Are you ready for the future?

hadoop Map Reduce

High Performance Solutions, Backend Development and Integration Services, Embedded Software Development, Big Data (Visualization, Architecture, Science), Business Intelligence & Analytics

Globant
we are ready

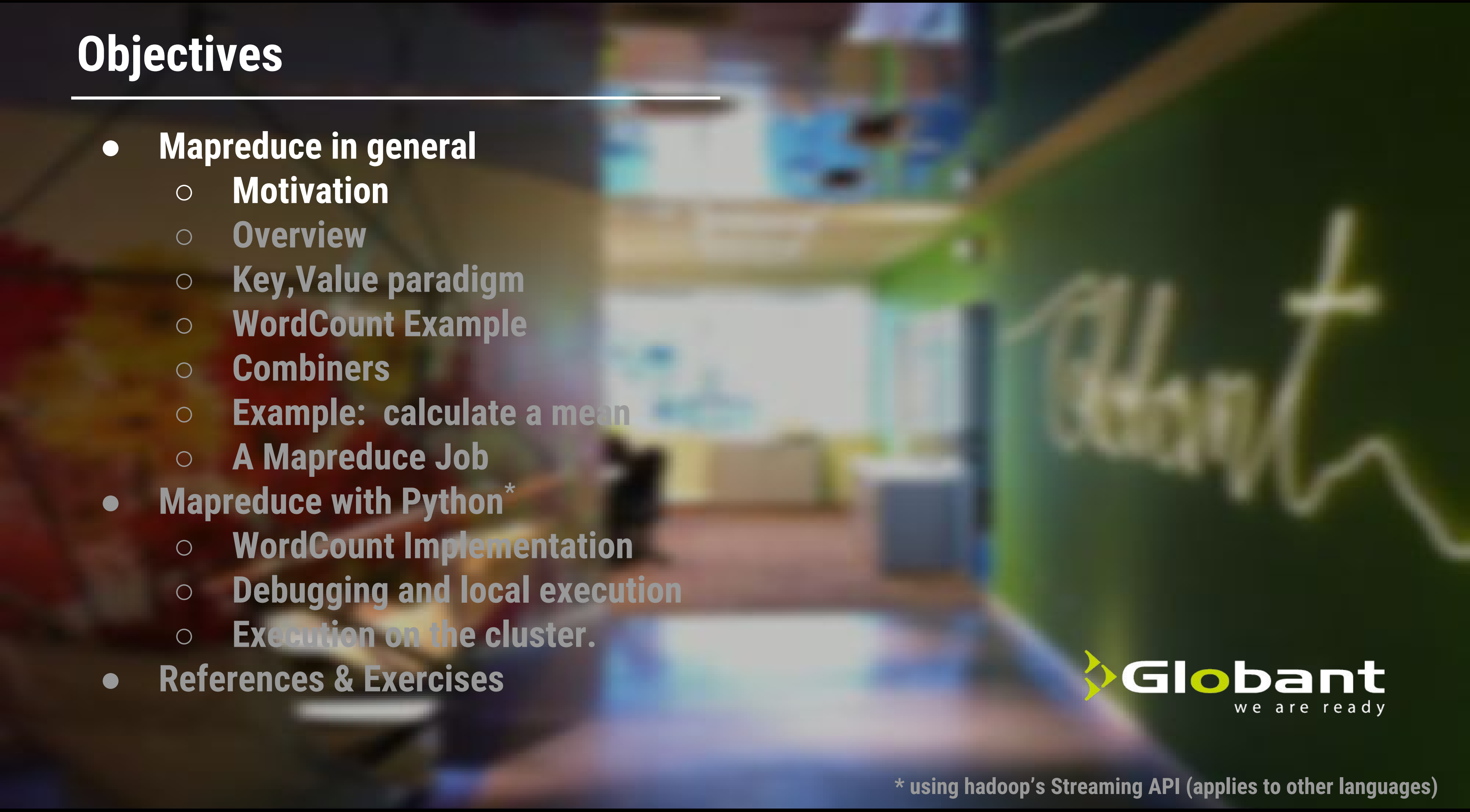
BIG DATA & HIGH PERFORMANCE

AUG 2014

Objectives

- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

* using hadoop’s Streaming API (applies to other languages)



Objectives

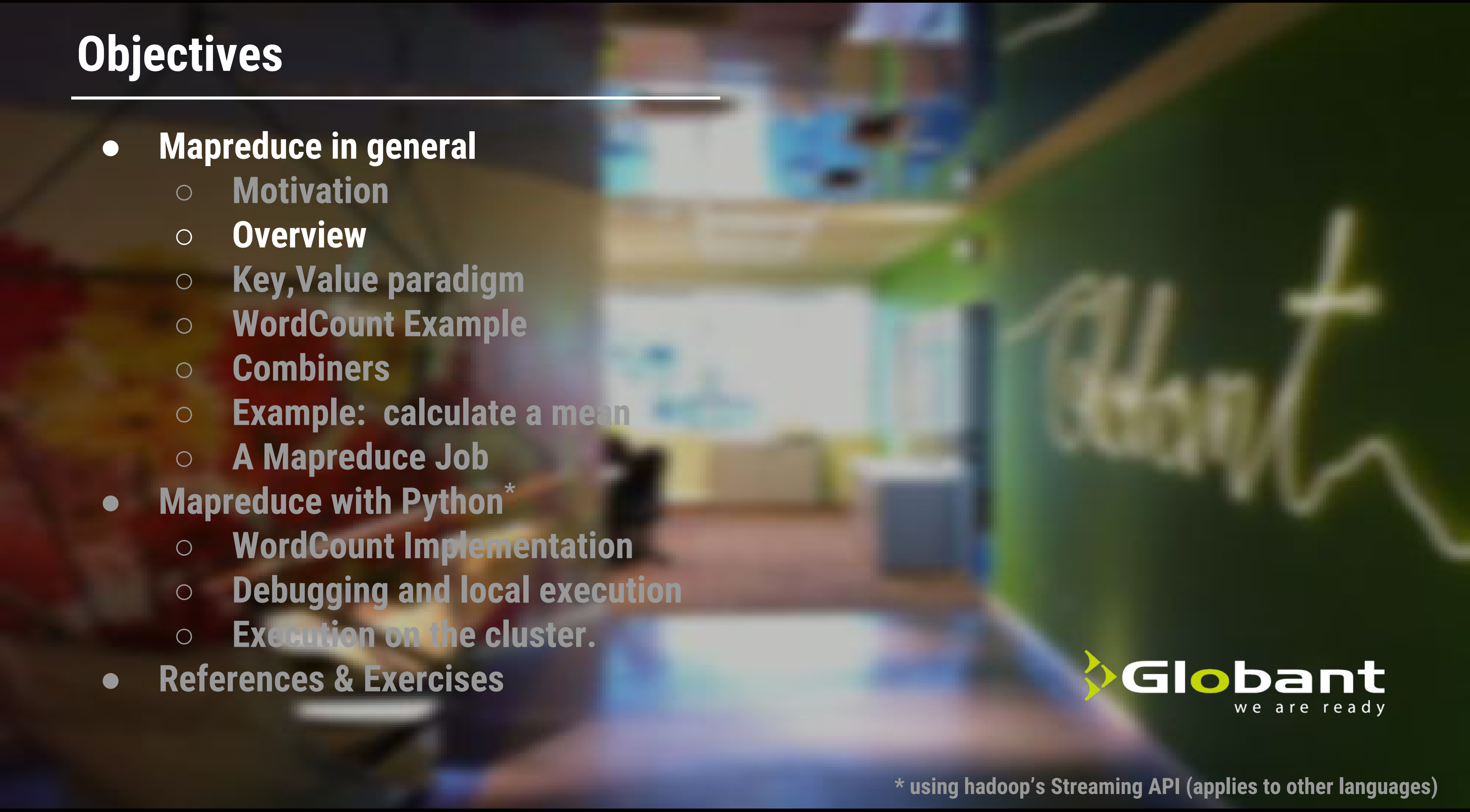
- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

* using hadoop's Streaming API (applies to other languages)

MapReduce - Motivation (1)

- Processors steadily increased speed but Disk IO speed “almost remained constant”.
- Moore’s Law reaching limits.
- Nowadays, disk storage is inexpensive and many-terabyte problems are common.
- Supercomputers (vertical scalability) are expensive but same amount of processors, RAM and disk over many machines (horizontal scalability) is cheap.

MapReduce - Motivation (2)



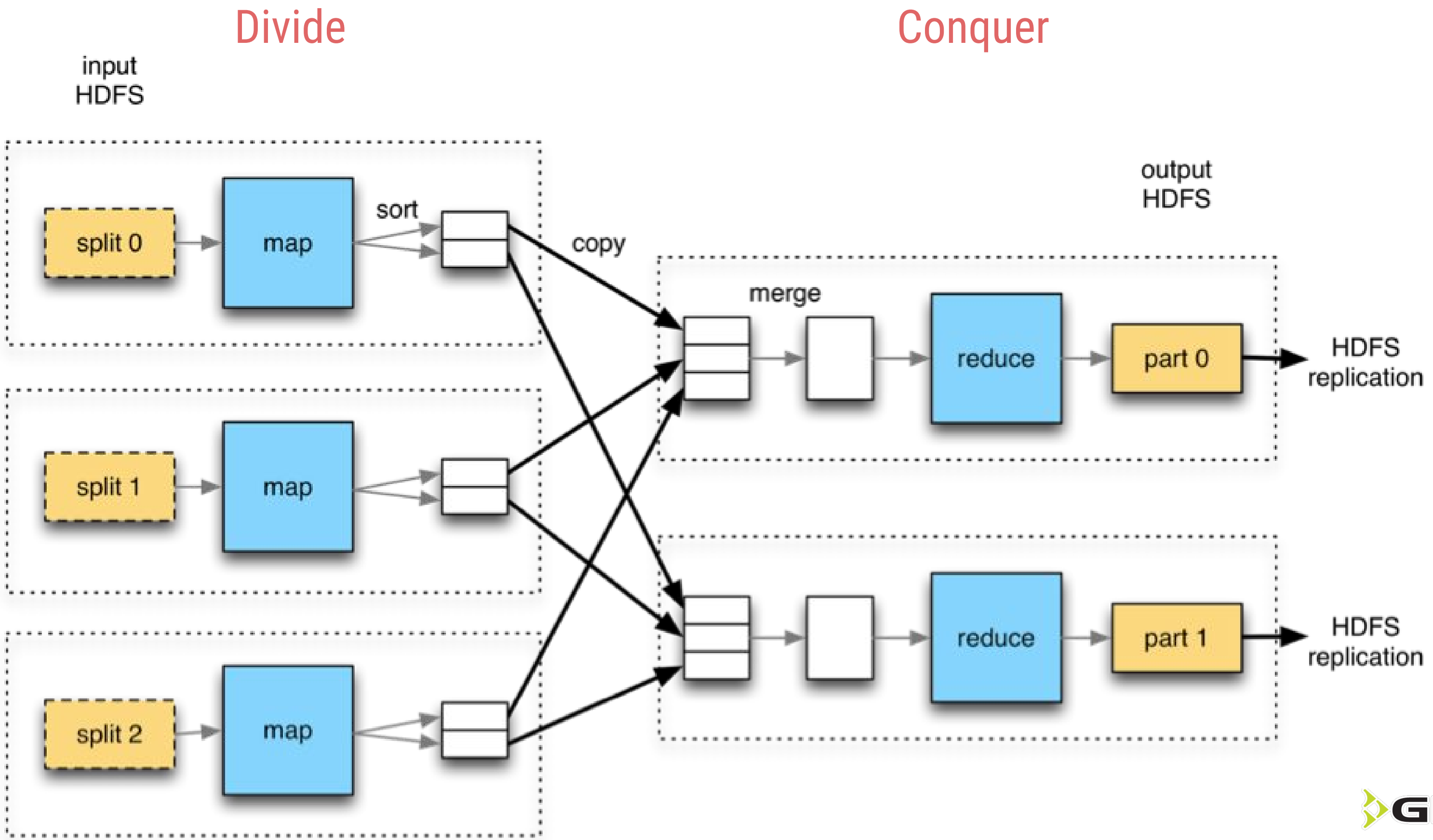
Objectives

- Mapreduce in general
  - ○ Motivation
  - ○ Overview
  - ○ Key,Value paradigm
  - ○ WordCount Example
  - ○ Combiners
  - ○ Example: calculate a mean
  - ○ A Mapreduce Job
- Mapreduce with Python*
  - ○ WordCount Implementation
  - ○ Debugging and local execution
  - ○ Execution on the cluster.
- References & Exercises

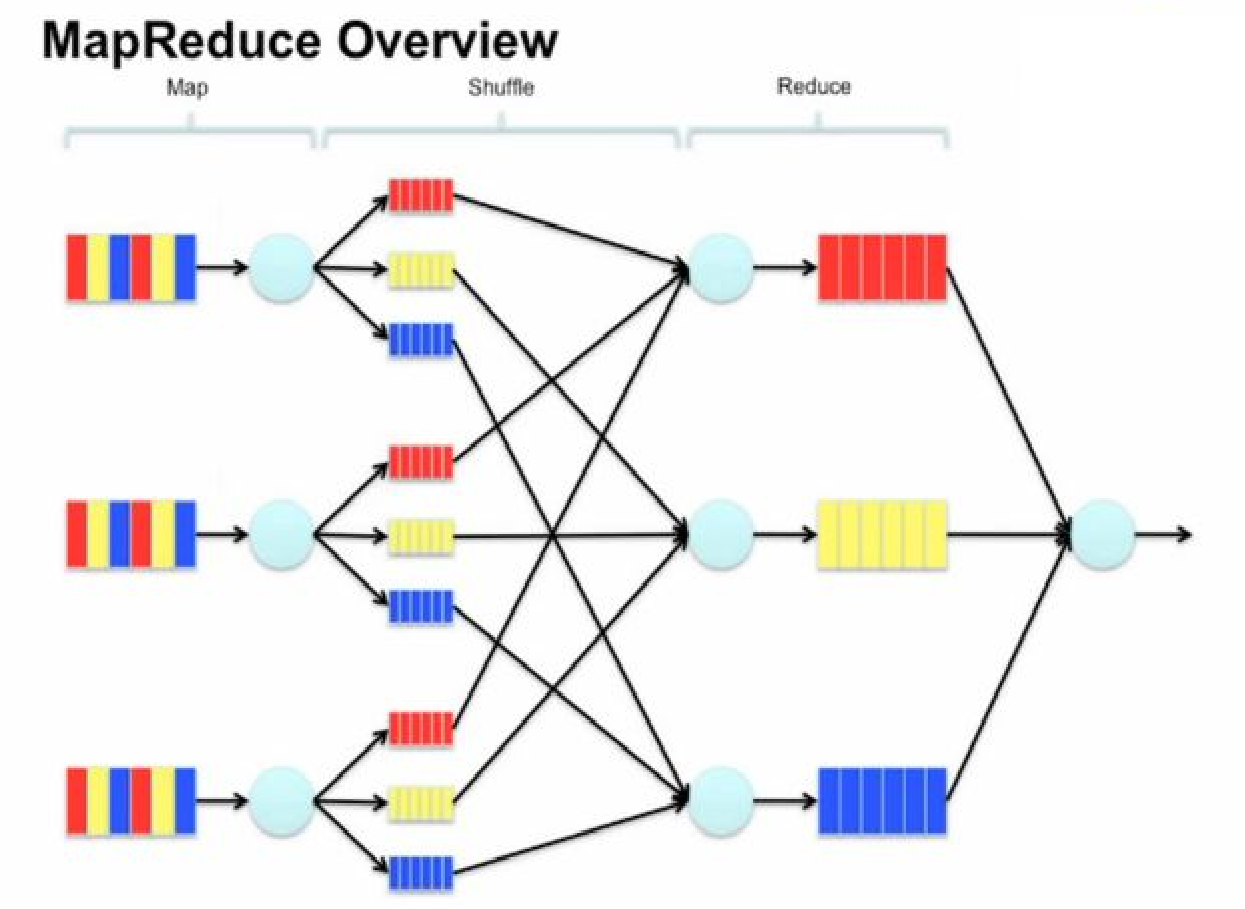
Globant
we are ready

* using hadoop's Streaming API (applies to other languages)

MapReduce - Overview (1)



MapReduce - Overview (2)



Objectives

- Mapreduce in general
  - ○ Motivation
  - ○ Overview
  - ○ Key,Value paradigm
  - ○ WordCount Example
  - ○ Combiners
  - ○ Example: calculate a mean
  - ○ A Mapreduce Job
- Mapreduce with Python*
  - ○ WordCount Implementation
  - ○ Debugging and local execution
  - ○ Execution on the cluster.
- References & Exercises

* using hadoop's Streaming API (applies to other languages)

MapReduce - Key,Value Paradigm (1)

- Map and Reduce tasks consume and emit "(key,value)" pairs
- keys and values can be of any data type: string, int, lists, custom objects,...
- Example:

input: $(k_{\alpha}, value_{\alpha})$ output: $(k_{B}, value_{B})$

Change of alphabets in subindices indicate possible different data types

e.g.,

input: `(5, "this is my pc")` output: `("pc", 1)`

input: `(10, "cat")` output: `("letters", ["c","a","t"])`

MapReduce - Key,Value Paradigm (2)

input

(k_{α}, v_{α})

$(k_{\text{ß}}, v_{\text{ß}})$

(k_{γ}, v_{γ})

. . .

(k_{ω}, v_{ω})

MapReduce - Key,Value Paradigm (3)

input

(k_{α}, v_{α})

(k_{β}, v_{β})

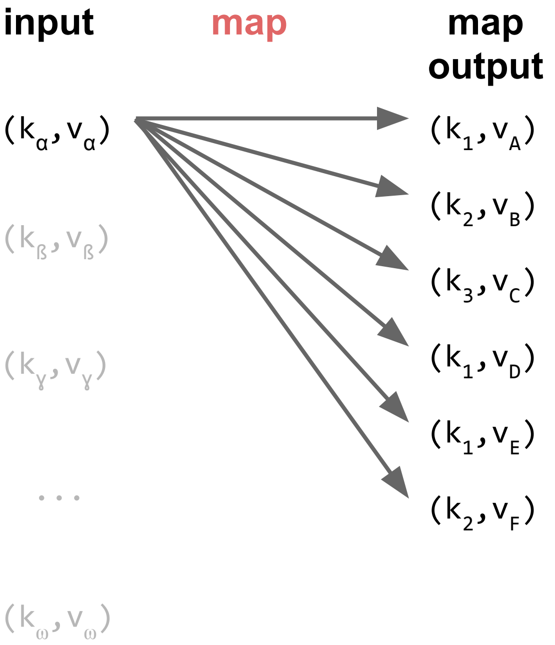
(k_{γ}, v_{γ})

. . .

(k_{ω}, v_{ω})

Let’s forget about these for a moment

MapReduce - Key,Value Paradigm (4)



input

map

map output

(k_α, v_α)

$(k_ß, v_ß)$

(k_γ, v_γ)

. . .

(k_ω, v_ω)

(k_1, v_A)

(k_2, v_B)

(k_3, v_C)

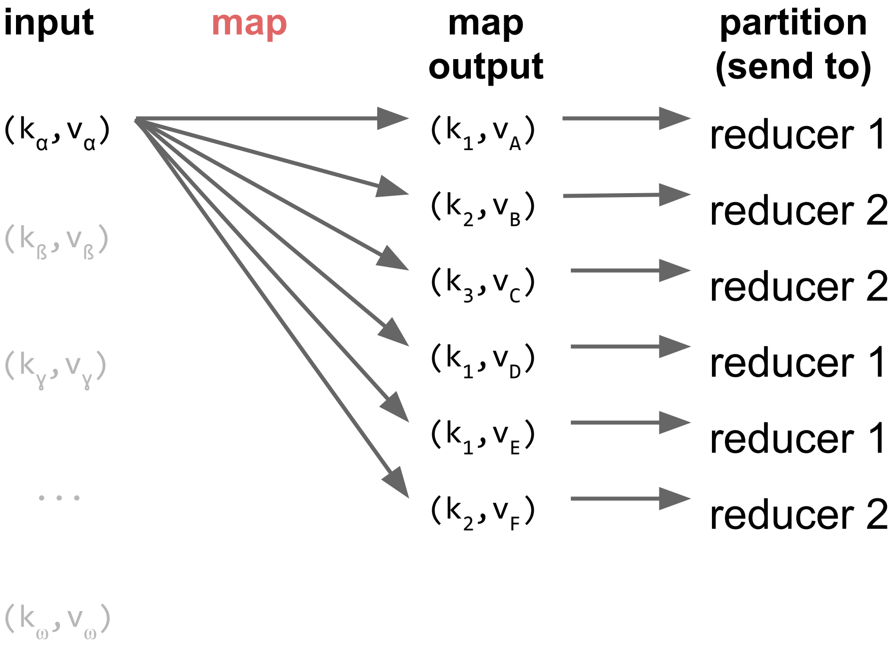
(k_1, v_D)

(k_1, v_E)

(k_2, v_F)

Change of alphabets in subindices indicate possible different data types

MapReduce - Key,Value Paradigm (5)



input

map

map output

partition (send to)

(k_α, v_α)

$(k_ß, v_ß)$

(k_γ, v_γ)

. . .

(k_ω, v_ω)

(k_1, v_A) → reducer 1

(k_2, v_B) → reducer 2

(k_3, v_C) → reducer 2

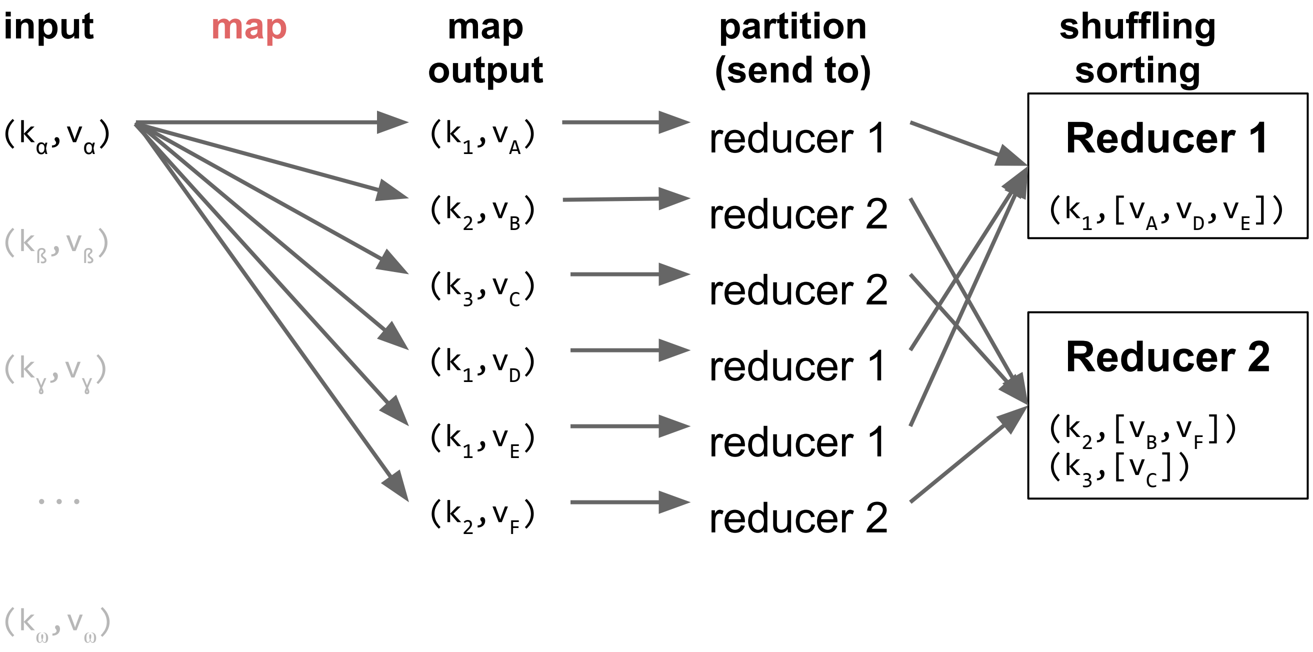
(k_1, v_D) → reducer 1

(k_1, v_E) → reducer 1

(k_2, v_F) → reducer 2

Change of alphabets in subindices indicate possible different data types

MapReduce - Key,Value Paradigm (6)



input

map

map output

partition (send to)

shuffling sorting

(k_α, v_α)

(k_β, v_β)

(k_γ, v_γ)

...

(k_ω, v_ω)

(k_1, v_A) reducer 1

(k_2, v_B) reducer 2

(k_3, v_C) reducer 2

(k_1, v_D) reducer 1

(k_1, v_E) reducer 1

(k_2, v_F) reducer 2

Reducer 1

$(k_1, [v_A, v_D, v_E])$

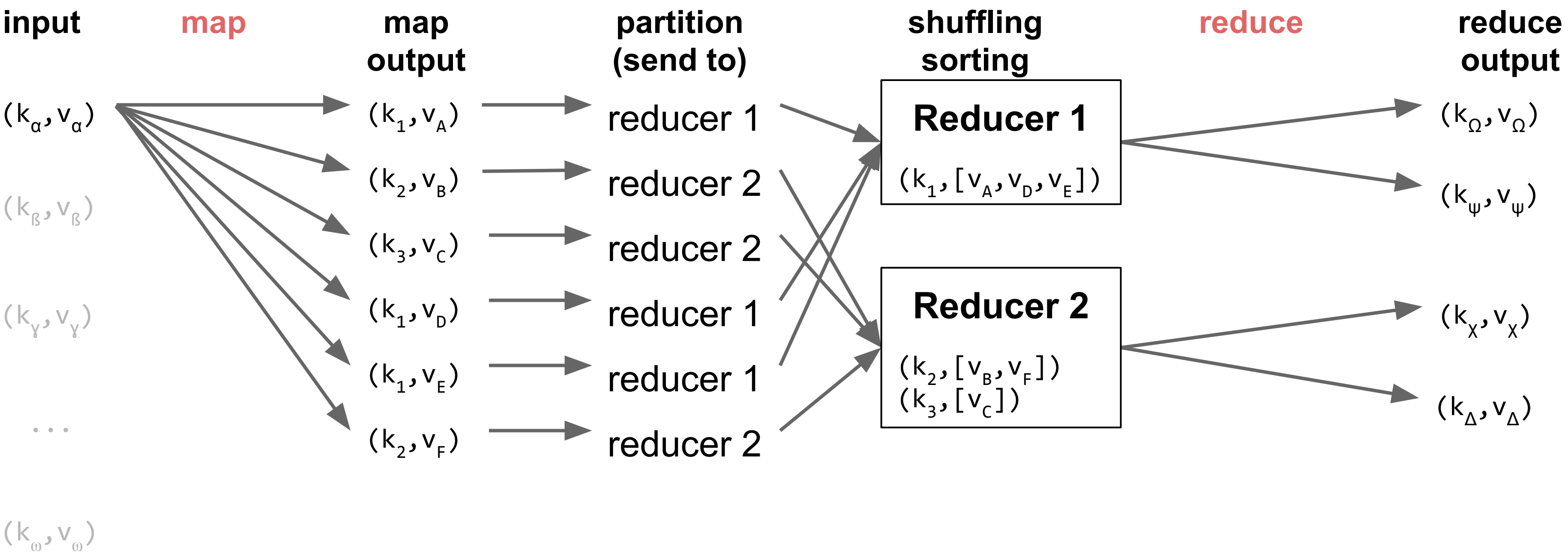
Reducer 2

$(k_2, [v_B, v_F])$

$(k_3, [v_C])$

Change of alphabets in subindices indicate possible different data types

MapReduce - Key,Value Paradigm (7)



input

map

map output

partition (send to)

shuffling sorting

reduce

reduce output

(k_α, v_α)

(k_β, v_β)

(k_γ, v_γ)

. . .

(k_ω, v_ω)

(k_1, v_A) → reducer 1

(k_2, v_B) → reducer 2

(k_3, v_C) → reducer 2

(k_1, v_D) → reducer 1

(k_1, v_E) → reducer 1

(k_2, v_F) → reducer 2

Reducer 1

$(k_1, [v_A, v_D, v_E])$

Reducer 2

$(k_2, [v_B, v_F])$

$(k_3, [v_C])$

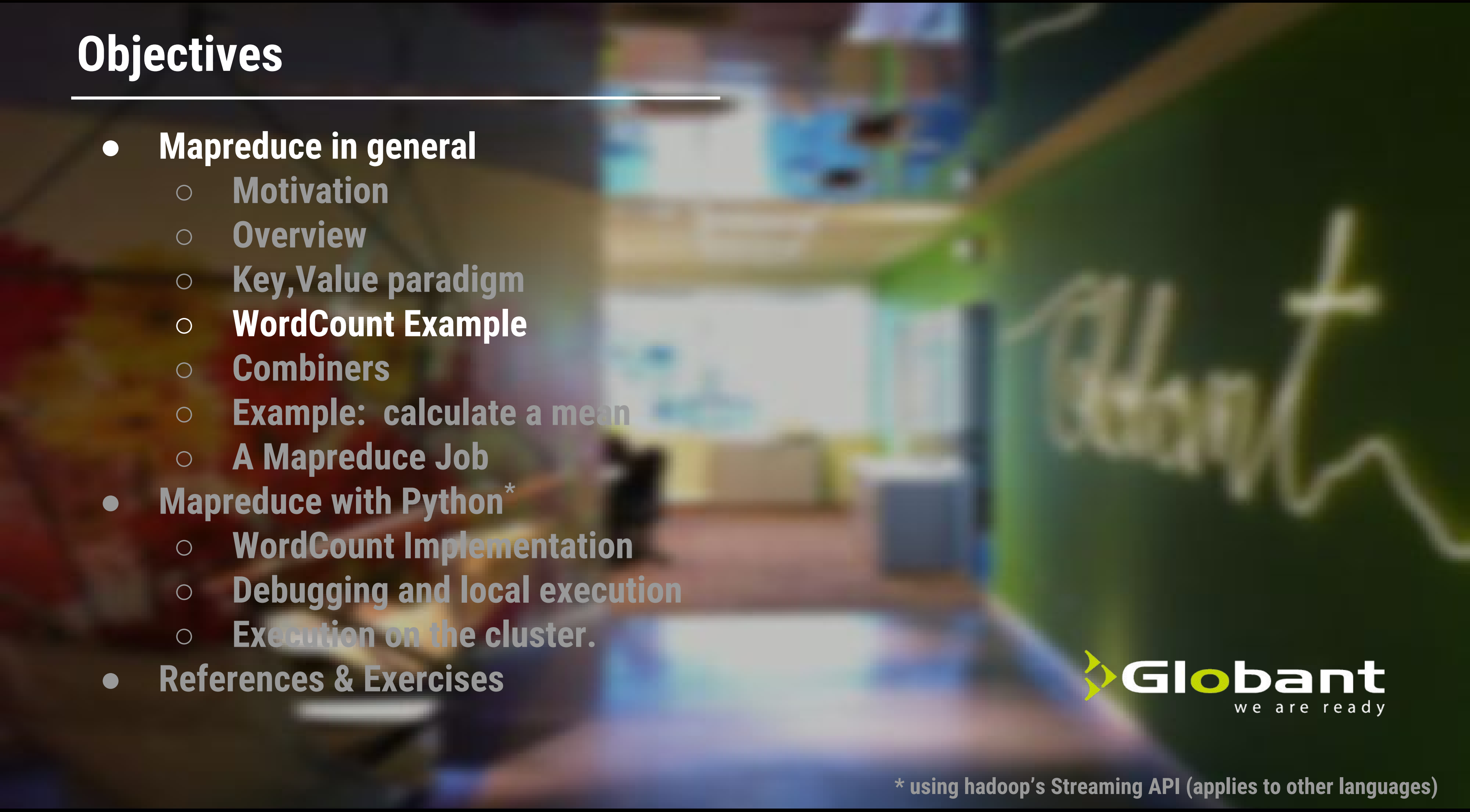
(k_Ω, v_Ω)

(k_Ψ, v_Ψ)

(k_X, v_X)

(k_Δ, v_Δ)

Change of alphabets in subindices indicate possible different data types



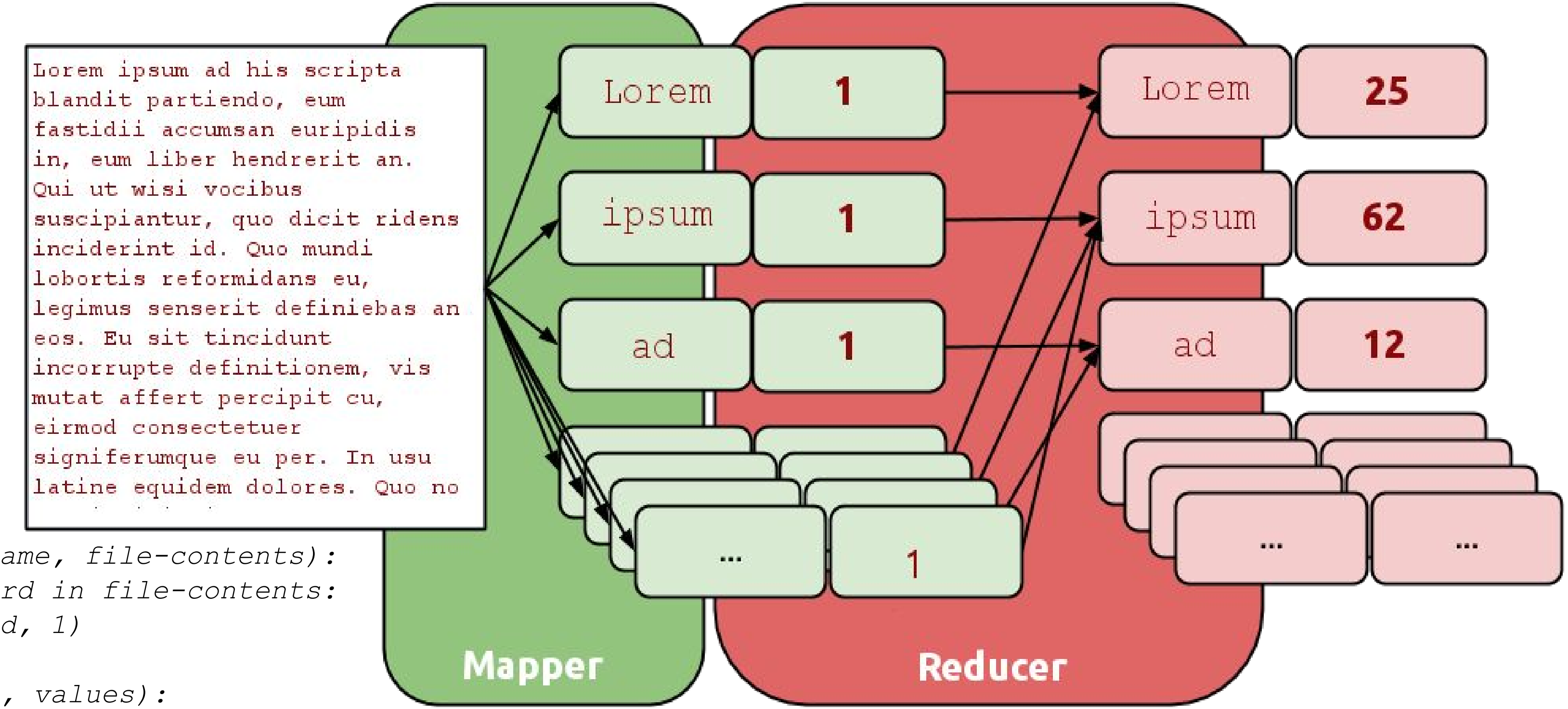
Objectives

- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

Globant
we are ready

* using hadoop's Streaming API (applies to other languages)

MapReduce - WordCount is hadoop’s “Hello World”



Lorem ipsum ad his scripta
blandit partiendo, eum
fastidii accumsan euripidis
in, eum liber hendrerit an.
Qui ut wisi vocibus
suscipiantur, quo dicit ridens
inciderint id. Quo mundi
lobortis reformidans eu,
legimus senserit definiebas an
eos. Eu sit tincidunt
incorrupte definitionem, vis
mutat affert percipit cu,
eirmod consectetuer
signiferumque eu per. In usu
latine equidem dolores. Quo no

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

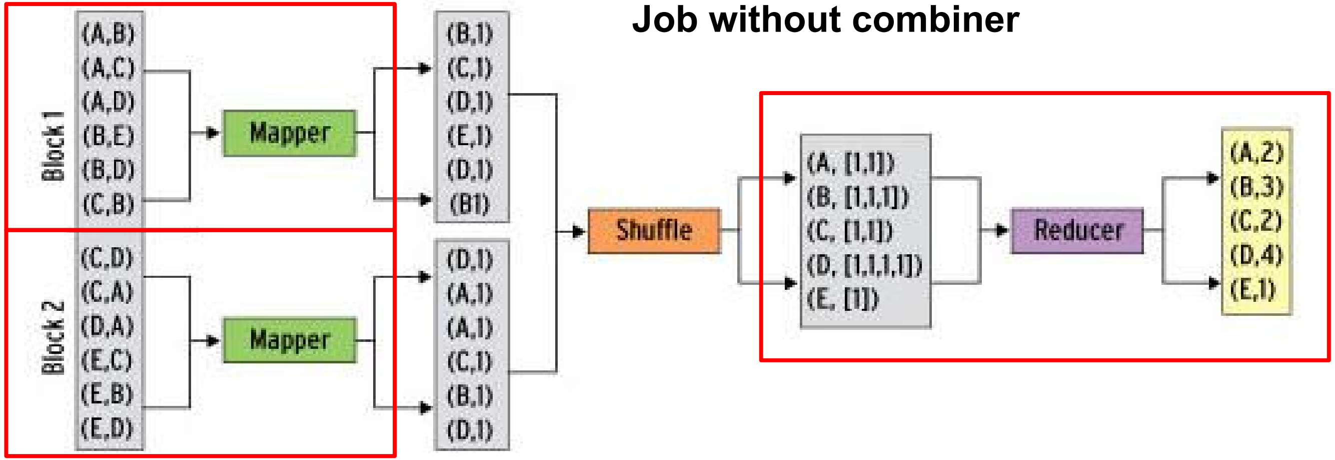
Objectives

- Mapreduce in general
    - Motivation
    - Overview
    - Key,Value paradigm
    - WordCount Example
    - Combiners
    - Example: calculate a mean
    - A Mapreduce Job
- Mapreduce with Python*
    - WordCount Implementation
    - Debugging and local execution
    - Execution on the cluster.
- References & Exercises

* using hadoop's Streaming API (applies to other languages)

MapReduce - Jobs with and without Combiner (1)

Job without combiner



Block 1

(A,B)
(A,C)
(A,D)
(B,E)
(B,D)
(C,B)

Mapper

(B,1)
(C,1)
(D,1)
(E,1)
(D,1)
(B1)

Block 2

(C,D)
(C,A)
(D,A)
(E,C)
(E,B)
(E,D)

Mapper

(D,1)
(A,1)
(A,1)
(C,1)
(B,1)
(D,1)

Shuffle

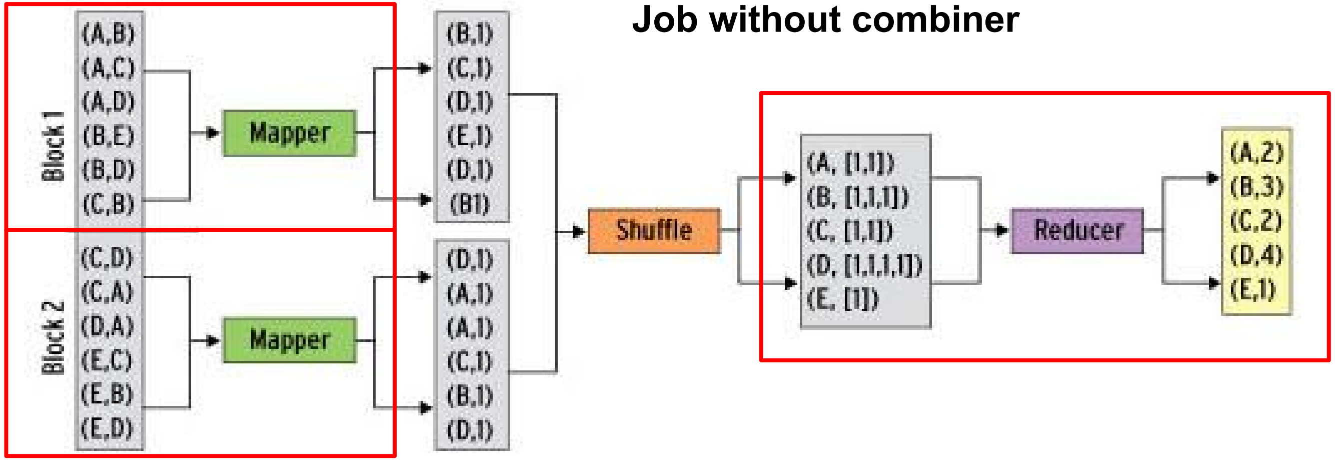
(A, [1,1])
(B, [1,1,1])
(C, [1,1])
(D, [1,1,1,1])
(E, [1])

Reducer

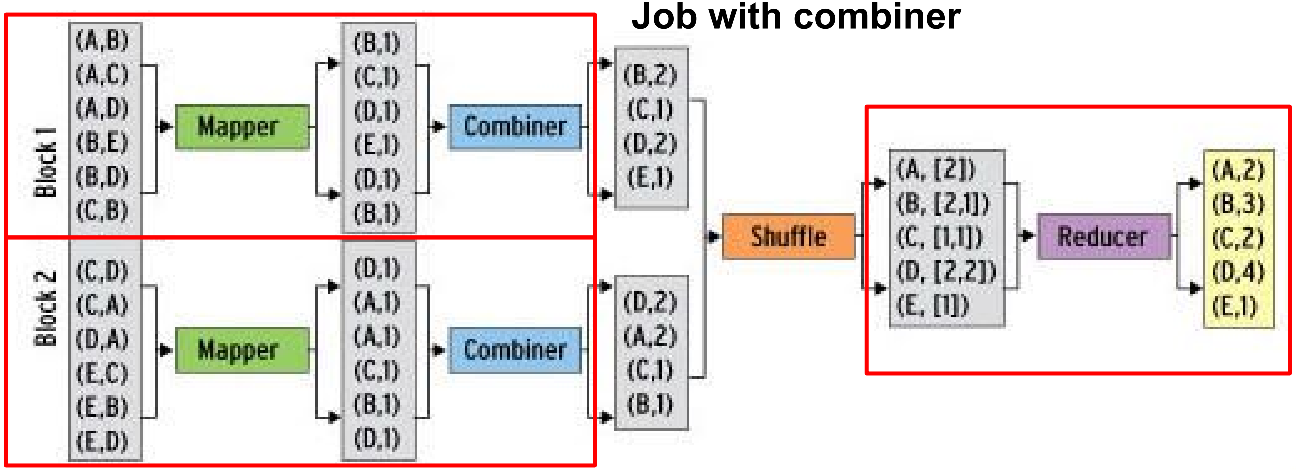
(A,2)
(B,3)
(C,2)
(D,4)
(E,1)

MapReduce - Jobs with and without Combiner (2)

Job without combiner



Job with combiner



Data sent through the network is much less!
Job significantly faster!

MapReduce - Jobs with and without Combiner (3)

Example,

Say we want to calculate the mean measurement in AR:

node1 mappers outputs:

```
AR,500
AR,1000
AR,750
```

node2 mappers outputs:

```
AR,500
AR,800
```

node3 mappers outputs:

```
AR,100
AR,200
```

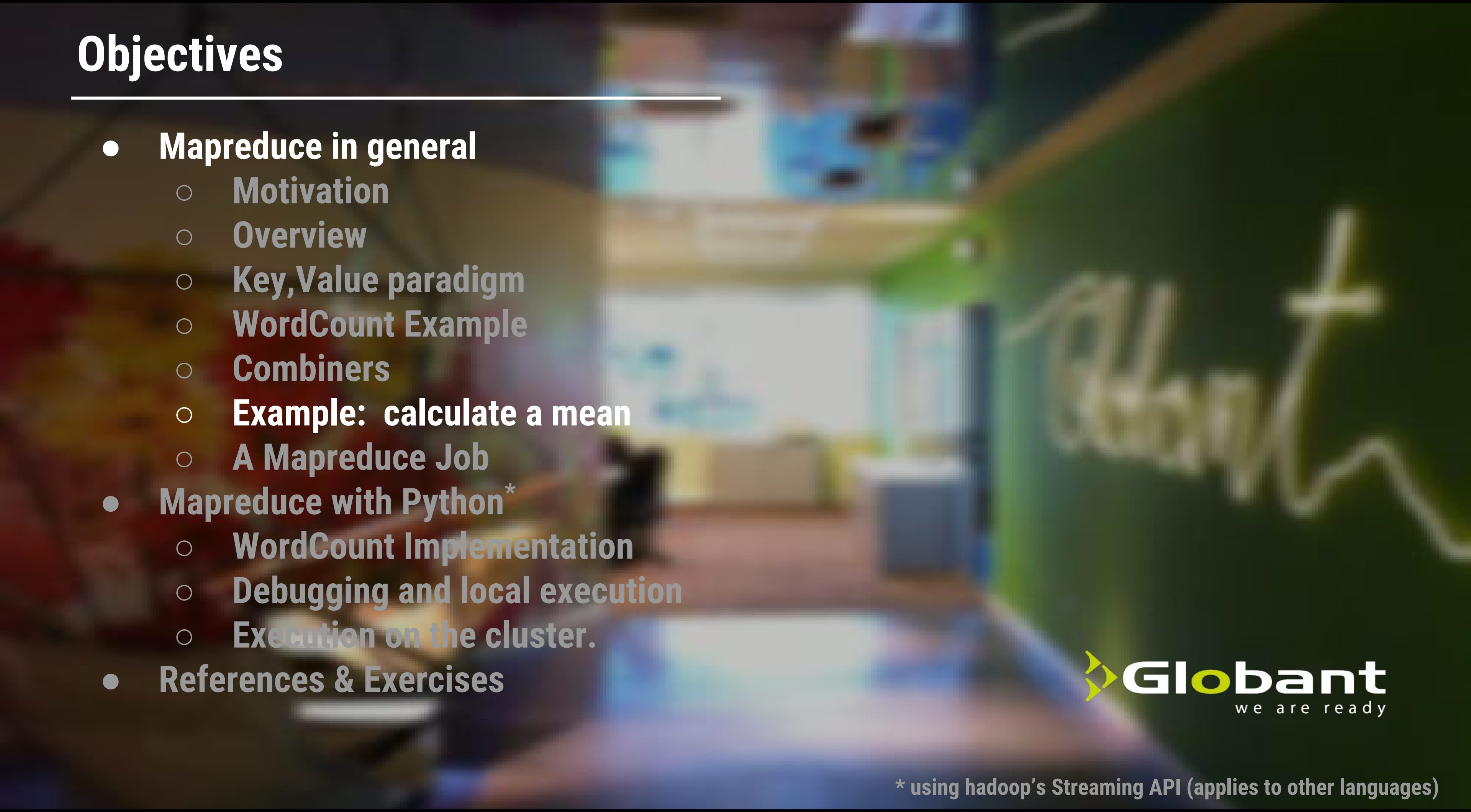
Without Combiner:

```
(500 + 1000 + 750 + 500 + 800 + 100 + 200) / 7  =   550
```

With Combiners (Common Pitfall):

```
(500+1000+750)/3 + (500+800)/2 + (100+200)/2     =
     (750         +     650       +     150)     / 3 =  516,66
```

Objectives



- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - **Example: calculate a mean**
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

Globant
we are ready

* using hadoop's Streaming API (applies to other languages)

MapReduce - Example: Calculate a mean (1)

Task: using mapreduce calculate the mean of the variables height, weight and age (column-wise).

txt input file		
Height (cm)	Weight (Kg)	Age (yrs)
180	80	20
160	65	25
175	85	50
192	100	60
181	78	35
150	50	13

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Idea:
accumulate partial sums and counts, leaving the calculation of the average to the reducer

use as value: <partial sum>_<partial count>

MapReduce - Example: Calculate a mean (2)

Using 3 mappers, no combiners and 2 reducers

input splits

```
{0: “180 80 20
     160 65 25
     175 85 50”}
```

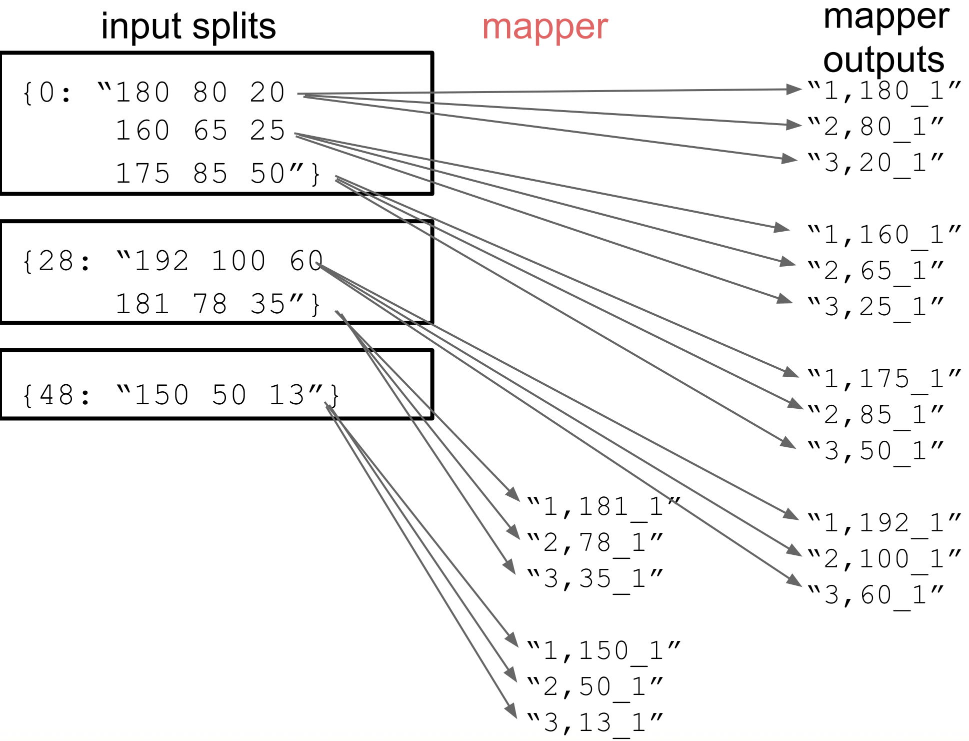
```
{28: “192 100 60
      181 78 35”}
```

```
{48: “150 50 13”}
```

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (3)

Using 3 mappers, no combiners and 2 reducers



input splits

mapper

mapper outputs

{0: “180 80 20
160 65 25
175 85 50”}

{28: “192 100 60
181 78 35”}

{48: “150 50 13”}

“1,180_1”
“2,80_1”
“3,20_1”

“1,160_1”
“2,65_1”
“3,25_1”

“1,175_1”
“2,85_1”
“3,50_1”

“1,192_1”
“2,100_1”
“3,60_1”

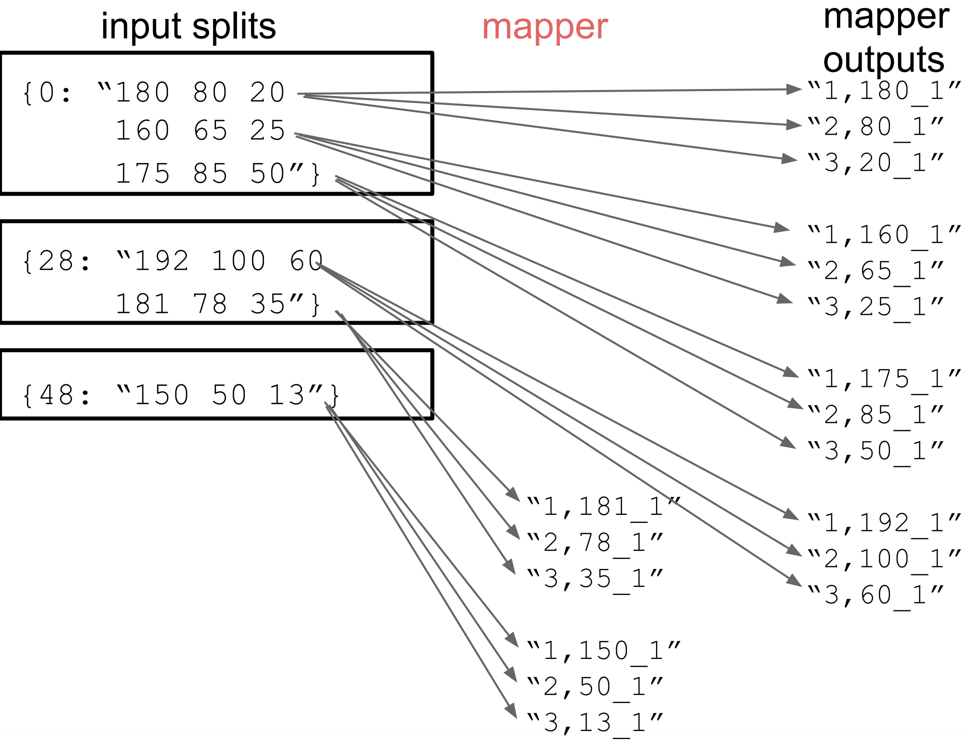
“1,181_1”
“2,78_1”
“3,35_1”

“1,150_1”
“2,50_1”
“3,13_1”

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (4)

Using 3 mappers, no combiners and 2 reducers



input splits

mapper

mapper outputs

{0: "180 80 20
160 65 25
175 85 50"}

{28: "192 100 60
181 78 35"}

{48: "150 50 13"}

"1,180_1"
"2,80_1"
"3,20_1"

"1,160_1"
"2,65_1"
"3,25_1"

"1,175_1"
"2,85_1"
"3,50_1"

"1,192_1"
"2,100_1"
"3,60_1"

"1,181_1"
"2,78_1"
"3,35_1"

"1,150_1"
"2,50_1"
"3,13_1"

AVOID THIS!!!

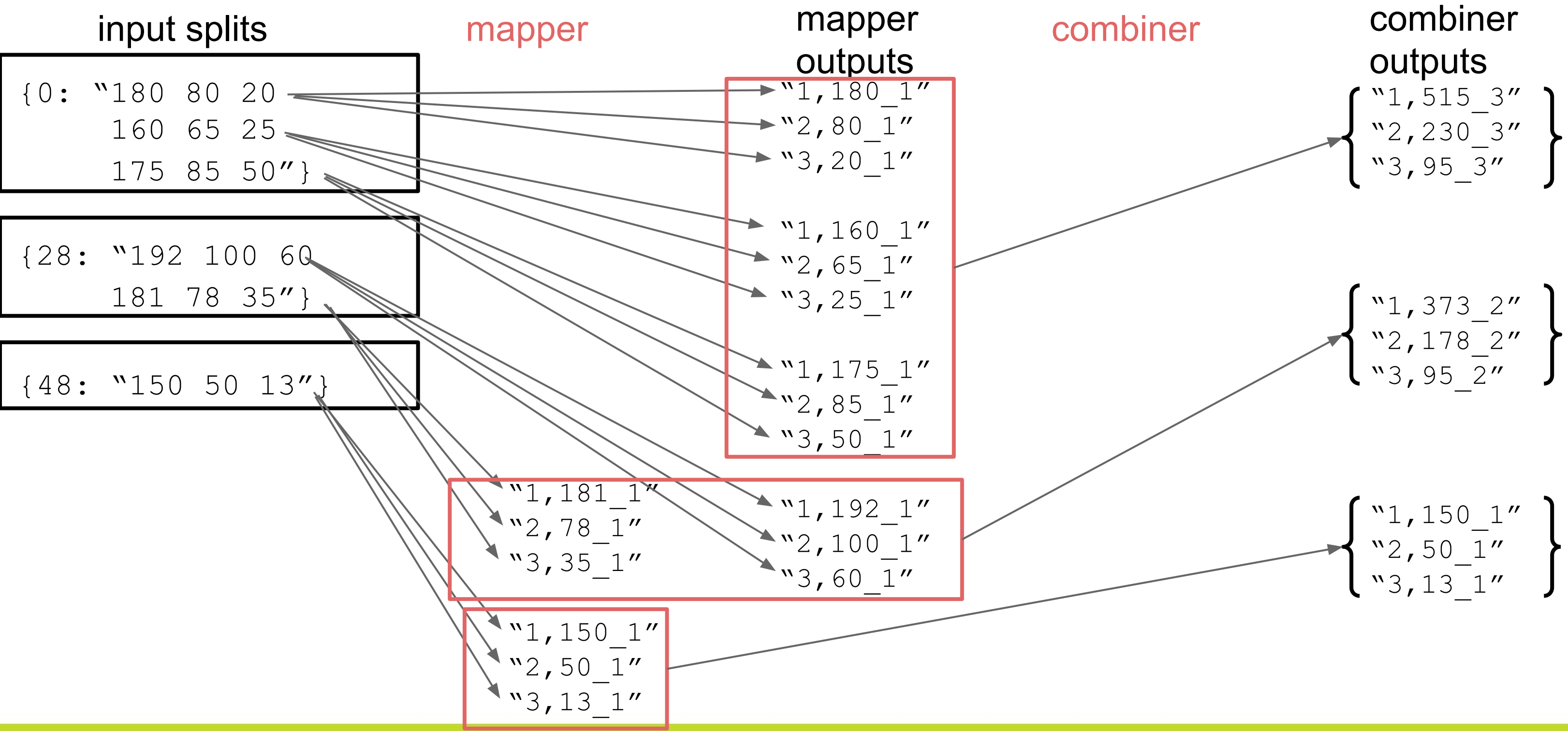
mapper's outputs are already larger than input!

Use combiners to summarize (or accumulators in the mappers if streaming API).

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (5)

Using 3 mappers, combiners and 2 reducers



input splits

```
{0: "180 80 20
     160 65 25
     175 85 50"}

{28: "192 100 60
      181 78 35"}

{48: "150 50 13"}
```

mapper

mapper outputs

```
"1,180_1"
"2,80_1"
"3,20_1"

"1,160_1"
"2,65_1"
"3,25_1"

"1,175_1"
"2,85_1"
"3,50_1"
```

```
"1,181_1"        "1,192_1"
"2,78_1"         "2,100_1"
"3,35_1"         "3,60_1"
```

```
"1,150_1"
"2,50_1"
"3,13_1"
```

combiner

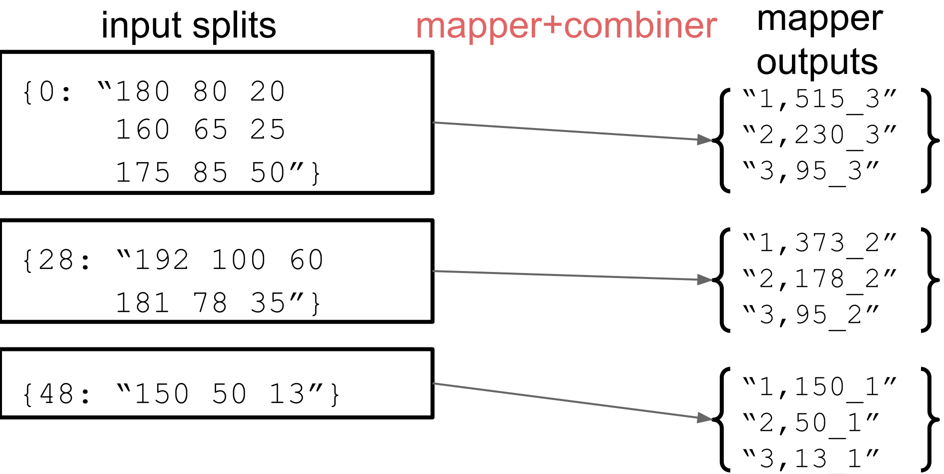
combiner outputs

```
{ "1,515_3"
  "2,230_3"
  "3,95_3" }

{ "1,373_2"
  "2,178_2"
  "3,95_2" }

{ "1,150_1"
  "2,50_1"
  "3,13_1" }
```

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (6)

Using 3 mappers, combiners and 2 reducers



input splits

mapper+combiner

mapper outputs

```
{0: “180 80 20
     160 65 25
     175 85 50”}
```

```
{28: “192 100 60
      181 78 35”}
```

```
{48: “150 50 13”}
```

```
{ “1,515_3”
  “2,230_3”
  “3,95_3” }
```

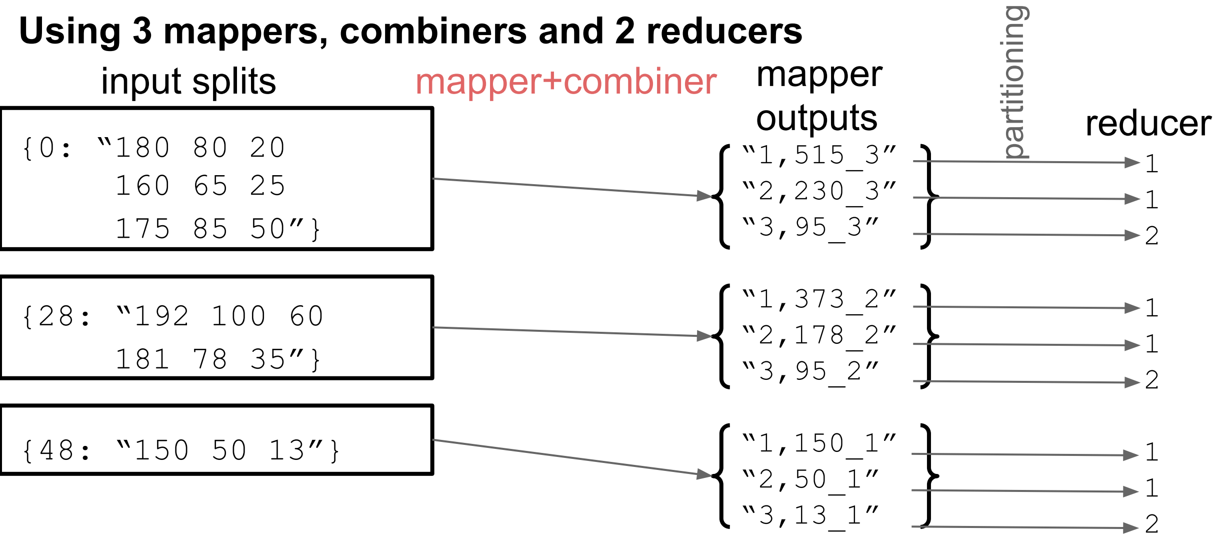
```
{ “1,373_2”
  “2,178_2”
  “3,95_2” }
```

```
{ “1,150_1”
  “2,50_1”
  “3,13_1” }
```

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (7)

Using 3 mappers, combiners and 2 reducers



input splits

mapper+combiner

mapper outputs

partitioning

reducer

```
{0: “180 80 20
     160 65 25
     175 85 50”}
```

“1,515_3” → 1
“2,230_3” → 1
“3,95_3” → 2

```
{28: “192 100 60
      181 78 35”}
```

“1,373_2” → 1
“2,178_2” → 1
“3,95_2” → 2

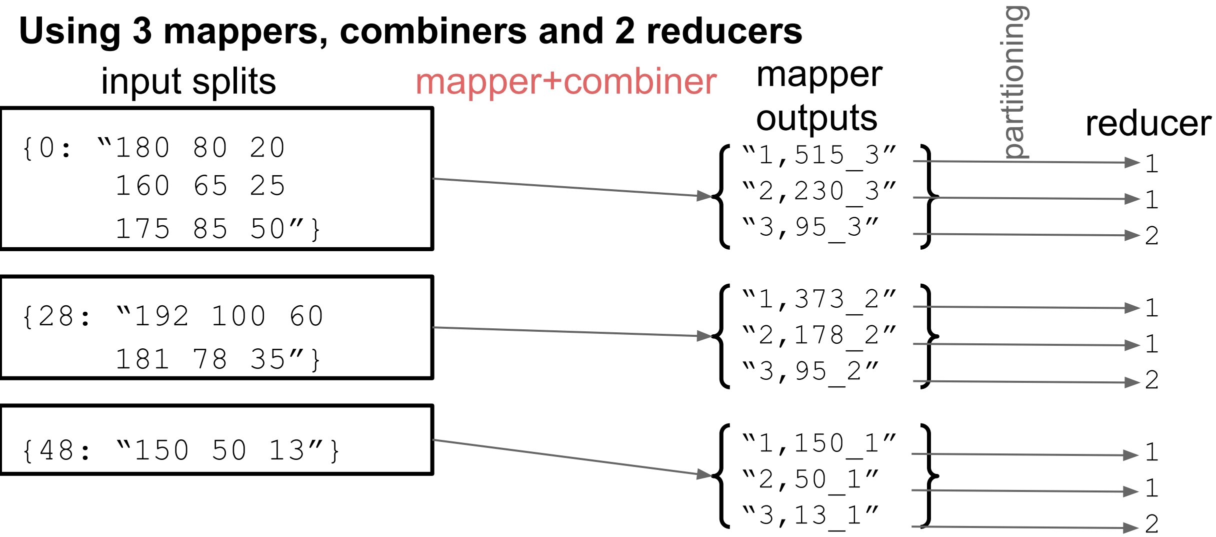
```
{48: “150 50 13”}
```

“1,150_1” → 1
“2,50_1” → 1
“3,13_1” → 2

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (8)

Using 3 mappers, combiners and 2 reducers



input splits

mapper+combiner

mapper outputs

partitioning

reducer

```
{0: "180 80 20
     160 65 25
     175 85 50"}
```

```
"1,515_3"  → 1
"2,230_3"  → 1
"3,95_3"   → 2
```

```
{28: "192 100 60
      181 78 35"}
```

```
"1,373_2"  → 1
"2,178_2"  → 1
"3,95_2"   → 2
```

```
{48: "150 50 13"}
```

```
"1,150_1"  → 1
"2,50_1"   → 1
"3,13_1"   → 2
```

Shuffling and sorting

reducer 1:

```
{1,[150_1,373_2,515_3]}
{2,[178_2,230_3,50_1]}
```

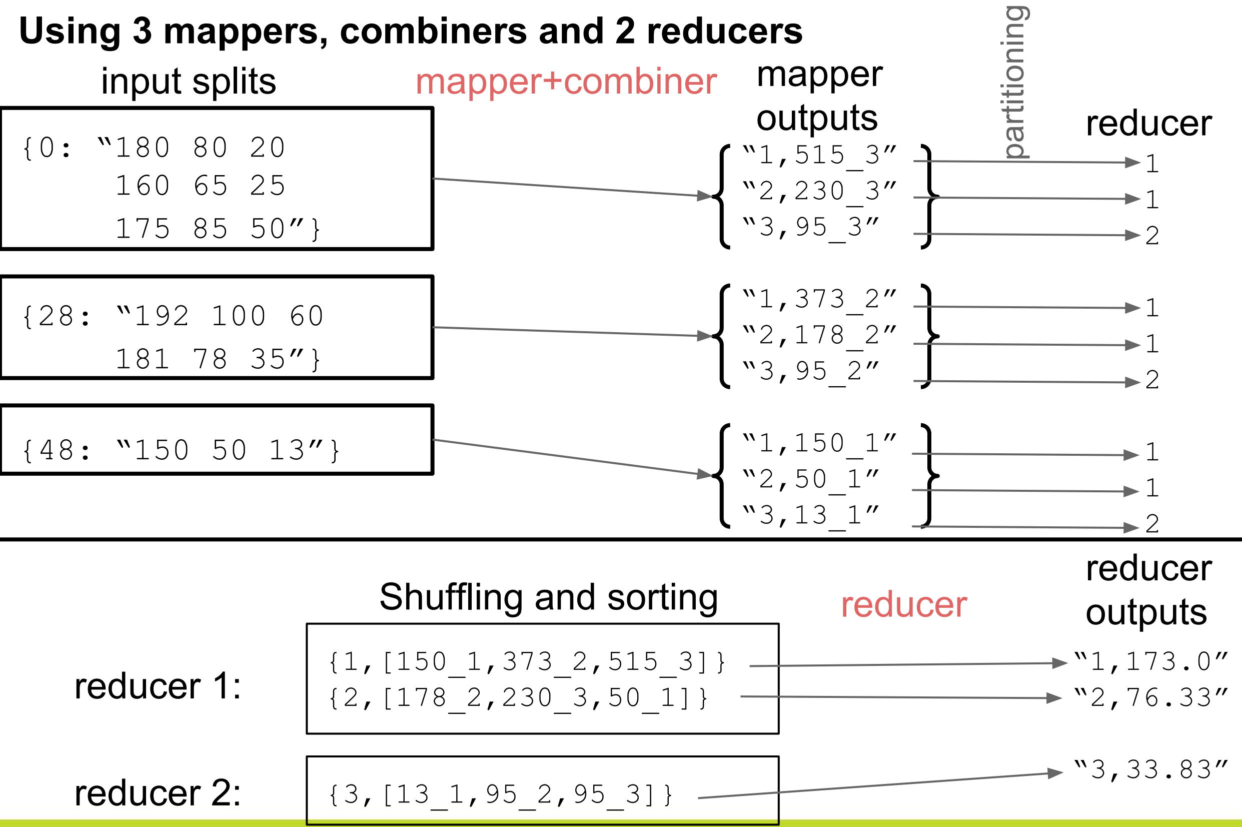
reducer 2:

```
{3,[13_1,95_2,95_3]}
```

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

MapReduce - Example: Calculate a mean (9)

Using 3 mappers, combiners and 2 reducers



input splits

mapper+combiner

mapper outputs

partitioning

reducer

```
{0: "180 80 20
     160 65 25
     175 85 50"}
```

```
"1,515_3"  → 1
"2,230_3"  → 1
"3,95_3"   → 2
```

```
{28: "192 100 60
      181 78 35"}
```

```
"1,373_2"  → 1
"2,178_2"  → 1
"3,95_2"   → 2
```

```
{48: "150 50 13"}
```

```
"1,150_1"  → 1
"2,50_1"   → 1
"3,13_1"   → 2
```

Shuffling and sorting

reducer

reducer outputs

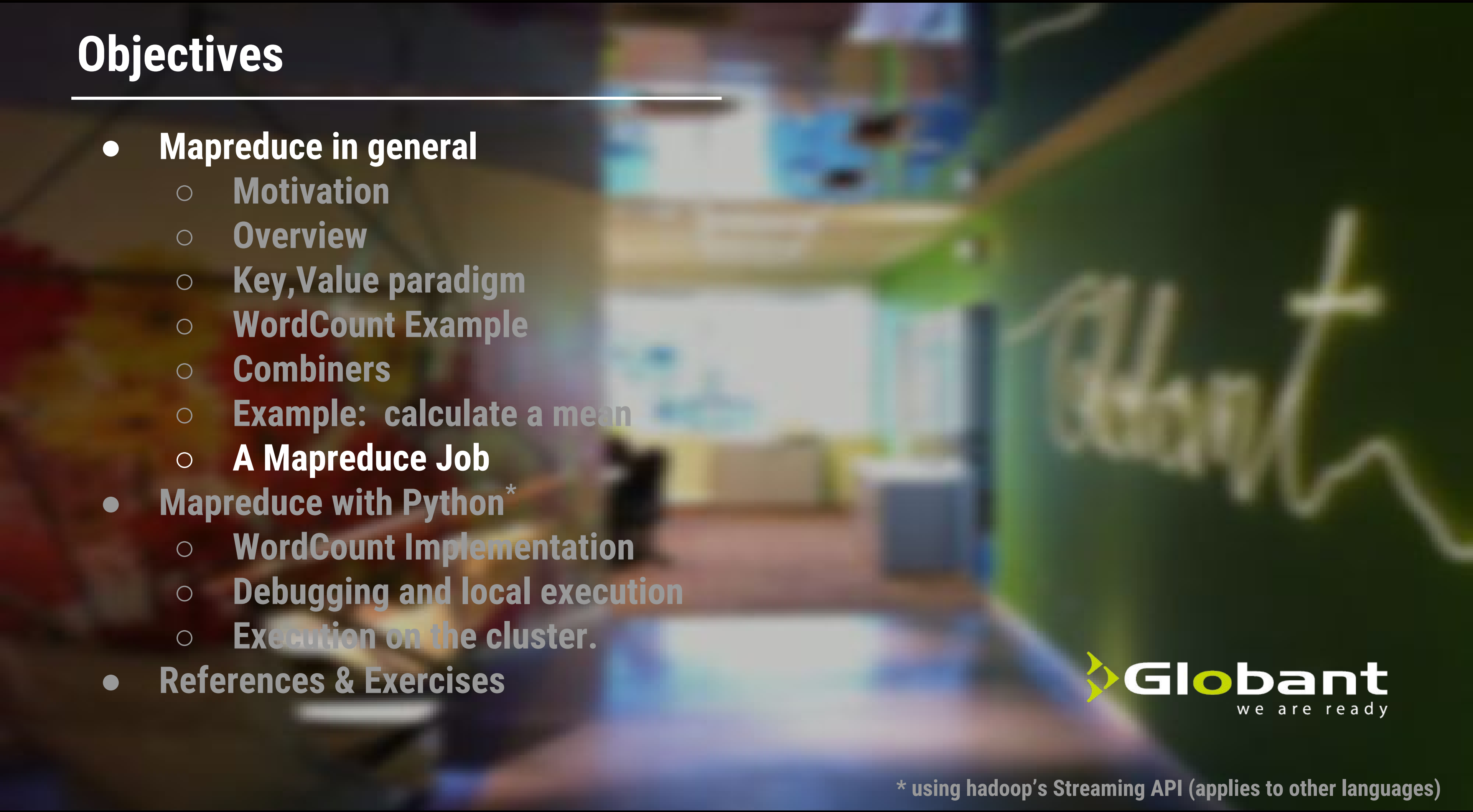
reducer 1:

```
{1,[150_1,373_2,515_3]}  → "1,173.0"
{2,[178_2,230_3,50_1]}   → "2,76.33"
```

reducer 2:

```
{3,[13_1,95_2,95_3]}     → "3,33.83"
```

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$



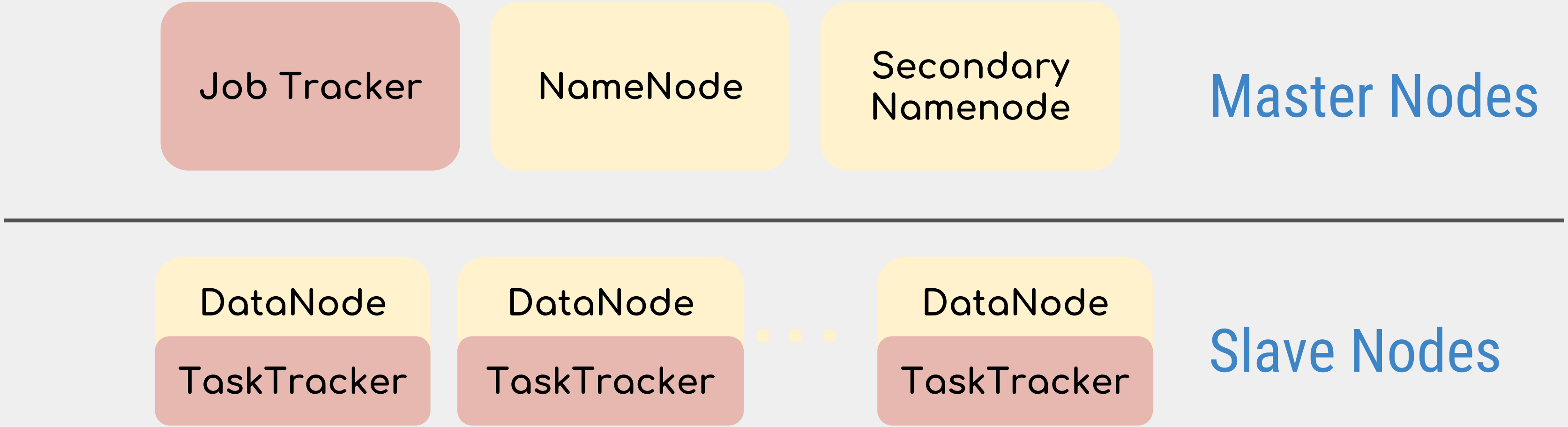
Objectives

- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

Globant
we are ready

* using hadoop's Streaming API (applies to other languages)

MapReduce - Architecture of a Cluster



Job Tracker

NameNode

Secondary Namenode

Master Nodes

DataNode

TaskTracker

DataNode

TaskTracker

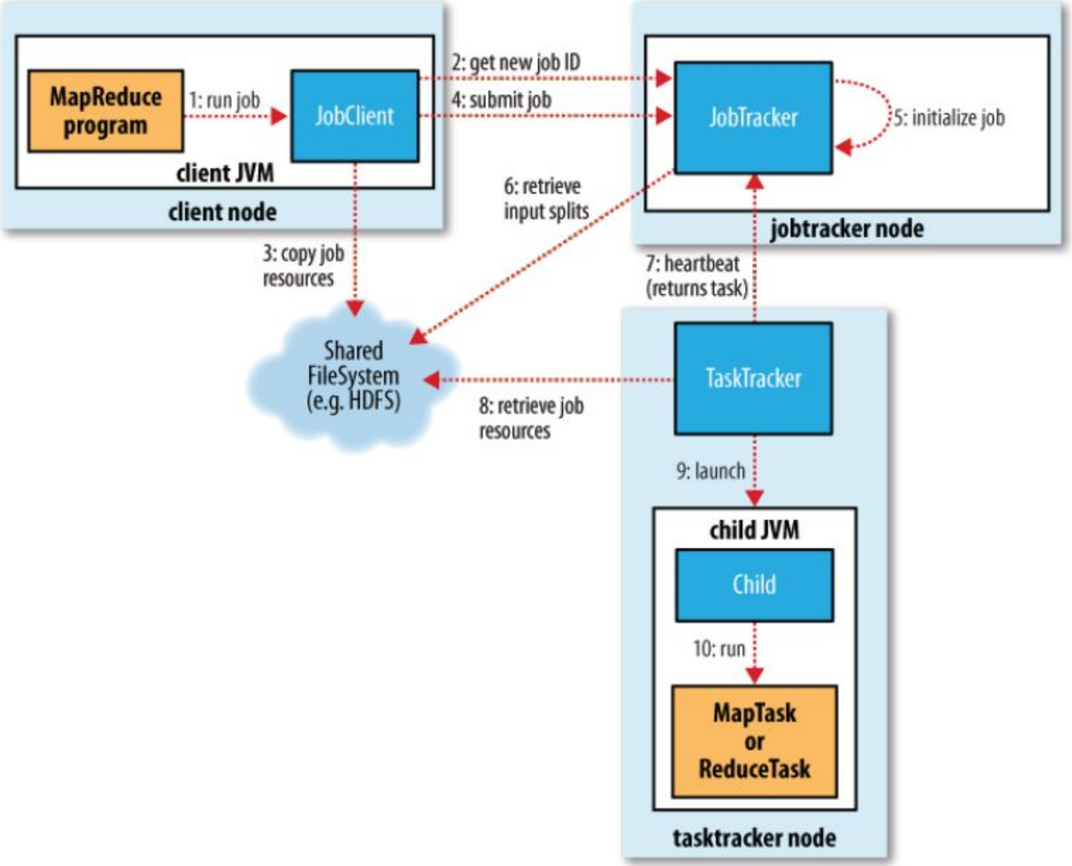
. . .

DataNode

TaskTracker

Slave Nodes

MapReduce - How is the code executed?

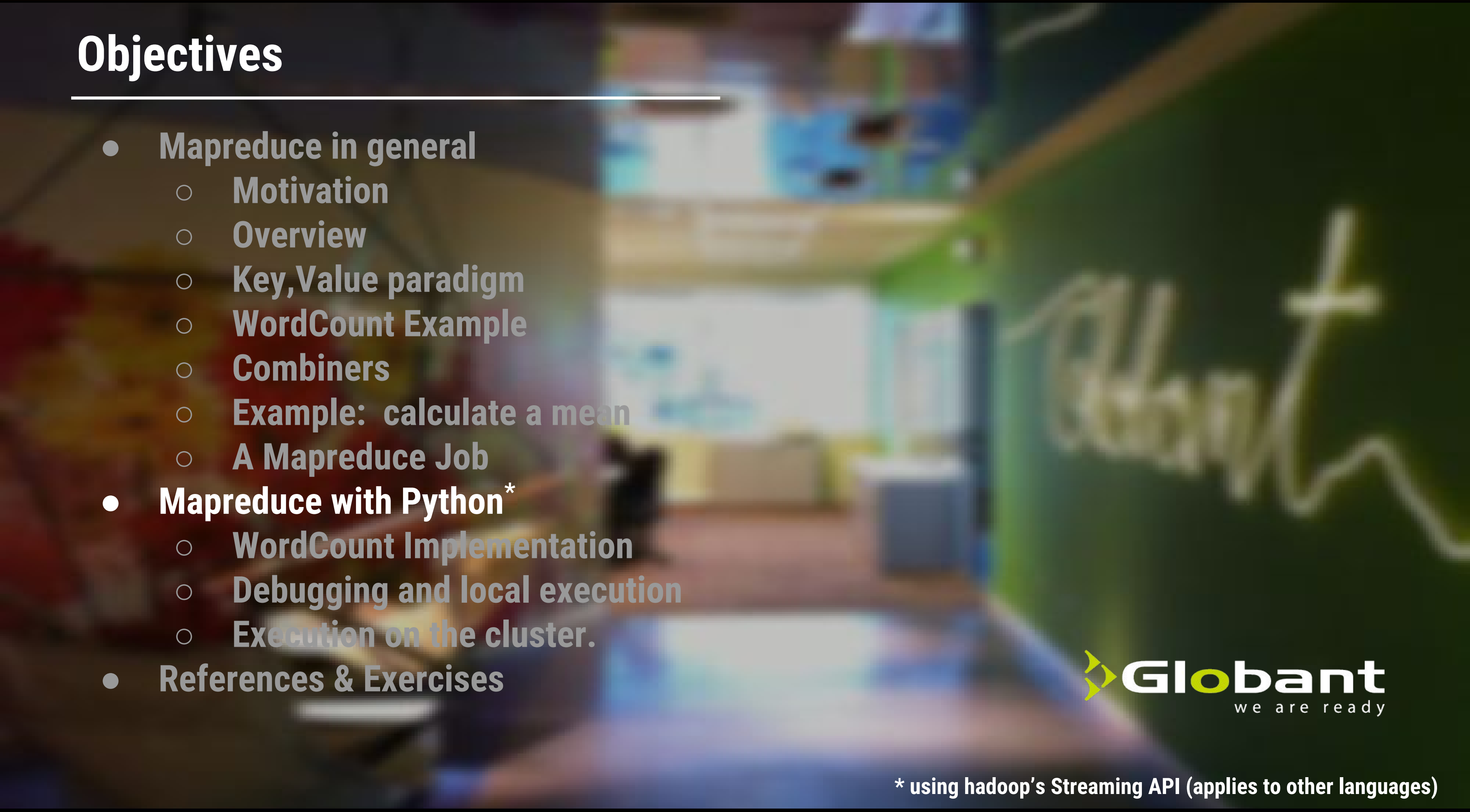


from Hadoop: The Definitive Guide Book

MapReduce - APIs

- Streaming API (today)
  - works practically any programming language (python, bash, perl, c++, etc.)
  - ideal for prototyping !
  - mapper script called one time per split.
  - reducer input are (key,value) pair sorted by key not (key,list-of-values)
  - reducer script called one time for all full pairs of (key, value)

- Java API (next session)
  - requires to know Java, hadoop packages, etc.
  - a bit better performance and more control
  - a bit lengthier to debug.
  - mapper method invoked one time per record.
  - reducer method invoked one time per each pair of (key, list of values)



Objectives

- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- **Mapreduce with Python***
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

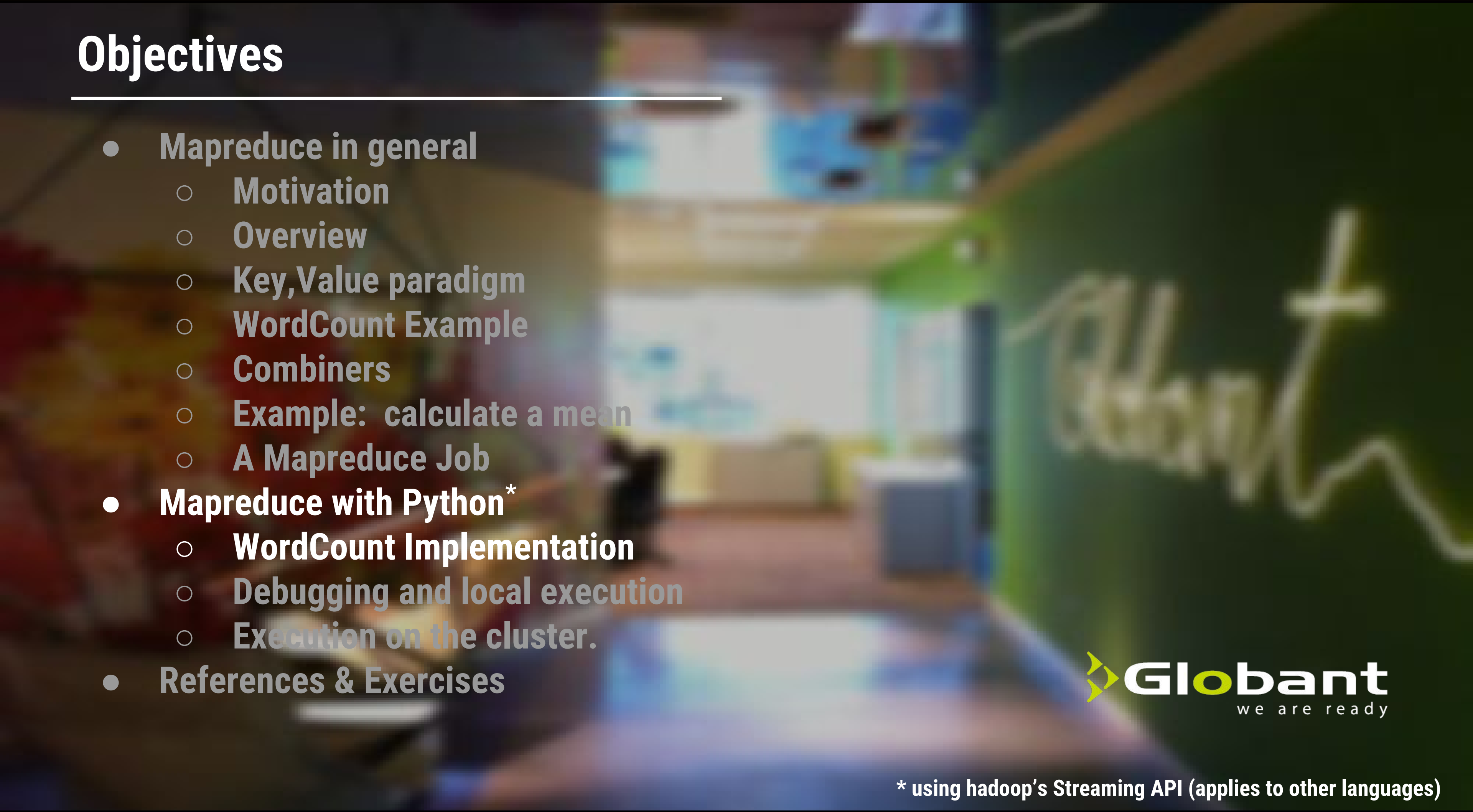
Globant
we are ready

* using hadoop’s Streaming API (applies to other languages)

MapReduce - Streaming API (python, php, c++, etc)

- This API allows practically any programming language (we'll focus on python)
- (Key,Value) pairs fed as a continuous stream of text with sorted keys.
- Data flows into mapper/reducer via *NIX pipes behind the scenes.
- the Developer must:
  - read data from STDIN (standard input)
  - define key,value separator
  - split and parse key and value strings to the correct data type
  - identify when keys change in the stream
  - emit by writing to STDOUT (standard output)

Objectives



- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

* using hadoop's Streaming API (applies to other languages)

MapReduce - WordCount pseudocode

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)





reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

MapReduce - wordcount_mapper.py

```python
#!/usr/bin/env python
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 a list of 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, tab-delimited.
        # The trivial word count is 1
        print '%s\t%s' % (word, 1)
```

MapReduce - wordcount_reducer.py

```
#!/usr/bin/env python
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 wordcount_mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    count = int(count)
    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)
```

Objectives

- Mapreduce in general
    - Motivation
    - Overview
    - Key,Value paradigm
    - WordCount Example
    - Combiners
    - Example: calculate a mean
    - A Mapreduce Job
- **Mapreduce with Python***
    - WordCount Implementation
    - **Debugging and local execution**
    - Execution on the cluster.
- References & Exercises

Globant
we are ready

* using hadoop’s Streaming API (applies to other languages)

MapReduce - debugging and local execution (1)

Testing Execution without Hadoop!! (useful to debug)

Run from shell:

```
cat <input txt file(s)> | python ./wordcount_mapper.py | sort -k1,1 | python ./wordcount_reducer.py
```

The output is shown directly on the screen:

```
"       28
"'A     1
"'About 1
"'Absolute       1
"'Ah!'  2
"'Ah,   2
"'Ample.'       1
"'And   10
"'Are   1
"'Arthur!'      1
"'As    1
...
```

⚠ This only works if the code is totally independent of the hadoop environment, e.g., no use of environment variables, etc.

MapReduce - debugging and local execution (2)

Execution with Hadoop using the VM (pseudo cluster)

After copying the code and the sample data to the VM, execute from shell:

```
hadoop \
jar <path-to-hadoop-parent-dir>/hadoop-2.6.0/share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar \
-D mapred.map.tasks=5 \      ←optional
-D mapred.reduce.tasks=2 \  ←optional
-files <code-dir>/wordcount_mapper.py,<code-dir>/wordcount_reducer.py  \
-mapper "python ./wordcount_mapper.py" \
-reducer "python ./wordcount_reducer.py" \
-input <hdfs-input-dir> \
-output <hdfs-output-dir>
```

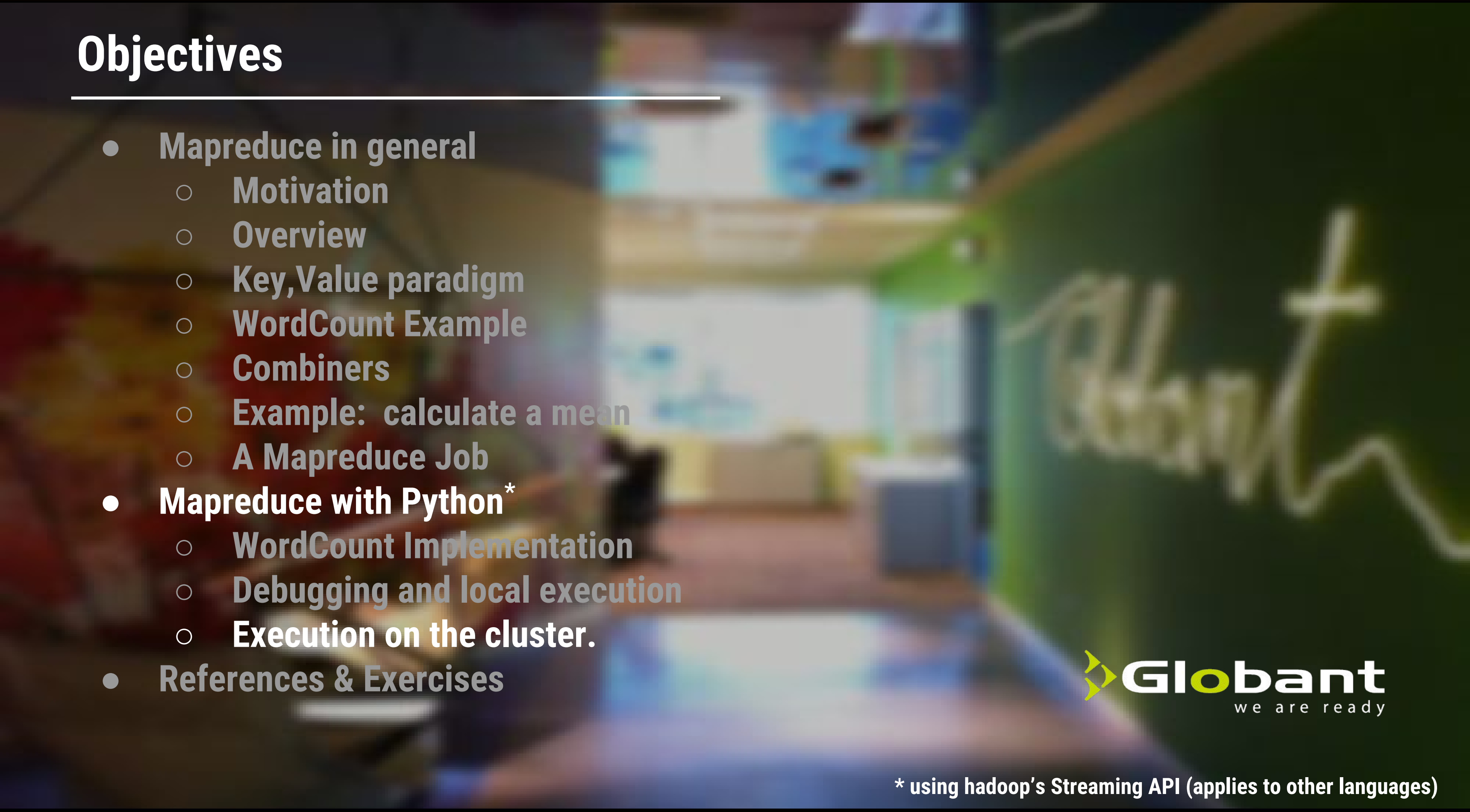
The output is written to the HDFS files:

```
<hdfs-output-dir>/part-00000
<hdfs-output-dir>/part-00001
```

To see a part of one of the files:

```
hadoop fs -tail <hdfs-output-dir>/part-00000
```

Objectives



- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- References & Exercises

* using hadoop's Streaming API (applies to other languages)

MapReduce - cluster execution (1)

Execution on the AWS cluster

After copying the python code to the main server, execute from shell:

```
hadoop \
jar /home/hadoop/hadoop/hadoop-2.5.2/share/hadoop/tools/lib/hadoop-streaming-2.5.2.jar \
-D mapred.map.tasks=5 \      ←optional
-D mapred.reduce.tasks=2 \  ←optional
-files <code-dir>/wordcount_mapper.py,<code-dir>/wordcount_reducer.py  \
-mapper "python ./wordcount_mapper.py" \
-reducer "python ./wordcount_reducer.py" \
-input /user/hadoop/mapreduce/data/books \
-output <hdfs-output-dir>
```

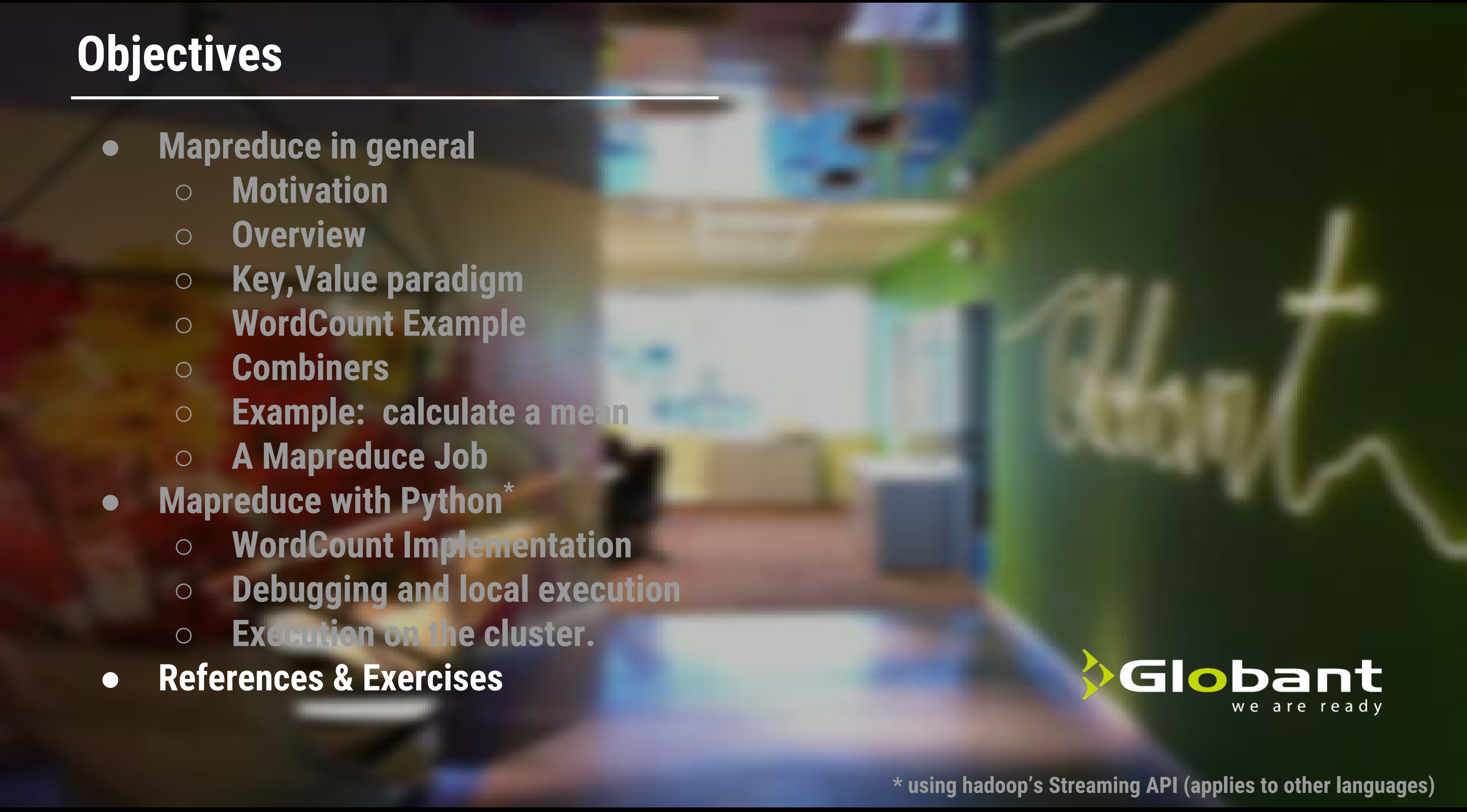
The output is written to the HDFS files:

```
<hdfs-output-dir>/part-00000
<hdfs-output-dir>/part-00001
```

To see a part of one of the files:

```
hadoop fs -tail <hdfs-output-dir>/part-00000
```

Objectives



- Mapreduce in general
  - Motivation
  - Overview
  - Key,Value paradigm
  - WordCount Example
  - Combiners
  - Example: calculate a mean
  - A Mapreduce Job
- Mapreduce with Python*
  - WordCount Implementation
  - Debugging and local execution
  - Execution on the cluster.
- **References & Exercises**

* using hadoop’s Streaming API (applies to other languages)

Mapreduce - References

- Yahoo Mapreduce tutorial

https://developer.yahoo.com/hadoop/tutorial/module4.html

- White, Tom. Hadoop: The definitive guide (a.k.a. “The Elephant Book”)

- Apache documentation for Streaming API

http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html

Mapreduce - Exercises

- Avoid peeking at the solutions on the web !
- Use WordCount code as starting point.
- Don’t code and debug right off using the full input dataset. Subset o create a simpler dataset first (better).
- Focus on the mapreduces first. If any other “plumbing code” is needed, do it after that and use a manual solution in the meantime.
- Don’t share coded solutions, instead share hints on how to proceed and give your peers the opportunity to learn for themselves.

Mapreduce - Exercise 1

Inverted Index

Given a directory with books in txt format, write a mapreduce which outputs an inverted index, i.e., a table that associates a word with the books and the corresponding positions at which it occurs (http://en.wikipedia.org/wiki/Inverted_index).

Dataset URL: here

HDFS Cluster DataSet path: `/user/hadoop/mapreduce/data/books`

hint 1: Suggested output example (not real data):

```
Love     alice_in_wonderland.txt:100,the_prince.txt:900,the_prince.txt:1050
```

hint 2: Given the mapper doesn't receive the filename as input. A Hadoop Configured Parameter (environment variable) could help to retrieve the filename from which the word comes.

Mapreduce - Exercise 2

Column-wise Variance of a matrix

Given a csv file without headers, calculate the sample variance (s^2) of each column. (http://en.wikipedia.org/wiki/Variance)

$$s^2 = \frac{1}{(N-1)} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

HDFS DataSet path: `/user/hadoop/mapreduce/data/matrix`

¡¡¡DON'T DOWNLOAD IT FROM THE CLUSTER $$$$!!!

hint 1: Suggested output: columnIndex<tab>sampleVariance.

Example:

```
0    135.6
1    2.2
2    536.9
...
```

hint 2: Assume the file has only numeric values and no entries are missing (no NULLs or empty).

hint 3: The python modules `os` and `sys` will come quite handy.

hint 4: Focus on the the mapreduce. (If any) other plumbing code is required, do it manually, afterwards implement it if you have the time (with python of course!).

Mapreduce - Exercise 2

Column-wise Variance of a matrix

question 1:
Assuming your matrix is very big (both rows and columns), how many reducers does it make sense to have?

question 2:
How many mapreduce jobs do you need?

question 3:
Implement combiners and notice the savings in bytes transferred to the reducers.

Thank You!

Globant
we are ready