# Cluster Based MapReduce

Anurag Nagar, Ph.D.

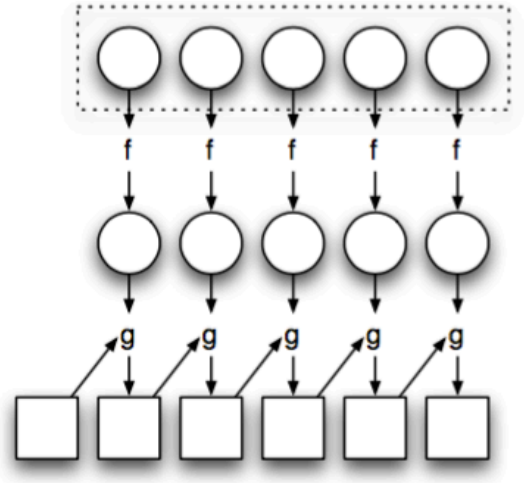
# MapReduce Basics

# MR has origins in Functional Programming

- Map is a higher order function that applies a function element-wise to a list of elements.
- Map transform **lists** of input data elements into **lists** of output data elements by applying a function to each element of the list.
- Reduce (also called Fold) is a higher order function that processes a list of elements by applying a function pairwise and finally returning a scalar.
- Reduce compacts a list into a scalar by applying a function pairwise.

# Functional Programming

- **Key feature: higher order functions**
  - Functions that accept other functions as arguments
  - **Map** and **Fold** **(Reduce)**




f is applied to every element and it results in a new list

g starts with an initial value and reduces every element i.e. compacts list to a scalar

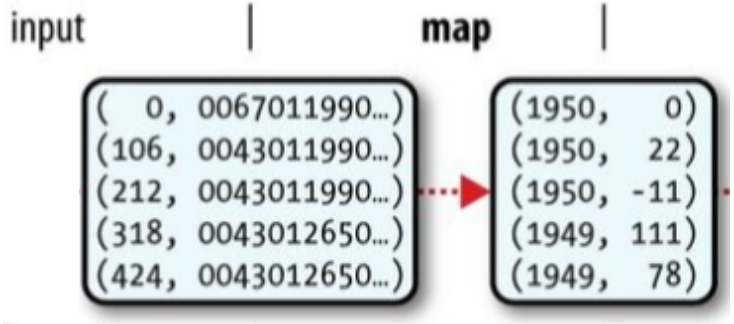
**Figure:** Illustration of *map* and *fold*.

# Map Operation

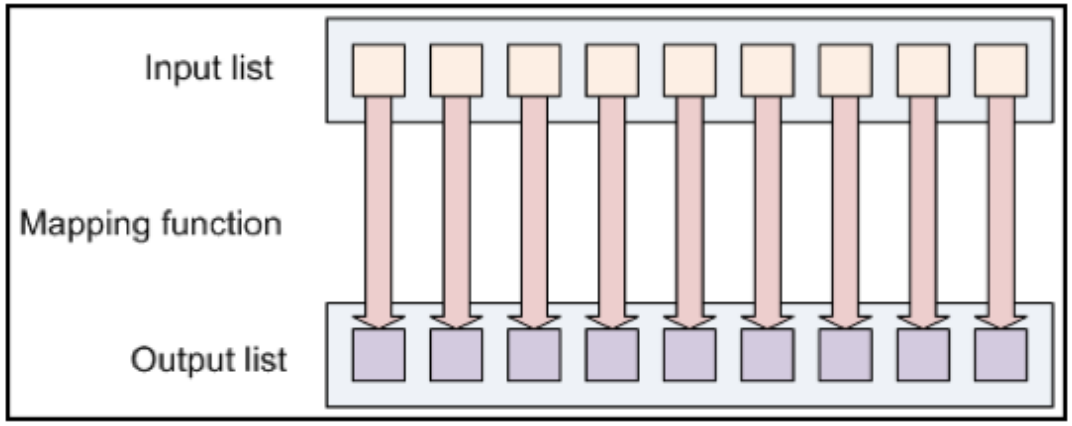
- Define a function:
```
square x = x * x
```
- Apply on a list:
```
>>> map square [1, 2, 3, 4, 5]
```
- Get another list: `[1, 4, 9, 16, 25]`.

# Map function

- Takes input (k, v) and outputs (k', v')
  => Generally input k has little meaning, but we try to find a meaningful output k'
- Example: You have input file with line number as key and text as value. A map function could extract and output year as key and temperature as value.



# Mapping




Mapper Process

# Reduce (Fold) Operation

- Define an operator: +
- Initial value = 0
- Apply on a list: `[1,2,3,4,5]`
- Get a scalar: 15

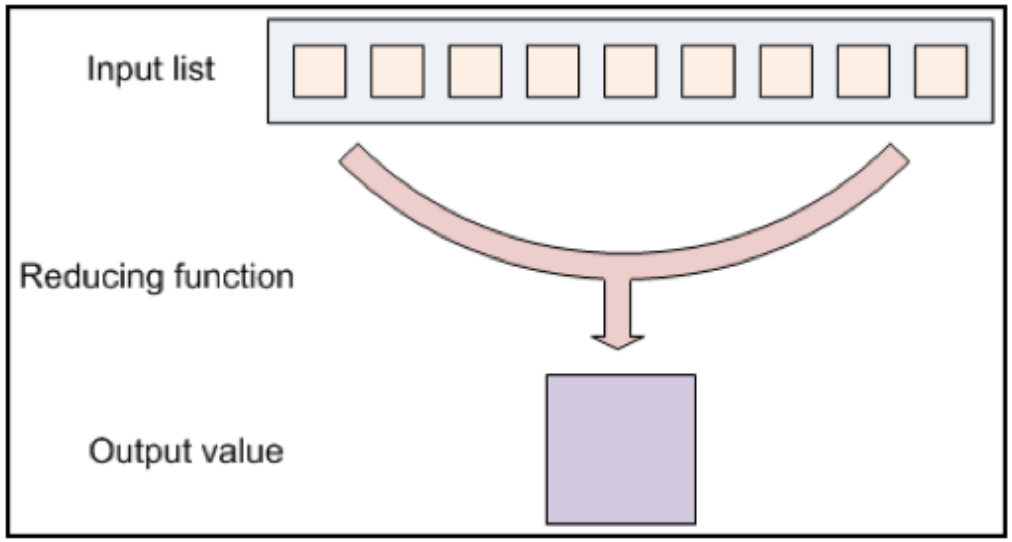
# Reduce function

- Reduce function generally receives a key and a list of values.
- It compacts the list to a single (generally) value.
- For example, input key is year and value is list of temperatures. Output could be key and maximum temperature.

  (1950, [30, 70, 50, 72, 18]) -> (1950, 72)

- A key point is that reduce is generally run on data from same key value.
  => Eg. Find average time spent by each visitor on a website
  Key = userID, Value = Time spent during each visit
  It makes sense to aggregate (reduce) for each key separately

# Reducing




Reducer Process

# MapReduce Data Structures

# Key-Value Structure

- Each data element needs to have a **key** associated with it.
- Uniquely identifies the data item.
- Example: Log of cars passing by.
  What's the key?
  Could be the license plate number.

```
AAA-123   65mph, 12:00pm
ZZZ-789   50mph, 12:02pm
AAA-123   40mph, 12:05pm
CCC-456   25mph, 12:15pm
...
```

- Does it have to be unique in entire dataset? No

# K-V pairs

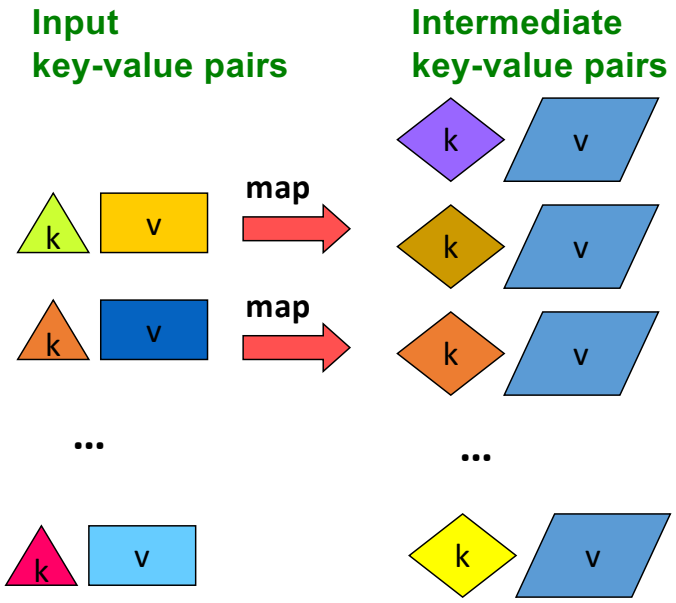
- Key-Value (K-V) pairs are one of the basic data structures for BD.
- Please keep this in mind for future discussion also.

# K-V pairs

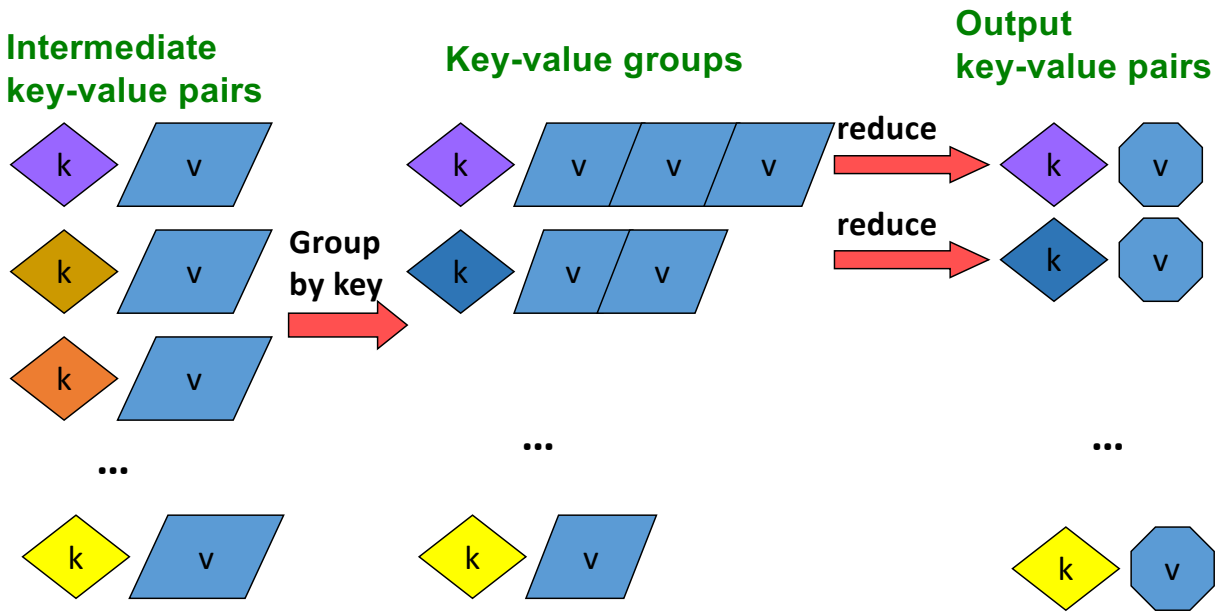
- A mapper is presented data that contains multiple keys.
- It transforms this data in a 1-1 fashion and outputs a meaningful K-V pair.
- The reducer is presented with data containing only a single key.
- It compacts (or aggregates) the values of the key.
- How does each reducer get data from only one key?
- Someone has to do the sorting and shuffling of data from mappers to reducers.
- That's the job of the Hadoop framework

# MapReduce in Practice

# MapReduce: The Map Step



# MapReduce: The Reduce Step



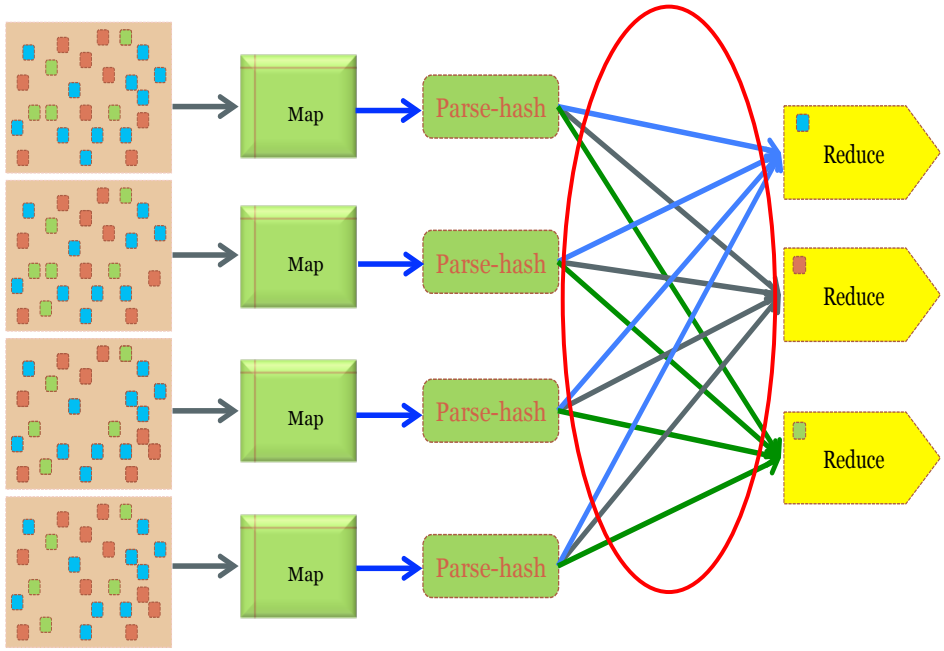
# Key-Value Pairs

- 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>>

# Example 1 – Color Count

# MapReduce Execution in Hadoop

- Suppose you are given a dataset where each item is keyed with a color – Red, Blue, or Green
- Aim is to compute the count of each colors.




*Dataset is divided into 4 blocks.*

*The map-reduce job consists of 4 map tasks and 3 reduce tasks*

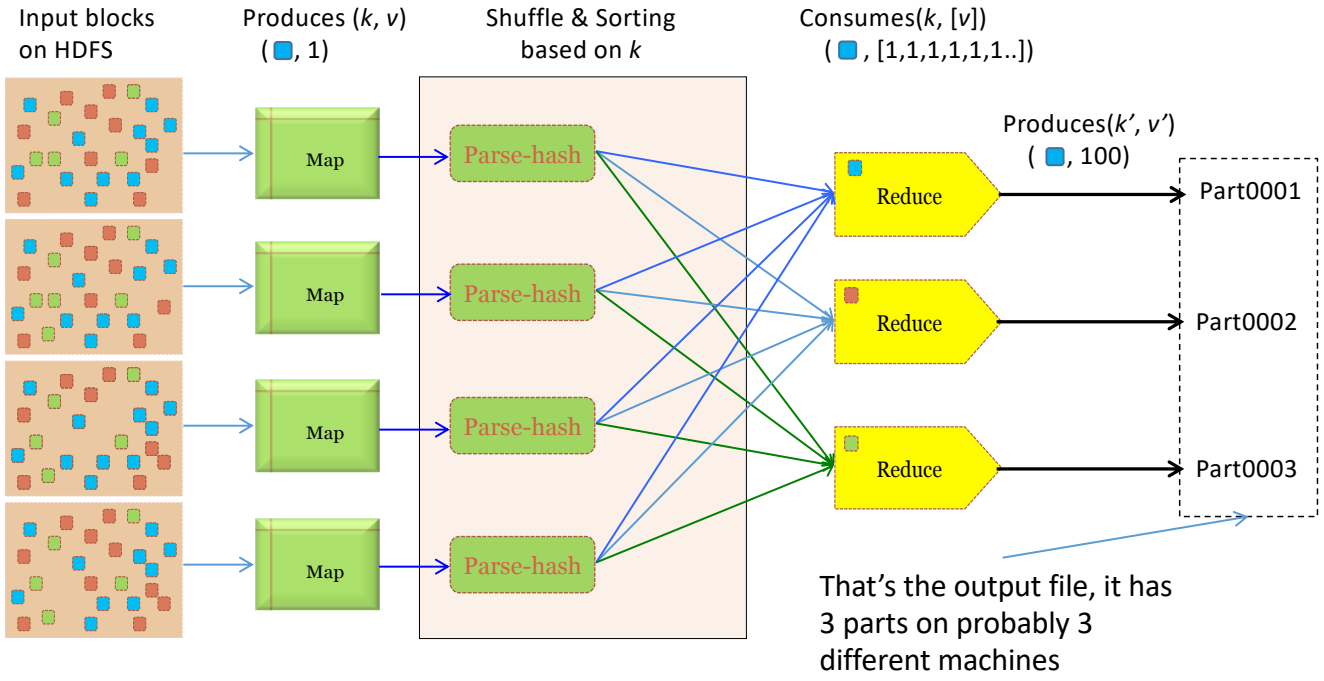
*Map task takes each data item and applies a transformation to it. Could be as simple as output (key, 1) e.g. (Red, 1)*

*Reduce task needs to get data of a single key.*

*Framework does the sorting and shuffling*

# Color Count Example

**Job: Count the number of each color in a data set**



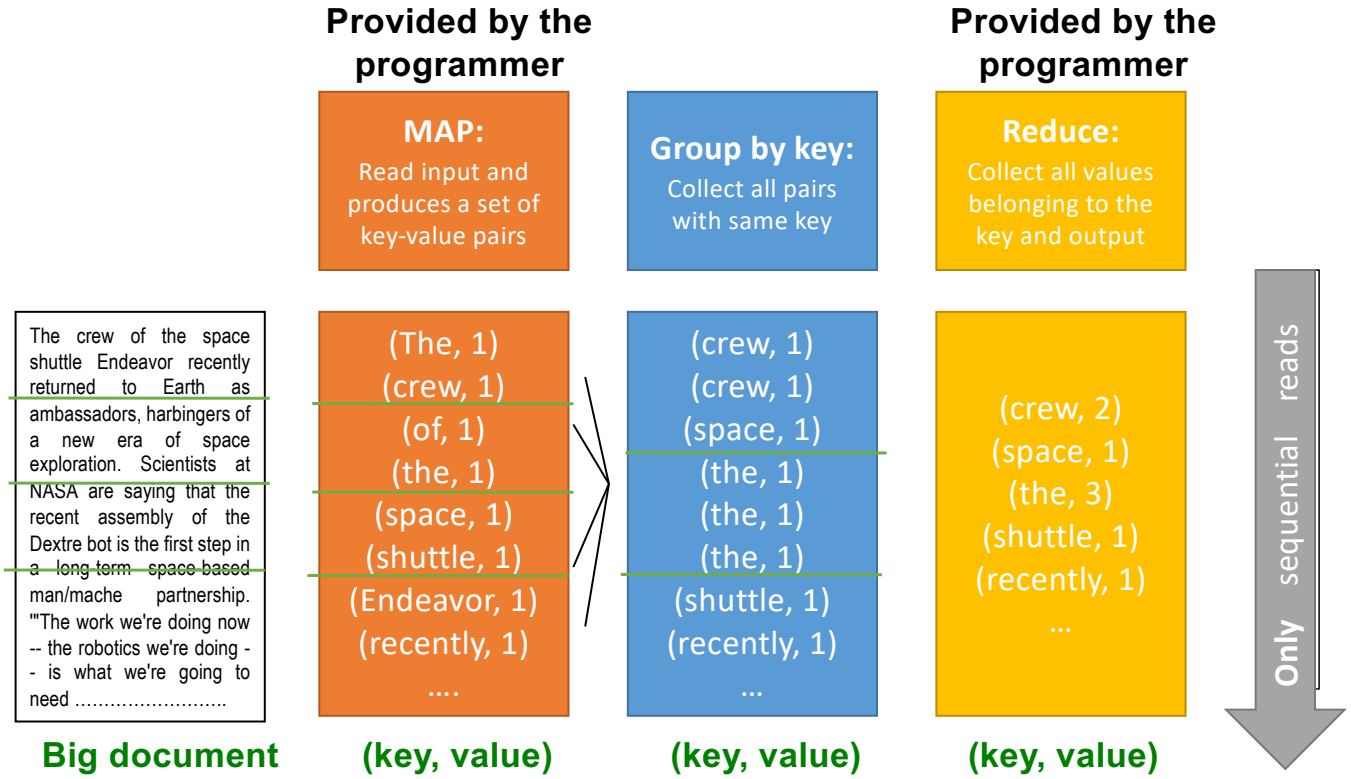
# Example 2 – Word Count

# Programming Model: MapReduce

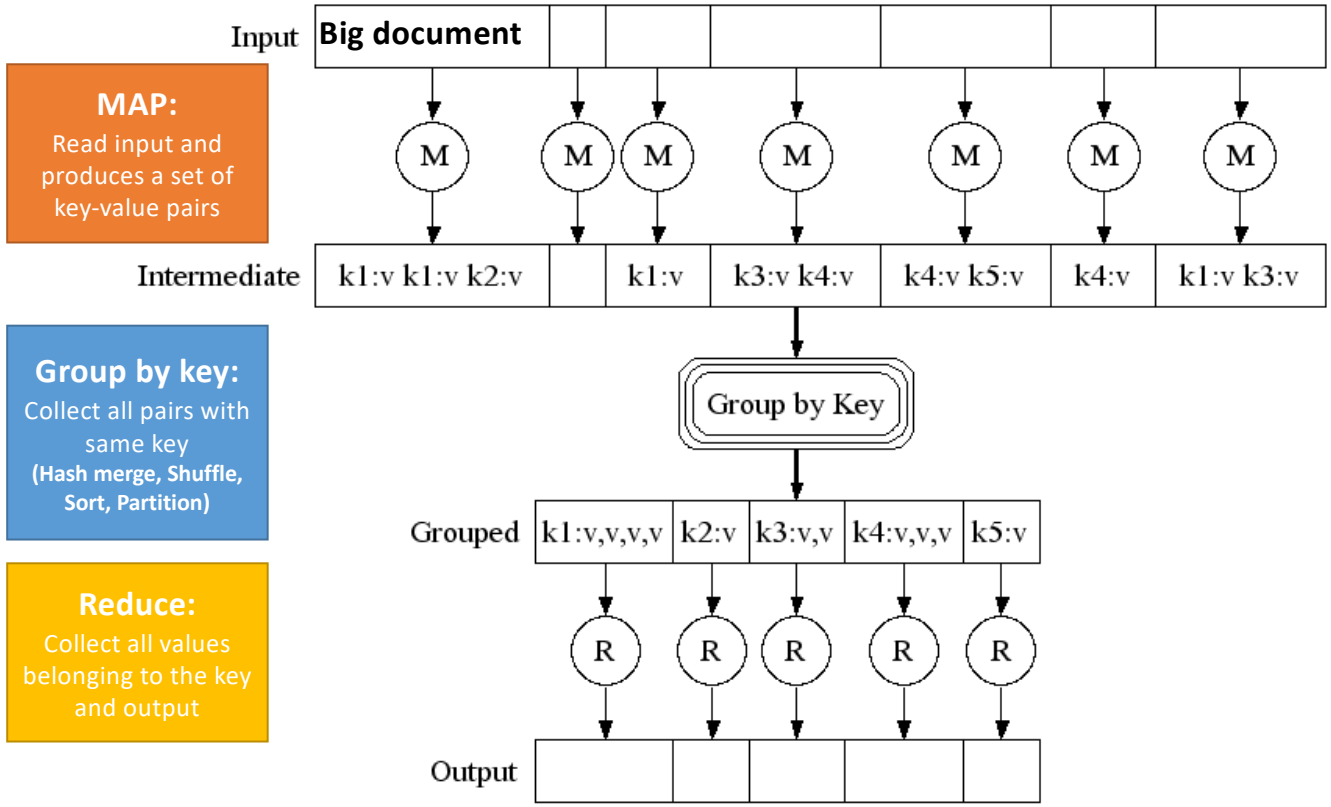
**Warm-up task:**

- We have a huge text document
- Count the number of times each distinct word appears in the file
- **Sample application:**
  - Analyze web server logs to find popular URLs

# MapReduce: Word Counting



# Map-Reduce: A diagram



# Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the document
  for each word w in value:
      emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

# Map-Reduce: Environment

**Map-Reduce environment takes care of:**

- Partitioning the input data (input splits)
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

# Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- **Workflow:**
  - Read inputs as a set of key-value-pairs
  - **Map** transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs to output nodes
  - All k'v'-pairs with a given k' are sent to the same **reduce**
  - **Reduce** processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work