



UNIVERSITY OF
WATERLOO

Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2017)

Week 2: MapReduce Algorithm Design (1/2)

January 10, 2017

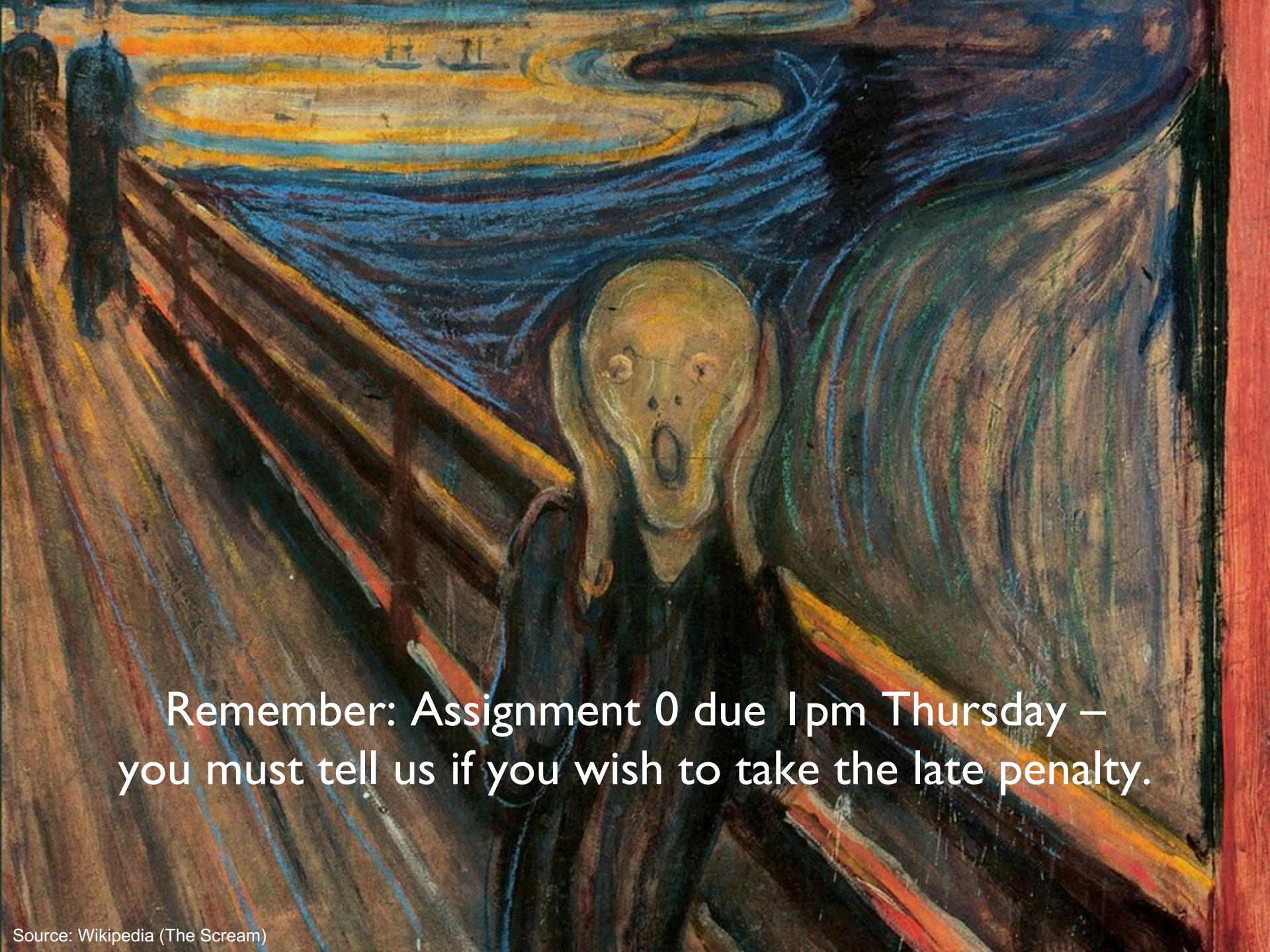
Jimmy Lin

David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at <http://lintool.github.io/bigdata-2017w/>



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



Remember: Assignment 0 due 1pm Thursday –
you must tell us if you wish to take the late penalty.



**KEEP
CALM
AND
RTFM**

RTFM





RTFM

YOU MUST

An aerial photograph of a large datacenter complex during sunset. The sky is a vibrant orange and yellow. In the foreground, there are several large industrial buildings, parking lots, and rows of white shipping containers. A major highway runs through the middle ground. The background shows a vast, green, agricultural landscape with rolling hills under the setting sun.

The datacenter *is* the computer!

An aerial photograph showing a vast expanse of white and grey clouds against a clear blue sky. The clouds are dense and layered, creating a textured pattern across the frame. In the lower right quadrant, a dark, mountainous landmass is visible, partially obscured by clouds.

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

Javascript! (ugh)

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 clients”)

Browser is the OS

GENERAL  ELECTRIC

R 13%

8 9 0 1 2 2 1 0 9 8 8 9 0 2 1 0 9 8 8 9 0 1 2
7 6 5 4 3 3 4 5 7 7 6 5 4 3 3 4 5 6 7 7 6 5 3
K I L O W A T T H O U R S

CL 200

TYPE I-60-S
SINGLE STATOR  FM 2S
WATTHOUR METER

TA 30

240V

3W

CAT. NO.

720X1G1

Kh 7.2
60~

397128

• 44 617 187 •

MADE IN U.S.A.

P
G
E
and

Utility Computing

What?

Computing resources as a metered service (“pay as you go”)

Why?

Cost: capital vs. operating expenses

Scalability: “infinite” capacity

Elasticity: scale up or down on demand

Does it make sense?

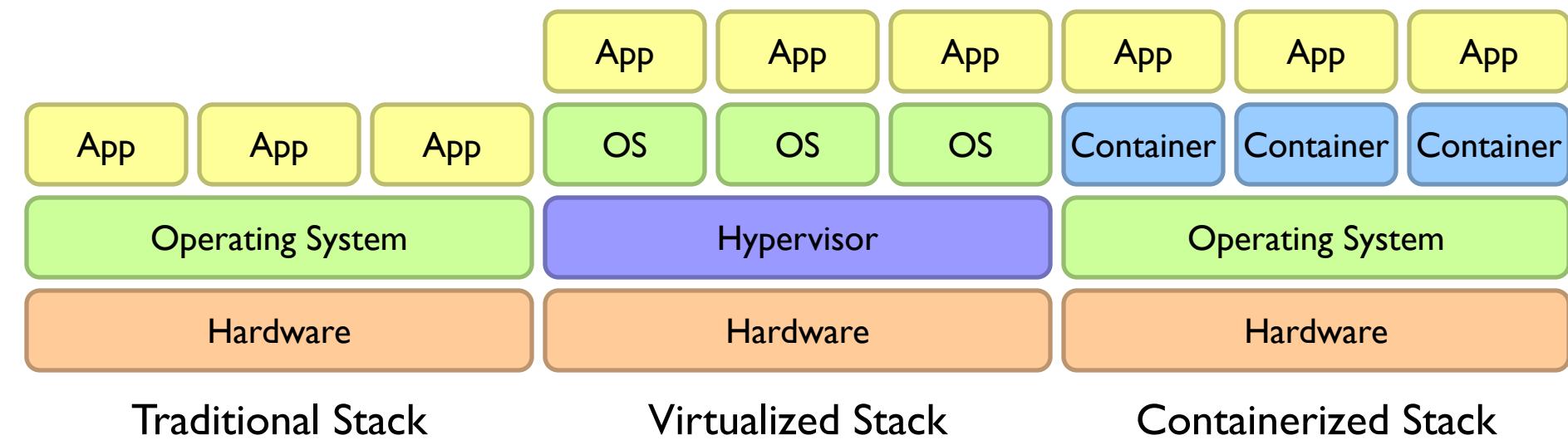
Benefits to cloud users

Business case for cloud providers

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



Evolution of the Stack



Everything as a Service

Infrastructure as a Service (IaaS)

Why buy machines when you rent them instead?

Examples: Amazon EC2, Microsoft Azure, Google Compute

Platform as a Service (PaaS)

Give me nice platform and take care of maintenance, upgrades, ...

Example: Google App Engine, Altiscale

Software as a Service (SaaS)

Just run the application for me!

Example: Gmail, Salesforce

Everything as a Service

Database as a Service

Run a database for me

Examples: Amazon RDS, Microsoft Azure SQL

Container as a Service

Run this container for me

Example: Amazon EC2 Container Service, Google Container Engine

Function as a Service

Run this function for me

Example: Amazon Lambda, Google Cloud Functions

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...

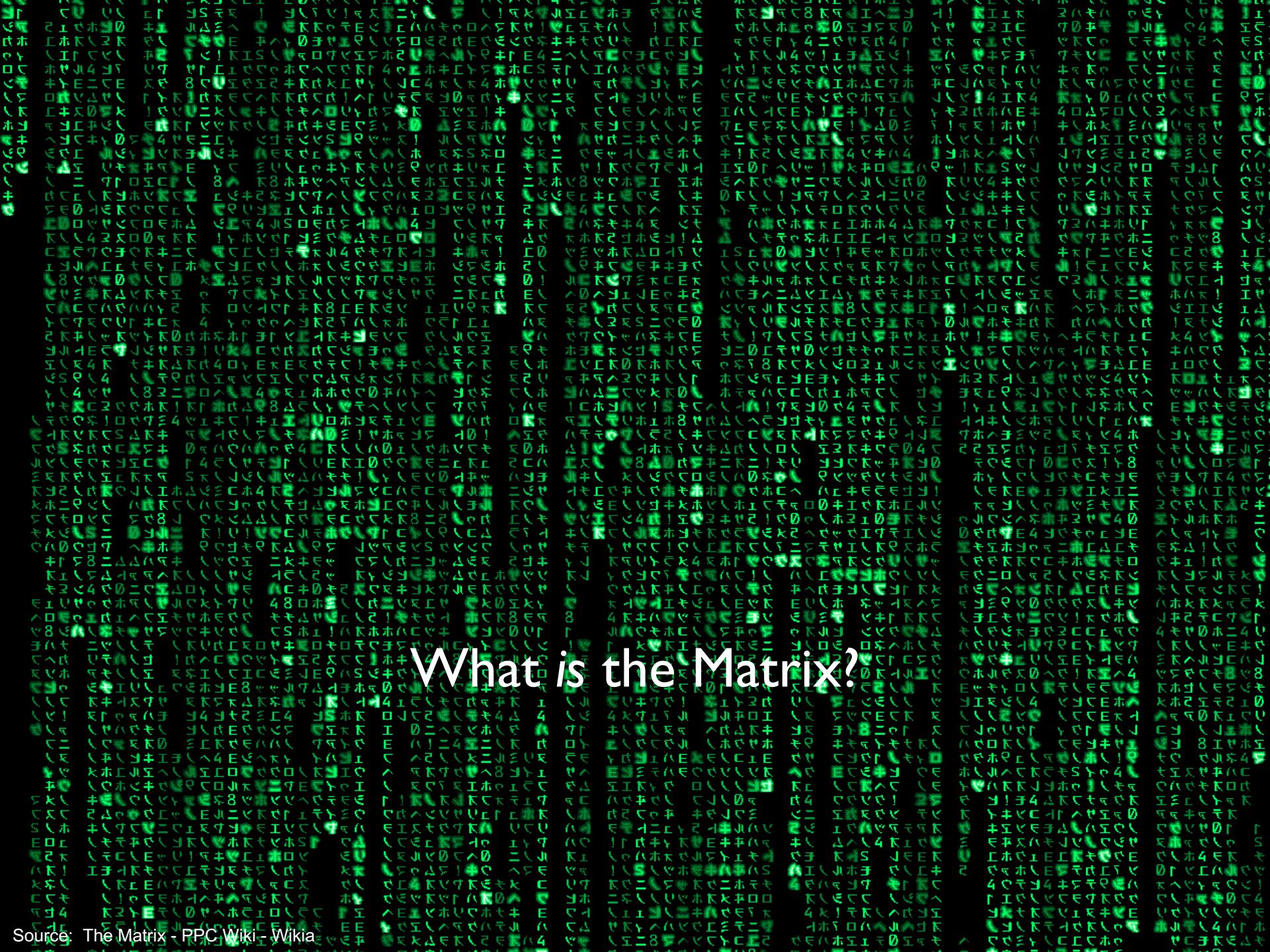
Ability to provision clusters on-demand in the cloud

Commoditization and democratization of big data capabilities

An aerial photograph showing a vast expanse of white and grey clouds against a clear blue sky. The clouds are layered and textured, creating a sense of depth. In the lower right corner, a dark, mountainous landmass is visible, partially obscured by clouds.

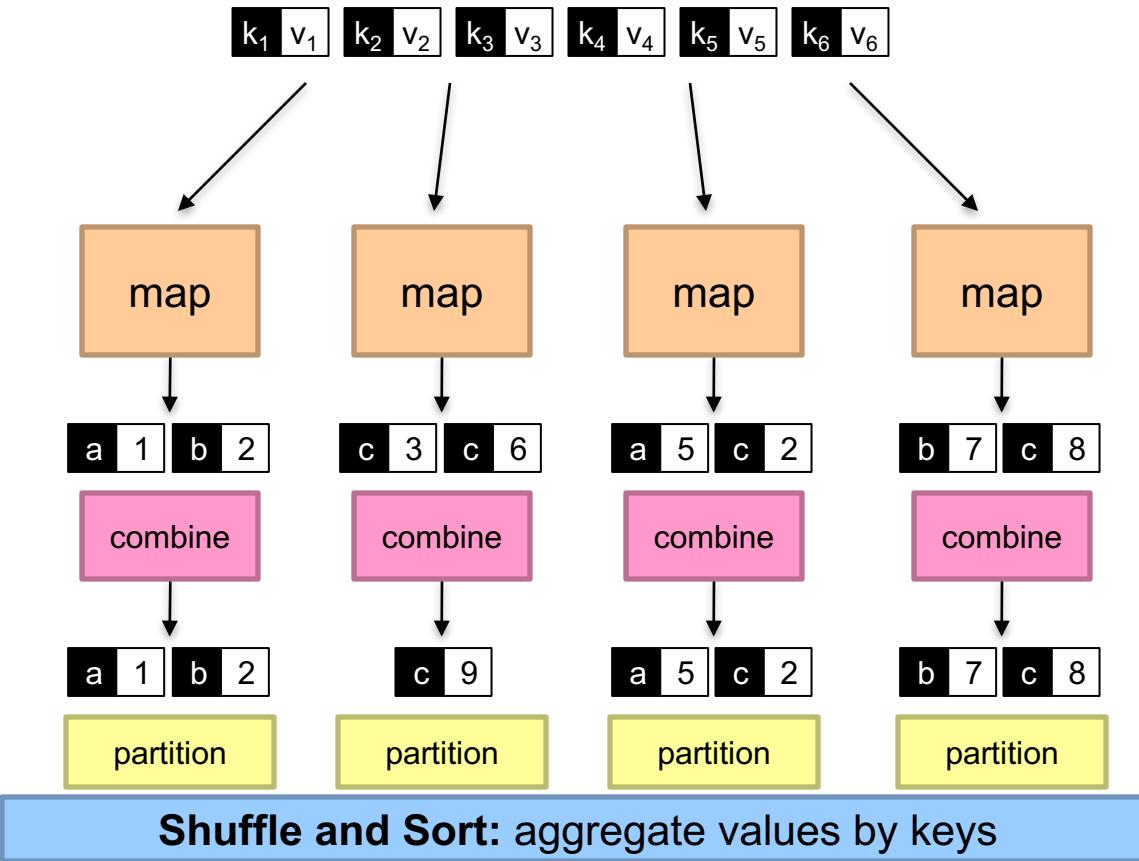
So, what *is* the cloud?

What is the Matrix?

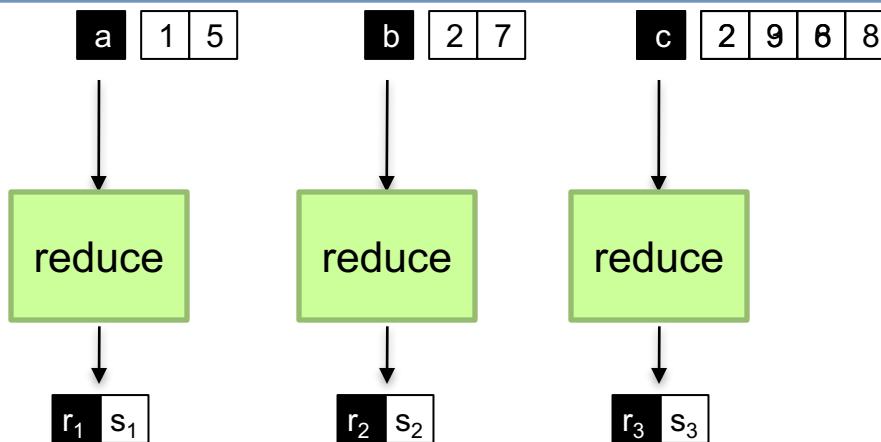


An aerial photograph of a large datacenter complex during sunset. The sky is a vibrant orange and yellow. In the foreground, there are several large industrial buildings, parking lots, and rows of white shipping containers. A major highway runs through the middle ground. The background shows a vast, green, agricultural landscape with rolling hills under the setting sun.

The datacenter *is* the computer!



Shuffle and Sort: aggregate values by keys





So you want to drive the elephant!



So you want to drive the elephant!
Aside, what about Spark?

A tale of two packages...

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



MapReduce API*

`Mapper<Kin, Vin, Kout, Vout>`

`void setup(Mapper.Context context)`

Called once at the start of the task

`void map(Kin key, Vin 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<Kin, Vin, Kout, Vout>/Combiner<Kin, Vin, Kout, Vout>`

`void setup(Reducer.Context context)`

Called once at the start of the task

`void reduce(Kin key, Iterable<Vin> values, Reducer.Context context)`

Called once for each key

`void cleanup(Reducer.Context context)`

Called once at the end of the task

MapReduce API*

Partitioner<K, V>

int getPartition(K key, V value, int numPartitions)

Returns 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

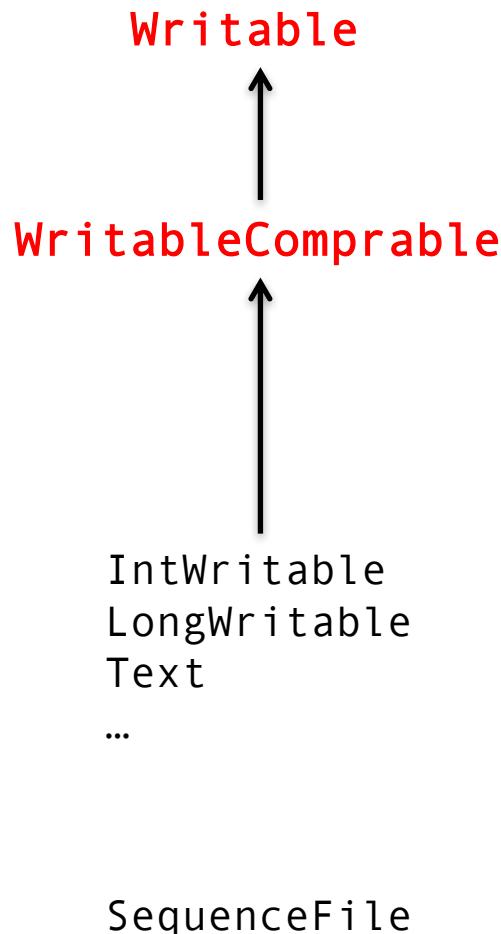
Need to specify mapper, reducer, combiner, partitioner classes

Need to specify intermediate/final key/value classes

Need to specify number of reducers (but not mappers, why?)

Don't depend on defaults!

Data Types in Hadoop: Keys and Values



`Writable`
Defines a de/serialization protocol.
Every data type in Hadoop is a `Writable`.

`WritableComparable`
Defines a sort order.
All keys must be of this type (but not values).

`IntWritable`
`LongWritable`
`Text`
...
`SequenceFile`
Concrete classes for different data types.
Note that these are container objects.

Binary-encoded sequence of key/value pairs.

“Hello World” MapReduce: 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);
```

Word Count Mapper

```
private static final 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 {
        for (String word : Tokenizer.tokenize(value.toString())) {
            WORD.set(word);
            context.write(WORD, ONE);
        }
    }
}
```

Word Count Reducer

```
private static final 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

Execution framework reuses value object in reducer

Passing parameters via class statics doesn't work!

Getting Data to Mappers and Reducers

Configuration parameters

Pass in via Job configuration object

“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:

Bespin (via lin.tl) offers various building blocks

Basic Cluster Components*

On the master:

Namenode (NN): master node for HDFS

Jobtracker (JT): master node for job submission

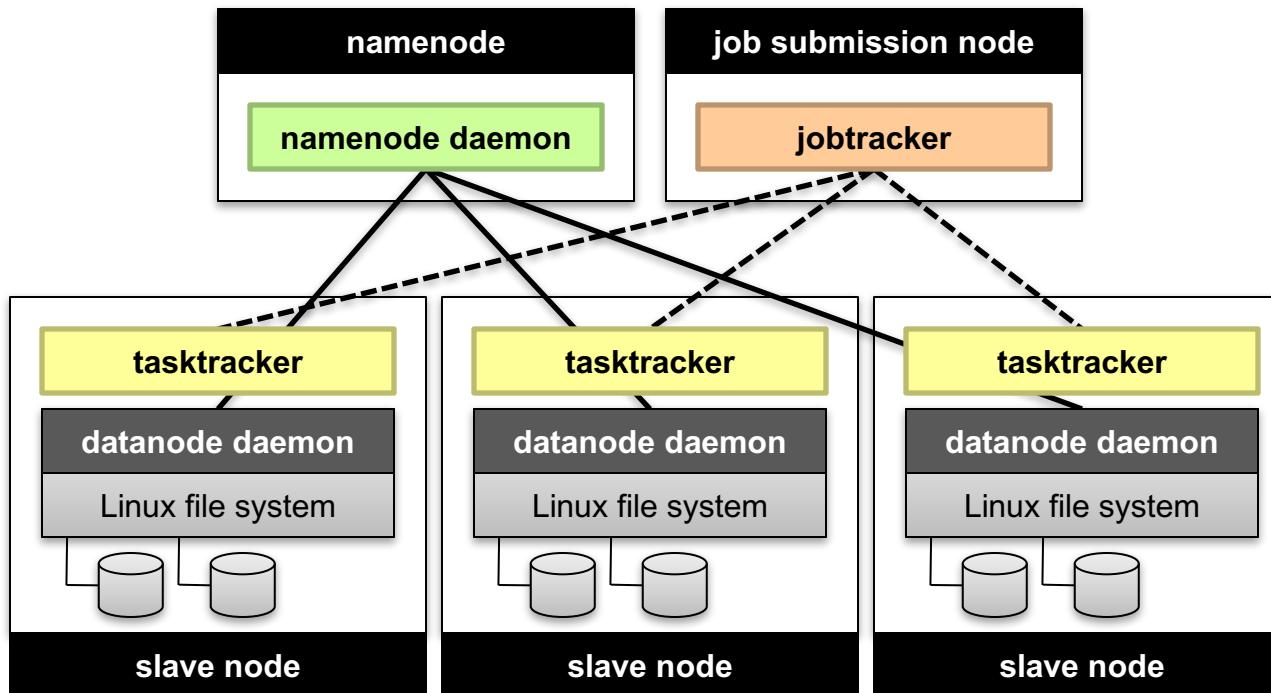
On each of the slave machines:

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

Hadoop MapReduce program = Hadoop job

Jobs are divided into map and reduce tasks

An instance of a running task is called a task attempt

Each task occupies a slot on the tasktracker

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:

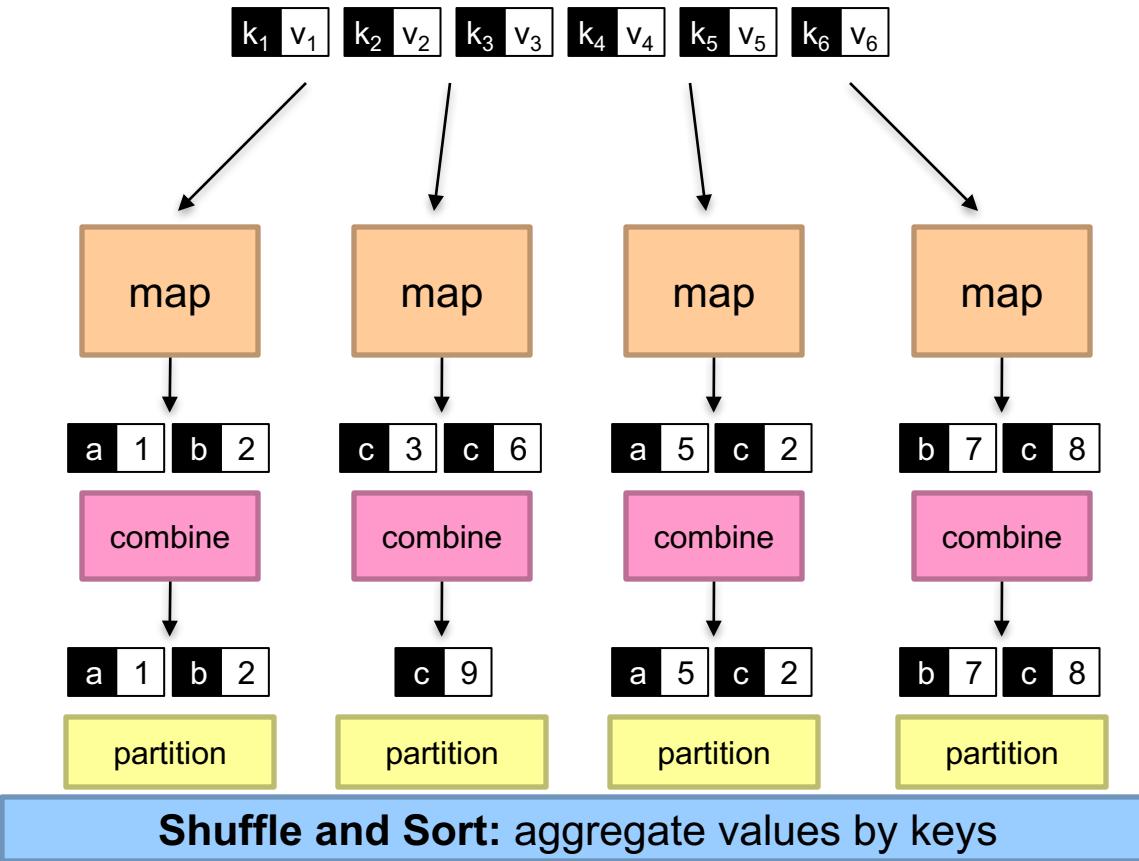
Input splits are computed (on client end)

Job data (jar, configuration XML) are sent to jobtracker

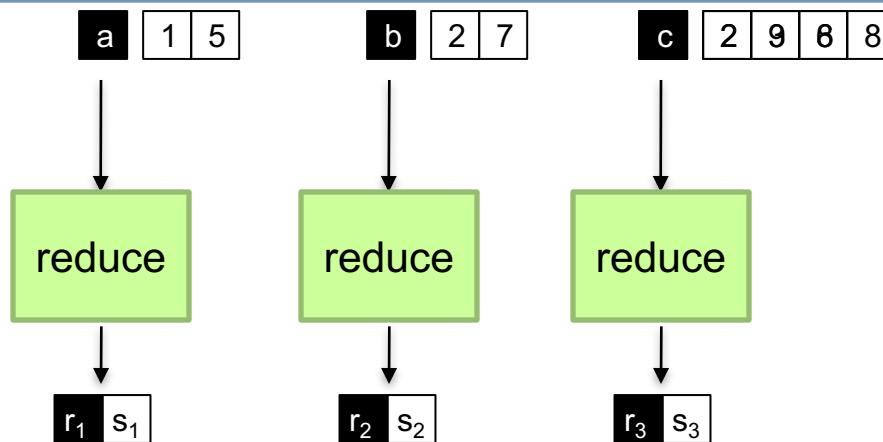
Jobtracker puts job data in shared location, enqueues tasks

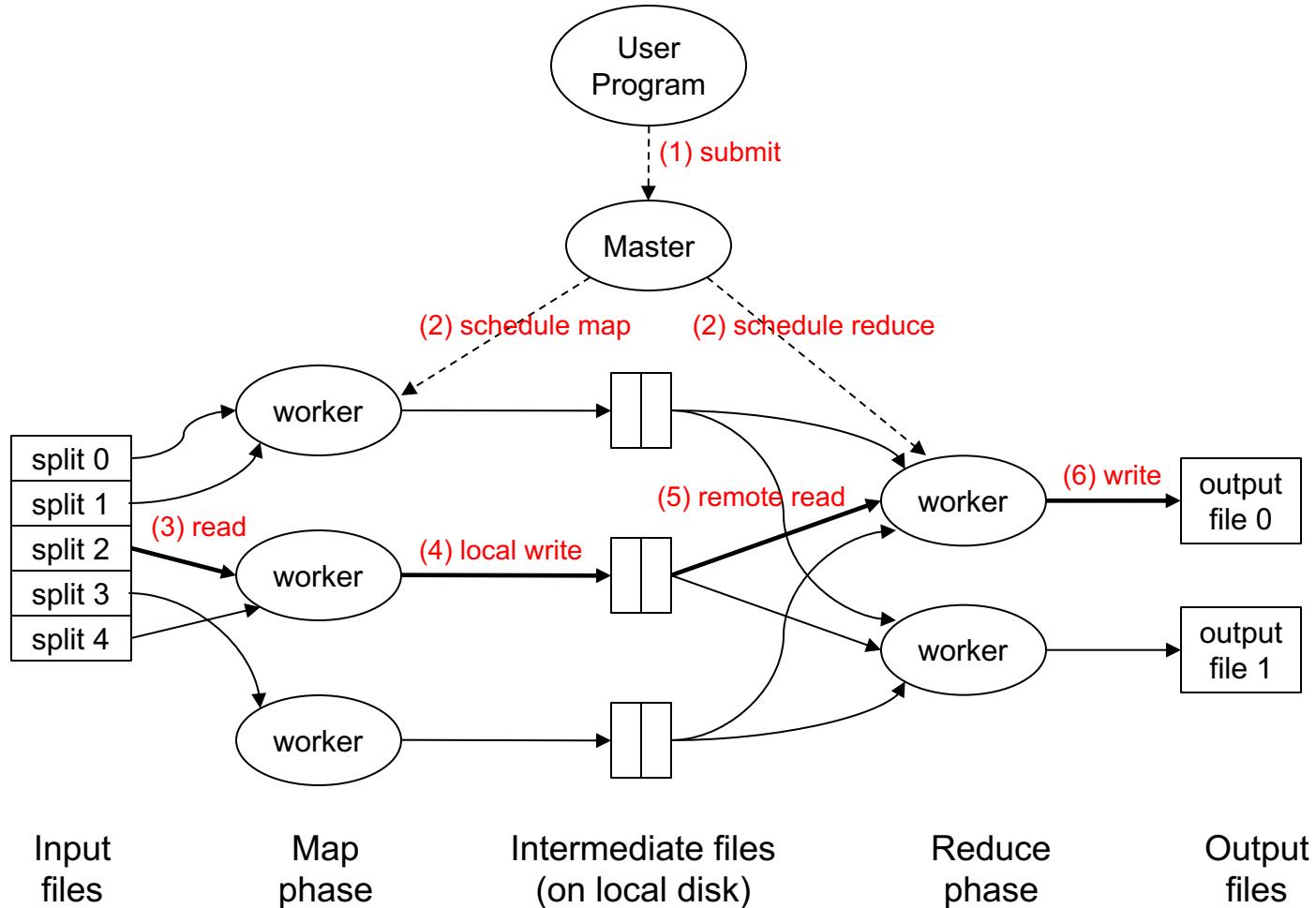
Tasktrackers poll for tasks

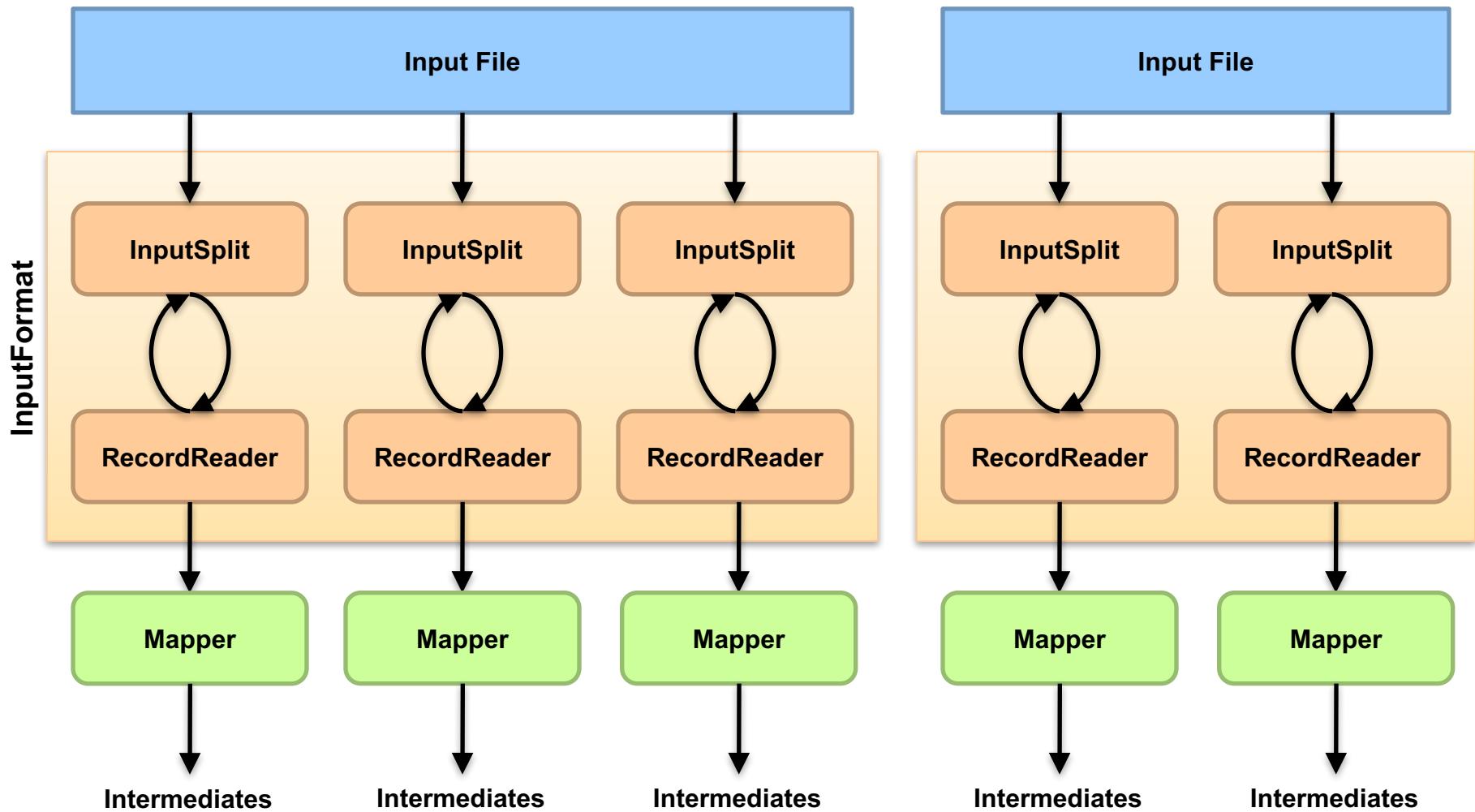
Off to the races...

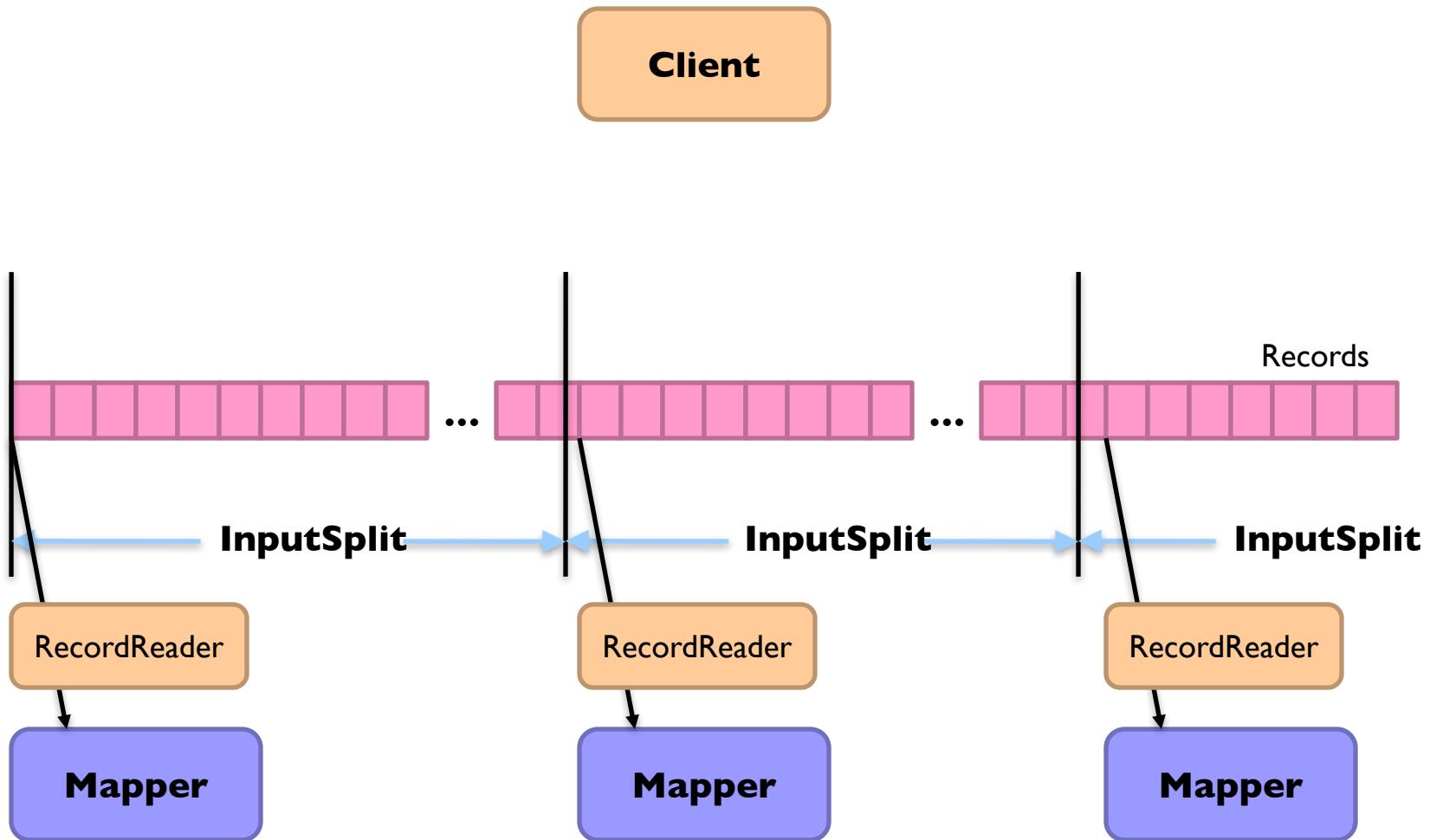


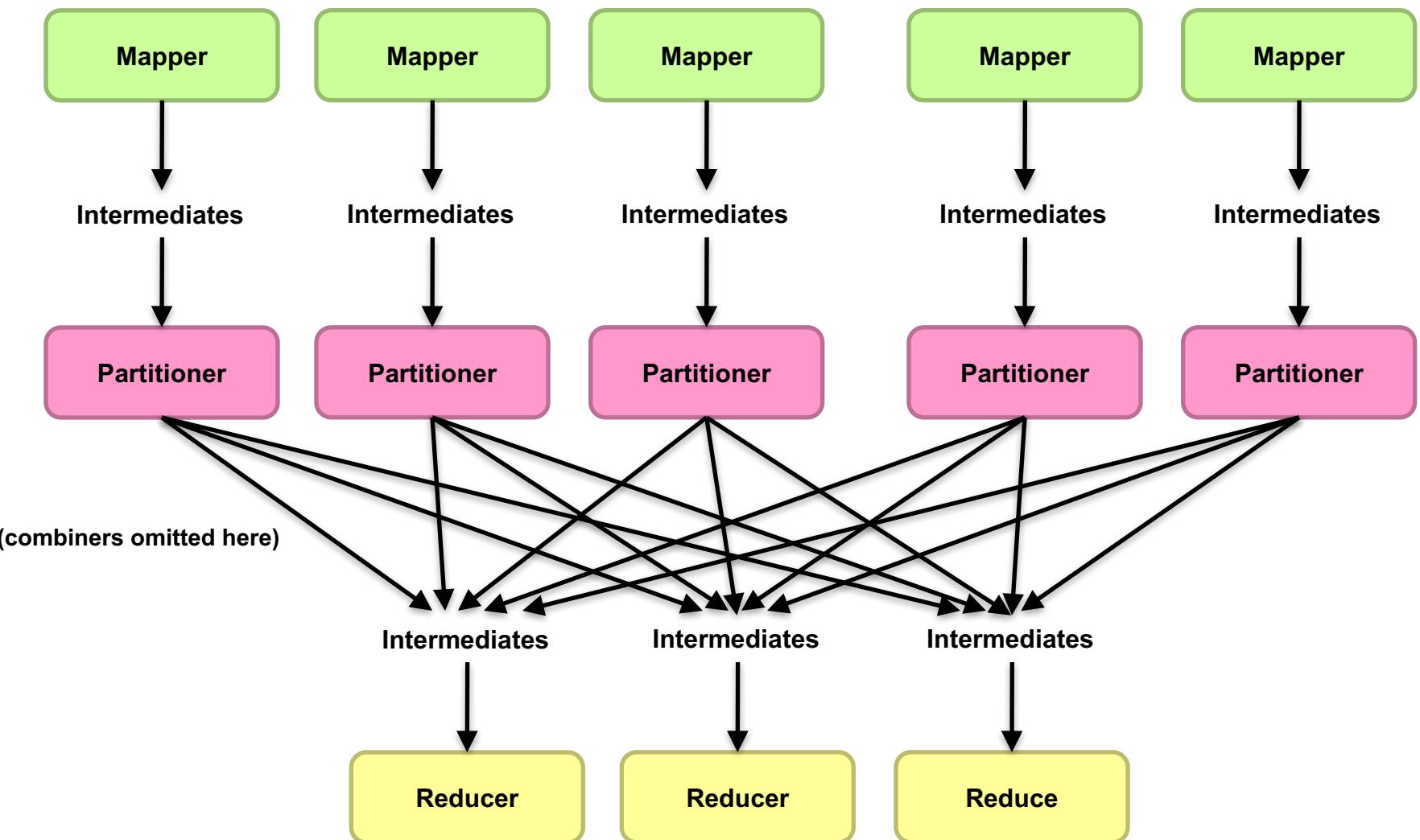
Shuffle and Sort: aggregate values by keys

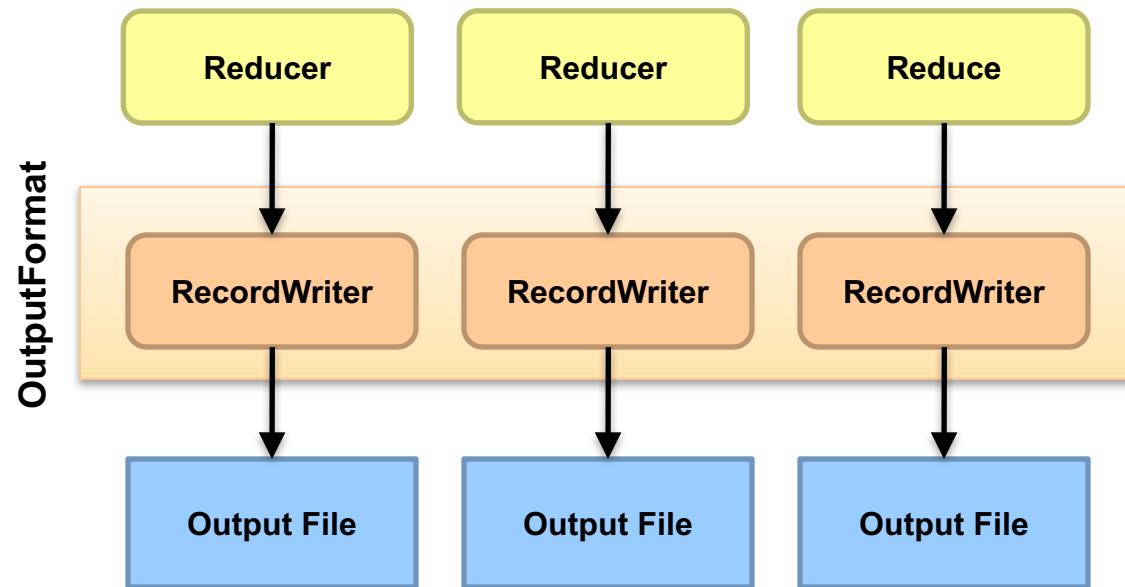












Input and Output

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 are merged into a single, partitioned file (sorted within each partition)

Combiner runs during the merges

Reduce side

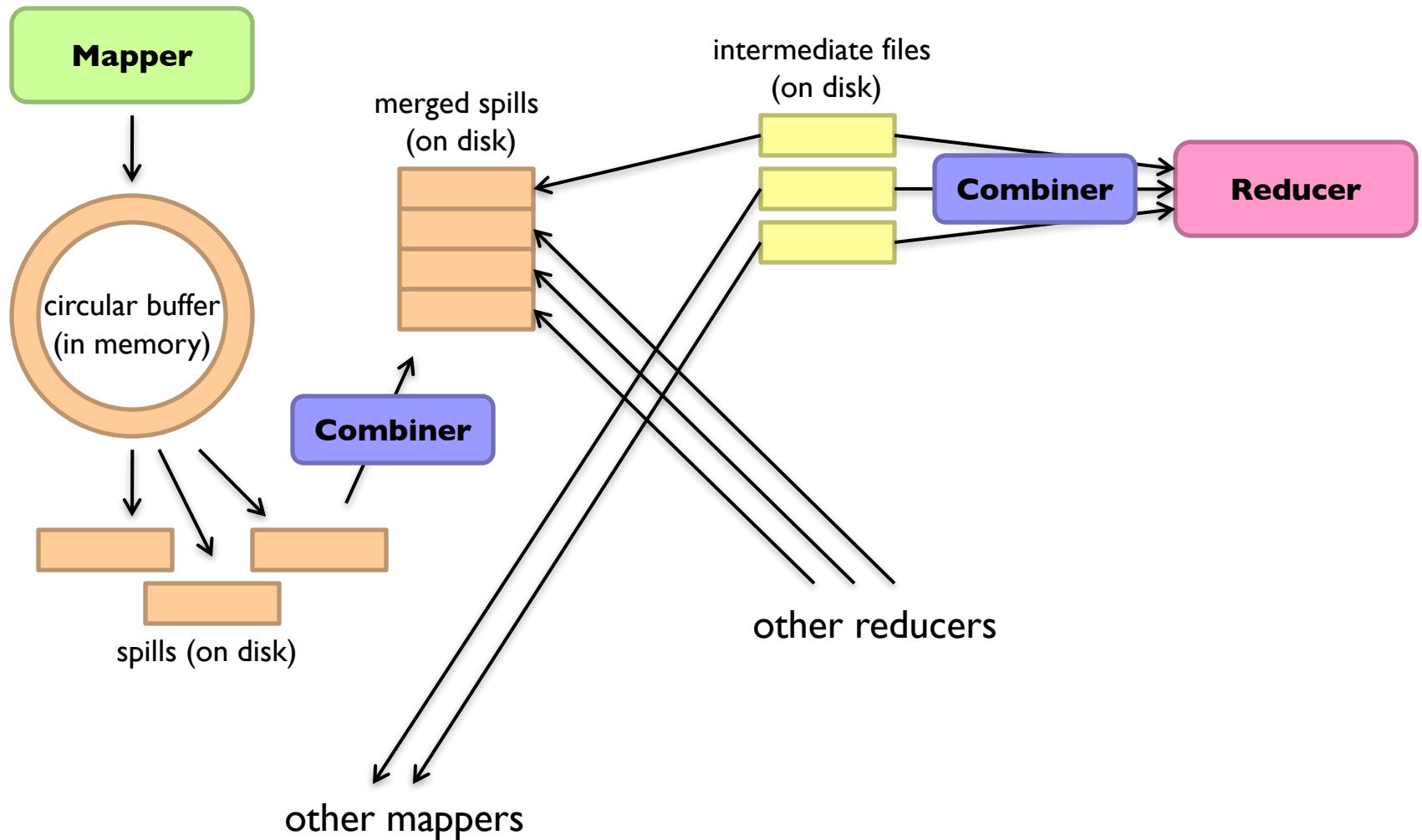
First, map outputs are copied over to reducer machine

“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 in MapReduce



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

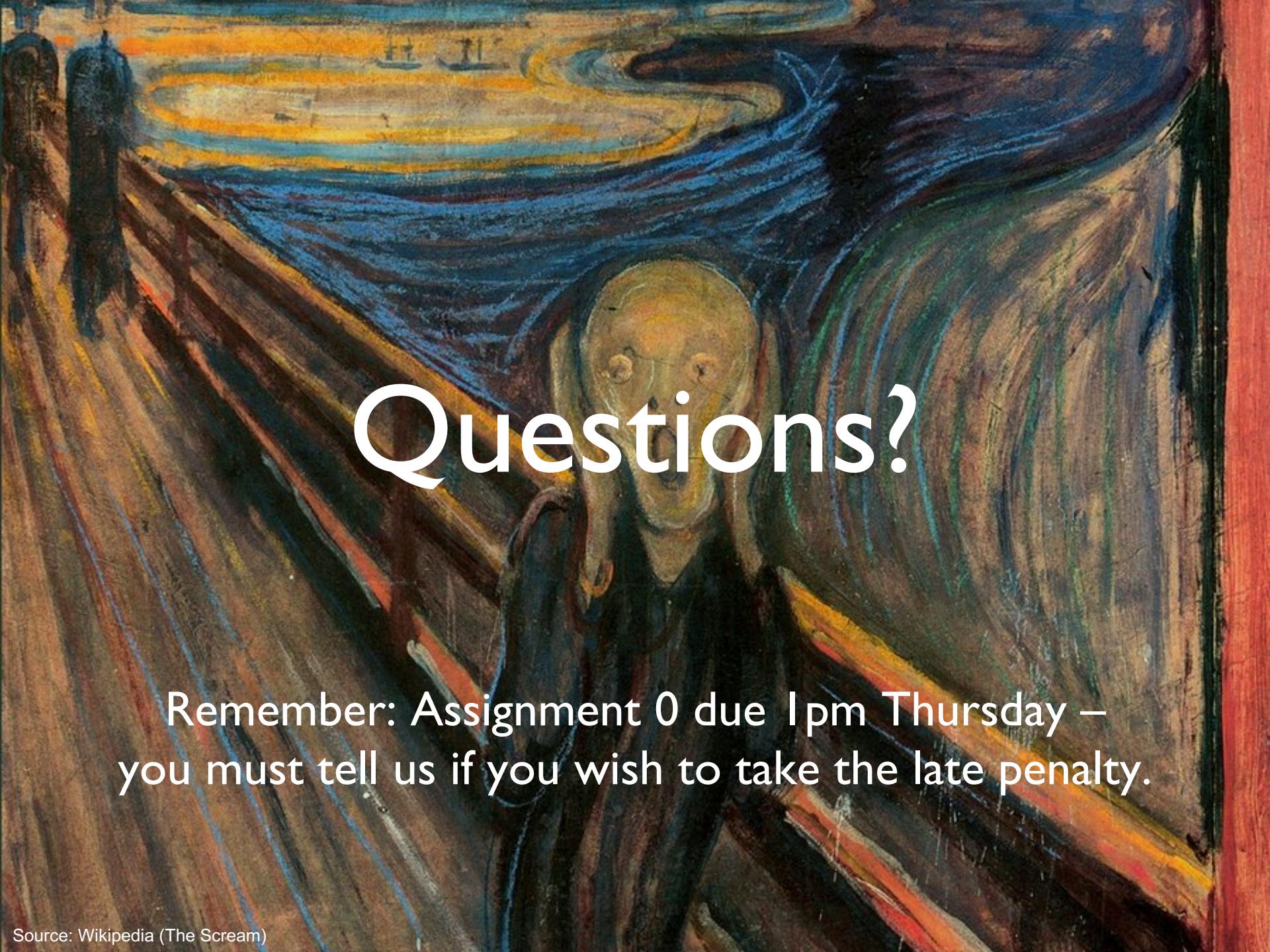
Throw `RuntimeExceptions` and capture state

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

A reproduction of Edvard Munch's painting 'The Scream'. It depicts a man with a pale face and a wide, agonized expression, with his hands clasped near his chin. He is set against a background of dark, swirling, and distorted shapes in shades of orange, yellow, and blue, suggesting a dreamlike or distressing scene. The style is characteristic of Expressionism.

Questions?

Remember: Assignment 0 due 1pm Thursday –
you must tell us if you wish to take the late penalty.