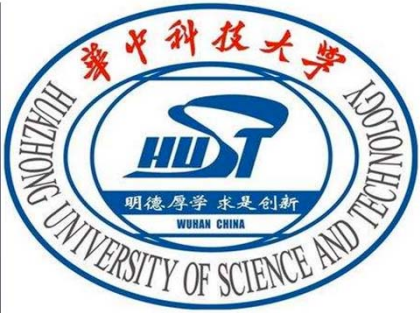
Chapter 2:
Map-Reduce and the New Software Stack

崔金华

邮箱：jhcui@hust.edu.cn
主页: https://csjhcui.github.io/
办公地址: 东湖广场柏景阁1单元1568 室



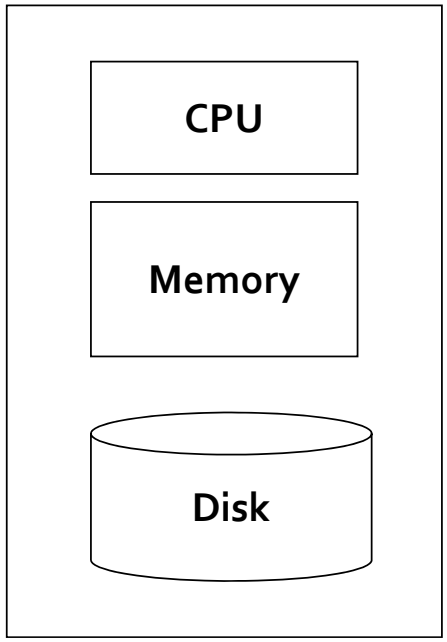
CONTENTS

- Distribute File System
- Computational Model
- Scheduling and Data Flow
- Refinements

MapReduce

- Much of the course will be devoted to large scale computing for data mining
- Challenges:
  - How to distribute computation?
  - Distributed/parallel programming is hard
- Map-reduce addresses all of the above
  - Google's computational/data manipulation model
  - Elegant way to work with big data

Single Node Architecture



Machine Learning, Statistics

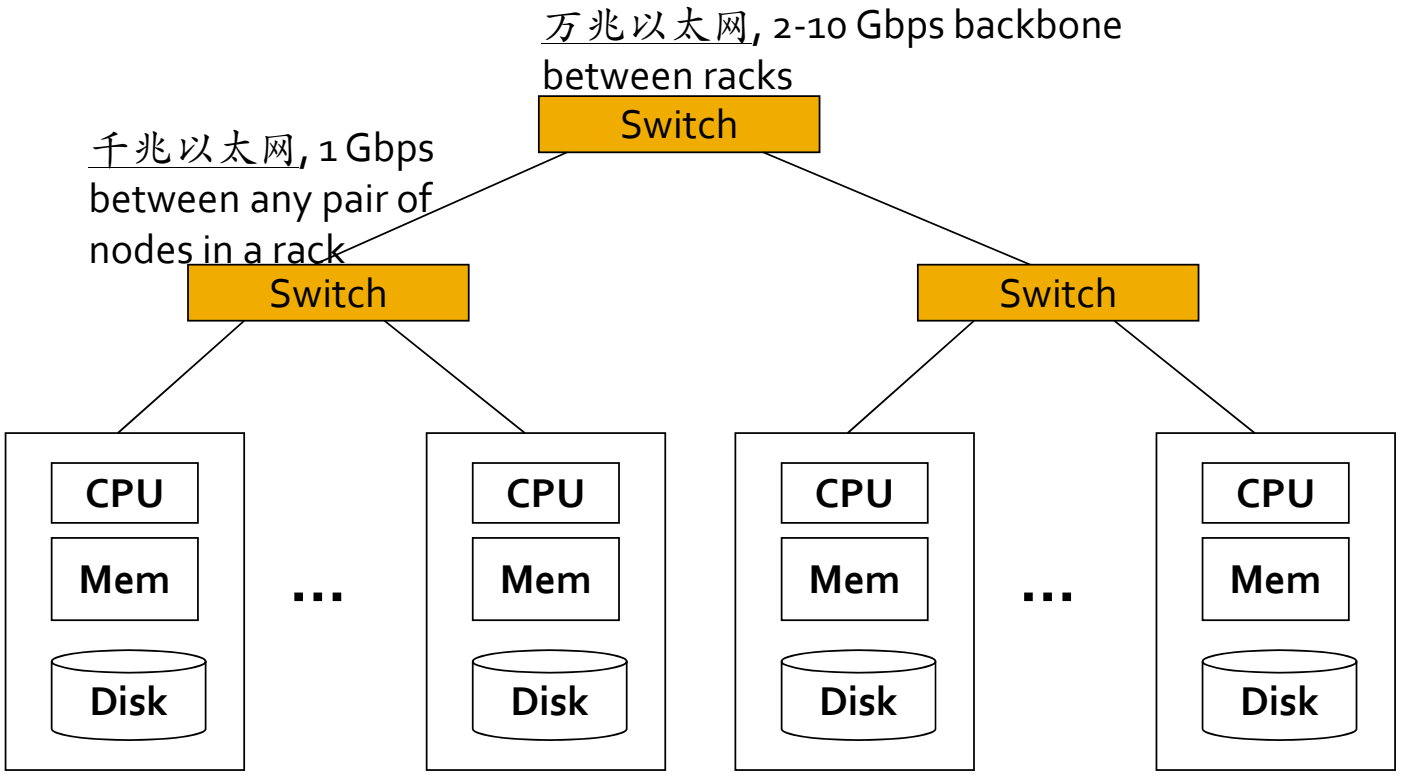
"Classical" Data Mining

Big→ Not Enough!

Motivation: Google Example

- 200亿网页 x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web (**unacceptable!**)
- ~1,000 hard drives to store the web ← then, 2 hours to read the web
- Takes even more to do something useful with the data!
- Today, a standard architecture for such problems is emerging:
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them

Cluster Architecture



Each rack contains 16-64 nodes

In 2011 it was guestimated that Google had 100万台 machines, http://bit.ly/Shh0RO

Large-scale Computing

- Large-scale computing for data mining problems on commodity hardware
- Challenges:
  - 1、Network bottleneck. How do you distribute computation?
  - 2、How can we make it easy to write distributed programs?
  - 3、Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
    - People estimated Google had ~1M machines in 2011
      - 1,000 machines fail every day!

Idea and Solution

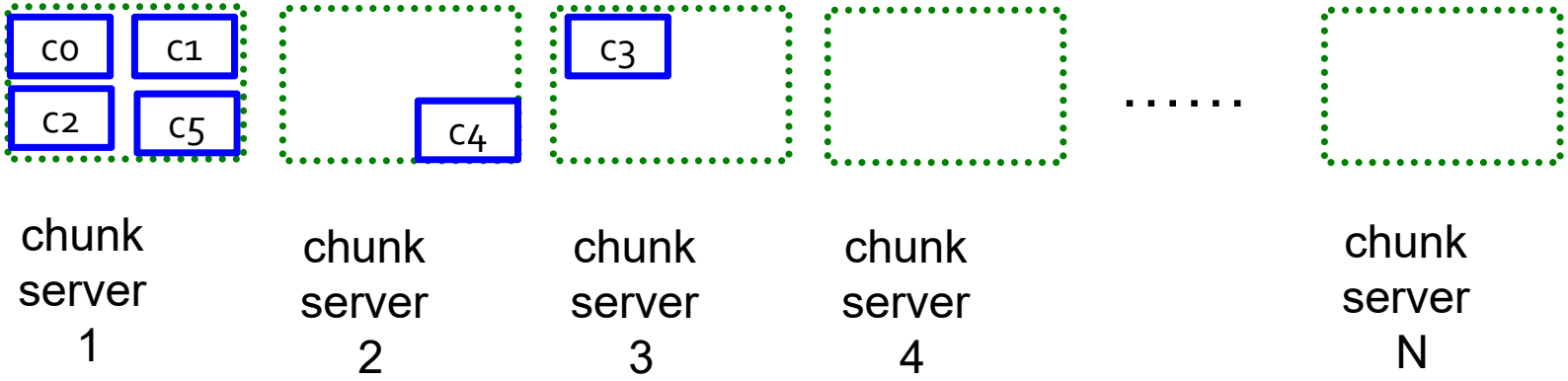
- Map-Reduce addresses these problems
  - Store data redundantly: Store files multiple times for reliability
  - Bring computation close to data: minimize data movement
  - Simple programming model: Map-Reduce, hide the complexity of distributed programs

Storage Infrastructure

- Problem:
  - If nodes fail, how to store data persistently?
- Answer:
  - Distributed File System:
    - Provides global file namespace
    - E.g., Google GFS; Hadoop HDFS;
- Typical usage pattern
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

Distributed File System

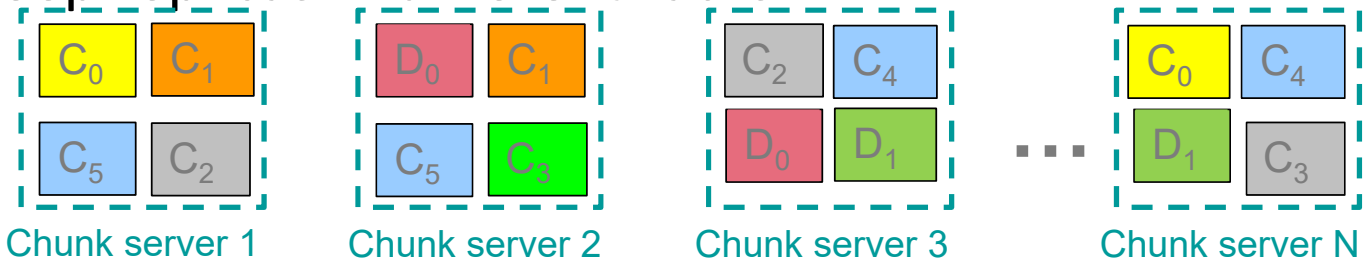
- Chunk servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB



Don't put all your eggs into one basket.

Distributed File System

- Chunk servers
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks



Bring computation directly to the data!

Chunk servers also serve as compute servers

Distributed File System

- Master node
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- Client library for file access
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

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

Task: Word Count

- Case 1:
  - File too large for memory, but all <word, count> pairs fit in memory
  - Method: HashTable
- Case 2:
  - Even the <word, count> pairs do not fit in memory
  - Count occurrences of words: words(doc.txt) | sort | uniq -c
    - where words takes a file and outputs the words in it, one per a line
- Case 2 captures the essence of MapReduce
  - Great thing is that it is naturally parallelizable

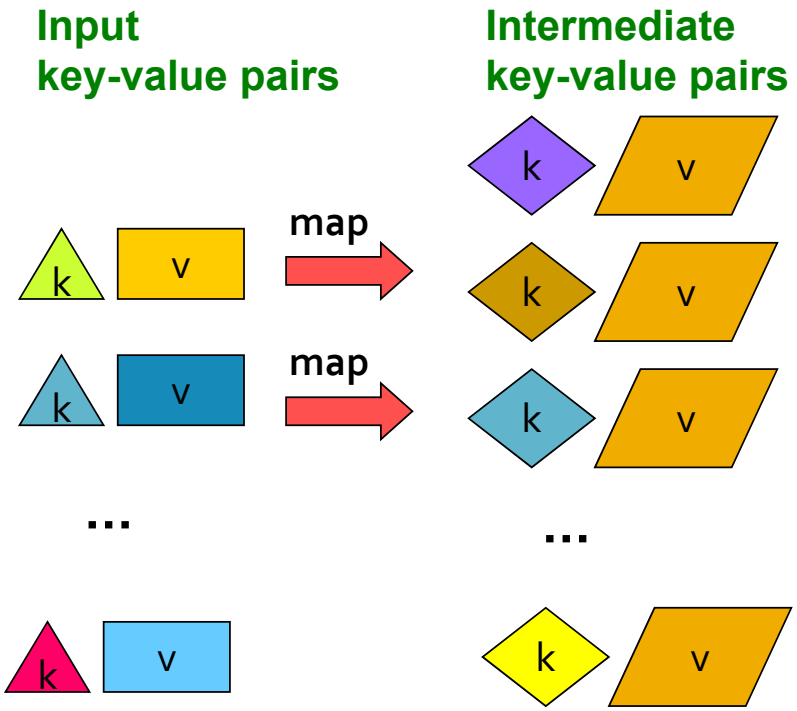
MapReduce: Overview

words(doc.txt) | sort | uniq -c

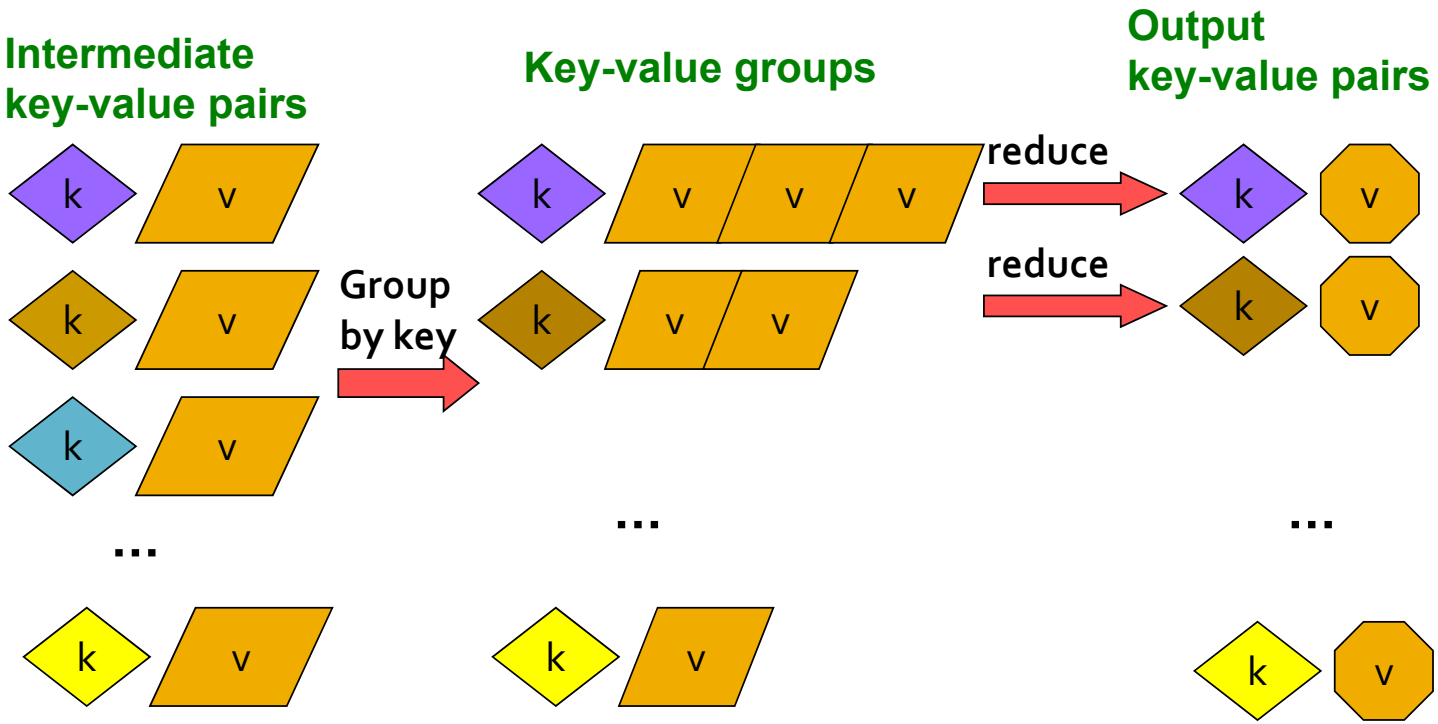
- Map:
  - Scan input file record at a time
  - Extract something you care about
- Group by key: Sort and Shuffle
- Reduce:
  - Aggregate, summarize, filter or transform
  - Write the result

Outline stays the same, **Map** and **Reduce** change to fit the problem

MapReduce: The Map Step



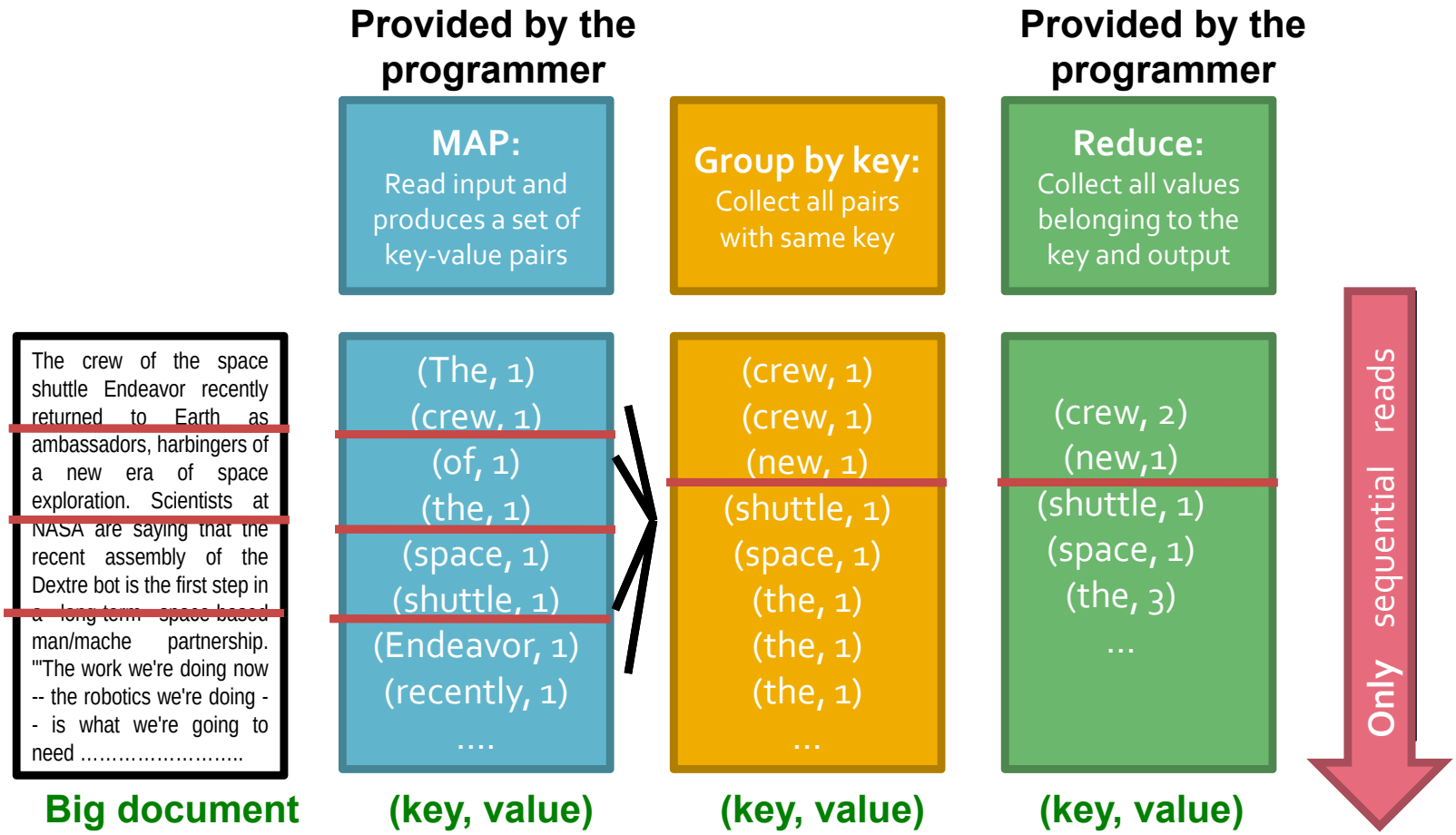
MapReduce: The Reduce Step



More Specifically

- Input: a set of key-value pairs
- Programmer specifies two methods:
  - Map(k, v) → <k', v'>*
    - Takes a key-value pair and outputs a set of key-value pairs
      - E.g., key is the filename, value is a single line in the file
    - There is one Map call for every (k,v) pair
  - Reduce(k', <v'>*) → <k', v''>*
    - All values v' with same key k' are reduced together and processed in v' order
    - There is one Reduce function call per unique key k'

MapReduce: Word Counting



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

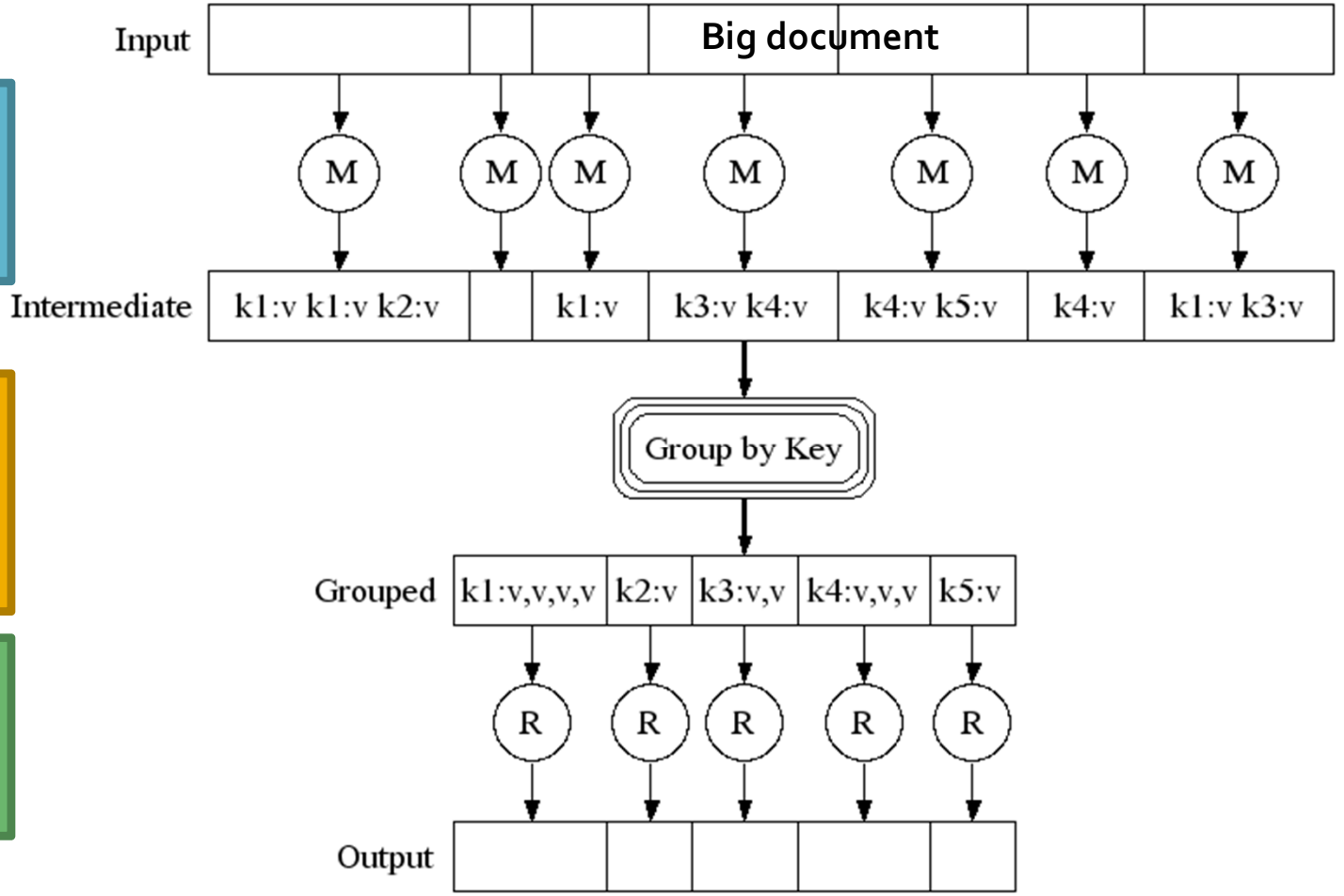
- Map-Reduce environment takes care of:
  - Partitioning the input data
  - Scheduling the program's execution across a set of machines
  - Performing the group by key
  - Handling machine failures
  - Managing required inter-machine communication

Map-Reduce: A diagram

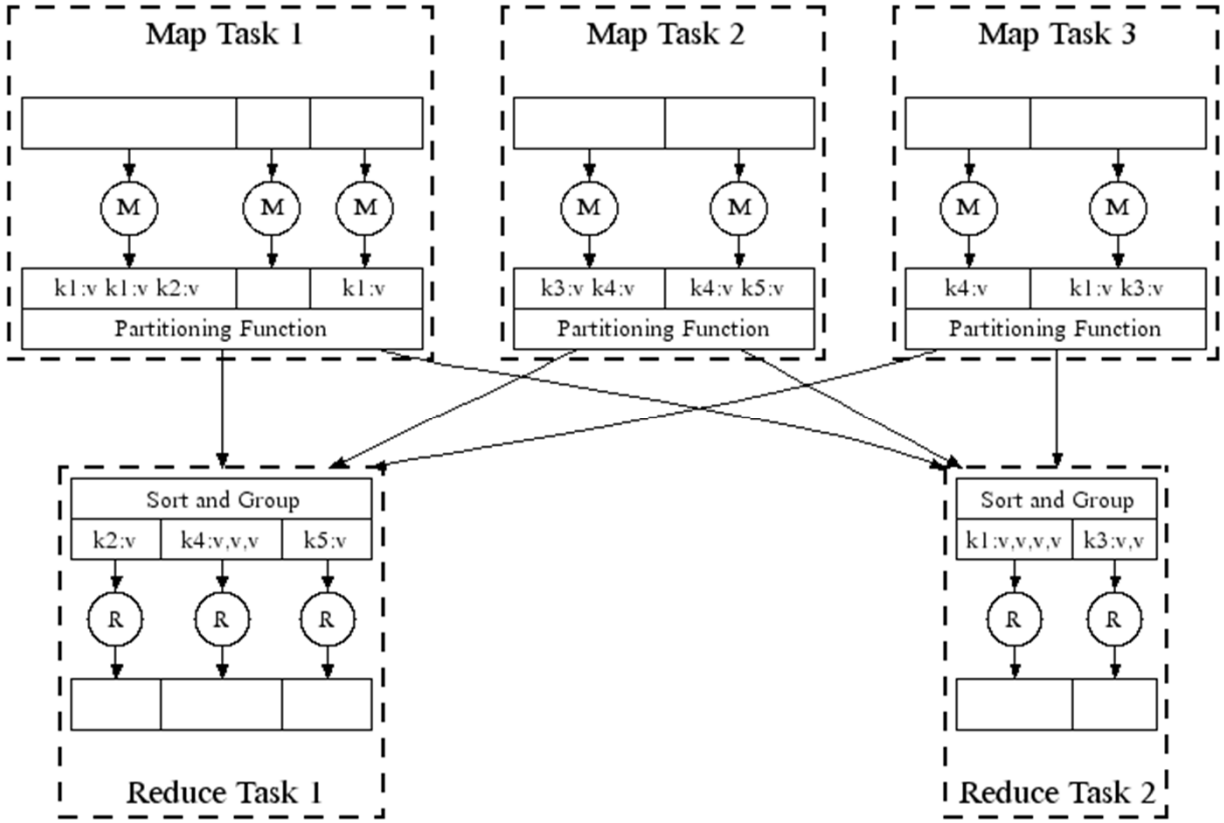
MAP:
Read input and produces a set of key-value pairs

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

Reduce:
Collect all values belonging to the key and output



Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

Data Flow

- Input and final output are stored on a distributed file system (FS):
  - 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

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.
  - How to deal with failures?