MapReduce

MIH-CPP

Learning Objectives

Understand the concepts behind MapReduce program model

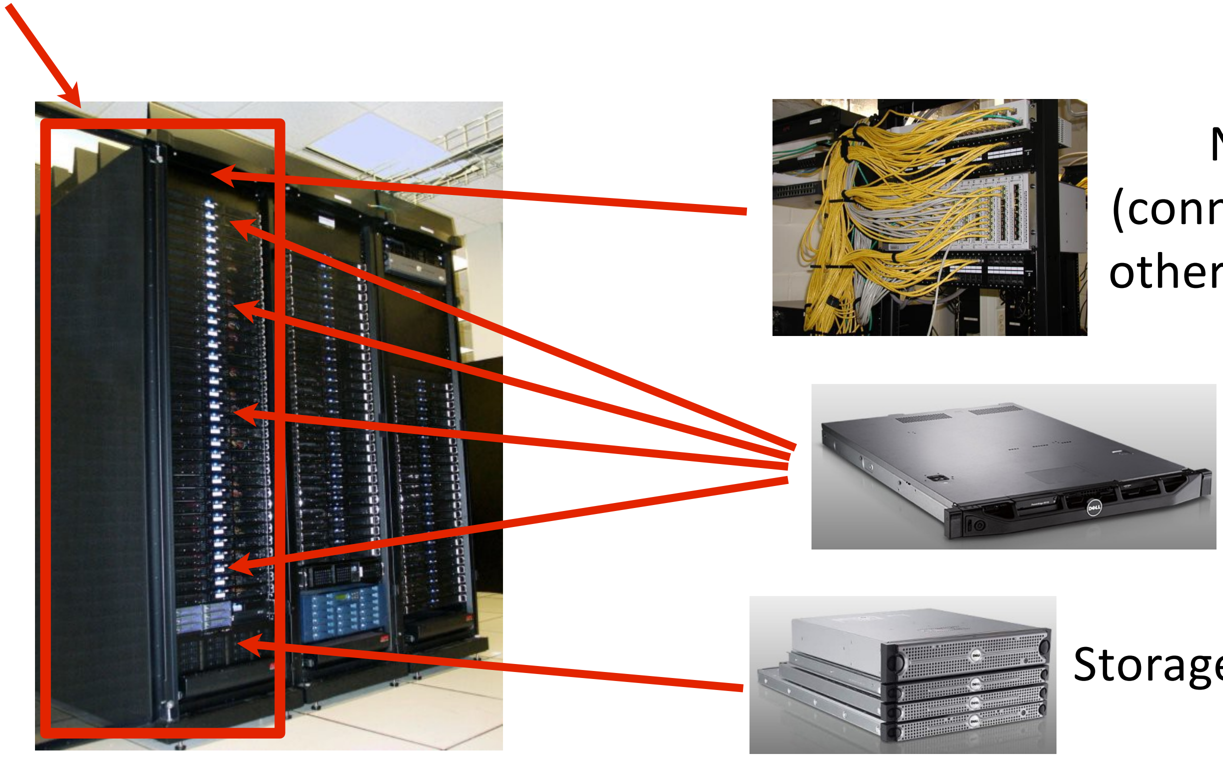
Explore the implementation of MapReduce architecture

Analyze a simple problem in MapReduce program model

Analyze a practical problem in MapReduce program model

Recall this diagram?

Rack



Network switches
(connects nodes with each other and with other racks)

Many nodes/blades
(often identical)

Storage device(s)

Processing and Analyzing Big Data

- A toy problem: The word count
  - We have 10 billion documents
  - Average document's size is 20KB => 10 billion docs = 200TB

- Cute solution:

```
for each document d
{ for each word w in d {word_count[w]++;}
```

Time elapsed..

Background

- Traditional programming is serial
- Parallel programming
  - Break processing into parts that can be executed concurrently on multiple processors
- Challenge
  - Identify tasks that can run concurrently
    - and/or groups of data that can be processed concurrently
  - Not all problems can be parallelized

MapReduce in Big Data Context

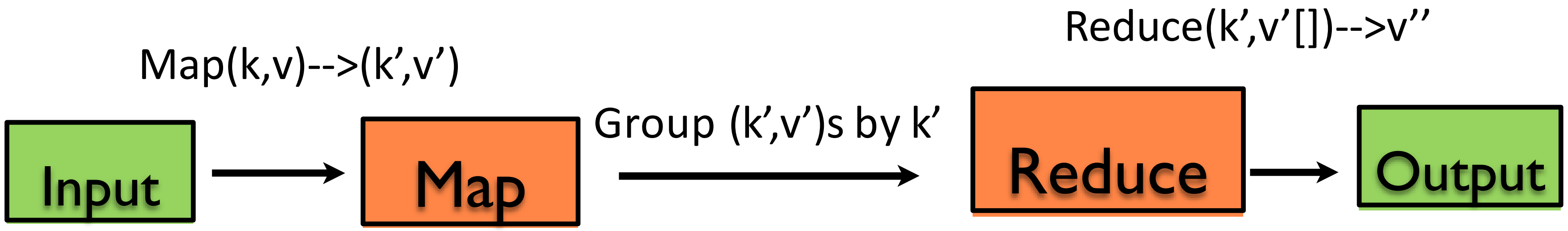
- Created by Google in 2004
  - Jeffrey Dean and Sanjay Ghemawat
- Inspired by LISP
  - `Map(function, set of values)`
    - Applies function to each value in the set
    - `(map ‘length ‘(() (a) (a b) (a b c)))` ⇒ `(0 1 2 3)`
  - `Reduce(function, set of values)`
    - Combines all the values using a binary function (e.g., +)
    - `(reduce #'+ ‘(1 2 3 4 5))` ⇒ `15`

MapReduce in Big Data Context

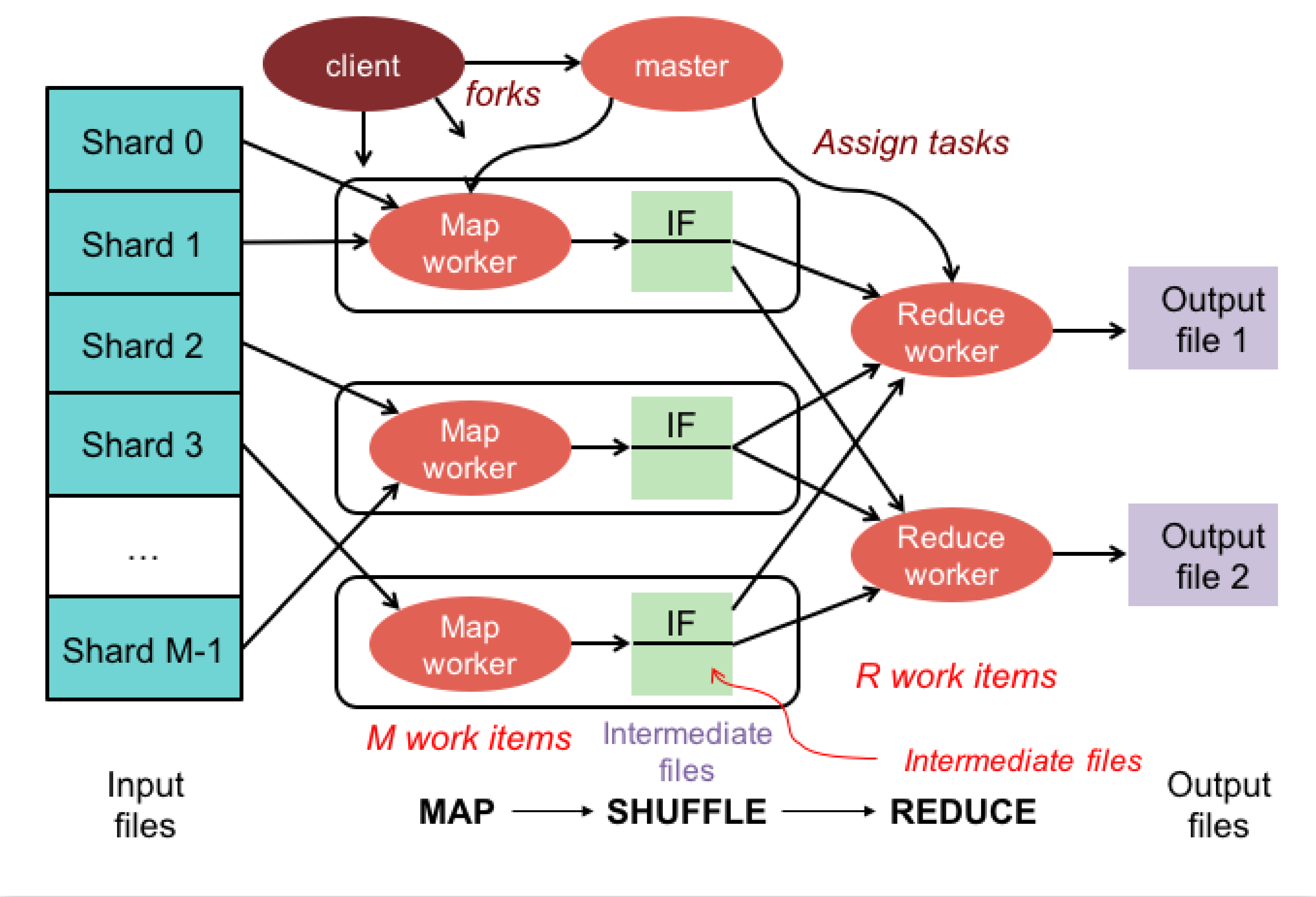
- Framework for parallel computing
- Programmers get simple API
- Don’t have to worry about handling
  - parallelization
  - data distribution
  - load balancing
  - fault tolerance

- *Allows one to process huge amounts of data (terabytes and petabytes) on thousands of processors*

MapReduce Functionality



- Users implement interface of two primary methods:
  - 1. Map: <key1, value1> → <key2, value 2>
  - 2. Reduce: <key2, value2[ ]> → <value3>
- After map phase, all the intermediate values for a given output key are combined together into a list and given to a reducer for aggregating/merging the result.



MapReduce Process Level Overview

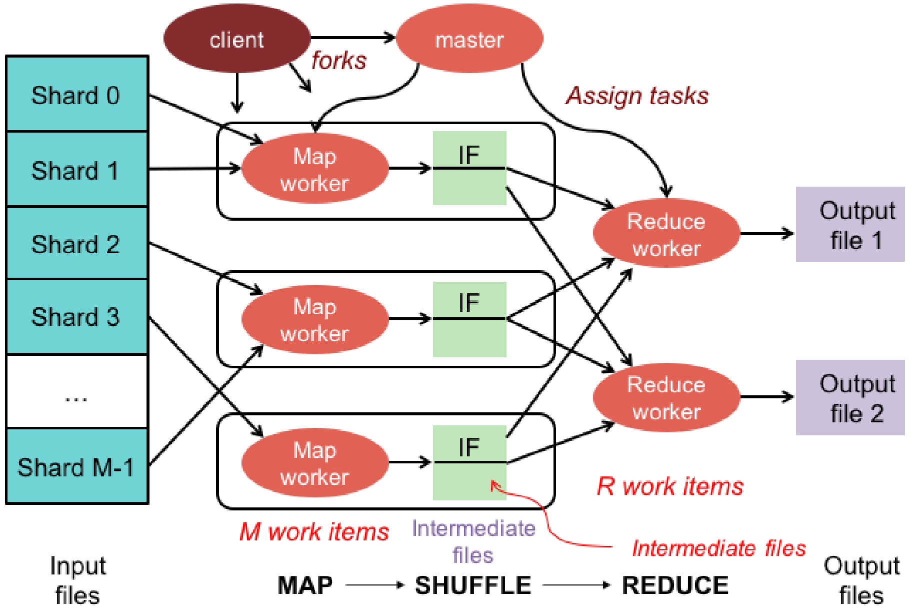
Step 1: Split input files into chunks (shards)

Break up the input data into *M* pieces (typically 64 MB)

Shard 0	Shard 1	Shard 2	Shard 3	…	Shard M-1

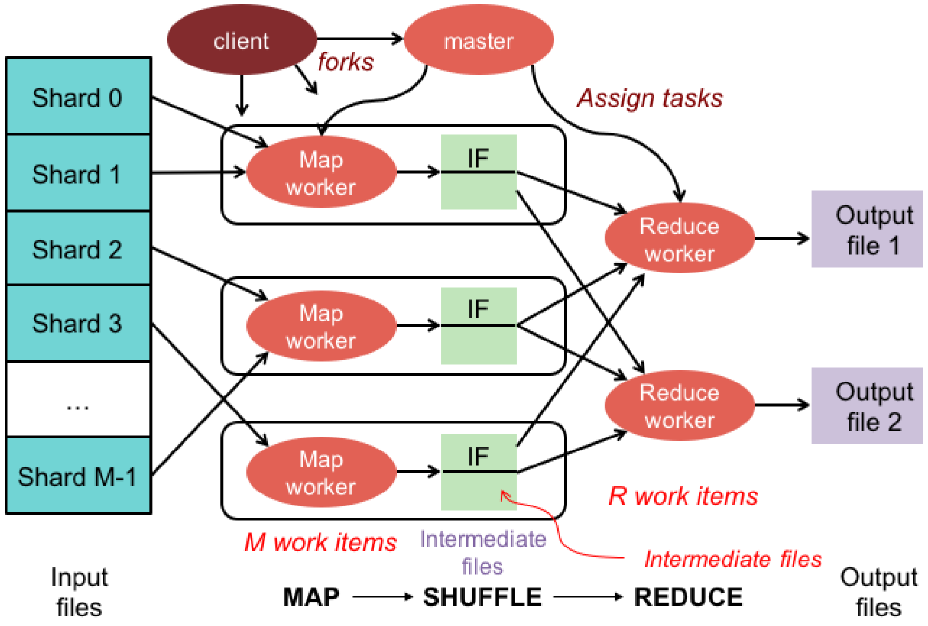
Input files

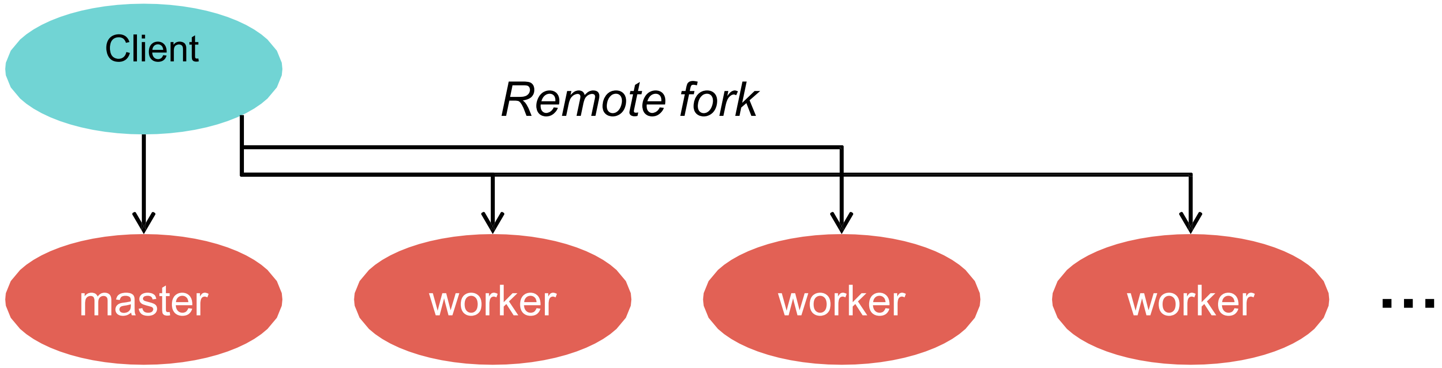
Divided into *M chunks*/shards



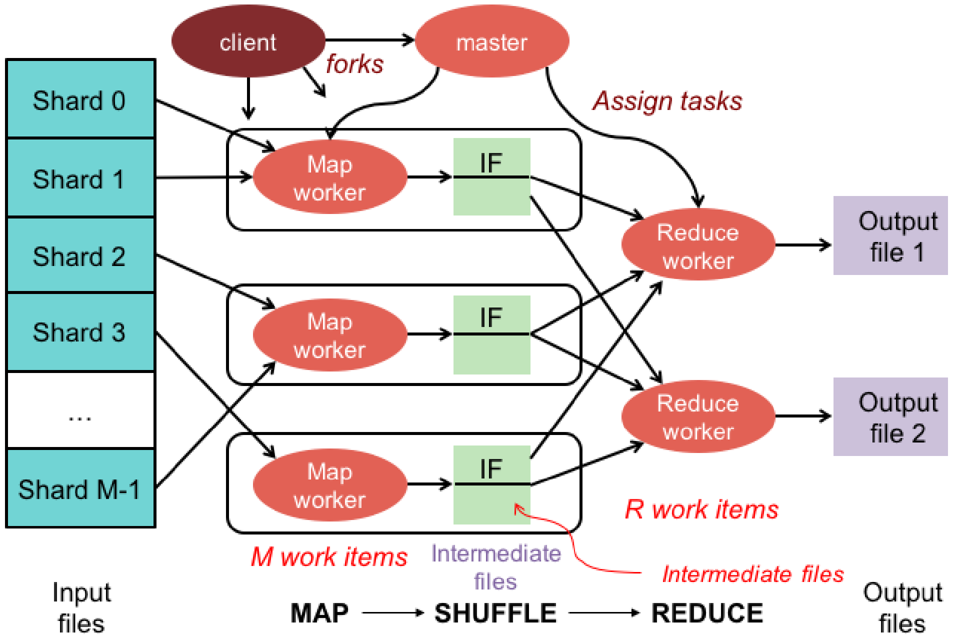
Step 2: Fork processes

- Start up many copies of the program on a cluster of machines
  - **One master**: scheduler & coordinator
  - Lots of workers
- Idle workers are assigned either:
  - map tasks (each works on a shard) – there are *M* map tasks
  - reduce tasks (each works on intermediate files) – there are *R* tasks
    - *R* = # partitions, defined by the user

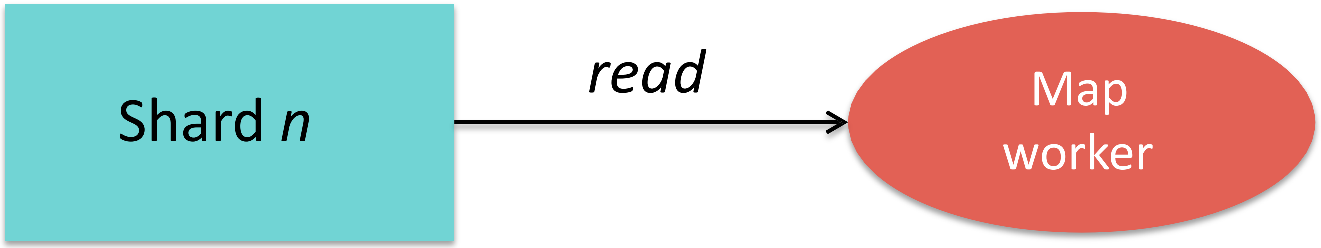




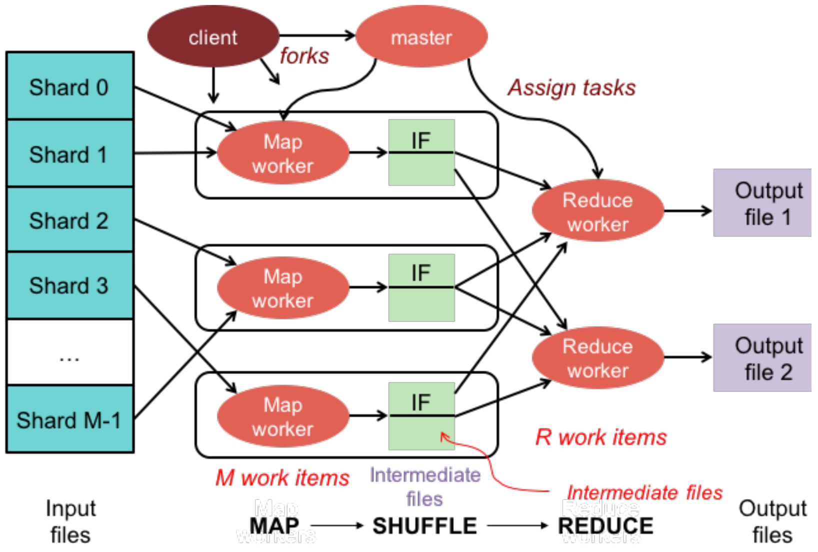
Step 3: Run Map Tasks



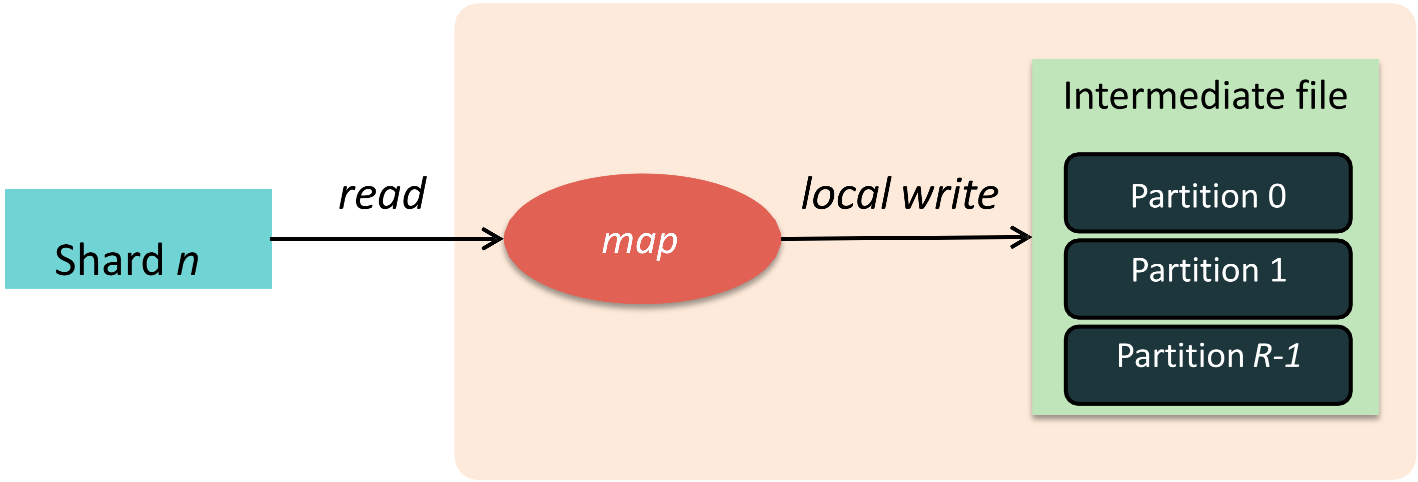
- Reads contents of the input shard assigned to it
- Parses key/value pairs out of the input data
- Passes each pair to a user-defined *map* function
  - Produces intermediate key/value pairs
  - These are buffered in memory



Step 4: Create intermediate files



- Intermediate key/value pairs produced by the user's *map* function buffered in memory and are periodically written to the local disk
  - Partitioned into *R* regions by a partitioning function
- Notifies master when complete
  - Passes locations of intermediate data to the master
  - Master forwards these locations to the reduce worker



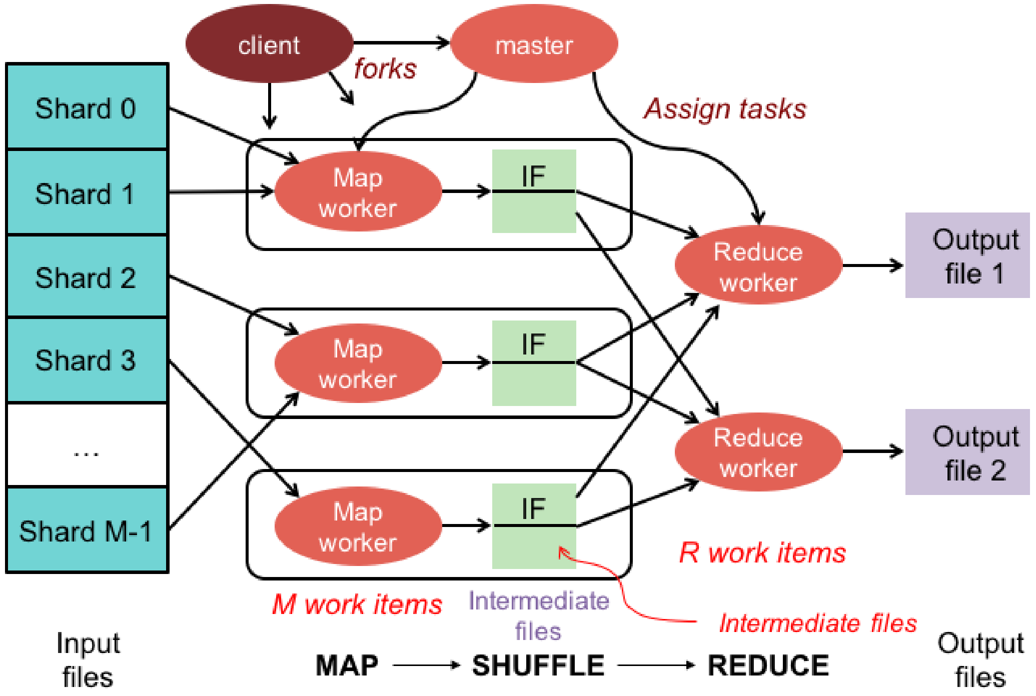
Step 4a. Partitioning

- Map data will be processed by Reduce workers
  - User's *Reduce* function will be called once per unique key generated by *Map*.

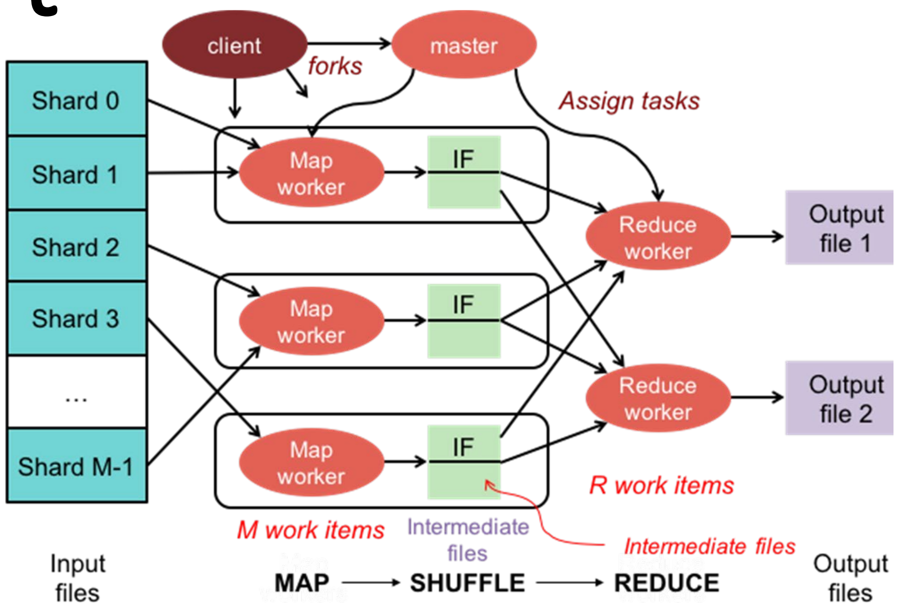
- We first need to group all the (*key, value*) data by keys and decide which Reduce worker processes which keys
  - The Reduce worker will later sort the values within the keys

- **Partition function**
  Decides which of *R* reduce workers will work on which key
  - Default function: *hash(key) mod R*
  - Map worker partitions the data by keys

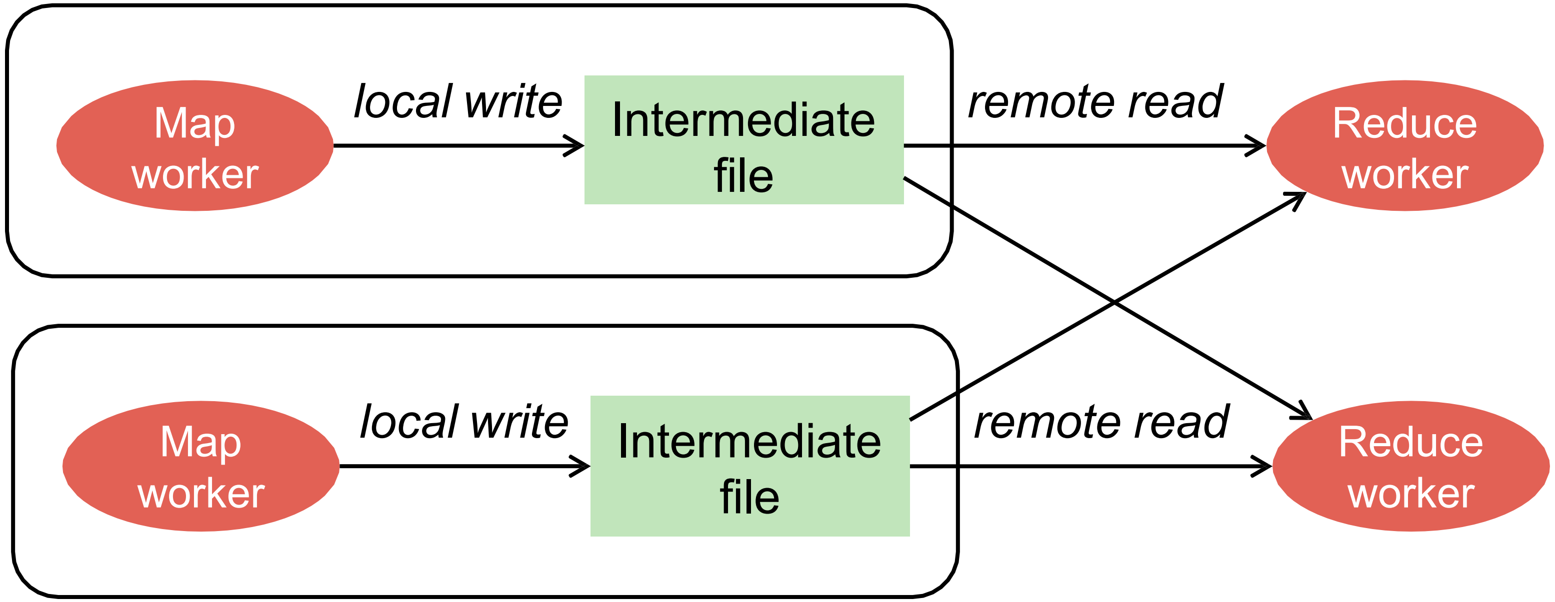
- Each Reduce worker will later read their partition from every Map worker



Step 5: Reduce Task: Shuffle & Sort



- Reduce worker gets notified by the master about the location of intermediate files for its partition
- Shuffle: Uses RPCs to read the data from the local disks of the map workers
- Sort: When the *reduce* worker gets all the (*key, value*) data for its partition from all workers
  - It sorts the data by the intermediate keys
  - All occurrences of the same key are grouped together

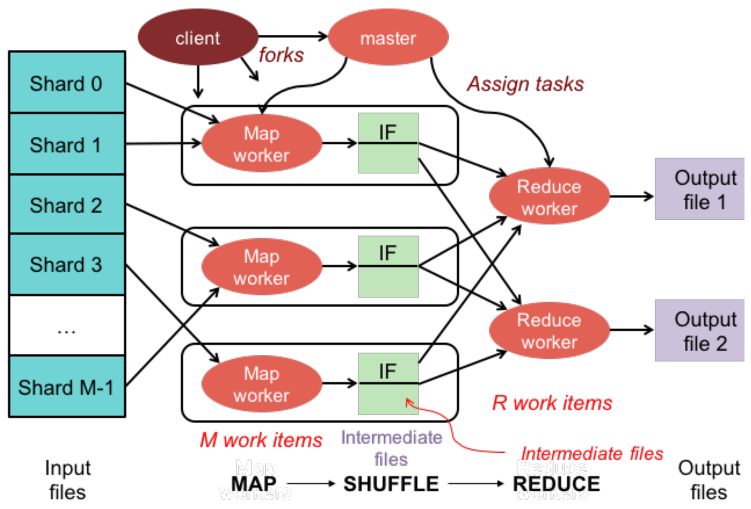


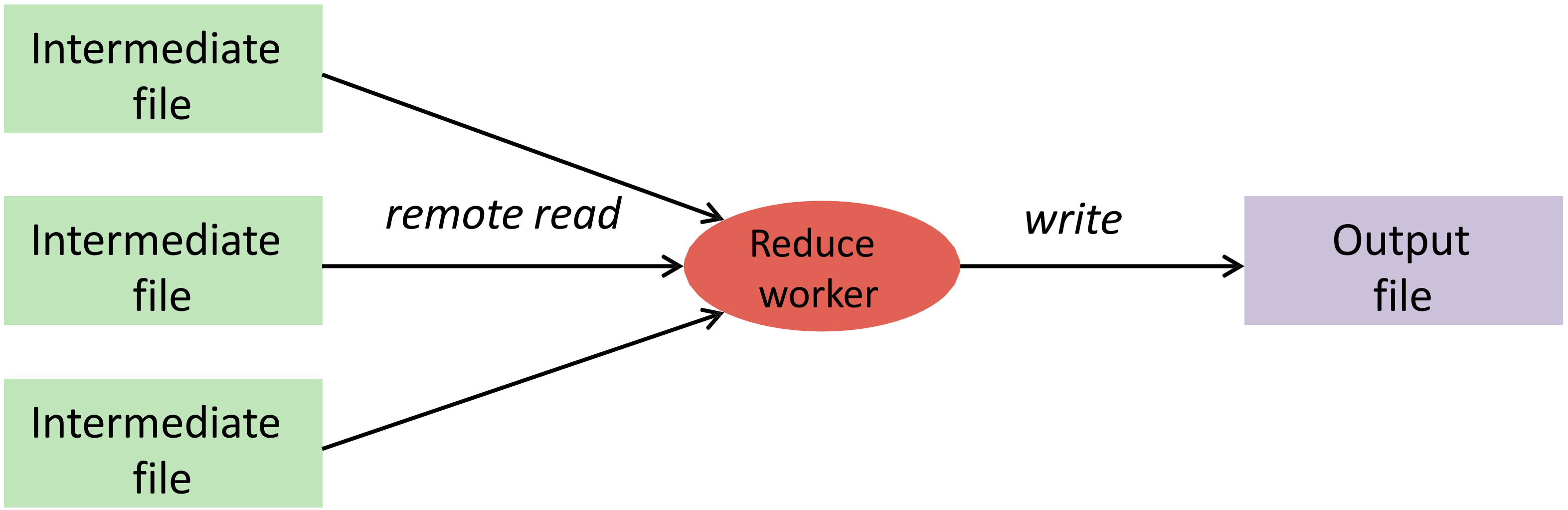
Step 6: Reduce Task

- The sort phase grouped data by keys
- User's ***Reduce*** function is given the key and the set of intermediate values for that key

  < key, (value1, value2, value3, value4, …) >

- The output of the *Reduce* function is appended to an output file

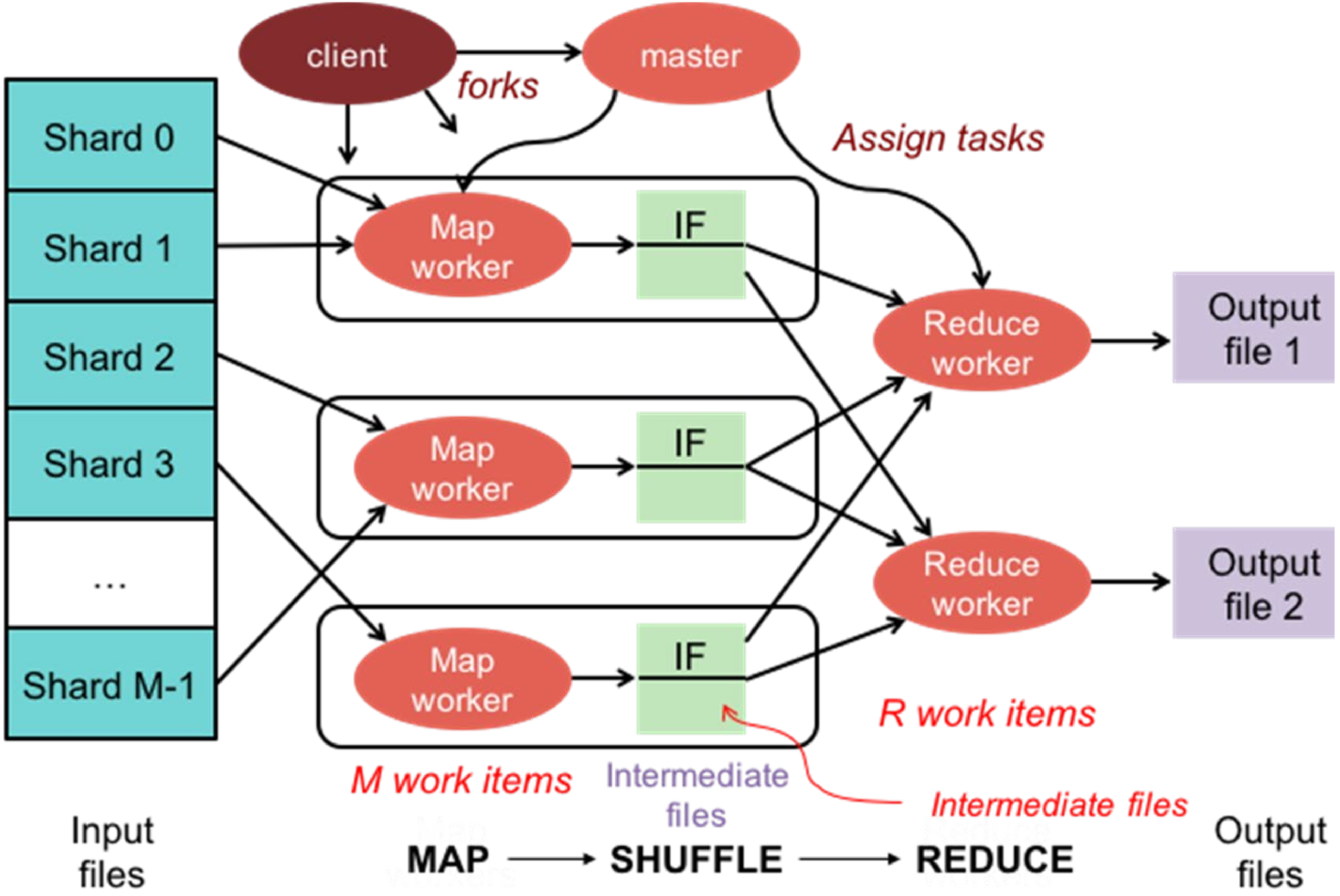




Step 7: Return to user

- When all *map* and *reduce* tasks have completed, the master wakes up the user program

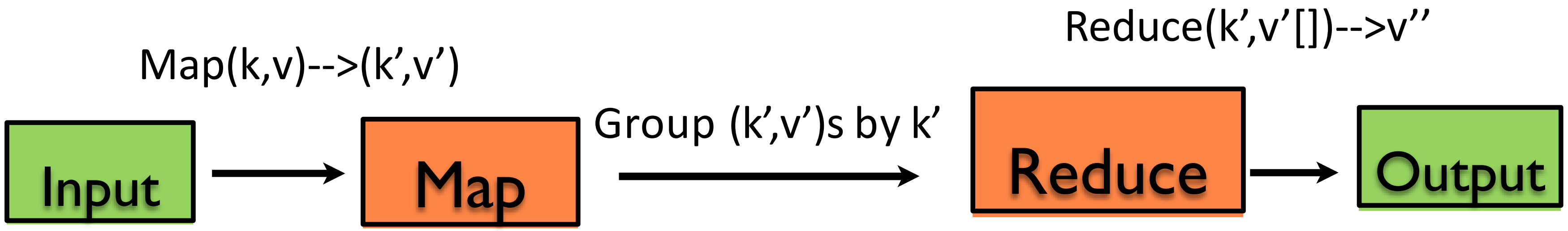
- The *MapReduce* call in the user program returns and the program can resume execution.
- Output of *MapReduce* is available in *R* output files


client
master
forks
Assign tasks
Shard 0
Shard 1
Shard 2
Shard 3
...
Shard M-1
Map worker
Map worker
Map worker
IF
IF
IF
Reduce worker
Reduce worker
Output file 1
Output file 2
R work items
M work items
Intermediate files
Intermediate files
Input files
MAP → SHUFFLE → REDUCE
Output files

A Simple

- Word count is challenging over massive amounts of data
- Fundamentals of statistics often are aggregate functions
- Most aggregation functions have distributive nature
- MapReduce breaks complex tasks into smaller pieces which can be executed in parallel

MapReduce Programming Model



- Users implement interface of two primary methods:
  - 1. Map: <key1, value1> → <key2, value 2>
  - 2. Reduce: <key2, value2[ ]> → <value3>

- After map phase, all the intermediate values for a given output key are combined together into a list and given to a reducer for aggregating/merging the result.

Map Phase

- Input to the Mapper

```
(3414, 'the cat sat on the mat')
(3437, 'the aardvark sat on the sofa')
```

Count the # of occurrences of each word in a large amount of input data

```
Map(input_key, input_value) {
  foreach word w in input_value:
    emit(w, 1);
}
```

- Output from the Mapper

```
('the', 1), ('cat', 1), ('sat', 1), ('on', 1),
('the', 1), ('mat', 1), ('the', 1), ('aardvark', 1),
('sat', 1), ('on', 1), ('the', 1), ('sofa', 1)
```

Shuffling/Sorting

- After the Map, all the intermediate values for a given intermediate key are combined together into a list

Mapper Output

```
('the', 1),
('cat', 1),
('sat', 1),
('on', 1),
('the', 1),
('mat', 1),
('the', 1),
('aardvark', 1),
('sat', 1),
('on', 1),
('the', 1),
('sofa', 1)
```

Reducer Input

```
aardvark, 1
cat, 1
mat, 1
on [1, 1]
sat [1, 1]
sofa, 1
the [1, 1, 1, 1]
```

Reducer

- Input of the shuffling/sorting

```
('the', 1), ('cat', 1), ('sat', 1), ('on', 1),
('the', 1), ('mat', 1), ('the', 1), ('aardvark', 1),
('sat', 1), ('on', 1), ('the', 1), ('sofa', 1)
```

Reducer Input

```
aardvark, 1
cat, 1
mat, 1
on [1, 1]
sat [1, 1]
sofa, 1
the [1, 1, 1, 1]
```

Add up all the values associated with each intermediate key:

```
Reduce(output_key, intermediate_vals) {
  set count = 0;
  foreach v in intermediate_vals:
    count += v;
  emit(output_key, count);
}
```

- Output from the Reducer

```
('the', 4), ('sat', 2), ('on', 2), ('sofa', 1),
('mat', 1), ('cat', 1), ('aardvark', 1)
```

Map+ Reduce At a glance

Mapper Input

```
the cat sat on the mat
the aardvark sat on the sofa
```

Mapping

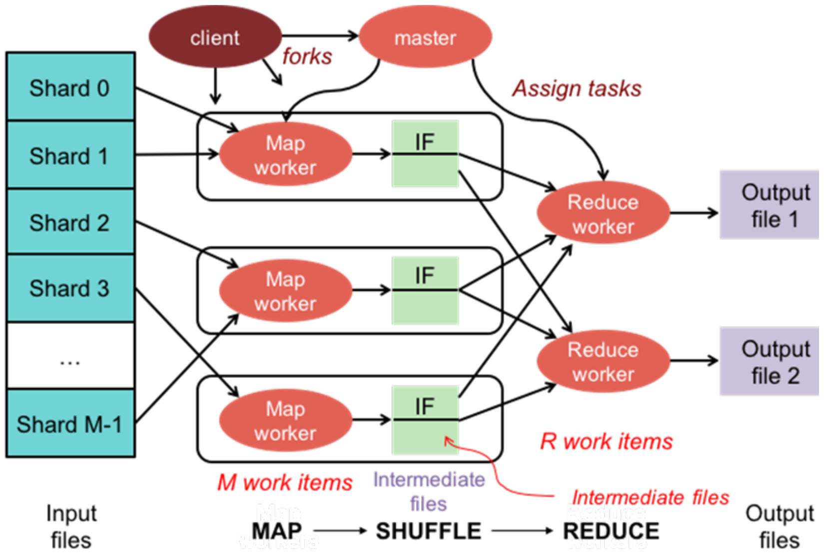
```
('the', 1),
('cat', 1),
('sat', 1),
('on', 1),
('the', 1),
('mat', 1),
('the', 1),
('aardvark', 1),
('sat', 1),
('on', 1),
('the', 1),
('sofa', 1)
```

Shuffle/ Sorting

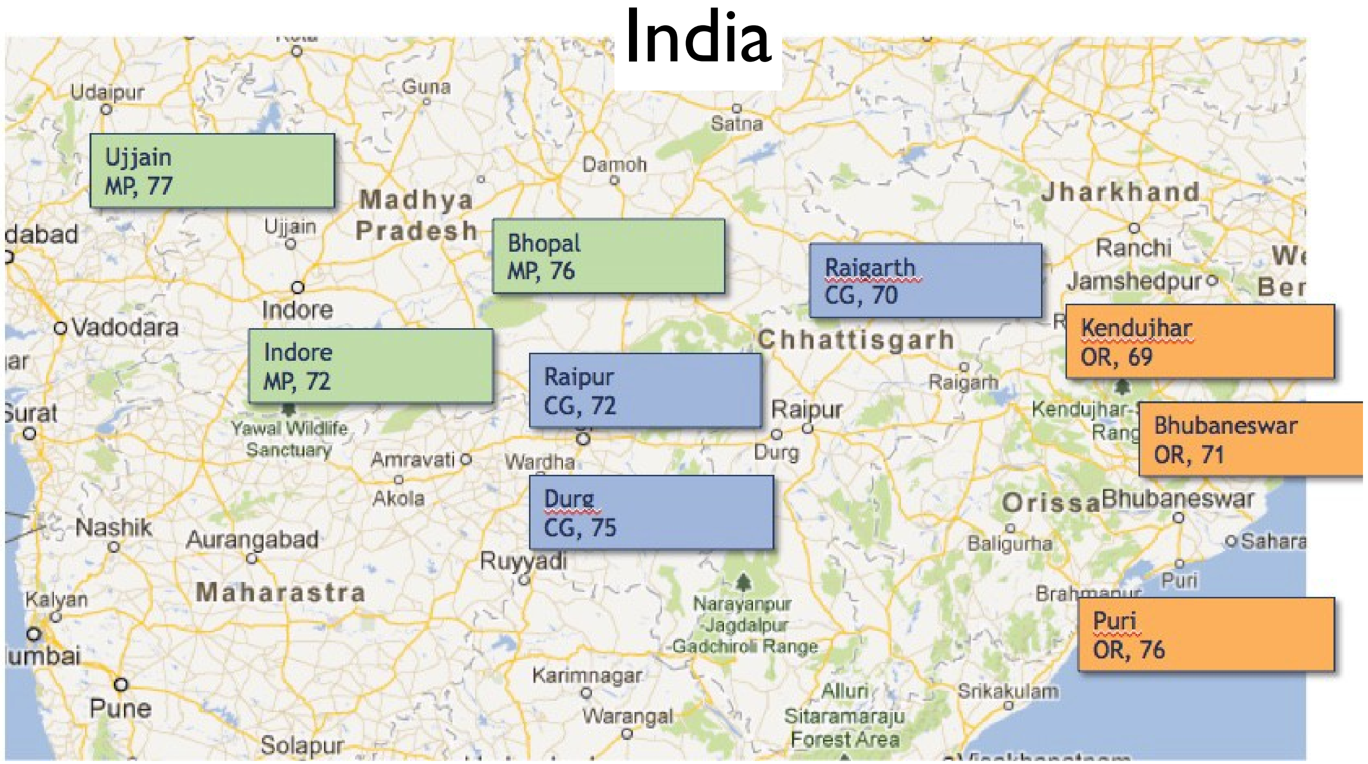
```
aardvark, 1
cat, 1
mat, 1
on [1, 1]
sat [1, 1]
sofa, 1
the [1, 1, 1, 1]
```

Reducing

```
aardvark, 1
cat, 1
mat, 1
on, 2
sat, 2
sofa, 1
the, 4
```

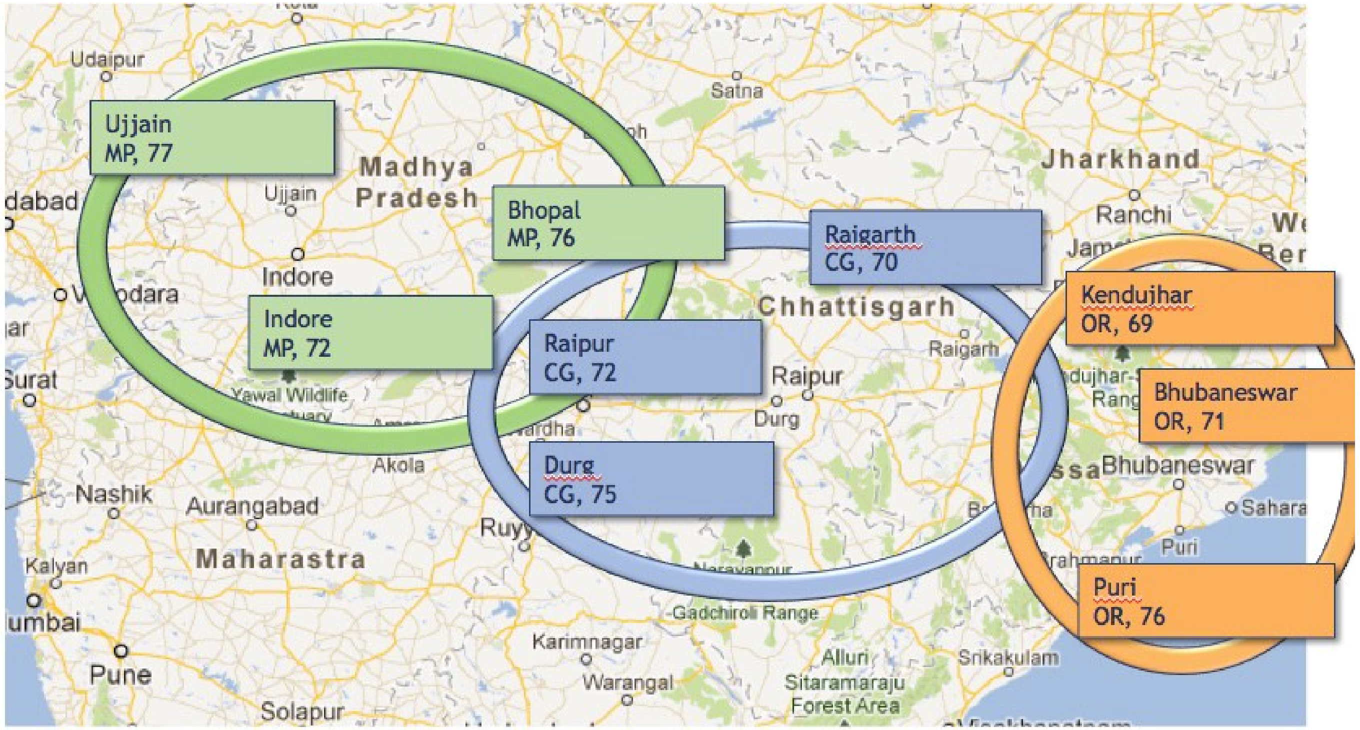


Let’s use MapReduce to help Google Map



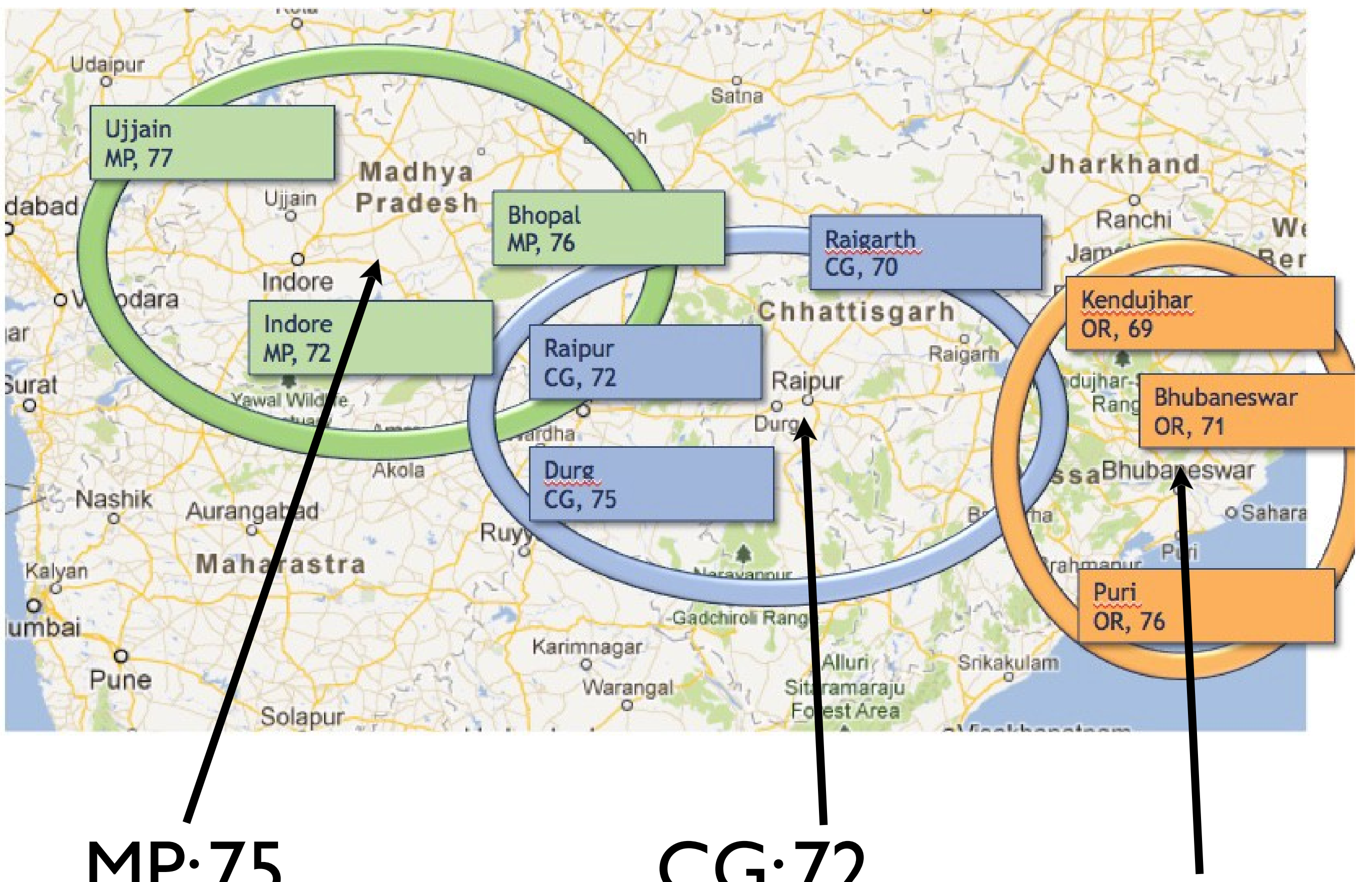
We want to compute the average temperature for each state

Let's use MapReduce to help Google Map



We want to compute the average temperature for each state

Let's use MapReduce to help Google Map

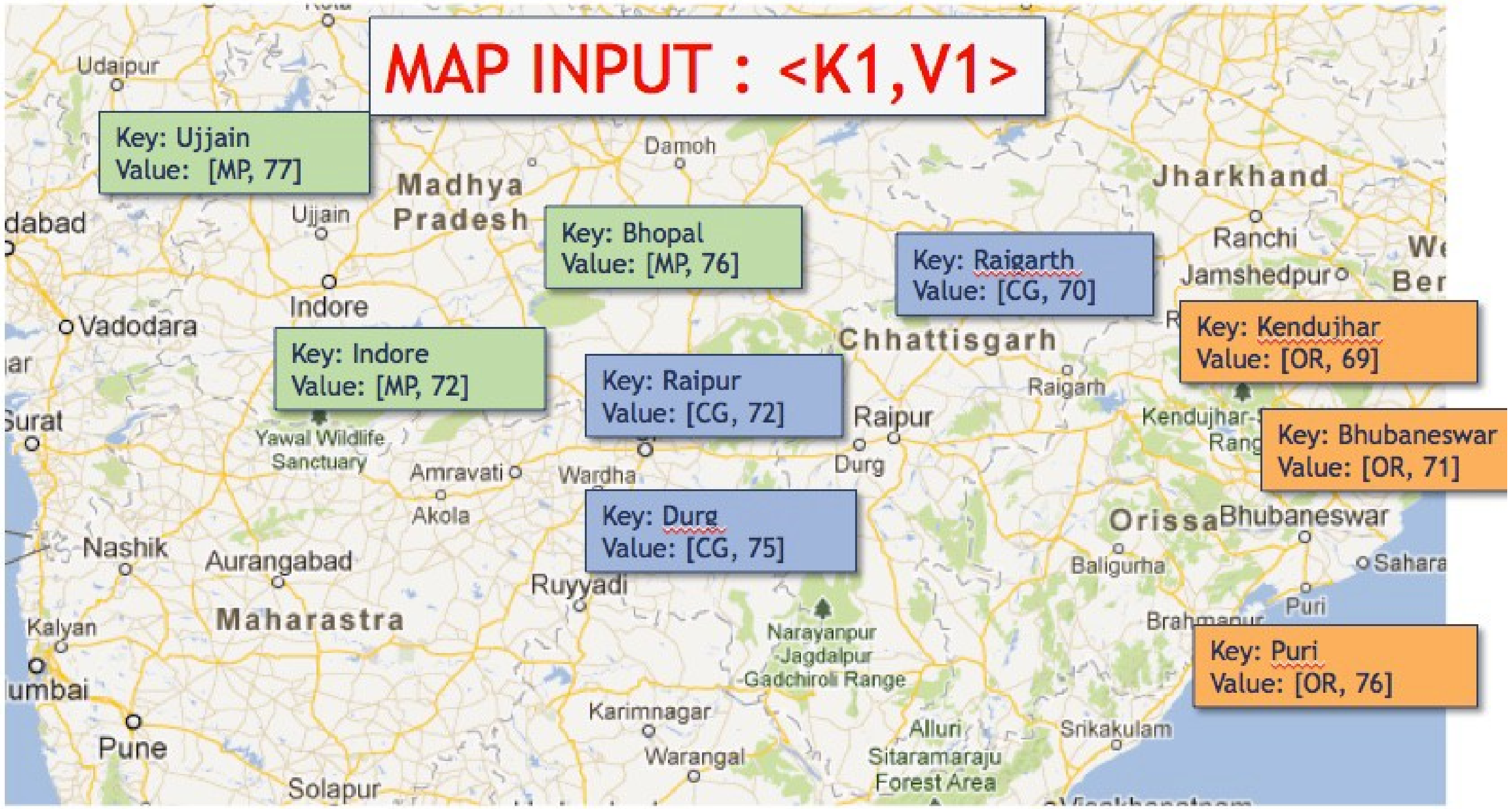


MP:75

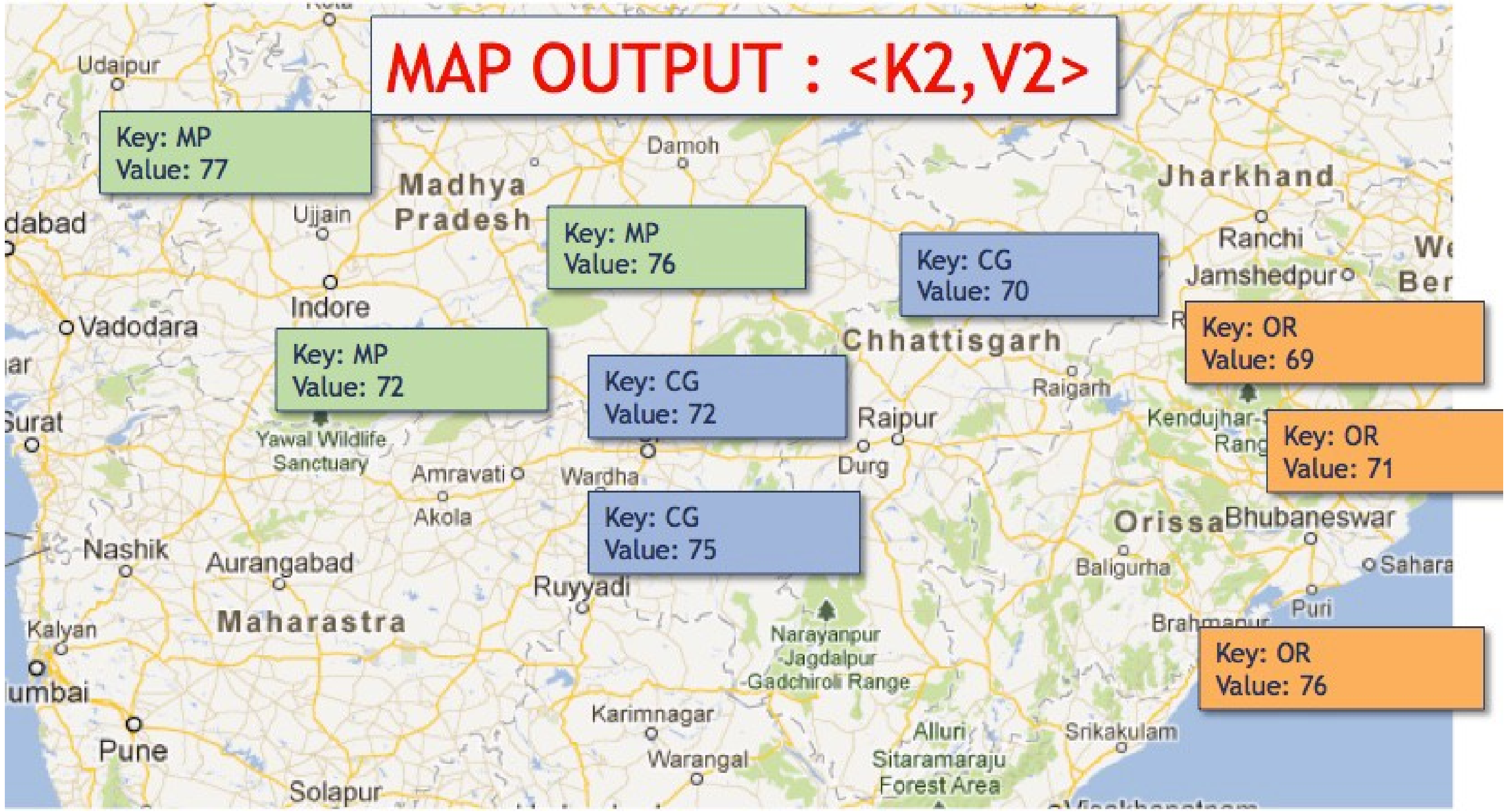
CG:72

OR:72

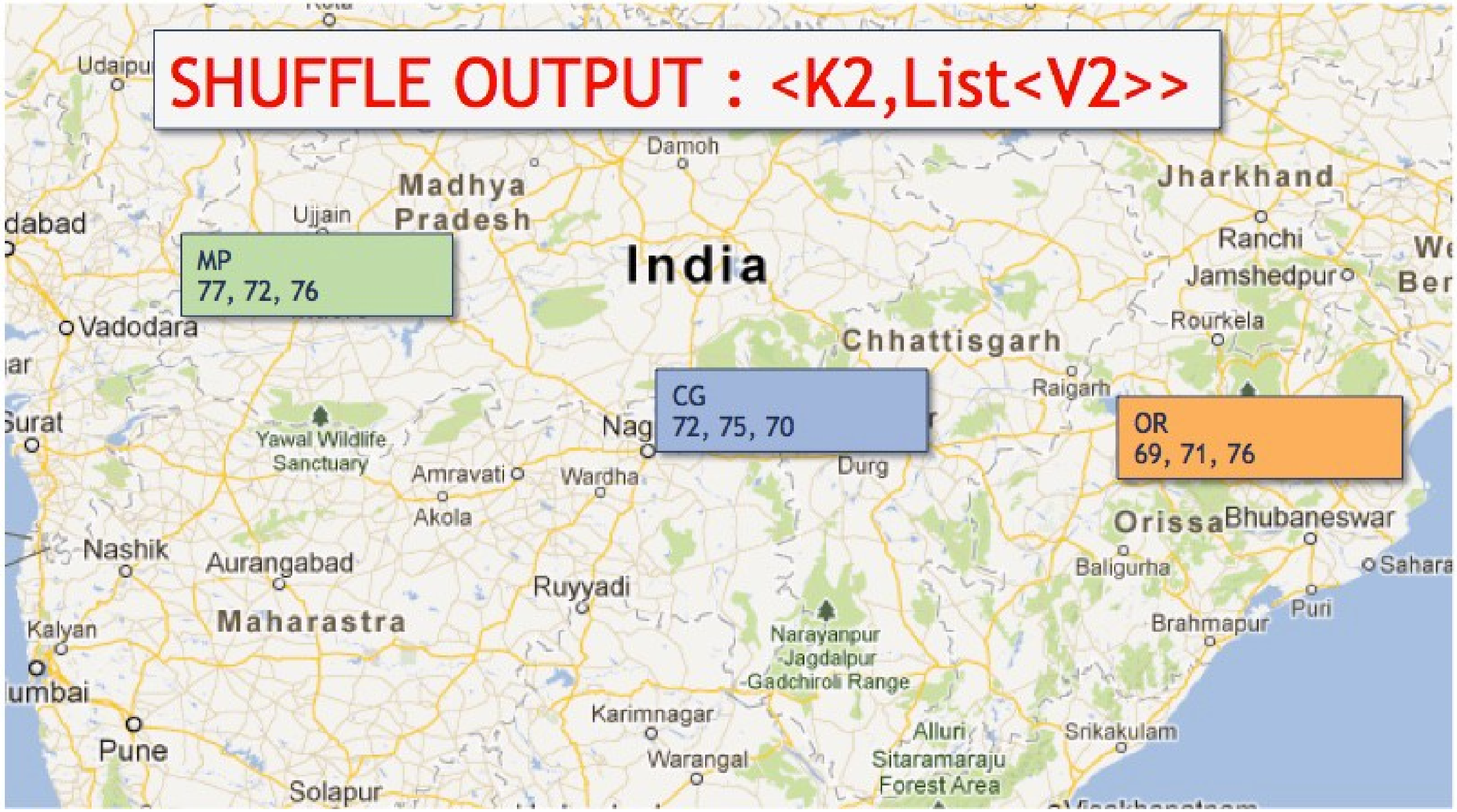
Let's use MapReduce to help Google Map



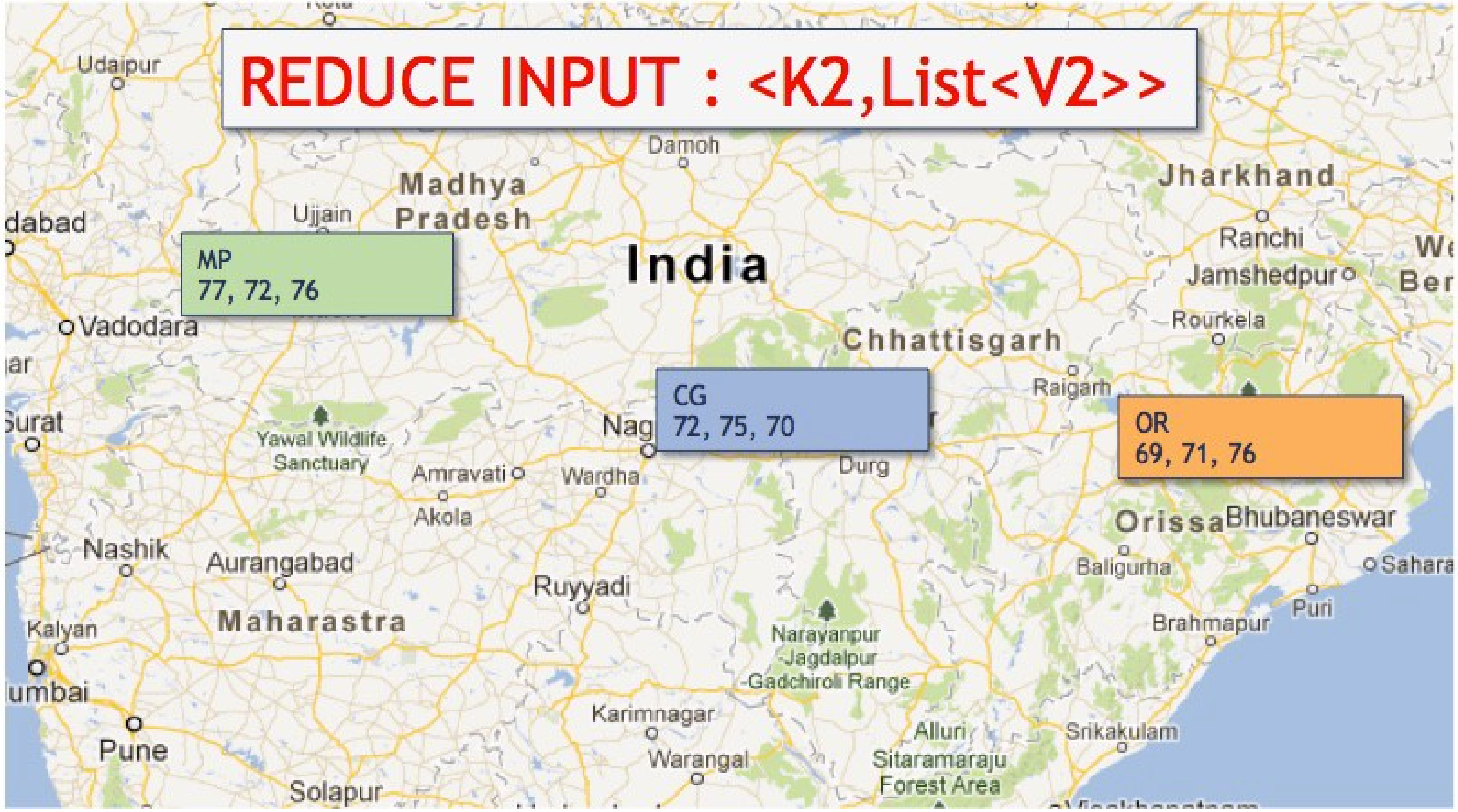
Let’s use MapReduce to help Google Map



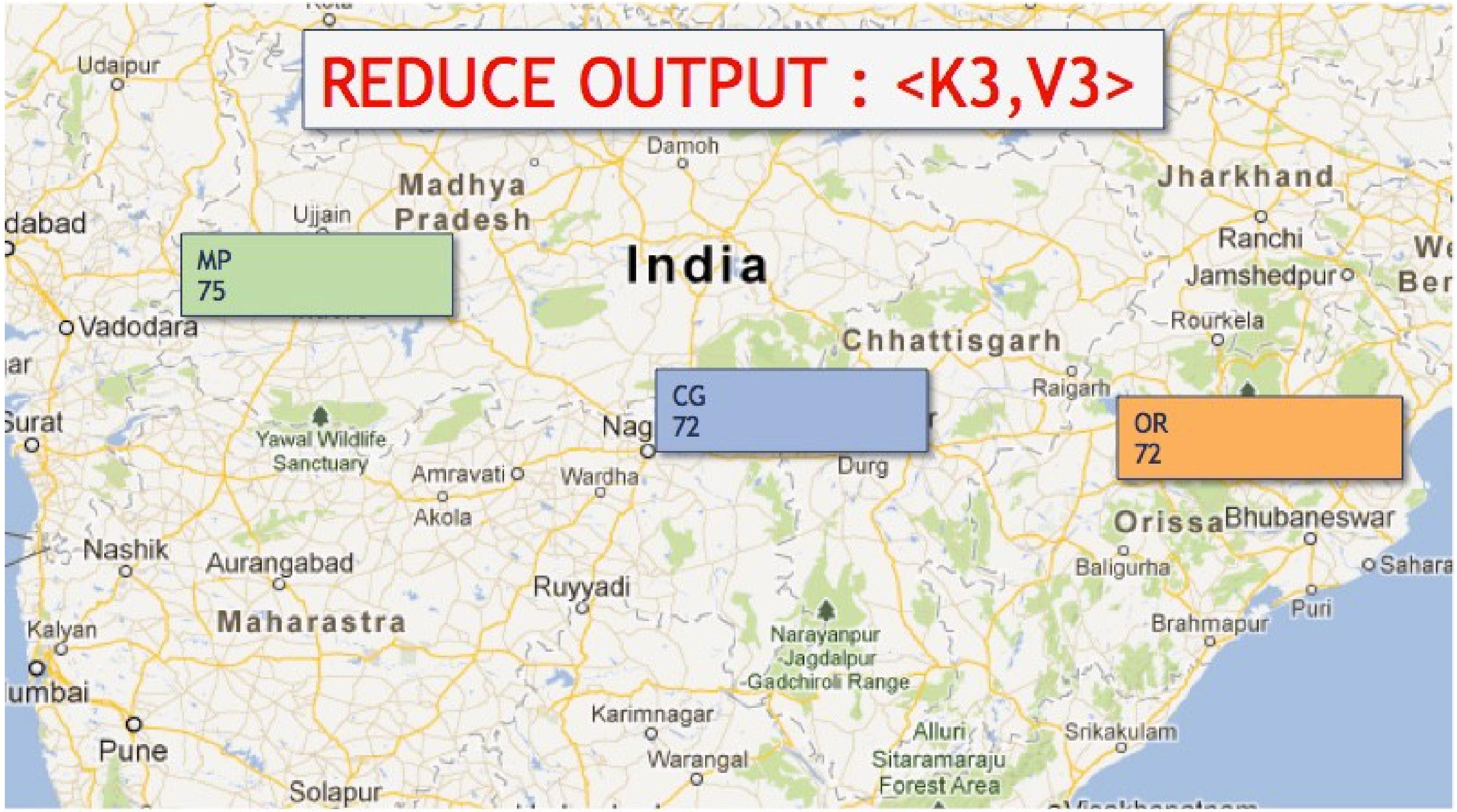
Let’s use MapReduce to help Google Map



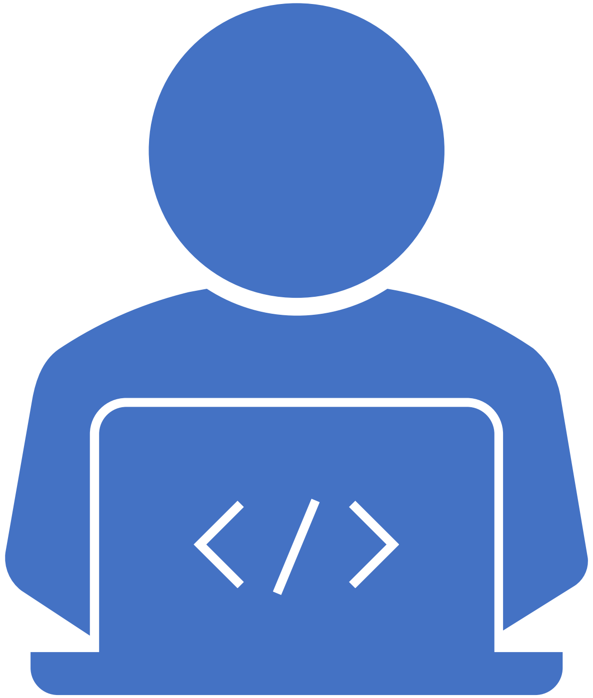
Let’s use MapReduce to help Google Map



Let's use MapReduce to help Google Map



Questions?



Paul Krzyzanowski, Rutgers University

Ennan Zhai, Computer Science Department, Yale University

Slides Acknowledgement