CSE 301
Software Design and Architecture

4. Introduction to Architectural Styles

Dr. Salisu Garba
Salisu.garba@slu.edu.ng

Outline

- Introduction to architectural styles
- Categorizations of architectural styles
- Hierarchical architectures
  - Layered Architecture
- Distributed architectures
  - Client Server Architecture
  - Multi-tier Architecture
  - Service Oriented Architecture

Architectural Styles & Its Components

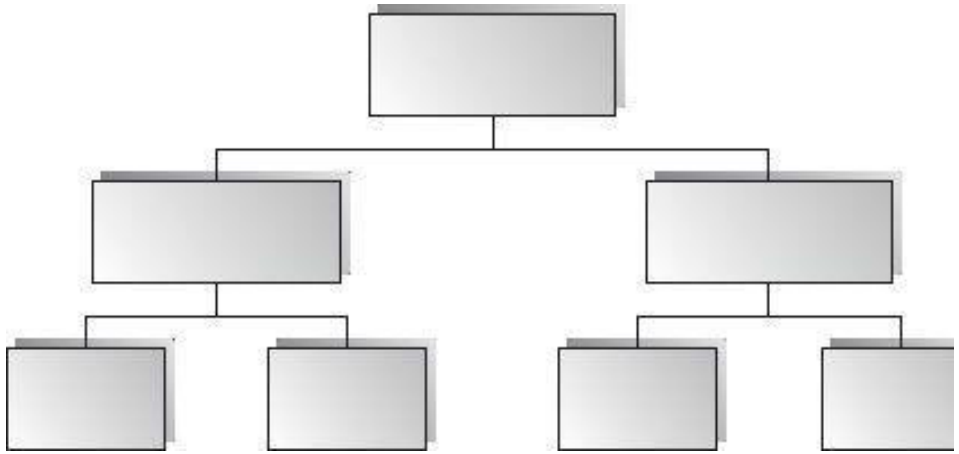
- An architectural style is a set of **characteristics and features** that make a software **notable** or historically **identifiable**.
- It's a coarse-grained pattern that provide an **abstract framework** for a **family of systems**.
- The **key elements** of an architecture style are:
- **Components**
  - that **perform functions** required by a system
- **Connectors**
  - that **enable communication, coordination**, and cooperation among elements
- **Constraints**
  - that define **how elements can be integrated** to form the system
- **Attributes**
  - that describe the **advantages and disadvantages** of the chosen **structure**

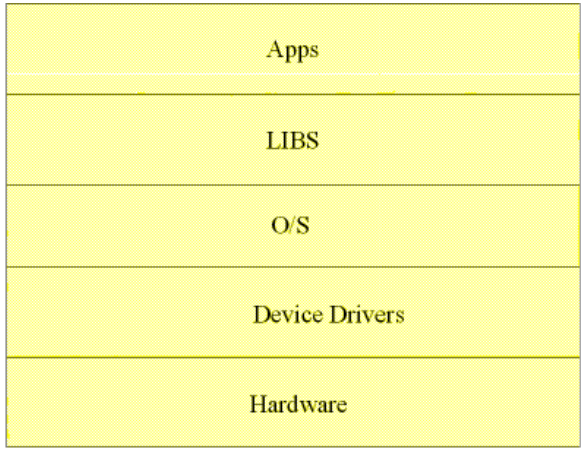
Categories of Architectural Styles

- Hierarchal Software Architecture
  - Layered
- Distributed Software Architecture
  - Client Server
  - SOA
- Data Flow Software Architecture
  - Pipe n Filter
  - Batch Sequential
- Event Based Software Architecture
- Data Centered Software Architecture
  - Black box
  - Shared Repository
- Interaction-Oriented Software Architectures
  - Model View Controller
- Component-Based Software Architecture

Hierarchical Style

- The hierarchical software architecture is characterized by viewing the entire system as a hierarchy structure. The software system is **decomposed into logical modules** (subsystems) at **different levels** in the hierarchy.
- Modules at different levels are connected by explicit or implicit method invocations.
  - a lower-level module provides services to its adjacent upper-level modules, which invokes the methods or procedures in the lower level.
- System software is typically designed using the hierarchical architecture style;
  - examples include Microsoft .NET, Unix operating system, TCP/IP, etc.





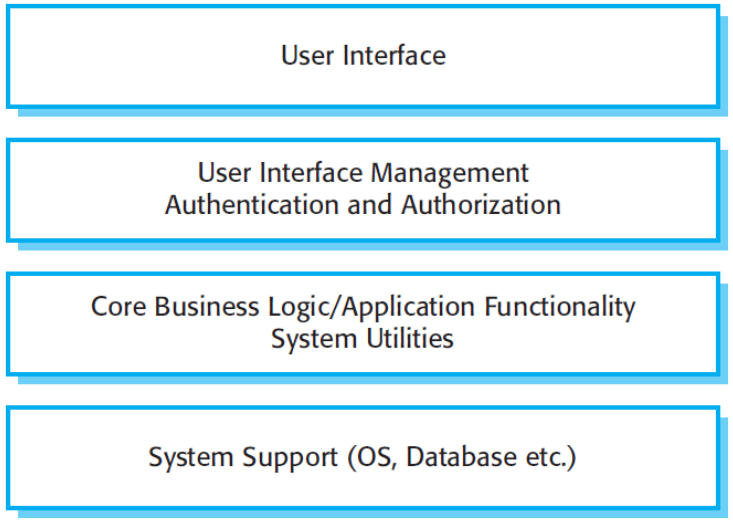
Hierarchal Style

- **Lower layers** provide more specific functionality down to fundamental **utility services**
  - such as I/O services, transaction, scheduling, and security services, etc.
- **Middle layers**, in an application setting, provide more **domain-dependent functions**
  - such as business logic or core processing services.
- Upper layers provide more abstract functionality in the form of **user interfaces**
  - such as command line interpreters, GUIs, Shell programming facilities, etc.

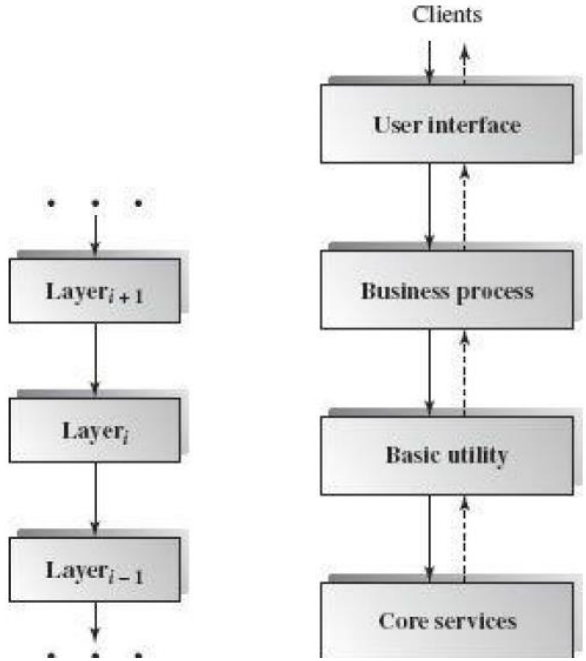
Layered Style

- Organized hierarchically into layers.
- Each layer provides service to the layer above it and serves as a client to the layer below.
- The connectors are defined by the protocols that determine how the layers will interact.

A generic Layered Architecture



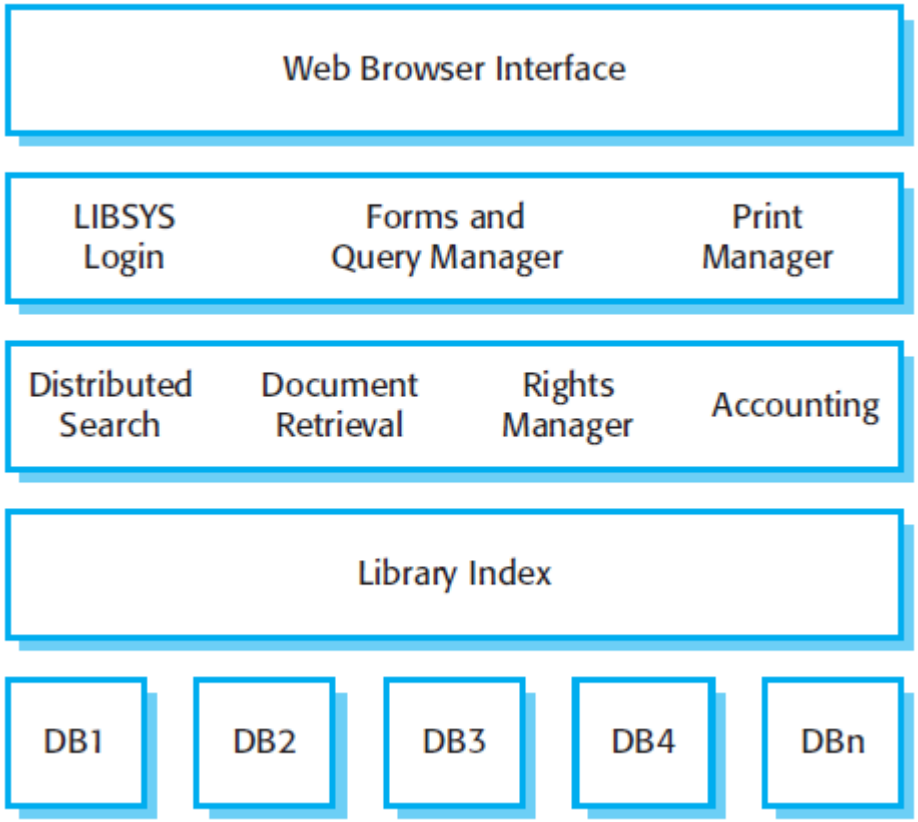
A partial layered architecture



Example: Library System

- Library system
- Allows controlled electronic access to copyright material from a group of university libraries.
- Bottom layer being the individual database in each library.

- Components:
  - UI
  - Authentication and forms
  - Search engine
  - Document retrieval
  - Rights manager
  - Accounts management
  - databases



- **Ref: Software Engineering - Sommerville**

Applicable domains of layered architecture:

- Any system that can be divided between the application-specific portions and platform-specific portions which provide generic services to the application of the system.

- Applications that have clean divisions between **core services**, **critical services**, **user interface services**, etc.

- Applications that have a number of classes that are closely related to each other so that they can be grouped together into a package to provide the services to others.

Benefits:

- Enhanced independence of upper layer to lower layer since there is no impact from the changes of lower layer services as long as their interfaces remain unchanged.
- **Enhanced flexibility**: interchangeability and reusability are enhanced due to the separation of the standard interface and its implementation.
- Component-based technology is a suitable technology to implement layered architecture; this makes it much easier for the system to allow for **plug-and-play of new components**.
- **Promotion of portability**: each layer can be an abstract machine deployed independently.

Limitations:

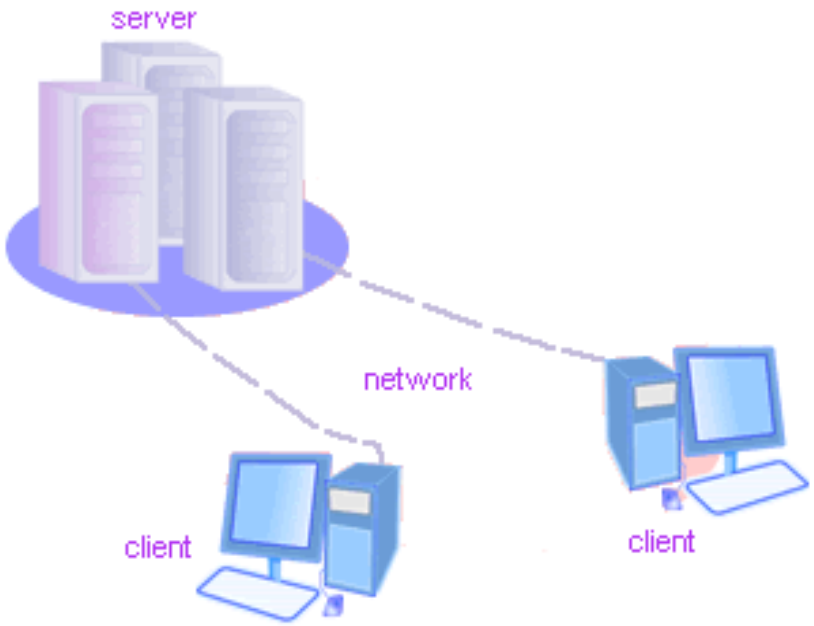
- **Lower runtime performance** since a client's request or a response to a client must go through potentially several layers.
- There are also performance concerns of overhead on the data processing and **buffering** by each layer.
- Many applications cannot fit this architecture design.
- Breach of interlayer communication may cause **deadlocks**, and “bridging” may cause tight coupling.
- Exceptions and error handling are issues in the layered architecture, since faults in one layer must propagate upward to all calling layers.

Distributed Software Architecture

- A distributed system is a collection of **computational** and **storage** devices **connected** through a communications **network**.
- Data, software, and users are distributed.
- Communication occurs using a number of methods including message passing, remote procedure calls, and remote method invocation.
  - **Client Server Architecture**
  - **Multi-tier Architecture**

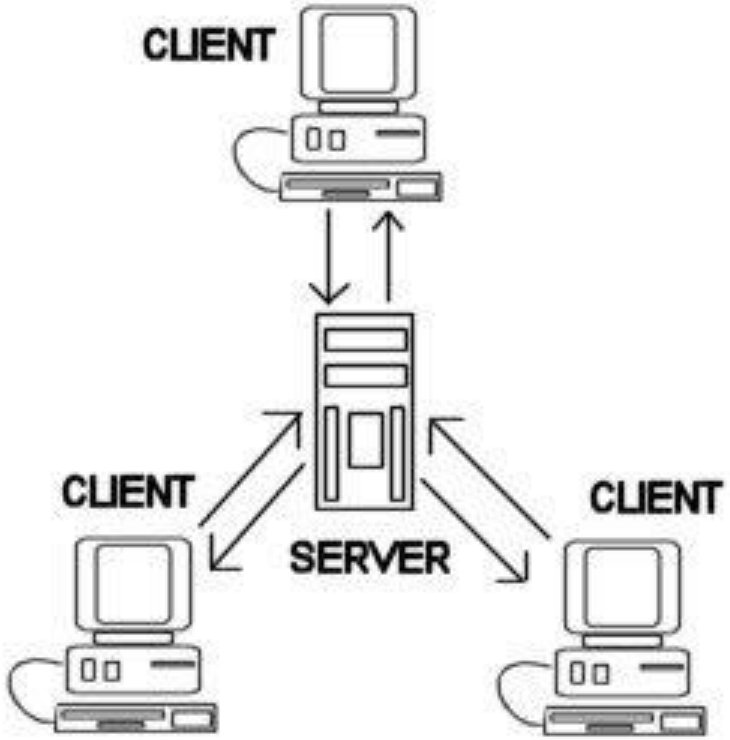
Client Server Architectural Style

- Client/server architecture illustrates the relationship between two computer programs in which one program is a client, and the other is Server.
- Suitable for applications that involve distributed data and processing across a range of components.
- **Components:**
  - **Servers:** Stand-alone components that provide specific services such as printing, data management, etc.
  - Server provides service to the request.
  - **Clients:** Components that call on the services provided by servers.
  - Client makes a service request to server.
- **Connector:** The network, which allows clients to access remote servers.



Client Server Architectural Style Example 1

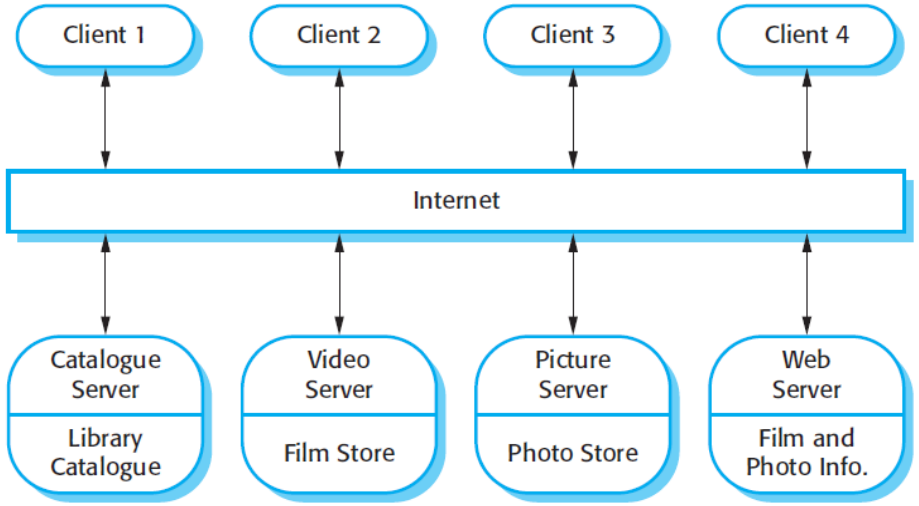
- The World Wide Web is an example of client-server architecture.
- Each computer that uses a **Web browser is a client**, and the data on the various **Web pages** that those clients access is **stored** on multiple **servers**.
- Although the client/server architecture can be used within a single computer by programs, but it is a more important idea in a network.
- In a network, the client/server architecture allows efficient way to interconnect programs that are distributed efficiently across different locations.



Types of Servers

- Servers commonly contain data files and applications that can be accessed across the network, by workstations or user computers.
- A user who wants to access data files, would use his or her client computer to access the data files on the server.
- **Application Servers:**
  - Applications are hosted on these servers
  - Clients can access various application features over the network
- **File Servers:**
  - Primitive form of data service.
  - Useful for sharing files across a network.
  - The client passes requests for files over the network to the file server.
- **Database Servers:**
  - More efficient use of distributing power than file servers.
  - Client passes SQL requests as messages to the DB server; results are returned over the network to the client.
  - Query processing done by the server.
  - No need for large data transfers.

Advantages & Disadvantages of Client Server Architecture



- **Advantages**
  - Straightforward distribution of data.
  - Transparency of location.
  - Mix and match heterogeneous platforms
  - Easy to add new servers or upgrade existing servers.

- **Disadvantages**
  - Performance of the system depends on the performance of the network.
  - Tricky to design and implement Client/Server systems.
  - Unless there is a central register of names and services, it may be hard to find out what services are available.

Types of Client Server

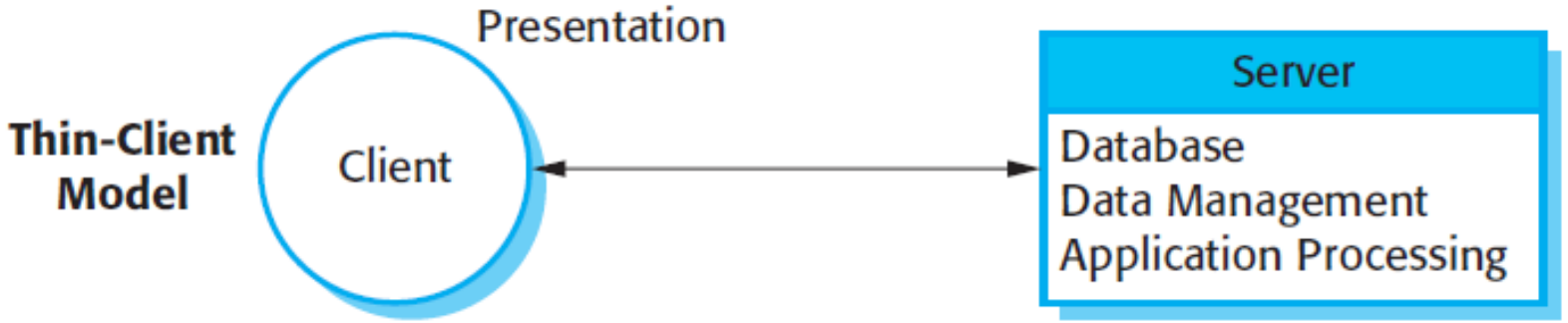
- Two-tier client–server architecture,
  - which is used for simple client–server systems, and in situations where it is **important to centralize** the system for security reasons.
  - In such cases, communication between the client and server is normally encrypted.

- Two forms of this architectural model:
  - **A thin-client model,**
  - **A fat-client model,**

- Multitier client–server architecture,
  - which is used when there is a **high volume of transactions** to be processed by the server.

Two-tier Client Server

- **A thin-client model**,
  - where the **presentation layer is implemented on the client** and all other layers (data management, application processing, and database) are implemented on a server.

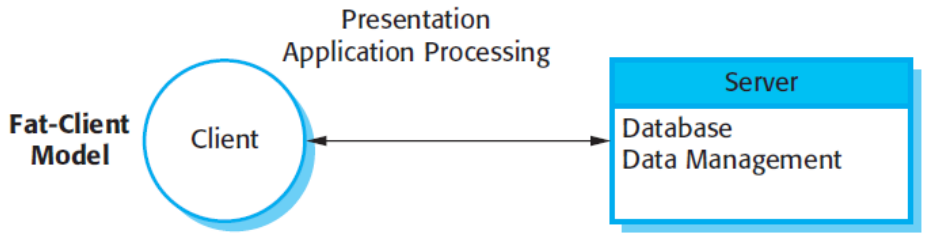


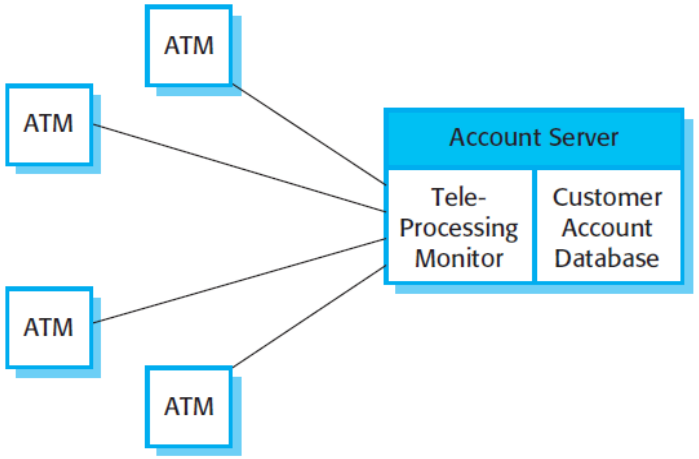
- **The advantage** of the thin-client model is that it is **simple to manage** the clients. If a **web browser** is used as the client, there is **no need to install any software**.

- **The disadvantage** of the thin-client approach, however is that it may place a **heavy processing load on both the server and the network**.

Two-tier Client Server

- **A fat-client model**,
  - where some or all of the **application processing is carried out on the client**.
  - Data management and database functions are implemented on the server.



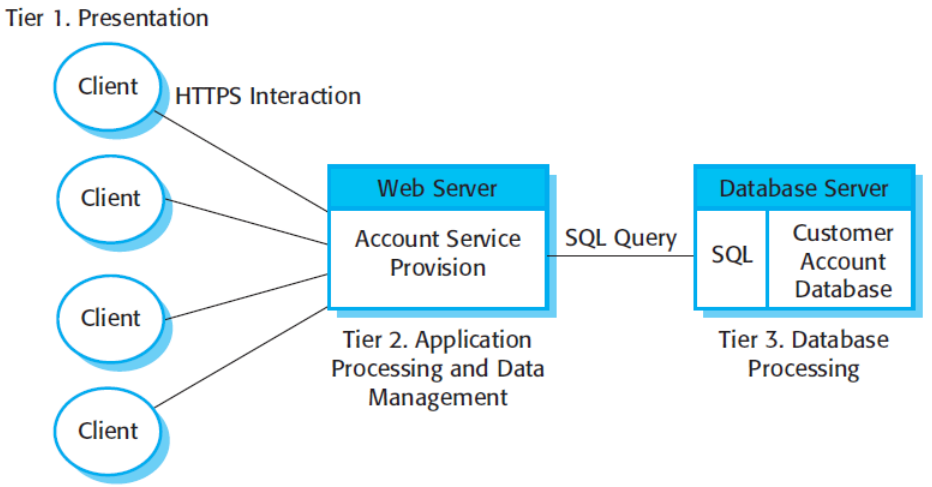


- **The advantage** of having a fat client system is that the **functionality of the application can be easily extended** by the users of the system

- **The disadvantage** is that software **changes to the system** have to be communicated to what could be a **large number of client computers**

Multi-tier client–server architectures

- The fundamental **problem** with a **two-tier client–server** approach is that the **logical layers in the system (**presentation, application processing, data management, and database) must be mapped onto **two computer systems**: the **client and the server**.
- This may lead to **problems with scalability & performance** if the thin-client model is chosen, or **problems of system management** if the fat-client model is used.
- Application services such as facilities to transfer cash, pay bills, etc. are implemented in the web server & as scripts that are executed by the client
  - **This system is scalable because it is relatively easy to add servers (scale out) as the number of customers increase.**
  - **In this case, the use of a three-tier architecture allows the information transfer between the web server and the database server to be optimized.**

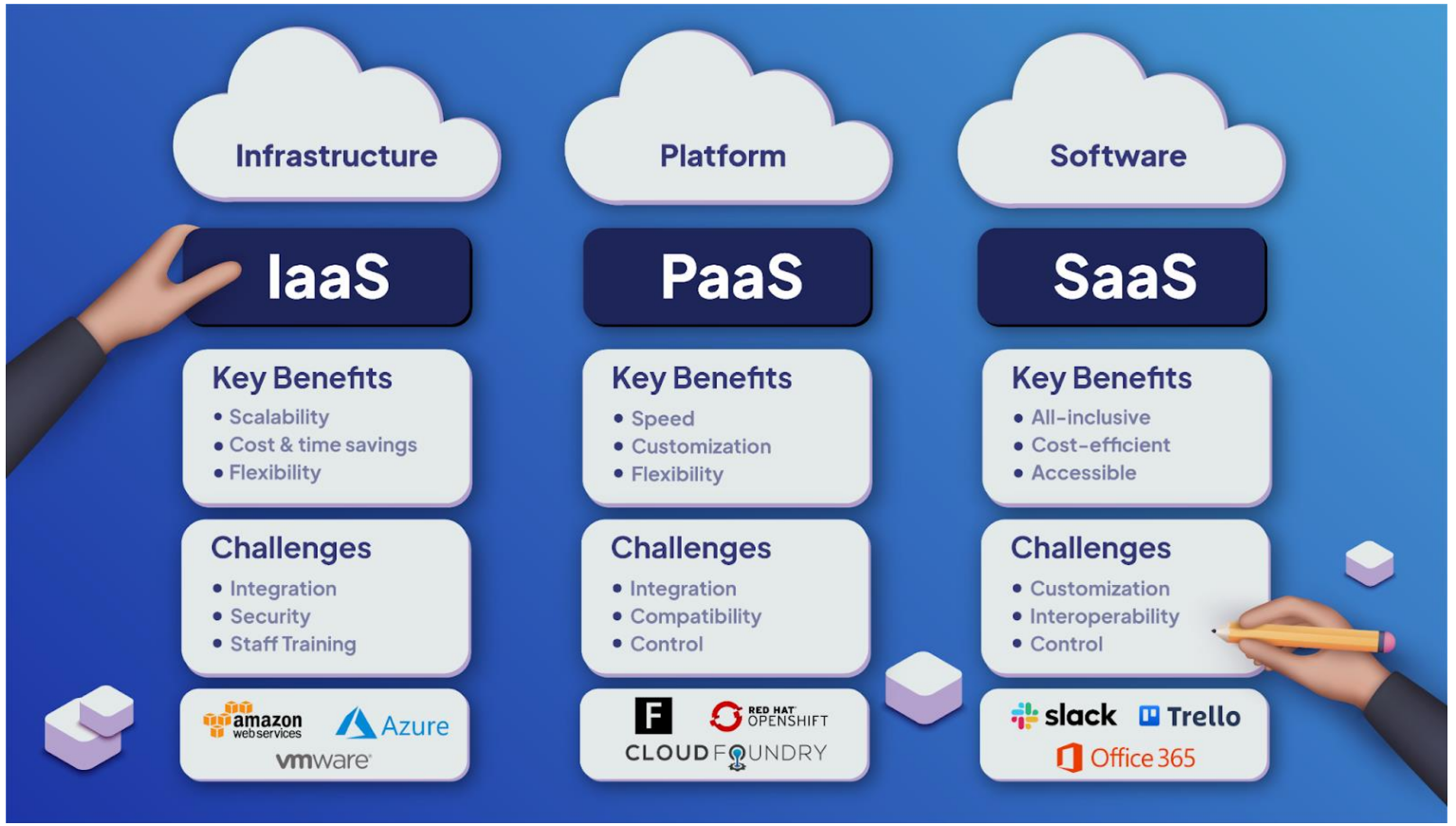


Service Oriented Architecture (SOA)

- Service Oriented Architecture (SOA) is a **business centric** IT architectural model in which **business functionality are logically grouped** and encapsulated into self contained, distinct and reusable units called **services**
- A service is a business functionality that is
  - **well-defined,**
  - **self-contained,**
  - **Independent from other services,**
- and published and available to be used via a standard programming interface.
- Service orientation is a particular strategy for separating concerns and dividing a system into components.
- Its fundamental characteristic is that every component provides a distinct service that can be used by **multiple consumers**.

Example of Web Services

- Different Systems require different group of **web services**
- It consists of four existing web services:
  - Airline reservation
  - Car rental
  - Hotel reservation
  - Attraction reservation
  - **Electronic mail**
  - **Payment**
  - **Maps/ Locations**
  - **Shopping Cart**
  - Storage
  - Grammar Checker
  - Plagiarism Checker

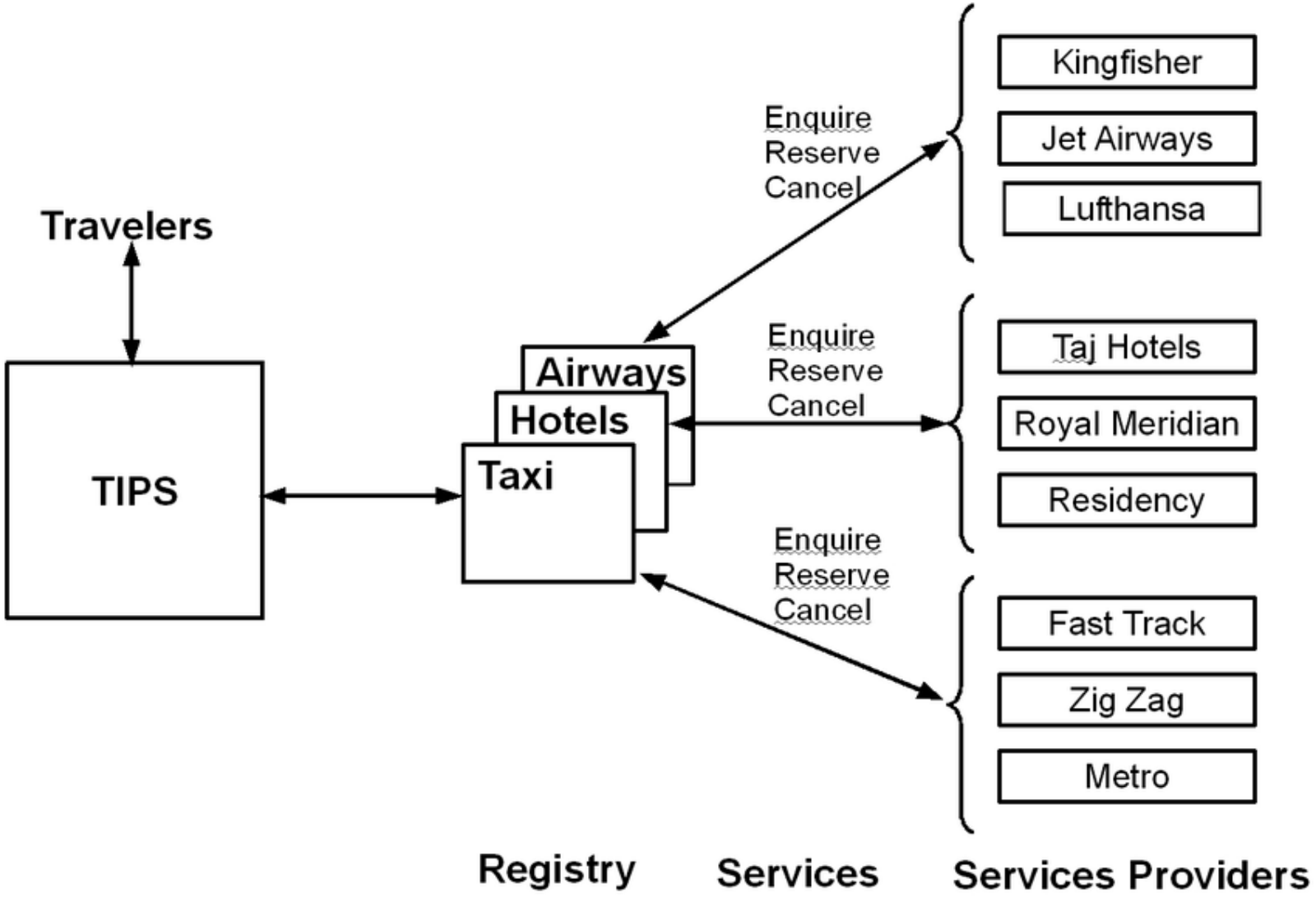


Types of Services

- There are several types of services used in SOA systems.
- Business services
  - Business service can be defined as the logical encapsulation of **business functions**. It has to be relevant to the business the organization is running. e.g. **legal, accounting**
- Entity services
  - An entity service usually represents **business entities** (e.g. Employee, Customer, Product, Invoice etc.). Such entity service usually expose CRUD (create, read, update, delete) operations. e.g. **login using google account**
- Functional services
  - It is usually a technology-oriented service and not a business oriented one. Task services can be thought of as controller of composition of services and hence its reusability is usually lower. e.g. **chatbots, auto-fit, auto generate**
- Utility services
  - Utility services offers common and reusable services that are usually not business centric. They might include notifications exception handling etc. e.g. **antivirus**

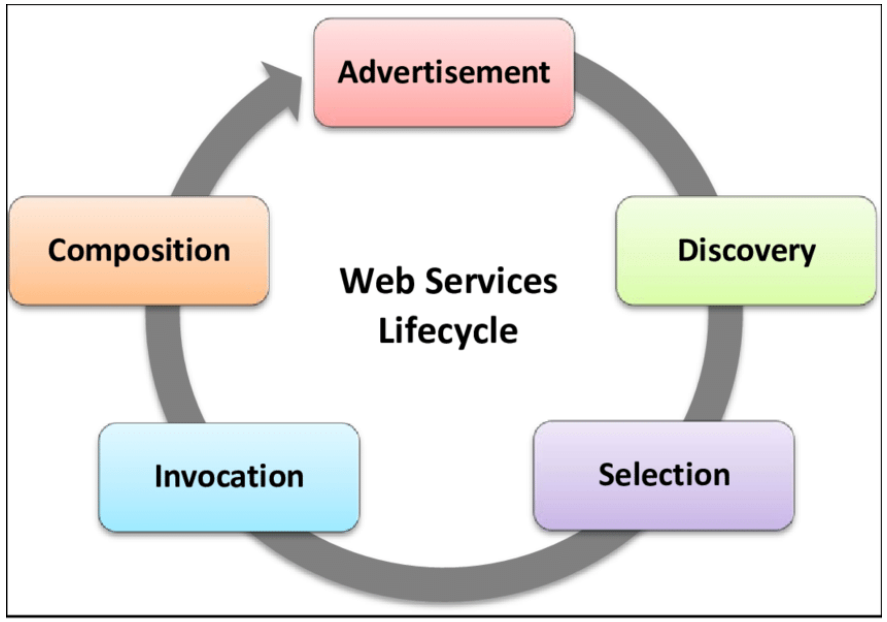
Example 1-SOA based model for Business

Example 2-SOA based model for Travel Itinerary Planning System

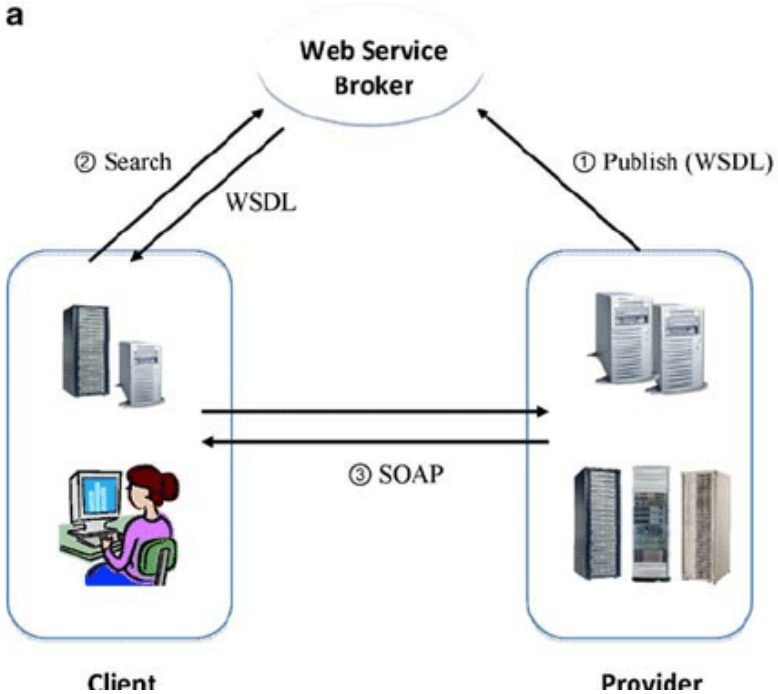


Web Services Lifecycle

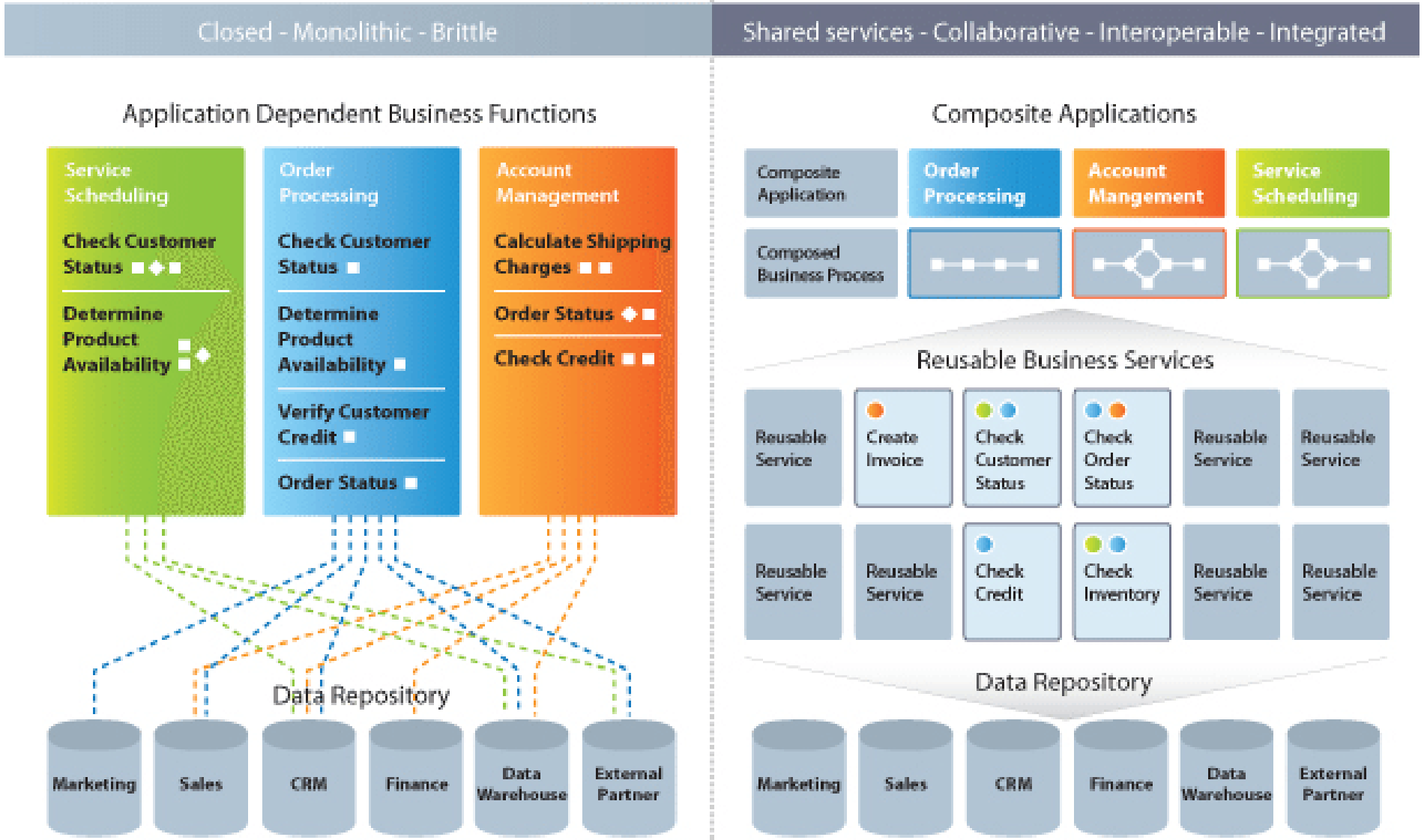
- A service can only be used if potential clients are **aware of its existence**, and have information about **how to use it**.
- This is achieved through **service description**.
  - A service description contains at least the name of the service, the location of the service, and the data exchange requirements.



a



Before SOA and After SOA



Need for SOA

- **Isolating business logic**
  - The biggest problem in programming is often it is very difficult to keep the **business logic separated from the so called "computer logic"**.
  - **Non-IT folks** can change the business logic any time, without understanding how a small change could result in possibly **disproportionate amount of work** required by the IT folks to implement the change.
- **Interoperability**
- Redundancies
  - There are **many similar yet slightly different applications** that are used throughout the organization.
  - **Each department** usually comes out with **its own version of software** components.
- Ability to build business applications faster and more easily
- Easy maintenance/update and Lower total cost of ownership

Summary

- Introduction to architectural styles
- Categorizations of architectural styles
- Hierarchical architectures
  - Layered Architecture
- Distributed architectures
  - Client Server Architecture
  - Multi-tier Architecture
  - Service Oriented Architecture

CSE301-SDA Assignment

- **Use any tool available E.g.** Lucidchart, Cacoo, EdrawMax, Microsoft Visio Pro, Astah, StarUML **to design a Software based on** 4 +1 View Model

- **Use any tool available E.g. FLUID, AppInventor (Android), LucidChart, Wavemaker, Visual Studio to design a GUI for your project**

Instructions:

1. Due date: **20-05-2024**
2. Maximum of **5 students per group**
3. Write your Name and Registration Number Clearly
4. Document/Compile/Zip and Submit to salisu.garba@slu.edu.ng
5. Plagiarism checker will be used and 0 marks will be awarded for plagiarized work.

Q & A

Any Question?