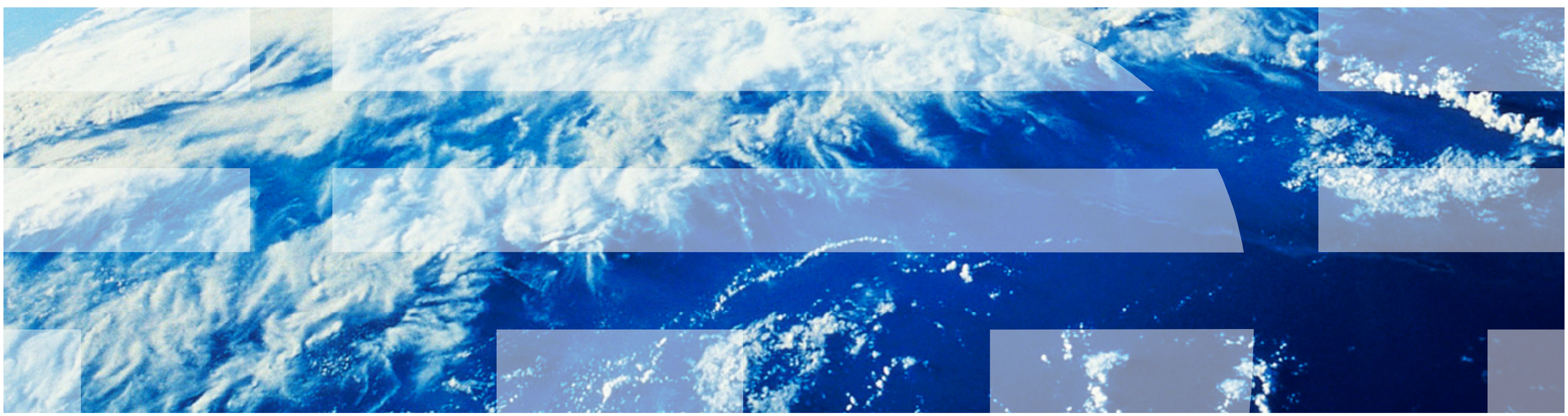
Big Data - Hadoop/MapReduce

Sambit Sahu
Credit to: Rohit Wagle and Juan Rodriguez



Agenda

- Why Big Data?
- Apache Hadoop
  - Introduction
  - Architecture
  - Programming

Hypothetical Job

- You just got an awesome job at data-mining start-up .. Congratulations !!
  - Free Snacks, Soda and Coffee --- Yayy!!

- Your first day of work you are given a task
  - The company has a new algorithm they want you to test.
  - Your boss gives you
    - The algorithm library
    - A test machine and
    - 1GB input data file

Java Document Scorer Program

```
public static long scoreDocument(String fileName, Scorer scorer) throws IOException  {

    BufferedReader reader = new BufferedReader(new FileReader(fileName));

    long totalScore = 0;

    String line = null;
    while((line = reader.readLine()) != null) {
        totalScore += scorer.getScore(line);
    }
    reader.close();

    return totalScore;
}
```

Read Input

Process Data

Throughput 1GB per hour.

What if we wanted to process 10GB data set? 10hours!!
How can we improve the performance?

Some Options

1. Faster CPU
2. More Memory
3. Increase the number of cores
4. Increase the number of threads
5. Increase the number of threads and cores

Java Document Scorer Program – Multi Threaded

```
public static long scoreDocument(String fileName, Scorer scorer, int threads)
        throws Exception  {

    BufferedReader reader =
            new BufferedReader(new FileReader(fileName));

    //thread safe structures
    BlockingQueue<String> queue =
            new LinkedBlockingQueue<String>(threads + 100);

    //initialize and start threads
    AtomicInteger completionCounter = new AtomicInteger();
    ExecutorService executors = Executors.newFixedThreadPool(threads);
    List<CounterThread> counters = new ArrayList<CounterThread>();
    for(int i=0;i<threads; i++) {
        executors.execute(new CounterThread(queue, scorer, completionCounter));
    }
    String line = null;
    while((line = reader.readLine()) != null) {
        queue.put(line);
    }

    //terminating condition for threads
    for(int i=0;i<threads; i++)
        queue.put("EXIT");
    while(completionCounter.intValue() < threads) {
        Thread.sleep(100);
    }

    executors.shutdown();
    reader.close();

    //summarize results
    long total = 0;
    for(CounterThread counter : counters)
        total+=counter.getTotal();

    return total;
}
```

Throughput 4GB per hour.

How long for 100GB?
What else can we do?

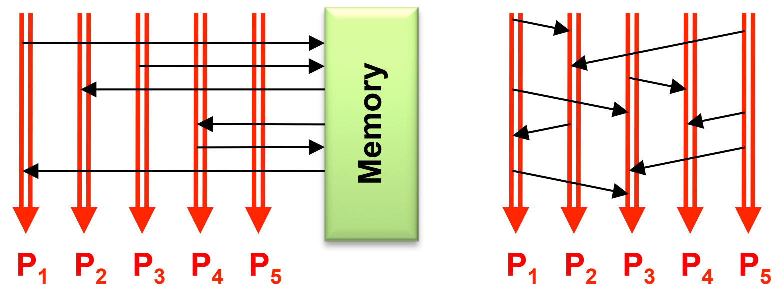
```
static class CounterThread implements Runnable {
    BlockingQueue<String> queue = null;
    Scorer scorer = null; AtomicInteger ai = null; long total=0;
    CounterThread(BlockingQueue<String> queue, Scorer scorer,
            AtomicInteger ai ) {
        this.queue=queue; this.scorer = scorer; this.ai = ai;
    }
    @Override
    public void run() {
        while(true) {
            try {
                String line = queue.take();
                //terminating condition
                if(line.equals("EXIT")) break;
                total+=scorer.getScore(line);
            }catch(Exception e) {
                break;
            }
        }
        ai.incrementAndGet();
    }

    public long getTotal() {
        return total;
    }
}
```

Get An Even Faster Machine with more Cores?

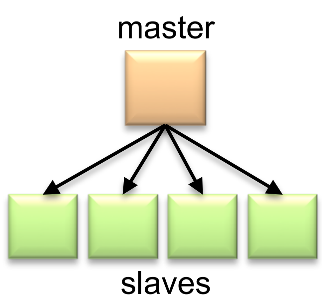
Source: MIT Open Courseware

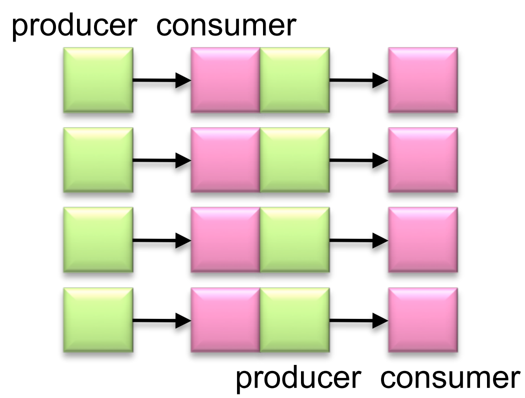
Current Tools

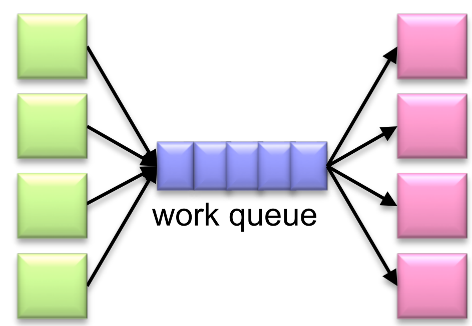
- Programming models
  - Shared memory (pthreads)
  - Message passing (MPI)



- Design Patterns
  - Master-slaves
  - Producer-consumer flows
  - Shared work queues







Where the rubber meets the road

- Concurrency is difficult to reason about
- Concurrency is even more difficult to reason about
  - At the scale of datacenters (even across datacenters)
  - In the presence of failures
  - In terms of multiple interacting services
- Not to mention debugging…
- The reality:
  - Lots of one-off solutions, custom code
  - Write you own dedicated library, then program with it
  - Burden on the programmer to explicitly manage everything

What's the common theme?

- To improve performance, you have to re-write the code

- The code has to adapt to the expected performance.
  - This doesn't work since you may not know the amount of data beforehand.

- The actual Intellectual Property (IP) of the company is the analytic algorithm
  - However a lot of effort is spent on scaling the analytic

Big Data - Motivation

- Google processes 20 PB a day (2008)
- Wayback Machine has 3 PB + 100 TB/month (3/2009)
- Facebook has 2.5 PB of user data + 15 TB/day (4/2009)
- eBay has 6.5 PB of user data + 50 TB/day (5/2009)
- CERN's LHC will generate 15 PB a year

Enter .. Apache Hadoop

- Hadoop is a high-level Open Source project
  - Under Apache Software Foundation
  - Inspired by Google's MapReduce and GFS papers
- It contains several individual projects
  - HDFS
  - MapReduce
  - Yarn
- It also has a slew of related projects
  - PIG
  - HIVE
  - Hbase
- Has been implemented for the most part in Java.

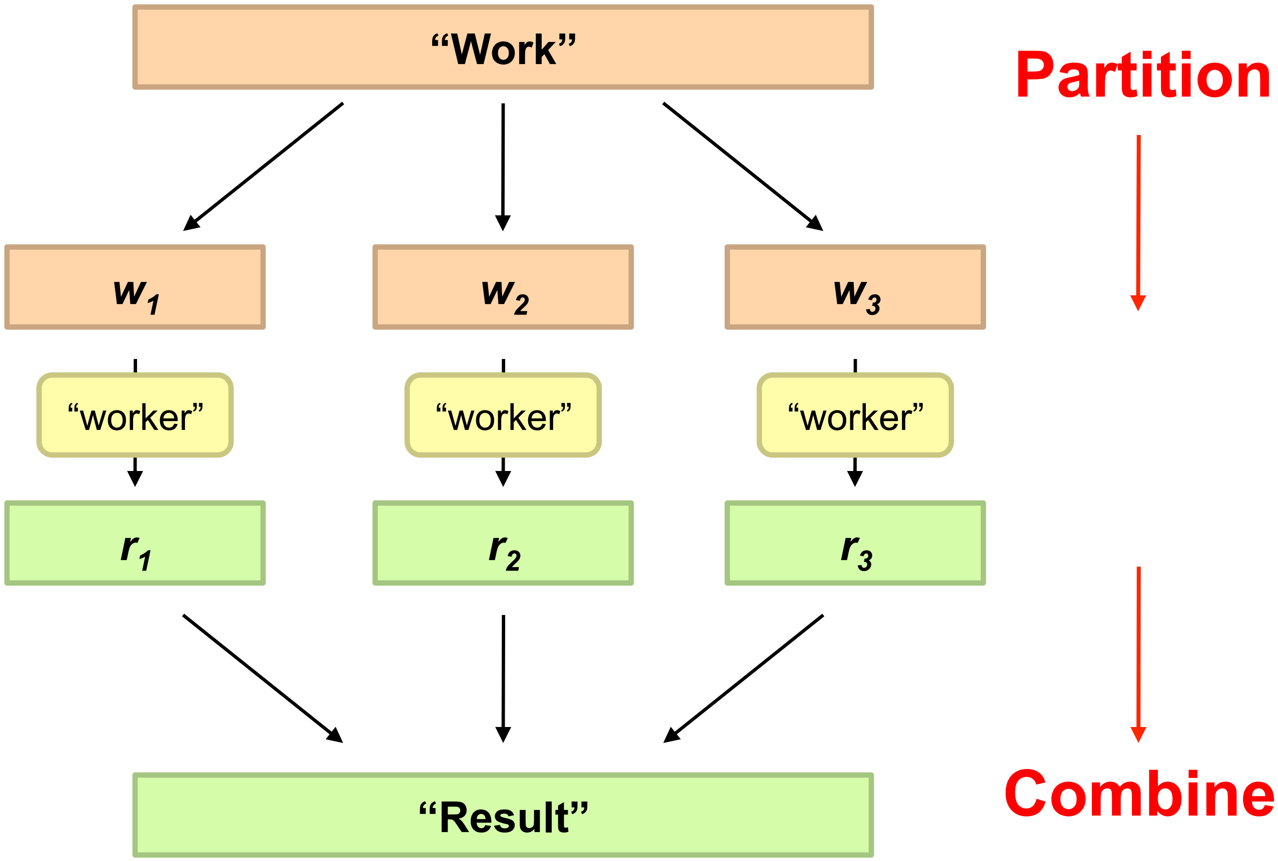
A closer look

```java
public static long scoreDocument(String fileName, Scorer scorer, int threads)
        throws Exception  {

    BufferedReader reader =
            new BufferedReader(new FileReader(fileName));

    //thread safe structures
    BlockingQueue<String> queue =
            new LinkedBlockingQueue<String>(threads + 100);

    //initialize and start threads
    AtomicInteger completionCounter = new AtomicInteger();
    ExecutorService executors = Executors.newFixedThreadPool(threads);
    List<CounterThread> counters = new ArrayList<CounterThread>();
    for(int i=0;i<threads; i++) {
        executors.execute(new CounterThread(queue, scorer, completionCounter));
    }
    String line = null;
    while((line = reader.readLine()) != null) {
        queue.put(line);
    }

    //terminating condition for threads
    for(int i=0;i<threads; i++)
        queue.put("EXIT");
    while(completionCounter.intValue() < threads) {
        Thread.sleep(100);
    }

    executors.shutdown();
    reader.close();

    //summarize results
    long total = 0;
    for(CounterThread counter : counters)
        total+=counter.getTotal();

    return total;
}
```

Partition Work

Combine Results

Divide and Conquer



Parallelization Challenges

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?

What is the common theme of all of these problems?

What’s the point?

- It’s all about the right level of abstraction
  - The von Neumann architecture has served us well, but is no longer appropriate for the multi-core/cluster environment
- Hide system-level details from the developers
  - No more race conditions, lock contention, etc.
- Separating the *what* from *how*
  - Developer specifies the computation that needs to be performed
  - Execution framework (“runtime”) handles actual execution

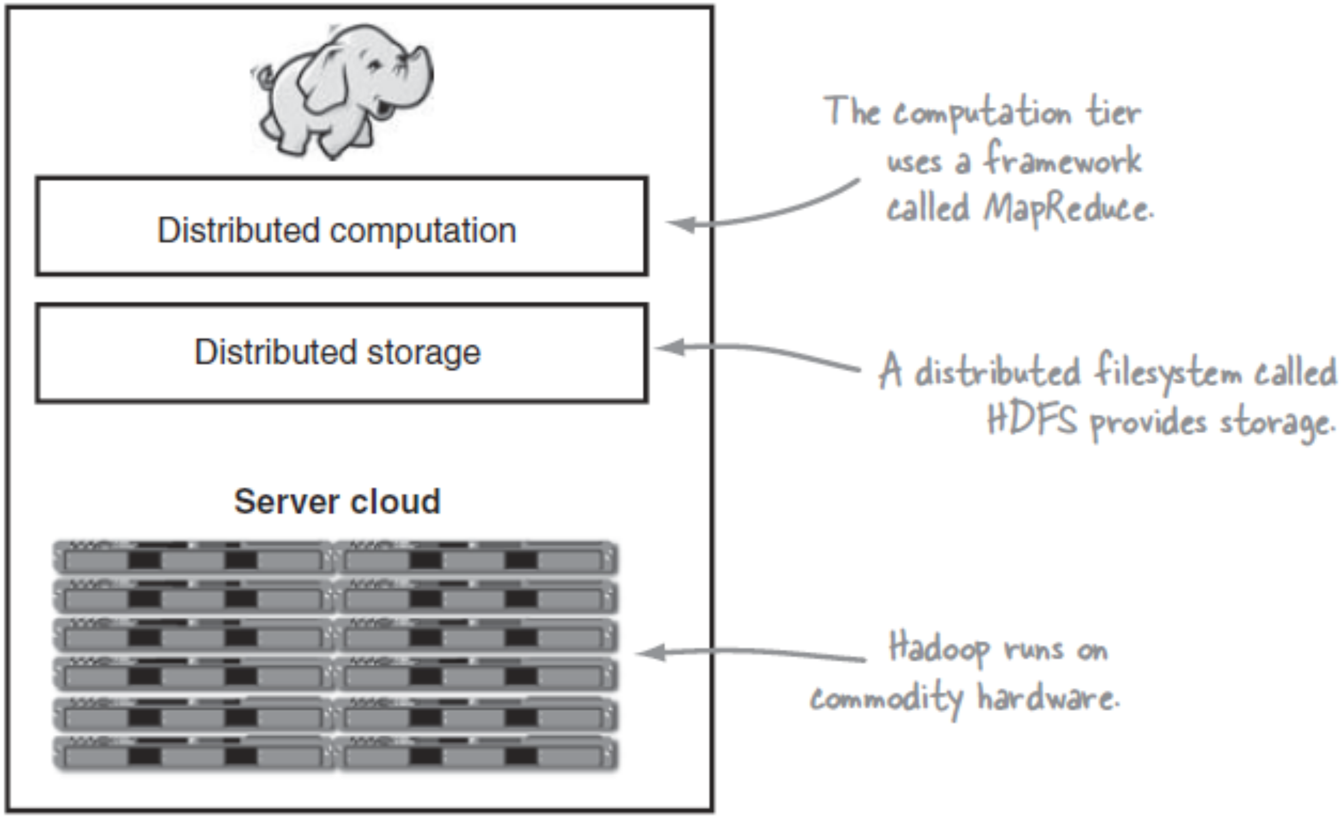
The datacenter *is* the computer!

“Big Ideas”

- Scale “out”, not “up”
  - Limits of SMP and large shared-memory machines
- Move processing to the data
  - Cluster have limited bandwidth
- Process data sequentially, avoid random access
  - Seeks are expensive, disk throughput is reasonable
- Seamless scalability
  - From the mythical man-month to the tradable machine-hour

Hadoop

- Platform for distributed **storage** and **computation**
  - **HDFS**
  - **MapReduce**
  - **Ecosystem**



Source: **Hadoop in Practice**, Alex Holmes, Manning Publications Co., 2012

What are we missing here?

```java
public static long scoreDocument(String fileName, Scorer scorer, int threads)
        throws Exception  {

    BufferedReader reader =
            new BufferedReader(new FileReader(fileName));

    //thread safe structures
    BlockingQueue<String> queue =
            new LinkedBlockingQueue<String>(threads + 100);

    //initialize and start threads
    AtomicInteger completionCounter = new AtomicInteger();
    ExecutorService executors = Executors.newFixedThreadPool(threads);
    List<CounterThread> counters = new ArrayList<CounterThread>();
    for(int i=0;i<threads; i++) {
        executors.execute(new CounterThread(queue, scorer, completionCounter));
    }
    String line = null;
    while((line = reader.readLine()) != null) {
        queue.put(line);
    }

    //terminating condition for threads
    for(int i=0;i<threads; i++)
        queue.put("EXIT");
    while(completionCounter.intValue() < threads) {
        Thread.sleep(100);
    }

    executors.shutdown();
    reader.close();

    //summarize results
    long total = 0;
    for(CounterThread counter : counters)
        total+=counter.getTotal();

    return total;
}
```

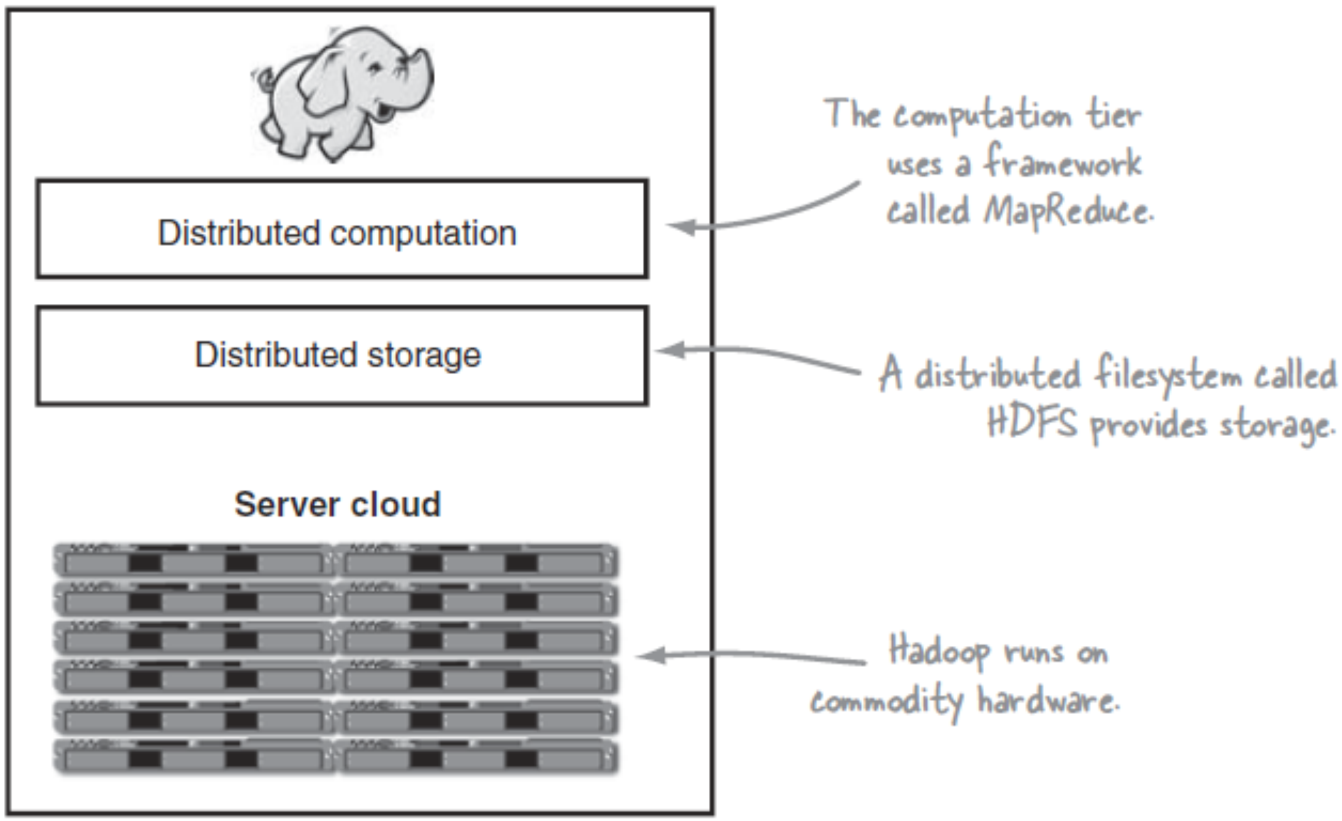
Sequential File Read

Partition Work

Combine Results

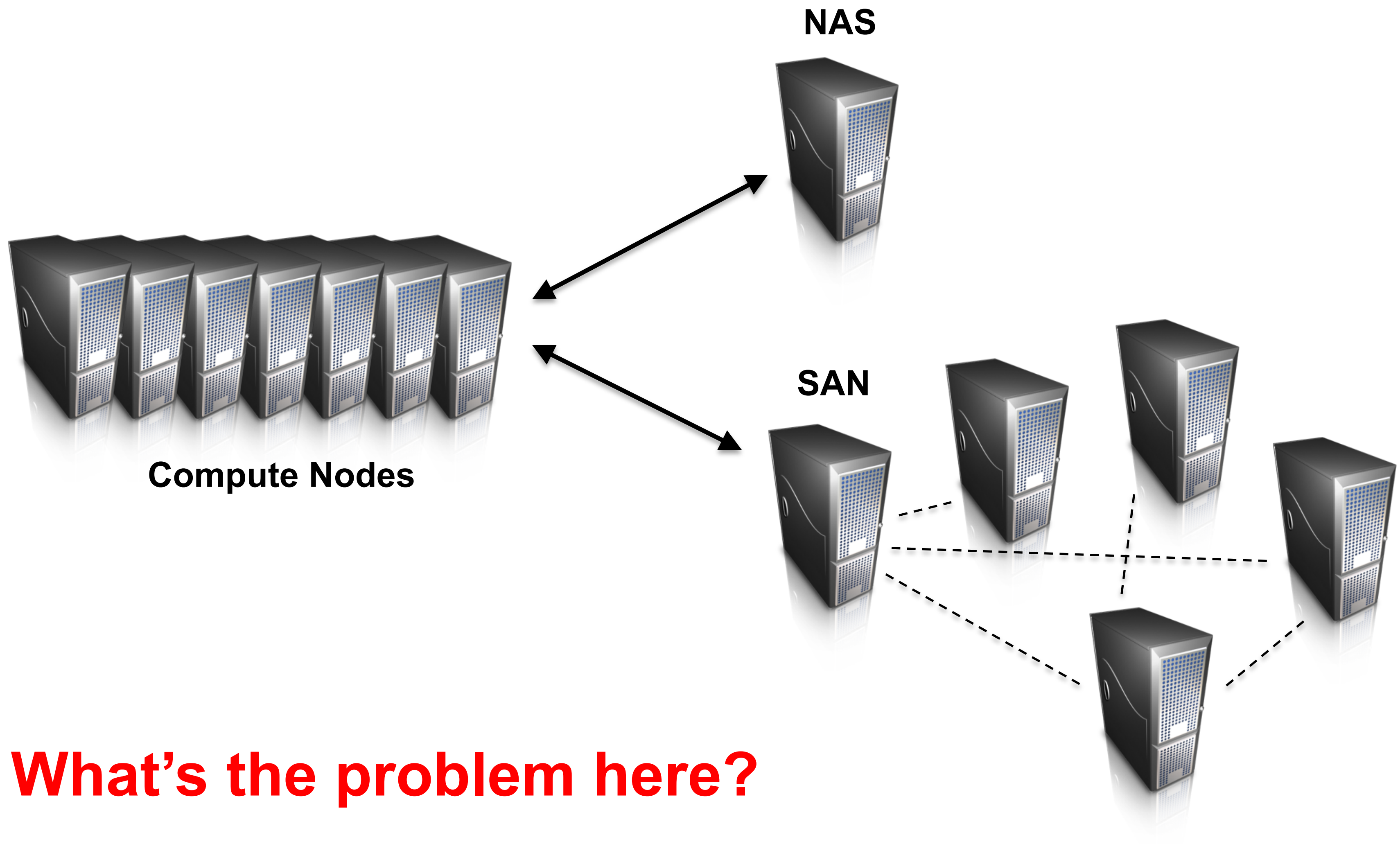
Hadoop

- Platform for distributed **storage** and **computation**
  - **HDFS**
  - **MapReduce**
  - **Ecosystem**



Source: **Hadoop in Practice**, Alex Holmes, Manning Publications Co., 2012

How do we get data to the workers?

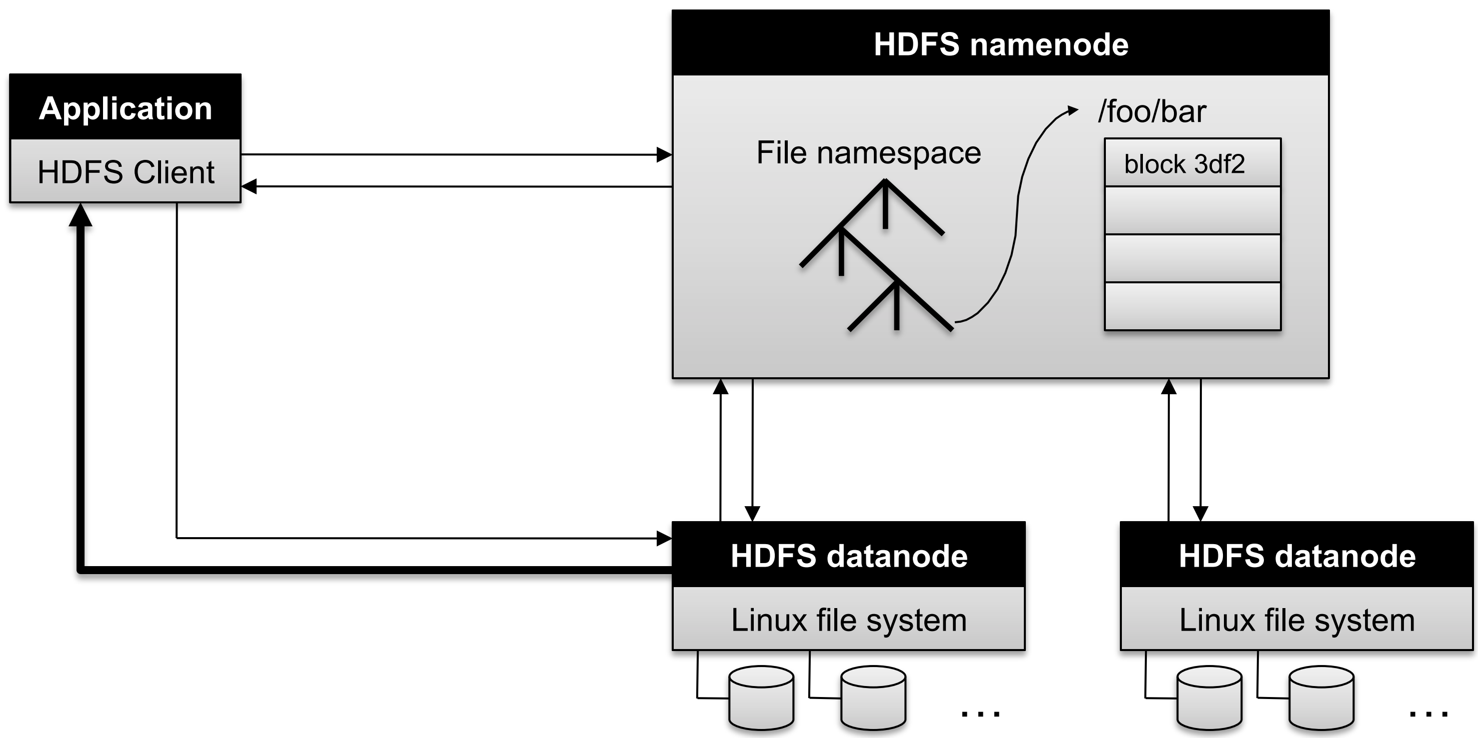


What's the problem here?

HDFS: Assumptions

- Commodity hardware over "exotic" hardware
  - Scale "out", not "up"
- High component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of huge files
  - Multi-gigabyte files are common, if not encouraged
- Files are write-once, read many
  - Perhaps concurrently
- Large streaming reads over random access
  - High sustained throughput over low latency

HDFS Architecture



Adapted from (Ghemawat et al., SOSP 2003)

How HDFS works

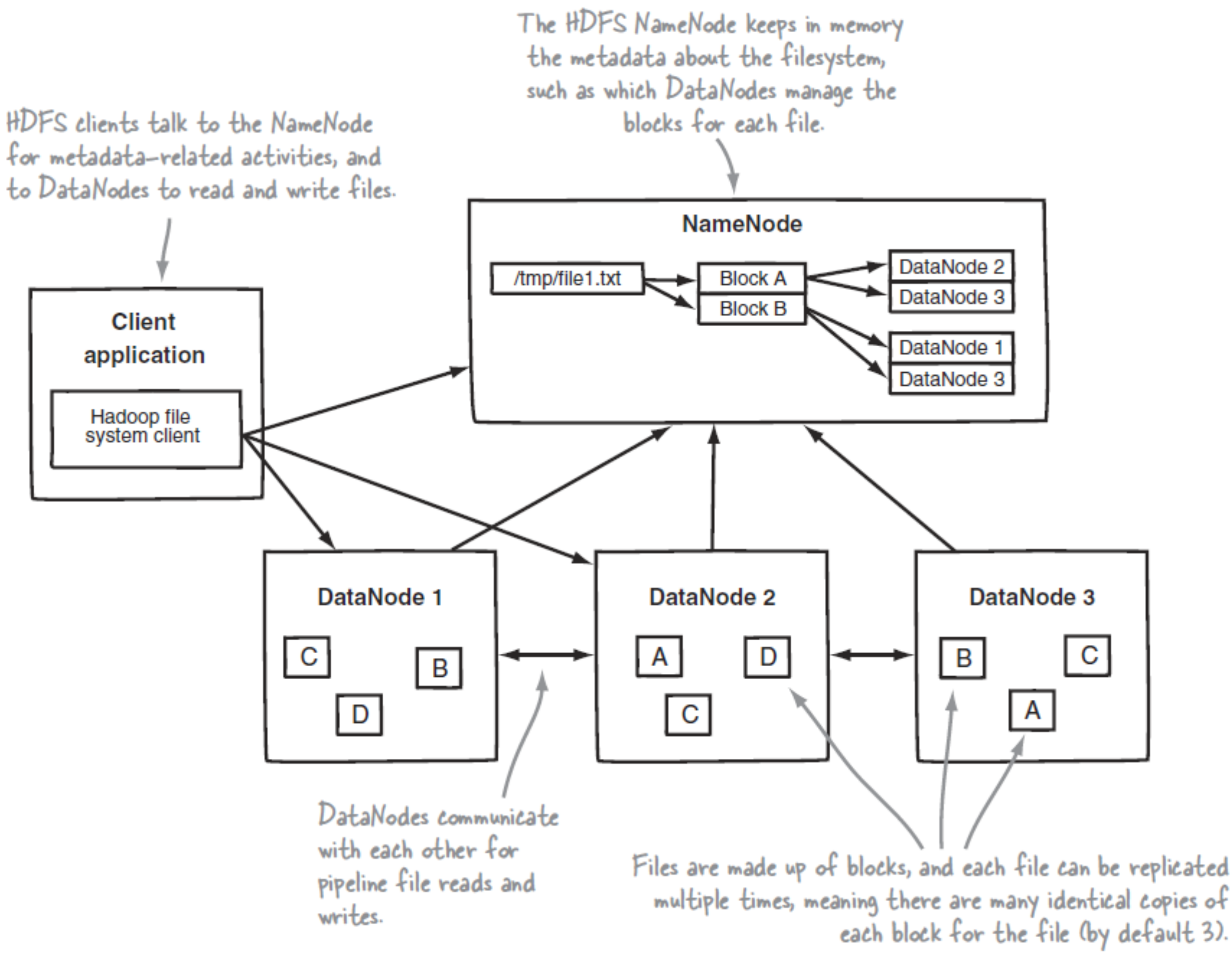
- When an input file is added to HDFS
  - File is split into smaller blocks of fixed size
  - Each block is replicated
  - Each replicated block is stored on a different host
- Block size is configurable. Default is 128/256MB.
- Replication level is configurable. Default is 3
  - Replication is necessary for
    - Scaling
    - High Availability
- In case a host crashes or is removed
  - All blocks on that host are automatically replicated to other hosts
- In case a host is added
  - Blocks will be rebalanced so that some blocks from other hosts will be placed on the new host

HDFS Component Responsibilities

- Name Node
  - Managing the file system namespace:
    - Holds file/directory structure, metadata, file-to-block mapping, access permissions, etc.
  - Coordinating file operations:
    - Directs clients to datanodes for reads and writes
    - No data is moved through the namenode
  - Maintaining overall health:
    - Periodic communication with the datanodes
    - Block re-replication and rebalancing
    - Garbage collection

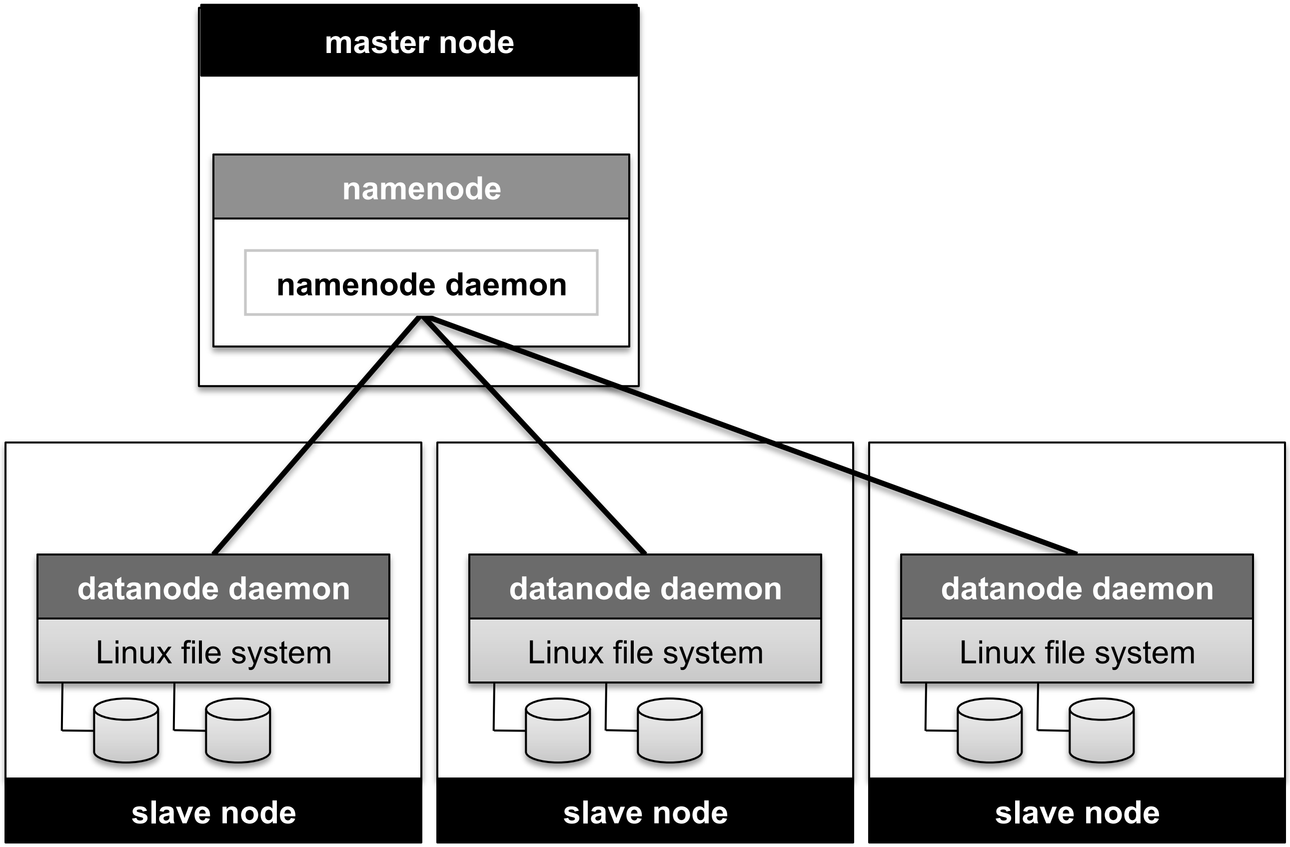
- Data Node
  - Actual storage and management of data block on a single host
  - Provides clients with access to data

HDFS



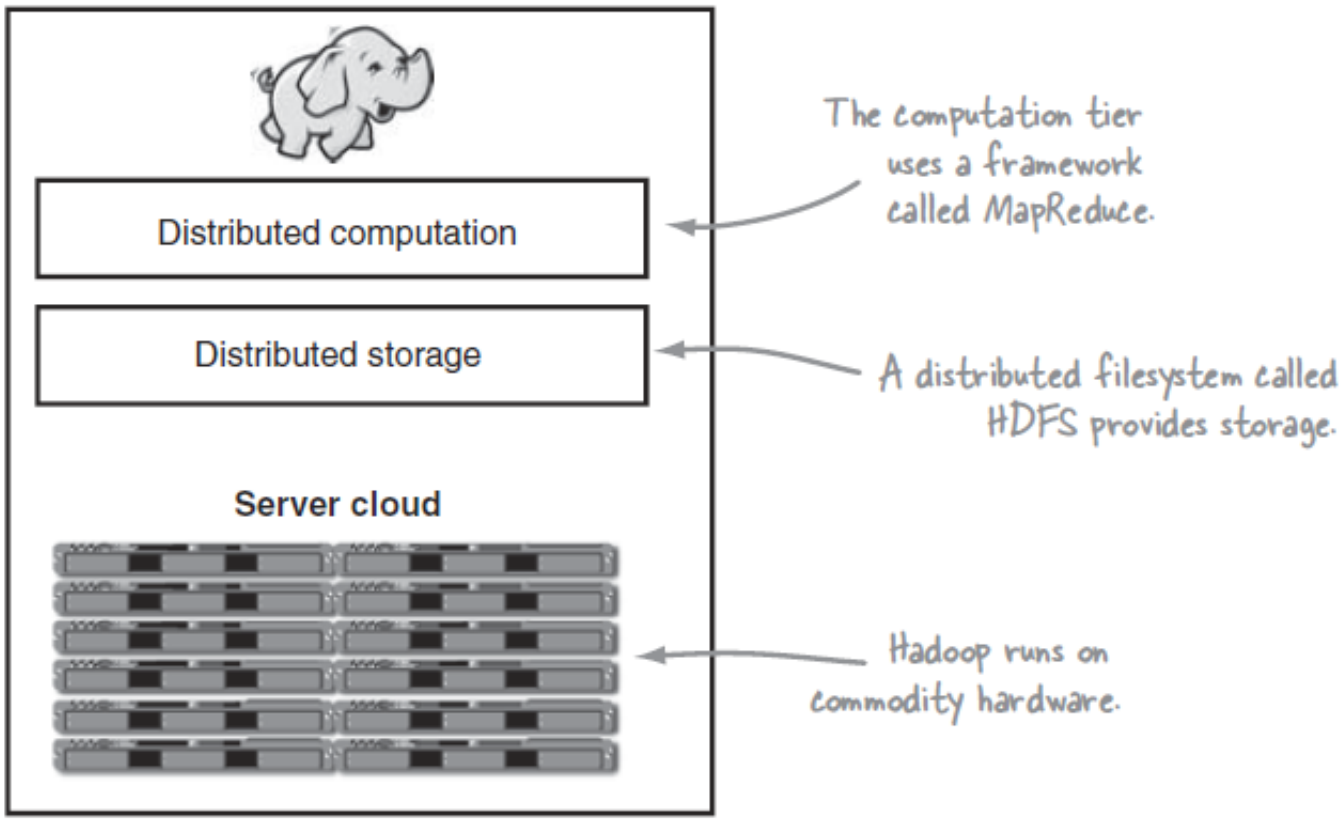
HDFS architecture shows an HDFS client communicating with the master NameNode

HDFS Components in Cluster



Hadoop

- Platform for distributed **storage** and **computation**
  - **HDFS**
  - **MapReduce**
  - **Ecosystem**



Source: **Hadoop in Practice**, Alex Holmes, Manning Publications Co., 2012

MapReduce (MR) can refer to…

- The execution framework (aka "runtime")
- The programming model
- The specific implementation

Usage is usually clear from context!

MR Framework Components

- Job Tracker
  - Central component responsible for managing job lifecycles
  - One Job Tracker per MR framework instance
  - Accepts job submissions, queries etc. from clients
  - Enqueues jobs and schedules individual tasks.
  - Communicates with Task Trackers to deploy and run tasks
  - Attempts to assign tasks to support Data Locality.

- Task Tracker
  - One Task Tracker per host
  - Runs and manages individual tasks
  - Communicates progress of tasks back to Job Tracker.

MR Programming Model

- Programmers specify two functions:

  map (k, v) → <k’, v’>*

  reduce (k’, v’) → <k’, v’>*

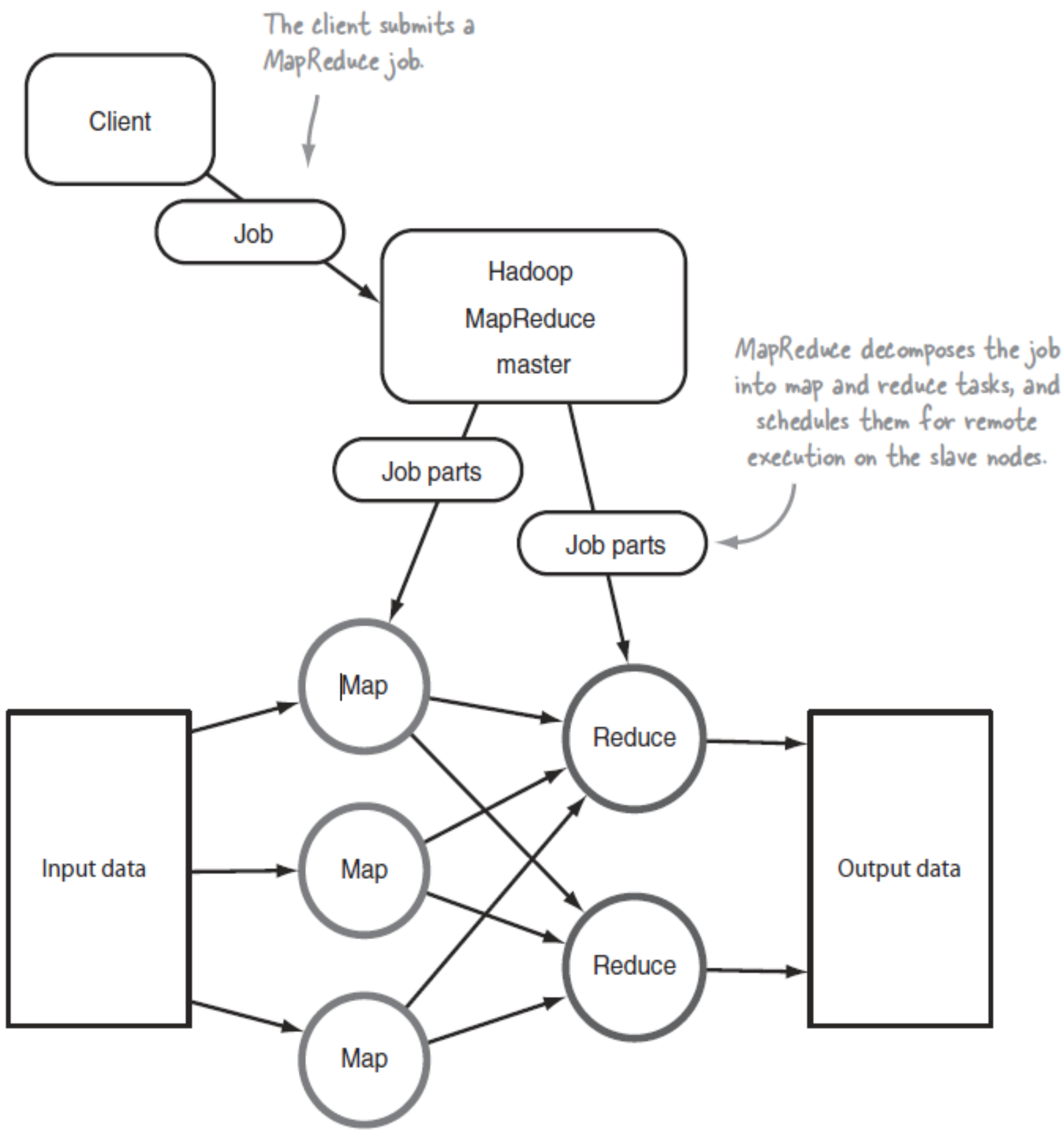
  - All values with the same key are sent to the same reducer

- The MR Execution framework handles everything else…

What’s “everything else”?

MapReduce

- Everything Else
- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (HDFS)



Our Scoring Algorithm as a Map Reduce Program

```
//MAPPER
public static class Map
    extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable> {

    private Scorer scorer = new MyScorer();

    public void map(
            LongWritable key, Text value, //map input
            OutputCollector<Text, LongWritable> output,//map output <key, value>
            Reporter reporter) throws IOException {
        String line = value.toString();
        output.collect(mykey, new LongWritable(scorer.getScore(line)));
    }
}
```

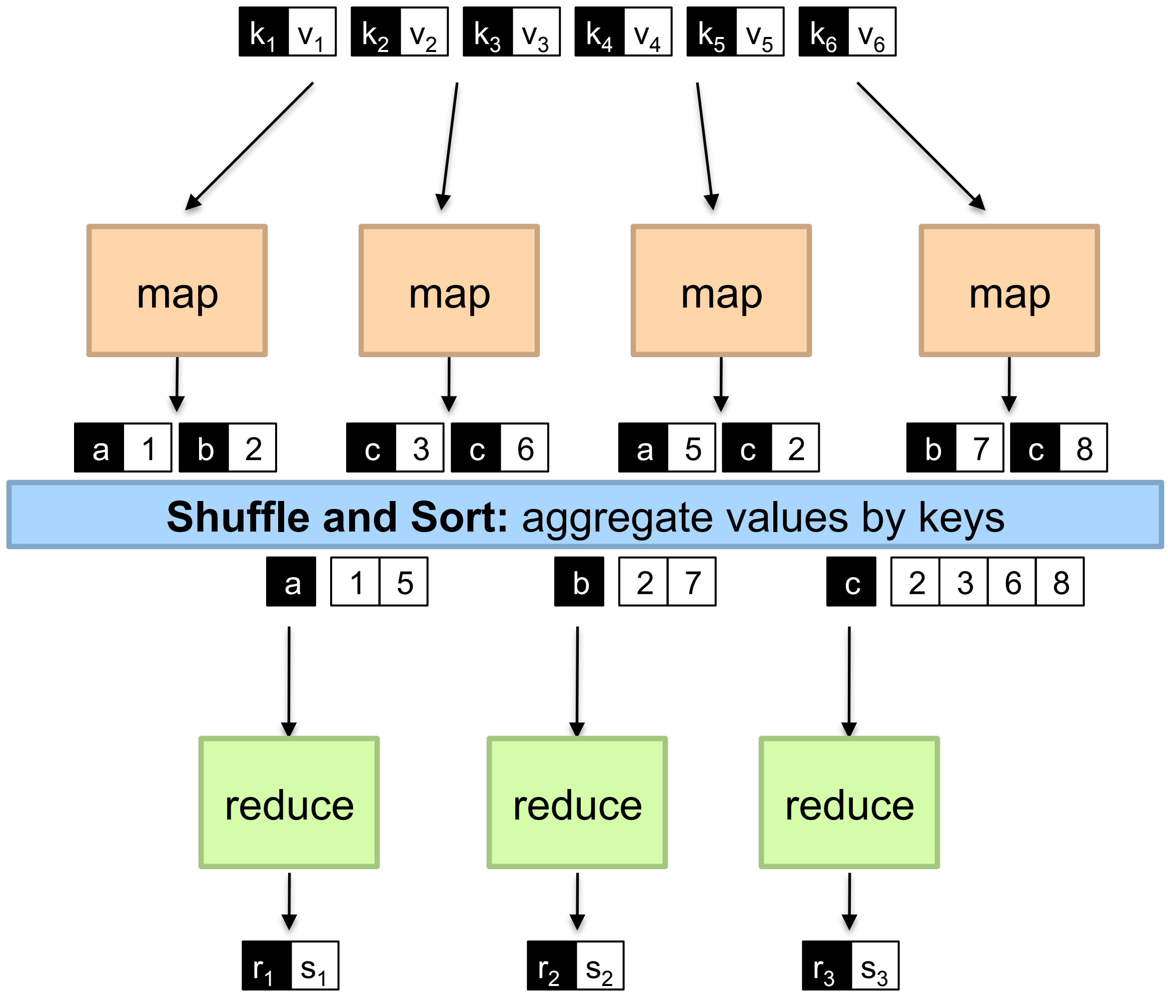
Our Analytic

```
//REDUCER
public static class Reduce
    extends MapReduceBase
    implements Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(
                Text key, Iterator<LongWritable> values, //reducer input
                OutputCollector<Text, LongWritable> output, //reducer output
                Reporter reporter) throws IOException {
        long sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
            output.collect(mykey, new LongWritable(sum));
        }
    }
}
```

Basic Hadoop API*

- Mapper
  - void map(K1 key, V1 value, OutputCollector<K2, V2> output, Reporter reporter)
  - void configure(JobConf job)
  - void close() throws IOException
- Reducer/Combiner
  - void reduce(K2 key, Iterator<V2> values, OutputCollector<K3,V3> output, Reporter reporter)
  - void configure(JobConf job)
  - void close() throws IOException
- Partitioner
  - void getPartition(K2 key, V2 value, int numPartitions)

*Note: forthcoming API changes…



k_1 v_1 k_2 v_2 k_3 v_3 k_4 v_4 k_5 v_5 k_6 v_6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r_1 s_1
r_2 s_2
r_3 s_3

Lets Talk Numbers

- How many mappers?
  - Depends on the size of input data
  - Typically 1 mapper per data block
  - So 1 GB input data will have around 8 Mappers
    - Assuming 128MB block size

- How many reducers?
  - Depends on cluster reducer capacity
  - Can be set depending on the expected number of keys
  - For large data sets, set it to cluster reducer capacity

MapReduce

- Programmers specify two functions:

  map (k, v) → <k’, v’>*

  reduce (k’, v’) → <k’, v’>*

  - All values with the same key are reduced together

- The execution framework handles everything else…

- Not quite…usually, programmers also specify:

  combine (k’, v’) → <k’, v’>*

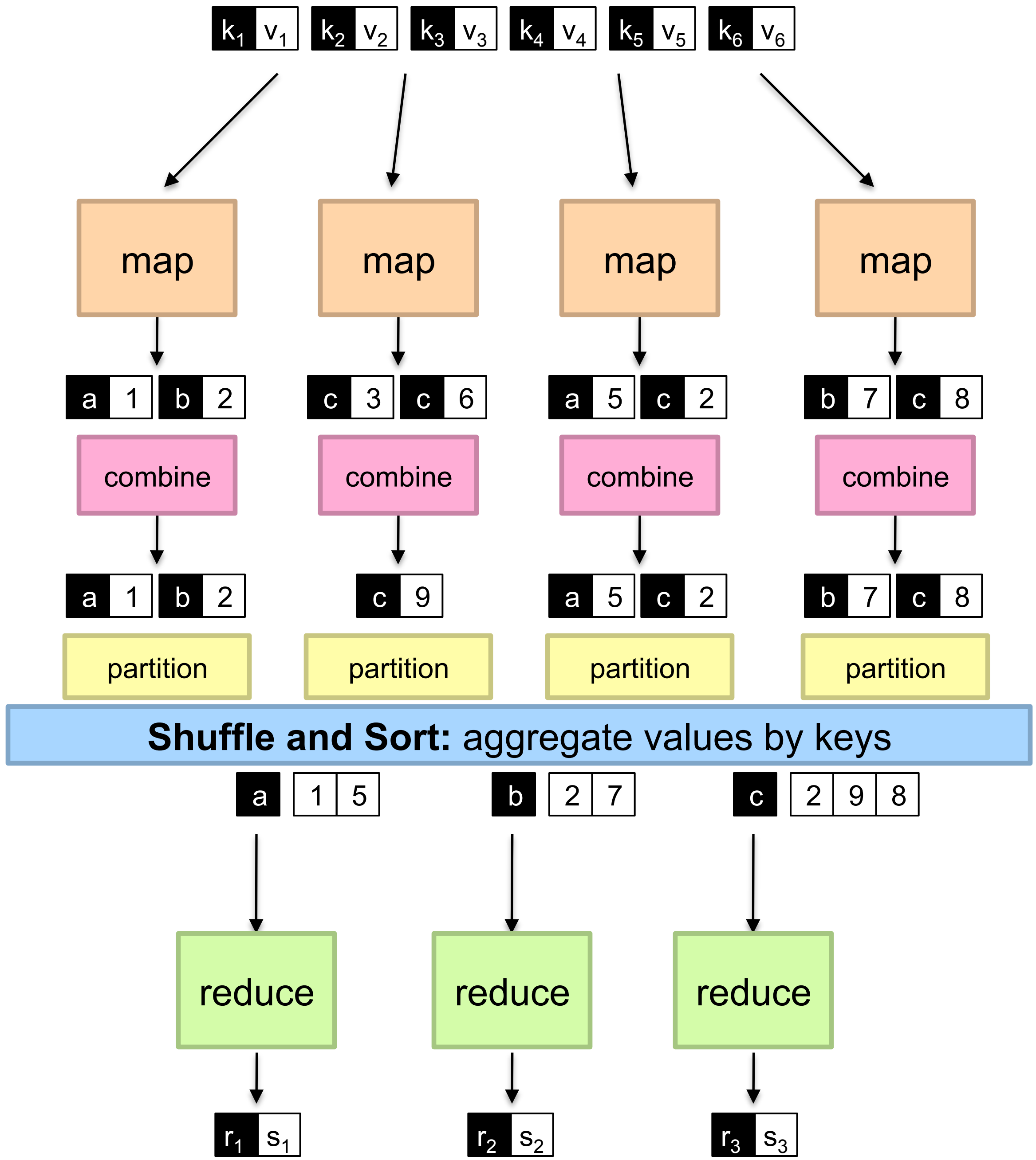
  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic

  partition (k’, number of partitions) → partition for k’

  - Often a simple hash of the key, e.g., hash(k’) mod n
  - Divides up key space for parallel reduce operations

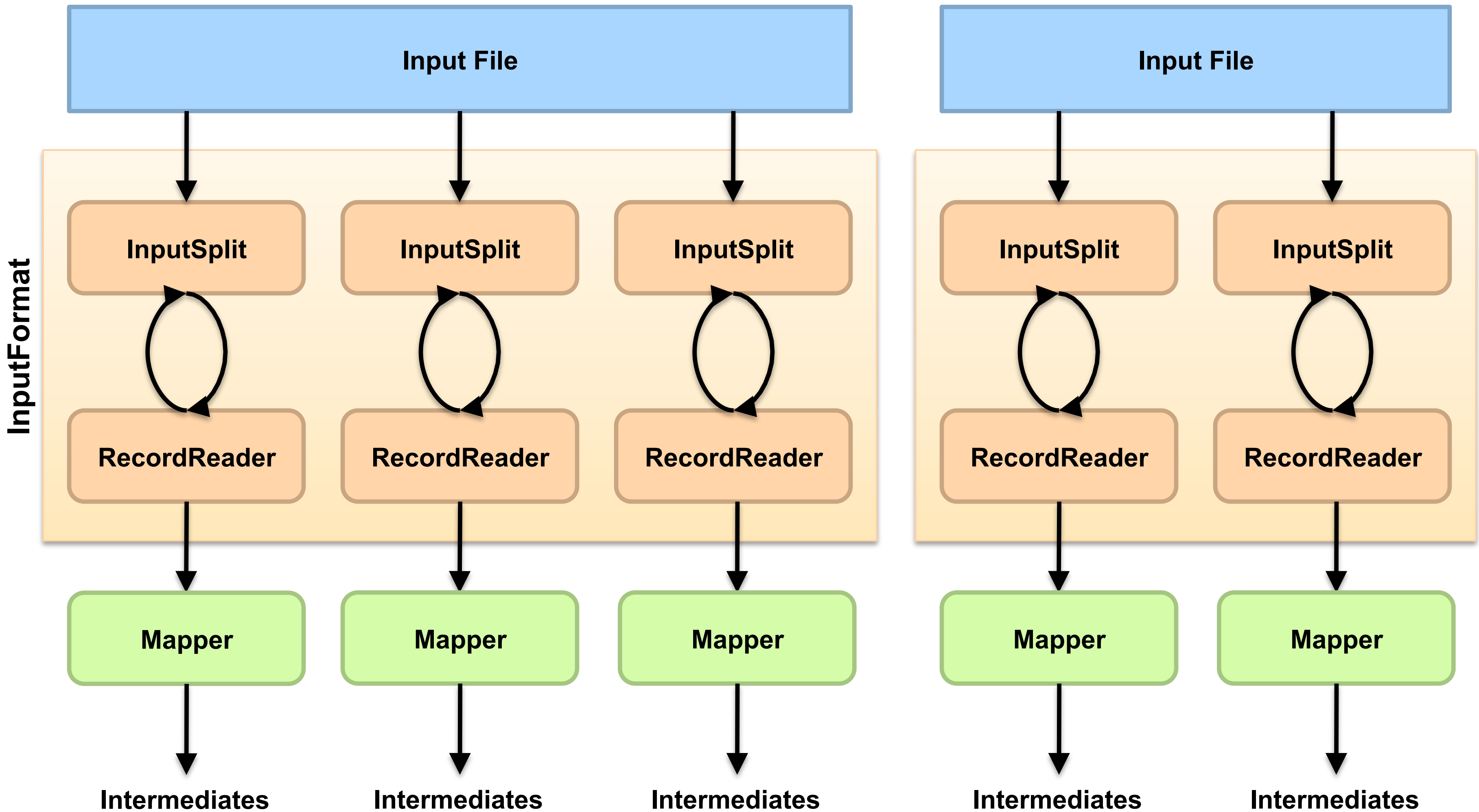
Two more details…

- Barrier between map and reduce phases
  - But we can begin copying intermediate data earlier
- Keys arrive at each reducer in sorted order
  - No enforced ordering *across* reducers



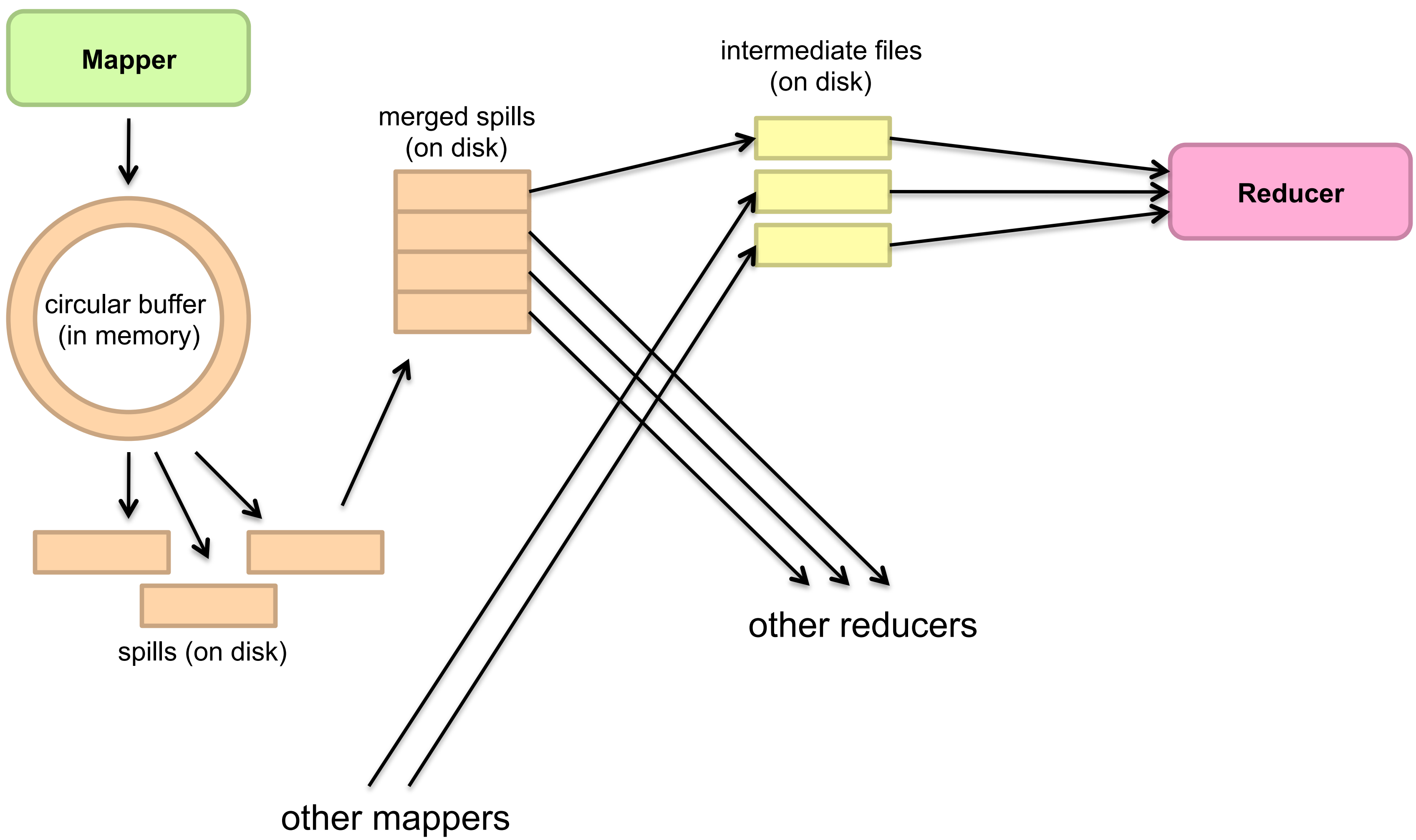
k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combine
combine
combine
combine
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Input To Mappers



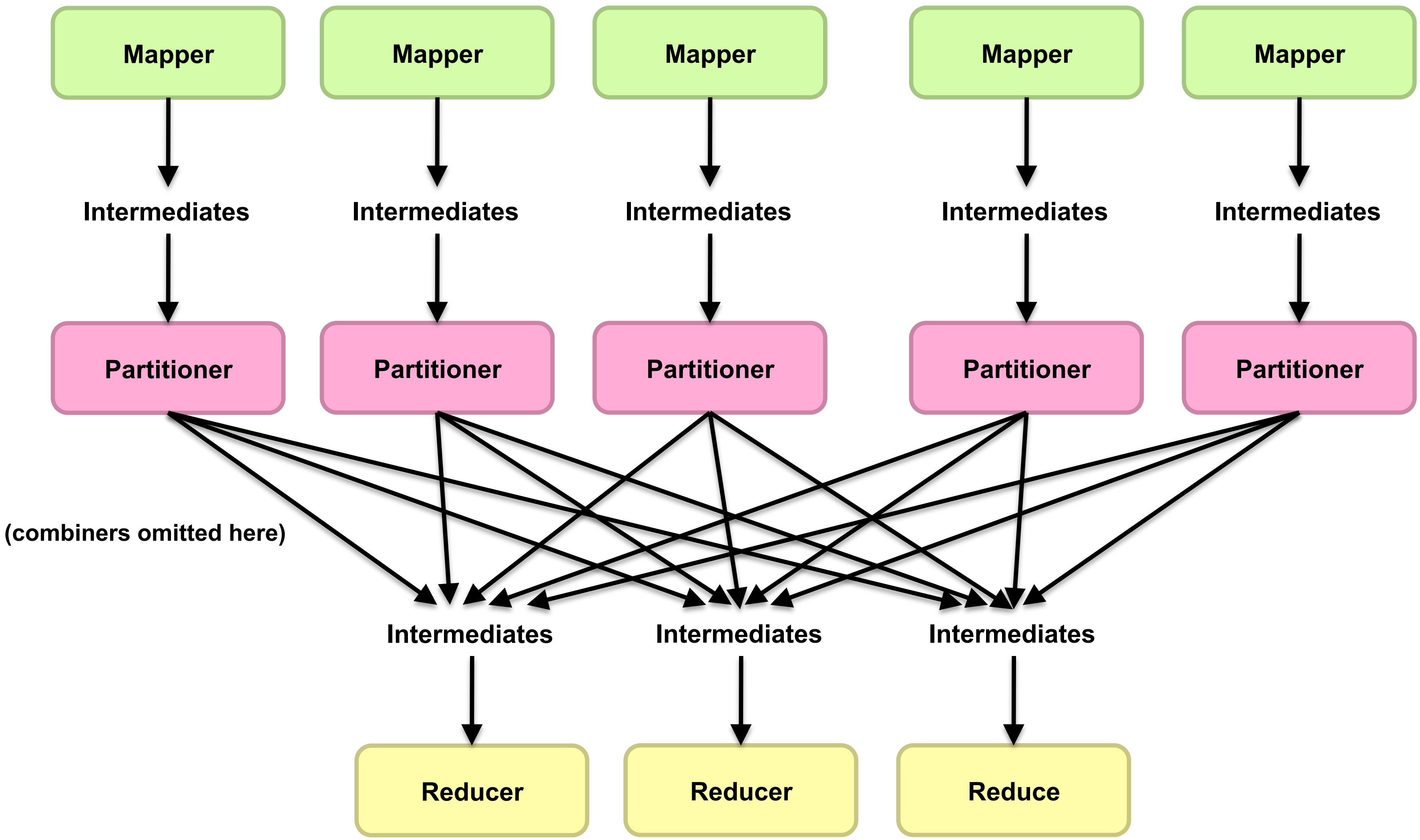
Source: redrawn from a slide by Cloduera, cc-licensed

Shuffle and Sort



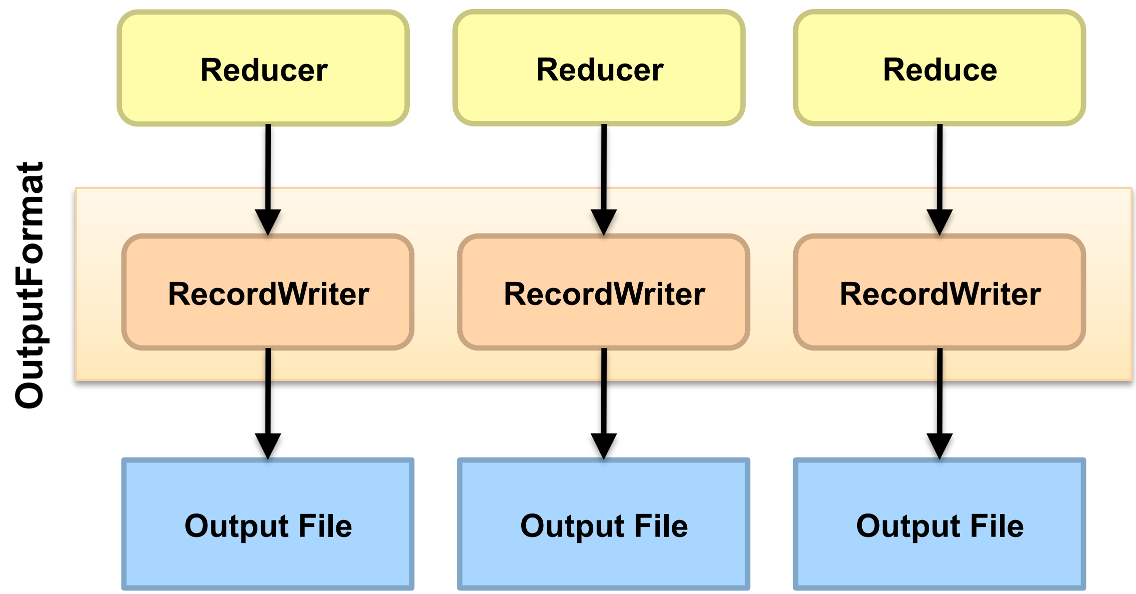
Shuffle and Sort in Hadoop

- Probably the most complex aspect of MapReduce!
- Map side
  - Map outputs are buffered in memory in a circular buffer
  - When buffer reaches threshold, contents are “spilled” to disk
  - Spills merged in a single, partitioned file (sorted within each partition): combiner runs here
- Reduce side
  - First, map outputs are copied over to reducer machine
  - “Sort” is a multi-pass merge of map outputs (happens in memory and on disk): combiner runs here
  - Final merge pass goes directly into reducer



Source: redrawn from a slide by Cloduera, cc-licensed

Reducer to Output

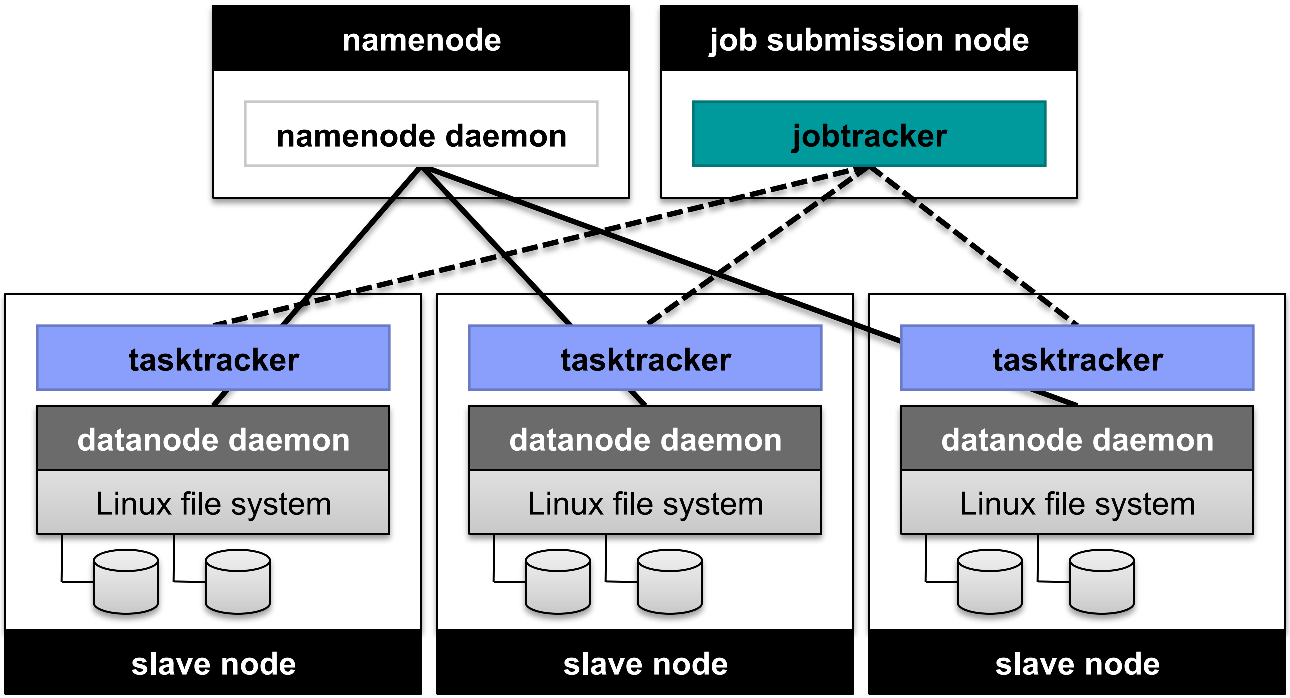


Source: redrawn from a slide by Cloduera, cc-licensed

Input and Output

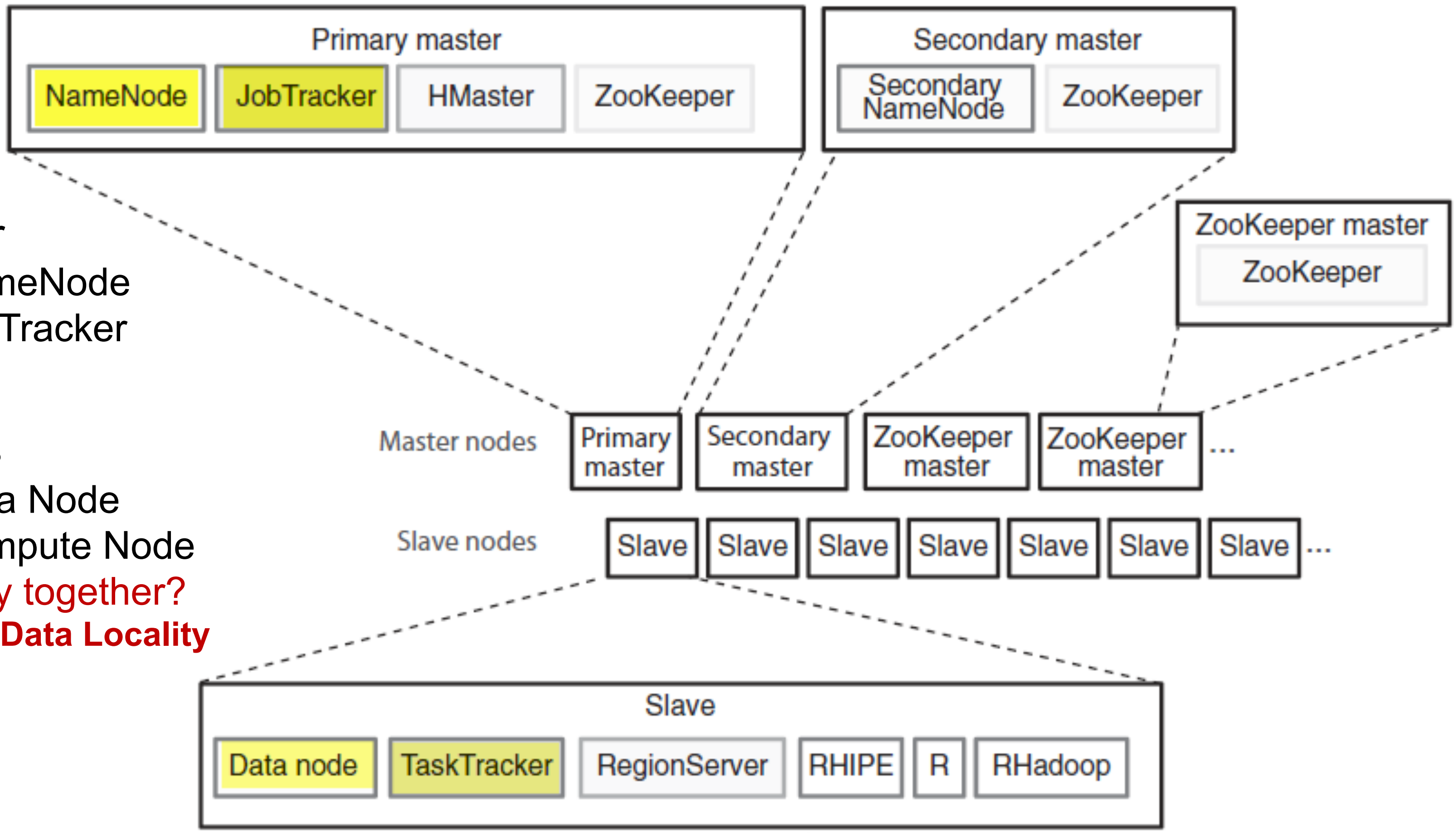
- InputFormat:
  - TextInputFormat
  - KeyValueTextInputFormat
  - SequenceFileInputFormat
  - …
- OutputFormat:
  - TextOutputFormat
  - SequenceFileOutputFormat
  - …

Putting everything together…



HADOOP Architecture

- Master
  - NameNode
  - JobTracker
- Slaves
  - Data Node
  - Compute Node
  - Why together?
    - **Data Locality**



One More Thing

- Distributed Cache
  - Usually used for files of small size
  - Provides a convenient way to propagate applications and configuration files
  - HDFS is not used handle such files due to their small size
  - Shared across all nodes in the MapReduce cluster

Dizzy Yet?

- OK, we went through a lot of details
- Whatever happened to the simplicity of programming??
- Do I really have to write a MapReduce program every time I want to run a new analytic?

We went from..

Multi-Threaded

```
public static long scoreDocument(String fileName, Scorer scorer, int threads)
        throws Exception  {

    BufferedReader reader =
            new BufferedReader(new FileReader(fileName));

    //thread safe structures
    BlockingQueue<String> queue =
            new LinkedBlockingQueue<String>(threads + 100);

    //initialize and start threads
    AtomicInteger completionCounter = new AtomicInteger();
    ExecutorService executors = Executors.newFixedThreadPool(threads);
    List<CounterThread> counters = new ArrayList<CounterThread>();
    for(int i=0;i<threads; i++) {
        executors.execute(new CounterThread(queue, scorer, completionCounter));
    }
    String line = null;
    while((line = reader.readLine()) != null) {
        queue.put(line);
    }

    //terminating condition for threads
    for(int i=0;i<threads; i++)
        queue.put("EXIT");
    while(completionCounter.intValue() < threads) {
        Thread.sleep(100);
    }

    executors.shutdown();
    reader.close();

    //summarize results
    long total = 0;
    for(CounterThread counter : counters)
        total+=counter.getTotal();

    return total;
}
```

Map-Reduce

```
//MAPPER
public static class Map
    extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, LongWritable> {

    private Scorer scorer = new MyScorer();

    public void map(
            LongWritable key, Text value, //map input
            OutputCollector<Text, LongWritable> output,//map output <key, value>
            Reporter reporter) throws IOException {
        String line = value.toString();
        output.collect(mykey, new LongWritable(scorer.getScore(line)));
    }
}

//REDUCER
public static class Reduce
    extends MapReduceBase
    implements Reducer<Text, LongWritable, Text, LongWritable> {

    public void reduce(
                Text key, Iterator<LongWritable> values, //reducer input
                OutputCollector<Text, LongWritable> output, //reducer output
                Reporter reporter) throws IOException {
        long sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
            output.collect(mykey, new LongWritable(sum));
        }
    }
}
```

Enter PIG … Oink!

- High Level Languages for Map-Reduce
  - PIG
    - Developed by Yahoo
  - HIVE
    - Developed by Facebook
  - JAQL
    - Developed by IBM
- All of these languages provide similar functionality
- All of them allow users to plug in their own user defined functions (UDFs)

Lets get Practical – From Setup to Results

Setting up a Hadoop Cluster

- Minimum recommended configuration (4 Hosts)
  - 1 Host Dedicated for Management Services (Job Tracker, Name Node etc)
  - 3 Hosts as Slave nodes (Data Node , Task Trackers)

- Data nodes should have high capacity local disks attached.
  - This is where all your data is going to be

- How much total disk space?
  - Depends on input data to be processed
  - Effective Storage Space Recommended: Typically 3 times your input data size
  - Actual Storage Space: Effective Storage Space * 3 (replication level)

- Single node installation is fine for development/testing on very small data
  - Perhaps not the best for testing performance

- Installation instructions vary from provider to provider

Some cluster configuration parameters

- HDFS configuration parameters
  - Stored in hdfs-site.xml
  - Block size
  - Default replication count
- MapReduce configuration parameters
  - Stored In “mapred-site.xml”
  - Java heap size for mappers/reducers
  - Number of mappers/reducers per host
    - See http://wiki.apache.org/hadoop/HowManyMapsAndReduces
- *IMPORTANT*
  - Job Tracker URL: http://<masterhost>:50030
  - Name Node URL: http://<masterhost>:50070

Job Tracker Web Page (port 50030)

Quick Links

tdma1 Hadoop Map/Reduce Administration

State: RUNNING
Started: Tue Feb 25 15:21:13 EST 2014
Version: 1.1.1, r70b5aad8822a30795c1acdb966c97316387e1fc0
Compiled: Thu May 30 17:51:51 PDT 2013 by jenkins
Identifier: 201402251521
SafeMode: OFF

Cluster Summary (Heap Size is 298.31 MB/2.08 GB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	1021	4	0	0	0	0	128	64	48.00	0	0	0

Working with data

- Lets say you have 1 GB of data in your local filesystem (mydata.txt)
- Load into HDFS
  - hadoop fs –mkdir /path/mydirectory
  - hadoop fs –put mydata.txt /path/mydirectory
  - where /path/mydirectory is in HDFS
- List the file you just uploaded
  - hadoop fs –ls /path/mydirectory
- "hadoop fs" works similar to linux filesystem commands
  - However HDFS is not POSIX compliant.
  - It cannot be mounted as a regular filesystem

Writing your program .. see the simplicity!!

- JAQL program for running our scorer

```
// import the module
import scorerModule(*);

//get the total score
mytotal =
    read(lines("/path/mydirectory/myfile.txt"))
    -> transform score($)
    -> sum();
```

- PIG program for running our scorer

```
register 'myudfs.jar';

//read data
mydata = load '/path/mydirectory/mydata.txt'
  using TextLoader() as (line : chararray);

//score each line
mylines = foreach mydata generate myudfs.score(line) as scorecount;

//find total score
alllines = group mylines all;
mytotal = foreach alllines generate SUM(mylines.scorecount);
```

All languages provide similar functionality

- LOAD (various data formats)
- JOIN
- FOR-EACH
- GROUP
- SORT
- FILTER
- Pluggable UDFs

Hadoop Programming Tips

- Thinking at scale
  - Filter unwanted data earlier in the flow
  - Store intermediate data
  - Use “sequence” format for storing data.

- These are not iterative languages
  - i.e. No *for* or *while* loops

- Watch out for obvious bottlenecks
  - Single key for all mapper output will send data to one reducer
  - Too much data sent to a UDF will result in OOM errors

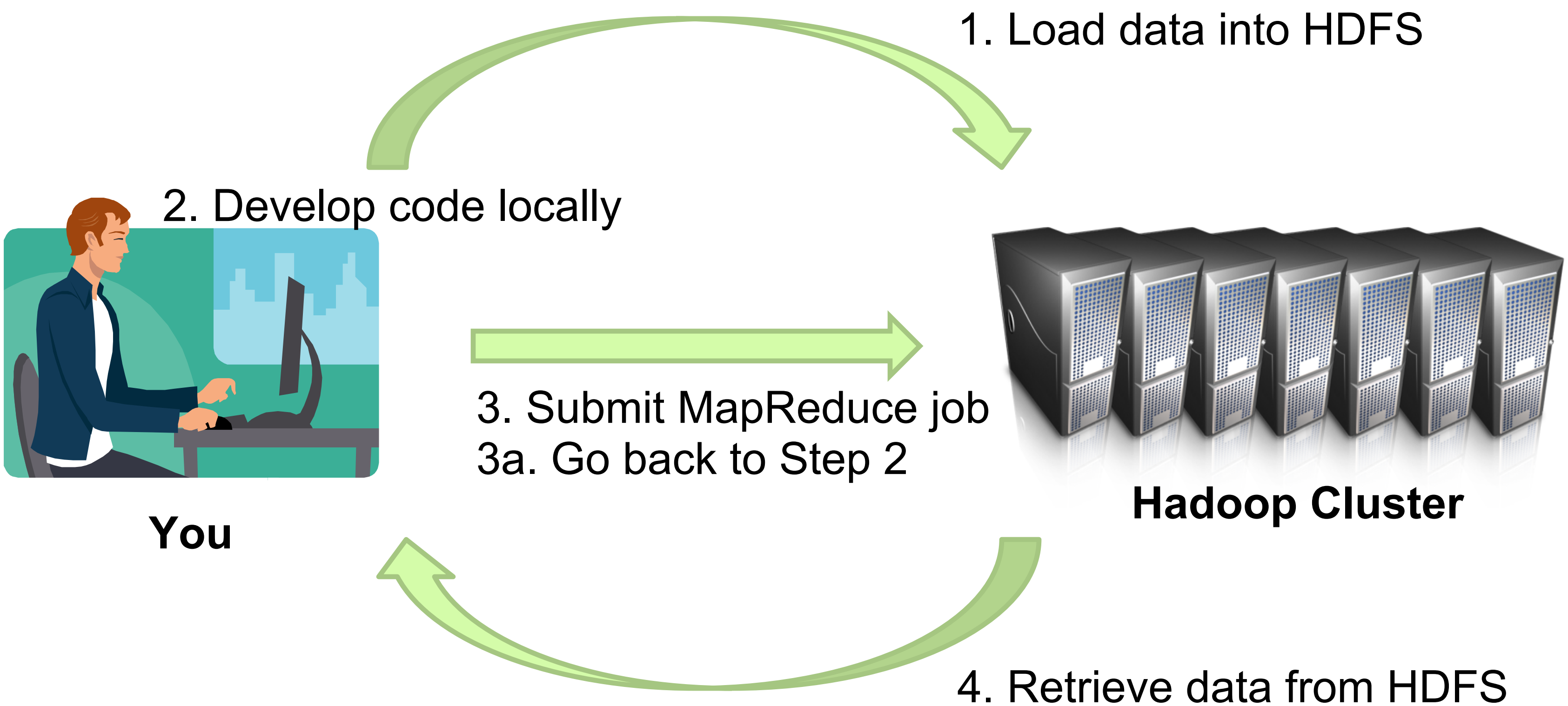
Submitting a Job

- Create and save your PIG script (myscript.pig)
- To deploy (pig command will be in your installation)
  - pig –f myscipt.pig
  - Command will complete once your job completes
- To check the status of your job
  - Use the Job Tracker URL (easiest) OR
  - hadoop job –list (will print all job ids)
  - hadoop job –status <jobid> (will print the job status)
- To get the results
  - hadoop fs –get /path/results.txt .

Anatomy of a Job

- MapReduce program in Hadoop = Hadoop job
  - Jobs are divided into map and reduce tasks
  - An instance of running a task is called a task attempt
  - Multiple jobs can be composed into a workflow
- Job submission process
  - Client (i.e., driver program) creates a job, configures it, and submits it to job tracker
  - JobClient computes input splits (on client end)
  - Job data (jar, configuration XML) are sent to JobTracker
  - JobTracker puts job data in shared location, enqueues tasks
  - TaskTrackers poll for tasks
  - Off to the races…

Hadoop Workflow



Uh Oh.. My Job Failed…Now what?

- First, take a deep breath
- Start small, start locally
- Strategies
  - Learn to use the webapp
  - Where does println go?
  - Don’t use println, use logging
  - Throw RuntimeExceptions
- Logs are most easily accessible via the Job Tracker URL

How about a Demo

Time for a Raise

- Finally you have mastered Hadoop Big Data
- Your applications are scaling.
  - You deserve a raise!!
- Boss
  - Can we query the data for specific entities?
  - How long will that take?
- Problem
  - Remember this is still sequential access
  - To find a specific entity, you still need to read the entire data set.
- What now?
  - How is this solved in traditional systems?

Databases

Enter - HBASE

- NOSQL Data Stores
- But that’s another discussion

Questions?

Resources

- Papers
  - Google File System, 2003
  - Google MapReduce, 2004
  - Google Bigtable, 2006
- URLS
  - Apache Hadoop: http://hadoop.apache.org
- Available Hadoop Distributions
  - Apache, IBM, Cloudera, Hortonworks

Other projects based on Hadoop

- HBase
- Hive
- PIG
- Spark
- Mahout