Big Data Analytics with Apache Spark

Matei Zaharia

Stanford

databricks

Outline

The big data problem

MapReduce

Apache Spark

How people are using Spark

The Big Data Problem

Data is growing faster than processor speeds

Growing data sources
- » Mostly machine generated

Cheap storage

Stalling CPU speeds

Examples

Facebook's daily event data: 4 PB

Google's web index: 100 PB

1000 genomes project: 200 TB

Cost of 1 TB of disk: $16

Time to read 1 TB from disk: 6 hours (50 MB/s)

The Big Data Problem

Single machine can no longer process or even store all the data!

Only solution is to **distribute** over large clusters


Google Datacenter
How do we program these?

Traditional Network Programming

Message-passing between nodes

Really hard to do at scale:

- How to split problem across nodes?
- How to deal with failures?
- Even worse: stragglers (node is not failed, but slow)

Data-Parallel Models

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

"Here's an operation, run it on all of the data"

- » I don't care *where* it runs (you schedule that)
- » In fact, feel free to run it *twice* on different nodes

Early example: MapReduce

MapReduce

First widely popular programming model for data-intensive apps on clusters

Published by Google in 2004

Popularized by open-source Hadoop project

MapReduce Programming Model

Data type: key-value records

Map function:

$$(K_{in}, V_{in}) \rightarrow \text{list}(K_{inter}, V_{inter})$$

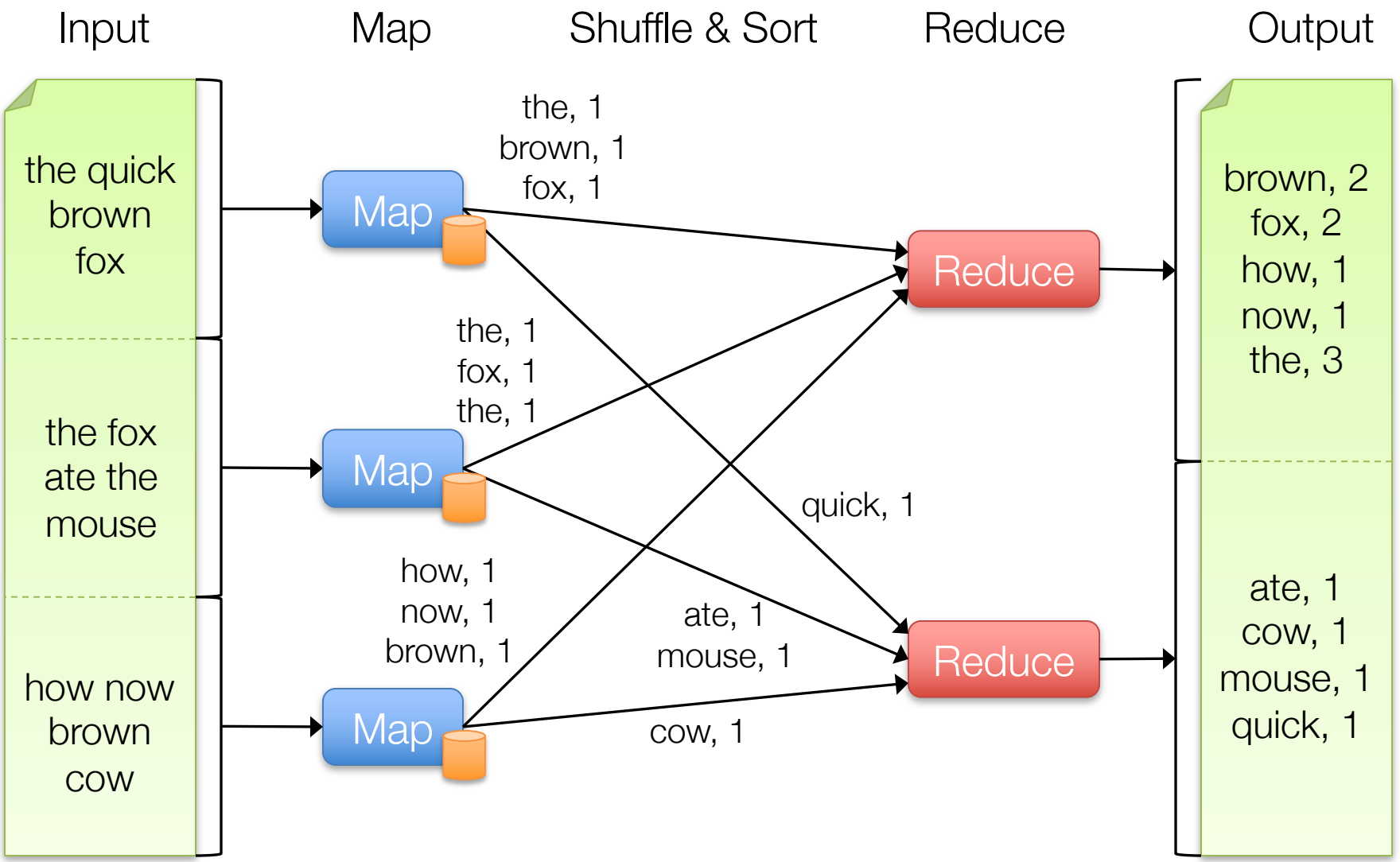
Reduce function:

$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

Example: Word Count

```
def map(line):
    foreach word in line.split():
        output(word, 1)


def reduce(key, values):
    output(key, sum(values))
```

Word Count Execution



Other MapReduce Apps

How would you write these with map & reduce?

Search: find all the records in a file that contain the word “Berkeley”

Top words: Find the top 10 most common words

MapReduce Execution

Mappers are scheduled on same node as their input block if possible

- » Minimize network use to improve performance

Mappers save outputs to local disk before serving to reducers

- » Allows recovery if a reducer crashes
- » Allows running more reducers than # of nodes

Fault Tolerance in MapReduce

1. If a task crashes:
   - » Retry on another node
     - OK for a map because it had no dependencies
     - OK for reduce because map outputs are on disk
   - » If the same task repeatedly fails, fail the job or ignore that input block

➢ Note: For fault tolerance to work, user tasks must be deterministic and side-effect-free

Fault Tolerance in MapReduce

2. If a node crashes:
   - » Relaunch its current tasks on other nodes
   - » Relaunch any maps the node previously ran
     - Necessary because their output files were lost along with the crashed node

Fault Tolerance in MapReduce

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node
   - Take the output of whichever copy finishes first, and kill the other one

Critical for performance in large clusters (many possible causes of stragglers)

Summary

Data-parallel programming models let systems automatically manage much of execution:

- » Assigning work, load balancing, fault recovery

But... the story doesn't end here!

Outline

The big data problem

MapReduce

Apache Spark

How people are using Spark

Limitations of MapReduce

Programmability: most applications require higher level functions than map / reduce

- » E.g. statistics, matrix multiply, graph search
- » Google ads pipeline had 20 MR steps

Performance: inefficient to combine multiple MapReduce steps into complex programs

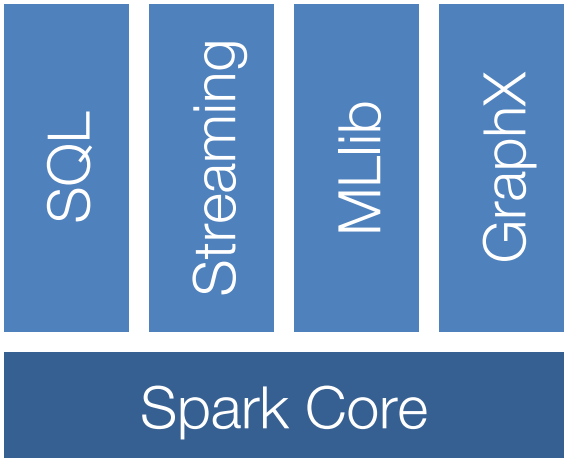
Apache Spark

Programming model that generalizes MapReduce to support more applications

» Adds efficient, in-memory **data sharing**

Large library of built-in functions

APIs in Python, Java, Scala, R



Spark Programmability

WordCount in MapReduce:

```
#include "mapreduce/mapreduce.h"

// User's map function
class SplitWords: public Mapper {
  public:
  virtual void Map(const MapInput& input)
  {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
      // Skip past leading whitespace
      while (i < n && isspace(text[i]))
        i++;
      // Find word end
      int start = i;
      while (i < n && !isspace(text[i]))
        i++;
      if (start < i)
        Emit(text.substr(
            start,i-start),"1");
    }
  }
};

REGISTER_MAPPER(SplitWords);
```

```
// User's reduce function
class Sum: public Reducer {
  public:
  virtual void Reduce(ReduceInput* input)
  {
    // Iterate over all entries with the
    // same key and add the values
    int64 value = 0;
    while (!input->done()) {
      value += StringToInt(
                 input->value());
      input->NextValue();
    }
    // Emit sum for input->key()
    Emit(IntToString(value));
  }
};

REGISTER_REDUCER(Sum);
```

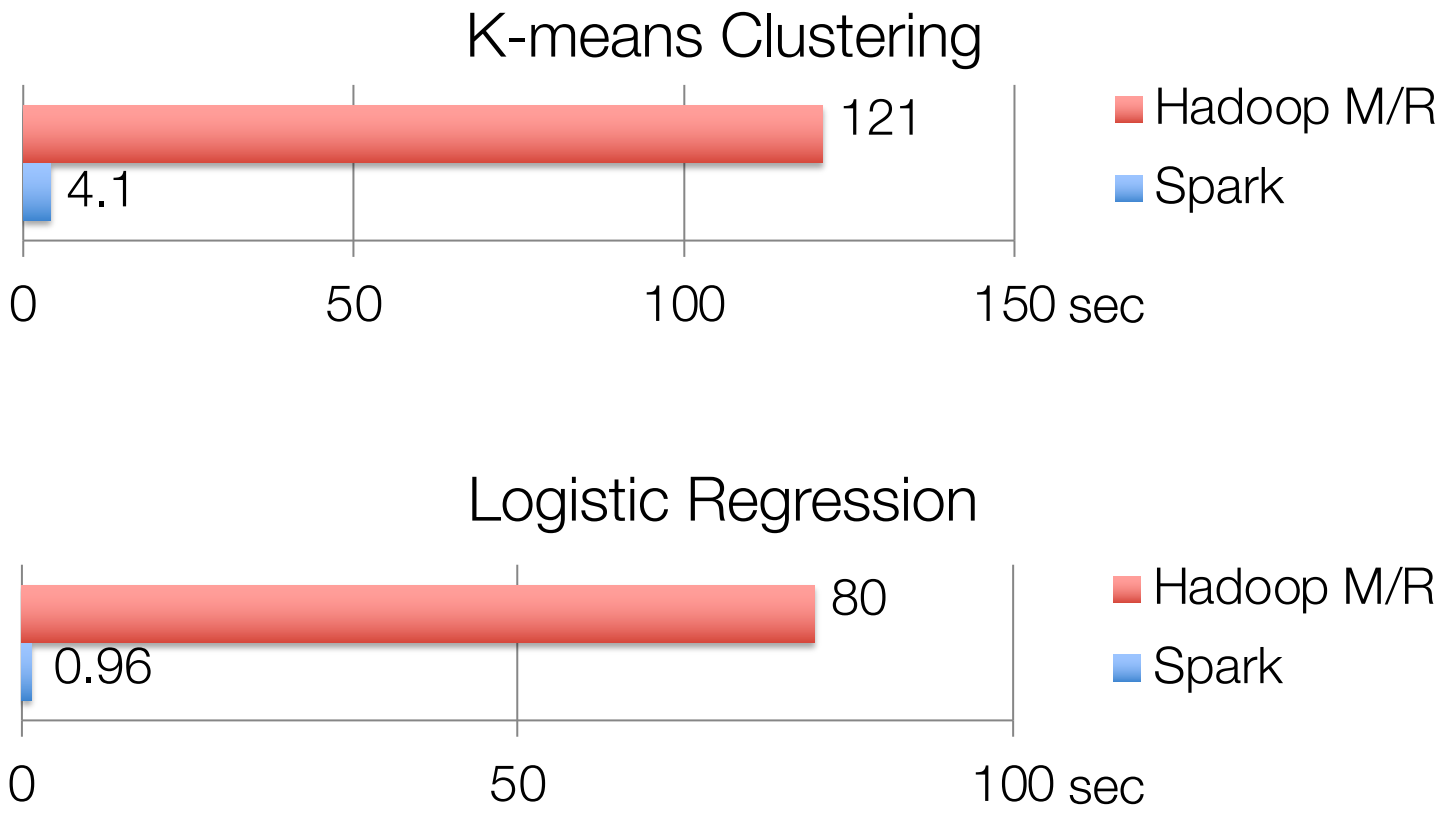
```
int main(int argc, char** argv) {
  ParseCommandLineFlags(argc, argv);
  MapReduceSpecification spec;
  for (int i = 1; i < argc; i++) {
    MapReduceInput* in= spec.add_input();
    in->set_format("text");
    in->set_filepattern(argv[i]);
    in->set_mapper_class("SplitWords");
  }

  // Specify the output files
  MapReduceOutput* out = spec.output();
  out->set_filebase("/gfs/test/freq");
  out->set_num_tasks(100);
  out->set_format("text");
  out->set_reducer_class("Sum");

  // Do partial sums within map
  out->set_combiner_class("Sum");

  // Tuning parameters
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);

  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  return 0;
}
```

Spark Programmability

WordCount in Spark:

```
file = spark.textFile("hdfs://...")

counts = file.flatMap(lambda line: line.split(" "))
              .map(lambda word: (word, 1))
              .reduceByKey(lambda a, b: a+b)

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

Spark Performance



Programming Model

Write programs in terms of transformations on distributed datasets

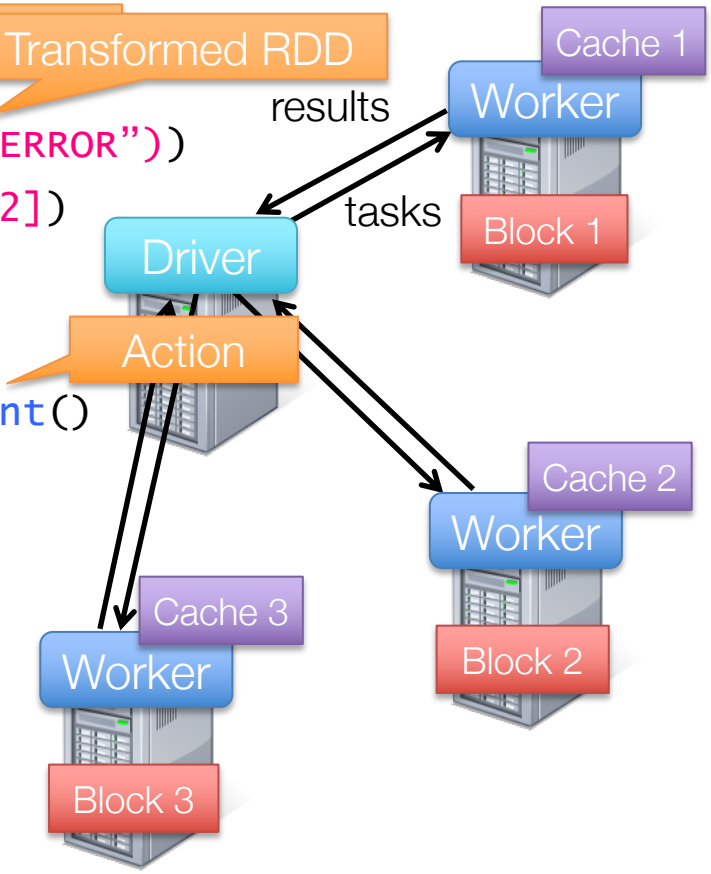
Low-level API: Resilient Distributed Dataset (RDD)

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

Example: Text Search

Load a large log file 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: "Berkeley" in s).count()
messages.filter(lambda s: "MIT" in s).count()
. . .
```

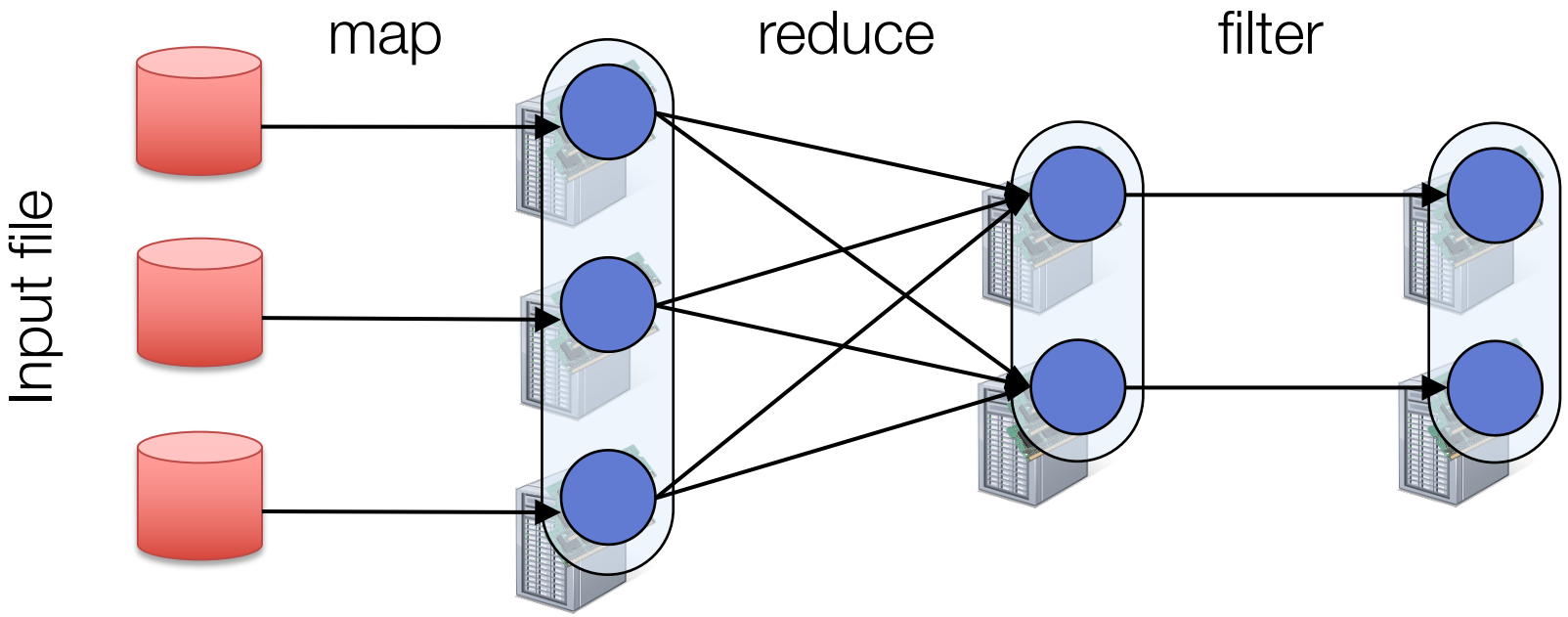


Example: full-text search of Wikipedia in 0.5 sec (vs 20 s for on-disk data)

Fault Tolerance

RDDs track *lineage* info to rebuild lost data

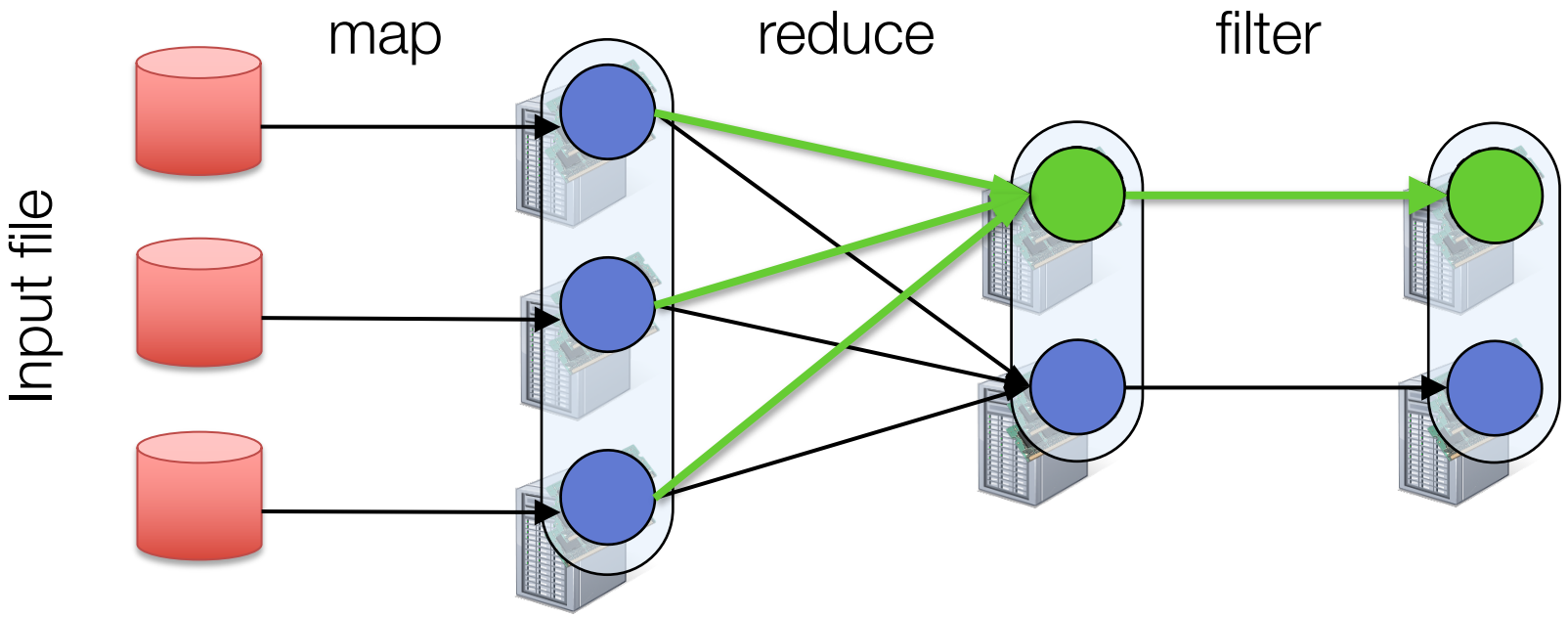
```
file.map(lambda record: (record.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda type, count: count > 10)
```



Fault Tolerance

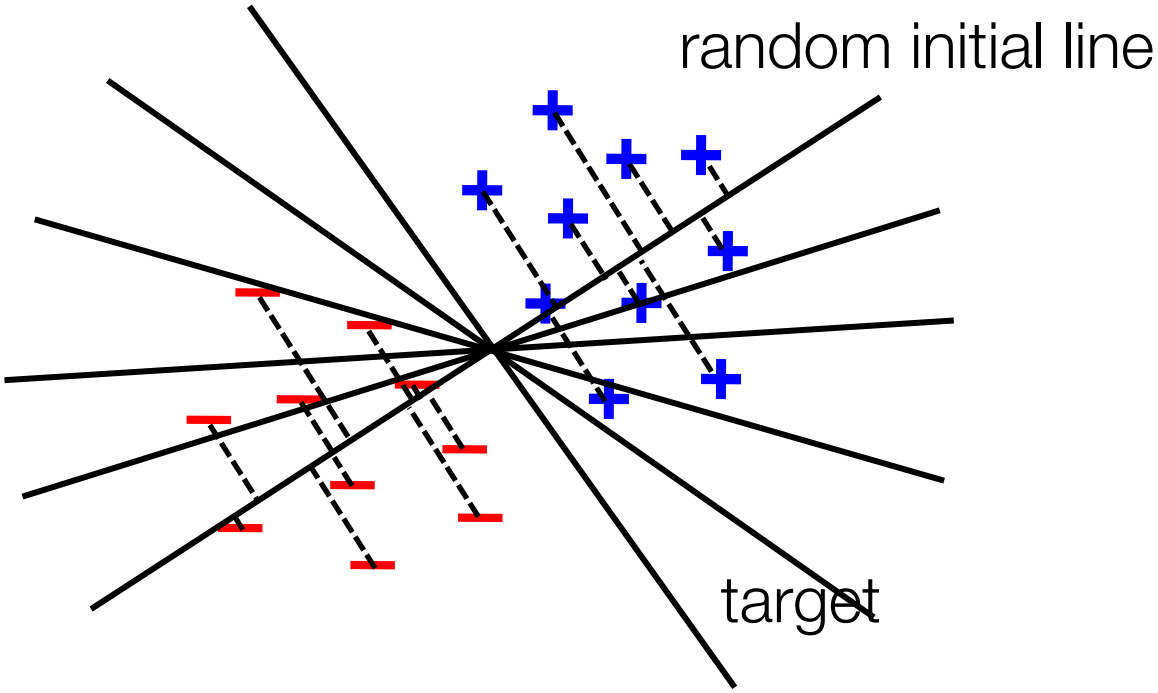
RDDs track *lineage* info to rebuild lost data

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

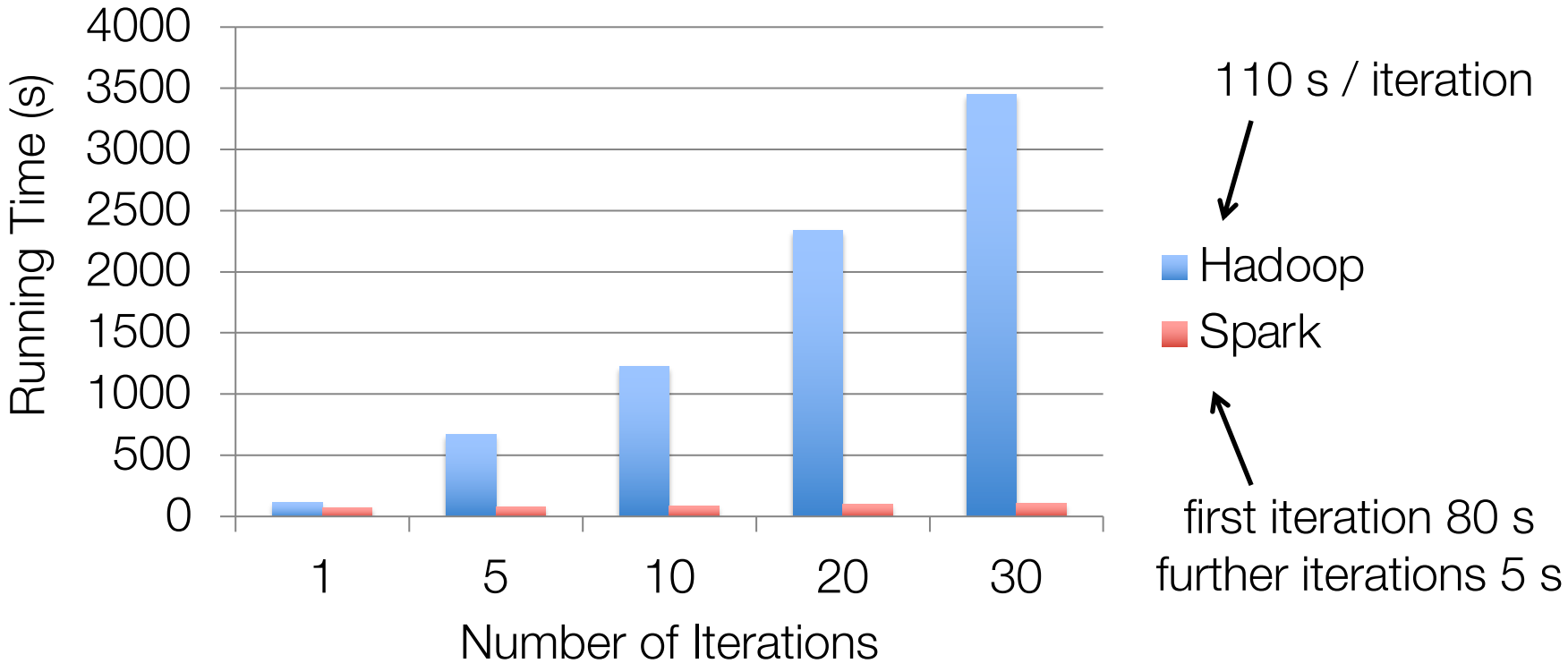


Example: Logistic Regression

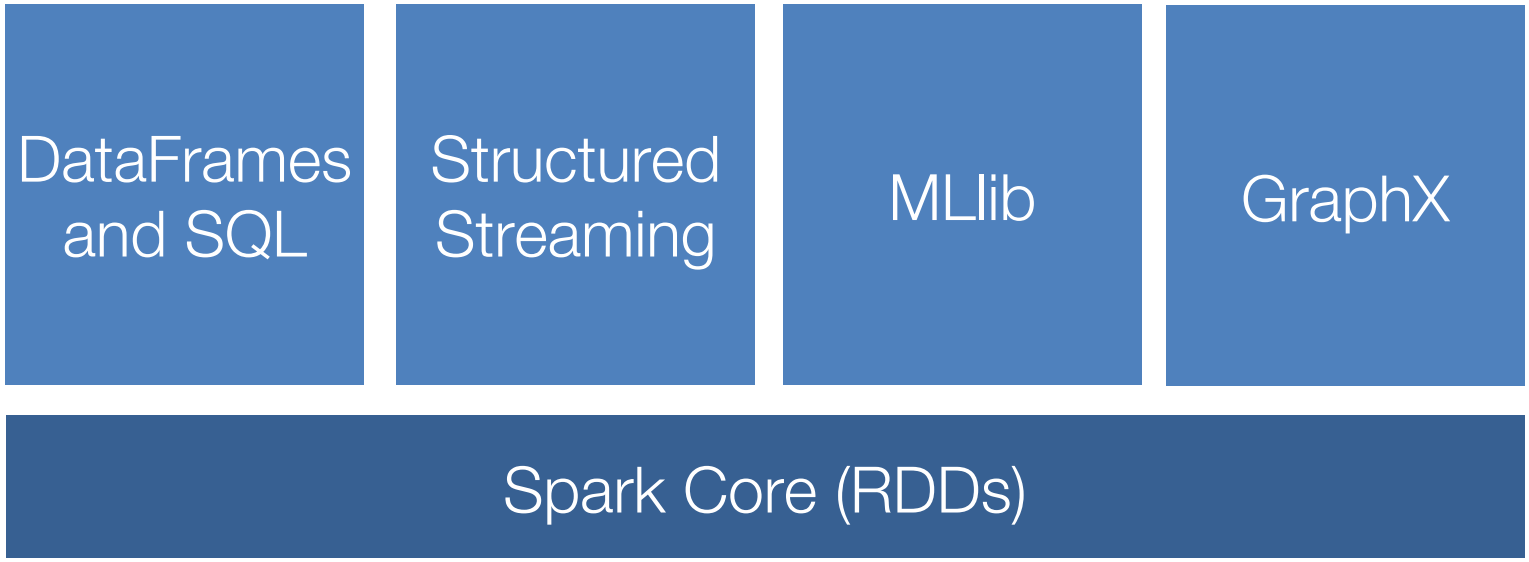
Goal: find line separating two sets of points



Logistic Regression Performance

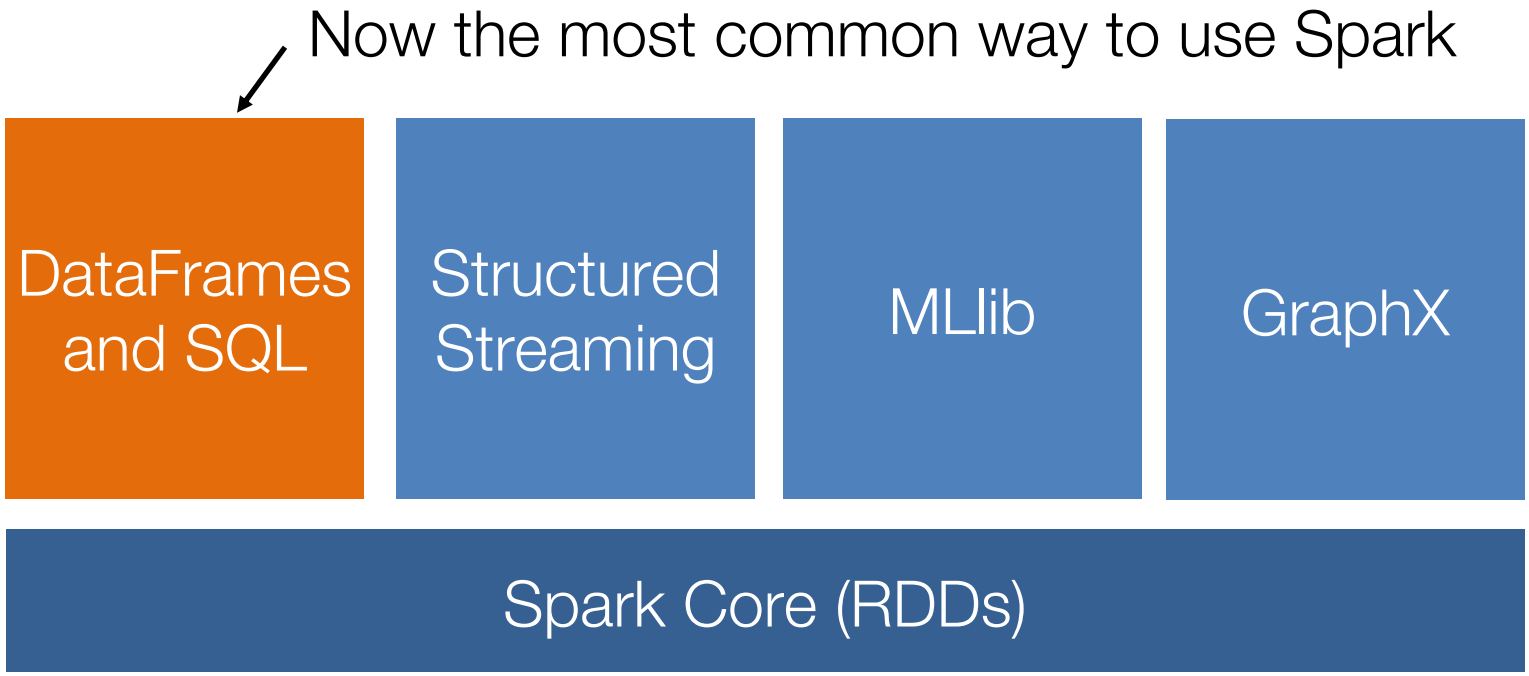


Built-in Libraries



Largest integrated standard library for big data

Built-in Libraries



Largest integrated standard library for big data

Challenges with RDD API

Looks high-level, but hides many semantics of computation from engine

- » Functions passed in are arbitrary blocks of code
- » Data stored is arbitrary Java/Python objects

Users can mix APIs in suboptimal ways

Example Problem

```
pairs = data.map(lambda word: (word, 1))

groups = pairs.groupByKey()

groups.map(lambda k, vs: (k, vs.sum))
```

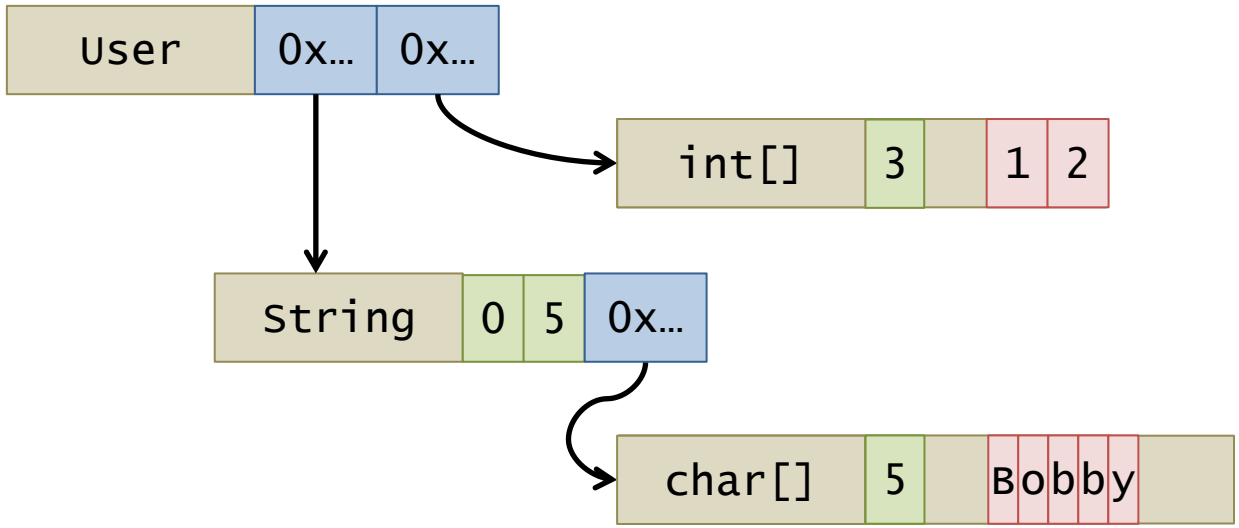
Writes all groups as lists of integers in memory

Then promptly aggregates them

Challenge: Data Representation

Java & Python objects often have high overhead

```
class User(name: String, friends: Array<Int>)
User("Bobby", Array(1, 2))
```

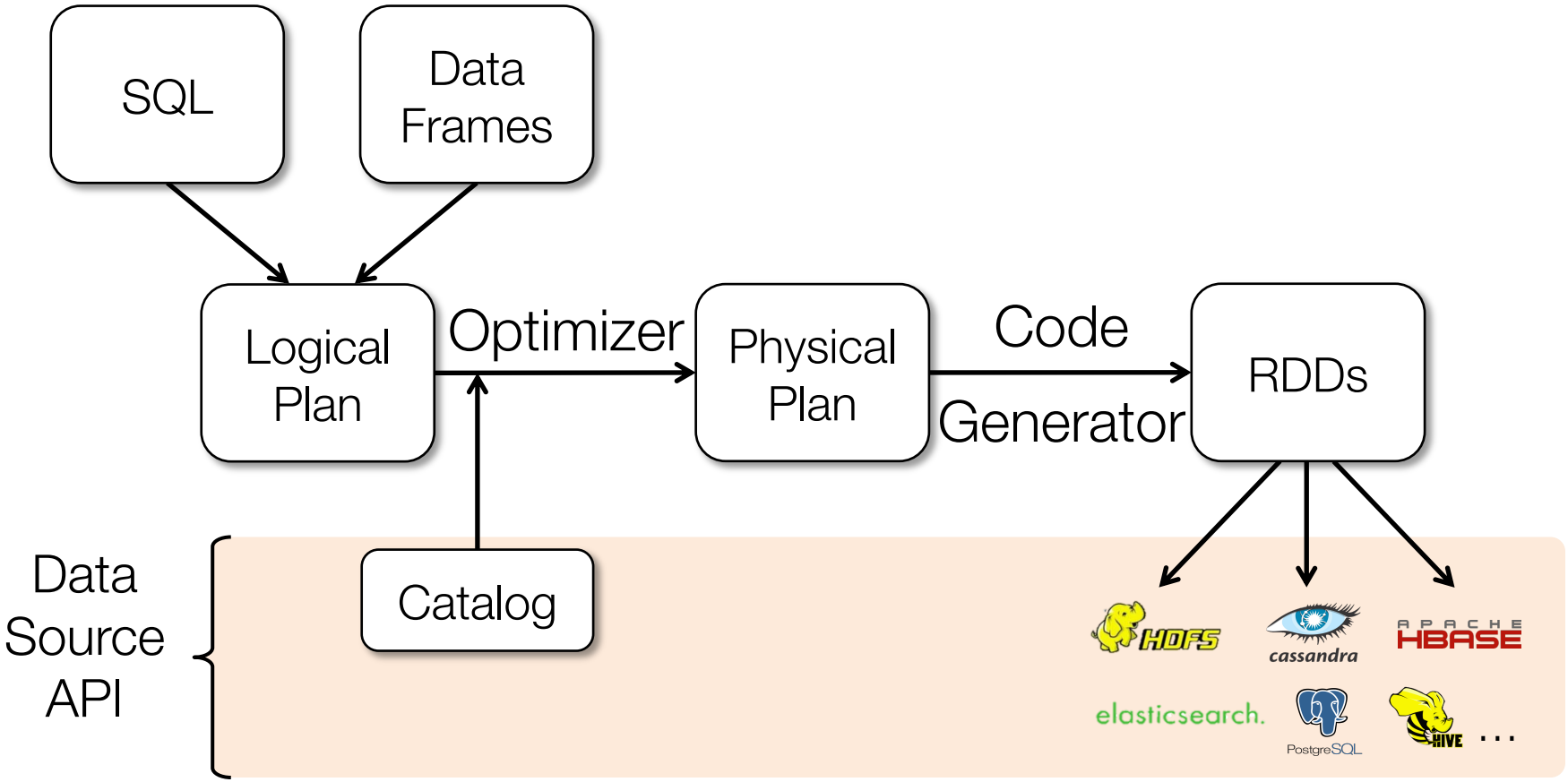


Spark SQL & DataFrames

Efficient library for working with structured data

- » Two interfaces: SQL for data analysts and external apps, DataFrames for complex programs
- » Automatically optimizes computation and storage underneath, as in a relational database

Spark SQL Architecture



DataFrame API

DataFrames hold rows with a known schema and offer relational operations through a DSL

```
users = spark.sql("select * from users")

ma_users = users[users.state == "MA"]

ma_users.count()

ma_users.groupBy("name").avg("age")

ma_users.map(lambda row: row.user.to_upper())
```

Expression tree (AST)

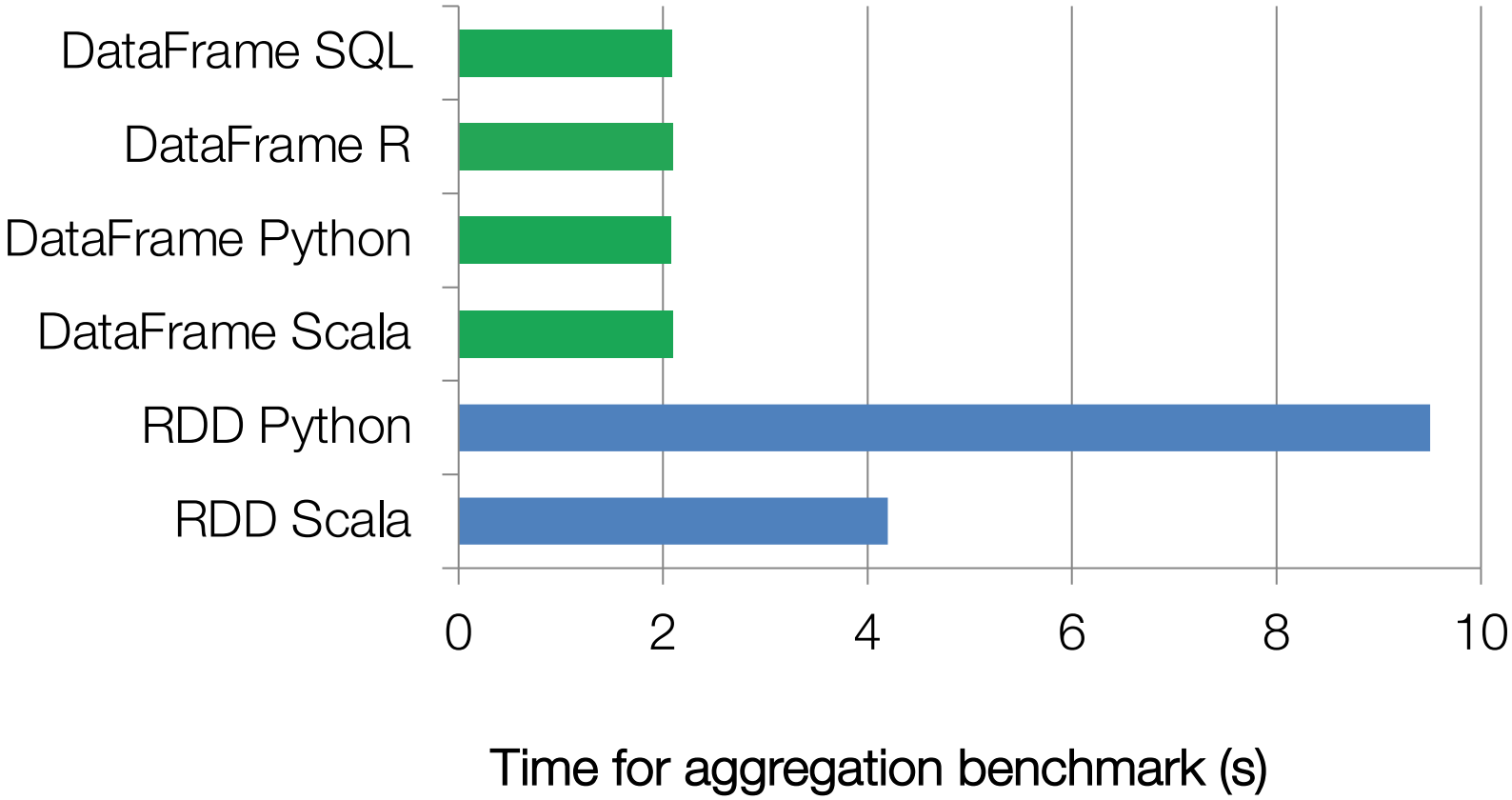
What DataFrames Enable

1. Compact binary representation
    - Columnar, compressed cache; rows for processing

2. Optimization across operators (join reordering, predicate pushdown, etc)

3. Runtime code generation

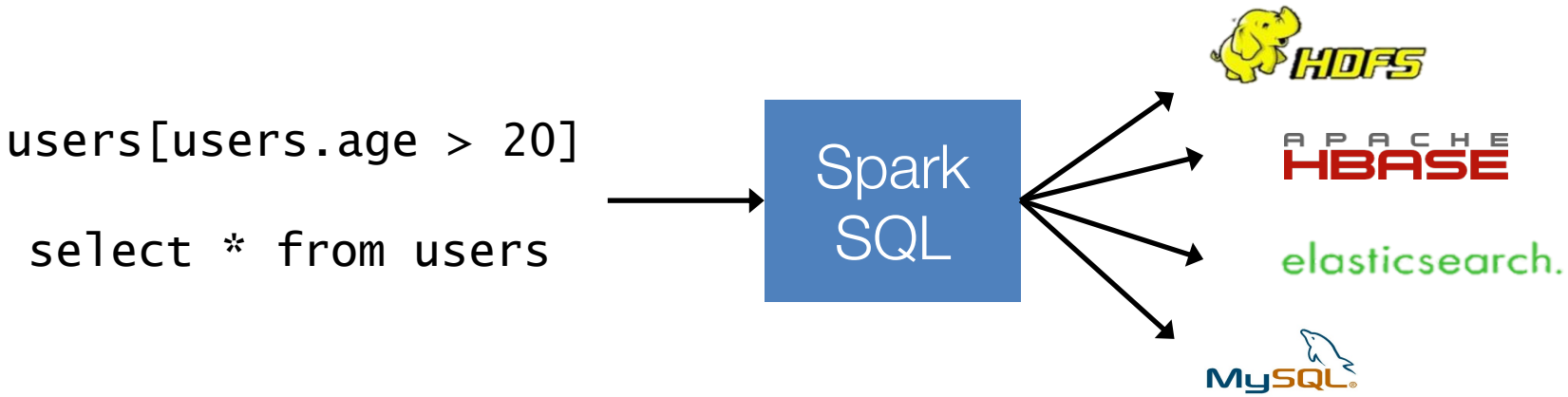
Performance



Data Sources

Uniform way to access structured data

- » Apps can migrate across Hive, Cassandra, JSON, …
- » Rich semantics allows *query pushdown* into sources



Examples

JSON:

```
select user.id, text from tweets
```

JDBC:

```
select age from users where lang = “en”
```

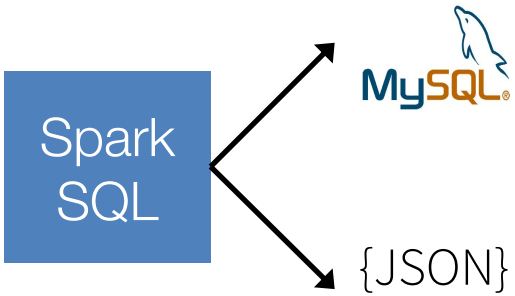
Together:

```
select t.text, u.age
from tweets t, users u
where t.user.id = u.id
and u.lang = “en”
```

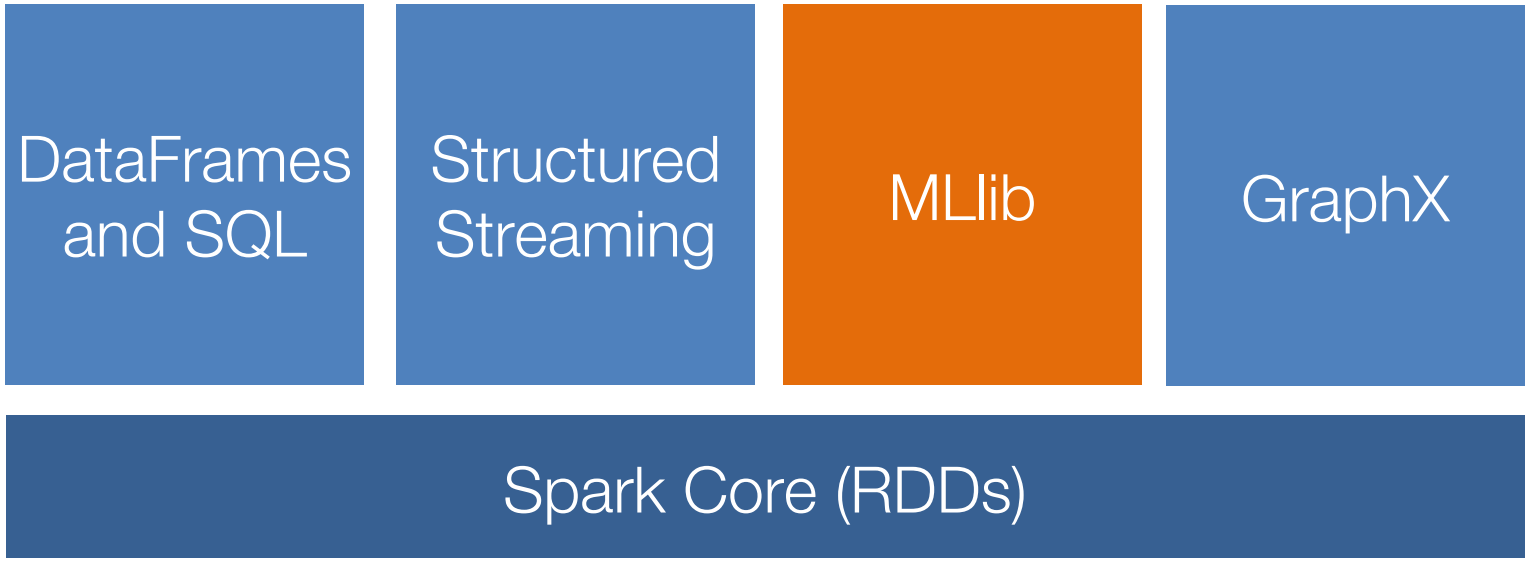
```
{
 “text”: “hi”,
 “user”: {
   “name”: “bob”,
   “id”: 15 }
}
```

tweets.json

```
select id, age from
users where lang=“en”
```



Built-in Libraries



Largest integrated standard library for big data

Spark MLlib

Machine learning library with 50+ algorithms:

- **Classification/regression:** linear models, decision trees, random forests, isotonic regression, …
- **Clustering:** k-means, Gaussian mixture, LDA, ...
- **Collaborative filtering:** ALS, NMF
- **Dimensionality reduction:** SVD, PCA

“Pipeline” API inspired by scikit-learn

MLlib Pipeline Example

```
# Configure an ML pipeline with three stages
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(),
                      outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)

pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training data
model = pipeline.fit(train_data)

# Make predictions on DataFrame of test data
prediction = model.transform(test_data)
```

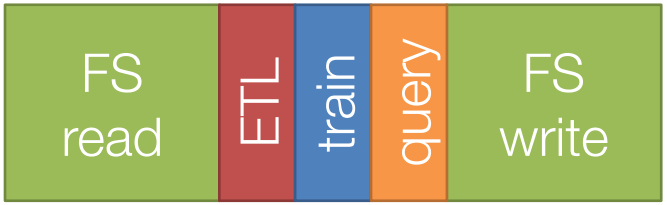
Combining Libraries

```
# Prepare a dataset using SQL
ctx.jsonFile(“tweets.json”).registerTempTable(“tweets”)
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)
```

Performance of Composition

Separate computing frameworks:

Spark:



Summary

Libraries + function-based interface let users write parallel programs similar to sequential code

Can use Spark interactively in Python, R, etc

Outline

The big data problem

MapReduce

Apache Spark

How people are using Spark

Spark Community

20M downloads/month, clusters up to 8000 nodes

1800 contributors to open source project

Example Use Cases

Correlate 500,000 patient records with DNA to design therapies

Query 170 PB of data that used to be in 14 databases

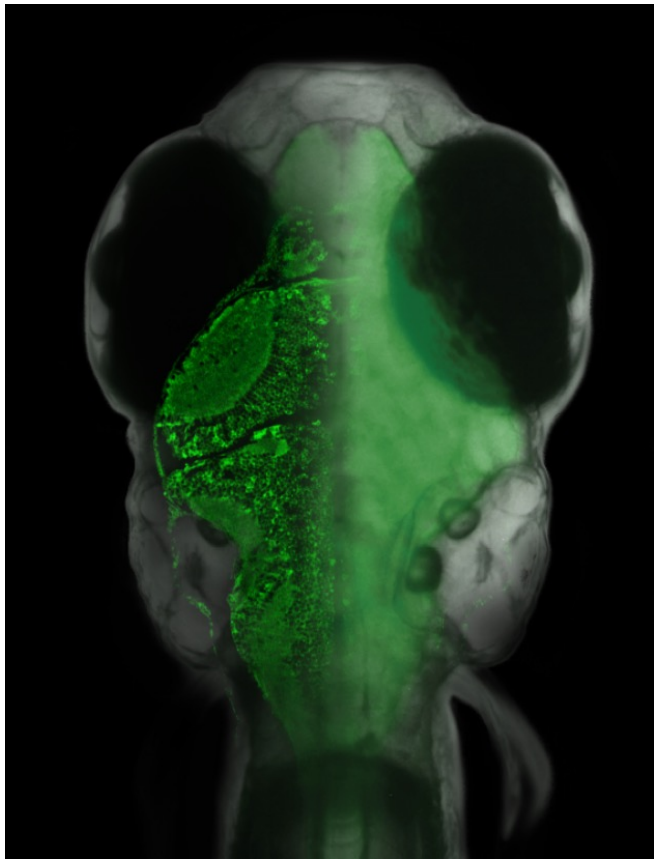
Optimize production using ML and SQL on petabyte-scale data

Identify securities fraud via ML on 30 PB of data

5000+ customers running 10 million VMs/day on Databricks alone

Early Use Case: Neuroscience

HHMI Janelia Farm analyzes data from full-brain imaging of neural activity



Larval zebrafish

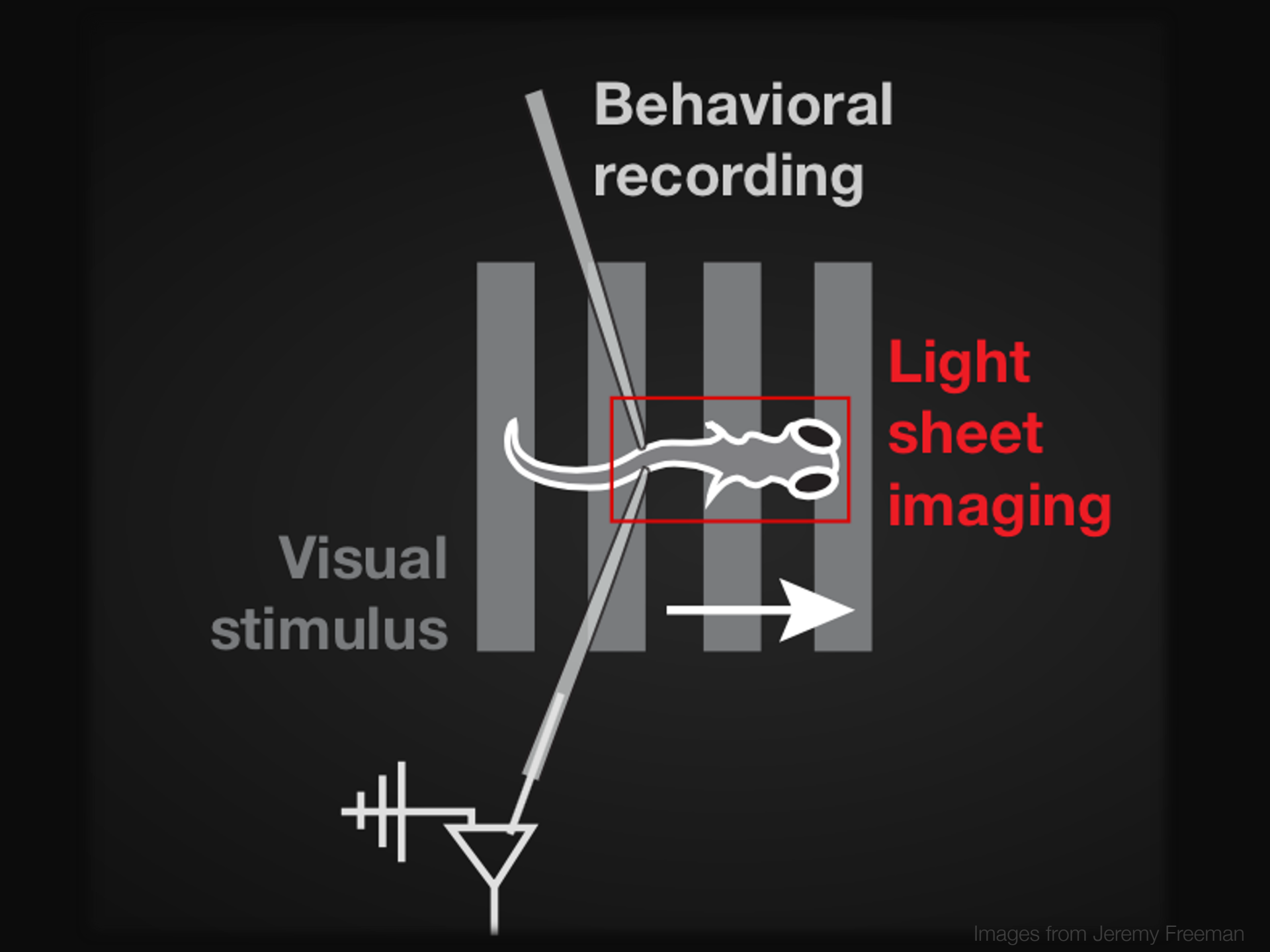
+

Light-sheet imaging

=

2 TB / hour of data

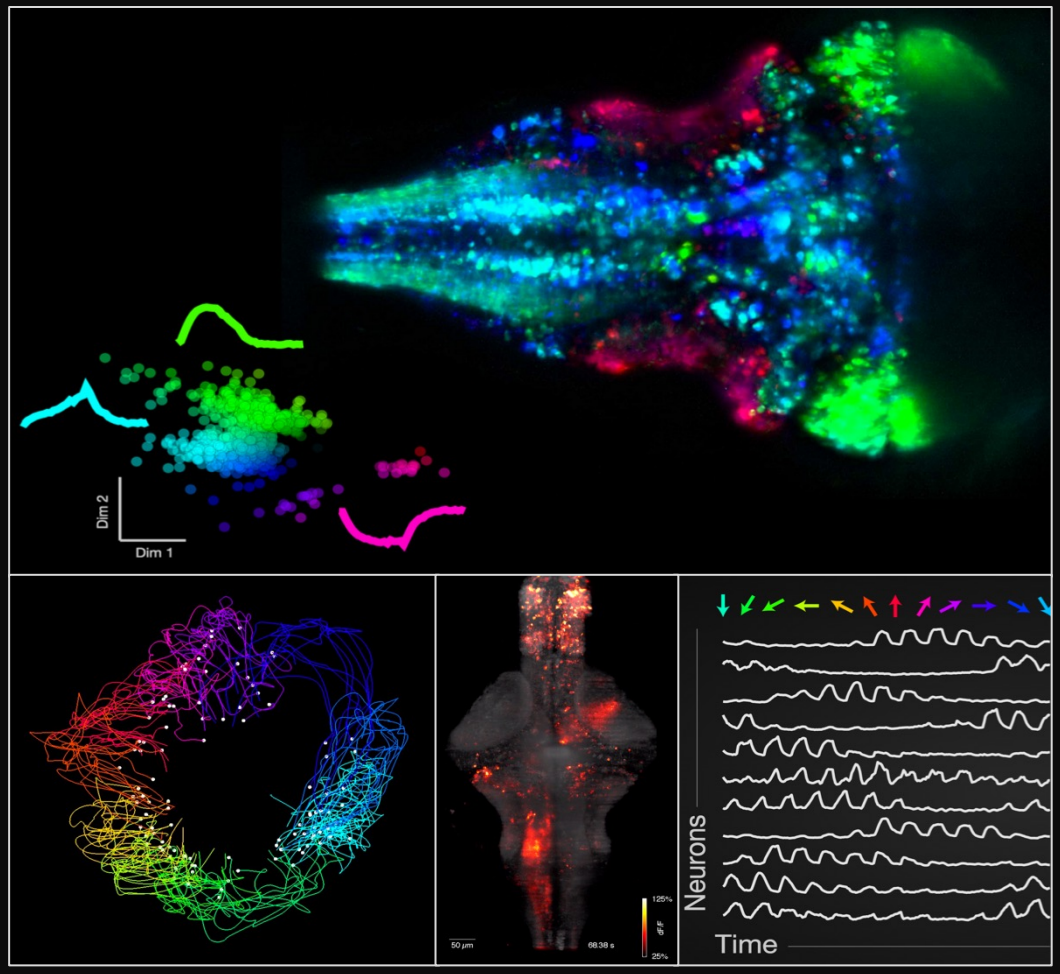
Images from Jeremy Freeman



Images from Jeremy Freeman

Data Analysis

Streaming code does clustering, dimens. reduction on 80-node cluster

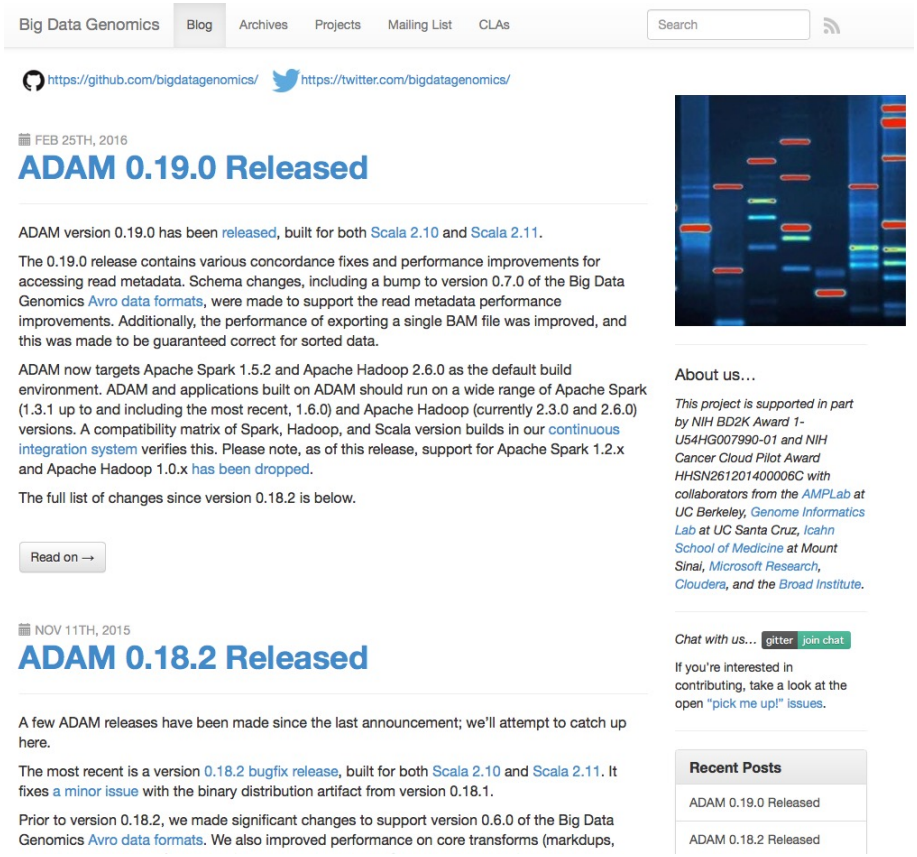


Images from Jeremy Freeman

Early Use Case: Genomics

Berkeley ADAM stores and processes reads with standard big data tools & formats

bdgenomics.org

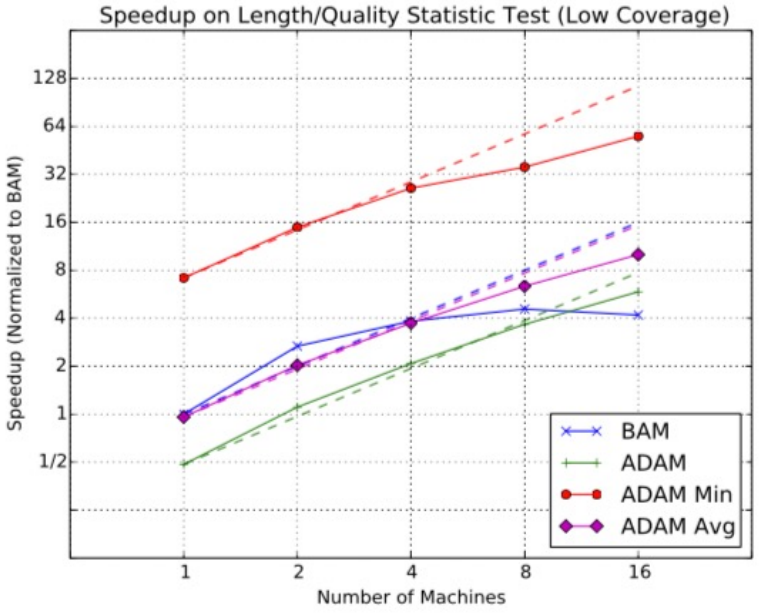


ADAM Results

25% reduction in file size over BAM, by using standard Apache Parquet format

Sorting, filtering, and base quality score recalibration (BQSR) scale up linearly

Interactive use in shell



Conclusion

Apache Spark offers a high-level interface to work with big data based on data-parallel model

Large set of existing libraries

Easy to try on just your laptop!
spark.apache.org