



SOFTWARE ENGINEERING

# Software Architecture

**Dr. Rathachai Chawuthai**

Department of Computer Engineering  
Faculty of Engineering

King Mongkut's Institute of Technology Ladkrabang

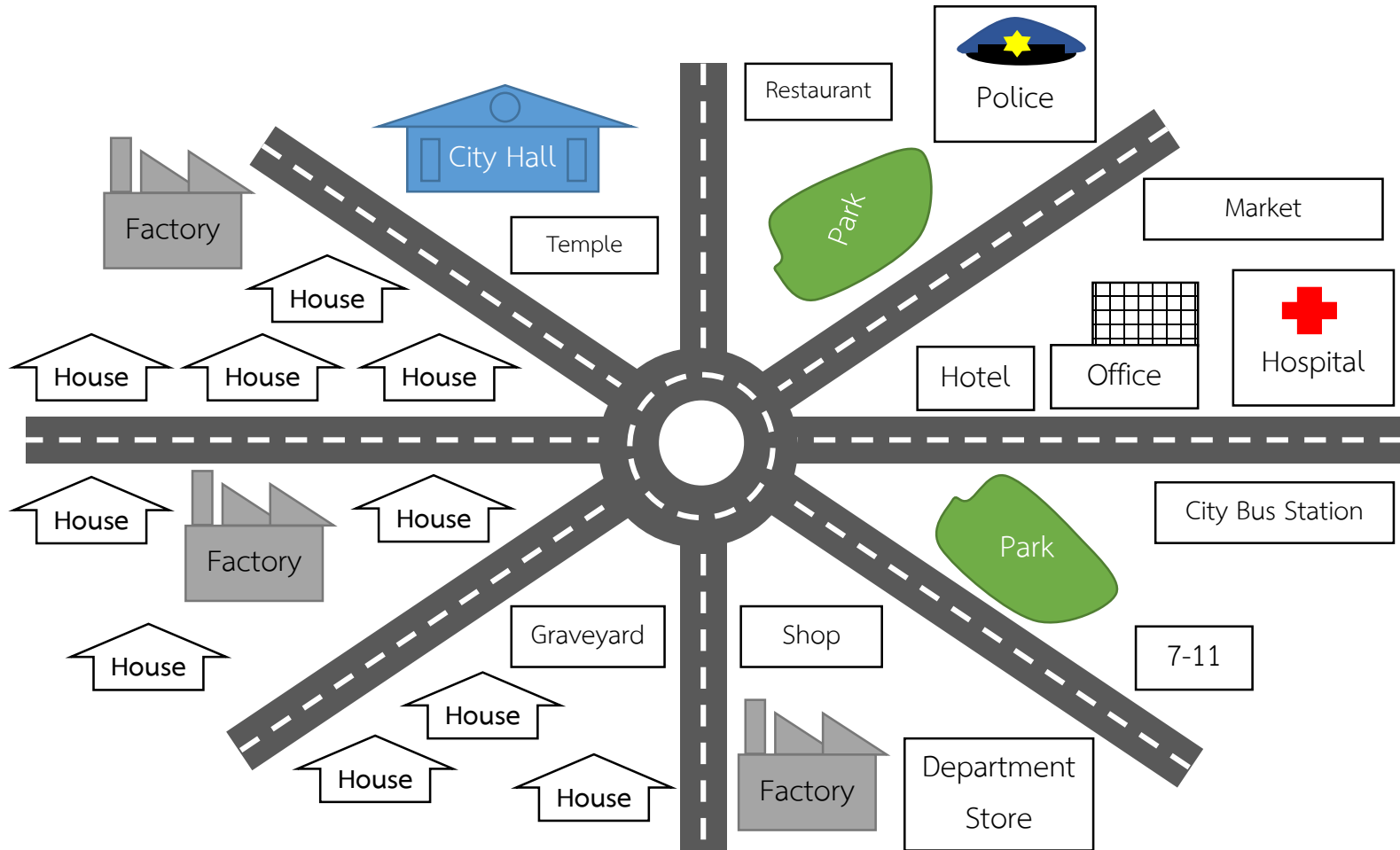
# Agenda

- Software Architecture
- Layered Architecture
- Client-Server Model
- 3-Tier Architecture
- Service-Oriented Architecture (SOA)
- Model-View-Controller (MVC)
- Microservice
- Distributed Systems

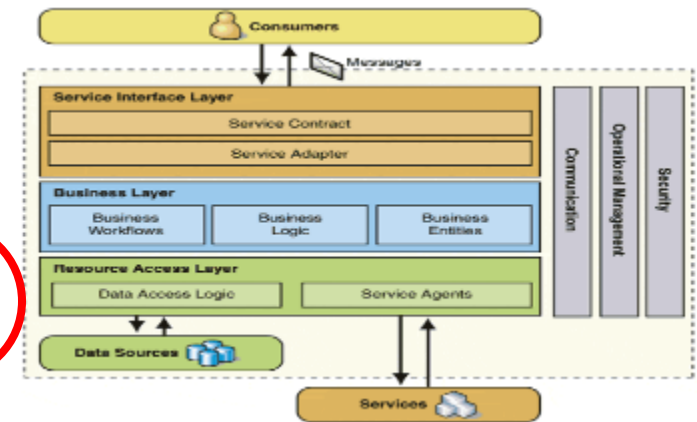
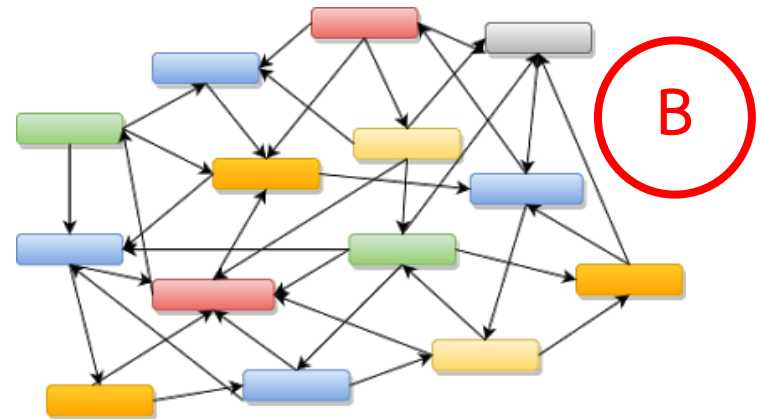
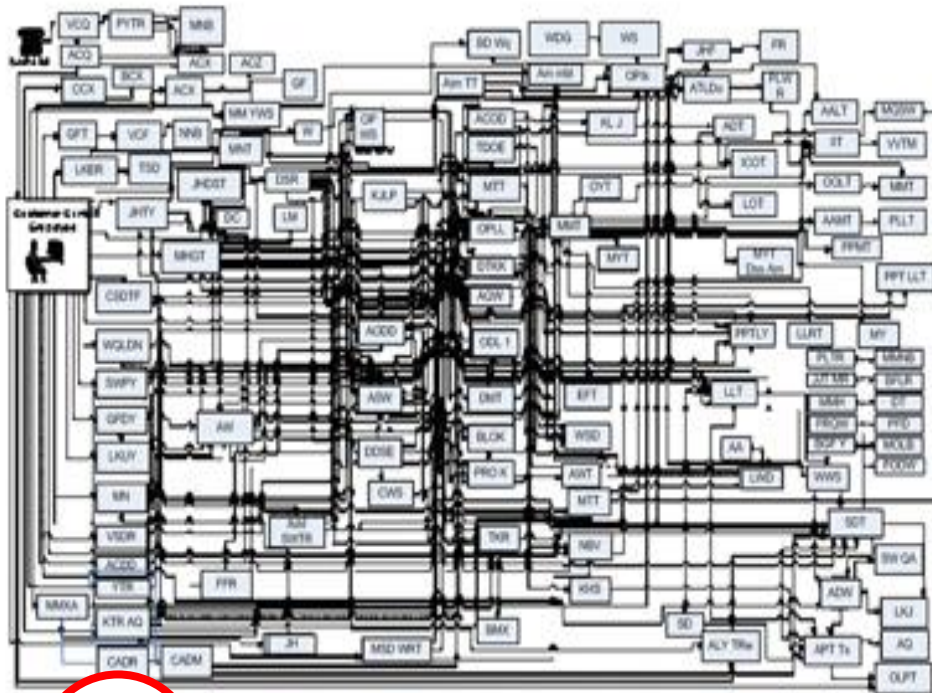
# Software Architecture



# Is it good?

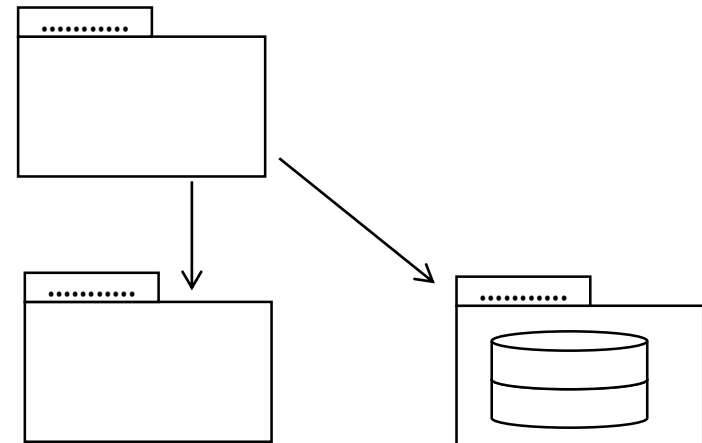
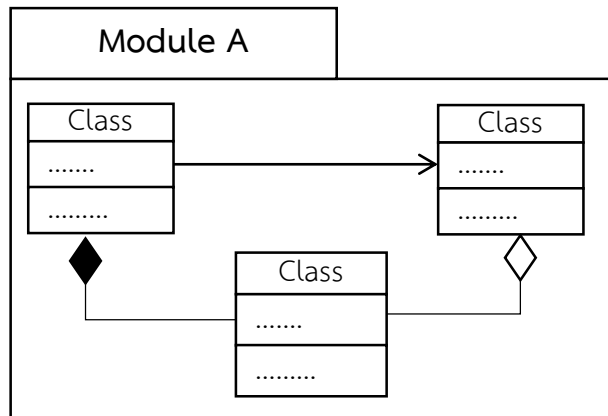


# Which one you like most?



# Software Design vs. Architecture

Software Design	Software Architecture
สนใจเพียงแค่การออกแบบ Module หรือ Component หนึ่งๆ	ออกแบบองค์รวมของระบบ และสนใจการประสานงานร่วมกันทั้งระบบ
ดูที่หน้าที่และ function ของ Module หนึ่งๆ หรือ Class หนึ่งๆ	มองระดับ High Level และสามารถบอกได้ว่าใช้หน่วยเก็บข้อมูลแบบใด



# Architectural Design

---

- สนใจการบริหารจัดการโครงสร้าง และ การจัดกลุ่มของ classes หรือ files ต่างๆ ในระบบซอฟต์แวร์
- การประสานงานระหว่างกลุ่มต่างๆ รวมถึง protocol และ โครงสร้างข้อมูล เพื่อใช้ในการสื่อสารระหว่างกัน
- คิดถึงเรื่อง Reuse เป็นสำคัญ
- เป็นภาพกว้างของกลุ่มต่างๆ ของระบบ
- เป็นโครงสร้างที่จะไม่แก้ไขบ่อย (ต้องทำให้ดีเป็นอันดับแรกของทุก Software Process)
- Software Architecture ที่ดีต้องเรียบง่าย สามารถนำไปสื่อสารให้ผู้อื่นเข้าใจได้ง่าย
- สามารถ maintain แต่ละส่วนได้ง่าย และ ประเมินความเสี่ยงผลกระทบได้
- มีหลากหลายรูปแบบ ให้เลือกใช้กับงานต่างๆ

# Architecture and System Characteristics

---

- **Performance**

- Localize critical operations and minimize communications. Use large rather than fine-grain components.

- **Security**

- Use a layered architecture with critical assets in the inner layers.

- **Safety**

- Localize safety-critical features in a small number of sub-systems.

- **Availability**

- Include redundant components and mechanisms for fault tolerance.

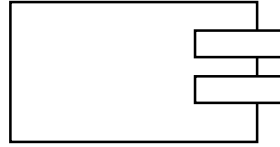
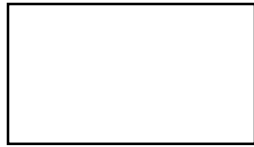
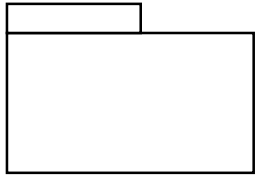
- **Maintainability**

- Use fine-grain, replaceable components.

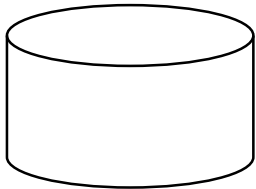


# Definitions

---



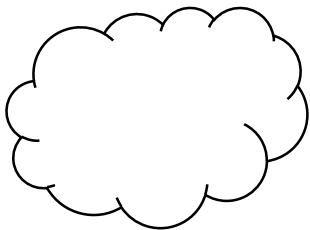
Module หรือ Component หรือ ระบบย่อย ที่  
บรรจุ Classes หรือ Files ต่างๆ ที่ใช้ทำงาน



ฐานข้อมูล (Database)



เส้นทางการ Request (ไม่ใช่ data flow) เพื่อร้องขอสิ่งต่างๆ



Internet หรือ Cloud

# Layered Architecture



# Layered Architecture

---

## Presentation

เป็น user interface ให้ผู้ใช้เรียกใช้ หรือเป็น service interface ให้ระบบอื่นเรียกใช้

## Business Logic

รวบรวม functions และ process ทั้งหมดของระบบ

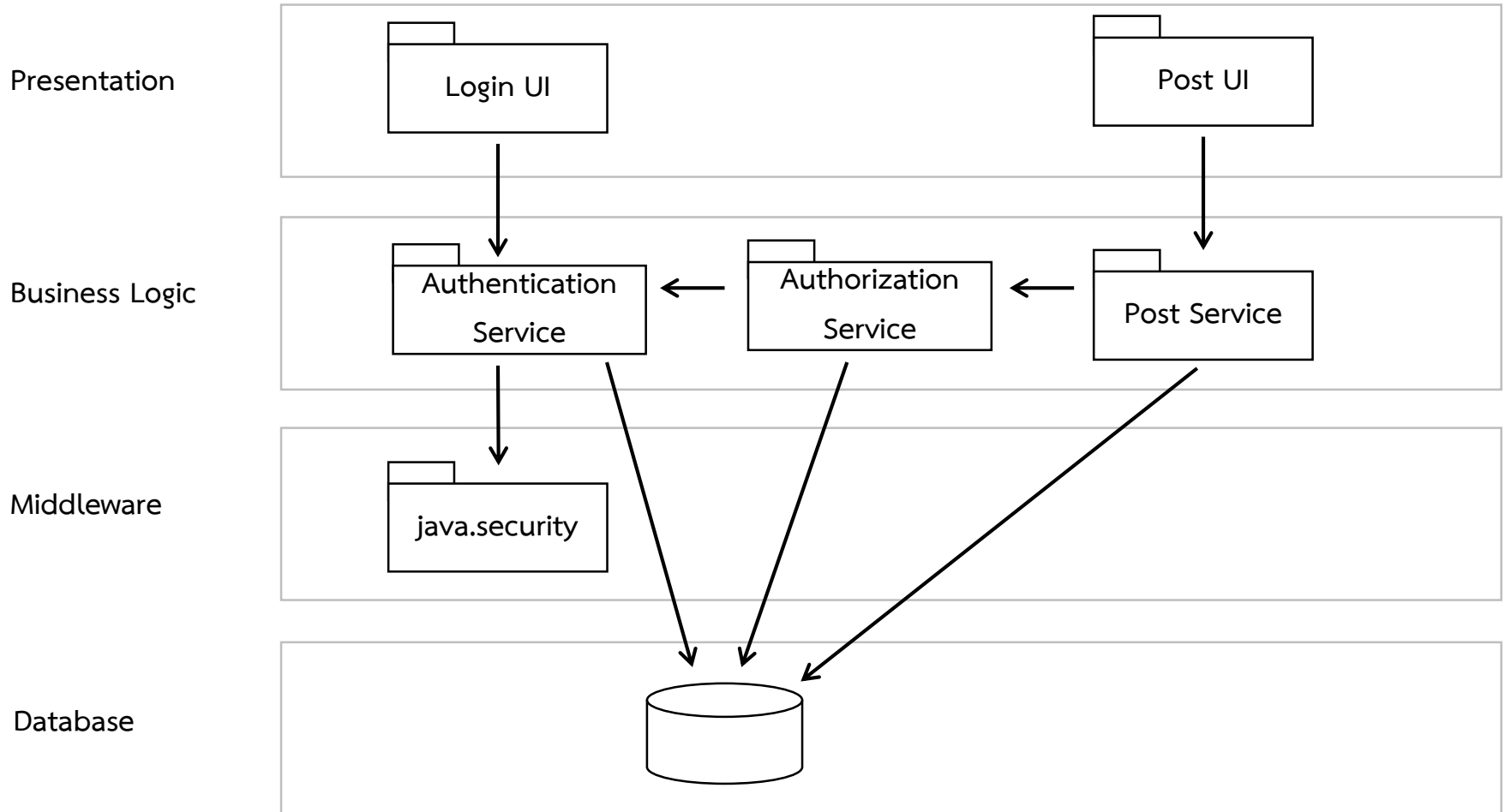
## Middleware

เป็น third-party ต่างๆ เช่น subsystem, service ภายนอก, API, libraries, algorithm ต่างๆ เป็นต้น

## Database

เก็บข้อมูล

# Layered Architecture



# Layered Architecture

---

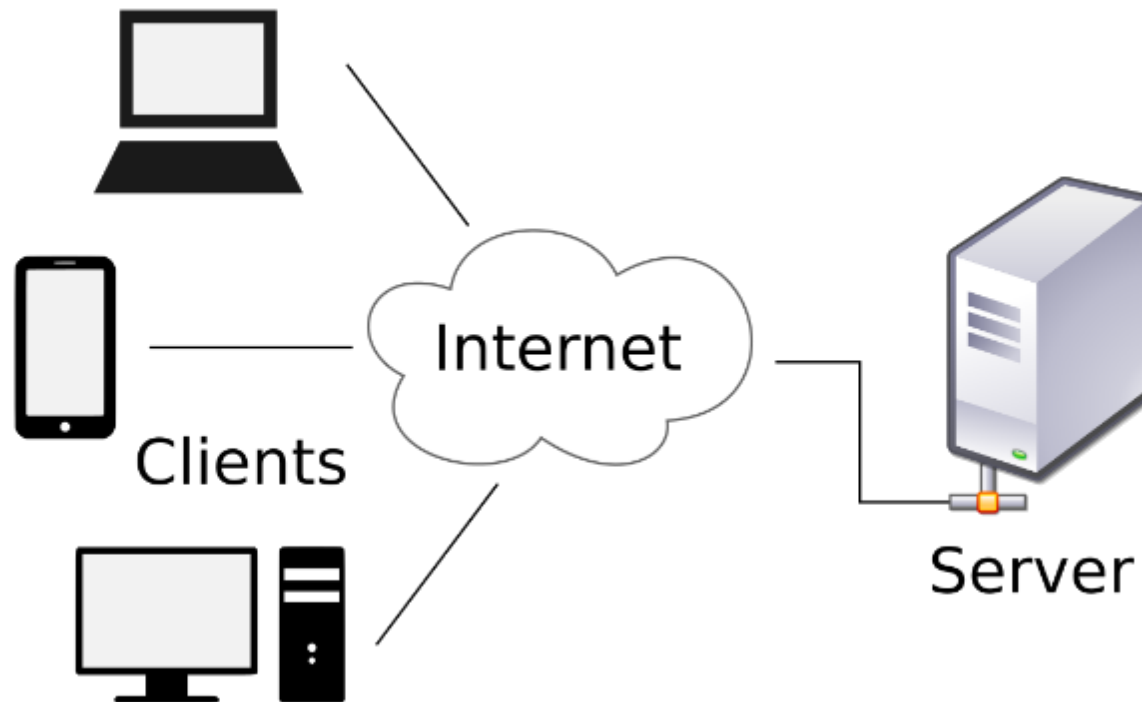
- รายละเอียด
  - แบ่ง Components ประเภทต่างๆ เป็น layer
  - มีการทำงานกันระหว่าง layer ที่ติดกันอย่างชัดเจน
- ข้อดี
  - มีการแบ่ง component และ dependency ที่ชัดเจน สามารถเพิ่มเติม component ได้สะดวก
- ข้อด้อย
  - ในทางปฏิบัติบางครั้งก็มีเหตุการณ์ที่ component จาก layer บนๆ เรียกใช้งานข้าม layer มายัง layer ล่างๆ

# Client-Server Model



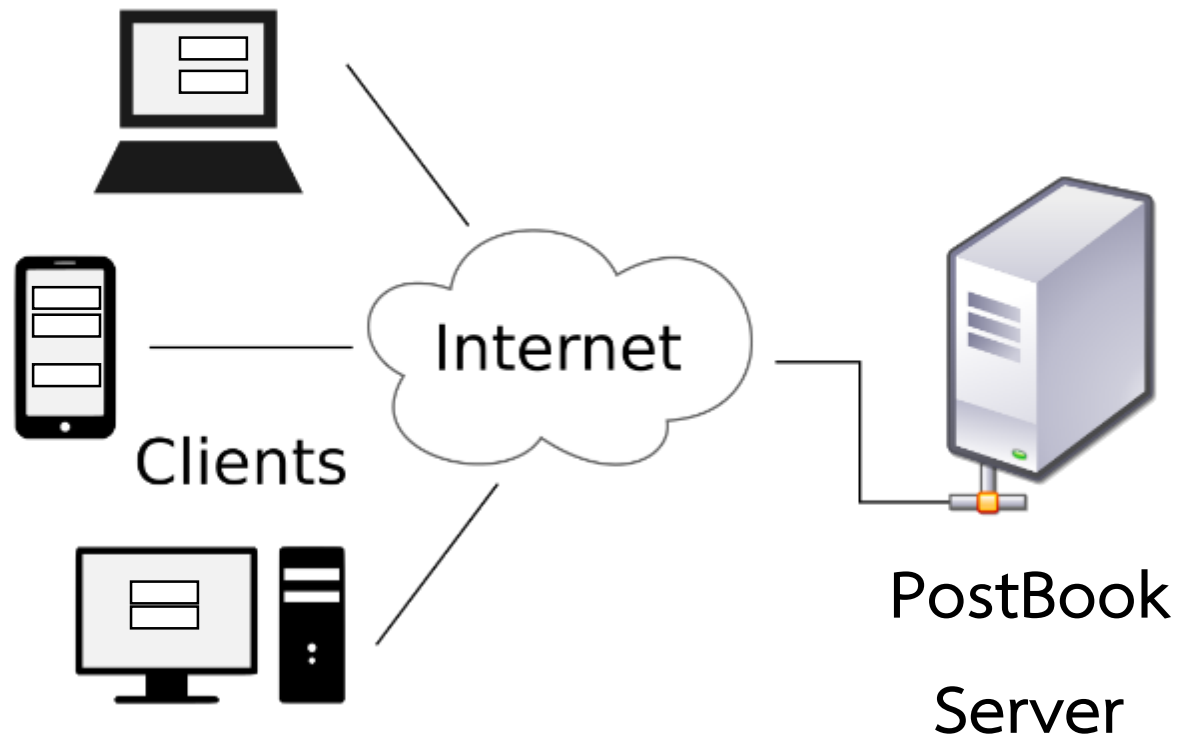
# Client-Server Model

---



# Client-Server Model

---





# Client-Server Model

---

- รายละเอียด

- มีการแบ่งระบบ Client กับ Server อย่างชัดเจน และเรียกใช้งานข้าม network เช่น Internet หรือ Intranet
- Client จะเป็น application ที่ผู้ใช้ใช้งาน
- Server จะมี Service กลางที่ไว้ให้บริการ Client

- ข้อดี

- Servers สามารถกระจายข้าม network ได้

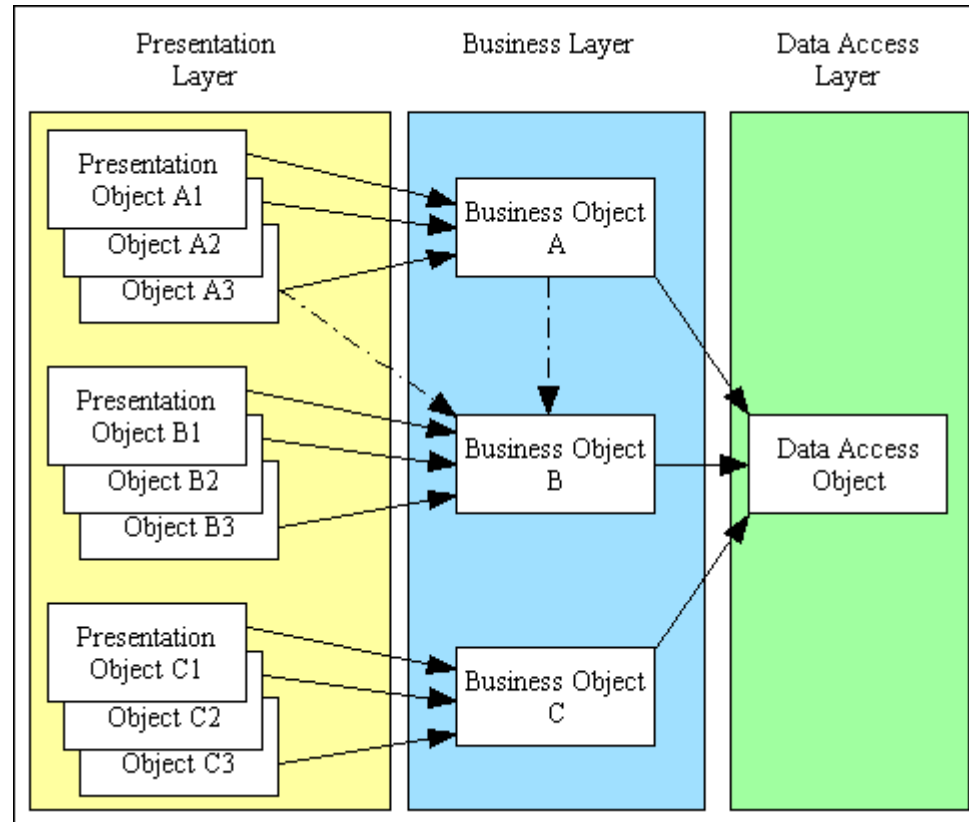
- ข้อด้อย

- มี server เป็น point of failure คือถ้า sever พัง client ก็ใช้งานไม่ได้

# 3-Tier Architecture

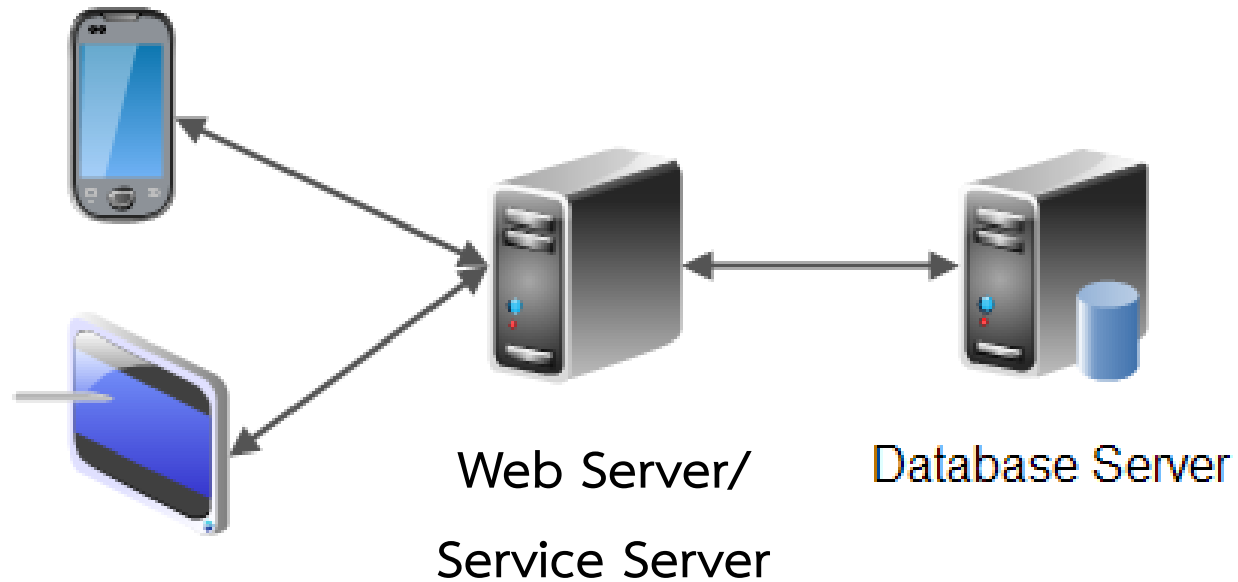


# 3-Tier Architecture

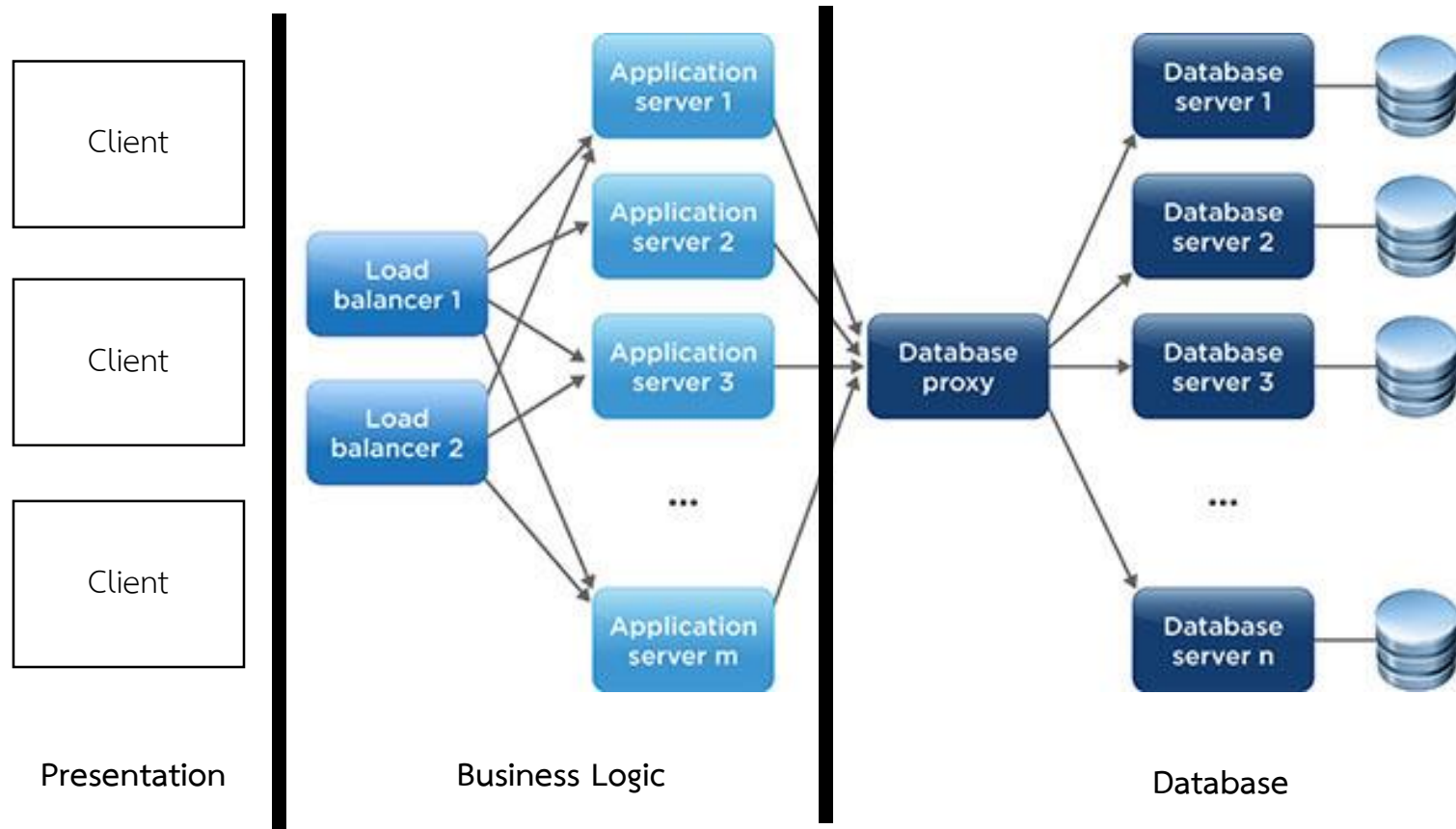


# 3-Tier Architecture

---



# Scaled 3-Tier Architecture



# Scaled 3-Tier Architecture

---

- รายละเอียด
  - มีการแบ่งระบบ Application, Business Logic และ Database อย่างชัดเจน
  - แต่ละส่วนสามารถอยู่คนละ servers และเรียกใช้งานข้าม network เช่น Internet หรือ Intranet ได้
- ข้อดี
  - เป็นรูปแบบที่ยืดหยุ่นและเป็นที่นิยมใช้สำหรับทำ Web Application
- ข้อด้อย
  - ถ้าออกแบบไม่ดีก็จะมี server เป็น point of failure โดยเฉพาะ Database Server

# Service-Oriented Architecture (SOA)



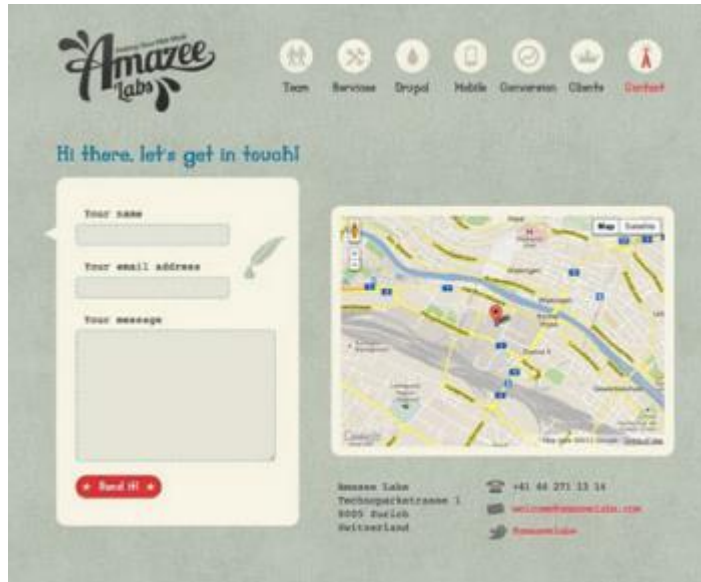
# SOA

---

- SOA เป็นกิจกรรมการให้บริการ (Service) จากฝ่ายผู้ให้บริการ (Service Provider) ไปยังอีกฝ่ายที่เป็นผู้รับบริการ (Consumer) ซึ่งทั้งสองฝ่ายอาจจะอยู่ในองค์กรเดียวกันหรือต่างองค์กรกันก็ได้ แต่ต้องมีการตกลงลักษณะการสื่อสารซึ่งกันและกัน
- ความสำคัญคือการให้ Service (ไม่ใช่ Application)
- ที่ต้องทำแบบนี้สาเหตุหนึ่งคือบาง functions นั้น ผู้พัฒนา application ไม่สามารถทำเองได้ หรือไม่คุ้มที่จะทำ เช่น การแสดงแผนที่ (เลือกใช้ service จาก Google Map)
- ดังนั้น Service Provider จะต้องคิดตั้งต้นว่า จะให้ผู้ใช้ได้ใช้ service เพื่อทำอะไร (ไม่ใช่ออกแบบหน้าจอให้ และต้องไม่ยึดติดว่าผู้ใช้จะใช้ Programming Language หรือ Platform อะไร)
- ส่วน Consumer จะนำ service เหล่านั้นไปใช้ในการพัฒนา Application เอง



# SOA : Google Map Service



- การพัฒนา Map ด้วยทีมพัฒนาตนเองก็เป็นเรื่องที่ไม่เกินความสามารถ แต่ต้องใช้ effort สูงมาก อาจจะไม่คุ้มที่จะทำ
- ซึ่งถ้าเรียกใช้จาก Google Map ก็เพียงแค่เรียกใช้ code ตามด้านล่าง

```
<div id="map"></div>
<script>
    function initMap() {
        var map = new google.maps.Map(document.getElementById('map'), {
            center: {lat: -34.397, lng: 150.644}, zoom: 8 });
    }
</script>
<script src=https://maps.googleapis.com/maps/api/js? callback=initMap async defer></script>
```

# Service Interface Design

---

- **Design**

- Involves thinking about the operations associated with the service and the messages exchanged
- The number of messages exchanged to complete a service request should normally be minimized.
- Service state information may have to be included in messages

- **Interface Design Stages**

- **Logical interface design**

- Starts with the service requirements and defines the operation names and parameters associated with the service. Exceptions should also be defined

- **Message design (SOAP)**

- For SOAP-based services, design the structure and organization of the input and output messages. Notations such as the UML are a more abstract representation than XML
- The logical specification is converted to a WSDL description

- **Interface design (REST)**

- Design how the required operations map onto REST operations and what resources are required.

# SOA : Google Map Service

---

- Operation      **Distance Matrix**

- Input      

```
{  
  origins: [{lat: 55.93, lng: -3.118}, 'Greenwich, England'],  
  destinations: ['Stockholm, Sweden', {lat: 50.087, lng: 14.421}],  
  travelMode: 'DRIVING',  
  drivingOptions: {  
    departureTime: new Date(Date.now() + N),  
    trafficModel: 'optimistic'  
  }  
}
```

- Output

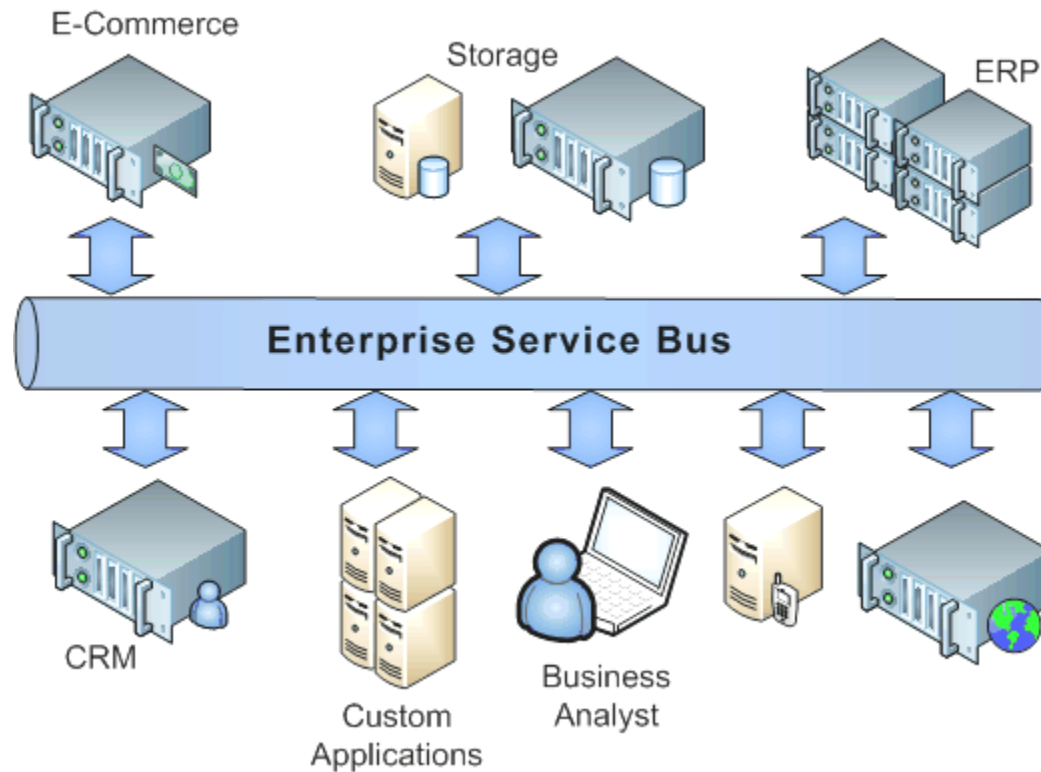
- Duration (e.g. 2 hours 20 mins)
- Distance (e.g. 100 km)

- Exception

- Place not found

# Enterprise Service Bus

- Service ที่มีการสื่อสารตรงกลางคือ ESB



# SOA

---

- รายละเอียด

- แบ่งซอฟต์แวร์เป็น component ที่เป็น service
- service provider เป็นผู้ให้บริการ ให้ consumer เรียกใช้
- มี protocol สำหรับสื่อสารกันระหว่าง service และมีการสื่อสารผ่าน ESB

- ข้อดี

- การใช้งานเป็นอิสระ ไม่ขึ้นอยู่กับ Application หรือ Platform
- การออกแบบเข้าใจง่ายและแบ่งงานให้ทีมพัฒนาได้ง่าย

- ข้อด้อย

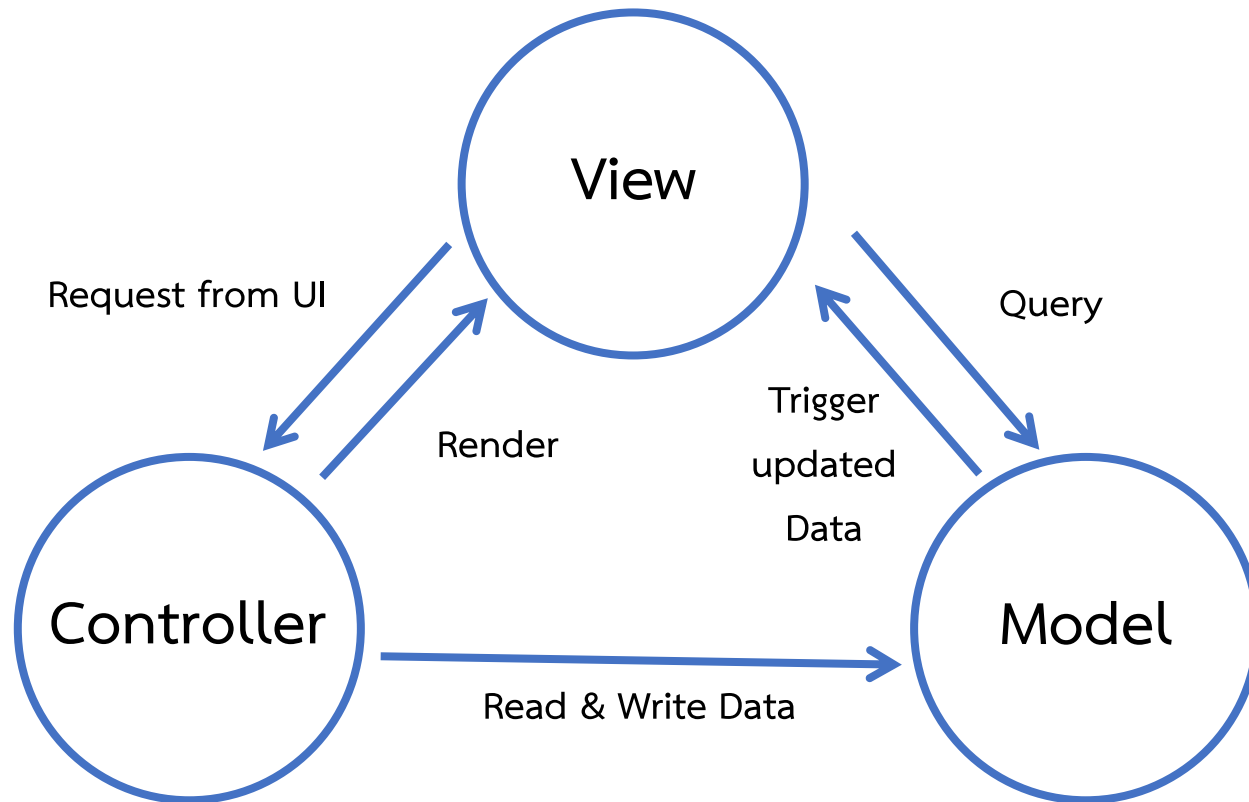
- แต่ละ service ไม่ได้ถูกพัฒนาให้เป็นอิสระต่อกัน ยังมีส่วนที่ใช้ร่วมกันเช่น database อยู่
- ESB อาจจะกลายเป็น Single Point of Failure

# Model-View-Controller (MVC)



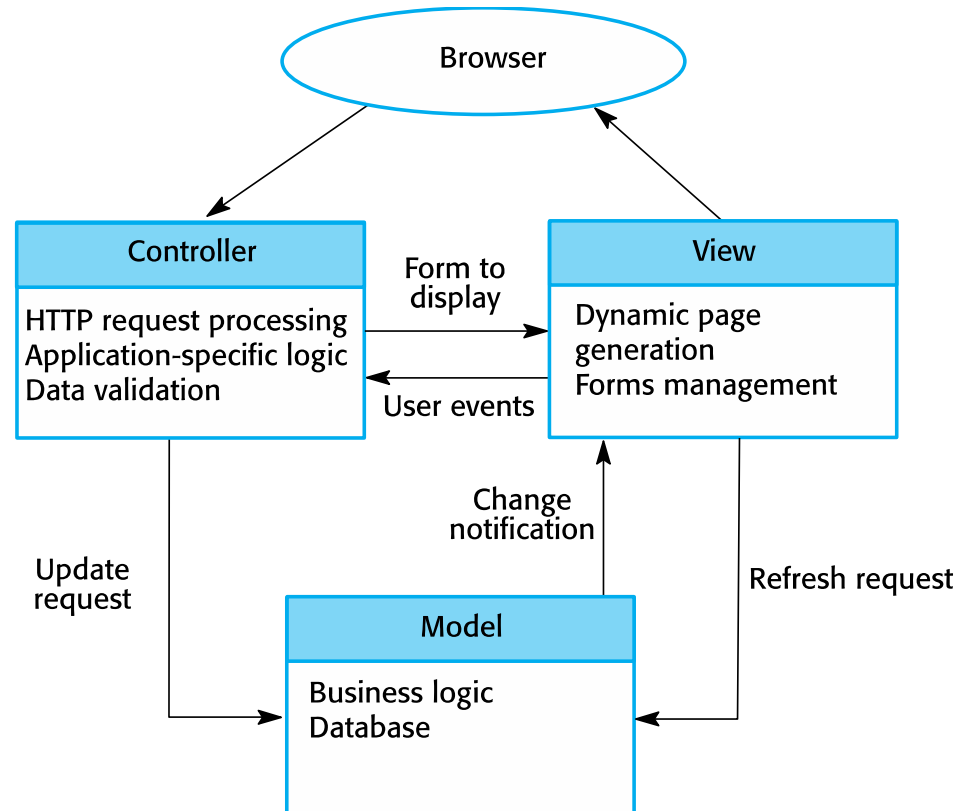
# MVC

---



# Web Application Architecture using MVC

---





# MVC

---

- รายละเอียด
  - แยกส่วน Presentation และ Interaction ออกจาก Data
  - มีใช้ใน Django, ASP.NET MVC, Rails, AngularJS, EmberJS, Backbone เป็นต้น
- ข้อดี
  - สามารถเปลี่ยน UI (view) ได้ โดยใช้ controller และ model ของเดิม
- ข้อด้อย
  - การเขียนโปรแกรมซับซ้อนขึ้น เพราะต้องแยก View ออกจาก Controller อย่างเด็ดขาด

# Microservice

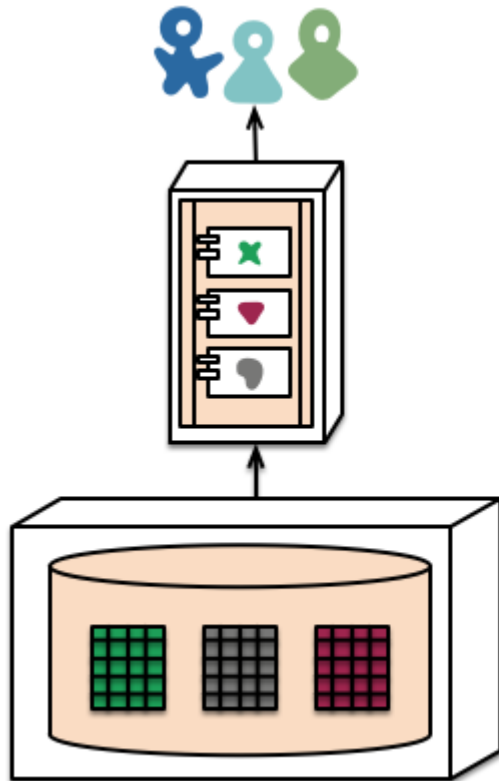


# To be Microservice

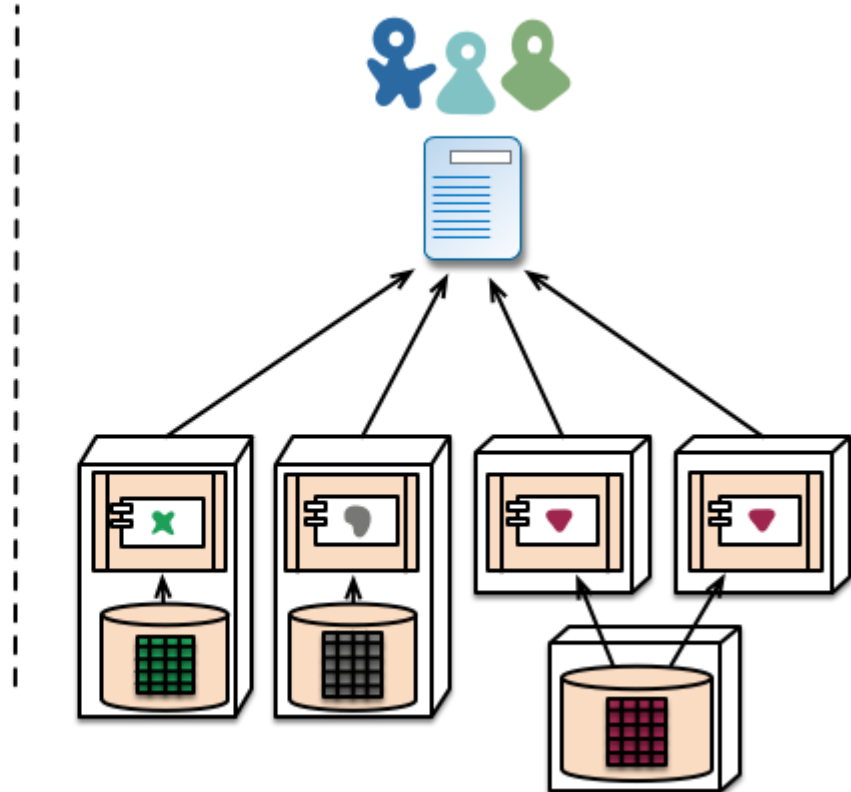
---



# Monolith vs. Microservice

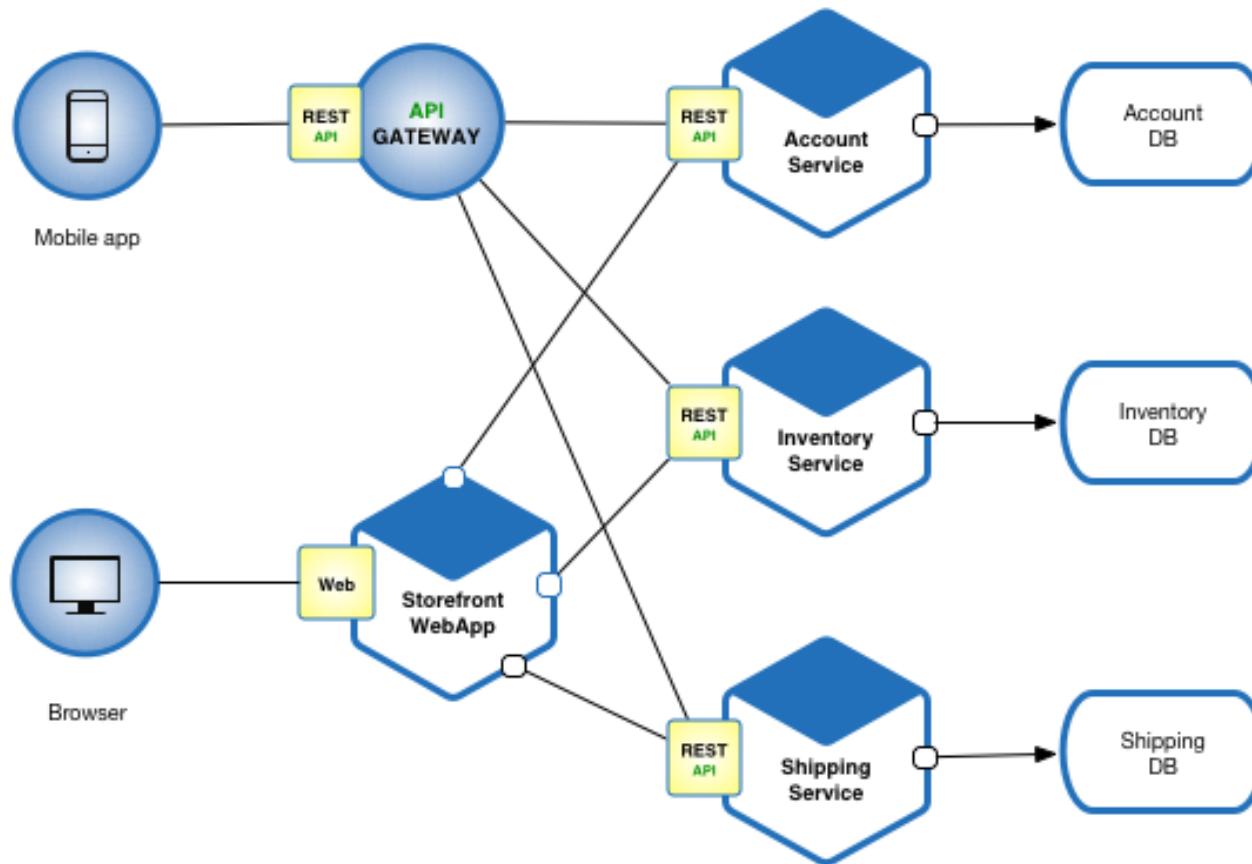


monolith - single database

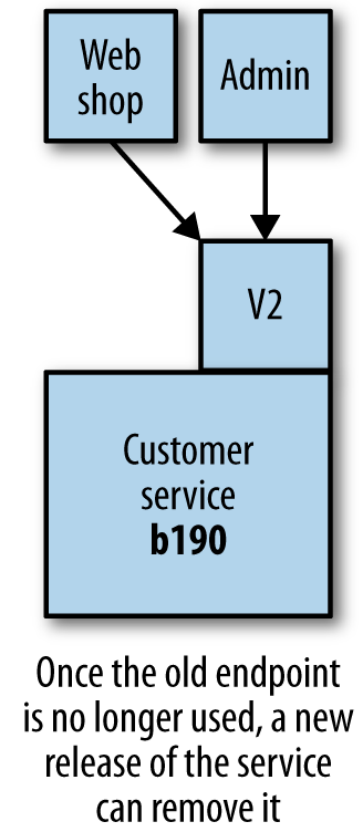
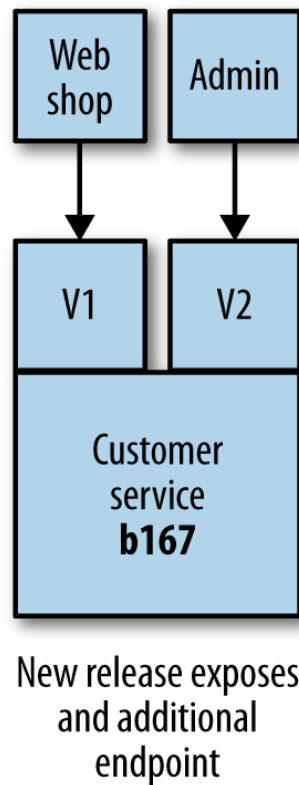
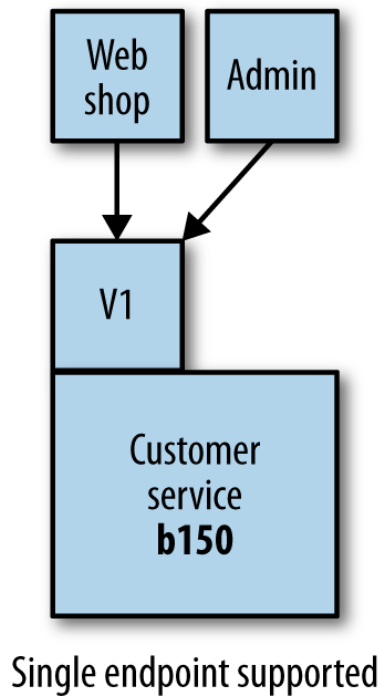


microservices - application databases

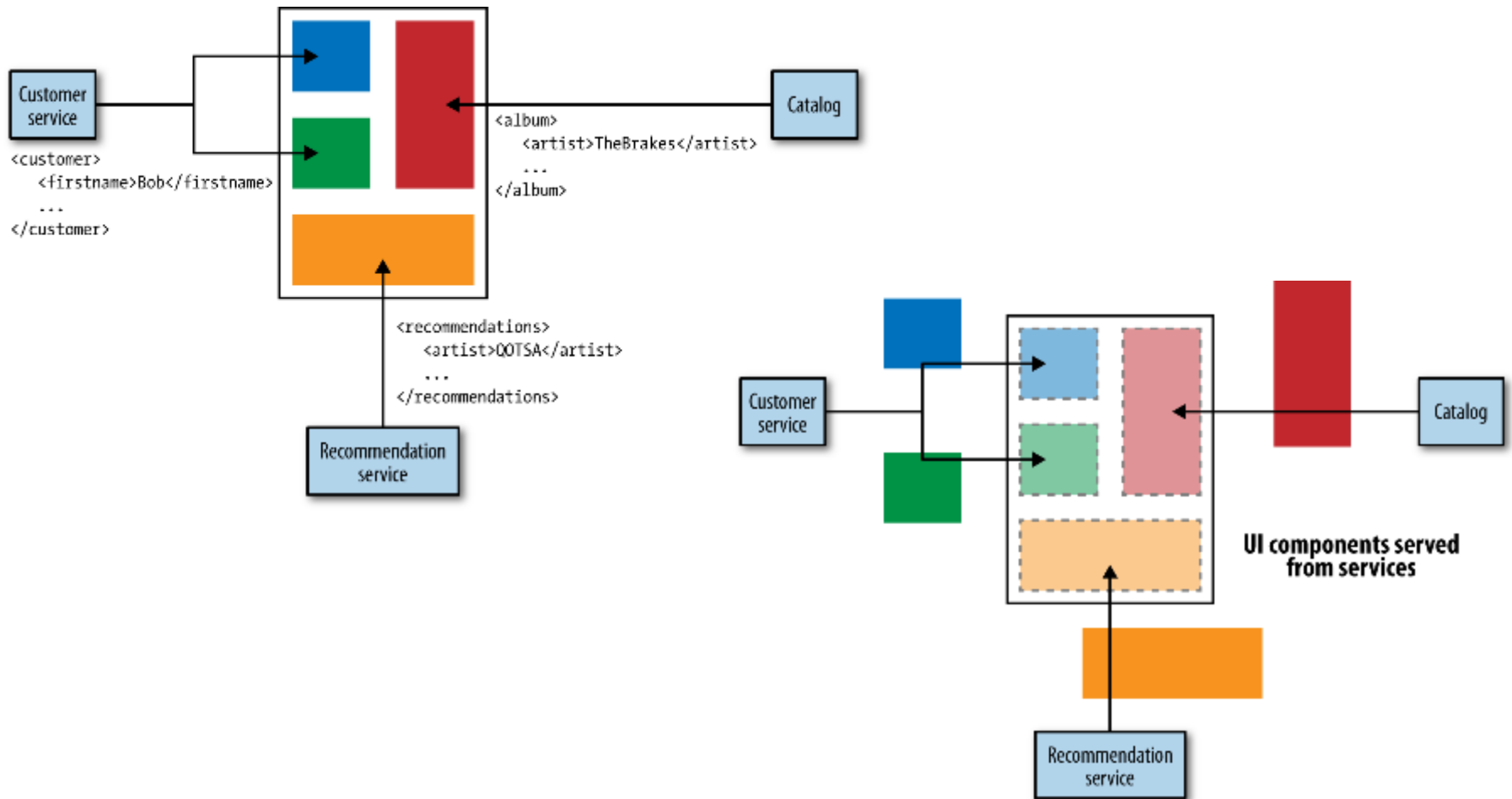
# Microservice



# Services from Versions



# Microservice and UI



# Microservice

---

- รายละเอียด
  - แบ่งระบบเป็น service ย่อยๆ โดยที่แต่ละ service จะติดต่อ data storage ที่เหมาะสมกับงาน
- ข้อดี
  - ซอฟต์แวร์มีขนาดเล็ก
  - สามารถพัฒนาคนละ programming language ได้
  - แต่ละ service ใช้ data storage คนละประเภทตามความเหมาะสมได้
  - สามารถขยายหรือปรับสัดส่วนการใช้ทรัพยากรของแต่ละ service ได้
- ข้อด้อย
  - การบริหารจัดการความถูกต้องของข้อมูลใน database ยากขึ้น
  - ต้องจัดการเรื่อง service version ให้รอบคอบ

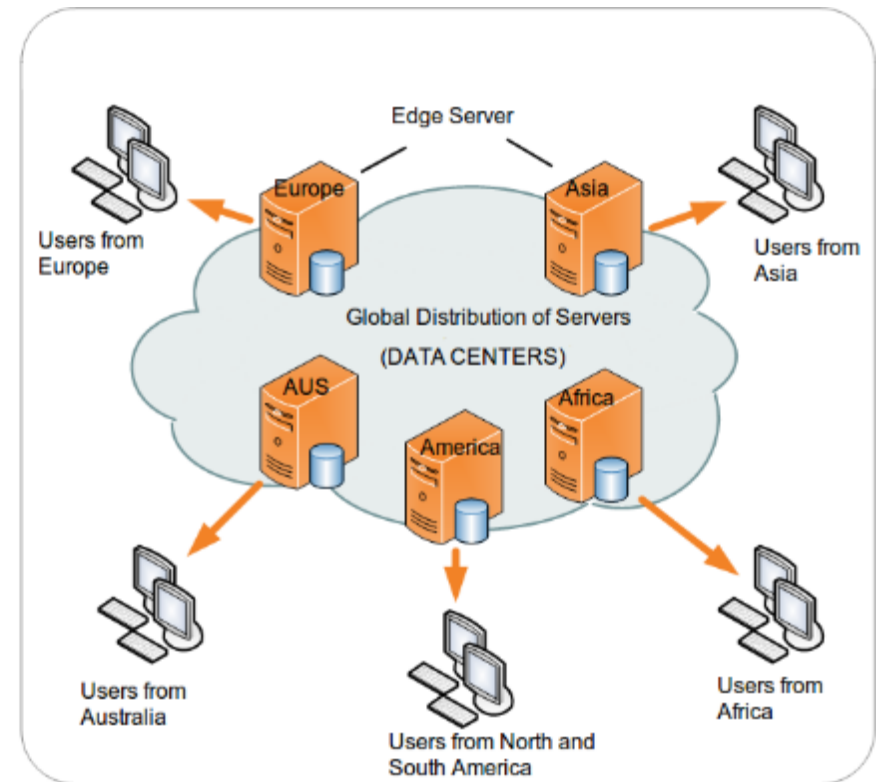
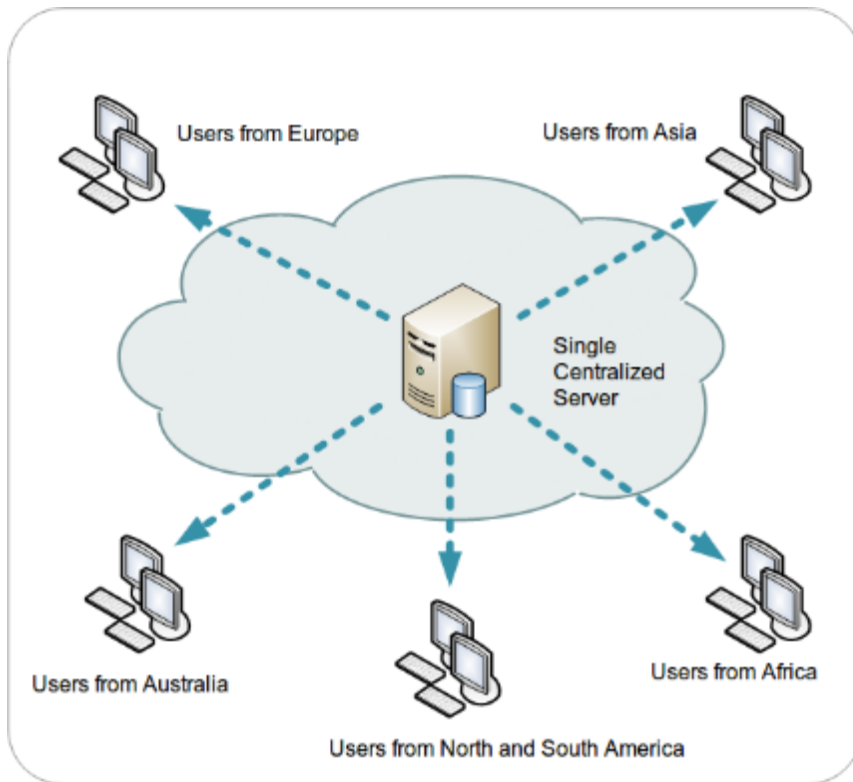


# Distributed Systems



# Centered Server vs. Distributed Servers

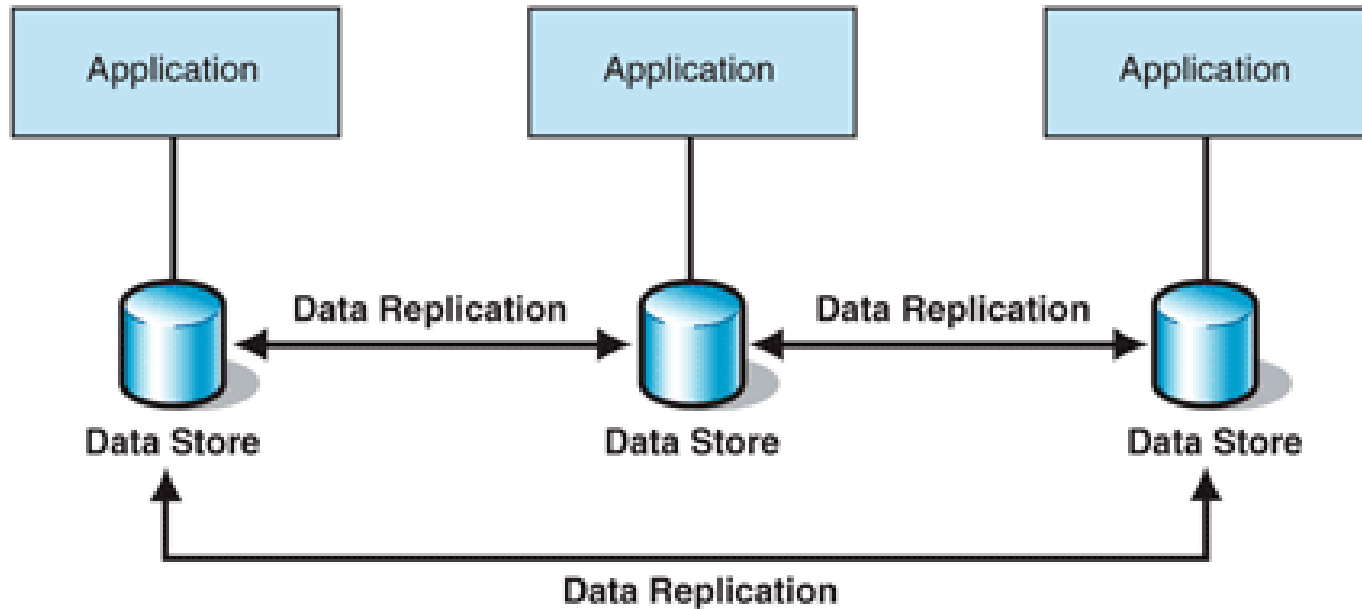
- ทำไม video hosting services อย่าง YouTube ถึงได้ตอบสนองรวดเร็วทั้งที่ใช้ bandwidth มหาศาล



# Data Replication

---

- การ update ข้อมูลให้เหมือนกันทุกๆ site



# Distributed Systems

---

- รายละเอียด

- มีการแบ่งระบบเป็น site โดยที่จะมี
- ในแต่ละ site จะมี database server เป็นของตนเอง และมีการ update ระหว่างกัน
- ข้อมูลในแต่ละ site อาจจะไม่จำเป็นต้องเหมือนกันก็ได้

- ข้อดี

- สามารถรองรับผู้ใช้ตามภูมิภาคได้อย่างดี
- หาก site หนึ่งมีปัญหา ก็ยังสามารถปรับให้ผู้ใช้ไปใช้ site อื่นได้

- ข้อด้อย

- การทำ data replication ตลอดเวลาเป็นเรื่องที่ดีเพราะผู้ใช้จะได้รับ data ที่ทันสมัยและถูกต้อง แต่จะทำให้เกิด traffic และสร้างปัญหาทาง network ได้ ดังนั้นจึงต้องกำหนด policy การทำ replication ให้รอบคอบ

# Summary



# Summary

---

- Software Architecture
- Layered Architecture
- Client-Server Model
- 3-Tier Architecture
- Service-Oriented Architecture (SOA)
- Model-View-Controller (MVC)
- Microservice
- Distributed Systems

“

Any work of architecture that has with  
it some discussion, some polemic,  
I think is good. It shows that people are  
interested, people are involved.

”

Richard Meier