COMSM0010 Cloud Computing

Lecture 15

Hadoop and related technologies

AKA "The Hadoop Ecosystem"

Dave Cliff

Department of Computer Science
University of Bristol

csdtc@bristol.ac.uk

University of BRISTOL

The schedule for the Big Data stuff…

- L13 Big Data I: Google's Core Tech (DC)
- L15 Big Data II: The Hadoop Ecosystem (DC)
- L16 Big Data III: Spark (DC)
- L17 Big Data IV: Cloud Databases (DC)
- L18: Big Data V: Graph Databases (DS)
- L19: Big Data VI: Stream Processing (DC)
- L20: Cloud Security (DC)

Summary of today's lecture, on one slide!

- **Pig:** a platform for batch-mode analysis of large data sets via a high-level programming language called Pig Latin, which is compiled down to sequences of MapReduce programs that can be executed by Hadoop.
- **Hive:** data-warehousing software that allows very large data sets in distributed storage to be read, written, and managed via SQL.-like language called HQL.
- **YARN:** (stands for Yet Another Resource Negotiator) performs cluster resource management and job-scheduling acting as the middleware layer between HDFS (and HDFS2&3) and the various other applications listed here.
- **Mahout**: Apache's scalable machine learning platform, runs on Hadoop/Spark: its provides tools for using Hadoop to create systems that perform clustering, classification, and collaborative filtering at high speed on large data sets.
- **Hoya/HBase:** Hoya is a tool for running HBase clusters on YARN. HBase is Hadoop's non-relational (NoSQL) distributed database, directly comparable to Google's *BigTable*. **More in L17.**
- **Storm:** a distributed real-time stream-processing system, originally by Twitter (now an Apache project). Storm allows for creation of event-driven processing networks (directed acyclic graphs) which operate continuously in real-time, in comparison to MapReduce's batch-mode style of working, and can run on YARN. **More in L19.**
- **Giraph:** is a graph-database, running MapReduce jobs to process graphs. It's been used extensively by Facebook to process their social network data. Giraph is comparable to Google's Pregel graph processing system. **More in L18.**
- **Spark:** high-performance analytics engine/framework for large-scale data-processing that runs on YARN (and other platforms including AWS EC2 and Kubernetes) It uses in-memory techniques to achieve significant speed-ups via implicit parallelism and execution-plan optimisation, and is highly fault-tolerant: this makes it much faster than Mapreduce. **More in L16.**

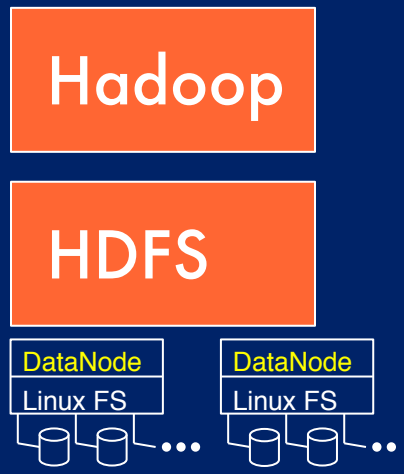
How it came to be…

- In the beginning there was Hadoop (Apache MapReduce)

Hadoop

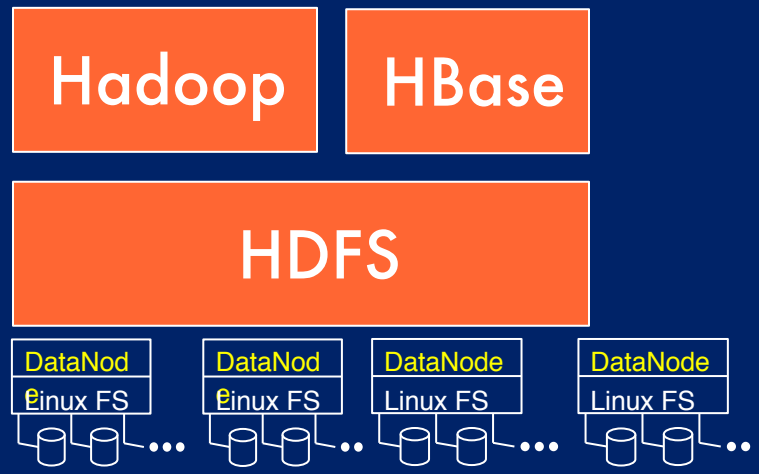
How it came to be…

- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13



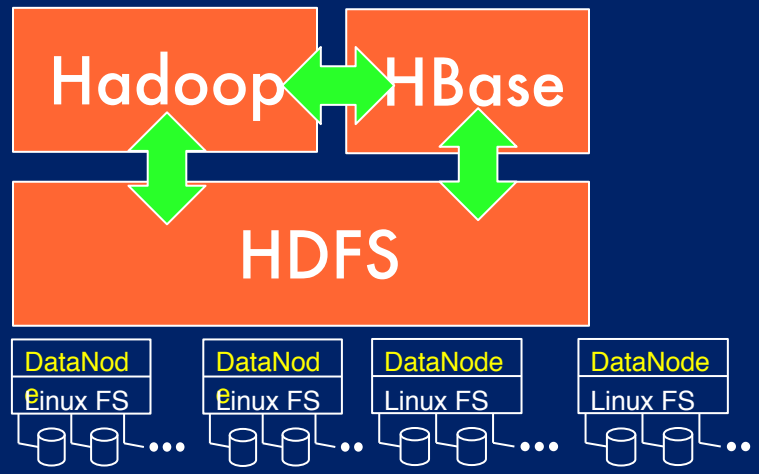
How it came to be…

- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase



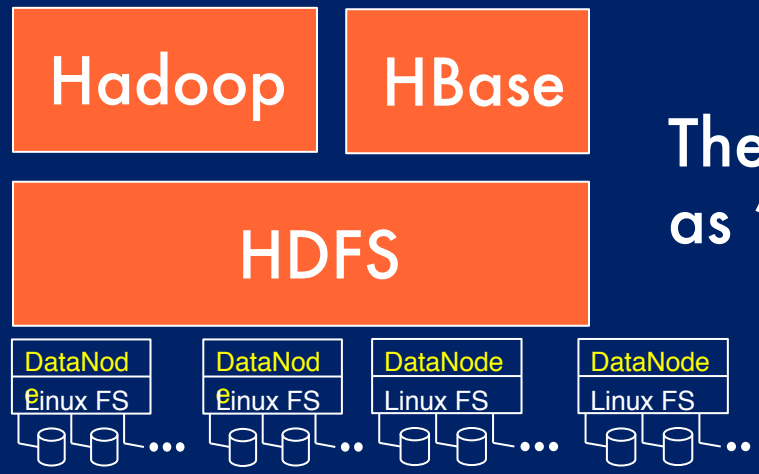
How it came to be…

- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase



How it came to be…

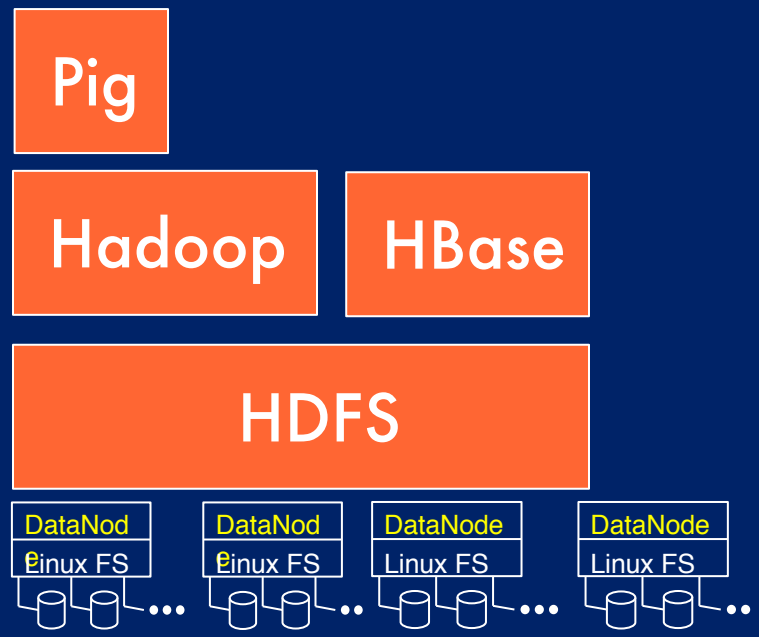
- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase



These three used to be known as "The Hadoop Core"

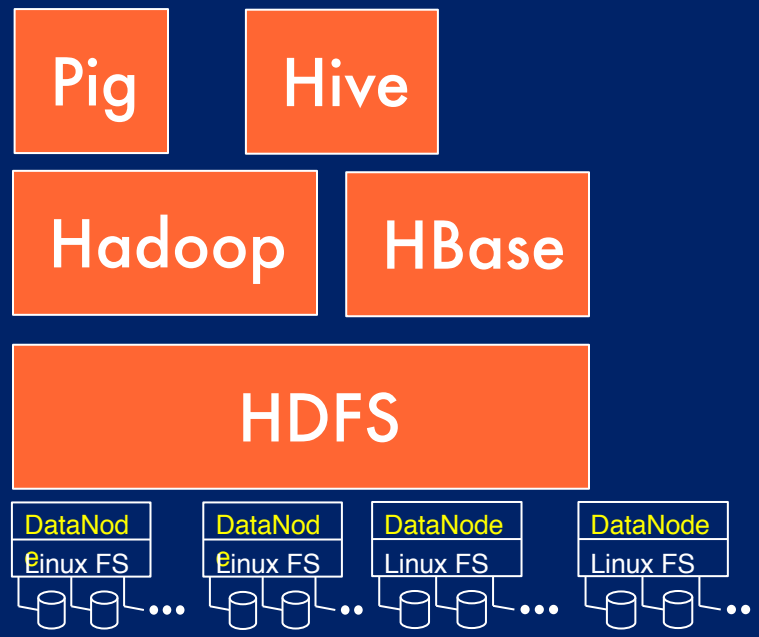
How it came to be…

- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase
- Writing MR jobs is tricky (1): programmatic abstraction = Apache Pig



How it came to be…

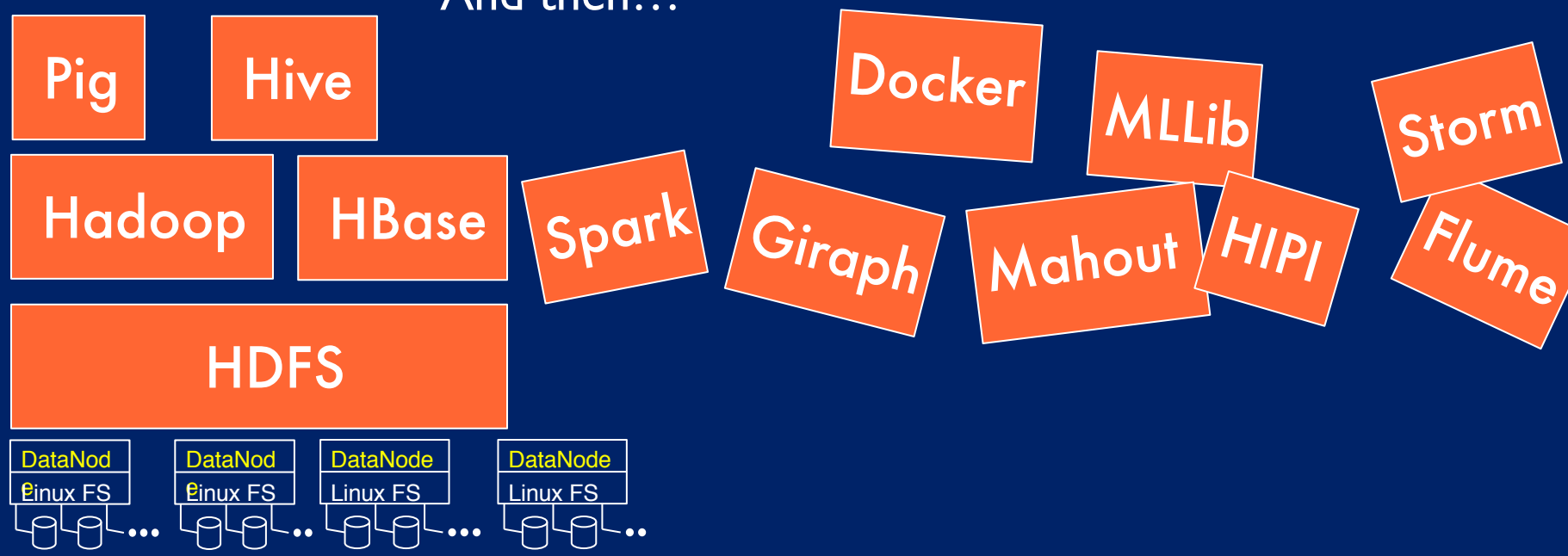
- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase
- Writing MR jobs is tricky (I): programmatic abstraction = Apache Pig
- Writing MR jobs is tricky (II): SQL-like query language = Apache Hive



How it came to be…

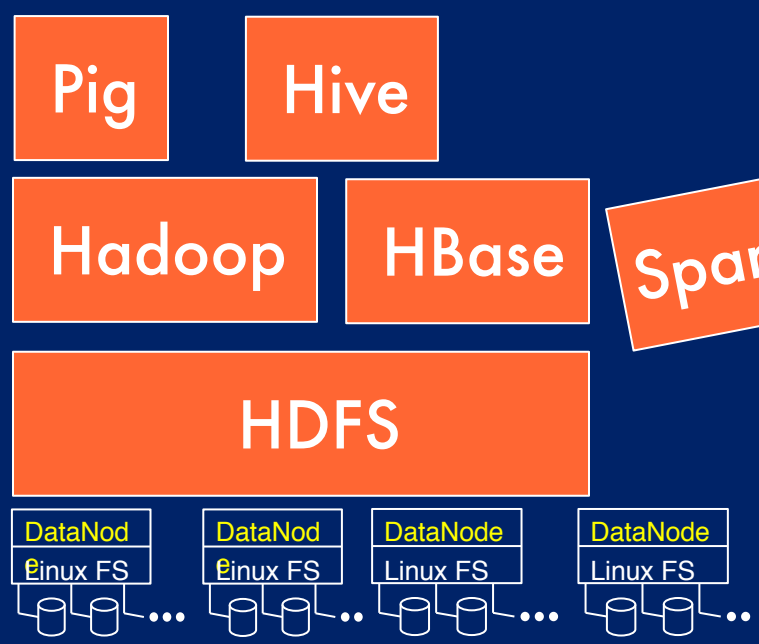
- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase
- Writing MR jobs is tricky (I): programmatic abstraction = Apache Pig
- Writing MR jobs is tricky (II): SQL-like query language = Apache Hive

And then…



How it came to be…

- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase
- Writing MR jobs is tricky (I): programmatic abstraction = Apache Pig
- Writing MR jobs is tricky (II): SQL-like query language = Apache Hive

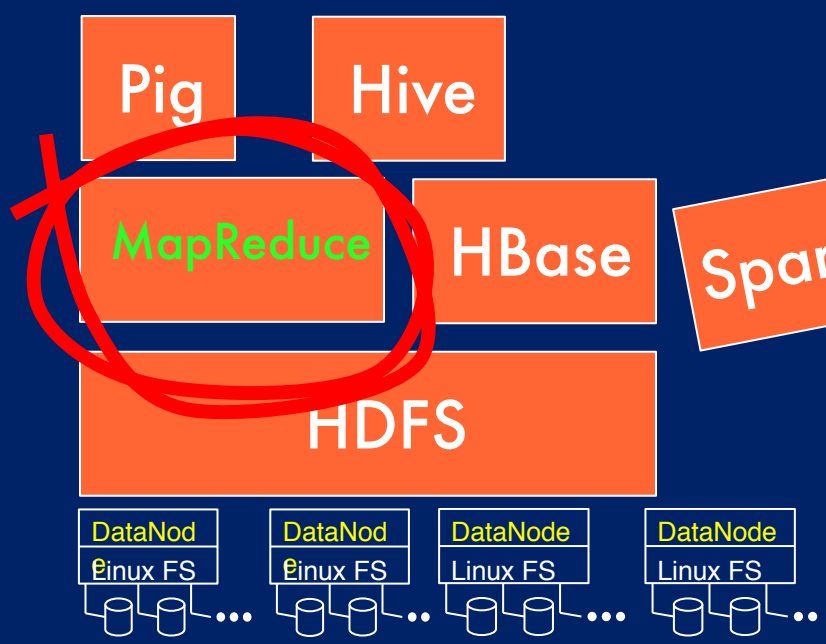


TIME TO REFACTOR!!!!

"Hadoop" is now a whole "ecosystem"

How it came to be…

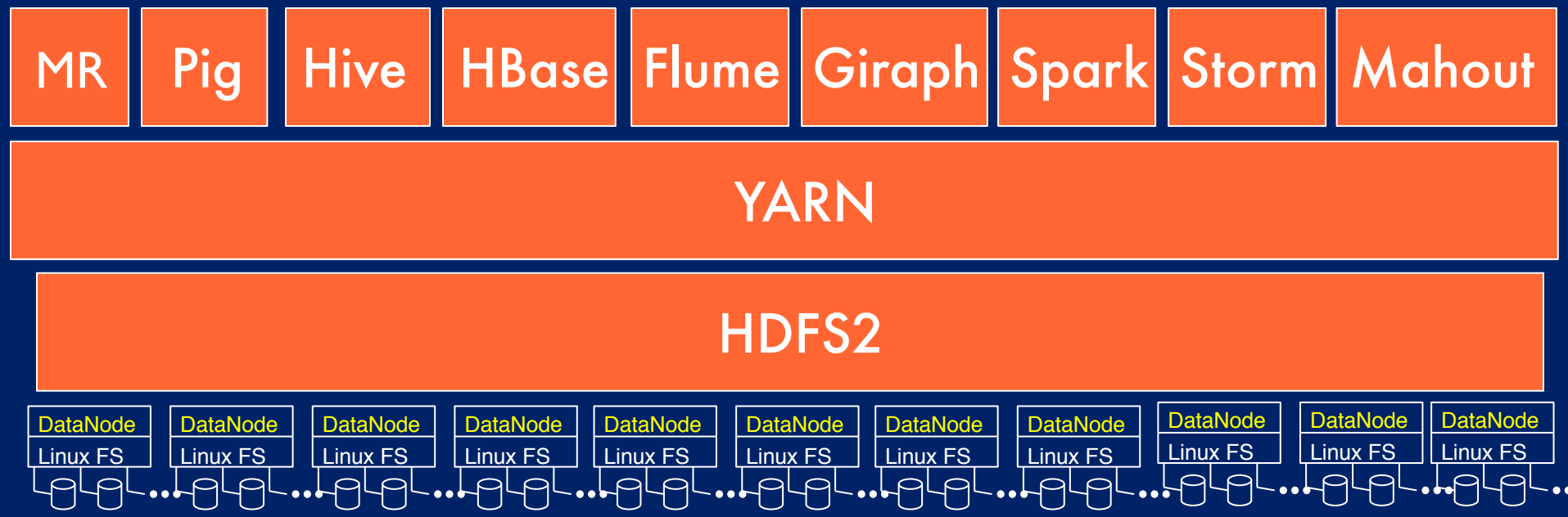
- In the beginning there was Hadoop (Apache MapReduce) – L13
- Hadoop needed something like GFS = Apache HDFS – L13
- Databases on HDFS need something like Google BigTable = Apache HBase
- Writing MR jobs is tricky (I): programmatic abstraction = Apache Pig
- Writing MR jobs is tricky (II): SQL-like query language = Apache Hive



TIME TO REFACTOR!!!!

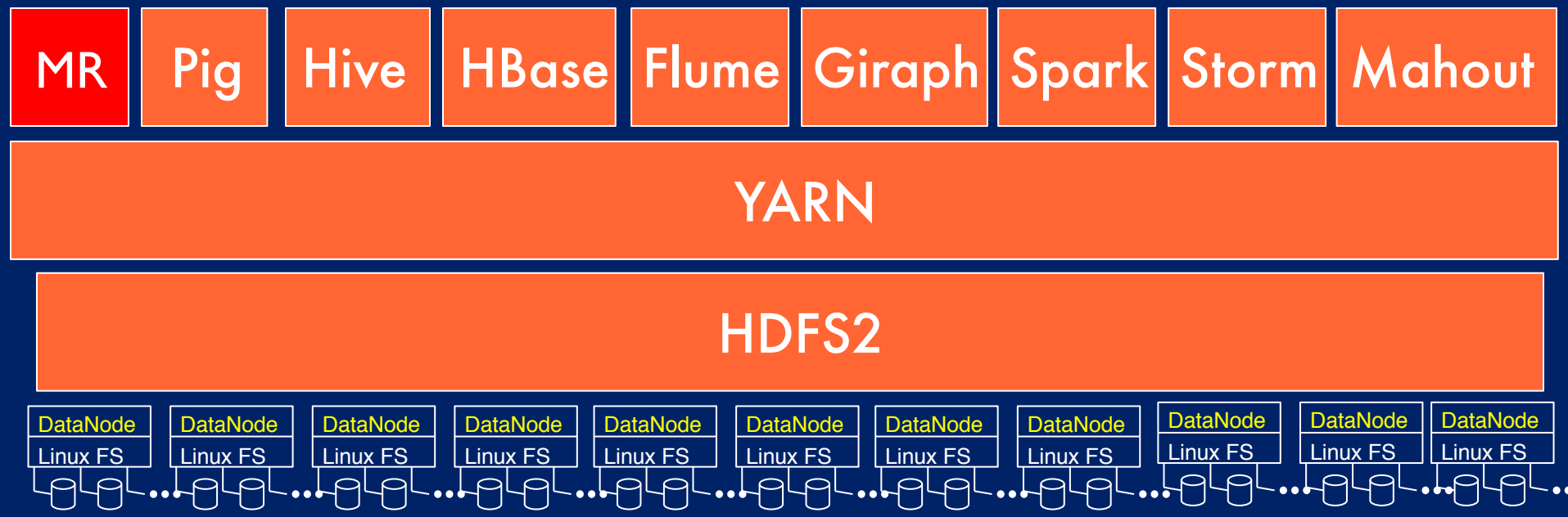
So the "Hadoop" component is renamed as "MapReduce" (MR) in Hadoop 2.0

Hadoop2.0


MR
Pig
Hive
HBase
Flume
Giraph
Spark
Storm
Mahout
YARN
HDFS2
DataNode
Linux FS

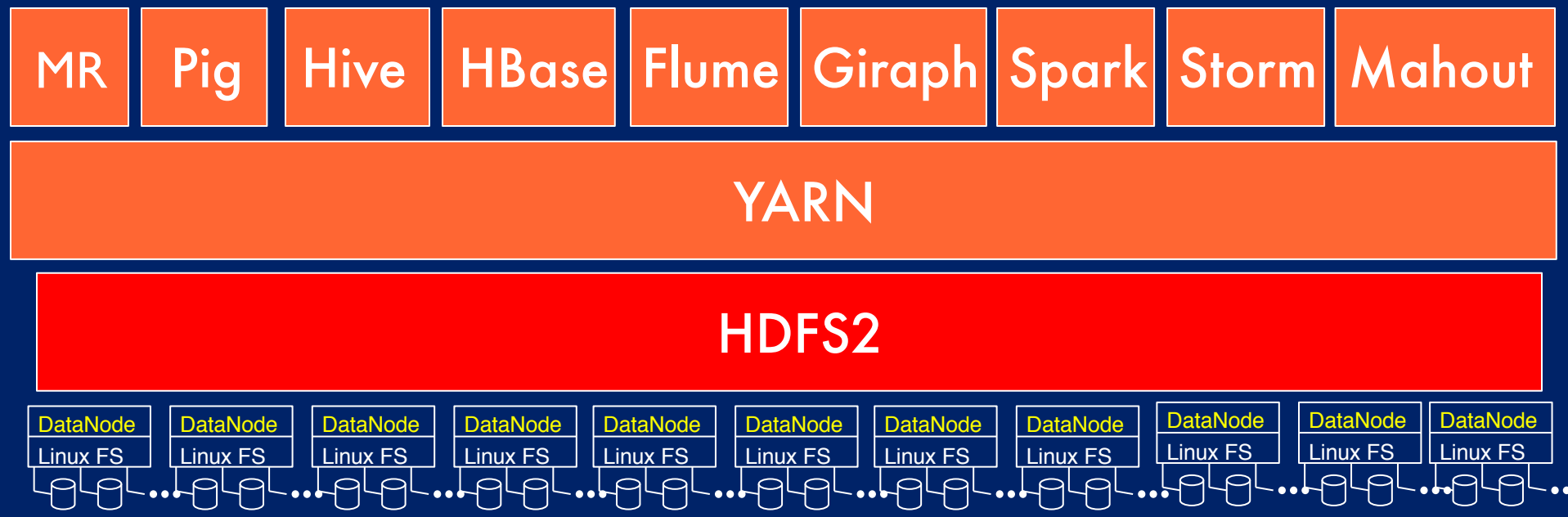
Hadoop2.0

- MapReduce is still there – we covered that in L13



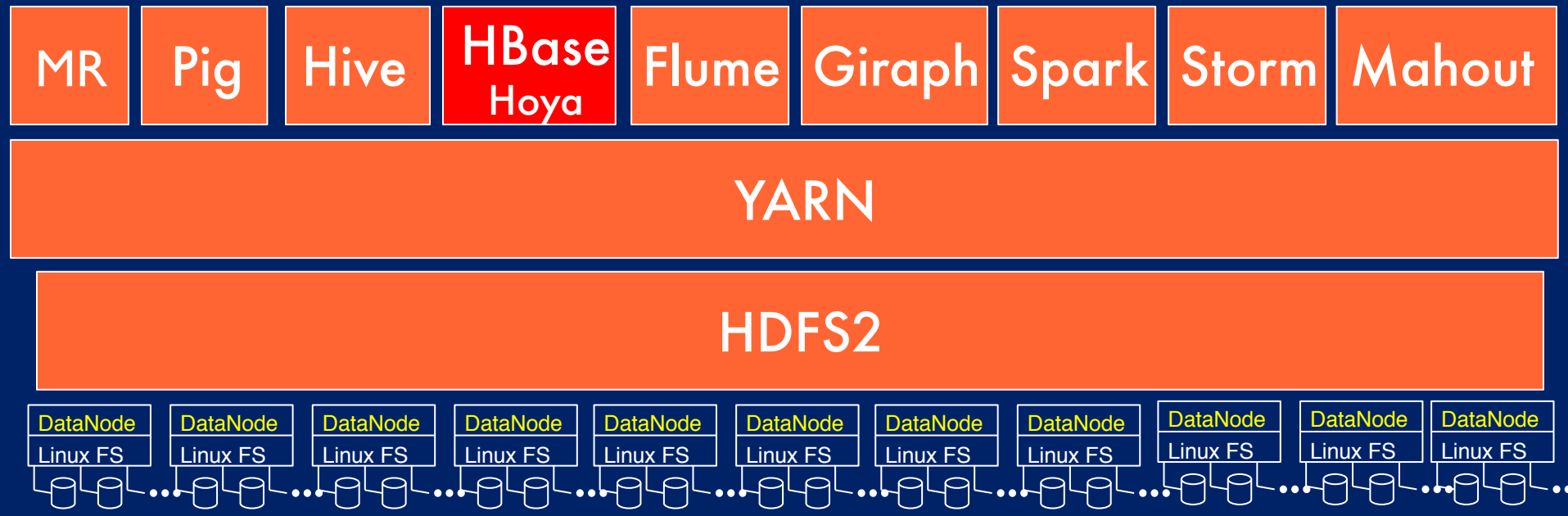
Hadoop2.0

- HDFS still there (as HDFS2, now moving to HDFS3 with *Erasure Coding*)



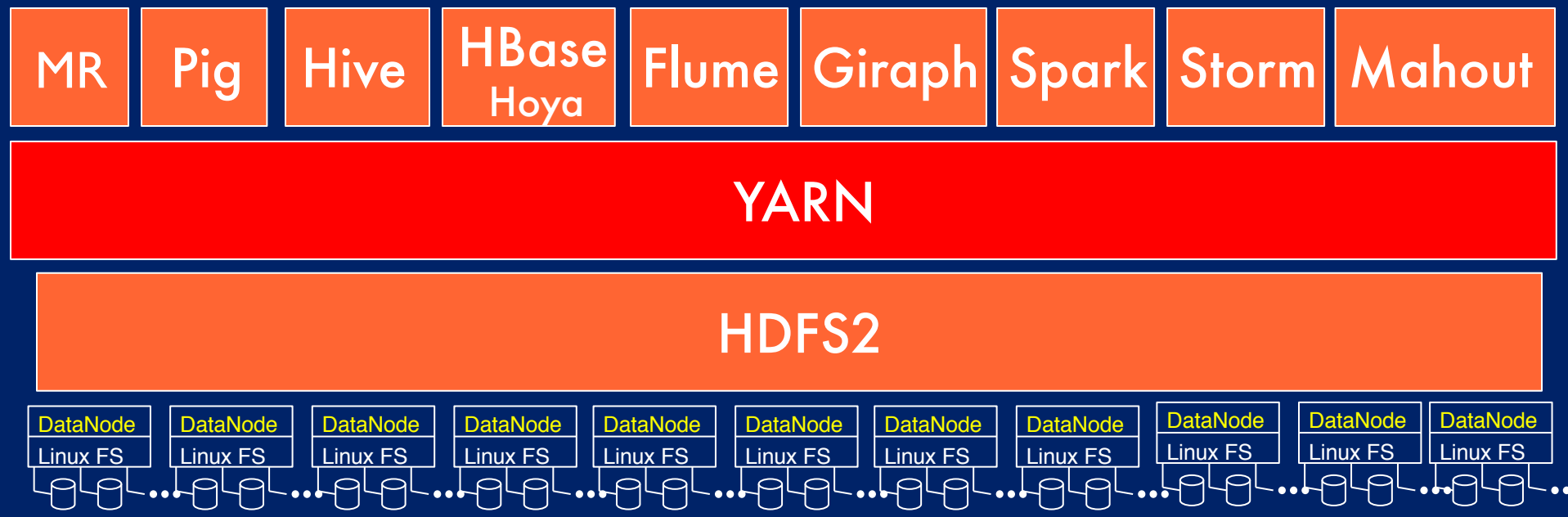
Hadoop2.0

- HBase still there
- Sometimes referred to as Hoya: HBase extended for Hadoop2.0/YARN



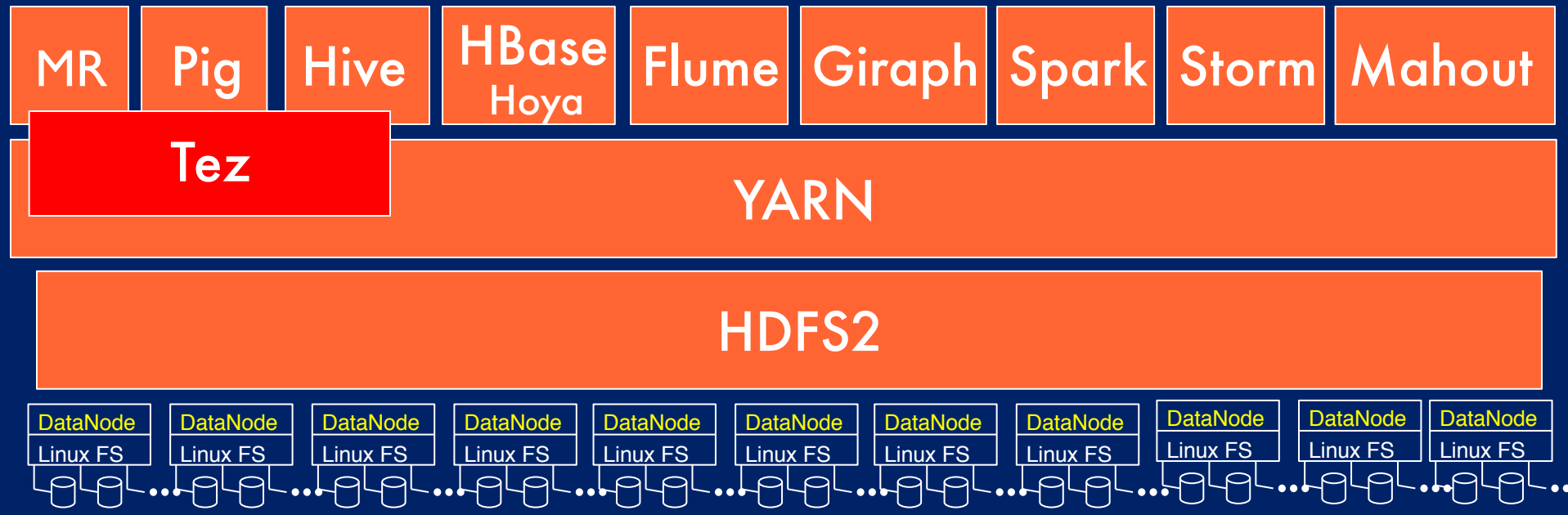
Hadoop2.0

- YARN (*Yet Another Resource Negotiator*) does cluster resource management – middleware through which other Hadoop components talk to HDFS



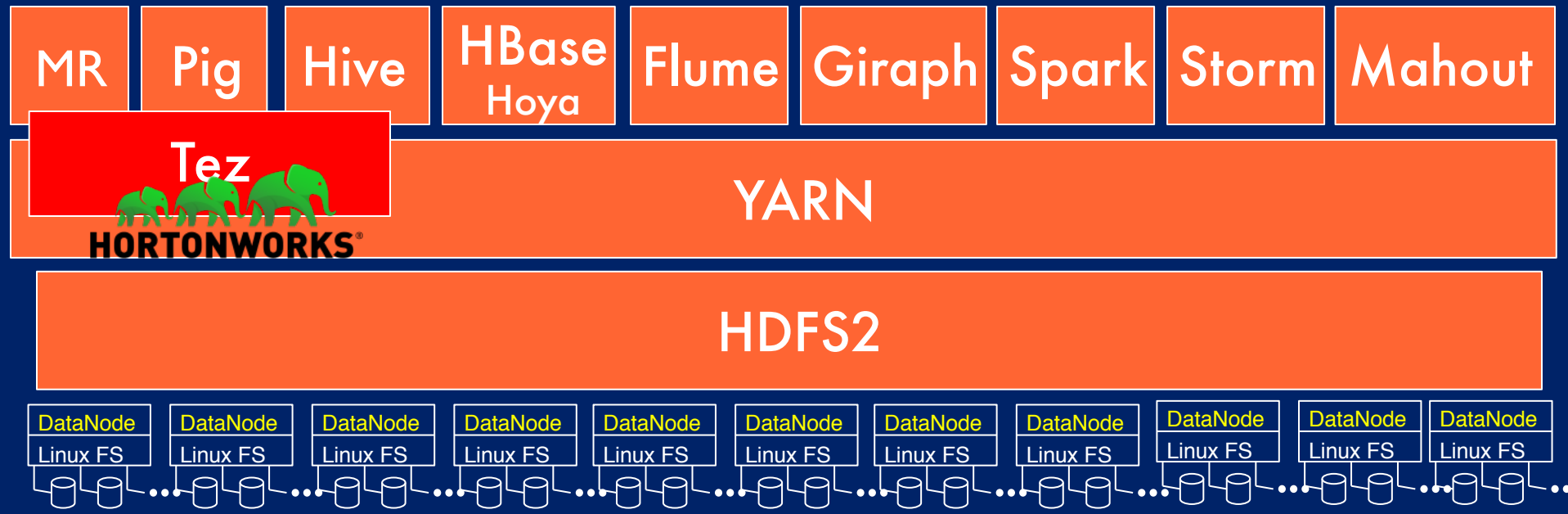
Hadoop2.0

- YARN (*Yet Another Resource Negotiator*) does cluster resource management – middleware through which other Hadoop components talk to HDFS
- NB in some documents you might see mention of Tez execution engine, sitting between MapReduce/Pig/Hive and YARN…
  - … strongly associated with Hortonworks (vs Cloudera rooting for Spark)
  - … seems to have largely lost out to Spark, so we won't explore it here



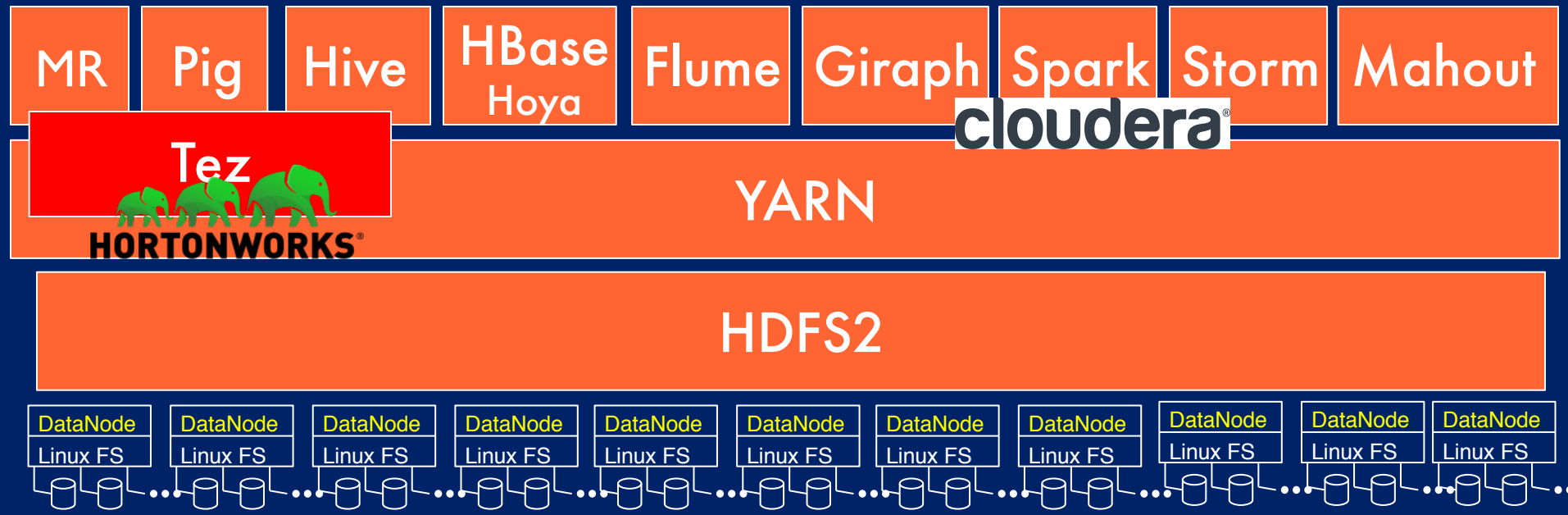
Hadoop2.0

- YARN (*Yet Another Resource Negotiator*) does cluster resource management – middleware through which other Hadoop components talk to HDFS
- NB in some documents you might see mention of Tez execution engine, sitting between MapReduce/Pig/Hive and YARN…
  - … strongly associated with Hortonworks (vs Cloudera rooting for Spark)
  - … seems to have largely lost out to Spark, so we won't explore it here



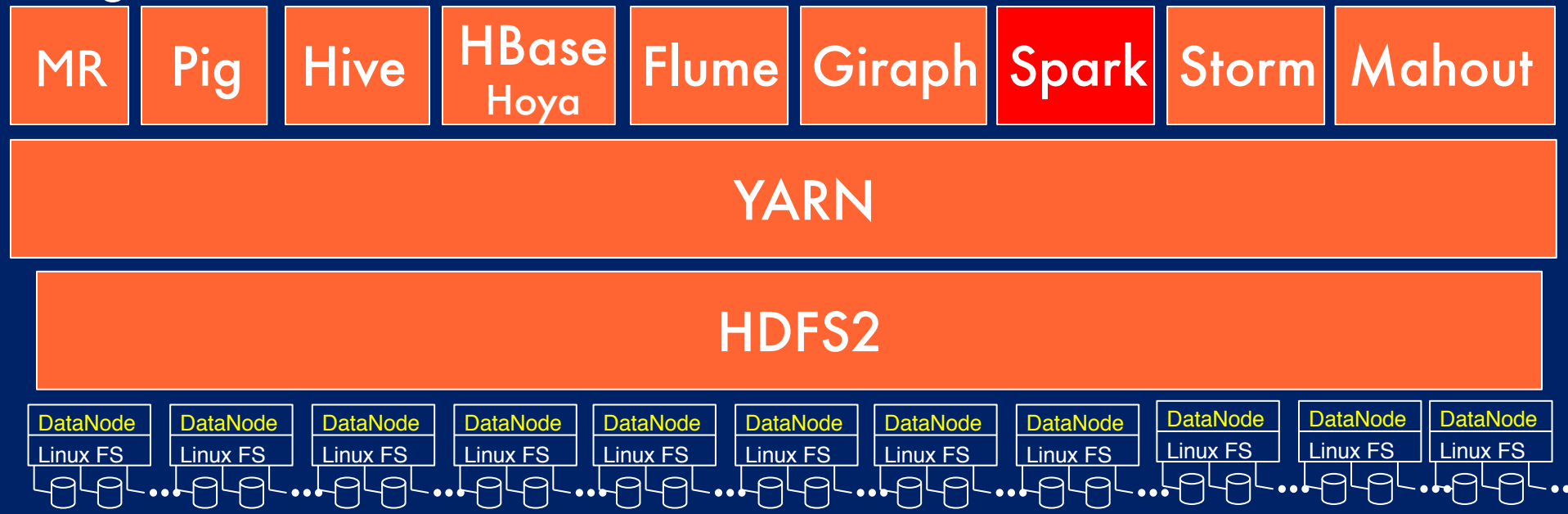
Hadoop2.0

- YARN (*Yet Another Resource Negotiator*) does cluster resource management – middleware through which other Hadoop components talk to HDFS
- NB in some documents you might see mention of Tez execution engine, sitting between MapReduce/Pig/Hive and YARN…
  - … strongly associated with Hortonworks (vs Cloudera rooting for Spark)
  - … seems to have largely lost out to Spark, so we won't explore it here



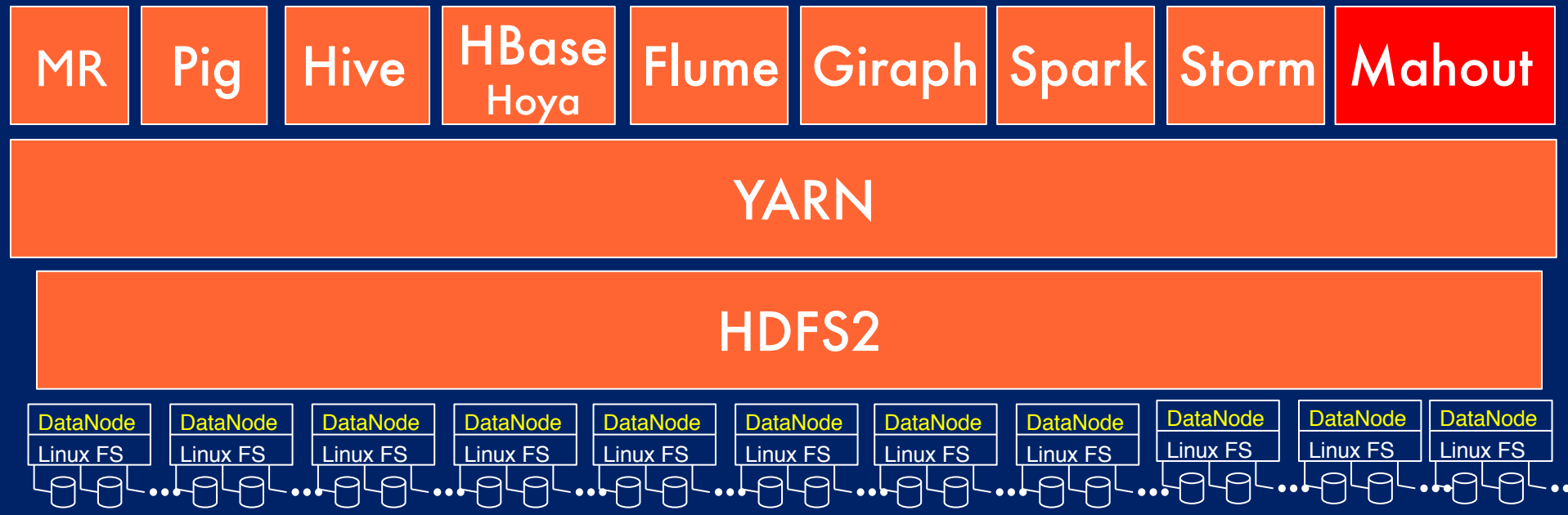
Hadoop2.0

- Spark (previously known as *Shark*) from Berkeley AMP Lab
- Runs *in-memory* (cf MR: peta-scale batch-mode processing of disk data)
- c.100 times faster than Hadoop-MR; runs on other platforms too.
- We cover Spark in detail in another lecture
- Some Hadoop2 projects now switching wholesale to Spark, dropping MR
- E.g. : Mahout



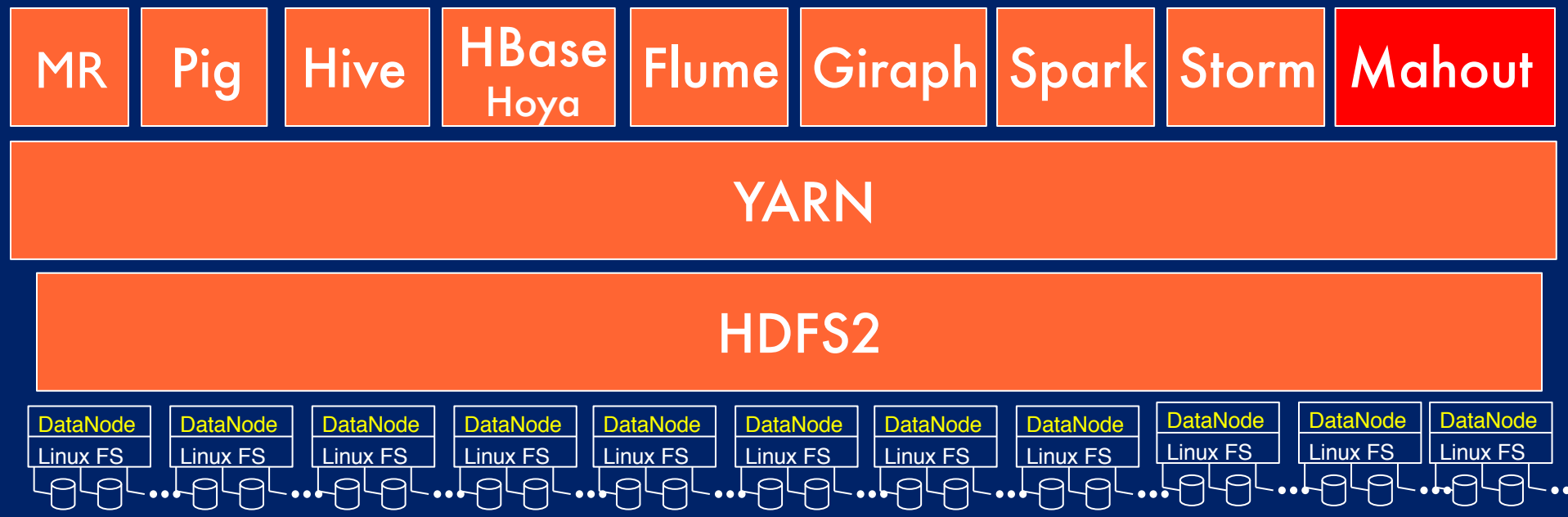
Hadoop2.0

- Mahout: Apache's scalable machine learning platform, runs on Hadoop/Spark: its provides tools for using Hadoop to create systems that perform clustering, classification, and collaborative filtering at high speed on large data sets.
- Various books available but many are out of date



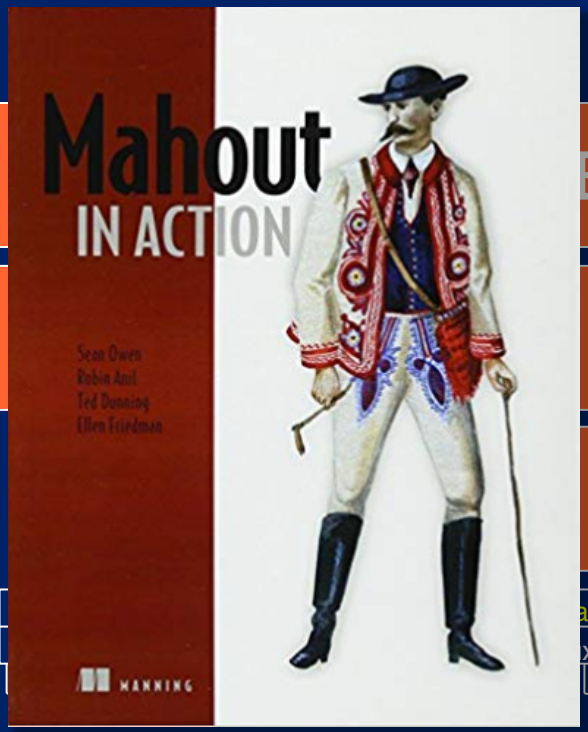
Hadoop2.0

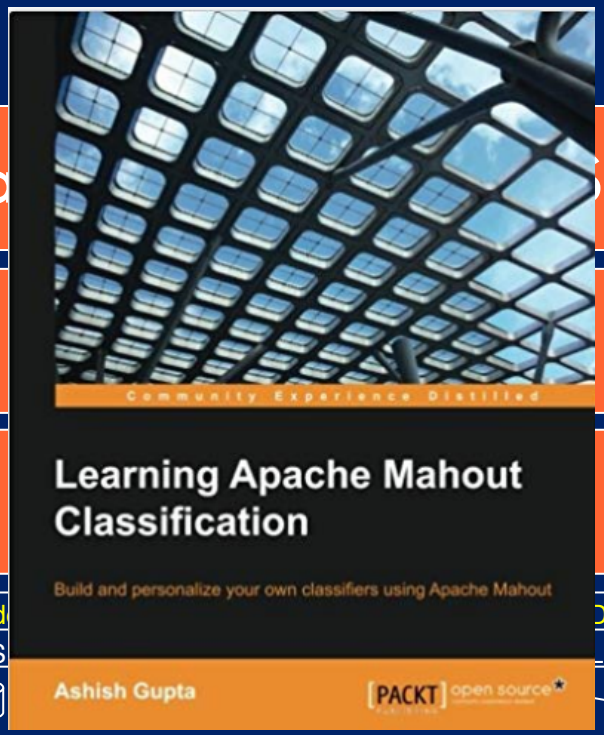
- Mahout: Apache's scalable machine learning platform, runs on Hadoop/Spark: its provides tools for using Hadoop to create systems that perform clustering, classification, and collaborative filtering at high speed on large data sets.
- Various books available but many are out of date (e.g. these…)

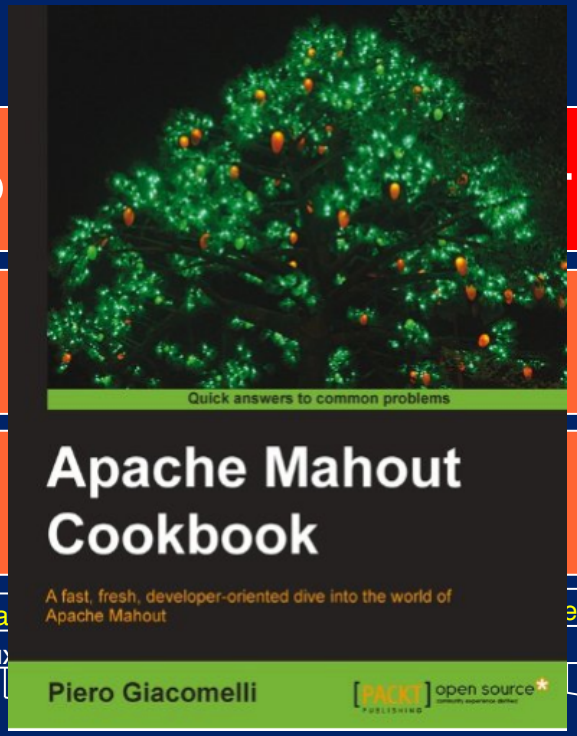


Hadoop2.0

- Mahout: Apache's scalable machine learning platform, runs on Hadoop/Spark: its provides tools for using Hadoop to create systems that perform clustering, classification, and collaborative filtering at high speed on large data sets.
- Various books available but many are out of date (e.g. these…)





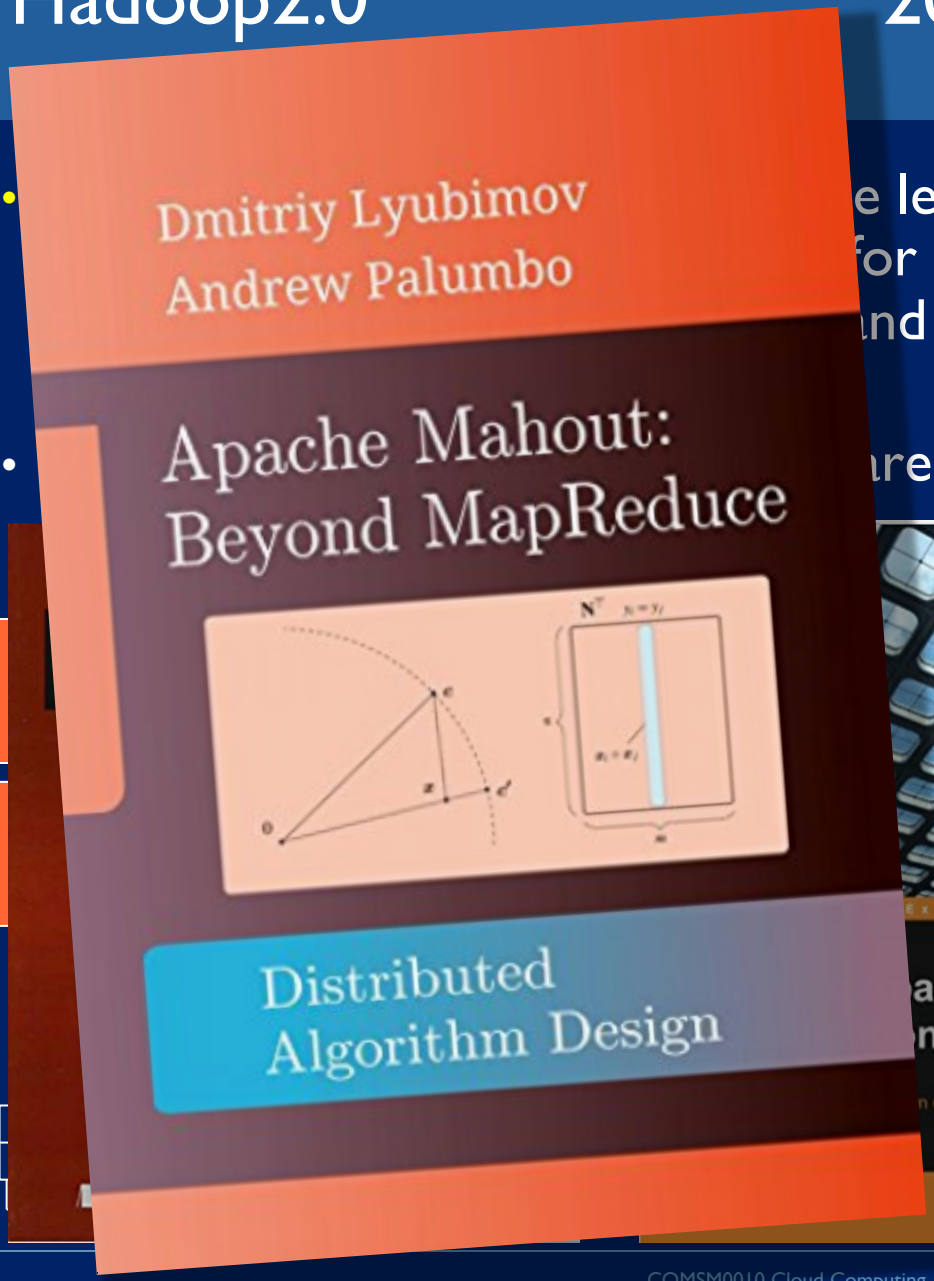


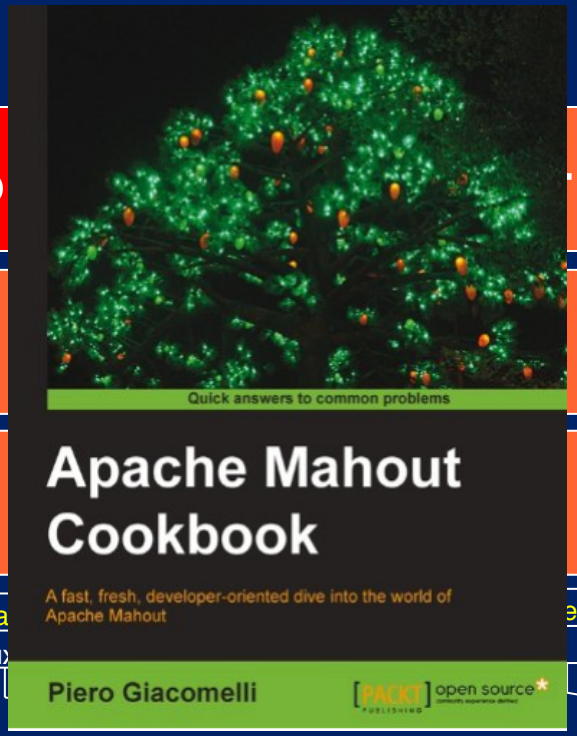
Hadoop2.0

2016 book: post-MapReduce

- e learning platform, runs on
  for using Hadoop to create systems that
  nd collaborative filtering at high speed

- re out of date

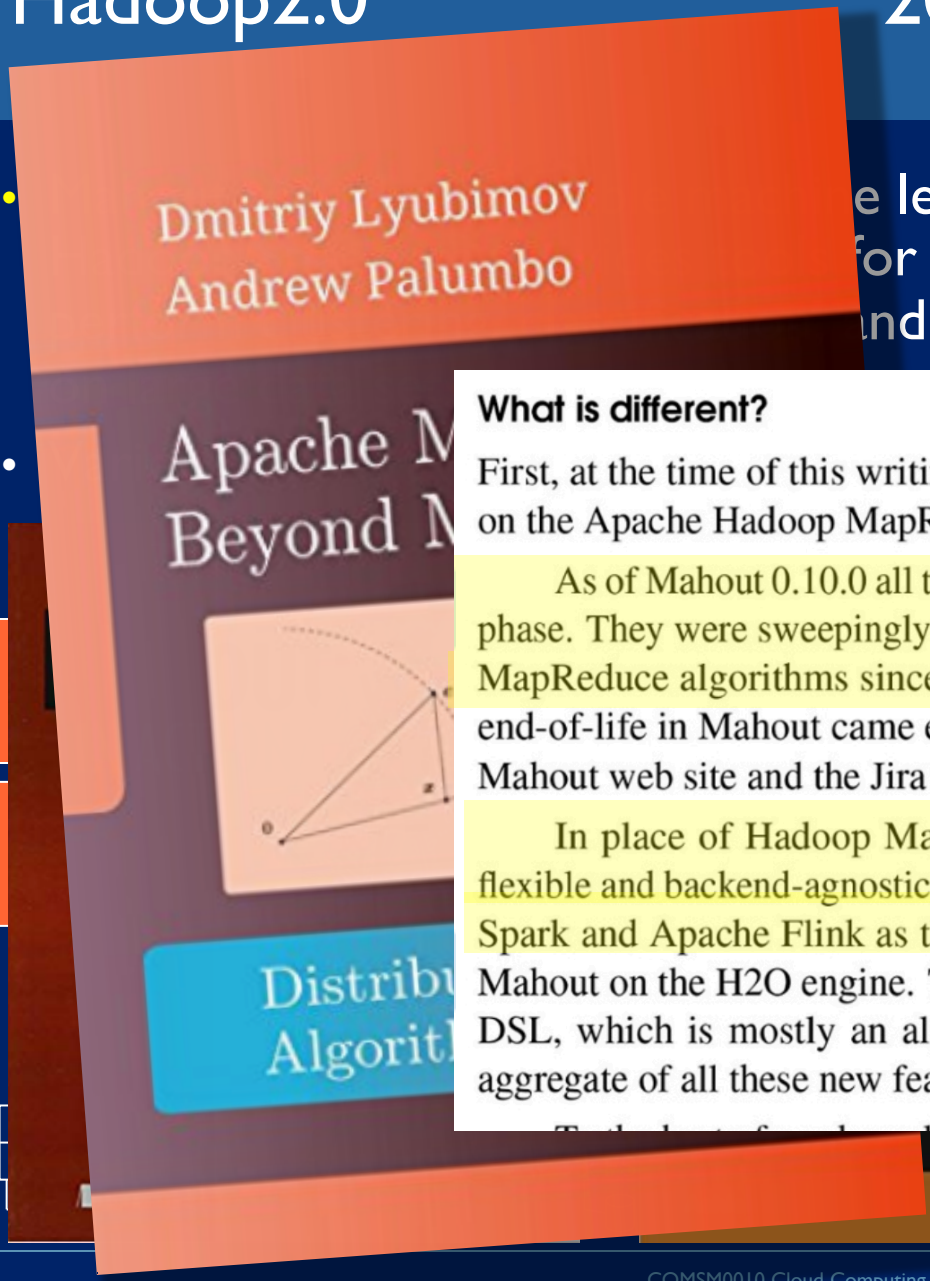




Hadoop2.0

2016 book: post-MapReduce

- e learning platform, runs on
  for using Hadoop to create systems that
  nd collaborative filtering at high speed
-



What is different?

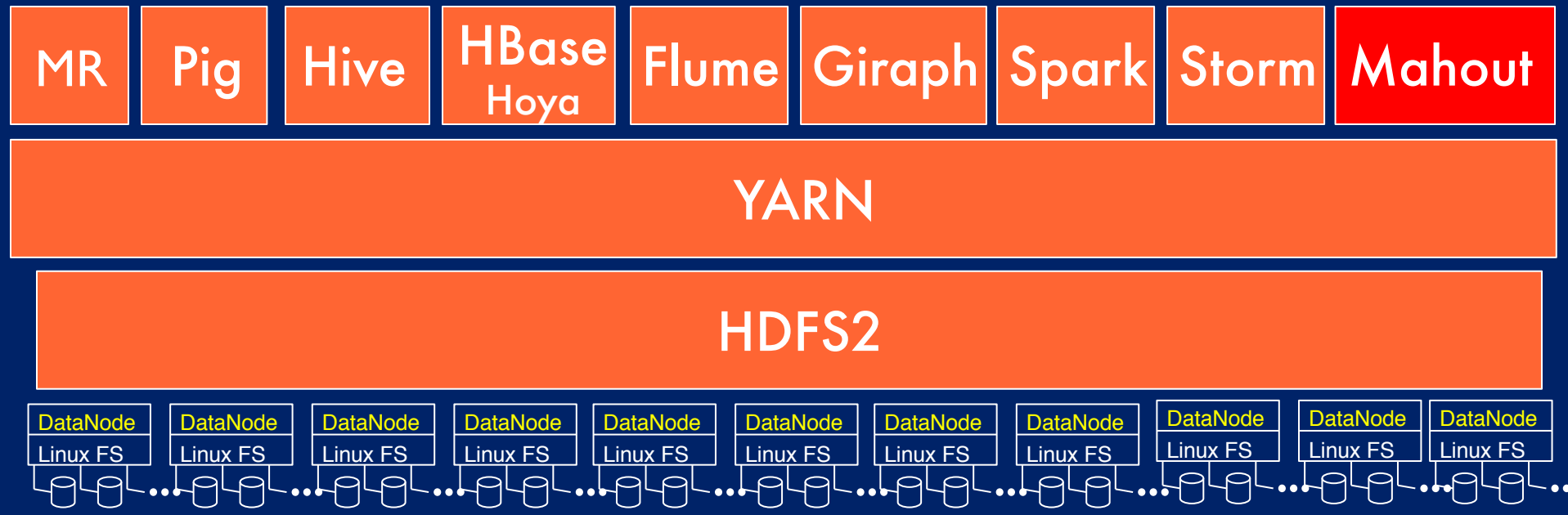
First, at the time of this writing, much of the prior printed matter on Mahout is focused on the Apache Hadoop MapReduce-based capabilities of Mahout.

As of Mahout 0.10.0 all the MapReduce algorithms officially entered their end-of-life phase. They were sweepingly deprecated, and the project has not been accepting any new MapReduce algorithms since. Strictly speaking, the announcement of the MapReduce end-of-life in Mahout came even earlier, in the first half of 2014, as a news post on the Mahout web site and the Jira issue MAHOUT-1510[1].

In place of Hadoop MapReduce, Mahout has been focusing on implementing a flexible and backend-agnostic machine learning environment. Mahout is targeting Apache Spark and Apache Flink as the main execution backends, and there is also support for Mahout on the H2O engine. The new programming API is completely based on a Scala DSL, which is mostly an algebraic DSL. Mahout Samsara is the code name for the aggregate of all these new features working together from the 0.10.0 release on.[2]

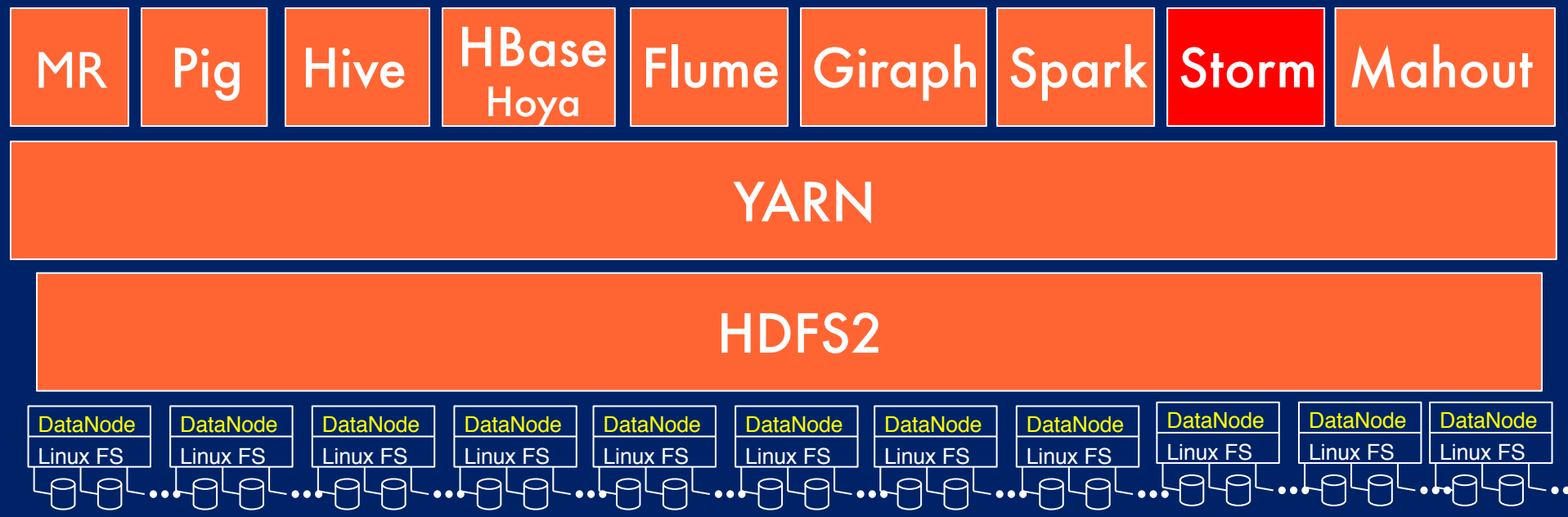
Hadoop2.0

- Mahout: Apache's scalable machine learning platform, runs on Hadoop/Spark: its provides tools for using Hadoop to create systems that perform clustering, classification, and collaborative filtering at high speed on large data sets.
- No longer based specifically on MapReduce



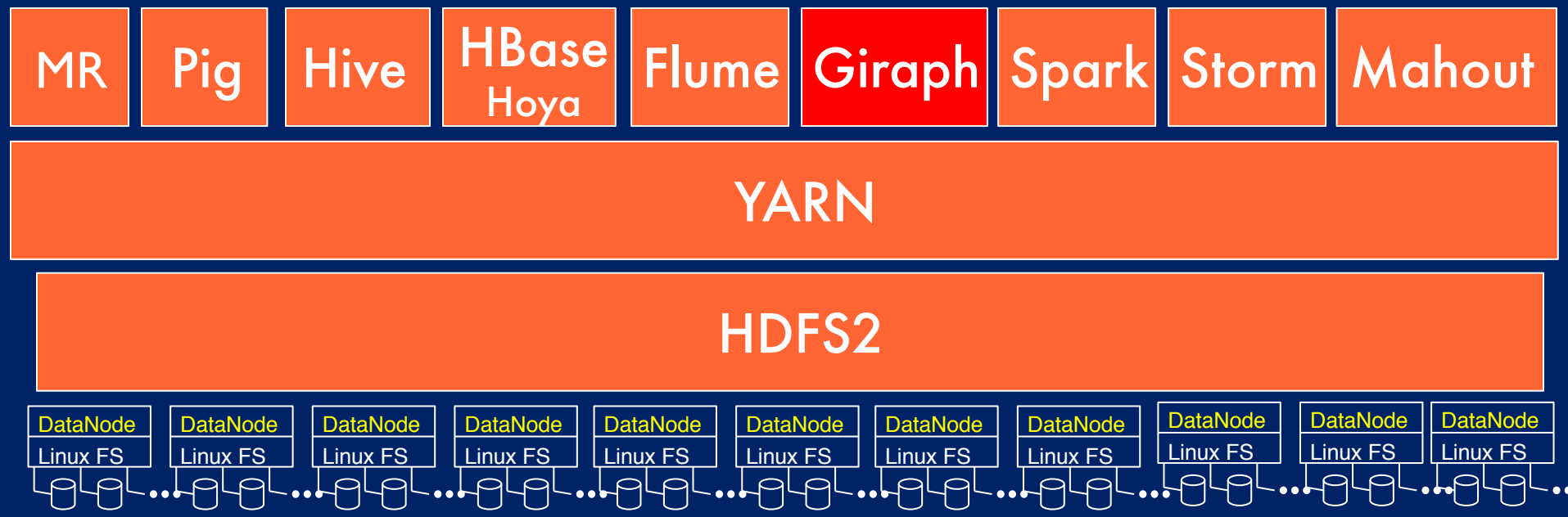
Hadoop2.0

- Storm



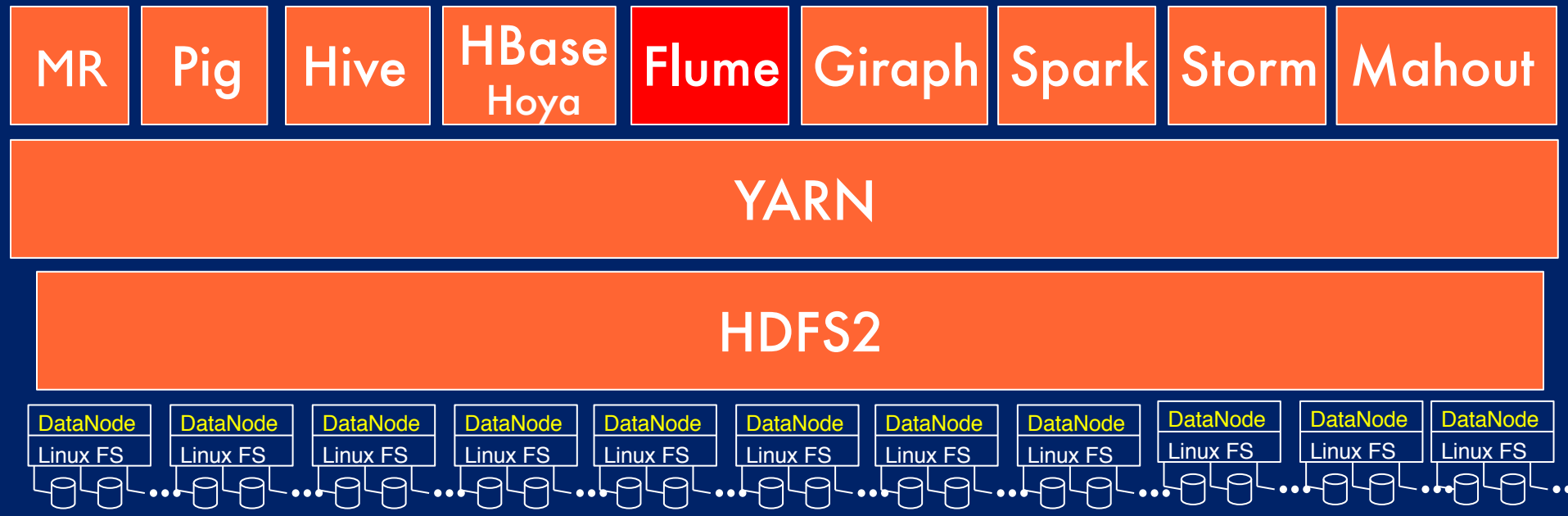
Hadoop2.0

- Giraph



Hadoop2.0

- Apache Flume



Ha

• A

FlumeJava: Easy, Efficient Data-Parallel Pipelines

Craig Chambers, Ashish Raniwala, Frances Perry,
Stephen Adams, Robert R. Henry,
Robert Bradshaw, Nathan Weizenbaum
Google, Inc.
{chambers,raniwala,fjp,sra,rrh,robertwb,nweiz}@google.com

Abstract

MapReduce and similar systems significantly ease the task of writing data-parallel code. However, many real-world computations require a pipeline of MapReduces, and programming and managing such pipelines can be difficult. We present FlumeJava, a Java library that makes it easy to develop, test, and run efficient data-parallel pipelines. At the core of the FlumeJava library are a couple of classes that represent immutable parallel collections, each supporting a modest number of operations for processing them in parallel. Parallel collections and their operations present a simple, high-level, uniform abstraction over different data representations and execution strategies. To enable parallel operations to run efficiently, FlumeJava defers their evaluation, instead internally constructing an execution plan dataflow graph. When the final results of the parallel operations are eventually needed, FlumeJava first optimizes the execution plan, and then executes the optimized operations on appropriate underlying primitives (e.g., MapReduces). The combination of high-level abstractions for parallel data and computation, deferred evaluation and optimization, and efficient parallel primitives yields an easy-to-use system that approaches the efficiency of hand-optimized pipelines. FlumeJava is in active use by hundreds of pipeline developers within Google.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel Programming

General Terms Algorithms, Languages, Performance

Keywords data-parallel programming, MapReduce, Java

1. Introduction

Building programs to process massive amounts of data in parallel can be very hard. MapReduce [6–8] greatly eased this task for data-parallel computations. It presented a simple abstraction to users for how to think about their computation, and it managed many of the difficult low-level tasks, such as distributing and coordinating the parallel work across many machines, and coping robustly with failures of machines, networks, and data. It has been used very successfully in practice by many developers. MapReduce's success in this domain inspired the development of a number of related systems, including Hadoop [2], LINQ/Dryad [20], and Pig [3].

MapReduce works well for computations that can be broken down into a map step, a shuffle step, and a reduce step, but for many real-world computations, a chain of MapReduce stages is required. Such data-parallel *pipelines* require additional coordination code to chain together the separate MapReduce stages, and require additional work to manage the creation and later deletion of the intermediate results between pipeline stages. The logical computation can become obscured by all these low-level coordination details, making it difficult for new developers to understand the computation. Moreover, the division of the pipeline into particular stages becomes "baked in" to the code and difficult to change later if the logical computation needs to evolve.

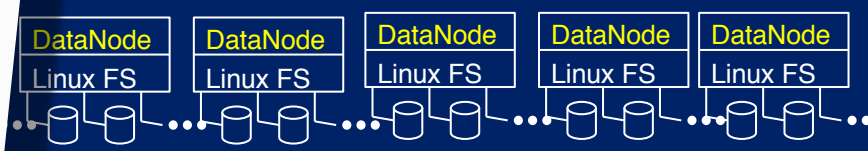
In this paper we present FlumeJava, a new system that aims to support the development of data-parallel pipelines. FlumeJava is a Java library centered around a few classes that represent *parallel collections*. Parallel collections support a modest number of *parallel operations* which are composed to implement data-parallel computations. An entire pipeline, or even multiple pipelines, can be implemented in a single Java program using the FlumeJava abstractions; there is no need to break up the logical computation into separate programs for each stage.

FlumeJava's parallel collections abstract away the details of how data is represented, including whether the data is represented as an in-memory data structure, as one or more files, or as an external storage service such as a MySql database or a Bigtable [5]. Similarly, FlumeJava's parallel operations abstract away their implementation strategy, such as whether an operation is implemented as a local sequential loop, or as a remote parallel MapReduce invocation, or (in the future) as a query on a database or as a streaming computation. These abstractions enable an entire pipeline to be initially developed and tested on small in-memory test data, running in a single process, and debugged using standard Java IDEs and debuggers, and then run completely unchanged over large production data. They also confer a degree of adaptability of the logical FlumeJava computations as new data storage mechanisms and execution services are developed.

To achieve good performance, FlumeJava internally implements parallel operations using *deferred evaluation*. The invocation of a parallel operation does not actually run the operation, but instead simply records the operation and its arguments in an internal *execution plan* graph structure. Once the execution plan for the whole computation has been constructed, FlumeJava *optimizes* the execution plan, for example *fusing* chains of parallel operations together into a small number of MapReduce operations. FlumeJava then runs the optimized execution plan. When running the execution plan, FlumeJava chooses which strategy to use to implement each operation (e.g., local sequential loop vs. remote parallel MapReduce, based in part on the size of the data being processed), places remote computations near the data they operate on, and per-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PLDI'10, June 5–10, 2010, Toronto, Ontario, Canada
Copyright © 2010 ACM 978-1-4503-0019-3/10/06...$10.00

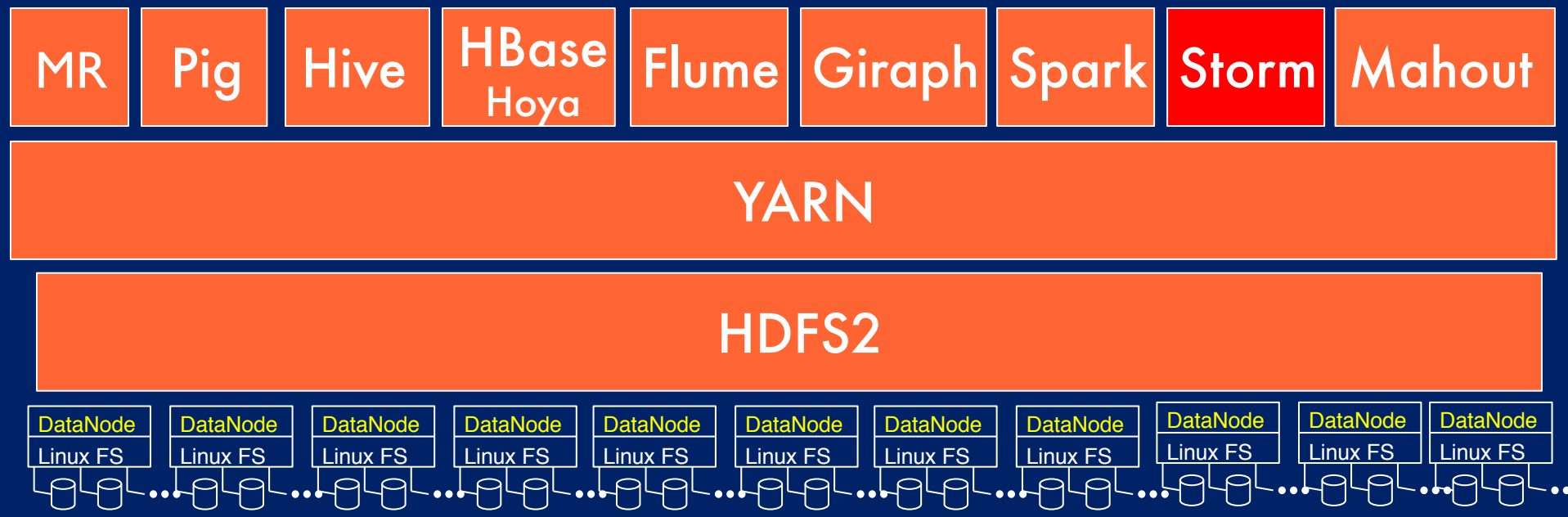
aph | Spark | Storm | Mahout



ight © 2019, Dave Cliff

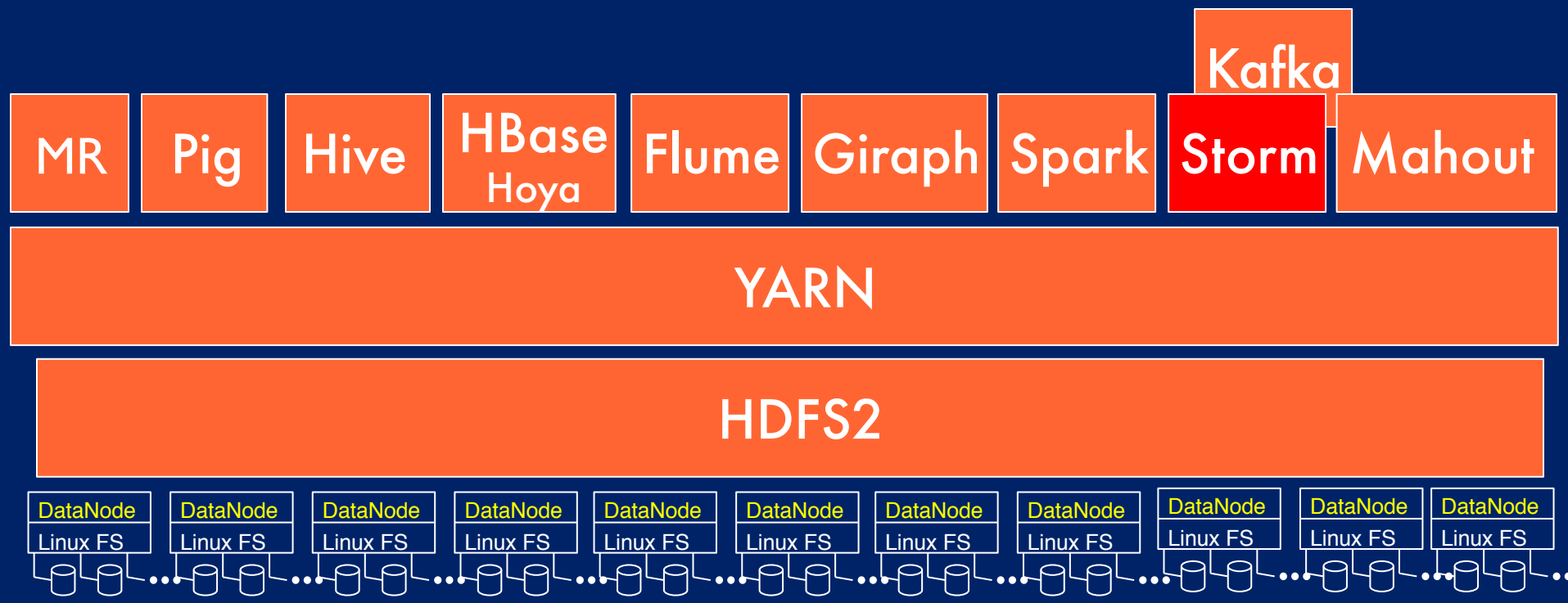
Hadoop2.0

- Storm: Apache's first true stream-processing system



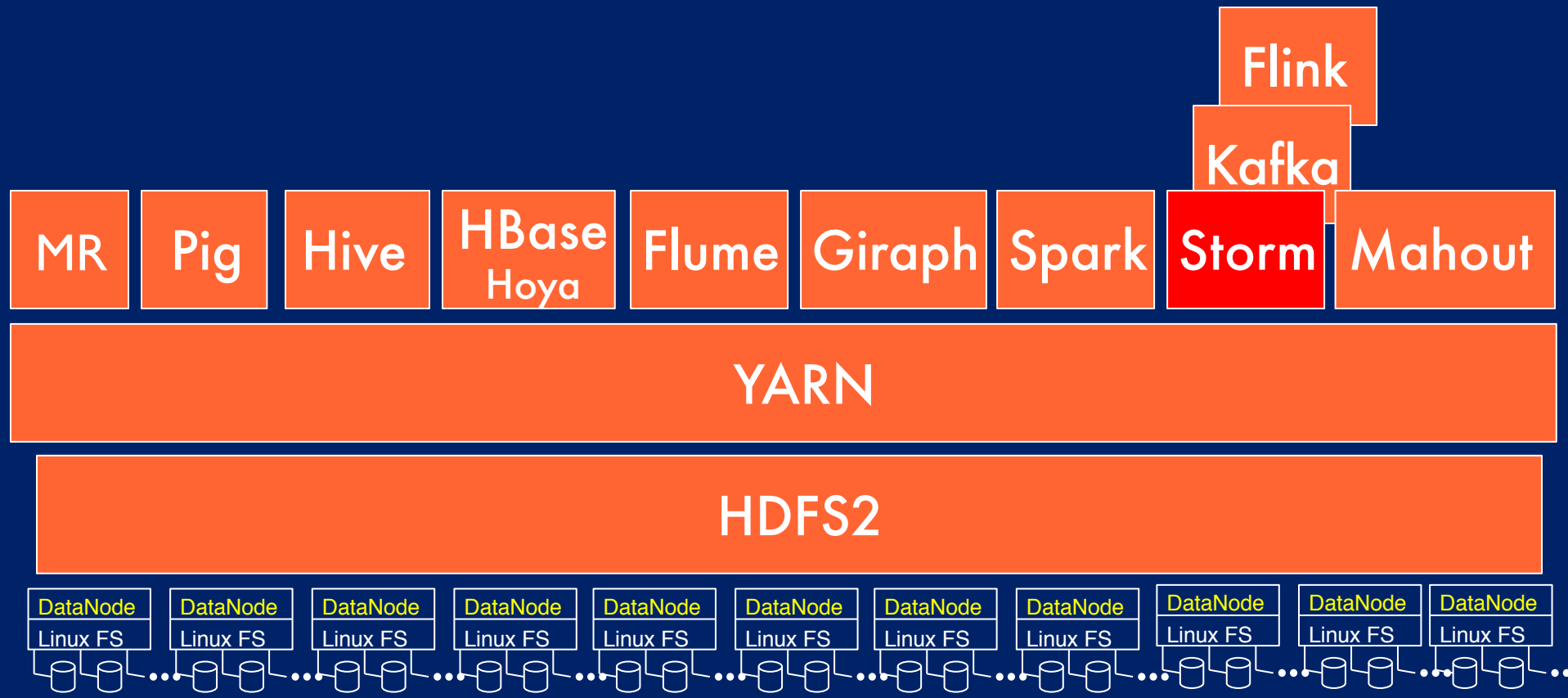
Hadoop2.0

- Storm: Apache's first true stream-processing system
- Kafka: pub-sub data exchange/messaging

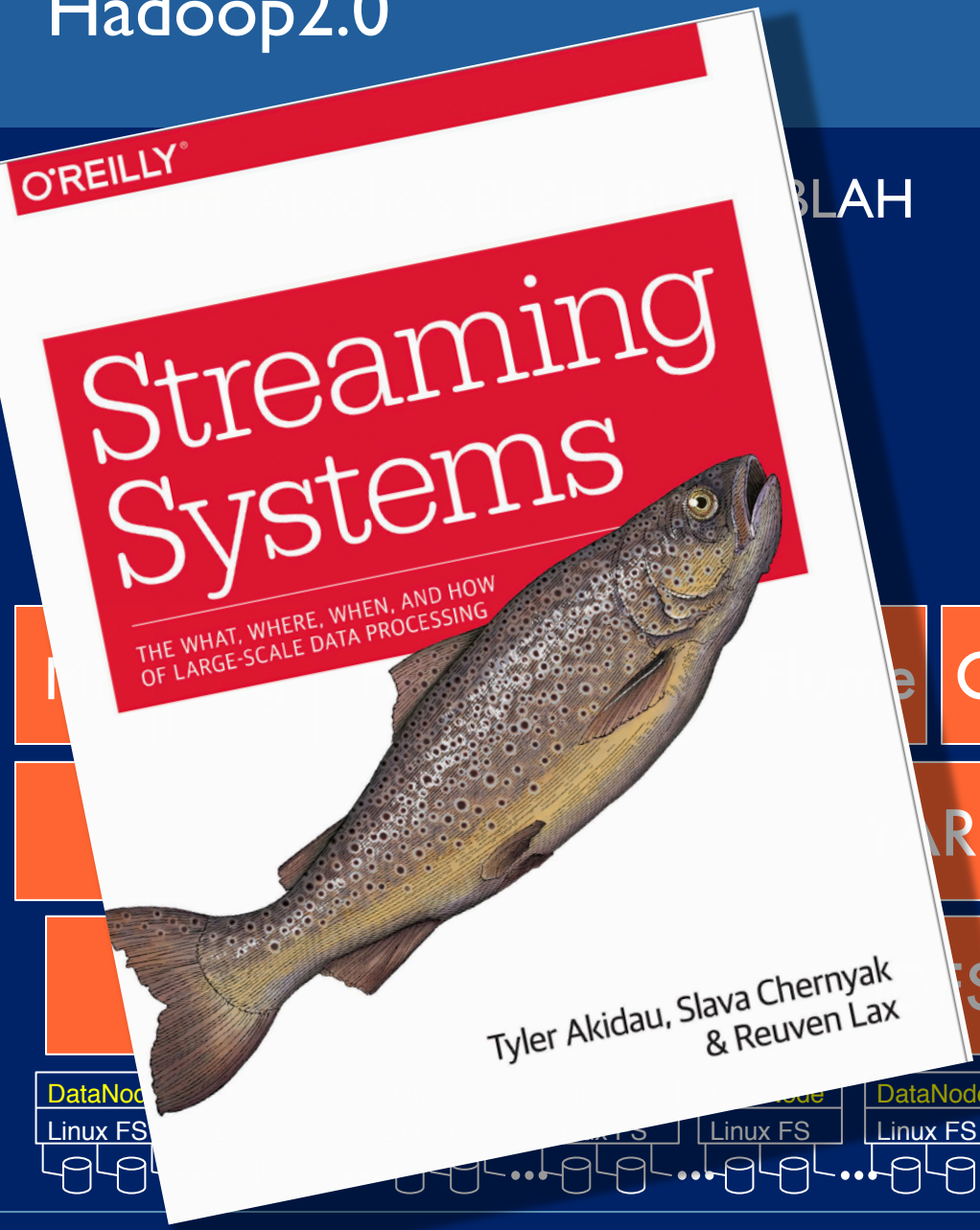


Hadoop2.0

- Storm: Apache's first true stream-processing system
- Kafka: pub-sub data exchange/messaging
- Flink: high-efficiency stream-processing



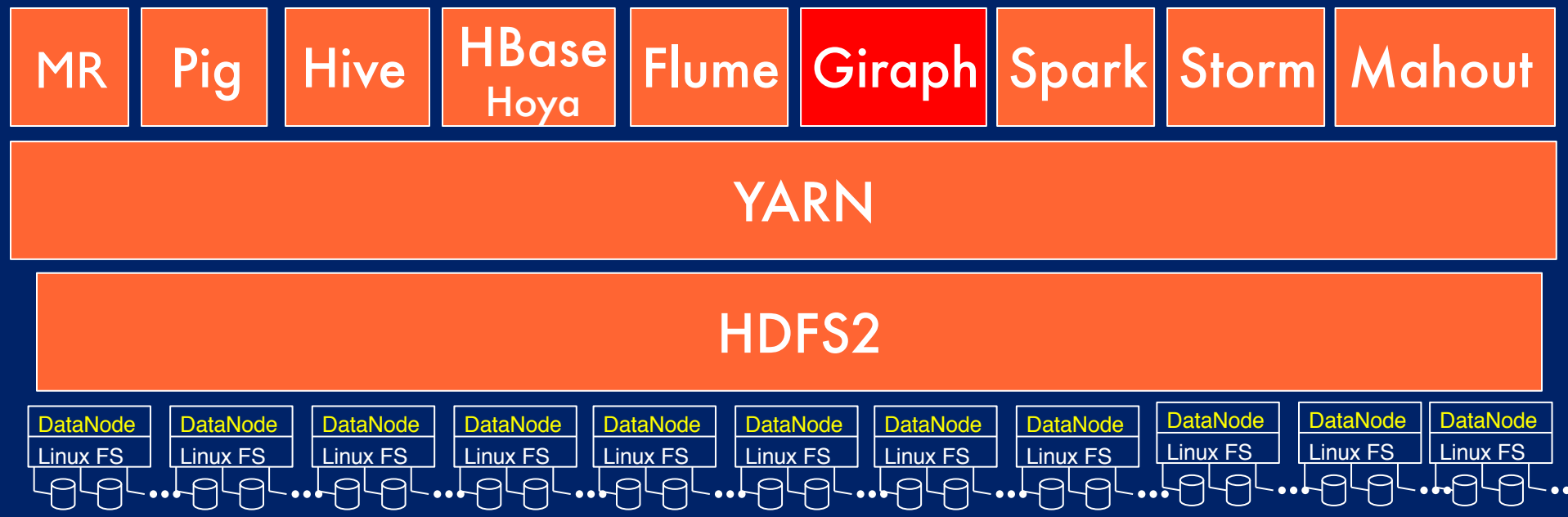
Hadoop2.0


O'REILLY
Streaming Systems
THE WHAT, WHERE, WHEN, AND HOW OF LARGE-SCALE DATA PROCESSING
Tyler Akidau, Slava Chernyak & Reuven Lax

Flink
Kafka
Giraph
Spark
Storm
Mahout
DataNode
Linux FS

Hadoop2.0

- Giraph: a graph database system



Hadoop2.0

- Giraph: a graph database system

TAO: Facebook's Distributed Data Store for the Social Graph

Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov
Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov
Dmitri Petrov, Lovro Puzar, Yee Jiun Song, Venkat Venkataramani
Facebook, Inc.

Abstract

We introduce a simple data model and API tailored for serving the social graph, and TAO, an implementation of this model. TAO is a geographically distributed data store that provides efficient and timely access to the social graph for Facebook's demanding workload using a fixed set of queries. It is deployed at Facebook, replacing memcache for many data types that fit its model. The system runs on thousands of machines, is widely distributed, and provides access to many petabytes of data. TAO can process a billion reads and millions of writes each second.

1 Introduction

Facebook has more than a billion active users who record their relationships, share their interests, upload text, images, and video, and curate semantic information about their data [2]. The personalized experience of social applications comes from timely, efficient, and scalable access to this flood of data, the *social graph*. In this paper we introduce TAO, a read-optimized graph data store we have built to handle a demanding Facebook workload.

Before TAO, Facebook's web servers directly accessed MySQL to read or write the social graph, aggressively using memcache [21] as a lookaside cache. TAO implements a graph abstraction directly, allowing it to avoid some of the fundamental shortcomings of a lookaside cache architecture. TAO continues to use MySQL for persistent storage, but mediates access to the database and uses its own graph-aware cache.

TAO is deployed at Facebook as a single geographically distributed instance. It has a minimal API and explicitly favors availability and per-machine efficiency over strong consistency; its novelty is its scale: TAO can sustain a billion reads per second on a changing data set of many petabytes.

Overall, this paper makes three contributions. We motivate (§ 2) and characterize (§ 7) a challenging workload: efficient and available read-mostly access to a changing graph. We describe objects and associations, a data model and API that we use to access the graph (§ 3). Lastly, we detail TAO, a geographically distributed system that implements this API (§§ 4–6), and evaluate its performance on our workload (§ 8).

a) Alice was at the Golden Gate Bridge with Bob
Cathy : Wish we were there! David likes this

b) id: 105, otype: USER name: Alice
id: 534, otype: LOCATION name: Golden Gate Bridge loc: 37°49'11"N, 122°28'43"W
id: 244, otype: USER name: Bob
id: 632, otype: CHECKIN
id: 379, otype: USER name: Cathy
id: 771, otype: COMMENT text: Wish we were there!
id: 471, otype: USER name: David
FRIEND AUTHORED_BY AUTHORED LOC CHKIN TAGGED TAGGED_AT otype: COMMENT time: 1334511670 LIKED_BY LIKES

Figure 1: A running example of how a user's checkin might be mapped to objects and associations.

2 Background

A single Facebook page may aggregate and filter hundreds of items from the social graph. We present each user with content tailored to them, and we filter every item with privacy checks that take into account the current viewer. This extreme customization makes it infeasible to perform most aggregation and filtering when content is created; instead we resolve data dependencies and check privacy each time the content is viewed. As much as possible we pull the social graph, rather than pushing it. This implementation strategy places extreme read demands on the graph data store; it must be efficient, highly available, and scale to high query rates.

2.1 Serving the Graph from Memcache

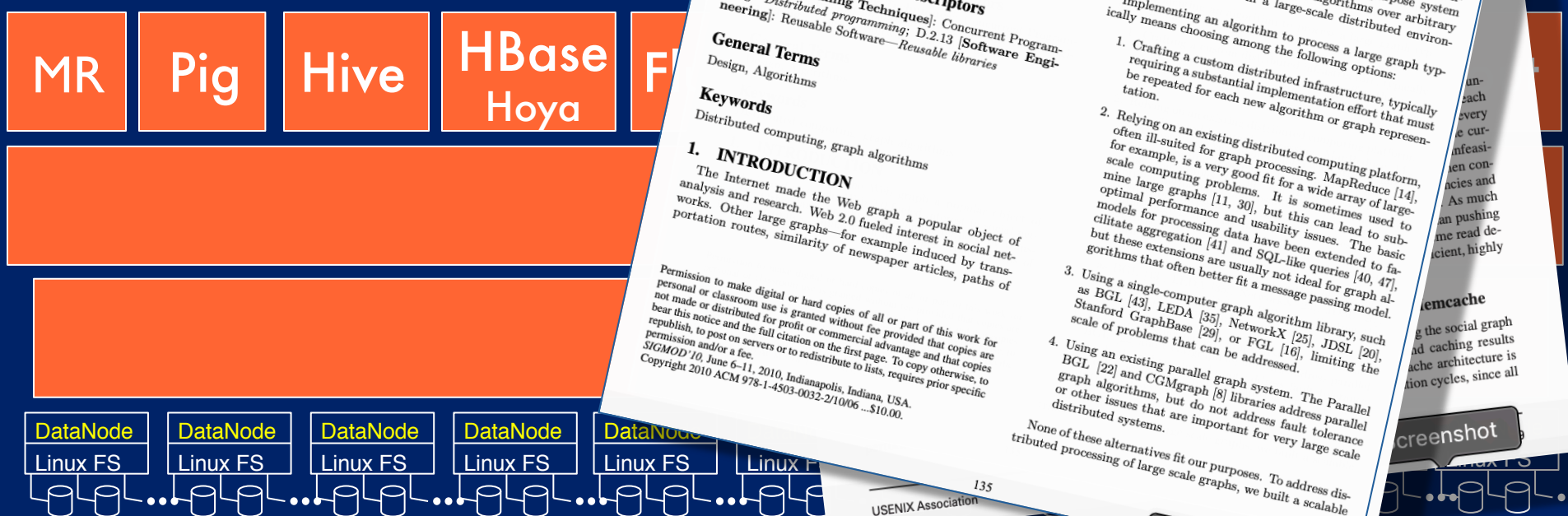
Facebook was originally built by storing the social graph in MySQL, querying it from PHP, and caching results in memcache [21]. This lookaside cache architecture is well suited to Facebook's rapid iteration cycles, since all

USENIX Association 2013 USENIX Annual Technical Co

Screenshot

Hadoop2.0

- Giraph: a graph database system



Pregel: A System for Large-Scale Graph Processing

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski
Google, Inc.
{malewicz,austern,ajcbik,dehnert,ilan,naty,gczaj}@google.com

ABSTRACT

Many practical computing problems concern large graphs. Standard examples include the Web graph and various social networks. The scale of these graphs—in some cases billions of vertices, trillions of edges—poses challenges to their efficient processing. In this paper we present a computational model suitable for this task. Programs are expressed as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. This vertex-centric approach is flexible enough to express a broad set of algorithms. The model has been designed for efficient, scalable and fault-tolerant implementation on clusters of thousands of commodity computers, and its implied synchronicity makes reasoning about programs easier. Distribution-related details are hidden behind an abstract API. The result is a framework for processing large graphs that is expressive and easy to program.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*; D.2.13 [**Software Engineering**]: Reusable Software—*Reusable libraries*

General Terms

Design, Algorithms

Keywords

Distributed computing, graph algorithms

1. INTRODUCTION

The Internet made the Web graph a popular object of analysis and research. Web 2.0 fueled interest in social networks. Other large graphs—for example induced by transportation routes, similarity of newspaper articles, paths of disease outbreaks, or citation relationships among published scientific work—have been processed for decades. Frequently applied algorithms include shortest paths computations, different flavors of clustering, and variations on the page rank theme. There are many other graph computing problems of practical value, *e.g.*, minimum cut and connected components.

Efficient processing of large graphs is challenging. Graph algorithms often exhibit poor locality of memory access, very little work per vertex, and a changing degree of parallelism over the course of execution [31, 39]. Distribution over many machines exacerbates the locality issue, and increases the probability that a machine will fail during computation. Despite the ubiquity of large graphs and their commercial importance, we know of no scalable general-purpose system for implementing arbitrary graph algorithms over arbitrary graph representations in a large-scale distributed environment.

Implementing an algorithm to process a large graph typically means choosing among the following options:

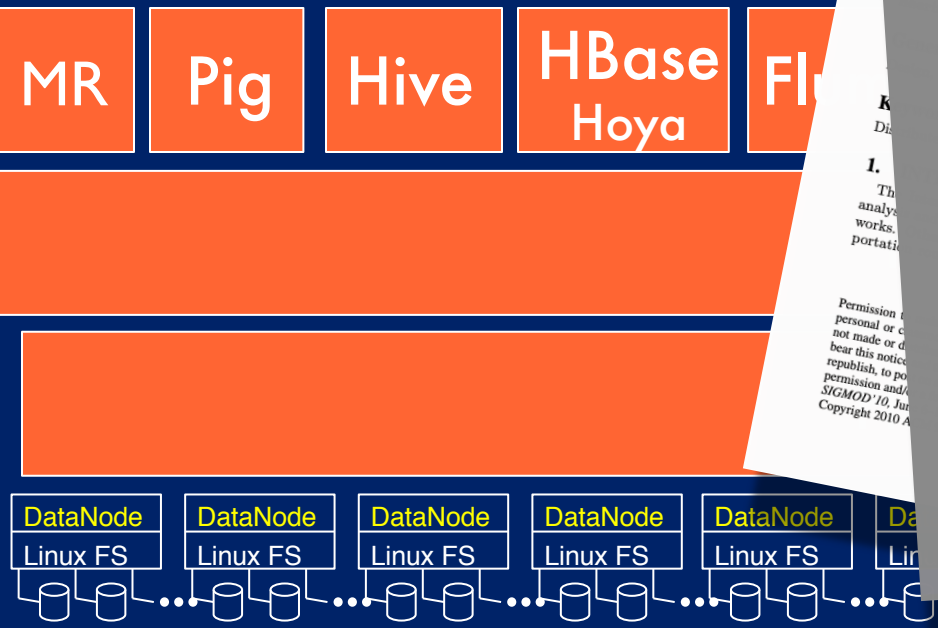
1. Crafting a custom distributed infrastructure, typically requiring a substantial implementation effort that must be repeated for each new algorithm or graph representation.
2. Relying on an existing distributed computing platform, often ill-suited for graph processing. MapReduce [14], for example, is a very good fit for a wide array of large-scale computing problems. It is sometimes used to mine large graphs [11, 30], but this can lead to sub-optimal performance and usability issues. The basic models for processing data have been extended to facilitate aggregation [41] and SQL-like queries [40, 47], but these extensions are usually not ideal for graph algorithms that often better fit a message passing model.
3. Using a single-computer graph algorithm library, such as BGL [43], LEDA [35], NetworkX [25], JDSL [20], Stanford GraphBase [29], or FGL [16], limiting the scale of problems that can be addressed.
4. Using an existing parallel graph system. The Parallel BGL [22] and CGMgraph [8] libraries address parallel graph algorithms, but do not address fault tolerance or other issues that are important for very large scale distributed systems.

None of these alternatives fit our purposes. To address distributed processing of large scale graphs, we built a scalable

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.
Copyright 2010 ACM 978-1-4503-0032-2/10/06 ...$10.00.

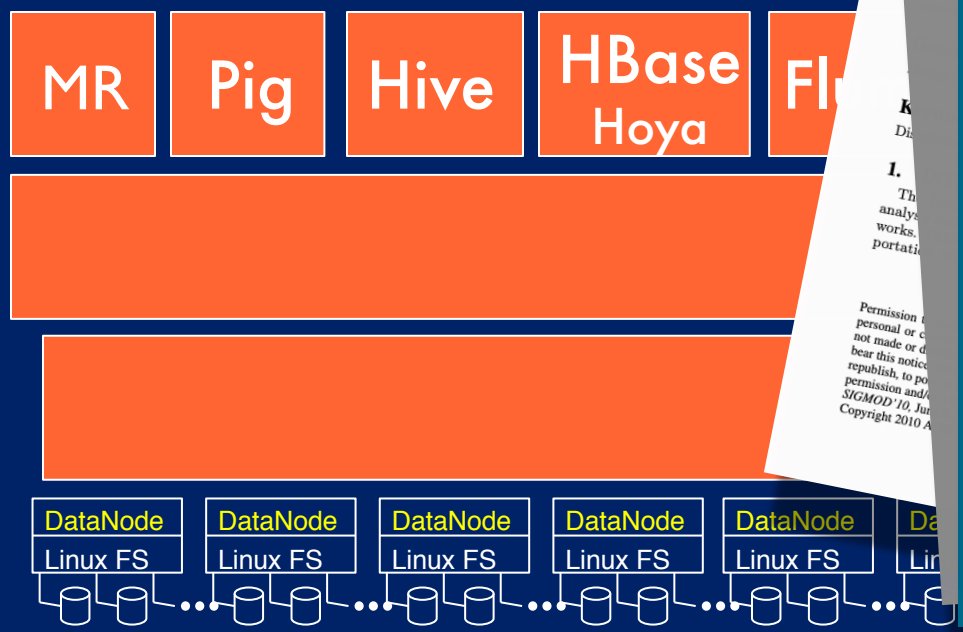
Hadoop2.0

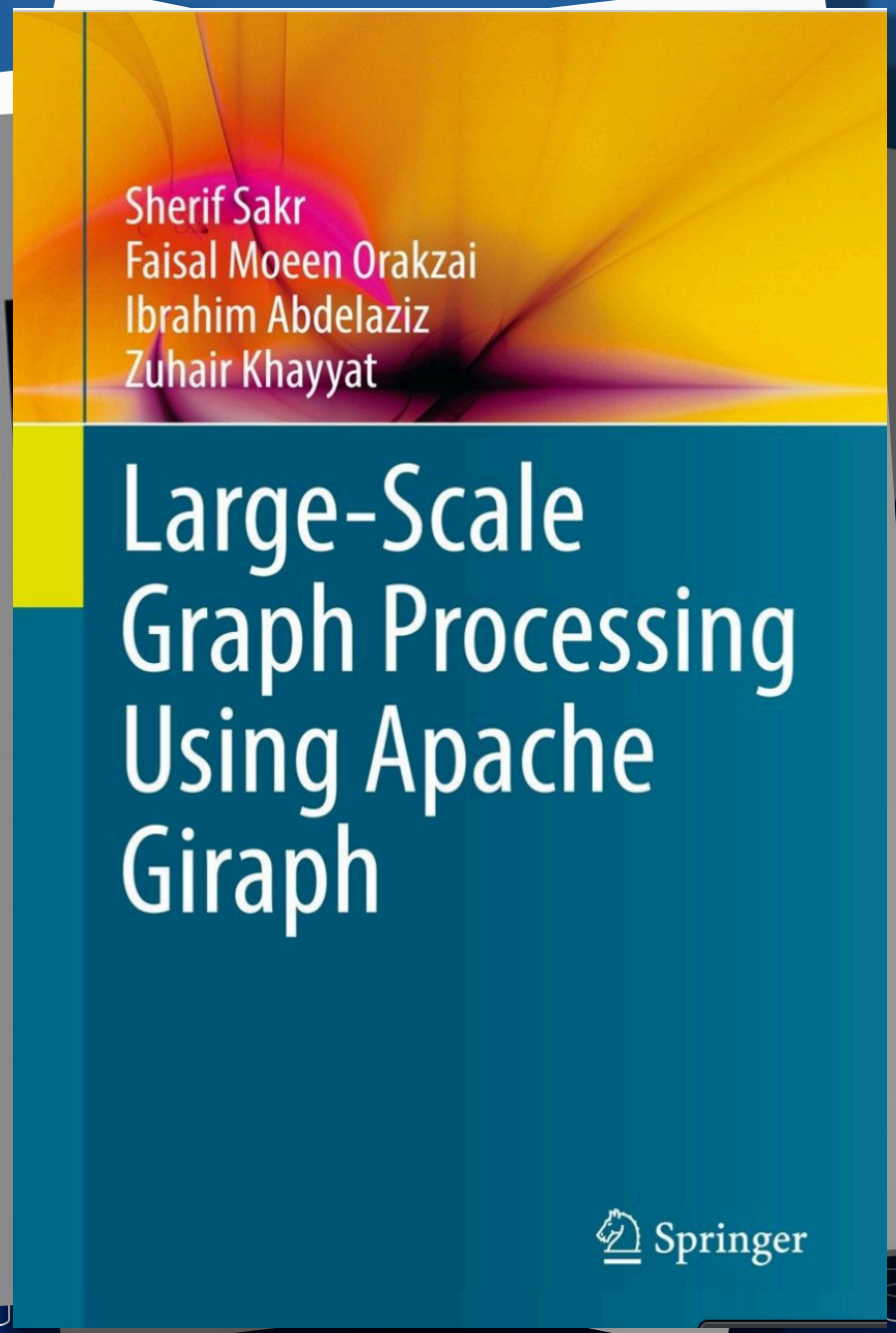
- Giraph: a graph database system



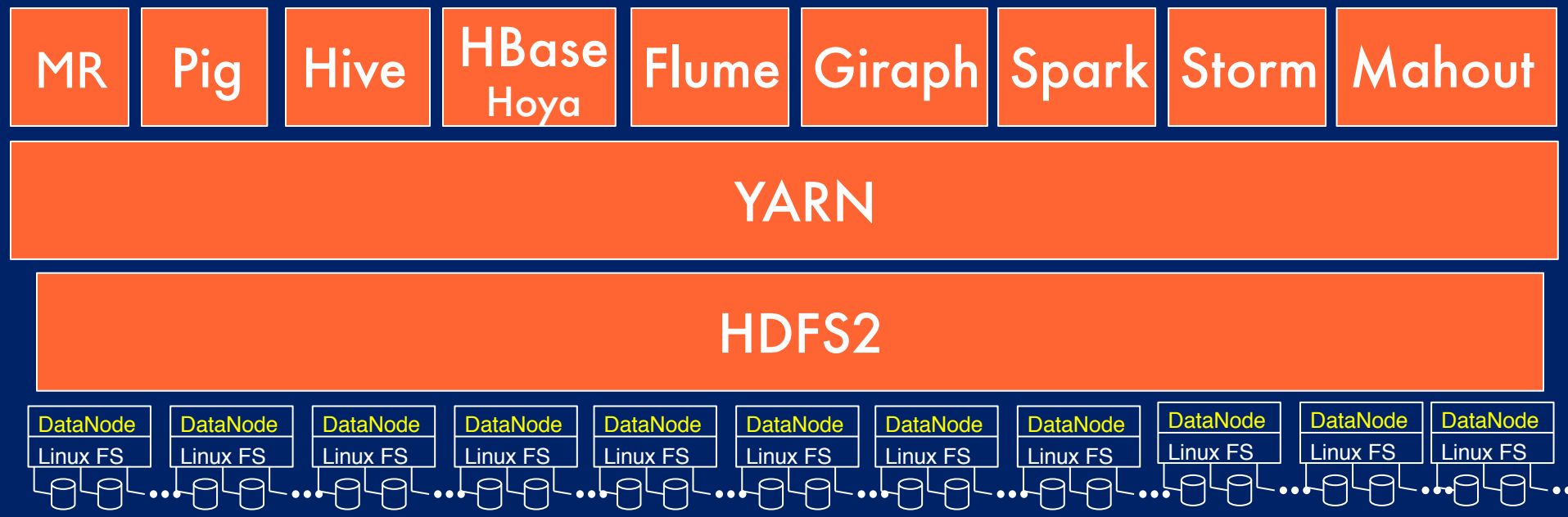
Hadoop2.0

- Giraph: a graph database system





Hadoop2.0

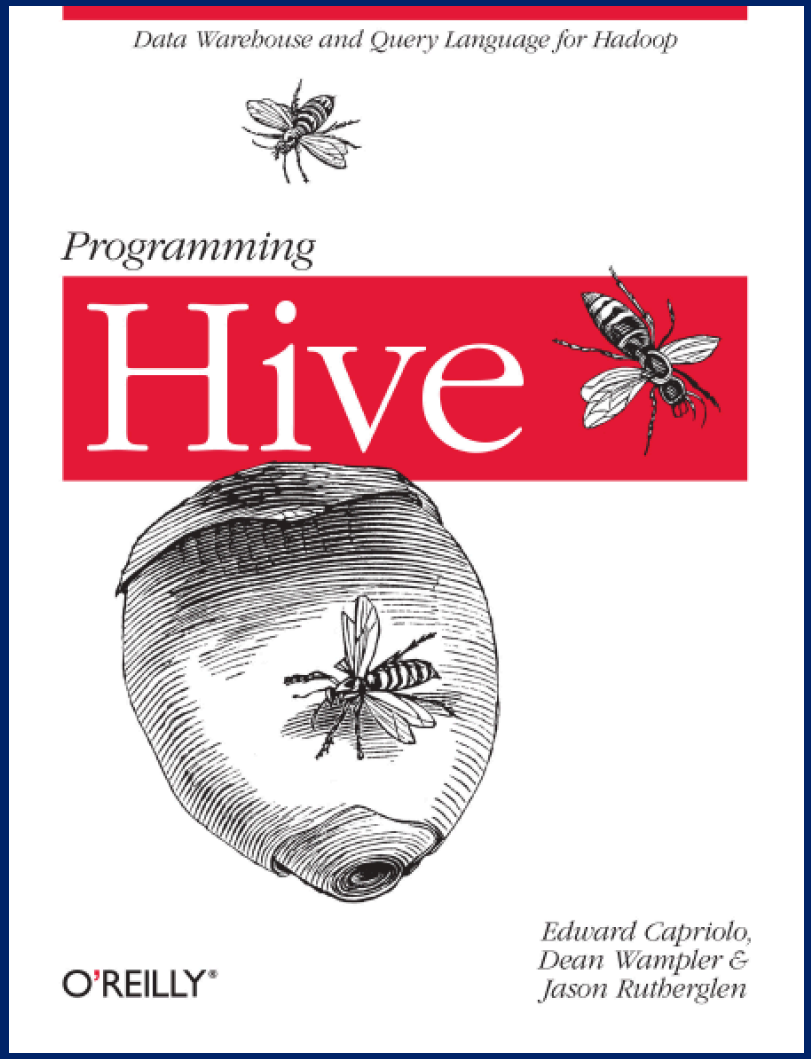

MR
Pig
Hive
HBase
Hoya
Flume
Giraph
Spark
Storm
Mahout
YARN
HDFS2
DataNode
Linux FS

Hive

Hive basics

- Best-suited to *Data Warehousing*
  - Relatively static data
  - Response time not critical
- Initially developed at Facebook, later made open-source via Apache
- Used & developed by Netflix, AWS
- Works on HDFS, and also AWS S3
- HiveQL/HQL
  SQL-like query language
  Converted to MR jobs (or to Spark)

- HQL much more concise and clear than lower-level MR-code



Hive architecture

JDBC: Java Database Connectivity

ODBC: Open Database Connectivity

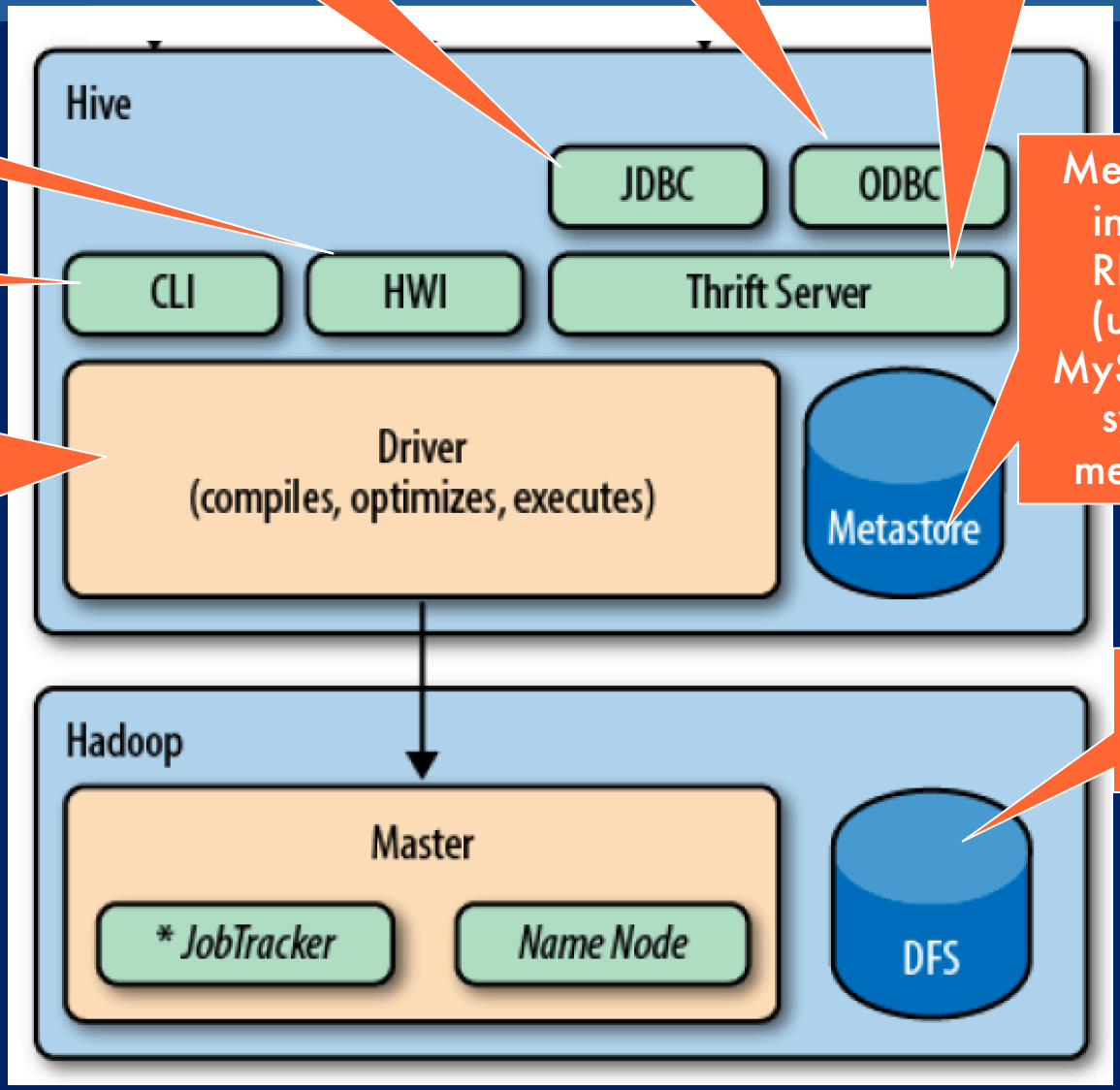
Thrift: cross-language remote procedure calling

HWI: Hive Web Interface

CLI: Command Line Interface

Driver:

compiles input; optimizes computation; & executes subjobs (usually via MR)



Metastore: internal RDBMS (usually MySQL) for system metadata

HDFS, S3, etc

Hadoop MapReduce Wordcount in Java

```java
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }

 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

nt © 2019, Dave Cliff

Hadoop MapReduce Wordcount in Hive / HQL

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Hadoop MapReduce Wordcount in Hive / HQL

Create a new data-table called docs
with a single column called line.

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

docs

line

Hadoop MapReduce Wordcount in Hive / HQL

Fill the table called docs
with data from the file called docs.

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

docs

line
the cat sat on the mat

Hadoop MapReduce Wordcount in Hive / HQL

Create a new data-table called word-counts
By doing the following…

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Hadoop MapReduce Wordcount in Hive / HQL

`count(1)` counts the number of rows that are not NULL

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Hadoop MapReduce Wordcount in Hive / HQL

explode(X) generates a table from a list/array X with each element of X on a separate row

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Hadoop MapReduce Wordcount in Hive / HQL

split(X) generates an array of substrings from line, split on occurrence of the pattern '\\s' (i.e., whitespace)

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Hadoop MapReduce Wordcount in Hive / HQL

explode(split(line,'\\s'))) **generates a temporary table** w…

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

w

the
cat
sat
on
the
mat

Hadoop MapReduce Wordcount in Hive / HQL

Make each row of word-counts
Be a (word, count) pair where …

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Hadoop MapReduce Wordcount in Hive / HQL

GROUP BY groups the result set by the specified column(s)

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

word_counts

word	count
the	2
cat	1
sat	1
on	1
mat	2

Hadoop MapReduce Wordcount in Hive / HQL

ORDER BY produces a total ordering of the query result set

```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

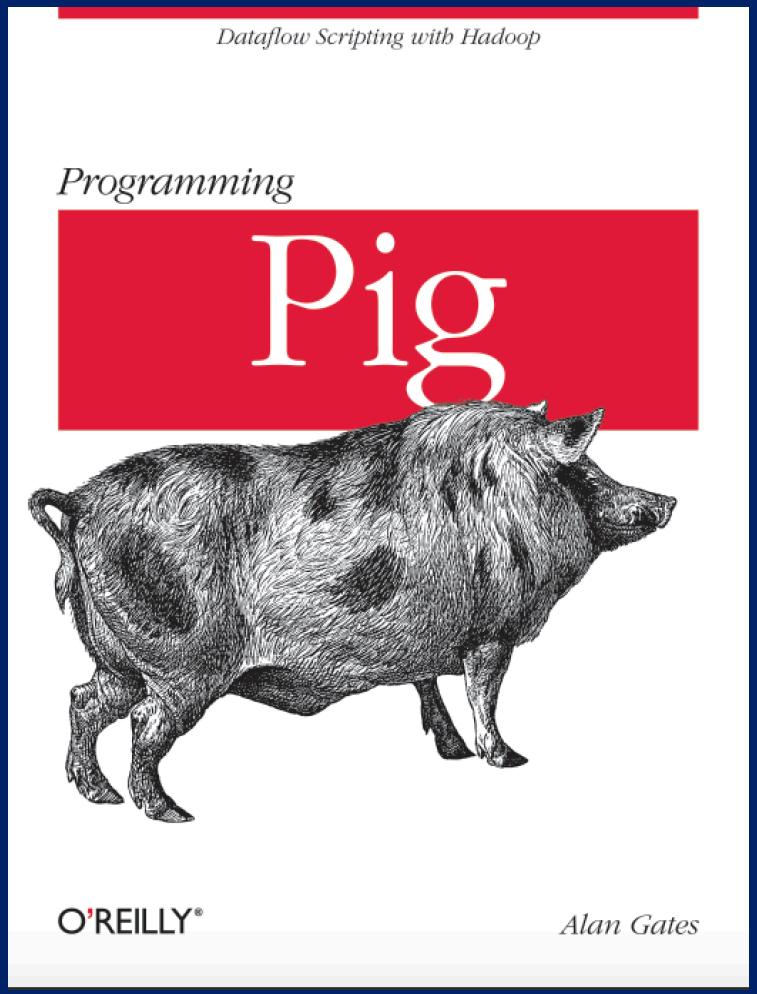
word_counts

word	count
cat	1
mat	1
on	1
sat	1
the	2

Pig

Pig basics

- Initially developed at Yahoo, later made open-source via Apache
- Pig: Pig Latin; Grunt; PiggyBank
- Works on HDFS, & noSQL (HBase etc)
- PigLatin is a Dataflow Language
- Users describe how data is read/processed/stored, in parallel
- Dataflows could be linear or DAG
- PigLatin has no if/for/while (control flow)
- Unlike SQL/HQL, Pig allows user to describe exactly how data will be processed (declarative langs just say what answer they want, not how it is generated)



2011

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, calling the single
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as word;

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, call
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as word;

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

This loads data in from a file called "Mary", give it field-name 'line'

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, calli the single
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as word;

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

This splits each line into separate words

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned
-- TOKENIZE and produce a separate record for each  alling the single
-- field in the record word.
words = foreach input generate flatten(TOK  E(line)) as word;

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

Group together by words

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, callin  single
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as w

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

Count the number of words in each group

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned
-- TOKENIZE and produce a separate record for each word, calling the single
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as word;

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

Generate printout

Hadoop MapReduce Wordcount in PigLatin

Example 1-1. Pig counts Mary and her lamb

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);

-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, calling the single
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as word;

-- Now group them together by each word.
grpd  = group words by word;

-- Count them.
cntd  = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

Cascading

(Hadoop Ecosystem? Probably not)

Cascading

www.cascading.org

Cascading | Application Pla

www.cascading.org

CASCADING

FORUMS PRODUCTS DOWNLOADS SDK DRIVEN LOGIN Go

PROJECTS EXTENSIONS RESOURCES SUPPORT NEWS

Cascading is the proven application development platform for building data applications on Hadoop.

GET CASCADING 2.6

PROJECTS

CASCADING

LINGUAL	Simplifies systems integration through ANSI SQL compatibility and a JDBC driver
PATTERN	Enables various machine learning scoring algorithms through PMML compatibility
SCALDING	Enables development with Scala, a powerful language for solving functional problems
CASCALOG	Enables development with Clojure, a Lisp dialect

RESOURCES

- DOCUMENTATION
- TUTORIALS
- TRAINING
- DRIVEN
- SUPPORT
- DOWNLOADS

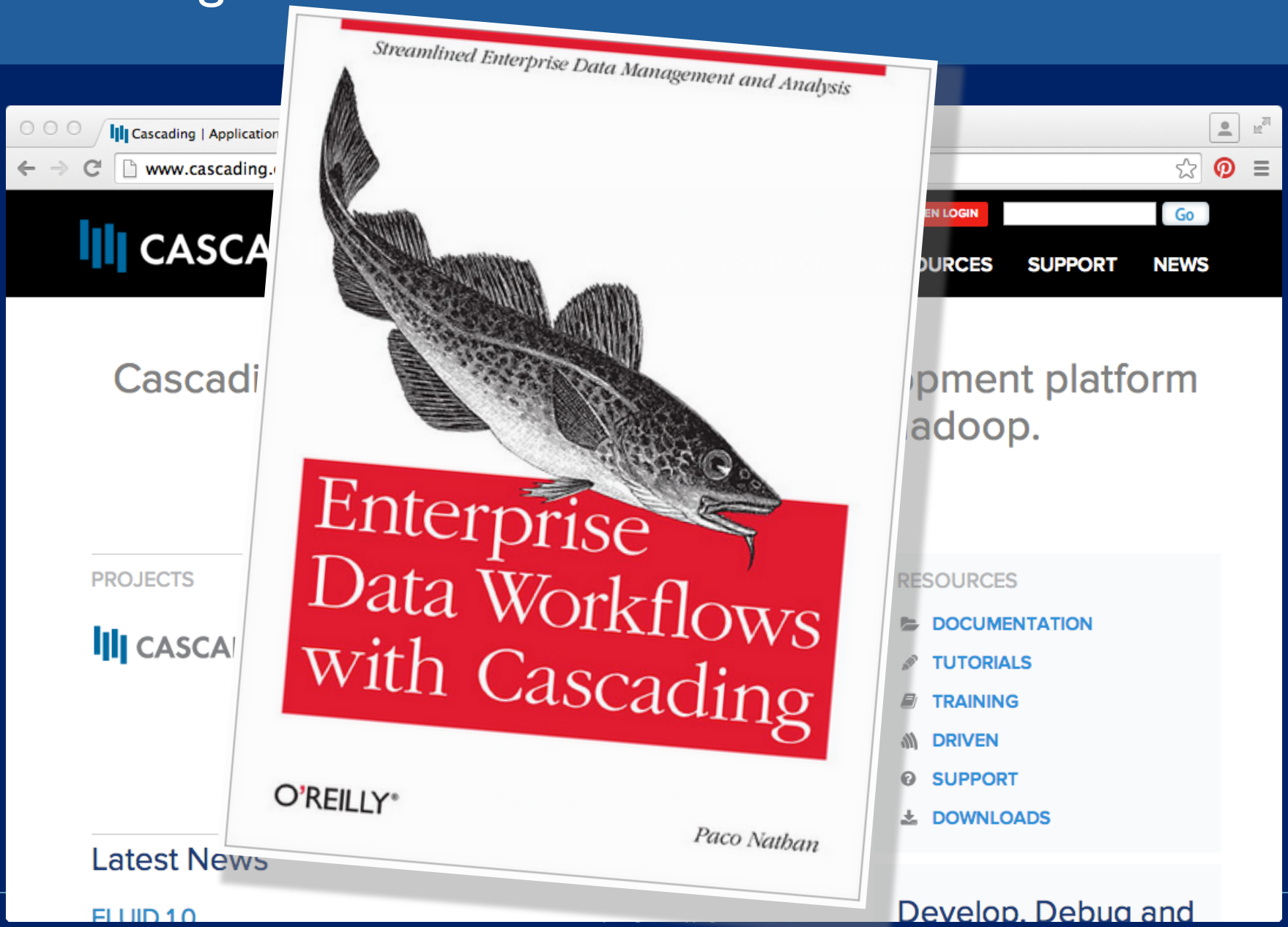
Latest News

FLUID 1.0

Develop, Debug and

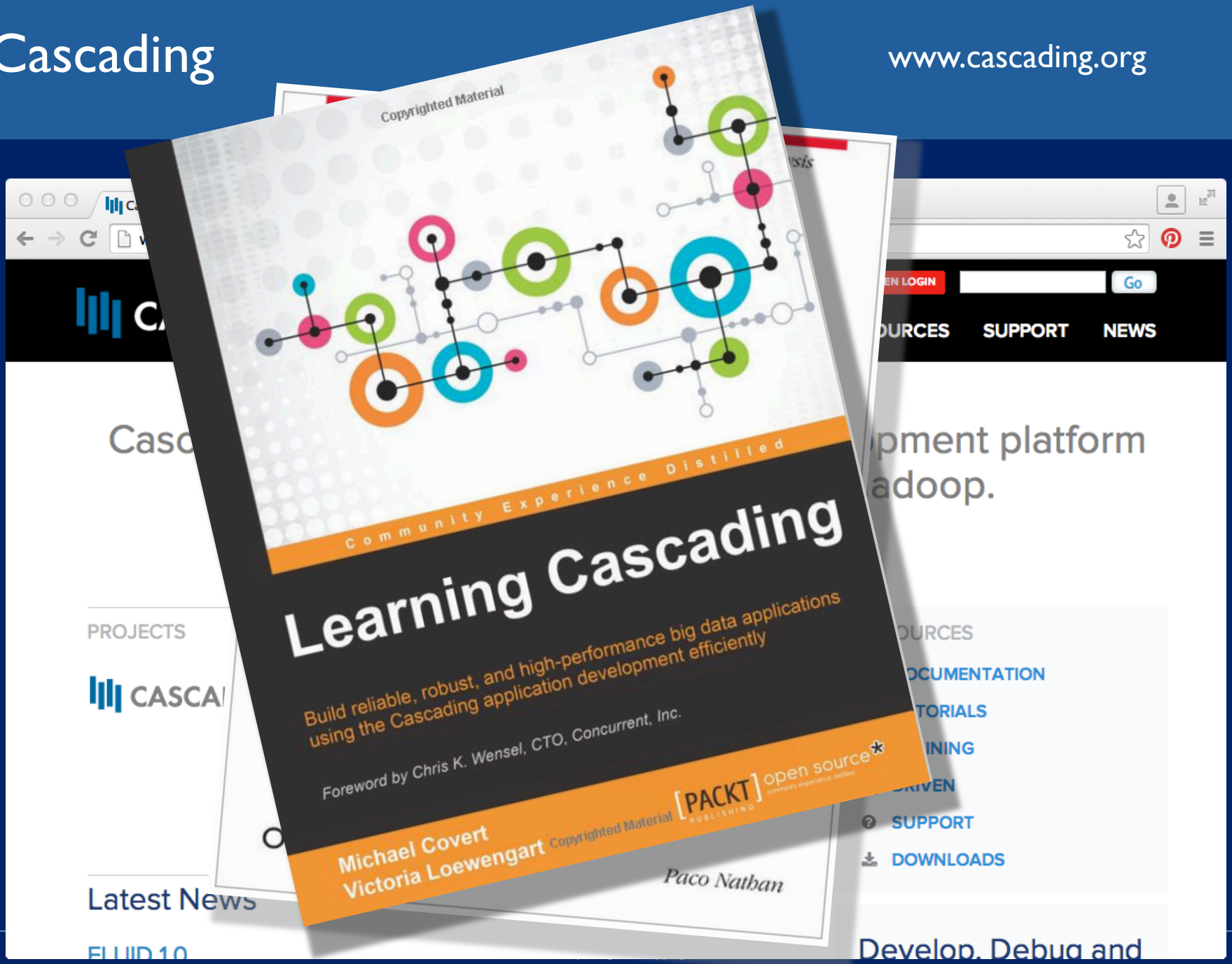
Cascading

www.cascading.org



Cascading

www.cascading.org



Cascading

www.cascading.org

Cascading | Application Pla

www.cascading.org

CASCADING

FORUMS PRODUCTS DOWNLOADS SDK DRIVEN LOGIN Go

PROJECTS EXTENSIONS RESOURCES SUPPORT NEWS

Cascading is the proven application development platform for building data applications on Hadoop.

GET CASCADING 2.6

PROJECTS

CASCADING

LINGUAL	Simplifies systems integration through ANSI SQL compatibility and a JDBC driver
PATTERN	Enables various machine learning scoring algorithms through PMML compatibility
SCALDING	Enables development with Scala, a powerful language for solving functional problems
CASCALOG	Enables development with Clojure, a Lisp dialect

RESOURCES

- DOCUMENTATION
- TUTORIALS
- TRAINING
- DRIVEN
- SUPPORT
- DOWNLOADS

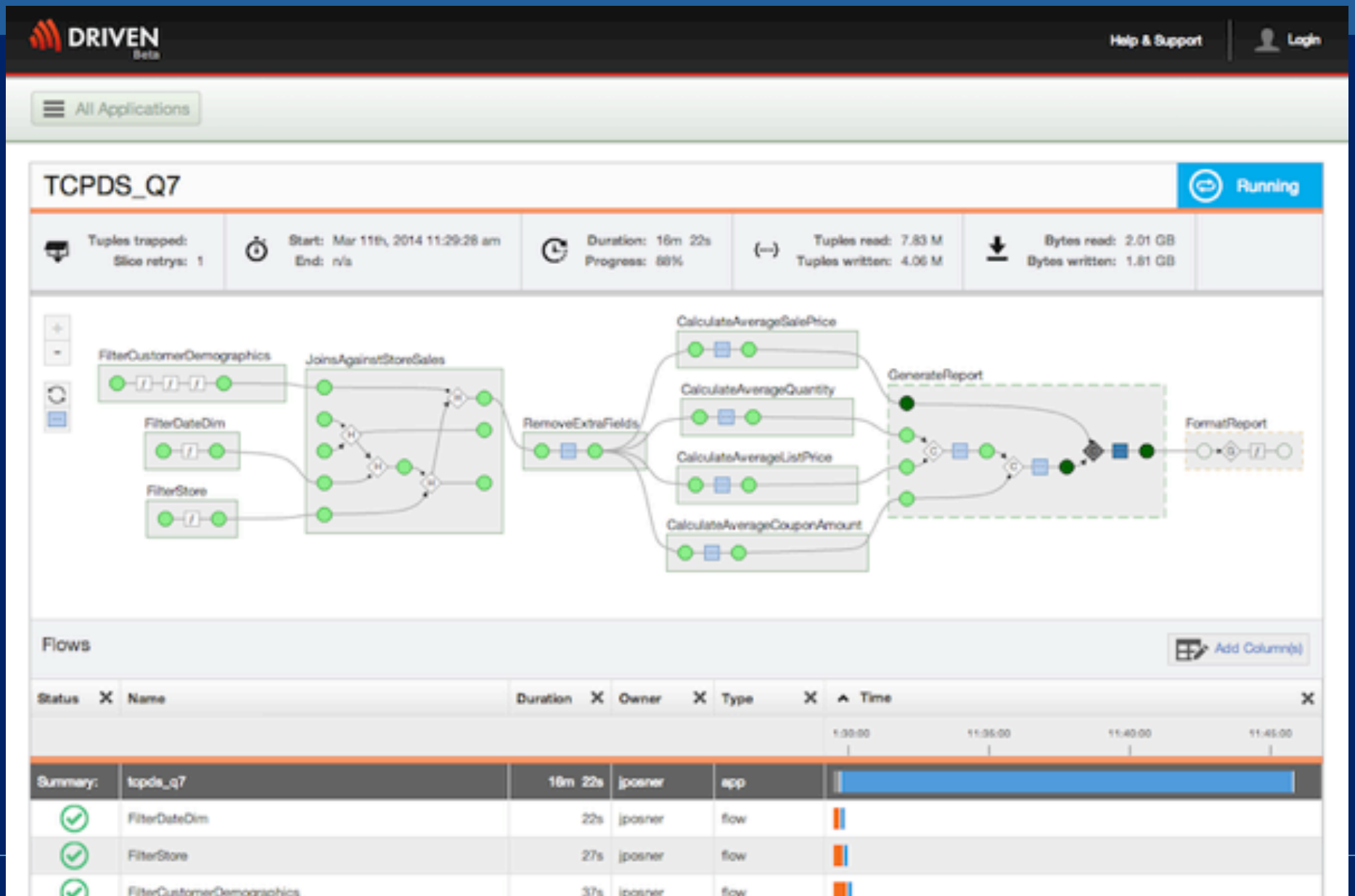
Latest News

FLUID 1.0

Develop, Debug and

Cascading

www.cascading.org



Cascading

www.cascading.org

DRIVEN Beta

Help & Support | Login

All Applications

INNERJOINS

Successful

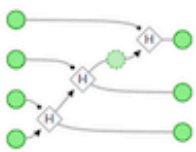
App: cascading-tpc-ds-100G-Q40 | Flow: InnerJoins

Start: Apr 23rd, 2014 10:21:25 pm
End: Apr 23rd, 2014 10:49:28 pm

Duration: 28m 2s
Progress:

Tuples read: 700.54 M
Tuples written: 540.31 M

Bytes read: 384.85 GB
Bytes written: 332.99 GB



InnerJoins : slice performance

Add Counters

Status	Step	Node	Graph	Slices	Run Time	ca.fl.SliceCounters Read_Duration	ca.fl.SliceCounters Tuples_Read	ca.fl.SliceCounters Tuples_Written	ca.fl.SliceCounters Write_Duration
				5087					
	(4/5) ...40/ItemToCatalogSales.dat 12m 37s	mappers_(4/5) ...40/ItemToCatalogSales.dat MAPPER 12m 12s		463	2.6m, 1.6m, 1m	36.27s, 7.68s, 0.88s, Σ: 36.5m	318.06 K, 312.33 K, 56.1 K, Σ: 144 M	298.15 K, 292.66 K, 52.55 K, Σ: 134.95 M	16.57s, 3.91s, 0.78s, Σ: 33.2m
	(2/5) 8m 9s	mappers_(2/5) MAPPER 7m 35s		463	3.3m, 1.4m	1m, 0.79s, Σ: 53m	318.06 K, 56.1 K, Σ: 144 M	296.65 K, 52.29 K, Σ: 134.28 M	1.3m, 7.38s, Σ: 4.9h

Cascading

www.cascading.org

DRIVEN Beta

Help & Support | Login

All Applications

INNERJOINS

Successful

App: cascading-tpc-ds-100G-Q40 | Flow: InnerJoins

Start: Apr 23rd, 2014 10:21:25 pm
End: Apr 23rd, 2014 10:49:28 pm

Duration: 28m 2s
Progress:

Tuples read: 700.54 M
Tuples written: 540.31 M

Bytes read: 384.85 GB
Bytes written: 332.99 GB

DAG

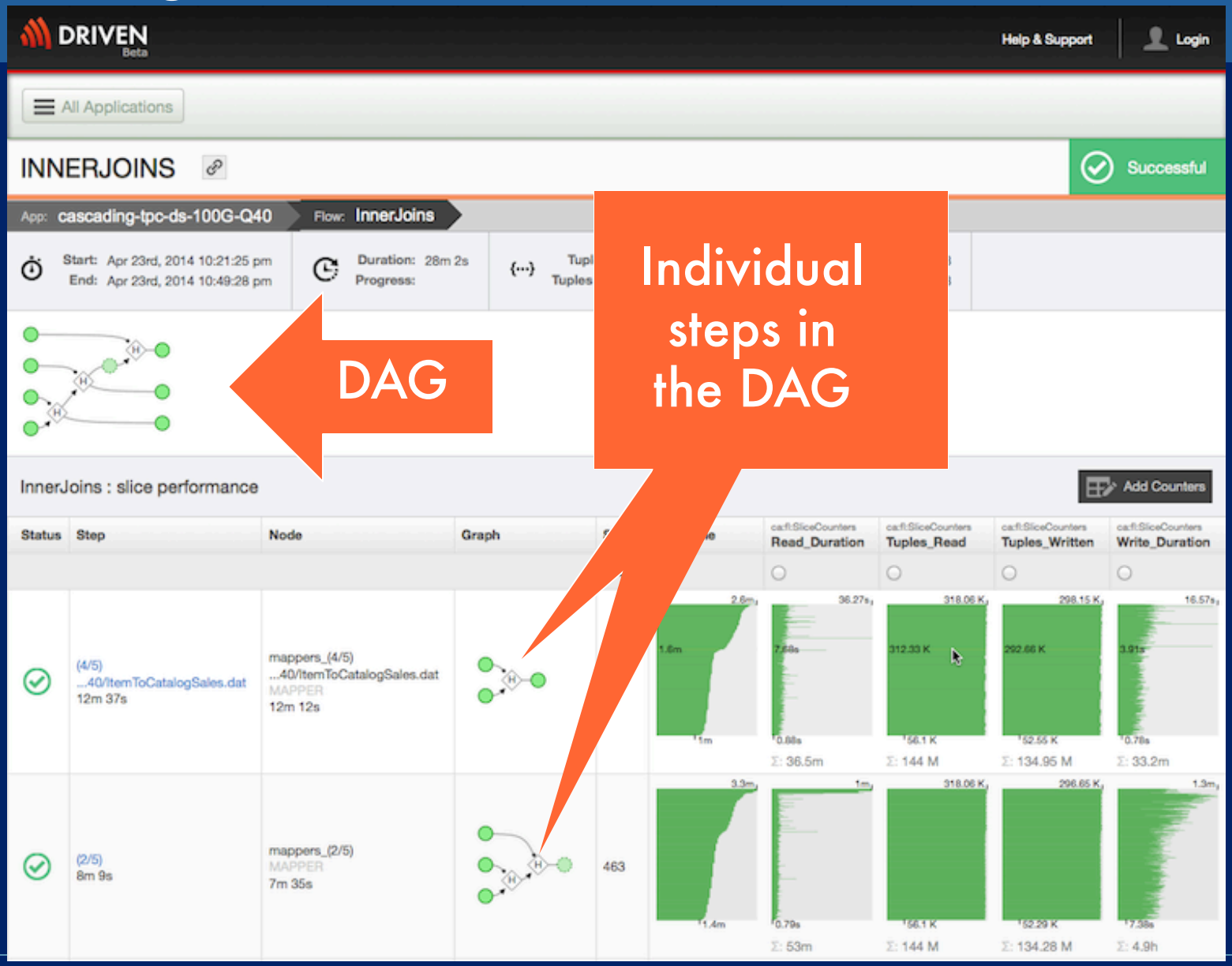
InnerJoins : slice performance

Add Counters

Status	Step	Node	Graph	Slices	Run Time	ca.fl.SliceCounters Read_Duration	ca.fl.SliceCounters Tuples_Read	ca.fl.SliceCounters Tuples_Written	ca.fl.SliceCounters Write_Duration
				5087					
	(4/5) ...40/ItemToCatalogSales.dat 12m 37s	mappers_(4/5) ...40/ItemToCatalogSales.dat MAPPER 12m 12s		463	2.6m, 1.6m, 1m	36.27s, 7.68s, 0.88s, Σ: 36.5m	318.06 K, 312.33 K, 56.1 K, Σ: 144 M	298.15 K, 292.66 K, 52.55 K, Σ: 134.95 M	16.57s, 3.91s, 0.78s, Σ: 33.2m
	(2/5) 8m 9s	mappers_(2/5) MAPPER 7m 35s		463	3.3m, 1.4m	1m, 0.79s, Σ: 53m	318.06 K, 56.1 K, Σ: 144 M	296.65 K, 52.29 K, Σ: 134.28 M	1.3m, 7.38s, Σ: 4.9h

Cascading

www.cascading.org



Cascading

www.cascading.org

DRIVEN Beta

Help & Support | Login

All Applications

INNERJOINS

Successful

App: cascading-tpc-ds-100G-Q40 | Flow: InnerJoins

Start: Apr 23rd, 2014 10:21:25 pm
End: Apr 23rd, 2014 10:49:28 pm

Duration: 28m 2s
Progress:

Tuples

DAG

Graphs of activity & performance metrics

InnerJoins : slice performance

Add Counters

Status	Step	Node	Graph	Slices	Run T	Read_Duration	Tuples_Read	Tuples_Written	Write_Duration
				5087					
	(4/5) ...40/ItemToCatalogSales.dat 12m 37s	mappers_(4/5) ...40/ItemToCatalogSales.dat MAPPER 12m 12s		463	2.6m, 1.6m, 1m	36.27s, 7.68s, 0.88s, Σ: 36.5m	318.06 K, 312.33 K, 56.1 K, Σ: 144 M	298.15 K, 292.66 K, 52.55 K, Σ: 134.95 M	16.57s, 3.91s, 0.78s, Σ: 33.2m
	(2/5) 8m 9s	mappers_(2/5) MAPPER 7m 35s		463	3.3m, 1.4m	1m, 0.79s, Σ: 53m	318.06 K, 56.1 K, Σ: 144 M	296.65 K, 52.29 K, Σ: 134.28 M	1.3m, 7.38s, Σ: 4.9h

Cascading

www.cascading.org

DRIVEN Beta

Help & Support | Login

All Applications

INNERJOINS

Successful

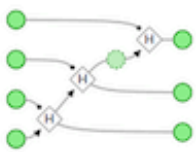
App: cascading-tpc-ds-100G-Q40 | Flow: InnerJoins

Start: Apr 23rd, 2014 10:21:25 pm
End: Apr 23rd, 2014 10:49:28 pm

Duration: 28m 2s
Progress:

Tuples read: 700.54 M
Tuples written: 540.31 M

Bytes read: 384.85 GB
Bytes written: 332.99 GB

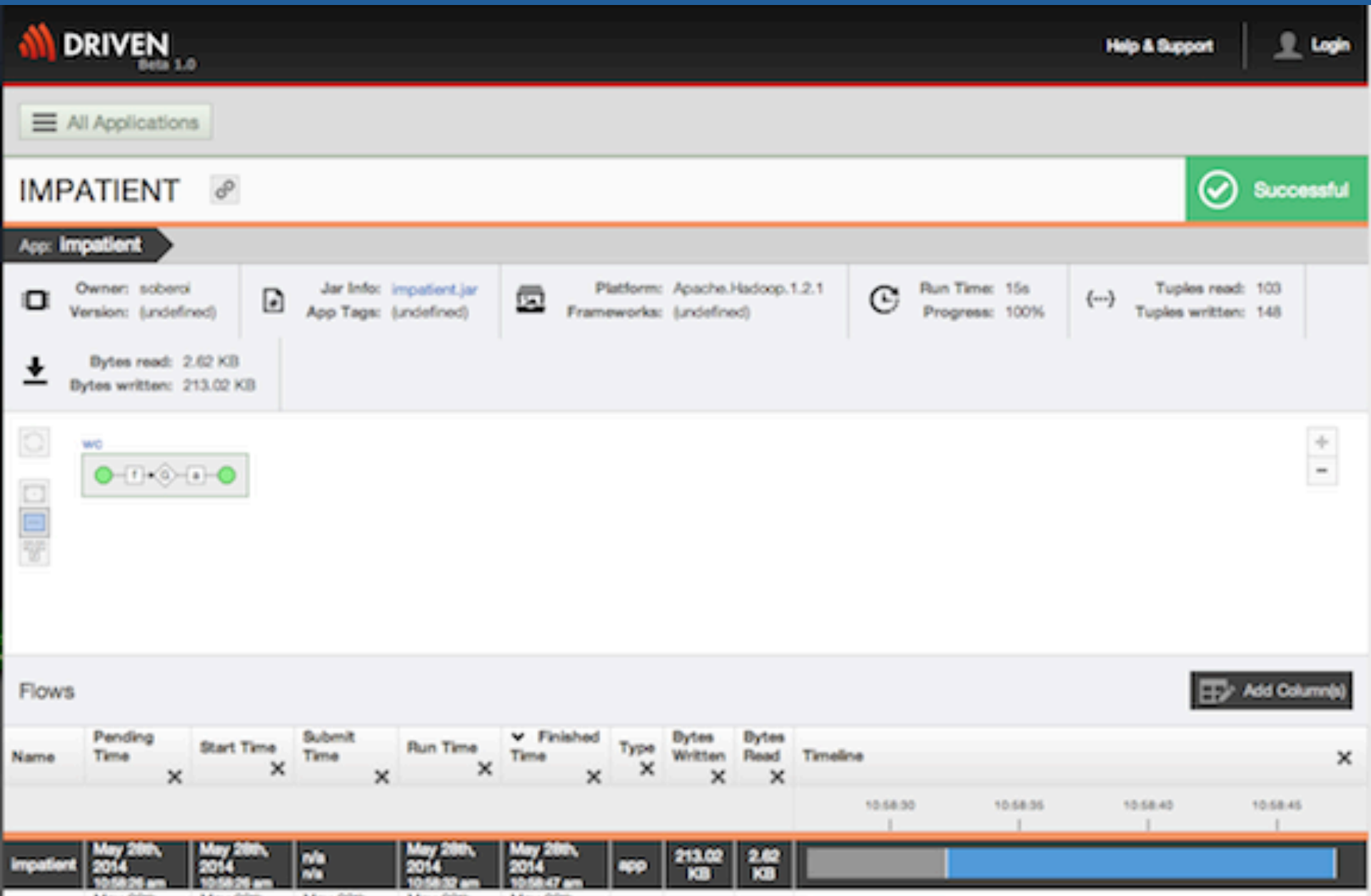


InnerJoins : slice performance

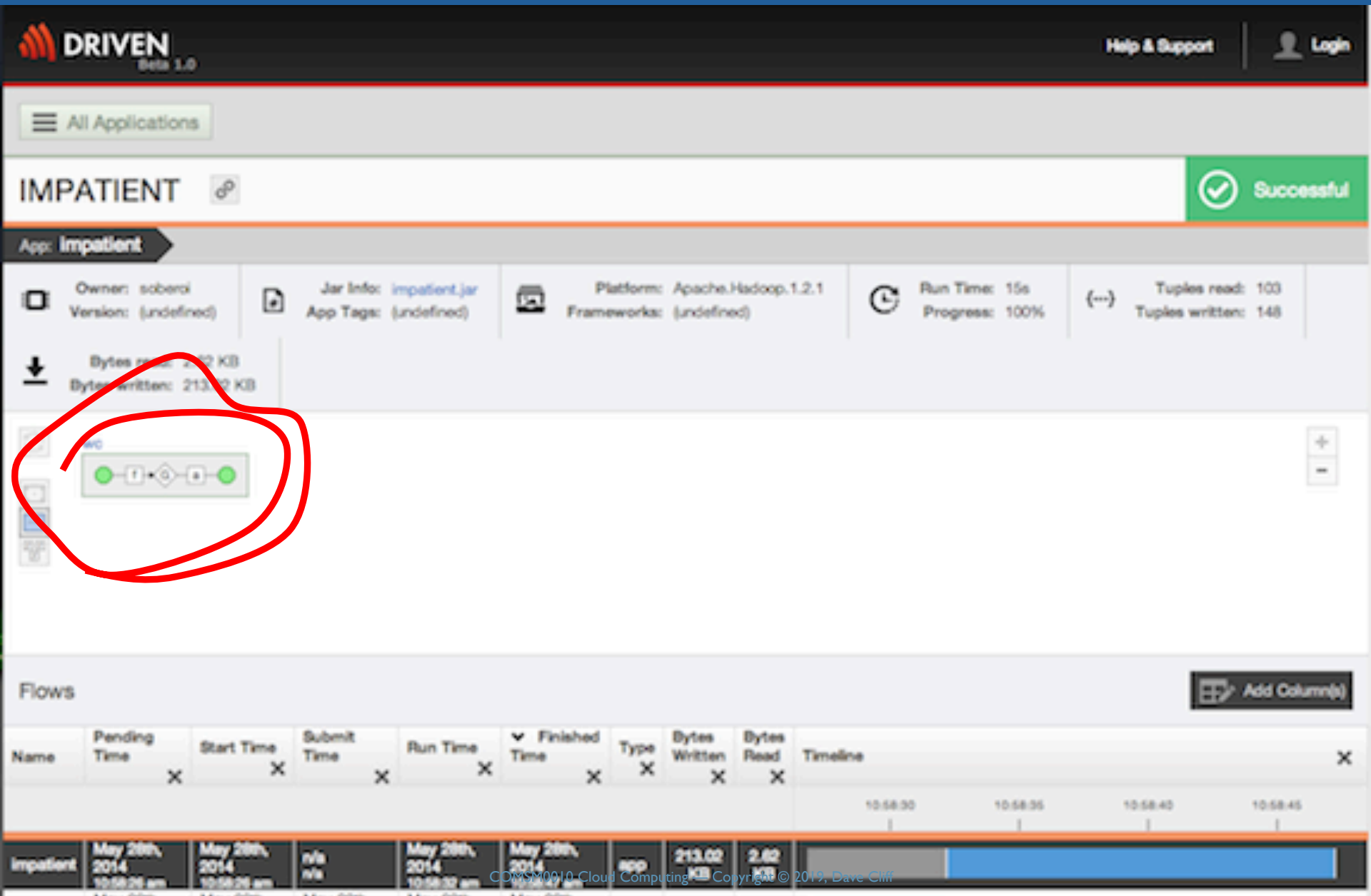
Add Counters

Status	Step	Node	Graph	Slices	Run Time	ca.fl.SliceCounters Read_Duration	ca.fl.SliceCounters Tuples_Read	ca.fl.SliceCounters Tuples_Written	ca.fl.SliceCounters Write_Duration
				5087					
	(4/5) ...40/ItemToCatalogSales.dat 12m 37s	mappers_(4/5) ...40/ItemToCatalogSales.dat MAPPER 12m 12s		463	2.6m, 1.6m, 1m	36.27s, 7.68s, 0.88s, Σ: 36.5m	318.06 K, 312.33 K, 56.1 K, Σ: 144 M	298.15 K, 292.66 K, 52.55 K, Σ: 134.95 M	16.57s, 3.91s, 0.78s, Σ: 33.2m
	(2/5) 8m 9s	mappers_(2/5) MAPPER 7m 35s		463	3.3m, 1.4m	1m, 0.79s, Σ: 53m	318.06 K, 56.1 K, Σ: 144 M	296.65 K, 52.29 K, Σ: 134.28 M	1.3m, 7.38s, Σ: 4.9h

http://docs.cascading.org/impatient/impatient2.html

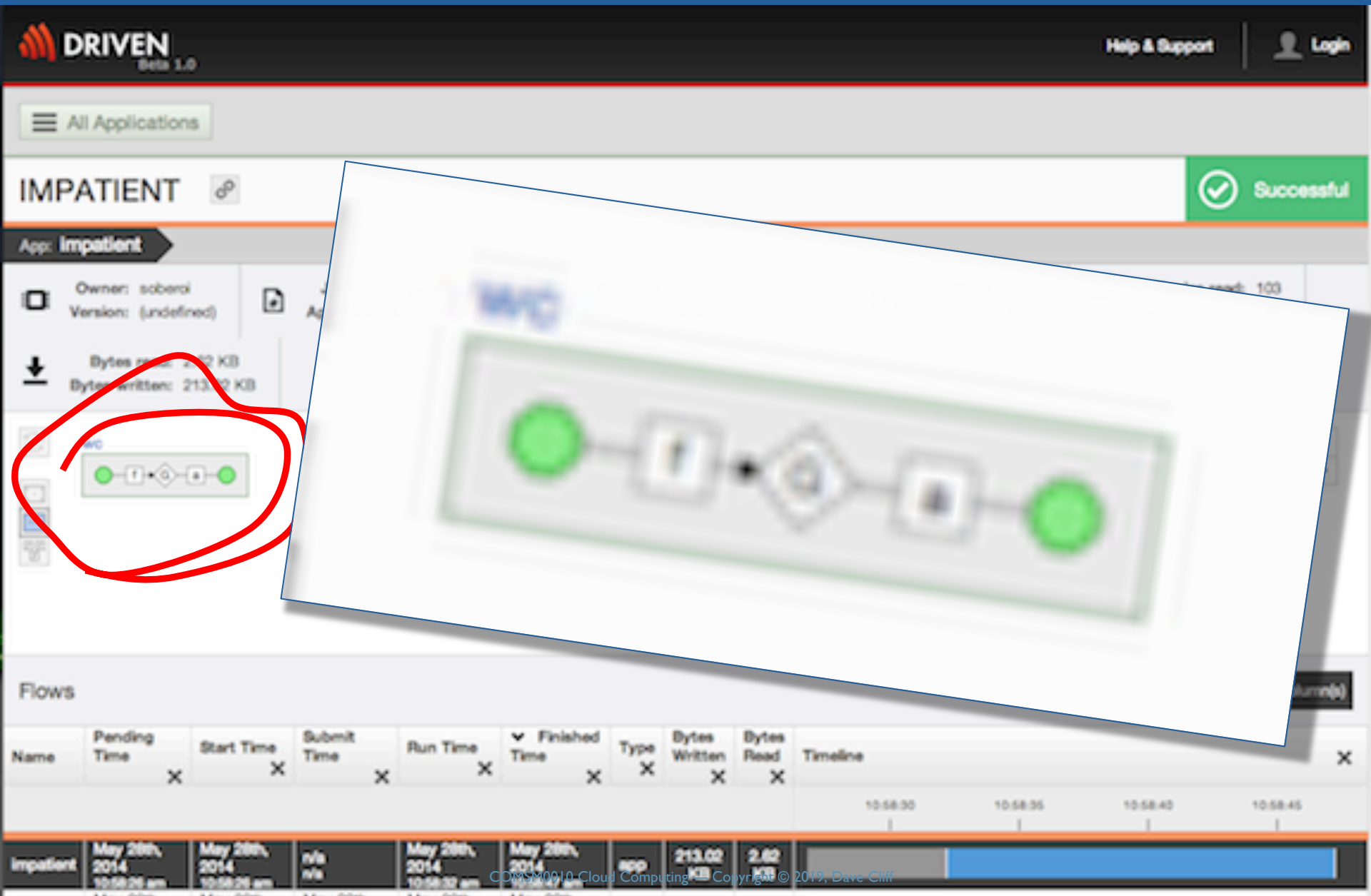


http://docs.cascading.org/impatient/impatient2.html



 Copyright © 2019, Dave Cliff

http://docs.cascading.org/impatient/impatient2.html



 Copyright © 2019, Dave Cliff

And finally…

Now do all of this yourself

Hadoop/Hive/Pig/Hbase DIY

Buy a few commodity servers,
Download & install Hadoop2,

And away you go…

There is
another way

There is another way

EMR

Elastic Map Reduce

https://aws.amazon.com/emr/

Contact Sales Support English My Account Sign In to the Console

Products Solutions Pricing Learn Partner Network AWS Marketplace Explore More

Amazon EMR Overview Features EMR Workloads Pricing Getting Started Resources FAQs Partners

Amazon EMR

Easily Run and Scale Apache Spark, Hadoop, HBase, Presto, Hive, and other Big Data Frameworks

Get started with Amazon EMR

Request support for your evaluation

Amazon EMR provides a managed Hadoop framework that makes it easy, fast, and cost-effective to process vast amounts of data across dynamically scalable Amazon EC2 instances. You can also run other popular distributed frameworks such as Apache Spark, HBase, Presto, and Flink in EMR, and interact with data in other AWS data stores such as Amazon S3 and Amazon DynamoDB. EMR Notebooks, based on the popular Jupyter Notebook, provide a development and collaboration environment for ad hoc querying and exploratory analysis.

Intro to Amazon EMR (3:06)

EMR securely and reliably handles a broad set of big data use cases, including log analysis,

https://cloud.google.com/free/docs/map-aws-google-cloud-platform

Google Cloud Platform Free Tier
- Program Overview
- GCP Free Tier
- Frequently Asked Questions
- Always Free Usage Limits
- Evaluate Google Cloud Platform
  - What makes Google Cloud Platform different?
  - Map AWS services to Google Cloud Platform products
  - Map Microsoft Azure services to Google Cloud Platform products
  - Estimate and compare costs
  - Measure and compare performance
  - Migrate data to and from Google Cloud Platform

Google Cloud Platform Free Tier

Map AWS services to Google Cloud Platform products

SEND FEEDBACK

Contents

What's next

If you are familiar with Amazon Web Services (AWS), a quick way to understand what the various Google Cloud Platform (GCP) services do is to map them to AWS services that offer similar functionality. The following table provides a high-level mapping of the services provided by the two platforms. For a detailed comparison of GCP and AWS services by service type, see Google Cloud Platform for AWS Professionals.

Service Category	Service	AWS	Google Cloud Platform
Compute	IaaS	Amazon Elastic Compute Cloud	Compute Engine
	PaaS	AWS Elastic Beanstalk	App Engine
	Containers	Amazon Elastic Container Service	Google Kubernetes Engine
	Serverless Functions	AWS Lambda	Cloud Functions

https://cloud.google.com/dataproc/

Google Cloud | Why Google | Products | Solutions | Pricing | Security | Documentation | Customers | P | Console

Data Analytics Products

PC_tower.jpg

Contact sales | Try free

CLOUD DATAPROC

A faster, easier, more cost-effective way to run Apache Spark and Apache Hadoop

TRY IT FREE

Cloud-native Apache Hadoop & Apache Spark

Cloud Dataproc is a fast, easy-to-use, fully-managed cloud service for running **Apache Spark** and **Apache Hadoop** clusters in a simpler, more cost-efficient way. Operations that used to take hours or days take seconds or minutes instead, and you pay only for the resources you use (with per-second billing). Cloud Dataproc also easily integrates

https://azure.microsoft.com/en-gb/services/hdinsight

Microsoft Azure

Contact Sales: 0800-026-0797 | Search | My account | Portal | Dave

Overview | Solutions | Products | Documentation | Pricing | Training | Marketplace | Partners | Support | Blog | More | Free account

HDInsight

Easy, cost-effective, enterprise-grade service for open source analytics

Start for free

Explore HDInsight: Pricing details | Documentation | Updates | Partners

Announcement | Azure HDInsight brings next generation Apache Hadoop 3.0 and enterprise security to the cloud

What are the benefits of HDInsight?

Easily run popular open source frameworks – including Apache Hadoop, Spark and Kafka – using Azure HDInsight, a cost-effective, enterprise-grade service for open source analytics. Effortlessly process massive amounts of data and get all the benefits of the broad open source ecosystem with the global scale of Azure.

- Quickly spin up big data clusters on demand, scale them up or down based on your
- Use HDInsight tools to easily get started in your favourite development

https://cloud.oracle.com/bigdata

ORACLE Cloud

Sign In | Contact | Chat | English | Estimate | Buy | Try for Free

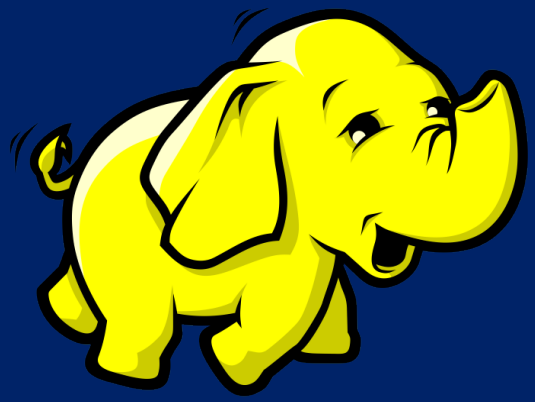
Applications | Platform | Infrastructure | Resources

Platform > Data Management > Big Data

Big Data

Overview | Features | Pricing | Learn More

Delivering Hadoop, Spark and Data Science with Oracle Security and Cloud Simplicity

The power of Hadoop and Spark delivered as a secure, automated, high-performance service, which can be fully integrated with existing enterprise data in Oracle Database and Oracle

The End