Making Big Data Processing Simple with Spark

Matei Zaharia

December 17, 2015

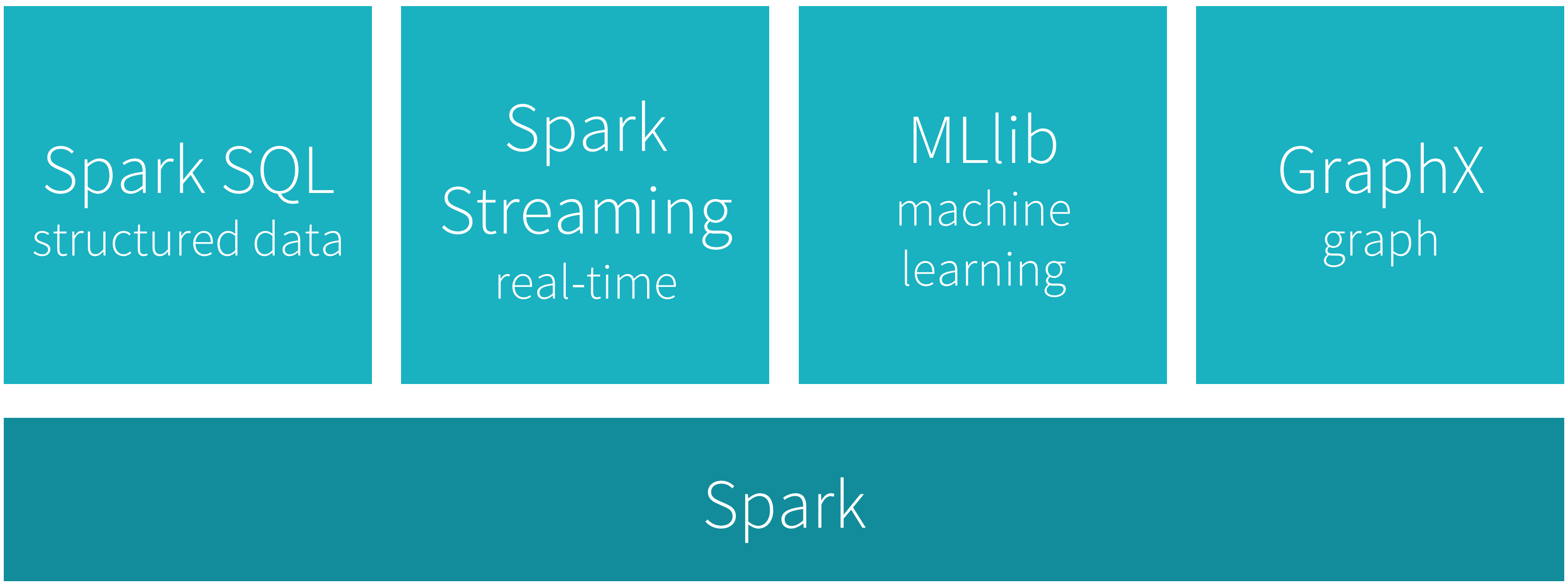
What is Apache Spark?

Fast and general cluster computing engine that generalizes the MapReduce model

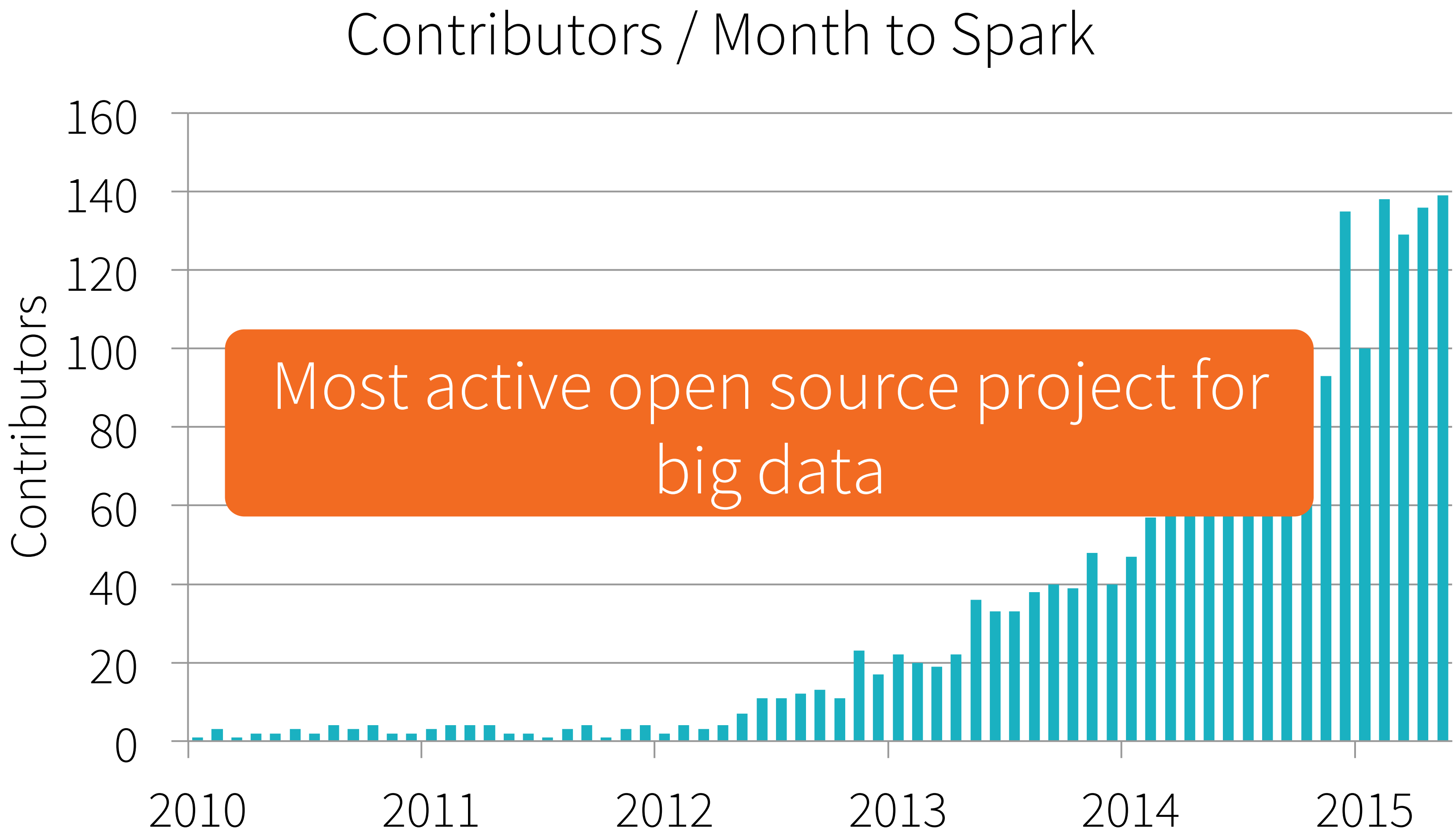
Makes it easy and fast to process large datasets

- High-level APIs in Java, Scala, Python, R
- Unified engine that can capture many workloads

A Unified Engine



A Large Community



Overview

Why a unified engine?

Spark programming model

Built-in libraries

Applications

History: Cluster Computing

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with

MapReduce

A *general* engine for batch processing

We wrote the first version of the MapReduce library in February of 2003, and made significant enhancements to it in August of 2003, including the locality optimization, dynamic load balancing of task execution across worker machines, etc. Since that time, we have been pleasantly surprised at how broadly applicable the MapReduce library has been for the kinds of problems we work on. It has been used across a wide range of domains within Google, including:

Beyond MapReduce

MapReduce was great for batch processing, but users quickly needed to do more:

- More complex, multi-pass algorithms
- More interactive ad-hoc queries
- More real-time stream processing

Result: *specialized* systems for these workloads

Big Data Systems Today

MapReduce → Pregel, Giraph, Dremel, Drill, Impala, Presto, Storm, S4 . . .

General batch processing

Specialized systems for new workloads

Problems with Specialized Systems

More systems to manage, tune, deploy

Can't easily *combine* processing types

- Even though most applications need to do this!
- E.g. load data with SQL, then run machine learning

In many cases, data transfer between engines is a dominant cost!

Big Data Systems Today

MapReduce

Pregel
Giraph
Dremel
Drill
Impala
Presto
Storm
S4
. . .

General batch processing

Specialized systems for new workloads

Unified engine

Overview

Why a unified engine?

Spark programming model

Built-in libraries

Applications

databricks™

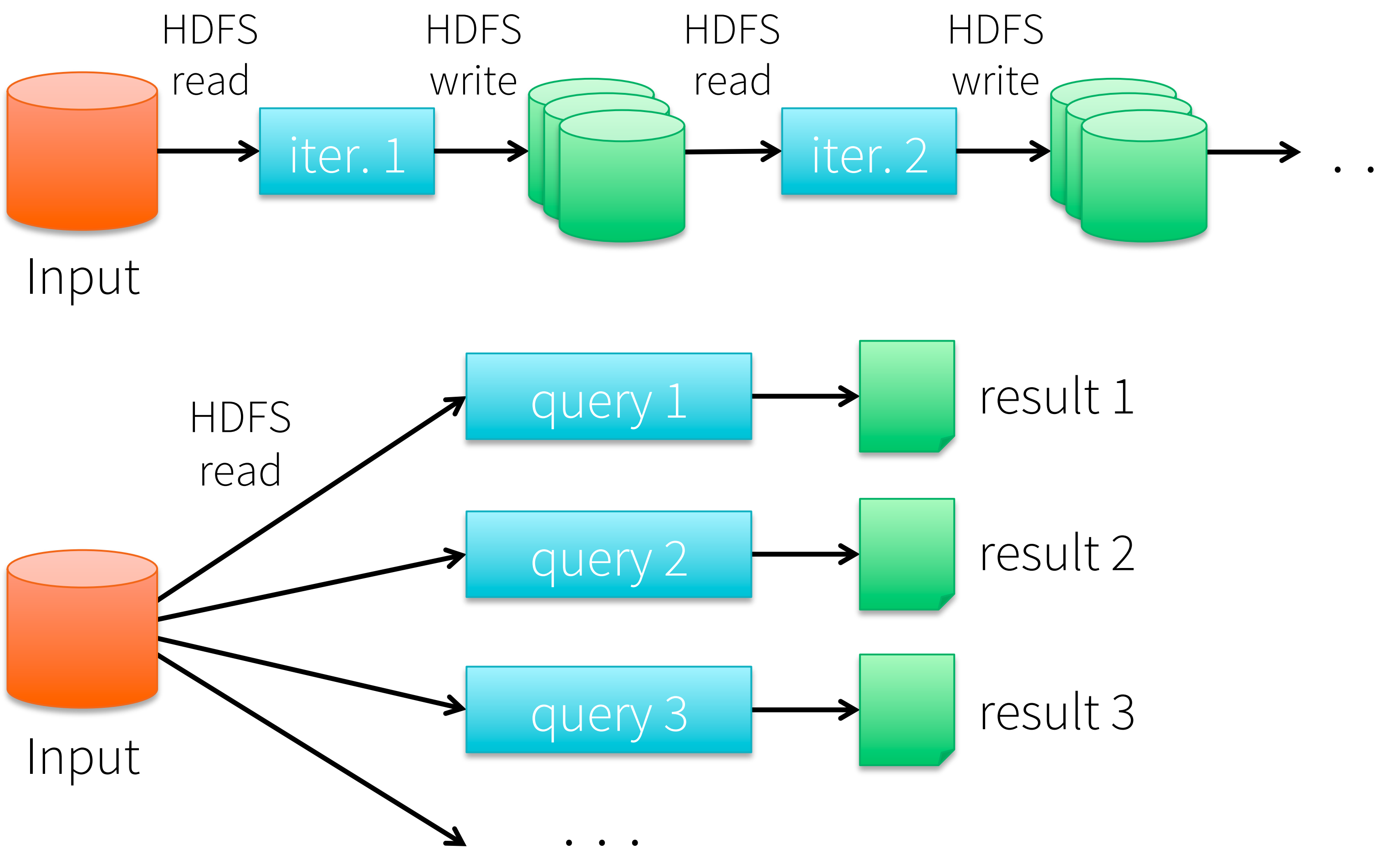
Background

Recall 3 workloads were issues for MapReduce:

- More complex, multi-pass algorithms
- More interactive ad-hoc queries
- More real-time stream processing

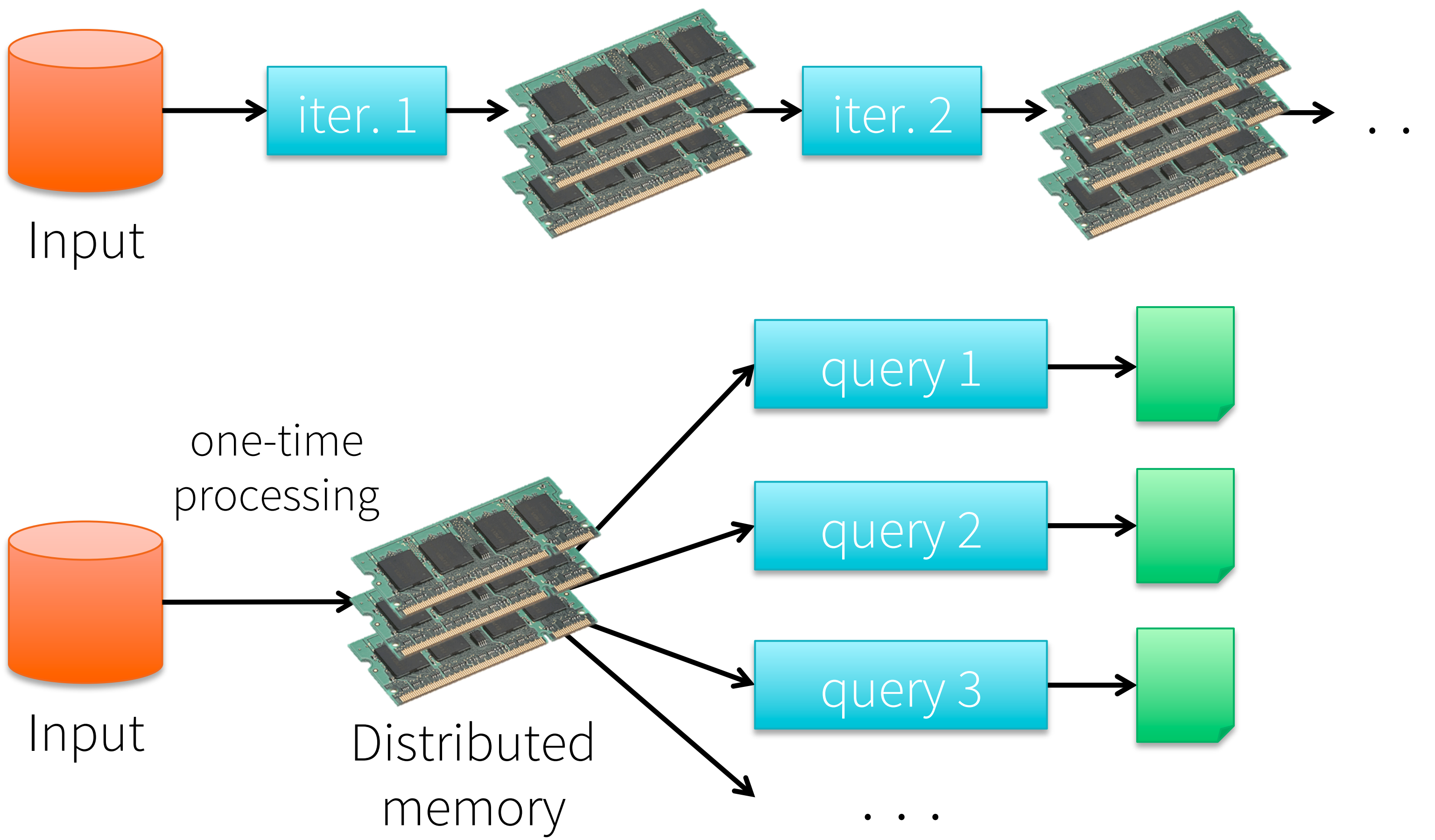
While these look different, all 3 need one thing that MapReduce lacks: efficient data sharing

Data Sharing in MapReduce



Slow due to replication and disk I/O

What We'd Like



10-100x faster than network and disk

databricks

Spark Programming Model

Resilient Distributed Datasets (RDDs)

- Collections of objects stored in RAM or disk across cluster
- Built via parallel transformations (map, filter, …)
- Automatically rebuilt on failure

Example: Log Mining

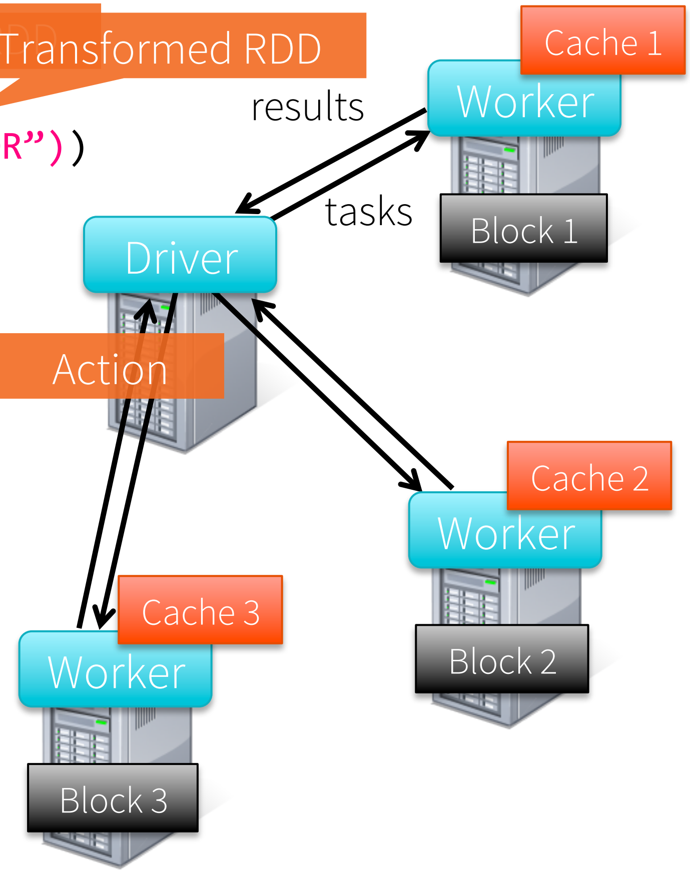
Load error messages from a log into memory, then interactively search for various patterns

```
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: "MySQL" in s).count()
messages.filter(lambda s: "Redis" in s).count()
. . .
```

Base RDD

Transformed RDD

Action

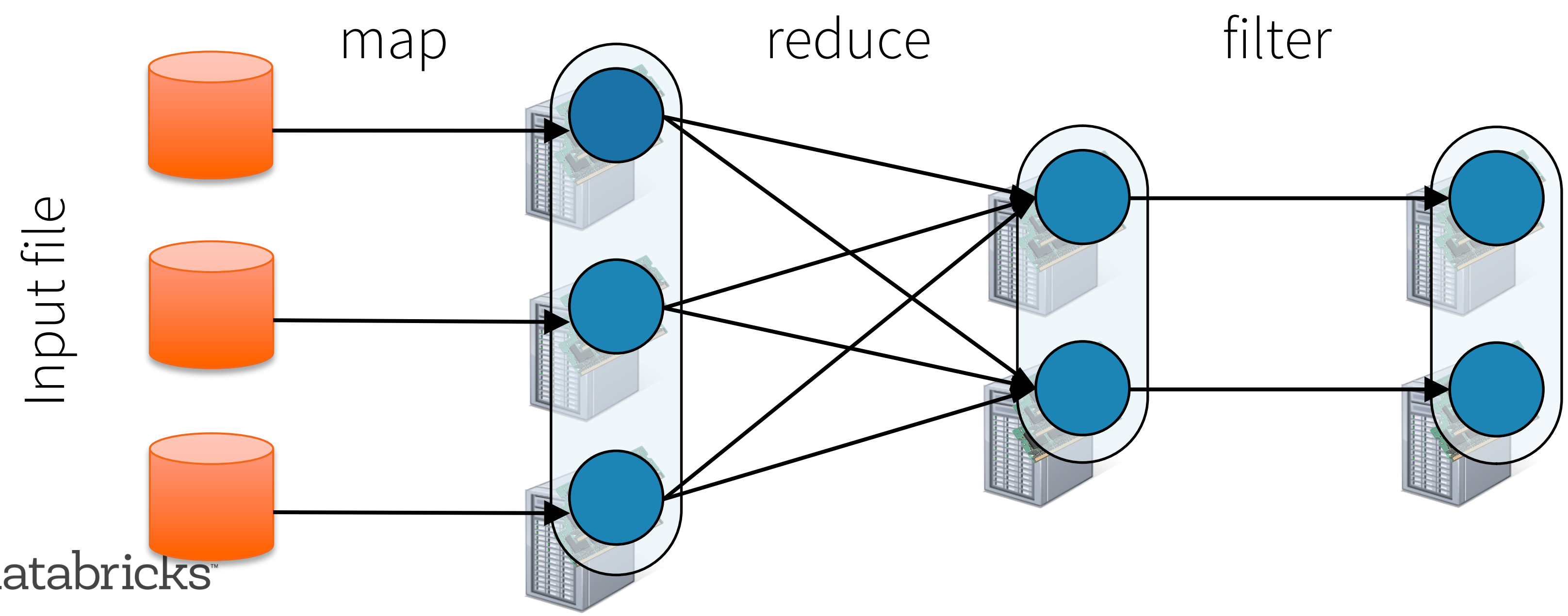


Example: full-text search of Wikipedia in 0.5 sec (vs 20s for on-disk data)

Fault Tolerance

RDDs track *lineage* info to rebuild lost data

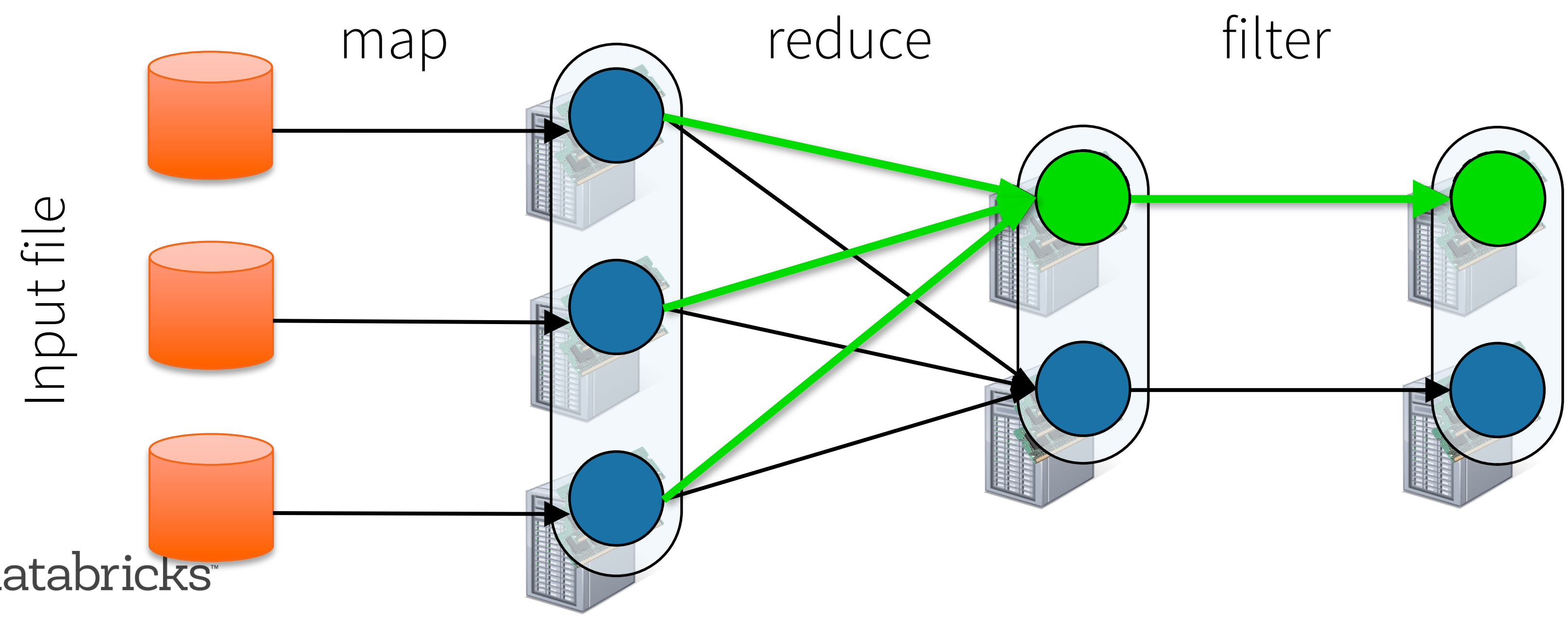
```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



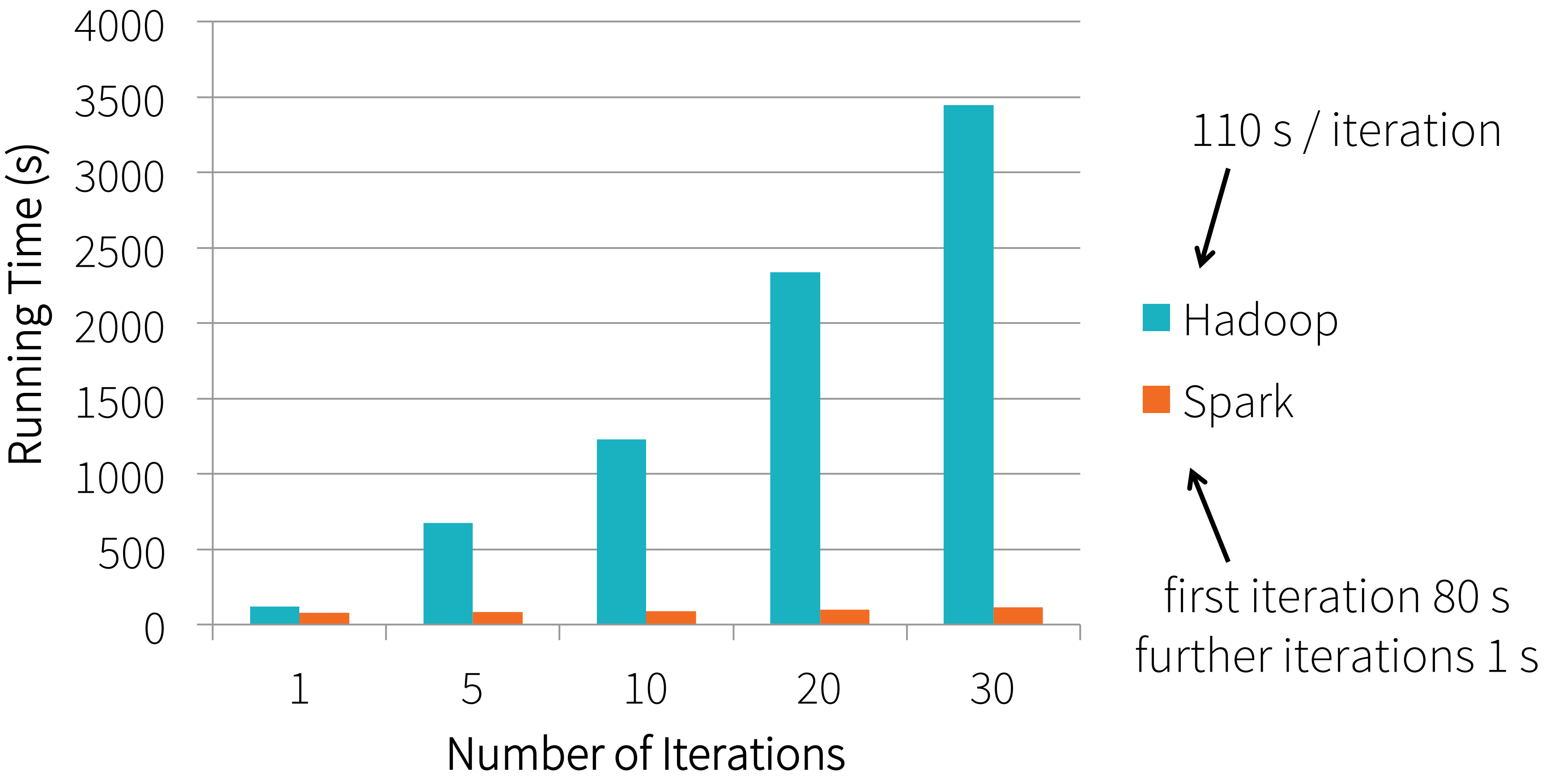
Fault Tolerance

RDDs track *lineage* info to rebuild lost data

```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



Example: Logistic Regression



databricks™

On-Disk Performance

Time to sort 100TB

2013 Record: Hadoop

2100 machines

72 minutes

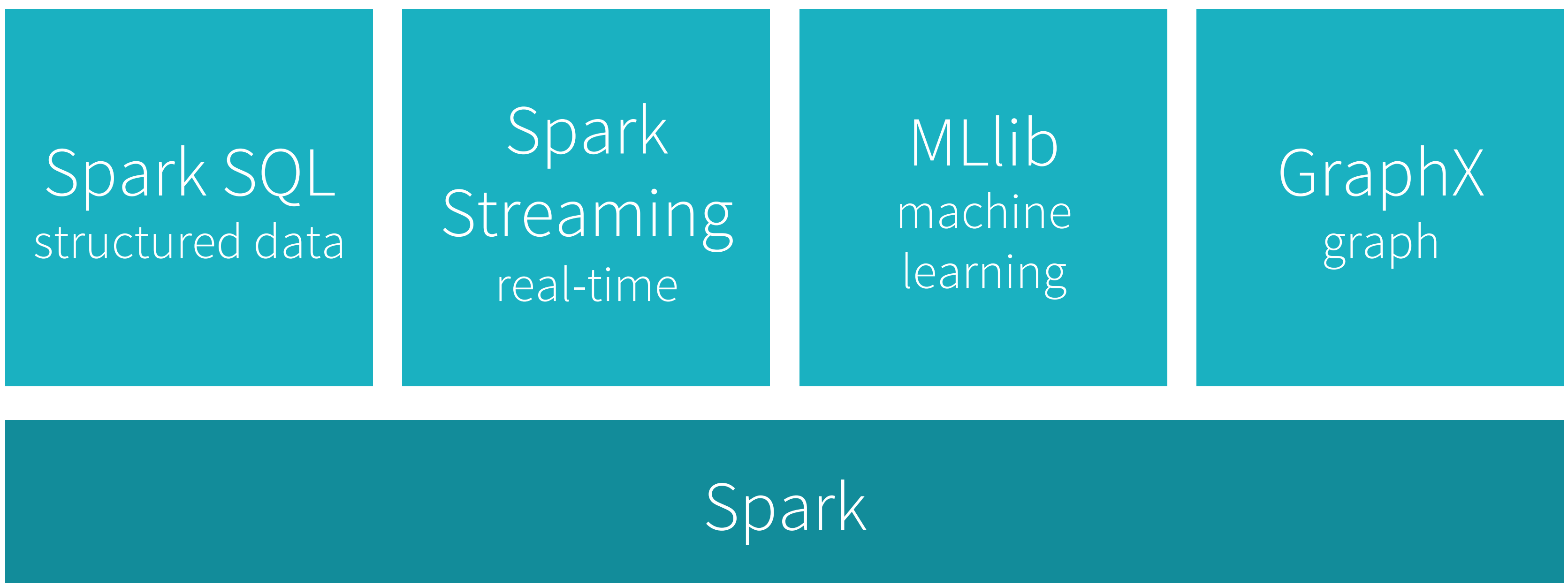
2014 Record: Spark

207 machines

23 minutes

Source: Daytona GraySort benchmark, sortbenchmark.org

Libraries Built on Spark



Combining Processing Types

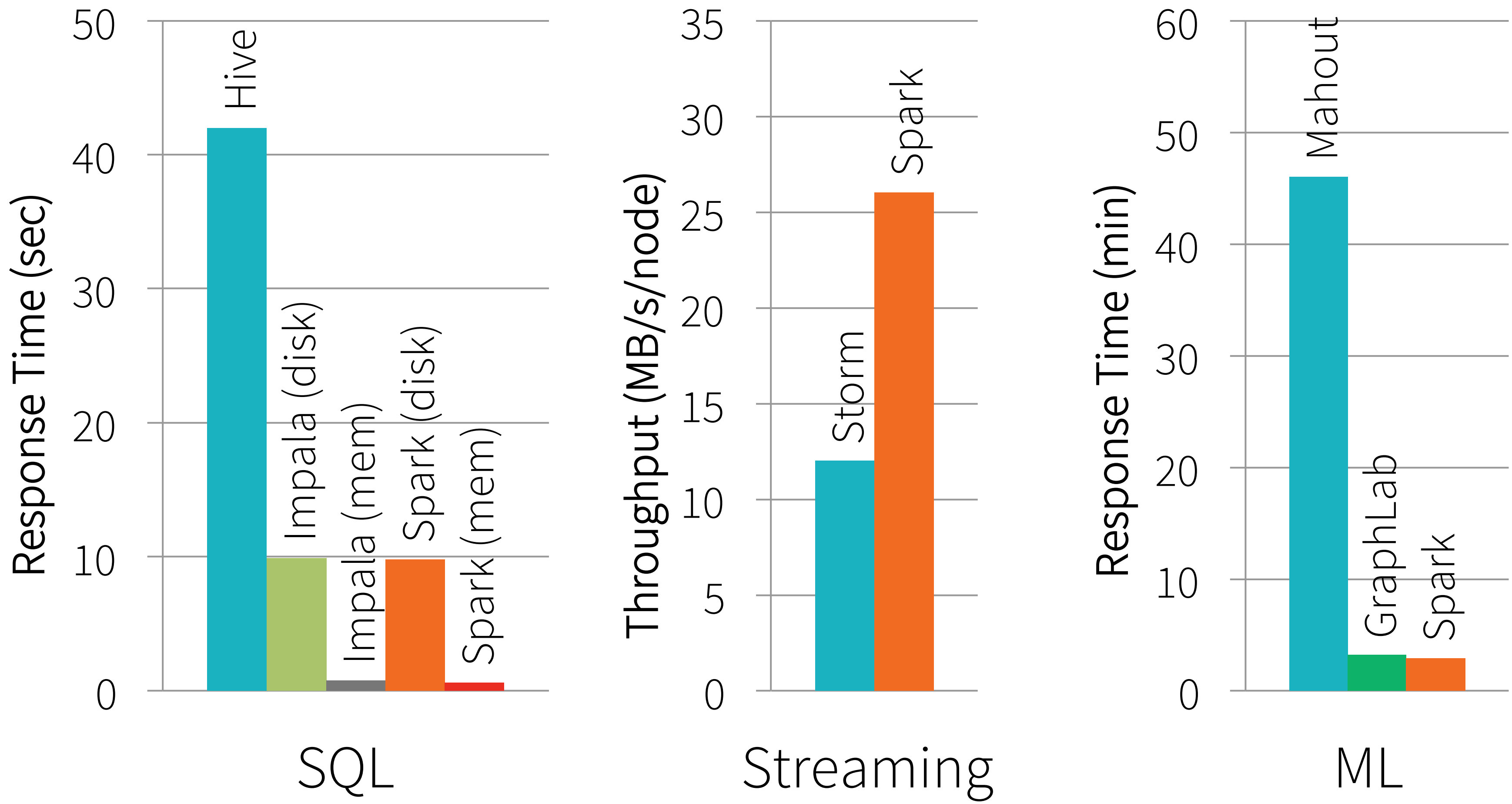
```
// Load data using SQL
points = ctx.sql("select latitude, longitude from tweets")

// Train a machine learning model
model = KMeans.train(points, 10)

// Apply it to a stream
sc.twitterStream(...)
  .map(lambda t: (model.predict(t.location), 1))
  .reduceByWindow("5s", lambda a, b: a + b)
```

Combining Processing Types

Separate systems:

Spark:

Performance vs Specialized Systems



databricks™

Some Recent Additions

DataFrame API (similar to R and Pandas)

- Easy programmatic way to work with structured data

R interface (SparkR)

Machine learning pipelines (like SciKit-learn)

Overview

Why a unified engine?

Spark programming model

Built-in libraries

Applications

Spark Community

Over 1000 deployments, clusters up to 8000 nodes

YAHOO! amazon web services Adobe IBM intel verizon

redhat ORACLE SAP ebay Goldman Sachs

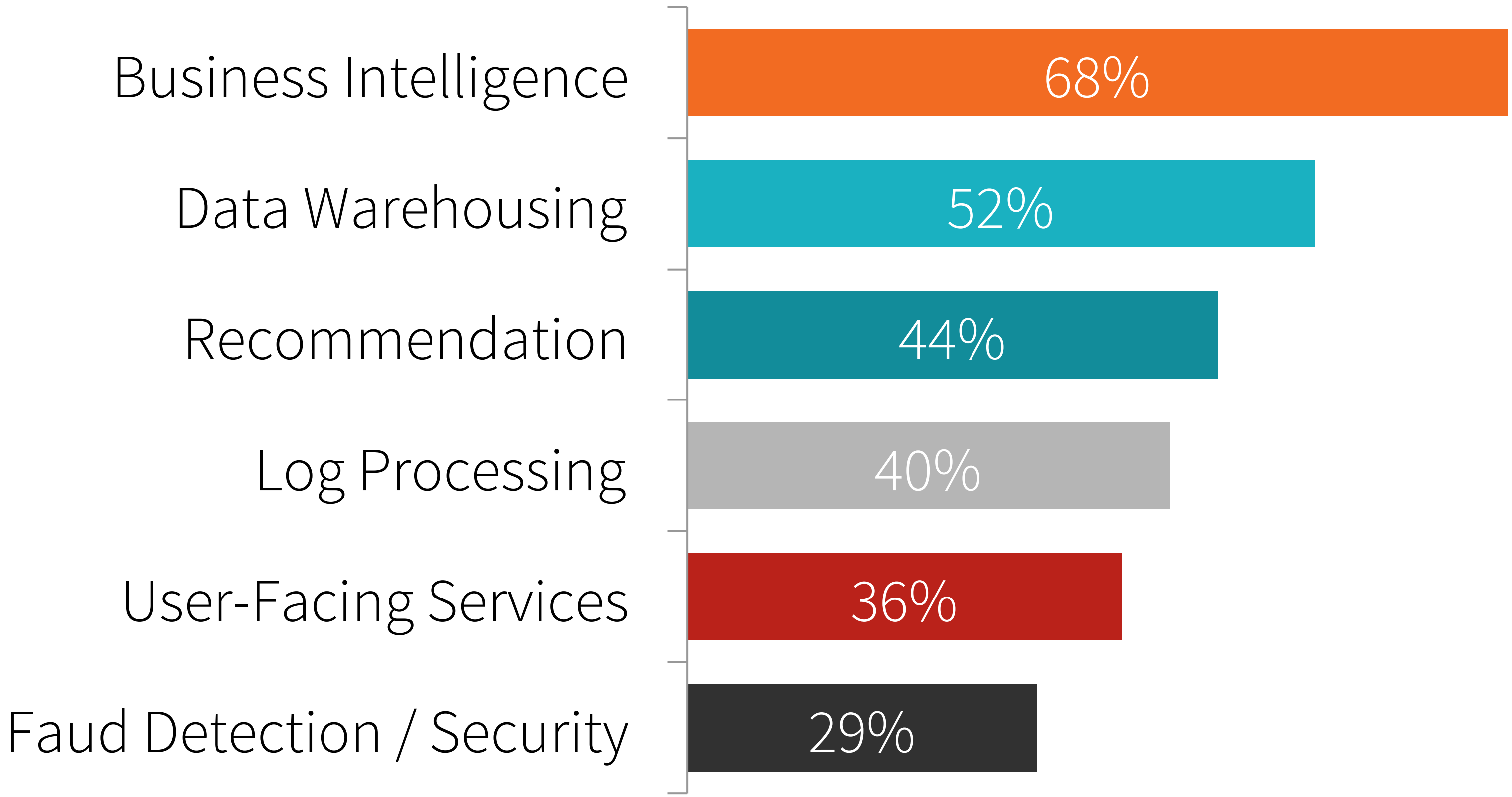
NBCUniversal 阿里巴巴 Alibaba.com NETFLIX NTT Data

CISCO HHMI janelia farm research campus Telefonica DATASTAX

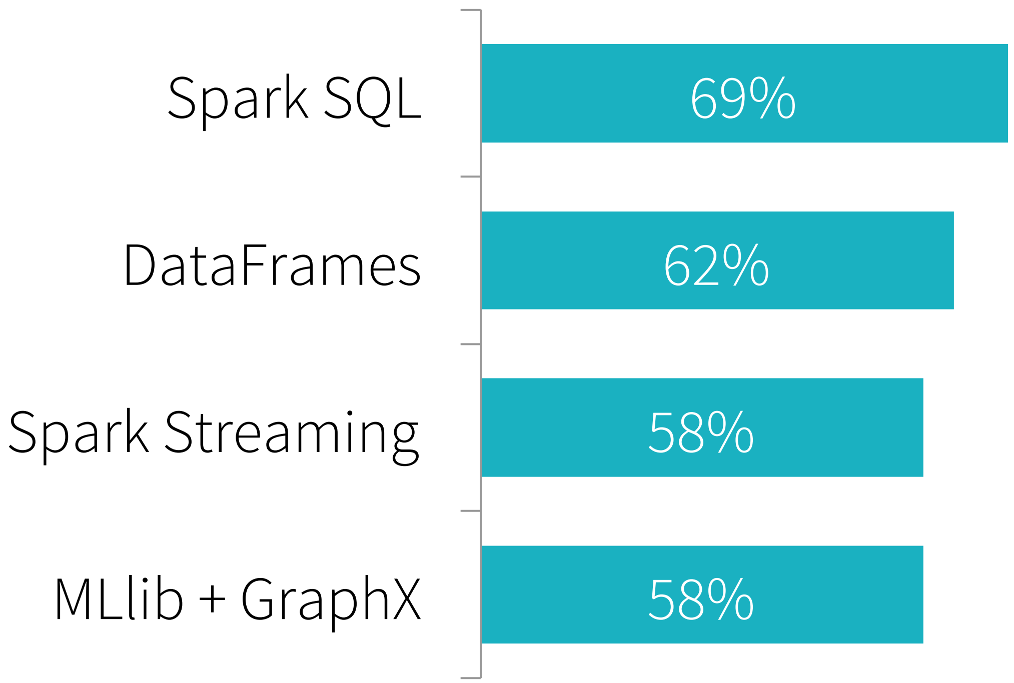
databricks Hortonworks cloudera MAPR

Many talks online at spark-summit.org

Top Applications



Spark Components Used



75%

of users use more than one component

Learn More

Get started on your laptop: spark.apache.org

Resources and MOOCs: sparkhub.databricks.com

Spark Summit: spark-summit.org

Thank You