CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

Distributed Data Management

13: Map Reduce Jobs

Outline

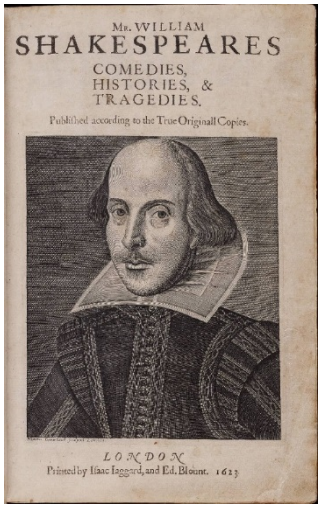
1. First Example: Shakespeare Word Average.
2. Second Example: Temperatures.

Outline

1. First Example: Shakespeare Word Average.
2. Second Example: Temperatures.

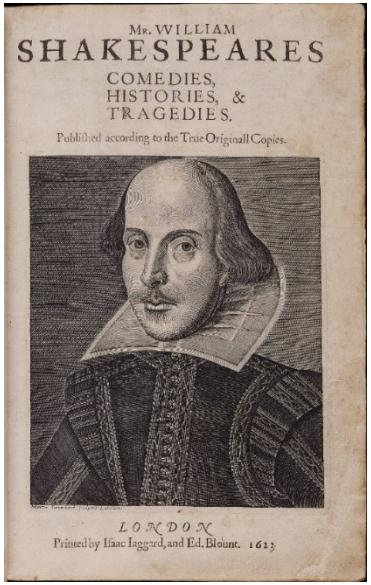
First Example: Shakespeare Word Average

- We have been provided with all the bibliographical works of William Shakespeare.
- They are contained (by categories) in the following 4 files:
  - comedies.txt
  - histories.txt
  - poems.txt
  - tragedies.txt



First Example: Shakespeare Word Average

- Our job in this intro example is not particularly interesting:
  - Get the average length of all words starting with ‘a’ (or ‘A’), with ‘b’ (or ‘B’), with ‘c’ (or ‘C’), …, and with ‘z’ (or ‘Z’).
  - We are not interested in words starting with any non-alphanumeric character (digit, ?, !, *, etc.)



First Example: Shakespeare Word Average

So, let's start hands on work…

First step: Get familiarised with the working environment.

1. Local Environment: (99% of our time)
   - OS: Windows (from vDesktop).
   - Python IDE: e.g. Spyder.
2. Actual Hadoop Environment: (1% of our time)
   - OS: Linux Centos (e.g. from VMware workstation or your own VirtualBox VM).
   - MapReduce Framework: VM Cloudera Quicksart v5.13.

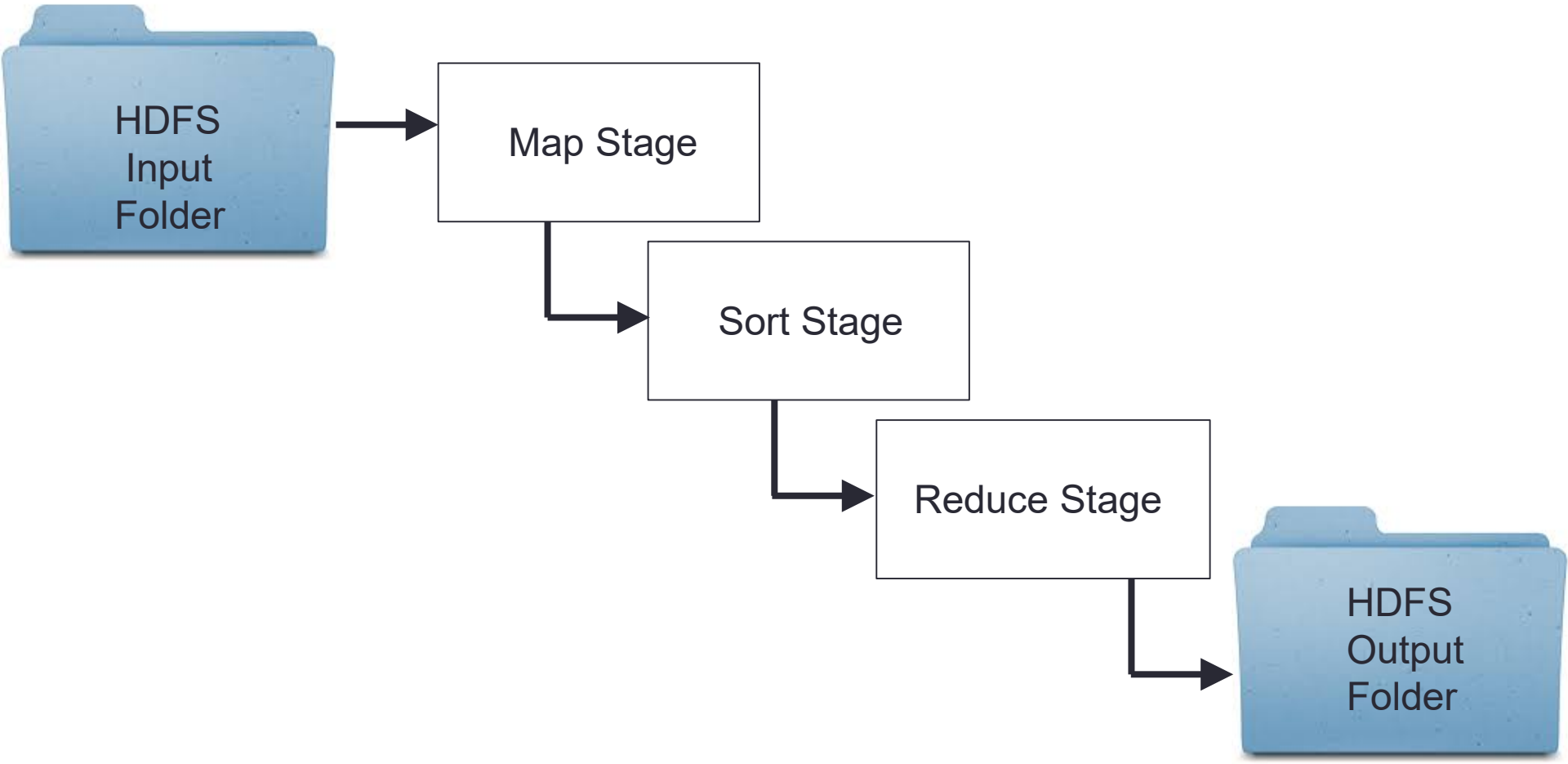
First Example: Shakespeare Word Average

- So, why a local environment first?
  - Easier for you → No need to adapt, you can just keep working as you did for assignments 1 and 2.
  - Due to the *small datasets* we are going to play with, it was possible to create a Windows-based environment *simulating* the entire process of MapReduce.
  - Thus, you can just use it to test on Windows whether your MapReduce job is actually working or not. And, most important:
    - In case it is not working, debug it with e.g. Spyder.

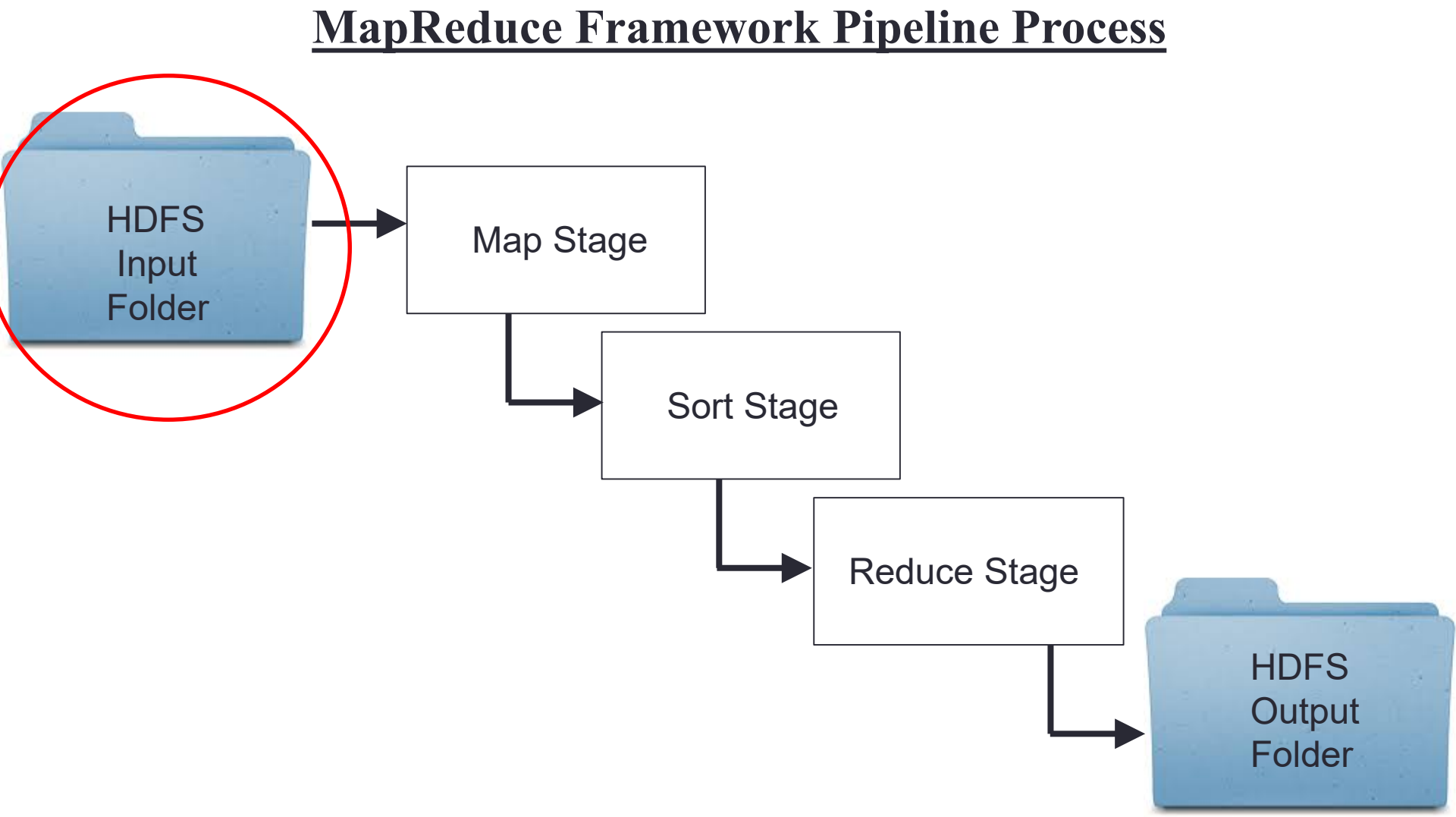
First Example: Shakespeare Word Average

- So, why a local environment first?
  - This methodology is also used in real-world examples.
    - You first develop and test your MapReduce job locally.
      To do so, you use only a small subset/slice of your dataset.
      - In our case, as the full dataset will be relatively small, we can try out tests directly for the entire dataset.
    - Once you know both your *mapper* and your *reducer* work fine:
      - You trigger the full MapReduce job (over the entire dataset) to the nodes of your cluster.

First Example: Shakespeare Word Average



First Example: Shakespeare Word Average



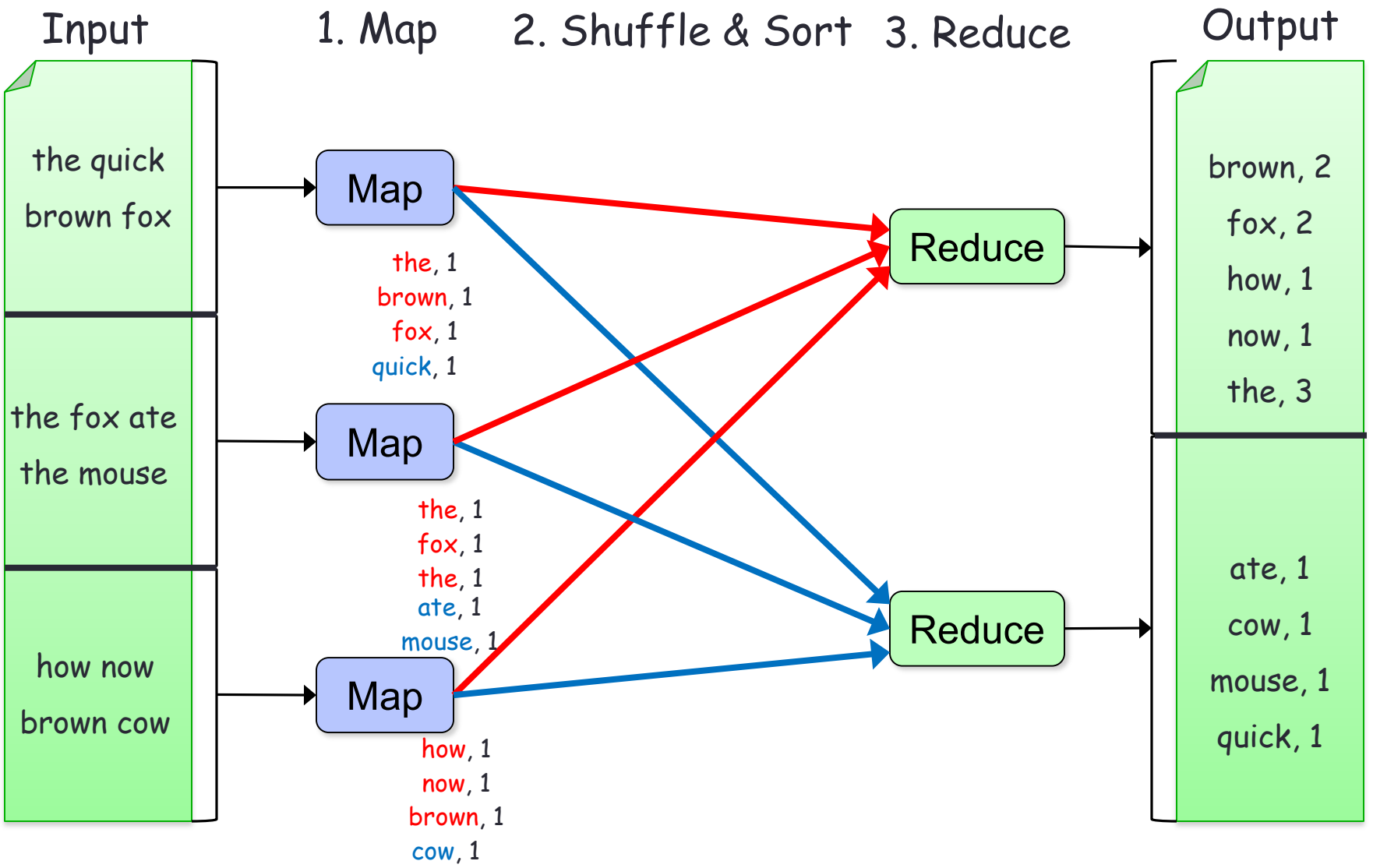
First Example: Shakespeare Word Average

Get to know your dataset files!

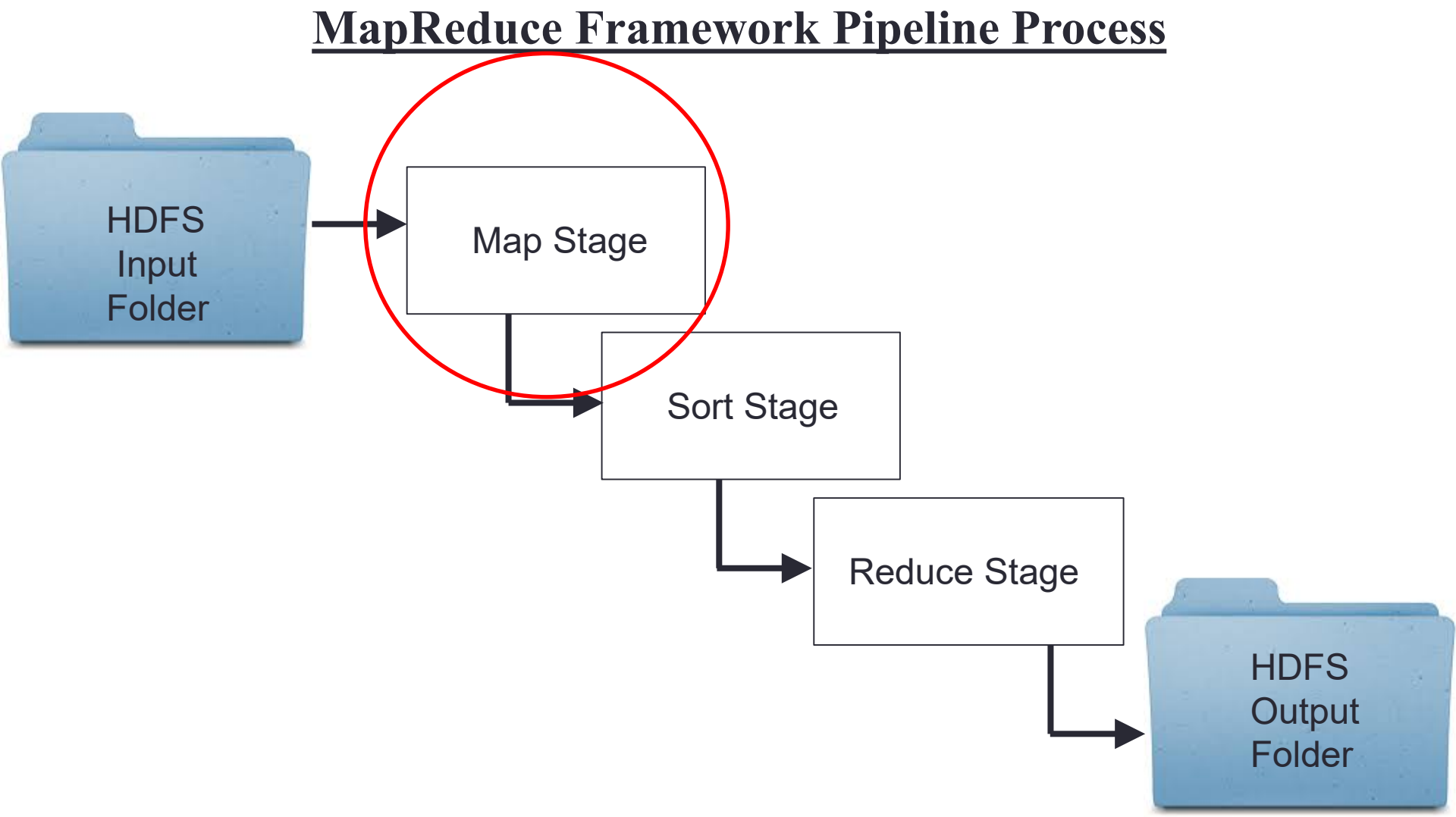
They contain the data you're going to process!

- Observation:
  1. Pay attention to their size.
     - Are they small enough or are they going to be split when moved to HDFS?
  2. Try to come up with a general enough structure of a file.
     - Each mapper will receive 1 file (or block of a file, if the file was big).
     - Now, you might have dozens/hundreds/thousands of mappers…
       …but you are just going to programme the mapper once!
       …all your dozens/hundreds/thousands of mappers, working over different files, are about to do the very same!
     - So, look at the files…how do you want each mapper to process them?

First Example: Shakespeare Word Average



First Example: Shakespeare Word Average



First Example: Shakespeare Word Average

Map Stage

So, what's the main idea of the entire map reduce process?

- You have so much data, that you cannot tackle it in one go.
  - Instead, you have to tackle it in pieces/slices.
  - Thus, the results you get by tackling a small piece cannot be considered a final result, as it is representative of a data slice.

First Example: Shakespeare Word Average

Map Stage

So, what's the main idea of the entire map reduce process?

- Thus, what a map stage is basically asking you is:
  - How do you want me to process this data knowing that the computed results are partial/intermediate, and they are going to be combined later on with other partial results so as to compute the final ones?
- Most of the time the best approach is just to process this slice of data as if it represents the entire dataset.
  - With this approach *you more or less guarantee* that your mappers and reducers are going to do nearly the same.
  - So, by programming the mapping you will more or less have your reducer *half way programmed* as well.

First Example: Shakespeare Word Average

Map Stage

- Our files are small, so we are going to have 4 map processes working in parallel, respectively dealing with:
  comedies.txt, histories.txt, poems.txt and tragedies.txt

- We need to come up with a method that is:
  - Specific enough to be applicable to each file to do the job.
  - Flexible enough to deal with *small variations* among the files.

Any ideas?

Tip: Do not think in Python, think at a high-level, in English!

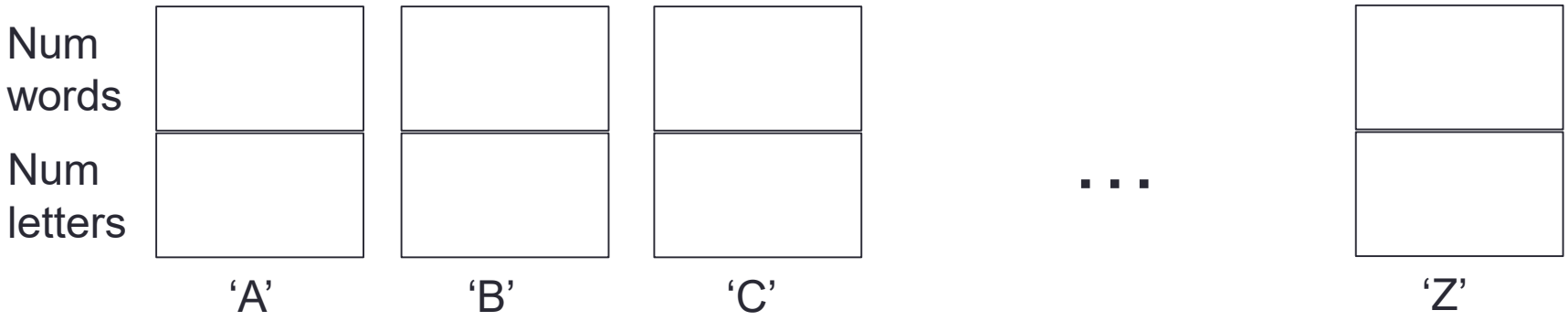
First Example: Shakespeare Word Average

Map Stage

- What about having boxes? Two boxes per letter:
  - One representing the amount of words found starting with <letter>.
  - One representing the total letters of the words starting with <letter>.

Key Concept:

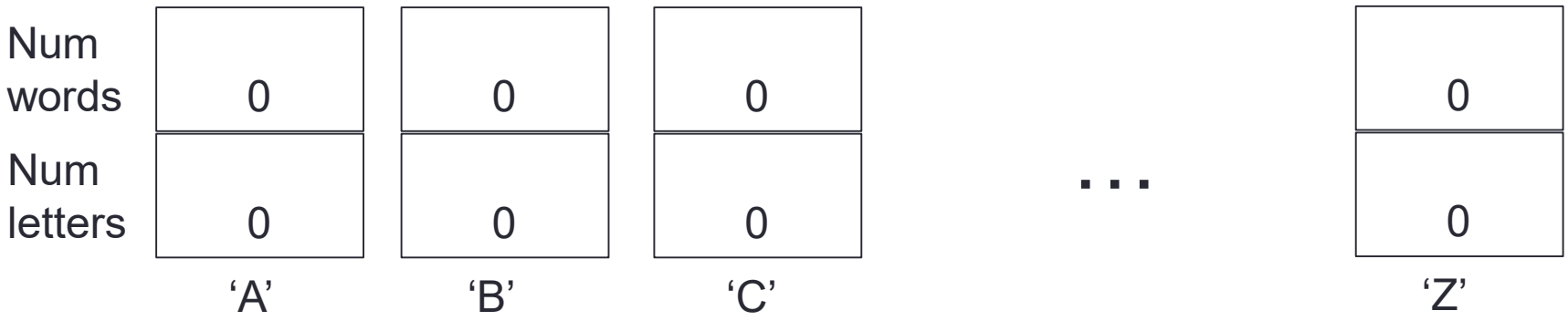
- We **must preserve consistency** between the portion of file we have processed so far and the content of these boxes.



First Example: Shakespeare Word Average

Map Stage

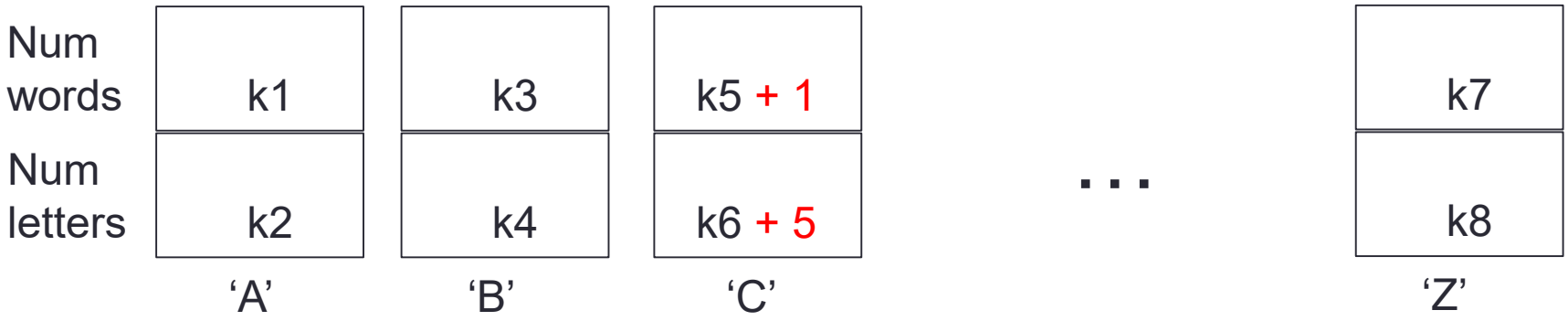
- What about having boxes? Two boxes per letter:
  - One representing the amount of words found starting with <letter>.
  - One representing the total letters of the words starting with <letter>.
- Thus, at the beginning of the process of the file, all boxes should contain 0.



First Example: Shakespeare Word Average

Map Stage

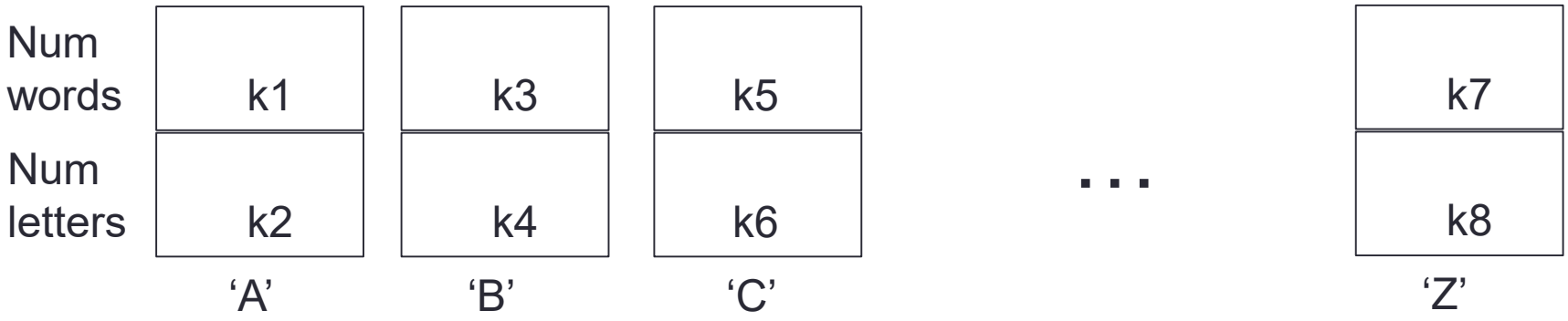
- What about having boxes? Two boxes per letter:
  - One representing the amount of words found starting with <letter>.
  - One representing the total letters of the words starting with <letter>.
- When a new word my_word is processed, we check its starting letter (e.g., suppose 'c' or 'C') and its number of letters (e.g., suppose 5) and we maintain the consistency of the boxes!



First Example: Shakespeare Word Average

Map Stage

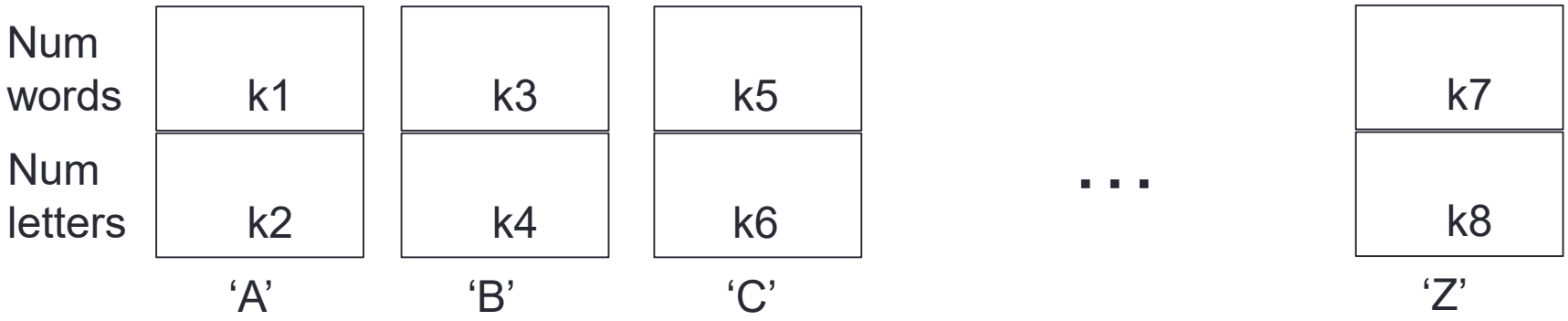
- What about having boxes? Two boxes per letter:
  - One representing the amount of words found starting with <letter>.
  - One representing the total letters of the words starting with <letter>.
- When the entire file has been processed, the content of our boxes is the result we wanted to compute!



First Example: Shakespeare Word Average

Map Stage

- Map input: Lines of a text file, read one by one via streaming.
- Approach:
  1. Read a line.
  2. Divide it into its words.
  3. Process the words one by one, keeping the consistency of boxes.



First Example: Shakespeare Word Average

Map Stage

And that's pretty much it!

- Let's analyse the file my_mapper.py to see the actual implementation of our algorithm in Python.

First Example: Shakespeare Word Average

Map Stage

- The Python program (the file) contains the following 5 functions:
  1. process_line(line)
  2. get_index_in_alphabet(letter)
  3. get_letter_from_index(value)
  4. my_map(input_stream, output_stream)
  5. my_main( )

- Executing the program means executing the function my_main() first

First Example: Shakespeare Word Average

Map Stage

1. process_line(line)

- Given a line of text of our file, this function **strips** out invalid characters and **splits** the line into a words list, then returns the list.

2. get_index_in_alphabet(letter)

- Given a letter, this function returns an integer representing its index in the alphabet (e.g., ‘a’ or ‘A’ would be 0, ‘b’ or ‘B’ would be 1, ‘c’ or ‘C’ would be 2, and so on (note: -1 for non-alphanumeric letters)).

3. get_letter_from_index(value)

- Symmetric to function 2. Now, given the integer index of the alphabet, it returns its associated letter.

First Example: Shakespeare Word Average

Map Stage

4. my_map(input_streaming, output_streaming)

❑ This function implements the English algorithm we explained before:

1. Creates the boxes → actually using two parallel lists. All elements initially 0.
2. For loop, reading one line from the file at a time…
3. When the entire file is processed: for loop in the boxes…
   - If the box contains at least one word…
     - Get the letter from the index (using 'get_letter_from_index')
     - Get the content of the boxes.
     - Print the following (key, value) to the output_stream:
       - ❖ letter (num_words, total_letters)

First Example: Shakespeare Word Average

Map Stage

5. my_main()

- Provides two operating modes:
  - Mode 1 → Testing:
    - Start using this mode to test if your program works. Practise with a dummy file, with very simple input, to see what your program outputs.
  - Mode 2 → For the actual MapReduce:
    - Once you know your program works, use this mode for the actual MapReduce job, as it uses the stdin and stdout standard I/O channels.
  - For correct mode comment / uncomment correct lines
- In both modes, the function triggers the execution of the function my_map using the input and output streaming selected by the mode.

First Example: Shakespeare Word Average

Map Stage

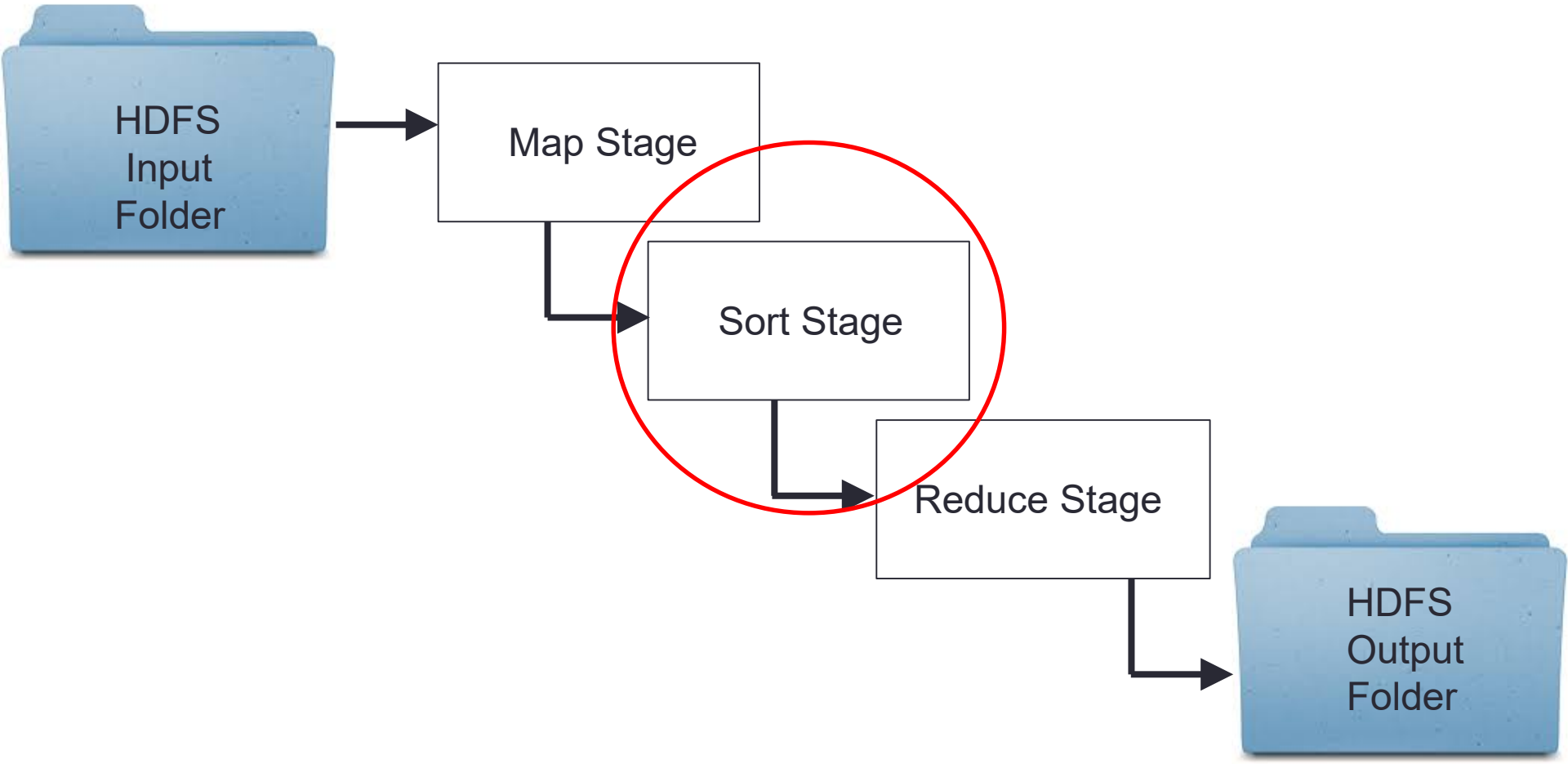
Key Concept:

- Once you have tested your map works for a single file…

…next step is to test it for the entire dataset!

- The program my_mapper_simulation.py does it for you! Just double click on it and see the content of the file map_simulation.txt being generated.
- This file contains the output of running the map stage for all the files of your dataset. If it does not do it, then it means your mapper contains some bugs as it cannot deal with some of the dataset files – maybe you need to use the full path to the my_dataset folder, for example.

First Example: Shakespeare Word Average



First Example: Shakespeare Word Average

Sort Stage

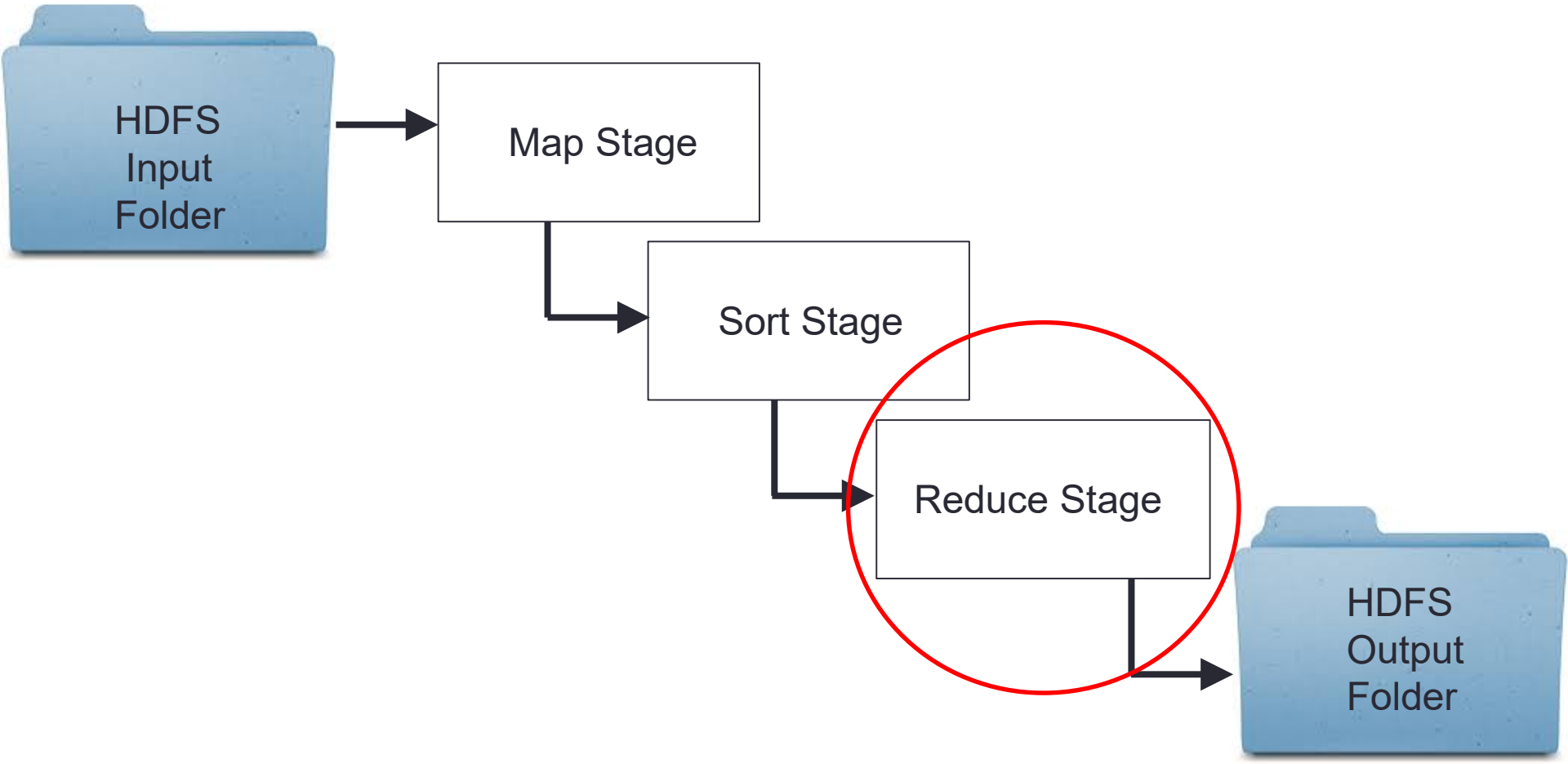
Key Concept:

- Once you have tested your map works for the entire dataset…

…the file map_simulation.txt contains the (key, value) pairs produced by your mappers for the entire dataset.

- However, these (key, value) pairs are not sorted.
- my_sort_simulation.py is a program that reads map_simulation.txt and produces sort_simulation.txt, pretty much with the same set of (key, value) pairs, but all of them sorted by the key.
- This is the way the reducers are going to receive the pairs.

First Example: Shakespeare Word Average



First Example: Shakespeare Word Average

Reduce Stage

Key Concept:

Let's think our MapReduce jobs are going to have a single reduce. This is the most common case (and the easiest to understand as well).

- If just one reduce process…
  - we can ensure that the input this reduce process is receiving is info representative of the entire dataset.
  - Not info representative of just a slice of the dataset.
  - Multiple mappers already worked on each of the slices, computing intermediate results.
  - All these intermediate results were collected and sorted, and now you are receiving them.

First Example: Shakespeare Word Average

Reduce Stage

- So, how do we process these (key, values) intermediate results?

a (24238,80727)
a (29063,104366)
a (30574,101950)
a (4103,14072)
b (13294,64813)
b (14676,68162)
b (16211,75212)
b (2842,13305)
c (10926,70107)
:

Any ideas?

Tip: Do not think in Python, think at a high-level, in English!

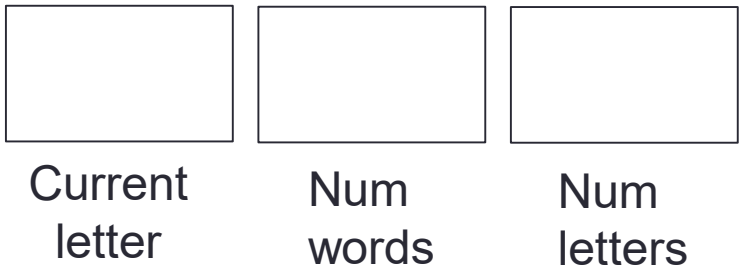
First Example: Shakespeare Word Average

Reduce Stage

- What about having only three boxes now?
  - One representing the 'letter' we are processing now.
  - One representing the amount of words found starting with 'letter'.
  - One representing the total letters of the words starting with 'letter'.

Key Concept:

- We **must preserve consistency** between the portion of file we have processed so far and the content of these boxes.



First Example: Shakespeare Word Average

Reduce Stage

- What about having only three boxes now?
  - One representing the 'letter' we are processing now.
  - One representing the amount of words found starting with 'letter'.
  - One representing the total letters of the words starting with 'letter'.

- Thus, at the beginning of the process of the file, current is '' (as we have processed no letter yet), and num words and num letters are 0.

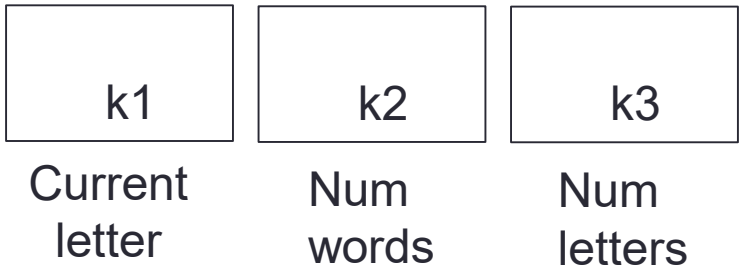
First Example: Shakespeare Word Average

Reduce Stage

- When a new (key, value) is processed from the file, we must maintain the consistency of the boxes.
- Let's see the format each (key, value) has:

letter (num_words, num_letters)

Let's reason case by case



First Example: Shakespeare Word Average

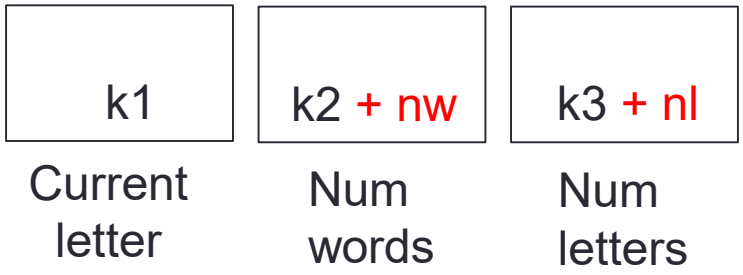
Reduce Stage

- **Case 1**: If 'letter' is equal to k1

letter (nw, nl)

In this case it means we are processing a new entry for the current letter we have been processing so far. Thus:

1. We just update our boxes.



First Example: Shakespeare Word Average

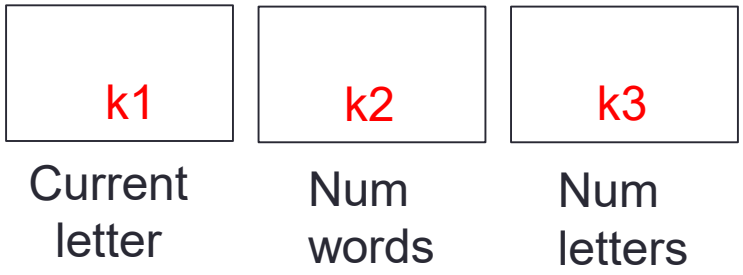
Reduce Stage

- **Case 2**: If 'letter' is different to k1

letter (nw, nl)

In this case it means there are no more entries for the current letter we have been processing so far. Thus:

1. We print (output) the result for the current letter (using the boxes).



First Example: Shakespeare Word Average

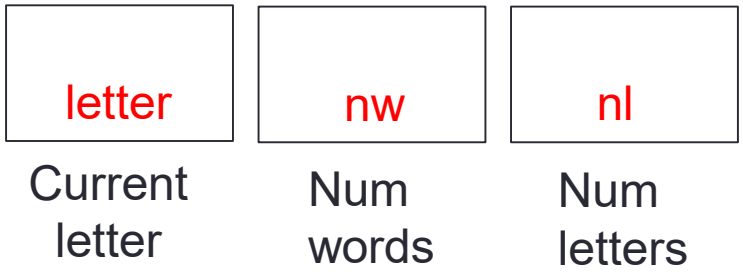
Reduce Stage

- **Case 2**: If 'letter' is different from k1

letter (nw, nl)

In this case it means there are no more entries for the current letter we have been processing so far. Thus:

1. We print (output) the result for the current letter (using the boxes).
2. We reset the boxes.



First Example: Shakespeare Word Average

Reduce Stage

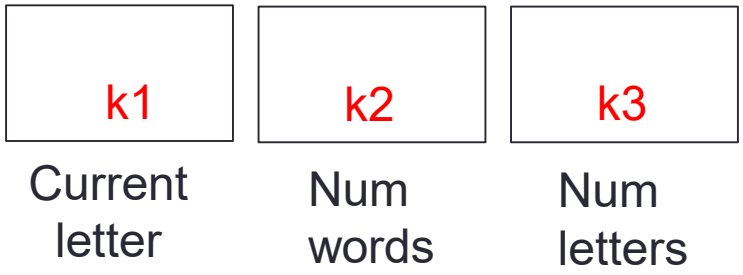
- **Case 3**: If 'letter' is different from k1, but k1 = ''

letter (nw, nl)

In this case it means we are processing the first line of the file. Thus:

~~1. We print (output) the result for the current letter (using the boxes).~~

2. We initialise the boxes.



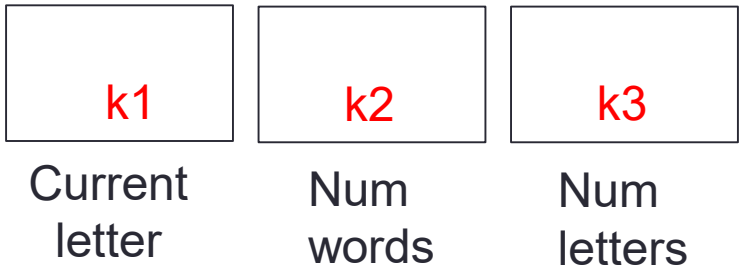
First Example: Shakespeare Word Average

Reduce Stage

- **Case 4**: When there are no more lines to process, but k1 != ‘’

  This means, we have finished processing the file, but there is a last letter which we have been processing but still not outputted its results. Thus:

  1. We print (output) the result for the current letter (using the boxes).
  2. ~~We reset the boxes.~~



First Example: Shakespeare Word Average

Reduce Stage

And that's pretty much it!

- Let's analyse the file my_reducer.py to see the actual implementation of our algorithm in Python.

First Example: Shakespeare Word Average

Reduce Stage

- The Python program (the file) contains the following 3 functions:
  1. get_key_value(line)
  2. print_key_value(letter, num_words, total_length, output_stream)
  3. my_reduce(input_stream, output_stream)
  4. my_main( )

- Executing the program means executing the function my_main() first

First Example: Shakespeare Word Average

Reduce Stage

1. get_key_value(line)

- Given a line of text in our file representing a (key, value), this function returns the letter, num_words, total_length.

2. print_key_value(letter, num_words, total_length, output_stream)

- Given a letter, the num_words and the total_length, it prints to the output stream the average length of the words starting with this letter.

First Example: Shakespeare Word Average

Reduce Stage

3. my_reduce(input_streaming, output_streaming)

- This function implements the English algorithm we explained before:

1. Assigns a current letter ‘’ and initialises boxes to 0.
2. For loop, reading one line from the file at each time…
   - Get the letter (num_words, total_words) key-value pair from the line (using ‘get_key_value’).
   - Implement cases 1, 2 and 3.
3. Implement case 4.

First Example: Shakespeare Word Average

Reduce Stage

4. my_main()

- Provides two operating modes:
  - Mode 1 → Testing:
    - Start using this mode to test if your program works. Practise it with "sort_simulation.txt" to see what your program outputs.
  - Mode 2 → For the actual MapReduce:
    - Once you know your program works, use this mode for the actual MapReduce job, as it uses the stdin and stdout standard I/O channels.

- In both modes, the function triggers the execution of the function my_reduce using the input and output streaming selected by the mode.

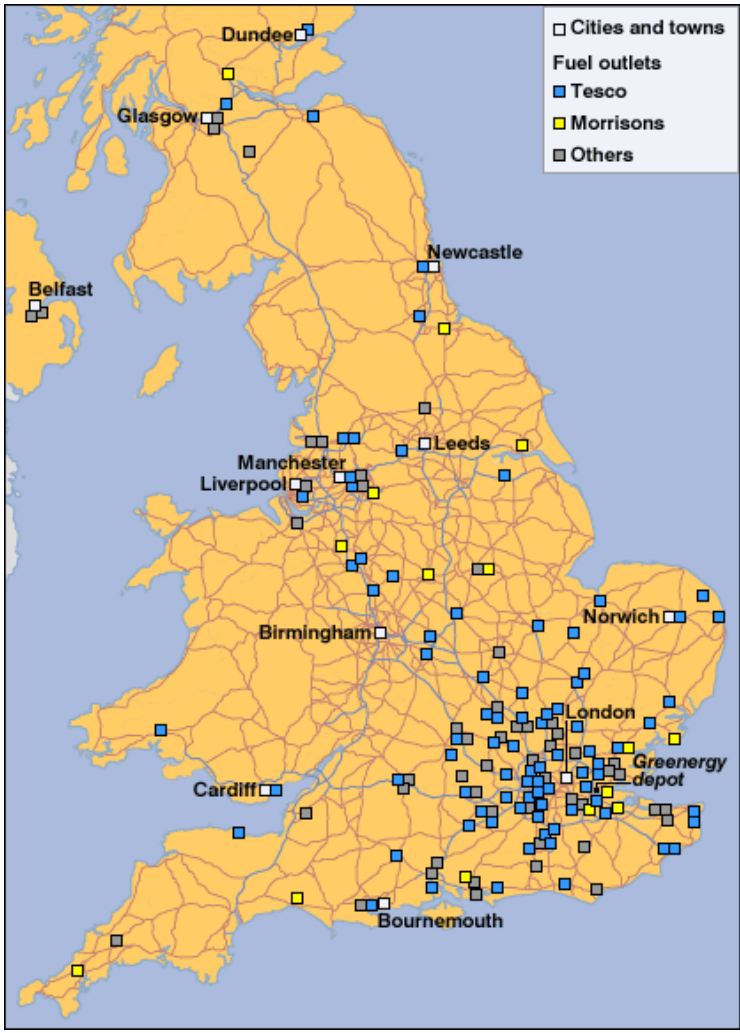
Outline

1. First Example: Shakespeare Word Average.
2. Second Example: Temperatures.

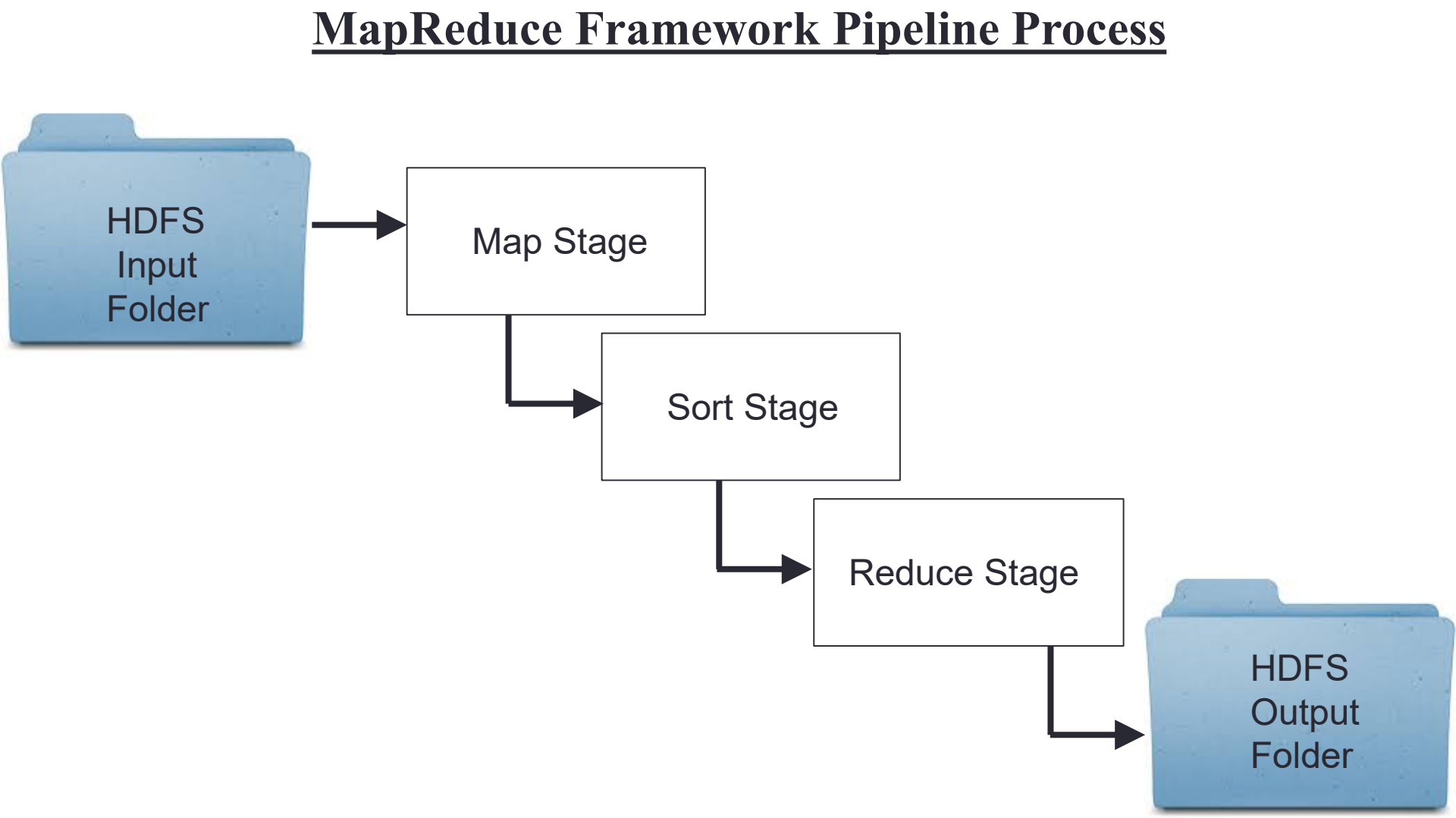
Second Example: Temperatures

- We have been provided with temperature records (over the last century) for 36 locations of the UK.
- Our job is to find out the lowest temperature ever registered for one of the cities:

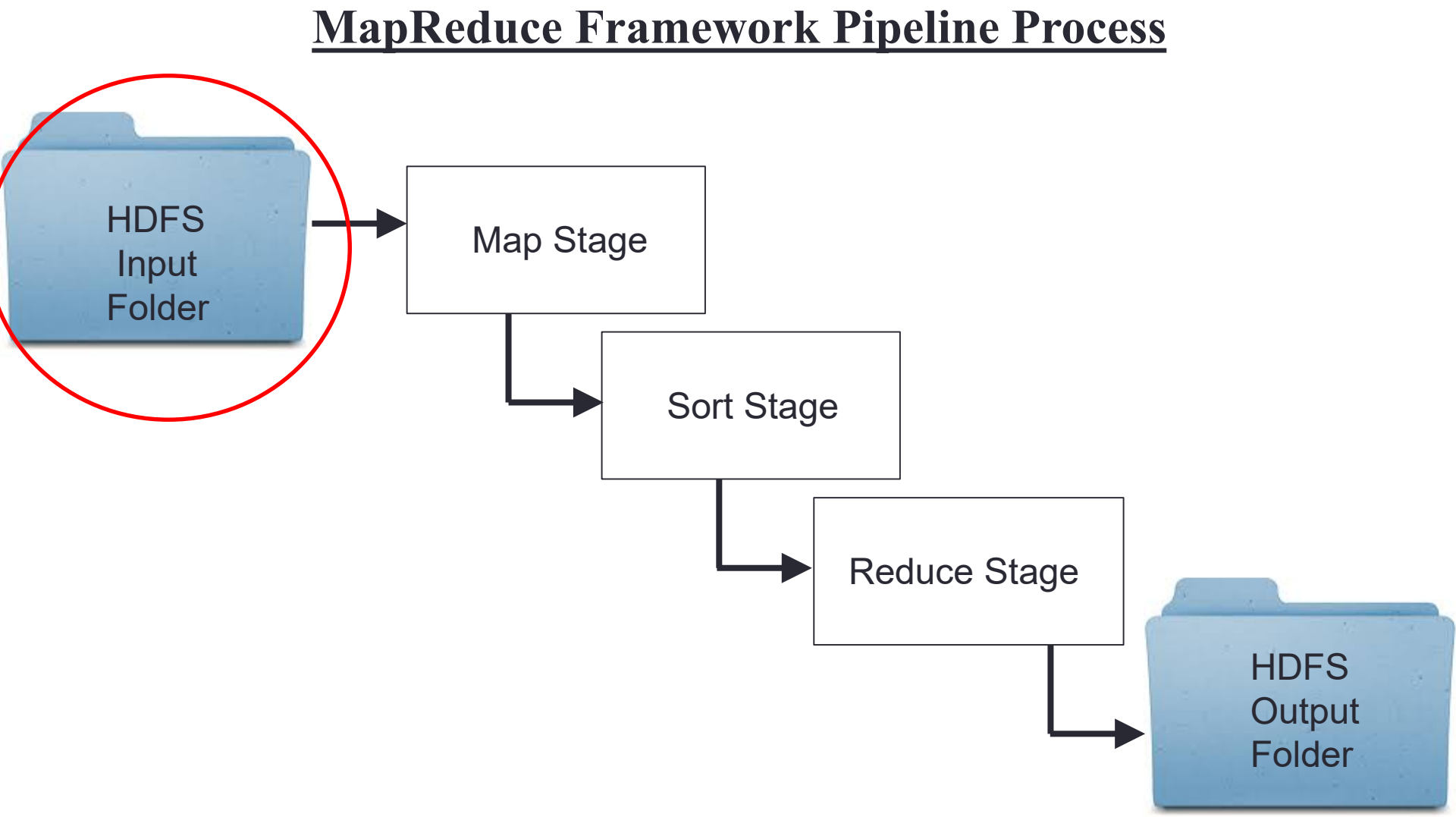
  (location, year, temperature)



Second Example: Temperatures



Second Example: Temperatures



Second Example: Temperatures

Get to know your dataset files!

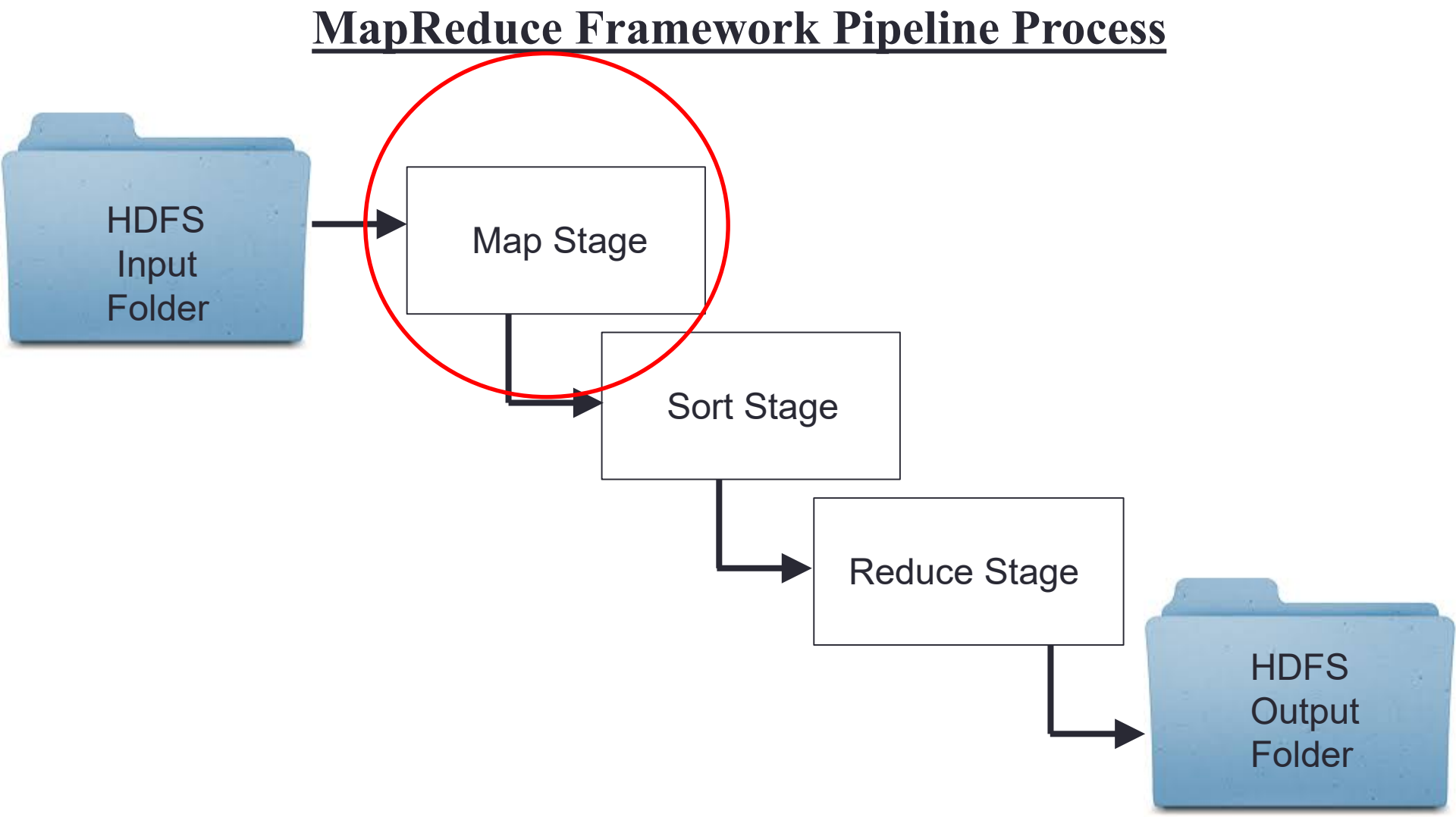
They contain the data you're going to process!

- Each location has its own file, which contains the following format:
  - locationName
  - four lines of text info about the location.
  - year mm tmax tmin af rain sun
  - measuring units for each of the info columns
  - year1 mm1 tmax1 tmin1 af1 rain1 sun1
  - year2 mm2 tmax2 tmin2 af2 rain2 sun2

    … … … … … … …
  - yearN mmN tmaxN tminN afN rainN sunN

- We are only interested in the information coloured in blue.

Second Example: Temperatures



Second Example: Temperatures

Map Stage

- Our files are small, so we are going to have 36 map processes working in parallel, respectively dealing with:
  Aberporth.txt, Braemar.txt, …, Oxford.txt, …, Yeovilton.txt

- We need to come up with a method that is:
  - Specific enough, to be applicable to each file to do the job.
  - Flexible enough, to deal with *small variations* among the files.

Any ideas?

Tip: Do not think in Python, think at a high-level, in English!

Second Example: Temperatures

Map Stage

- Approach → Treat the slice of data the map is working with as if it were the entire dataset.

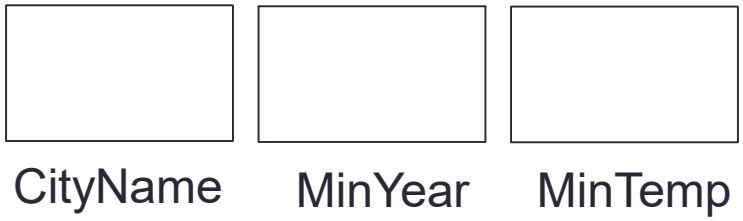
- What about three boxes:
  - One representing the city.
  - One representing the year with minimum temperature we have found so far.
  - One representing the actual minimum temperature found so far.

Second Example: Temperatures

Map Stage

Key Concept:

- We **must preserve consistency** between the portion of file we have processed so far and the content of these boxes.



Second Example: Temperatures

Map Stage

- We initialise the boxes by reading the first line of the file.
  - It contains the name of the city.
  - For the year and temp we pick dummy values (extremely large, so that they will be defeated by any other text file line).

Second Example: Temperatures

Map Stage

- When a new line my_word is processed, extract its year and its tmin.
- If tmin beats MinTemp we update the boxes.

Second Example: Temperatures

Map Stage

- When the entire file has been processed, the content of our boxes is the result we wanted to compute!
- So we just print it under the format:
  - k1 (k2, k3)

Second Example: Temperatures

Map Stage

And that's pretty much it!

- Let's analyse the file my_mapper.py to see the actual implementation of our algorithm in Python.

Second Example: Temperatures

Map Stage

- The Python program (the file) contains the following 5 functions:
  1. is_a_int(value)
  2. is_a_float(value)
  3. process_first_line(line)
  4. process_year_line(line)
  5. my_map(input_stream, output_stream)
  6. my_main( )

- Executing the program means executing the function my_main() first

Second Example: Temperatures

Map Stage

1. is_a_int(value)

- Given a value, it returns a Boolean stating if the value is an int or not.

2. is_a_int(value)

- Same as previous function, but for a float.

3. process_first_line(line)

- Given the first line of the file, it returns the name of the city.

4. process_year_line(line)

- Given a line of the file, it returns the year and min temperature registered on the line.

Second Example: Temperatures

Map Stage

5. my_map(input_streaming, output_streaming)

- This function implements the English algorithm we explained before:

1. Reads the first line for getting the city. Initialises year and temp to dummy values.
2. Reads the next 6 lines of the file to discard them.
3. For loop, reading one line from the file at each time…
   - Processes the line (using 'process_year_line')
   - If the new temperature beats the best one, then updates the boxes.
4. Once the file has been entirely processed:
   - Print the content of the boxes as the output.

Second Example: Temperatures

Map Stage

6. my_main()

- Provides two operating modes:
  - Mode 1 → Testing:
    - Start using this mode to test if your program works. Practise with a dummy file, with very simple input, to see what your program outputs.
  - Mode 2 → For the actual MapReduce:
    - Once you know your program works, use this mode for the actual MapReduce job, as it uses the stdin and stdout standard I/O channels.

- In both modes, the function triggers the execution of the function my_map using the input and output streaming selected by the mode.

Second Example: Temperatures

Map Stage

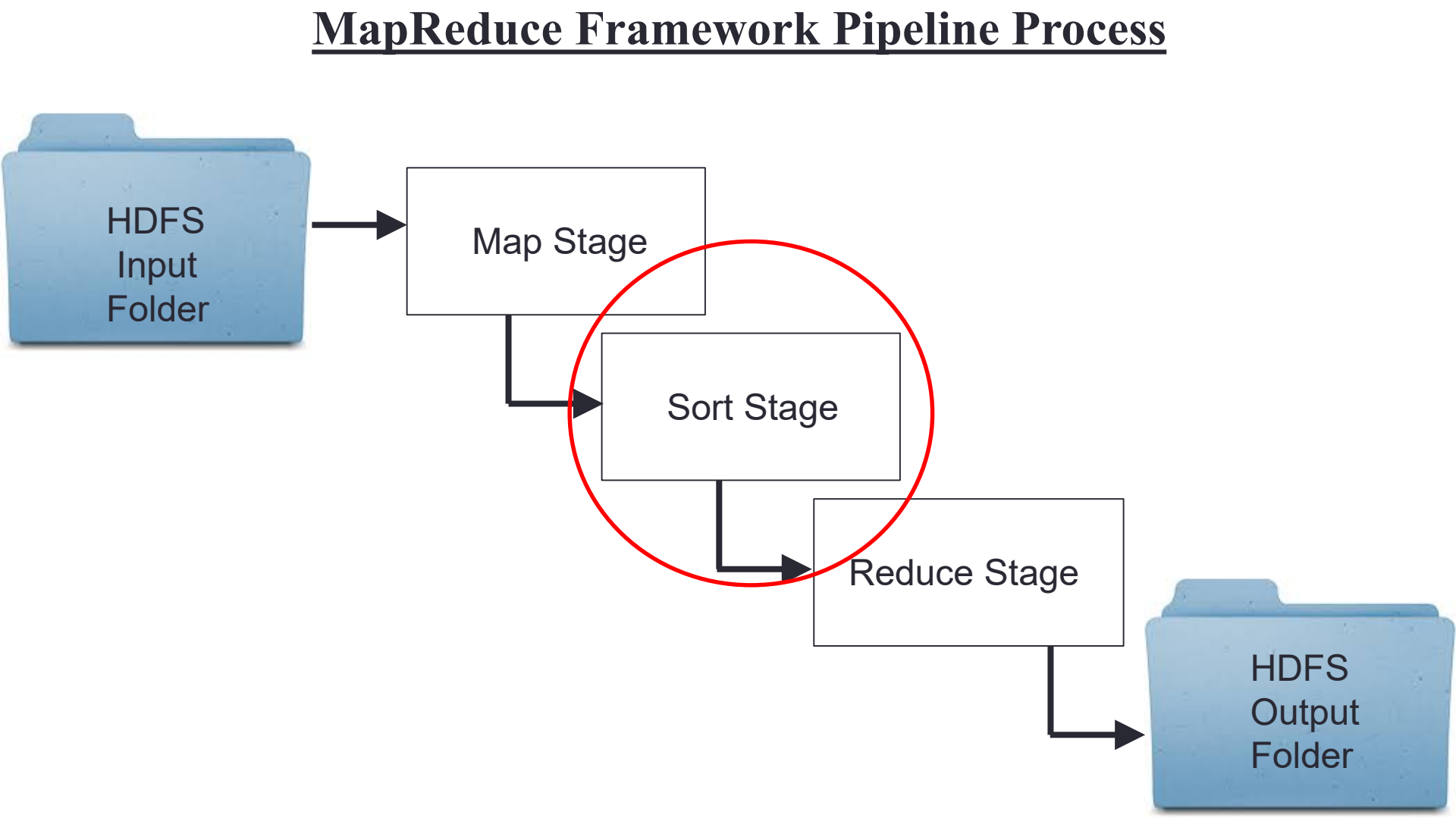
Key Concept:

- Once you have tested your map works for a single file…

…next step is to test it for the entire dataset!

- The program my_mapper_simulation.py does it for you! Just double click on it and see the content of the file map_simulation.txt being generated.
- This file contains the output of running the map stage for all the files of your dataset. If it does not do it, then it means your mapper contains some bugs as it cannot deal with some of the dataset files. E.g. are you running the script from its director?

Second Example: Temperatures



Second Example: Temperatures

Sort Stage

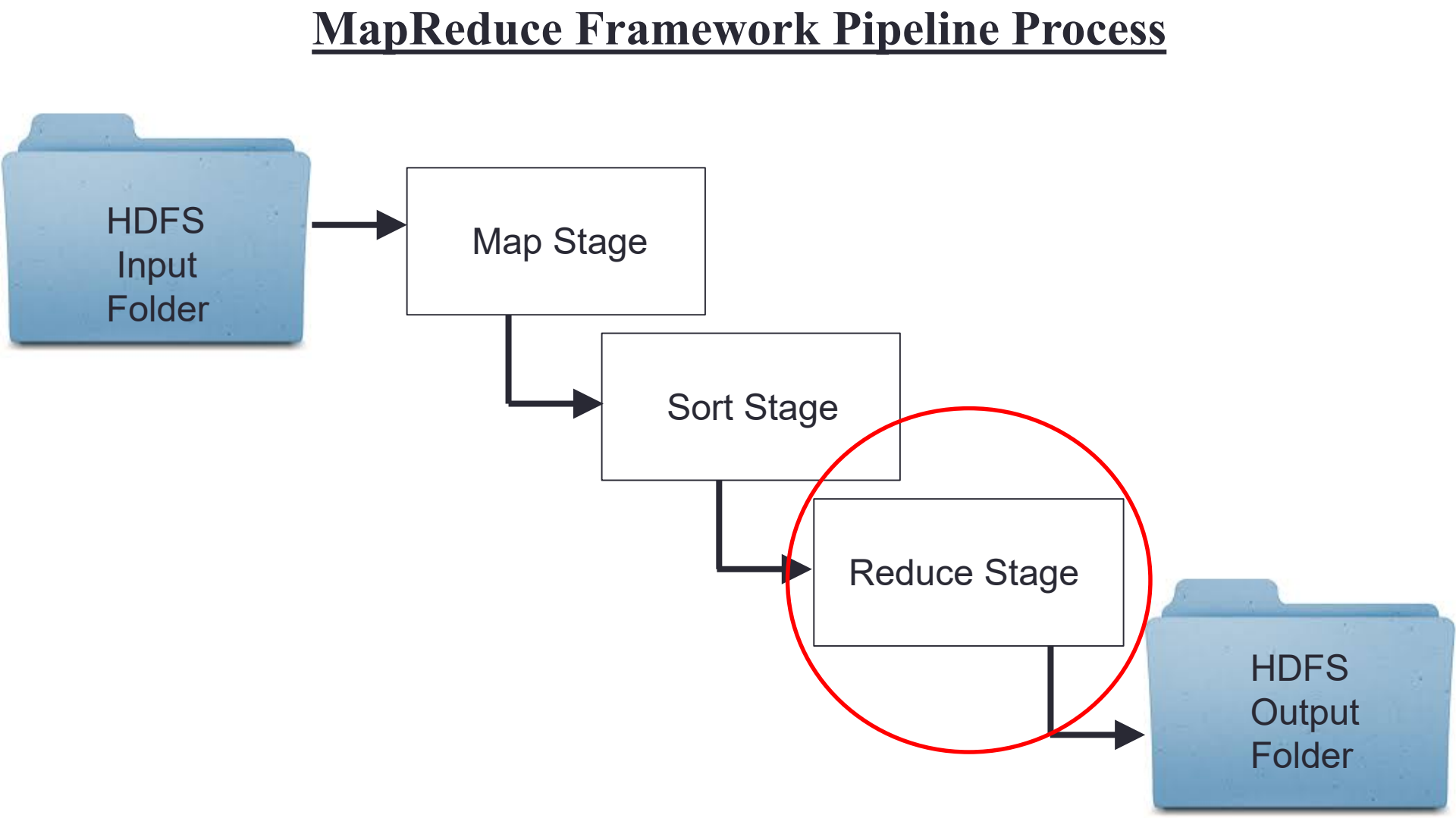
Key Concept:

- Once you have tested your map works for the entire dataset…

…the file map_simulation.txt contains the (key, value) pairs produced by your mappers for the entire dataset.

- However, these (key, value) pairs are not sorted.
- my_sort_simulation.py is a program that reads map_simulation.txt and produces sort_simulation.txt, pretty much with the same set of (key, value) pairs, but all of them sorted by the key.
- This is the way the reducers are going to receive the pairs!

Second Example: Temperatures



Second Example: Temperatures

Reduce Stage

- So, how do we process these (key, values) intermediate results?

Aberporth (1963,-3.5)
Armagh (1878,-4.2)
Bradford (1940,-4.9)
Braemar (1963,-8.6)
…
Yeovilton (2010,-4.5)

Any ideas?

Tip: Do not think in Python, think at a high-level, in English!

Second Example: Temperatures

Reduce Stage

- Well, the approach of the reducer does not differ that much from the one of the mapper.
- Let's use the same boxes once again.

Key Concept:

- We **must preserve consistency** between the portion of file we have processed so far and the content of these boxes.

Second Example: Temperatures

Reduce Stage

- Thus, at the beginning of the process of the file, cityName is " " (as we have processed no city yet), and minYear and MinTemp are set to dummy (extremely big) values.

Second Example: Temperatures

Reduce Stage

- When a new (key, value) is processed from the file, we must maintain the consistency of the boxes.
- Let's see the format each (key, value) has:

new_city (new_year, new_temp)

- **Case1:**

If new_temp does not beat k3, we maintain the boxes as they are.

Second Example: Temperatures

Reduce Stage

- When a new (key, value) is processed from the file, we must maintain the consistency of the boxes.
- Let's see the format each (key, value) has:

new_city (new_year, new_temp)

- **Case2:**

If new_temp beats k3, we update the boxes.

Second Example: Temperatures

Reduce Stage

- When the entire file has been processed, the content of our boxes is the result we wanted to compute!
- So we just print it under the format:
  - k1 (k2, k3)

Second Example: Temperatures

Reduce Stage

And that's pretty much it!

- Let's analyse the file my_reducer.py to see the actual implementation of our algorithm in Python.

Second Example: Temperatures

Reduce Stage

- The Python program (the file) contains the following 3 functions:
  1. get_key_value(line)
  2. my_reduce(input_stream, output_stream)
  3. my_main( )

- Executing the program means executing the function my_main() first

Second Example: Temperatures

Reduce Stage

1. get_key_value(line)

- Given a line of text in our file representing a (key, value), this function returns the city, year, temp.

Second Example: Temperatures

Reduce Stage

2. my_reduce(input_streaming, output_streaming)

- This function implements the English algorithm we explained before:

1. Initialises city to “”. Initialises year and temp to dummy values.
2. For loop, reading one line from the file at each time…
   - Get (key, value) from the line (using ‘get_key_value’).
   - If the new temperature beats the best one, then updates the boxes.
3. Once the file has been entirely processed:
   - Print the content of the boxes as the final output.

Second Example: Temperatures

Reduce Stage

3. my_main()

- Provides two operating modes:
  - Mode 1 → Testing:
    - Start using this mode to test if your program works. Practise it with "sort_simulation.txt" to see what your program outputs.
  - Mode 2 → For the actual MapReduce:
    - Once you know your program works, use this mode for the actual MapReduce job, as it uses the stdin and stdout standard I/O channels.

- In both modes, the function triggers the execution of the function my_reduce using the input and output streaming selected by the mode.

Outline

1. First Example: Shakespeare Word Average.
2. Second Example: Temperatures.

Thank you for your attention!