Lecture #5

云计算入门
Introduction to Cloud Computing
GESC1001

Philippe Fournier-Viger
Professor
School of Humanities and Social Sciences
philfv8@yahoo.com

Fall 2020

Course schedule

Part 1	Introduction and overview
Part 2	Distributed and parallel systems
Part 3	Cloud infrastructure
Part 4	Cloud application paradigm (1)
Part 5	**Cloud application paradigm (2)**
Part 6	Cloud virtualization and resource management
Part 7 & 8	Cloud computing storage systems Cloud computing security
	Final exam

Introduction

Last week:

- Review
- **Chapter 4**: Cloud application paradigm (part 1)

Today:

- **Chapter 4**: Cloud applications (part 2) – the *Map Reduce* model
- Assignment 1

How to ask questions

We can discuss immediately after lectures

You may use the QQ group to contact teaching assistants

My e-mail: philfv8@yahoo.com

4-CLOUD APPLICATIONS (云应用) PART 2

Introduction

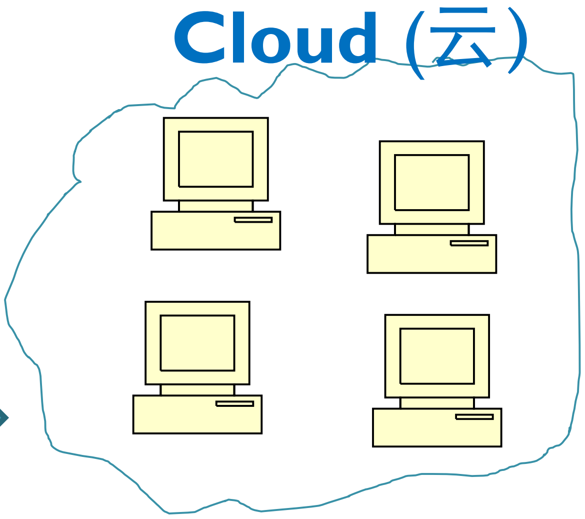
- We discussed challenges for developing cloud applications
- Today, we will talk about the details of how cloud applications are created.
- To make **cloud applications**, the **MapReduce model** is very popular.
- It is a "**programming model**" (编程模型 - a way of developing applications for the cloud).
- It was proposed by **Google** in a research paper, published in 2004.

MapReduce: Simplified Data Processing on Large Clusters,Jeffrey Dean and Sanjay Ghemawat, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

Introduction

Why MapReduce is popular?

- Because it is a simple programming model.
- A **programmer（程序员）** can easily write an **application** that run on a distributed system (the cloud), without much experience about how distributed systems work.



Introduction

- One of the most popular version of **MapReduce** is **Hadoop**.
- It is an **open-source** (开放源码) implementation of **MapReduce**.
- I will explain the main idea.
- We will also discuss **three examples**.

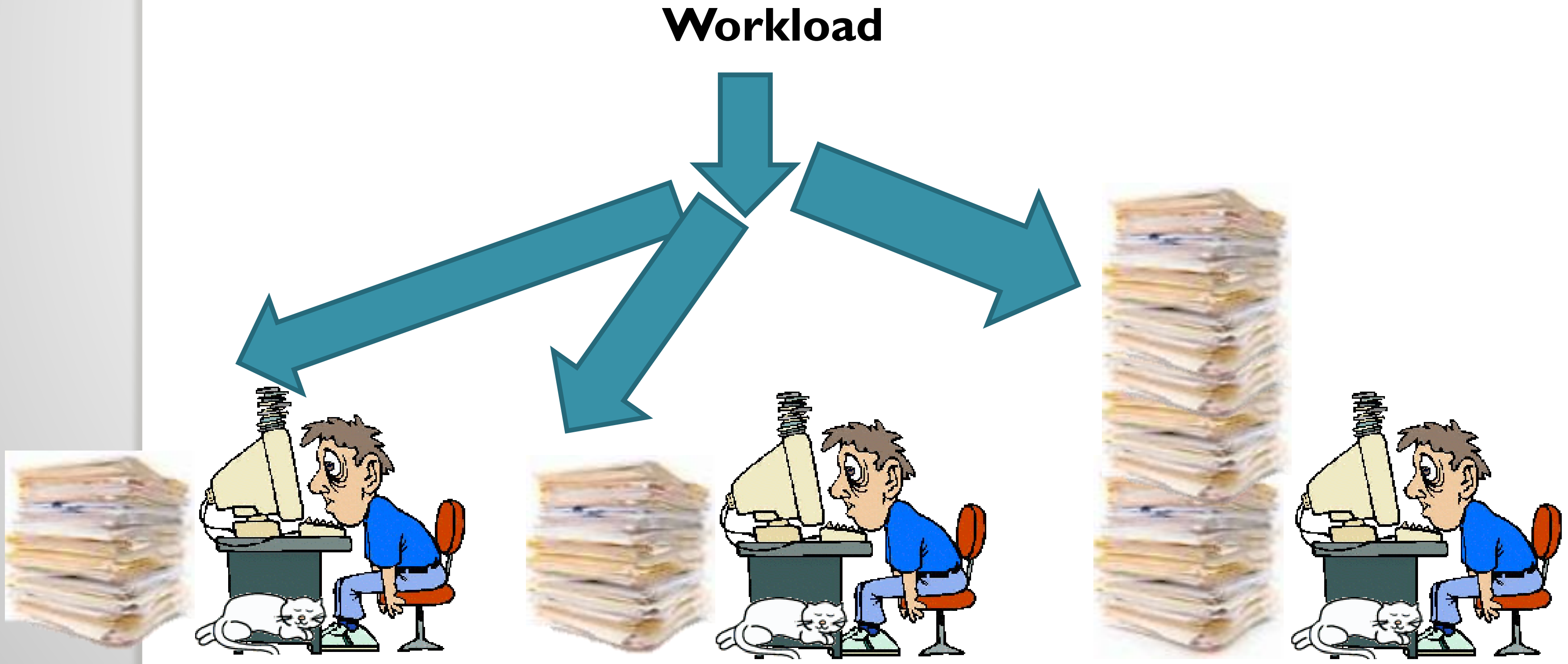
http://hadoop.apache.org/

Introduction

- The main advantage of the cloud is **elasticity** (云的弹性) .
  - Using as many computers as needed to address the **cost** (元) and **timing** constraints of an application.
  - Sharing the **workload** (工作负载) between several computers.
  - It must be divided into **sub-tasks** that can be accomplished in parallel by several computers.
- But how to do this?

Introduction

The **workload** should be divided (分配) approximately equally between computers.



Introduction

- **Partitioning** (分配) **the workload** is not always easy.
- **Three main types of workloads**:
  - **modularly divisible** (模块化分割) **workload**: the workload is already divided into sub-tasks.
  - **arbitrarily divisible** (可任意划分) **workload**: the workload can be partitioned into an arbitrarily large number of sub-tasks of equal or similar size.
  - Others.

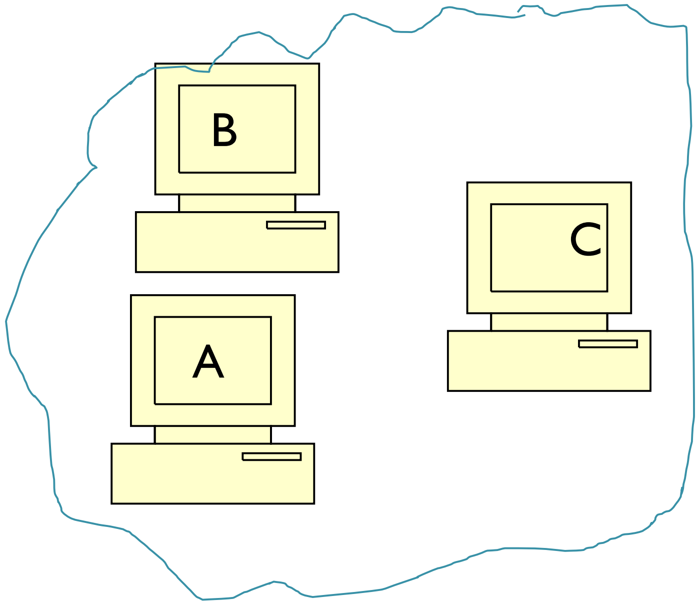
Map Reduce

- Designed for **arbitrarily divisible （可任意划分） workloads**.
- It is used to perform **parallel processing** (并行处理) for **data-intensive** (数据密集型) applications.
- It has many applications: **e.g.** physics, biology, etc.
- Once a cloud applications is created using MapReduce, it can run in the cloud on as many computers as needed.

Basic idea of MapReduce

Phase I (Map)

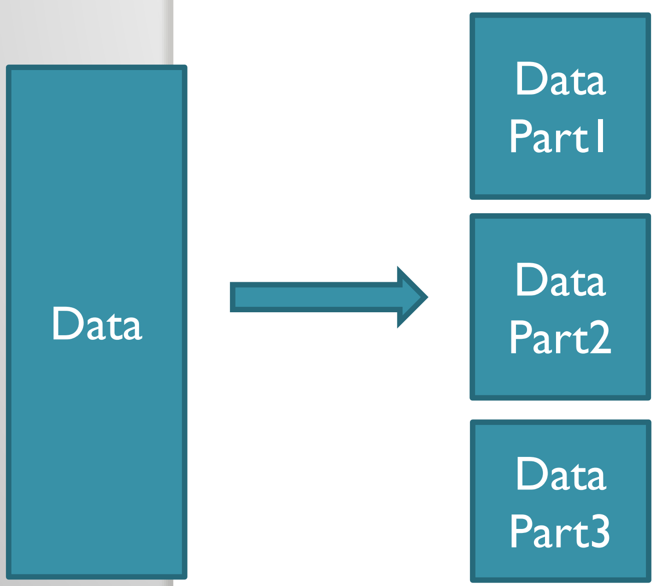
1. Split the data into blocks
2. Assign each block to an instance (实例) (**e.g.** a computer or virtual machine)
3. Run these instances in parallel

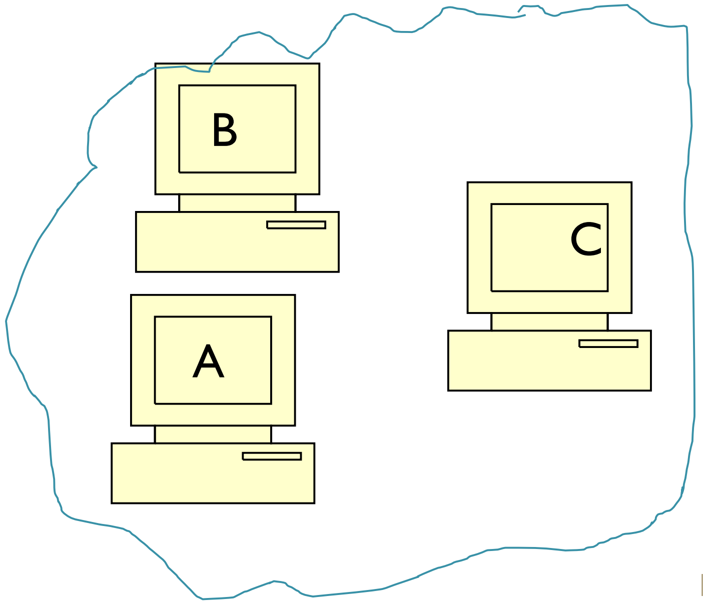


Basic idea of MapReduce

Phase 1 (Map)

1. Split the data into blocks
2. Assign each block to an instance (实例) (**e.g.** a computer or virtual machine)
3. Run these instances in parallel

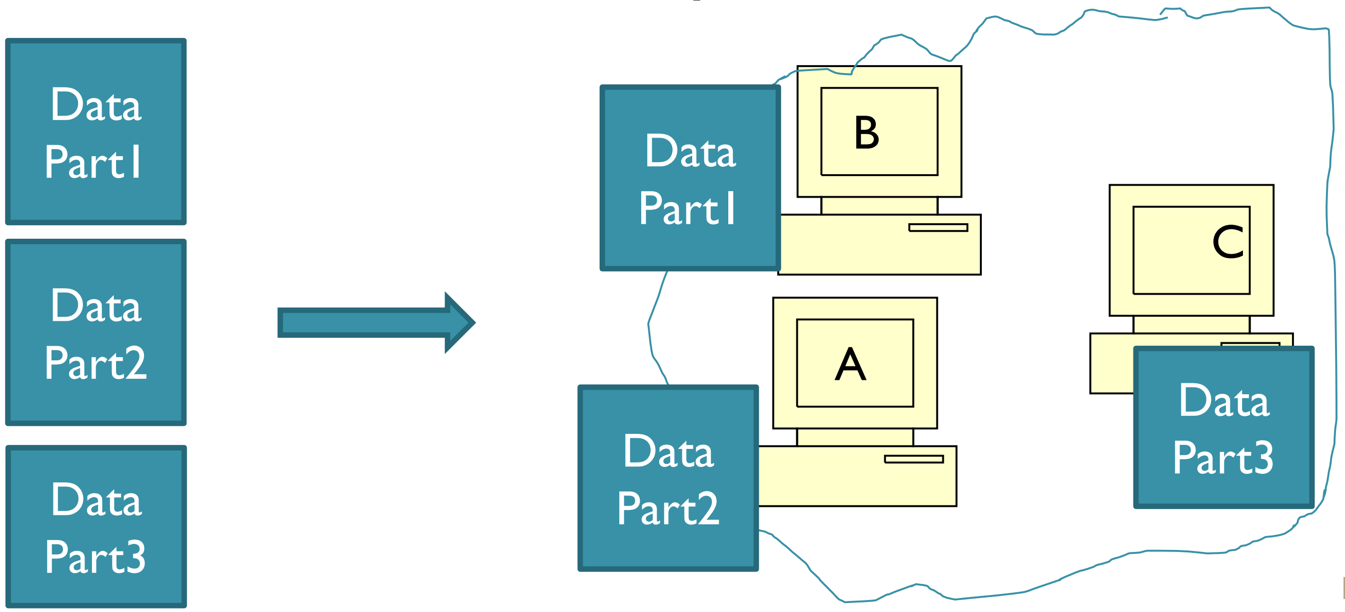




Basic idea of MapReduce

Phase 1 (Map)

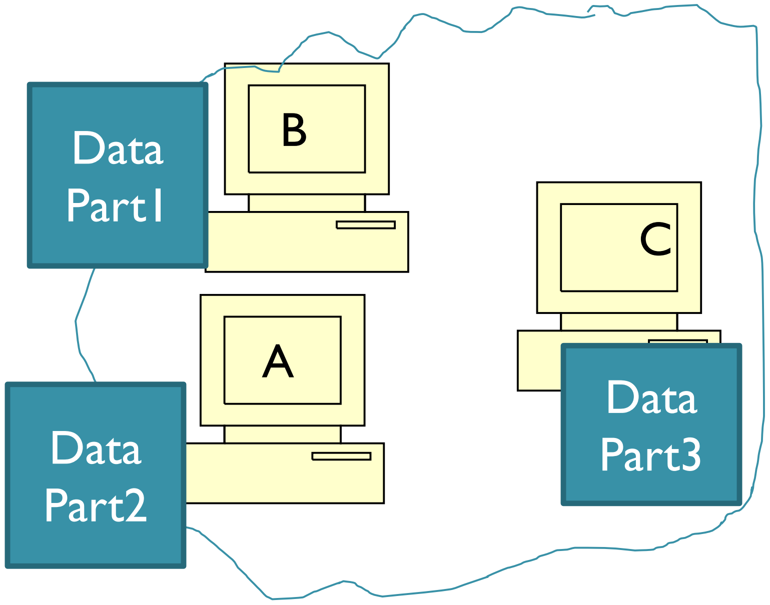
1. Split the data into blocks
2. Assign each block to an instance (实例) (**e.g.** a computer or virtual machine)
3. Run these instances in parallel



Basic idea of MapReduce

Phase 1 (Map)

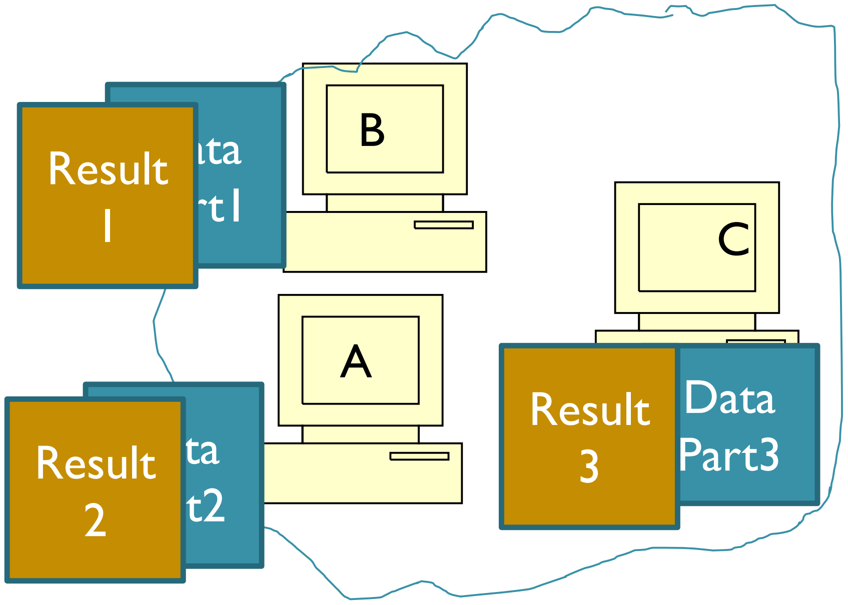
1. Split the data into blocks
2. Assign each block to an instance (实例) (**e.g.** a computer or virtual machine)
3. Run these instances in parallel



Basic idea of MapReduce

Phase 2 (Reduce)

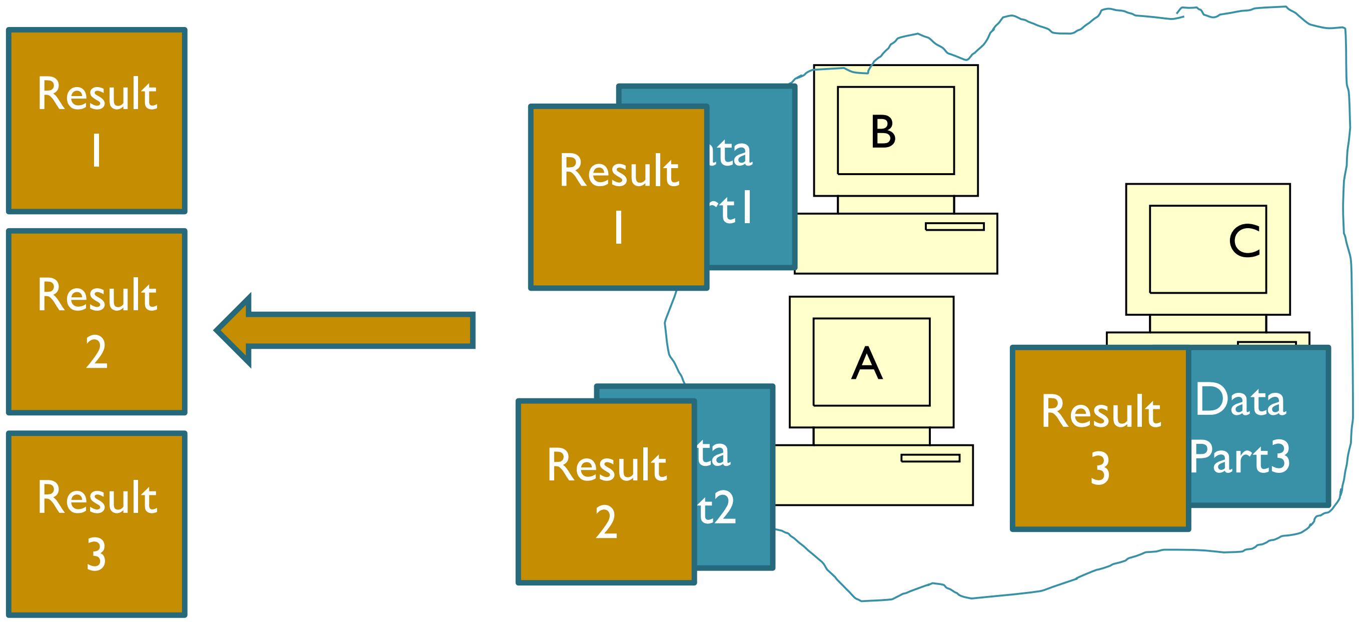
1. Once all the instances have finished their sub-tasks, they send their results.
2. Results are merged to obtain the final result.



Basic idea of MapReduce

Phase 2 (Reduce)

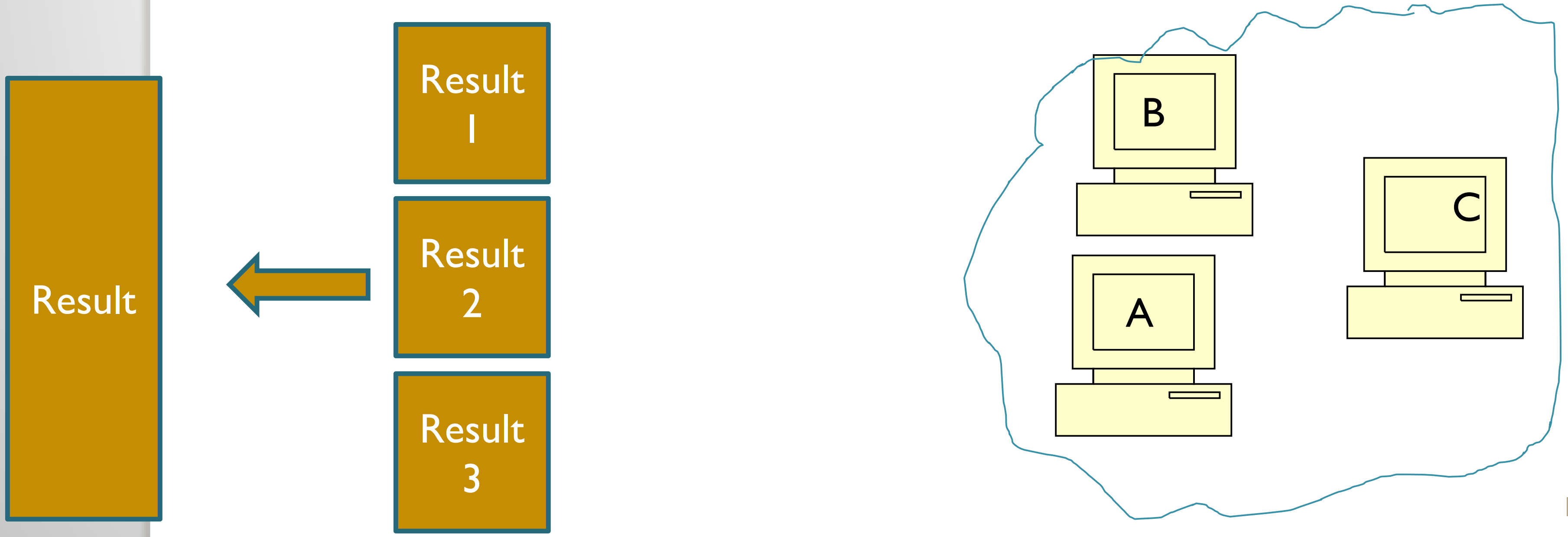
1. Once all the instances have finished their sub-tasks, they send their results.
2. Results are merged to obtain the final result.



Basic idea of MapReduce

Phase 2 (Reduce)

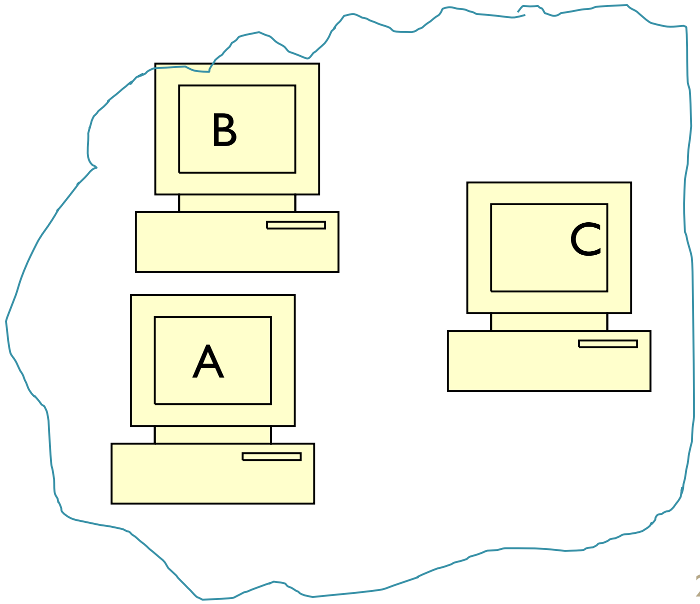
1. Once all the instances have finished their sub-tasks, they send their results.
2. Results are merged to obtain the final result.



Basic idea of MapReduce

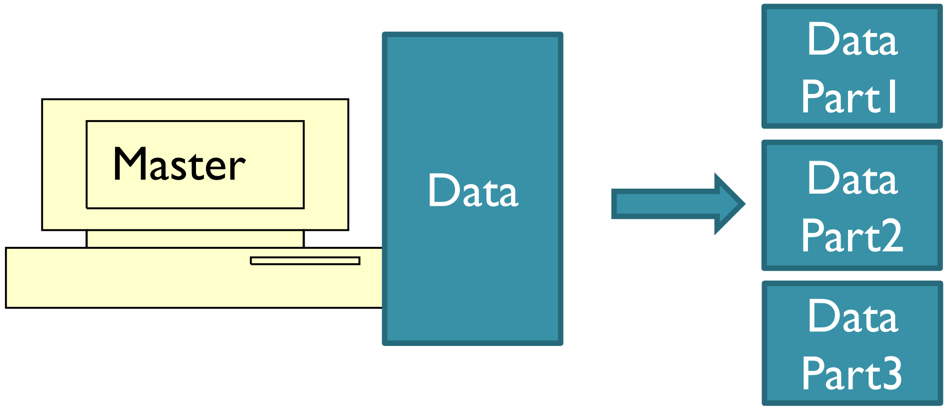
Phase 2 (Reduce)

1. Once all the instances have finished their sub-tasks, they send their results.
2. Results are merged to obtain the final result.

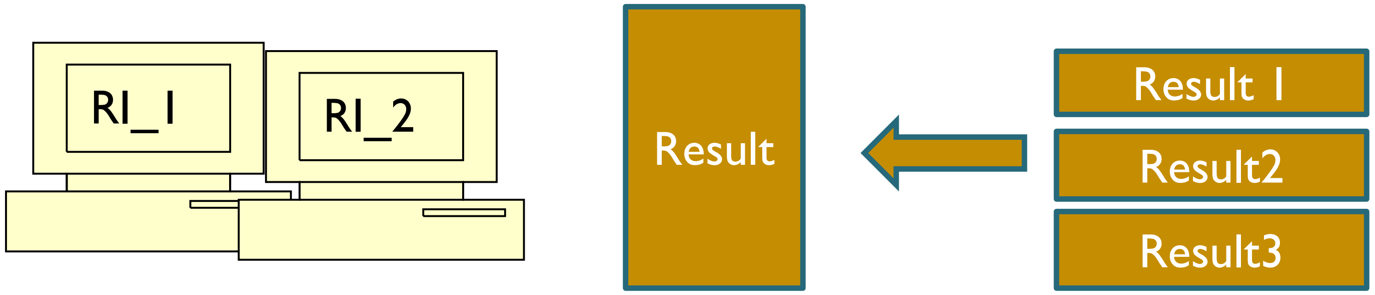


Who split the data and the gather results?

- A "**master instance**" takes care of splitting the data.



- **Merging the results** can be done by a set of instances called the "**reducing instances**"



How data is represented?

The **input data** (输入数据) can be any kind of files.

But it is converted to a set of
<key, value> **pairs** (键值对).

e.g.: (key= CN, value = Shenzhen)
(key= CN, value = Beijing)

A **key** (键) is some information that is used to group values together.

How data is represented?

The **output data（输出数据）** is a also set of <key, value> pairs.

e.g.: (key= CN, value = Shenzhen)
(key= CN, value = Beijing)
…..

A **key** (键) is some information that is used to group values together.

MapReduce

- **MapReduce** is a **programming model** (编程模型)
- It is inspired by the **Map** and the **Reduce** operations of the **LISP** programming language.
- It is designed to process **large datasets** on computing clusters (the **cloud – 云**).
- It is often used with the **Java** language.
- A programmer has to define **map()** and **reduce()** functions

A simple example

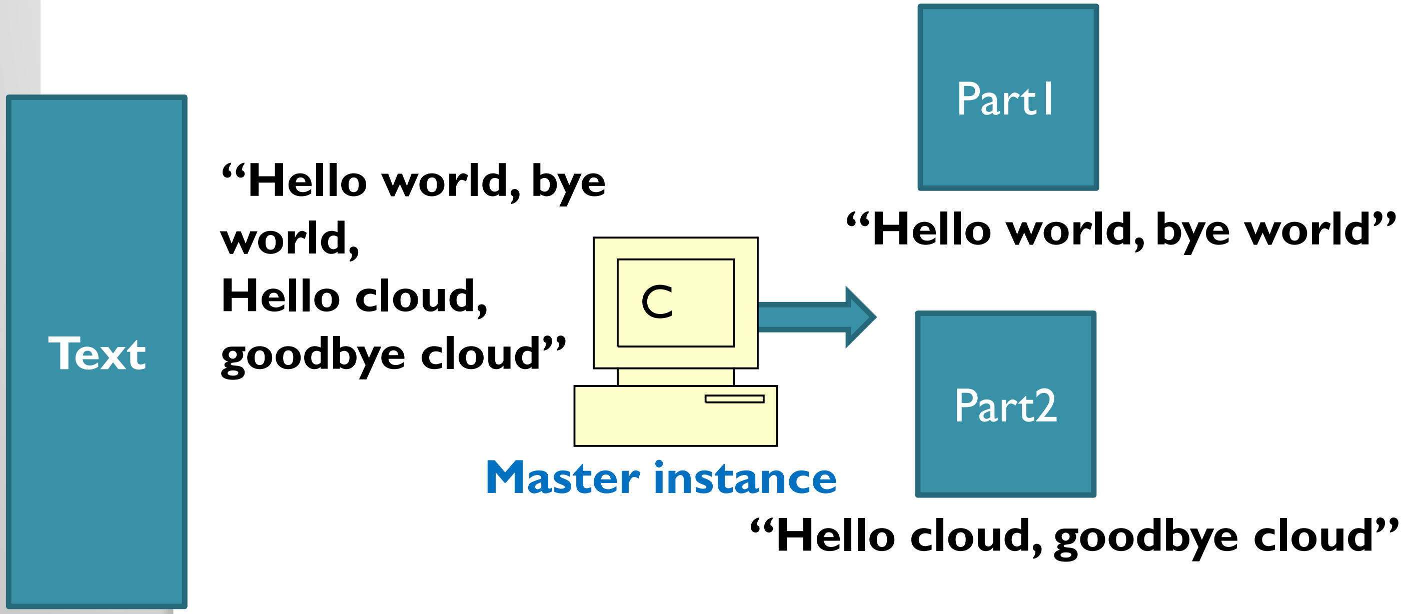
Consider that we want to count how many times each word appear in a very large text document.

Text

**“Hello world, bye world,
Hello cloud,
goodbye cloud”**

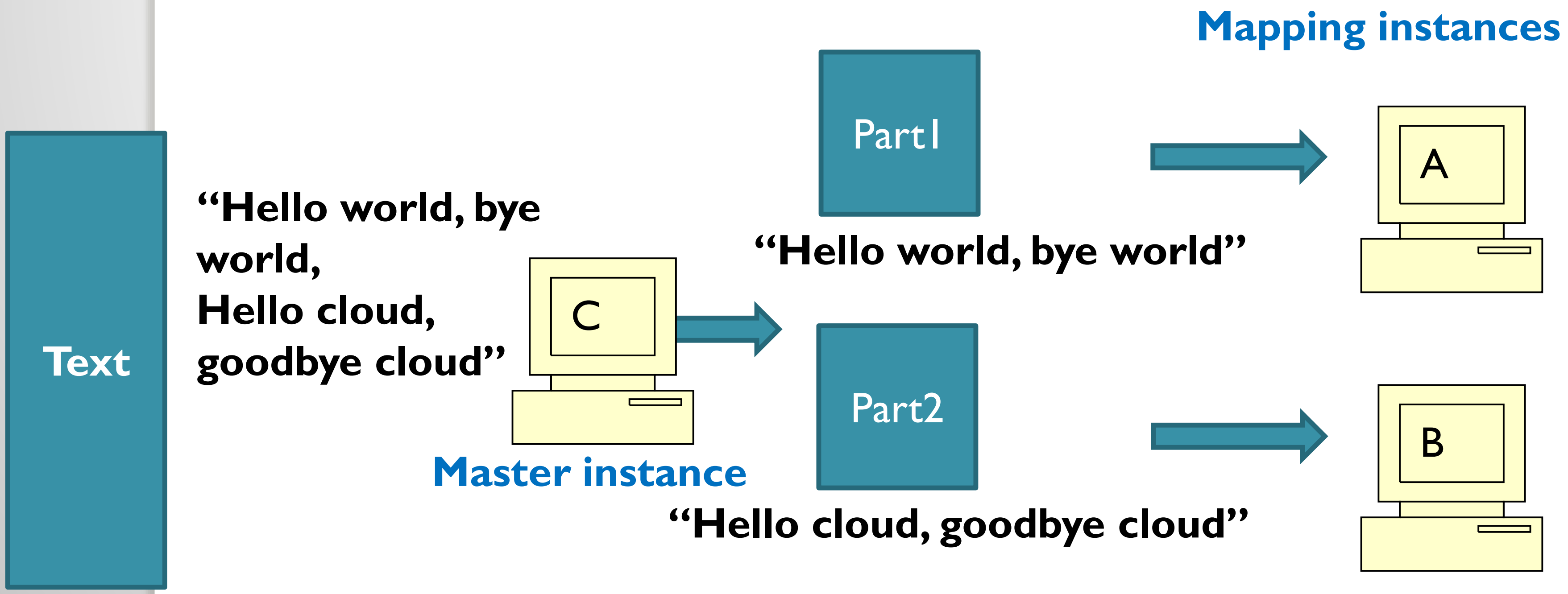
A simple example

The **master instance** first splits the data into **M** data blocks.



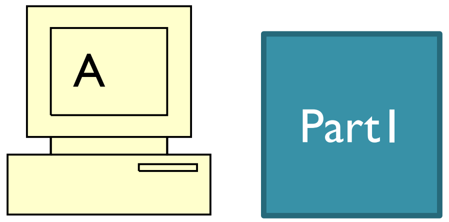
A simple example

Then, it starts **M** mapping instances and gives a data block to each instance.


Mapping instances
Text
"Hello world, bye world, Hello cloud, goodbye cloud"
C
Master instance
Part1
"Hello world, bye world"
Part2
"Hello cloud, goodbye cloud"
A
B

A simple example - map

- **All instances work in parallel.**
- **Consider the <u>first instance</u>.** It reads its data.
- It creates a <key,value> pair (键值对) for each word that it reads. A key (键) is a word and the corresponding value (值) is the number 1.

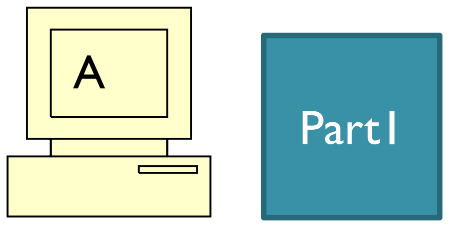


"Hello world, bye world"

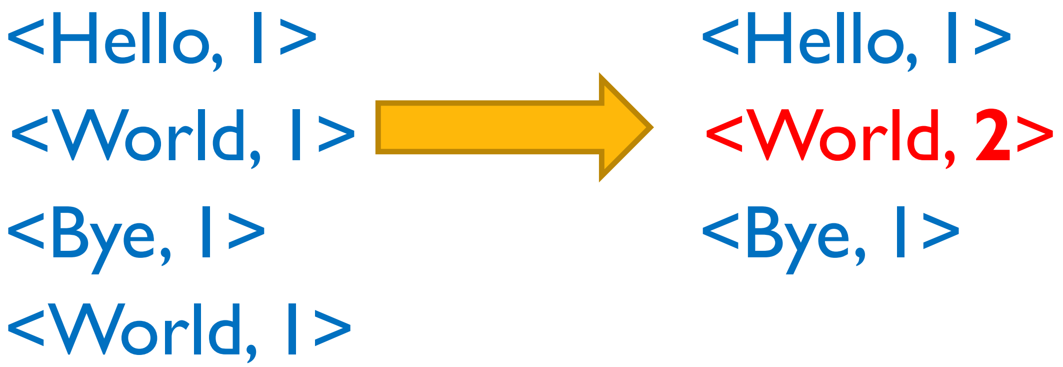
<Hello, 1>
<World, 1>
<Bye, 1>
<World, 1>

A simple example - map

- Some words like "World" appear multiple times in the result.
- All values that have the same key are grouped together.

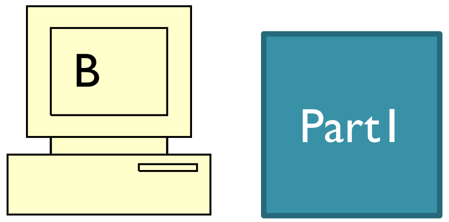


"Hello world, bye world"



A simple example - map

- **Consider the <u>second instance</u>.**
- The second instance reads its data.
- It creates a <key,value> pair for each word that it reads. A key is a word and the corresponding value is the number 1.



"Hello cloud, goodbye cloud"

<Hello, 1>

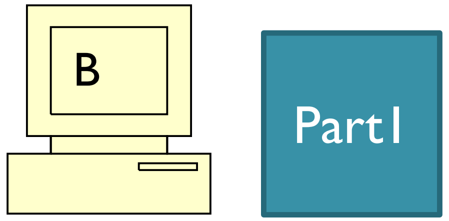
<Cloud, 1>

<Goodbye, 1>

<Cloud, 1>

A simple example - map

Then, the **second instance** groups all values that have the same key together.

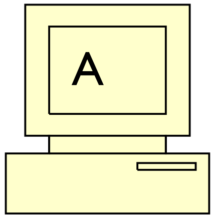


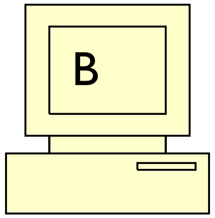
"Hello cloud, goodbye cloud"

<Hello, 1>	<Hello, 1>
<Cloud, 1>	<Cloud, **2**>
<Goodbye, 1>	< Goodbye, 1>
<Cloud, 1>	

A simple example

So until now, we have:



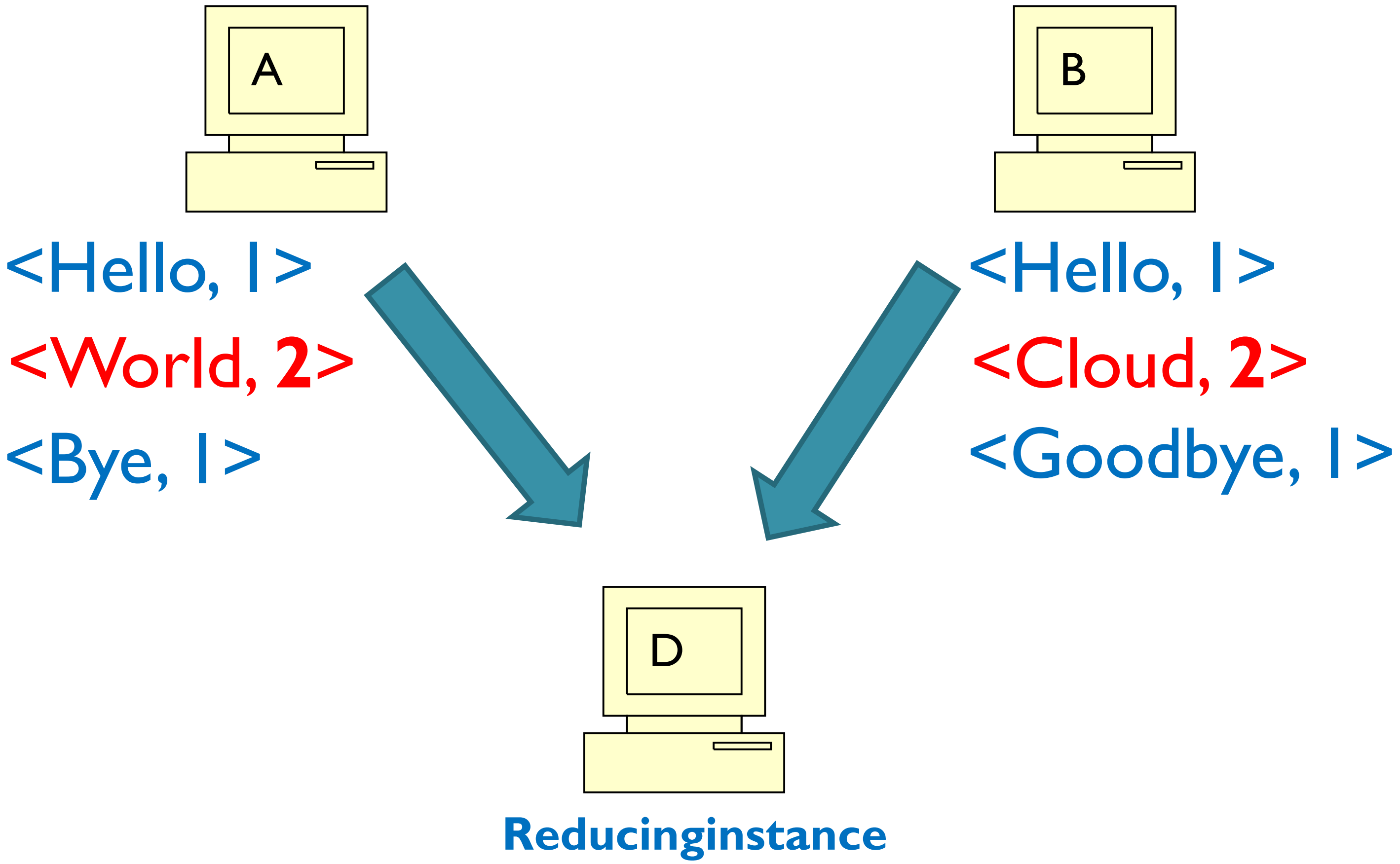


<Hello, 1>	<Hello, 1>
<World, **2**>	<Cloud, **2**>
<Bye, 1>	<Goodbye, 1>

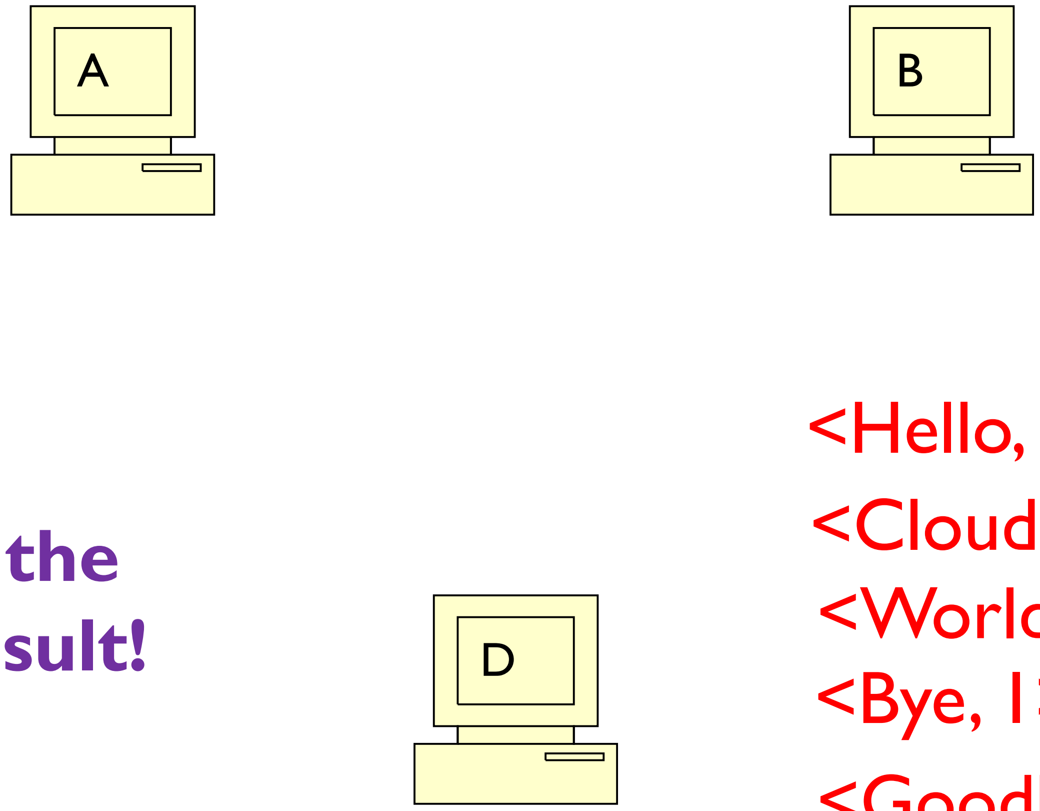
Next, the **reduce** **phase** will combine the local results found by all instances. →

A simple example - reduce

- The **master instance** will start **R** reducing instances for combining results of mapping instances.
- In this example, only one reducing instance is used (instance D)



A simple example - reduce



Reducinginstance

This is the final result!

<Hello, 2>

<Cloud, 2>

<World, 2>

<Bye, 1>

<Goodbye, 1>

A simple example

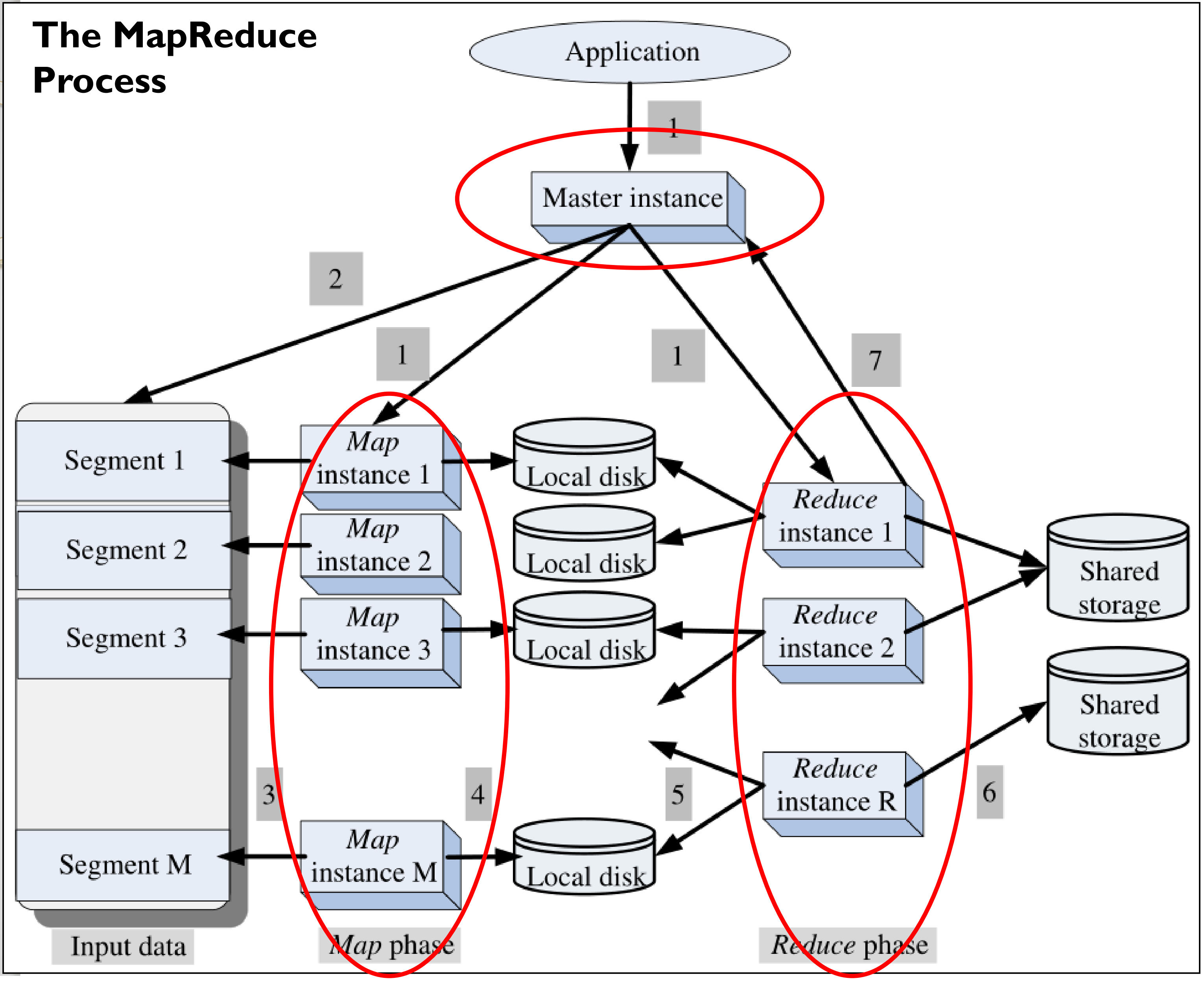
This is the code for this example:

```
map(String key, String value):
  //key: document name; value: document contents
  for each word w in value:
  EmitIntermediate (w, "1");
```

Combine local results

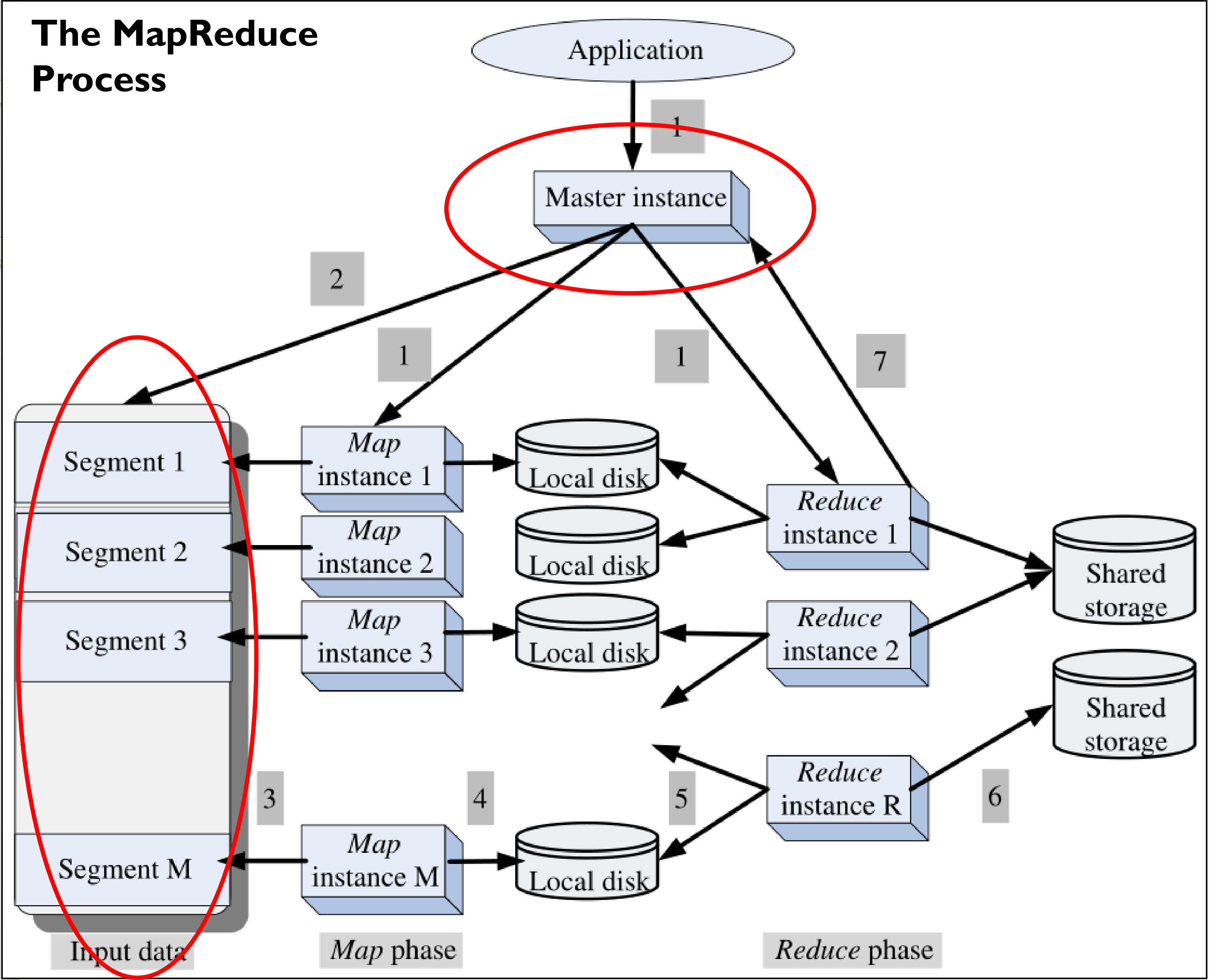
```
reduce (String key, Iterator values):
  // key: a word; values: a list of counts
  int result  = 0;
  for each v in values:
  result += ParseInt (v);
  Emit (AsString (result));
```

The MapReduce Process



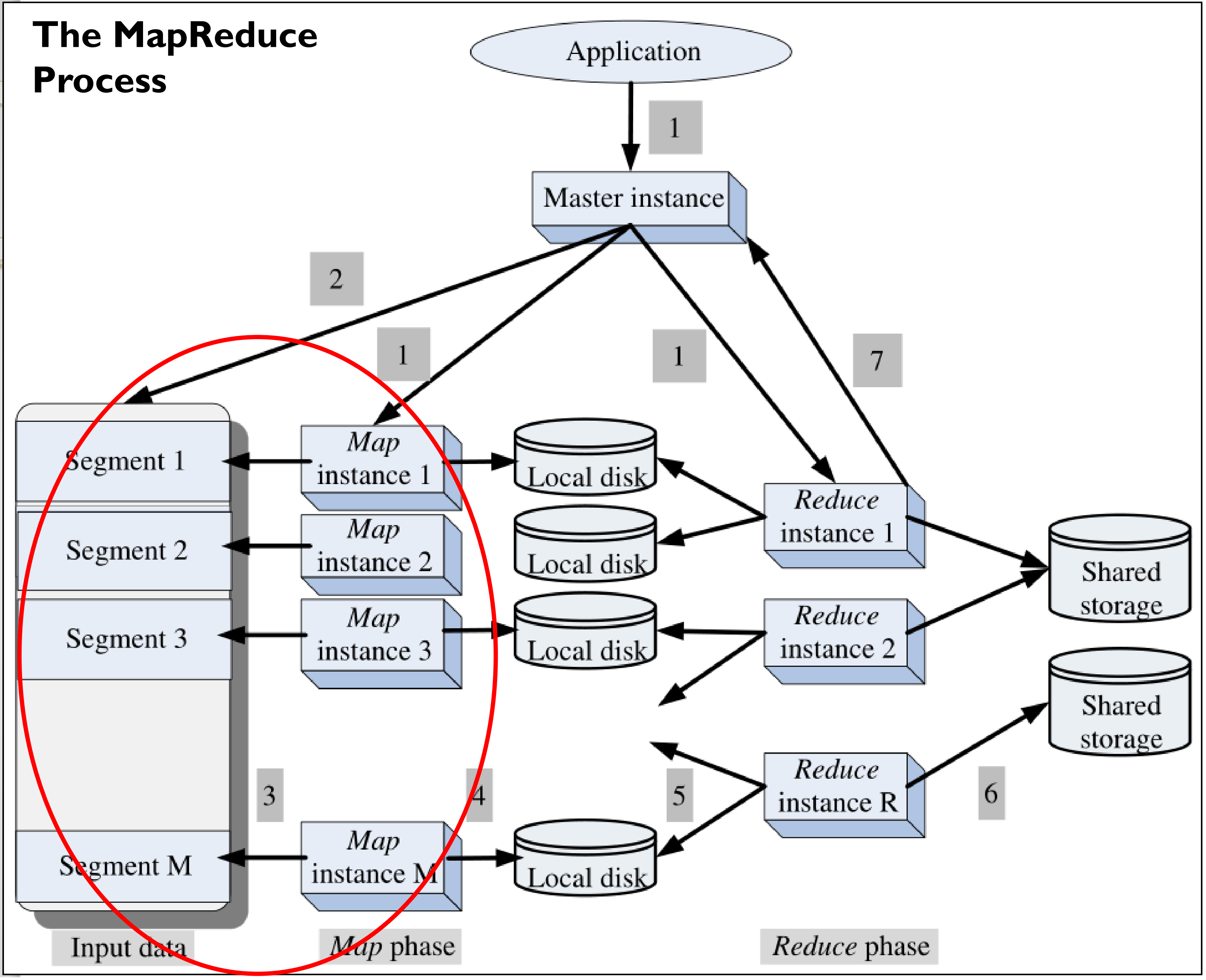
(1) An application starts a master instance and M worker instances for the Map phase and, later, R worker instances for the Reduce phase.

The MapReduce Process



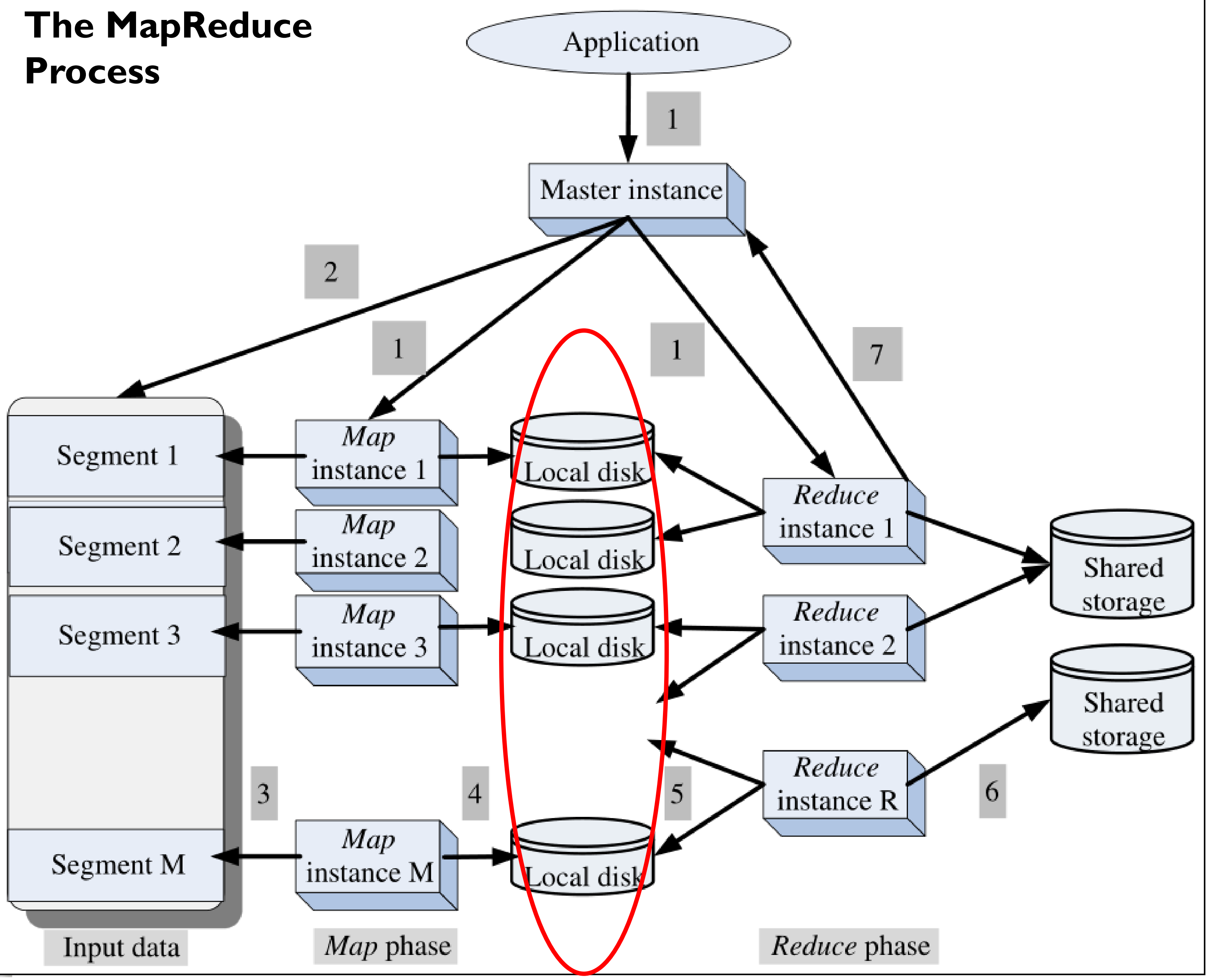
(2) The master split (分配) the input data in M segments (parts).

The MapReduce Process



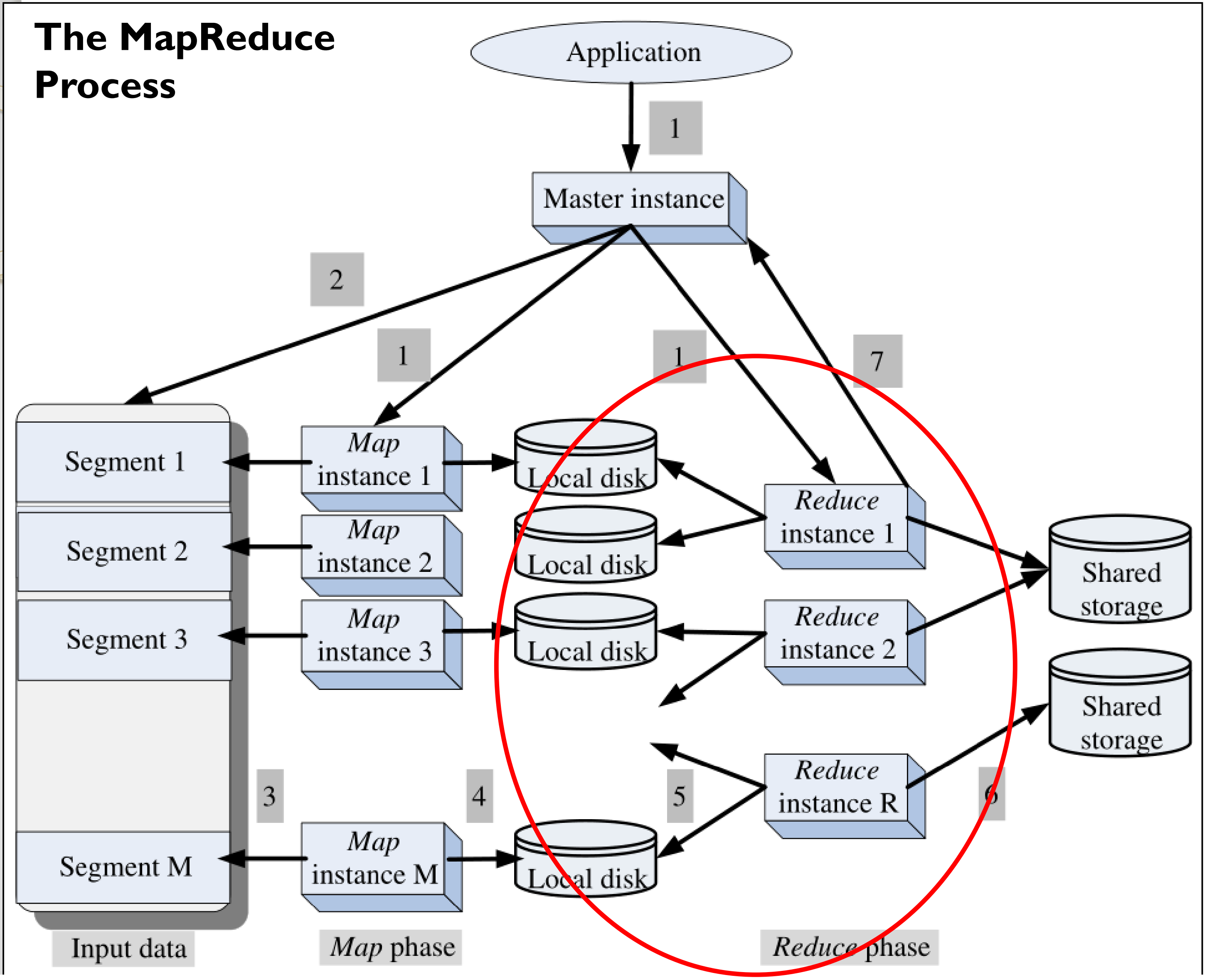
(3) Each Map instance reads its input data segment and processes the data

The MapReduce Process



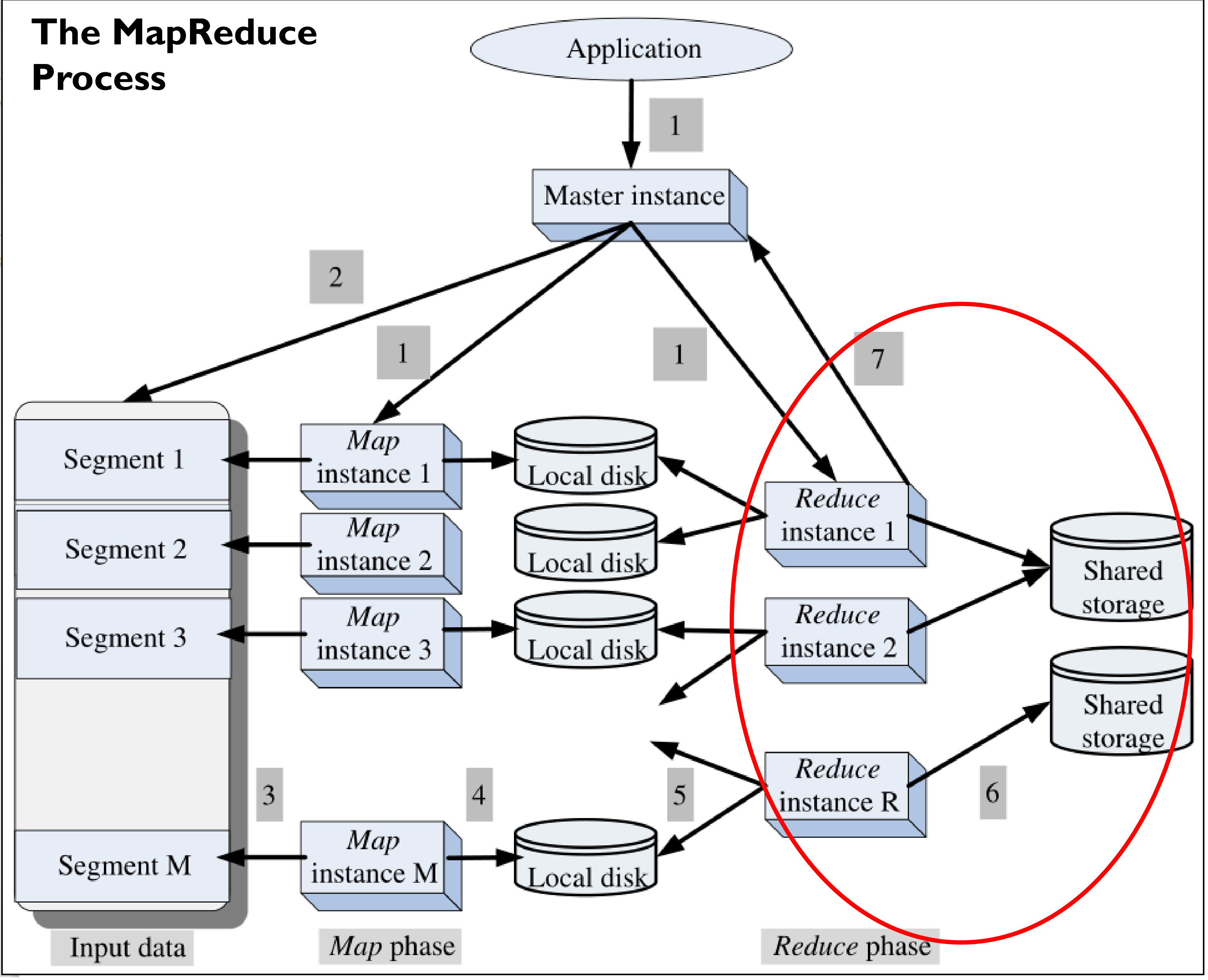
(4) The local results are stored on the local disks of the computers where the Map instances are executed.

The MapReduce Process



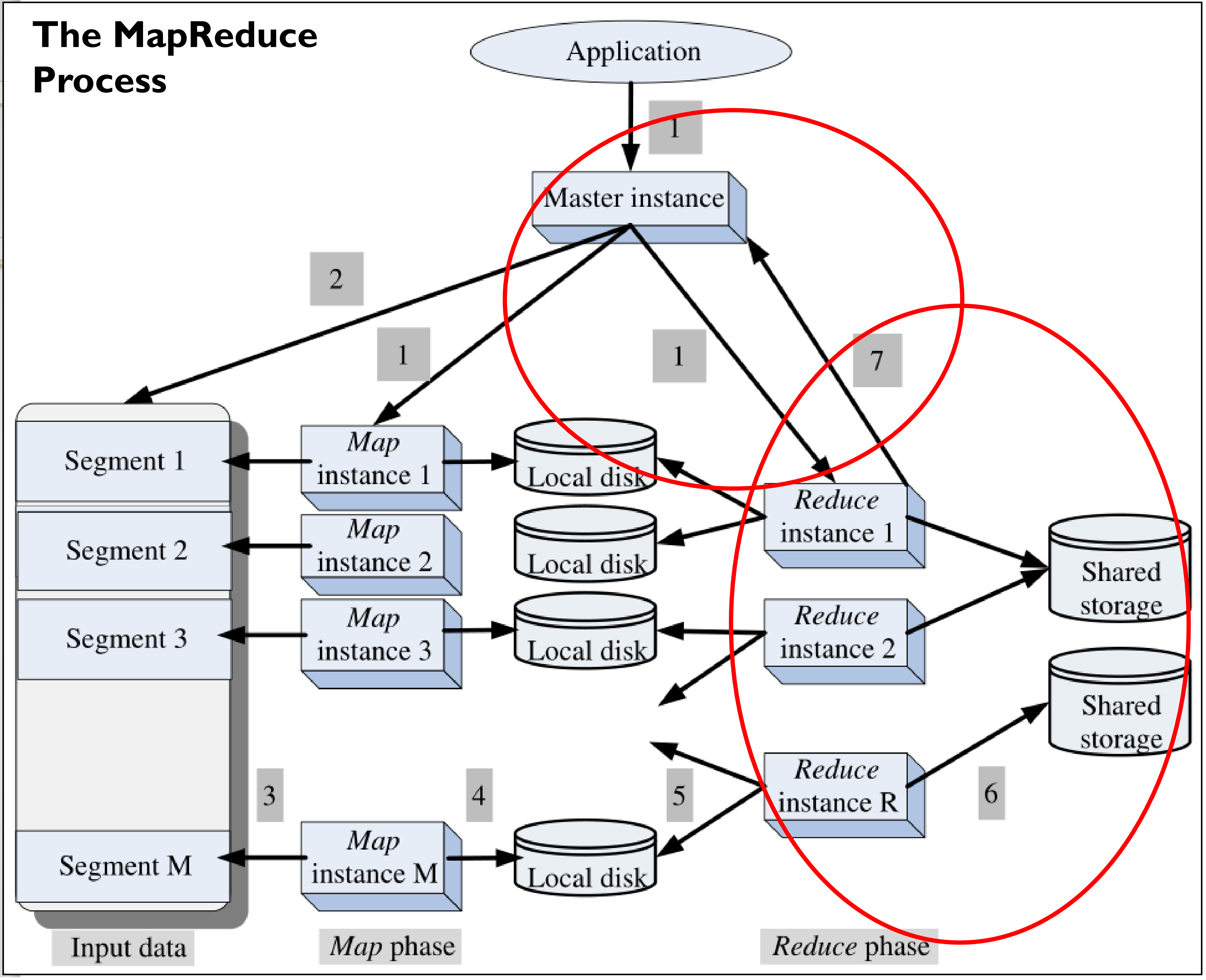
(5) The R reduce instances read the local results and merge the results.

The MapReduce Process



(6) The final results are written by the Reduce instances to a shared storage (共享存储)

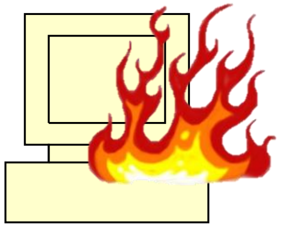
The MapReduce Process



(7) The master instance monitors the Reduce instances and. When all of them have finished, it is the **END**.

More details

- The data is usually split in blocks of 16 MB to 64 MB (megabytes - 兆字节).
- The number of instances can be a few to hundreds, or thousands of instances.
- What if some instances crashes? →



What happen if an instance fails?

- **Fault-tolerance (容错)**: to ensure that a task is accomplished properly even if some machines stop working.
- The **master instance** asks each worker machine about their **state** (idle 空闲状态, in-progress 正在进行, or completed 完成任务) and **identity**.
- If the worker machine does not respond, the master instance considers that this machine's sub-task has failed.

Master
What is your state?
…

What happen if an instance fails?

- A **task in progress** (正在进行) on a **failed worker** is set to **idle** (空闲状态).
- The task can then be given to another worker (computer).
- The master writes takes of note of the tasks that have been completed.
- The **data** is stored using the
  GFS (Google File System).

What is a typical MapReduce machine in a cluster?

According to the book, in **2012**, a typical computer for experimenting with MapReduce has the following characteristics:

- dual-processor x86 running Linux,
- 2–4 GB of memory,
- Network card: 100–1,000 Mbps.
- Data is stored on IDE 7 disks attached directly to individual machines.
- The file system uses replication (复制)

What is a typical MapReduce machine in a cluster?

- A cluster consists of hundreds or thousands of machines.
- It provides **availability** (可利用性) and **reliability** (可靠) using **unreliable hardware**.
- The **input data** is stored on the **local disk** of each instance to reduce communication between computers.

A second example

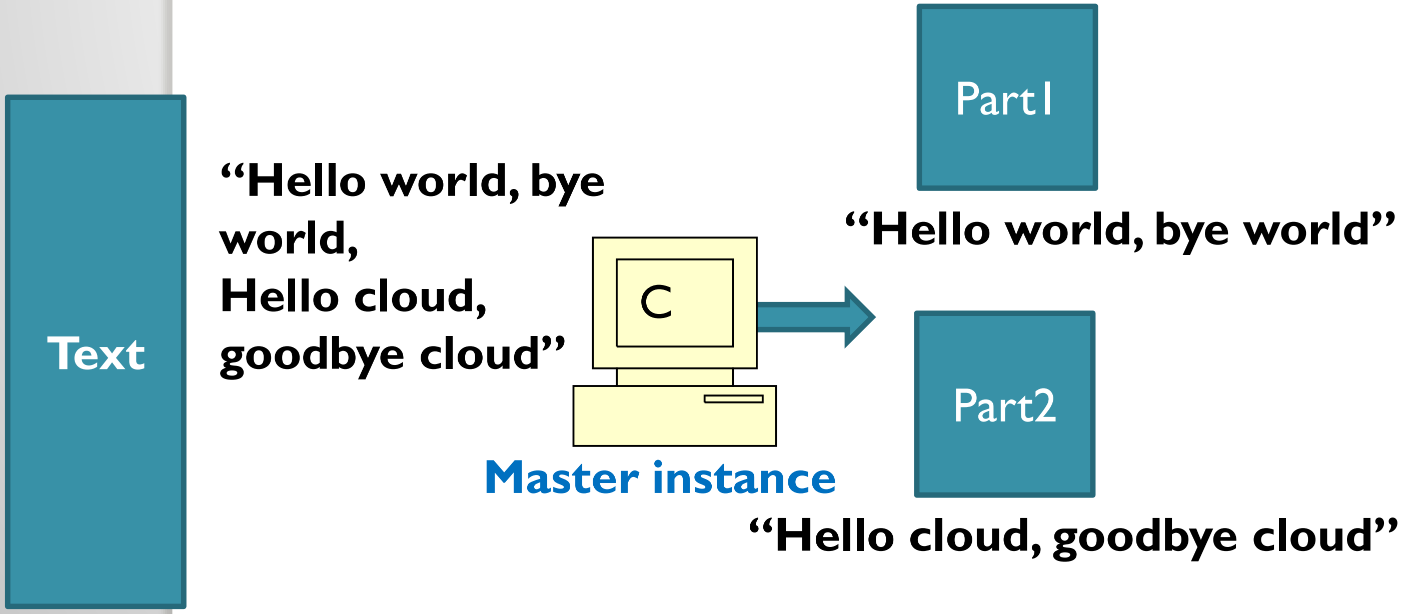
Task: analyze a text to count how many words with 1 letters, with 2 letters, with 3 letters, with 4 letters…

Text

**“Hello world, bye world,
Hello cloud,
goodbye cloud”**

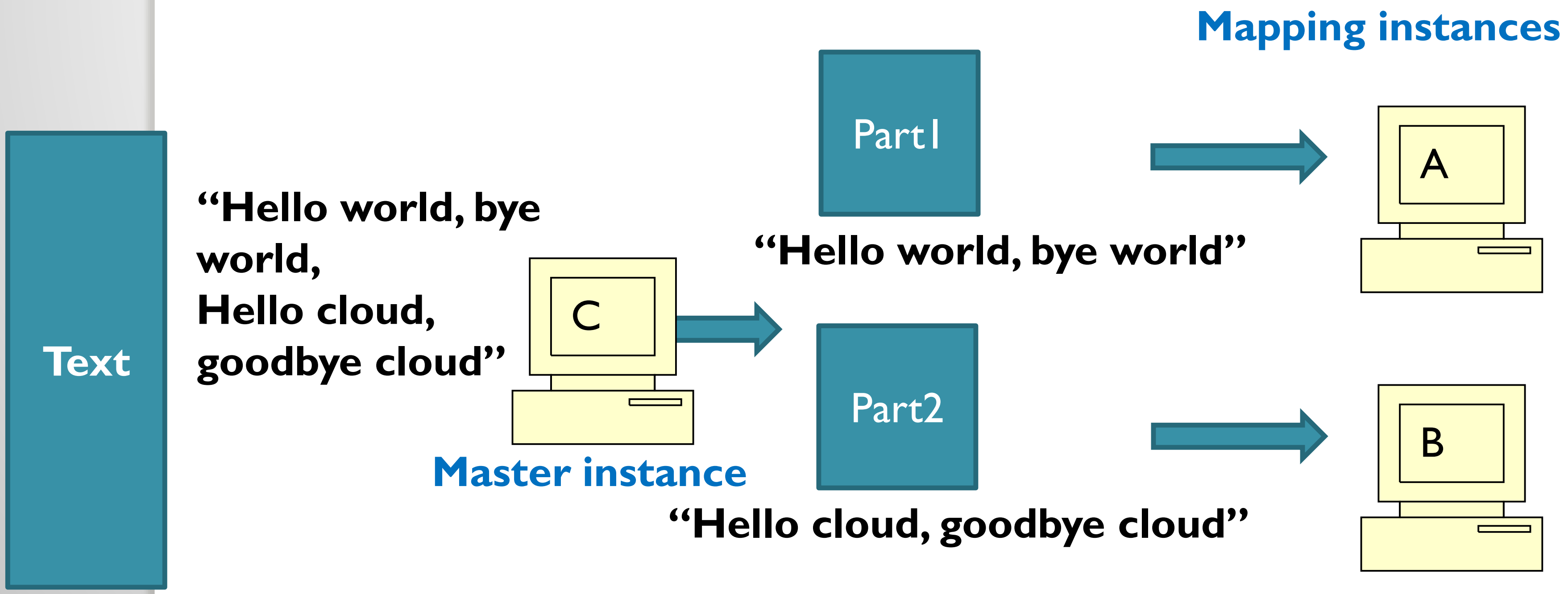
A second example

The **master instance** first splits the data into **M** data blocks. Here M = 2.


Text
"Hello world, bye world, Hello cloud, goodbye cloud"
C
Master instance
Part1
"Hello world, bye world"
Part2
"Hello cloud, goodbye cloud"

A second example

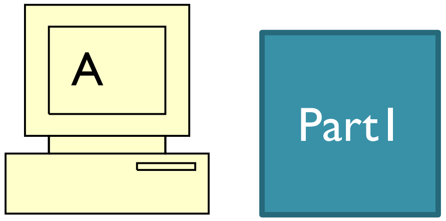
Then, it starts **M instances** and gives a data block to each instance.


Mapping instances
Part1
Text
"Hello world, bye world, Hello cloud, goodbye cloud"
C
Master instance
"Hello world, bye world"
A
Part2
B
"Hello cloud, goodbye cloud"

A second example - map

All instances work in parallel.

Consider the <u>first instance</u>. The first instance reads its data. It creates a <key,value> pair for each word that it reads. A key is the number of letters in the word and the value is the word.



"Hello world, bye world"

<5, Hello>

<5, World>

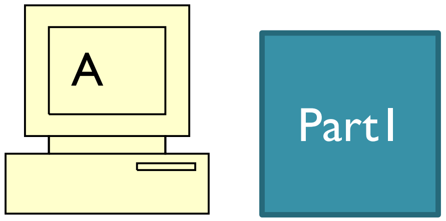
<3, Bye>

<5, World>

A second example - map

Some words like "World" appear multiple times in the result.

All values that have the same **key** are grouped together.



"Hello world, bye world"

<5, Hello>

<5, World>

<3, Bye>

<5, World>

<3, Bye>

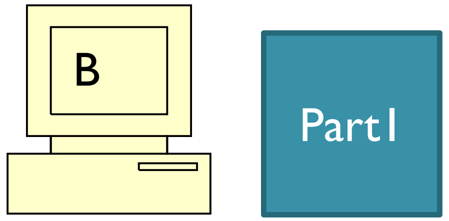
<5, World, World, Hello>

Note: value having the same key are automatically grouped

A second example - map

Consider the <u>second instance</u>.

The second instance reads its data. It creates a <key,value> pair for each word that it reads, where a key is a number of letters and the corresponding value is a word.



"Hello cloud, goodbye cloud"

<5, Hello>

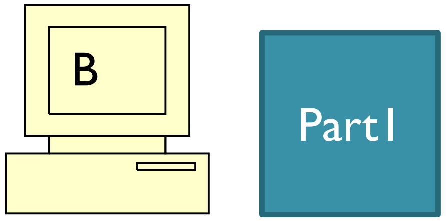
<5, Cloud>

<7,Goodbye>

<5, Cloud>

A second example - map

Then, all values that have the same **key** are grouped together.



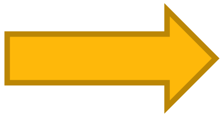
"Hello cloud, goodbye cloud"

<5, Hello>

<5, Cloud>

<7,Goodbye>

<5, Cloud>



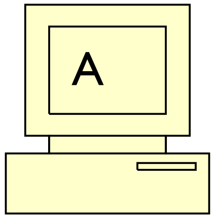
<5, Hello, Cloud, Cloud>

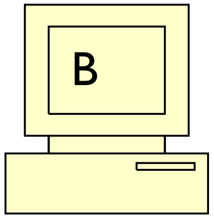
<7, Goodbye>

Note: value having the same key are automatically grouped

A second example

So until now, we have:





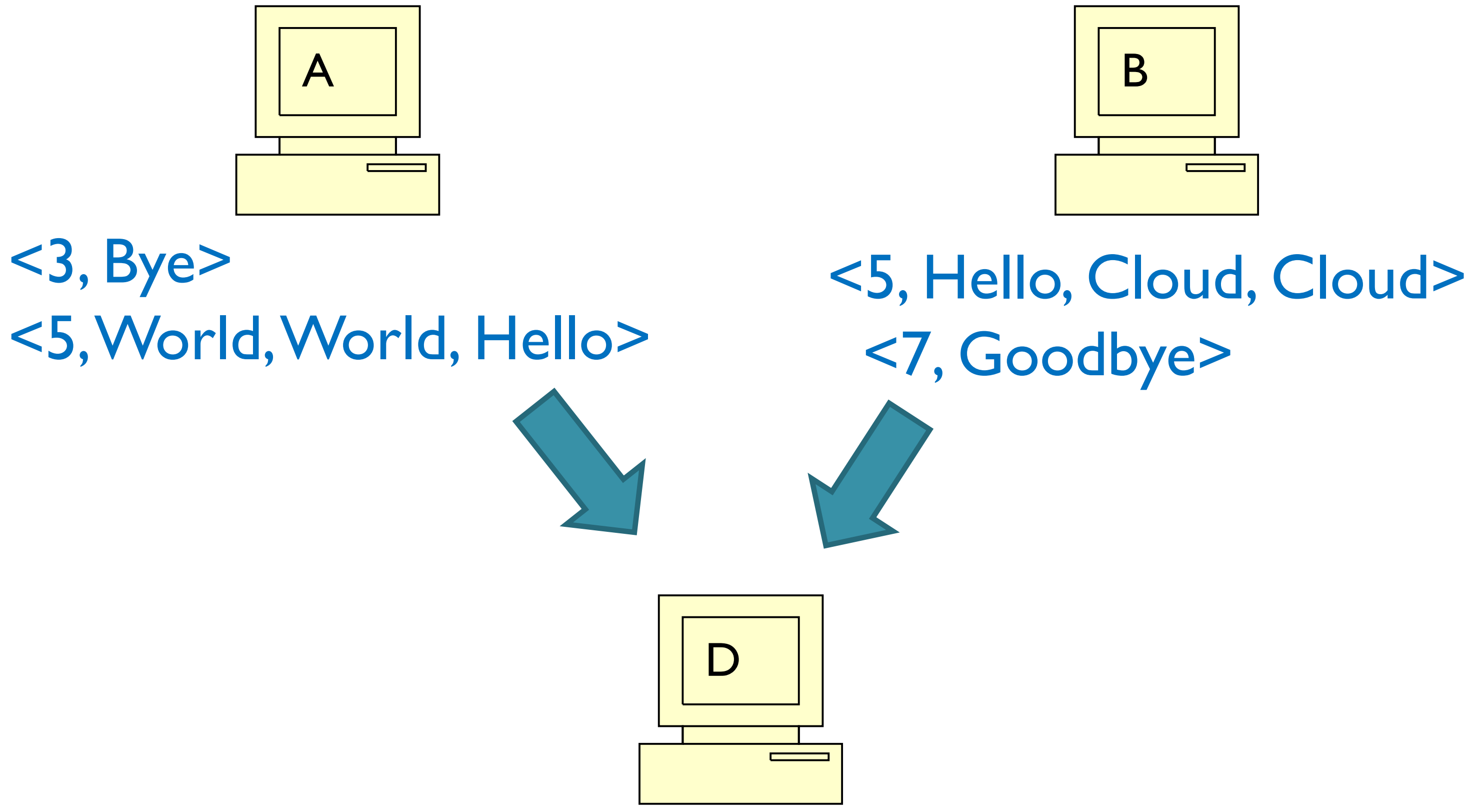
<3, Bye>
<5, World, World, Hello>

<5, Hello, Cloud, Cloud>
<7, Goodbye>

Now, the **reduce phase** will take place to combine the local results found by each instance

A second example - reduce

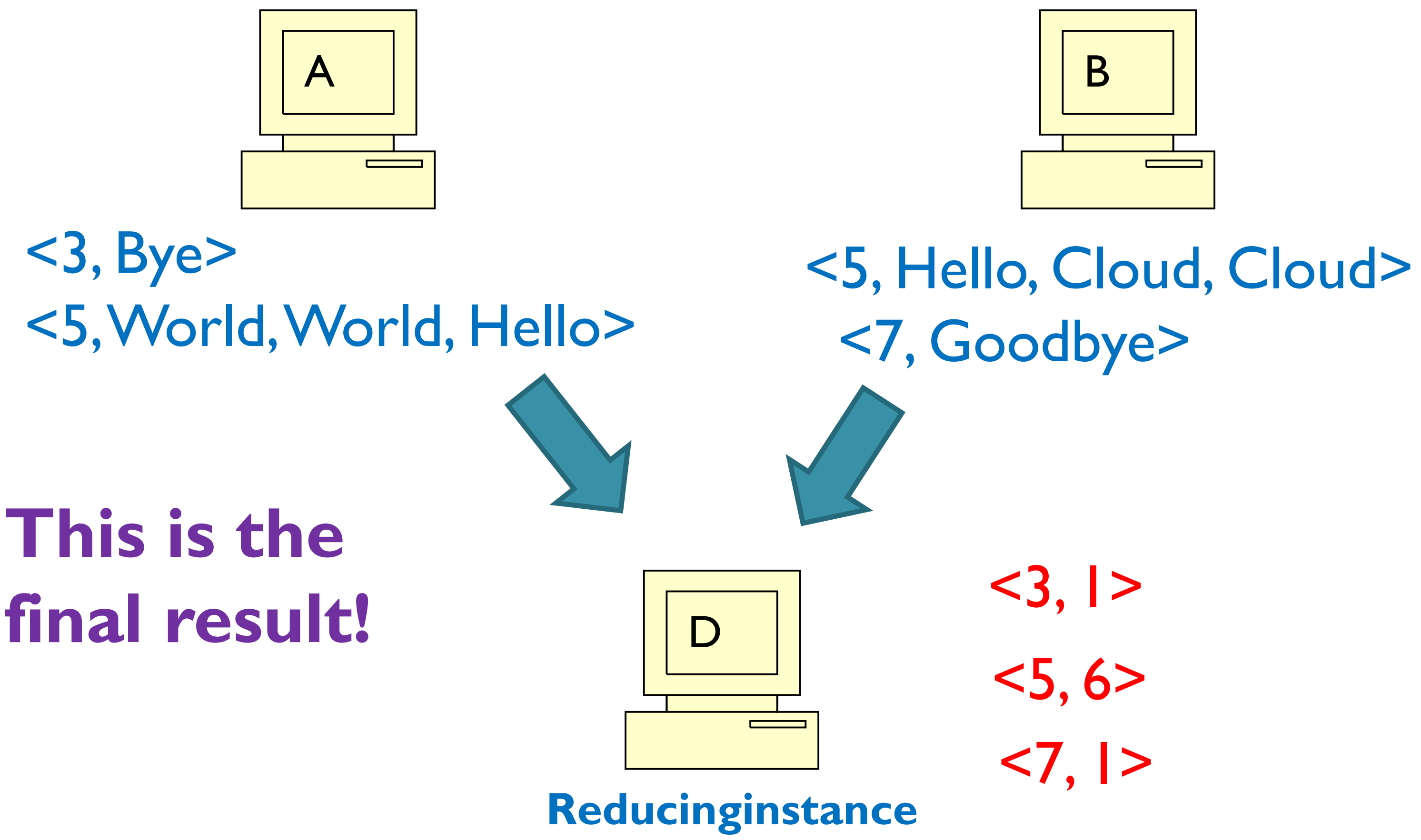
- The **master instance** starts **R** reducing instances for combining results of mapping instances.
- In this example, only one reducing instance is used (instance D)



Reducinginstance

A second example - reduce

The result is shown below. It means that there is one word containing three letters, six words containing five letters, and one word containing seven letters.

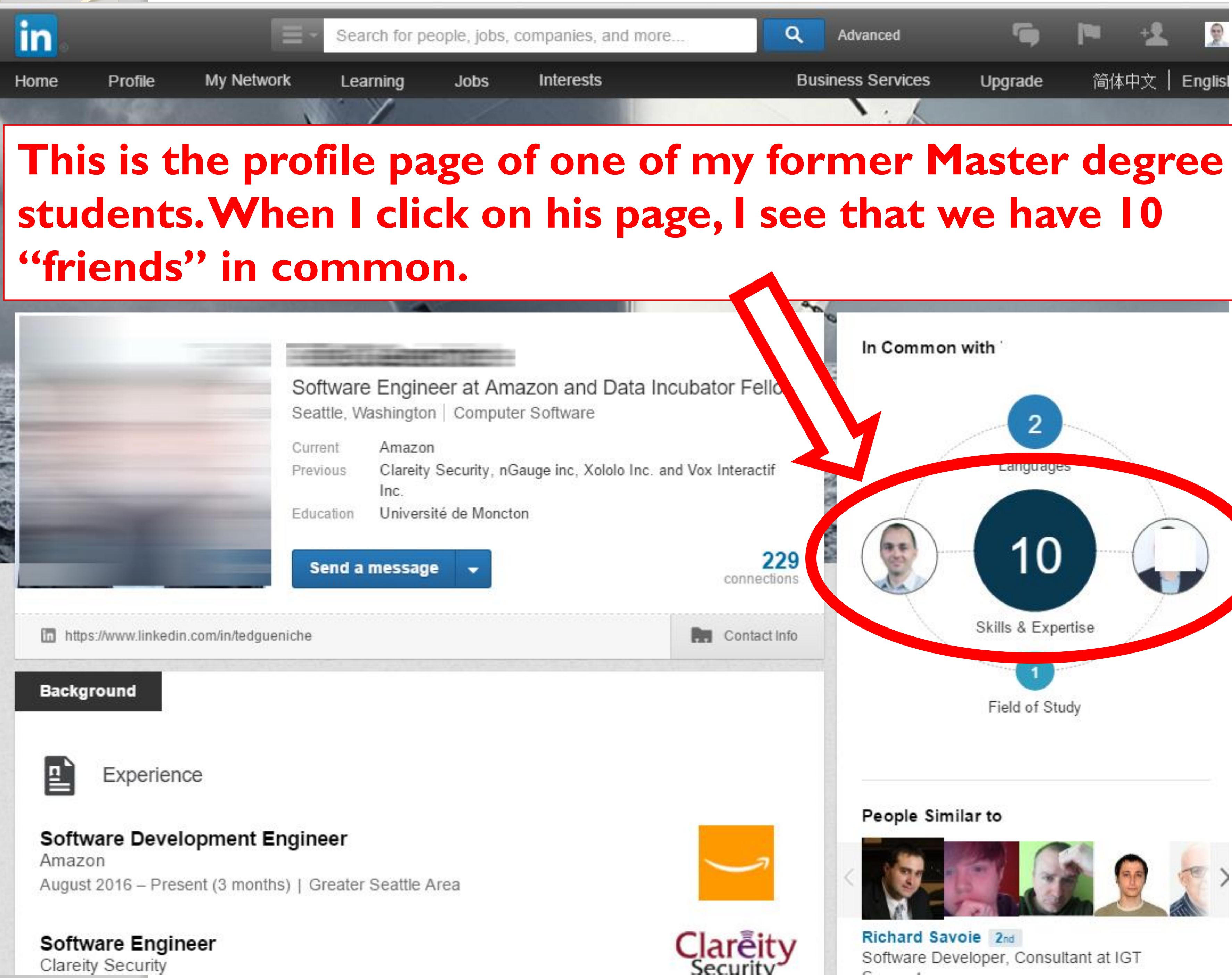


This is the final result!

A third example

- Consider a **social network** (社会网络) **like Wechat, QQ, LinkedIn** where you can be friend with other people.
- If you are a **LinkedIn user** and you view the LinkedIn page of a friend, the page will indicates **how many friends you have in common.**
- **Illustration →**

This is the profile page of one of my former Master degree students. When I click on his page, I see that we have 10 "friends" in common.



Search for people, jobs, companies, and more... Advanced

Home Profile My Network Learning Jobs Interests Business Services Upgrade 简体中文 | Englis

[redacted]

Software Engineer at Amazon and Data Incubator Fello
Seattle, Washington | Computer Software

Current	Amazon
Previous	Clareity Security, nGauge inc, Xololo Inc. and Vox Interactif Inc.
Education	Université de Moncton

Send a message

229 connections

https://www.linkedin.com/in/tedgueniche

Contact Info

Background

Experience

Software Development Engineer
Amazon
August 2016 – Present (3 months) | Greater Seattle Area

Software Engineer
Clareity Security

In Common with

2 Languages

10 Skills & Expertise

1 Field of Study

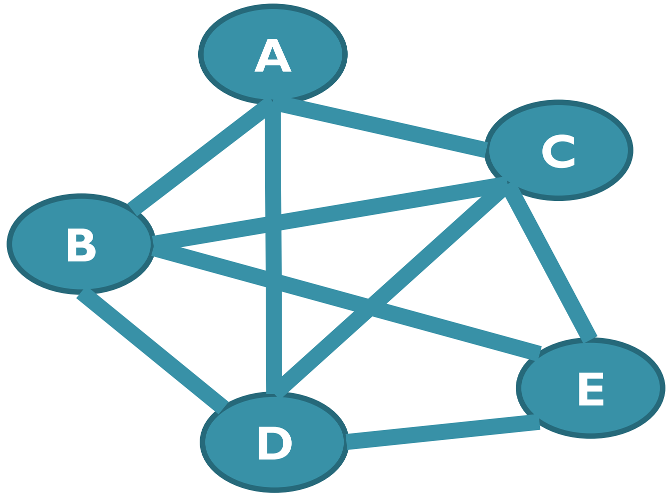
People Similar to

Richard Savoie 2nd
Software Developer, Consultant at IGT

A third example

- Suppose that we have a social network with five users: **A,B, C, D, E**
- We assume that **friendship** (友谊) is a **bidirectional relationship** (双向关系).
- In other words, if you are a friend of someone, s/he is also your friend.

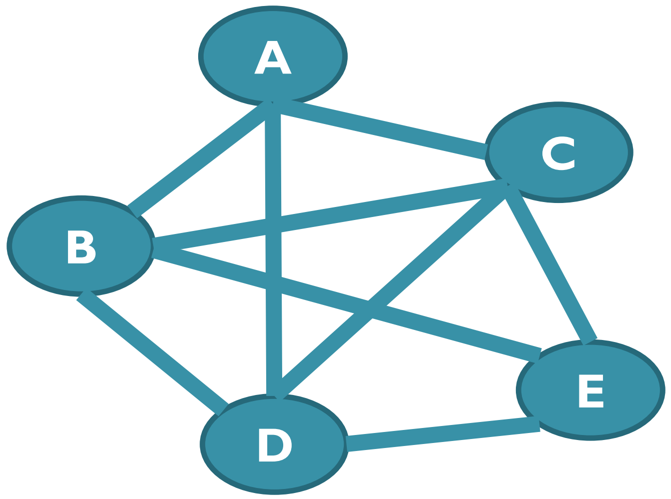
- Assume that this is the friendship graph:



A third example

Assume that data about friendship between users is stored in a text file as follows:

- A -> B C D
- B -> A C D E
- C -> A B D E
- D -> A B C E
- E -> B C D



A third example

The data file will be split and sent to various **mapping instances**.

Mapping instances will process each line that they receive as follows: →

A third example - map

The first line A -> B C D is transformed as:

Key		Value
(A B)	->	B C D
(A C)	->	B C D
(A D)	->	B C D

by combining A with each of his friend.

A third example - map

The second line B -> A C D E is transformed as:

Key		Value
(A B)	->	A C D E
(B C)	->	A C D E
(B D)	->	A C D E
(B E)	->	A C D E

A third example - map

The third line C -> A B D E is transformed as:

Key	Value
(A C) ->	A B D E
(B C) ->	A B D E
(C D) ->	A B D E
(C E) ->	A B D E

A third example - map

The fourth line D -> A B C E is transformed as:

Key	Value
(A D)	-> A B C E
(B D)	-> A B C E
(C D)	-> A B C E
(D E)	-> A B C E

A third example - map

The fifth line E -> B C D is transformed as:

Key		**Value**
(B E)	->	B C D
(C E)	->	B C D
(D E)	->	B C D

A third example – map (sort)

The values are then grouped by their key:

- (A B) -> (A C D E) (B C D)
- (A C) -> (A B D E) (B C D)
- (A D) -> (A B C E) (B C D)
- (B C) -> (A B D E) (A C D E)
- (B D) -> (A B C E) (A C D E)
- (B E) -> (A C D E) (B C D)
- (C D) -> (A B C E) (A B D E)
- (C E) -> (A B D E) (B C D)
- (D E) -> (A B C E) (B C D)

Furthermore, they are sorted (as above)

A third example - reduction

This data is then split and sent to reducers

- (A B) -> (A C D E) (B C D)
- (A C) -> (A B D E) (B C D)
- (A D) -> (A B C E) (B C D)
- (B C) -> (A B D E) (A C D E)
- (B D) -> (A B C E) (A C D E)
- (B E) -> (A C D E) (B C D)
- (C D) -> (A B C E) (A B D E)
- (C E) -> (A B D E) (B C D)
- (D E) -> (A B C E) (B C D)

A third example - reduction

Each **reducer will intersect the list of value on each line:**

The **first line**:

(A B) -> (A C D E) (B C D)

will thus become:

(A B) -> (C D)

A third example - reduction

Each **reducer will intersect the list of value on each line:**

The **second line**:

(A C) -> (A B D E) (B C D)

will thus become:

(A C) -> (B D)

and so on….

A third example – final result

The final result is:

Key		Value
(A B)	->	(C D)
(A C)	->	(B D)
(A D)	->	(B C)
(B C)	->	(A D E)
(B D)	->	(A C E)
(B E)	->	(C D)
(C D)	->	(A B E)
(C E)	->	(B D)
(D E)	->	(B C)

A third example – final result

The final result is:

Key	Value
(A B) ->	(C D)
(A C) ->	(B D)
(A D) ->	(B C)
(B C) ->	(A D E)
(B D) ->	(A C E)
(B E) ->	(C D)
(C D) ->	(A B E)
(C E) ->	(B D)
(D E) ->	(B C)

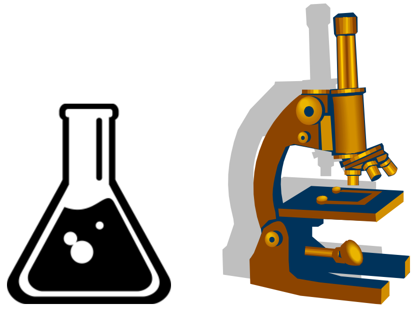
Having calculated this information, we know the friends in common between any pairs of persons.

For example: A and D have the friends B and C in common

A third example - conclusion

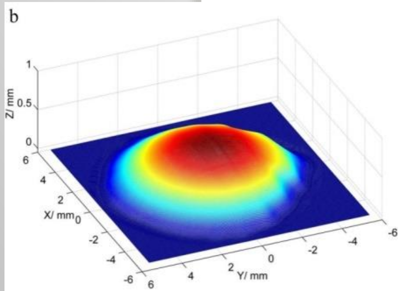
- In this example, we have explained how the **MapReduce** framework can be used to calculate common friends in a social network.
- **Why doing this?**
  - Big social networks such as **LinkedIn** have a lot of money.
  - By precaculating (预先计算) information about common friends, a social network can provide the information more quickly to users.
  - This can be recalculated every day.

4.8-CLOUD FOR SCIENCE AND ENGINEERING



Cloud for science/engineering

- In the **last 2000 years**, science was mostly empirical.
- In **recent decades,** computational science (计算科学) has emerged where **computers** are used to **simulate complex phenomena**.
- Science may now combine:
  - theory, experiment, and simulation (仿真)



Cloud for science/engineering

Generic problems involving data, in science:

- Collecting experimental data.
- Managing very large volumes of data.
- Building and executing models.
- Integrating data and literature.
- Documenting experiments.
- Sharing the data with others; data preservation for long periods of time.

All these activities require powerful computing systems.

Cloud for science/engineering

Example of large databases:

- The **Chinese National Space Administration** may collect huge amount of **data about space** using various equipment.
- The **Chinese Meteorological Administration** may collect huge amount of data about the **weather.**

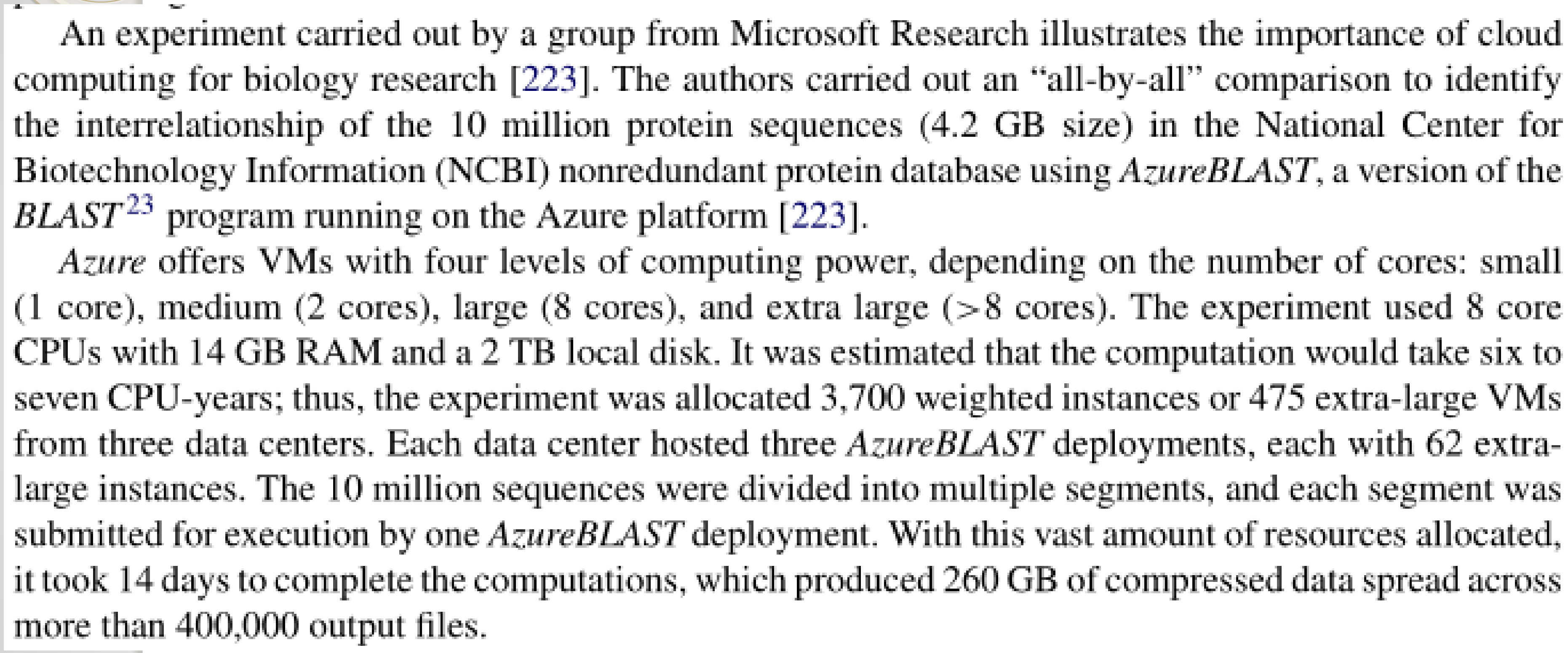
The cloud is useful to analyze such large amount of data.

Biology research

- Cloud computing is very important for biology research.
  - **Computation of molecular dynamics** is CPU intensive.
  - **Protein alignment** (蛋白质序列) is data-intensive.

- An example →

Biology research - example



An experiment carried out by a group from Microsoft Research illustrates the importance of cloud computing for biology research [223]. The authors carried out an "all-by-all" comparison to identify the interrelationship of the 10 million protein sequences (4.2 GB size) in the National Center for Biotechnology Information (NCBI) nonredundant protein database using *AzureBLAST*, a version of the *BLAST*[23] program running on the Azure platform [223].

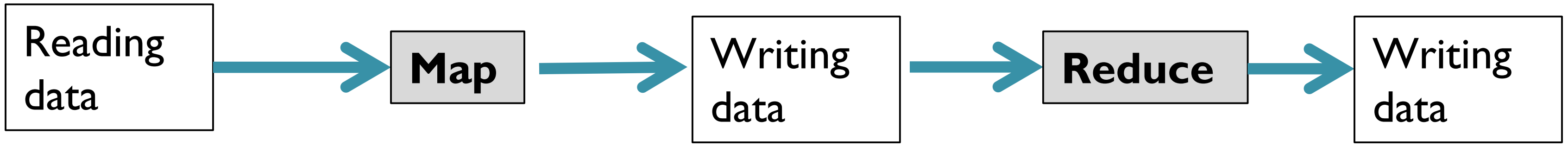
Azure offers VMs with four levels of computing power, depending on the number of cores: small (1 core), medium (2 cores), large (8 cores), and extra large (>8 cores). The experiment used 8 core CPUs with 14 GB RAM and a 2 TB local disk. It was estimated that the computation would take six to seven CPU-years; thus, the experiment was allocated 3,700 weighted instances or 475 extra-large VMs from three data centers. Each data center hosted three *AzureBLAST* deployments, each with 62 extra-large instances. The 10 million sequences were divided into multiple segments, and each segment was submitted for execution by one *AzureBLAST* deployment. With this vast amount of resources allocated, it took 14 days to complete the computations, which produced 260 GB of compressed data spread across more than 400,000 output files.

Using 3,700 instances, a task that would took about 7 years on a single computer was done in 14 days!

ADDITIONAL INFORMATION

Introduction

- Last week, we talked about **MapReduce.**
- **MapReduce** is a model to create cloud applications.
- It is used for developing applications that can be used in the cloud.
- It is called **MapReduce** because there are two steps called "**Map**" and "**Reduce**".



Introduction

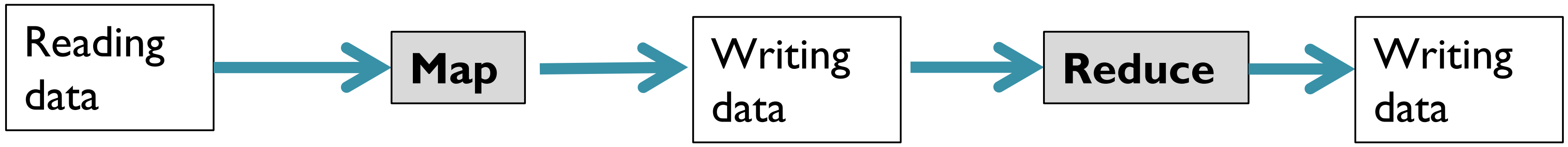
- **MapReduce** is a popular model.
- There are many **other models** for developing cloud applications.
- For example:
  - **Apache Spark**
  - **Apache Storm**
  - …

Apache SPARK

- **Spark** is more complicated than **MapReduce**.
- **Spark** offers more than **100 operators** to transform data.
- **Spark** can be used with the **Java, Python** and **Scala** programming languages (编程语言).

Apache SPARK

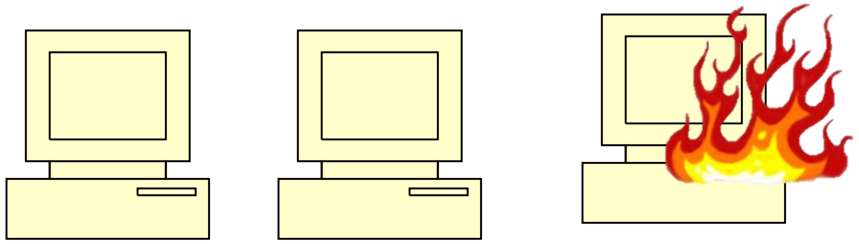
- A problem of **MapReduce** is that it reads and write data many times to the storage 存储 (**before** and **after** each **Map** or **Reduce** operation).



- This can make a cloud application **slower**.
- **Solution**:
  - Using **Spark**, data can be kept in memory.
  - In other words, data is not read and written many times.
  - Spark can read and transform data. However, Spark is "lazy". It only read and transform data when an action needs to be performed on the data.

Apache SPARK

- When Spark transforms data, the data is then stored in a structure called:
  Resilient Distributed Dataset (RDD).
  Resilient = 能复原的
  Distributed = 分布式
  Dataset = 数据
- All the transformations that are applied to data are remembered so that a dataset can be recovered if some failure happen.



Conclusion

- In this part, I have presented the MapReduce model, which is widely used for cloud computing.
- The first assignment is announced today.

http://philippe-fournier-viger.com/COURSES/CLOUD/

References

- Chaptre 4. D. C. Marinescu. Cloud Computing Theory and Practice, Morgan Kaufmann, 2013.
- http://stevekrenzel.com/finding-friends-with-mapreduce
- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html