UNIVERSITY OF WATERLOO

Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

Week 3: From MapReduce to Spark (1/2)

January 19, 2016

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

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

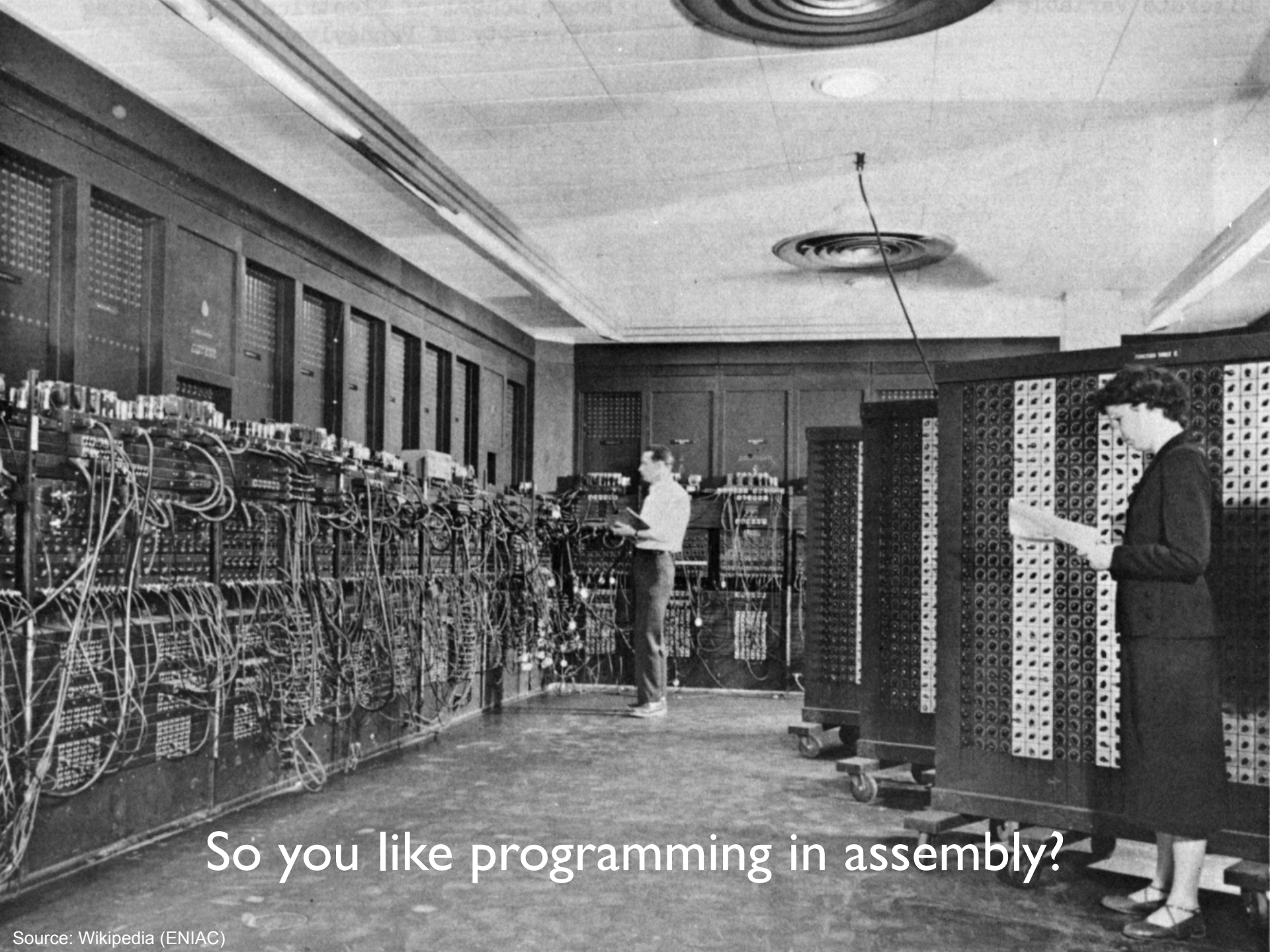
This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See http://creativecommons.org/licenses/by-nc-sa/3.0/us/ for details

Source: Wikipedia (The Scream)

Debugging at Scale

- Works on small datasets, won't scale… why?
  - Memory management issues (buffering and object creation)
  - Too much intermediate data
  - Mangled input records
- Real-world data is messy!
  - There's no such thing as "consistent data"
  - Watch out for corner cases
  - Isolate unexpected behavior, bring local


The datacenter *is* the computer!
What's the instruction set?
Source: Google



Source: Wikipedia (ENIAC)

Source: Wikipedia (DeLorean time machine)

What's the solution?

Design a higher-level language
Write a compiler

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

What we really need is SQL!

Answer:

What we really need is a scripting language!

Answer:

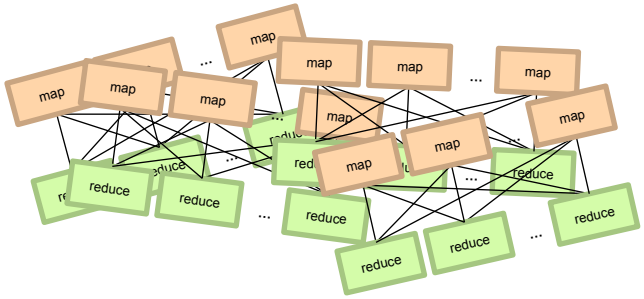
facebook

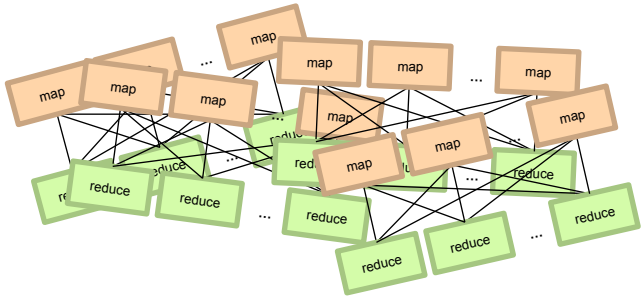
YAHOO!

SQL

Pig Scripts

Aside: Why not just use a database?





Both open-source projects today!

Story for another day….

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."

Source: Wikipedia (Pig)

Pig: Example

Task: Find the top 10 most visited pages in each category

Visits

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

URL Info

Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

Pig Slides adapted from Olston et al. (SIGMOD 2008)

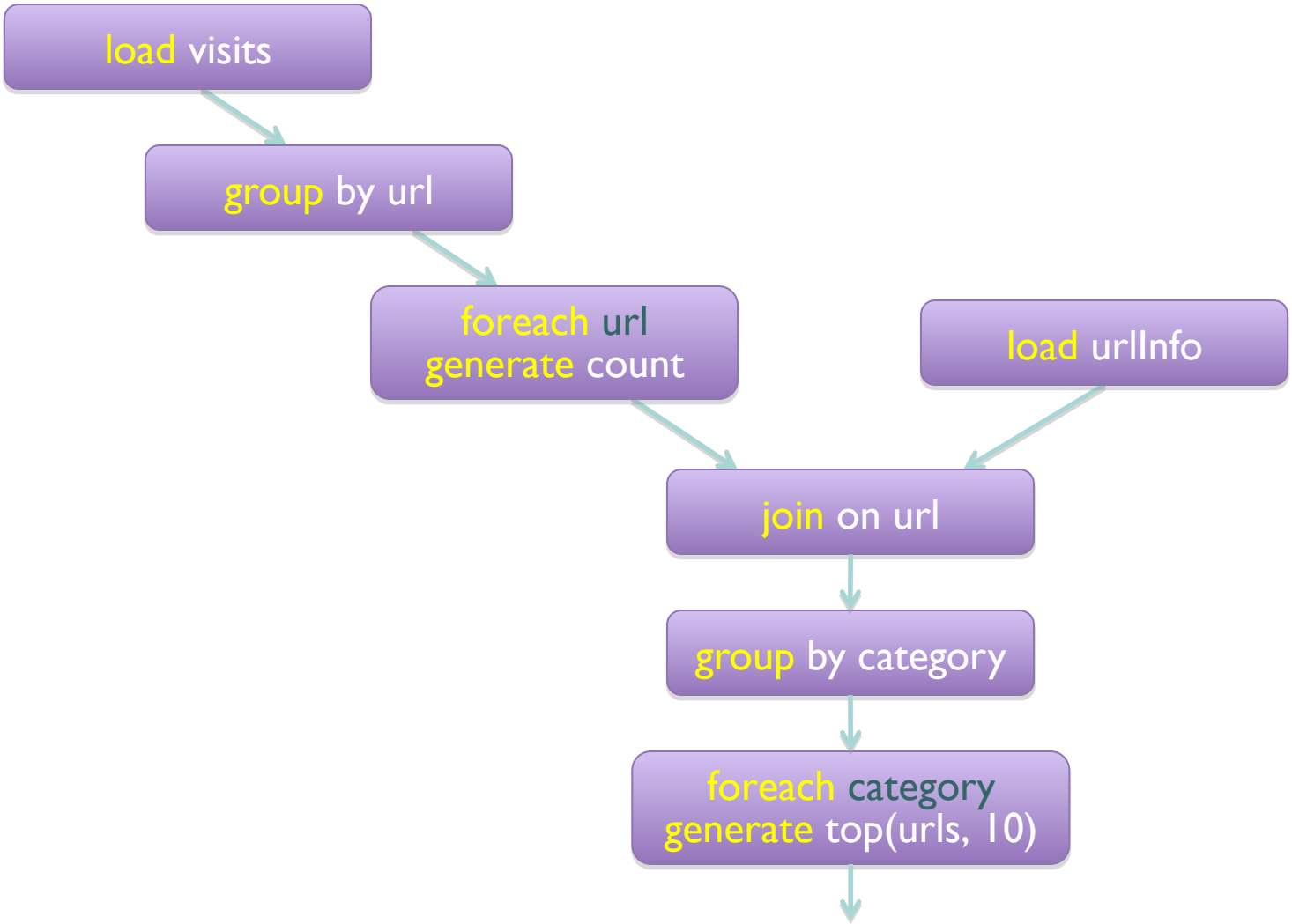
Pig: Example Script

```
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';
```

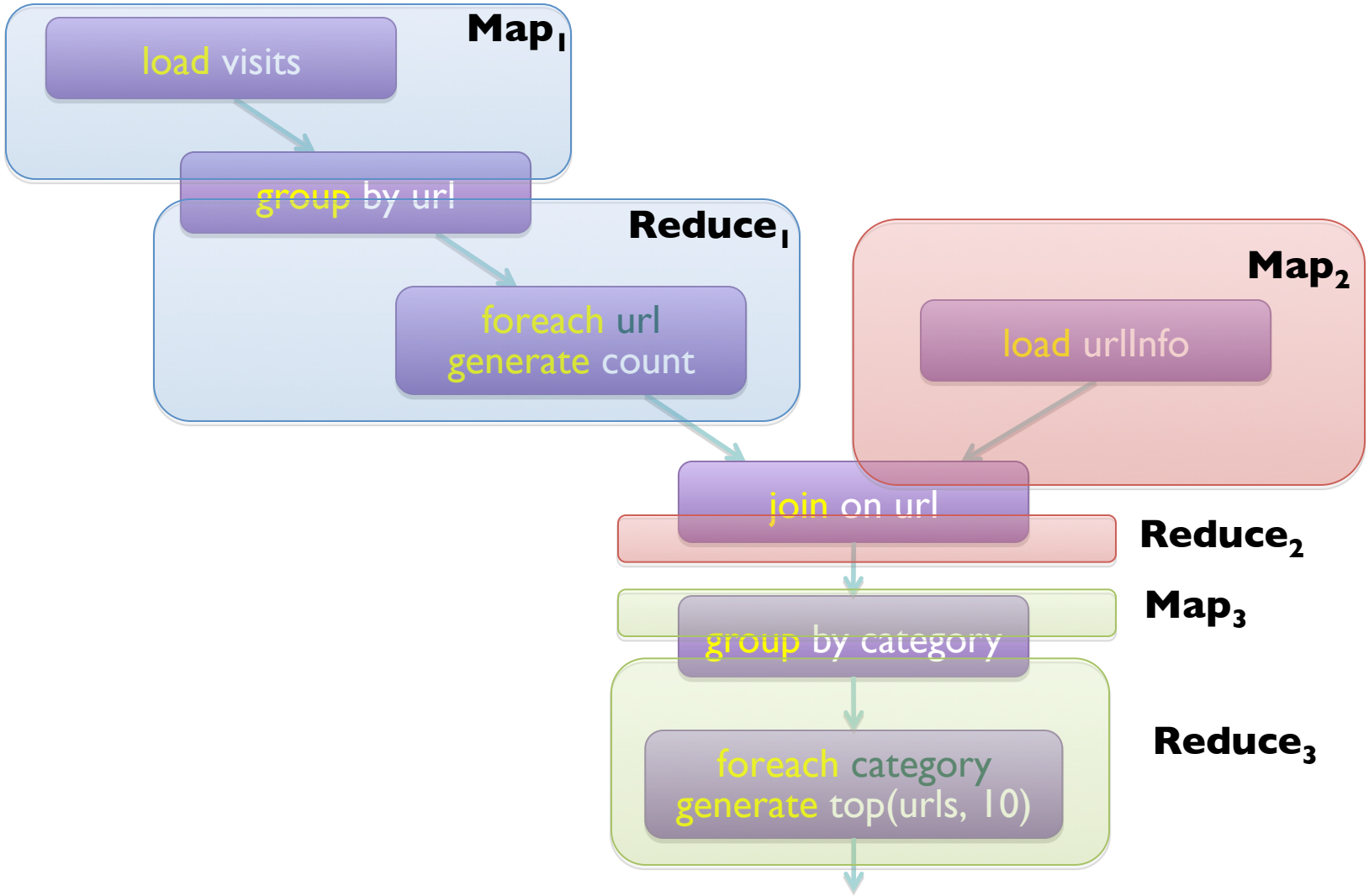
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Pig Query Plan



Pig Slides adapted from Olston et al. (SIGMOD 2008)

MapReduce Execution



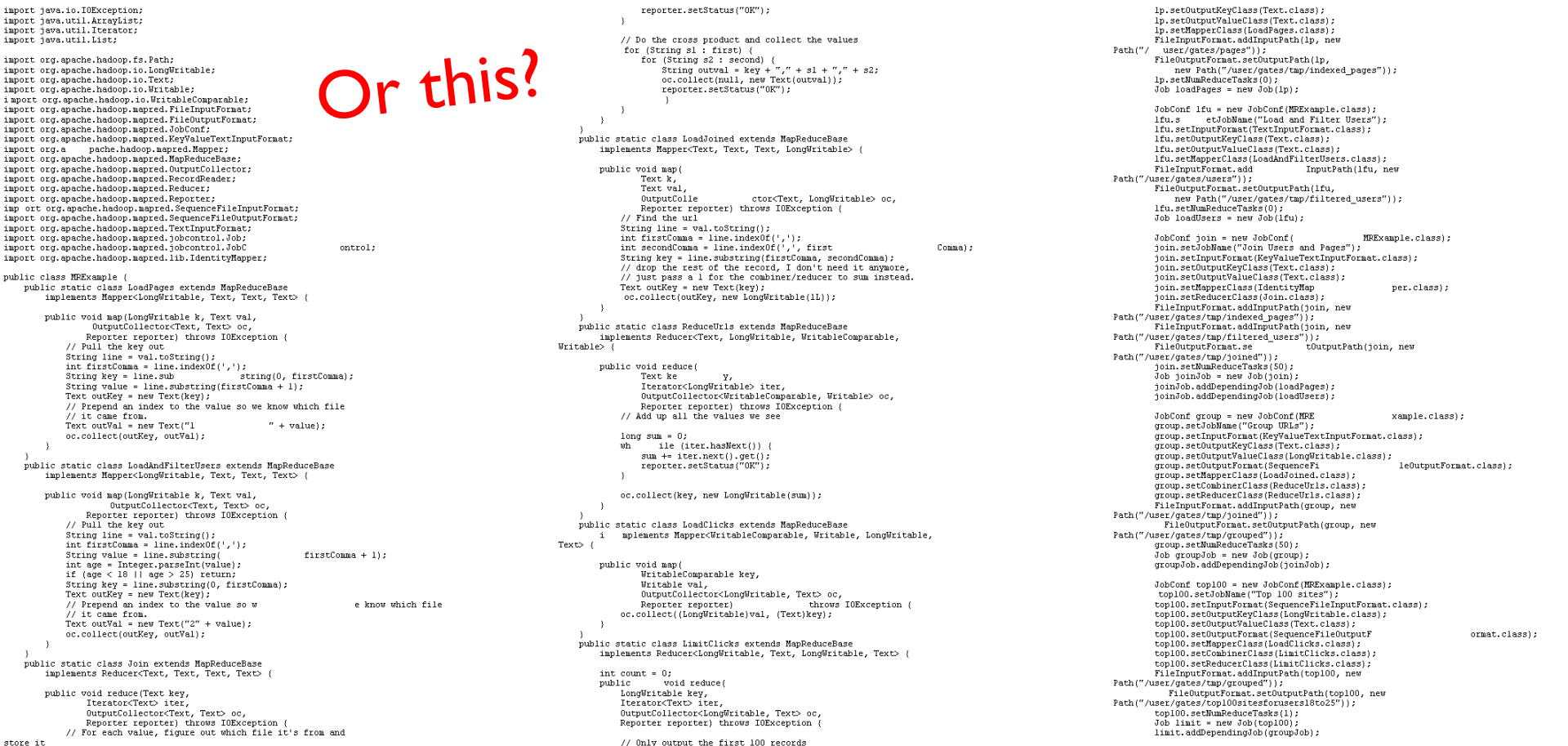
Pig Slides adapted from Olston et al. (SIGMOD 2008)

```
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’;
```

This?

Or this?



```
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;
i mport 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.a     pache.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;
imp ort 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("l                  " + 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
                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,
                    OutputColle          ctor<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 ke         y,
                    Iterator<LongWritable> iter,
                    OutputCollector<WritableComparable, Writable> oc,
                    Reporter reporter) throws IOException {
                // Add up all the values we see

                long sum = 0;
                wh     ile (iter.hasNext()) {
                    sum += iter.next().get();
                    reporter.setStatus("OK");
                }

                oc.collect(key, new LongWritable(sum));
            }
        }
        public static class LoadClicks extends MapReduceBase
            i   mplements 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
                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.s     etJobName("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);
```

But isn't Pig slower?

Sure, but c can be slower than assembly too…

Pig: Basics

- Sequence of statements manipulating relations (aliases)
- Data model
  - atoms
  - tuples
  - bags
  - maps
  - json

Pig: Common Operations

- LOAD: load data (from HDFS)
- FOREACH … GENERATE: per tuple processing
- FILTER: discard unwanted tuples
- GROUP/COGROUP: group tuples
- JOIN: relational join
- STORE: store data (to HDFS)

"map"

"reduce"

Pig: GROUPing

```
A = LOAD 'myfile.txt' AS (f1: int, f2: int, f3: int);

    (1, 2, 3)
    (4, 2, 1)
    (8, 3, 4)
    (4, 3, 3)
    (7, 2, 5)
    (8, 4, 3)



X = GROUP A BY f1;

    (1, {(1, 2, 3)})
    (4, {(4, 2, 1), (4, 3, 3)})
    (7, {(7, 2, 5)})
    (8, {(8, 3, 4), (8, 4, 3)})
```

Pig: COGROUPing

```
A:
(1, 2, 3)
(4, 2, 1)
(8, 3, 4)
(4, 3, 3)
(7, 2, 5)
(8, 4, 3)
```

```
B:
(2, 4)
(8, 9)
(1, 3)
(2, 7)
(2, 9)
(4, 6)
(4, 9)
```

```
X = COGROUP A BY $0, B BY $0;
```

```
(1, {(1, 2, 3)}, {(1, 3)})
(2, {}, {(2, 4), (2, 7), (2, 9)})
(4, {(4, 2, 1), (4, 3, 3)}, {(4, 6),(4, 9)})
(7, {(7, 2, 5)}, {})
(8, {(8, 3, 4), (8, 4, 3)}, {(8, 9)})
```

Pig: JOINing

```
A:
(1, 2, 3)
(4, 2, 1)
(8, 3, 4)
(4, 3, 3)
(7, 2, 5)
(8, 4, 3)
```

```
B:
(2, 4)
(8, 9)
(1, 3)
(2, 7)
(2, 9)
(4, 6)
(4, 9)
```

```
X = JOIN A BY $0, B BY $0;
```

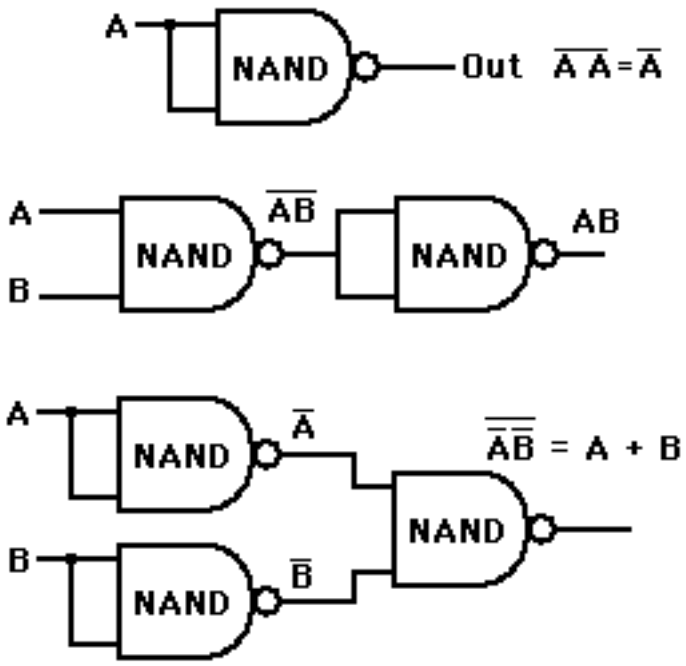
```
(1,2,3,1,3)
(4,2,1,4,6)
(4,3,3,4,6)
(4,2,1,4,9)
(4,3,3,4,9)
(8,3,4,8,9)
(8,4,3,8,9)
```

Pig UDFs

- User-defined functions:
  - Java
  - Python
  - JavaScript
  - Ruby
  - …
- UDFs make Pig arbitrarily extensible
  - Express "core" computations in UDFs
  - Take advantage of Pig as glue code for scale-out plumbing

Source: Google

Analogy: NAND Gates are universal



Let's design a data processing language "from scratch"!

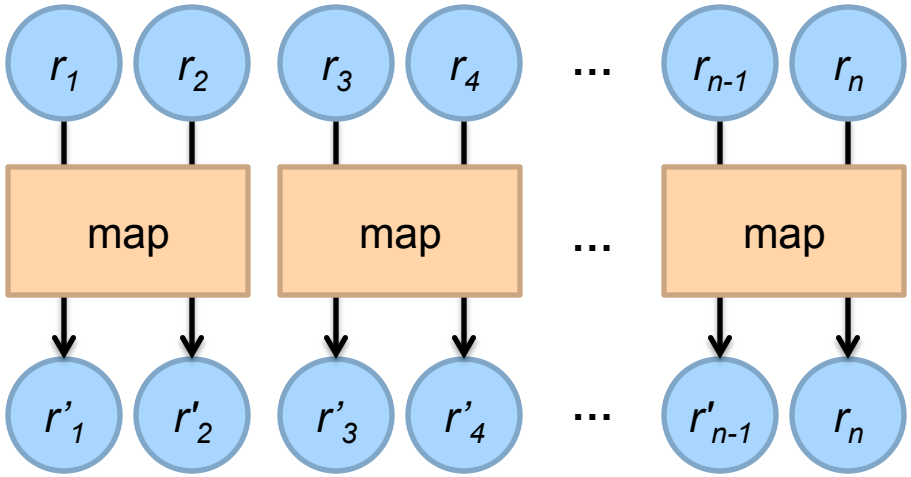
(Why is MapReduce the way it is?)

Data-Parallel Dataflow Languages

We have a collection of records,
want to apply a bunch of transformations
to compute some result

Assumptions: static collection,
records (not necessarily key-value pairs)

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

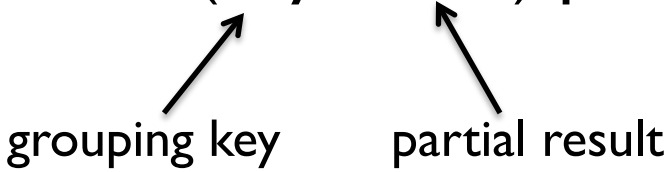


Remarks: Easy to parallelize maps,
record to "mapper" assignment is an implementation detail

Map alone isn't enough
(If we want more than embarrassingly parallel processing)

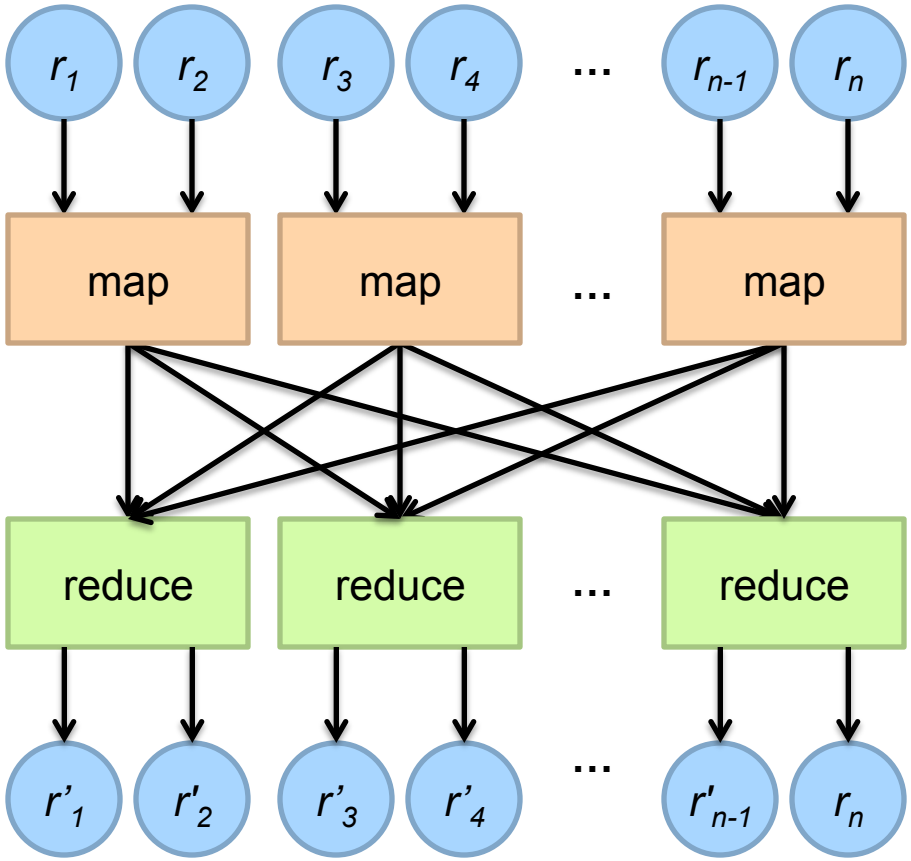
We need a way to group partial results

Intermediate (key, value) pairs

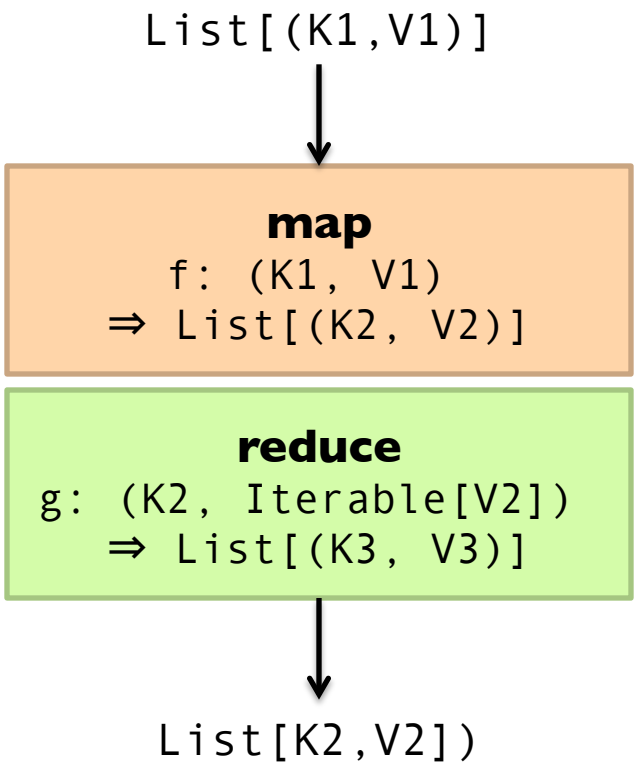


For each key, we can apply some computation

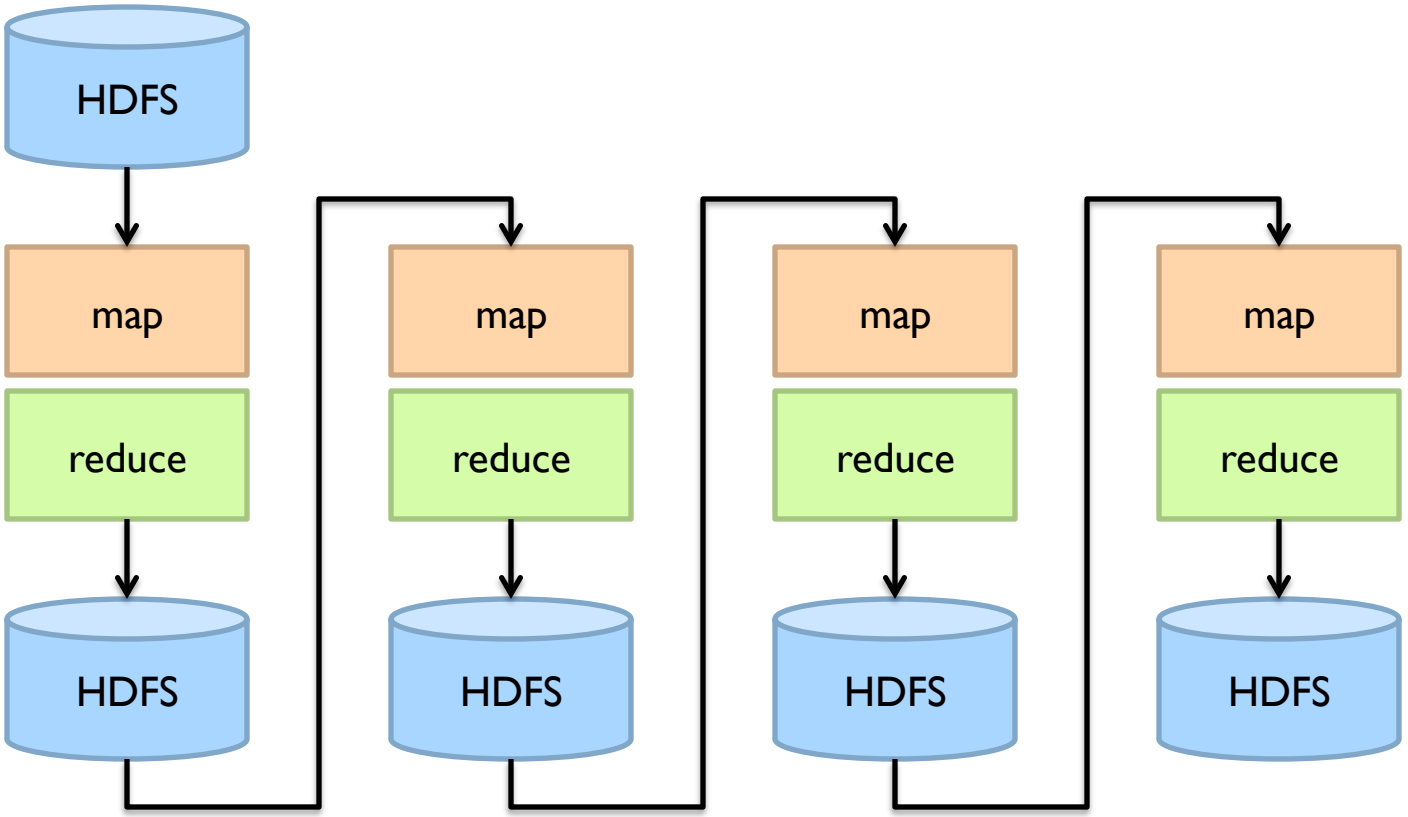
MapReduce


r_1
r_2
r_3
r_4
...
r_{n-1}
r_n
map
map
...
map
reduce
reduce
...
reduce
r'_1
r'_2
r'_3
r'_4
...
r'_{n-1}
r_n

MapReduce

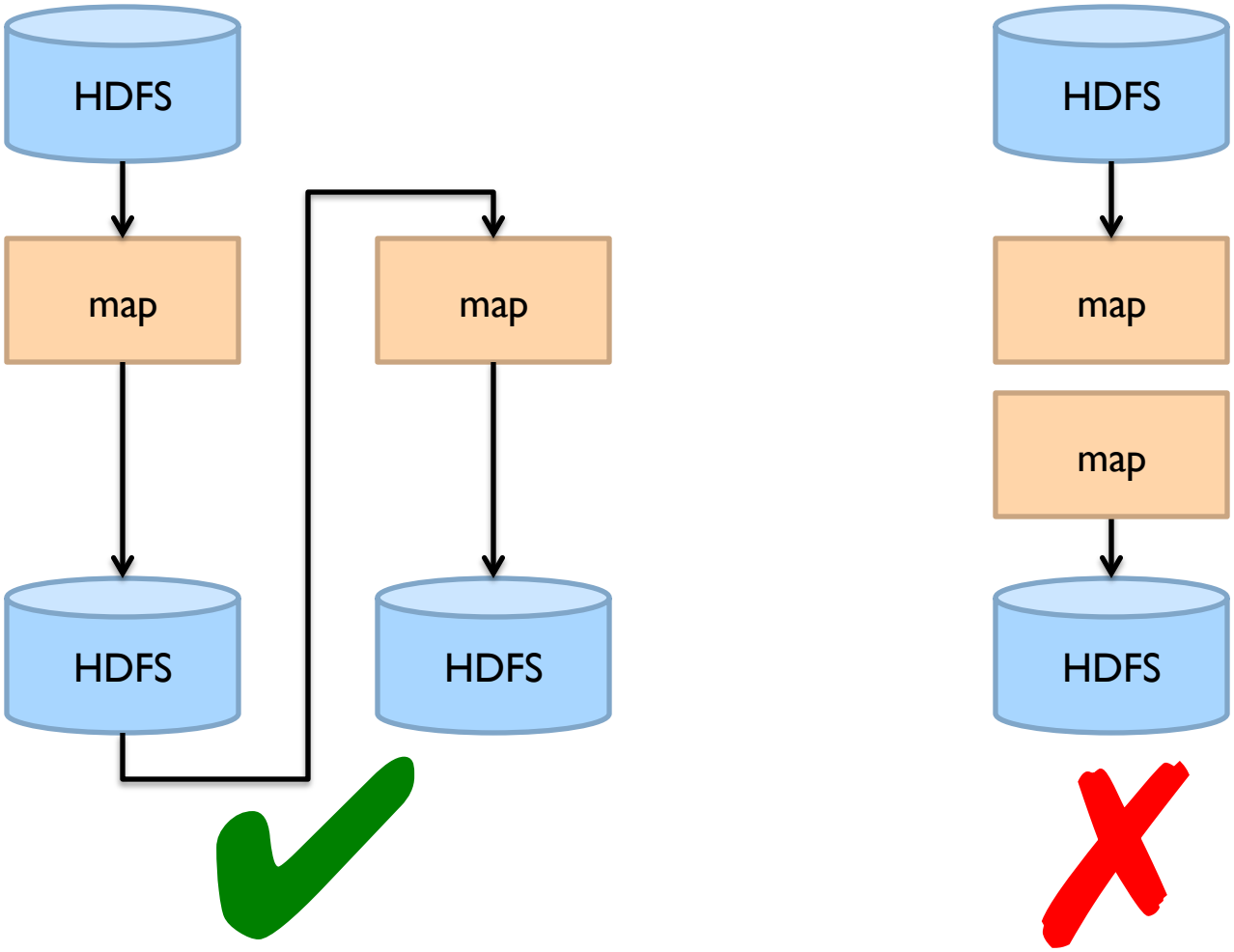

List[(K1,V1)]
map
f: (K1, V1)
⇒ List[(K2, V2)]
reduce
g: (K2, Iterable[V2])
⇒ List[(K3, V3)]
List[K2,V2])

MapReduce Workflows

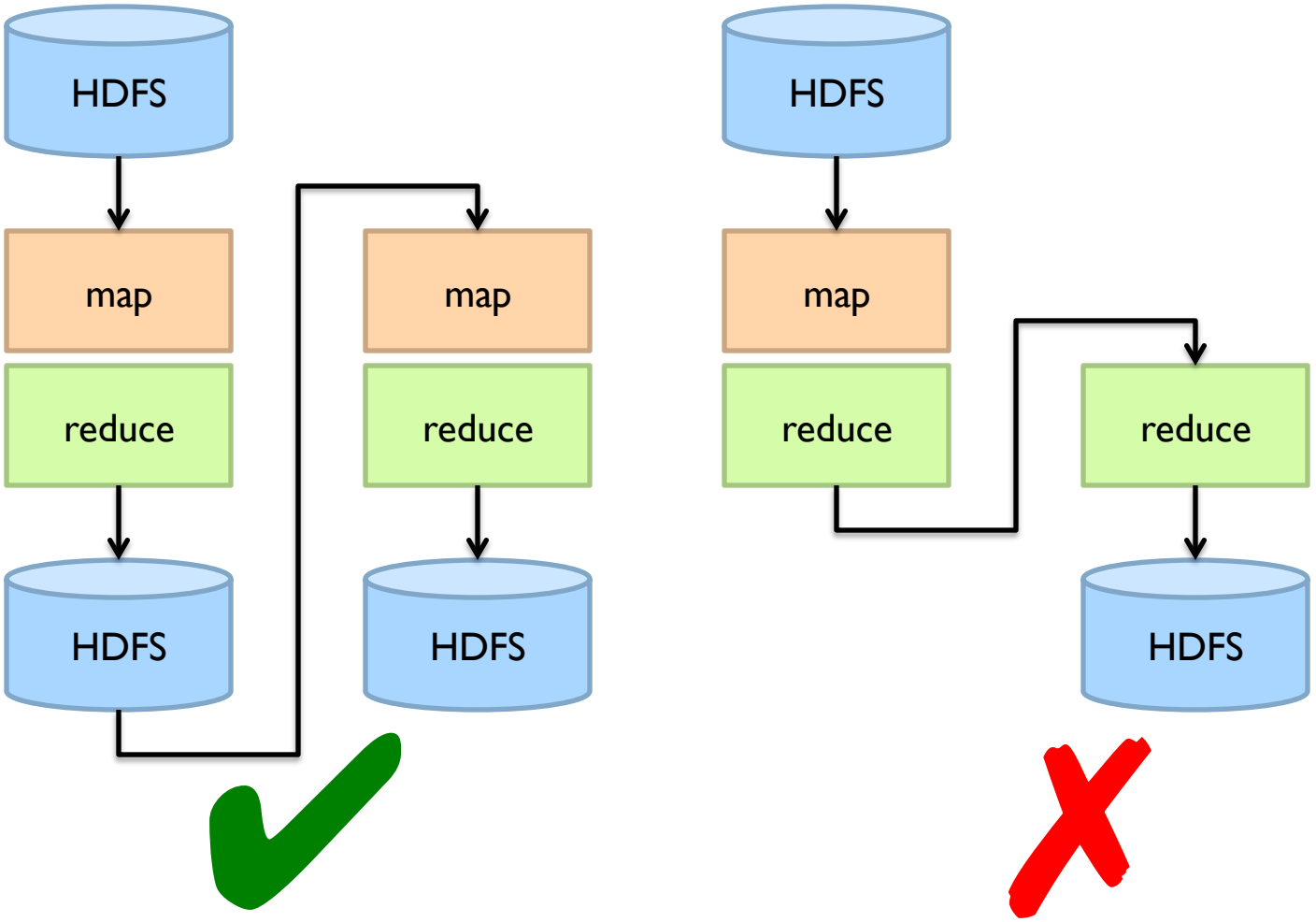


What's wrong?

Want MM...?

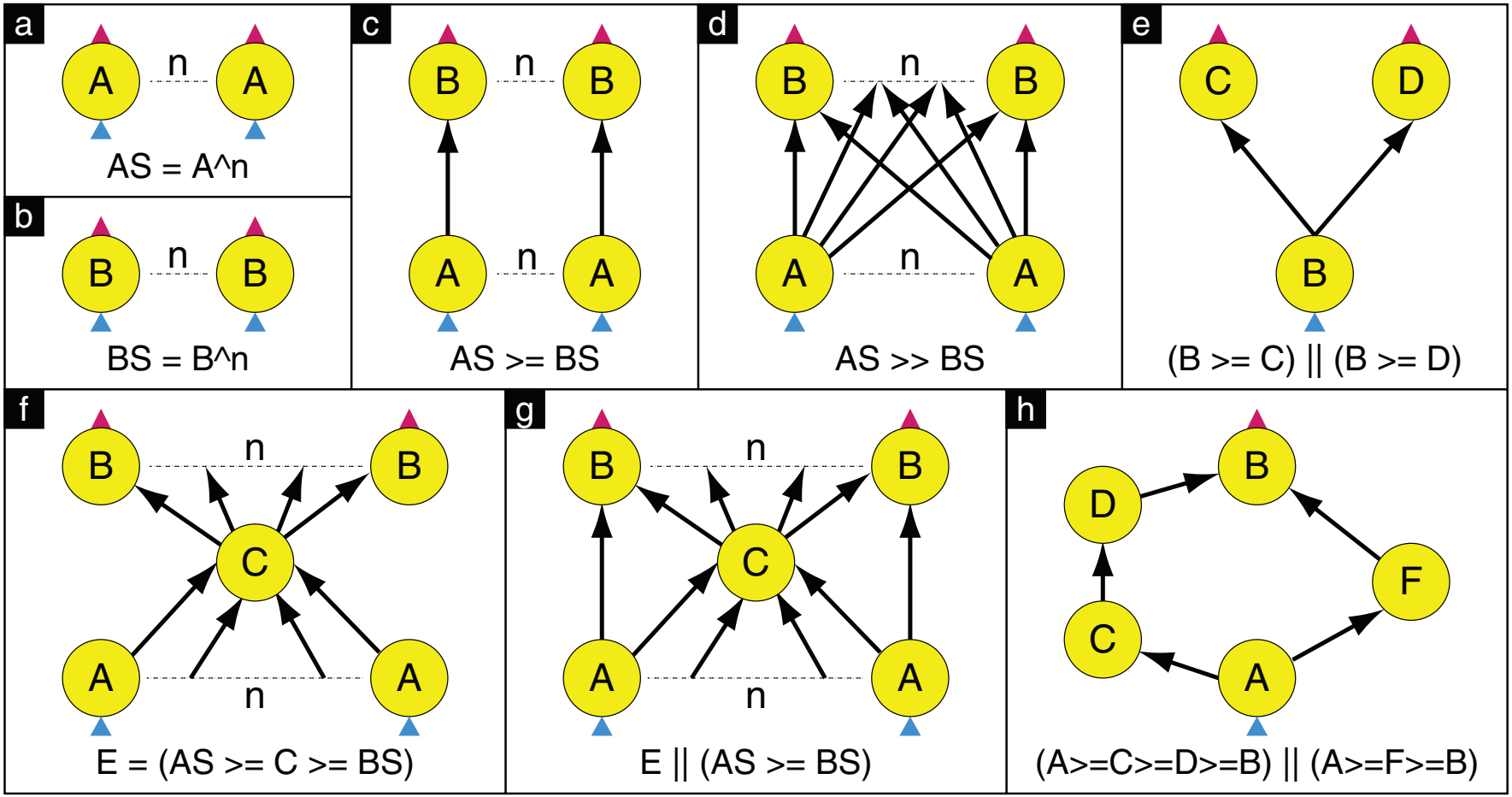


Want MRR?


HDFS
map
reduce
HDFS
map
reduce
HDFS
HDFS
map
reduce
reduce
HDFS

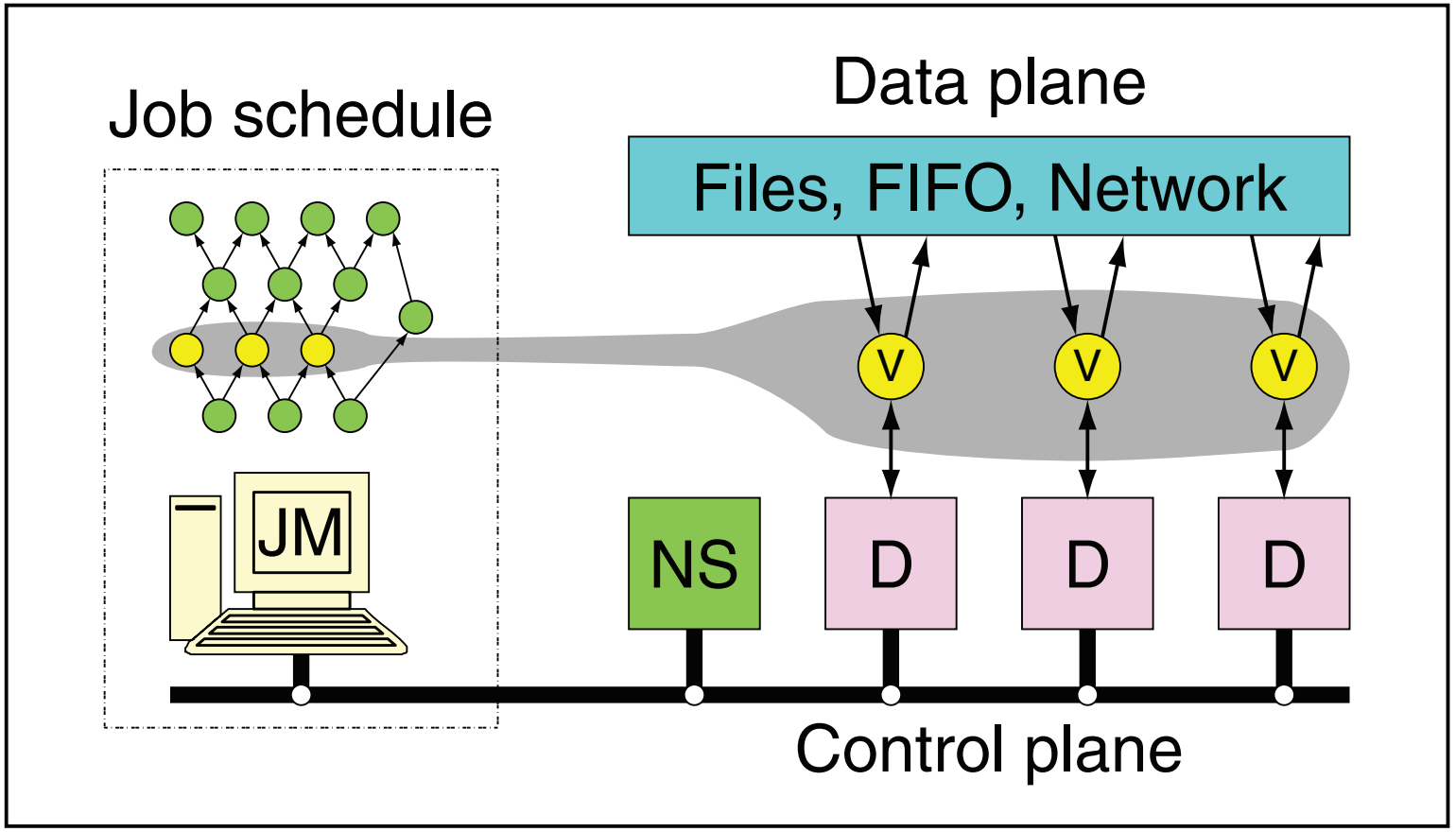

The datacenter *is* the computer!
Let's enrich the instruction set!
Source: Google

Dryad: Graph Operators



Source: Isard et al. (2007) Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys.

Dryad: Architecture



The Dryad system organization. The job manager (JM) consults the name server (NS) to discover the list of available computers. It maintains the job graph and schedules running vertices (V) as computers become available using the daemon (D) as a proxy. Vertices exchange data through files, TCP pipes, or shared-memory channels. The shaded bar indicates the vertices in the job that are currently running.

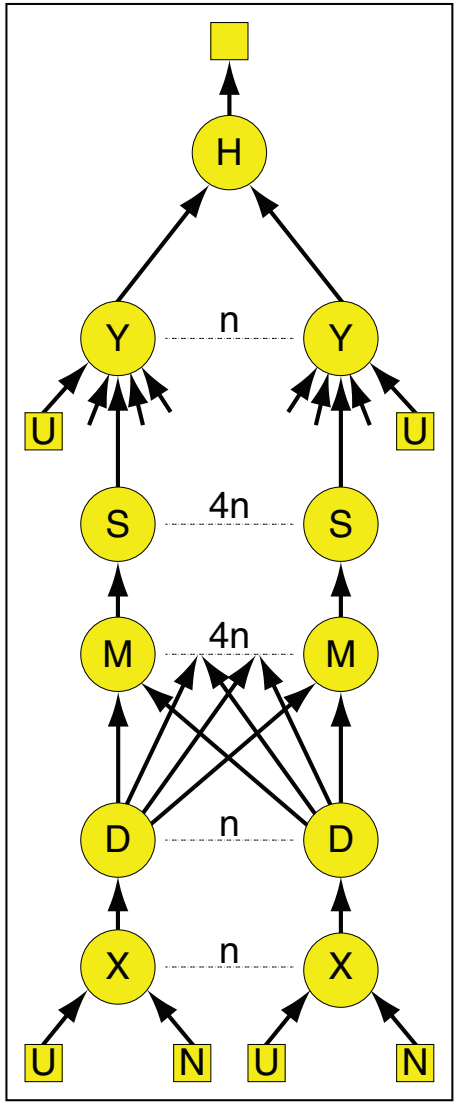
Source: Isard et al. (2007) Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys.

Dryad: Cool Tricks

- Channel: abstraction for vertex-to-vertex communication
  - File
  - TCP pipe
  - Shared memory
- Runtime graph refinement
  - Size of input is not known until runtime
  - Automatically rewrite graph based on invariant properties

Source: Isard et al. (2007) Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys.

Dryad: Sample Program




```
GraphBuilder XSet = moduleX^N;
GraphBuilder DSet = moduleD^N;
GraphBuilder MSet = moduleM^(N*4);
GraphBuilder SSet = moduleS^(N*4);
GraphBuilder YSet = moduleY^N;
GraphBuilder HSet = moduleH^1;

GraphBuilder XInputs = (ugriz1 >= XSet) || (neighbor >= XSet);
GraphBuilder YInputs = ugriz2 >= YSet;

GraphBuilder XToY = XSet >= DSet >> MSet >= SSet;
for (i = 0; i < N*4; ++i)
{
    XToY = XToY || (SSet.GetVertex(i) >= YSet.GetVertex(i/4));
}

GraphBuilder YToH = YSet >= HSet;
GraphBuilder HOutputs = HSet >= output;

GraphBuilder final = XInputs || YInputs || XToY || YToH || HOutputs;
```

Source: Isard et al. (2007) Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys.

DryadLINQ

- LINQ = Language INtegrated Query
  - .NET constructs for combining imperative and declarative programming
- Developers write in DryadLINQ
  - Program compiled into computations that run on Dryad

Sound familiar?

Source: Yu et al. (2008) DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. OSDI.

DryadLINQ: Word Count

```
PartitionedTable<LineRecord> inputTable =
  PartitionedTable.Get<LineRecord>(uri);

IQueryable<string> words = inputTable.SelectMany(x => x.line.Split(' '));
IQueryable<IGrouping<string, string>> groups = words.GroupBy(x => x);
IQueryable<Pair> counts = groups.Select(x => new Pair(x.Key, x.Count()));
IQueryable<Pair> ordered = counts.OrderByDescending(x => x.Count);
IQueryable<Pair> top = ordered.Take(k);
```

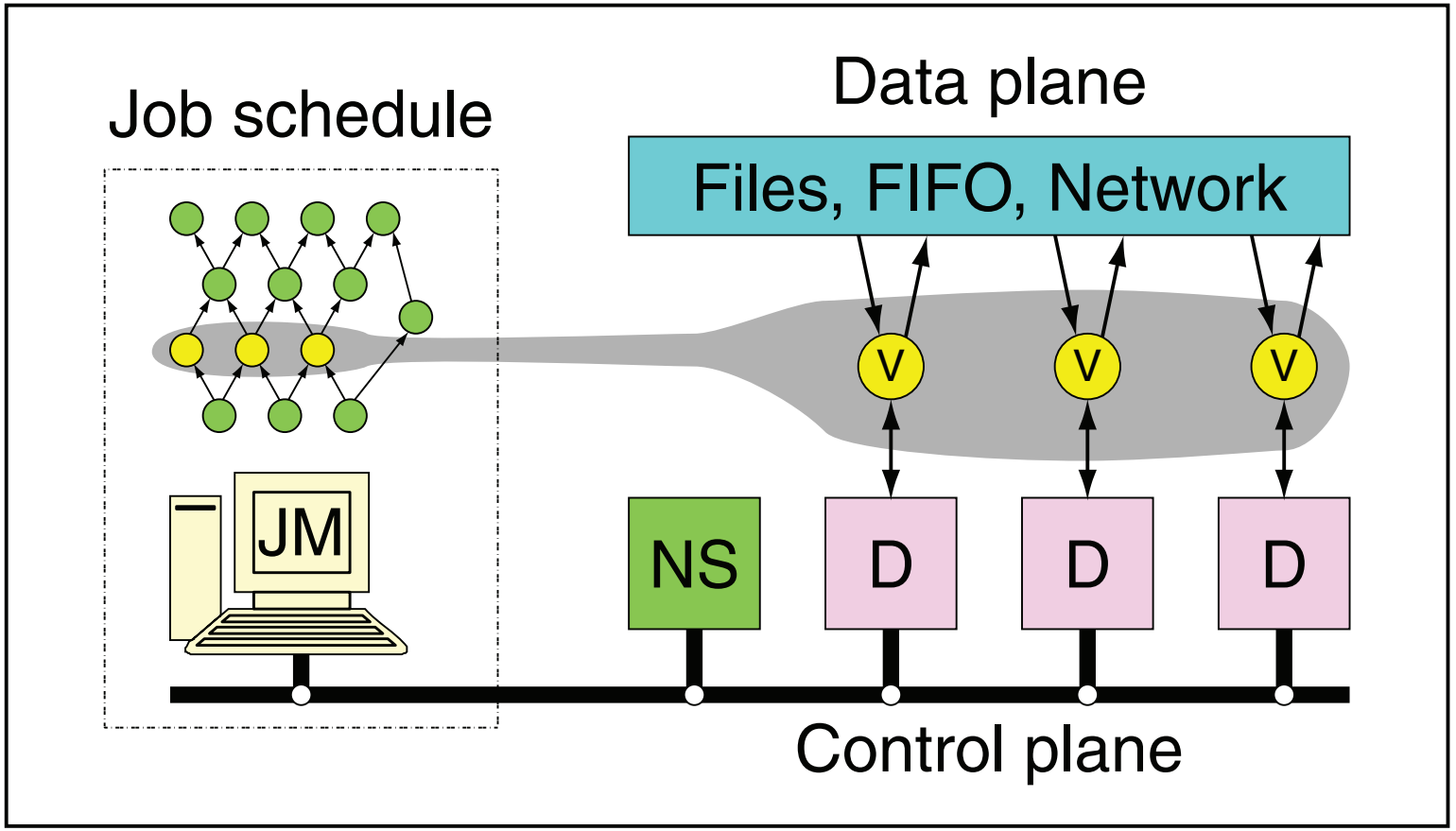
Compare:

```
a = load 'file.txt' as (text: chararray);
b = foreach a generate flatten(TOKENIZE(text)) as term;
c = group b by term;
d = foreach c generate group as term, COUNT(b) as count;

store d into 'cnt';
```

Compare and contrast…

What happened to Dryad?



The Dryad system organization. The job manager (JM) consults the name server (NS) to discover the list of available computers. It maintains the job graph and schedules running vertices (V) as computers become available using the daemon (D) as a proxy. Vertices exchange data through files, TCP pipes, or shared-memory channels. The shaded bar indicates the vertices in the job that are currently running.

Source: Isard et al. (2007) Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys.

Data-Parallel Dataflow Languages

We have a collection of records,
want to apply a bunch of transformations
to compute some result

What are the operators?

MapReduce?
Spark?

Spark

- 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 February 2014
  - Commercial support provided by DataBricks

Spark vs. Hadoop

Google Trends

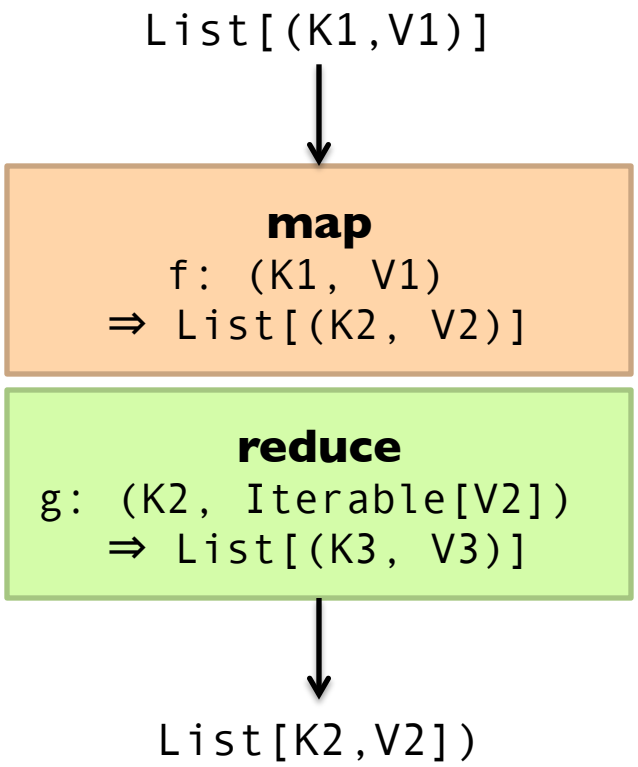
Source: Datanami (2014): http://www.datanami.com/2014/11/21/spark-just-passed-hadoop-popularity-web-heres/

What's an RDD?

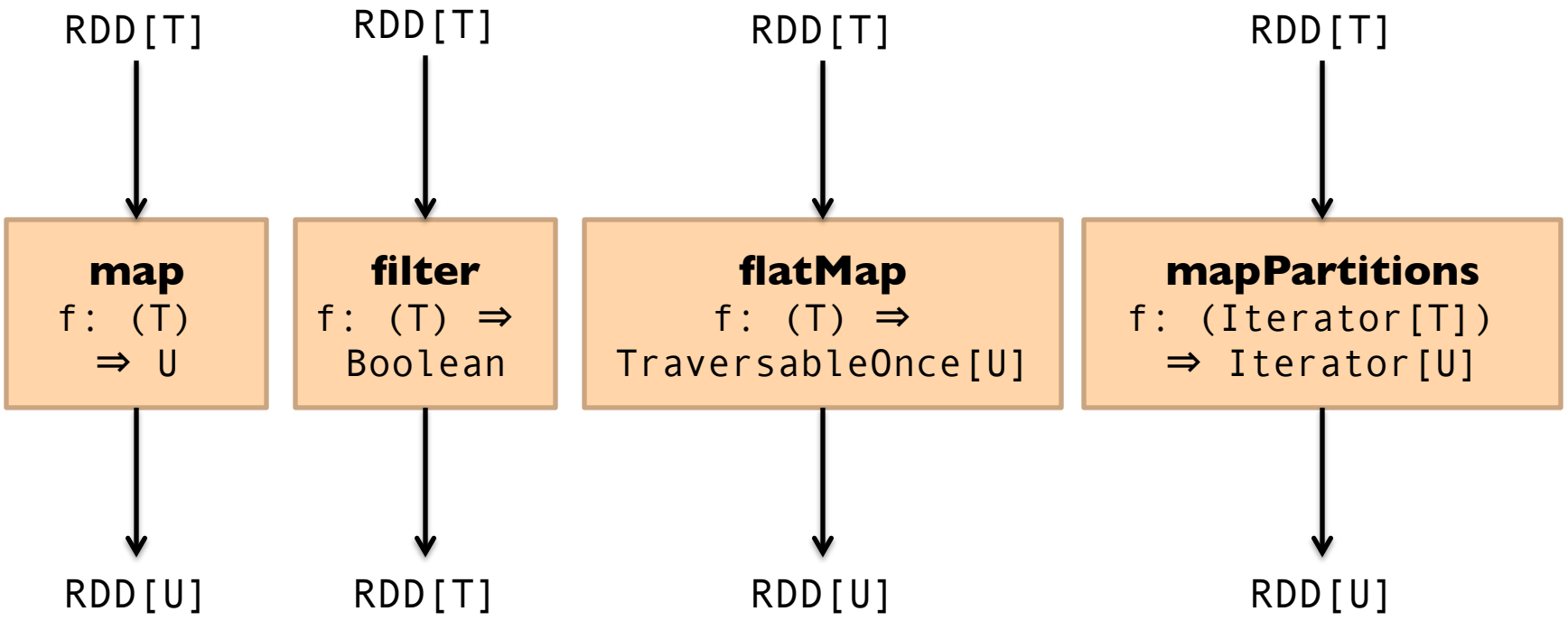
Resilient Distributed Dataset (RDD)

Much more next session…

MapReduce

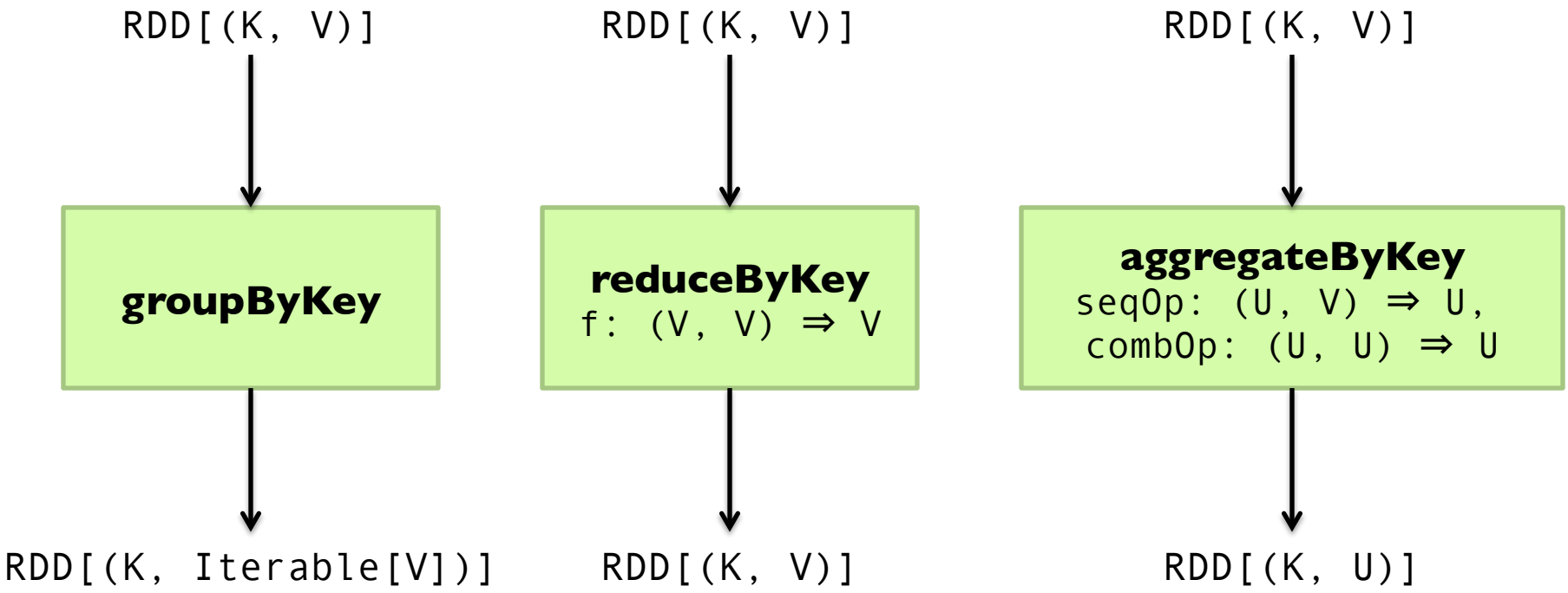

List[(K1,V1)]
map
f: (K1, V1)
⇒ List[(K2, V2)]
reduce
g: (K2, Iterable[V2])
⇒ List[(K3, V3)]
List[K2,V2])

Map-like Operations



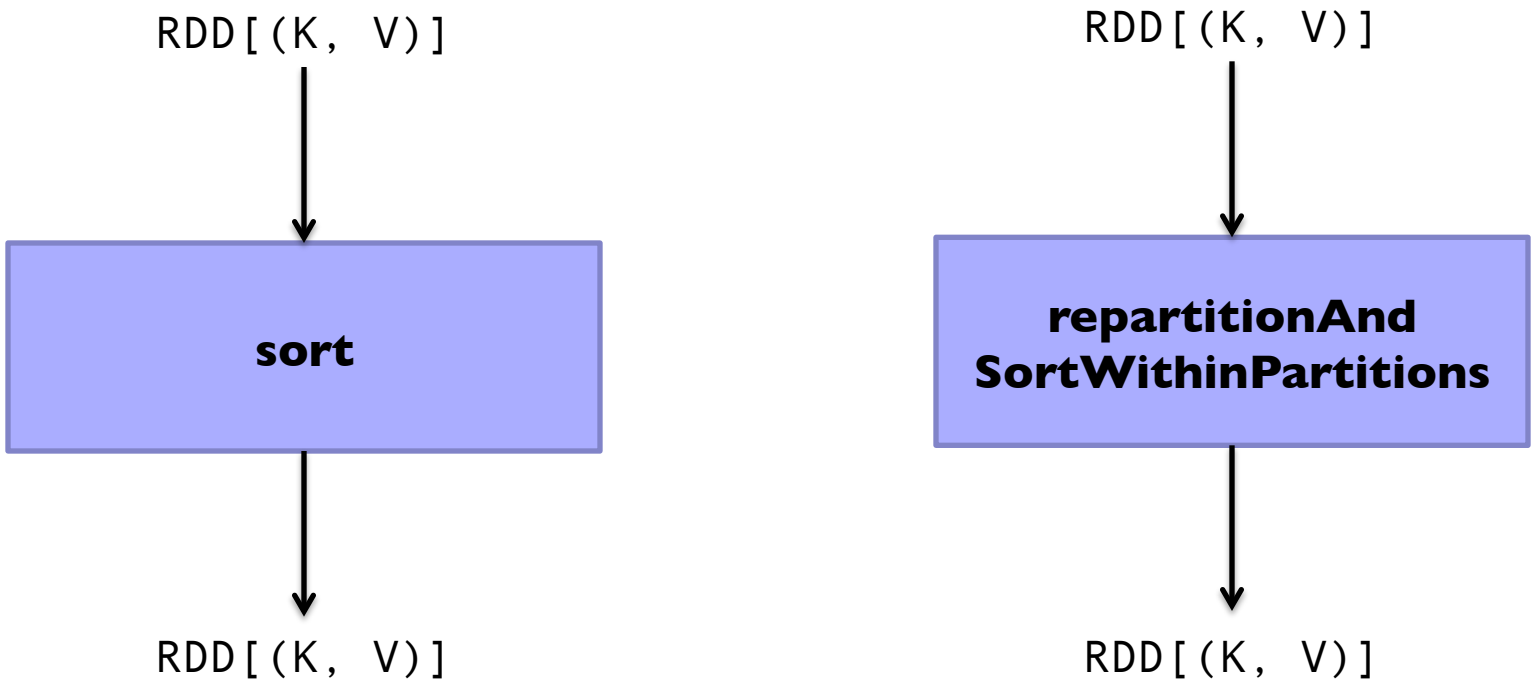
(Not meant to be exhaustive)

Reduce-like Operations



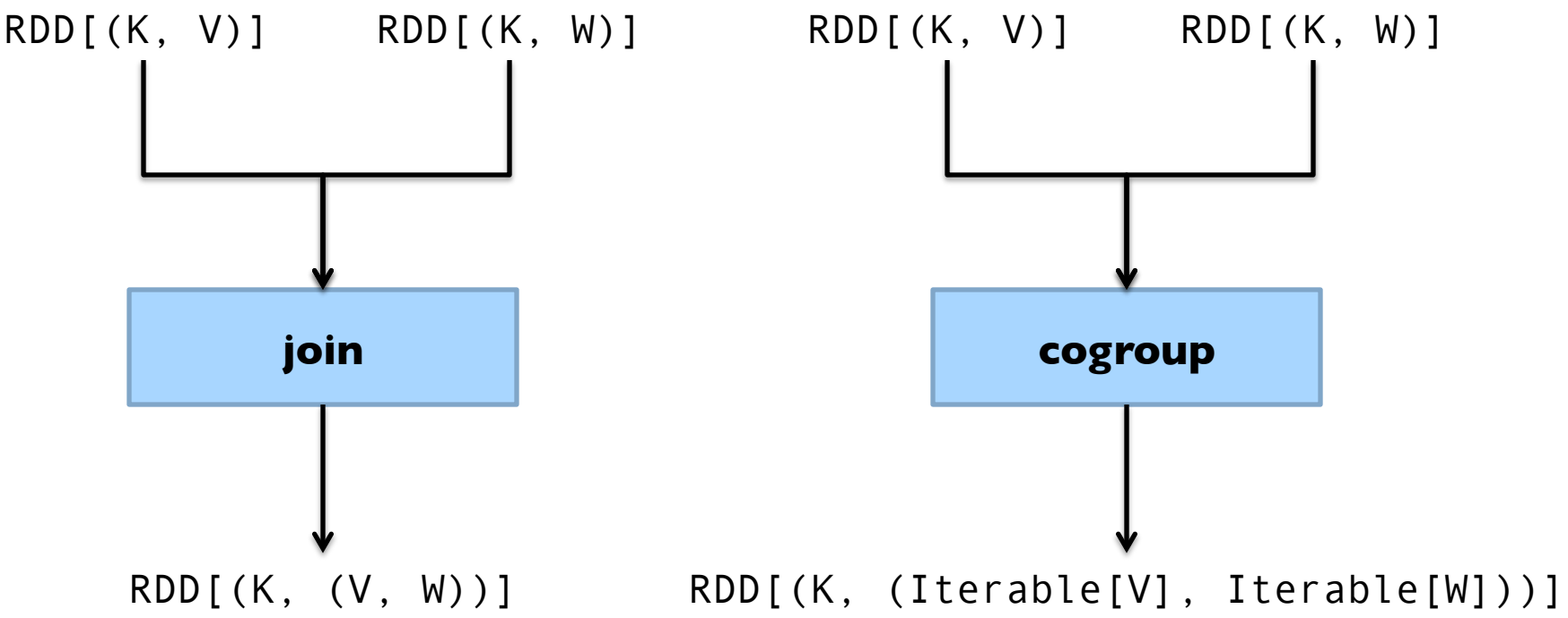
(Not meant to be exhaustive)

Sort Operations



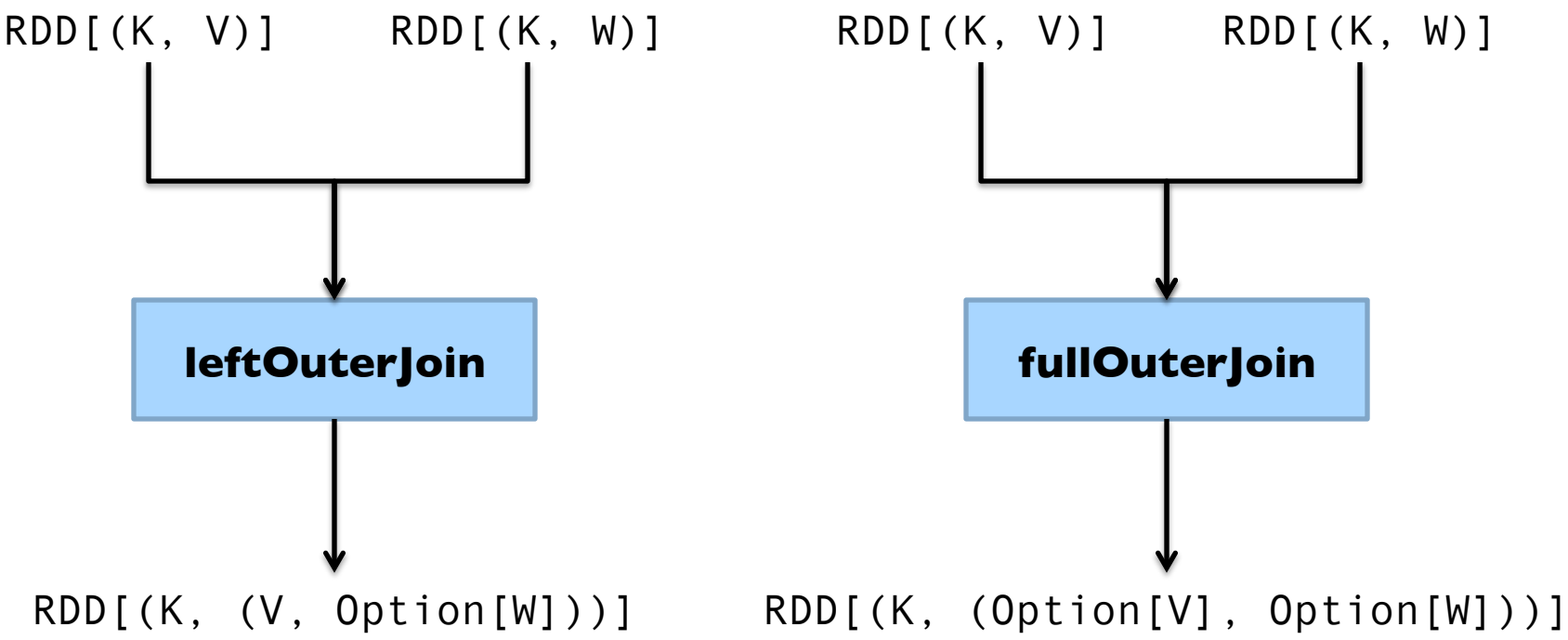
(Not meant to be exhaustive)

Join-like Operations



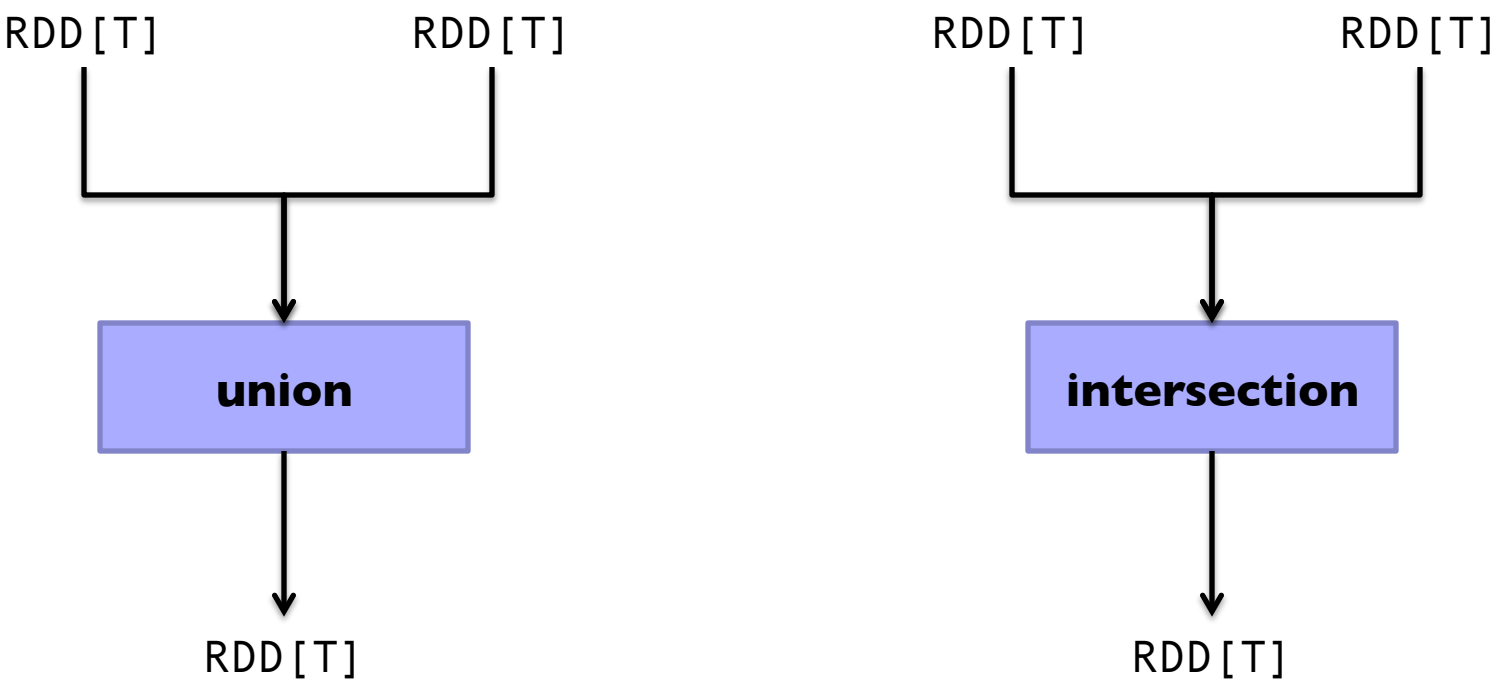
(Not meant to be exhaustive)

Join-like Operations



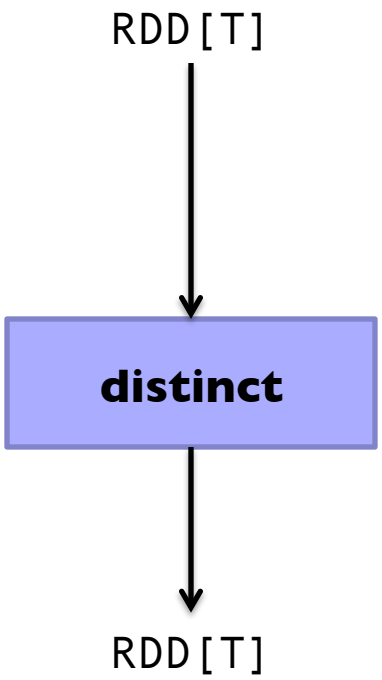
(Not meant to be exhaustive)

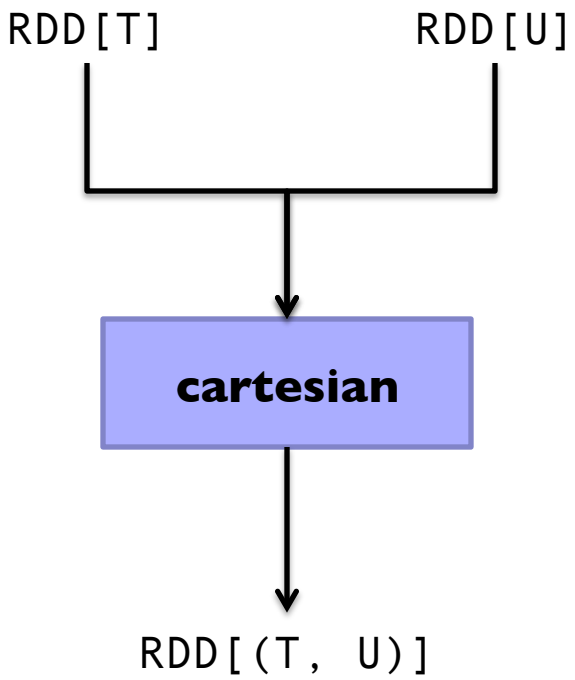
Set-ish Operations



(Not meant to be exhaustive)

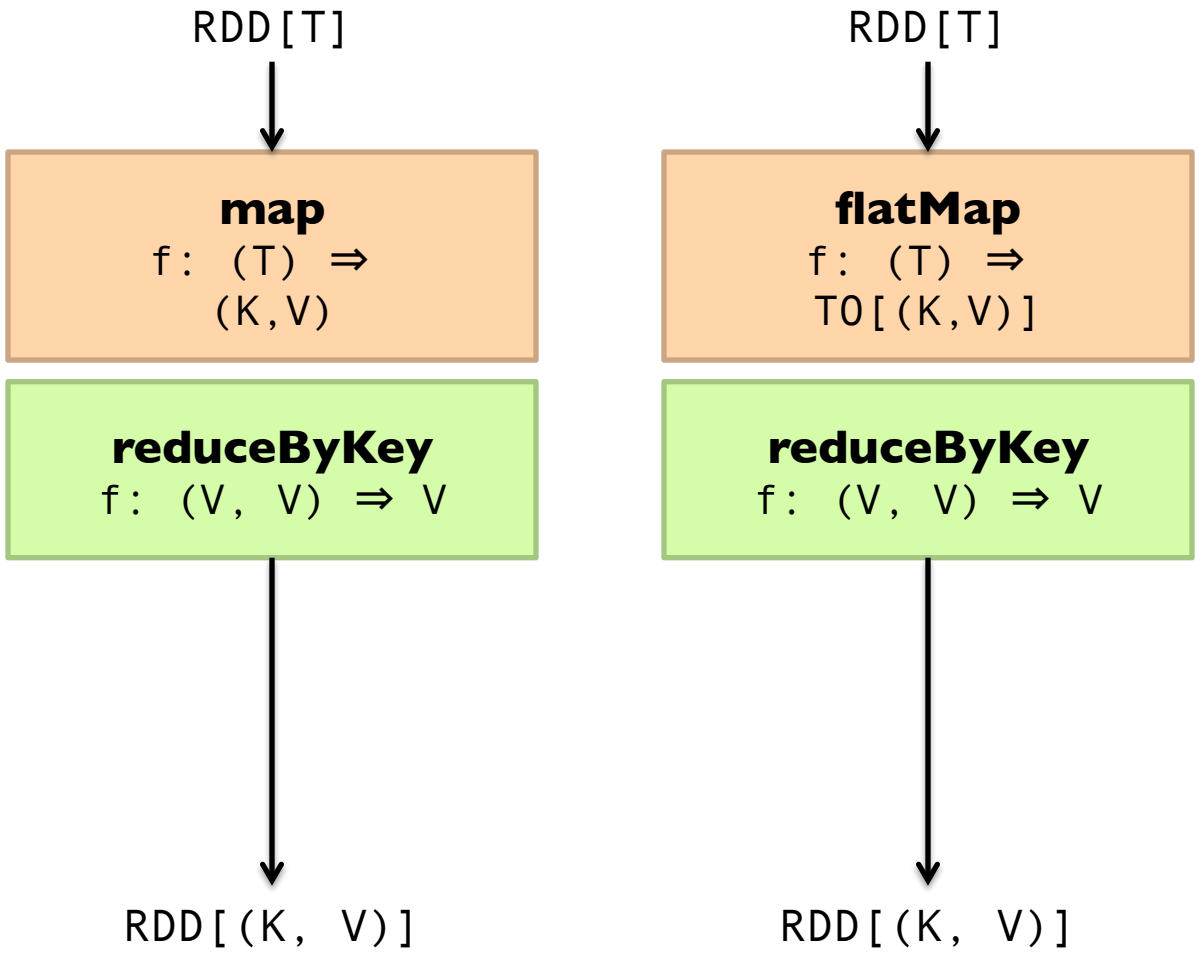
Set-ish Operations





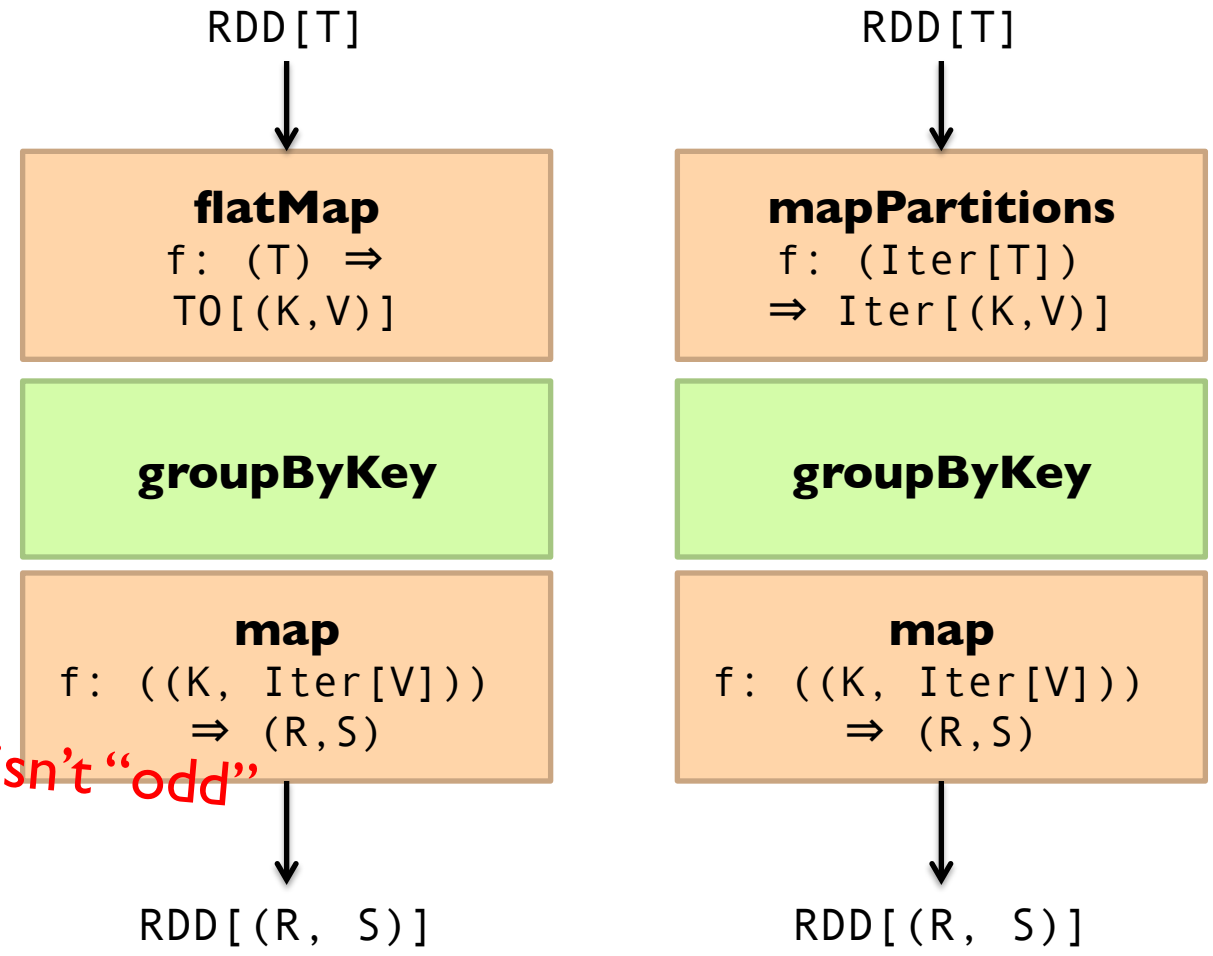
(Not meant to be exhaustive)

MapReduce in Spark?



Not quite…

MapReduce in Spark?



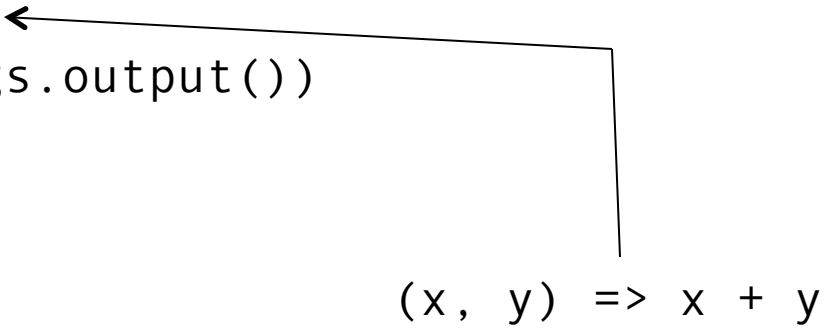
Nope, *this isn't "odd"*

Still not quite…

Spark Word Count

```
val textFile = sc.textFile(args.input())

textFile
  .flatMap(line => tokenize(line))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
  .saveAsTextFile(args.output())
```



Don’t focus on Java verbosity!

```
val textFile = sc.textFile(args.input())

textFile
  .map(object mapper {
    def map(key: Long, value: Text) =
      tokenize(value).foreach(word => write(word, 1))
    })
  .reduce(object reducer {
    def reduce(key: Text, values: Iterable[Int]) = {
      var sum = 0
      for (value <- values) sum += value
      write(key, sum)
    })
  .saveAsTextFile(args.output())
```

Next Time

- What's an RDD?
- How does Spark actually work?
- Algorithm design: redux

Source: Wikipedia (Japanese rock garden)