ĐẠI HỌC CÔNG NGHIỆP TP.HCM
Khoa Công nghệ Thông tin

# CÁC KIỂU VÀ MẪU KIẾN TRÚC


**TS. Huỳnh Hữu Nghĩa**

luckerhuynhvn@gmail.com

# Nội dung:

- Tóm tắt các kiểu kiến trúc chính
- Kiểu kiến trúc Client/Server
- Kiểu kiến trúc Component-Based
- Kiểu kiến trúc Domain Driven Design
- Kiểu kiến trúc Layered
- Kiểu kiến trúc Message-bus
- Kiểu kiến trúc N-Tier / 3-Tier
- Kiểu kiến trúc Object-Oriented
- Kiểu kiến trúc Service-Oriented

# Một kiểu kiến trúc là gì?

- Một kiểu kiến trúc (mẫu kiến trúc) là một tập nguyên tắc (khuôn mẫu), nó cung cấp một khung làm việc trừu tượng cho hệ thống, định hình cho một ứng dụng.

- Một kiểu kiến trúc cải thiện việc phân chia và thúc đẩy việc sử dụng lại thiết kế bằng cách cung cấp những giải pháp cho các vấn đề thường xuyên xảy ra.

# Một kiểu kiến trúc là gì?

- Sự hiểu biết về những kiểu kiến trúc cung cấp một số lợi ích. Lợi ích quan trọng nhất là chúng cung cấp một ngôn ngữ chung và cung cấp những cơ hội trò chuyện không liên quan đến công nghệ.

- Điều này tạo điều kiện ở mức độ cao của cuộc nói chuyện bao gồm các mẫu và các nguyên tắt, mà không đi vào những chi tiết cụ thể. *Ví dụ có thể bàn luận về kiến trúc client/server so với n-tier*.

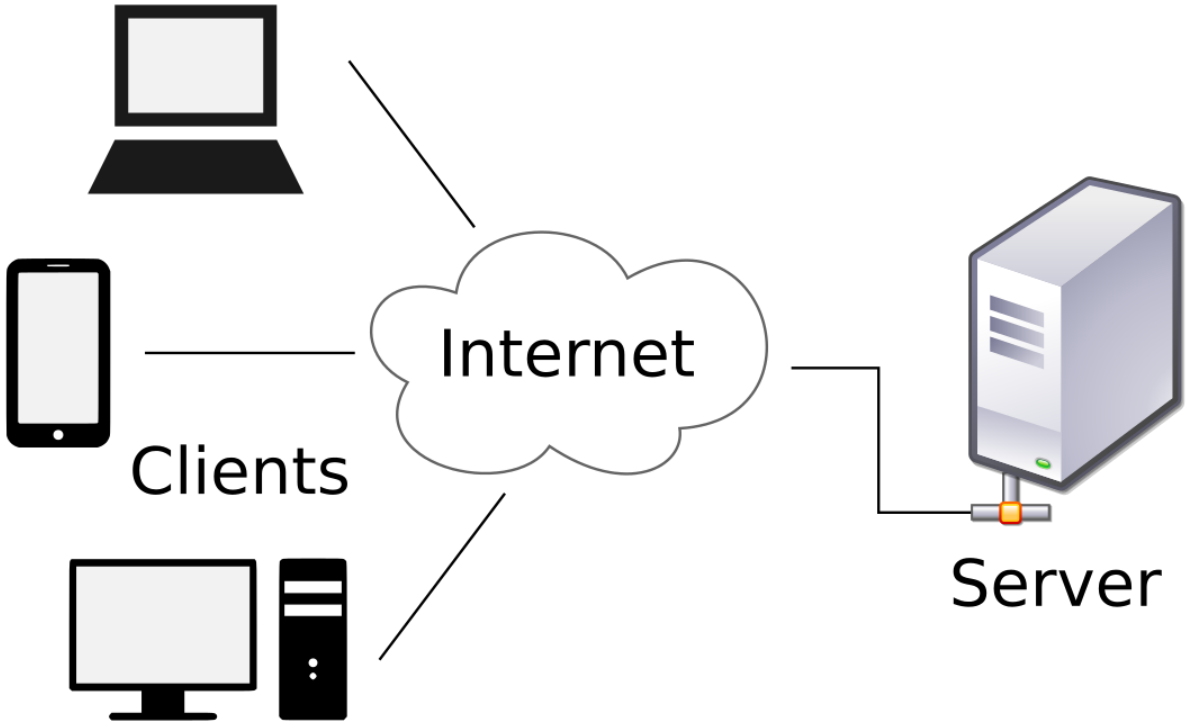
# Các kiểu kiến trúc kết hợp

- Kiến trúc của một hệ thống phần mềm hầu như không bao giờ giới hạn chỉ một kiểu kiến trúc duy nhất, thường thì kết hợp nhiều kiểu kiến trúc để tạo nên hệ thống hoàn chỉnh.

- Ví dụ, một thiết kế SOA bao gồm các dịch vụ được phát triển sử dụng hướng tiếp cận kiến trúc theo lớp và kiến trúc hướng đối tượng.

# Kiến trúc Client/Server

- Kiểu kiến trúc client/server mô tả các hệ thống phân tán liên quan đến hệ thống client và server riêng biệt, và kết nối thông qua mạng.

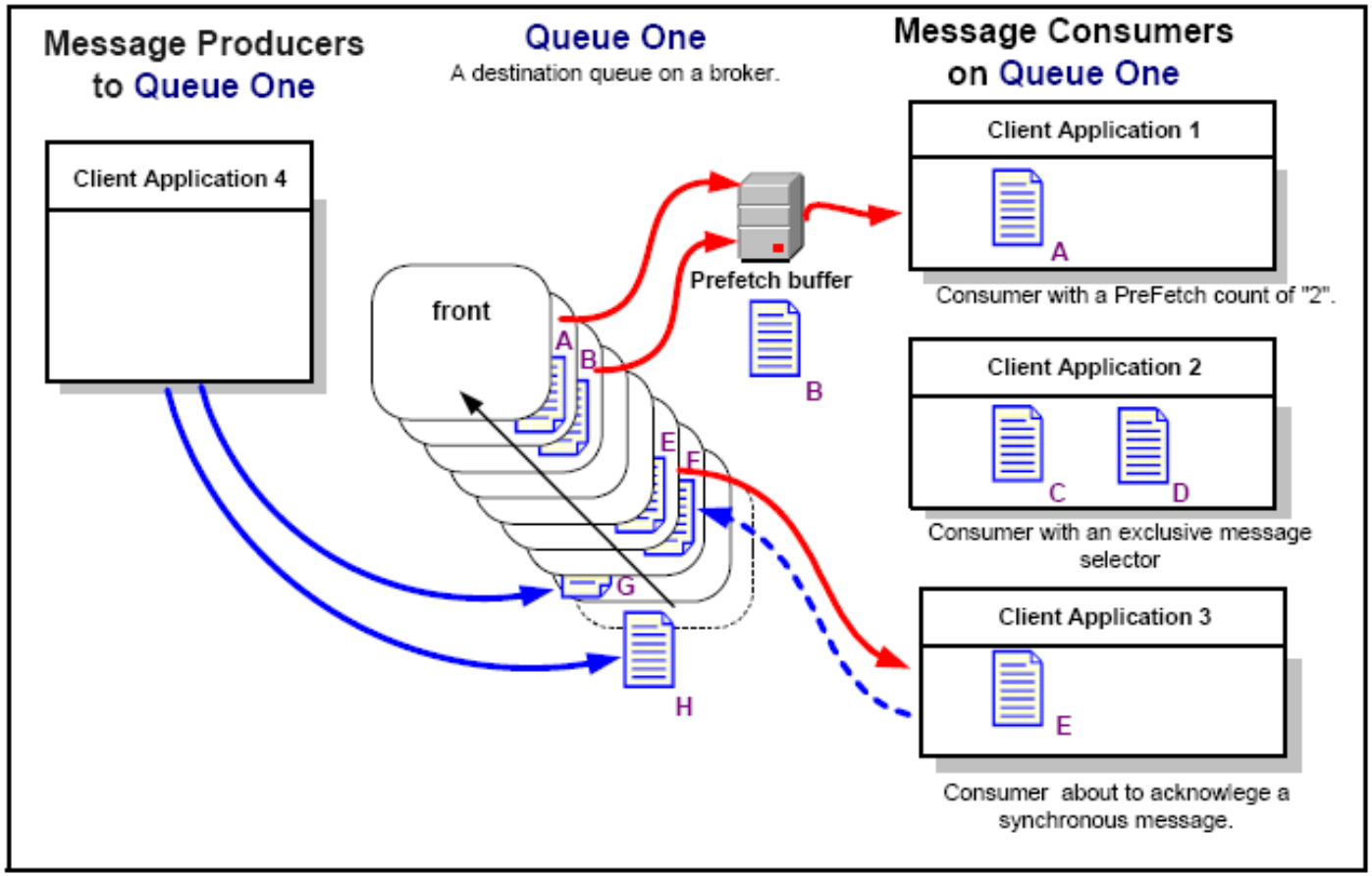


# Kiến trúc Client/Server

- Các ứng dụng dựa trên trình duyệt web chạy trên Internet hoặc Intranet.
- Các ứng dụng dựa trên hệ điều hành Microsoft truy cập các dịch vụ dữ liệu nối mạng.
- Các ứng dụng truy cập kho dữ liệu từ xa như: email, FTP clients, và các công cụ truy vấn dữ liệu.
- Các công cụ và tiện ích điều khiển các hệ thống từ xa như: các công cụ quản lý hệ thống và giám sát mạng.

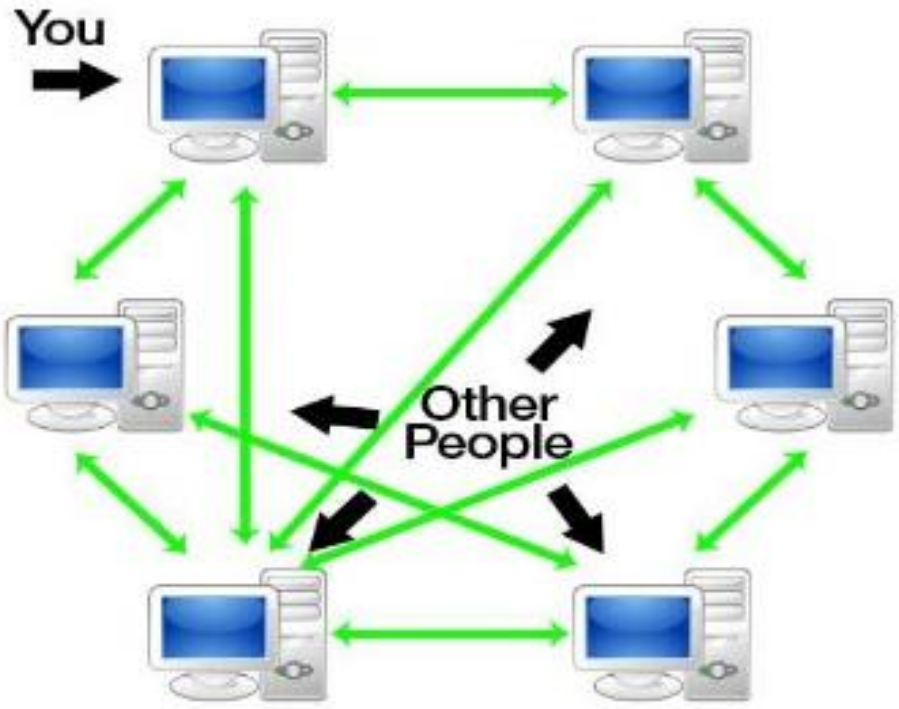
# Các biến thể của Client/Server

- Hệ thống **Client – Queue – Client** *(cho phép các client phân tán và đồng bộ file và thông tin).*



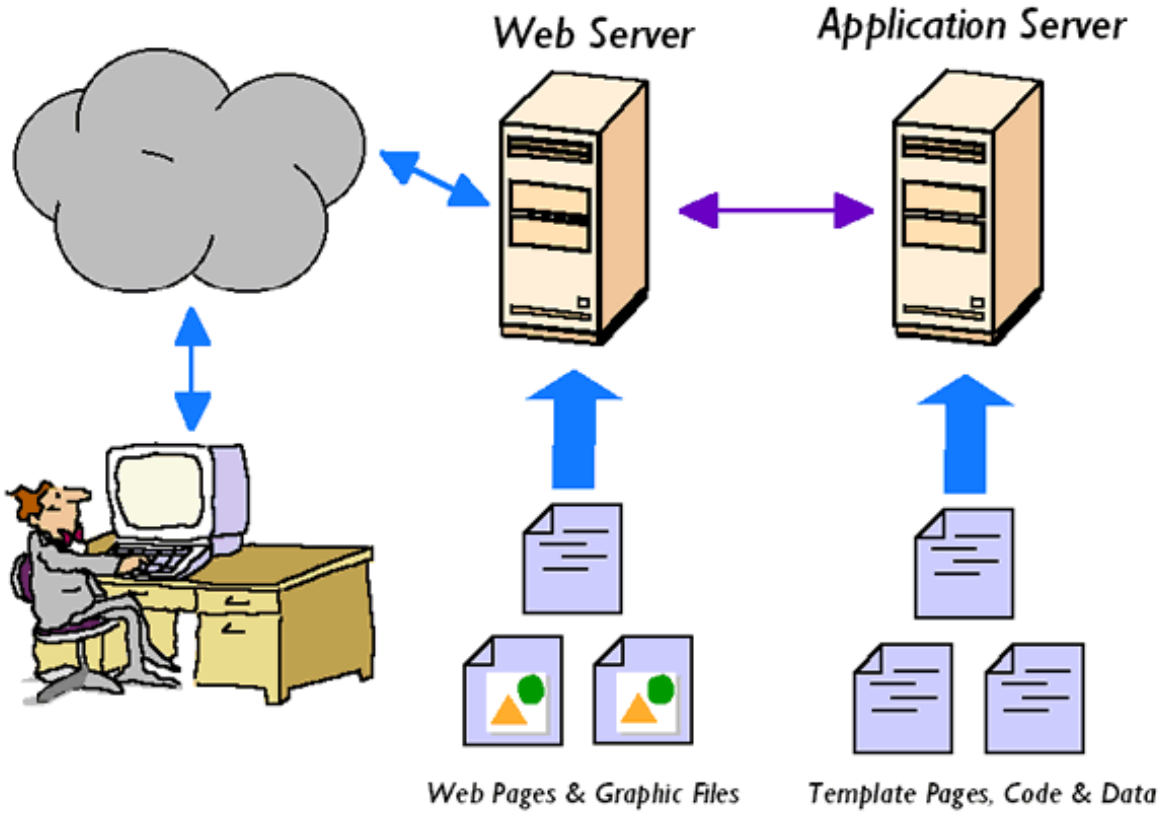
# Các biến thể của Client/Server

- Các ứng dụng **Peer – to – Peer** *(cho phép client và server trao đổi vai trò cho nhau để phân tán và đồng bộ file và thông tin trên nhiều client; đáp ứng các yêu cầu chia sẽ dữ liệu, khám phá tài nguyên, và khả năng phục hồi)*



# Các biến thể của Client/Server

- Các **server ứng dụng** *(các máy chủ server và ứng dụng thực thi và dịch vụ mà một client có thể truy cập thông qua một trình duyệt hoặc phần mềm được cài đặt chuyên biệt ở client).*



# Lợi ích của kiến trúc Client/Server

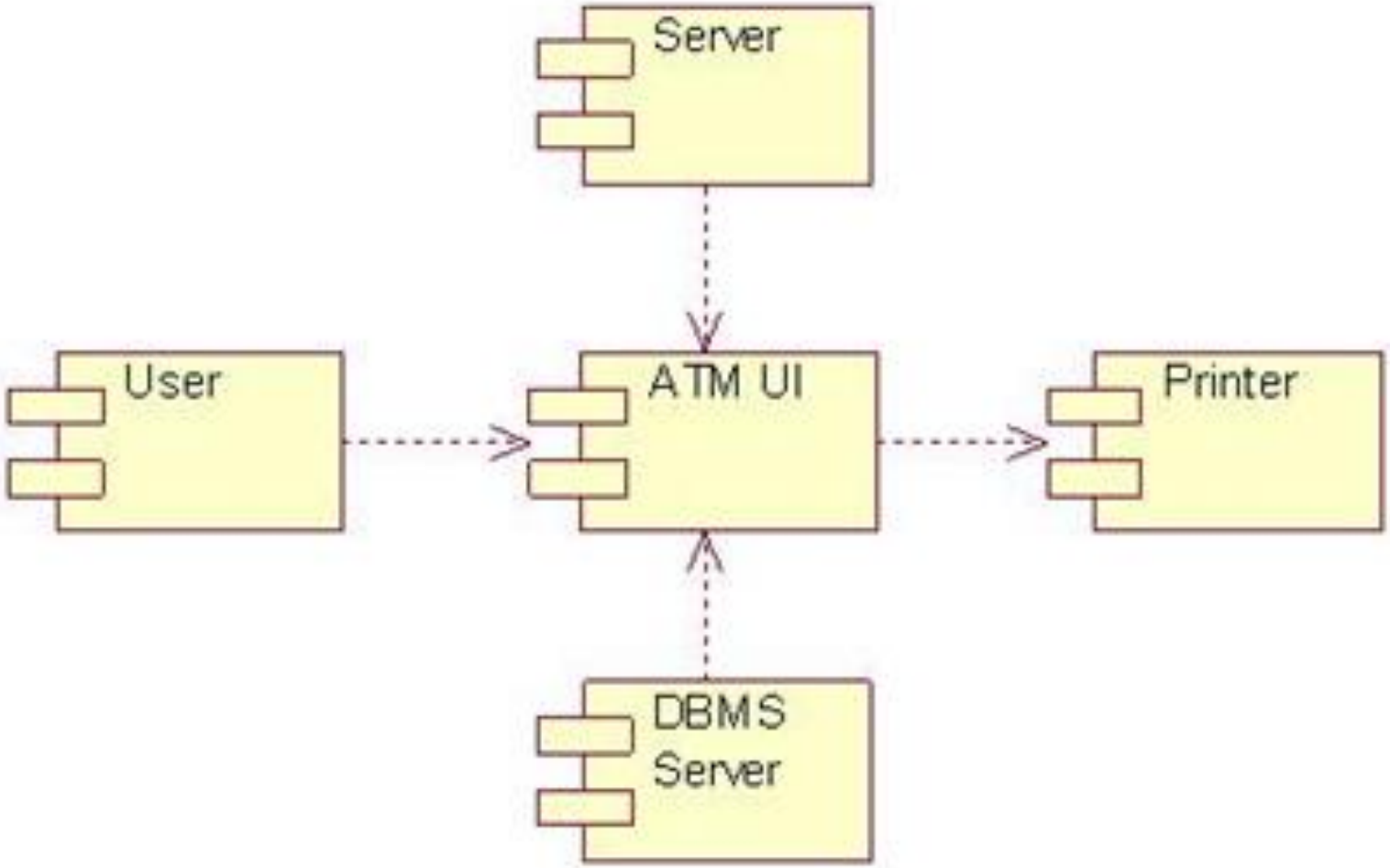
- Bảo mật cao hơn *(dữ liệu được lưu trữ trên server, nó cung cấp khả năng kiểm soát bảo mật tốt hơn client).*
- Truy cập dữ liệu tập trung *(dữ liệu chỉ lưu trữ trên server nên truy cập và cập nhật dữ liệu đươc dễ dàng hơn kiểu kiến trúc khác).*
- Dễ bảo trì.

# Kiến trúc Component-Based

- Phân chia thiết kế ứng dụng thành những thành phần luận lý hoặc chức năng khác nhau, trình bày những giao tiếp được xác định rõ ràng chứa các phương thức, sự kiện, và thuộc tính.

- Cung cấp mức trừu tượng cao hơn nguyên tắc thiết kế hướng đối tượng, và không tập trung vào các vấn đề như giao thức truyền thông và trạng thái chia sẽ.

# Kiến trúc Component-Based



# Kiến trúc Component-Based

- **Các nguyên tắc chính:**
  - Có thể tái sử dụng
  - Thay thế
  - Không phụ thuộc vào ngữ cảnh cụ thể *(được thiết kế để hoạt động trong môi trường và ngữ cảnh khác nhau)*
  - Mở rộng *(có thể mở rộng từ các thành phần hiện có để cung cấp hành vi mới)*
  - Đóng gói
  - Độc lập

# Kiến trúc Component-Based

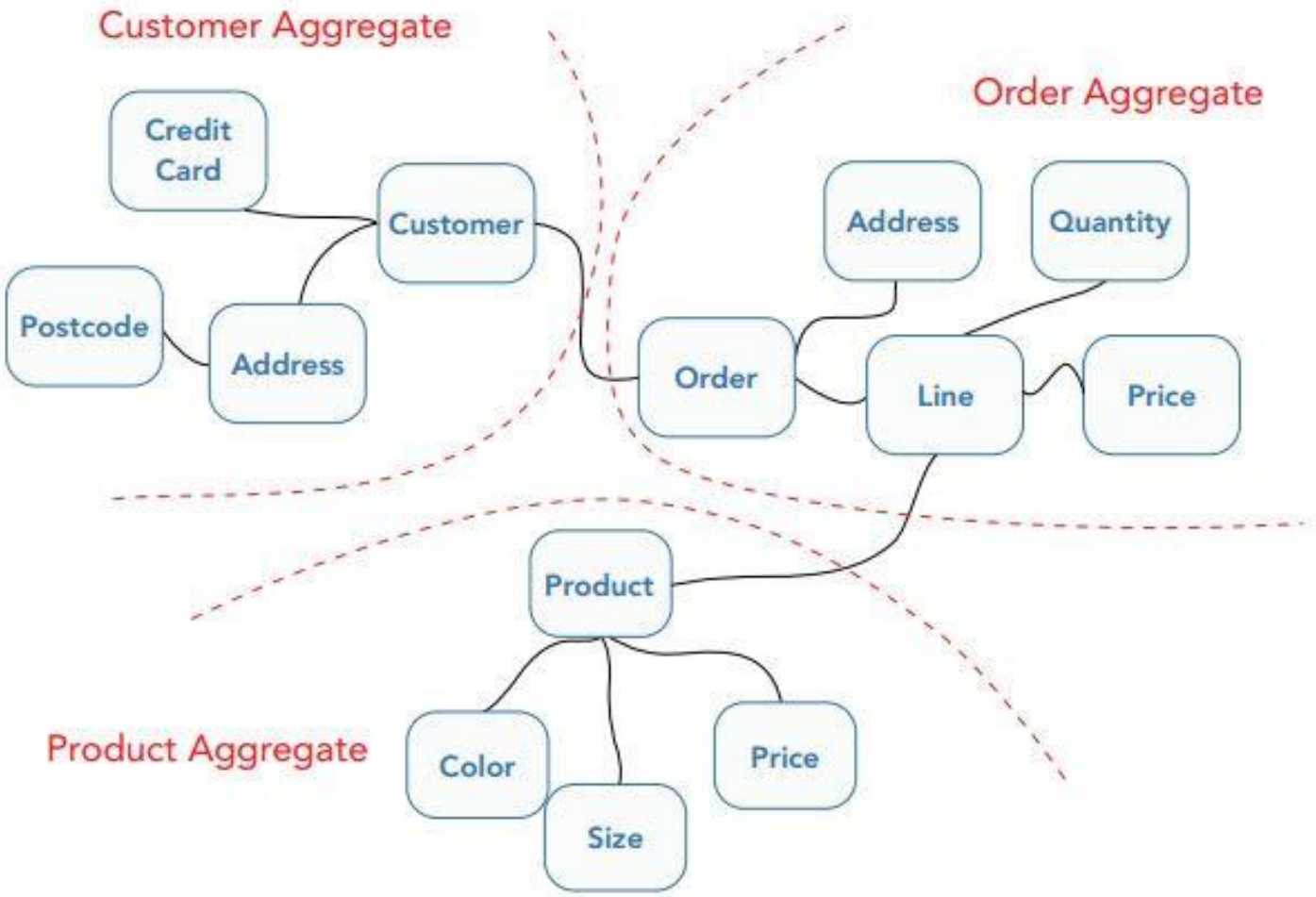
- **Các lợi ích:**
  - Dễ triển khai *(dễ dàng thay thế phiên bản mà không ảnh hưởng các thành phần khác hay hệ thống tổng thể)*
  - Giảm chi phí *(sử dụng thành phần bên thứ 3)*
  - Dễ phát triển
  - Có thể tái sử dụng
  - Giảm thiểu sự phức tạp về mặt kỹ thuật

# Kiến trúc Domain Driven Design

- DDD là một hướng tiếp cận hướng đối tượng để thiết kế phần mềm dựa trên lĩnh vực kinh doanh, các yếu tố và hành vi nghiệp vụ, và các mối quan hệ giữa chúng.

- Cho phép các hệ thống phần mềm hiểu rỏ lĩnh vực kinh doanh nền tảng bằng cách xác định mô hình lĩnh vực thể hiện trong ngôn ngữ của những chuyên gia lĩnh vực kinh doanh.

# Kiến trúc Domain Driven Design

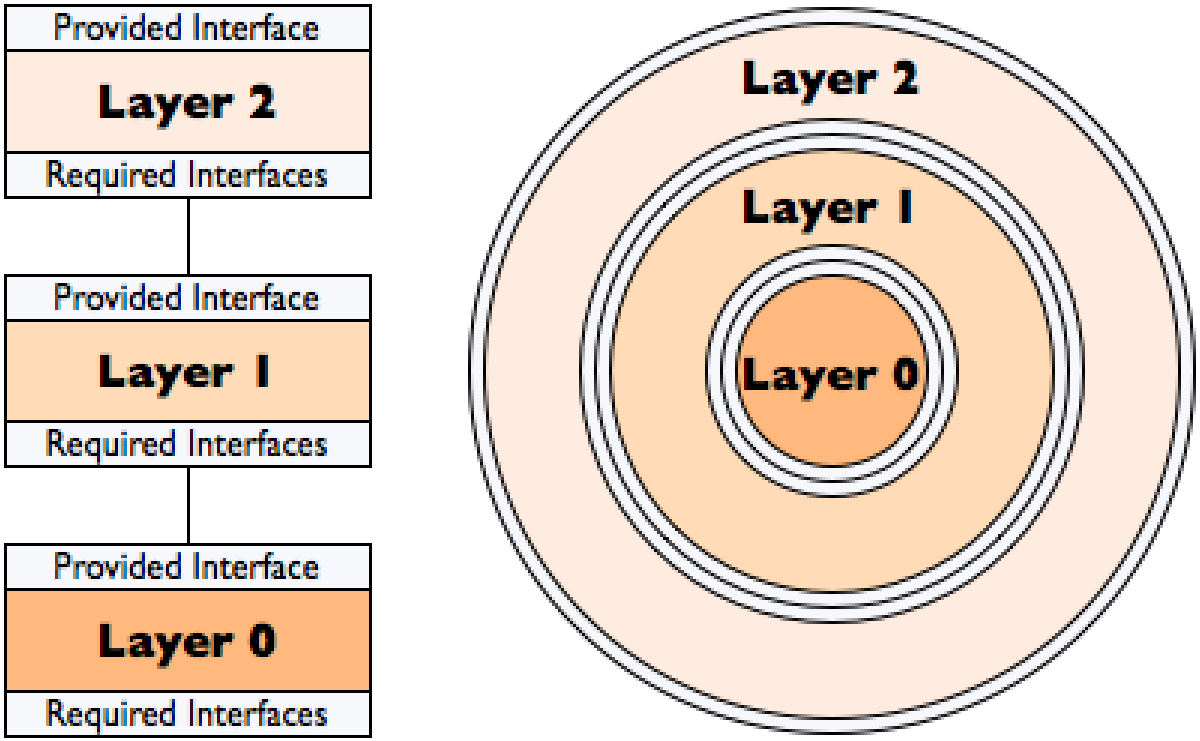


# Kiến trúc Domain Driven Design

- **Các lợi ích:**
  - **Giao tiếp** *(Các bên trong nhóm phát triển có thể sử dụng mô hình lĩnh vực và các thực thể được xác định để giao tiếp tri thức kinh doanh và các yêu cầu sử dụng ngôn ngữ lĩnh vực kinh doanh phổ biến mà không đòi hỏi thuật ngữ kỹ thuật)*
  - **Mở rộng** *(Mô hình lĩnh vực thường mô đun hóa và linh hoạt, làm cho dễ cập nhật và mở rộng thay đổi)*
  - **Có thể kiểm tra** *(Các đối tượng mô hình lĩnh vực được kết nối lỏng lẻo và gắn kết, cho phép kiểm tra dễ dàng)*

# Kiến trúc Layered

- Kiến trúc dựa trên lớp tập trung vào việc nhóm chức năng có liên quan bên trong một ứng dụng thành các lớp riêng biệt được xếp chồng lên nhau.



# Kiến trúc Layered

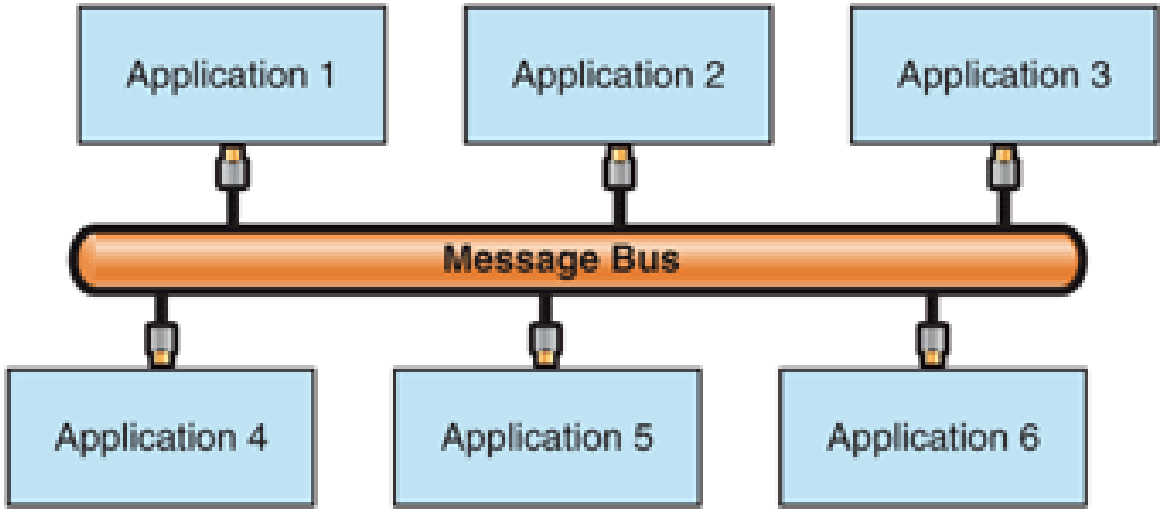
- **Các nguyên tắc chính:**
  - Trừu tượng *(Kiến trúc dựa trên lớp trừu tượng View của hệ thống, cung cấp đủ chi tiết để hiểu vai trò và trách nhiệm của lớp riêng biệt và các mối quan hệ)*
  - Đóng gói
  - Các lớp chức năng được xác định rõ ràng
  - Sự kết dính cao *(xác định rõ ràng ranh giới cho từng lớp, đảm bảo mỗi lớp chứa chức năng liên quan trực tiếp đến công việc của lớp đó)*
  - Có thể tái sử dụng *(lớp dưới không phụ thuộc lớp trên)*
  - Khớp nối lỏng lẻo

# Kiến trúc Layered

- **Các lợi ích quan trọng:**
  - Trừu tượng *(Cho phép thay đổi để thực hiện ở cấp trừu tượng)*
  - Cô lập *(Cho phép cô lập các nâng cấp công nghệ cho các lớp riêng biệt để giảm rủi ro và ảnh hưởng đến hệ thống)*
  - Dễ quản lý
  - Hiệu quả *(Phân tán các lớp trên nhiều tầng vật lý, cải thiện khả năng mở rộng, chịu lỗi và hiệu quả)*
  - Khả năng tái sử dụng
  - Khả năng kiểm tra

# Kiến trúc Message Bus

- Kiến trúc message bus được xem là một hệ thống phần mềm có thể nhận và gửi thông điệp, sử dụng một hay nhiều kênh truyền thông, cho nên các ứng dụng có thể tương tác mà không cần biết chi tiết cụ thể về nhau.



# Kiến trúc Message Bus

- **Một message bus cung cấp khả năng xử lý:**
  - Các giao tiếp theo hướng thông điệp
  - Logic xử lý phức tạp *(hoạt động phức tạp có thể thực hiện bằng cách kết hợp một tập các hoạt động nhỏ hơn)*
  - Sửa đổi logic xử lý *(do sự tương tác với bus dựa trên các lược đồ và dòng lệnh chung nên có thể chèn hay xóa các ứng dụng trên bus để thay đổi logic xử lý các thông điệp)*
  - Tương tác với các môi trường khác nhau *(Microsoft .NET và Java)*

# Kiến trúc Message Bus

- **Các biến thể trên kiểu message bus:**
  - Enterprise Service Bus (ESB) *(sử dụng các dịch vụ để giao tiếp giữa bus và các thành phần gắn trong bus, cung cấp các dịch vụ chuyển đổi thông điệp từ định dạng này sang định dạng khác, cho phép các client sử dụng định dạng thông điệp không tương thích với những cái khác)*
  - Internet Service Bus (ISB) *(tương tự như ESB nhưng các ứng dụng được đặt trên cloud thay vì trên mạng doanh nghiệp, sử dụng URIs và các chính sách kiểm soát định tuyến logic thông qua ứng dụng và dịch vụ trên cloud)*

# Kiến trúc Message Bus

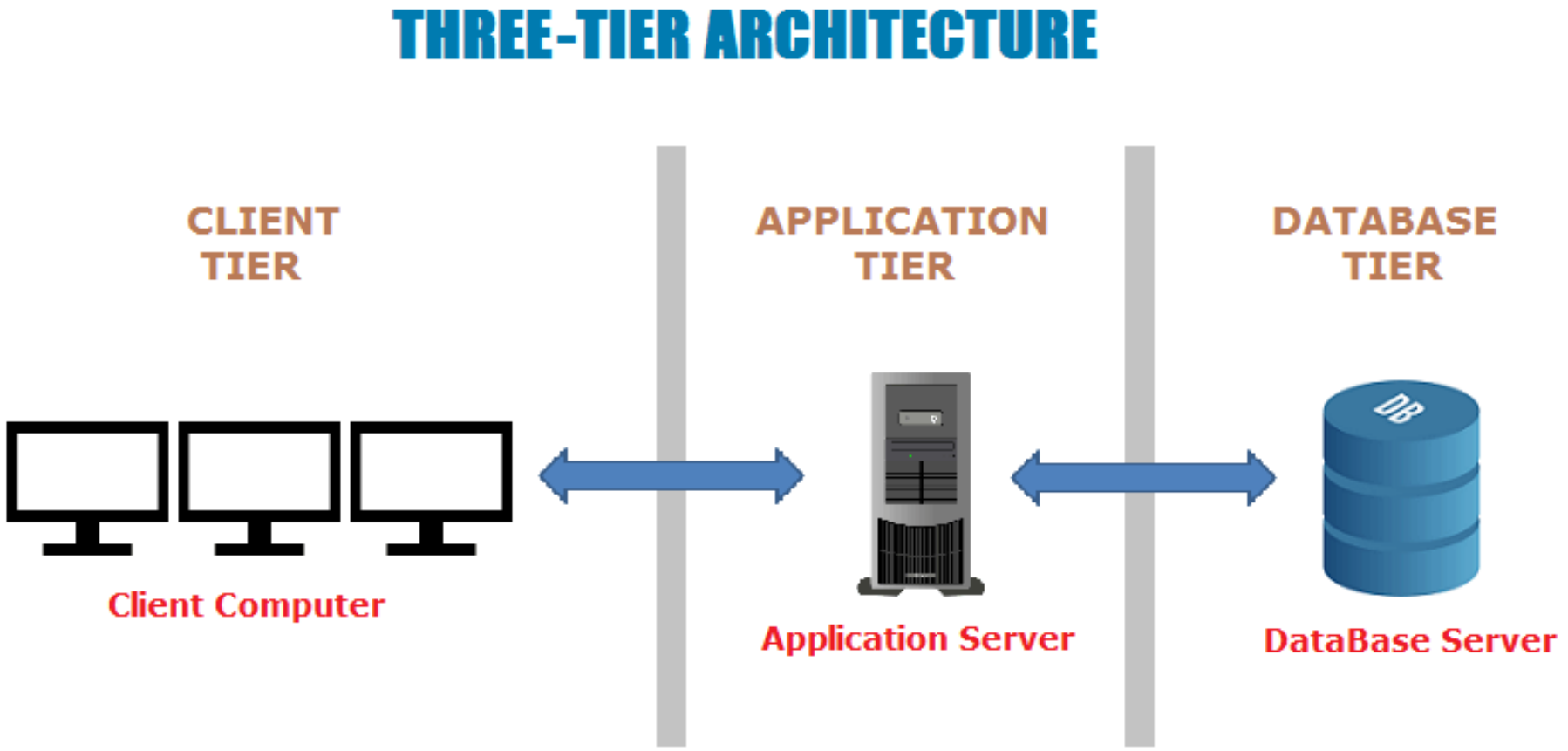
- **Các lợi ích của kiểu kiến trúc message bus:**
  - Khả năng mở rộng *(các ứng dụng có thể thêm vào hay bỏ ra từ bus mà không ảnh hưởng đến các ứng dụng hiện có)*
  - Độ phức tạp thấp *(do mỗi ứng dụng chỉ cần biết giao tiếp thế nào với bus)*
  - Mềm dẻo *(có thể thay đổi dễ dàng phù hợp yêu cầu nghiệp vụ hay người dùng, đơn giản thông qua thay đổi cấu hình hay các tham số kiểm soát định tuyến)*
  - Khả năng mở rộng *(có thể gắn vào bus để xử lý nhiều yêu cầu ở cùng thời điểm)*
  - Ứng dụng đơn giản *(chỉ cần kết nối duy nhất đến bus)*

# Kiến trúc N-Tier/3-Tier

- N-Tier/3Tier là các kiểu phát triển kiến trúc mô tả sự tách biệt chức năng thành các phân đoạn theo cách tương tự như kiểu lớp, nhưng với mỗi phân đoạn là một tầng nó có thể là được đặt trên máy tính riêng biệt vật lý.

- Phát triển thông qua hướng tiếp cận theo hướng thành phần, thường sử dụng nền tảng các phương thức cụ thể cho giao tiếp thay vì hướng tiếp cận dựa trên thông điệp.

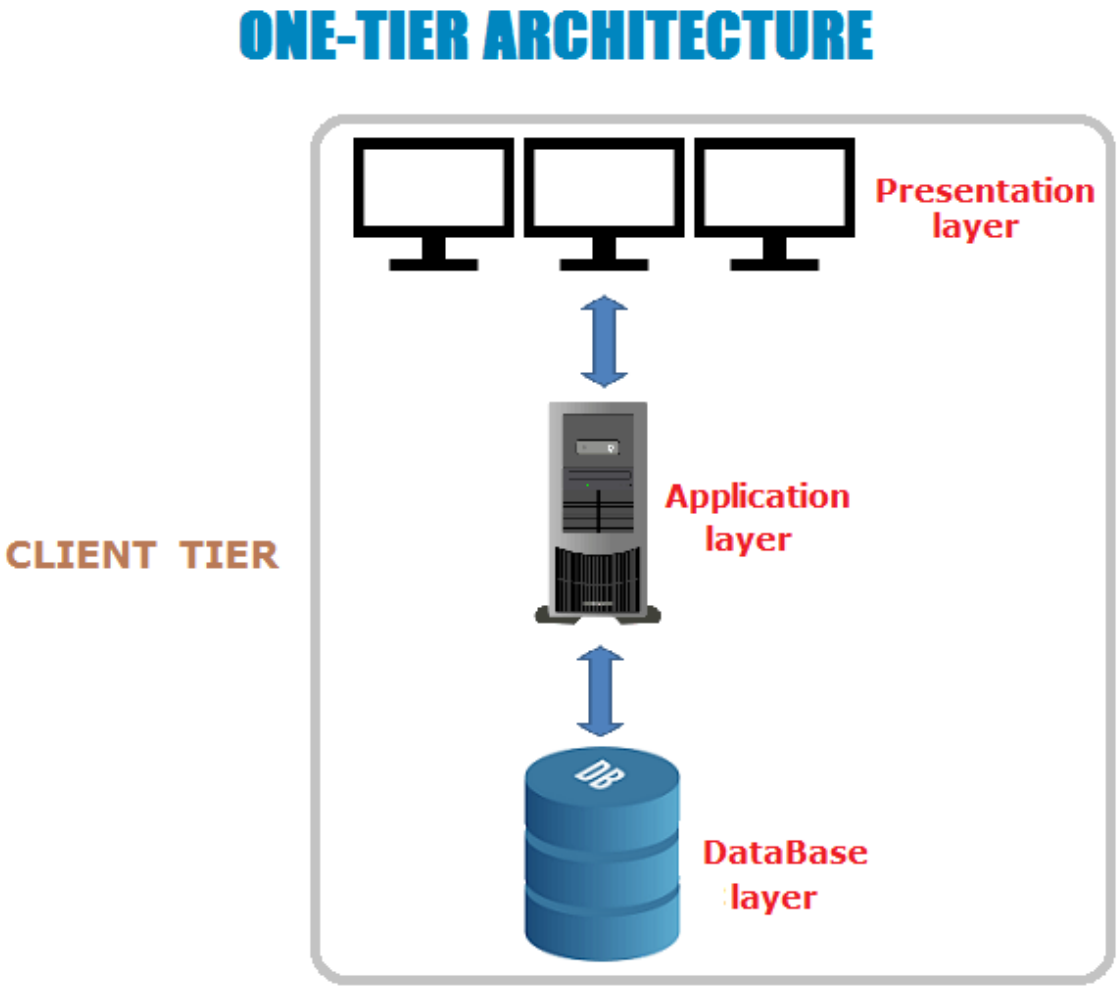
# Kiến trúc N-Tier/3-Tier



# Kiến trúc N-Tier/3-Tier

- **Các lợi ích của kiểu kiến trúc N-tier/3-tier:**
  - Bảo trì *(mỗi tầng là độc lập, cập nhật hay thay đổi mà không ảnh hưởng đến toàn bộ ứng dụng)*
  - Khả năng mở rộng *(các tầng dựa trên việc triển khai các lớp, việc mở rộng ứng dụng thì minh bạch và hợp lý)*
  - Mềm dẻo *(mỗi tầng có thể quản lý hay mở rộng một cách độc lập, tính linh hoạt tăng lên)*
  - Khả dụng *(Các ứng dụng có thể khai thác kiến trúc mô đun cho phép các hệ thống sử dụng các thành phần có thể mở rộng dễ dàng, nó làm tăng tính khả dụng)*

# Kiến trúc N-Tier/3-Tier



# Kiến trúc Object-Oriented

- Là một lược đồ thiết kế dựa trên việc phân chia trách nhiệm đối với một ứng dụng hay hệ thống thành các đối tượng có thể tái sử dụng và tự cung cấp riêng lẻ, mỗi đối tượng chứa dữ liệu và hành vi liên quan đến đối tượng.

- Các đối tượng rời rạc, độc lập, và kết hợp lỏng lẻo; chúng giao tiếp thông qua *Interface*, bằng cách gọi các phương thức hay truy cập các thuộc tính trong các đối tượng khác, và bằng cách gửi hay nhận thông điệp.

# Kiến trúc Object-Oriented

- **Nguyên tắc của kiểu kiến trúc là:**
  - **Trừu tượng** *(cho phép giảm một hoạt động phức tạp thành tổng quát hóa, nó giữ lại những đặc tính cơ bản của hoạt động)*
  - **Hợp thành** *(các đối tượng có thể được lắp ráp từ các đối tượng khác, có thể ẩn các đối tượng bên trong các lớp khác hay trưng bày như các Interface đơn giản)*
  - **Kế thừa** *(các đối tượng có thể kế thừa từ các đối tượng khác, sử dụng tính năng trong đối tượng cơ sở hay ghi đề lên để triển khai hành vi mới; bảo trì và cập nhật dễ hơn)*

# Kiến trúc Object-Oriented

- **Nguyên tắc của kiểu kiến trúc là:**
  - **Đóng gói** *(các đối tượng trưng bày tính năng chỉ thông qua phương thức, thuộc tính và sự kiện; và ẩn chi tiết bên trong như trạng thái và các biến)*
  - **Đa hình** *(cho phép ghi đè lên hành vi của một loại cơ sở nhằm hỗ trợ các hoạt động trong ứng dụng bằng cách triển khai loại mới có thể hoán đổi cho các đối tượng hiện có)*
  - **Tách riêng** *(Đối tượng được tách rời khỏi người tiêu dùng bằng cách định nghĩa một Interface trừu tượng mà các triển khai và tiêu dùng đối tượng có thể hiểu được; cho phép triển khai thay đổi mà không ảnh hưởng người tiêu dùng)*

# Kiến trúc Object-Oriented

- **Các lợi ích của kiến trúc:**
  - Có thể hiểu được *(nó ánh xạ ứng dụng chặt chẽ hơn với các đối tượng thế giới thực; làm cho dễ hiểu hơn nhiều)*
  - Có thể tái sử dụng *(khả năng tái sử dụng thông qua đa hình và trừu tượng)*
  - Có thể kiểm tra được *(kiểm tra được cải thiện thông qua đóng gói)*
  - Mở rộng *(đóng gói, đa hình và trừu tượng đảm bảo một thay đổi trong biểu diễn dữ liệu không ảnh hưởng đến các interface của đối tượng)*
  - Tính kết hợp cao

# Kiến trúc Service-Oriented (SOA)

- Kiến trúc hướng dịch vụ (SOA) cho phép chức năng ứng dụng được cung cấp như một tập các dịch vụ, và tạo ra các ứng dụng sử dụng các dịch vụ phần mềm.

- Các dịch vụ trong SOA tập trung vào việc cung cấp một lược đồ và sự tương tác dựa trên thông điệp với một ứng dụng thông qua các giao diện trong phạm vi ứng dụng, và không phải là thành phần hay dựa trên đối tượng.

# Kiến trúc Service-Oriented (SOA)

- Kiểu SOA có thể đóng gói quá trình xử lý nghiệp vụ vào các dịch vụ tương tác, sử dụng một loạt các giao thức và định dạng dữ liệu để truyền thông tin.

- Các client và những dịch vụ khác có thể truy cập các dịch vụ cục bộ chạy trên cùng tầng, hay truy cập các dịch vụ từ xa qua mạng kết nối.

# Kiến trúc Service-Oriented (SOA)

## ➢ Các nguyên tắc của SOA:

- Các dịch vụ là tự trị *(mỗi dịch vụ có thể được duy trì, phát triển, triển khai, và được phiên bản độc lập)*
- Các dịch vụ được phân tán *(các dịch vụ có thể được đặt ở bất kỳ nơi nào trên mạng, nội bộ hay từ xa, miễn là mạng hỗ trợ các giao thức giao tiếp theo yêu cầu)*
- Các dịch vụ kết hợp lỏng lẻo *(Mỗi dịch vụ là độc lập với những cái khác, có thể thay thế hay cập nhật mà không vi phạm các ứng dụng miễn là giao diện vẫn tương thích)*

# Kiến trúc Service-Oriented (SOA)

## ➢ Các nguyên tắc của SOA:

- Các dịch vụ chia sẽ lược đồ và định ước, không phải lớp *(các dịch vụ chia sẽ các định ước và lược đồ khi giao tiếp, chứ không phải các lớp nội bộ)*

- Khả năng tương thích được dựa trên chính sách *(Chính sách trong trường hợp này nghĩa là việc xác định các tính năng như chuyển tải, giao thức và an ninh)*

# Kiến trúc Service-Oriented (SOA)

## ➢ Những lợi ích của SOA:

- Điều chỉnh lĩnh vực *(Tái sử dụng các dịch vụ phổ biến với các giao diện chuẩn làm tăng cơ hội kinh doanh và công nghệ và giảm chi phí)*
- Trừu tượng *(Các dịch vụ là tự trị và được truy cập thông qua định ước định dạng, cung cấp kết nối lỏng lẻo và trừu tượng)*
- Khả năng khám phá *(Các dịch vụ có thể trưng bày những mô tả cho phép các ứng dụng và các dịch vụ khác nhau định vị và tự động xác định interface)*

# Kiến trúc Service-Oriented (SOA)

## ➢ Những lợi ích của SOA:

- Khả năng tương tác *(Do các giao thức và định dạng dữ liệu được dựa trên các tiêu chuẩn ngành công nghiệp, nhà cung cấp và tiêu thụ của dịch vụ có thể được xây dựng và phát triển trên các nền tảng khác nhau)*

- Hợp lý hóa *(Các dịch vụ có thể được chi tiết để cung cấp tính năng cụ thể, chứ không phải nhân bản tính năng trong một số các ứng dụng, loại bỏ trùng lặp)*


Q&A