



**Now with
Hadoop 2.0**



uweseiler

codecentric 

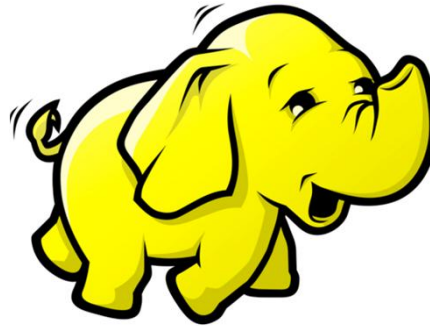


Introduction to the Hadoop Ecosystem

About me



Big Data Nerd



Hadoop Trainer



MongoDB Author



Photography Enthusiast



Travelpirate

About us

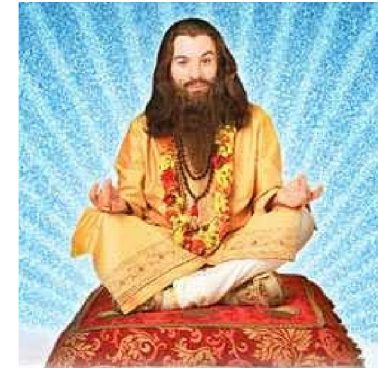
codecentric  is a bunch of...



Big Data Nerds



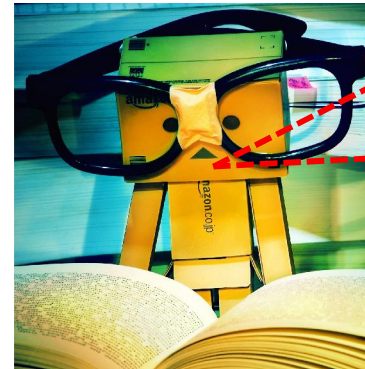
Agile Ninjas



Continuous Delivery Gurus



Enterprise Java Specialists



Performance Geeks

Join us!

Agenda

- **What is Big Data & Hadoop?**
- **Core Hadoop**
- **The Hadoop Ecosystem**
- **Use Cases**
- **What's next? Hadoop 2.0!**

Agenda

- **What is Big Data & Hadoop?**
- **Core Hadoop**
- **The Hadoop Ecosystem**
- **Use Cases**
- **What's next? Hadoop 2.0!**

Big Data

Big Data is like teenage sex:

everybody talks about it,
nobody really knows how to
do it, everyone thinks
everyone else is doing it, so
everyone claims they are
doing it...

\$7 to 10 billion

spent annually on technology by each of the **4 largest universal banks**

Around
3.8 petabytes
of data is managed per firm

They combine a database of

1.5 billion

pieces of information, with publicly available economic statistics from the U.S. government

Consumer trend reports are generated in seconds, instead of weeks or months

J.P.Morgan Chase & Co.

BANK

2 of the best techniques to improve client retention:

+ Sentiment analysis
Predictive analytics

Financial Industries

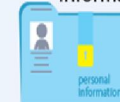
25% of Big Data use comes from the financial industry

Commercial banks collect data such as:

credit cards



customer information

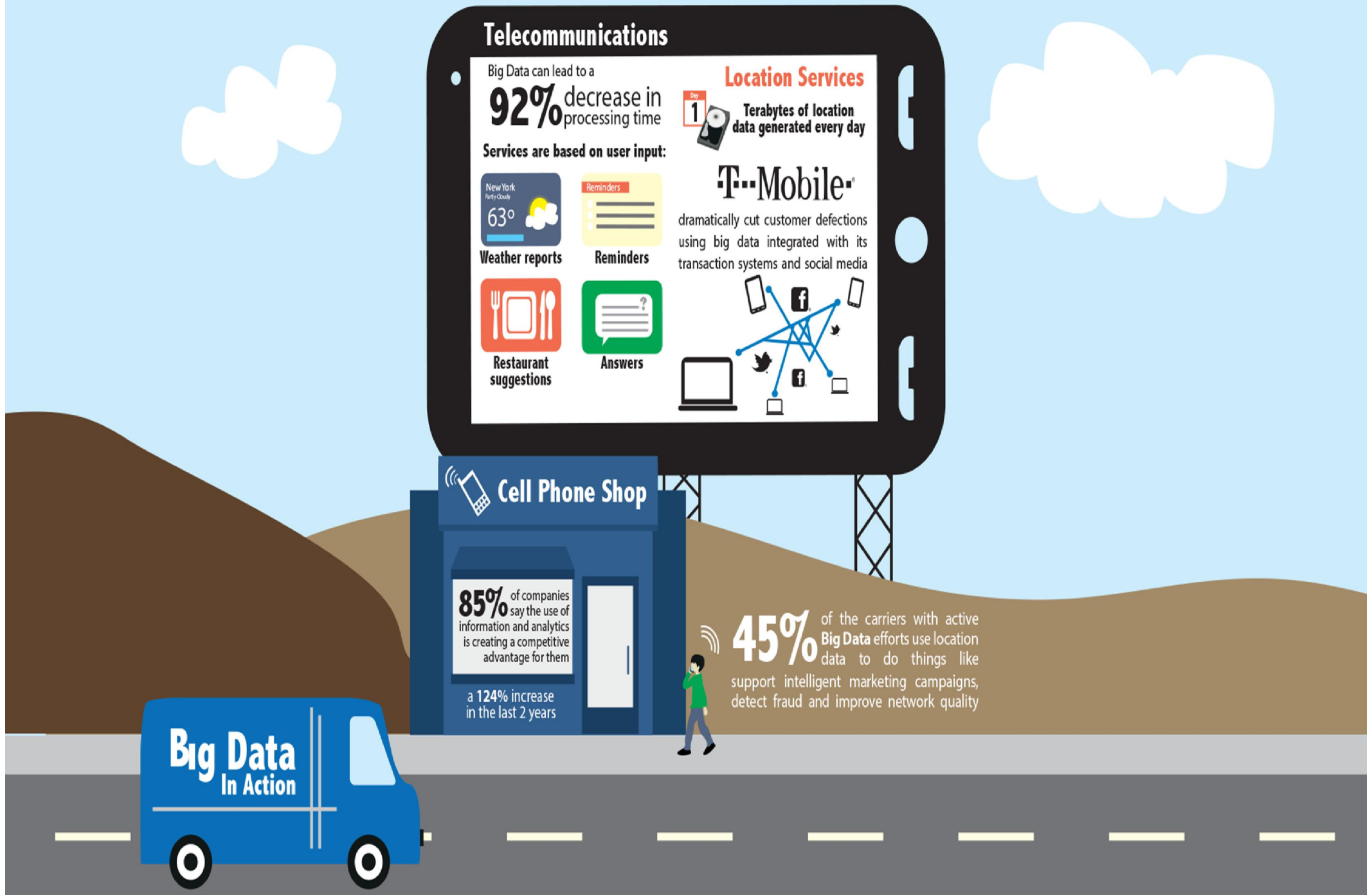


deposits
salaries

mortgages



loans



Retail Industry

A point-of-sale system holds
3 million to 15 million
transactions

Walmart

collects more than
2.5 petabytes
of data every hour
from its customer
transactions.

10 gigabytes
of data

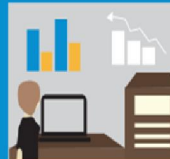
McKinsey & Company.

Estimates that a retailer using
big data analytics to its fullest
potential could increase its
operating margin by
more than **60%**

Combining data from:



- Web browsing patterns
- Social media

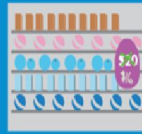


- Industry forecasts
- Existing customer records

Can help to:

- Predict trends
- Prepare for demand
- Pinpoint customers
- Optimize pricing and promotions
- Monitor real-time analytics and results

Retailers analyze:



- Sales
- Pricing
- Weather data
- Demographic data

Merchants can use POS data to:

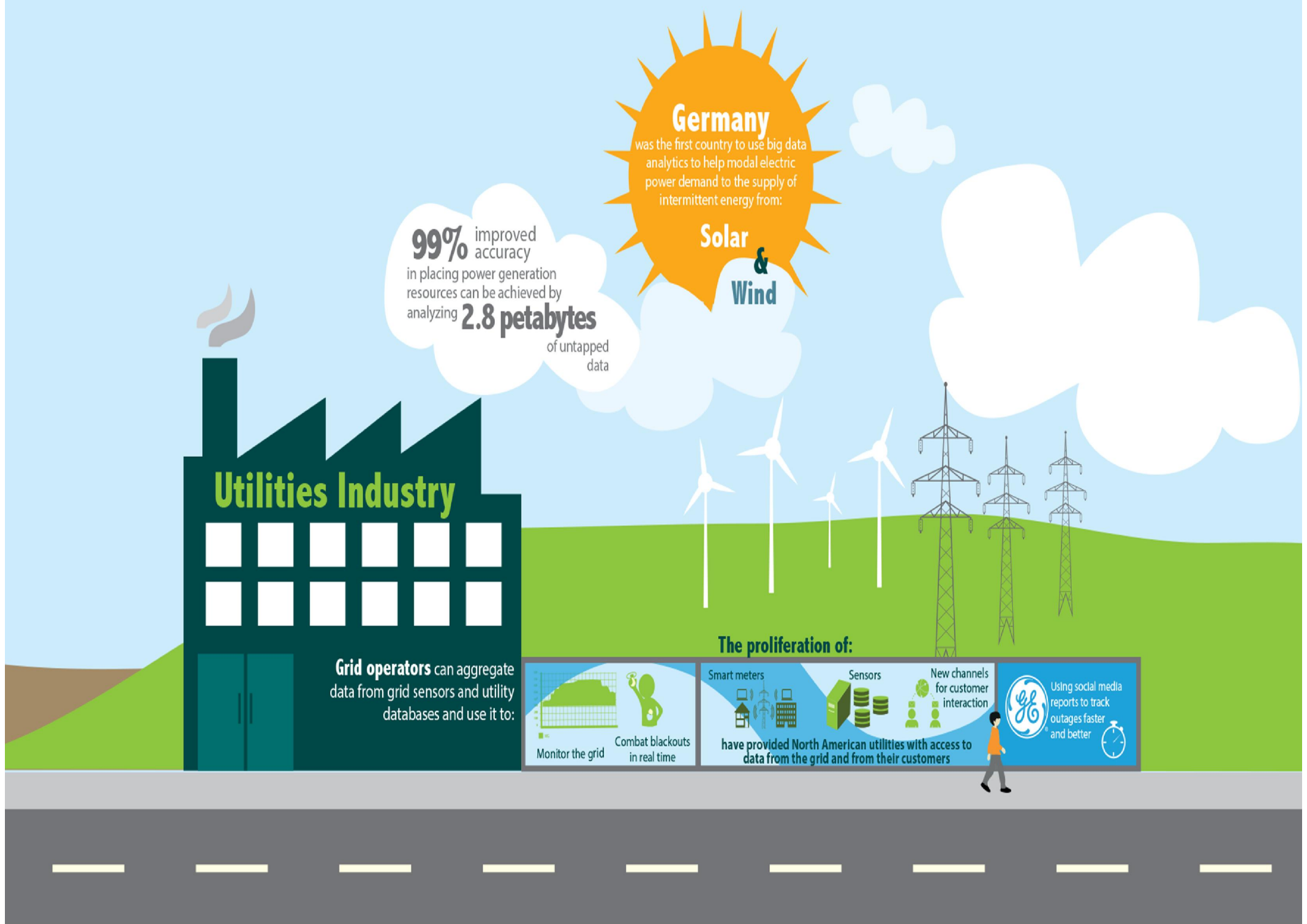
- Reduce out-of-stock situations
- Predict the next top-selling item

Product selections
at particular stores

This allows them to tailor:

Timing of price
markdowns





Using text-mining algorithms, researchers:

- Culled more than 10,000 mentions
- Summarized the most relevant comments
- To make an informed decision
- While minimizing costs

Big Data analysis of social media comments

can predict trends in automotive equipment failures.



Automobile Industry

Ford Motor Co.™ scoured auto-enthusiast websites and owner forums to see what drivers were saying about turn indicators instead of launching a full-scale market-research effort

There are upwards of 10,000+ sensors per vehicle generating data including:

MPH
MPG
RPM

Tire wear
Oil viscosity
Fuel efficiency

Oil pressure
Water temperature
Engine temperature



Auto-Service

New sources of Big Data can be leveraged to:

- Accelerate product design
- Improve vehicle performance
- Enhance the driver experience





Sports

The Oakland A's™ used analytics to identify and obtain relatively undervalued players



VS



In 2001 they won:



With:



...and nearly beat them in the MLB™ playoffs

Analytics are used to drive decisions in:

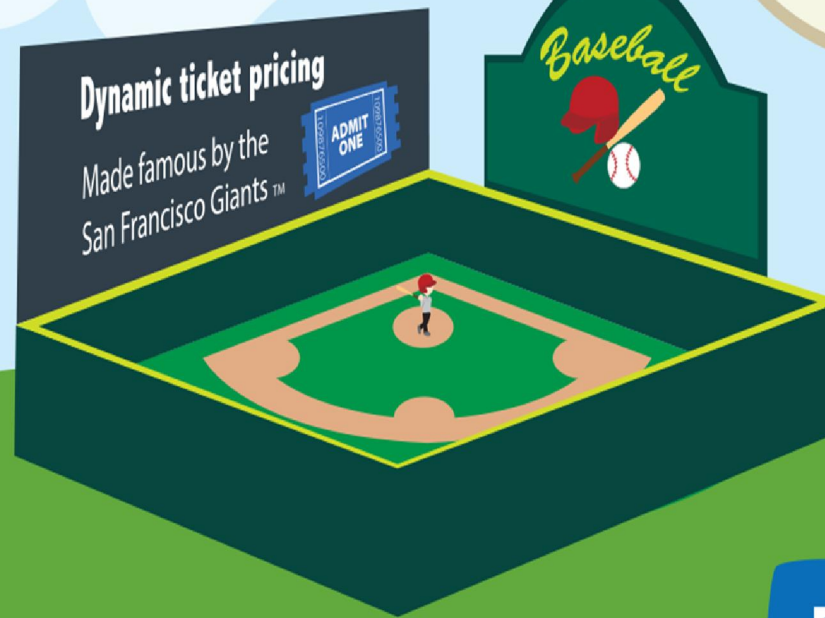


The competition is now between professional sports teams and third parties like StubHub!™

Price tickets based on demand

Big Data is used to predict attendance and price tickets accordingly

Teams can keep attendance up for less-popular games, and not leave money on the table for the most in-demand games





Healthcare Industry

Researchers have found that:

A couple of weeks **before there is an increase in flu patients** coming to hospital emergency rooms in a region, **there is a spike in Google™ search requests for terms like "flu symptoms" and "flu treatments"**

More and more **start-ups** are searching for ways to comb through **large volumes of clinical data** on:

By **adopting the electronic patient medical records**:



Medication



Allergies



Procedures

Medical researchers can **aggregate information** about health care outcomes to **reveal patterns** that lead to:

A **20% decrease in patient mortality** can be achieved by analyzing patient data

Over \$300 billion in additional annual value can be added to the **US Healthcare industry** if big data is managed properly

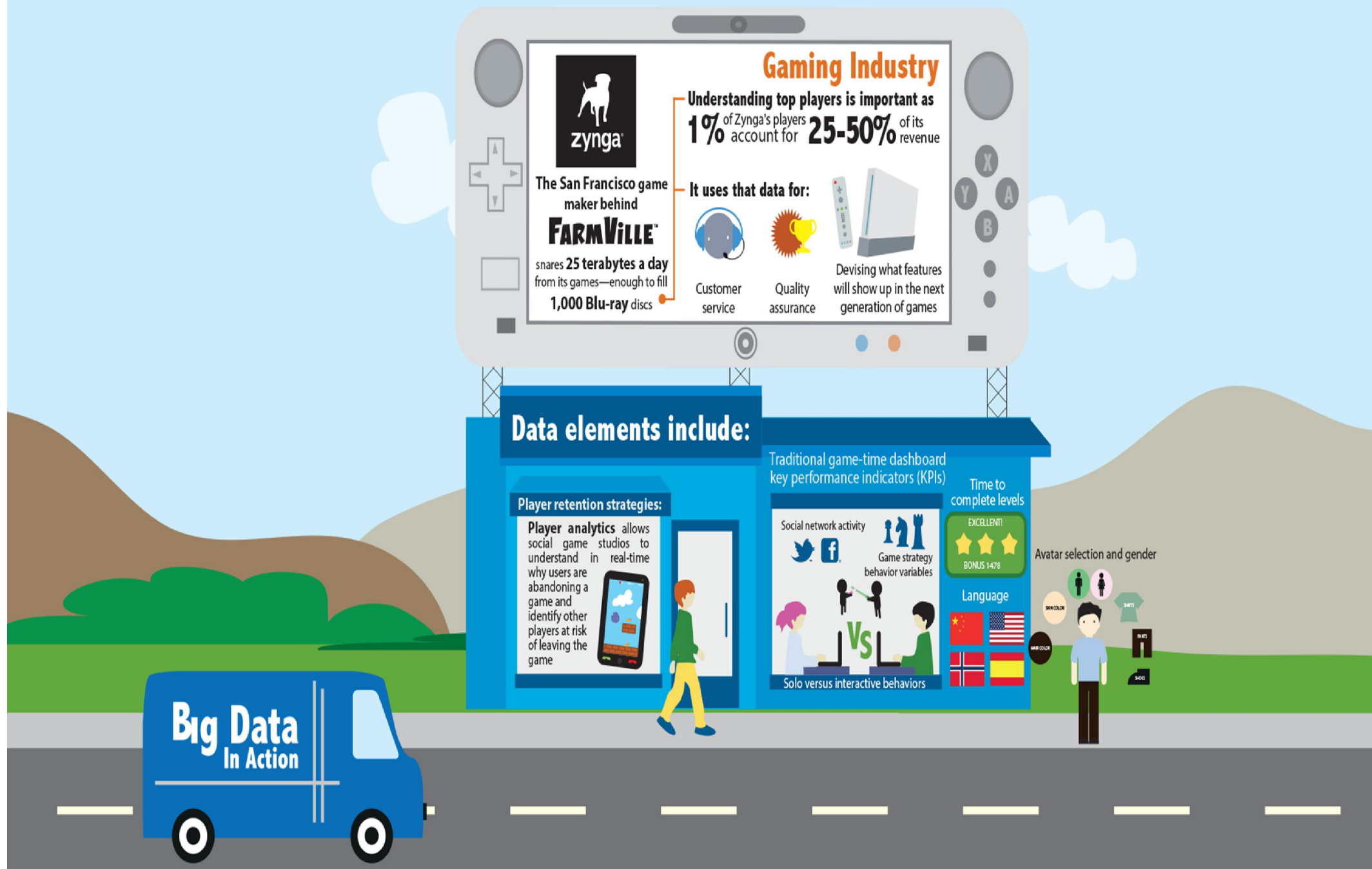
More effective treatments

Detection of outbreaks



Kaiser Permanente, has fully implemented its **HealthConnect System**, to ensure information exchange across all medical facilities

It has **reduced total office visits** by **26.2%**



Online Dating Industry

Today, approximately **20% of relationships** begin through some sort of online or mobile dating service.

Online dating services, like **match.com**, constantly improve the algorithms for **matching men and women on dates** sifting through their Web listings of:

- ♥ Personal characteristics
- ♥ Communications
- ♥ Reactions

The average age of an online dater is **48**

When new users sign up, they are required to fill out a **400-question profile**

The system **compares the user's answers** to the site's **20 million users**

The process requires **a billion calculations** for each bachelor or bachelorette before providing matches



eHarmony®



Their dataset alone exceeds **4 terabytes of total data**, not including photos and other information

More than **40% of the country's single population** have an online dating profile

Out of eHarmony's 44 million registered users, **1.2 million** have been married

City Impact

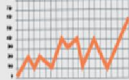
Oslo, Norway

reduced street lighting **energy consumption by 62%** with a smart solution provided by big data analysis



US will implement a **predictive analytics** and smart metering project to:

Monitor consumption remotely; **expected 20% reduction** in water consumption and savings of **\$1 million per year**



Identify water leaks

NYC has leveraged big data to **double the city's hit rate** in finding stores selling bootleg cigarettes; sped the removal of trees destroyed by Hurricane Sandy and helped steer overburdened housing inspectors directly to lawbreaking buildings



Portland, Oregon used technology to optimize the timing of its traffic signals and was able to eliminate more than **157,000 metric tons** of CO2 emissions in just six years – the equivalent of taking **30,000 passenger vehicles** off the roads for an entire year

In **Los Angeles**, Xerox implemented dynamic parking systems that adjust parking rates to influence parking behavior, and the preliminary results show:



A **15% decrease** in parking occupancy in the congested areas



This influenced a **10% increase** in less desirable parking areas

Each city is built on information technology that operates scientifically, based on data.



In China

the Ministry of Housing released a list of **90 cities** that will be turned into Smart Cities in the next 3 to 5 years

The smart city project of **Rivas Vaciamadrid in Spain** through combination of smart grid and energy management, access control, air quality monitoring, and traffic management, has realized:

A **50% reduction** in **Information and Communication Technology** spending

Energy savings of **35%**

For example, electricity is allocated according to consumption data provided by local factories.



My favorite definition

Big Data *is when the data itself becomes part of the problem.*

The classic definition

Volume



Velocity



Variety



« Big Data » != Hadoop

Not

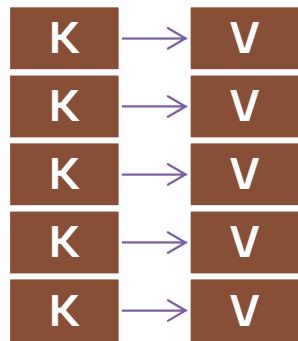
Only

~~NOSQL~~

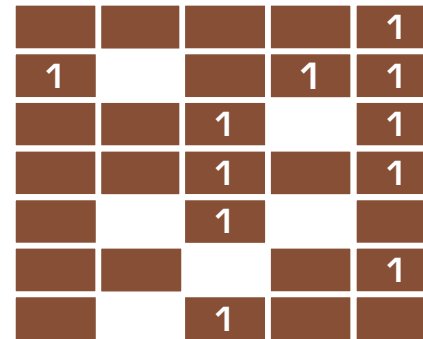
Relational

Classification of NoSQL

Key-Value Stores



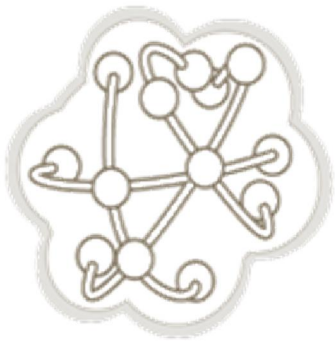
Column Stores



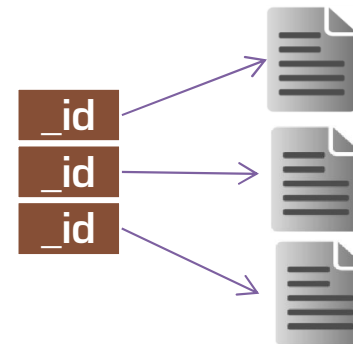
A diagram illustrating Column Stores. It shows a 6x5 grid of brown boxes. Some boxes contain the number '1', while others are empty, representing a sparse matrix of data. The '1's are located at the following positions (row, column): (1, 5), (2, 1), (2, 4), (3, 3), (4, 3), (4, 5), (5, 3), (5, 5), and (6, 3).



Graph Databases



Document Stores



Horizontal Scaling

Vertical Scaling

RAM
CPU
Storage

Vertical Scaling

RAM
CPU
Storage

Vertical Scaling

RAM
CPU
Storage

Horizontal Scaling

RAM
CPU
Storage

Horizontal Scaling

RAM
CPU
Storage

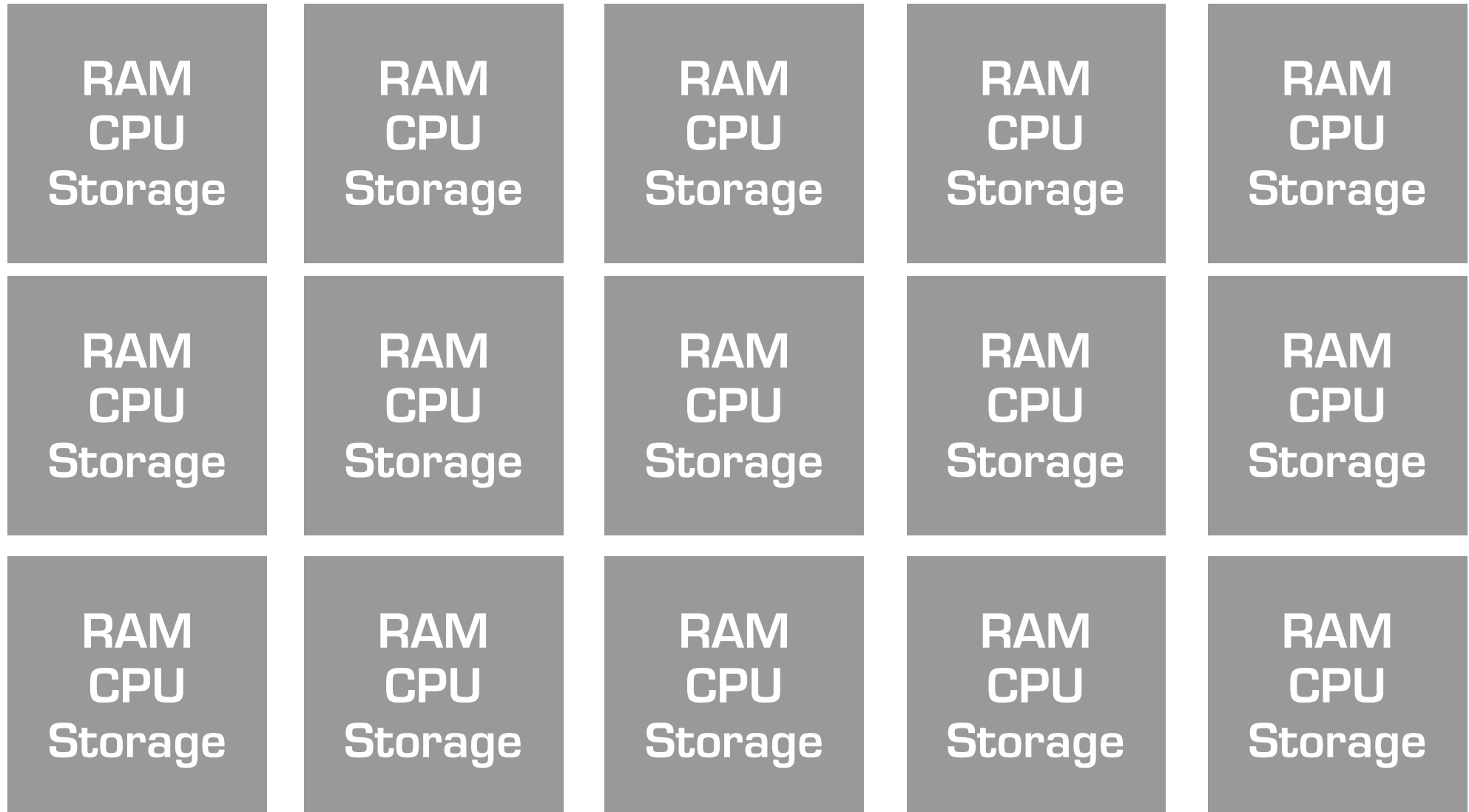
RAM
CPU
Storage

RAM
CPU
Storage

RAM
CPU
Storage

RAM
CPU
Storage

Horizontal Scaling



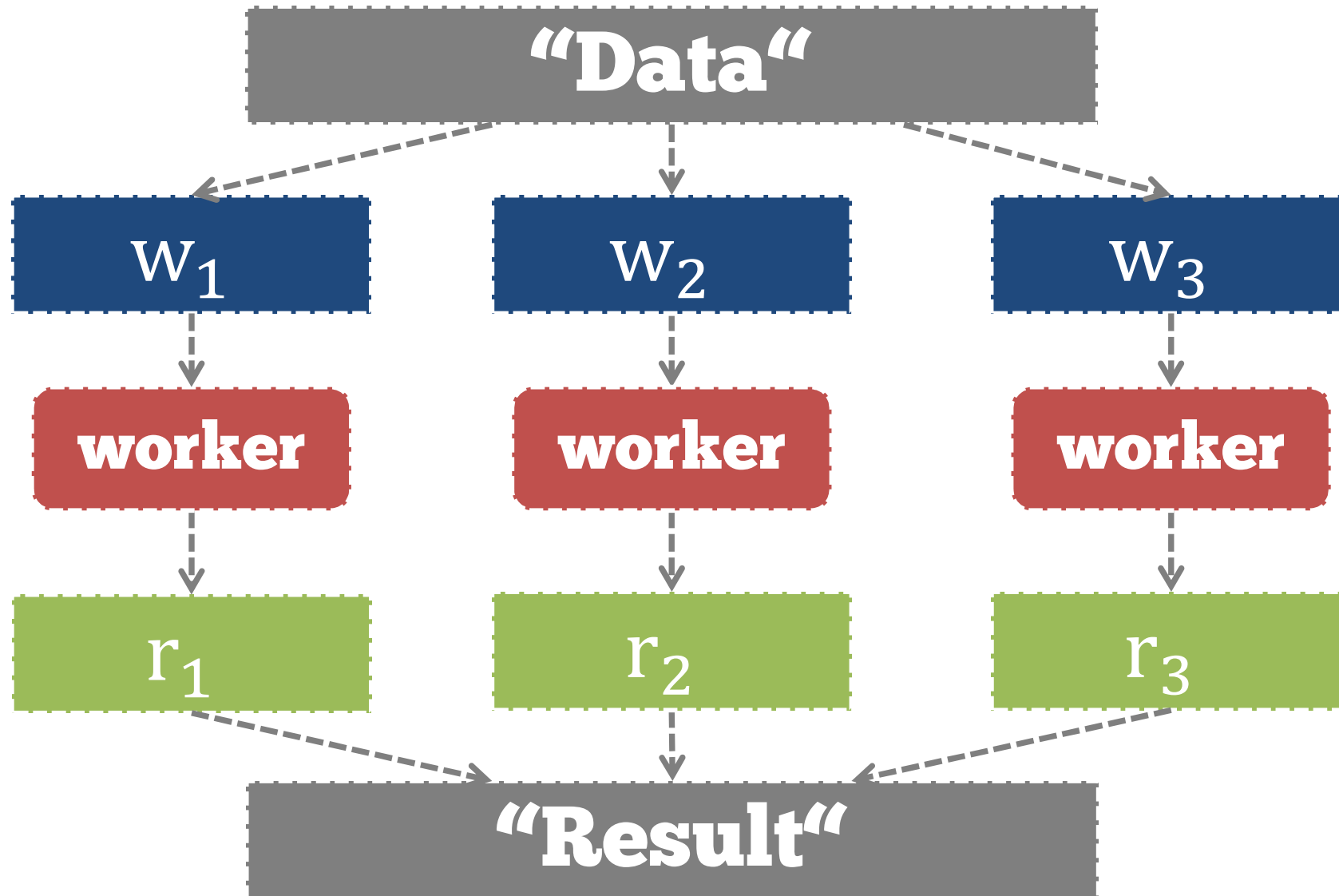
Why Hadoop?

**TRADITIONAL DATASTORES ARE EXPENSIVE TO SCALE
AND BY DESIGN DIFFICULT TO DISTRIBUTE**

COMMODITY HARDWARE IS CHEAP AND POWERFUL

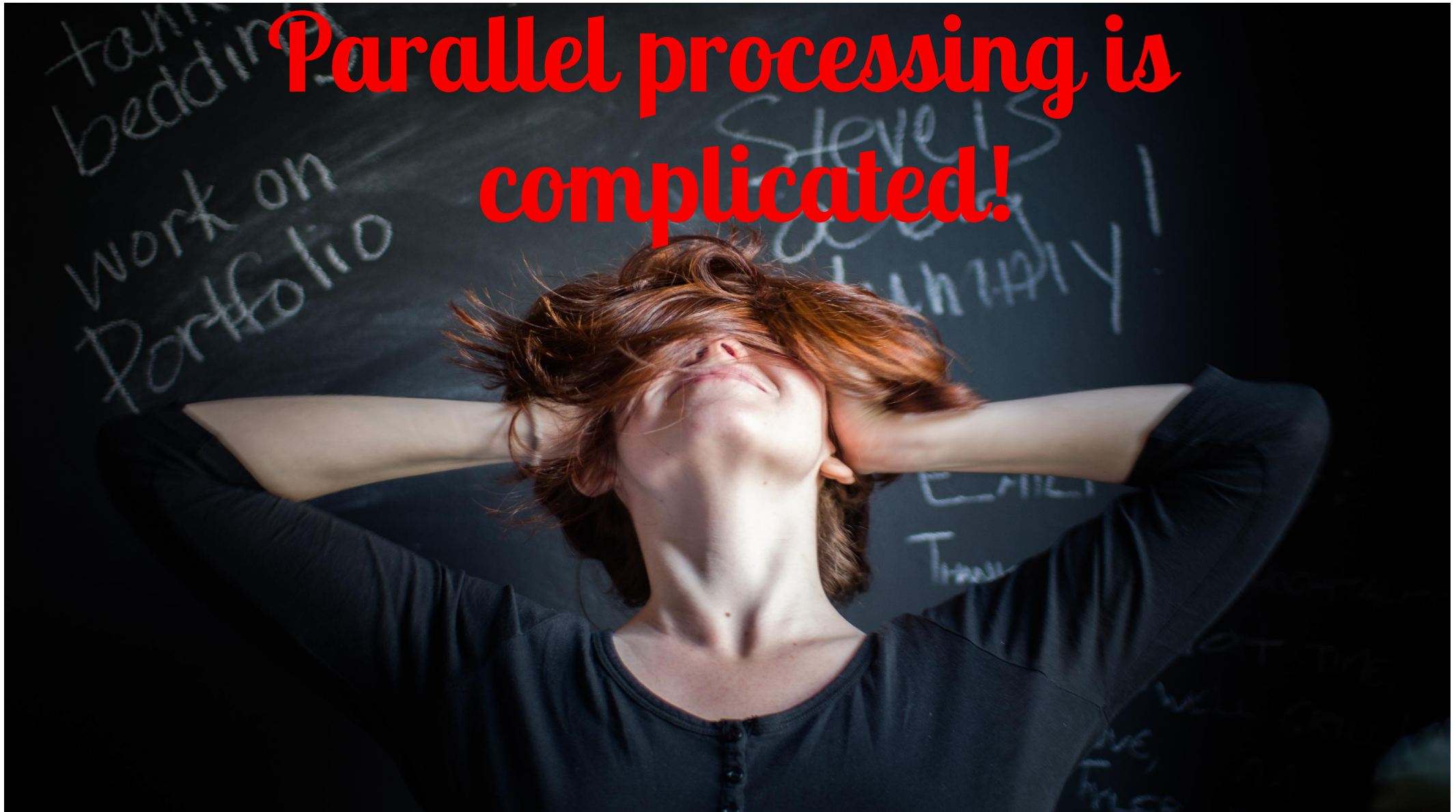
Scale out is the way to go!

How to scale data?



But...

Parallel processing is complicated!



But...

**Data storage is not
trivial!**

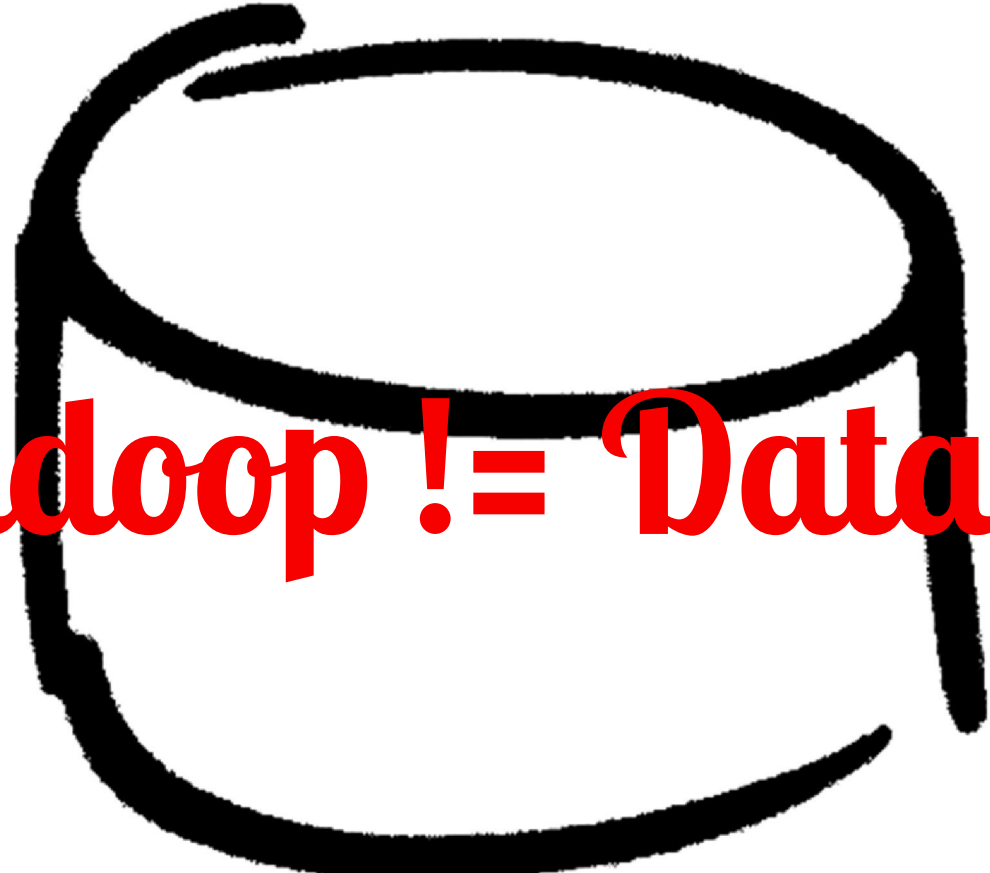


What is Hadoop?



Distributed Storage and Computation Framework

What is Hadoop?



Hadoop != Database

What is Hadoop?

**“Swiss army knife
of the 21st century”**



<http://www.guardian.co.uk/technology/2011/mar/25/media-guardian-innovation-awards-apache-hadoop>

The Hadoop App Store



HDFS



MapRed



HCat



Pig



Hive



HBase



Ambari



Avro



Cassandra



Chukwa



Flume



Hama



HyperT



Impala



Mahout



Nutch



Oozie



Scoop



Scribe



Tez



Vertica



Whirr



ZooKeeper



Horton



Cloudera



MapR



EMC



Intel



IBM



Talend



TeraData



Pivotal



Informatica



Microsoft



Pentaho



Jasper



Sync



Kognitio



Tableau



Splunk



Platfora



Rack



Karma



Actuate



MicStrat

The Hadoop App Store

Apache Hadoop



- HDFS
- MapReduce
- Hadoop Ecosystem
- Hadoop YARN

Hadoop Distributions



- Test & Packaging
- Installation
- Monitoring
- Business Support



- Integrated Environment
- Visualization
- (Near-)Realtime analysis
- Modeling
- ETL & Connectors

Big Data Suites

less

Functionality

more →

Agenda

- **What is Big Data & Hadoop?**
- **Core Hadoop**
- **The Hadoop Ecosystem**
- **Use Cases**
- **What's next? Hadoop 2.0!**

**OK, first things
first!**

I want to **store all of
my <<Big Data>>**

There is an app for that!



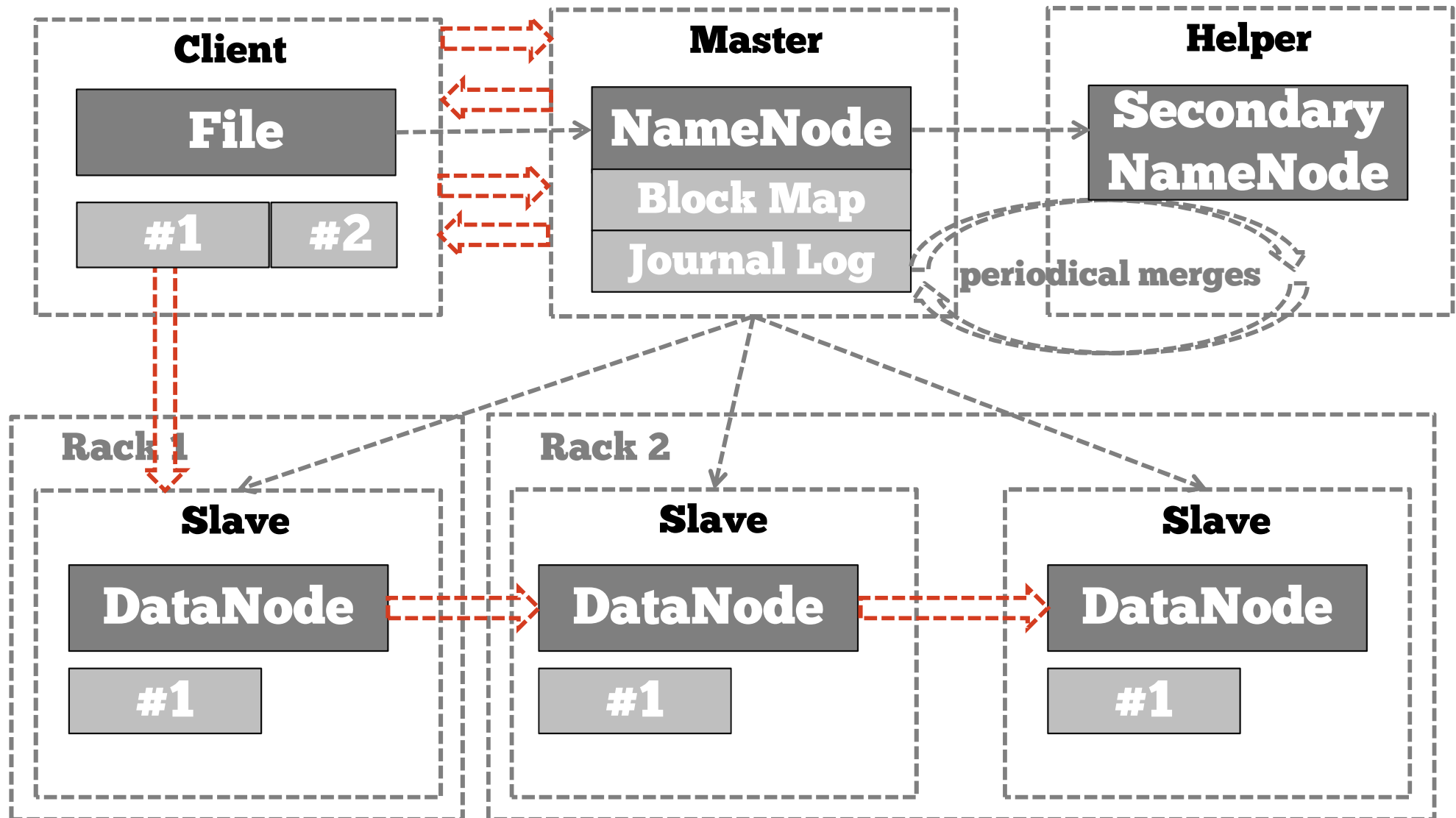
- **Distributed file system for redundant storage**
- **Designed to reliably store data on commodity hardware**
- **Built to expect hardware failures**

Intended for

- **large files**
- **batch inserts**

Write once, read many times

HDFS Architecture



Let's have a look...

Data stored, check!

**Now I want to
create **insights**
from my data!**

Data Processing

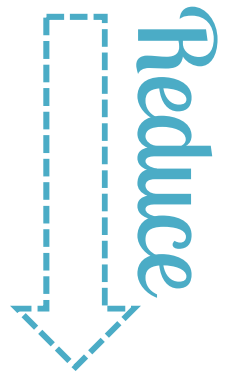
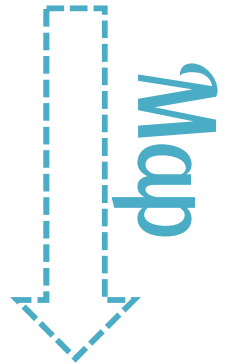
There is an app for that!



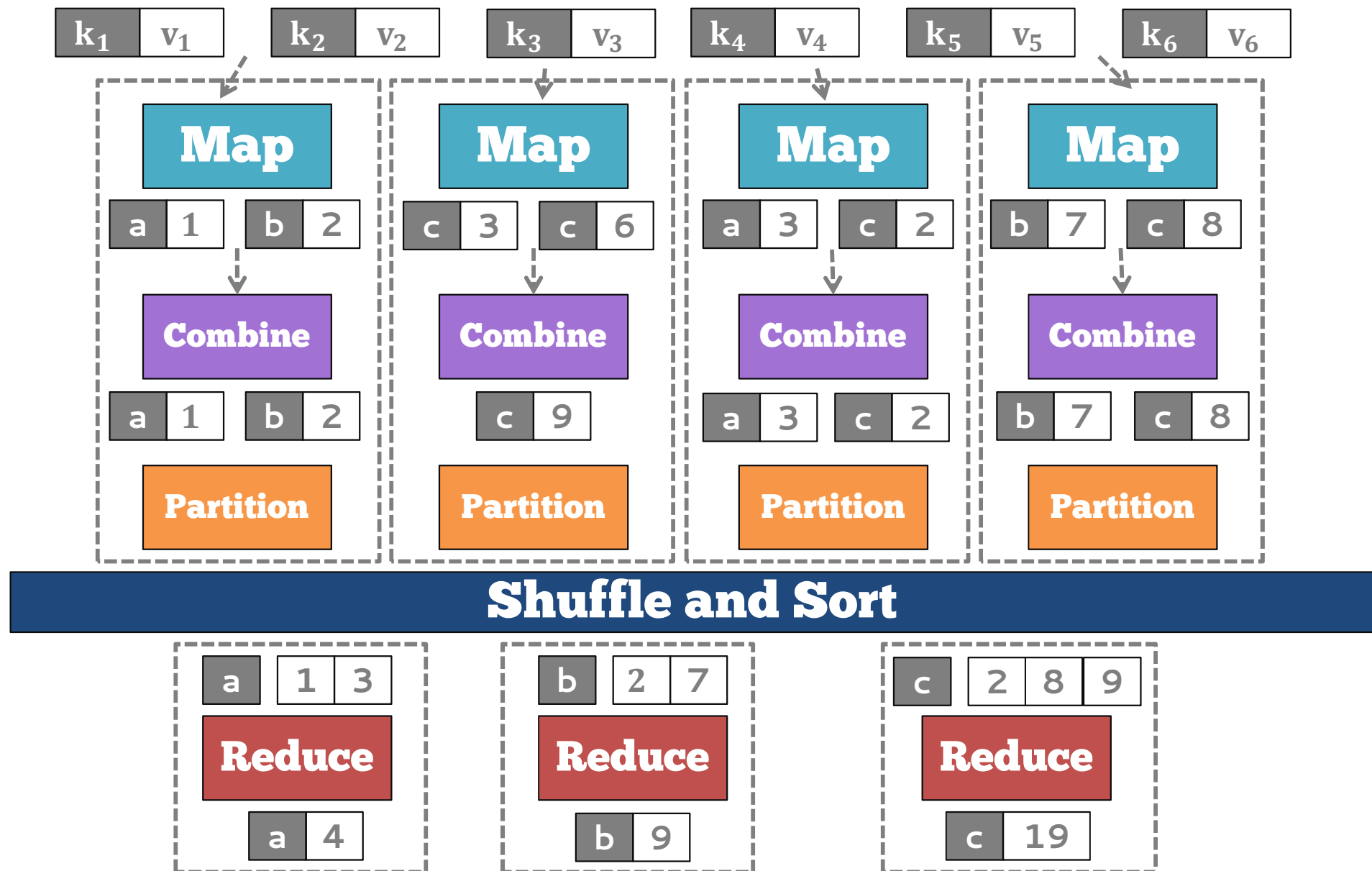
- **Programming model for distributed computations at a massive scale**
- **Execution framework for organizing and performing such computations**
- **Data locality is king**

Typical large-data problem

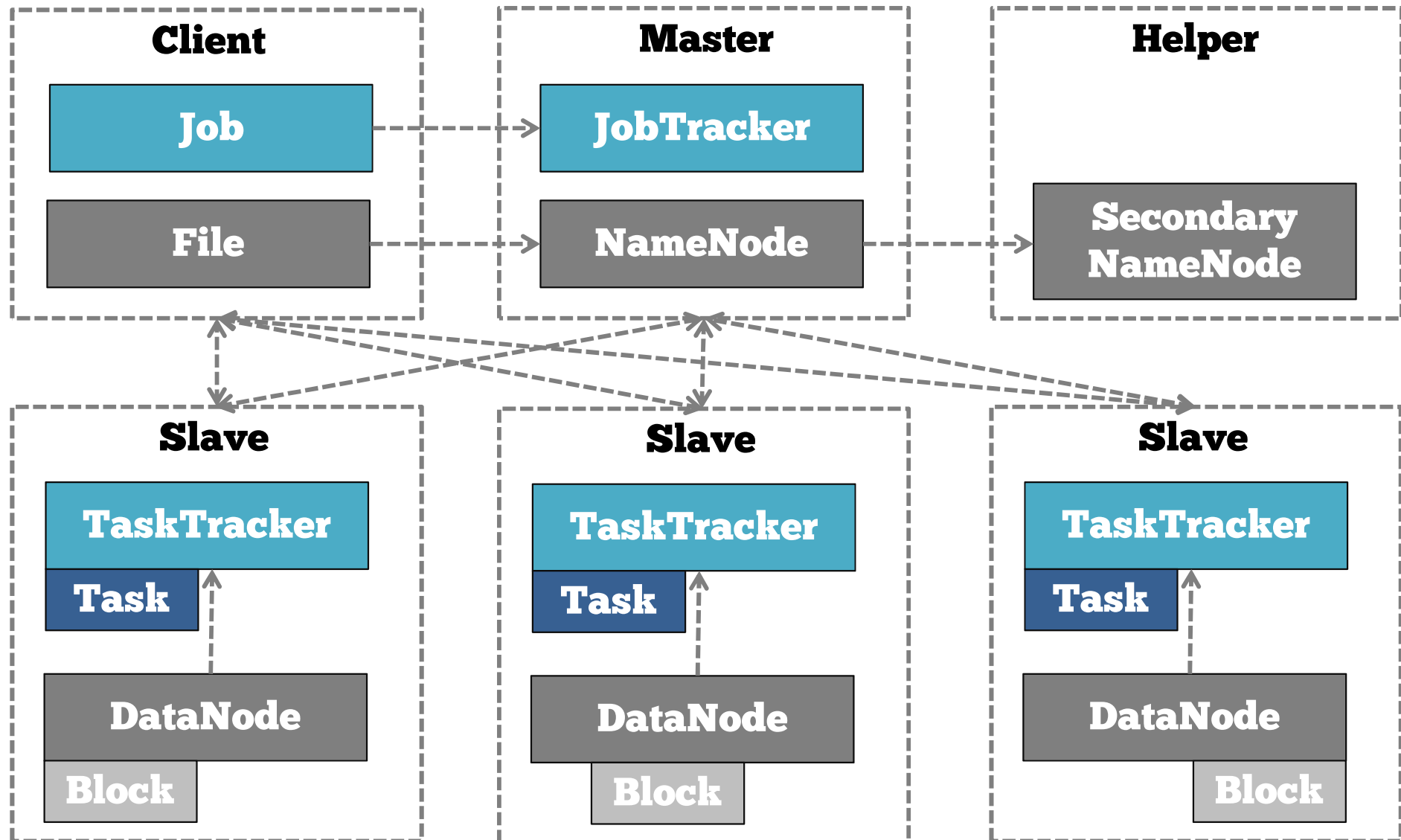
- **Iterate over a large number of records**
- **Extract something of interest from each**
- **Shuffle and sort intermediate results**
- **Aggregate intermediate results**
- **Generate final output**



MapReduce Flow



Combined Hadoop Architecture



Word Count Mapper in Java

```
public class WordCountMapper extends MapReduceBase implements
    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, OutputCollector<Text,
        IntWritable> output, Reporter reporter) throws IOException
    {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens())
        {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

Word Count Reducer in Java

```
public class WordCountReducer extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable>
{
    public void reduce(Text key, Iterator values, OutputCollector
        output, Reporter reporter) throws IOException
    {
        int sum = 0;
        while (values.hasNext())
        {
            IntWritable value = (IntWritable) values.next();
            sum += value.get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```


Let's have a look...

Agenda

- **What is Big Data & Hadoop?**
- **Core Hadoop**
- **The Hadoop Ecosystem**
- **Use Cases**
- **What's next? Hadoop 2.0!**

Java for MapReduce?

I dunno, dude...

I'm more of a **scripting** guy...

Scripting for Hadoop

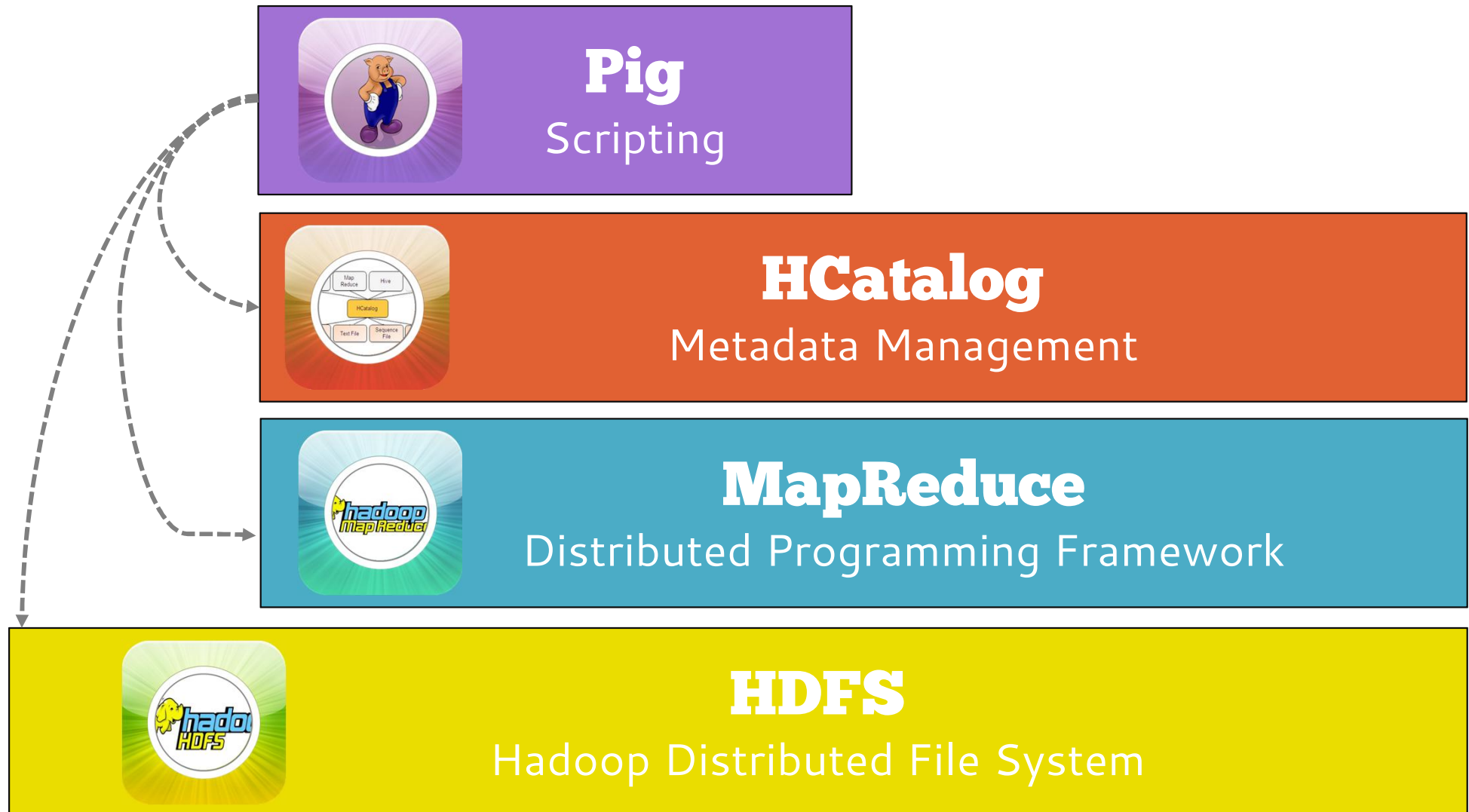
There is an app for that!



- **High-level data flow language**
- **Made of two components:**
 - **Data processing language Pig Latin**
 - **Compiler to translate Pig Latin to MapReduce**

*Abstracts you from specific details and
allows you to focus on data processing*

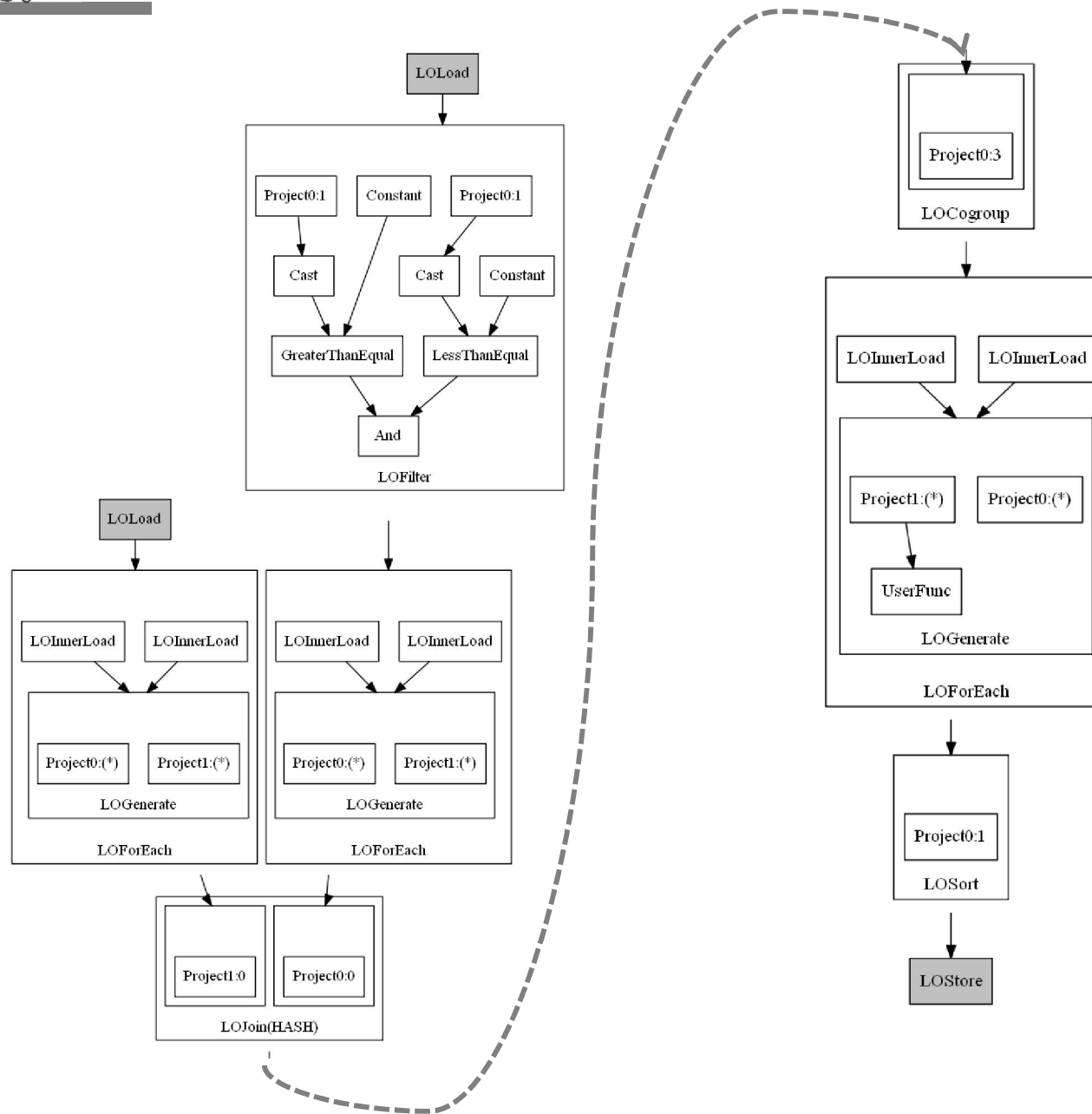
Pig in the Hadoop ecosystem



Pig Latin

```
users = LOAD 'users.txt' USING PigStorage(',') AS (name,  
    age);  
pages = LOAD 'pages.txt' USING PigStorage(',') AS (user,  
    url);  
  
filteredUsers = FILTER users BY age >= 18 and age <=50;  
joinResult = JOIN filteredUsers BY name, pages by user;  
grouped = GROUP joinResult BY url;  
summed = FOREACH grouped GENERATE group,  
    COUNT(joinResult) as clicks;  
sorted = ORDER summed BY clicks desc;  
top10 = LIMIT sorted 10;  
  
STORE top10 INTO 'top10sites';
```

Pig Execution Plan





Try that with Java...

```

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.MapperContext;
import org.apache.hadoop.mapred.MapperRunner;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.ReducerContext;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.JobControl;
import org.apache.hadoop.mapred.JobID;
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
            // 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 = s1 + "," + s2 + "," + s2;
                oc.collect(outval, new Text(outval));
                reporter.setStatus("OK");
            }
        }
    }

    public static class LoadJoins 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(1));
        }
    }

    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 jp = new JobConf(MRExample.class);
        jp.setJobName("Load Pages");
        jp.setInputFormat(TextInputFormat.class);

        jp.setOutputKeyClass(Text.class);
        jp.setOutputValueClass(Text.class);
        jp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(jp, new
            Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(jp,
            new Path("/user/gates/tmp/indexed_pages"));
        jp.setNumReduceTasks(1);
        Job loadPages = new Job(jp);

        JobConf jfu = new JobConf(MRExample.class);
        jfu.setJobName("Load and Filter Users");
        jfu.setInputFormat(TextInputFormat.class);
        jfu.setOutputKeyClass(Text.class);
        jfu.setOutputValueClass(Text.class);
        jfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(jfu, new
            Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(jfu,
            new Path("/user/gates/tmp/filtered_users"));
        jfu.setNumReduceTasks(1);
        Job loadUsers = new Job(jfu);

        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(LoadJoins.class);
        join.setReducerClass(ReduceURLs.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(1);
        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.setMapperClass(LoadClicks.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(1);
        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.setMapperClass(LoadClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
            Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
            Path("/user/gates/top100sitesforusers1to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

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

```

Let's have a look...

**OK, Pig seems quite
useful...**

**But I'm more of a
SQL person...**

SQL for Hadoop

There is an app for that!



- **Data Warehousing Layer on top of Hadoop**
- **Allows analysis and queries using a SQL-like language**

Hive is best for data analysts familiar with SQL who need to do dynamic queries, summarization and data analysis

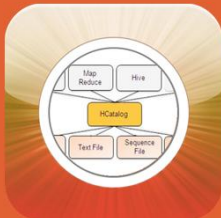
Hive in the Hadoop ecosystem



Pig
Scripting



Hive
Query



HCatalog
Metadata Management



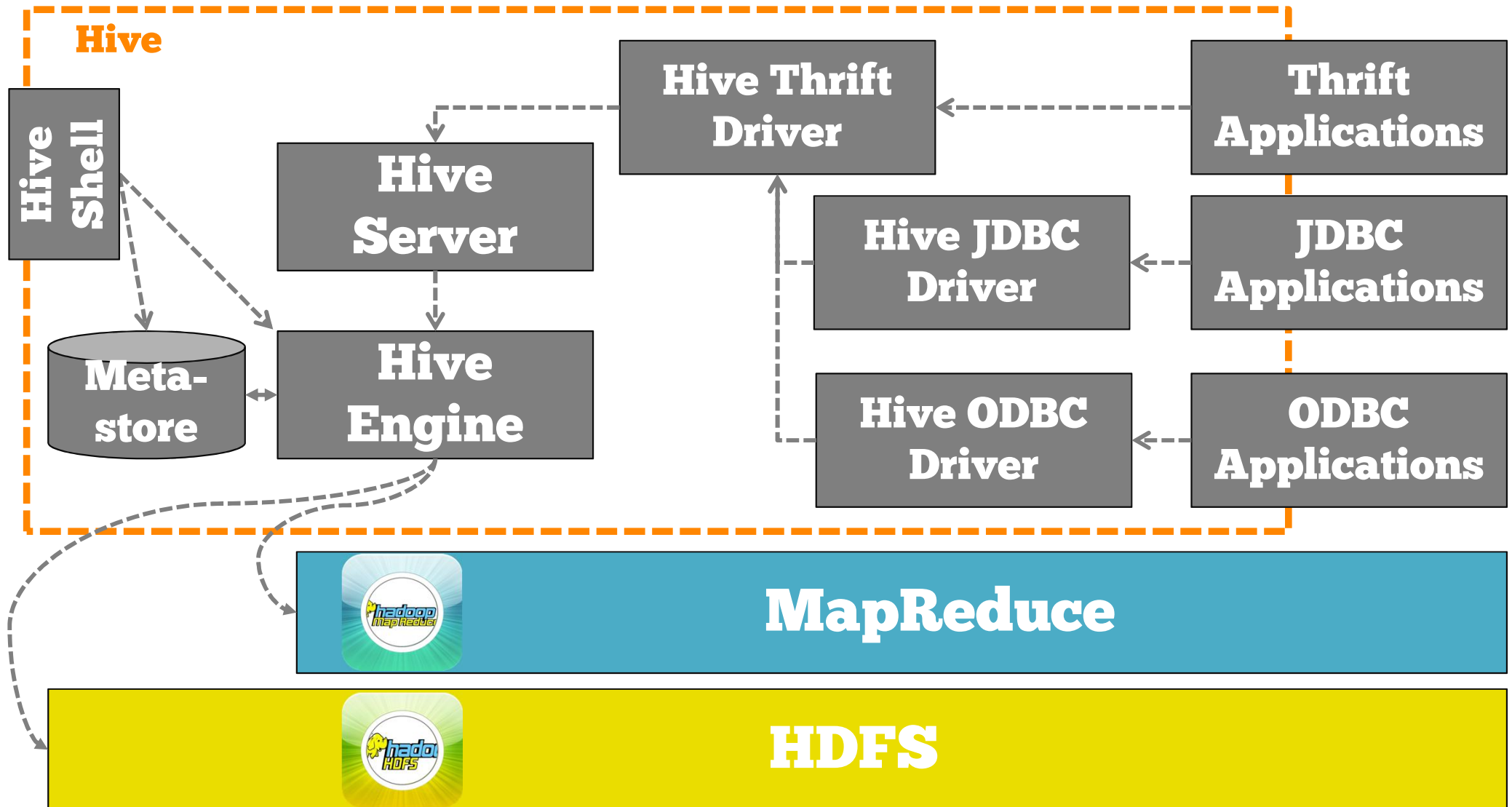
MapReduce
Distributed Programming Framework



HDFS
Hadoop Distributed File System



Hive Architecture



Hive Example

```
CREATE TABLE users(name STRING, age INT);  
CREATE TABLE pages(user STRING, url STRING);  
  
LOAD DATA INPATH '/user/sandbox/users.txt' INTO  
TABLE 'users';  
LOAD DATA INPATH '/user/sandbox/pages.txt' INTO  
TABLE 'pages';  
  
SELECT pages.url, count(*) AS clicks FROM users JOIN  
pages ON (users.name = pages.user)  
WHERE users.age >= 18 AND users.age <= 50  
GROUP BY pages.url  
SORT BY clicks DESC  
LIMIT 10;
```

Let's have a look...

But wait, there's still more!

More components of the **Hadoop Ecosystem**



Mahout

Machine Learning



HBase
NoSQL Database



Pig
Scripting



Hive
SQL-like queries



HCatalog
Metadata Management



MapReduce
Data processing



HDFS
Data storage



Scoop
Import & Export of
relational data



Flume
Import & Export of
data flows



ZooKeeper
Cluster Coordination



Ambari
Cluster installation & management



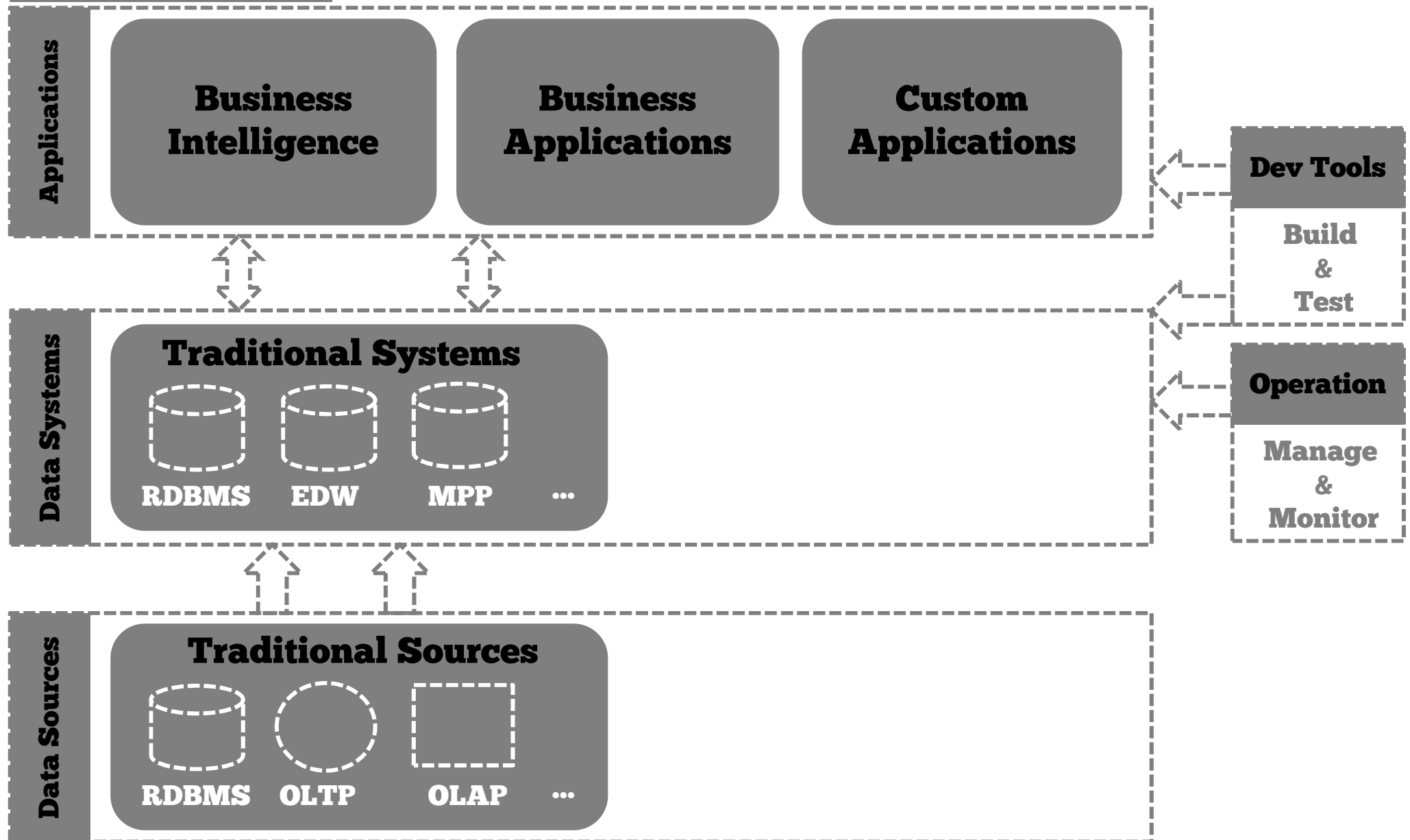
Oozie
Workflow automatization

Agenda

- **What is Big Data & Hadoop?**
- **Core Hadoop**
- **The Hadoop Ecosystem**
- **Use Cases**
- **What's next? Hadoop 2.0!**

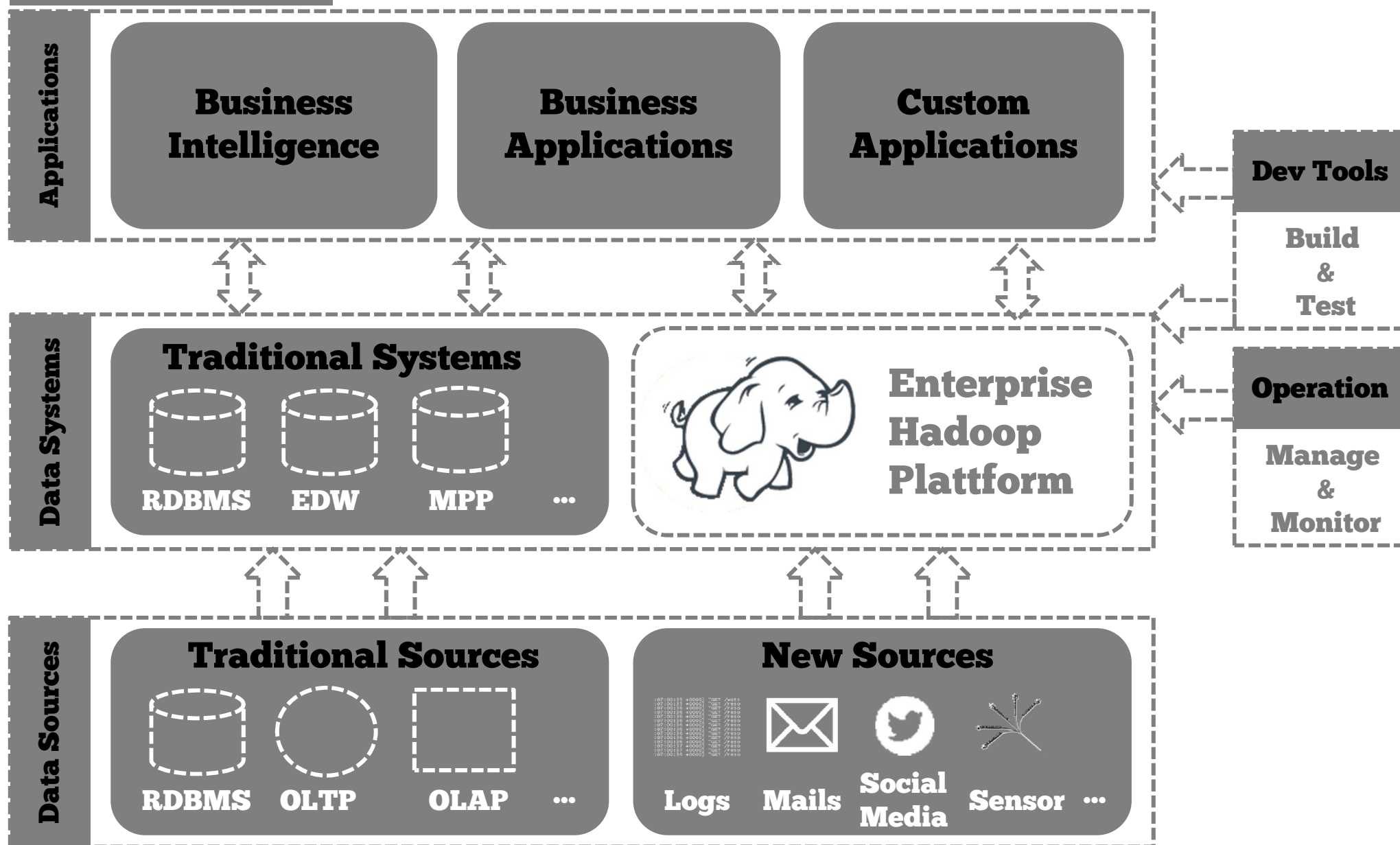


Classical enterprise platform

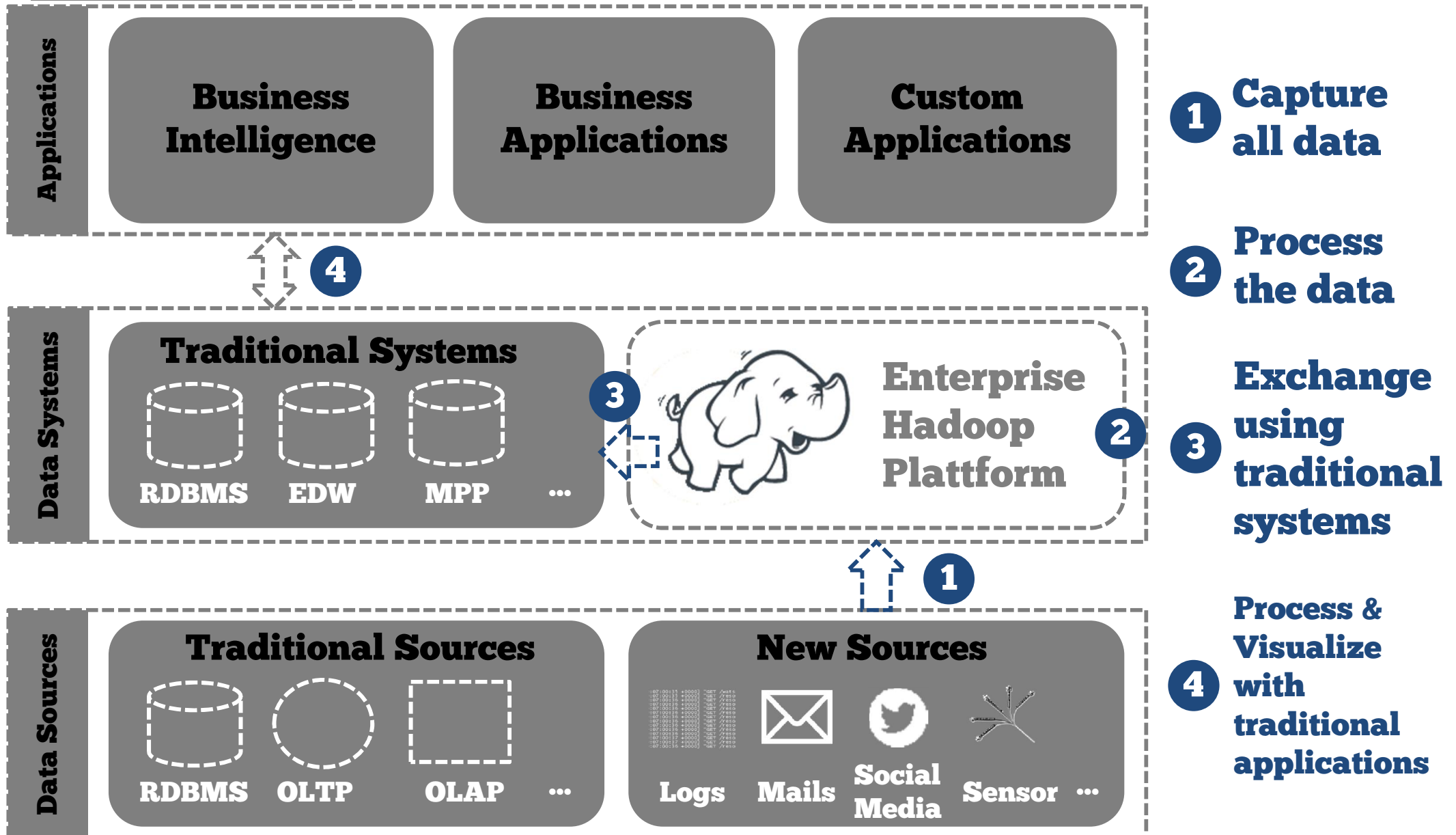




Big Data Platform

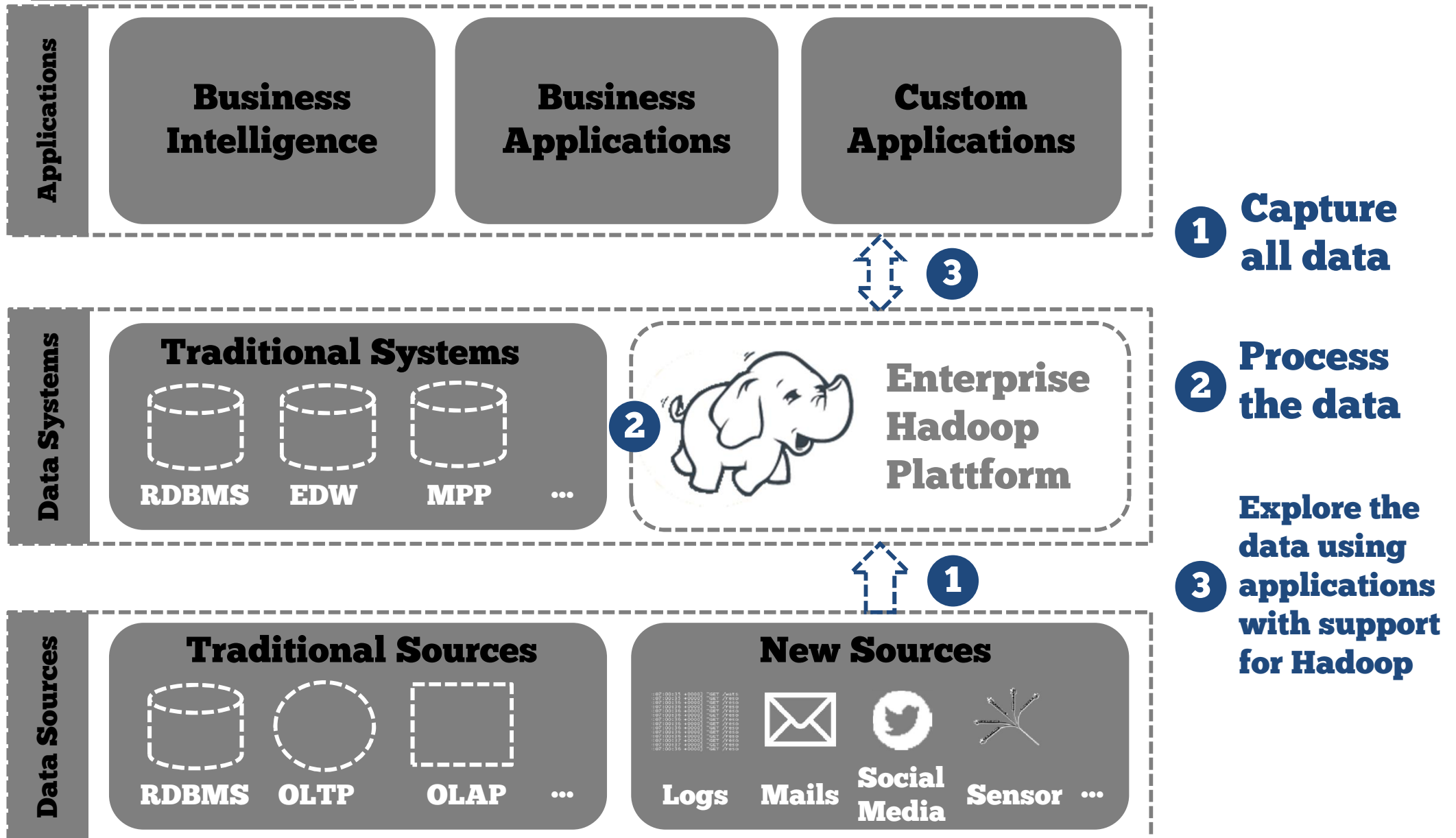


Pattern #1: Refine data

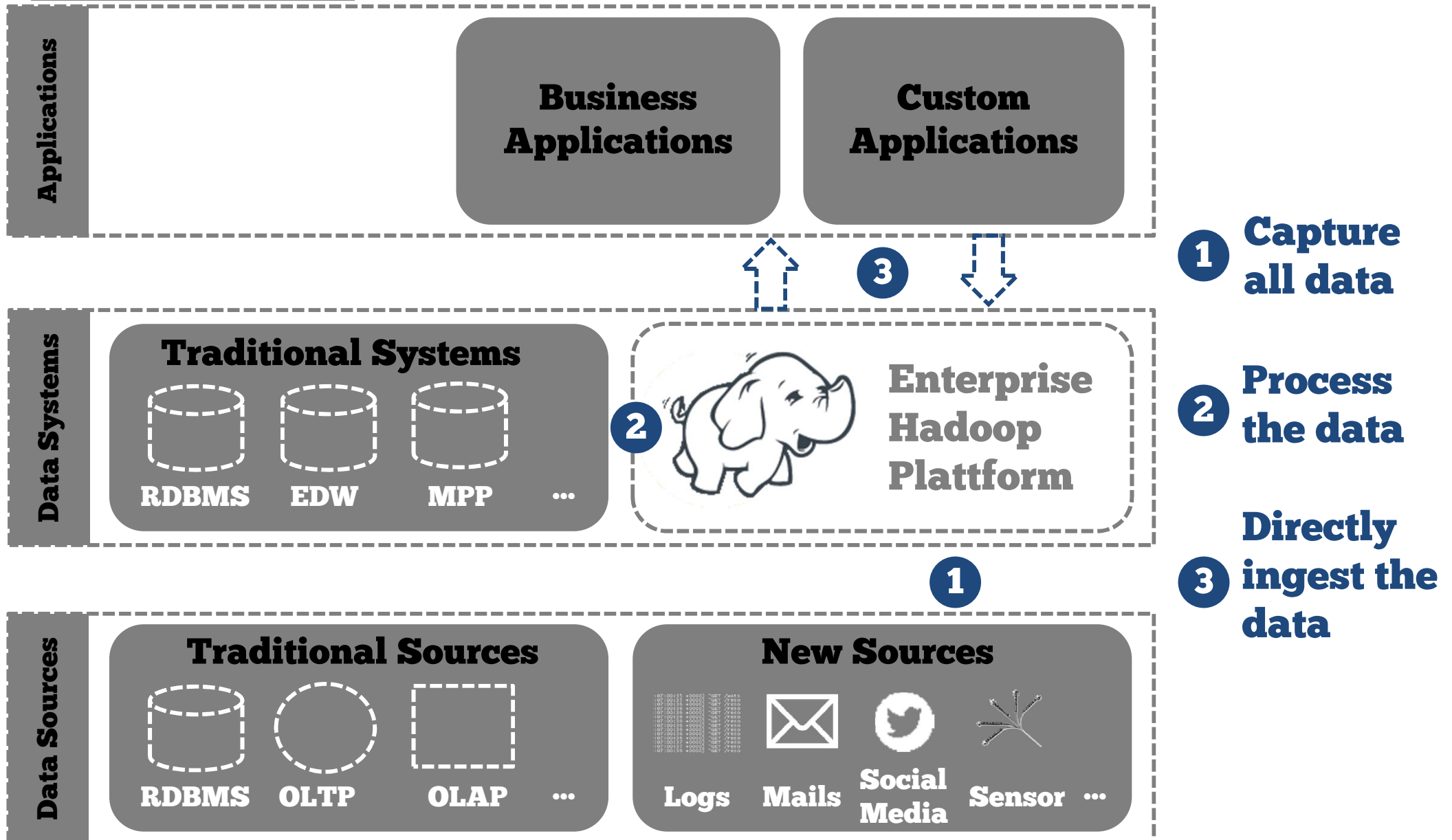




Pattern #2: Explore data



Pattern #3: Enrich data



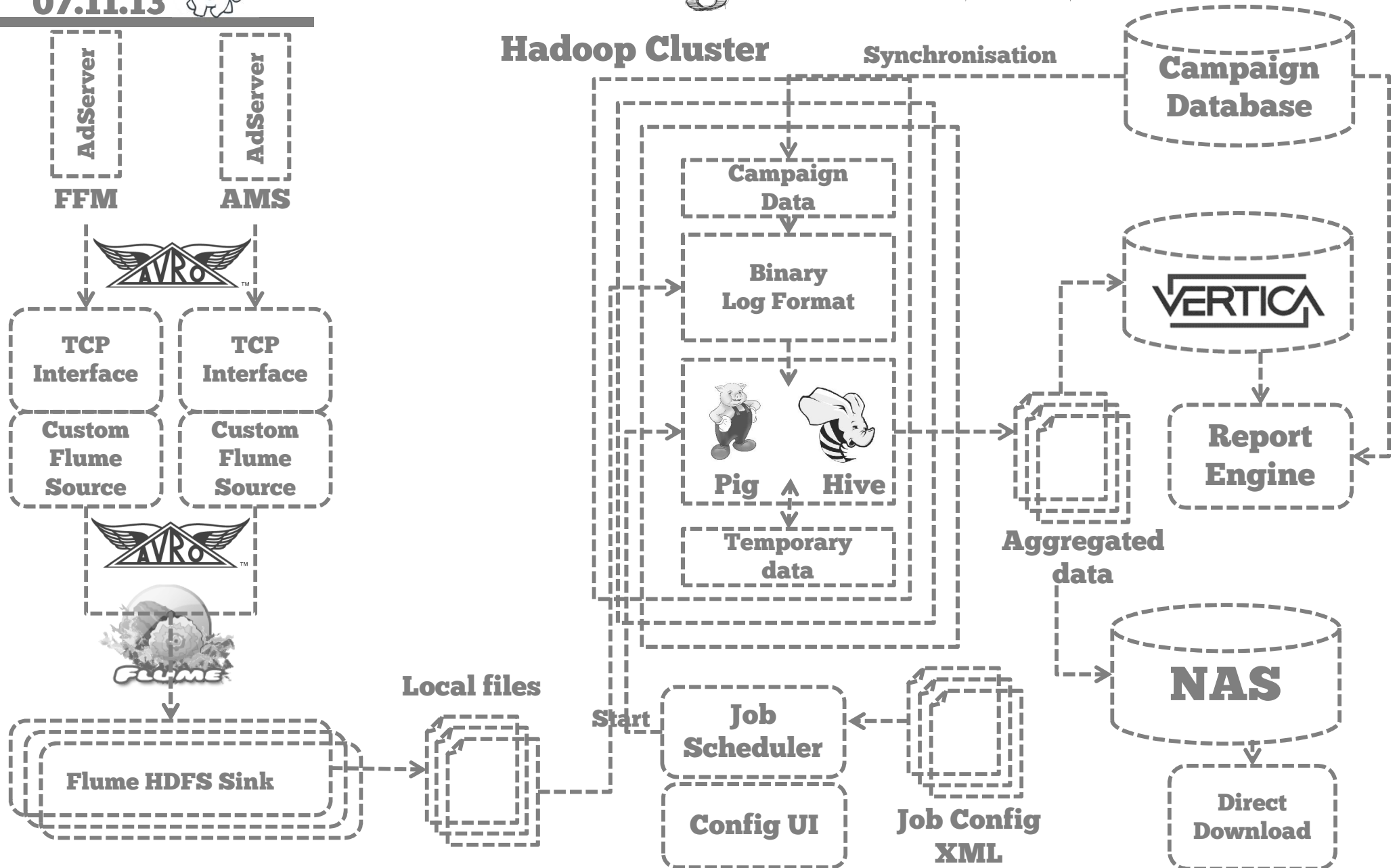
Bringing it all together...

One example...

- **6 billion ad deliveries per day**
- **Reports (and bills) for the advertising companies needed**
- **Own C++ solution did not scale**
- **Adding functions was a nightmare**



AdServing Architecture

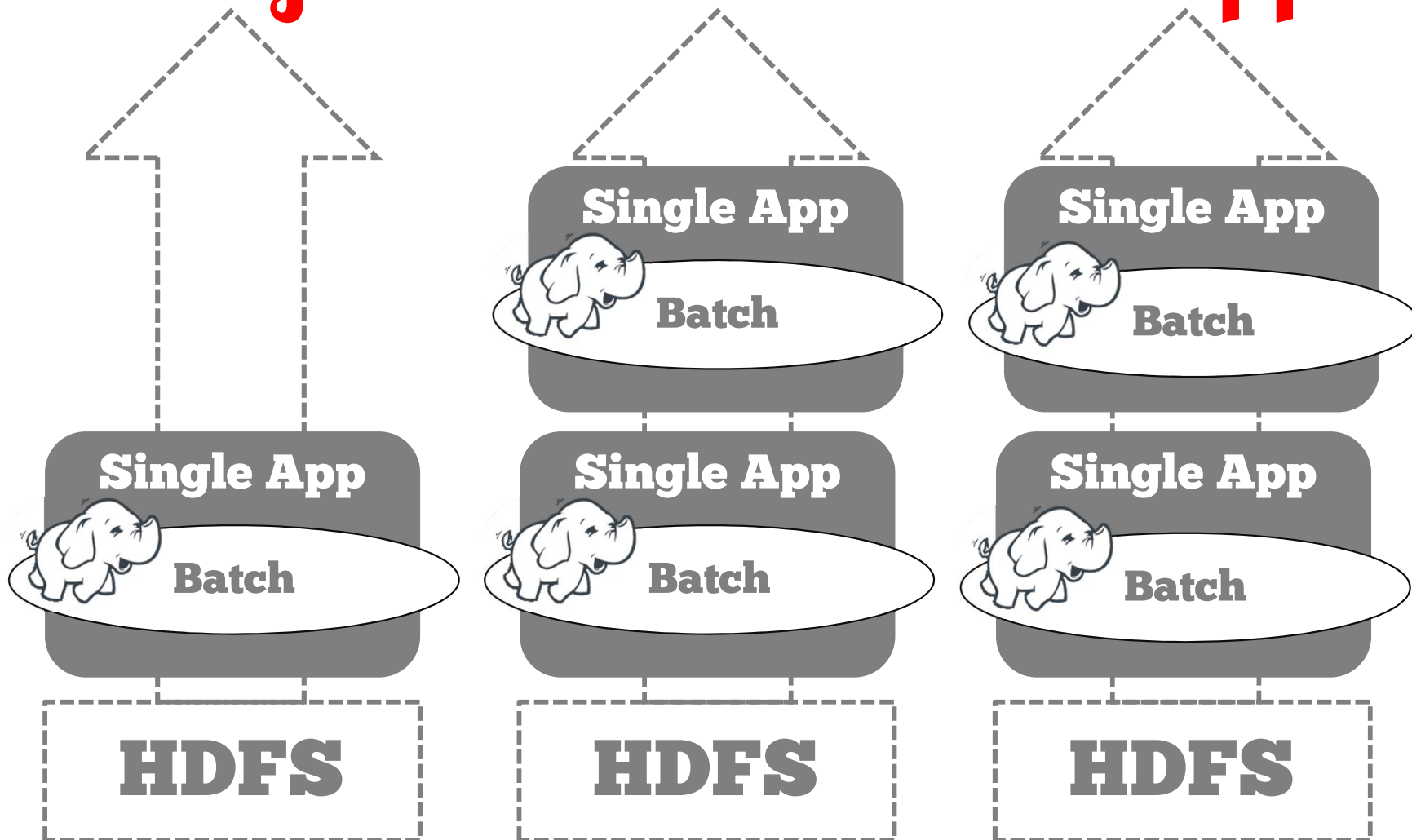


What's next?

Hadoop 2.0 aka YARN



Built for web-scale batch apps



MapReduce is good for...

- **Embarrassingly parallel algorithms**
- **Summing, grouping, filtering, joining**
- **Off-line batch jobs on massive data sets**
- **Analyzing an entire large dataset**

MapReduce is OK for...

- **Iterative jobs (i.e., graph algorithms)**
 - **Each iteration must read/write data to disk**
 - **I/O and latency cost of an iteration is high**

MapReduce is not good for...

- **Jobs that need shared state/coordination**
 - **Tasks are shared-nothing**
 - **Shared-state requires scalable state store**
- **Low-latency jobs**
- **Jobs on small datasets**
- **Finding individual records**

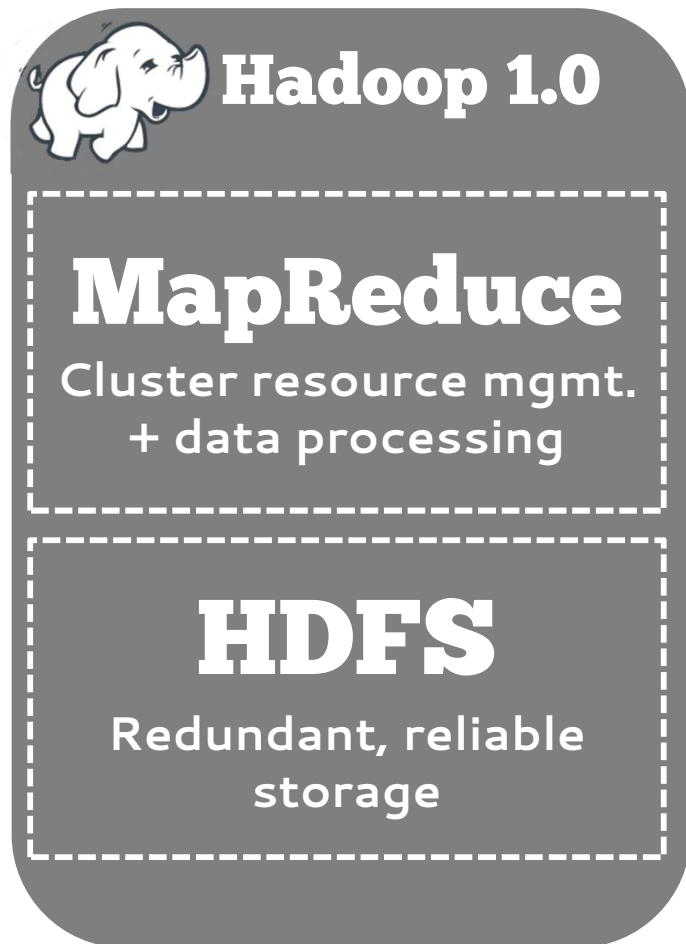
MapReduce limitations

- **Scalability**
 - **Maximum cluster size ~ 4,500 nodes**
 - **Maximum concurrent tasks - 40,000**
 - **Coarse synchronization in JobTracker**
- **Availability**
 - **Failure kills all queued and running jobs**
- **Hard partition of resources into map & reduce slots**
 - **Low resource utilization**
- **Lacks support for alternate paradigms and services**
 - **Iterative applications implemented using MapReduce are 10x slower**

Hadoop 2.0: Next-gen platform

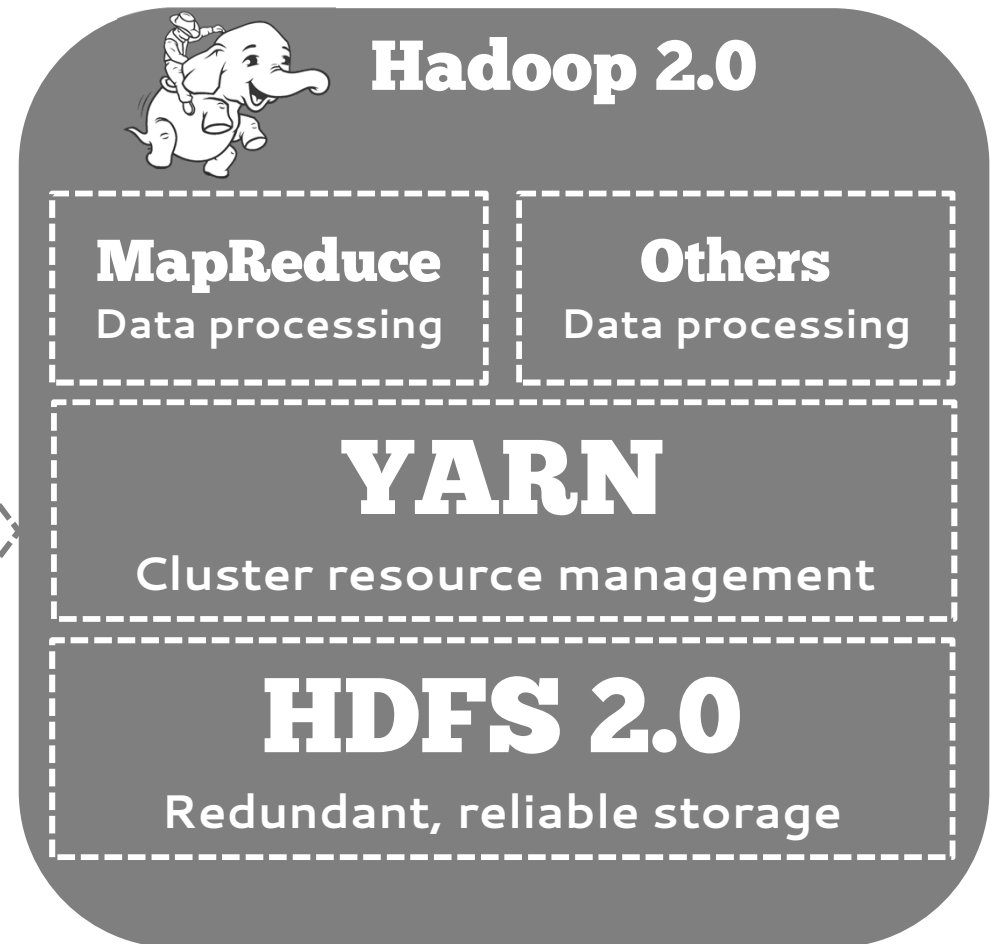
Single use system

Batch Apps



Multi-purpose platform

Batch, Interactive, Streaming, ...



Taking Hadoop beyond batch

Store all data in one place

Interact with data in multiple ways



Applications run natively in Hadoop

Batch MapReduce	Interactive Tez	Online HOYA	Streaming Storm, ...	Graph Giraph	In-Memory Spark	Other Search, ...
---------------------------	---------------------------	-----------------------	--------------------------------	------------------------	---------------------------	-----------------------------

YARN

Cluster resource management

HDFS 2.0

Redundant, reliable storage

A brief history of Hadoop 2.0

- **Originally conceived & architected by the team at Yahoo!**
 - **Arun Murthy created the original JIRA in 2008 and now is the YARN release manager**
- **The team at Hortonworks has been working on YARN for 4 years:**
 - **90% of code from Hortonworks & Yahoo!**
- **Hadoop 2.0 based architecture running at scale at Yahoo!**
 - **Deployed on 35,000 nodes for 6+ months**

GA in October as Hadoop 2.2

Hadoop 2.0 Projects

- **YARN**
- **HDFS Federation aka HDFS 2.0**
- **Stinger & Tez aka Hive 2.0**

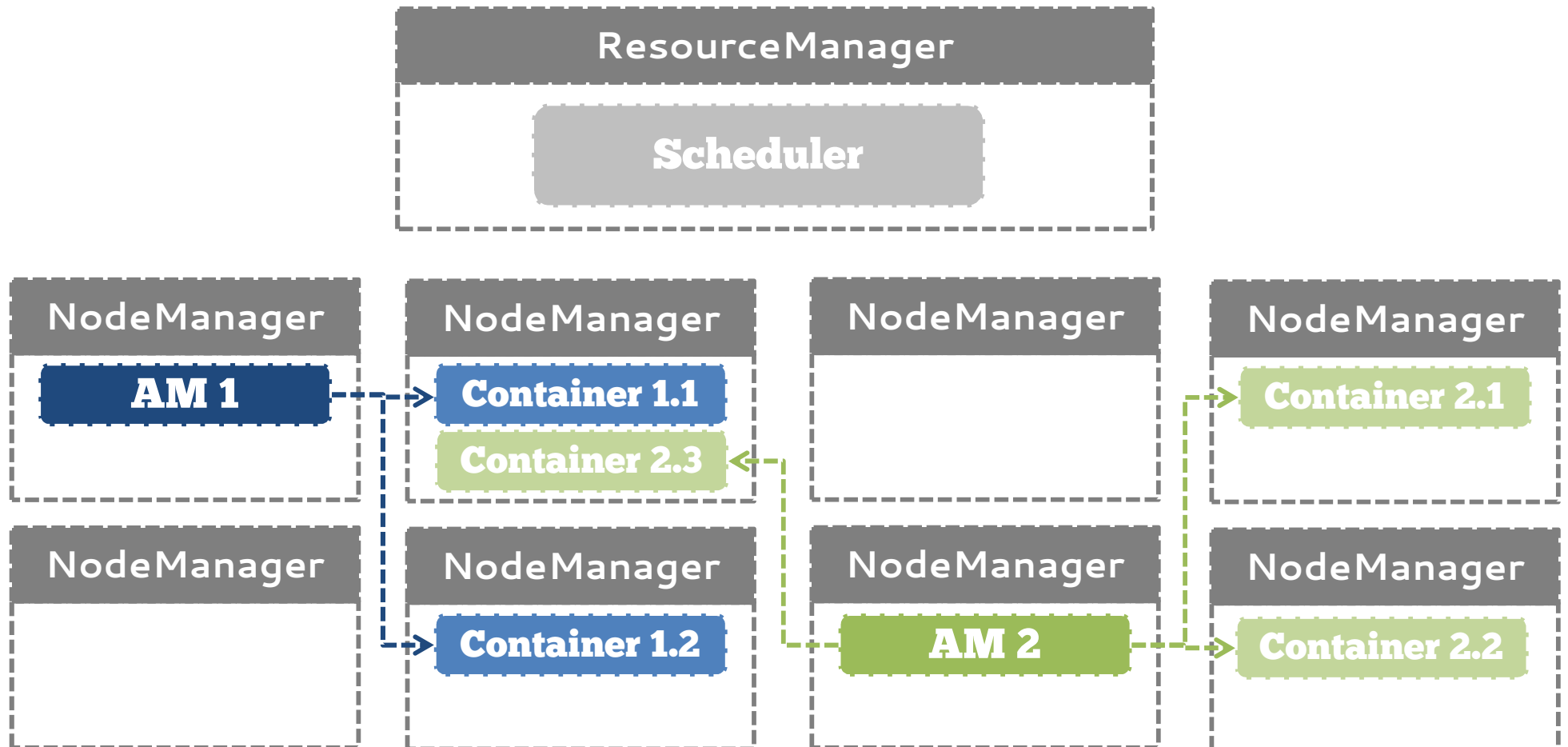
Hadoop 2.0 Projects

- **YARN**
- **HDFS Federation aka HDFS 2.0**
- **Stinger & Tez aka Hive 2.0**

YARN: Architecture

Split up the two major functions of the JobTracker

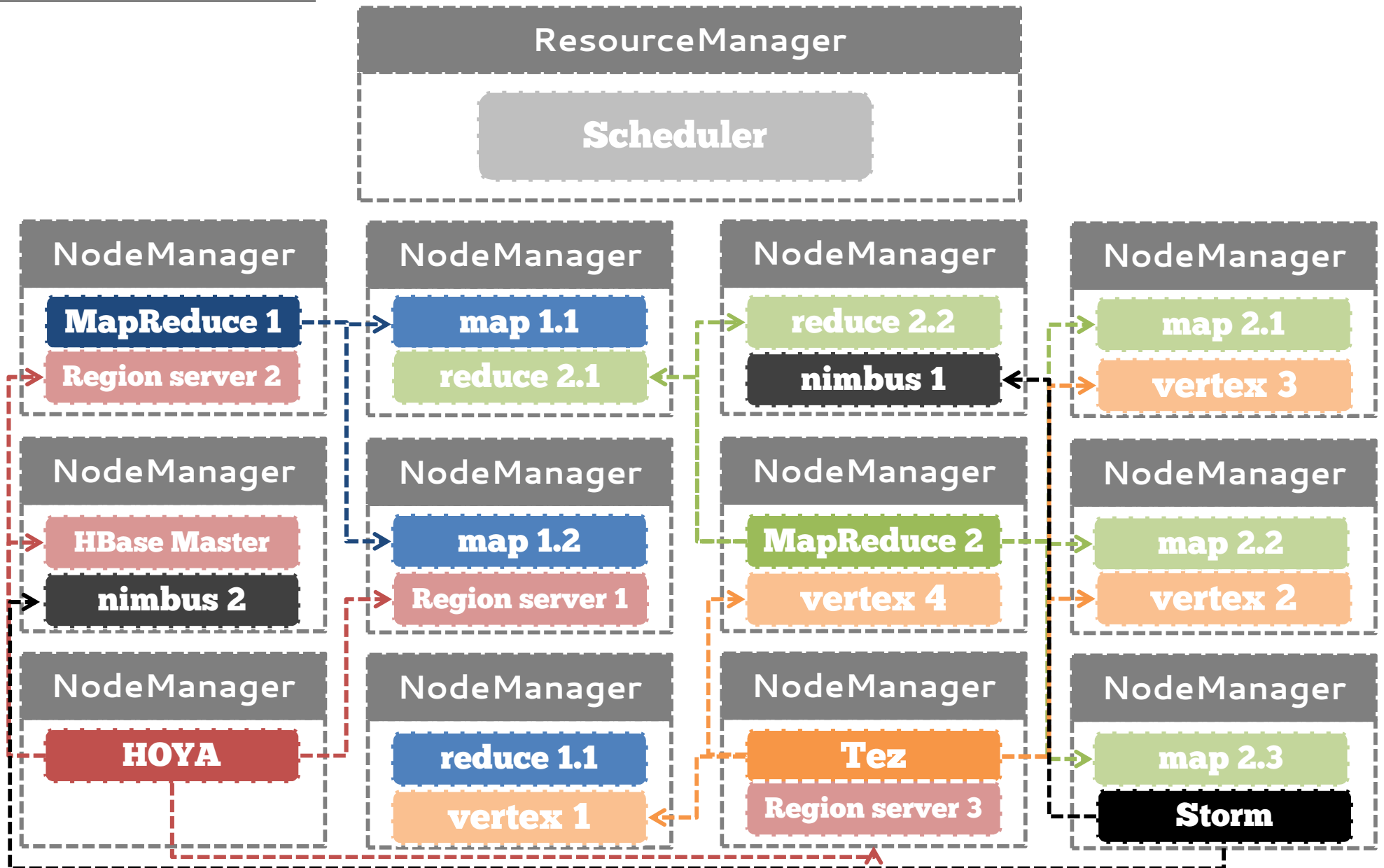
Cluster resource management & Application life-cycle management



YARN: Architecture

- **Resource Manager**
 - Global resource scheduler
 - Hierarchical queues
- **Node Manager**
 - Per-machine agent
 - Manages the life-cycle of container
 - Container resource monitoring
- **Application Master**
 - Per-application
 - Manages application scheduling and task execution
 - e.g. MapReduce Application Master

YARN: Architecture



Hadoop 2.0 Projects

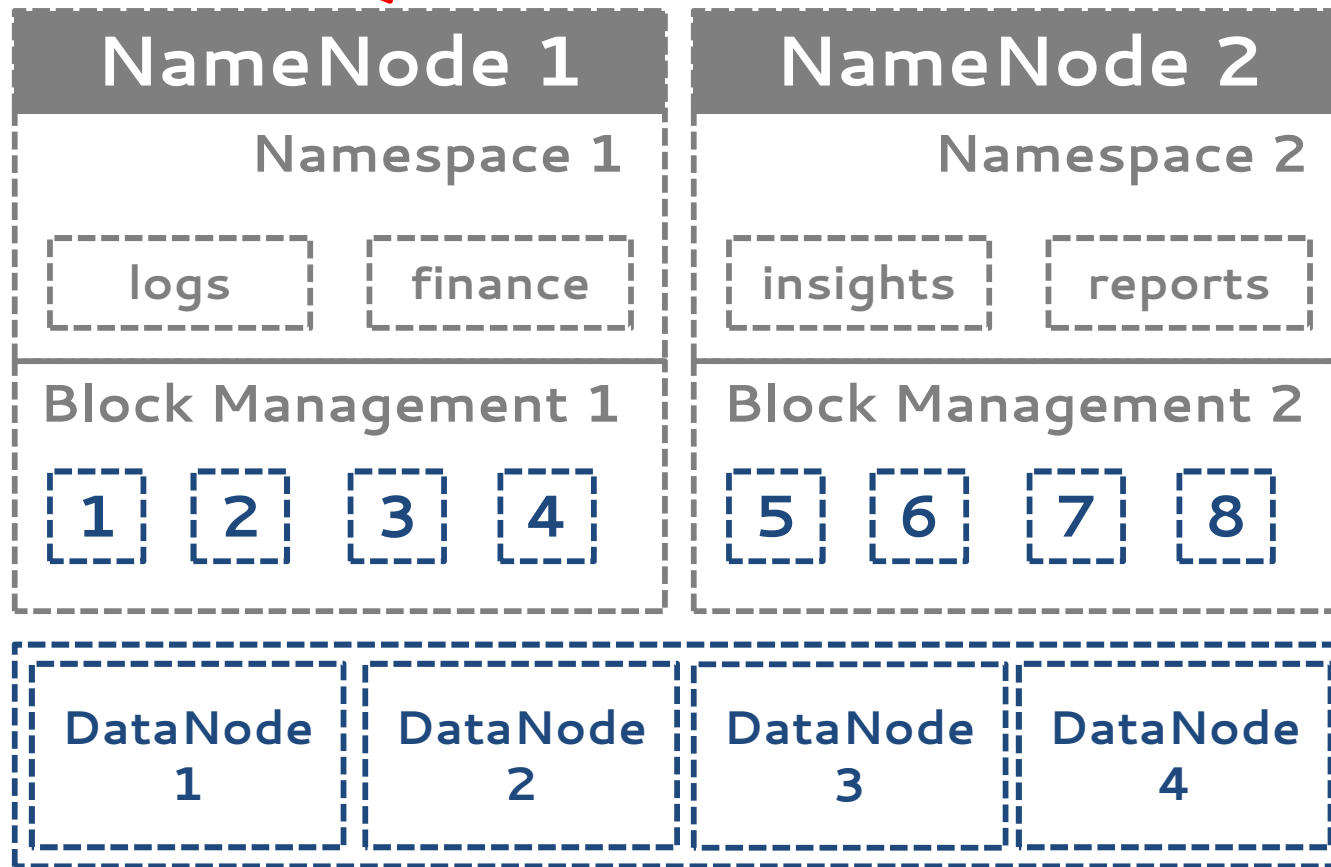
- **YARN**
- **HDFS Federation aka HDFS 2.0**
- **Stinger & Tez aka Hive 2.0**

- **Removes tight coupling of Block Storage and Namespace**
- **Scalability & Isolation**
- **High Availability**
- **Increased performance**

Details: <https://issues.apache.org/jira/browse/HDFS-1052>

HDFS Federation: Architecture

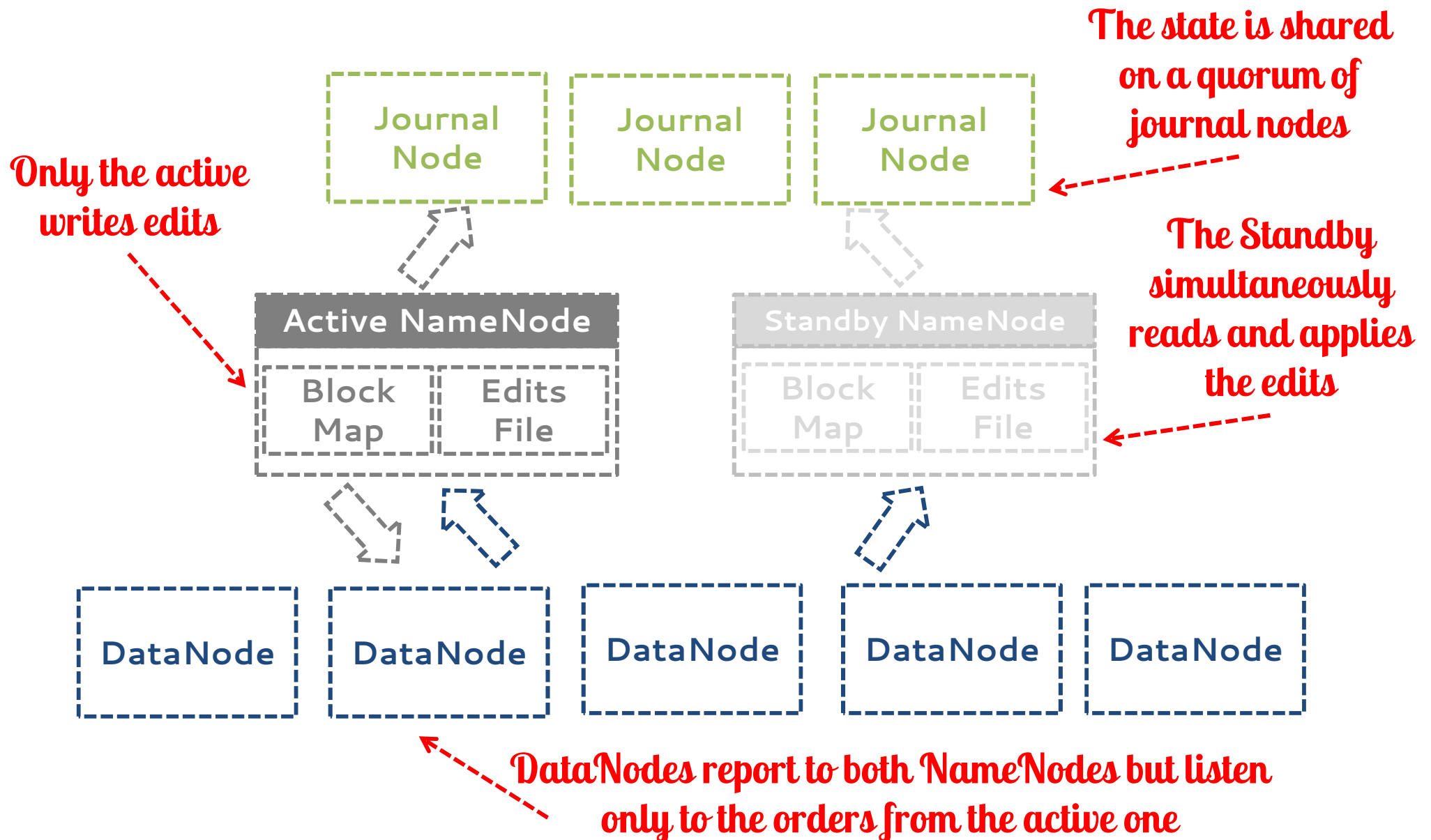
NameNodes do not talk to each other



NameNodes manages only slice of namespace

DataNodes can store blocks managed by any NameNode

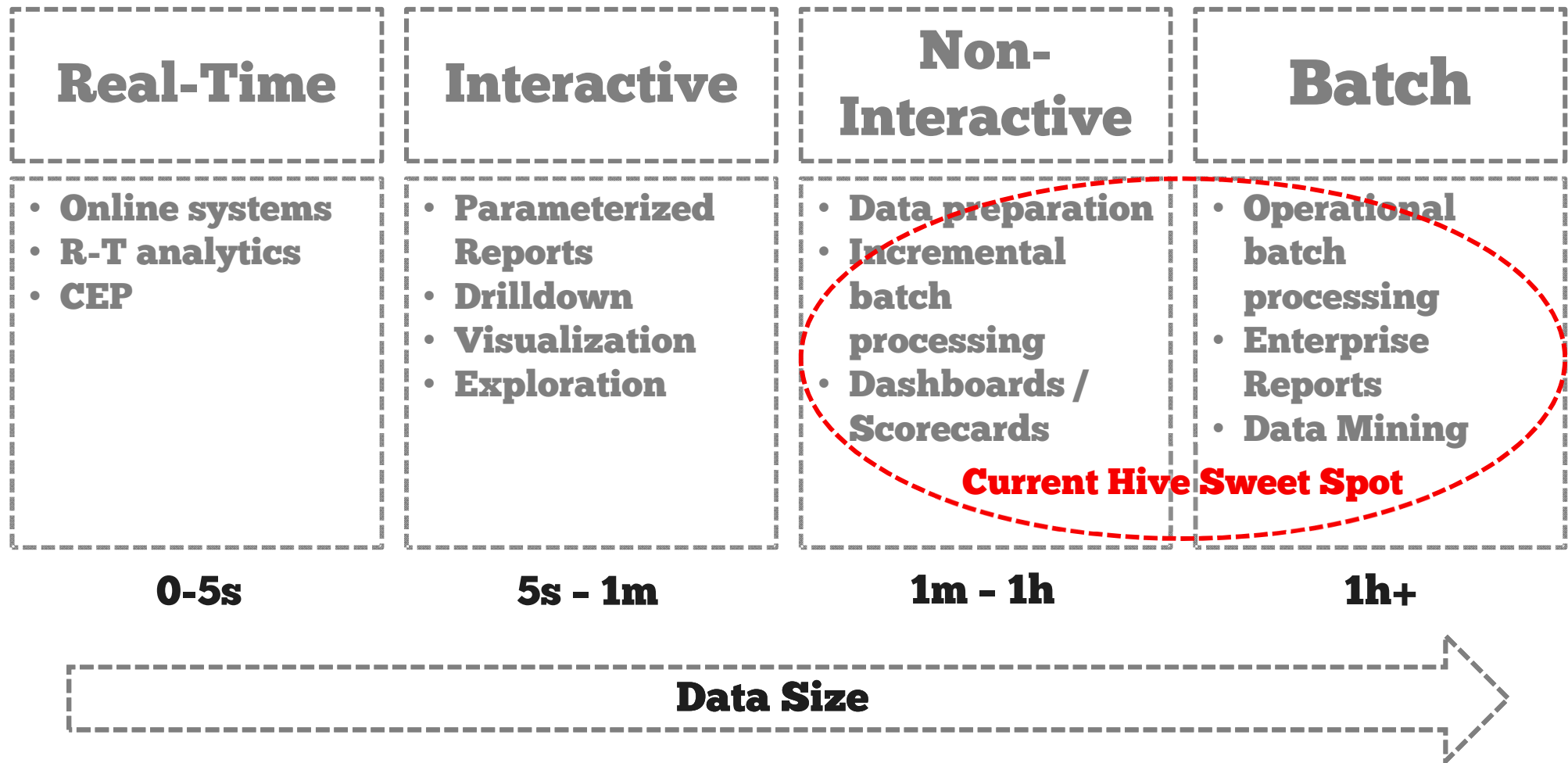
HDFS: Quorum based storage



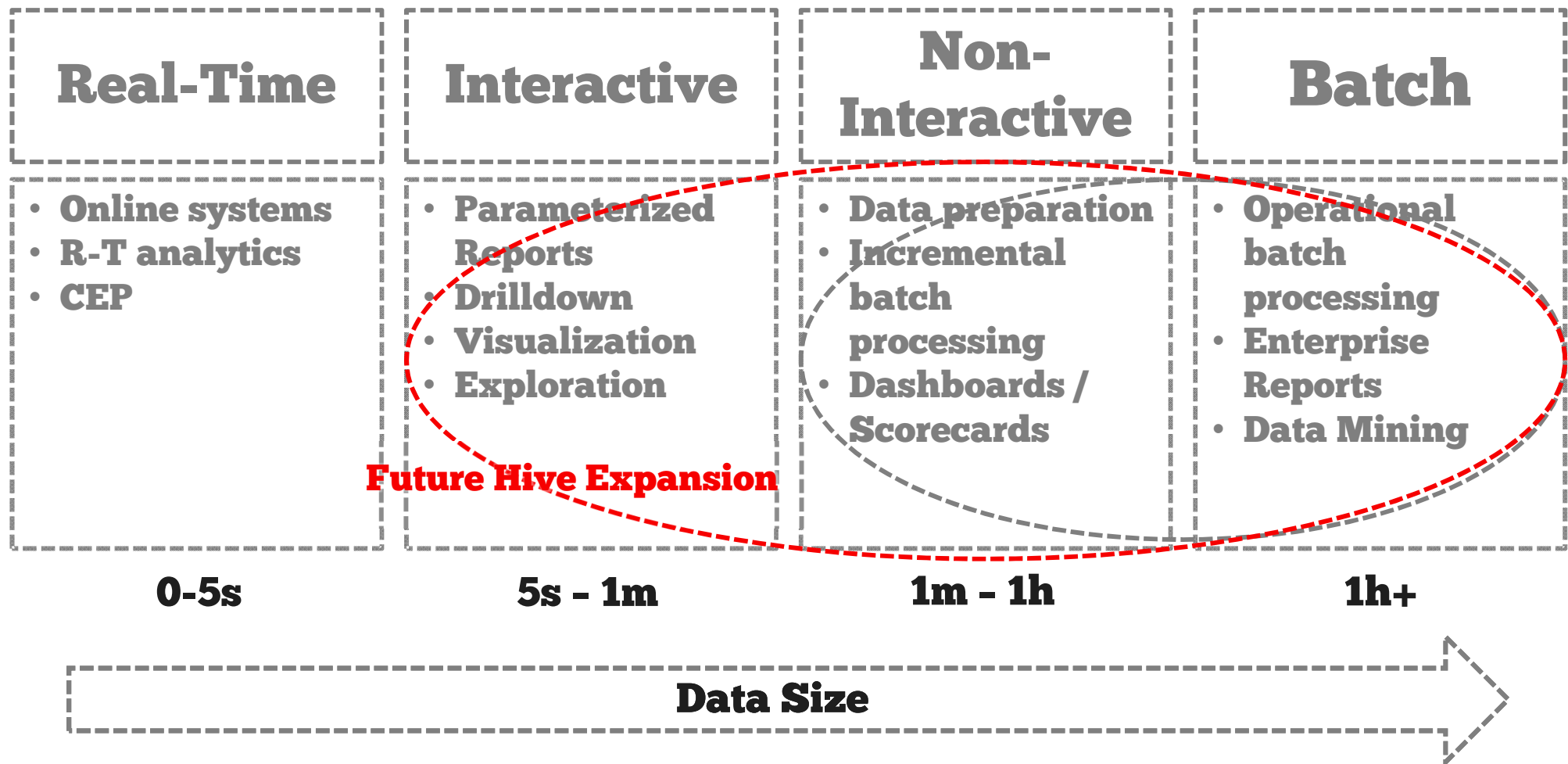
Hadoop 2.0 Projects

- **YARN**
- **HDFS Federation aka HDFS 2.0**
- **Stinger & Tez aka Hive 2.0**

Hive: Current Focus Area



Stinger: Extending the sweet spot



Improve Latency & Throughput

- Query engine improvements
- New "Optimized RCFile" column store
- Next-gen runtime (elim's M/R latency)

Extend Deep Analytical Ability

- Analytics functions
- Improved SQL coverage
- Continued focus on core Hive use cases

Stinger Initiative at a glance

The Stinger Initiative Making Apache Hive 100x Faster

Base Optimizations

Generate simplified DAGs
In-memory Hash Joins

Deep Analytics

SQL Compatible Types
SQL Compatible Windowing
More SQL Subqueries

Vector Query Engine

Optimized for modern
processor architectures

Query Planner

Intelligent Cost-Based
Optimizer

Hive Service

Pre-warmed Containers
Low-latency dispatch

Buffer Caching

Cache accessed data
Optimized for vector engine

Tez

Express data processing
tasks more simply
Eliminate disk writes

ORCFile

Column Store
High Compression
Predicate / Filter Pushdowns

YARN

Next-gen Hadoop data
processing framework

Hadoop

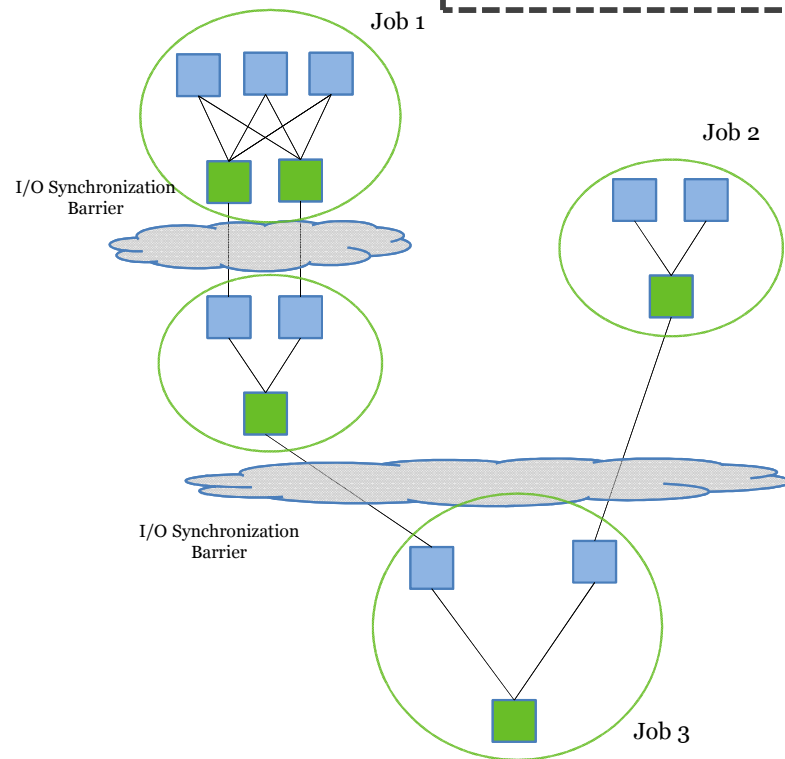
-  Phase 1
-  Phase 2
-  Phase 3

Tez: The Execution Engine

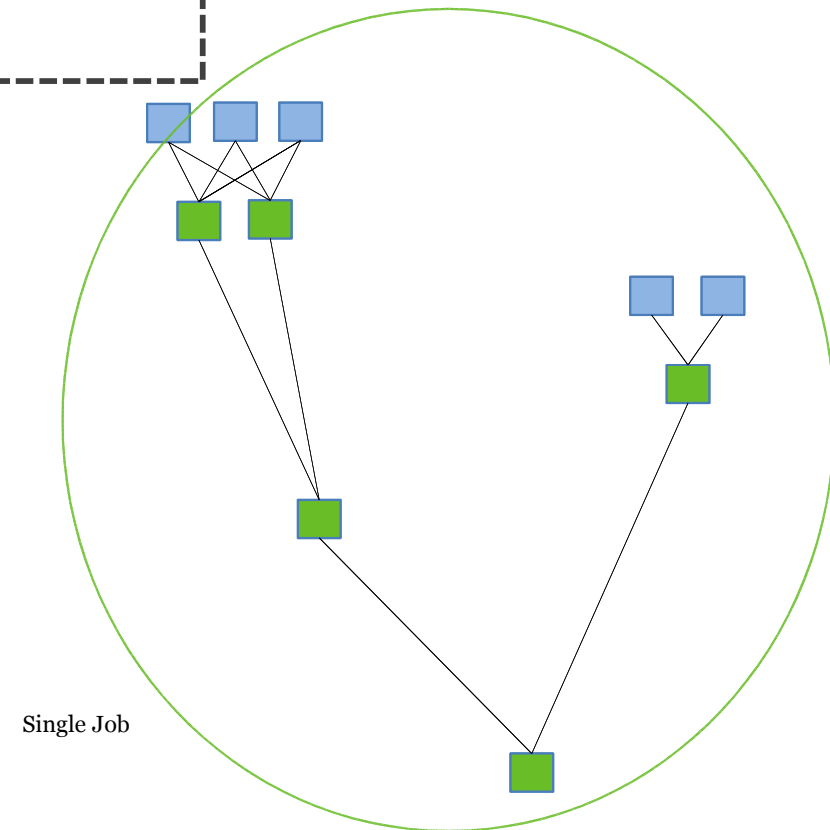
- **Low level data-processing **execution engine****
- **Use it for the base of MapReduce, Hive, Pig, etc.**
- **Enables pipelining of jobs**
- **Removes task and job launch times**
- **Hive and Pig jobs no longer need to move to the end of the queue between steps in the pipeline**
- **Does not write intermediate output to HDFS**
 - **Much lighter disk and network usage**
- **Built on YARN**

Pig/Hive MR vs. Pig/Hive Tez

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



Pig/Hive - MR



Pig/Hive - Tez

- **MapReduce Query Startup is expensive:**
 - Job launch & task-launch latencies are fatal for short queries (in order of 5s to 30s)
- **Solution:**
 - Tez Service (= Preallocated Application Master)
 - Removes job-launch overhead (Application Master)
 - Removes task-launch overhead (Pre-warmed Containers)
 - Hive/Pig
 - Submit query-plan to Tez Service
 - Native Hadoop service, not ad-hoc

Tez: Low latency

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

Existing Hive	
Parse Query	0.5s
Create Plan	0.5s
Launch Map-Reduce	20s
Process Map-Reduce	10s
Total	31s

Hive/Tez	
Parse Query	0.5s
Create Plan	0.5s
Launch Map-Reduce	20s
Process Map-Reduce	2s
Total	23s

Tez & Tez Service	
Parse Query	0.5s
Create Plan	0.5s
Submit to Tez Service	0.5s
Process Map-Reduce	2s
Total	3.5s

*** No exact numbers, for illustration only**

Stinger: Summary

File Type	Number of MR Jobs	Input Size	Mappers	Time
Text/Hive 10	5	43.1 GB	179	21 minutes
Text/Hive 11	1	38 GB	151	246 seconds
RC/Hive 11	1	8.21 GB	76	136 seconds
ORC/Hive 11	1	2.83 GB	38	104 seconds
RC/Hive 11/ Partitioned/ Bucketed	1	1.73 GB	19	104 seconds
ORC/Hive 11/ Partitioned/ Bucketed	1	687 MB	27	79.62

*** Real numbers, but handle with care!**

Hadoop 2.0 Applications

- **MapReduce 2.0**
- **HOYA - HBase on YARN**
- **Storm, Spark, Apache S4**
- **Hamster (MPI on Hadoop)**
- **Apache Giraph**
- **Apache Hama**
- **Distributed Shell**
- **Tez**

Hadoop 2.0 Applications

- **MapReduce 2.0**
- **HOYA - HBase on YARN**
- **Storm, Spark, Apache S4**
- **Hamster (MPI on Hadoop)**
- **Apache Giraph**
- **Apache Hama**
- **Distributed Shell**
- **Tez**

MapReduce 2.0

- **Basically a porting to the YARN architecture**
- **MapReduce becomes a user-land library**
- **No need to rewrite MapReduce jobs**
- **Increased scalability & availability**
- **Better cluster utilization**

Hadoop 2.0 Applications

- **MapReduce 2.0**
- **HOYA - HBase on YARN**
- **Storm, Spark, Apache S4**
- **Hamster (MPI on Hadoop)**
- **Apache Giraph**
- **Apache Hama**
- **Distributed Shell**
- **Tez**

HOYA: HBase on YARN

- **Create on-demand HBase clusters**
- **Configure different HBase instances differently**
- **Better isolation**
- **Create (transient) HBase clusters from MapReduce jobs**
- **Elasticity of clusters for analytic / batch workload processing**
- **Better cluster resources utilization**

Hadoop 2.0 Applications

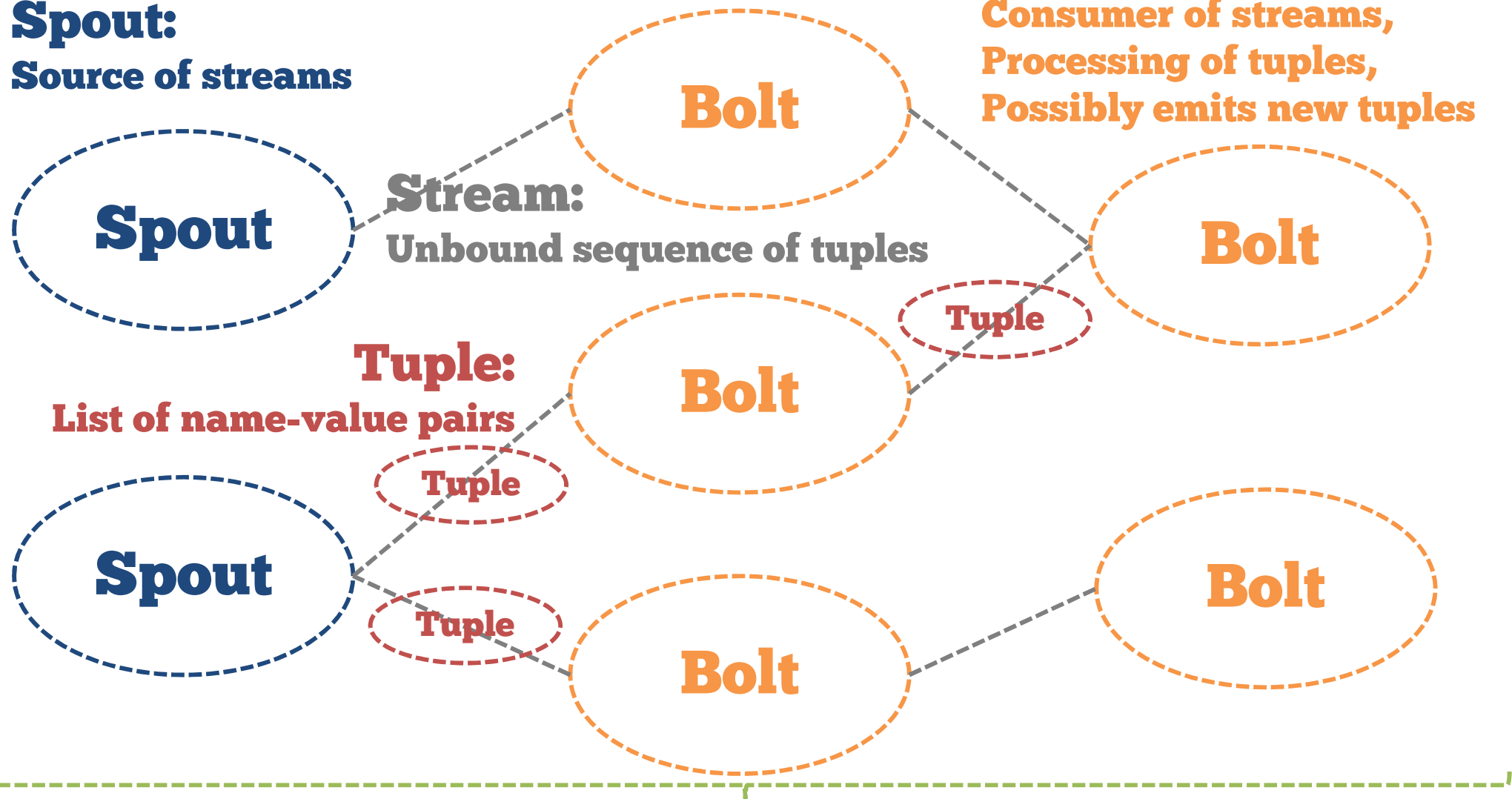
- **MapReduce 2.0**
- **HOYA - HBase on YARN**
- **Storm**, **Spark**, **Apache S4**
- **Hamster (MPI on Hadoop)**
- **Apache Giraph**
- **Apache Hama**
- **Distributed Shell**
- **Tez**

- **Stream-processing**
- **Real-time processing**
- **Developed as standalone application**
 - <https://github.com/nathanmarz/storm>
- **Ported on YARN**
 - <https://github.com/yahoo/storm-yarn>

Storm: Conceptual view

Spout:
Source of streams

Bolt:
Consumer of streams,
Processing of tuples,
Possibly emits new tuples



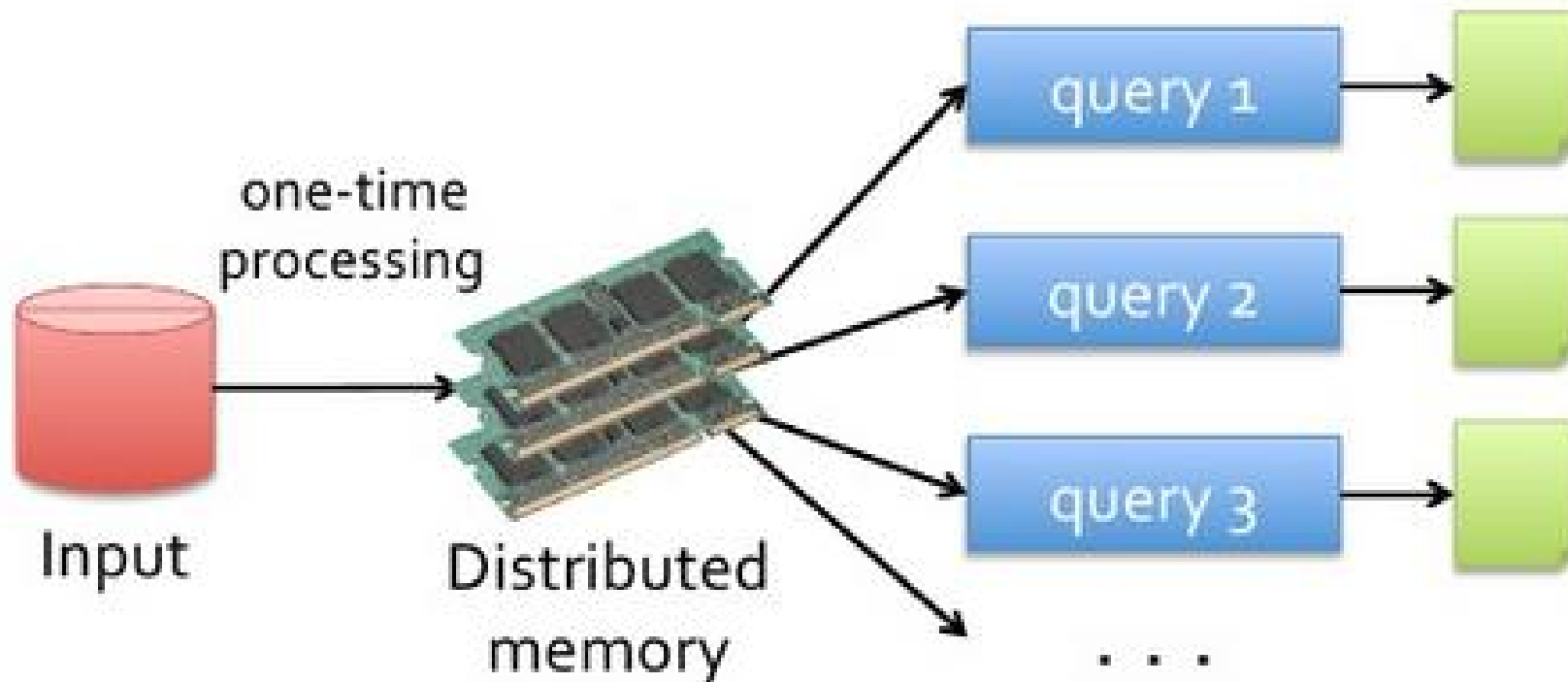
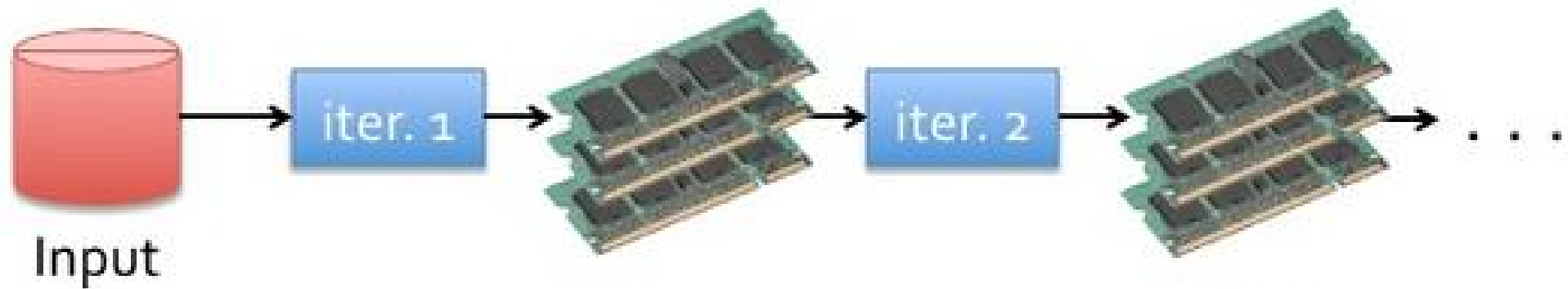
Topology: Network of Spouts & Bolts as the nodes and stream as the edge

Hadoop 2.0 Applications

- **MapReduce 2.0**
- **HOYA - HBase on YARN**
- **Storm, **Spark**, Apache S4**
- **Hamster (MPI on Hadoop)**
- **Apache Giraph**
- **Apache Hama**
- **Distributed Shell**
- **Tez**

- **High-speed **in-memory** analytics over Hadoop and Hive**
- **Separate MapReduce-like engine**
 - **Speedup of up to 100x**
 - **On-disk queries 5-10x faster**
- **Compatible with Hadoop's Storage API**
- **Available as standalone application**
 - <https://github.com/mesos/spark>
- **Experimental support for YARN since 0.6**
 - <http://spark.incubator.apache.org/docs/0.6.0/running-on-yarn.html>

Data Sharing in Spark



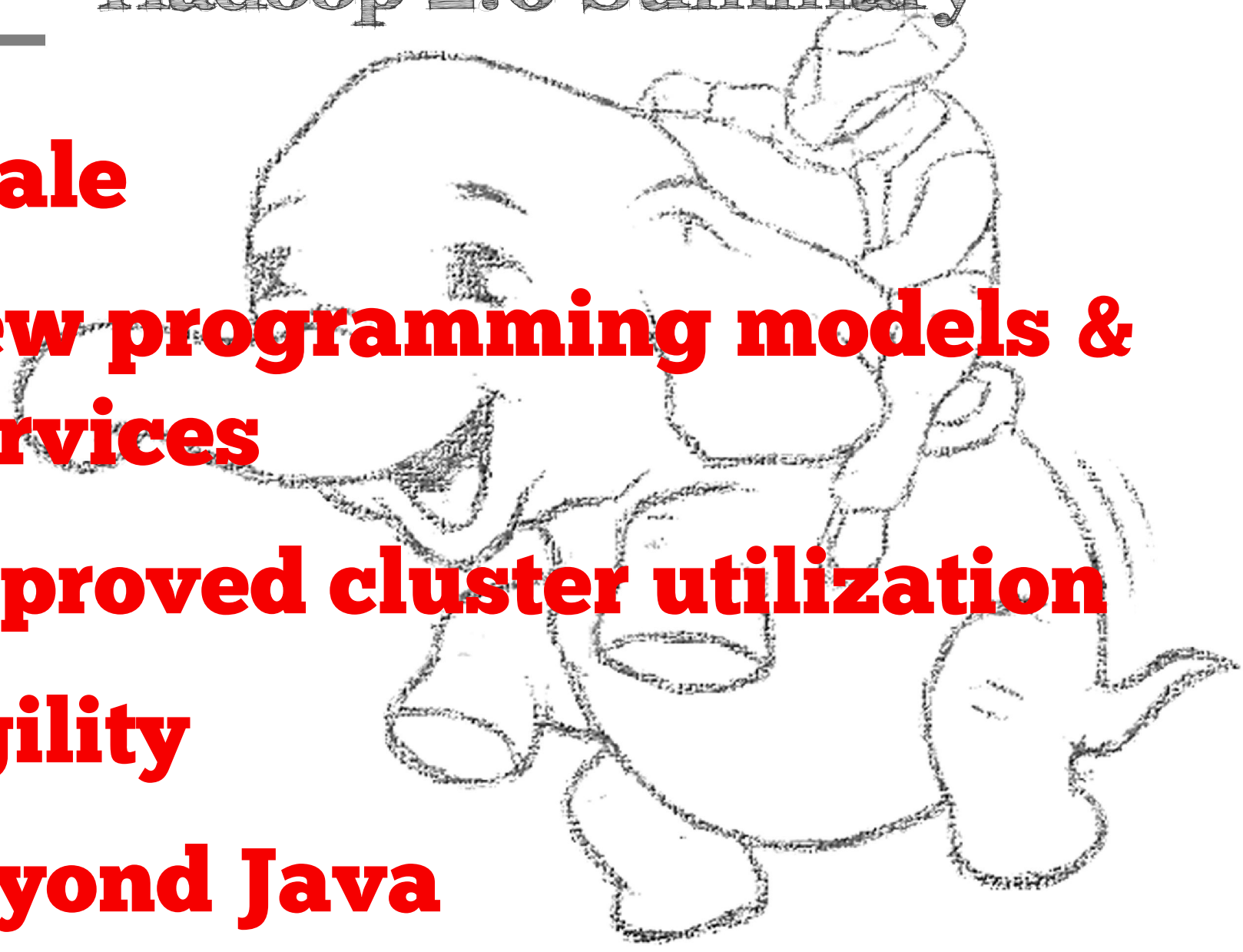
Hadoop 2.0 Applications

- **MapReduce 2.0**
- **HOYA - HBase on YARN**
- **Storm, Spark, Apache S4**
- **Hamster (MPI on Hadoop)**
- **Apache Giraph**
- **Apache Hama**
- **Distributed Shell**
- **Tez**

Apache Giraph

- **Giraph is a framework for processing semi-structured **graph data** on a massive scale.**
- **Giraph is loosely based upon Google's Pregel**
- **Giraph performs **iterative** calculations on top of an existing Hadoop cluster.**
- **Available on GitHub**
 - **<https://github.com/apache/giraph>**

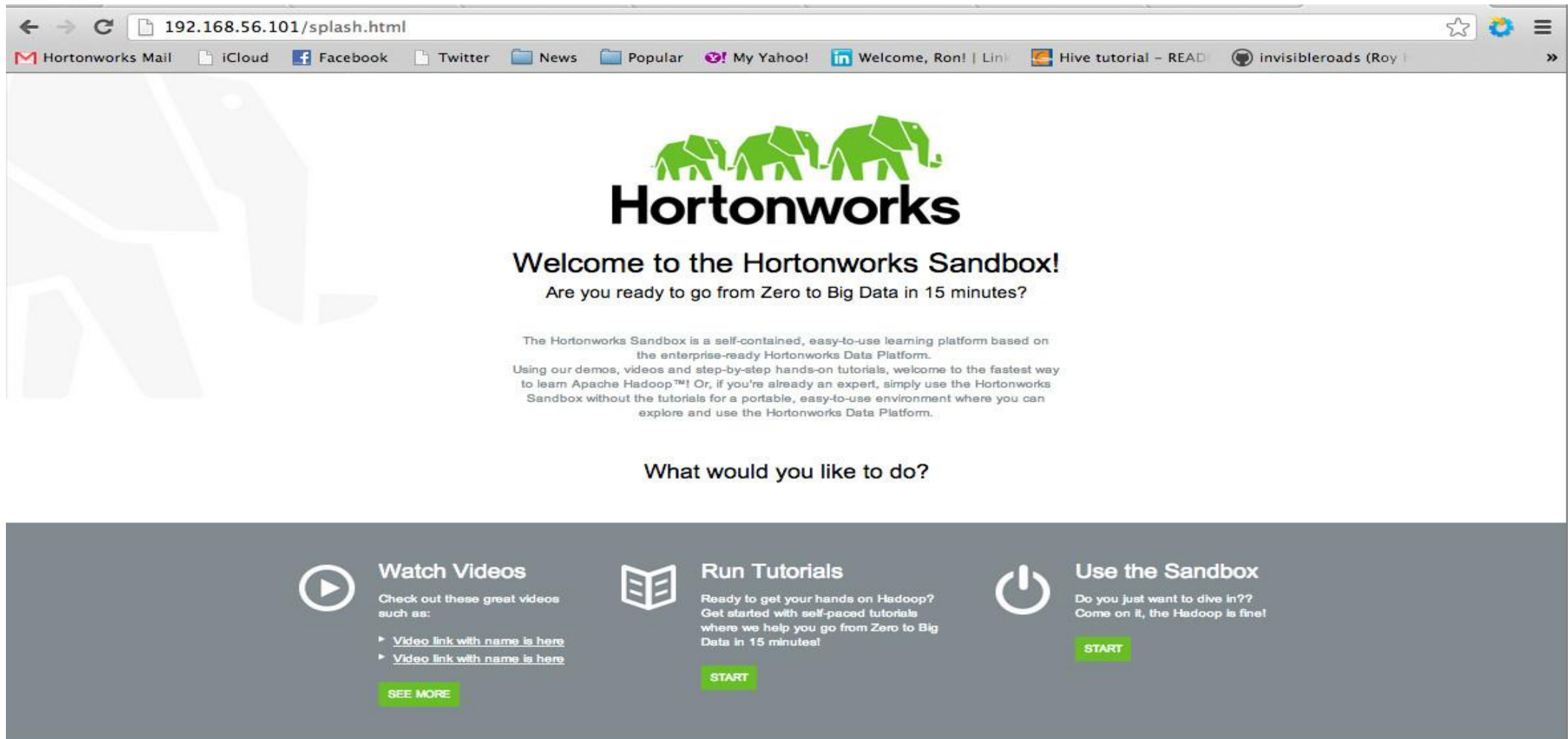
Hadoop 2.0 Summary

- 
- 1. Scale**
 - 2. New programming models & Services**
 - 3. Improved cluster utilization**
 - 4. Agility**
 - 5. Beyond Java**

Getting started...

One more thing...

Hortonworks Sandbox



← → ↺ 192.168.56.101/splash.html ☆ ⚙ ☰

✉ Hortonworks Mail 📁 iCloud 📘 Facebook 📧 Twitter 📰 News 📁 Popular 🎉 My Yahoo! 🔗 Welcome, Ron! | Link 📺 Hive tutorial - READ 🗣 invisibleroads (Roy) >>



Hortonworks

Welcome to the Hortonworks Sandbox!

Are you ready to go from Zero to Big Data in 15 minutes?

The Hortonworks Sandbox is a self-contained, easy-to-use learning platform based on the enterprise-ready Hortonworks Data Platform. Using our demos, videos and step-by-step hands-on tutorials, welcome to the fastest way to learn Apache Hadoop™! Or, if you're already an expert, simply use the Hortonworks Sandbox without the tutorials for a portable, easy-to-use environment where you can explore and use the Hortonworks Data Platform.

What would you like to do?



Watch Videos

Check out these great videos such as:

- ▶ [Video link with name is here](#)
- ▶ [Video link with name is here](#)

[SEE MORE](#)



Run Tutorials

Ready to get your hands on Hadoop? Get started with self-paced tutorials where we help you go from Zero to Big Data in 15 minutes!

[START](#)



Use the Sandbox

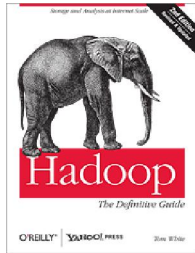
Do you just want to dive in?? Come on it, the Hadoop is fine!

[START](#)

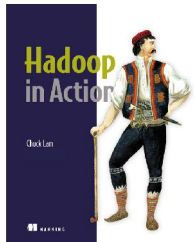
©Hortonworks Inc 2013. All Rights Reserved. Hadoop and the Hadoop elephant logo are trademarks of the Apache Software Foundation.

<http://hortonworks.com/products/hortonworsk-sandbox>

Books about Hadoop



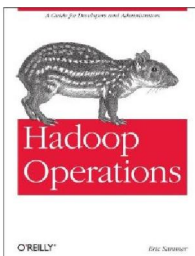
**Hadoop - The Definite Guide, Tom White,
3rd ed., O'Reilly, 2012.**



**Hadoop in Action, Chuck Lam,
Manning, 2011**



**Programming Pig, Alan Gates
O'Reilly, 2011**



**Hadoop Operations, Eric Sammer,
O'Reilly, 2012**

The end...or the beginning?

Questions?

