

# Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

Week 2: MapReduce Algorithm Design (1/2)  
January 12, 2016

Jimmy Lin  
David R. Cheriton School of Computer Science  
University of Waterloo

These slides are available at <http://lintool.github.io/bigdata-2016w/>

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States  
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details





An aerial photograph showing a vast, dense layer of white, fluffy clouds stretching across the horizon. The sky above is a clear, deep blue. The clouds are illuminated from the side, creating soft shadows and highlights that emphasize their texture and volume. The overall scene is serene and expansive.

# **Aside: Cloud Computing**

# The best thing since sliced bread?

- Before clouds...
  - Grids
  - Connection machine
  - Vector supercomputers
  - ...
- Cloud computing means many different things:
  - Big data
  - Rebranding of web 2.0
  - Utility computing
  - Everything as a service

# Rebranding of web 2.0

- Rich, interactive web applications
  - Clouds refer to the servers that run them
  - AJAX as the de facto standard (for better or worse)
  - Examples: Facebook, YouTube, Gmail, ...
- “The network is the computer”: take two
  - User data is stored “in the clouds”
  - Rise of the tablets, smartphones, etc. (“thin client”)
  - Browser *is* the OS

GENERAL  ELECTRIC

Rr13<sup>8</sup>/<sub>9</sub>



KILOWATTHOURS

CL 200

240V

3W

TYPE I-60-S  
SINGLE STATOR

CAT. NO.



FM 2S  
WATTHOUR METER

720X1G1

TA 30

Kh 7.2

60~

7  
P  
E  
R  
G  
E

397128

•44 617 187•

MADE IN U.S.A.

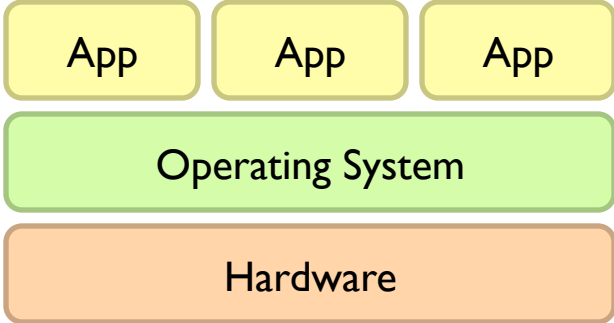
# Utility Computing

- What?
  - Computing resources as a metered service (“pay as you go”)
  - Ability to dynamically provision virtual machines
- Why?
  - Cost: capital vs. operating expenses
  - Scalability: “infinite” capacity
  - Elasticity: scale up or down on demand
- Does it make sense?
  - Benefits to cloud users
  - Business case for cloud providers

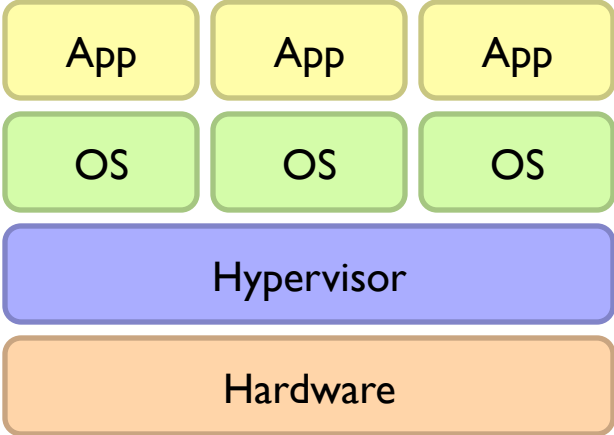
I think there is a world market for about five computers.



# Enabling Technology: Virtualization



Traditional Stack



Virtualized Stack

Today's buzzword: Docker



# Everything as a Service

- Utility computing = Infrastructure as a Service (IaaS)
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, Rackspace
- Platform as a Service (PaaS)
  - Give me nice API and take care of the maintenance, upgrades, ...
  - Example: Google App Engine, Altiscale
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail, Salesforce

# Who cares?

- A source of problems...
  - Cloud-based services *generate* big data
  - Clouds make it easier to start companies that *generate* big data
- As well as a solution...
  - Ability to provision analytics clusters on-demand in the cloud
  - Commoditization and democratization of big data capabilities

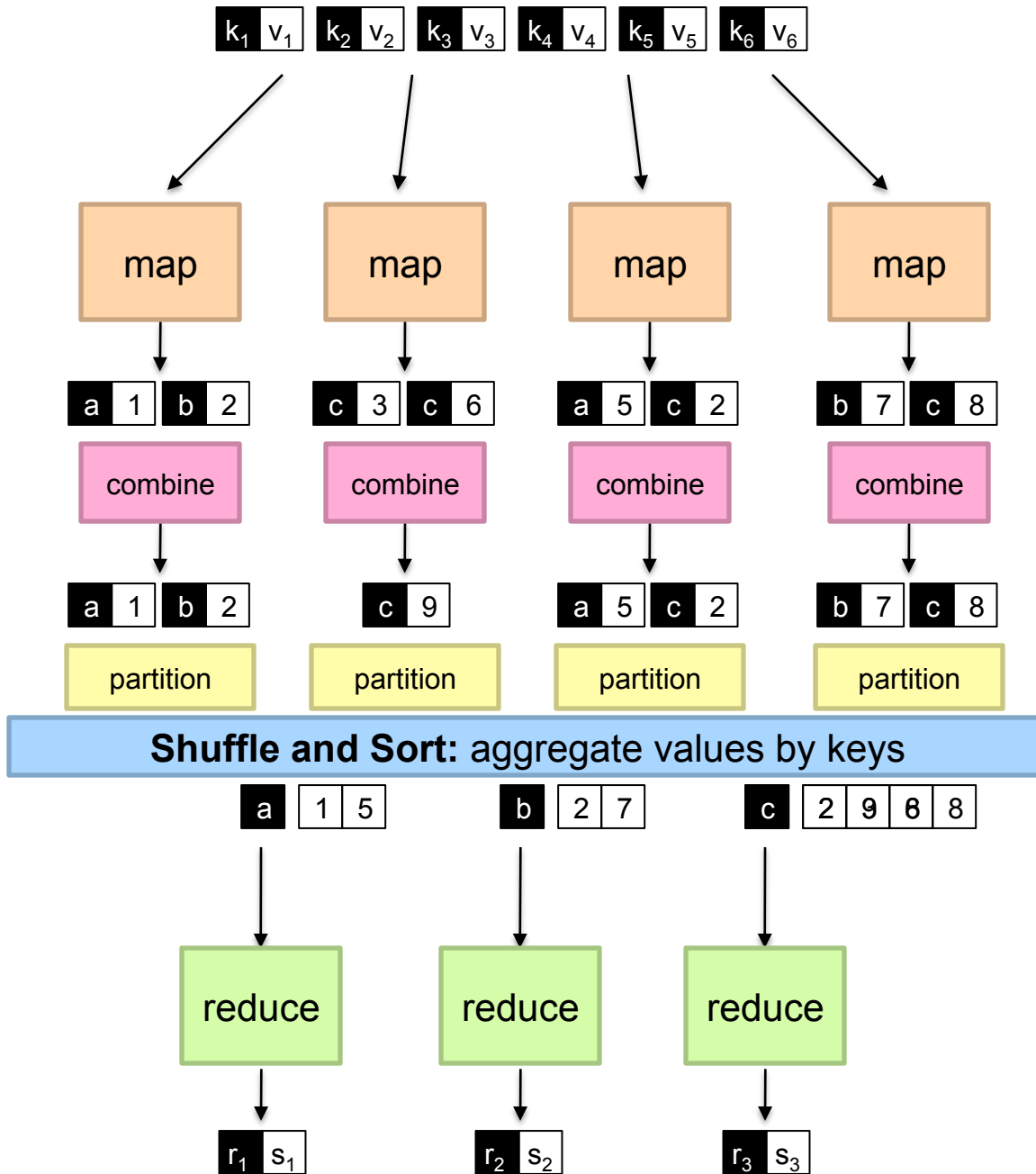
An aerial photograph showing a vast, dense layer of white, fluffy clouds stretching across the horizon. The clouds are illuminated from the side, creating soft shadows and highlights. The sky above is a clear, deep blue. The overall scene is a beautiful, high-angle view of a cloud deck.

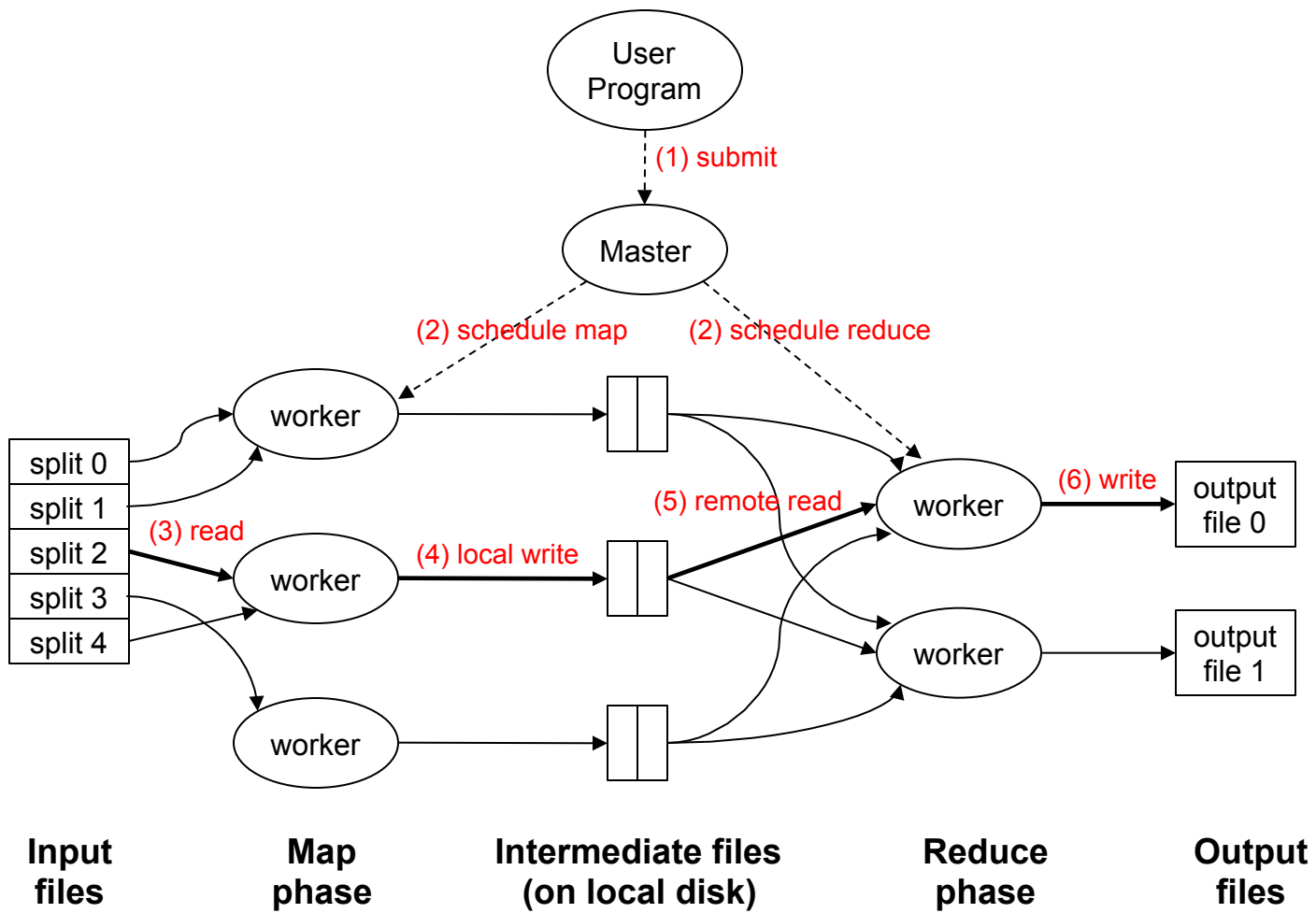
**So, what *is* the cloud?**

# What is the Matrix?

An aerial photograph of a large datacenter facility during sunset. The sun is low on the horizon, casting a warm orange glow over the scene. The datacenter consists of several large, white, rectangular buildings with flat roofs, arranged in a grid-like pattern. A prominent feature is a large, open-air area with rows of white, cylindrical cooling towers or condensers. The facility is surrounded by green fields and some smaller buildings in the distance. The overall atmosphere is serene and industrial.

**The datacenter *is* the computer!**







**Aside, what about Spark?**



# MapReduce API\*

- Mapper<K<sub>in</sub>, V<sub>in</sub>, K<sub>out</sub>, V<sub>out</sub>>
  - void setup(Mapper.Context context)  
Called once at the beginning of the task
  - void map(K<sub>in</sub> key, V<sub>in</sub> value, Mapper.Context context)  
Called once for each key/value pair in the input split
  - void cleanup(Mapper.Context context)  
Called once at the end of the task
- Reducer<K<sub>in</sub>, V<sub>in</sub>, K<sub>out</sub>, V<sub>out</sub>>/Combiner<K<sub>in</sub>, V<sub>in</sub>, K<sub>out</sub>, V<sub>out</sub>>
  - void setup(Reducer.Context context)  
Called once at the start of the task
  - void reduce(K<sub>in</sub> key, Iterable<V<sub>in</sub>> values, Reducer.Context context)  
Called once for each key
  - void cleanup(Reducer.Context context)  
Called once at the end of the task

\*Note that there are two versions of the API!

# MapReduce API\*

## ○ Partitioner

- `int getPartition(K key, V value, int numPartitions)`  
Get the partition number given total number of partitions

## ○ Job

- Represents a packaged Hadoop job for submission to cluster
- Need to specify input and output paths
- Need to specify input and output formats
- Need to specify mapper, reducer, combiner, partitioner classes
- Need to specify intermediate/final key/value classes
- Need to specify number of reducers (but not mappers, why?)
- Don't depend of defaults!

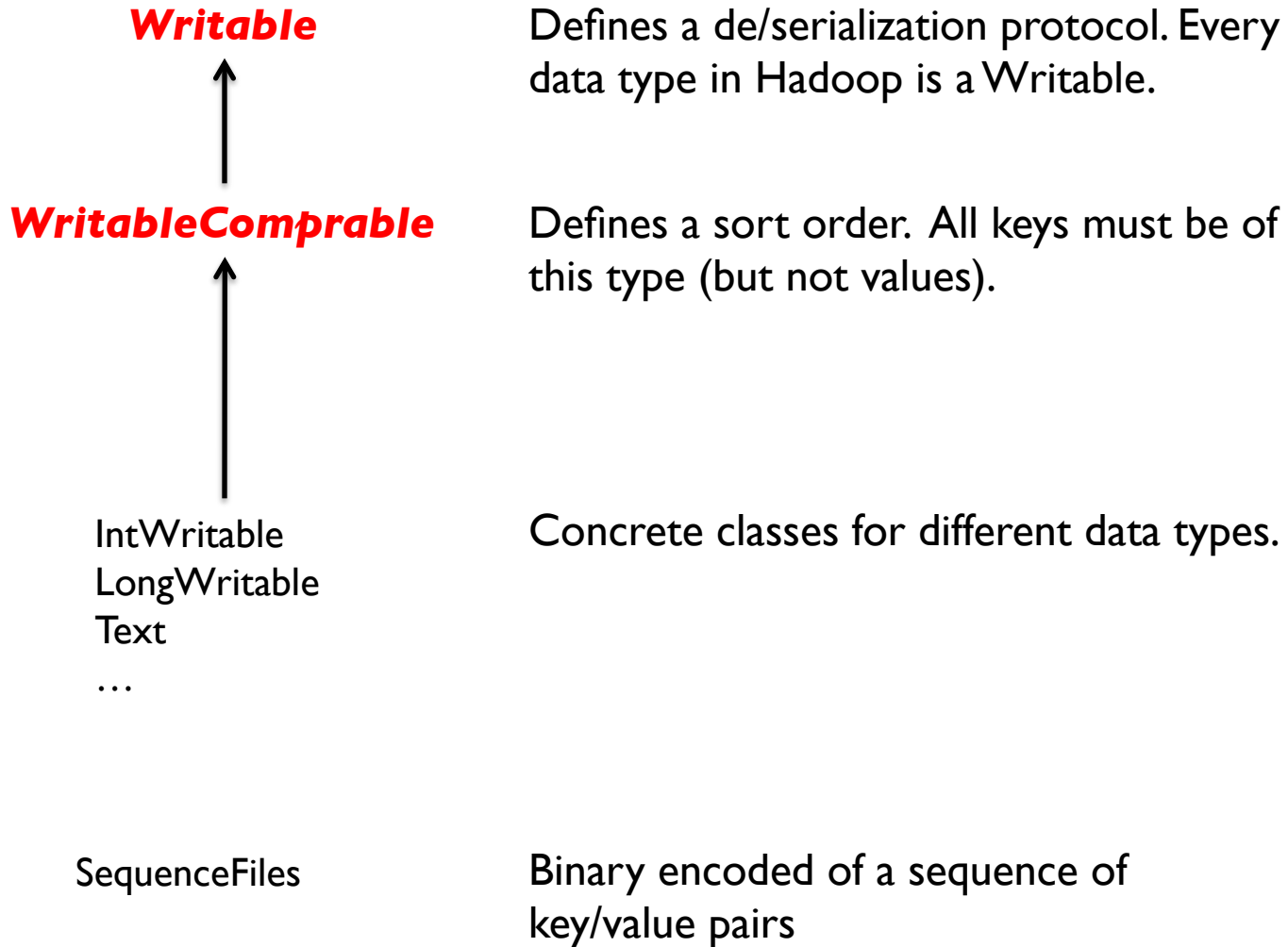
\*Note that there are two versions of the API!

# A tale of two packages...

org.apache.hadoop.mapreduce  
org.apache.hadoop.mapred



# Data Types in Hadoop: Keys and Values



# “Hello World”: Word Count

## **Map(String docid, String text):**

for each word  $w$  in text:

Emit( $w$ , 1);

## **Reduce(String term, Iterator<Int> values):**

int sum = 0;

for each  $v$  in values:

sum +=  $v$ ;

Emit(term, sum);

# “Hello World”: Word Count

```
private static class MyMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable ONE = new IntWritable(1);
    private final static Text WORD = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String line = ((Text) value).toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            WORD.set(itr.nextToken());
            context.write(WORD, ONE);
        }
    }
}
```

# “Hello World”: Word Count

```
private static class MyReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    private final static IntWritable SUM = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        Iterator<IntWritable> iter = values.iterator();
        int sum = 0;
        while (iter.hasNext()) {
            sum += iter.next().get();
        }
        SUM.set(sum);
        context.write(key, SUM);
    }
}
```

# Three Gotchas

- Avoid object creation if possible
  - Reuse Writable objects, change the payload
- Execution framework reuses value object in reducer
- Passing parameters via class statics



# Getting Data to Mappers and Reducers

- Configuration parameters
  - Directly in the Job object for parameters
- “Side data”
  - DistributedCache
  - Mappers/reducers read from HDFS in setup method

# Complex Data Types in Hadoop

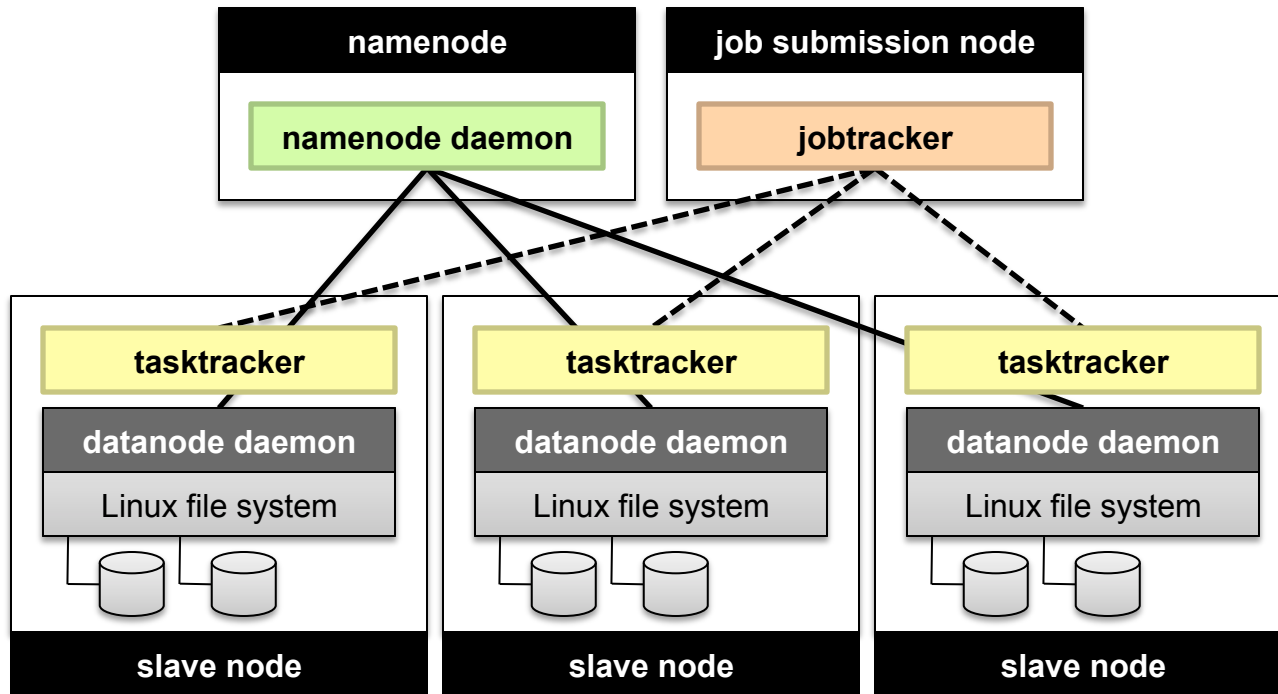
- How do you implement complex data types?
- The easiest way:
  - Encoded it as Text, e.g., (a, b) = “a:b”
  - Use regular expressions to parse and extract data
  - Works, but janky
- The hard way:
  - Define a custom implementation of Writable(Comparable)
  - Must implement: readFields, write, (compareTo)
  - Computationally efficient, but slow for rapid prototyping
  - Implement WritableComparator hook for performance
- Somewhere in the middle:
  - Bspin (via [lin.tl](http://lin.tl)) offers JSON support and lots of useful Hadoop types

# Basic Cluster Components\*

- One of each:
  - Namenode (NN): master node for HDFS
  - Jobtracker (JT): master node for job submission
- Set of each per slave machine:
  - Tasktracker (TT): contains multiple task slots
  - Datanode (DN): serves HDFS data blocks

\* Not quite... leaving aside YARN for now

# Putting everything together...

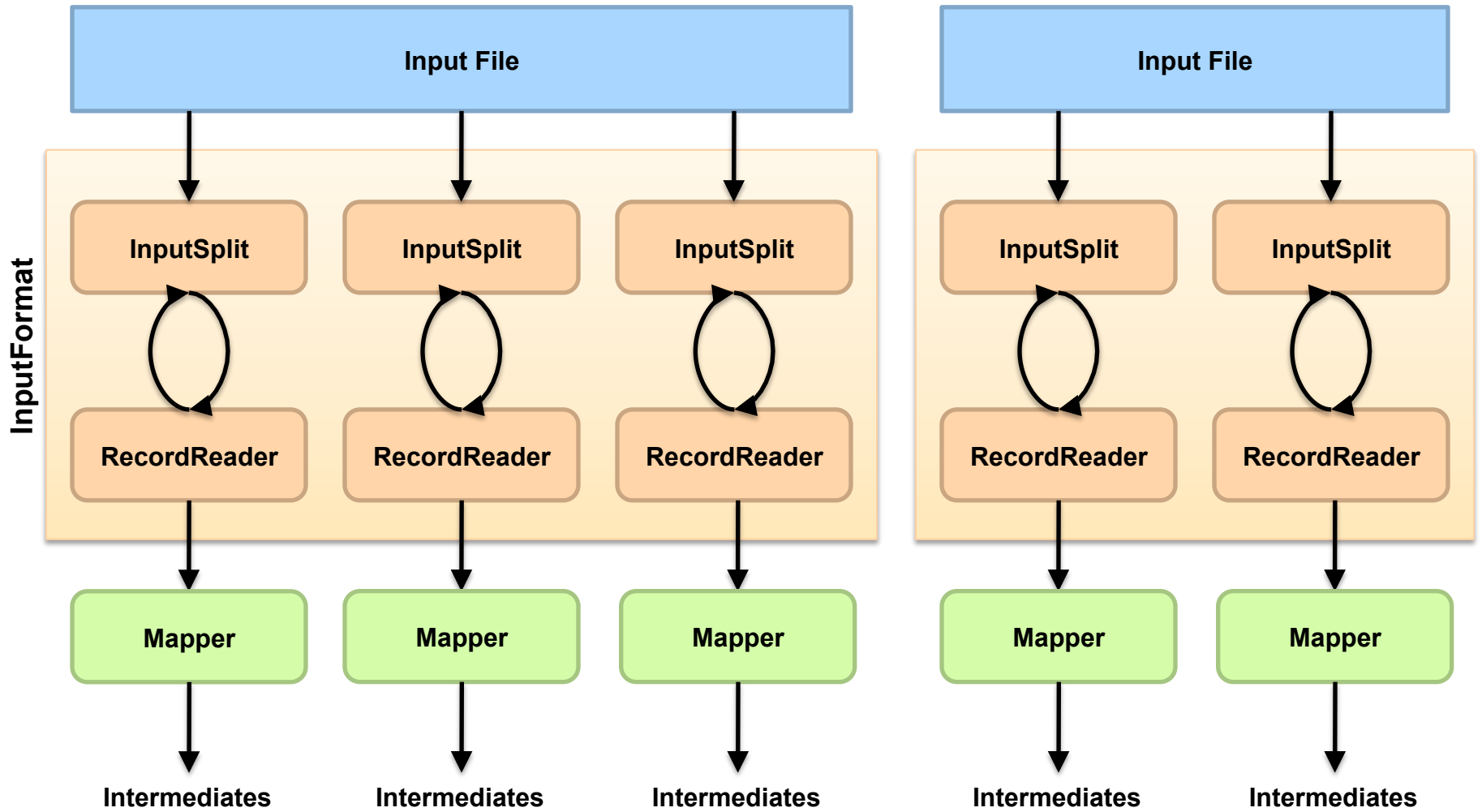


# Anatomy of a Job

- 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 (occupies a slot)
  - Multiple jobs can be composed into a workflow
- Job submission:
  - Client (i.e., driver program) creates a job, configures it, and submits it to jobtracker
  - That's it! The Hadoop cluster takes over...

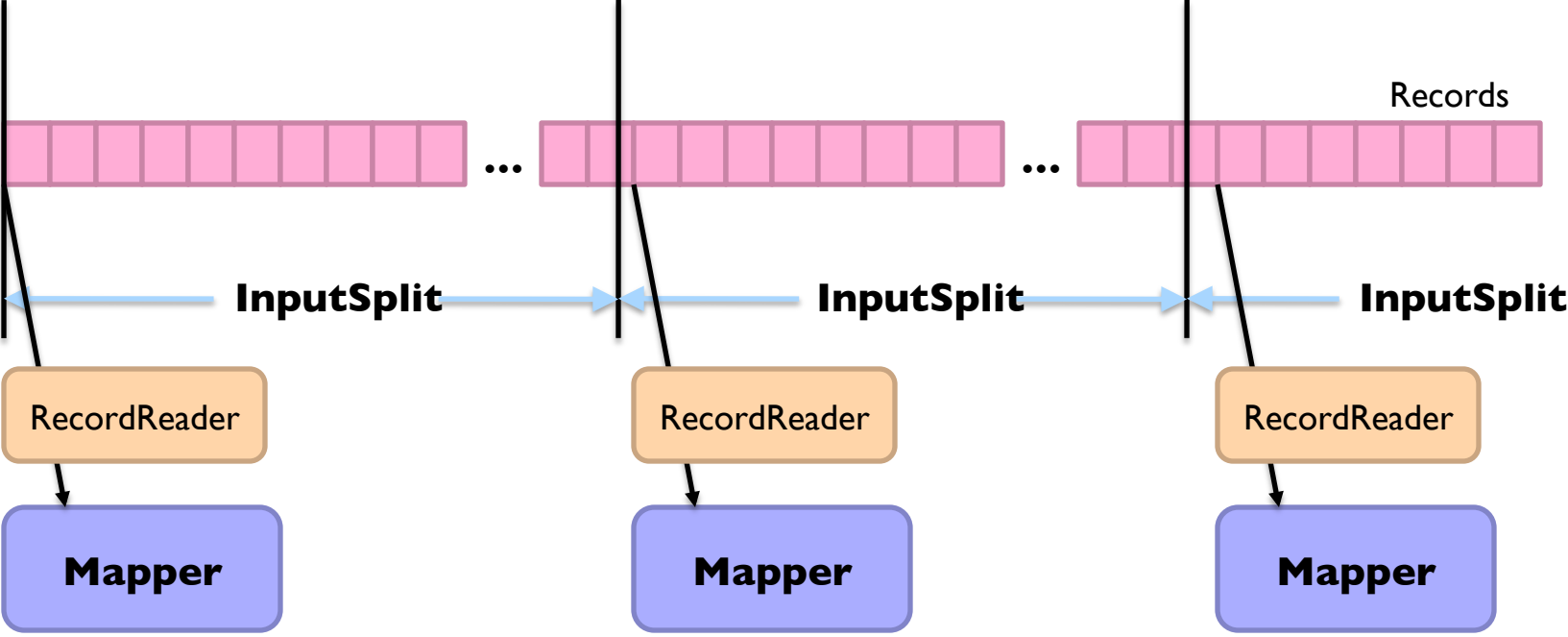
# Anatomy of a Job

- Behind the scenes:
  - Input splits are computed (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...

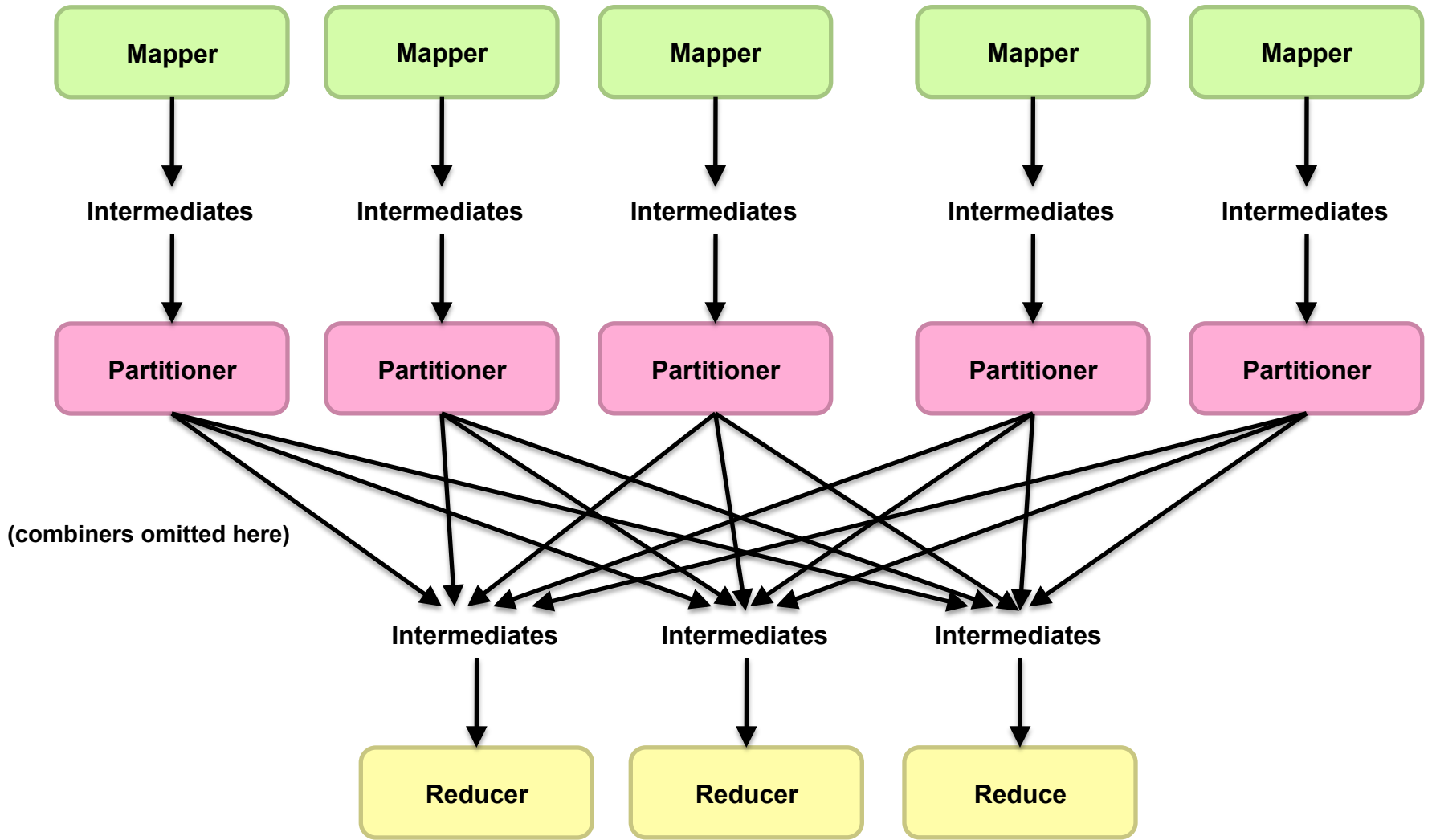


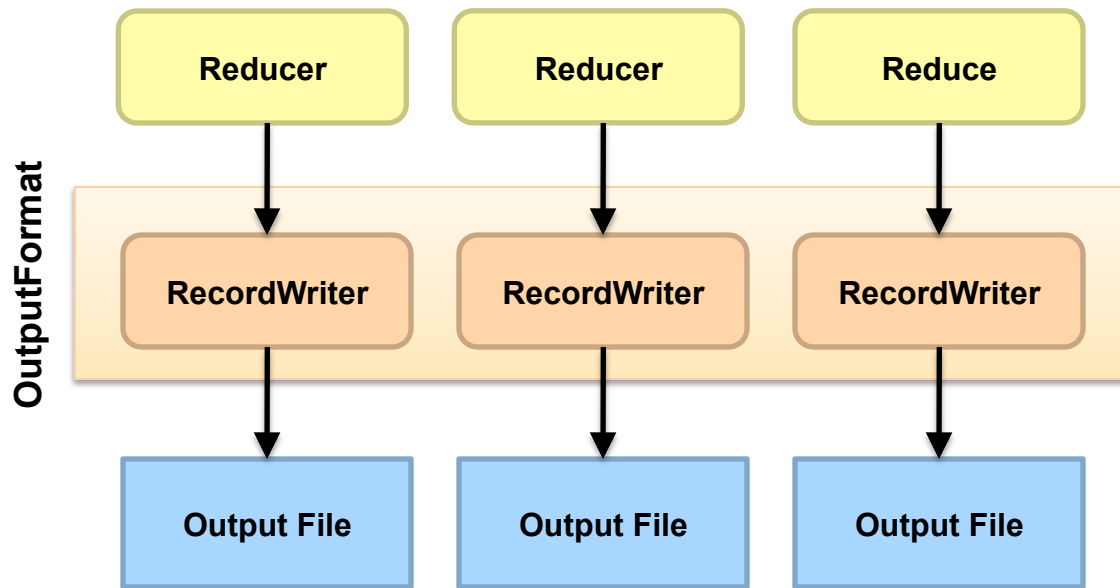
Source: redrawn from a slide by Cloudera, cc-licensed

**Client**









# Input and Output

- InputFormat:

- TextInputFormat
- KeyValueTextInputFormat
- SequenceFileInputFormat
- ...

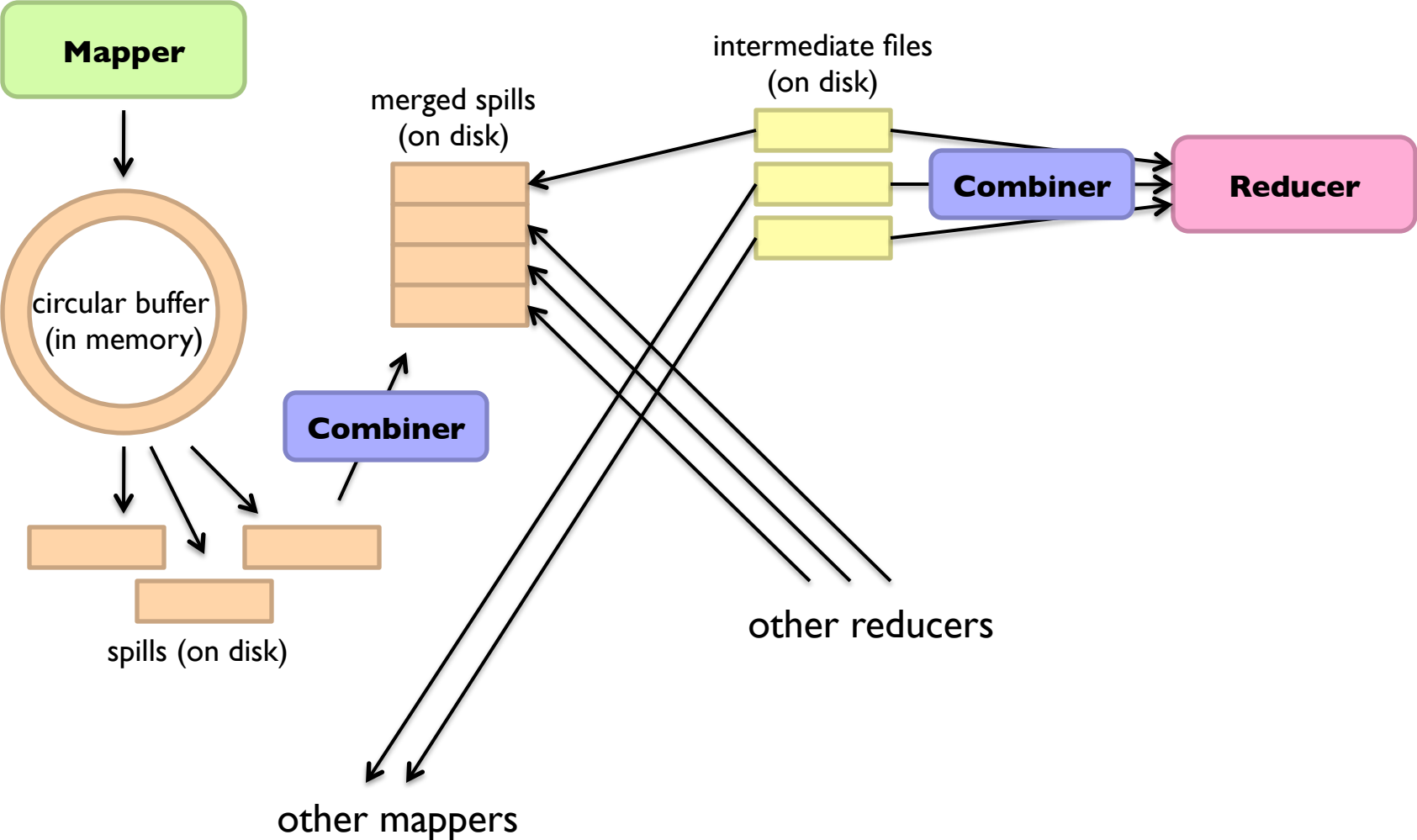
- OutputFormat:

- TextOutputFormat
- SequenceFileOutputFormat
- ...

# Shuffle and Sort in MapReduce

- Probably the most complex aspect of MapReduce execution
- 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 during the merges
- 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 during the merges
  - Final merge pass goes directly into reducer

# Shuffle and Sort



# Hadoop Workflow



**You**



**Submit node**  
(workspace)



**Hadoop Cluster**

Getting data in?

Writing code?

Getting data out?

# Debugging Hadoop

- First, take a deep breath
- Start small, start locally
- Build incrementally





# Code Execution Environments

- Different ways to run code:
  - Local (standalone) mode
  - Pseudo-distributed mode
  - Fully-distributed mode
- Learn what's good for what

# Hadoop Debugging Strategies

- Good ol' `System.out.println`
  - Learn to use the webapp to access logs
  - Logging preferred over `System.out.println`
  - Be careful how much you log!
- Fail on success
  - Throw `RuntimeExceptions` and capture state
- Programming is still programming
  - Use Hadoop as the “glue”
  - Implement core functionality outside mappers and reducers
  - Independently test (e.g., unit testing)
  - Compose (tested) components in mappers and reducers



# Questions?