

Characteristics of Distributed Systems

莊 裕 澤

Yuh-Jzer Joung

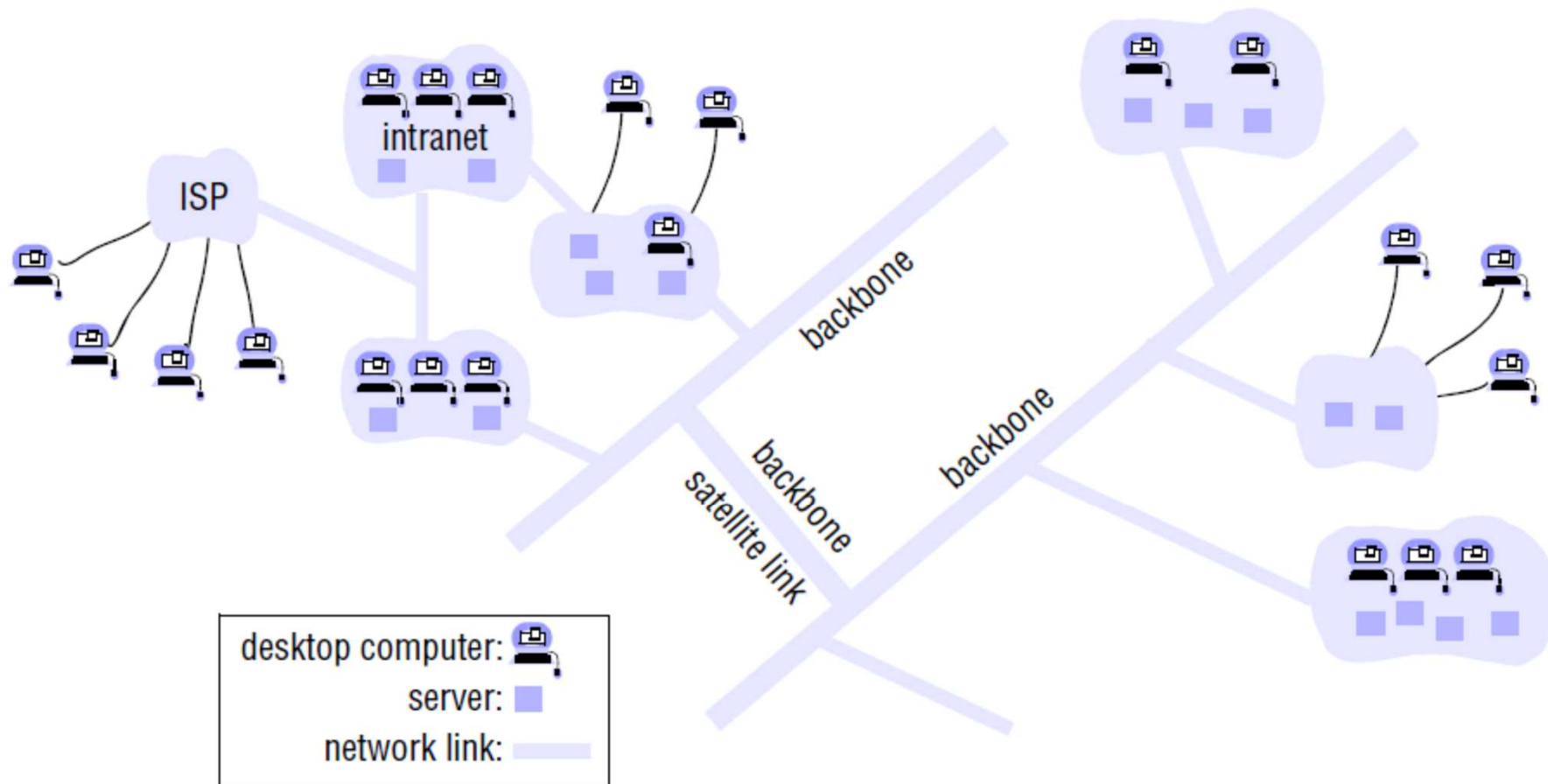
**Dept. of Information Management
National Taiwan University**

Distributed Systems

- ❑ **Distributed Systems:** A system whose components are spread across multiple computing devices on a network.
 - Range from Internet-scale applications to embedded systems in micro devices.
 - Examples: Intranets, Mobile and ubiquitous computing, Sensor networks, World Wide Web, Banking Systems, e-commerce, stock trading, edge computing ...

Example: Internet

Figure 1.3 A typical portion of the Internet

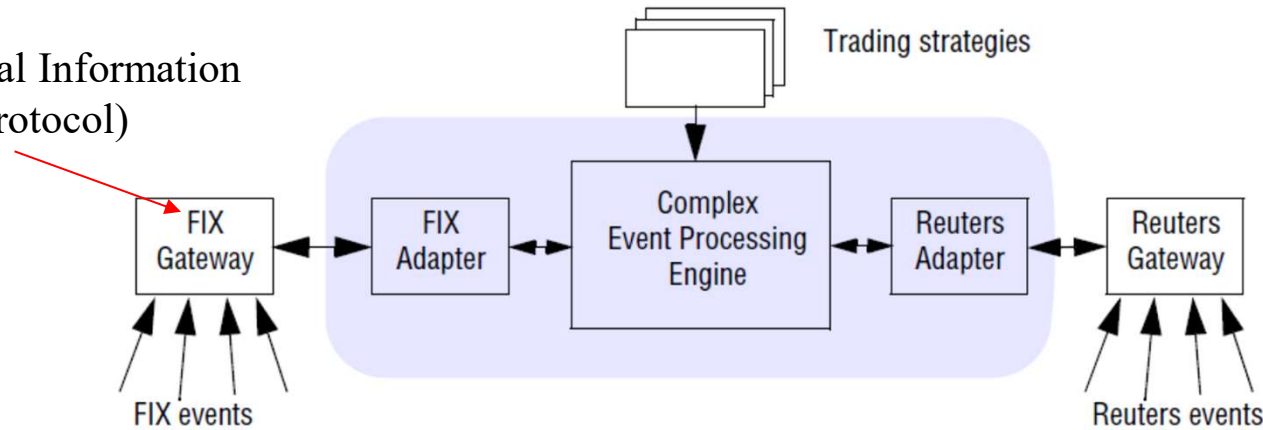


Source: [Distributed Systems: Concepts and Design 5th ed.](#), C. Coulouris et al., 5th ed., 2011.

Example: Financial Trading

Figure 1.2 An example financial trading system

FIX: Financial Information
eXchange (protocol)

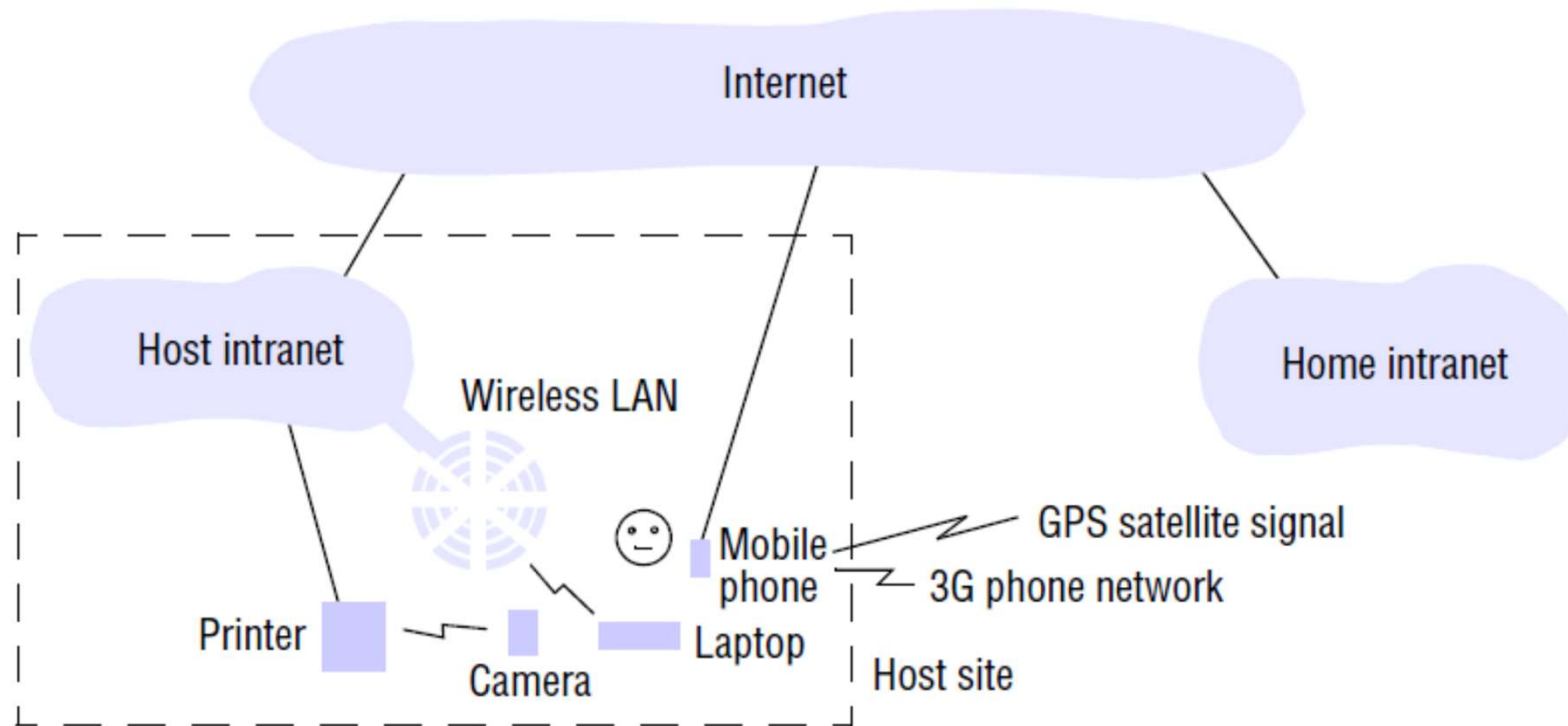


```
WHEN
    MSFT price moves outside 2% of MSFT Moving Average
FOLLOWED-BY (
    MyBasket moves up by 0.5%
    AND
    HPQ' s price moves up by 5%
    OR
    MSFT' s price moves down by 2%
)
)
ALL WITHIN
    any 2 minute time period
THEN
    BUY MSFT
    SELL HPQ
```

Source: Distributed Systems: Concepts and Design 5th ed., C. Coulouris et al., 5th ed., 2011.

Example: Mobile and Ubiquitous Computing

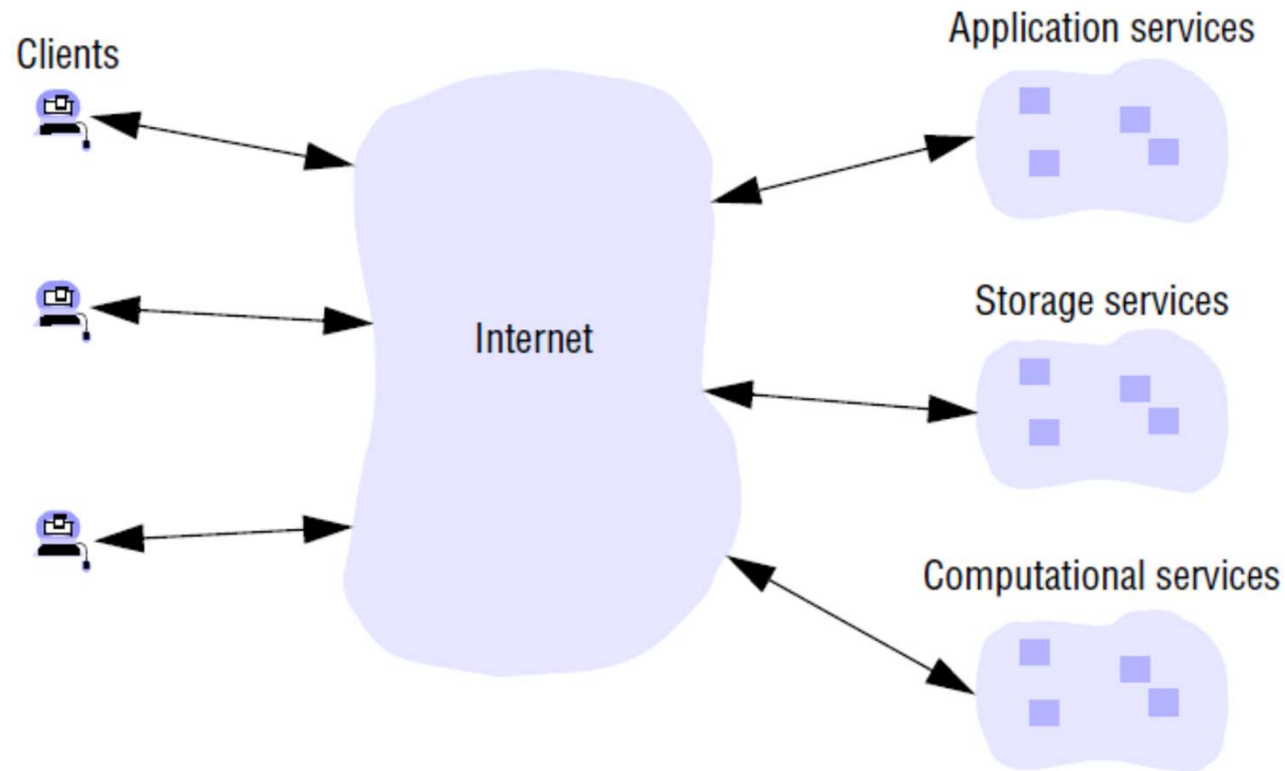
Portable and handheld devices in a distributed system



Source: [Distributed Systems: Concepts and Design 5th ed.](#), C. Coulouris et al., 5th ed., 2011.

Example: Cloud Computing

Cloud computing



Source: [Distributed Systems: Concepts and Design 5th ed.](#), C. Coulouris et al., 5th ed., 2011.

Why Distributed Systems?

- ❑ **Resource sharing.** Distributed database, peer-to-peer networks, aggregate computing power (e.g., SETI@HOME)
- ❑ **Sometimes, it is natural.** Think of geographically distributed applications: ATMs, servers, battlefields
- ❑ **Speed-up.** Example: grid computing
- ❑ **Fault-tolerance.** Trade-off between total collapse and graceful degradation.

Syllabus

□ 課程目標：

- 提供分散式系統與雲端應用服務開發所需要的基礎理論知識與實務技能

□ 課程要求：

- 具基本的網路技術知識與 C++和 web service程式設計能力

Topics to be covered

❑ Basics of Distributed Systems

- System Models, Name Services, Synchronization, Coordination, Time and Security, Transactions Processing and Concurrency Control
- Distributed Computing & Distributed Algorithms
- Fault-tolerance, Consensus, and Byzantine Generals

❑ P2P & Distributed Hash Tables

❑ Large Distributed File & Storage Systems

- Hadoop Distributed File System (HDFS), The Google File System, Ceph, Amazon Dynamo

❑ Docker Containers, Kubernetes

❑ Blockchains

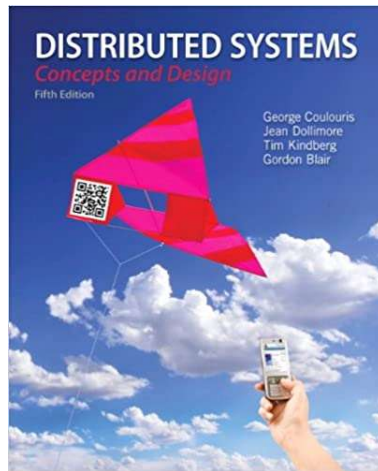
新加區塊練和期中project

Syllabus

參考書目

- ❑ Distributed Systems: Concepts and Design 5th Ed., C. Coulouris et al., 2011.
- ❑ 隨個單元指定相關論文 (~20篇)、網路教材與資源
- ❑ Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems,
Martin Kleppmann, 2017, O'Reilly.

汰舊 別買



評量方式

❑ Lab practices: 5%

作業1.2小時可完成，共3,4次，考試期末考很難。
Project: 期中需要訂題目。

❑ Midterm project: 25%

❑ Term project: 20%

■ 3~4人一組, 期末搭配組內參與度互評防止freerider

❑ 期中考：22%

❑ 期末考：28%

Ps. 各部分成績可能先正規化調整後再綜合計算

Does this course suit you?

課程負荷：🙄🙄🙄🙄🙄

加退選

學術倫理

作業做不出來，寧願不交，切勿抄襲。

期末抽查比對所有作業，一次抄襲，所有作業以0分計，情節重大者送校方處理。

提供版本給他人者，所有作業以50%計。

教學理念

**Tell me and I forget.
Teach me and I remember.
Involve me and I learn**



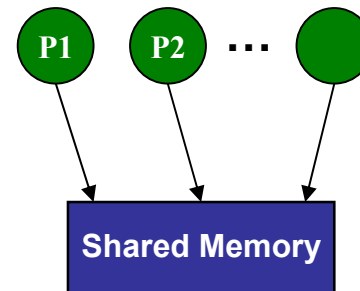
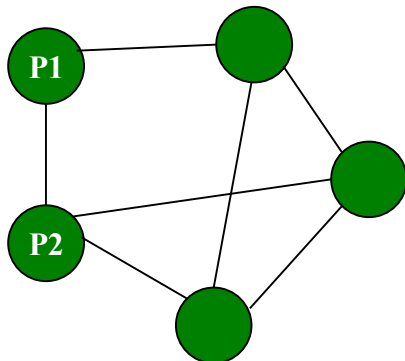
-- Benjamin Franklin

留意我上課提醒要回去思考的問題!!!

Source: [wiki](#)

Distributed Computing

- ❑ **Distributed Computing**: a field of computer science that studies distributed systems, including algorithms, properties, and theories.
- ❑ Two paradigms for modeling distributed systems:
 - **Message Passing** 適合實體式分散，沒特別說明，都是 message passing
 - To model systems whose components are physically apart and can communicate only by sending messages over the network
 - assumed by most distributed systems
 - **Shared Memory** 把資料寫入記憶體，讓另一個行程讀
 - To model systems where processes can access a shared memory



Challenges of the Design

- ❑ Concurrency
- ❑ Lack of global clock
- ❑ Lack of global knowledge
- ❑ Heterogeneity
- ❑ Openness
- ❑ Security
- ❑ Scalability
- ❑ Failure handling
- ❑ Transparency
- ❑ Quality of service

Concurrency

並行，和 synchronisation(同步、需等待) 相反

❑ Interference between components

1. READ x from data
1. READ x from data
2. ADD 1 to x
3. WRITE x to data

2. ADD 1 to x

3. WRITE x to data

1. READ x from data
1. READ x from data
2. ADD 1 to x
3. WRITE x to data

2. ADD 1 to x

3. WRITE x to data

在 concurrency 的形況下，
兩個元件會互相干擾

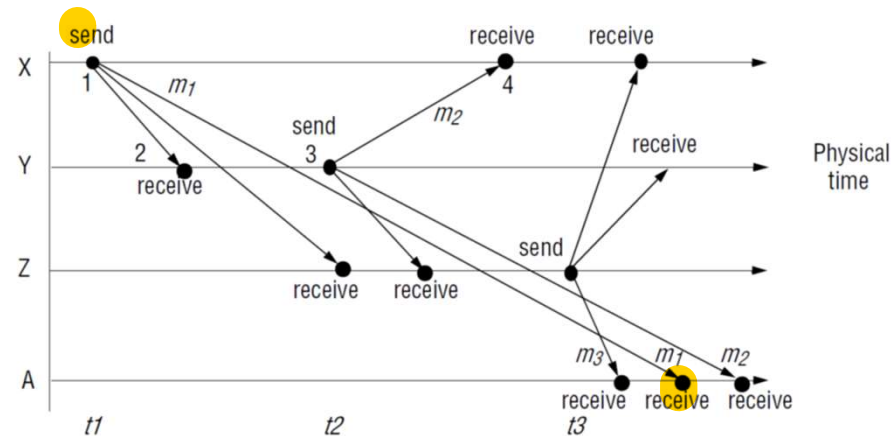
怎麼處理？

A: Lock (如DB鎖row)

Lack of Global Clock

- ❑ Time is important in computer systems
 - Files are stamped with the time it was created, modified, and accessed.
 - Every email, transaction, ... are also timestamped.
 - Setting timeouts and measuring latencies
- ❑ However, in a distributed systems where programs communicate only by sending messages through a network, there is no notion of “global time” among programs.
- ❑ However, there is “logical time” concept to reflect causal ordering of events.

在分散式系統，沒有一個全域的時間，但是有邏輯時間，可記錄因果關係；如邏輯上X要先send，A才能receive。

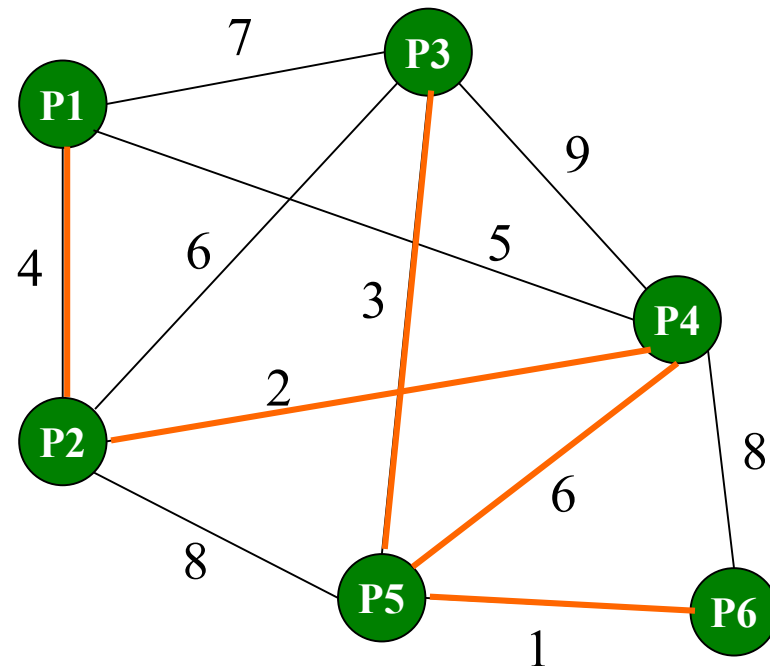


Lack of Global Knowledge

- ❑ Example: **Minimum Spanning Tree (MST)**
- ❑ How do you construct an MST in the network?
 - Sequential algorithm?
 - Kruskal's (1956), Prim's (1957)
 - Recall Dijkstra's shortest path algorithm (1959)
 - Distributed algorithm?

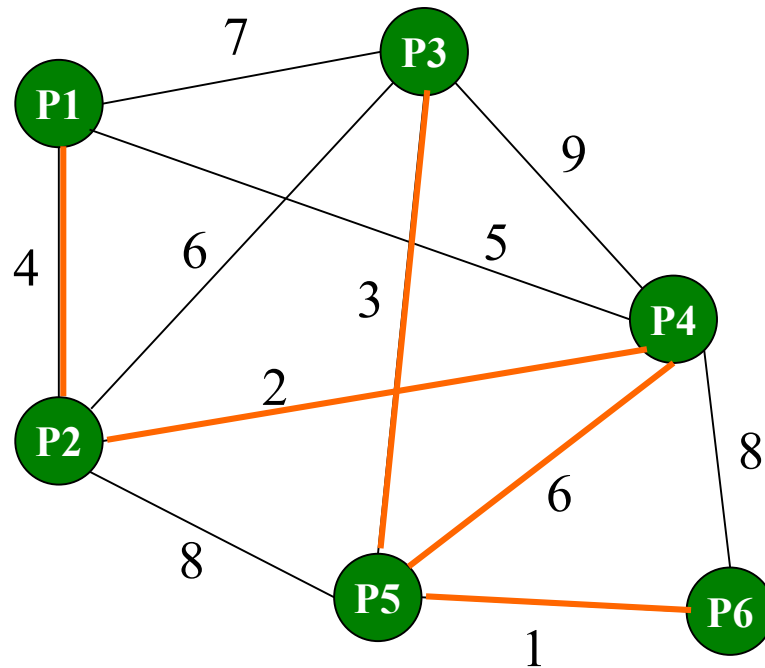
Complexity? 🗨️

How to cope with failures?



Kruskal's Algorithm for MST

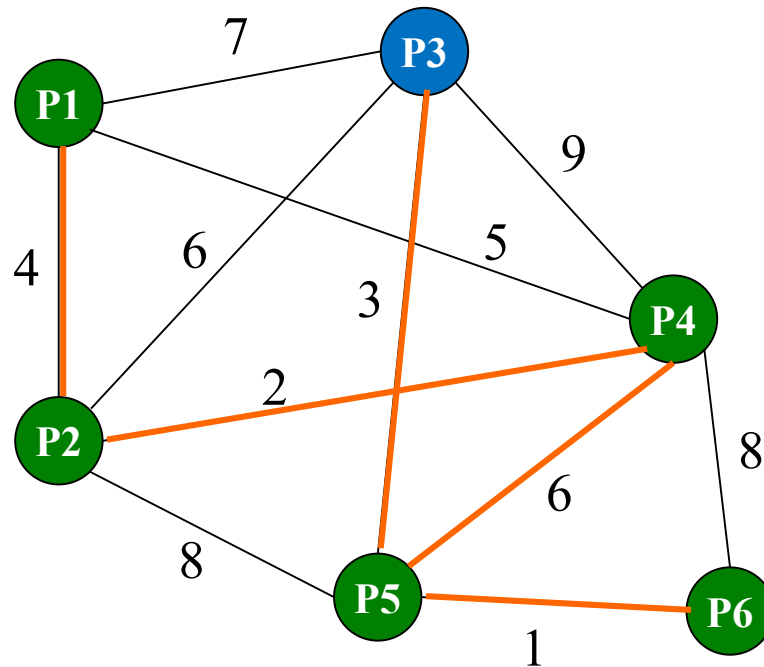
1. Given a connected graph $G=(V, E)$, sort all the edges E in non-decreasing order of their weight. Initialize tree T to empty.
2. Pick the smallest edge e in E . If adding e to T does not result in a cycle, add e to T ; otherwise discard e .
3. Repeat step 2. until all vertices are in T .



Can the idea of this algorithm be used in a distributed environment?
Why?

Prim's Algorithm for MST

1. Given a connected graph $G=(V, E)$, initialize a tree T with a single vertex, chosen arbitrarily from V .
2. Let e be minimum-weight edge that connects T to the vertices not yet in the tree, add e to T .
3. Repeat step 2 until all vertices are in the tree.



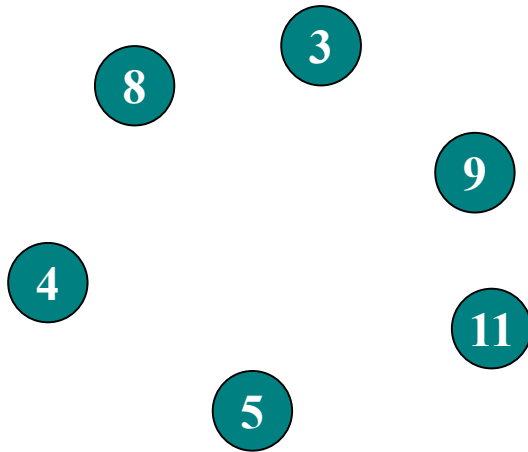
Can the idea of this algorithm be used in a distributed environment?
Why?

Lack of Global Knowledge (2)

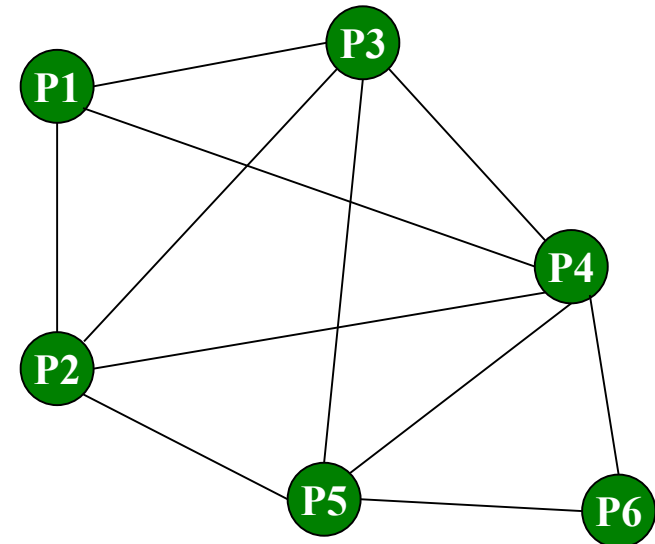
❑ **Leader Election:** how to elect a leader among a set of processes?

■ Problem application:

- Token generation
- Coordinator election



A **fully connected** network where every node can communicate with one another.



incomplete graph

Heterogeneity

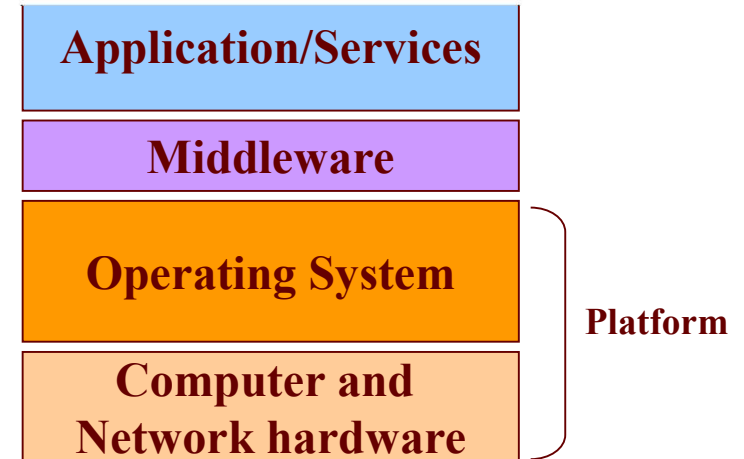
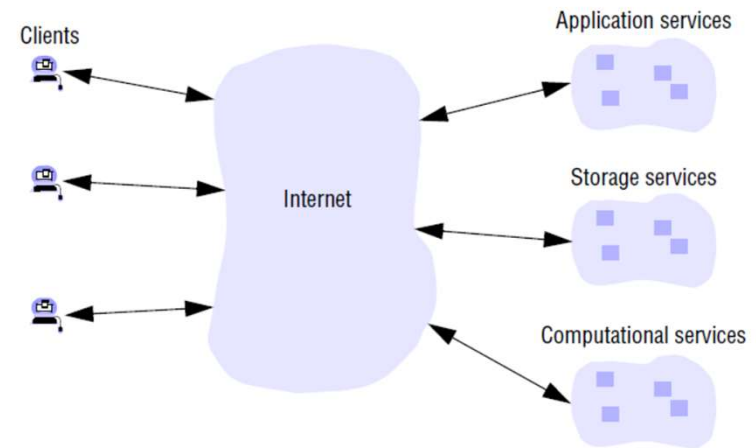
Source of heterogeneity:

- ❑ Networks
- ❑ Computer hardware
- ❑ Operating systems
- ❑ Programming languages
- ❑ Implementations by different developers

Possible solutions:

- ❑ Standard **Protocols** (e.g., TCP/IP)
- ❑ **Middleware** (e.g., CORBA, DCOM, RPC, Java RMI),
- ❑ Virtual machine & mobile code (e.g., Java byte code)

Cloud computing



Software and hardware service layers in distributed system

Openness

- A system can be open or closed w.r.t.
 - Hardware extensions (e.g., the addition of peripherals, memory, or communication interfaces) or
 - Software extensions (e.g., the addition of operating system features, communication protocols, and resource-sharing services).
 - ✚ Internet-related documents and specifications are published through “Requests For Comments” (RFC), each of which is identified by a number.
- Openness enables systems to be extended in various ways.
- Openness is achieved by specifying and documenting the key software interfaces of a system and making them available to developers.

The Value of Open Standards

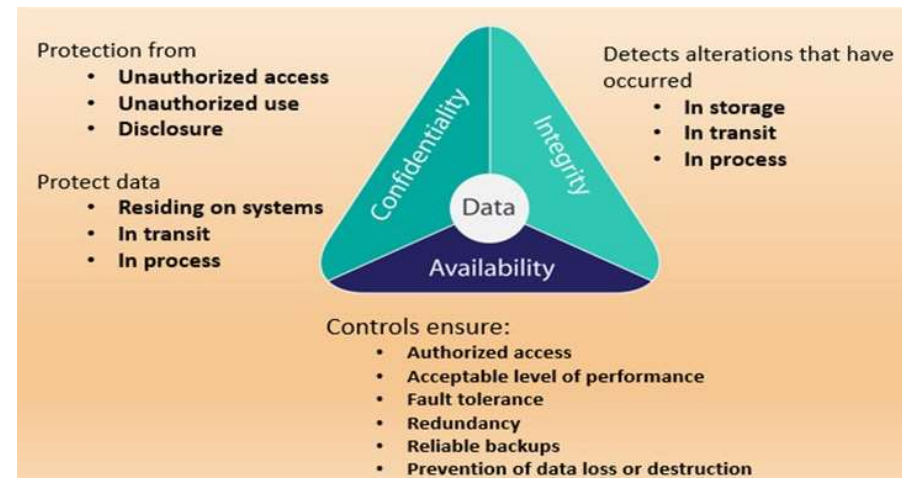
- ❑ Networking
 - *The Internet (TCP/IP)*
- ❑ Communications
 - *e-mail (pop3, SMTP, Mime)*
- ❑ Information
 - *World-wide Web (html, http, j2ee, xml)*
- ❑ Operating System
 - *Linux*
- ❑ Applications
 - *Web Services (SOAP, WSDL, UDDI)*
- ❑ Distributed Computing:
 - *Grid (Globus)*

Security

- ❑ Distributed systems are vulnerable to security threats due to their openness. Security needs to guarantee:

- Confidentiality
- Integrity
- Availability
- Authentication
- Nonrepudiation
- Access control

CIA-triad

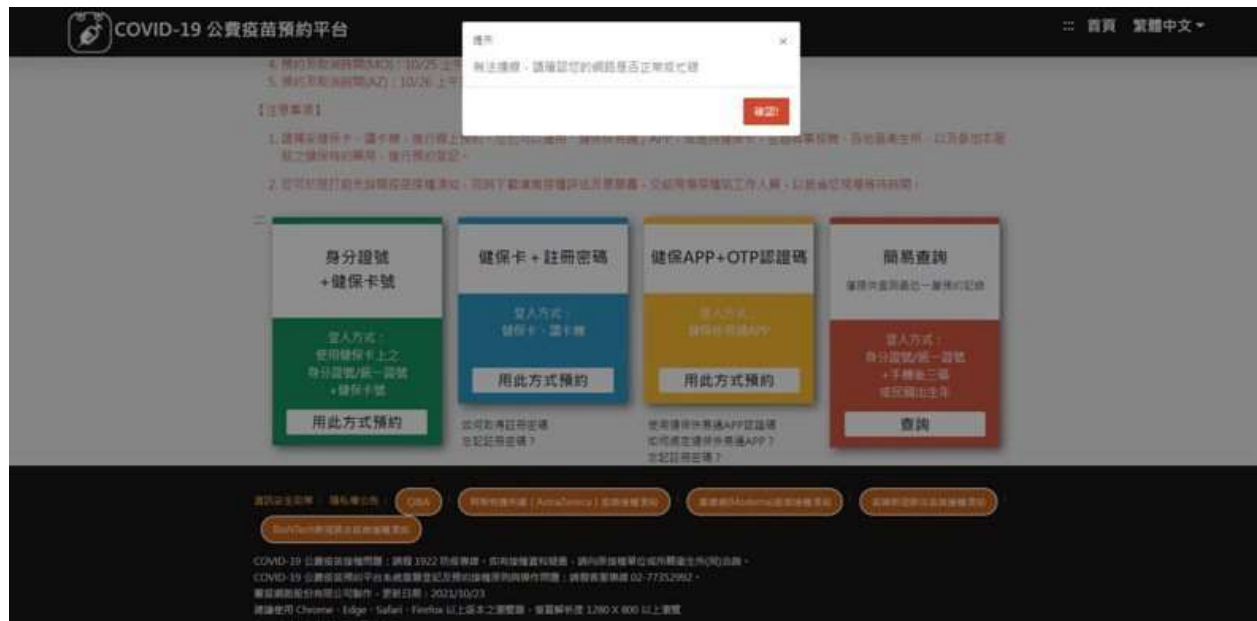


- ❑ Challenging issues:

- Distributed denial of service (DDOS) attacks
- Security of mobile code
- ...

Source: https://twitter.com/glenn_axelrod/status/1090449316706160643

Scalability



中央今日開放BNT第一劑與莫德納第二劑疫苗預約接種，許多民眾搶在上午10時前守在電腦前，未料民眾進入「唐鳳系統」時，就會出現「無法連線，請確認你的網路是否正常或忙碌」。

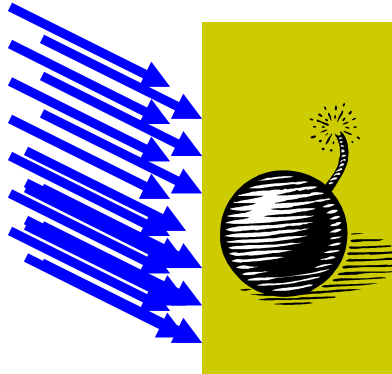
部分民眾就算進入排隊頁面，一樣會卡住，動彈不得，網路上哀鴻遍野...

疫苗預約接種系統又當機 民眾批網站「只有登出不用等」，UDN 2021.10.25

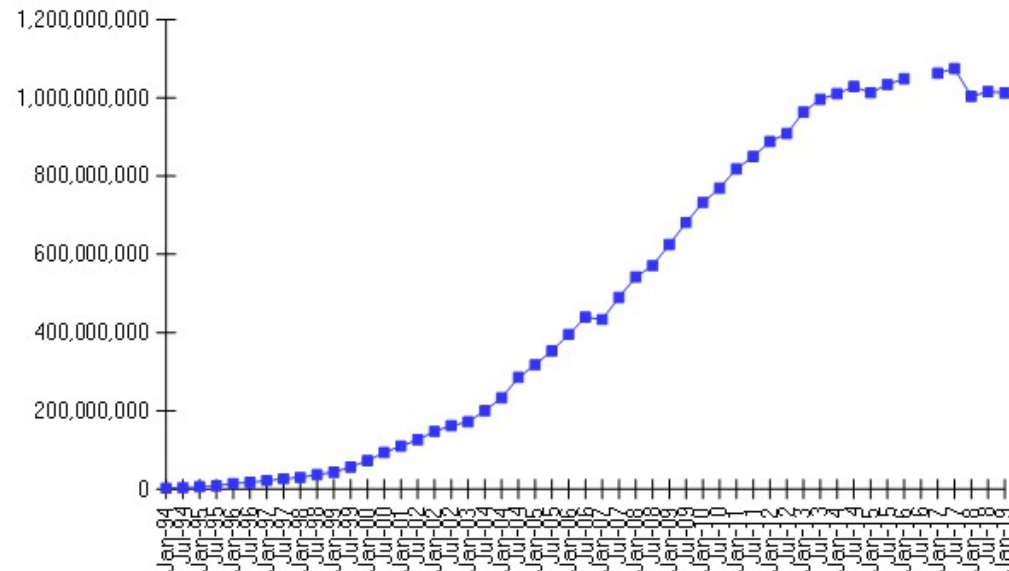
Scalability

- ❑ Traditionally designs concern only hundreds to thousands of simultaneously requests, but today millions of requests could easily arrive simultaneously.

Request




Internet Domain Survey Host Count



Source: Internet Systems Consortium (www.isc.org)

In Jan. 2019, approximately 1.01B internet hosts were available on the DNS. (source: <https://www.isc.org/survey/>)

Scalability

- ❑ A system is **scalable** if it will remain effective when there is a significant increase in the number of resources and the number of users. 
 - In distributed systems, we are generally concerned with “load scalability”
 - Scalability may also concern software resources, e.g., IP addresses (IPv4 vs. IPv6)
- ❑ Examples:
 - The domain name system (DNS)
 - Peer-to-Peer (P2P) systems, in particular, BitTorrent (BT)
- ❑ Ideally, the system and application software should not need to change when the scale of the system increases, but this is difficult to achieve.

Techniques for Scaling

- ❑ Decentralized data and computations across multiple machines
 - Decentralized naming services (DNS)
 - Decentralized information systems (WWW)
 - Move computations to clients (Java applets)
 - Distributed loads (Clusters, Grids)
 - ...
- ❑ Caching: Allow client processes to access local copies
- ❑ Replication: Make copies of data available at different machines
- ❑ Redesign the system
 - Poor performance is often a consequence of poor design.
 - Redesign, however, is costly

Scale Up vs. Scale Out

- ❑ **Scale up:** scale vertically
 - adding more resources (e.g., CPU, memory, hard disks) to a single node in a system
 - Complexity is handled by operating systems, and transparent to system operator
- ❑ **Scale out:** scale horizontally
 - E.g., adding more nodes to a system
 - More complex to manage
 - Not all applications are designed to embrace this model

Both approaches may have a limit --- Amdahl's Law

Amdahl's Law

- ❑ A law governing the speedup of using parallel processors on a problem, versus using only one serial processor.

speedup of a process:

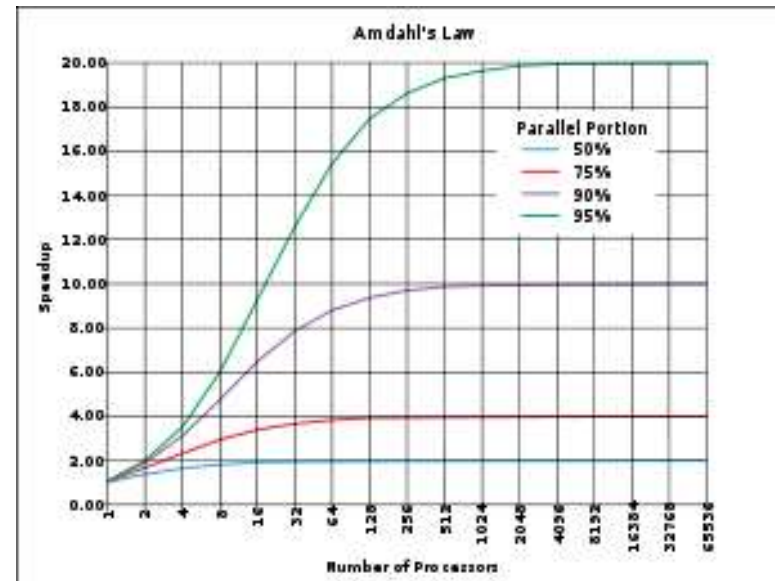
$$\frac{1}{(1 - \alpha) + \frac{\alpha}{P}}$$

where

α : the fraction of the process that can be parallelized

P : no. of processors t used to parallelize the process

Maximum speedup: $\frac{1}{1-\alpha}$

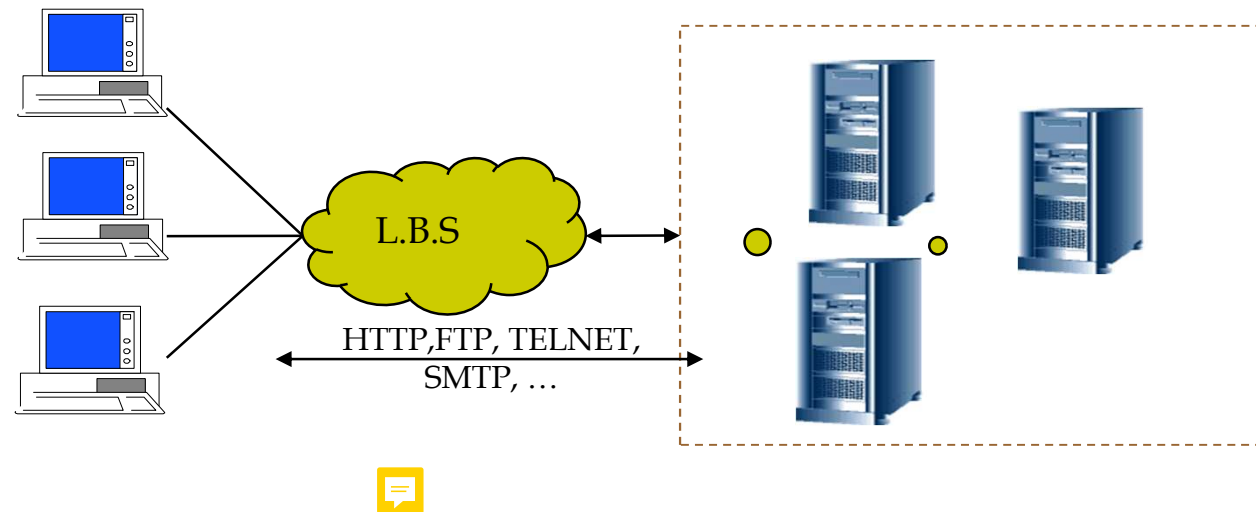


source: wiki

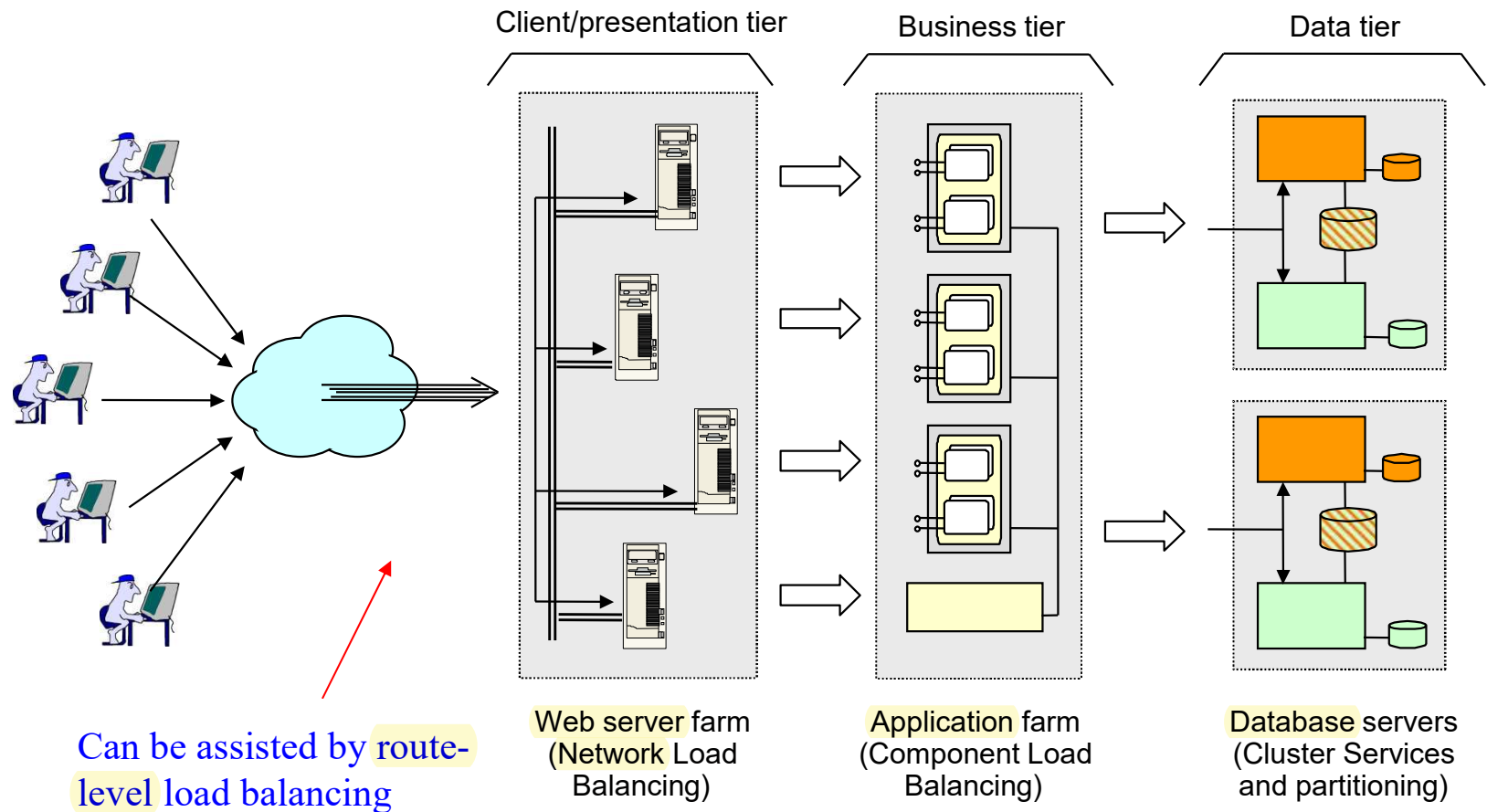
Scaling Out: Load Balancing

❑ Load balancing

- distribute workload evenly across multiple servers
- in addition to CPU cycles, load balance may also concern storage, bandwidth, or other types of computing resources

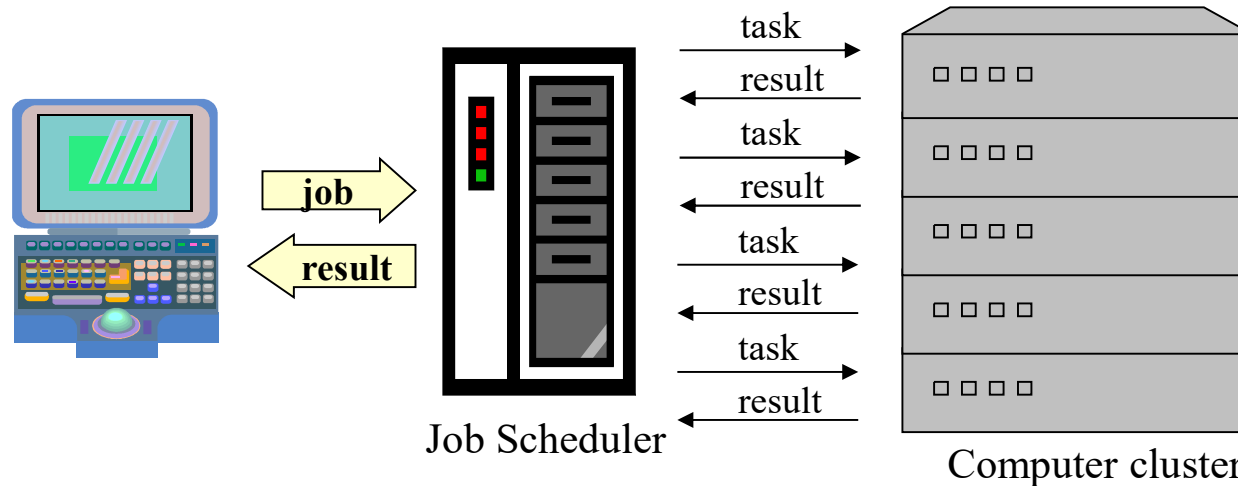


Load Balance: A Typical Multi-tier Example in Business



Scaling Out: Cluster Computing

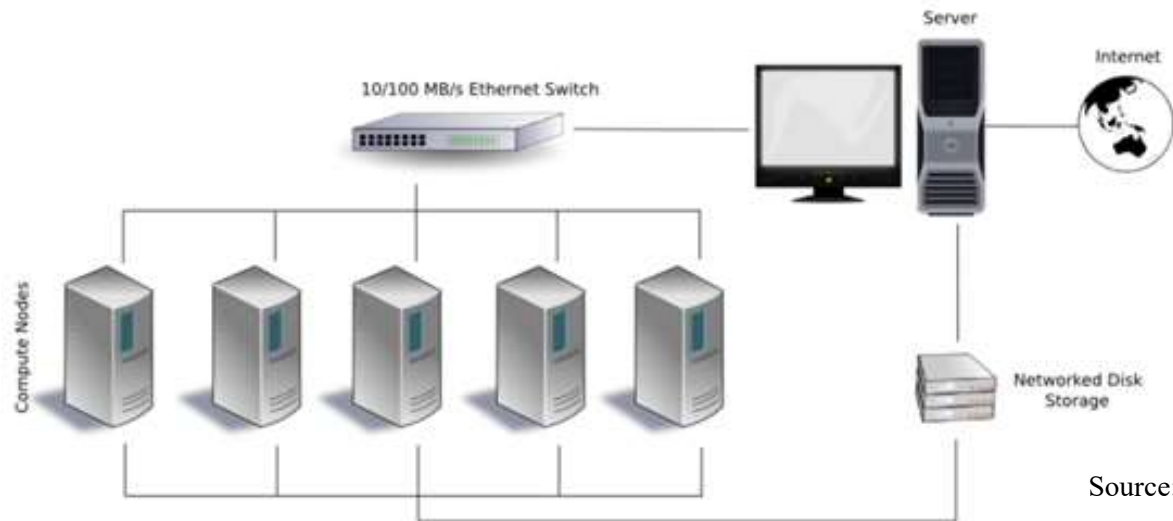
- ❑ Computer cluster: a group of tightly coupled computers



- ❑ Advantages of using a computer cluster:
 - Performance
 - Load Balance
 - Availability
 - Cost-effective
 - Nodes in a cluster are typically low cost “commodity” grade

A Typical (Beowulf) Cluster

- ❑ Consisting of **one server node** plus one or more client nodes connected together via LAN.
- The **server node controls the whole cluster**, and provides user interface, management, and job scheduling services to the cluster.



High-performance Clusters

- ❑ Today's super computers are primarily based on cluster computers technology.
 - High performance, high availability, low cost



台灣杉二號在11月發布的全球超級電腦500強中，以9 PFLOPS的實測計算效能，躋身第20名，創下臺灣超級電腦史上最佳紀錄, 2018.11.25

TOP500 Supercomputer List

□ NOVEMBER 2020



富岳(Fugaku) 超級電腦, No. 1 in 2020.06, 2020.11

富士通與日本理化學研究所(RIKEN)共同開發，運算速度是第二名的3倍之多，造價US\$1B

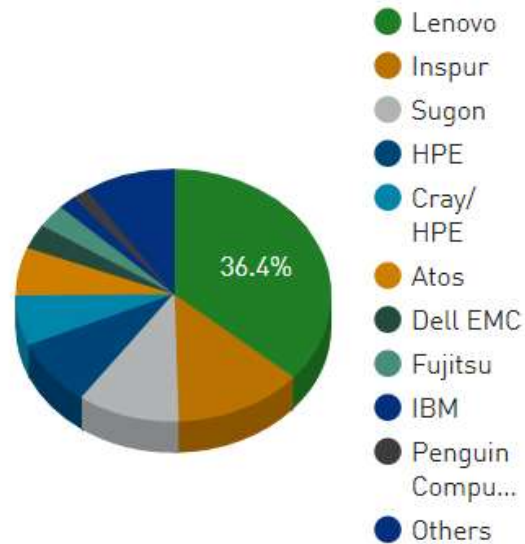


天河二號, No. 1 in 2013-15

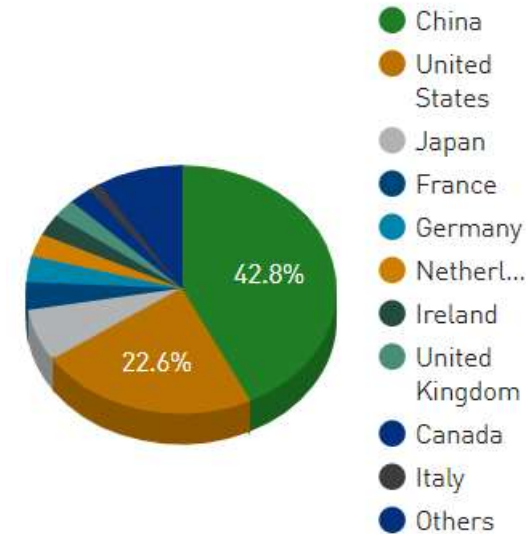


TOP500 Supercomputer Statistics

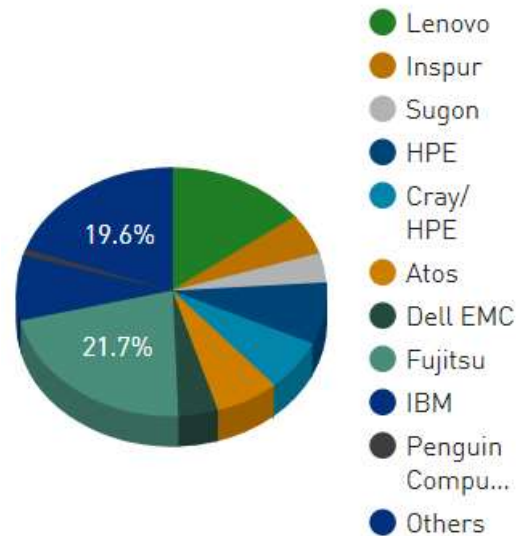
Vendors System Share



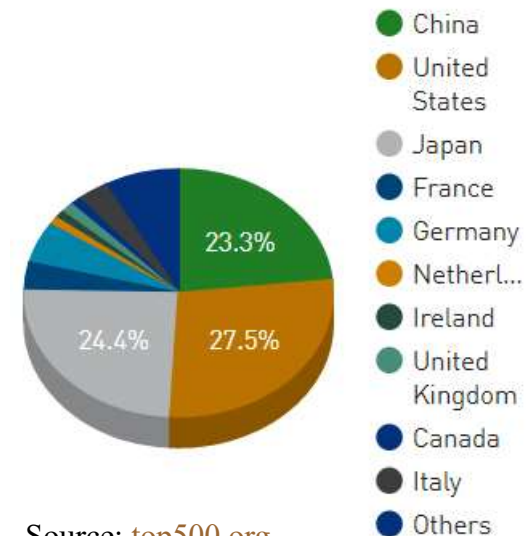
Countries System Share



Vendors Performance Share



Countries Performance Share



Scalability: Database Partitioning

❑ Advantages:

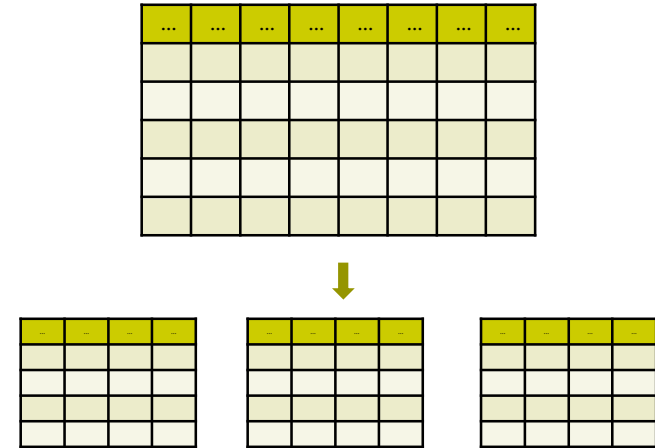
- manageability
- performance
- availability

❑ Horizontal vs. Vertical partitioning

■ Horizontal: partitioning rows

- Range/list partitioning
 - Zipcode, Last Name, IDs, Country...
 - Partition needs not be disjoint (so as to increase availability)
- Hash partitioning

■ Vertical: partitioning columns

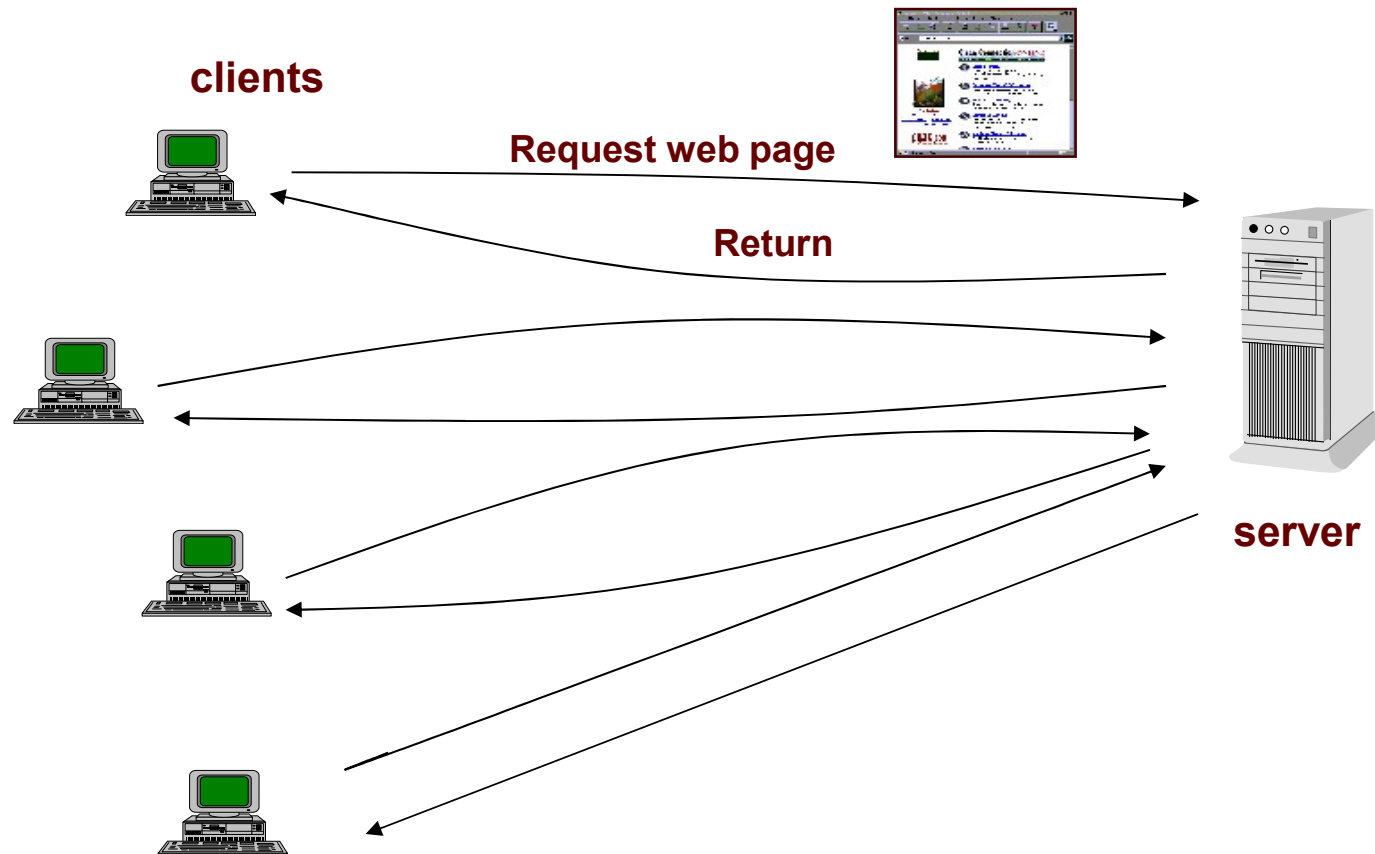


疫苗預約接種系統又當機 民眾批網站「只有登出不用等」, UDN 2021.10.25

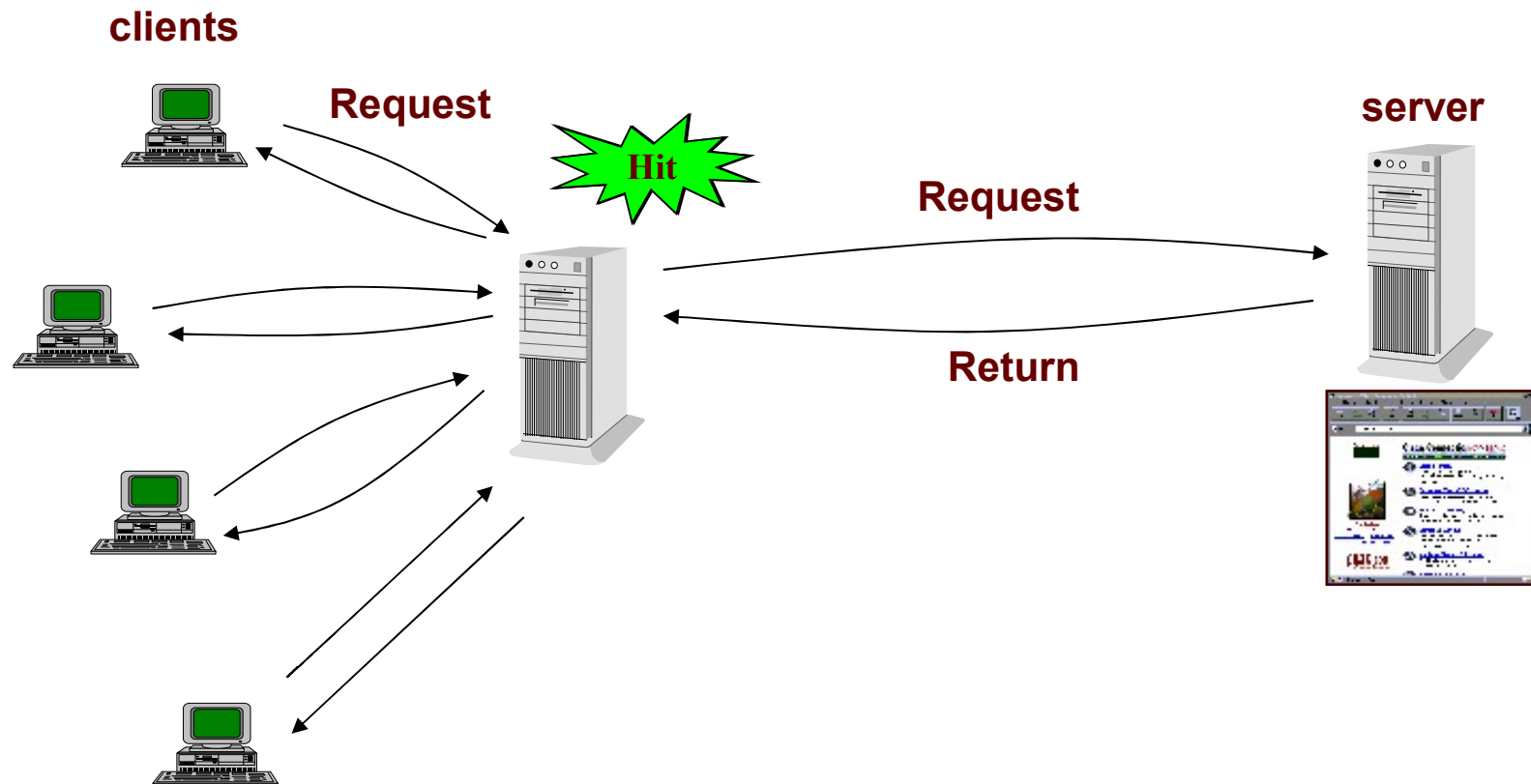
Caching

- ❑ Assumption: Data that is recently accessed will be accessed again in the near future with high probability.
 - When you retrieve/download some data, leave it at your hand for a while before giving it away
- ❑ Where has this concept been applied?
 - Web Proxy
 - ADSL
 - Hard Disk
 - File caching (e.g., coda file system)
 - Mobile Apps...

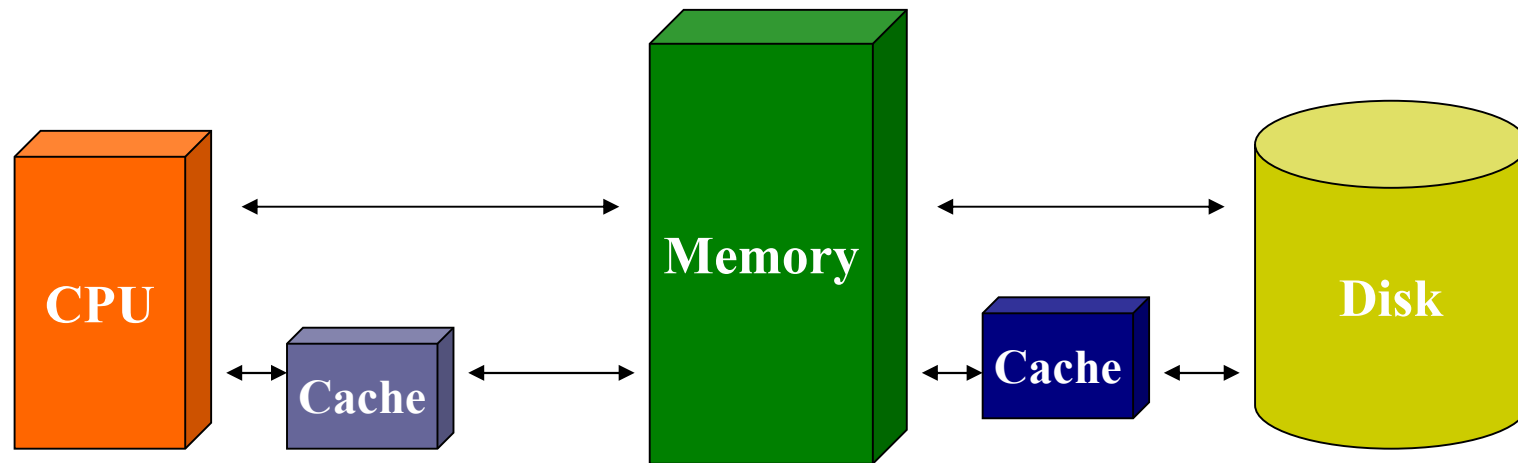
Web---Without Proxy



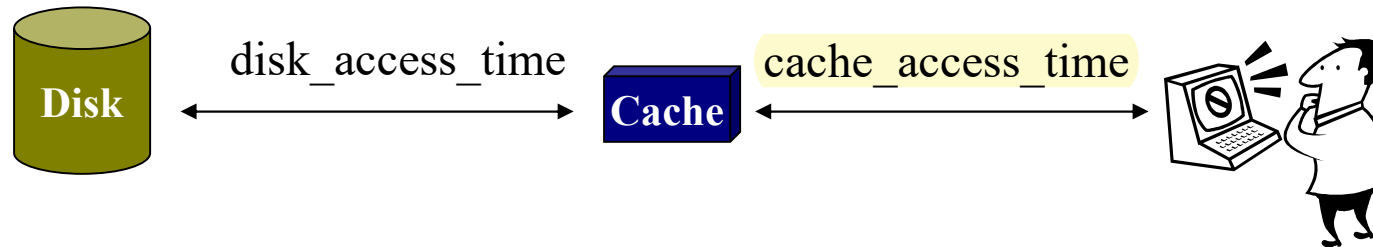
Web---with Proxy



Cache in Computer Memory



Performance Boosted



Average access time:

$$\text{cache_hit_rate} \times \text{cache_access_time} + (1 - \text{cache_hit_rate}) \times (\text{disk_access_time} + \text{cache_access_time})$$

Example:

$$\text{cache_hit_rate} = 0.9$$

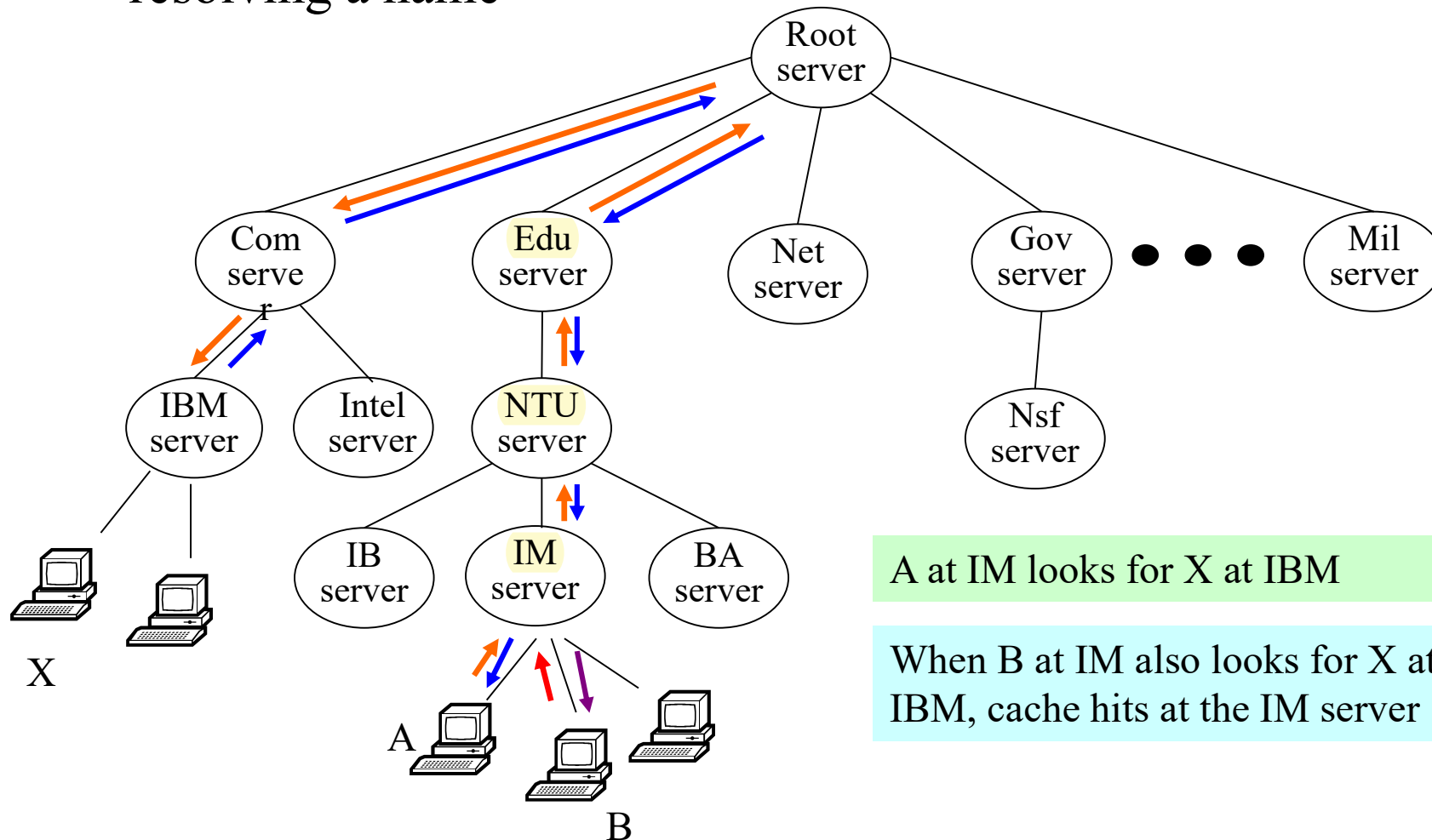
$$\text{disk_access_time} = 1$$

$$\text{cache_access_time} = 0.2$$

$$\text{Average access time} = 0.9 \times 0.2 + 0.1 \times (1 + 0.2) = 0.3$$

Domain Name System (DNS)

- ❑ By using cache, very few servers need to be contacted when resolving a name

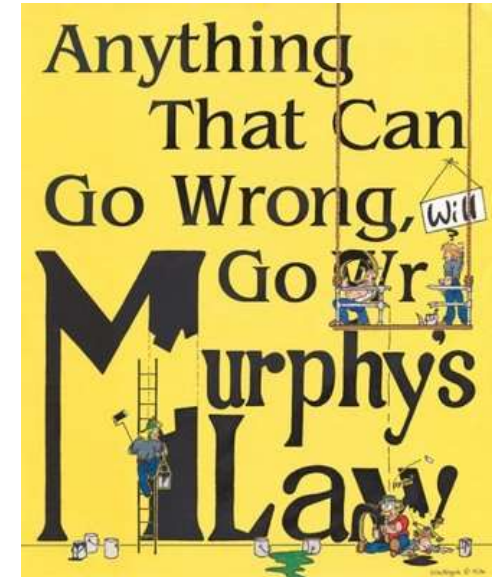


Dealing with Cache

- ❑ Size of cache
 - Application dependent
- ❑ How long is the cached content valid?
 - Time-To-Live (TTL)
 - Again, application dependent
- ❑ How to replace old content
 - Least Recently used (LRU)
 - First in first out (FIFO)
 - Least frequently used
 - Random
 - ...

Failure Handling

- Detect failures
 - ▶ Checksums, timeout
- Mask failures
 - ▶ Handling failures in different levels
 - ✓ ISO 7 layers
- Tolerate failures
- Recover from failures (“rolled back”)
- Fault tolerance is usually achieved by **redundancy**, which also increases availability.
 - ▶ **Replication**



Why Replication

❑ Redundancy is the basis of fault tolerance

■ Performance enhancement

- by placing copies of recently and/or frequently accessed data at both clients and servers to eliminate bottlenecks due to processing and network inadequacies.
- **replication factor**: how many copies of data should exist

■ Increased Availability and Reliability

- For n servers each with a probability of failure of p the availability becomes: $1 - p^n$.
 - For 2 servers, each with a 5% probability of failure, the likelihood of the provided service being unavailable decreases to 99.25%.
 - E.g., Disk Array, RAID

Issues for Replication

❑ Transparency

- Clients using the service are unaware of the replication.

❑ Consistency

- The result of an operation should be independent of which replica the operation is applied.
 - How to ensure that the different copies of data have the same value?
 - Is this really needed?
 - Tradeoff between consistency and performance

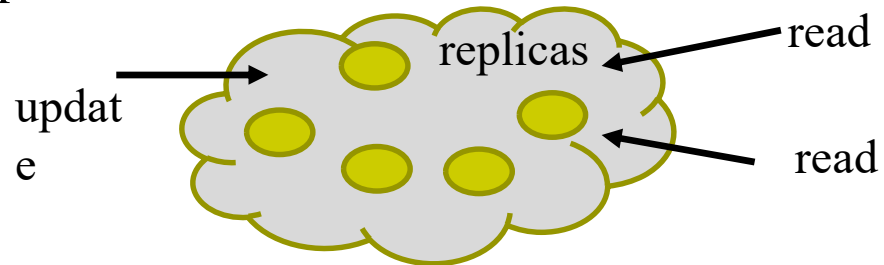
Strong vs. Eventual Consistency

❑ Strong consistency

- Any two reads after an update see the same update
- Various implementations, but all are costly
 - E.g., two-phase locking
 - Paxos algorithm + quorums

❑ Eventual Consistency

- If no new update to an entity, all reads of the entity will eventually see the last updated value
- Some implementations
 - “lazy” updates or gossip protocols



Need some mechanism to ensure consistency

Consistency in the presence of failures

- ❑ Consistency can theoretically be impossible to achieve in some failure models
- ❑ But practically there are algorithms to guarantee consistency in most usage scenarios
- ❑ Also some techniques to deal with eventual consistency
 - Hinted Handoff
 - Store a “hint” about where a temporary replica belongs to
 - Used in Apache Cassandra and DynamoDB

Availability and Reliability

□ Availability

$$\begin{aligned} &= \frac{uptime}{uptime + downtime} \\ &= \frac{MTBF}{MTBF + MTTR} \end{aligned}$$

High availability does not necessarily imply high reliability, and vice versa, although often times the two terms have been considered as equivalent.

MTBF: Mean Time Between Failure

MTTR: Mean Time To Recovery/Repair

■ Metric of “nines”

- “four nines” = 0.9999 (99.99%) availability.

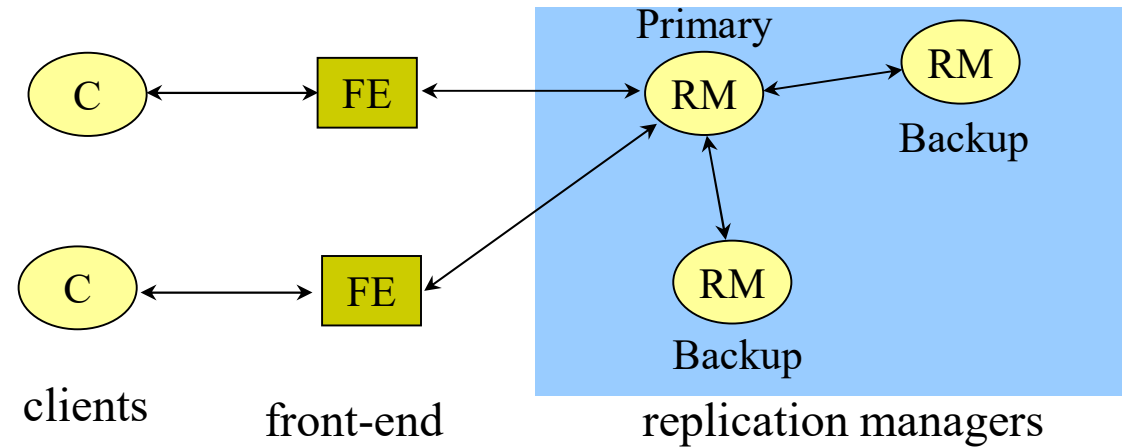
□ Reliability: how “reliable” a system is

■ several metrics

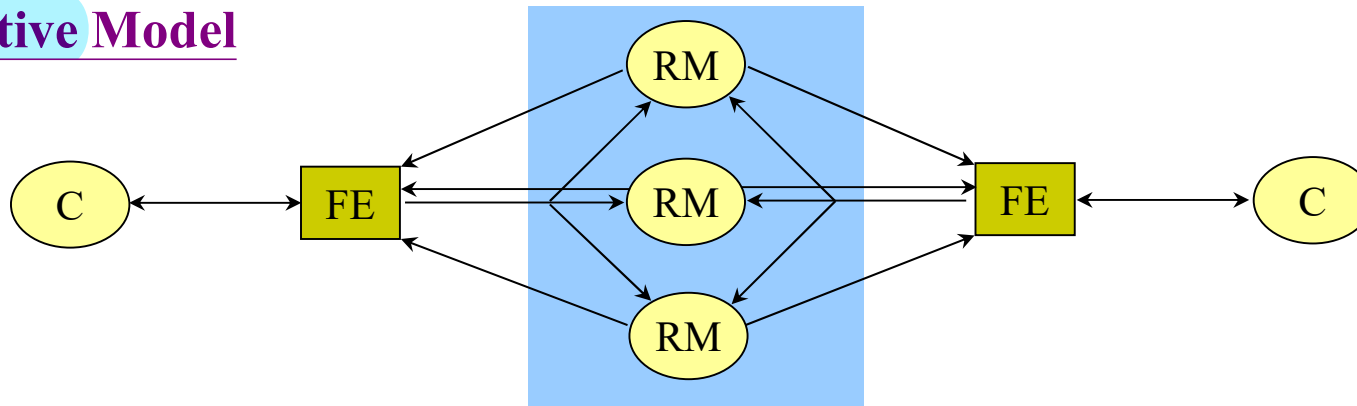
- failure rate, MTBF,

Typical Replication Models

The (Passive) Primary-Backup Model

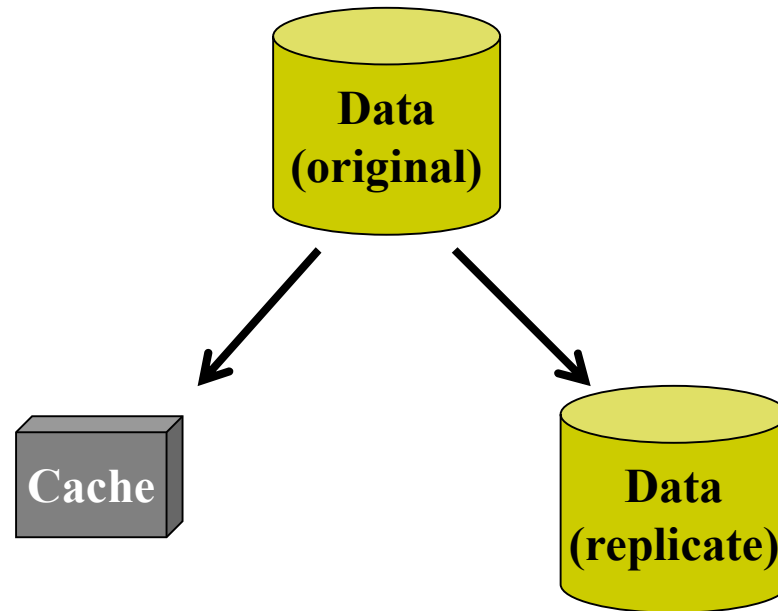


The Active Model



Replication vs. Cache

- ❑ Replicated Data
 - Permanent
- ❑ Cached Data
 - Transient
- ❑ Both need to deal with **data consistency**



Transparency

- ❑ Transparency conceals the users and the applications programs from the separation of components in a distributed system.
 - In a nutshell, to make the system look like there is only one component, and only one user is in it.
- ❑ ISO Reference Model for Open Distributed Processing (RM-ODP) [1992] identifies 8 forms of transparency:
 - **Access transparency:** enables local and remote information objects to be accessed using identical operations.
 - **Location transparency:** enables resources to be accessed without knowledge of their physical or network location.
 - Are URLs location-transparent?
 - access and location transparency are sometimes referred to together as **Network transparency**

