Data-Intensive Distributed Computing

CS 451/651 (Fall 2025)

UNIVERSITY OF WATERLOO

Batch Processing II

(v1.00)

Week 4: September 23

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

These slides are available at `https://lintool.github.io/cs451-2025f/`

Key Questions

In what ways does Spark improve over MapReduce?

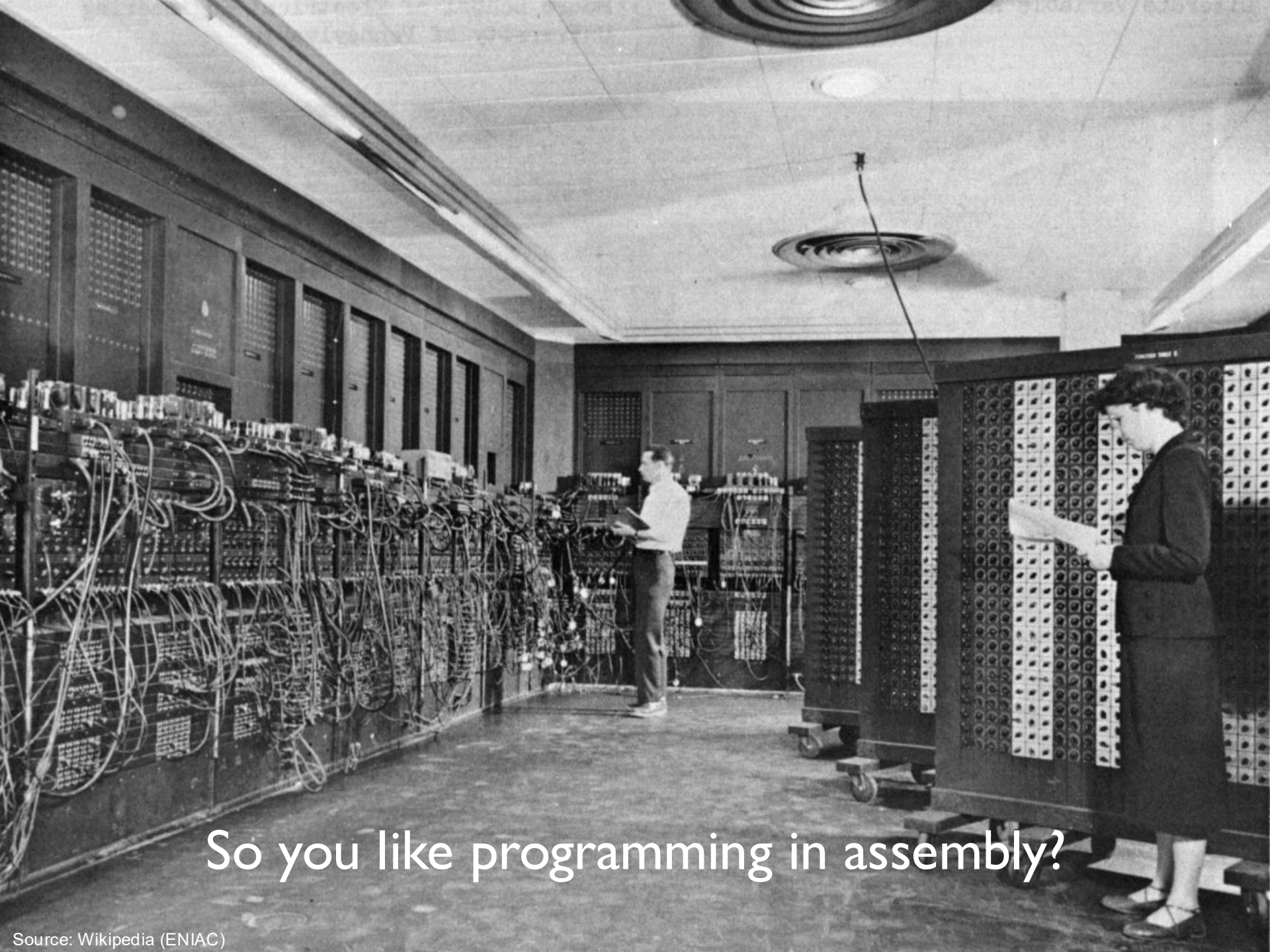
How is distributed group by implemented efficiently at scale?

How do commutative and associative operations
contribute to efficient distributed execution?

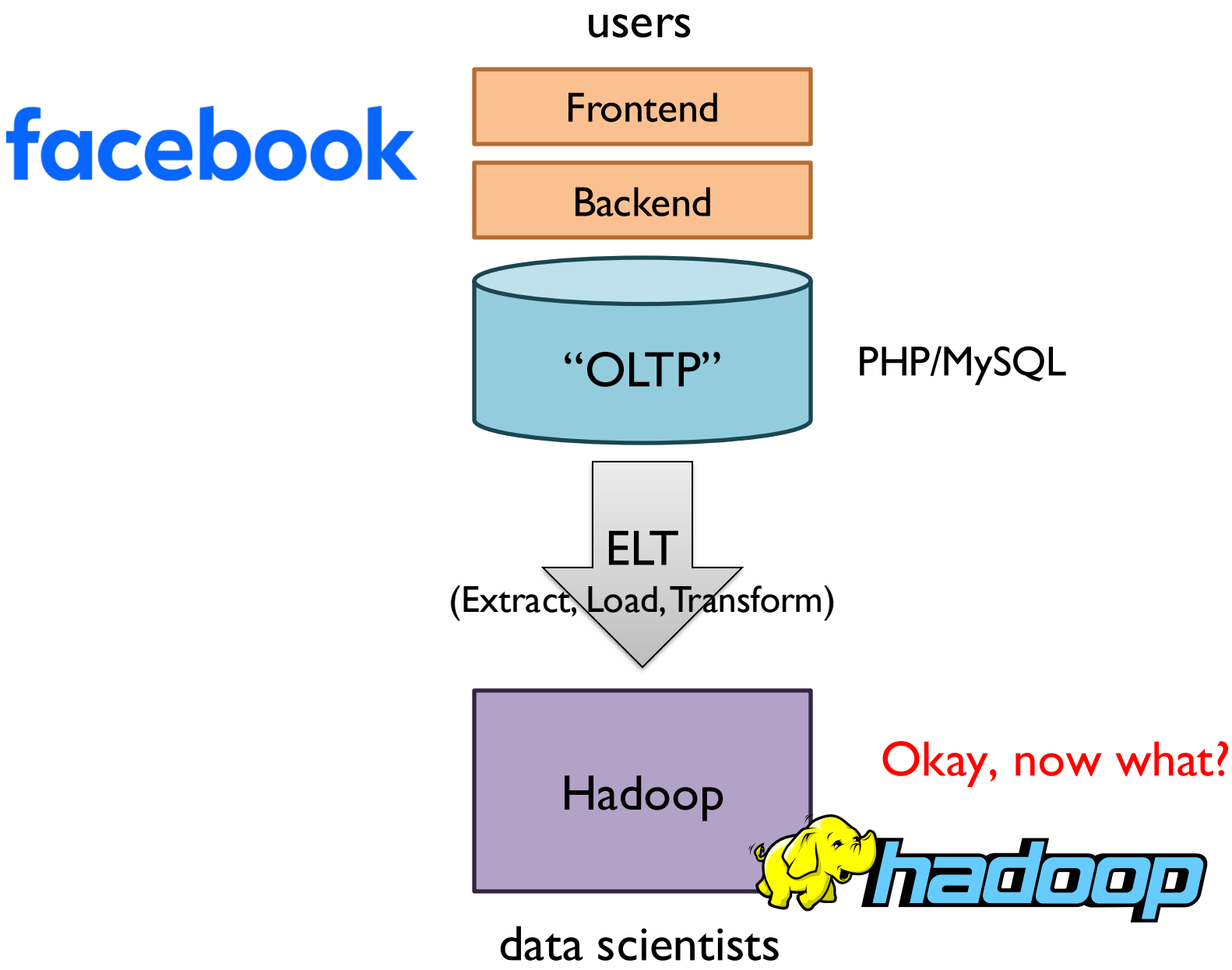
How does partitioning contribute to efficient distributed execution?

How do these concepts come together in efficient joins at scale?


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



Source: Wikipedia (ENIAC)


users
facebook
Frontend
Backend
"OLTP"
PHP/MySQL
ELT
(Extract, Load, Transform)
Hadoop
Okay, now what?
hadoop
data scientists

What's the problem? (circa 2007)

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

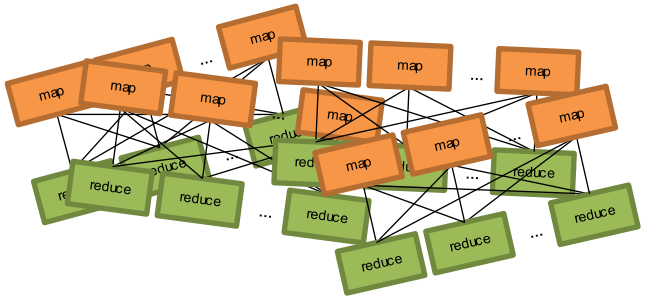
What's the solution?

Abstraction to the rescue – build a higher-level language!

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

facebook YAHOO!®

With Hadoop, all we got is a bunch of these…



What we really need
is SQL!

What we really need
is a scripting language!

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

What we really need
is SQL!

What we really need
is a scripting language!

Answer:

Answer:

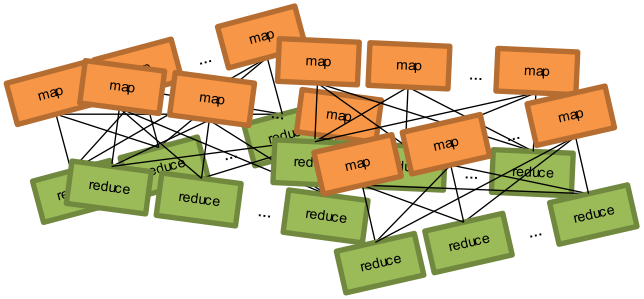
Hadoop is great, but it's really waaaaay too low level!
(circa 2007)

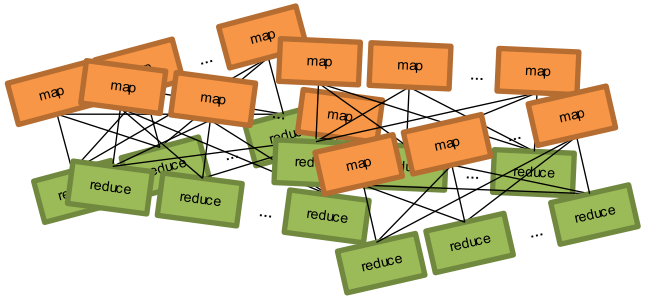
facebook

YAHOO!

SQL

Pig Scripts





Both (still) open-source projects today!

Reminder…

Why not just use an existing analytical database?

Scalability. Cost. Flexibility.

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."

Hive: Example

Relational join on two tables:

Table of word counts from Shakespeare collection

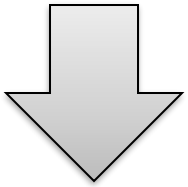
Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word)
  WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;

the      25848    62394
I        23031    8854
and      19671    38985
to       18038    13526
of       16700    34654
a        14170    8057
you      12702    2720
my       11297    4135
in       10797    12445
is       8882     6884
```

Source: Material drawn from Cloudera training VM

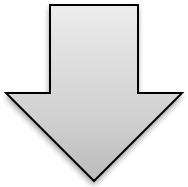
Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word)
  WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```



(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (.
(TOK_TABLE_OR_COL s) word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION
(TOK_DIR TOK_TMP_FILE)) (TOK_SELECT (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (.
(TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq) 1))) (TOK_ORDERBY
(TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```



(one or more of MapReduce jobs)

Hive: Behind the Scenes

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                    expr: _col1
                    type: string
                    expr: _col0
                    type: int
                    expr: _col2
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

Stage: Stage-2
   Map Reduce
     Alias -> Map Operator Tree:
       hdfs://localhost:8022/tmp/hive-training/364214370/10002
           Reduce Output Operator
             key expressions:
                   expr: _col1
                   type: int
             sort order: -
             tag: -1
             value expressions:
                   expr: _col0
                   type: string
                   expr: _col1
                   type: int
                   expr: _col2
                   type: int
     Reduce Operator Tree:
       Extract
         Limit
           File Output Operator
             compressed: false
             GlobalTableId: 0
             table:
                 input format: org.apache.hadoop.mapred.TextInputFormat
                 output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

Stage: Stage-0
   Fetch Operator
     limit: 10
```

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

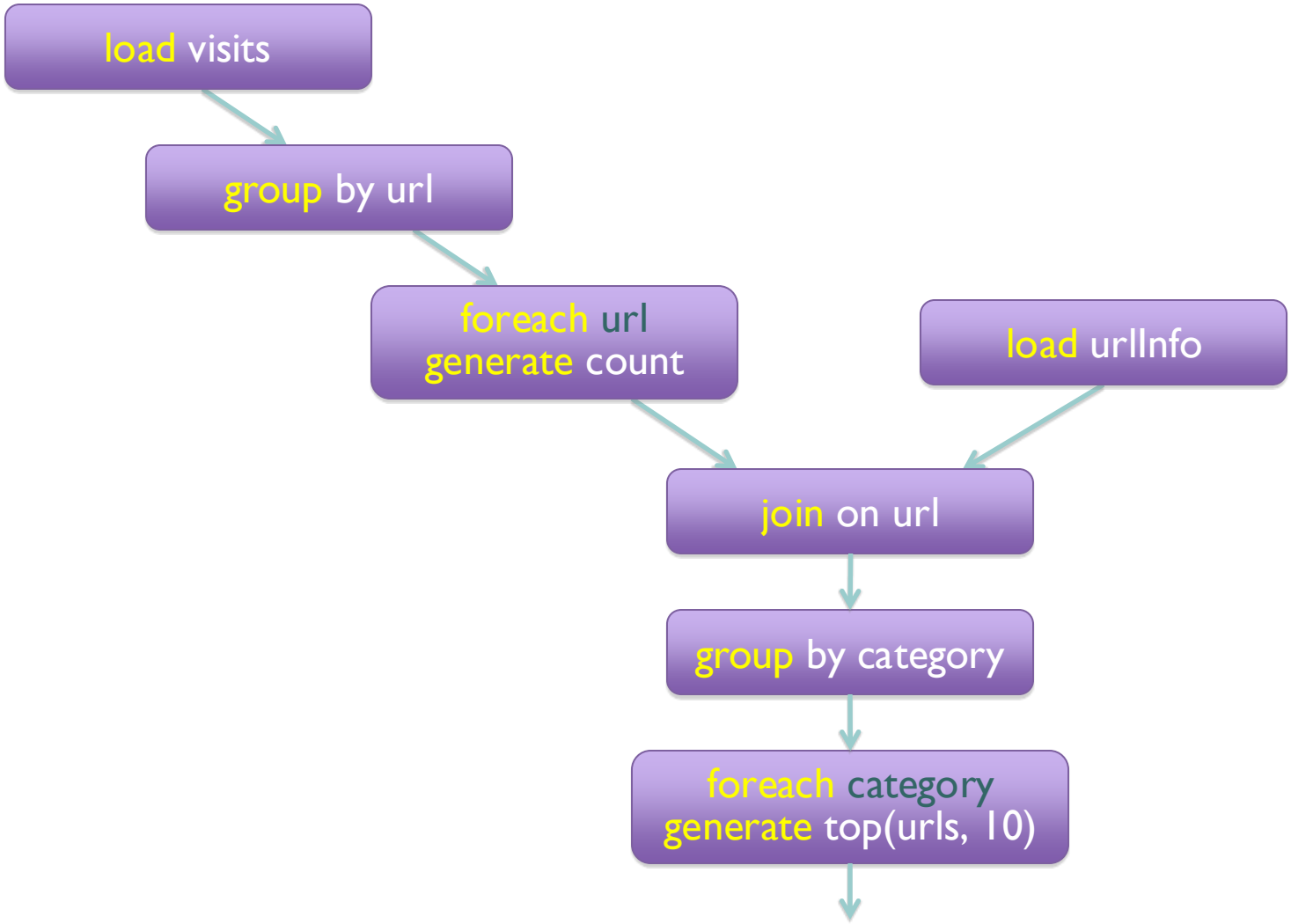
Pig Slides adapted from Olston et al. (SIGMOD 2008)

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';
```

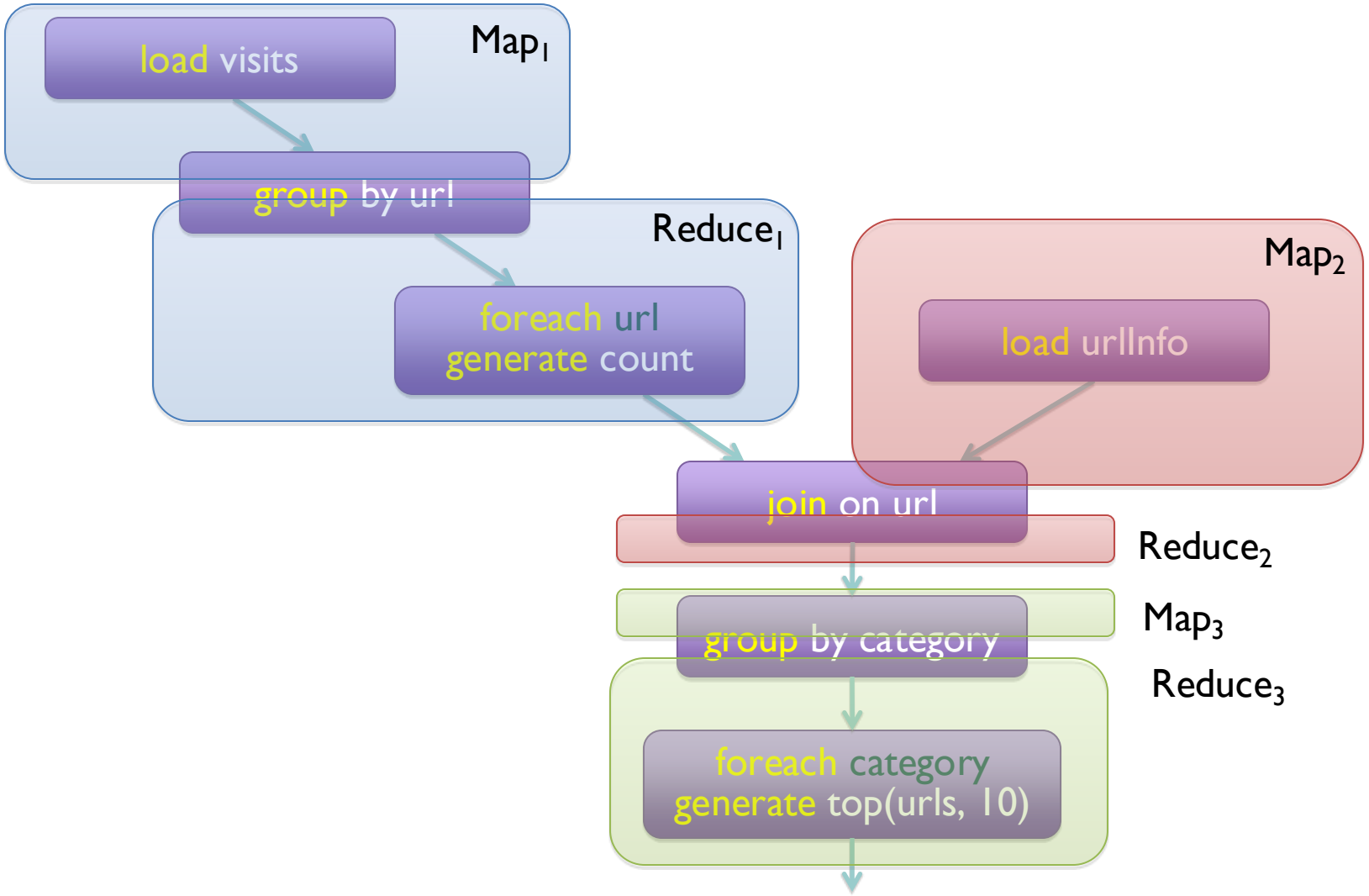
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Pig Query Plan



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

Pig: 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("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 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 Hive / Pig slower?

Why don't you program in assembly all the time?


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

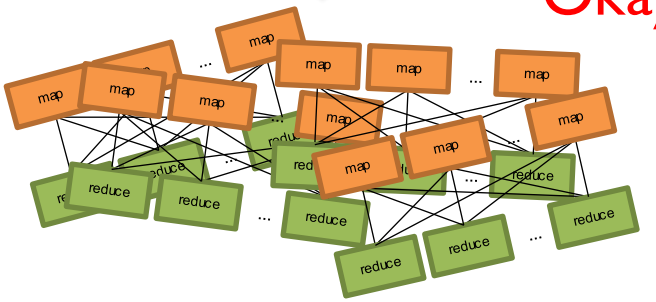
Hadoop is great, but it's really waaaaay too low level!
(circa 2007)

facebook

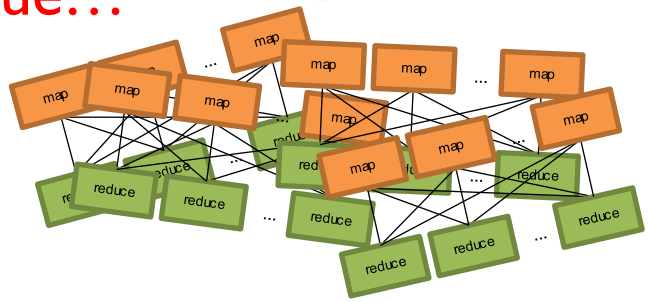
YAHOO!

SQL

Pig Scripts



Okay, next issue…

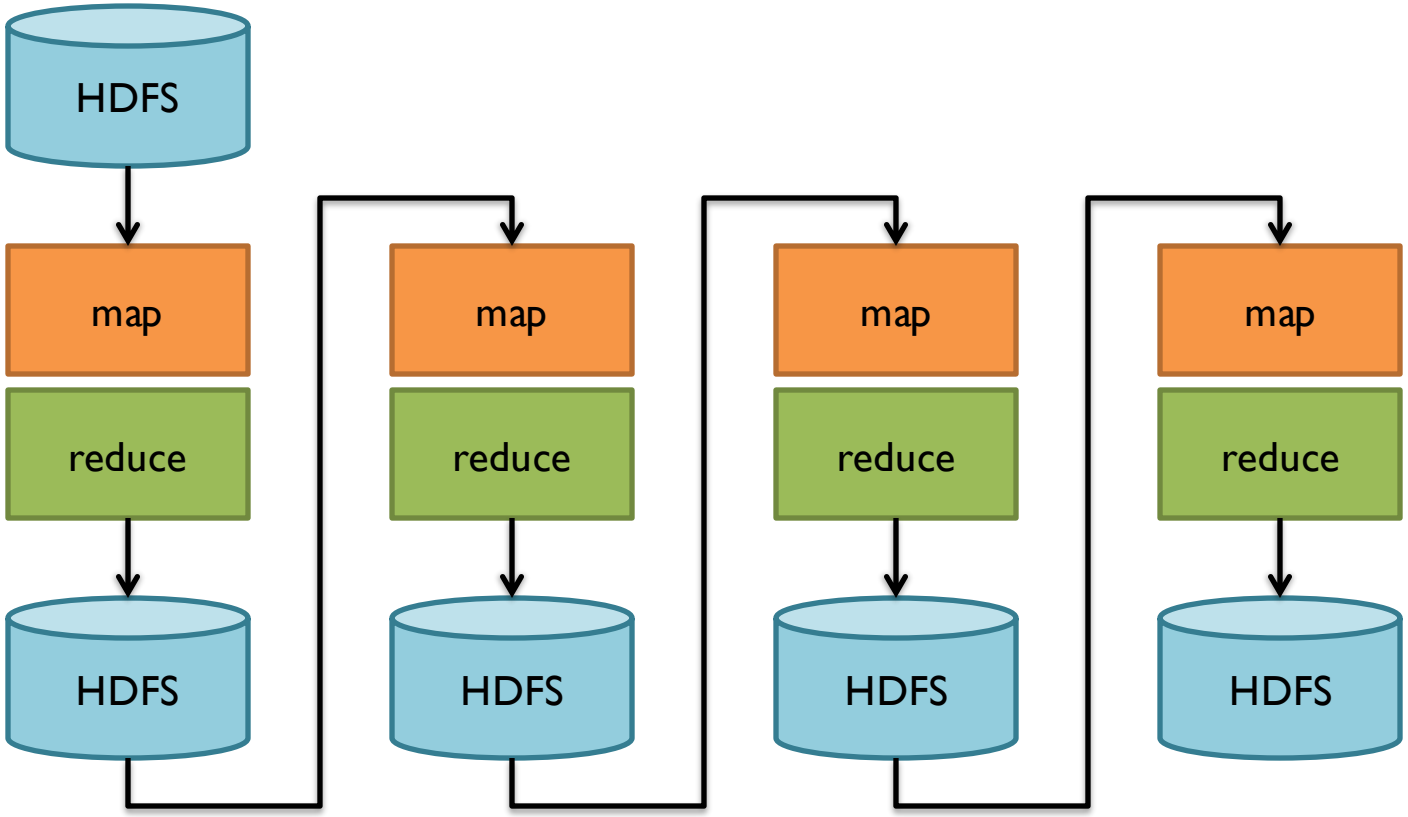


Both (still) open-source projects today!

MapReduce

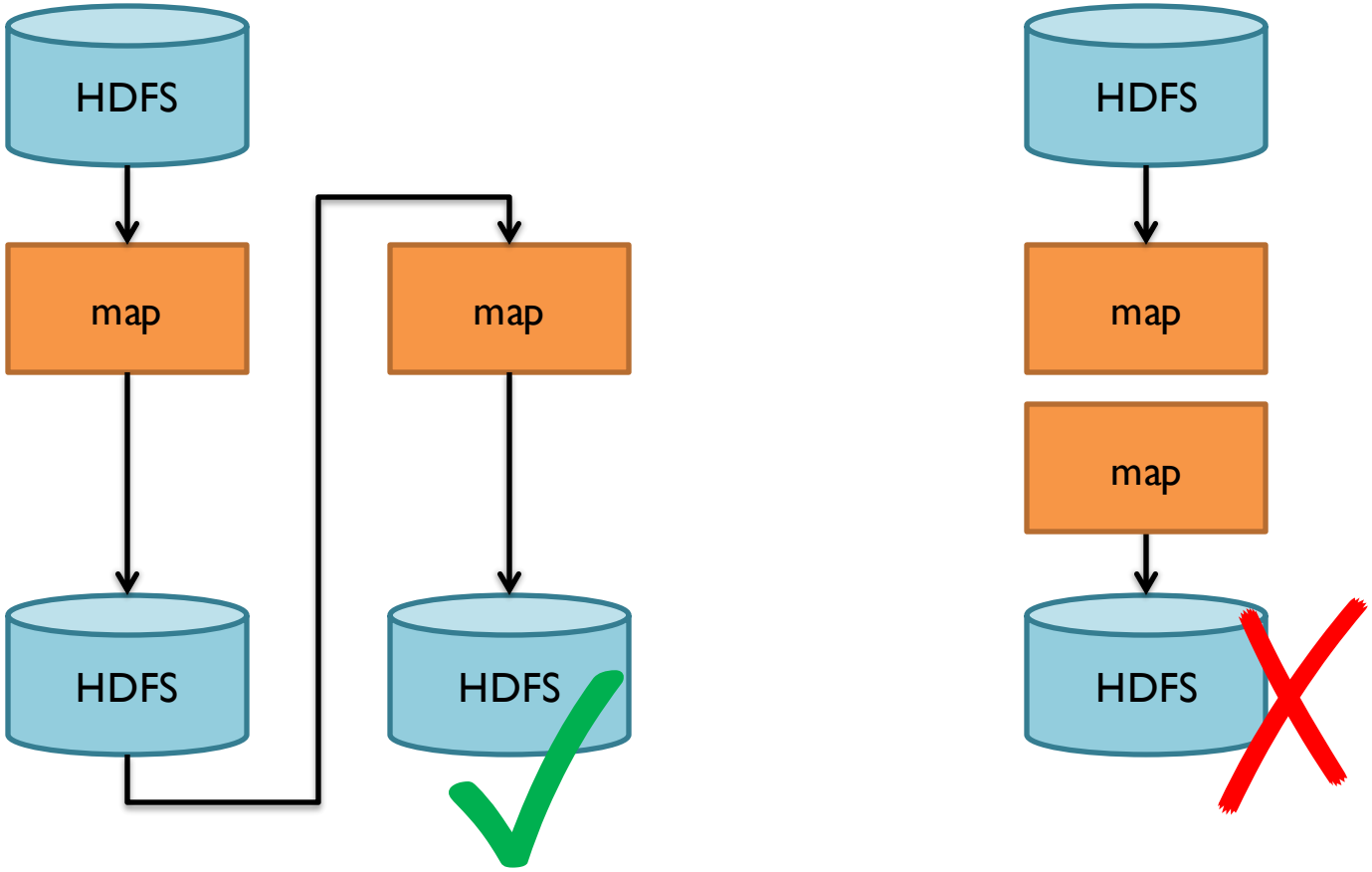
The “instruction set” is restrictive…
Can we do better?

MapReduce Workflows

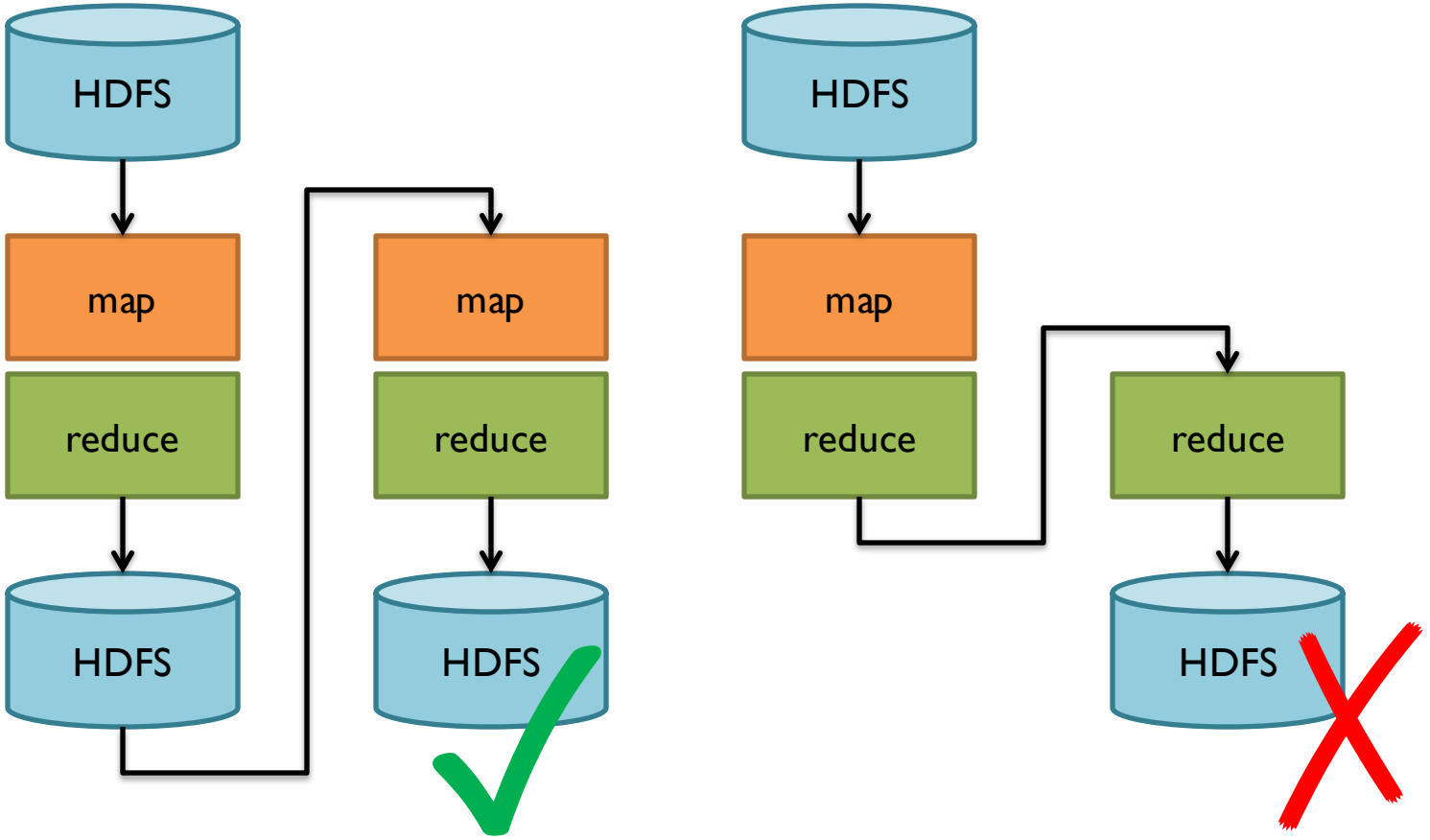


What's wrong?

Want MM?



Want MRR?


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

MapReduce

The “instruction set” is restrictive…
Can we do better?

Let's design a data processing framework from scratch!

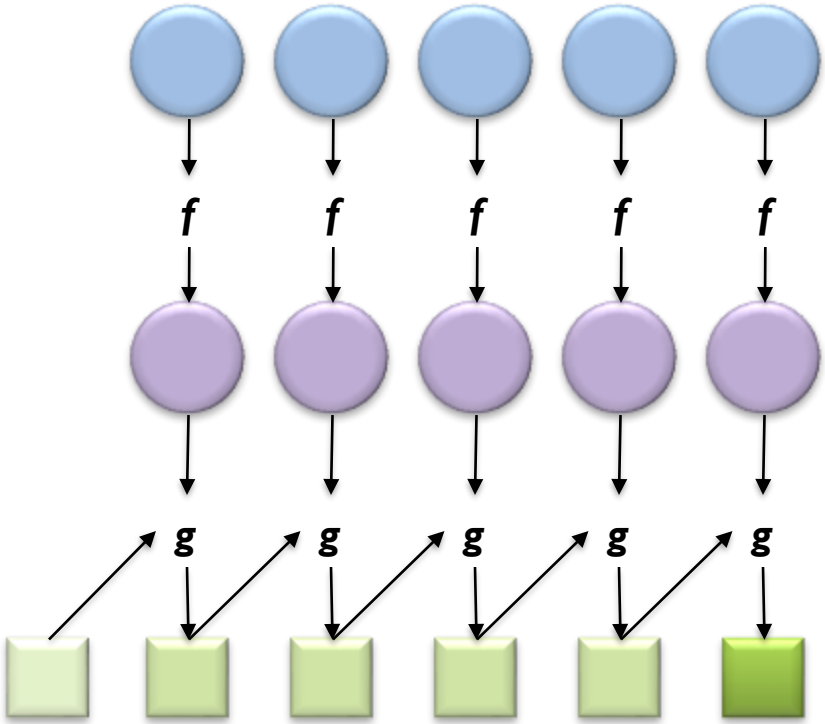
What ops do you need?

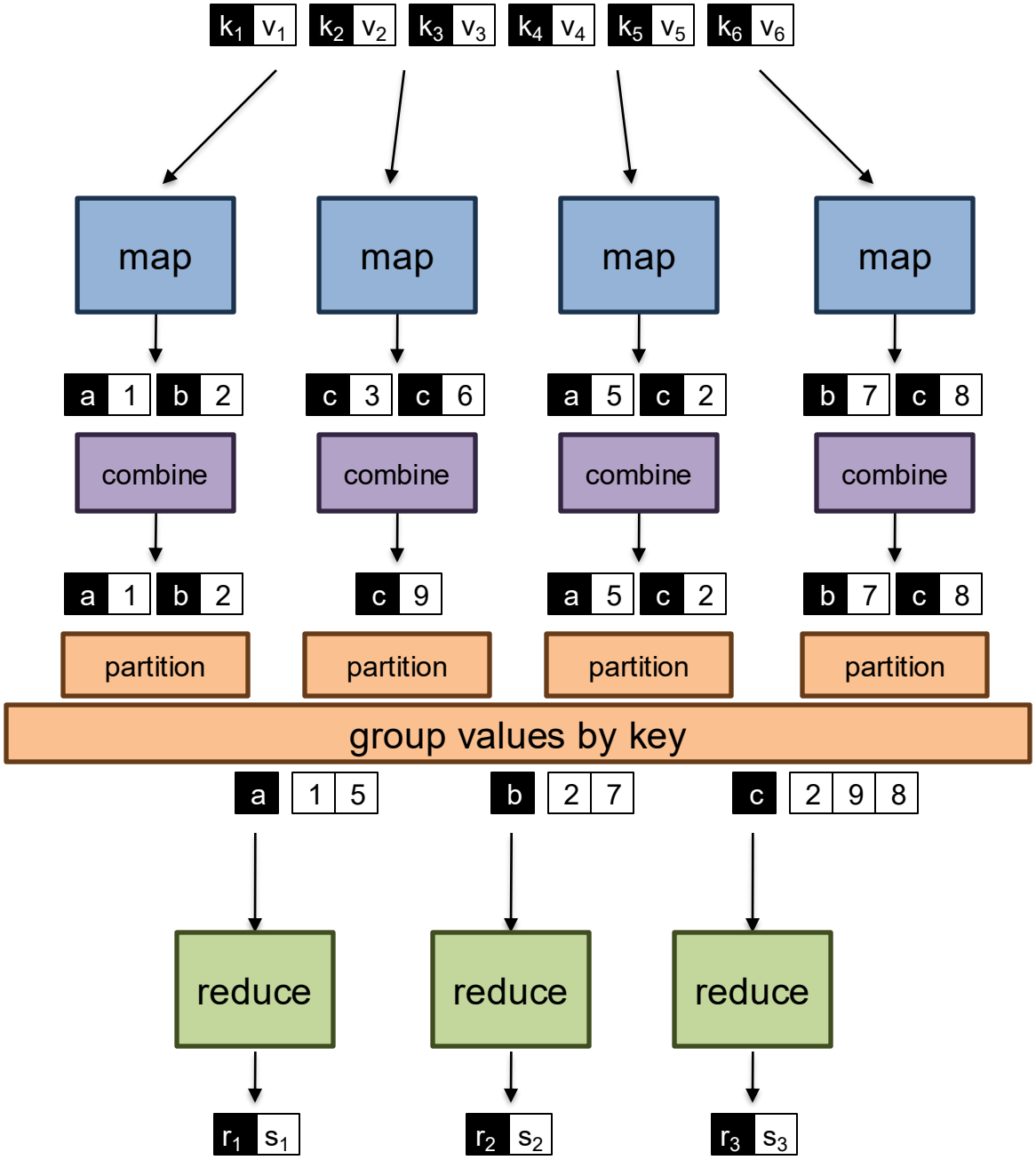
(Why is MapReduce the way it is?)

Roots in Functional Programming

Map

Fold




k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
combine
combine
combine
combine
a 1
b 2
c 9
a 5
c 2
b 7
c 8
partition
partition
partition
partition
group values by key
a 1 5
b 2 7
c 2 9 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Data-Parallel Dataflows

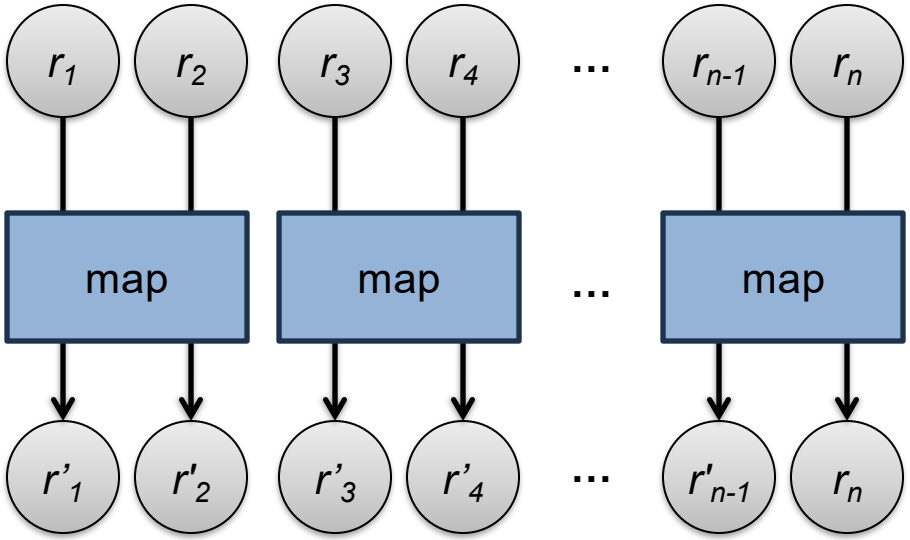
We have a collection of records,
want to apply a bunch of operations
to compute some result

Assumption: static collection of records
(What's the limitation here?)

Immutable Truth #1: At scale, you must distribute work across multiple machines.
(So, it's gotta be distributed)

Immutable Truth #2: At scale, computing components break all the time.
(So, fault tolerance is a must)

We need per-record processing

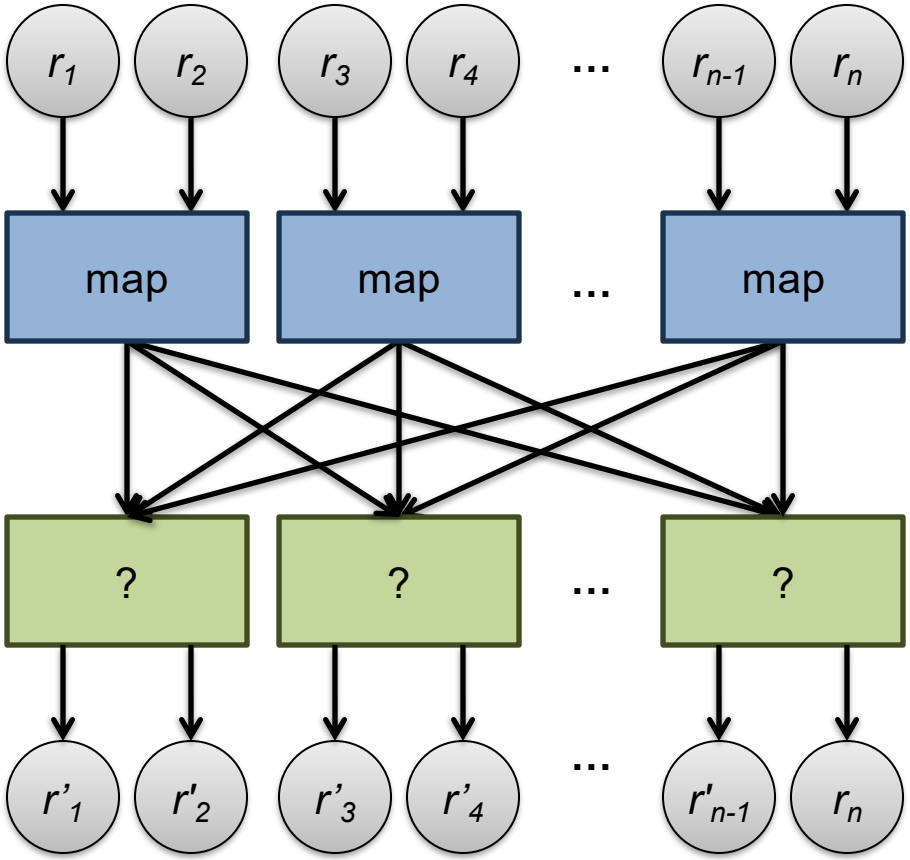


Okay, now what?

We need to exchange intermediate results…
We need an "addressing mechanism" and a "delivery mechanism"…

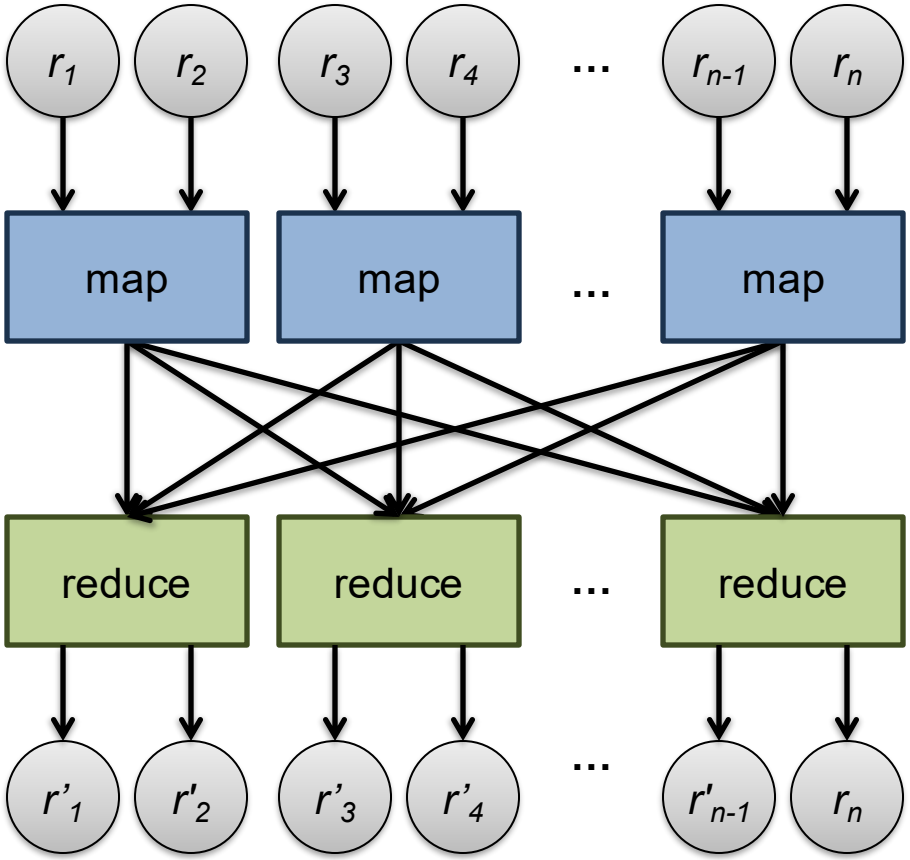
And once we do that, we can continue with processing…

We need communication

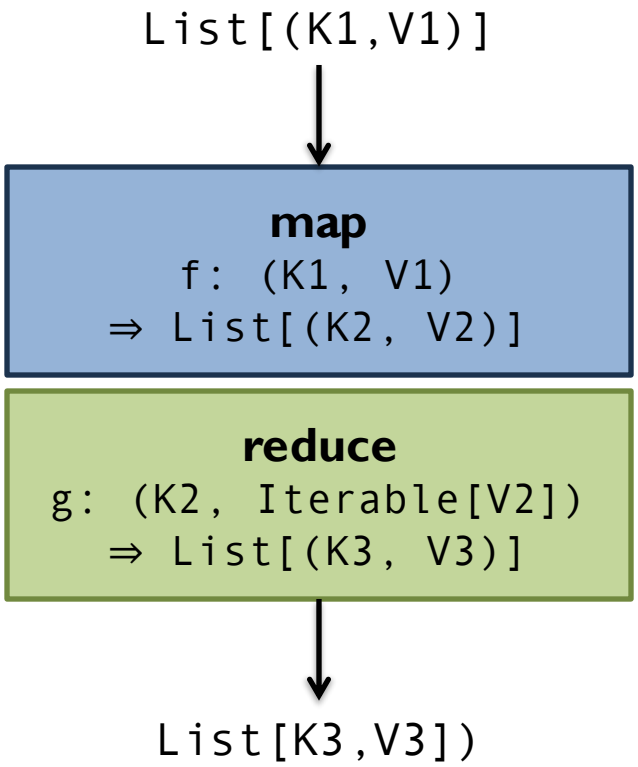


What's the role of the group by?

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

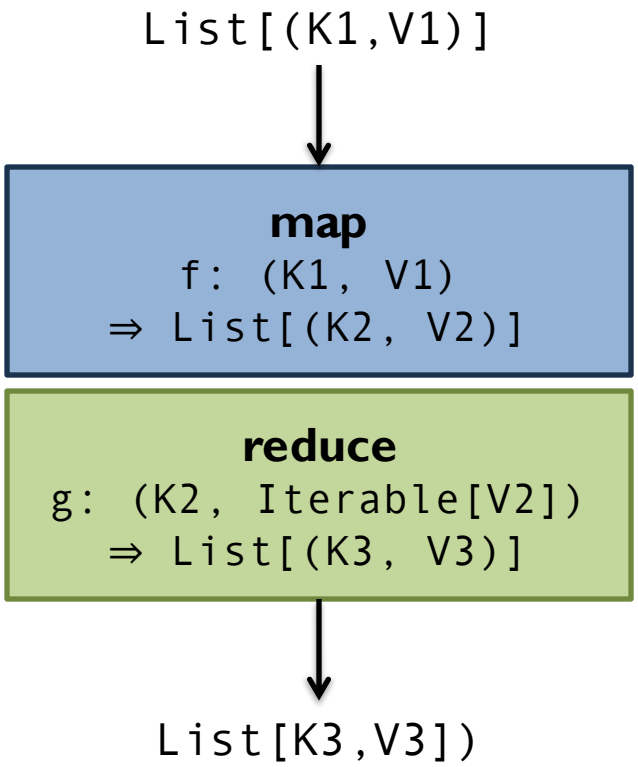


(note we're abstracting the data-parallel part)

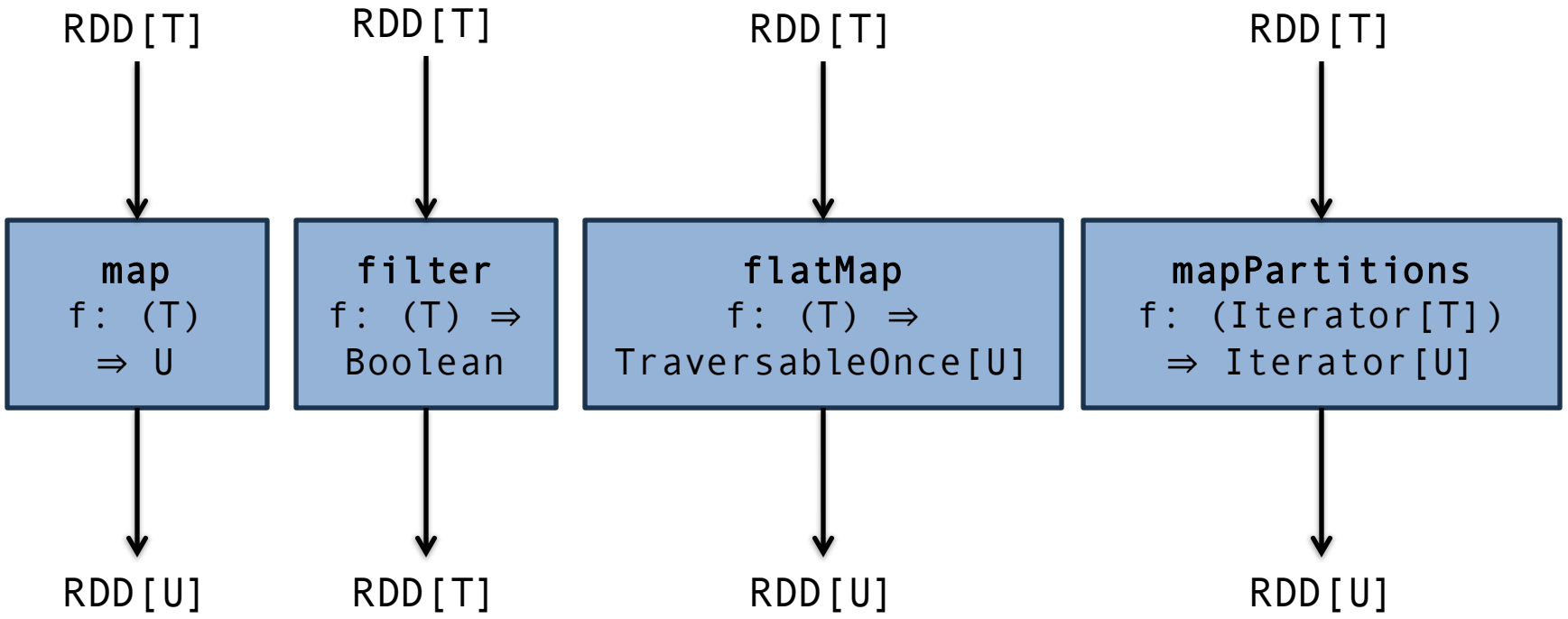
Well, if you put it like that…

`map` and `reduce` comprise the instruction set
What else can we do?

MapReduce

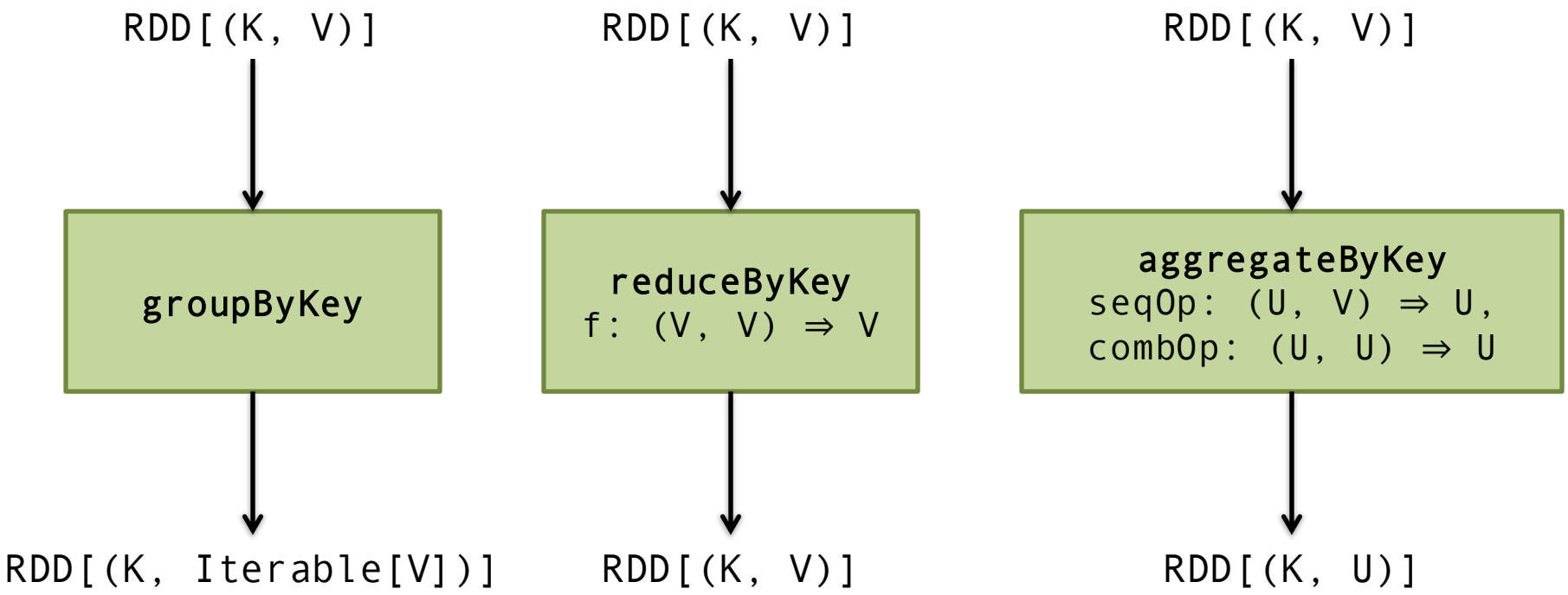

List[(K1,V1)]
map
f: (K1, V1)
⇒ List[(K2, V2)]
reduce
g: (K2, Iterable[V2])
⇒ List[(K3, V3)]
List[K3,V3])

Map-like Operations



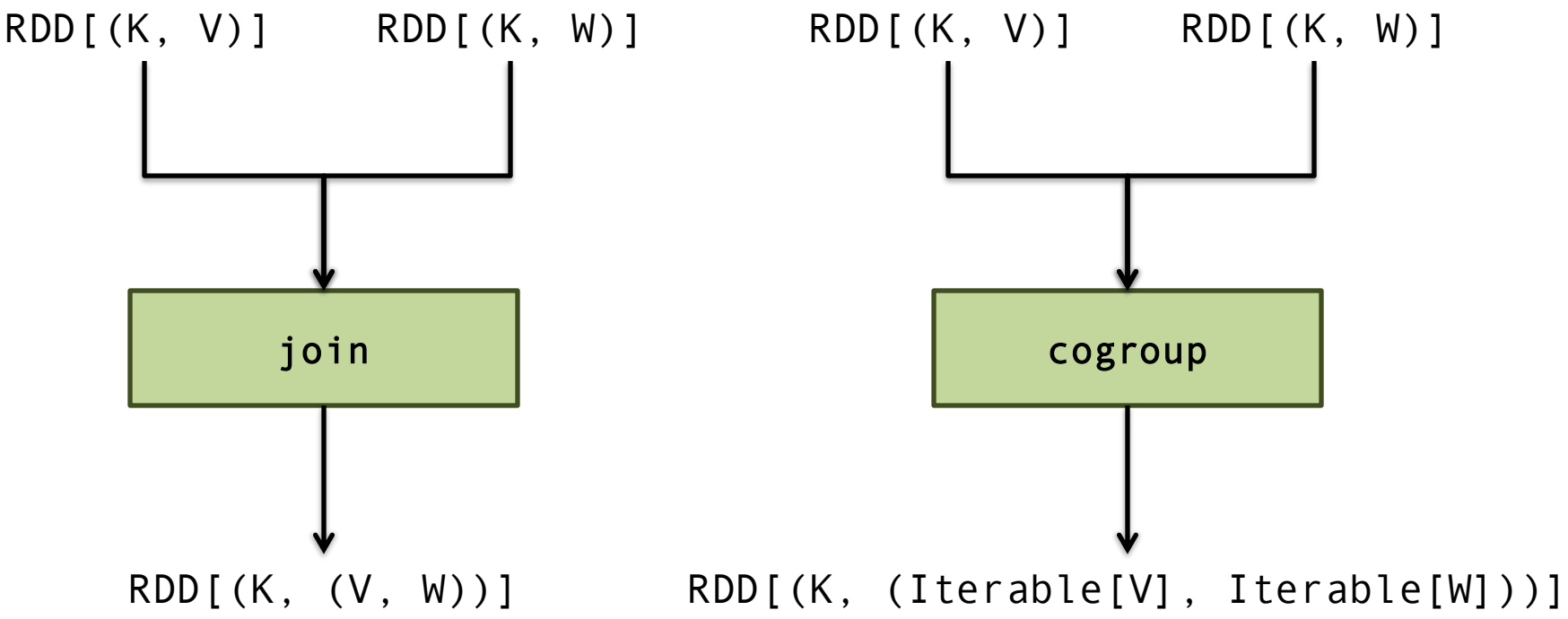
(Not meant to be exhaustive)

Reduce-like Operations



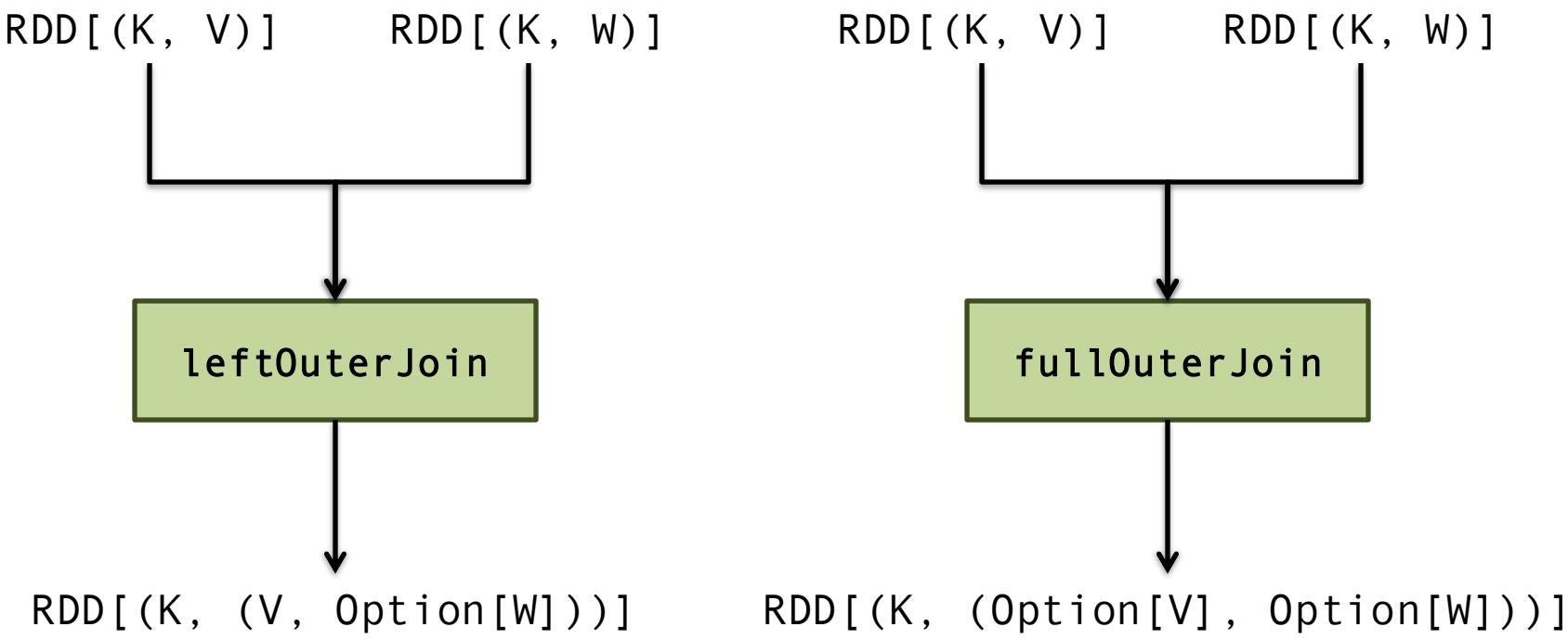
(Not meant to be exhaustive)

Join-like Operations



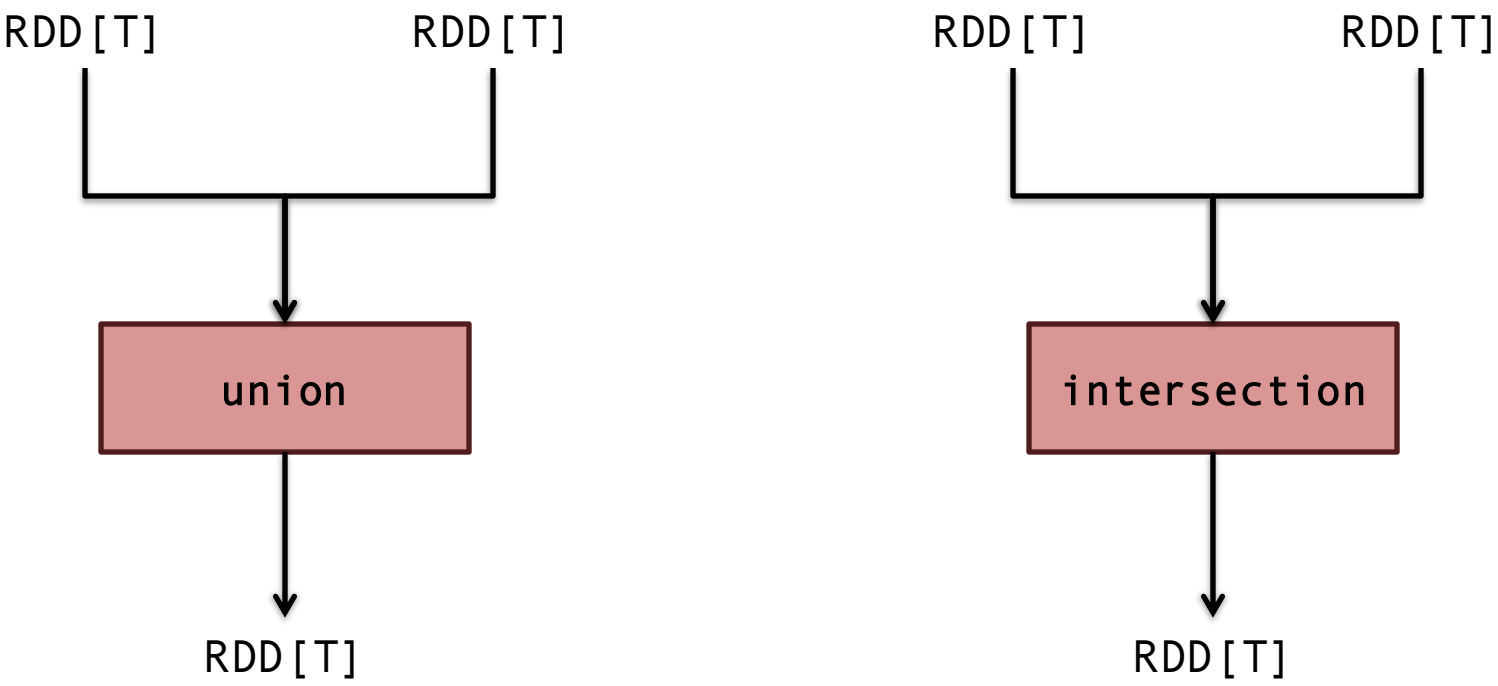
(Not meant to be exhaustive)

Join-like Operations



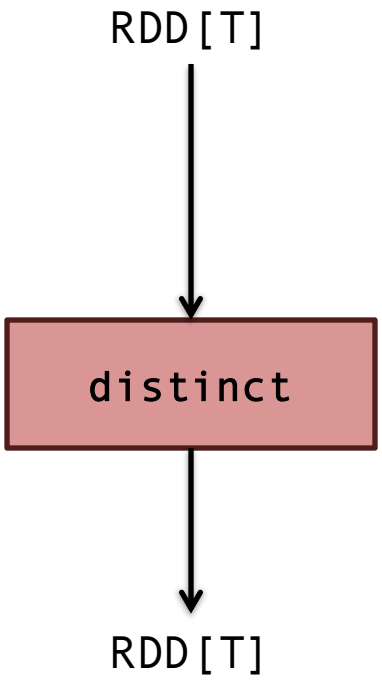
(Not meant to be exhaustive)

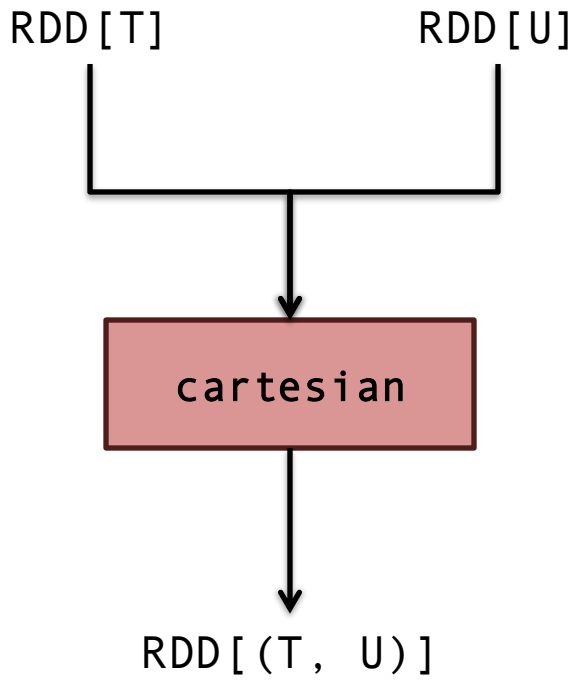
Set-ish Operations



(Not meant to be exhaustive)

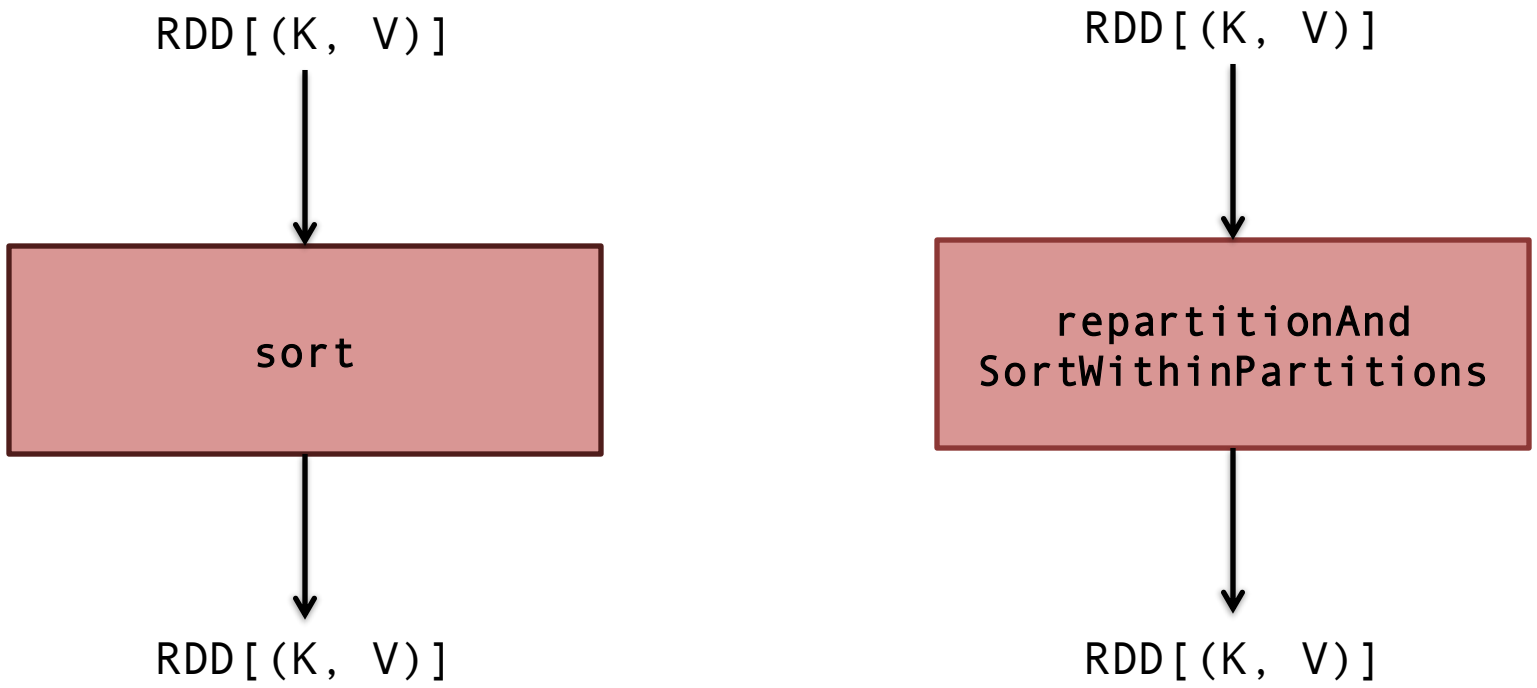
Set-ish Operations





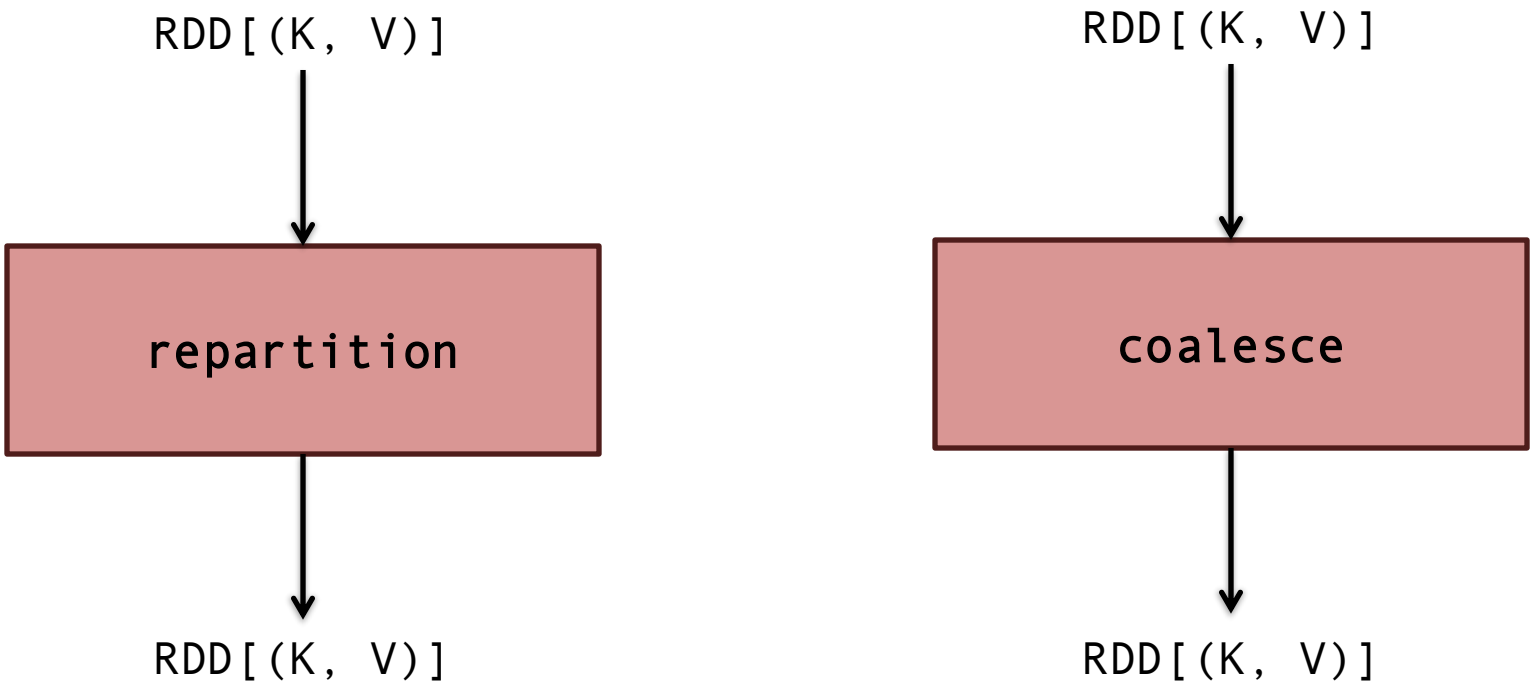
(Not meant to be exhaustive)

Sort Operations



(Not meant to be exhaustive)

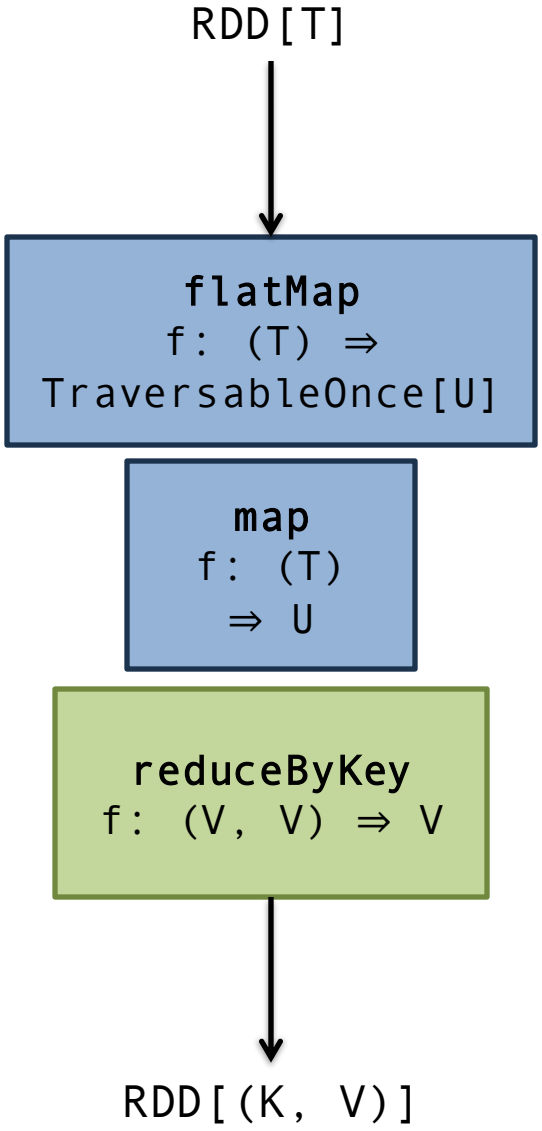
Partition Operations



(Not meant to be exhaustive)

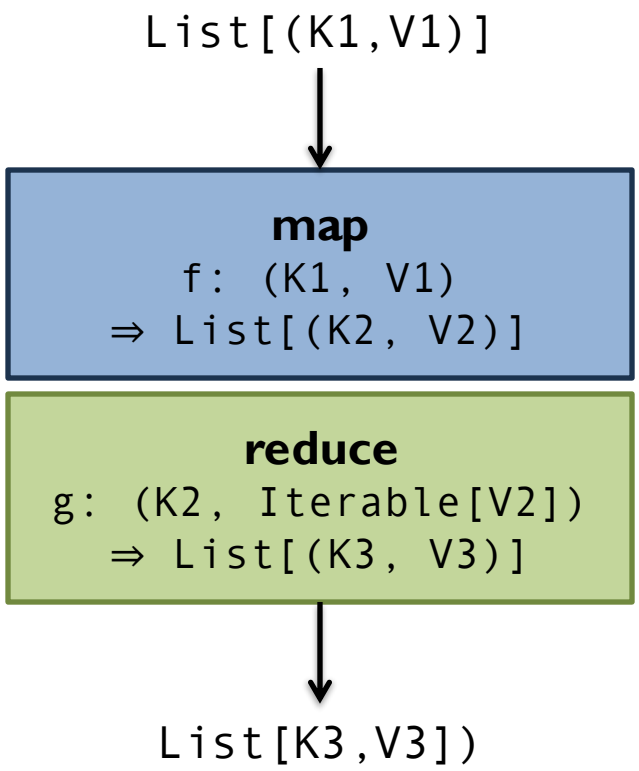
Example: Spark Word Count

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

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



Hypothetical Example: MapReduce

```
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())
```



That's it.

MapReduce

```
results = records.map(...)
                 .reduce(...)

results2 = results1.map(...)
                   .reduce(...)
```

Spark

```
results = rdd.foo(...)
             .bar(...)
             .baz(...)
```

Data-Parallel Dataflows

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

Assumption: static collection of records
(What's the limitation here?)

Immutable Truth #1: At scale, you must distribute work across multiple machines.
(So, it's gotta be distributed)

We defined a bunch of transformations…

Immutable Truth #2: At scale, computing components break all the time.
(So, fault tolerance is a must)

Let's work on this part now…

What's an RDD?

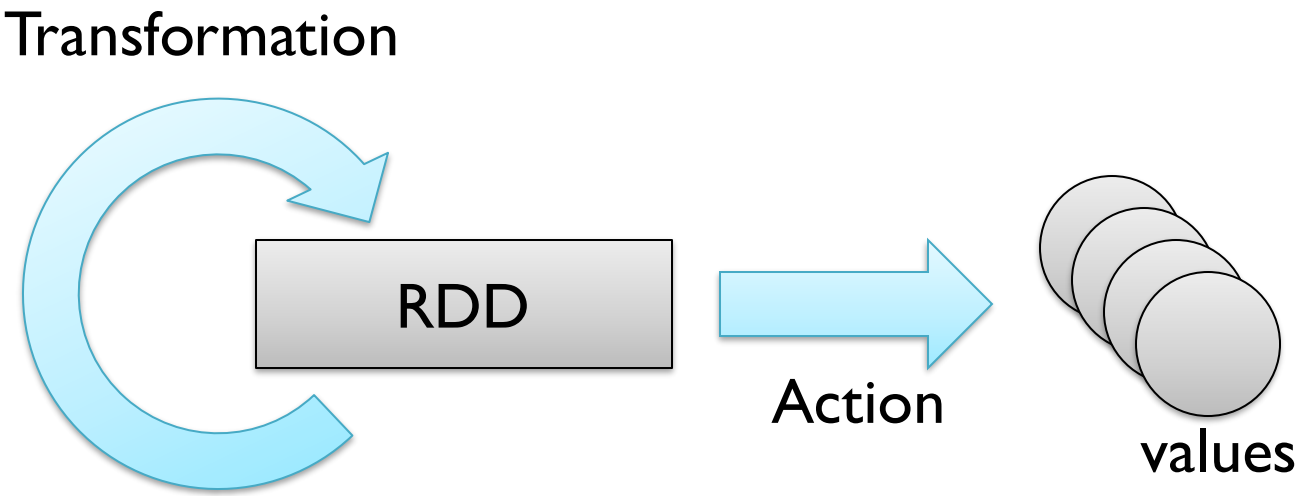
Resilient Distributed Dataset (RDD)

= immutable = partitioned

So how do you *actually* do something?

- Developers define transformations on RDDs
- Framework keeps track of lineage

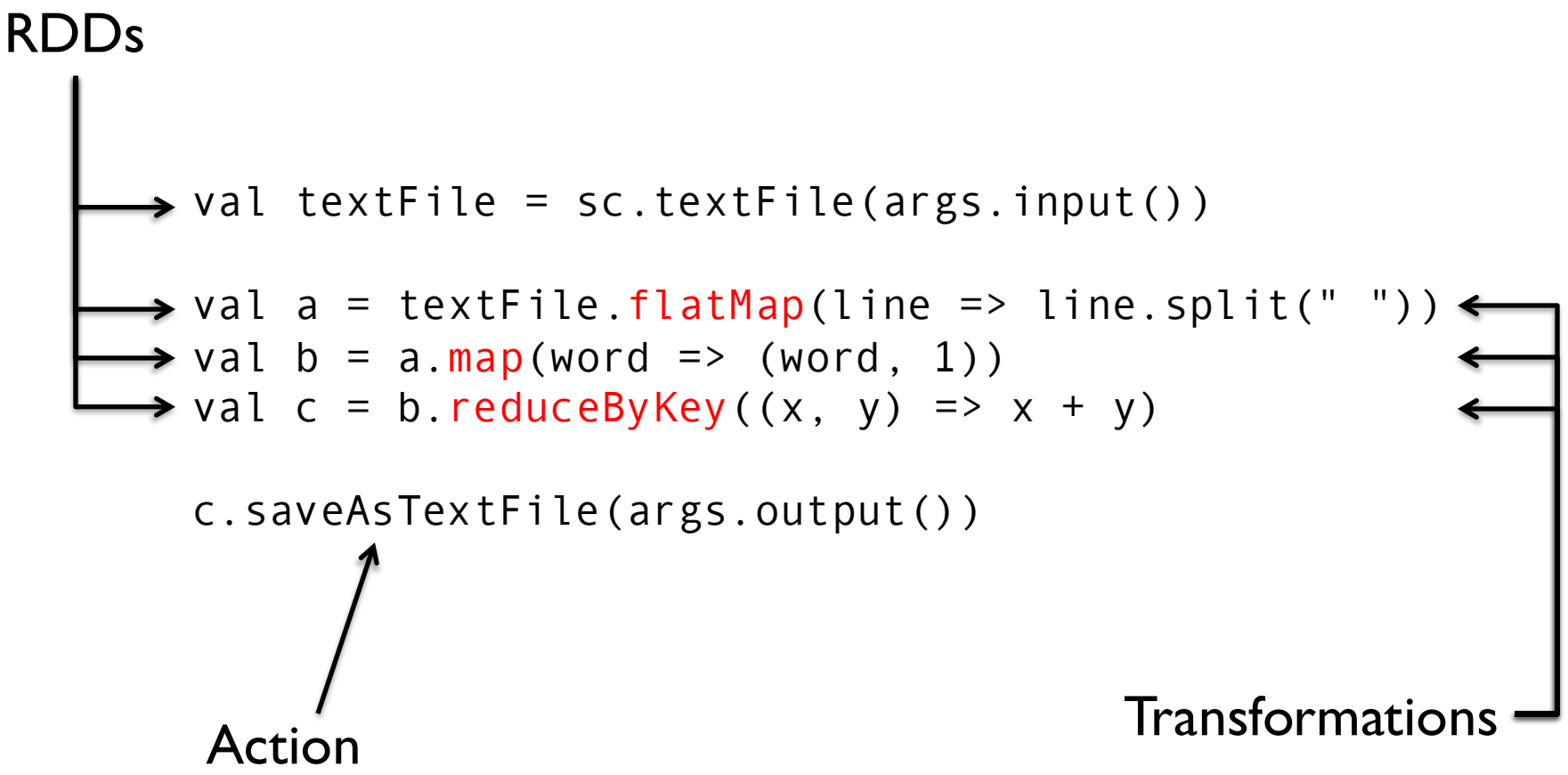
RDD Lifecycle



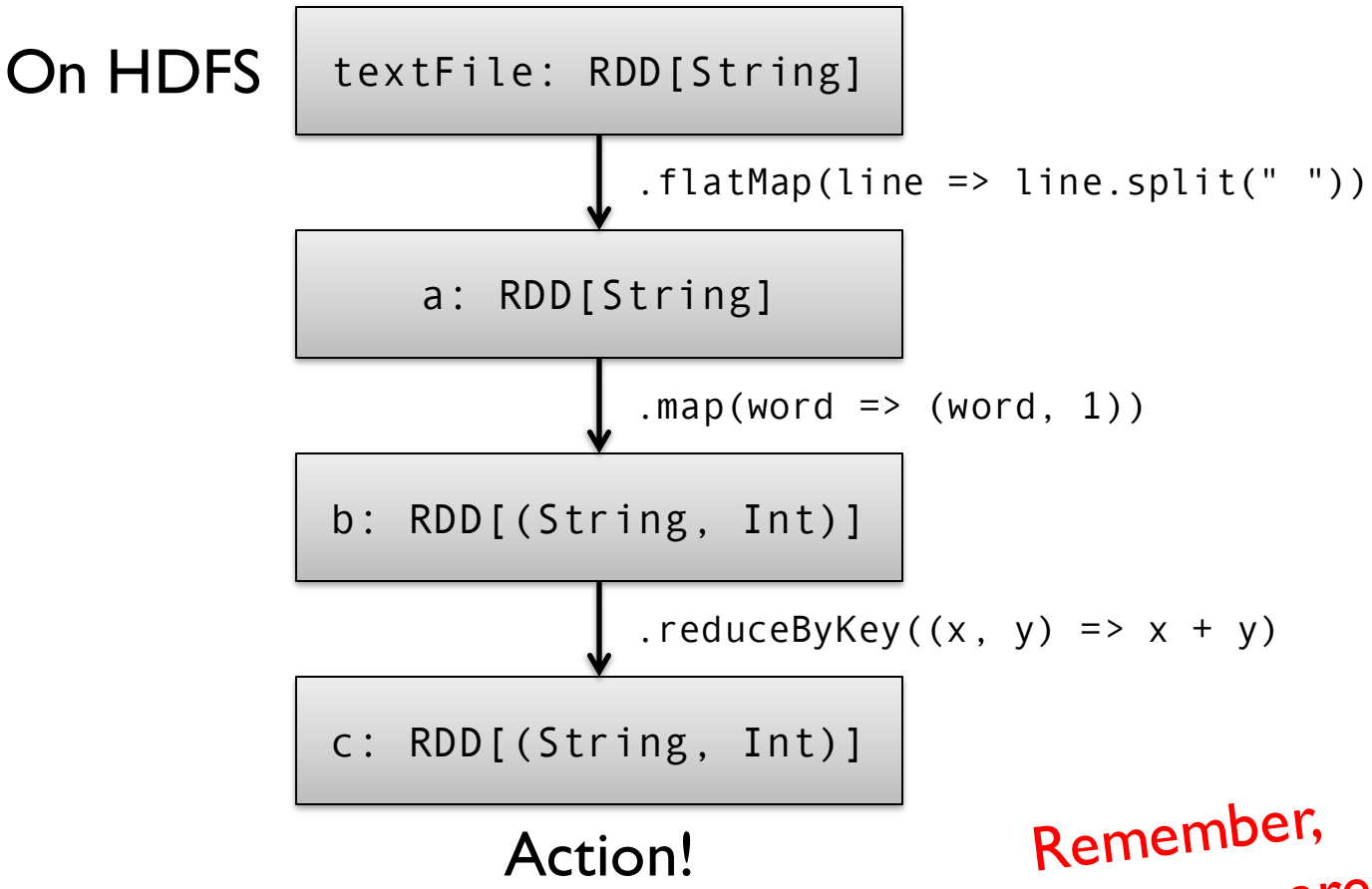
Transformations are lazy:
Framework keeps track of lineage

Actions trigger actual execution

Spark Word Count



RDDs and Lineage



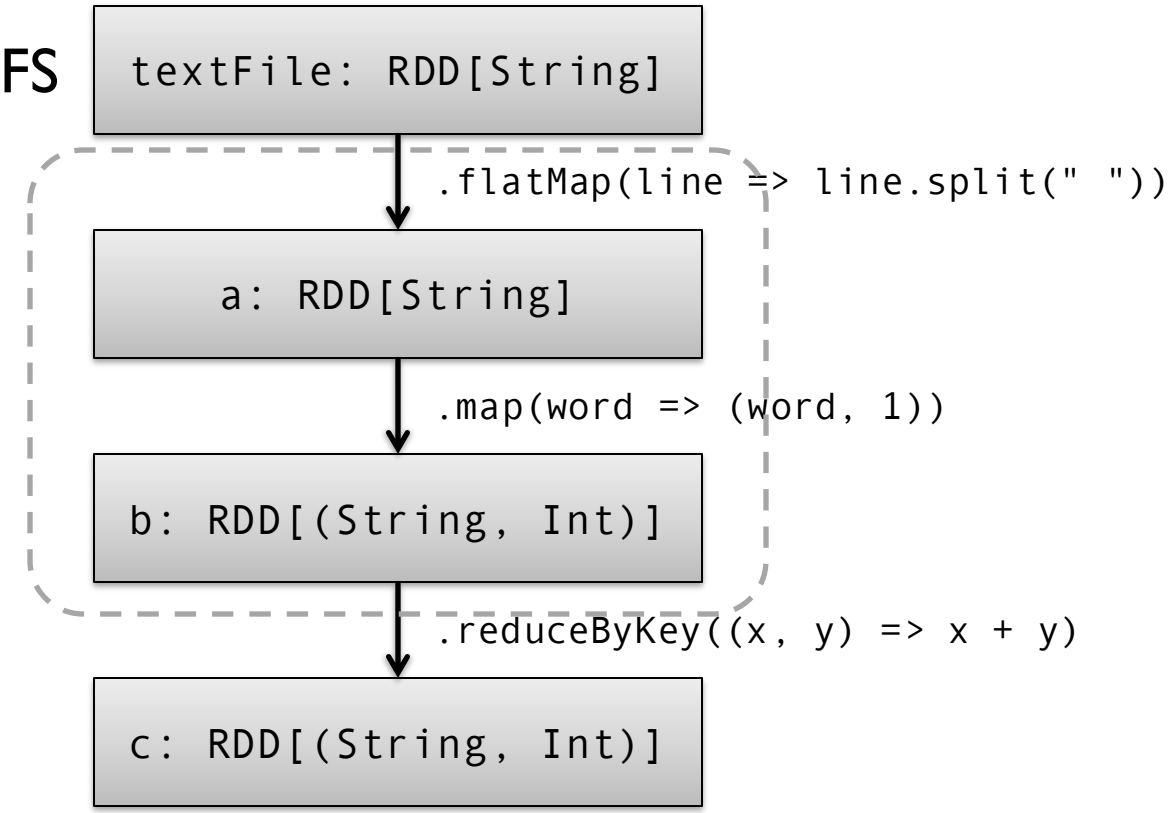
RDDs and Optimizations

Lazy evaluation creates optimization opportunities

On HDFS

RDDs don't need to be materialized!

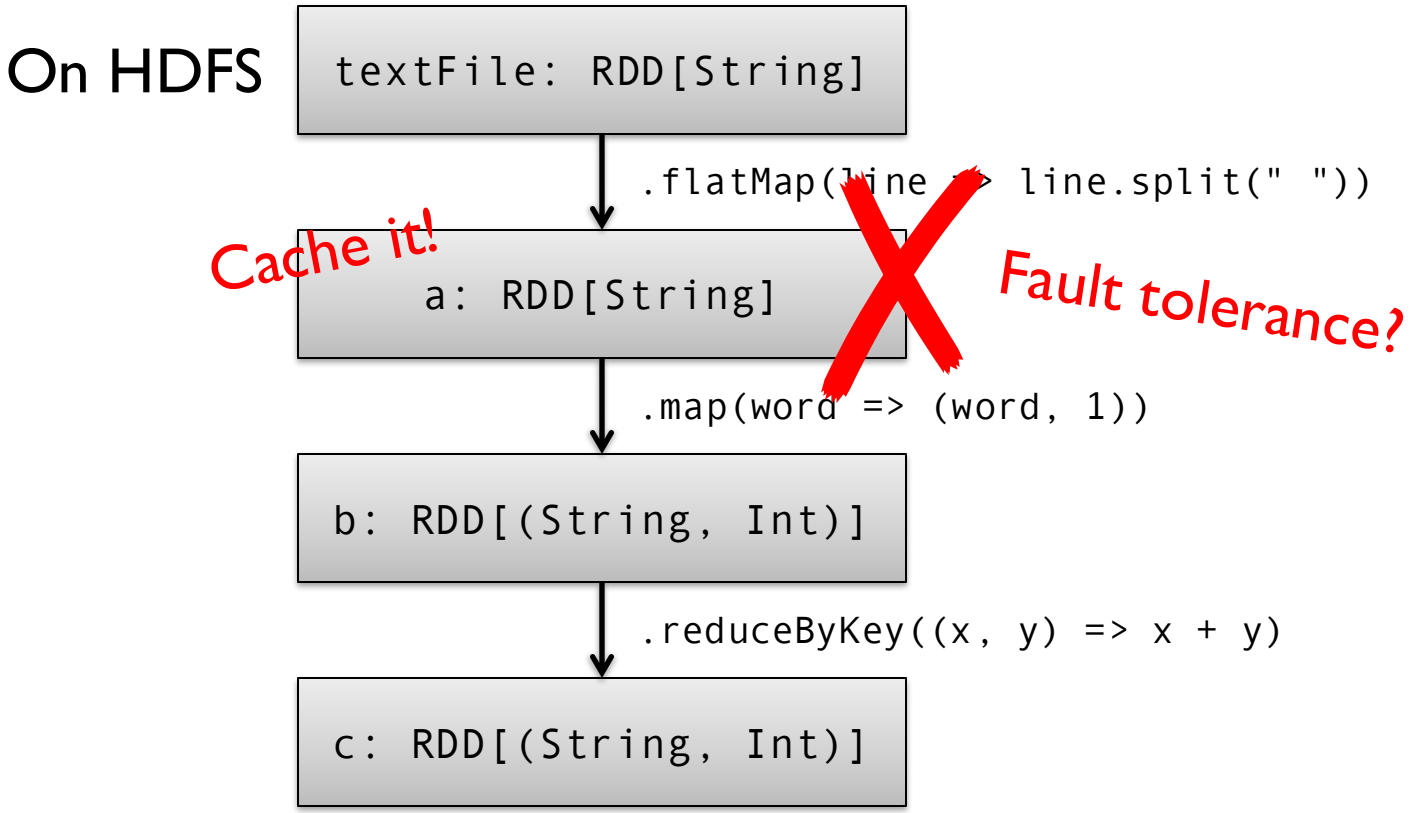
Want MM?



Action!

RDDs and Caching

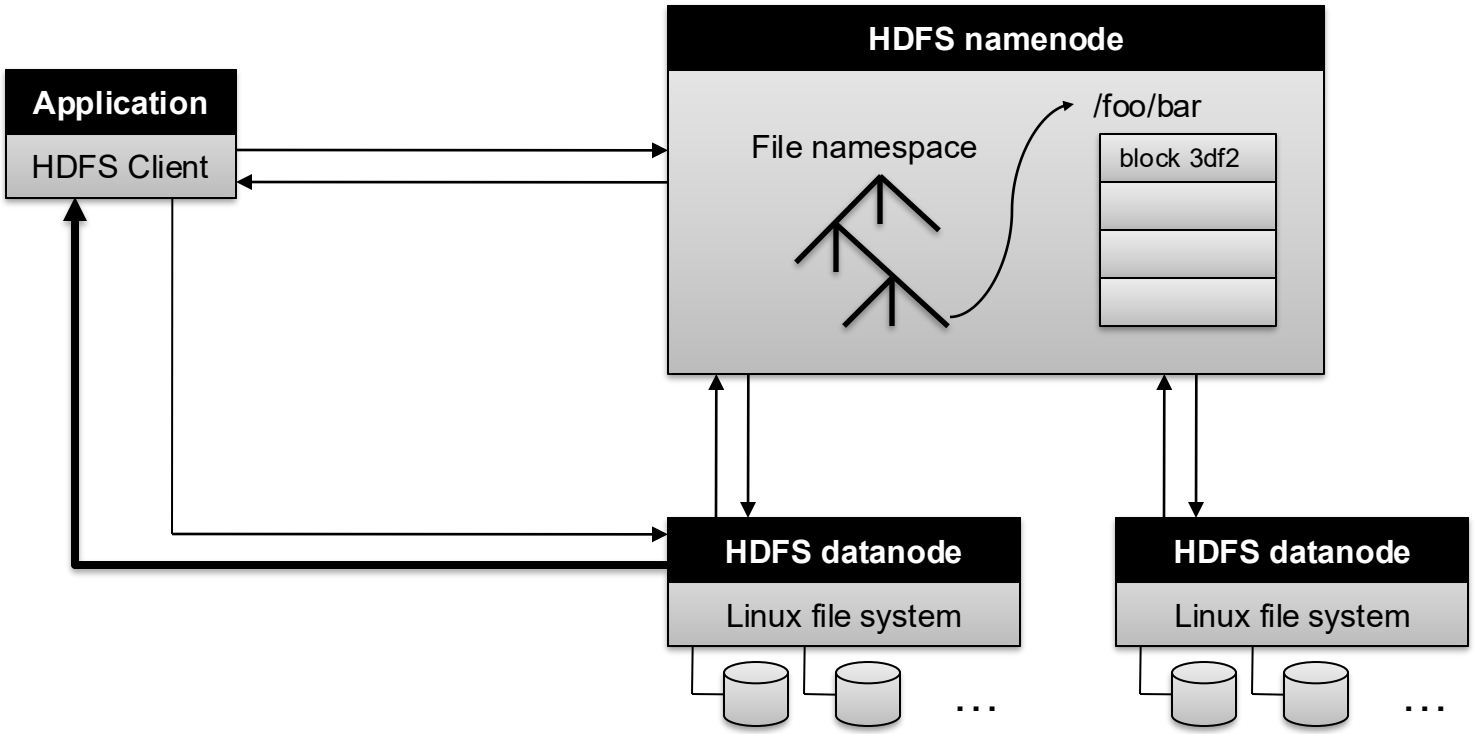
RDDs can be materialized in memory (and on disk)!



Action!

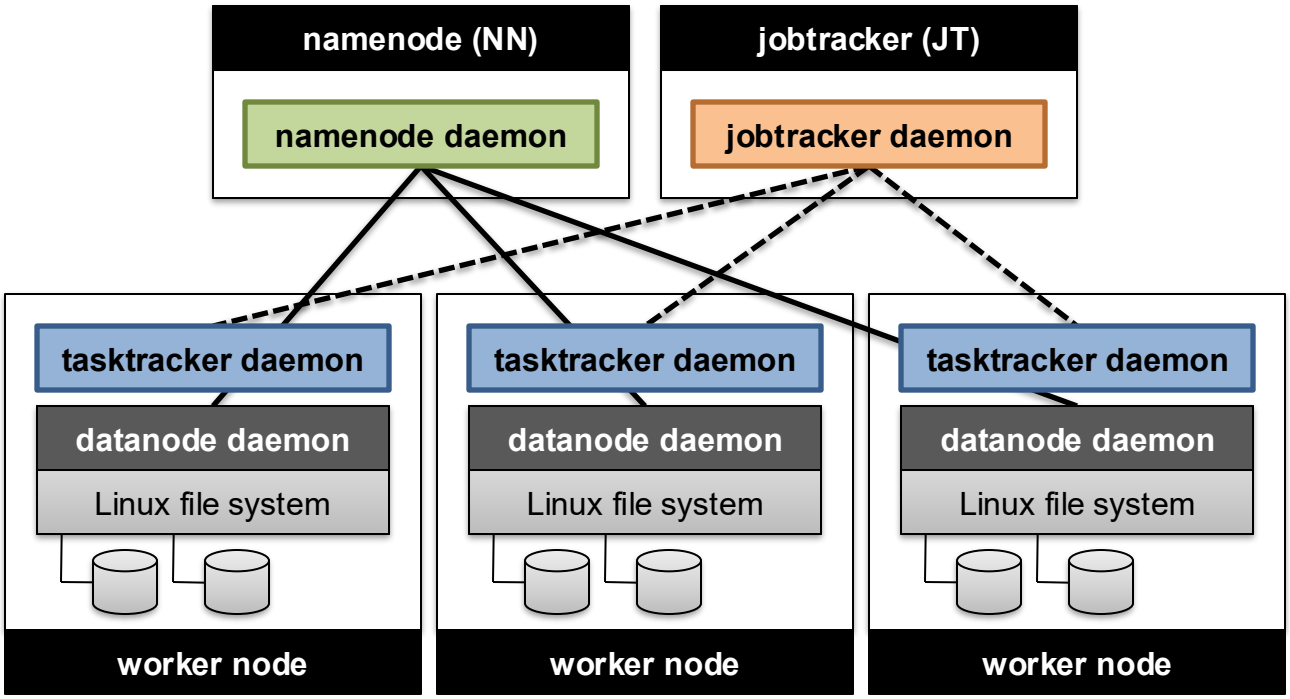
Spark works even if the RDDs are *partially* cached!

HDFS Architecture

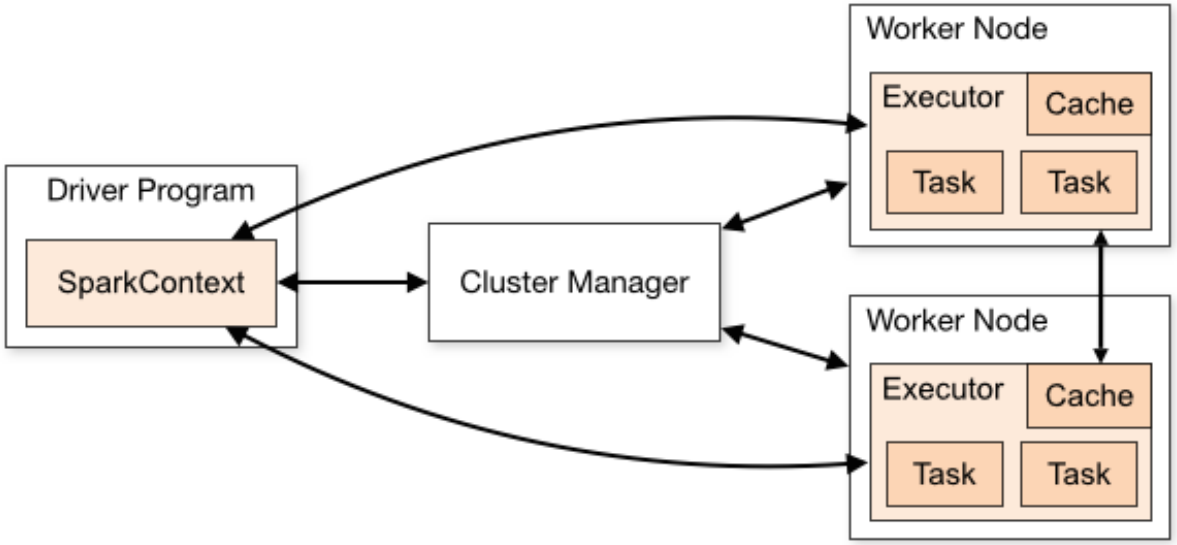


Adapted from (Ghemawat et al., SOSP 2003)

Hadoop Cluster

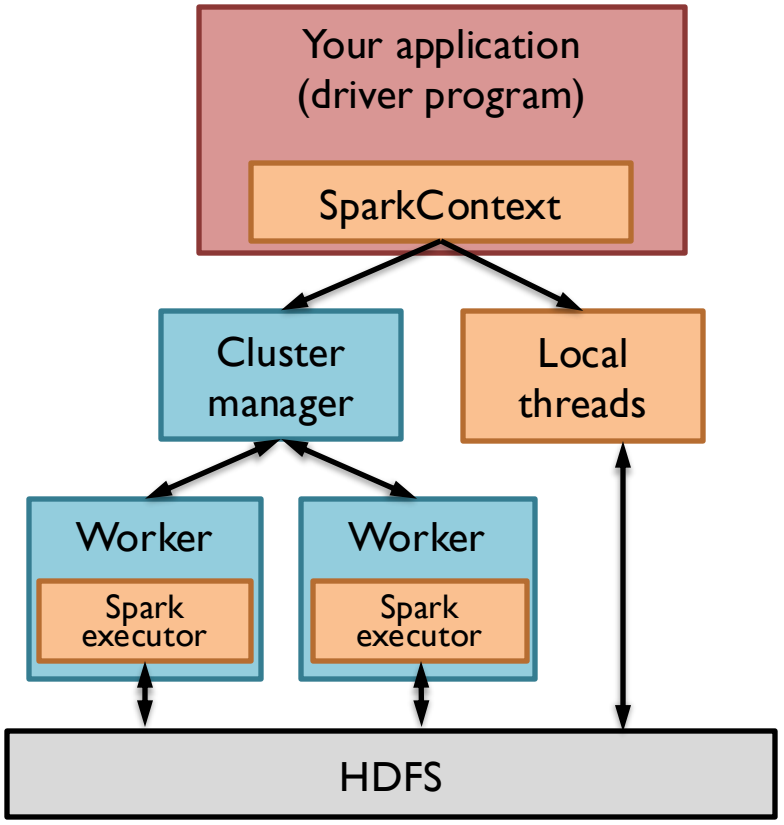


Spark Cluster



Spark Programs
Scala, Java, Python, R

`spark-shell` `spark-submit`



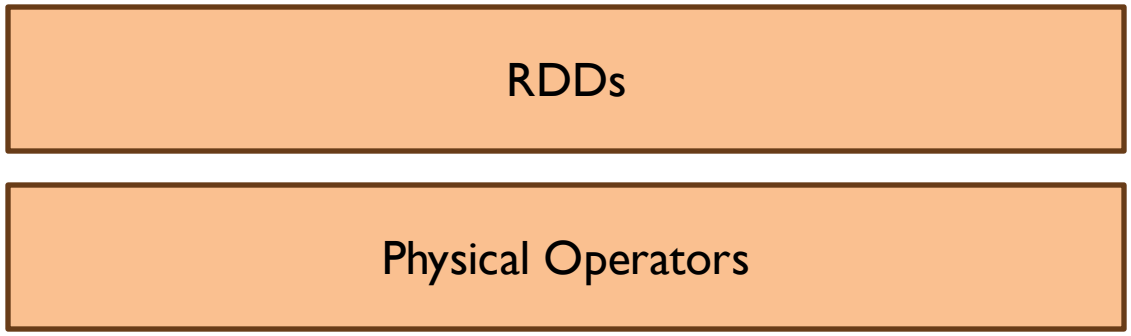
Spark context: tells the framework where to find the cluster

Use it to create RDDs to get started…

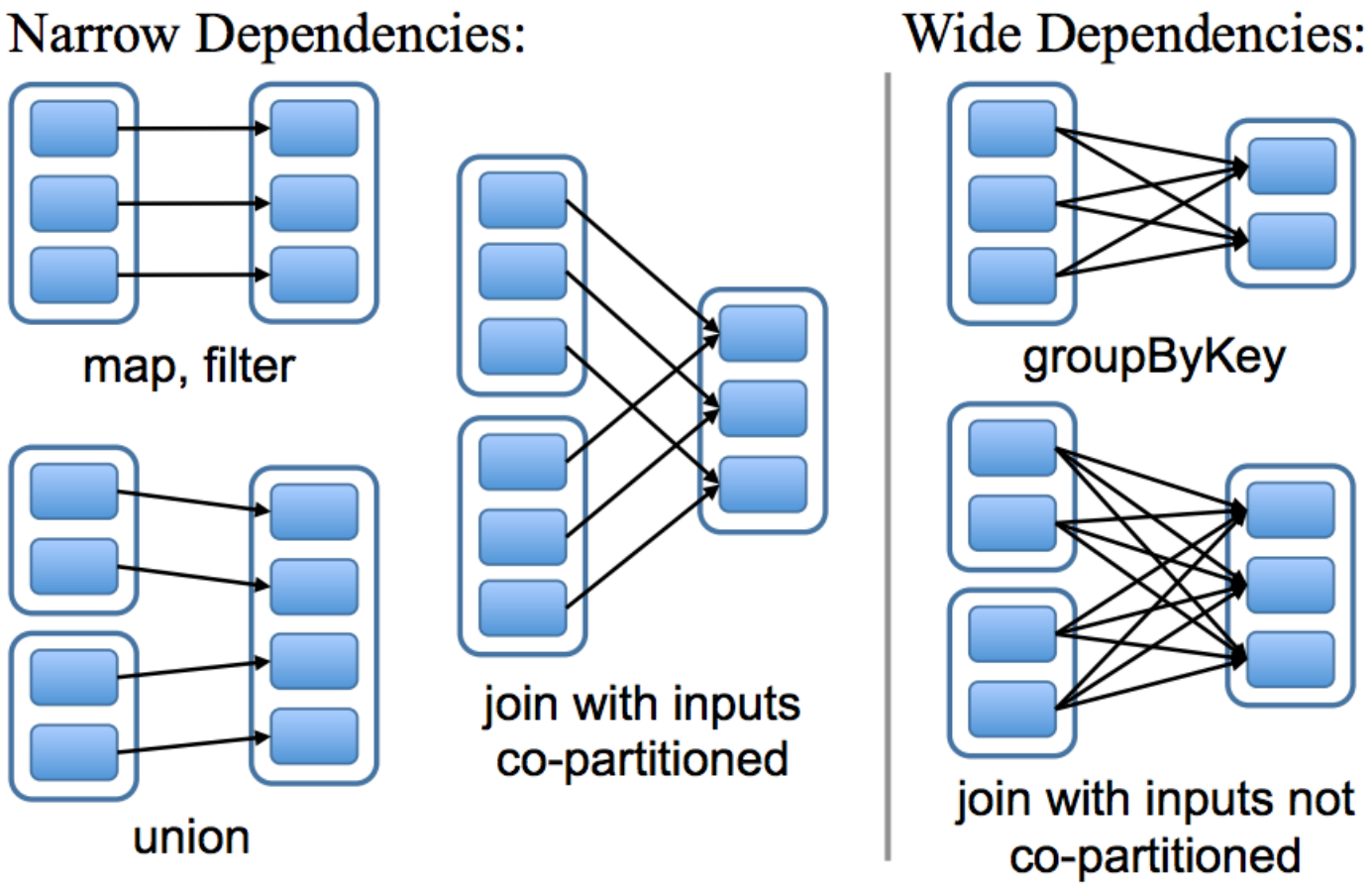
```
val textFile =
  sc.textFile(args.input())

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

What happens to the functions?
Where do the results go?
Beware of `collect()`!

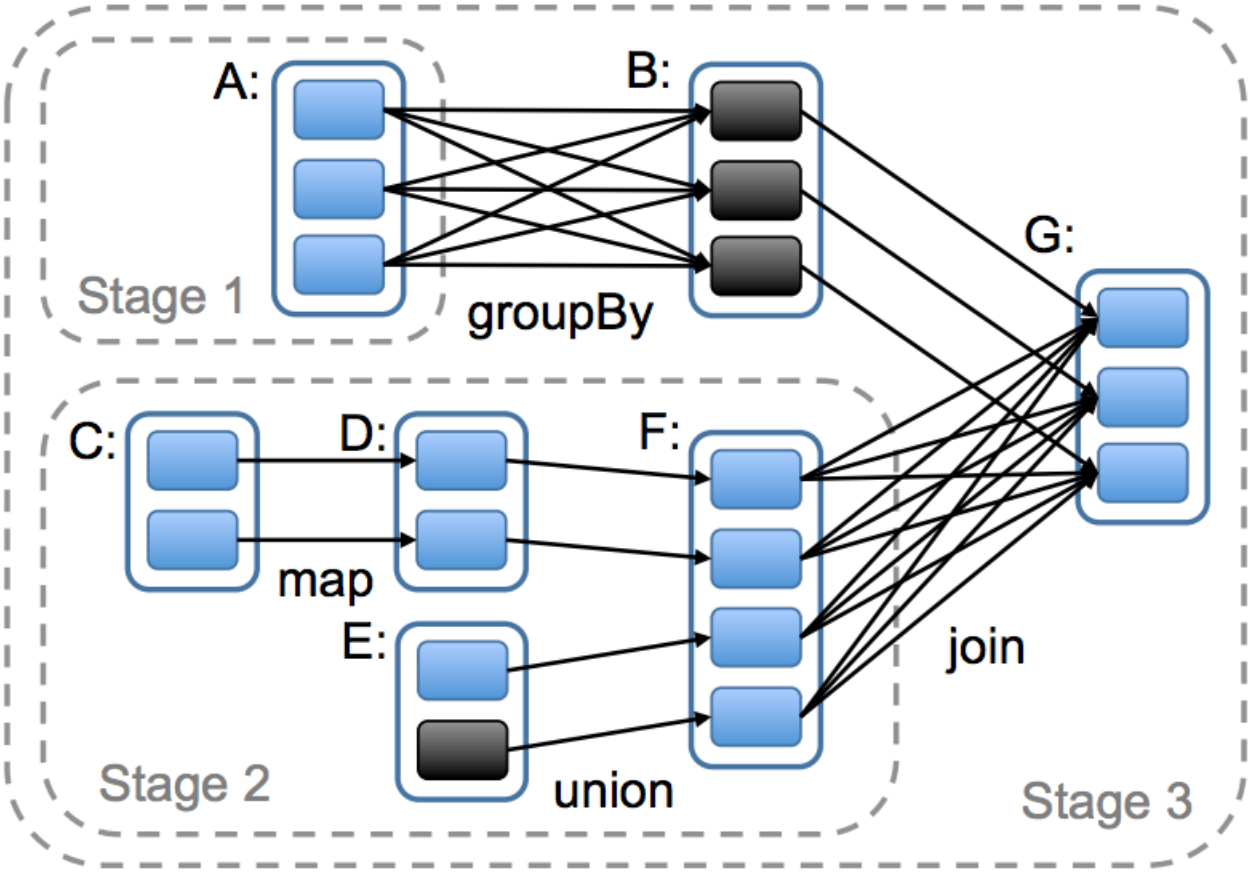
Spark Stack



Spark Physical Operators



Spark Execution Plan



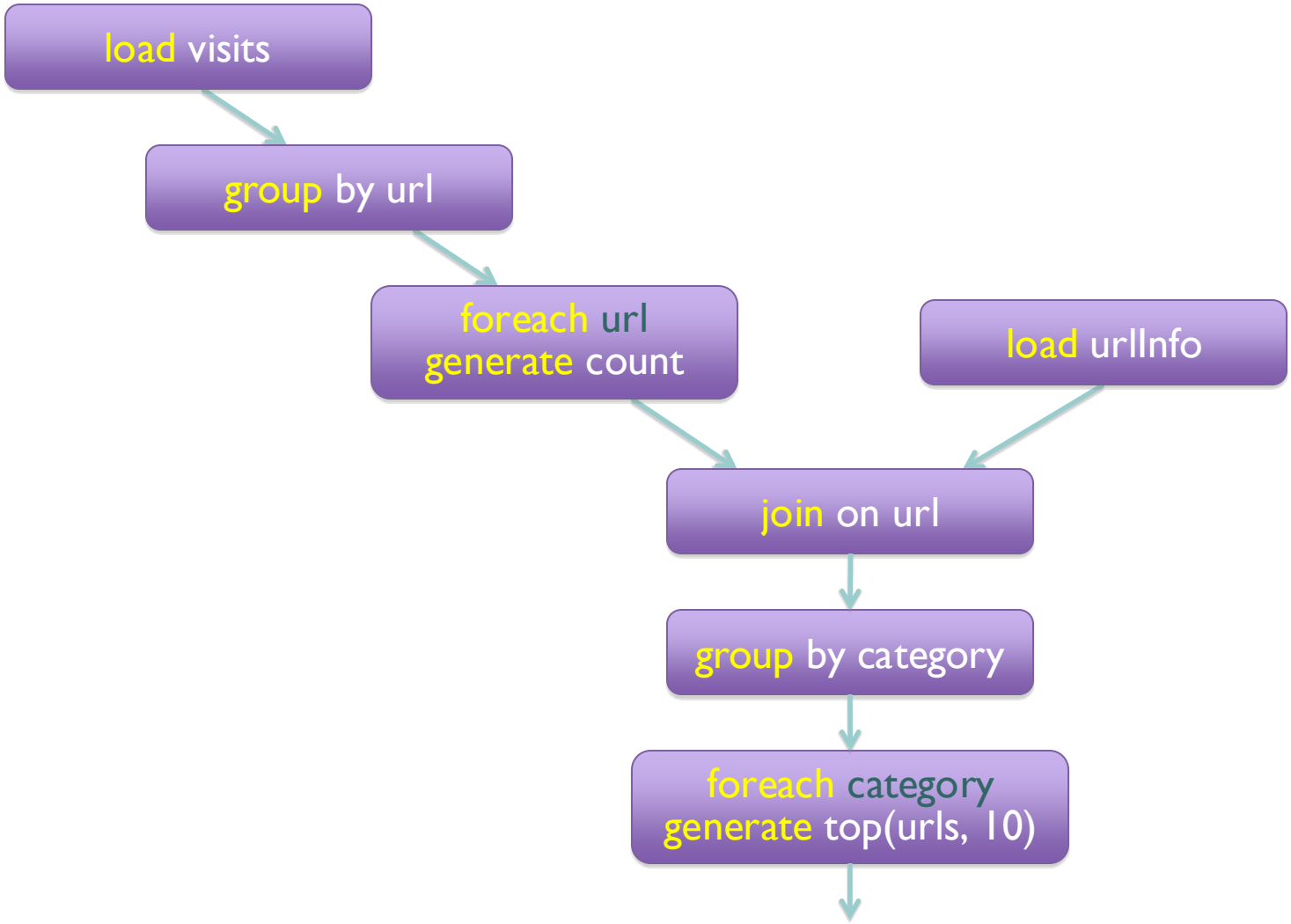
Wait, where have we seen this before?

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';
```

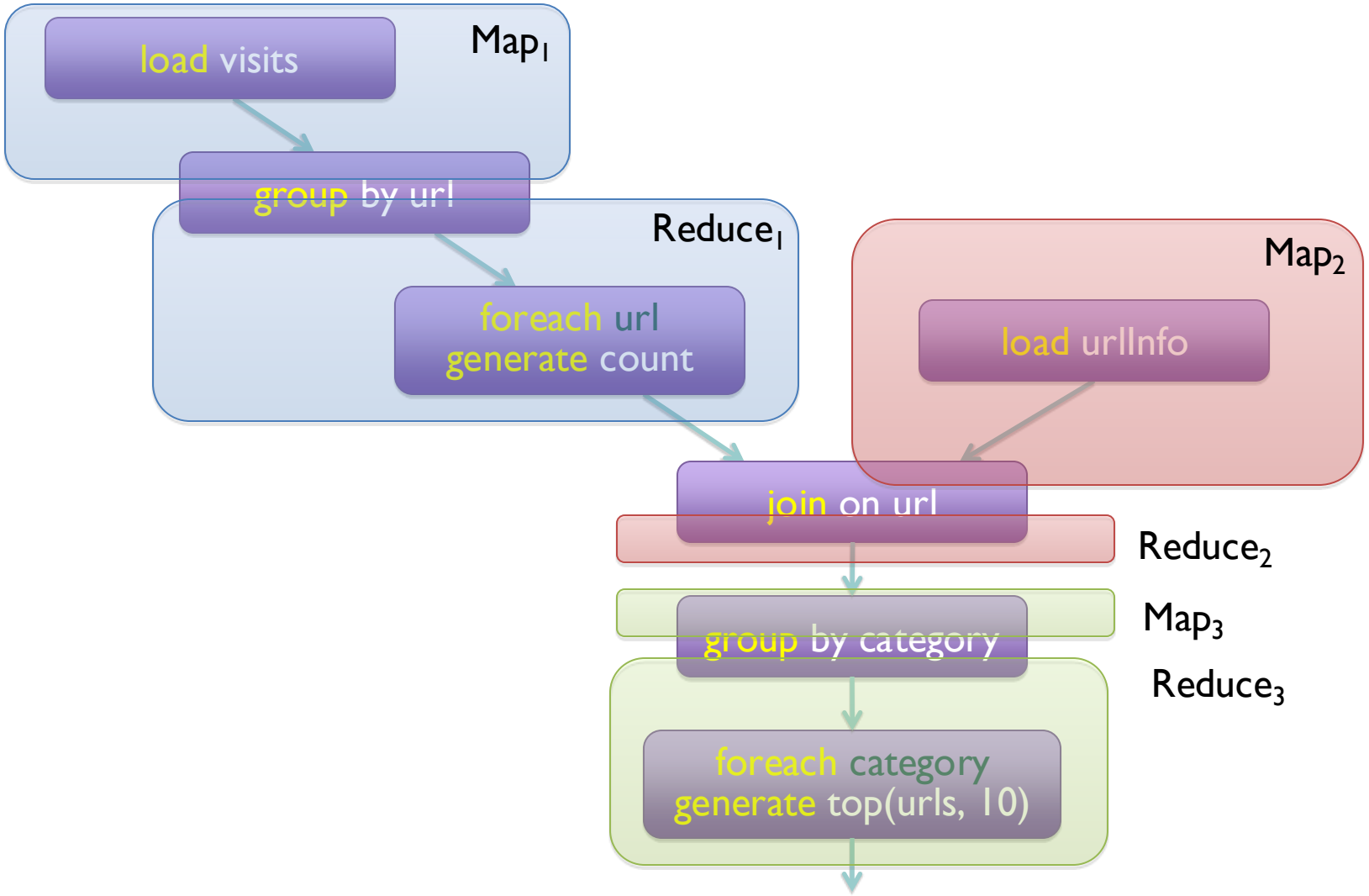
Pig Slides adapted from Olston et al. (SIGMOD 2008)

Pig Query Plan



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

Pig: MapReduce Execution



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

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

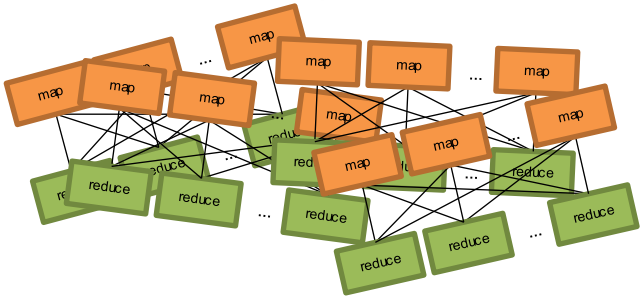
facebook YAHOO!

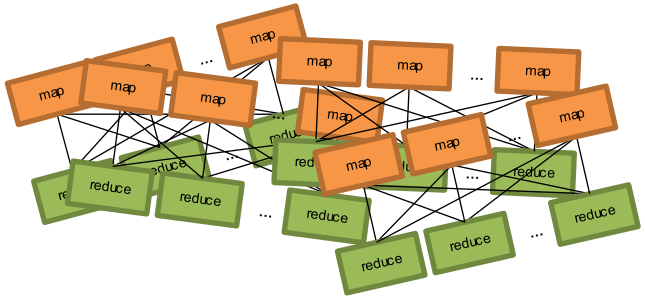
Abstractions to the rescue!

SQL

Pig Scripts

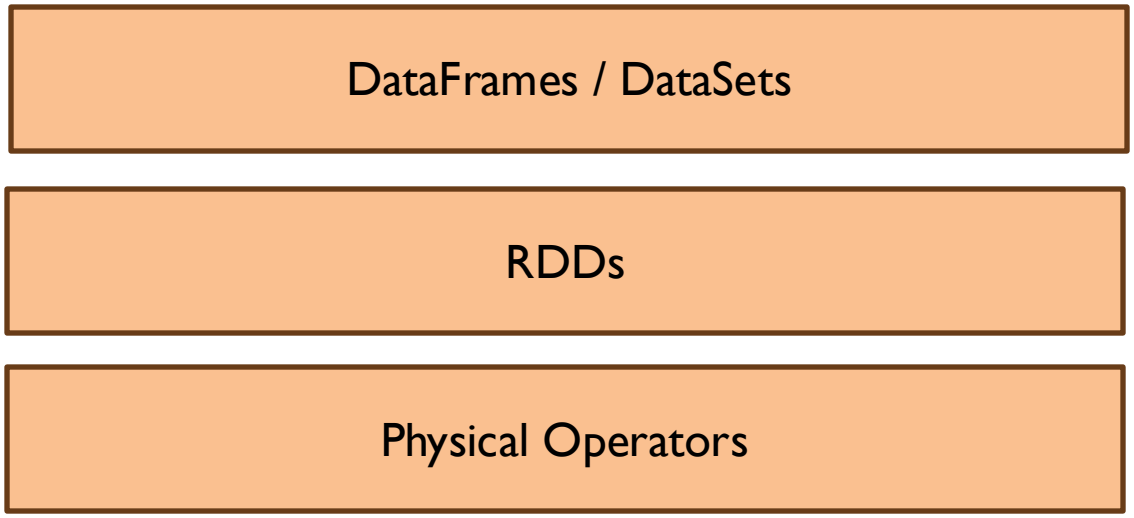
Spark provides a richer "instruction set"





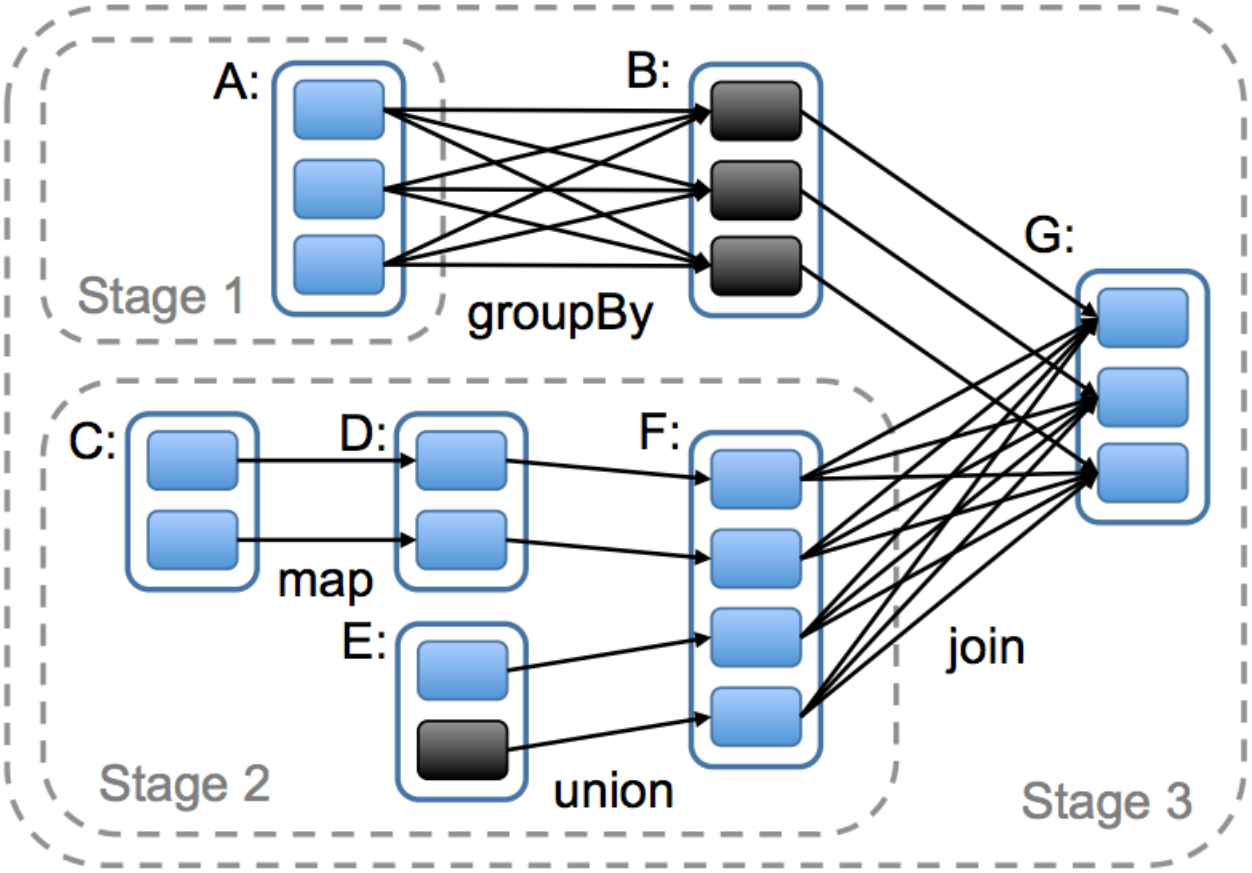
Both (still) open-source projects today!

Spark Stack

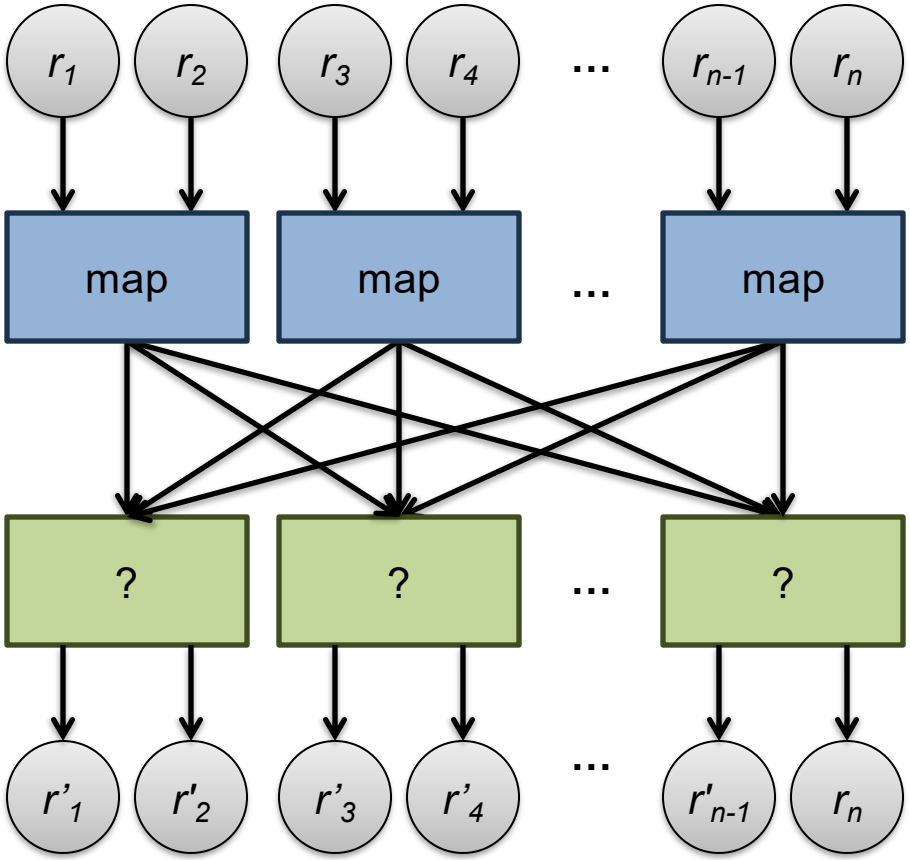


Who lives where?

Spark Execution Plan



We need communication



Next time!

Source: Wikmedia