Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Introduction to Scalable Systems

Chun-Feng Liao

廖峻鋒

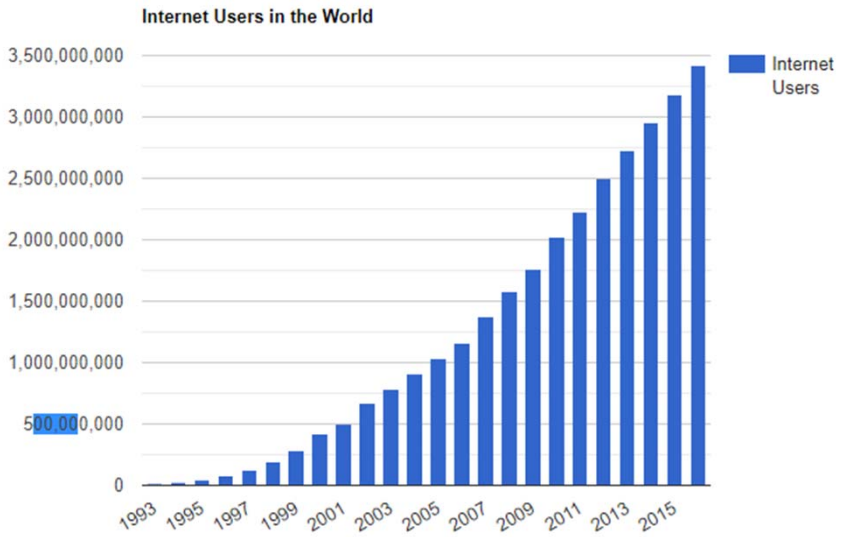
Dept. of Computer Science

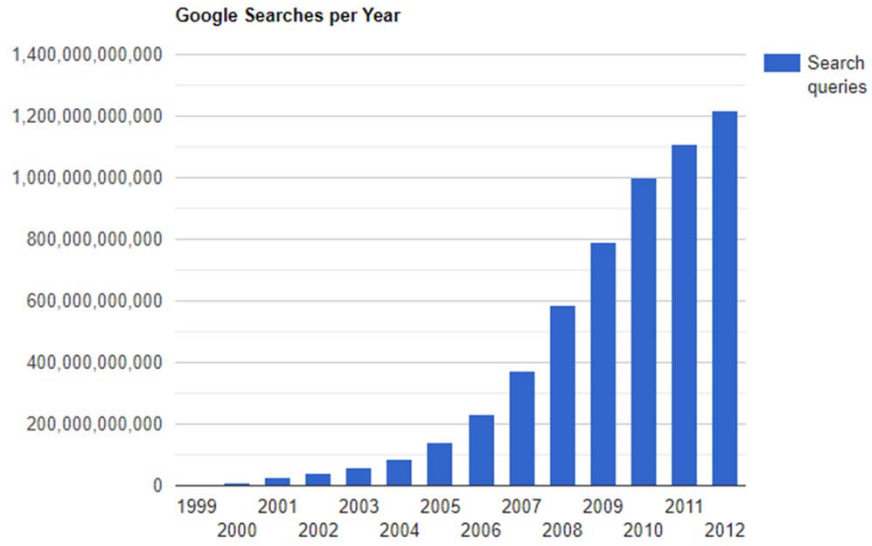
National Chengchi University

Trends of Distributed Computing

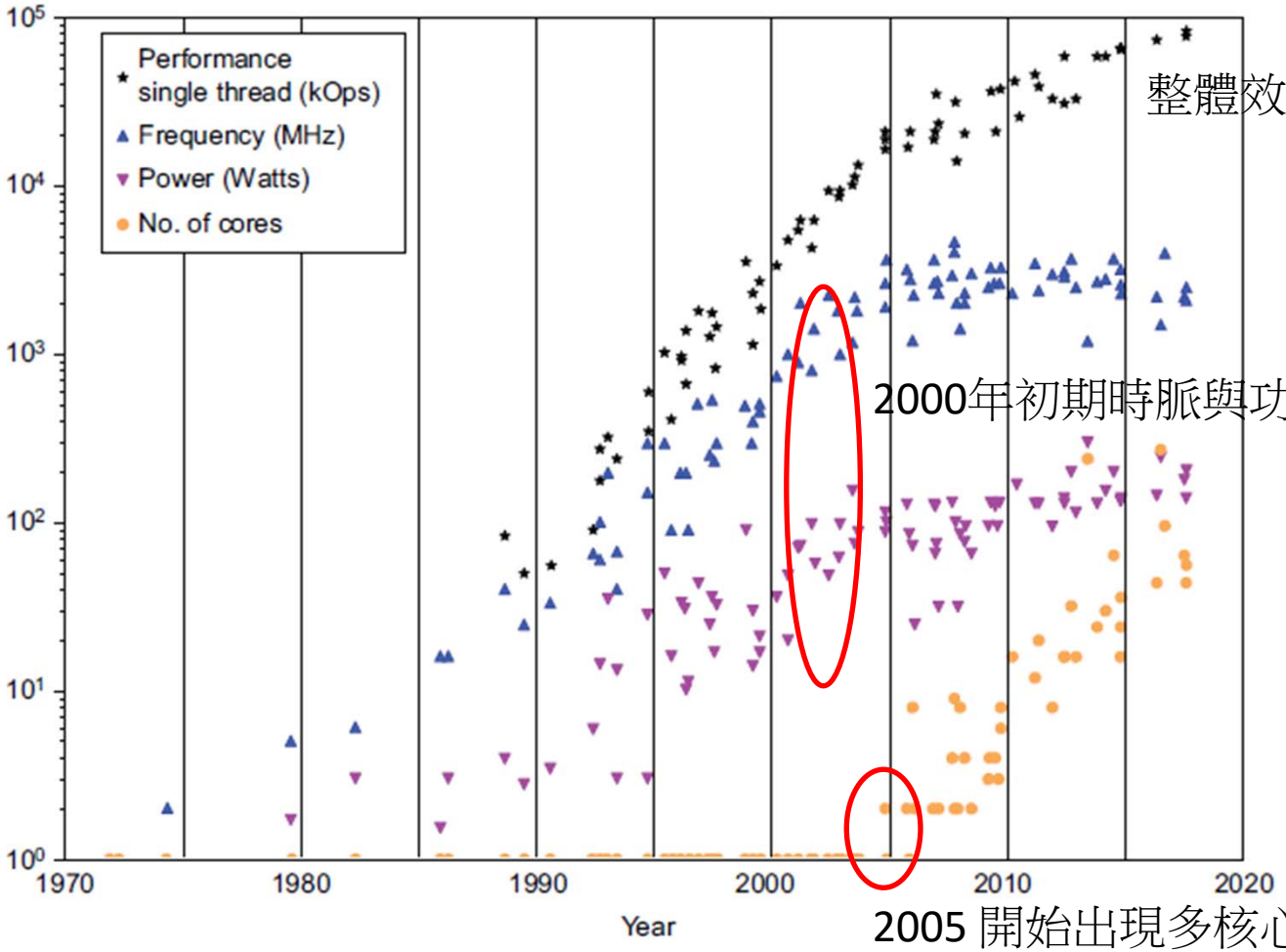
- Cloud native
  - Hyper scalable
  - Constantly changing
  - 24x7 availability

* Google handles around 3.5 billion search requests a day
* Google (2016) 內部系統包含20億行程式碼，共86TB
*Instagram uploads about 65 million photos per day





Technological Trends



整體效能還是不斷有所提升

2000年初期時脈與功率就不再增加

2005 開始出現多核心

A Brief History of Distributed Systems

- 1980s
  - PCs before 1980s are rarely networked
  - Most used networked apps: email
- 1990-1995
  - WWW
  - Yahoo! ; Amazon; eBay

↗ e-commerce

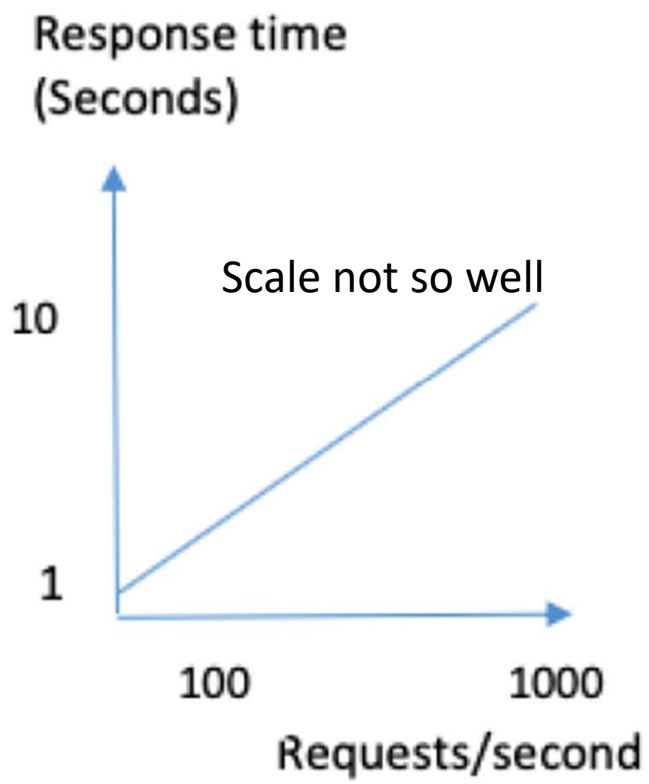
- 1996-2000
  - Web sites grew from 10000 to 10 million
  - E-business
  - .com bubble

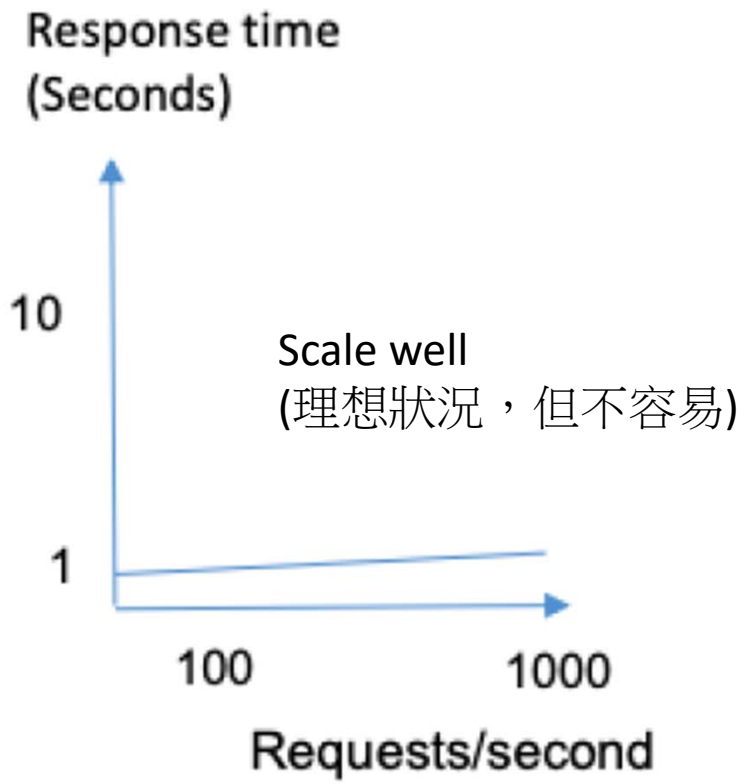
the period between 1995 and 2000 when investors pumped money into Internet-based startups in the hopes that these fledgling companies would soon turn a profit.

A Brief History of Distributed Systems

- 2000-2010 → Amazon Elastic Compute Cloud
  - AWS EC2 (2004)
  - Youtube (2005)
  - Facebook (2006)
- 2010-2015
  - Cloud computing, IoT, Blockchain
  - NoSQL
  - Netflix
- 2016-now
  - Cloud native and hyper scale services
  - Large scale data processing (AI)

Scalable Systems

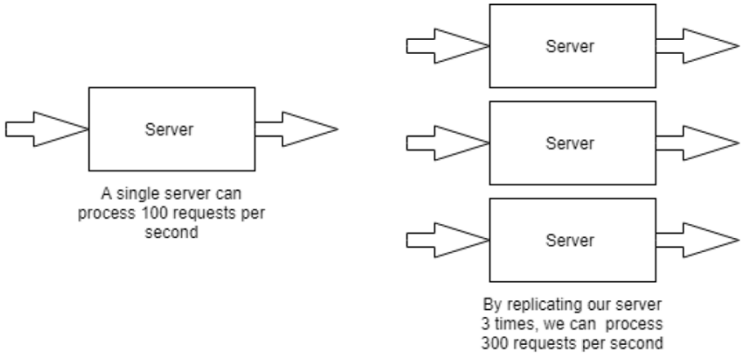




Scalability Basic Design Principles

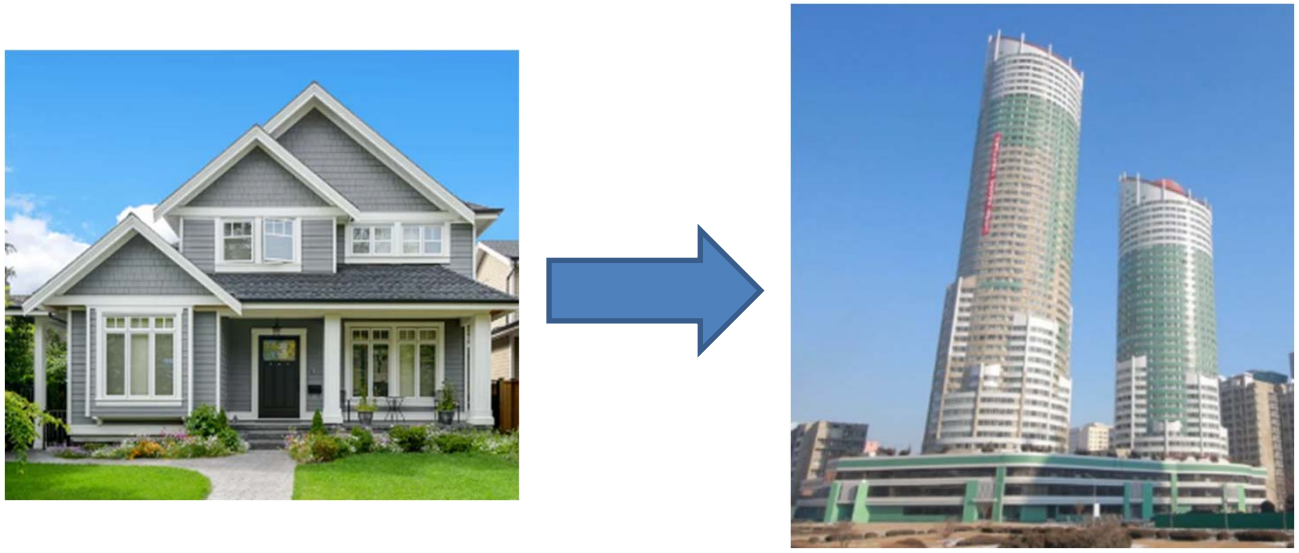
- Two strategies
  - Replication
    - To add capacity
    - Ex: Nippon Clip-ons

  - Optimization
    - To observe characteristics of the traffics
    - 觀察流量特性，並加以調控
    - Ex: 車流: 早上多入城，晚上多出城



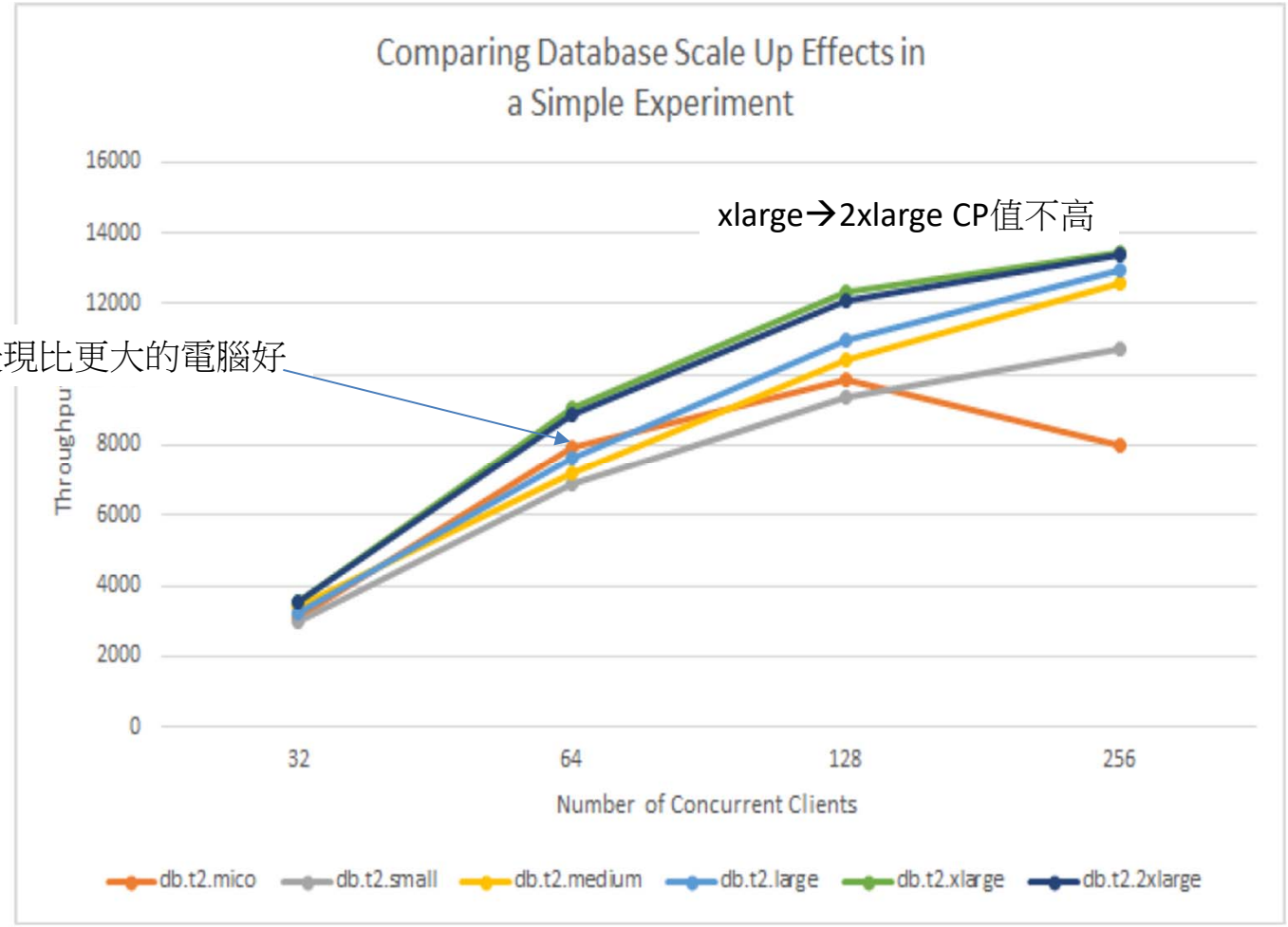
Sometimes it is not possible to scale

- The underlying software architecture limits the scalability



Adding more resources makes your system faster?

Micro在60 clients以下表現比更大的電腦好

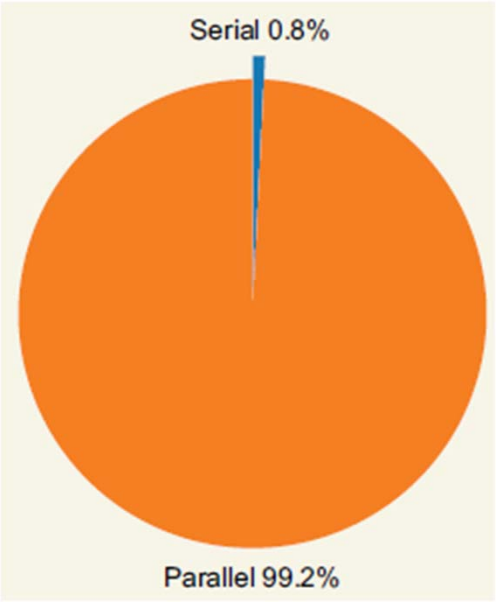


Scale之後延伸的問題

- Consistency
  - 資料複製多份後，若其中一份修改，則其它副本都要進行同步
  - 產生更多流量
- Performance
  - 網路本身的Latency
  - 因管理更多資源造成的新流量
- Availability
  - 要管的資源愈多，出錯機會愈大
- Security
  - 要管的資源愈多，攻擊平面愈大

效能提高的假象

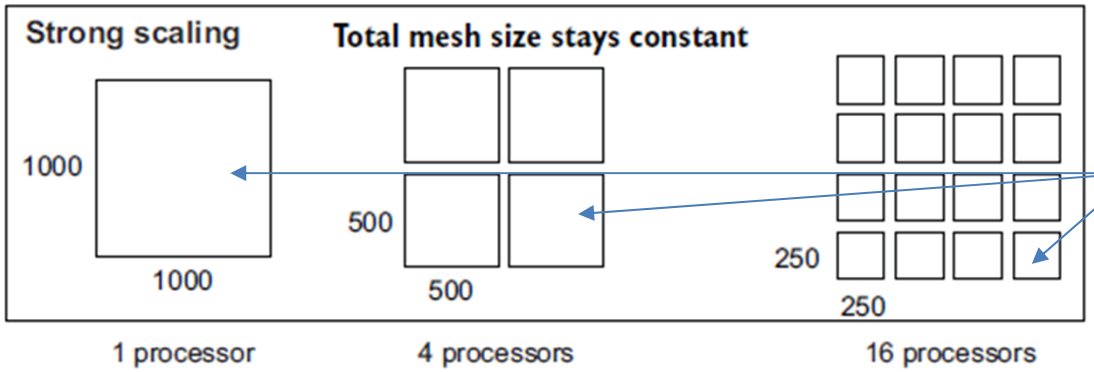
- 16-core 2-way hyperthreading CPU
  - 256-bit vector unit
    - 一次可執行4個64bits指令
    - 64*4=256
  - 同時可執行128(=16 cores * 2 way * 4)個指令
- 該電腦只運行循序程式的時段
  - 同時間只能執行1個指令 (e.g., critical section)
  - 所使用CPU資源 1/128 ~ 0.8%
- 觀察
  - 如果大部份程式都無法平行化，那麼電腦效能提高只是一種假象



Amdahl's Law ['emdol]

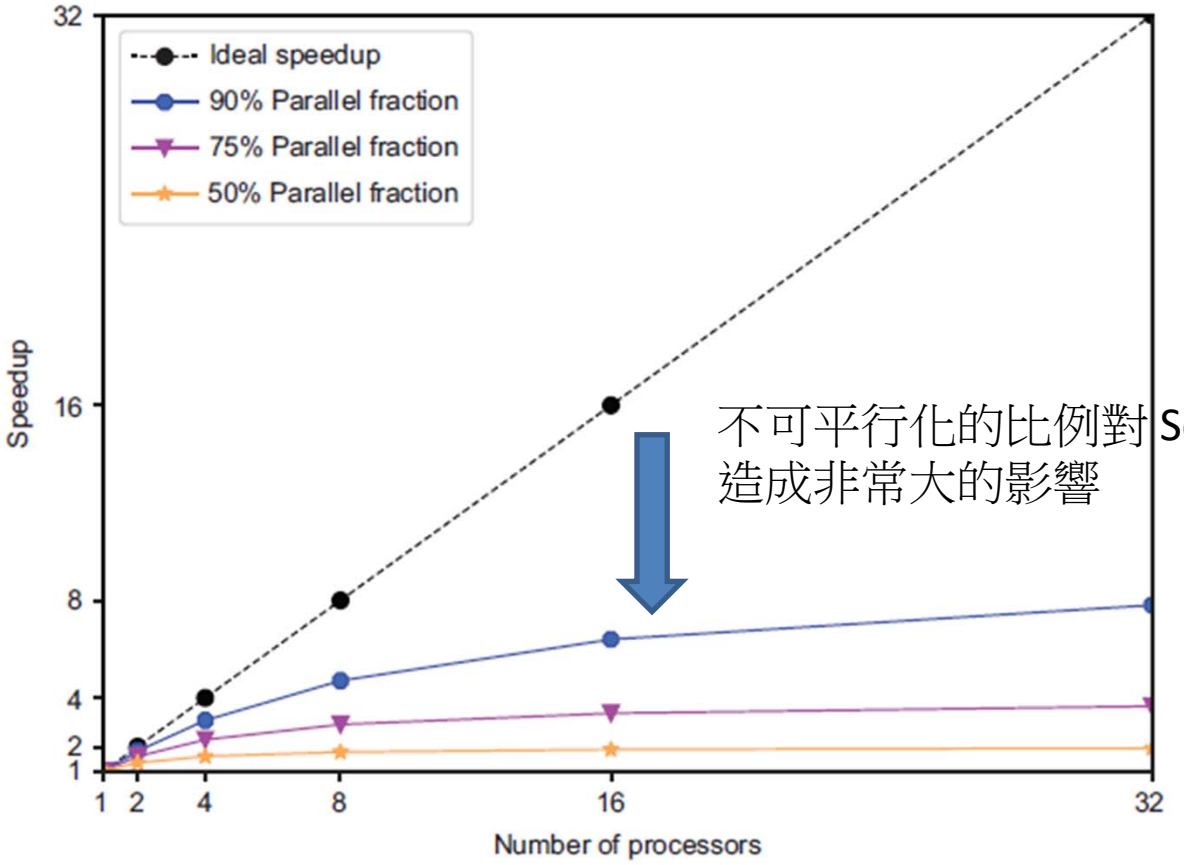
- 增加處理器所能造成的效能提升，受限於不可平行化的比例
  - N: 處理器數量
  - P: 可平行化的工時比例
  - S: 不可平行化的工時比例

$$SpeedUp(N) = \frac{1}{S + \frac{P}{N}}$$



總工作量固定
但每個CPU的可平行化工作量隨N上升而變少

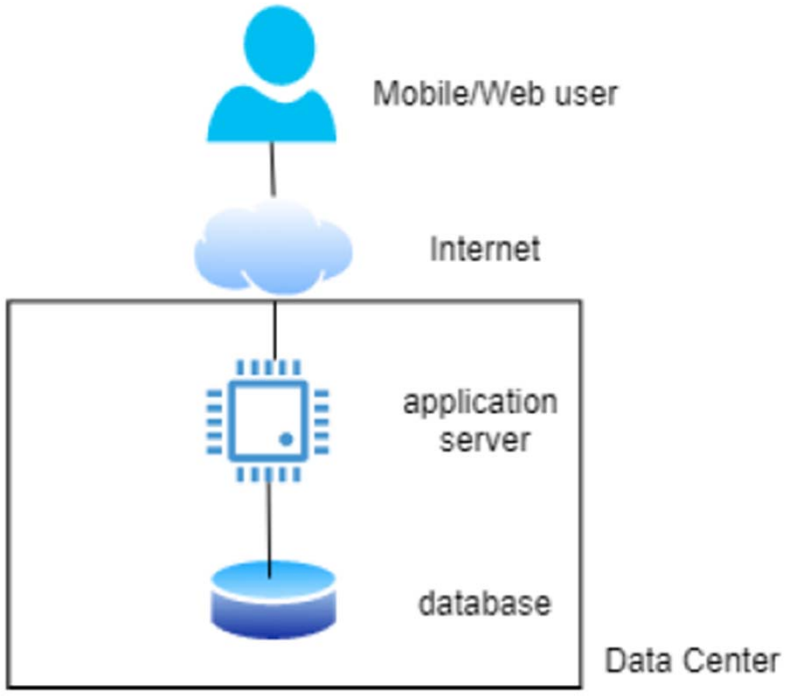
Amdahl's Law ['emdol]



不可平行化的比例對 Scaling的效度造成非常大的影響

A Scalable Design Example

- The baseline system (monolithic architecture)

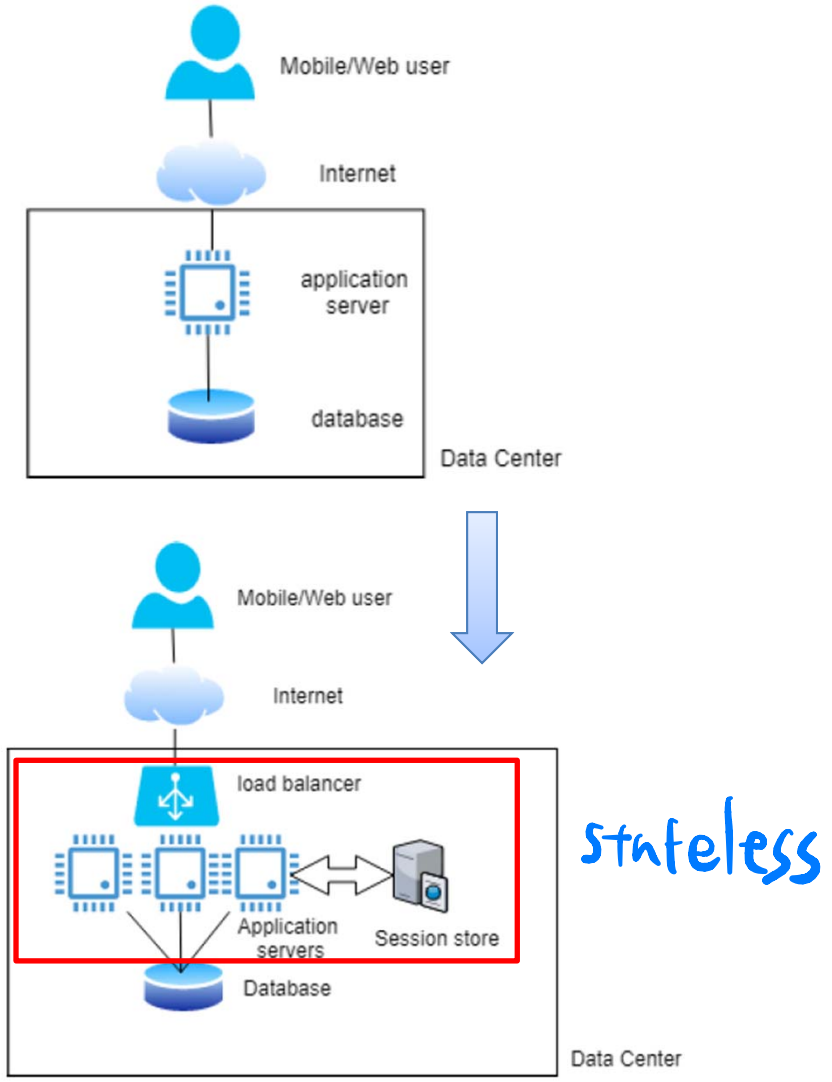


Scale方法:
Scale up: 更好的硬體
Scale out: 更多的硬體

verfical
horizontal

Scale Out

- Approach
  - A load balancer
  - Stateless services
    - 執行時期states統一存在session store
- Outcome
  - Application servers are scalable
  - Application servers are fault tolerant
    - 但無法回復未記錄的session states
  - DB eventually becomes the bottleneck



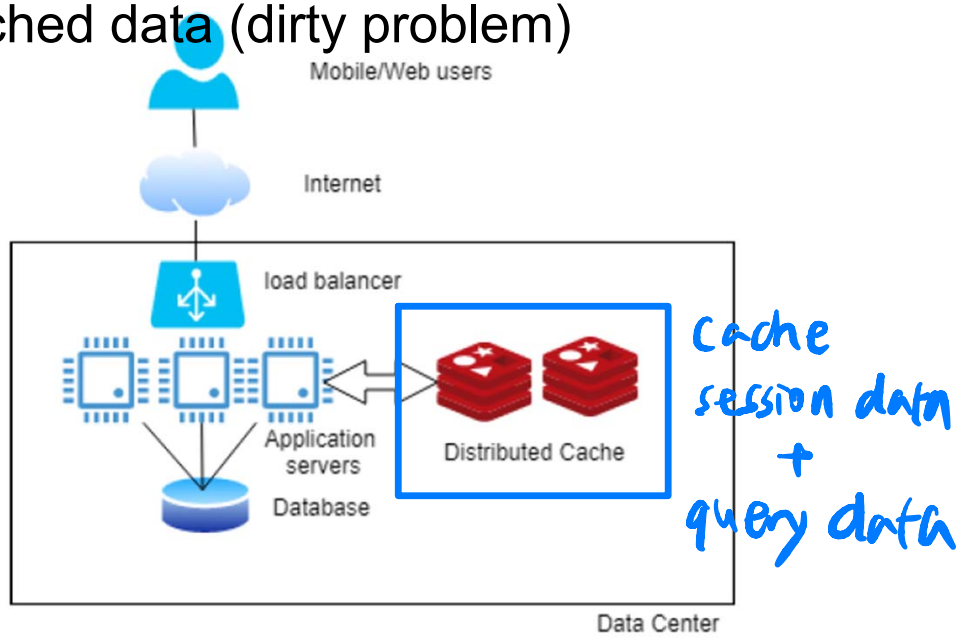
Stateless and stateful services differ mainly in how they handle data and user sessions:

Stateless Service: A stateless service doesn't store any data about a user's session. Each request from the client must include all the information the server needs to understand and respond to the request. This makes stateless services easier to scale because each request can be processed independently.

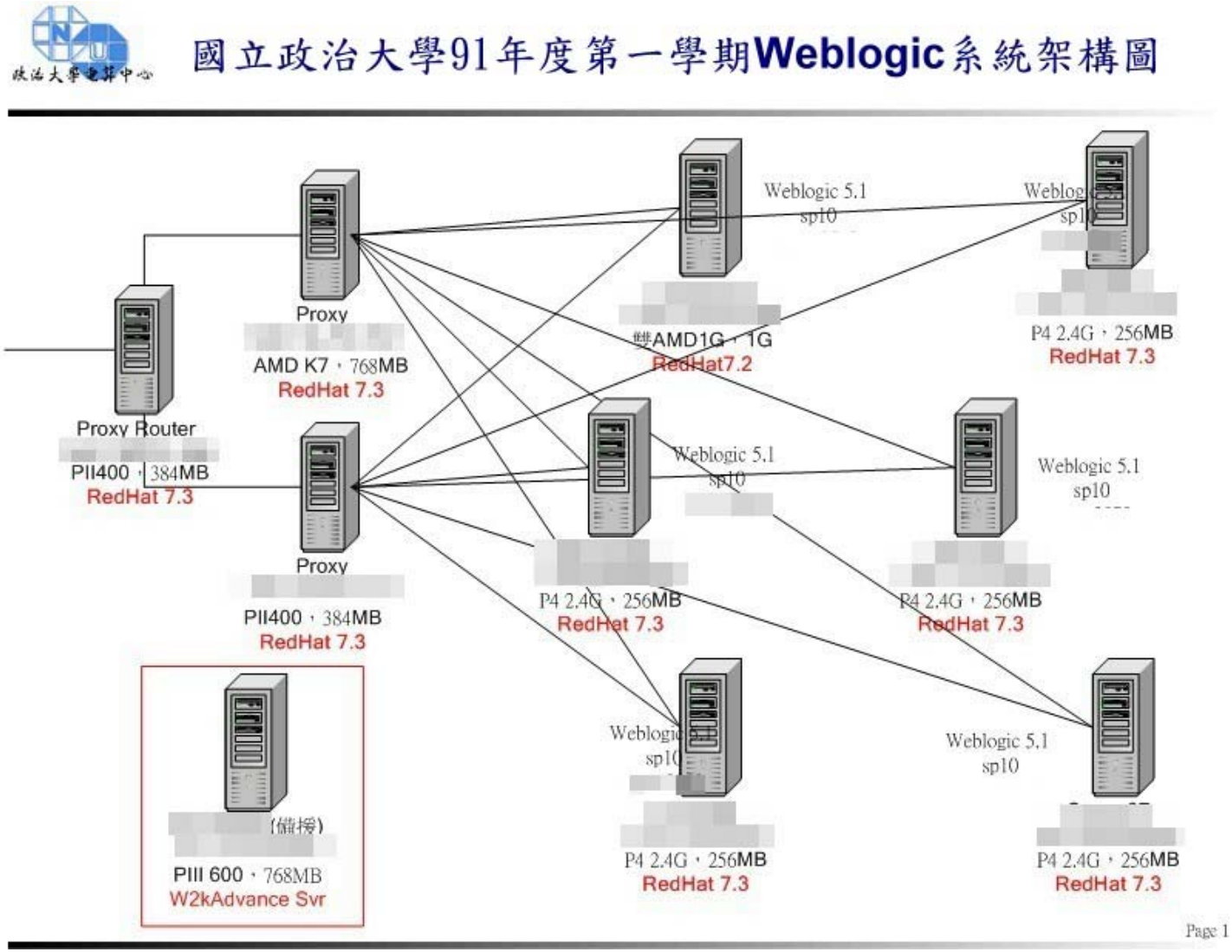
Stateful Service: A stateful service, in contrast, retains data about a user's session from one request to the next. This can simplify the client's job as the server remembers the state of user interactions. However, this makes scaling more complex because subsequent requests need to be routed to the server instance that holds the session data, or the session data needs to be replicated across multiple instances.

A Scalable Design Example

- Distributed cache
  - To make DB more scalable, we cache
    - Session data
    - Query data
  - Need to manage the status of cached data (dirty problem)
    - 比較適合
      - 少變、常被存取的資料



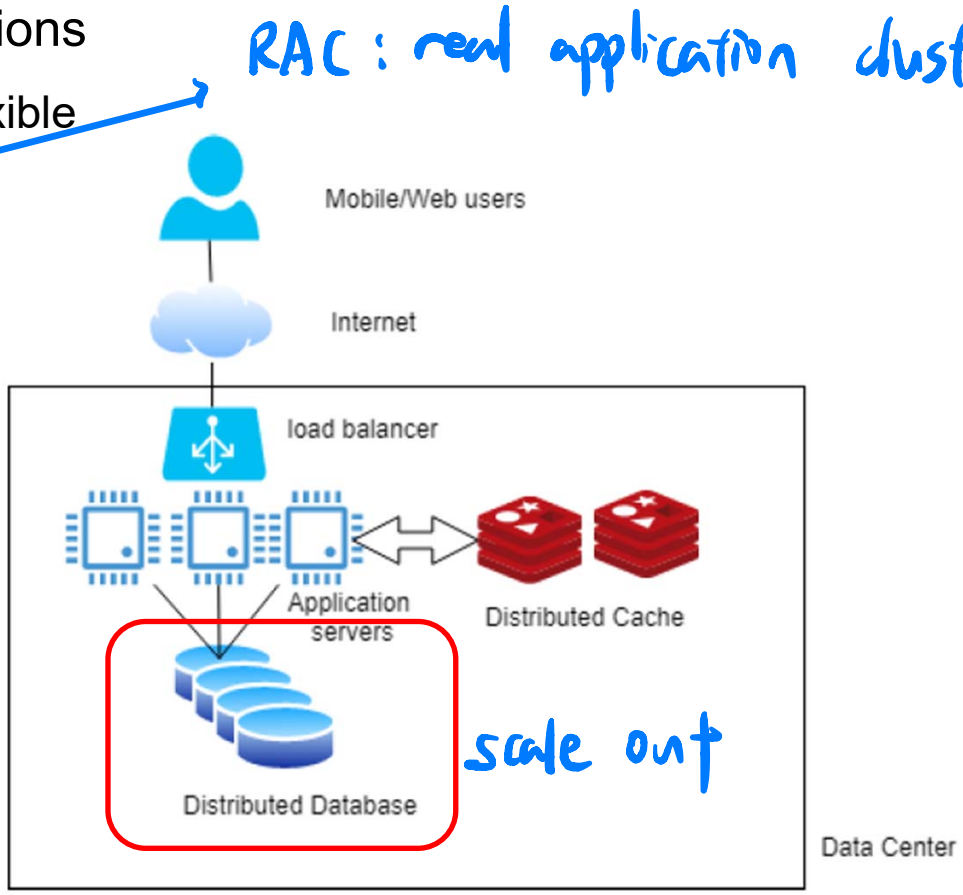
Example：選課系統



A Scalable Design Example

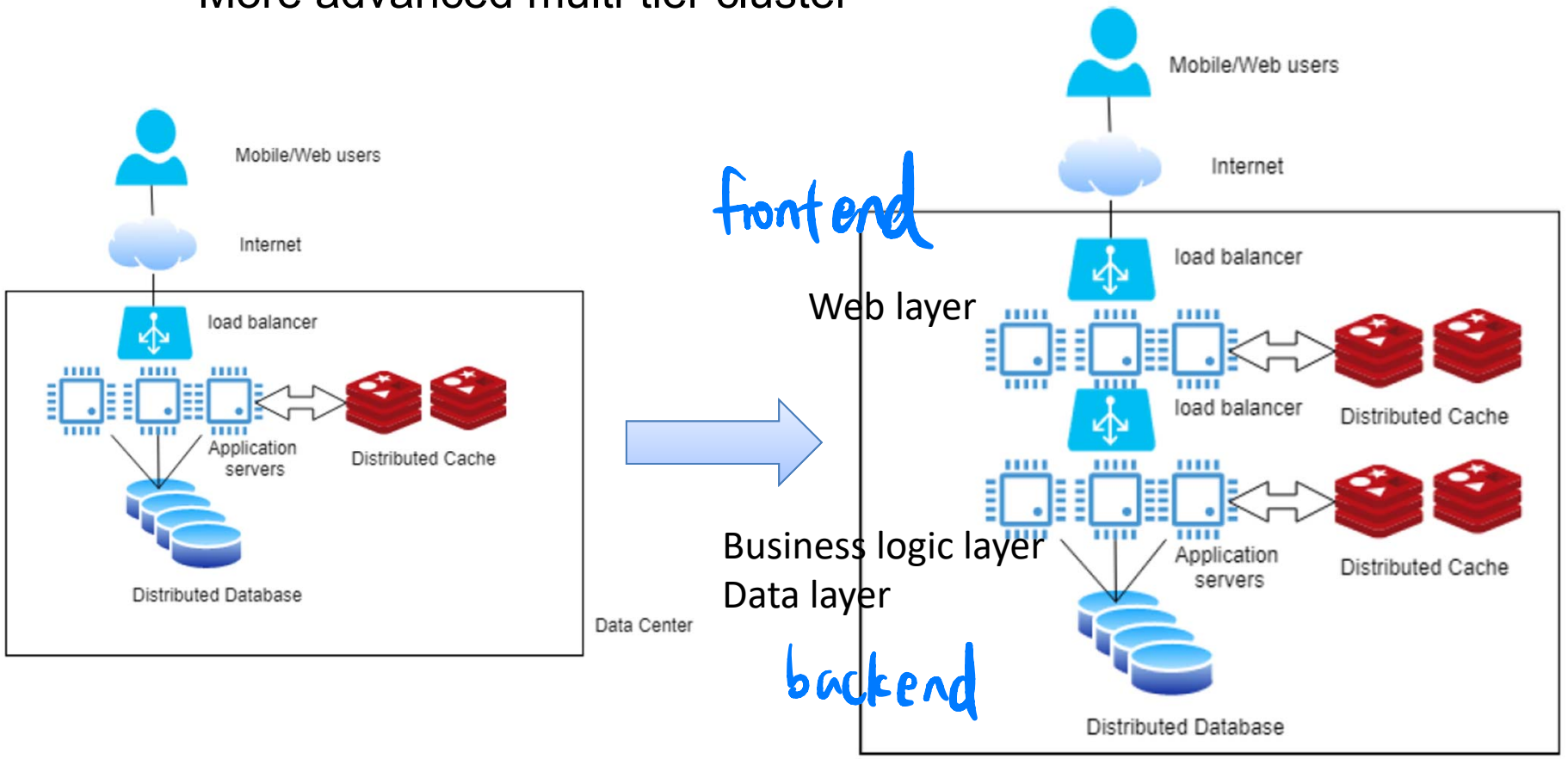
- Distributed database
  - Buy distributed database solutions
    - More expensive, but more flexible
    - Ex: Oracle RAC
  - 最主要的挑戰: 副本間的同步
- Alternatives
  - Use cloud hosted db
    - Amazon, Azure…
  - Use NoSQL/NewSQL
    - Cheaper, but limited
    - Ex: Mongo DB or TiDB

NoSQL NewSQL



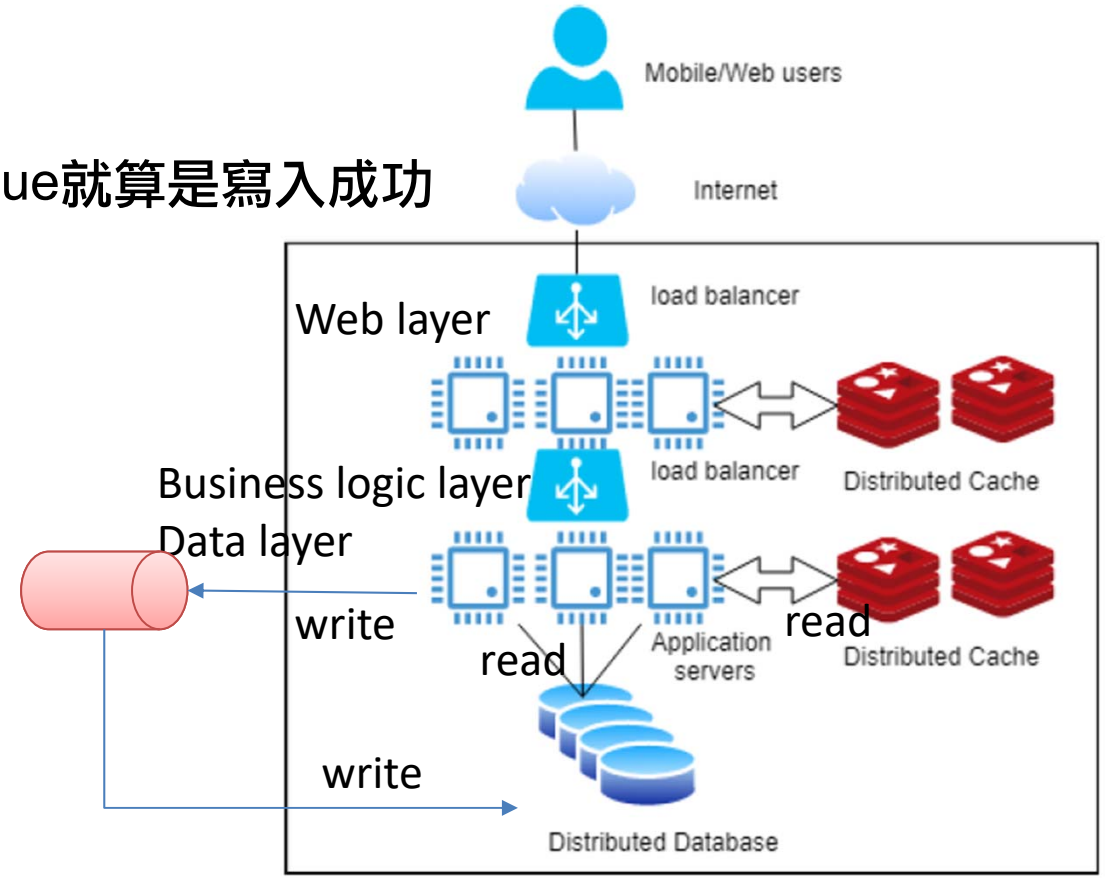
A Scalable Design Example

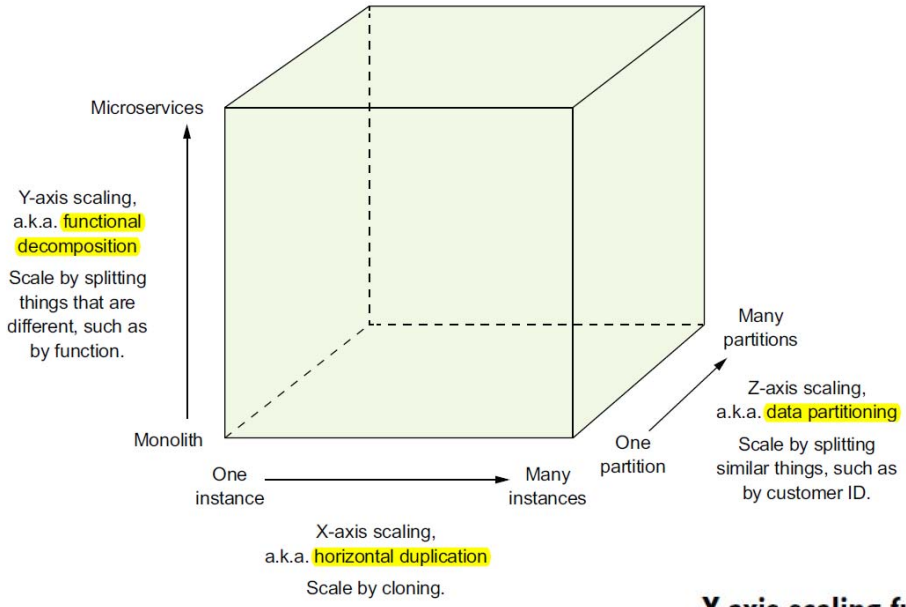
- Backend for frontend pattern (BFF)
  - More advanced multi-tier cluster



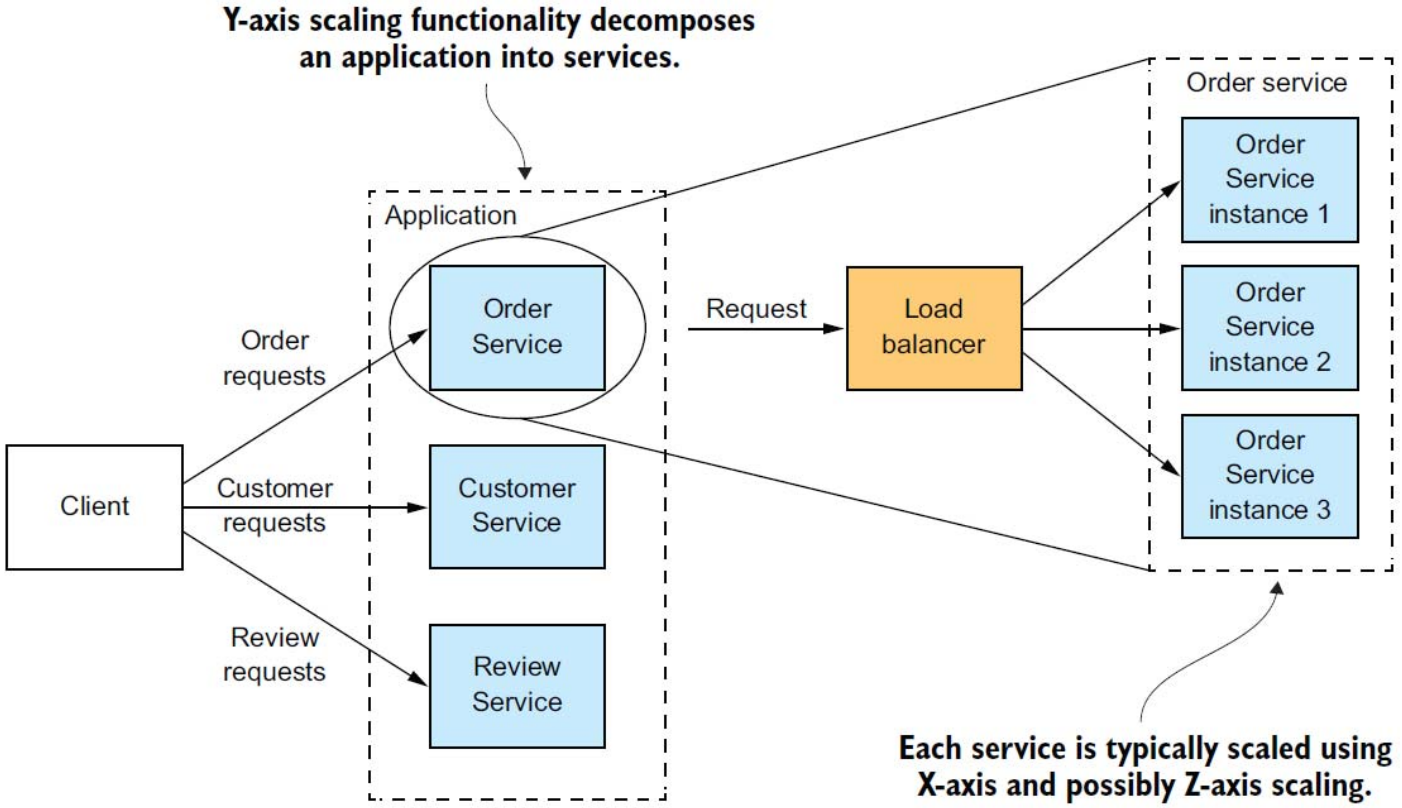
A Scalable Design Example

- 讀寫分離
  - 提升response time
  - 寫入動作只要丟到Queue就算是寫入成功
  - 非同步
    - 晚點才能確認結果

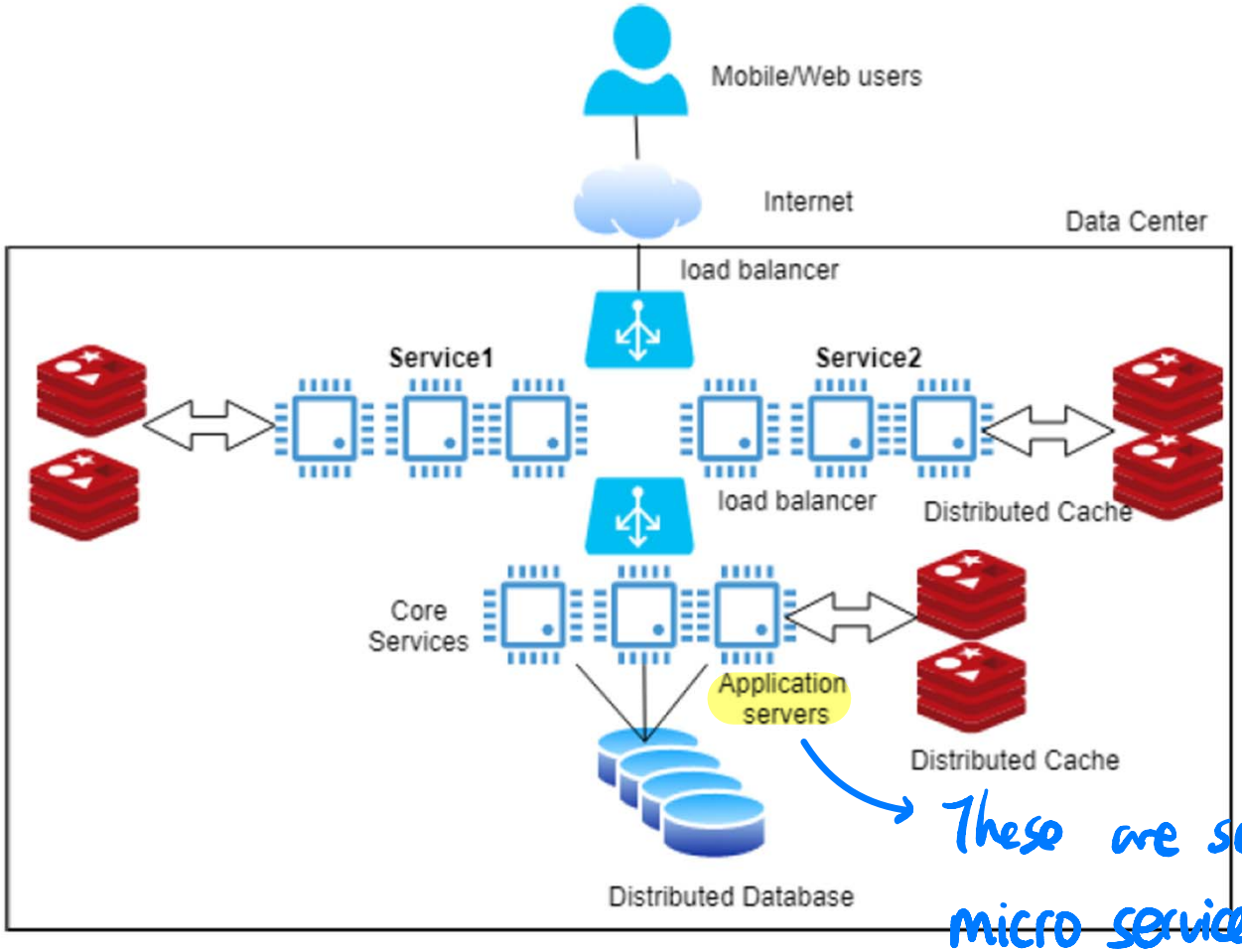




Microservice scaling
Layer 1: Functional Decomposition (Y)
Layer 2: Horizontal (X) + data (Z)



Microservices Architecture



These are servers where micro services run, they could be physical servers, virtual machines or containers

Foundations of Hyper Scale Architecture

- Virtualization
  - "Hardware as Software"
- Containerization (ex: Docker)
  - Packaging system as OCI-compatible images
- Container Orchestration (ex: K8S)
  - Platform that runs OCI-compatible images
- Massive configuration management
  - Infrastructure as Code (IaC)

Open Container Initiative

build, distribute & run containers

manages deployment, scaling & networking for containers

A key DevOps practice that involves managing & provisioning computing infrastructure through machine readable files, rather than physical H/W configuration or interactive configuration tools.

Q & A