CPT-S 415

Topics in Computer Science

Big Data

Yinghui Wu

EME B45

CPT-S 415
Big Data

MapReduce/Hadoop

- ✓ Hadoop: history, features and design
- ✓ Hadoop ecosystem

Hadoop

Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.

- A flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.

✓ open-source implementation for Google ***MapReduce***

✓ based on *MapReduce*

✓ based on a simple data model for *any data*

Doug Cutting

Large-Scale Data Analytics

- MapReduce computing paradigm (e.g., Hadoop) vs. Traditional database systems

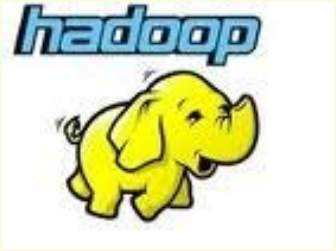
vs.

Adopted by many enterprises

- esp. applications generating ***big data***
- Web applications, social networks, scientific applications

- Google: Inventors of MapReduce computing paradigm
- Yahoo: Developing Hadoop open-source of MapReduce
- IBM, Microsoft, Oracle
- Facebook, Amazon, AOL, NetFlex
- Many others + universities and research labs

Hadoop vs. Traditional DBMS



vs.

Scalability (petabytes of data, thousands of machines)

Flexibility in accepting all data formats (no schema)

Simple fault-tolerant mechanism

Commodity inexpensive hardware

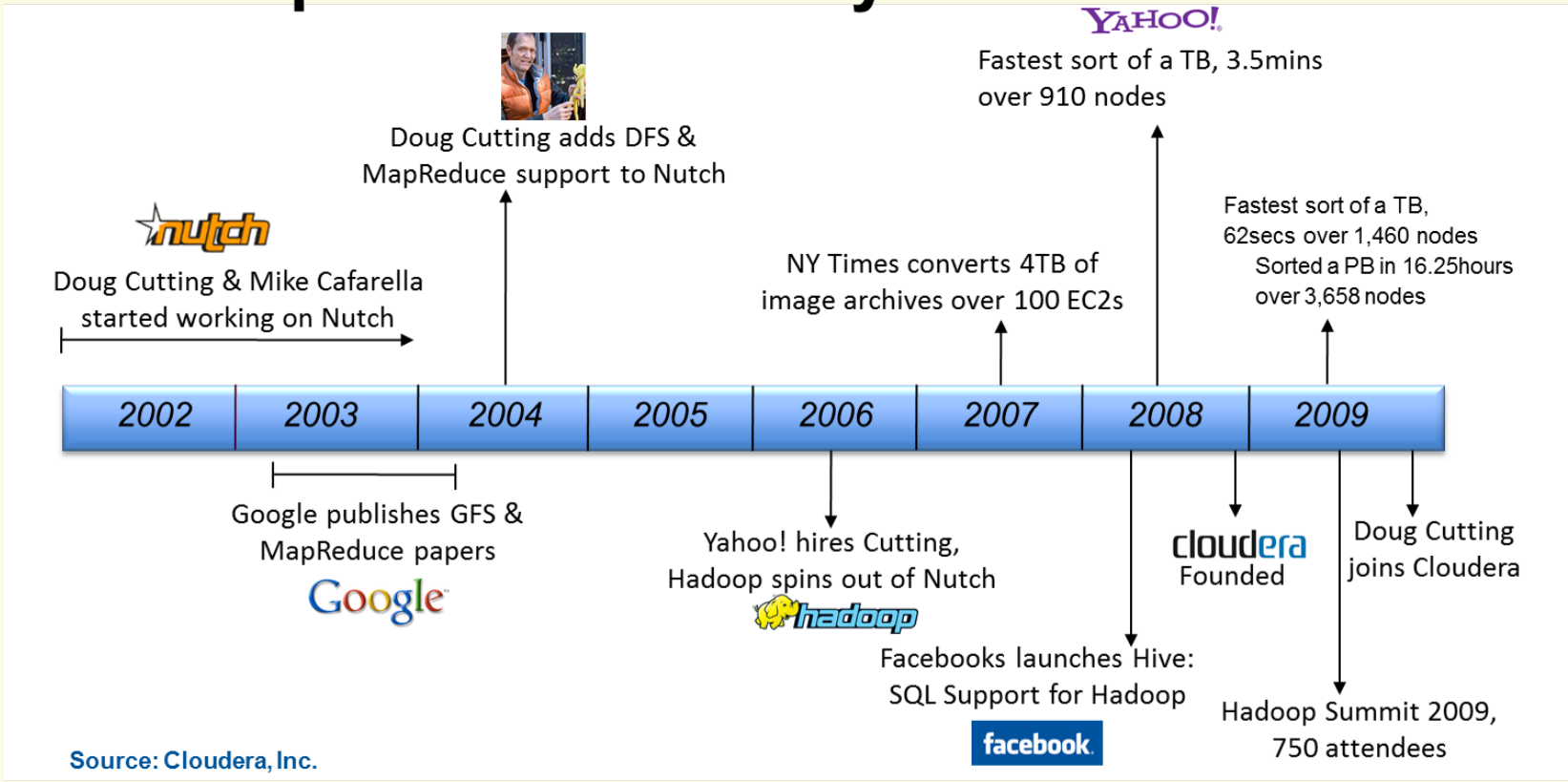
Performance (indexing, tuning, data organization tech.)

Features:

- Provenance tracking
- Annotation management
- ….

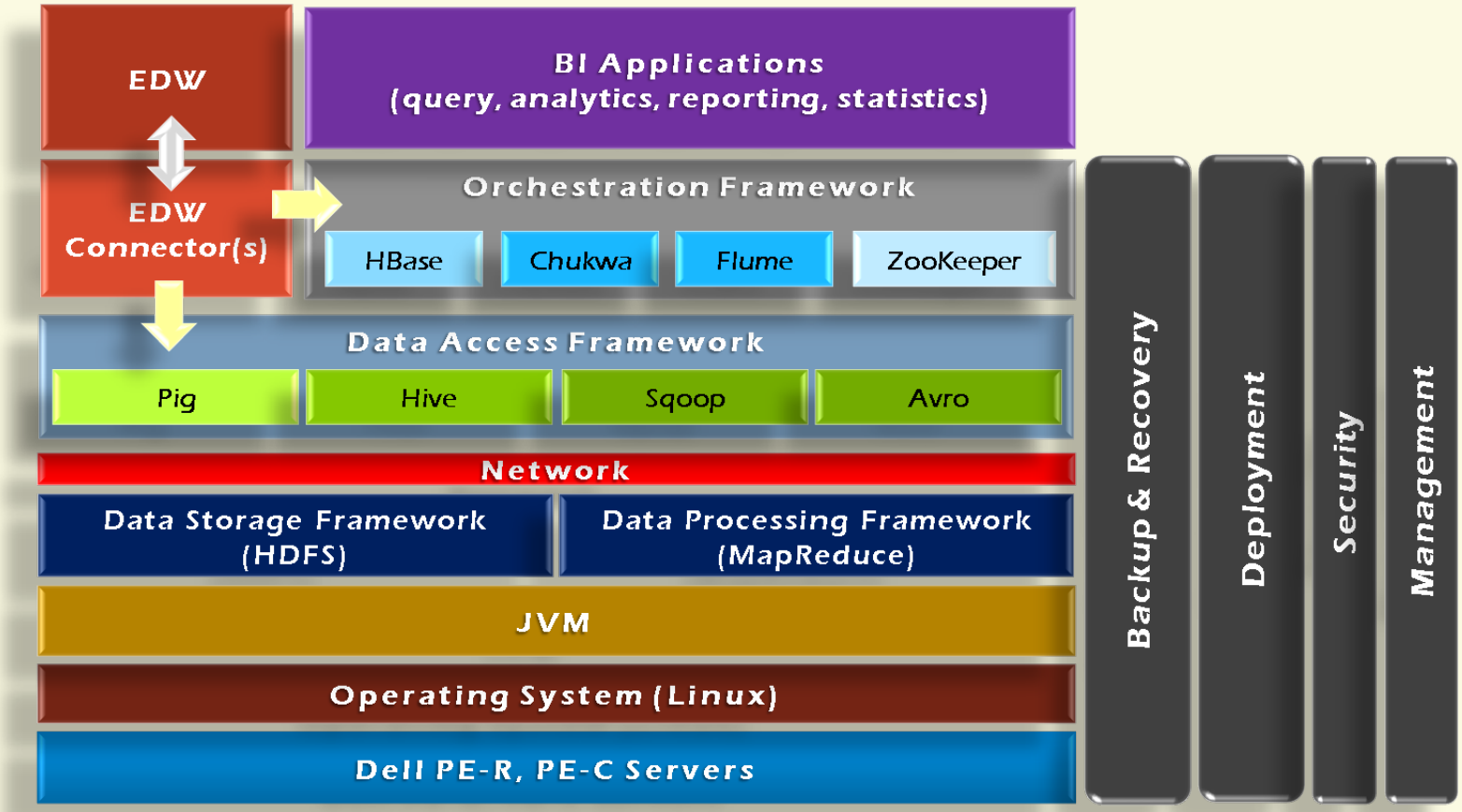
Hadoop History

Hadoop Creation History



Source: Cloudera, Inc.

Hadoop Framework Tools



Design Principles of Hadoop

- ✓ Need to parallelize computation across thousands of nodes
- ✓ **Commodity hardware**
  - Large number of low-end cheap machines working in parallel to solve a computing problem
  - in contrast to **Parallel DBs:** Small number of high-end expensive machines
- ✓ **Automatic parallelization & distribution**
  - Hidden from the end-user
- ✓ **Fault tolerance and automatic recovery**
  - Nodes/tasks will fail and will recover automatically
- ✓ **Clean and simple programming abstraction**
  - Users only provide two functions “map” and “reduce”

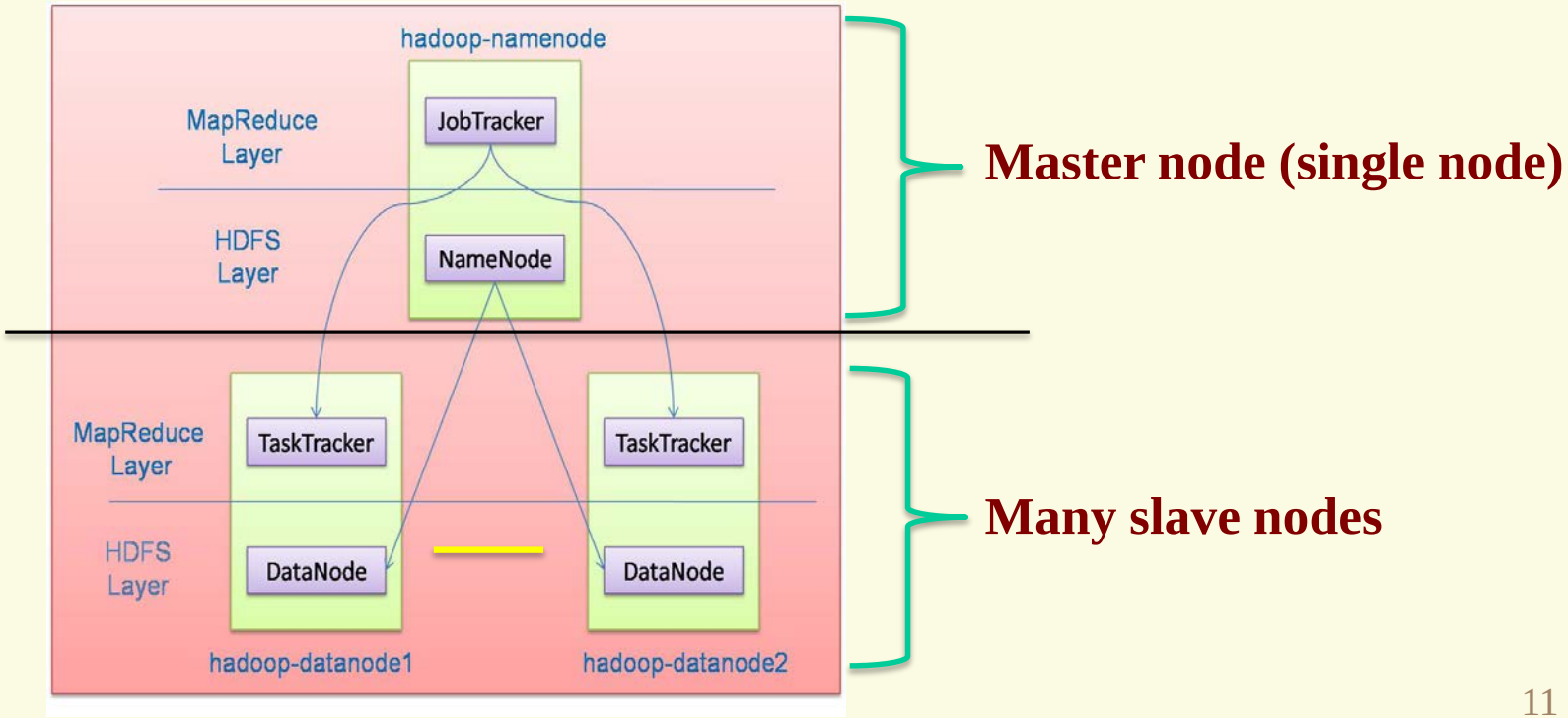
HDFS

HDFS: Main Properties

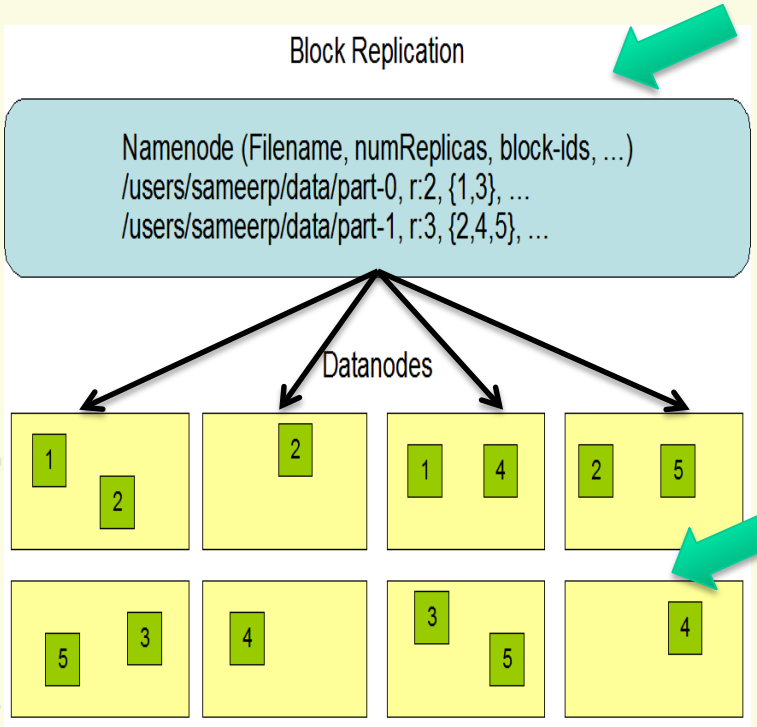
- ***Large:*** A HDFS instance may consist of thousands of server machines, each storing part of the file system's data
- ***Replication:*** Each data block is replicated many times (default is 3)
- ***Fault Tolerance:*** Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS
  - Namenode is consistently checking Datanodes: The Namenode receives a Heartbeat and a BlockReport from each DataNode in the cluster.

Hadoop Master/Slave Architecture

- Hadoop is designed as a *master-slave* *shared-nothing* architecture
  - Distributed file system (HDFS)
  - Execution engine (MapReduce)



Hadoop Distributed File System (HDFS)



Centralized namenode

- Maintains metadata info about files

  FsImage + EditLog

File *F*

Blocks (64 MB)

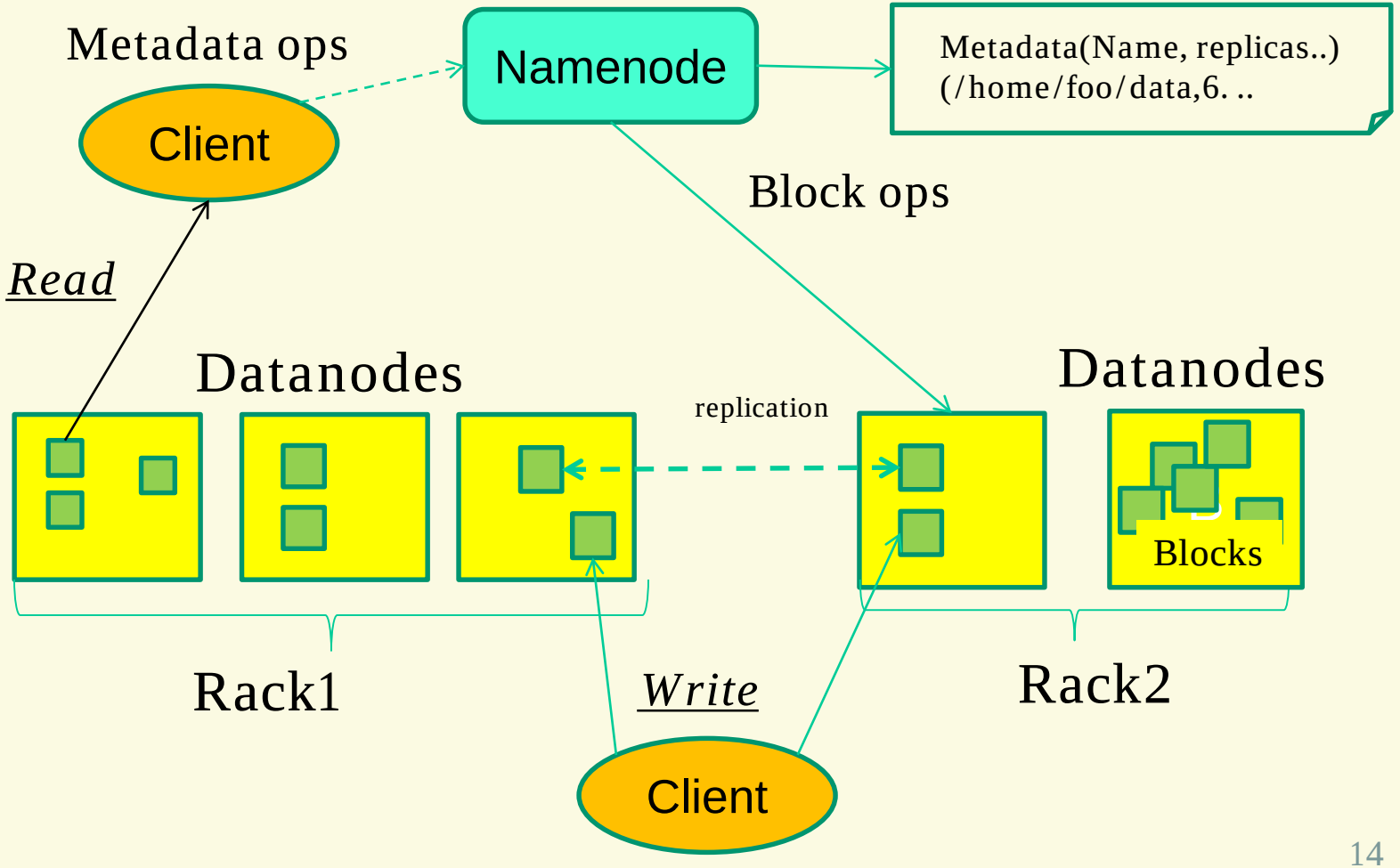
Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

Functions of “nodes”

- NameNode
  - Manages File System Namespace
    - Maps a file name to a set of blocks
    - Maps a block to the DataNodes where it resides
    - FsImage + EditLog
  - Cluster Configuration Management
  - Replication Engine for Blocks
- DataNode
  - A Block Server: Stores data in the local file system (e.g. ext3); Stores metadata of a block; Serves data and metadata to Clients
  - Block Report
    - Periodically sends a report of all existing blocks to the NameNode
  - Facilitates Pipelining of Data
    - Forwards data to other specified DataNodes

HDFS Architecture



HDFS command

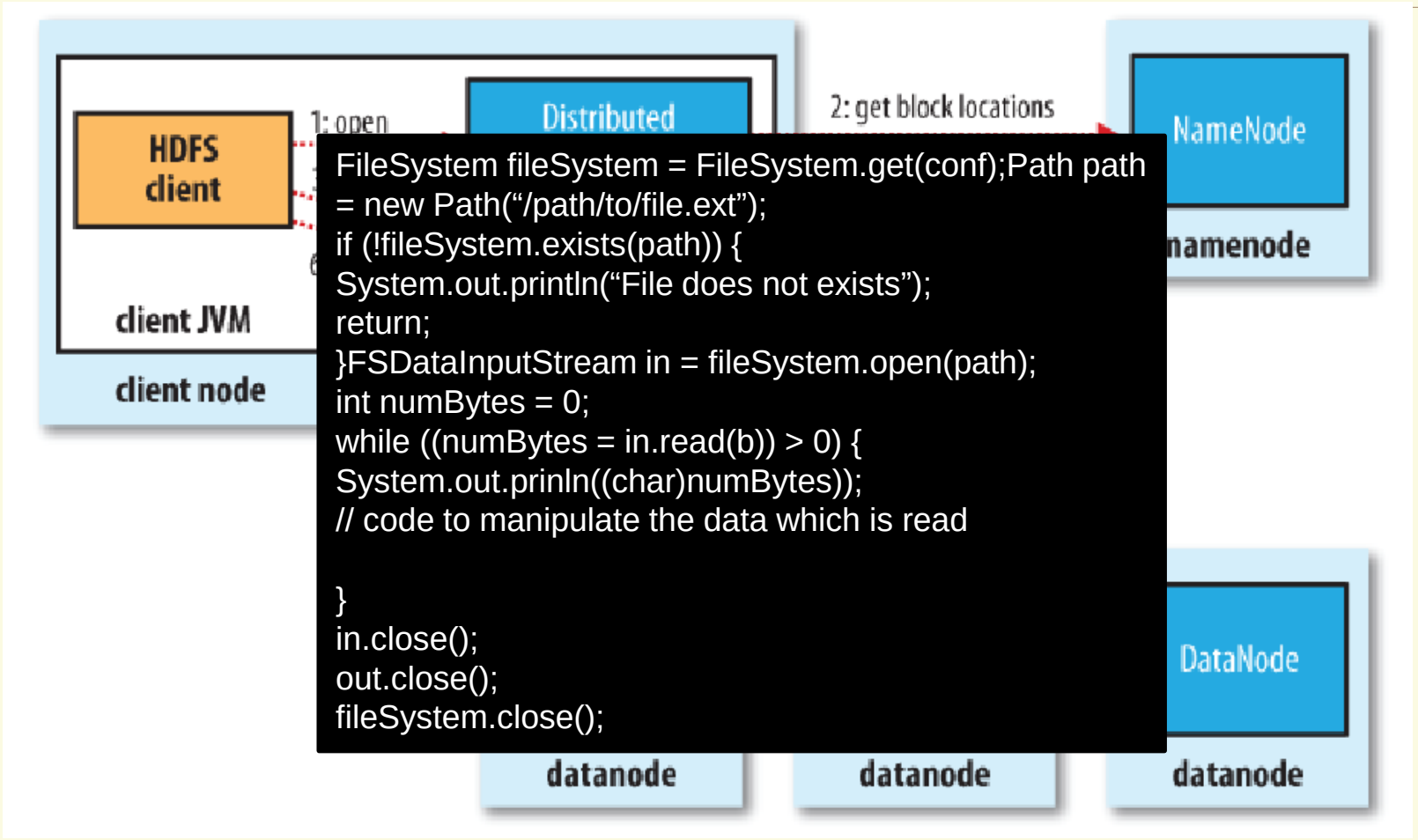
- Shell command: *most common: fs*
  hadoop fs [genericOptions] [commandOptions]
- hadoop fs -ls <path>: display detailed file info specified by path
- hadoop fs -mkdir <path>: create folder

```
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -mkdir hdfs://127.0.0.1:9000/tempDir

administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -ls hdfs://127.0.0.1:9000/
Found 4 items
drwxr-xr-x   - administrator supergroup          0 2015-04-26 16:30 /hbase
drwxr-xr-x   - administrator supergroup          0 2015-04-26 15:44 /home
drwxr-xr-x   - administrator supergroup          0 2015-04-26 16:46 /tempDir
drwxr-xr-x   - administrator supergroup          0 2015-04-26 15:55 /user
```

- hadoop fs -cat <path>: stdout file content
- hadoop fs –copyFromLocal <localsrc> <dst>: copy file

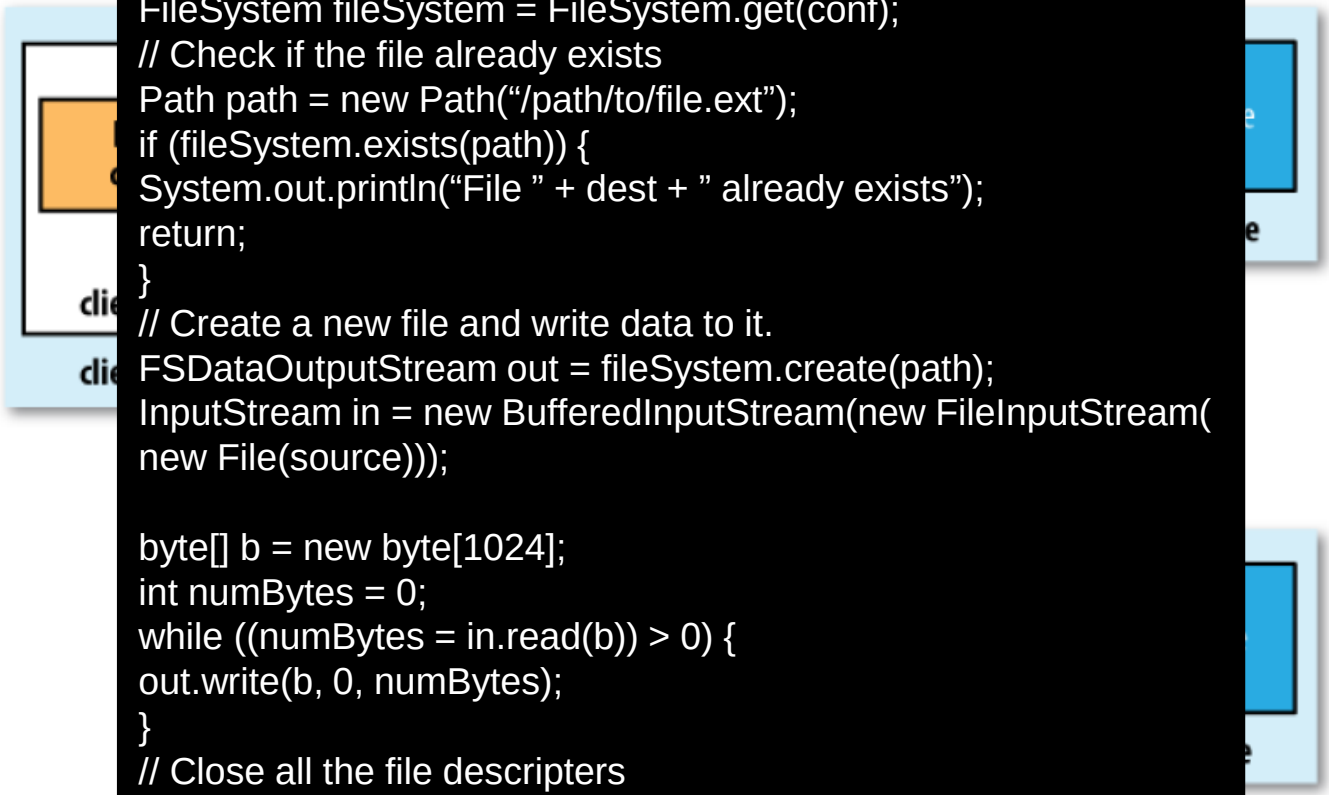
```
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -copyFromLocal /home/administrator/t
empfile/* hdfs://127.0.0.1:9000/tempDir
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -ls  hdfs://127.0.0.1:9000/tempDir/
Found 8 items
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file1.txt
-rw-r--r--   1 administrator supergroup         14 2015-04-26 16:48 /tempDir/file1.txt~
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file2.txt
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file3.txt
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file4.abc
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file5.abc
-rw-r--r--   1 administrator supergroup         17 2015-04-26 16:48 /tempDir/testFile
-rw-r--r--   1 administrator supergroup          0 2015-04-26 16:48 /tempDir/testFile~
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -cat  hdfs://127.0.0.1:9000/tempDir/
*
this is file1.txt
this is file1
this is file2.txt
this is file3.txt
this is file4.abc
this is file5.abc
welcome to DBLab
```

HDFS read




```
FileSystem fileSystem = FileSystem.get(conf);Path path
= new Path("/path/to/file.ext");
if (!fileSystem.exists(path)) {
System.out.println("File does not exists");
return;
}FSDataInputStream in = fileSystem.open(path);
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
System.out.prinln((char)numBytes));
// code to manipulate the data which is read

}
in.close();
out.close();
fileSystem.close();
```

HDFS write



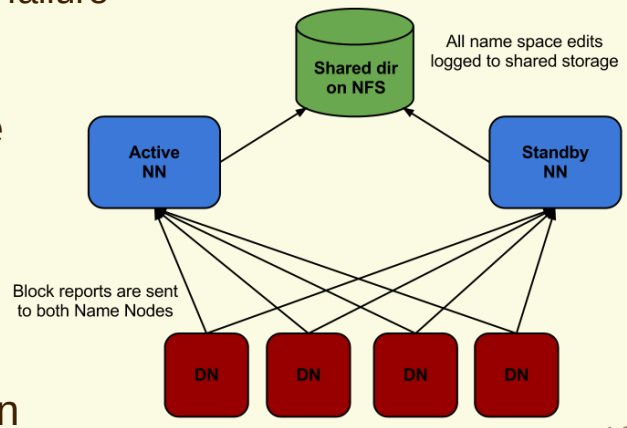
```
FileSystem fileSystem = FileSystem.get(conf);
// Check if the file already exists
Path path = new Path("/path/to/file.ext");
if (fileSystem.exists(path)) {
System.out.println("File " + dest + " already exists");
return;
}
// Create a new file and write data to it.
FSDataOutputStream out = fileSystem.create(path);
InputStream in = new BufferedInputStream(new FileInputStream(
new File(source)));

byte[] b = new byte[1024];
int numBytes = 0;
while ((numBytes = in.read(b)) > 0) {
out.write(b, 0, numBytes);
}
// Close all the file descripters
in.close();
out.close();
fileSystem.close();
```

System issues

- ✓ Block Placement: How to place data blocks?
  - One replica on local node, second/third on same remote rack, additional replica randomly placed
  - Clients read from nearest replicas
- ✓ Replication Engine
  - NameNode detects DataNode failures
    - Chooses new DataNodes for new replicas
    - Balances disk usage
    - Balances communication traffic to DataNodes
- ✓ Rebalancer: % of disk full on DataNodes should be similar
  - Run when new datanodes are added

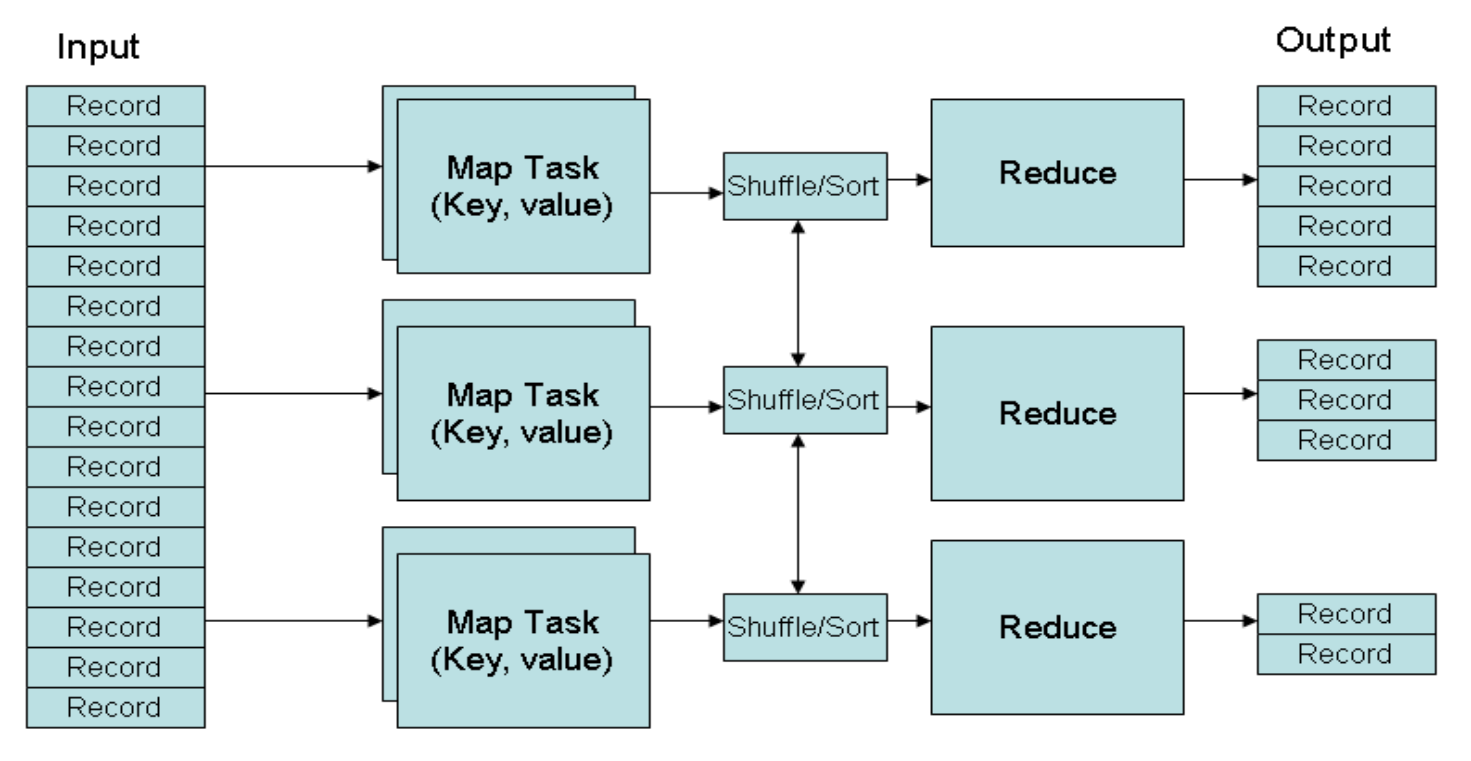
Data recovery and error tolerance

- ✓ HDFS treats fault as norm not exception
  - Namenode failure
  - Datanode failure
  - Data error
- ✓ Heartbeats
  - DataNodes send hearbeat to the NameNode
    - Once every 3 seconds
  - NameNode uses heartbeats to detect DataNode failure
- ✓ Namenode failure:
  - FsImage, Editlog -> SecondaryNameNode
  - Transaction Log + standby NN
- ✓ Data error
  - md5/sha1 validation
  - client check/report -> namenode replication



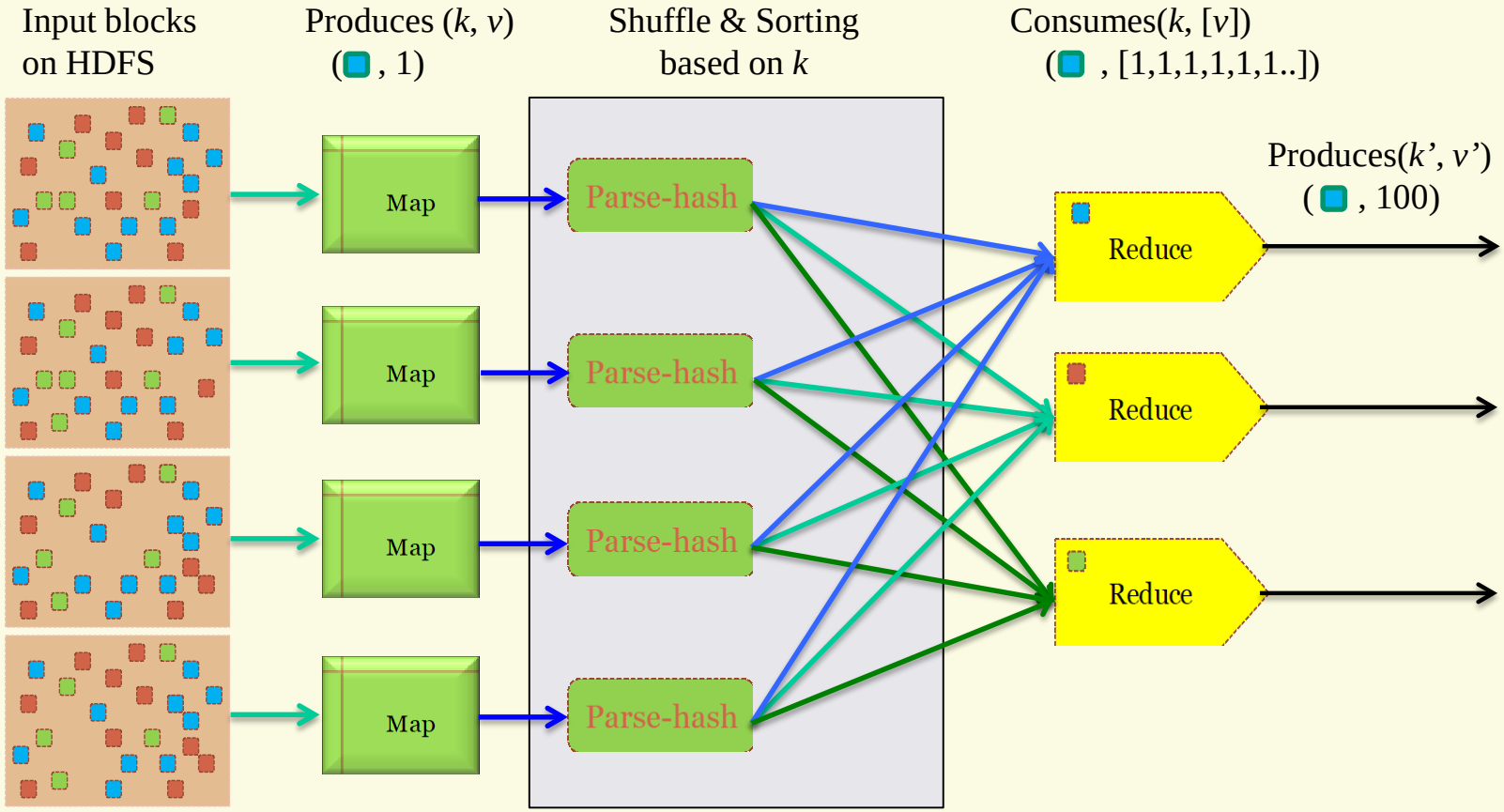
MapReduce layer

MapReduce Phases



Deciding on what will be the key and what will be the value ➔ developer's responsibility

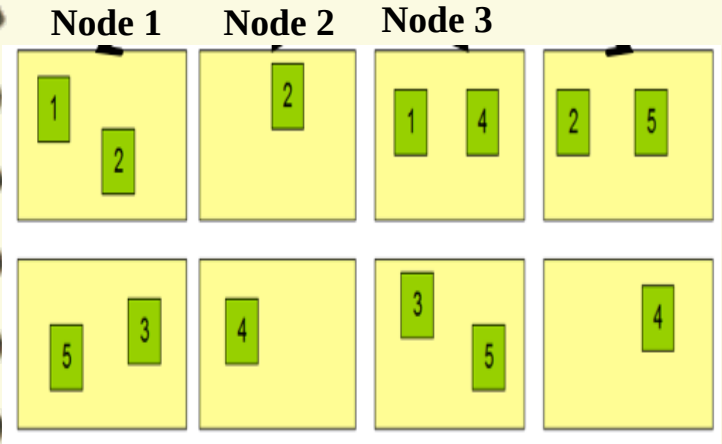
Map-Reduce Execution Engine (Example: Color Count)



Users only provide the "Map" and "Reduce" functions

Properties of MapReduce Engine

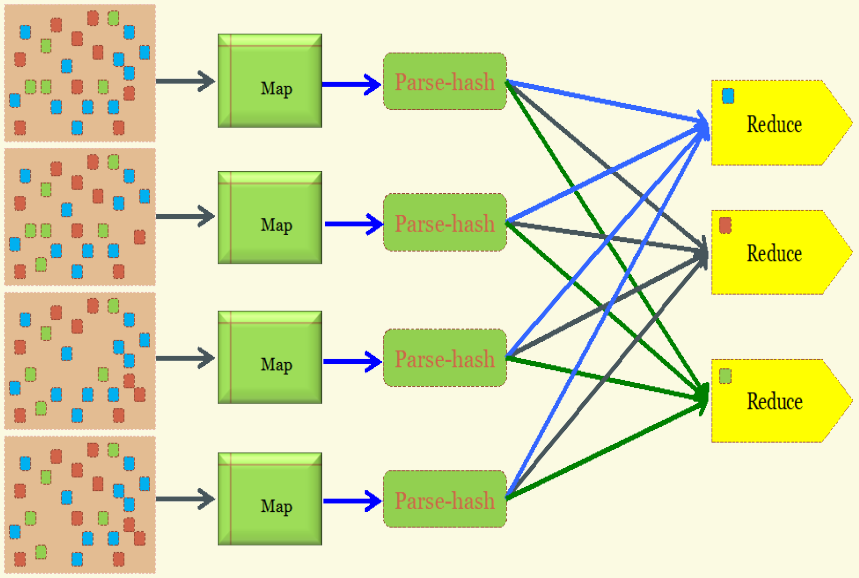
- ✓ **Job Tracker is the master node (runs with the namenode)**
  - Receives the user's job
  - Decides on how many tasks will run (number of mappers)
  - Decides on where to run each mapper (concept of locality)



- This file has 5 Blocks → run 5 map tasks
- Where to run the task reading block "1"
  - *Try to run it on Node 1 or Node 3*

Properties of MapReduce Engine (Cont'd)

- **Task Tracker is the slave node (runs on each datanode)**
  - Receives the task from Job Tracker
  - Runs the task until completion (either map or reduce task)
  - Always in communication with the Job Tracker reporting progress

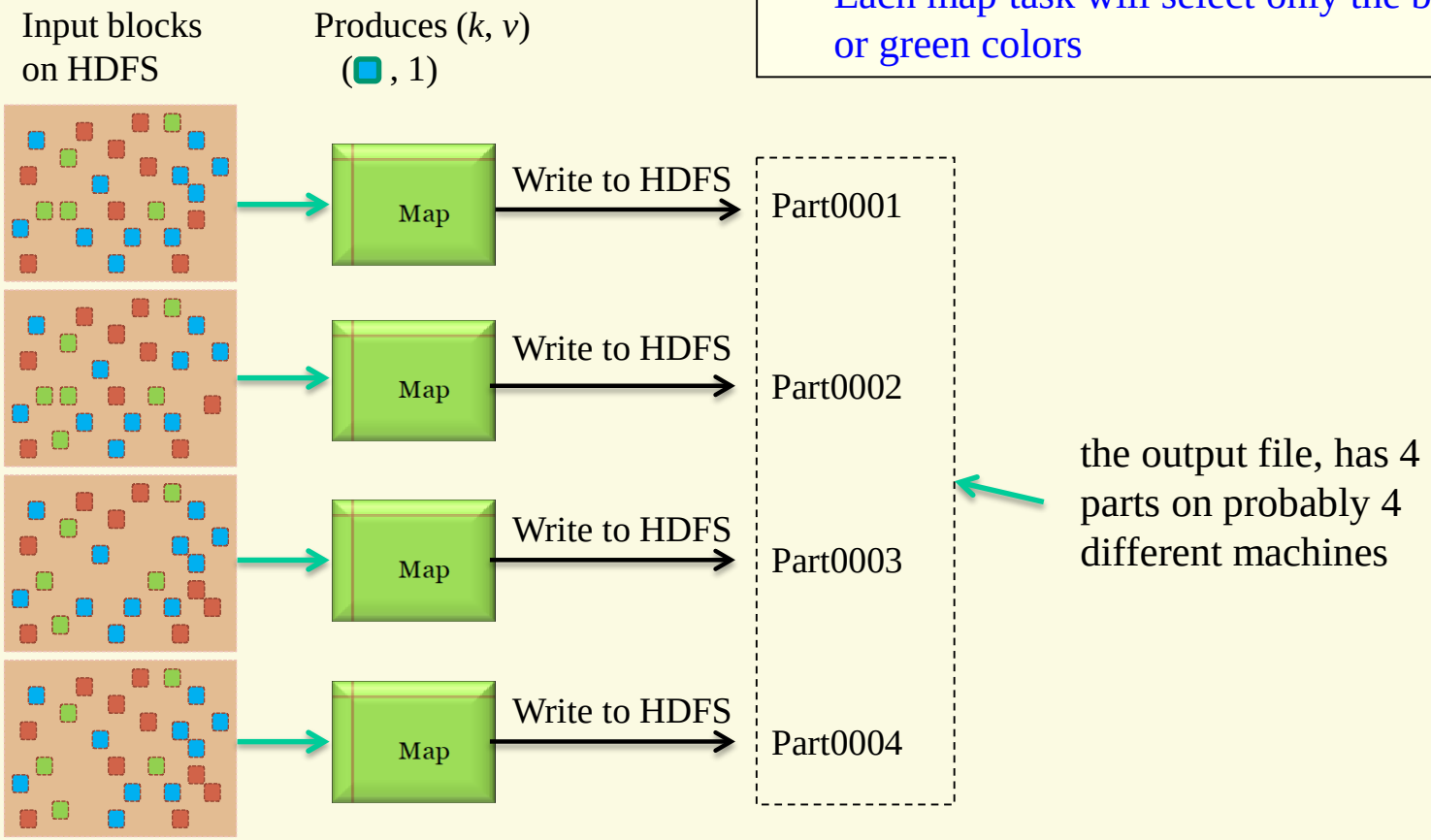


In this example, 1 map-reduce job consists of 4 map tasks and 3 reduce tasks

Example: Color Filter

Job: Select only the blue and the green colors

- Each map task will select only the blue or green colors



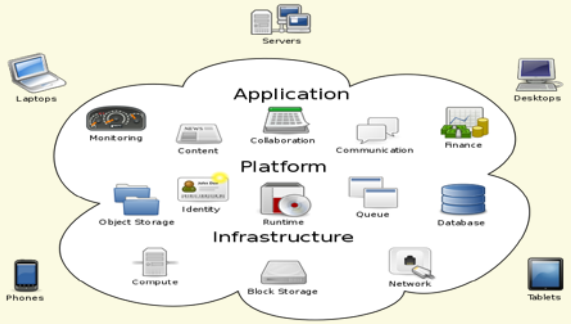
Putting it all together

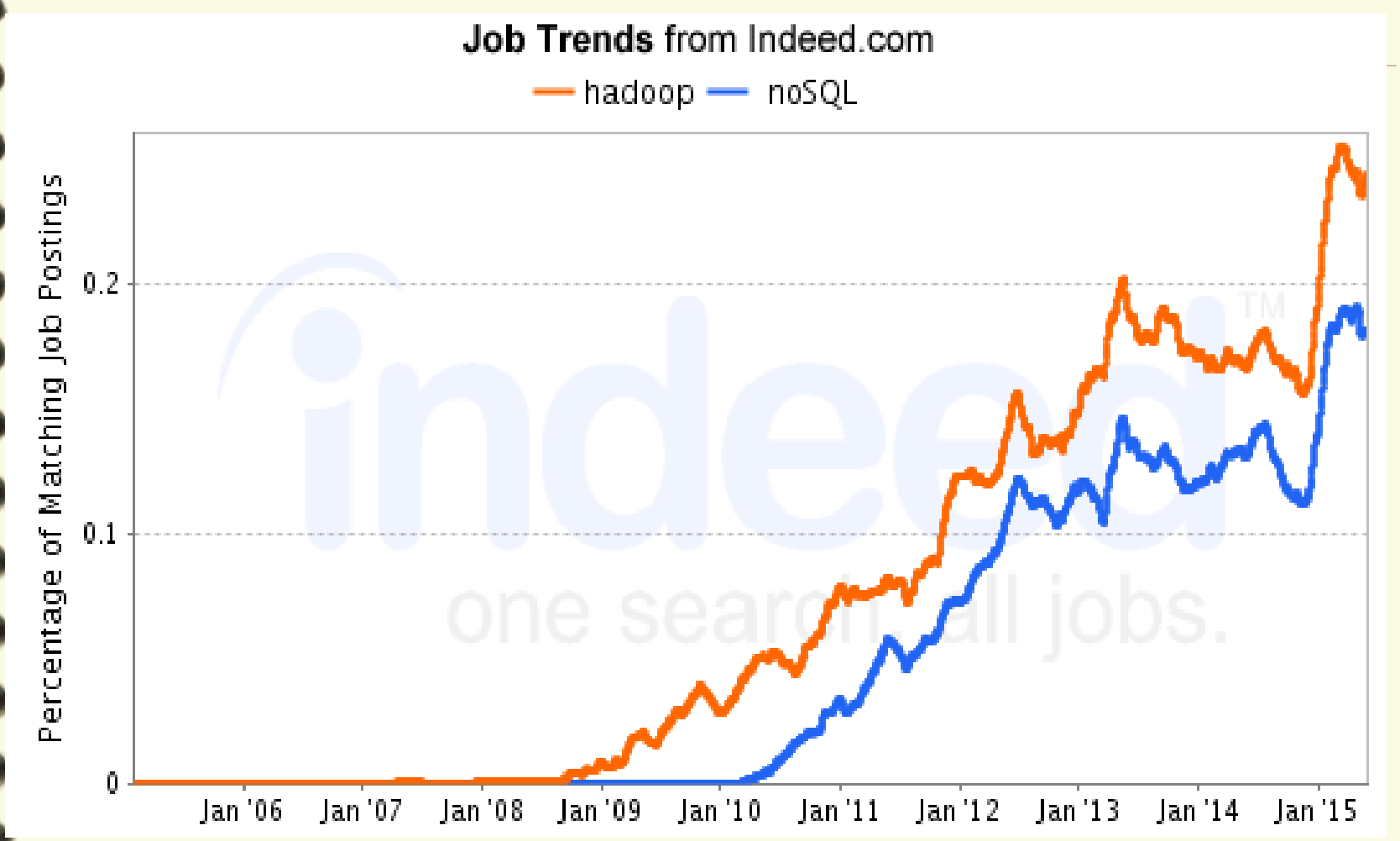
- Create a launching program for your application
- The launching program configures:
  - The *Mapper* and *Reducer* to use
  - The output key and value types (input types are inferred from the *InputFormat*)
  - The locations for your input and output
- The launching program then submits the job and typically waits for it to complete

Bigger Picture: Hadoop vs. Other Systems

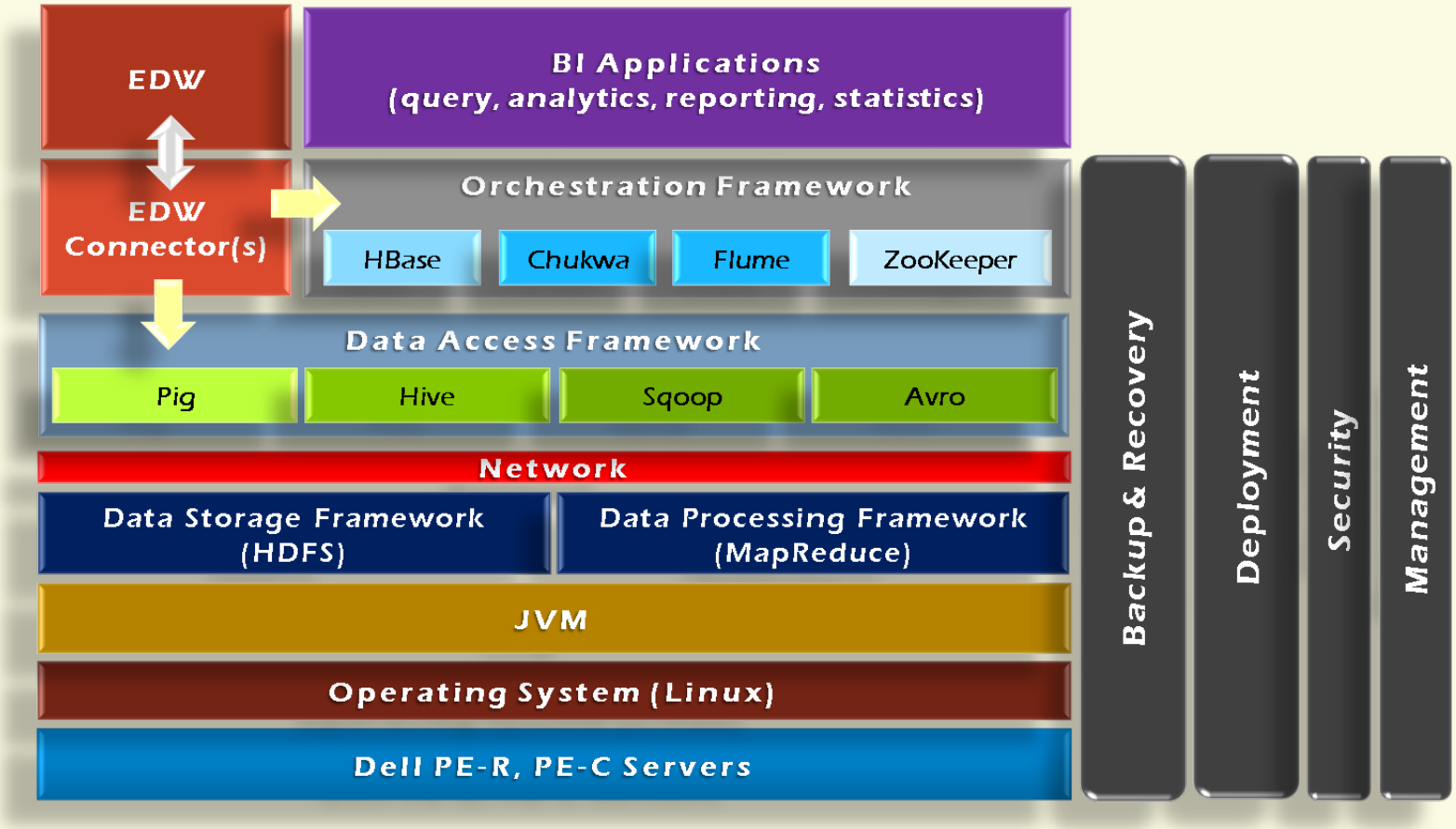
	Distributed Databases	Hadoop
Computing Model	- Notion of transactions - Transaction is the unit of work - ACID properties, Concurrency control	- Notion of jobs - Job is the unit of work - No concurrency control
Data Model	- Structured data with known schema - Read/Write mode	- Any data will fit in any format - (un)(semi)structured - ReadOnly mode
Cost Model	- Expensive servers	- Cheap commodity machines
Fault Tolerance	- Failures are rare - Recovery mechanisms	- Failures are common over thousands of machines - Simple yet efficient fault tolerance
Key Characteristics	- Efficiency, optimizations, fine-tuning	- Scalability, flexibility, fault tolerance

- ***Cloud Computing***
  - A computing model where any computing infrastructure can run on the cloud
  - Hardware & Software are provided as remote services
  - Elastic: grows and shrinks based on the user's demand
  - Example: Amazon EC2



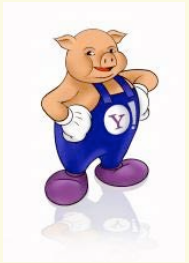

Job Trends from Indeed.com
hadoop
noSQL
Percentage of Matching Job Postings
0.2
0.1
0
Jan '06
Jan '07
Jan '08
Jan '09
Jan '10
Jan '11
Jan '12
Jan '13
Jan '14
Jan '15
indeed
one search. all jobs.

Hadoop Family



Hadoop Related Subprojects

- ✓ Pig
  - High-level language for data analysis
- ✓ HBase
  - Table storage for semi-structured data
- ✓ Hive
  - SQL-like Query language and Metastore
- ✓ Mahout
  - Machine learning
- ✓ Zookeeper
  - Coordinating distributed applications



Hive & Pig

Hive and Pig

- 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

- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs

- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language “compiles down” to Hadoop jobs

Need for High-Level Languages

- Hadoop scales well 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
  - Hive: HQL is like SQL
  - Pig: “a bit like Perl”

Hive

- ✓ Developed at Facebook
- ✓ Used for majority of Facebook jobs
- ✓ “Relational database” built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HiveQL)
  - Can call Hadoop Streaming scripts from HiveQL
  - Supports table partitioning, clustering, complex data types, some optimizations
- ✓ Utilized by individuals with strong SQL Skills and limited programming ability.

Hive Data Model

- ✓ Tables
  - Typed columns (int, float, string, boolean)
  - Also, list: map (for JSON-like data)
- ✓ Partitions
  - For example, range-partition tables by date
- ✓ Buckets
  - Hash partitions within ranges (useful for sampling, join optimization)

Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
- Actual data stored in flat files
  - Control char-delimited text, or SequenceFiles
  - With custom SerDe, can use arbitrary format

Creating a Hive Table

```
CREATE TABLE page_views(viewTime INT, userid BIGINT,
                    page_url STRING, referrer_url STRING,
                    ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

- ✓ Partitioning breaks table into separate files for each (dt, country) pair

  Ex: /hive/page_view/dt=2008-06-08,country=USA

  /hive/page_view/dt=2008-06-08,country=CA

A Simple Query

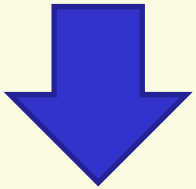
- Find all page views coming from xyz.com on March 31st:

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
AND page_views.referrer_url like '%xyz.com';
```

  - Hive only reads partition `2008-03-01,*` instead of scanning entire table

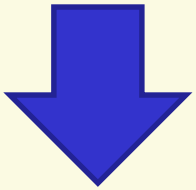
Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```



(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s) word)
(. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k)
freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```



(one or more of MapReduce jobs)

Pig

- ✓ Started at Yahoo! Research
- ✓ Now runs about 30% of Yahoo!'s jobs
- ✓ Features
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions
  - https://pig.apache.org/

Example Data Analysis Task

Find users who tend to visit “good” pages.

Visits

user	url	time
Amy	www.cnn.com	8:00
Amy	www.crap.com	8:05
Amy	www.myblog.com	10:00
Amy	www.flickr.com	10:05
Fred	cnn.com/index.htm	12:00

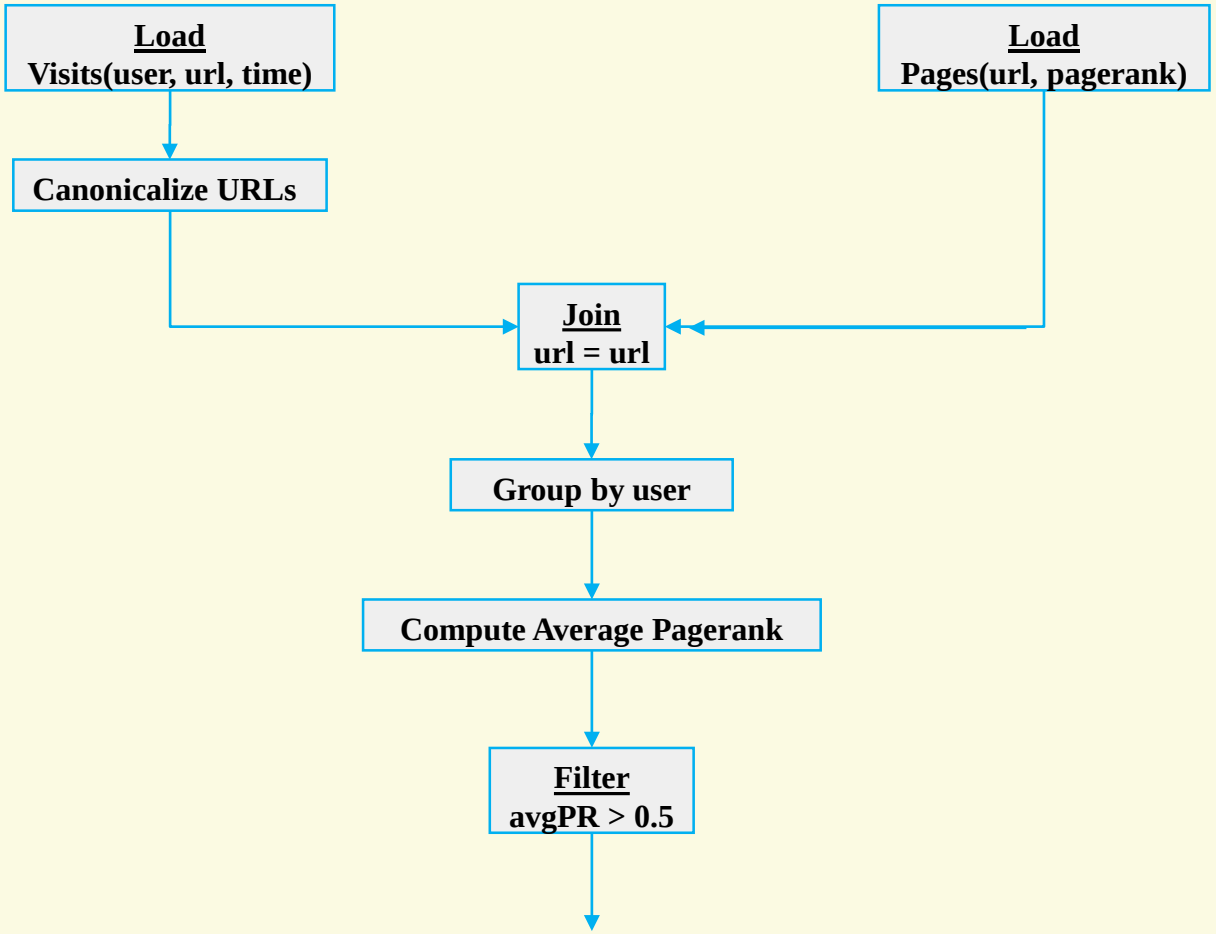
⋮

Pages

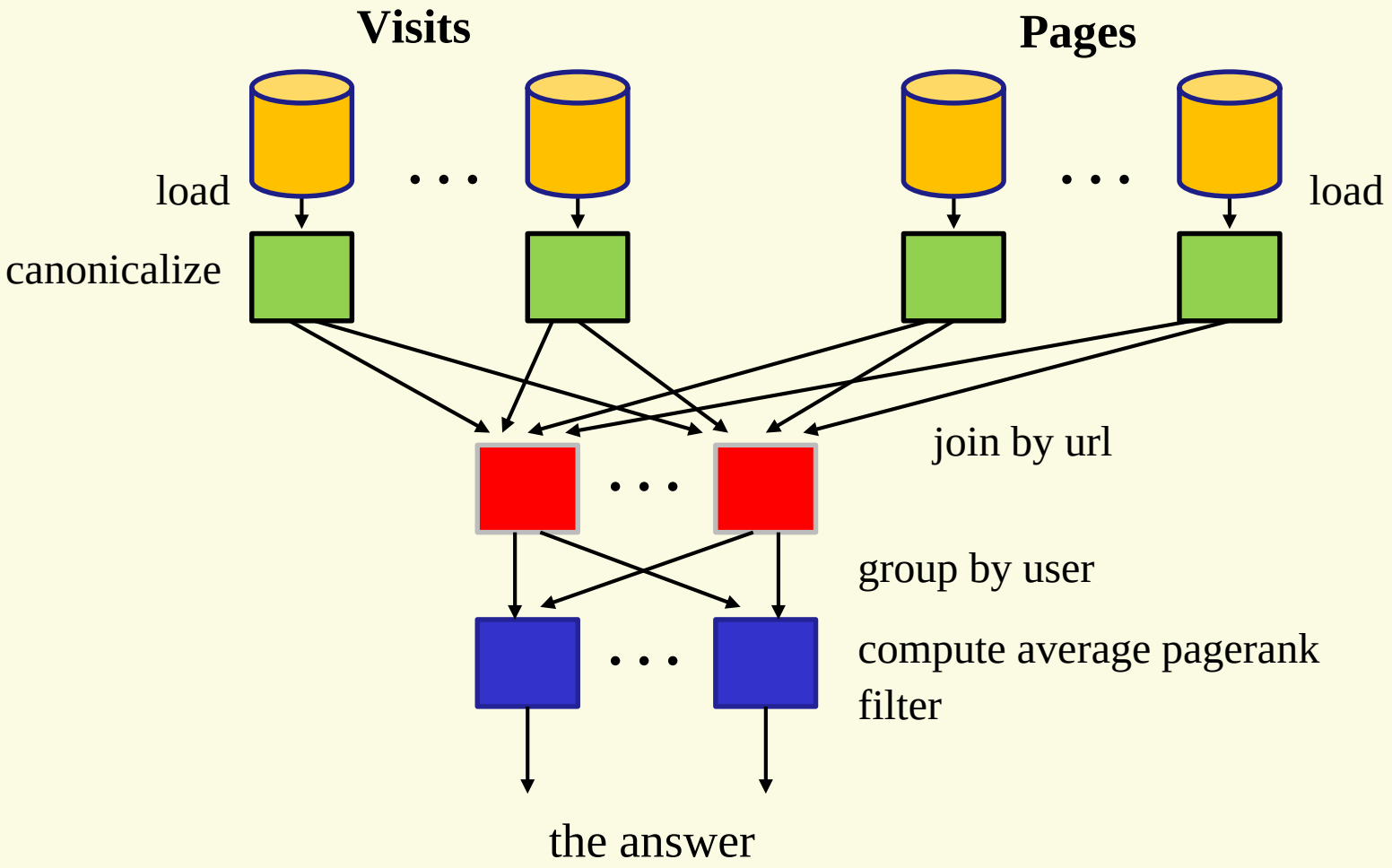
url	pagerank
www.cnn.com	0.9
www.flickr.com	0.9
www.myblog.com	0.7
www.crap.com	0.2

⋮

Conceptual Dataflow



System-Level Dataflow



MapReduce Code

```
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.a  pache.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.JobC     ontrol;
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.sub    string(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 w      e 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.to    String();
                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,
                OutputColle   ctor<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', first      Comma);
            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 ke   y,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            wh  ile (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        i mplements 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.se  tJobName("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.s  etJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add   InputPath(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(IdentityMap     per.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.se   tOutputPath(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(MRE     xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi     leOutputFormat.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(SequenceFileOutputF      ormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
         FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

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

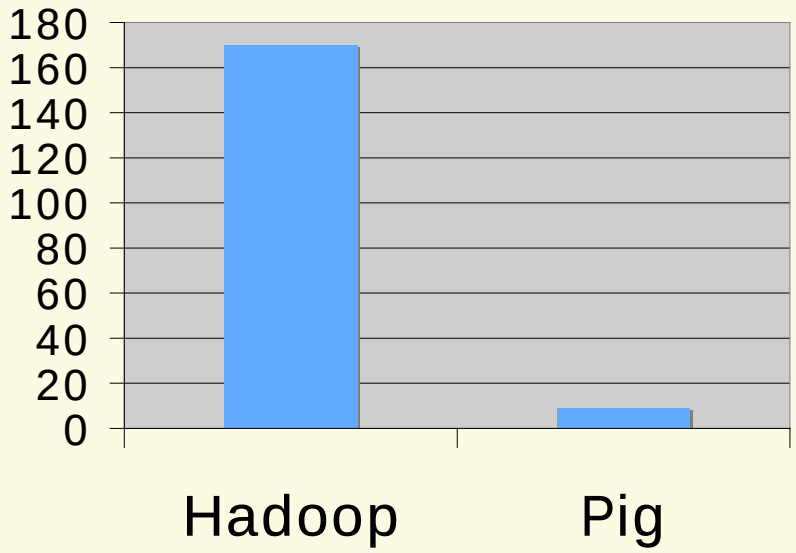
Pig Slides adapted from Olston et al.

Pig Latin Script

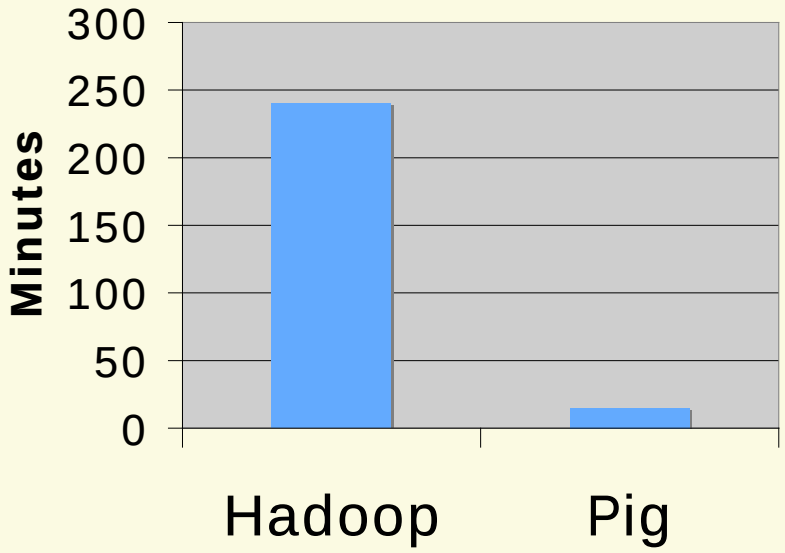
```
Visits = load    ‘/data/visits’ as (user, url, time);
Visits = foreach Visits generate user,
Canonicalize(url), time;

Pages = load    ‘/data/pages’ as (url, pagerank);

VP = join    Visits by url, Pages by url;
UserVisits = group   VP by user;
UserPageranks = foreach UserVisits generate user,
AVG(VP.pagerank) as avgpr;
GoodUsers = filter  UserPageranks by avgpr > ‘0.5’;

store   GoodUsers into '/data/good_users';
```

Java vs. Pig Latin

1/20 the lines of code



1/16 the development time



Performance on par with raw Hadoop

Pig takes care of…

- ✓ Schema and type checking
- ✓ Translating into efficient physical dataflow
  - (i.e., sequence of one or more MapReduce jobs)
- ✓ Exploiting data reduction opportunities
  - (e.g., early partial aggregation via a combiner)
- ✓ Executing the system-level dataflow
  - (i.e., running the MapReduce jobs)
- ✓ Tracking progress, errors, etc.

HBase

HBase - What?

- ✓ an **open-source**, **distributed**, **column-oriented** database built on top of HDFS based on BigTable
- ✓ Modeled on Google's Bigtable
- ✓ Row/column store
- ✓ Billions of rows/millions on columns
- ✓ Column-oriented - nulls are free
- ✓ Untyped - stores byte[]

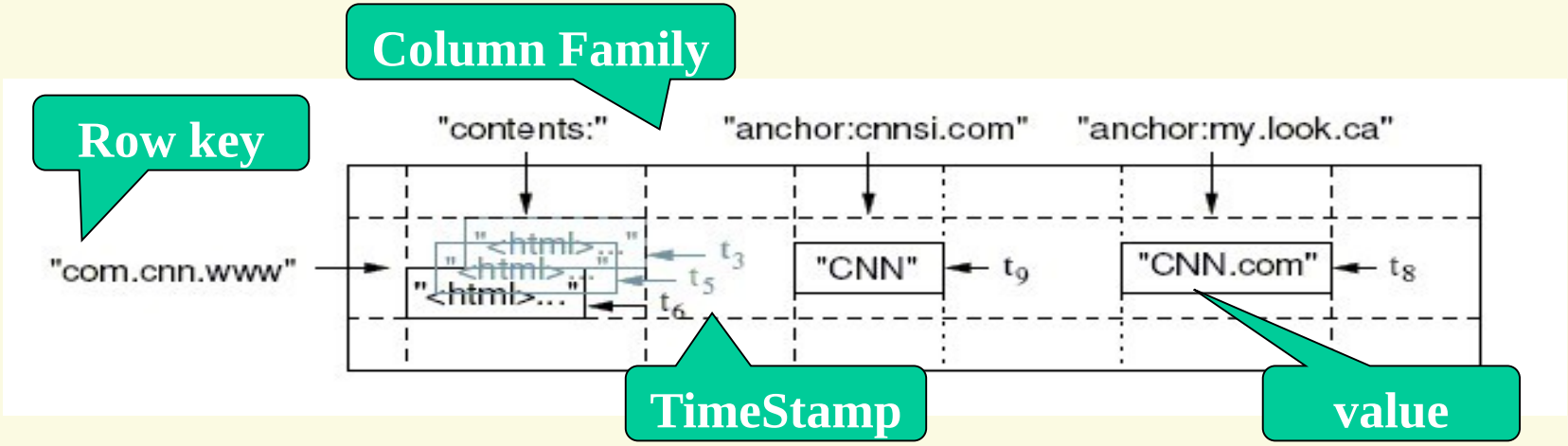
Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	…	…	…	…

HBase Is Not …

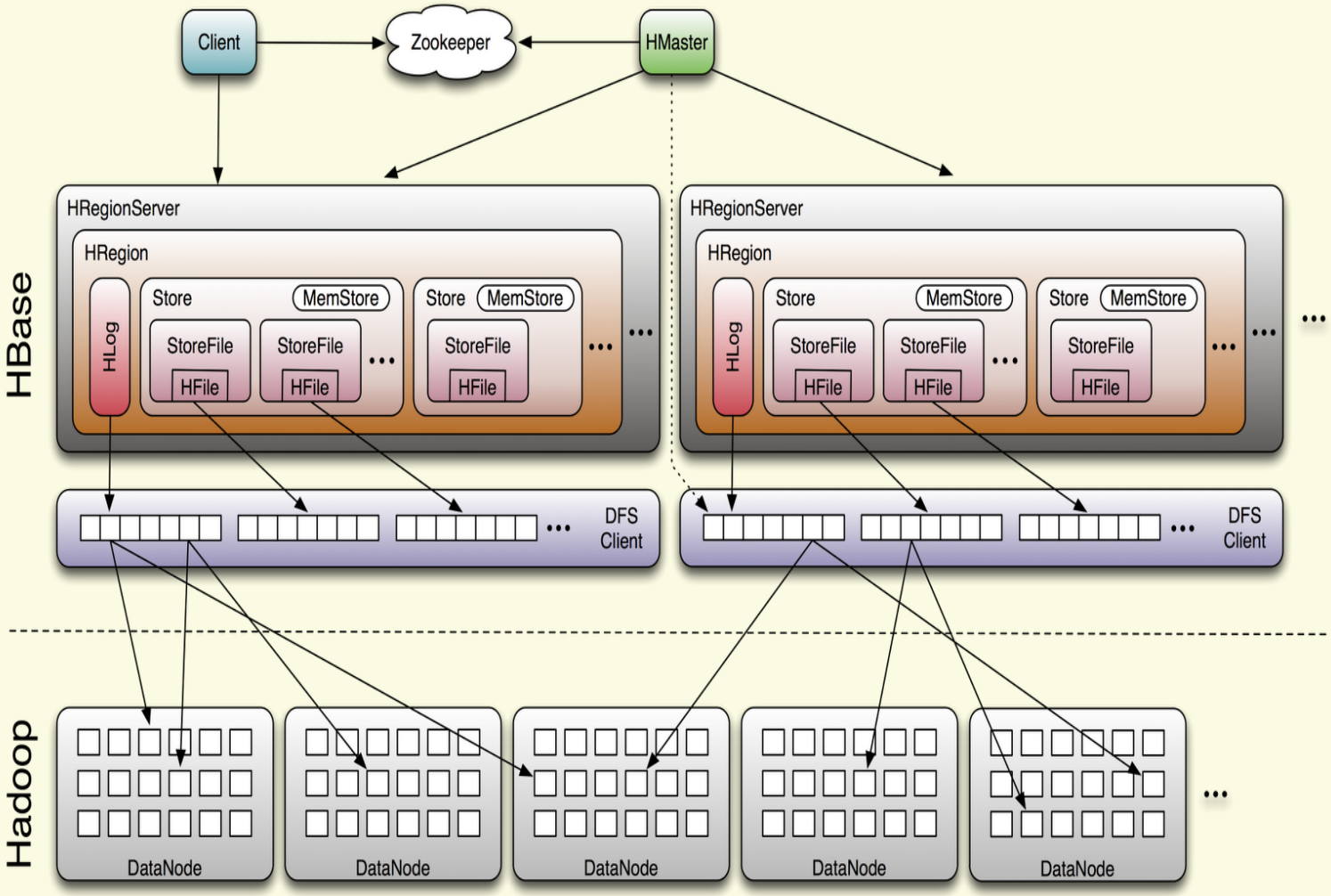
- ✓ Tables have one primary index, the *row key*.
- ✓ No join operators.
- ✓ Scans and queries can select a subset of available columns, perhaps by using a wildcard.
- ✓ There are three types of lookups:
  - Fast lookup using row key and optional timestamp.
  - Full table scan
  - Range scan from region start to end.
- ✓ Limited atomicity and transaction support.
  - HBase supports multiple batched mutations of single rows only.
  - Data is unstructured and untyped.
- ✓ No accessed or manipulated via SQL.
  - Programmatic access via Java, REST, or Thrift APIs.

Data Model

- ✓ Tables are sorted by Row
- ✓ Table schema only define it's *column families .*
  - Each family consists of any number of columns
  - Each column consists of any number of versions
  - Columns only exist when inserted, NULLs are free.
  - Columns within a family are sorted and stored together
- ✓ Everything except table names are byte[]
- ✓ (Row, Family: Column, Timestamp) → Value



HBase Architecture



HBase - Code

```
HTable table = …
Text row = new Text(“enclosure1”);
Text col1 = new Text(“animal:type”);
Text col2 = new Text(“animal:size”);
BatchUpdate update = new BatchUpdate(row);
update.put(col1, “lion”.getBytes(“UTF-8”));
update.put(col2, “big”.getBytes(“UTF-8));
table.commit(update);

update = new BatchUpdate(row);
update.put(col1, “zebra”.getBytes(“UTF-8”));
table.commit(update);
```

HBase - Querying

- Retrieve a cell

  Cell = table.getRow(“enclosure1”).getColumn(“animal:type”).getValue();
- Retrieve a row

  RowResult = table.getRow(“enclosure1”);
- Scan through a range of rows

  Scanner s = table.getScanner(new String[] { “animal:type” });

Aggregation and Joins

- Count users who visited each page by gender:

```
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

- Sample output:

page_url	gender	count(userid)
home.php	MALE	12,141,412
home.php	FEMALE	15,431,579
photo.php	MALE	23,941,451
photo.php	FEMALE	21,231,314

Apache ZooKeeper

- ✓ *Coordination*: An act that multiple nodes must perform together.
- ✓ Examples:
  - Group membership
  - Locking
  - Publisher/Subscriber
  - Leader Election
  - Synchronization
- ✓ Getting node coordination correct is very hard!

What is ZooKeeper?

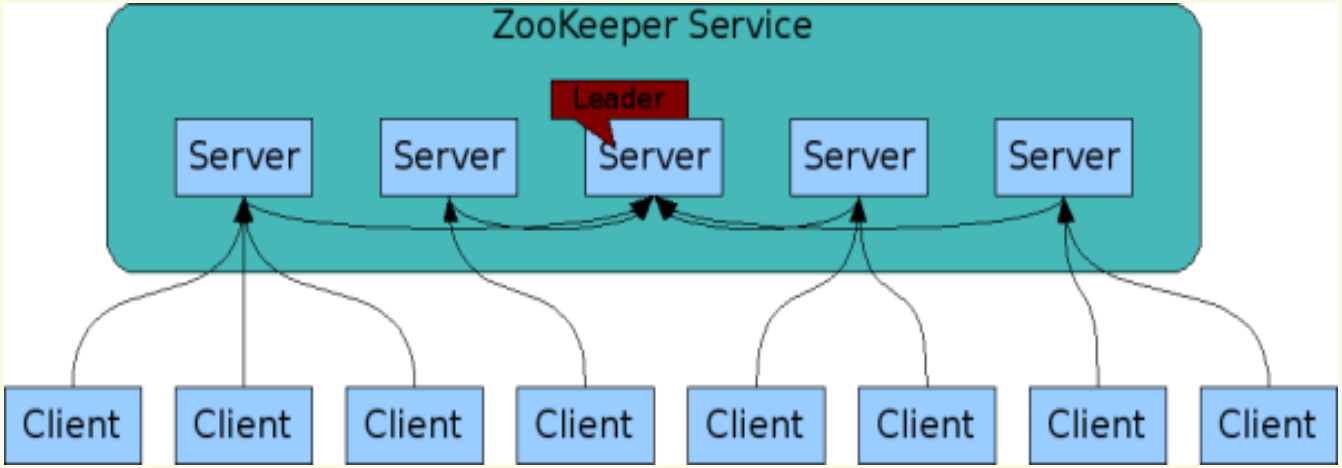
- An open source, high-performance coordination service for distributed applications.
- Exposes common services in simple interface:
  - naming
  - configuration management
  - locks & synchronization
  - group services

    … developers don't have to write them from scratch
- Build your own on it for specific needs.

ZooKeeper Use Cases

- ✓ Configuration Management
  - Cluster member nodes bootstrapping configuration from a centralized source in unattended way
  - Easier, simpler deployment/provisioning
- ✓ Distributed Cluster Management
  - Node join / leave
  - Node statuses in real time
- ✓ Naming service – e.g. DNS
- ✓ Distributed synchronization - locks, barriers, queues
- ✓ Leader election in a distributed system.
- ✓ Centralized and highly reliable (simple) data registry

The ZooKeeper Service

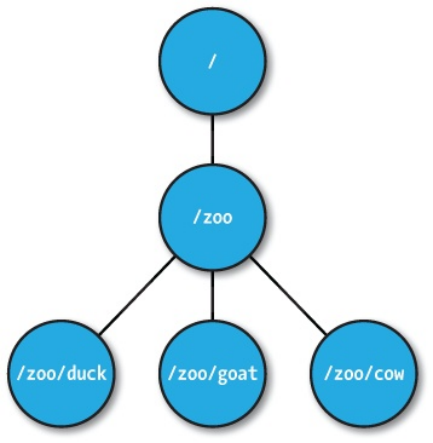


- ZooKeeper Service is replicated over a set of machines
- All machines store a copy of the data (in memory)
- A leader is elected on service startup
- Clients only connect to a single ZooKeeper server & maintains a TCP connection.
- Client can read from any Zookeeper server, writes go through the leader & needs majority consensus.

Image: https://cwiki.apache.org/confluence/display/ZOOKEEPER/ProjectDescription

ZNodes

- ✓ Maintain a stat structure with version numbers for data changes, ACL changes and timestamps.
- ✓ Version numbers increases with changes
- ✓ Data is read and written automatically



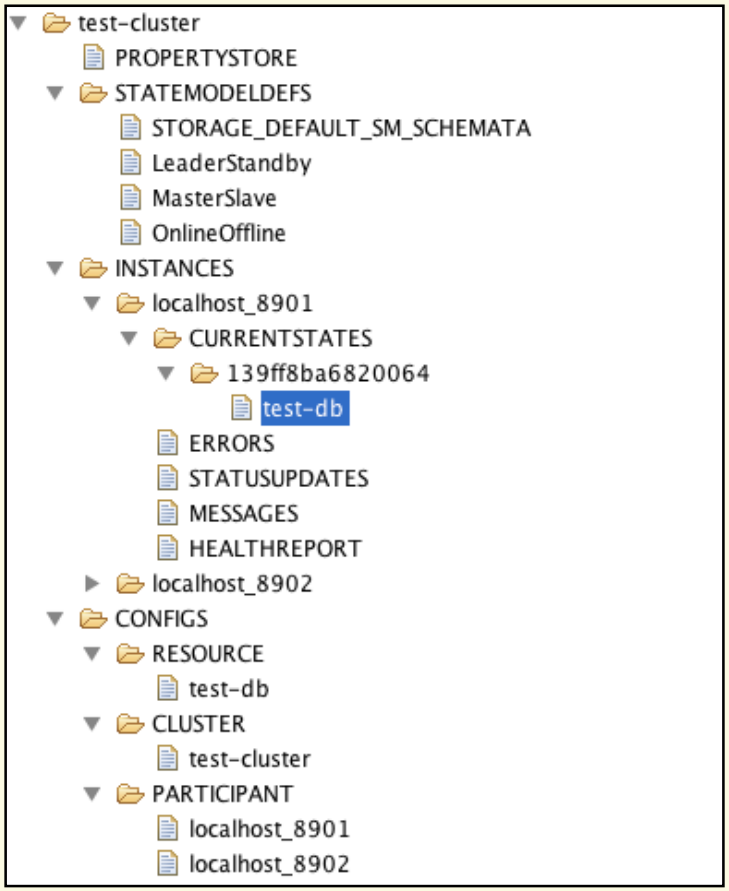


Image: http://helix.incubator.apache.org/Architecture.html

Consistency Guarantees

- ✓ **Sequential Consistency**: Updates are applied in order
- ✓ **Atomicity**: Updates either succeed or fail
- ✓ **Single System Image**: A client sees the same view of the service regardless of the ZK server it connects to.
- ✓ **Reliability**: Updates persists once applied, till overwritten by some clients.
- ✓ **Timeliness**: The clients' view of the system is guaranteed to be up-to-date within a certain time bound. (Eventual Consistency)

Who uses ZooKeeper?

Companies:

- Yahoo!
- Zynga
- Rackspace
- LinkedIn
- Netflix
- *and many more…*

Projects in FOSS:

- Apache Map/Reduce (Yarn)
- Apache HBase
- Apache Solr
- Neo4j
- Katta
- *and many more…*

Reference: https://cwiki.apache.org/confluence/display/ZOOKEEPER/PoweredBy

ZooKeeper In Action @Twitter

- ✓ Used within Twitter for service discovery
- ✓ How?
  - Services register themselves in ZooKeeper
  - Clients query the production cluster for service “A” in data center “XYZ”
  - An up-to-date host list for each service is maintained
  - Whenever new capacity is added the client will automatically be aware
  - Also, enables load balancing across all servers.

Reference: http://engineering.twitter.com/

Reading list

- ✓ The Google File System
- ✓ The Hadoop Distributed File System
- ✓ MapReduce: Simplified Data Processing on Large Clusters
- ✓ Bigtable: A Distributed Storage System for Structured Data
- ✓ PNUTS: Yahoo!’s Hosted Data Serving Platform
- ✓ Dynamo: Amazon's Highly Available Key-value Store
- ✓ Spanner: Google's Globally Distributed Database
- ✓ Centrifuge: Integrated Lease Management and Partitioning Cloud Services (Microsoft)
- ✓ ZAB: A simple totally ordered broadcast protocol (Yahoo!)
- ✓ Paxos Made Simple by Leslie Lamport.
- ✓ Eventually Consistent by Werner Vogel (CTO, Amazon)
- ✓ http://www.highscalability.com/