DASCI

Instituto Andaluz Interuniversitario
en Ciencia de Datos e
Inteligencia Computacional

Ciencia de Datos a través del Big Data

Diego García (djgarcia@ugr.es)
Isaac Triguero (isaaktriguero@ugr.es)

Financiado por
la Unión Europea
NextGenerationEU

GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
E INTELIGENCIA ARTIFICIAL

Plan de
Recuperación,
Transformación
y Resiliencia

UNIVERSIDAD
DE GRANADA

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

REPSOL

CAMBRIDGE UNIVERSITY PRESS

Chapter 2

MapReduce II

© Isaac Triguero and Mikel Galar

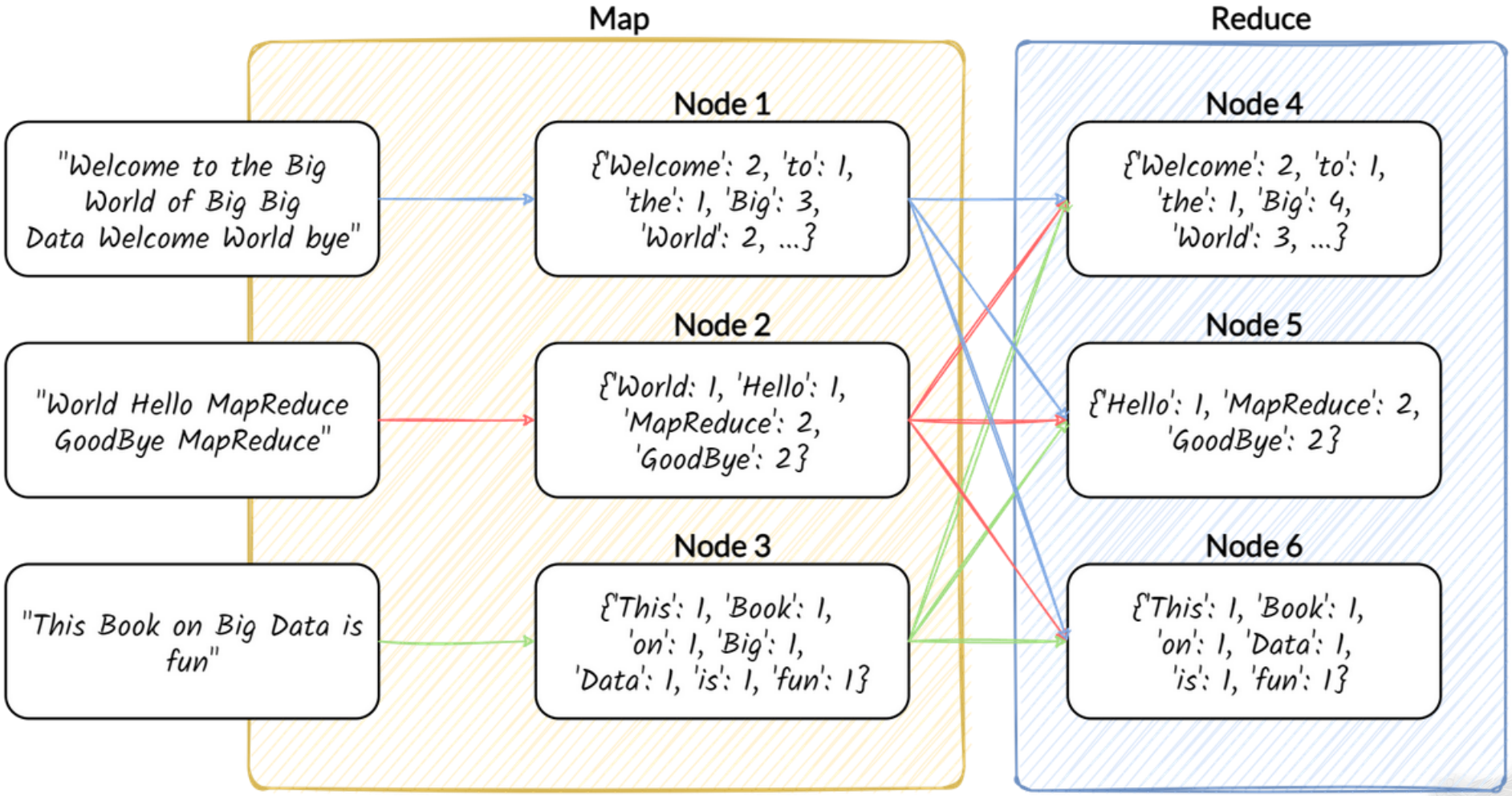
Learning Outcomes

- Learn a functional programming paradigm for distributed computing with Big Data [KU]
- Understand why transparency and fault-tolerance are desirable in Big Data [KU]
- How to use map reduce in practice for problem solving [IS]
- How to exploit map reduce to preserve data locality principles, minimizing data movement [IS]

CAMBRIDGE UNIVERSITY PRESS

What if the document is really BIG?

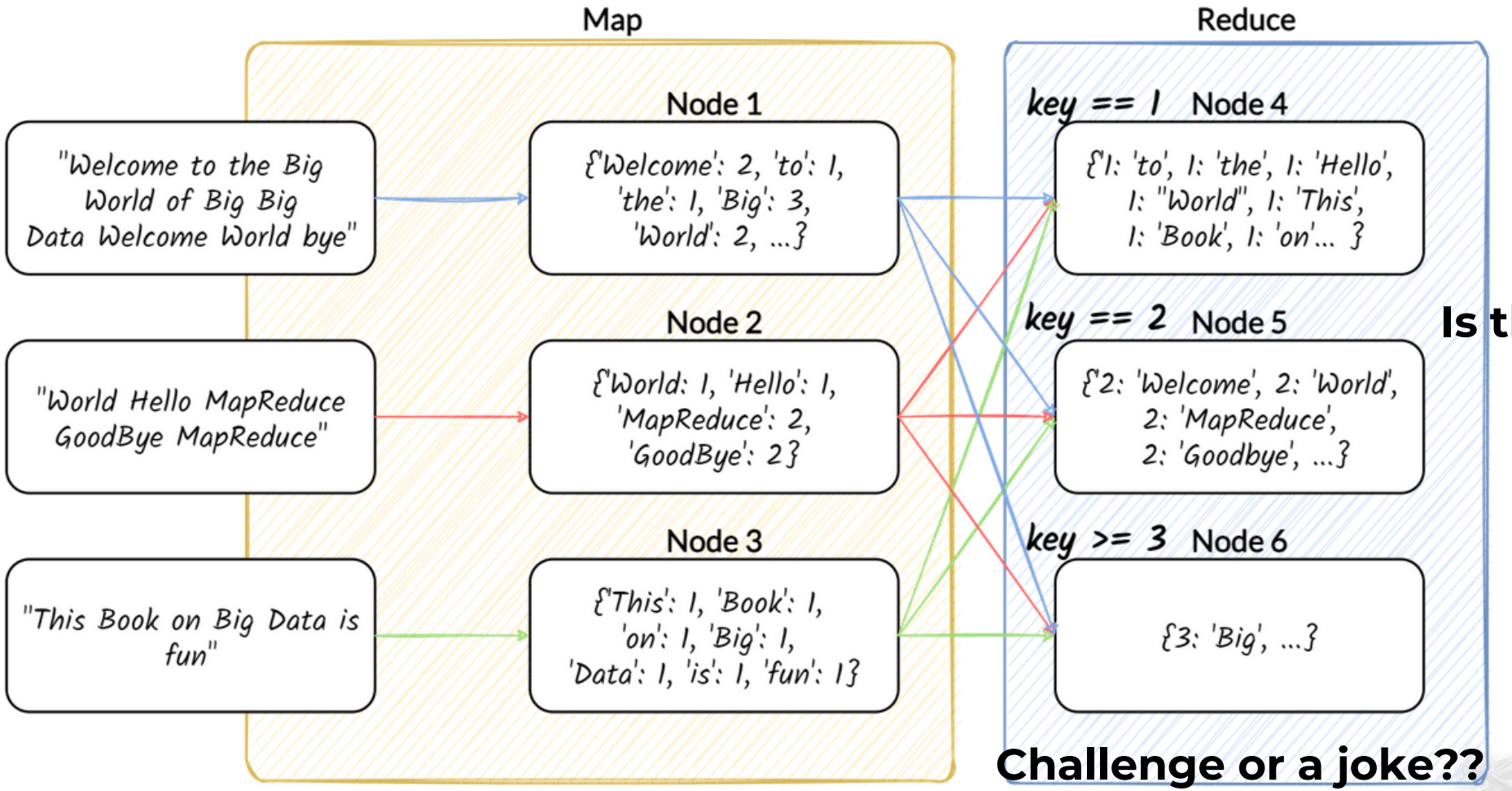
Let's 'MapReduce' this!



Challenge #1: MapReduce to sort words by frequency

What if the document is really BIG?

The Map Phase inverts key/values



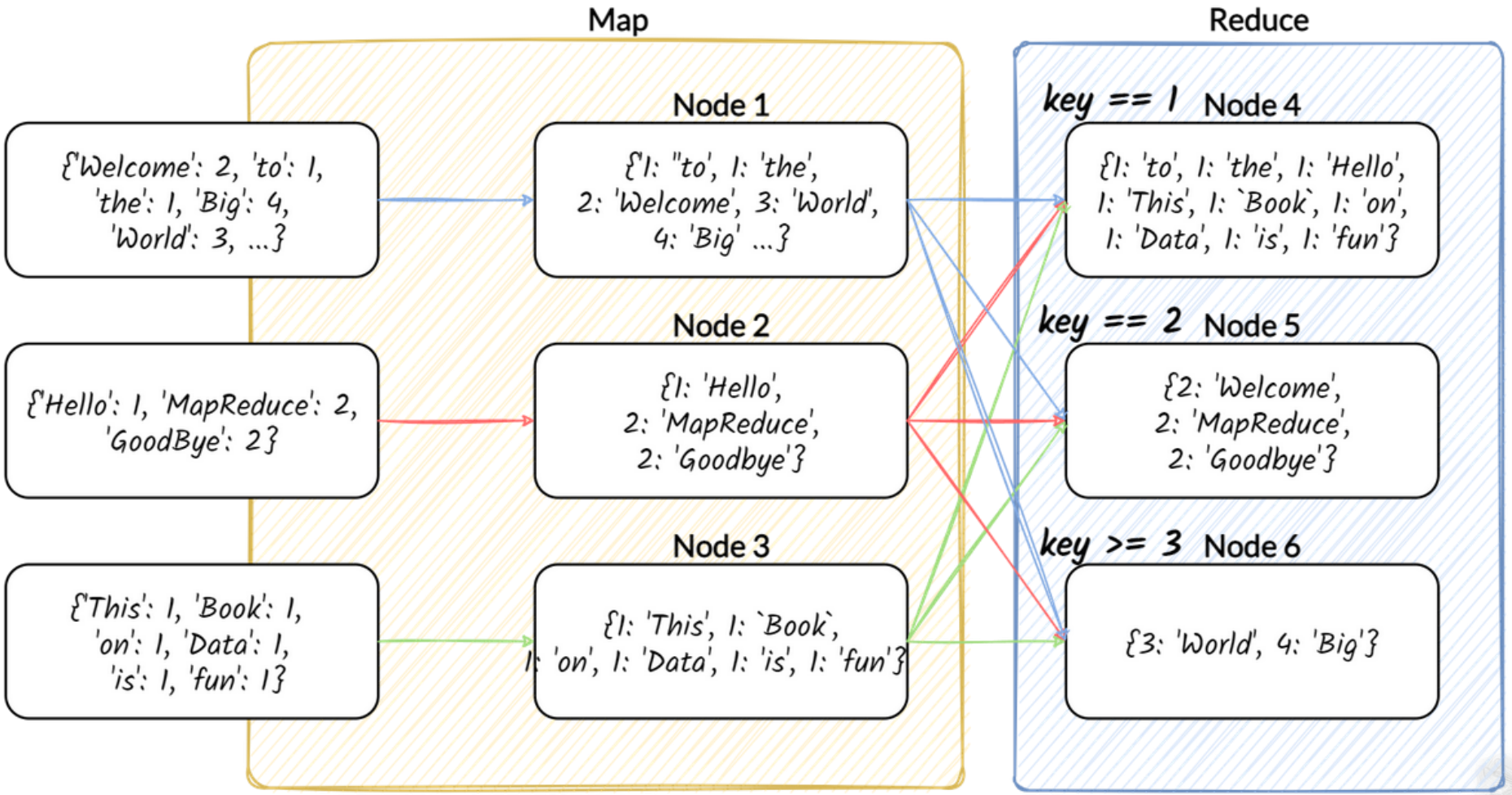
Is this correct?

Challenge or a joke??

~~Challenge~~ Joke #1: MapReduce to sort words by frequency

What if the document is really BIG?

The Map Phase inverts key/values



Input: the output of our Word Count

Objective for today

- Working with Key-Value Pairs
- Combiners
- Internal Working

Working with Key-Value Pairs

- Based on **functional programming** (e.g., Haskell)
  - Operates on **<key, value>** pairs, for example:
    - Text document example:

      key = line number ; **value** = text in that line
    - Classification dataset:

      key = instance number ; **value** = instance itself
    - Graph-based example:

      key = node, **value** = adjacency list
  - These documents are going to be on a distributed file system which is accessible from different nodes

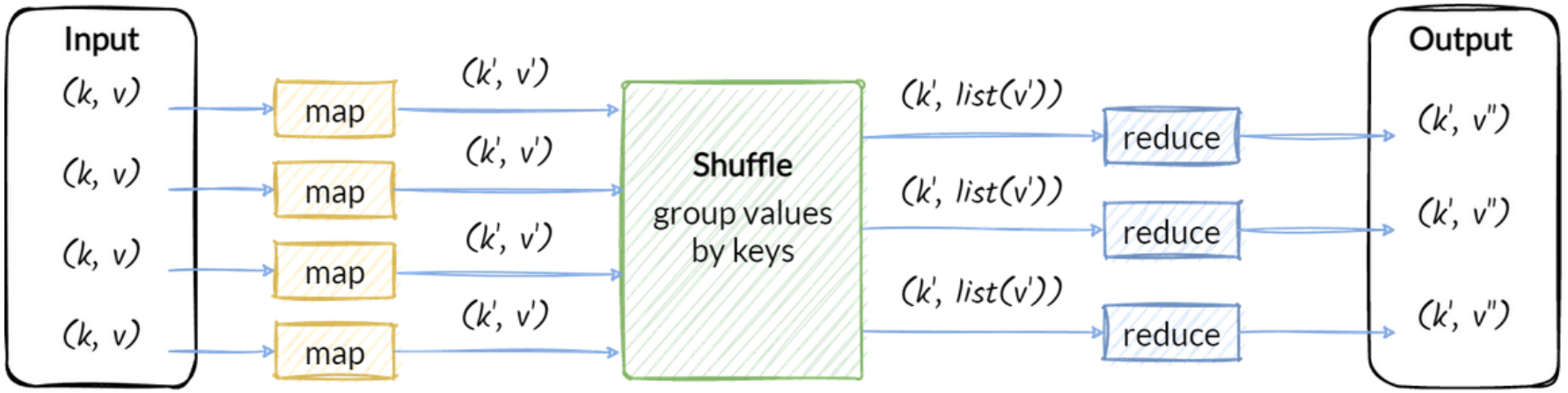
Working with Key-Value Pairs

- Users specifies two functions:

map: `(k,v) → list[k', v']`

reduce: `(k', list[v']) → list[k', v'']`

- The input "key" for the map function is typically not of interest (it could simply be metadata, like a filename)
- Automatic sorting of intermediate keys between map and reduce phases



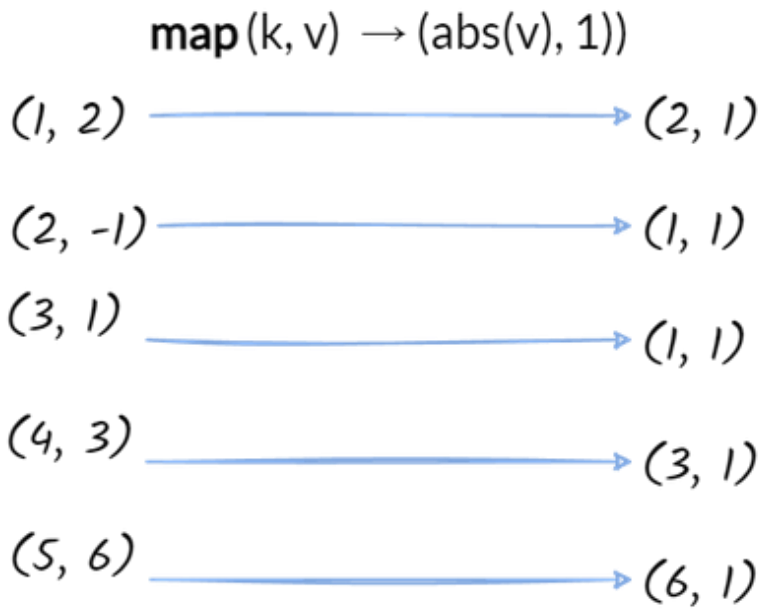
Dean, Jeffrey, and Ghemawat, Sanjay. 2004. MapReduce: Simplified Data Processing on Large Clusters. Pages 137–150 of: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. Link

Working with Key-Value Pairs

- The **map function** takes each <k, v> pair and returns a number of key-values <k', v'>, by performing an operation in each pair

```
map(k, v) = (abs(v), 1)
```

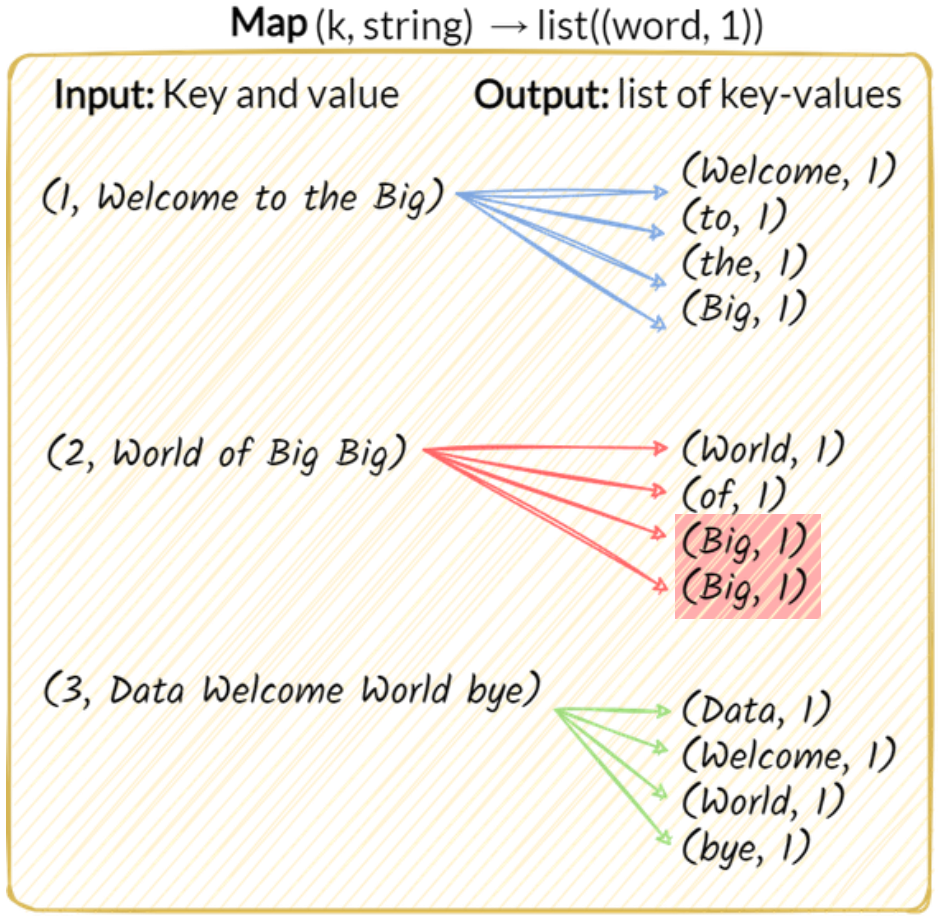
`[(1, 2), (2, -1), (3, 1), (4, 3), (5, 6)]` → `[(2, 1), (1, 1), (1, 1), (3, 1), (6, 1)]`



In this example, the map function returns only one element (k2, v2) for each input

Working with Key-Value Pairs

- The **map function** takes each <k, v> pair and returns a number of key-values <k', v'>, by performing an operation in each pair



Combiners!

Working with Key-Value Pairs

- **Shuffle phase (automatic!)**: Group by key
  - The key-value pairs from the map function are grouped together by key! The values are put together in a list list(v') .

`list(k',v')` → `(k', list(v'))`

  - All pairs with the same key will be sent to the same node, and the pair `(k', list(v'))` is then processed in that node

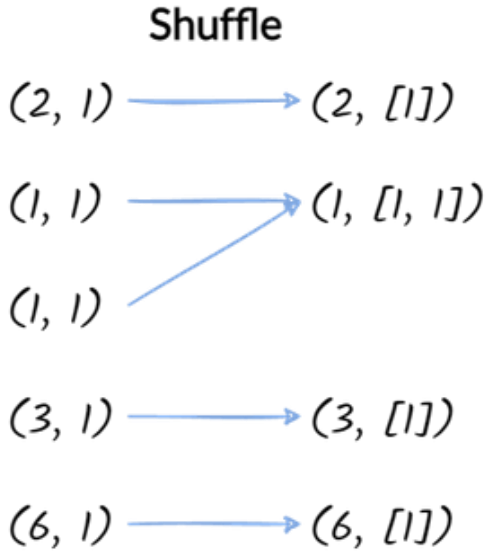
`[(1, 2), (2, -1), (3, 1), (4, 3), (5, 6)]` →

`map(k, v) = (abs(v), 1)`

`[(2, 1), (1, 1), (1, 1), (3, 1), (6, 1)]` →

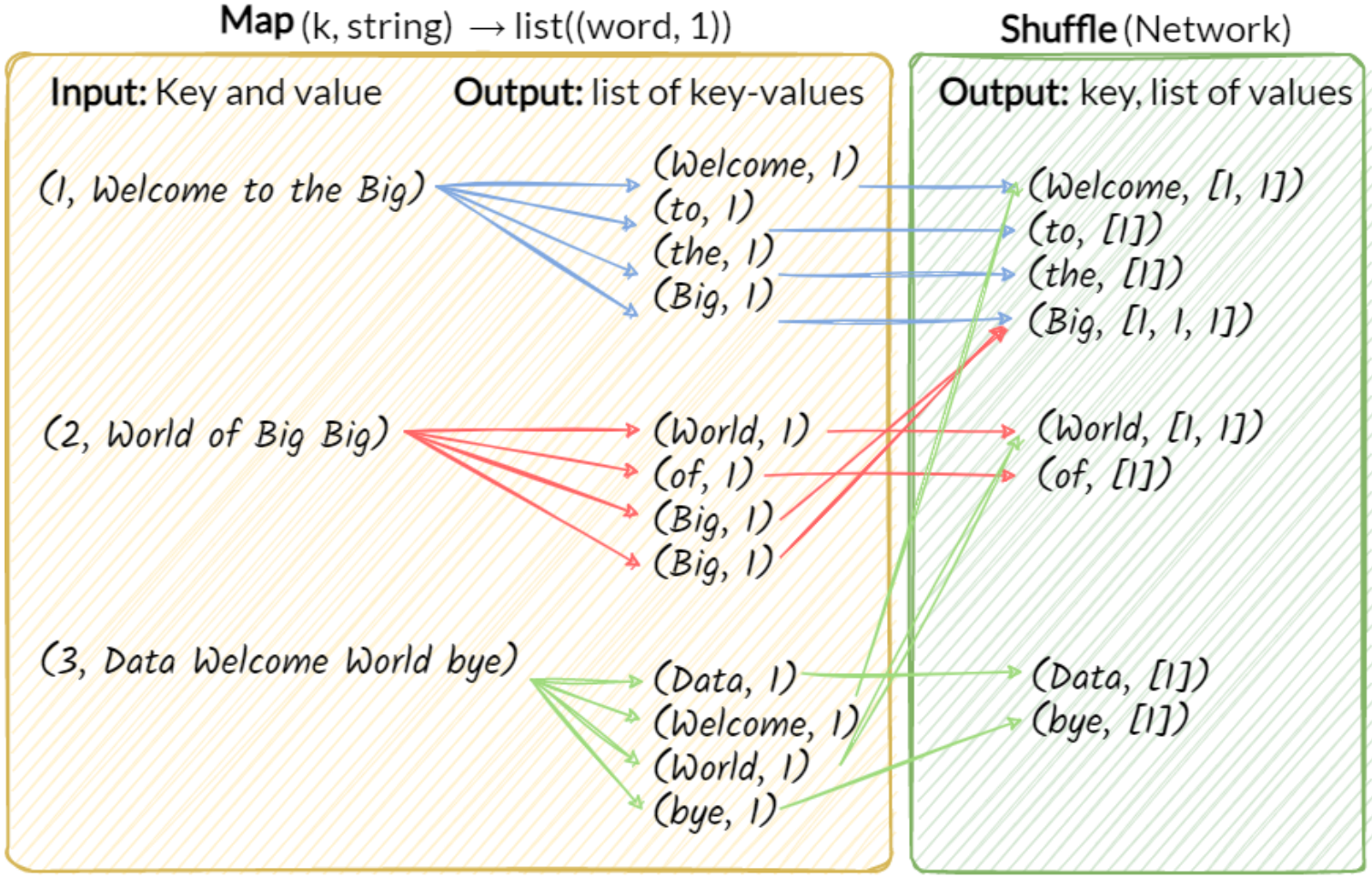
`shuffle (automatic)`

`[(2, [1]), (1, [1, 1]), (3, [1]), (6, [1])]`



Working with Key-Value Pairs

- **Shuffle phase (automatic!):** `list(k',v') → (k', list(v'))`



Working with Key-Value Pairs

- **Reduce phase**
  - Takes a `(k', list(v'))` pair and reduces the `list(v')` into a single v'', returning a new pair `(k', v'')`

`reduce(k',list(v')) → (k',v'')`

  - The reduce function is (independently) applied to each different key!

`[(1, 2), (2, -1), (3, 1), (4, 3), (5, 6)] →`

`map(k, v) = (abs(v), 1)`

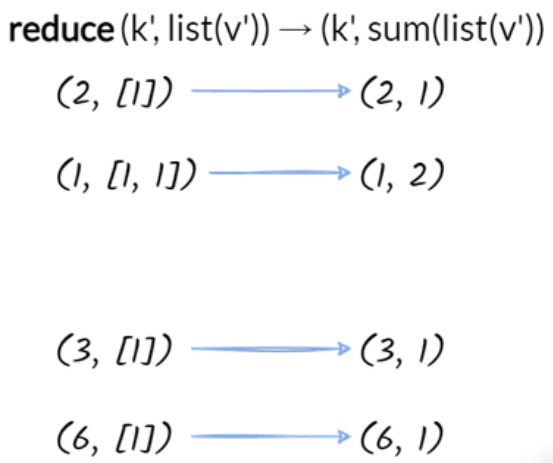
`[(2, 1), (1, 1), (1, 1), (3, 1), (6, 1)] →`

`shuffle (automatic)`

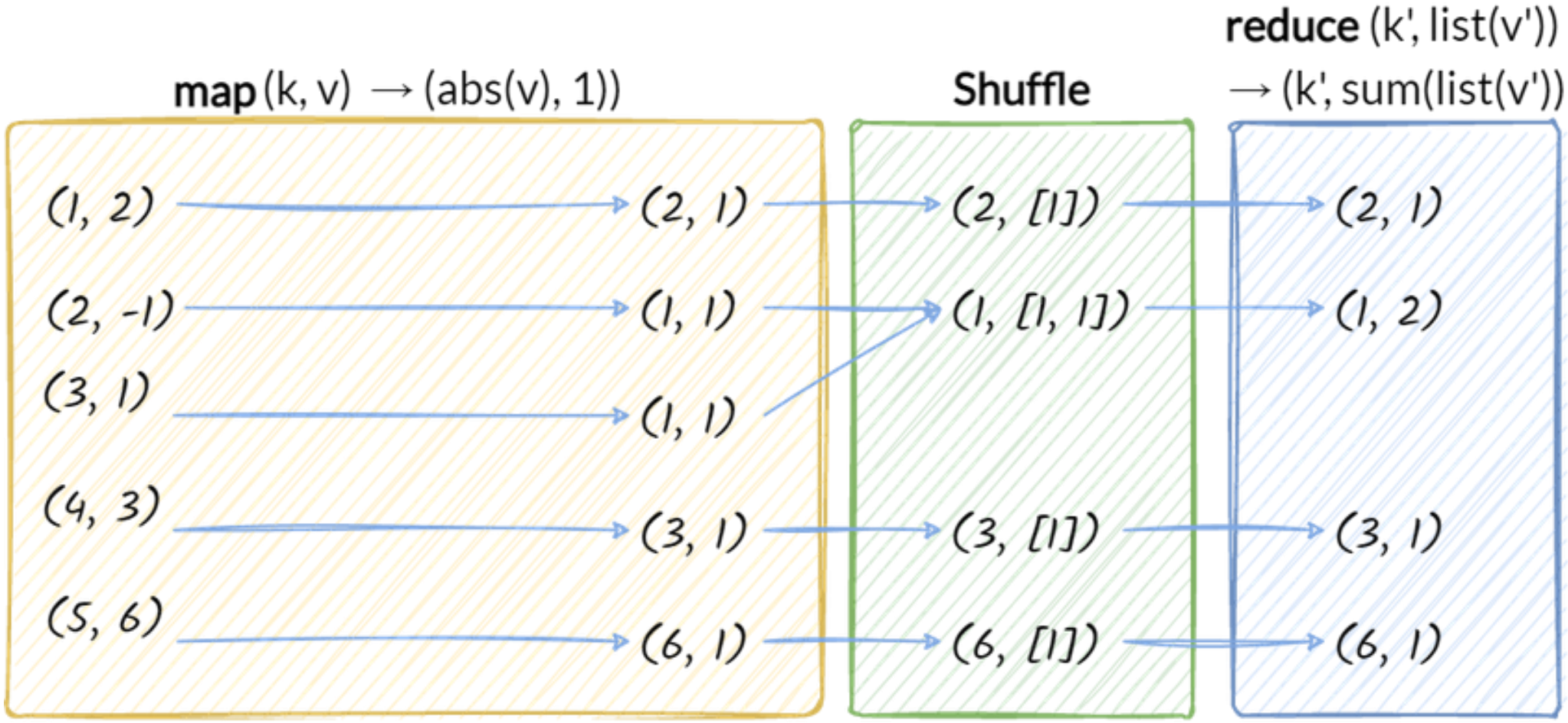
`[(2, [1]), (1, [1, 1]), (3, [1]), (6, [1])] →`

`reduce(k', list(v')) = (k', sum(list(v'')))`

`[(2, 1), (1, 2), (3, 1), (6, 1)]`

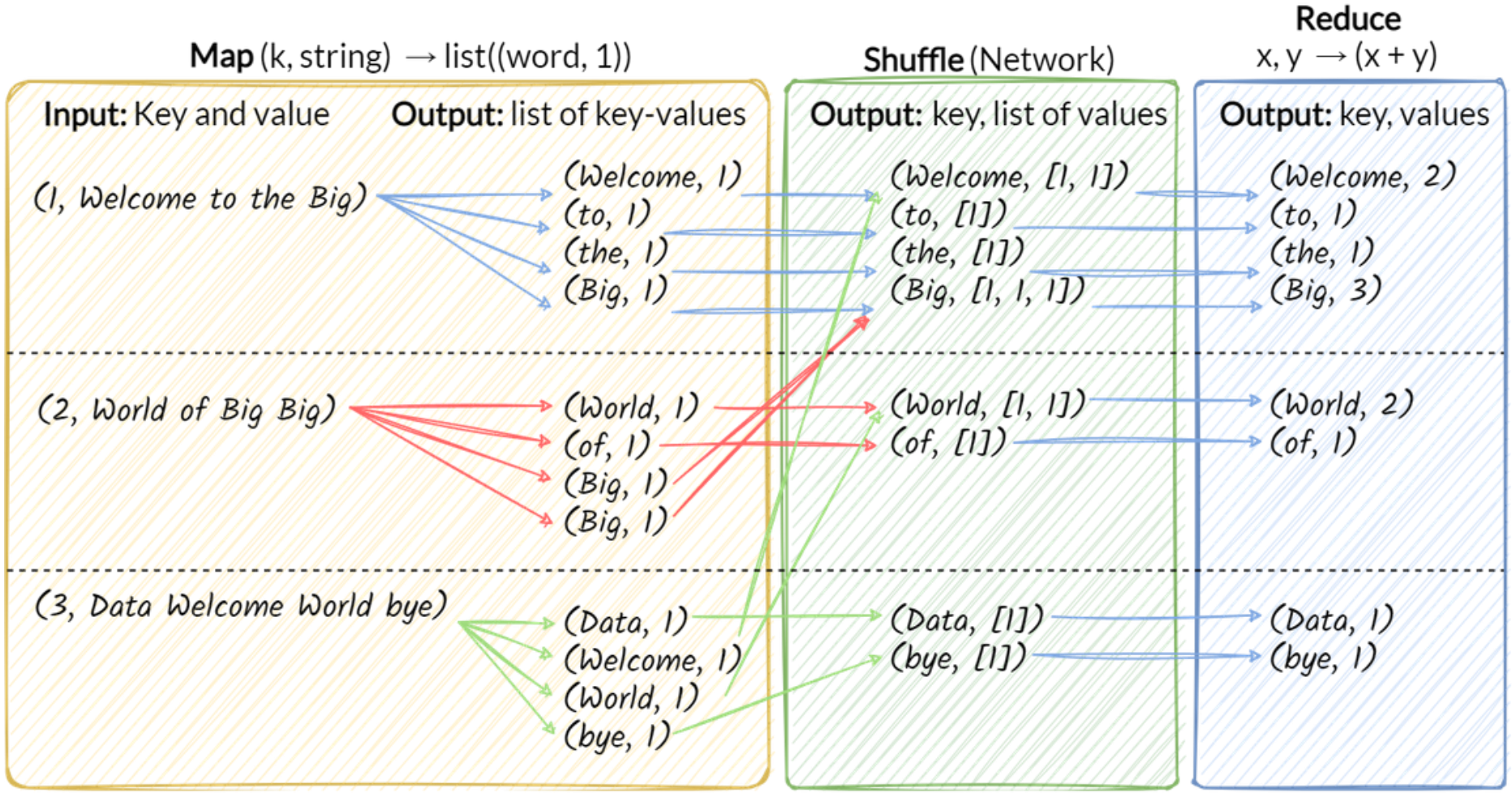


Working with Key-Value Pairs



Working with Key-Value Pairs

- **Reduce phase**: reduce values of the list

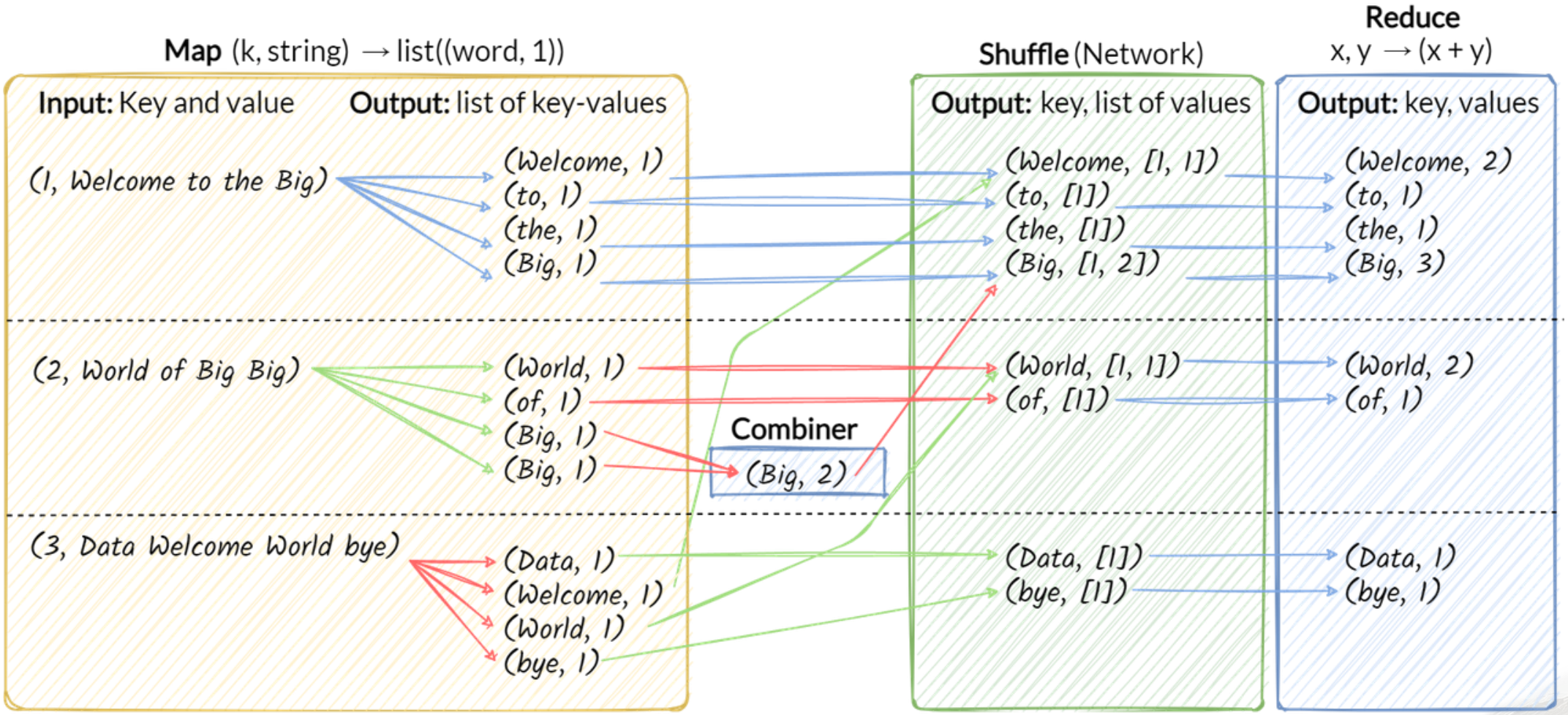


Combiners

- **Combiners**: local aggregators
  - One of the bottleneck of a MapReduce program could be in the shuffle phase if we send out lots of data through the network
  - They work exactly like a reduce, taking `(k', list(v'))` pair and reduces the `list(v')` into a single `v''`, returning a new pair `(k', v'')`
  - Useful when there is a significant repetition in the intermediate keys produced by each map task (e.g., Word Count)
  - Optional, it is not always run, the implementation of MapReduce (e.g., Hadoop or Spark) may decide not to if sending the data is simply faster

Combiners

- **Combiners**: local aggregators



Challenge: Compute Max Temperature

- We have been provided with a large file that contains temperature data. In each line we have the date (`YYYY-MM-DD`) and the temperature, e.g.,
  - `2015-07-13; 24`
  - `2012-07-11; 24`
  - `2011-08-13; 43`
  - `2015-09-11; 17`
  - …
- **Objective**: Determine the maximum temperature each year.
- **Exercise**: Come up with a MapReduce solution for this

Internal Working

- Data is distributed across nodes in data blocks (Distributed File system, usually blocks of 128MB)
- Two main processes: **mapper** and **reducer**
  - A **mapper** is a process that executes the map function for each key-value pair in a block of data. A mapper could manage multiple blocks
  - The **shuffle** phase sends the `(k', list(v'))` to the same reducer
  - The **reducer** is also a process that runs the reduce function for each k'

Internal Working

- **Limitation**: too many HDD usage
  - **Mappers**
    - Write to a buffer in memory which will might be written on the drive if they don't fit in main memory
  - **Shuffle**
    - All data transfer occur in this process (it could be very slow!)
  - **Reducers**
    - Read the data again from drive (if they are not in the buffer), and then write the result into the drive again
  - **Remember**: HDDs are also quite slow…
    - More details when we talk about Hadoop MapReduce

Internal Working

- **Note**: The number of **mappers** is independent of the number of available nodes. It will be defined by the size and characteristics of the dataset tackled

- **Caveat**: The map phase may create very small subsets. In data mining, this could be an issue!

Take-home message

- **MapReduce** key distinguishing features
  - **A new level of abstraction** (transparent to the programmer)
  - **Fault tolerant** (MapReduce)
  - **Data locality and redundancy** (HDFS) through Hadoop implementation
  - **Automatic parallelization**
    - Depending on the size of the input data → there will be multiple MAP tasks!
    - Depending on the number of Keys `<k, value>` → there will be multiple REDUCE tasks!
  - **Scalability**
    - It may work over every data center or cluster of computers

Take-home message

- **MapReduce working**
  - Three main stages: `map`, `shuffle` and `reduce`
  - It is not magic! We need to know how to implement `map` and `reduce` functions to avoid data movement
    - Understanding the `shuffle` operation is important for that
  - The `reduce` function must be commutative and associative
  - Data is split into partitions (degree of parallelism)

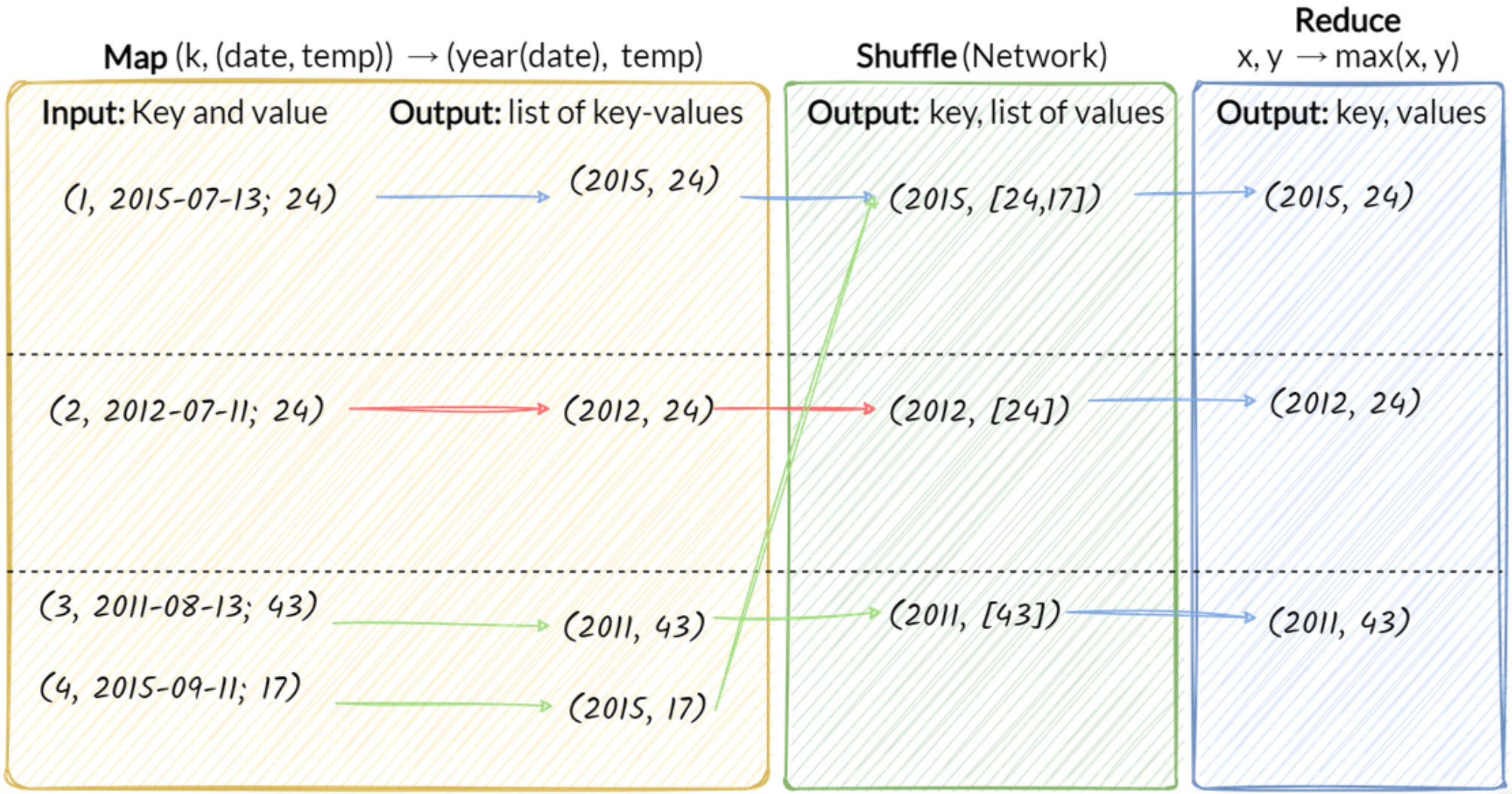
Next Lecture:

- Apache Hadoop

To Learn More

- **Google MapReduce**
  - Google MapReduce paper (Dean and Ghemawat, 2004)
  - MapReduce: a flexible data processing tool (Dean and Ghemawat, 2010)
- **Functional programming**
  - Programming in Haskell book (Hutton, 2016)
  - Functional Programming in Scala book (Chiusano and Bjarnason, 2014)

Challenge: Compute Max Temperature



CAMBRIDGE UNIVERSITY PRESS

Chapter 2

MapReduce II

© Isaac Triguero and Mikel Galar