Inspire…Educate…Transform.

Engineering Big Data

The Rest of Hadoop

PIG, Oozie, Mahout, Kafka…

Dr. Sreerama KV Murthy
CEO, Quadratyx

Sep 12, 2015

Wake-Up Quiz

Pig : Building High-Level Dataflows over Map-Reduce

- Developed by Yahoo!, now open source
- Roughly 1/3 of all Yahoo! internal jobs
- Pig Latin Data Flow Language

Example Data Analysis Task

Find the top 10 most visited pages in each category

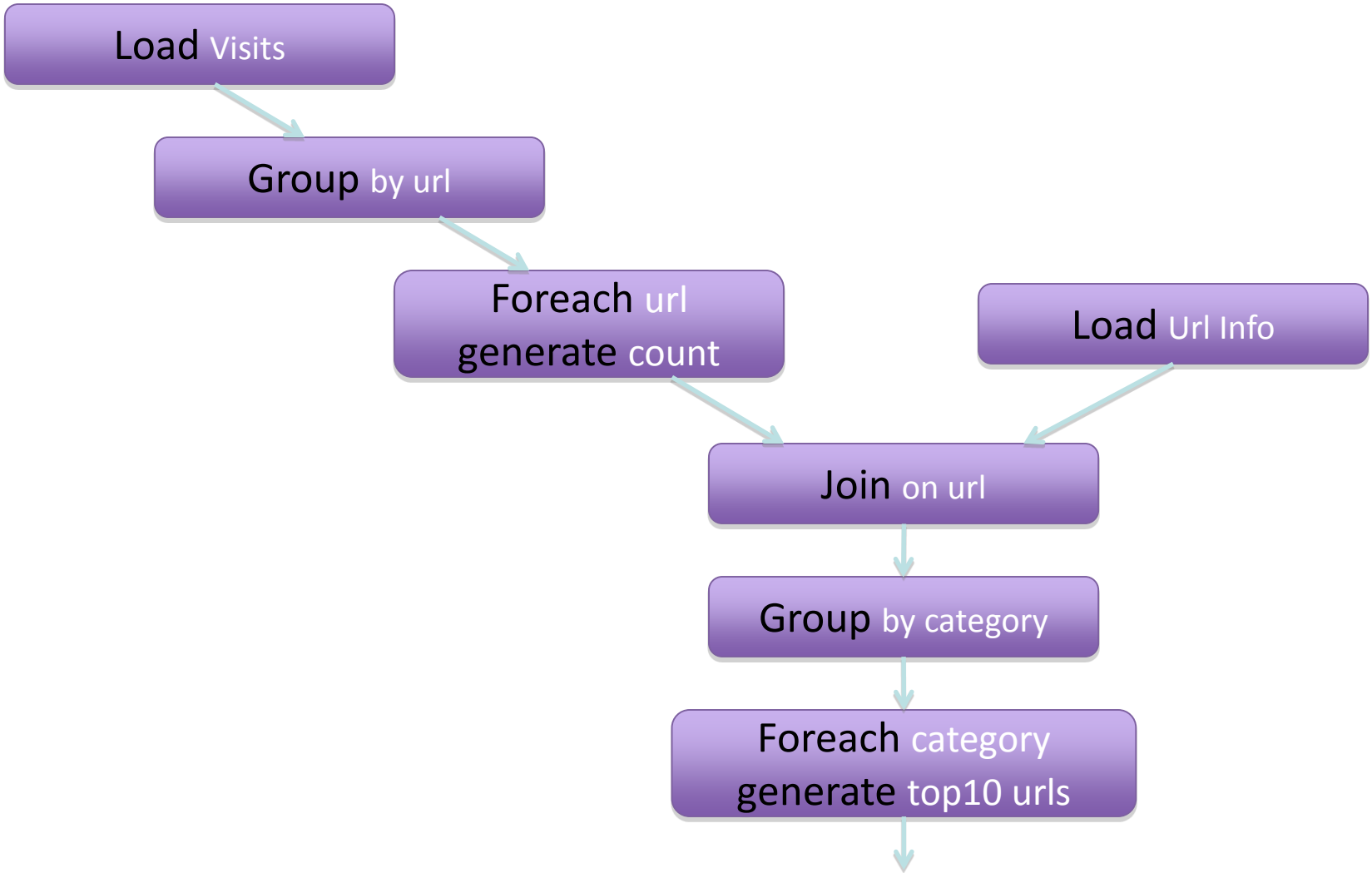
Visits

User	Url	Time
Amy	cnn.com	8:00
Amy	bbc.com	10:00
Amy	flickr.com	10:05
Fred	cnn.com	12:00

Url Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

Data Flow


Load Visits
Group by url
Foreach url generate count
Load Url Info
Join on url
Group by category
Foreach category generate top10 urls

In Pig Latin

```
visits       = load '/data/visits' as (user, url, time);
gVisits      = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo      = load '/data/urlInfo' as (url, category, pRank);
visitCounts  = join visitCounts by url, urlInfo by url;

gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

Quick Start and Interoperability

```
visits       = load '/data/visits' as (user, url, time);
gVisits      = group visits by url;
visitCounts  = foreach gVisits generate url, count(visits);

urlInfo      = load '/data/urlInfo' as (url, category, pRank);

visitCounts  =                                              url;
gCategories  =
topUrls = fore                                              Counts,10);

store topUrls into '/data/topUrls';
```

Operates directly over files

Quick Start and Interoperability

```
visits      = load '/data/visits' as (user, url, time);
gVisits     = group visits by url;
visitCounts = foreach gVisits generate url, count(urlVisits);

urlInfo     = load '/data/urlInfo' as (url, category, pRank);

visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = fo                                    Counts,10);

store topUrl
```

Schemas optional;
Can be assigned dynamically

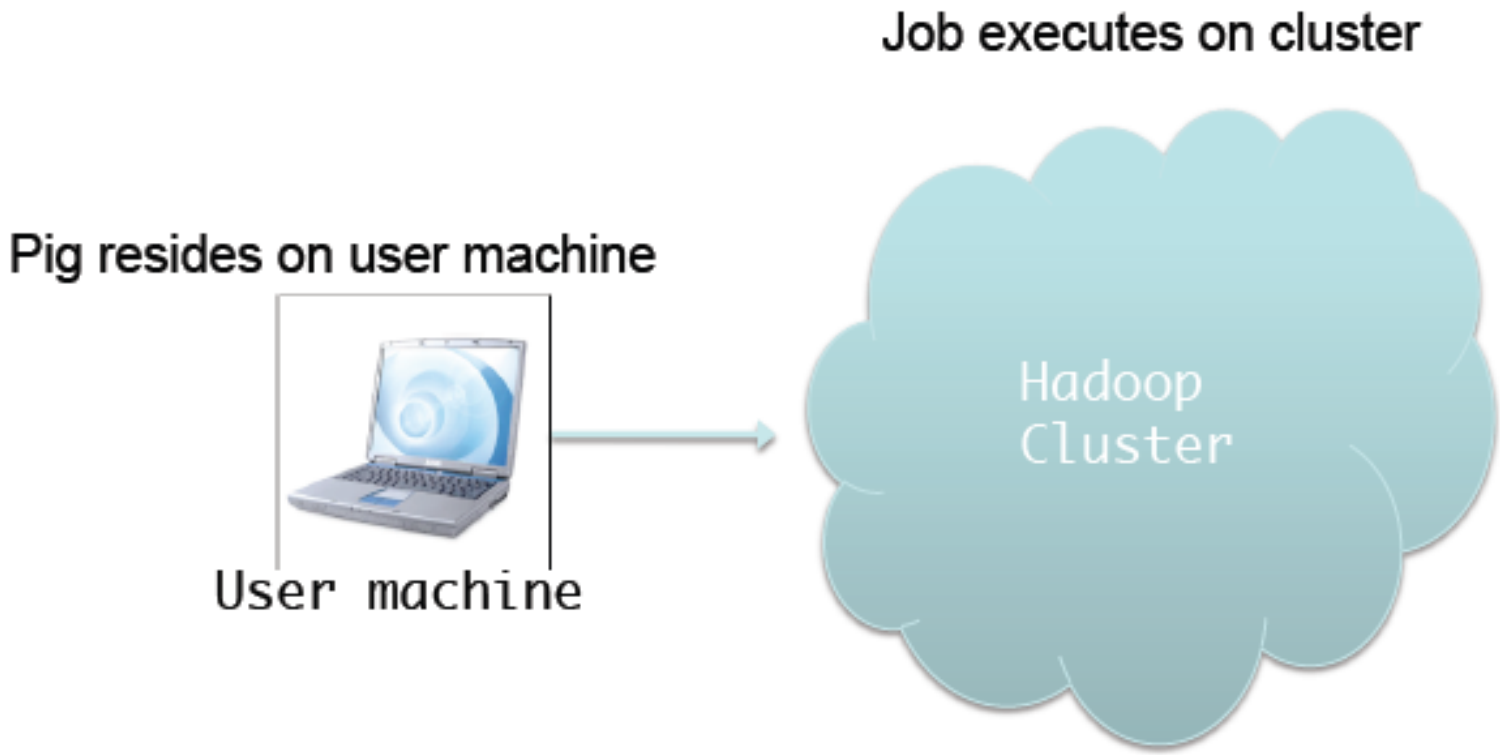
User-Code as a First-Class Citizen

```
visits      = load '/data/visits' as (user, url, time);
gVisits
visitCou                                    ount(urlVisits);

urlInfo                                     ategory, pRank);

visitCounts  = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

User-defined functions (UDFs) can be used in every construct

- Load, Store
- Group, Filter, Foreach

Components of PIG



No need to install anything extra on your Hadoop cluster.

Some slides are Cloudera copyrighted. We replicate them with zero modifications and include references to Cloudera.

Data Model

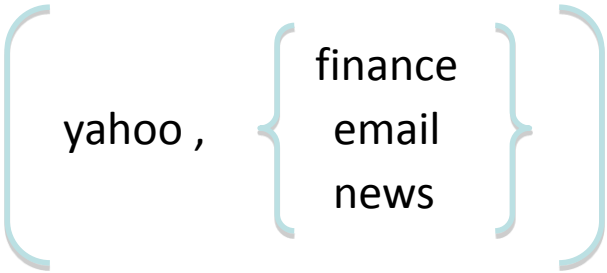
- Supports four basic types
  - **Atom**: a simple atomic value (*int*, *long*, *double*, *string*)
    - ex: ‘Peter’
  - **Tuple**: a sequence of fields that can be any of the data types
    - ex: (‘Peter’, 14)
  - **Bag**: a collection of tuples of potentially varying structures, can contain duplicates
    - ex: {(‘Peter’), (‘Bob’, (14, 21))}
  - **Map**: an associative array, the key must be a *chararray* but the value can be any type

Data Model (continued)

- By default Pig treats undeclared fields as *bytearrays* (collection of uninterpreted bytes)
- Can infer a field's type based on:
  - Use of operators that expect a certain type of field
  - UDFs with a known or explicitly set return type
  - Schema information provided by a LOAD function or explicitly declared using an AS clause
- Type conversion is lazy

Nested Data Model

- Pig Latin has a fully-nestable data model with:
  - Atomic values, tuples, bags (lists), and maps



- More natural to programmers than flat tuples
- Avoids expensive joins

Type Casting

- By default Pig treats data as un-typed.
- User can declare types of data at load time.

```
log = LOAD 'shakespeare_freq'
          AS (freq:int, word: chararray);
```

- If data type is not declared but script treats value as a certain type, Pig will assume it is of that type and cast it.

```
log      = LOAD 'shakespeare_freq' AS (freq, word);
weighted = FOREACH log
           GENERATE freq * 100; --freq cast to int
```

Some PIG Commands

Pig Command	What it does
load	Read data from file system.
store	Write data to file system.
foreach	Apply expression to each record and output one or more records.
filter	Apply predicate and remove records that do not return true.
group/cogroup	Collect records with the same key from one or more inputs.
join	Join two or more inputs based on a key.
order	Sort records based on a key.
distinct	Remove duplicate records.
union	Merge two data sets.
split	Split data into 2 or more sets, based on filter conditions.
stream	Send all records through a user provided binary.
dump	Write output to stdout.
limit	Limit the number of records.

CoGroup

results

query	url	rank
Lakers	nba.com	1
Lakers	espn.com	2
Kings	nhl.com	1
Kings	nba.com	2

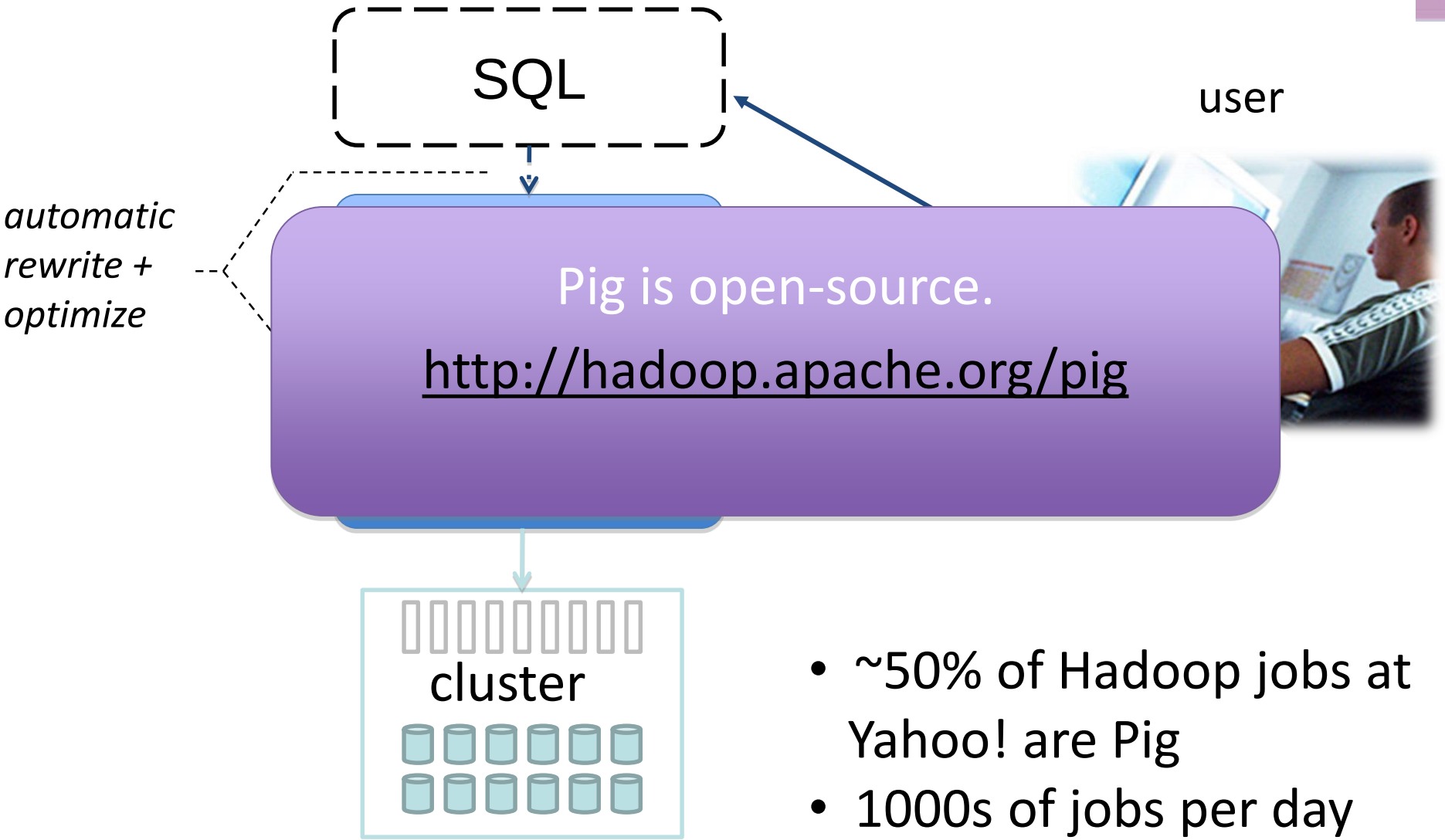
revenue

query	adSlot	amount
Lakers	top	50
Lakers	side	20
Kings	top	30
Kings	side	10

group	results			revenue		
Lakers	Lakers	nba.com	1	Lakers	top	50
	Lakers	espn.com	2	Lakers	side	20
Kings	Kings	nhl.com	1	Kings	top	30
	Kings	nba.com	2	Kings	side	10

Cross-product of the 2 bags would give natural join

Implementation



- ~50% of Hadoop jobs at Yahoo! are Pig
- 1000s of jobs per day

How does Pig work? (contd.)

Pig Latin

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

pig.jar:

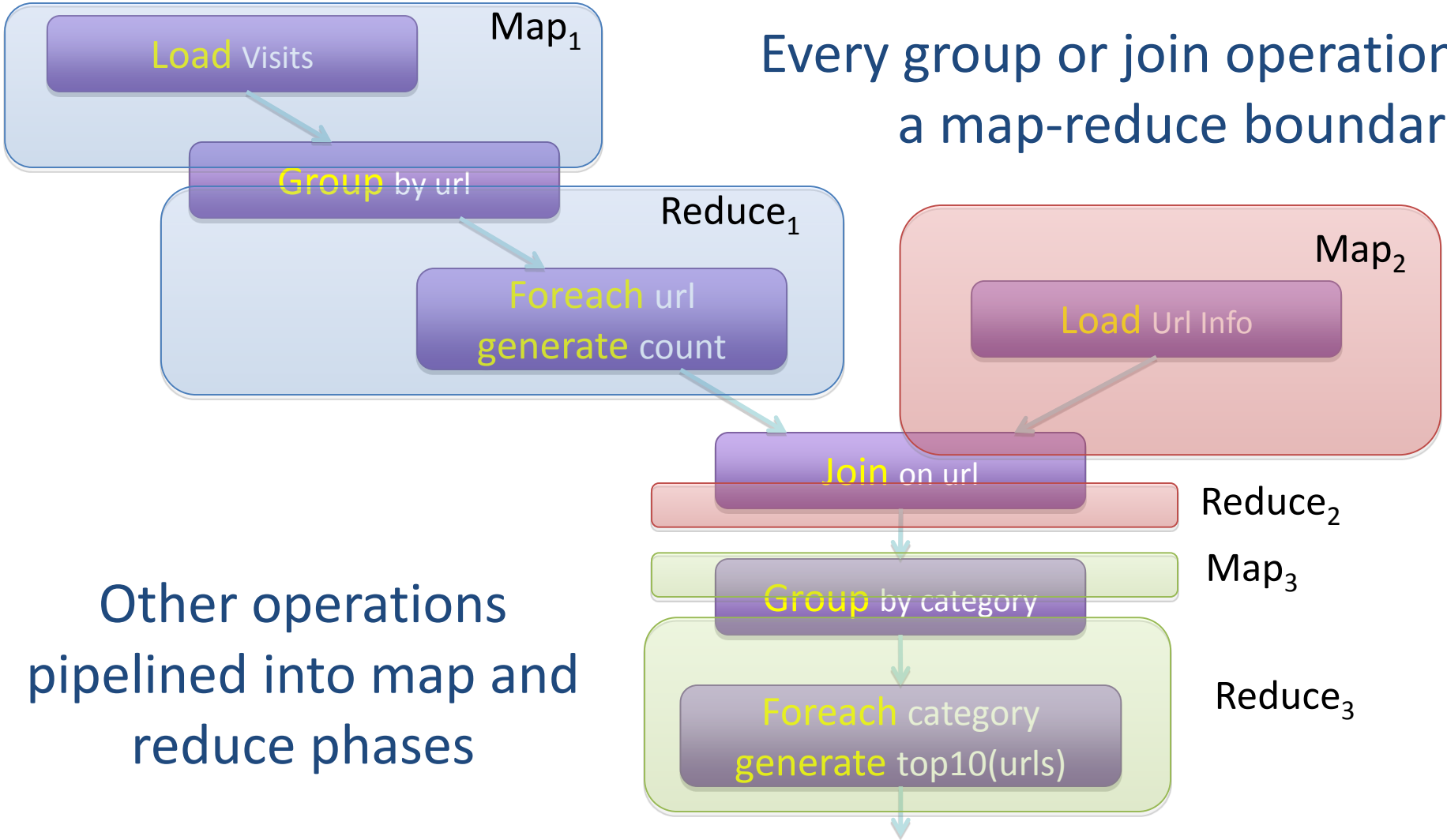
- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan

Map:
Filter

Reduce:
Count

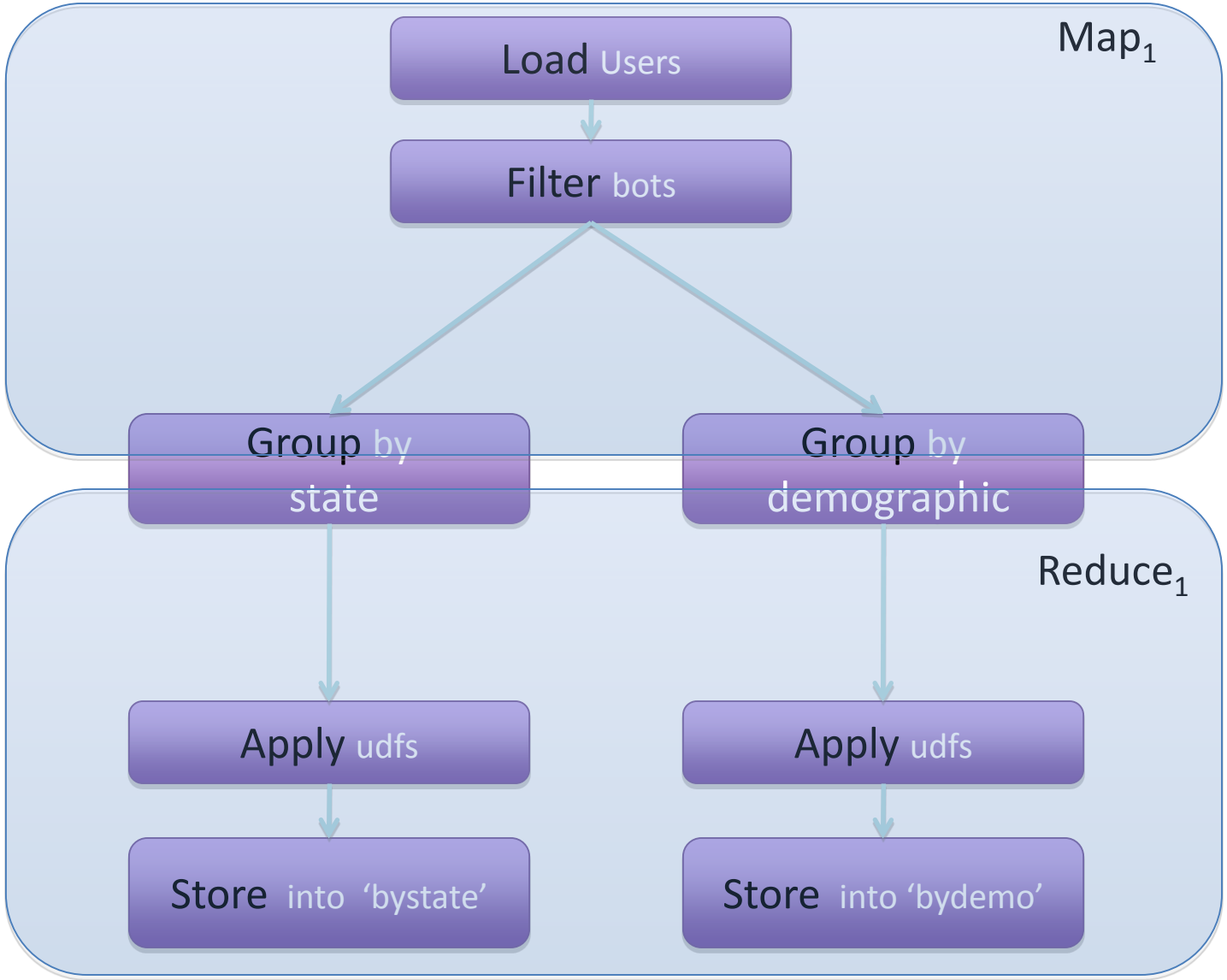
Compilation into Map-Reduce



Every group or join operation forms a map-reduce boundary

Other operations pipelined into map and reduce phases

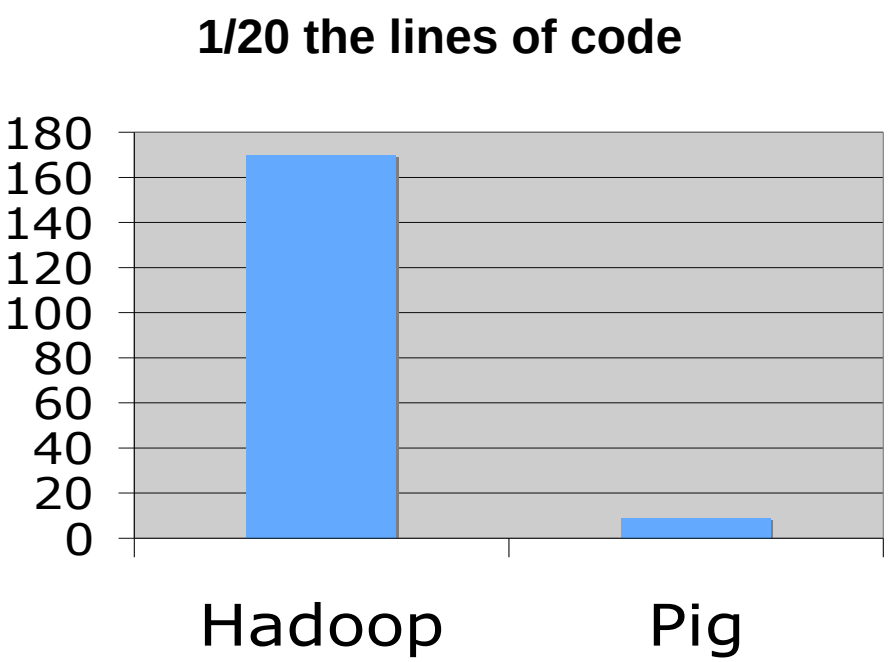
Optimizations: Multiple Data Flows

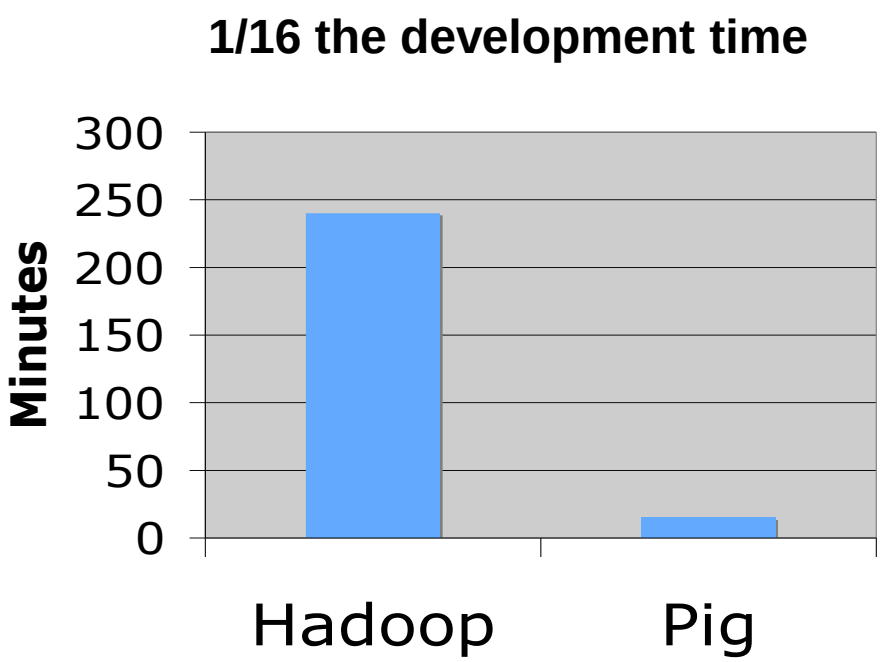


MapReduce Code

```
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.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
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to
                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.setOutputKeyClas
        lp.setOutputValueCl
        lp.setMapperClass(L
        FileInputFormat.add
        FileOutputFormat.se
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class)
        lfu.setMapperClass(LoadAndFilterUse
        FileInputFormat.addInputPath(lfu, new
        FileOutputFormat.setOutputPath(lfu,
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pag
        join.setInputFormat(KeyValueTextInp
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join,
        FileInputFormat.addInputPath(join,
        FileOutputFormat.setOutputPath(join,
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRE
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextIn
        group.setOutputKeyClass(Text.class)
        group.setOutputValueClass(LongWrita
        group.setOutputFormat(SequenceFileI
        group.setMapperClass(LoadJoined.cla
        group.setCombinerClass(ReduceUrls.c
        group.setReducerClass(ReduceUrls.cl
        FileInputFormat.addInputPath(group,
        FileOutputFormat.setOutputPath(group,
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExam
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileI
        top100.setOutputKeyClass(LongWritab
        top100.setOutputValueClass(Text.cla
        top100.setOutputFormat(SequenceFileO
        top100.setMapperClass(LoadClicks.cl
        top100.setCombinerClass(LimitClicks
        top100.setReducerClass(LimitClicks.
        FileInputFormat.addInputPath(top100
        FileOutputFormat.setOutputPath(top100,
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Fin
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

Java vs. Pig Latin

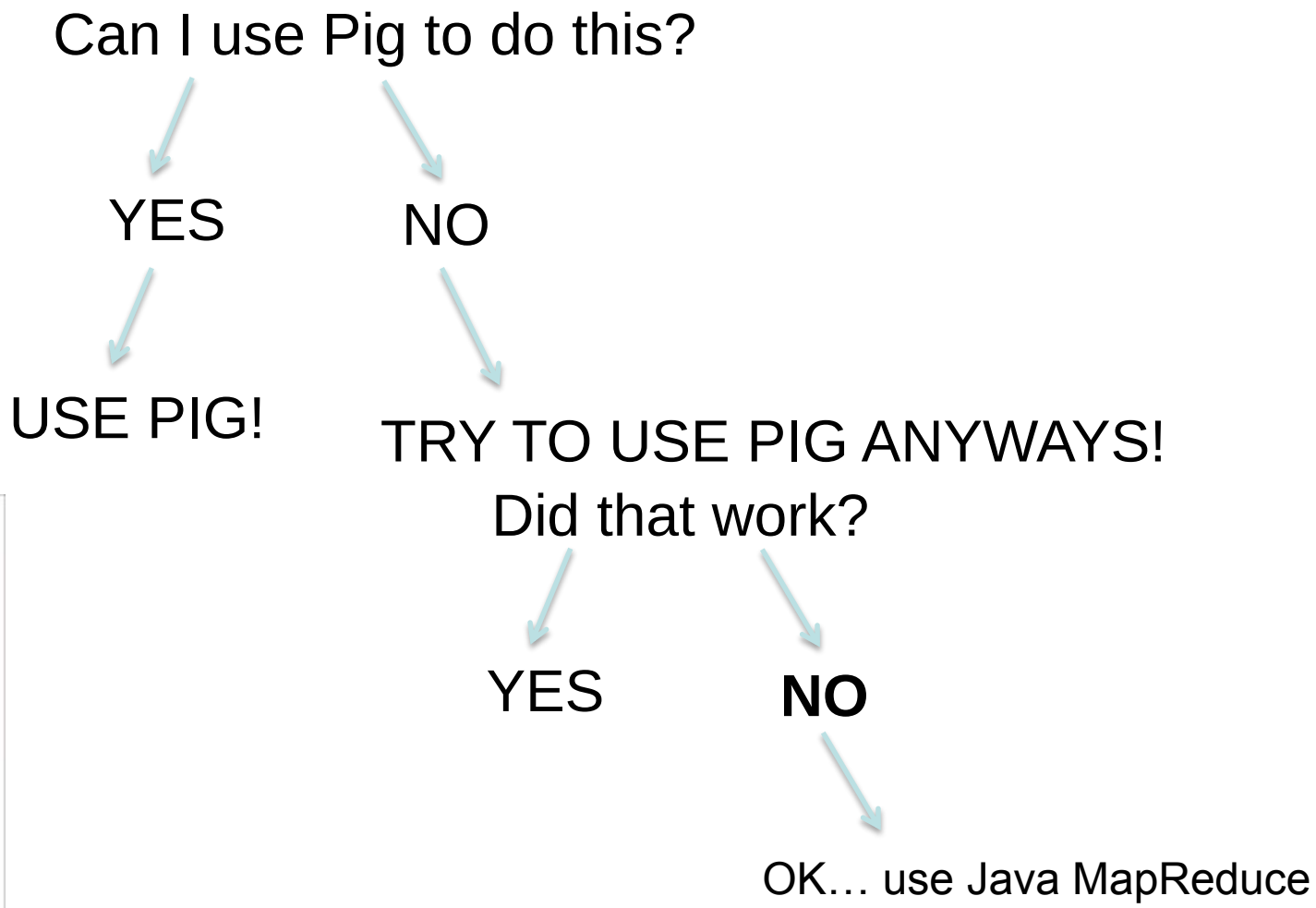


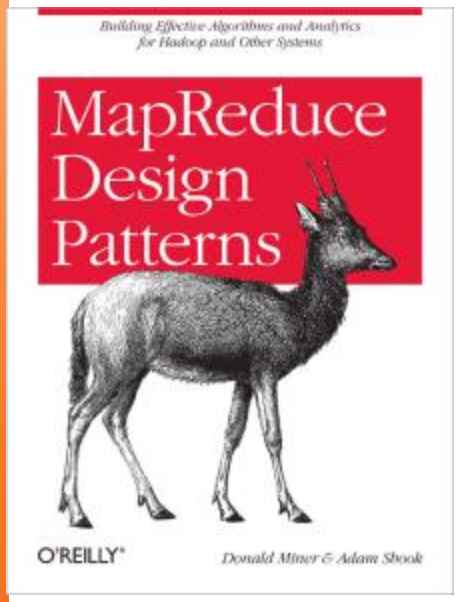


Performance on par with raw Hadoop!

When to use PIG, When to use Map-Reduce?

Donald Miner (August 2013)

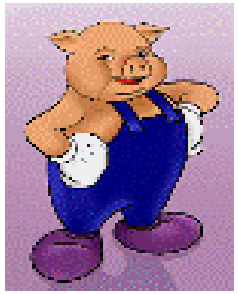




Which is faster, Pig or Java MapReduce?

Hypothetically, any Pig job could be rewritten using MapReduce… so Java MR can only be *faster*.

The TRUE battle is the
Pig optimizer vs. the developer



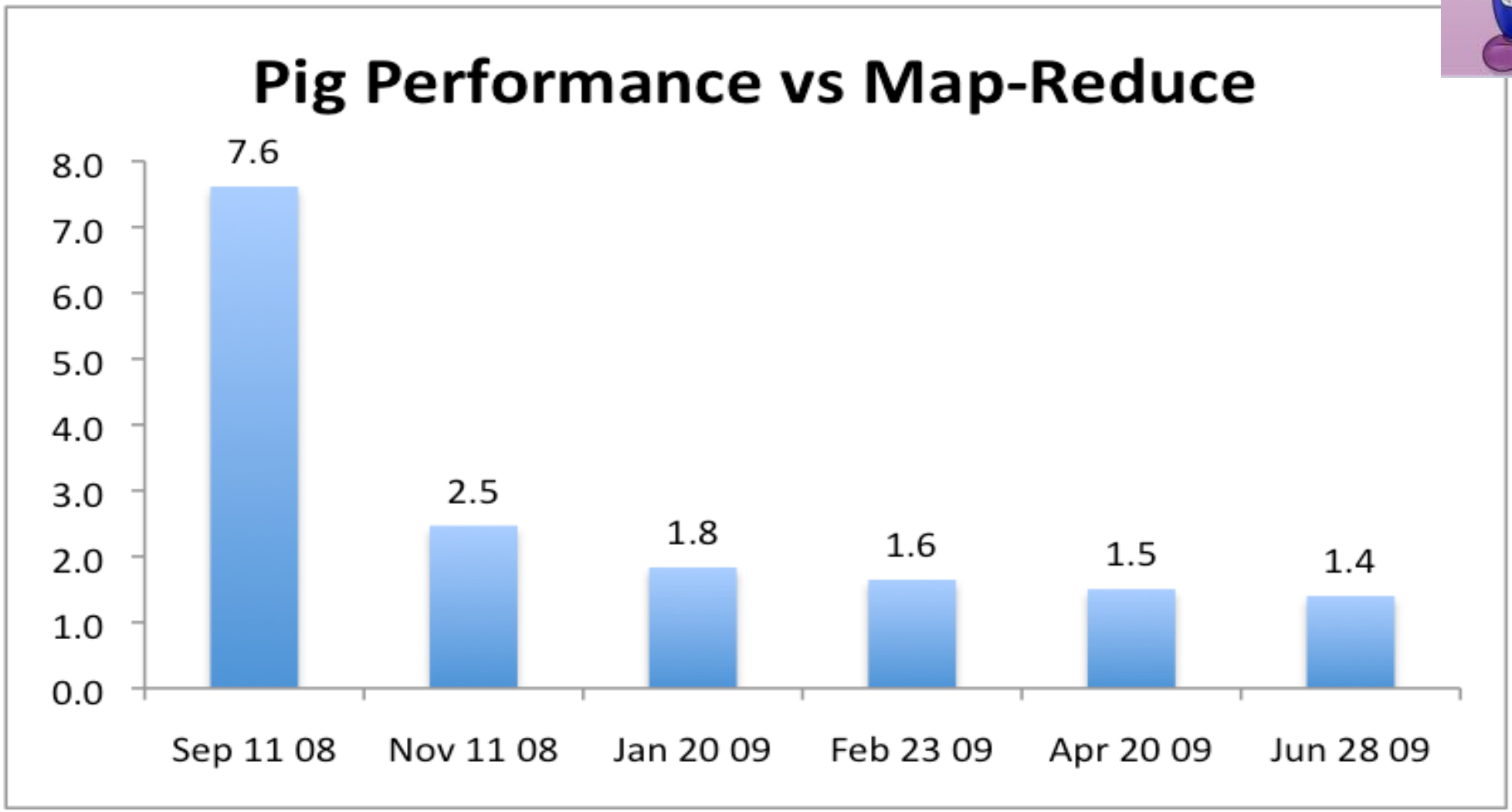
vs

Are you better than the Pig optimizer than figuring out how to string multiple jobs together (and other things)?

PIG Optimizer

- Combiner
  - When the Combiner is Used
  - When the Combiner is Not Used
- Hash-based Aggregation in Map Task
- Memory Management
- Reducer Estimation
- Multi-Query Execution
  - Turning it On or Off
  - How it Works
  - Store vs. Dump
  - Error Handling
  - Backward Compatibility
  - Implicit Dependencies
- Optimization Rules
  - FilterLogicExpressionSimplifier
  - SplitFilter
  - PushUpFilter
  - MergeFilter
  - PushDownForEachFlatten
  - LimitOptimizer
  - ColumnMapKeyPrune
  - AddForEach
  - MergeForEach
  - GroupByConstParallelSetter
- Performance Enhancers
  - Use Optimization
  - Use Types
  - Project Early and Often
  - Filter Early and Often
  - Reduce Your Operator Pipeline
  - Make Your UDFs Algebraic
  - Use the Accumulator Interface
  - Drop Nulls Before a Join
  - Take Advantage of Join Optimizations
  - Use the Parallel Features
  - Use the LIMIT Operator
  - Prefer DISTINCT over GROUP BY/GENERATE
  - Compress the Results of Intermediate Jobs
  - Combine Small Input Files

Performance



Also see: https://cwiki.apache.org/confluence/display/PIG/PigMix

How does Pig work? (contd.)

- Submit a script directly.
- Grunt, the pig shell.
- PigServer Java class, a JDBC like interface.
- PigPen, an eclipse plugin
  - Allows textual and graphical scripting.
  - Samples data and shows example data flow.

- Start a terminal and run
  ```
  $ cd /usr/share/cloudera/pig/
  $ bin/pig -x local
  ```
- Should see a prompt like:
  ```
  grunt>
  ```

Typical applications of Pig

- Web log processing.
- Data processing for web search platforms.
- Ad hoc queries across large data sets.
- Rapid prototyping of algorithms for processing large data sets.

Some important Pig points

Nothing really happens until a DUMP or STORE is performed.

Use FILTER and FOREACH early to remove unneeded columns or rows to reduce temporary output

Use PARALLEL keyword on GROUP operations to run more reduce tasks

http://pig.apache.org/docs/r0.7.0/cookbook.html#Use+the+PARALLEL+Clause

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.

Future / In-Progress Tasks

- Columnar-storage layer
- Metadata repository
- Profiling and Performance Optimizations
- Tight integration with a scripting language
  - Use loops, conditionals, functions of host language
- Memory Management
- Project Suggestions at:

  http://wiki.apache.org/pig/ProposedProjects

Pig Latin

Sweet spot between map-reduce and SQL

VERSUS

Pig procedural v/s SQL declarative

PIG

- Users = load 'users' as (name, age, ipaddr);
- Clicks = load 'clicks' as (user, url, value);
- ValuableClicks = filter Clicks by value > 0;
- UserClicks = join Users by name, ValuableClicks by user;
- Geoinfo = load 'geoinfo' as (ipaddr, dma);
- UserGeo = join UserClicks by ipaddr, Geoinfo by ipaddr; ByDMA = group UserGeo by dma;
- ValuableClicksPerDMA = foreach ByDMA generate group, COUNT(UserGeo);
- store ValuableClicksPerDMA into 'ValuableClicksPerDMA';

SQL

```
insert into ValuableClicksPerDMA
    select dma, count(*) from
    geoinfo join ( select name,
    ipaddr from users join clicks on
    (users.name = clicks.user)
    where value > 0; ) using
    ipaddr group by dma;
```

Pig v/s SQL

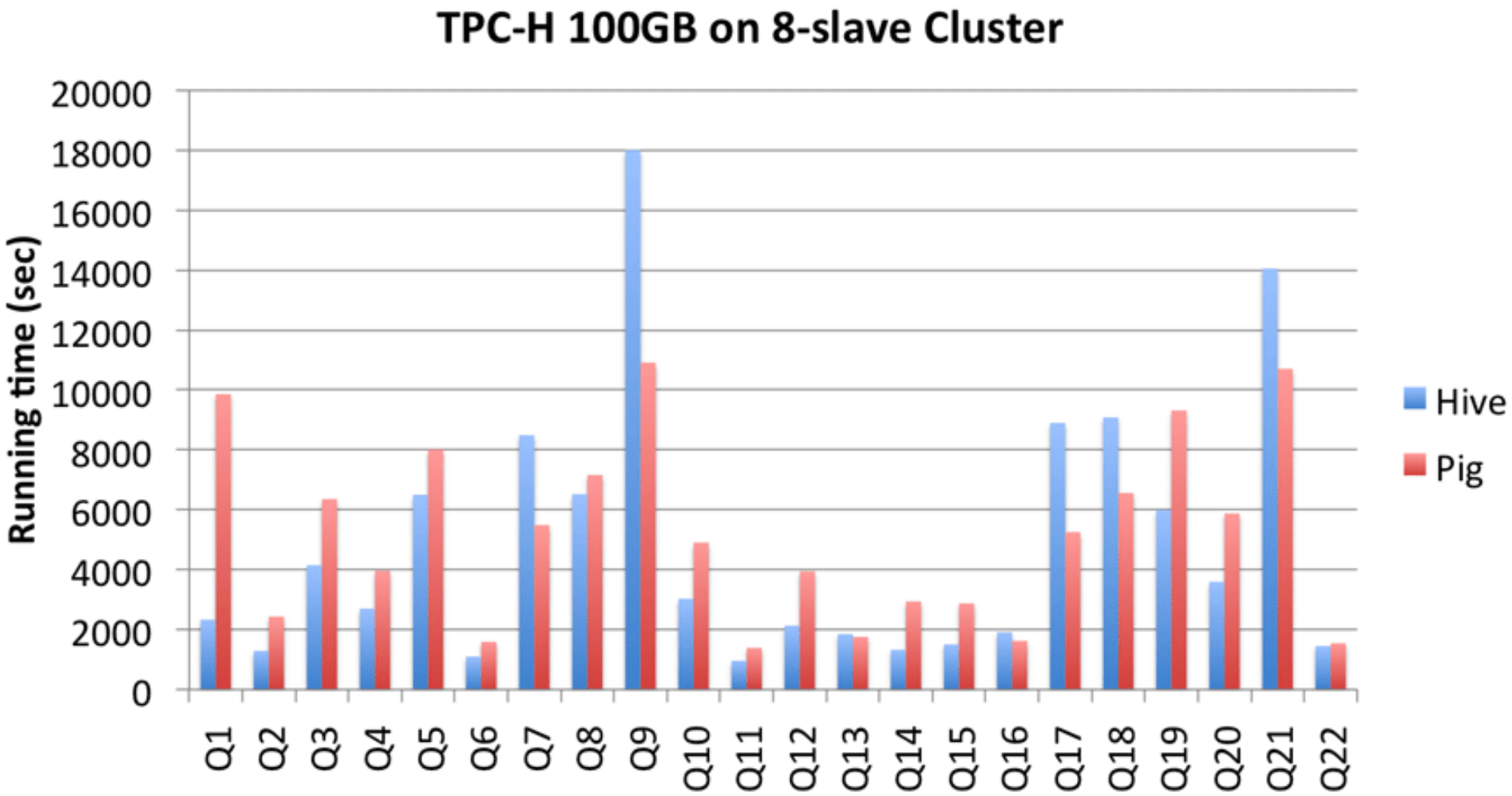
Pig

- Pig is procedural
- Nested relational data model (No constraints on Data Types)
- Schema is optional
- Scan-centric analytic workloads (No Random reads or writes)
- Limited query optimization

SQL

- SQL is declarative
- Flat relational data model (Data is tied to a specific Data Type)
- Schema is required
- OLTP + OLAP workloads
- Significant opportunity for query optimization

Performance Comparison



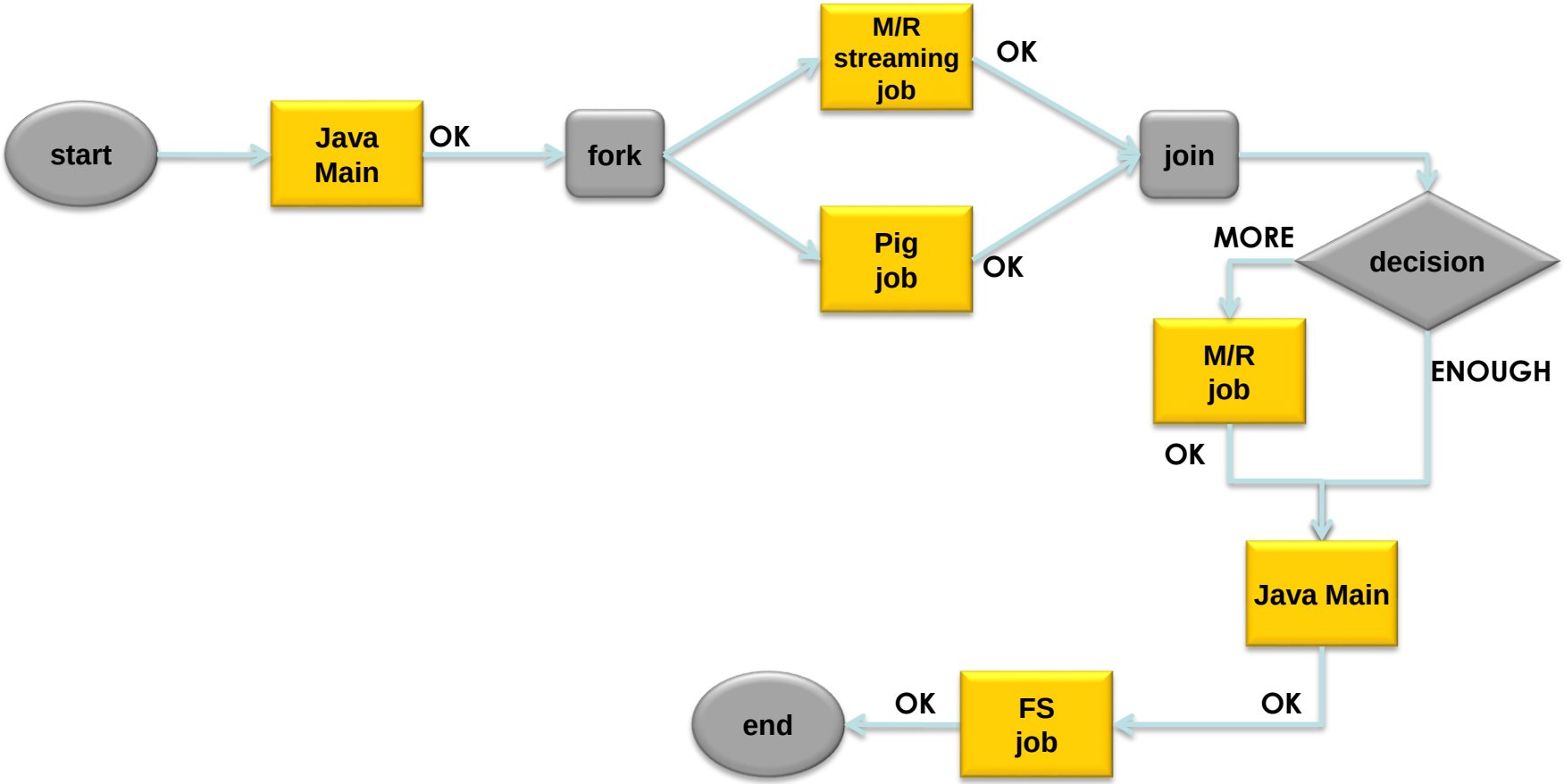
http://www.ibm.com/developerworks/library/ba-pigvhive/pighivebenchmarking.pdf

http://hortonworks.com/blog/pig-performance-and-optimization-analysis/ (Sep 2012)


OOZIE

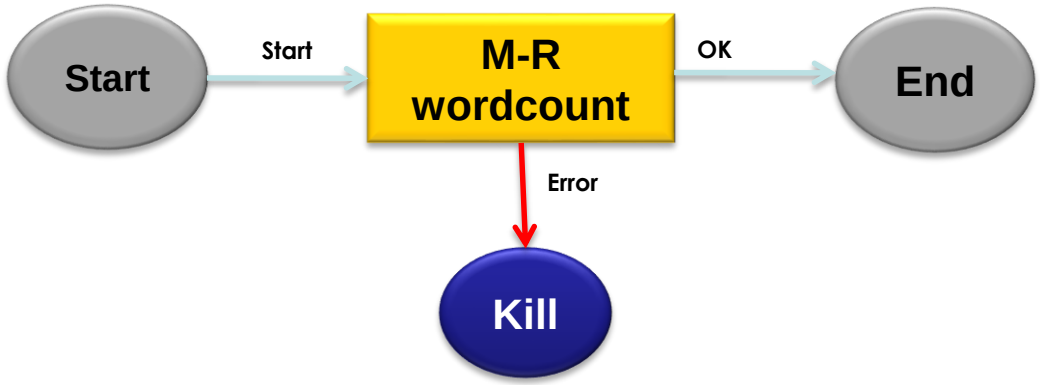
Oozie Workflow

Directed Acyclic Graph of Jobs



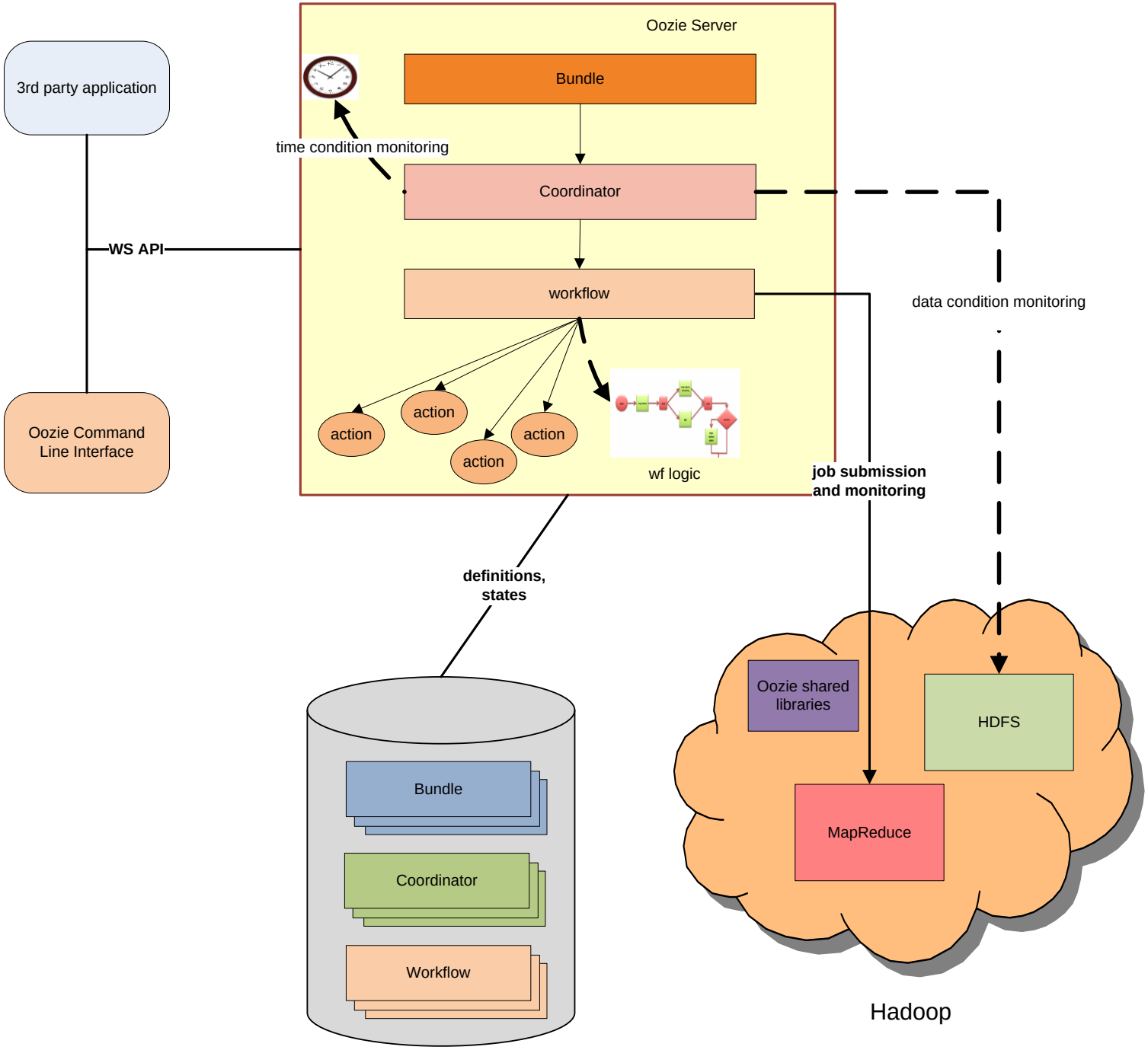
Oozie Workflow Example

```
<workflow-app name='wordcount-wf'>
  <start to='wordcount'/>

  <action name='wordcount'>
    <map-reduce>
      <job-tracker>foo.com:9001</job-tracker>
      <name-node>hdfs://bar.com:9000</name-node>
      <configuration>
        <property>
           <name>mapred.input.dir</name>
           <value>${inputDir}</value>
        </property>
        <property>
           <name>mapred.output.dir</name>
           <value>${outputDir}</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to='end'/>
    <error to='kill'/>
  </action>

  <kill name='kill'/>
  <end name='end'/>
</workflow-app>
```



https://issues.apache.org/jira/browse/OOZIE-883

Oozie Architecture



Oozie Elements

Flow-control node	XML element type	Description
Decision	workflow:DECISION	expressing “switch-case” logic
Fork	workflow:FORK	splits one path of execution into multiple concurrent paths
Join	workflow:JOIN	waits until every concurrent execution path of a previous fork node arrives to it
Kill	workflow:kill	forces a workflow job to kill (abort) itself

Action node	XML element type	Description
java	workflow:JAVA	invokes the main() method from the specified java class
fs	workflow:FS	manipulate files and directories in HDFS; supports commands: move, delete, mkdir
MapReduce	workflow:MAP-REDUCE	starts a Hadoop map/reduce job; that could be java MR job, streaming job or pipe job
Pig	workflow:pig	runs a Pig job
Sub workflow	workflow:SUB-WORKFLOW	runs a child workflow job
Hive *	workflow:HIVE	runs a Hive job
Shell *	workflow:SHELL	runs a Shell command
ssh *	workflow:SSH	starts a shell command on a remote machine as a remote secure shell
Sqoop *	workflow:SQOOP	runs a Sqoop job
Email *	workflow:EMAIL	sending emails from Oozie workflow application
Distcp ?		Under development (Yahoo)

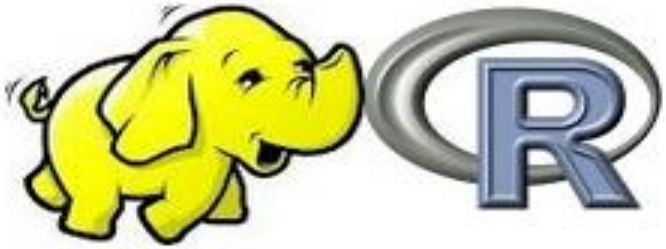
Apache Mahout

http://www.slideshare.net/Cataldo/tutorial-mahout-2015

MLearning on Hadoop - Options

- **Apache Mahout**
  - Open-source package on Hadoop for data mining and machine learning

- **R-Hadoop**
  - Extensions to R package to run on Hadoop



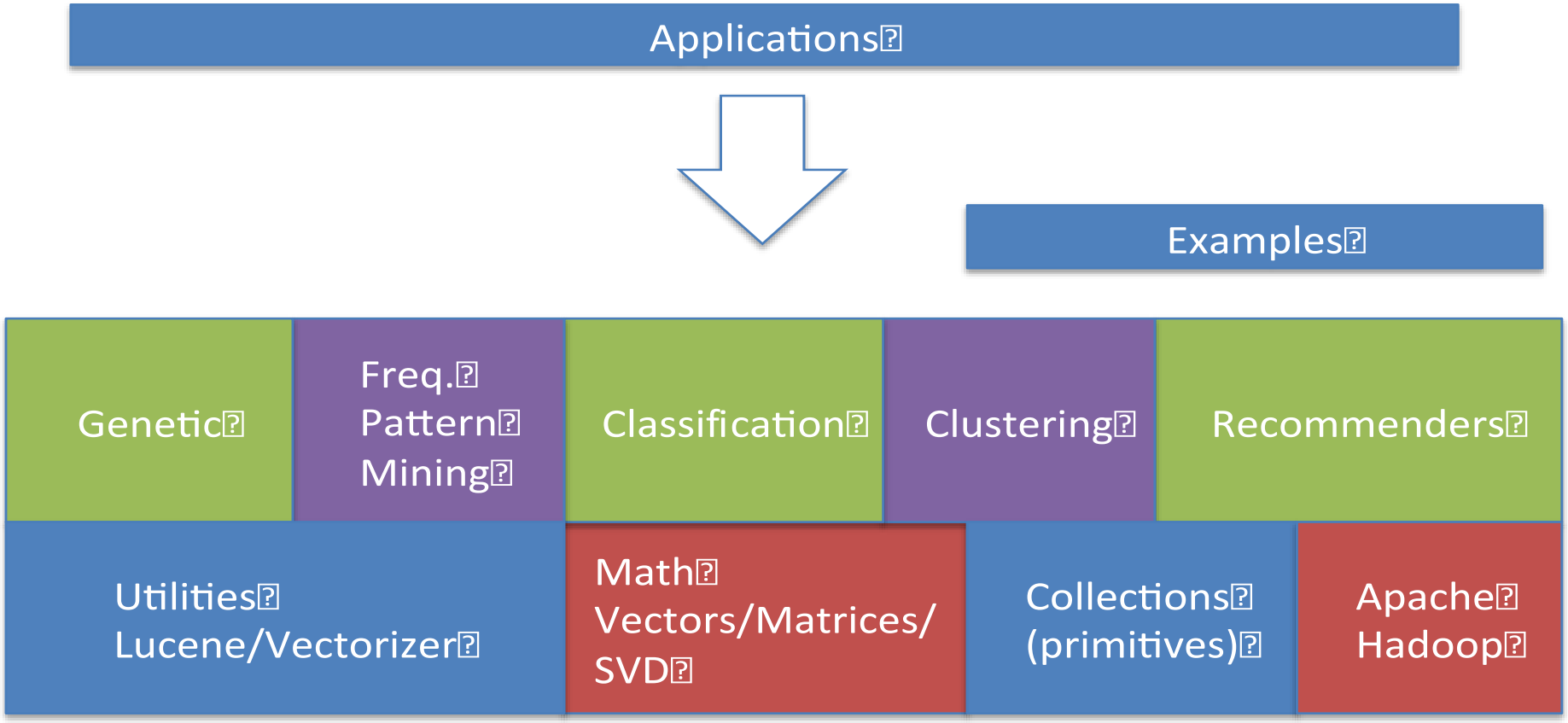
- **Hama**
  - Machine learning implemented on BSP

Apache Mahout

- Apache project
- Create scalable machine learning libraries
- **Why Mahout?** Many Open Source ML libraries either:
  - Lack Community
  - Lack Documentation and Examples
  - Lack Scalability
  - Or are research-oriented

Mahout Overview

- Built over existing production quality libraries


Applications
Examples
Genetic
Freq. Pattern Mining
Classification
Clustering
Recommenders
Utilities Lucene/Vectorizer
Math Vectors/Matrices/ SVD
Collections (primitives)
Apache Hadoop

Three major components of Mahout

1. An environment for building scalable algorithms
2. Many new Scala + Spark algorithms
3. Mature Hadoop MapReduce algorithms.

11 Apr 2015 - Apache Mahout next gen version 0.10.0 released

Mahout Samsara Environment

- Distributed Algebraic optimizer
- R-Like DSL Scala API
- Linear algebra operations
- Ops are extensions to Scala
- IScala REPL based interactive shell
- Integrates with compatible libraries like MLLib
- Interactive shell that runs distributed operations on a Spark cluster
- H2O in progress

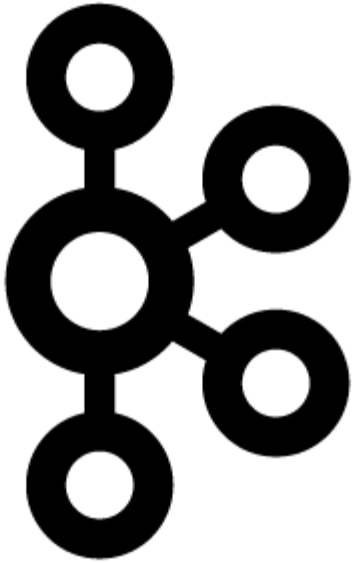
Mahout Samsara based Algorithms

- Stochastic Singular Value Decomposition (ssvd, dssvd)
- Stochastic Principal Component Analysis (spca, dspca)
- Distributed Cholesky QR (thinQR)
- Distributed regularized Alternating Least Squares (dals)
- Collaborative Filtering: Item and Row Similarity
- Naive Bayes Classification
- Distributed and in-core

MAHOUT 0.10.0 FEATURES BY ENGINE

http://mahout.apache.org/users/basics/algorithms.html

SPARK MLLIB VS. MAHOUT

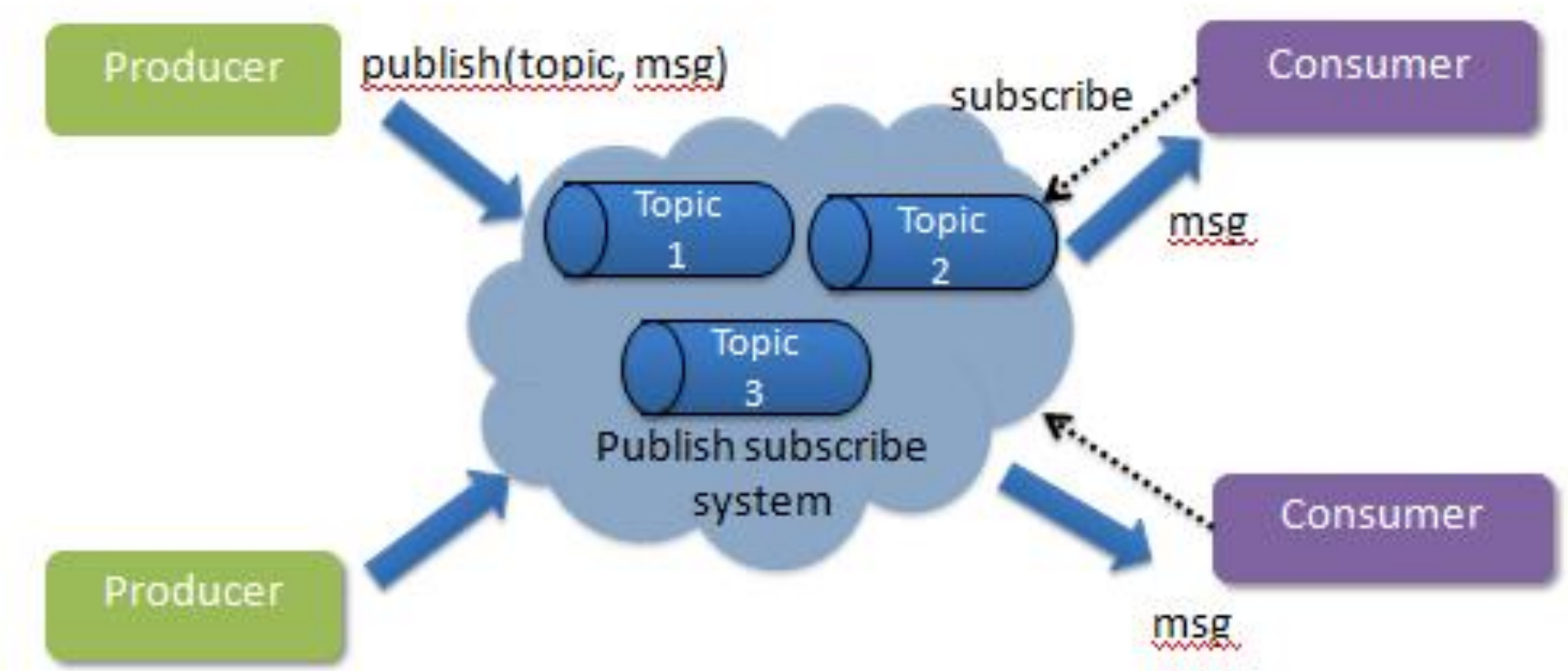


kafka

- **High throughput, distributed publish-subscribe based messaging system**
- **LinkedIn**
- **Scala**

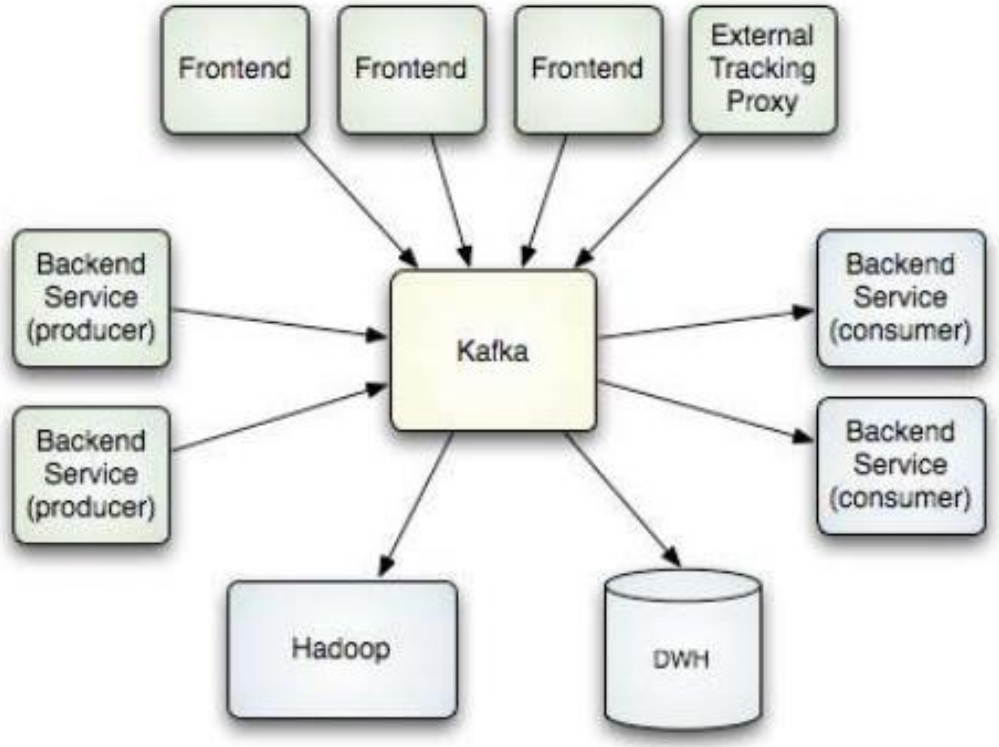
http://kafka.apache.org

A Publish-Subscribe System



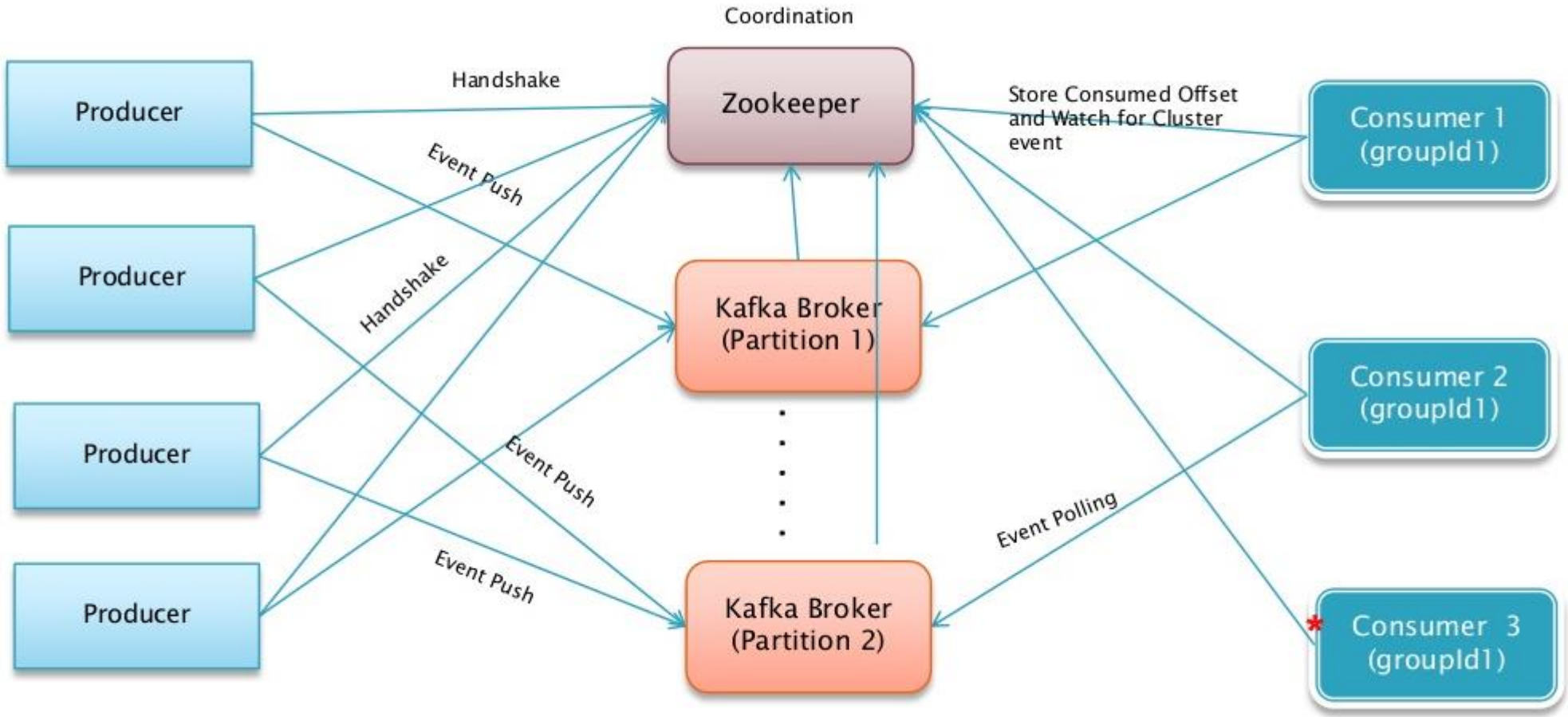
- **HornetQ**
- **ActiveMQ**
- **ZeroMQ**
- **RabbitMQ**

Kafka Design

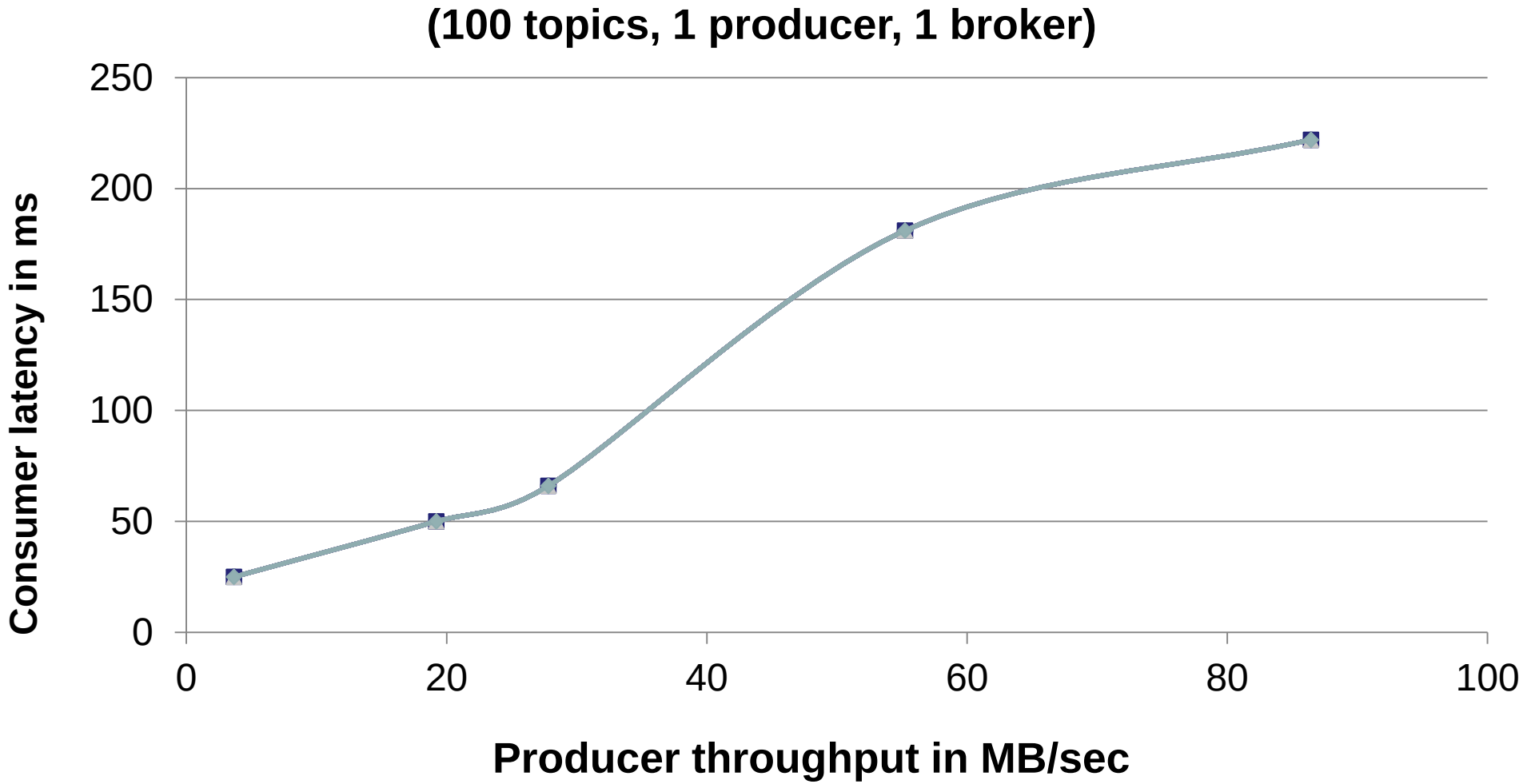


- Filesystem Cache
- Zero-copy transfer of messages
- Batching of Messages
- Batch Compression
- Automatic Producer Load balancing.
- Broker does not **Push** messages to Consumer, Consumer **Polls** messages from Broker.

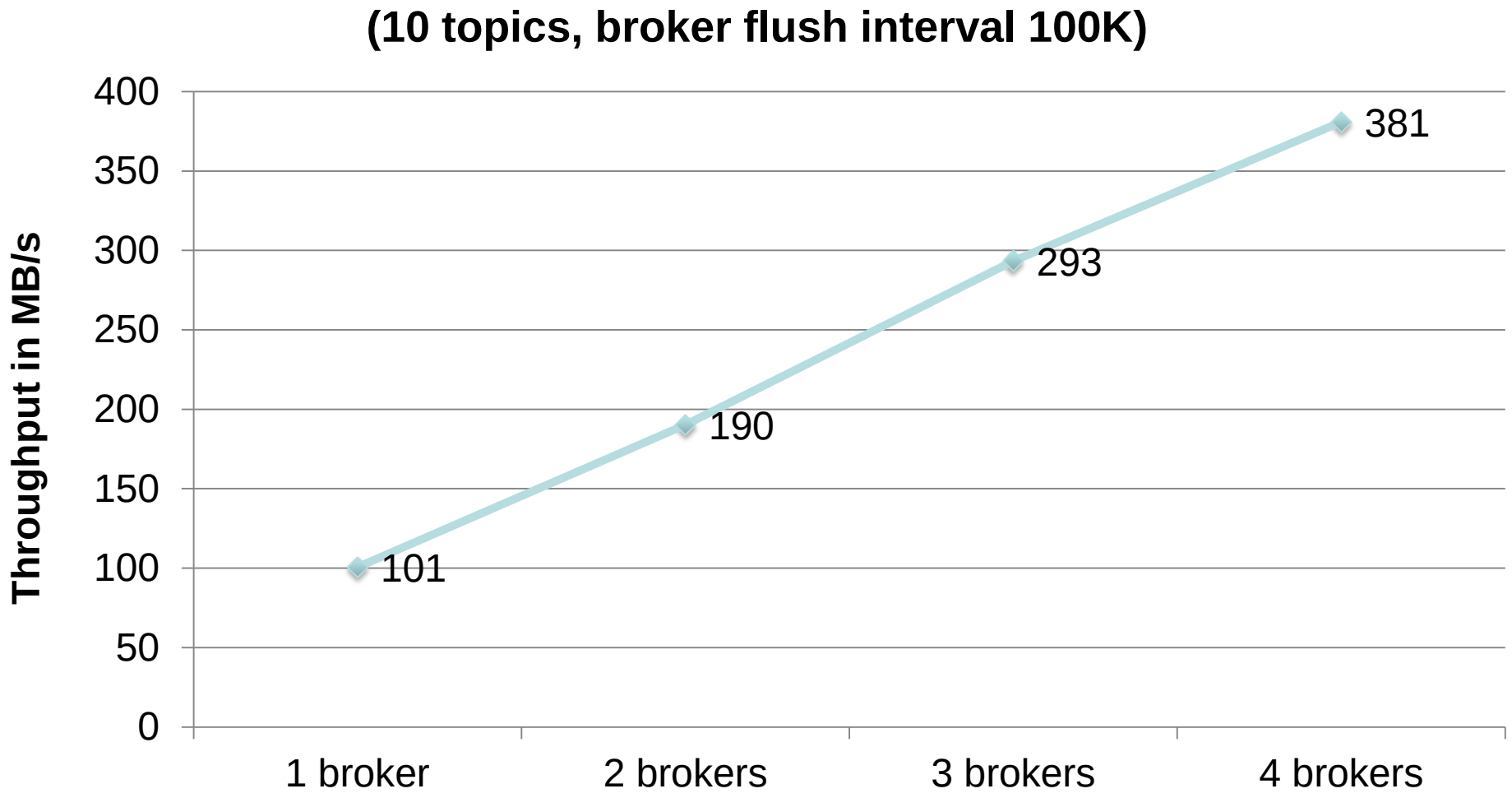
How Kafka Works



Latency vs throughput



Scalability



Apache ZooKeeper™

http://zookeeper.apache.org/

Coordination in a distributed system

- *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!

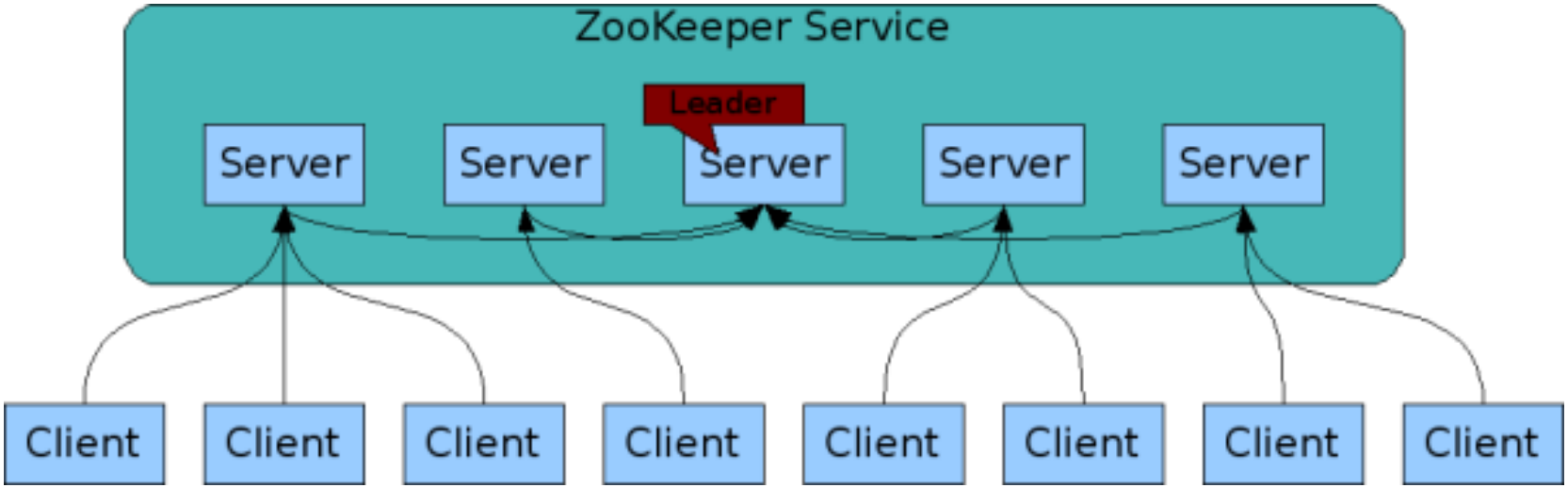

Copyright by Shamus O'Reilly via Flickr

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.

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

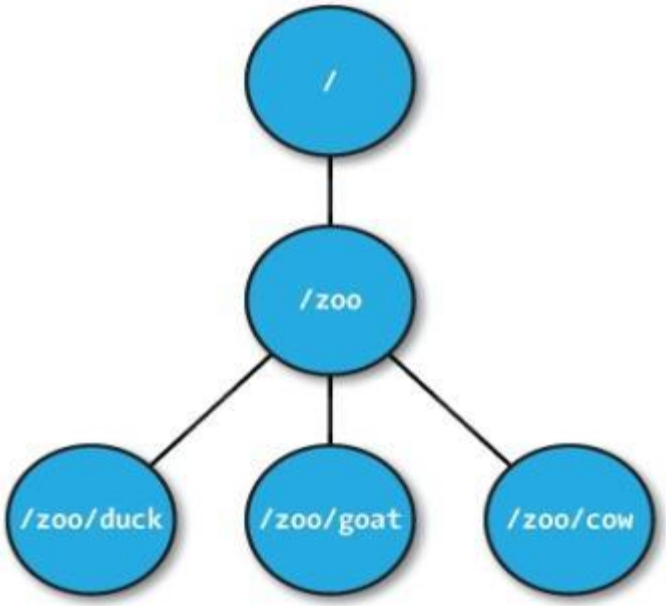
The ZooKeeper Data Model

- ZooKeeper has a hierarchal name space.
- Each node in the namespace is called as a *ZNode.*
- Every ZNode has data (given as byte[]) and can optionally have children.

```
parent : "foo"
   |-- child1 : "bar"
   |-- child2 : "spam"
   `-- child3 : "eggs"
         `-- grandchild1 : "42”
```

ZNodes

- Maintain a static structure with version numbers for data changes and timestamps.
- Version numbers increase with changes
- Data is read and written in its entirety



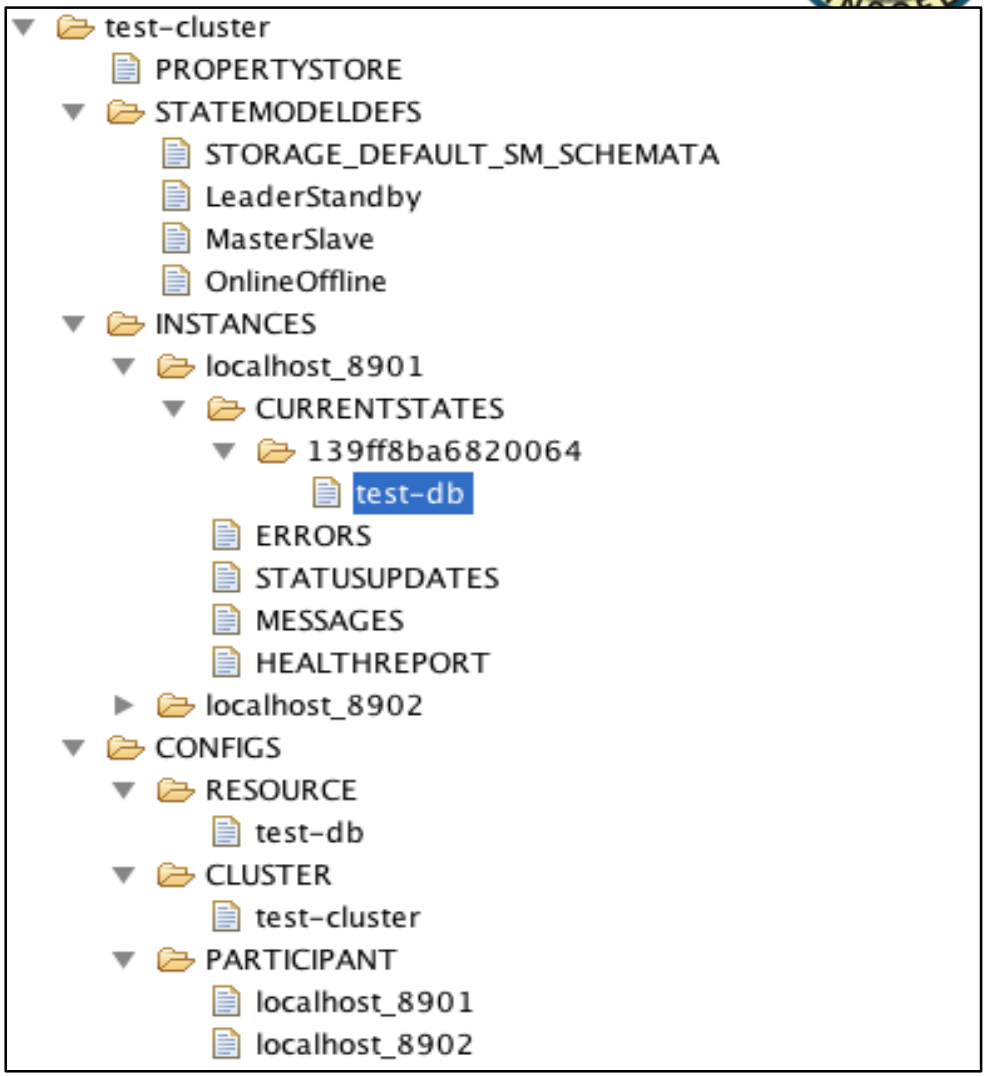
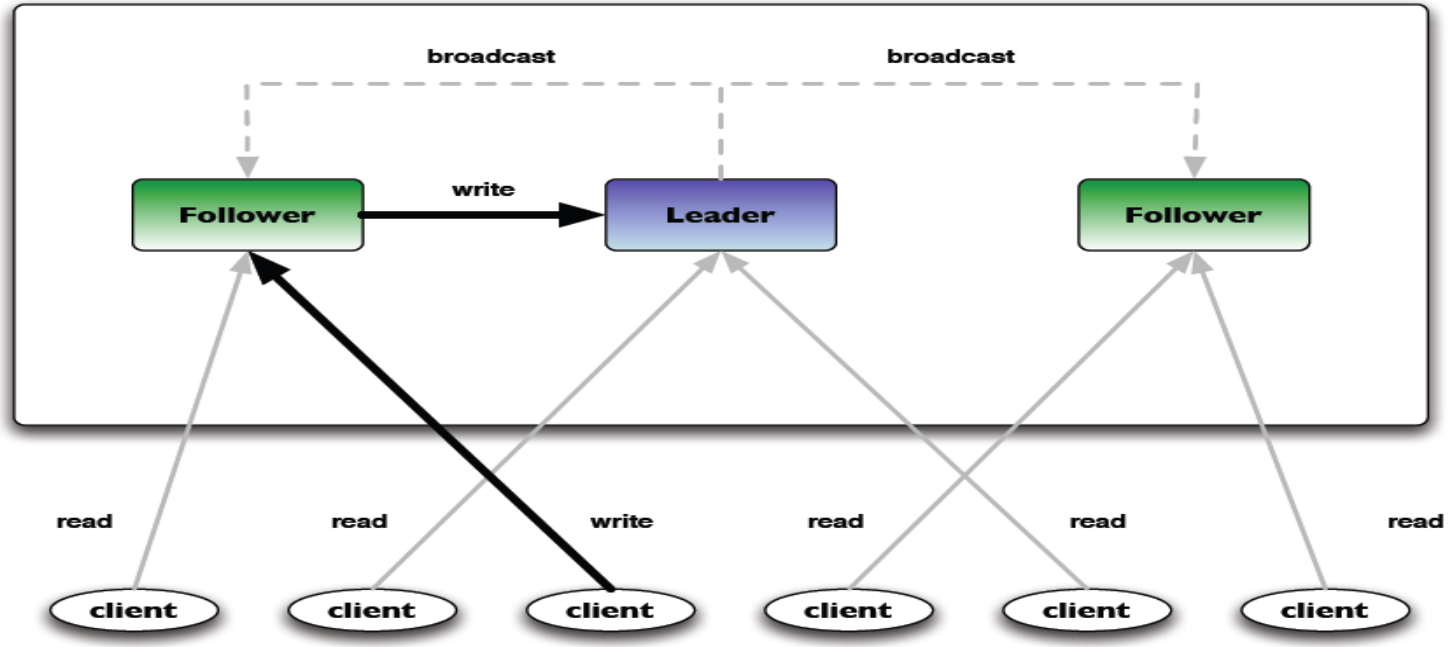


Image: http://helix.incubator.apache.org/Architecture.html

ZNode Reads & Writes



- Read requests are processed locally at the ZooKeeper server to which the client is currently connected
- Write requests are forwarded to the leader and go through majority consensus before a response is generated.

Image: http://www.slideshare.net/scottleber/apache-zookeeper

ZNode Operations

Operation	Type
create	Write
delete	Write
exists	Read
getChildren	Read
getData	Read
setData	Write
getACL	Read
setACL	Write
sync	Read

ZNodes are the main entity that a programmer access.

ZooKeeper Shell


```
[zk: localhost:2181(CONNECTED) 0] help
ZooKeeper -server host:port cmd args
        connect host:port
        get path [watch]
        ls path [watch]
        set path data [version]
        rmr path
        delquota [-n|-b] path
        quit
        printwatches on|off
        create [-s] [-e] path data acl
        stat path [watch]
        close
        ls2 path [watch]
        history
        listquota path
        setAcl path acl
        getAcl path
HelloWorld
        sync path
        redo cmdno
        addauth scheme auth
        delete path [version]
        setquota -n|-b val path
```

```
[zk: localhost:2181(CONNECTED) 1] ls /
[hbase, zookeeper]

        [zk: localhost:2181(CONNECTED) 2] ls2 /zookeeper
        [quota]
        cZxid = 0x0
ctime = Tue Jan 01 05:30:00 IST 2013
        mZxid = 0x0
mtime = Tue Jan 01 05:30:00 IST 2013
        pZxid = 0x0
        cversion = -1
        dataVersion = 0
aclVersion = 0
        ephemeralOwner = 0x0
dataLength = 0
        numChildren = 1

[zk: localhost:2181(CONNECTED) 3] create /test-znode

      Created /test-znode
      [zk: localhost:2181(CONNECTED) 4] ls /
    [test-znode, hbase, zookeeper]
[zk: localhost:2181(CONNECTED) 5] get /test-znode
    HelloWorld
```

DATA SECURITY & GOVERNANCE

Managing security on Hadoop

Authentication	Authorization	Audit	Data Protection
Who am I/prove it? Control access to cluster.	Restrict access to explicit data	Understand who did what	Encrypt data at rest & motion
Kerberos in native Apache Hadoop Perimeter Security with Apache Knox Gateway	Native in Apache Hadoop • MapReduce Access Control Lists • HDFS Permissions • Process Execution audit trail Cell level access control in Apache Accumulo Service level Authorization with Knox Access Audit with Knox		Wire encryption in native Apache Hadoop Wire Encryption with Knox Orchestrated encryption with 3rd party tools

Ranger Architecture

Security Administrators

Enterprise Legacy Systems

Define Policies
Audit user activity

Policy Integration

Enterprise Directory Services

Sync Users

Ranger User Sync Server

Sync Users

Ranger Policy Admin Server

Authenticate

Enterprise Users

Access Data in Hadoop Systems

HDFS — Ranger Plugin

Hive — Ranger Plugin

HBase — Ranger Plugin

Knox — Ranger Plugin

Storm — Ranger Plugin

Sync Policies

Ranger Policy DB

Ranger Audit Store

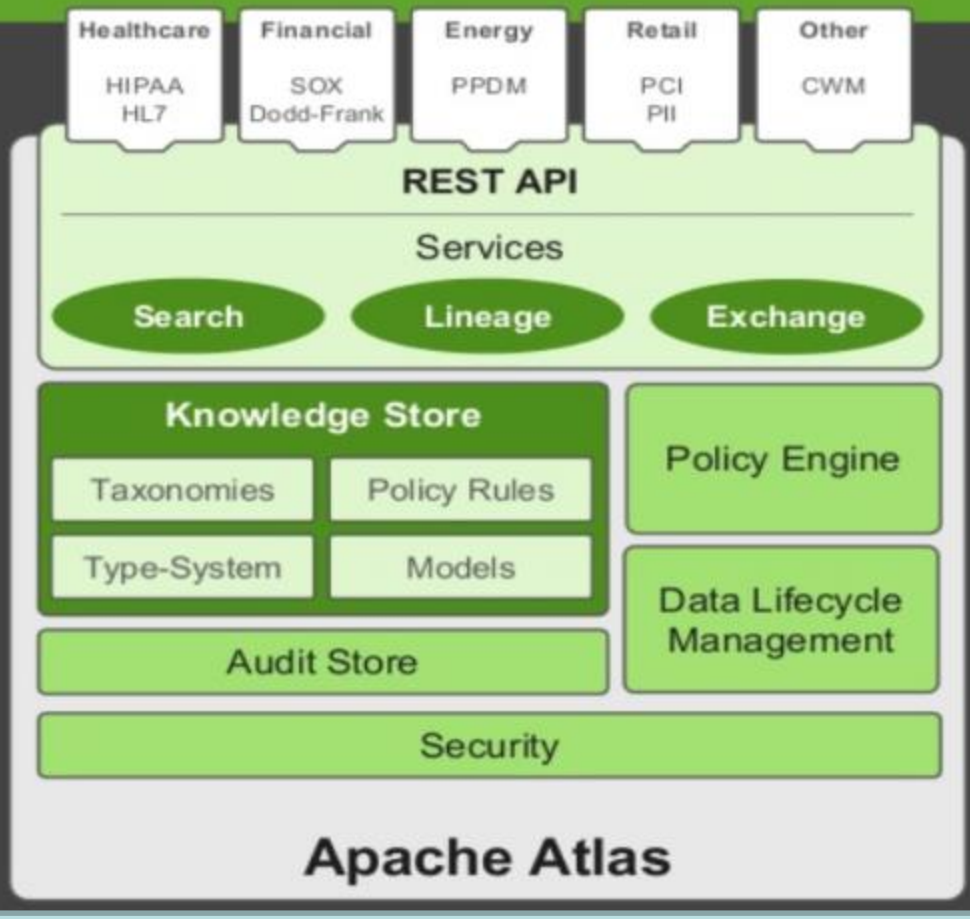
Store Audit Activity

Ranger Plugins Authorize the Access & Audit the Activity

Apache Atlas: Improved data governance for Hadoop

Atlas: Capabilities

- Data Classification
- Metadata Exchange
- Centralized Auditing
- Search & Lineage
- Policy Engine
- Security



Inspire...Educate...Transform.

International School of Engineering

Plot 63/A, 1st Floor, Road # 13, Film Nagar, Jubilee Hills, Hyderabad - 500 033

For Individuals: +91-9502334561/63 or 040-65743991

For Corporates: +91-9618483483

Web: http://www.insofe.edu.in

Facebook: https://www.facebook.com/insofe

Twitter: https://twitter.com/Insofeedu

YouTube: http://www.youtube.com/InsofeVideos

SlideShare: http://www.slideshare.net/INSOFE

LinkedIn: http://www.linkedin.com/company/international-school-of-engineering