



Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

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

January 19, 2016

Jimmy Lin

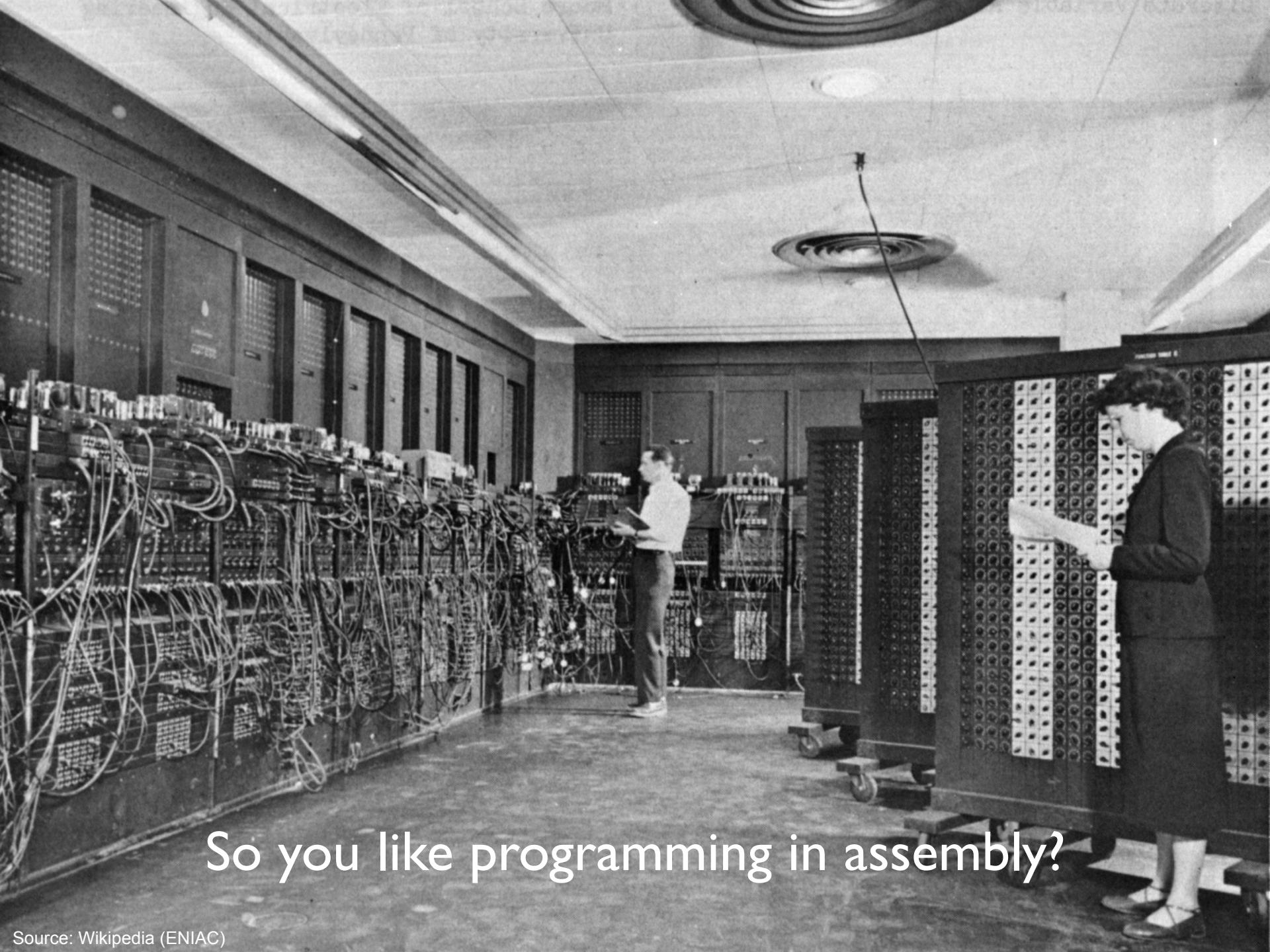
David R. Cheriton School of Computer Science
University of Waterloo

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

An aerial photograph of a large data center complex during sunset. The sky is a warm orange and yellow. In the foreground, there are several large white industrial buildings, some with flat roofs and others with gabled roofs. A parking lot with many cars is visible. To the right, there is a large building with a green roof and a row of white cylindrical storage tanks. In the background, there are more buildings, roads, and fields stretching towards the horizon under a hazy sky.

The datacenter *is* the 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 waaaaay too low level!
(circa 2007)



What we really need is SQL!

Answer:



What we really need is a
scripting language!

Answer:



facebook®

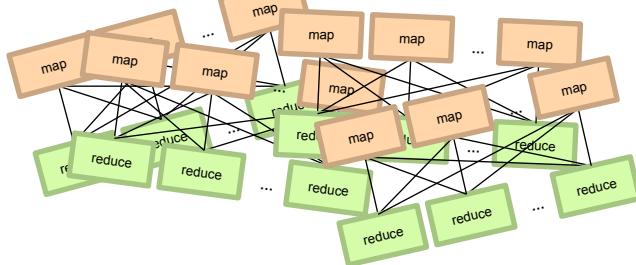


YAHOO!®

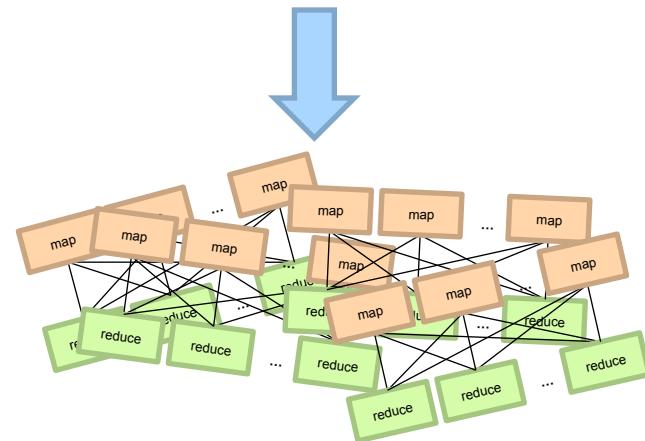


SQL

Aside: Why not just
use a database?

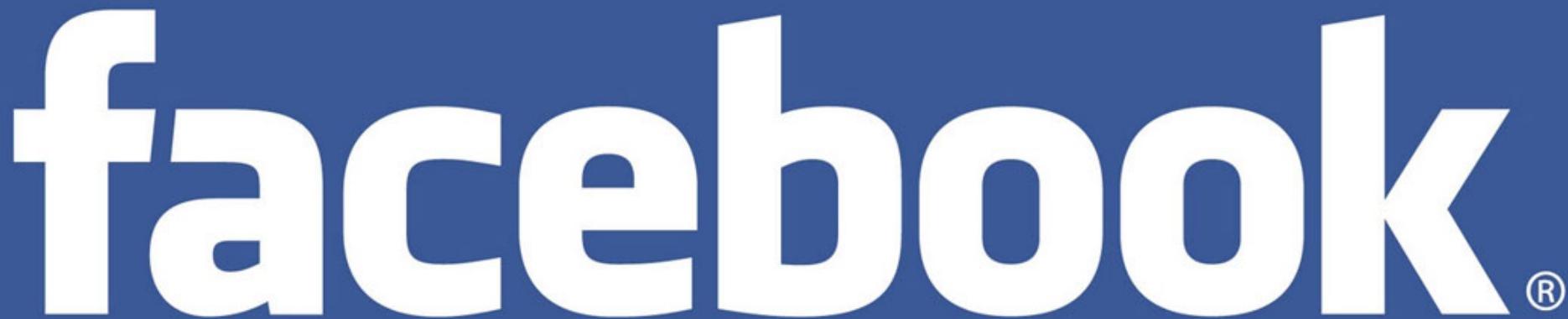


Pig Scripts



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.”

A close-up photograph of a large, light-colored pig, possibly cream or yellowish-orange, lying on a bed of dry straw. The pig is positioned diagonally, facing towards the bottom left of the frame. Its skin is smooth and slightly wrinkled, with some darker pink areas around its ears and legs. The straw is a golden-yellow color and fills the entire background and foreground.

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

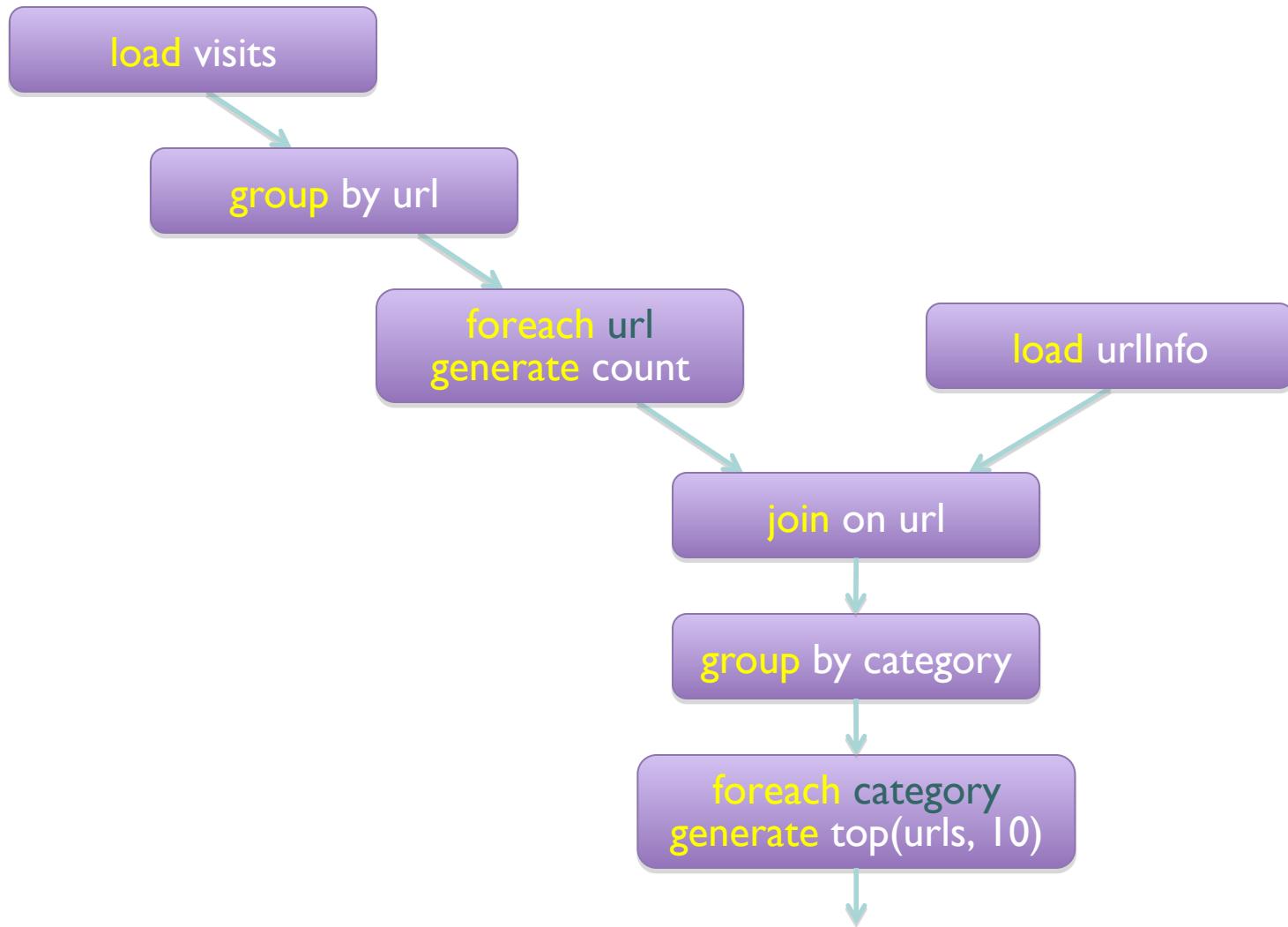
Url	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9



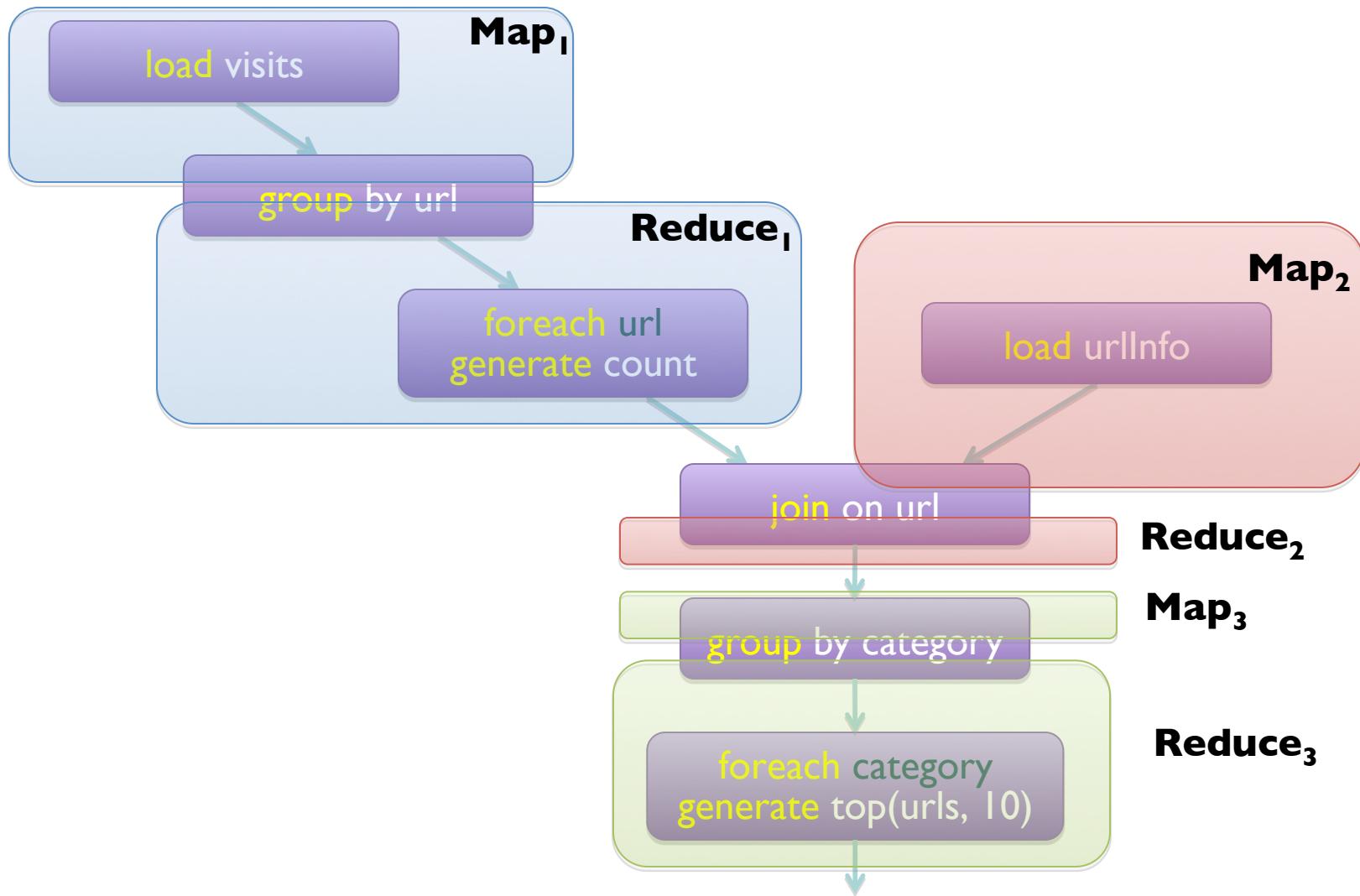
Pig: Example Script

```
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



MapReduce Execution



```

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.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.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.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.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
    // reporter.setStatus("OK");
}

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, LongWritable> {
    public void map(
        Text k,
        Text val,
        OutputColle ctor<Text, LongWritable> oc,
        Reporter reporter) throws IOException {
        // Find the first comma
        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(1L));
    }
}

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().get();
            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
    }
}

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();
join.setJobName("Join Users and Pages");
join.setInputFormat(KeyValueTextInputFormat.class);
join.setOutputKeyClass(Text.class);
join.setOutputValueClass(Text.class);
join.setMapperClass(JoinMapper.class);
join.setCombinerClass(JoinCombiner.class);
join.setReducerClass(JoinReducer.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/join"));
join.setNumReduceTasks(50);
Job joinJob = new Job(join);
joinJob.addDependingJob(loadPages);
joinJob.addDependingJob(loadUsers);
joinJob.setJarByClass(MRExample.class);
joinJob.setMapperClass(JoinMapper.class);
joinJob.setCombinerClass(JoinCombiner.class);
joinJob.setReducerClass(JoinReducer.class);
FileInputFormat.addInputPath(join, new Path("/user/gates/tmp/grouped"));
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.setMapperClass(SequenceFileMapper.class);
top100.setCombinerClass(ReduceUrls.class);
top100.setReducerClass(LimitClicks.class);
FileInputFormat.addInputPath(top100, new Path("/user/gates/tmp/grouped"));
FileOutputFormat.setOutputPath(top100, new Path("/user/gates/tmp/grouped1"));
top100.setNumReduceTasks(1);
Job limitJob = new Job(top100);
limitJob.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)

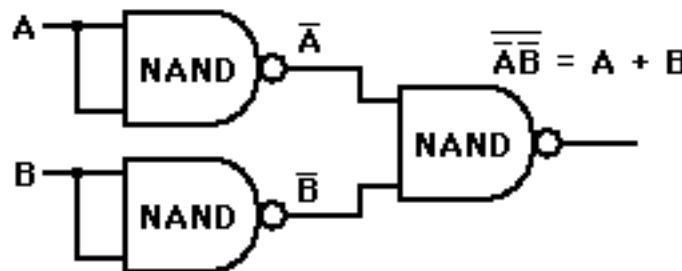
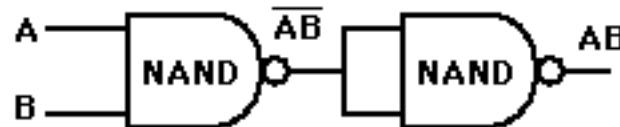
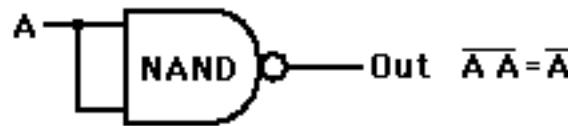
Pig UDFs

- 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

An aerial photograph of a large data center complex during sunset. The sky is a warm orange and yellow. In the foreground, there's a mix of green fields and industrial buildings, including several large white data center pods arranged in rows. A highway runs through the middle ground. The overall scene is a blend of rural and industrial landscapes.

The datacenter *is* the computer!
What's the instruction set?
Okay, let's fix this!

Analogy: NAND Gates are universal



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

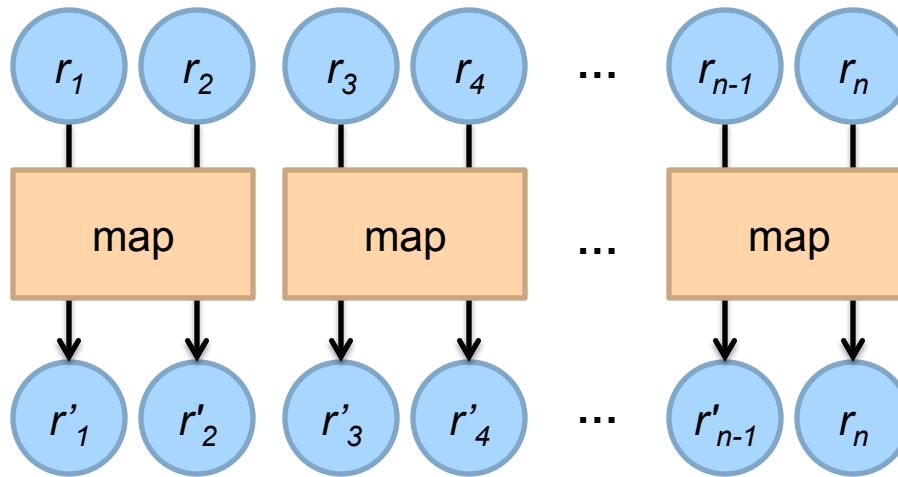
(Why is MapReduce the way it is?)

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

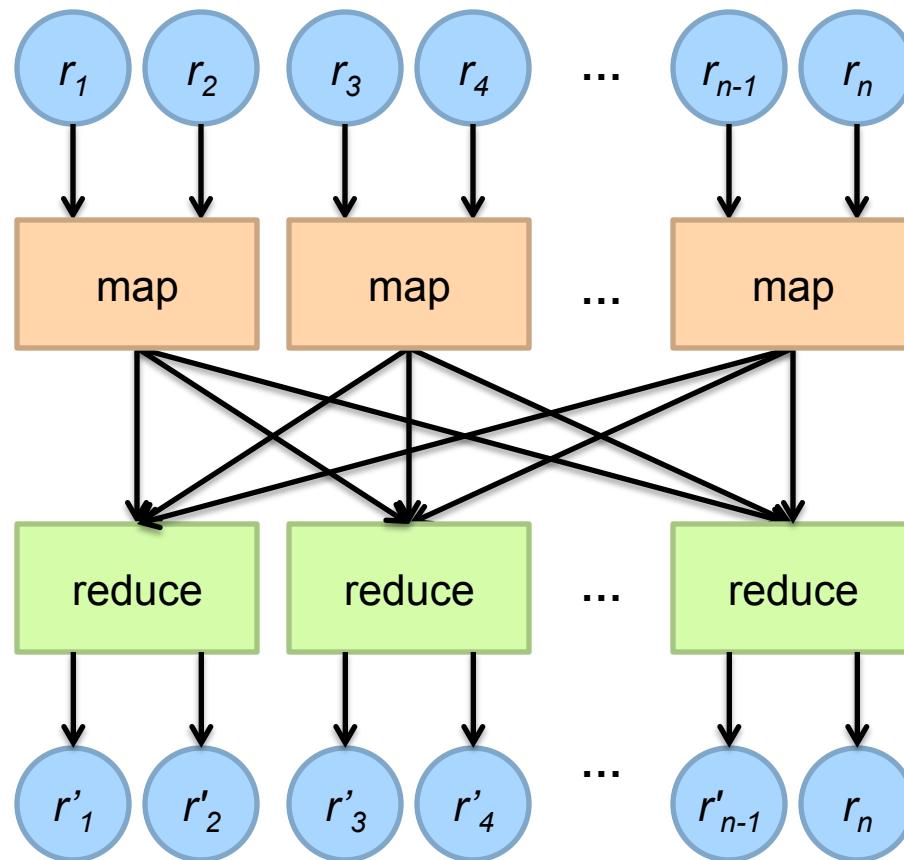
(If we want more than embarrassingly parallel processing)

We need a way to group partial results
Intermediate (key, value) pairs

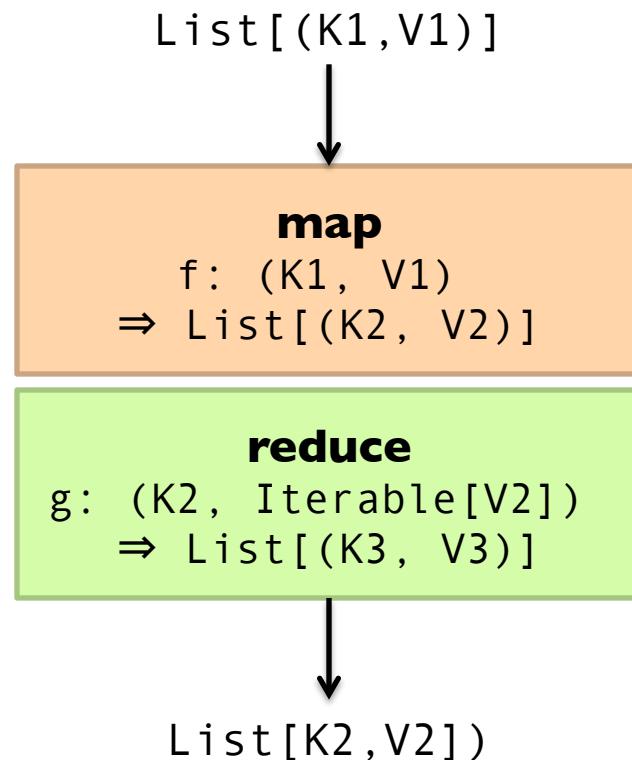


For each key, we can apply some computation

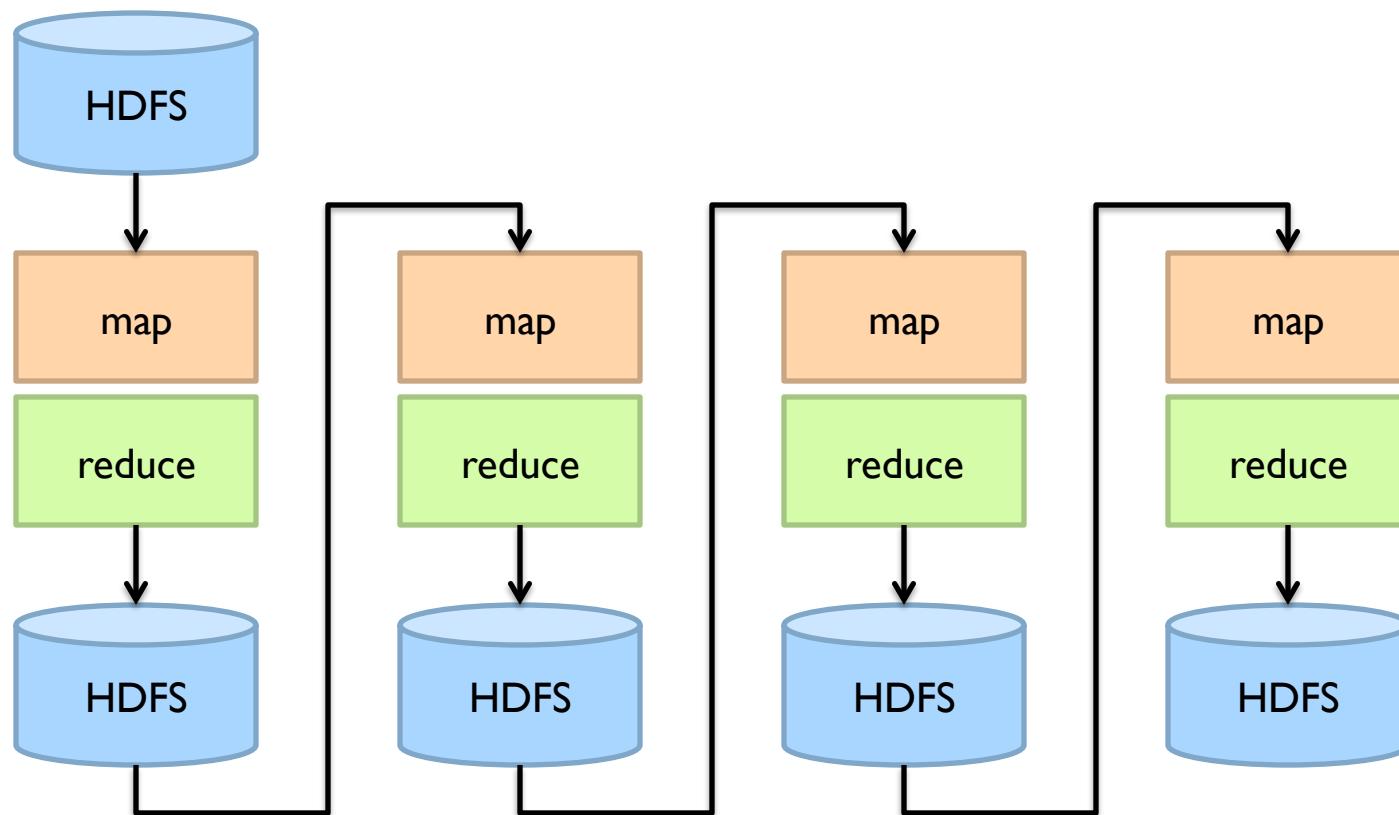
MapReduce



MapReduce

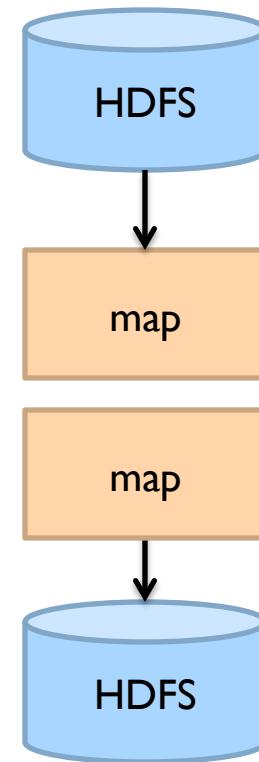
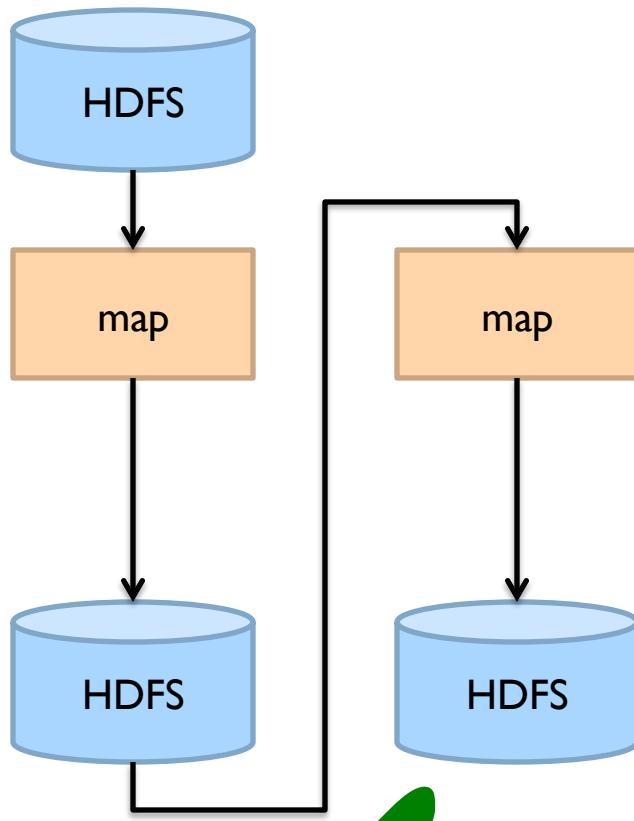


MapReduce Workflows

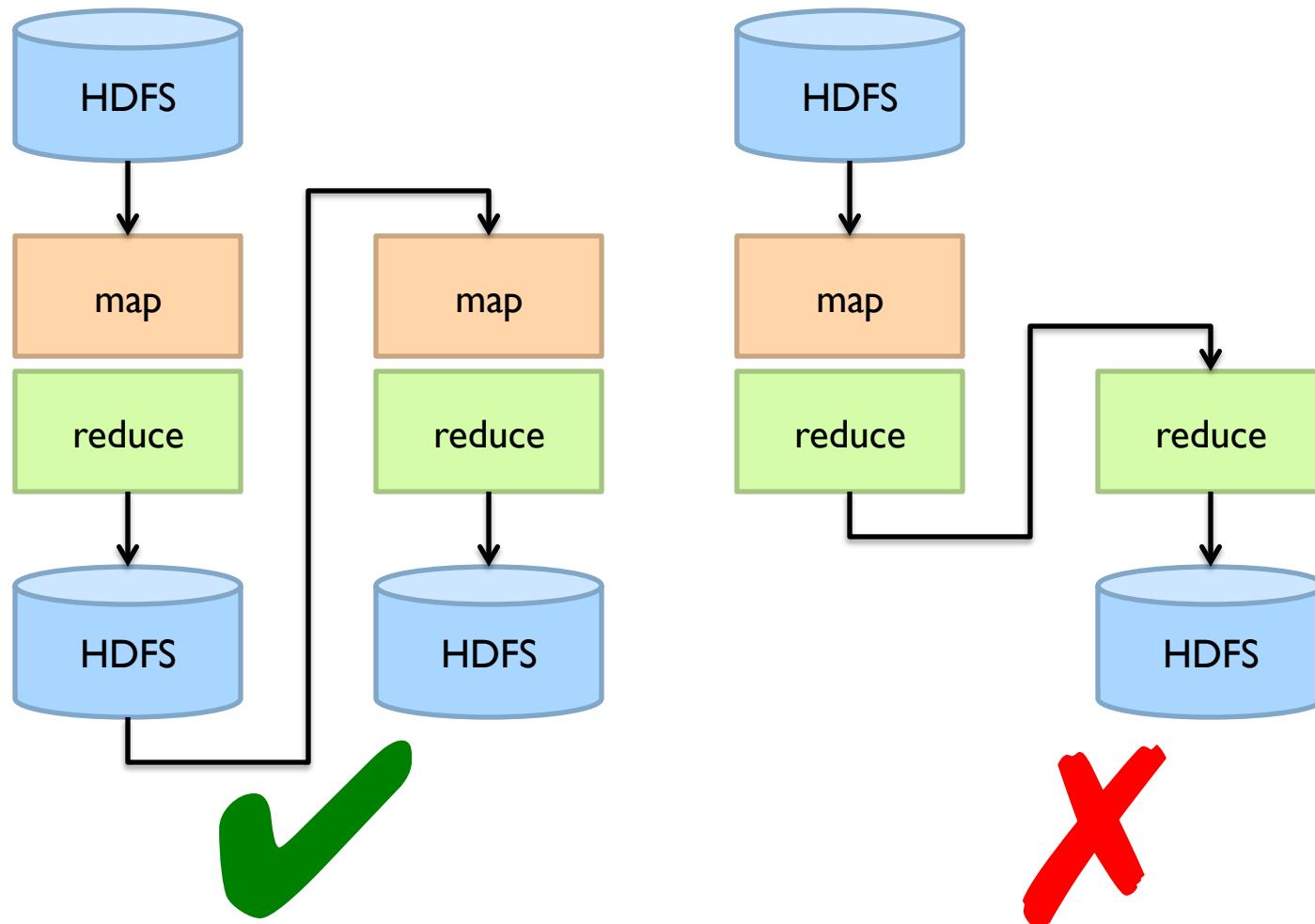


What's wrong?

Want MM...?



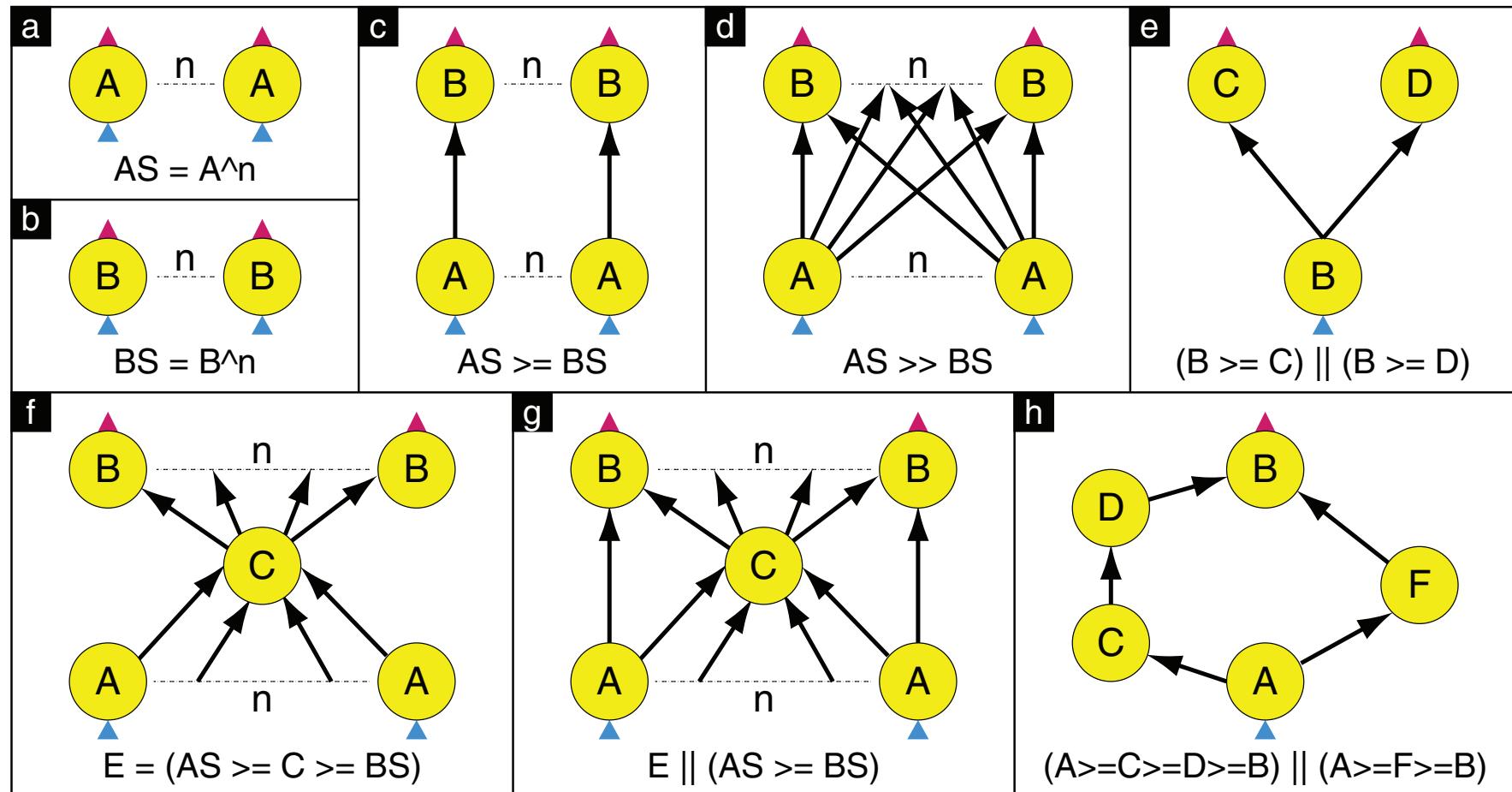
Want MRR?



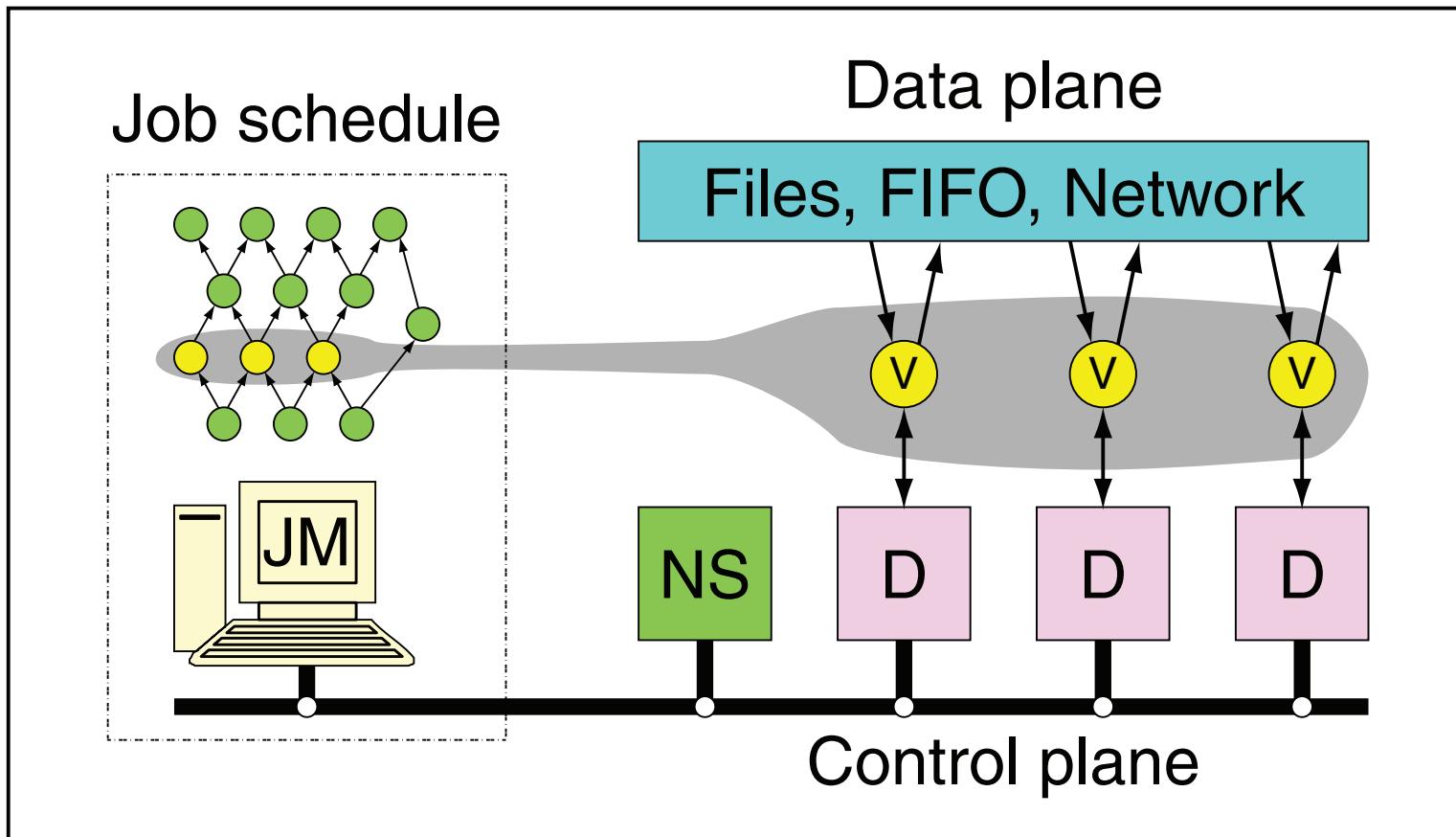
An aerial photograph of a large data center complex during sunset. The sky is a warm orange and yellow. In the foreground, there are several large white industrial buildings, some with flat roofs and others with gabled roofs. A parking lot with many cars is visible in front of one of the buildings. To the right, there is a large building with a green roof and a parking lot. In the background, there are more buildings and a highway. The surrounding area is a mix of green fields and brown dirt roads.

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

Dryad: Graph Operators



Dryad: Architecture

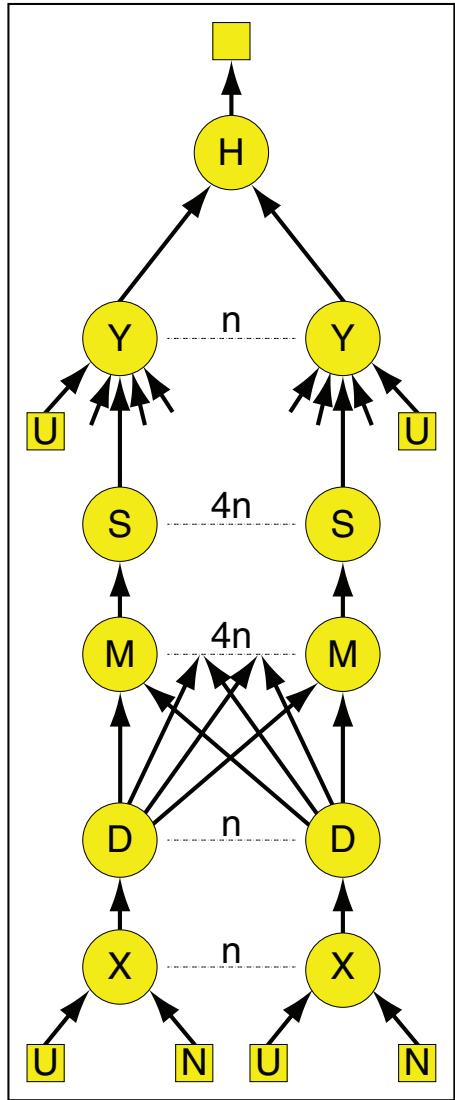


The Dryad system organization. The job manager (JM) consults the name server (NS) to discover the list of available computers. It maintains the job graph and schedules running vertices (V) as computers become available using the daemon (D) as a proxy. Vertices exchange data through files, TCP pipes,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

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

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;
```

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?

DryadLINQ: Word Count

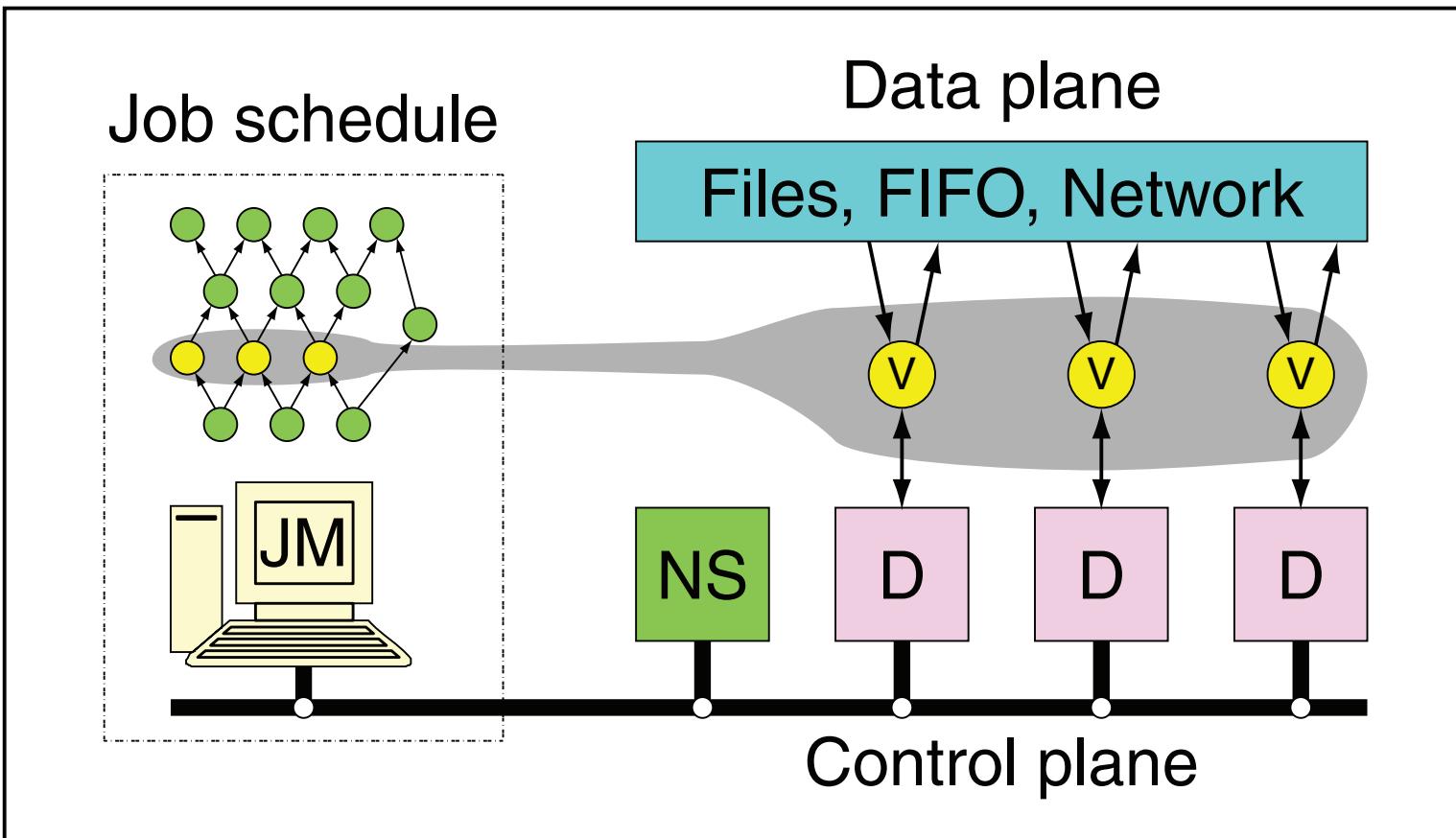
```
PartitionedTable<LineRecord> inputTable =  
    PartitionedTable.Get<LineRecord>(uri);  
  
IQueryable<string> words = inputTable.SelectMany(x => x.line.Split(' '));  
IQueryable<IGrouping<string, string>> groups = words.GroupBy(x => x);  
IQueryable<Pair> counts = groups.Select(x => new Pair(x.Key, 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.

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?

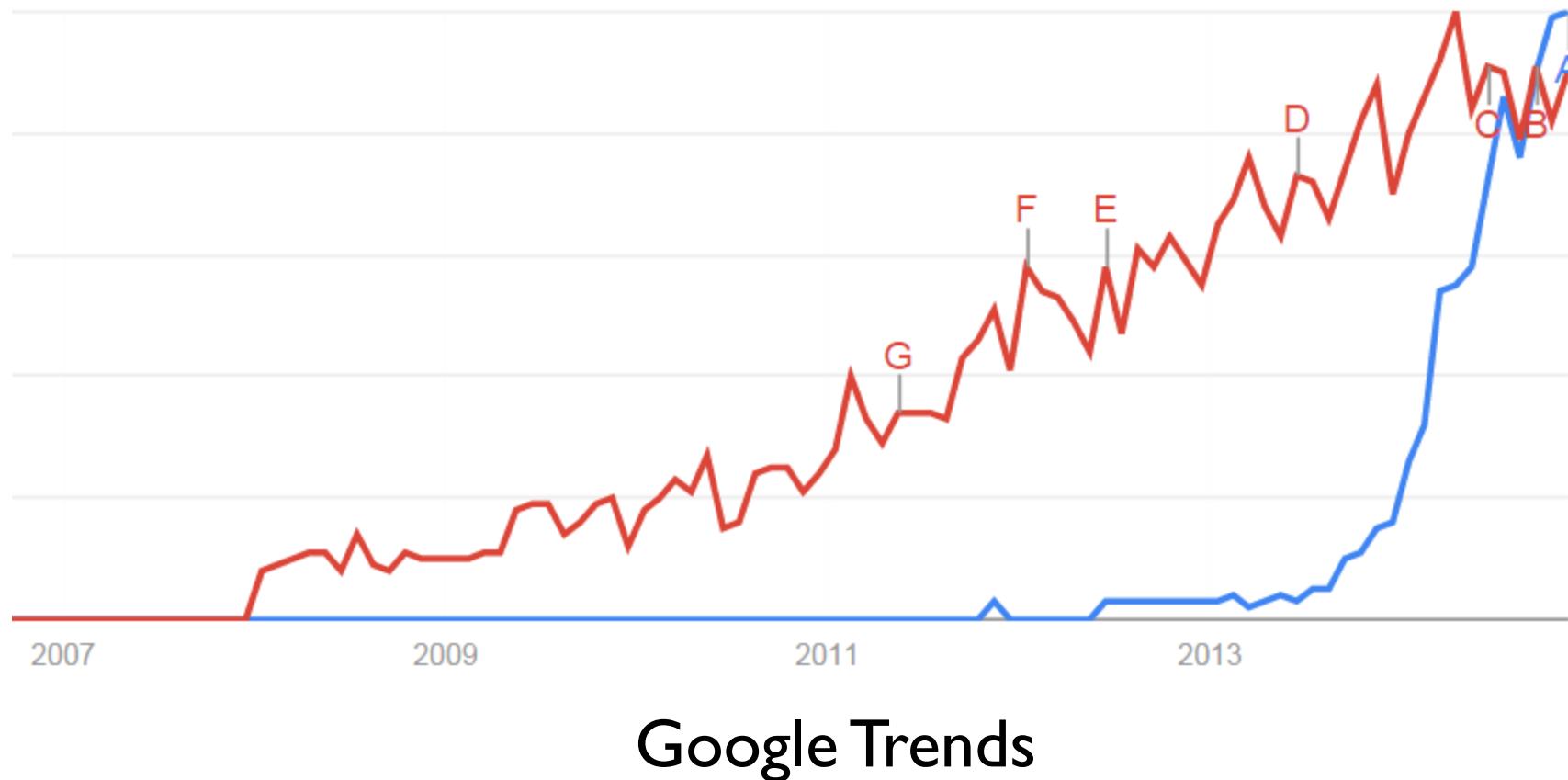
MapReduce?
Spark?

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 Just Passed Hadoop in Popularity on the Web!

November 2014

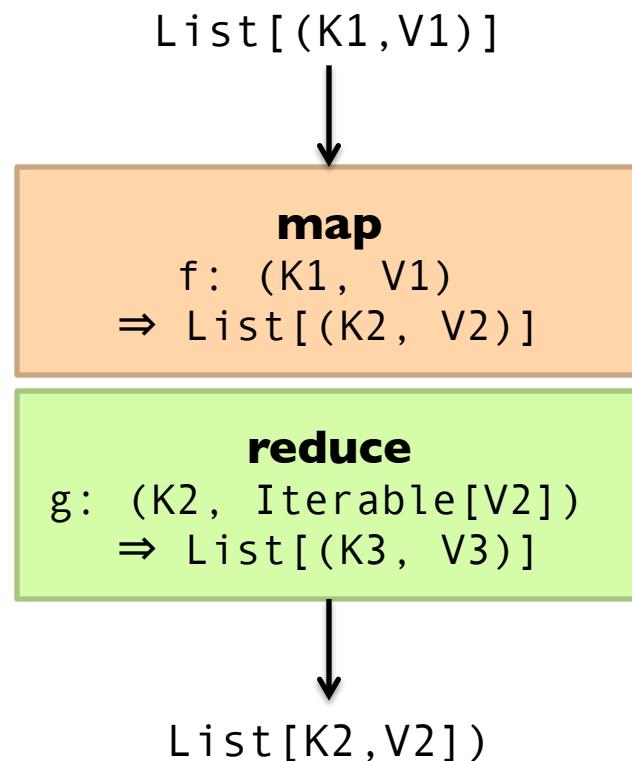


What's an RDD?

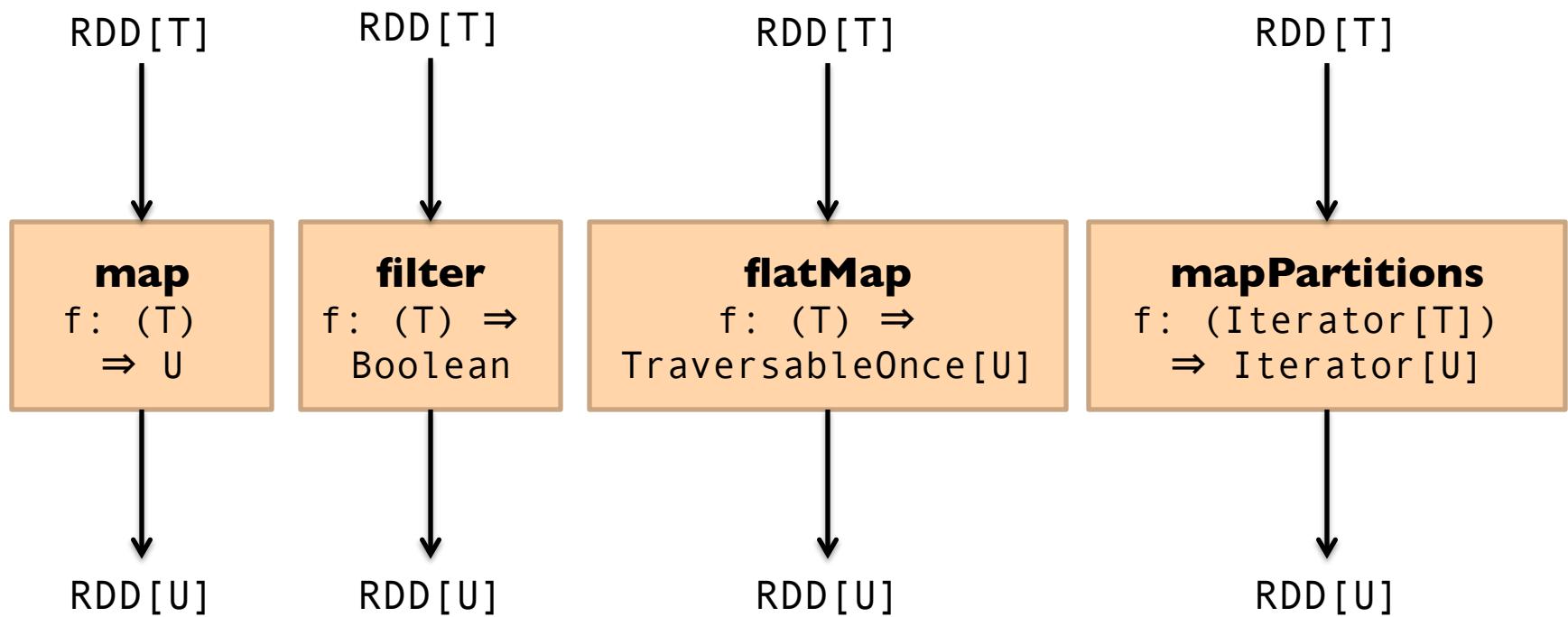
Resilient Distributed Dataset (RDD)

Much more next session...

MapReduce

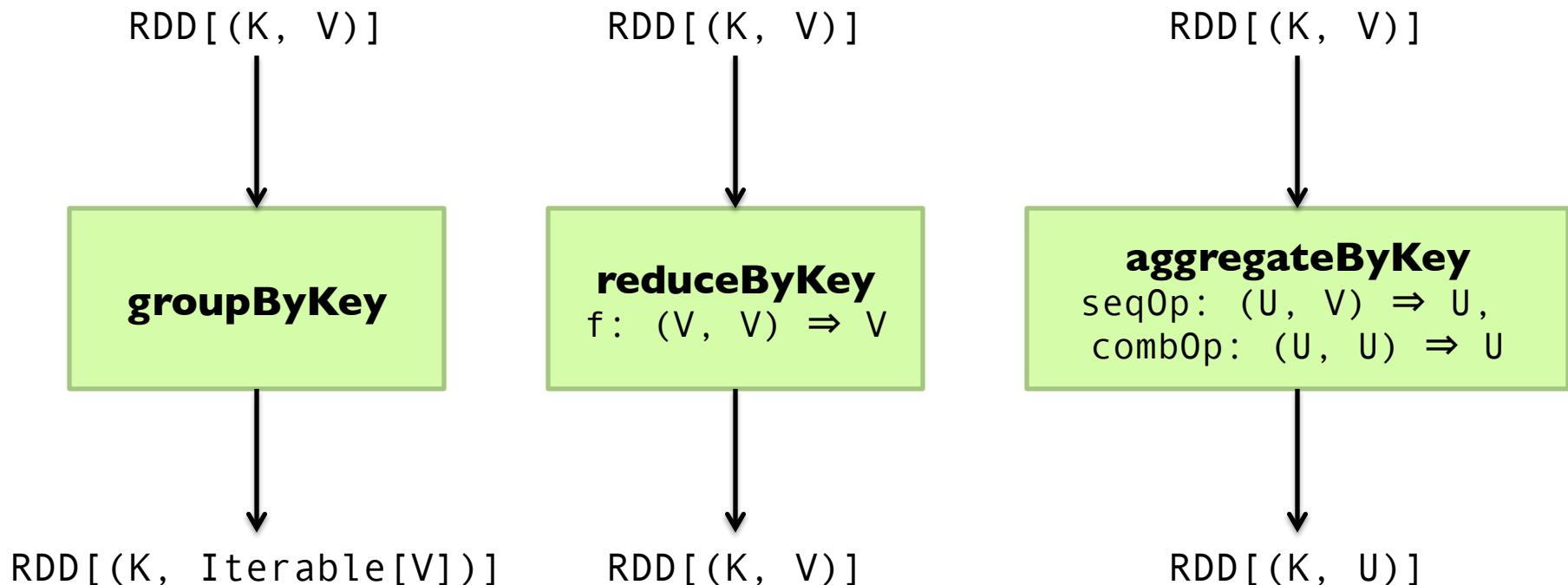


Map-like Operations



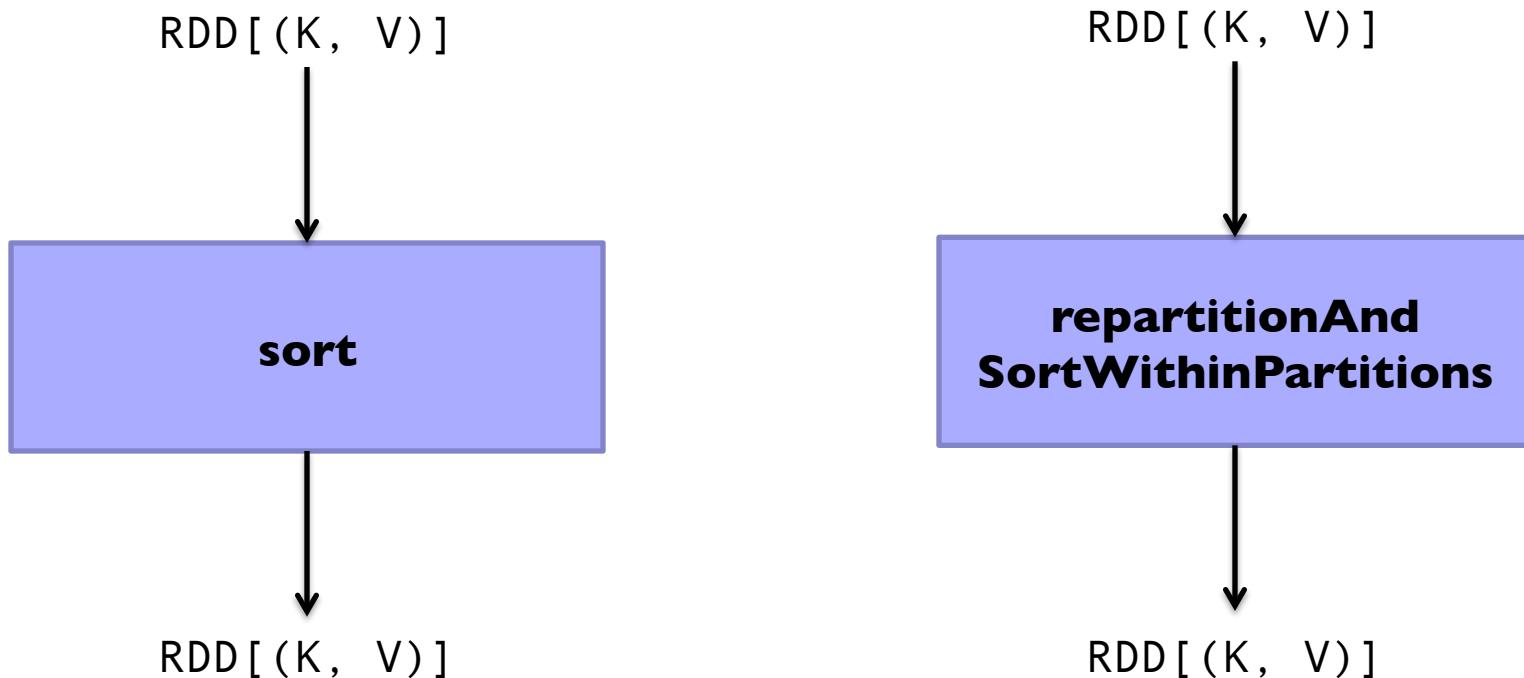
(Not meant to be exhaustive)

Reduce-like Operations



(Not meant to be exhaustive)

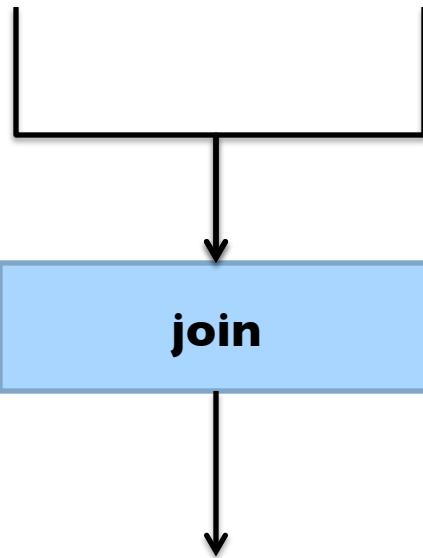
Sort Operations



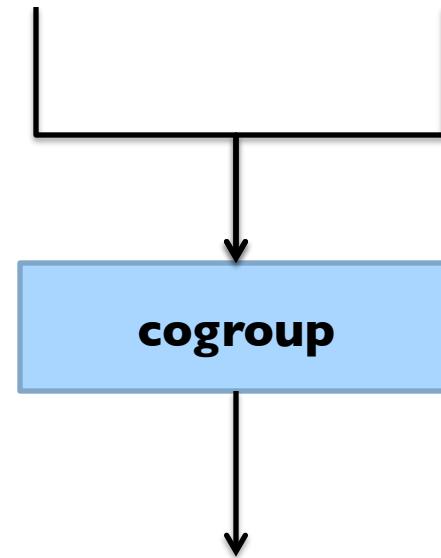
(Not meant to be exhaustive)

Join-like Operations

RDD [(K, V)] RDD [(K, W)]

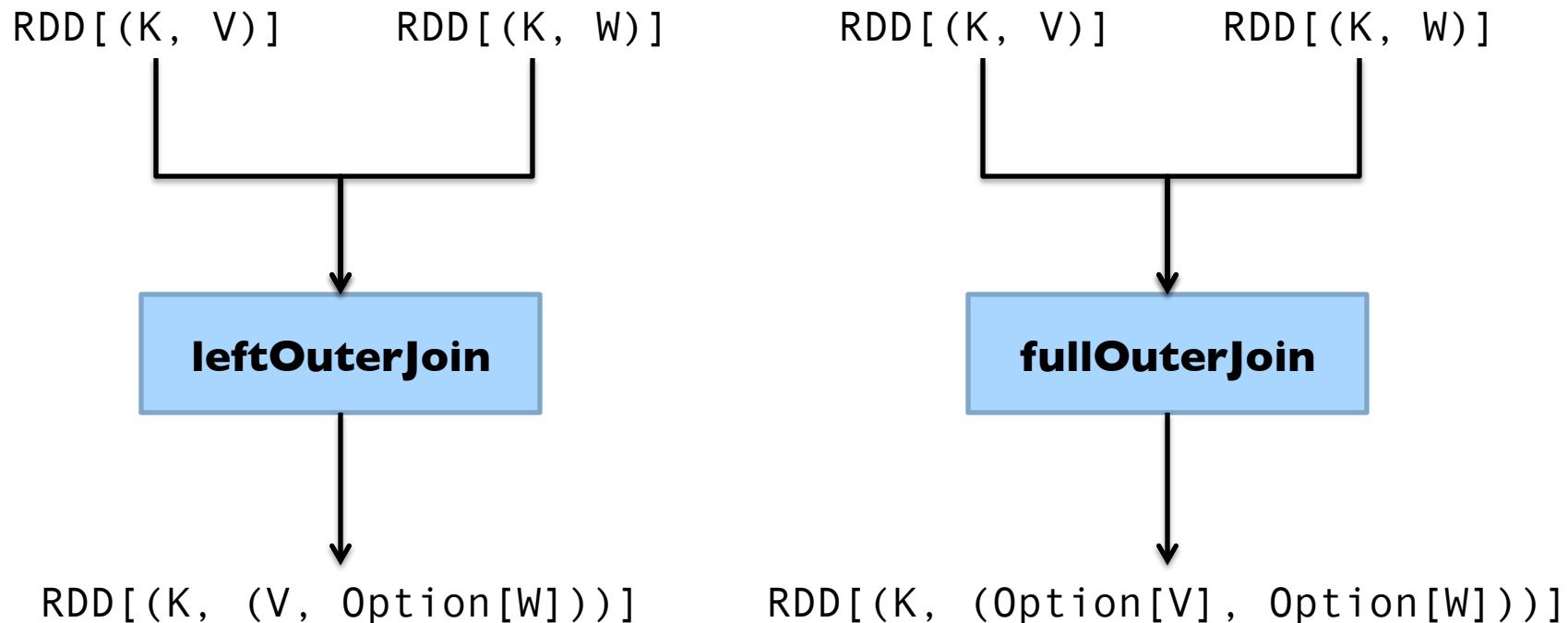


RDD [(K, V)] RDD [(K, W)]



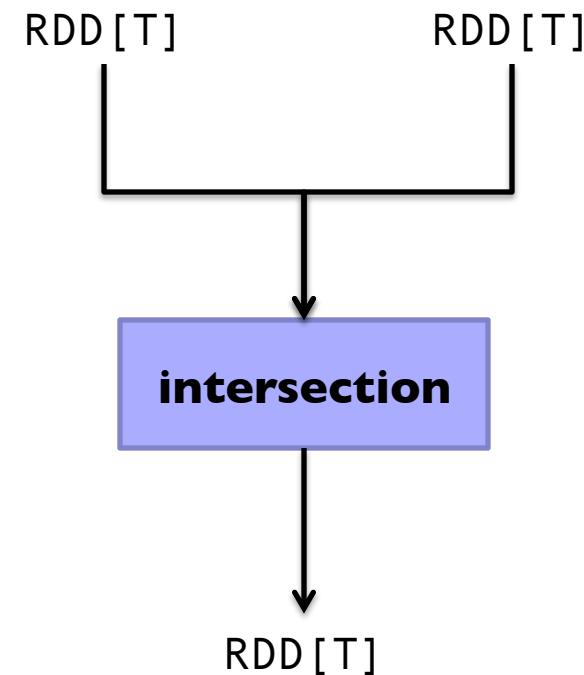
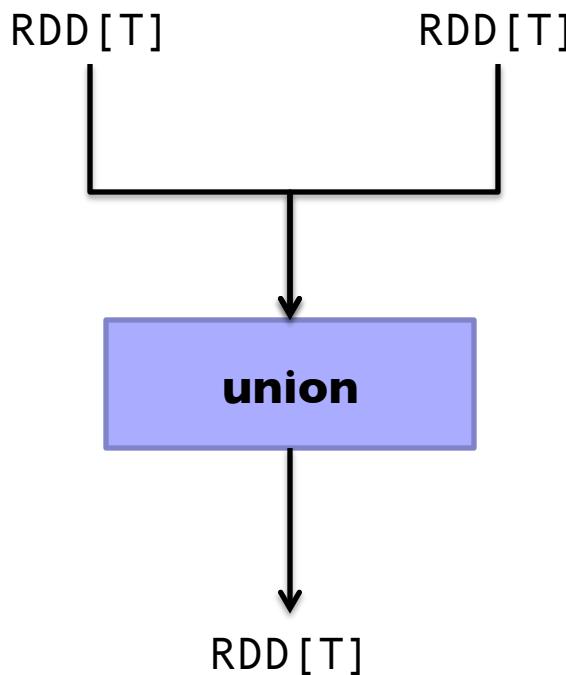
(Not meant to be exhaustive)

Join-like Operations



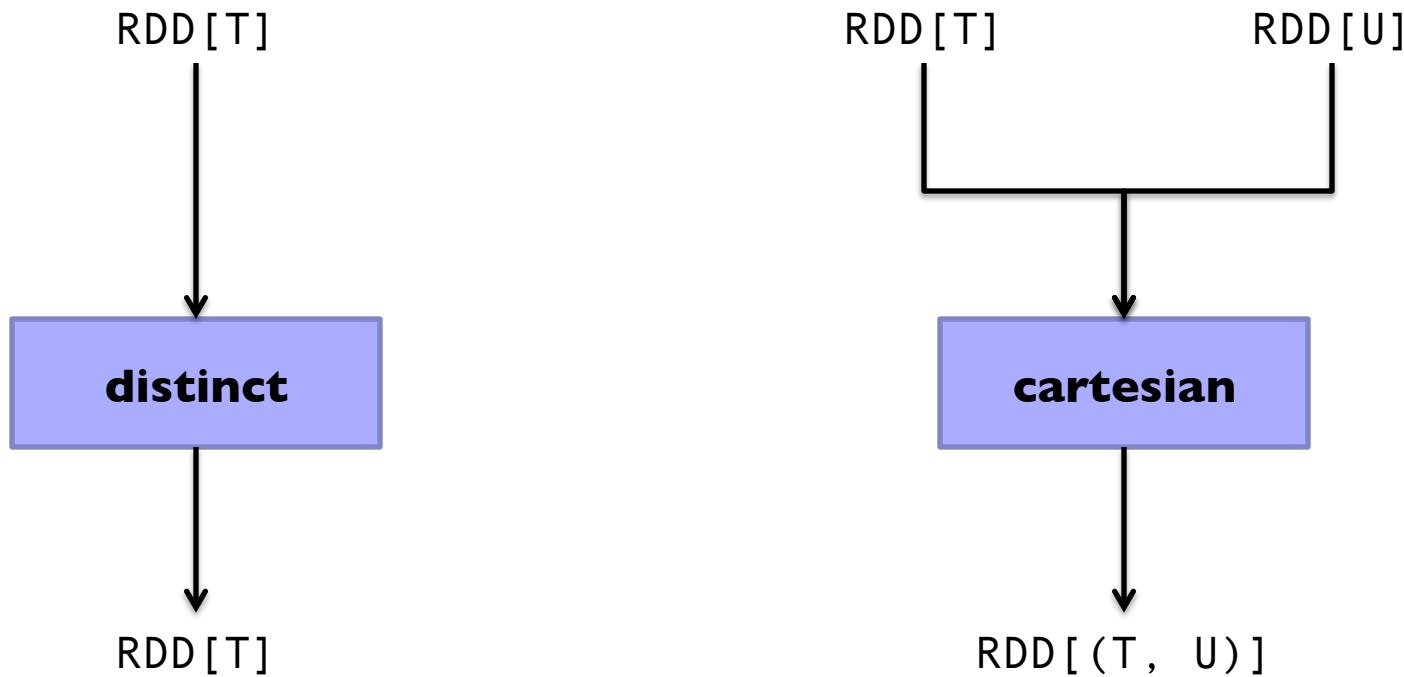
(Not meant to be exhaustive)

Set-ish Operations



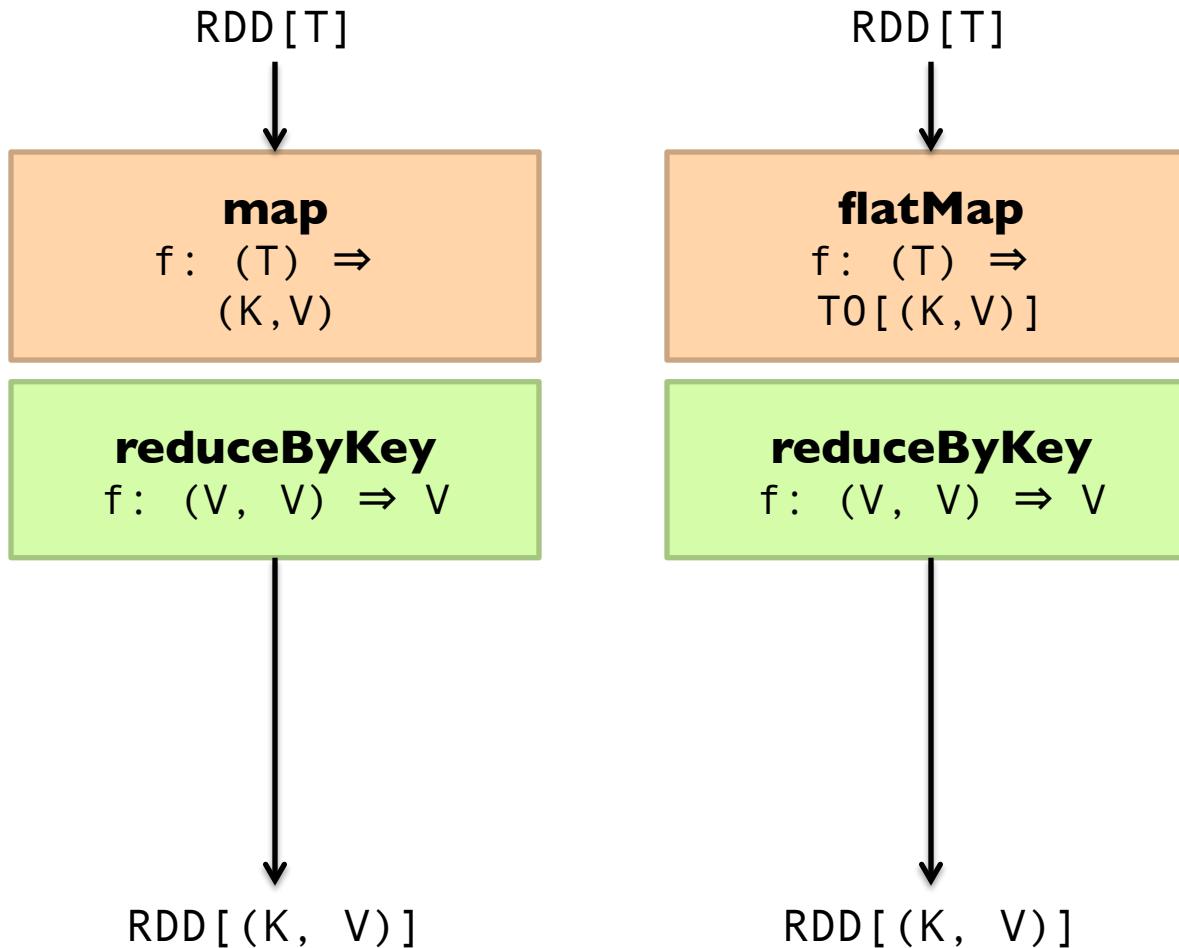
(Not meant to be exhaustive)

Set-ish Operations



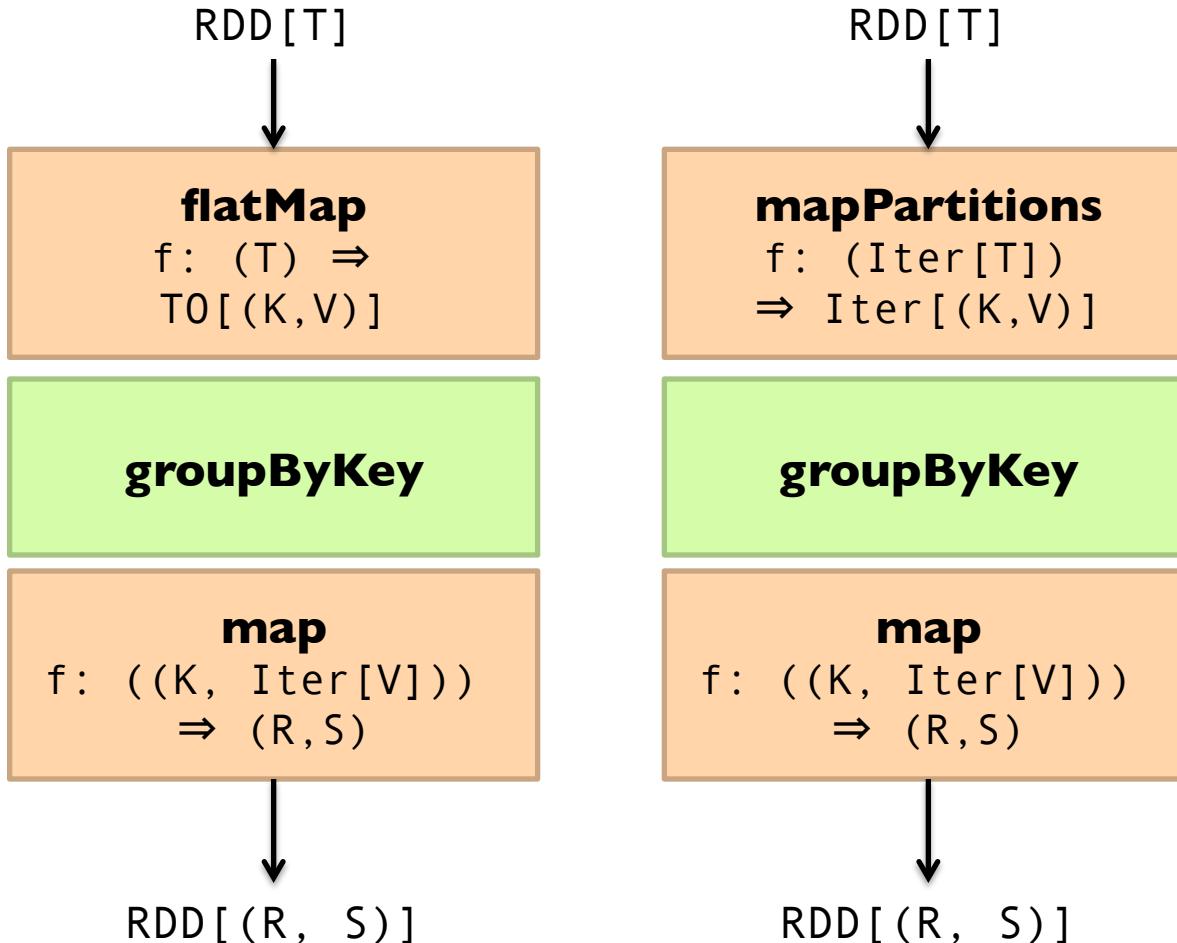
(Not meant to be exhaustive)

MapReduce in Spark?



Not quite...

MapReduce in Spark?



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())
```

$$(x, y) \Rightarrow x + y$$

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

A photograph of a traditional Japanese rock garden. In the foreground, a gravel path is raked into fine, parallel lines. Several large, dark, irregular stones are scattered across the garden. A small, shallow pond is visible in the middle ground, surrounded by more stones and low-lying green plants. In the background, there are more trees and shrubs, and the wooden buildings of a residence are visible behind the garden wall.

Questions?