



**MAPREDUCE
FUNDAMENTAL
CONCEPTS**



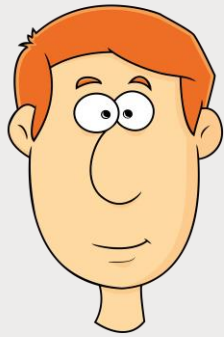
Why MapReduce?

- Distributes the processing of data on your cluster
- Divides your data up into partitions that are MAPPED (transformed) and REDUCED (aggregated) by mapper and reducer functions you define
- Resilient to failure - an application master monitors your mappers and reducers on each partition



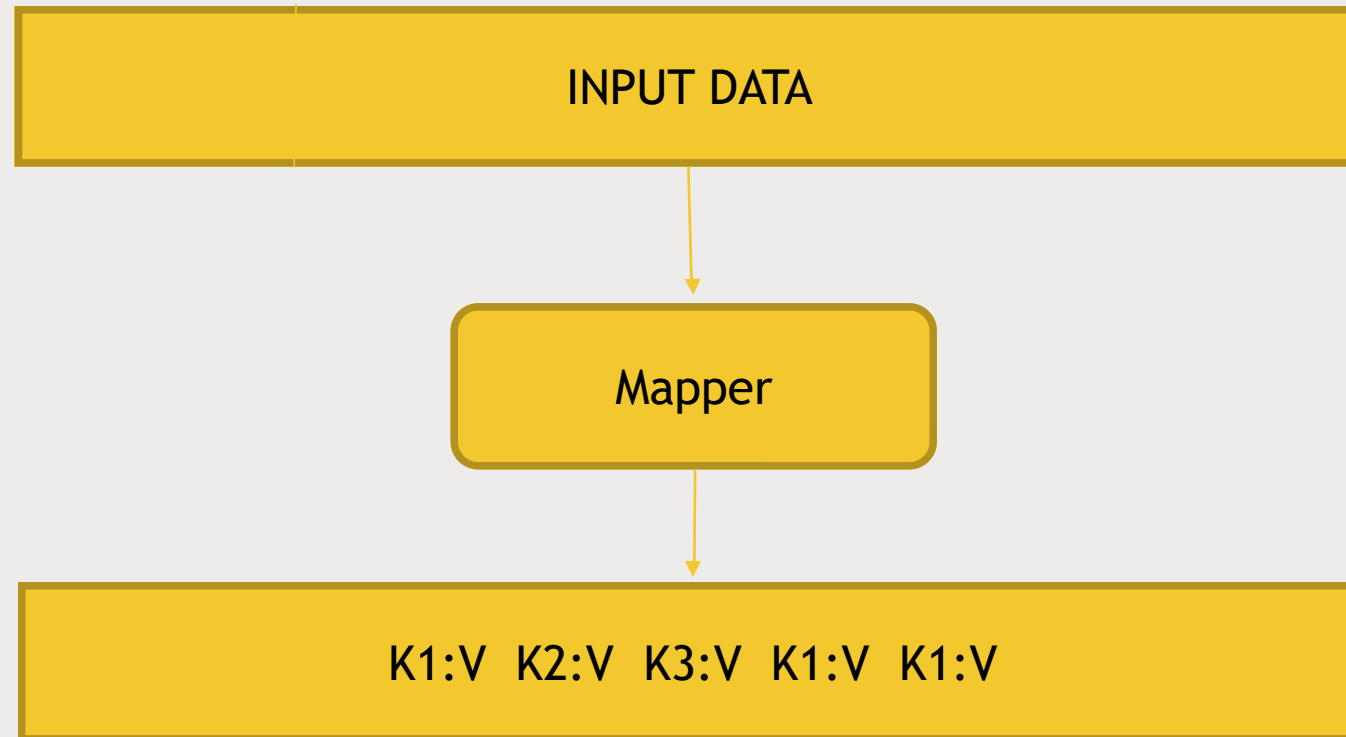
Let's illustrate with an example

- How many movies did each user rate in the MovieLens data set?



How MapReduce Works: Mapping

- The MAPPER converts raw source data into **key/value** pairs



Example: MovieLens Data (u.data file)

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

Map users to movies they watched

USER ID|MOVIE ID|RATING|TIMESTAMP

196 242 3 881250949

186 302 3 891717742

196 377 1 878887116

244 51 2 880606923

166 346 1 886397596

186 474 4 884182806

186 265 2 881171488



Mapper



196:242 186:302 196:377 244:51 166:346 186:274 186:265

Extract and Organize What We Care About

196:242 186:302 196:377 244:51 166:346 186:474 186:265



MapReduce Sorts and Groups the Mapped Data (“Shuffle and Sort”)

196:242 186:302 196:377 244:51 166:346 186:474 186:265



166:346 186:302,474,265 196:242,377 244:51

The REDUCER Processes Each Key's Values

166:346 186:302,474,265 196:242,377 244:51



len(movies)



166:1 186:3 196:2 244:1

Putting it All Together

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488



MAPPER



196:242 186:302 196:377 244:51 166:346 186:474 186:265



SHUFFLE AND SORT



166:346 186:302,474,265 196:242,377 244:51



REDUCER



166:1 186:3 196:2 244:1

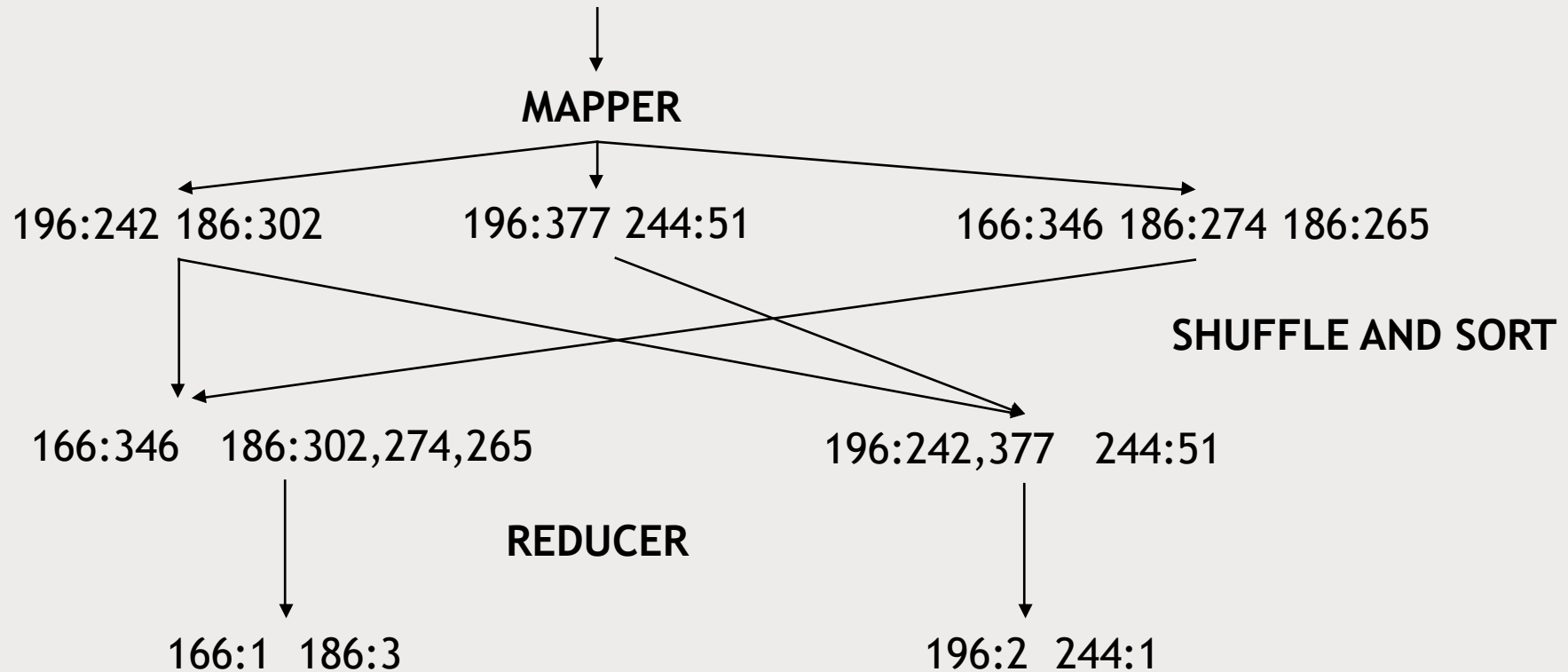


MAPREDUCE ON A CLUSTER

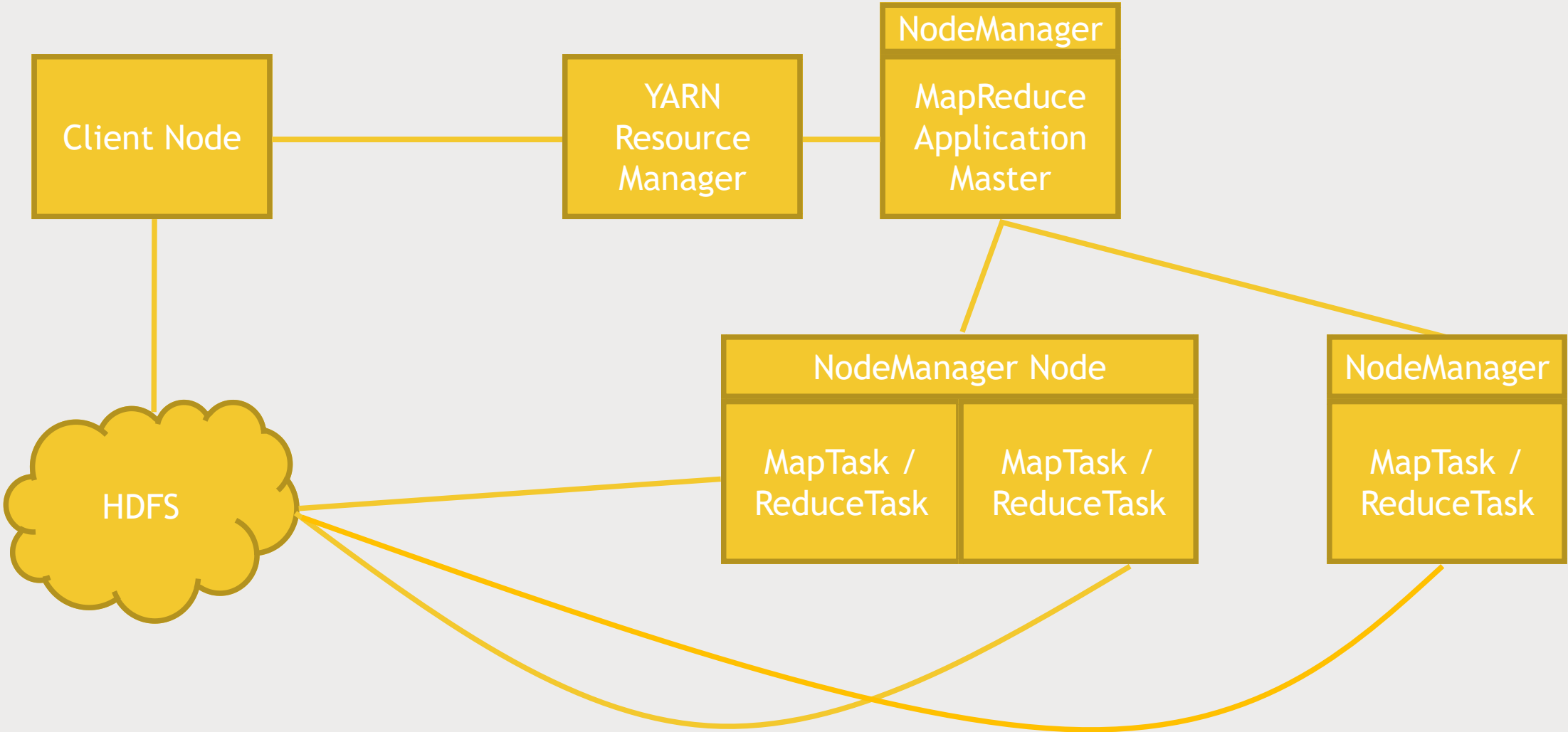
How MapReduce Scales

Putting it All Together

USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

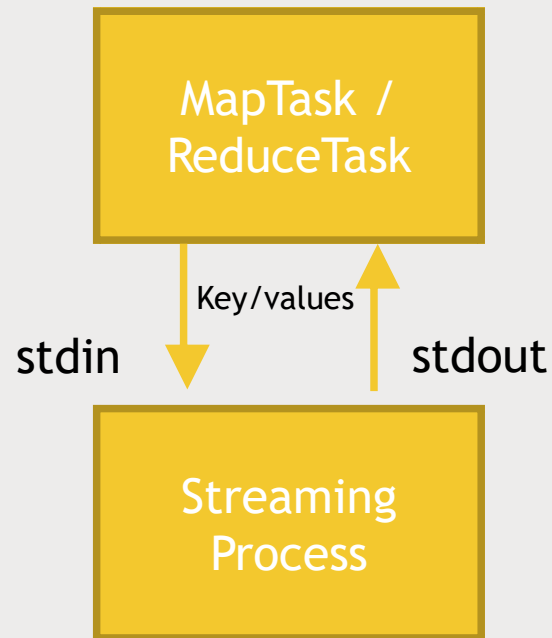


What's Happening



How are mappers and reducers written?

- MapReduce is natively Java
- STREAMING allows interfacing to other languages (ie Python)




Handling Failure



- Application master monitors worker tasks for errors or hanging
 - *Restarts as needed*
 - *Preferably on a different node*
- What if the application master goes down?
 - *YARN can try to restart it*
- What if an entire Node goes down?
 - *This could be the application master*
 - *The resource manager will try to restart it*
- What if the resource manager goes down?
 - *Can set up “high availability” (HA) using Zookeeper to have a hot standby*





MAPREDUCE: A REAL EXAMPLE

How many of each rating type exist?

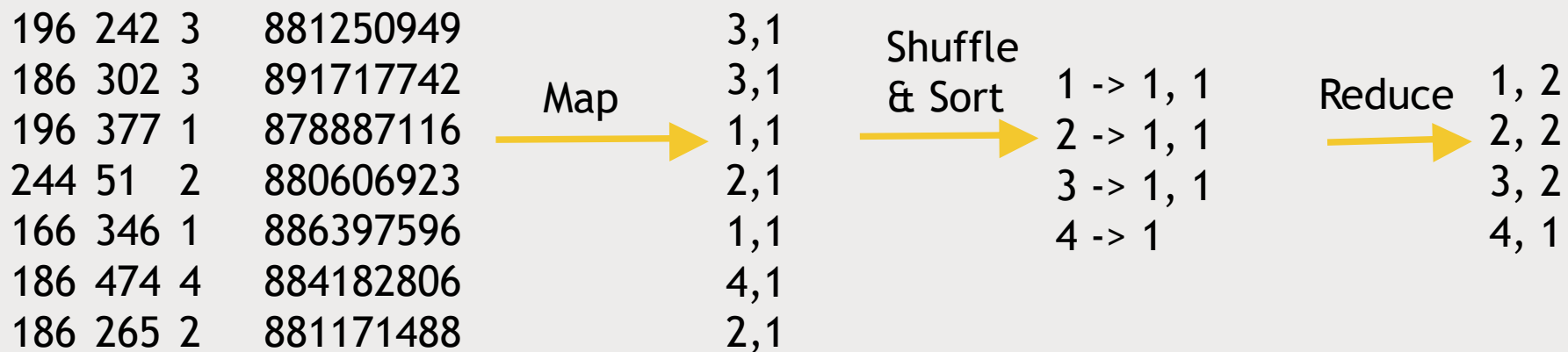


How many of each movie rating exist?



Writing the Mapper

USER ID|MOVIE ID|RATING|TIMESTAMP



```
def mapper_get_ratings(self, _, line):  
    (userID, movieID, rating, timestamp) = line.split('\t')  
    yield rating, 1
```

Writing the Reducer

USER ID | MOVIE ID | RATING | TIMESTAMP

196 242 3 881250949
186 302 3 891717742
196 377 1 878887116
244 51 2 880606923
166 346 1 886397596
186 474 4 884182806
186 265 2 881171488

Map

3,1
3,1
1,1
2,1
1,1
4,1
2,1

Shuffle
& Sort

1 -> 1, 1
2 -> 1, 1
3 -> 1, 1
4 -> 1

Reduce

1, 2
2, 2
3, 2
4, 1

```
def reducer_count_ratings(self, key, values):  
    yield key, sum(values)
```

Putting it all together


```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                  reducer=self.reducer_count_ratings)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```



RUNNING MAPREDUCE WITH MRJOB

Run our *MapReduce* job in our Hadoop installation



Installing what we need: HDP 2.6.5

- PIP

- *Utility for installing Python packages*
- `yum install python-pip`

- MRJob

- `pip install mrjob==0.5.11`

- Nano

- `yum install nano`

- Data files and the script

- `wget http://arif.works/hadoop/ml-100k/u.data`
- `wget http://arif.works/hadoop/RatingsBreakdown.py`

Installing what we need: HDP 2.5

■ PIP

- `cd /etc/yum.repos.d`
- `cp sandbox.repo /tmp`
- `rm sandbox.repo`
- `cd ~`
- `yum install python-pip`

■ MRJob

- `pip install google-api-python-client==1.6.4`
- `pip install mrjob==0.5.11`

■ Nano

- `yum install nano`

■ Data files and the script

- `wget http://arif.works/hadoop/ml-100k/u.data`
- `wget http://arif.works/hadoop/RatingsBreakdown.py`

Running with mrjob

- Run locally
 - *python RatingsBreakdown.py u.data*
- Run with Hadoop
 - *python MostPopularMovie.py -r hadoop --hadoop-streaming-jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar u.data*



YOUR CHALLENGE

Sort movies by popularity with Hadoop

Challenge exercise

- Count up ratings given for each movie
 - *All you need is to change one thing in the mapper - we don't care about ratings now, we care about movie ID's!*
 - *Start with this and make sure you can do it.*
 - *You can use nano to just edit the existing RatingsBreakdown.py script*

Stretch goal

- Sort the movies by their numbers of ratings
- Strategy:
 - *Map to (movieID, 1) key/value pairs*
 - *Reduce with output of (rating count, movieID)*
 - *Send this to a second reducer so we end up with things sorted by rating count!*
- Gotchas:
 - *How do we set up more than one MapReduce step?*
 - *How do we ensure the rating counts are sorted properly?*

Multi-stage jobs

- You can chain map/reduce stages together like this:

```
def steps(self):  
    return [  
        MRStep(mapper=self.mapper_get_ratings,  
            reducer=self.reducer_count_ratings),  
        MRStep(reducer=self.reducer_sorted_output)  
    ]
```

Ensuring proper sorting

- By default, streaming treats all input and output as strings. So things get sorted as strings, not numerically.
- There are different formats you can specify. But for now let's just zero-pad our numbers so they'll sort properly.

- The second reducer will look like this:

```
def reducer_count_ratings(self, key, values):  
    yield str(sum(values)).zfill(5), key
```

Iterating through the results

- Spoiler alert!

```
def reducer_sorted_output(self, count, movies):  
    for movie in movies:  
        yield movie, count
```

CHECK YOUR RESULTS

Did it work?



My solution

```
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                  reducer=self.reducer_count_ratings),
            MRStep(reducer=self.reducer_sorted_output)
        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield movieID, 1

    def reducer_count_ratings(self, key, values):
        yield str(sum(values)).zfill(5), key

    def reducer_sorted_output(self, count, movies):
        for movie in movies:
            yield movie, count

if __name__ == '__main__':
    RatingsBreakdown.run()
```