BIG DATA

- The Three Vs:

  - Volume

    - not just sampling

  - Velocity

    - real-time

  - Variety

    - text, audio, images, video

DISTRIBUTED COMPUTING

▸ vertical vs. horizontal scaling

▸ what's your limit?

▸ computing time (weeks vs. days vs. hours vs. minutes)

KEY REQUIREMENTS OF DISTRIBUTED COMPUTING

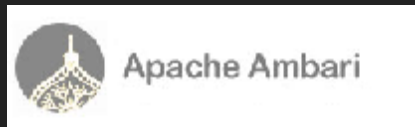▸ Fault tolerance

▸ Recoverability

▸ Consistency

▸ Scalability

HADOOP

▸ "The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing"

HADOOP
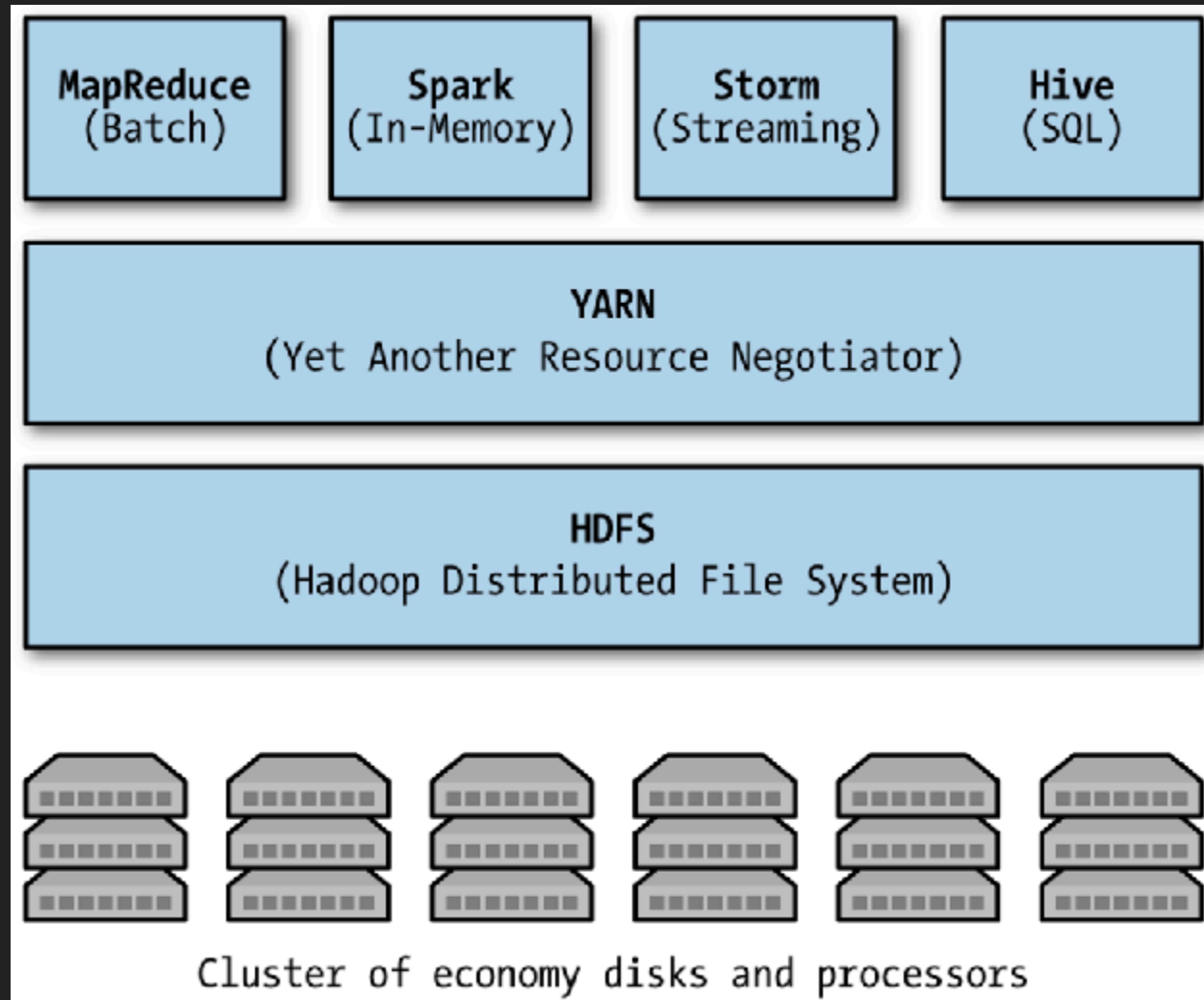
- HDFS
- YARN
- MapReduce
- Ambari
- Hive
- Pig
- Spark

HADOOP'S ANSWER TO KEY REQUIREMENTS

▸ data distribution

▸ fixed data block size and replication

▸ jobs and tasks

▸ high-level programming

▸ minimizing network traffic

▸ redundant jobs

▸ master-worker paradigm

HADOOP ARCHITECTURAL STACK

HDFS: HADOOP DISTRIBUTED FILE SYSTEM

▸ getting the data where it needs to be

  ▸ (i.e. close to where the computing is going to happen)

▸ typically 64MB, 128MB or 256MB blocks

▸ access is provided by NameNode and Secondary NameNode (think pilot and co-pilot)

YARN: YET ANOTHER RESOURCE NAVIGATOR

▸ decouples job and workload management from cluster and resource management

▸ YARN is to computing resources as HDFS is to storage

HDFS: A PRIMER

▸ getting your data onto a Hadoop machine isn't enough: you need to make it available to computations

▸ How? Put it into HDFS!

▸ HDFS looks like a plain old file system (but it's not)

▸ hadoop fs provides an interface to the filesystem that uses familiar syntax

HADOOP FS

▸ hadoop fs -ls

   ▸ lists the files in your HDFS directory

▸ hadoop fs -copyFromLocal <filename>

   ▸ copies <filename> from your file system to HDFS

▸ hadoop fs -rm <filename>

   ▸ deletes <filename> from your HDFS directory

▸ hadoop fs -cat <filename>

   ▸ shows contents of <filename>

ABOUT THE CONCEPT OF 'LOCAL'

▸ we are going to be using hadoop in three modes:

  ▸ local (e.g. MRJob): no real hadoop infrastructure required

  ▸ local virtual machine (e.g. Hortonworks): single-node hadoop infrastructure

  ▸ Advanced Research Computing (ARC) Flux Cluster

WHY USE LOCAL?

▸ Local

   ▸ Pros: easy to set up (almost none required), good for teaching concepts about MapReduce

   ▸ Cons: not really Hadoop, can't go beyond MapReduce

WHY USE LOCAL VIRTUAL MACHINE?

▸ Pros:

  ▸ Complete Hadoop ecosystem on your laptop

▸ Cons:

  ▸ 11GB required for image, plus more for storage

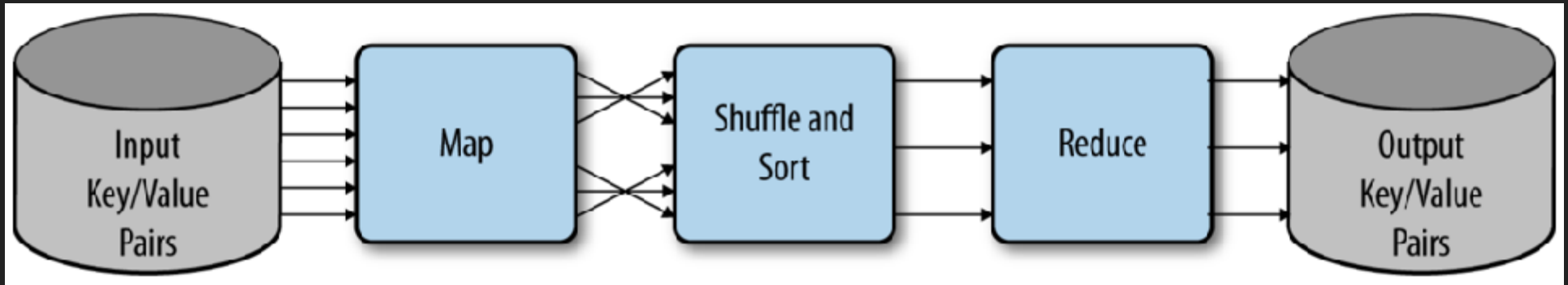 ▸ only provides single-node functionality

WHY USE FLUX (ARC)?

▸ Pros:

  ▸ Provides complete Hadoop experience
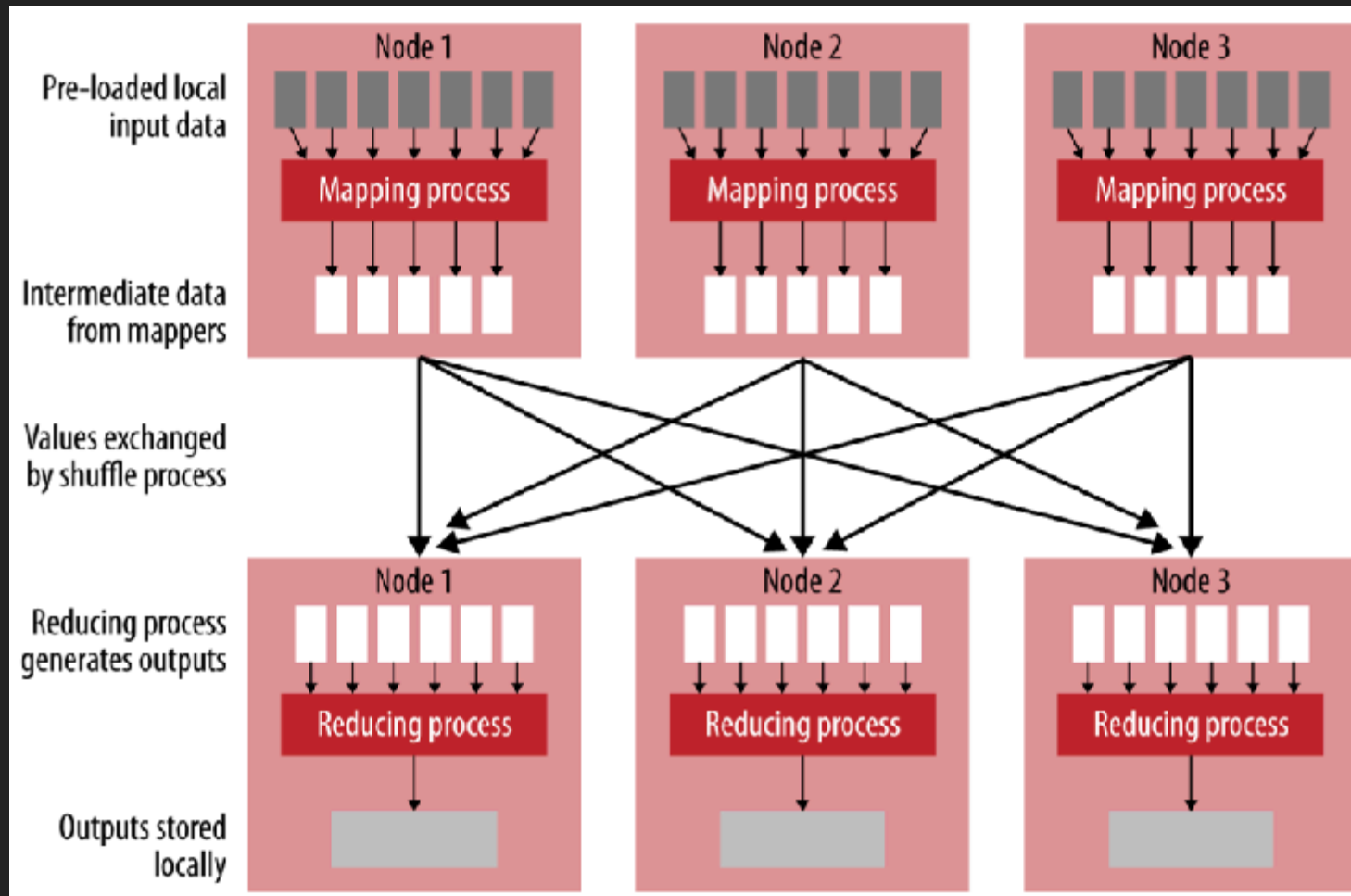
▸ Cons:

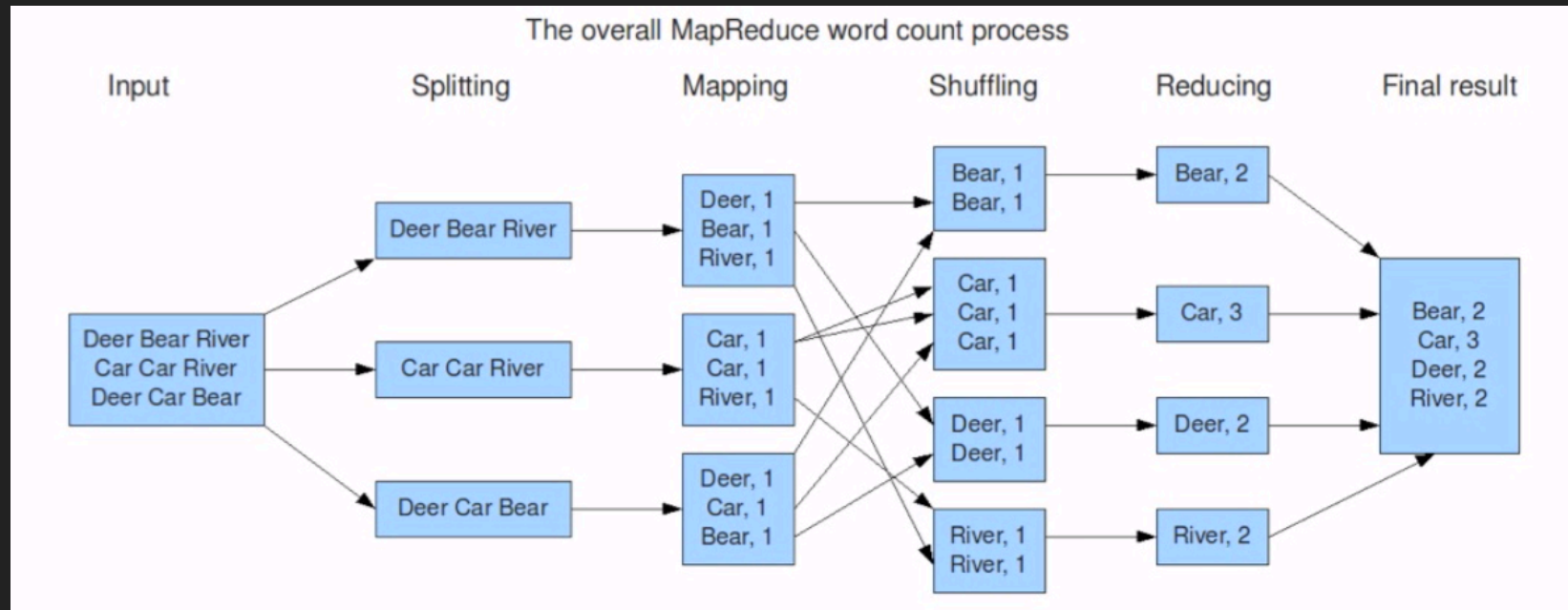 ▸ Provides complete Hadoop experience

LET'S LOOK AT MAPREDUCE

▸ (book_id, rating) -> count ratings -> (book_id, #ratings)

MAPREDUCE: MAP, SORT & SHUFFLE, REDUCE

MAPREDUCE: WORD-COUNT EXAMPLE

The overall MapReduce word count process

CLASS QUESTION:

▸ What are the necessary conditions to be able to use MapReduce?

TWO WAYS TO TACKLE MAPREDUCE

▸ Step-by-step:

  ▸ You provide the mapper, the reducer, optionally more

  ▸ mrjob

▸ Table-level:

  ▸ Load, transform, dump (saaaaay....)

  ▸ High-level SQL-like manipulation of data

  ▸ Hive, Pig, Spark

LET'S START WITH THE DIY APPROACH: MRJOB

▸ conda install mrjob

▸ stand-alone: can use entirely locally or hook up to more powerful back-end (like Amazon's Elastic MapReduce) Text

▸ see http://pythonhosted.org/mrjob for documentation

▸ must define at least one of: mapper, reducer, combiner
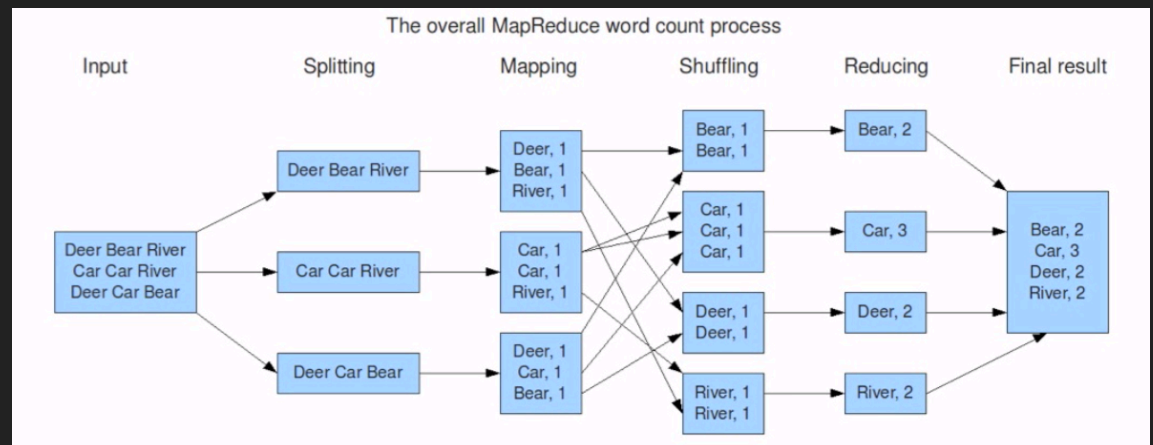
MRJOB FRAMEWORK AND WORD COUNTS

▸ Mapper:

   ▸ break input line into key, value pairs

▸ Reducer:

   ▸ take all (key, value) pairs with same key and compute aggregate function (e.g. sum, count)

▸ MRJob takes care of:
Sorting mapper output, invoking reduce tasks, assembling final output, scheduling and monitoring tasks

The overall MapReduce word count process

Input	Splitting	Mapping	Shuffling	Reducing	Final result

Deer Bear River
Car Car River
Deer Car Bear

Deer Bear River
Car Car River
Deer Car Bear

Deer, 1 Bear, 1 River, 1
Car, 1 Car, 1 River, 1
Deer, 1 Car, 1 Bear, 1

Bear, 1 Bear, 1
Car, 1 Car, 1 Car, 1
Deer, 1 Deer, 1
River, 1 River, 1

Bear, 2
Car, 3
Deer, 2
River, 2

Bear, 2
Car, 3
Deer, 2
River, 2

A BASIC MRJOB MAPREDUCE EXAMPLE

```python
#!/usr/bin/python

from mrjob.job import MRJob
import re


class MRWordFrequencyCount(MRJob):

    ### input: self, in_key, in_value
    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    ### input: self, in_key from mapper, in_value from mapper
    def reducer(self, key, values):
        yield key, sum(values)


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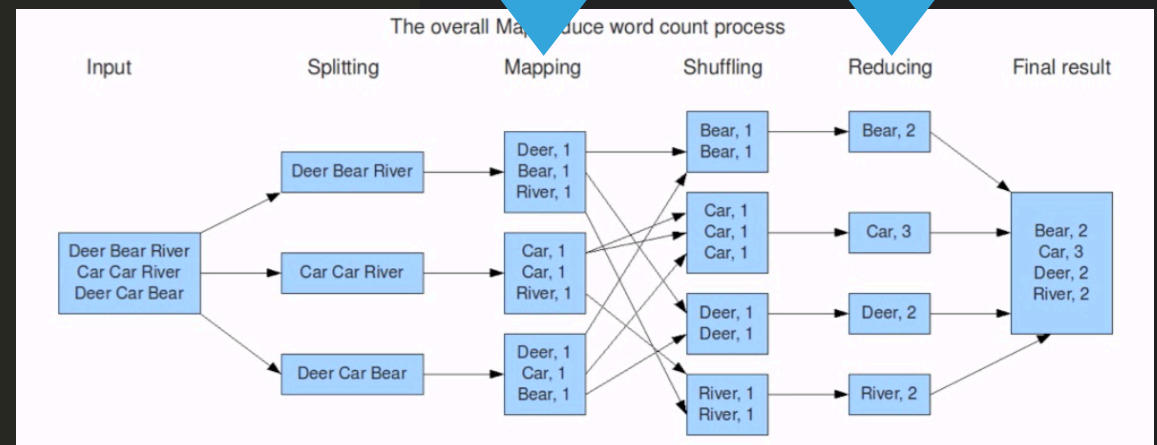

The overall MapReduce word count process

WHOA, WHOA, WHOA!

```python
#!/usr/bin/python

from mrjob.job import MRJob
import re


class MRWordFrequencyCount(MRJob):

  ### input: self, in_key, in_value
  def mapper(self, _, line):
    yield "chars", len(line)
    yield "words", len(line.split())
    yield "lines", 1

  ### input: self, in_key from mapper, in_value from mapper
  def reducer(self, key, values):
    yield key, sum(values)

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

- yield?

- it's a generator thing

- what?

GENERATORS

```
def f123():
    yield 1
    yield 2
    yield 3
for item in f123():
    print item
```

▸ A GENERATOR is a lazy iterable object: lazy objects wait to produce results only when they absolutely have to.

▸ When f123() is called, it does not return any of the values in the yield statements.

▸ It returns a generator object. Also, the function does not really exit - it goes into a suspended state.

▸ When the for loop tries to loop over the generator object, the function:

   ▸ Resumes from its suspended state

   ▸ Runs until the next yield statement

   ▸ Returns that yield result as the next item.

▸ This happens until the function exits, at which point the generator raises StopIteration, and the loop exits.

MAPPERS AND REDUCERS

▸ mappers and reducers are generators supplied by YOU!

▸ too weird?

   ▸ basically, yield means return and then pick up here next time through

WORDCOUNT BY MAPPING AND REDUCING

```python
#!/usr/bin/python

from mrjob.job import MRJob
import re

class MRWordFrequencyCount(MRJob):

  ### input: self, in_key, in_value
  def mapper(self, _, line):
    yield "chars", len(line)
    yield "words", len(line.split())
    yield "lines", 1

  ### input: self, in_key from mapper, in_value from mapper
  def reducer(self, key, values):
    yield key, sum(values)

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

▸ "To be or not to be,
That is the question."

▸ ("chars", 18), ("words", 6), ("lines", 1)
("chars", 20), ("words", 4), ("lines", 1)

▸ "chars": [18,20], "words": [6,4], "lines": [1,1]

▸ ("chars", 38), "words", 10), ("lines", 2)

LET'S RUN THAT MRJOB WORDCOUNT PROGRAM

```
↳ python wordcountmr.py andalusian.txt
No configs found; falling back on auto-configuration
No configs specified for inline runner
Running step 1 of 1...
Creating temp directory /var/folders/1r/ht9dh7xn5j1ddwl8p7frd1282yw4hz/T/wordcountmr.cteplovs.20171105.212554.677036
Streaming final output from /var/folders/1r/ht9dh7xn5j1ddwl8p7frd1282yw4hz/T/wordcountmr.cteplovs.20171105.212554.677036/output...
"chars" 655
"lines" 15
"words" 102
Removing temp directory /var/folders/1r/ht9dh7xn5j1ddwl8p7frd1282yw4hz/T/wordcountmr.cteplovs.20171105.212554.677036...
```

WHAT JUST HAPPENED?

▶ we ran our wordcount MapReduce python script on our local machine