

Spark & Spark SQL

High-Speed In-Memory Analytics
over Hadoop and Hive Data

Instructor: Duen Horng (Polo) Chau

Slides adopted from Matei Zaharia (MIT) and Oliver Vagner (NCR)

What is Spark

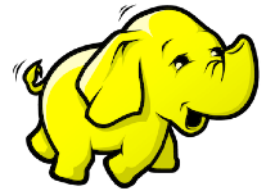
<http://spark.apache.org>

Not a modified version of Hadoop

Separate, fast, MapReduce-like engine

- » **In-memory** data storage for very fast iterative queries
- » General execution graphs and powerful optimizations
- » Up to 40x faster than Hadoop

Compatible with Hadoop's storage APIs



- » Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc.

What is Spark SQL?

(Formally called Shark)

Port of Apache Hive to run on Spark

Compatible with existing Hive data, metastores, and queries (HiveQL, UDFs, etc)

Similar speedups of up to 40X

Project History [latest: v2.1]

Spark project started in 2009 at UC Berkeley AMP lab,
open sourced 2010



Became **Apache Top-Level Project** in Feb 2014

Shark/Spark SQL started summer 2011

Built by 250+ developers and people from 50 companies

Scale to **1000+ nodes** in production

In use at Berkeley, Princeton, Klout, Foursquare, Conviva,
Quantifind, Yahoo! Research, ...

Why a New Programming Model?

MapReduce greatly simplified big data analysis

But as soon as it got popular, users wanted more:

- » More **complex**, multi-stage applications (e.g. iterative **graph algorithms** and **machine learning**)
- » More **interactive** ad-hoc queries

Why a New Programming Model?

MapReduce greatly simplified big data analysis

But as soon as it got popular, users wanted more:

- » More **complex**, multi-stage applications (e.g. iterative **graph algorithms** and **machine learning**)
- » More **interactive** ad-hoc queries

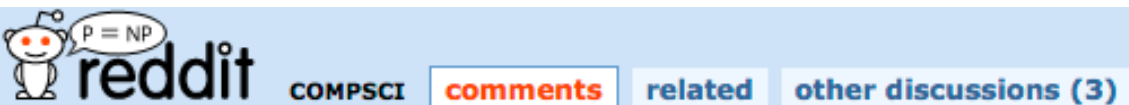
Require faster **data sharing** across parallel jobs

Is MapReduce dead? Not really.

Google Dumps MapReduce in Favor of New Hyper-Scale Analytics System

<http://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system/>

http://www.reddit.com/r/compsci/comments/296aqr/on_the_death_of_mapreduce_at_google/



↑ 87 On the Death of Map-Reduce at Google. (the-paper-trail.org)
submitted 3 months ago by qkdhfjdjdhd
↓ 20 comments share

all 20 comments

sorted by: best ▼

↑ [-] **tazzy531** 47 points 3 months ago

↓ As an employee, I was surprised by this headline, considering I just ran some mapreduces this past week.

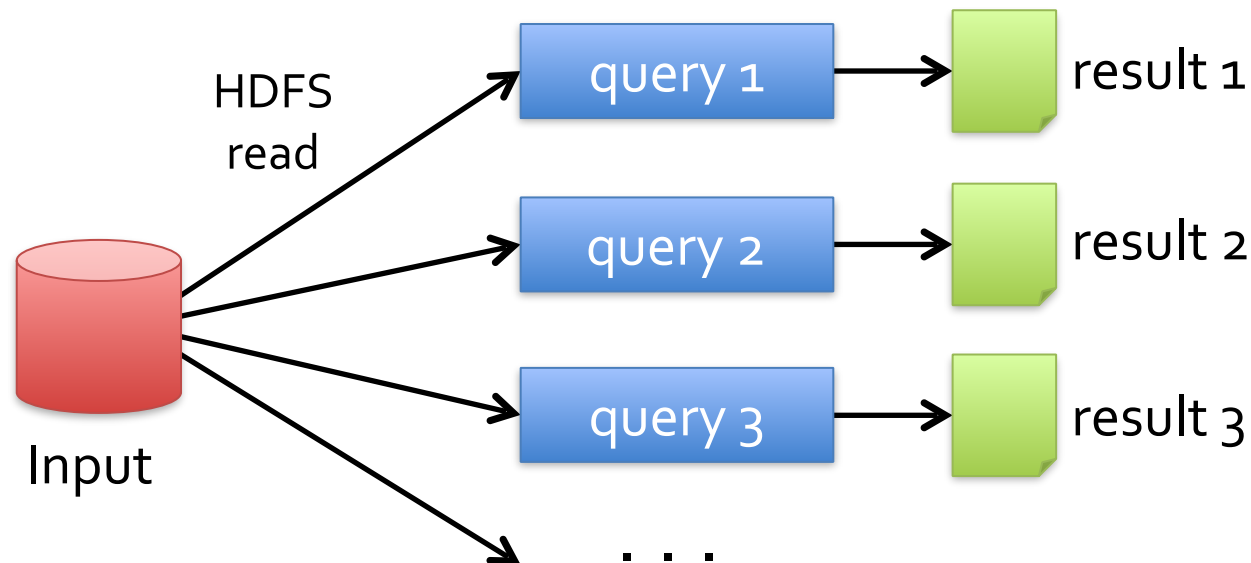
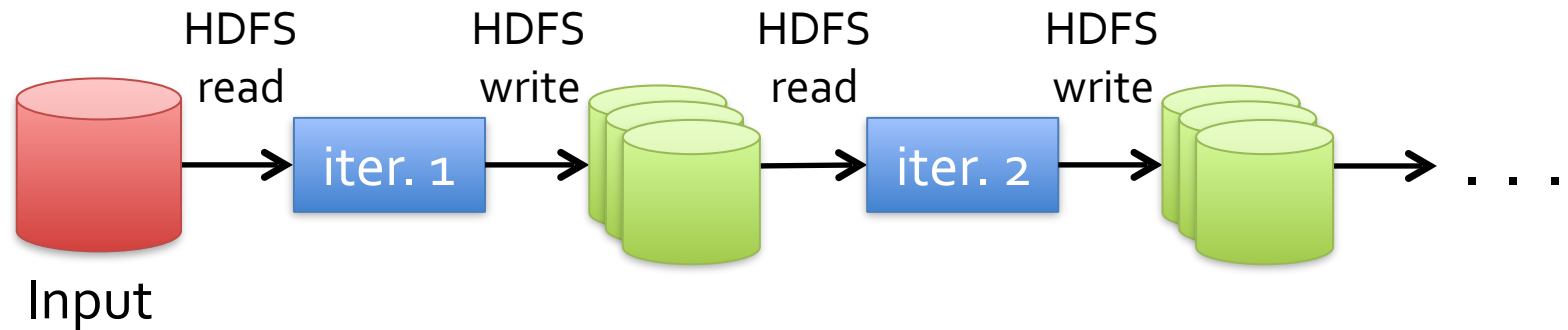
After digging further, this headline and article is rather inaccurate.

Cloud DataFlow is the external name for what is internally called Flume.

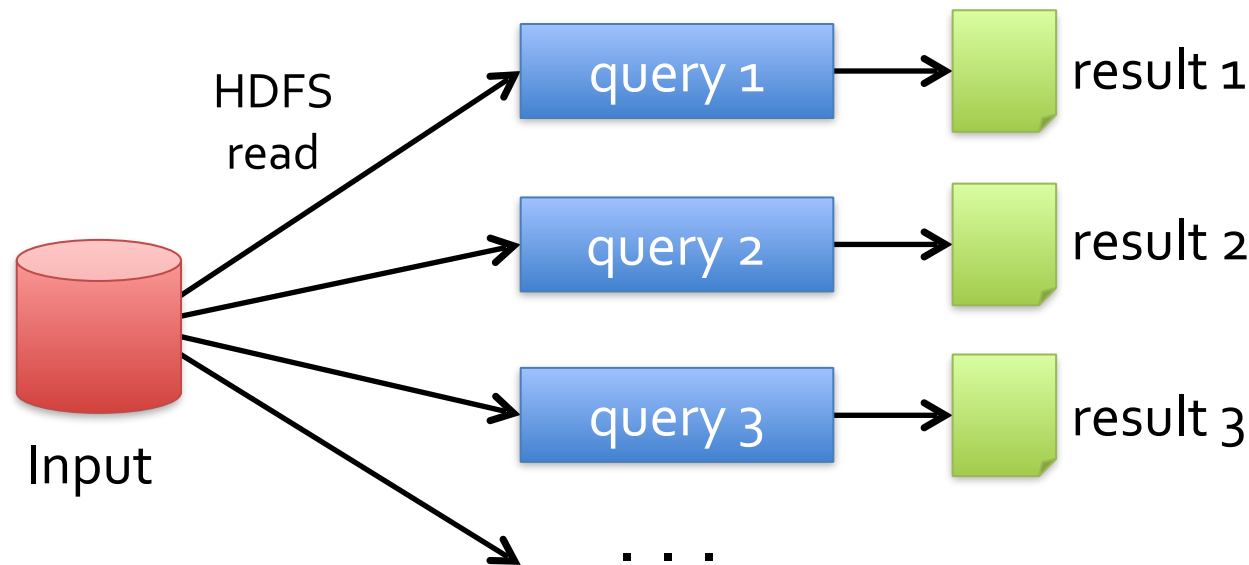
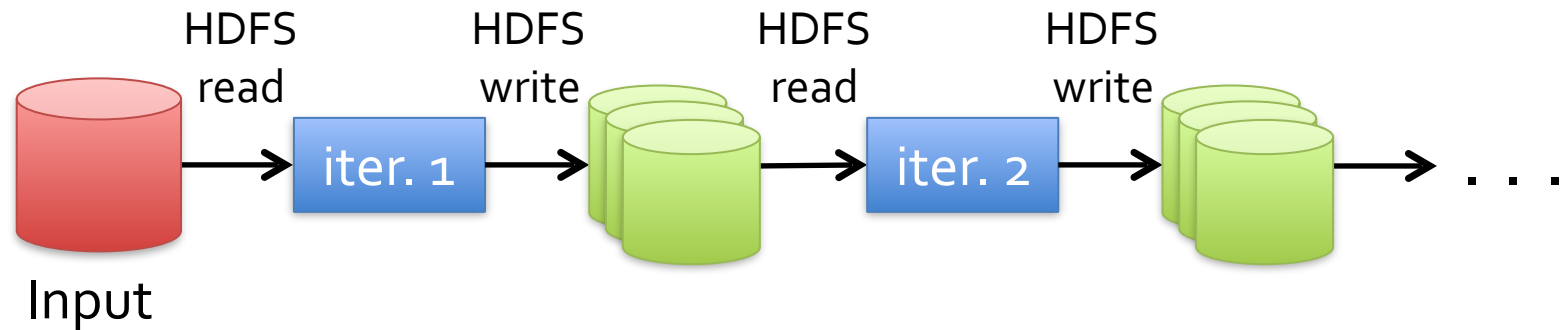
Flume is a layer that runs on top of MapReduce that abstracts away the complexity into something that is much easier



Data Sharing in MapReduce

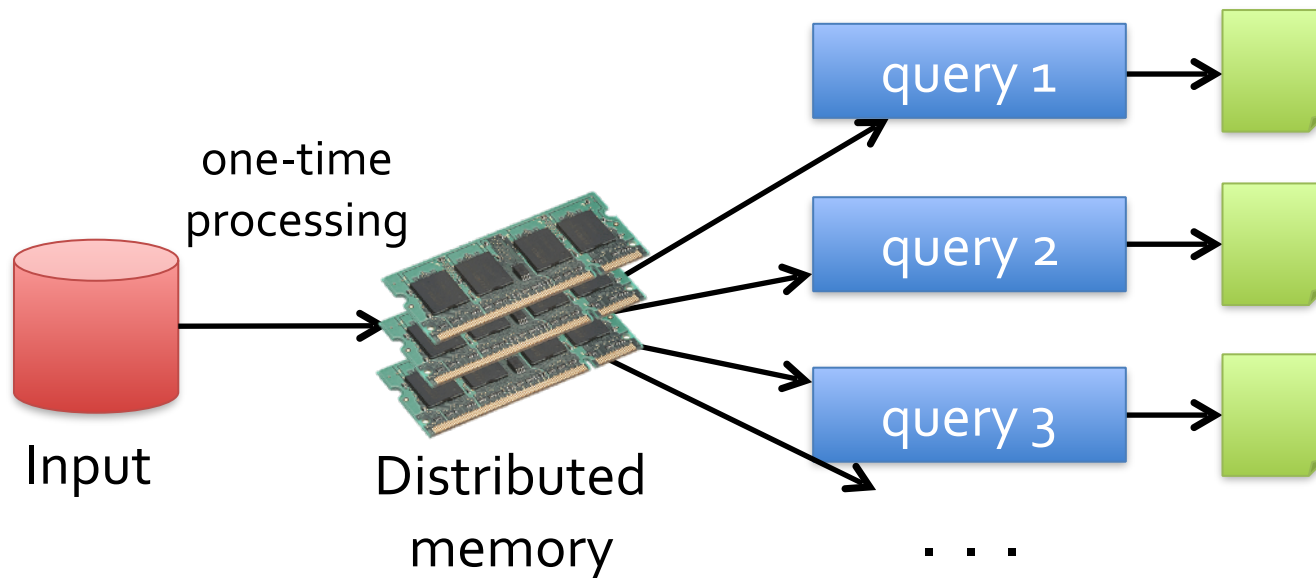
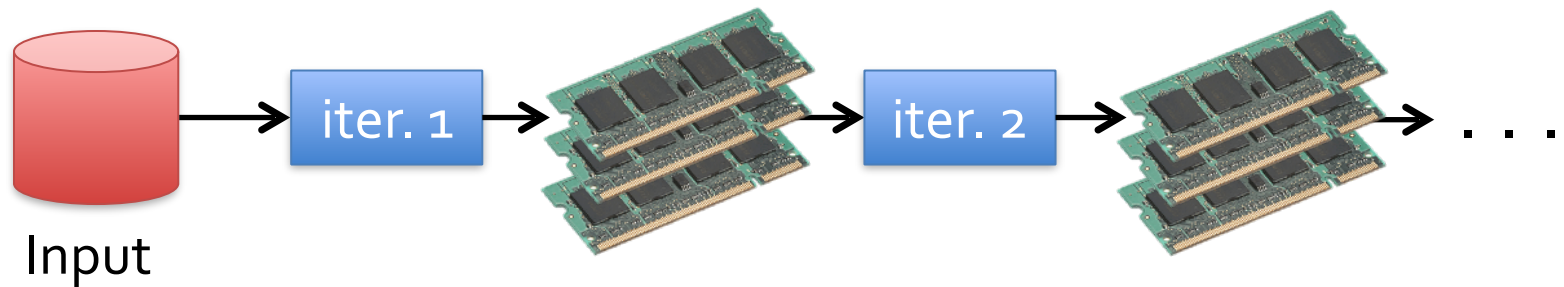


Data Sharing in MapReduce

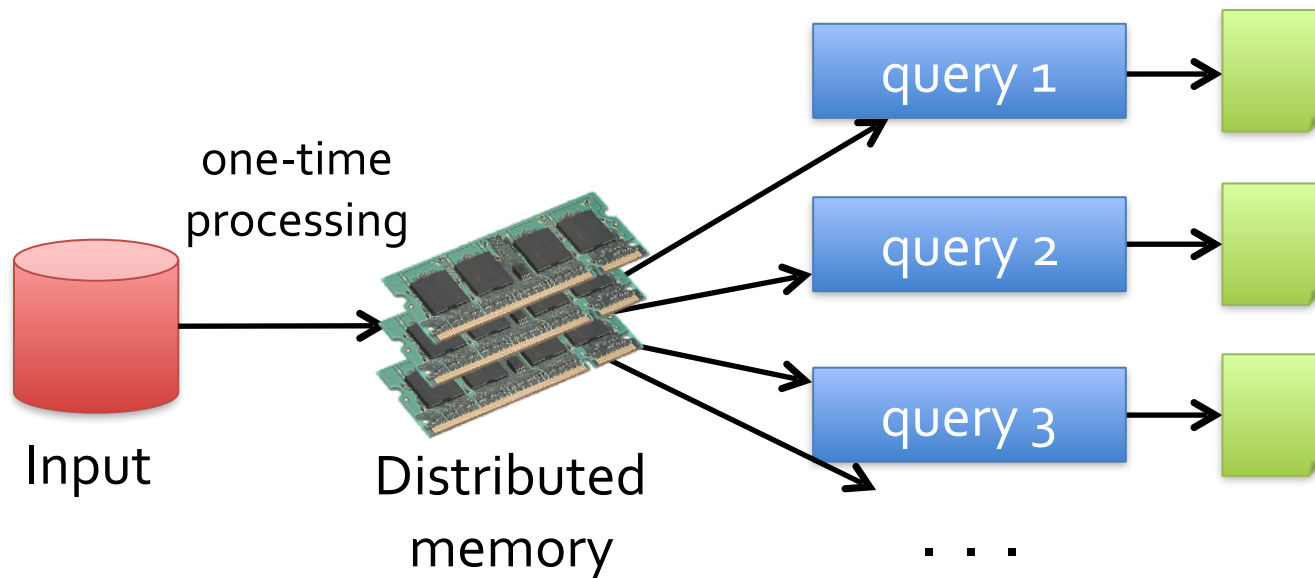
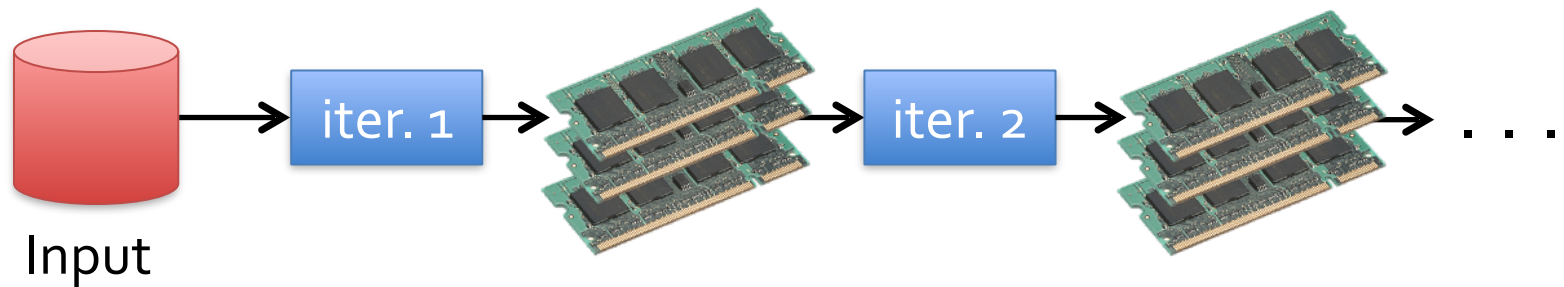


Slow due to replication, serialization, and disk IO

Data Sharing in Spark



Data Sharing in Spark



10-100× faster than network and disk

Spark Programming Model

Key idea: *resilient distributed datasets (RDDs)*

- » Distributed collections of objects that can be cached in memory across cluster nodes
- » Manipulated through various parallel operators
- » Automatically rebuilt on failure

Interface

- » Clean language-integrated API in Scala
- » Can be used *interactively* from Scala, Python console
- » Supported languages: Java, **Scala**, Python, R

<http://www.scala-lang.org/old/faq/4>

Functional programming in D3: <http://sleptons.blogspot.com/2015/01/functional-programming-d3js-good-example.html>

Scala vs Java 8: <http://kukuruku.co/hub/scala/java-8-vs-scala-the-difference-in-approaches-and-mutual-innovations>



[DOCUMENTATION](#)

[DOWNLOAD](#)

[COMMUNITY](#)

[CONTRIBUTE](#)



Object-Oriented Meets Functional

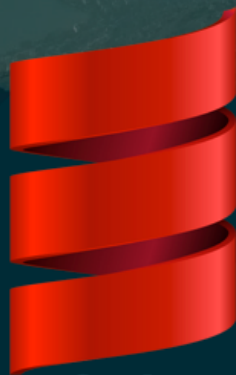
Have the best of both worlds. Construct elegant class hierarchies for maximum code reuse and extensibility, implement their behavior using higher-order functions. Or anything in-between.

[LEARN MORE](#)

[DOWNLOAD](#)

[Getting Started](#)

- [Milestones, nightlies, etc.](#)
- [All Previous Releases](#)



[API DOCS](#)

[API: Current | Nightly](#)

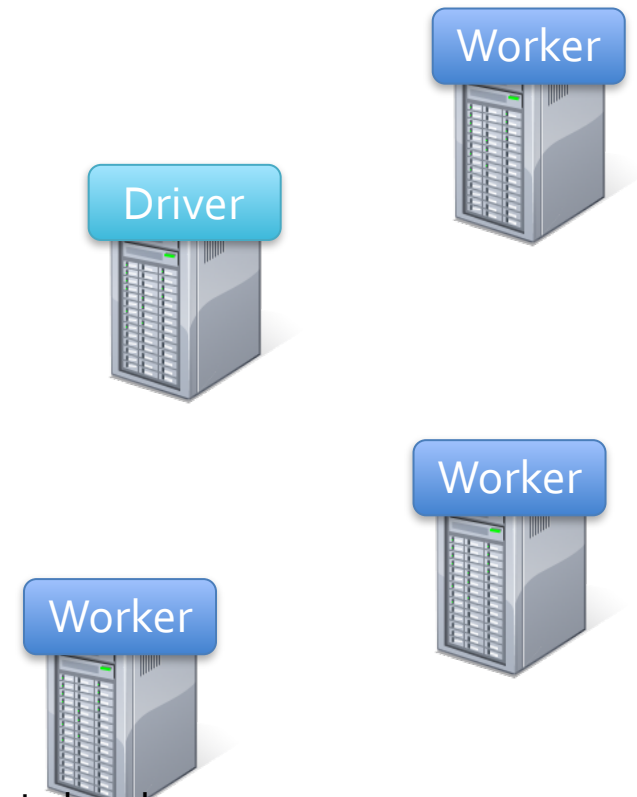
- [All Previous API Docs](#)
- [Scala Documentation](#)
- [Language Specification](#)

Example: Log Mining

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

Example: Log Mining

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



Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```



Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

Base RDD

Driver

Worker

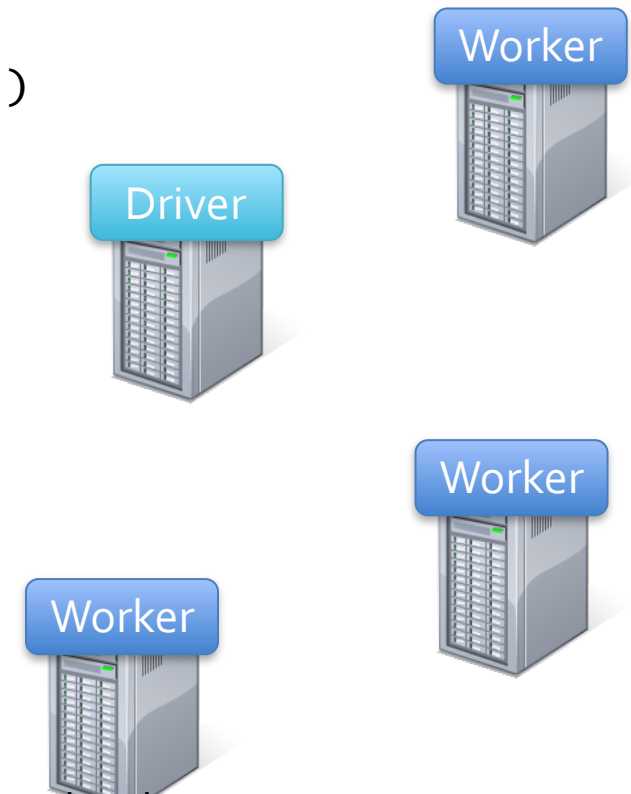
Worker

Worker

Example: Log Mining

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

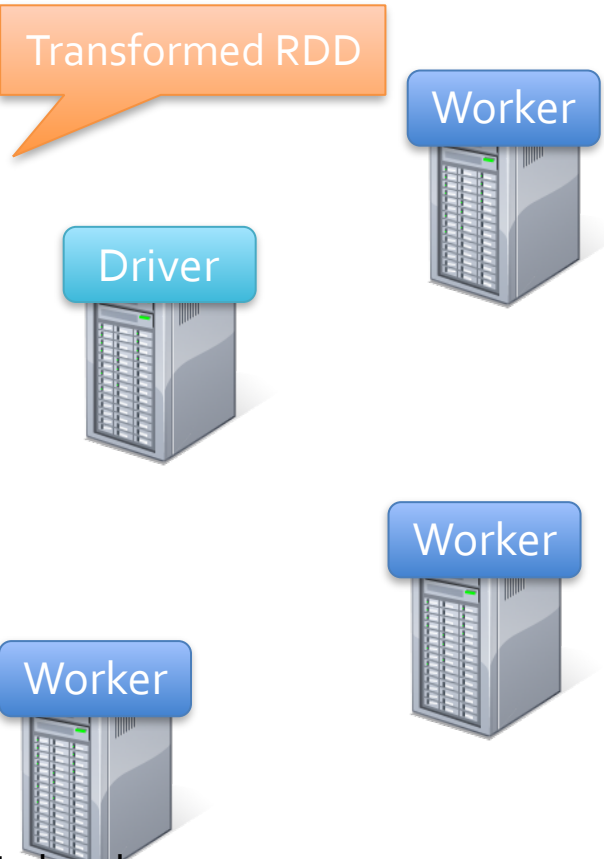
```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```



Example: Log Mining

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

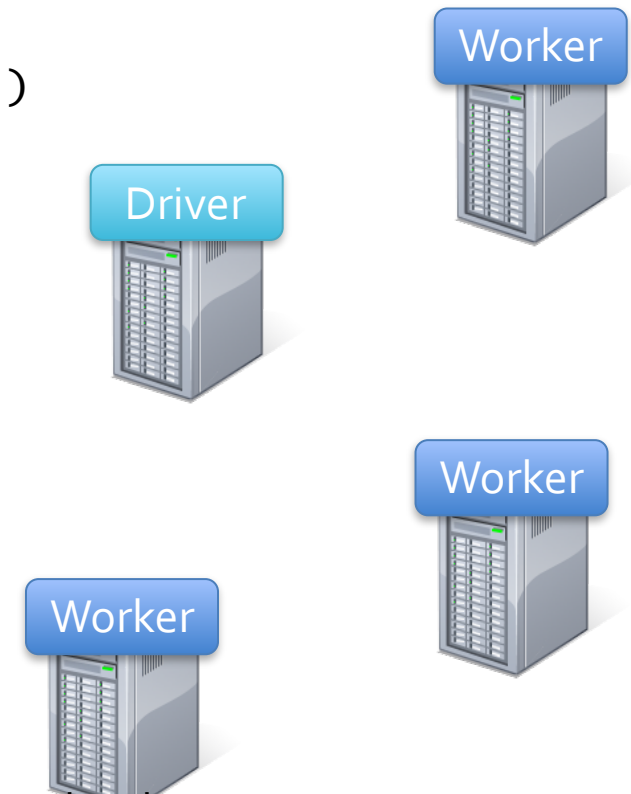
```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```



Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```



Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```

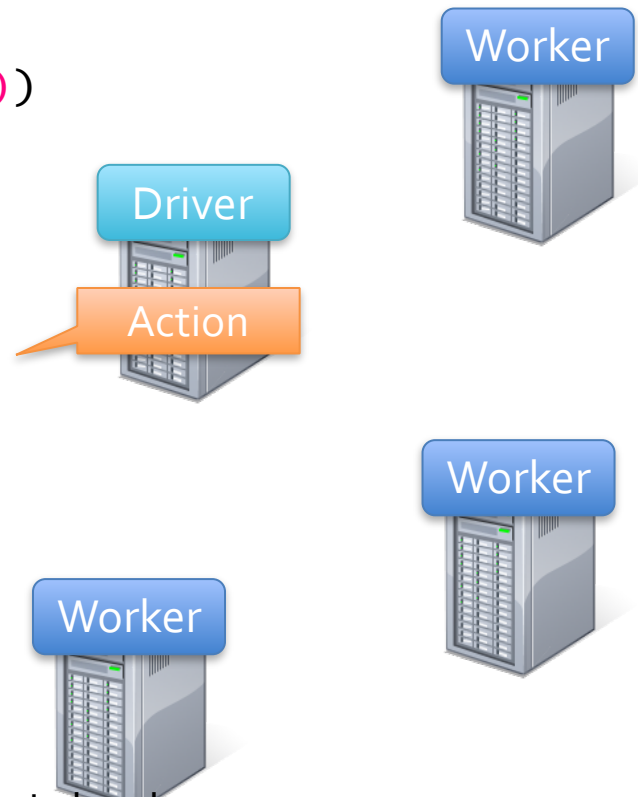


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```



Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```

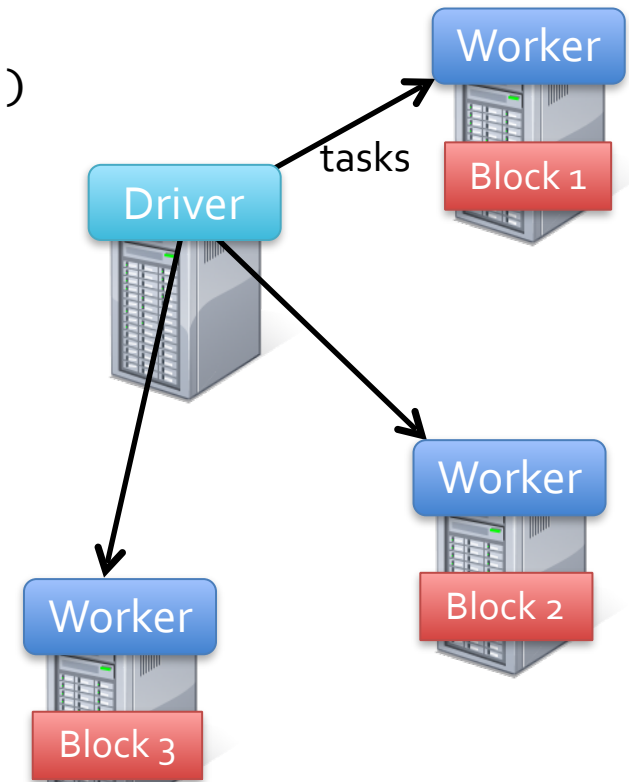


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```

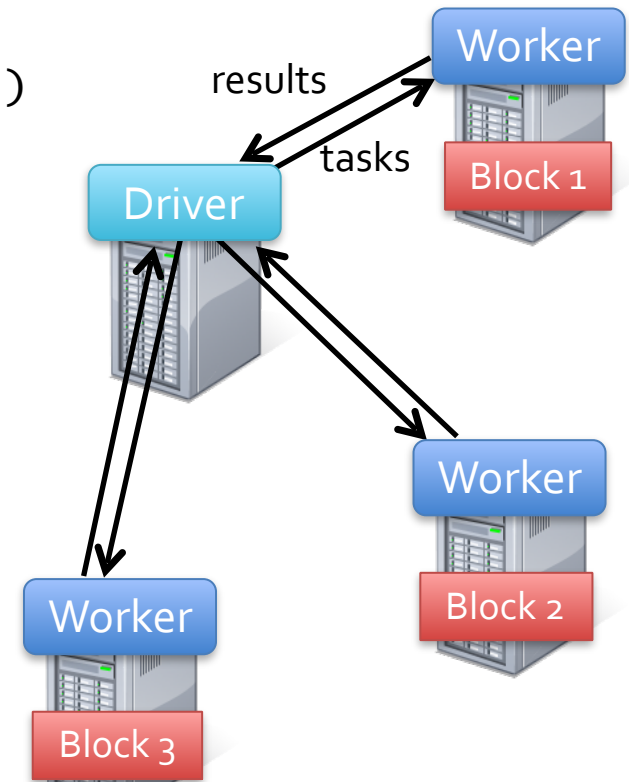


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```

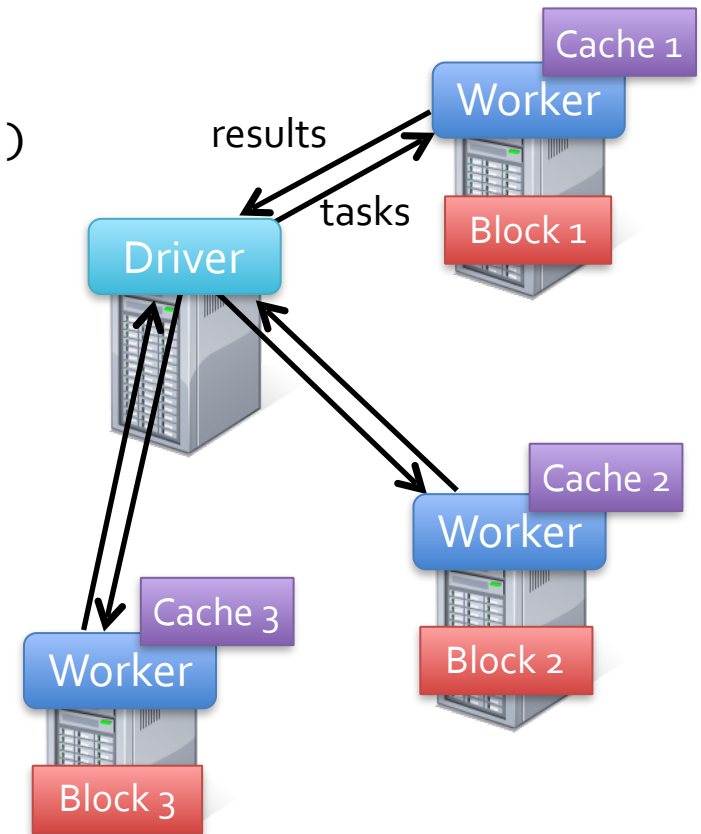


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```

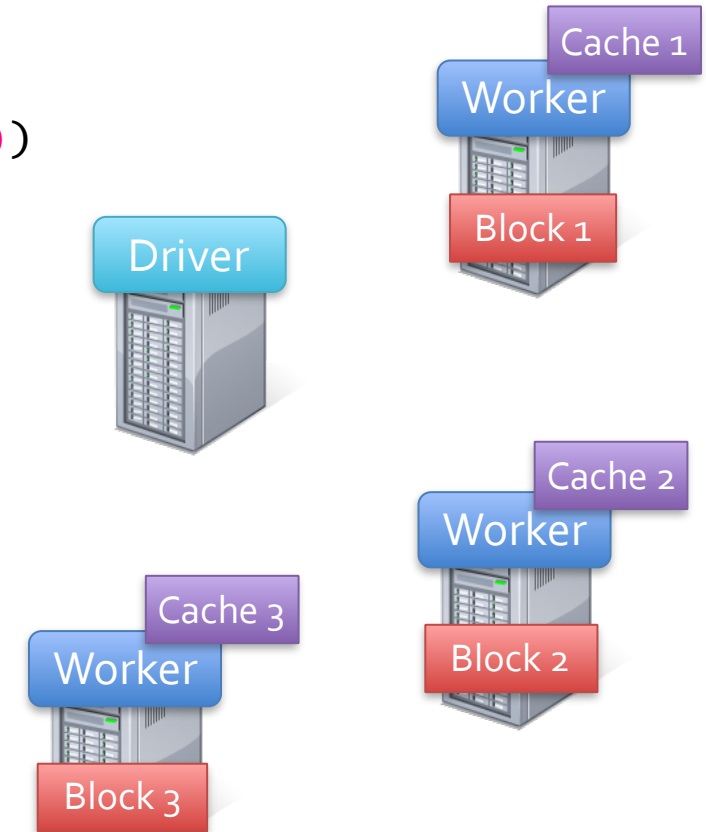


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
```

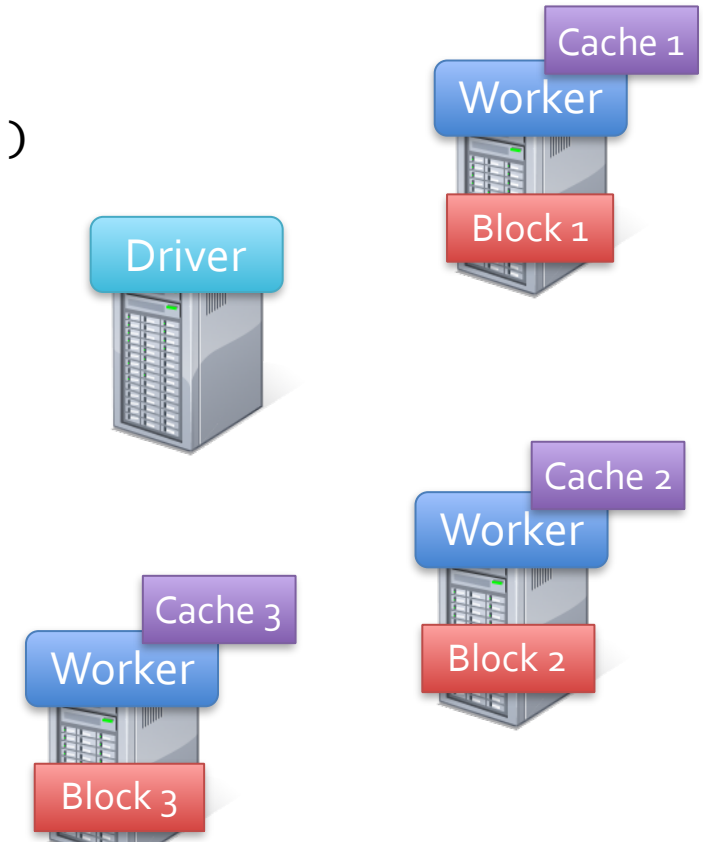


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count  
cachedMsgs.filter(_.contains("bar")).count
```

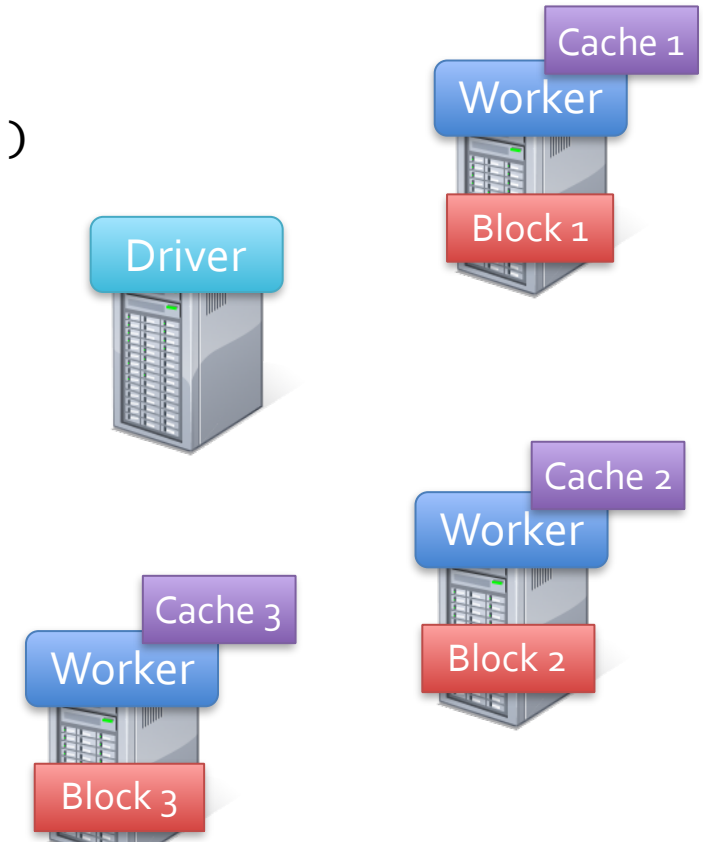


Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(_.startsWith("ERROR"))  
messages = errors.map(_.split('\t')(2))  
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count  
cachedMsgs.filter(_.contains("bar")).count  
... .
```



Example: Log Mining

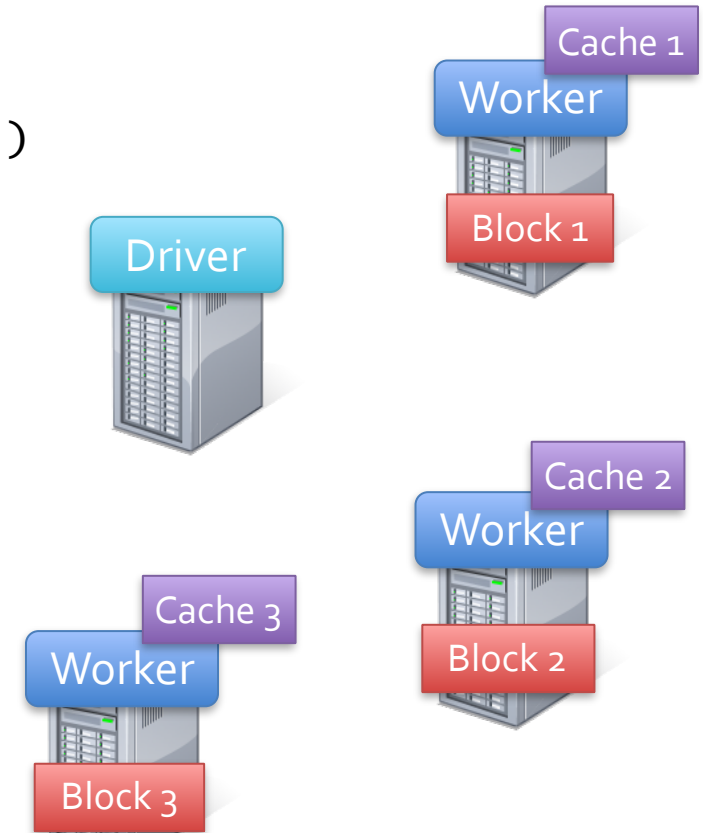
Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

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

<http://ananthakumaran.in/2010/03/29/scala-underscore-magic.html>
<http://www.slideshare.net/normation/scala-dreaded>



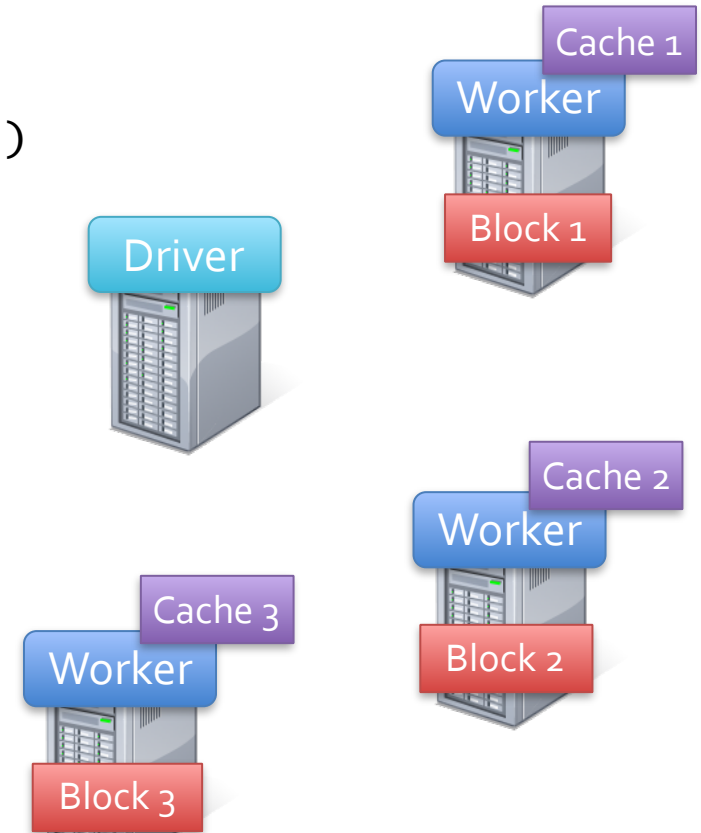
Example: Log Mining

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

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()
```

```
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

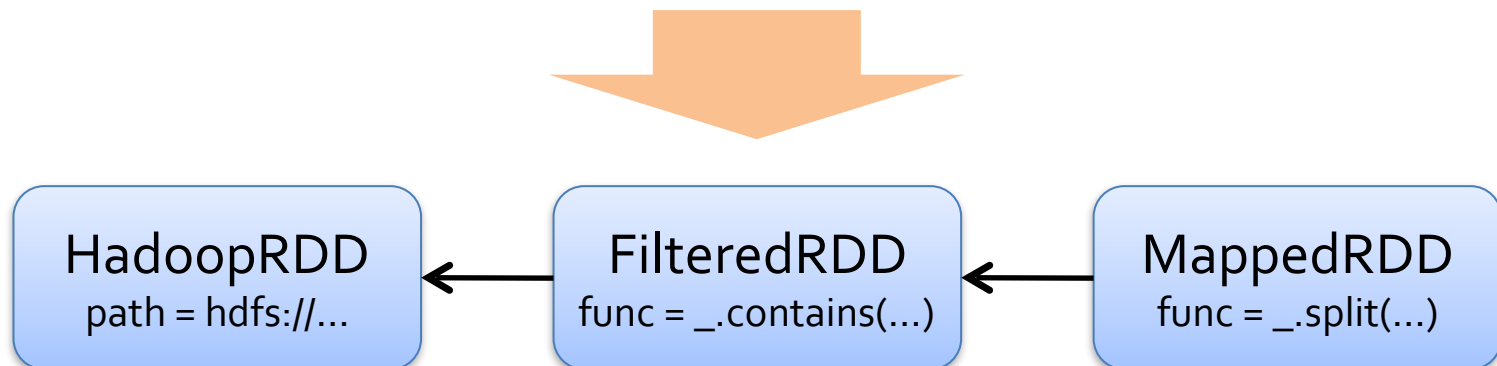
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



Fault Tolerance

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

E.g: `messages = textFile(...).filter(_.contains("error")).map(_.split('\t')(2))`



Example: Logistic Regression

```
val data = spark.textFile(...).map(readPoint).cache()
var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Example: Logistic Regression


```
val data = spark.textFile(...).map(readPoint).cache()
var w = Vector.random(D)
for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}
println("Final w: " + w)
```



Load data in memory once

Example: Logistic Regression

```
val data = spark.textFile(...).map(readPoint).cache()
var w = Vector.random(D)
for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}
println("Final w: " + w)
```



Initial parameter vector

Example: Logistic Regression

```
val data = spark.textFile(...).map(readPoint).cache()
var w = Vector.random(D)

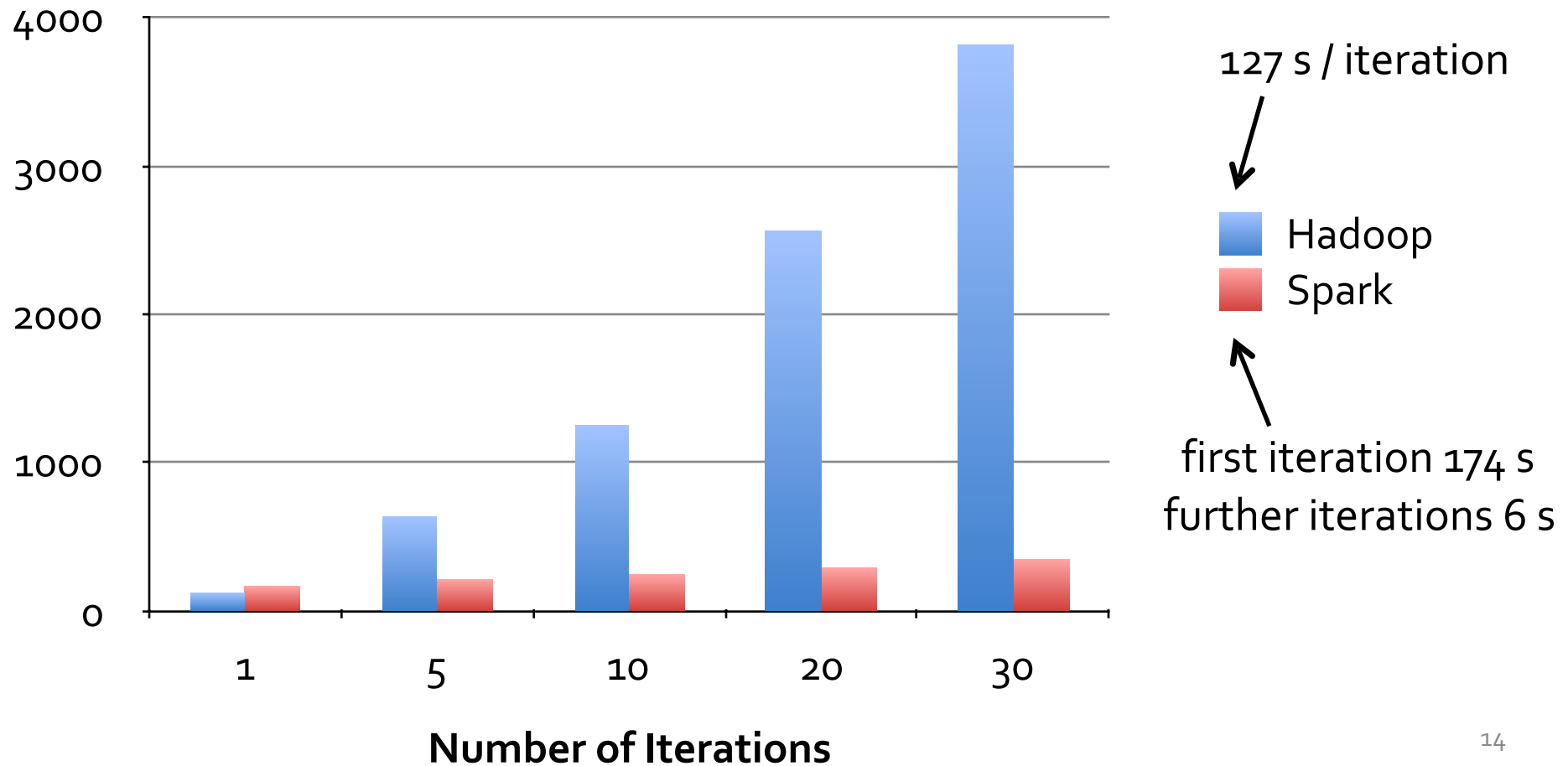
for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```



Repeated MapReduce steps
to do gradient descent

Logistic Regression Performance



Supported Operators

map	reduce	sample
filter	count	cogroup
groupBy	reduceByKey	take
sort	groupByKey	partitionBy
join	first	pipe
leftOuterJoin	union	save
rightOuterJoin	cross	...

Spark Users

CONVIVA®

foursquare

quantifind

KLOUT

YAHOO!
RESEARCH

University of California
Berkeley



PRINCETON
UNIVERSITY

UCSF

Spark SQL: Hive on Spark

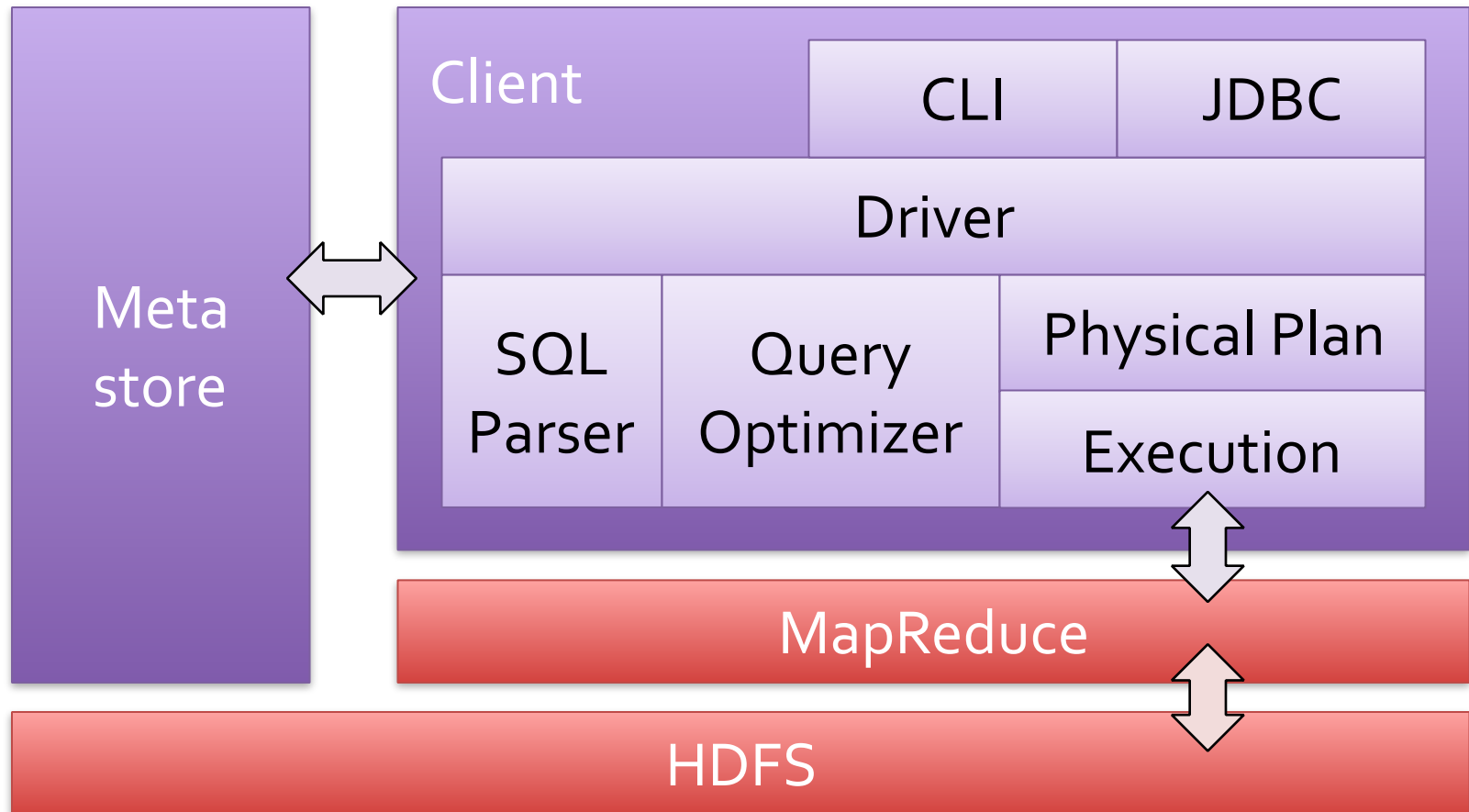
Motivation

Hive is great, but Hadoop's execution engine makes even the smallest queries take minutes

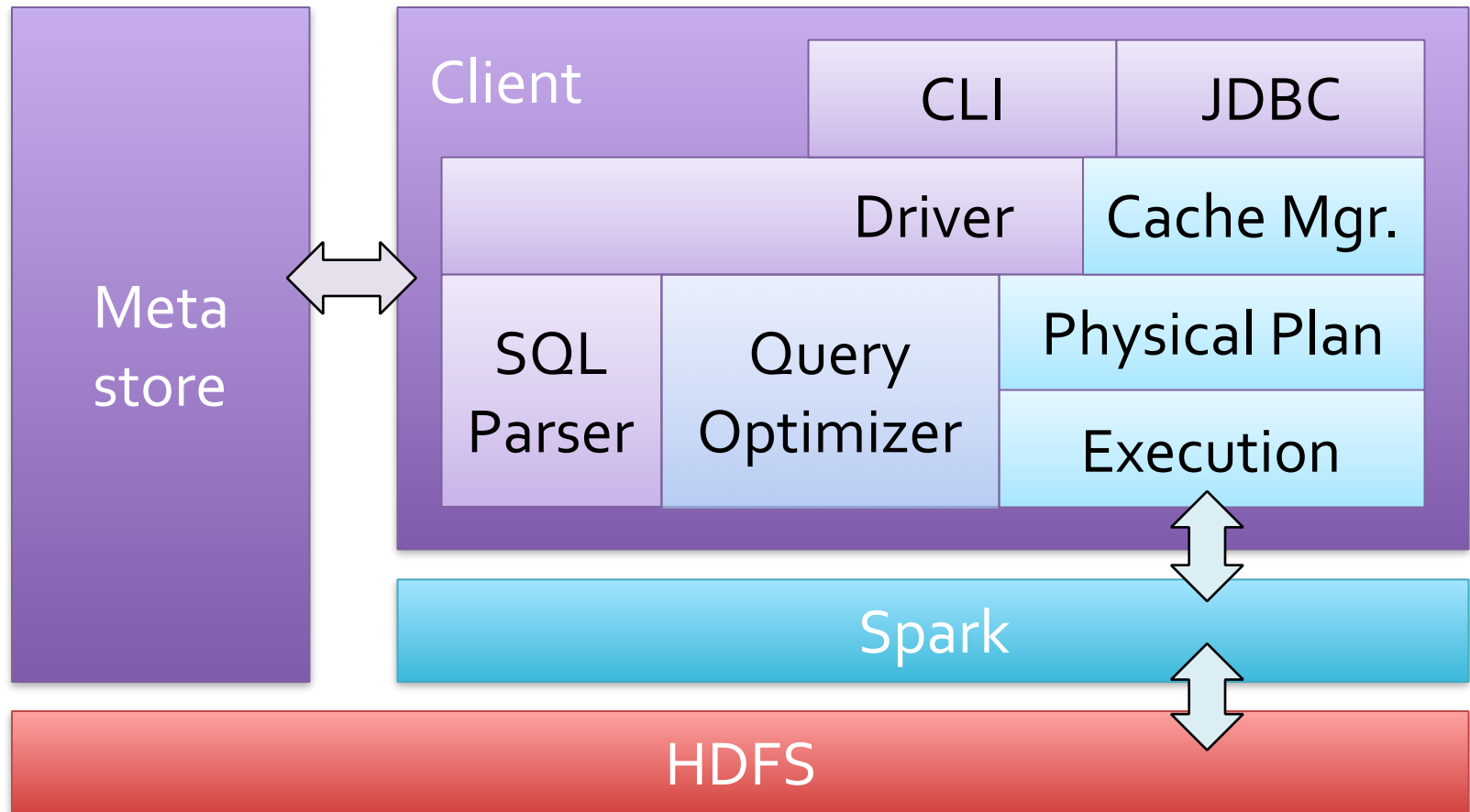
Scala is good for programmers, but many data users only know SQL

Can we extend Hive to run on Spark?

Hive Architecture



Spark SQL Architecture



Using Spark SQL

```
CREATE TABLE mydata_cached AS SELECT ...
```

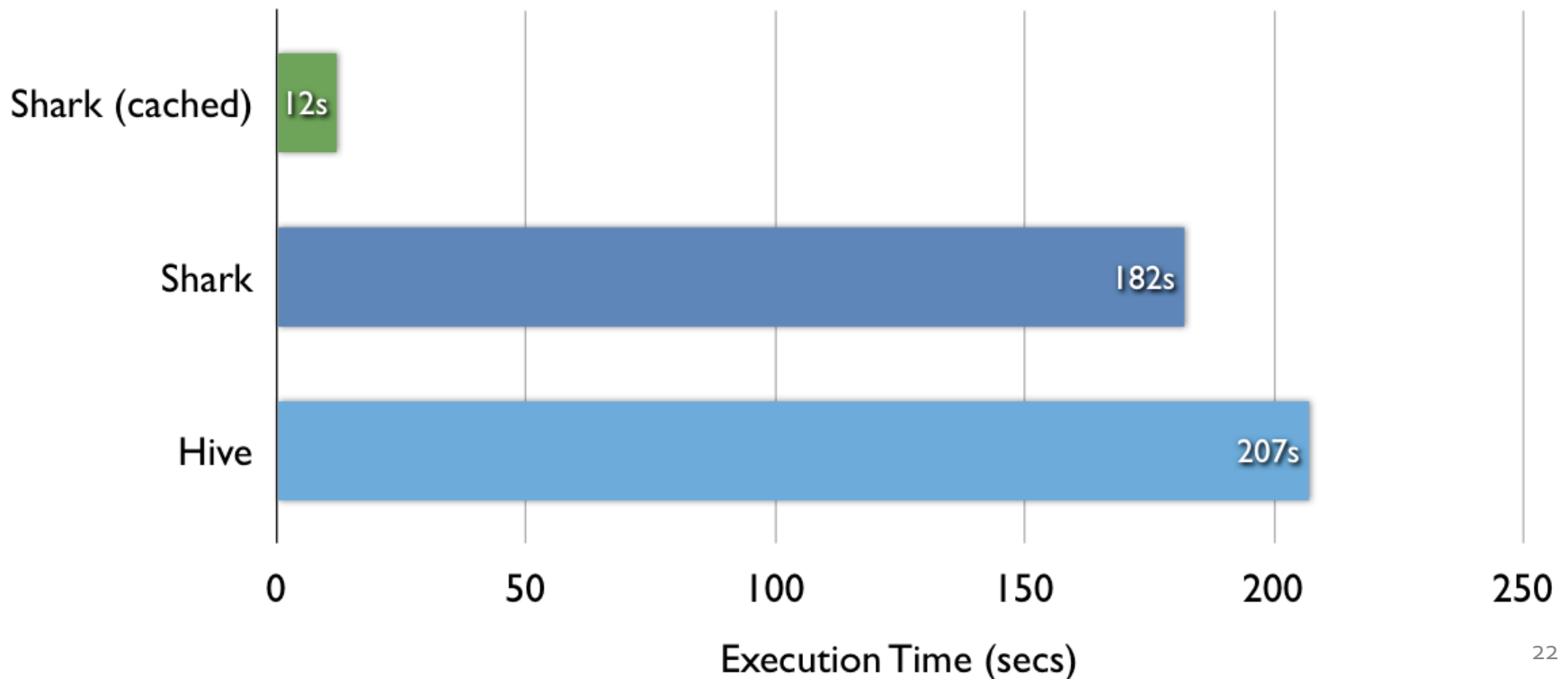
Run standard HiveQL on it, including UDFs

» A few esoteric features are not yet supported

Can also call from Scala to mix with Spark

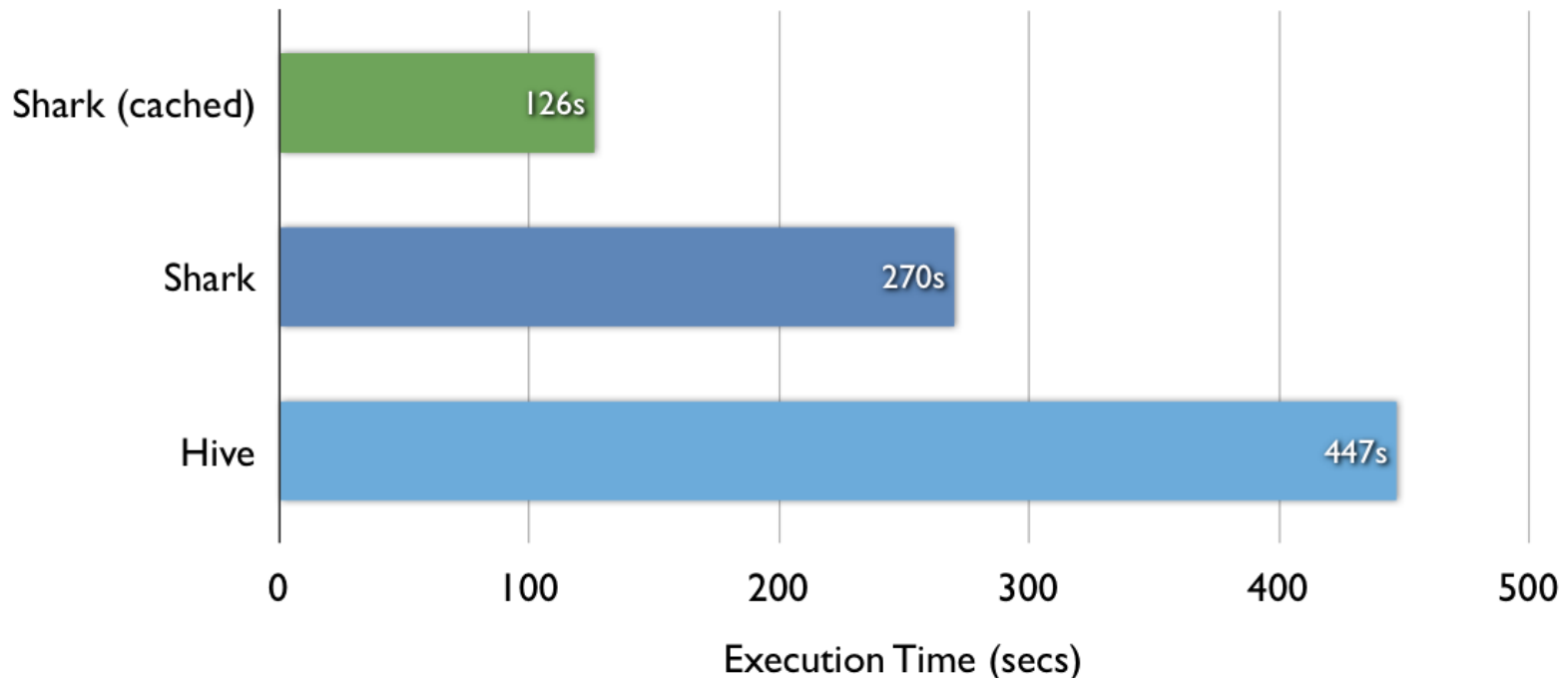
Benchmark Query 1

```
SELECT * FROM grep WHERE field LIKE '%XYZ%';
```

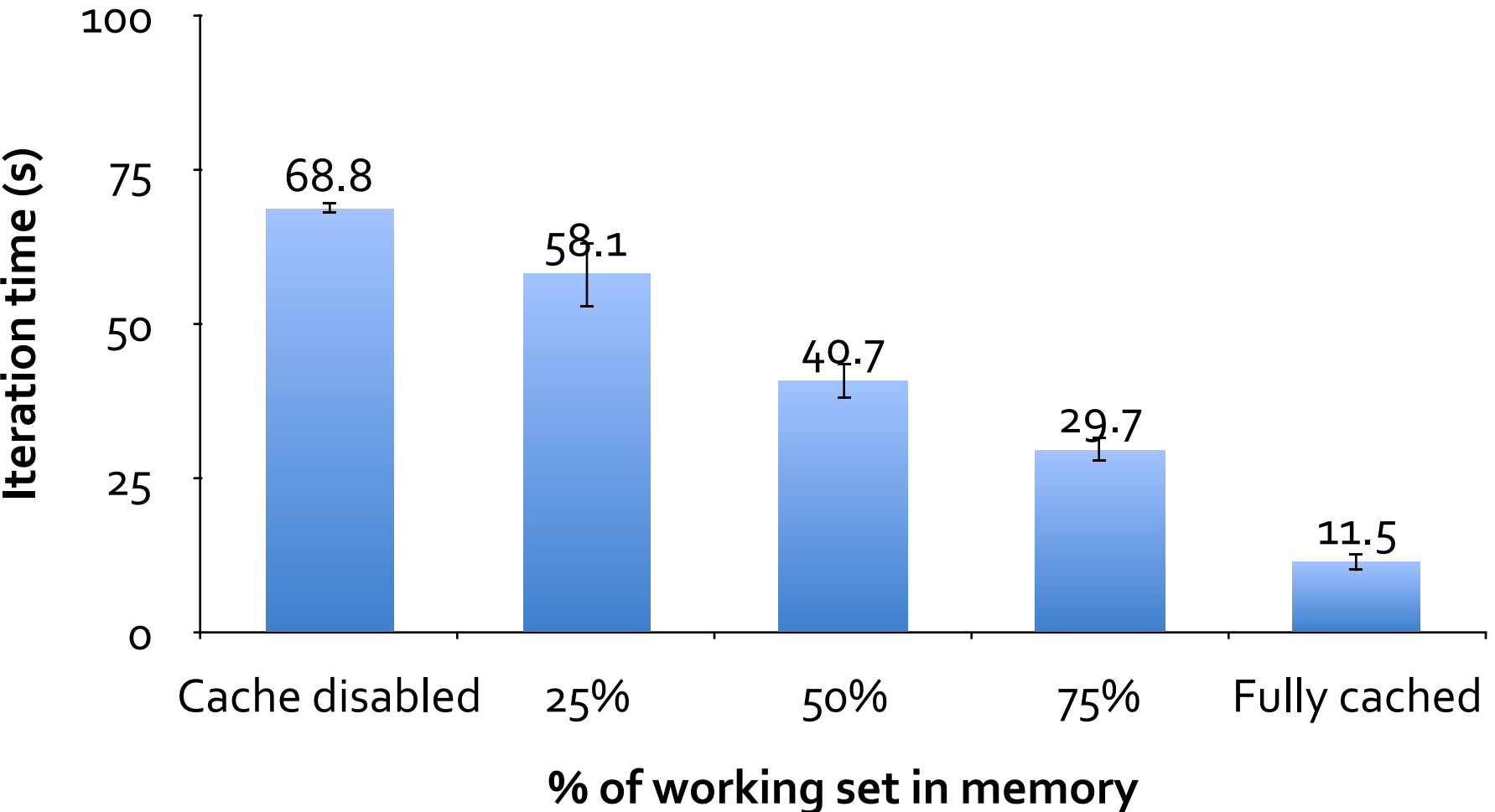


Benchmark Query 2

```
SELECT sourceIP, AVG(pageRank), SUM(adRevenue) AS earnings  
FROM rankings AS R, userVisits AS V ON R.pageURL = V.destURL  
WHERE V.visitDate BETWEEN '1999-01-01' AND '2000-01-01'  
GROUP BY V.sourceIP  
ORDER BY earnings DESC  
LIMIT 1;
```



Behavior with Not Enough RAM



What's Next?

Recall that Spark's model was motivated by two emerging uses (interactive and multi-stage apps)

Another emerging use case that needs fast data sharing is **stream processing**

- » Track and update state in memory as events arrive
- » Large-scale reporting, click analysis, spam filtering, etc

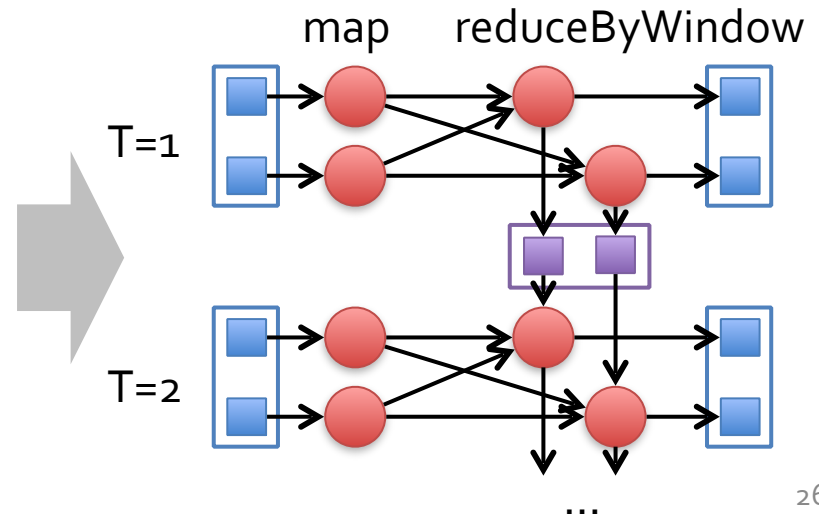
Streaming Spark

Extends Spark to perform streaming computations

Runs as a series of small (~1 s) batch jobs, keeping state in memory as fault-tolerant RDDs

Intermix seamlessly with batch and ad-hoc queries

```
tweetStream  
  .flatMap(_.toLowerCase.split)  
  .map(word => (word, 1))  
  .reduceByWindow("5s", _ + _)
```



map() vs flatMap()

The best explanation:

<https://www.linkedin.com/pulse/difference-between-map-flatmap-transformations-spark-pyspark-pandey>

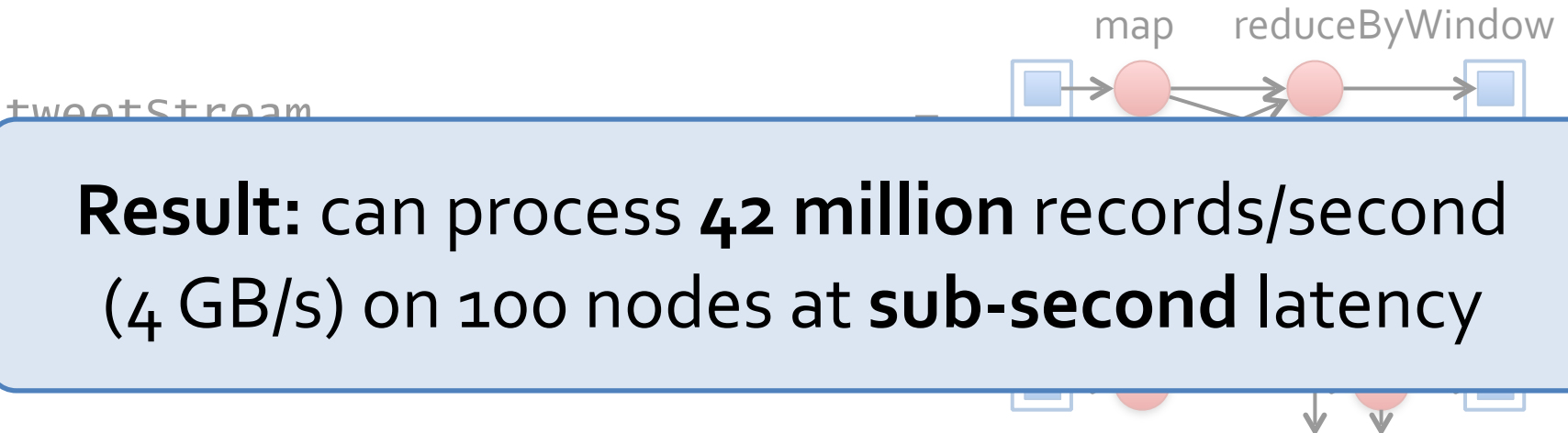
flatMap = map + flatten

Streaming Spark

Extends Spark to perform streaming computations

Runs as a series of small (~1 s) batch jobs, keeping state in memory as fault-tolerant RDDs

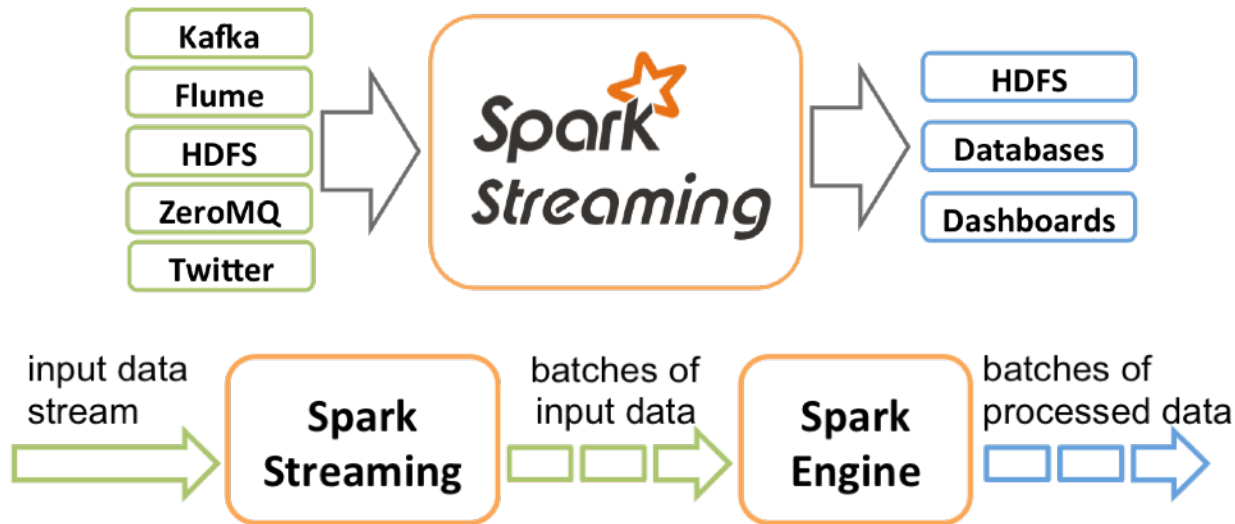
Intermix seamlessly with batch and ad-hoc queries



Result: can process **42 million** records/second
(4 GB/s) on 100 nodes at **sub-second** latency

Spark Streaming

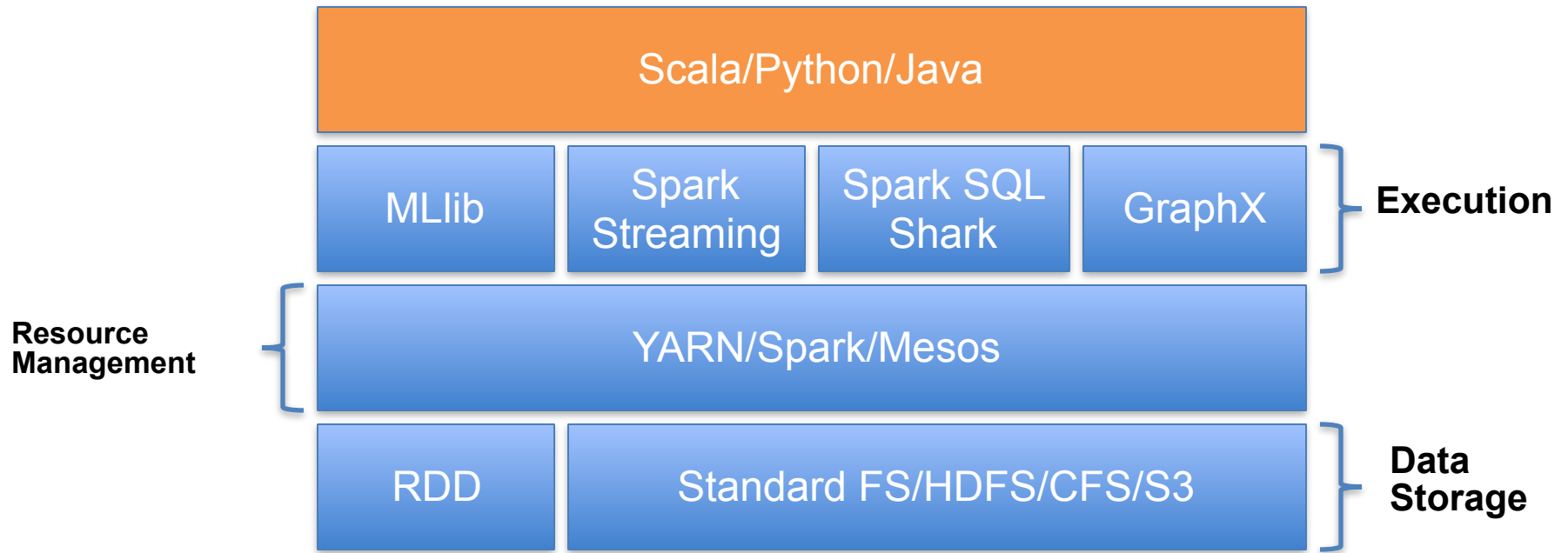
Create and operate on RDDs from live data streams at set intervals



Data is divided into batches for processing

Streams may be combined as a part of processing or analyzed with higher level transforms

SPARK PLATFORM



GraphX

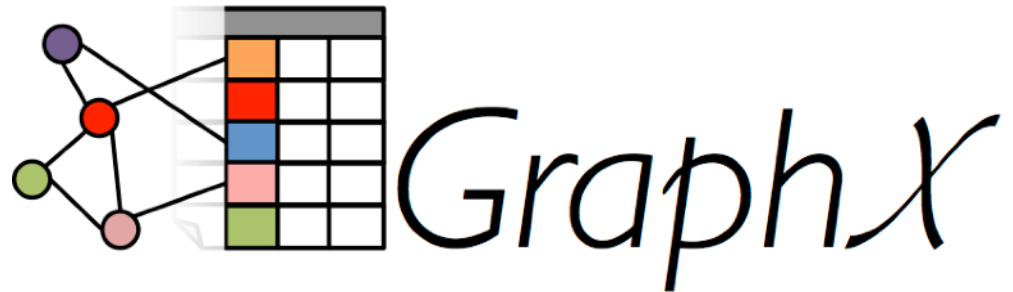
Parallel graph processing

Extends RDD -> Resilient Distributed Property Graph

- » Directed multigraph with properties attached to each vertex and edge

Limited algorithms

- » PageRank
- » Connected Components
- » Triangle Counts



Alpha component

MLlib

Scalable machine learning library

Interoperates with NumPy

Available algorithms in **1.0**

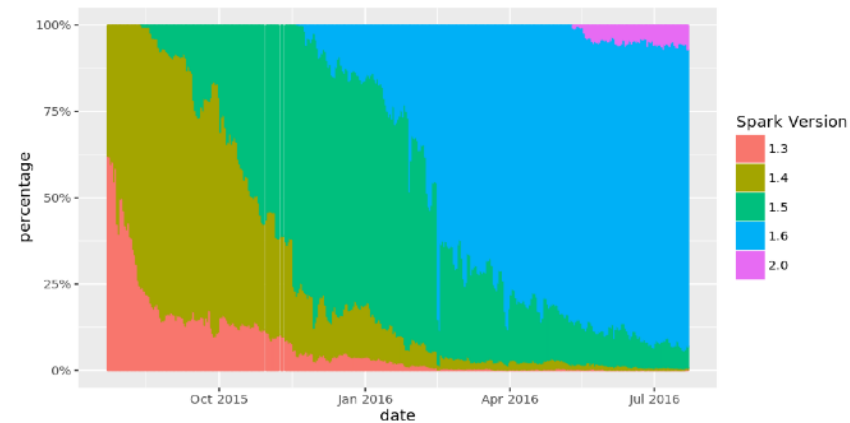
- » Linear Support Vector Machine (SVM)
- » Logistic Regression
- » Linear Least Squares
- » Decision Trees
- » Naïve Bayes
- » Collaborative Filtering with ALS
- » K-means
- » Singular Value Decomposition
- » Principal Component Analysis
- » Gradient Descent

MLlib 2.0 (part of Spark 2.0)

- Basic statistics
 - summary statistics
 - correlations
 - stratified sampling
 - hypothesis testing
 - streaming significance testing
 - random data generation
- Classification and regression
 - linear models (SVMs, logistic regression, linear regression)
 - naive Bayes
 - decision trees
 - ensembles of trees (Random Forests and Gradient-Boosted Trees)
 - isotonic regression
- Collaborative filtering
 - alternating least squares (ALS)
- Clustering
 - k-means
 - Gaussian mixture
 - power iteration clustering (PIC)
 - latent Dirichlet allocation (LDA)
 - bisecting k-means
 - streaming k-means
- Dimensionality reduction
 - singular value decomposition (SVD)
 - principal component analysis (PCA)
- Feature extraction and transformation
- Frequent pattern mining
 - FP-growth
 - association rules
 - PrefixSpan
- Evaluation metrics
- PMML model export
- Optimization (developer)
 - stochastic gradient descent
 - limited-memory BFGS (L-BFGS)

Spark 2

(very new still)



New feature highlights

<https://databricks.com/blog/2016/07/26/introducing-apache-spark-2-0.html>

Spark 2.0.0 **has API breaking changes**

Partly why HW3 uses Spark 1.6 (also, Cloudera distribution's Spark 2 support is in beta)

More details: <https://spark.apache.org/releases/spark-release-2-0-0.html>