
databricks™

Unified Big Data Processing with Apache Spark

Matei Zaharia
@matei_zaharia

What is Apache Spark?

Fast & general engine for big data processing

Generalizes MapReduce model to support more types of processing

Most active open source project in big data

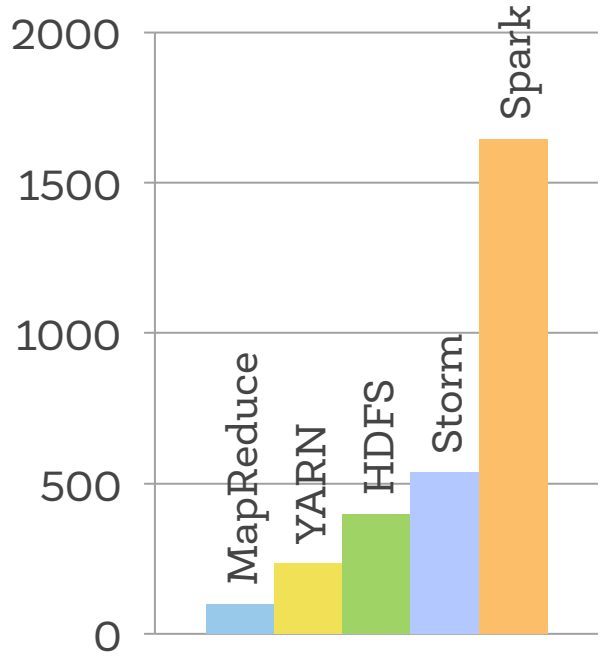
About Databricks

Founded by the creators of Spark in 2013

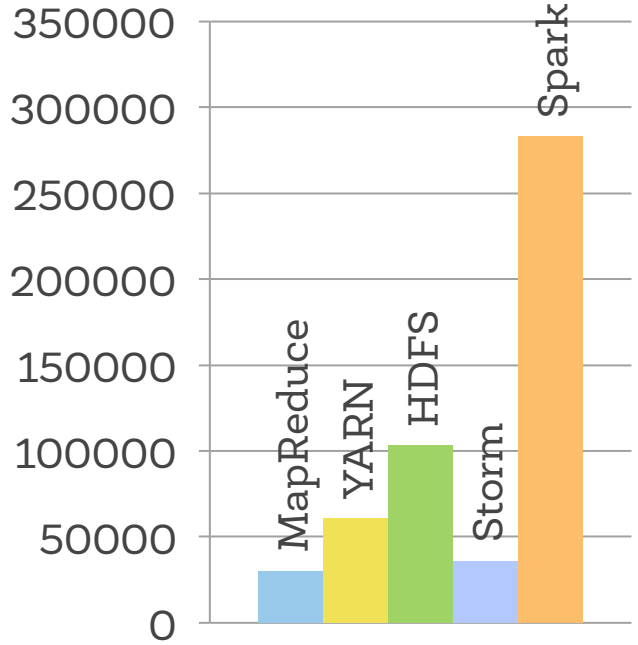
Continues to drive open source Spark development, and offers a cloud service (Databricks Cloud)

Partners to support Spark with Cloudera, MapR, Hortonworks, Datastax

Spark Community



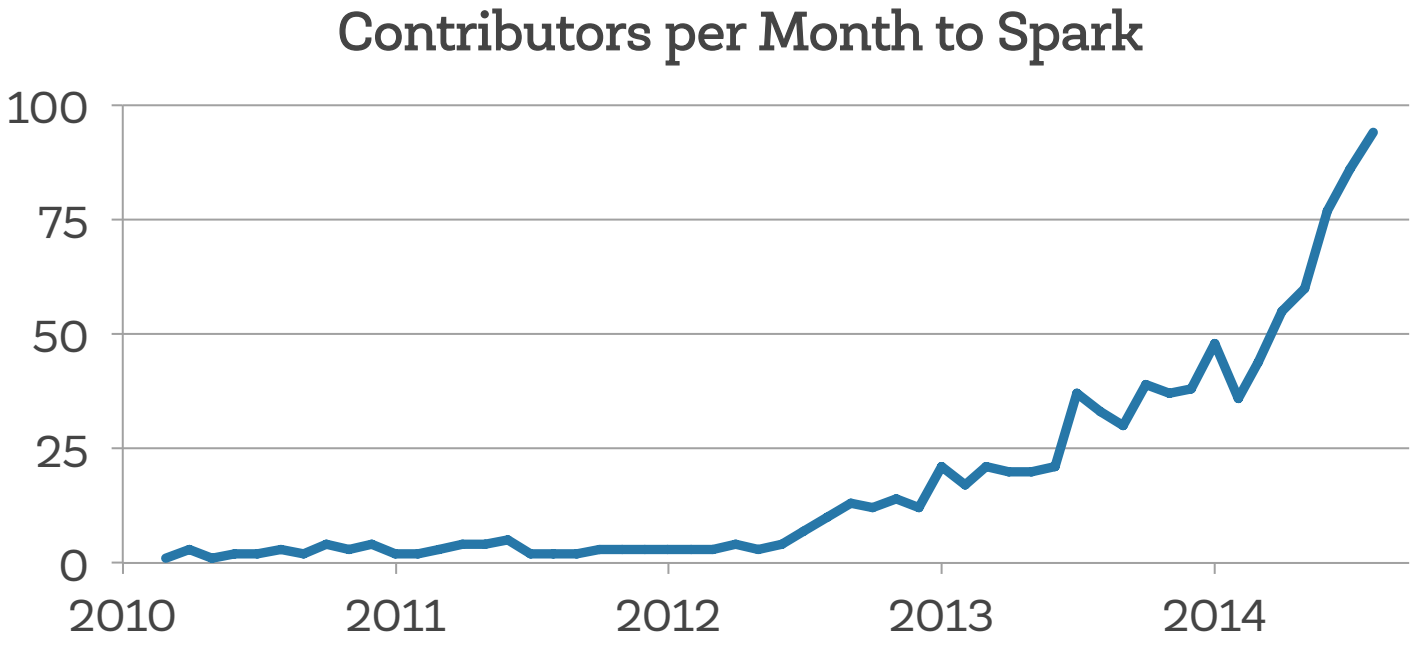
Commits



Lines of Code Changed

Activity in past 6 months

Community Growth



2-3x more activity than Hadoop, Storm, MongoDB, NumPy, D3, Julia, …

Overview

Why a unified engine?

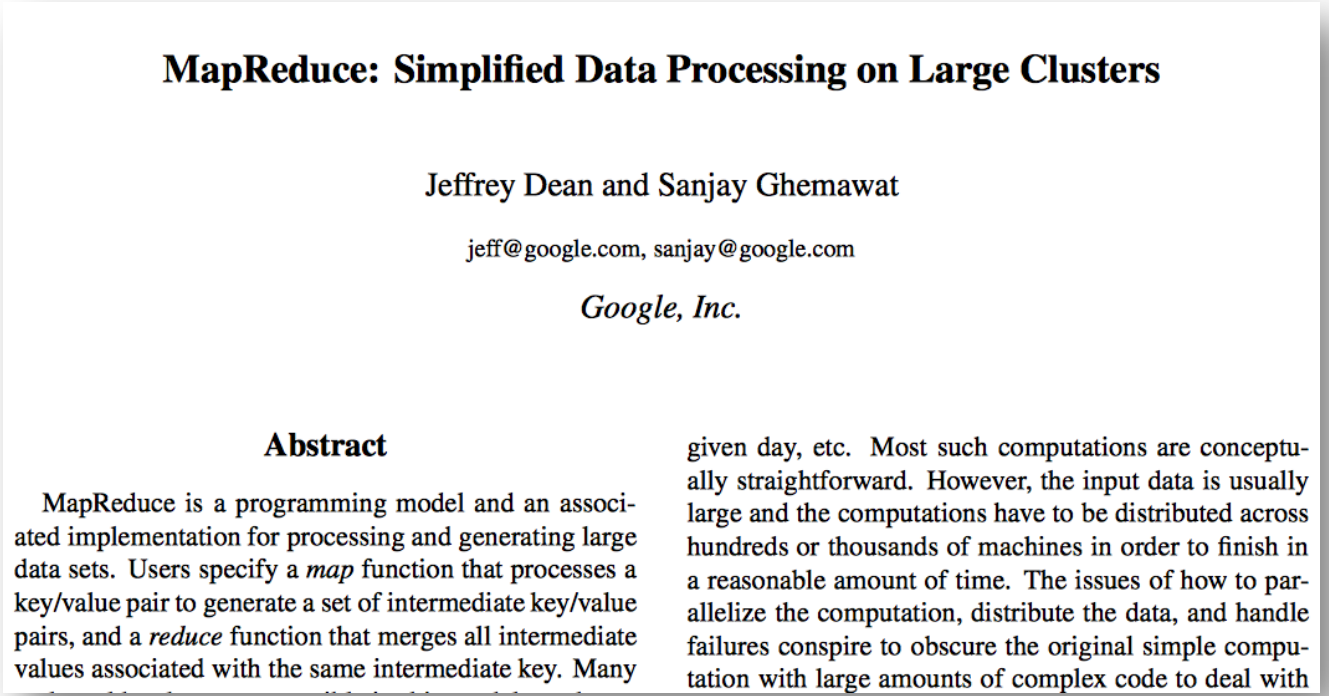
Spark execution model

Why was Spark so general?

What's next

History: Cluster Programming Models

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

databricks™

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:

databricks™

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: many *specialized* systems for these workloads

Big Data Systems Today

MapReduce

Pregel
Dremel
Giraph
Drill
Impala
Presto
Storm
S4
. . .

General batch processing

Specialized systems for new workloads

databricks™

Problems with Specialized Systems

More systems to manage, tune, deploy

Can't *combine* processing types in one application

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

In many pipelines, data exchange between engines is the dominant cost!

Big Data Systems Today

MapReduce

Pregel Dremel

Giraph Drill

Impala Presto

Storm S4 . . .

General batch processing

Specialized systems for new workloads

Unified engine

databricks™

Overview

Why a unified engine?

Spark execution model

Why was Spark so general?

What's next

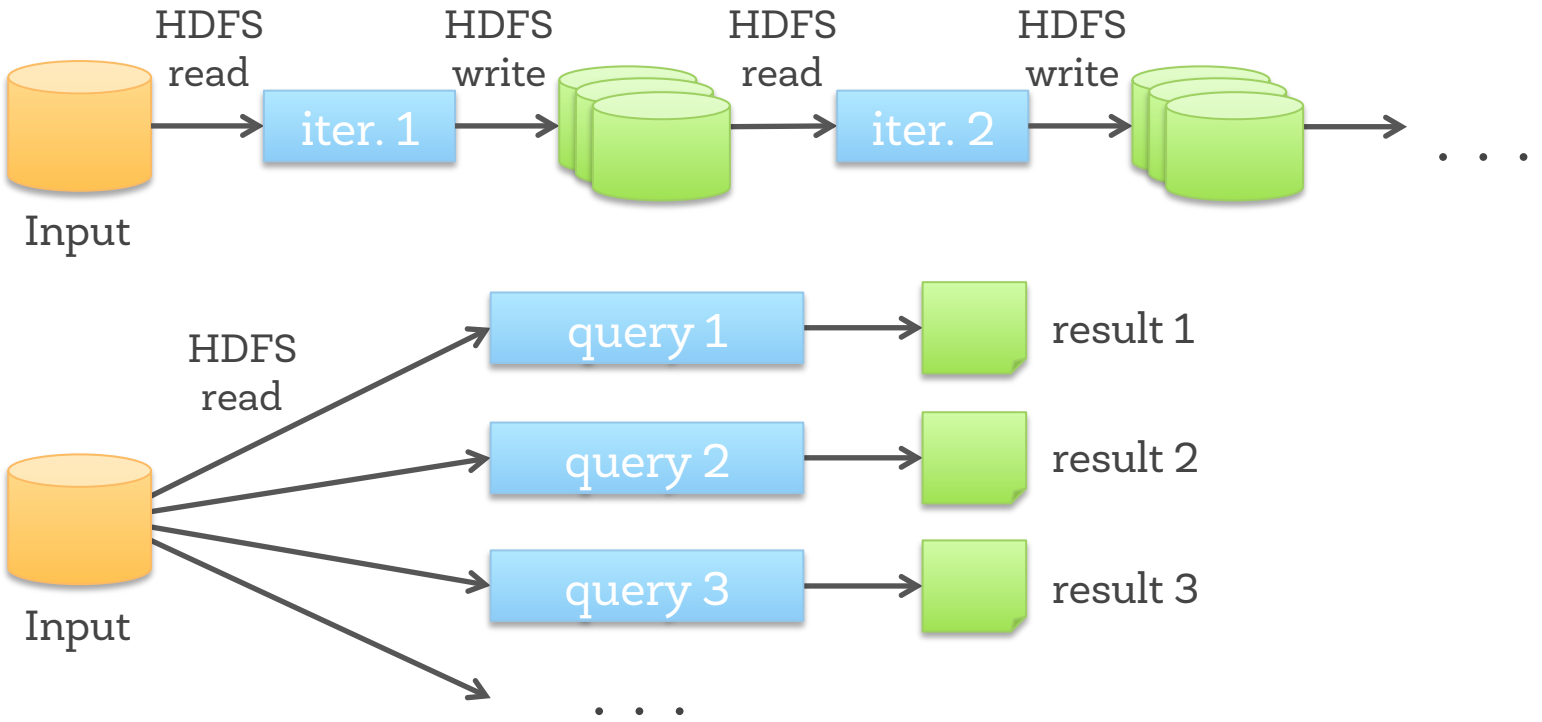
Background

Recall 3 workloads were issues for MapReduce:

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

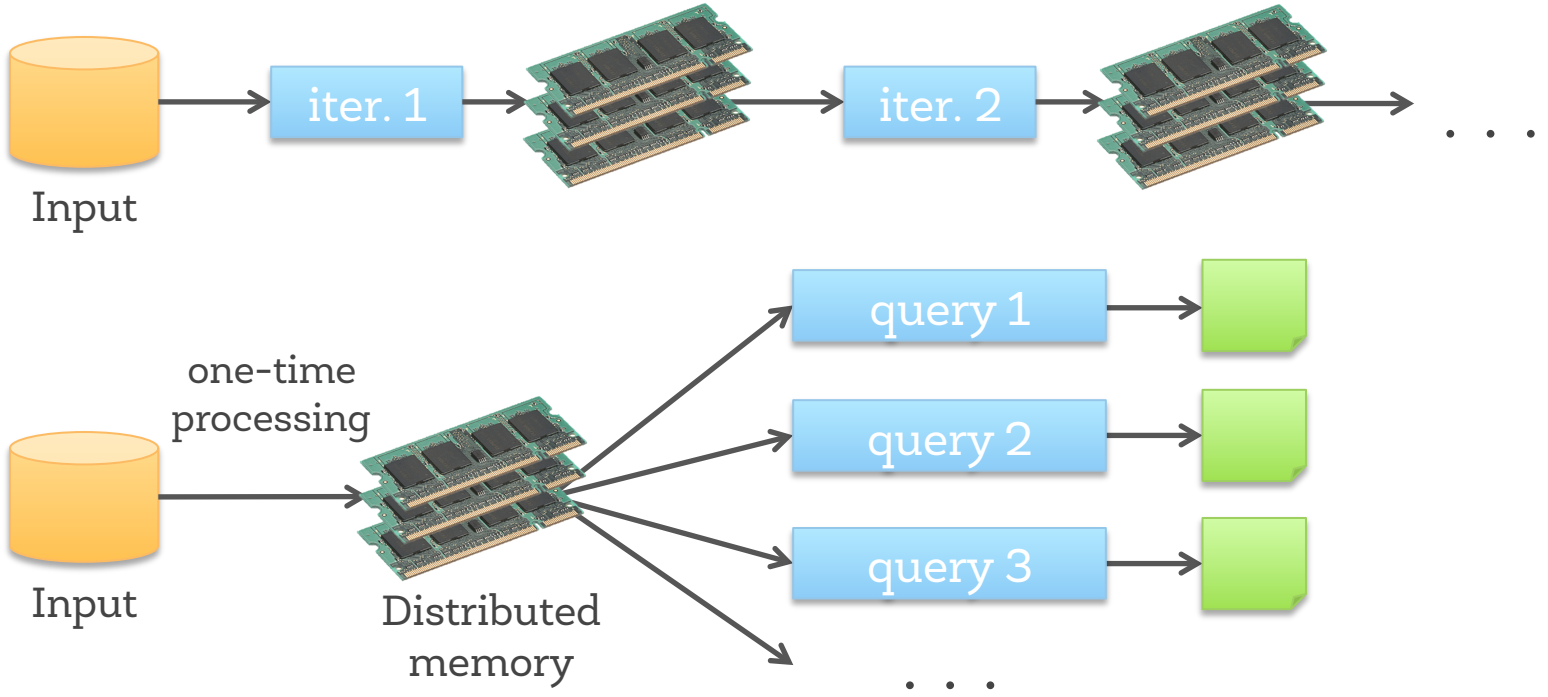
While these look different, all 3 need one thing that MapReduce lacks: efficient data sharing

Data Sharing in MapReduce



Slow due to data replication and disk I/O

What We'd Like



10-100× faster than network and disk

Spark Model

Resilient Distributed Datasets (RDDs)

- Collections of objects that can be stored in memory or disk across a cluster
- Built via parallel transformations (map, filter, …)
- Fault-tolerant *without* replication

Example: Log Mining

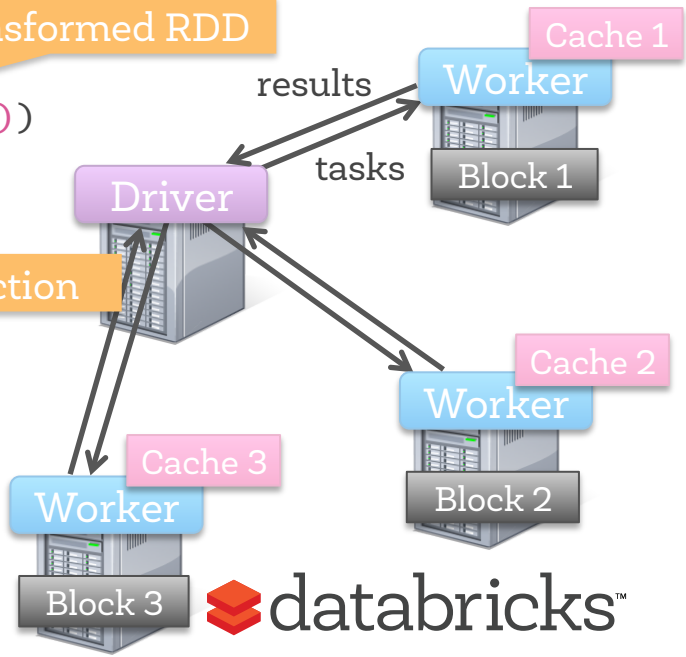
Load error messages from a log into memory, then interactively search for various patterns

```
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()
. . .
```

Base

Transformed RDD

Action

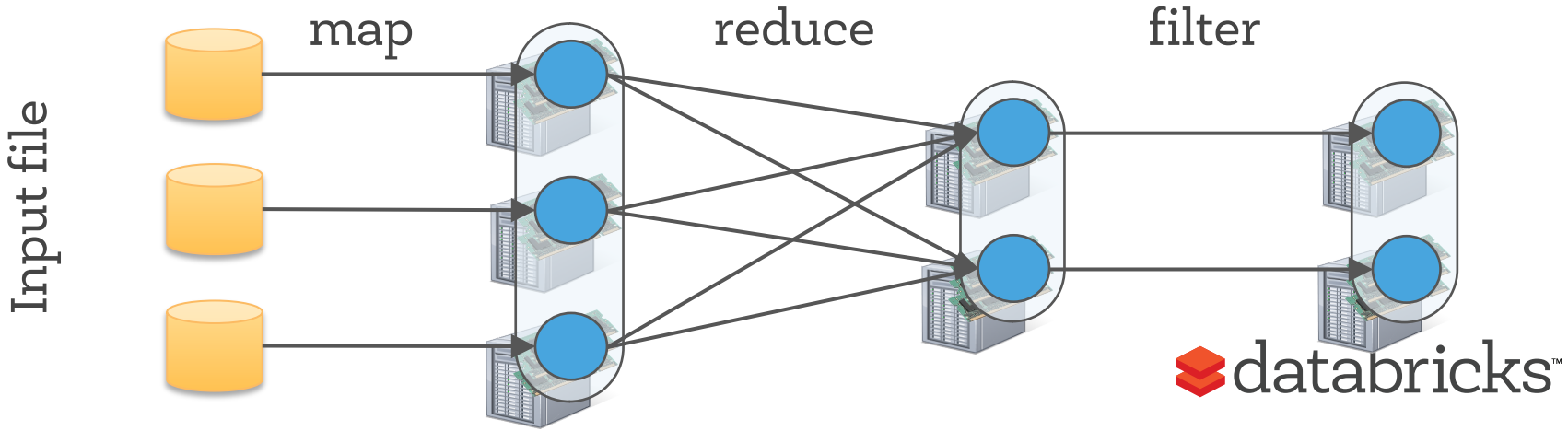


Full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

Fault Tolerance

RDDs track *lineage* info to rebuild lost data

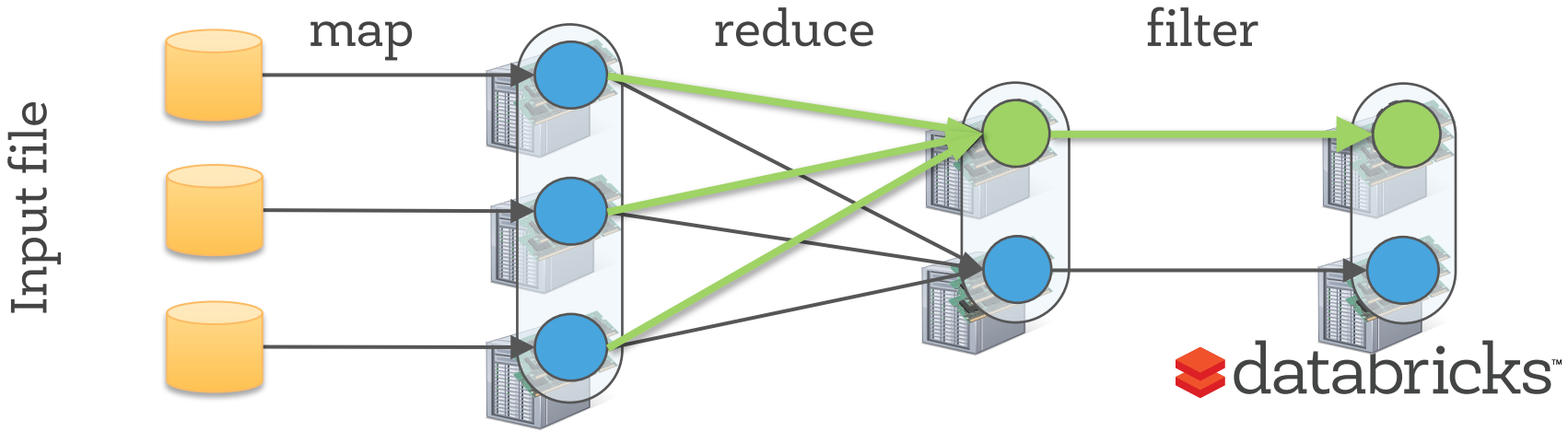
```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



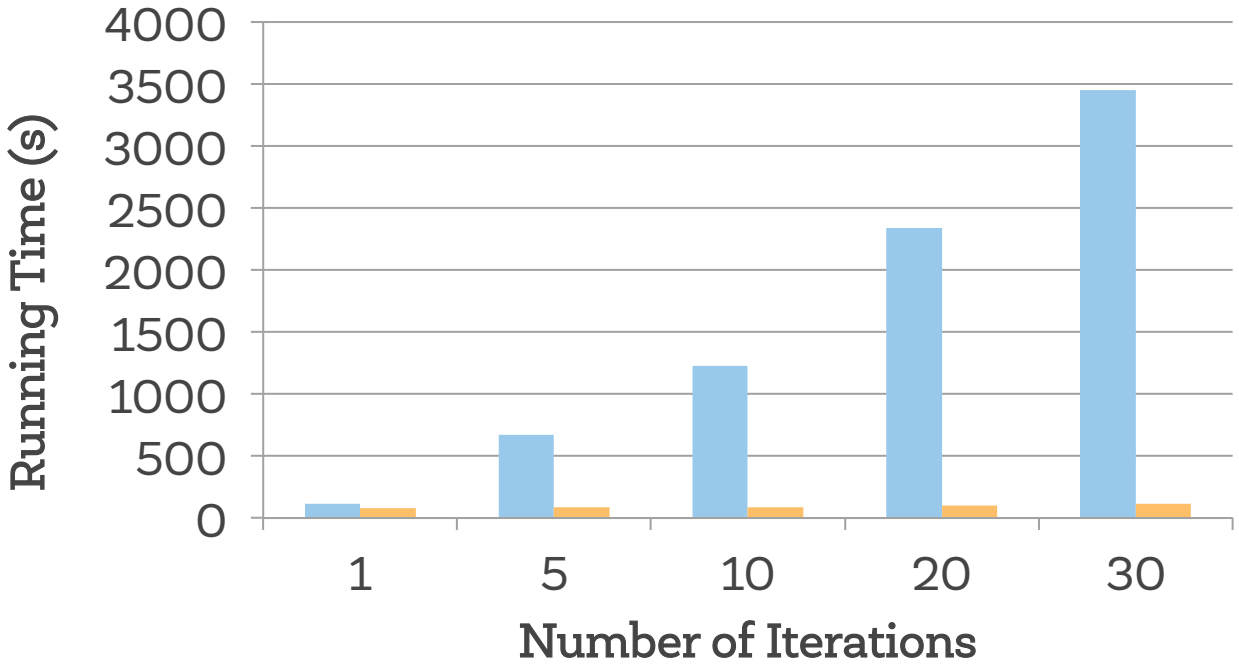
Fault Tolerance

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



110 s / iteration

Hadoop

Spark

first iteration 80 s
later iterations 1 s

databricks™

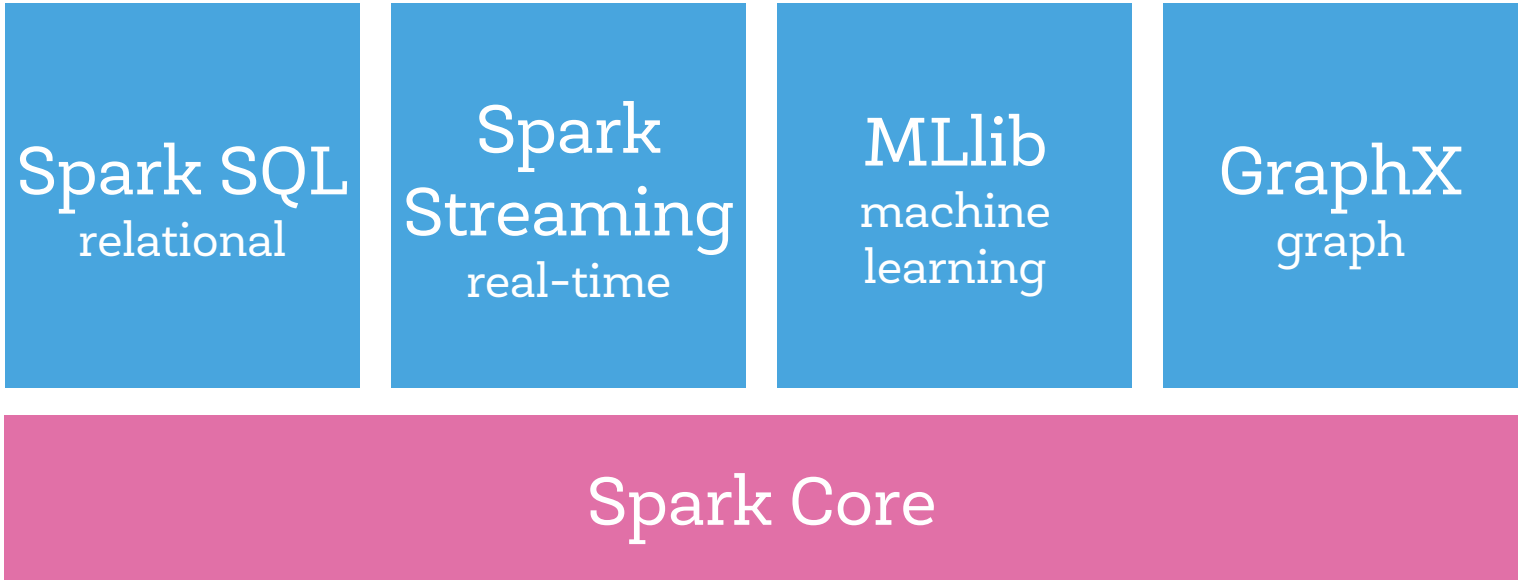
Spark in Scala and Java

```
// Scala:

val lines = sc.textFile(...)
lines.filter(s => s.contains(“ERROR”)).count()


// Java:

JavaRDD<String> lines = sc.textFile(...);
lines.filter(s -> s.contains(“ERROR”)).count();
```

databricks™

How General Is It?

Libraries Built on Spark



databricks™

Spark SQL

Represents tables as RDDs

Tables = Schema + Data

Spark SQL

Represents tables as RDDs

Tables = Schema + Data = SchemaRDD

From Hive:

```
c = HiveContext(sc)
rows = c.sql("select text, year from hivetable")
rows.filter(lambda r: r.year > 2013).collect()
```

From JSON:

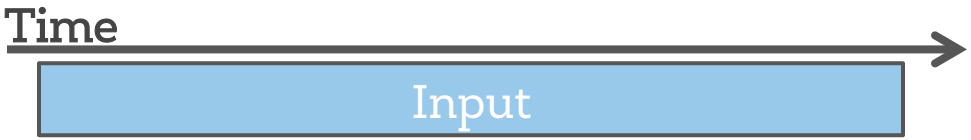
```
c.jsonFile("tweets.json").registerTempTable("tweets")
c.sql("select text, user.name from tweets")
```

tweets.json

```
{"text": "hi",
 "user": {
   "name": "matei",
   "id": 123
}}
```

databricks™

Spark Streaming



databricks™

Spark Streaming

Represents streams as a series of RDDs over time

```
val spammers = sc.sequenceFile("hdfs://spammers.seq")

sc.twitterStream(...)
  .filter(t => t.text.contains("QCon"))
  .transform(tweets => tweets.map(t => (t.user, t)).join(spammers))
  .print()
```

databricks™

MLlib

Vectors, Matrices

MLlib

Vectors, Matrices = RDD[Vector]

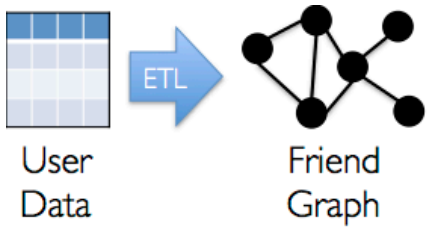
Iterative computation

```
points = sc.textFile(“data.txt”).map(parsePoint)
model = KMeans.train(points, 10)

model.predict(newPoint)
```

databricks™

GraphX

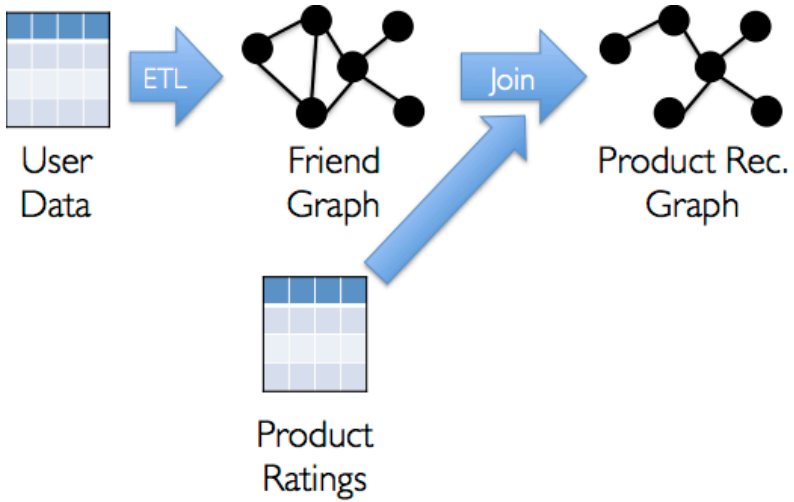
Represents graphs as RDDs of edges and vertices



databricks™

GraphX

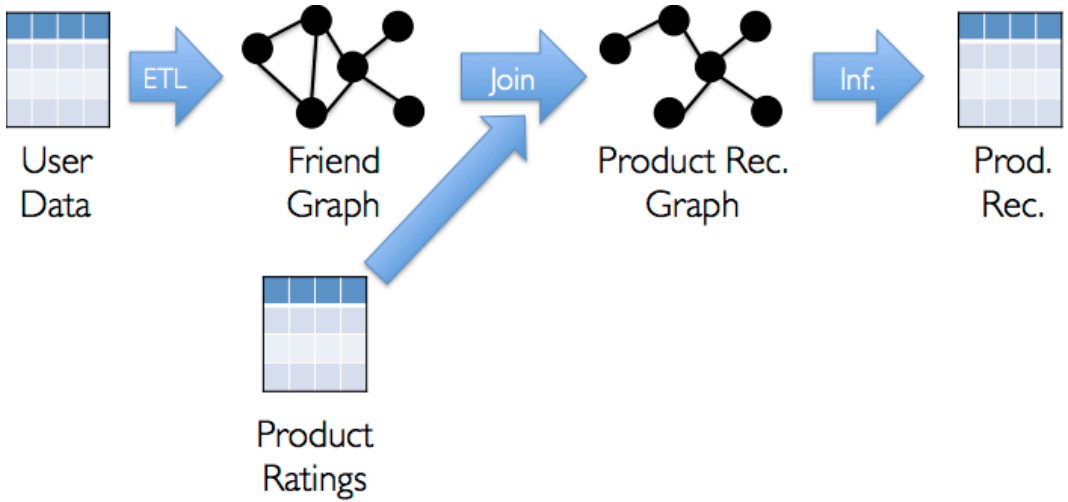
Represents graphs as RDDs of edges and vertices



databricks™

GraphX

Represents graphs as RDDs of edges and vertices



databricks™

Combining Processing Types

```
// Load data using SQL
val points = ctx.sql(
  “select latitude, longitude from historic_tweets”)

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

// Apply it to a stream
sc.twitterStream(...)
  .map(t => (model.closestCenter(t.location), 1))
  .reduceByWindow(“5s”, _ + _)
```

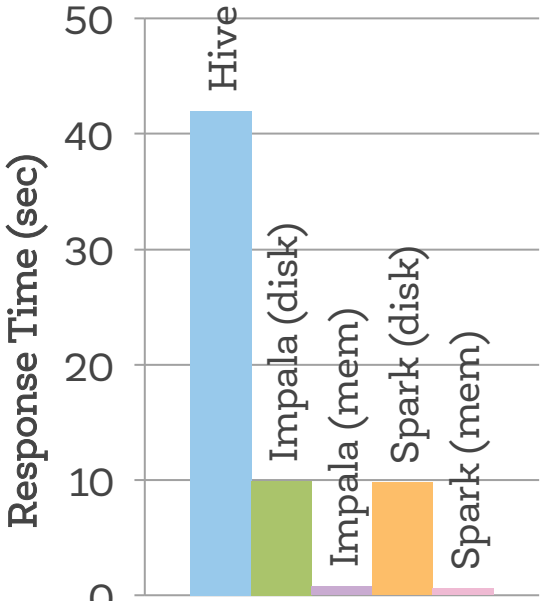
Composing Workloads

Separate systems:

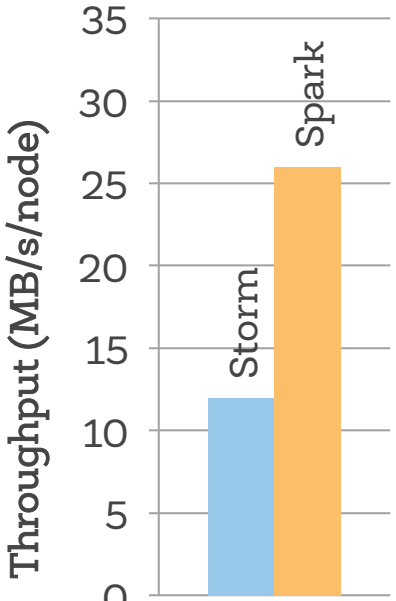
Spark:

databricks™

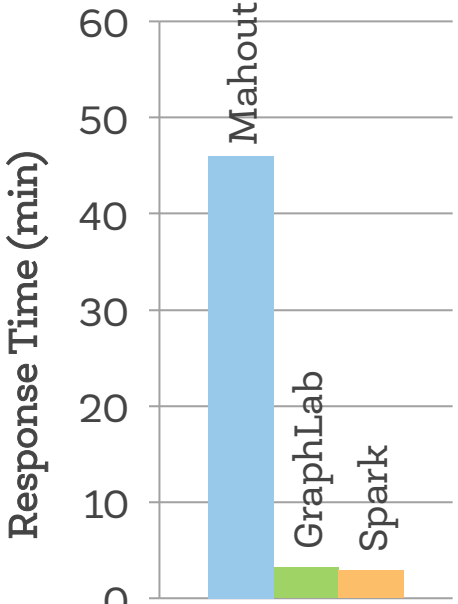
Performance vs Specialized Systems



SQL



Streaming



ML

databricks™

On-Disk Performance: Petabyte Sort

Spark beat last year's Sort Benchmark winner, Hadoop, by 3× using 10× fewer machines

	2013 Record (Hadoop)	Spark 100 TB	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Time	72 min	23 min	234 min
Nodes	2100	206	190
Cores	50400	6592	6080
Rate/Node	0.67 GB/min	20.7 GB/min	22.5 GB/min

tinyurl.com/spark-sort

databricks™

Overview

Why a unified engine?

Spark execution model

Why was Spark so general?

What's next

databricks™

Why was Spark so General?

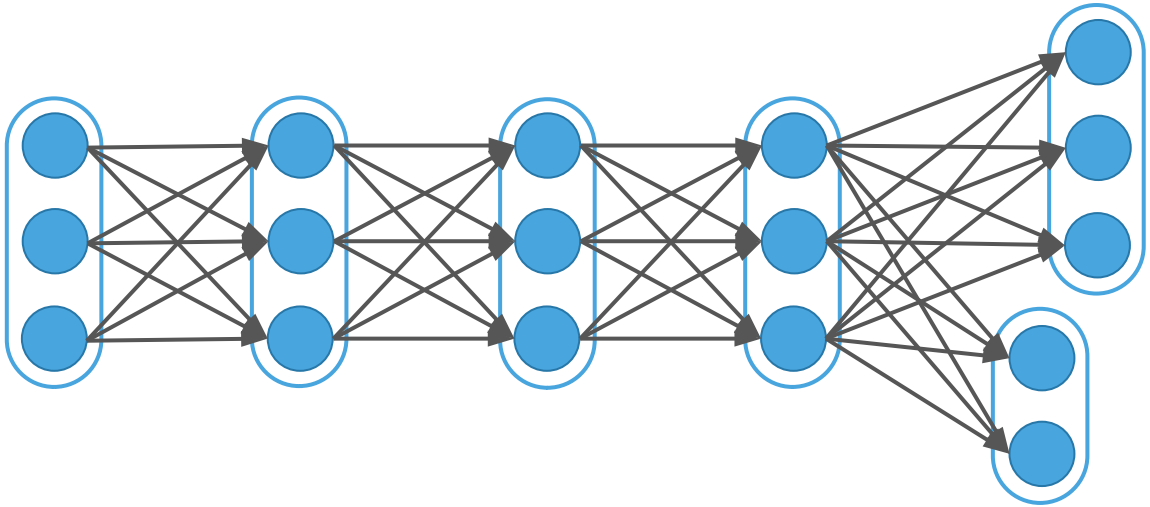
In a world of growing data complexity, understanding this can help us design new tools / pipelines

Two perspectives:

- Expressiveness perspective
- Systems perspective

1. Expressiveness Perspective

Spark ≈ MapReduce + fast data sharing



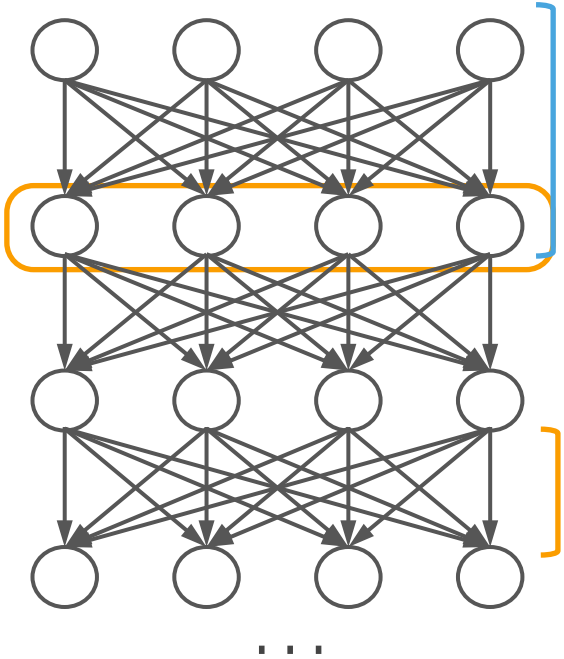
databricks™

1. Expressiveness Perspective

MapReduce can emulate *any* distributed system!

Local computation

All-to-all communication



One MR step

How to share data quickly across steps?

Spark: RDDs

How low is this latency?

Spark: ~100 ms

databricks™

2. Systems Perspective

Main bottlenecks in clusters are network and I/O

Any system that lets apps control these resources can match speed of specialized ones

In Spark:

- Users control data partitioning & caching
- We implement the data structures and algorithms of specialized systems *within* Spark records

Examples

Spark SQL

- A SchemaRDD holds records for each *chunk* of data (multiple rows), with columnar compression

GraphX

- GraphX represents graphs as an RDD of HashMaps so that it can join quickly against each partition

Result

Spark can leverage most of the latest innovations in databases, graph processing, machine learning, …

Users get a single API that composes very efficiently

More info: tinyurl.com/matei-thesis

Overview

Why a unified engine?

Spark execution model

Why was Spark so general?

What's next

What's Next for Spark

While Spark has been around since 2009, many pieces are just beginning

300 contributors, 2 whole libraries new this year

Big features in the works

Spark 1.2 (Coming in Dec)

New machine learning pipelines API

- Featurization & parameter search, similar to SciKit-Learn

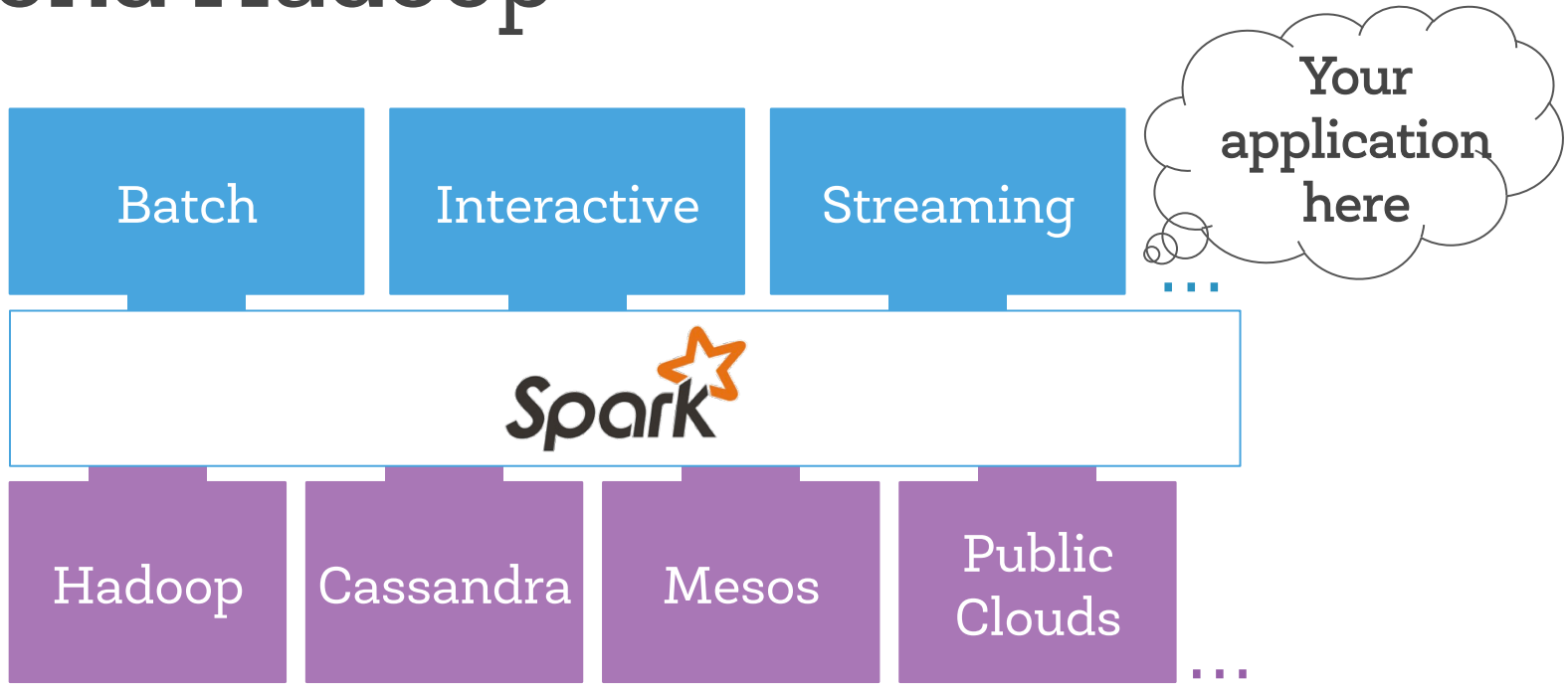
Python API for Spark Streaming

Spark SQL pluggable data sources

- Hive, JSON, Parquet, Cassandra, ORC, …

Scala 2.11 support

Beyond Hadoop



Unified API across workloads, storage systems and environments

Learn More

Downloads and tutorials: spark.apache.org

Training: databricks.com/training (free videos)

Databricks Cloud: databricks.com/cloud


Spark Summit East 2015
Spark Summit 2015
New York March 18-19, 2015
San Francisco June 15-17, 2015
www.spark-summit.org


databricks™