

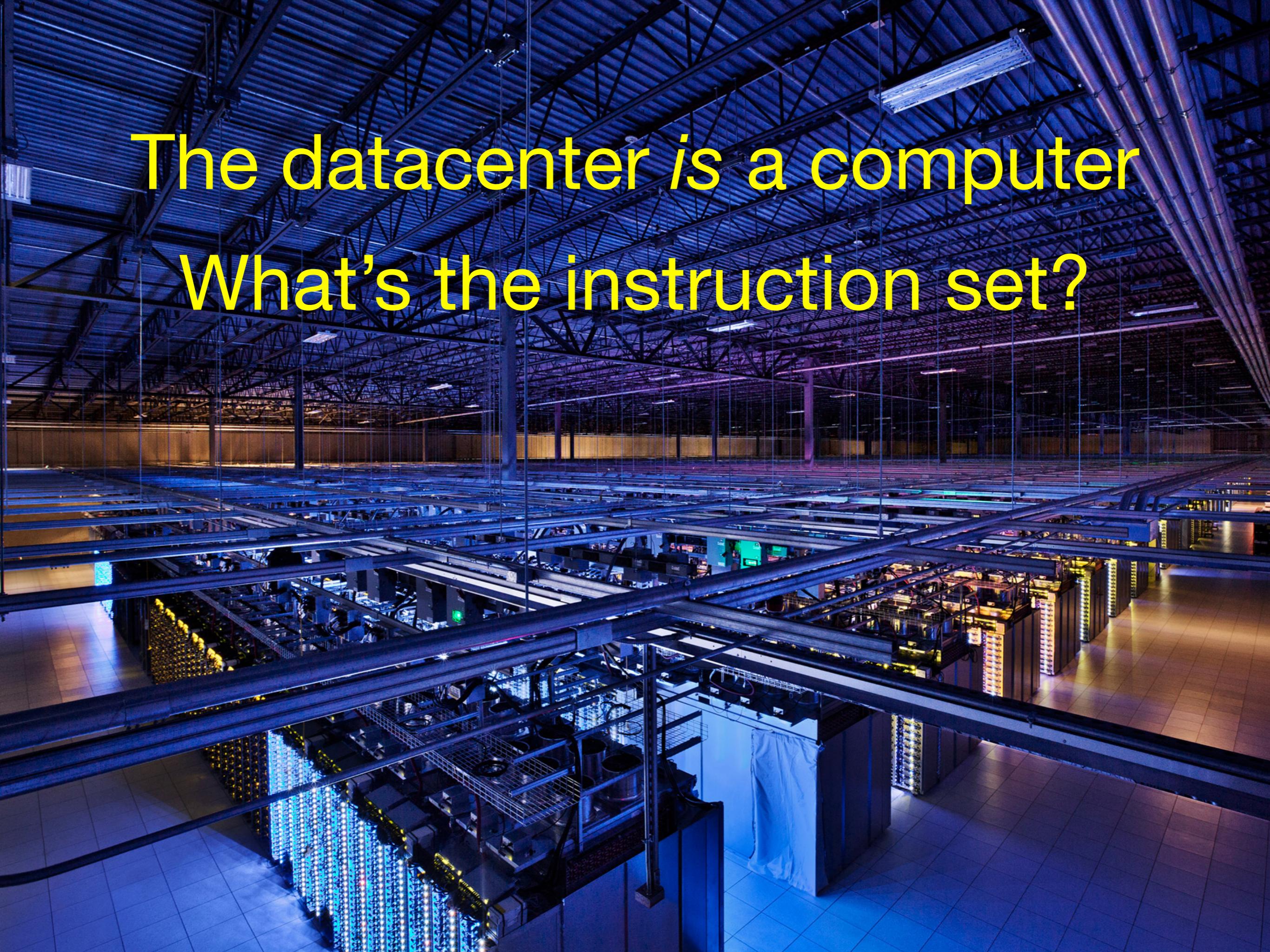
Advanced Cloud Computing

From MapReduce to Spark

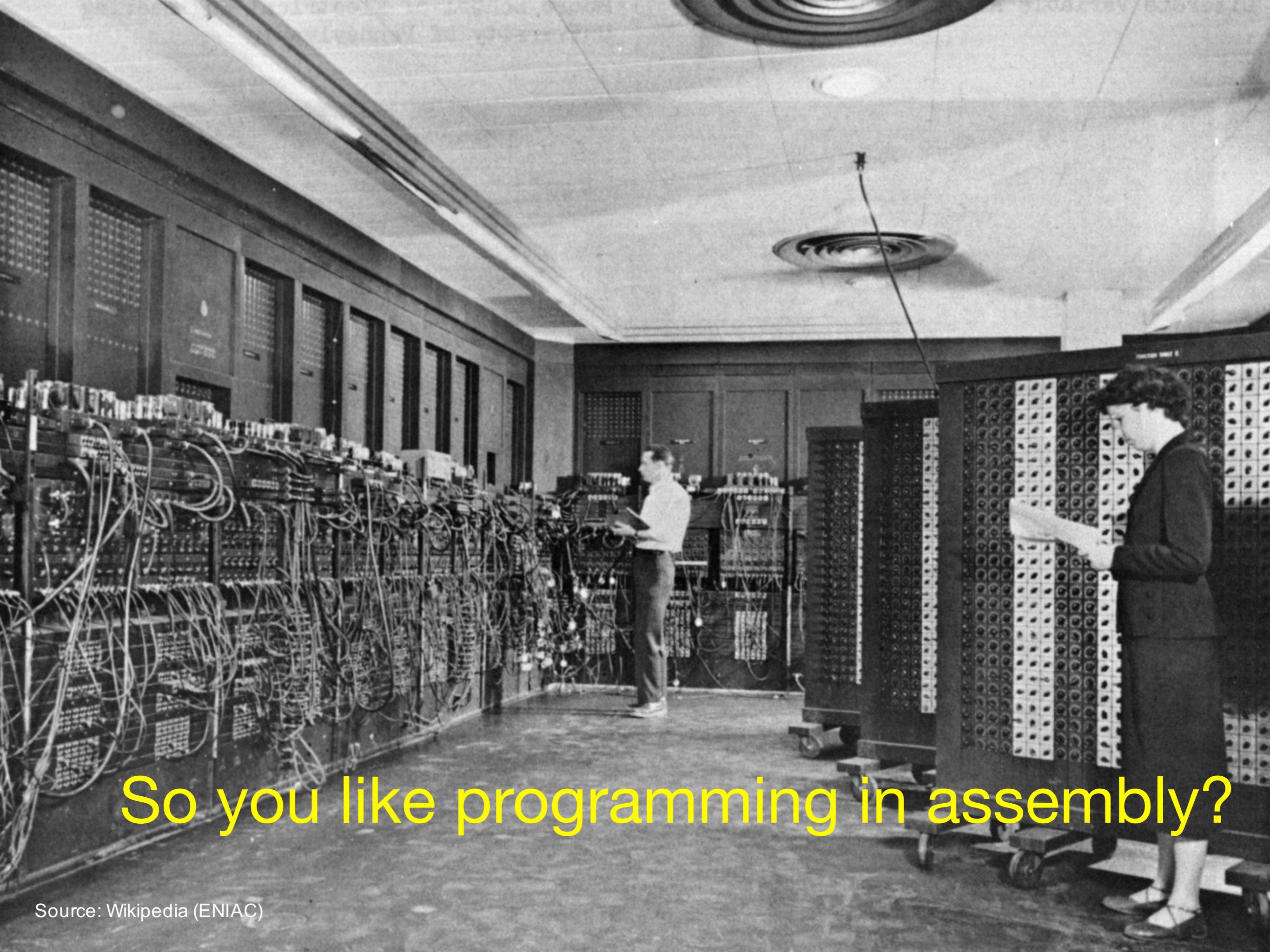
Wei Wang
CSE@HKUST
Spring 2025



THE DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系



The datacenter *is* a computer
What's the instruction set?



So you like programming in assembly?

Hadoop is great, but it's really
waaaaay too low level! (circa 2007)

What's the solution?

Design a higher-level language
Write a compiler

Hadoop is great, but it's really waaaay too low level!



What we really need is SQL

What we really need is
scripting language



facebook®

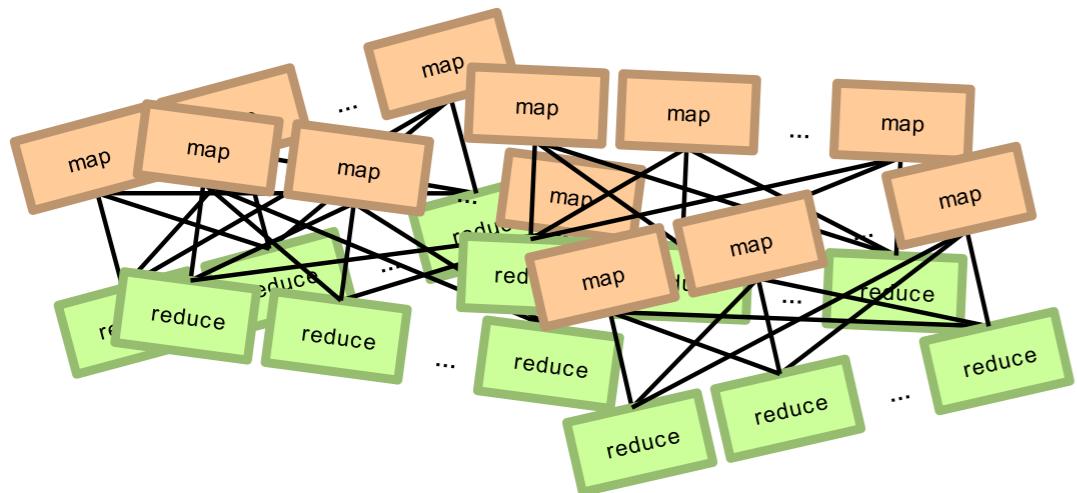


YAHOO!®

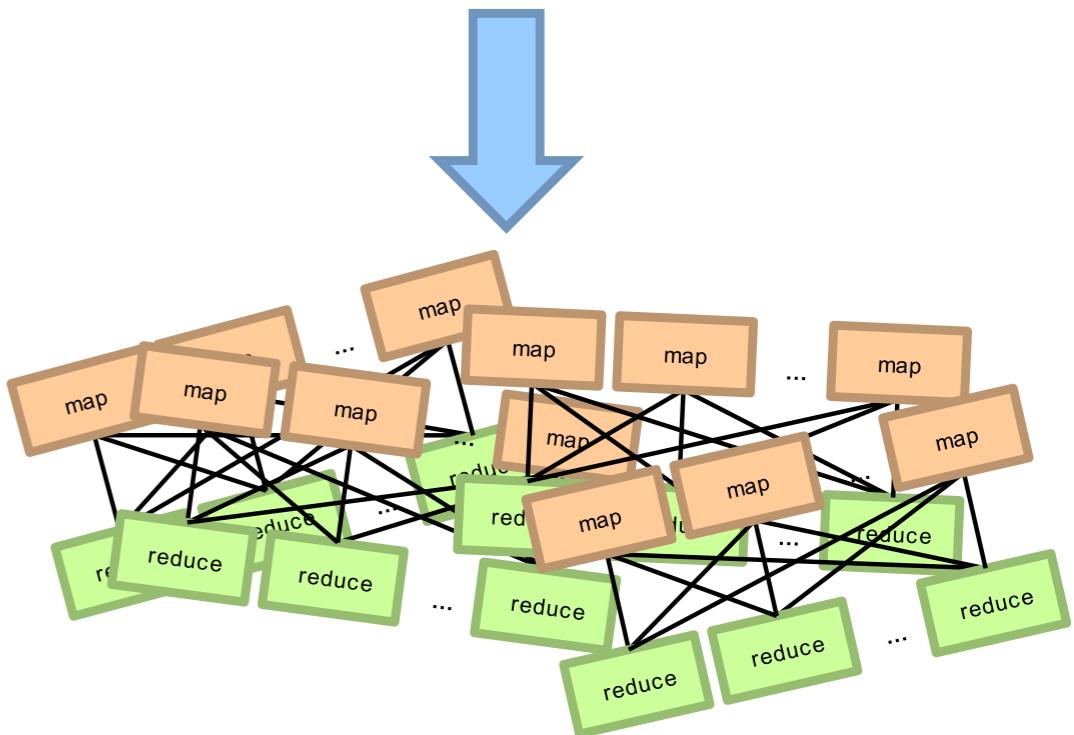


SQL

Why not just use
a database?



Pig Scripts





Jeff Hammerbacher, Information Platforms and the Rise of the Data Scientist. In *Beautiful Data*, O'Reilly, 2009.

“On the first day of logging the Facebook clickstream, more than 400 gigabytes of data was collected. The load, index, and aggregation processes for this data set really taxed the Oracle data warehouse. Even after significant tuning, we were unable to aggregate a day of clickstream data in less than 24 hours.”

Story for another day...

YAHOO!®



Pig: Example

Find the top 10 most visited pages in each category

Visits

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

URL Info

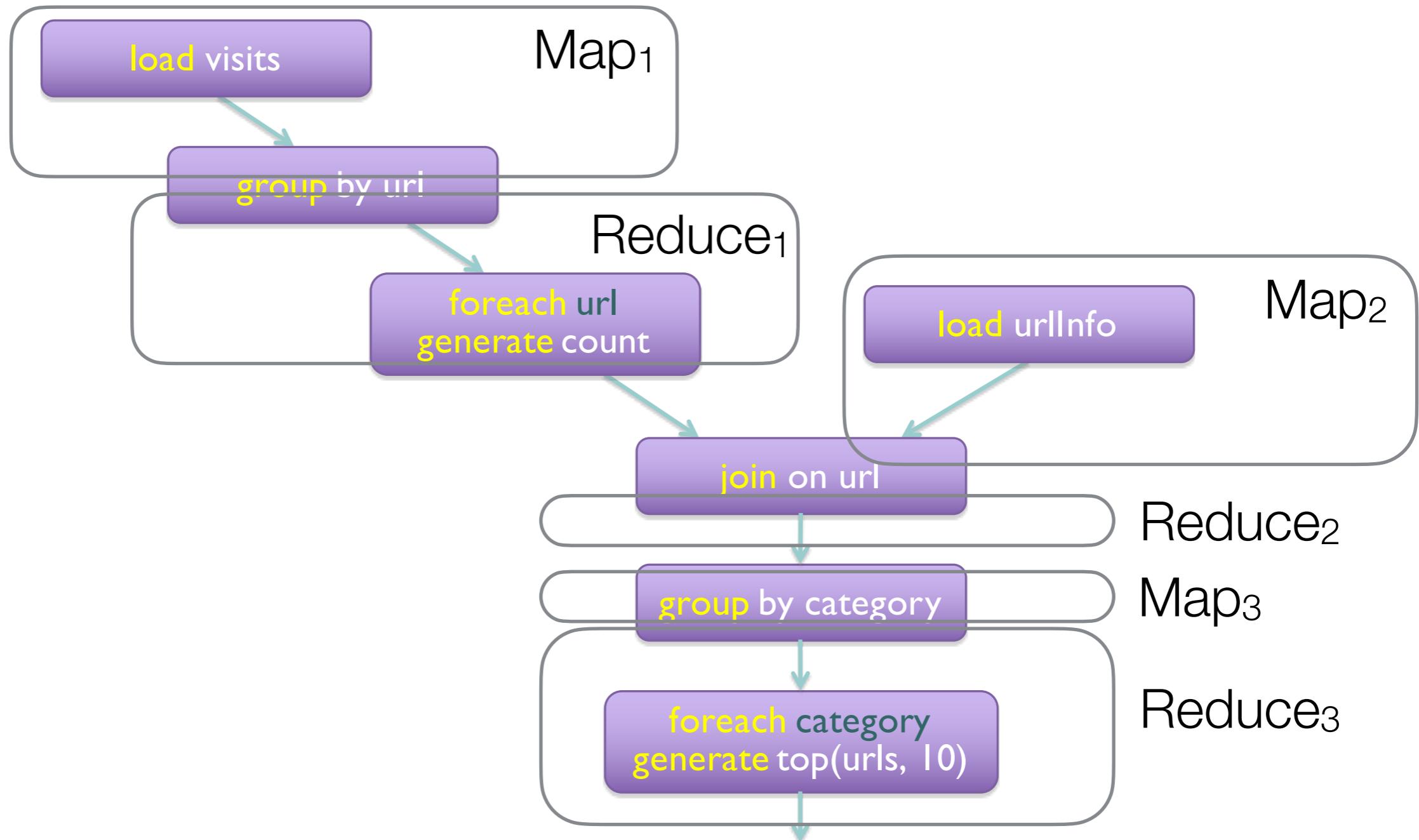
Url	Category	PageRank
<u>cnn.com</u>	News	0.9
<u>bbc.com</u>	News	0.8
<u>flickr.com</u>	Photos	0.7
<u>espn.com</u>	Sports	0.9

Pig: Example script

```
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';
```

Pig query plan



```

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';

```

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.mapred.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
            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.toString();
                if (value.charAt(0) == '1')
                    first.add(value.substring(1));
                else second.add(value.substring(1));
            }
        }
    }
}

```

Or this?

```

        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 + 1);
        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(1));
    }
}

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();
        }
        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.addInputPath(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.setOutputPath(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
        .example.class);
    group.setJobName("Group URLs");
    group.setInputFormat(KeyValueTextInputFormat.class);
    group.setOutputKeyClass(Text.class);
    group.setOutputValueClass(LongWritable.class);
    group.setMapperFormat(SequenceFileInputFormat
        .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(MREexample.class);
    top100.setJobName("Top 100 sites");
    top100.setInputFormat(SequenceFileInputFormat.class);
    top100.setOutputKeyClass(LongWritable.class);
    top100.setOutputValueClass(Text.class);
    top100.setMapperFormat(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/top100sitesforusers1to25"));
    top100.setNumReduceTasks(1);
    Job limit = new Job(top100);
    limit.addDependingJob(groupJob);

    JobControl jc = new JobControl("Find top
18 to 25");
    jc.addJob(loadPages);
    jc.addJob(loadUsers);
    jc.addJob(joinJob);
    jc.addJob(groupJob);
    jc.addJob(limit);
    jc.run();
}

100 sites for users

```

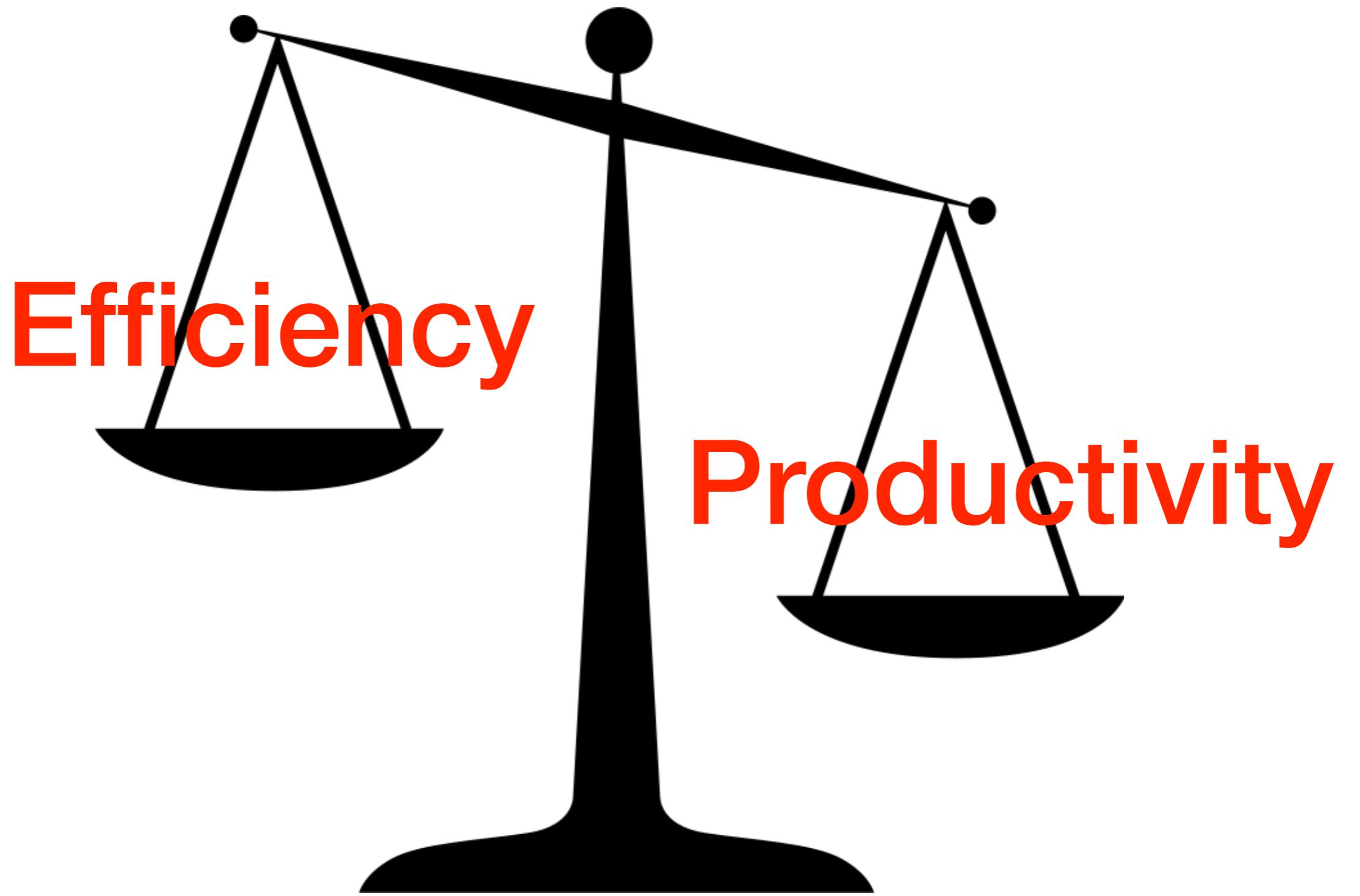
Programming at a high level

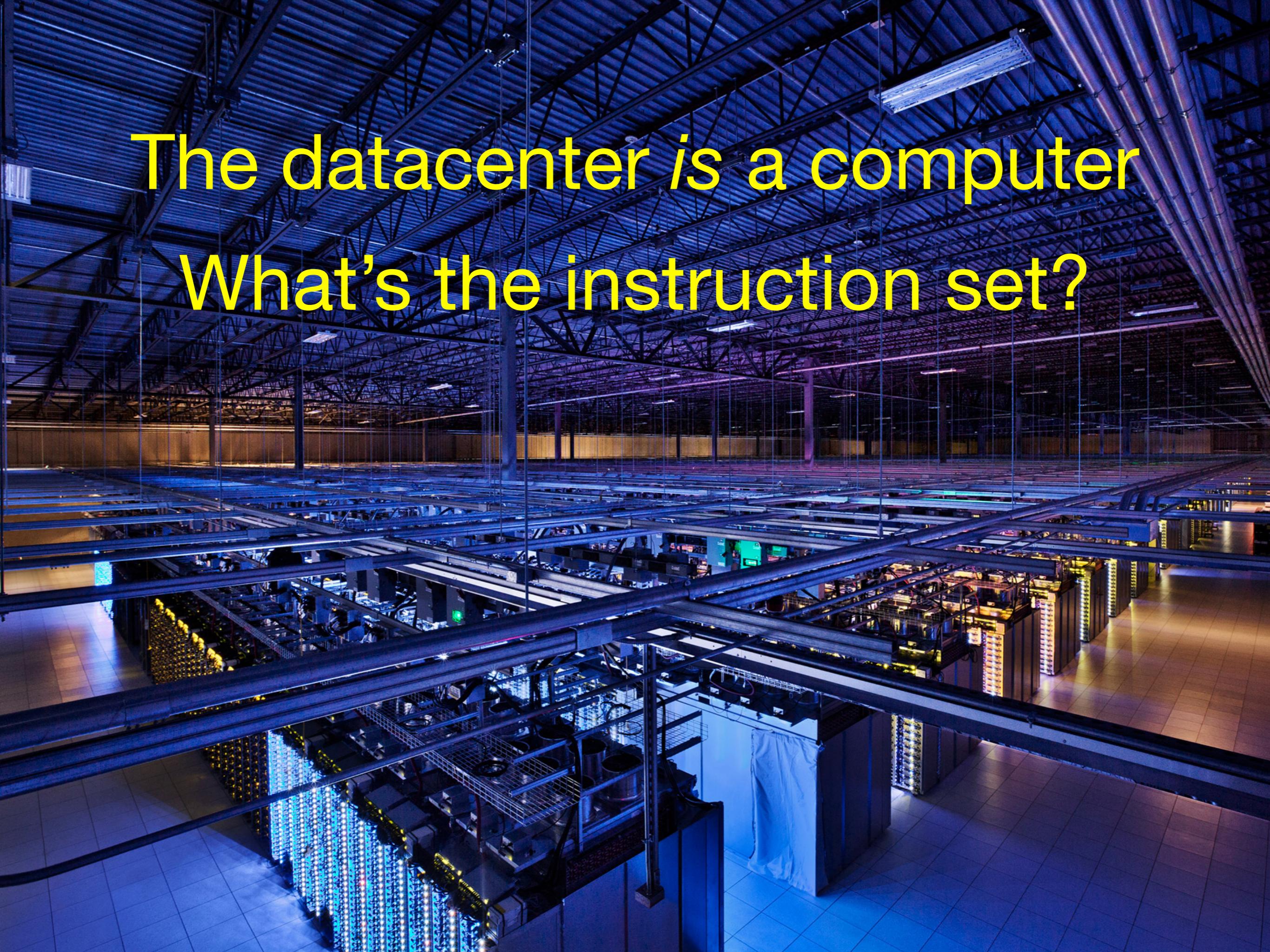
Specify “**what**”, and let the system figure out “**how**”

But isn't Pig slower?

Sure, but C can be slower than assembly too...

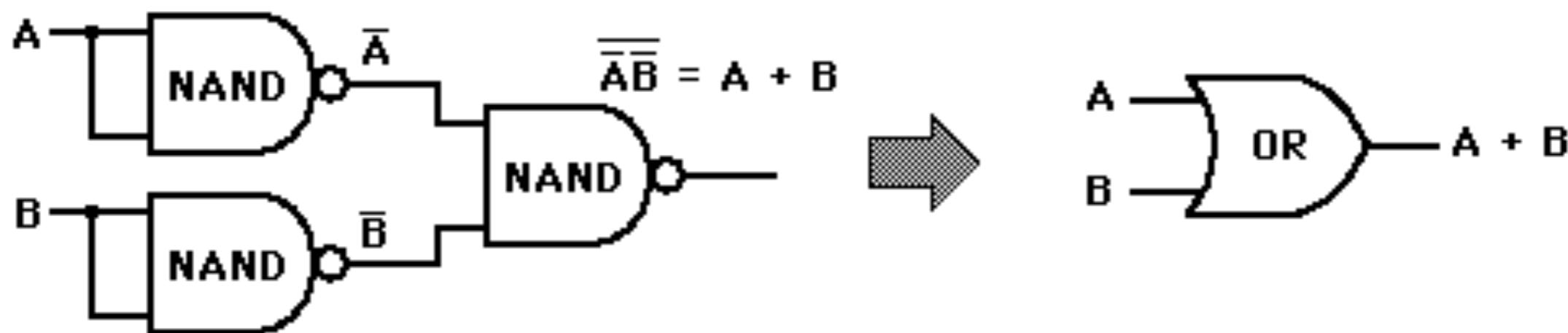
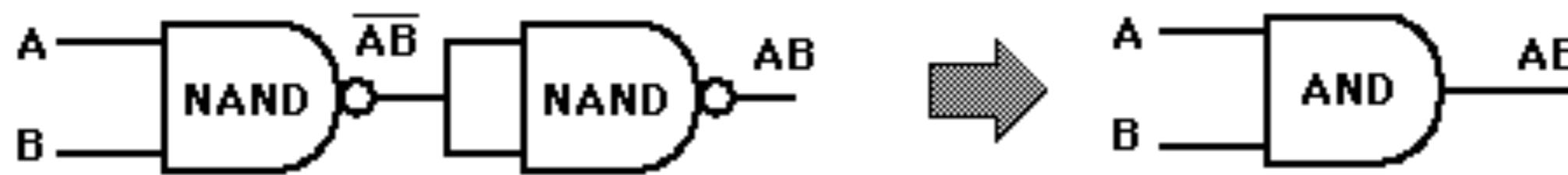
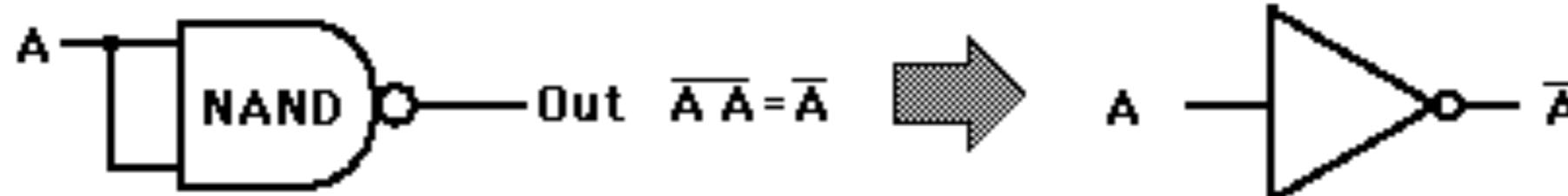






The datacenter *is* a computer
What's the instruction set?

Analogy: NAND Gates are universal



Let's design a data processing language “from scratch”!

Why is MapReduce the way it is?

Data-parallel dataflow languages

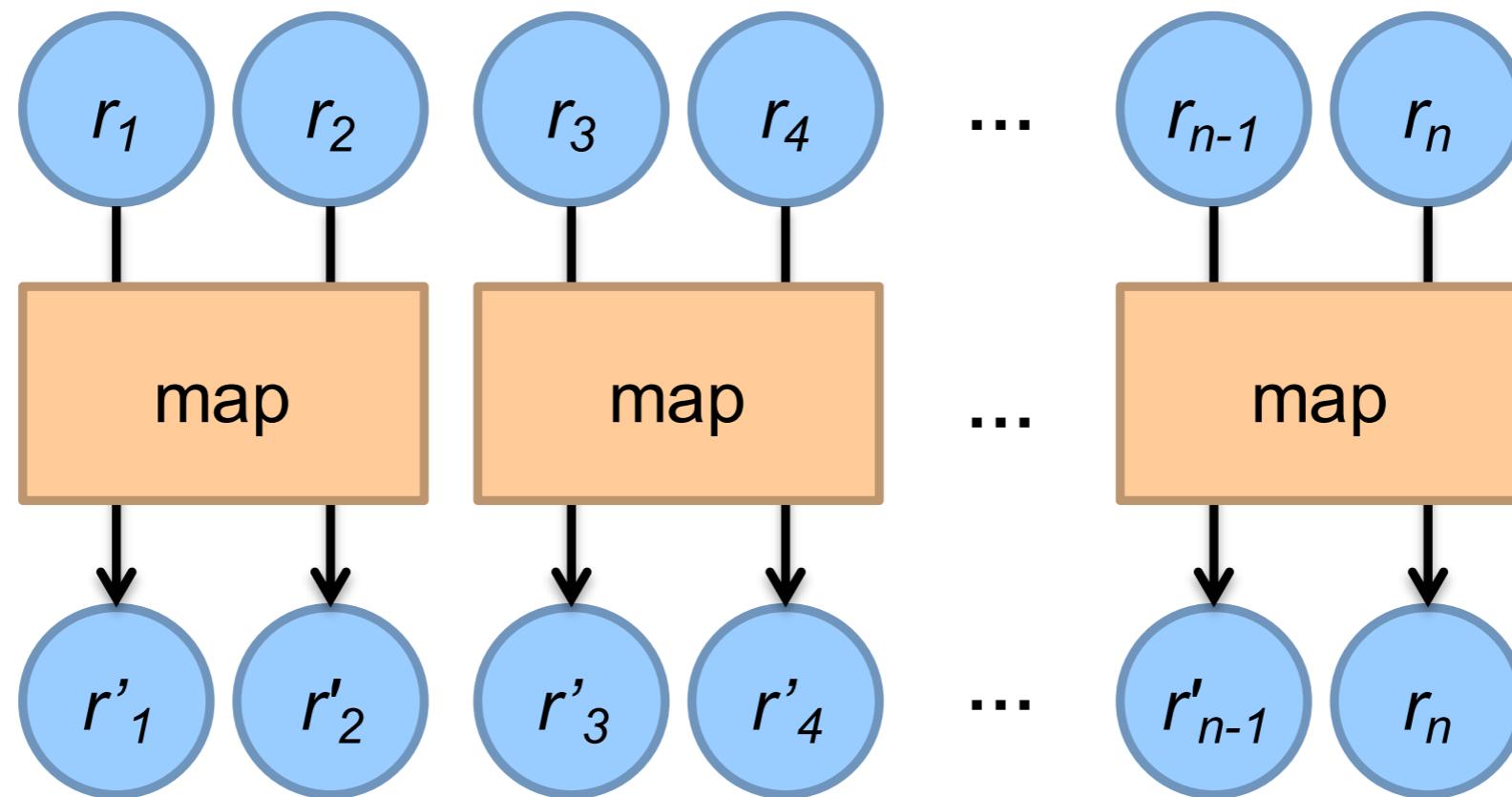
We have a collection of **records**

- ▶ want to apply a bunch of transformations to compute some results

Assumptions

- ▶ static collection records (not necessarily key-value pairs)

We need per-record processing
(note: not necessarily key-value pairs)



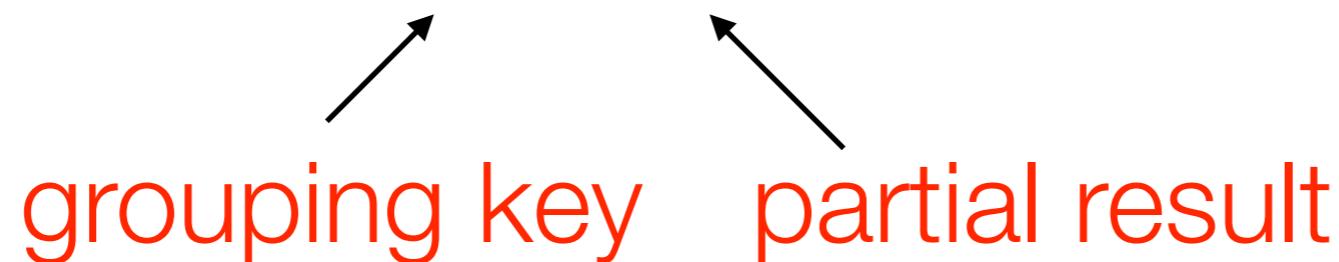
Easy to parallelize maps

Record to “mapper” assignment is an implementation detail

Map alone isn't enough

We need a way to group partial results

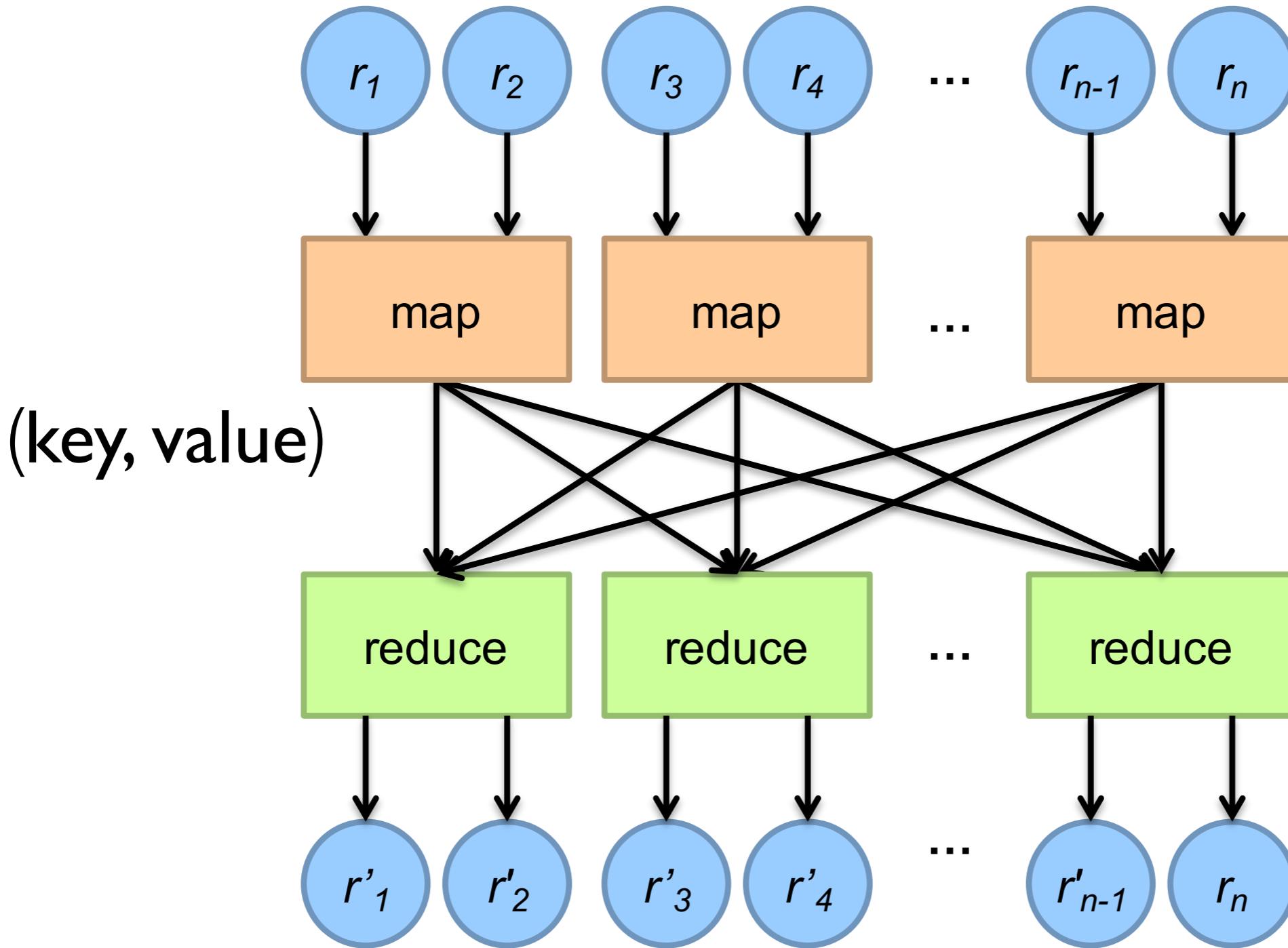
- ▶ intermediate (**key, value**) pairs



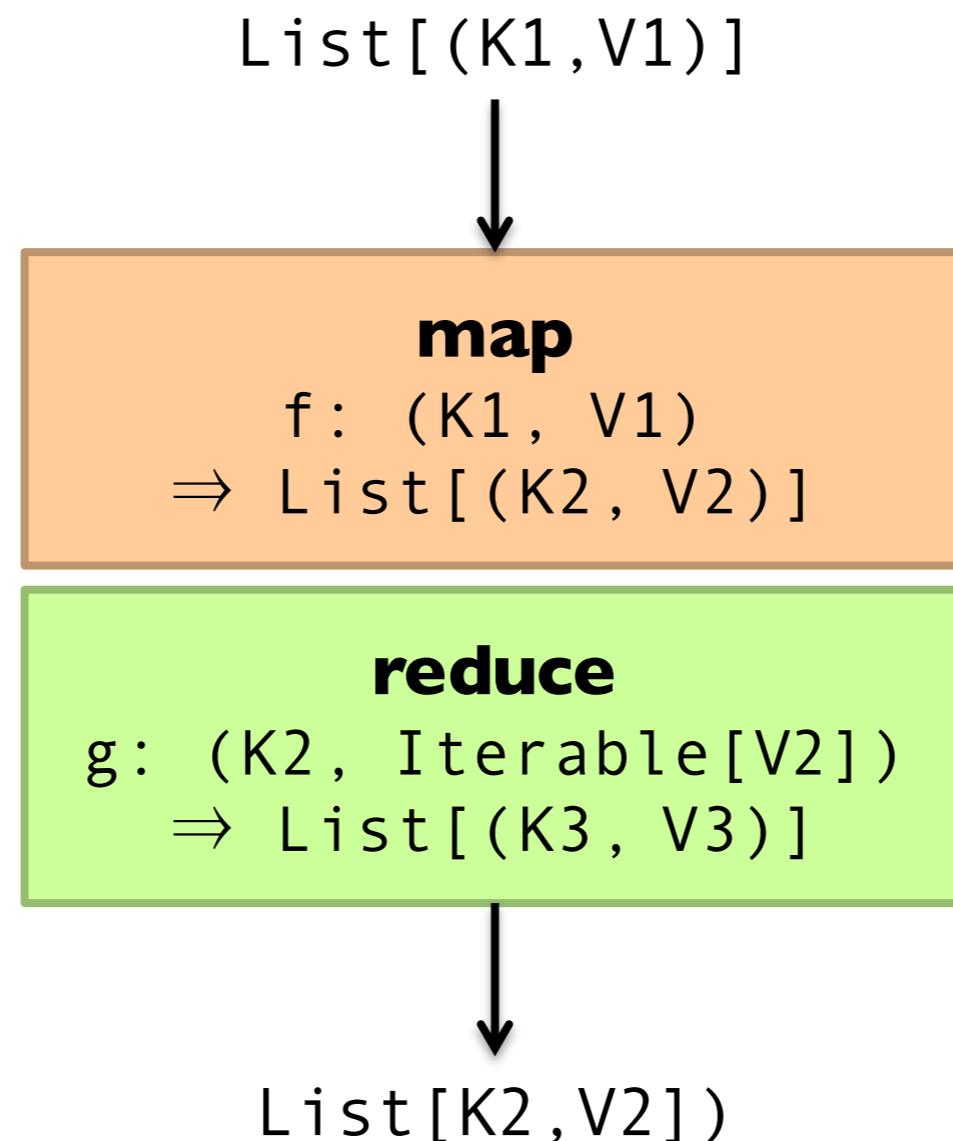
For each key, we can apply some computation

- ▶ e.g., aggregation, counting

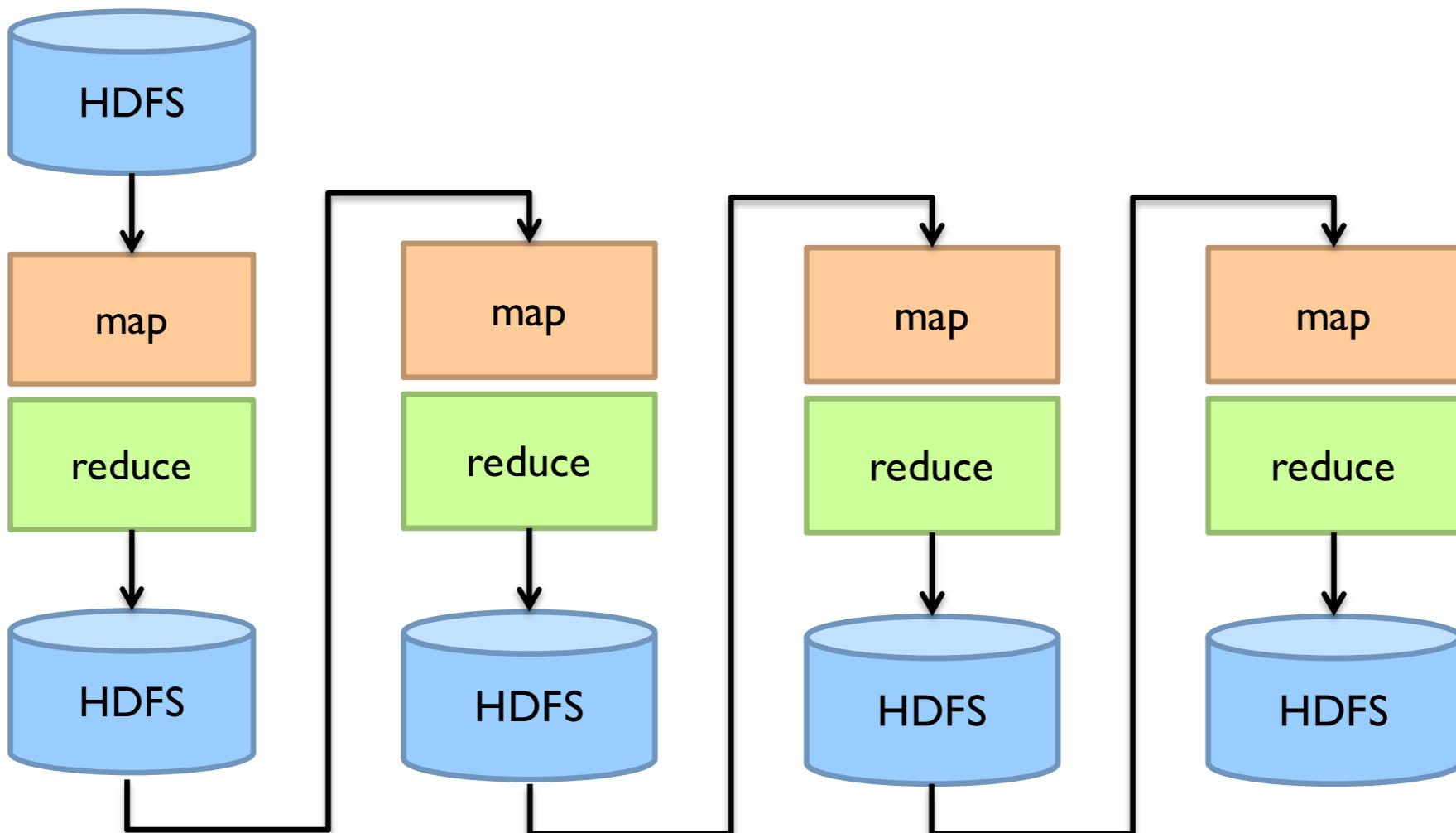
MapReduce



MapReduce

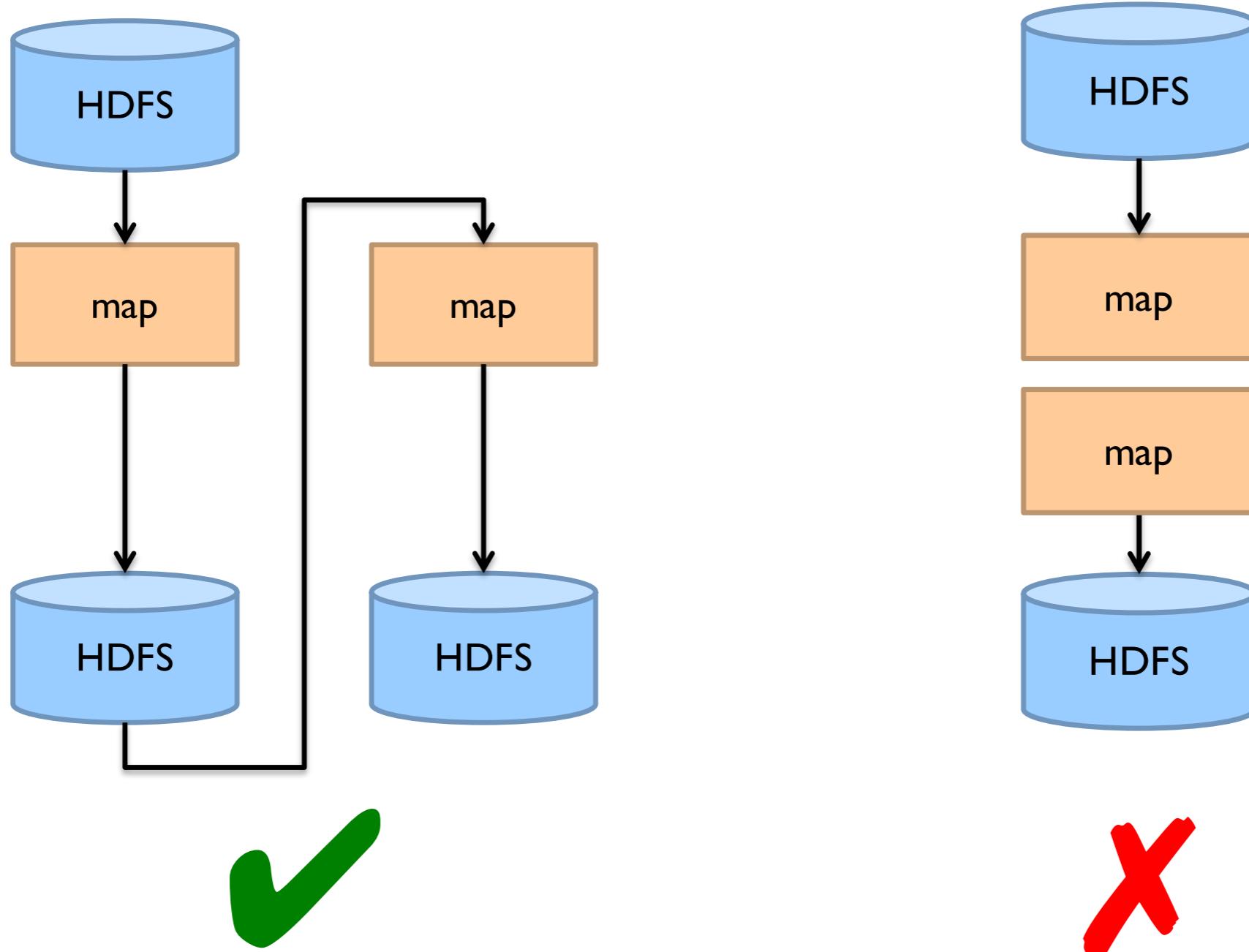


MapReduce workflows

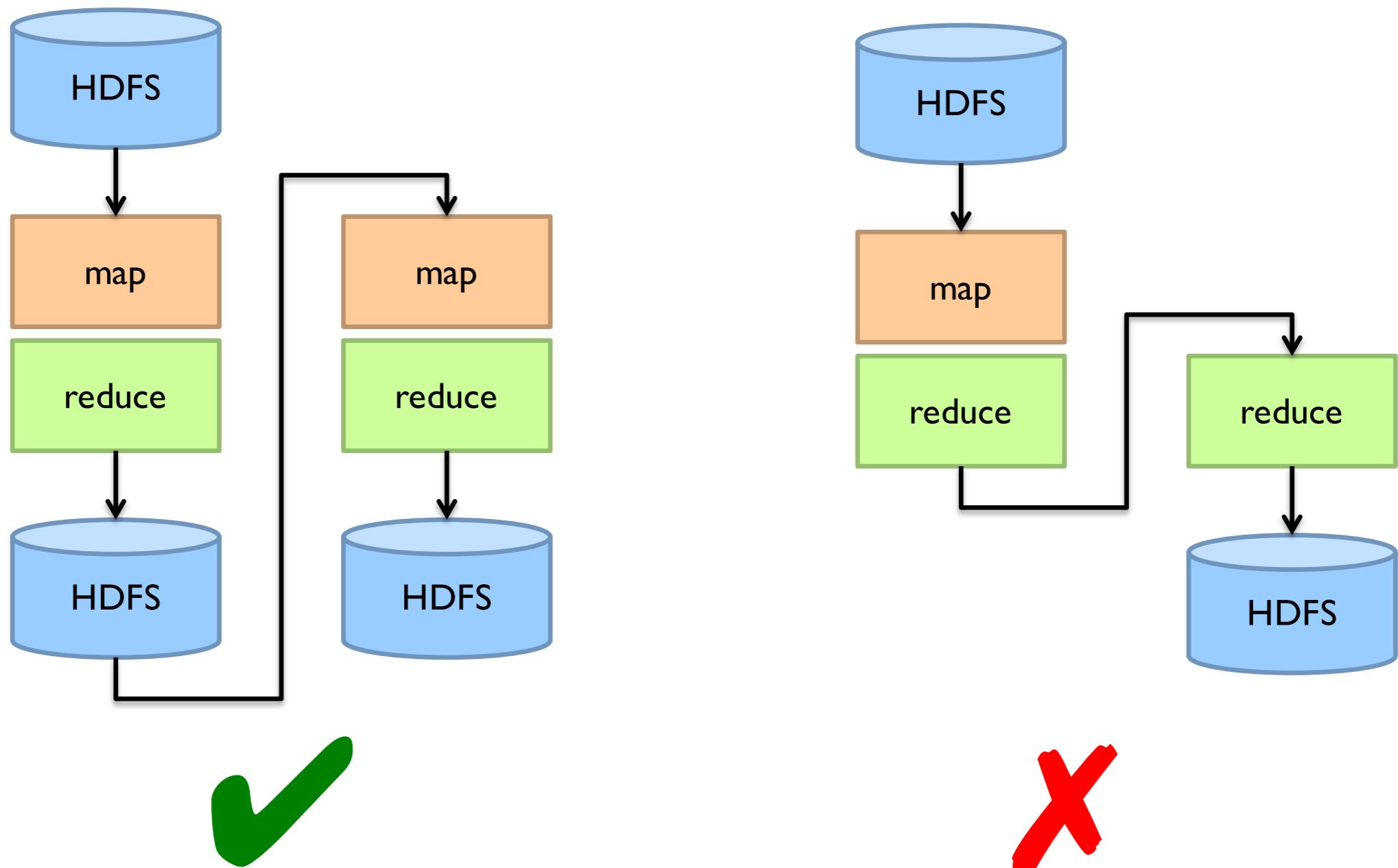


What's wrong?

Want MM?



Want MRR?



In MapReduce, the only way to share data across jobs is stable storage

Disk I/O is waaaaaaay too slow!

Data-parallel dataflow languages

We have a collection of **records**

- ▶ want to apply a bunch of transformations to compute some results

What are the operators?

- ▶ MapReduce?





Where the hype is!

- ▶ answer to “What’s beyond MapReduce?”

Brief history:

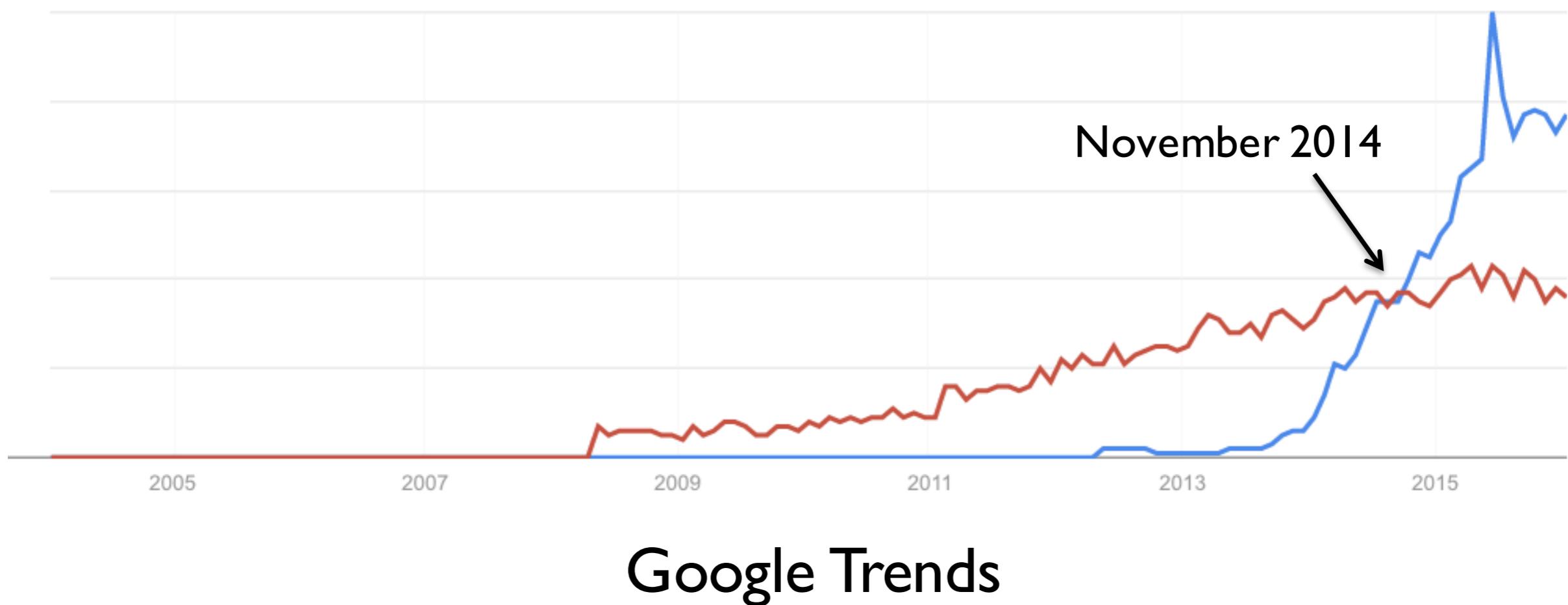
- ▶ developed at UC Berkeley AMPLab in 2009
- ▶ open-sourced in 2010
- ▶ became top-level Apache project in Feb 2014
- ▶ commercial support provided by Databricks



Matei Zaharia
Stanford University, CTO at Databrick



vs.



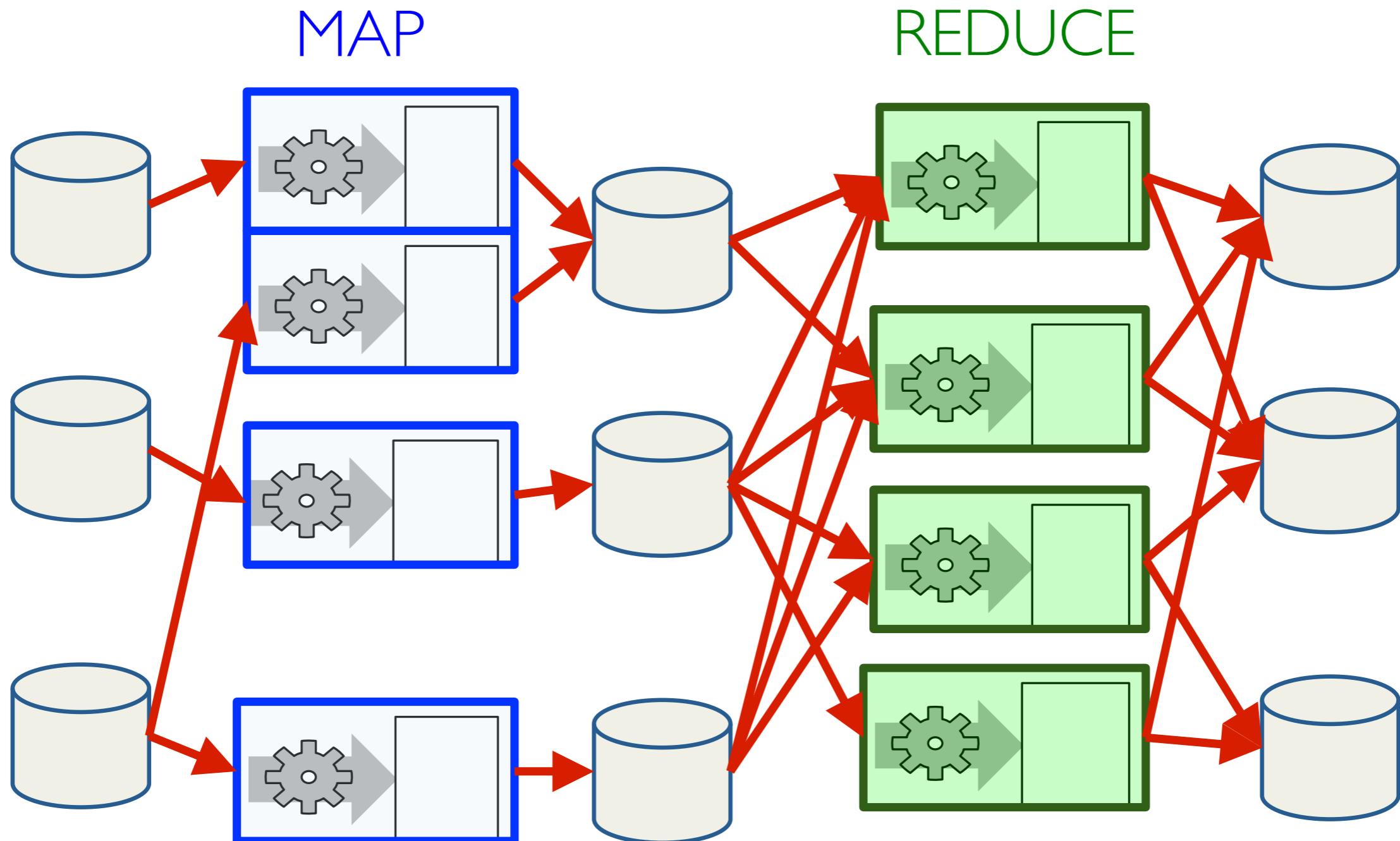
Motivation

MapReduce is a restrictive programming model for batch processing jobs

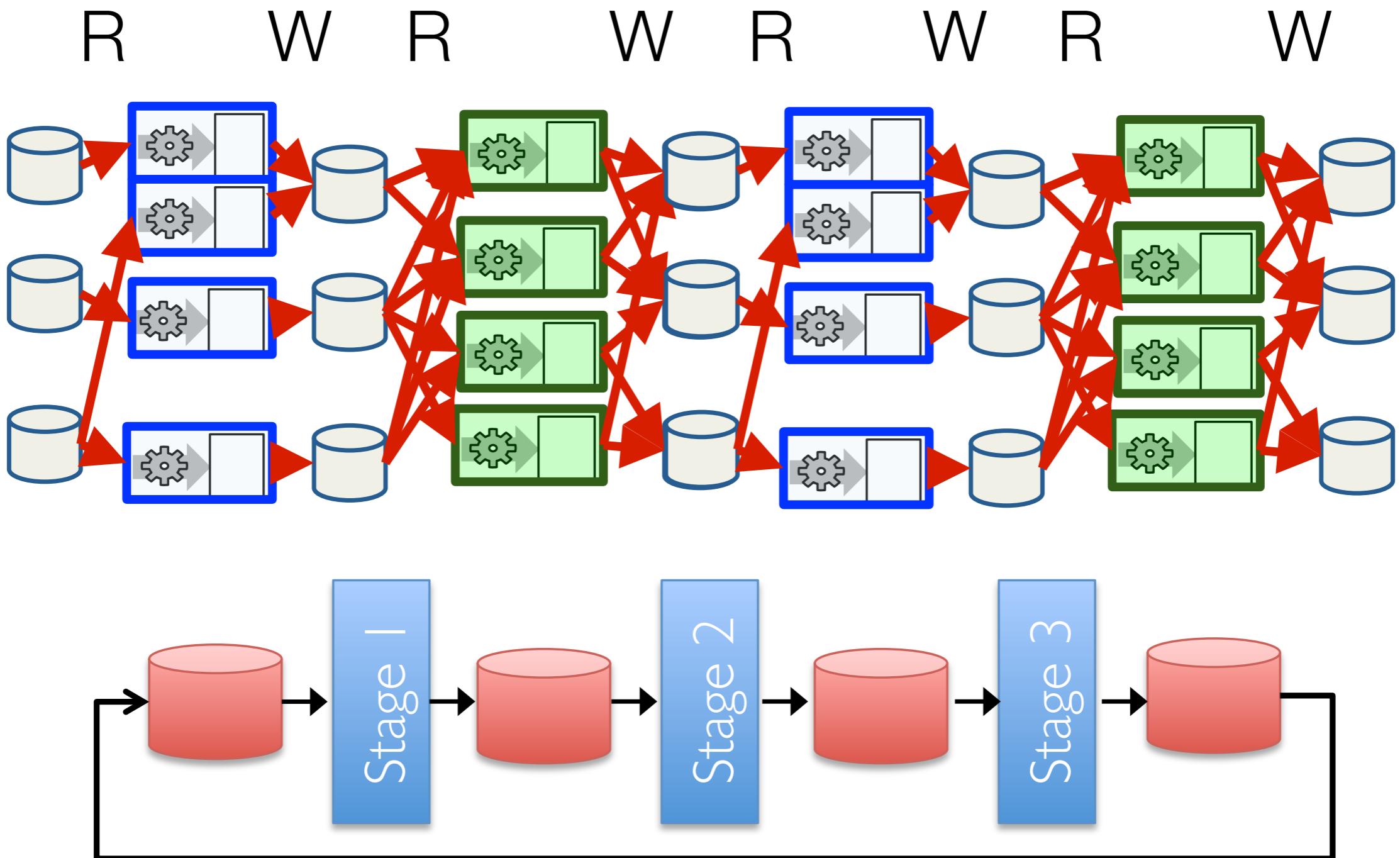
But we want more...

- ▶ More **complex**, multi-stage data processing, e.g., iterative machine learning
- ▶ Low latency, **interactive** ad-hoc data analytics, e.g., streaming processing

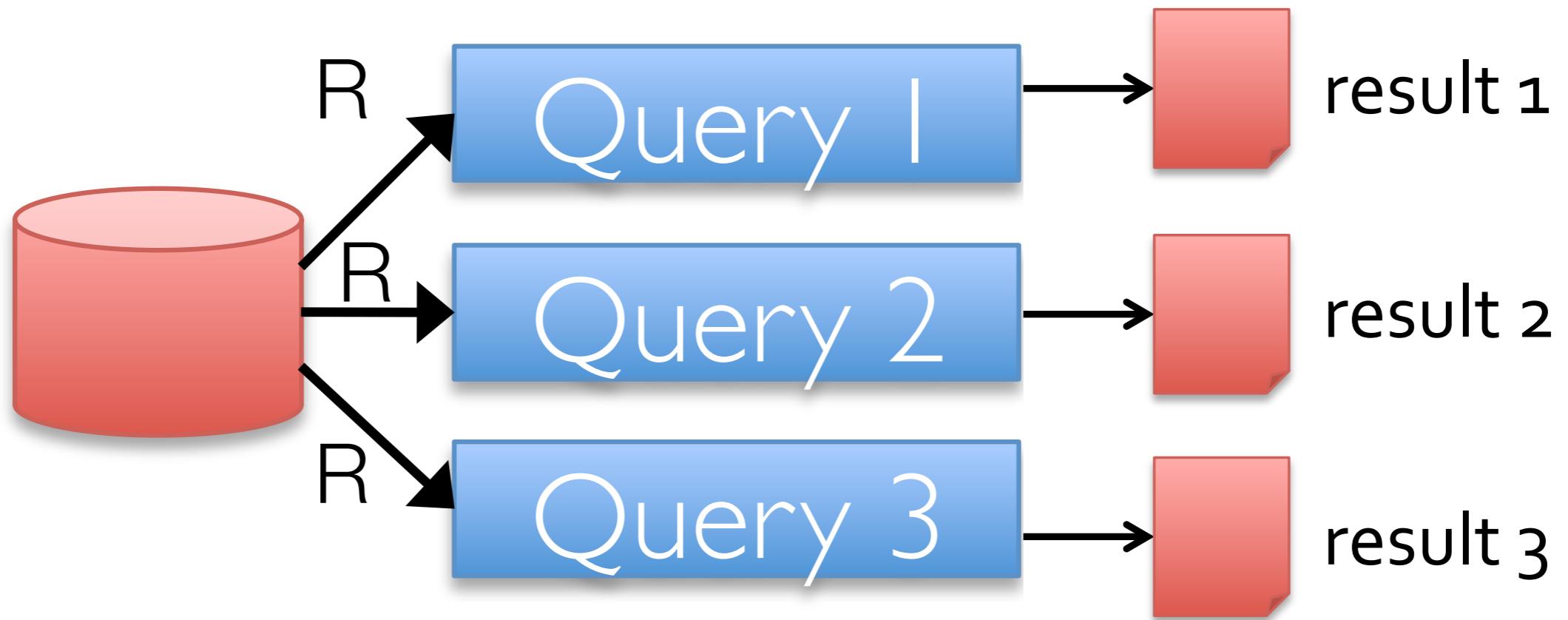
Each stage passes through the stable storage!



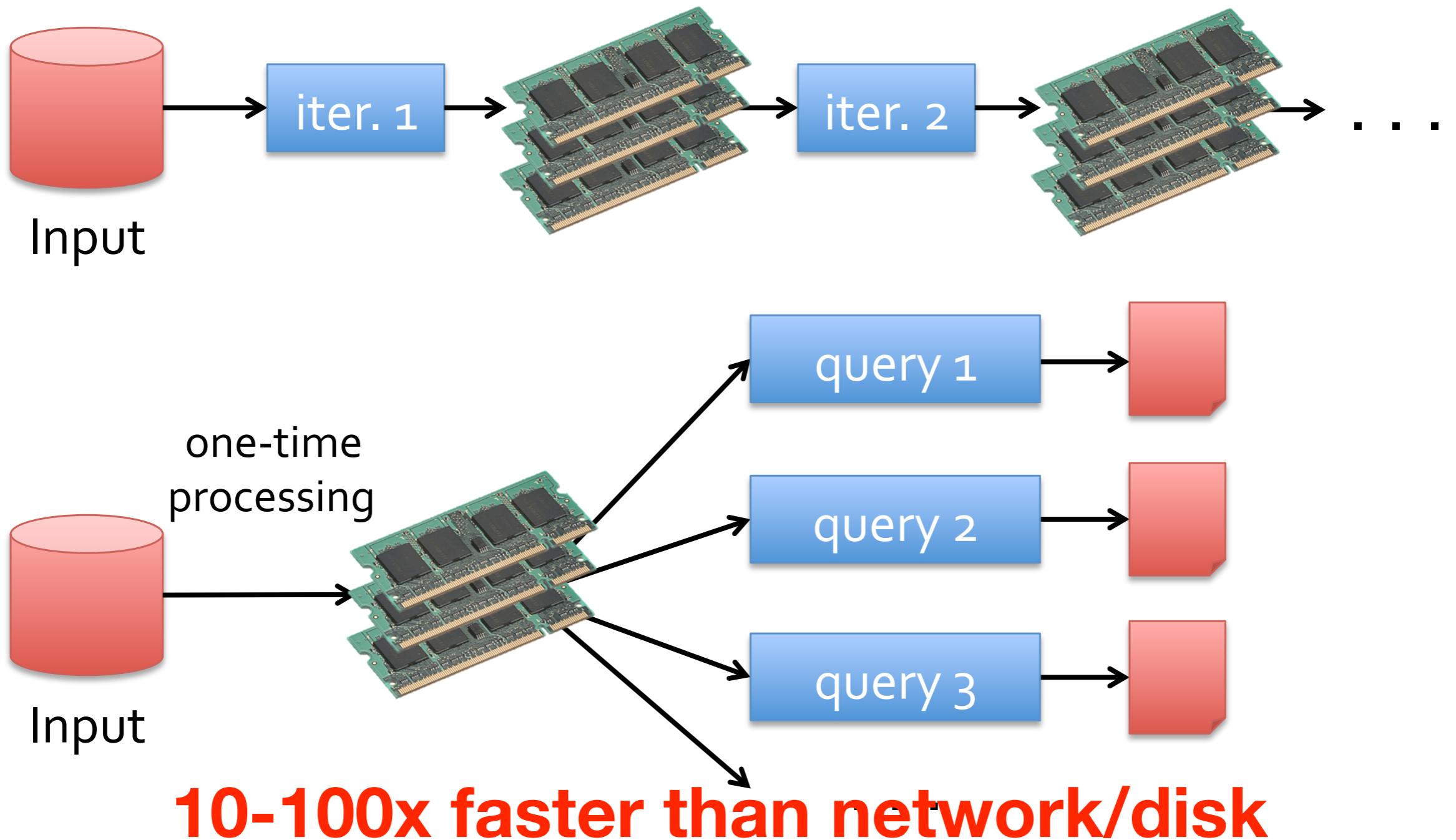
Iterative computations



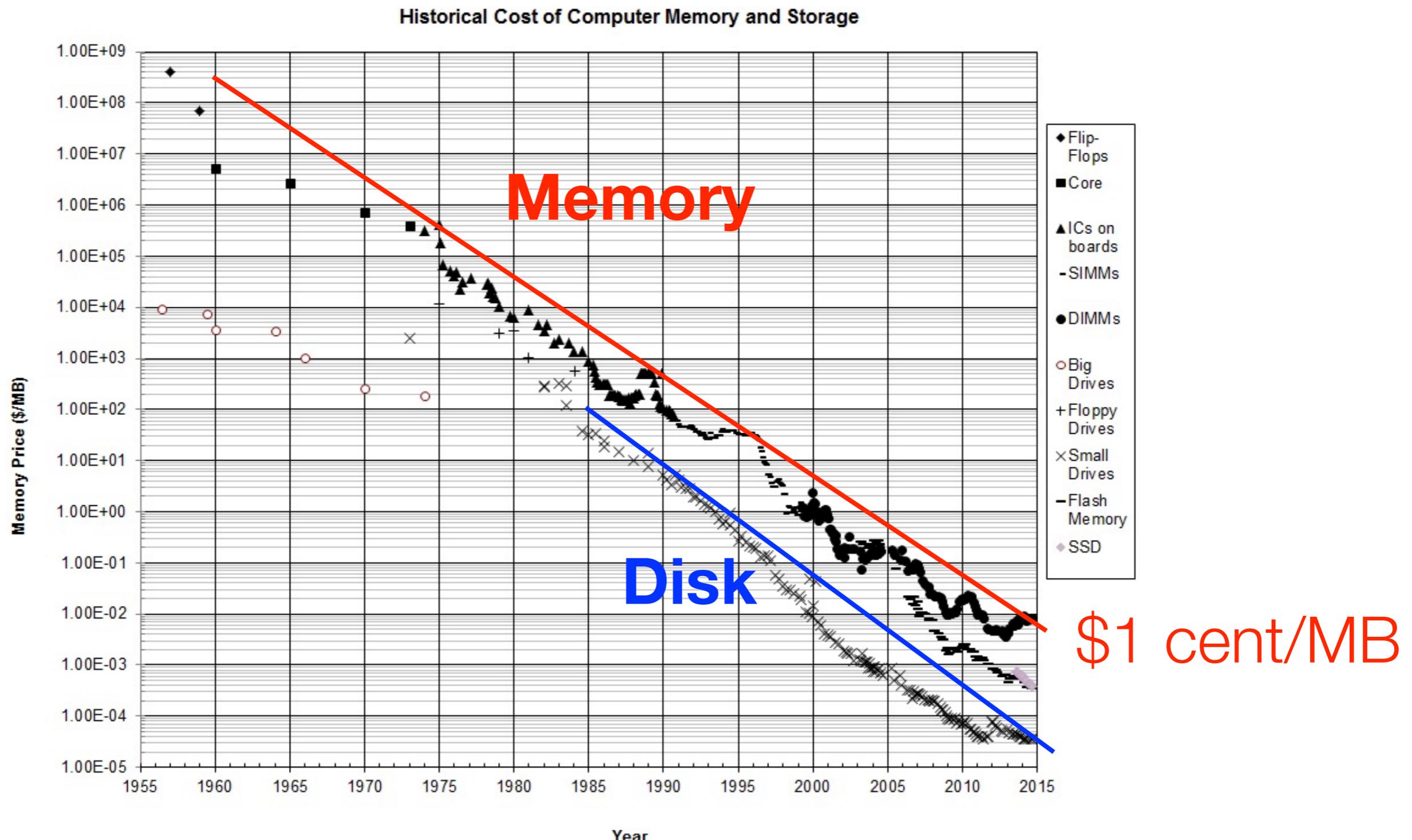
Interactive queries



Intuition: In-memory data sharing



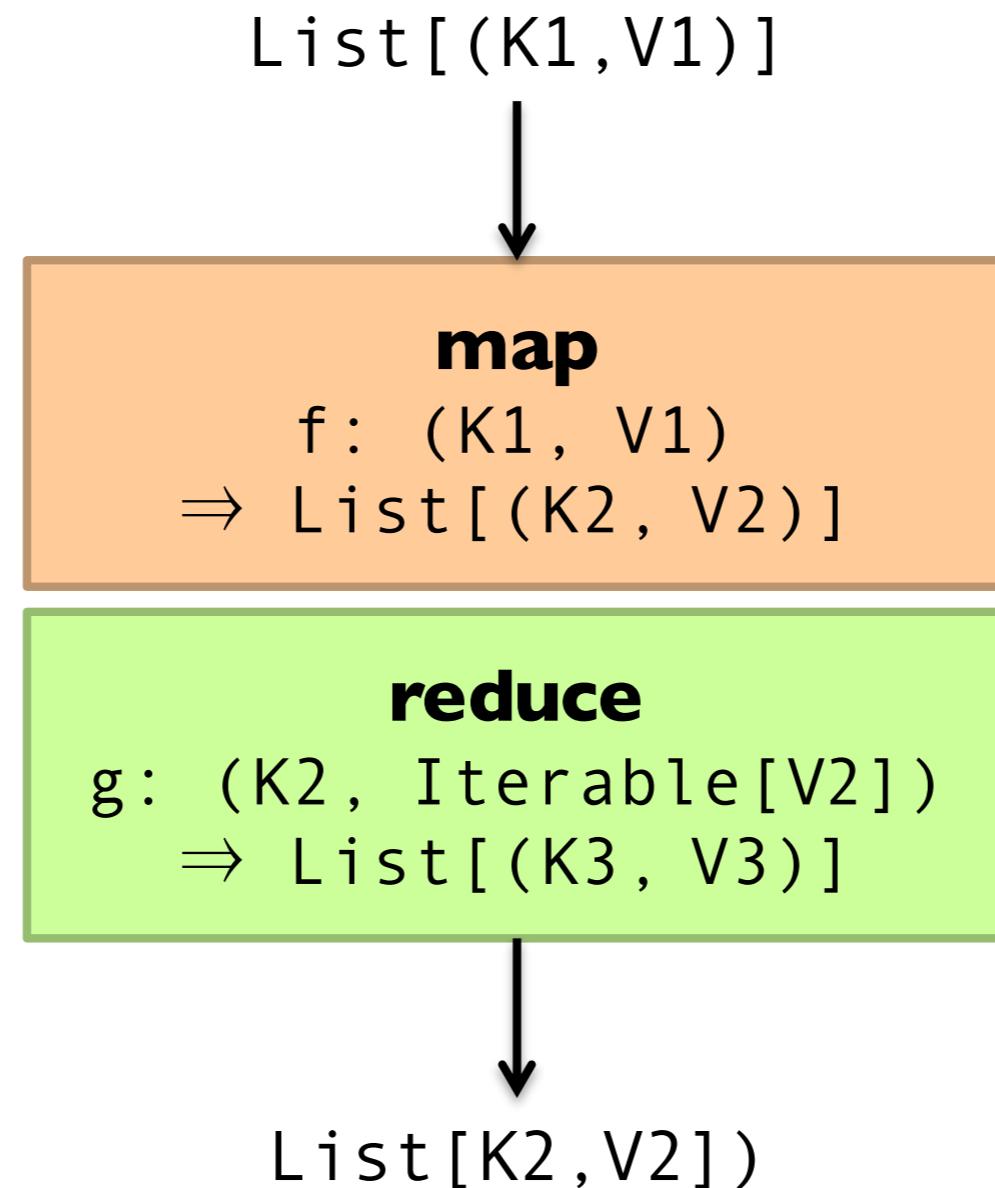
Cost of Memory



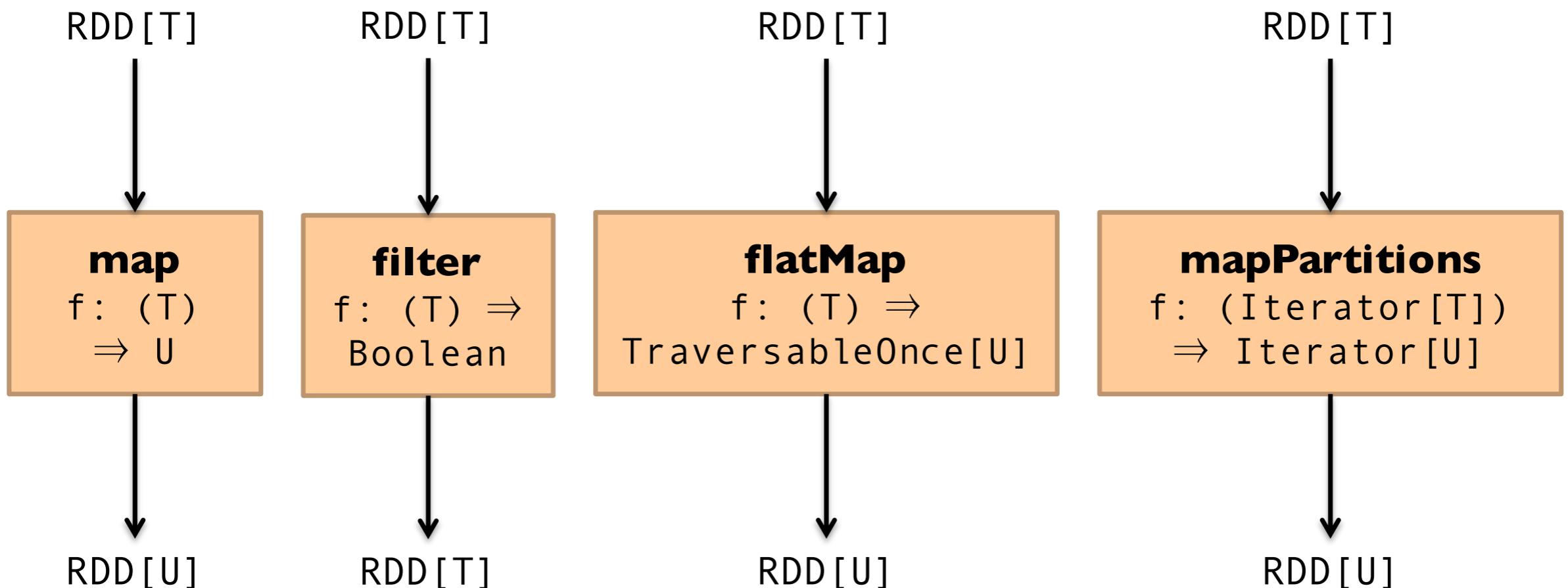
Intermediate results are stored in-memory
as **Resilient Distributed Datasets (RDDs)**

Much more next lecture...

MapReduce

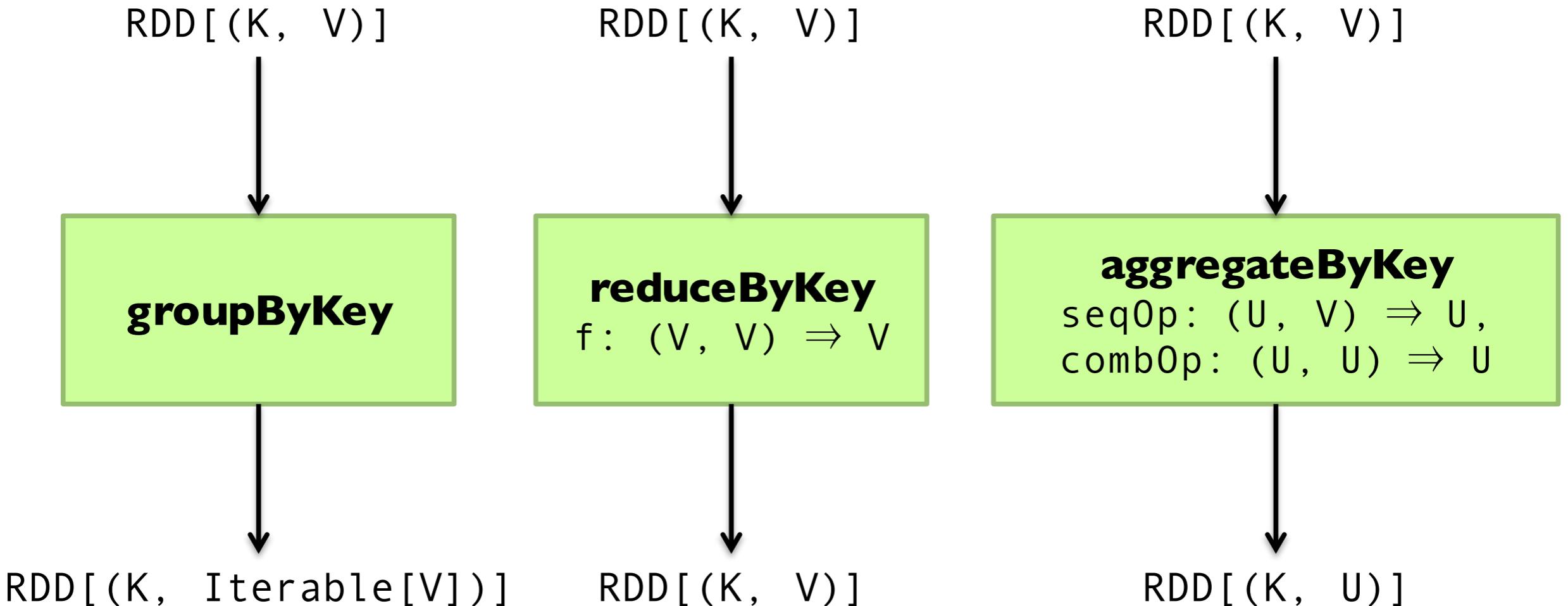


Map-like operations



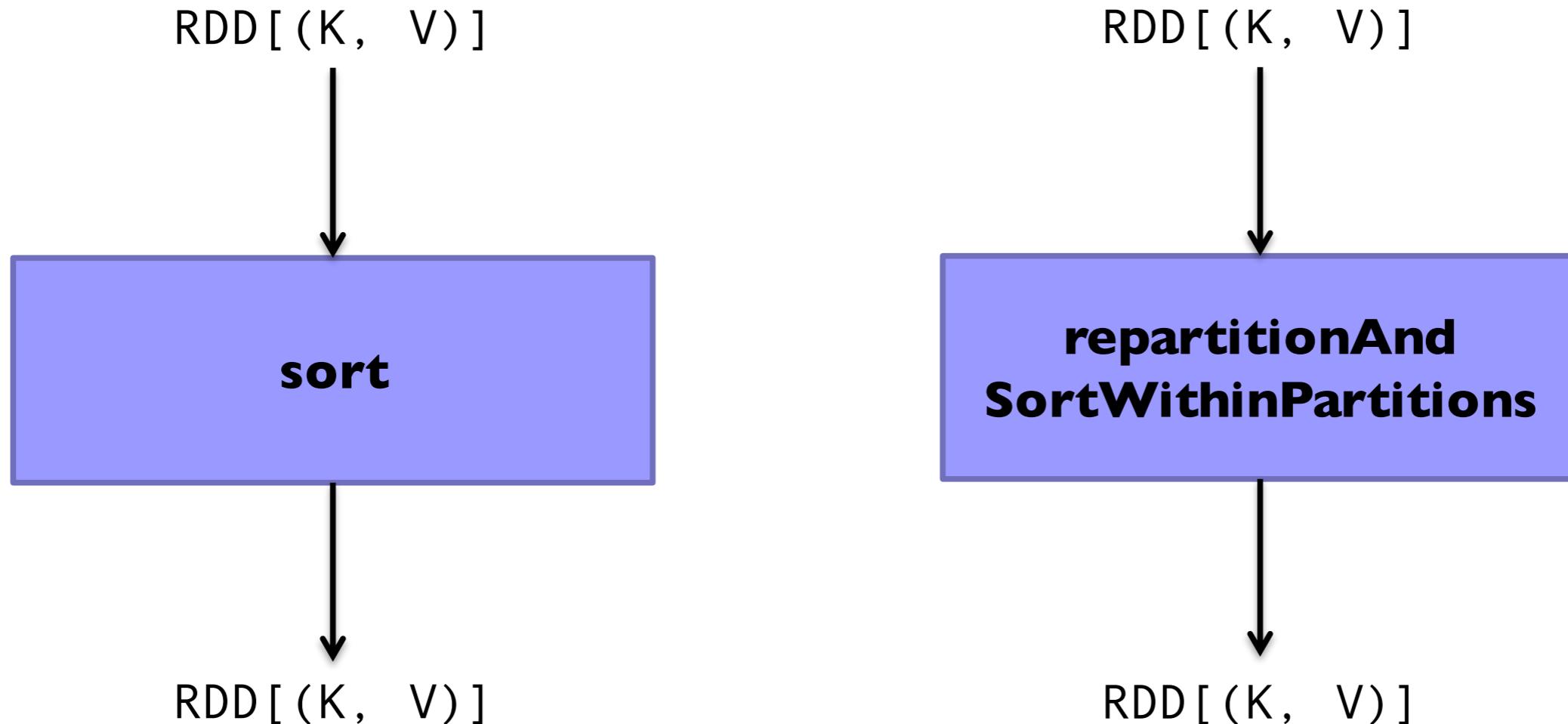
Not meant to be exhaustive

Reduce-like operations



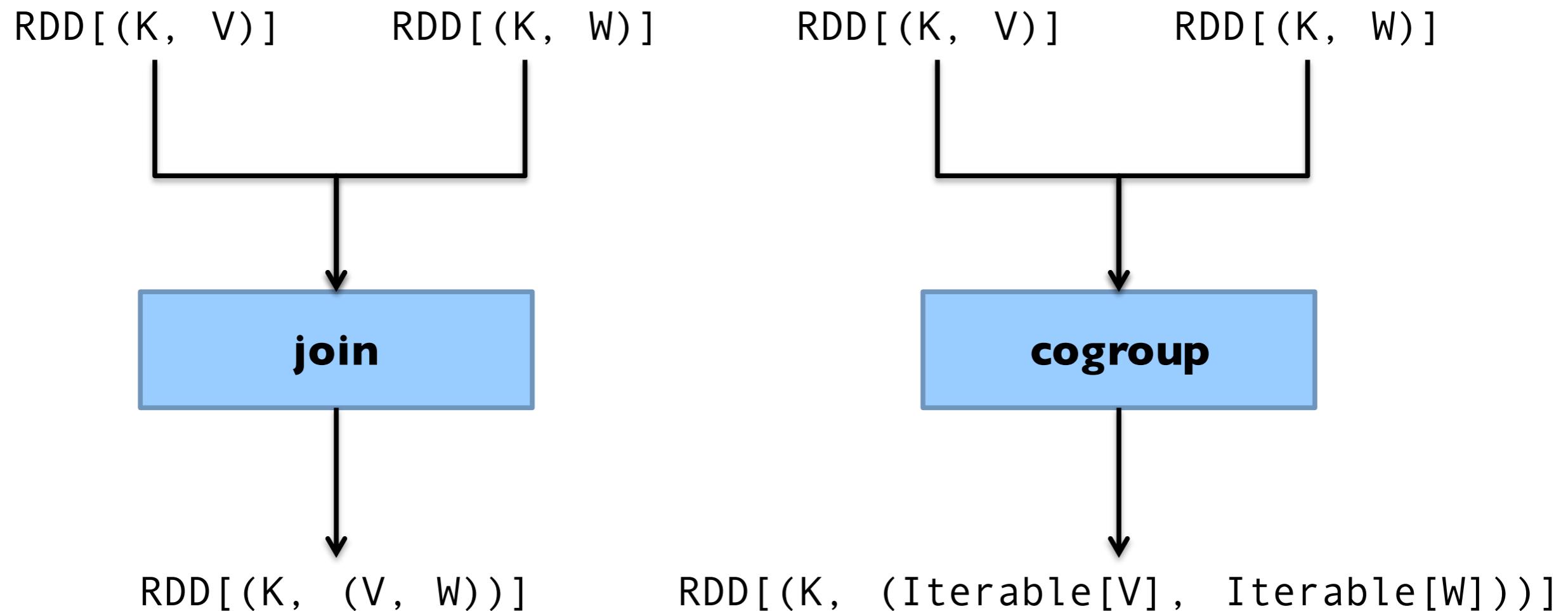
Not meant to be exhaustive

Sort operations



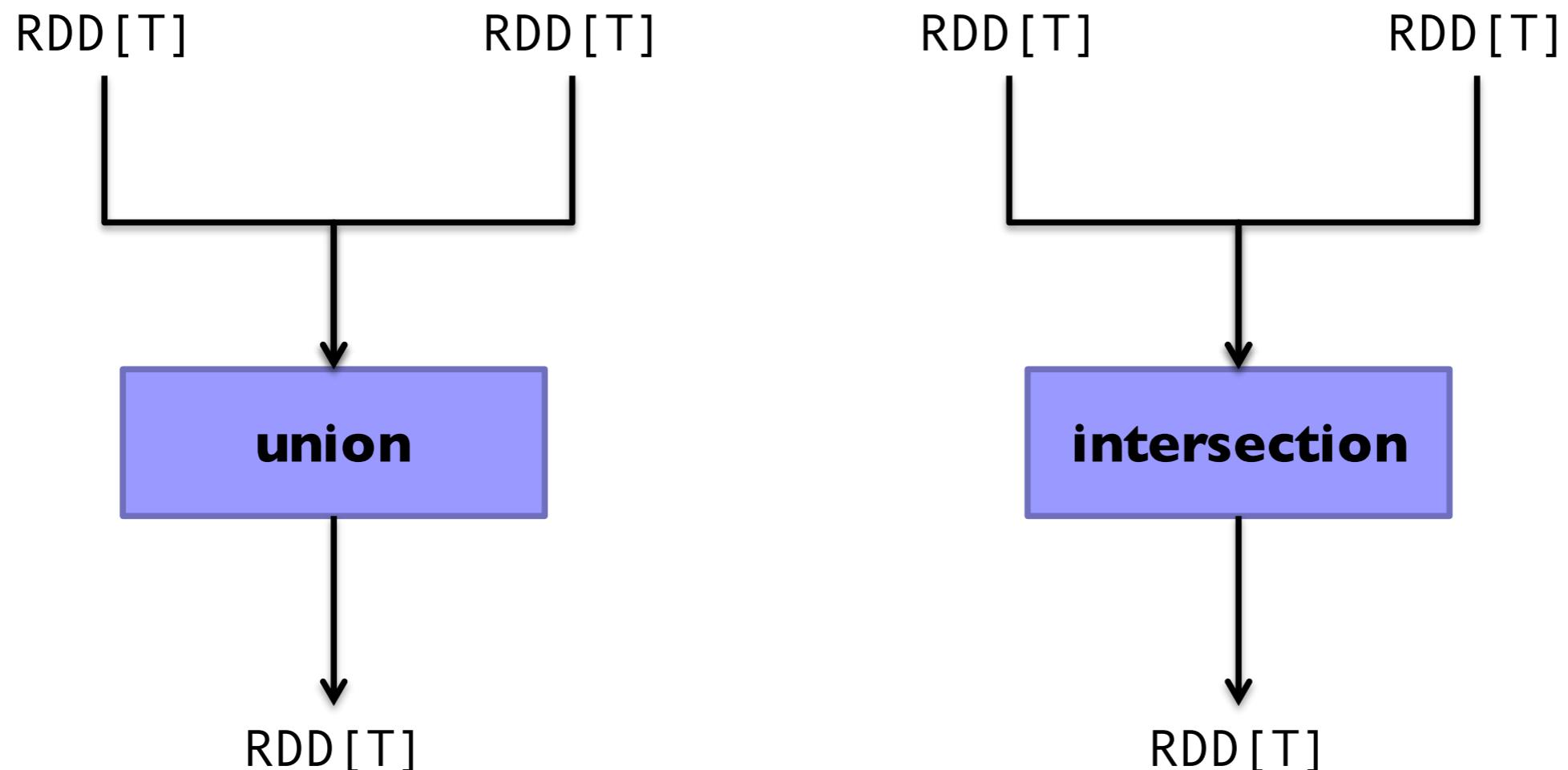
Not meant to be exhaustive

Join-like operations



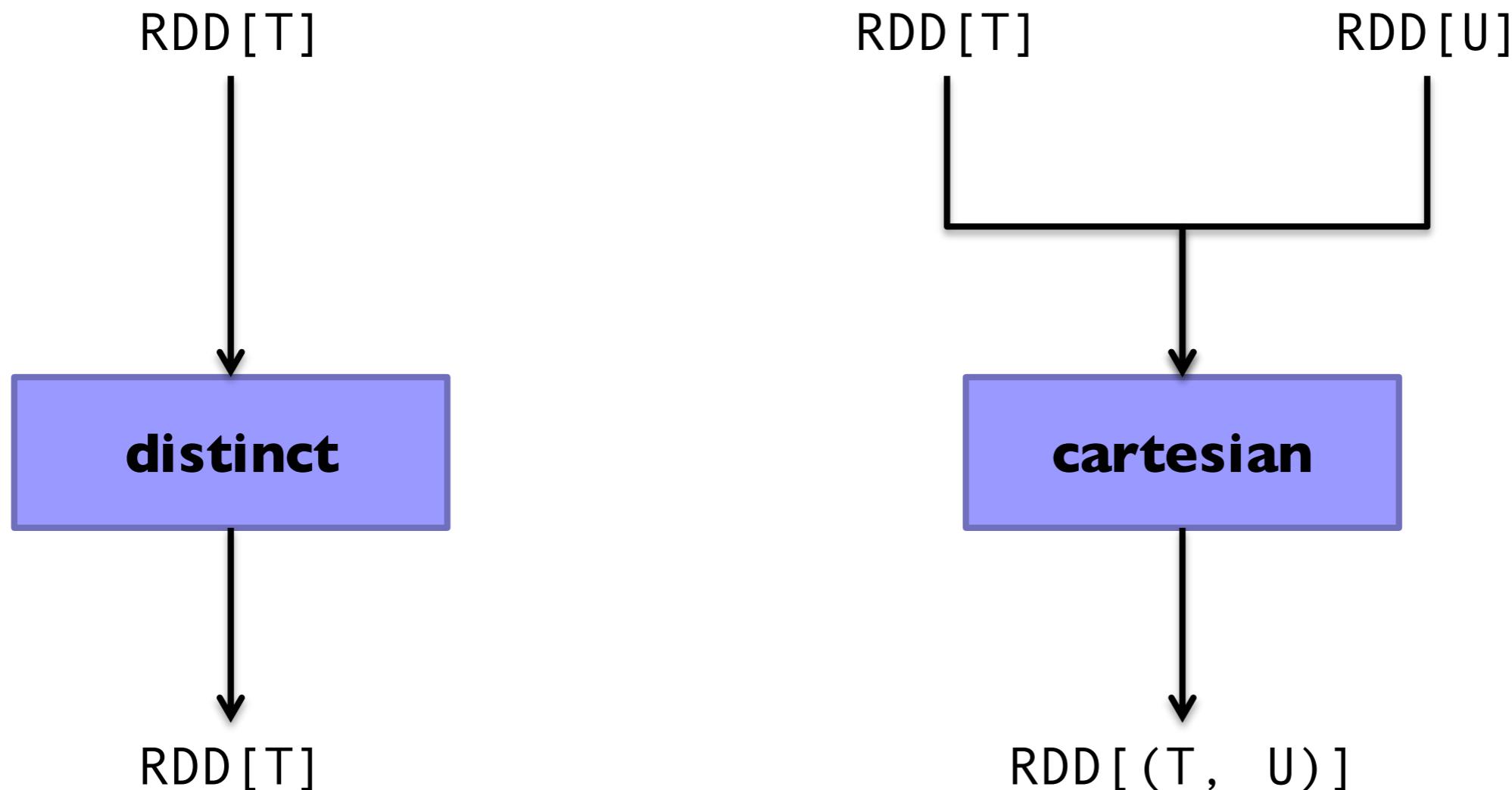
Not meant to be exhaustive

Set-ish operations



Not meant to be exhaustive

Set-ish operations



Not meant to be exhaustive

Spark WordCount: 3-liner

```
text_file = sc.textFile("hdfs://...")  
counts = text_file.flatMap(lambda line: line.split(" ")) \  
    .map(lambda word: (word, 1)) \  
    .reduceByKey(lambda a, b: a + b)  
counts.saveAsTextFile("hdfs://...")
```

This?

```
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();  
        String[] words = line.trim().split("\\s+");  
        for (String w: words) {  
            WORD.set(w);  
            context.write(WORD, ONE);  
        }  
    }  
  
    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 {  
            int sum = 0;  
            for (IntWritable v: values) {  
                sum += v.get();  
            }  
            SUM.set(sum);  
            context.write(key, SUM);  
        }  
  
        Configuration conf = getConf();  
        Job job = Job.getInstance(conf);  
        job.setJobName("Word Count");  
        job.setJarByClass(WordCount.class);  
        job.setNumReduceTasks(reduceTasks);  
        FileInputFormat.setInputPaths(job, new Path(inputPath));  
        FileOutputFormat.setOutputPath(job, new Path(outputPath));  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
        job.setMapperClass(WordCountMapper.class);  
        job.setCombinerClass(WordCountReducer.class);  
        job.setReducerClass(WordCountReducer.class);  
        System.exit(job.waitForCompletion(true) ? 0: 1);  
    }
```

Or this?

Don't focus on Java
verbosity!

Credits

Slides are adapted from Prof. Jimmy Lin's slides at the University of Waterloo