Lecture 3: The MapReduce Programming Model

Instructor: Michela Taufer

THE UNIVERSITY OF TENNESSEE KNOXVILLE

BIG ORANGE. BIG IDEAS.®

Today Outline

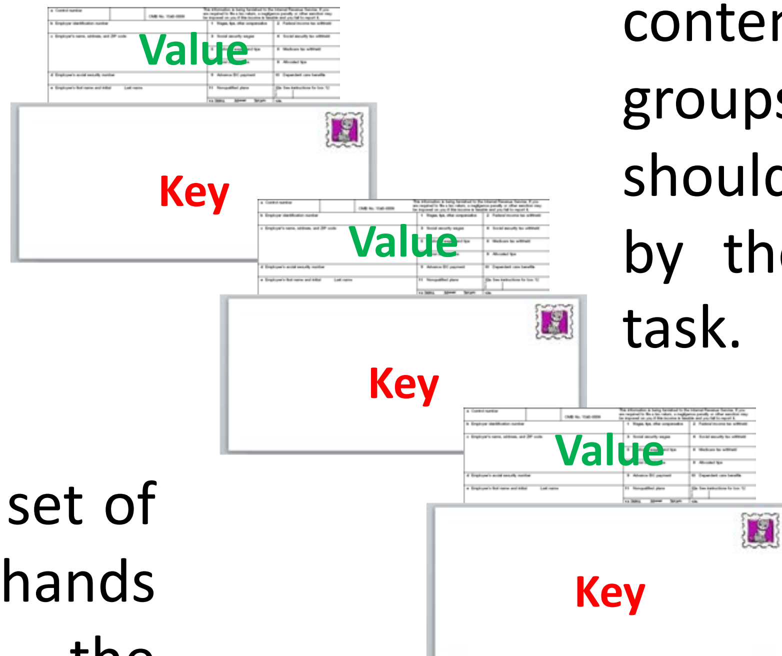
- Learn a new programming models: MapReduce
  - MapReduce programming model: principles and definitions
  - Word Count: a concrete example of MapReduce
  - MapReduce algorithms: different approaches to solve the same problem
  - Data: Where is the data kept?
- Continue building our expertise with Jupyter and Python
  - Sequential manipulation of a classical in literature
  - Visualization of statistics

The Canonical MapReduce Programing Model

The MapReduce Workflow

Step 1: Map

The **map** task creates a set of Key-Value pairs and hands them off to the sorting/shuffling task.

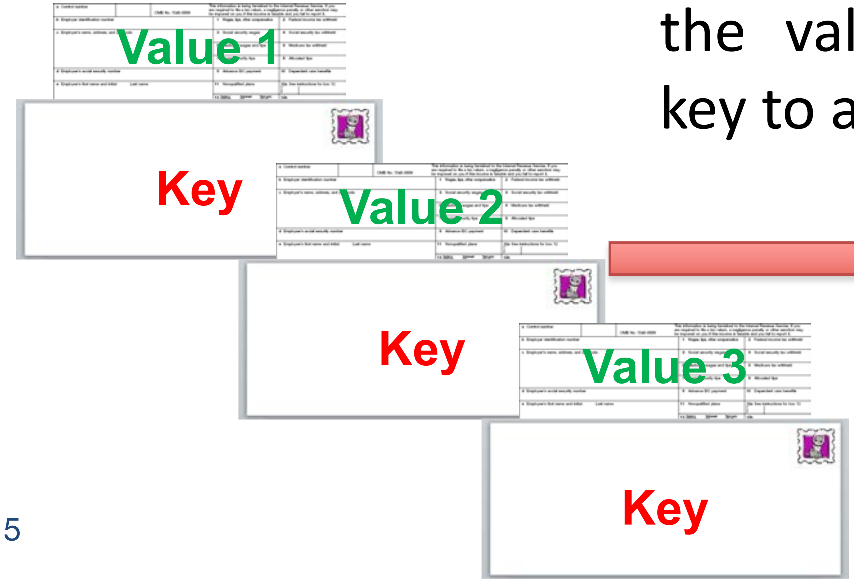


The Value is the content, the Key groups content that should be processed by the same **reduce** task.

The MapReduce Workflow

Step 2: Sort/Shuffle

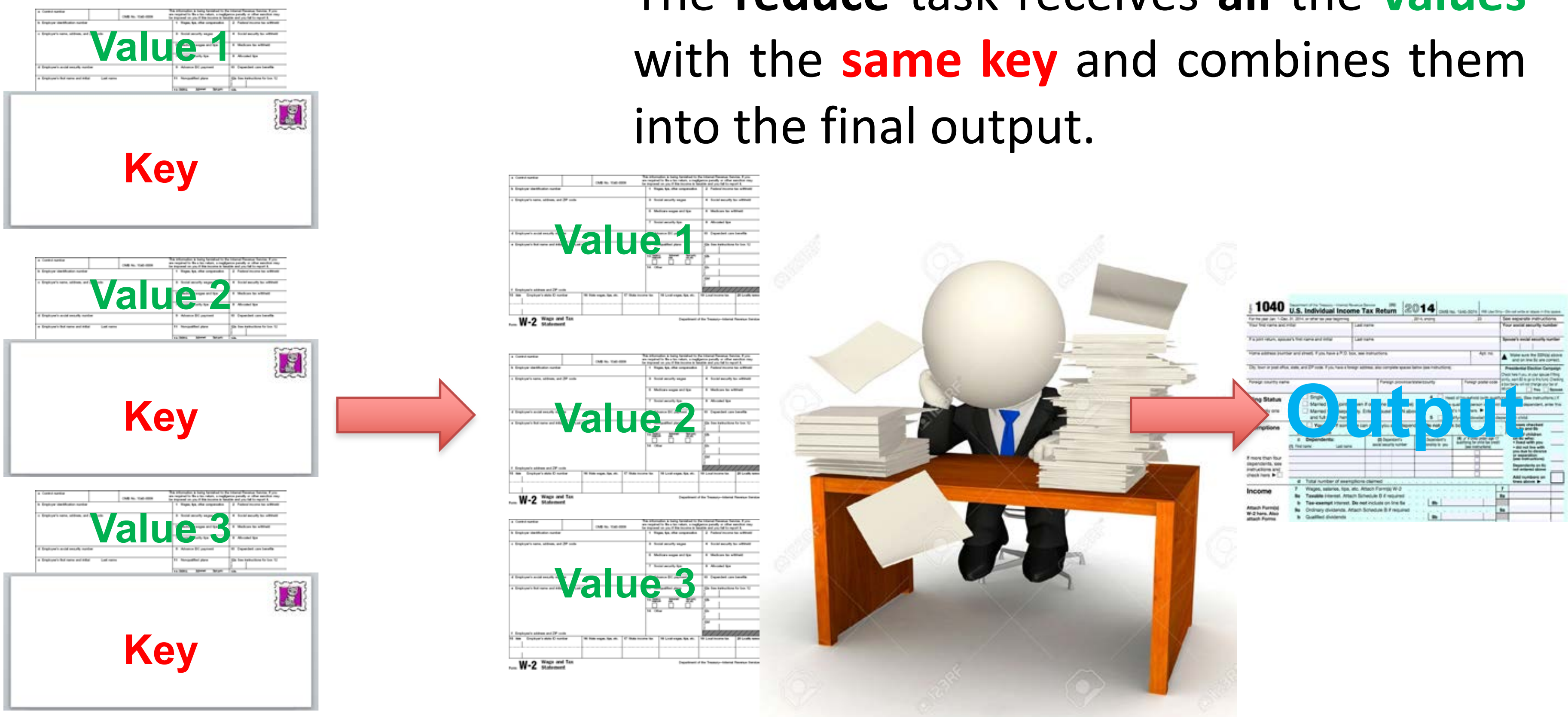
The **sorting** and **shuffling** tasks collect all the **key-value** pairs and deliver all the values with the same key to a **reduce** task.



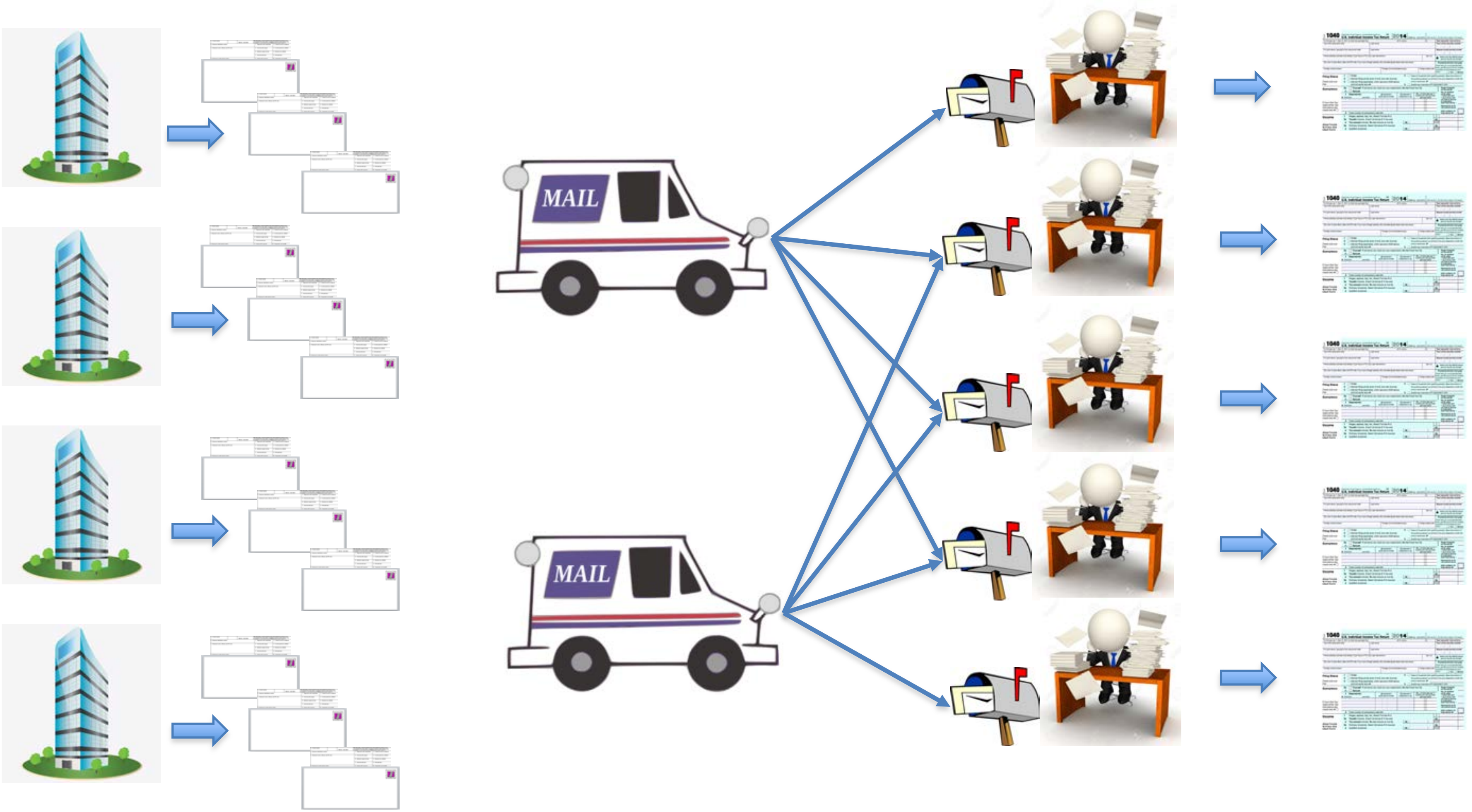
The MapReduce Workflow

Step 3: Reduce

The **reduce** task receives **all** the **values** with the **same key** and combines them into the final output.



The MapReduce Workflow



Map **Shuffle** **Reduce**

Map

- The map function is implemented by the user.
- Each map task processes a single line of an input file at a time.
- Map tasks do not communicate with other map tasks.
- Map tasks communicate with reduce tasks **only** through the content of the value in the KV pair.
- A single map task may emit **any number** of KV pairs (including none).

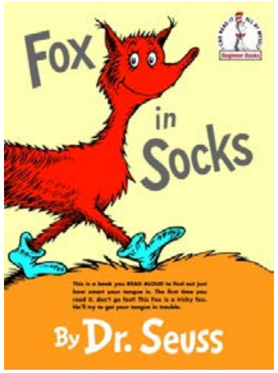
Sort/Shuffle

- The sorting and shuffling process is implemented at the framework level and is opaque to the user.

Reduce

- The reduce function is implemented by the user.
- Each reduce task receives **all** the **values** associated with a given **key**.
- Reduce tasks do not communicate with each other
- A single reduce task may emit **any number** of result values for each **key** it processes.

WordCount: an example of MapReduce



Map

Key — Value

When tweetle beetles fight, → (When, 1), **(tweetle, 1)**, (beetles, 1), (fight, 1)

it's called a tweetle beetle battle. → (it's, 1), (called, 1), (a, 1), (tweetle, 1), (beetle, 1), (battle, 1)

And when they battle in a puddle, → (And, 1), (when, 1), (they, 1), (battle, 1), (in, 1), (a, 1), (puddle, 1)

it's a tweetle beetle puddle battle. → (it's, 1), (a, 1), (tweetle, 1), (beetle, 1), (puddle, 1), (battle, 1)

And when tweetle beetles battle with paddles in a puddle, → (And, 1), (when, 1), (tweetle, 1), (beetles, 1), (battle, 1), (with, 1), (paddles, 1), (in, 1), (a, 1), (puddle, 1)

They call it a tweetle beetle puddle paddle battle. → (They, 1), (call, 1), (it, 1), (a, 1), (tweetle, 1), (beetle, 1), (puddle, 1), (paddle, 1), (battle, 1)

WordCount: an example of MapReduce

Sort/Shuffle

(When, 1), **(tweetle, 1),** (beetles, 1), (fight, 1)

(it's, 1), (called, 1), (a, 1), (**tweetle**, 1), (beetle, 1), (battle, 1)

(And, 1), (when, 1), (they, 1), (battle, 1), (in, 1), (a, 1), (puddle, 1)

(it's, 1), (a, 1), (**tweetle**, 1), (beetle, 1), (puddle, 1), (battle, 1)

(And, 1), (when, 1), (**tweetle**, 1), (beetles, 1), (battle, 1),
(with, 1), (paddles, 1), (in, 1), (a, 1), (puddle, 1)
(They, 1), (call, 1), (it, 1), (a, 1), (**tweetle**, 1), (beetle, 1),
(puddle, 1), (paddle, 1), (battle, 1)

Reduce

(**tweetle**, 1), (**tweetle**, 1), (**tweetle**, 1), (**tweetle**, 1), (**tweetle**, 1)	→	(**tweetle**, 5)
(When, 1)	→	(When, 1)
(when, 1), (when, 1)	→	(when, 2)
(battle, 1), (battle, 1), (battle, 1), (battle, 1), (battle, 1)	→	(battle, 5)
(puddle, 1), (puddle, 1), (puddle, 1), (puddle, 1)	→	(puddle, 4)
(beetle, 1), (beetle, 1), (beetle, 1)	→	(beetle, 3)
(beetles, 1), (beetles, 1)	→	(beetles, 2)

⋮

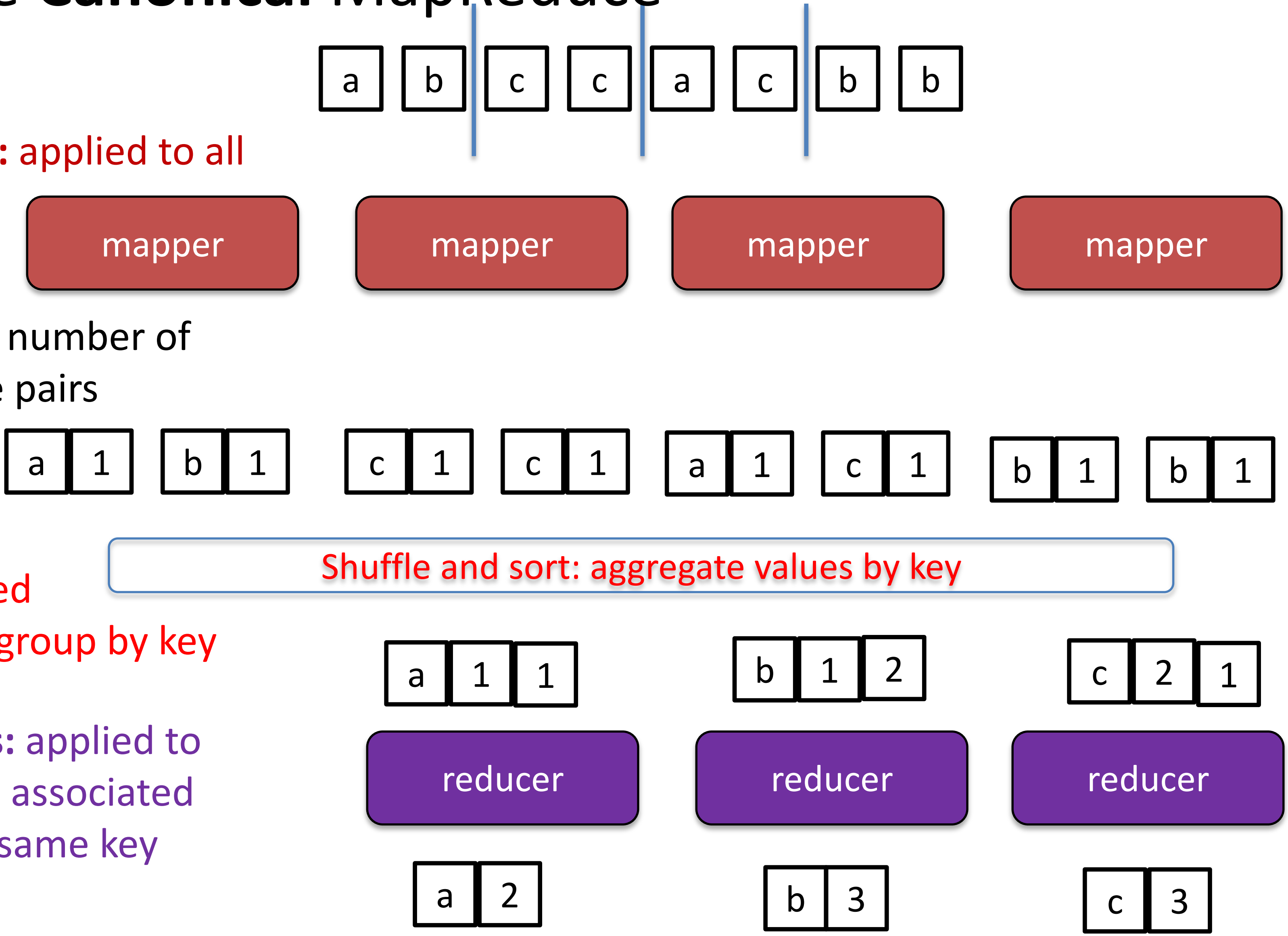
The **Canonical** MapReduce

Mappers: applied to all input data

Arbitrary number of key-value pairs

Barrier: distributed sort and group by key

Reducers: applied to all values associated with the same key



Pseudo-code: WordCount in MapReduce

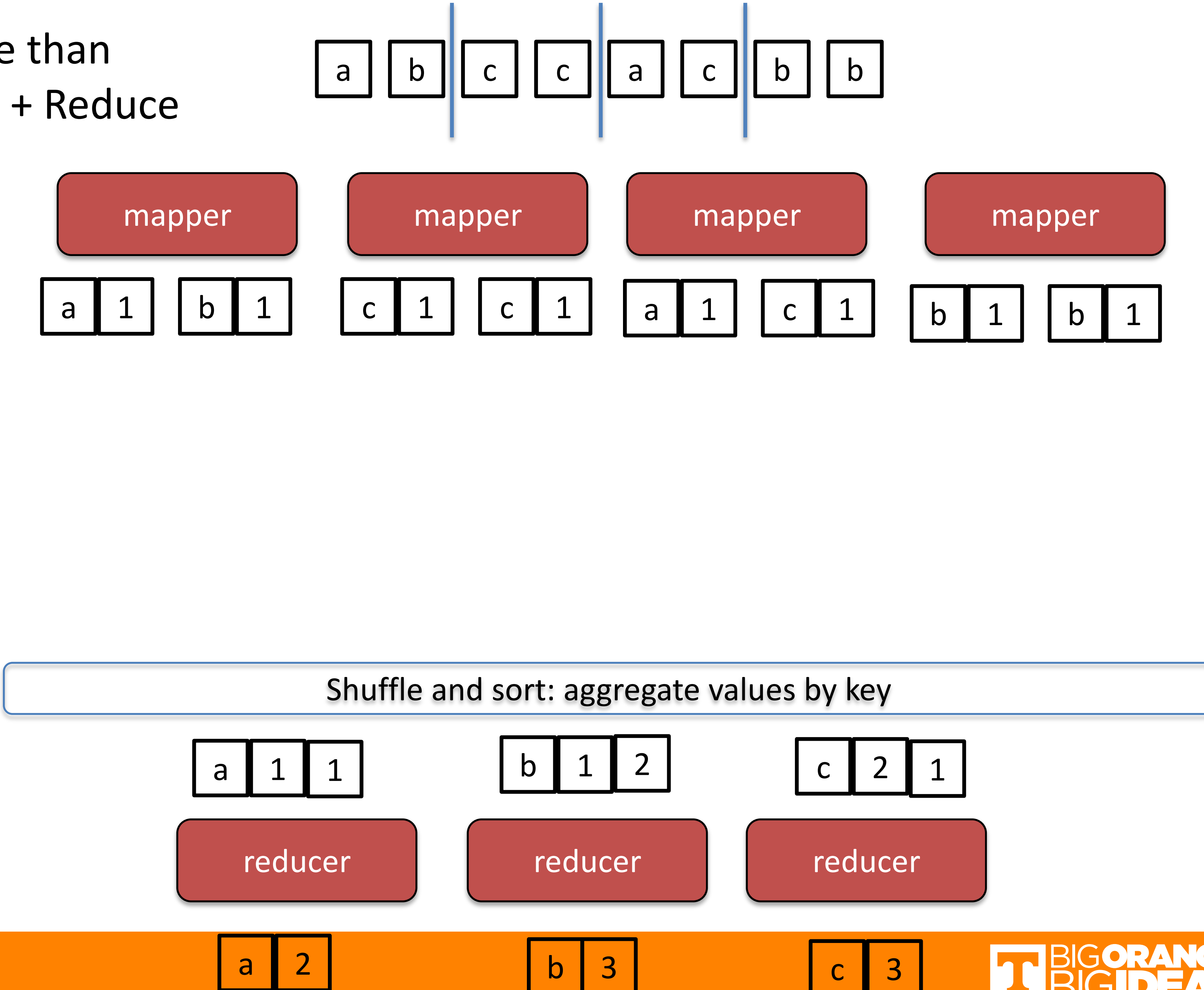
```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)

class Reducer
    method Reduce(term t, counts [c1, c2, . . .])
        sum ← 0
        for all count c ∈ counts [c1, c2, . . .] do
            sum ← sum + c
        Emit(term t, count sum)
```

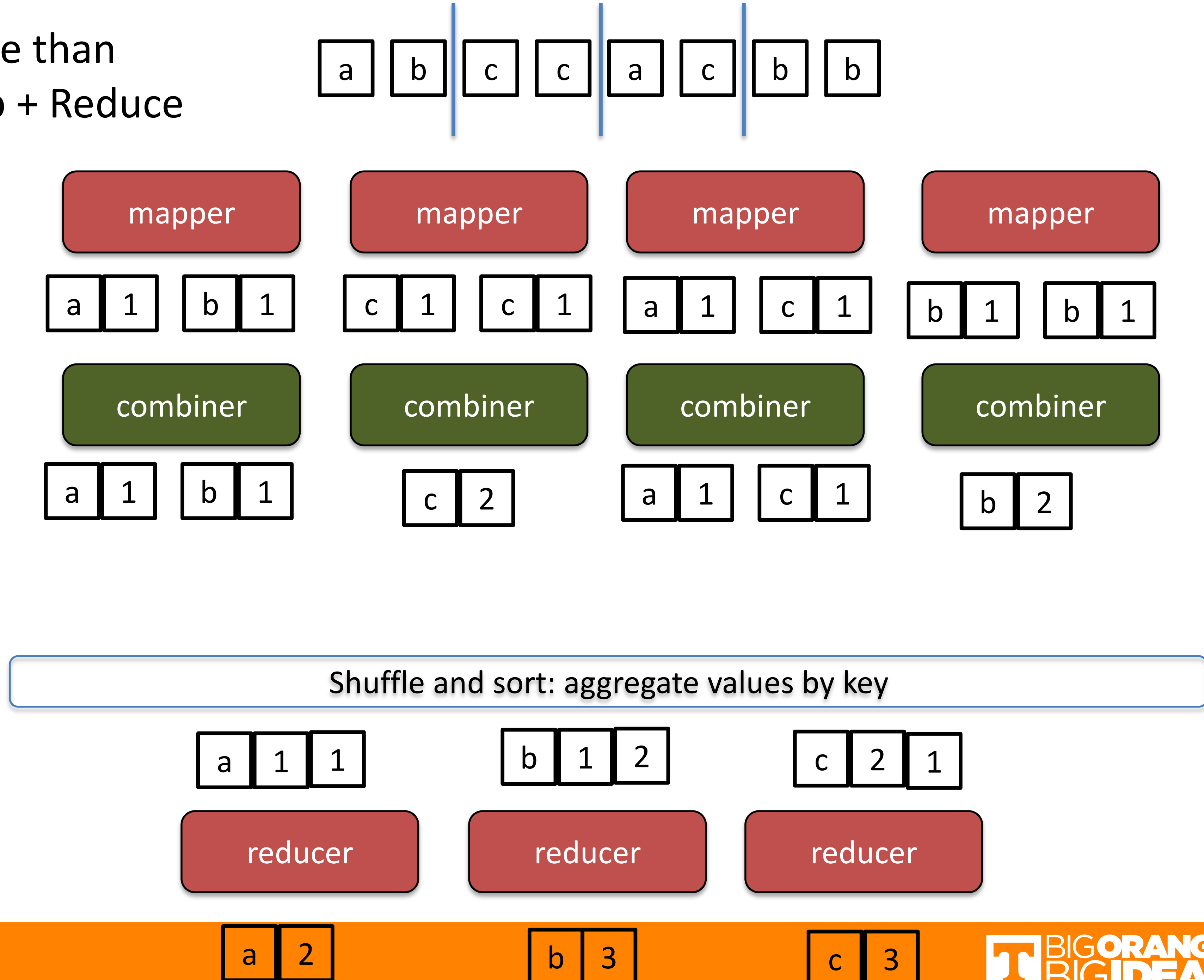
More than “Map + Reduce”

- Canonical MapReduce processing workflow:
  - Map + Reduce
- Variations:
  - Map + Combiners + Partitioners + Reduce (in Apache Hadoop and Mimir implementations)
  - Automatic implementation of combiners on the map side (in Apache Spark - more in the next lecture)

More than Map + Reduce



More than Map + Reduce

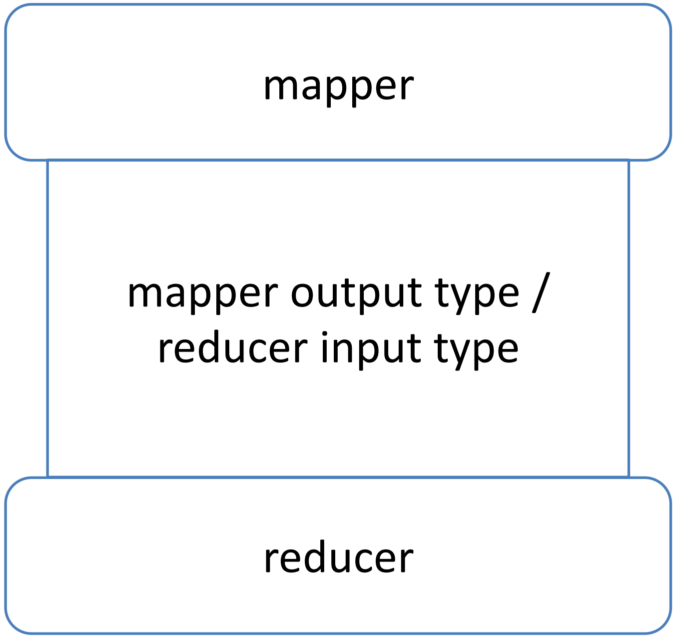


More than M + R: Combiners

- Combiners allow for local aggregation before the shuffle and sort phase
  - “mini-reducers” that take place locally, on the output of the mappers
- Strengths:
  - Number of intermediate key-value pairs moved among reducers to be **at most the number of unique words** in the collection times the number of mappers
- Weaknesses:
  - Reducers and combiners are not interchangeable unless the operation is both associative and commutative
  - Operation performed in isolation and therefore does not have access to intermediate output from other mappers

More than M + R: Combiners Constraints

- Combiners in, for example, Apache Hadoop cannot change the correctness of the MapReduce algorithm
- Combiners must have same input and output key-value types



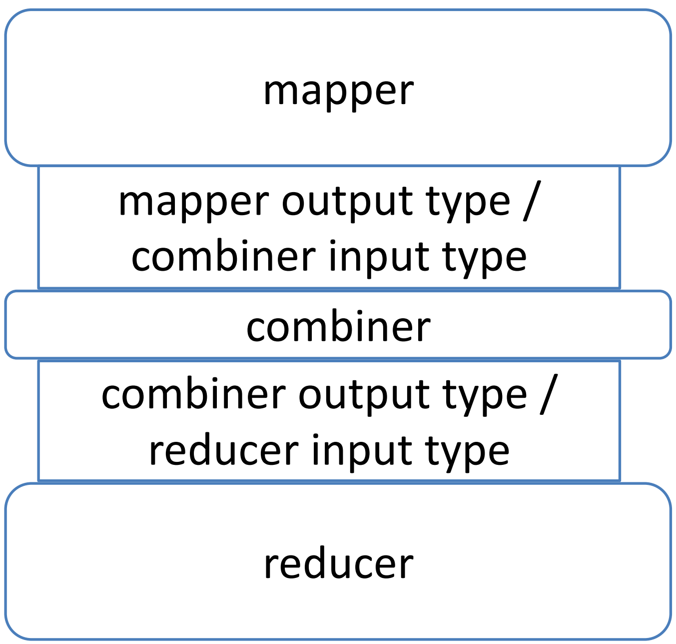
mapper output type

==

reducer input type

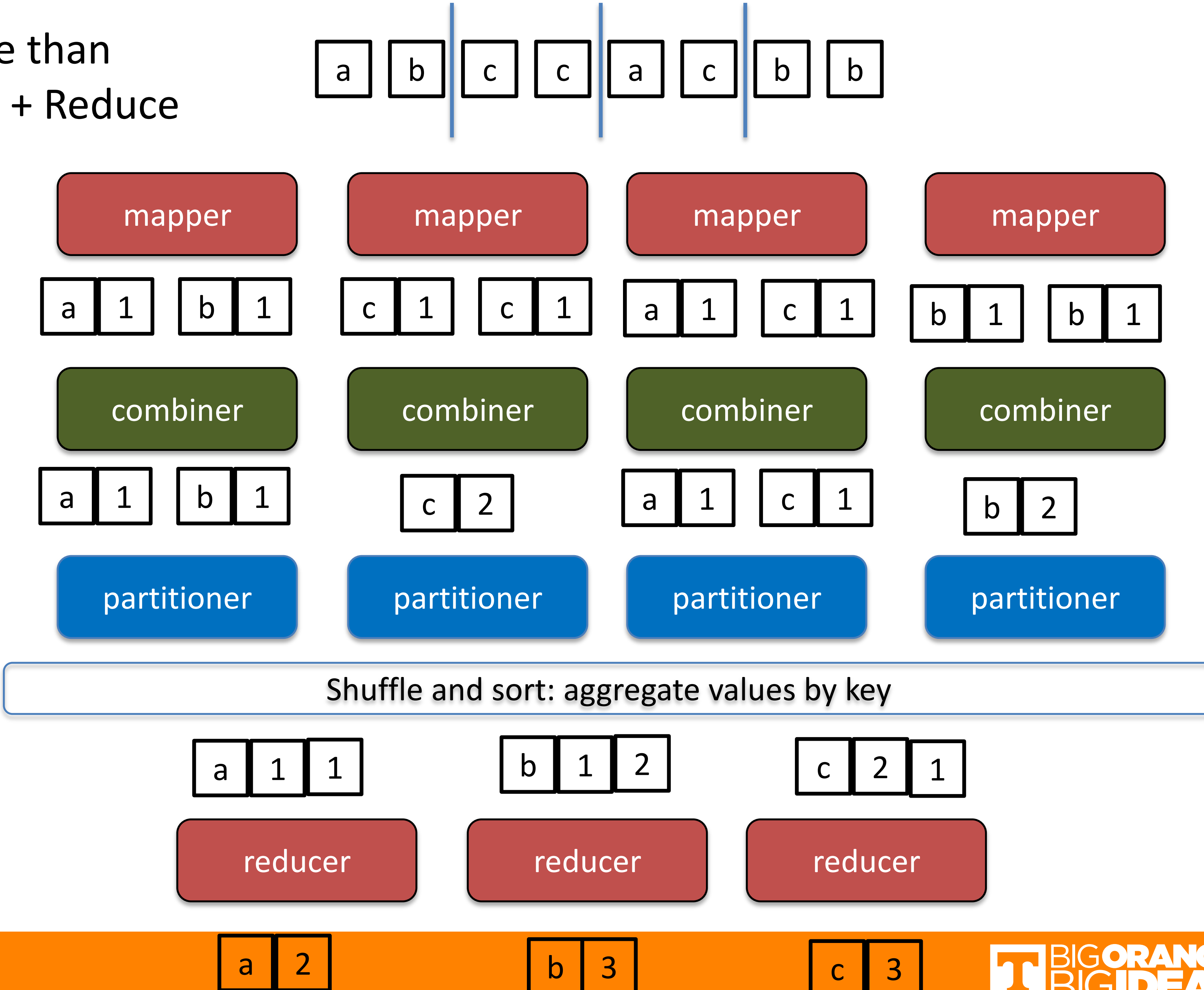
More than M + R: Combiners Constraints

- Combiners in, for example, Apache Hadoop cannot change the correctness of the MapReduce algorithm
- Combiners must have same input and output key-value types



mapper output type
==
combiner input type
==
combiner output type
==
reducer input type

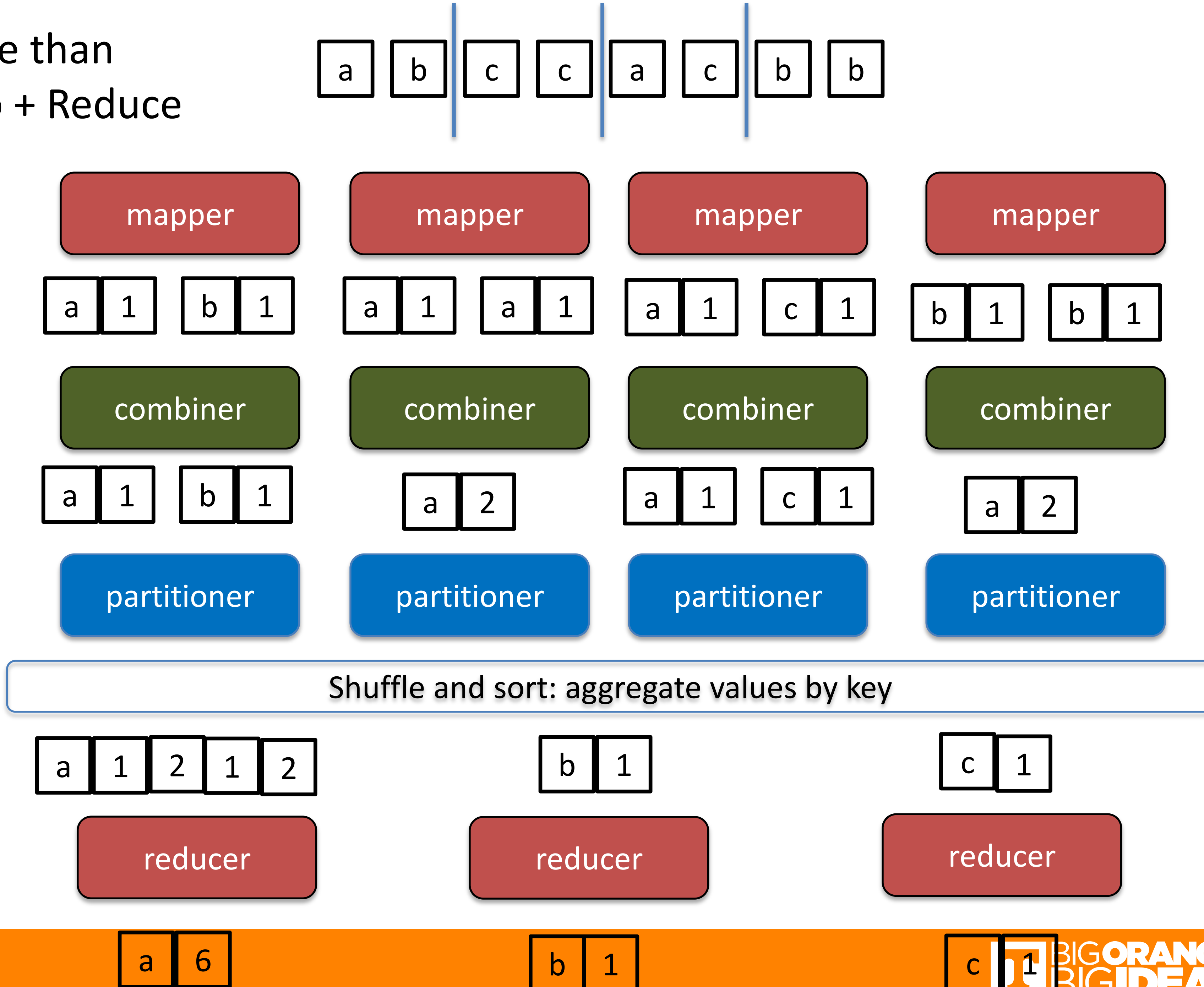
More than Map + Reduce



More than M + R: Partitioners

- Divide up the intermediate key space and **assign intermediate key-value pairs to reducers**
- Example of simple partitioner:
  - Compute hash value of a key
  - Take the mod of the value with the number of reducers
- Strengths:
  - Assign approximately the same number of keys to each reducer (dependent on the quality of the hash function)
- Weaknesses:
  - Ignore the value and different keys may have different numbers of associated values causing imbalance in the amount of data associated with each key

More than Map + Reduce



WordCount in MapReduce (Review)

class MAPPER
  method MAP(docid a, doc d)
    for all term $t \in$ doc d **do**
      EMIT(term t, count 1)

class REDUCER
  method REDUCE(term t, counts $[c_1, c_2, \ldots]$)
    $sum \leftarrow 0$
    for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
      $sum \leftarrow sum + c$
    EMIT(term t, count sum)

Using Associative Array

Associative array to aggregate term counts on a per-document basis

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)
```

```
class Mapper
    method Map(docid a, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(term t, count H{t})
```

In-mapper Combining

```
class Mapper
    method Map(docid a, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
        for all term t ∈ H do
            Emit(term t, count H{t})
```

```
class Mapper
    method Initialize
        H ← new AssociativeArray
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            H{t} ← H{t} + 1
    method Close
        for all term t ∈ H do
            Emit(term t, count H{t})
```

- Initialize an associative array for holding term counts
- Accumulate partial counts in associative array across multiple documents
- incorporate combiner functionality directly inside the mapper (in-mapper combining)

One problems,
many solutions

A real problem

- Given a very large log report from a soccer website with the *number of scores per game* of all the soccer players worldwide for seasons 2013 – 2018
  - Keys represent soccer player name
  - Values represent the number of scores a player get per game
  - Data is chronologically sorted based on the date of the game
- **Which players shall be awarded the Soccer's Best Player Award?**
  - Criteria: the winner has the highest mean number of scores per played game
- Compute the mean number of scores on a per-player basis

Things to remember …

Mean(1; 2; 3; 4; 5) **IS NOT** Mean(Mean(1; 2); Mean(3; 4; 5))

A solution

class MAPPER
  method MAP(string t, integer r)
    EMIT(string t, integer r)

class REDUCER
  method REDUCE(string t, integers $[r_1, r_2, \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
      $sum \leftarrow sum + r$
      $cnt \leftarrow cnt + 1$
    $r_{avg} \leftarrow sum/cnt$
    EMIT(string t, integer r_{avg})

A solution

Q: Does the solution work?

class MAPPER
  method MAP(string t, integer r)
    EMIT(string t, integer r)

class REDUCER
  method REDUCE(string t, integers $[r_1, r_2, \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
      $sum \leftarrow sum + r$
      $cnt \leftarrow cnt + 1$
    $r_{avg} \leftarrow sum/cnt$
    EMIT(string t, integer r_{avg})

A solution

Q: Is this an efficient solution?

class MAPPER
    method MAP(string t, integer r)
        EMIT(string t, integer r)

class REDUCER
    method REDUCE(string t, integers $[r_1, r_2, \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
            $sum \leftarrow sum + r$
            $cnt \leftarrow cnt + 1$
        $r_{avg} \leftarrow sum/cnt$
        EMIT(string t, integer r_{avg})

Another solution?

```
class Mapper
    method Map(string t, integer r)
        Emit(string t, integer r)
```

```
class Combiner
    method Combine(string t, integers [r_1, r_2, ...])
        sum ← 0
        cnt ← 0
        for all integer r ∈ integers [r_1, r_2, ...] do
            sum ← sum + r
            cnt ← cnt + 1
        Emit(string t, pair (sum, cnt))
```

```
class Reducer
    method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
```

Another portable solution?

class MAPPER
  method MAP(string t, integer r)
    EMIT(string t, integer r)

class COMBINER
  method COMBINE(string t, integers $[r_1, r_2, \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
      $sum \leftarrow sum + r$
      $cnt \leftarrow cnt + 1$
    EMIT(string t, pair (sum, cnt))

class REDUCER
  method REDUCE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    $r_{avg} \leftarrow sum/cnt$
    EMIT(string t, integer r_{avg})

Another portable solution?

class MAPPER
    method MAP(string t, integer r)
        EMIT(string t, integer r)

class COMBINER
    method COMBINE(string t, integers $[r_1, r_2, \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
            $sum \leftarrow sum + r$
            $cnt \leftarrow cnt + 1$
        EMIT(string t, pair (sum, cnt))

class REDUCER
    method REDUCE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
            $sum \leftarrow sum + s$
            $cnt \leftarrow cnt + c$
        $r_{avg} \leftarrow sum/cnt$
        EMIT(string t, integer r_{avg})

Another portable solution?

class Mapper
  method Map(string t, integer r)
    Emit$\boxed{(\text{string } t, \text{integer } r)}$

class Combiner
  method Combine$\boxed{(\text{string } t, \text{integers } [r_1, r_2, \ldots])}$
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
      $sum \leftarrow sum + r$
      $cnt \leftarrow cnt + 1$
    Emit$\boxed{(\text{string } t, \text{pair } (sum, cnt))}$

class Reducer
  method Reduce$\boxed{(\text{string } t, \text{pairs } [(s_1, c_1), (s_2, c_2) \ldots])}$

Mismatch between combiner input key-value type and output key-value type violates the MapReduce programming model!!!

A portable and efficient solution

class MAPPER
  method MAP(string t, integer r)
    EMIT(string t, pair $(r, 1)$)

class COMBINER
  method COMBINE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    EMIT(string t, pair (sum, cnt))

class REDUCER
  method REDUCE(string t, pairs $[(s_1, c_1), (s_2, c_2) \ldots]$)
    $sum \leftarrow 0$
    $cnt \leftarrow 0$
    for all pair $(s, c) \in$ pairs $[(s_1, c_1), (s_2, c_2) \ldots]$ **do**
      $sum \leftarrow sum + s$
      $cnt \leftarrow cnt + c$
    $r_{avg} \leftarrow sum/cnt$
    EMIT(string t, integer r_{avg})

In-mapper Combining Design Pattern Solution

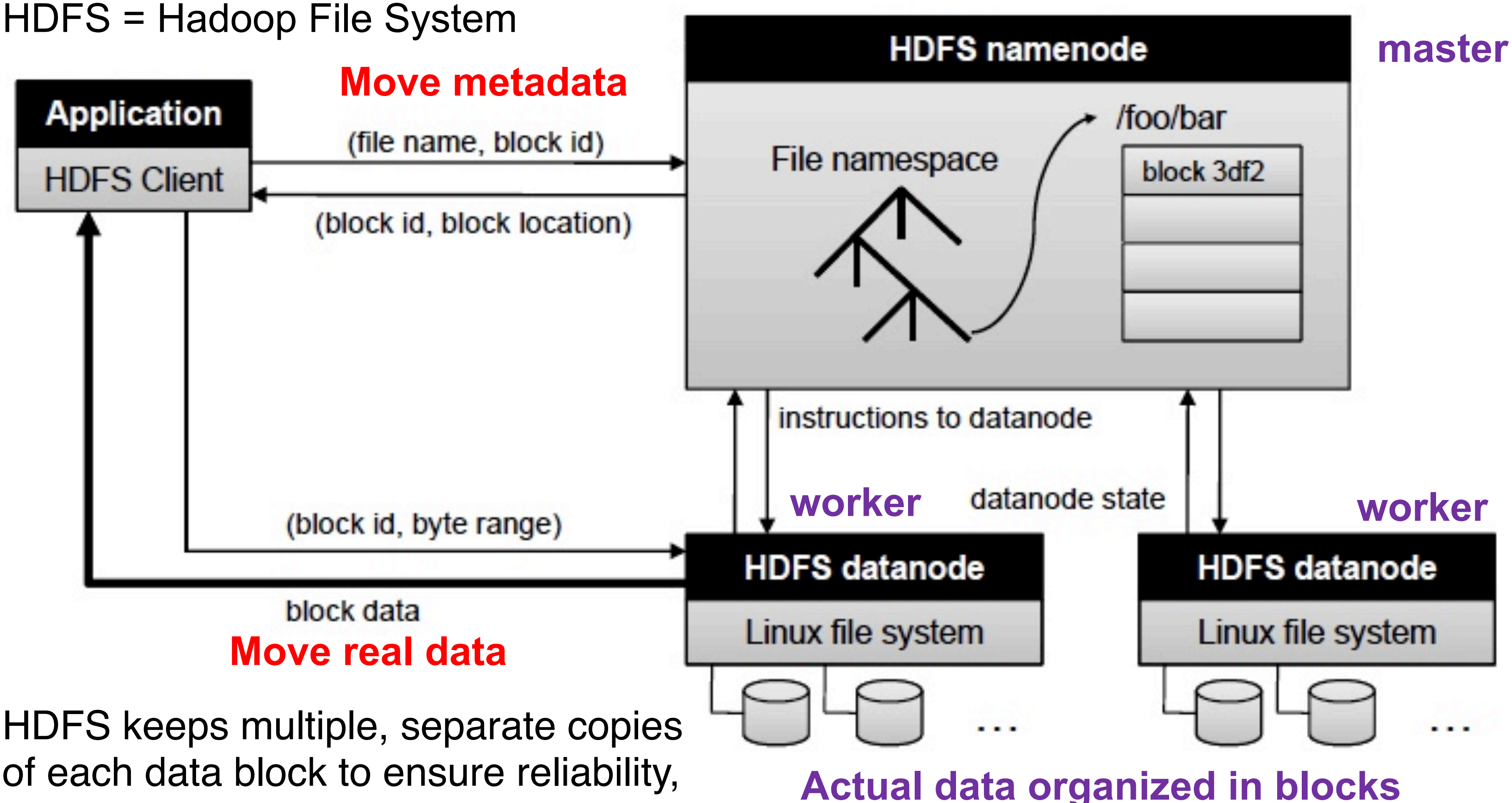
```
class Mapper
    method Initialize
        S ← new AssociativeArray
        C ← new AssociativeArray
    method Map(string t, integer r)
        S{t} ← S{t} + r
        C{t} ← C{t} + 1
    method Close
        for all term t ∈ S do
            Emit(term t, pair (S{t}, C{t}))
```

```
class Reducer
    method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) . . .])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) . . .] do
            sum ← sum + s
            cnt ← cnt + c
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
```

Where is the data kept?

The Hadoop File System

HDFS = Hadoop File System



HDFS keeps multiple, separate copies of each data block to ensure reliability, availability, and performance

Assignment 3

Assignment 3 - CS 594 / CS 690

- Goal: Continue building our expertise with Jupyter and Python
  - Sequential manipulation of a classical in literature
  - Visualization of statistics
- Deadline: ***September 24, 8AM ET***

Assignment 3 - CS 594 / CS 690

- Given a literature classic such as the “The Count of Monte Cristo”
- Problem 1: Analyze the text for word length frequency
- Problem 2: Analyze the text for letter frequency
- Problem 3: Count the positional frequencies of each letter (first, interior, and last)
- Problem 4: Visualize your findings in histograms (one for each one of Problems 1-3)
  - One code is give to you, write the other two codes

Deadline: ***September 24 - 8AM ET***

Assignment 3 - CS 690

- Read paper "MapReduce: Simplied Data Processing on Large Clusters" *by* Jeffrey Dean and Sanjay Ghemawat, Google Inc.
- Submit summary:
  - Add summary to your private GitHub repository
  - Use the template provided
  - Follow mandatory requirements for your summary

Deadline: ***October 1 - 8AM ET***

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE
BIG ORANGE. BIG IDEAS.®

BIG ORANGE
BIG IDEAS