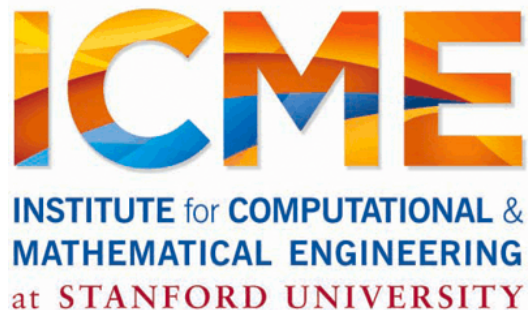


# Spark and the Big Data Library

Reza Zadeh



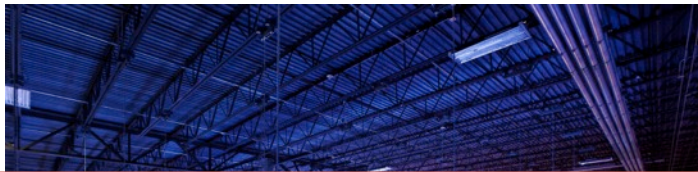
Thanks to Matei Zaharia

# Problem

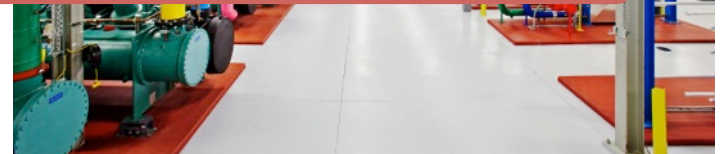
Data growing faster than processing speeds

Only solution is to parallelize on large clusters

» Wide use in both enterprises and web industry



How do we program these things?



# Outline

Data flow vs. traditional network programming

Limitations of MapReduce

Spark computing engine

Machine Learning Example

Current State of Spark Ecosystem

Built-in Libraries

Data flow vs. traditional network programming

# Traditional Network Programming

Message-passing between nodes (e.g. MPI)

**Very difficult** to do at scale:

- » How to split problem across nodes?
  - Must consider network & data locality
- » How to deal with failures? (inevitable at scale)
- » Even worse: stragglers (node not failed, but slow)
- » Ethernet networking not fast
- » Have to write programs for each machine

Rarely used in commodity datacenters

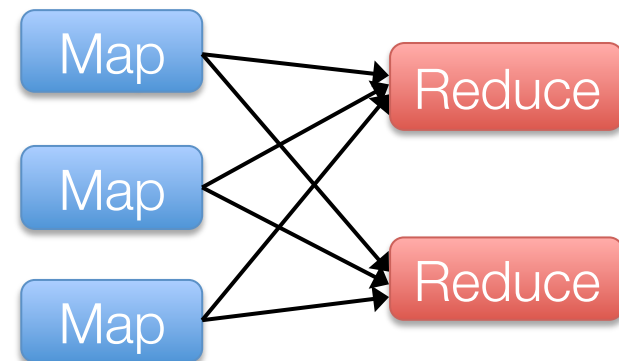
# Data Flow Models

Restrict the programming interface so that the system can do more automatically

Express jobs as graphs of high-level operators

- » System picks how to split each operator into tasks and where to run each task
- » Run parts twice fault recovery

Biggest example: MapReduce



# Example MapReduce Algorithms

Matrix-vector multiplication

Power iteration (e.g. PageRank)

Gradient descent methods

Stochastic SVD

Tall skinny QR

Many others!

# Why Use a Data Flow Engine?

Ease of programming

- » High-level functions instead of message passing

Wide deployment

- » More common than MPI, especially “near” data

Scalability to very largest clusters

- » Even HPC world is now concerned about resilience

Examples: Pig, Hive, Scalding, Storm



# Limitations of MapReduce

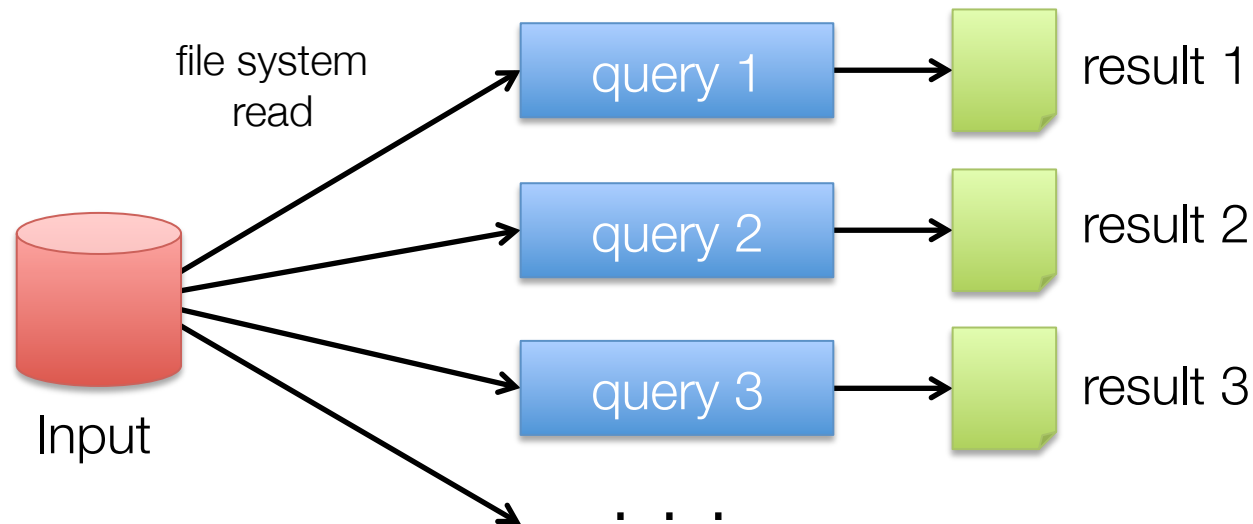
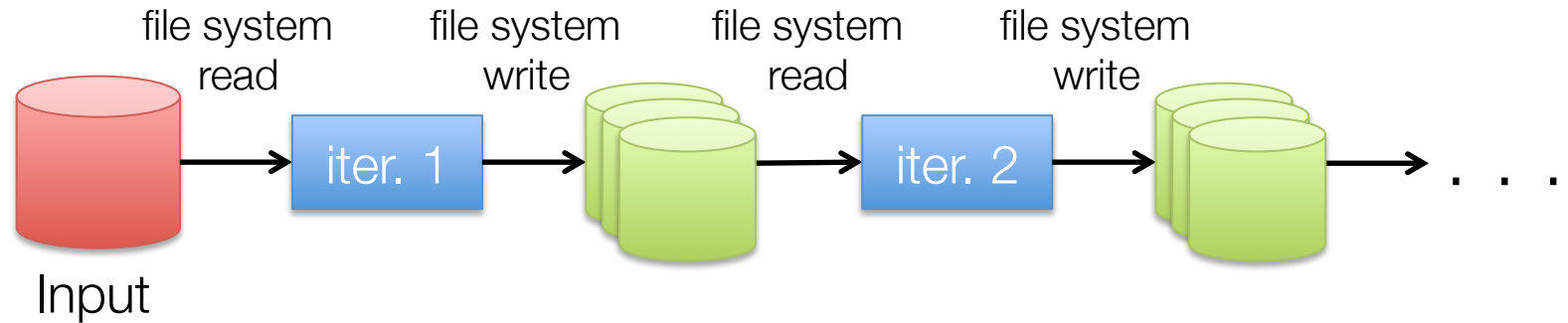
# Limitations of MapReduce

MapReduce is great at one-pass computation, but inefficient for *multi-pass* algorithms

No efficient primitives for data sharing

- » State between steps goes to distributed file system
- » Slow due to replication & disk storage

# Example: Iterative Apps

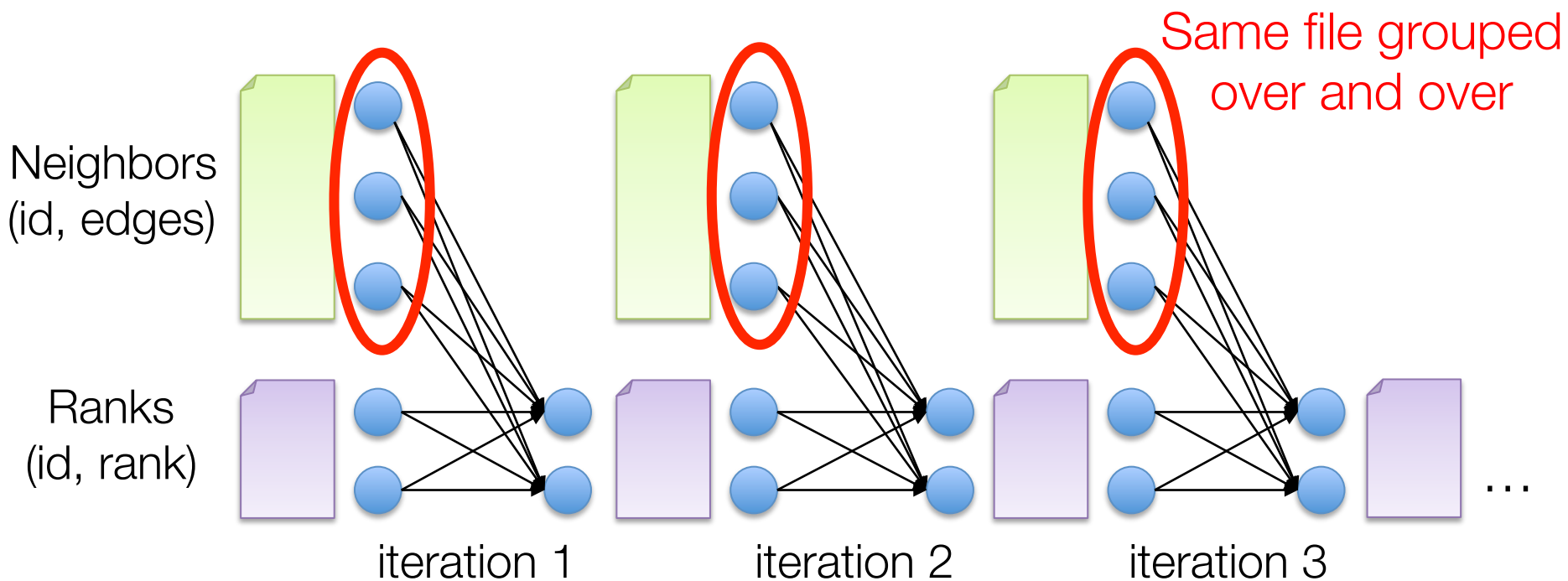


Commonly spend 90% of time doing I/O

# Example: PageRank

Repeatedly multiply sparse matrix and vector

Requires repeatedly hashing together page adjacency lists and rank vector



# Result

While MapReduce is simple, it can require *asymptotically* more communication or I/O

Spark computing engine

# Spark Computing Engine

Extends a programming language with a distributed collection data-structure

» “Resilient distributed datasets” (RDD)

Open source at Apache

» Most active community in big data, with 50+ companies contributing

Clean APIs in Java, Scala, Python, R

# Resilient Distributed Datasets (RDDs)

## Main idea: Resilient Distributed Datasets

- » Immutable collections of objects, spread across cluster
- » Statically typed: `RDD[T]` has objects of type `T`

```
val sc = new SparkContext()
val lines = sc.textFile("log.txt") // RDD[String]
```

```
// Transform using standard collection operations
```

```
val errors = lines.filter(_.startsWith("ERROR"))
```

```
val messages = errors.map(_.split('\t')(2))
```

→ lazily evaluated

```
messages.saveAsTextFile("errors.txt")
```

→ kicks off a computation



# Key Idea

## Resilient Distributed Datasets (RDDs)

- » Collections of objects across a cluster with user controlled partitioning & storage (memory, disk, ...)
- » Built via parallel transformations (map, filter, ...)
- » The world only lets you make make RDDs such that they can be:

Automatically rebuilt on failure

# Python, Java, Scala, R

// scala:

```
val lines = sc.textFile(...)
lines.filter(x => x.contains("ERROR")).count()
```

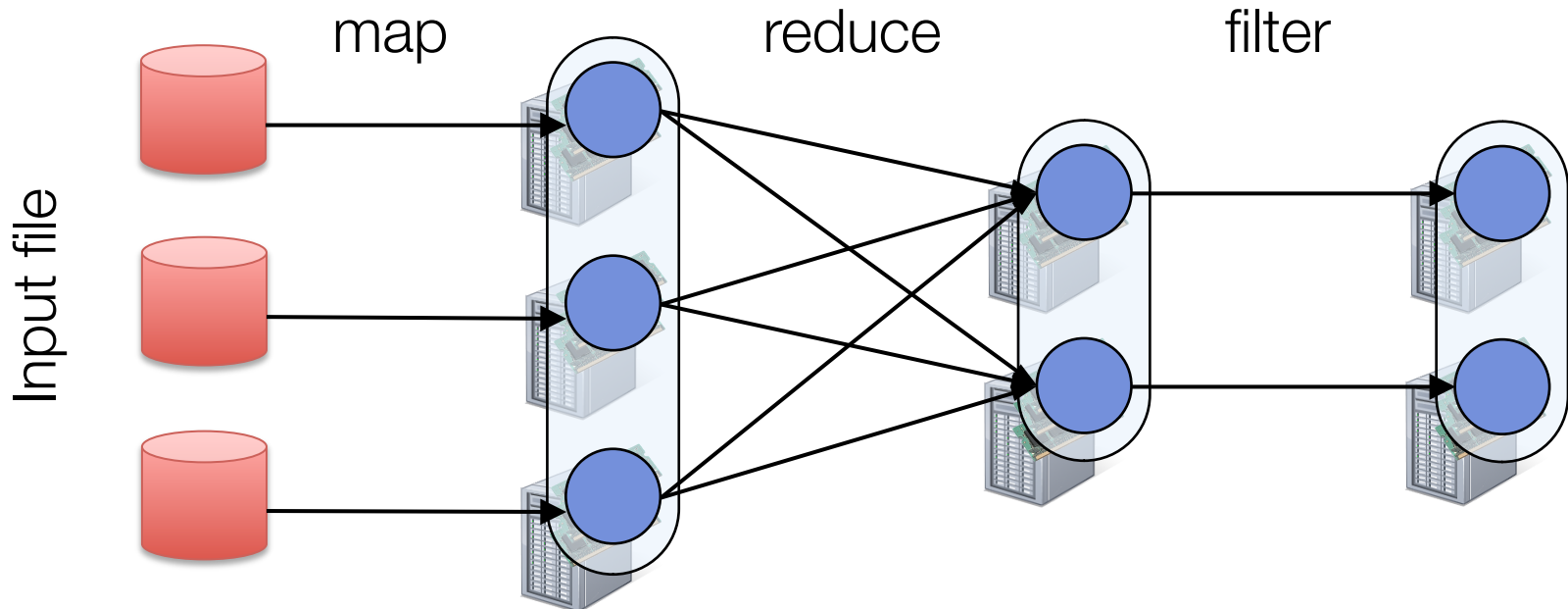
// Java:

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
    Boolean call(String s) {
        return s.contains("error");
    }
}).count();
```

# Fault Tolerance

RDDs track *lineage* info to rebuild lost data

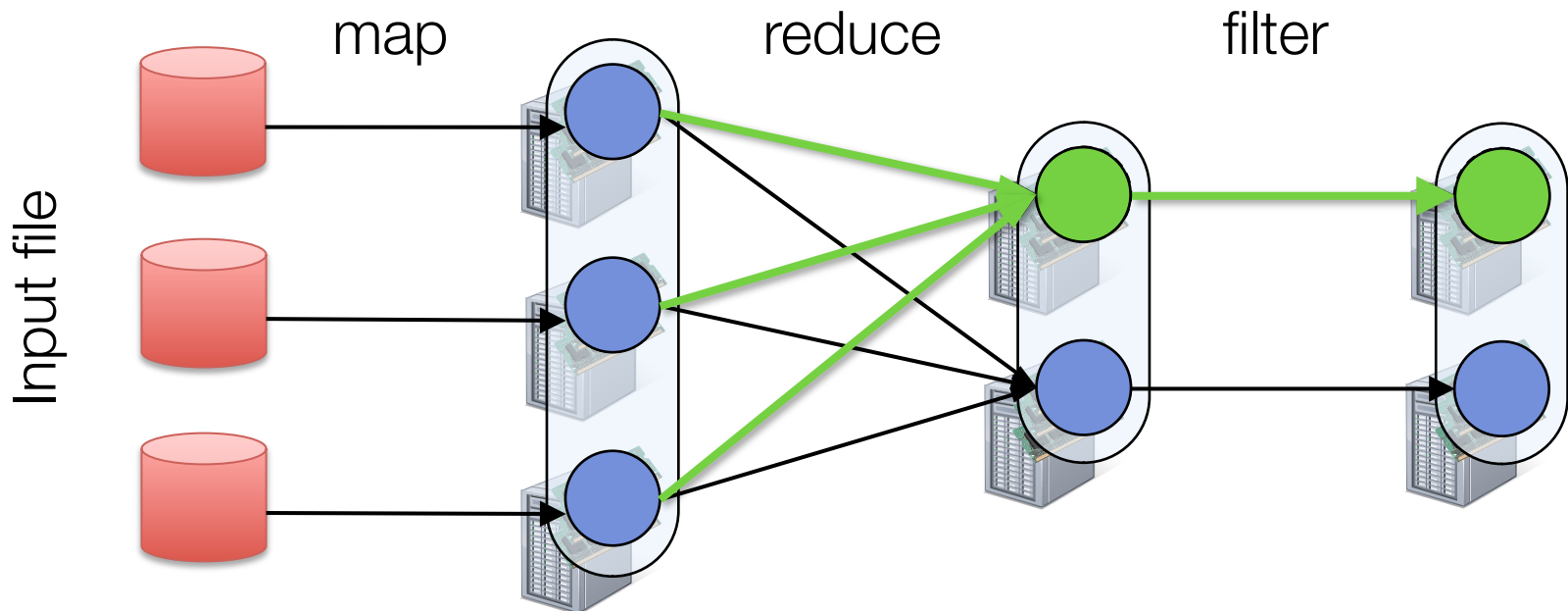
```
file.map(lambda rec: (rec.type, 1))  
    .reduceByKey(lambda x, y: x + y)  
    .filter(lambda (type, count): count > 10)
```



# 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)
```

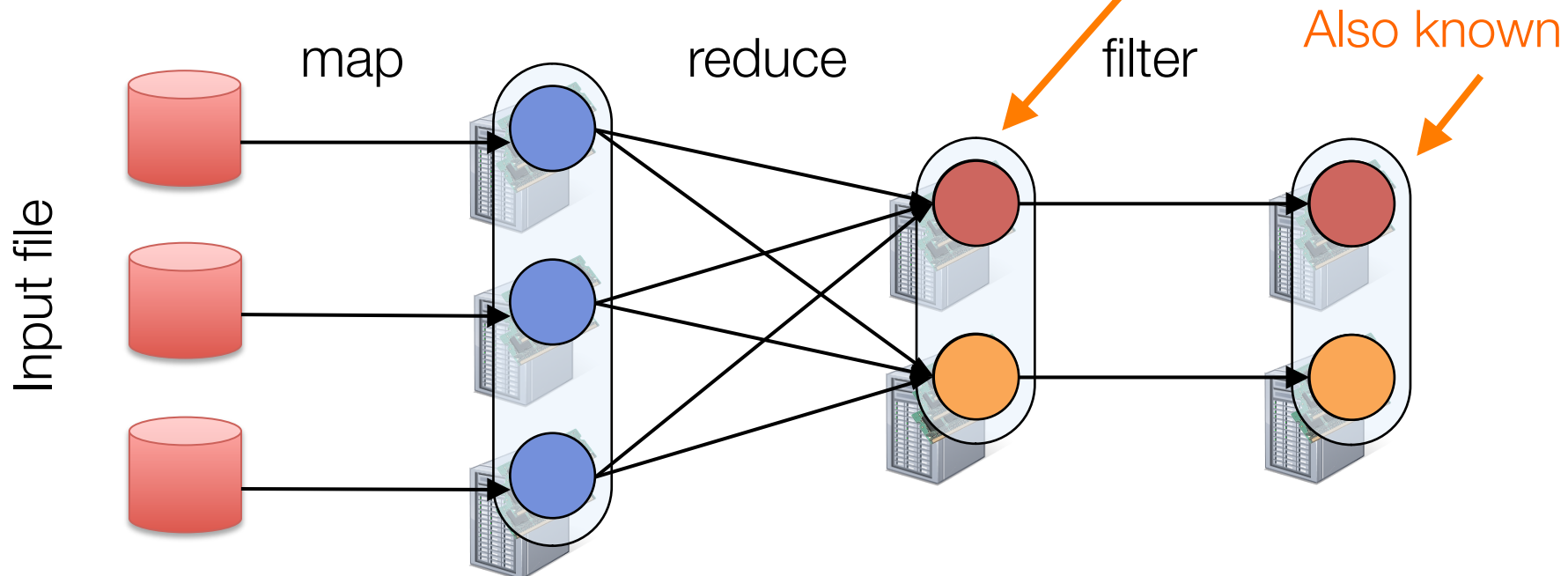


# Partitioning

RDDs know their partitioning functions

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

Known to be  
hash-partitioned



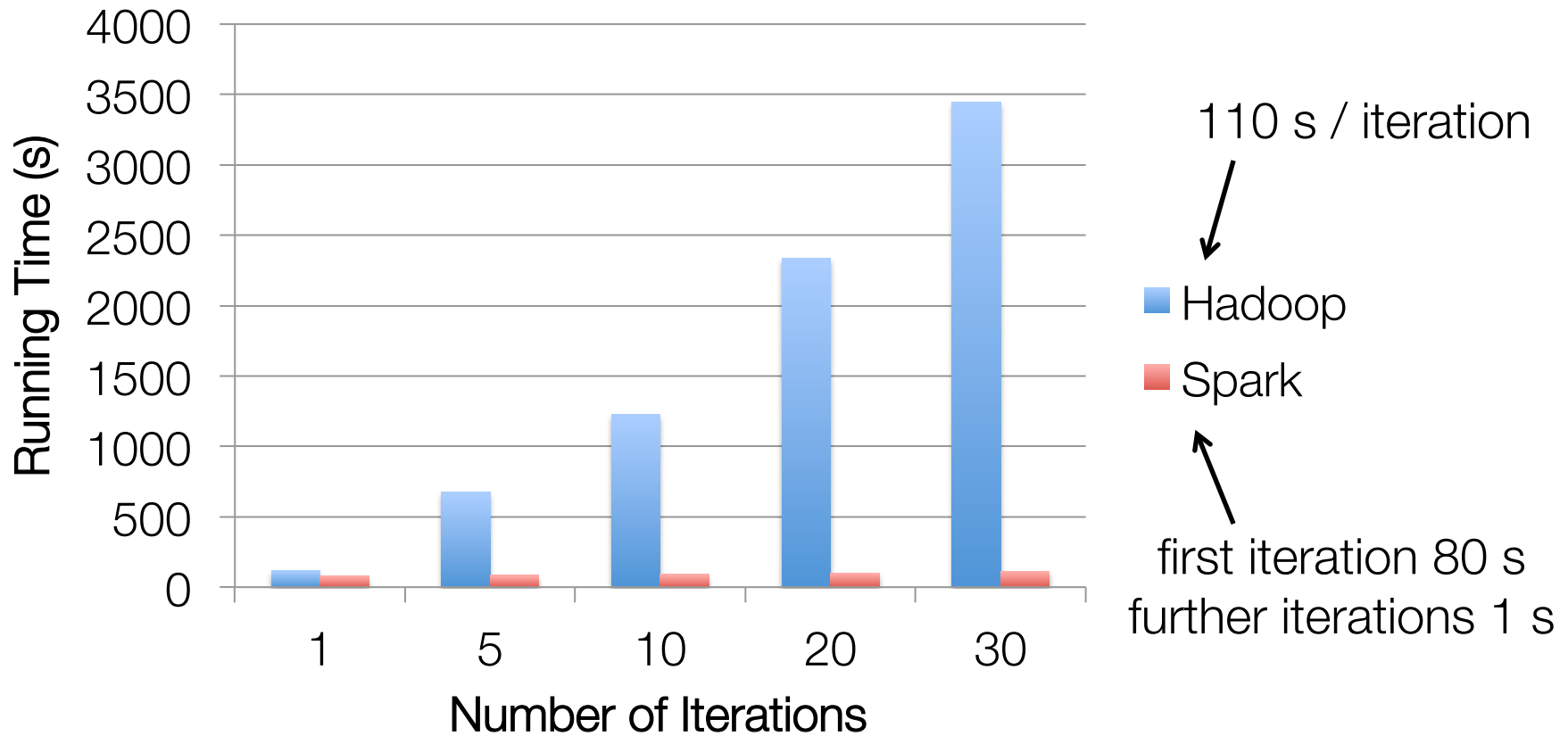
Machine Learning example

# Logistic Regression

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^n g(w; x_i, y_i)$$

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x))) - 1) * p.y * p.x
  }.reduce(_ + _)
  w -= alpha * gradient
}
```

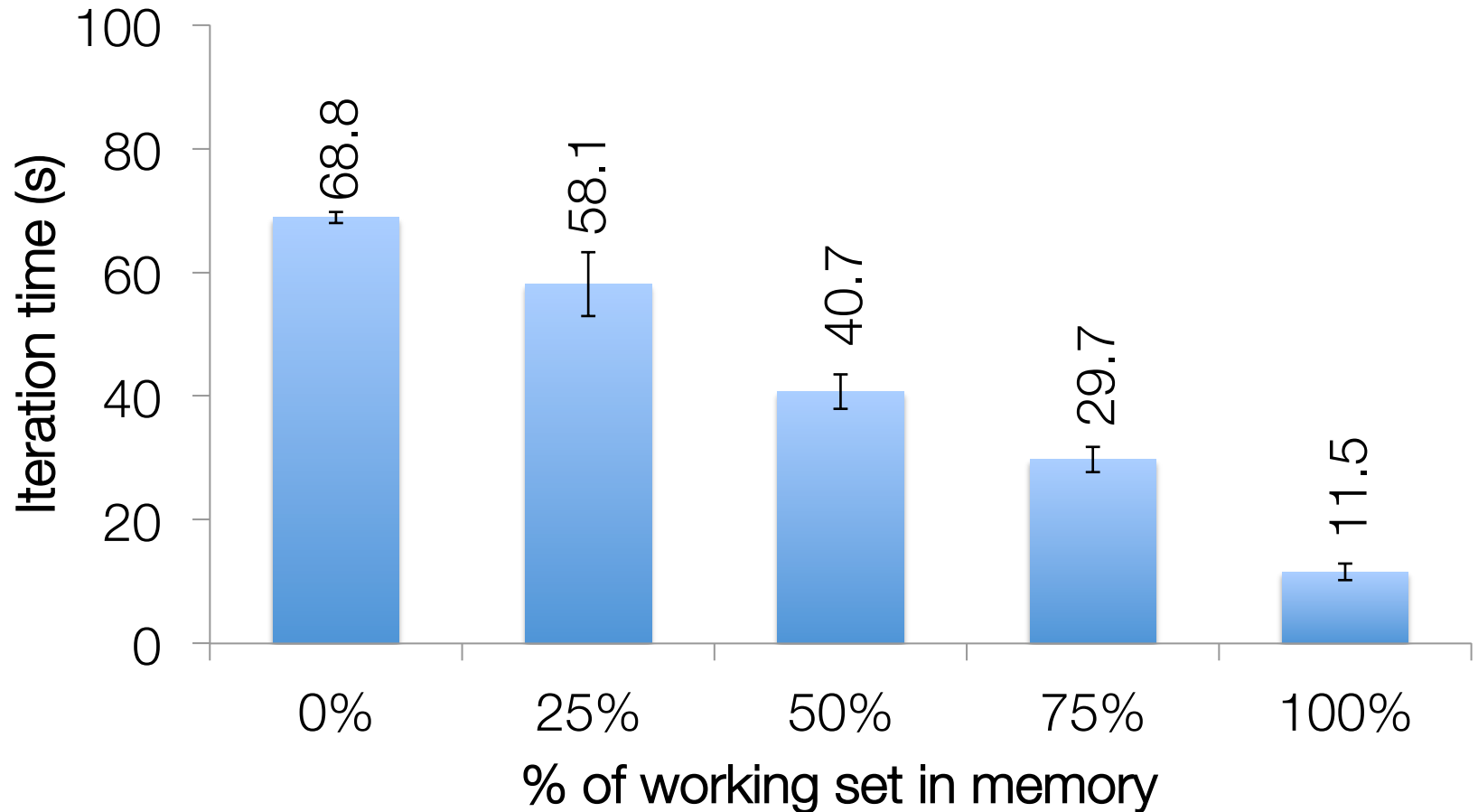
# Logistic Regression Results



100 GB of data on 50 m1.xlarge EC2 machines



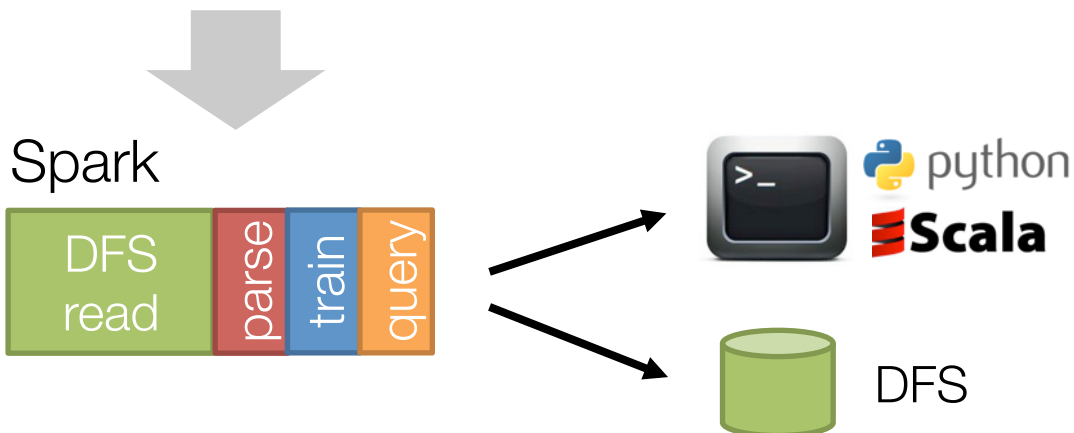
# Behavior with Less RAM



# Benefit for Users

Same engine performs data extraction, model training and interactive queries

Separate engines



State of the Spark ecosystem

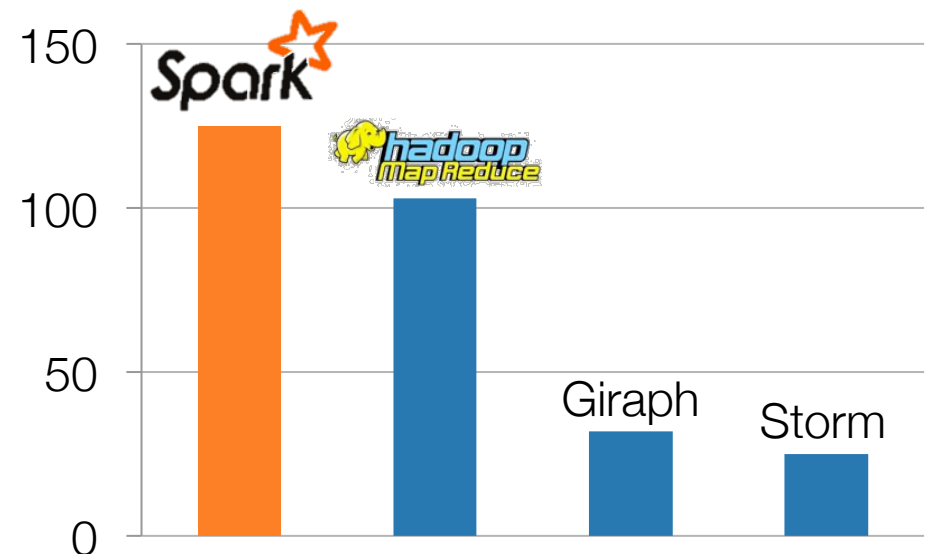
# Spark Community

Most active open source community in big data

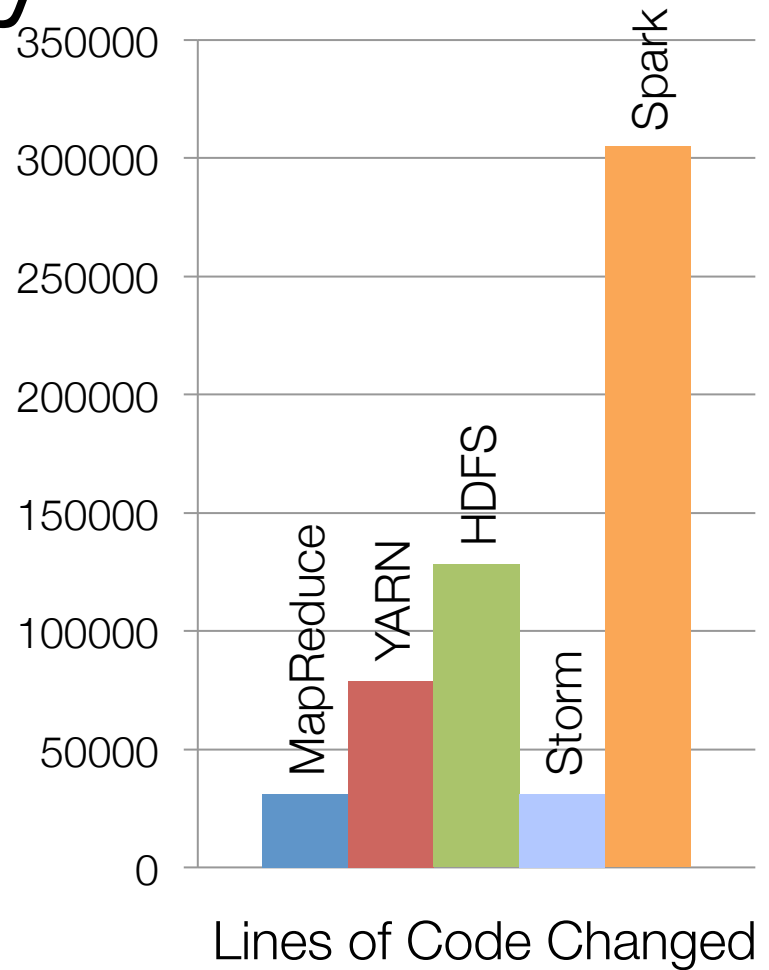
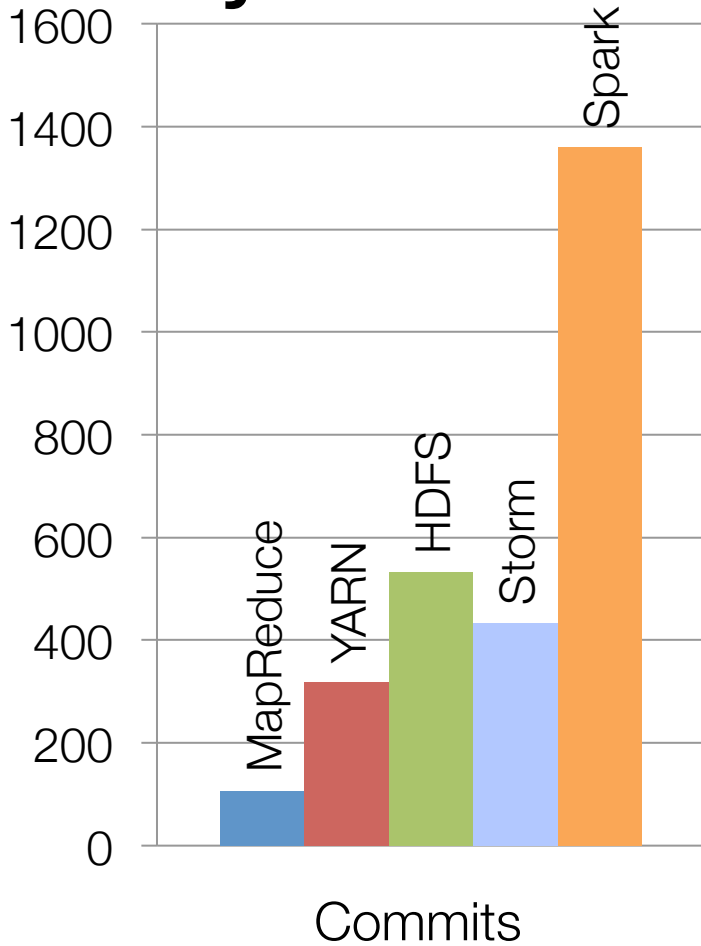
200+ developers, 50+ companies contributing



Contributors in past year



# Project Activity



Activity in past 6 months

# Continuing Growth



Contributors per month to Spark

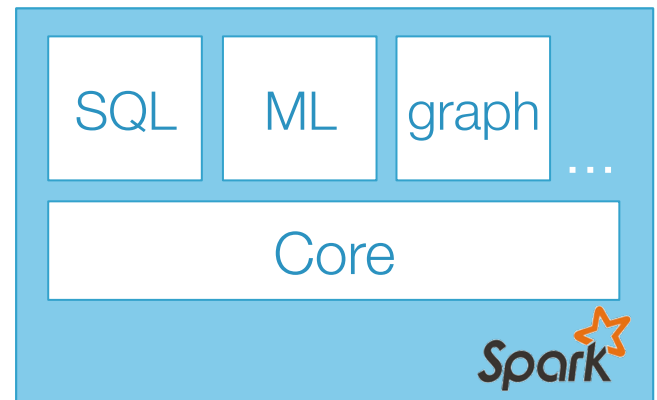
Built-in libraries

# Standard Library for Big Data

Python Scala Java R

Big data apps lack libraries of common algorithms

Spark's generality + support for multiple languages make suitable to offer this

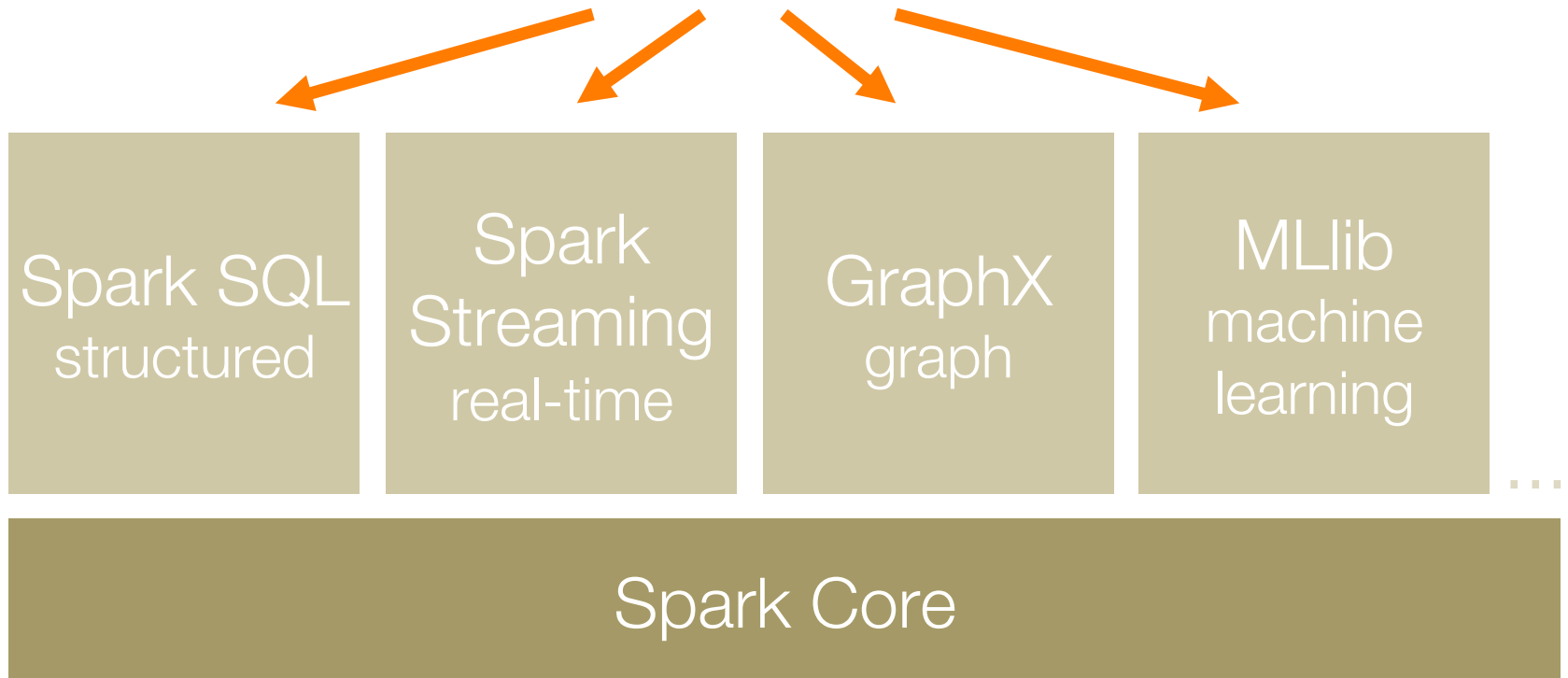


Much of future activity will be in these libraries



# A General Platform

Standard libraries included with Spark



# Machine Learning Library (MLlib)

```
points = context.sql("select latitude, longitude from tweets")  
model = KMeans.train(points, 10)
```

40 contributors in  
past year

# MLlib algorithms

**classification:** logistic regression, linear SVM, naïve Bayes, classification tree

**regression:** generalized linear models (GLMs), regression tree

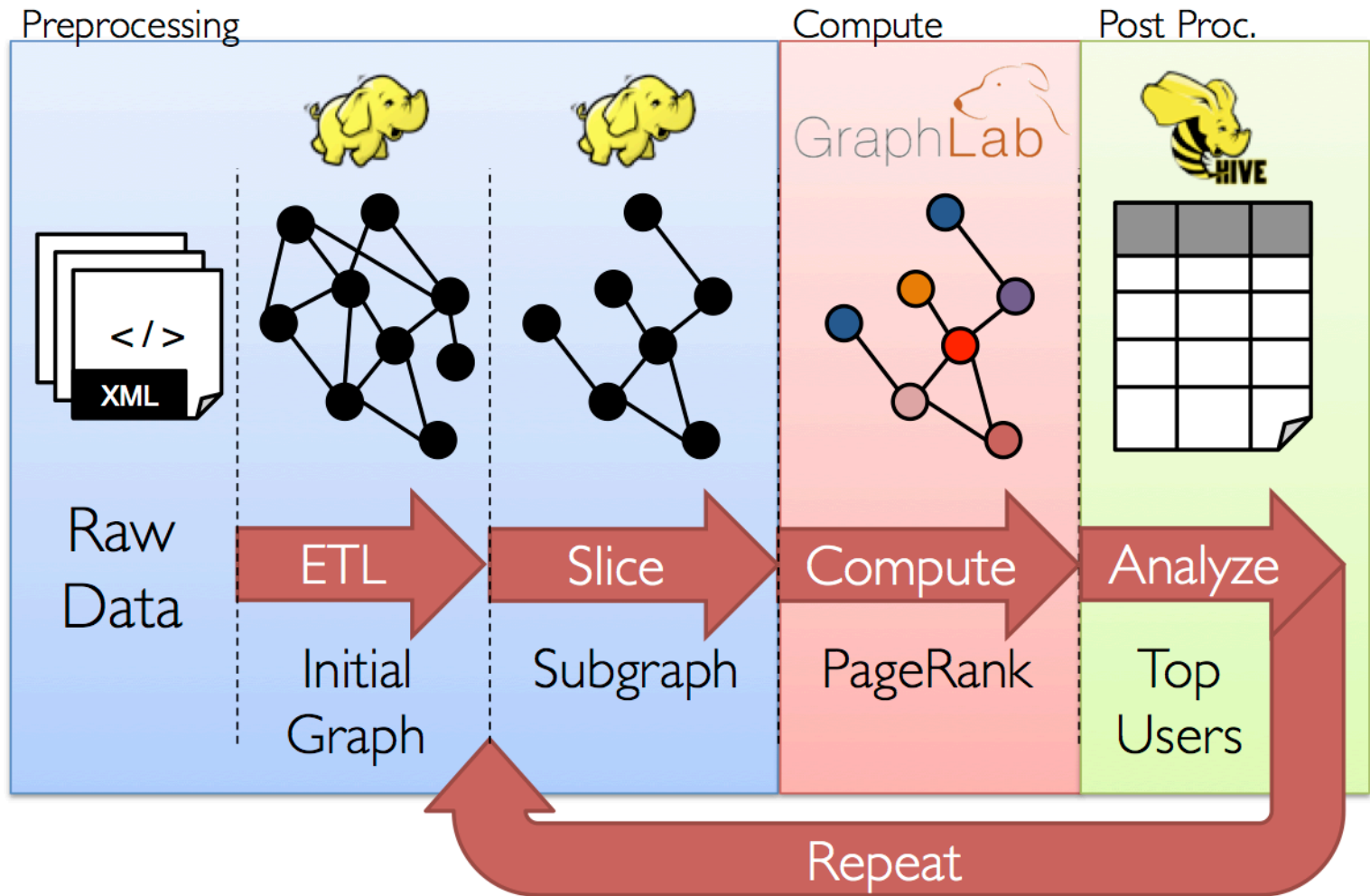
**collaborative filtering:** alternating least squares (ALS), non-negative matrix factorization (NMF)

**clustering:** k-means||

**decomposition:** SVD, PCA

**optimization:** stochastic gradient descent, L-BFGS

# GraphX



# GraphX

General graph processing library

Build graph using RDDs of nodes and edges

Large library of graph algorithms with composable steps

# GraphX Algorithms

## Collaborative Filtering

- » Alternating Least Squares
- » Stochastic Gradient Descent
- » Tensor Factorization

## Community Detection

- » Triangle-Counting
- » K-core Decomposition
- » K-Truss

## Structured Prediction

- » Loopy Belief Propagation
- » Max-Product Linear Programs
- » Gibbs Sampling

## Graph Analytics

- » PageRank
- » Personalized PageRank
- » Shortest Path
- » Graph Coloring

## Semi-supervised ML

- » Graph SSL
- » CoEM

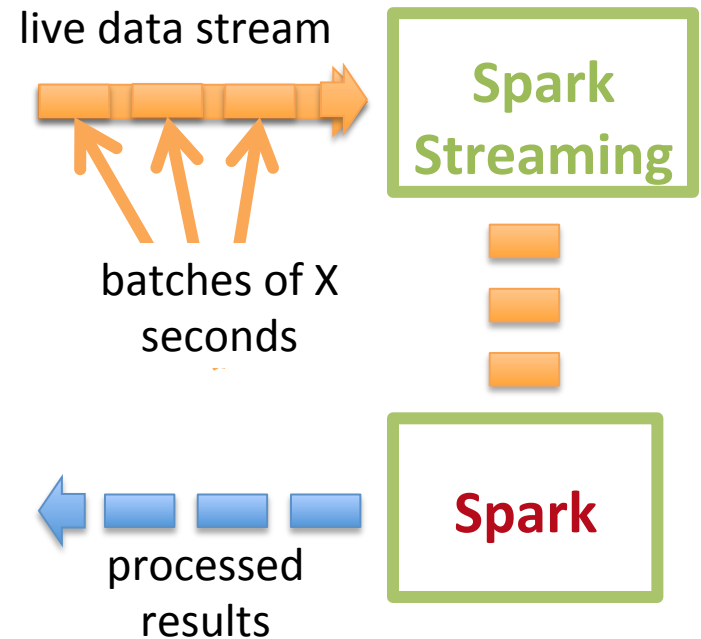
## Classification

- » Neural Networks

# Spark Streaming

Run a streaming computation as a **series** of very small, deterministic batch jobs

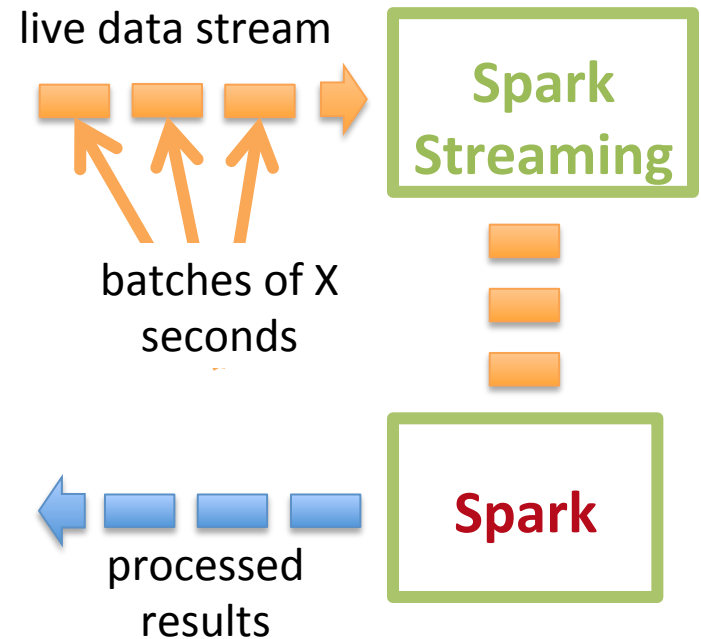
- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



# Spark Streaming

Run a streaming computation as a **series** of very small, deterministic batch jobs

- Batch sizes as low as  $\frac{1}{2}$  second, latency  $\sim 1$  second
- Potential for combining batch processing and streaming processing in the same system





# Spark SQL

*// Run SQL statements*

```
val teenagers = context.sql(  
    "SELECT name FROM people WHERE age >= 13 AND age <= 19")
```

*// The results of SQL queries are RDDs of Row objects*

```
val names = teenagers.map(t => "Name: " + t(0)).collect()
```

# Spark SQL

Enables loading & querying structured data in Spark

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").registerAsTable("tweets")
c.sql("select text, user.name from tweets")
```

tweets.json

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

# Conclusions

# Spark and Research

Spark has all its roots in research, so we hope to keep incorporating new ideas!

# Conclusion

Data flow engines are becoming an important platform for numerical algorithms

While early models like MapReduce were inefficient, new ones like Spark close this gap

More info: [spark.apache.org](http://spark.apache.org)



# Class Schedule

# Schedule

Today and tomorrow

Hands-on exercises, download course materials and slides:

<http://stanford.edu/~rezab/sparkclass/>

Friday

Advanced talks on Spark libraries and uses