



UC Berkeley  
Teaching Professor  
Dan Garcia

# CS61C

## Great Ideas in Computer Architecture (a.k.a. Machine Structures)



UC Berkeley  
Lecturer  
Justin Yokota

## MapReduce & Spark

# Request-Level and Data-Level Parallelism

# New-School Machine Structures

## Software

### Parallel Requests

Assigned to computer  
e.g., Search "Cats"

### Parallel Threads

Assigned to core e.g., Lookup, Ads

### Parallel Instructions

>1 instruction @ one time  
e.g., 5 pipelined instructions

### Parallel Data

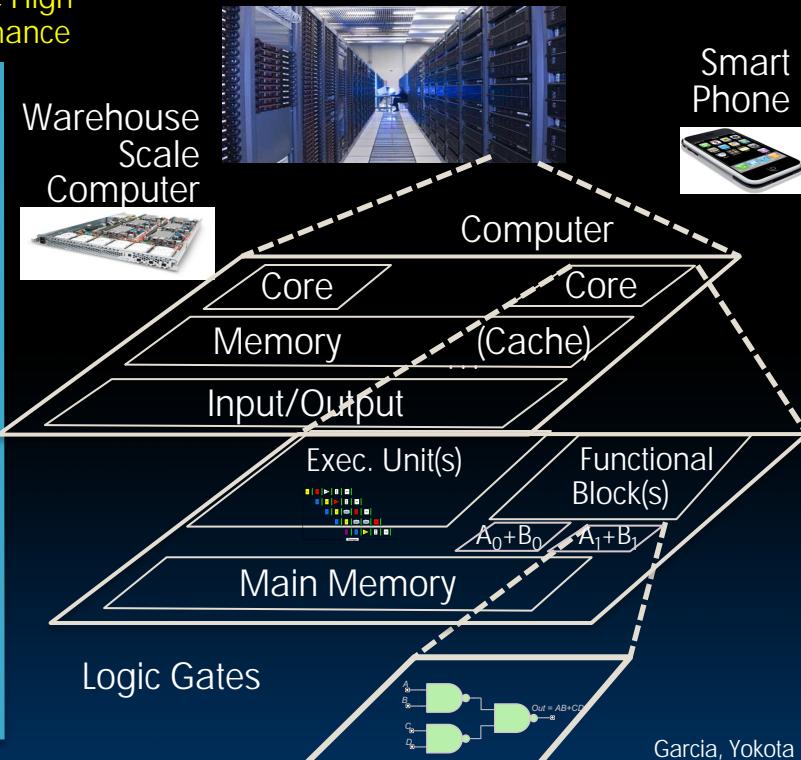
>1 data item @ one time  
e.g., Add of 4 pairs of words

## Hardware descriptions

All gates work in parallel at same time

Harness  
Parallelism &  
Achieve High  
Performance

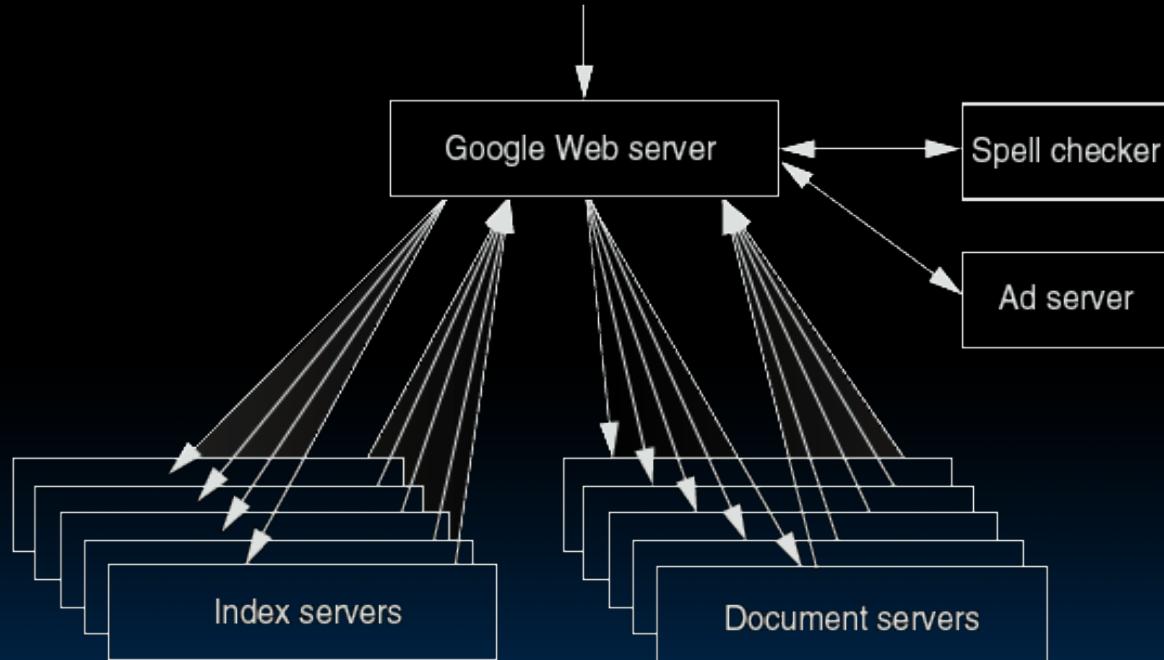
## Hardware



# Request-Level Parallelism (RLP)

- **Hundreds or thousands of requests/sec**
  - Not your laptop or cell-phone, but popular Internet services like web search, social networking, ...
  - Such requests are largely independent
    - Often involve read-mostly databases
    - Rarely involve strict read–write data sharing or synchronization across requests
- **Computation easily partitioned within a request and across different requests**

# Google Query-Serving Architecture



# Data-Level Parallelism (DLP)

- Two kinds:
  - Lots of **data in memory** that can be operated on in parallel (e.g. adding together 2 arrays)
  - Lots of **data on many disks** that can be operated on in parallel (e.g. searching for documents)
- Today's lecture: DLP across many servers and disks using **MapReduce**



# MapReduce

# What is MapReduce?

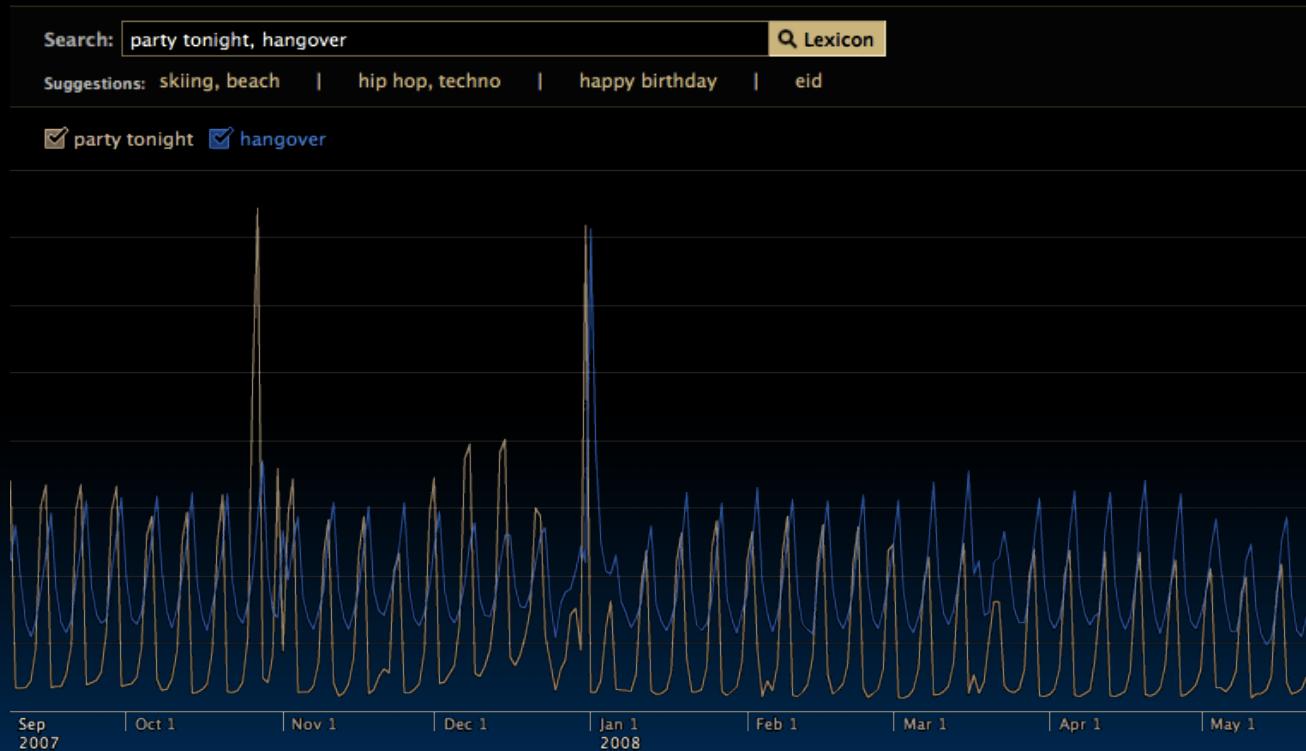
- Simple data-parallel programming model designed for scalability and fault-tolerance
- Pioneered by Google
  - Processes > 25 petabytes of data per day
- Open-source Hadoop project
  - Used at Yahoo!, Facebook, Amazon, ...



# What is MapReduce used for?

- At Google:
  - Index construction for Google Search
  - Article clustering for Google News
  - Statistical machine translation
  - For computing multi-layer street maps
- At Yahoo!:
  - “Web map” powering Yahoo! Search
  - Spam detection for Yahoo! Mail
- At Facebook:
  - Data mining
  - Ad optimization
  - Spam detection

# Example: Facebook Lexicon

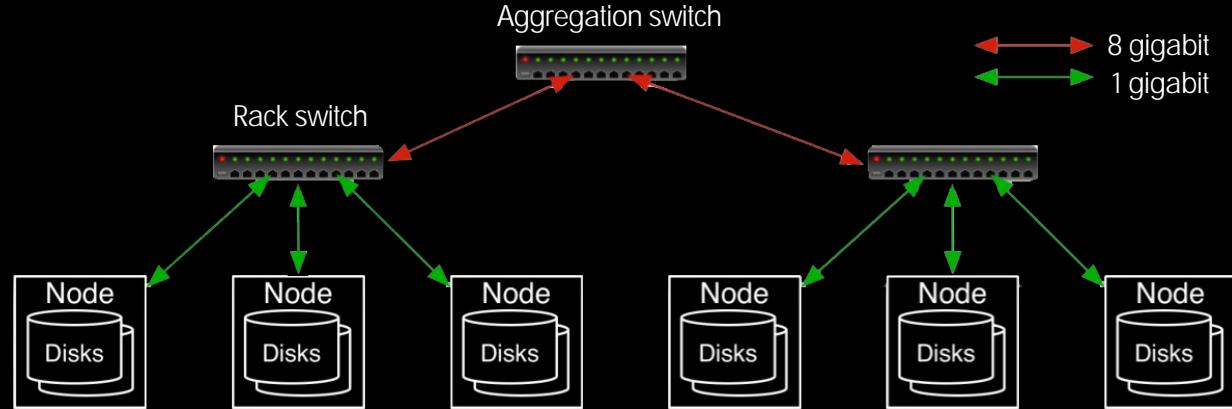




# MapReduce Design Goals

- **Scalability to large data volumes:**
  - 1000's of machines, 10,000's of disks
- **Cost-efficiency:**
  - Commodity machines (cheap, but unreliable)
  - Commodity network
  - Automatic fault-tolerance via re-execution (fewer administrators)
  - Easy, fun to use (fewer programmers)
- **Jeffrey Dean and Sanjay Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” 6th USENIX Symposium on Operating Systems Design and Implementation, 2004.**
  - optional reading, linked on course homepage – a digestible CS paper at the 61C level

# Typical Hadoop Cluster



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- Node specs (Yahoo terasort):  
8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

# MapReduce in CS10 & CS61A{,S}

combine with  $\bullet + \bullet$  items of

map square  $\bullet$  over list 1 20 3 10 ◀ ▶

510

square N

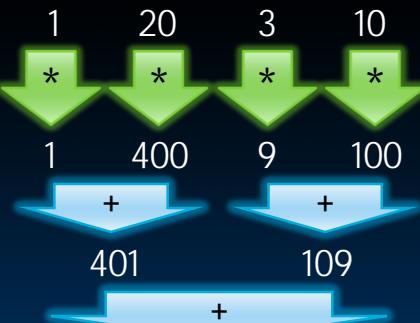
report N × N

```
> (reduce +
    (map square '(1 20 3 10)))
```

510

Input: 1 20 3 10

Note:  
only  
two  
data  
types!



Output: 510

```
>>> from functools import reduce
>>> def plus(x,y): return x+y
>>> def square(x): return x*x
>>> reduce(plus,
          map(square, (1,20,3,10)))
```

510

# MapReduce Programming Model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

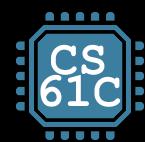
```
map (in_key, in_value) →  
      list(interm_key, interm_value)
```

- Processes input key/value pair
- Slices data into “shards” or “splits”; distributed to workers
- Produces set of intermediate pairs

```
reduce (interm_key, list(interm_value)) →  
       list(out_value)
```

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usu just one)

[code.google.com/edu/parallel/mapreduce-tutorial.html](http://code.google.com/edu/parallel/mapreduce-tutorial.html)



# MapReduce WordCount Example

- "Mapper" nodes are responsible for the **map** function

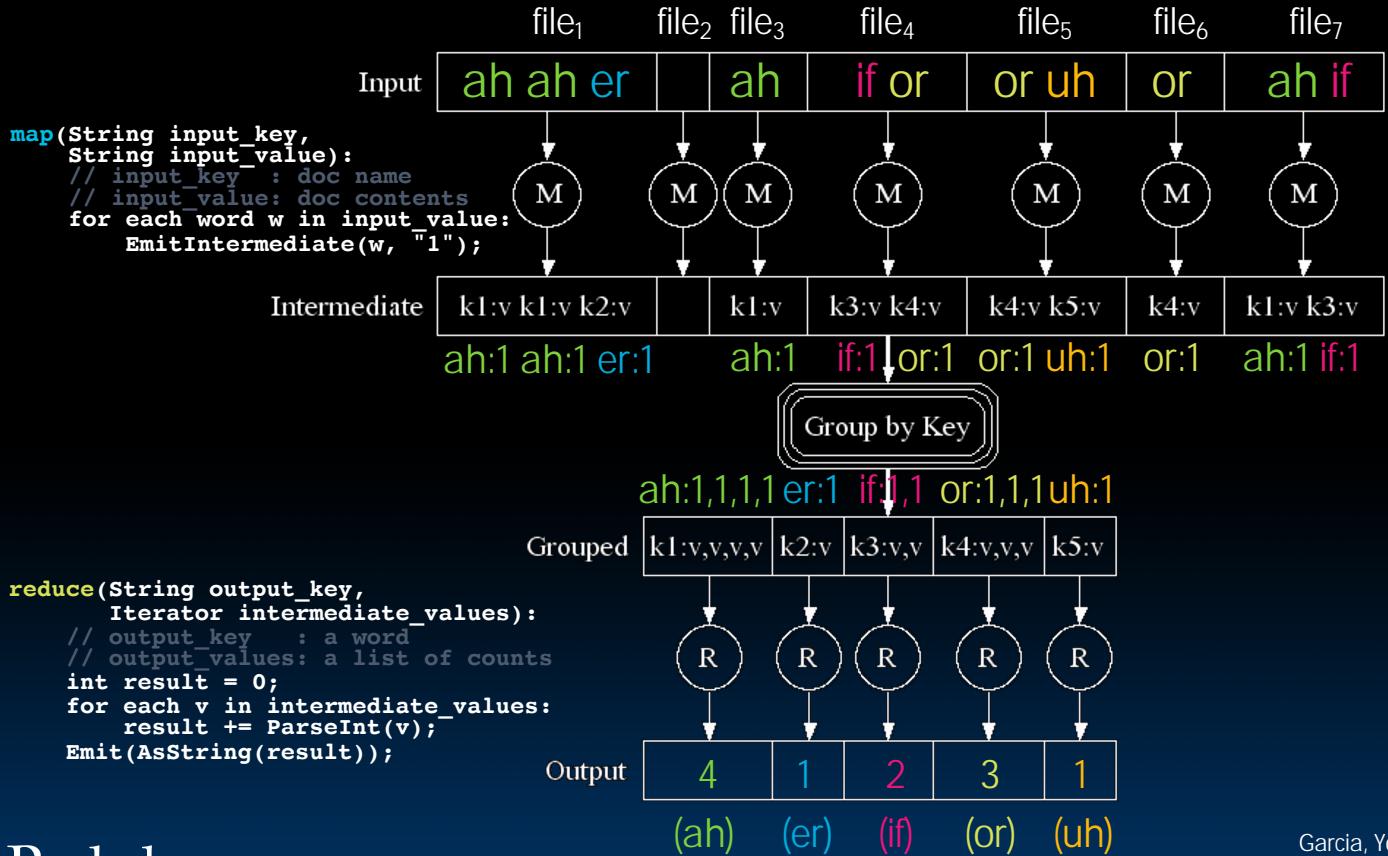
```
// "I do I learn" → ("I",1), ("do",1), ("I",1), ("learn",1)
map(String input_key,
      String input_value):
    // input_key : document name (or line of text)
    // input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");
```

- "Reducer" nodes are responsible for the **reduce** function

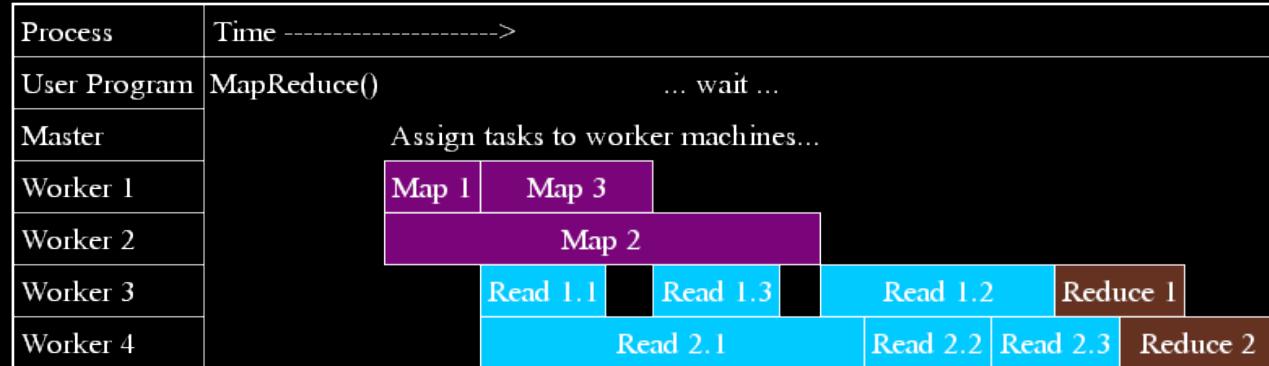
```
// ("I", [1,1]) → ("I", 2)
reduce(String output_key,
        Iterator intermediate_values):
    // output_key : a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(AsString(result));
```

- Data on a distributed file system (DFS)

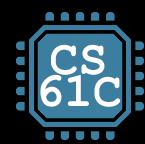
# MapReduce WordCount Diagram



# MapReduce Processing Time Line



- Master assigns map + reduce tasks to “worker” servers
- As soon as a map task finishes, worker server can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
- To tolerate faults, reassign task if a worker server “dies”



# MapReduce WordCount Java code

```
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WCMap.class);
    conf.setCombinerClass(WCReduce.class);
    conf.setReducerClass(WCReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

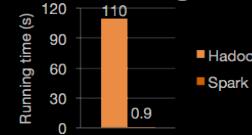
public class WCMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.nextToken()), ONE);
        }
    }
}

public class WCReduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values,
                      OutputCollector output,
                      Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```



# Spark

- Apache Spark™ is a fast and general engine for large-scale data processing.
- Speed
  - Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
  - Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.
- Ease of Use
  - Write applications quickly in Java, Scala or Python.
  - Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala and Python shells.



# Word Count in Spark's Python API

```
file.flatMap(lambda line: line.split()
            .map(lambda word: (word, 1))
            .reduceByKey(lambda a, b: a+b)
```

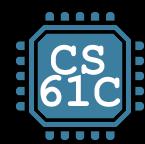
Cf Java:

```
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setMapperClass(WMap.class);
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WMap.class);
    conf.setCombinerClass(WReduce.class);
    conf.setReducerClass(WReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

public class WMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.nextToken()), ONE);
        }
    }
}

public class WReduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values,
                      OutputCollector output,
                      Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```





# flatMap in Spark's Python API

```
>>> def neighbor(n):
...     return [n-1,n,n+1]
>>> R = sc.parallelize(range(5))
>>> R.collect()
[0, 1, 2, 3, 4]
>>> R.map(neighbor).collect()
[[-1, 0, 1], [0, 1, 2], [1, 2,
3], [2, 3, 4], [3, 4, 5]]
>>> R.flatMap(neighbor).collect()
[-1, 0, 1, 0, 1, 2, 1, 2, 3, 2,
3, 4, 3, 4, 5]
```

# Word Count in Spark's Python API

```
unix% cat file.txt
```

```
ah ah er
```

```
ah
```

```
if or
```

```
or uh
```

```
or
```

```
ah if
```

```
>>> w = sc.textFile("file.txt")
```

```
>>> w.flatMap(lambda line: line.split()).collect()
```

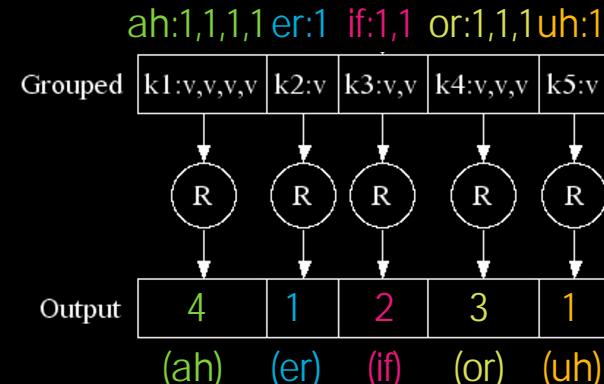
```
['ah', 'ah', 'er', 'ah', 'if', 'or', 'or', 'uh', 'or', 'ah', 'if']
```

```
>>> w.flatMap(lambda line: line.split()).map(lambda word:  
(word,1)).collect()
```

```
[('ah', 1), ('ah', 1), ('er', 1), ('ah', 1), ('if', 1), ('or', 1),  
('or', 1), ('uh', 1), ('or', 1), ('ah', 1), ('if', 1)]
```

```
>>> w.flatMap(lambda line: line.split()).map(lambda word:  
(word,1)).reduceByKey(lambda a,b: a+b).collect()
```

```
[('er', 1), ('ah', 4), ('if', 2), ('or', 3), ('uh', 1)]
```





# Parallel? Let's sanity-check...

```
>>> def crunch(n):
...     time.sleep(5) ## to simulate number crunching
...     return n*n
...
...
>>> crunch(10) ## 5 seconds later
100
>>> list(map(crunch,range(4))) ## 20 seconds later
[0, 1, 4, 9]
>>> R = sc.parallelize(range(4))
>>> R.map(crunch).collect() ## 5 seconds later
[0, 1, 4, 9]
```

When poll is active, respond at [pollev.com/ddg](http://pollev.com/ddg)

Text **DDG** to **22333** once to join

## L36

- Writing & managing SETI@Home is relatively straightforward; just hand out and gather data
- Parallel programs run on N identical processors will yield close to Nx speed increase
- The world's computing power mostly lives in datacenters

FFF  
FFT  
FTF  
FTT  
TFF  
TFT  
TTF  
TTT

- 4th big idea is parallelism
- MapReduce is a wonderful abstraction for programming lots of machines
  - Abstracts machine failures, slow machines
  - File-based
- Spark does it even better
  - Memory-based
  - Lazy evaluation

