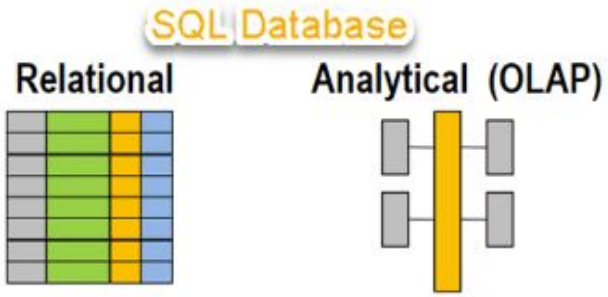
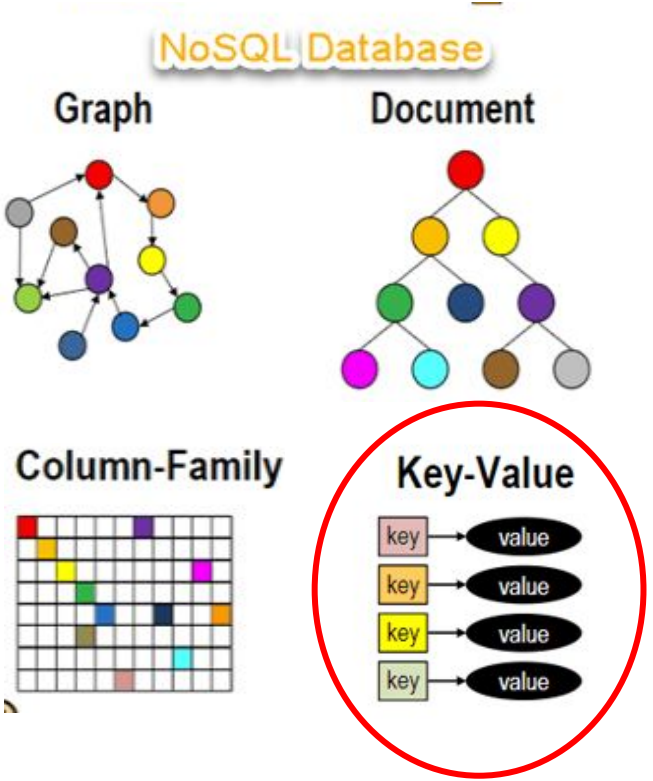
Assignment 8 NoSQL

Computing Lab - II (CS69012)
IIT Kharagpur

What is NoSQL?

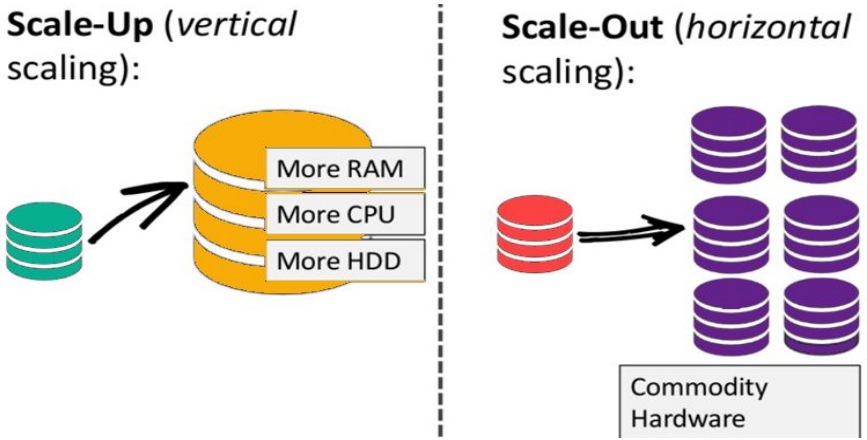
- "Not Only SQL" or "Not SQL" is a non-relational Database Management System, that does not require a fixed schema, avoids joins, and is easy to scale
- Used for distributed data stores with humongous data storage needs (Big data and real-time web apps)
- Can store structured, semi-structured, unstructured and polymorphic data





Why NoSQL?

- System response time becomes slow when RDBMS is used for massive volumes of data
- To resolve this problem, we could "scale up" our systems by upgrading existing hardware, which is expensive
- The alternative for this problem is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out"
- NoSQL database is non-relational, so it scales out better than relational databases



Understanding Map-Reduce

Map:

- Grab the relevant data from the source.
- User function gets called for each chunk of input.
- Spits out (key, value) pairs.

Reduce:

- Aggregate the results.
- User function gets called for each unique key with all values corresponding to that key.

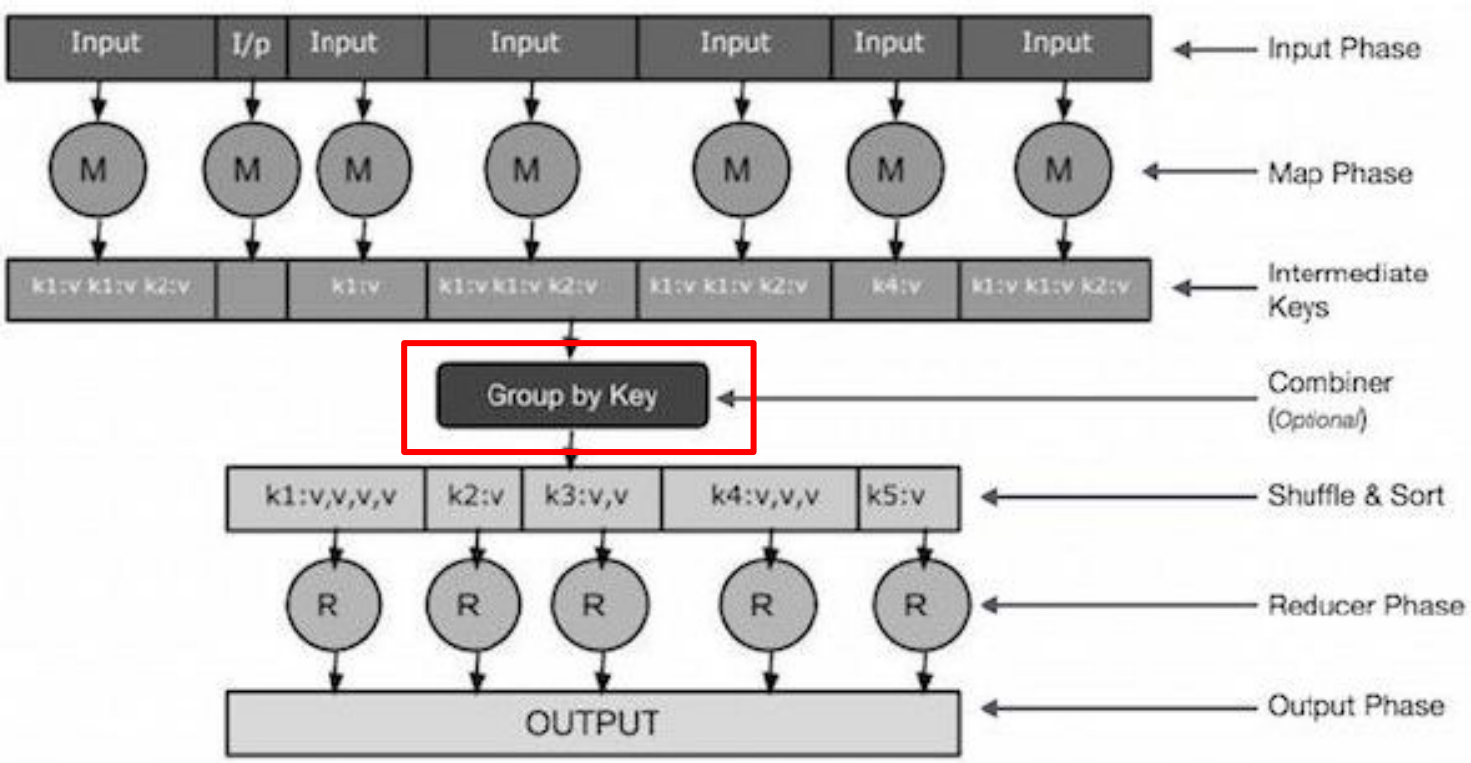
Map-Reduce: What happens in between?

- Map
  - Grab the relevant data from the source (parse into key, value)
  - Write it to an intermediate file
- Partition
  - Partitioning: identify which of *R* reducers will handle which keys
  - Map partitions data to target it to one of *R* Reduce workers based on a partitioning function (both *R* and partitioning function user defined)

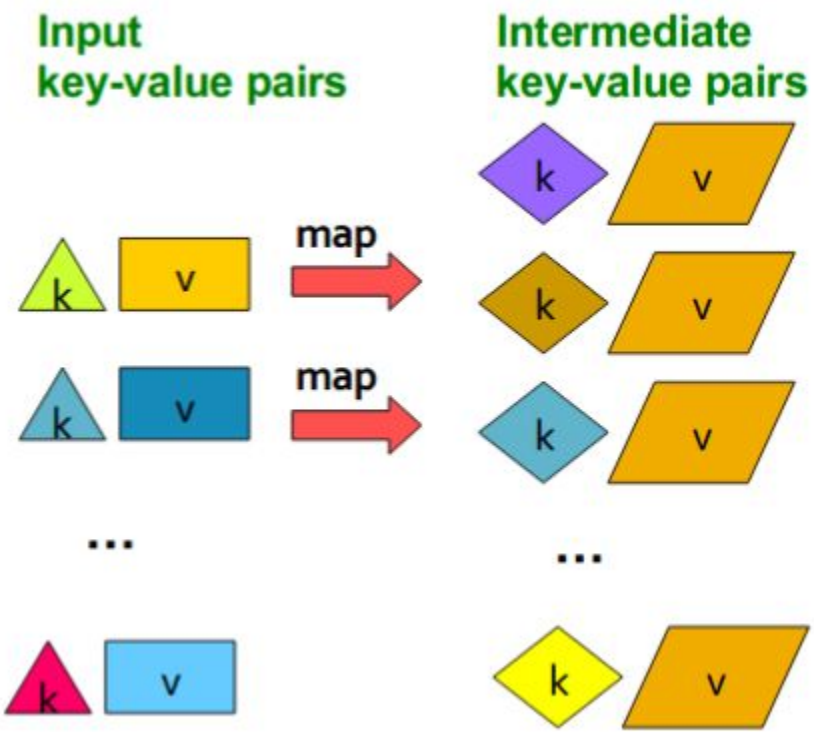
Map Worker

- Shuffle & Sort
  - Shuffle: Fetch the relevant partition of the output from all mappers
  - Sort by keys (different mappers may have sent data with the same key)
- Reduce
  - Input is the sorted output of mappers
  - Call the user *Reduce* function per key with the list of values for that key to aggregate the results

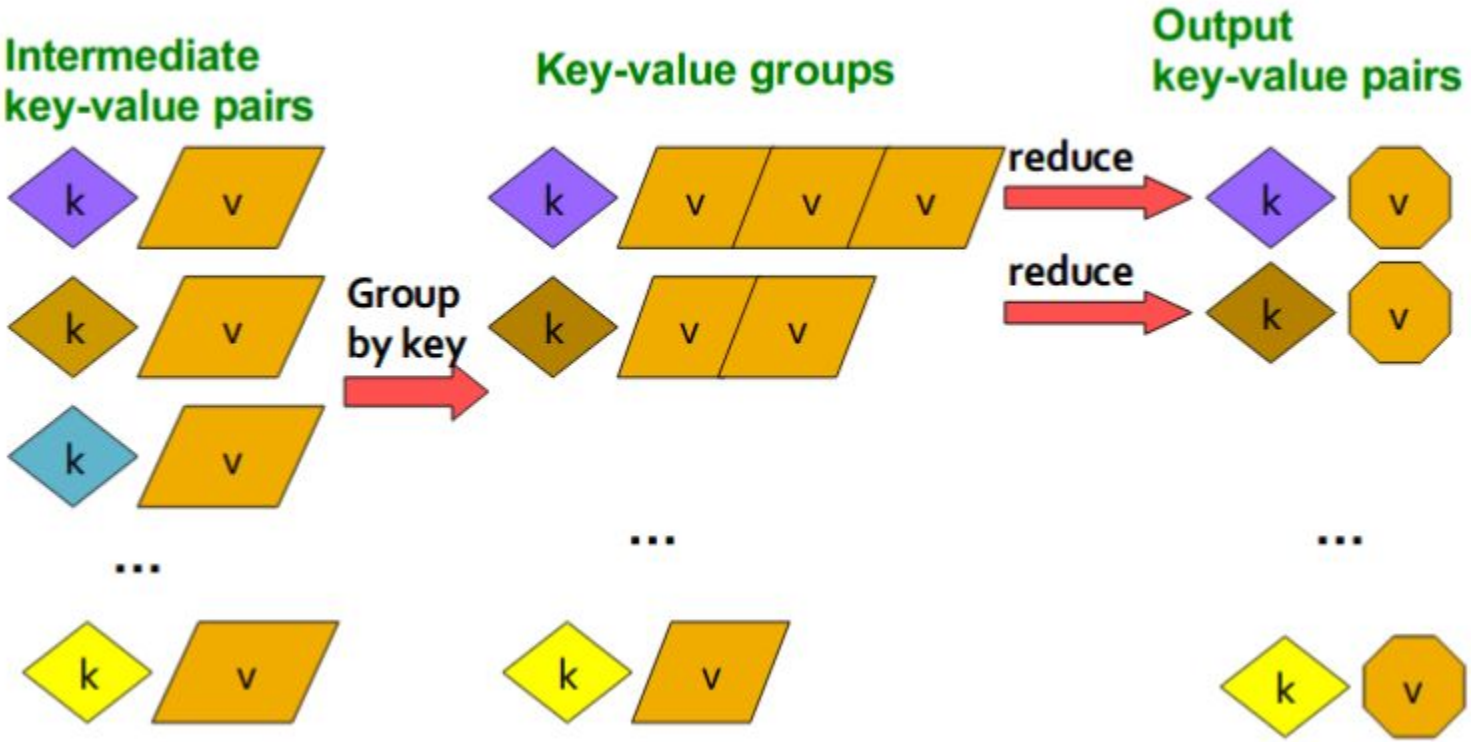
Reduce Worker

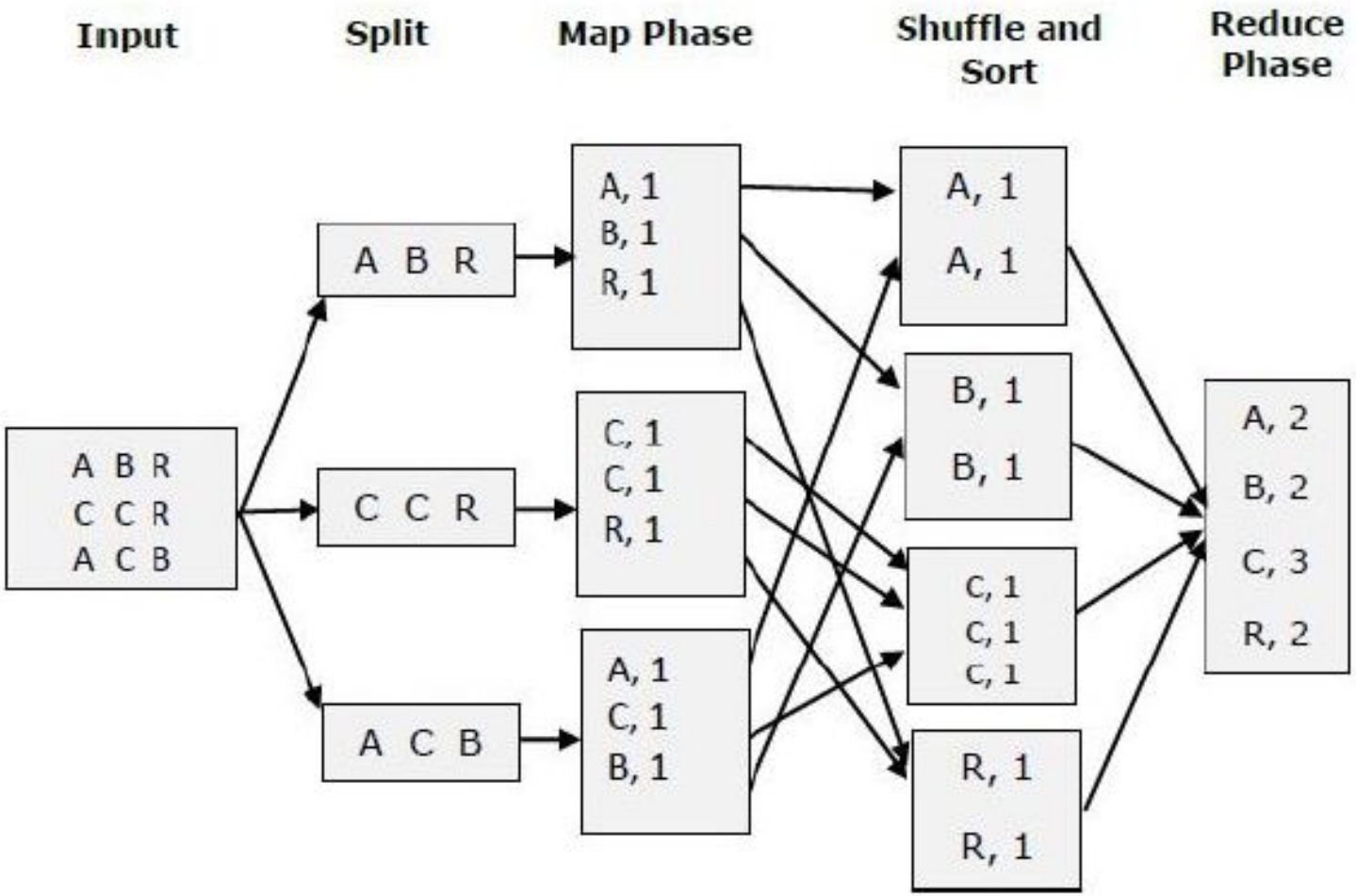

Input
I/p
Input
Input
Input
Input
Input
Input Phase
M
Map Phase
k1:v k1:v k2:v
k1:v
k1:v k1:v k2:v
k4:v
Intermediate Keys
Group by Key
Combiner
(Optional)
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
Shuffle & Sort
R
Reducer Phase
OUTPUT
Output Phase

Mapper Phase

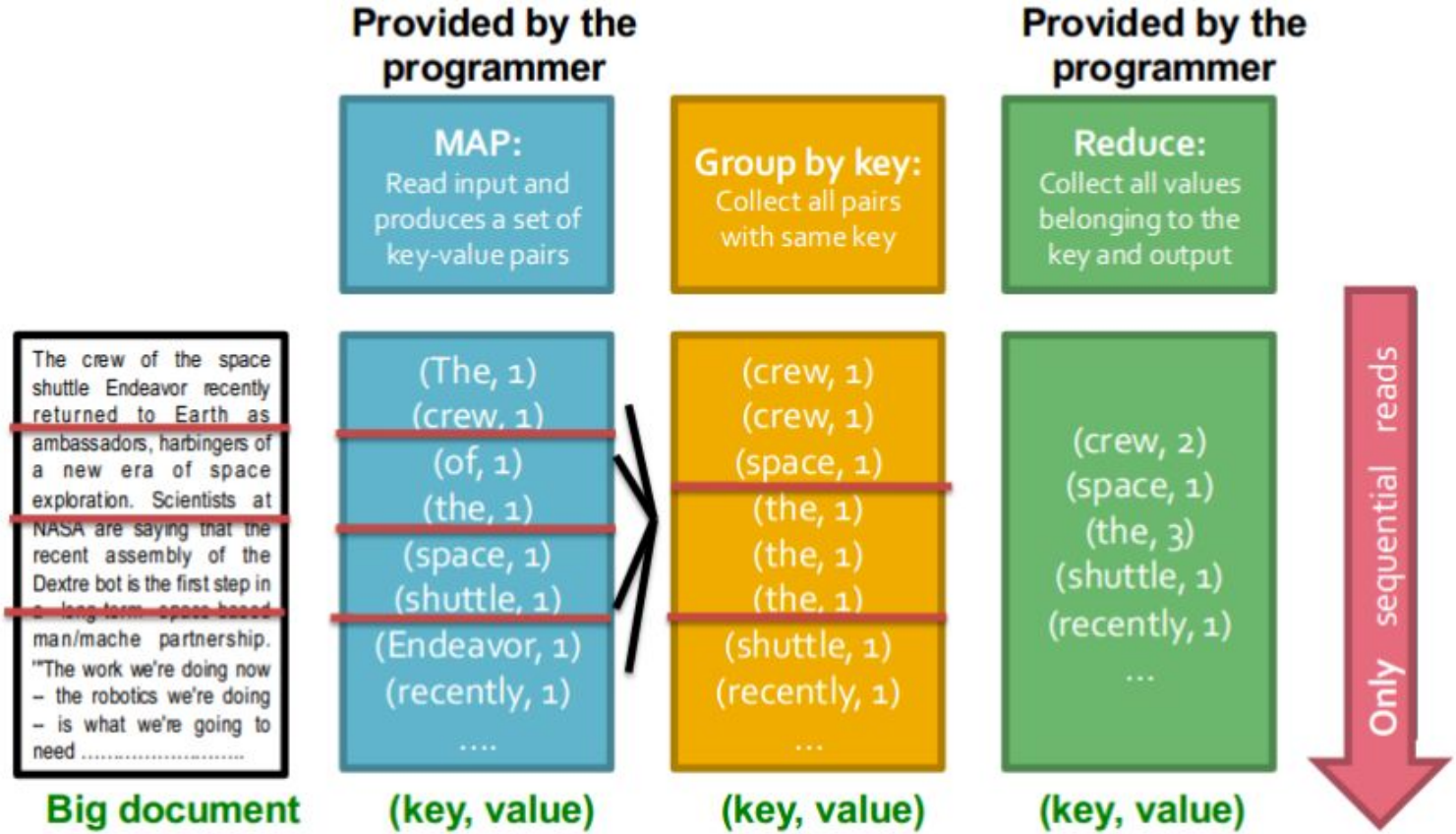


Reducer Phase




Input
Split
Map Phase
Shuffle and Sort
Reduce Phase
A B R
C C R
A C B
A B R
C C R
A C B
A, 1
B, 1
R, 1
C, 1
C, 1
R, 1
A, 1
C, 1
B, 1
A, 1
A, 1
B, 1
B, 1
C, 1
C, 1
C, 1
R, 1
R, 1
A, 2
B, 2
C, 3
R, 2

Map-Reduce Example



Understanding the Assignment

Data Set:

- The dataset describes **reviews data** from Amazon.com.
- There are 2 files.
  - Reviews file
  - Items file

Data Set: Reviews file

- python dictionary objects
- Can be read using **eval()**


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano. .....",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```

Data Set: Items file

- python dictionary objects
- Can be read using **eval()**


```
{
  "asin": "0000031852",
  "title": "Girls Ballet Tutu Zebra Hot Pink",
  "description":"This is real vanilla extract made with only 3 premium ingredients. GMO free,.....",
  "price": 3.17,
  "imUrl": "http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
  "related":
  {
    "also_bought": ["B00JHONN1S"],
    "also_viewed": ["B002BZX8Z6"],
    "bought_together": ["B002BZX8Z6"]
  },
  "salesRank": {"Toys & Games": 211836},
  "brand": "Coxlures",
  "categories": [["Sports & Outdoors", "Other Sports", "Dance"]]
}
```

Sample Code (Word Count):

Mapper.py

```
import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Sort

Reducer.py

```
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        continue
    # this IF-switch only works because Hadoop sorts map
output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Example Query

- Find all the user ids who have rated at least n items.

➔ Write a script mapper.py which takes whole data into input and splits data into m chunks specified during runtime also mapper should transform chunk into key-value pairs. (here <user-id,item-id>)

➔ Print the key-value pairs and use sort function to arrange it.

➔ The sorted key-value pairs are then passed to the reducer to group by key and perform necessary actions.

➔ Here reducer should count the entries of unique items for each particular user and if it is greater than n, then it should print that user.

Since we do not have access to a hadoop cluster, we will be testing our codes on a linux system as follows:

```
cat input.json | python mapper.py | sort | python reducer.py
```

```
python mapper.py | sort | python reducer.py
```

Deliverables

- For each of the query, make a directory with name “Query<no.>”
- Each directory should contain:
  - mapper.py
  - reducer.py
  - result.txt
  - readme + makefile

Thank you.