Introduction to Data Science

(Lecture 22)

Dr. Mohammad Pourhomayoun

Assistant Professor

Computer Science Department

California State University, Los Angeles

BIG
DATA
ALSO
TOOLS
DISK
TARGET
APPLIED
SENSOR
MAY
EXAMPLES
SHARED
CURRENT
MOVING
DEFINITION
EVERY
RECONSIDER
WITHIN
PRACTITIONERS
CAPTURE
ZETTABYTES
THOUGHT
BUSINESS
INTERNET
DISTRIBUTED
SETS
GENOMICS
COMPLEXITY
DESCRIBING
RADIO-FREQUENCY
MANAGEMENT
TERABYTES
CASE
ABILITY
PETABYTES
INCLUDE
ELAPSED
FC
SYSTEMS
TOLERABLE
FORMS
AMOUNT
STORAGE
SIZE
MPP
PARALLEL
SAN
QUALITIES
SINCE
MASSIVELY
GROW
CONNECTOMICS
ORGANIZATIONS
SET
RELATIONAL
USE
LARGER
CONTINUES
SOCIAL
INDEXING
TENS
COMPLEX
CITATION
ANALYTICS
NOW
BIOLOGICAL
PROCESSING
UBIQUITOUS
TYPES
CAPACITY
BURIED
WORLD'S
SOLID
HUNDREDS
NETWORKS
DATABASES
DESKTOP
RECORDS
CURRENTLY
GARTNER
WORKING
OPPORTUNITIES
SEARCH
DIFFICULTY

Map-Reduce

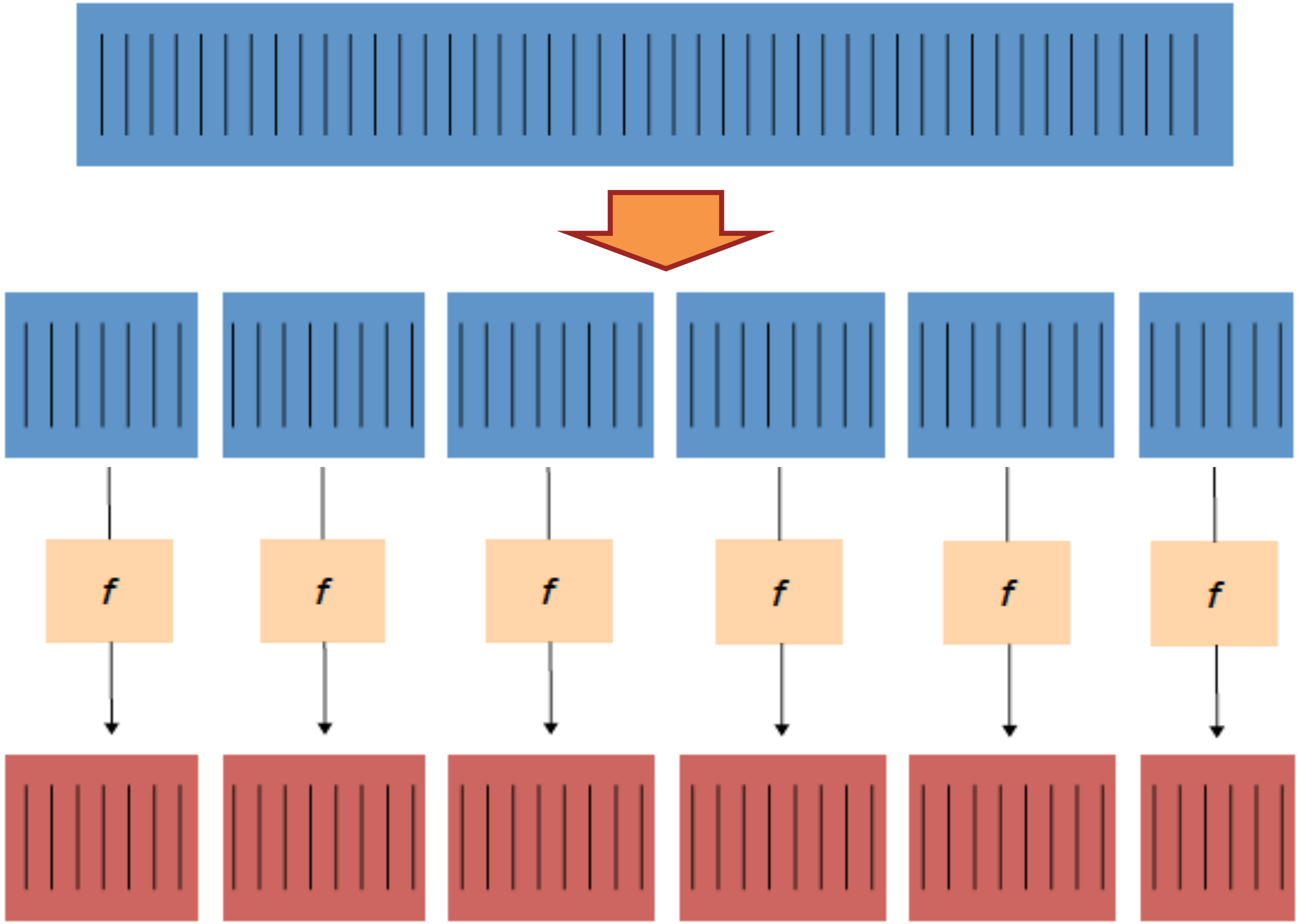
Map-Reduce

- **Map-Reduce** is a programming model for processing big data sets with a parallel and distributed algorithm.

- **map function** processes input key/value pairs to generate a set of intermediate key/value pairs.
- **reduce function** merges all intermediate values associated with the same intermediate key.

[Ref]: Dean, Jeffrey & Ghemawat, Sanjay. (2004). MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM.

Example1: Converting 405k TIFF Images to PNG[1]



You are given TIFF images

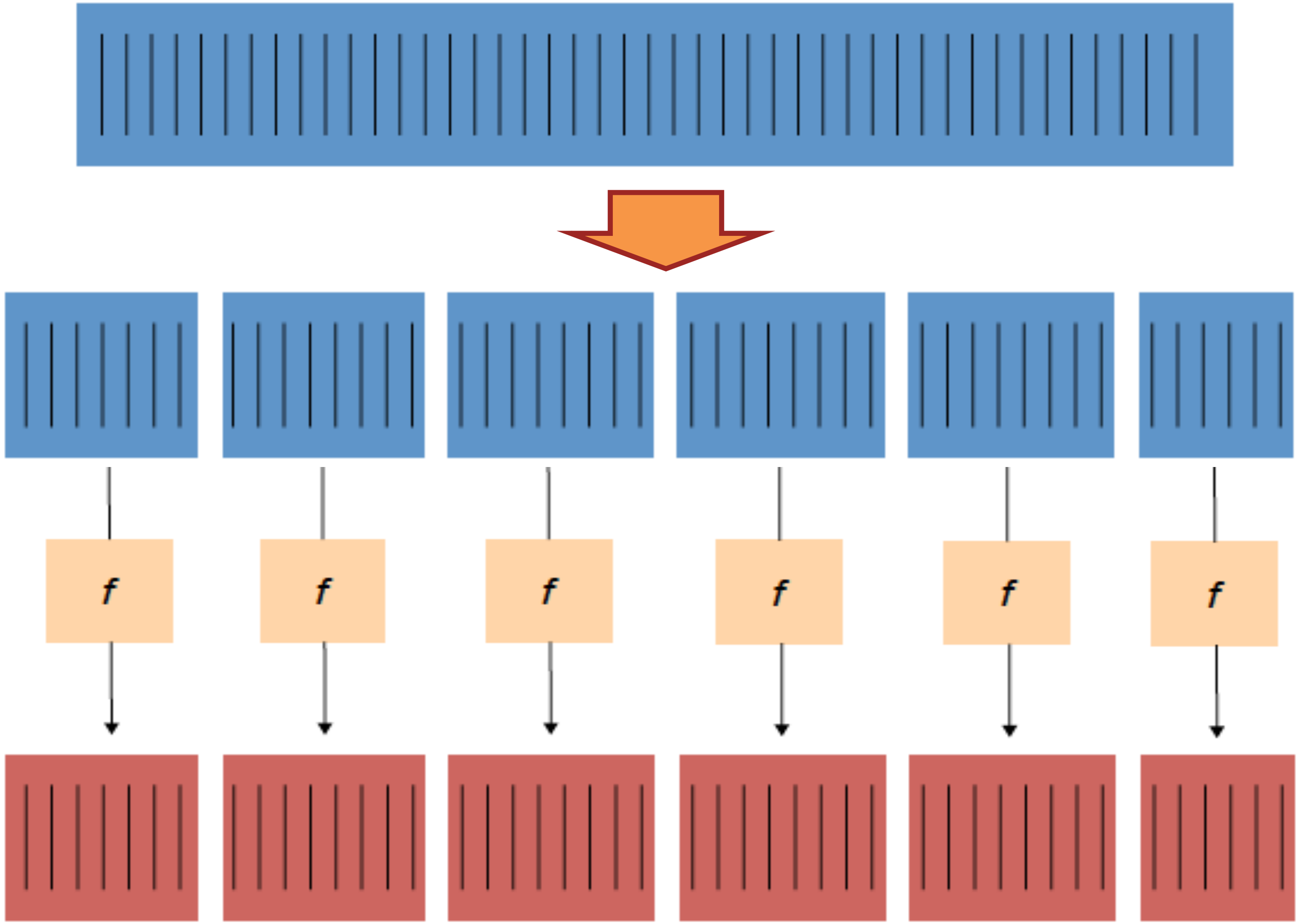
Distribute the images among k computers

f is a function to convert TIFF to PNG; apply it to every item

Now we have a big distributed set of converted images

* Example from blogs.nytimes.com, and Bill Howe, University of Washington

Example2: We have 5M documents. Find the most common word in each document



You have millions of documents

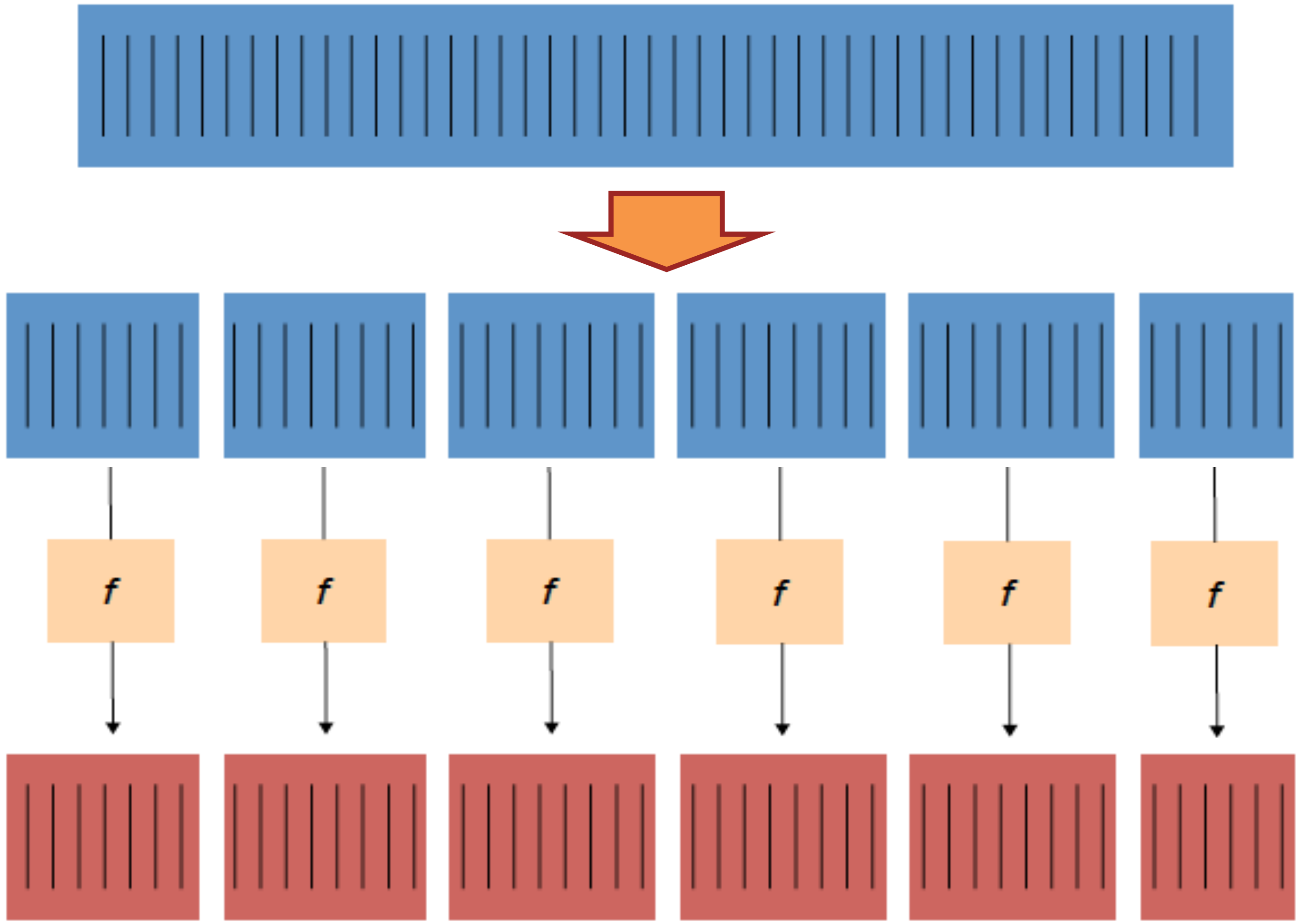
Distribute the documents among k computers

f finds the most common word in each document

Now we have a big distributed list of **(doc_id, word)** pairs

* Example from Bill Howe, University of Washington

Example3: Compute overall word frequency across 5M docs



You have millions of documents

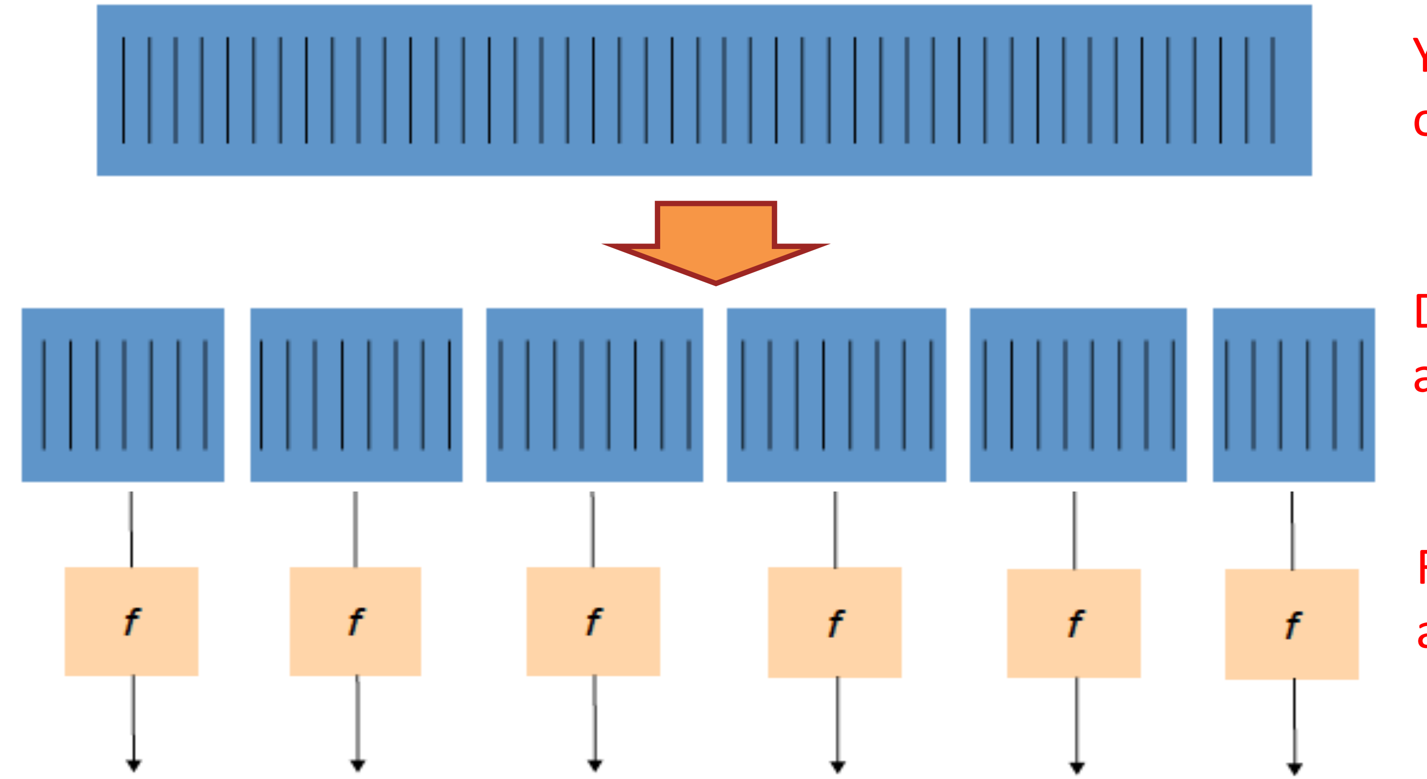
Distribute the documents among k computers

For each document, ***f*** returns a set of (word, freq) pairs

Now we have a big distributed list of **(word,freqs)** pairs for each set of docs

* Example from Bill Howe, University of Washington

Continue Example3 : Compute overall word frequency across 5M docs



f

f

f

f

f

f

You have millions of documents

Distribute the documents among k computers

For each document, f returns a set of **(word, freq)** pairs

[(people, 128)
(book, 323)
(student, 123)
(history, 78)
…]

[(people, 178)
(book, 312)
(student, 123)
(history, 81)
…]

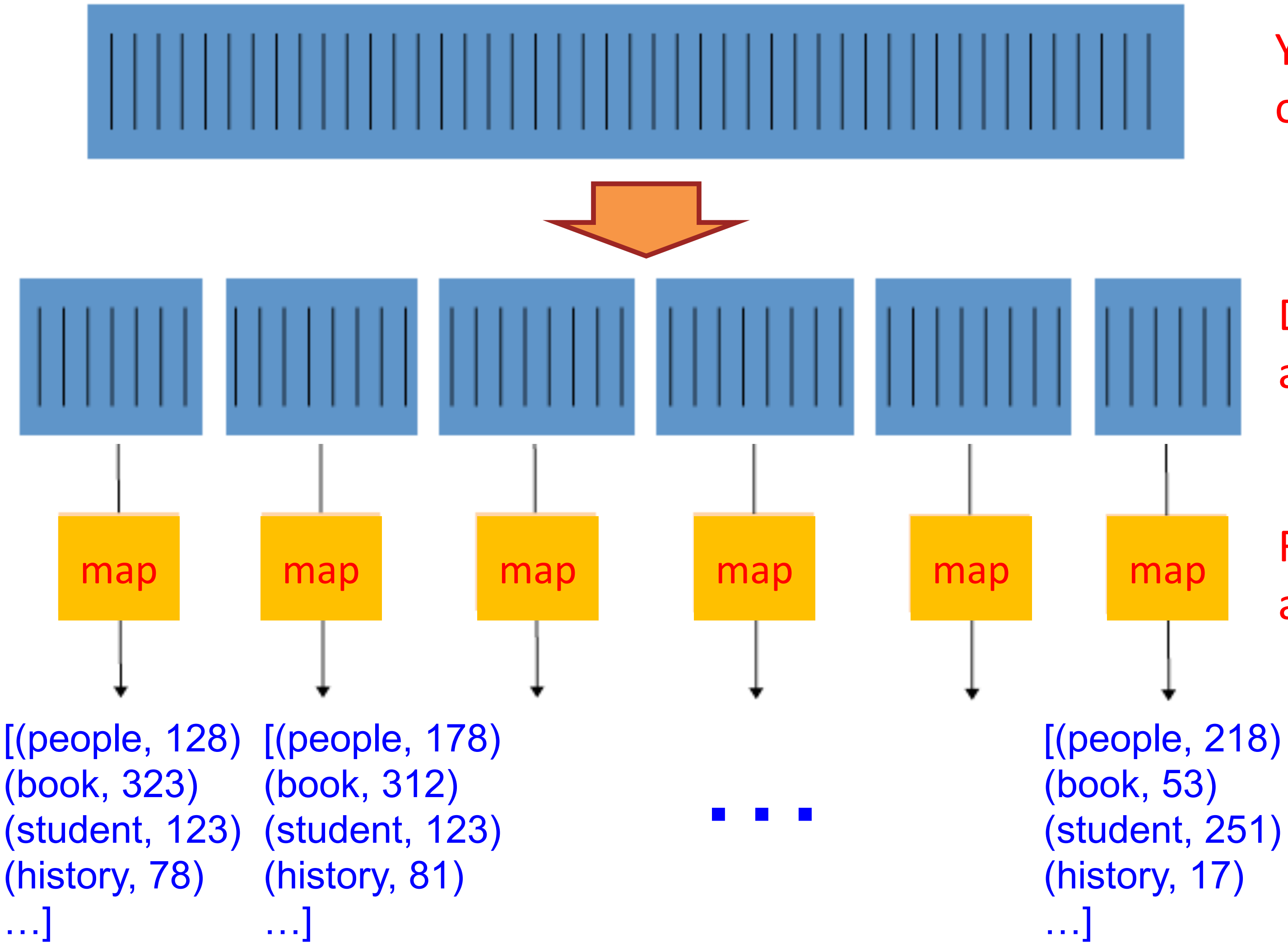
. . .

[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]

MAP

- **map function** processes input key/value pairs to generate a set of intermediate key/value pairs.
- In the 1st example, function “f” **maps** a **TIFF image to a PNG image**.
- In the 2nd example, function “f” **maps** a **document to its most common word**.
- In the 3rd example, function “f” **maps** a **set of documents to its word frequencies**.

Continue Example3 : Compute overall word frequency across 5M docs



[(people, 128)
(book, 323)
(student, 123)
(history, 78)
…]

[(people, 178)
(book, 312)
(student, 123)
(history, 81)
…]

. . .

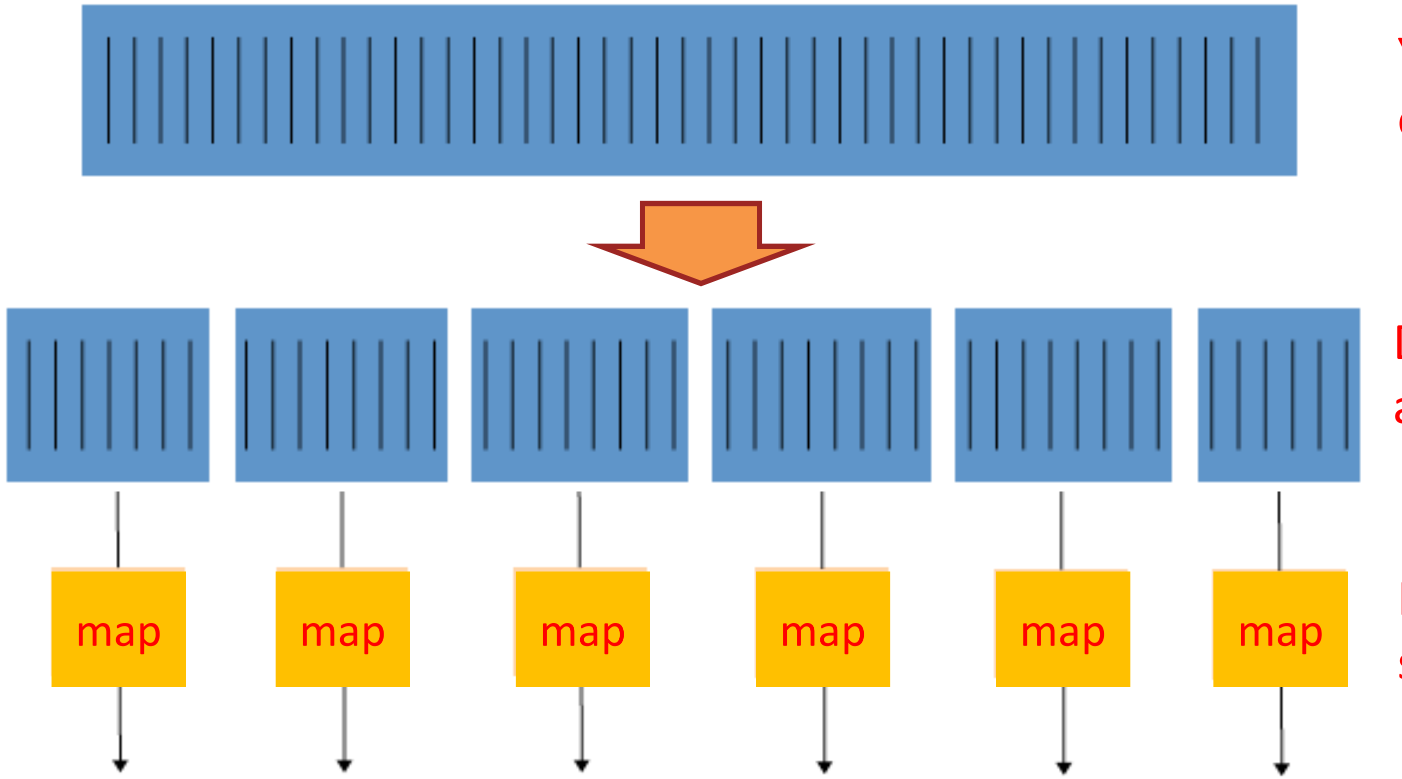
[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]

You have millions of documents

Distribute the documents among k computers

For each document f returns a set of **(word, freq)** pairs

Continue Example3 : Compute overall word frequency across 5M docs

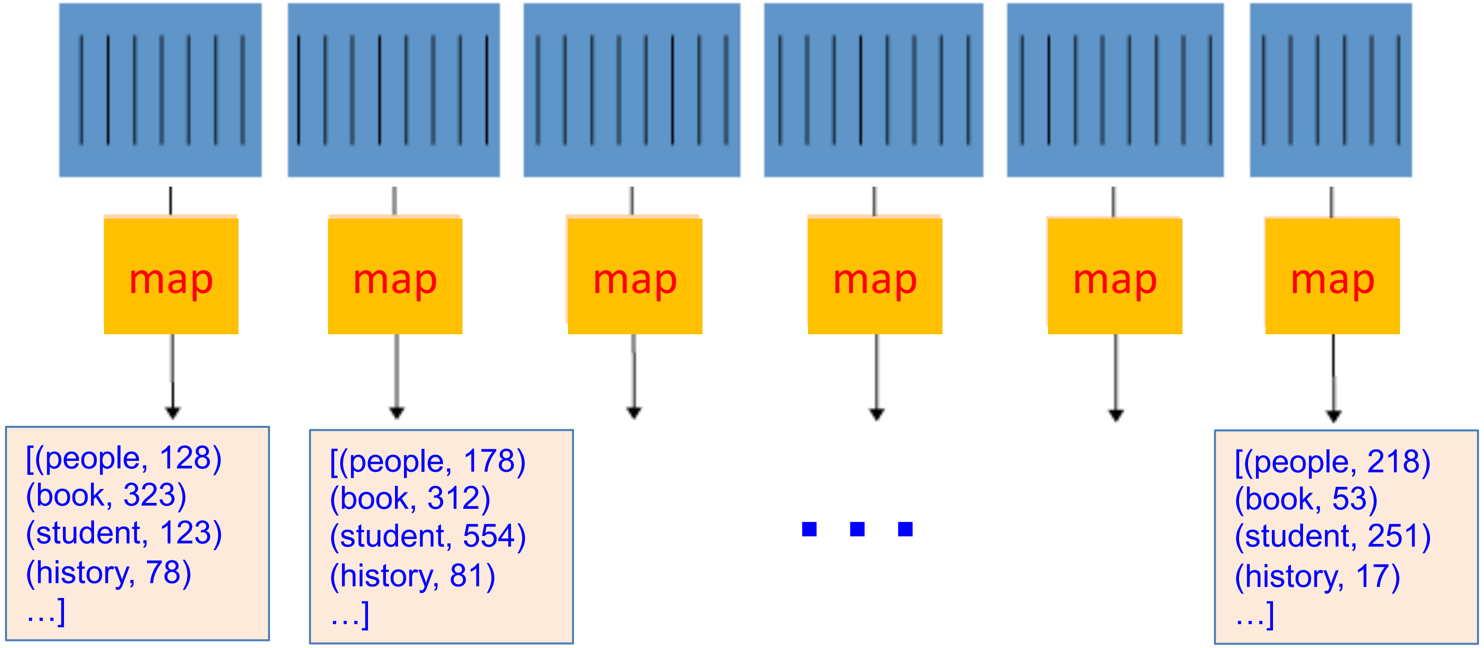


You have millions of documents

Distribute the documents among k computers

For each document f returns a set of **(word, freq)** pairs

Now, we have a list of **(word,freq) pairs for** each set **(chunk) of docs**.
We have to combine all individual lists to build an **overall** list of **(word,freq)** for all documents together.



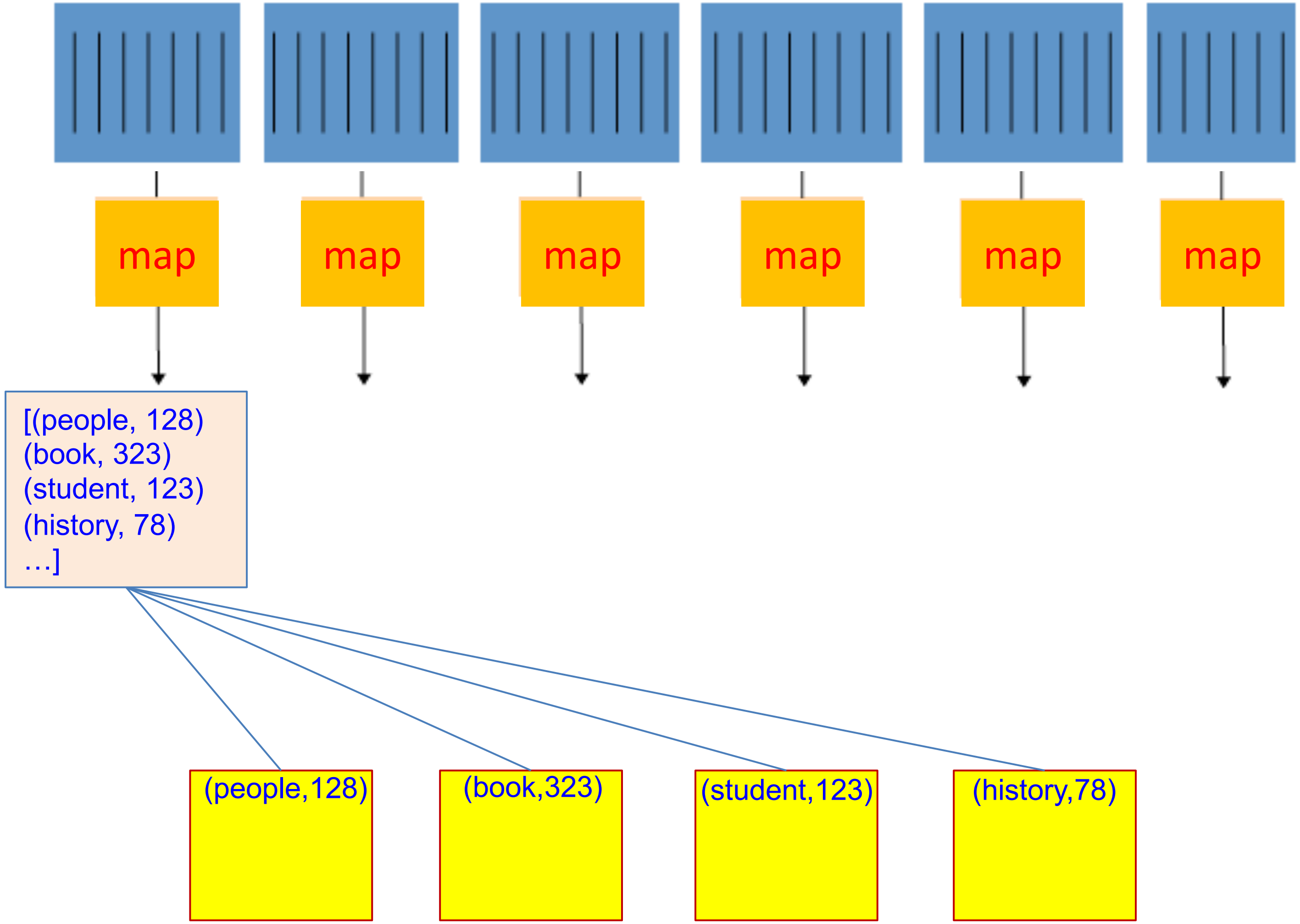
map map map map map map

[(people, 128)
(book, 323)
(student, 123)
(history, 78)
…]

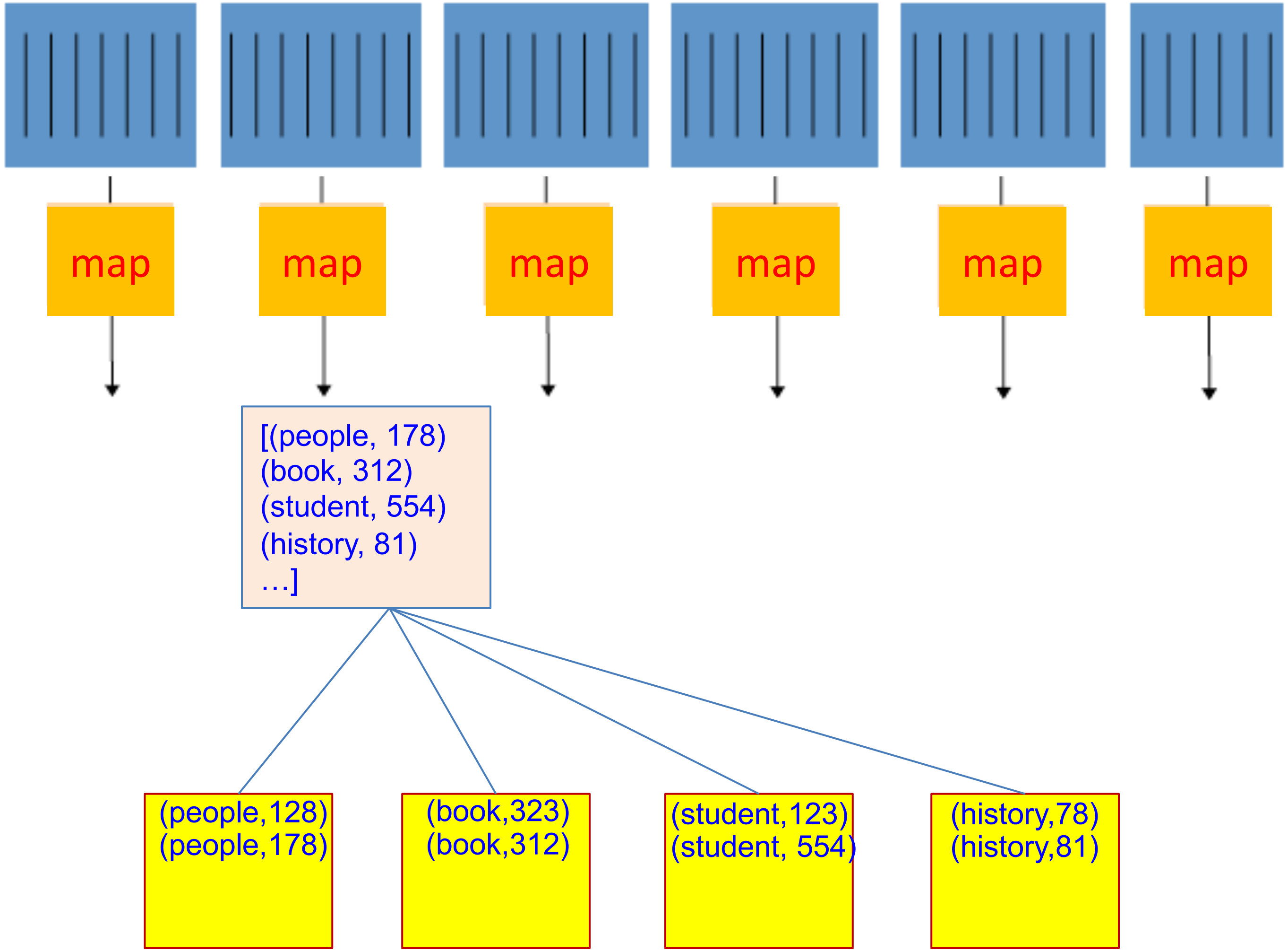
[(people, 178)
(book, 312)
(student, 554)
(history, 81)
…]

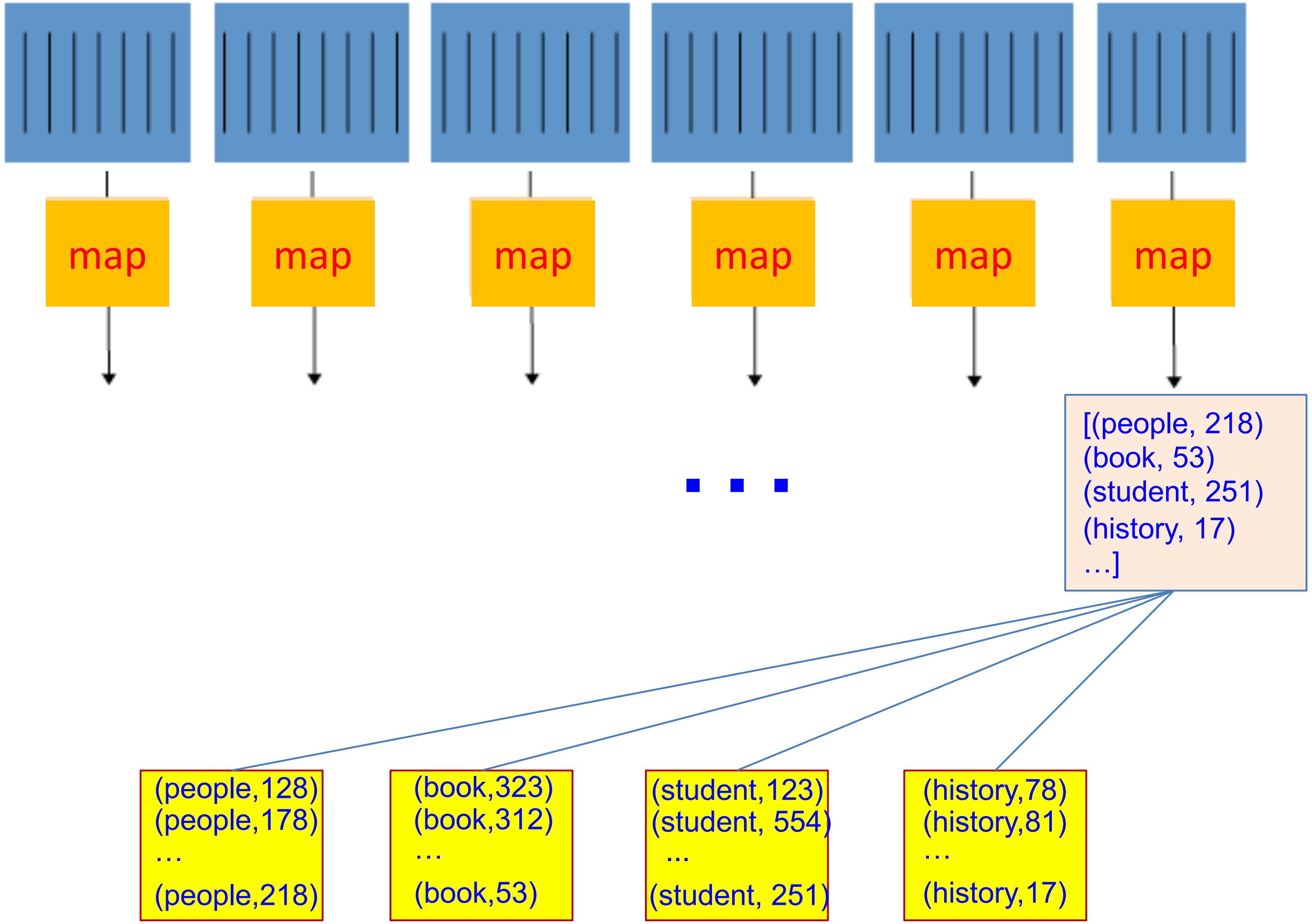
. . .

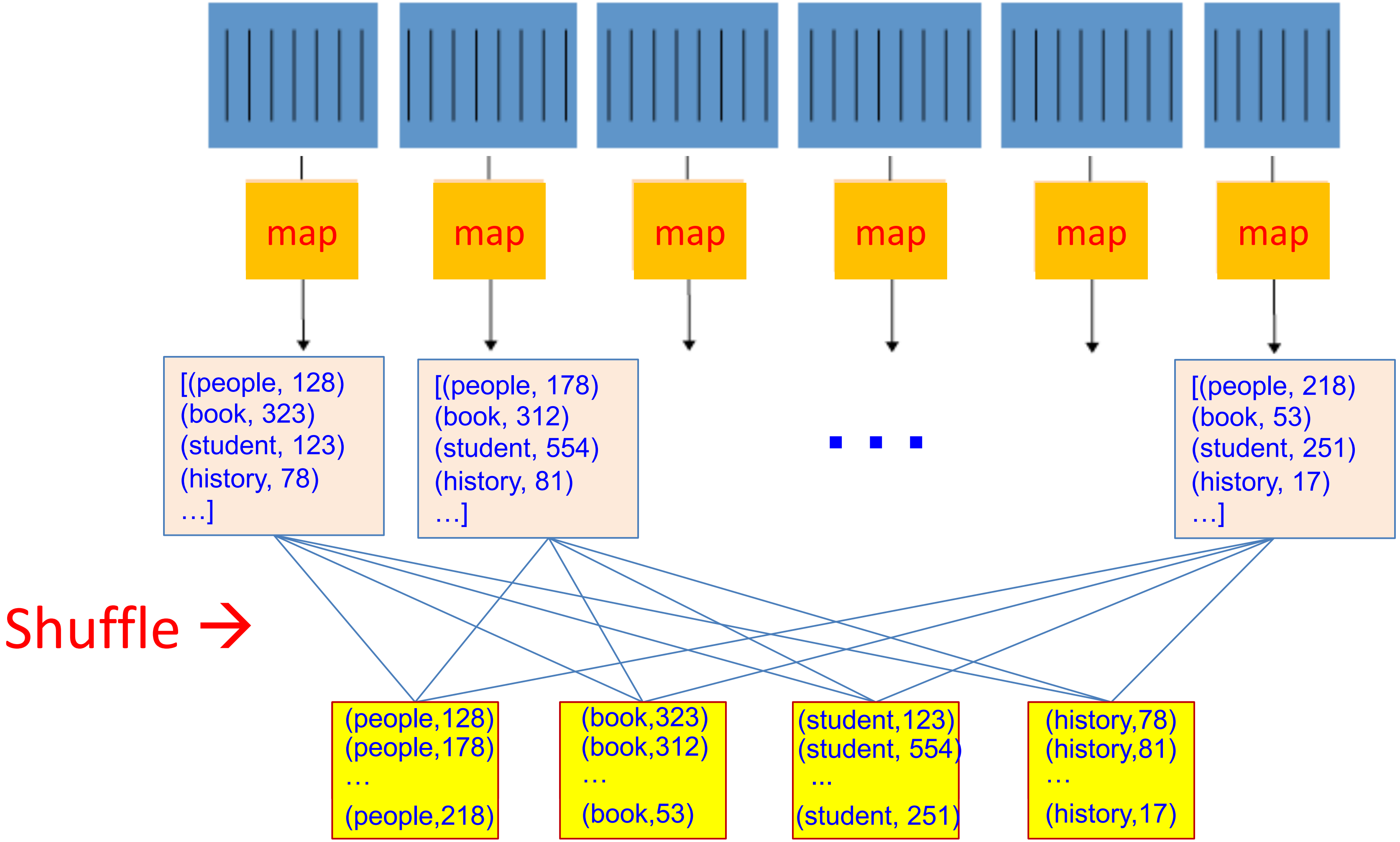
[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]



* Example from Bill Howe, University of Washington


map
map
map
map
map
map
[(people, 178)
(book, 312)
(student, 554)
(history, 81)
…]
(people,128)
(people,178)
(book,323)
(book,312)
(student,123)
(student, 554)
(history,78)
(history,81)


map
map
map
map
map
map
[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]
. . .
(people,128)
(people,178)
…
(people,218)
(book,323)
(book,312)
…
(book,53)
(student,123)
(student, 554)
…
(student, 251)
(history,78)
(history,81)
…
(history,17)



map

map

map

map

map

map

[(people, 128)
(book, 323)
(student, 123)
(history, 78)
…]

[(people, 178)
(book, 312)
(student, 554)
(history, 81)
…]

. . .

[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]

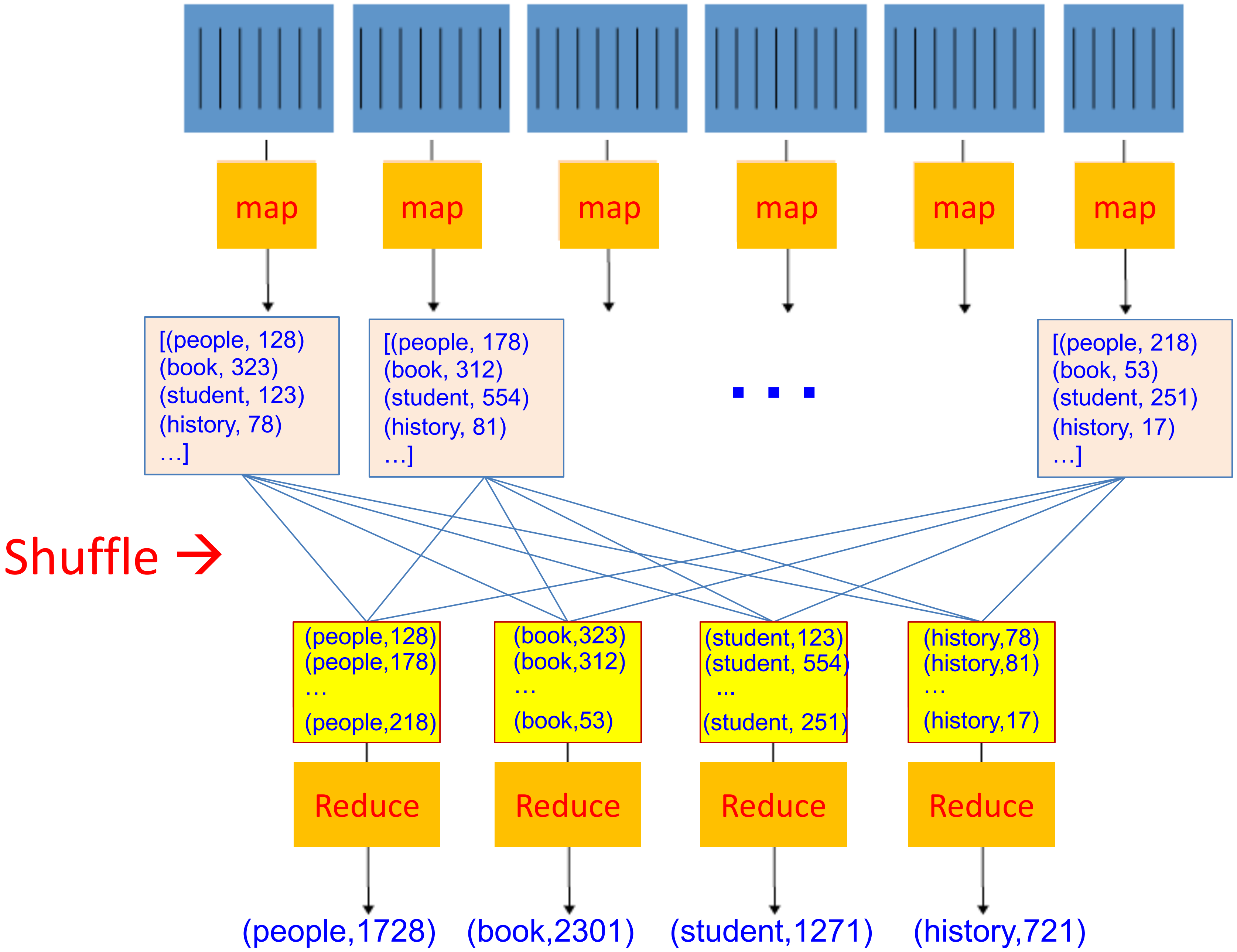
Shuffle →

(people,128)
(people,178)
…
(people,218)

(book,323)
(book,312)
…
(book,53)

(student,123)
(student, 554)
…
(student, 251)

(history,78)
(history,81)
…
(history,17)



map map map map map map

[(people, 128)
(book, 323)
(student, 123)
(history, 78)
…]

[(people, 178)
(book, 312)
(student, 554)
(history, 81)
…]

. . .

[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]

Shuffle →

(people,128)
(people,178)
…
(people,218)

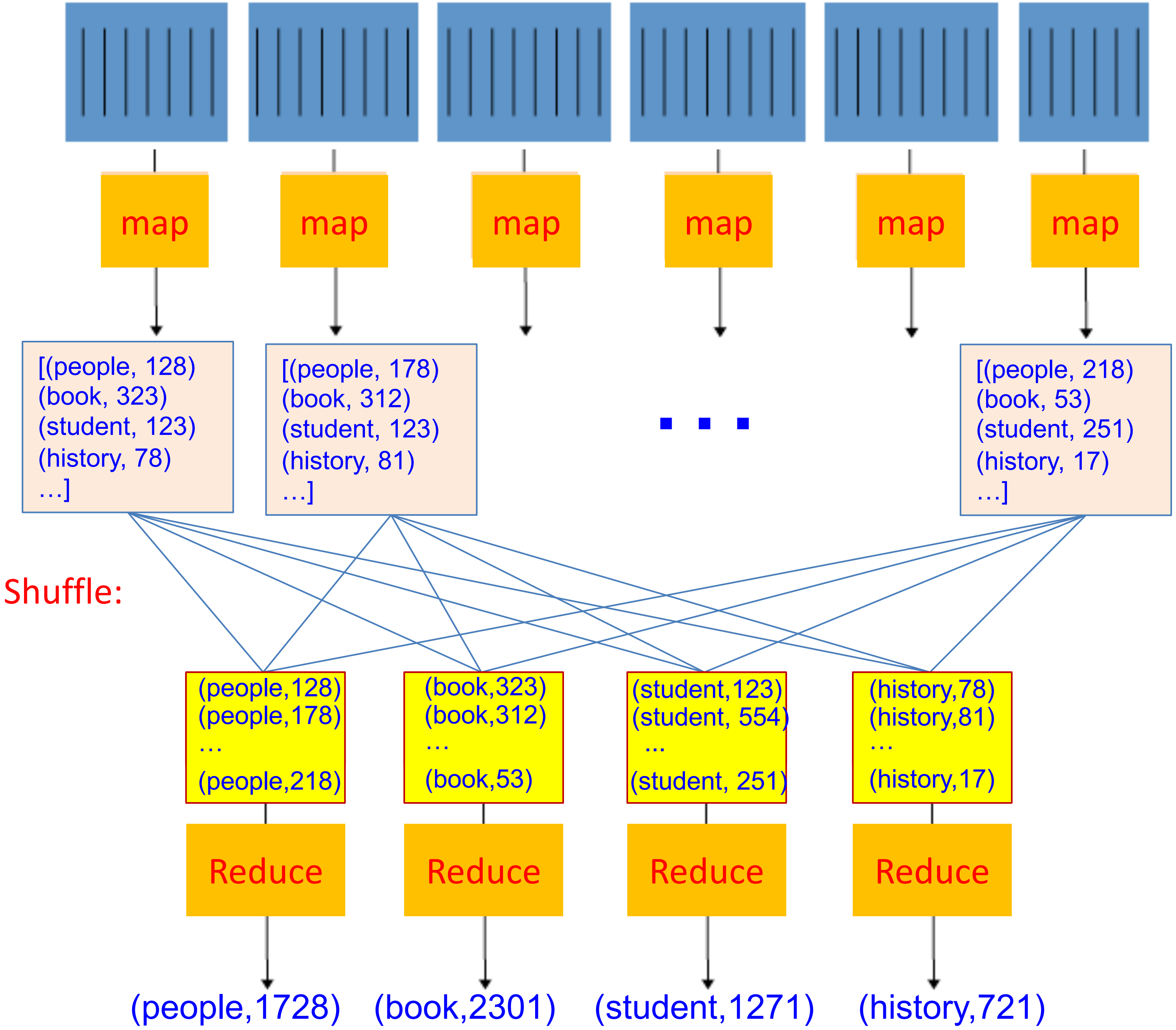
(book,323)
(book,312)
…
(book,53)

(student,123)
(student, 554)
…
(student, 251)

(history,78)
(history,81)
…
(history,17)

Reduce Reduce Reduce Reduce

(people,1728) (book,2301) (student,1271) (history,721)



map map map map map map

[(people, 128)
(book, 323)
(student, 123)
(history, 78)
…]

[(people, 178)
(book, 312)
(student, 123)
(history, 81)
…]

. . .

[(people, 218)
(book, 53)
(student, 251)
(history, 17)
…]

Shuffle:

(people,128)
(people,178)
…
(people,218)

(book,323)
(book,312)
…
(book,53)

(student,123)
(student, 554)
…
(student, 251)

(history,78)
(history,81)
…
(history,17)

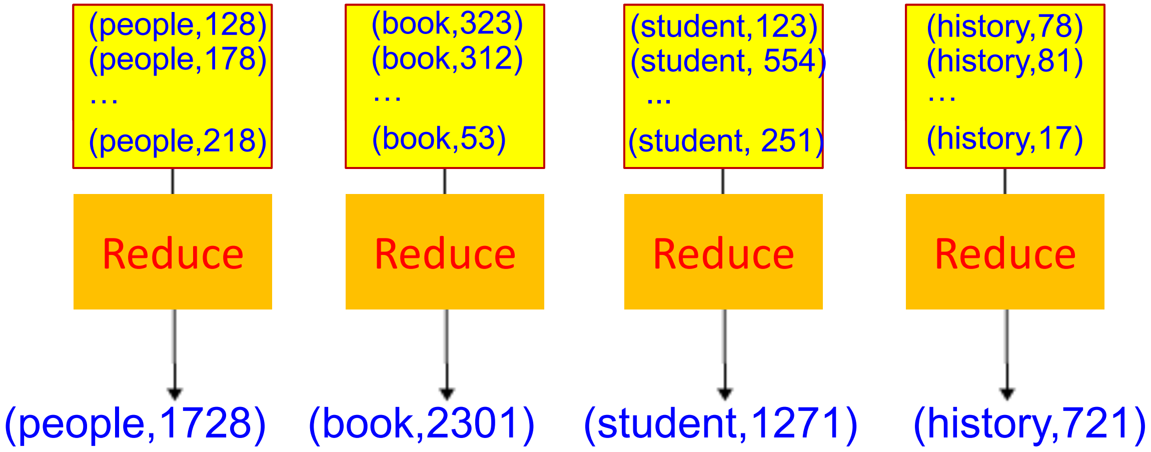
Reduce Reduce Reduce Reduce

(people,1728) (book,2301) (student,1271) (history,721)

Map: Counts **All words in Each chunk of data**

Reduce: Counts **Each word in the Entire data**

REDUCE



- **reduce function** merges all intermediate values associated with the same intermediate key.
- Note: For the sake of simplicity, In this example we only assigned a single word to each machine in Reduce stage. In practice, each machine in Reduce stage will take care of a set of words!

MAP-REDUCE

- In this example:
  - **Map:** Counts **All** words in **Each** chunk of data
  - **Reduce:** Counts **Each** word in the **Entire** dataset

MapReduce Programming Model

- In MapReduce model the Input and Output data is in the form of Key-Value Pairs:
- **Input** : a set of (in_key , in_value) pairs
- **Output**: a set of (out_key , out_value) pairs

- Programmer specifies two functions:

map (in_key, in_value) -> list of (out_key, intermediate_value)

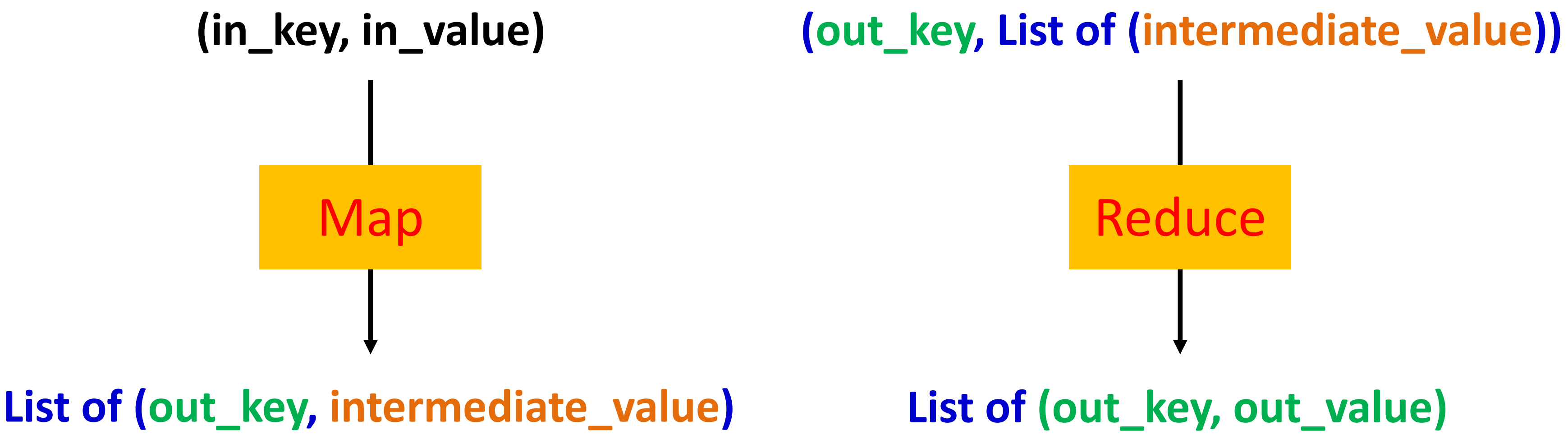
- Processes input (key,value) pairs
- Produces set of intermediate pairs

reduce (out_key, list of (intermediate_value)) -> list of (out_key, out_value)

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

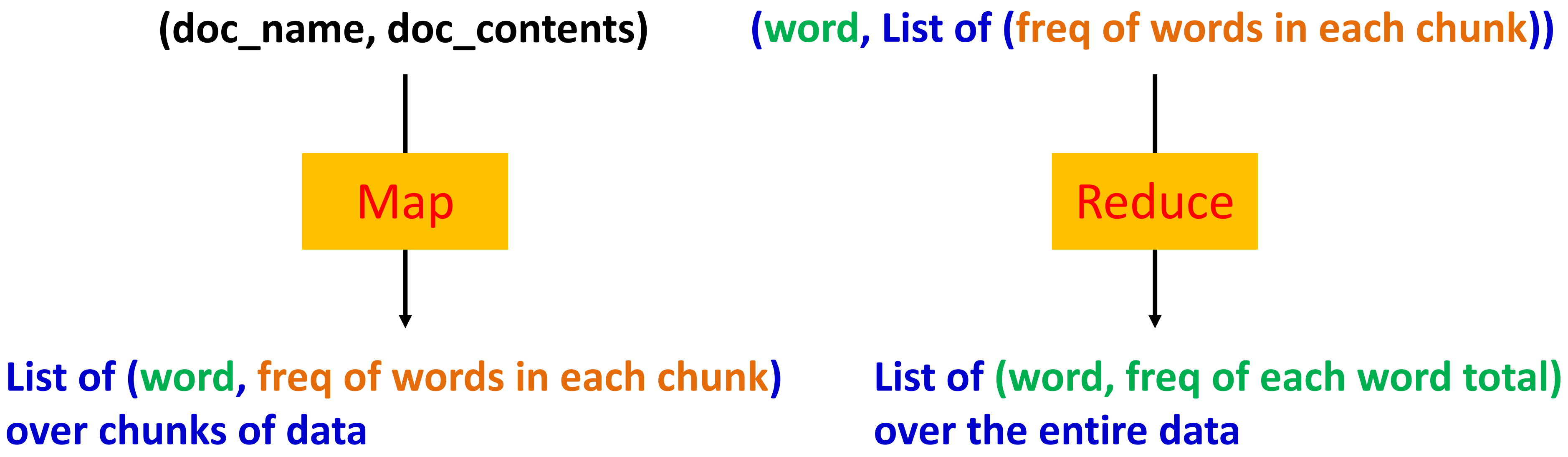
Inputs and Outputs

- Input : a set of (in_key,in_value) pairs
- Output: a set of (out_key,out_value) pairs


(in_key, in_value)
Map
List of (out_key, intermediate_value)
(out_key, List of (intermediate_value))
Reduce
List of (out_key, out_value)

Inputs and Outputs (Example)

- Input : a set of (in_key,in_value) pairs
- Output: a set of (out_key,out_value) pairs



(doc_name, doc_contents)

Map

List of (word, freq of words in each chunk) over chunks of data

(word, List of (freq of words in each chunk))

Reduce

List of (word, freq of each word total) over the entire data

Example: Compute the overall word frequency across 5M documents

```
map(String input_key, String input_value):
Intermediate_values = {}
for each word w in input_value:
        Intermediate_values[w] += 1



reduce(String output_key, Iterator intermediate_values):
output_values = 0;
for each v in intermediate_values:
        output_values += v
        Emit(output_key,output_values )
```

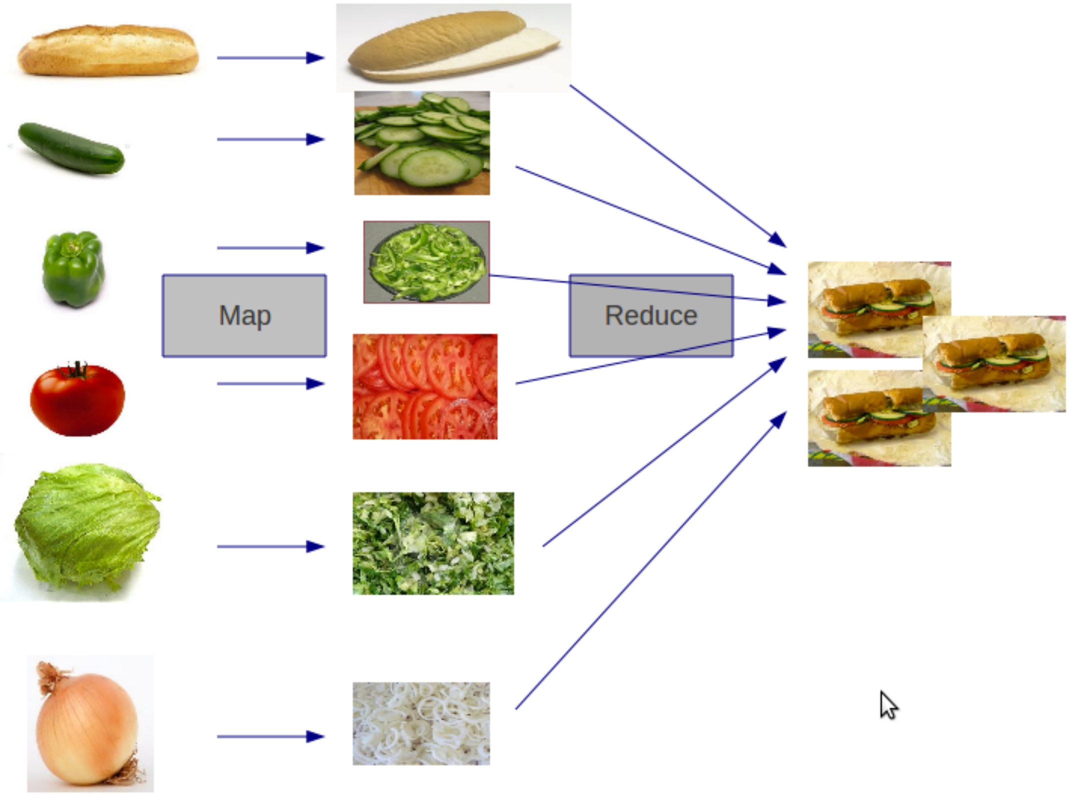
- **Very important to distinguish between (in_key, in_value) and (out_key, out_value)!!!**

```
map(String input_key, String input_value):
        # input_key: document name/id
        # input_value: document contents
Intermediate_values = {}
for each word w in input_value:
        Intermediate_values[w] += 1


reduce(String output_key, List of [intermediate_values]):
        # output_key: word
        # output_values: freq of each word
output_values = 0;
for each v in intermediate_values:
        output_values += v
return(output_key,output_values )
```

Some Notes about Map Reduce

- Everything is in the form of **key-value** pairs!
- In map stage, **parallelism** is achieved since different parts of data can be processed by different machines simultaneously.
- In reduce stage, **parallelism** is achieved as reducers operating on different keys simultaneously.
- Mappers manipulate the keys, but reducers do not usually change the keys.
- All mappers need to finish before reducers can begin.
- A map-reduce program may consist of several rounds of different map and reduce functions.



Thank You!

Questions?