poloclub.github.io/#cse6242

CSE6242/CX4242: Data & Visual Analytics

Spark & Spark SQL

Duen Horng (Polo) Chau
Associate Professor, College of Computing
Associate Director, MS Analytics
Georgia Tech

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

What is  ?

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

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, …

http://en.wikipedia.org/wiki/Apache_Spark

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

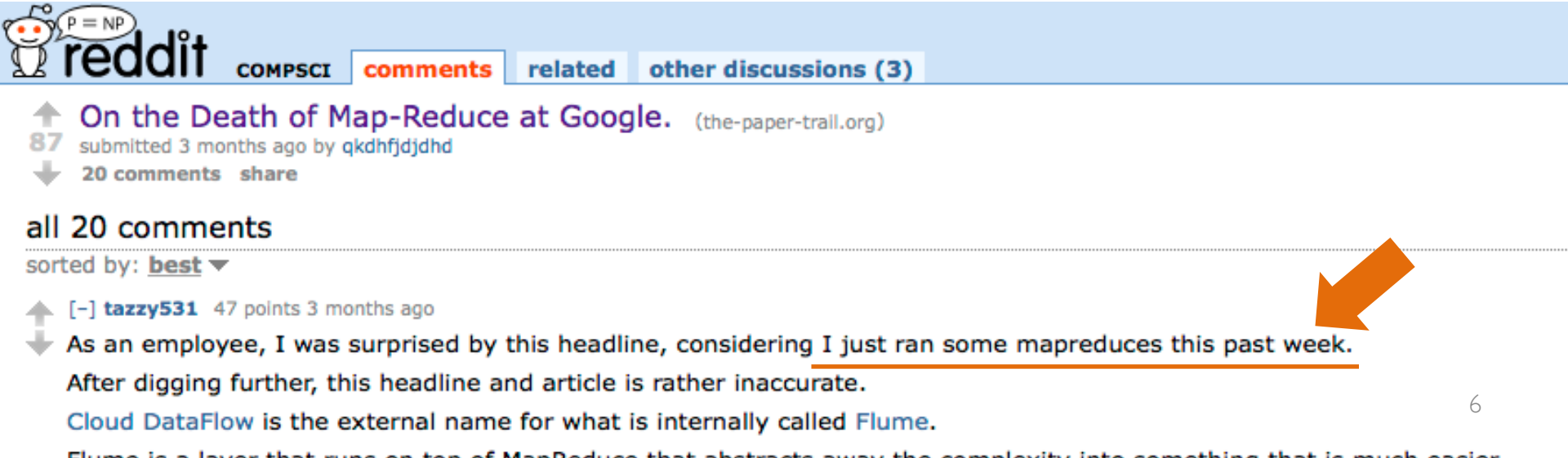
Require faster **data sharing** across parallel jobs

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/



Is Hadoop still Relephant?

https://www.linkedin.com/pulse/18-years-later-hadoop-still-relephant-ramon-chen

While newer technologies offer alternative approaches to big data management, Hadoop's distributed nature, scalability, and integration capabilities ensure its relevance in diverse use cases.

… Hadoop will remain a valuable component together with new technologies that continue to emerge, enabling businesses to process, store, and analyze vast amounts of data efficiently.

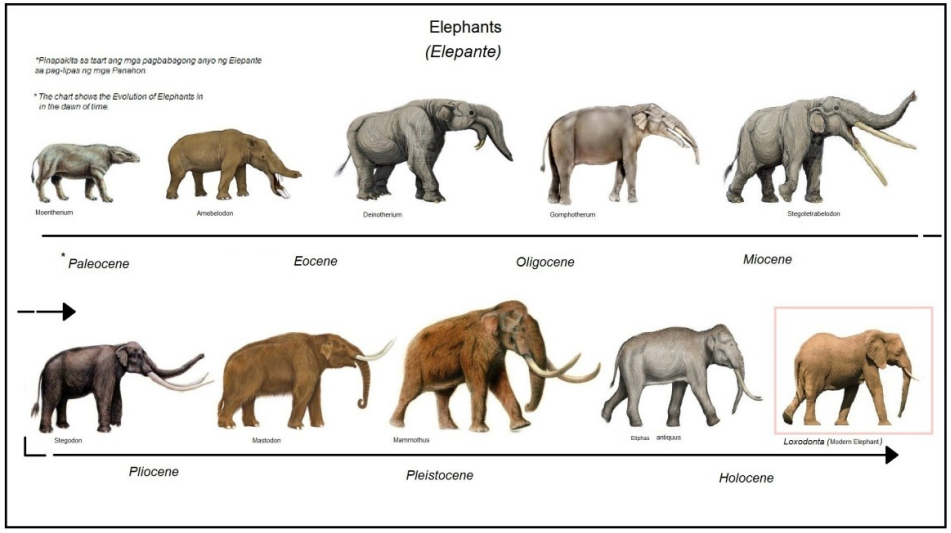
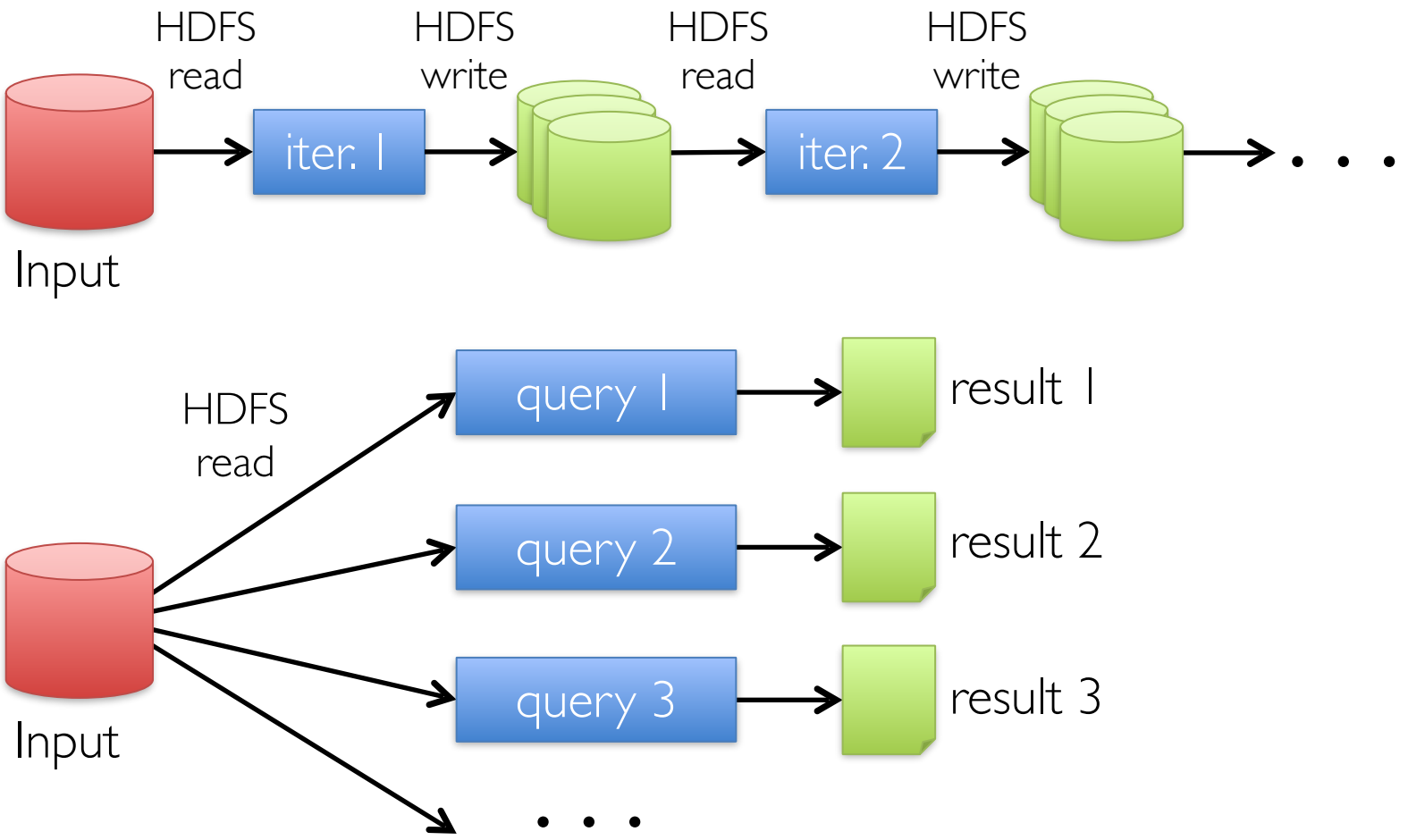


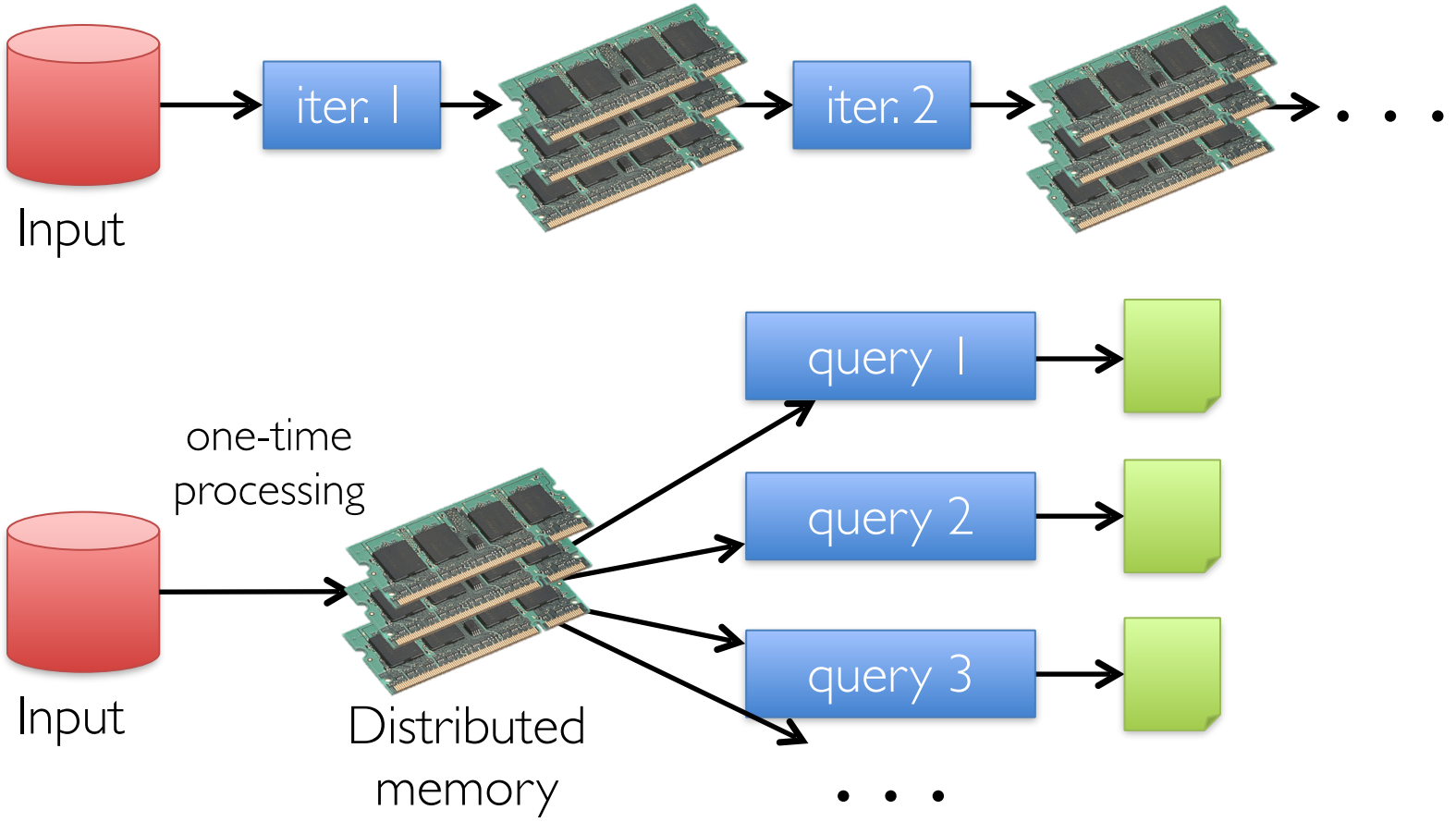
Image source: https://sites.google.com/site/evolutionoftheelephant/ances
Idea for using this image comes from the LinkedIn post.

Data Sharing in MapReduce



Slow due to replication, serialization, and disk IO

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

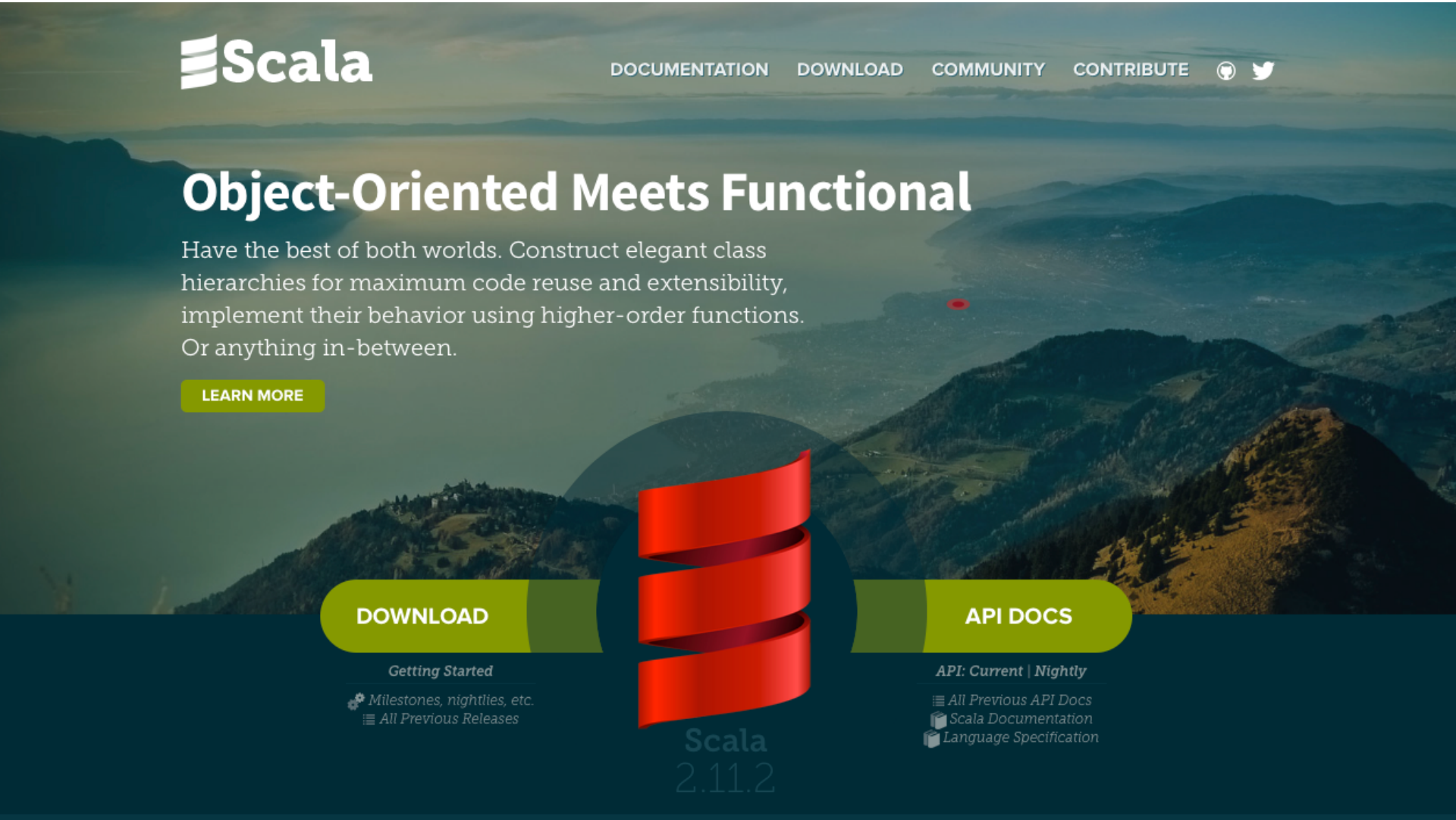
Interface

- » 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



Example: Log Mining

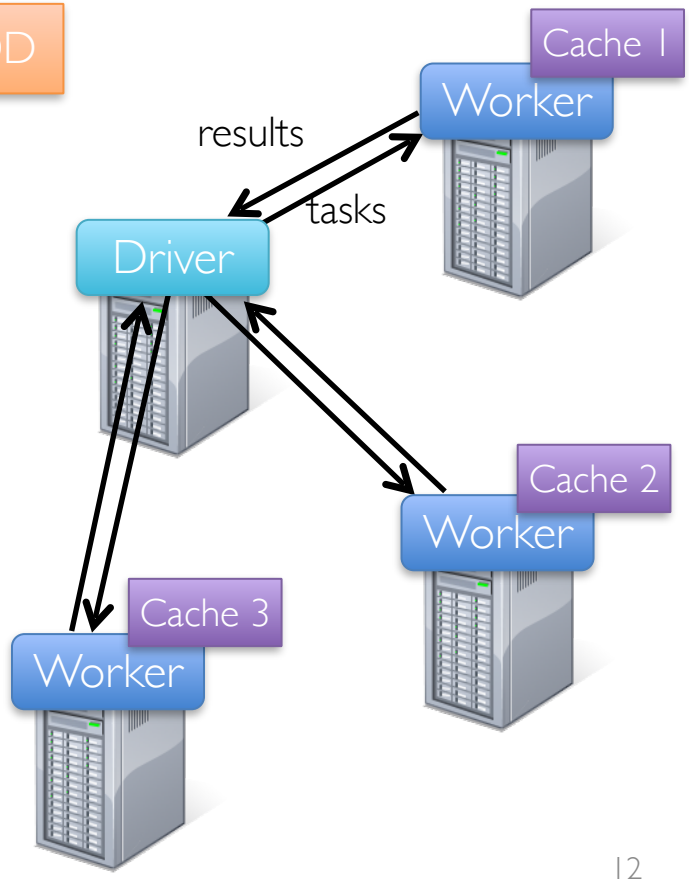
Load error messages from a log into memory, then interactively search for various patterns

Base RDD

Transformed RDD

```
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
. . .
```

Action



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

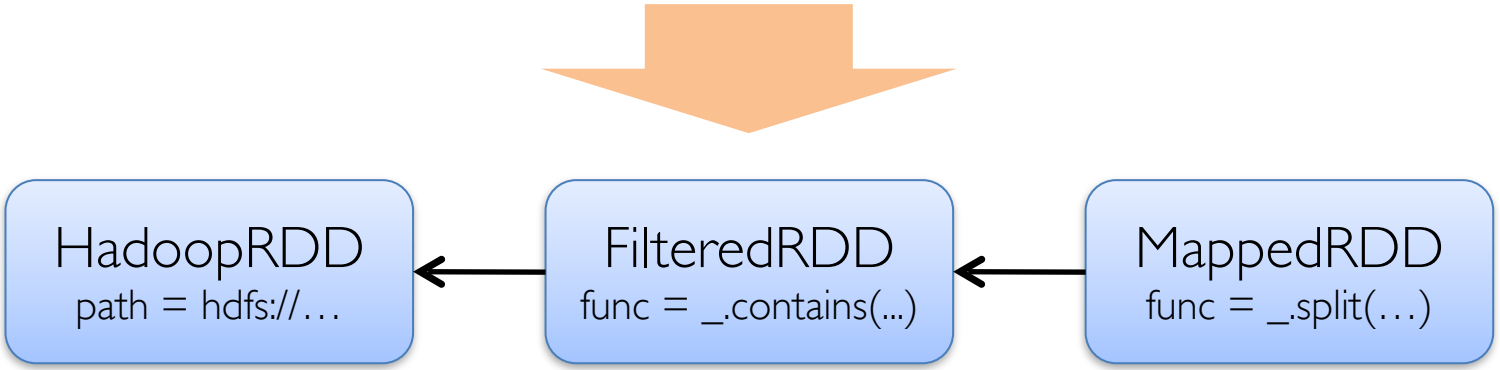
http://ananthakumaran.in/2010/03/29/scala-underscore-magic.html
http://www.slideshare.net/normation/scala-dreaded

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

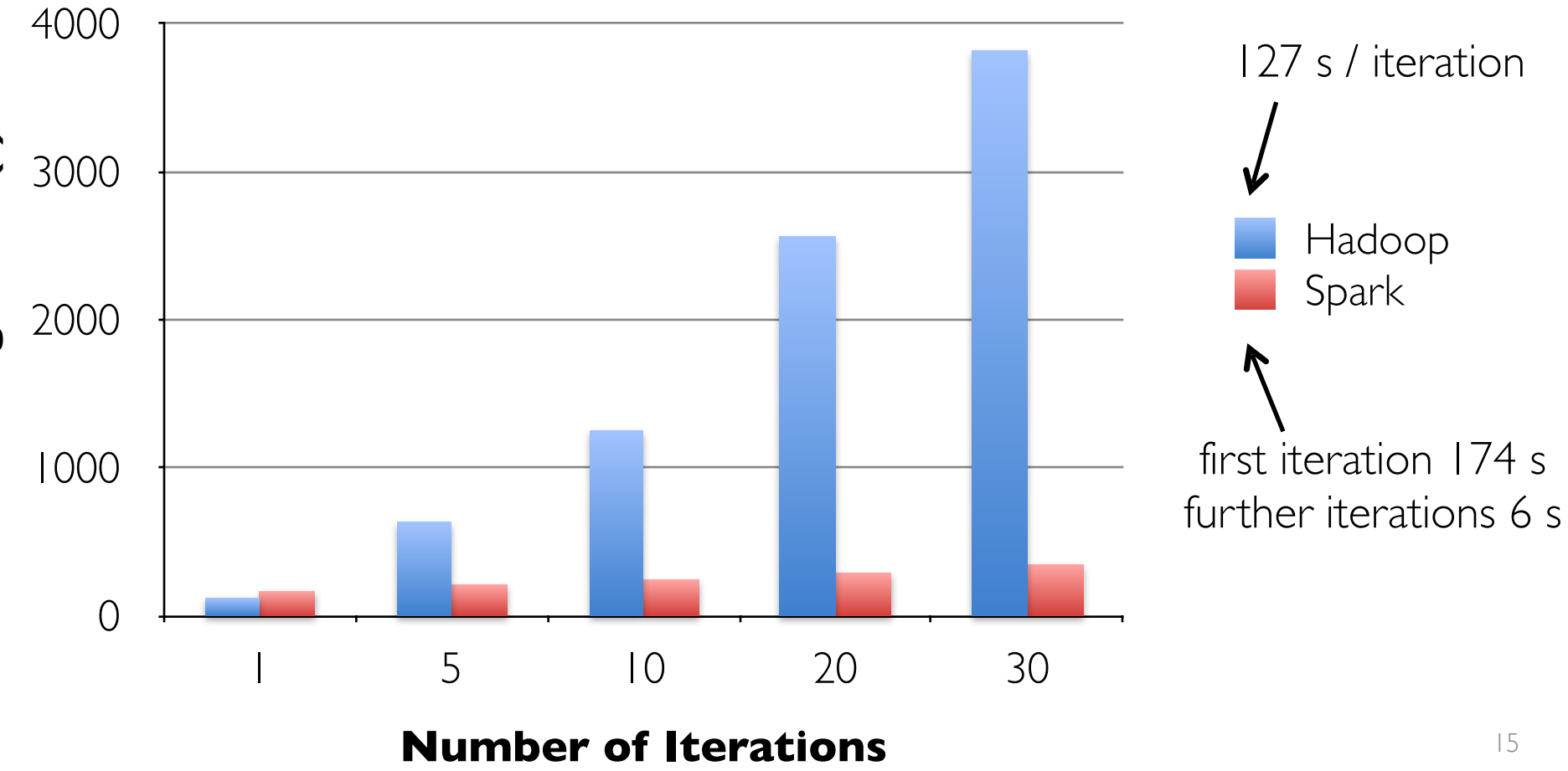
```
val data = spark.textFile(...).map(readPoint).cache()

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)
```

Load data in memory once

Initial parameter vector

Repeated MapReduce steps to do gradient descent

Logistic Regression Performance



Supported Operators

map

filter

groupBy

sort

join

leftOuterJoin

rightOuterJoin

reduce

count

reduceByKey

groupByKey

first

union

cross

sample

cogroup

take

partitionBy

pipe

save

...

Spark SQL: Hive on Spark

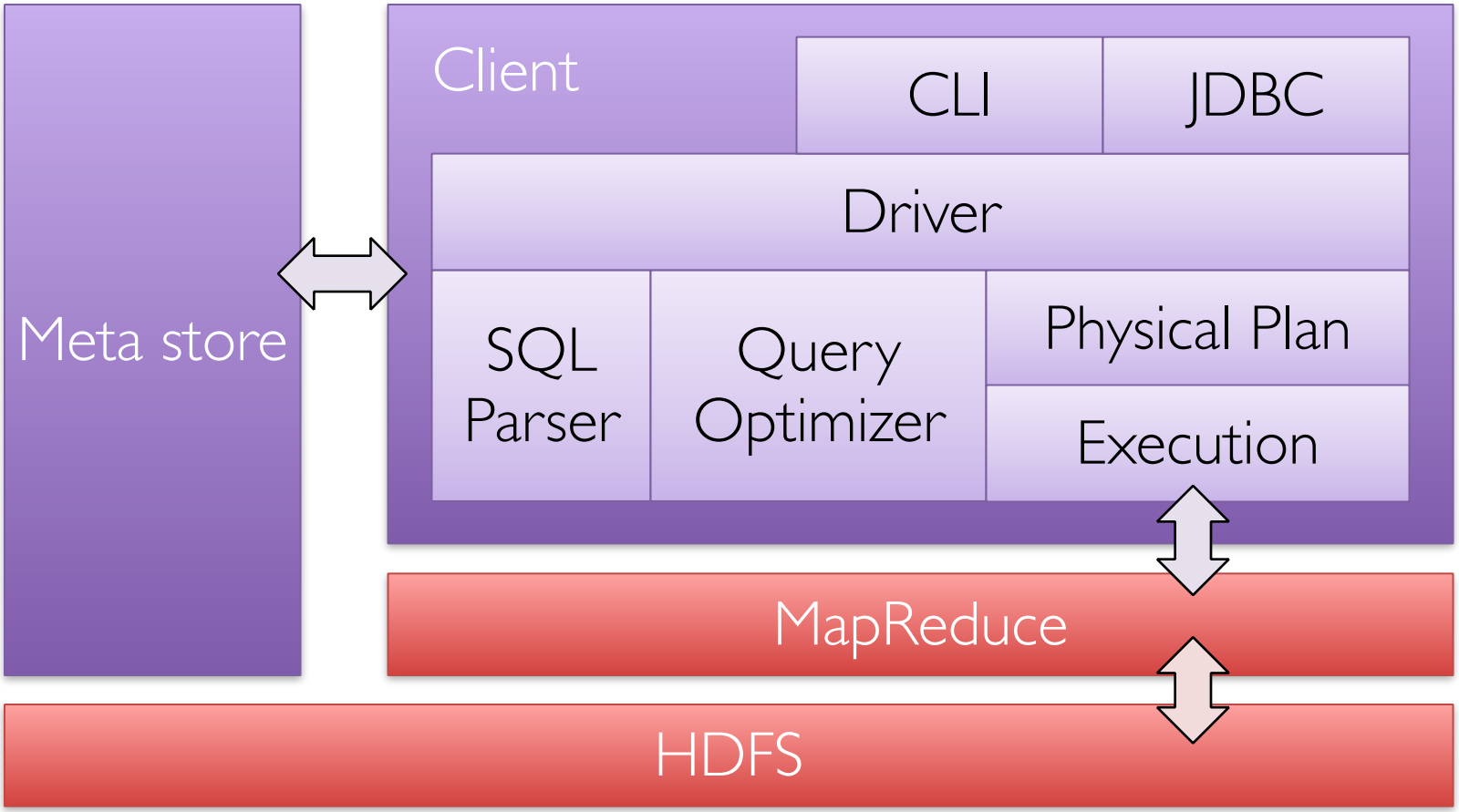
Motivation

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

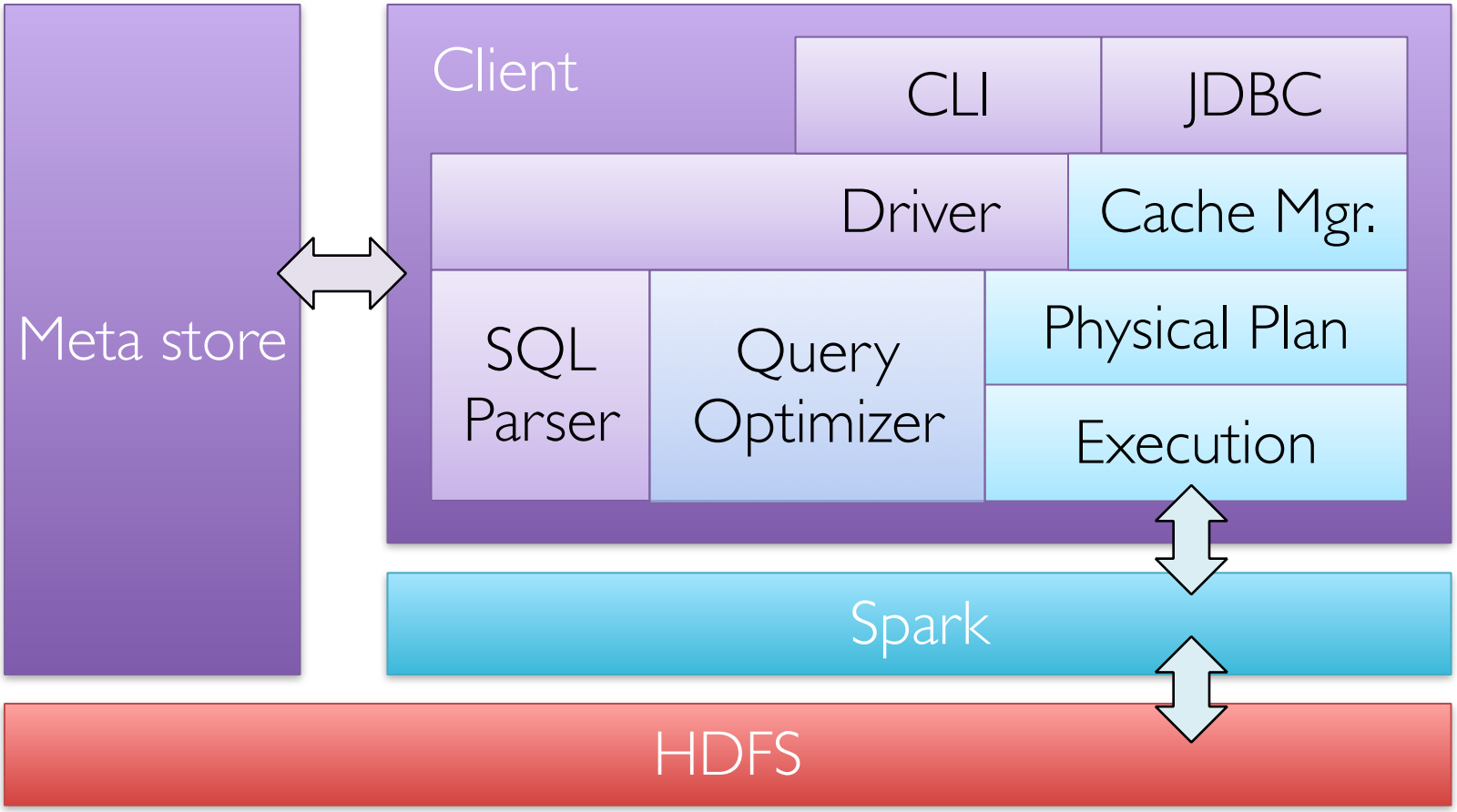
Scala is good for programmers, but many data users only know SQL

Can we extend Hive to run on Spark?

Hive Architecture



Spark SQL Architecture



[Engle et al, SIGMOD 2012]

Using Spark SQL

```
CREATE TABLE mydata_cached AS SELECT …
```

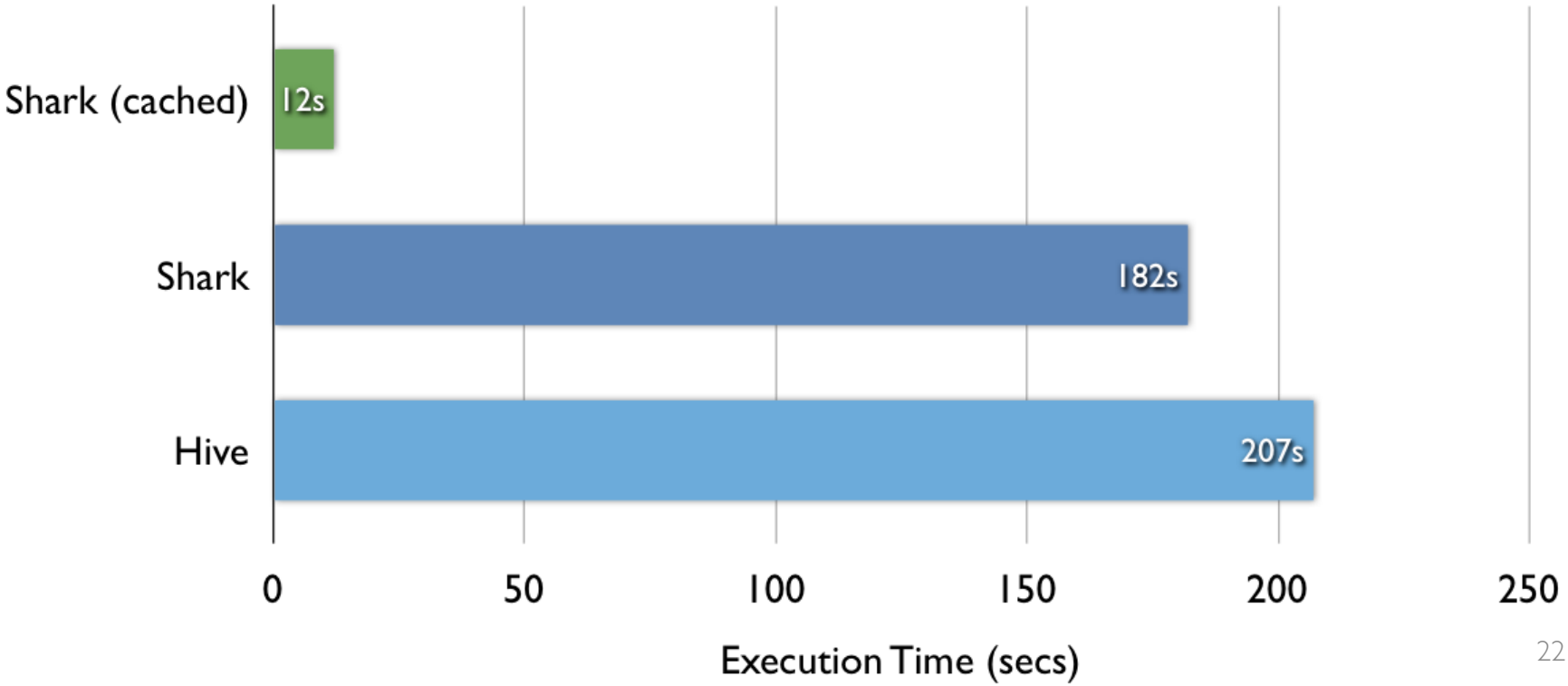
Run standard HiveQL on it, including UDFs

- A few esoteric features are not yet supported

Can also call from Scala to mix with Spark

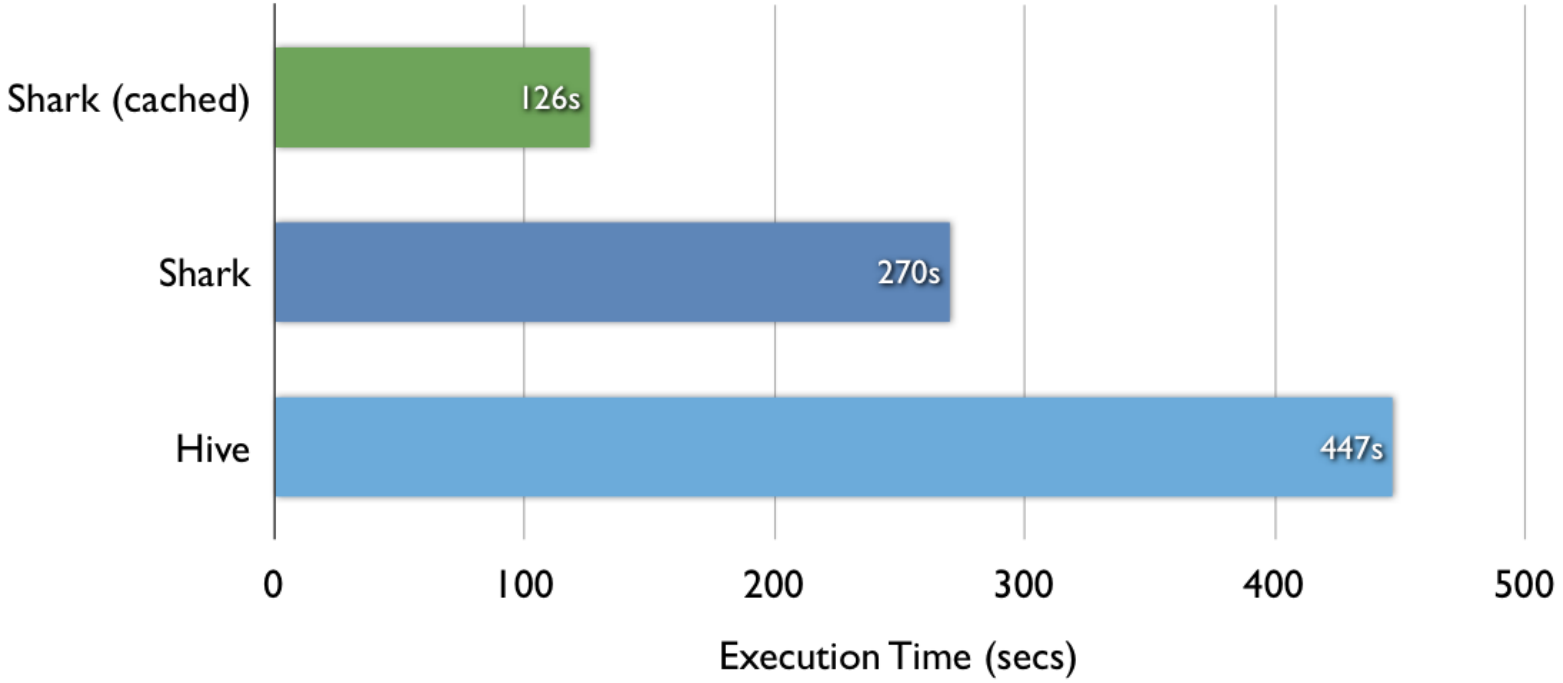
Benchmark Query 1

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

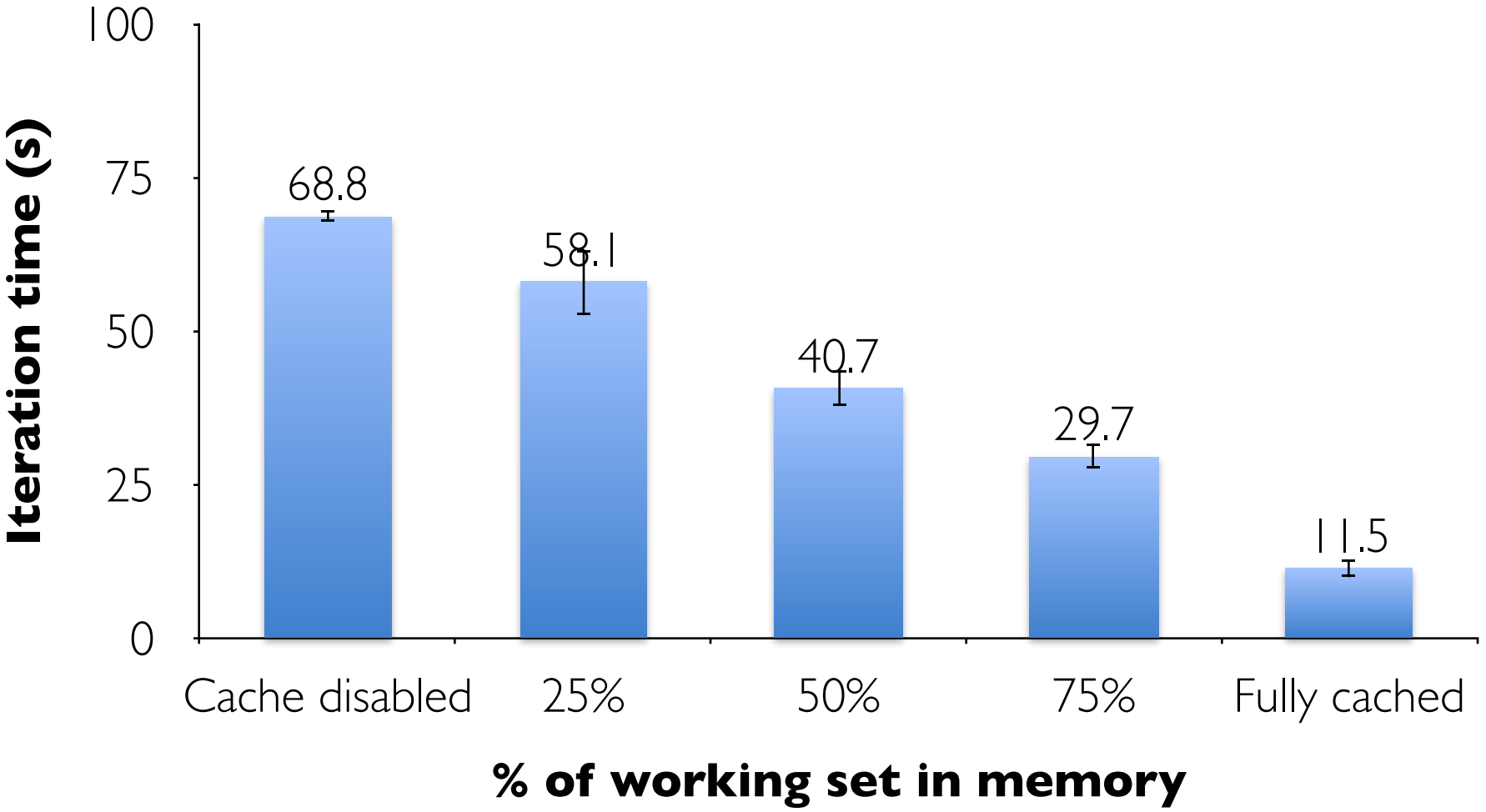


Benchmark Query 2

```
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

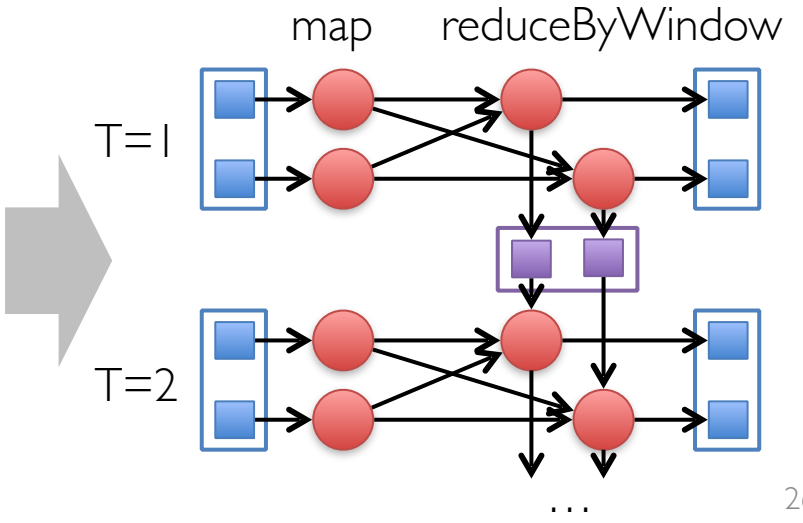
Streaming Spark

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



[Zaharia et al, HotCloud 2012]

map() vs flatMap()

The best explanation:

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

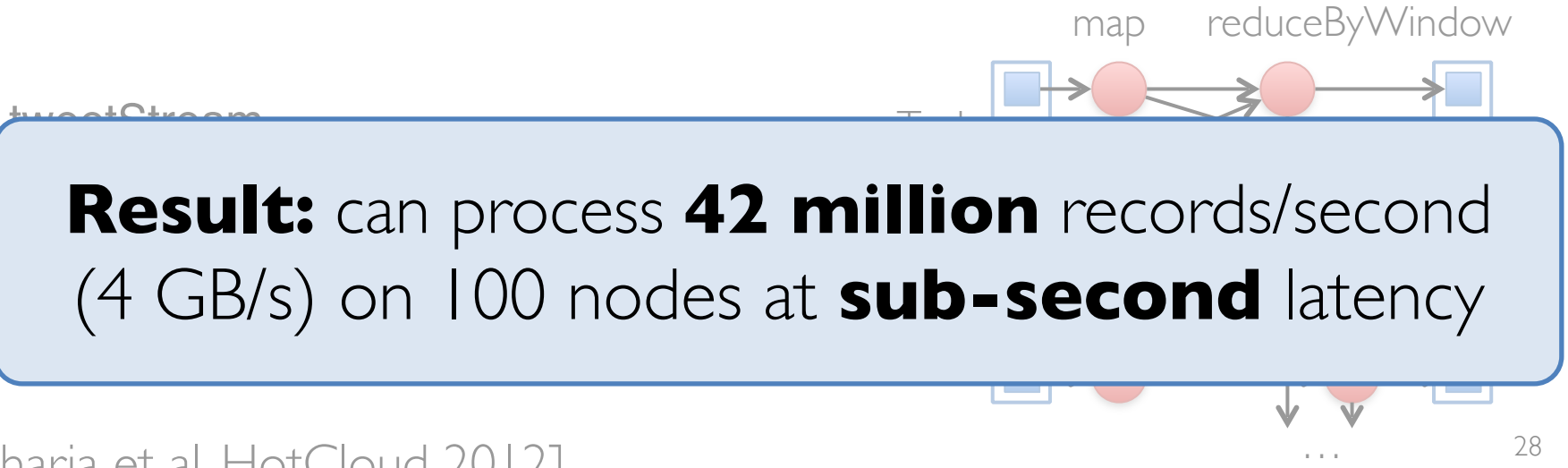
flatMap = map + flatten

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

[Zaharia et al, HotCloud 2012]

GraphX

Parallel graph processing

Extends RDD -> Resilient Distributed Property Graph

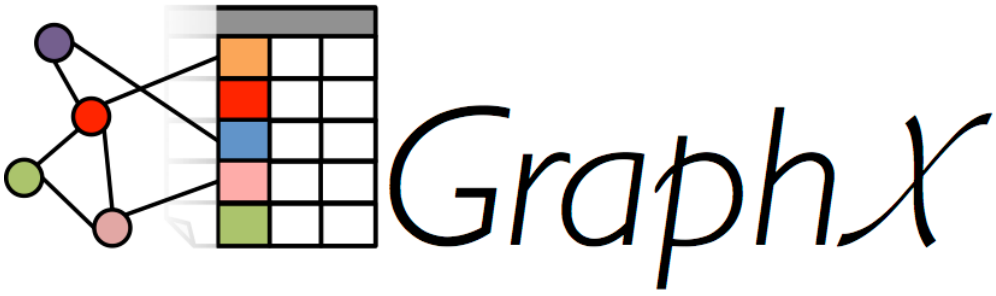
» Directed multigraph with properties attached to each vertex and edge

Limited algorithms

» PageRank

» Connected Components

» Triangle Counts



MLlib

- 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)

https://spark.apache.org/docs/latest/mllib-guide.html