



# Introduction to Data Mining

## Lecture #3: MapReduce-1

**U Kang**  
**Seoul National University**



# Outline

➔ ☐ MapReduce

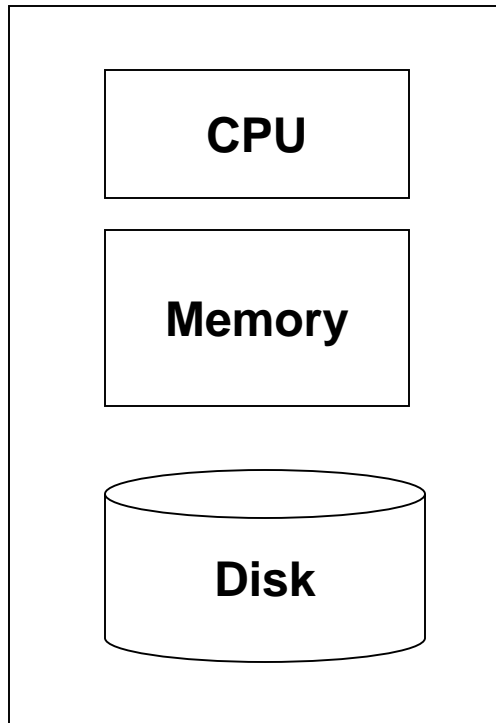


# MapReduce

- 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**



# Motivation: Google Example

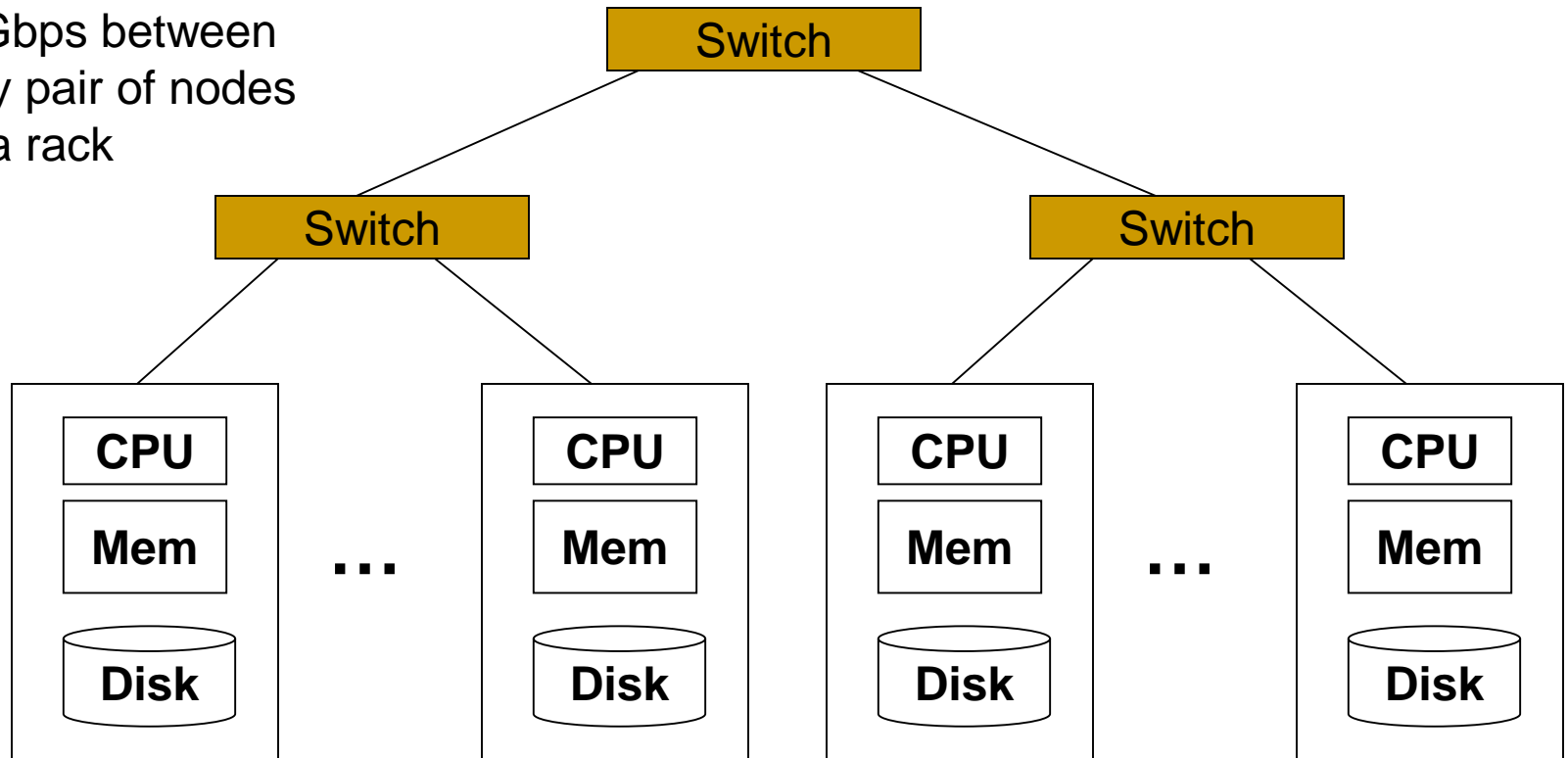
- 20+ billion web pages x 20KB = 400+ TB
- 1 computer reads 30-35 MB/sec from disk
  - ~4 months to read the web
- ~1,000 hard drives to store the web
- Takes even more to **do something useful with the data!**
- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity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



# Cluster Architecture

2-10 Gbps backbone between racks

1 Gbps between any pair of nodes in a rack



Each rack contains 16-64 nodes

In 2011, Google used ~1M machines, <http://bit.ly/Shh0RO>





# Large-scale Computing

- **Large-scale computing for data mining problems on commodity hardware**
- **Challenges:**
  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - 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

- **Issue:** Copying data over a network takes time
- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Map-reduce addresses these problems**
  - Google's computational/data manipulation model
  - Elegant way to work with big data
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop (open source): HDFS
  - **Programming model**
    - Map-Reduce



# Storage Infrastructure

## ■ Problem:

- If nodes fail, how to store data persistently?

## ■ Answer:

### □ **Distributed File System (DFS):**

- Provides global file namespace
- 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
- ❑ Each chunk replicated (usually 2x or 3x)
- ❑ Try to keep replicas in different racks

## ■ **Master node**

- ❑ a.k.a. Name Node in Hadoop's HDFS
- ❑ Stores metadata about where files are stored
- ❑ Might be replicated

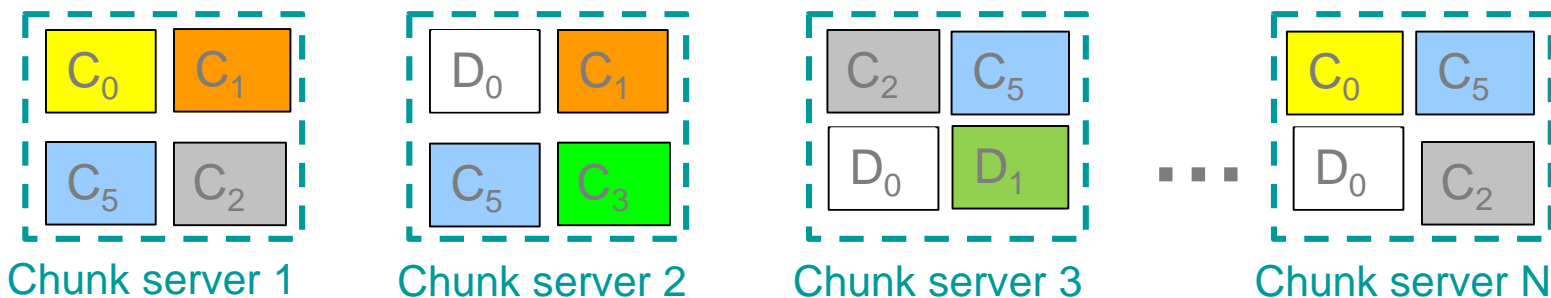
## ■ **Client library for file access**

- ❑ Talks to master to find chunk servers
- ❑ Connects to chunk servers to access data



# Distributed File System

- **Reliable distributed file system**
- Data kept in “chunks” spread across machines
- Each chunk **replicated** on different machines
  - Seamless recovery from disk or machine failure



Bring computation to the data!  $\Leftrightarrow$  data to computer

Chunk servers also serve as compute servers



# 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

```
9/1/99, 10:46:11, 1578, 509, 5397, 200, 0, GET, /cfdocs/akonline/paintbrush.JPG, -,
9/1/99, 10:46:49, 37703, 577, 24402, 200, 0, GET, /cfdocs/akonline/email_book.cfm,
tfirstname=francis&tlastname=smitt&tid=270&PASSWORD=teachme&USERNAME=francis,
9/1/99, 10:49:11, 181500, 579, 114331, 200, 0, GET, /cfdocs/akonline/update_table.cfm,
tfirstname=francis&tlastname=smitt&tid=270&PASSWORD=teachme&USERNAME=francis,
9/1/99, 10:52:04, 354641, 662, 163301, 200, 64, GET, /cfdocs/AKONLINE/assess/PAT11B.pdf
tfirstname=francis&tlastname=smitt&tid=270&PASSWORD=teachme&USERNAME=francis,
```



# Task: Word Count

## Case 1 (easy)

- ❑ File too large for memory, but all <word, count> pairs fit in memory

## Case 2 (hard)

- ❑ Both file and all <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

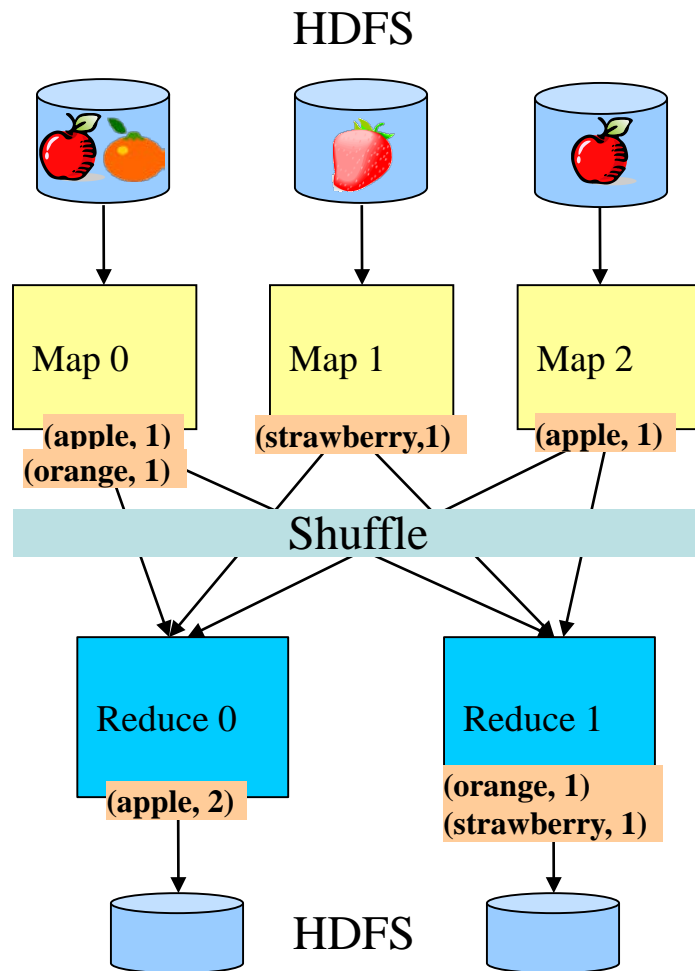
- Sequentially read a lot of data
- **Map:**
  - Extract something you care about
- **Group by key:** Sort and Shuffle
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

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



# MapReduce: Example

- Assume the words are fruit names



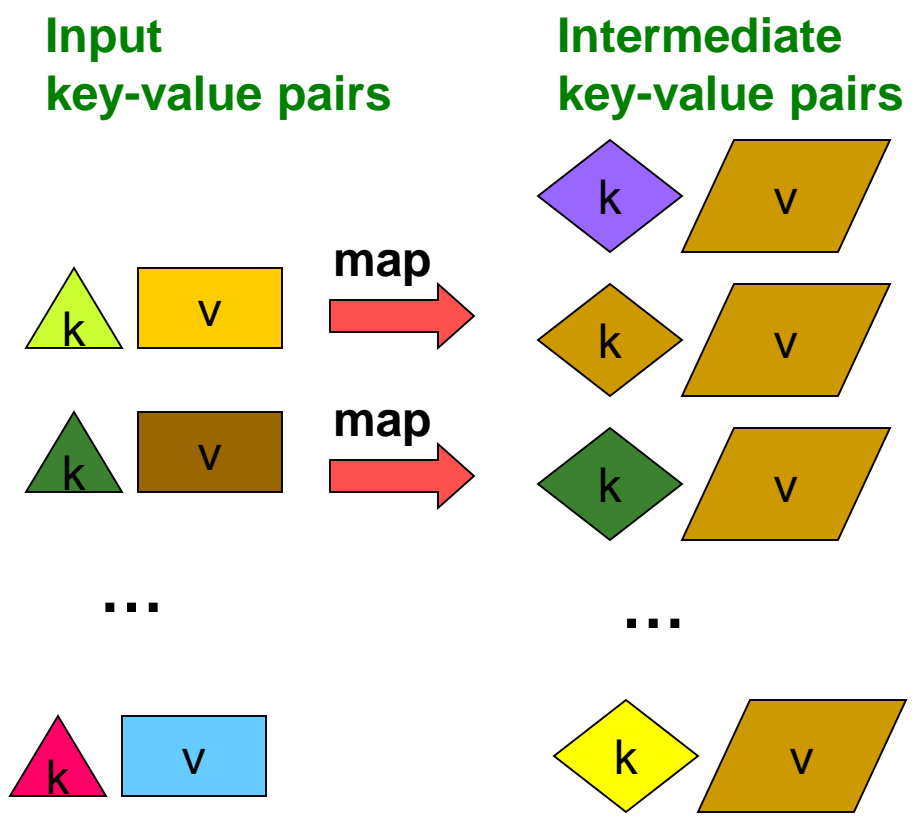
```
map( fruit ) {  
    output(fruit, 1);  
}
```

```
reduce( fruit, v[1..n] ) {  
    for(i=1; i <=n; i++)  
        sum = sum + v[i];  
    output(fruit, sum);  
}
```



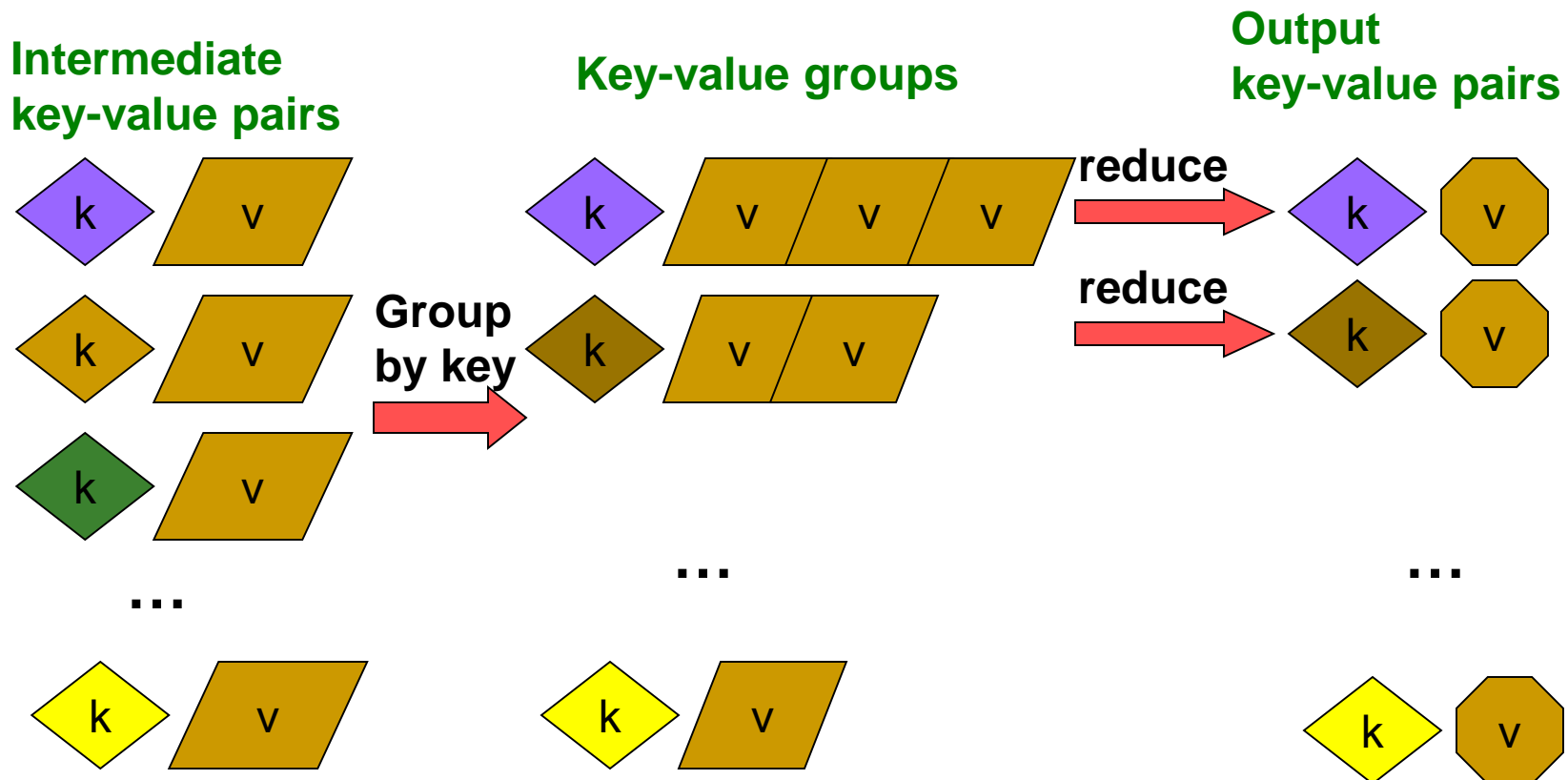


# MapReduce: The Map Step





# MapReduce: The Reduce Step





# More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map( $k, v$ )**  $\rightarrow \langle k', v' \rangle^*$ 
    - 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 function call for every  $(k, v)$  pair (input)
  - **Reduce( $k', \langle v' \rangle^*$ )**  $\rightarrow \langle k', v'' \rangle^*$ 
    - All values  $v'$  with same key  $k'$  are reduced together and processed
    - There is one Reduce function call per unique key  $k'$



# MapReduce: Word Counting

Provided by the programmer

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

**Group by key:**  
Collect all pairs with same key

Provided by the programmer

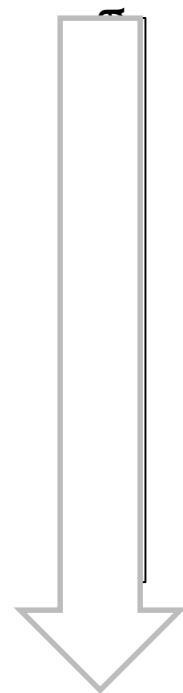
**Reduce:**  
Collect all values belonging to the key and output

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long term space based man/mache partnership. "The work we're doing now -- the robotics we're doing - is what we're going to need .....

(The, 1)  
(crew, 1)  
(of, 1)  
(the, 1)  
(space, 1)  
(shuttle, 1)  
(Endeavor, 1)  
(recently, 1)  
....

(crew, 1)  
(crew, 1)  
(space, 1)  
(the, 1)  
(the, 1)  
(the, 1)  
(shuttle, 1)  
(recently, 1)  
...

(crew, 2)  
(space, 1)  
(the, 3)  
(shuttle, 1)  
(recently, 1)  
...



Big document

(key, value)

(key, value)

(key, value)



# 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** step
- Handling machine failures
- Managing required inter-machine communication

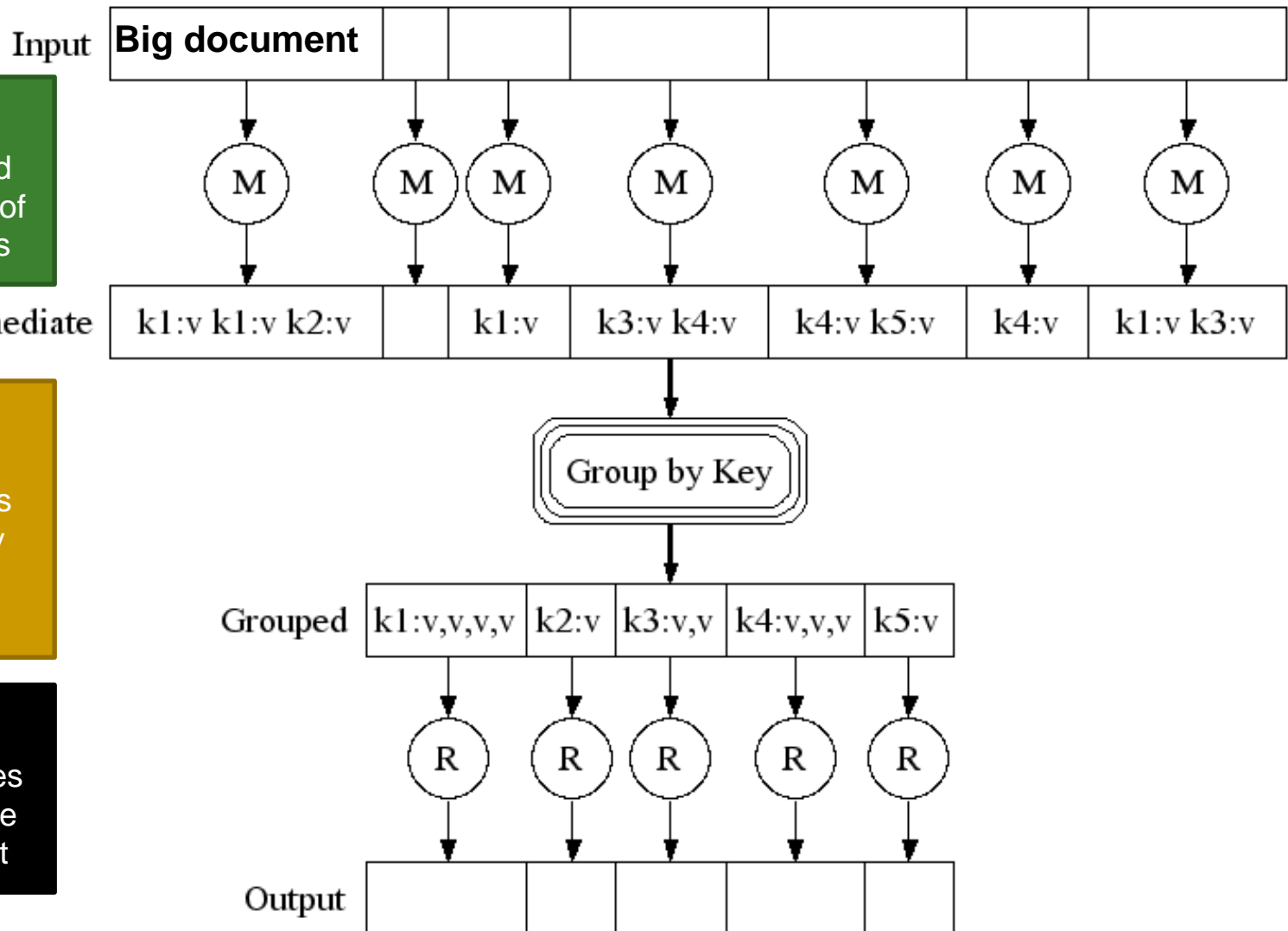


# Map-Reduce: A diagram

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

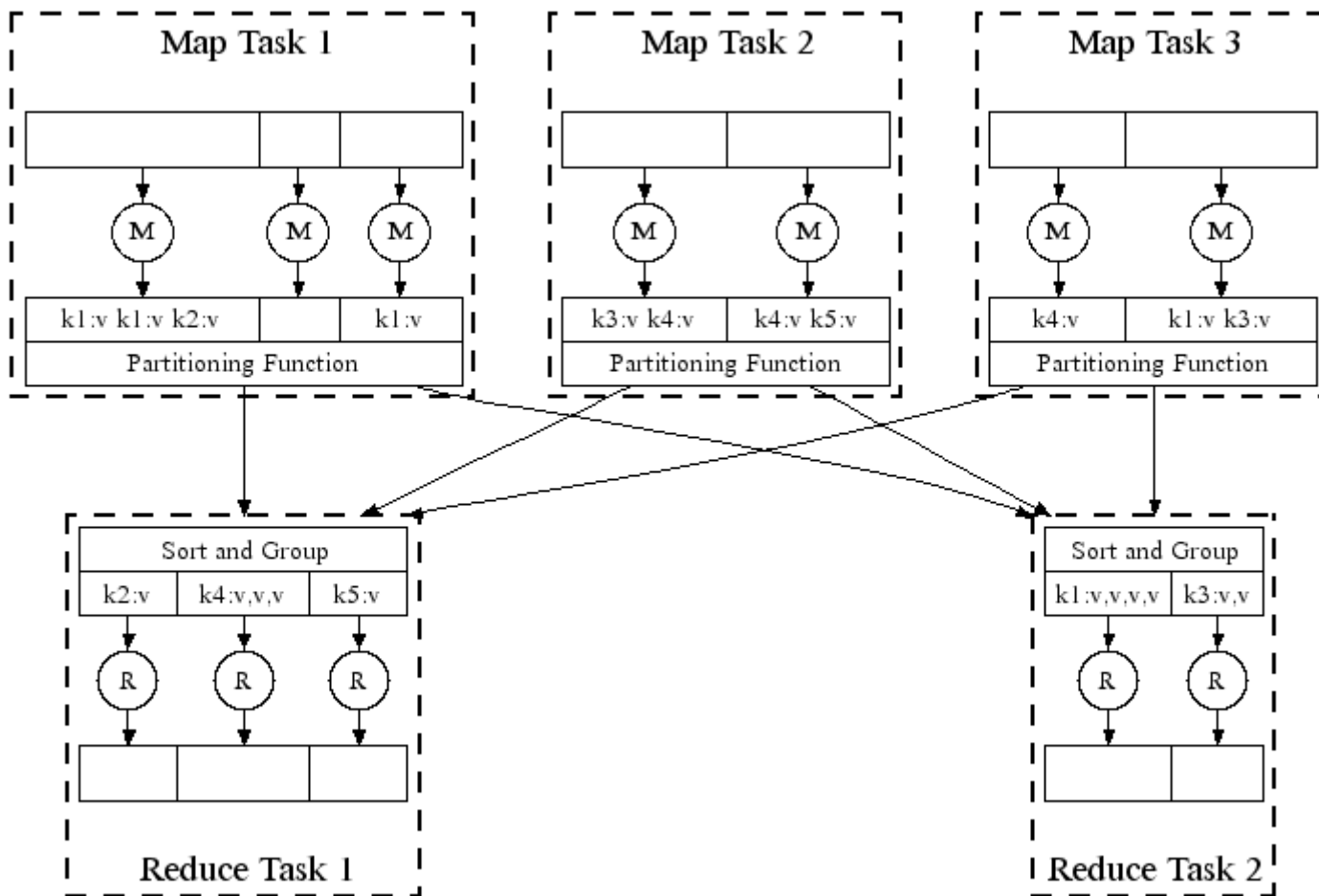
**Group by key:**  
Collect all pairs with same key (Hash merge, Shuffle, Sort, Partition)

**Reduce:**  
Collect all values belonging to the key and output





# Map-Reduce: In Parallel



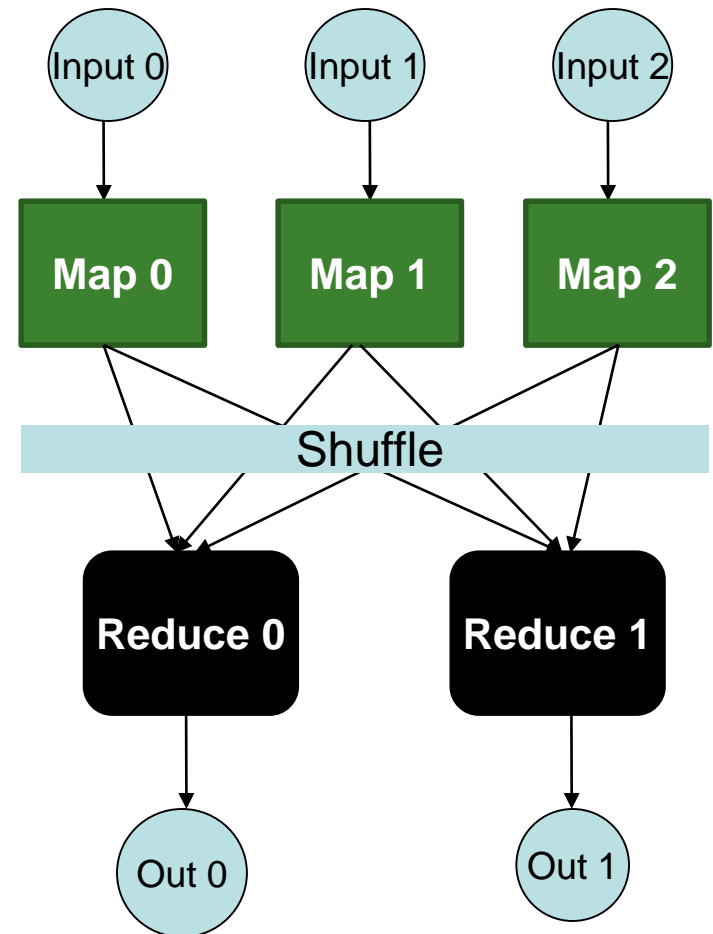
All phases are distributed with many tasks doing the work





# 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





# 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



# Dealing with Failures

## ■ Map worker failure

- ❑ Map tasks completed or in-progress at worker are reset to idle
- ❑ Reduce workers are notified when task is rescheduled on another worker

## ■ Reduce worker failure

- ❑ Only in-progress tasks are reset to idle
- ❑ Reduce task is restarted on another worker

## ■ Master failure

- ❑ MapReduce task is aborted and client is notified



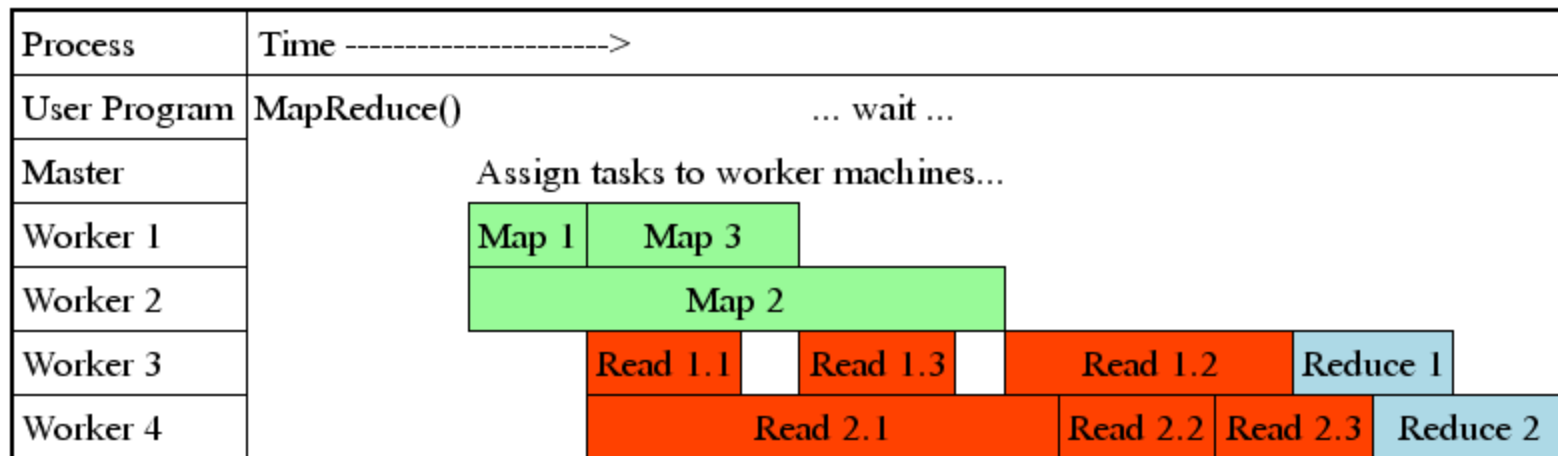
# How many Map and Reduce jobs?


- $M$  map tasks,  $R$  reduce tasks
- **Rule of a 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 (each reduce task creates one file in DFS)



# Task Granularity & Pipelining

- **Fine granularity tasks:** map tasks  $\gg$  machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing



 : shuffle

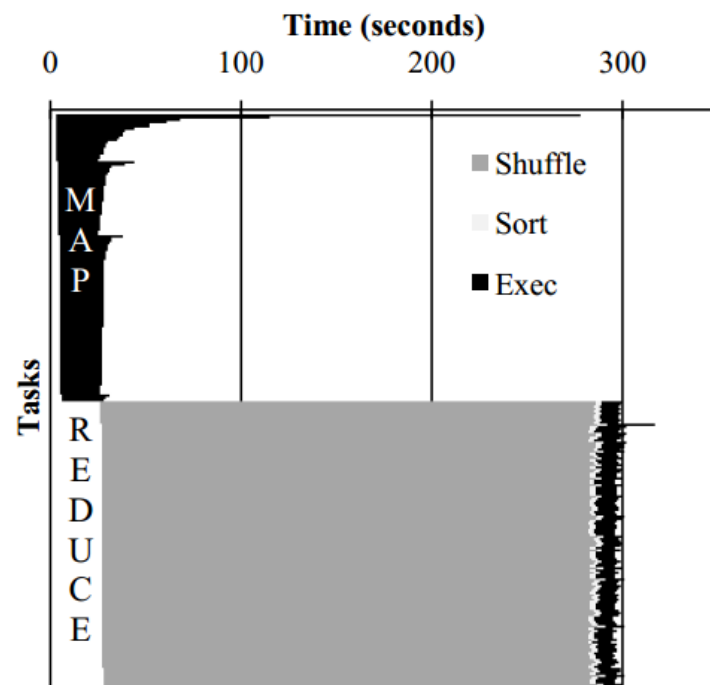


# Refinements: Backup Tasks

## ■ Problem

□ Slow workers significantly lengthen the job completion time:

- Other jobs on the machine
- A machine may be slow
- Bad disks
- Weird things



[YongChul Kwon et al, SkewTune: Mitigating Skew in MapReduce Applications, SIGMOD 2012]



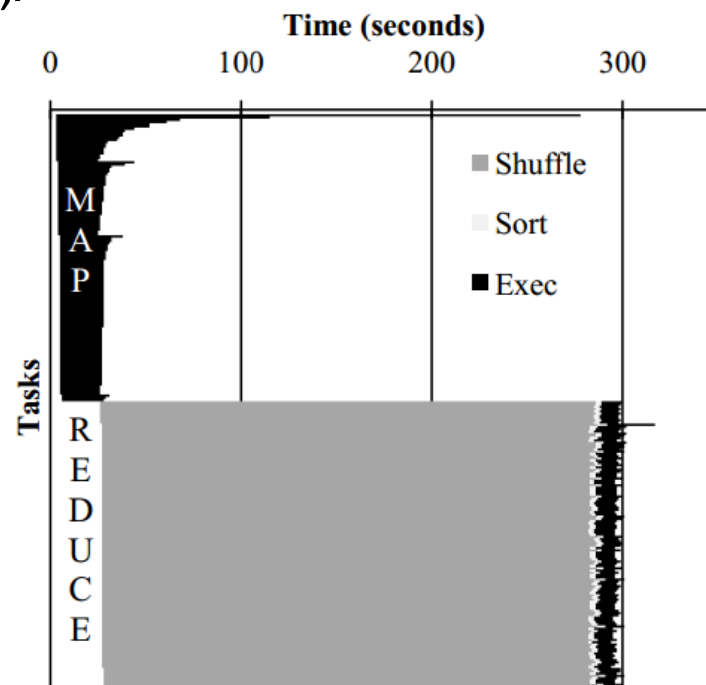
# Refinements: Backup Tasks

## ■ Solution

- Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first “wins”

## ■ Effect

- Shortens job completion time



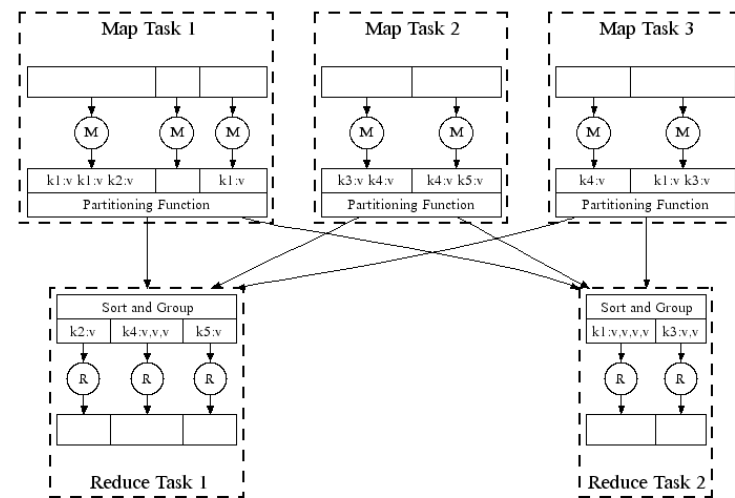
[YongChul Kwon et al, SkewTune: Mitigating Skew in MapReduce Applications, SIGMOD 2012]





# Refinement: Combiners

- Often a Map task will produce many pairs of the form  $(k, v_1), (k, v_2), \dots$  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**
  - $\text{combine}(k, \text{list}(v_1)) \rightarrow (k, v_2)$
  - Combiner is usually the same as the reduce function
- Works only if reduce function is commutative and associative





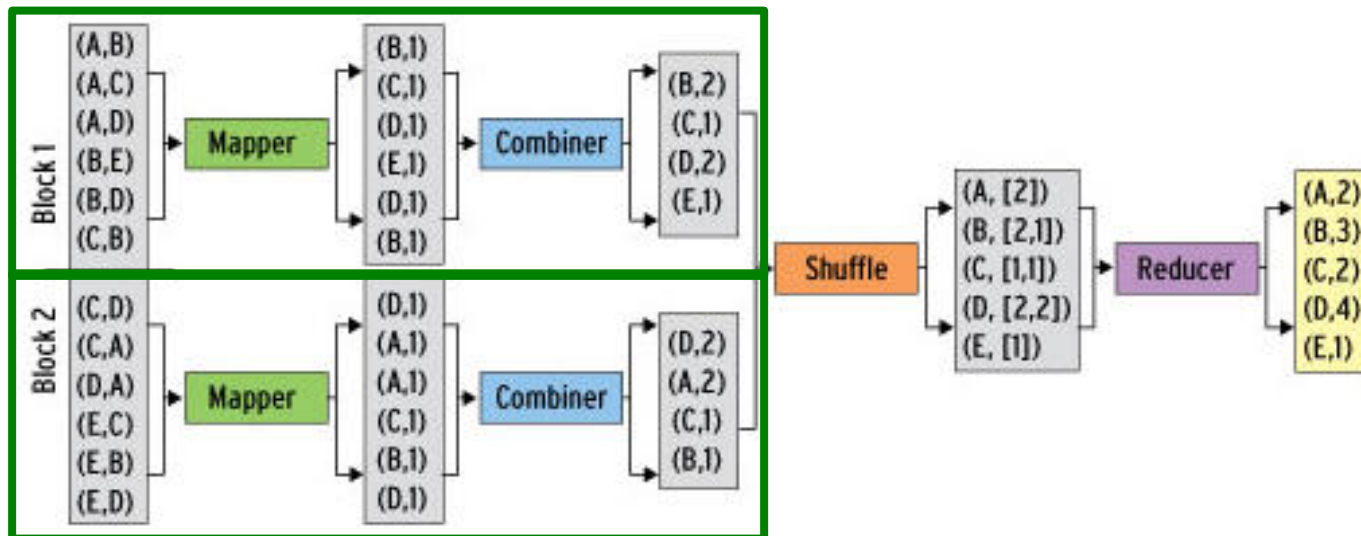
# Refinement: Combiners

- Works only if reduce function is commutative and associative
  - Commutative (e.g.  $x * y = y * x$ )
  - Associative (e.g.  $(x * y) * z = x * (y * z)$ )
- When shouldn't we use? Examples?



# Refinement: Combiners

- **Back to our word counting example:**
  - Combiner combines the values of all keys of a single machine:



- Much less data need to be copied and shuffled!



# Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **Partition function: key  $\rightarrow$  reduce task id**
- **System uses a default partition function:**
  - **$\text{hash}(\text{key}) \bmod R$**  (R: # of reduce tasks)
- **Sometimes useful to override the hash function:**
  - E.g.,  **$\text{hash}(\text{hostname}(\text{URL})) \bmod R$**  ensures URLs from a host end up in the same output file



# Conclusion

- MapReduce : a simplified model for large scale computation
  - Hides the details of parallelization, fault-tolerance, data distribution, and load balancing
- Used widely in industry as well as academia



# Questions?