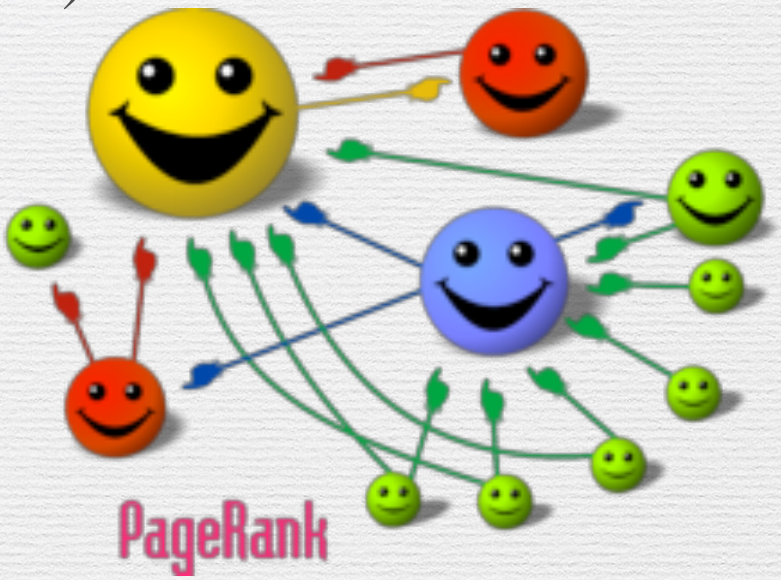
MapReduce

1997

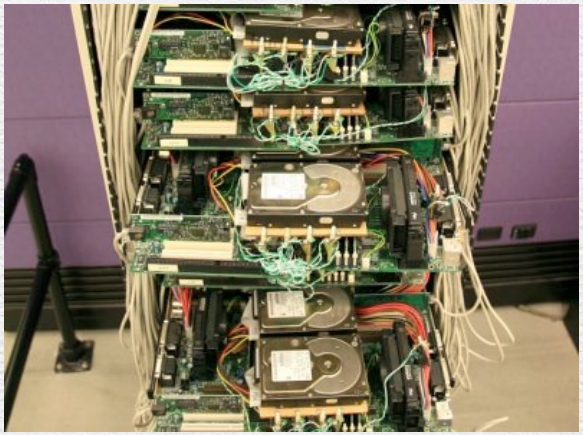
Data	Webpages (URLs)
Problem	Find URLs
Idea: Algorithm	PageRank



Larry Page and Sergey Brin

2003

Data	Webpages (URLs)
Problem	Find Webpage
Idea: Algorithm	PageRank
System	MapReduce



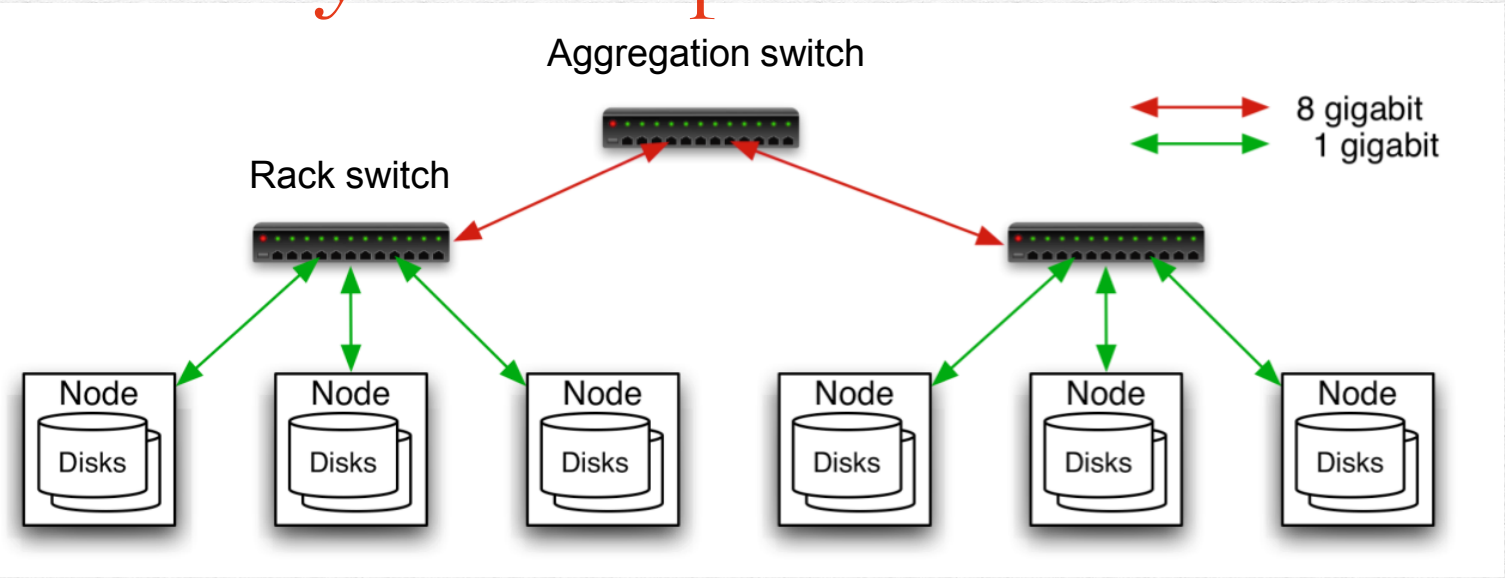
Google

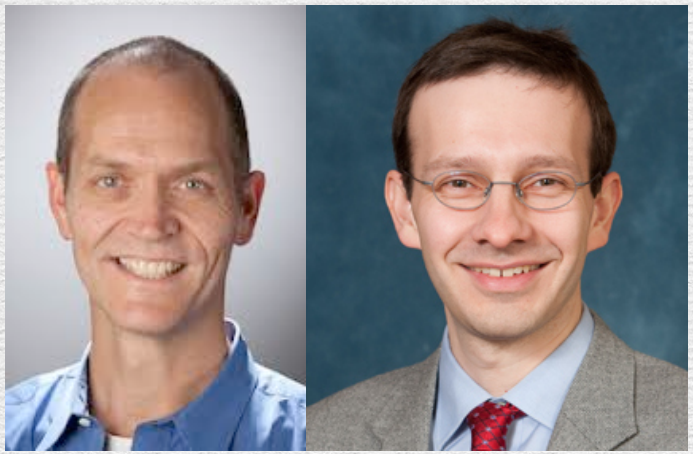
Sanjay Ghemawat and Jeffrey Dean

System

2005

MapReduce System + Open source





Doug Cutting and Mike Cafarella

now used by Yahoo
Facebook Amazon

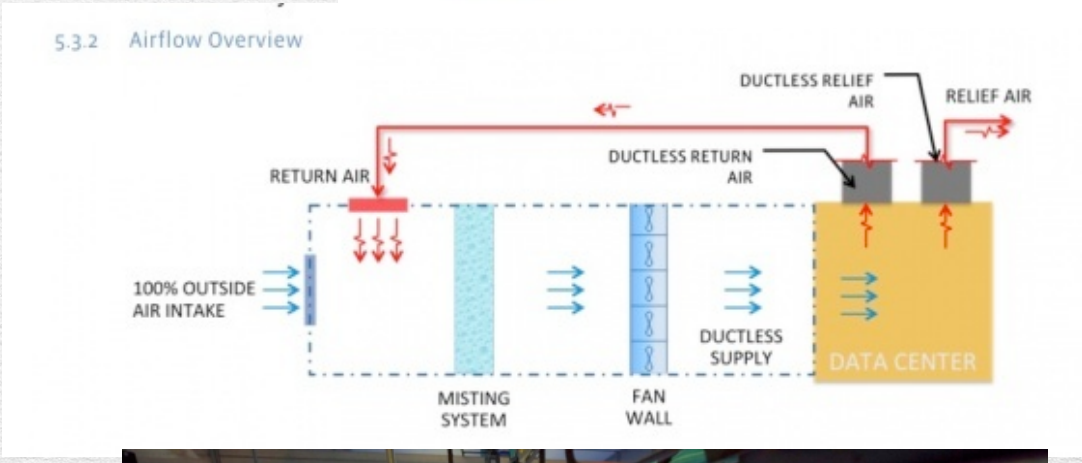
Hardware

2011

Data Center Design + Open source

https://www.youtube.com/watch?v=_r97qdyQtIk

OPEN Compute Project



facebook

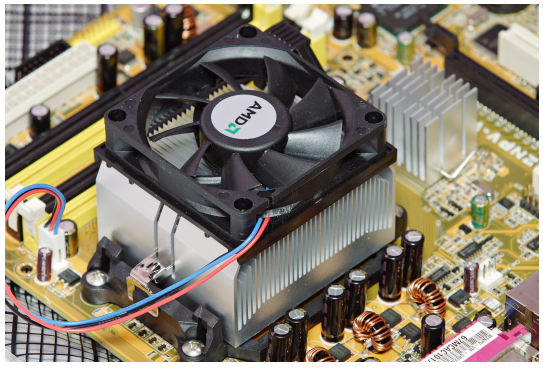
data center cooling system storage power plant

Programming: **MapReduce**, Spark …

Programming: Python, Java, …

laptop

fan



hard drive

power cord

MapReduce

- MapReduce
  - Framework for parallel computing
  - Programmers get simple API
  - Don't have to worry about handling
    - parallelization
    - data distribution
    - load balancing
    - fault tolerance

- Allows one to process huge amounts of data (terabytes and petabytes) on thousands of processors

Mappers

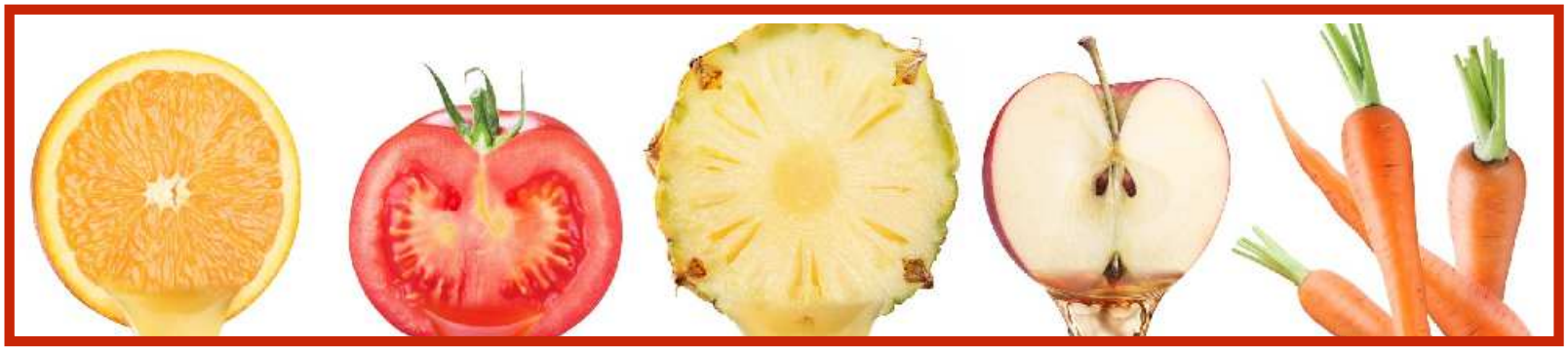
Input:

one data record

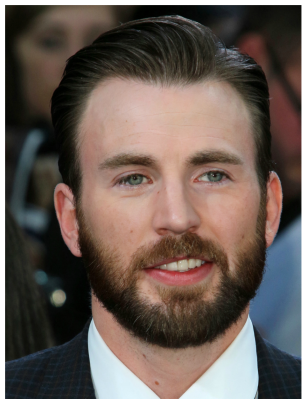
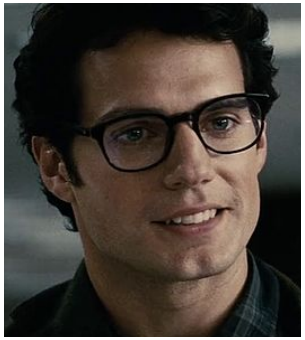
Output:

one/more results

Reducer

Input:	a list of data records 
Output: one/more results	

Mappers

Input: one data record					
Output: one/more results					

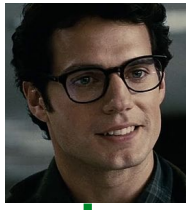
Reducer

Input:

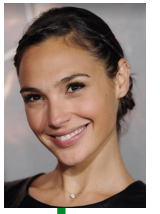
a collection of data records

Output:

one (or multiple) result(s)



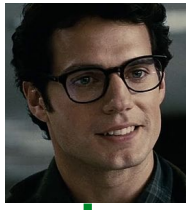

map
map
map
map
map
map



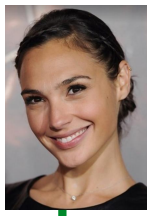





reduce
reduce





map

map

map

map

map

map



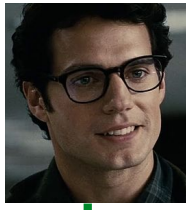



reduce

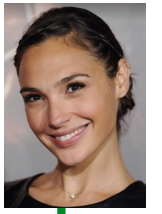
reduce

4

2




map
map
map
map
map
map

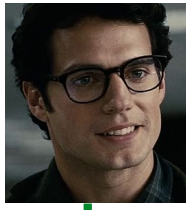




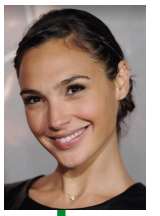


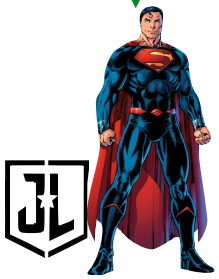

reduce
reduce







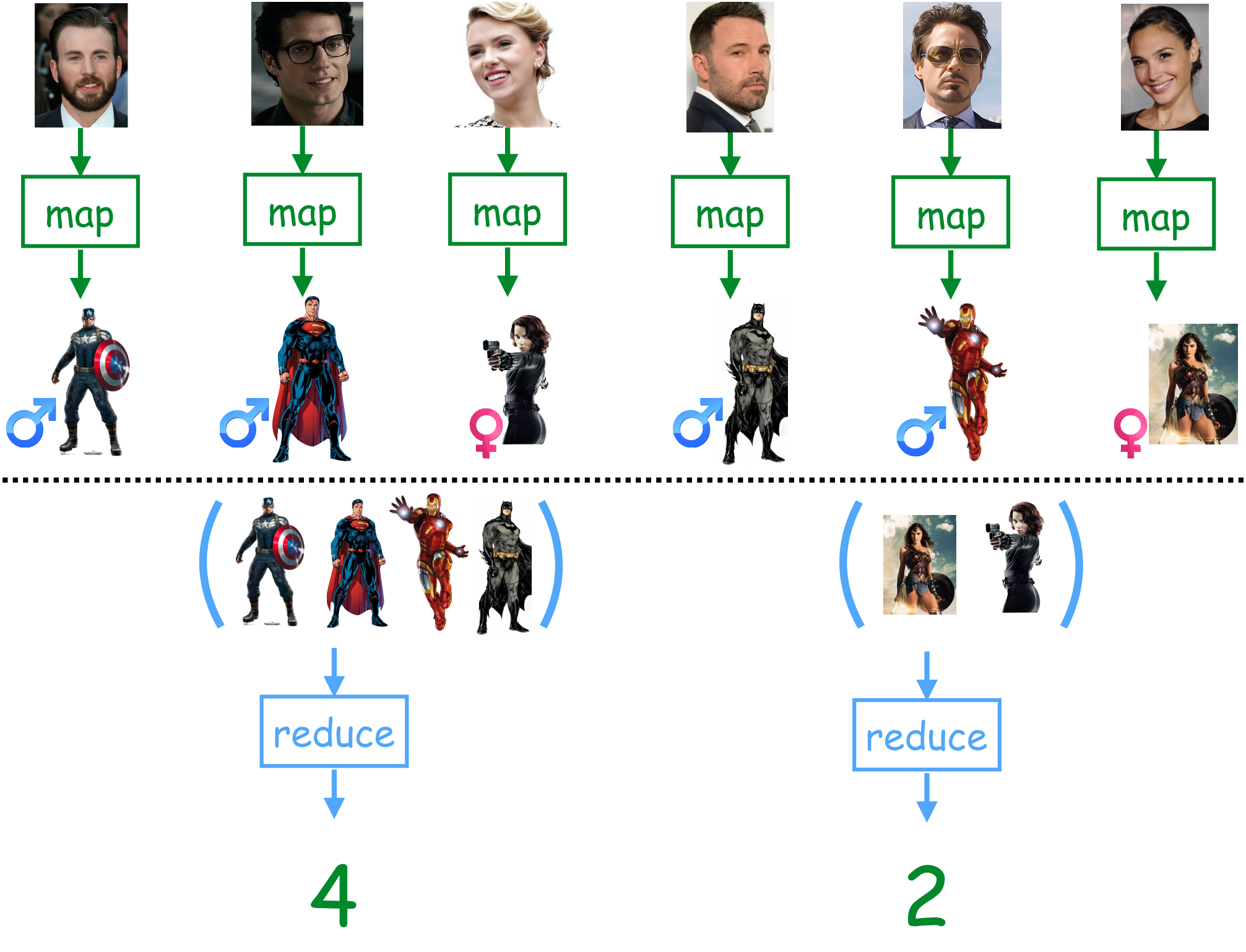


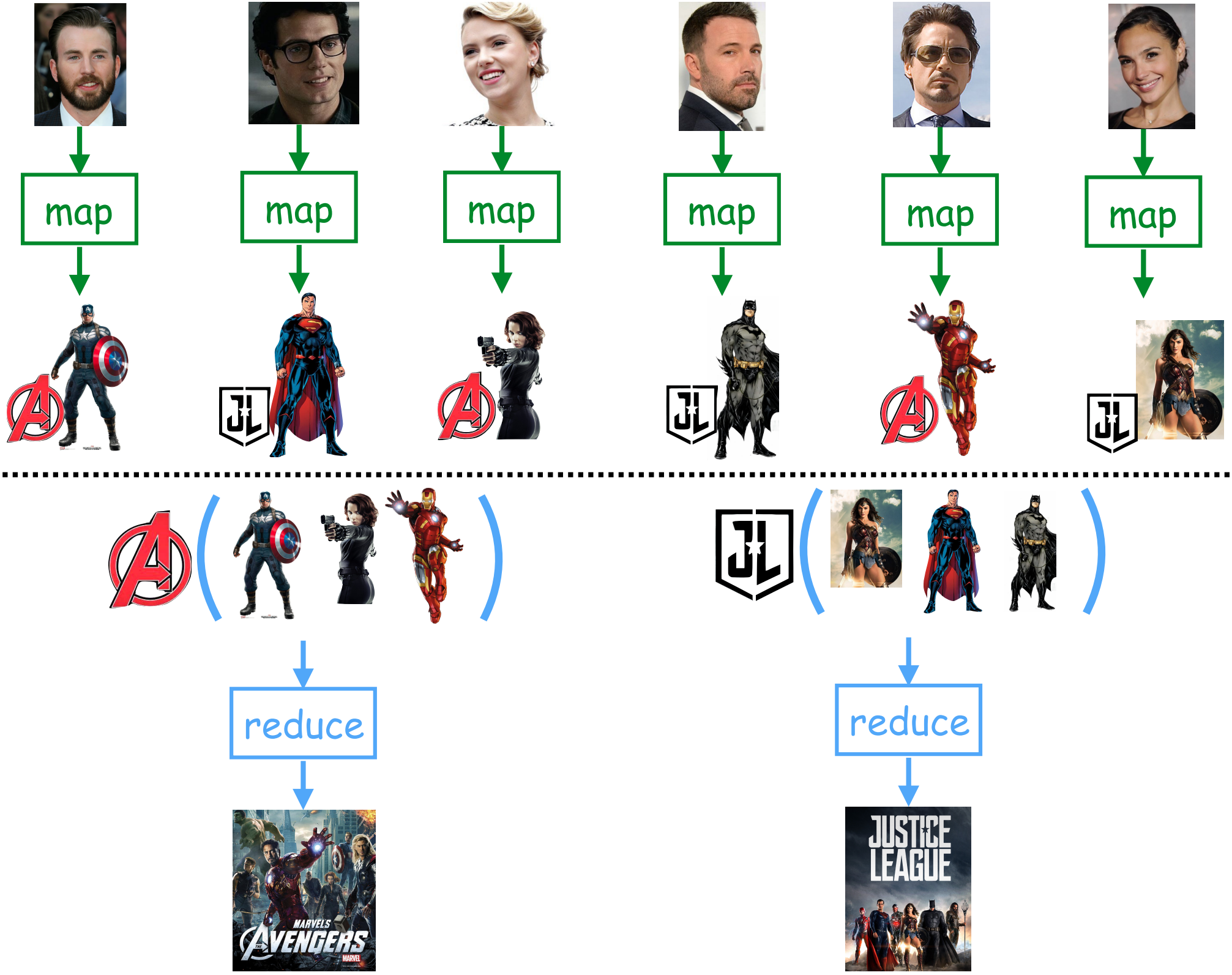


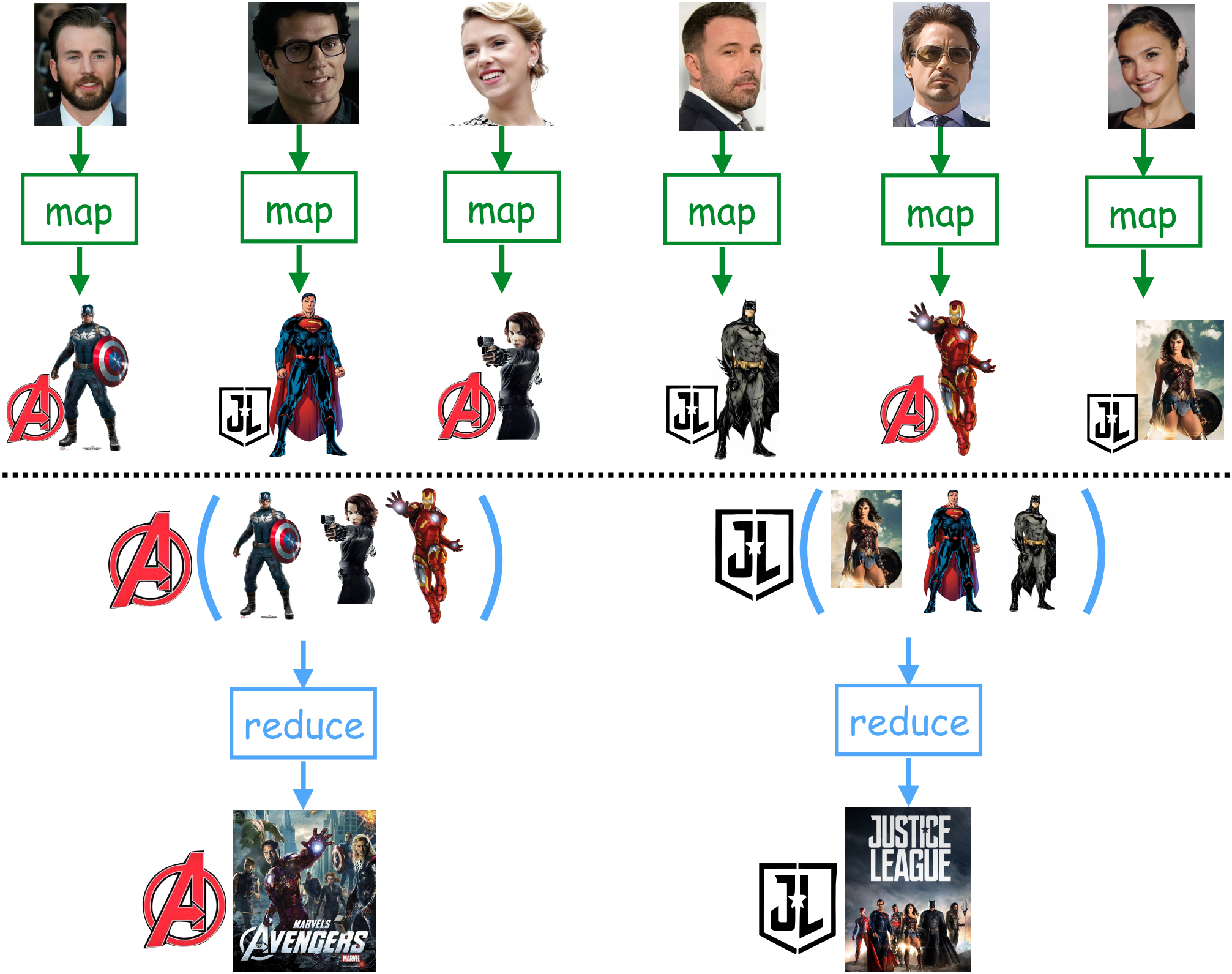


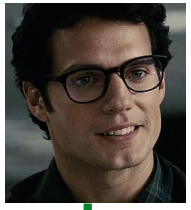



map
map
map
map
map
map
reduce
reduce

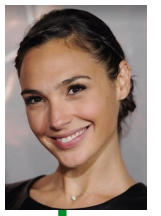

map
map
map
map
map
map
reduce
reduce
4
2


map
map
map
map
map
map
reduce
reduce
MARVELS
AVENGERS
JUSTICE
LEAGUE


map
map
map
map
map
map
reduce
reduce
MARVELS
AVENGERS
JUSTICE
LEAGUE





map map map map map map

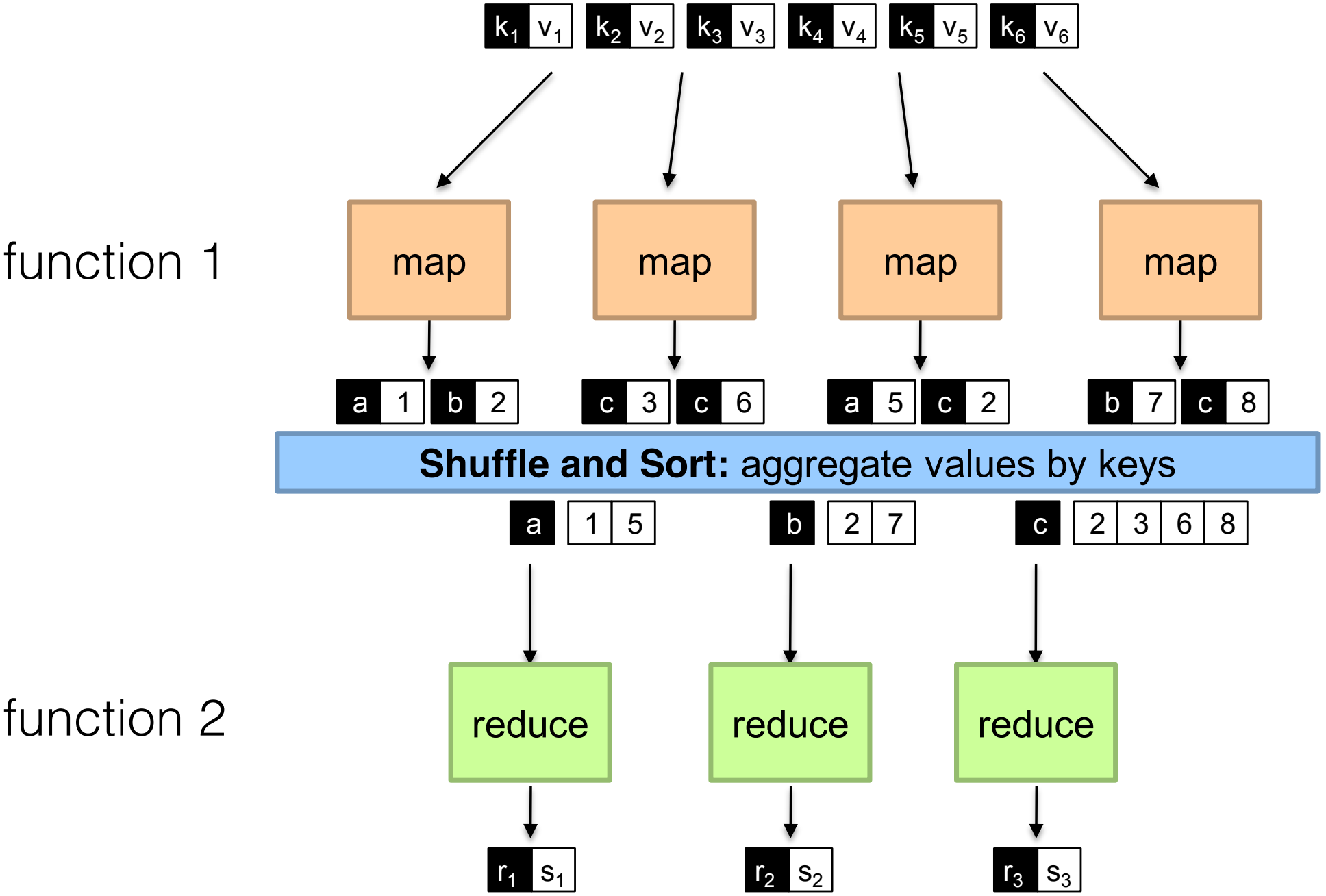






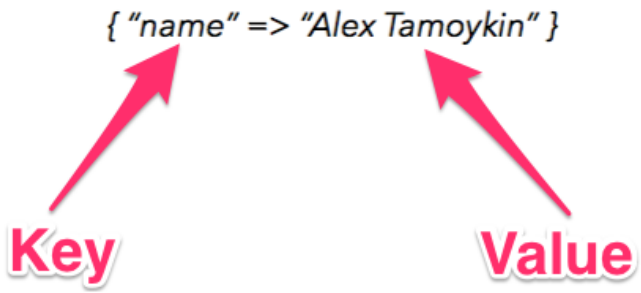
reduce reduce

♂4 ♀2


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
function 1
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
function 2
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Key- Value Pairs

- Mappers and Reducers are users' code (provided functions)
- Just need to obey the Key-Value pairs interface
- **Mappers:**
  - Consume <key, value> pairs
  - Produce <key, value> pairs
- **Reducers:**
  - Consume <key, <list of values>>
  - Produce <key, value>
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a certain reducer in the form of <key, <list of values>>



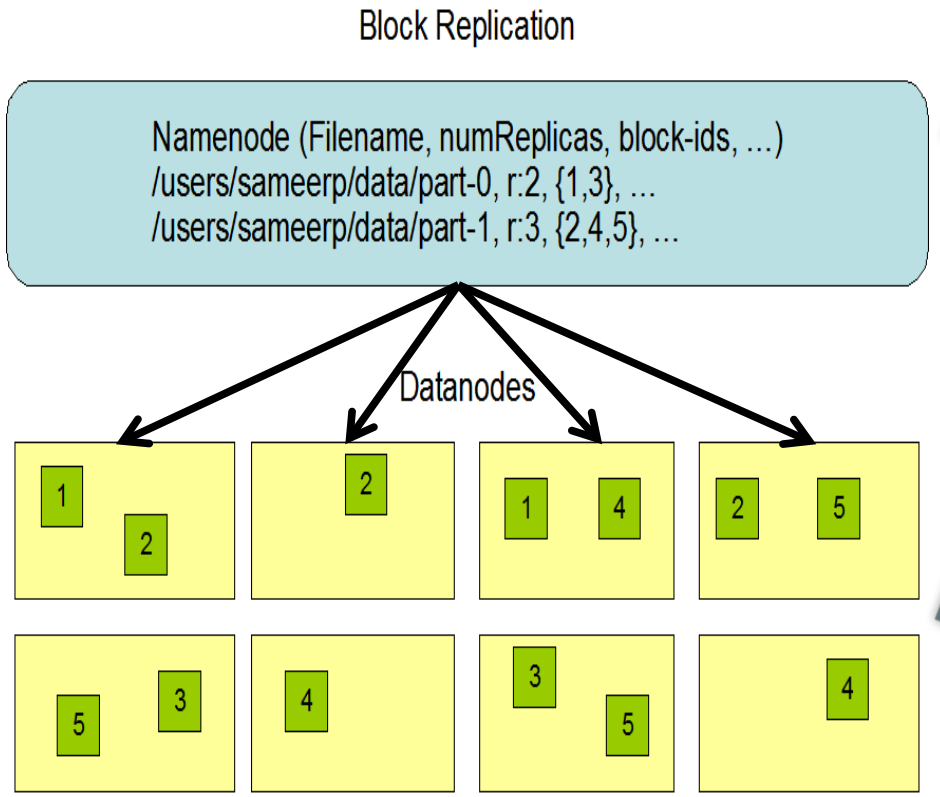
Game

The four French playing card suits used primarily in the English speaking world: spades (♠), hearts (♥), diamonds (♦) and clubs (♣).

Main Properties of HDFS

- ***Large:*** A HDFS instance may consist of thousands of server machines, each storing part of the file system's data
- ***Replication:*** Each data block is replicated many times (default is 3)
- ***Failure:*** Failure is the norm rather than exception
- ***Fault Tolerance:*** Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS
  - Namenode is consistently checking Datanodes

Hadoop Distributed File System (HDFS)



Centralized namenode

- Maintains metadata info about files

File *F*

Blocks (64 MB)

Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

MRjob package

pythonhosted.org/mrjob/guides/quickstart.html

mrjob v0.4.2 documentation

Home » Guides

← Why mrjob? | Concepts →

Table Of Contents

Need help?

Join the mailing list by visiting the Google group page or sending an email to mrjob+subscribe@googlegroups.com.

type to search

Fundamentals

Installation

Install with `pip`:

```
pip install mrjob
```

or from a git clone of the source code:

```
python setup.py test && python setup.py install
```

Writing your first job

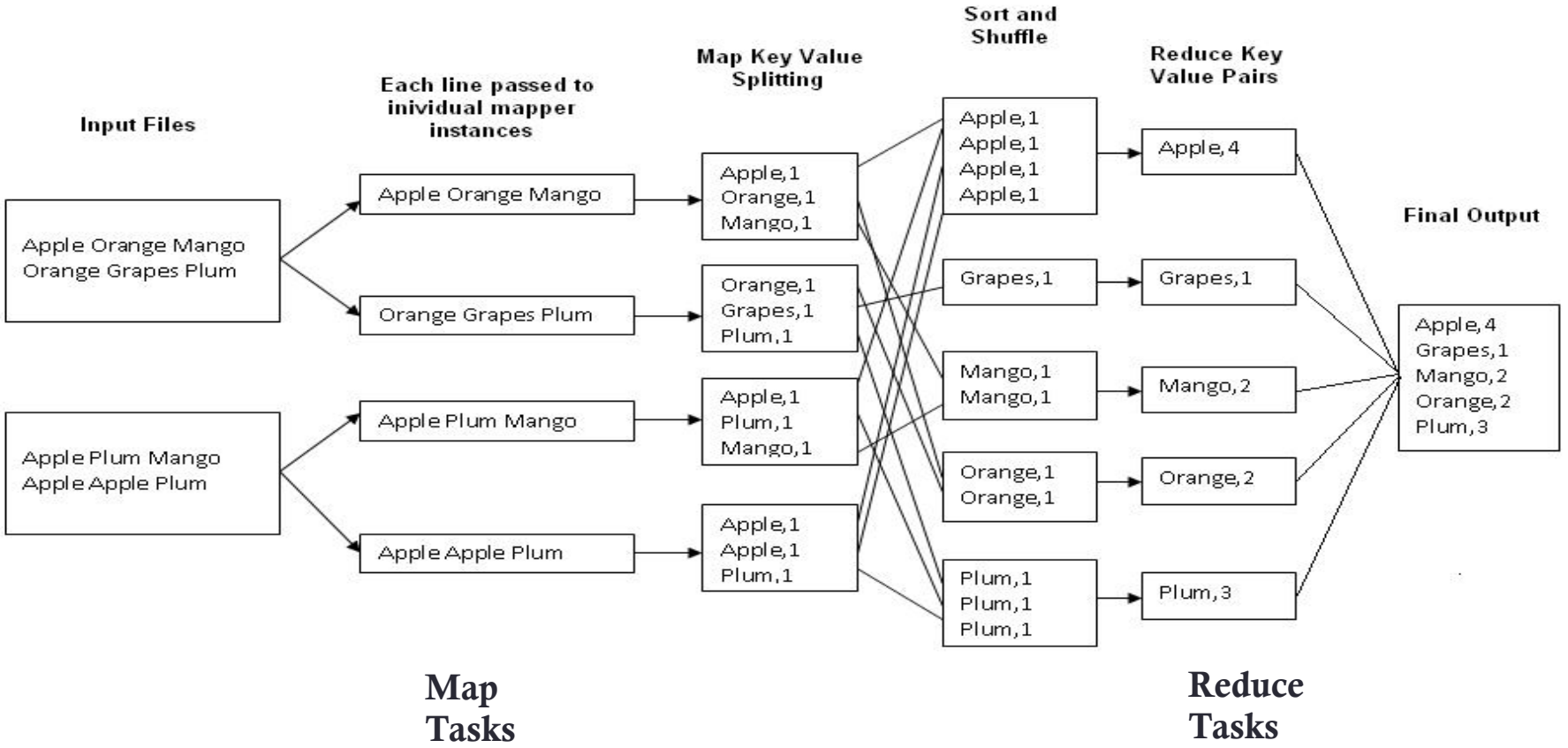
Open a file called `word_count.py` and type this into it:

```
from mrjob.job import MRJob


class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)
```

http://pythonhosted.org/mrjob/guides/quickstart.html

Example 1: Word Count

- **Job: Count the occurrences of each word in a data set**



The Famous Word Count Example

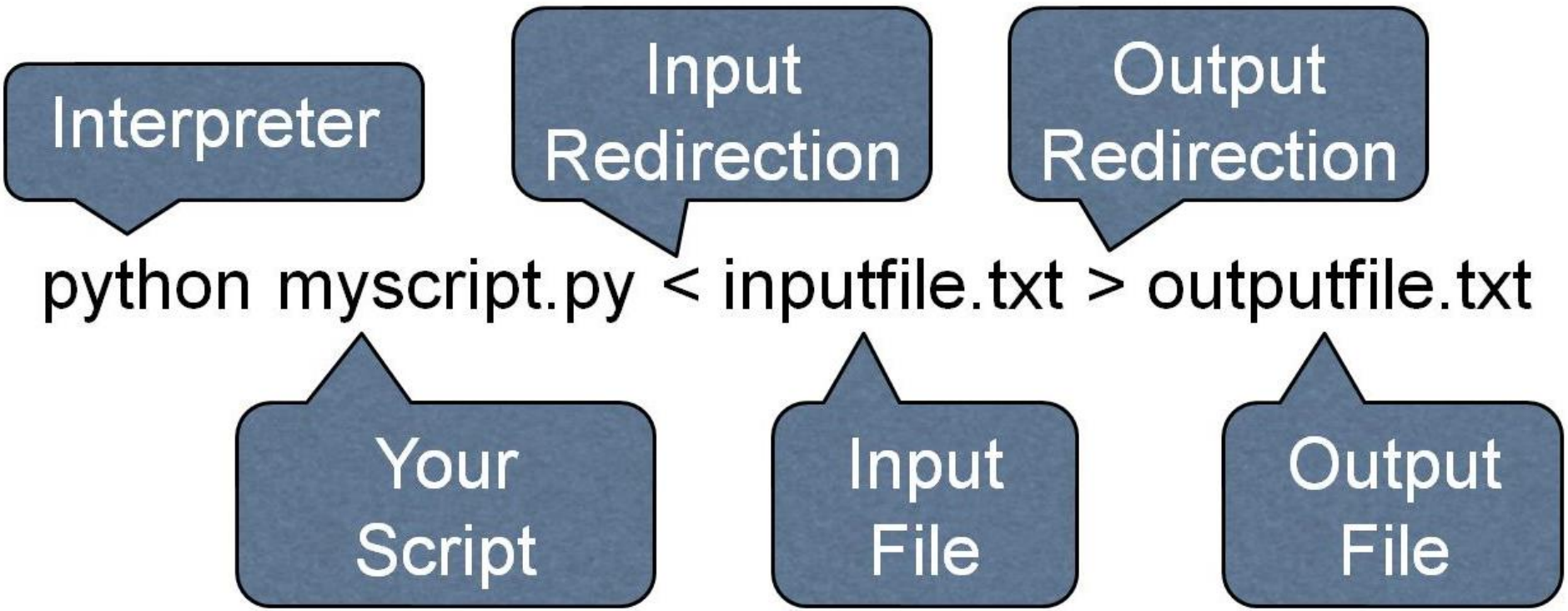
```
from mrjob.job import MRJob

class mrWordCount(MRJob):

    def mapper(self,key,line):
        for word in line.split(' '):
            yield word.lower(),1

    def reducer(self, word, occurrences):
        yield word, sum(occurrences)

if __name__ == '__main__':
    mrWordCount.run()
```

Example Input Data

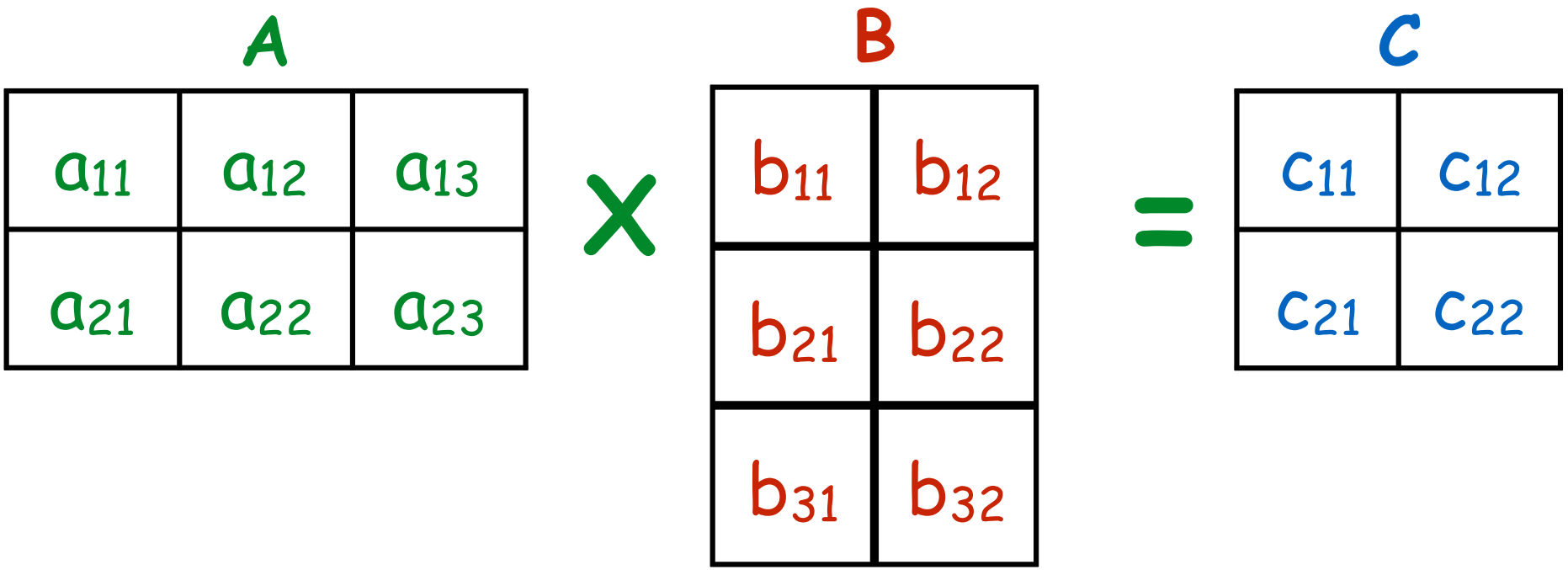
Hamlet

Shakespeare

Launching the Job

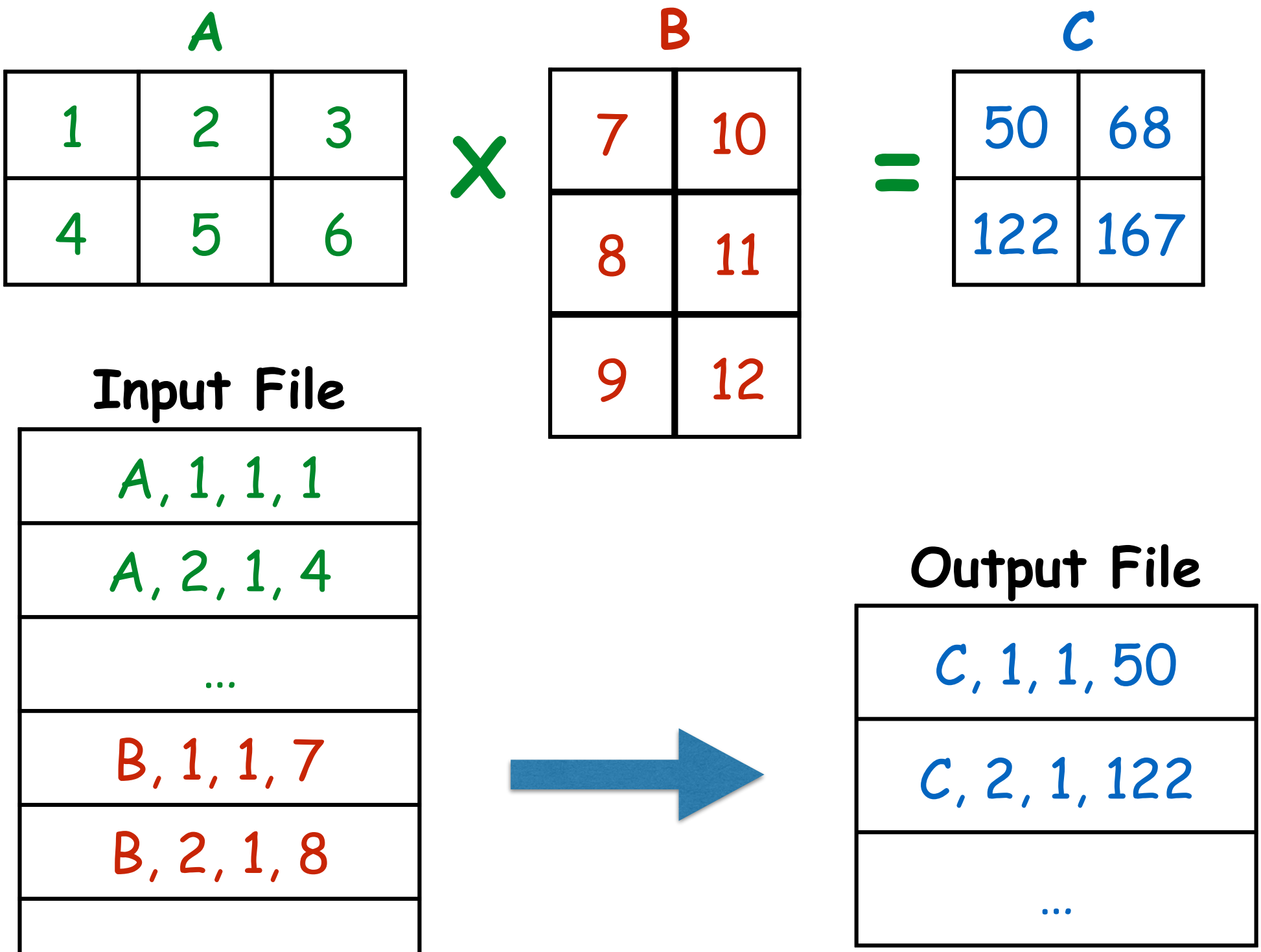


Matrix Multiplication



$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31}$$

$$c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$$


A
1 2 3
4 5 6
X
B
7 10
8 11
9 12
=
C
50 68
122 167
Input File
A, 1, 1, 1
A, 2, 1, 4
...
B, 1, 1, 7
B, 2, 1, 8
...
Output File
C, 1, 1, 50
C, 2, 1, 122
...

Mapper

Input:

one element of an input matrix

A, 1, 1, 1	A, 2, 1, 4	B, 1, 1, 7	B, 2, 2, 11
A[1,1]	A[2,1]	B[1,1]	B[2,2]
1 2 3 4 5 6	1 2 3 4 5 6	7 10 8 11 9 12	7 10 8 11 9 12

Output:

(C[1,1], A[1,1])	(C[2,1], A[2,1])	(C[1,1], B[1,1])	(C[1,2], B[2,2])
(C[1,2], A[1,1])	(C[2,2], A[2,1])	(C[2,1], B[1,1])	(C[2,2], B[2,2])
key, value	key, value	key, value	key, value

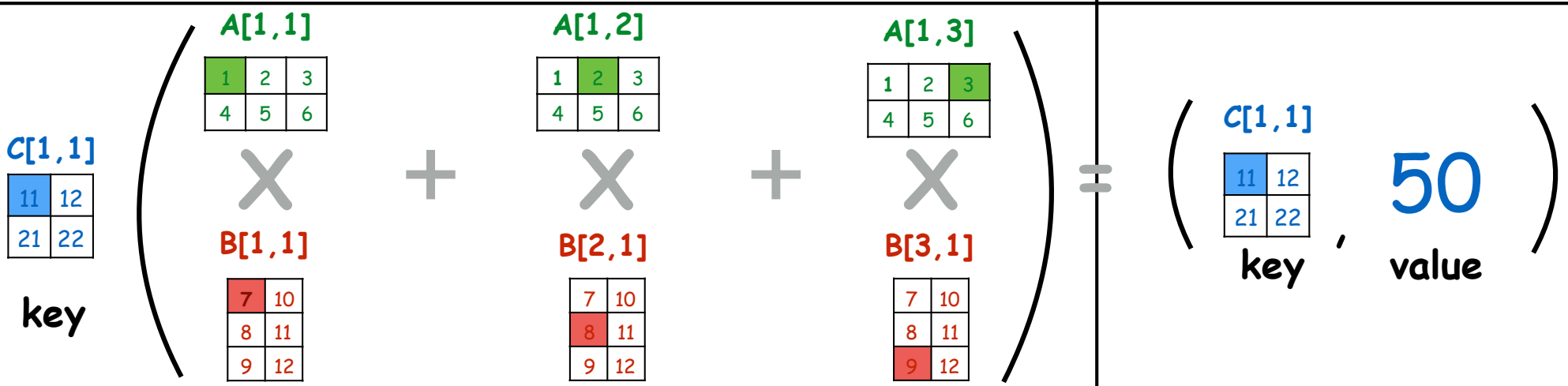
Reducer

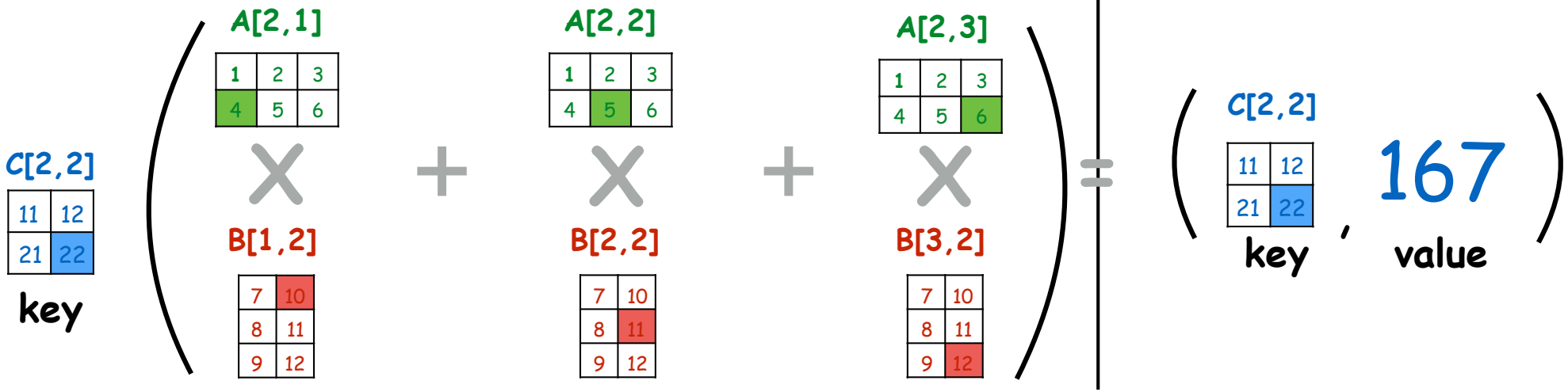
Input:

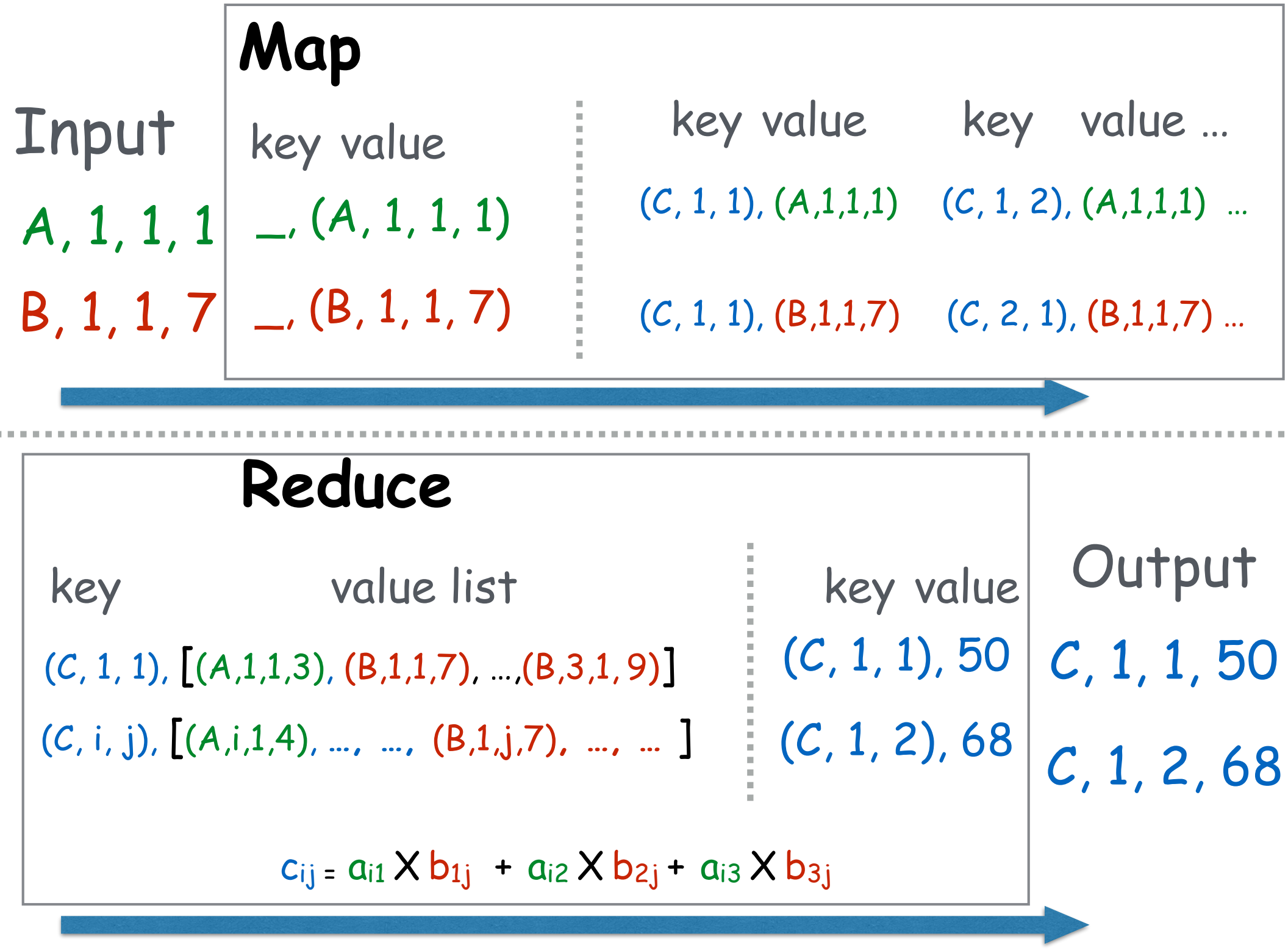
a collection of data records

Output:

one element of matrix C






Map
Input
key value
A, 1, 1, 1
_, (A, 1, 1, 1)
B, 1, 1, 7
_, (B, 1, 1, 7)
key value
key value ...
(C, 1, 1), (A,1,1,1)
(C, 1, 2), (A,1,1,1) ...
(C, 1, 1), (B,1,1,7)
(C, 2, 1), (B,1,1,7) ...
Reduce
key
value list
(C, 1, 1), [(A,1,1,3), (B,1,1,7), ...,(B,3,1, 9)]
(C, i, j), [(A,i,1,4), ..., ..., (B,1,j,7), ..., ...]
key value
(C, 1, 1), 50
(C, 1, 2), 68
Output
C, 1, 1, 50
C, 1, 2, 68
$c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$