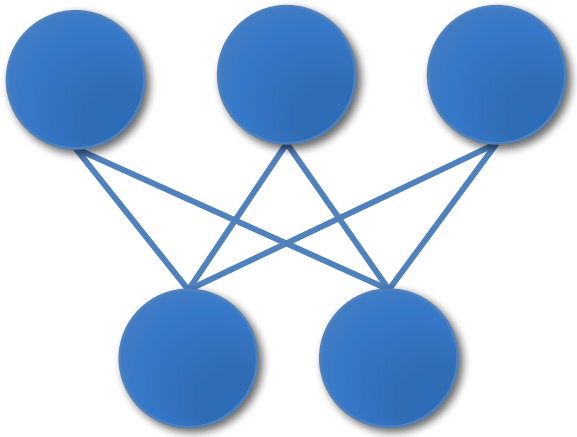
deep se
dependable evolvable pervasive software engineering group



Introduction to Big Data and MapReduce

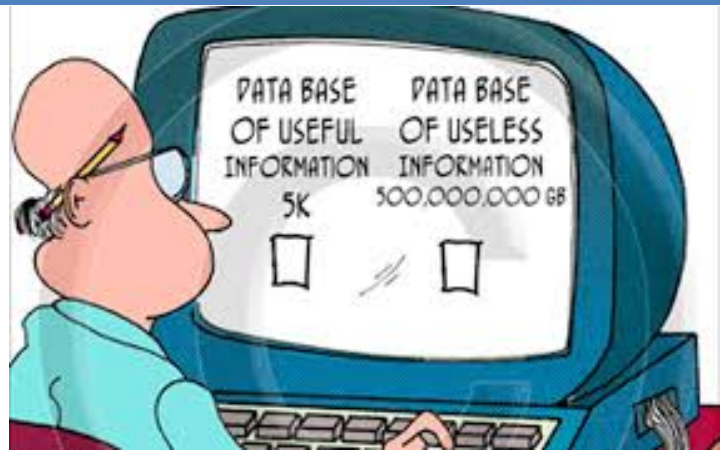
Danilo Ardagna

Politecnico di Milano

danilo.ardagna@polimi.it

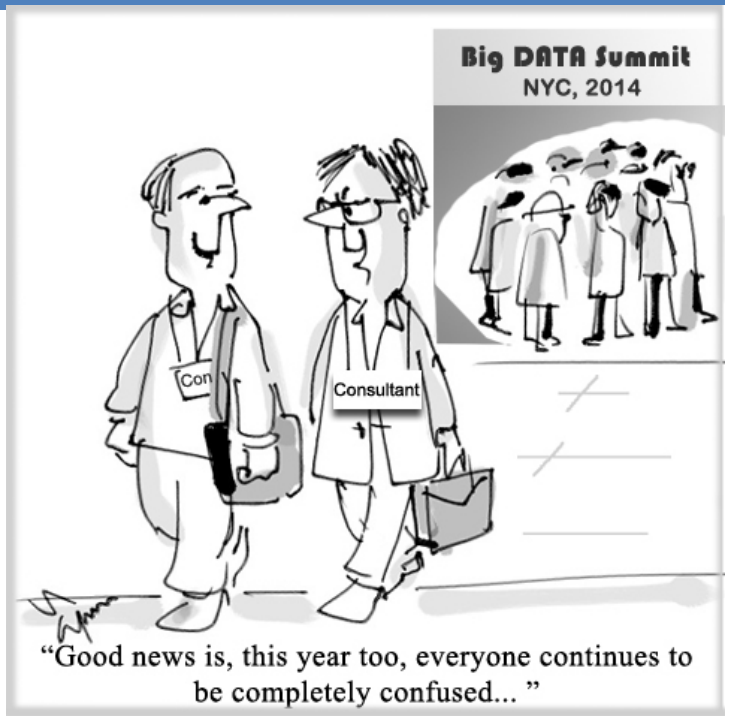
POLITECNICO DI MILANO

Content



- Big Data Analytics and Data Science
- Machine Learning overview
- Map-reduce and Hadoop fundamentals
- Hadoop Evolution

What is Big Data?



- *Big Data is a collection of **very huge data sets** with a great diversity* [Chen & Zhang 2014]

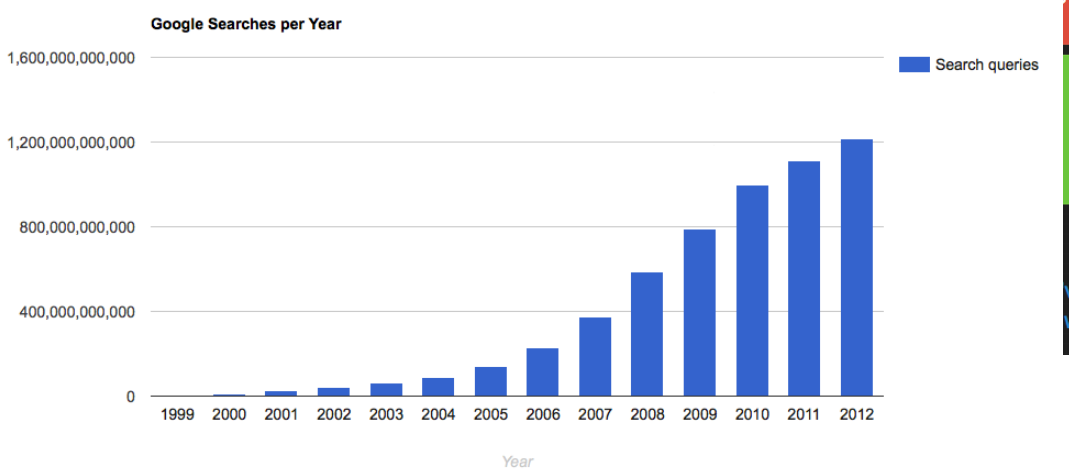
- *Big Data are high-**volume**, high-**velocity**, and/or high-**variety** information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization* [Gartner 2012]

- *Big data should be thought of as a process — how to get to new insights, how to turn them into action, resulting in business **value*** [Gartner 2015]

Big Data Trends

- Software market rapidly shifting to Big Data
  - **32%** compound annual **growth rate** in **EU** through 2016
  - 35% Big data projects are successful [CapGemini 2015]
- Example of use cases:
  - A 360 degree **view** of the **customer**
  - **Internet of Things**
  - Big Data **Predictive** Analytics
- Fast evolving and interesting technologies:
  - MapReduce, Hadoop
  - Spark
  - Streaming systems

Why now?



http://pennystocks.la/internet-in-real-time/
http://www.internetlivestats.com/twitter-statistics/
http://www.internetlivestats.com/google-search-statistics/

Some stats

- The data volumes are exploding, **more data** has been created in the **past two years** than in the entire previous **history**. By 2020:
  - about 1.7 megabytes of new information will be created every second for every human being
  - accumulated data will grow from 4.4 zettabytes today to around 44 zettabytes (44 trillion gigabytes)
- We perform 40,000 search queries every second (on Google alone)
- In Aug 2015, over 1 billion people used Facebook in a single day:
  - FB users send on average 31.25 million messages and view 2.77 million videos every minute
- In 2015, over 1.4 billion smart phones were shipped (sensors!). By 2020: 6.1 billion smartphone users globally

http://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#2ae133b66c1d

Some stats

- By 2020 over **50 billion** smart **connected devices** in the world, all developed to collect, analyze and share data
- Distributed computing:
  - Google uses about **1,000 servers** in answering a **single search query**, which takes no more than 0.2 seconds to complete
- Hadoop market is forecast to grow at a compound annual growth rate 58% surpassing $1 billion by 2020
- Estimates suggest that by better integrating big data, healthcare could save as much as $300 billion a year — that's equal to reducing costs by $1000 a year for every man, woman, and child
- In 2015, less than 0.5% of all data is ever analysed and used

http://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#2ae133b66c1d

Characteristics

Volume

Data at scale

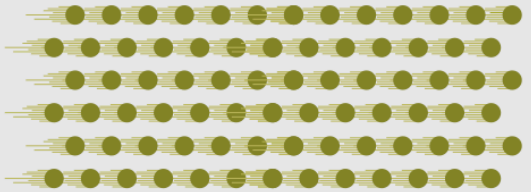
Terabytes to petabytes of data

Variety

Data in many forms

Structured, unstructured, text, multimedia

Velocity



Data in motion

Analysis of streaming data to enable decisions within fractions of a second

Veracity

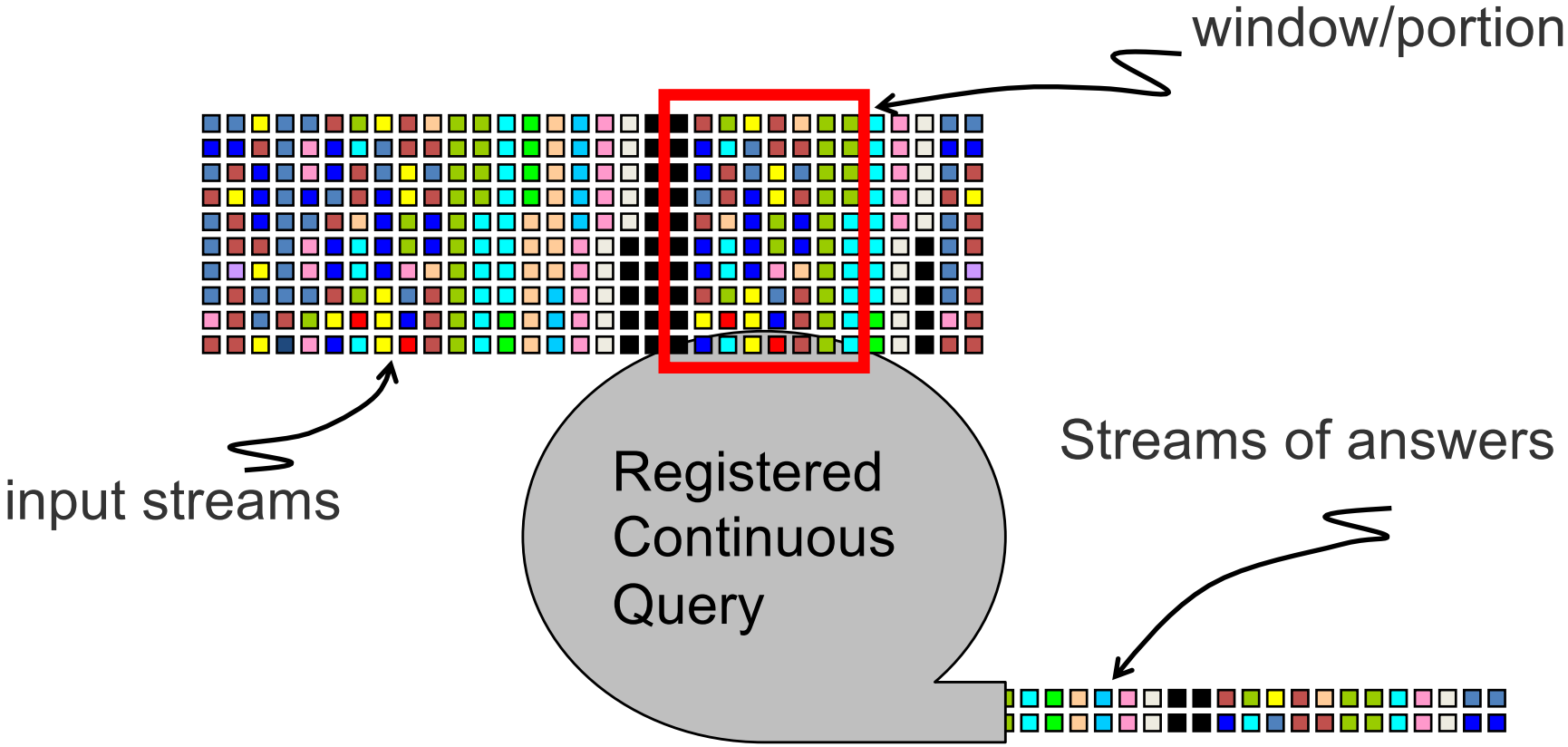
Data uncertainty

Managing the reliability and predictability of inherently imprecise data types

Courtesy of IBM

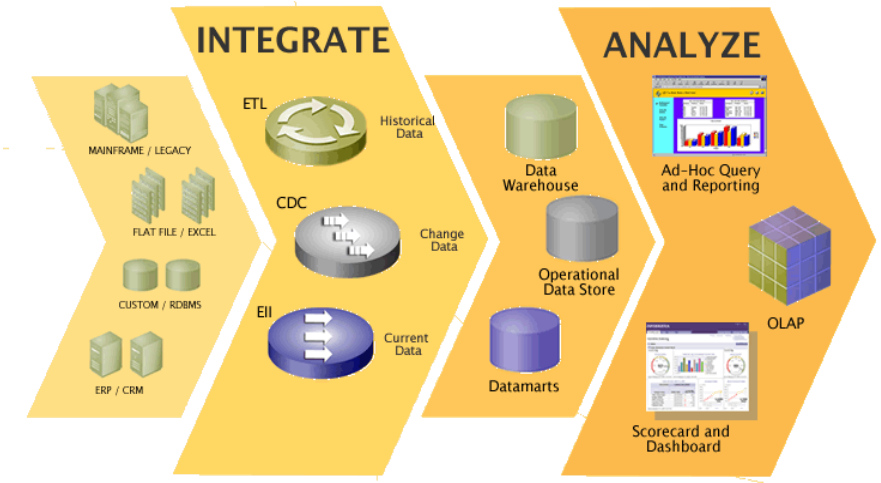
High Velocity

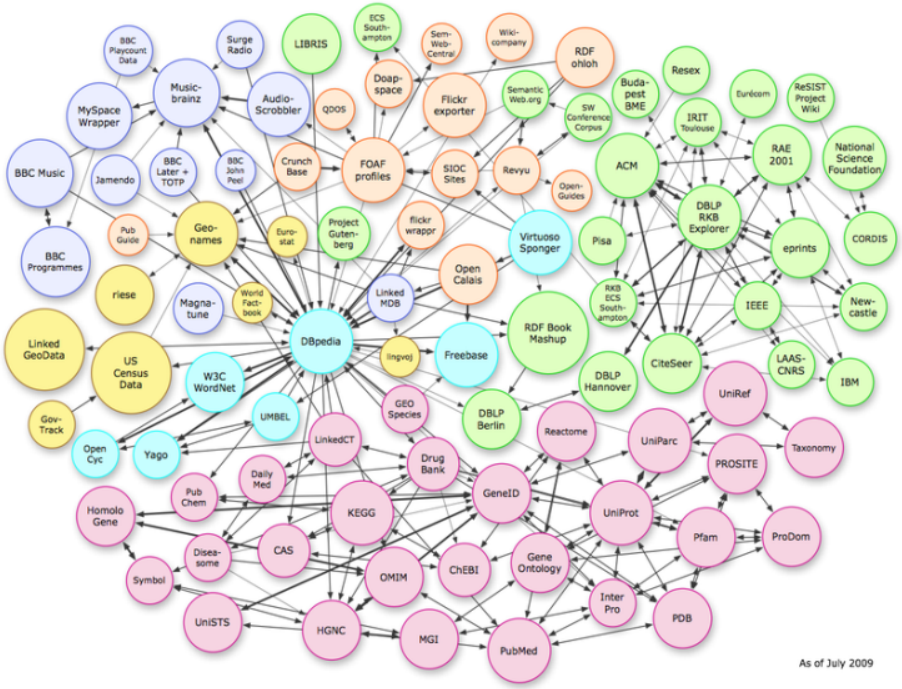
- From persistent data to be queried on demand
- To **transient data** to be consumed on the fly



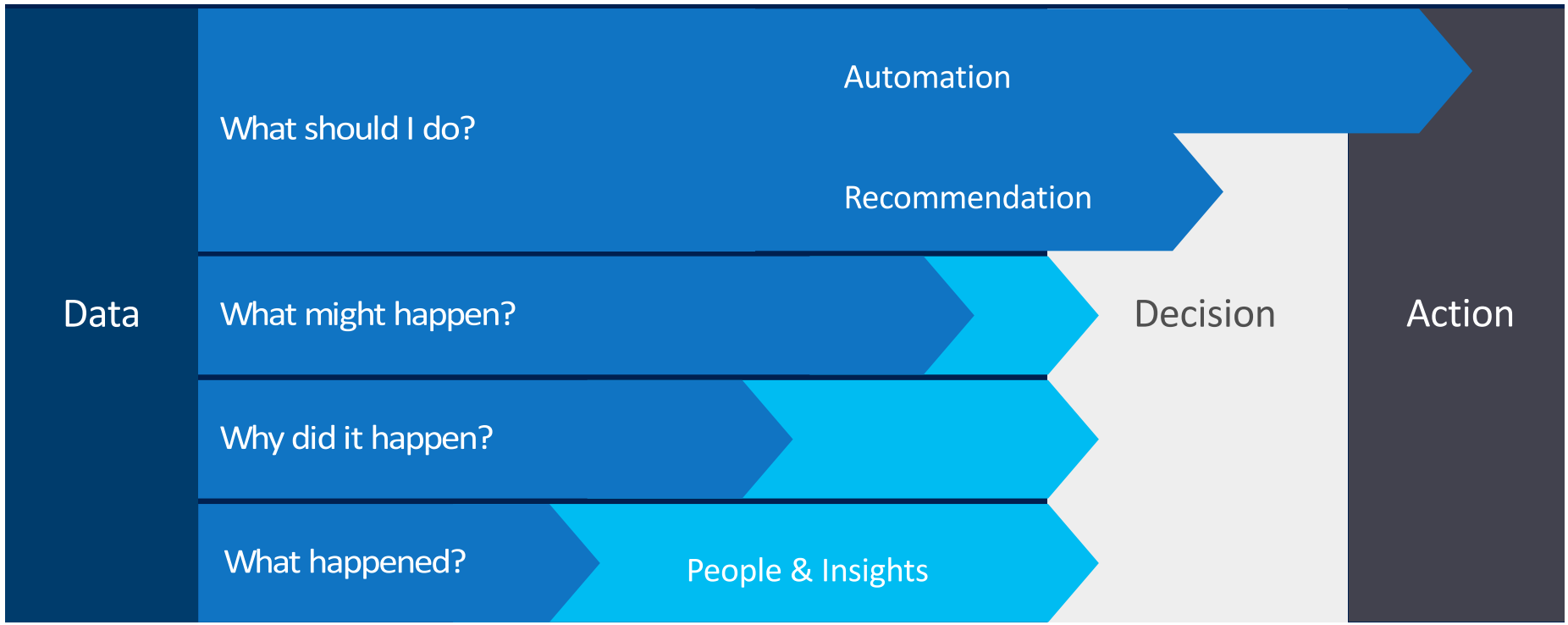
High Variety - Veracity

- **Heterogeneity**/mobility
- Incompleteness/**uncertainty**
- Interaction with the real-world





Data is not Information



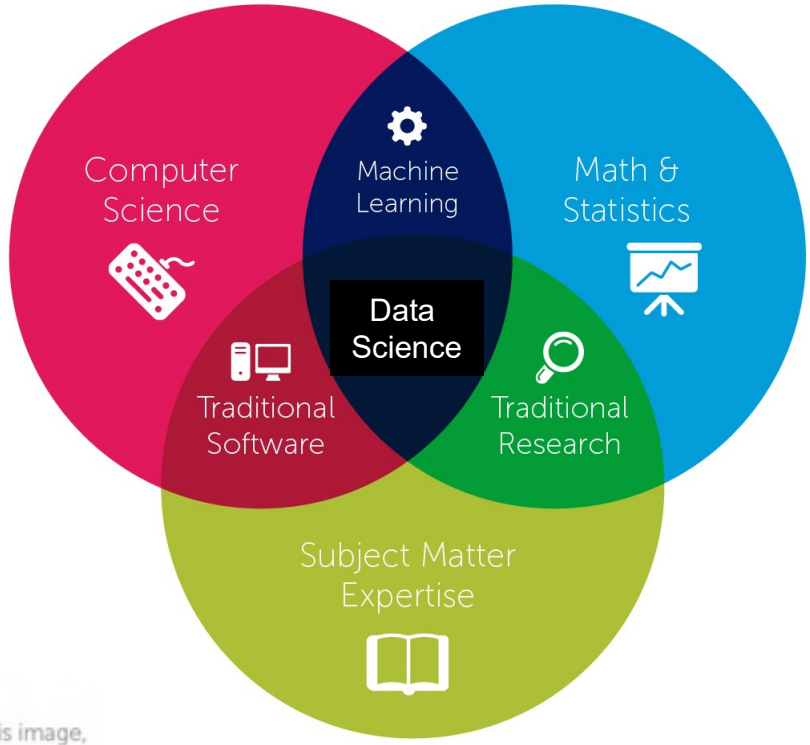
Courtesy of Microsoft

What is Data Science?

- *Data Science* aims to derive knowledge from big data, efficiently and intelligently
- *Data Science* encompasses the set of activities, tools, and methods that enable data-driven activities in science, business, medicine, and government

http://www.oreilly.com/data/free/what-is-data-science.csp

What is Data Science?



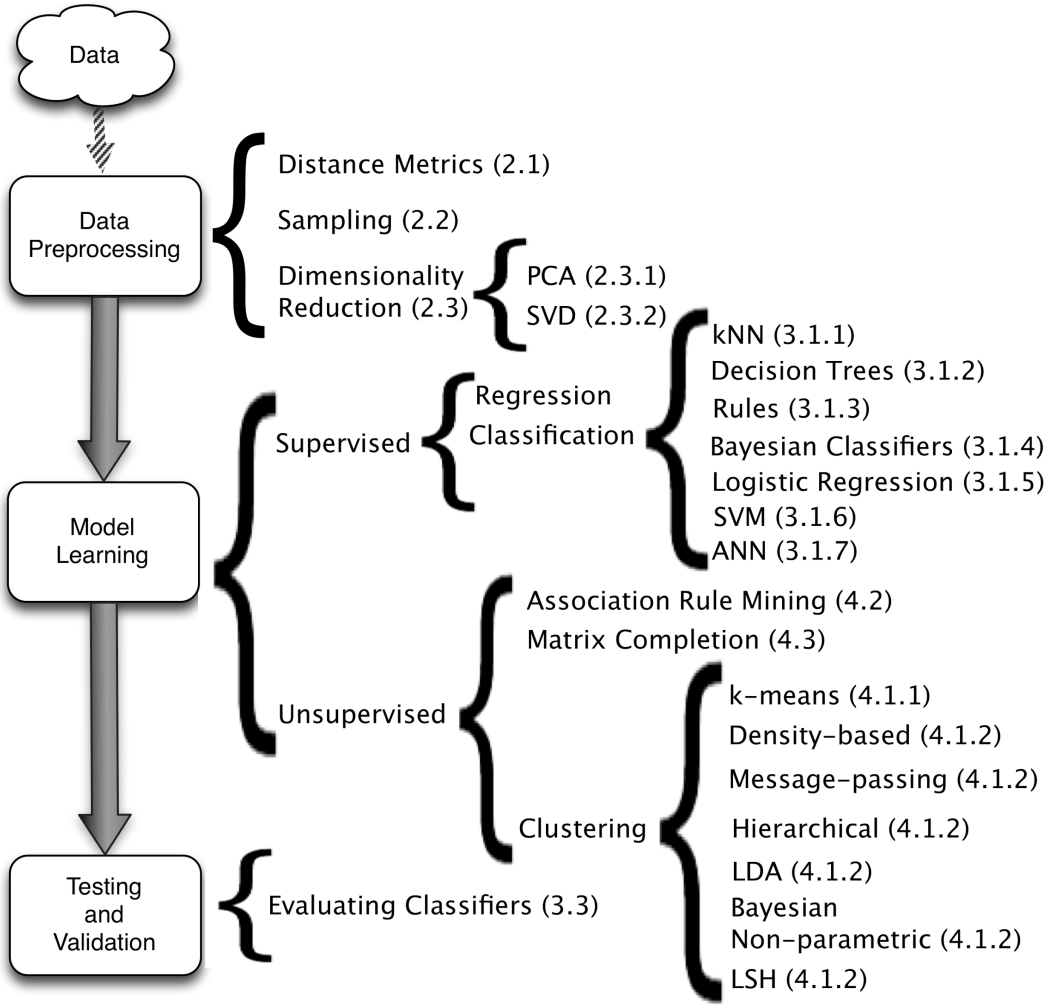
http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram

Machine Learning Overview

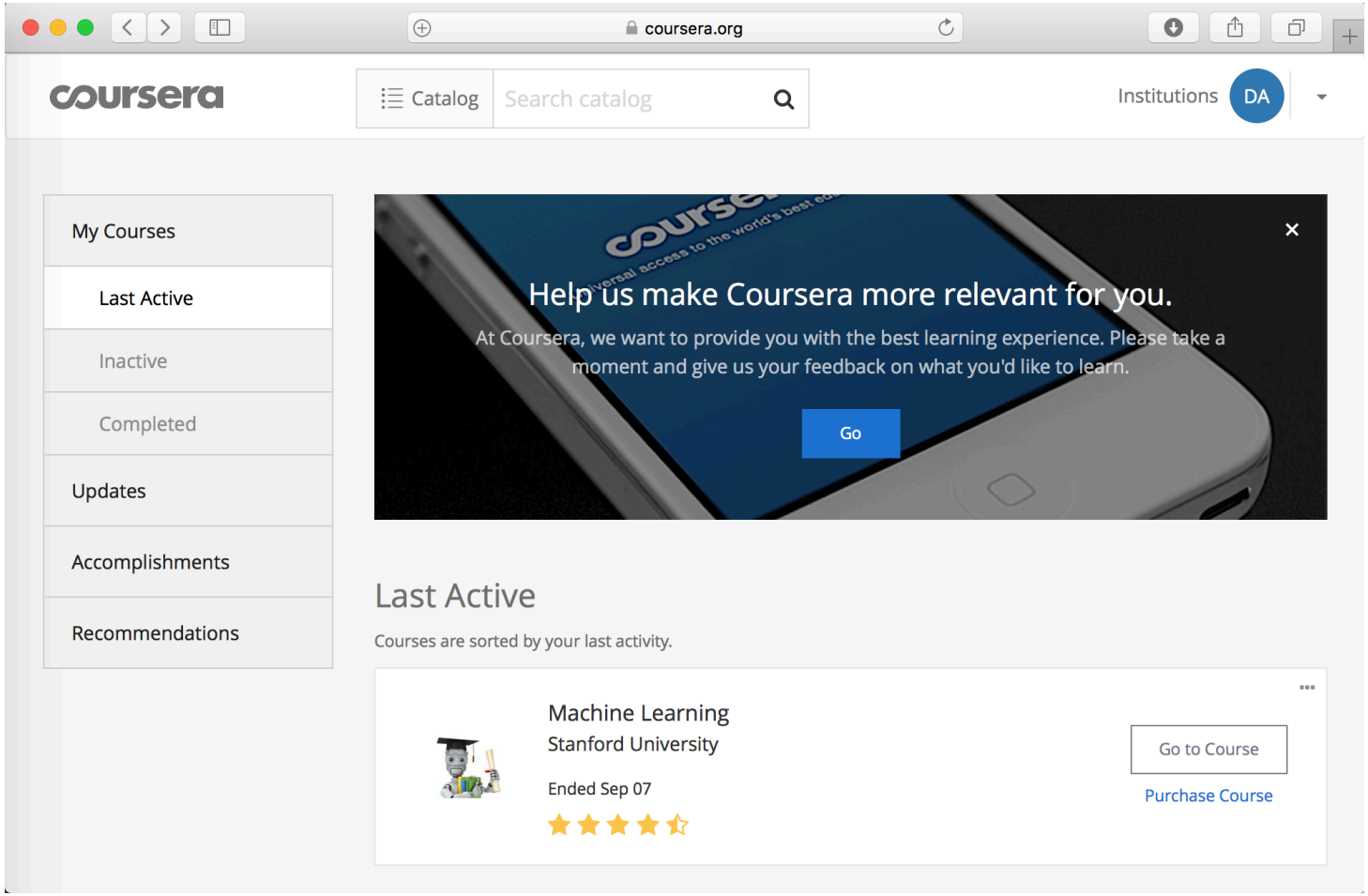
Recommendation as Data Mining/Machine Learning Problems

The core of the Recommendation Engine can be assimilated to a general data mining problem

(Amatriain et al. Data Mining Methods for Recommender Systems in Recommender Systems Handbook)



Coursera ML course



- Stanford Machine Learning by Andrew Ng

Machine Learning Problems

	Supervised Learning	*Unsupervised Learning*
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

What do we mean by learning?

- Given:
  - a data set **D**,
  - a task **T**, and
  - a performance measure **M**

- A computer system is said to learn from **D** to perform the task **T** if after learning the system performance on **T** improves as measured by **M**
- In other words, the learned model helps the system to perform T better as compared to no learning

Machine learning (supervised learning)

- Humans learn from **past experiences**
- A computer does not have “experiences”
  - A computer system learns from **data**, which represent some “**past experiences**” of an application domain
- Goal: learn a **target function** that can be used to **predict** the values of
  - a discrete class attribute, e.g., approve or not-approved, and high-risk or low risk (discrete world)
  - a continuous value, e.g., flight delays, cash at a bank branch/ATM (continuous setting)

The machine learning framework (classification)

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = “apple”

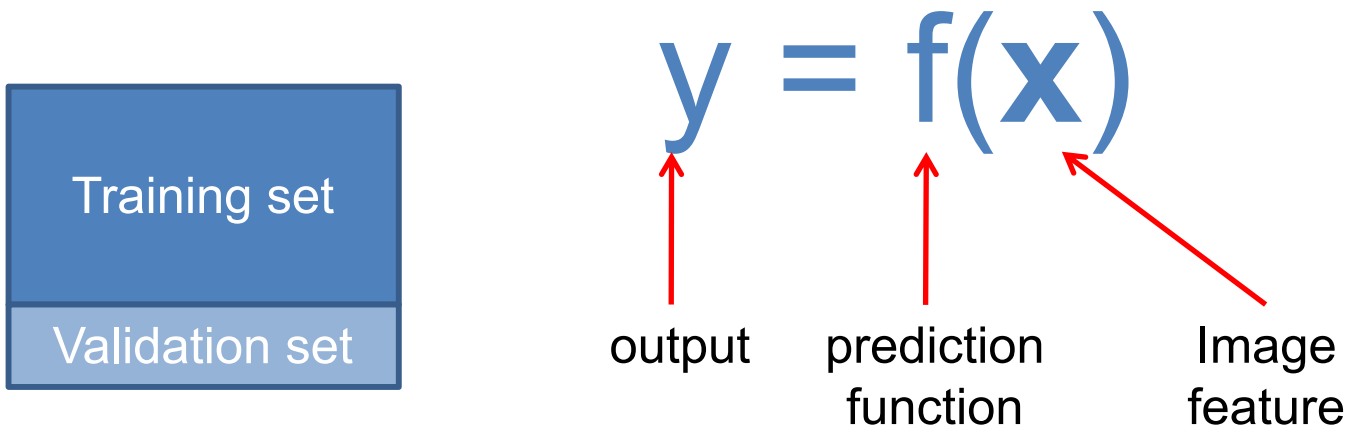
f() = “tomato”

f() = “cow”

Supervised vs. unsupervised Learning

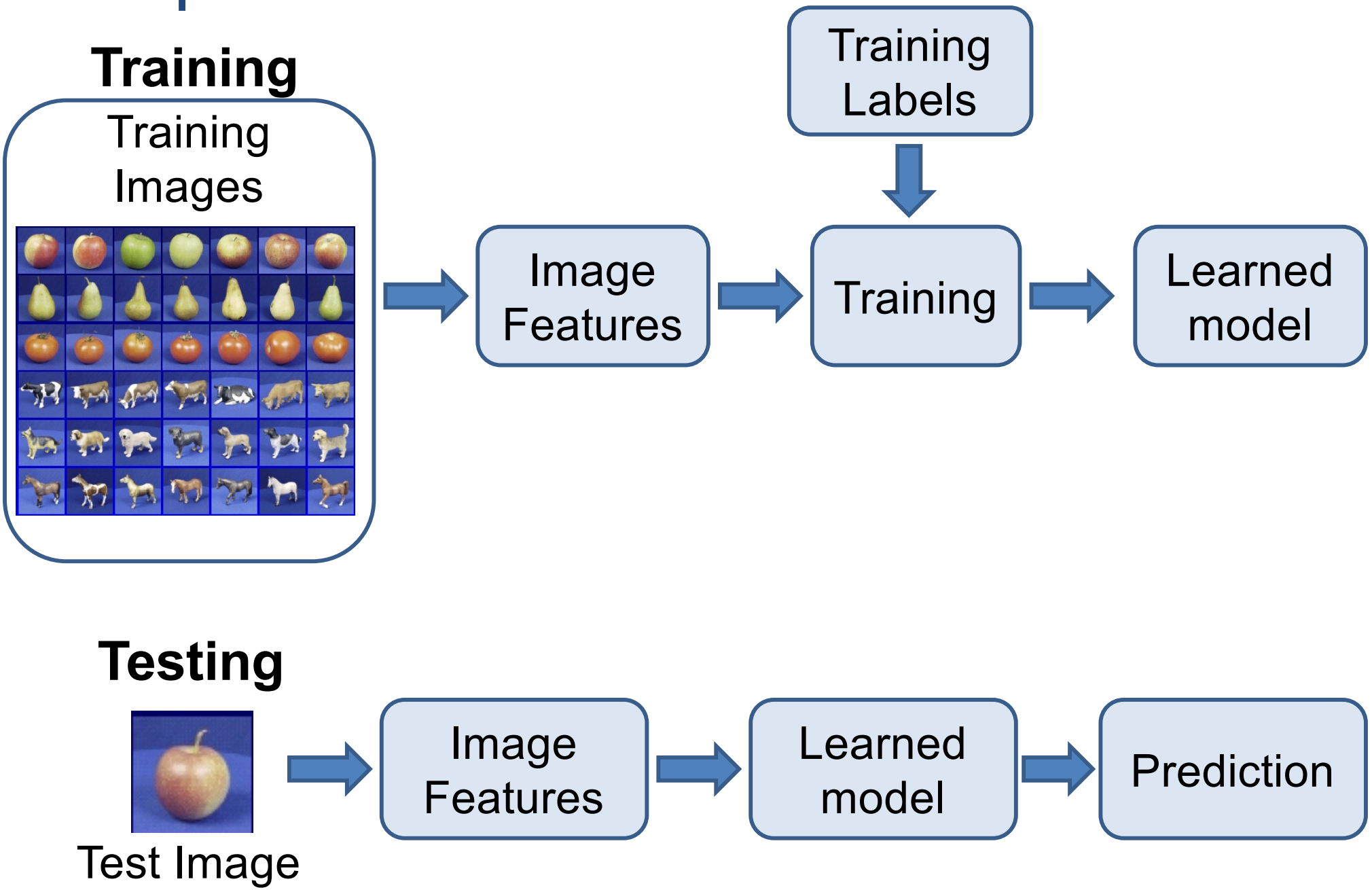
- Supervised learning: classification is seen as supervised learning from examples
  - Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a “teacher” gives the classes (supervision)
  - Test data are classified into these classes too
- Unsupervised learning (clustering)
  - Class labels of the data are unknown
  - Given a set of data, the task is to establish the existence of classes or clusters in the data

The machine learning framework



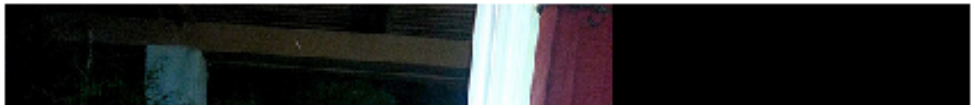
- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* **x** and output the predicted value $y = f(\mathbf{x})$

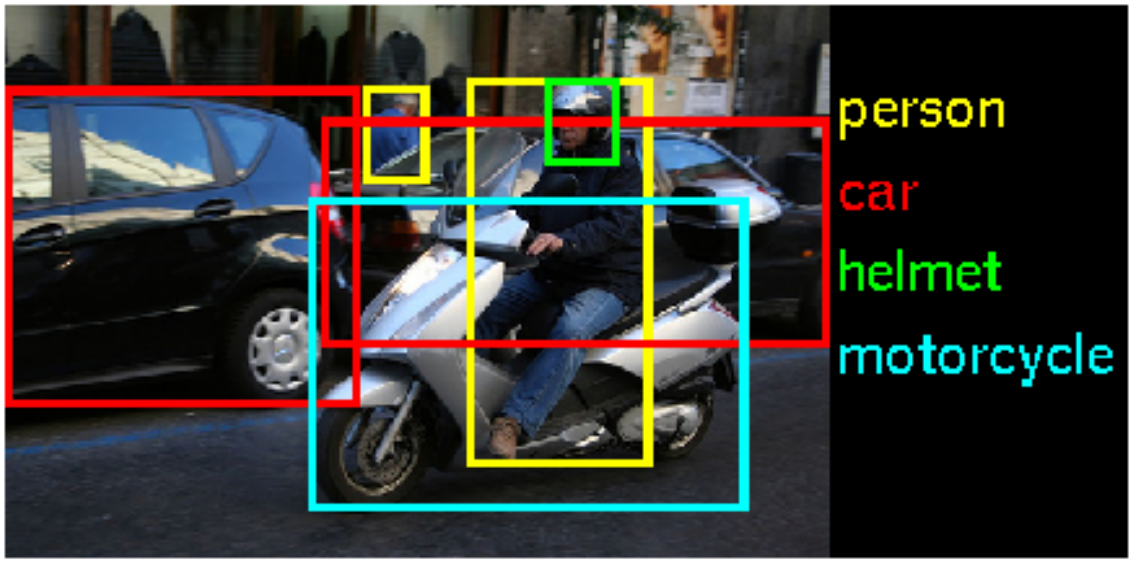
Steps


Training
Training Images
Training Labels
Image Features
Training
Learned model
Testing
Test Image
Image Features
Learned model
Prediction

Advanced image processing

Is an apple



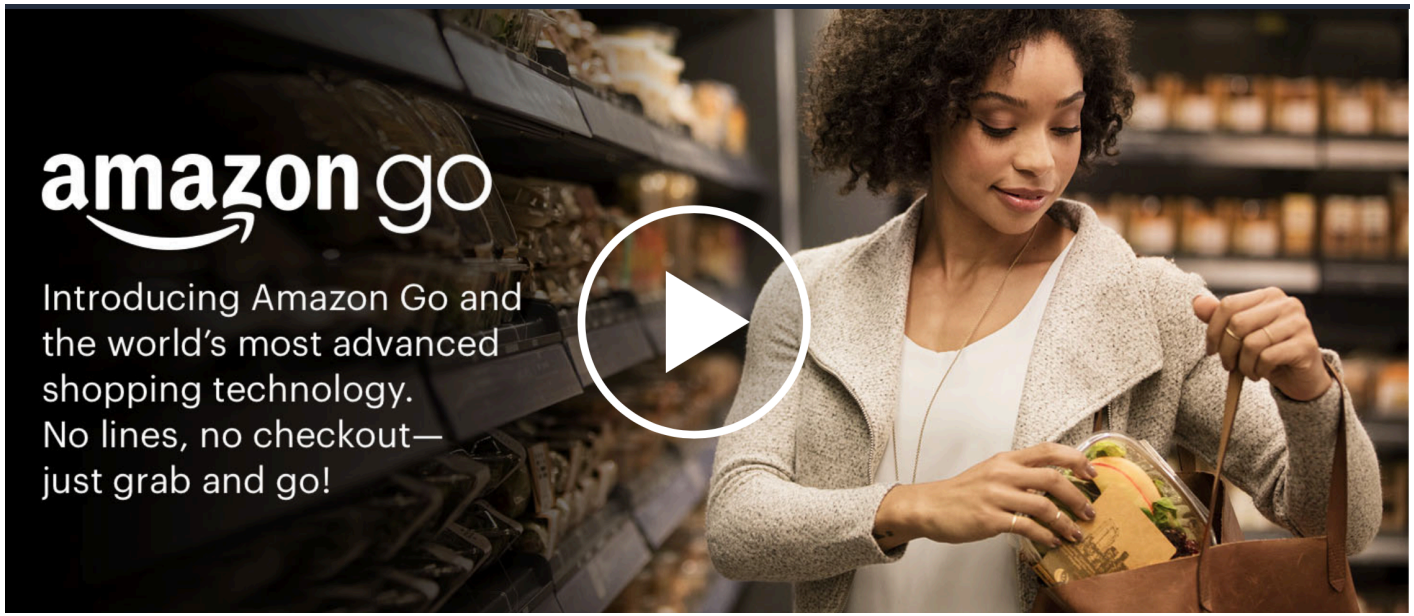


http://image-net.org/challenges/LSVRC/2014/

Advanced image processing

https://youtu.be/bS_n47I3pdg
https://youtu.be/ghrrWXYXdUM
https://youtu.be/j1WUrbnAqMg
https://www.bustle.com/articles/186603-facial-recognition-analysis-of-the-first-2016-presidential-debate-reveals-some-subtle-insights-videos

Advanced image processing

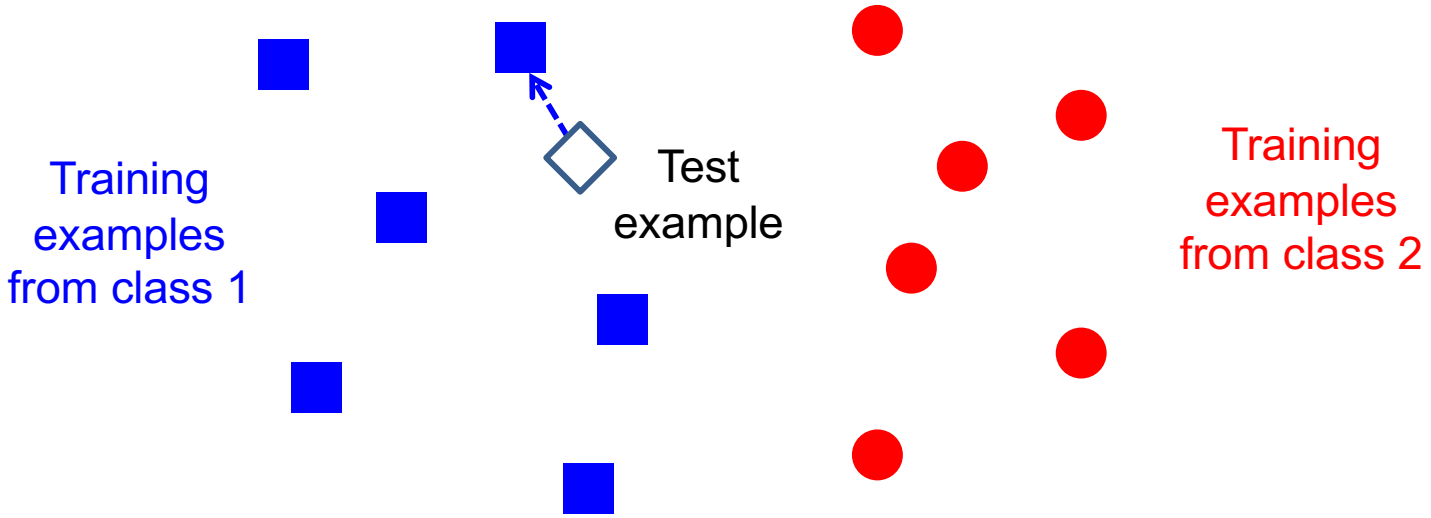


- Amazon Go store: https://youtu.be/NrmMk1Myrxc
  - no checkout required
  - use the Amazon Go app to enter the store, take the products you want, and go
  - automatically detects when products are taken from or returned to the shelves and keeps track of them in a virtual cart
- Same types of technologies used in self-driving cars:
  - computer vision
  - sensor fusion
  - deep learning

Other examples of classification problems

- Heath care:
  - An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients
  - A decision is needed: whether to put a new patient in an intensive-care unit
  - Due to the high cost of ICU, those patients who may survive less than a month are given higher priority
  - Problem: to predict high-risk patients and discriminate them from low-risk patients
- Credit card/Financial services company:
  - Thousands of applications for new cards/loans. Each application contains information about an applicant (age, marital status, annual salary, outstanding debts, credit rating, etc)
  - Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved

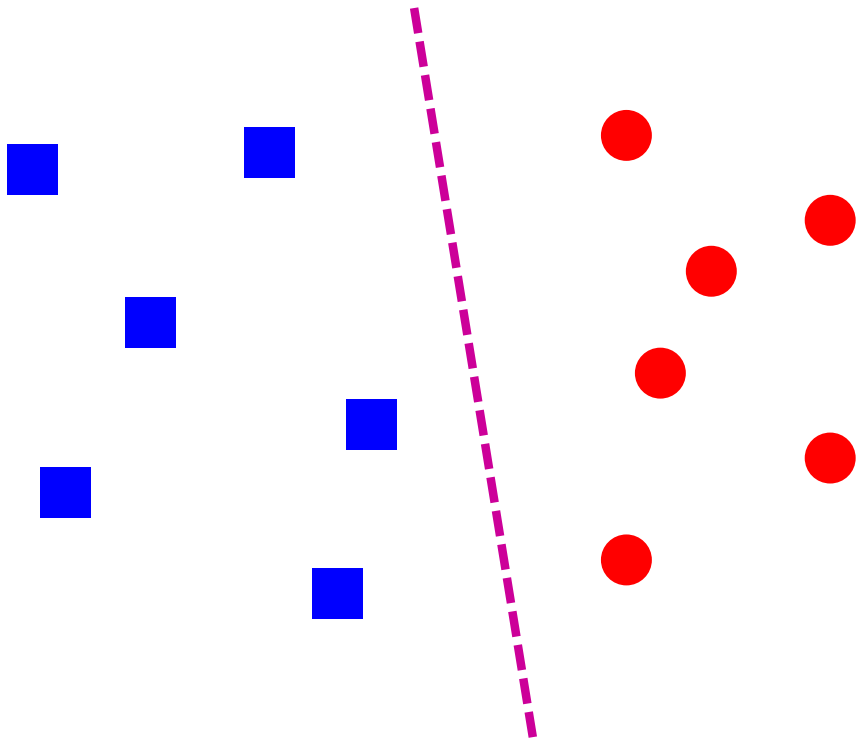
Classifiers: Nearest neighbor



f(**x**) = label of the training example nearest to **x**

- All we need is a distance function for our inputs
- No training required!

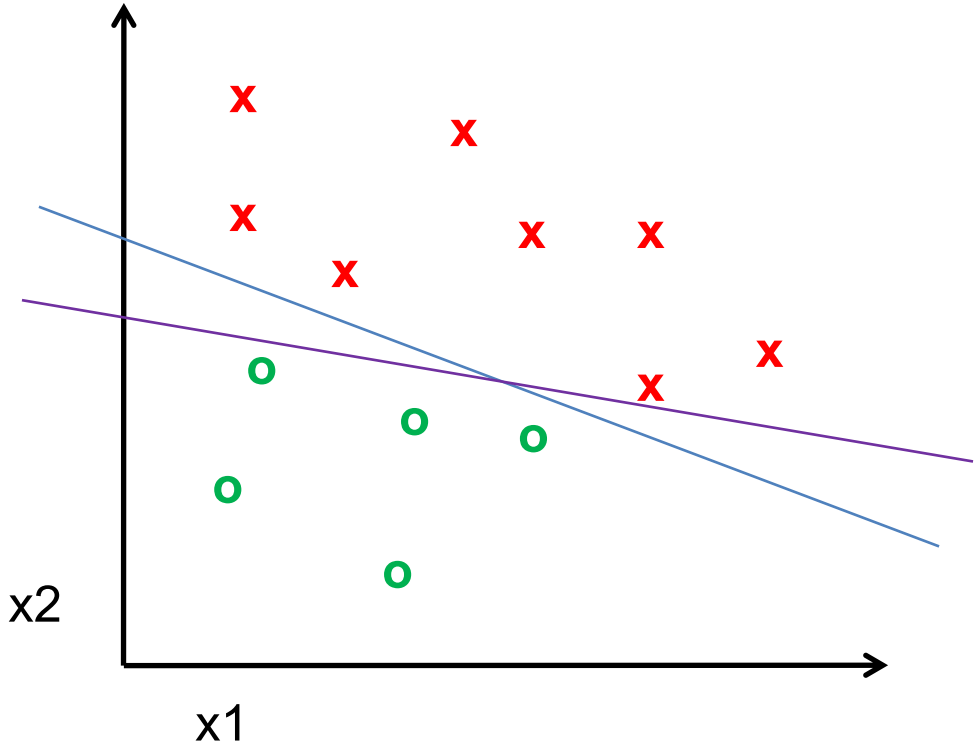
Classifiers: Linear



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

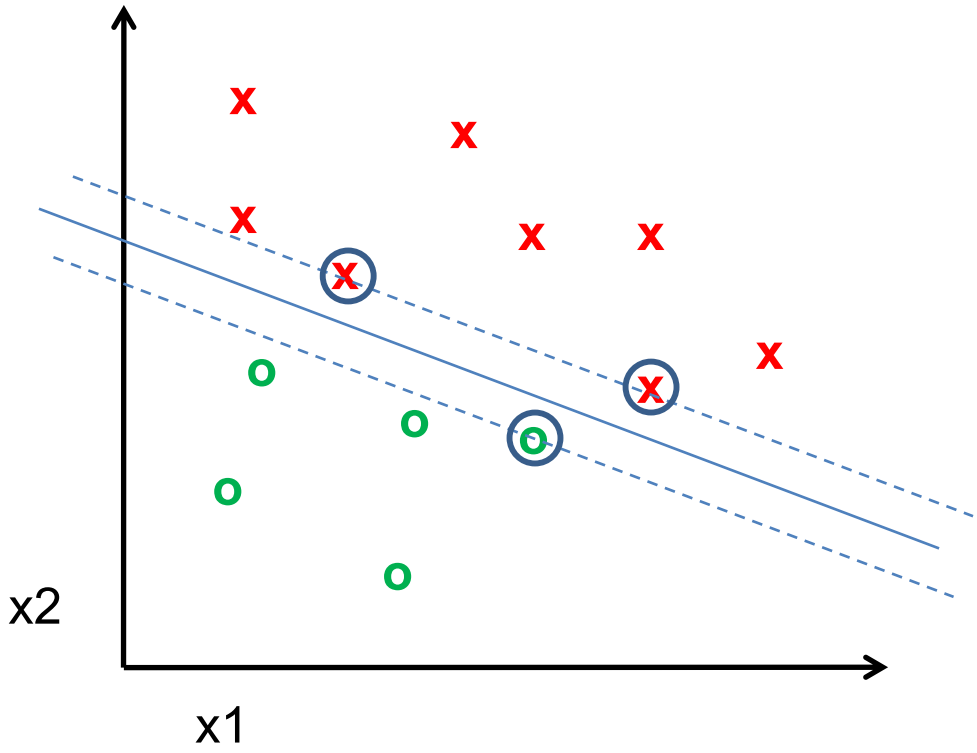
Classifiers: Linear SVM



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Classifiers: Linear SVM

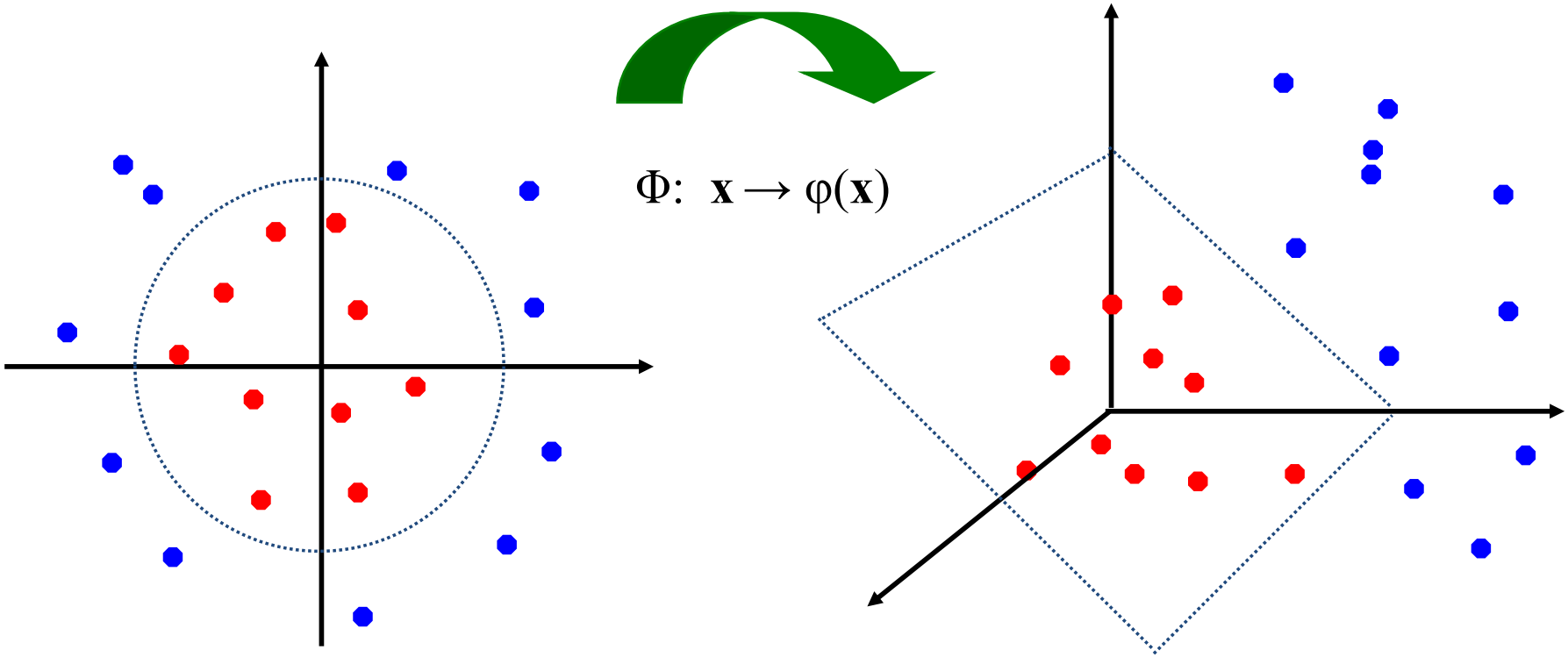


- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable



Generalization

Training set (labels known)

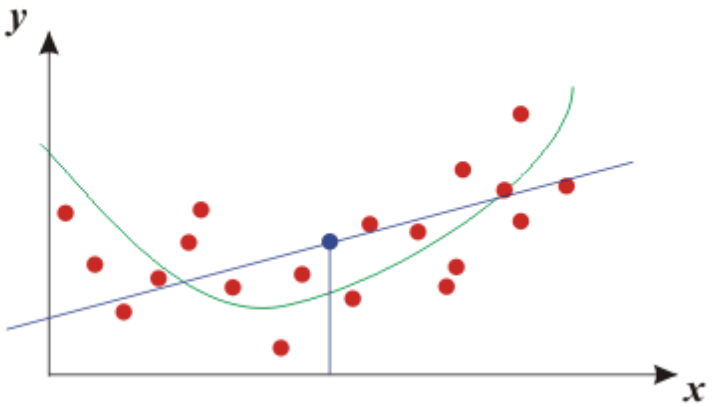
Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

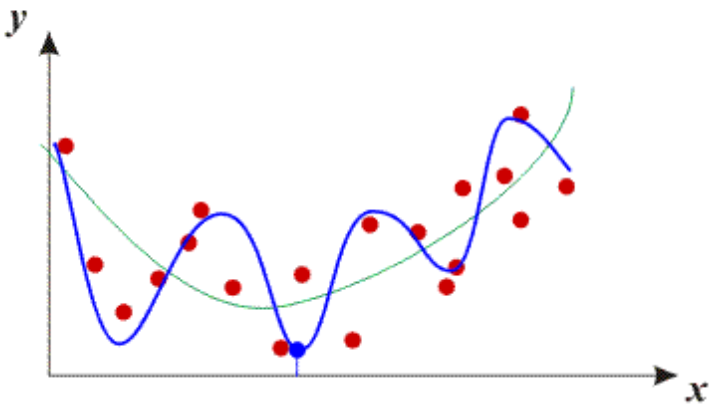
Generalization

- Components of generalization error
  - **Bias**: how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model
  - **Variance**: how much models estimated from different training sets differ from each other?
- **Underfitting**: model is too "simple" to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- **Overfitting**: model is too "complex" and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

Bias-Variance Trade-off



- Models with **too few parameters** are inaccurate because of a **large bias** (not enough flexibility)



- Models with **too many parameters** are inaccurate because of a **large variance** (too much sensitivity to the sample)

Unsupervised Learning

- **Learning** useful **structure** without labeled classes, optimization criterion, feedback signal, or **any** other **information** beyond the **raw data**

- Examples of applications:
  - Find **natural groupings** of Xs (X=human languages, stocks, gene sequences, animal species, customers, …)→

    prelude to discovery of underlying properties
  - Summarize the news for the past month→

    cluster first, then report centroids
  - Sequence extrapolation: e.g. predict cancer incidence next decade; predict rise in antibiotic-resistant bacteria

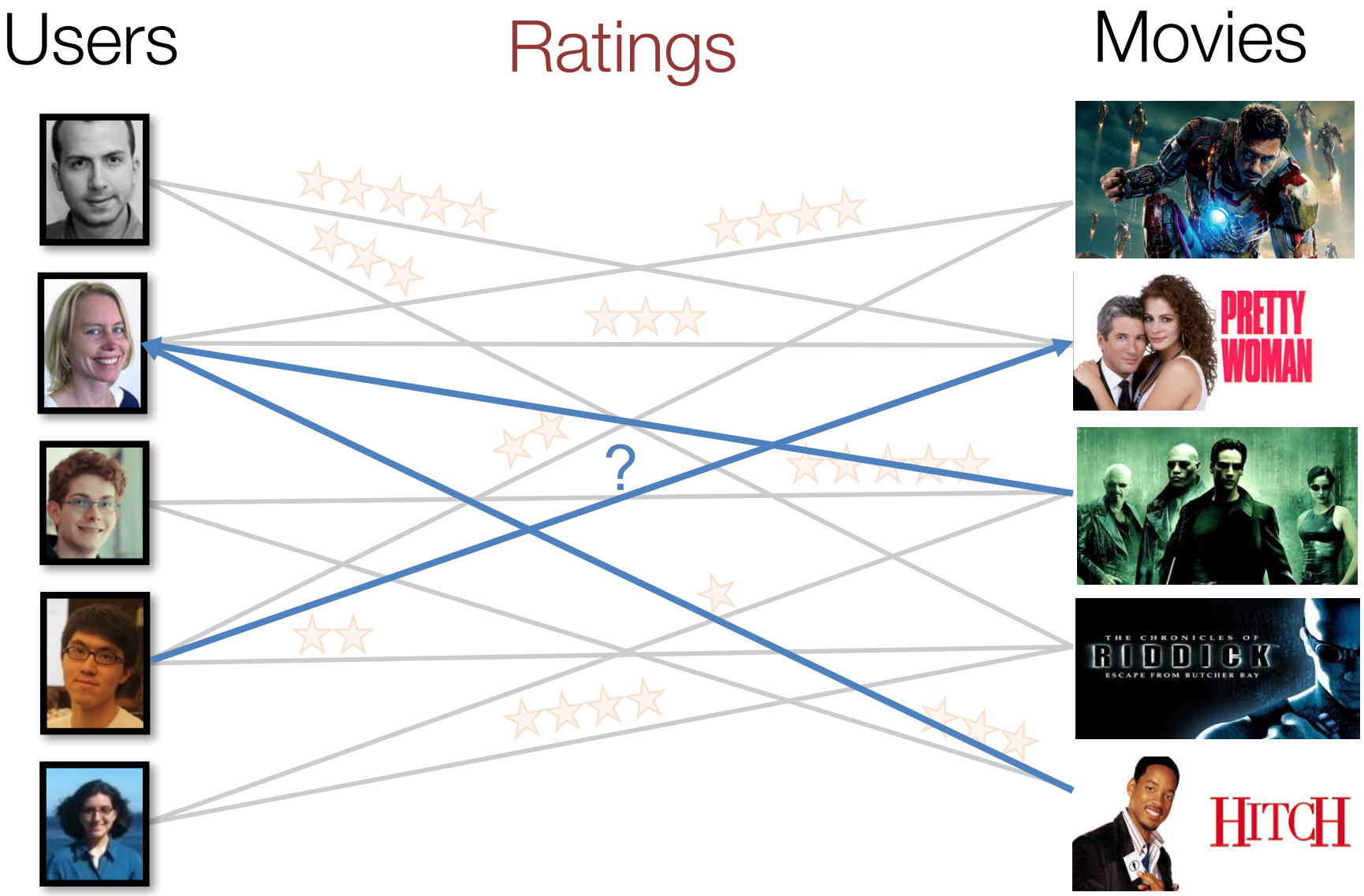
Netflix Case study

NETFLIX

- The recommender problem
  - Estimate a utility function that automatically predicts how a user will like an item

- Based on:
  - Past behavior
  - Relations to other users
  - Item similarity
  - Context
  - ….

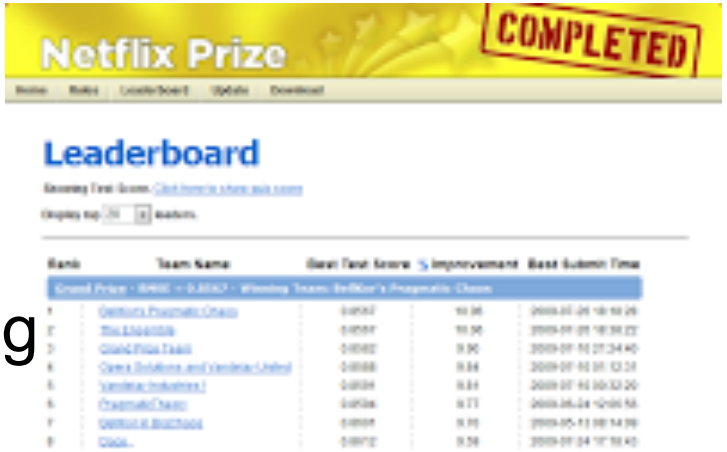
Recommending Products



Collaborative filtering

- Filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.
- Very large data sets to work
- Typical workflow:
  - A user expresses his or her preferences by rating items
  - The system matches this user's ratings against other users' and finds the people with most "similar" tastes
  - With similar users, the system recommends items that the similar users gave a high rate but not yet being rated by this user

Netflix Prize



- Announced in 2006, a machine learning and data mining competition
- $1 million to whoever improved the accuracy of the existing system called Cinematch by 10%
- Evaluation metric: root mean squared error (RMSE) of the predicted rating
- A year into the competition, the Korbell team won the first Progress Prize with an 8.43% improvement. They reported more than 2000 hours of work in order to come up with the final combination of 107 algorithms that gave them this prize

Netflix Prize

- Two underlying algorithms with the best performance in the ensemble:
  - Matrix Factorization (generally called SVD, Singular Value Decomposition)
  - Restricted Boltzmann Machines (RBM)
  - SVD by itself provided a 0.8914 RMSE, while RBM alone provided a competitive but slightly worse 0.8990 RMSE
  - A linear blend of these two reduced the error to 0.88
- To put these algorithms in production:
  - Improve scalability: built to handle 100 million ratings, instead of the more than 5 billion available at Netflix
  - Algorithms were not built to adapt as members added more ratings

Netflix Prize

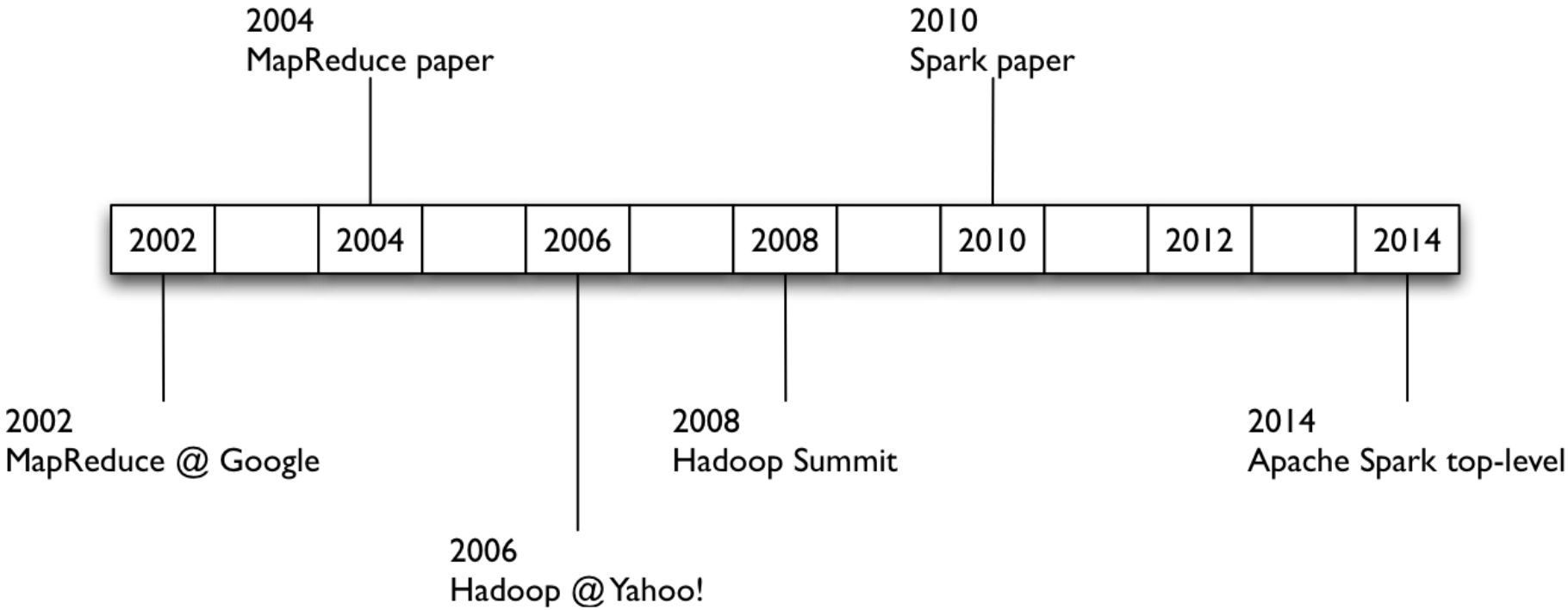
- The Grand Prize ensemble won by BellKor's Pragmatic Chaos group two years later
- Impressive blending of hundreds of predictive models
- The additional accuracy gains measured did not seem to justify the engineering effort needed to bring them into a production environment

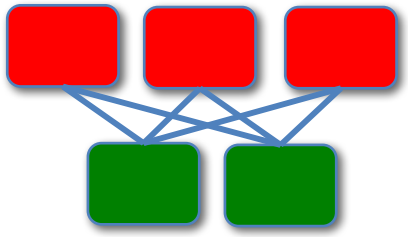
Netflix Case study

- Before Hadoop
  - Nightly processing of logs
  - Imported into a database
  - Analysis/BI
- As data volume grew, it took more than 24 hours to process and load a day worth of logs
- Today, an hourly Hadoop job processes logs for quicker availability to the data for analysis/BI
- Currently ingesting approx. 1TB/day

Map-Reduce and Hadoop Fundamentals

A Brief History



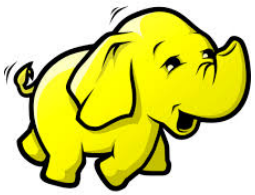


Map-reduce

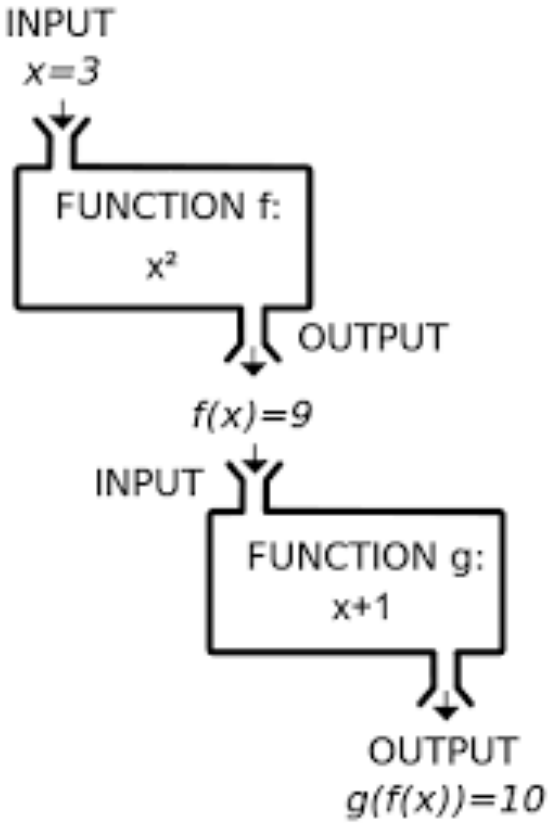
- Map-reduce: A general **algorithm**, and is prevalent in **functional programming** languages, which supports the notion of **map** and **reduce functions**

- MapReduce: The **patented software framework** from **Google** that the company applied in the realm of managing large datasets over clusters or other distributed topologies

- Hadoop: **Apache project open source** implementation of the MapReduce framework



Functional programming



- Computation as **evaluation of** mathematical **functions avoiding changing-state and mutable data**
  - functions are **expressions** and running a program means evaluating such expressions to get a **value**
- Mathematical functions have no side effects
  - output value depends only on input arguments
  - calling a ***f*** twice with the same value for an argument ***x*** will produce the same result **f(x)**
  - eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make much easier to understand and predict the behavior of a program
- Examples: **Scala**, Clojure, Haskell, Lisp

Why high interest on Map-reduce?

- **Enabling** technology for **Big Data** Applications
- **Analyzing large** amount of **unstructured data** is a **high priority** task for many companies
- The steep increase in volume of information being produced often **exceeds** the **capabilities** of existing commercial **databases**
- Moreover, the performance of **physical storage** is **not** keeping up with **improvements** in network and CPU speeds

Why high interest on Map-reduce?

- MapReduce offers a **scalable distributed** computing platform for handling **large volumes** of data and mining petabytes of **unstructured information**

- It has been increasingly used across for advanced **data analytics** and enabling new applications, e.g., **sentiment**, **clickstream**, **sensors**, **geo-location**, **server logs** analyses

Motivations for MapReduce

- Large-Scale Data Processing:
  - Want to use 1000s of CPUs
  - But don’t want hassle of ***managing*** things
- Map-reduce Architecture provides:
  - **Automatic** parallelization & distribution
  - **Fault tolerance**
  - I/O **scheduling**
  - **Monitoring** & status updates

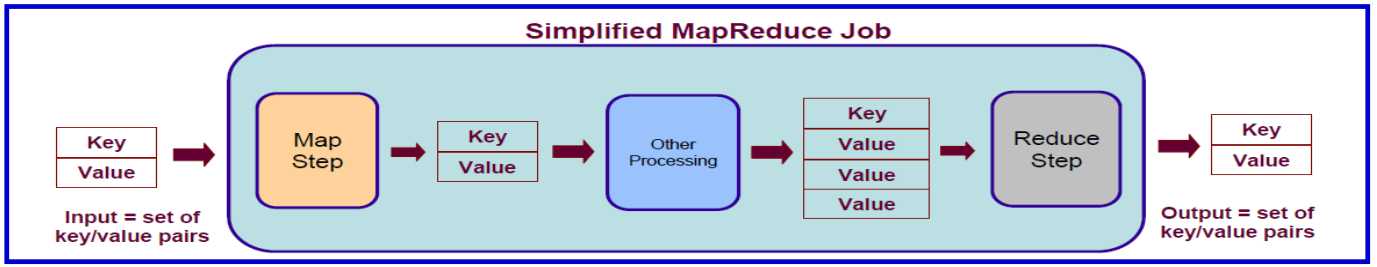
Motivations for MapReduce

- The largest known production environments:
  - By 2011: **Facebook**, which consisted of **4,000 nodes** (including 8- and 16-core CPUs) and supported **21PB** of storage
  - By (October) 2013: **Yahoo!**, spanning more than **35,000 nodes**
- Facebook Datawarehousing Hadoop cluster (2011 data):
  - **12 TB** of **compressed data added** per day
  - **800 TB** of **compressed data scanned** per day
  - **25,000** map-reduce **jobs** per day
  - **65 millions files** in HDFS
  - **30,000** simultaneous **clients** to the **HDFS NameNode**
- Nowadays Facebook Datawarehouse is **300 PB** and **add** about **600 TB** of **compressed data** per day

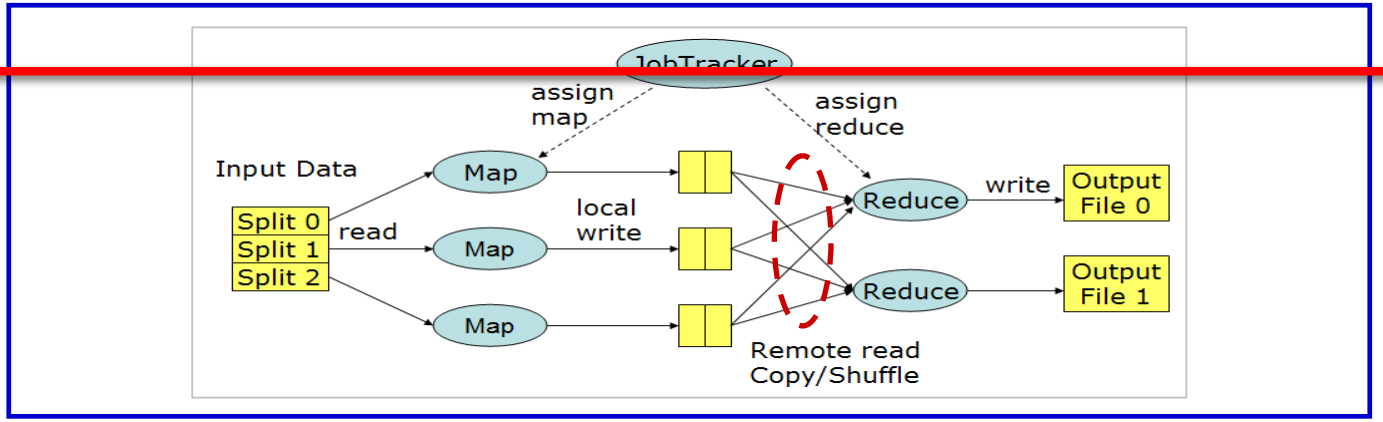
Map-reduce overview

Courtesy of IBM

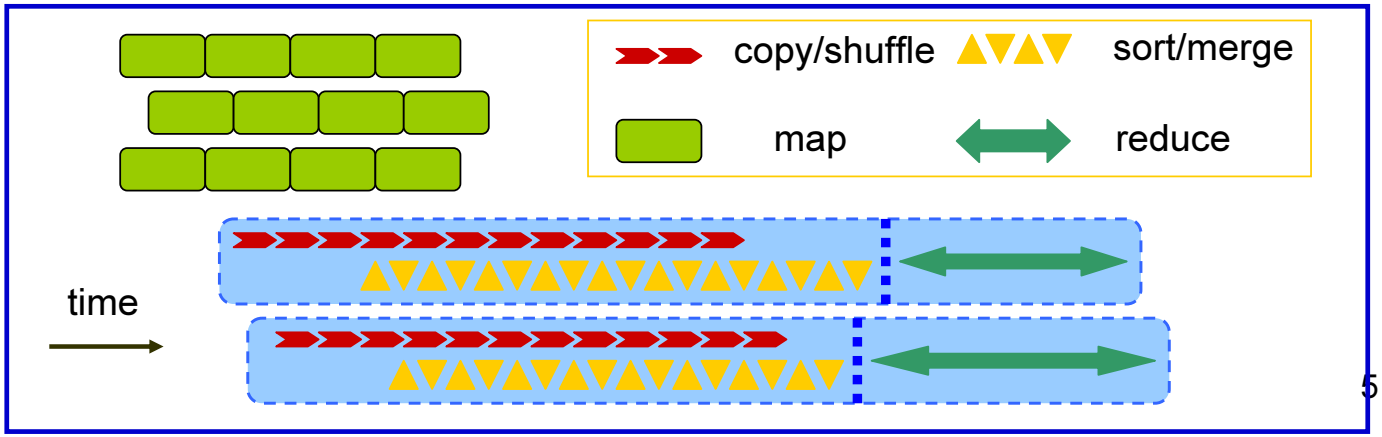
Function View



System View

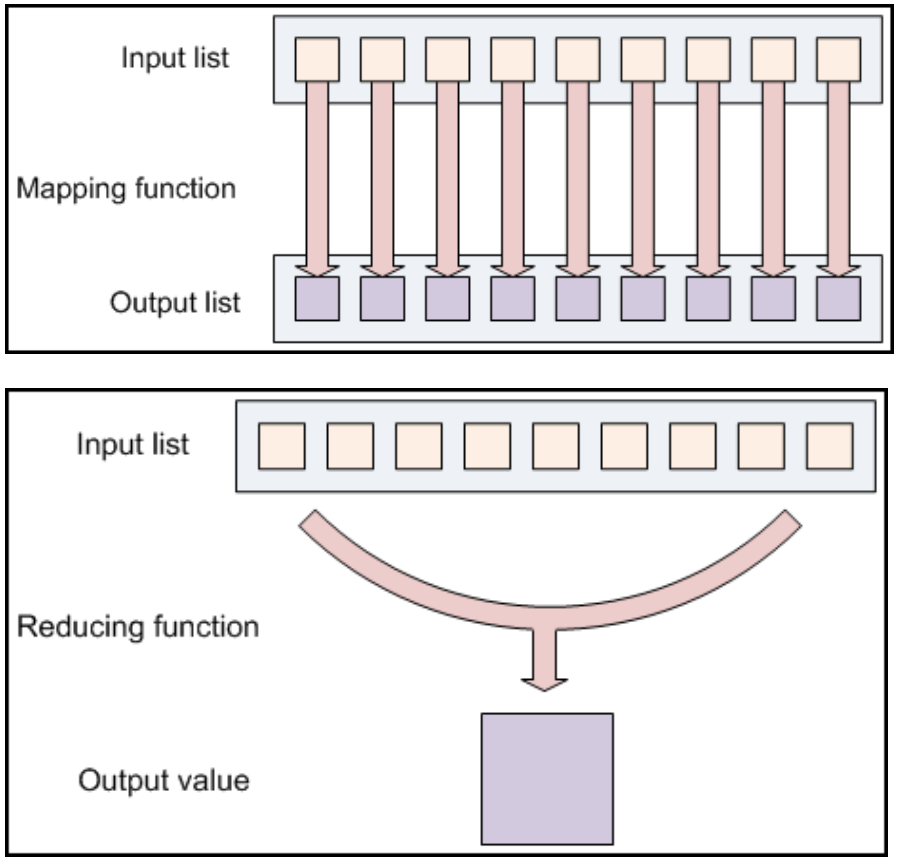


Process View



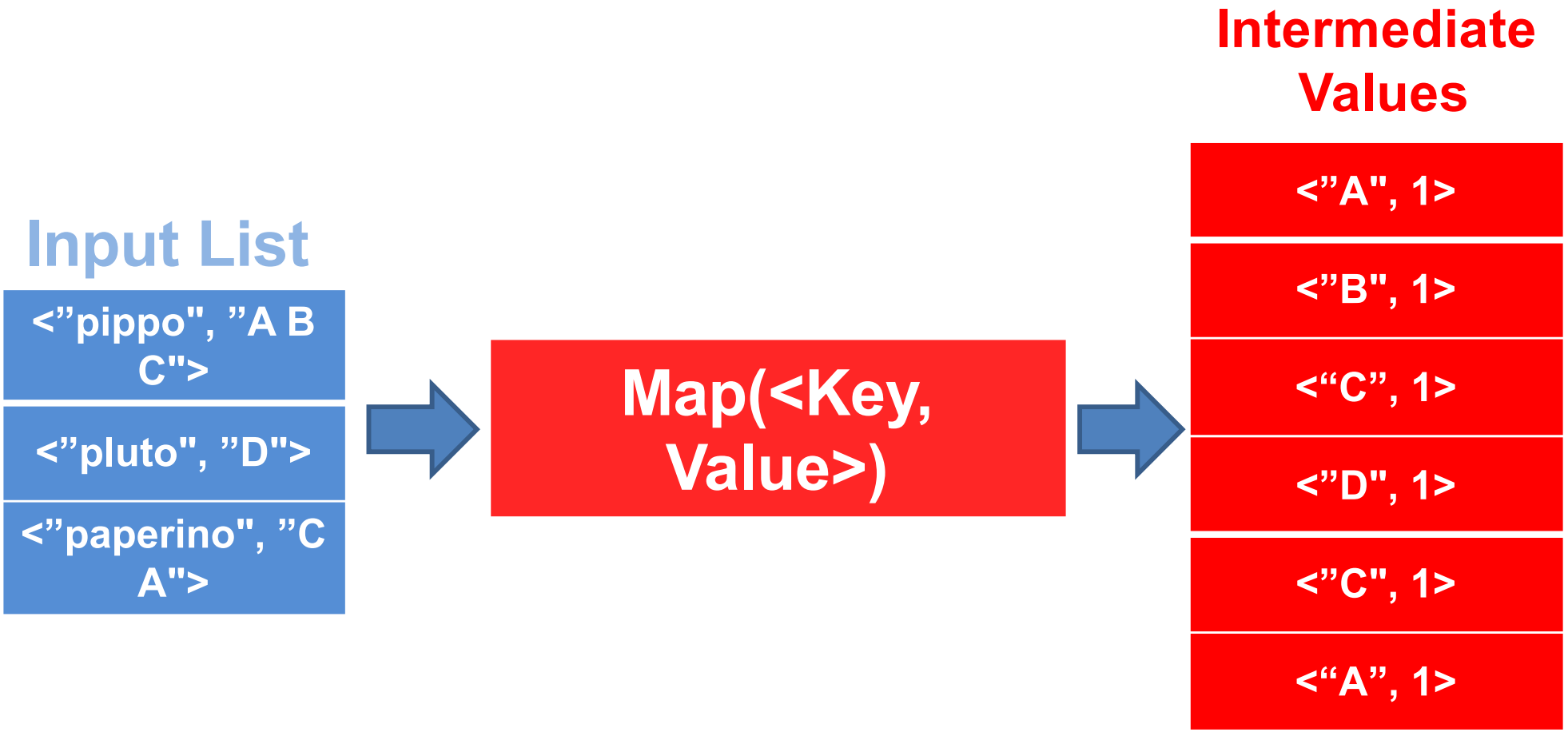
Function view

- Input: a **set** of **key/value pairs**
- User supplies two functions:
  - **map(k,v) → list(k1,v1)**
  - **reduce(k1, list(v1)) → v2**
- **(k1,v1)** is an **intermediate** key/value pair
- **Output** is the **set** of **(k1,v2)** pairs

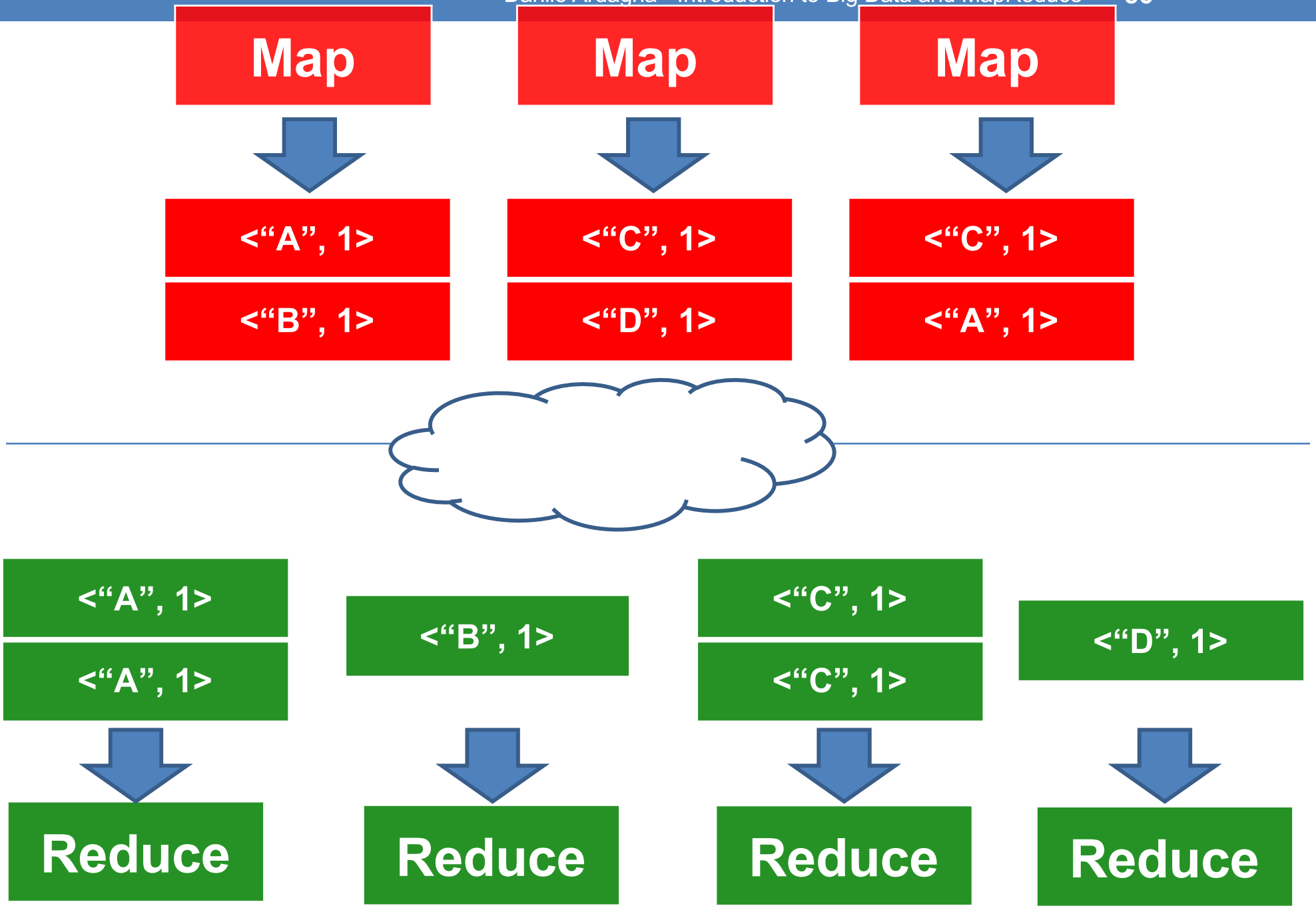


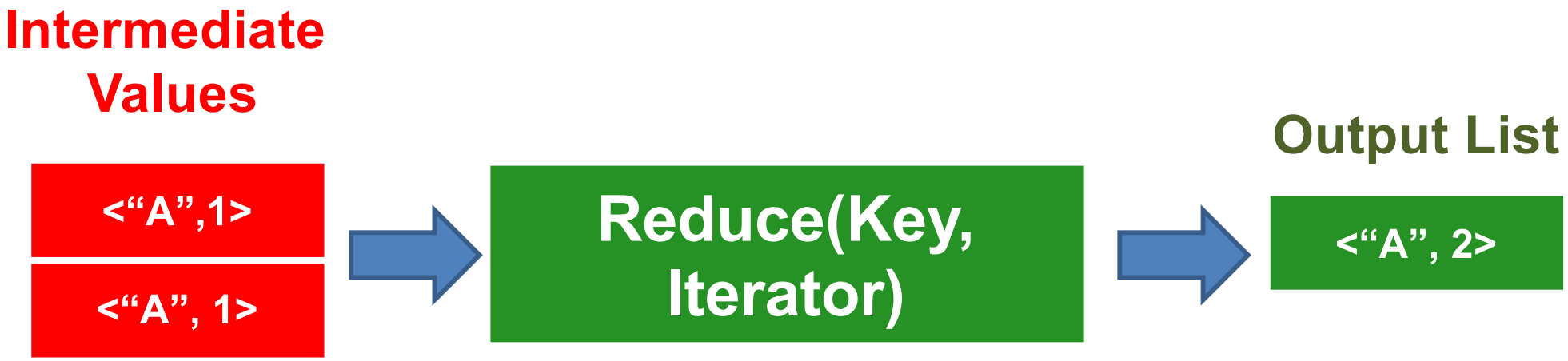
An example: Word Count

- We have many large files of words
- **Count** the number of times each **distinct word** appears in the files
- *Sample application*: analyze web server logs to find **popular URLs**




```
let map(String document_content)=
foreach Word word in document_content :
        emit(word, 1)
```

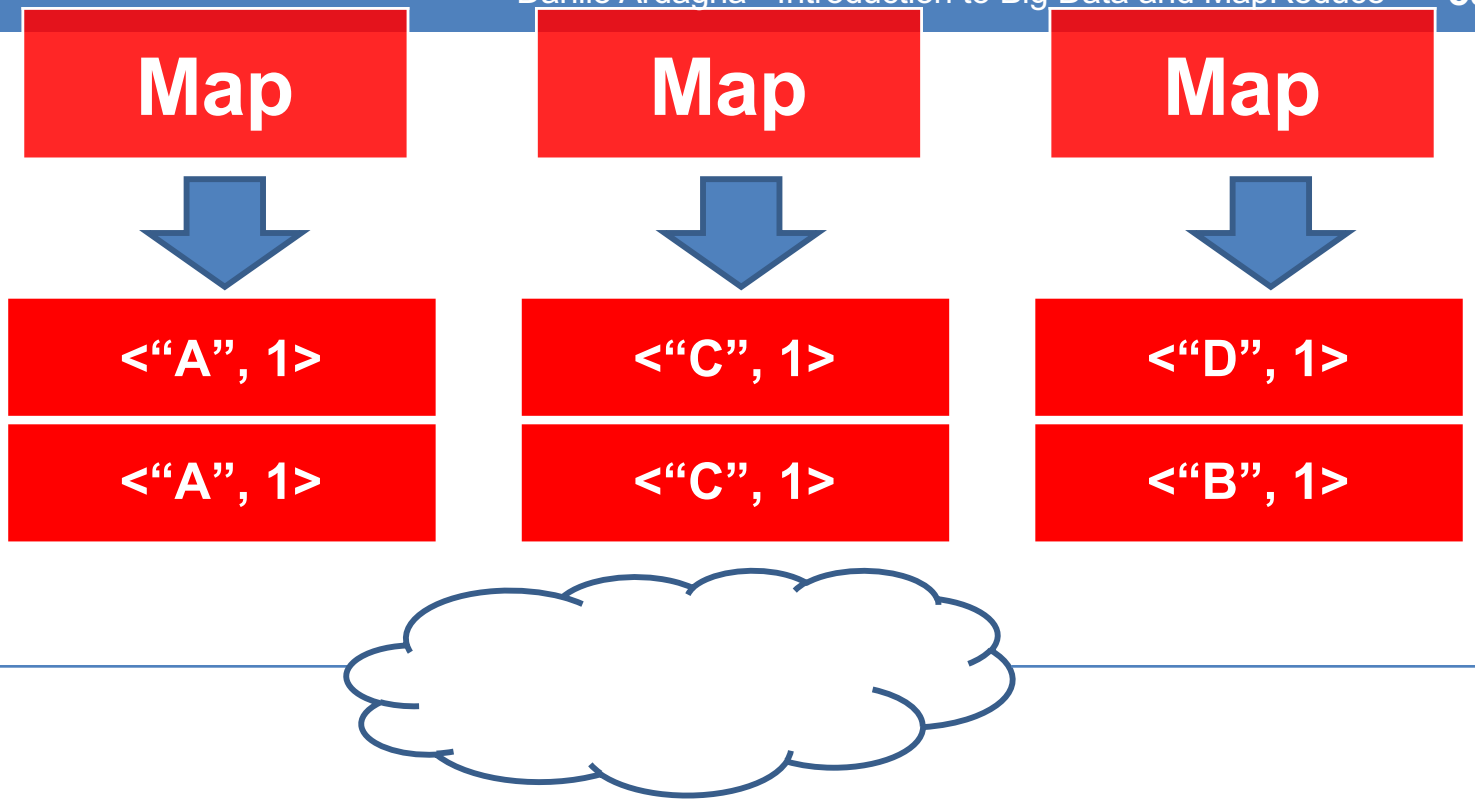

Map
Map
Map
<“A”, 1>
<“B”, 1>
<“C”, 1>
<“D”, 1>
<“C”, 1>
<“A”, 1>
<“A”, 1>
<“A”, 1>
<“B”, 1>
<“C”, 1>
<“C”, 1>
<“D”, 1>
Reduce
Reduce
Reduce
Reduce



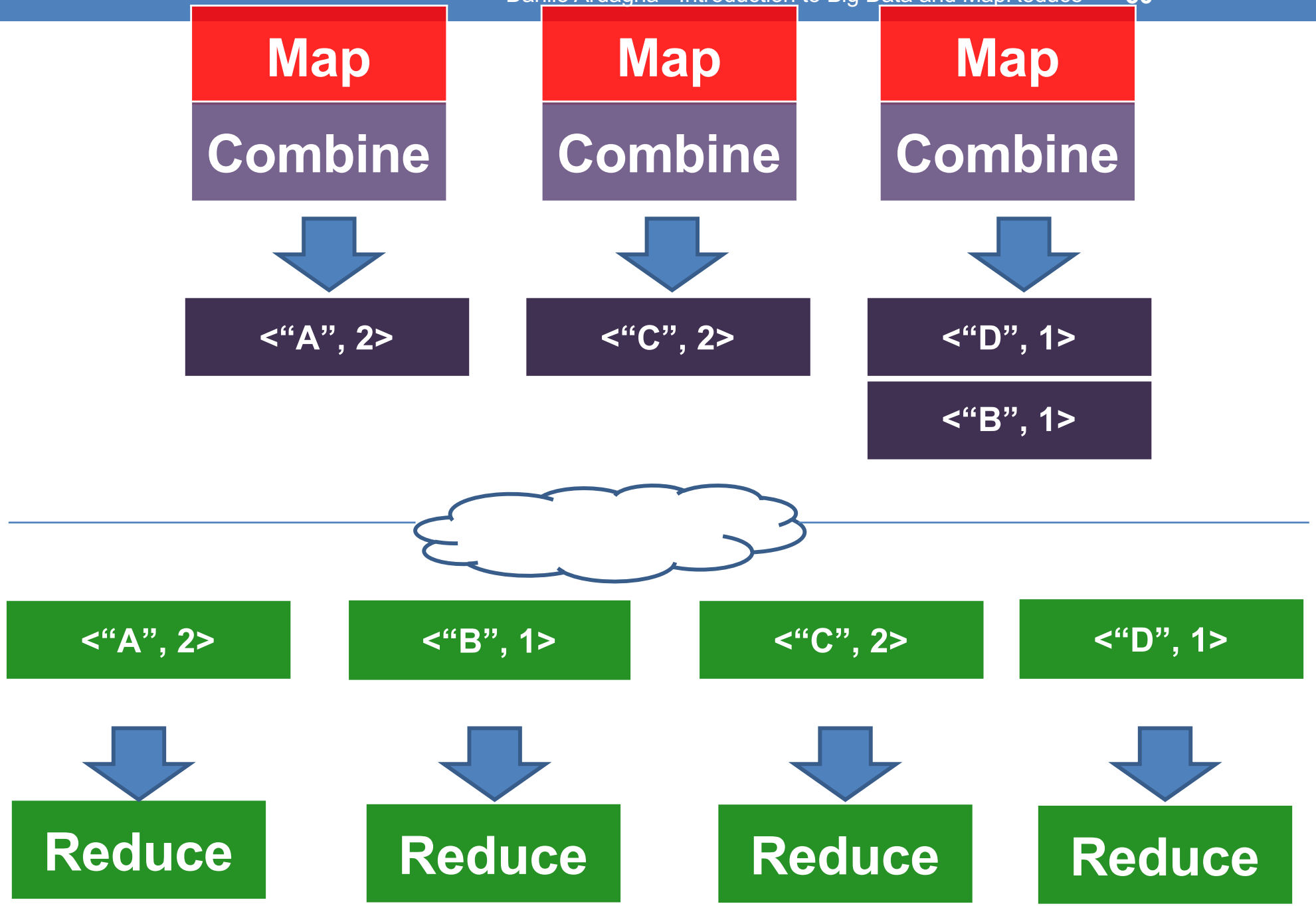

```
let reduce(Word word, Iterator<int> occurences) =
int total_occurences = 0;
foreach int o in occurences :
        total_occurences += o;
emit(word, total_occurences);
```

Combiners

- Often a **map** task will **produce many pairs** of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can **save network** time by **pre-aggregating** at mapper
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
- **Works only if reduce function is commutative and associative**

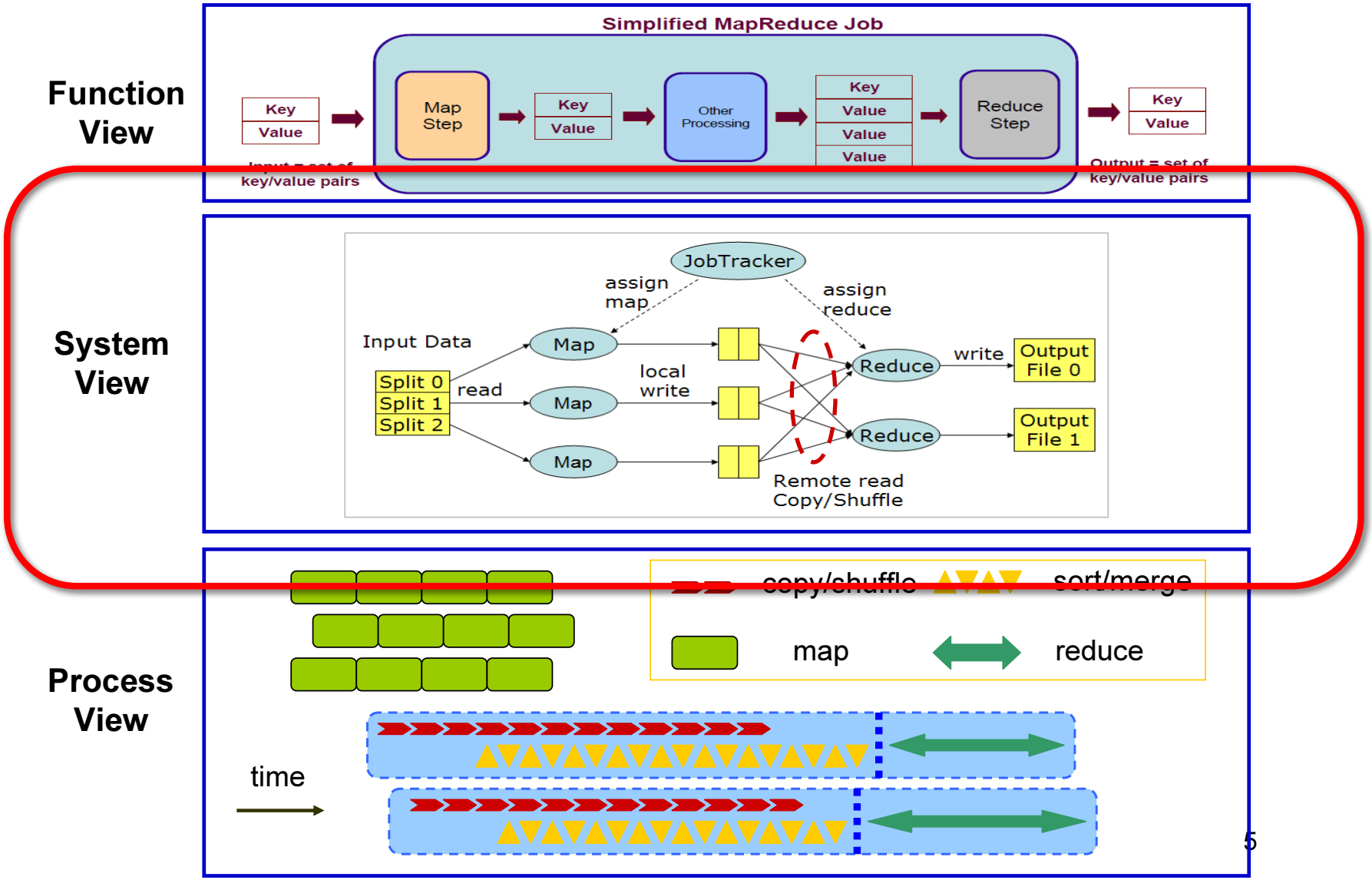


Reduce Reduce Reduce Reduce


Map
Combine
Map
Combine
Map
Combine
<“A”, 2>
<“C”, 2>
<“D”, 1>
<“B”, 1>
<“A”, 2>
<“B”, 1>
<“C”, 2>
<“D”, 1>
Reduce
Reduce
Reduce
Reduce

Map-reduce overview

Courtesy of IBM



System view: Hadoop 1.0 implementation

- Metadata management
- Access control
- Single node

- Job scheduling
- Single node

Master Nodes

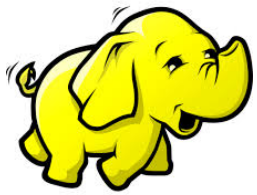
Name Node

Job Tracker

Slave Nodes

Data Node

Task Tracker



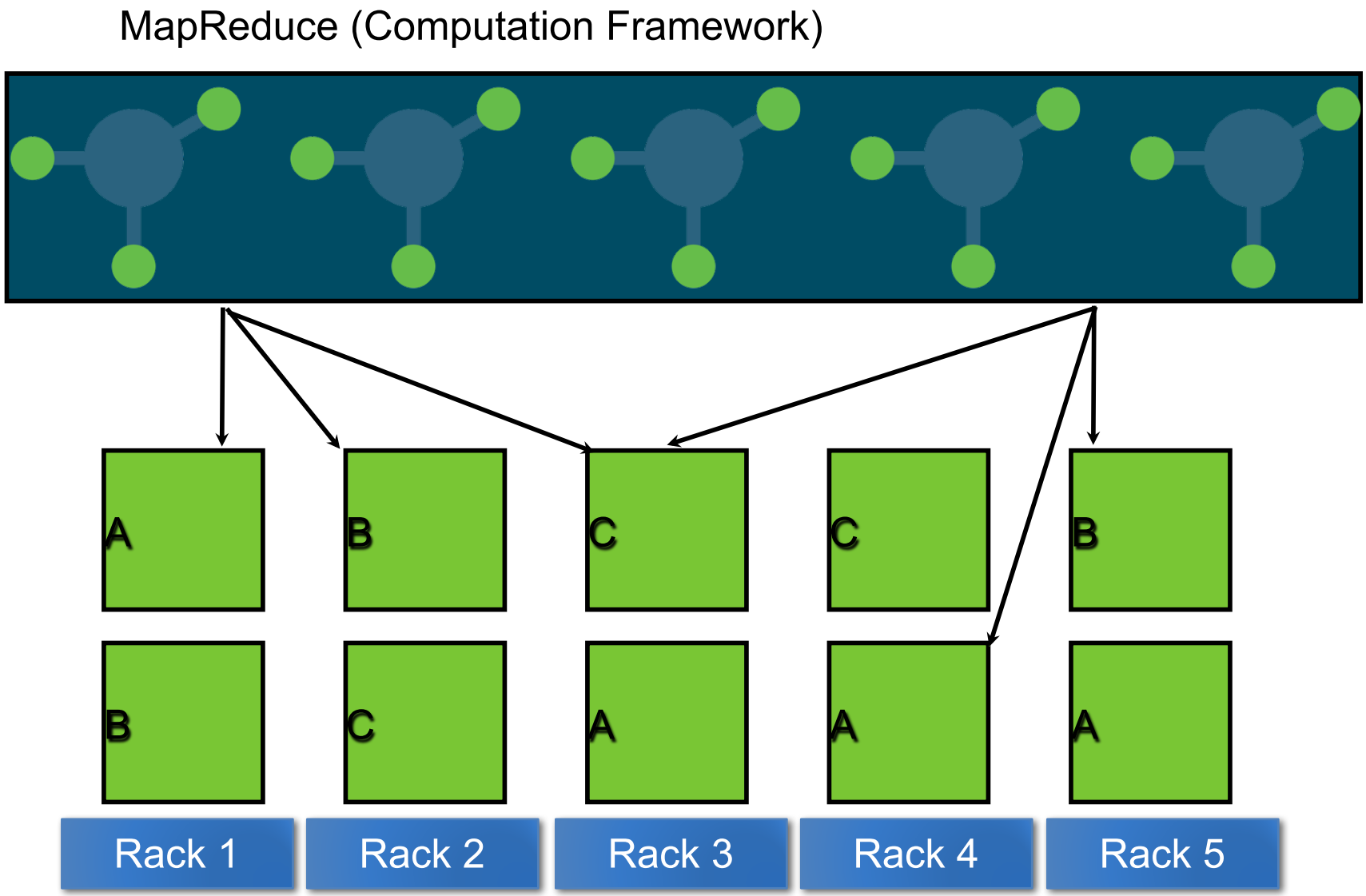
- Block management
- Multiple nodes

- Job management
- Multiple nodes

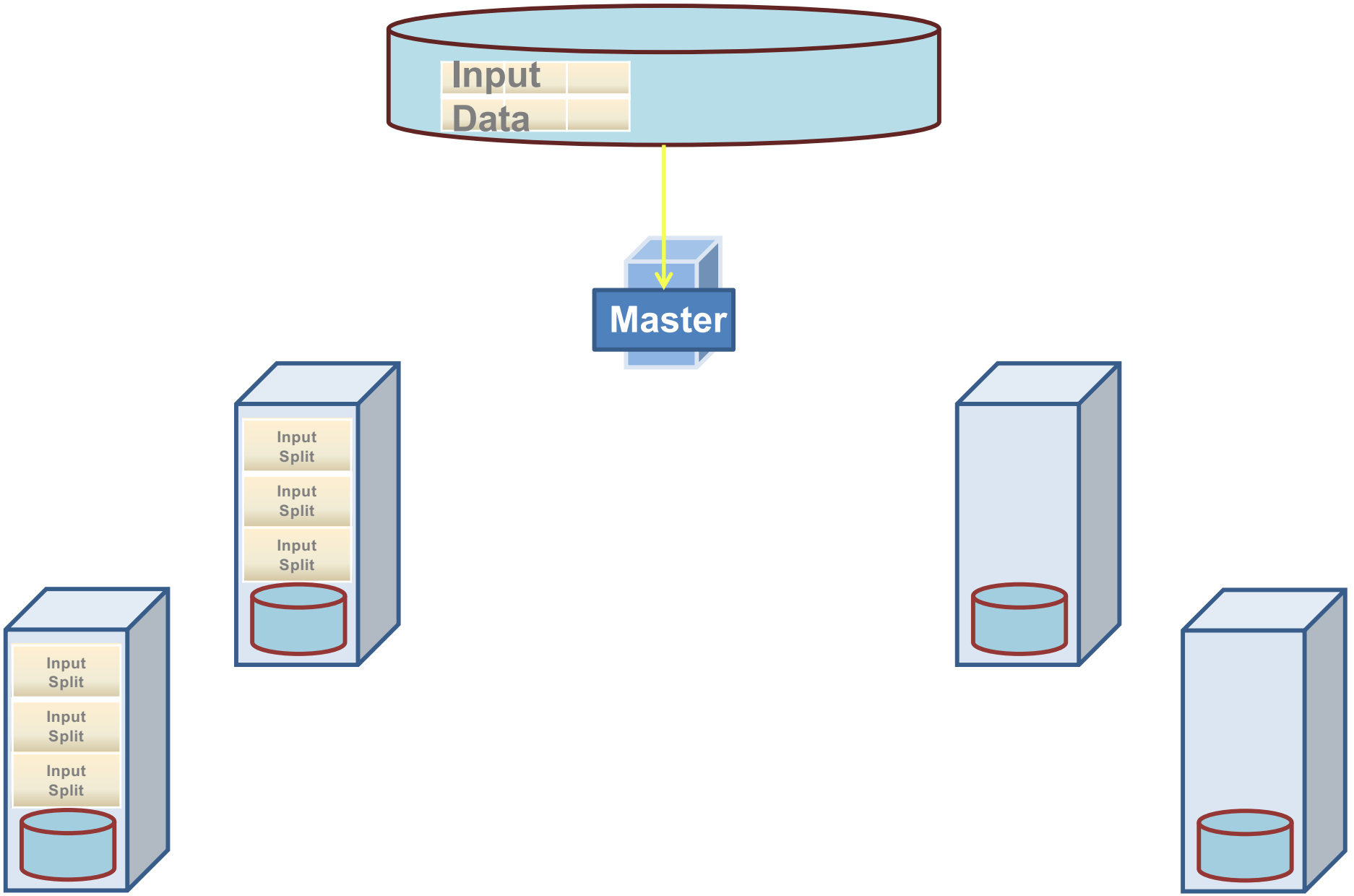
Storing data: HDFS

- HDFS: Hadoop Distributed File System
- **Data** is **broken** down into **smaller pieces** (called blocks) and **distributed throughout** the **cluster**
- Goal: use **commonly** available **servers** in a very large cluster, where each **server** has a set of **inexpensive** internal **disk** drives (data locality)
- Every block is replicated (default: **replication factor 3**)

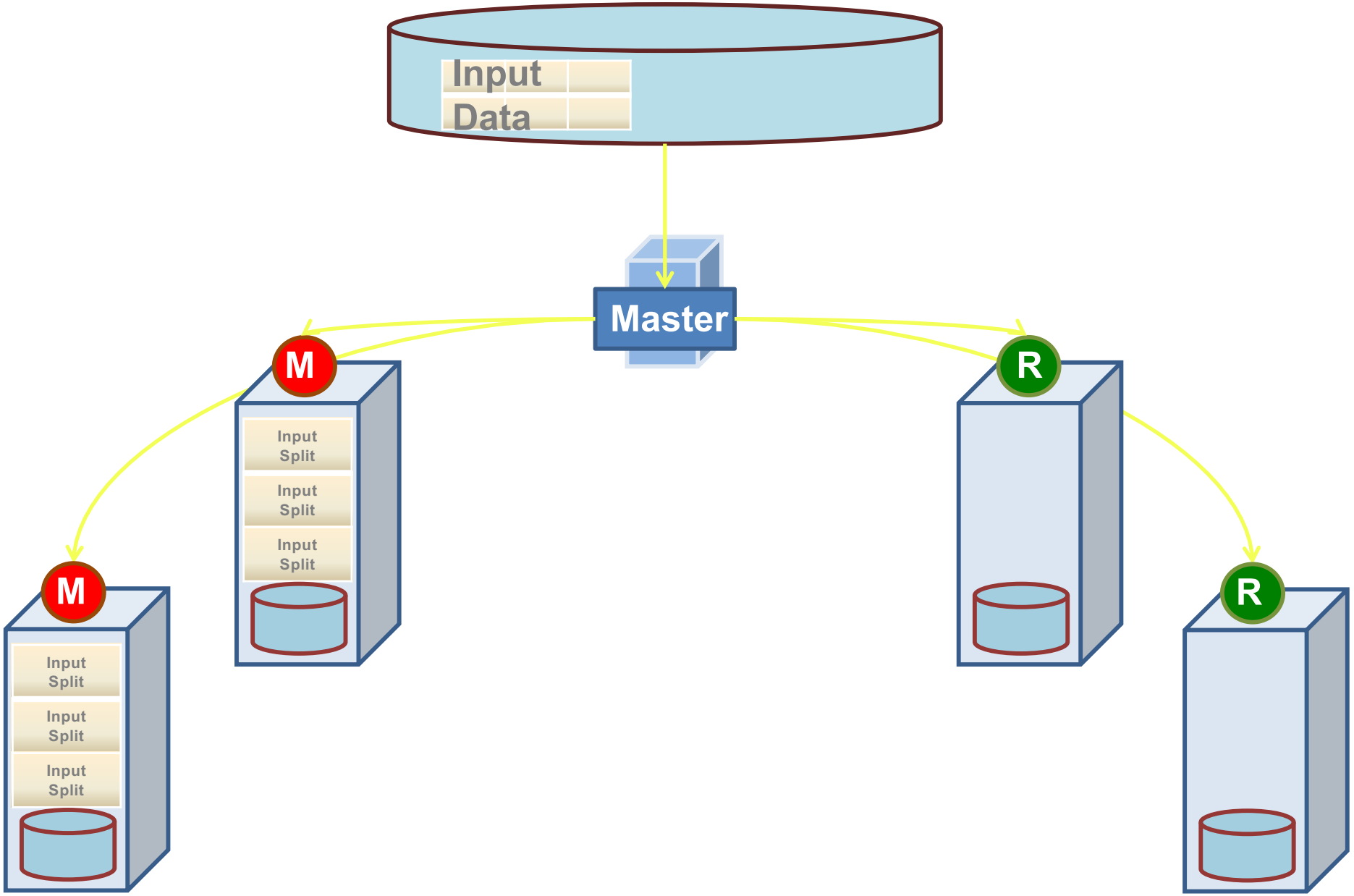
Storing data: HDFS



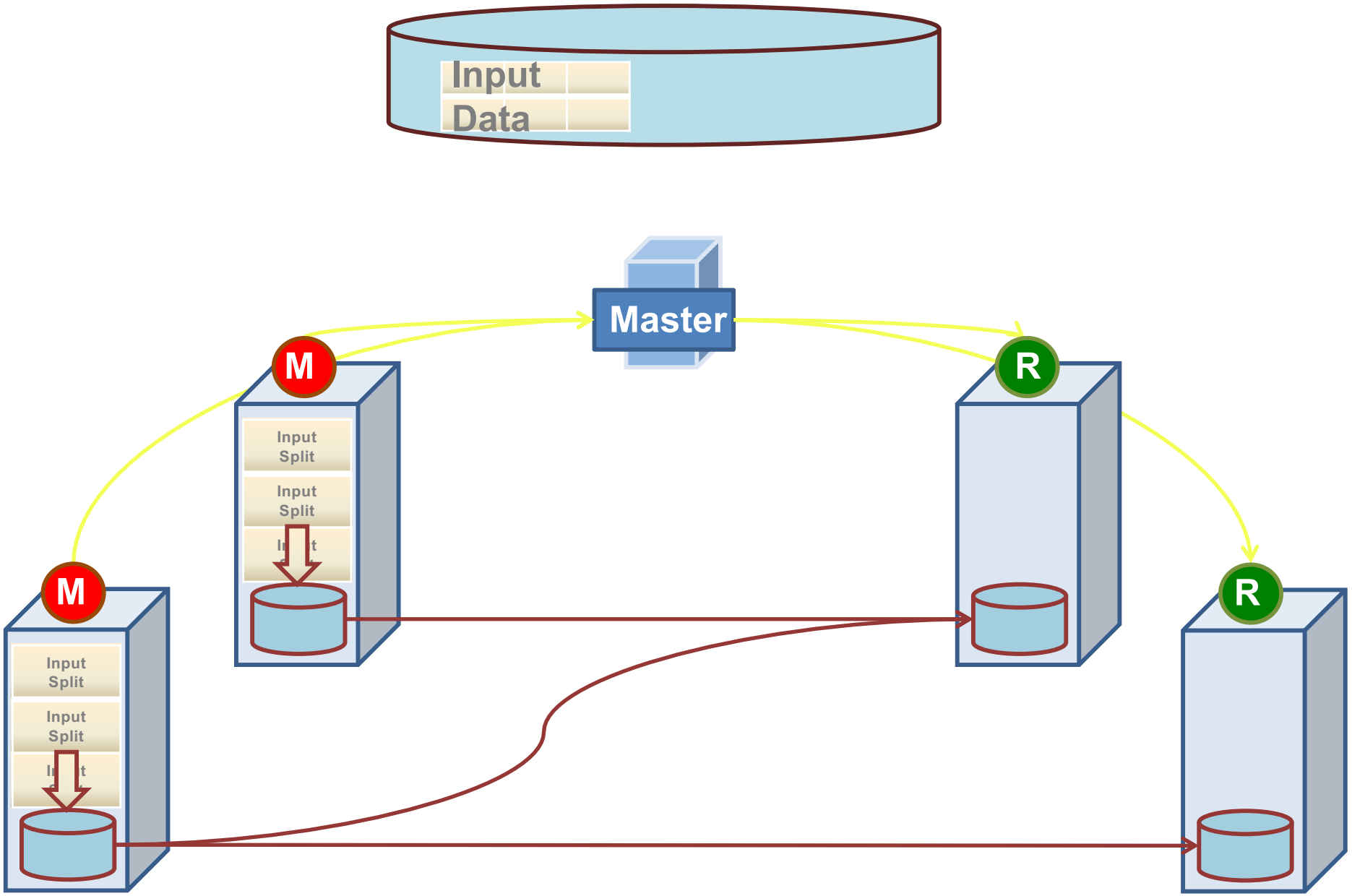
System view – Execution (Hadoop 1.0)



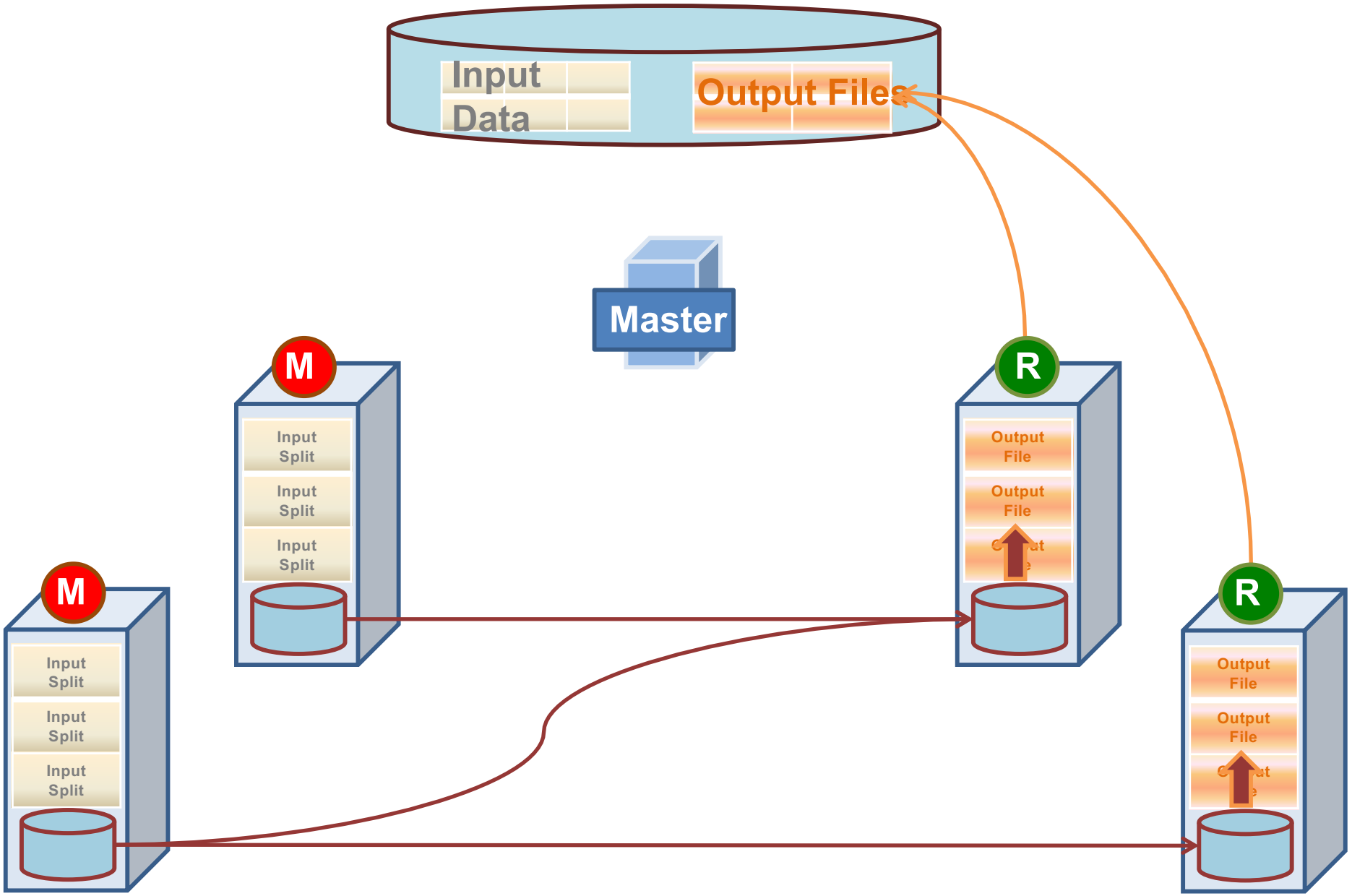
System view – Execution (Hadoop 1.0)



System view – Execution (Hadoop 1.0)



System view – Execution (Hadoop 1.0)



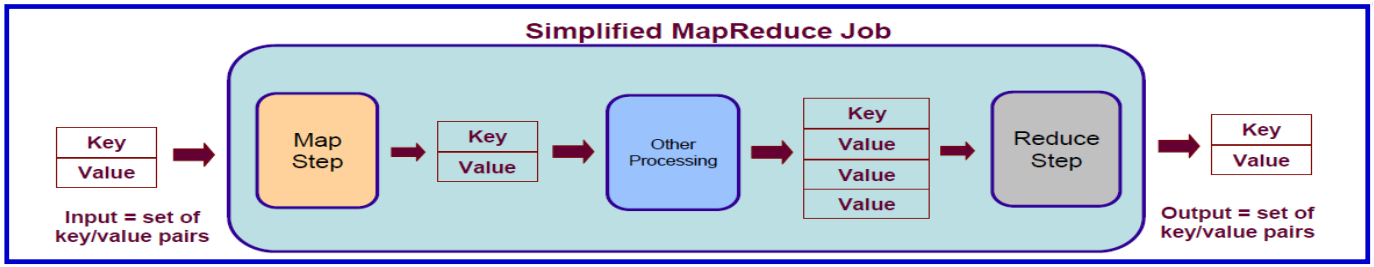
Failures

- Master **pings workers** periodically to detect failures
- Map worker failure:
  - Possible intermediate data loss
  - Map tasks restarted on another node with the same input splits
- Reduce worker failure:
  - Reducer tasks are restarted on another node
- **Highly fault-tolerant framework** (1.2 failures per analysis job at Google)

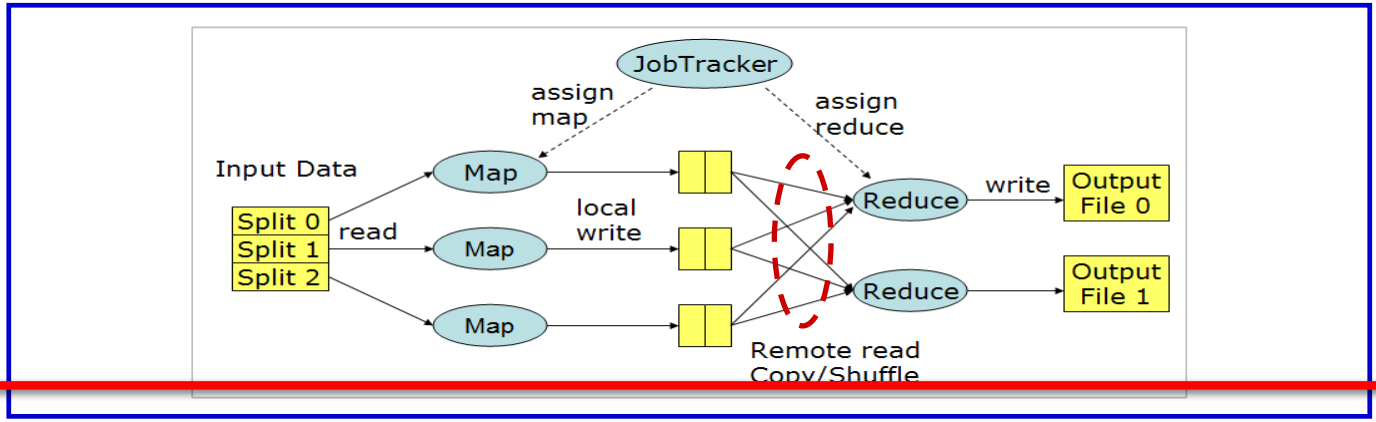
Map-reduce overview

Courtesy of IBM

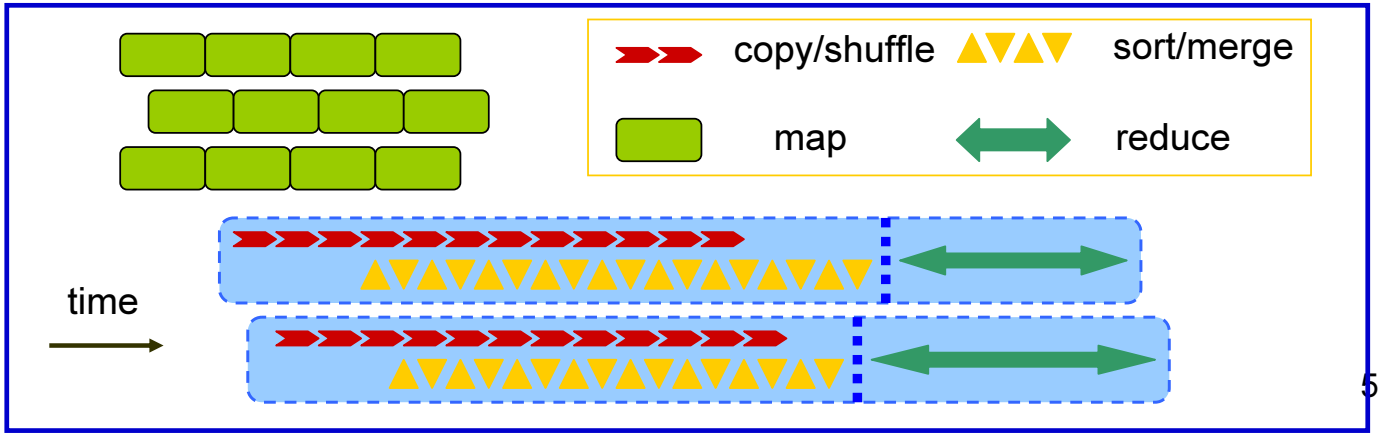
Function View



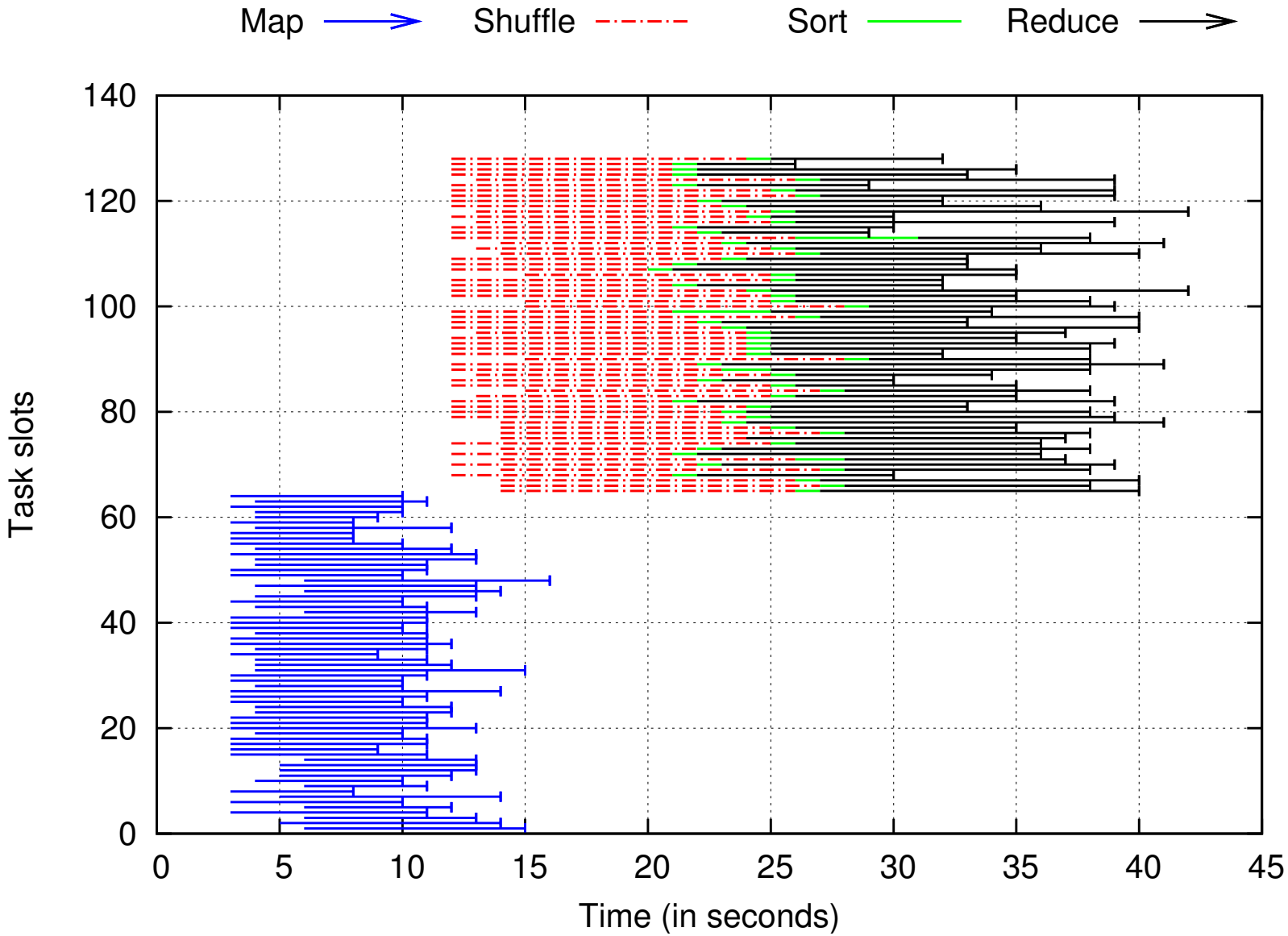
System View



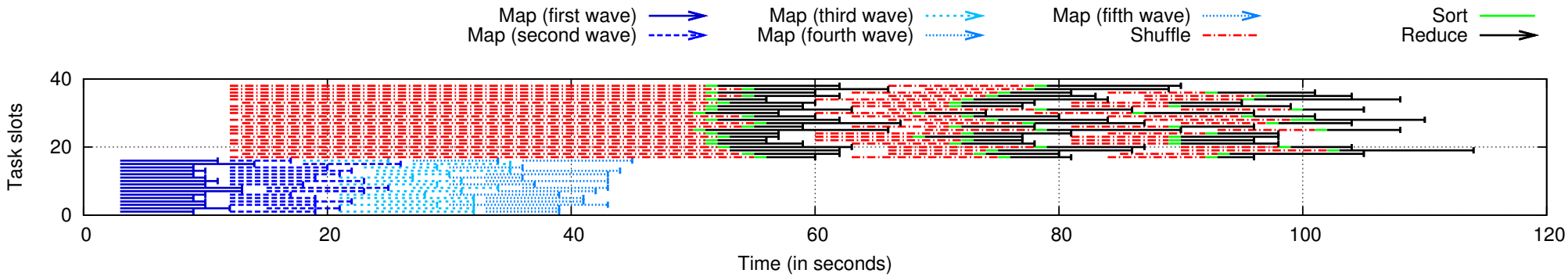
Process View



Process view

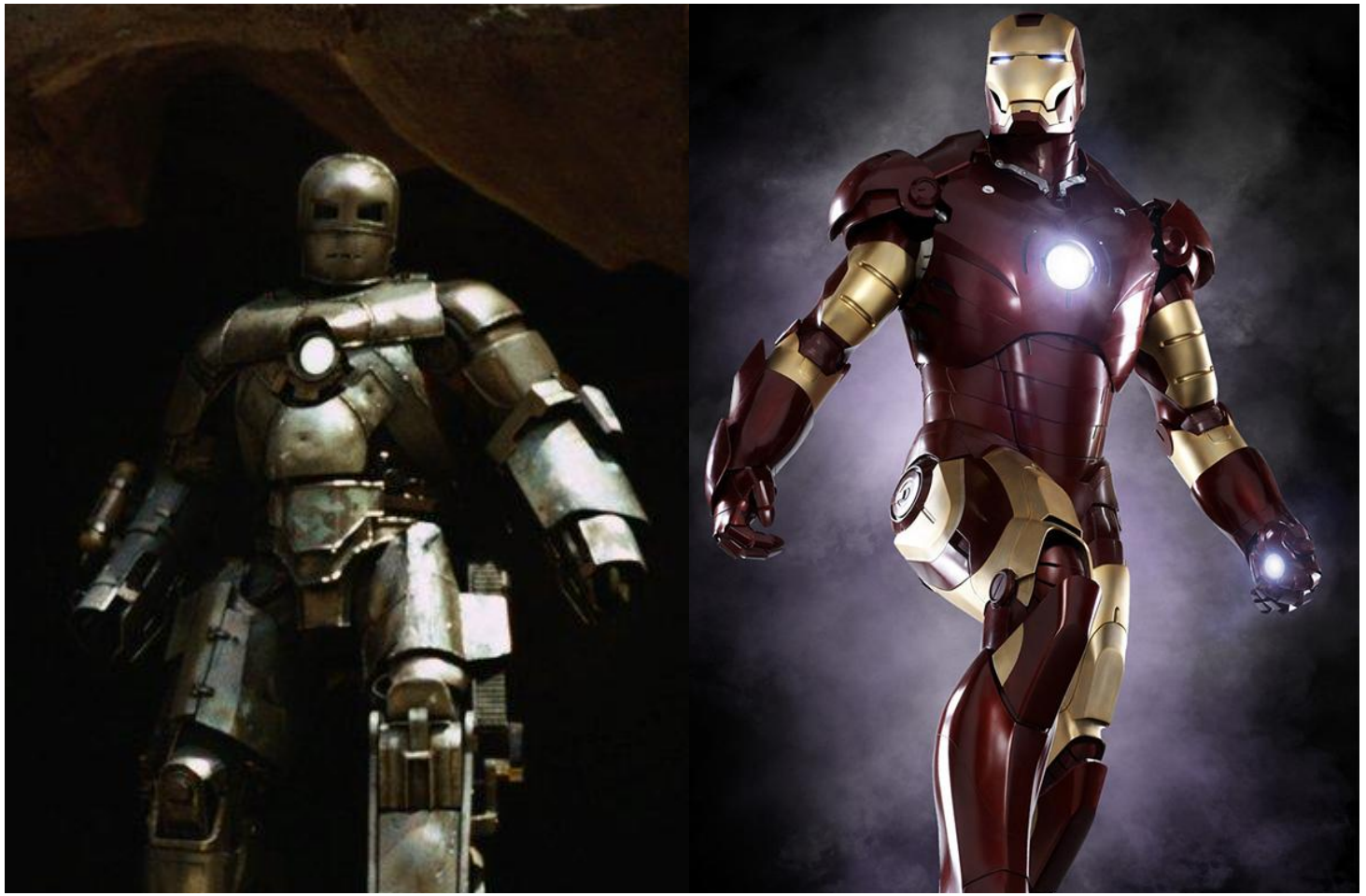


Process view



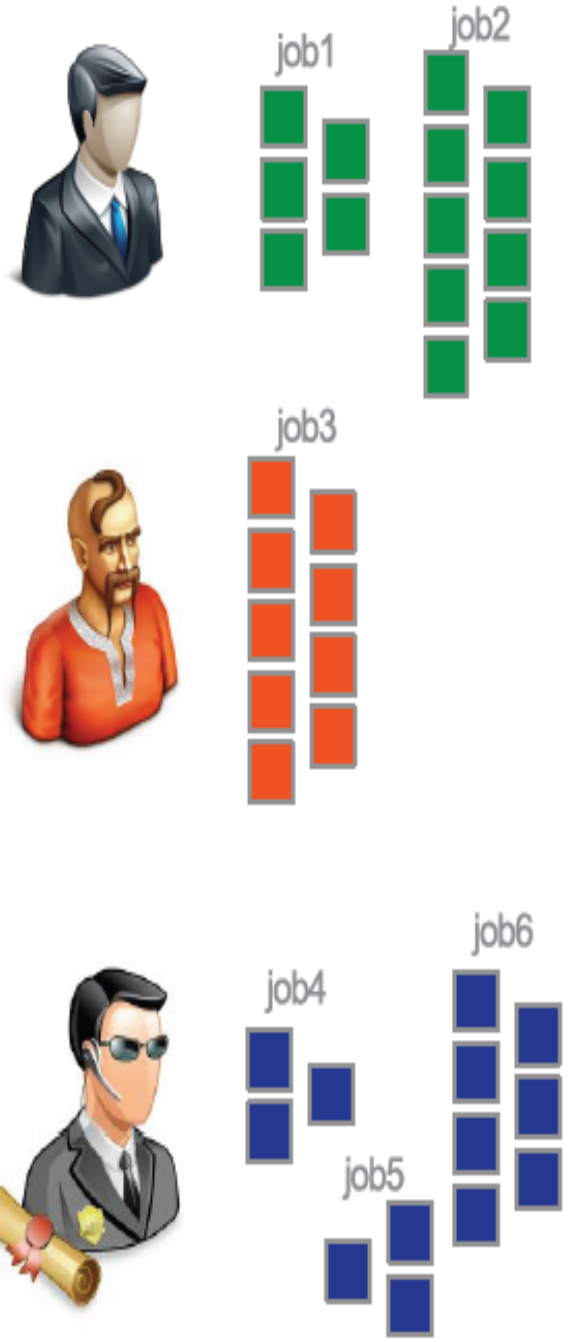
Source: A. Verma, L. Cherkasova, R.H. Campbell.
ARIA: Automatic Resource Inference and Allocation for mapreduce environments, ICAC 2011.

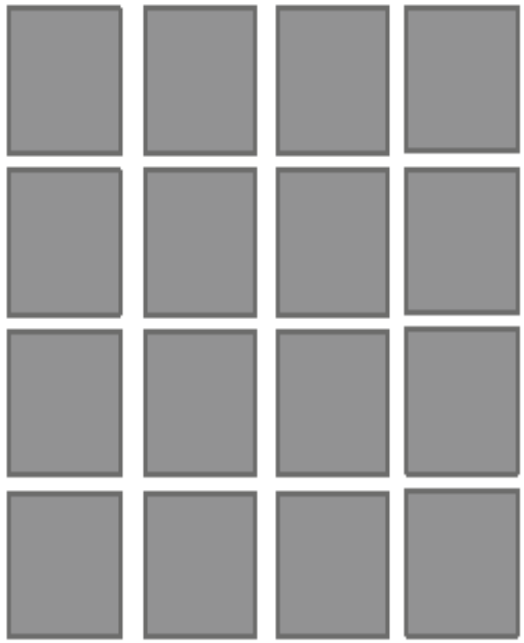
Hadoop evolution



Hadoop 1.0 → Hadoop 2.7.3

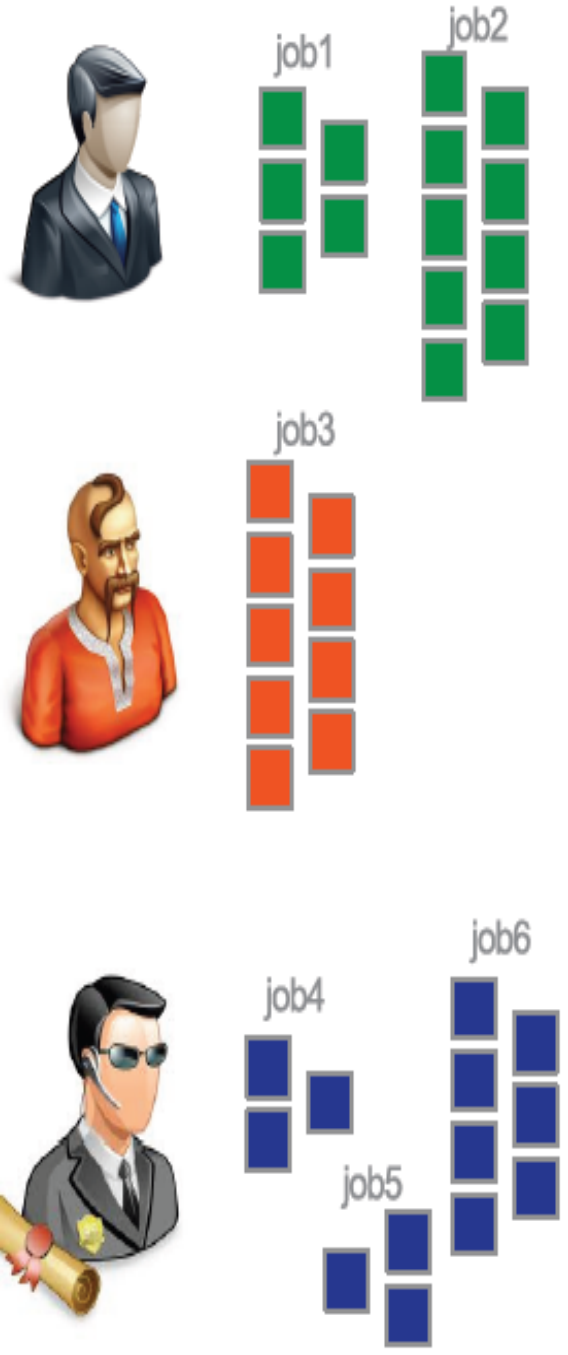
Hadoop 1.0





Courtesy of Microsoft

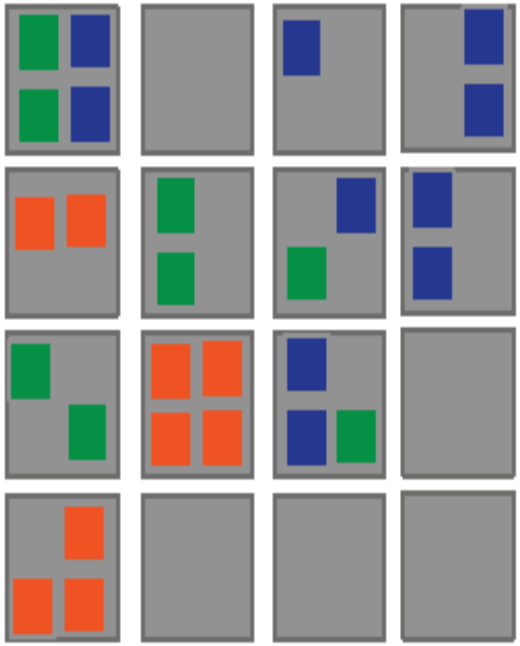
Hadoop 1.0



Problems to solve

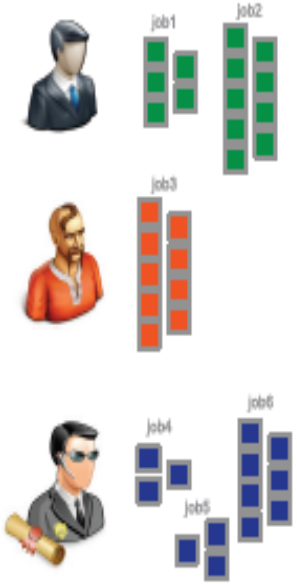
- Who runs?
- Where?
- How much resources?
- Order of execution
- Monitor progress
- Handle failures

Large Shared Cluster



Courtesy of Microsoft

Hadoop 2.x



Resource Management

- Who runs?
- Where?
- How much resources?

→ Goals

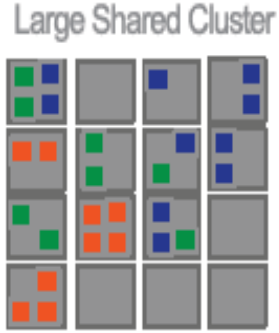
- Cluster throughput
- Scheduling Invariants
- Efficiency (locality)

Application Workflow

- Order of execution
- Monitor progress
- Handle failures

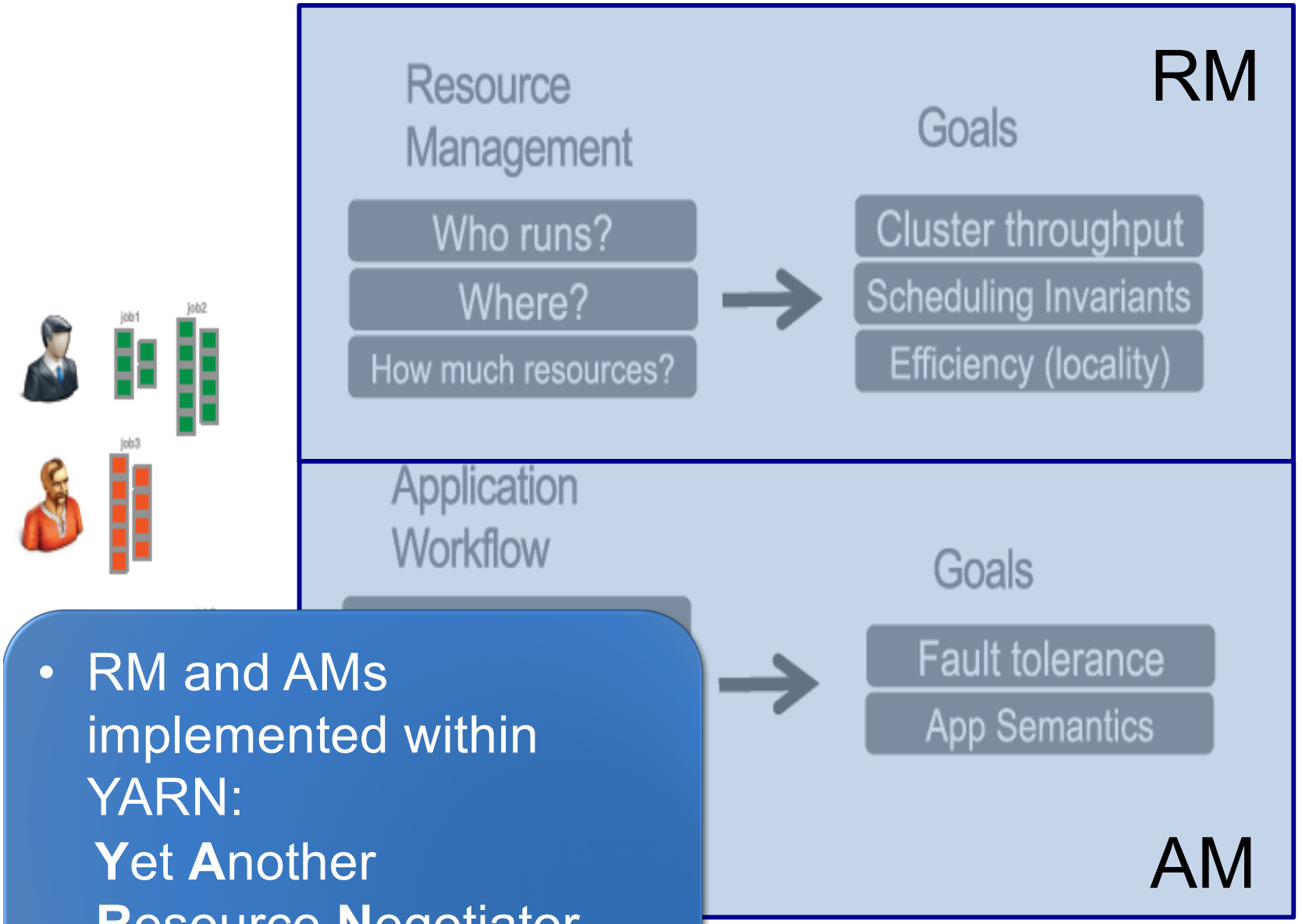
→ Goals

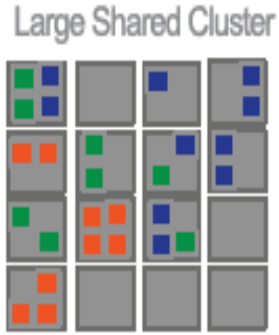
- Fault tolerance
- App Semantics



Courtesy of Microsoft

Hadoop 2.x

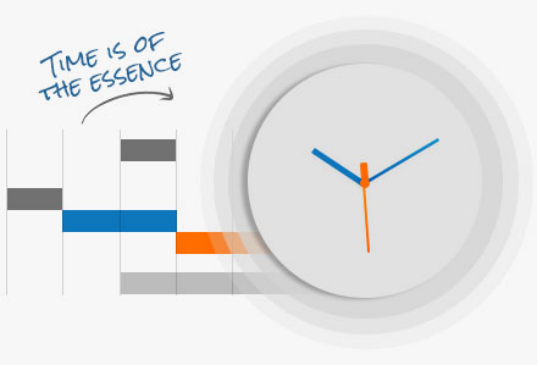




- RM and AMs implemented within YARN:
  Yet **A**nother **R**esource **N**egotiator

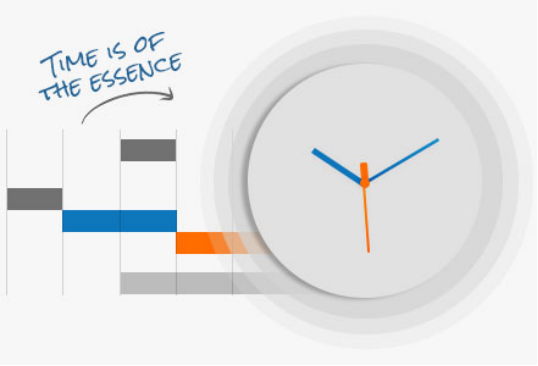
Courtesy of Microsoft

Scheduling



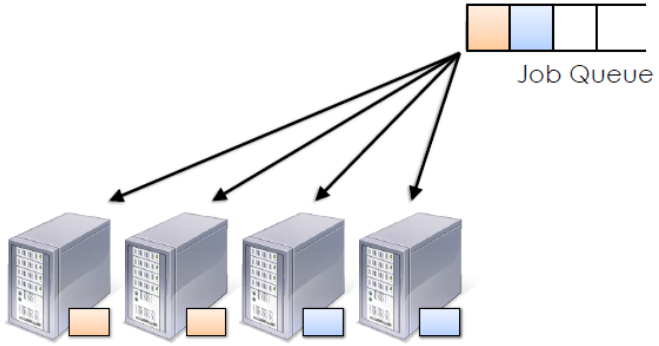
- Hadoop framework executes its task based on **runtime scheduling** scheme
- This means that **no execution plan** is built a priory (huge difference wrt. databases)
- In this way **fault tolerance** and **load balancing** can be achieved
- **Speculative** and **redundant execution** can also be used

Scheduling



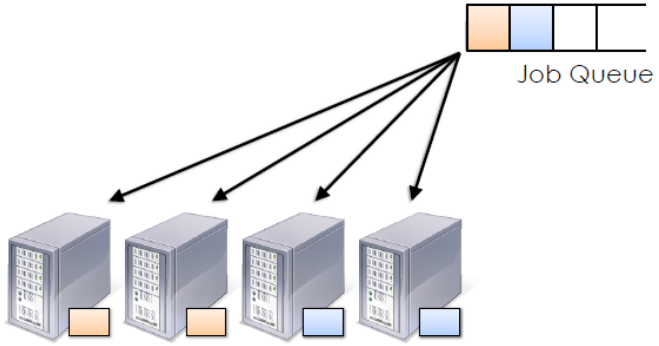
- **Map** and **reduce** tasks can be executed **without communication among other tasks**, no contention and synchronization cost between running jobs
- The first **scheduler** implementation was **First In First Out** (FIFO)
- Hadoop 2.x:
  - **Fair** and **Capacity** schedulers
  - **Work conserving preemption**

Fair Scheduler



- Assigning resources to jobs such that all jobs get, **on average, an equal share** of resources over time
- A single job running uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs
- Jobs organized into **pools**
  - Assign each pool a **guaranteed minimum** share
  - Divide **excess** capacity **evenly** between pools
  - Extensions: pools are organized **hierarchically**

Capacity Scheduler

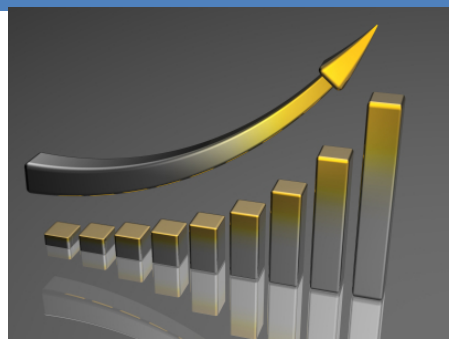


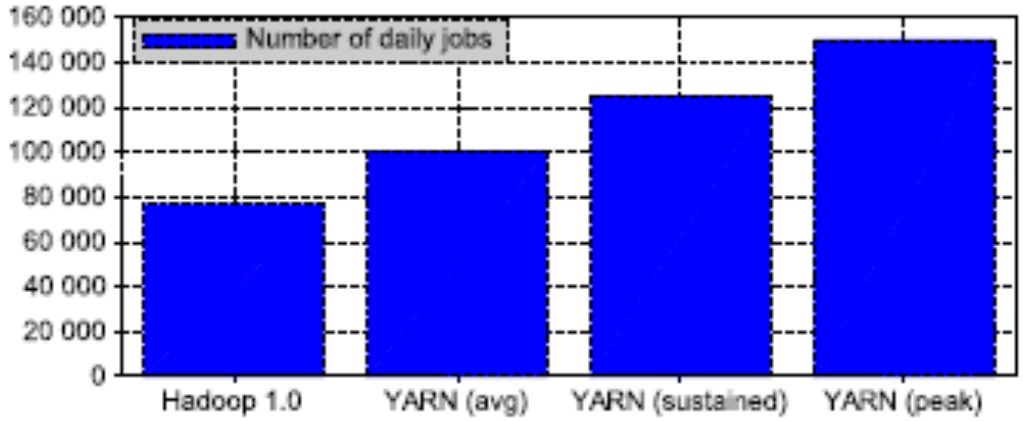
- Allow sharing a **large cluster** while giving each organization a **minimum capacity guarantee**
- Organizes **jobs into queue hierarchy**
- **Queue shares** as %'s of cluster
- **FIFO** scheduling within each queue but **also priority**

Capacity vs. Fair Schedulers

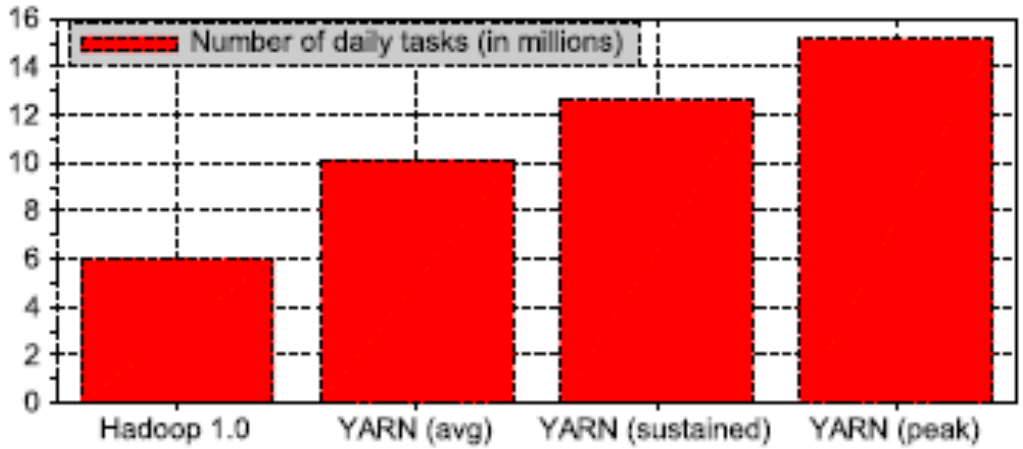
- Capacity scheduler:
  - Implements strict access controls to queues
  - Large Hadoop cluster, with multiple clients and different types and priorities of jobs
- Fair scheduler: both small and large clusters used by the same organization with a limited number of workloads

YARN performance improvements





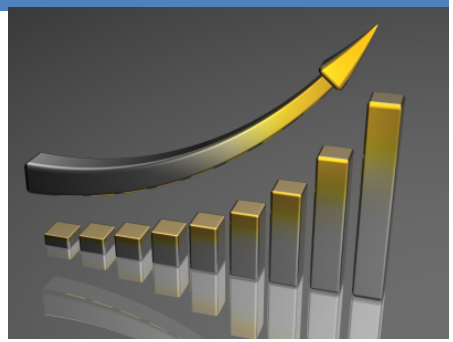
(a) Daily jobs

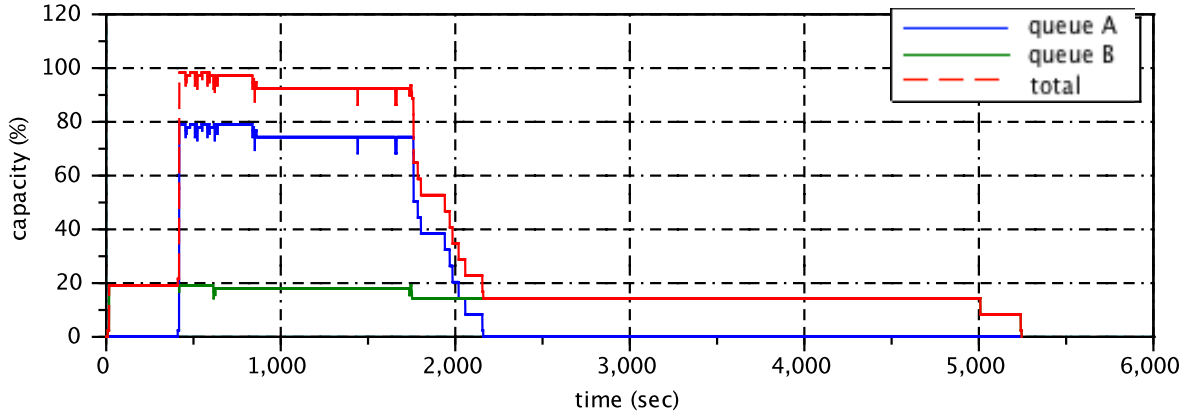


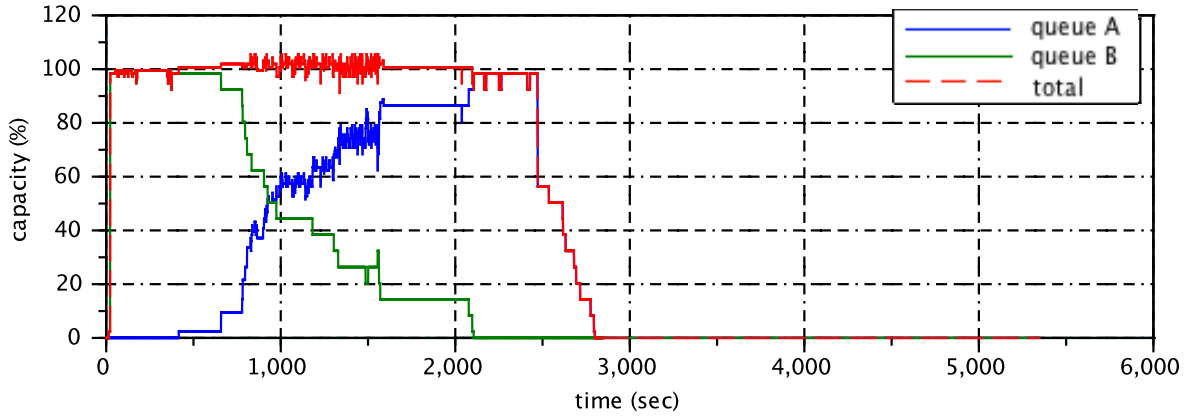
(b) Daily tasks

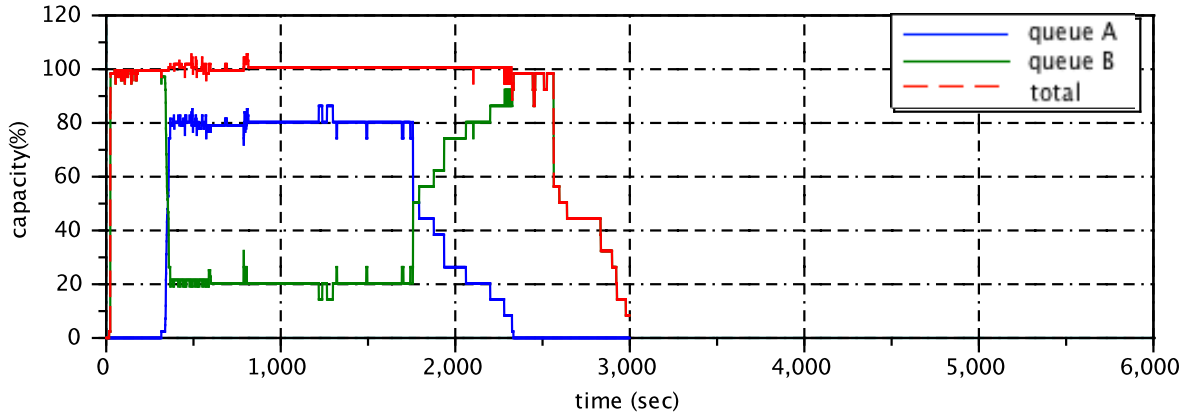
Source: V. K. Vavilapalli et al. Apache Hadoop YARN: Yet Another Resource Negotiator. SoCC'13, Oct. 2013

YARN performance improvements





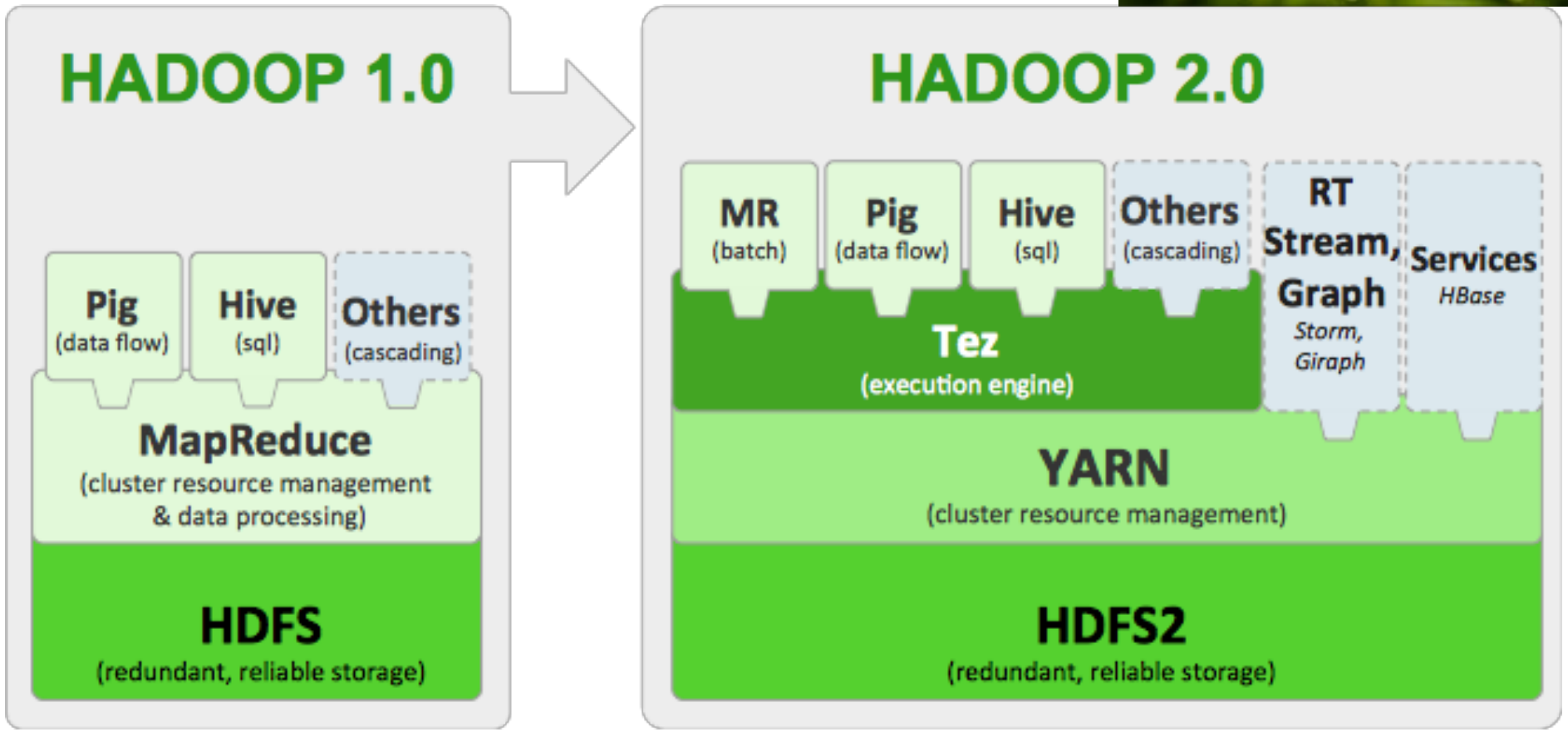




Performance degradation

- The network is a bottleneck:
  - Copy shuffle phase limit jobs execution performance
  - Network optimizations
    - load balancing across multiple paths
    - traffic prioritization and isolation
    - data aggregation and compression
- The disk is a bottleneck:
  - Caching data in memory
- Stragglers:
  - Tasks that take much longer to complete than other tasks in the job
  - Outliers caused by poorly performing machines that cause tasks scheduled on them to take longer
  - Work may have been unevenly divided across tasks, skews

Hadoop Eco-system



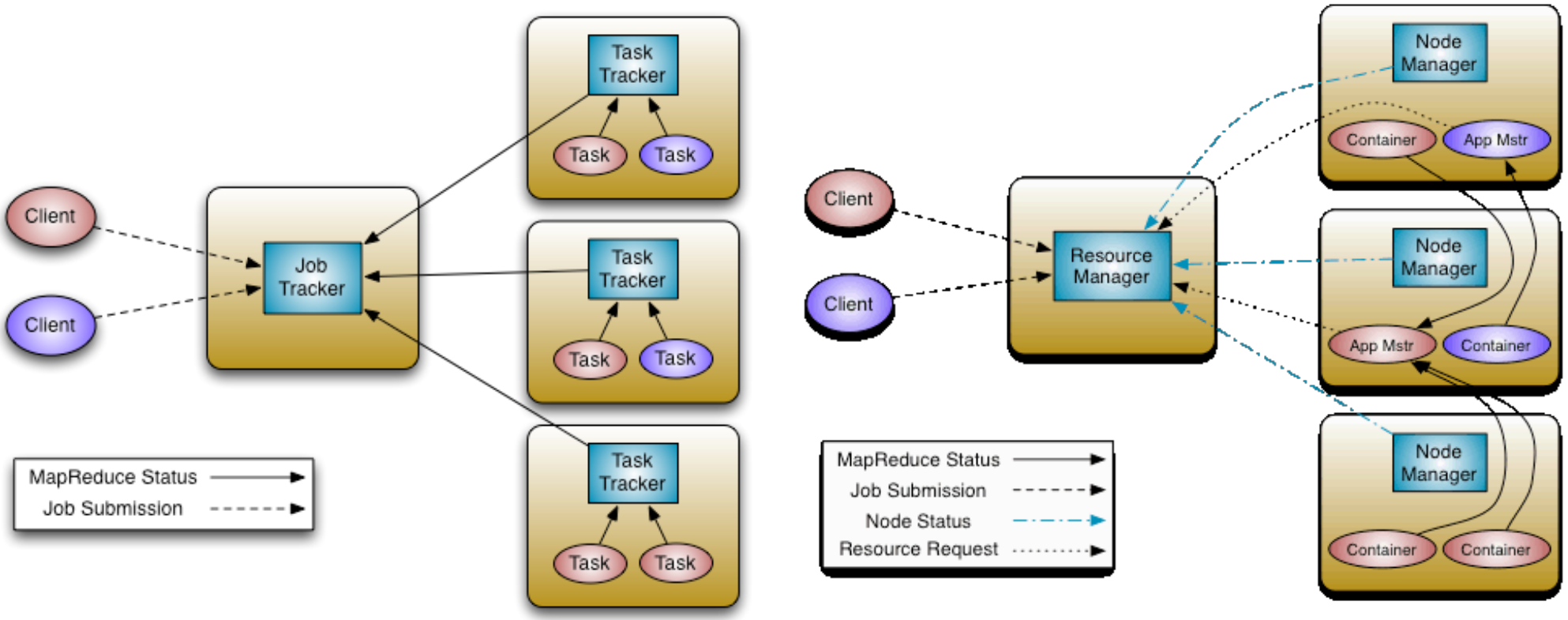
https://www.youtube.com/watch?v=Z5kQR71yJpE

http://www.tableausoftware.com

Pig and Hive

Computational nodes architecture

- Job and task trackers have been replaced
- YARN: Yet Another Resource Negotiator

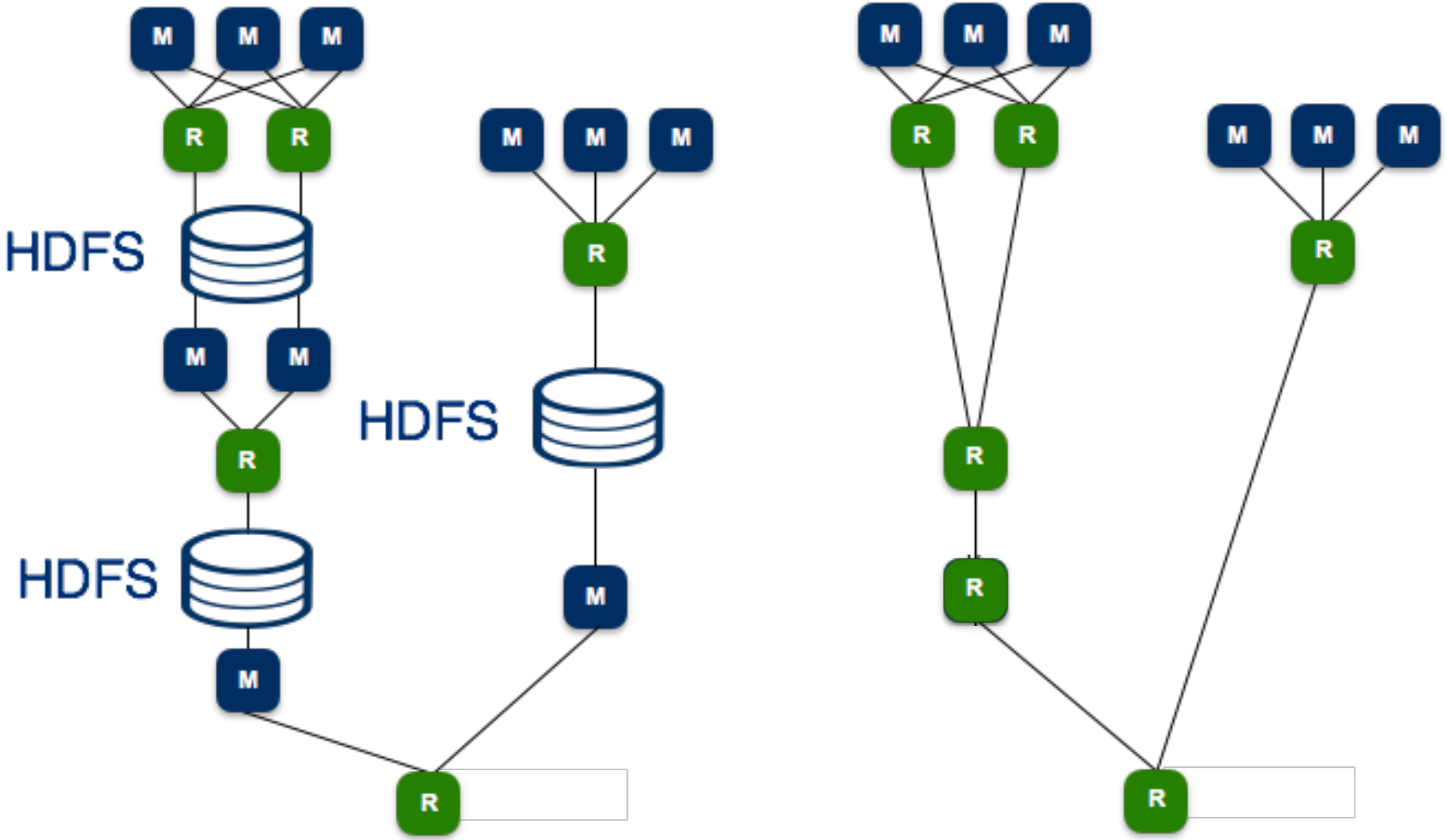


Computational nodes architecture

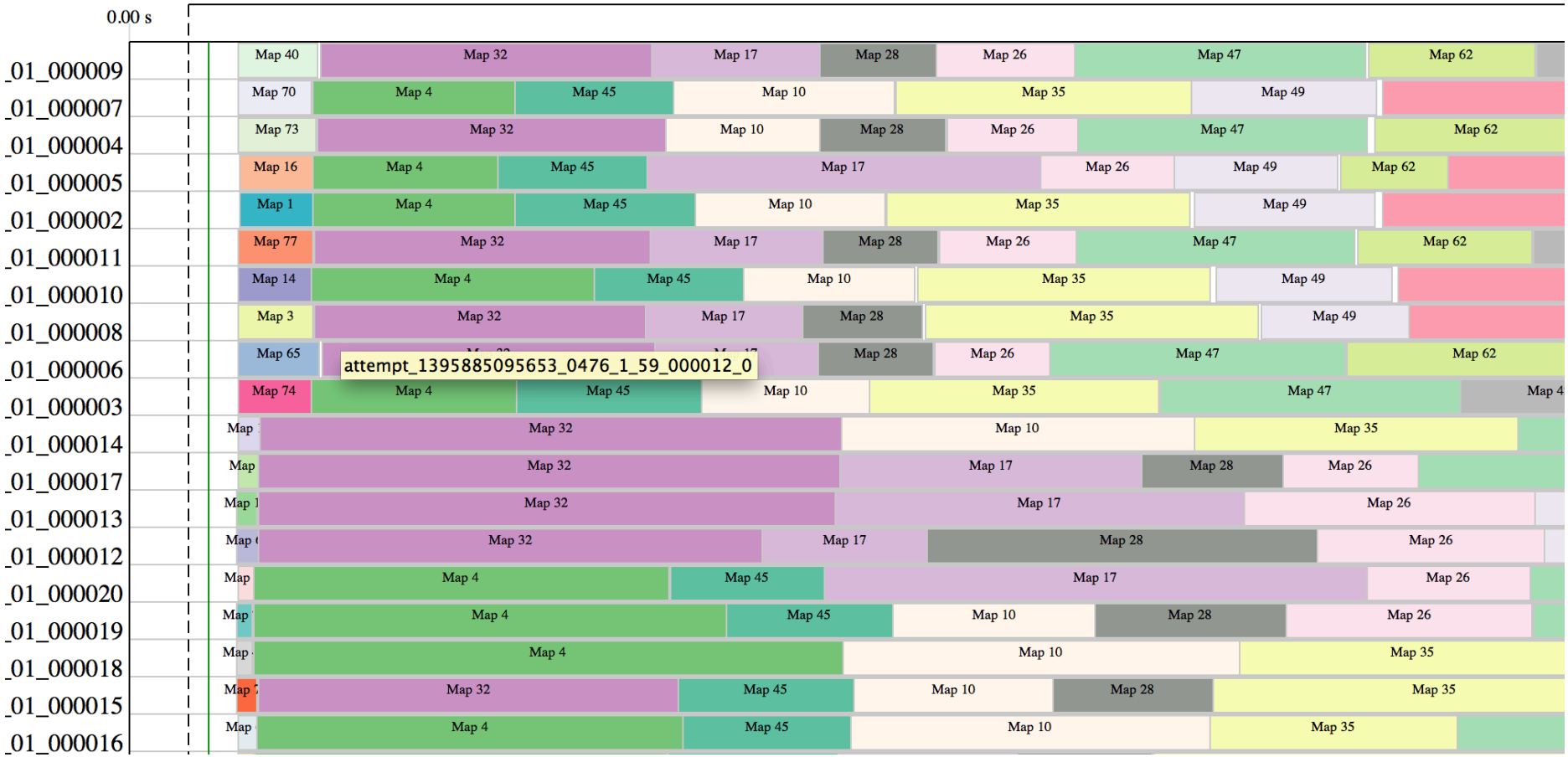
- Resource Manager:
  - Global resource scheduler
  - Hierarchical queues
- Application Master:
  - Per-application
  - Manages application scheduling and task execution
- Node Manager:
  - Per-machine agent
  - Manages the life-cycle of containers
  - Container resource monitoring

- Generic DAGs and dynamic resource allocation

Generic DAGs (Apache Tez)

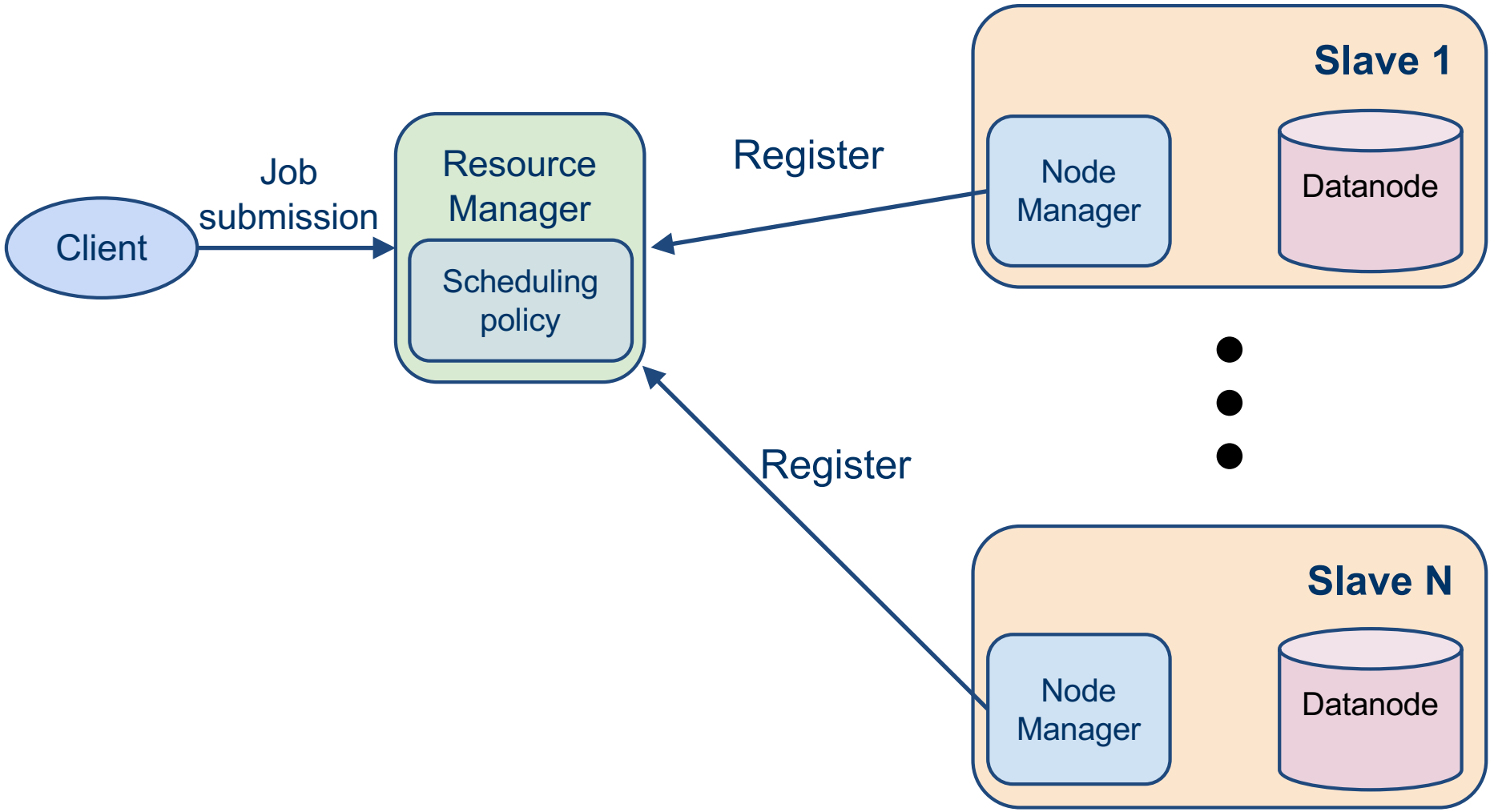


YARN Dynamic resource allocation

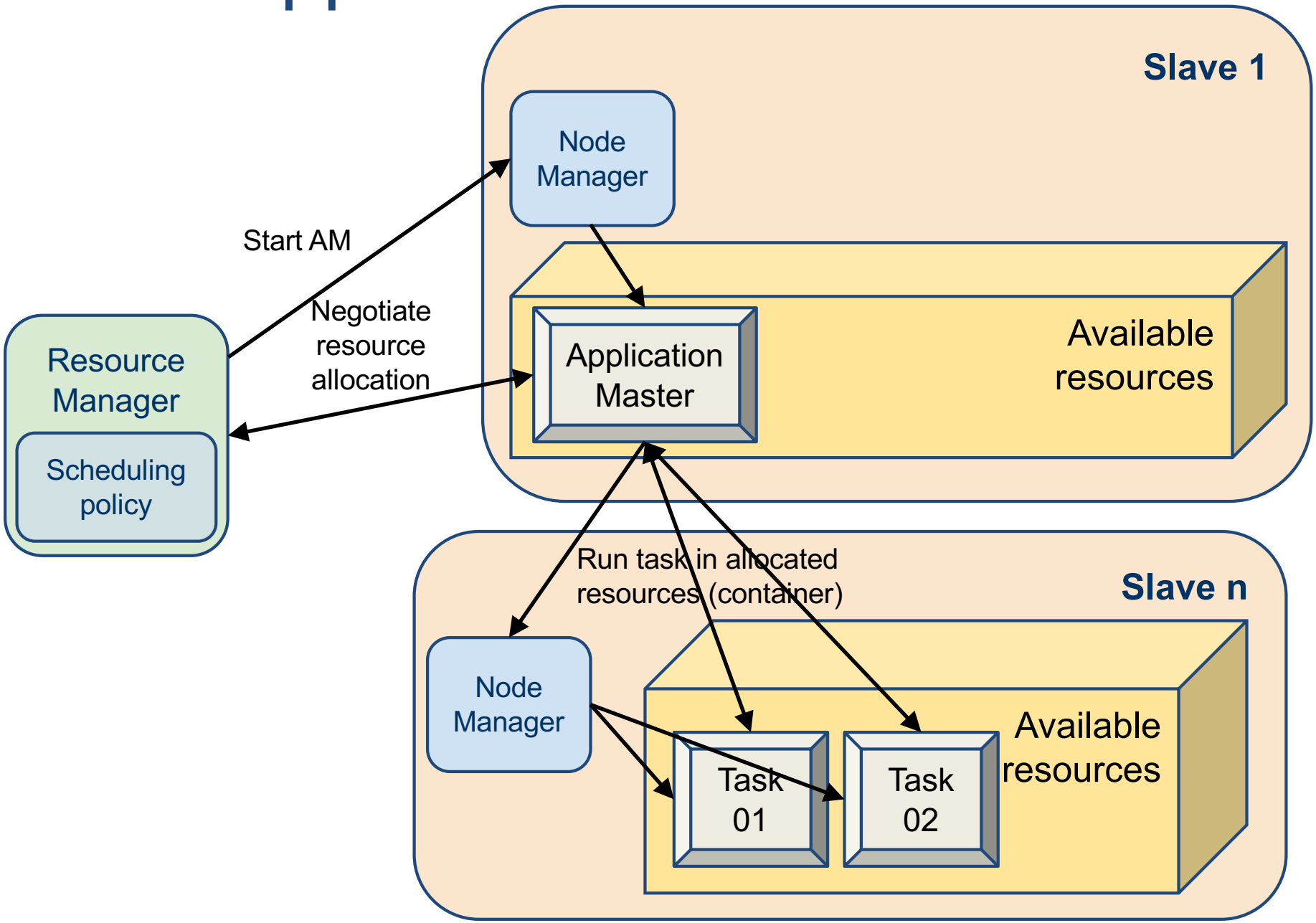


Bikas Saha Apache Tez: Accelerating Hadoop Data Processing
http://www.slideshare.net/hortonworks/apache-tez-accelerating-hadoop-query-processing

YARN application launch



YARN application launch



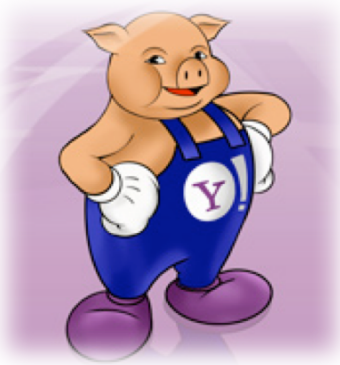
Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code

- Solution: develop higher-level data processing languages
  - Pig: Pig Latin is a bit like Perl
  - Hive: HQL is like SQL

Pig and Hive

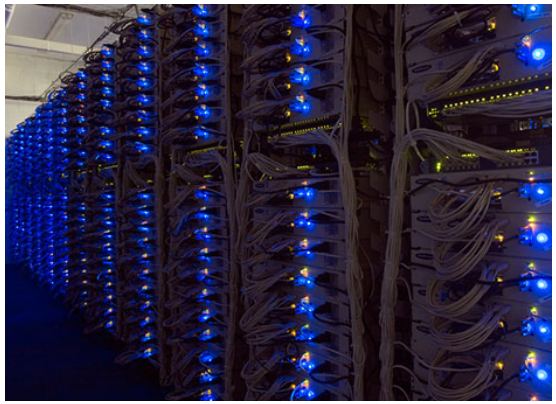
- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source

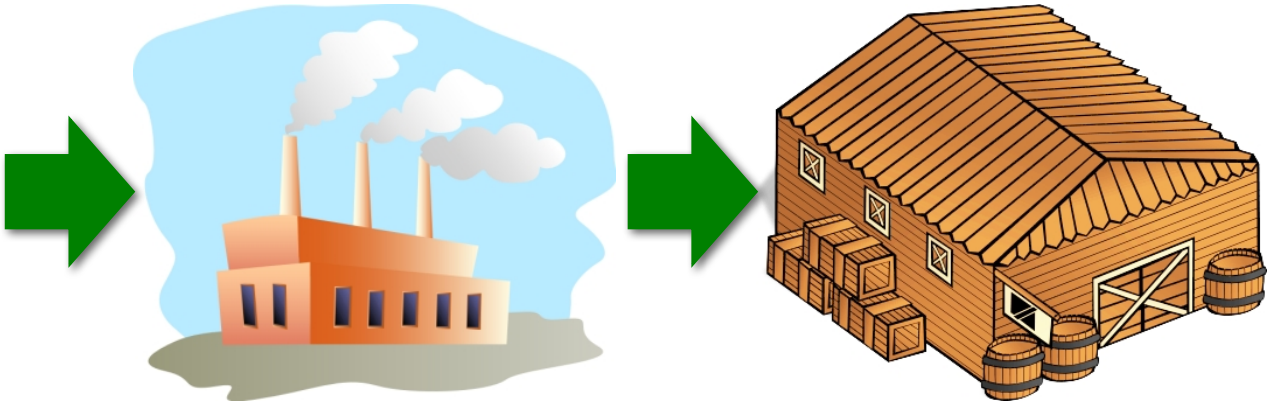


- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source

- Common idea:
  - Higher-level language “compiles down” to Hadoop jobs

Big data analysis workflow





Data Collection

Data Factory
Pig

Pipelines
Iterative Processing

Data Warehouse
Hive

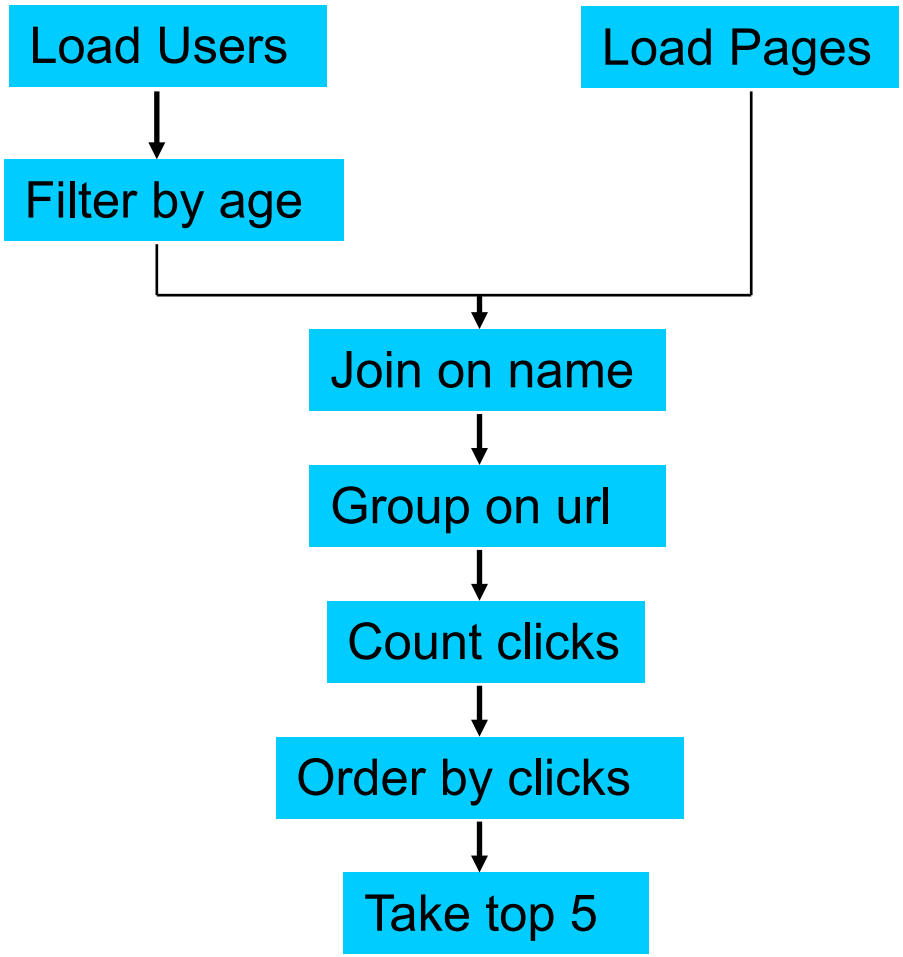
BI Tools
Analysis

High level languages: Pig

- Pig's language (**Pig Latin**) which has the following key properties:
  - **Ease of programming:** It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated **data transformations** are explicitly encoded as **dataflow sequences**, making them easy to write, understand, and maintain
  - **Optimization opportunities:** The way in which tasks are encoded permits the system to **optimize** their **execution** automatically, allowing the user to focus on semantics rather than efficiency
  - **Extensibility:** Users can **create** their **own functions** to do special-purpose processing

Motivation by example

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited pages by users aged 18 - 25.



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();
    }
}
```

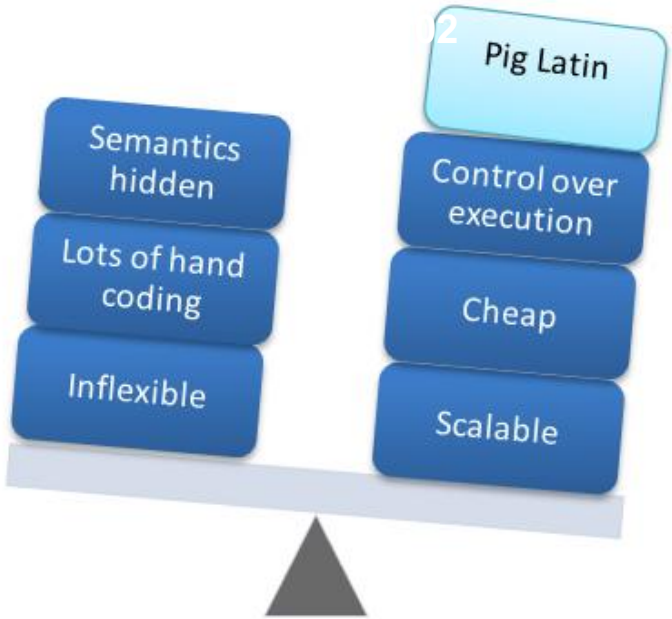
In Pig Latin

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

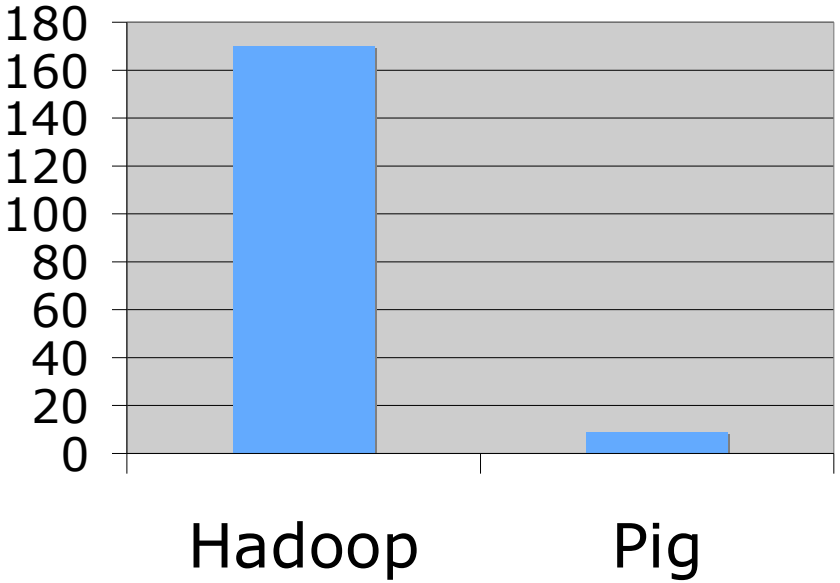
Word Count using Pig

```
Lines=LOAD 'input/hadoop.log' AS (line:
chararray);
Words = FOREACH Lines GENERATE
FLATTEN(TOKENIZE(line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE  group,
COUNT(Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO '/output/top5words';
```

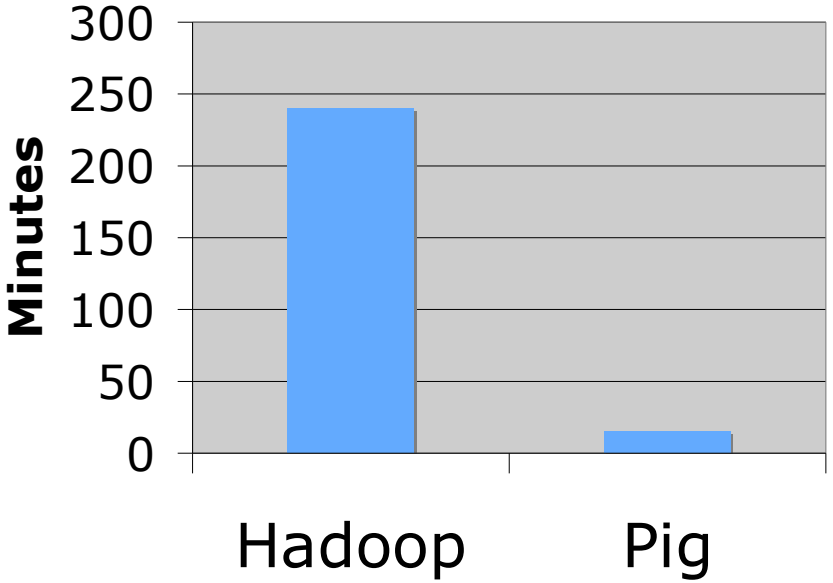
Java vs. Pig Latin



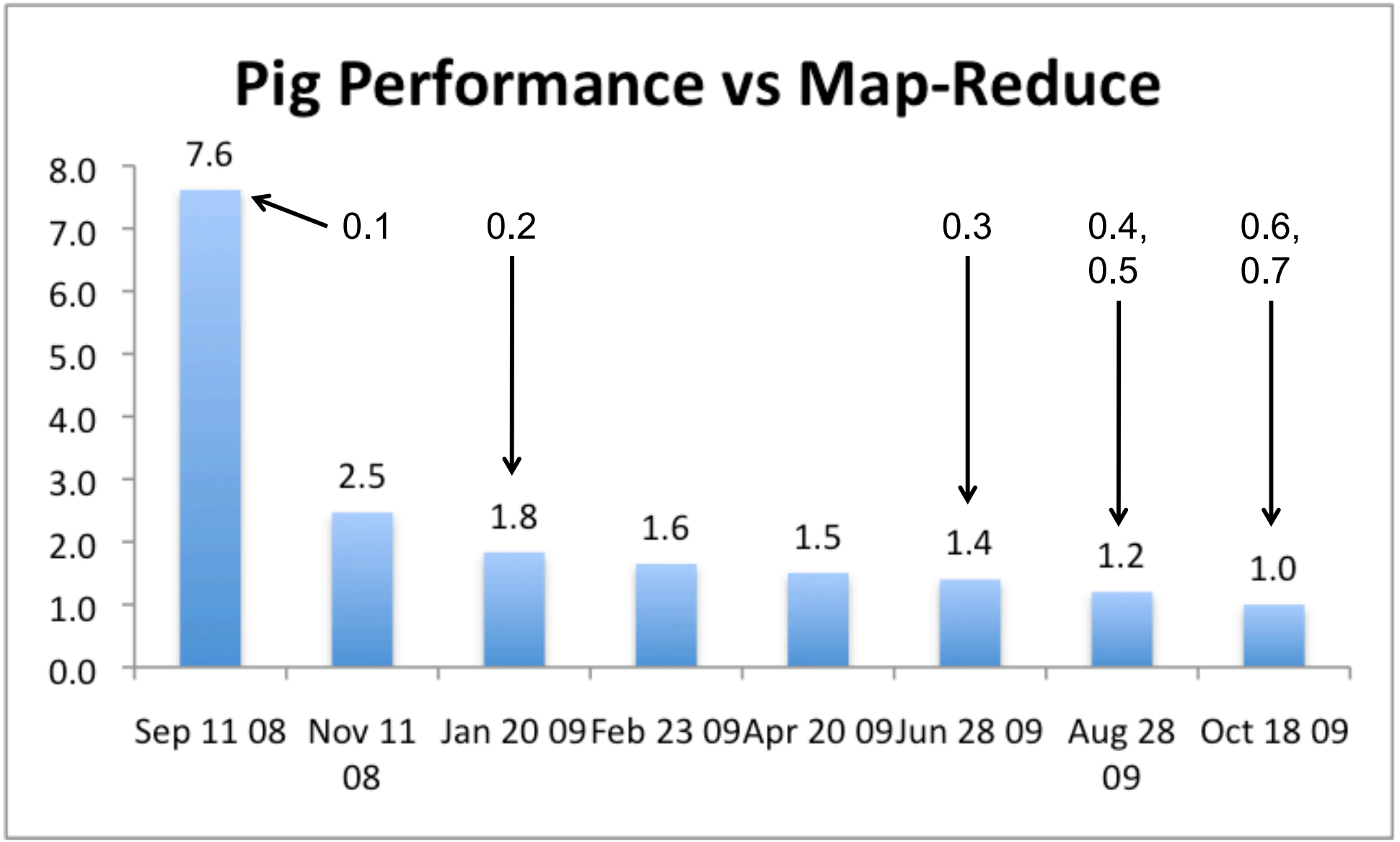
1/20 the lines of code



1/16 the development time



Performance



High level languages: Hive

- A **data warehouse** system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the **analysis** of large **datasets** stored in Hadoop compatible file systems
- Make the **unstructured data** looks **like tables** regardless how it really lay out
- Provides a mechanism to project structure onto this data and query the data using a **SQL-like language** called **HiveQL**
- At the same time this language also allows traditional map/reduce programmers to plug in their **custom mappers** and **reducers** when it is inconvenient or inefficient to express this logic in HiveQL

Hive Applications @Facebook

- Log processing
  - Daily Report
  - User Activity Measurement
- Data/Text mining
  - Machine learning
- Business intelligence
  - Advertising Delivery
  - Spam Detection

HiveQL

- It doesn't fully conform to any particular revision of the ANSI SQL standard (close to MySQL's dialect)
- No support for row-level inserts, updates, and deletes and transactions
- Still, much of HiveQL will be familiar

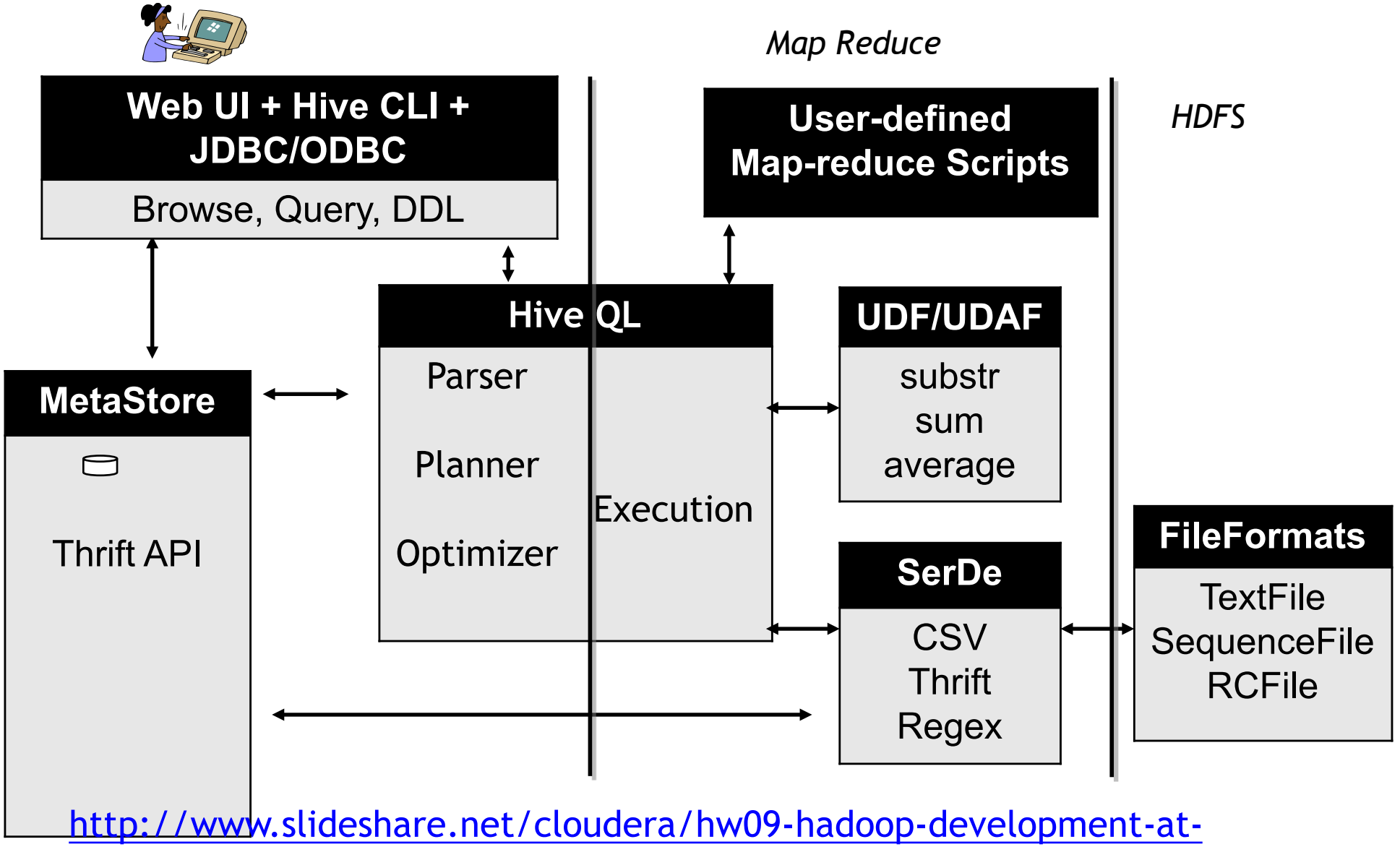
HiveQL

- Basic SQL
  - From clause subquery
  - ANSI JOIN (equi-join only)
  - Sampling
  - Objects traversal
- Extensibility
  - Pluggable Map-reduce scripts using TRANSFORM
  - UDFs and UDAFs

Data Model

- **Databases**: Namespaces separating tables and other data units from naming confliction
- **Tables**: Homogeneous units of data which have the same schema
- **Partitions**: Each Table can have one or more partition Keys which determines how the data is stored. Partitions also allow the user to efficiently identify the rows that satisfy a certain criteria
- **Buckets** (or **Clusters**): Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table

Architecture



http://www.slideshare.net/cloudera/hw09-hadoop-development-at-facebook-hive-and-hdfs

MetaStore

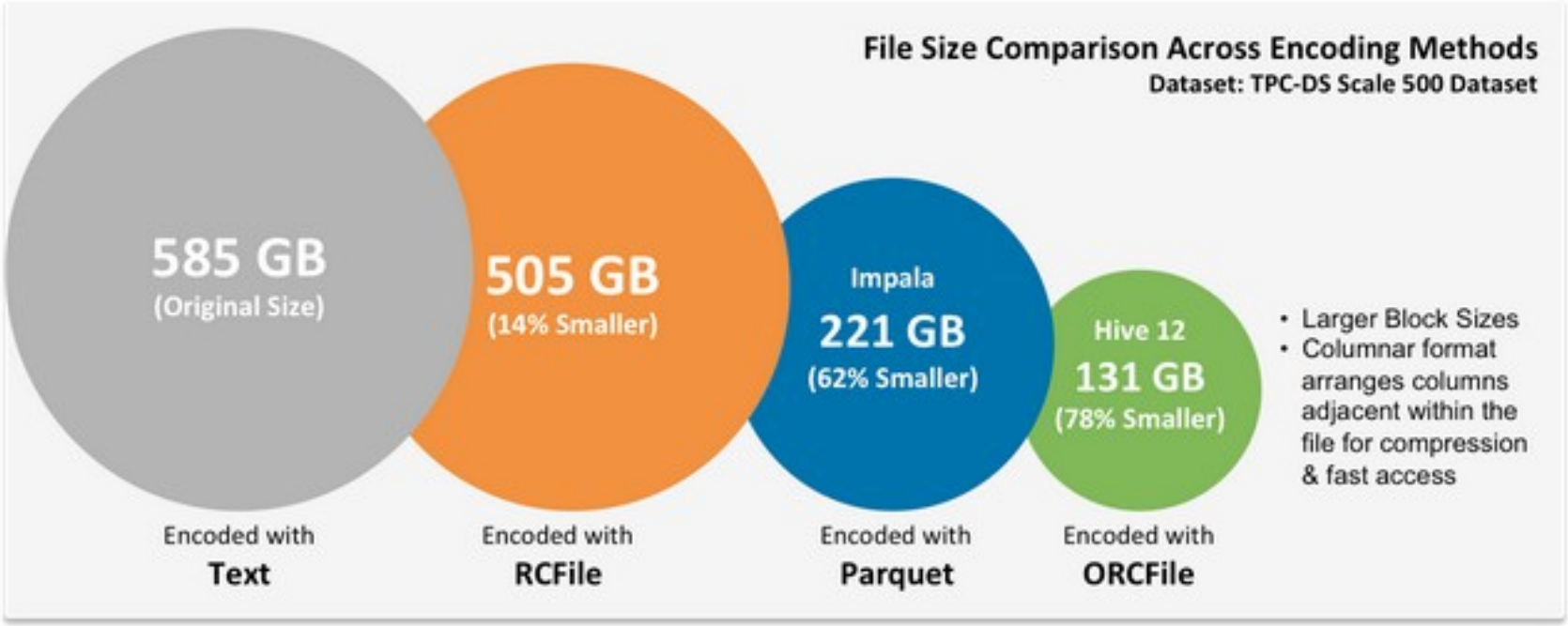
- Stores Table/Partition properties:
  - Table schema and SerDe library
  - Table Location on HDFS
  - Logical Partitioning keys and types
  - Partition level metadata
  - Other information
- Thrift API
  - Current clients in Php (Web Interface), Python interface to Hive, Java (Query Engine and CLI)
- Metadata stored in any SQL backend

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

Creating an external table

```
CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
exchange        STRING
symbol          STRING
ymd             STRING
price_open      FLOAT
price_high      FLOAT
price_low       FLOAT
price_close     FLOAT
volume          INT
price_adj_close FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/data/stocks';
```

Because it's external, Hive does not assume it *owns* the data.
Dropping the table *does not* delete the data (only *metadata* will be deleted)

Partitioned managed tables

- Our HR people often run queries with WHERE clauses that restrict the results to a particular country or to a particular *first-level subdivision* (e.g., *state* in the United States or *province* in Canada). Let's use *state* for simplicity
- Let's partition the data first by country and then by state
- We have redundant country and state information in the address field. We could remove them from address

Partitioned managed tables

```
CREATE TABLE employees (
name STRING,
salary FLOAT,
subordinates ARRAY<STRING>,
deductions MAP<STRING, FLOAT>,
address STRUCT<street:STRING, city:STRING, zip:INT>
)
PARTITIONED BY (country STRING, state STRING);
```

Partitioned managed tables

If we create this table in the mydb database, there will still be an *employees* directory for the table:

hdfs://master_server/user/hive/warehouse/mydb.db/employees

However, Hive will now create subdirectories reflecting the partitioning structure. For example:

.../employees/country=CA/state=AB
.../employees/country=CA/state=BC
...
.../employees/country=US/state=AL
.../employees/country=US/state=AK
...

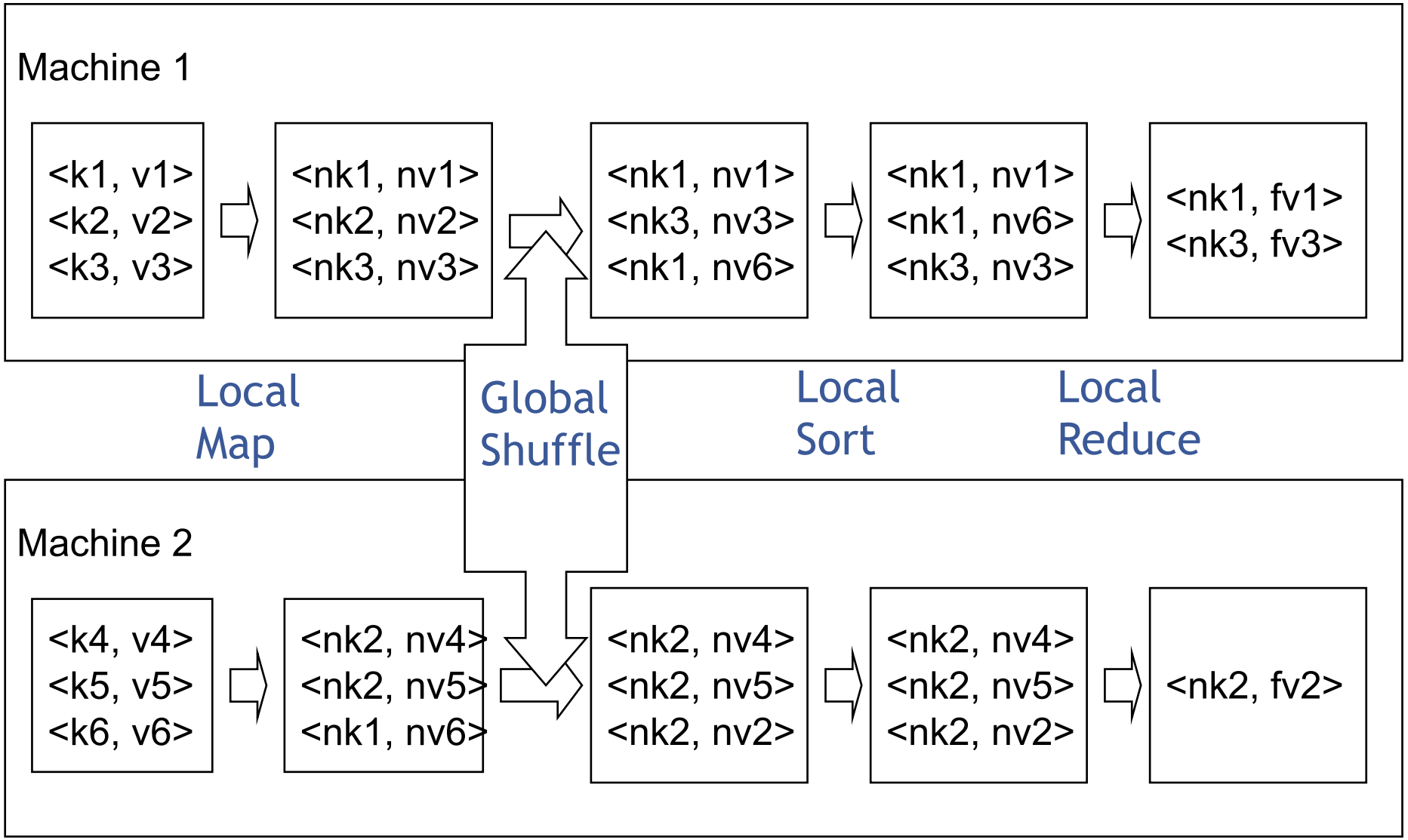
Partitioned managed tables

- Once created, the partition *keys* (country and state, in this case) behave like regular columns

```
SELECT *
FROM employees
WHERE country = 'US' AND state = 'IL';
```

- Of course, if you need to do a query for all employees around the globe, you can still do it. Hive will have to read every directory
- A query across all partitions could trigger an enormous MapReduce job
- A highly suggested safety measure is putting Hive into “strict” mode, which prohibits queries of partitioned tables without a WHERE clause that filters on partitions

(Simplified) Map Reduce



Hive QL – Join

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

X

user

userid	age	gender
111	25	female
222	32	male

=

pv_users

pageid	age
1	25
2	25
1	32

- SQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

Hive QL - Join in Map Reduce

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

user

userid	age	gender
111	25	female
222	32	male

Map

key	value
111	<**1**,1>
111	<**1**,2>
222	<**1**,1>

key	value
111	<**2**,25>
222	<**2**,32>

Shuffle
Sort

key	value
111	<**1**,1>
111	<**1**,2>
111	<**2**,25>

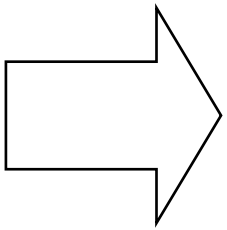
key	value
222	<**1**,1>
222	<**2**,32>

Reduce

Hive QL – Group By

pv_users

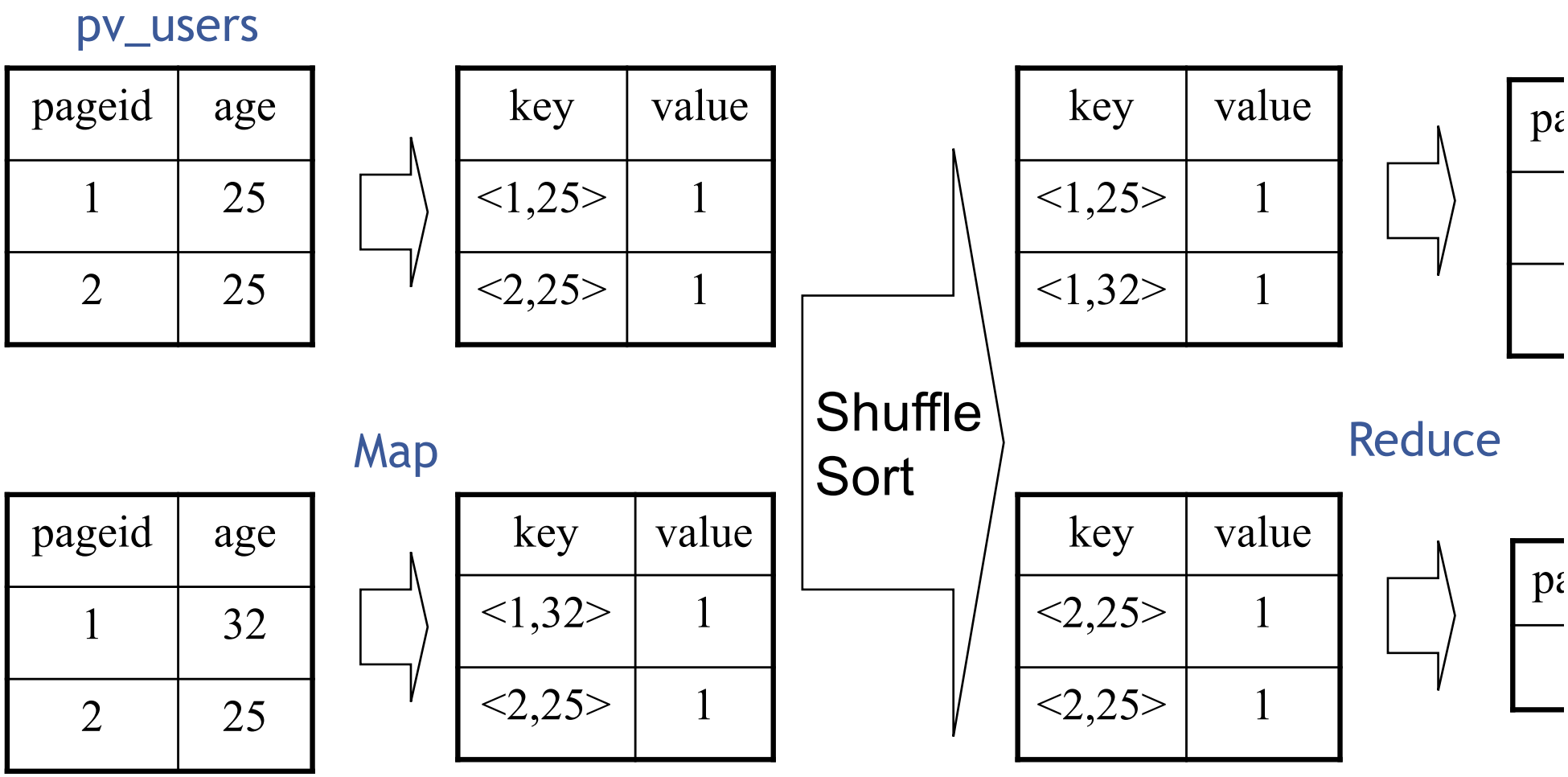
pageid	age
1	25
2	25
1	32
2	25



pageid	age	count
1	25	1
2	25	2
1	32	1

```
SELECT pageid, age, count(1)
FROM pv_users
GROUP BY pageid, age;
```

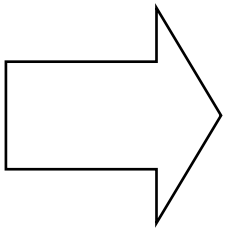
Hive QL – Group By in Map Reduce



Hive QL – Group By with Distinct

page_view

pageid	userid	time
1	111	9:08:01
2	111	9:08:13
1	222	9:08:14
2	111	9:08:20



pageid	count_distinct_userid
1	2
2	1

```
SELECT pageid, COUNT(DISTINCT userid)
FROM page_view GROUP BY pageid
```

Hive QL - Group By with Distinct in Map Reduce

page_view

pageid	userid	time
1	111	9:08:01
2	111	9:08:13

pageid	userid	time
1	222	9:08:14
2	111	9:08:20

Shuffle and Sort

key	v
<1,111>	1
<2,111>	1
<2,111>	1

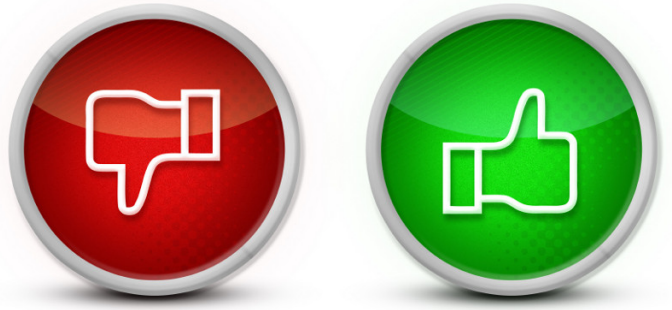
key	v
<1,222>	1

Reduce

pageid	count
1	1
2	1

pageid	count
1	1

Pros & Cons



- Pros:
  - A easy way to process large scale data
  - Support SQL-based queries
  - Programmability
  - Efficient execution plans for performance
  - Interoperability with other database tools

- Cons:
  - No easy way to append data
  - Files in HDFS are immutable

Pig vs. Hive

"I much prefer writing in Pig [Latin] versus SQL. The step-by-step method of creating a program in Pig [Latin] is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data."

-- Jasmine Novak, Engineer, Yahoo!

"PIG seems to give the necessary parallel programming construct (FOREACH, FLATTEN, COGROUP .. etc) and also give sufficient control (which purely declarative approach like [SQL on top of Map-Reduce] doesn't)."

-- Ricky Ho, Adobe Software

Pig vs. Hive

- PigLatin is procedural, where Hive is declarative
- Pig Latin allows pipeline developers to decide where to check point data in the pipeline
- PigLatin allows the developer to select specific operator implementations directly rather than relying on the optimizer
- PigLatin supports splits in the pipeline
- Pig Latin allows developers to insert their own code almost anywhere in the data pipeline
- HIVE has a MetaStore, Pig doesn’t

Pig vs. Hive

- Use Hive when you have warehousing needs and you are good at SQL and don't want to write MapReduce jobs. But each and every problem cannot be solved using HiveQL

- Use Pig when you want to do a lot of transformations on your data and don't want to take the pain of writing MapReduce jobs

Map Reduce Cloud based solutions

"It was much nicer before people started storing all their personal information in the cloud."

Map Reduce Cloud based solutions

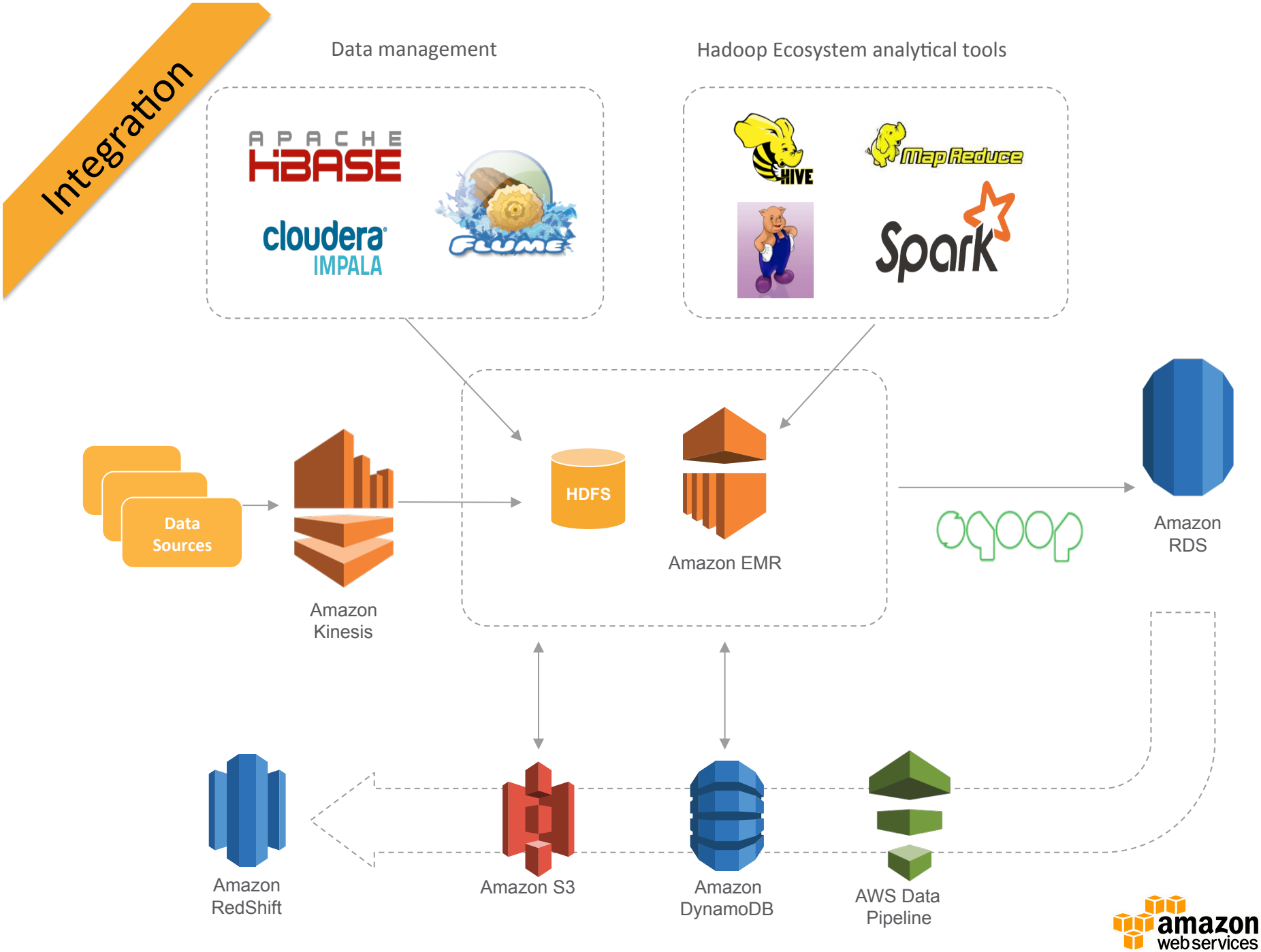
Your company starts a Big Data initiative

You are asked to…

1) Build a Hadoop cluster
   (IT) → Clusters are hard to setup and manage
2) Build a data pipeline
   (engineers, data scientist) → Need to integrate a zoo of tools
3) Get insight & build data products
   (engineers, data scientist, analysts) → Tools are hard to use

Elastic Map Reduce

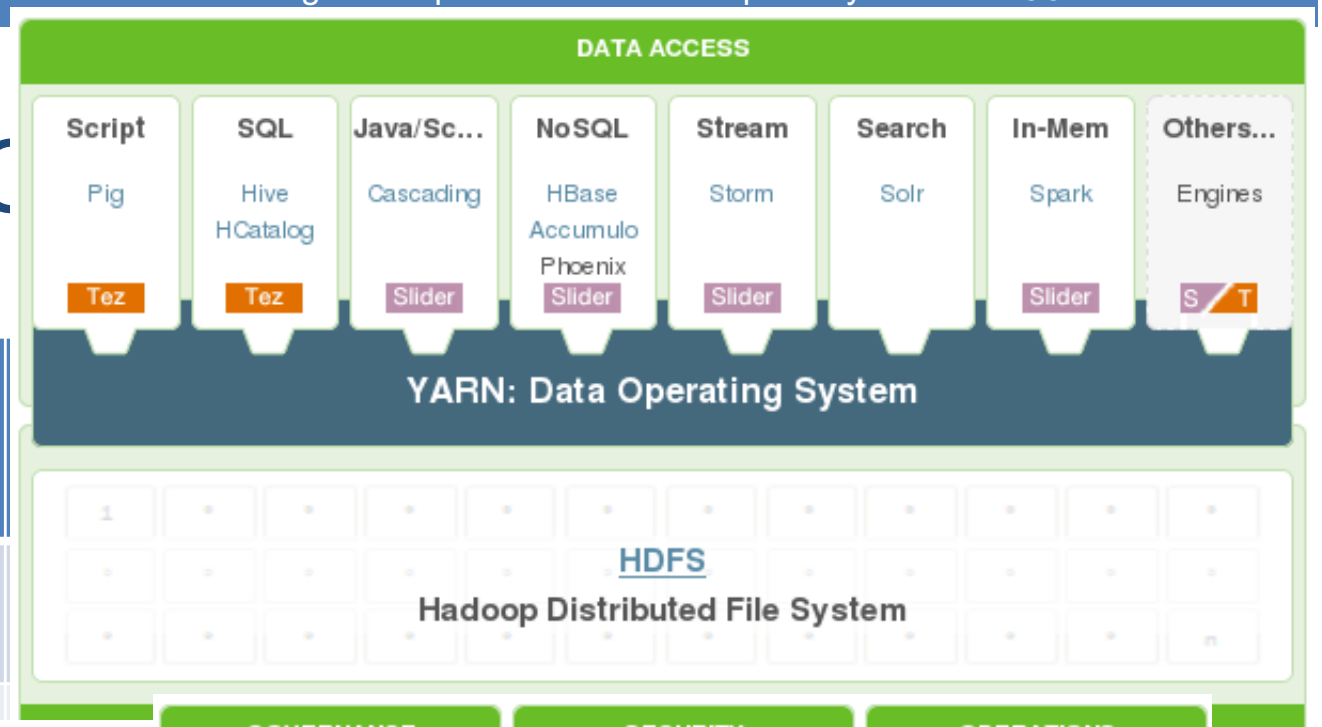
- No CAPEX
- Add resources on demand
- Key feature: integration

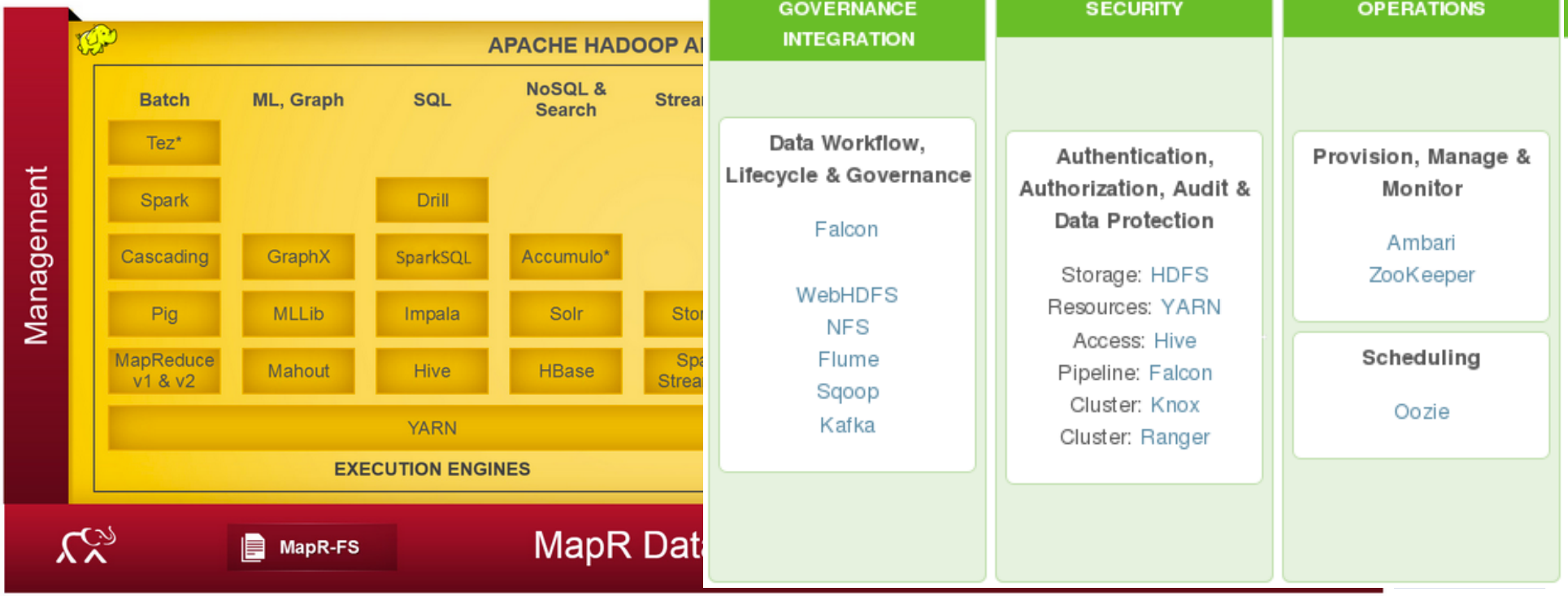

Integration
Data management
Hadoop Ecosystem analytical tools
APACHE HBASE
cloudera IMPALA
FLUME
HIVE
MapReduce
Spark
Data Sources
Amazon Kinesis
HDFS
Amazon EMR
Sqoop
Amazon RDS
Amazon RedShift
Amazon S3
Amazon DynamoDB
AWS Data Pipeline
amazon web services

Map Reduce C

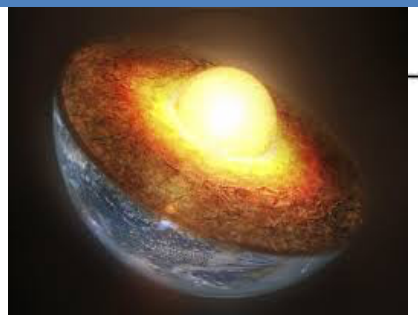
Automated install

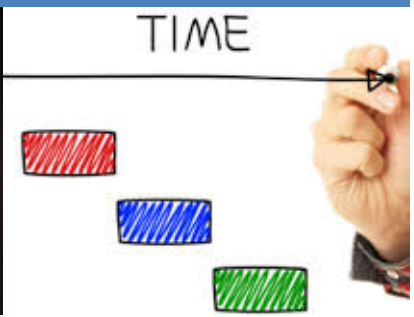
Storage input/output

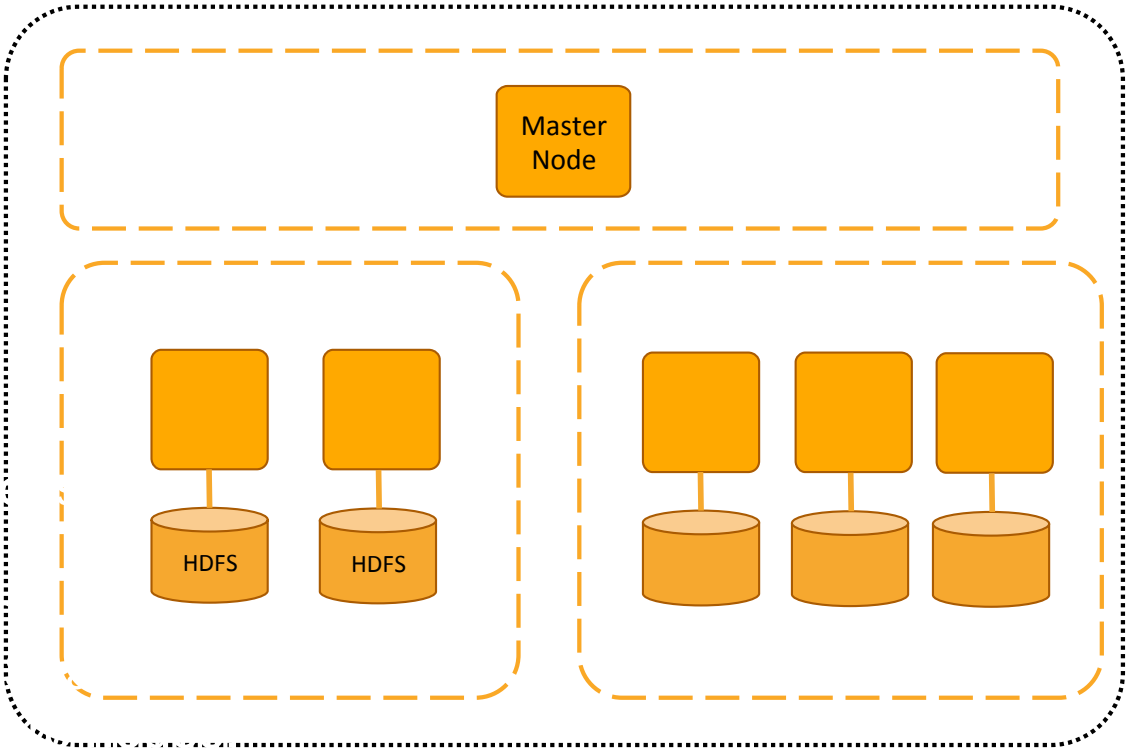




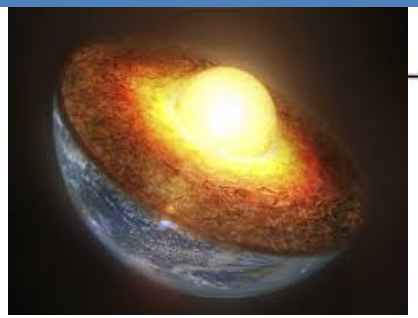
Core and Task nodes

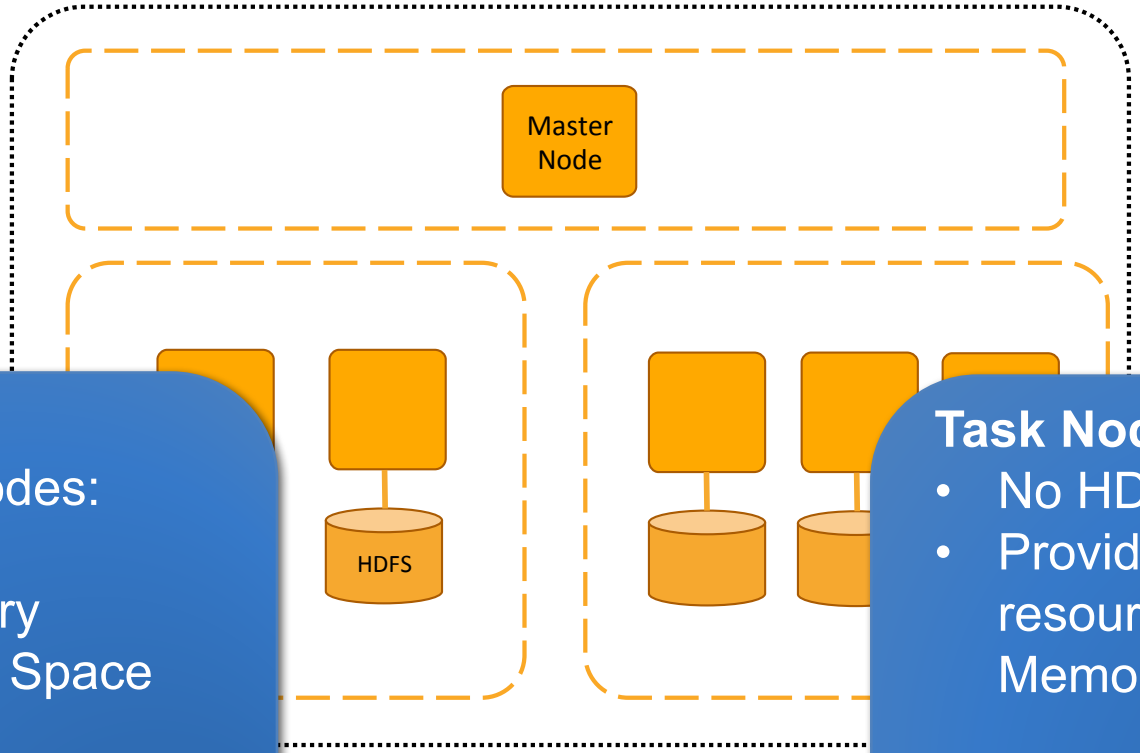






Core and Task nodes





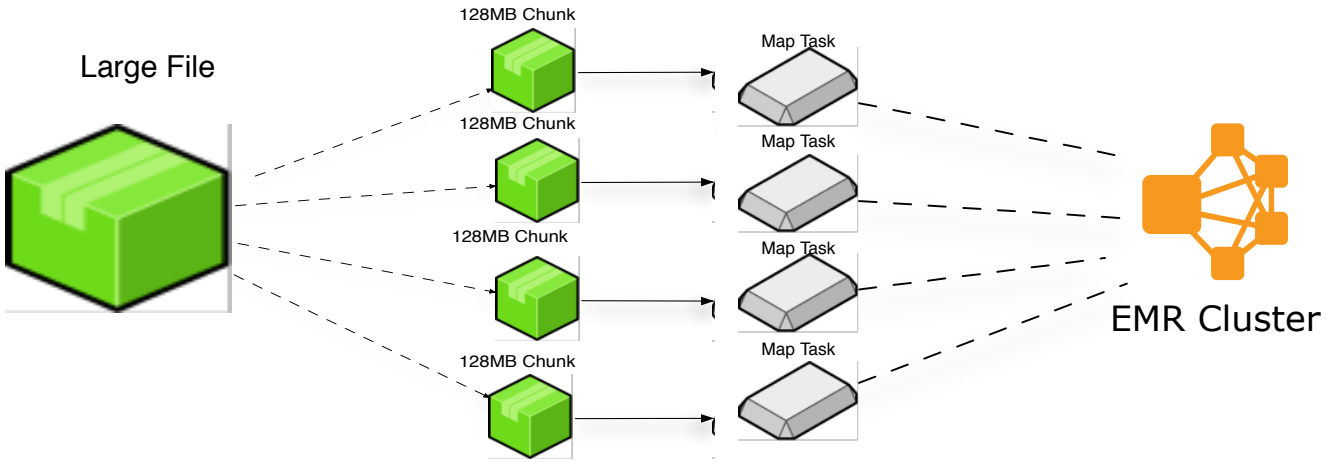
Core Nodes
Adding core nodes:
- More CPUs
- More Memory
- More HDFS Space

Can't remove core nodes:
- **HDFS corruptions**

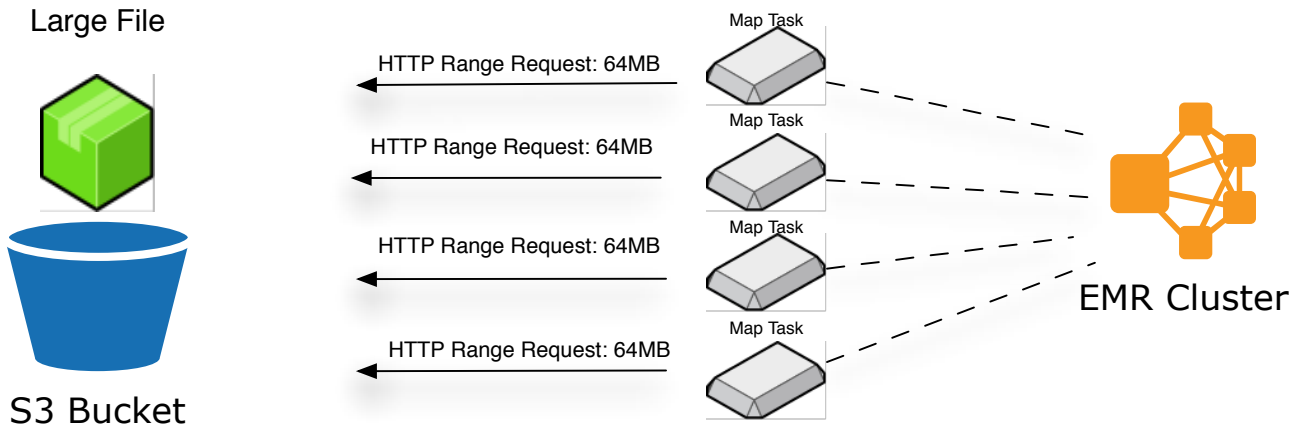
Task Nodes
- No HDFS
- Provides compute resources: CPU Memory

- Can add and **remove** task nodes

EMR and S3 best practice

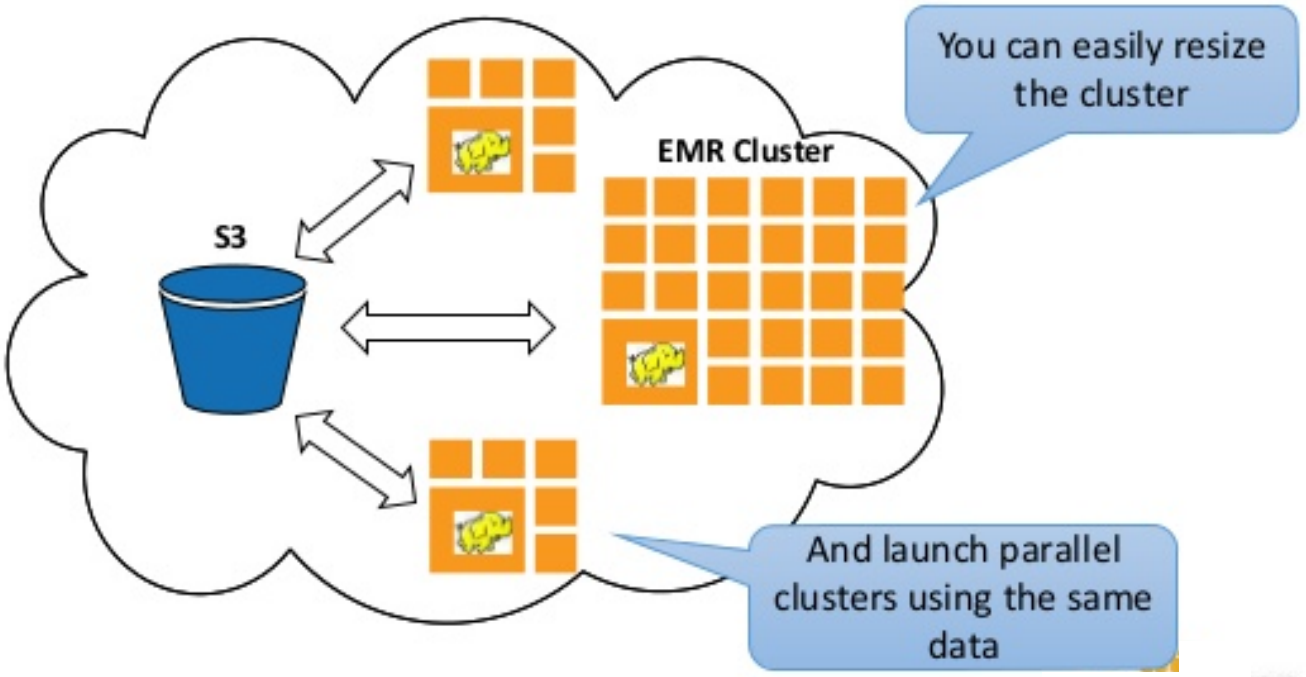


HDFS on
Ephemeral storage



S3 data deployment

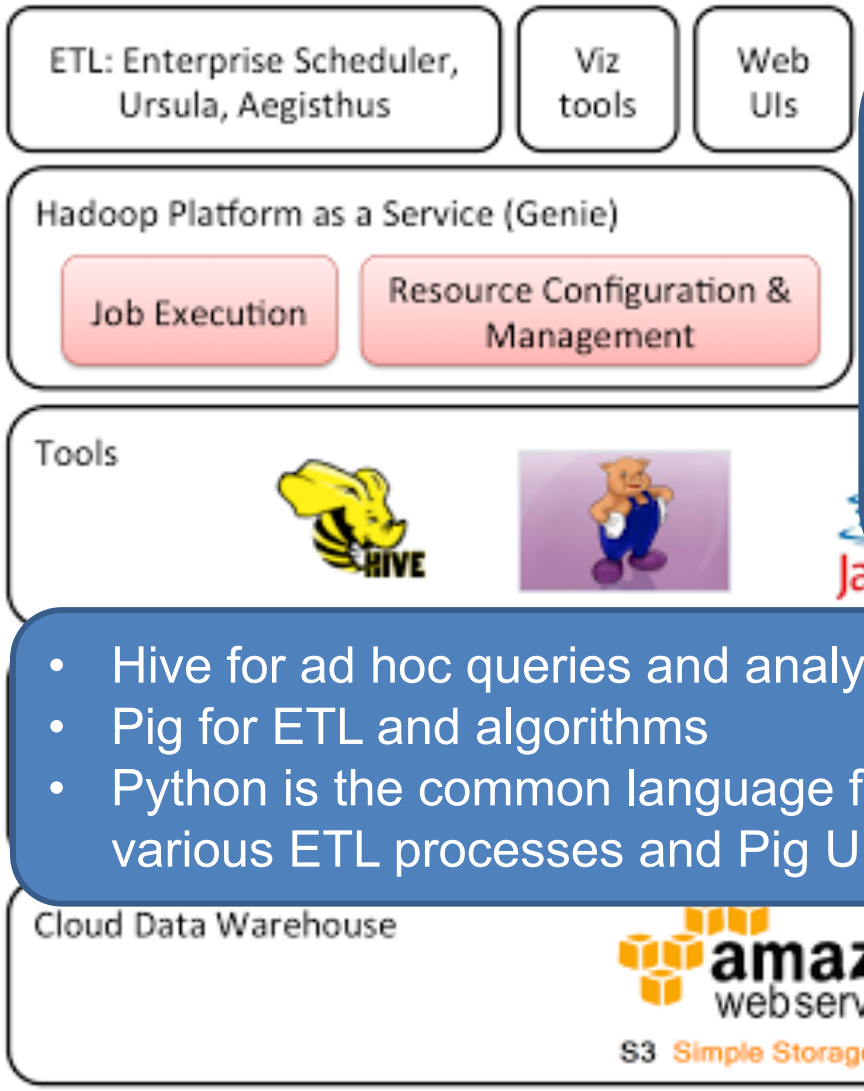
EMR and S3 best practice



	HDFS on Ephemeral Storage	Amazon S3
Read	350 mbps/node	120 mbps/node
Write	200 mbps/node	100 mbps/node

https://www.xplenty.com/blog/2014/03/storing-apache-hadoop-data-cloud-hdfs-vs-s3/

Netflix Case study - part 2



- Genie is a set of REST-full services for job and resource management
- Execution Service: provides a REST-full API to submit and manage Hadoop, Hive and Pig jobs
- Configuration Service: repository of available Hadoop resources, along with the metadata required to connect to and run jobs

- Hive for ad hoc queries and analytics
- Pig for ETL and algorithms
- Python is the common language for scripting various ETL processes and Pig UDF

Netflix Case study - S3

- Stores any dataset that is worth retaining:
  - Data from billions of streaming events from (Netflix-enabled) televisions, laptops, and mobile devices every hour captured by the log data pipeline (called Ursula)
  - Dimension data from Cassandra supplied by our Aegisthus pipeline
- Designed for 99.999999999% durability and 99.99% availability
- Provides bucket versioning, which we use to protect against inadvertent data loss
- Elastic, and provides practically “unlimited” size
- Enables to run multiple, highly dynamic clusters that are adaptable to failures and load
- On the flip side:
  - Reading and writing from S3 can be slower than writing to HDFS
  - Most queries and processes tend to be multi-stage MapReduce jobs
  - HDFS and local storage are used for all intermediate and transient data

Netflix Case study - EMR

- Multiple Hadoop clusters for different workloads, all accessing the exact same data
- A 500+ node "query" cluster is used by engineers, data scientists and analysts to perform ad hoc queries
- Production (or “SLA”) cluster, around the same size, runs SLA-driven ETL (extract, transform, load) jobs
- Several other “dev” clusters that are spun up as needed
- Dynamically resize both our query and production clusters daily
- Query cluster smaller at night when there are fewer developers logged in, whereas the production cluster must be larger at night, when most of our ETL is run
- Production and query clusters are long-running clusters but treated as completely transient