MapReduce: Programming in the Very Large

Ari Rabkin: asrabkin@gmail.com

for David Walker's FP class

December 2012

"The datacenter is the computer"

Google versus Hadoop vs MR

- Google published the MapReduce paper in 2004.
- Doug Cutting had been working on an Open Source MapReduce. Linked up with Yahoo! to scale it up.
- Has taken off and become very popular
- Other MapReduce implementations also exist

Indexing

- 1: Some Words
- 2: Some other words
- 3: Other words

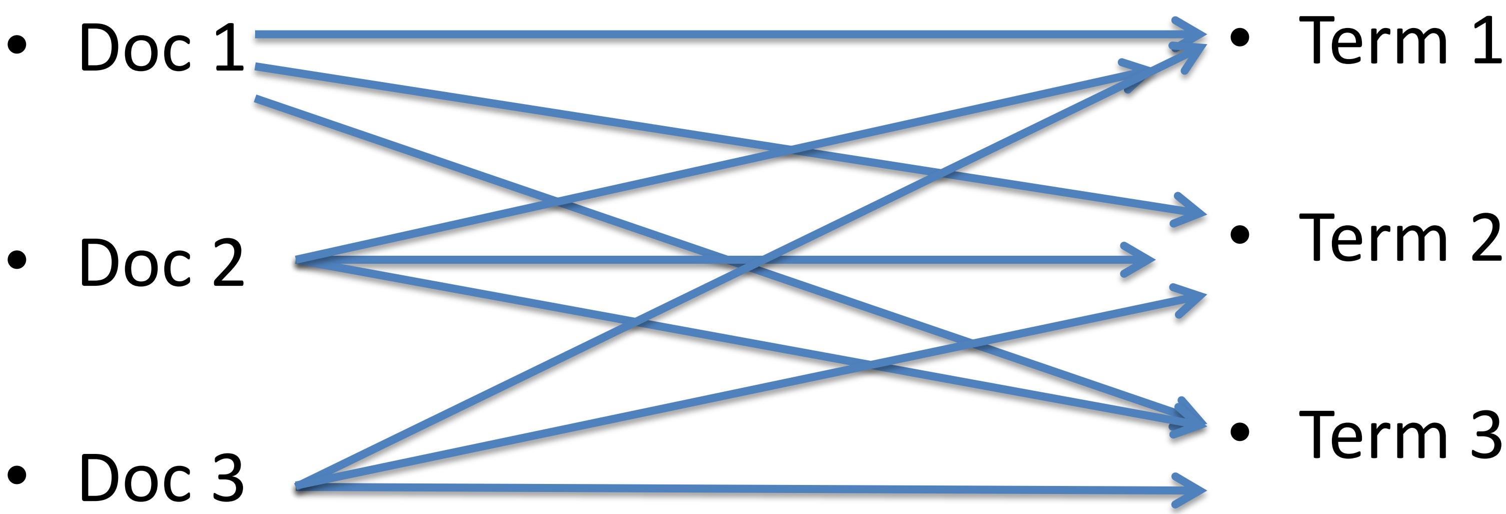
- Other: 2,3
- Some: 1, 2
- Words: 1,2,3

Note that Index is **sorted** by key.
Helpful for quick lookup of
approximate matches

Generalizing

Inputs

Outputs

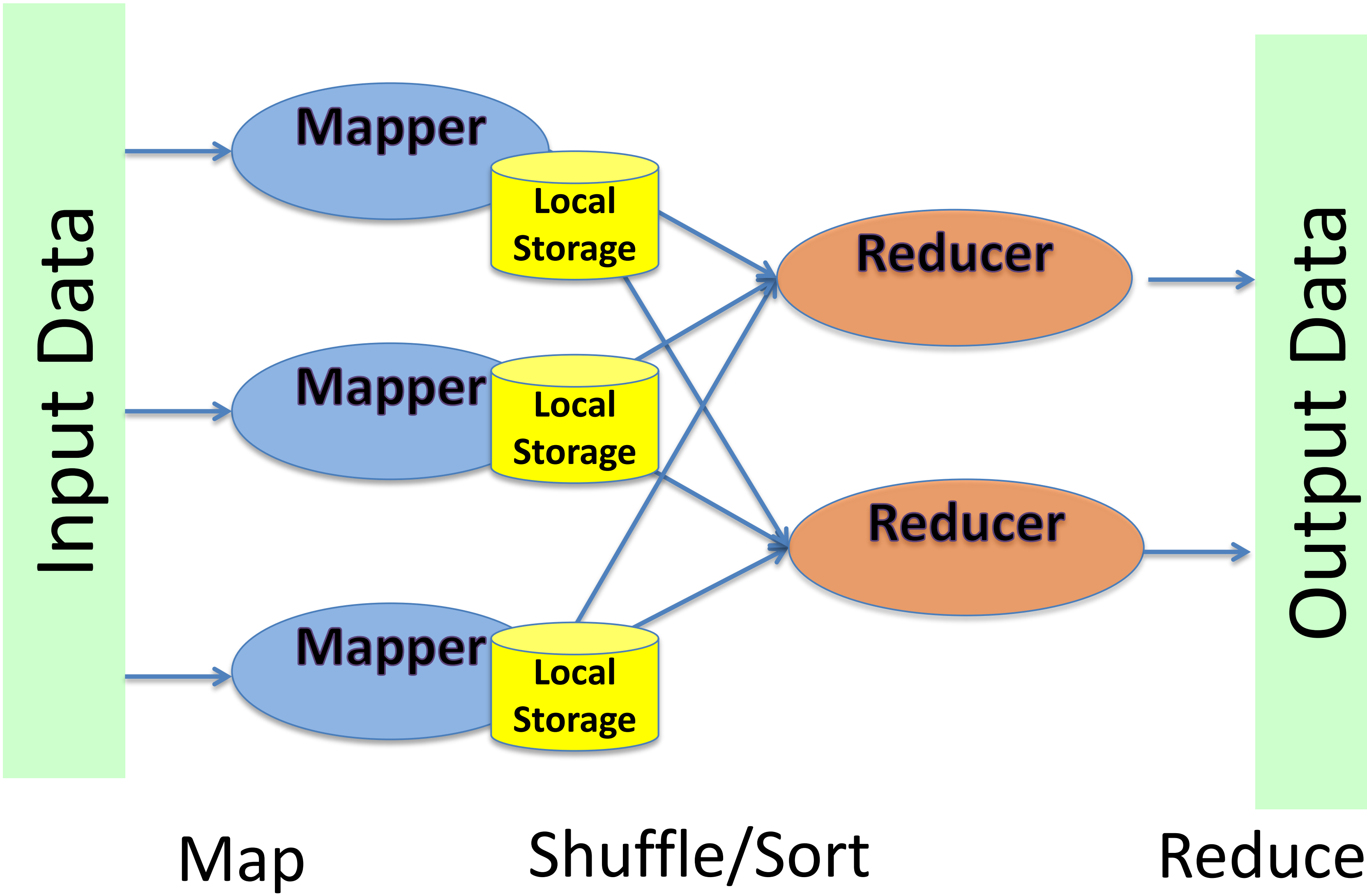


Map → Shuffle/Sort → Reduce

Performance Numbers

- Biggest production Hadoop clusters are ~4000 nodes
- Facebook has 100 PB in Hadoop
- Best MapReduce-like system (TritonSort from UCSD) can sort 900 GB/minute on a 52-node, 800-disk cluster.

Distributed Implementation



A modern software stack

Workload Manager

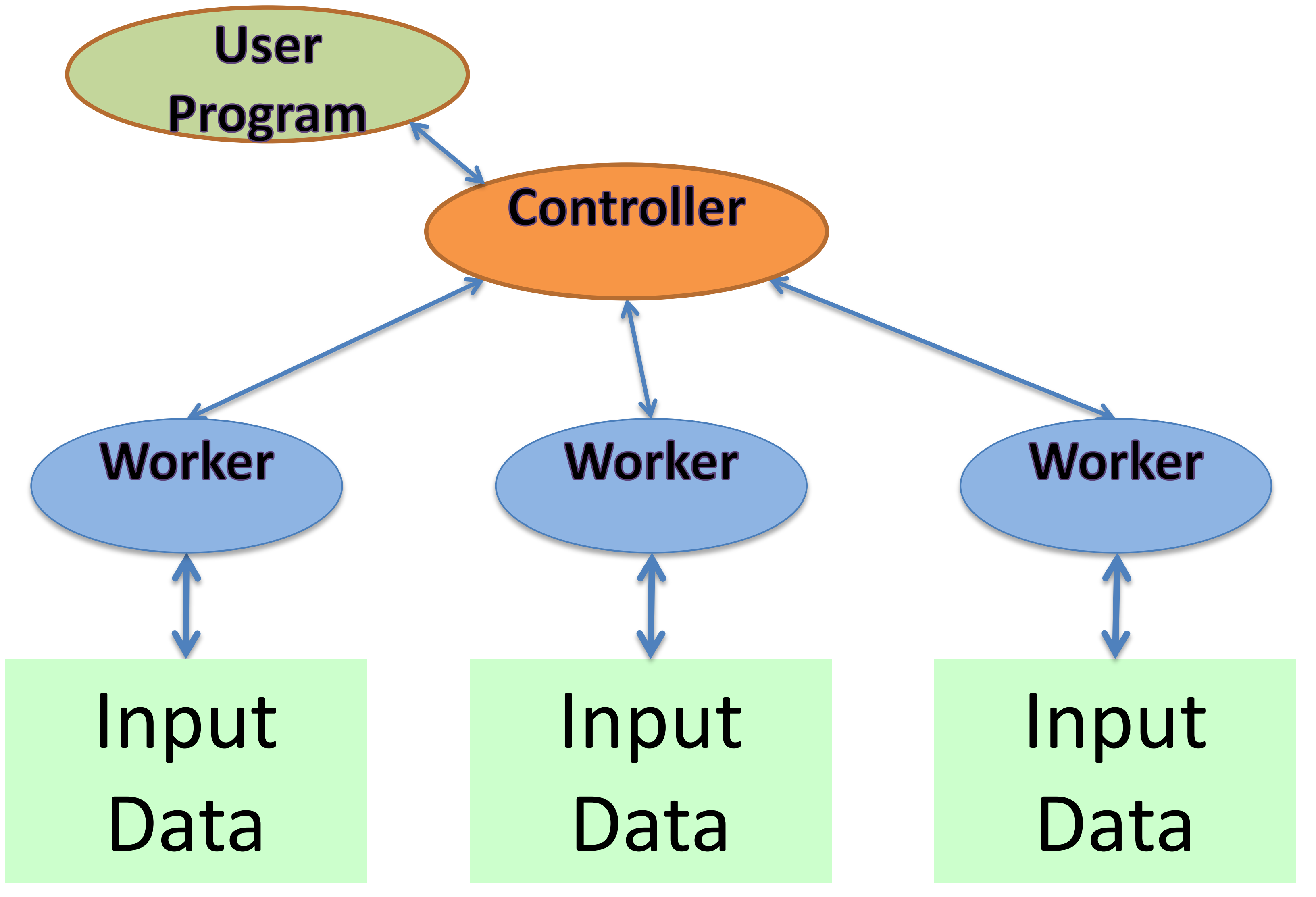
High-level scripting language

e

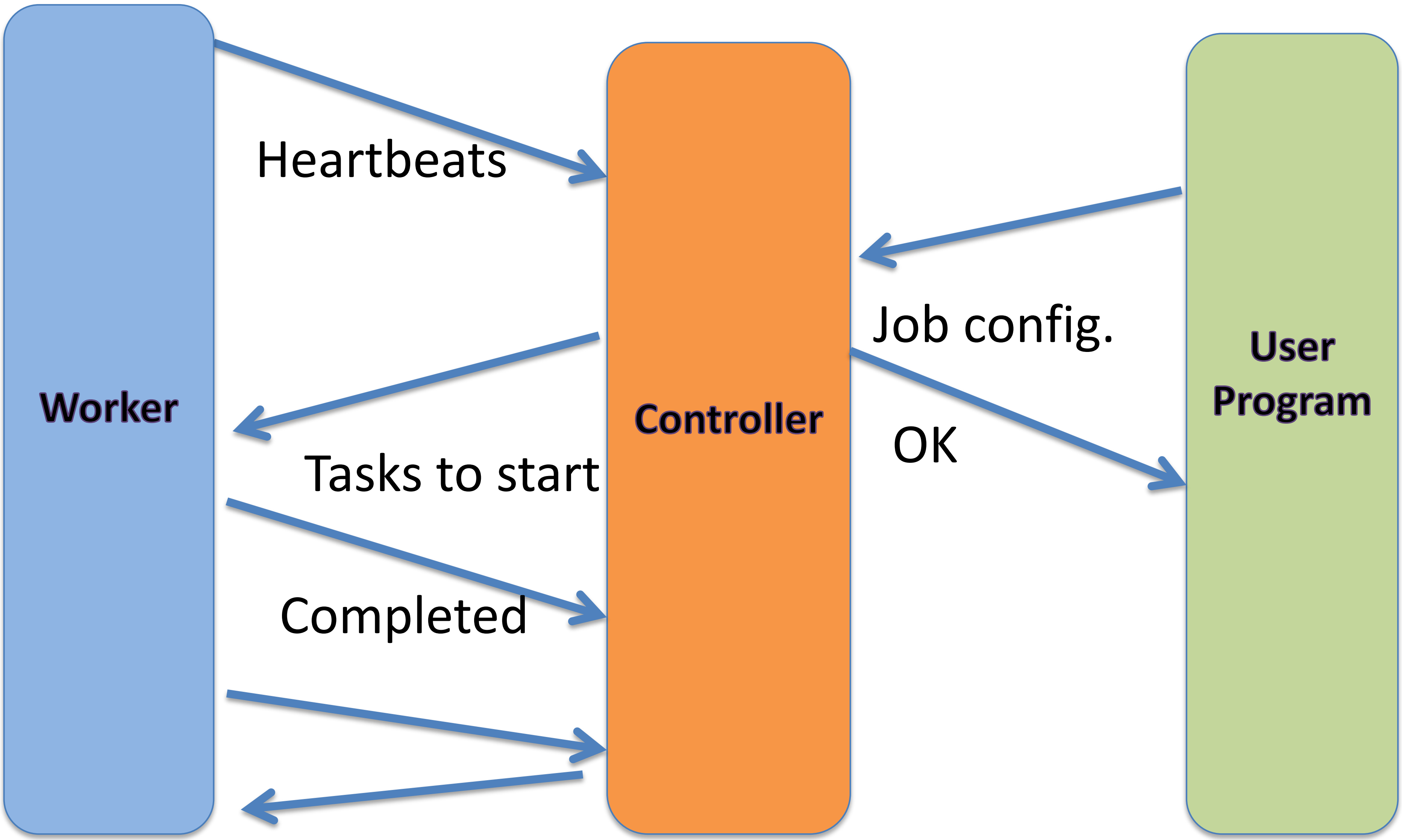
l ue

Cluster Node | Cluster Node | Cluster Node | Cluster Node

The control plane

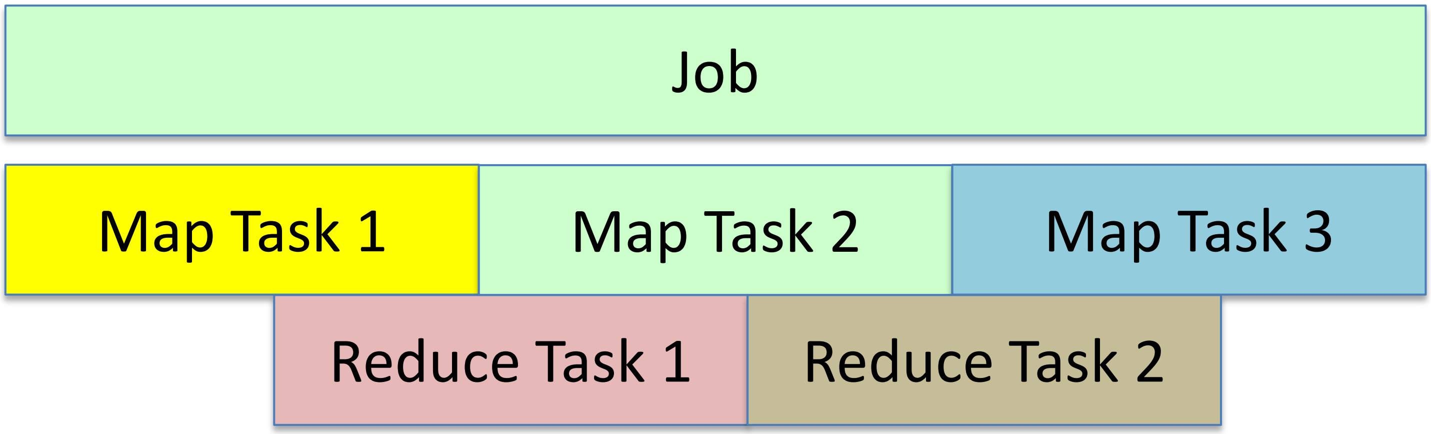


The flow of information



Slots, Tasks, and Attempts

- A **job** is split into **tasks**. Each **task** includes many calls to map() or reduce()

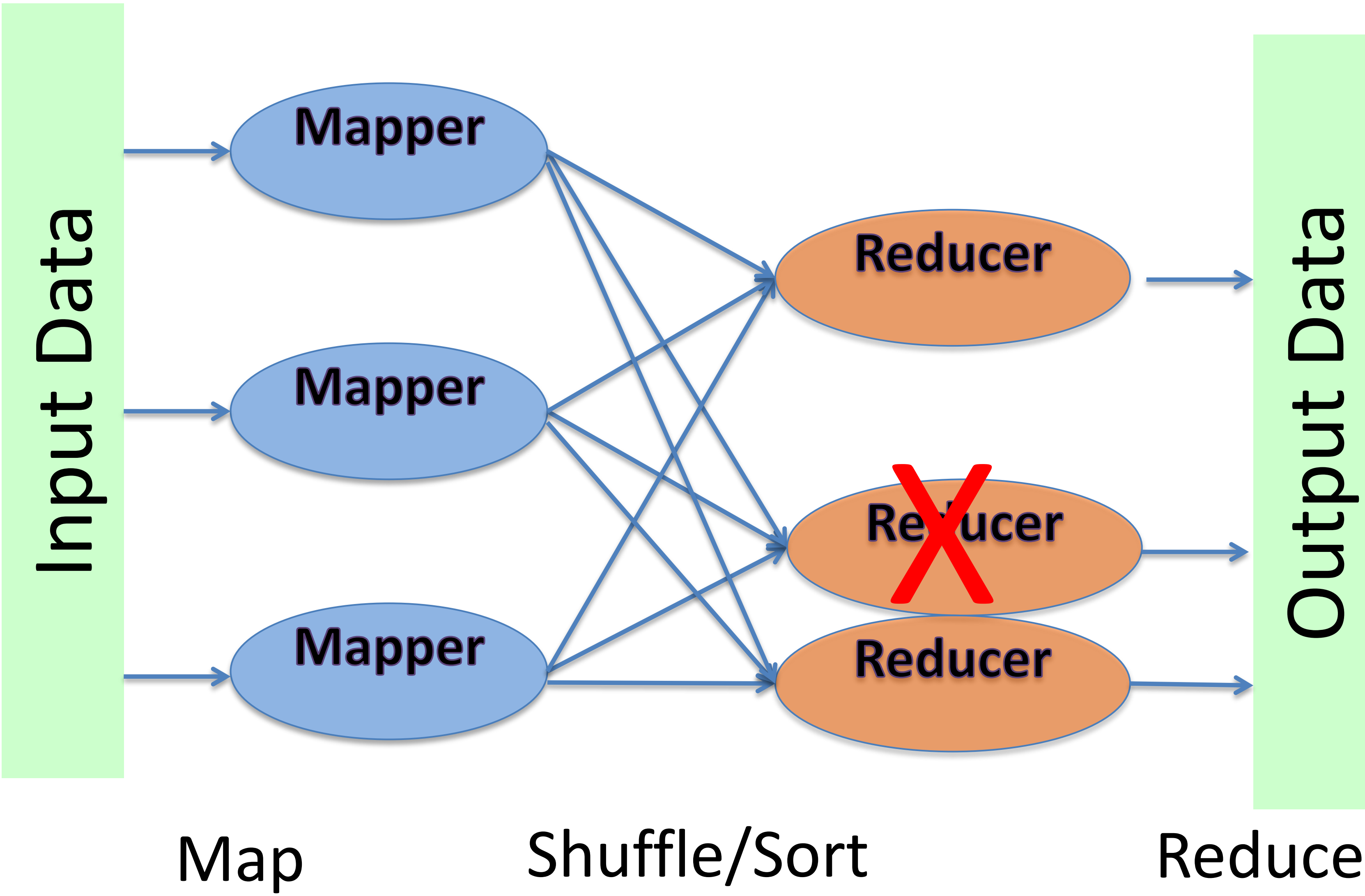


- Workers are long-running processes that are assigned tasks
- Multiple workers can be assigned the same task; these are termed separate **attempts.**

Size and Failures

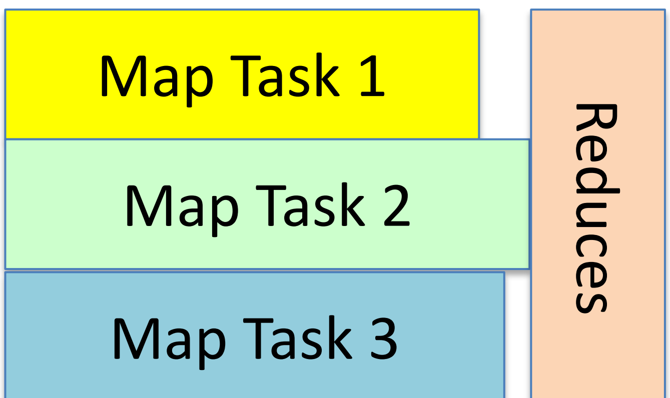
- Suppose you have a cluster of a thousand servers. How long between failures?
- How long for one machine to fail?
  - Intuition: machines fail once a year or two?
  - Depending on model, perhaps 5% of high-end hard disks fail each year (Schroder, FAST ’07). A server might have ten hard disks.
- So for a thousand machines, we would expect failures **more than once a day**

Handling Failures

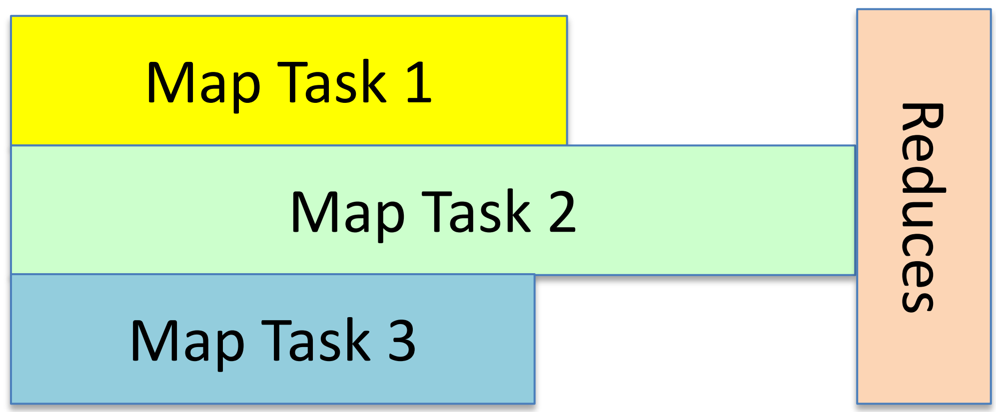


Failures aren’t absolute

- Some failures make nodes slow
- Reduces can’t start until ALL maps finish



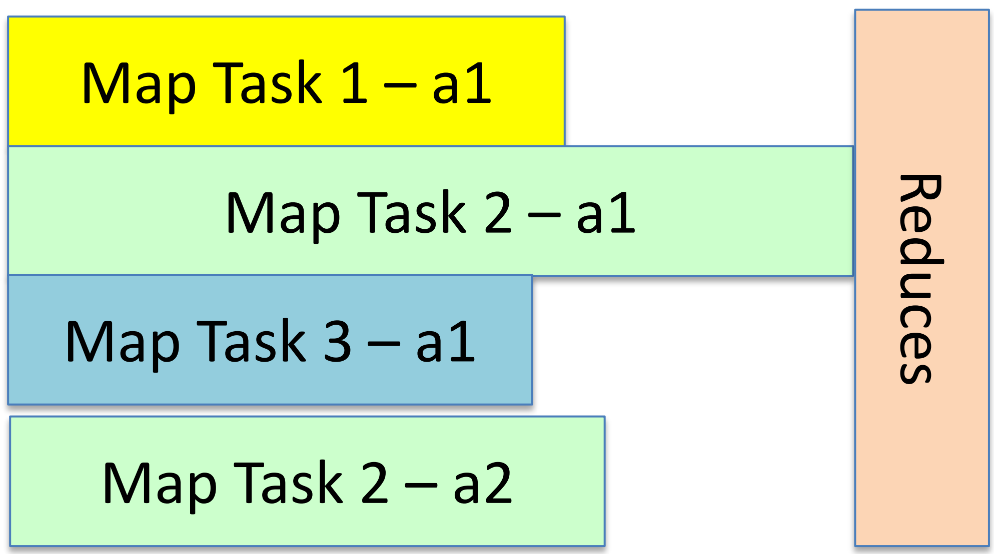
OK



Bad!

Fix: speculation

- Multiple tries at same task; pick first to finish
- Subtlety in deciding which tasks to try to speculatively execute



Types for Map + Reduce functions

- Map:

  (‘K1 * ‘V1 → (‘K2 * ‘V2) bag) -> (‘K1 * ‘V1) bag -> (‘K2 * ‘V2) bag

- Reduce:

  (‘K2 * (‘V2 list) → (‘K3 * ’V3) bag) -> ‘K2 * (‘V2 list) bag -> (‘K3 * ’V3) bag

Indexing

Map: (DocID * word bag) → (word * DocID) bag

Reduce: (word * DocID list) → (word * DocID) bags

The Java versions

```
interface Mapper<K1,V1,K2,V2> {
      public void map (K1 key,
            V1 val,
            OutputCollector<K2, V2> output);
       ...
}
```

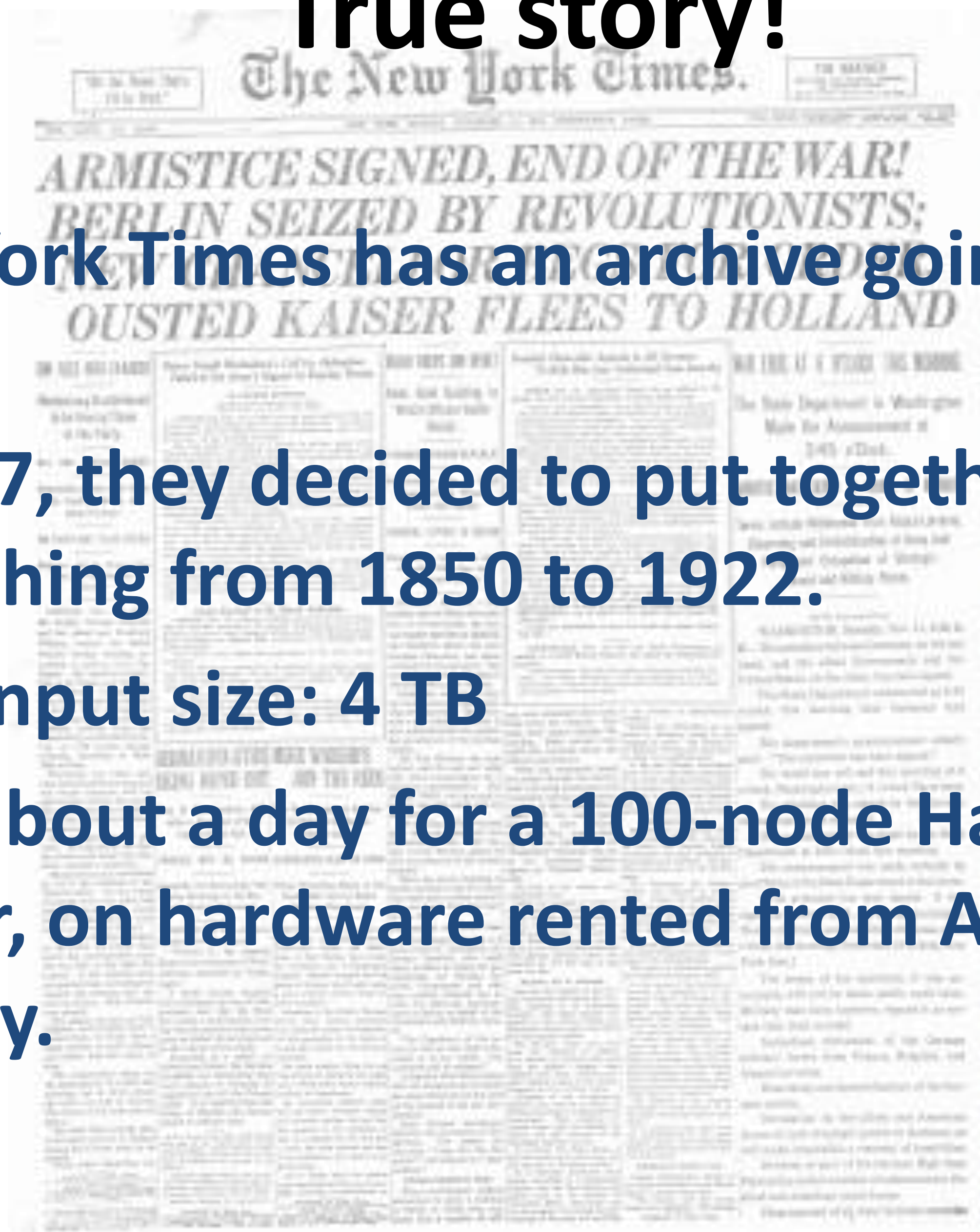
The Java versions

```
interface Reducer<K2,V2,K3,V3> {
      public void reduce(K2 key,
             Iterator<V2> values,
             OutputCollector<K3, V3> output);
       ...
}
```

Image to Text

- Can use MapReduce for simple parallelization.
- Imagine we have code to convert an image to text. How do we convert a million scanned images of book pages?
- Can just wrap the conversion routine in our Map() method; reduce is identity
- The embarrassingly parallel becomes trivial; real power of framework is in harder parallel problems.

True story!



The New York Times.

ARMISTICE SIGNED, END OF THE WAR!
BERLIN SEIZED BY REVOLUTIONISTS;
OUSTED KAISER FLEES TO HOLLAND

- New York Times has an archive going back to 1850.
- In 2007, they decided to put together PDFs of everything from 1850 to 1922.
- Total input size: 4 TB
- Took about a day for a 100-node Hadoop cluster, on hardware rented from Amazon for the day.

Word count?

- Similar to indexing except we only want counts, not locations
- Map:
  (DocID, String list) -> ?

- Reduce:
  .... -> (String, int)

Word count?

- Similar to indexing except we only want counts, not locations
- Map:
  (DocID, String list) -> (String, _) bag

- Reduce:
  (String, _ list) -> (String, int)

Word count?

- Similar to indexing except we only want counts, not locations
- Map:
  (DocID, string list) -> (string, unit) bag
  emit (w, ()) for each word w in list
- Reduce:
  (string, unit list) -> (string, int)
  emit length of list

Map in Java

```
class WordCountMap implements Map {
 public void map (DocID key,
 List<String> val,
 OutputCollector<String, Integer> output) {

   for (String s: val)
      output.collect(s, 1)

  }
 }
```

Reduce in Java

```
class WordCountReduce {
  public void reduce(String key,
  Iterator<Integer> vals,
  OutputCollector<String, Integer> output) {

    int count = 0;
    for (int v: vals)
       count += 1;
    output.collect(key, count)
   }
  }
```

Map + Reduce, and Combine functions

- Map:

  ('K1 * 'V1) → ('K2 * 'V2) bag

- Reduce:

  ('K2 * 'V2 list) → ('K3 * 'V3) bag

- **Combine**

  ('K2 * 'V2 list) → ('K2 * 'V2) bag

Reduce / Combine in Java

```
class WordCountReduce {
  public void reduce(String key,
  Iterator<Integer> vals,
  OutputCollector<String, Integer> output) {

    int count = 0;
    for (int v: vals)
        count += v;
    output.collect(key, count)
   }
  }
```

Word Count with Combine

- Almost the same functional code, different configuration

```
conf.setOutputKeyClass(String.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(WordCountMap.class);
conf.setReducerClass(WordCountReduce.class);

conf.setCombinerClass(WordCountReduce.class);
```

A hypothetical....

```
HashMap<String, Integer> counts =
   new HashMap<String, Integer>();

public void map (DocID key,
                 List<String> val,
   OutputCollector<String, Integer>
   output) {

     for (String s: val) {
        count = 1;
        if (counts.contains(s))
           count += counts.get(s);
        counts.put(s, count);
     }
   }
```

A: Correct program

B: Compiler Error

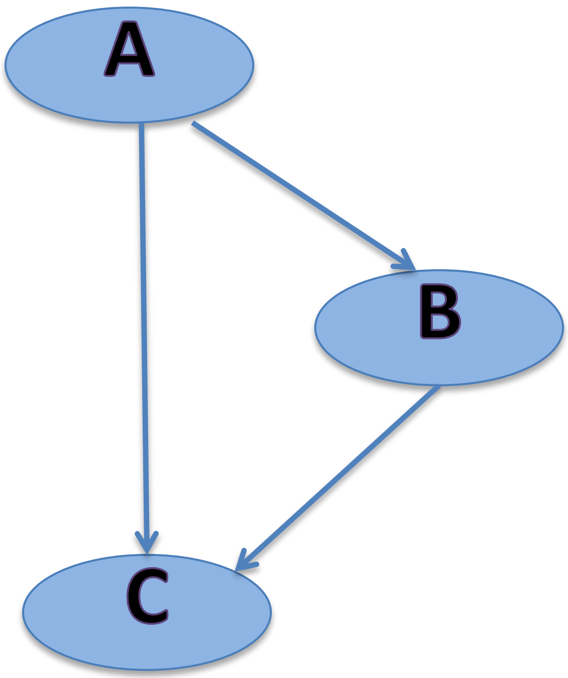
C: Program produces wrong answer

PageRank: measuring how much a webpage matters

- Model: user is clicking around randomly.
- With probability k, will start over at random; else follows a [random] link off current page.
- Matrix M encodes probabilities of transition from page p to page q

- Pr[on page] = $\mathbf{M} \bullet \mathbf{e}_p$

PageRank: The link matrix



	A	B	C
A	0	1	1
B	0	0	1
C	0	0	0

The Stable State

- Distribution has a stationary point where
  $v = \mathbf{M} \bullet v$ (v is an eigenvector)
- Can solve by iteration: $v_{k+1} = \mathbf{M} \bullet v_k$
- We can compute this as a MapReduce job

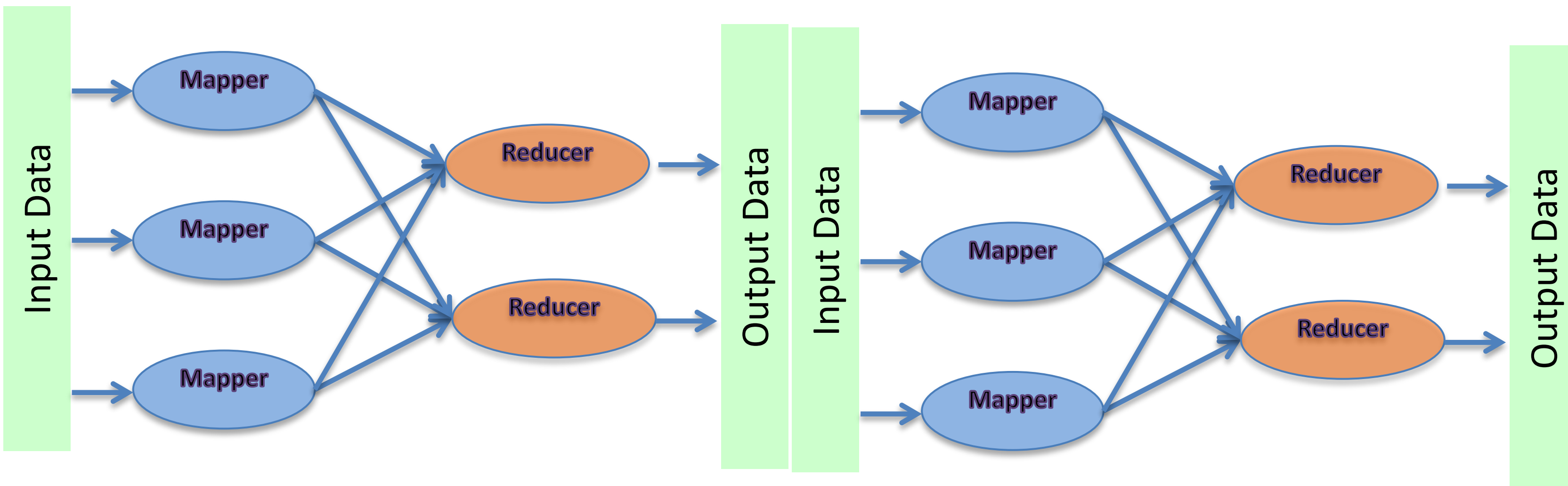
Defining the types

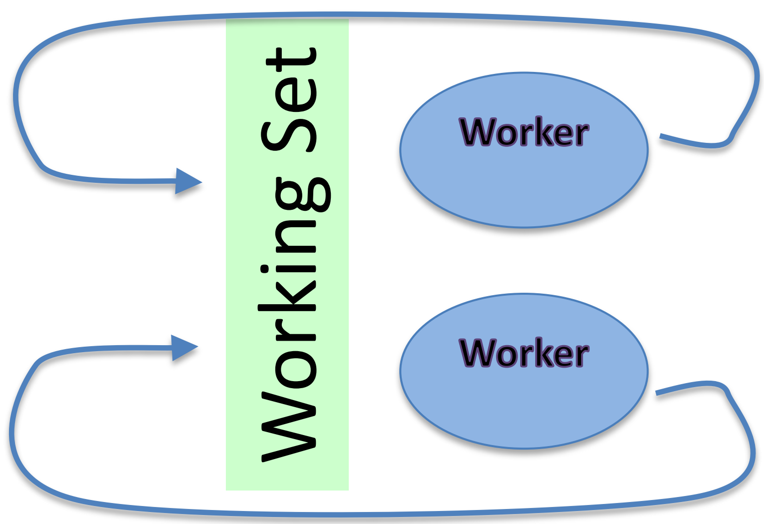
- `Class PageInfo;`
- ```
  Class LinksInfo extends PageInfo {
      List<DocID> links;
  }
  ```
- ```
  Class Increment extends PageInfo {
      double inWeight;
  }
  ```

The logic

```
Reduce(DocID key, Iterator<PageInfo> vals,
OutputCollector<DocID, PageInfo> output {
  double total_score = 0;
  LinksInfo  info;
  for ( PageInfo i: vals) {
    if (i instanceof LinksInfo) {
        info = (LinksInfo) vals.next();
        output.collect(key, info)
     } else
        total_score += ((Increment) i).inWeight;
   }
  double s = total_score / info.links.size()
  for (DocID out: links.links)
    output.collect(key, Increment(s))
}
```

Iterative Jobs are common…





Joins

Name	ZIP Code
John Doe	08540
Richard Roe	20037

ZIP code	State
08540	NJ
14850	NY
20037	DC

Name	ZIP Code	State
John Doe	08540	NJ
Richard Roe	20037	DC

Joins with MapReduce

Name	ZIP Code
John Doe	08540
Richard Roe	20037

ZIP code	State
08540	NJ
14850	NY
20037	DC

- If one table is small, just keep it in memory at every location and join in the Map method
- Can also join on Reduce side
  - Can emit whole contents of both tables in Map.
  - Use "join column" as sort key, then join in reduce().
- Higher-level languages help. (Pig, Hive, etc)

Joins with MR, continued

- Class TableCell [could be int, string, etc]
- Class RowWithSource

```
map(NullWritable inKey, TableRow val …){
  int fileId = getInputFileNumber();
  int joinCol = config.get("join_column_" +
fileId);
  TableCell c = val.get(joinCol);
  RowWithSource v2 =new RowWithSource(val, fileId);
  output.collect(c, v2);
}
```

Joins with MR, continued

Initialize joinCol1 and joinCol2 [class members] somewhere

```
reduce(TableCell key, Iterator <RowWithSource> values …){
  List<RowWithSource> src1 = new List<RowWithSource> ();
  List<RowWithSource> src2 = new List<RowWithSource> ();

  for (RowWithSource r: values)
    if (r.src == "1")
      src1.append(r);
    else
      src2.append(r);

  for (RowWithSource r1: src1)
    for (RowWithSource r2: src2) {
      TableRow res = join(r1, r2, joinCol1, joinCol2);
      output.collect(null, res);
    }
}
```

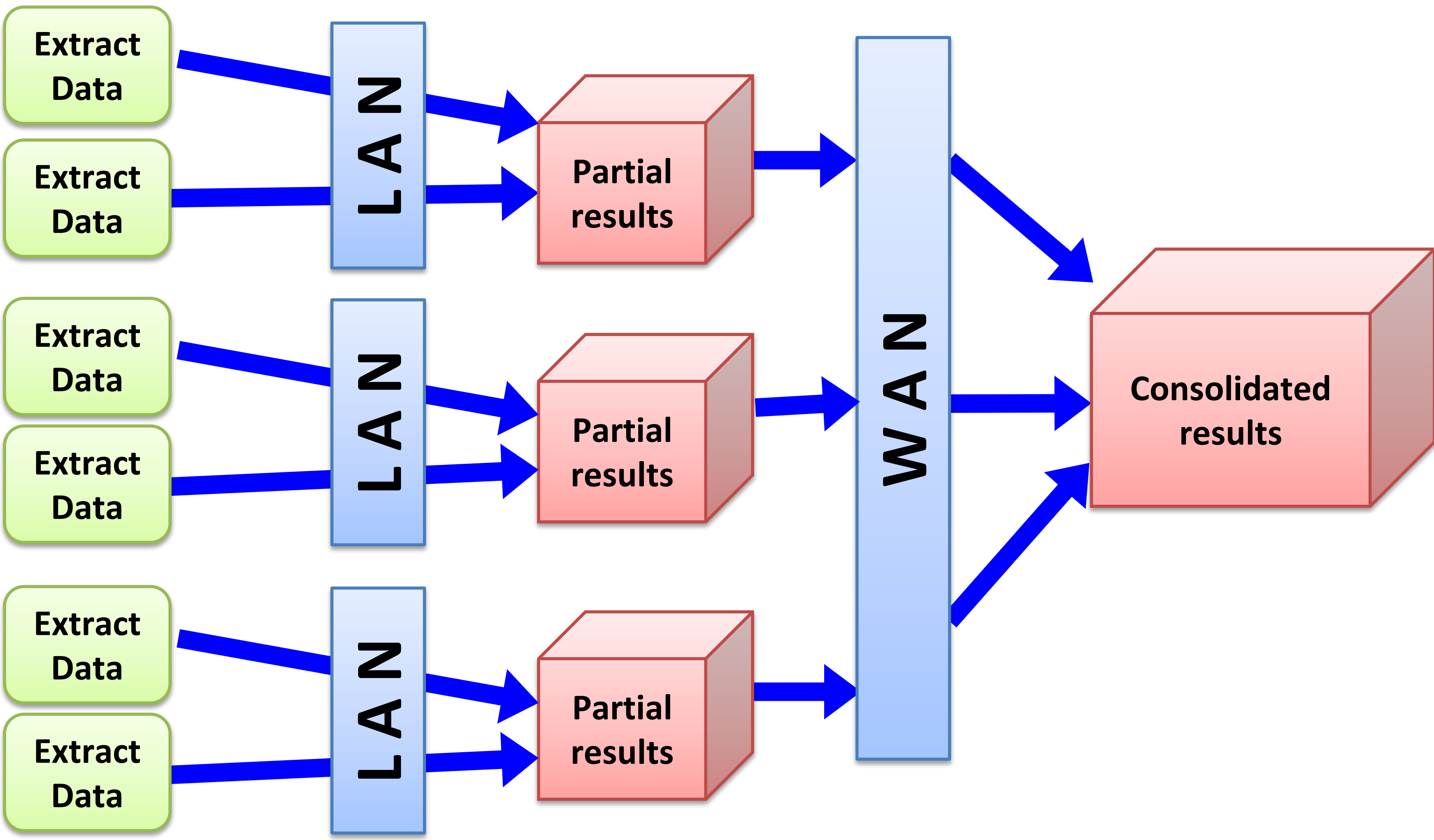
Observations

- Code is basically doing nested-loops over all pairs of rows which match on the join key.
- This doesn’t require materializing the whole set of results, but does materialize the sets of inputs on each side.
- This code would be a lot easier with product types (e.g. Pair<A,B>)

What I work on

MapReduce is the Wrong Thing if the data is spread out: need more optimization to reduce wide-area transfer costs

Deeper pipes for more locality



Take-aways

- Big data needs specialized tools to process.
- Higher-order functions help manage complexity.
- Determinism and the absence of side-effects make parallelism and failure recovery simpler.
- If you have complicated functionality, consider building a language

For more information

- Hadoop is public and open source.
- See http://hadoop.apache.org for information.
- Amazon's EC2 will let you run stuff at large scale for low (and incremental) costs.