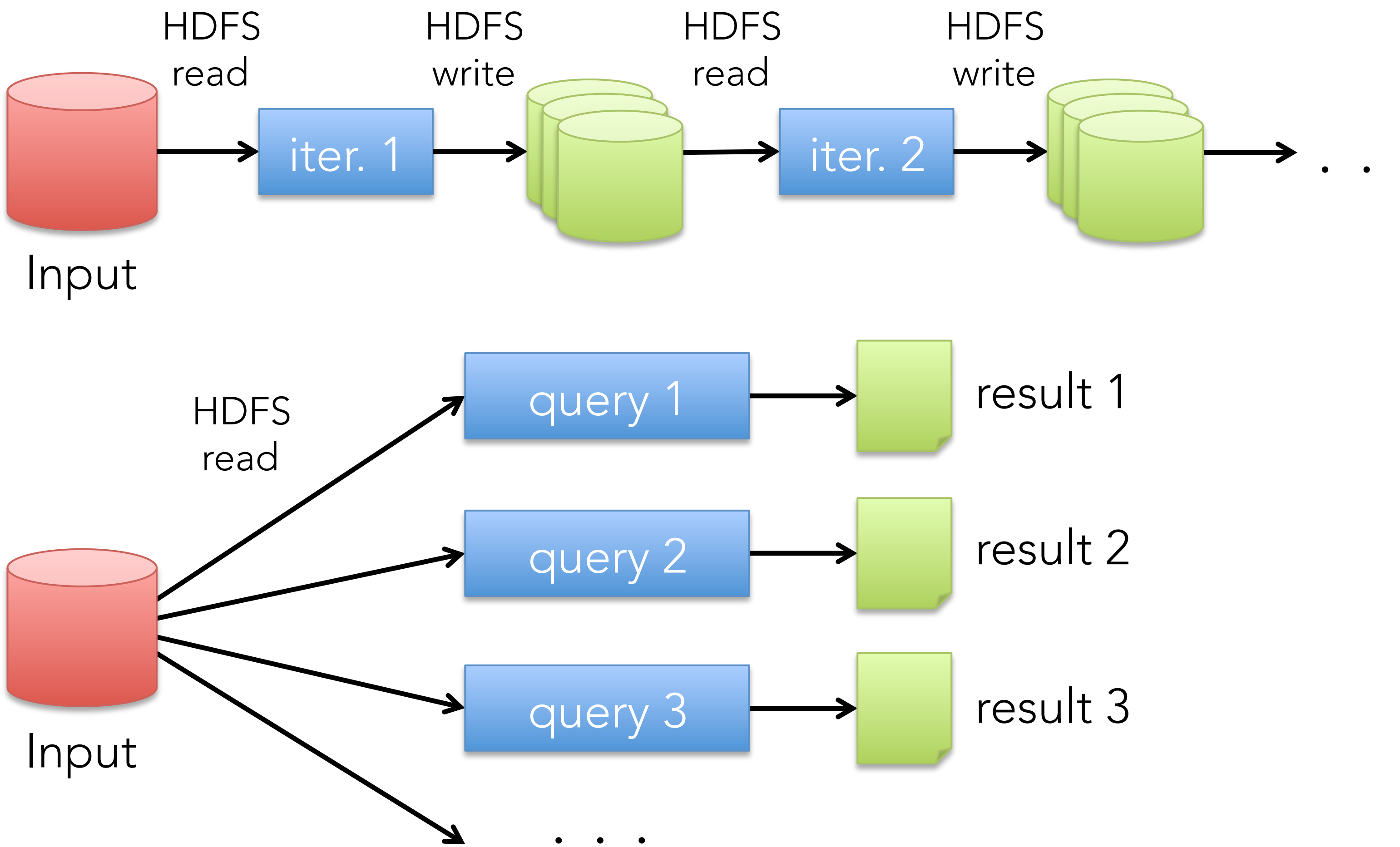
MapReduce simplified big data processing, but users quickly found two problems:

Programmability: tangle of map/red functions

Speed: MapReduce inefficient for apps that share data across multiple steps

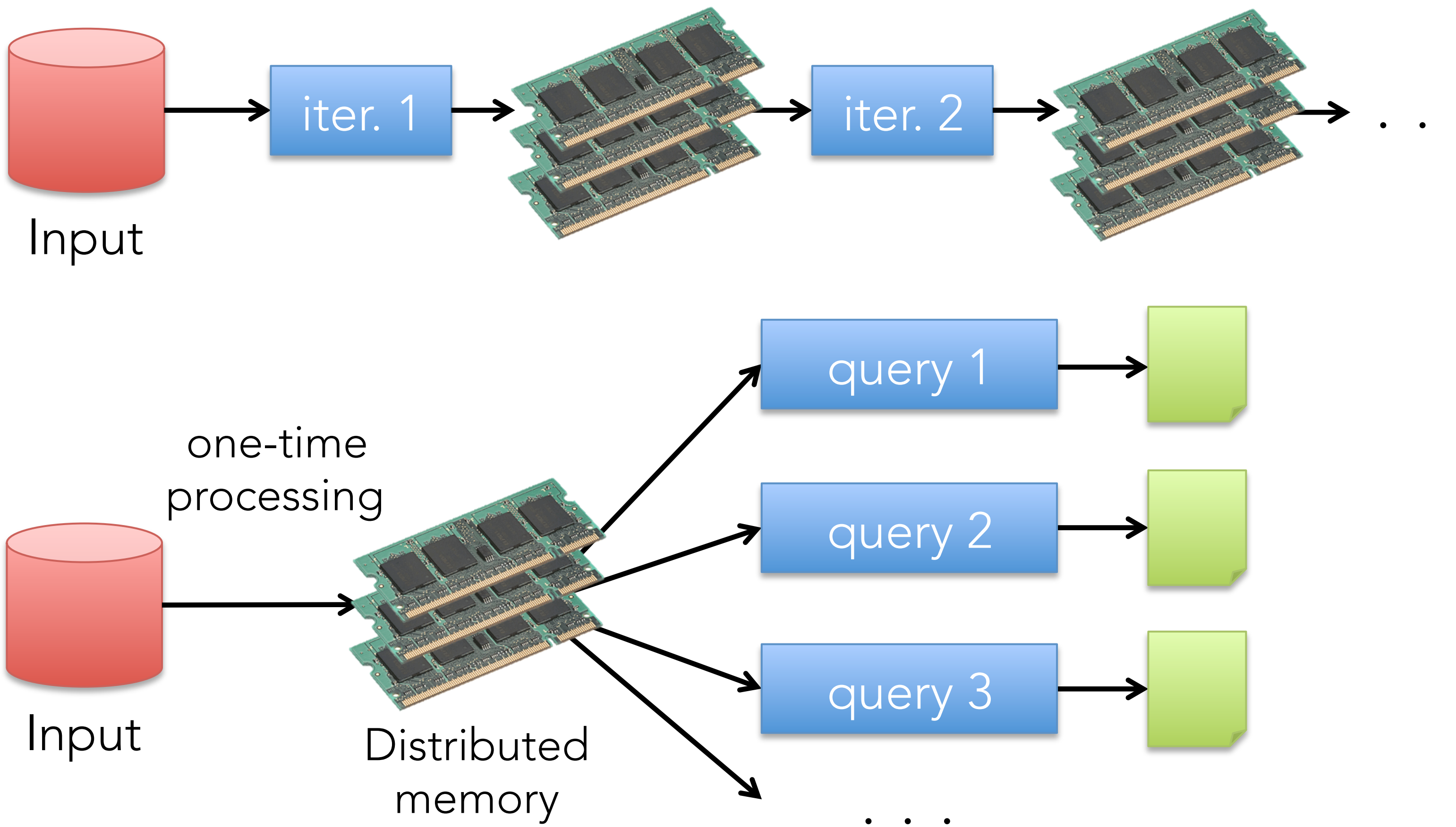
- » Iterative algorithms, interactive queries

Data Sharing in MapReduce



Slow due to data replication and disk I/O

What We'd Like



10-100× faster than network and disk

Spark Model

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets (RDDs)

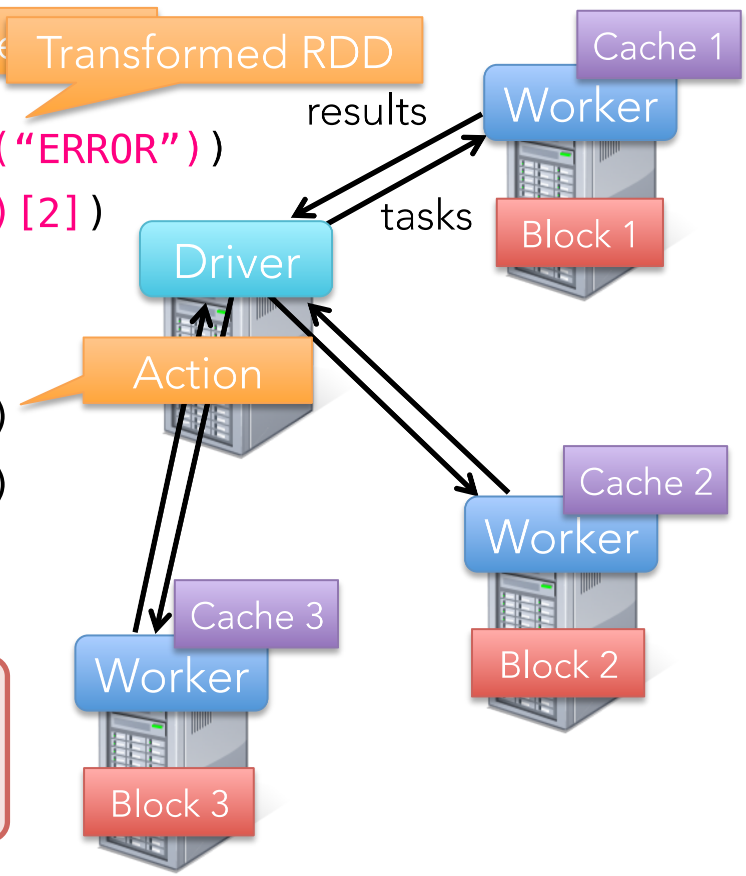
» Collections of objects that can be stored in memory or disk across a cluster
» Built via parallel transformations (map, filter, …)
» Automatically rebuilt on failure

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()

messages.filter(lambda s: “foo” in s).count()
messages.filter(lambda s: “bar” in s).count()
. . .
```

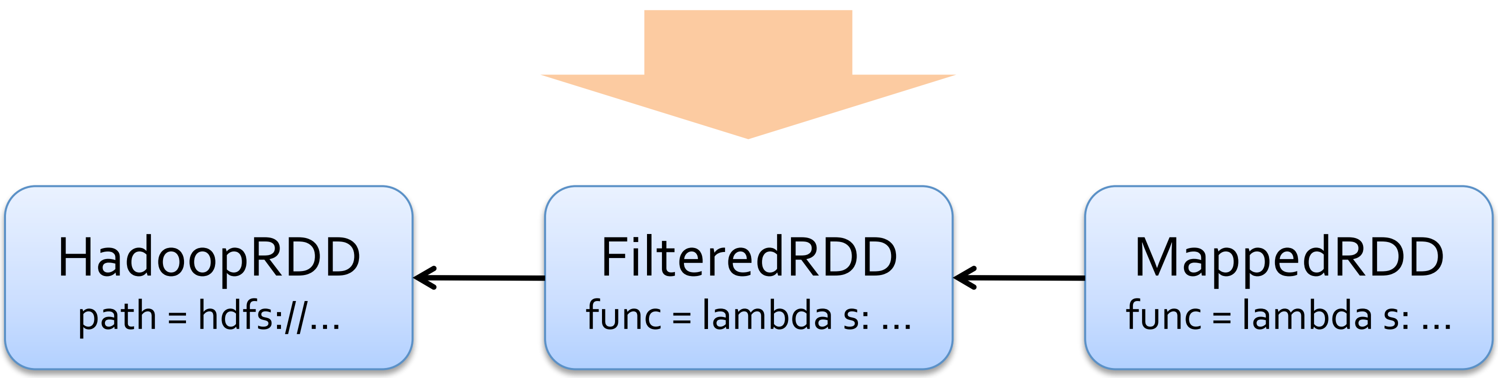


Result: scaled to 1 TB data in 7 sec (vs 180 sec for on-disk data)

Fault Tolerance

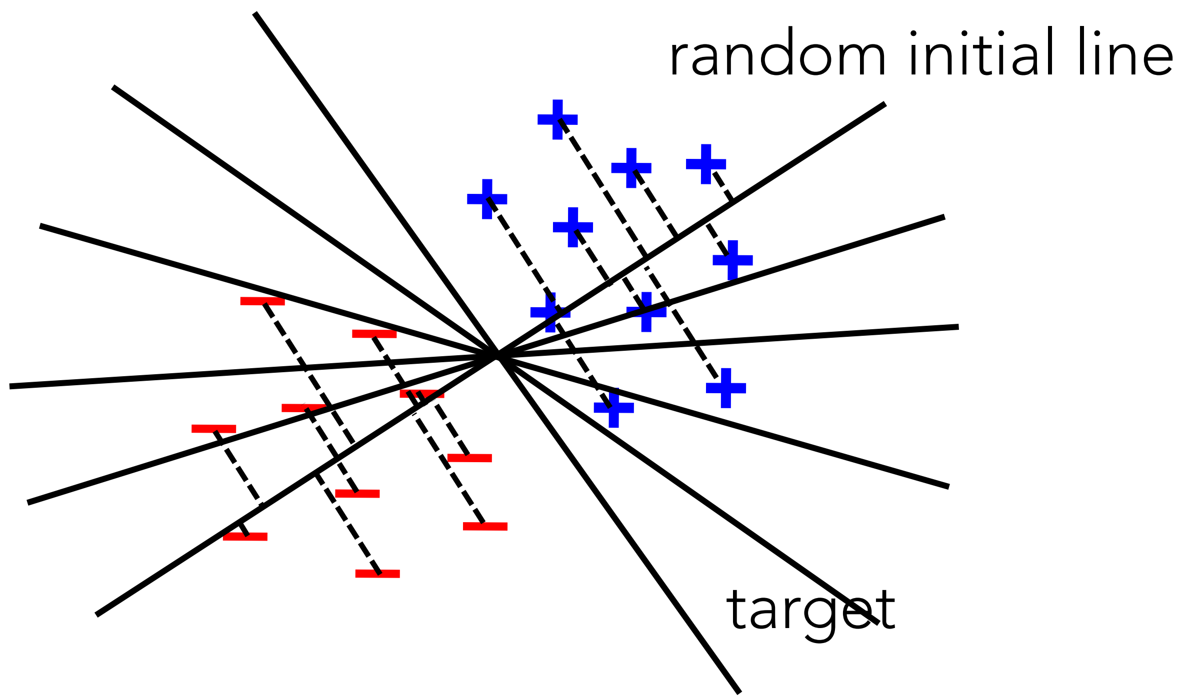
RDDs track the transformations used to build them (their *lineage*) to recompute lost data

```
messages = textFile(...).filter(lambda s: “ERROR” in s)
                        .map(lambda s: s.split(“\t”)[2])
```



Example: Logistic Regression

Goal: find line separating two sets of points



Example: Logistic Regression

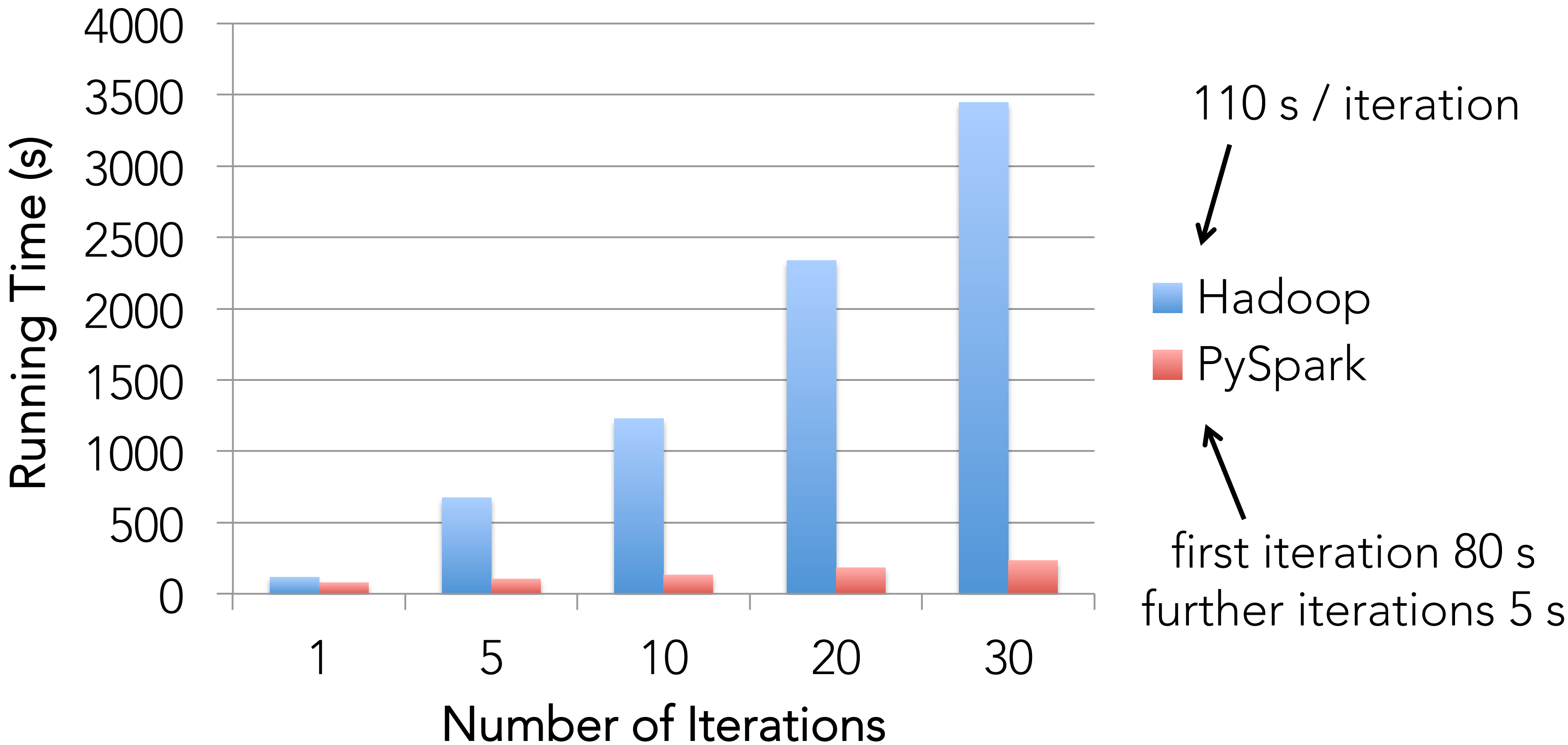
```
data = spark.textFile(...).map(readPoint).cache()

w = numpy.random.rand(D)

for i in range(iterations):
    gradient = data.map(lambda p:
        (1 / (1 + exp(-p.y * w.dot(p.x)))) * p.y * p.x
    ).reduce(lambda x, y: x + y)
    w -= gradient

print “Final w: %s” % w
```

Logistic Regression Performance



Demo

Supported Operators

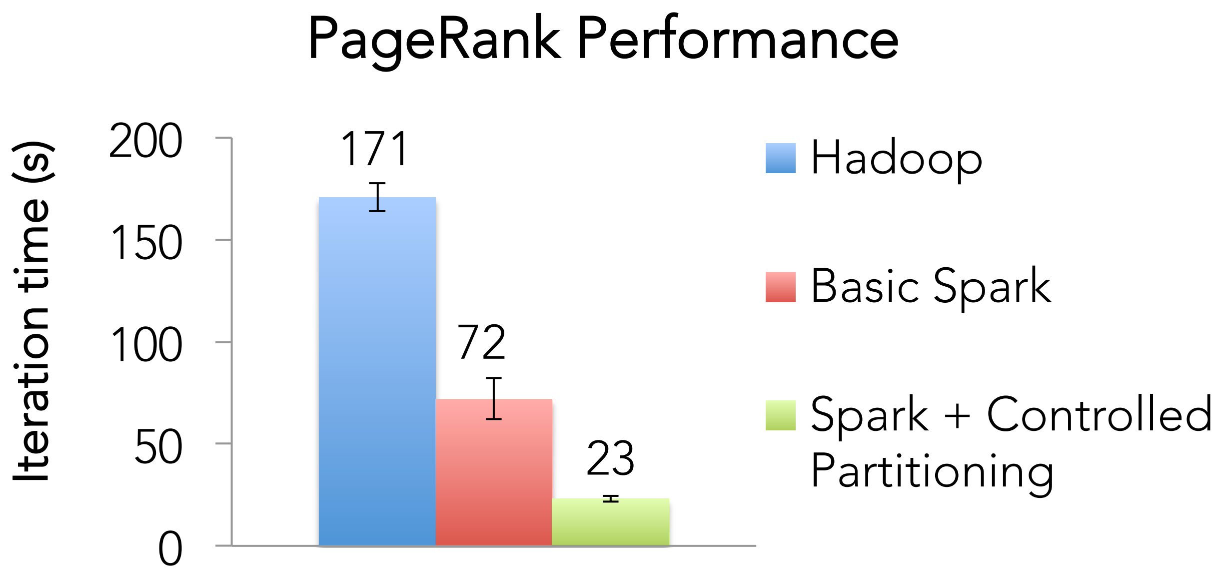
`map`	`reduce`	`take`
`filter`	`count`	`first`
`groupBy`	`fold`	`partitionBy`
`union`	`reduceByKey`	`pipe`
`join`	`groupByKey`	`distinct`
`leftOuterJoin`	`cogroup`	`save`
`rightOuterJoin`	`flatMap`	`...`

Other Engine Features

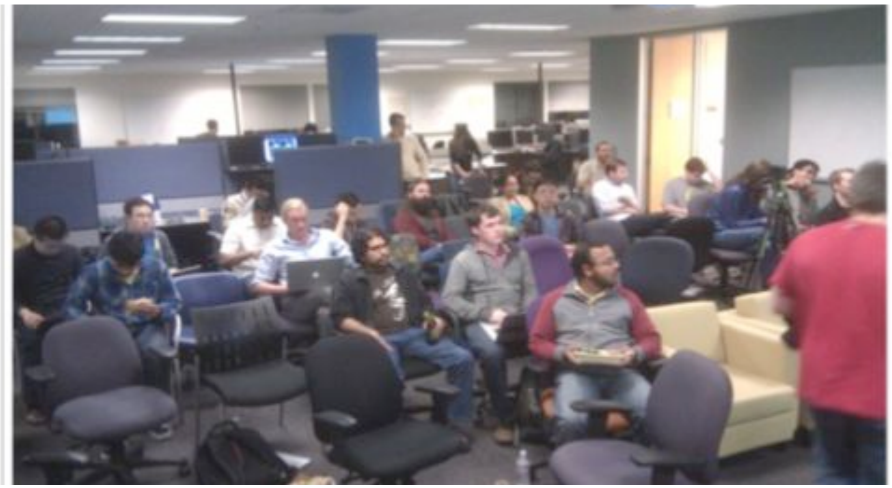
General operator graphs (not just map-reduce)

Hash-based reduces (faster than Hadoop's sort)

Controlled data partitioning to save communication



Spark Community



1000+ meetup members

60+ contributors

17 companies contributing

YAHOO! intel Adobe UCSF 阿里巴巴 Alibaba.com

celtra webtrends ClearStory DATA Now You See It AdMobius

CONVIVA bizo TAGGED quantiFind

This Talk

Spark programming model

Examples

Demo

Implementation

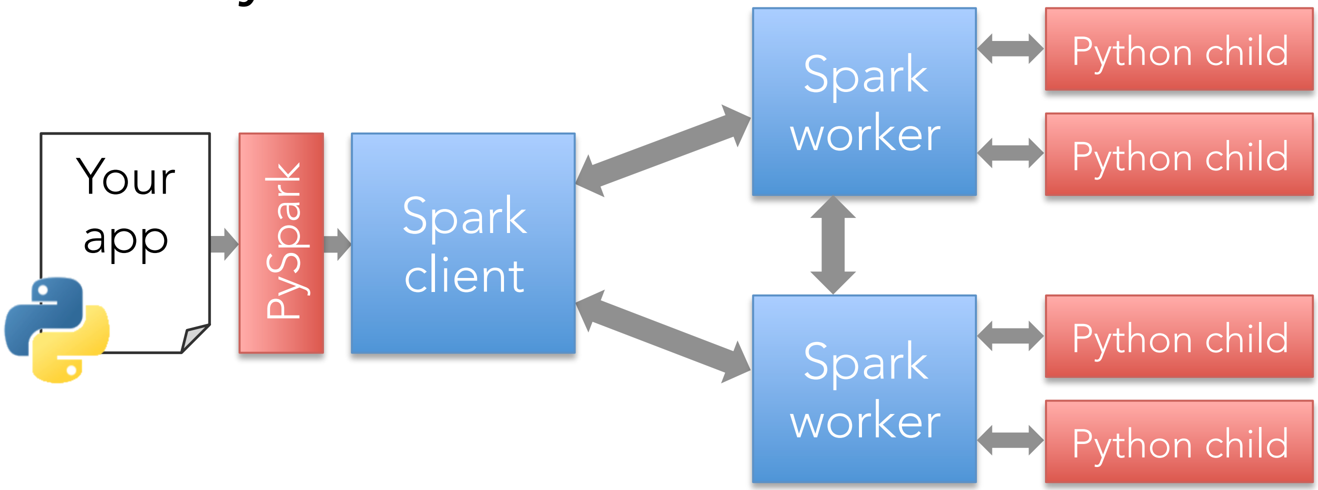
Trying it out

Overview

Spark core is written in Scala

PySpark calls existing scheduler, cache and networking layer (2K-line wrapper)

No changes to Python



Overview

Spark core is written in Scala

PySpark calls existing scheduler, cache and networking layer (2K-line wrapper)

No changes to Python

Main PySpark author:
Josh Rosen

cs.berkeley.edu/~joshrosen

Object Marshaling

Uses pickle library for both communication and cached data

- » Much cheaper than Python objects in RAM

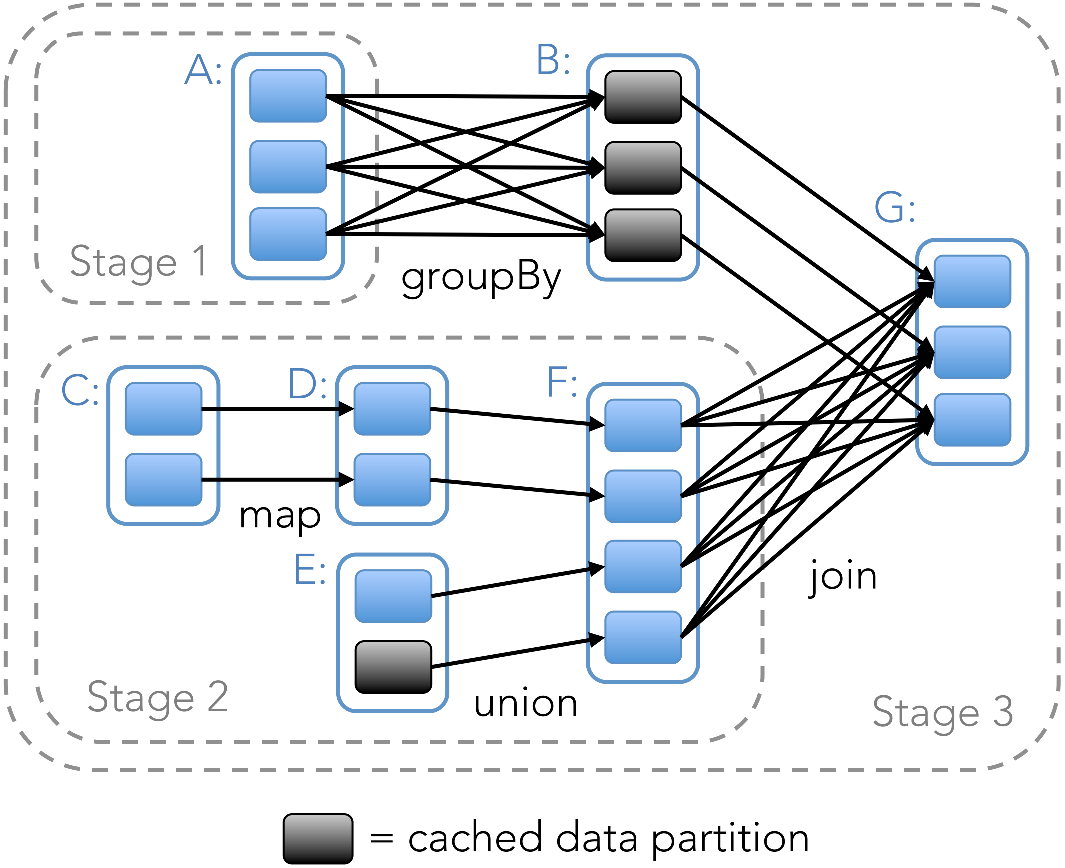
Lambda marshaling library by PiCloud

Job Scheduler

Supports general operator graphs

Automatically pipelines functions

Aware of data locality and partitioning



Interoperability

Runs in standard CPython, on Linux / Mac

» Works fine with extensions, e.g. NumPy

Input from local file system, NFS, HDFS, S3

» Only text files for now

Works in IPython, including notebook

Works in doctests – see our tests!

Getting Started

Visit spark-project.org for video tutorials, online exercises, docs

Easy to run in local mode (multicore), standalone clusters, or EC2

Training camp at Berkeley in August (free video): ampcamp.berkeley.edu

Getting Started

Easiest way to learn is the shell:

```
$ ./pyspark

>>> nums = sc.parallelize([1,2,3]) # make RDD from array

>>> nums.count()
3

>>> nums.map(lambda x: 2 * x).collect()
[2, 4, 6]
```

Writing Standalone Jobs

```
from pyspark import SparkContext

if __name__ == "__main__":
    sc = SparkContext("local", "WordCount")
    lines = sc.textFile("in.txt")

    counts = lines.flatMap(lambda s: s.split()) \
                  .map(lambda word: (word, 1)) \
                  .reduceByKey(lambda x, y: x + y)

    counts.saveAsTextFile("out.txt")
```

Conclusion

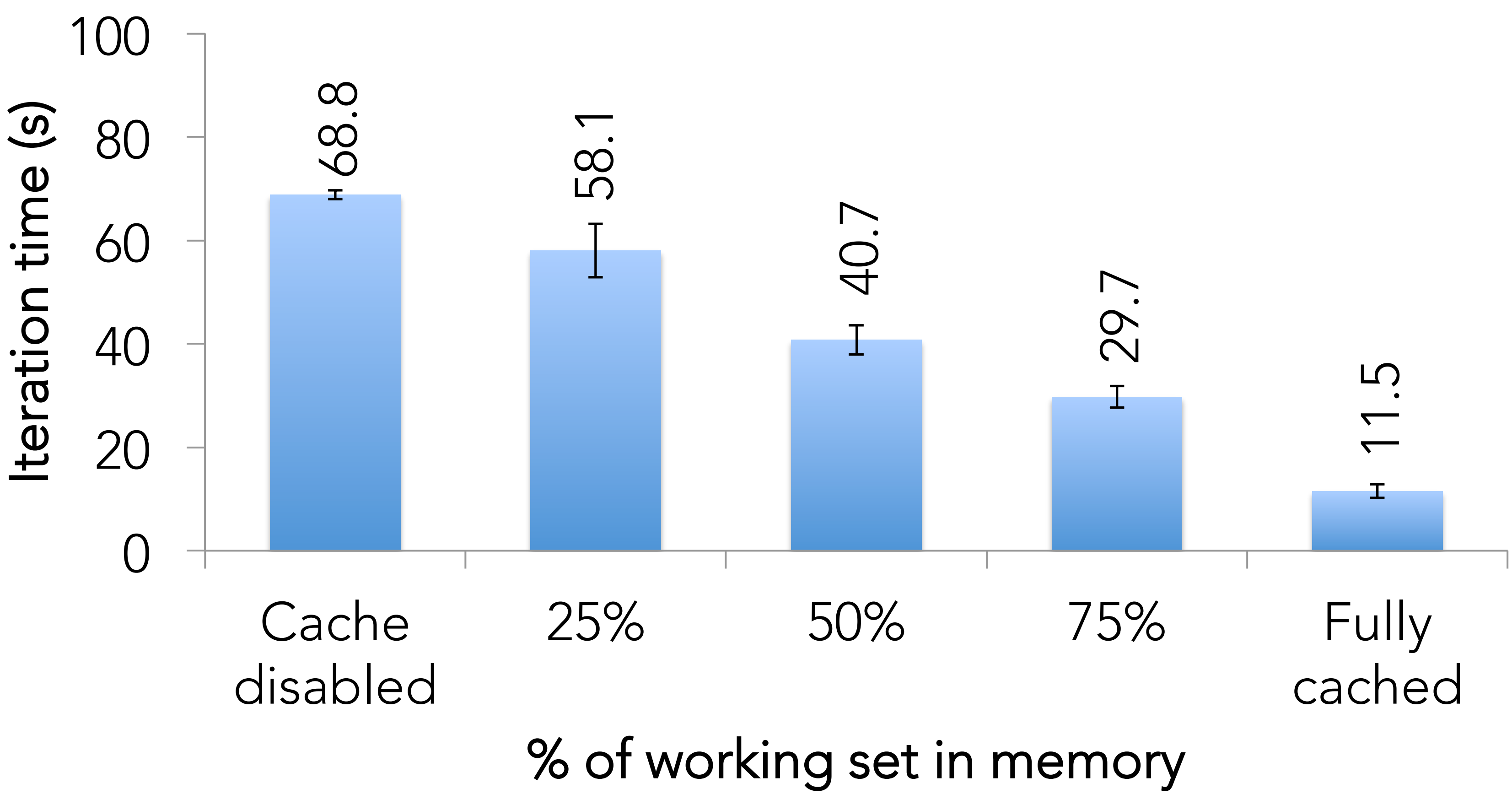
PySpark provides a fast and simple way to analyze big datasets from Python

Learn more or contribute at spark-project.org

Look for our training camp on August 29-30!

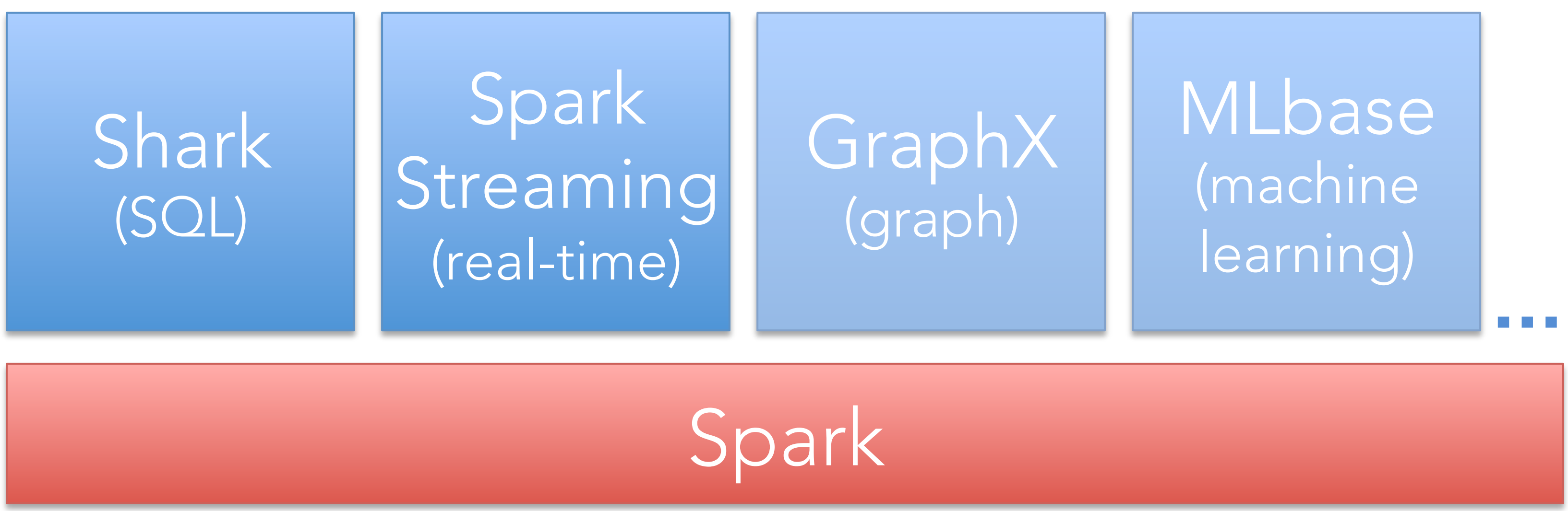
My email: matei@berkeley.edu

Behavior with Not Enough RAM



The Rest of the Stack

Spark is the foundation for wide set of projects in the Berkeley Data Analytics Stack (BDAS)



More details: amplab.berkeley.edu

Performance Comparison