HADOOP

AND OTHERS

INTRODUCTION TO DATA SCIENCE

TIM KRASKA

teaching
datascience
.org

LAST LECTURE

- Cloud Computing
- HDFS
- MapReduce

THIS LECTURE

- MapReduce ctd
- Other large scale processing frameworks
- Small scale processing frameworks
- (NO SQL)

CLICKER

Input to the ________ is the sorted output of the mappers.

a) Reducer
b) Mapper
c) Shuffle
d) All of the above

MAPREDUCE PROGRAMMING MODEL

Data type: key-value *records*

Map function:

$$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$

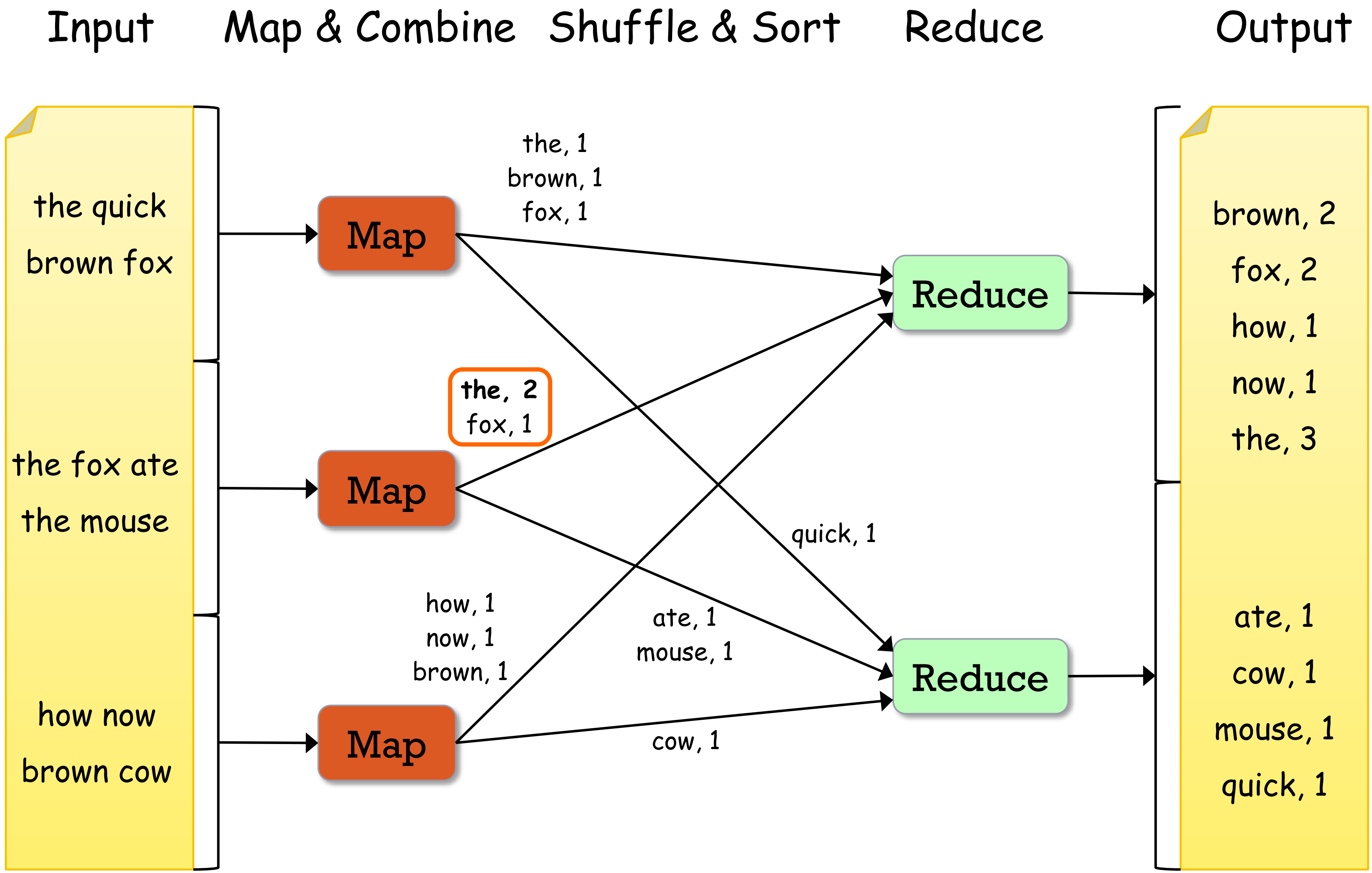
Reduce function:

$$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

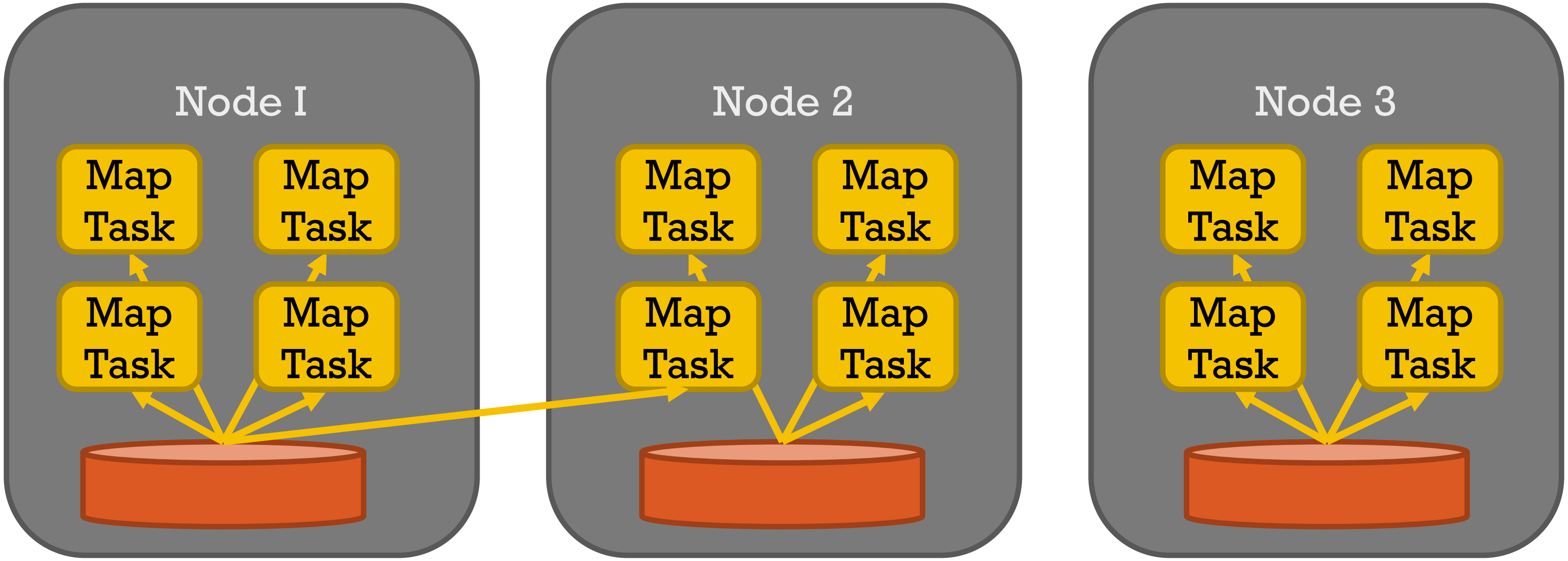
EXAMPLE: WORD COUNT

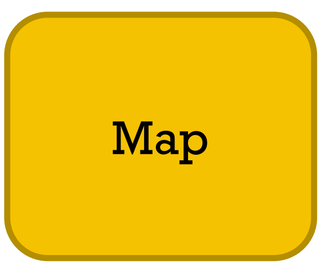
```
def mapper(line):
    foreach word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```

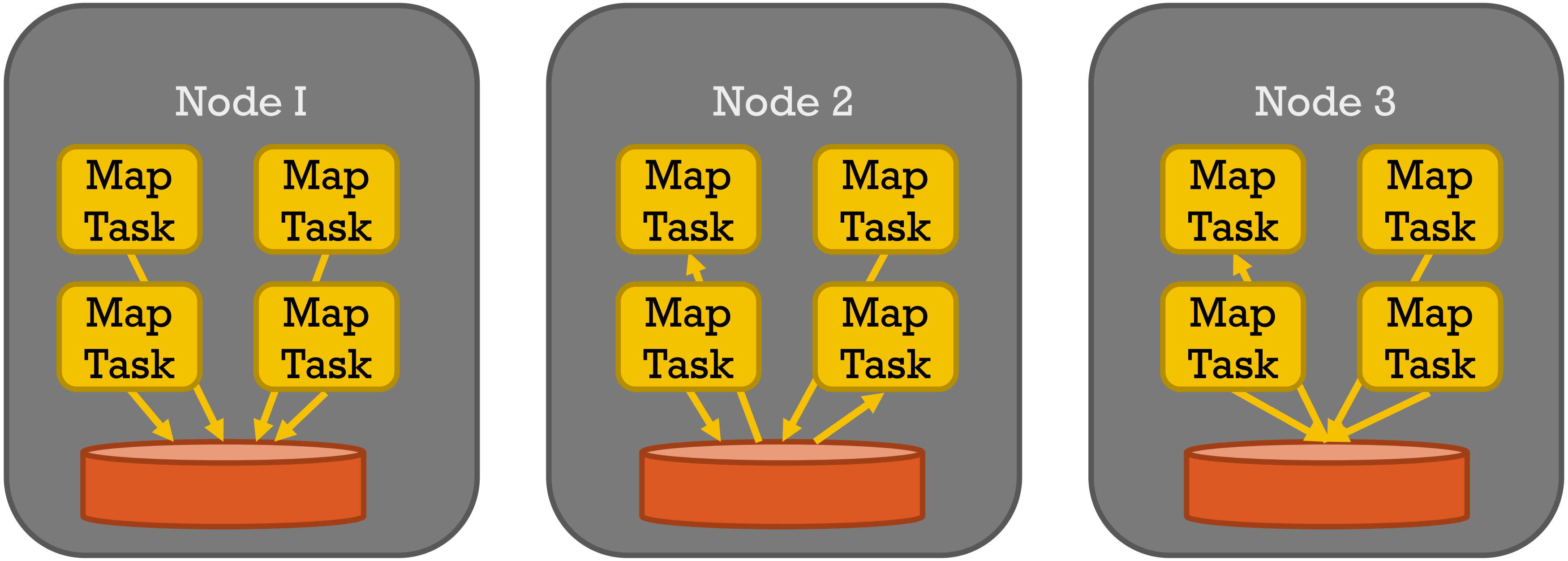
WORD COUNT WITH COMBINER

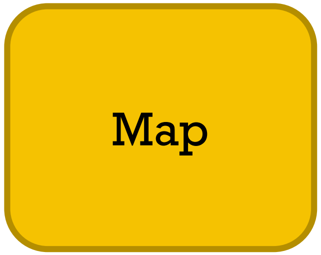


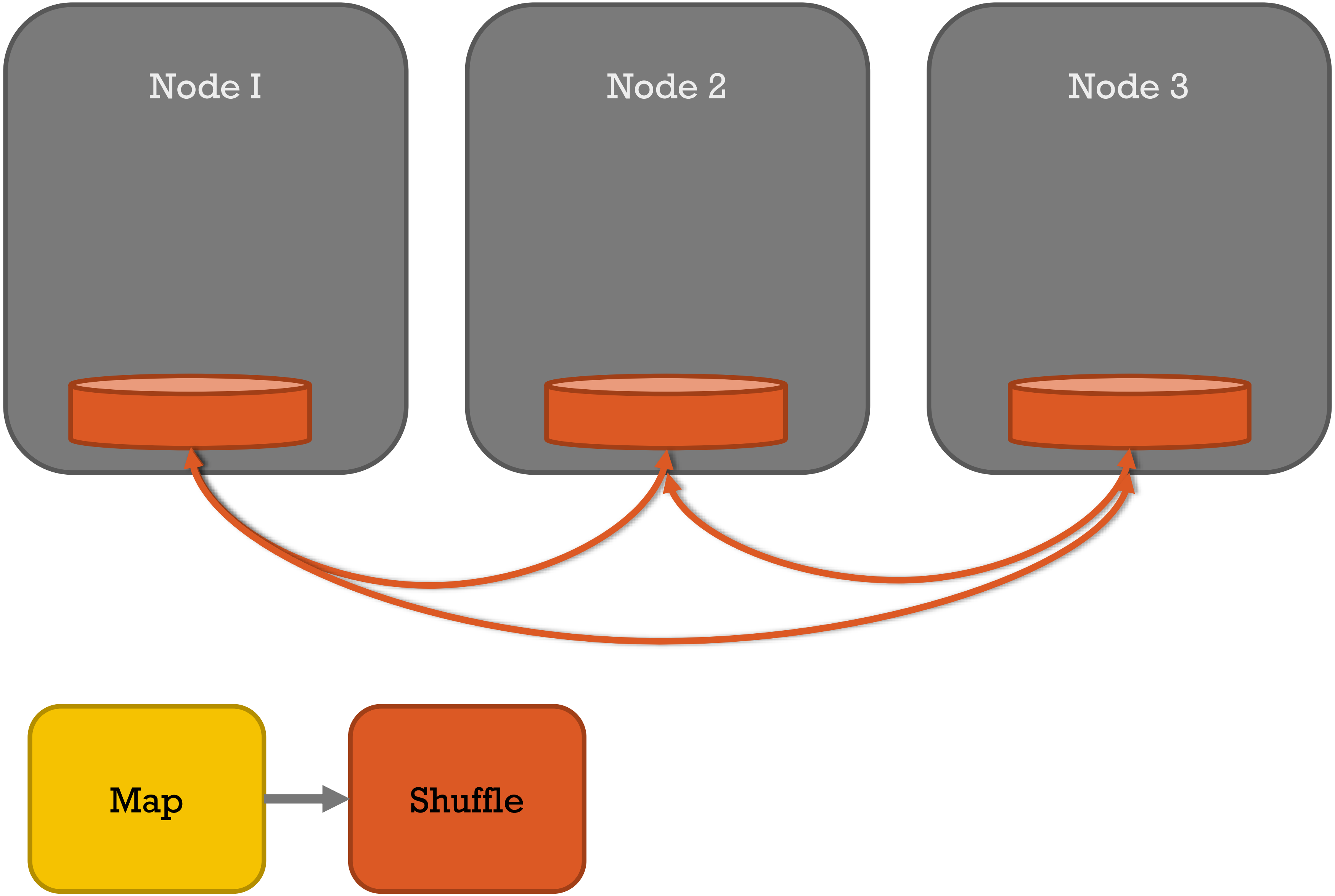

Node 1
Node 2
Node 3

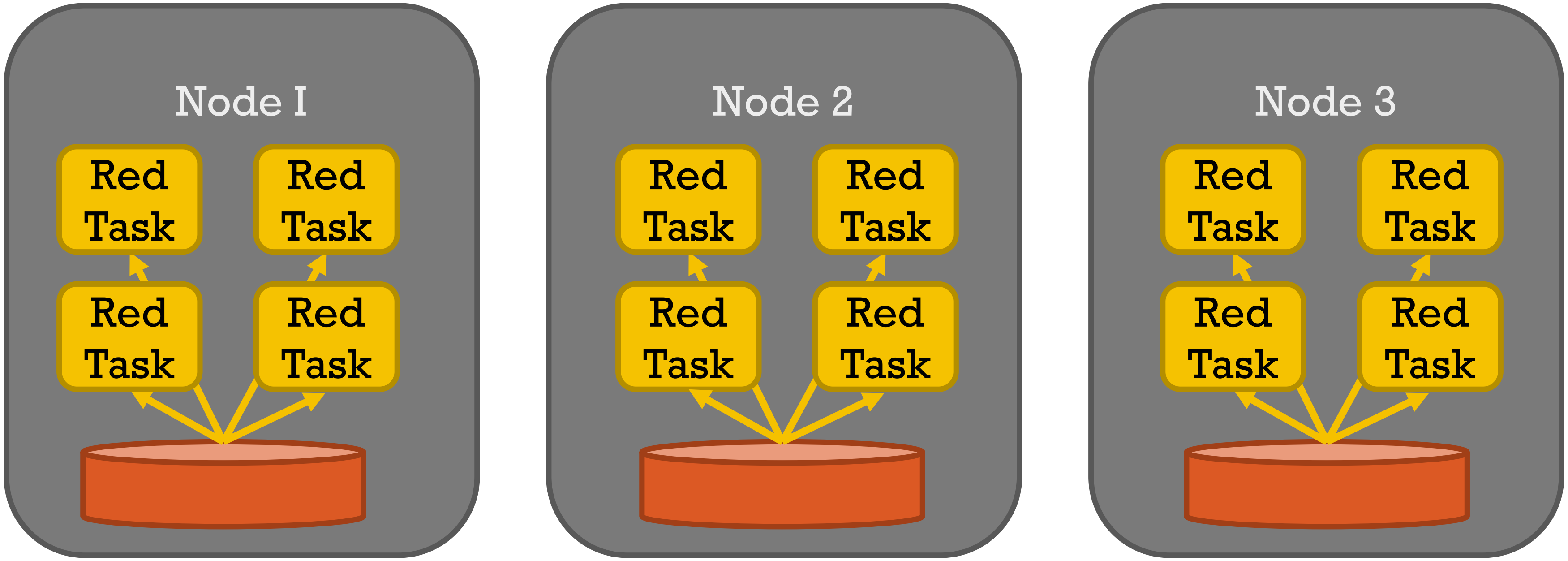

Node 1
Map Task
Map Task
Map Task
Map Task
Node 2
Map Task
Map Task
Map Task
Map Task
Node 3
Map Task
Map Task
Map Task
Map Task


Map


Node 1
Map Task
Map Task
Map Task
Map Task
Node 2
Map Task
Map Task
Map Task
Map Task
Node 3
Map Task
Map Task
Map Task
Map Task


Map


Node 1
Node 2
Node 3
Map
Shuffle


Node 1
Red Task
Red Task
Red Task
Red Task
Node 2
Red Task
Red Task
Red Task
Red Task
Node 3
Red Task
Red Task
Red Task
Red Task


Map
Shuffle
Reduce

OTHER MAP/REDUCE PARAMETERS

- **One or more Map tasks**
- **Zero or more Reduce tasks**
- **Zero or more Combiner tasks**
- **Shuffle / Partitioning function (distributed)**
- **Sort function (locally executed)**
- **Context for Map,Reduce Combiner**
- **Others (e.g., InputSplit)**
- **Configuration (more on that later)**

MAP/REDUCE PROS

Distribution is completely transparent

- Not a single line of distributed programming (ease, correctness)

Automatic fault-tolerance

- Determinism enables running failed tasks somewhere else again
- Saved intermediate data enables just re-running failed reducers

Automatic scaling

- As operations as side-effect free, they can be distributed to any number of machines dynamically

Automatic load-balancing

- Move tasks and speculatively execute duplicate copies of slow tasks (*stragglers)*

A FEW EXAMPLES

1. SEARCH

Input: (lineNumber, line) records

Output: lines matching a given pattern

```
Map:      if(line matches pattern):
              output(line)
```

Reduce: identity function

- Alternative: no reducer (map-only job)

2. SORT

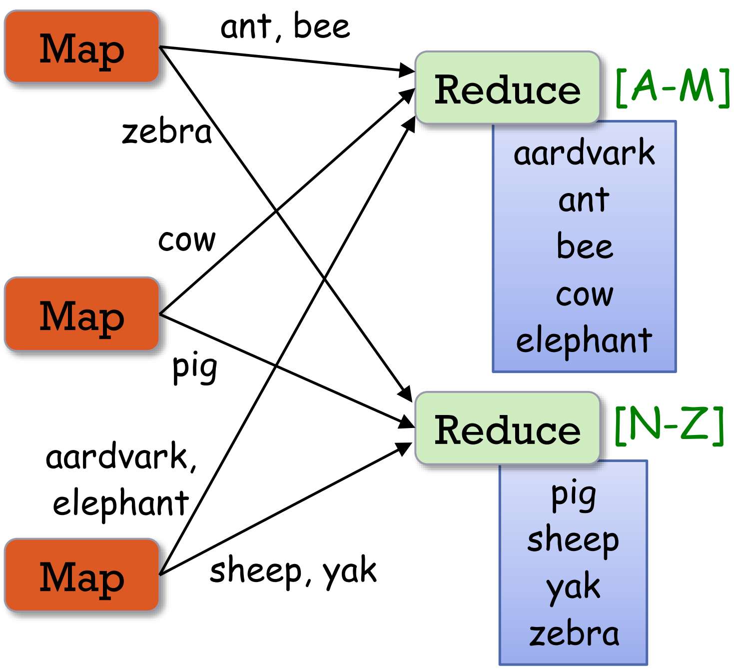
Input: (key, value) records

Output: same records, sorted by key

Map: identity function

Reduce: identity function

Trick: Pick partitioning function h such that $k_1 < k_2 \Rightarrow h(k_1) < h(k_2)$



CLICKER: INVERTED INDEX

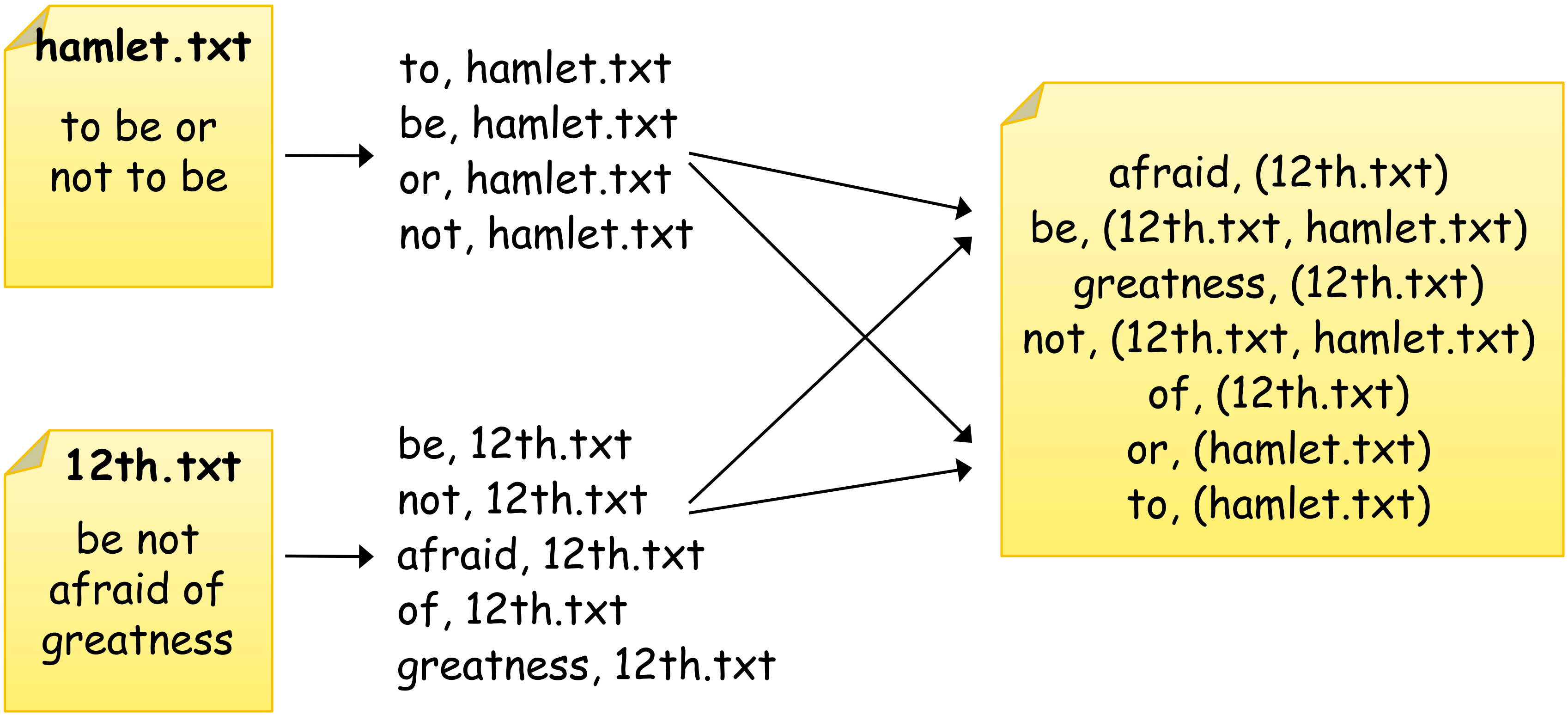
What MapReduce tasks do you need to build an inverted index

A) **def Map (filename, text):**
```
    foreach word in text.split(){
            output(word, filename)}
```
def Reduce(word, list(filename)):
```
    output(word, sort(filenames))
```

B) **def Map (filename, text):**
```
    foreach word in text.split(){
            output(word, filename)}
```
def Combine(word, filenames):
```
    output(word, set(filenames))
```
def Reduce(word, filenames):
```
    output(word, sort(filenames))
```

C) **var globalHashMap = new HashMap on master-node**
def Map (filename, text):
```
    foreach word in text.split(){
            output(word, filename)}
```
def Reduce(word, filenames):
```
    globalHashMap.add(word, sort(filenames)
```

INVERTED INDEX EXAMPLE



3. MOST POPULAR WORDS

Input: (filename, text) records

Output: top 100 words occurring in the most files drop rare words

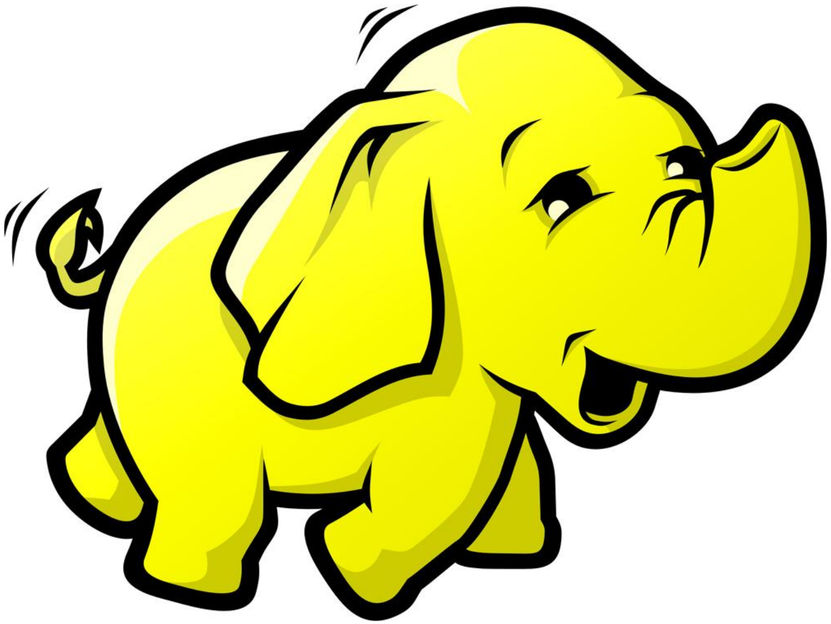
Two-stage solution:

- **Job 1:**
  - Create inverted index, giving (word, list(file)) records
  - Important: do not remove duplicates
- **Job 2:**
  - Map each (word, list(file)) to (count, word)
  - Sort these records by count as in sort job

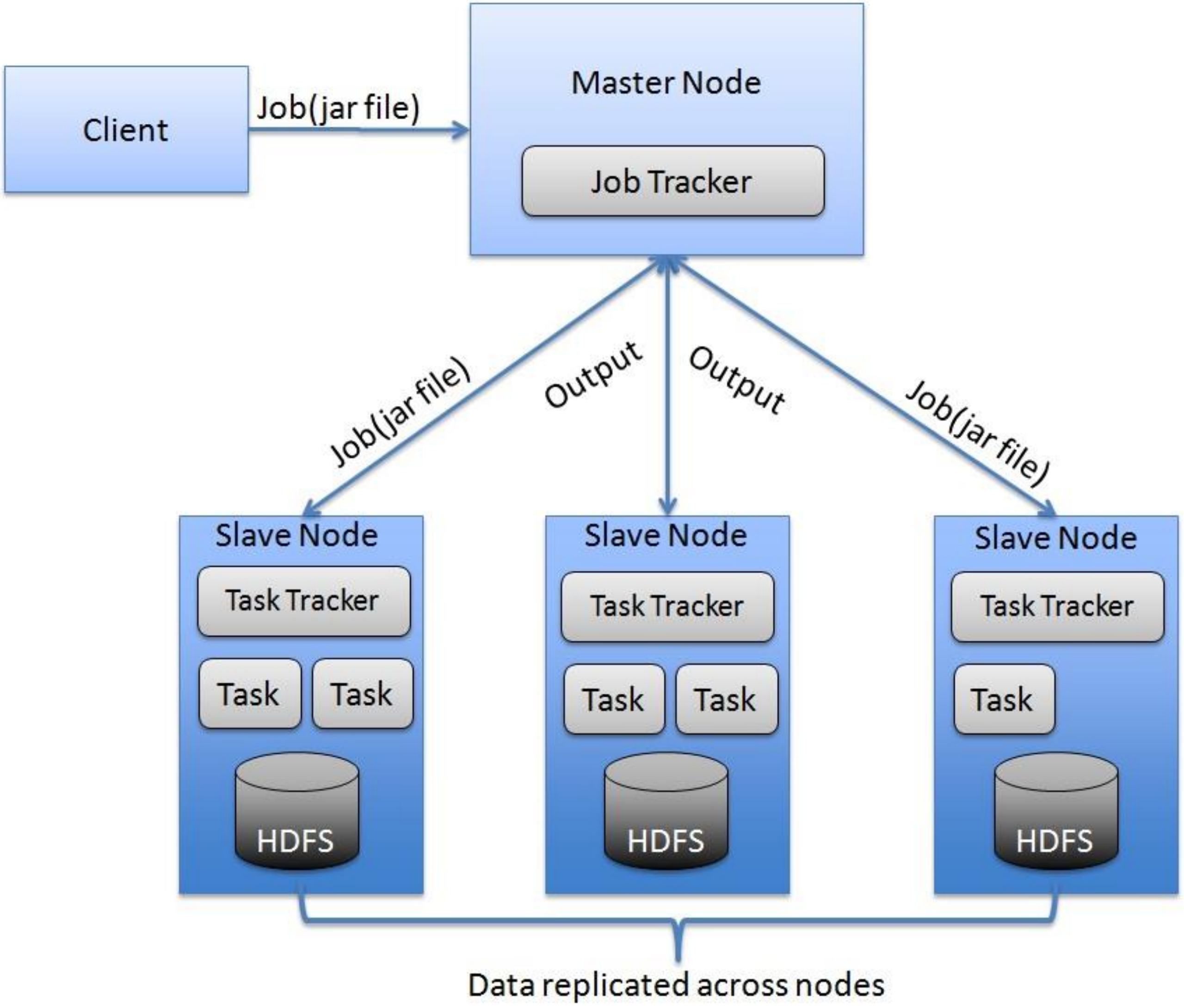
Optimizations:

- Map to (word, 1) instead of (word, file) in Job 1
- Count files in job 1's reducer rather than job 2's mapper
- Estimate count distribution in advance and drop rare words

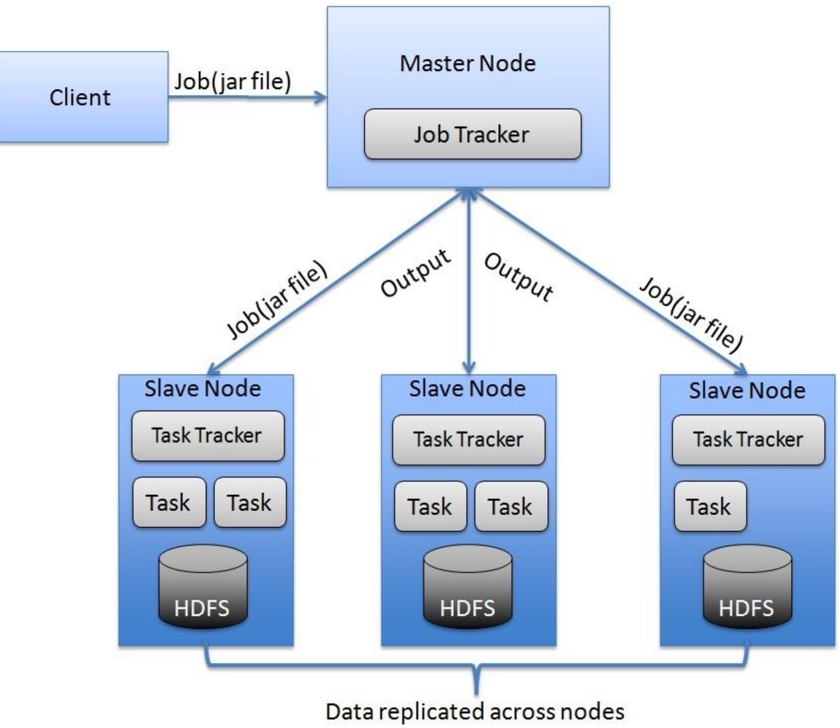
HADOOP: THE 1ST OPEN-SOURCE SYSTEM IMPLEMENTING THE MAPREDUCE PARADIGM

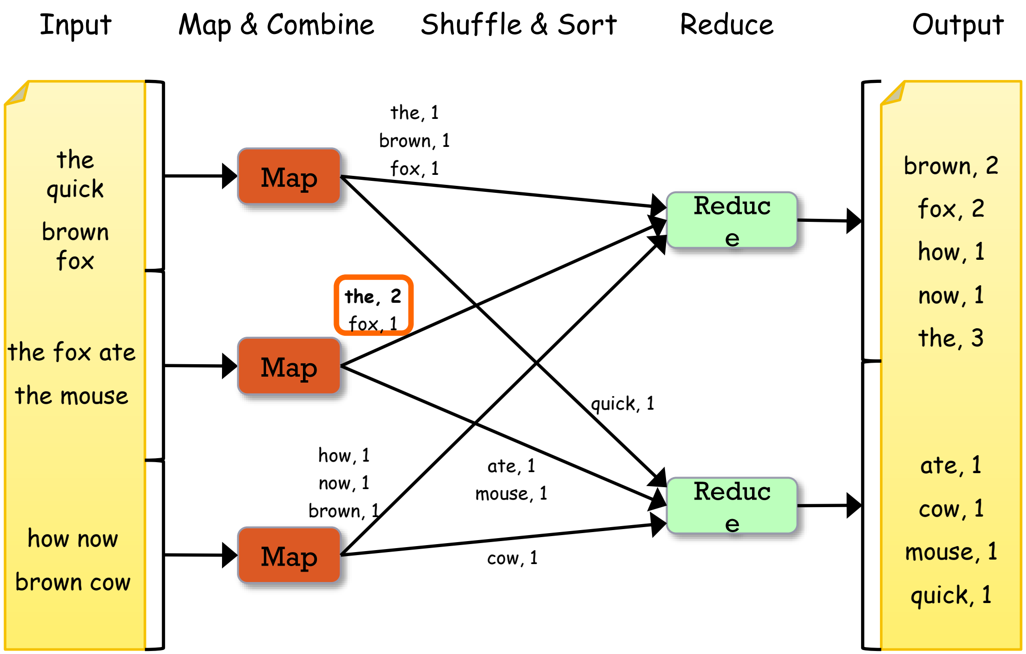


HADOOP ARCHITECTURE



HADOOP ARCHITECTURE





CLICKER

For a very simple word count application on a cluster with 1000 nodes, each having two CPUs, 10 cores each, how many parallel **MAP tasks** (i.e., threads) per node should you use?

a) 20

b) 2 * 20

c) 60 – 100

d) More than 100

CLICKER

For a very simple word count application on a cluster with 1000 nodes, each having two CPUs, 10 cores each, how many parallel **REDUCE tasks** per node should you use?

a) The same number as map tasks

b) 2 * 1.75 * 20 per node

c) 2 * 0.95 * 20 per node

d) Needs to be fine tuned so that the output is a multiple of a block size

e) Needs to be fined tuned so that a reduce task takes between 5 and 10 minutes

Iterative Algorithms in MapReduce Example KMeans

```
Select K random data points {s1, s2,… sK} as centroids cj.
Until clustering converges or other stopping criterion{
  For each data point xi:
     Assign xi to the closes centroid such that
     dist(xi, cj) is minimal.
  For each cluster cj, update the centroids
     cj = μ(cj)
}
```

How do you express K-Means in the Map/Reduce paradigm?

Iterative Algorithms in MapReduce Example KMeans

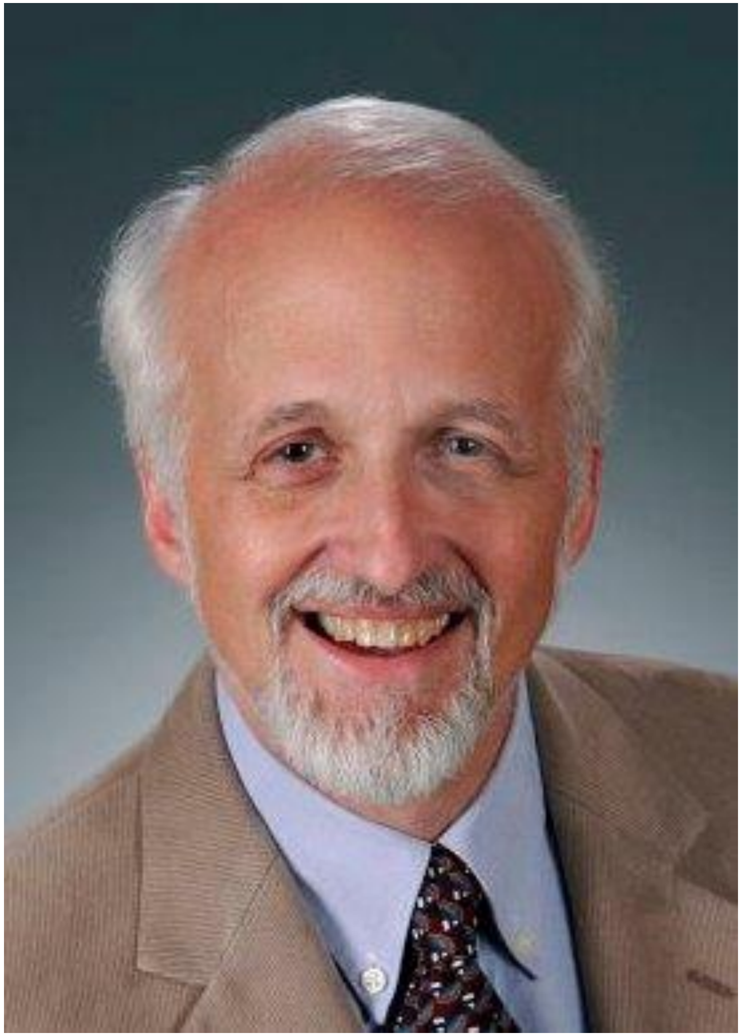
Map1(filename, data) := emit data as (r-id, features)

centroids[] = read-centroids from disk
Configure map2 job with centroids[]
Map2(r-id, features) :=
compare features (i.e., coordinates) with centroids
return (Closest-Centroid-ID, features)

Reduce(Centroid-ID, List[features]) :=
average features (i.e., coordinates) and emit (Centroid-ID, New-Coordinates)

Write new centroids to disk
Check if converged, if not do **Map2** and **Reduce** again

WHAT DO YOU THINK WERE THE REACTION OF THE DATABASE COMMUNITY?




I JUST
DON'T
CARE

MR VS. DATABASES

HADOOP VS. RDBMS

Pavlo et al. 2009

Comparison of 3 systems

- Hadoop
- Vertica (a column-oriented database)
- DBMS-X (a row-oriented database)
  - rhymes with "schmoracle"

Qualitative

- Programming model, ease of setup, features, etc.

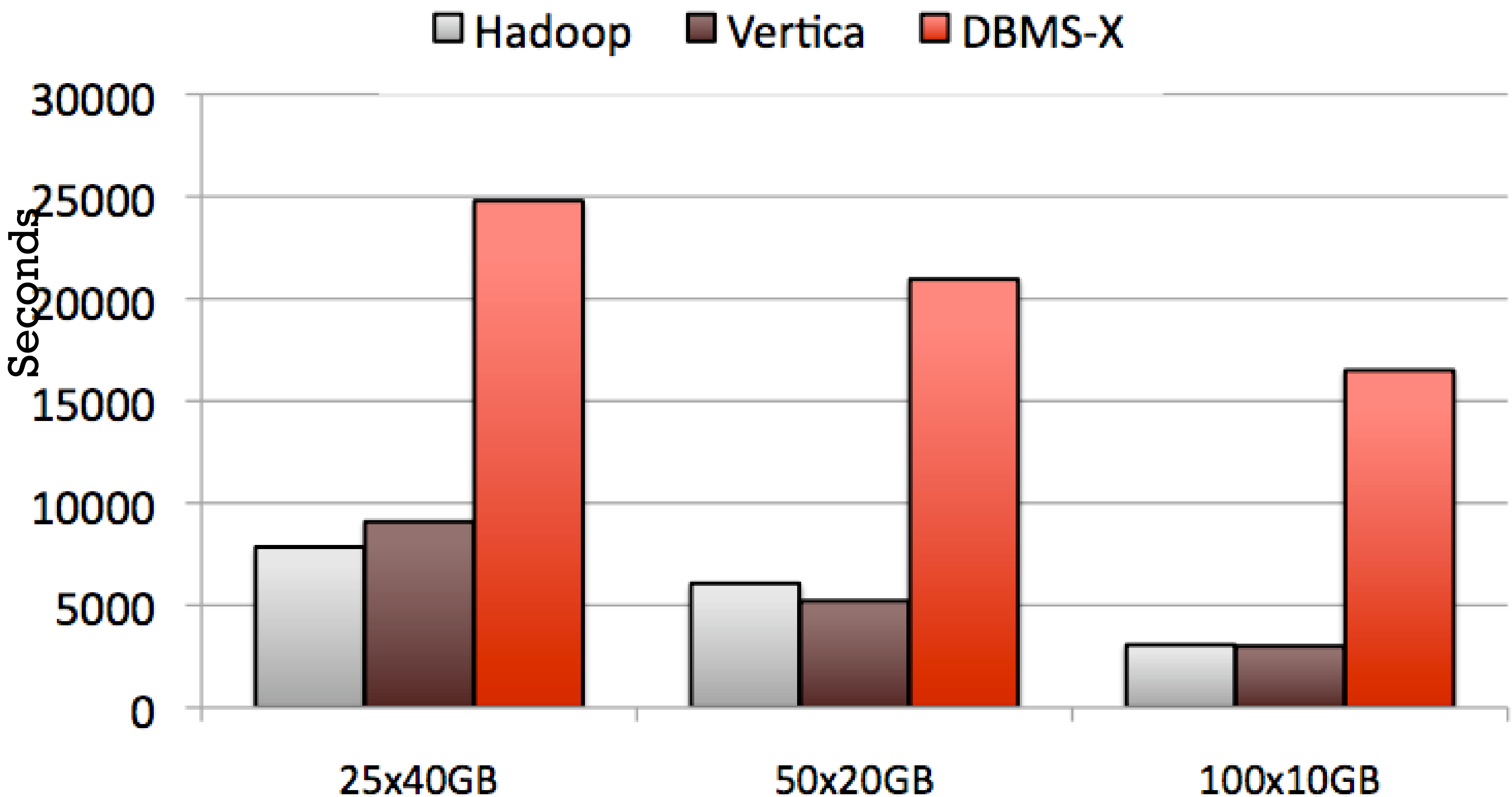
Quantitative

- Data loading, different types of queries

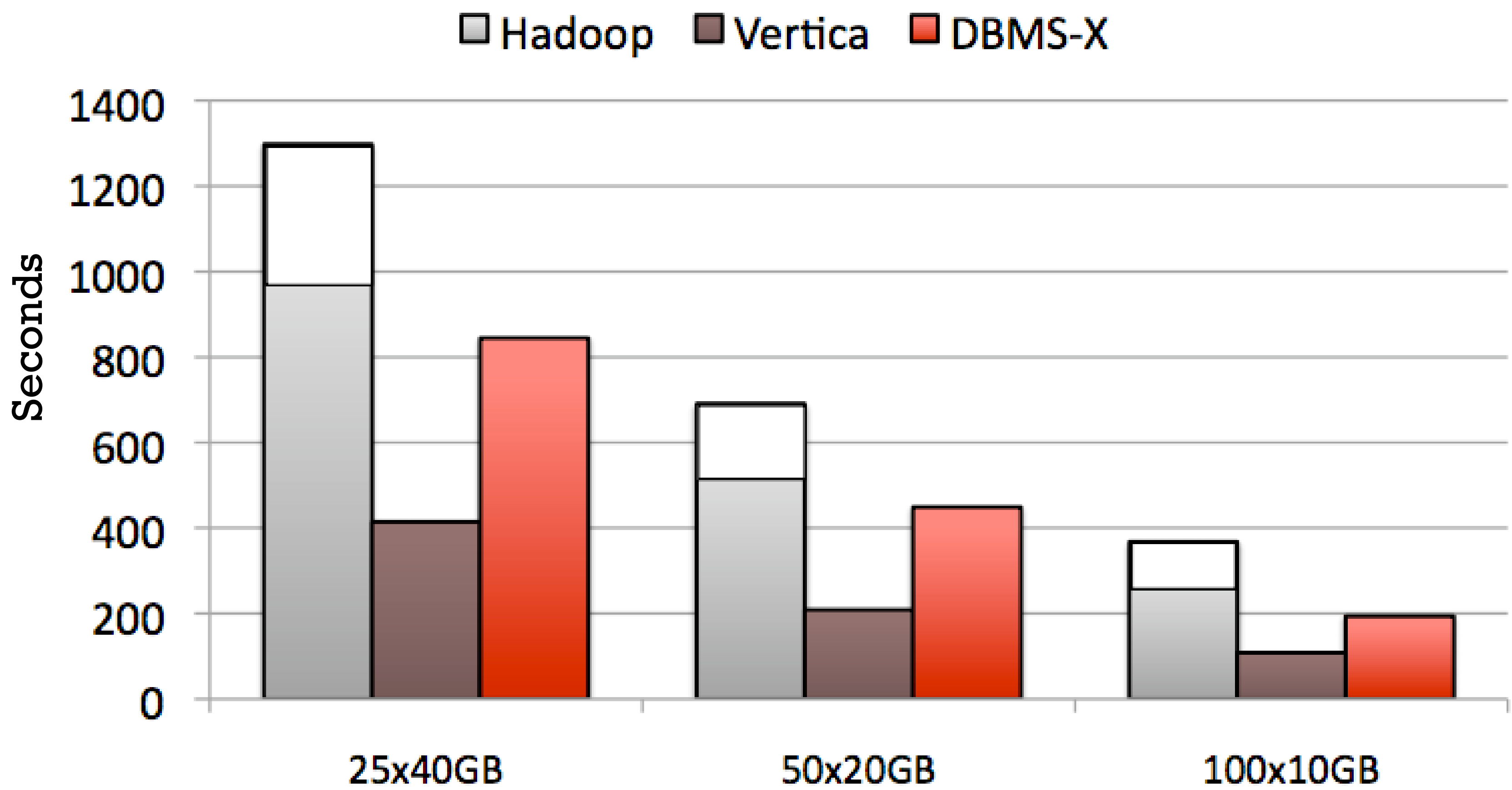
Grep Task

- Find 3-byte pattern in 100-byte record
  - *1 match per 10,000 records*
- Data set:
  - *10-byte unique key, 90-byte value*
  - *1TB spread across 25, 50, or 100 nodes*
  - *10 billion records*
- Original MR Paper (Dean et al. 2004)

Grep Task Loading Results

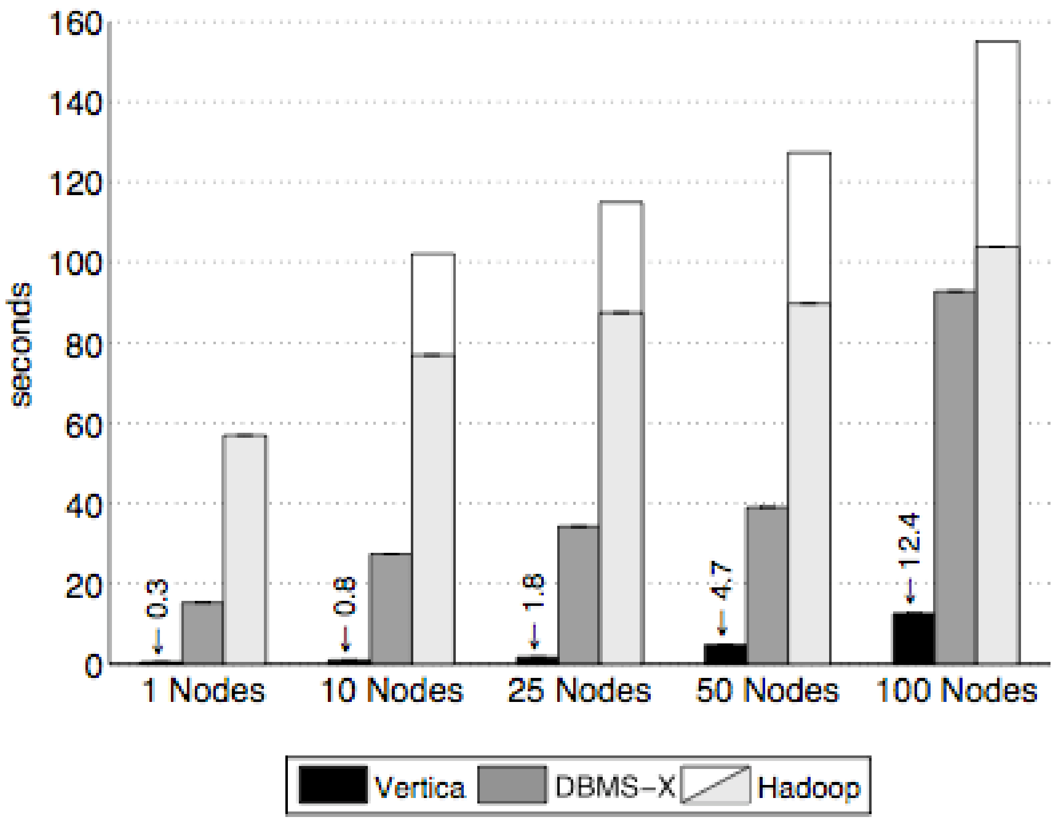


Grep Task Execution Results



SELECTION TASK

```
SELECT pageURL, pageRank
FROM Rankings WHERE pageRank > X
```



1 GB / node

Analytical Tasks

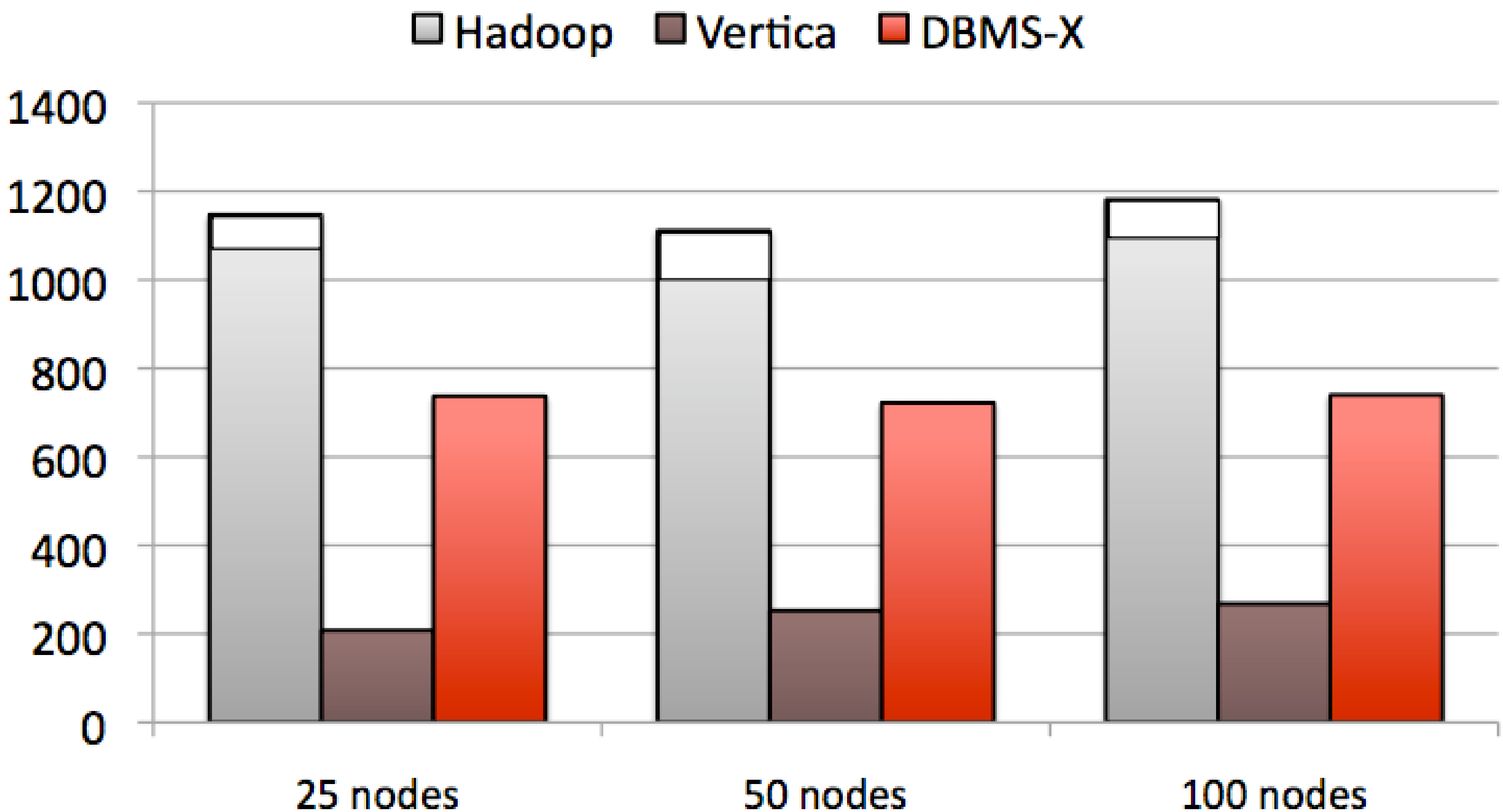
- Simple web processing schema
- Data set:
  - *600k HTML Documents (6GB/node)*
  - *155 million UserVisit records (20GB/node)*
  - *18 million Rankings records (1GB/node)*

Aggregate Task

- Simple query to find adRevenue by IP prefix

```
SELECT SUBSTR(sourceIP, 1, 7),
       SUM(adRevenue)
  FROM userVistits
 GROUP BY SUBSTR(sourceIP, 1, 7)
```

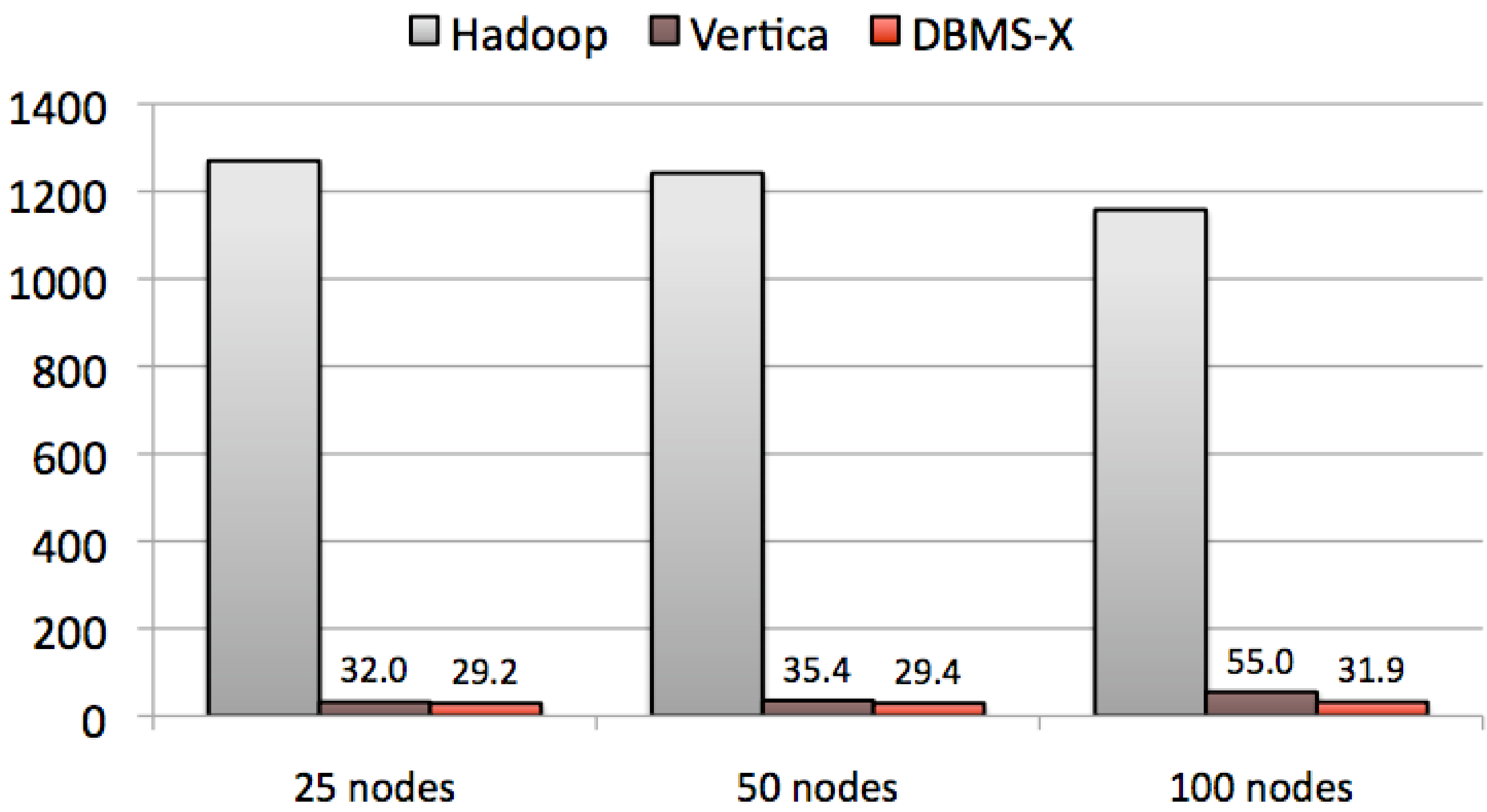
Aggregate Task Results



Join Task

- **Find the sourceIP that generated the most adRevenue along with its average pageRank.**
- **Implementations:**
  - ***DBMSs – Complex SQL using temporary table.***
  - ***MapReduce – Three separate MR programs.***

Join Task Results



PROBLEMS WITH THIS ANALYSIS?

Other ways to avoid sequential scans?

Fault-tolerance in large clusters?

Tasks that cannot be expressed as queries?

Google's Response: Cluster Size

- Largest known database installations:
  - *Greenplum – 96 nodes – 4.5 PB (eBay) [1]*
  - *Teradata – 72 nodes – 2+ PB (eBay) [1]*

- Largest known MR installations:
  - *Hadoop – 3658 nodes – 1 PB (Yahoo) [2]*
  - *Hive – 600+ nodes – 2.5 PB (Facebook) [3]*

[1] **eBay's two enormous data warehouses** – April 30th, 2009
http://www.dbms2.com/2009/04/30/ebays-two-enormous-data-warehouses/

[2] **Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds** – May 11th, 2009
http://developer.yahoo.net/blogs/hadoop/2009/05/hadoop_sorts_a_petabyte_in_162.html

[3] **Hive - A Petabyte Scale Data Warehouse using Hadoop** – June 10th, 2009
http://www.facebook.com/note.php?note_id=89508453919

Concluding Remarks

- What can *MapReduce* learn from *Databases*?
  - *Declarative languages are a good thing.*
  - *Schemas are important.*

- What can *Databases* learn from *MapReduce*?
  - *Query fault-tolerance.*
  - *Support for in situ data.*
  - *Embrace open-source.*

APACHE PIG

High-level language:

- Expresses sequences of MapReduce jobs
- Provides relational (SQL) operators
  (JOIN, GROUP BY, etc)
- Easy to plug in Java functions

Started at Yahoo! Research

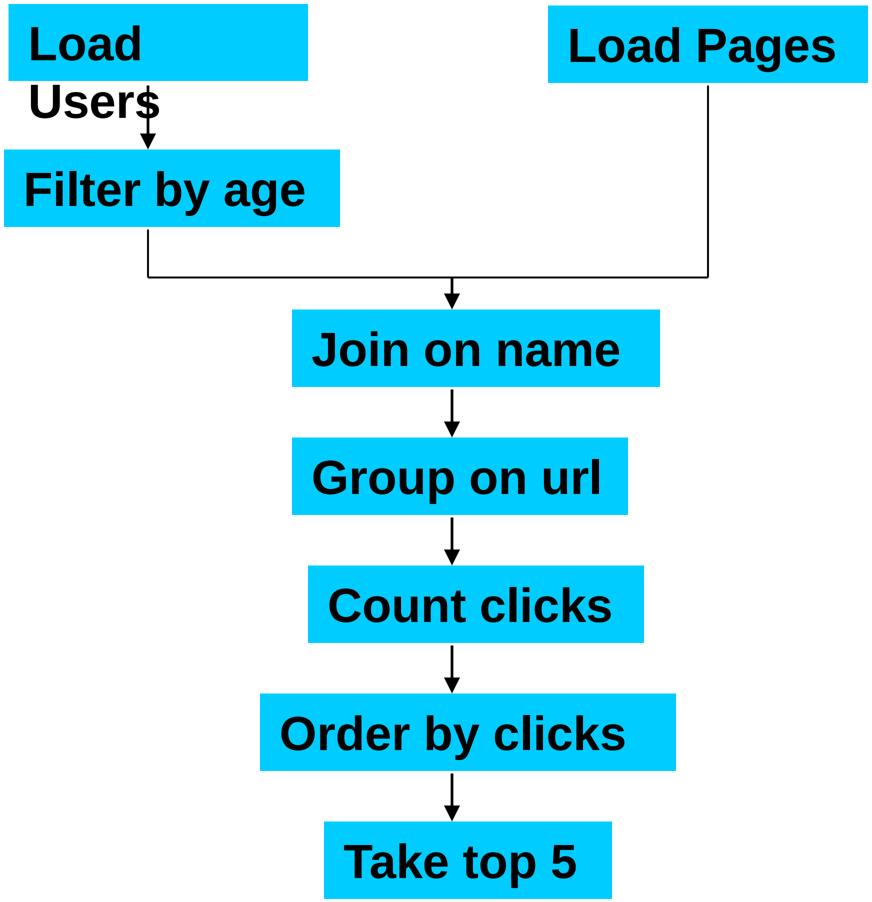
- Runs about 50% of Yahoo!'s jobs

https://pig.apache.org/

Similar to Google's (internal) Sawzall project

EXAMPLE PROBLEM

Given *user data* in one file, and *website data* in another, find the *top 5 most visited pages by users aged 18-25*



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

IN MAPREDUCE

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

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.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,
                OutputCollector<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 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.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();
    }
}
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

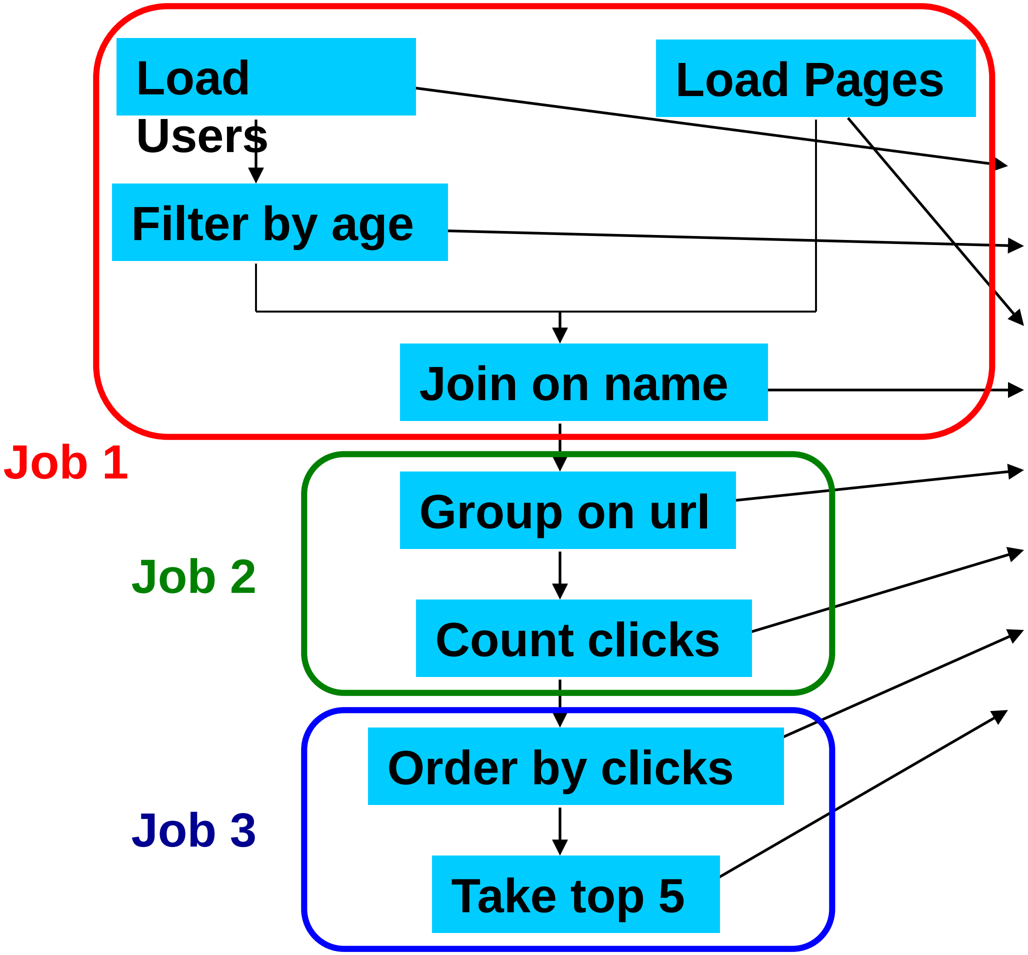
IN PIG LATIN

```
Users    = load 'users' as (name, age);
Filtered = filter Users by
                 age >= 18 and age <= 25;
Pages    = load 'pages' as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                  count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into 'top5sites';
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

TRANSLATION TO MAPREDUCE

Notice how naturally the components of the job translate into Pig Latin




```
Users = load …
Filtered = filter …
Pages = load …
Joined = join …
Grouped = group …
Summed = … count()…
Sorted = order …
Top5 = limit …
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

APACHE HIVE

Relational database built on Hadoop

- Maintains table schemas
- SQL-like query language (which can also call Hadoop Streaming scripts)
- Supports table partitioning, complex data types, sampling, some query optimization

Developed at Facebook

- Used for many Facebook jobs

Now used by many others

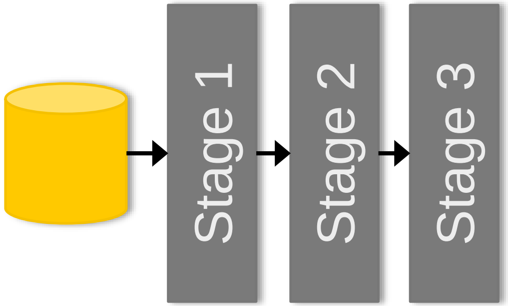
- Netfix, Amazon, …

http://hive.apache.org/

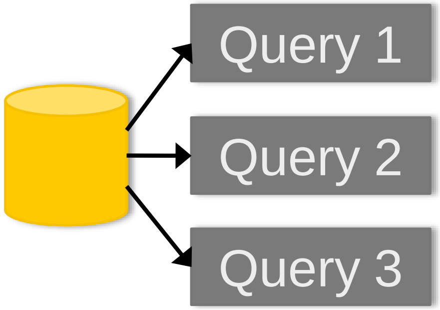
APACHE SPARK MOTIVATION

Complex jobs, interactive queries and online processing all need one thing that MR lacks:

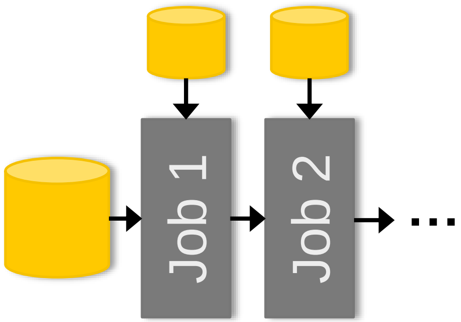
Efficient primitives for data sharing



Iterative job



Interactive mining

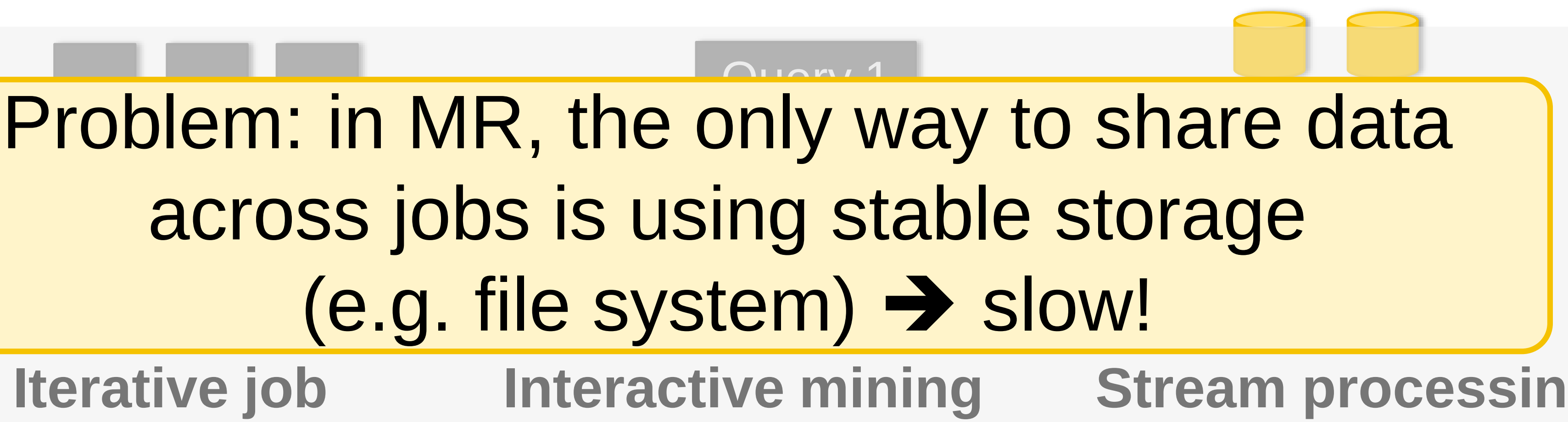


Stream processing

SPARK MOTIVATION

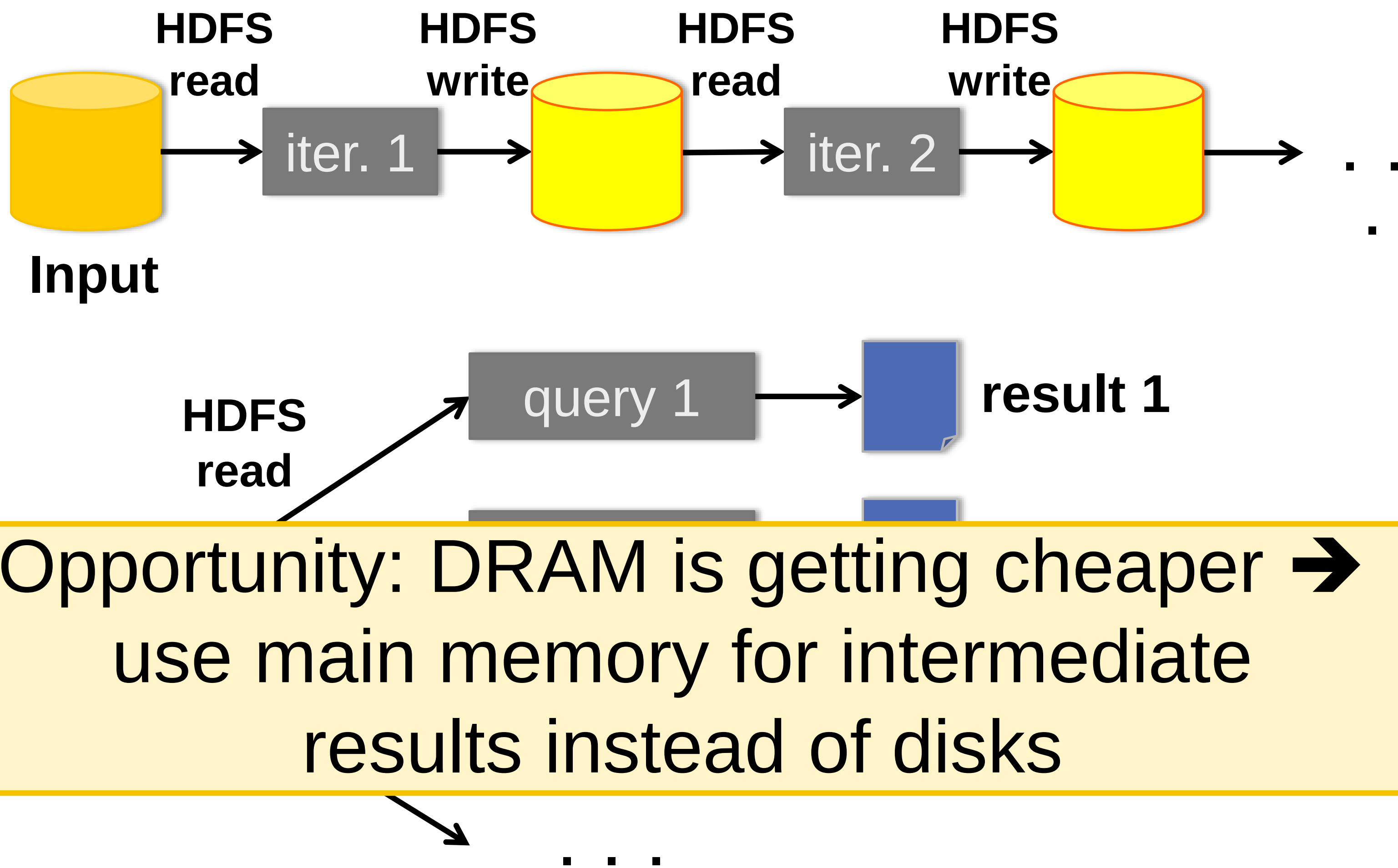
Complex jobs, interactive queries and online processing all need one thing that MR lacks:

Efficient primitives for data sharing



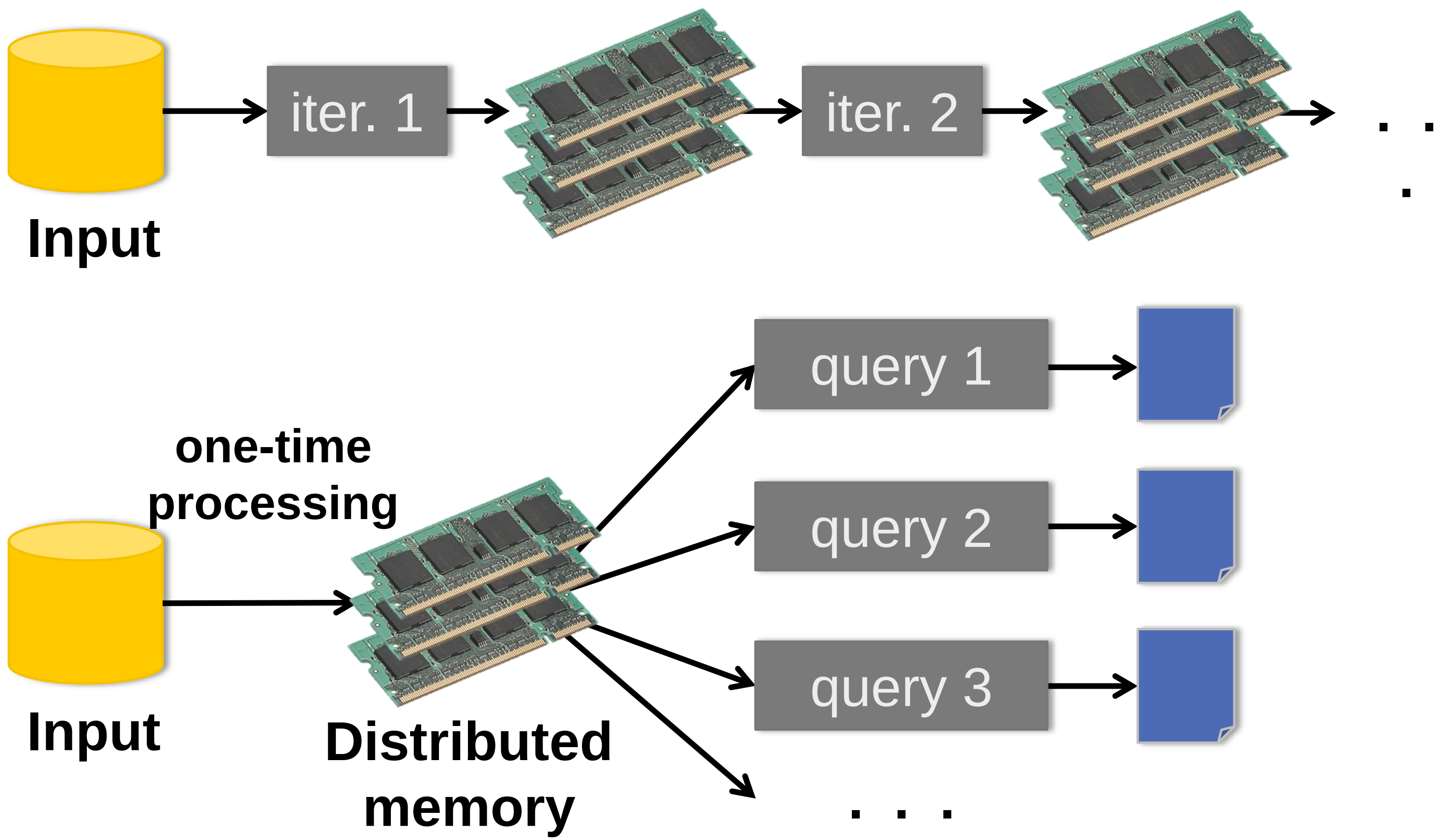
Problem: in MR, the only way to share data across jobs is using stable storage (e.g. file system) ➔ slow!

EXAMPLES



Opportunity: DRAM is getting cheaper ➔ use main memory for intermediate results instead of disks

GOAL: IN-MEMORY DATA SHARING



10-100 × faster than network and disk

SPARK PROGRAMMING MODEL

Resilient distributed datasets (RDDs)

- Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
- Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
- Can be *cached* across parallel operations

Parallel operations on RDDs

- Reduce, collect, count, save, …

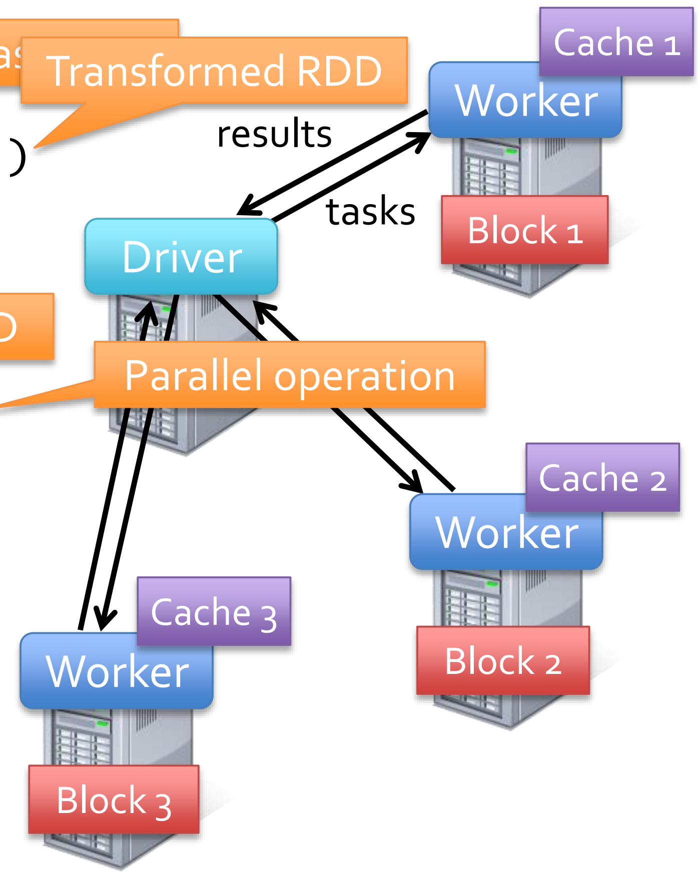
Restricted shared variables

- Accumulators, broadcast variables

Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

RDDS IN MORE DETAIL

An RDD is an immutable, partitioned, logical collection of records

- Need not be materialized, but rather contains information to rebuild a dataset from stable storage

Partitioning can be based on a key in each record (using hash or range partitioning)

Built using bulk transformations on other RDDs

Can be cached for future reuse

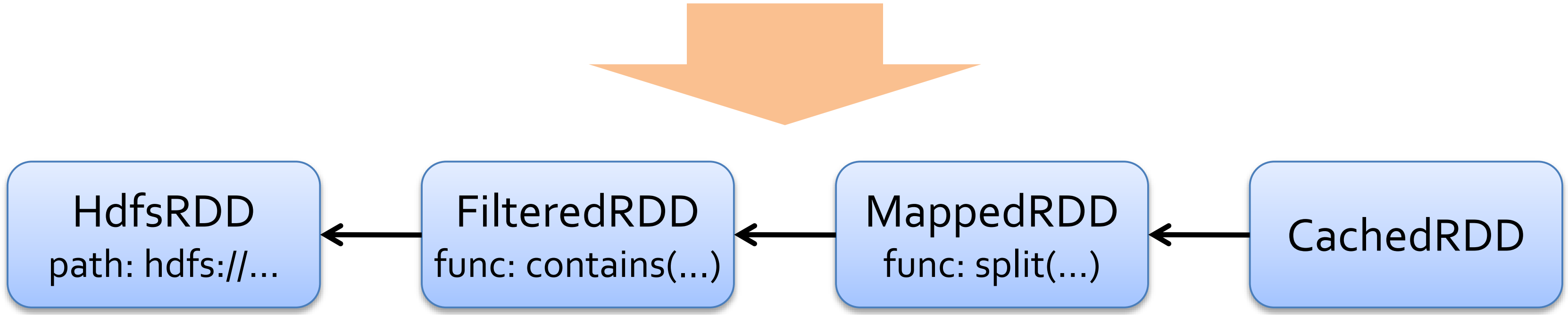
RDD OPERATIONS

Transformations (define a new RDD)	Parallel operations (return a result to driver)
map	reduce
filter	collect
sample	count
union	save
groupByKey	lookupKey
reduceByKey	...
join	
cache	
...	

RDD FAULT TOLERANCE

RDDs maintain *lineage* information that can be used to reconstruct lost partitions

Ex:
```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .cache()
```



BENEFITS OF RDD MODEL

Consistency is easy due to immutability

Inexpensive fault tolerance (log lineage rather than replicating/checkpointing data)

Locality-aware scheduling of tasks on partitions

Despite being restricted, model seems applicable to a broad variety of applications

RDDS VS DISTRIBUTED SHARED MEMORY

Concern	RDDs	Distr. Shared Mem.
Reads	Fine-grained	Fine-grained
Writes	Bulk transformations	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using speculative execution	Difficult
Work placement	Automatic based on data locality	Up to app (but runtime aims for transparency)