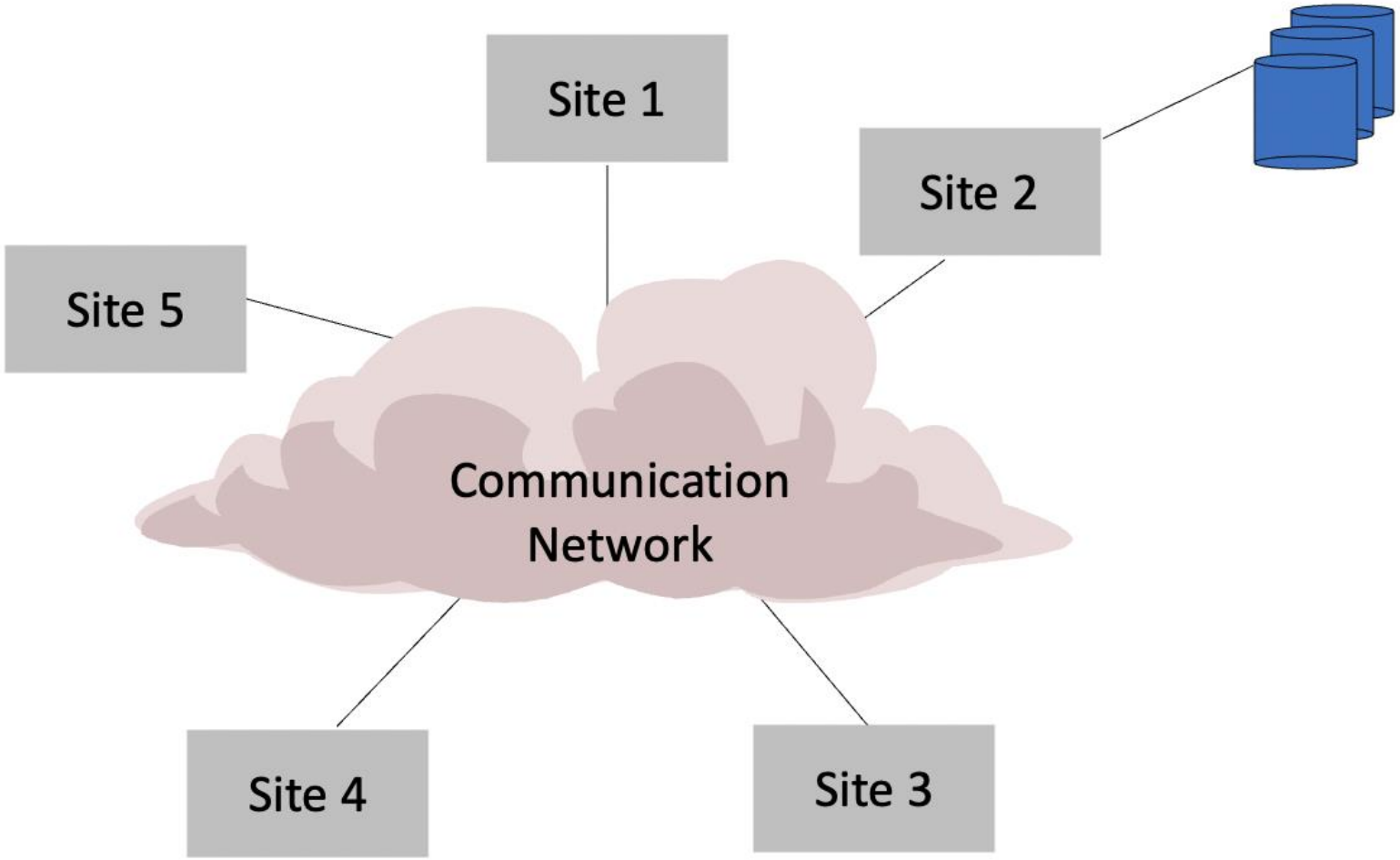
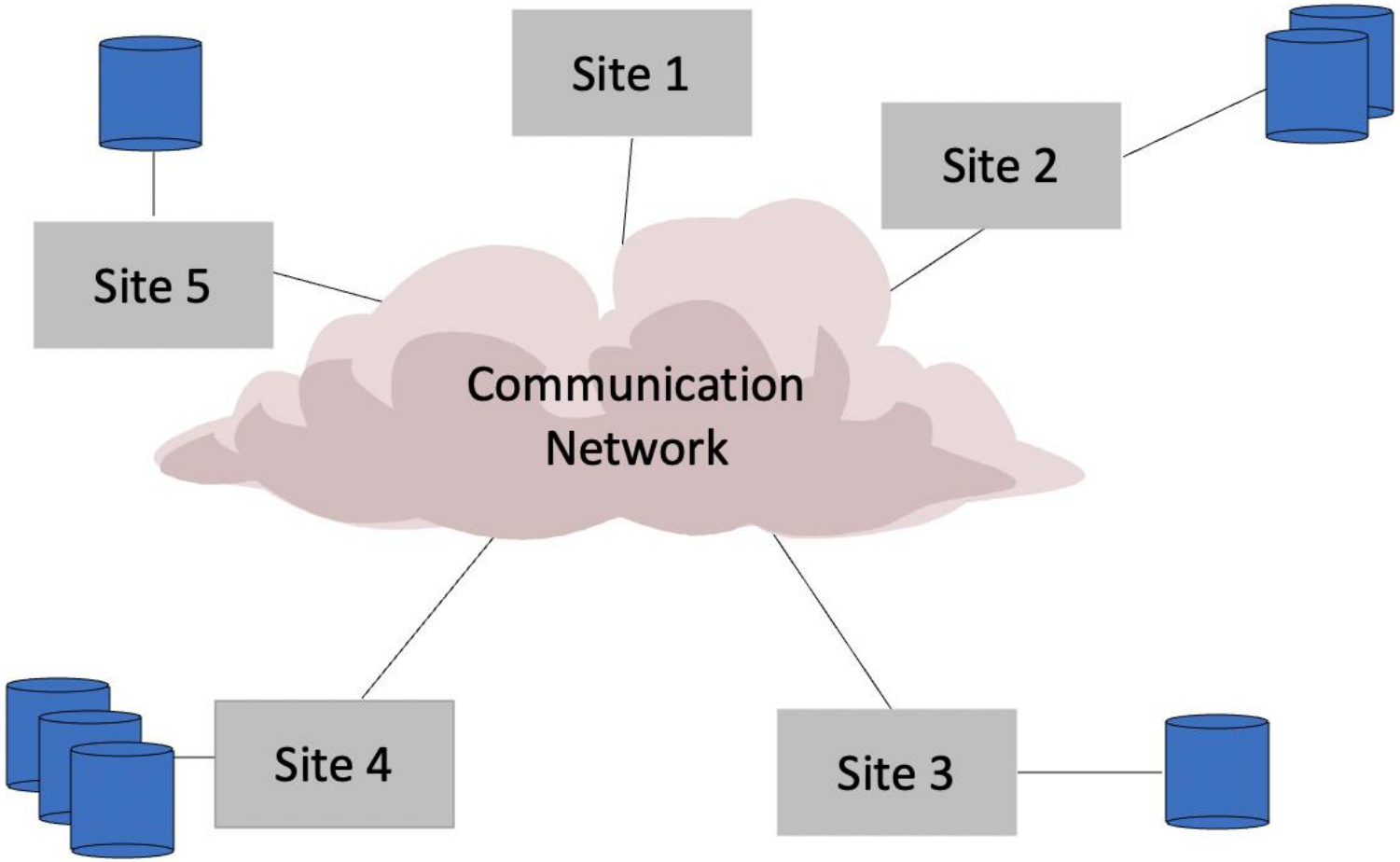
Cloud Data Engineering

Intrudction to MapReduce

Centralized DBMS

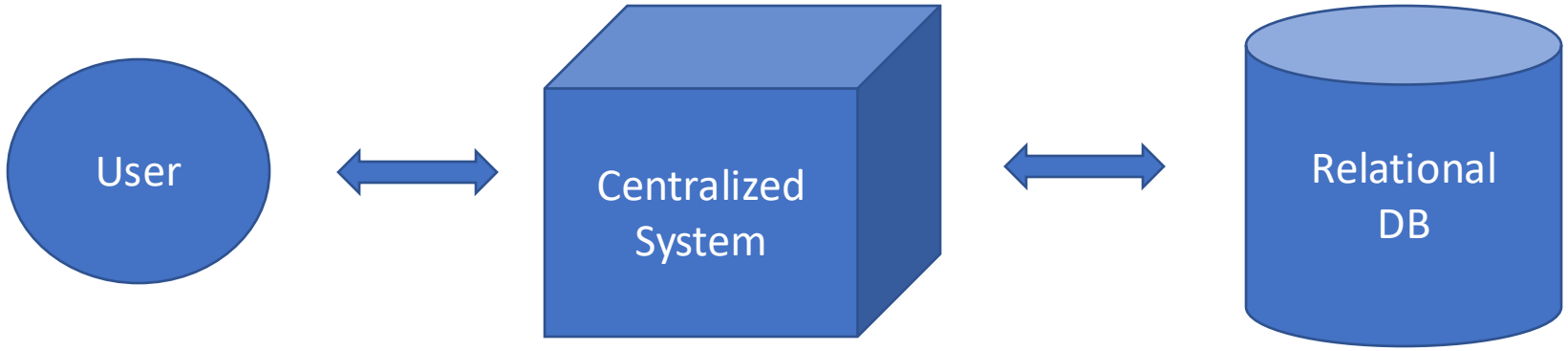


Distributed DBMS



Introduction to Big Data Solutions – Traditional Approach

An enterprise will have a computer to store and process big data. For storage purpose, the programmers will take the help of their choice of database vendors such as Oracle, IBM, etc. In this approach, the user interacts with the application, which in turn handles the part of data storage and analysis.



Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles* (pp. 29-43).

Traditional Approach - What is the limit?

This approach works fine with those applications that process **less voluminous data** that can be accommodated by standard database servers, or up to the limit of the processor that is processing the data.

For example, we have 10TB documents with average size of 20KB (total of 200TB). We want to build a model to count occurrences of words. Simple iterative loop will take ~1 month on simple computer.

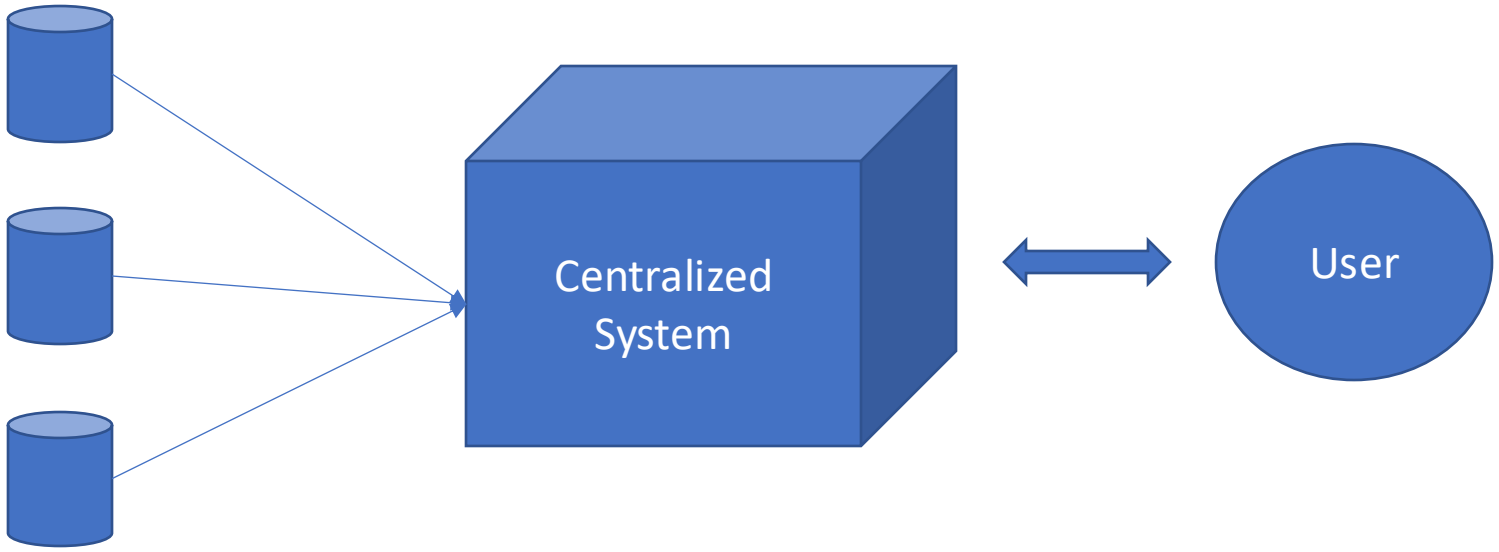
What's next?

Motivation

- Process lots of data
  - Google processed about 24 petabytes of data per day in 2009.
- **A single machine** cannot serve all the data
  - You need a distributed system to store and process **in parallel**
- Parallel programming?
  - **Threading** is hard!
  - How do you facilitate communication between nodes?
  - How do you **scale to more machines**?
  - How do you handle machine failures?

Google's Solution

Google solved this problem using an algorithm called **MapReduce**. This algorithm divides the task into small parts and assigns them to many computers, and collects the results from them which when integrated, form the result dataset.



Google's Solution

Problem: Can't use a single computer to process the data (take too long to process data).

Solution: Use a group of interconnected computers (processor, and memory independent).

Problem: Conventional algorithms are not designed around memory independence.

Solution: MapReduce

Definition. *MapReduce* is a programming paradigm model of using parallel, distributed algorithms to process or generate data sets.

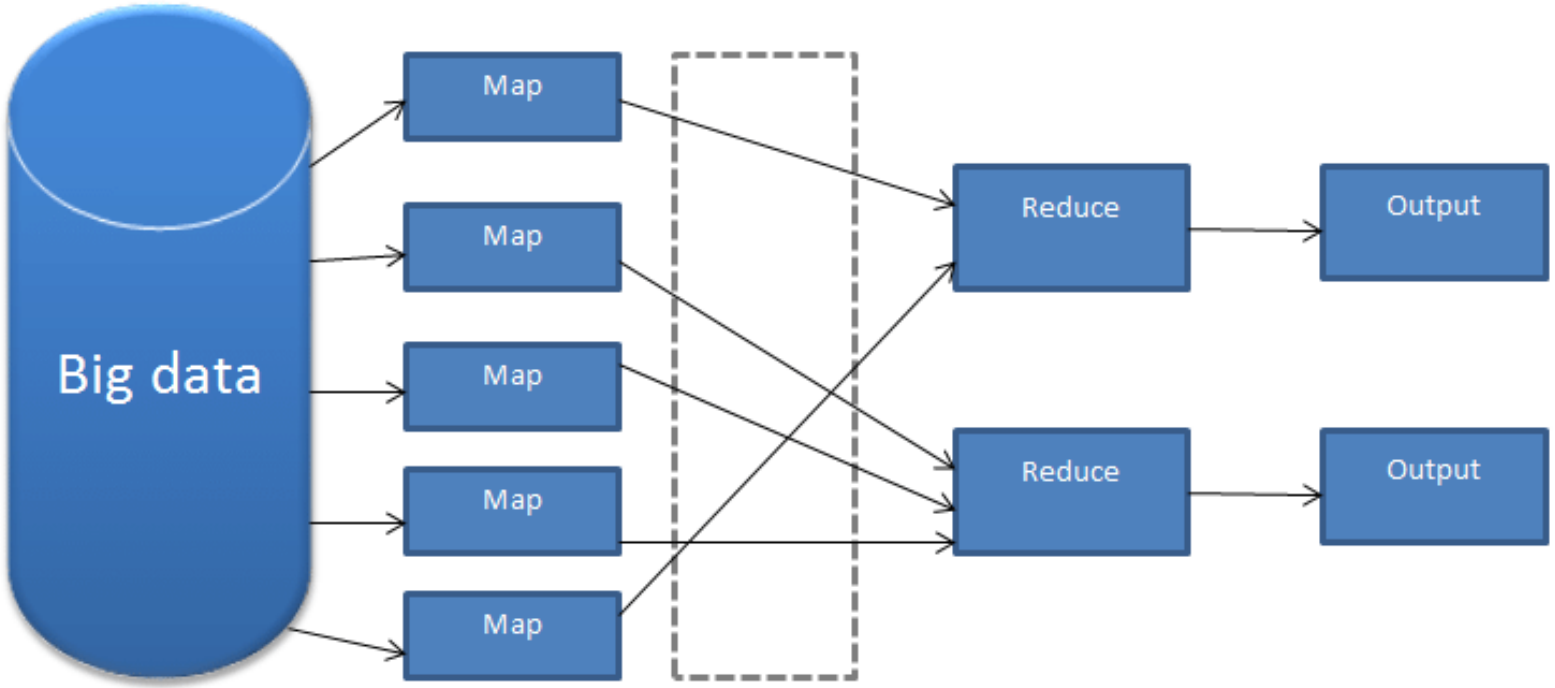
How Is It Work?

The MapReduce algorithm contains two important tasks:

1. The **Map** task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).
2. The **Reduce** task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

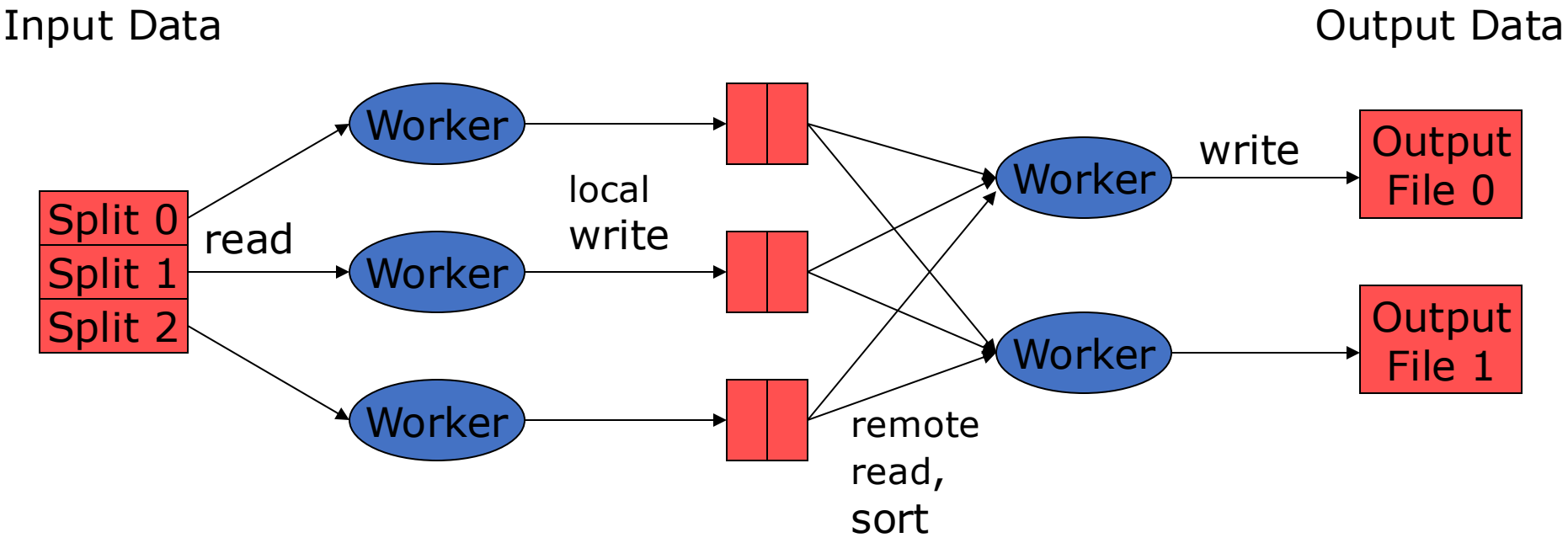
The reduce task is always performed after the map job.

How Is It Work?



The reduce task is always performed after the map job.

MapReduce workflow



Map

extract something you care about from each record

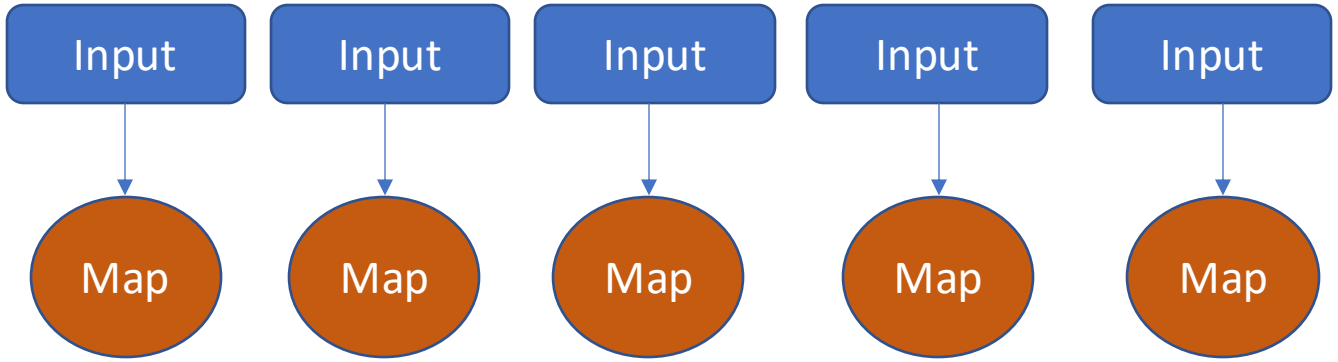
Reduce

aggregate, summarize, filter, or transform

MapReduce Architecture

Input Phase − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.

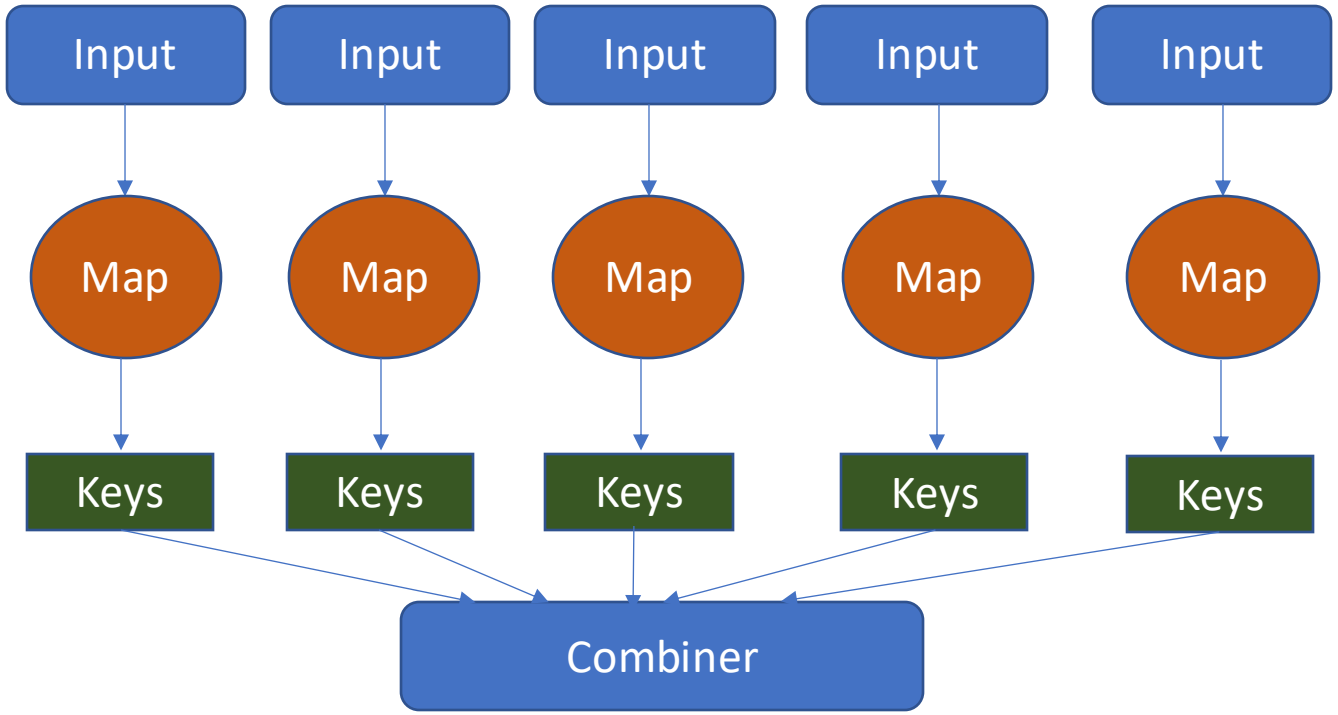
Map − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.



MapReduce Architecture

Intermediate Keys – They key-value pairs generated by the mapper are known as intermediate keys.

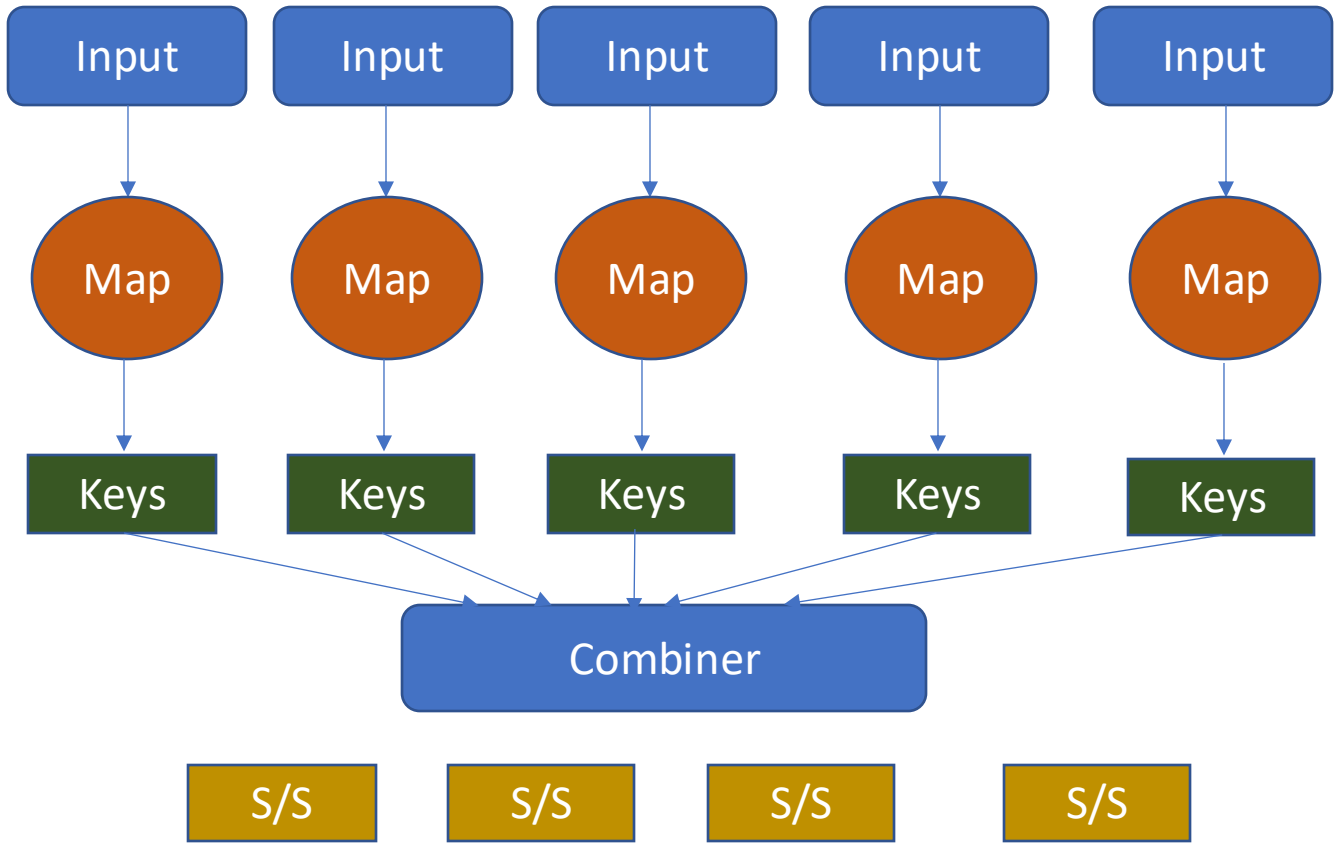
Combiner (Optional) – A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets.



MapReduce Architecture

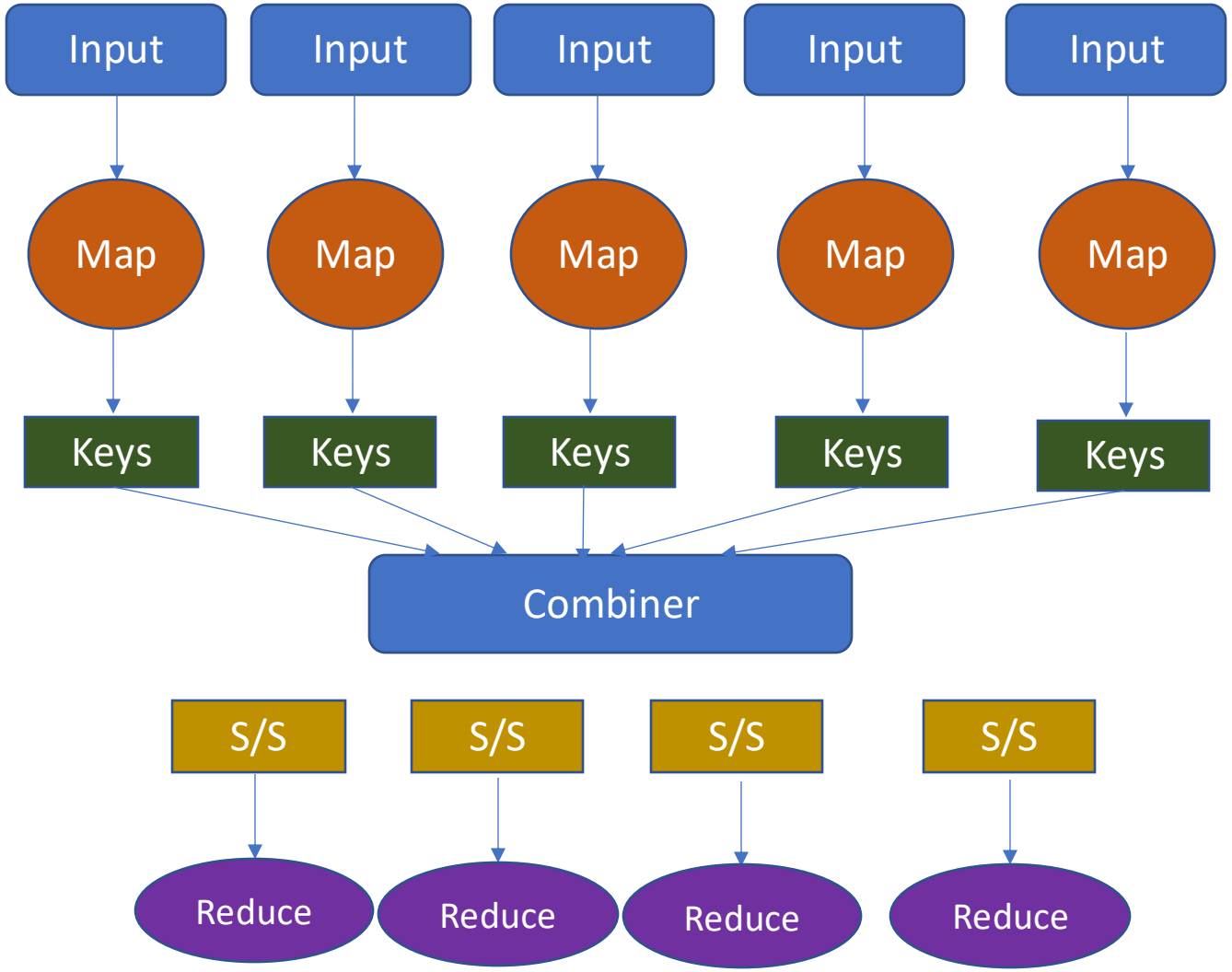
Shuffle and Sort – The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running.

The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.



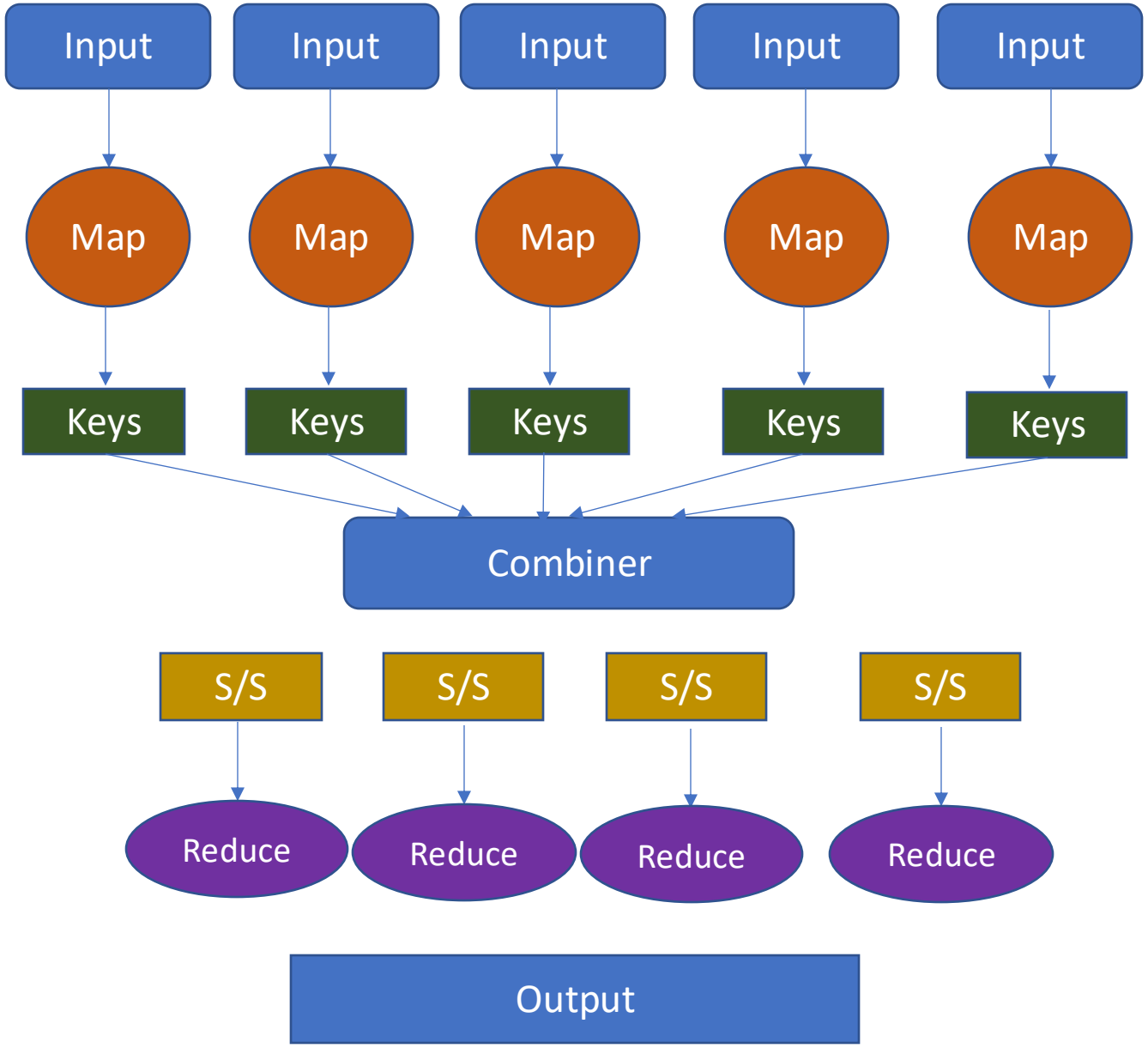
MapReduce Architecture

Reducer – The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.



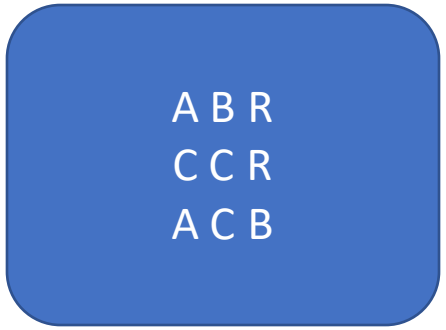
MapReduce Architecture

Output Phase – In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.



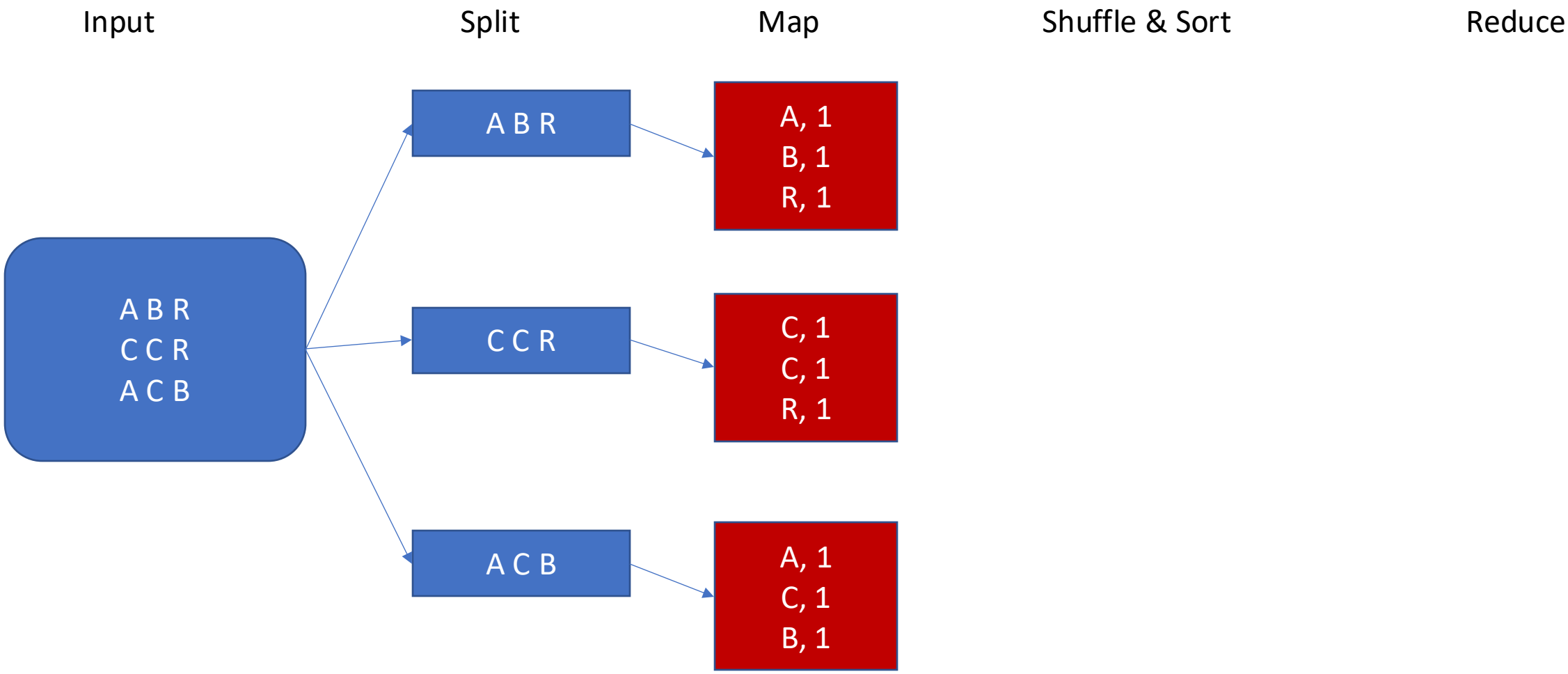
Example

Input | Split | Map | Shuffle & Sort | Reduce

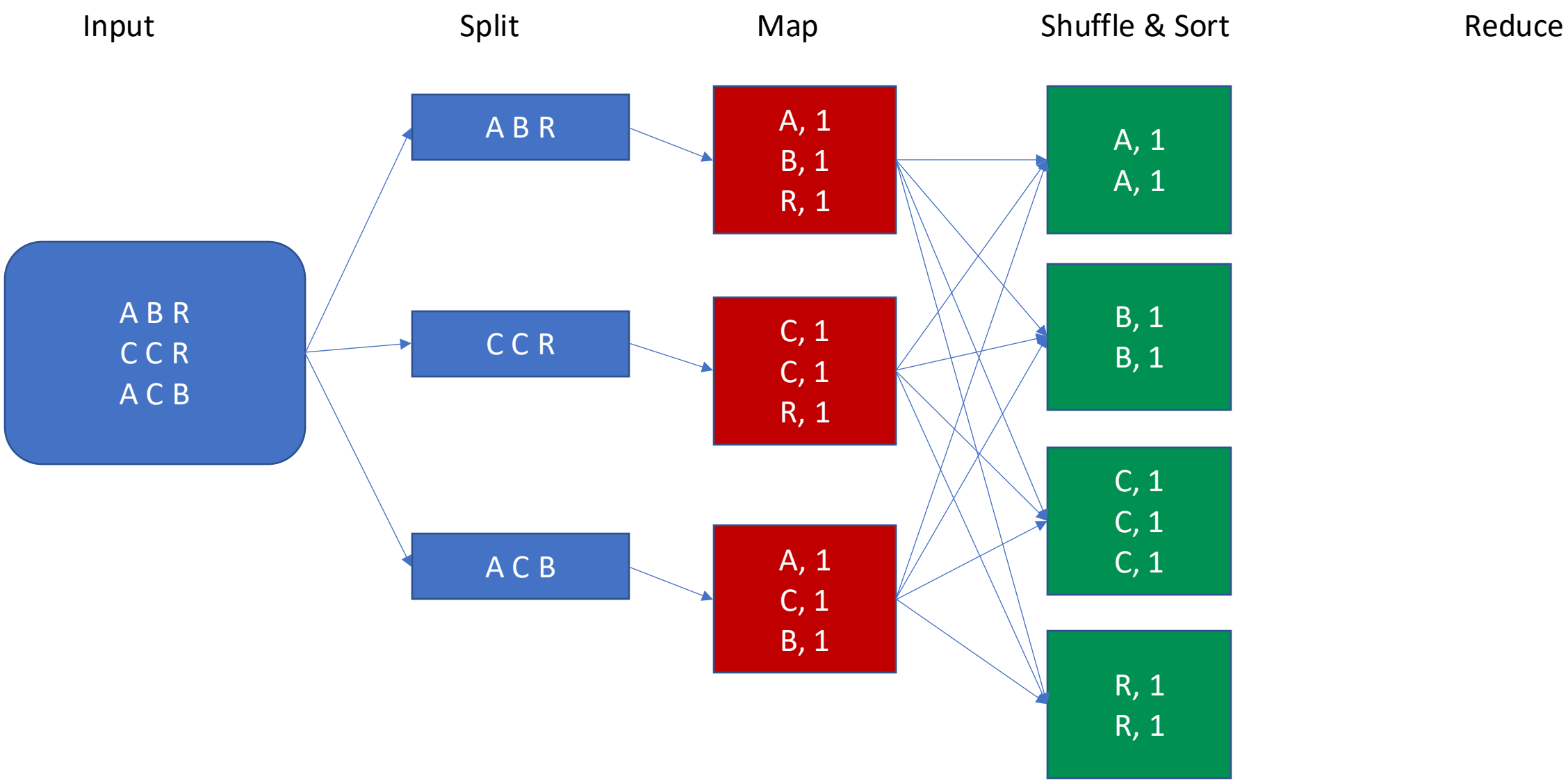


Example

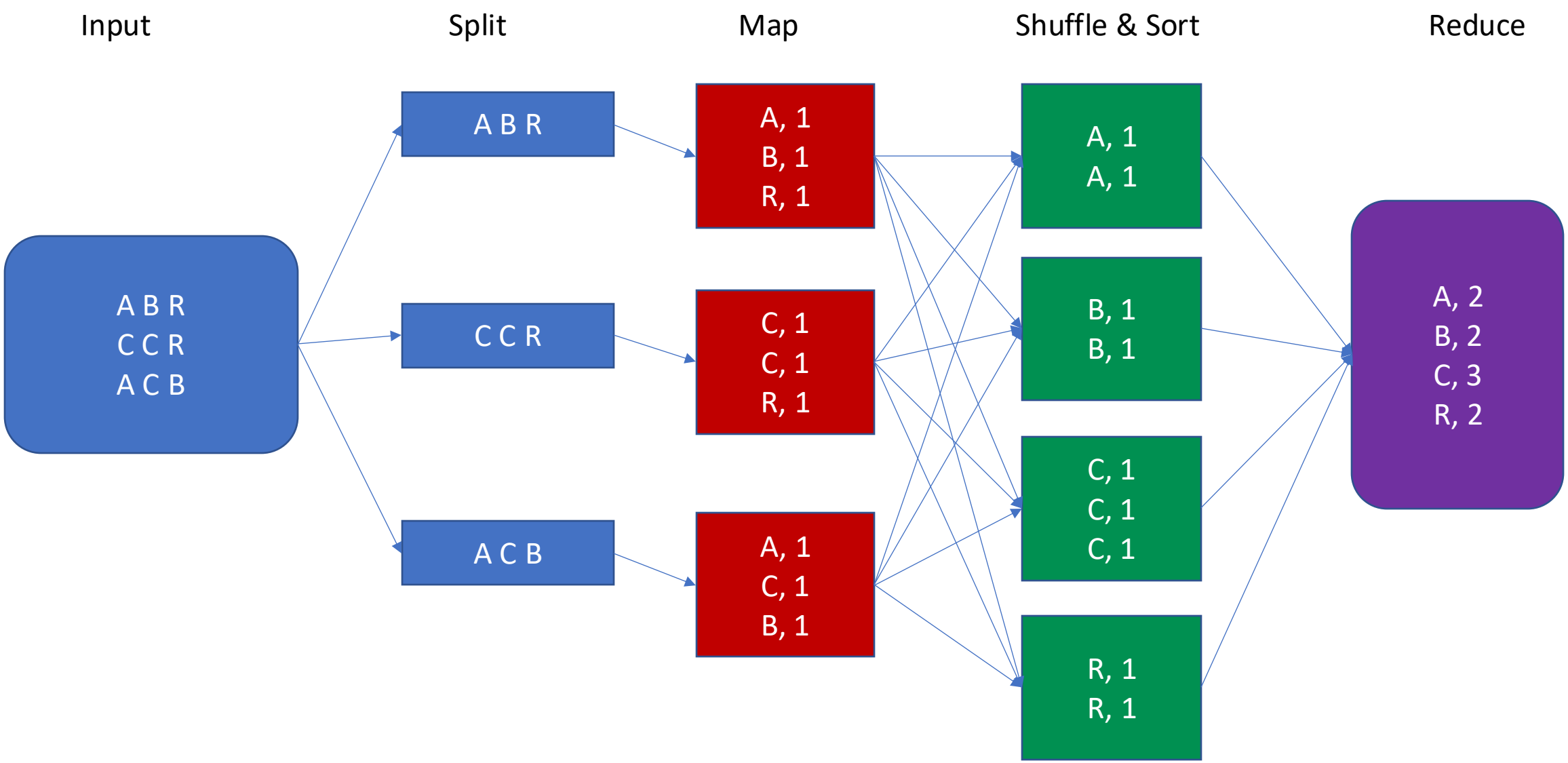
Example


Input
Split
Map
Shuffle & Sort
Reduce
A B R
C C R
A C B
A B R
C C R
A C B
A, 1
B, 1
R, 1
C, 1
C, 1
R, 1
A, 1
C, 1
B, 1

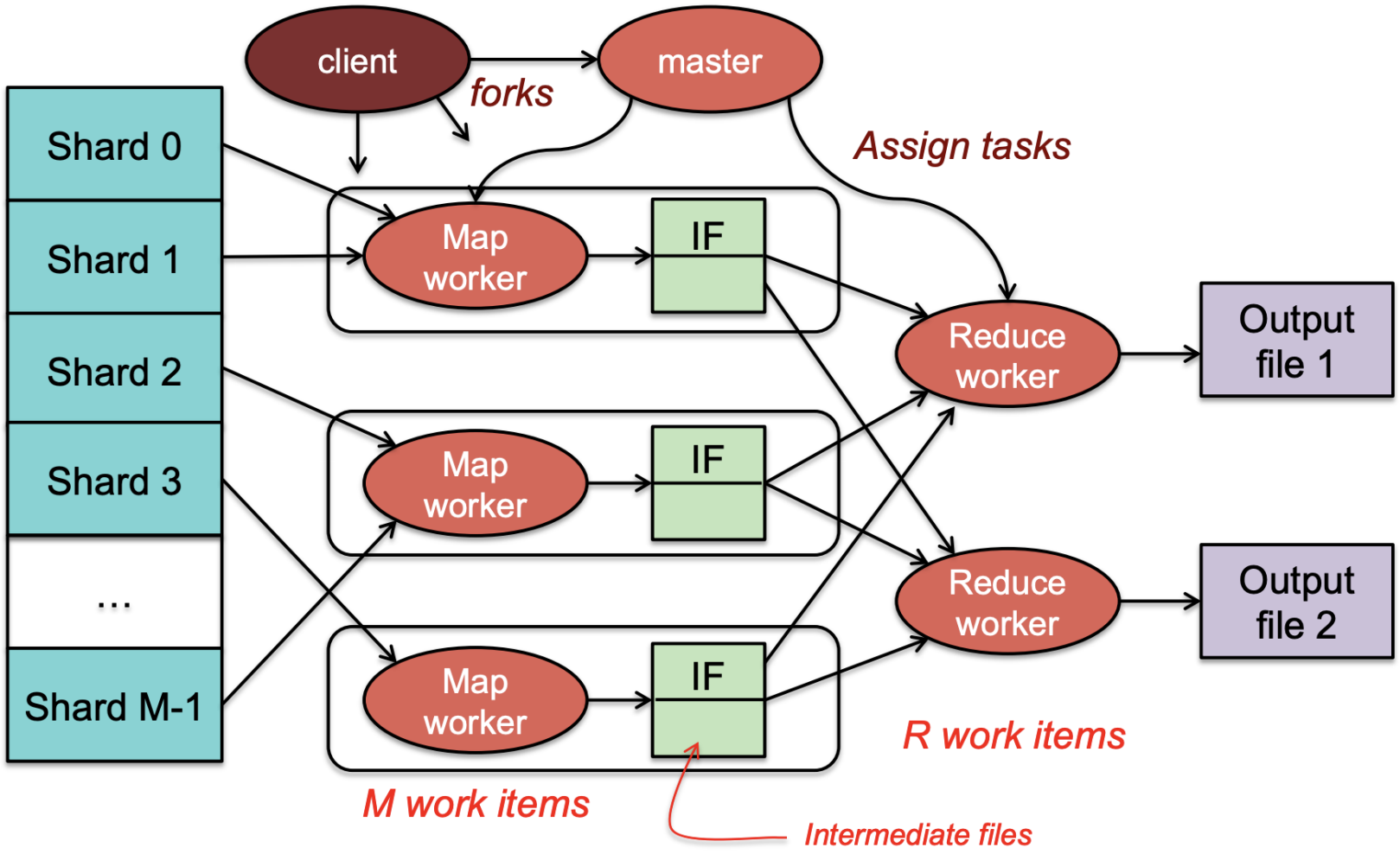
Example


Input
Split
Map
Shuffle & Sort
Reduce
A B R
C C R
A C B
A B R
C C R
A C B
A, 1
B, 1
R, 1
C, 1
C, 1
R, 1
A, 1
C, 1
B, 1
A, 1
A, 1
B, 1
B, 1
C, 1
C, 1
C, 1
R, 1
R, 1

Example


Input
Split
Map
Shuffle & Sort
Reduce
A B R
C C R
A C B
A B R
C C R
A C B
A, 1
B, 1
R, 1
C, 1
C, 1
R, 1
A, 1
C, 1
B, 1
A, 1
A, 1
B, 1
B, 1
C, 1
C, 1
C, 1
R, 1
R, 1
A, 2
B, 2
C, 3
R, 2

The Power of Map-Reduce



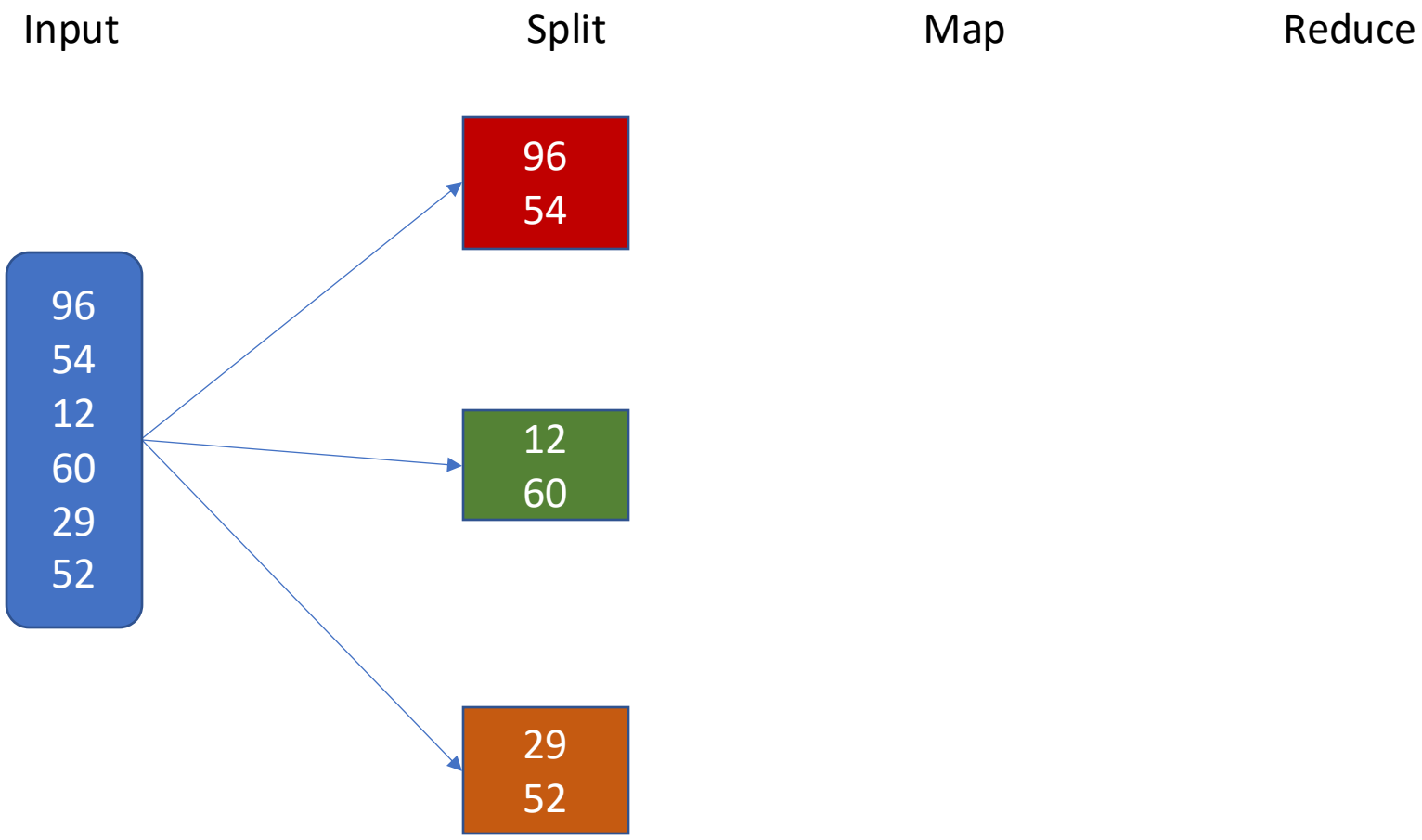
Why is Map-Reduce Popular?

- Distributed computation before MapReduce
  - Divide the workload among multiple machines
  - Distribute data and program to other machines
- Distributed computation after MapReduce
  - Write Map function
  - Write Reduce function
- Developers' tasks made easy
- Divide the problem into sub-problems (divide and conquer paradigm)

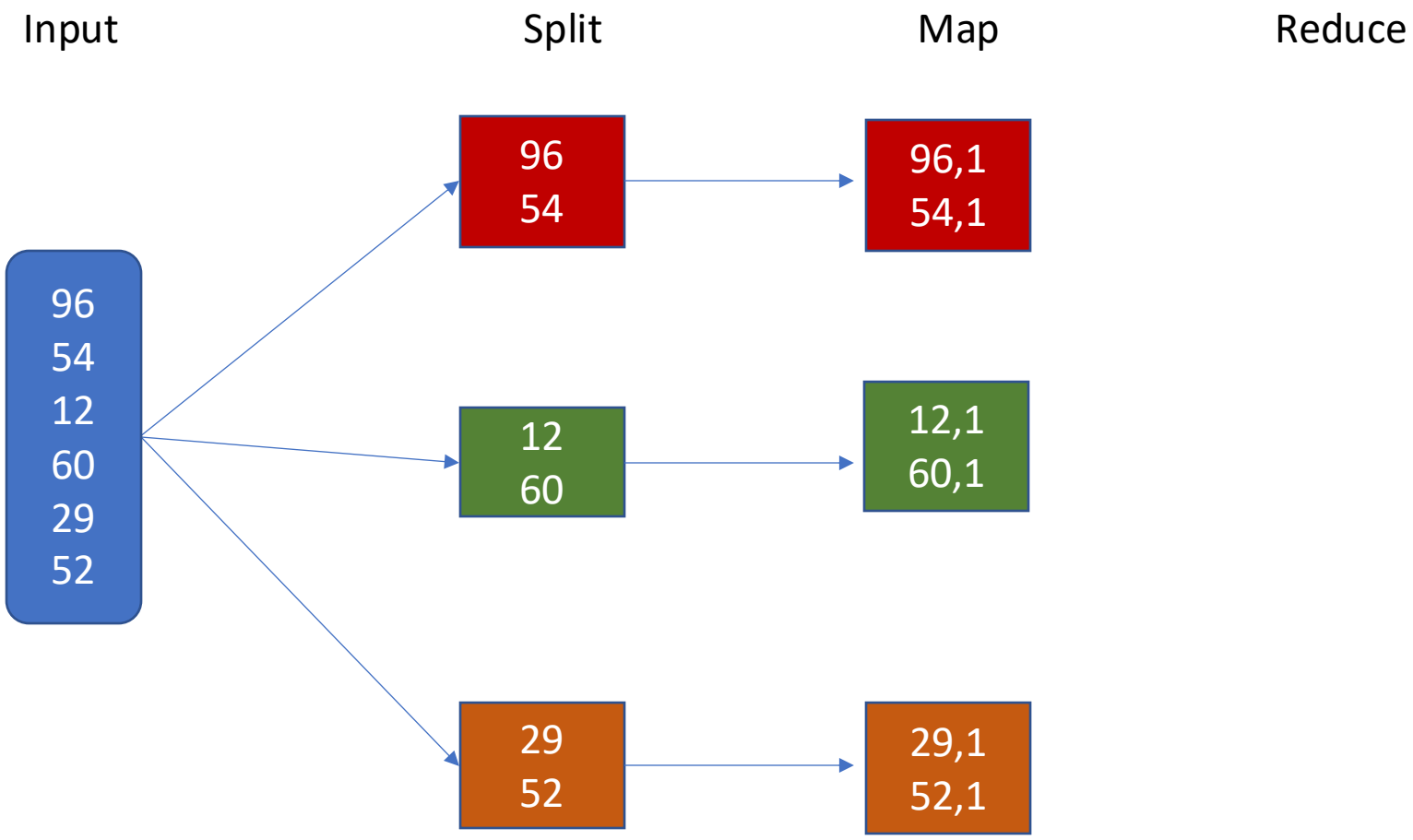
Mean Example

Input | Split | Map | Reduce

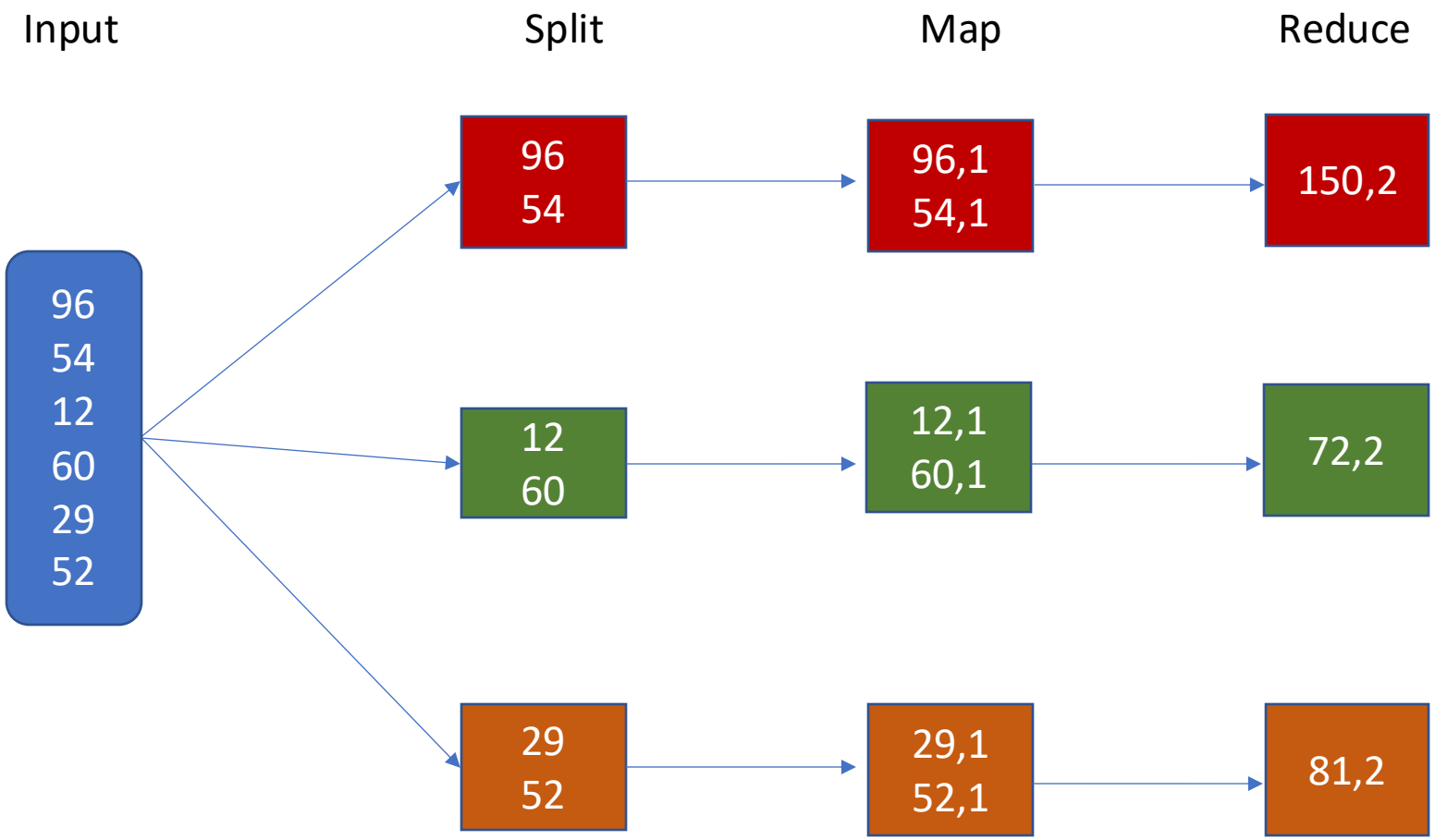
Mean Example



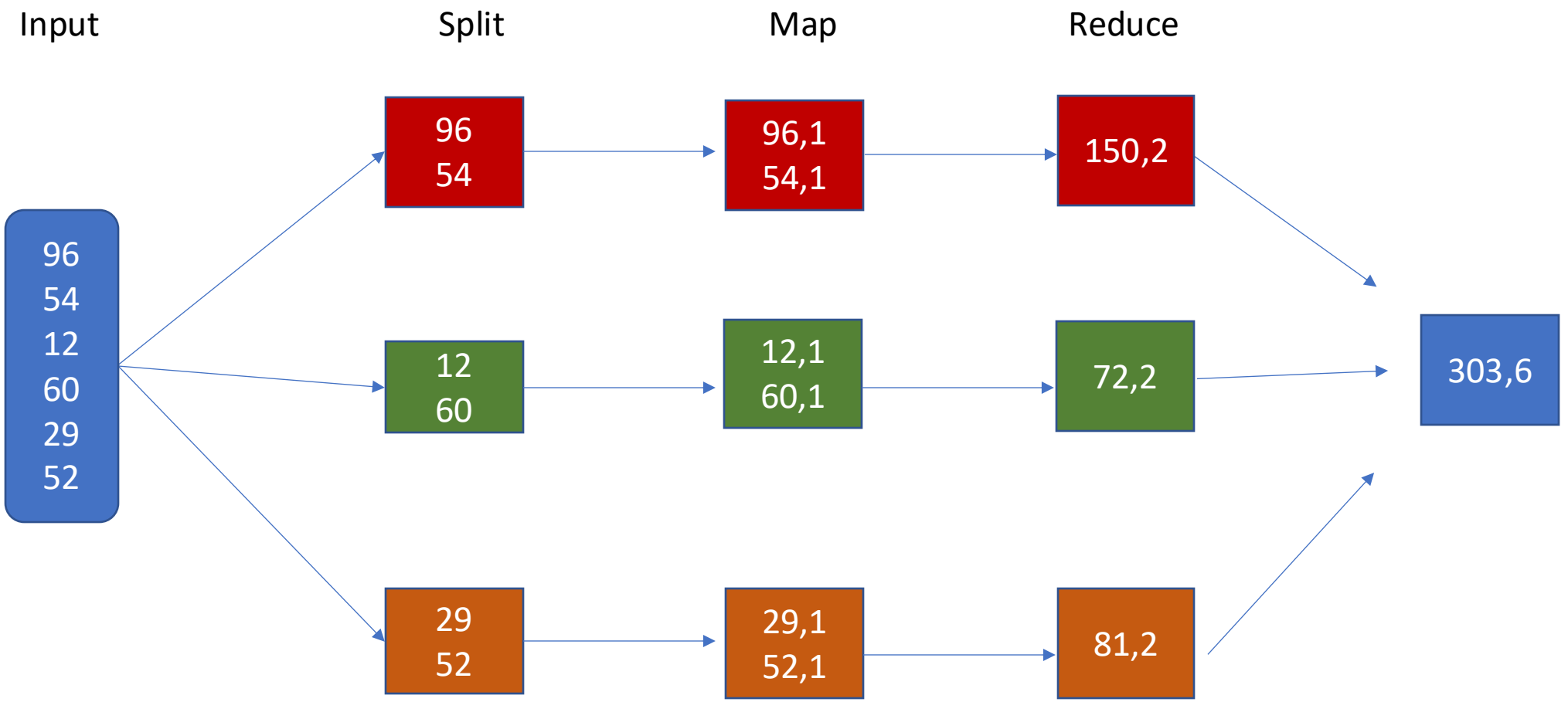
Mean Example



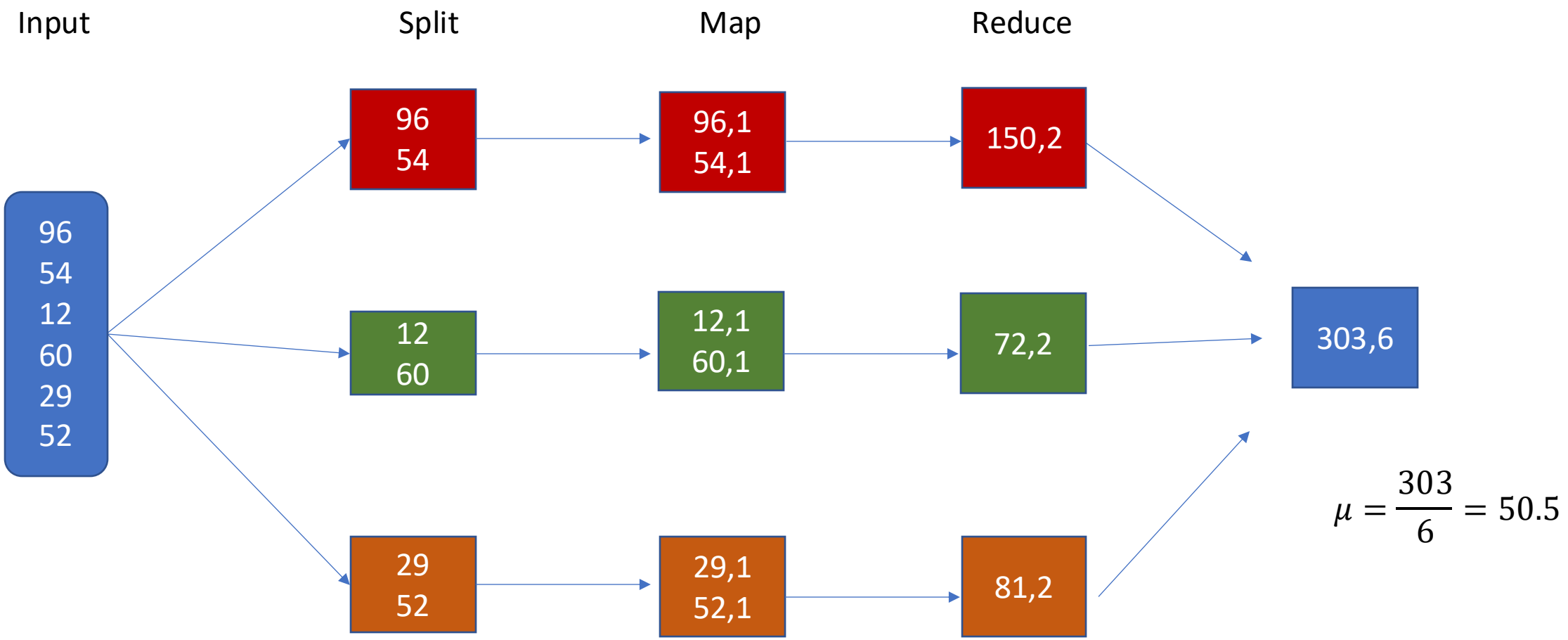
Mean Example



Mean Example



Mean Example



$$\mu = \frac{303}{6} = 50.5$$

Mappers and Reducers

- Need to handle more data? Just add more Mappers/Reducers!
- No need to handle multithreaded code ☺
  - Mappers and Reducers are typically single threaded and deterministic
    - Determinism allows for restarting of failed jobs
  - Mappers/Reducers run entirely independent of each other
    - In Hadoop, they run in separate JVMs