

# Cloud Computing

## MapReduce to Spark

---

Minchen Yu  
SDS@CUHK-SZ  
Fall 2024



香港中文大學(深圳)  
The Chinese University of Hong Kong, Shenzhen



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

# What's the solution?

解决方案是什么？

Design a higher-level language      设计一种更高级的语言

Write a compiler      编写一个编译器

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

Hadoop 很棒，但它实在是太低级了！



What we really need is SQL

我们真正需要的是 SQL



What we really need is  
scripting language

我们真正需要的是脚本语言

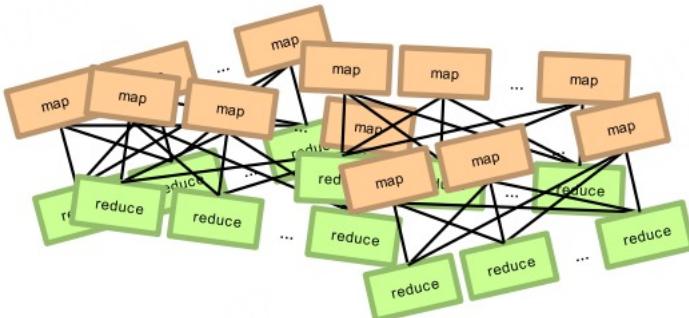




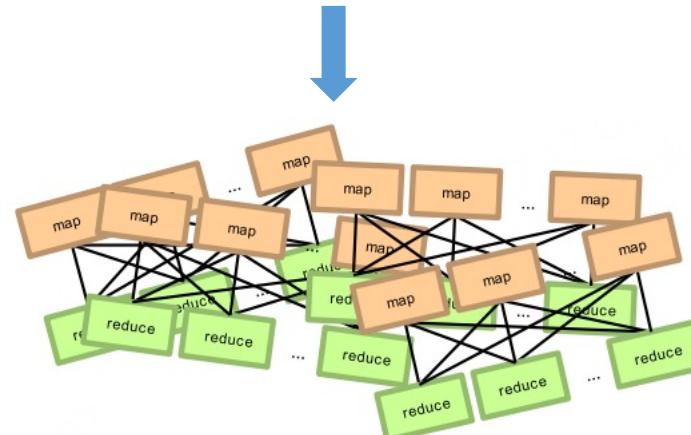
SQL



Why not just use a  
database?



Pig Scripts





信息平台与数据科学家的崛起

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.”

“在记录Facebook点击流的第一天，收集了超过400GB的数据。这些数据的加载、索引和聚合过程对Oracle数据仓库的负载极大。即使经过显著的优化，我们仍无法在24小时内聚合一天的点击流数据。”

Story for another day...

YAHOO!<sup>®</sup>



# Pig: Example

找到每个类别中访问最多的前10个页面。

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

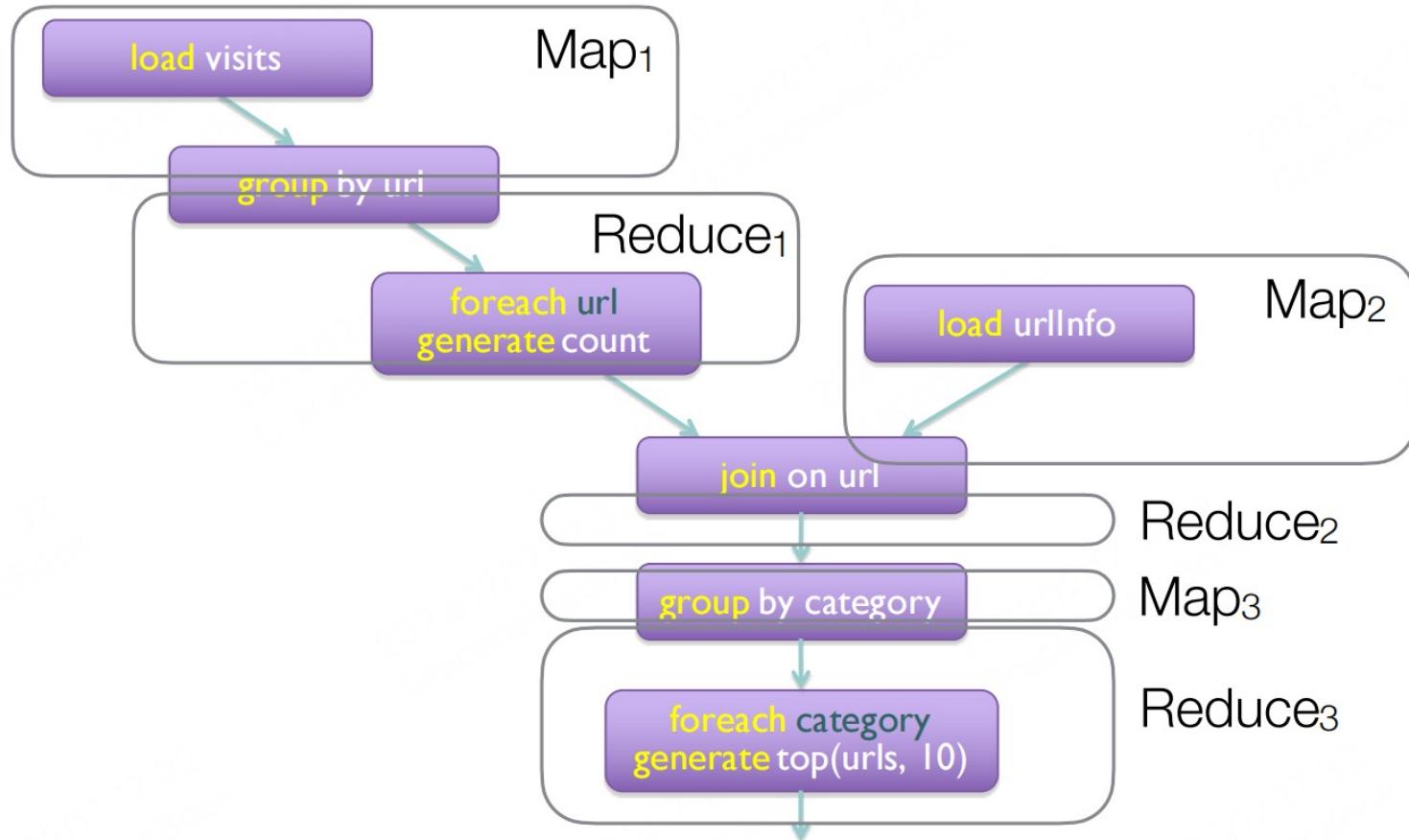
User	Category	PageRank
cnn.com	News	0.9
bbc.com	News	0.8
flickr.com	Photos	0.7
espn.com	Sports	0.9

# Pig: Example

```
visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

# Pig query plan



```

visits = load '/data/visits' as (user, url, time);
gVisits = group visits by url;
visitCounts = foreach gVisits generate url, count(visits);
urlInfo = load '/data/urlInfo' as (url, category, pRank);
visitCounts = join visitCounts by url, urlInfo by url;
gCategories = group visitCounts by category;
topUrls = foreach gCategories generate top(visitCounts,10);

```

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.lib.comparator;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.ReduceTask;
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> {
            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);
                Text outKey = new Text(key);
                // Fprepend an index to the value so we know which file
                // it came from
                Text outVal = new Text("0" + value);
                oc.collect(outKey, outVal);
            }
        }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, 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) {
                    String key = line.substring(0, firstComma);
                    Text outKey = new Text(key);
                    // Fprepend 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> {
            public void reduce(Text key,
                              Iterator<Text> iter,
                              OutputCollector<Text, Text> oc,
                              Reporter reporter) throws IOException {
                // For each value, figure out which file it's from and
                store it // accordingly.
                List<String> first = new ArrayList<String>();
                List<String> second = new ArrayList<String>();
                while (iter.hasNext()) {
                    Text t = iter.next();
                    String value = t.toString();
                    if (value.charAt(0) == '1')
                        first.add(value.substring(1));
                    else second.add(value.substring(1));
                }
            }
        }
}

```

This?

```

    reporter.setStatus("OK");
)
// Do the cross product and collect the values
for (String s1 : first) {
    for (String s2 : second) {
        String outval = key + "," + s1 + "," + s2;
        oc.collect(null, new Text(outval));
        reporter.setStatus("OK");
    }
}
}

public static class LoadJoined extends MapReduceBase
    implements Mapper<Text, Text, LongWritable> {
    public void map(
        Text key,
        Text val,
        OutputCollector<Text, LongWritable> oc,
        Reporter reporter) throws IOException {
        // Find the first comma
        String line = val.toString();
        int firstComma = line.indexOf(',');
        String value = line.substring(firstComma + 1);
        // drop the rest of the record, I don't need it anymore,
        // just pass a 1 for the combiner/reducer to sum instead.
        Text outKey = new Text(key);
        oc.collect(outKey, new LongWritable(1));
    }
}

public static class ReduceUrls extends MapReduceBase
    implements Reducer<Text, LongWritable, WritableComparable,
    Writable> {
    public void reduce(
        Text key,
        Iterator<LongWritable> iter,
        OutputCollector<WritableComparable, Writable> oc,
        Reporter reporter) throws IOException {
        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
        while (count < 100 && iter.hasNext()) {
            oc.collect(key, iter.next());
            count++;
        }
    }
}

public static void main(String[] args) throws IOException {
    JobConf ip = new JobConf(MRExample.class);
    ip.setJobName("Load Pages");
    ip.setInputFormat(TextInputFormat.class);
    Path[] user Gates Pages = new Path[1];
    user Gates Pages[0] = new Path("/user/gates/pages");
    ip.setNumReduceTasks(0);
    Job loadPages = new Job(ip);
    JobConf ltu = new JobConf(MRExample.class);
    ltu.setJobName("Load and Filter Users");
    ltu.setInputFormat(TextInputFormat.class);
    ltu.setOutputFormat(TextOutputFormat.class);
    ltu.setMapperClass(LoadAndFilterUsers.class);
    ltu.setReducerClass(LoadAndFilterUsers.class);
    FileInputFormat.addInputPath(ltu, new Path("/user/gates/urls"));
    FileOutputFormat.setOutputPath(ltu,
        new Path("/user/gates/tmp/indexed_pages"));
    ip.setNumReduceTasks(0);
    Job loadUsers = new Job(ltu);
    JobConf join = new JobConf(MRExample.class);
    join.setJobName("Join Users and Pages");
    join.setInputFormat(TextInputFormat.class);
    join.setOutputValueClass(Text.class);
    join.setOutputKeyClass(Text.class);
    join.setMapperClass(Join.class);
    join.setCombinerClass(Join.class);
    FileInputFormat.addInputPath(join, new Path("/user/gates/tmp/indexed_pages"));
    FileInputFormat.addInputPath(join, new Path("/user/gates/tmp/filterred_users"));
    FileOutputFormat.setOutputPath(join, new Path("/user/gates/tmp/joined"));
    join.addAsDependingJob(loadPages);
    join.addAsDependingJob(loadUsers);
    JobConf group = new JobConf(MRExample.class);
    group.setJobName("Group URLs");
    group.setInputFormat(KeyValueTextInputFormat.class);
    group.setOutputValueClass(Text.class);
    group.setOutputKeyClass(Text.class);
    group.setOutputFormat(TextOutputFormat.class);
    group.setMapperClass(LoadJoined.class);
    group.setCombinerClass(ReduceUrls.class);
    group.setReducerClass(LimitClicks.class);
    FileInputFormat.addInputPath(group, new Path("/user/gates/tmp/joined"));
    FileInputFormat.setOutputPath(group, new Path("/user/gates/tmp/grouped"));
    group.setNumReduceTasks(50);
    Job groupJob = new Job(group);
    groupJob.addAsDependingJob(join);
    JobConf top100 = new JobConf(MRExample.class);
    top100.setJobName("Top 100 sites");
    top100.setInputFormat(SequenceFileInputFormat.class);
    top100.setOutputFormat(SequenceFileOutputFormat.class);
    top100.setOutputValueClass(Text.class);
    top100.setMapperClass(LoadClicks.class);
    top100.setCombinerClass(ReduceUrls.class);
    top100.setReducerClass(LimitClicks.class);
    FileInputFormat.addInputPath(top100, new Path("/user/gates/tmp/grouped"));
    Path[] top100sitesforusers = new Path[1];
    top100sitesforusers[0] = new Path("top100sitesforusers");
    top100.setNumReduceTasks(1);
    Job limit = new Job(top100);
    limit.addAsDependingJob(groupJob);
    JobControl jc = new JobControl("Find top
100 sites for users
18 to 25");
    jc.addJob(loadPages);
    jc.addJob(loadUsers);
    jc.addJob(join);
    jc.addJob(groupJob);
    jc.addJob(limit);
    jc.run();
}

```

# Programming at a high level

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

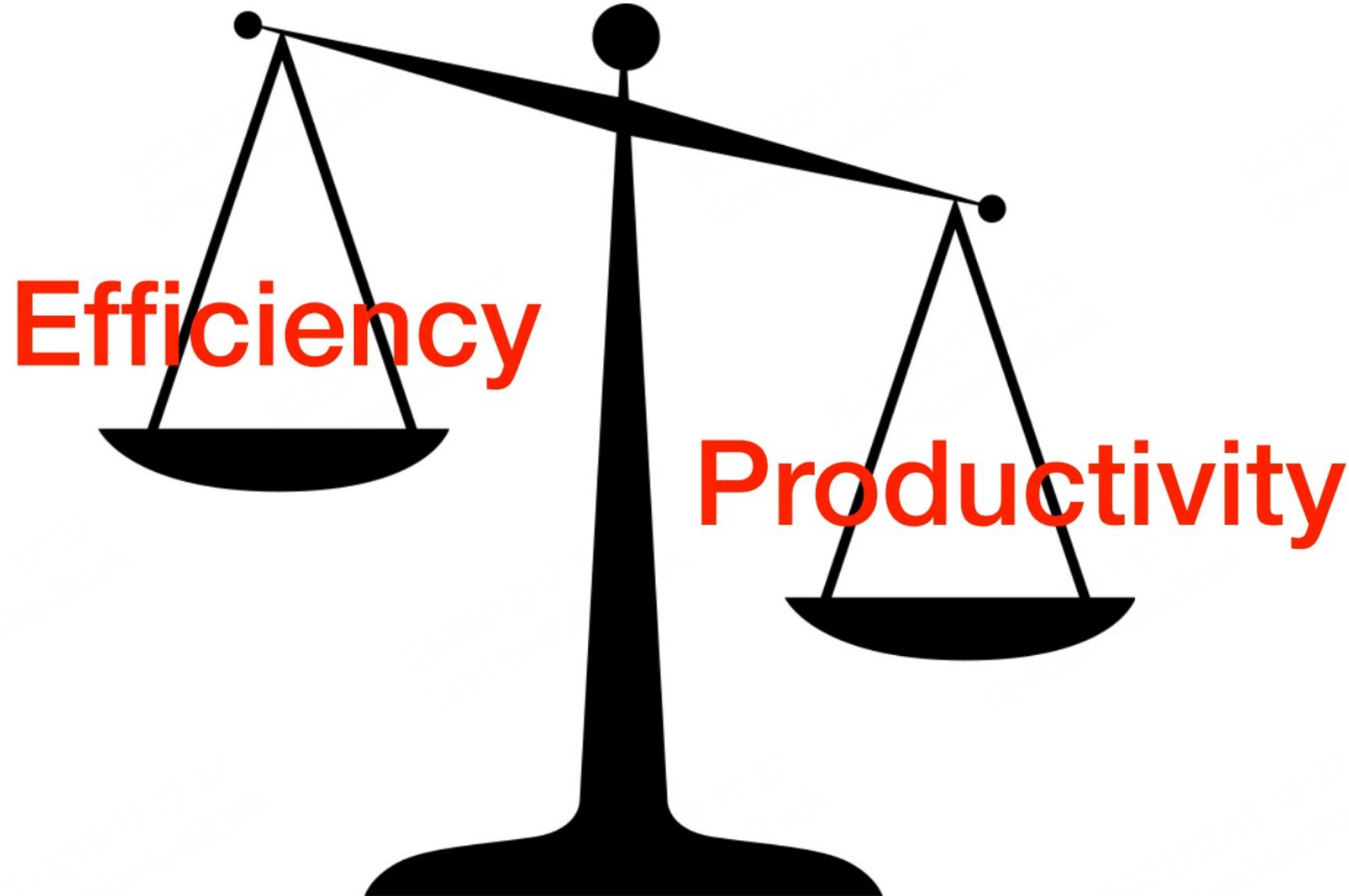
指定“要做什么”，系统会自己找出“怎么做”

# But isn't Pig slower?

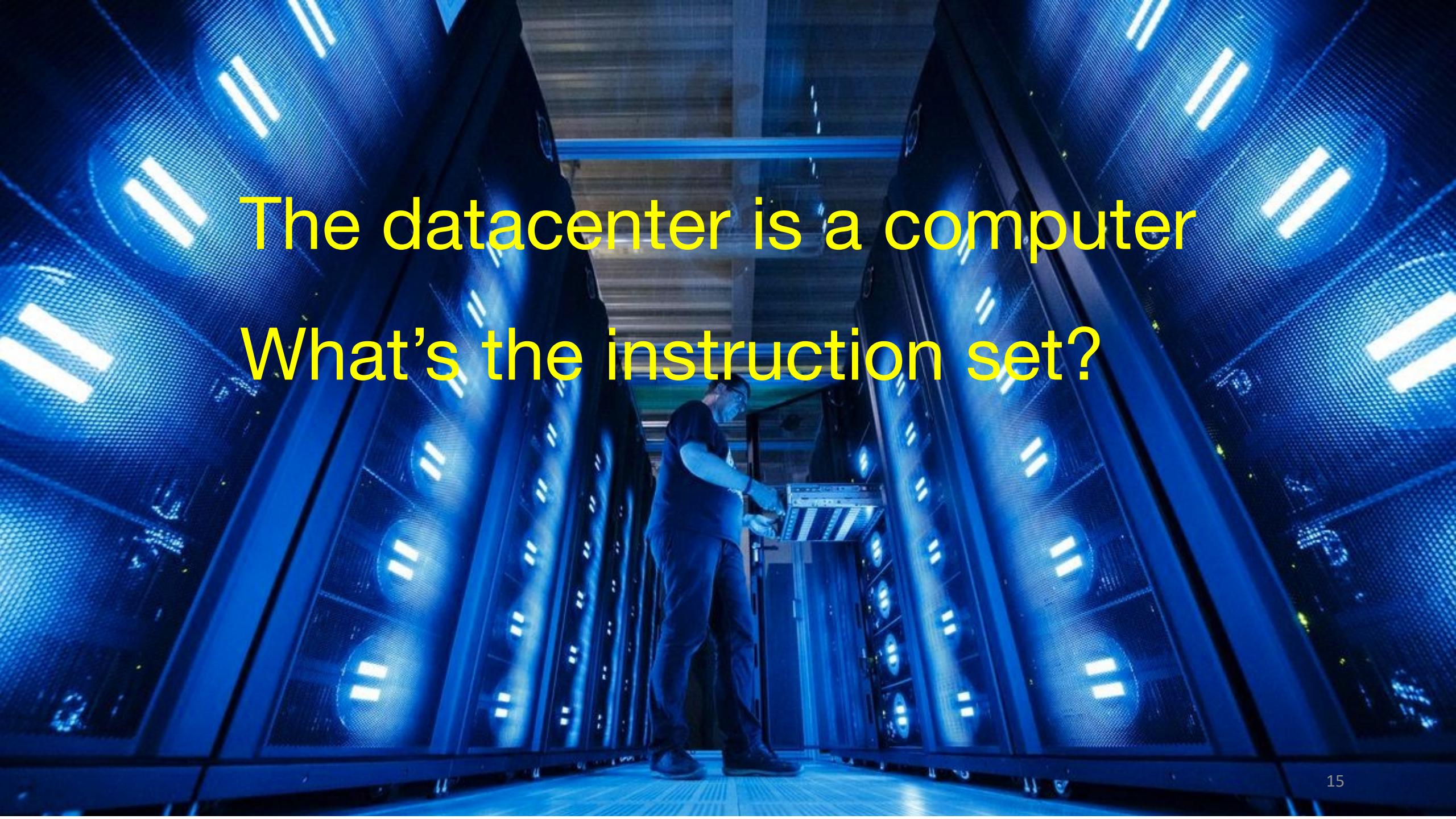
Sure, but C can be slower than assembly too...

但是 Pig 不是更慢吗 ?  
当然 , 但是 C 语言有时也比汇编慢 ...





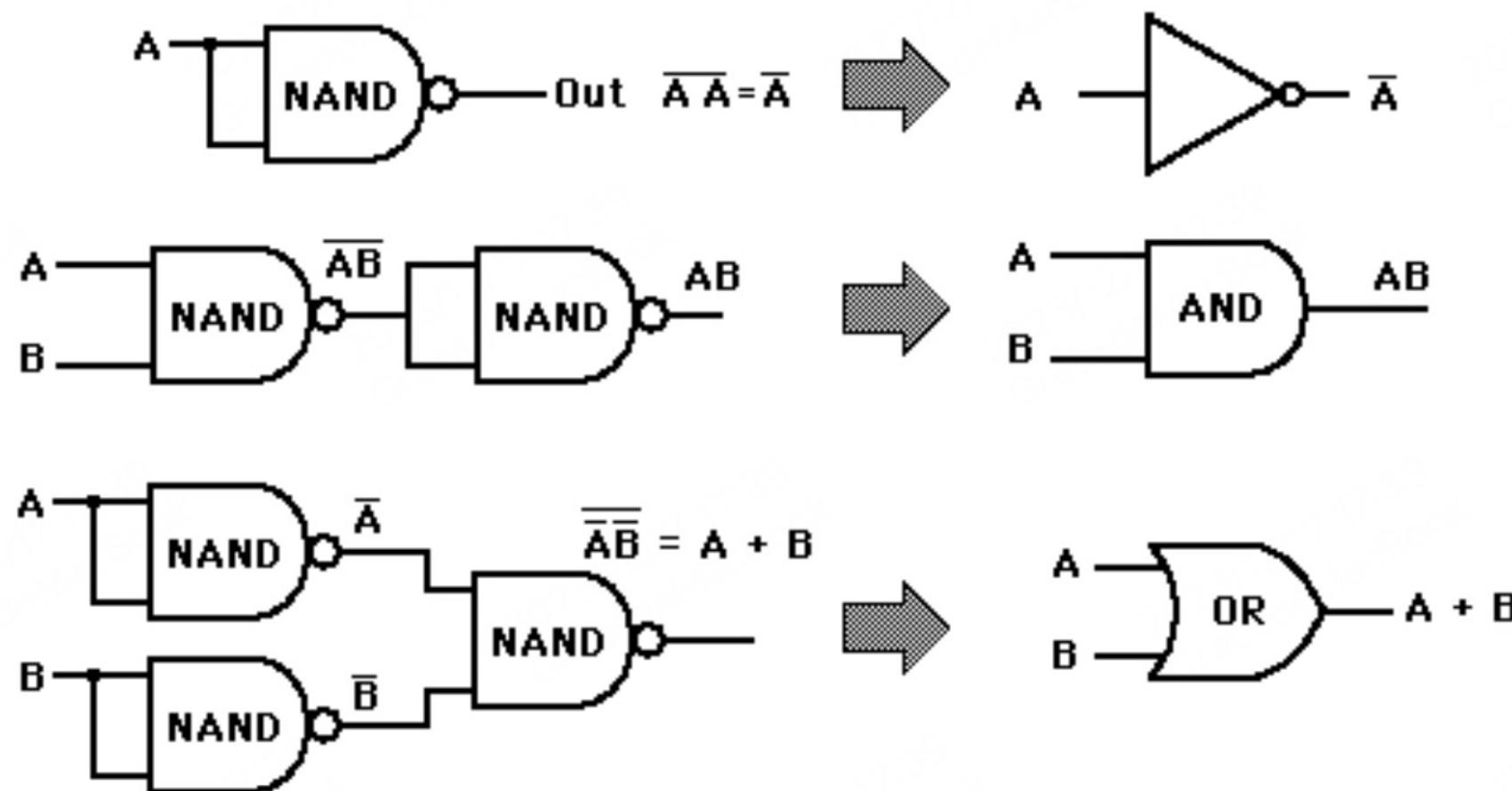
数据中心就是一台计算机  
那么它的指令集是什么？

A photograph of a datacenter aisle. A person in a dark t-shirt and pants is standing in the center, facing a row of server racks. The racks are illuminated from within, casting a blue glow. The ceiling has recessed lighting, and the floor is a polished blue-tinted surface.

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

# Analogy: NAND Gates are universal

类比：NAND门是通用的



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

让我们从头设计一种数据处理语言！  
为什么 MapReduce 是这样的？

Why is MapReduce the way it is?

# Data-parallel dataflow languages

数据并行的数据流语言

We have a collection of **records** 我们有一组记录  
希望应用一系列转换来计算结果

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

Assumptions

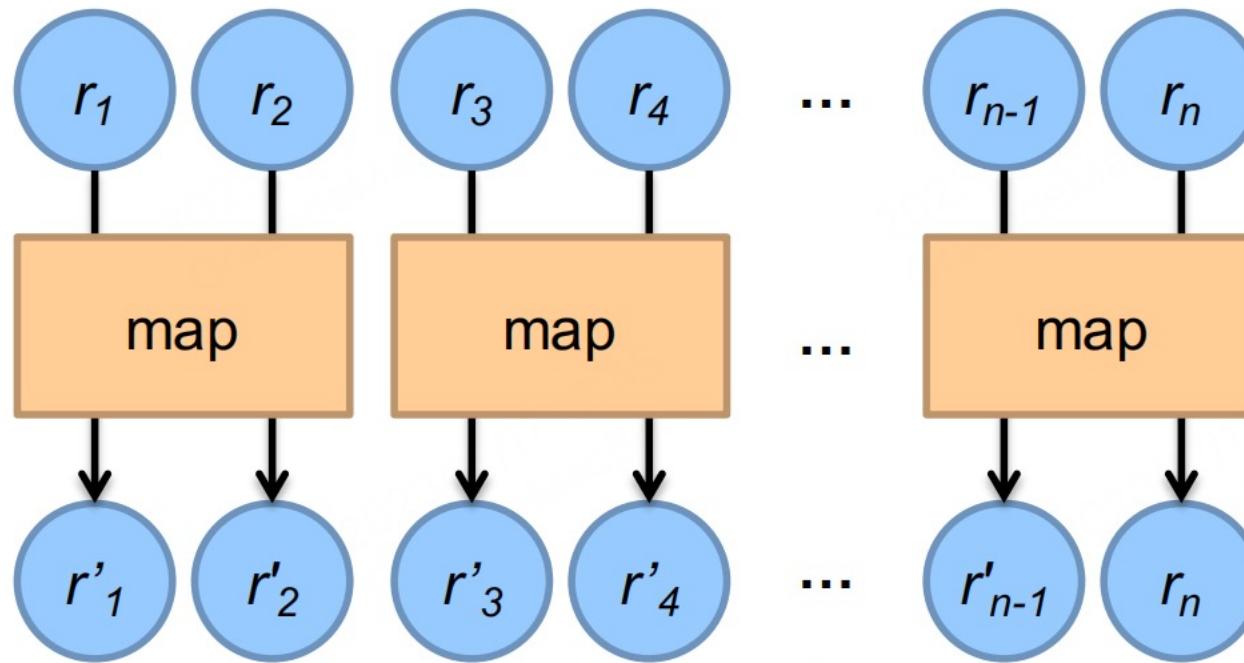
假设：  
静态集合记录（不一定是键值对）

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

# We need per-record processing

(note: not necessarily key-value pairs) (注意：不一定是键值对)

我们需要逐条记录处理



Easy to parallelize maps

轻松并行化的映射  
记录到“映射器”的分配是实现细节

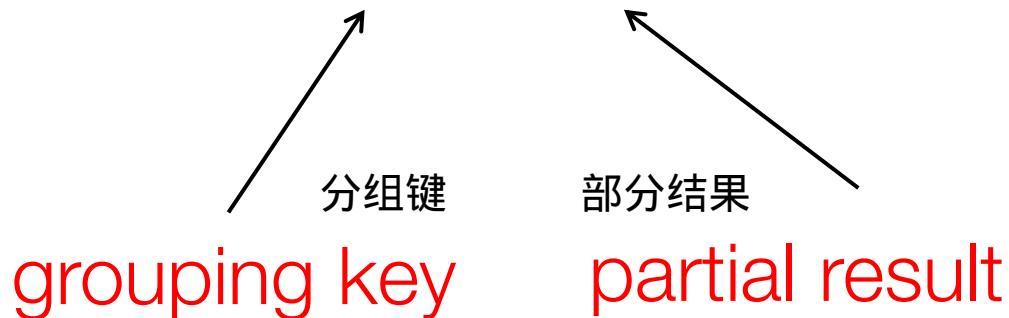
Record to “mapper” assignment is an implementation detail

# Map alone isn't enough

单靠 Map 不够

We need a way to group partial results 我们需要一种方法来对部分结果进行分组

- intermediate **(key, value)** pairs 中间的（键，值）对

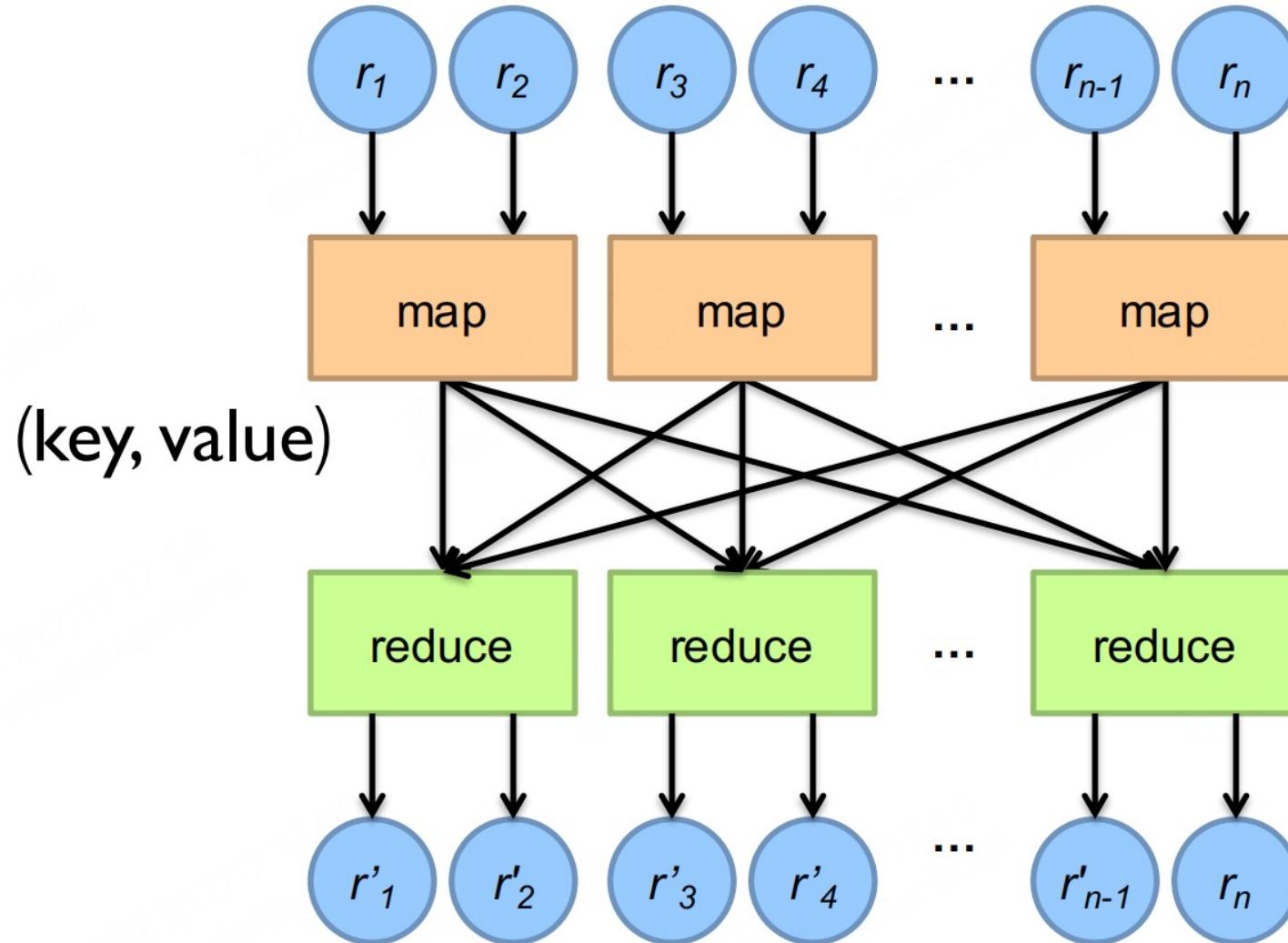


For each key, we can apply some computation

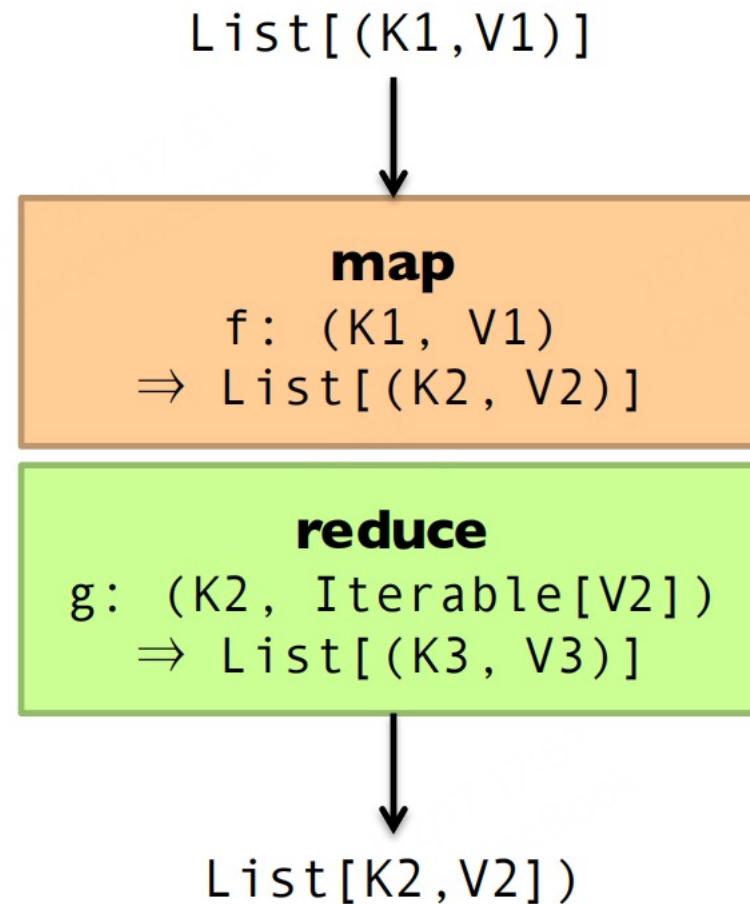
- e.g., aggregation, counting 例如，聚合，计数

对每个键，我们可以应用一些计算

# MapReduce

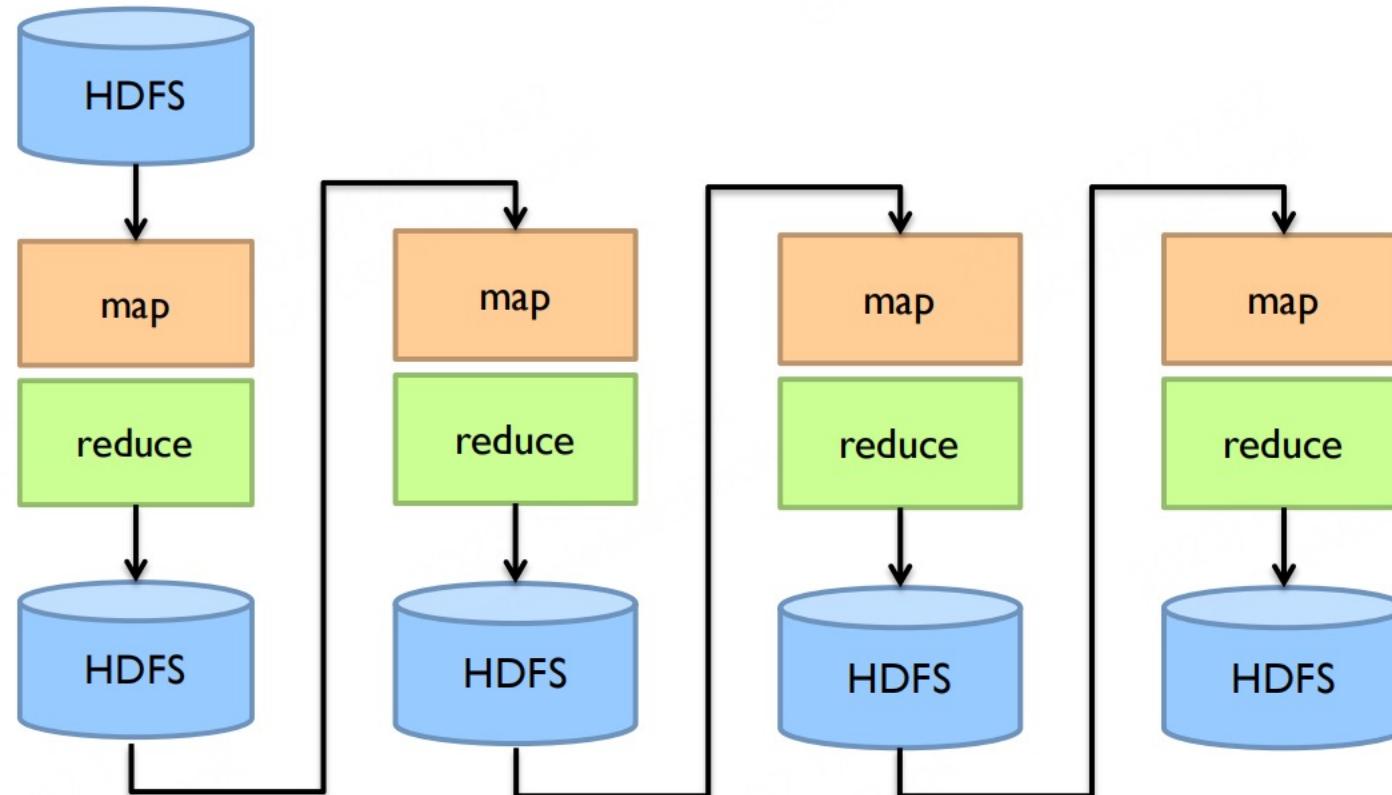


# MapReduce



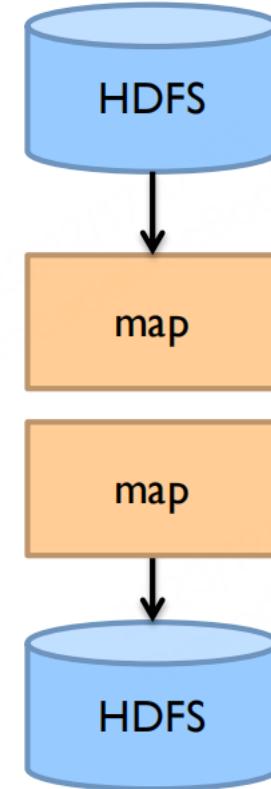
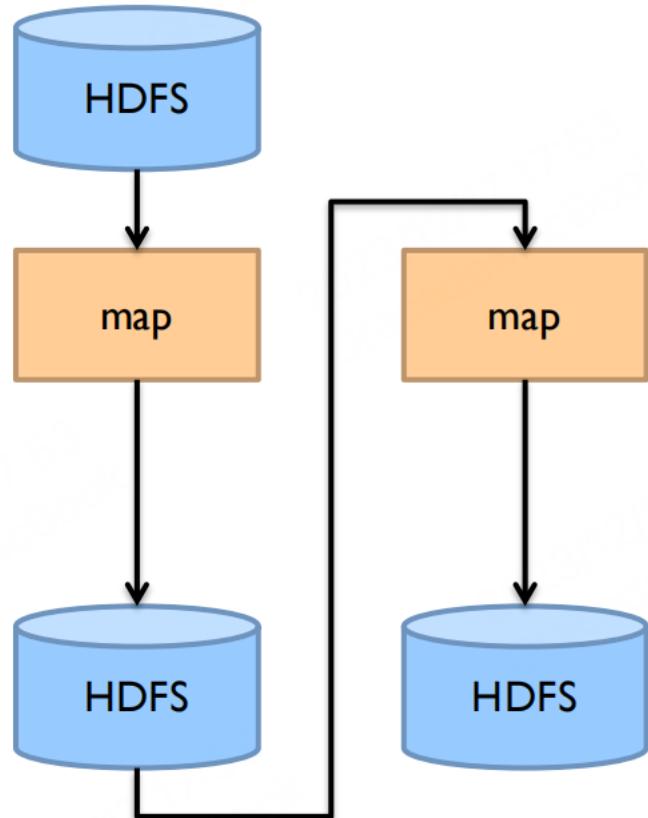
# MapReduce workflows

MapReduce 工作流

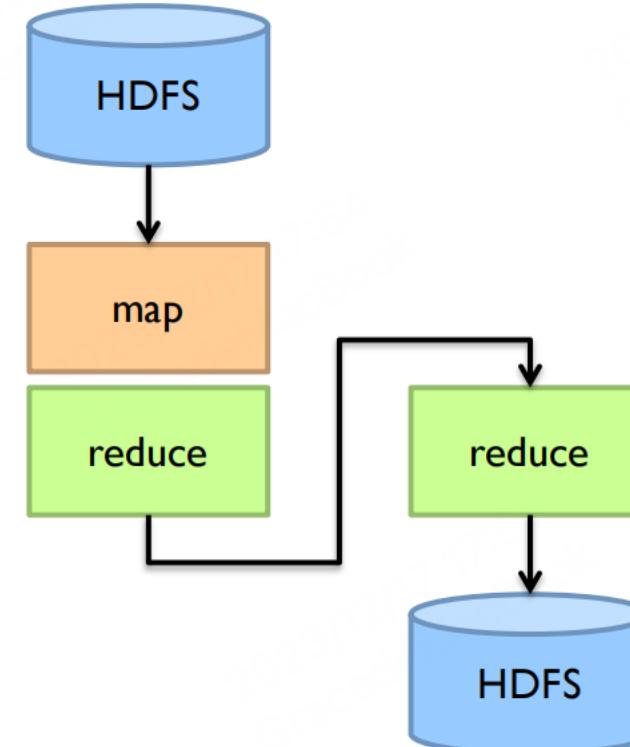
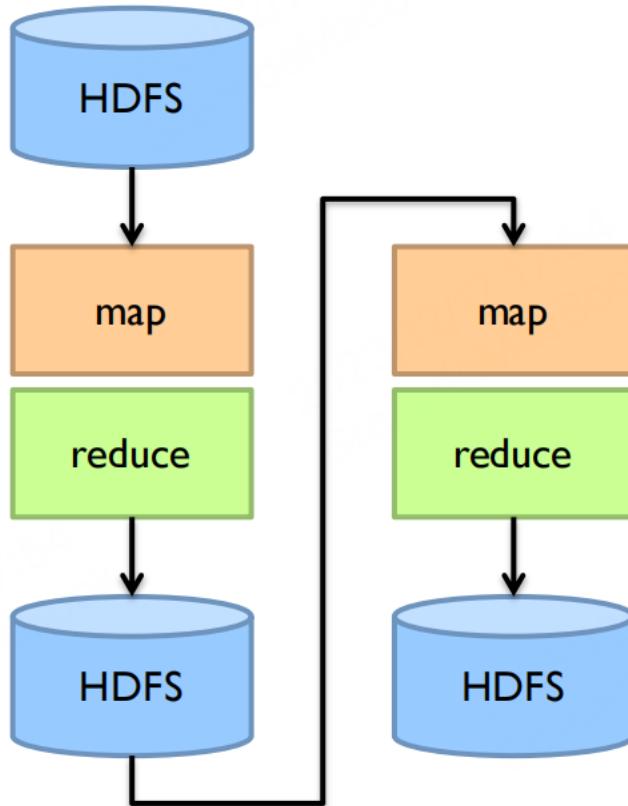


What's wrong?

# Want MM?



# Want MRR?



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

在 MapReduce 中，跨作业共享数据的唯一方法是稳定存储

Disk I/O is waaaaaaay too slow!

磁盘I/O实在是太慢了！

# Data-parallel dataflow languages

数据并行的数据流语言

We have a collection of **records** 我们有一组记录。 希望应用一系列转换来计算结果

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

What are the operators? 操作符是什么 ?

MapReduce ?

- MapReduce?





Where the hype is! 这里有炒作的地方！回答“MapReduce 之后是什么？”

- answer to “What’s beyond MapReduce?”

Brief history:

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

简史：

2009年在加州大学伯克利AMPLab开发

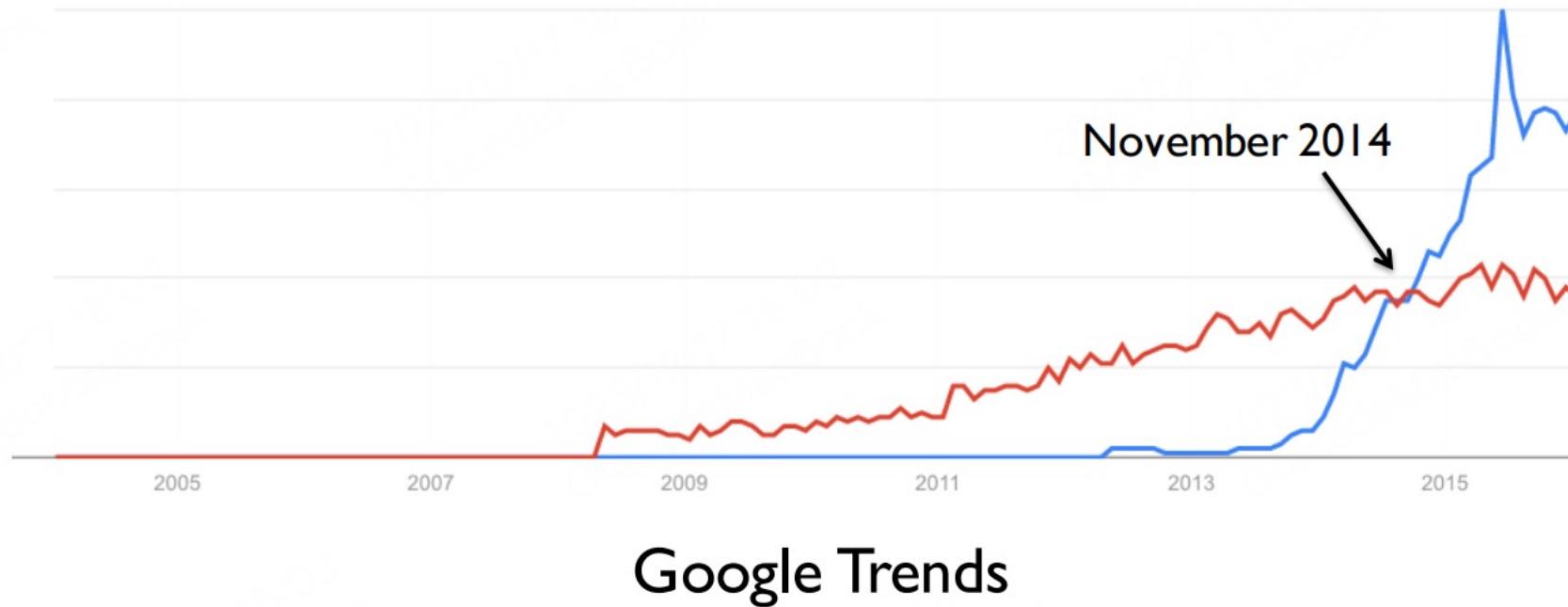
2010年开源

2014年2月成为Apache顶级项目

Databricks提供商业支持



vs.



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

# Motivation

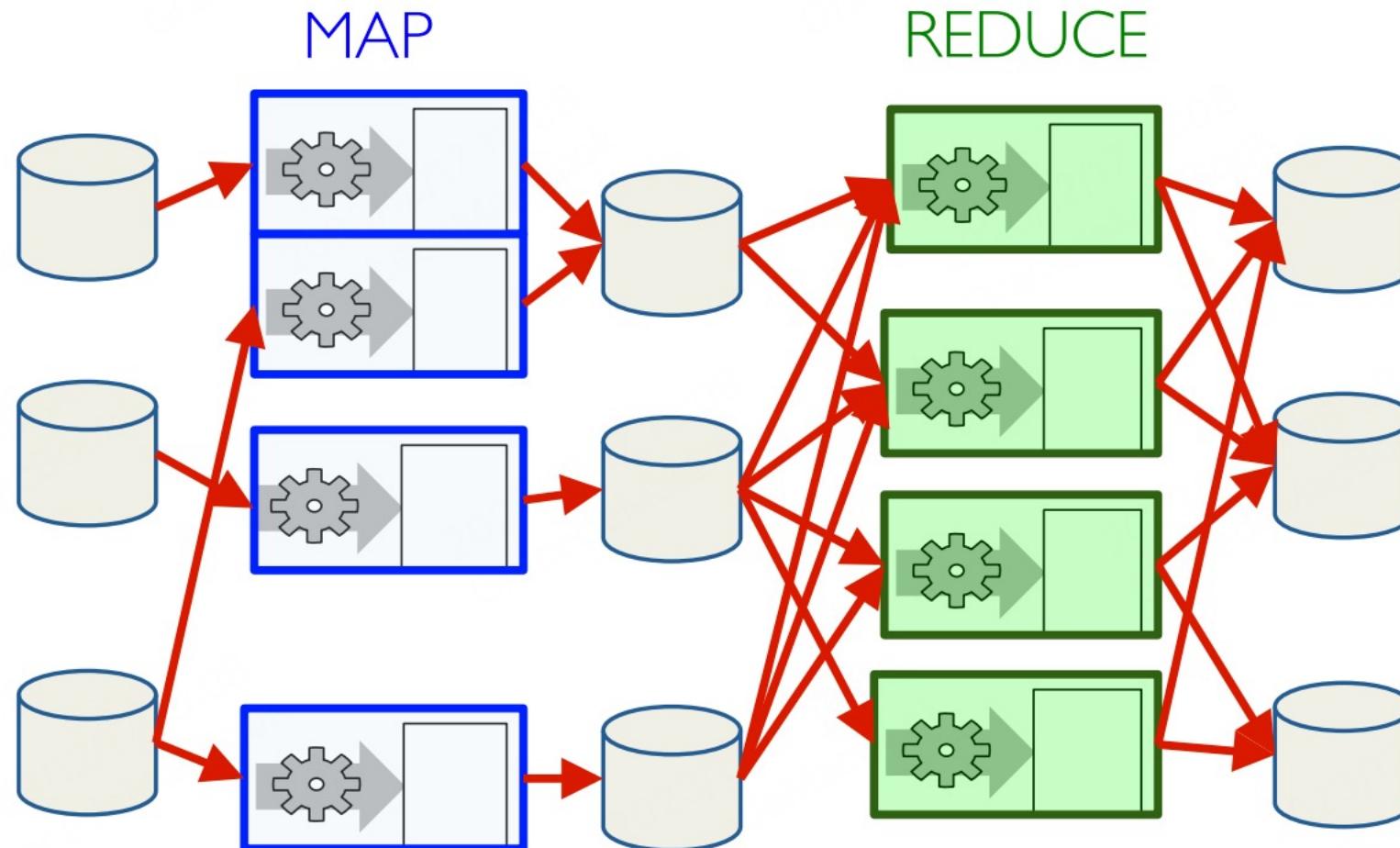
MapReduce is a restrictive programming model for batch processing jobs  
MapReduce 对批处理任务来说是一个限制性编程模型

But we want more... 但是我们想要更多...

- More **complex**, multi-stage data processing, e.g., iterative machine learning  
更复杂的多阶段数据处理，例如迭代机器学习
- Low latency, **interactive** ad-hoc data analytics, e.g., streaming processing  
低延迟、交互式的即席数据分析，例如流处理

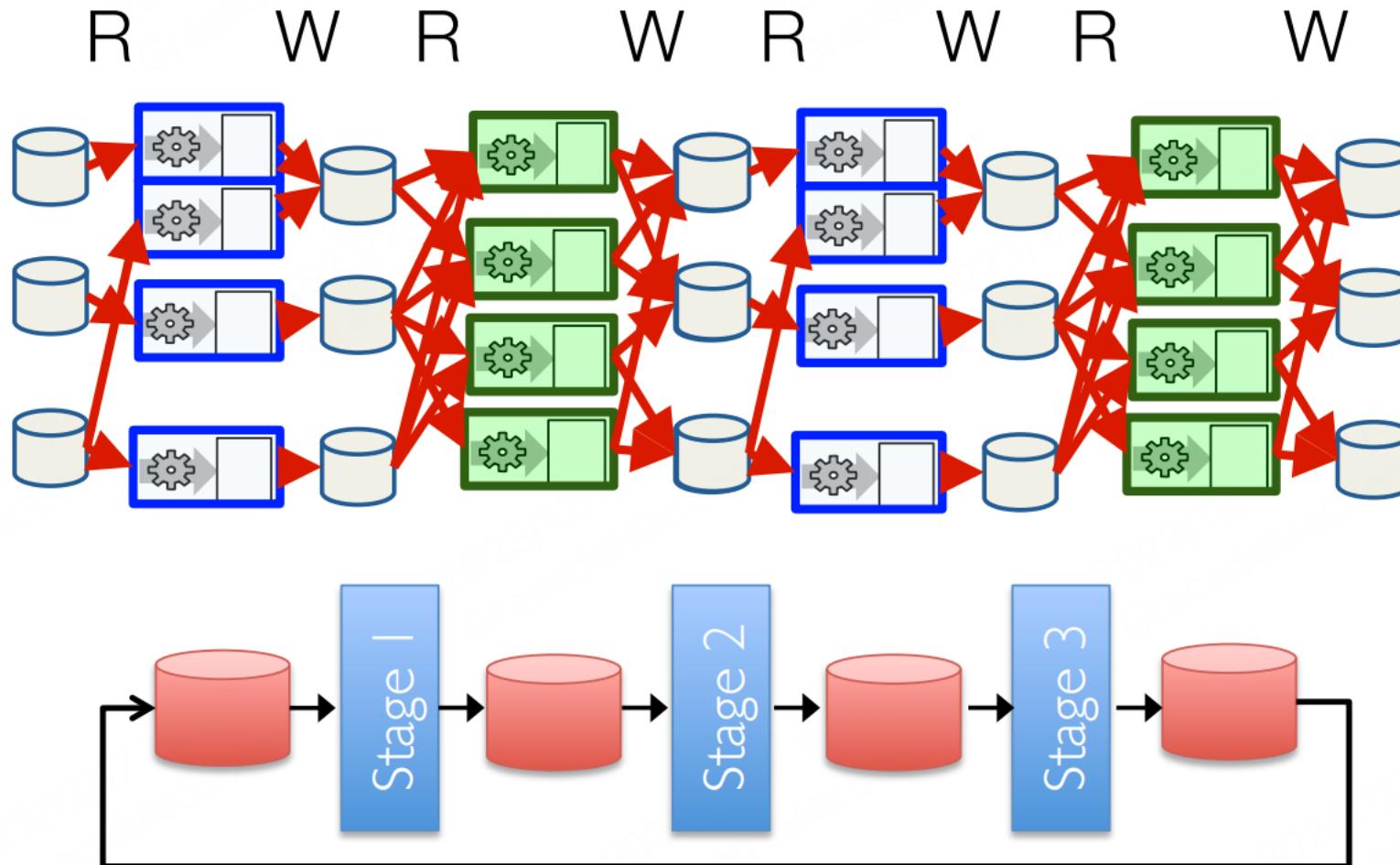
每个阶段都通过稳定存储传递！

Each stage passes through the stable storage!

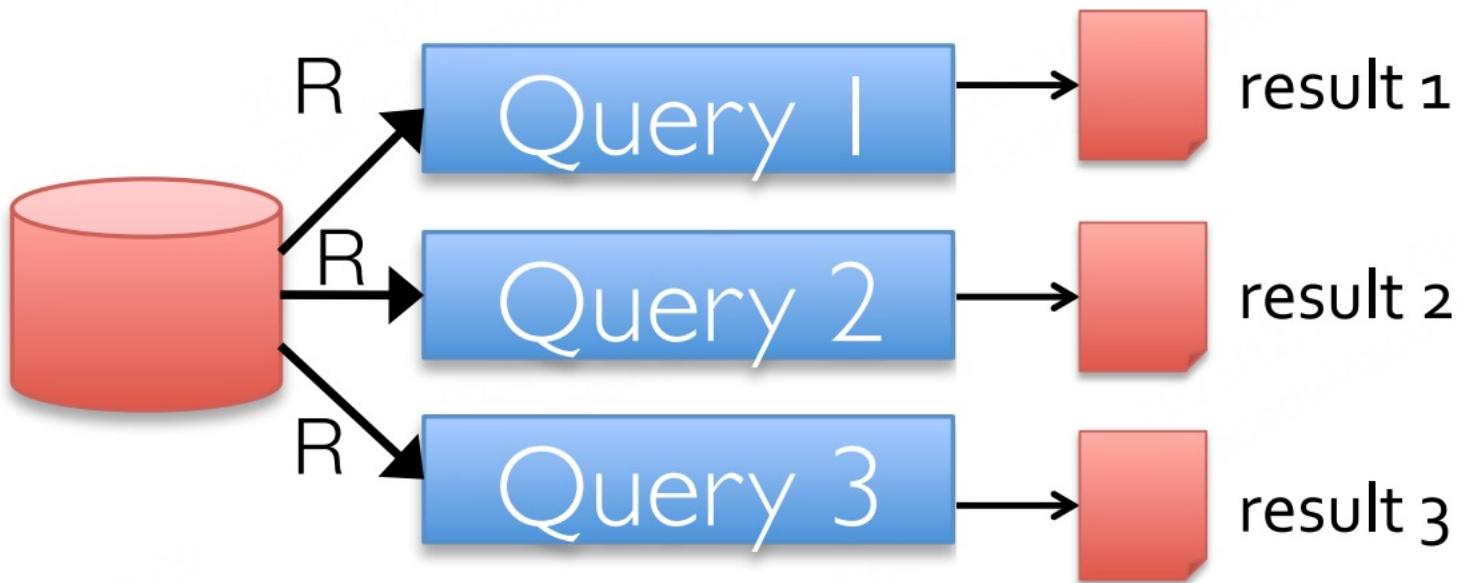


# Iterative computations

迭代计算

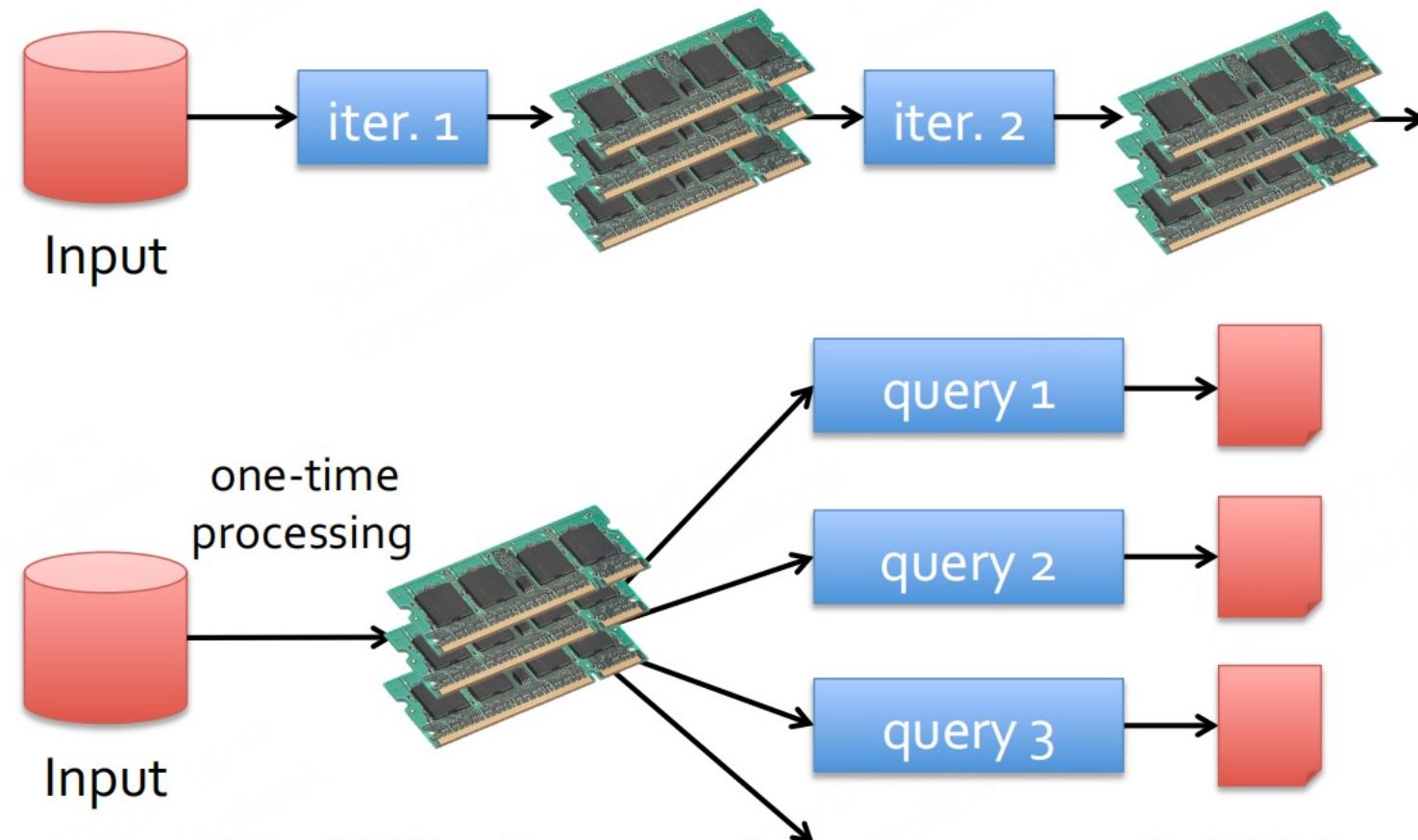


# Iterative queries



直觉：内存数据共享  
比网络/磁盘快10-100倍

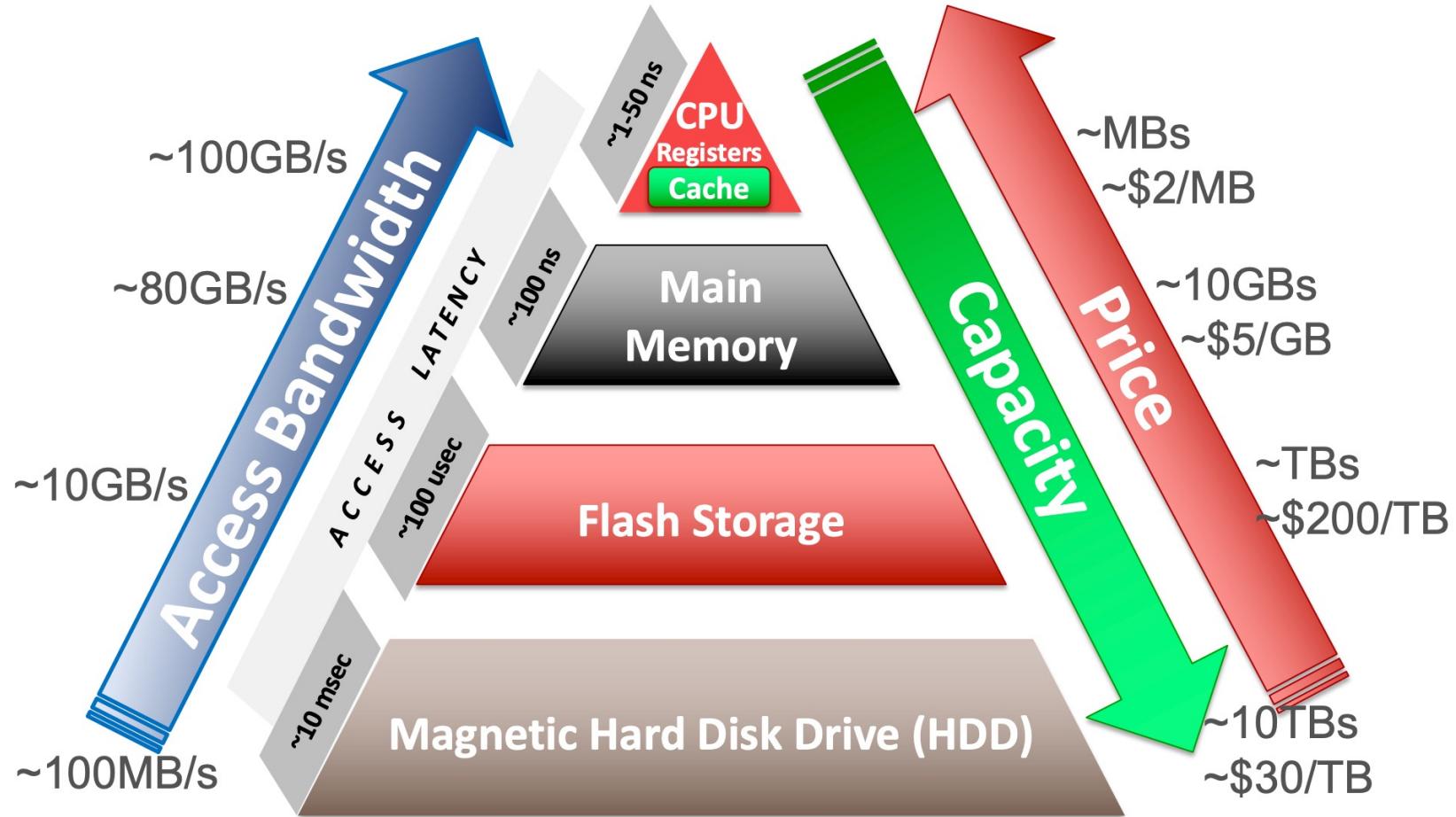
# Intuition: In-memory data sharing



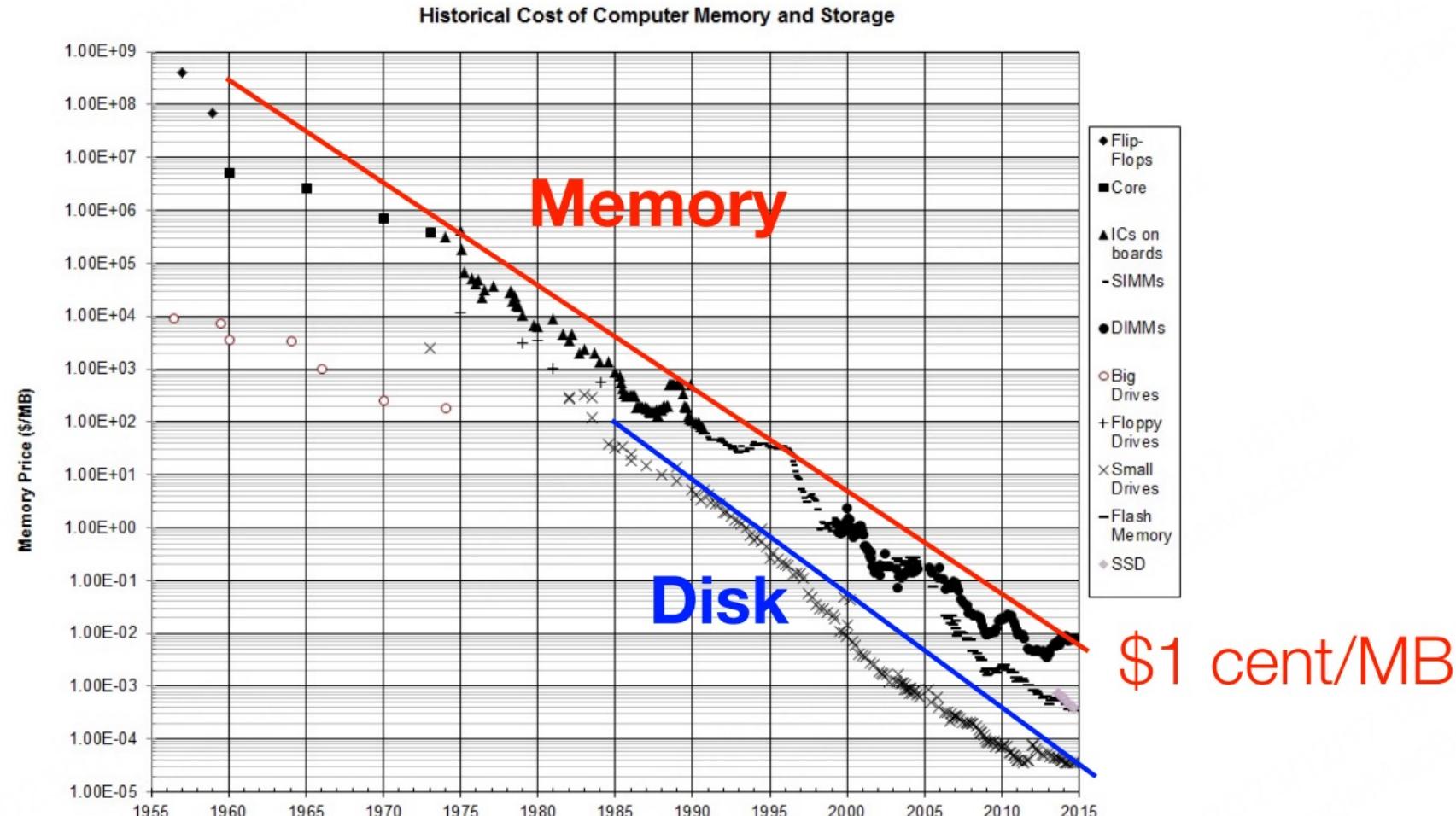
10-100x faster than network/disk

# Memory-storage hierarchy

内存存储层次结构



# Cost of Memory

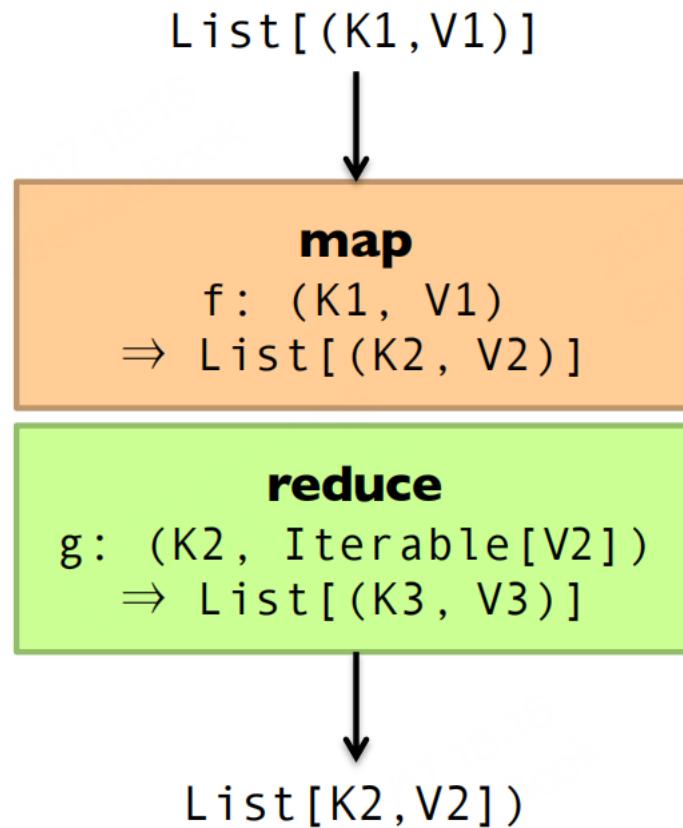


Intermediate results are stored in-memory  
as **Resilient Distributed Datasets (RDDs)**

中间结果存储在内存中作为弹性分布式数据集 ( RDD )

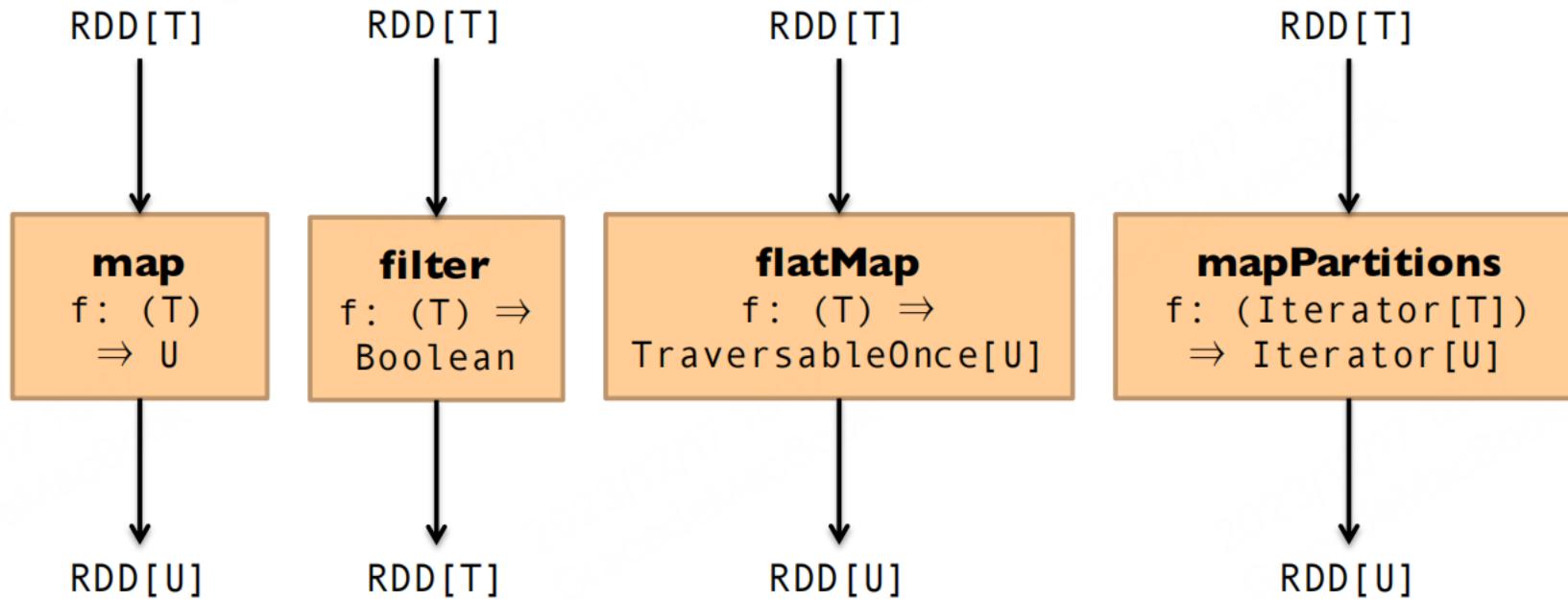
Much more next lecture...

# MapReduce



# Map-like operations

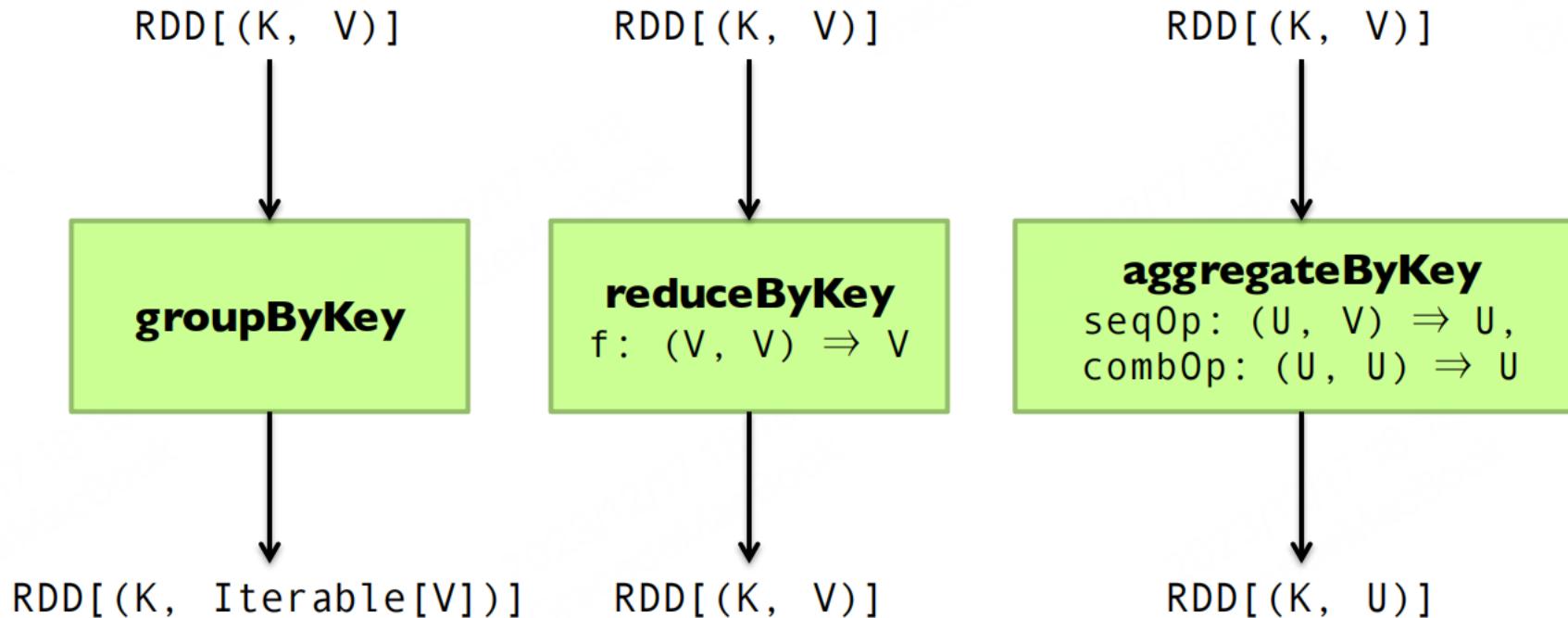
类似Map的操作



Not meant to be exhaustive

# Reduce-like operations

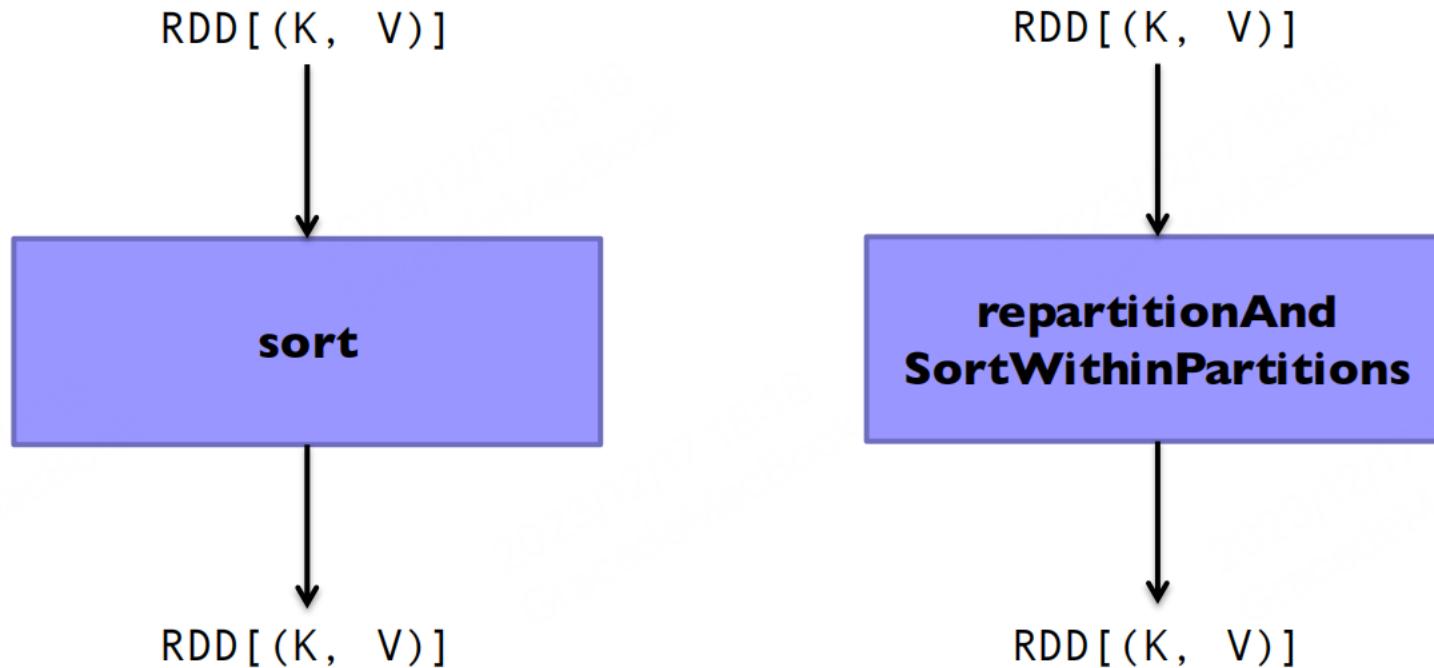
类似Reduce的操作



Not meant to be exhaustive

# Sort operations

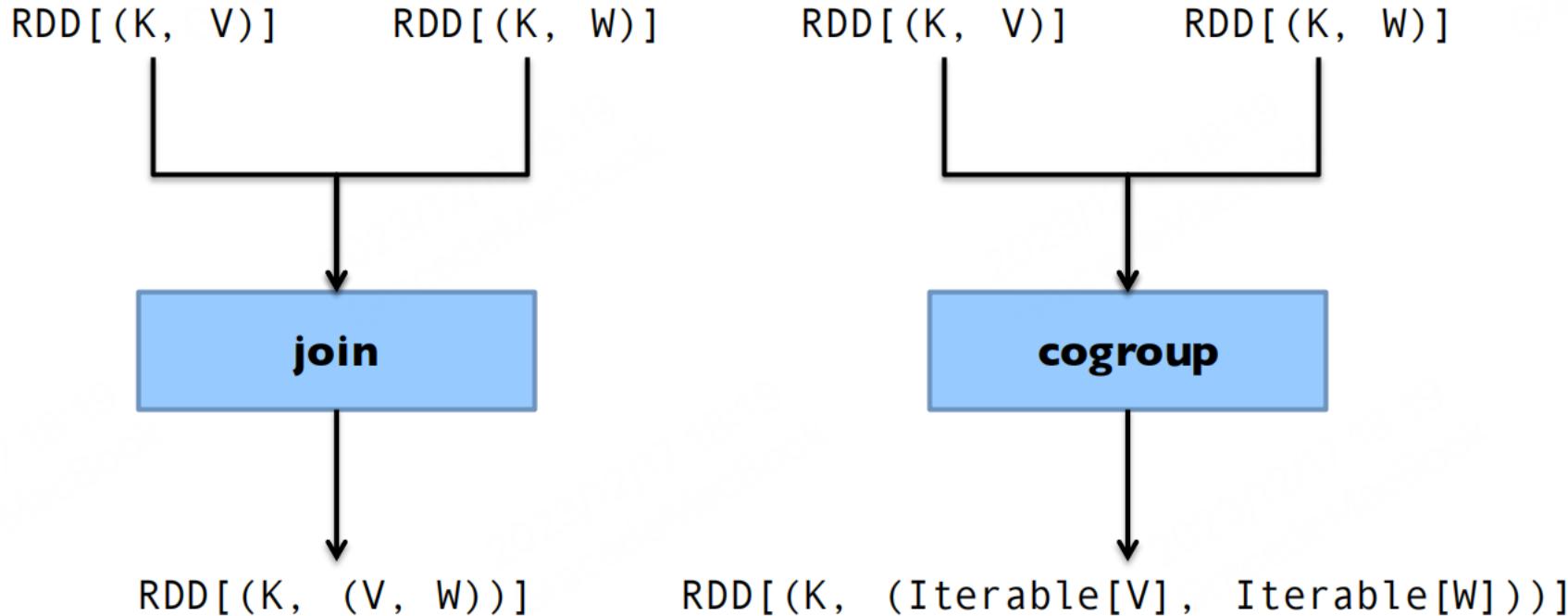
排序操作



Not meant to be exhaustive

# Join-like operations

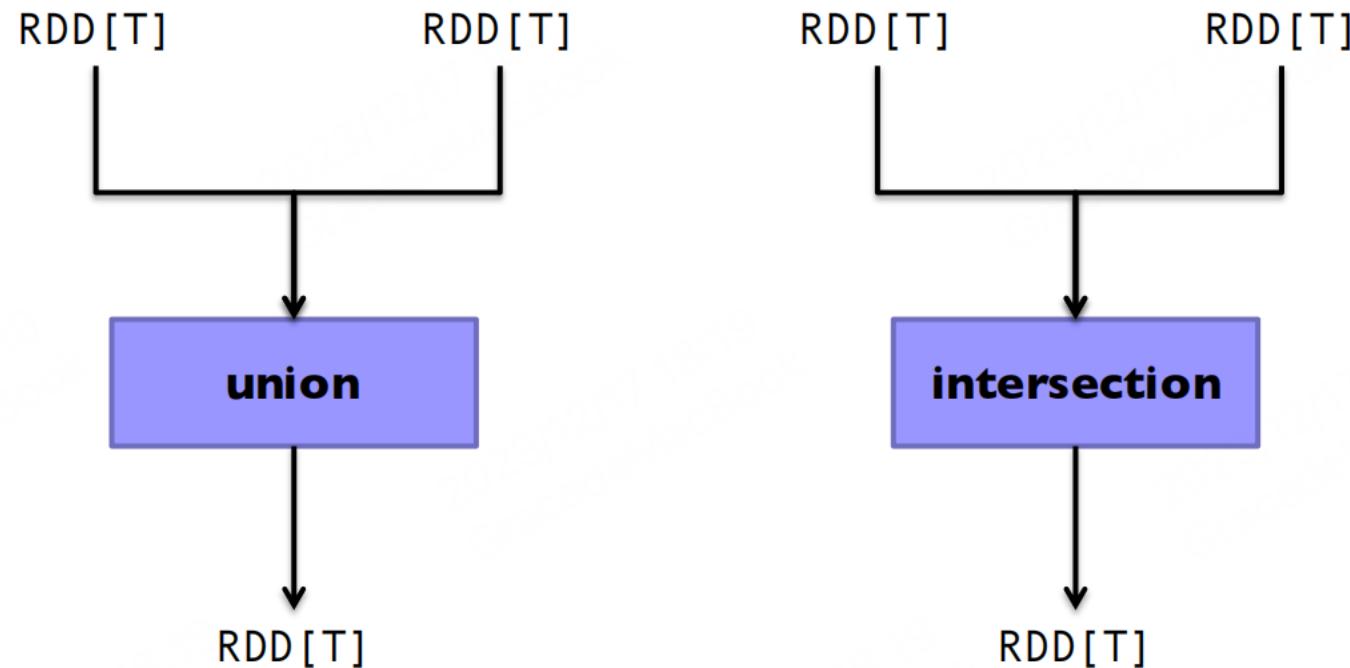
类似Join的操作



Not meant to be exhaustive

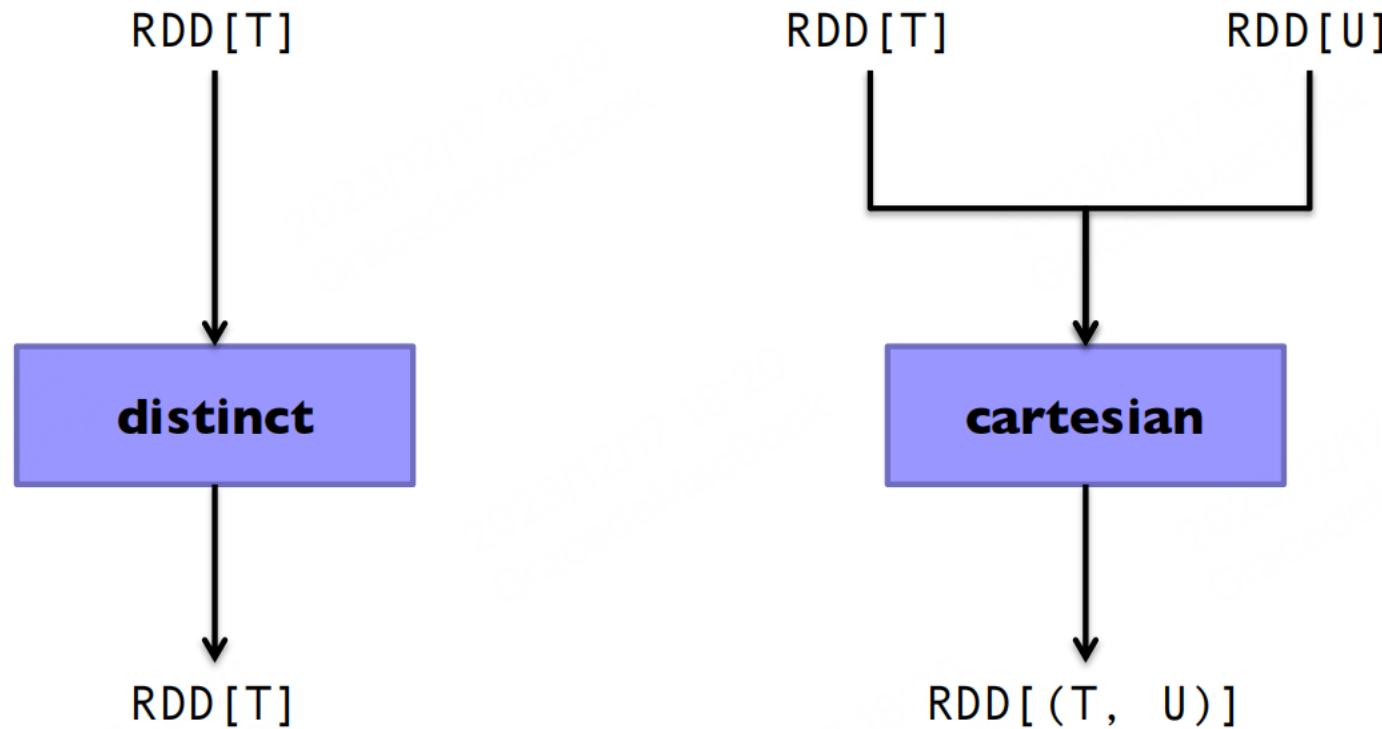
# Set-ish operations

类似集合的操作



Not meant to be exhaustive

# Set-ish operations



Not meant to be exhaustive

# Spark WordCount: 3-liner

Spark WordCount : 三行代码

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

This?

```
private static class MyMapper  
    extends Mapper<LongWritable, Text, Text, IntWritable> {  
  
    private final static IntWritable ONE = new IntWritable(1);  
    private final static Text WORD = new Text();  
  
    @Override  
    public void map(LongWritable key, Text value, Context context)  
        throws IOException, InterruptedException {  
        String line = ((Text) value).toString();  
        String[] words = line.trim().split("\\s+");  
        for (String w: words) {  
            WORD.set(w);  
            context.write(WORD, ONE);  
        }  
    }  
  
    private static class MyReducer  
        extends Reducer<Text, IntWritable, Text, IntWritable> {  
  
        private final static IntWritable SUM = new IntWritable();  
  
        @Override  
        public void reduce(Text key, Iterable<IntWritable> values,  
            Context context) throws IOException, InterruptedException {  
            int sum = 0;  
            for (IntWritable v: values) {  
                sum += v.get();  
            }  
            SUM.set(sum);  
            context.write(key, SUM);  
        }  
  
        Configuration conf = getConf();  
        Job job = Job.getInstance(conf);  
        job.setJobName("Word Count");  
        job.setJarByClass(WordCount.class);  
        job.setNumReduceTasks(reduceTasks);  
        FileInputFormat.setInputPaths(job, new Path(inputPath));  
        FileOutputFormat.setOutputPath(job, new Path(outputPath));  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
        job.setMapperClass(WordCountMapper.class);  
        job.setCombinerClass(WordCountReducer.class);  
        job.setReducerClass(WordCountReducer.class);  
        System.exit(job.waitForCompletion(true) ? 0: 1);  
    }  
}
```

Or this?

# Credits

- Some slides are adapted from course slides of COMP 4651 in HKUST
- Some slides adapted from course slides of DS5110 in UVA