Hortonworks

2015 HADOOP SUMMIT

Hadoop Crash Course

Hadoop Interest Group

Meetup

Jules Damji

jdamji@hortonworks.com

@2twitme

Rafael Coss

rafael@hortonworks.com

@racoss

Summer 2015

Version 1.0

Hadoop Crash Course

- ~~**Why Hadoop?**~~

- **Hadoop Ecosystem & Distribution**
- **Store Data (HDFS)**
- **Process Data in Hadoop 1 (MapReduce)**
- **Process Data in Hadoop 2 (Yarn + MapReduce/Tez)**
- **Data Access**

- **Lab**

Hortonworks®

Hadoop Ecosystem

Apache Ambari
http://incubator.apache.org/ambari
Cluster System Operations

ETL

Data Ingestion

SQL

Management

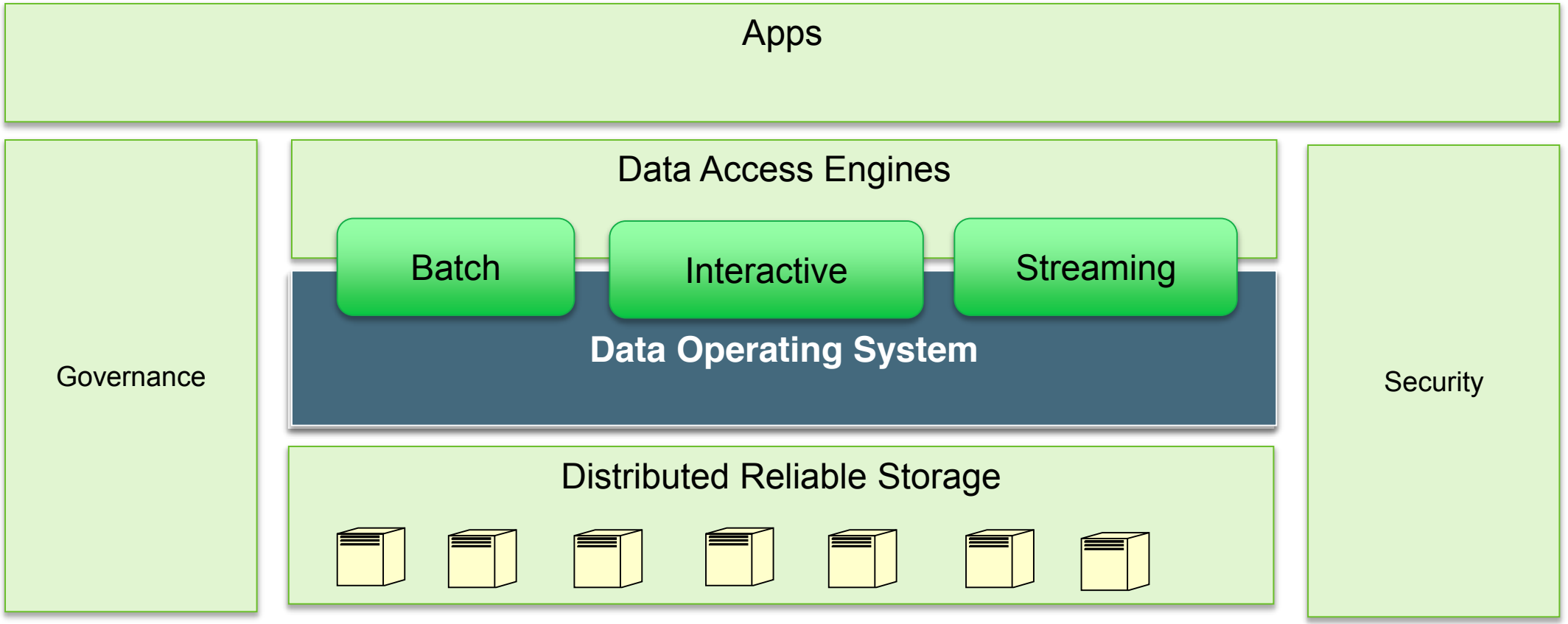
Management

SQL on HBase

Data Processing

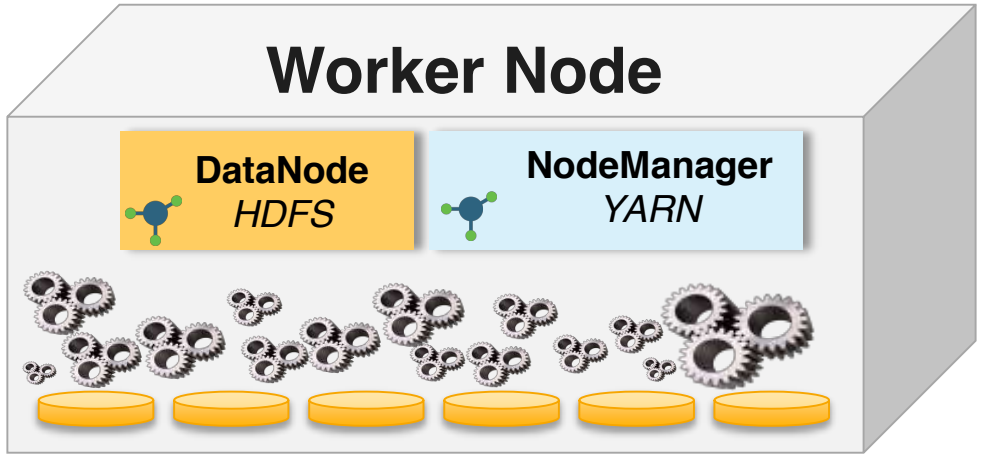
runs on

Hadoop Architecture



Hortonworks®

Core hadoop



Disk, CPU, Memory

+

NameNode
HDFS

`/directory/structure/in/memory.txt`

ResourceManager
YARN

Resource management + scheduling

Hadoop daemon

User application

Hortonworks®

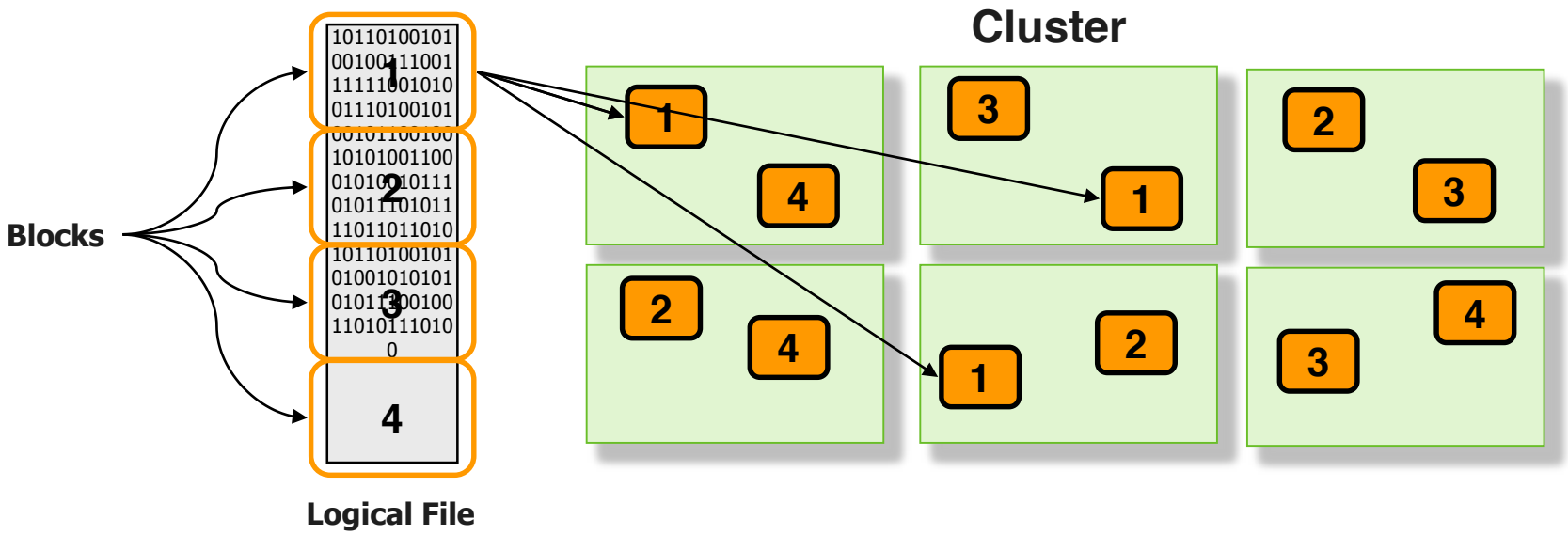
Joys of Real Hardware (Jeff Dean)

Typical first year for a new cluster:
~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc

Hadoop Distributed File System (HDFS)

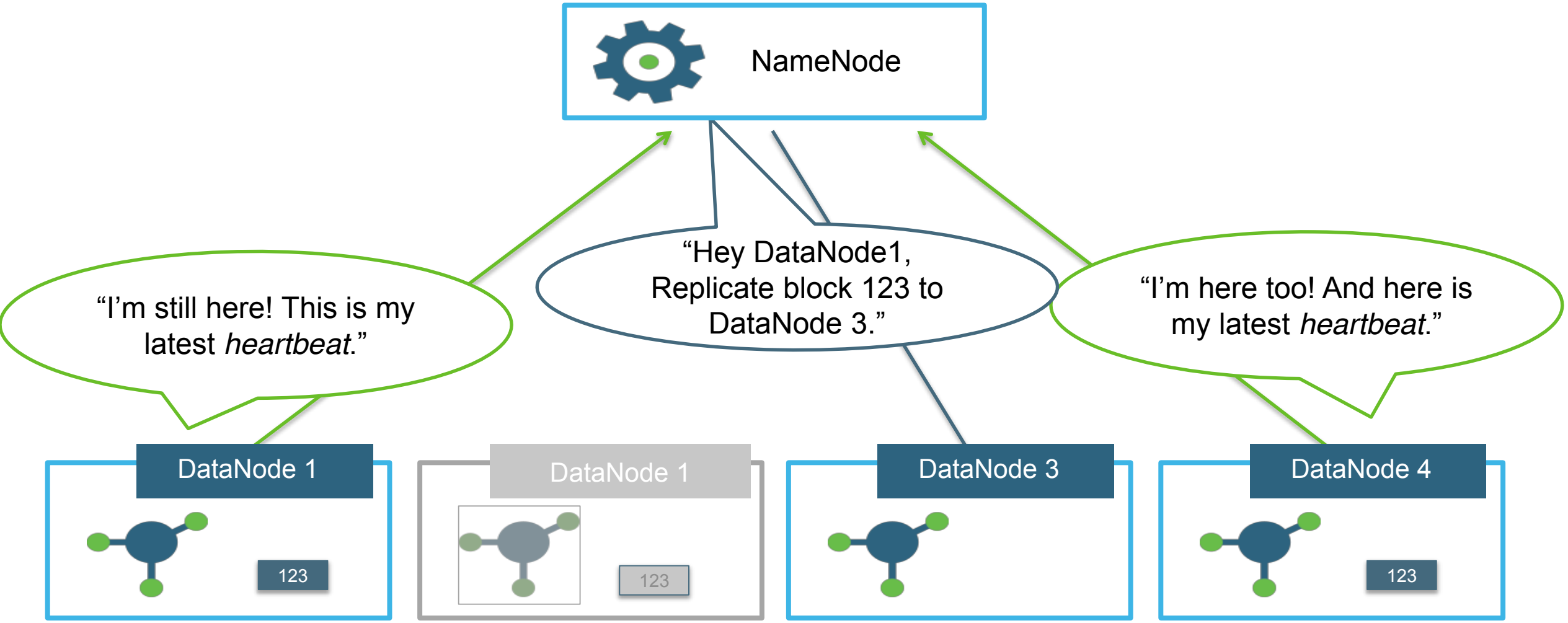
Fault Tolerant Distributed Storage

- Divide files into big blocks and distribute 3 copies **randomly** across the cluster
- Processing Data Locality
  - Not Just storage but computation



Hortonworks®

The DataNodes



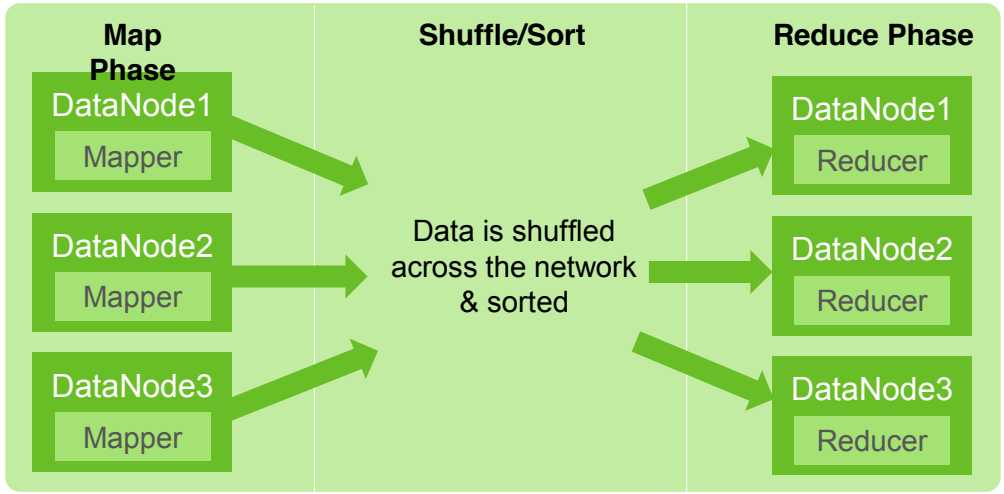
Batch Processing in Hadoop

MapReduce

Batch Access to Data

Original data access mechanism for Hadoop

- **Framework**
  Made for developing distributed applications to process vast amounts of data in-parallel on large clusters
- **Proven**
  Reliable interface to Hadoop which works from GB to PB. But, batch oriented – Speed is not it's strong point.
- **Ecosystem**
  Ported to Hadoop 2 to run on YARN. Supports original investments in Hadoop by customers and partner ecosystem.



MapReduce Job Lifecycle

Saying that MapReduce is dead is preposterous

- Would limits us to only new workloads
- ALL Hadoop clusters use map reduce
- Why rewrite everything immediately?

Hortonworks®

What is MapReduce?

Break a large problem into sub-solutions

Map

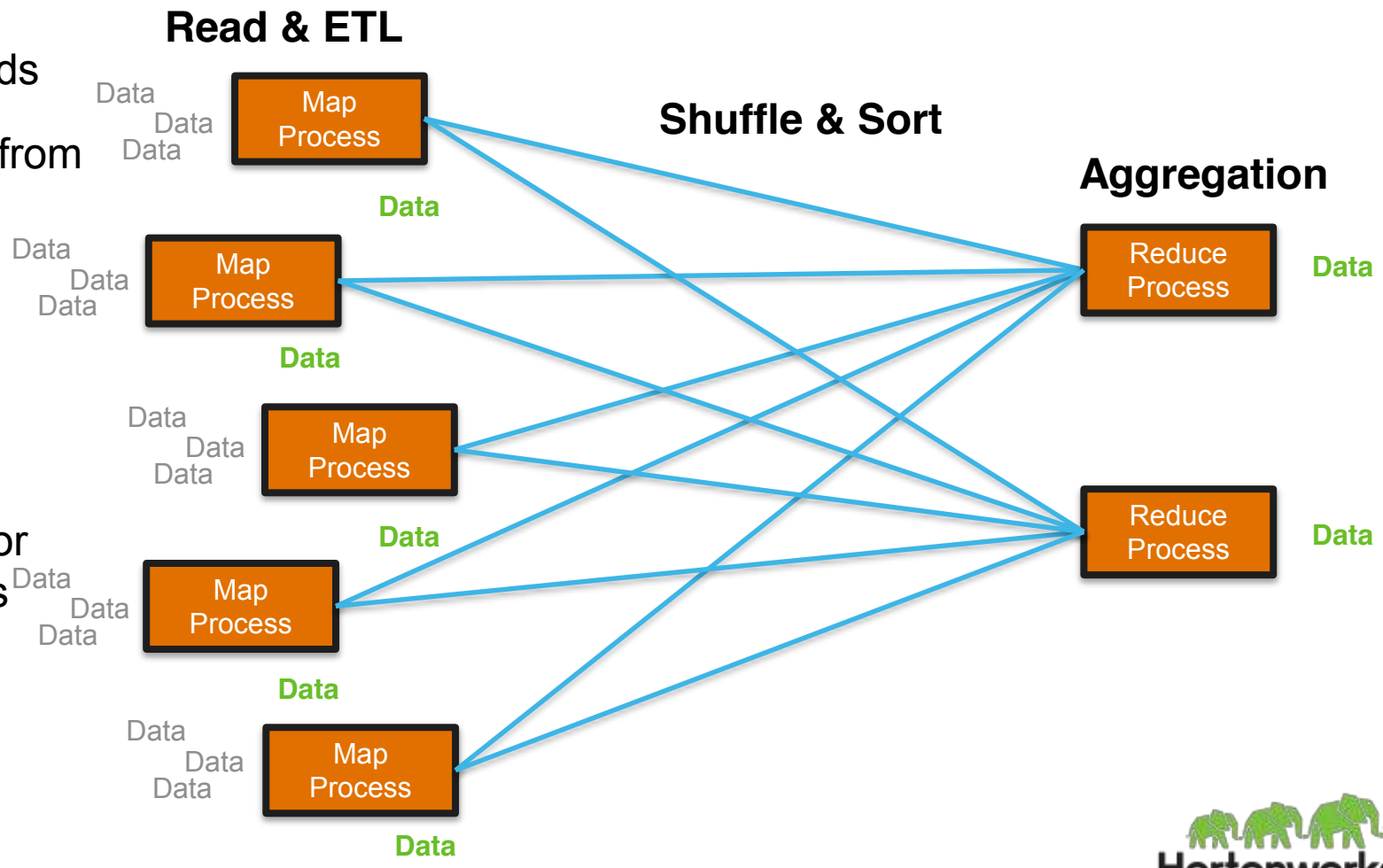
- Iterate over a large # of records
- Extract something of interest from each record

Shuffle

- Sort Intermediate results

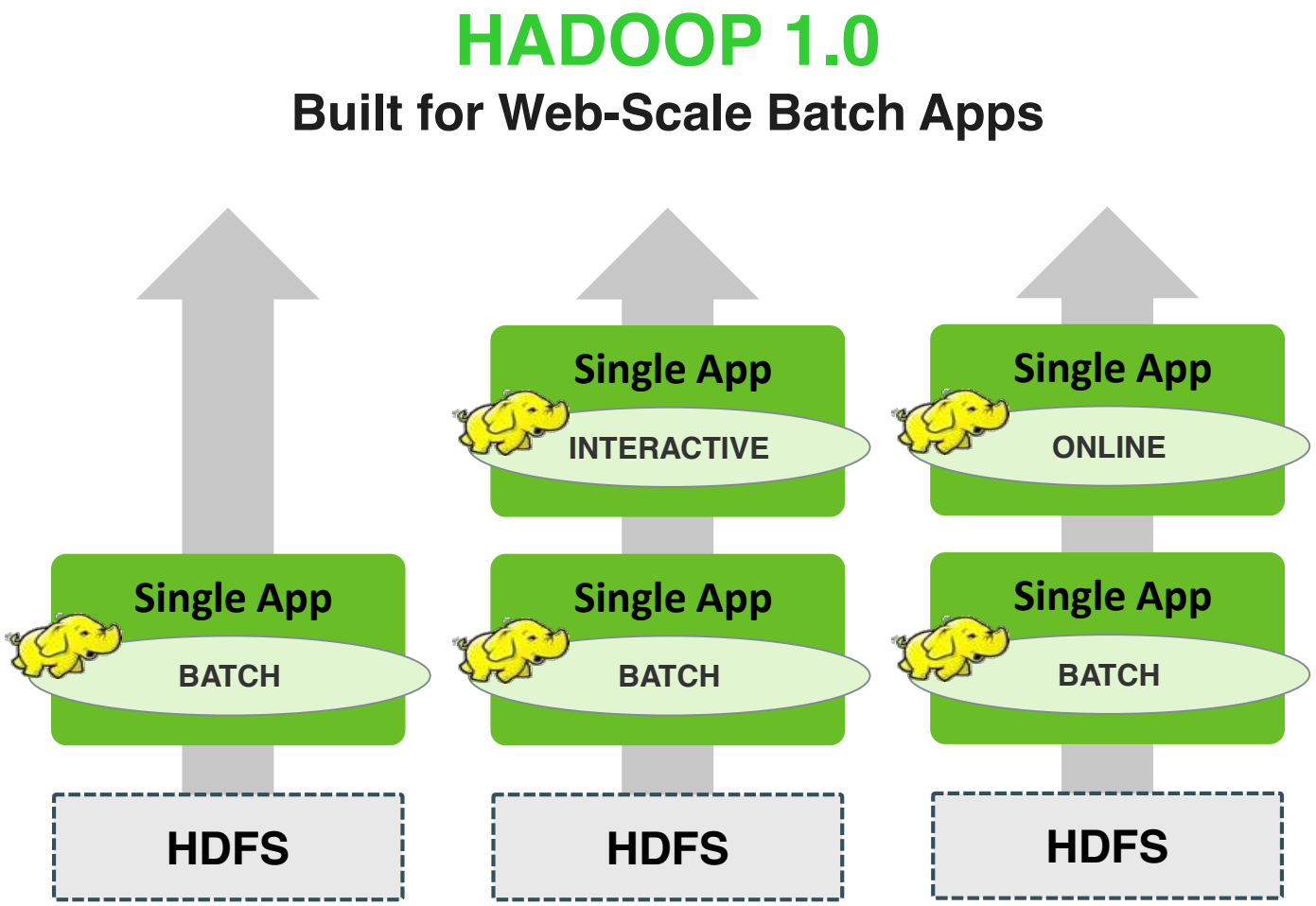
Reduce

- Aggregate, summarize, filter or transform intermediate results
- Generate final output



Hortonworks®

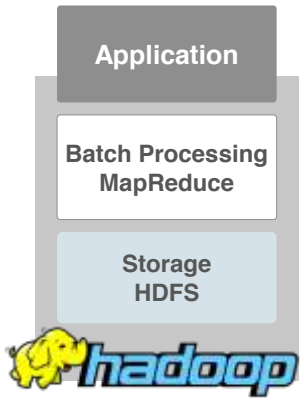
1st Gen Hadoop: Cost Effective Batch at Scale



Silos created for distinct use cases

Hortonworks®

Hadoop emerged as foundation of new data architecture



Apache Hadoop is an open source data platform for managing large volumes of high velocity and variety of data

- Built by Yahoo! to be the heartbeat of its ad & search business
- Donated to Apache Software Foundation in 2005 with rapid adoption by large web properties & early adopter enterprises
- Incredibly disruptive to current platform economics

Traditional Hadoop Advantages

- ✓ Manages new data paradigm
- ✓ Handles data at scale
- ✓ Cost effective
- ✓ Open source

Traditional Hadoop Had Limitations

- ✘ Batch-only architecture
- ✘ Single purpose clusters, specific data sets
- ✘ Difficult to integrate with existing investments
- ✘ Not enterprise-grade

Hortonworks®

What is Data Access?

Data Access defines *ALL* the channels through which data can be accessed, analyzed, cleansed and consumed within Hadoop. Each channel can be categorized into THREE core patterns; Batch, Interactive and Real-time.

Multiple engines provide optimized access to your mission critical data.

Hortonworks

Hadoop Beyond Batch with YARN

A shift from the old to the new…

Single Use Sysztem

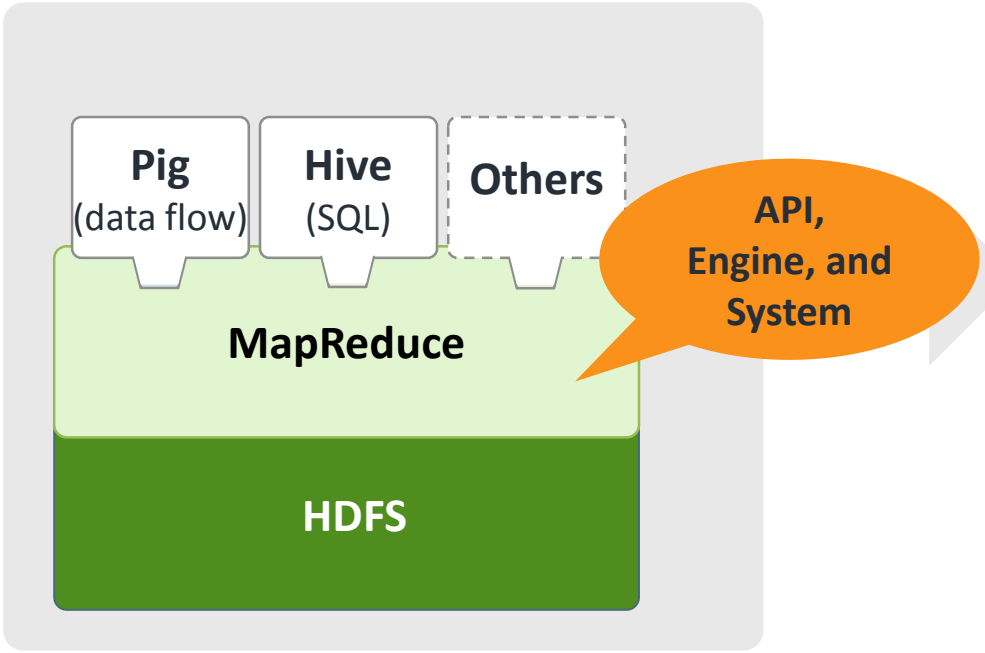
Batch Apps

Multi Use Data Platform

Batch, Interactive, Online, Streaming, …

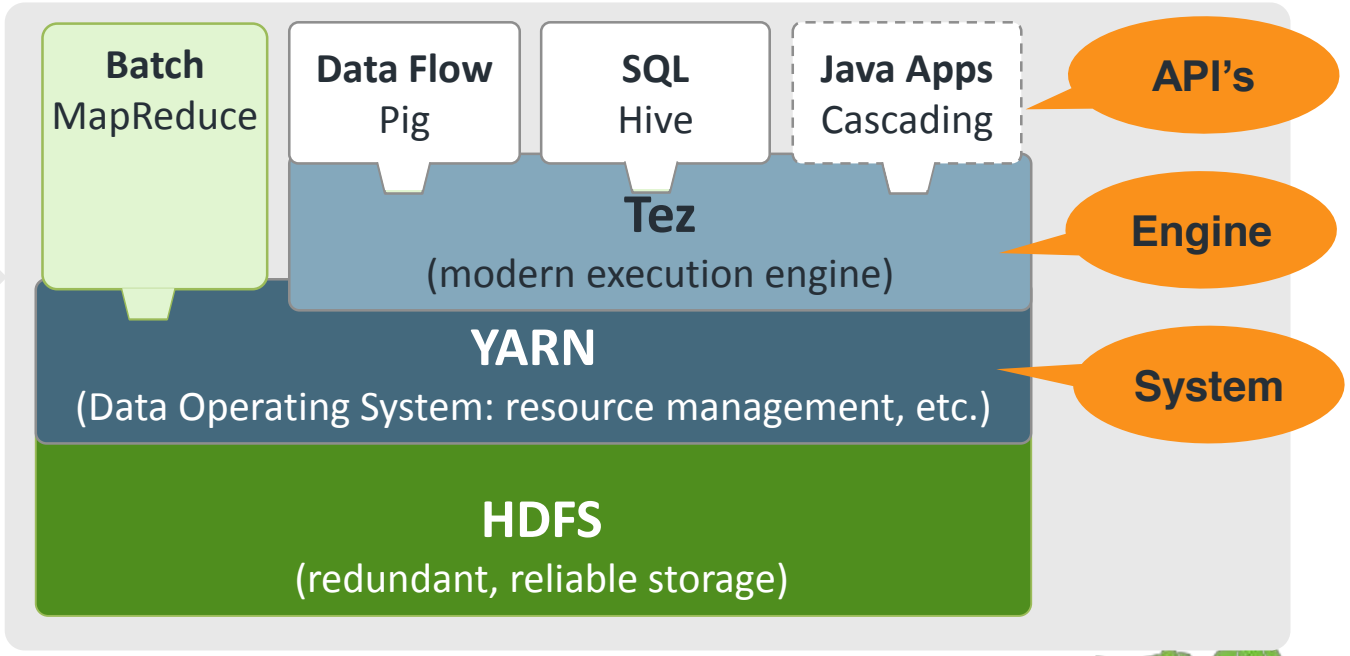
Hadoop 1

MapReduce as the Base



Hadoop 2

Apache Yarn as a Base



Hortonworks®

{Processing + Storage}

=

{MapReduce/YARN + HDFS}

=

{Core Hadoop}

Access patterns enabled by YARN

Batch

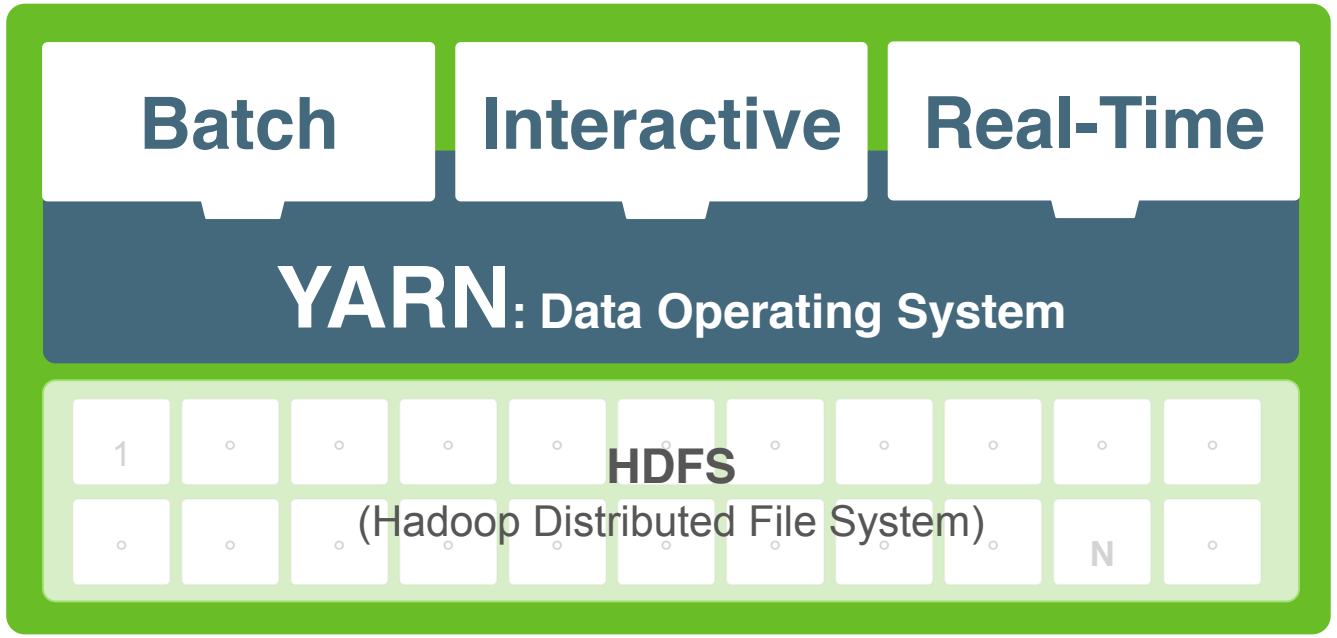
Needs to happen but, no timeframe limitations

Interactive

Needs to happen at Human time

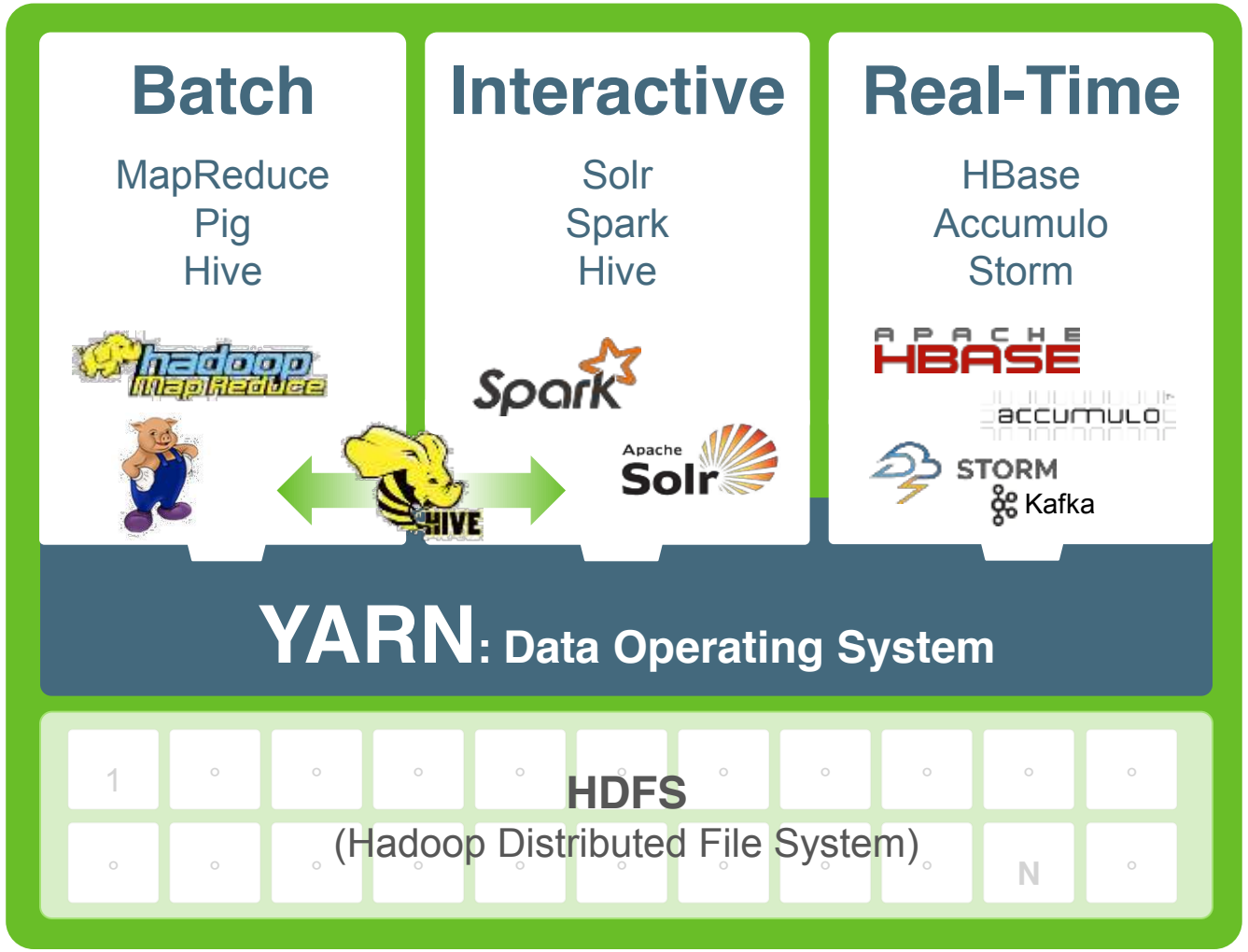
Real-Time

Needs to happen at Machine Execution time.



Apache Projects Enable Access Patterns

- **Various Open Source projects have incubated in order to meet these access pattern needs**
- **Today, they can all run on a single cluster on a Single set of data because of YARN!**
- **ALL powered by a BROAD Open Community**



Hortonworks®

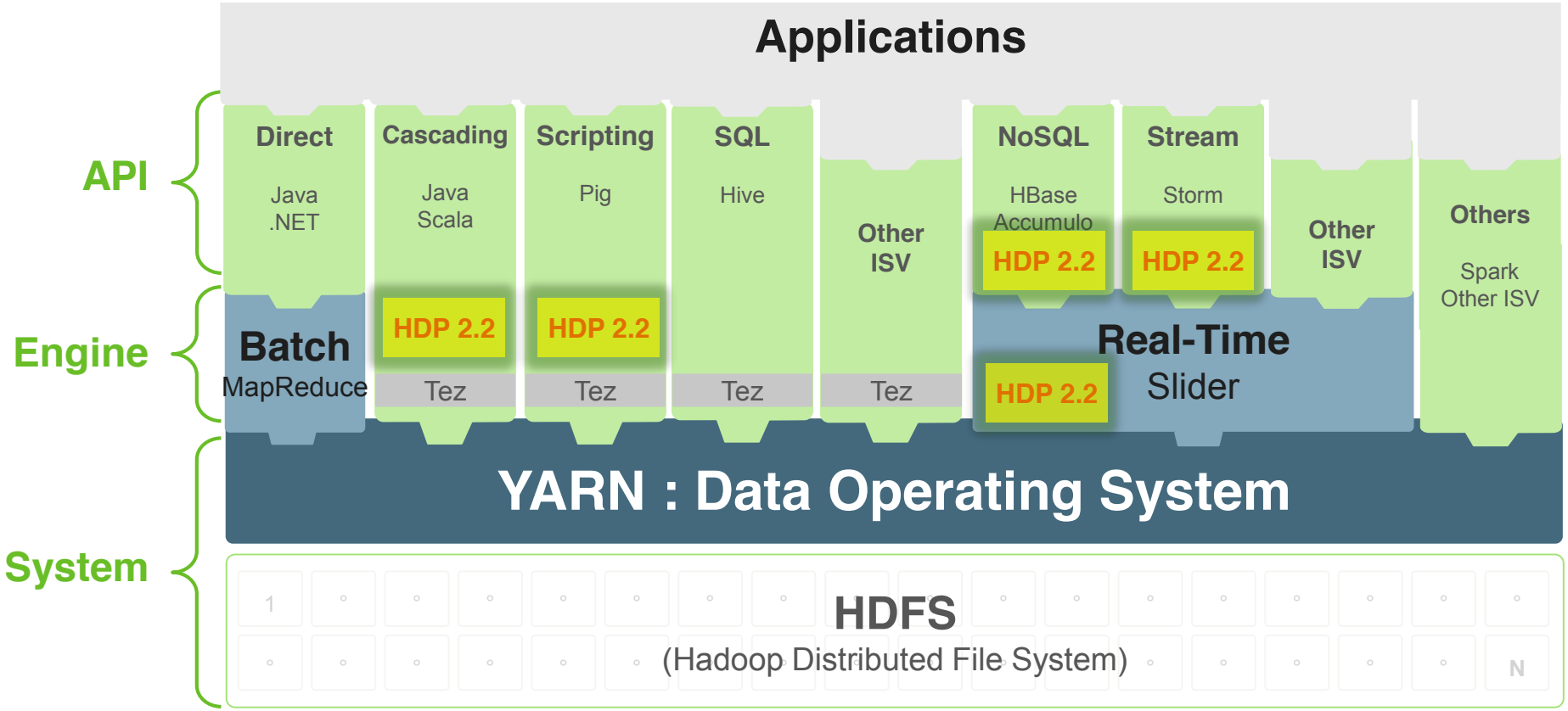
YARN: Resource Manager for Hadoop 2.0

Data Processing Engines Run Natively IN Hadoop

Flexible
Enables other purpose-built data processing models beyond MapReduce (batch), such as interactive and streaming

Efficient
Double processing **IN** Hadoop on the same hardware while providing predictable performance & quality of service

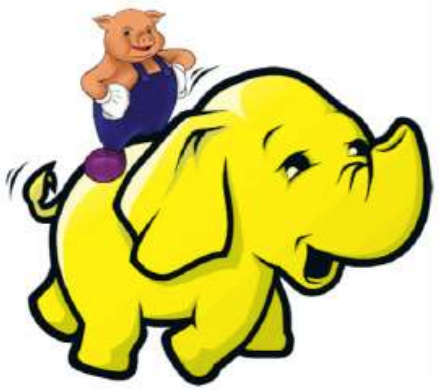
Shared
Provides a stable, reliable, secure foundation and shared operational services across multiple workloads



Scripting Data Flow & ETL

Apache Pig

- Data flow engine and scripting language (Pig Latin)
- Allows you to **transform** data and datasets



Advantages over MapReduce

- Reduces time to write jobs
- Community support
- Piggybank has a significant number of UDF's to help adoption
- There are a large number of existing shops using PIG

Hortonworks®

Why use Pig?

- Maybe we want to join two datasets, from different sources, on a common value, and want to filter, and sort, and get top 5 sites

```
1  users = LOAD 'input/users' USING PigStorage(',')
2          AS (name:chararray, age:int);
3
4  filtrd = FILTER users BY age >= 18 and age <= 25;
5
6  pages = LOAD 'input/pages' USING PigStorage(',')
7          AS (user:chararray, url:chararray);
8
9  jnd = JOIN filtrd BY name, pages BY user;
10
11 grpd = GROUP jnd BY url;
12
13 smmd = FOREACH grpd GENERATE group, COUNT(jnd) AS clicks;
14
15 srtd = ORDER smmd BY clicks DESC;
16
17 top5 = LIMIT srtd 5;
18
19 STORE Top5 INTO 'output/top5sites' USING PigStorage(',');
```

In Map Reduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see
            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top  100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

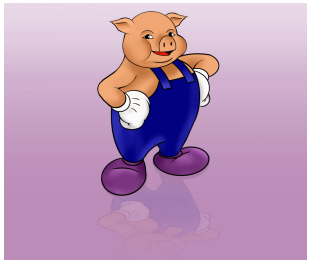
Pig Latin

- Pig executes in a unique fashion:
  - During execution, each statement is processed by the Pig interpreter
  - If a statement is valid, it gets added to a **logical plan** built by the interpreter
  - The steps in the logical plan do not actually execute until a DUMP or STORE command is used

How It Works

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```



pig.jar:

- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan
Map:
Filter
Count

Combine/Reduce:
Sum

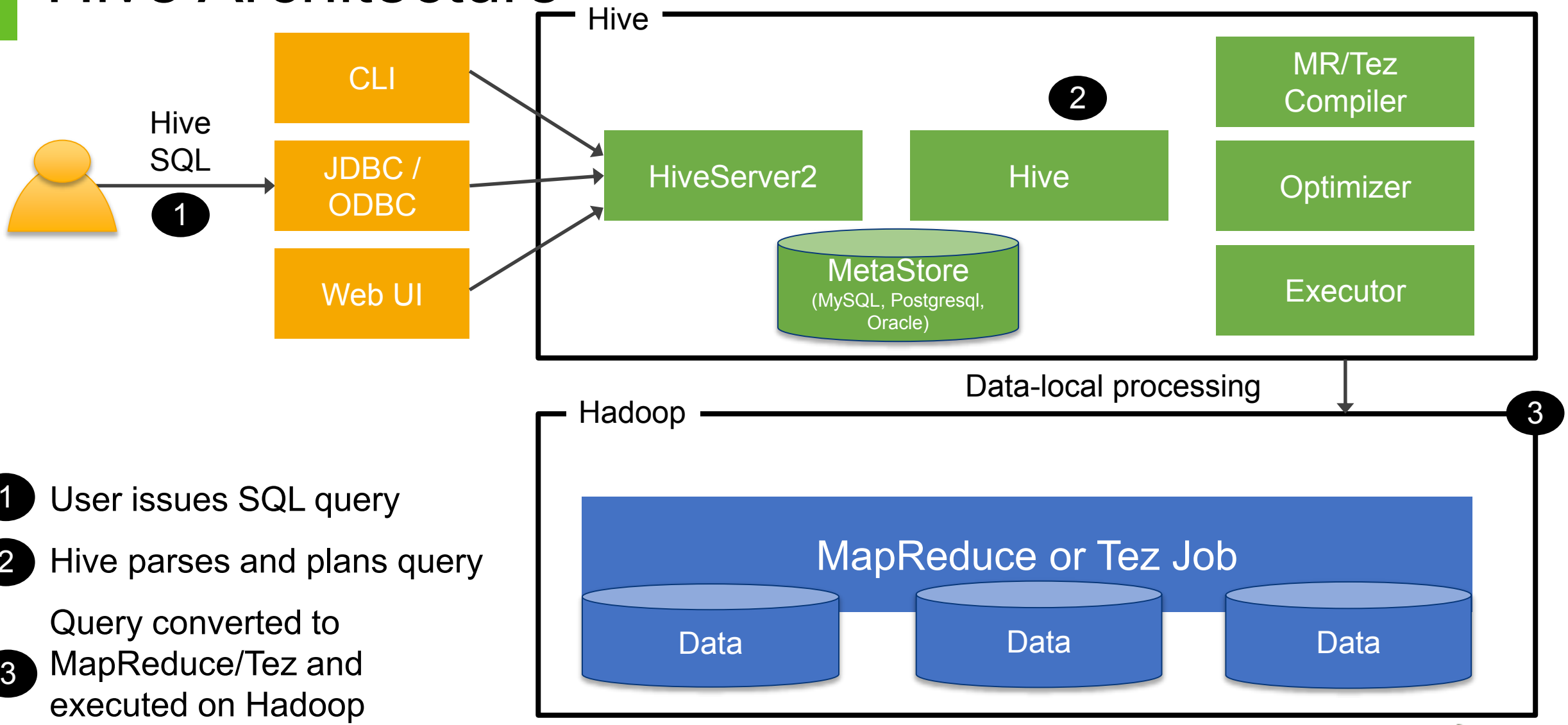
Apache Hive: THE defacto standard for SQL in Hadoop

- What?
  - Treat your data in Hadoop as tables
  - Provides a standard SQL 92 interface to data in Hadoop

- Why?
  - Shipped in every distribution… you already have it (although some do not ship complete versions) Quickly find value in raw data files
  - Proven at petabyte scale for both batch and interactive queries
  - Compatible with ALL major BI tools such as Tableau, Excel, MicroStrategy, Business Objects, etc…

Hortonworks®

Hive Architecture



1. User issues SQL query
2. Hive parses and plans query
3. Query converted to MapReduce/Tez and executed on Hadoop

Hortonworks

SQL Compliance

Evolution of SQL Compliance in Hive

SQL Datatypes

- INT/TINYINT/SMALLINT/BIGINT
- FLOAT/DOUBLE
- BOOLEAN
- ARRAY, MAP, STRUCT, UNION
- STRING
- BINARY
- TIMESTAMP
- DECIMAL
- DATE
- VARCHAR
- CHAR
- Interval Types

SQL Semantics

- SELECT, INSERT
- GROUP BY, ORDER BY, HAVING
- JOIN on explicit join key
- Inner, outer, cross and semi joins
- Sub-queries in the FROM clause
- ROLLUP and CUBE
- UNION
- Standard aggregations (sum, avg, etc.)
- Custom Java UDFs
- Windowing functions (OVER, RANK, etc.)
- Advanced UDFs (ngram, XPath, URL)
- Sub-queries for IN/NOT IN, HAVING
- JOINs in WHERE Clause
- INSERT/UPDATE/DELETE

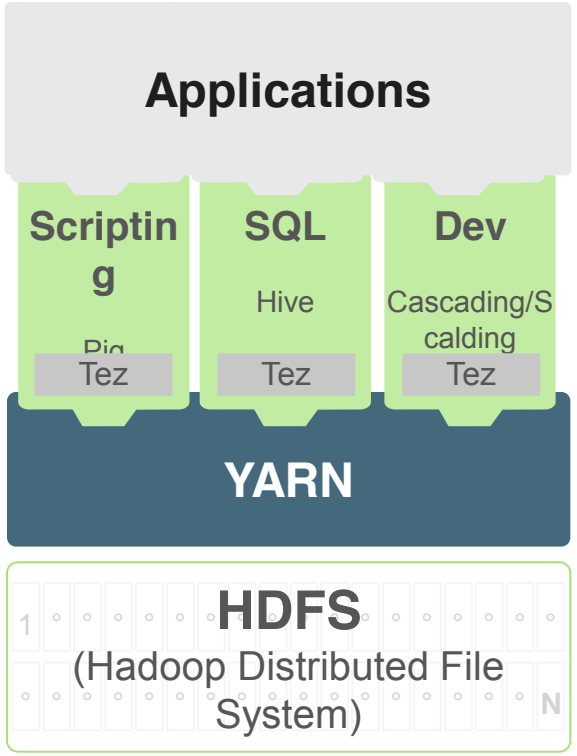
Legend

- Hive 10 or earlier
- Hive 11
- Hive 12
- Hive 13
- Roadmap

Hortonworks®

Why is Tez Important?

Apache Tez is a critical innovation of the Stinger Initiative.



- **Along with YARN, Tez not only improves Hive, but** improves all things batch *and* interactive for Hadoop; Pig, Cascading…
- **More Efficient Processing than MapReduce**

- Reduce operations and complexity of back end processing
- Allows for Map Reduce Reduce which saves hard disk operations
- Implements a “service” which is always on, decreasing start times of jobs
- Allows Caching of Data in Memory

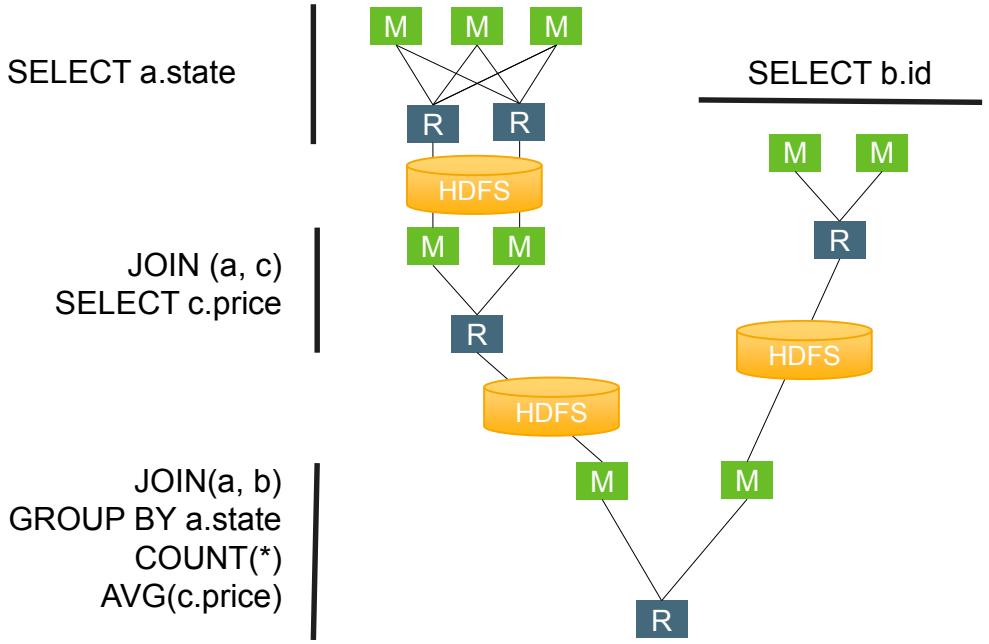
Hortonworks®

Tez

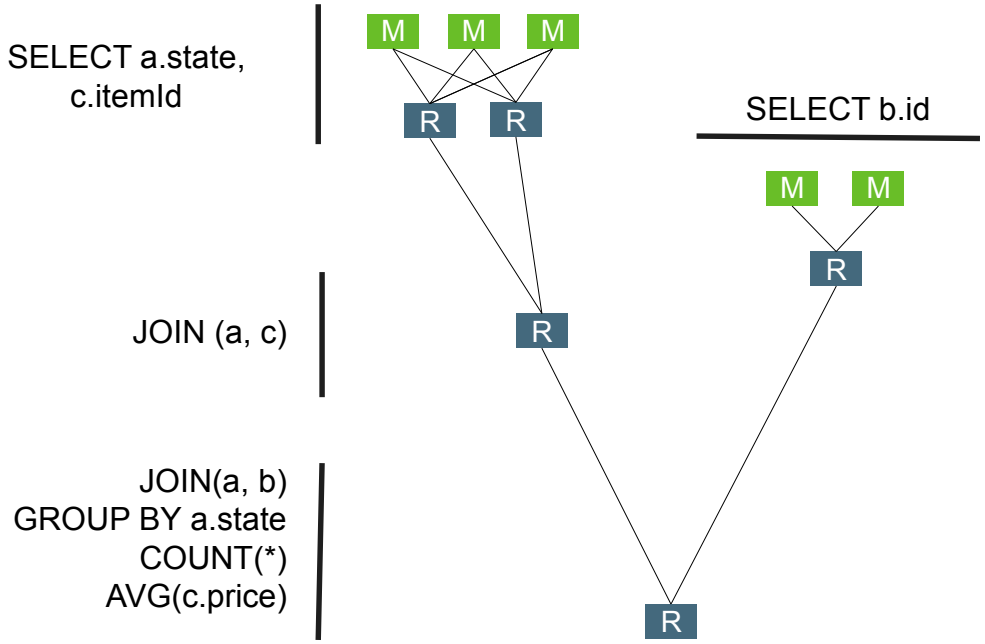
```
SELECT a.state, COUNT(*), AVG(c.price)
FROM a
JOIN b ON (a.id = b.id)
JOIN c ON (a.itemId = c.itemId)
GROUP BY a.state
```

Tez avoids unneeded writes to HDFS

Hive – MapReduce



Hive – Tez



Hortonworks®

Overview of Stinger

Performance Optimizations

100X+ Faster Time to Insight

Deeper Analytical Capabilities

Base Optimizations

Generate simplified DAGs
In-memory Hash Joins

YARN

Next-gen Hadoop data processing framework

+

Tez

Express tasks more simply
Eliminate disk writes
Pre-warmed Containers

ORCFile

Column Store
High Compression
Predicate / Filter Pushdowns

+

Vector Query Engine

Optimized for modern processor architectures

Query Planner

Intelligent Cost-Based Optimizer

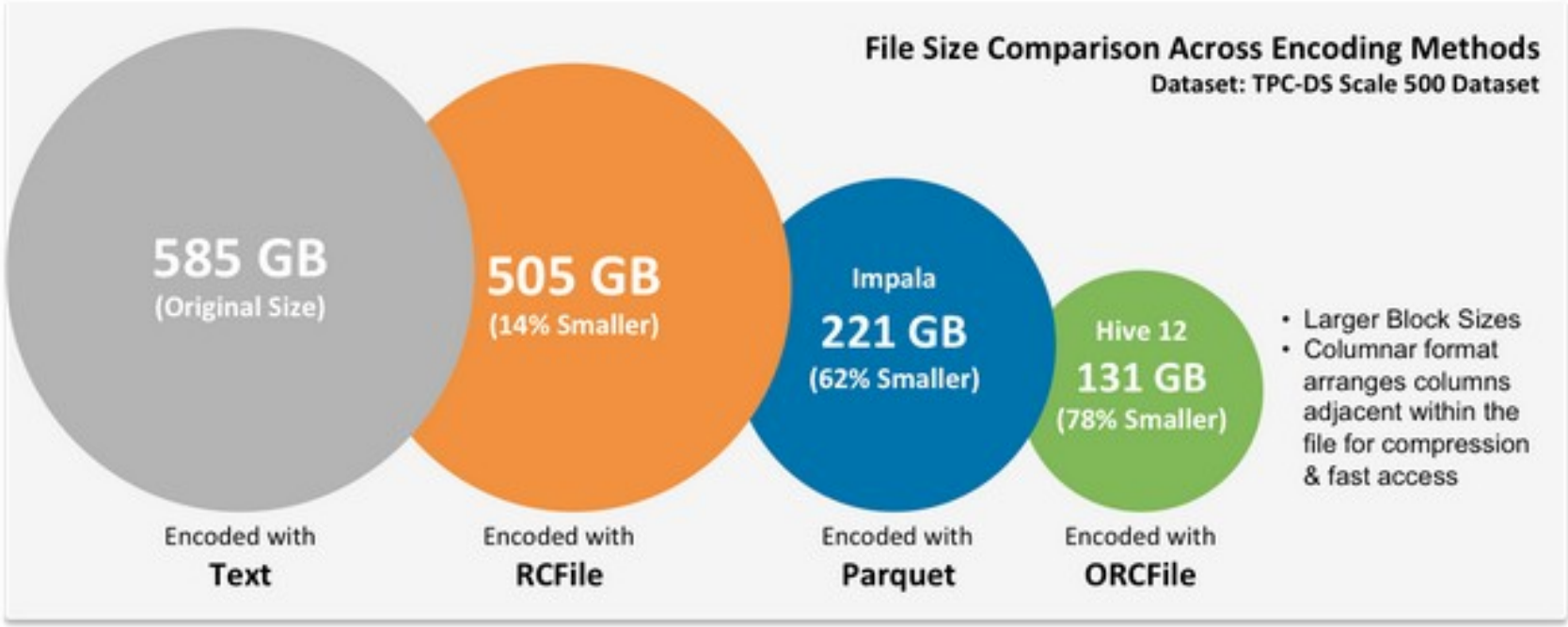
Hortonworks®

Existing File Format

	TEXTFILE	SEQUENCEFILE	RCFILE
Data type	text only	text/binary	text/binary
Internal Storage order	Row-based	Row-based	Column-based
Compression	File-based	Block-based	Block-based
Splitable*	YES	YES	YES
Splitable* after compression	NO	YES	YES

*** Splitable: Capable of splitting the file so that a single huge file can be processed by multiple mappers in parallel.**

Existing File Format



http://www.enterprisetech.com/2014/04/11/facebook-compresses-300-pb-data-warehouse/

How Facebook Compresses Its 300 PB Data Warehouse

April 11, 2014 by Timothy Prickett Morgan

When you have a 300 PB data warehouse running atop Hadoop, you do everything in your power to keep from adding another rack of disk drives to this beast. While social network giant Facebook is not afraid to throw a lot of hardware at a scalability problem, its software engineers are always looking for

Hive & Pig

Hive & Pig work well together and many customers use both

Hive is a good choice:

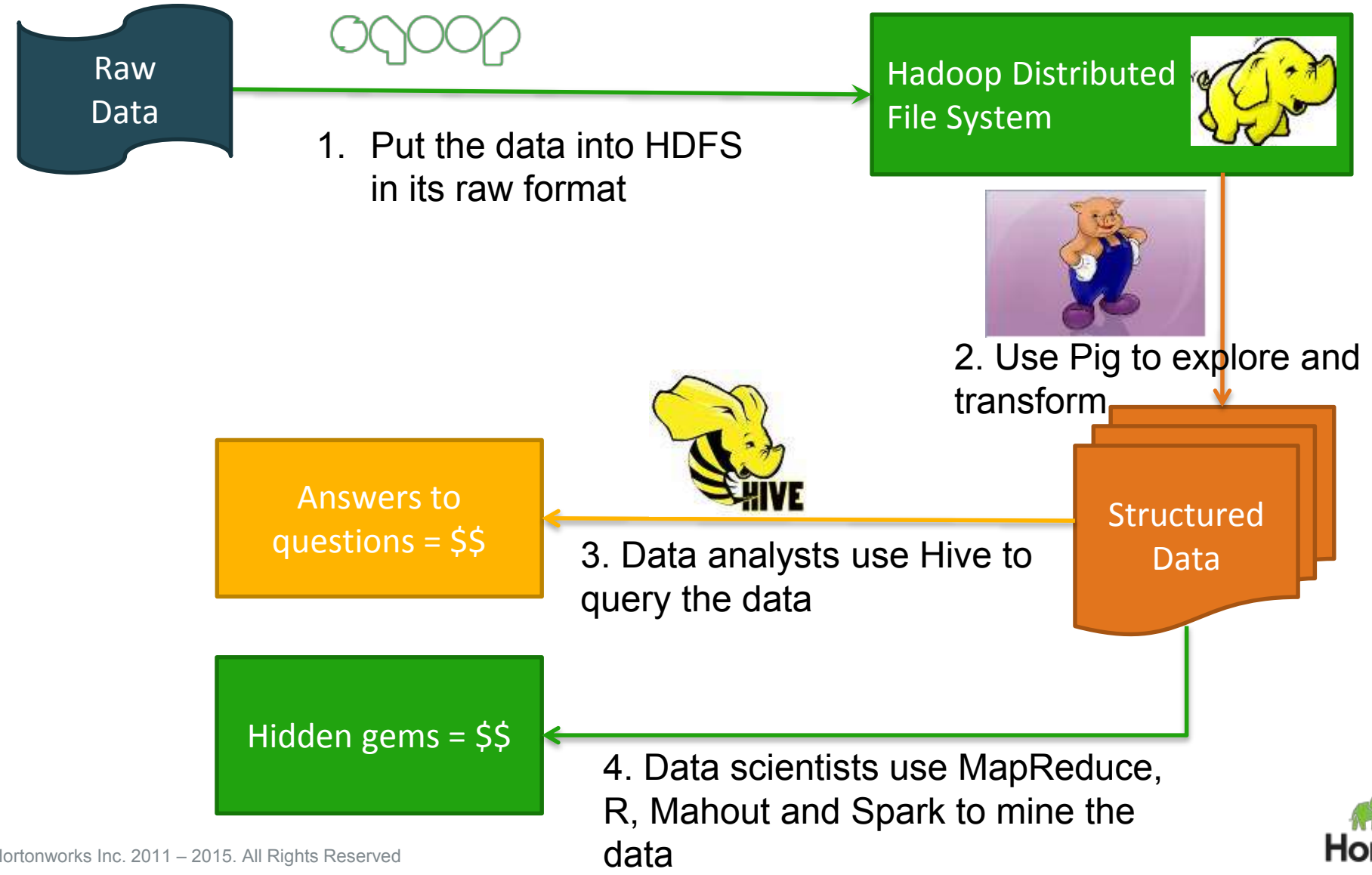
- if you are familiar with SQL
- when you want to query data
- when you need an answer to specific questions

Pig is a good choice:

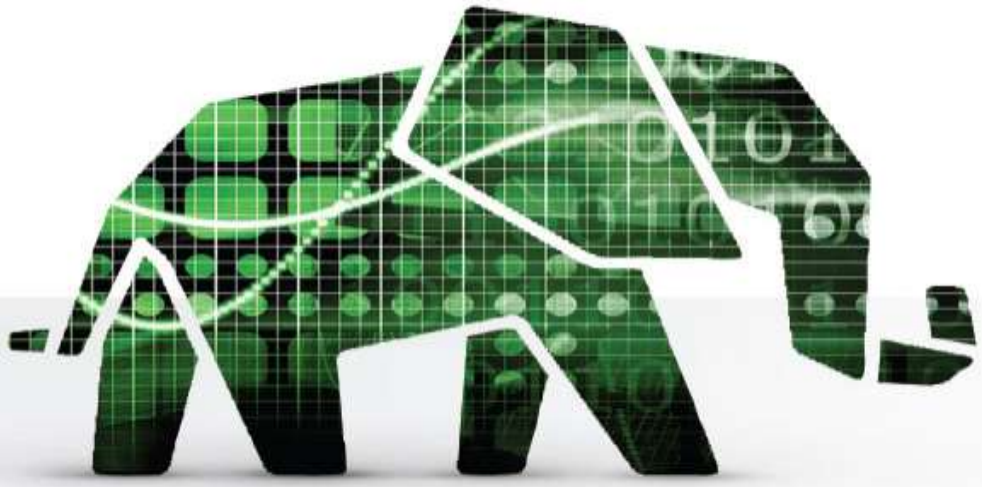
- For ETL (Extract, Transform, Load)
- for preparing data for analysis
- when you have a long series of steps to perform

Hortonworks®

Pig and Hive Sample Scenario



Thank you!



rafael@hortonworks.com

@racoss

Hortonworks®