



# 云计算入门

## Introduction to Cloud Computing GESC1001

**Philippe Fournier-Viger**

Professor

School of Humanities and Social Sciences

[philfv8@yahoo.com](mailto:philfv8@yahoo.com)

Fall 2020

QQ group: **1127433879**



# Introduction

## Last week:

- **Introduction** (continued)
- **Chapter 2:** challenges of distributed and parallel systems

## • Today:

- The challenge of **mutual exclusion** (互斥) for distributed/parallel systems.
- **Chapter 3: cloud infrastructure**, with real-life examples from **Amazon** (亚马逊).

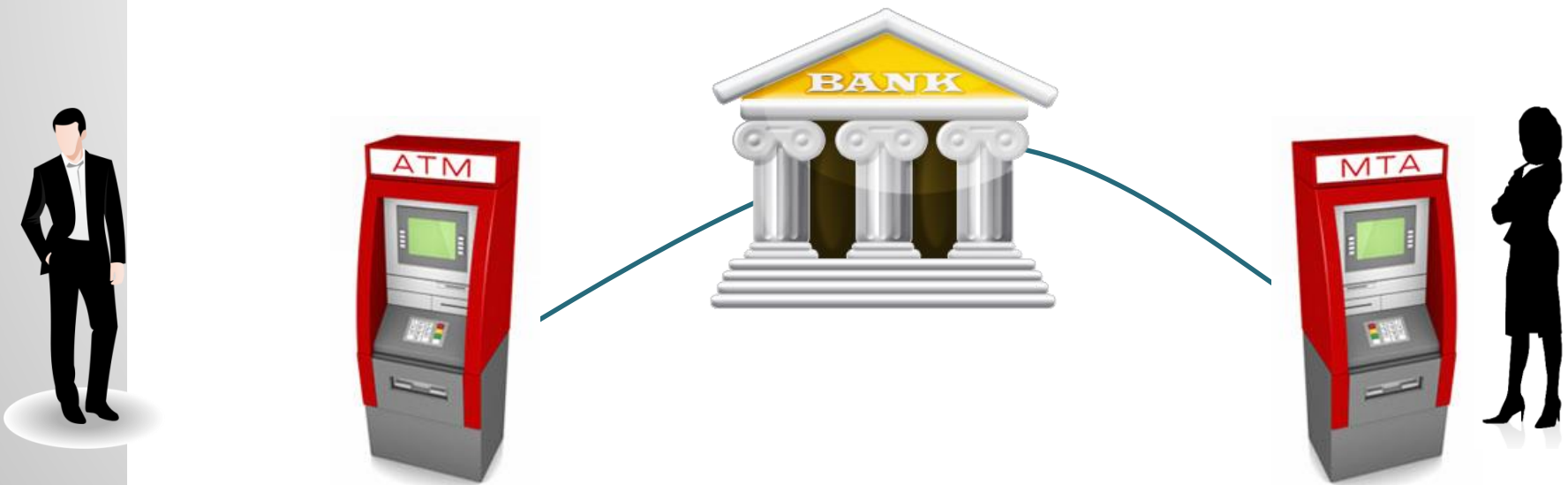
# Course schedule

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

◦ **MUTUAL EXCLUSION**  
(互斥)

# Concurrent accesses to data (并发访问)

- Problems may occur when applications in a distributed system access the same resource at the same time.
- **Example:** some money may be lost when two bank transfers are performed at the same time on the same bank account.



# A solution: Mutual exclusion (互斥)

- **Mutual exclusion** (互斥): using techniques to ensure that two applications do not access the same resources at the same time.
- **Critical resource** (关键资源): a resource that must be accessed/modified by no more than one computer at a time.
- **Examples:**
  - 2 computers request to know the exchange rate for CNY (元) to USD (\$) at the same time. → **no problem!**
  - 2 computers reserve the same airplane seat (飞机的座位) at the same time → **problem!**

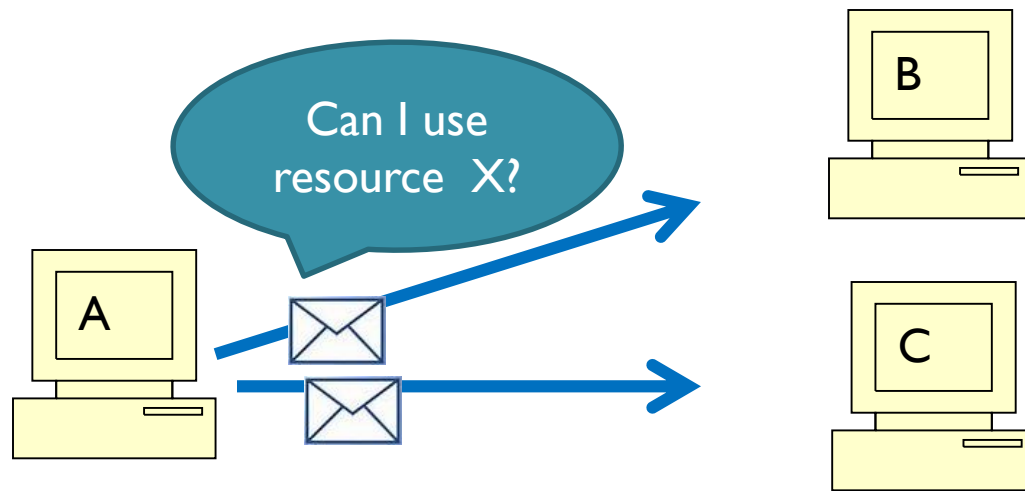
# Some observations

- **Concurrent access problems** do not occur when computers are **only reading** data.
- **Problems may occur if:**
  - a computer is **overwriting** data that was read and is used by another computer.
  - a computer is **overwriting** data written by another computer.



# How to ensure mutual exclusion?

**Naive approach (质朴的方):** If a computer needs to access a resource (data), it sends a message to all the other computers to ask if it can use the resource.

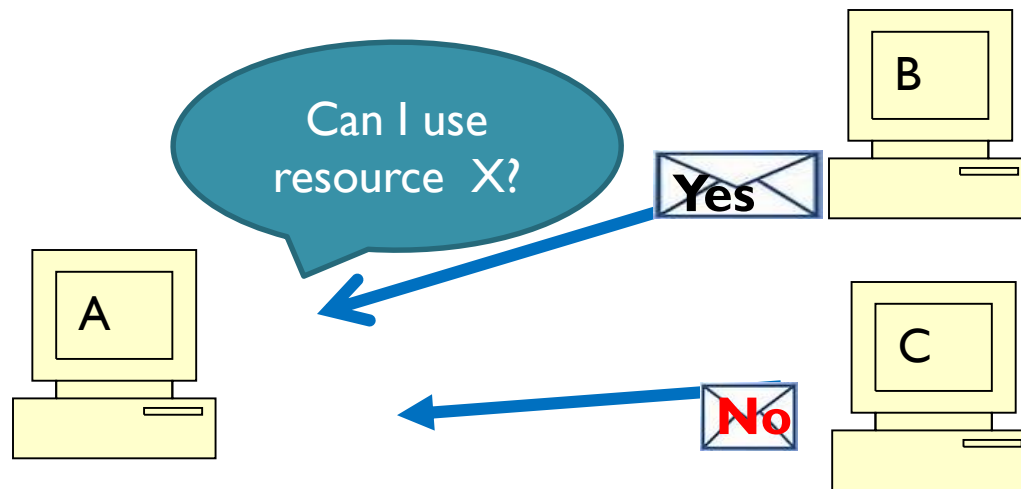


**Computer A** requests to use **resource X**



# How to ensure mutual exclusion?

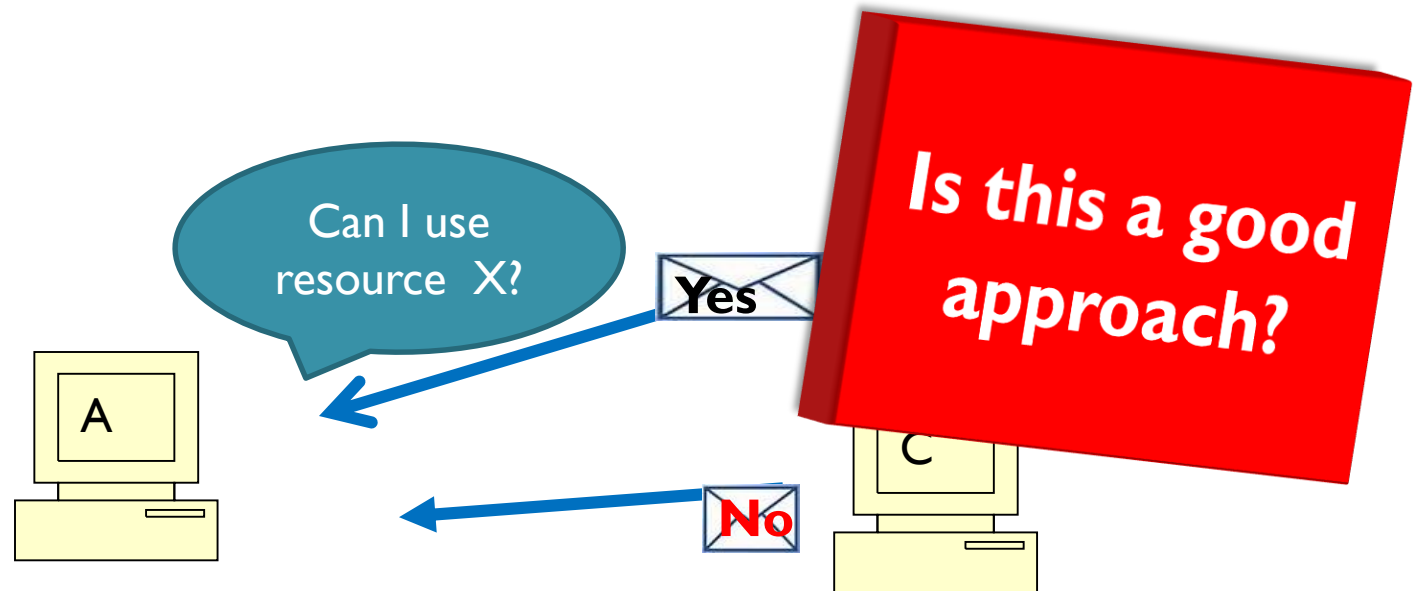
**Naive approach (质朴的方):** If a computer needs to access a resource (data), it sends a message to all the other computers to ask if it can use the resource.



**Computer B and C** answer the request made by **Computer A**.

# How to ensure mutual exclusion?

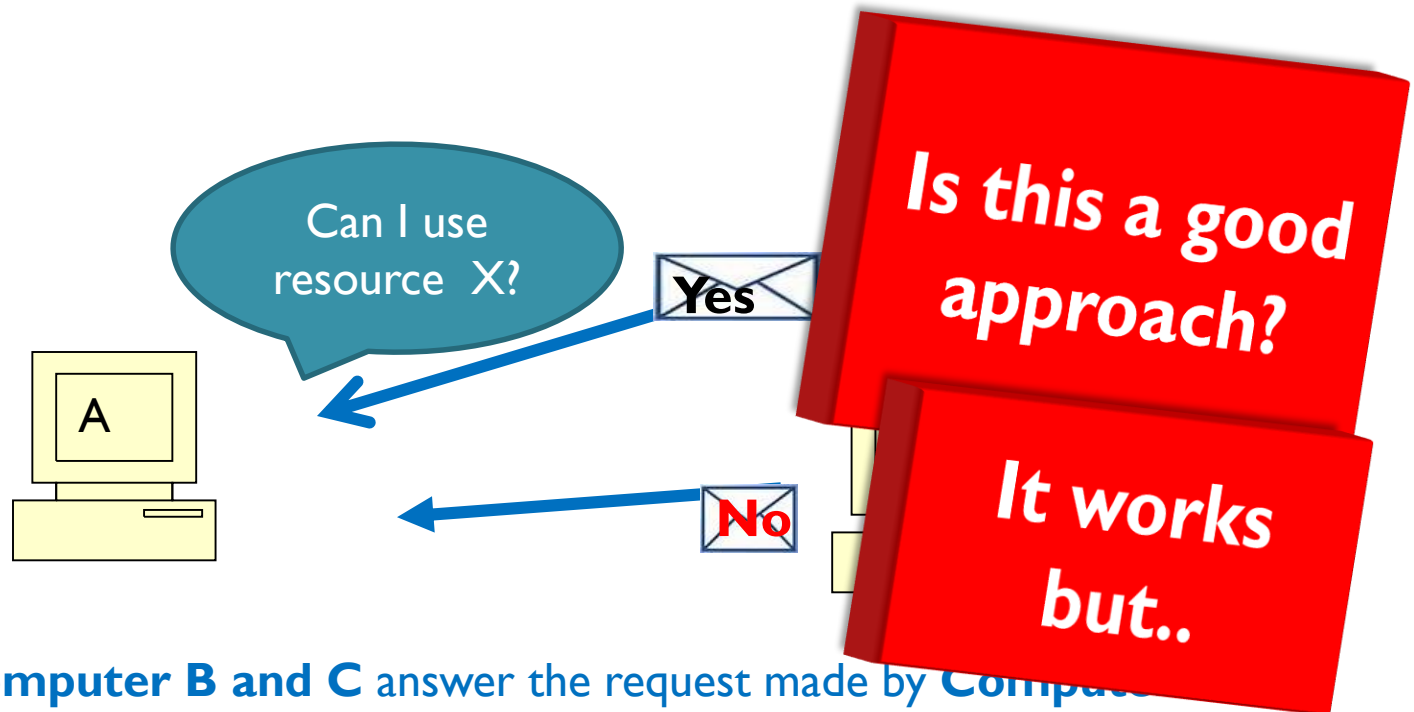
**Naive approach (质朴的方)**: If a computer needs to access a resource (data), it sends a message to all the other computers to ask if it can use the resource.



**Computer B and C** answer the request made by **Computer A**.

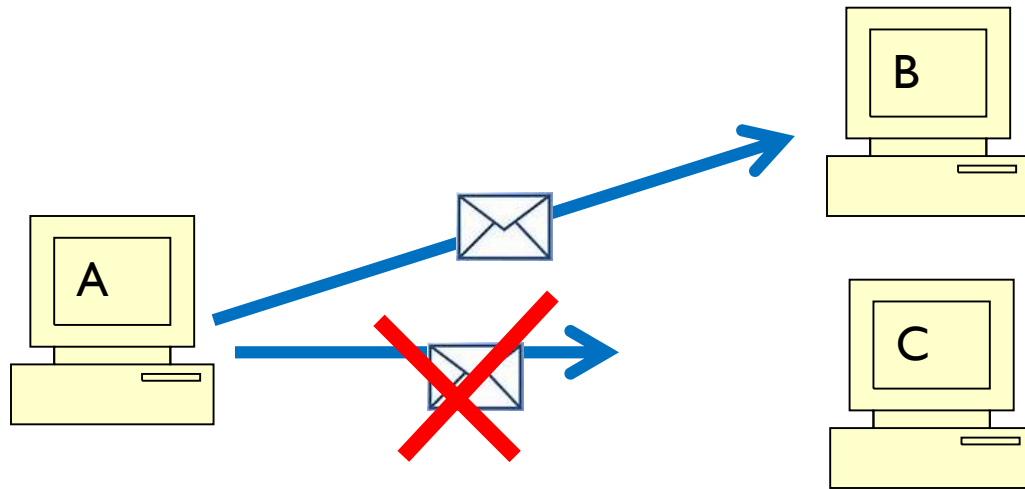
# How to ensure mutual exclusion?

**Naive approach (质朴的方):** If a computer needs to access a resource (data), it sends a message to all the other computers to ask if it can use the resource.



# Problem 1 – message loss (消息丢失)

If message(s) are lost, **computer A** could wait forever for an answer!



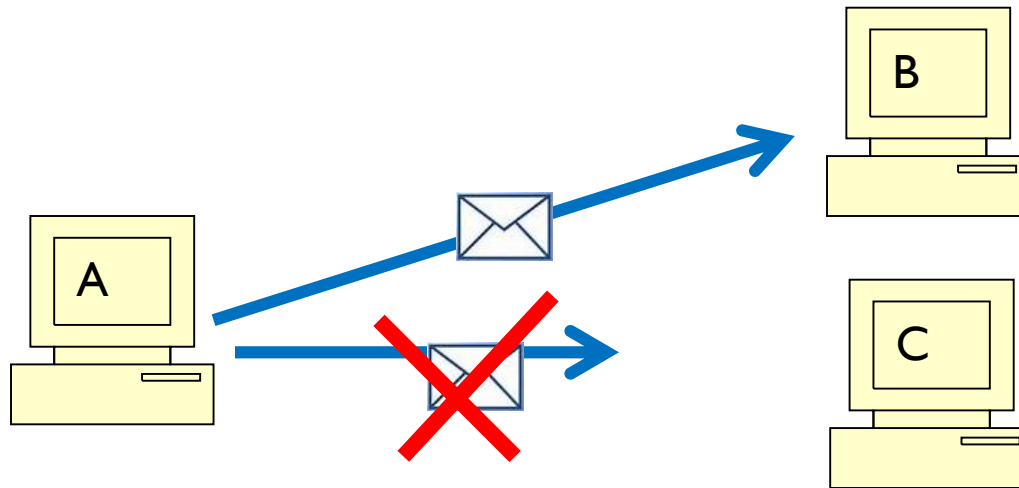
A message is lost!

# Problem 1 – message loss (消息丢失)

If message(s) are lost, **computer A** could wait forever for an answer!

**Solution:** use a **timeout** (超时).

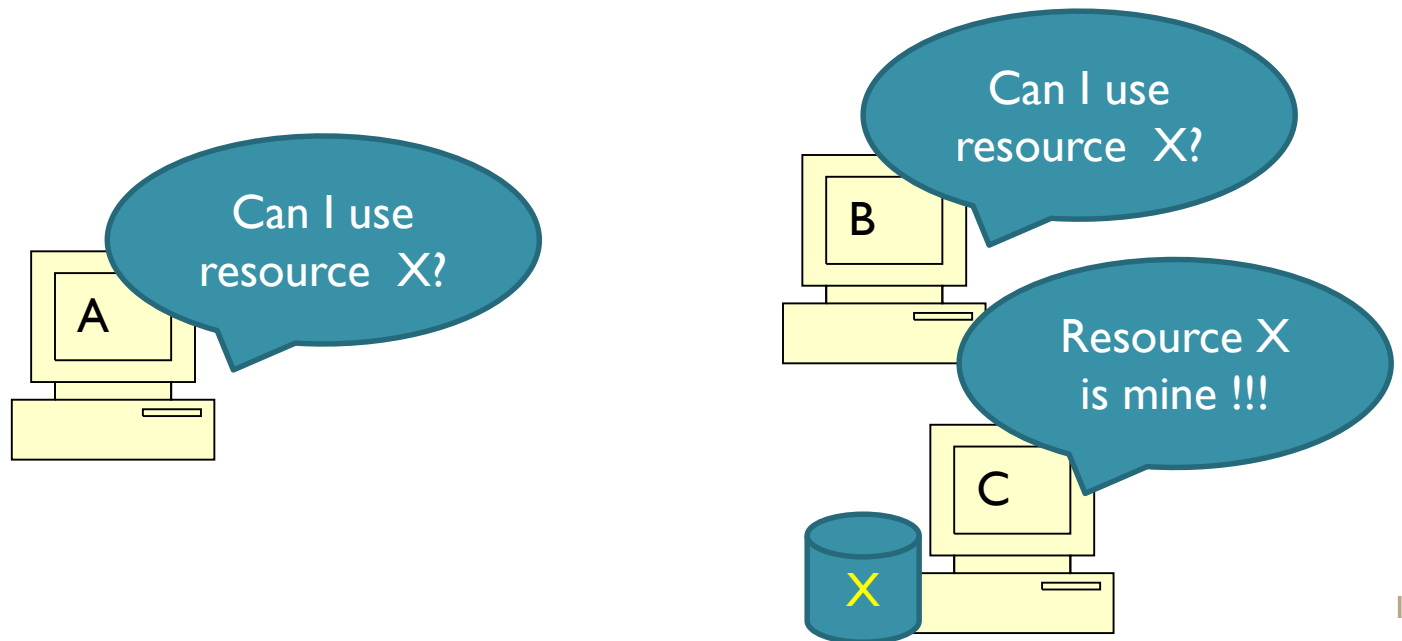
If **computer A** do not receive an answer within a predefined time limit (e.g. **5 seconds**), **computer A** sends the message again.



A message is lost!

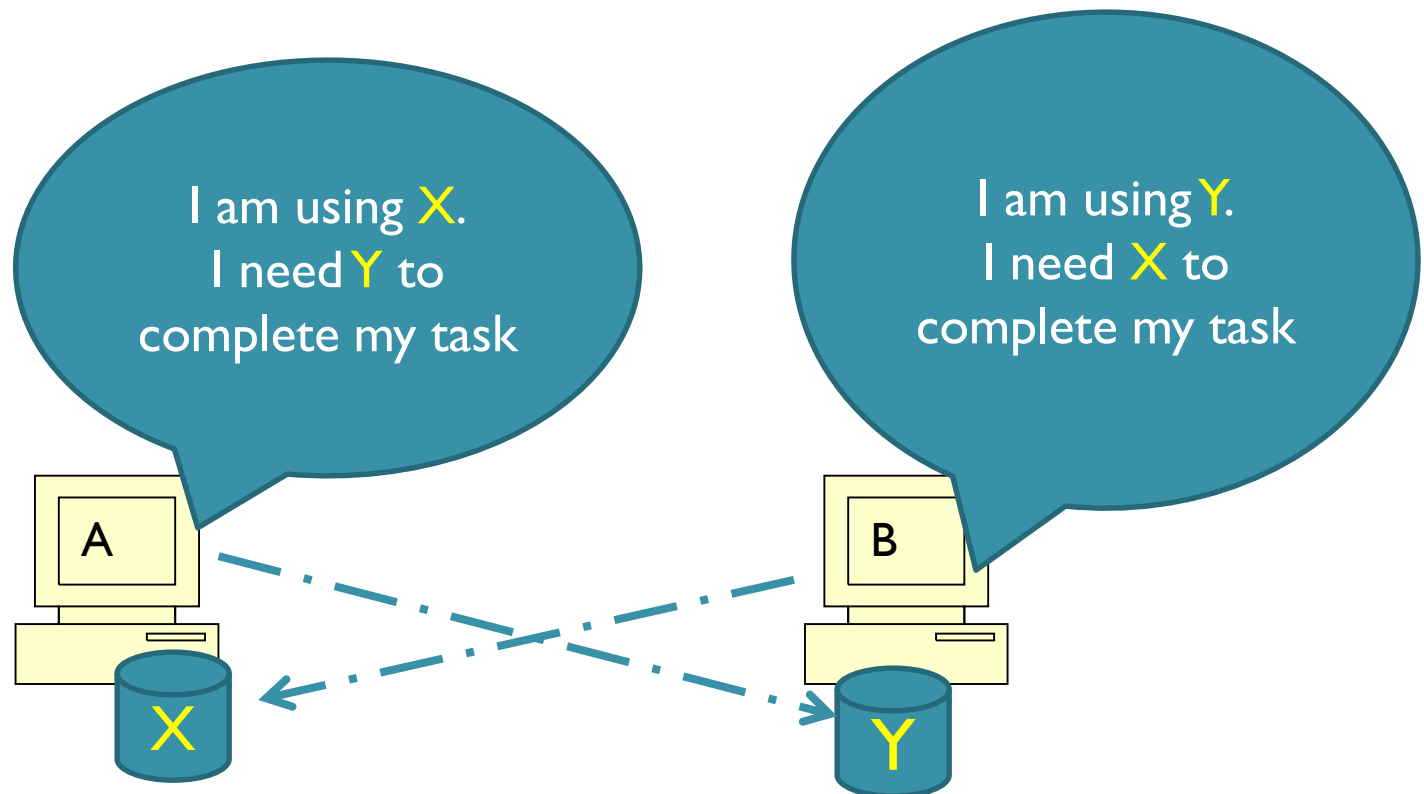
# Problem 2 – fairness (公平)

- **Fairness (公平)** is not guaranteed!
- **Fairness (公平)**: if a computer want to access a resource, **it will eventually be able to access the resource.**
- In other words, **no computer will wait forever** to access a resource.



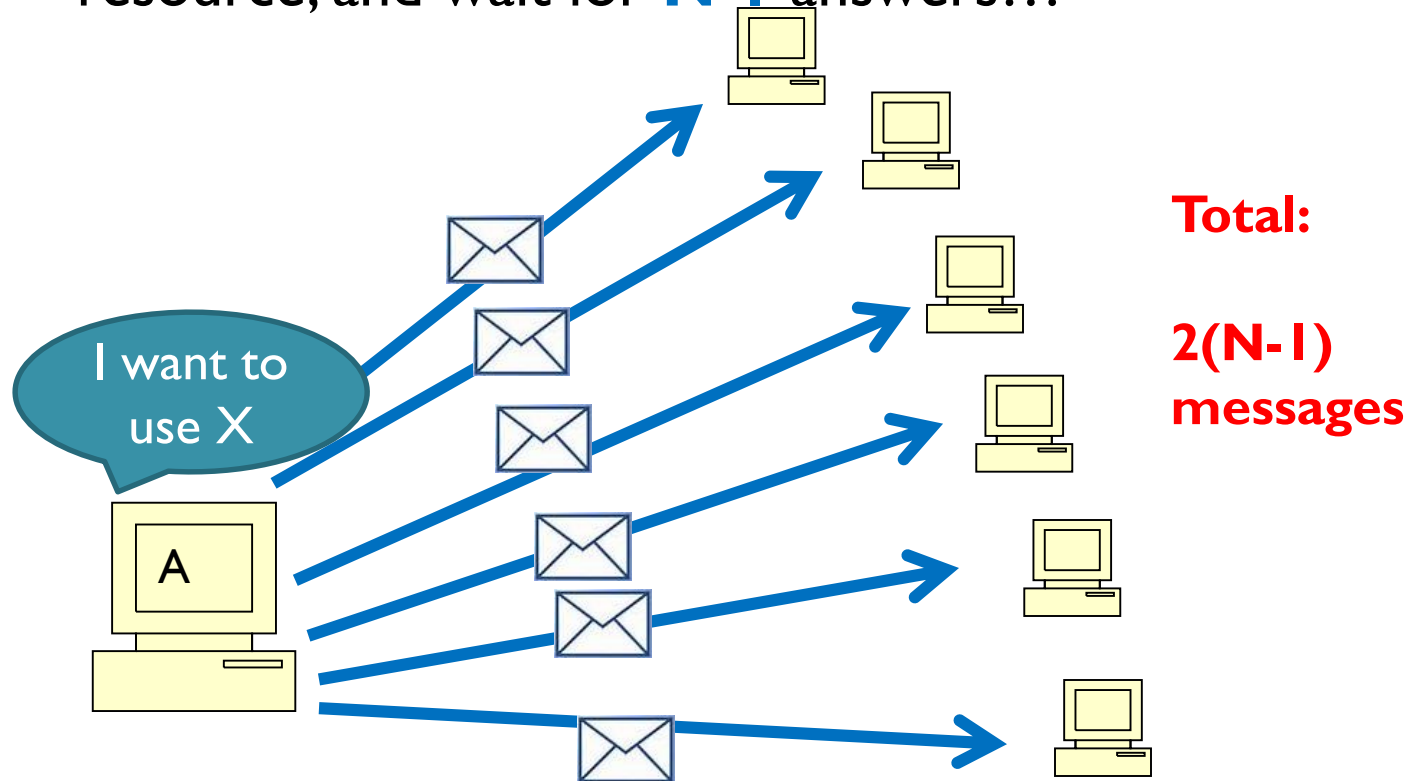
# Problem 3 – livelocks (活锁)

**Livelock (活锁):** Two or more computers wait forever because they need to use a resource that is used by the another computer.



# Problem 4 – too many messages!

- Many messages need to be sent.
- If there are  $N$  computers, then a computer must send  $N-1$  messages to other computers to ask to use a resource, and wait for  $N-1$  answers...





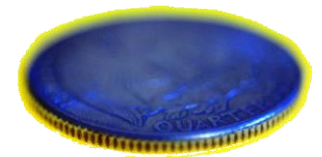
# Is there some better approaches?

Many approaches (*algorithms* 算法).

- Lamport's algorithm
- Ricart-Agrawal's algorithm
- Maekawa's algorithm
- Suzuki-Kasami's algorithm...

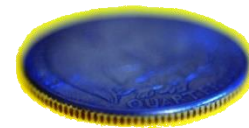
I will explain one →

° **THE  
SUZUKI-KASAMI  
ALGORITHM (算法)**



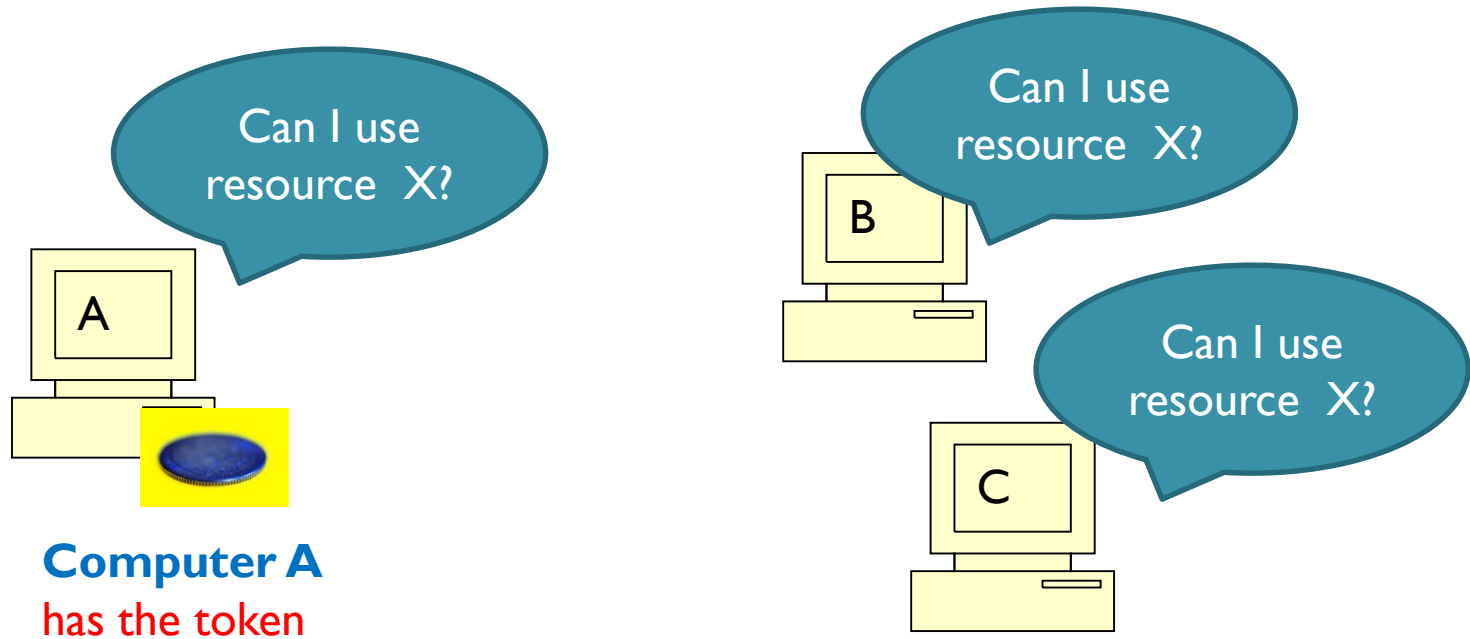
# Introduction

- Based on a **special type of message** called a **token (代币)**.
- There is always **only one token**.
- The token is sent from one computer to another.
- When a computer has the token, it can access any critical resources.
- When a computer does not have the token, it cannot access critical resources (it must wait until it has the token).

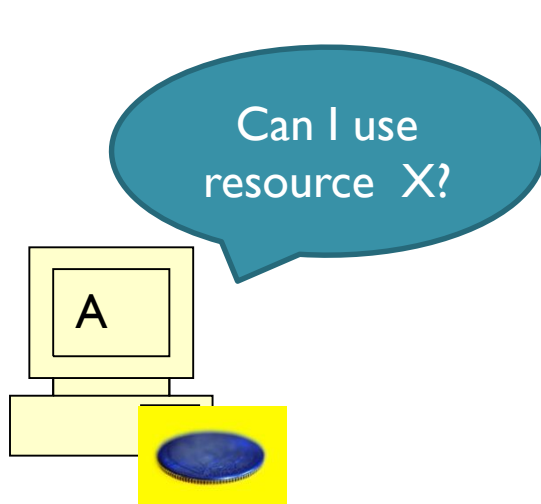


**Token (代币)**

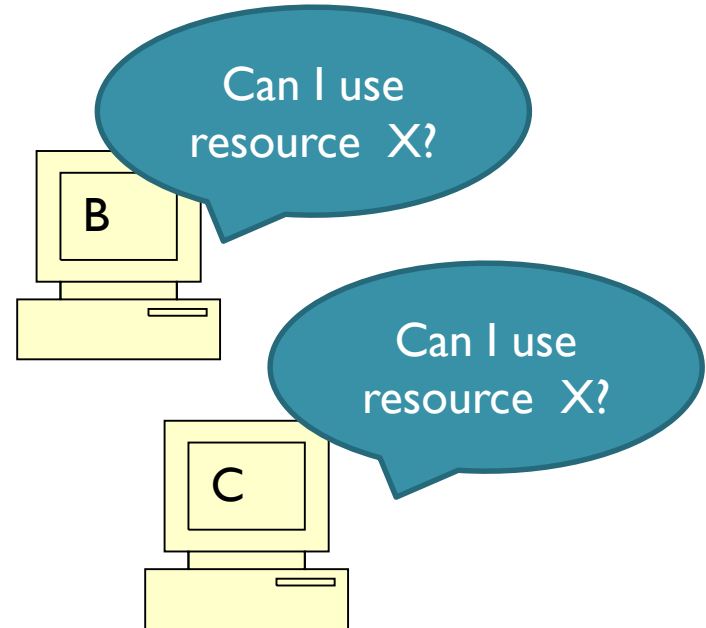
# Illustration of the basic idea



# Illustration of the basic idea

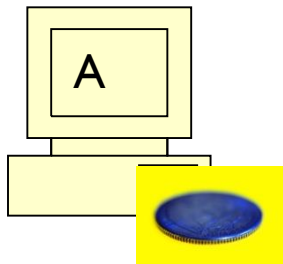


**Computer A** can use critical resources because he has the token.

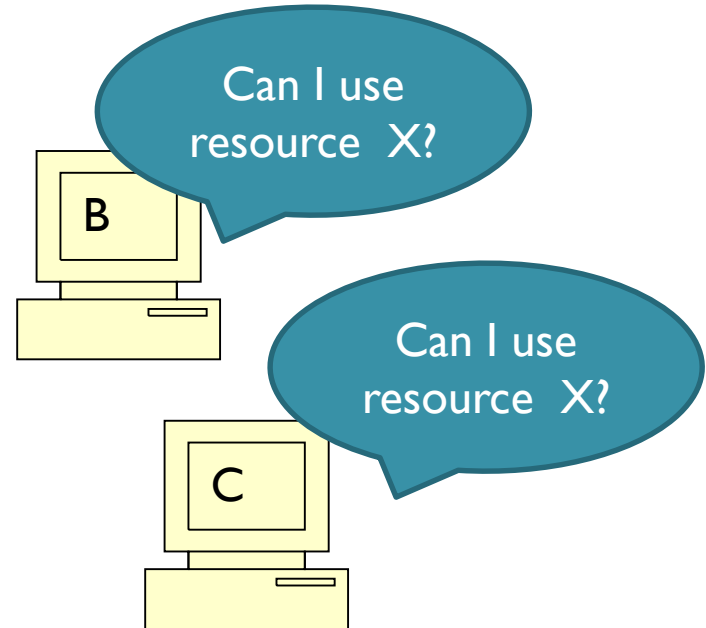


**Computer B** and **C** cannot use critical resources because they do not have the token.

# Illustration of the basic idea

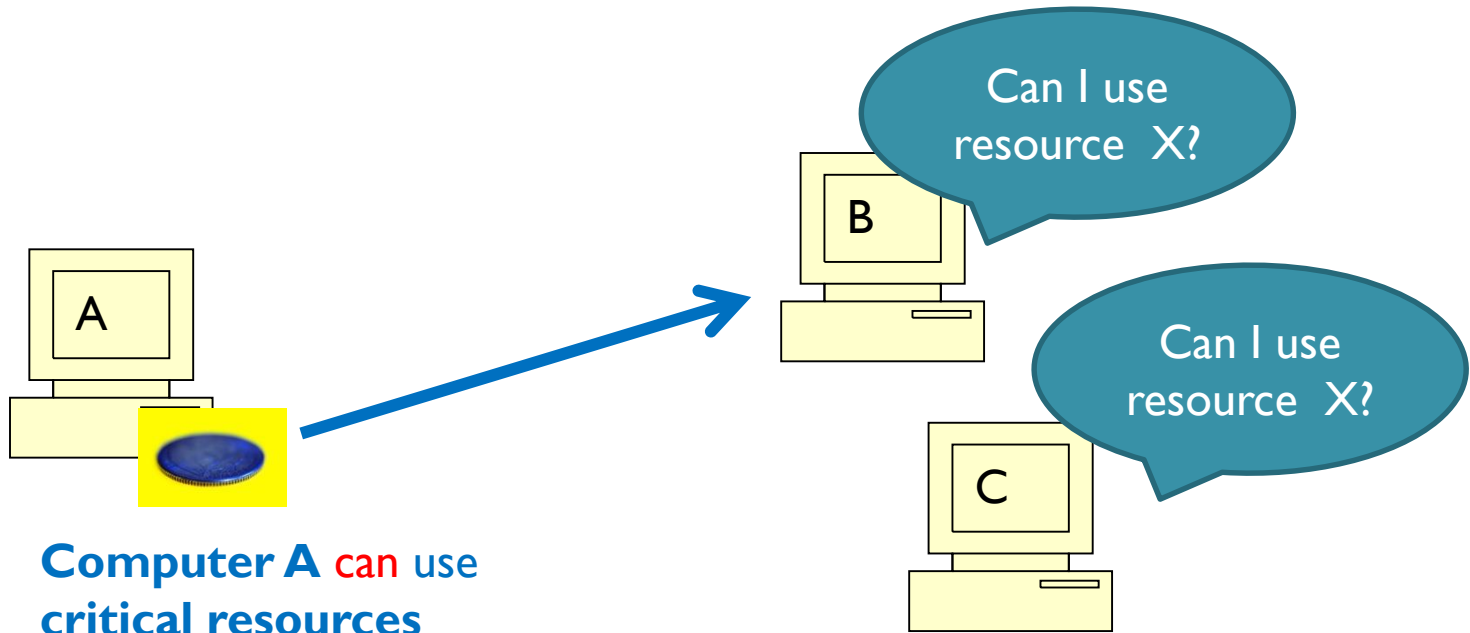


**Computer A** can use critical resources because he has the token.



**Computer B** and **C** cannot use critical resources because they do not have the token.

# Illustration of the basic idea

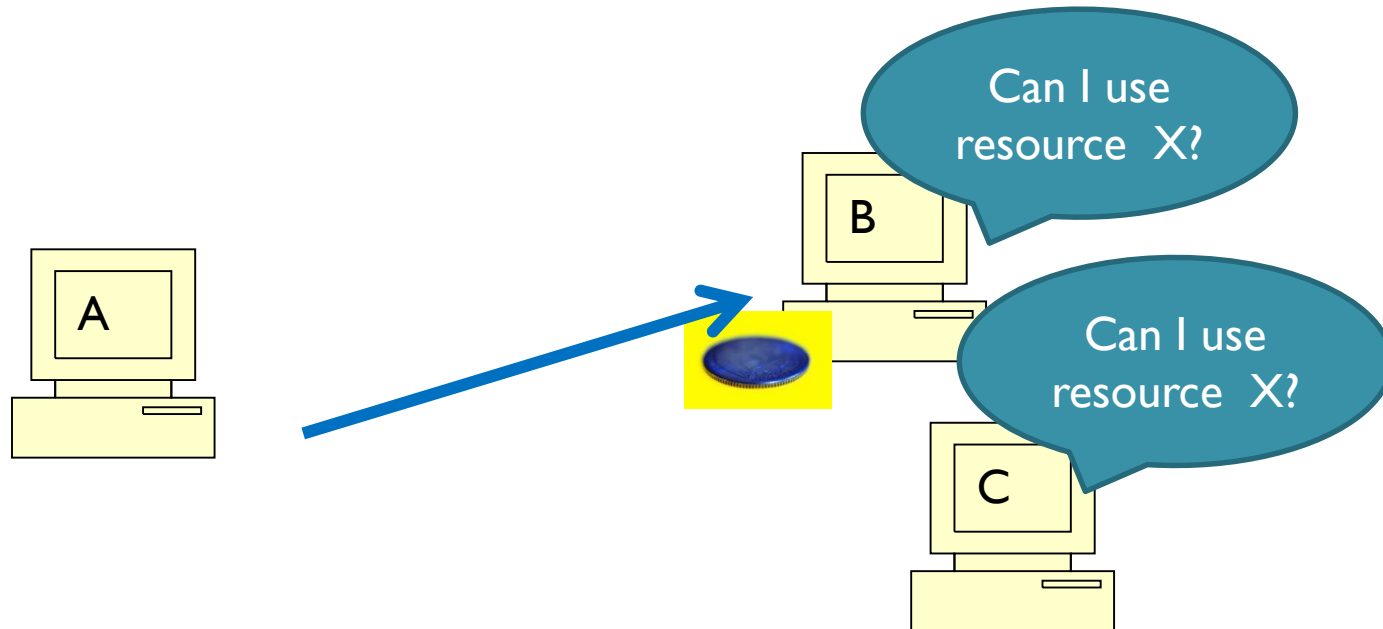


**Computer A** can use critical resources because he has the token.

**Computer B** and **C** cannot use critical resources because they do not have the token.

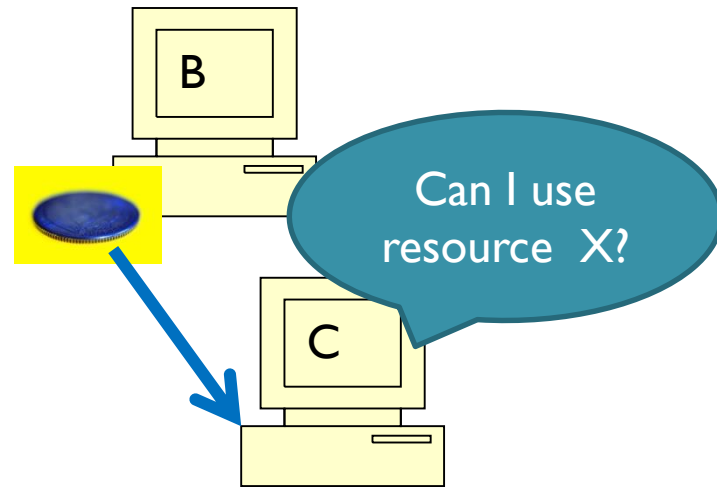
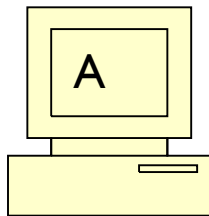
# Illustration of the basic idea

**Computer B** can use critical resources because he has the token.

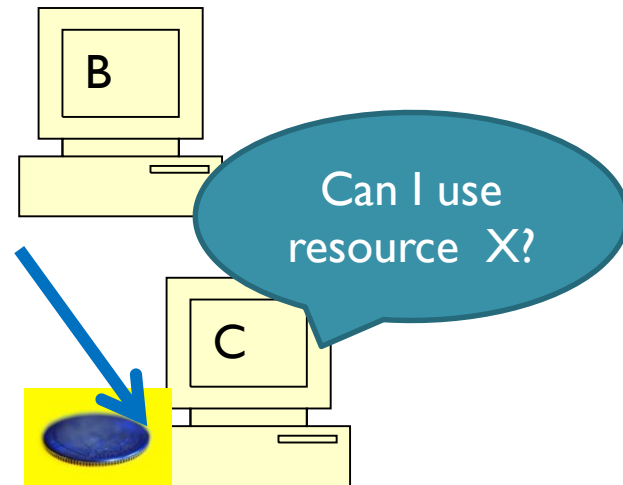
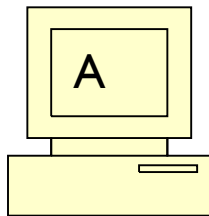




# Illustration of the basic idea



# Illustration of the basic idea



**Computer C** can use **critical resources** because he has the token.

# Does it work?

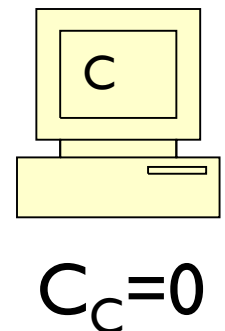
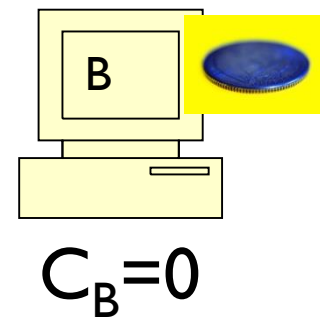
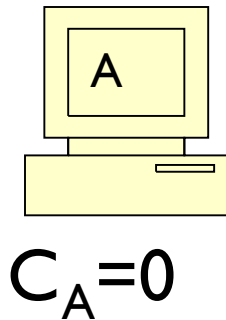
- **Does it guarantee mutual exclusion?**
  - **Yes**, since there is only one token, no more than one computer can access critical resources at the same time.
  - **What happen if the token is **lost**?**
    - After a timeout (超时), a computer creates a new token.

But can we ensure **fairness** (公平)?  
can we avoid **livelocks** (活锁)? →

# Sequence numbers (序列号) and token (代币)

Each computer has a **sequence number** (序列号).

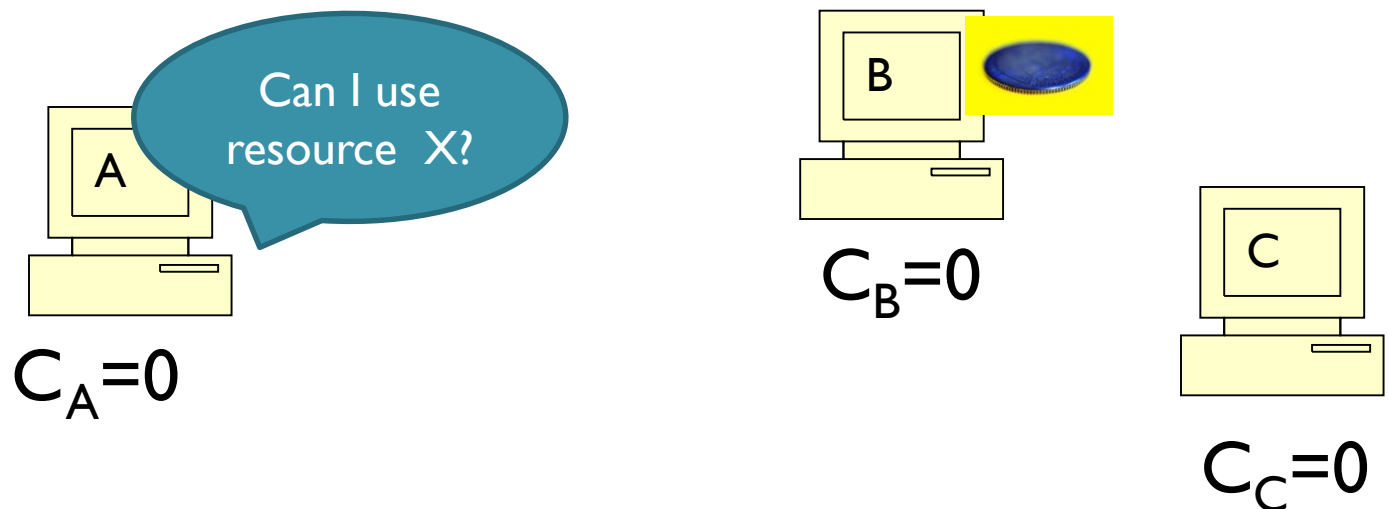
One computer has the **token** (代币)



# To access critical resources

**If** a computer needs to access a critical resource:

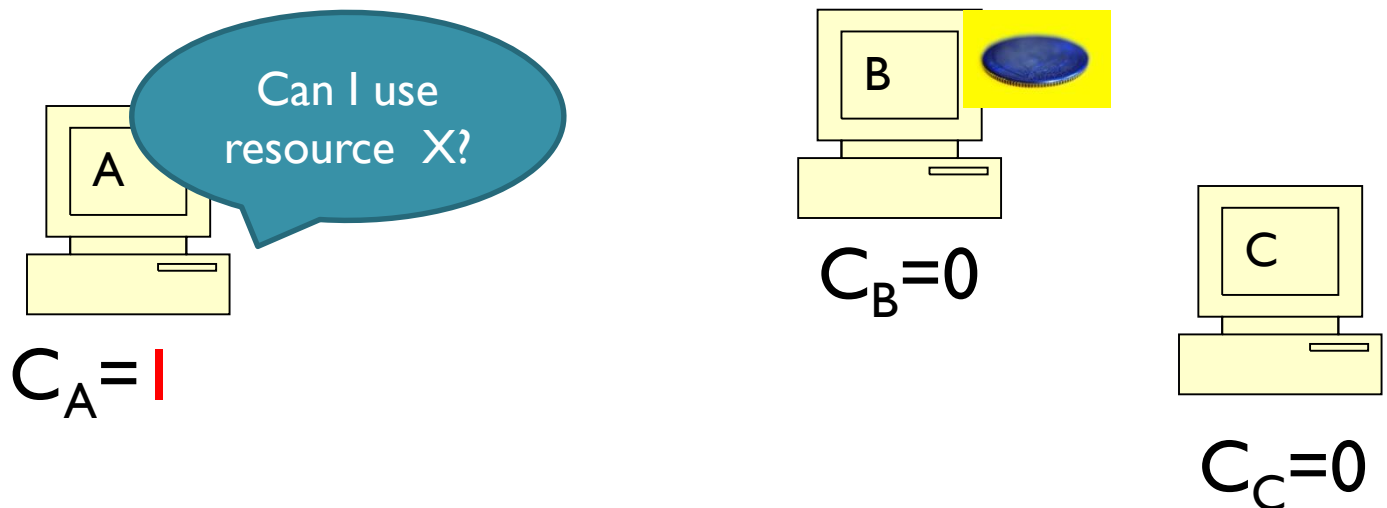
1. It increases its sequence number by 1.
2. It sends a **REQUEST** (a message) to the other computers, indicating its sequence number.



# To access critical resources

**If** a computer needs to access a critical resource:

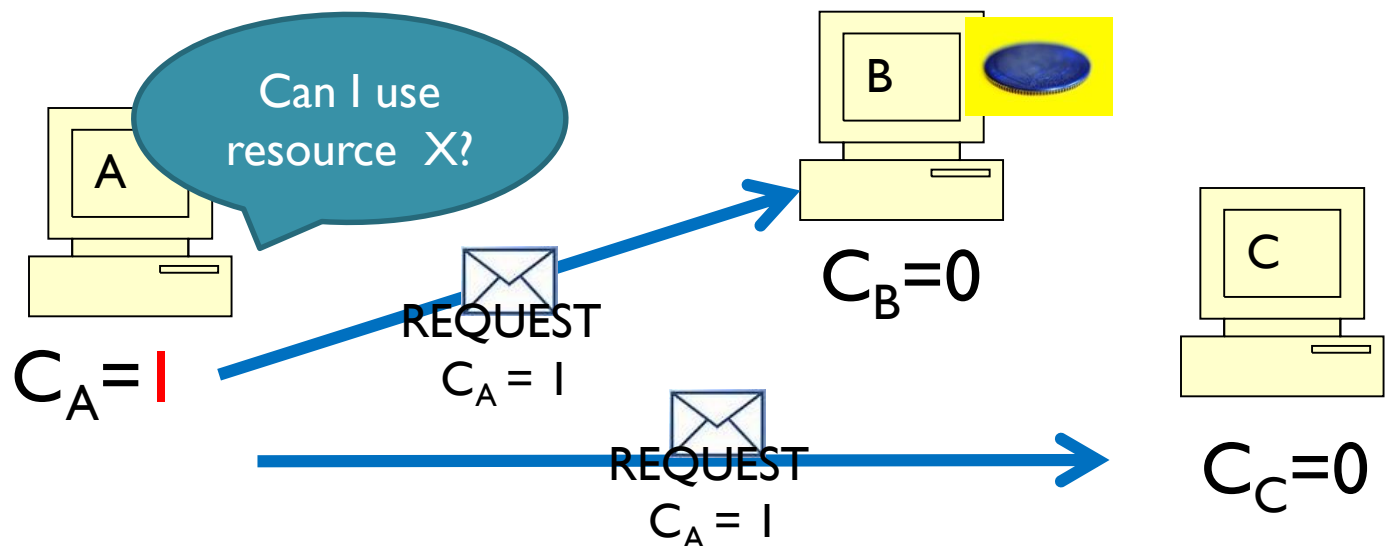
1. It increases its sequence number by 1.
2. It sends a **REQUEST** (a message) to the other computers, indicating its sequence number.



# To access critical resources

If a computer needs to access a critical resource:

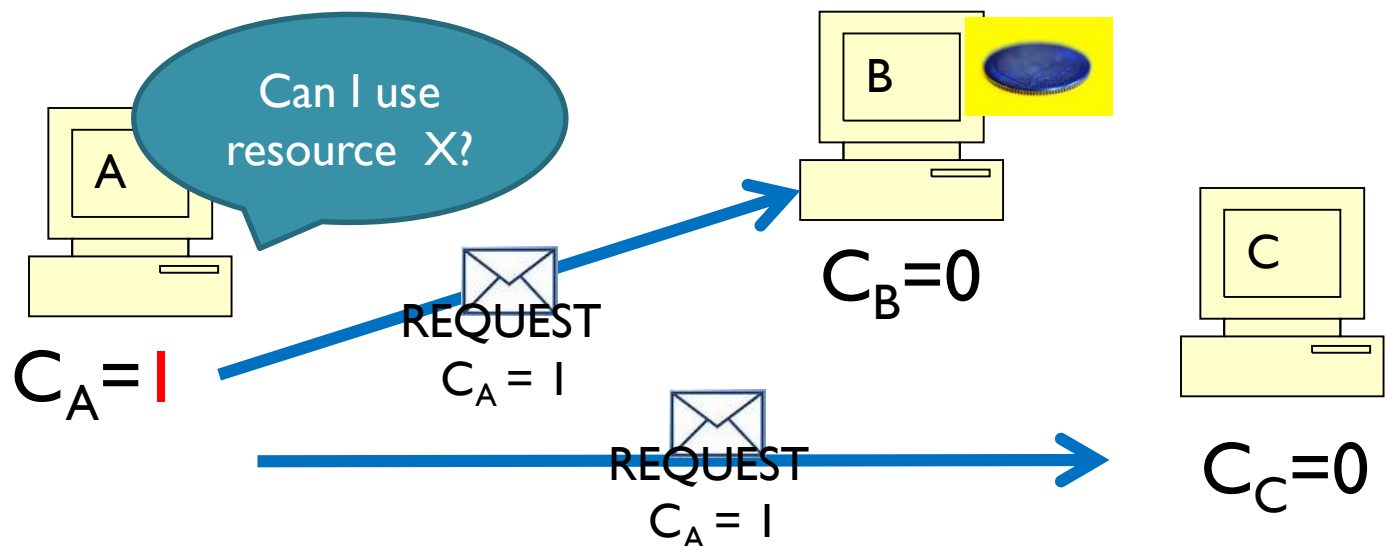
1. It increases its sequence number by 1.
2. It sends a **REQUEST** (a message) to the other computers, indicating its sequence number.



# Answering a request message

If a computer receives a **REQUEST** message and has the **token**:

1. It takes note of the request.
2. When it finishes using critical resources, it sends the token to one of the computers that had sent a **REQUEST** message.

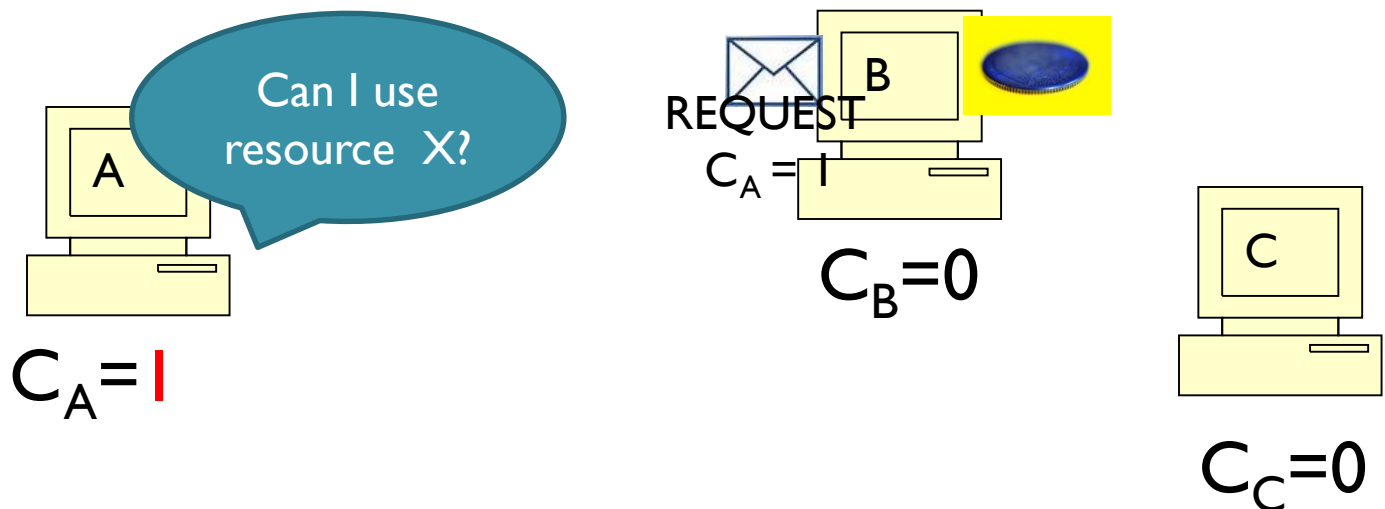




# Answering a request message

If a computer receives a **REQUEST** message and has the **token**:

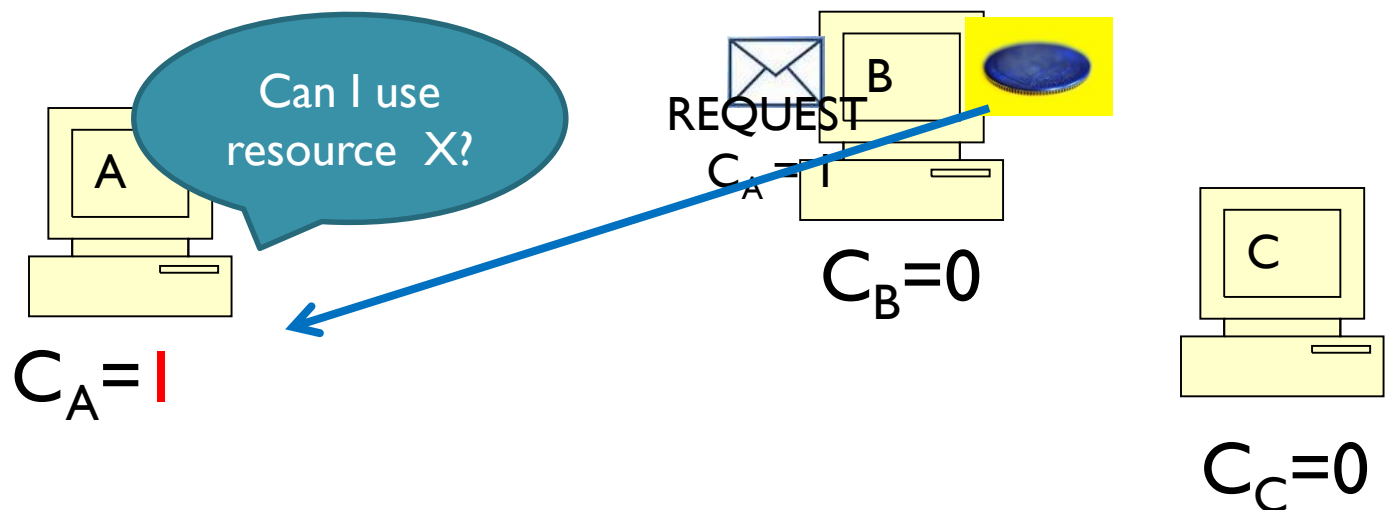
1. It takes note of the request.
2. When it finishes using critical resources, it sends the token to one of the computers that had sent a **REQUEST** message.



# Answering a request message

If a computer receives a **REQUEST** message and has the **token**:

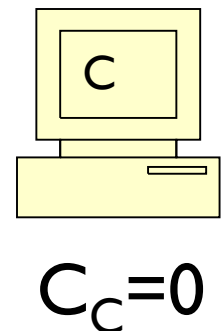
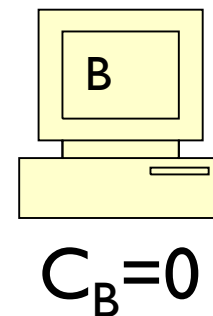
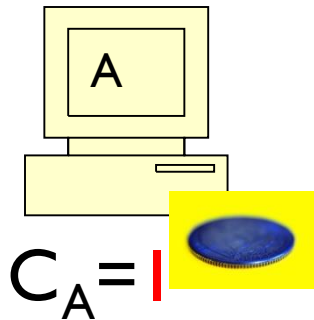
1. It takes note of the request.
2. When it finishes using critical resources, it sends the token to one of the computers that had sent a **REQUEST** message.

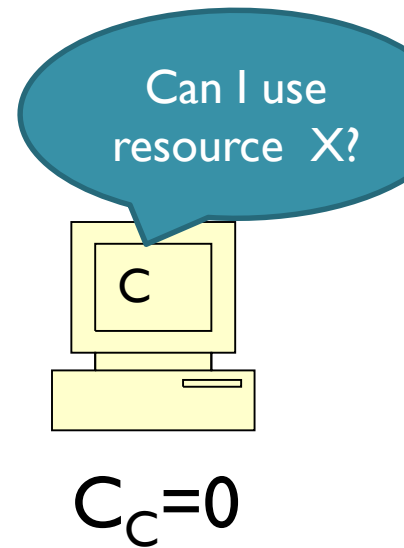
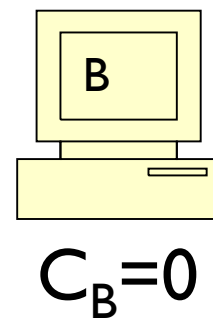
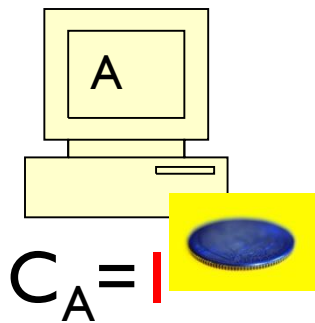


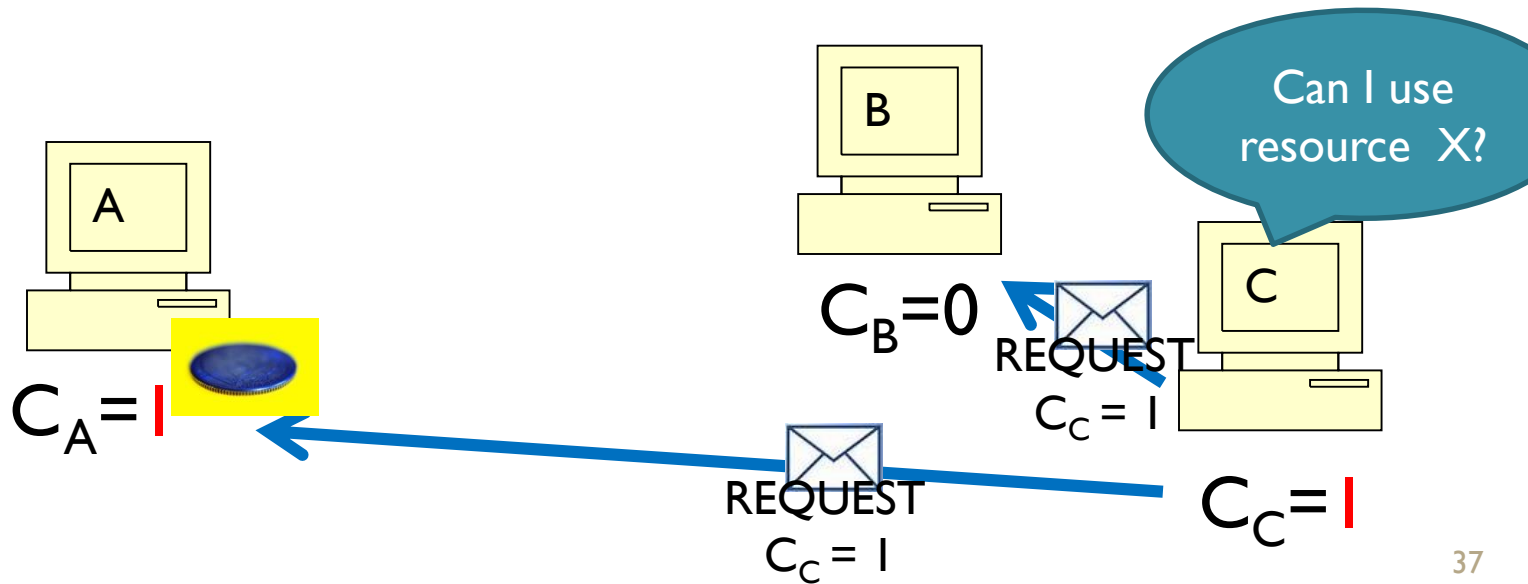
# Answering a request message

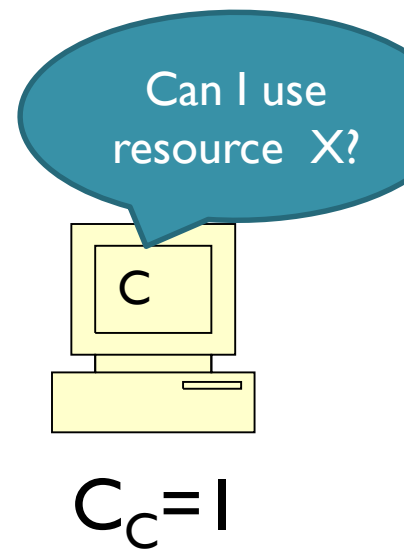
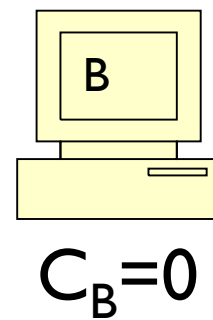
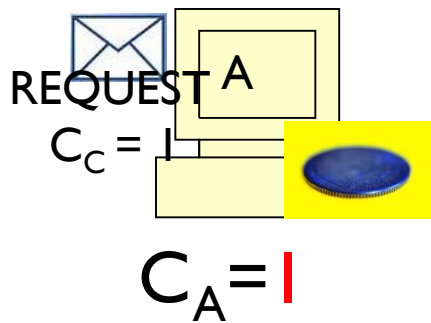
If a computer receives a **REQUEST** message and has the **token**:

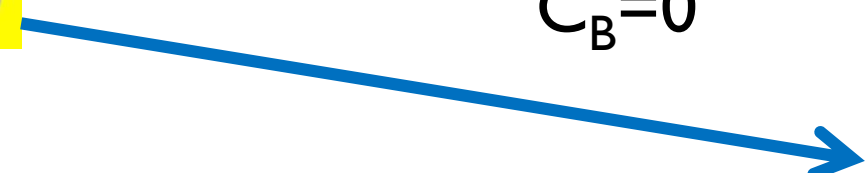
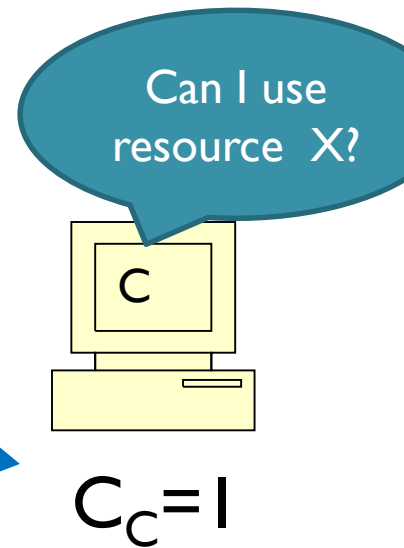
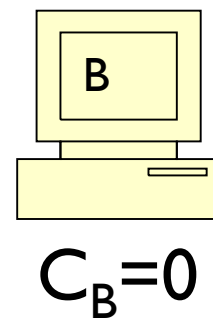
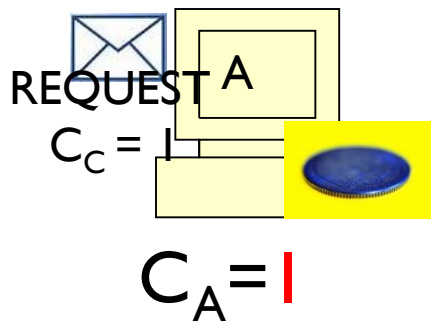
1. It takes note of the request.
2. When it finishes using critical resources, it sends the token to one of the computers that had sent a **REQUEST** message.

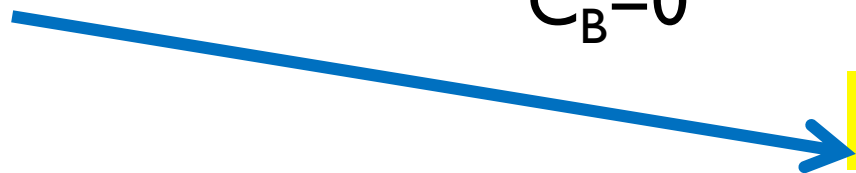
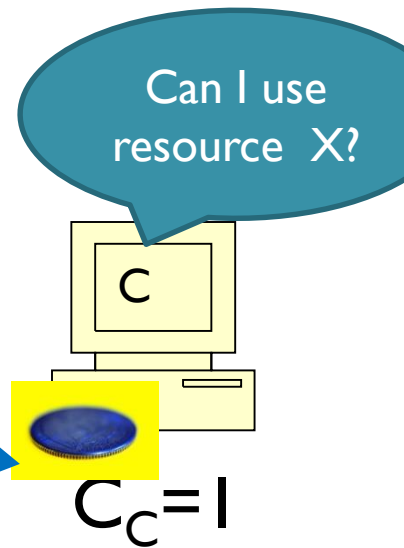
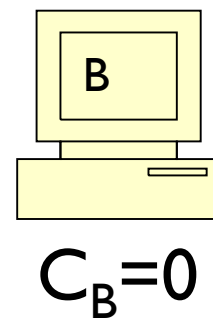
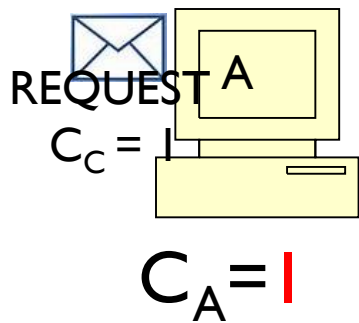




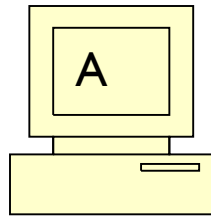




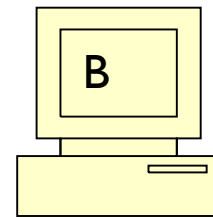




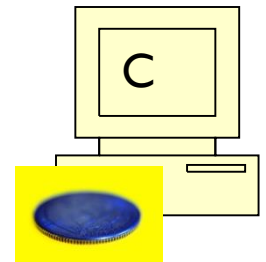




$$C_A = 1$$



$$C_B = 0$$



$$C_C = 1$$



# Problem 1

- Because of **message transmission delays** (传输延时), a computer may receive a request that has already been satisfied.
- If the token is sent to a computer that do not need it, time is wasted! (other computers have to wait).
  - **How to distinguish between *new* and an *old* requests?**
  - **How to know if a request has been satisfied?**

## Problem 2

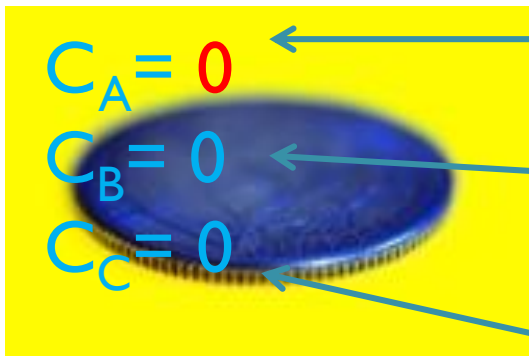
- When a computer does not need the token anymore, he must send the token to another computer.
- **How to choose the next computer?**

We want to ensure **fairness** (公平)  
and avoid **livelocks** (活锁).

- **SOLUTION →**

# Solution

**Key idea:** put information in the token!



The sequence number of the last satisfied request for **computer A**

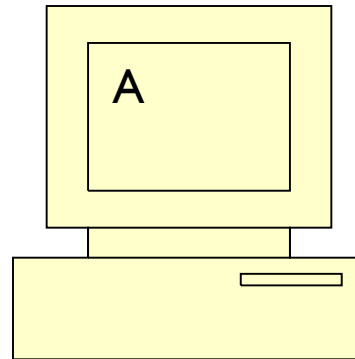
The sequence number of the last satisfied request for **computer B**

The sequence number of the last satisfied request for **computer C**

**Hence, the token keeps track of all the requests that have been satisfied!**

# Solution

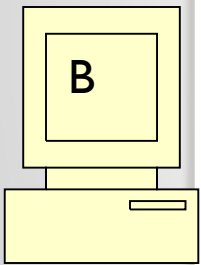
When a computer has the token and access critical resources, it increases its sequence number in the token.



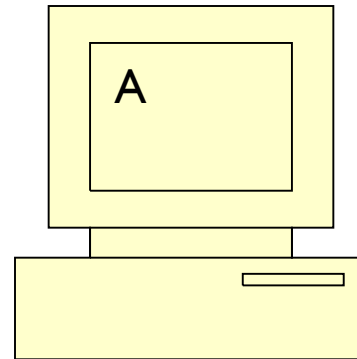
$$C_A = 1$$

# Solution


When a computer has the token and access critical resources, it increases its sequence number in the token.



$$C_B = 1$$

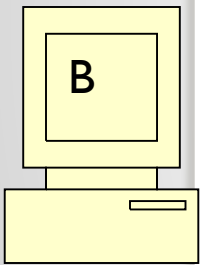


$$C_A = 1$$

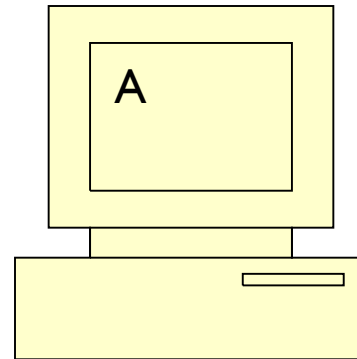
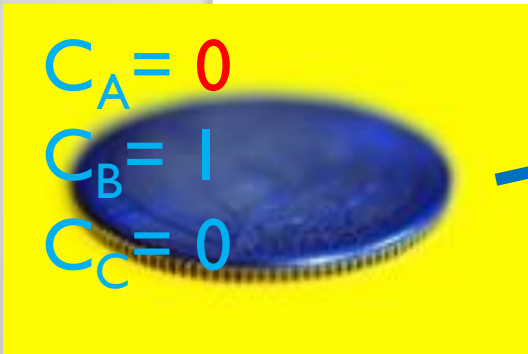
A blue coin with a textured surface is shown on a yellow background.
$$C_A = 0$$
$$C_B = 1$$
$$C_C = 0$$

# Solution

When a computer has the token and access critical resources, it increases its sequence number in the token.



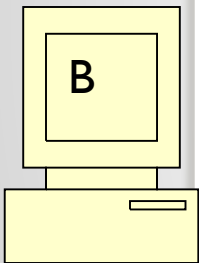
$$C_B = 1$$



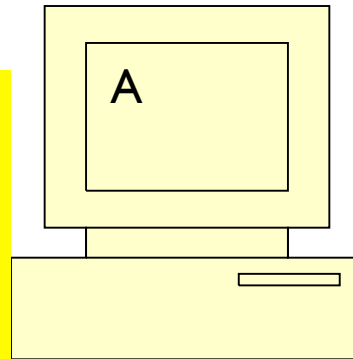
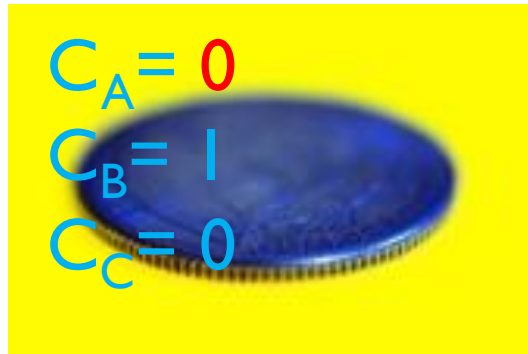
$$C_A = 1$$

# Solution

When a computer has the token and access critical resources, it increases its sequence number in the token.



$$C_B = 1$$

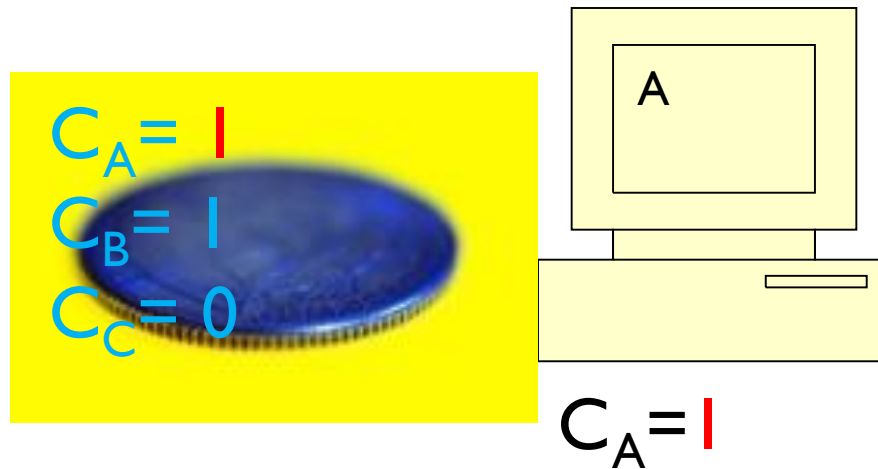


$$C_A = 1$$



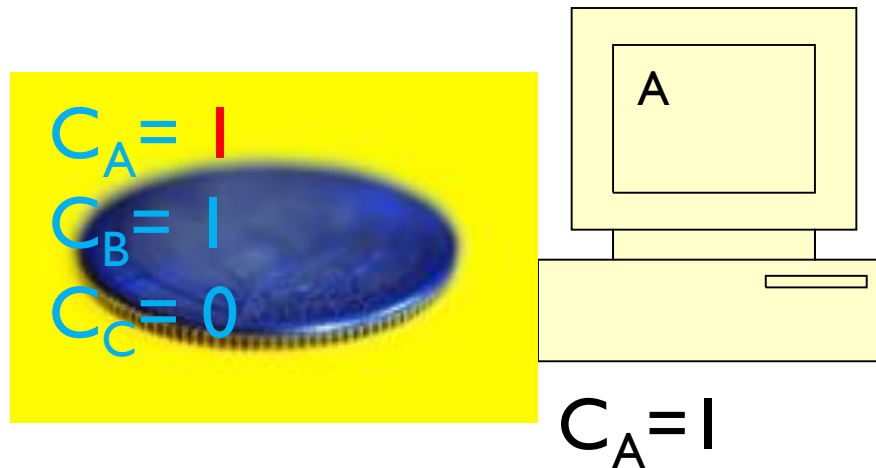
# Solution

When a computer has the token and access critical resources, it increases its sequence number in the token.



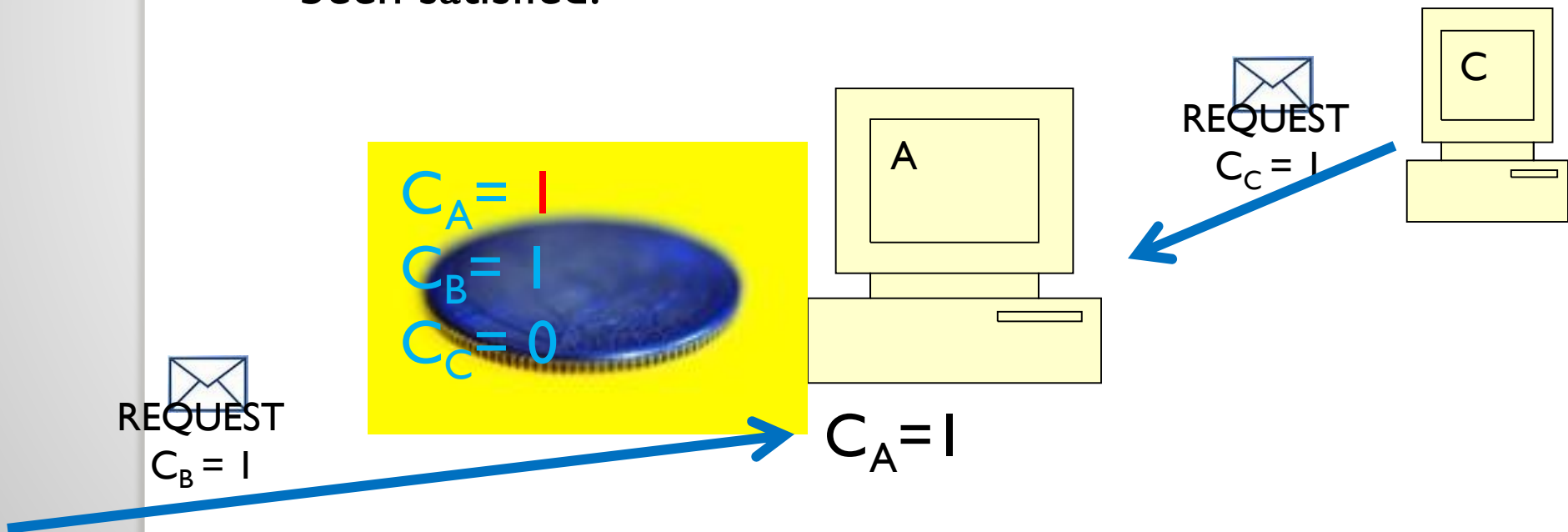
# Solution

- Moreover, **each computer keeps all the requests that it receives.**
- The computer having the token can compare each request with the token to know which requests have been satisfied.



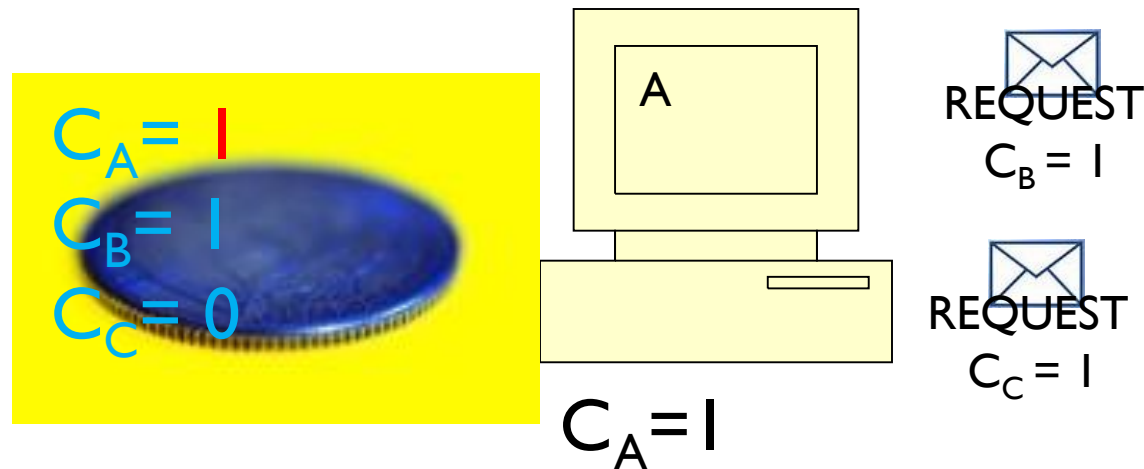
# Solution

- Moreover, **each computer keeps all the requests that it receives.**
- The computer having the token can compare each request with the token to know which requests have been satisfied.



# Solution

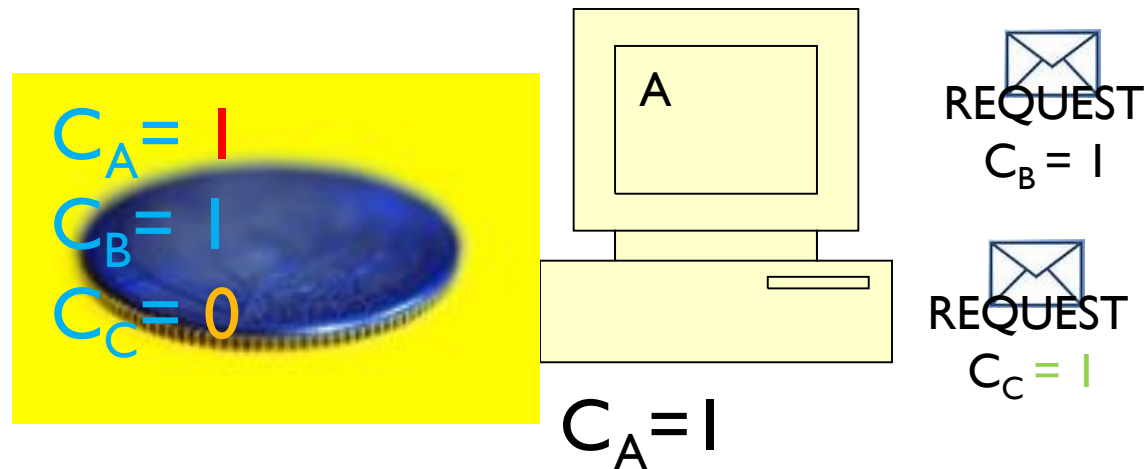
- Moreover, **each computer keeps all the requests that it receives.**
- The computer having the token can compare each request with the token to know which requests have been satisfied.



**Computer A should send the token to which computer next?**

# Solution

- Moreover, **each computer keeps all the requests that it receives.**
- The computer having the token can compare each request with the token to know which requests have been satisfied.



**Answer:** Computer C because  $0 \leq 1$

# How to ensure fairness (公平)?

**To ensure that each computer will eventually receive the token:**

- A **queue** (队列) is added to the token.
- Before a computer send the token to another computer, it puts all the non-answered requests in the queue inside the token.
- The token is sent to the first computer in the queue that has a non-answered request.

# Does it work well?

- Assume that there are **N** computers.
- **Computer A** wants to access critical resource,
  - it will send a **REQUEST** message to the **N-1** other computers.
  - Eventually another computer will send the token to **Computer A**.
- **N** messages are sent.
- At most **N-1** computers will have the token before computer **A** receives the token.

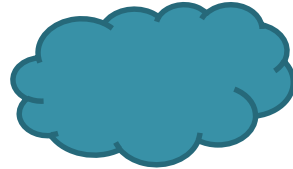
# Comparison

- **Naive approach**
  - $2(N - 1)$  messages
  - does not guarantee fairness
- **Algorithm of Suzuki-Kasami**
  - $N$  messages,
  - guarantees fairness.

**Note:** more messages may be needed if some messages are lost..

END





° **CLOUD  
INFRASTRUCTURE**  
(云基础结构)

**CHAPTER 3**

# Introduction

In the United States (2010):

- **Startup companies (初创公司) *using the cloud:***  
100,000\$ per month on average  
10 engineers,
- **Startups companies (初创公司) *not using the cloud:***  
2,000,000 \$ for computing infrastructure  
60 engineers

# Introduction

We will discuss the **infrastructure** (基础设施) of some popular **cloud providers** (云提供商):

- **Amazon**: pioneer for the “*Infrastructure-as-a-service*” model
- **Microsoft**: focuses on “*software-as-a-service*” and “*platform-as-a-service*”.

# Cloud computing at Amazon

- Initially, **Amazon** created its **cloud** to support its e-commerce business (电子商务): **selling books, clothing, foods, etc.**
- Then, Amazon decided to use its infrastructure to provide cloud computing services to other enterprises and organizations



# Cloud computing at Amazon

- **~2000:** Amazon created **Amazon Web Service (AWS)** - <https://aws.amazon.com/>.
  - Based on the **Infrastructure-as-a-service** model.
  - **It offers** computers, storage space, and a high-speed network.
  - The user pays to use a **virtual machine** (虚拟机).
  - The user installs an operating system (操作系统) and his own applications (应用) on the virtual machine.
- **~2012:** **AWS** is used in more than 200 countries.

# What is a virtual machine (虚拟机)?

It is an application that works like a separate computer inside the main computer.



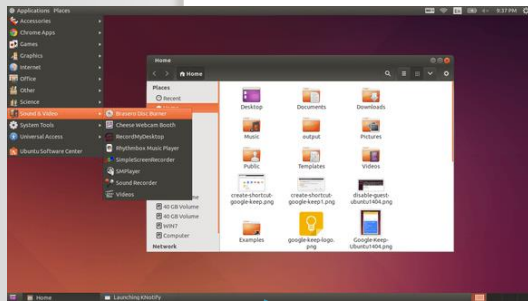
# Amazon EC2 (Elastic Compute Cloud)

**EC2** is a service offered by the **Amazon cloud** for running virtual machines in the cloud.

- The user must create a **virtual machine image** (虚拟机镜像) (called **AMI, Amazon Machine Image**).
- It contains the operating system (操作系统) and the application(s) that the user wants to run.
- The user can start several virtual machines (**instances** - 实例) using an image.
- The user can stop a virtual machine.

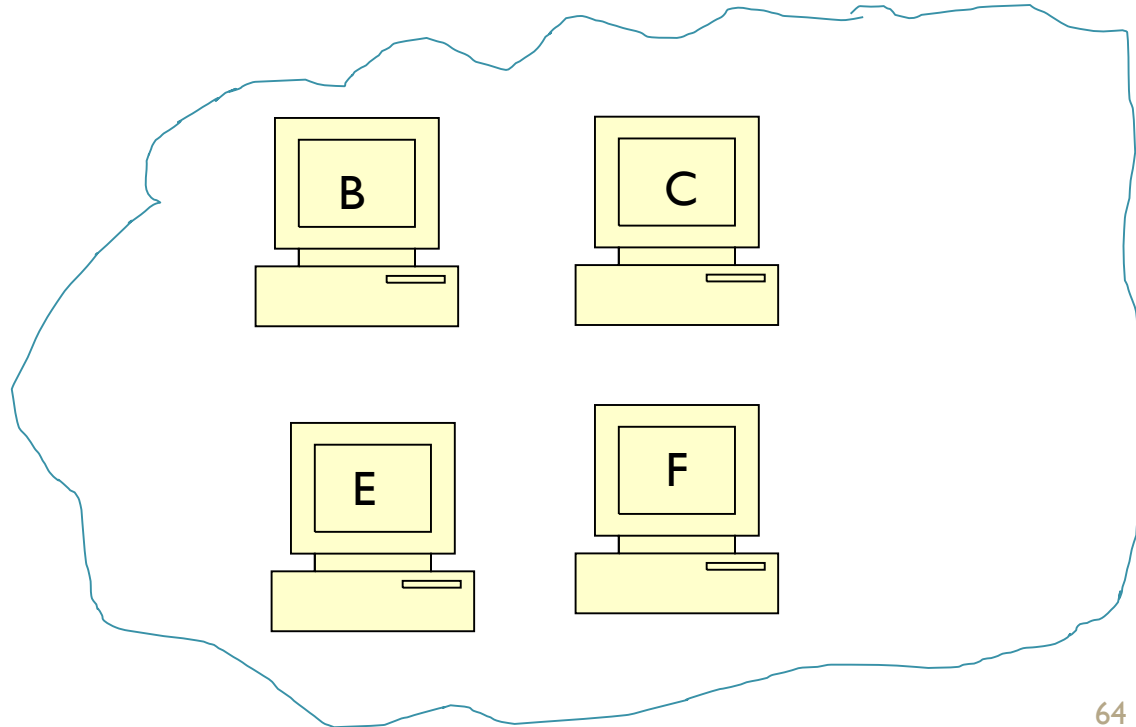
# Illustration

An **image** (虚拟机镜像) is the state of a computer that has been saved into a file



User

Image  
AMI



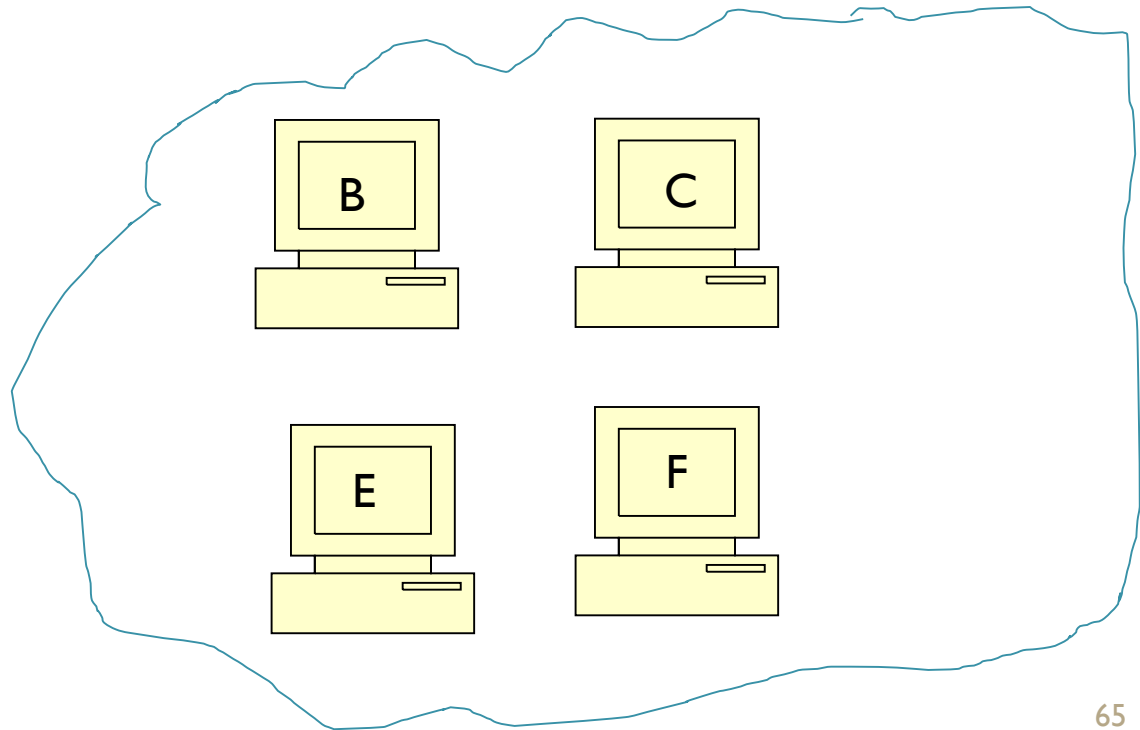


# Illustration



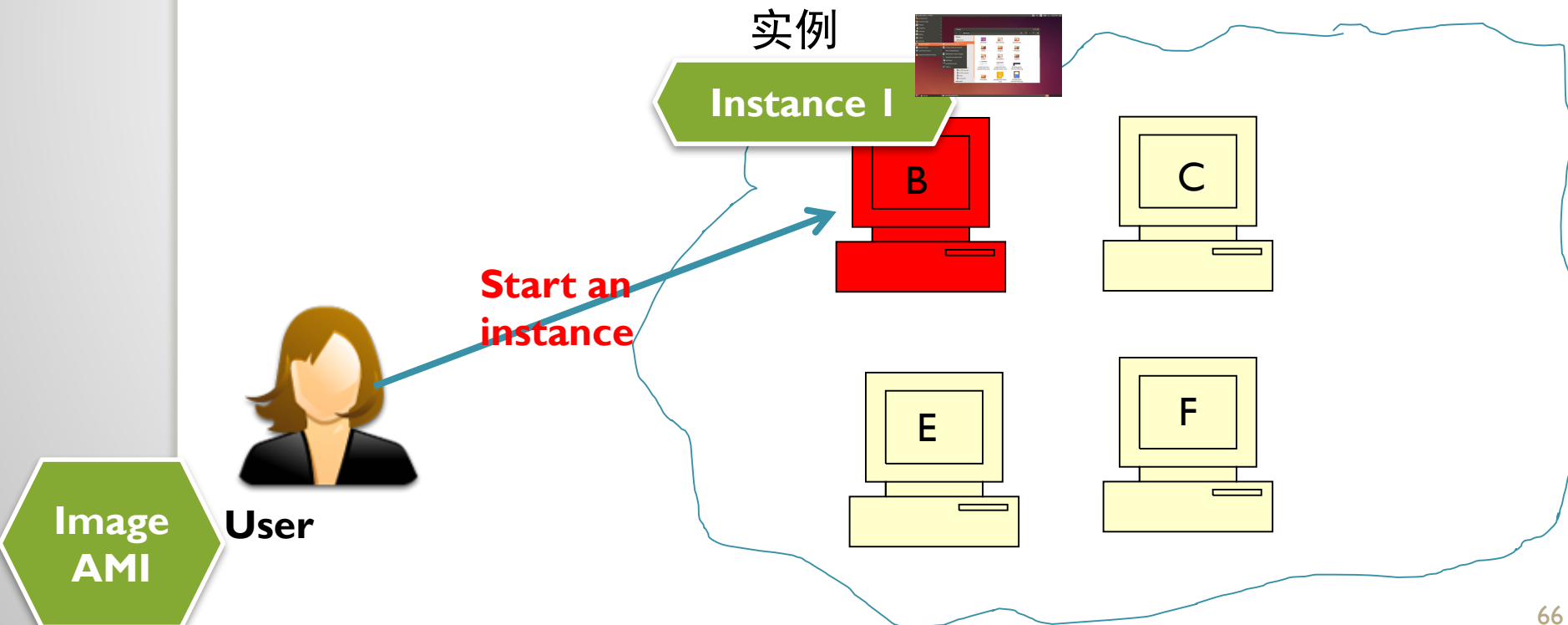
**User**

**Image  
AMI**



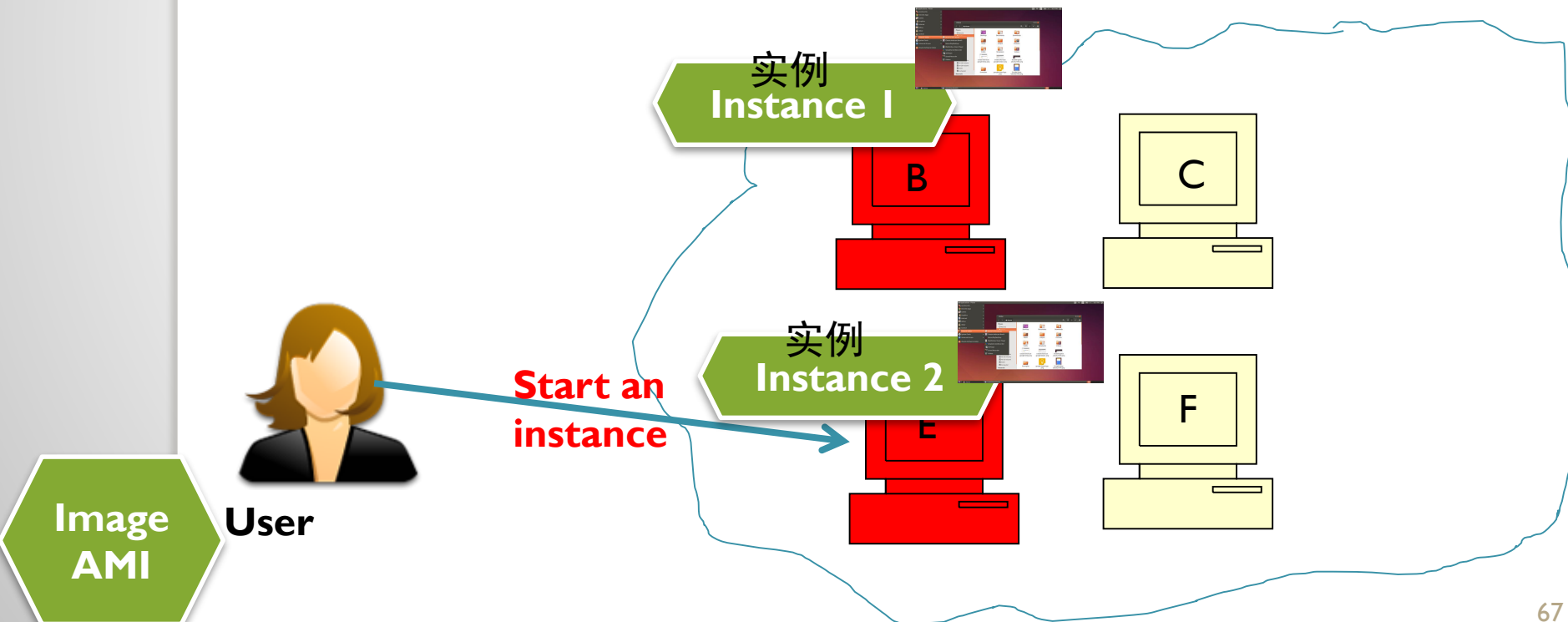
# Illustration

An **image** (虚拟机镜像) can be used to start an **instance** in the cloud (a virtual machine虚拟机)



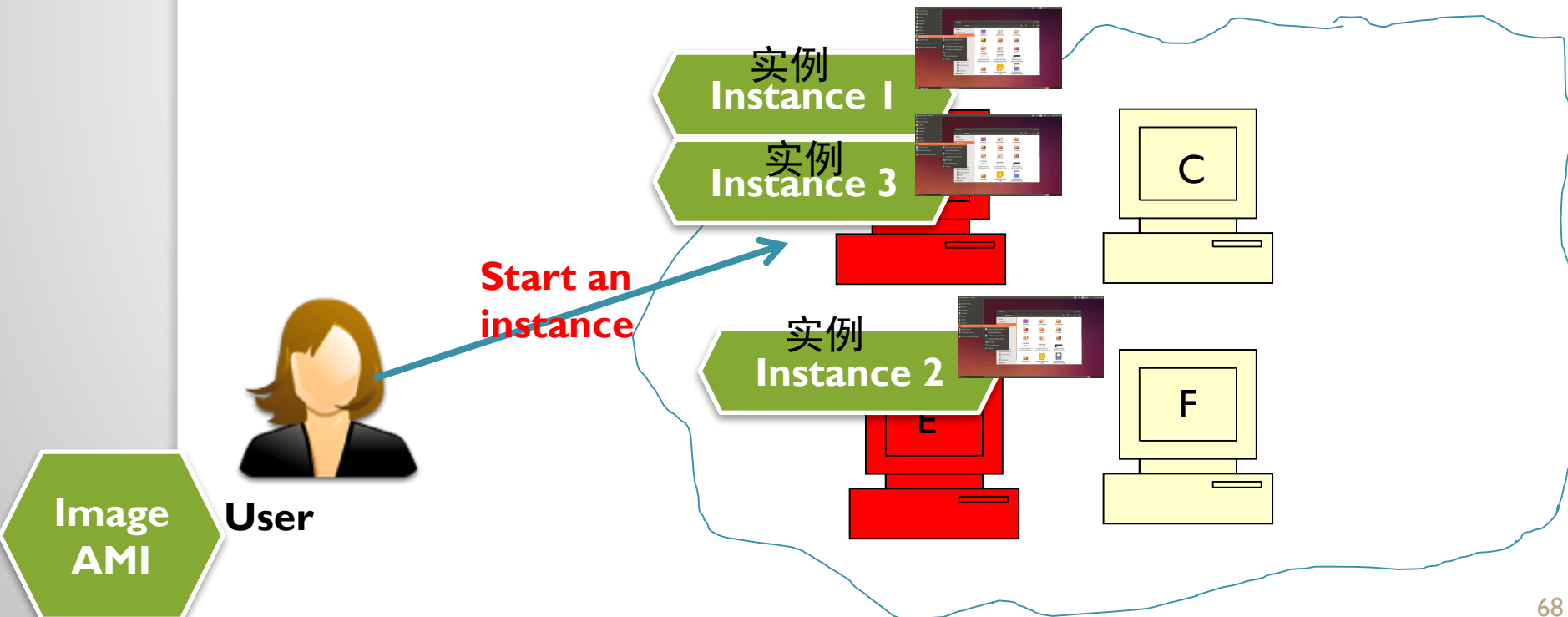
# Illustration

The same image can be used to start many instances



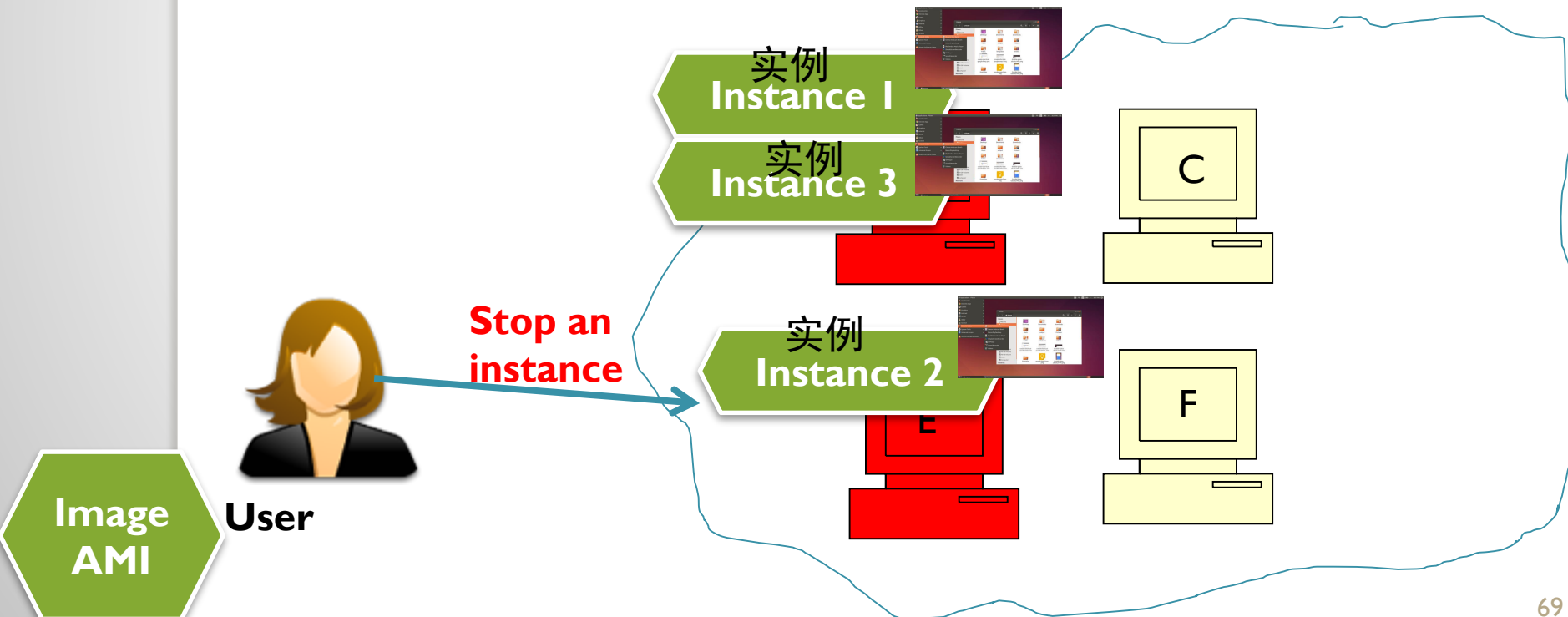
# Illustration

Multiple instances can be run on the same computer



# Illustration

The user can stop an instance.  
The user can restart an instance.

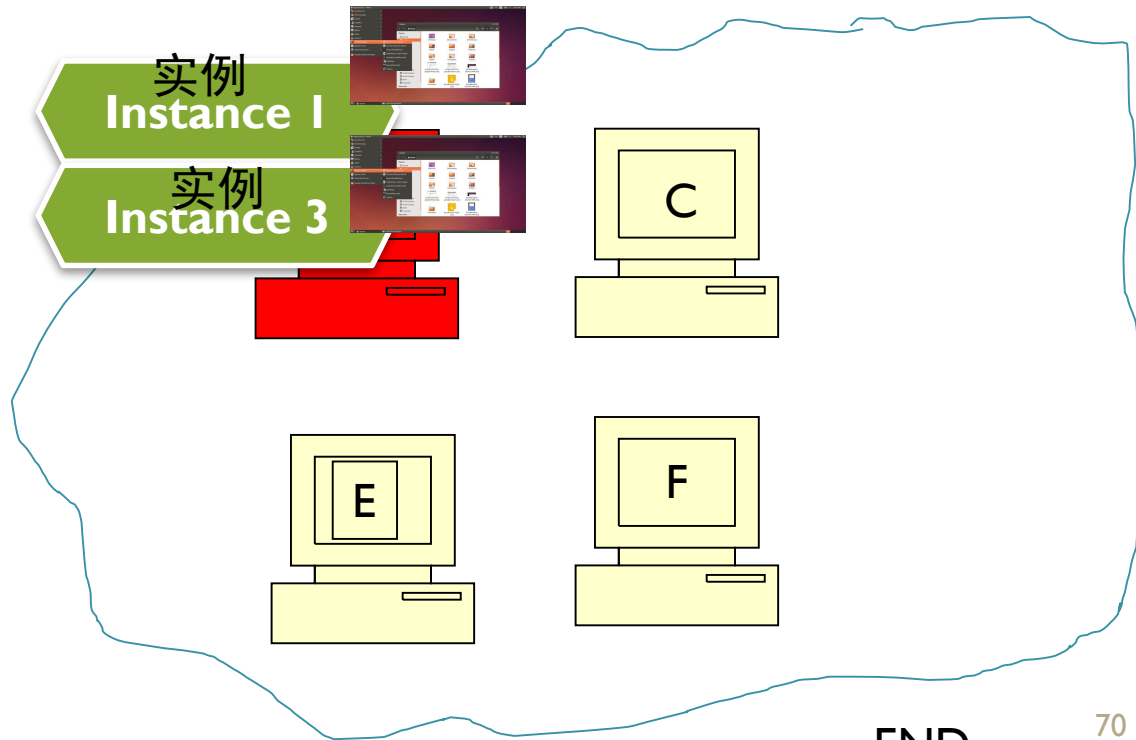


# Illustration

Image  
AMI

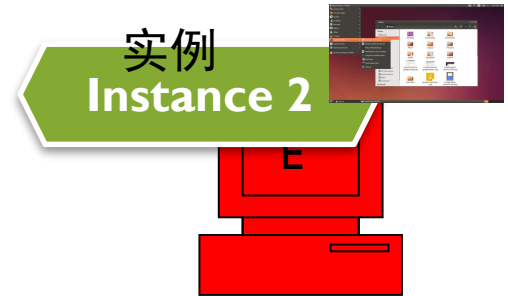


User



END

# Amazon EC2

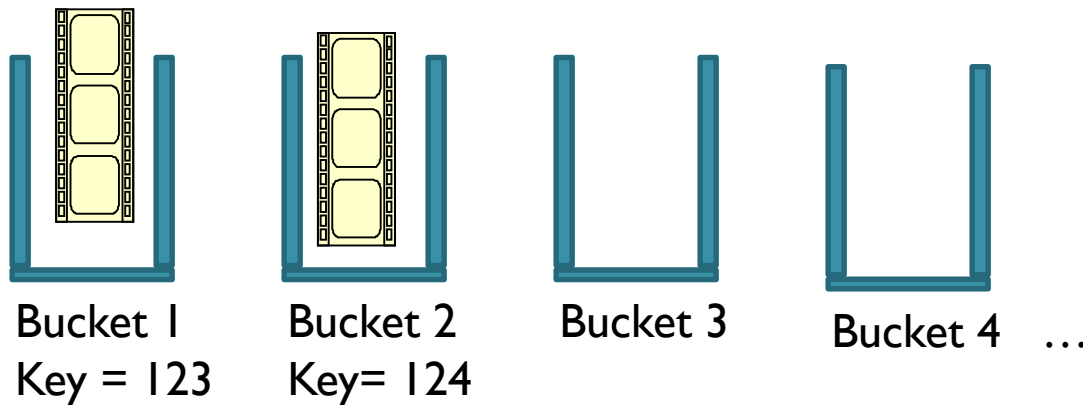


- An **instance (virtual machine)**
  - can use a limited amount of resources,
  - has a cost per hour.
  - is assigned to a location (e.g. [Beijing](#), [Singapore](#))
- **EC2** can be used for **elastic computing** (弹性计算)
- **EC2** can perform **load-balancing** (负载均衡), that is make sure that work is shared between instances

# Amazon S3

## Simple Storage System (s3)

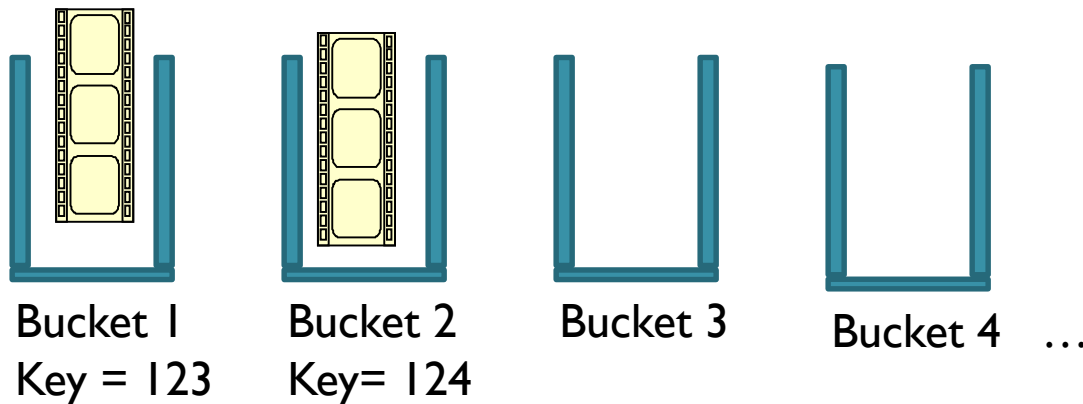
- A technology to store large objects in the cloud.
- **Three operations:** write, read and delete
- **Maximum number of objects:** unlimited
- **Size of an object:** up to 5 TB (5百万兆字节).
- Each object is stored in a bucket having a key.
- A object is retrieved from storage using its key.





# Amazon S3

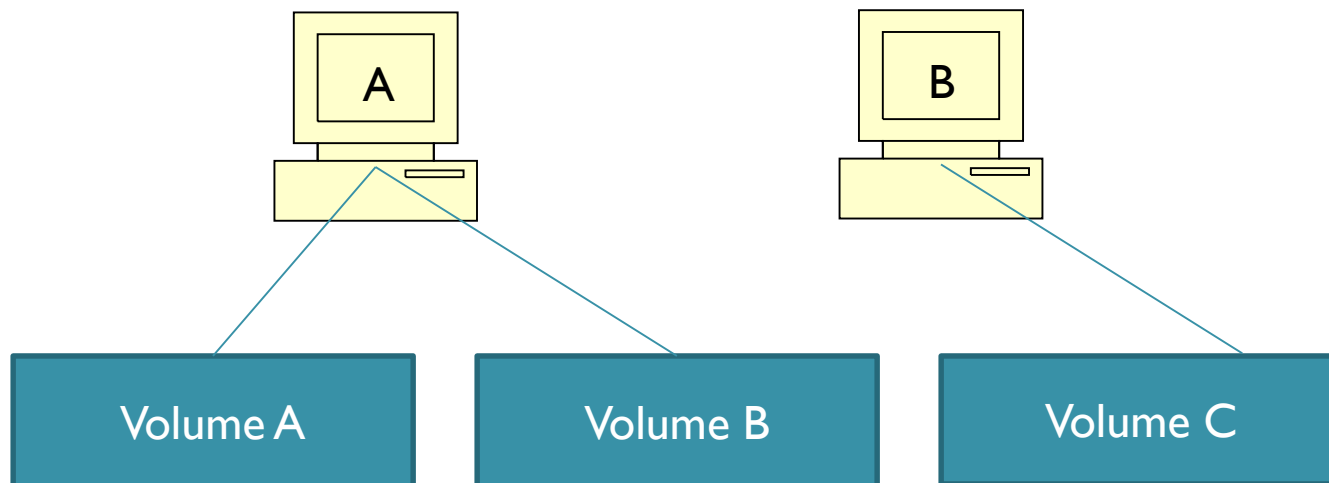
- **Data about objects:** creation date (创建日期), name, **access control list** (访问控制表), etc.
- **S3** offers some **authentication** mechanism (身份验证机制) to check the identify of users.
- **Technical details:** an **error detection code** (错误检测码 - MD5 hash) is used to check if files have been correctly transmitted.



# Amazon EBS

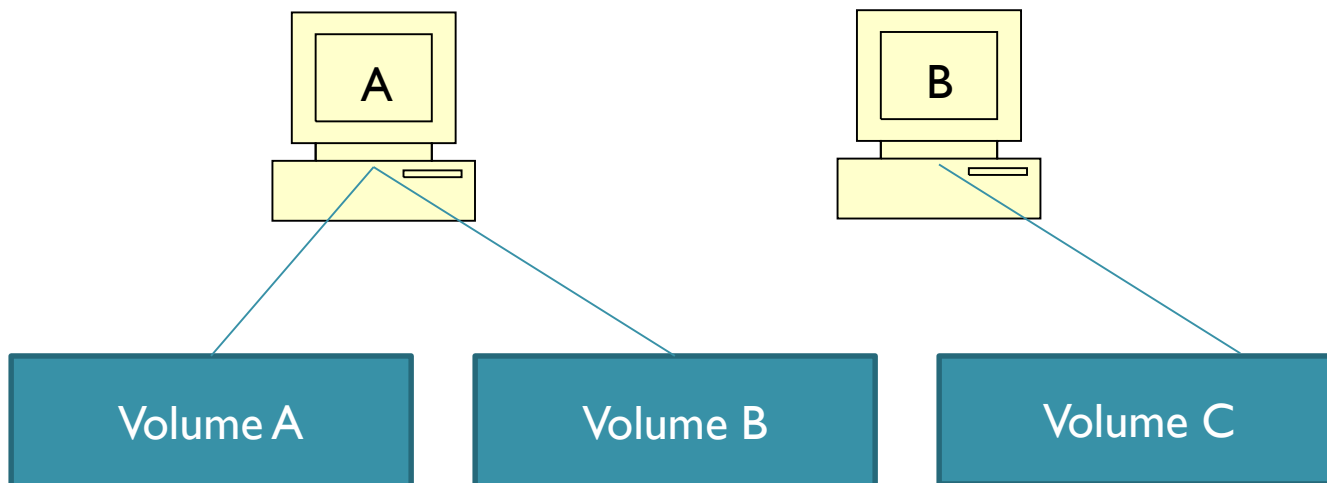
## Elastic Block Store (EBS)

- Another technology to store data in the Amazon Cloud (an alternative to S3).
- Data is stored as **volumes** (卷).
- **A volume** (卷) is a storage space of 1 GB to 1 TB.
- It is used like a virtual hard drive (虚拟硬盘) .



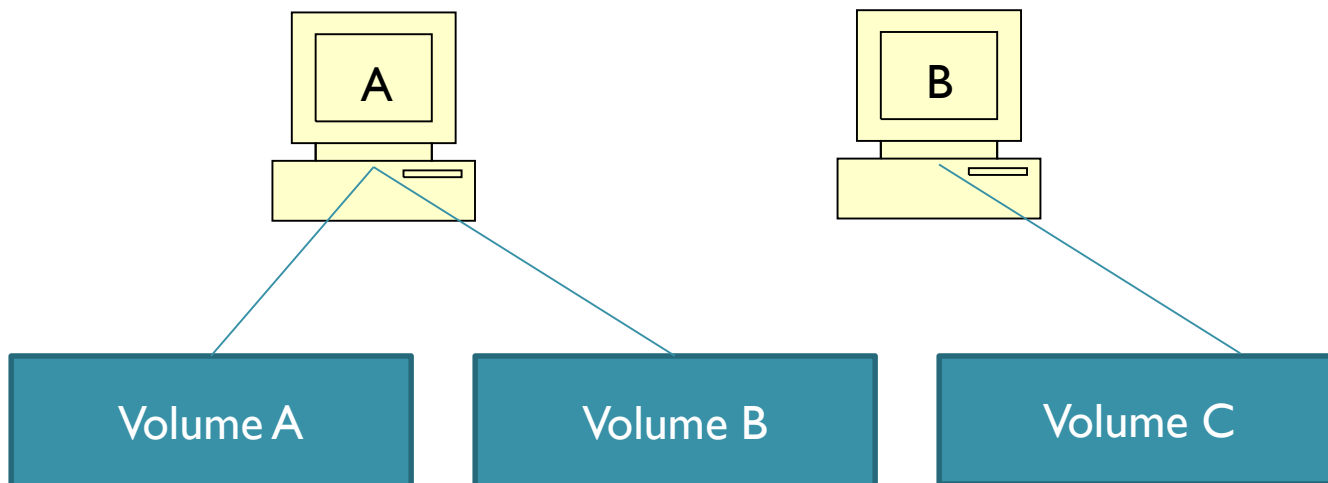
# Amazon EBS

- **Operations** typically performed on a **hard drive** (硬盘驱动器) can be performed on a **volume** (卷)  
e.g. **formatting** 格式化
- An **instance** can use many volumes.
- A **volume** can be used by **one instance** at a time.



# Amazon EBS

- A volume can be copied (**data replication** -数据复制).
- **Data replication** (数据复制) is automatically performed by the Amazon Cloud to avoid data loss.



# Amazon SimpleDB

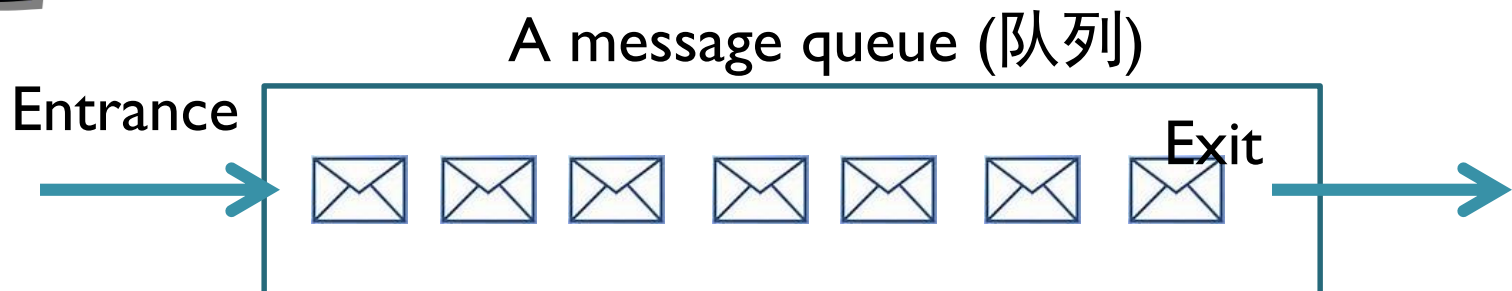
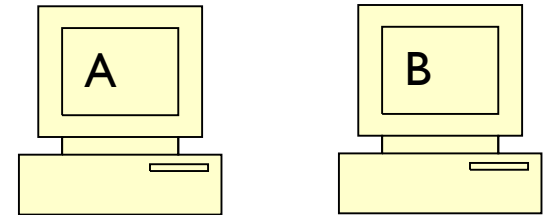
## Simple DB

- Another technology to store data in the Amazon EC2 Cloud.
- Similar to a database (数据库).
- The user stores data and perform queries (查询) to access data.
- Pay for storage space and data access.
- **Technical details:**
  - It is a NoSQL database (key/value pairs).
  - Access by webservice requests
  - Property of “eventual consistency”
  - Metadata can be added to each object

# Amazon SQS

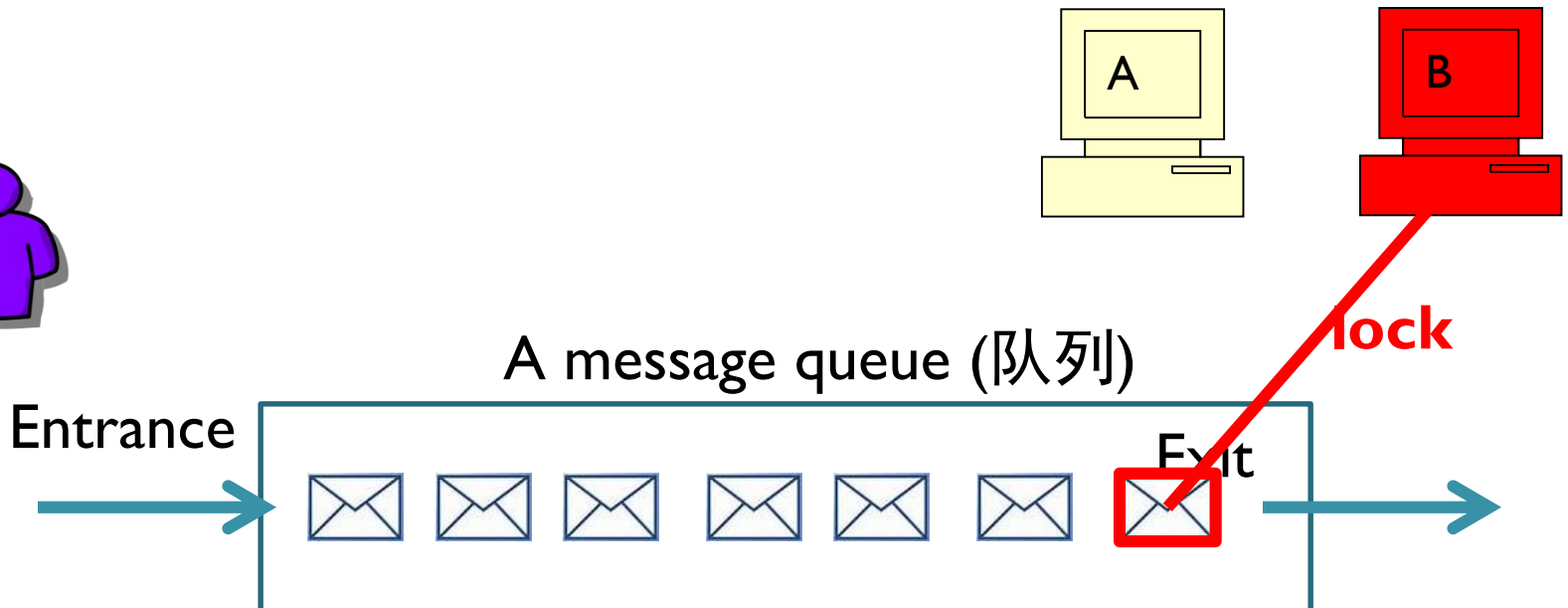
## Simple Queue Service

- A communication system for the Amazon Cloud.
- SQS is a system that allows **EC2 instances** to **coordinate** (协调) their activities by **sending** and **receiving messages**.
- **Add** or **read** messages.



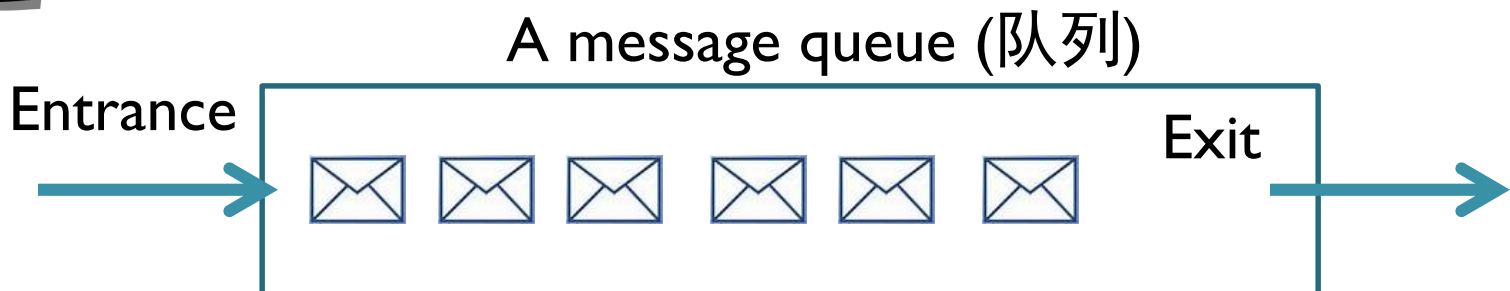
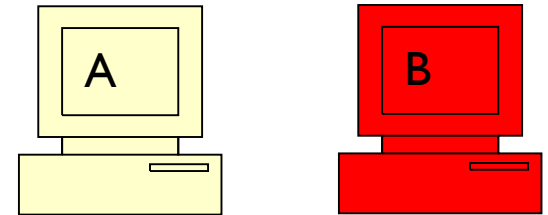
# Amazon SQS

- An instance will “lock” a message and then process it.
- If processing fails, the lock expires and the message is available to other instances.



# Amazon SQS

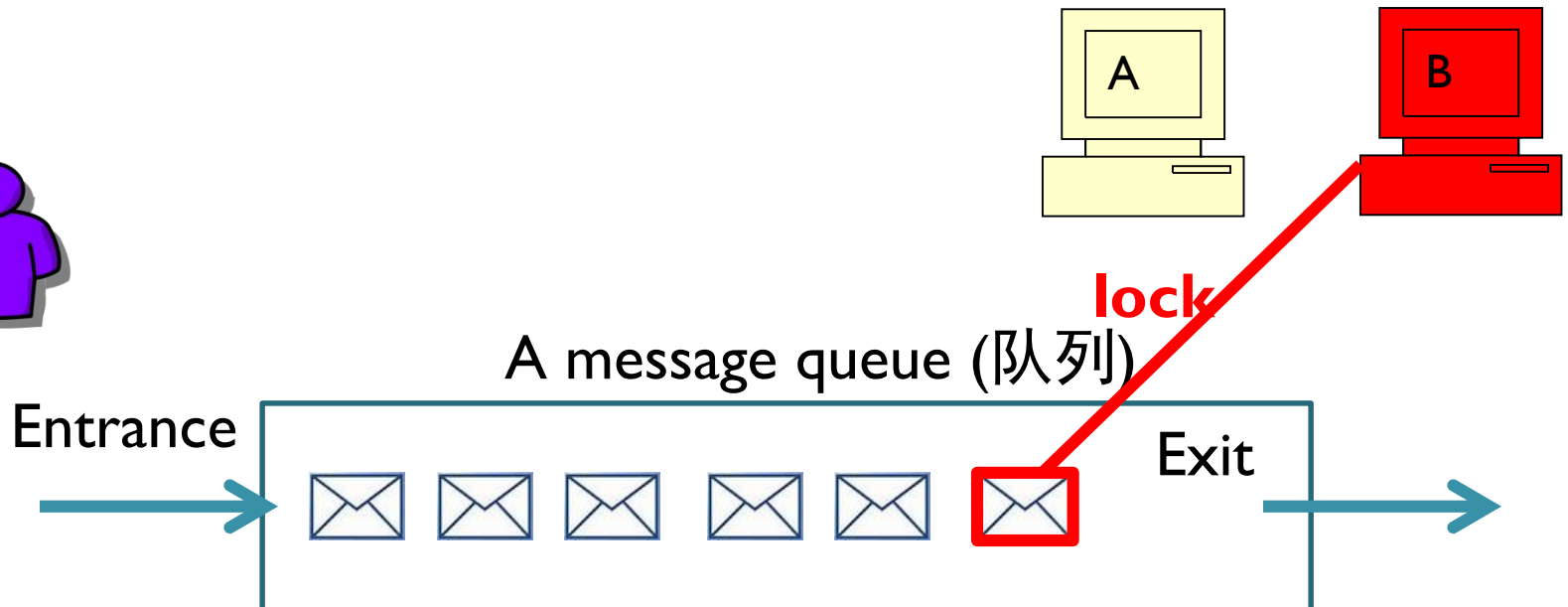
- An instance will “lock” a message and then process it.
- If processing fails, the lock expires and the message is available to other instances.





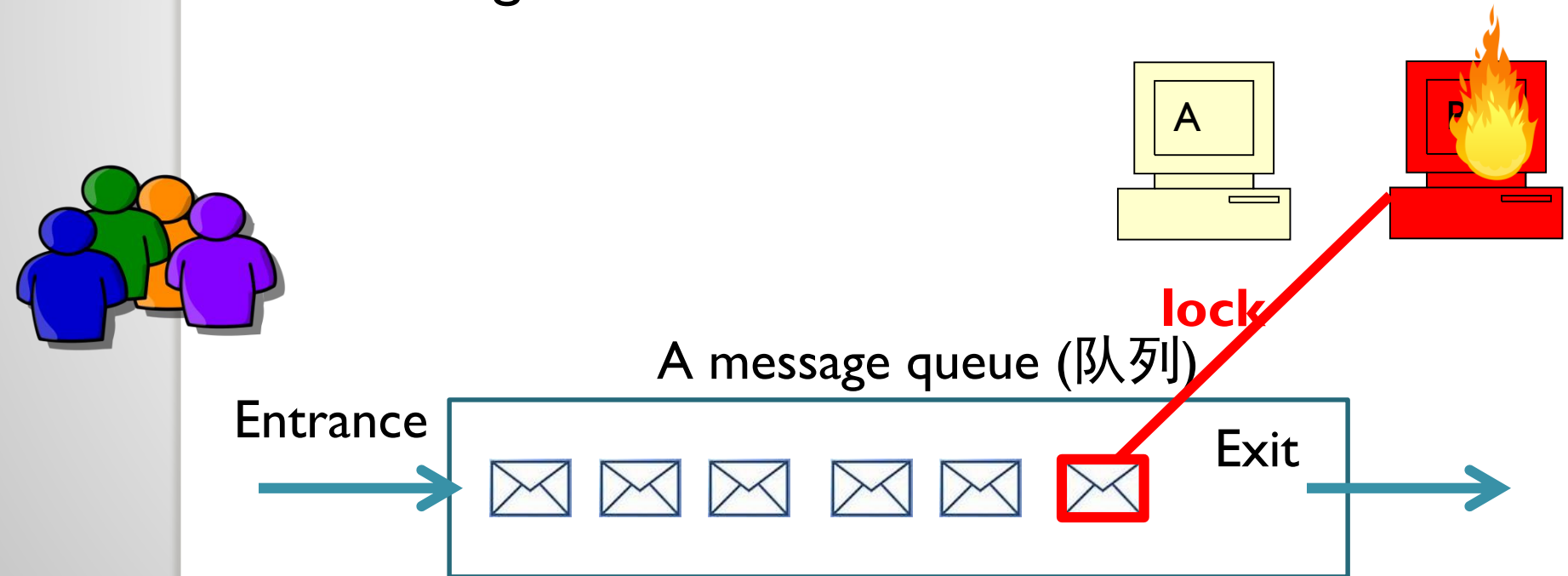
# Amazon SQS

- An instance will “lock” a message and then process it.
- If processing fails, the lock expires and the message is available to other instances.



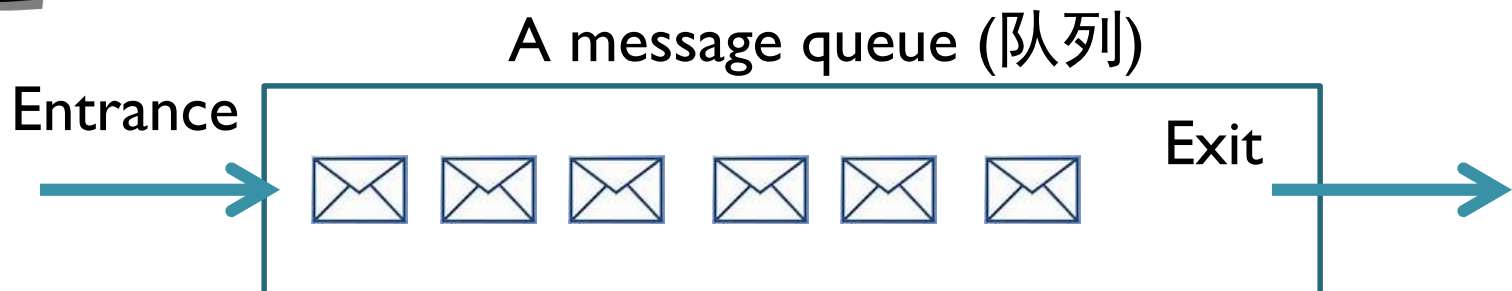
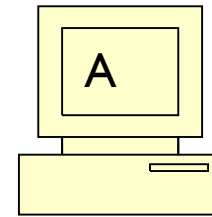
# Amazon SQS

- An instance will “lock” a message and then process it.
- If processing fails, the lock expires and the message is available to other instances.



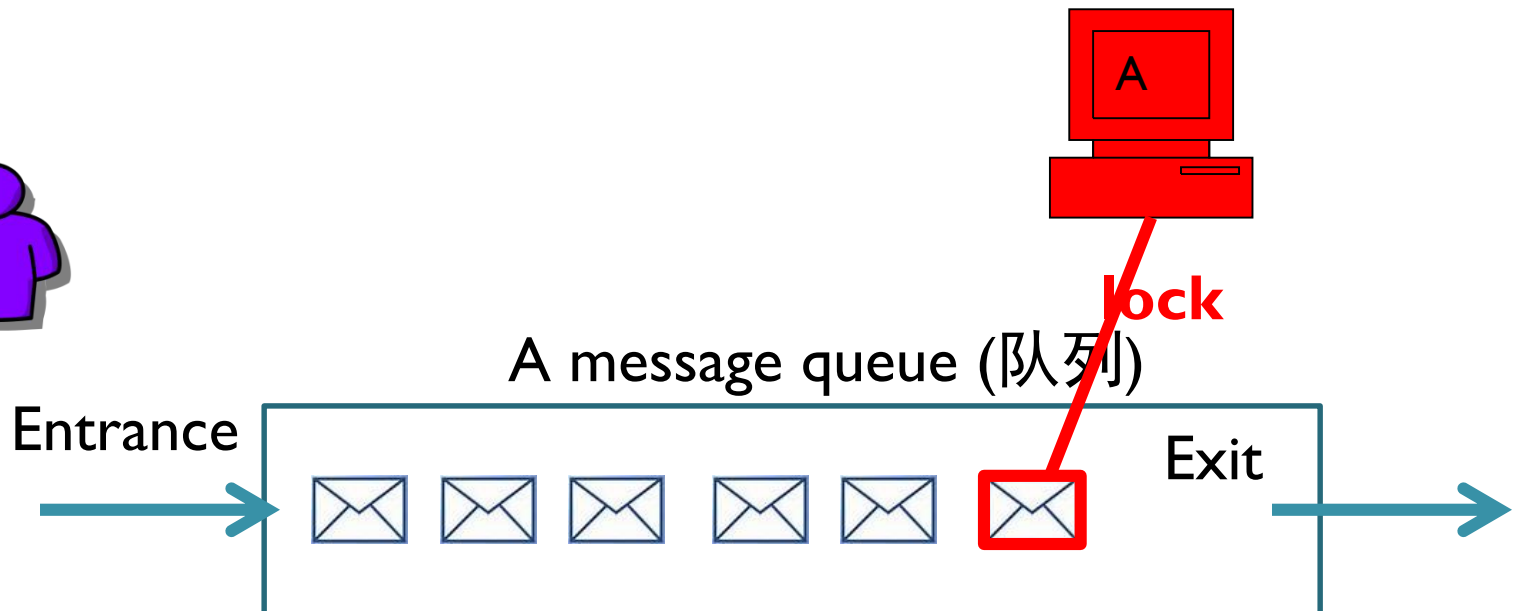
# Amazon SQS

- An instance will “lock” a message and then process it.
- If processing fails, the lock expires and the message is available to other instances.



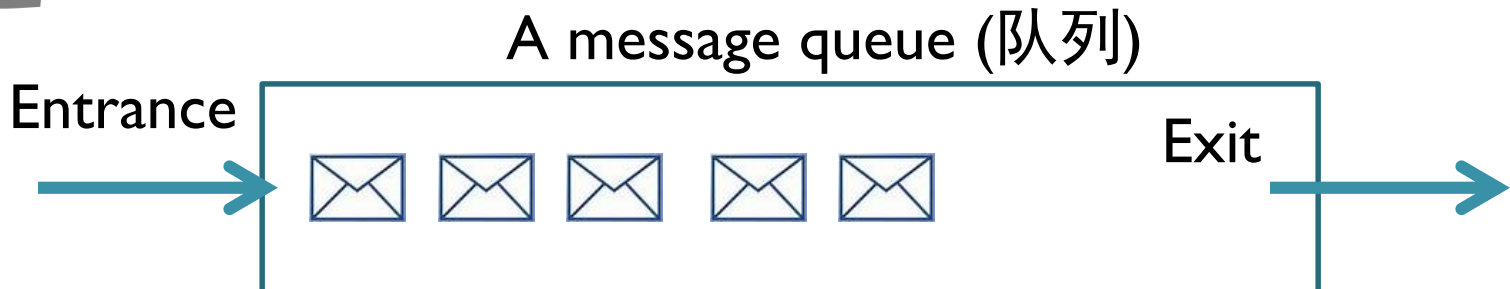
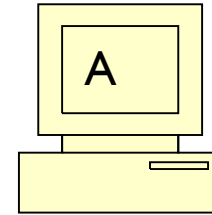
# Amazon SQS

- An instance will “lock” a message and then process it.
- If processing fails, the lock expires and the message is available to other instances.



# Amazon SQS

- **Why using a queue (队列)?**
  - Useful to process a large amount of requests (e.g. coming from persons using a website).
  - Ensures that all requests will be processed following some order (first come first served)
  - Multiple computers can be used to process requests to increase performance.

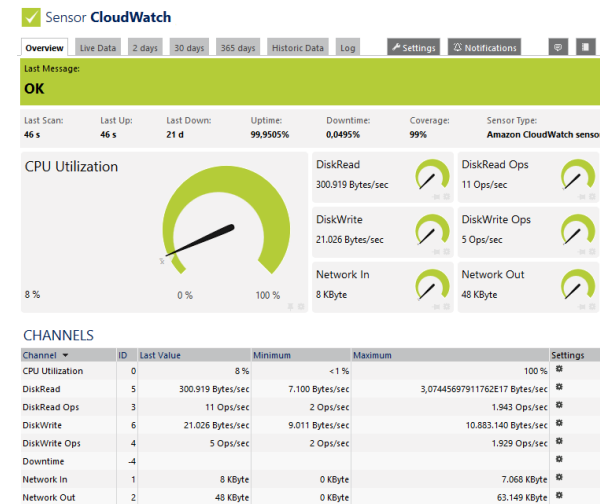


# Amazon Cloud Watch



## Cloud Watch

- A service for monitoring the infrastructure used in the cloud.
- Cloud Watch allows collecting and tracking metrics about the performance of applications and the efficiency of resource utilization.
- A user can monitor approximately a dozen metrics and view graphs and statistics for these metrics.
- Monitoring every 5 minutes : free of charge
- Monitoring every 1 minute: need to pay (\$)



# Amazon AutoScaling

- This service provides **cloud elasticity** (云的弹性).
- Instances in the cloud are put into **groups**.
- The user can defines conditions for increasing or decreasing the size of groups automatically.
  - A minimum number of instances,
  - A maximum number of instance,
  - A regular size for each group.
  - **Example 1.** a policy is that if the CPU utilization of instances in a group is higher than 90 %, then a new instance is created.

# Amazon AutoScaling

- **Example 2.** if the CPU utilization of an instance is less than 20 %, then the instance is shut down



# Other Amazon services

The Amazon Cloud provides many other services such as:

- **Amazon Elastic MapReduce (EMR)**: provides support for Hadoop in an EC2 cloud.
- **Amazon Route 53**: a low-latency DNS service.
- **CloudFront**: for content delivery (内容传递)
- **Amazon Elastic Load Balancer**

# Licensing Agreement (授权协议)

To use the **cloud of Amazon**, one must accept a **Licensing Agreement**.

- **Amazon can terminate the service** to any customer at any time **for any reasons**, and one should not sue Amazon for any damages.
- One should not use the cloud of Amazon for direct marketing, spamming...
- **No illegal material** should be stored in the cloud.

# Amazon Data Center

## They are located in **multiple locations**:

- Billing rates differ from one region to another based on energy, communication and maintenance costs.
- Each region is divided into availability zones.
- **For example:**

Region	Location	Availability Zones	Cost
US West	Oregon	us-west-2a/2b/2c	Low
US West	North California	us-west-1a/1b/1c	High
US East	North Virginia	us-east-1a/2a/3a/4a	Low
Europe	Ireland	eu-west-1a/1b/1c	Medium
South America	Sao Paulo, Brazil	sa-east-1a/1b	Very high
Asia/Pacific	Tokyo, Japan	ap-northeast-1a/1b	High
Asia/Pacific	Singapore	ap-southeast-1a/1b	Medium

# Amazon Data Center

- An **availability zone** is a **data center** containing multiple servers.
- A **server** (服务器) may run multiple virtual machines or instances (for different users).
- **Storage** is automatically replicated in a **region**.
- It is recommended to also replicate the content in multiple regions to avoid problems if some catastrophic events occur in a region.

Region	Location	Availability Zones	Cost
US West	Oregon	us-west-2a/2b/2c	Low
US West	North California	us-west-1a/1b/1c	High
US East	North Virginia	us-east-1a/2a/3a/4a	Low
Europe	Ireland	eu-west-1a/1b/1c	Medium
South America	Sao Paulo, Brazil	sa-east-1a/1b	Very high
Asia/Pacific	Tokyo, Japan	ap-northeast-1a/1b	High
Asia/Pacific	Singapore	ap-southeast-1a/1b	Medium

# Amazon Data Center locations

How to choose a region?

- Based on cost, communication latency (通信延迟), reliability (可靠性), security, etc.

To create an instance, one must select:

- the **region** and **availability zone**,
- the **type** of instances from a small set of options (CPU type, main memory, ...)

# IP Addresses (IP地址) of instances

- Each **instance** needs an **IP address (IP地址)** to be able to communicate with other computers.
- IP addresses can be viewed as some kind of “postal addresses” (邮政地址) for computers on a network, used for sending messages.



# IP Addresses of instances

- An **AMI image** does not provide an IP address.
- **When a user create an instance**, Amazon automatically assign public/private IP addresses to each new instance.



# IP Addresses of instances

- The **private IP address** is used for **communicating with computers inside the cloud.**
- The **public IP address** is used for **communicating with computers on the Internet.**
- When an instance is shut down, its addresses may be given to another instance.





# Three pricing models

- **On-demand instance:**
  - the user pay per hour that an instance is running.
  - The most popular model
- **Reserved instance:**
  - The user pay at one time fee for a given number of hours. This provides a discount on the regular rate.
- **Spot instance:**
  - the user bides on unused capacity in the cloud,
  - Instances are launched when the market price reaches a threshold specified by the user.

# Types of instances

- **The EC2 cloud offers several types of instances:**
- **For example:**

- *Standard instances.* Micro (StdM), small (StdS), large (StdL), extra large (StdXL); small is the default.
- *High memory instances.* High-memory extra-large (HmXL), high-memory double extra-large (Hm2XL), and high-memory quadruple extra-large (Hm4XL).
- *High CPU instances.* High-CPU extra-large (HcpuXL).
- *Cluster computing.* Cluster computing quadruple extra-large (Cl4XL).

**Table 3.2** The nine instances supported by *EC2*. The cluster computing *c14XL* (quadruple extra-large) instance uses two Intel Xeon X5570, Quad-Core Nehalem Architecture processors. The instance memory (I-memory) refers to persistent storage; the I/O performance can be moderate (M) or high (H).

Instance Name	API Name	Platform (32/64-bit)	Memory (GB)	Max <i>EC2</i> Compute Units	I-Memory (GB)	I/O (M/H)
StdM		32 and 64	0.633	1 VC; 2 CUs		
StdS	m1.small	32	1.7	1 VC; 1 CU	160	M
StdL	m1.large	64	7.5	2 VCs; 2 × 2 CUs	85	H
StdXL	m1.xlarge	64	15	4 VCs; 4 × 2 CUs	1,690	H
HmXL	m2.xlarge	64	17.1	2 VCs; 2 × 3.25 CUs	420	M
Hm2XL	m2.2xlarge	64	34.2	4 VCs; 4 × 3.25 CUs	850	H
Hm4XL	m2.4xlarge	64	68.4	8 VCs; 8 × 3.25 CUs	1,690	H
HcpuXL	c1.xlarge	64	7	8 VCs; 8 × 2.5 CUs	1,690	H
Cl4XL	cc1.4xlarge	64	18	33.5 CUs	1,690	H

This data is shown as an example (it is from 2012)

VC = Virtual computers      CU = Computing Unit

**Table 3.3** The charges in dollars for one hour of Amazon's cloud services running under *Linux* or *Unix* and under *Microsoft Windows* for several *EC2* instances.

Instance	<i>Linux/Unix</i>	<i>Windows</i>
StdM	0.007	0.013
StdS	0.03	0.048
StdL	0.124	0.208
StdXL	0.249	0.381
HmXL	0.175	0.231
Hm2XL	0.4	0.575
Hm4XL	0.799	1.1
HcpuXL	0.246	0.516
Cl4XL	0.544	N/A

**Table 3.4** Monthly charges in dollars for data transfer out of the US West (Oregon) region.

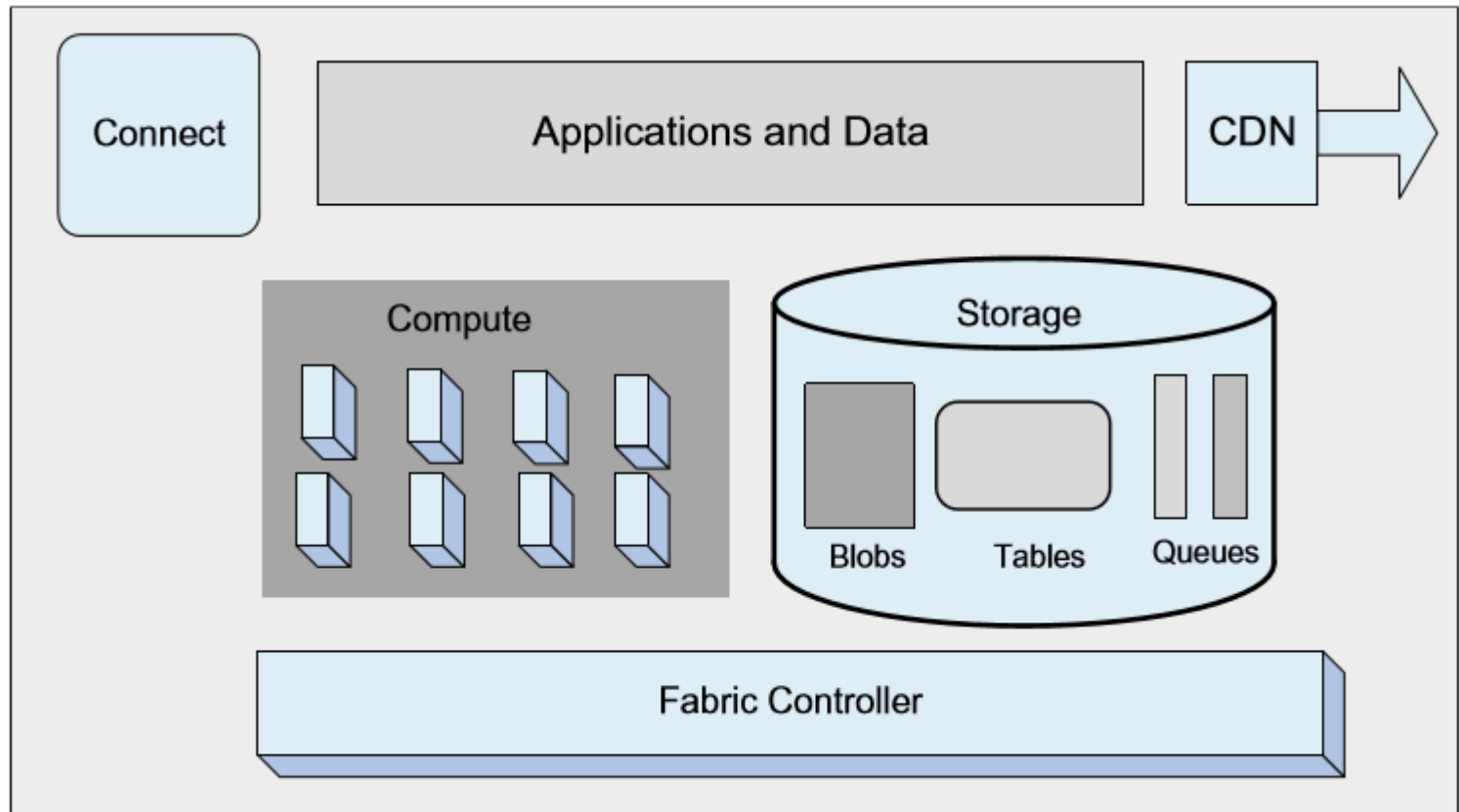
Amount of Data	Charge \$
First 1 GB	0.00
Up to 10 TB	0.12
Next 40 TB	0.09
Next 100 TB	0.07
Next 350 TB	0.05

This data is shown as an example (it is from 2012)

# Microsoft

- **Microsoft Online services:** software-as-a-service
- **Microsoft Azure:** platform-as-a-service
  - **Windows Azure:** an operating system
  - **SQLAzure:** a database for storing data based on SQLServer
  - **AzureAppFabric:** a collection of services for cloud applications

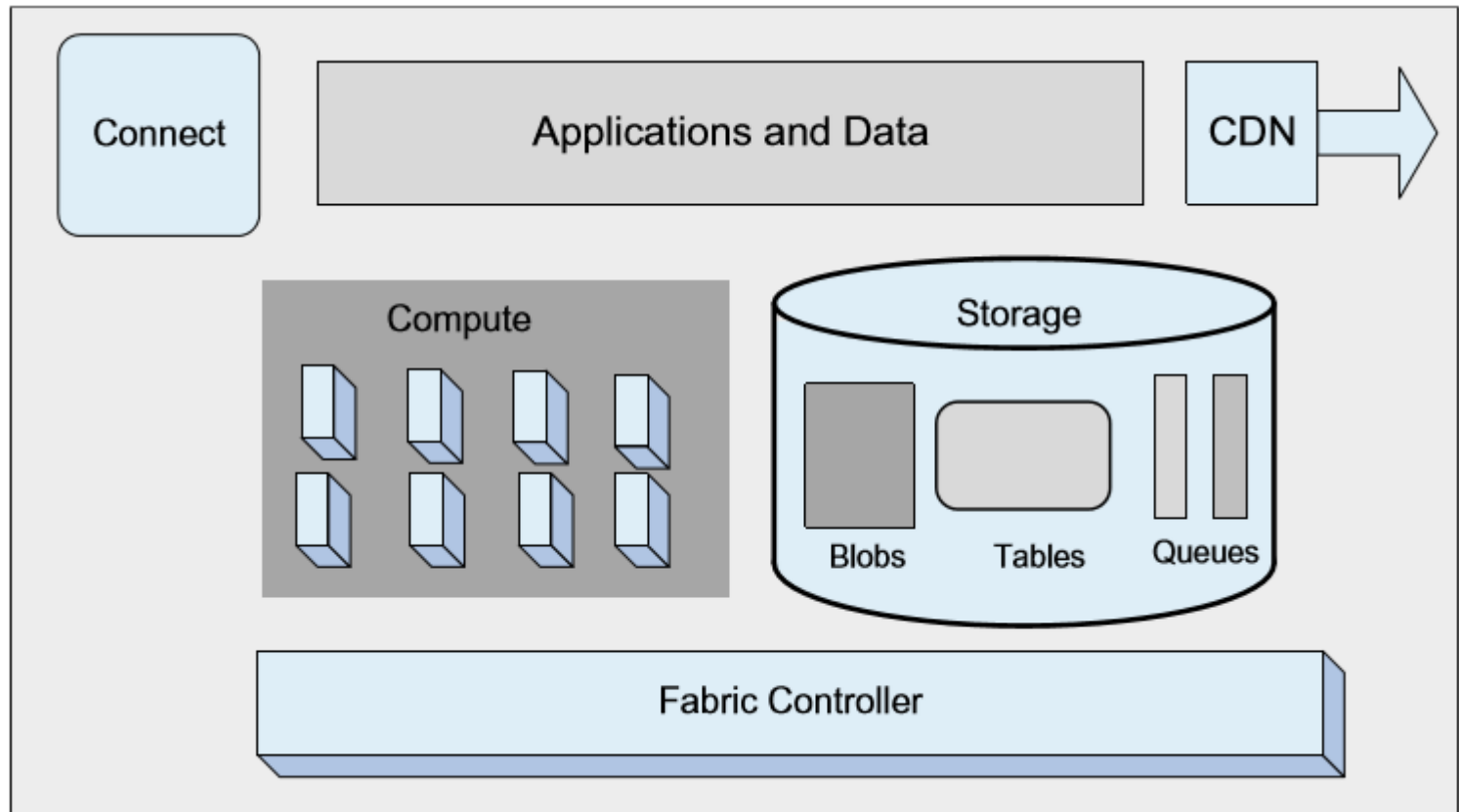
# Microsoft Azure



Three main components:

- **Computer:** provides a computation environment
- **Storage:** scalable storage
- **Fabric Controller:** deploys, manages, and monitors applications; it interconnects nodes consisting of servers, high-speed connections, and switches

# Microsoft Azure



- **Storage** uses blobs, tables, and queues to store data,
- **Fabric controller**: provides scaling, load balancing, memory management, and reliability
- **CDN**: maintains cache copies of data, for faster access.

# Open-source platforms for private clouds

There exists some **free** and open-source software to setup a **private cloud**.

**Table 3.5** A side-by-side comparison of *Eucalyptus*, *OpenNebula*, and *Nimbus*.

	<i>Eucalyptus</i>	<i>OpenNebula</i>	<i>Nimbus</i>
Design	Emulate EC2	Customizable	Based on Globus
Cloud type	Private	Private	Public/Private
User population	Large	Small	Large
Applications	All	All	Scientific
Customizability	Administrators and limited users	Administrators and users	All but image storage and credentials
Internal security	Strict	Loose	Strict
User access	User credentials	User credentials	x509 credentials
Network access	To cluster controller	—	To each compute node



# Eucalyptus

- <http://www.eucalyptus.com>
- Similar to Amazon EC2
- Offers some compatibility with Amazon Web Services.

# Conclusion

- Today, we have discussed:
  - mutual exclusion
  - cloud infrastructure





# References

Chapter 2 and 3. D. C. Marinescu. Cloud Computing Theory and Practice, Morgan Kaufmann, 2013.