Lecture 10

CS 537- Big Data Analytics

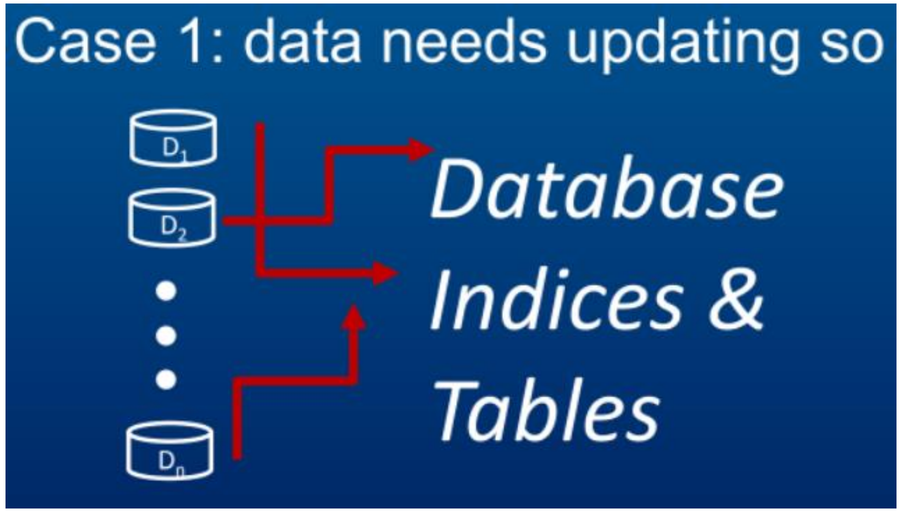
Dr. Faisal Kamiran

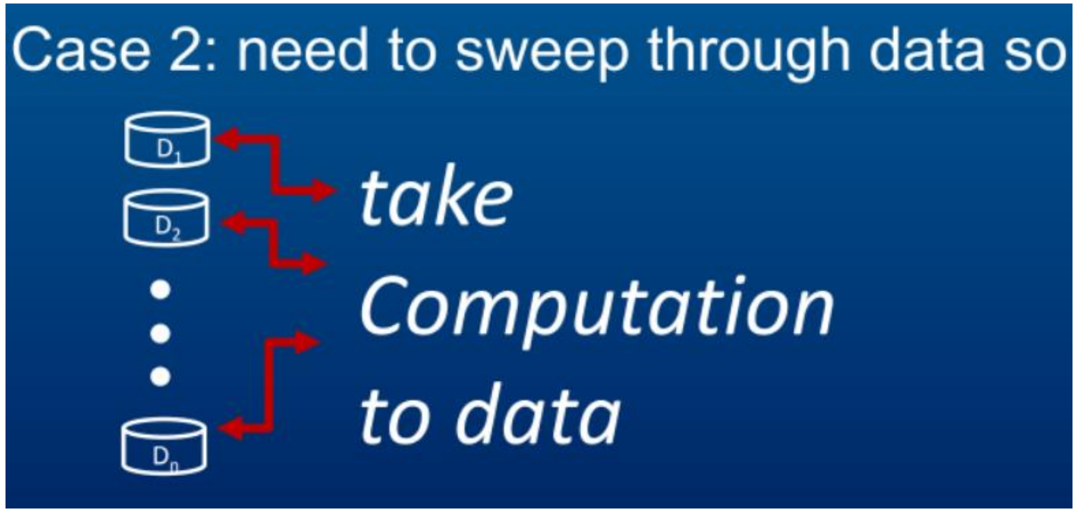
Map Reduce

Scheduling and Data Flow

The Problem and The Solution

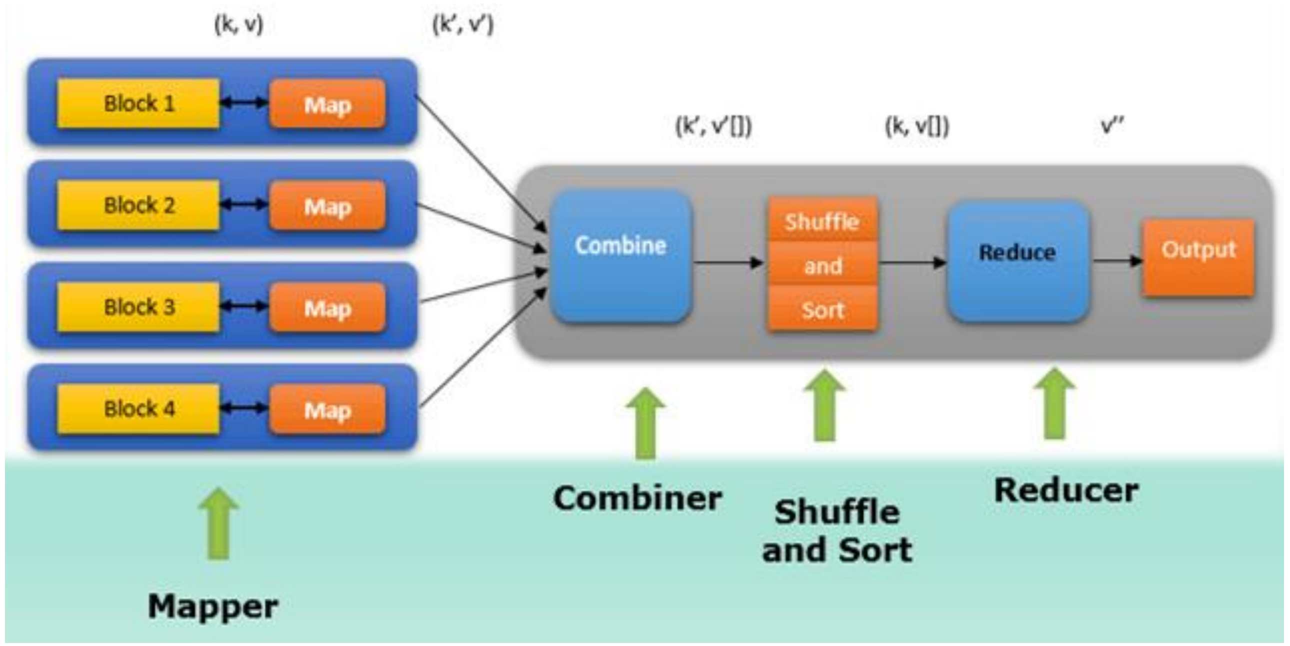
- **Problem:**
  - Big Data -> Large amount of data stored in large amount of devices
- **Solution:**
  - Bring computations to data
- **Possibilities:**





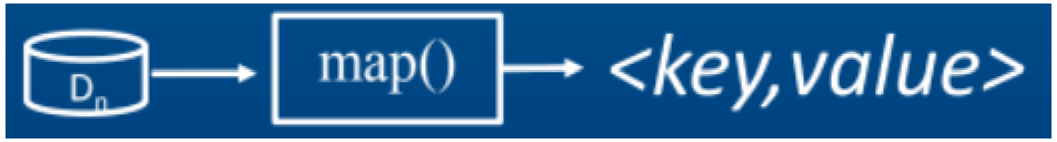
MapReduce Framework

- User defines
  - <key, value>
  - mapper & reducer functions
- Logistics:
  - Hadoop handles the distribution and execution

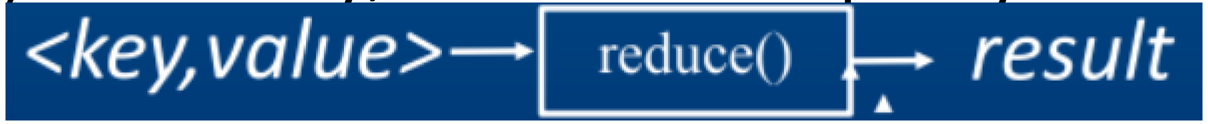


MapReduce Flow

- User defines a map function
  - map() reads data and outputs <key,value>



- User defines a reduce function
  - reduce() reads <key,value> and outputs your result

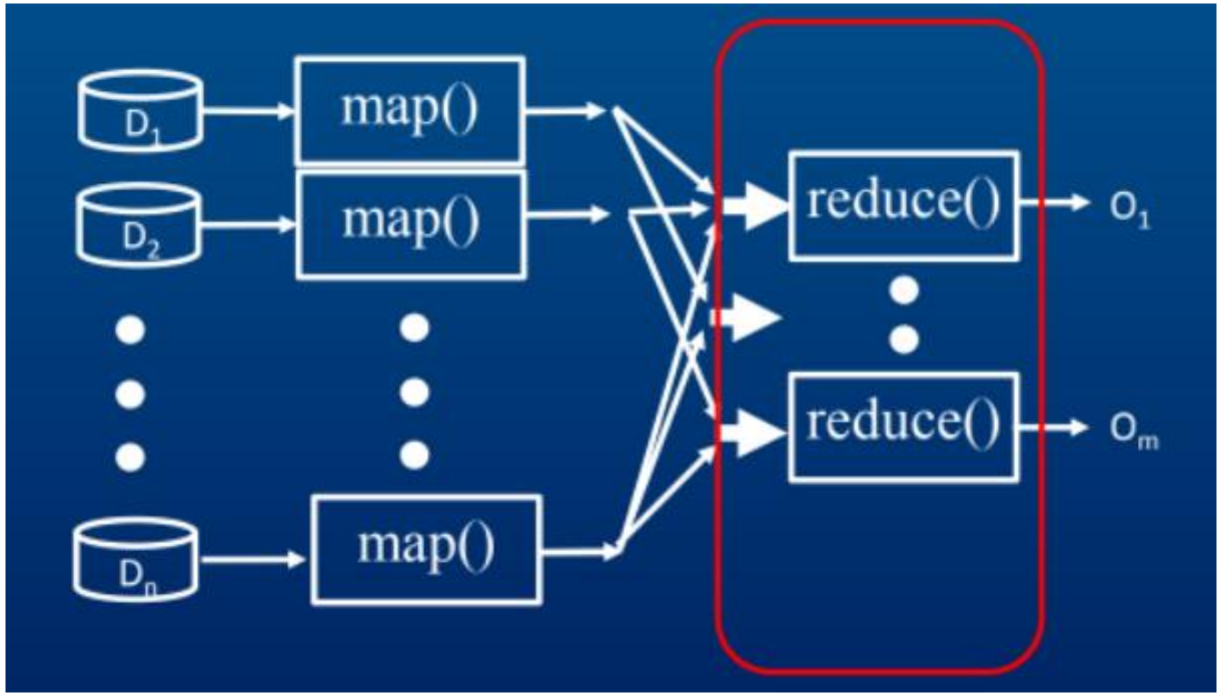


Hadoop Rule of Thumb:

- 1 mapper per data split (typically)
- 1 reducer per computer node (best parallelism)

MapReduce Flow

- Hadoop distributes map() to data
- Hadoop groups <key,value> data
- Hadoop distributes groups to reducers()

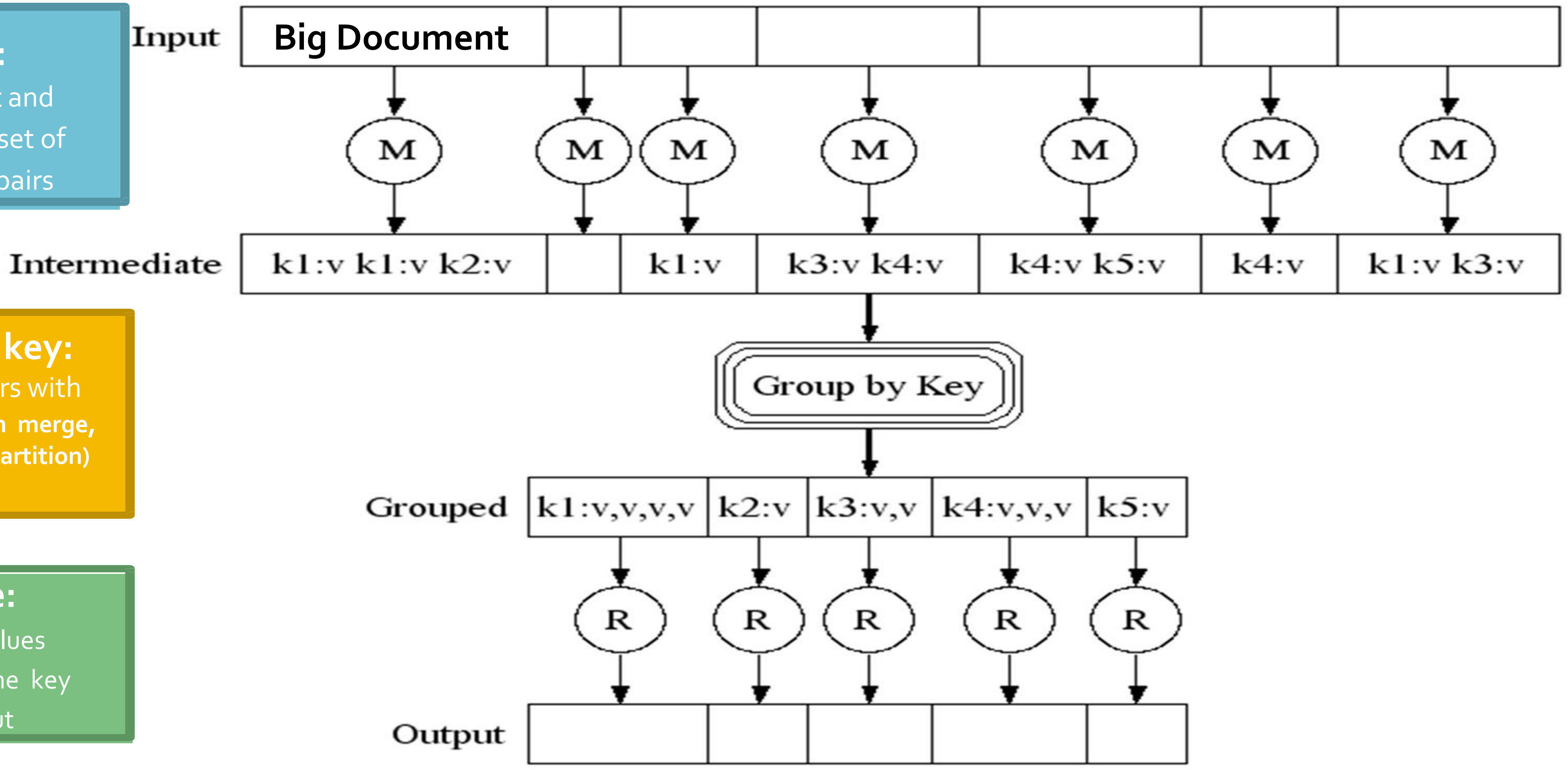


MapReduce Working Diagram

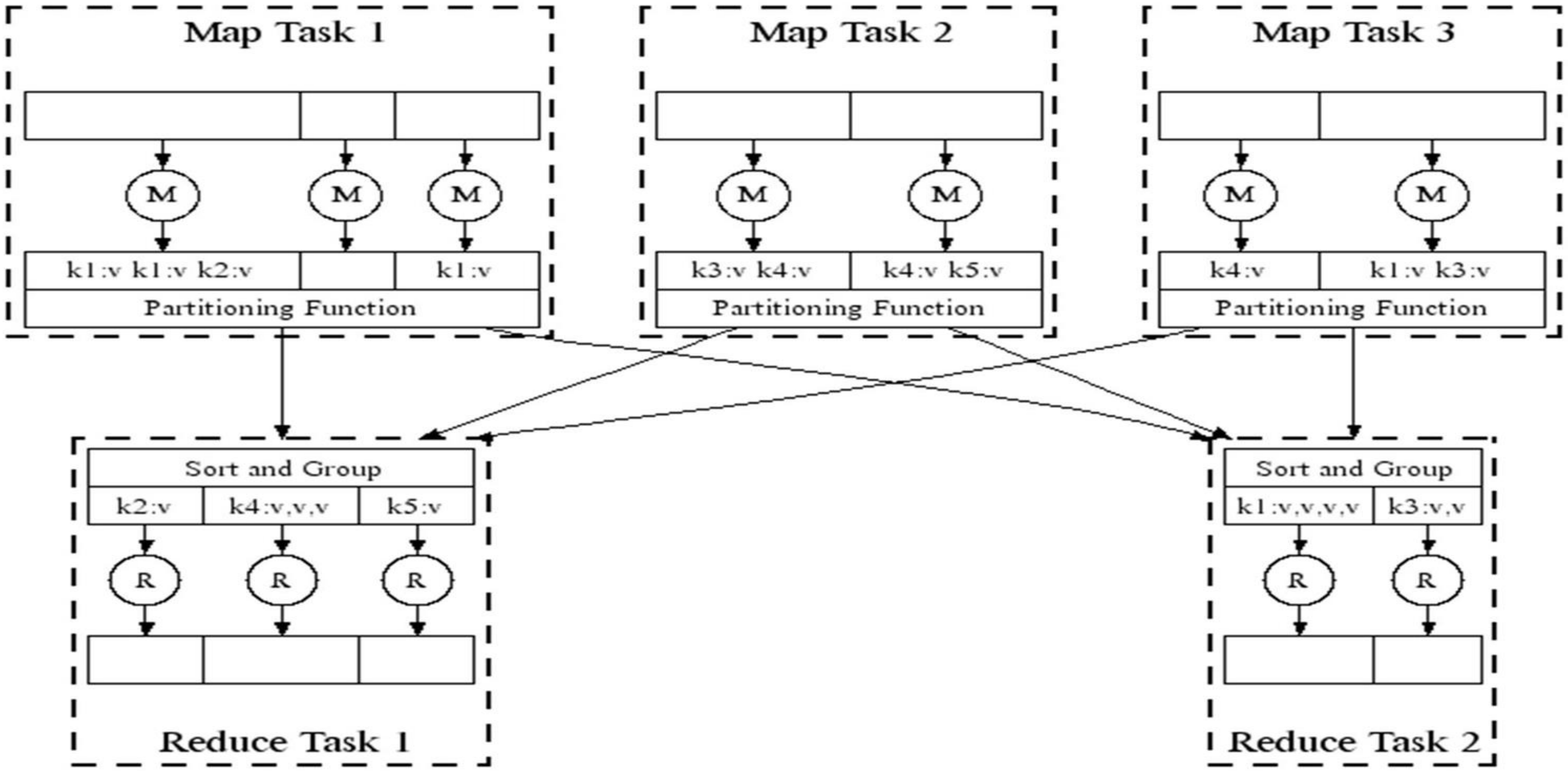
MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key **(Hash merge, Shuffle, Sort, Partition)**

Reduce:
Collect all values belonging to the key and output



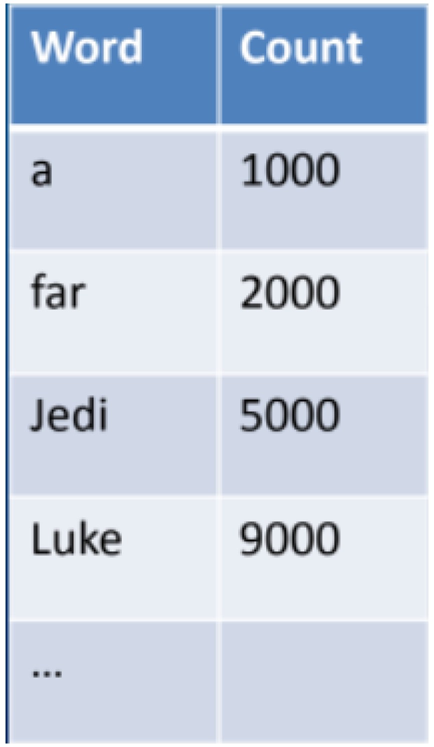
MapReduce: Parallel Processing



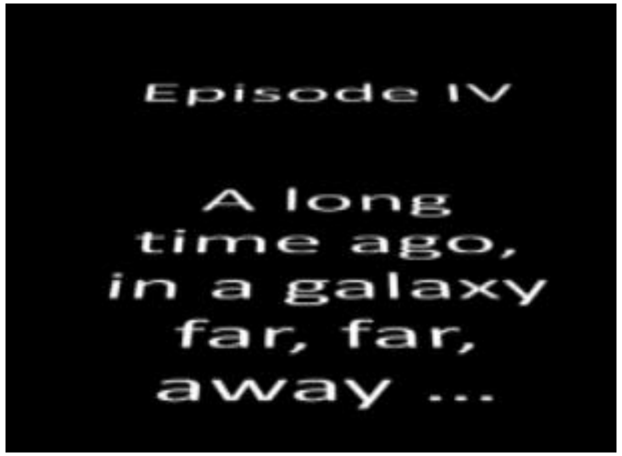
All phases are distributed with many tasks doing the work

Word Count Example

- Count word frequencies
- How would you count all the words in Star Wars?
- In a nutshell:
    - Get word
    - Look up word in table
    - Add 1 to count
- How would you count all the words in all the Star Wars scripts and books, blogs, and fan-fiction?



Word	Count
a	1000
far	2000
Jedi	5000
Luke	9000
…	

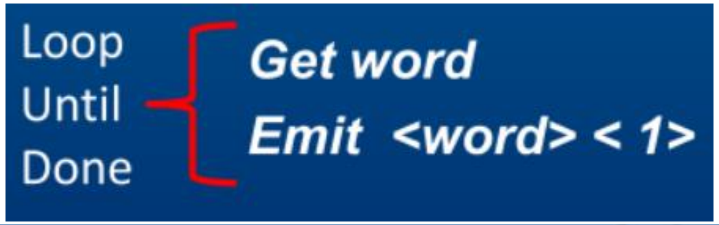


Episode IV

A long time ago, in a galaxy far, far, away …

Strategy: Word Count

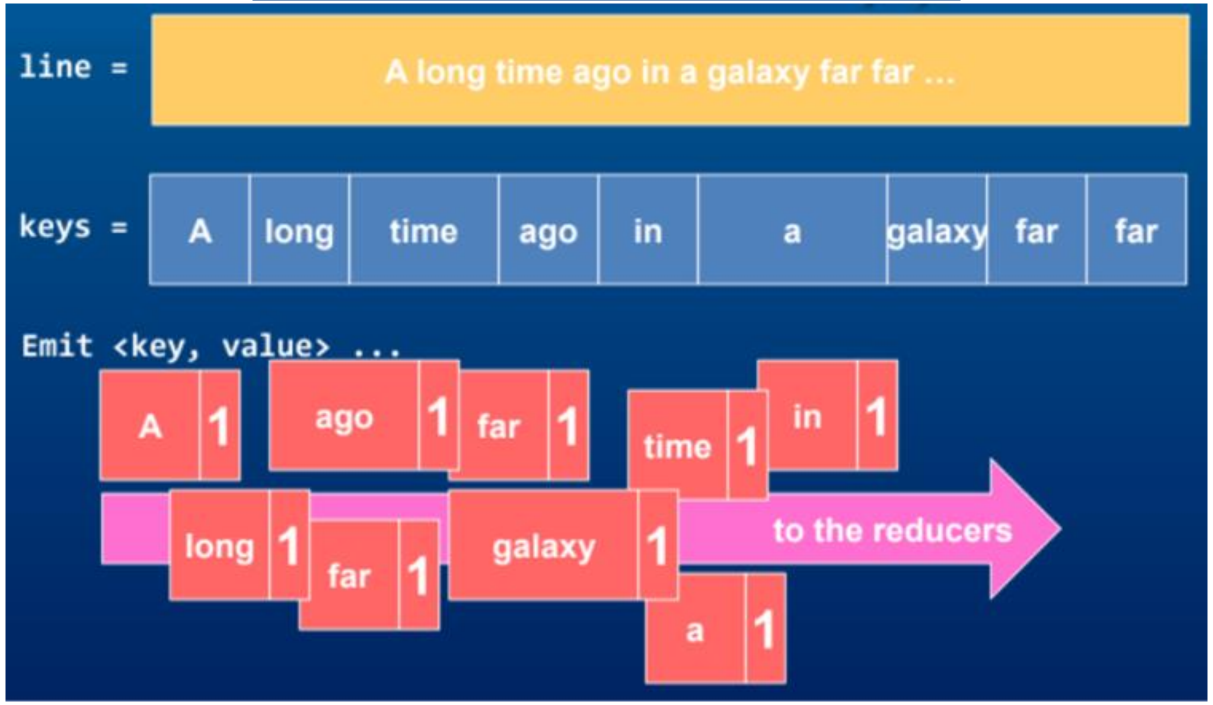
- Keep it simple (remember big data and simple aggregations)
  - Let <word, 1> be the <key,value>
  - The mapper:



```
Loop
Until   { Get word
          Emit <word> < 1>
Done
```

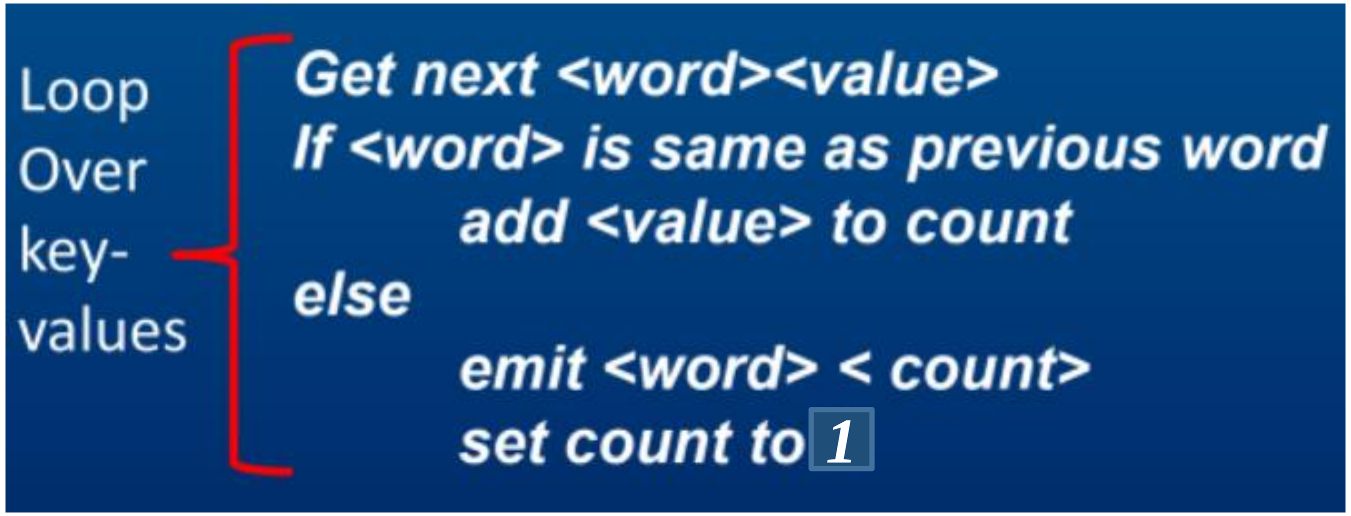
Mappers are separate and independent

Mappers work on data parts



Strategy: Word Count

- Lets Hadoop do the hard work
  - The reducer:

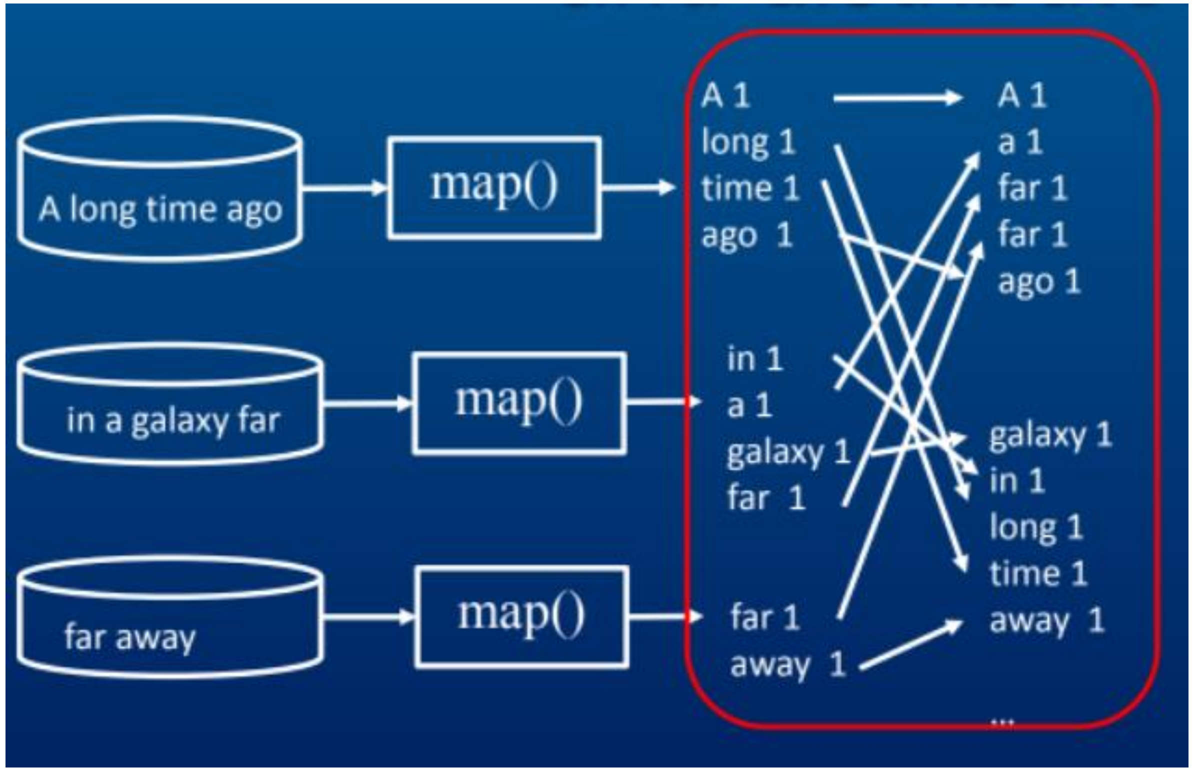


Loop Over key-values

```
Get next <word><value>
If <word> is same as previous word
        add <value> to count
else
        emit <word> < count>
        set count to 1
```

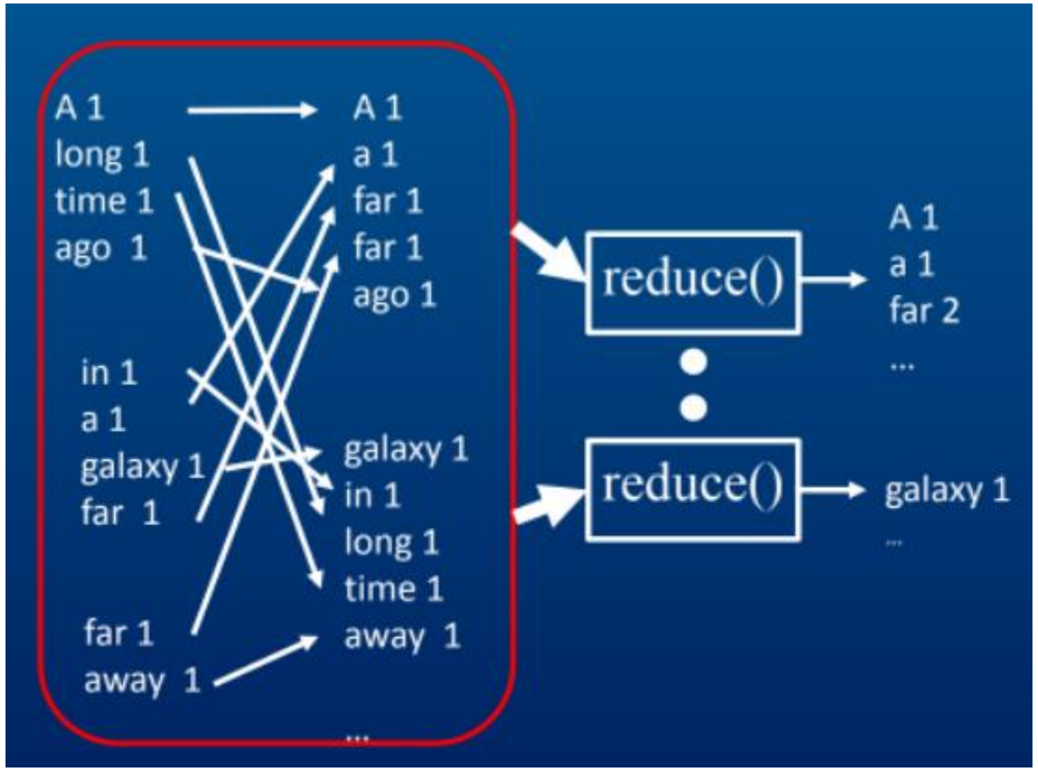
Strategy: Word Count

- Hadoop shuffles, groups, and distributes:



Strategy: Word Count

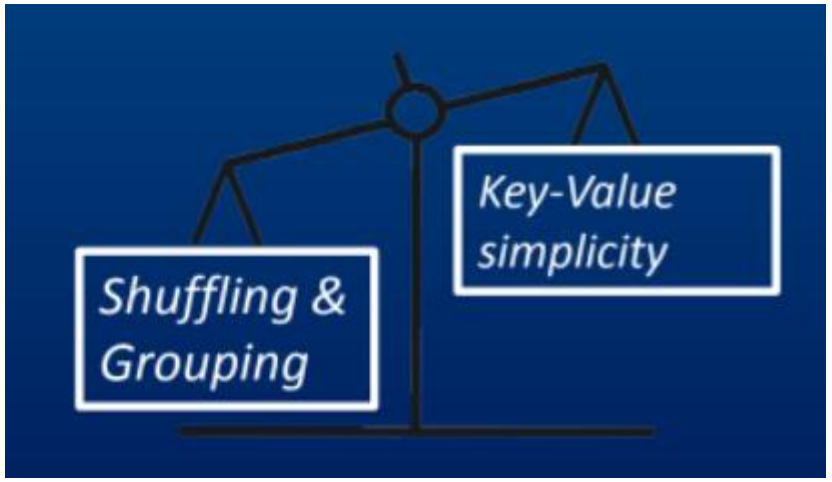
- reduce() aggregates



Ideal properties

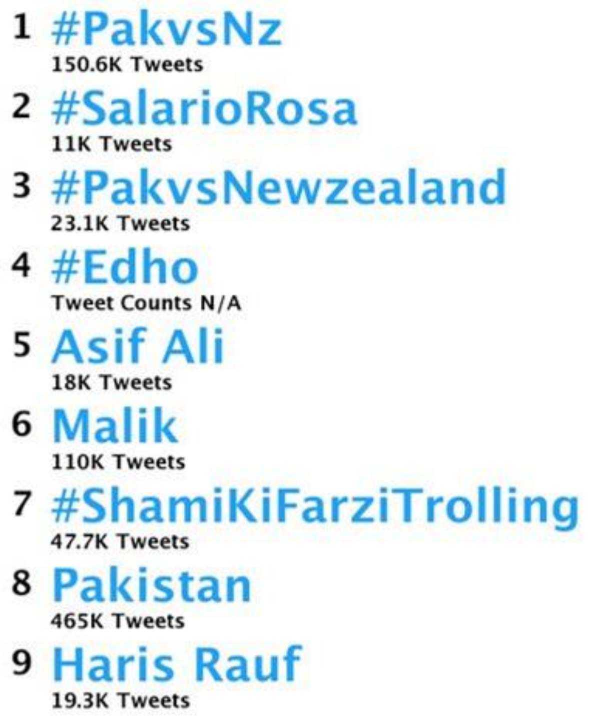
- Good key-value properties
  - Simple
  - Enables reducers to get correct output

- Good Task Decomposition:
  - Mappers: simple and separable
  - Reducers: easy consolidation



Trending Word Counts

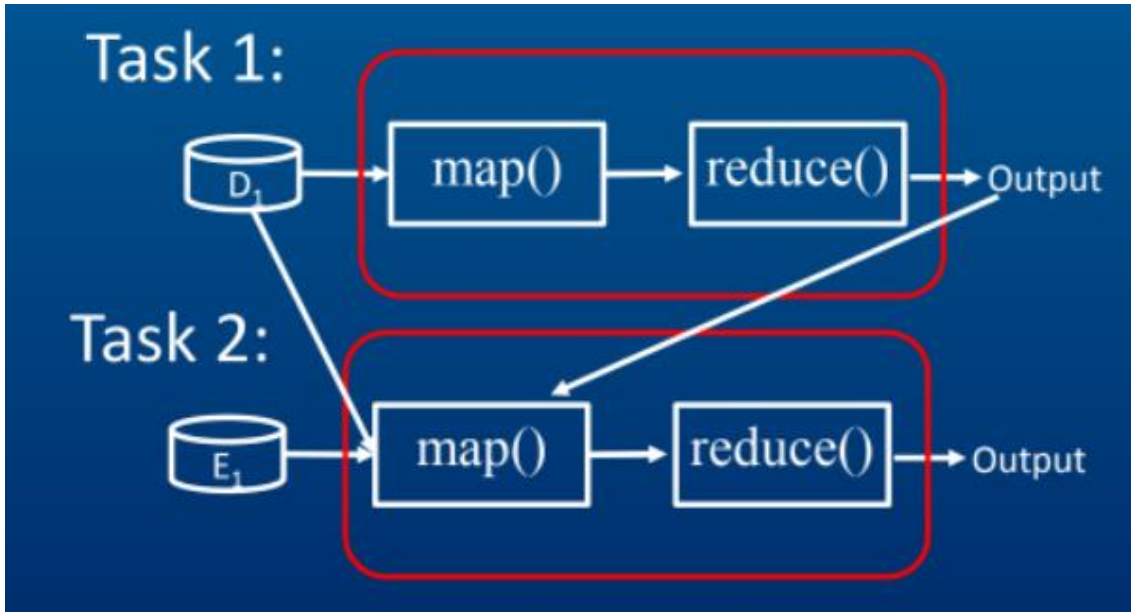
- Let's make first example little complicated:
  - We need to calculate word count in twitter tweets **by day**
    - To find trending topics
  - Twitter Data:
    - Date
    - Message
    - Location
    - Other metadata
  - Tasks
    - Task 1: Get word count by day
    - Task 2: Get total word count



1 #PakvsNz
150.6K Tweets
2 #SalarioRosa
11K Tweets
3 #PakvsNewzealand
23.1K Tweets
4 #Edho
Tweet Counts N/A
5 Asif Ali
18K Tweets
6 Malik
110K Tweets
7 #ShamiKiFarziTrolling
47.7K Tweets
8 Pakistan
465K Tweets
9 Haris Rauf
19.3K Tweets

Trending Word Counts: Task Decomposition

- For task 1 we need to use composite key:
  - Map/Reduce: <**date word**,count>
- For task 2 we can:
  - Reuse previous word count example
  - Use the output of task one

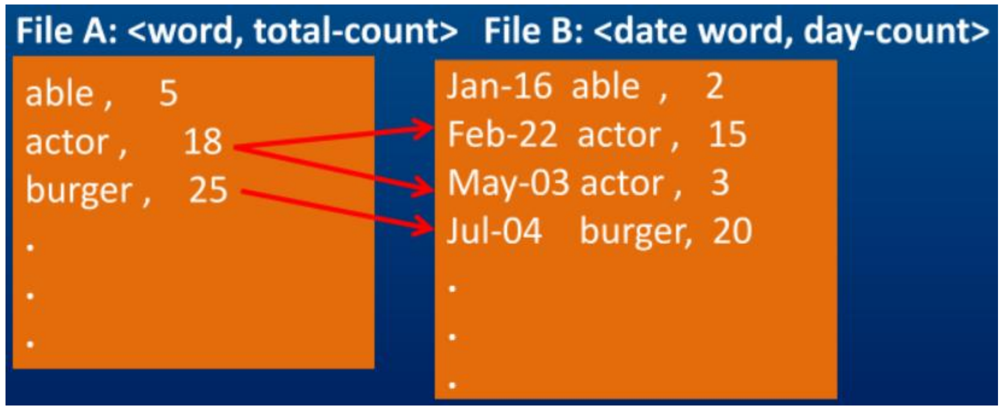


Joining Data

- Task: combine datasets by key
  - – A standard data management function
  - – In pseudo SQL

Select * from table A, table B, where A.key=B.key

- Joins can be inner, left or right outer
- Task: given two wordcount datasets as following:

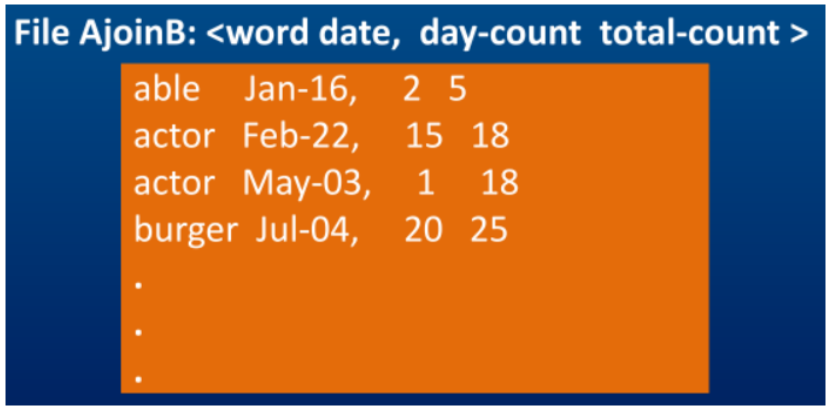


File A: <word, total-count>

```
able ,    5
actor ,    18
burger ,   25
.
.
.
```

File B: <date word, day-count>

```
Jan-16 able ,   2
Feb-22 actor ,  15
May-03 actor ,  3
Jul-04   burger, 20
.
.
.
```

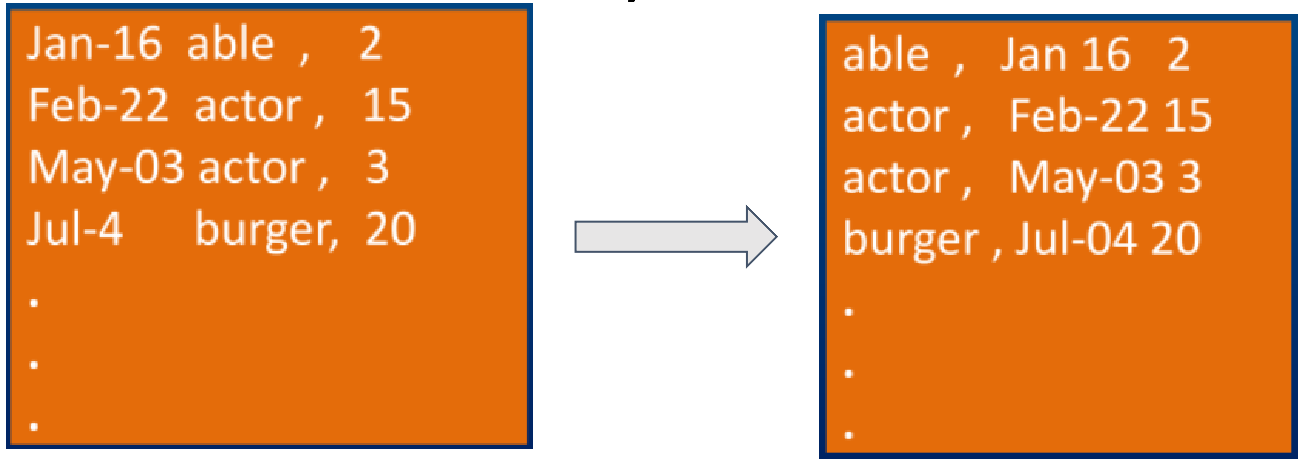


File AjoinB: <word date, day-count total-count >

```
able     Jan-16,    2   5
actor   Feb-22,    15  18
actor   May-03,    1    18
burger  Jul-04,    20  25
.
.
.
```

Joining Data (Cont)

- For joining keys should be same but here:
  - File A: <word, total-count>
  - File B: <date word, day-count>
    - Word is same in both keys but date is not present in File A so we need to filter out date for key of File B
    - Now: Put Date into value field
      - File B: <word, date day-count total-count >

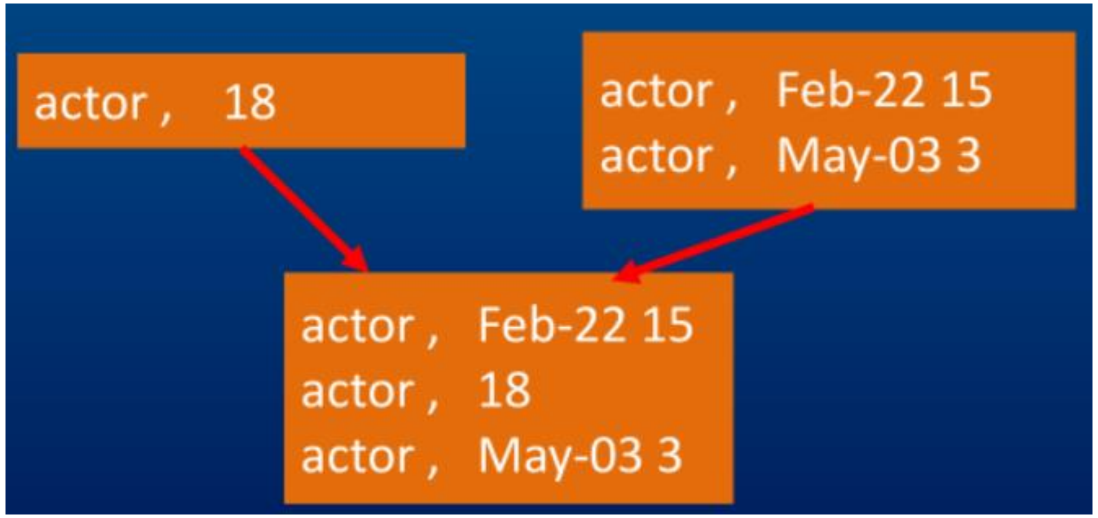


```
Jan-16  able  ,   2
Feb-22  actor ,   15
May-03 actor ,   3
Jul-4      burger,  20
.
.
.
```

```
able  ,   Jan 16   2
actor ,   Feb-22 15
actor ,   May-03 3
burger , Jul-04 20
.
.
.
```

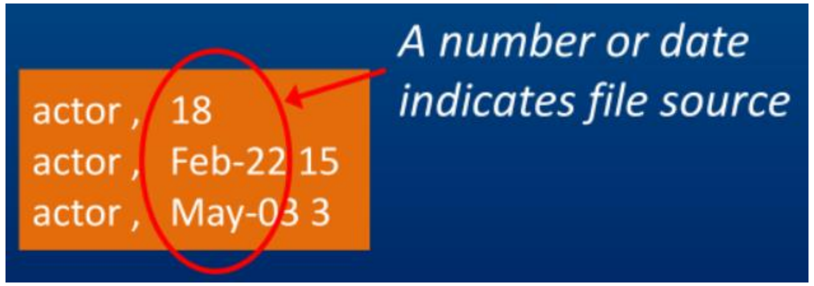
Task Decomposition

- How will Hadoop shuffle & group these?
- Let’s focus on 1 key:
- Hadoop gathers the data for a join

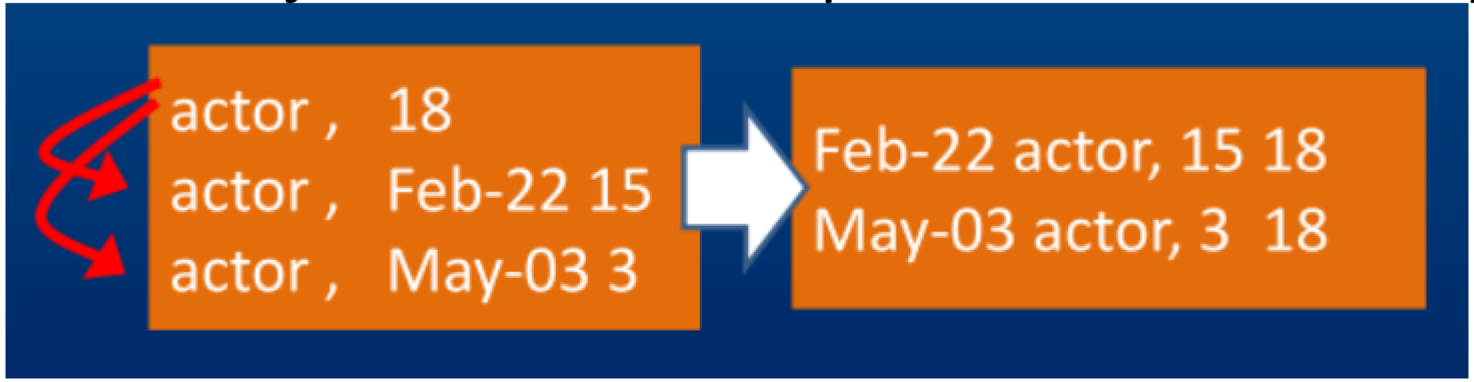


Task Decomposition

- Reducer now has all the data for same word grouped together



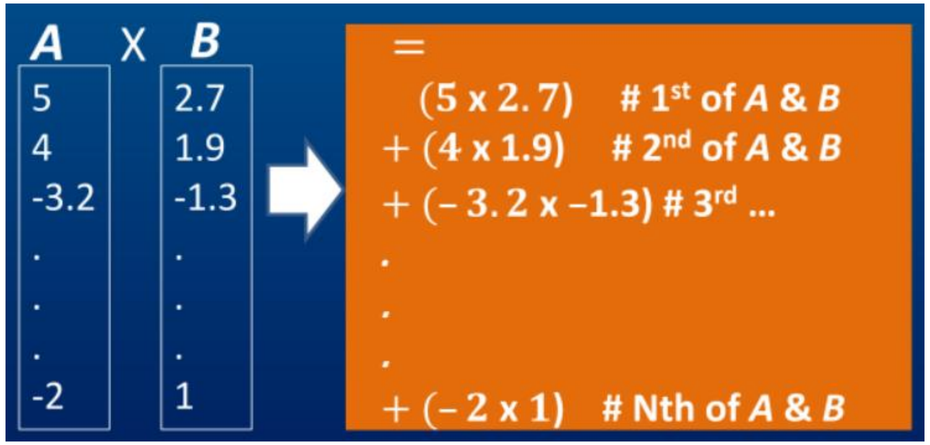
- Reducer can now join the data and put date back into key



Vector Multiplication

- Task: multiply 2 arrays of N numbers
  - – A basic mathematical operation
  - – Let's assume N is very large
  - Data is distributed in HDFS
  - We need elements with same index together

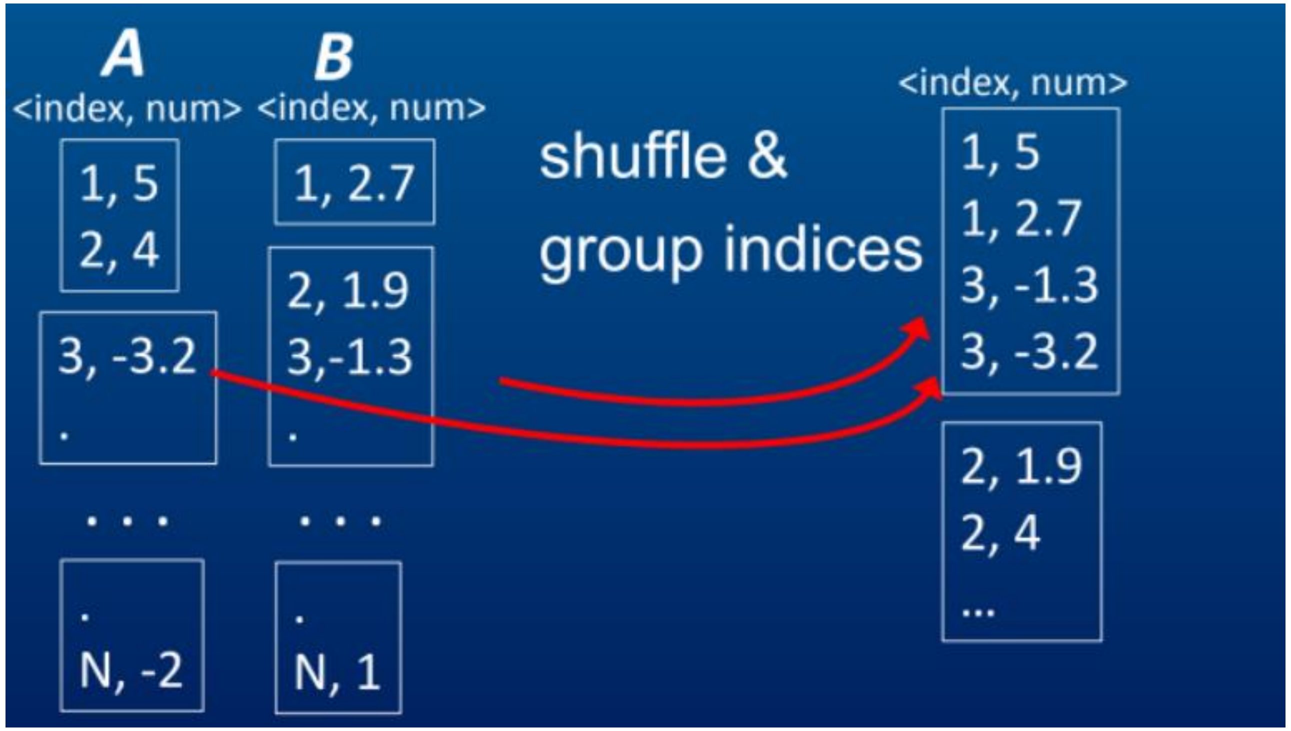
Let <key, value> = <index, number>



A X B
5 4 -3.2 . . . -2
2.7 1.9 -1.3 . . . 1
=
(5 x 2.7) # 1st of A & B
+ (4 x 1.9) # 2nd of A & B
+ (– 3.2 x –1.3) # 3rd ...
.
.
.
+ (– 2 x 1) # Nth of A & B

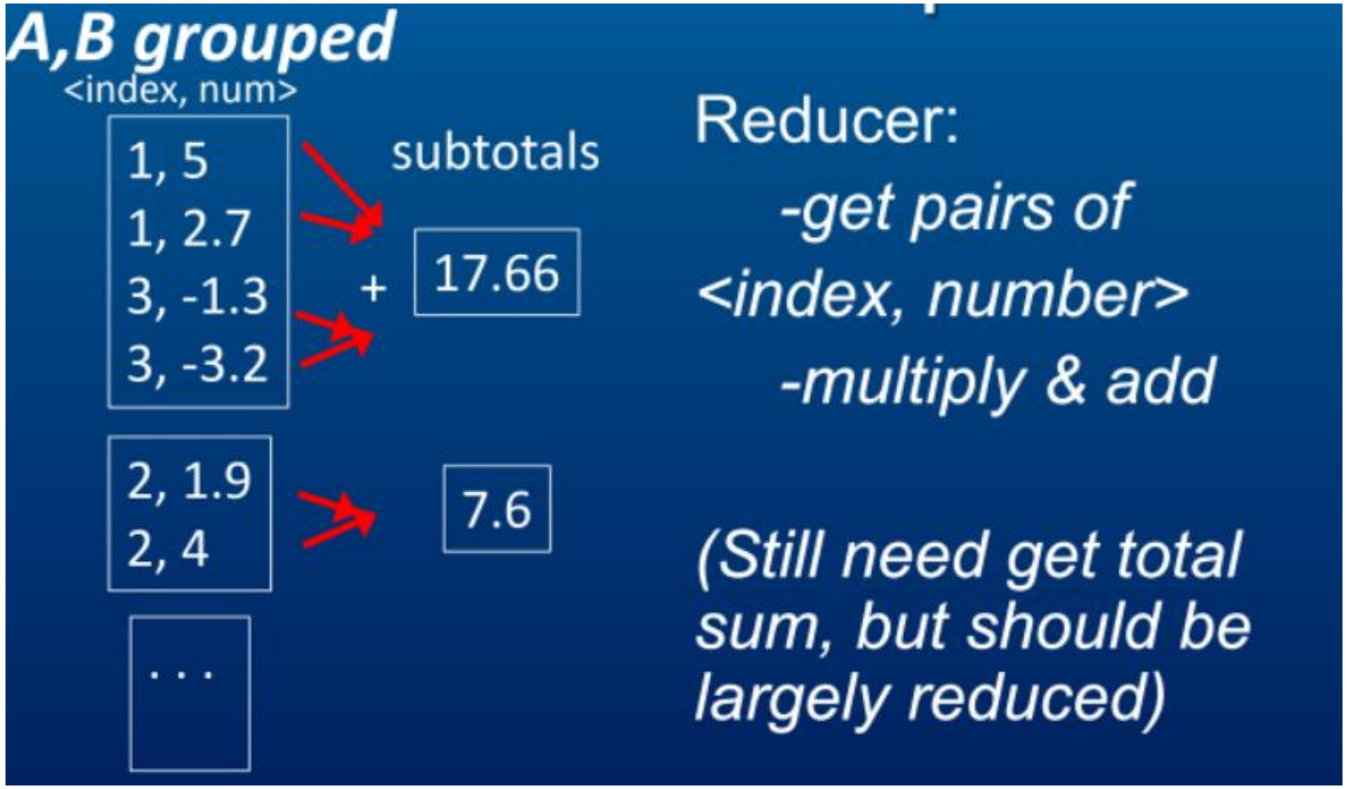
Vector Multiplication

- Lets assume we already have indexes of elements stored
- Mapper task is as following:



Vector Multiplication

- Lets assume we already have indexes of elements stored
- Reducer task is as following:

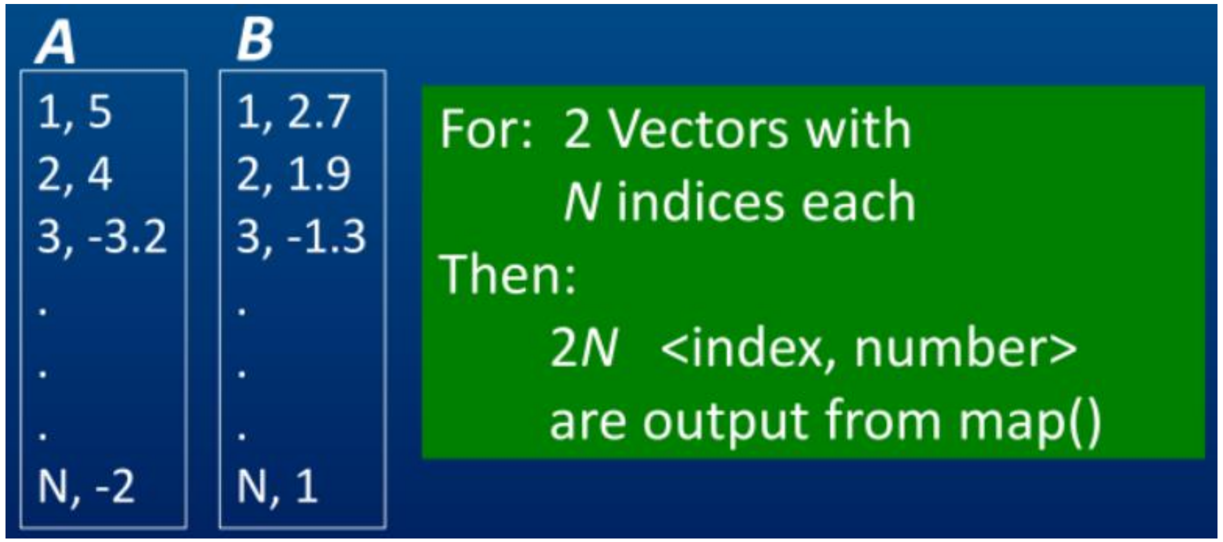


Reducer:
-get pairs of
<index, number>
-multiply & add

(Still need get total sum, but should be largely reduced)

Computational Costs

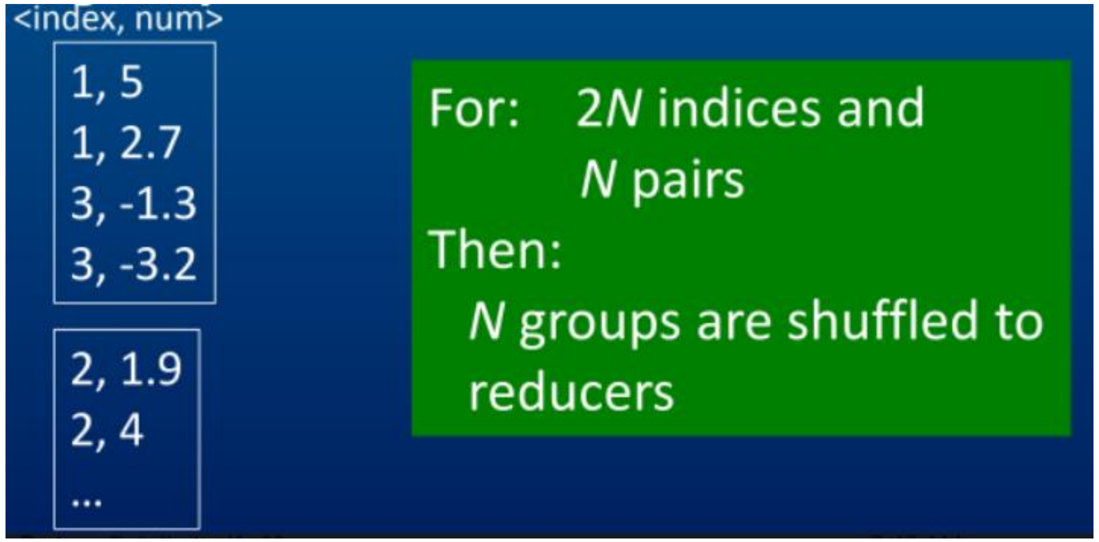
- For Vector Multiplication
  - – How many <index, number> are output from map()?
  - – How many <index> groups have to be shuffled?



A	B
1, 5	1, 2.7
2, 4	2, 1.9
3, -3.2	3, -1.3
.	.
.	.
.	.
N, -2	N, 1

For: 2 Vectors with
N indices each
Then:
2*N* <index, number>
are output from map()

How many <index, number> are output from map()



<index, num>

1, 5
1, 2.7
3, -1.3
3, -3.2

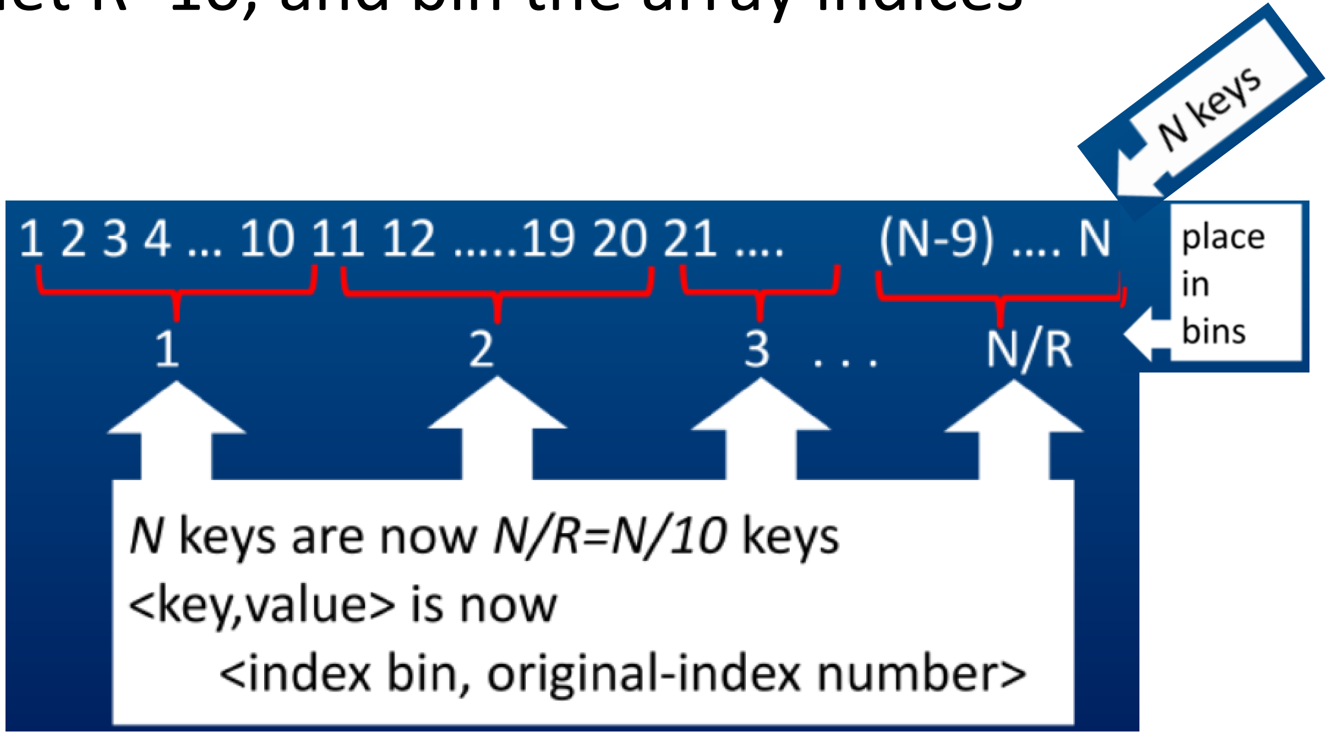
2, 1.9
2, 4
...

For: 2*N* indices and
N pairs
Then:
N groups are shuffled to
reducers

How many <index> groups have to be shuffled?

Computational Costs

- We can reduce shuffling by:
  - Try: ‘combine’ map indices in mapper (works better for Wordcount)
  - Or Try: use index ranges of length R
- For example, let R=10, and bin the array indices


N keys
1 2 3 4 … 10 11 12 …..19 20 21 …. (N-9) …. N
place in bins
1
2
3 . . .
N/R
N keys are now N/R=N/10 keys
<key,value> is now
<index bin, original-index number>

Computational Costs

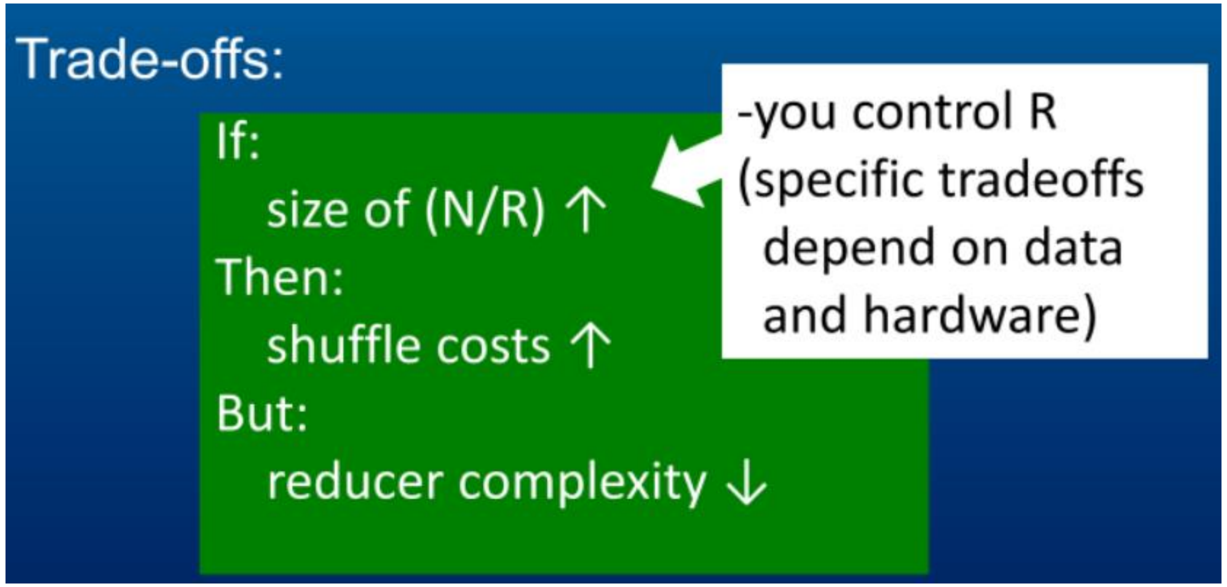
- Now shuffling costs depend on N/R groups

  If: R=1

  Then: N/R=N groups (same as before)

  If: R>1

  Then: N/R<N (less shuffling to do)



Trade-offs:

If:
size of (N/R) ↑
Then:
shuffle costs ↑
But:
reducer complexity ↓

-you control R
(specific tradeoffs depend on data and hardware)

Note: Matrix multiplication needs row-index and col-index in the keys

MapReduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
  - Scheduling the program’s execution across a set of machines
- Performing the group by key step
- Handling node failures
- Managing required inter-machine
- communication

MapReduce: Environment

- Input and final output are stored on the distributed file system (DFS):
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
- Intermediate results are stored on local FS of Map and Reduce workers
- Output is often input to another MapReduce task

MapReduce: Coordination Master

- Master node takes care of coordination:
  - Task status: (idle, in-progress, completed)
    - Idle tasks get scheduled as workers become available
    - When a map task completes, it sends the master the location and sizes of its R intermediate files, one for each reducer
  - Master pushes this info to reducers
  - Master pings workers periodically to detect failures

MapReduce: Dealing with Failures

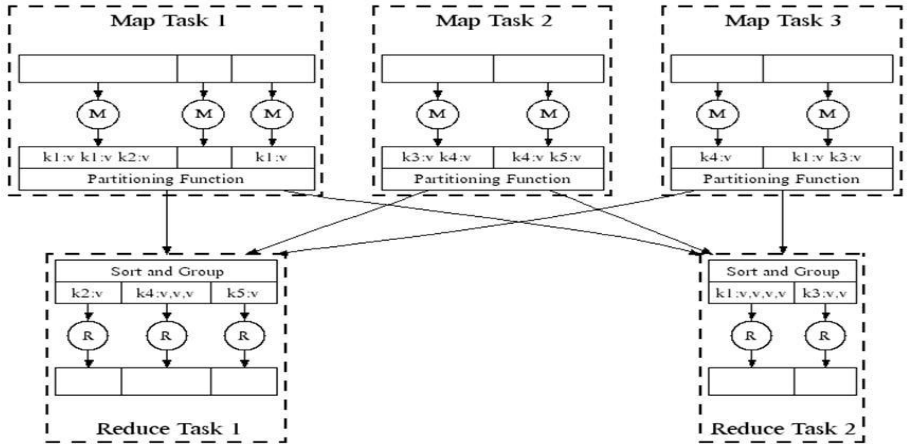
- Map worker failure
  - Map tasks completed or in-progress at worker are reset to idle.
  - Idle tasks eventually rescheduled on other worker(s)
- Reduce worker failure
  - Only in-progress tasks are reset to idle
  - Idle Reduce tasks restarted on other worker(s)
- Master failure
  - MapReduce task is aborted and client is notified

How many Map and Reduce Jobs

- Suppose we have M map tasks, R reduce tasks
- Rule of thumb:
  - Make M much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
  - **Usually R is smaller than M**
  - Because output is spread across R files

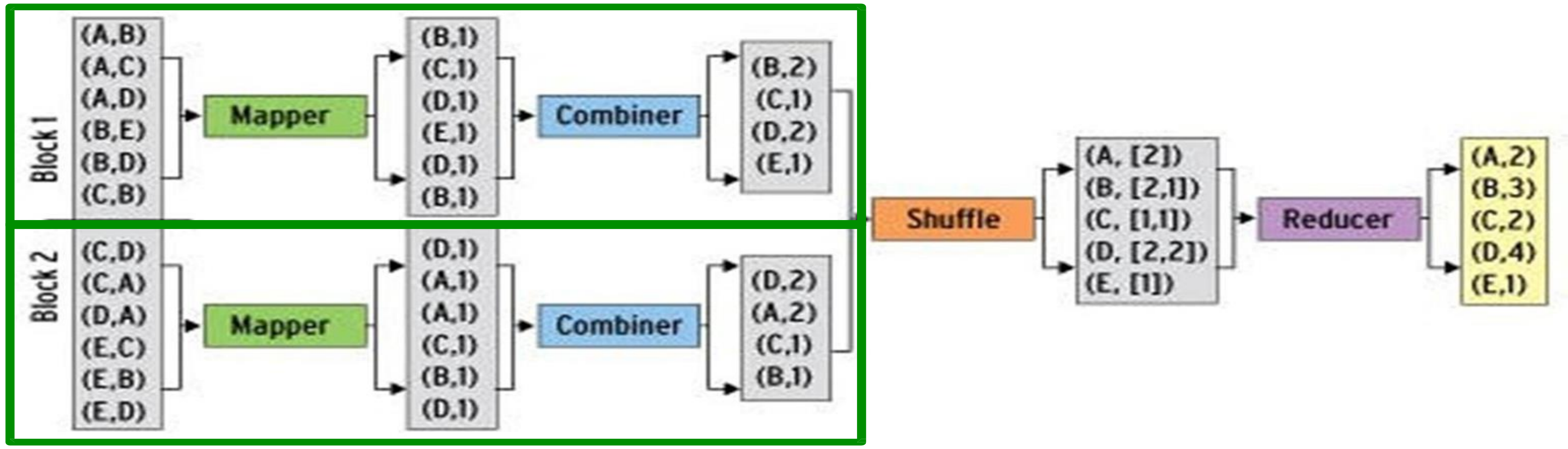
Combiners

- Often a Map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in the word count example
  - Can save network time by pre-aggregating values in the mapper:
- combine(k, list(v1)) v2
- Combiner is usually same as the reduce function



Combiners (Cont.)

- Back to our word counting example:
  - Combiner combines the values of all keys of a single mapper (single node):



- Much less data needs to be copied and shuffled!

Combiners (Cont.)

- Combiner trick works only if reduce function is commutative and associative.
- Sum:

$$2 + (5 + 7) = (2 + 5) + 7$$

- Average
- Median

Partition Function

- Want to control how keys get partitioned
    - The set of keys that go to a single reduce worker
- System uses a default partition function:
    - Hash (key) mod R
- Sometimes useful to override the hash function:
    - E.g., hash (hostname(URL)) mod R ensures URLs from a host end up in the same output file

Limitations of MapReduce

- Must fit <key, value> paradigm
- Map/Reduce data not persistent
- Requires programming/debugging
- Not interactive

That’s all for today.