



CSE- 321

Software Engineering

Lecture : 13

Software design

Fahad Ahmed

Lecturer, Dept. of CSE

E-mail: fahadahmed@uap-bd.edu

Lecture Outlines

- ✧ **Architectural Design**
- ✧ **Abstract machine (layered) Design**
- ✧ **Distributed Systems Architectures**
- ✧ **Client-Server Architecture**
- ✧ **Service-Oriented Architecture (SOA)**
- ✧ **Cloud based Software engineering**

Model-View-Controller



The Model-View-Controller (MVC) pattern

- Serves as a basis of interaction management in many **web-based systems**.
- Decouples three major interconnected components:
 - The **model** is the central component of the pattern that directly manages the data, logic and rules of the application. It is the application's dynamic data structure, independent of the user interface.
 - A **view** can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible.
 - The **controller** accepts input and converts it to commands for the model or view.
- Supported by most language frameworks.

The Model-View-Controller (MVC) pattern

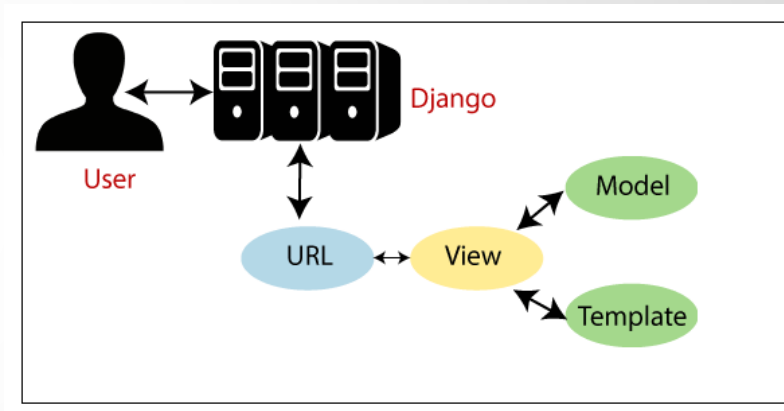
- Some other design patterns followed by MVC, such as

- **MVVM (Model-View-Viewmodel),**

- MVP (Model-View-Presenter)

- MVW (Model-View-Whatever).

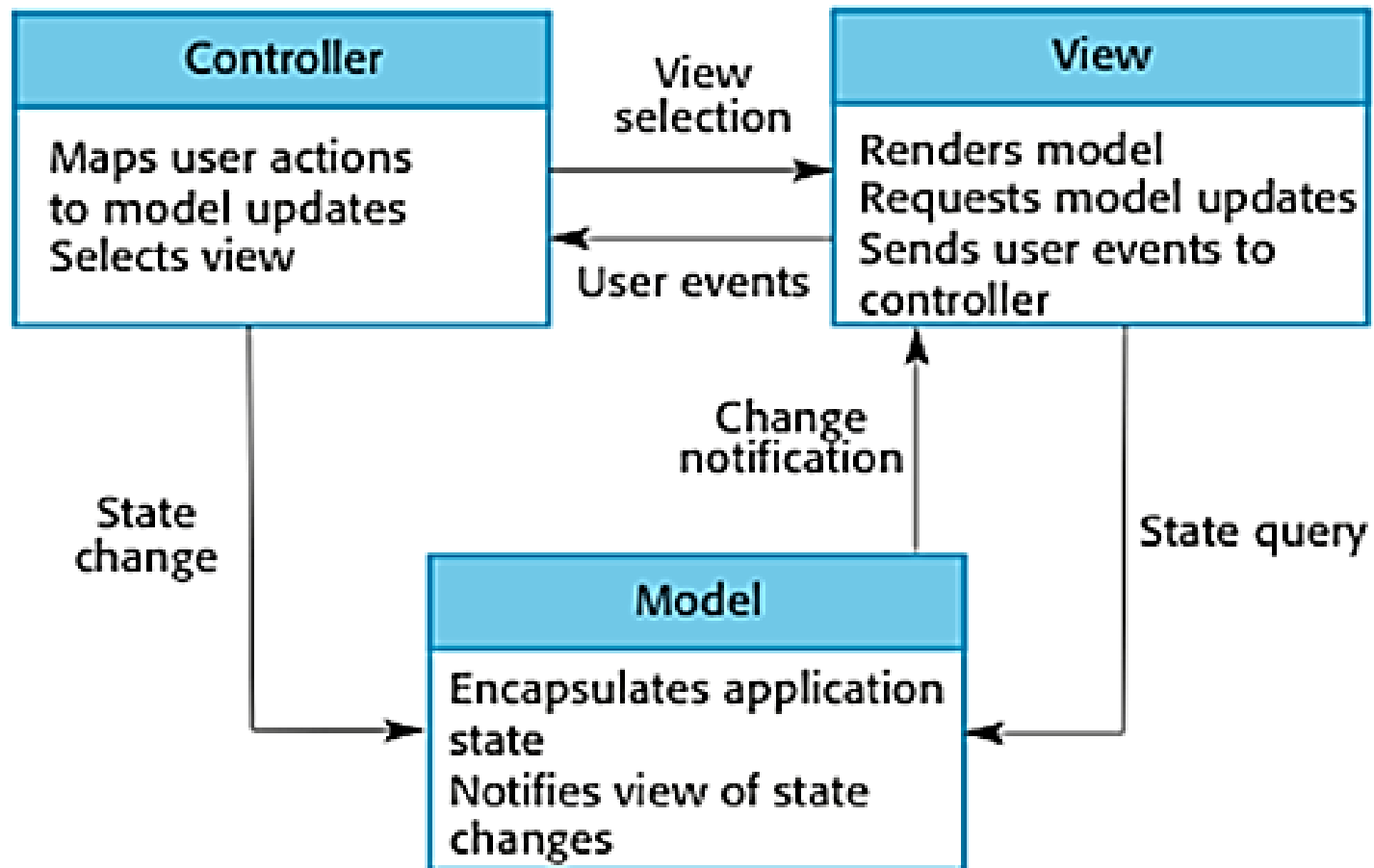
- **Model View Template**



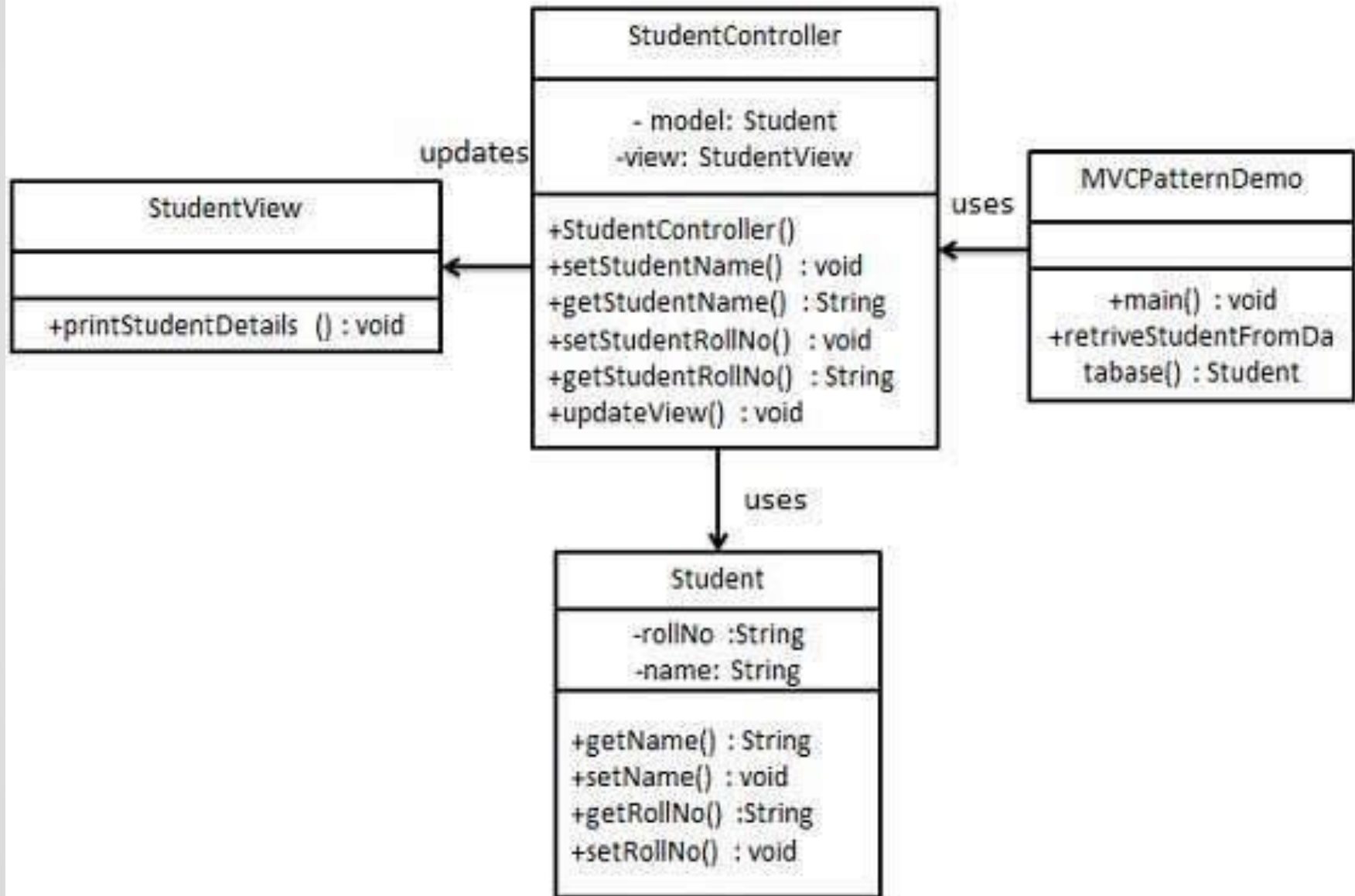
It is a collection of three important components **Model View and Template**. The Model helps to handle **database**. It is a data access layer which handles the data. **In an MVT, the controller part is taken care of by the framework itself.**

The Model-View-Controller (MVC) pattern

MVC architectural pattern



Coding Style mapping: MVC



Design Patterns - MVC

Step 1 Create Model

```
public class Student {  
    private String rollNo;  
    private String name;  
  
    public String getRollNo() {  
        return rollNo;  
    }  
  
    public void setRollNo(String rollNo) {  
        this.rollNo = rollNo;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```


Design Patterns - MVC

Step 2

Create view

```
public class StudentView {  
    public void printStudentDetails(String studentName, String studentRollNo){  
        System.out.println("Student: ");  
        System.out.println("Name: " + studentName);  
        System.out.println("Roll No: " + studentRollNo);  
    }  
}
```

Design Patterns - MVC

Step 3 Create Controller

```
public class StudentController {  
    private Student model;  
    private StudentView view;  
  
    public StudentController(Student model, StudentView view){  
        this.model = model;  
        this.view = view;  
    }  
  
    public void setStudentName(String name){  
        model.setName(name);  
    }  
    public String getStudentName(){  
        return model.getName();  
    }  
    public void setStudentRollNo(String rollNo){  
        model.setRollNo(rollNo);  
    }  
    public String getStudentRollNo(){  
        return model.getRollNo();  
    }  
  
    public void updateView(){  
        view.printStudentDetails(model.getName(), model.getRollNo());  
    }  
}
```

Design Patterns - MVC

Step 4 MVCPatternDemo

```
public class MVCPatternDemo {  
    public static void main(String[] args) {  
  
        //fetch student record based on his roll no from the database  
        Student model = retrieveStudentFromDatabase();  
  
        //Create a view : to write student details on console  
        StudentView view = new StudentView();  
  
        StudentController controller = new StudentController(model, view);  
  
        controller.updateView();  
  
        //update model data  
        controller.setStudentName("John");  
  
        controller.updateView();  
    }  
  
    private static Student retrieveStudentFromDatabase(){  
        Student student = new Student();  
        student.setName("Robert");  
        student.setRollNo("10");  
        return student;  
    }  
}
```

In Python

https://www.tutorialspoint.com/python_design_patterns/python_design_patterns_model_view_controller.htm

Step 5

Verify the output.

Student:

Name: Robert

Roll No: 10

Home Task on MVC

What is the difference between MVC and the usual 3-tier architecture and why do developers say the MVC architecture is better?

3-Tier is a software architecture approach, in which the user interface (**Presentation Layer**), business process are **logic, data tier** developed independently, most often on separate platforms and can be physically deployed on multiple servers.

3T is **not presentation oriented** , business logic layer no nothing about view(presentation). We can add service layer turn whole system as service to other application.

MVC is more suitable for application that are **presentation orientated** like web app. Where controller trade between view and model.

Home Task on MVC

Which is better, MVC or .NET Core?

MVC is an architectural pattern where as .NET Core is a framework.

Is MVC different from a 3 layered architecture?



What is MVVM?

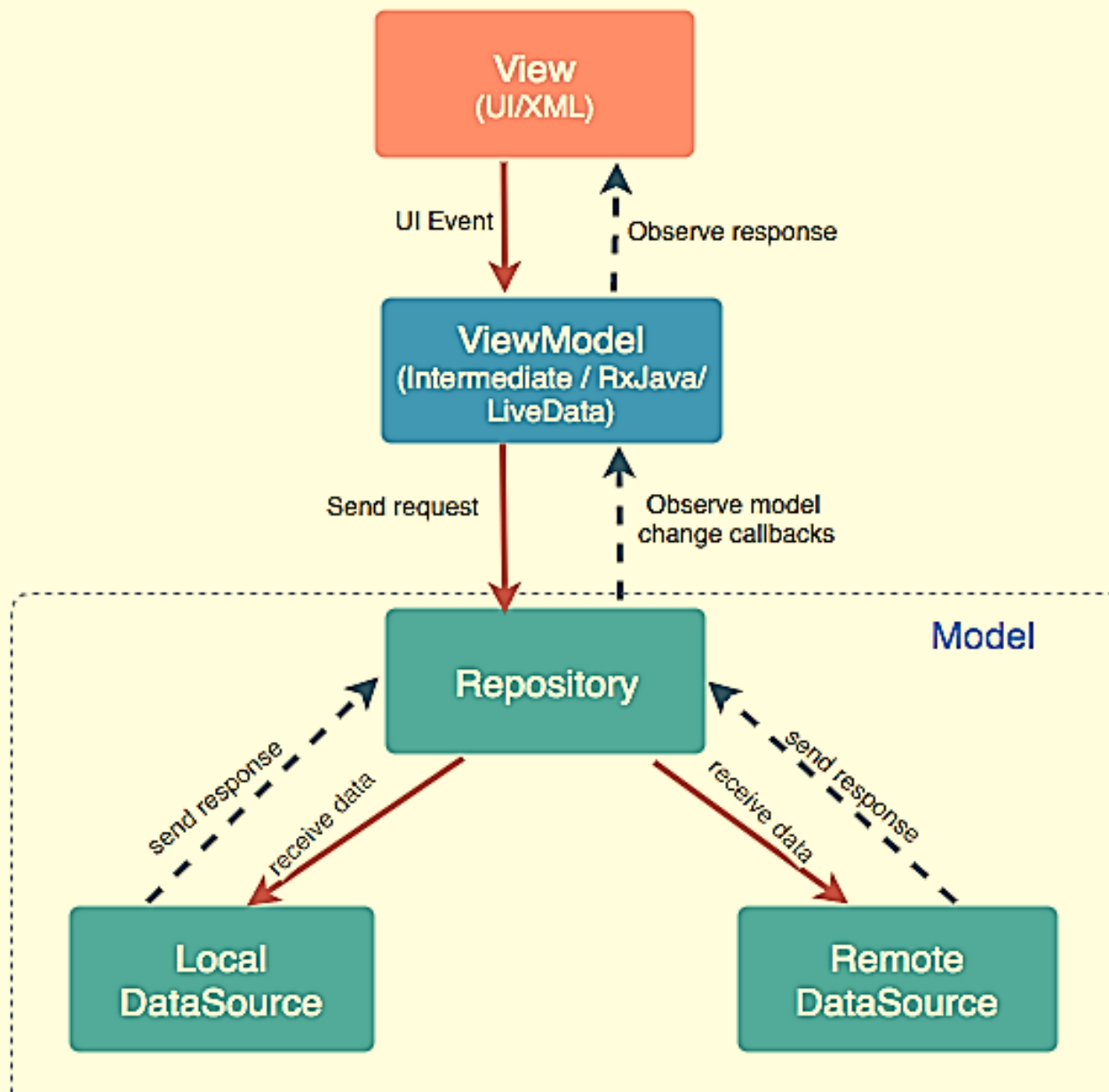
MVVM architecture facilitates a separation of development of the graphical user interface with the **help of mark-up language** or GUI code.

The full form of MVVM is **Model–View–ViewModel**.

MVVM removes the **tight coupling** between each component.

Most importantly, in this architecture, **the children don't have the direct reference to the parent**, they only have the **reference by observables**.

MVVM



Model: It represents the **data and the business logic** of the Application. It consists of the business logic - **local and remote data source, model classes, repository**.

View: It consists of the **UI Code** (Activity, Fragment), XML. It sends the user action to the ViewModel but does **not** get the response back directly. To get the response, it has to subscribe to the observables which ViewModel exposes to it.

ViewModel: It is a **bridge** between the View and Model(business logic). It does not have any clue **which View has to use it as it does not have a direct reference to the View**. So basically, the ViewModel should not be aware of the view who is interacting with. It interacts with the Model and exposes the observable that can be observed by the View.

Layered architecture

Have you ever wondered how Google makes **Gmail work** in different languages all over the world? Users can use Gmail every day in English, Spanish, French, Russian, and many more languages.

Did Google develop different Gmail applications for each country?
Of course not.

They developed **an internal version** that does all the message processing, and then developed different external user interfaces that work in many languages.

Layered architecture

Google developed the Gmail application in **different layers**:

1. There is an **internal layer** that does all the processing.
2. There is an **external layer** that communicates with the users in their language.
3. There is also **another layer that interacts with a database** where user email messages are stored (millions or maybe billions).

Gmail is divided into at least three layers, every one of them has a mission, and they exist separately to handle different processes at different levels. It is an excellent example of **a layered architecture**.

Layered architecture

- Used to model the **interfacing of sub-systems**.
- Organizes the system into a set of layers (or abstract machines) each of which **provide a set of services**.
- Supports **the incremental development** of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

A generic layered architecture

User interface

User interface management
Authentication and authorization

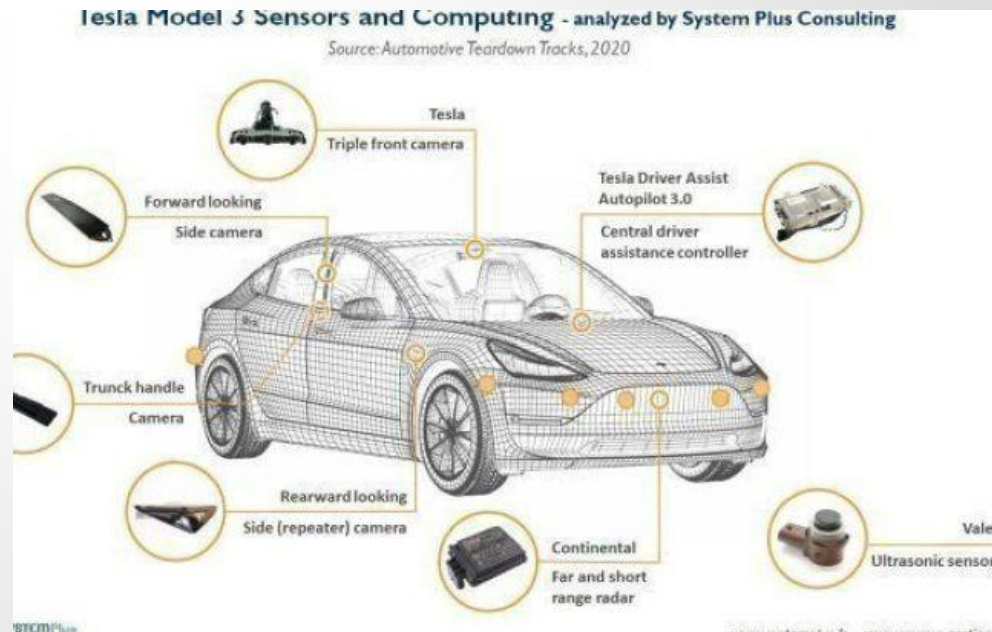
Core business logic/application functionality
System utilities

System support (OS, database etc.)

Layered architecture

Real-World Example

- Tesla Control Module (tesla.com): Control system for self-driving vehicles
- Waymo (waymo.com)



Case-Study: Solving a Business Problem With Layered Architecture

Amaze is a project management software company.

Their product is sold globally with a monthly pay-per-user model and widely known among the project management community for being easy to use and able to operate on many different devices (PCs, Notebooks, laptops, tablets, iPhones, iPads, and Android phones).

Case-Study: Solving a Business Problem With Layered Architecture

What's the Business Problem?

The business problem is very straightforward: Amaze must work on any popular device on the market and be **able to support future devices**.

There must be **only one version** of the software for all devices. No special cases, no exceptions allowed.

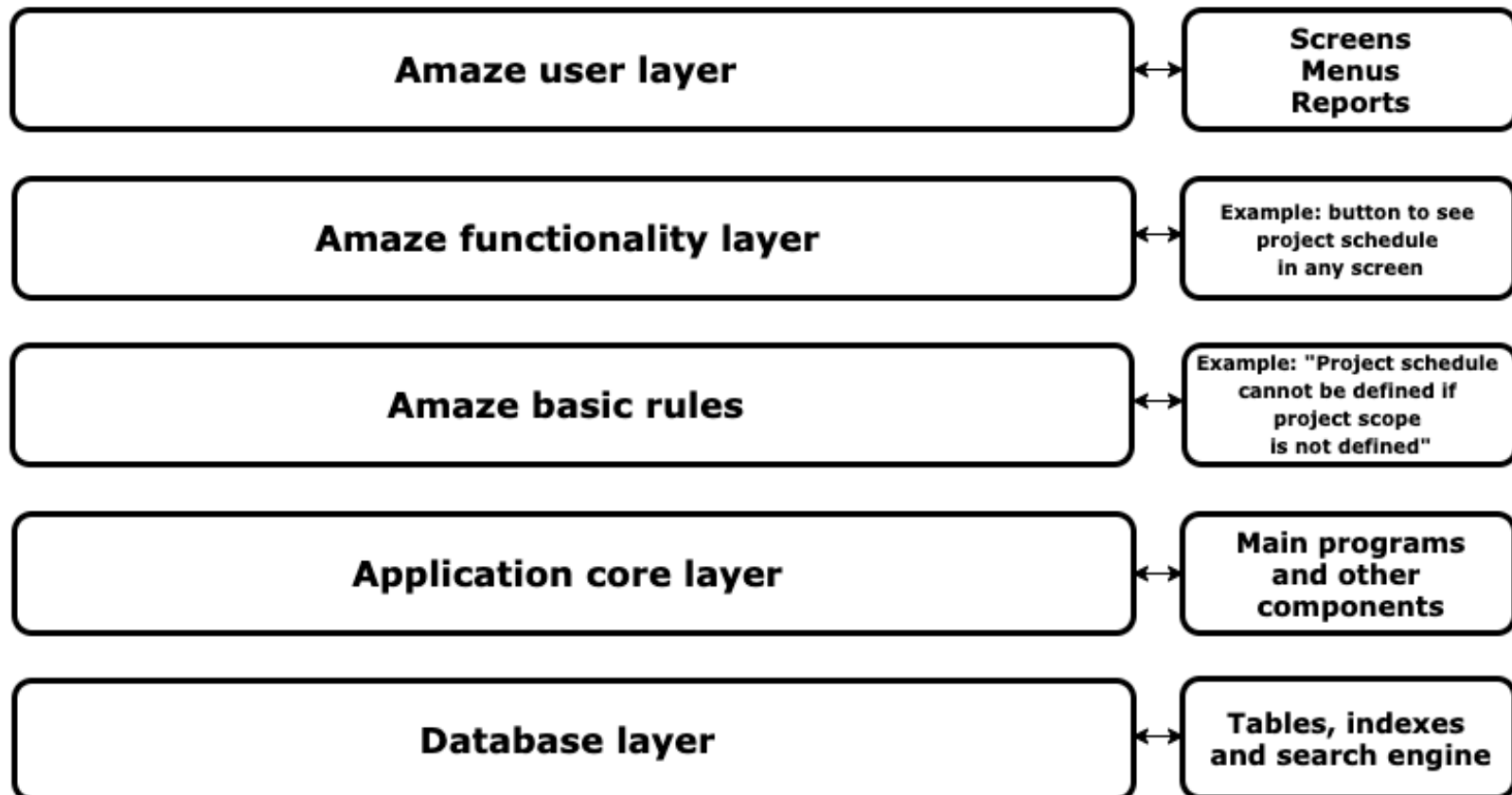
So to sum up:

- We know that **users have different devices**.
- There must be **only one software application** because the company wants to have low software maintenance costs.
- When **new device launches**, we do not want to change the whole software product.

Case-Study: Solving a Business Problem With Layered Architecture

What's the Solution?

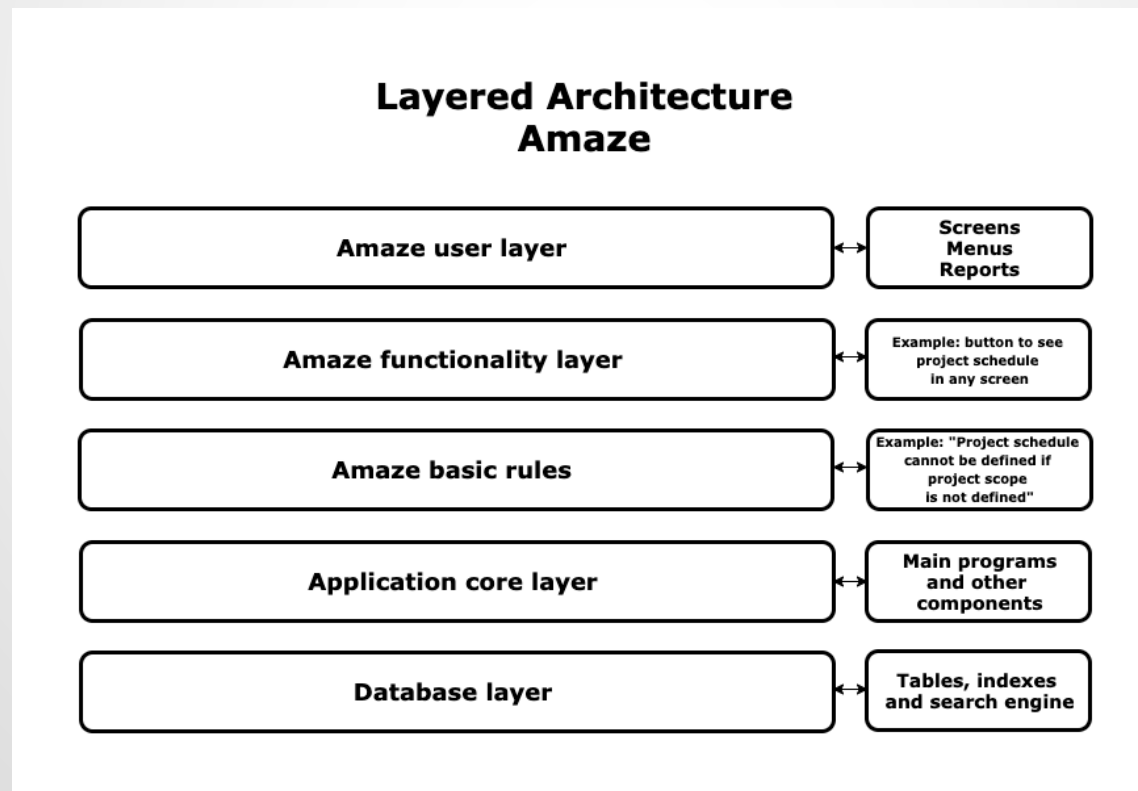
Layered Architecture Amaze



Case-Study: Solving a Business Problem With Layered Architecture

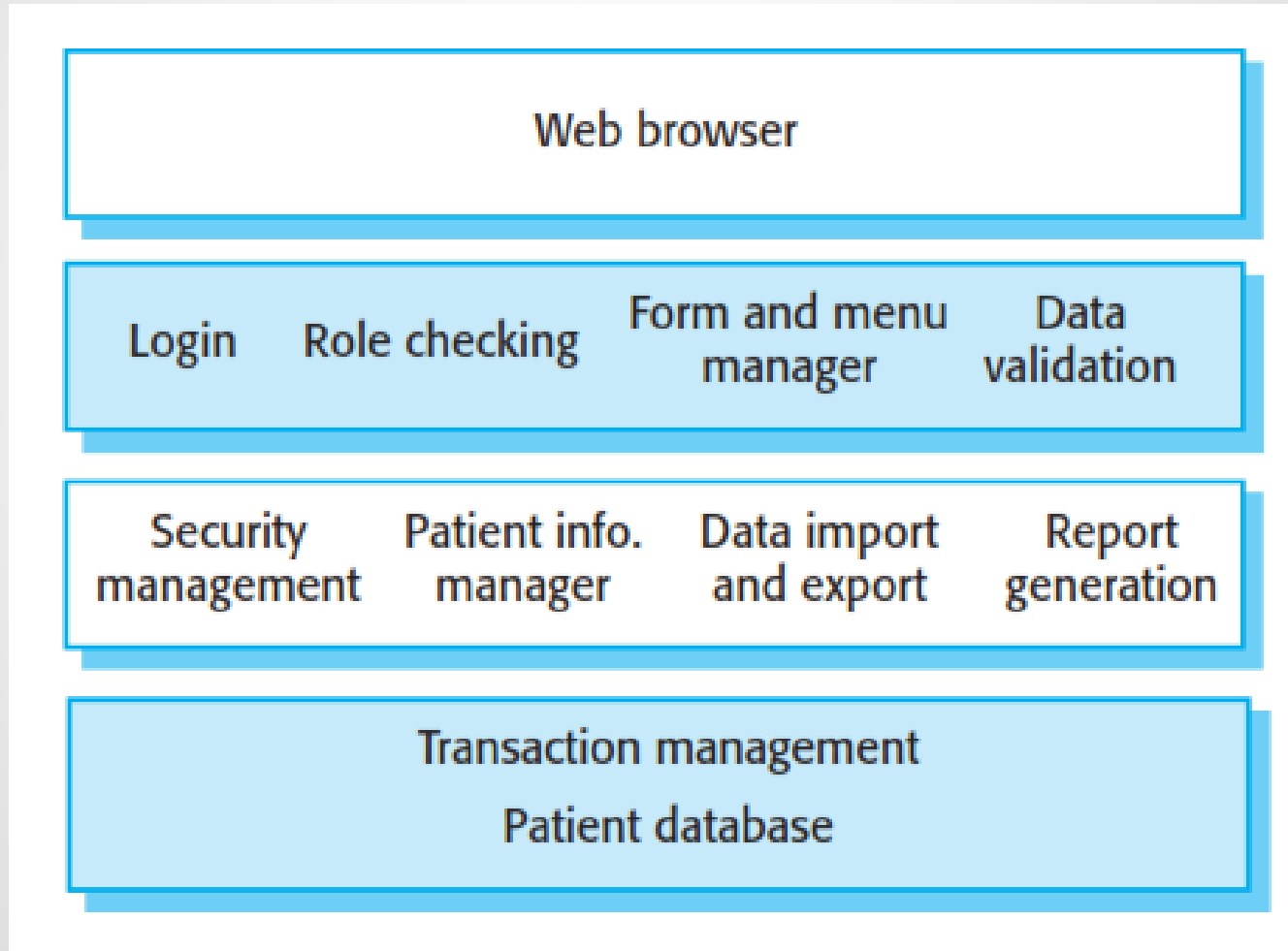
In our case, the **Amaze basic rules layer** is **critical**. This layer contains rules that determine the **behaviour of the whole application**, such as, "You can create a project schedule only if the project scope is defined."

The product's intelligence is in this layer: all special features that come from decades of experience in projects are developed there. The **application core layer** will be the most significant part of the application code.



The architecture of the Mentcare system

This system maintains and manages details of patients who are consulting specialist doctors about mental health problems.



The architecture of the Mentcare system

1. The top layer is a **browser-based user interface**.
2. The second layer provides the **user interface functionality** that is delivered through the web browser. It includes components to allow users to **log-in** to the system and checking components that ensure that the operations they use are allowed by their role. This layer includes form and menu management components that present information to users, and data validation components that check information consistency.
3. The third layer implements the functionality of the system and provides components that implement **system security**, patient information creation and updating, import and export of patient data from other databases, and report generators that create management reports.
4. Finally, the lowest layer, which is built using a **commercial database management system**, provides transaction management and persistent data storage.

The architecture of the iLearn system

Browser-based user interface

iLearn app

Configuration services

Group
management

Application
management

Identity
management

Application services

Email Messaging Video conferencing Newspaper archive
Word processing Simulation Video storage Resource finder
Spreadsheet Virtual learning environment History archive

Utility services

Authentication
User storage

Logging and monitoring
Application storage

Interfacing
Search

Layered architecture

There are several **advantages** to using layered architecture:

- **Layers are autonomous**: A group of changes in one layer does not affect the others. This is good because we can increase the functionality of a layer, for example, making an application that works only on PCs to work on phones and tablets, without having to rewrite the whole application.
- Layers allow **better system customization**.

There are also a few key **disadvantages**:

- Layers make an application more difficult to maintain. **Each change requires analysis**.
- Layers may affect application performance because they create overhead in execution: each layer in the upper levels must connect to those in the lower levels for each operation in the system.

Home Task on MVC

Is MVC different from a 3 layered architecture?

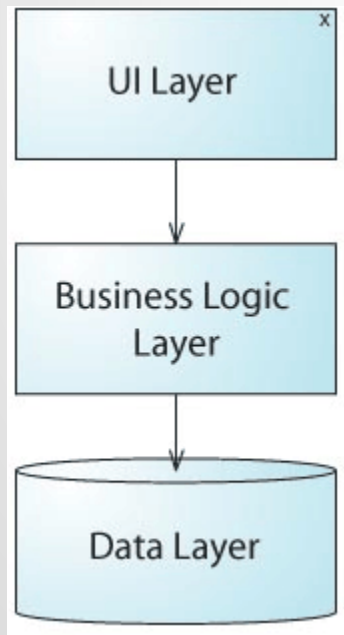


Home Task on MVC

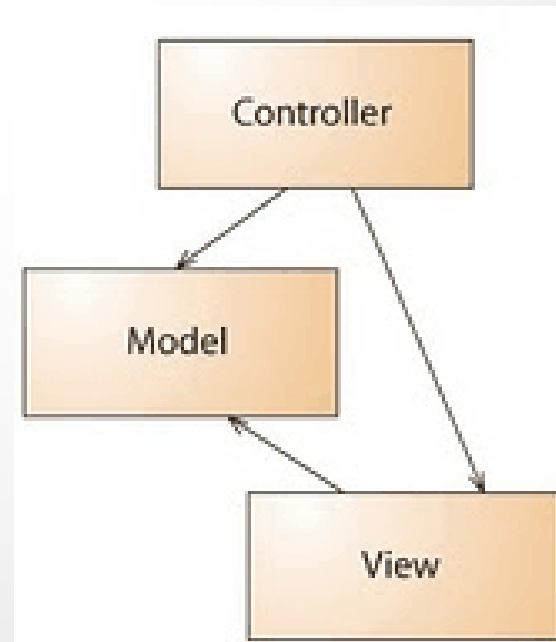
Is MVC different from a 3 layered architecture?

In 3-layer architecture

- 3-layer architecture separates the application into 3 components which consists of **Presentation Layer Business Layer and Data Access Layer**.
- In 3-layer architecture, **user interacts with the Presentation layer**.
- 3-layer is a **linear** architecture.



3-layer



MVC

Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.
- **Centralised** control
 - One sub-system has overall responsibility for control and starts and stops other sub-systems.
- **Event-based** control
 - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.
 - An event-driven architecture uses **events to trigger and communicate between decoupled services** and is common in modern applications built with micro services.
 - An event is a **change in state**, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).

Architectural Design : Control styles



Control styles

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.
- **Centralised** control
 - One sub-system has overall responsibility for control and starts and stops other sub-systems.
- **Event-based** control
 - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.
 - An event-driven architecture uses **events to trigger and communicate between decoupled services** and is common in modern applications built with micro services.
 - An event is a **change in state**, or an update, like an item being placed in a shopping cart on an e-commerce website. Events can either carry the state (the item purchased, its price, and a delivery address) or events can be identifiers (a notification that an order was shipped).

- There are two types of components –

1. Central Data

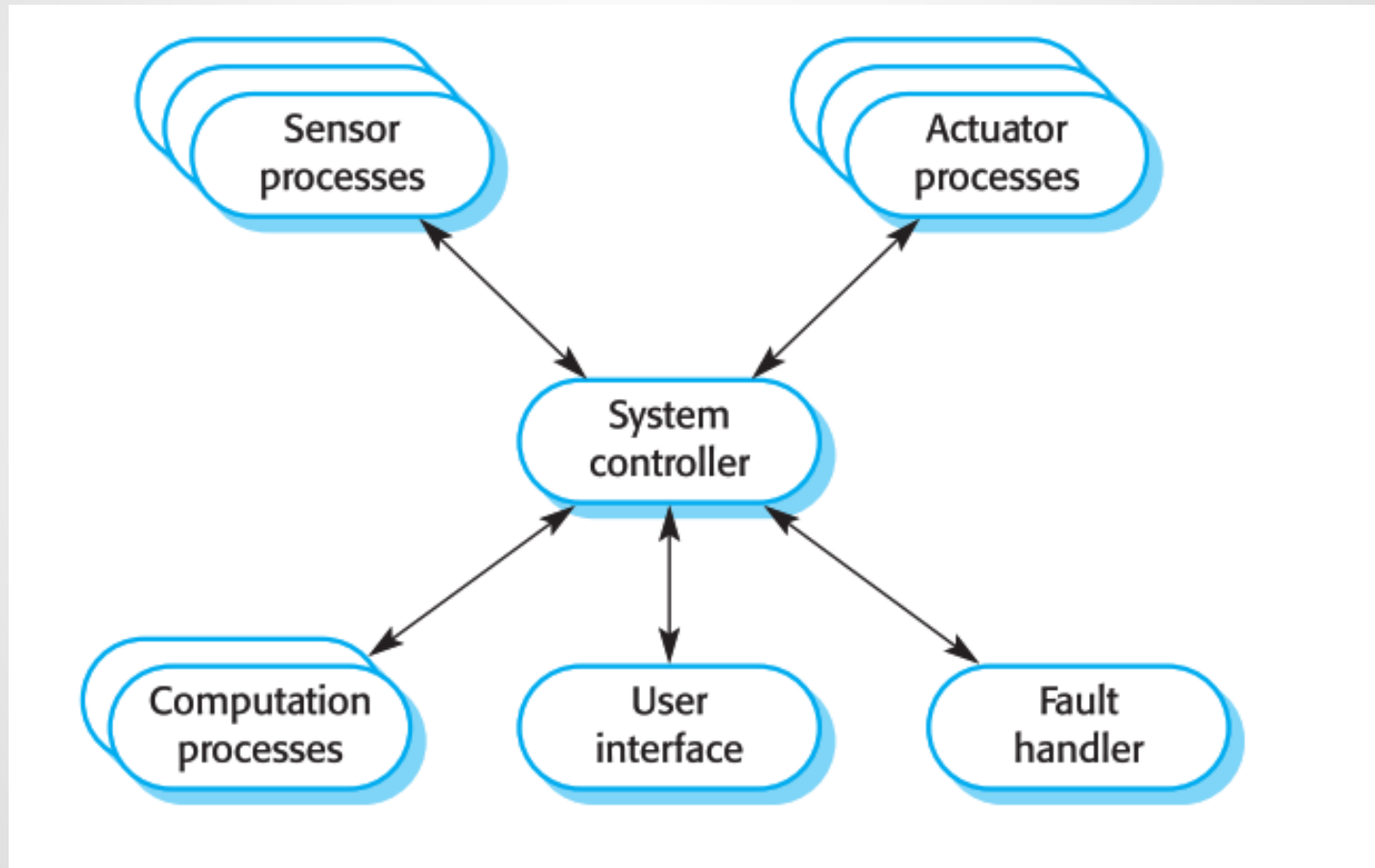
- Central data provides permanent data storage.
- Central data represents the current state.

2. Data Accessor

- Data accessor is a collection of independent components.
- It operates on the central data store, performs computations and displays the results.
- Communication can be done between the data accessors is only through the data store.

Centralised control

A centralized control model for a real-time system



The repository model

A repository architecture consists of a **central data structure** (often a database) and a collection of independent components which operate on the central data structure.

Sub-systems must **exchange data**. This may be done in two ways:

- Shared data is held in a **central database** or repository and may be accessed by all sub-systems;
- Each sub-system maintains its **own database** and passes data explicitly to other sub-systems.

When large amounts of data are to be shared, the repository model of sharing is most commonly used.

The repository model

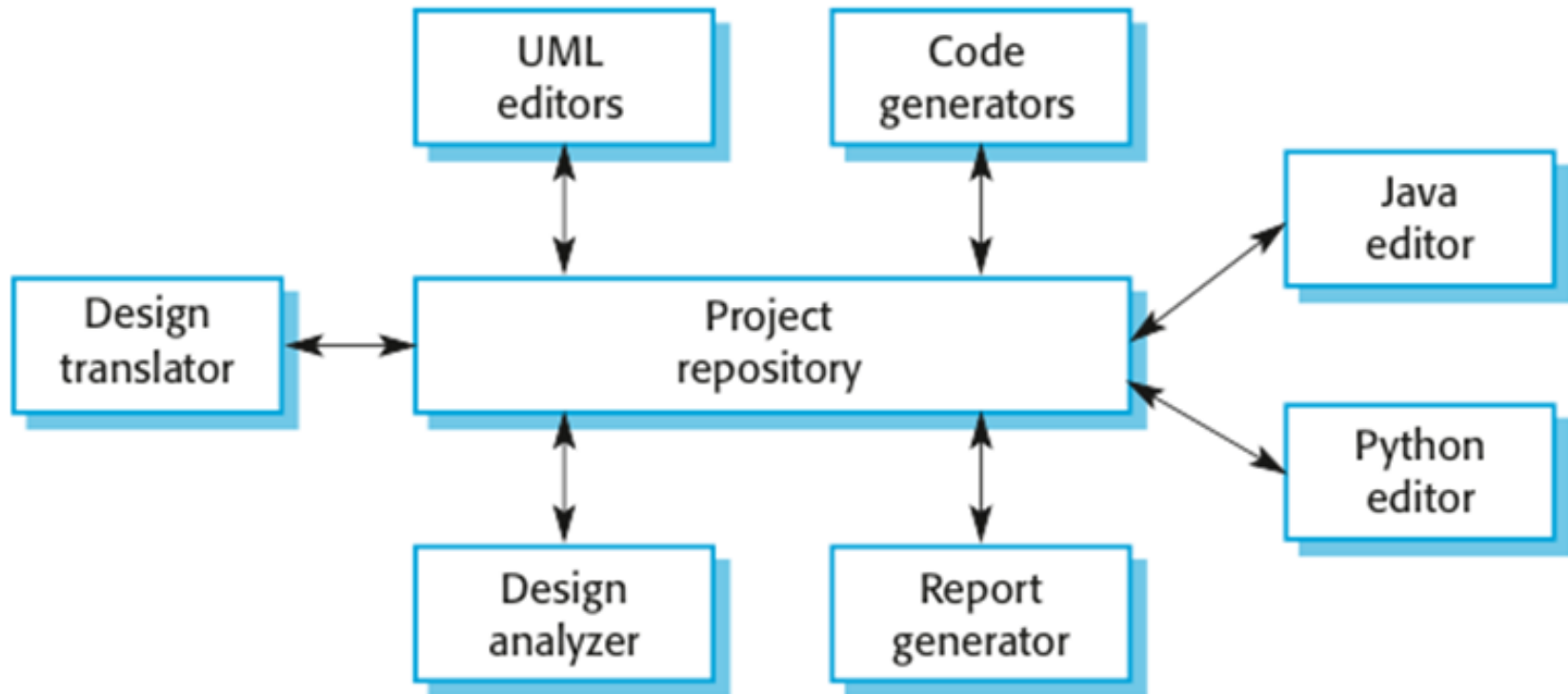
Advantages

- Efficient way to share **large amounts** of data;
- Sub-systems need not be concerned with how data is **produced**
- **Centralised** management e.g. backup, security, etc.
- **Sharing** model is published as the repository schema.

Disadvantages

- Sub-systems must agree on a repository data model. Inevitably a **compromise**;
- **Data evolution** is difficult and expensive;
- No scope for **specific** management policies;
- Difficult to **distribute** efficiently.

A repository architecture for an IDE



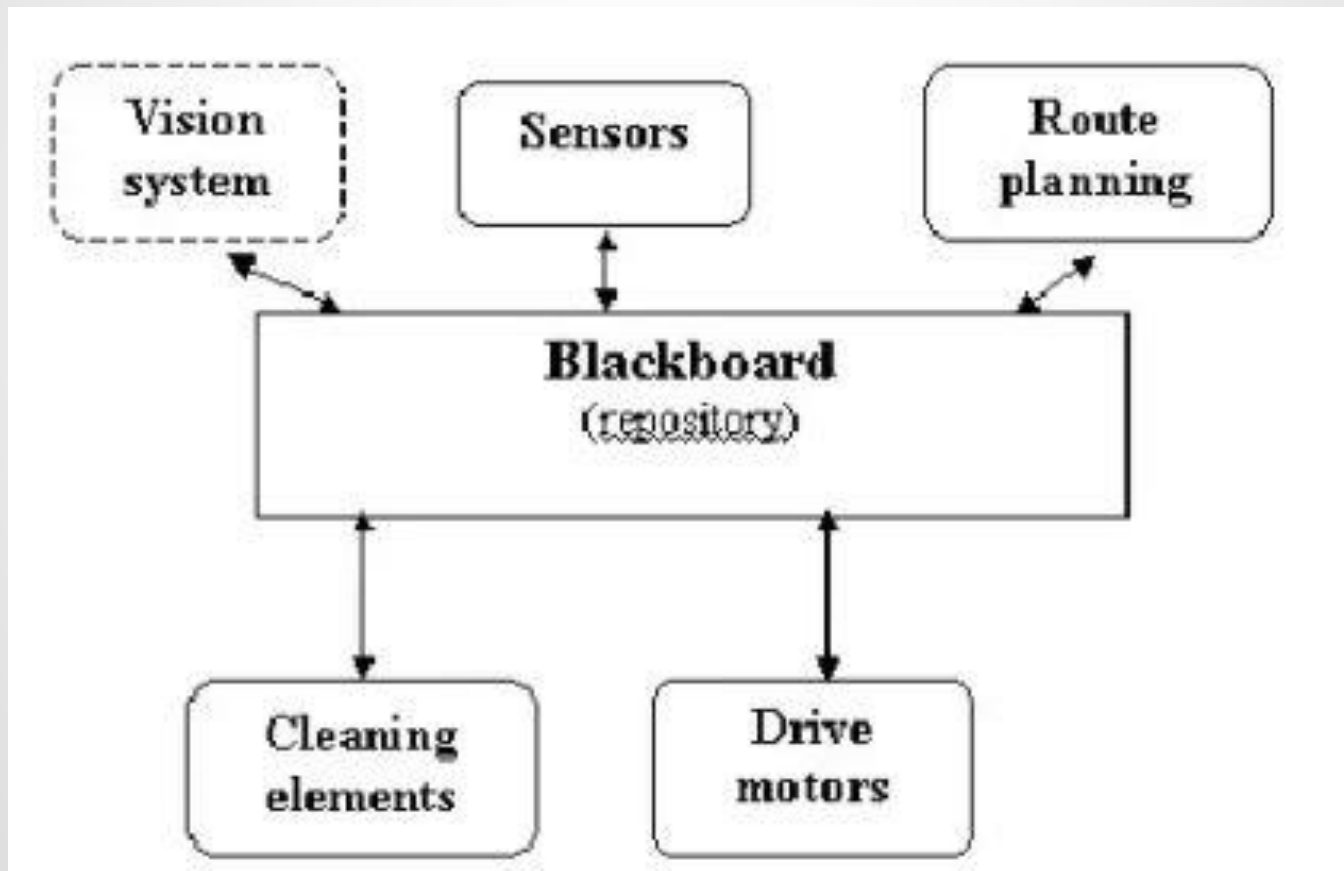
A repository architecture

Given reasons for your answer, suggest an appropriate structural model for the following systems:

- A **robot floor-cleaner** (standalone system) that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.

A repository architecture

- A robot floor-cleaner that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.



A repository architecture

- A robot floor-cleaner that is intended to clean relatively clear spaces such as corridors. The cleaner must be able to sense walls and other obstructions.
- The most appropriate model is a **repository model**, with each of the subsystems (wall and obstacle sensors, path planning, vision (perhaps), etc.) placing information in the repository for other subsystems to use.
- Robotic applications are in the realm of Artificial intelligence, and for the AI systems such as this, a special kind of repository called a **blackboard** (where the presence of data activates particular subsystems) is normally used

Plug-In Architecture

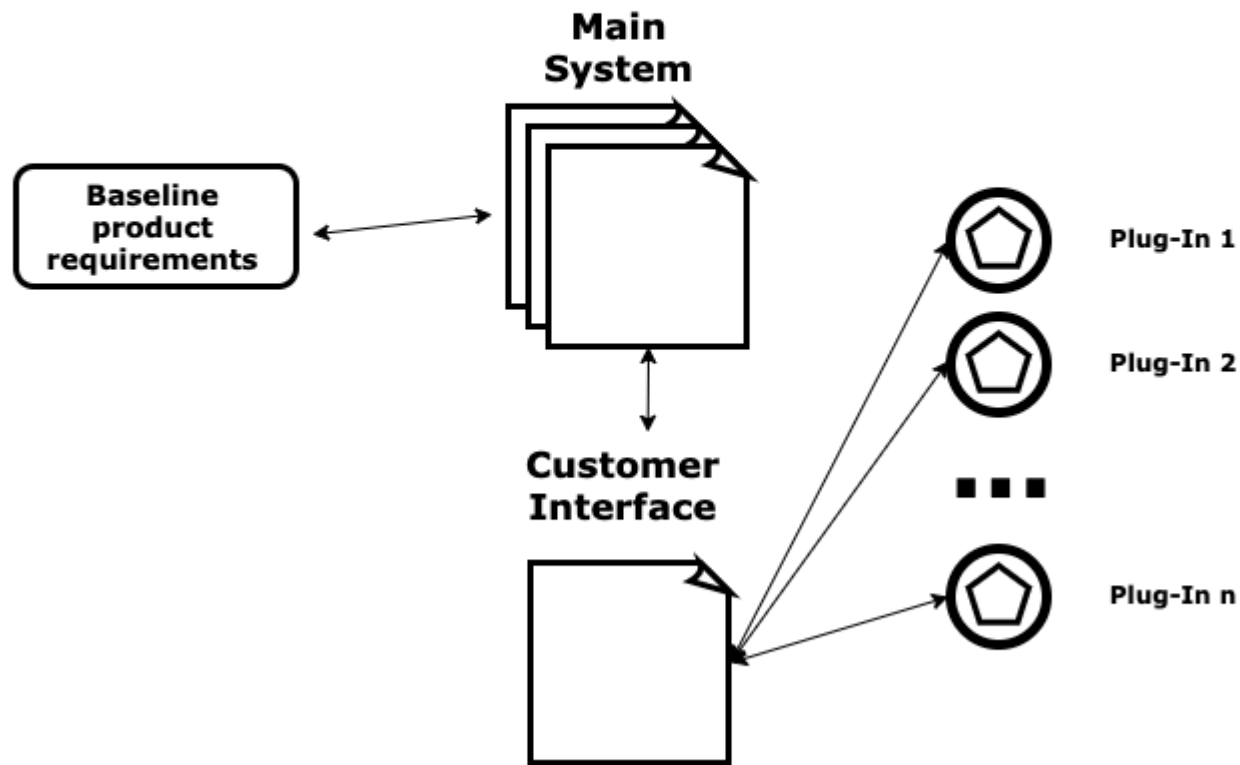
- ✧ Many people are worried about privacy, whether it's protecting their banking data, passwords, or making sure no one is reading their messages or emails. One way to ensure privacy protection is via encryption, and many software programs do so.
- ✧ Most encryption applications are installed in our browser (Chrome, Mozilla, Explorer, etc.) as an add-on: a module that works on top of the browser for any communication and encrypts it. This application is called a **plug-in**.

Plug-In Architecture

- ✧ A plugin architecture is an architecture that will **call external code at certain points without knowing all the details of that code in advance.**
- ✧ A plug-in is a **bundle** that **adds functionality** to an application, called the **host application**, through some well-defined architecture for extensibility. This **allows third-party developers** to add functionality to an application without having access to the source code.
- ✧ This also allows users to add new features to an application just by installing a new bundle in the appropriate folder.

Plug-In Architecture

Plug-In Architecture High Level Diagram



Plug-In Architecture

A standard plug-in architecture has four parts:

- **Baseline product requirements:** This is the set of minimal requirements that define the application, determined at the beginning of the development process when an initial set of features were included in the product.
- **Main system:** This is the application we plug the plug-ins to. The main system needs to provide a way to integrate plug-ins, and therefore will slightly vary the original baseline product to ensure compatibility.
- **Customer Interface:** This is the module that interacts with the customer, for example, a web browser (Chrome, Mozilla, etc.).
- **Plug-ins:** These are add-ons that enlarge the minimal requirements of the application and give it extra functionality.

Plug-In Architecture

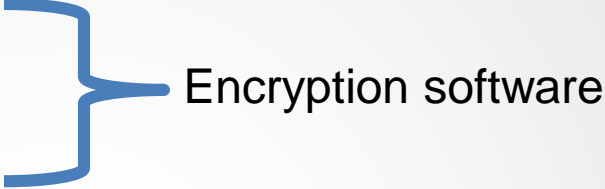
There are several **advantages** to using plug-in architecture:

- The plug-in architecture is the **best way to add particular functionality** to a system that was not initially designed to support it.
- This architecture removes limits on the amount of functionality an application can have. We can add infinite plug-ins (The Chrome browser has hundreds of plug-ins, called extensions).
- No rewriting the system.

There are also a few key **disadvantages**:

- Plug-ins **can be a source of viruses and attacks from external players**.
- Having many plug-ins in an application may affect its performance.
- Plug-ins frequently **crash with each other** and produce malfunctions in the main system.

Real-World Example

- SendSafely (sendsafely.com)
 - Mailvelope (mailvelope.com)
- 
- Encryption software
- Rapportive (rapportive.com) connects LinkedIn to your browser.
 - Trello (trello.com) connects Trello to your browser.

Case-Study: Solving a Business Problem With Plug-In Architecture

Pacific is a retail website that is very successful in Southeast Asia. Here are some **quick facts**:

- The company sells **more than 1,64,000 items** on its website, mainly to Southeast Asian customers.
- About 112 items are sold every **second** on the website, 24 hours a day, 365 days a year.
- There are heavy users of the site: users that buy many items at once, and they need to do it quickly.

What's the Business Problem?

Each time a heavy user wants to buy many items at once, he or she must **open separate browser windows** for each item, causing confusion and frustration, which can stop the user from making a purchase.

For example, some users select ten items, but in the end they buy only six because they go back and forth from the shopping cart to the item pages.

How can this problem be solved?

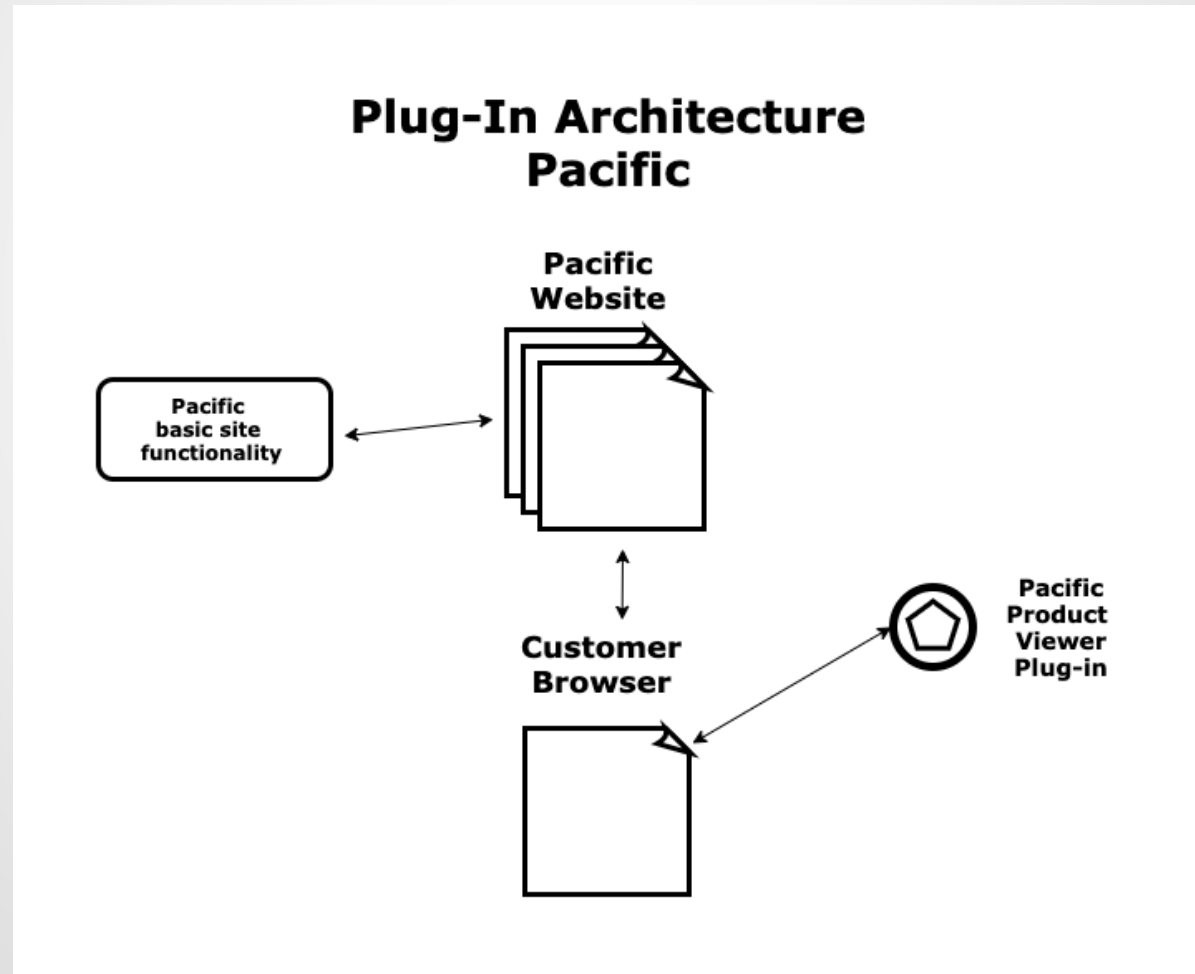
Pacific Product Viewer, a plug-in for any browser

Pacific has developed a plug-in for Chrome called Pacific Product Viewer: when the user installs the plug-in, a list of interesting items is shown in a pop-up window, and there is no need to open multiple tabs.

This window allows the user to see many items without having to switch tabs. It is usually located in the upper-right corner of the screen.

Case-Study: Solving a Business Problem With Plug-In Architecture

- **Pacific website:** It is the main Pacific website, developed according to a minimal set of requirements.
- **Customer browser:** Chrome, Mozilla, Explorer, etc.
- **Pacific Product Viewer Plug-in:** The piece of software that was developed as an add-on for extra functionality.



What is the technical difference between an API and a plugin?

API: the initials stand for Application Programming Interface. It refers to a set of routines, protocols and tools for building software and applications. An API basically defines how a component interacts with a system, facilitating the communication between them.

Plugin: also called an **extension**, a plugin is a software component that makes it possible **to modify an existing computer program or platform**, for instance, adding new features to it. Such modification does not usually alter the design of the system, which often has a Plugin API defining how the extensions can interact with the whole system.

Plugin is an extension added to your code - so as to bypass : or perform something specific without effecting your existing code !!

Distributed Systems Architectures



Distributed Architecture

- A distributed system is "**a collection of independent computers that appears to the user as a single coherent system.**" Information processing is distributed over several computers rather than confined to a single machine.
- Virtually **all large computer-based systems** are now distributed systems.
- **Distributed software engineering** is therefore very important for enterprise computing systems.

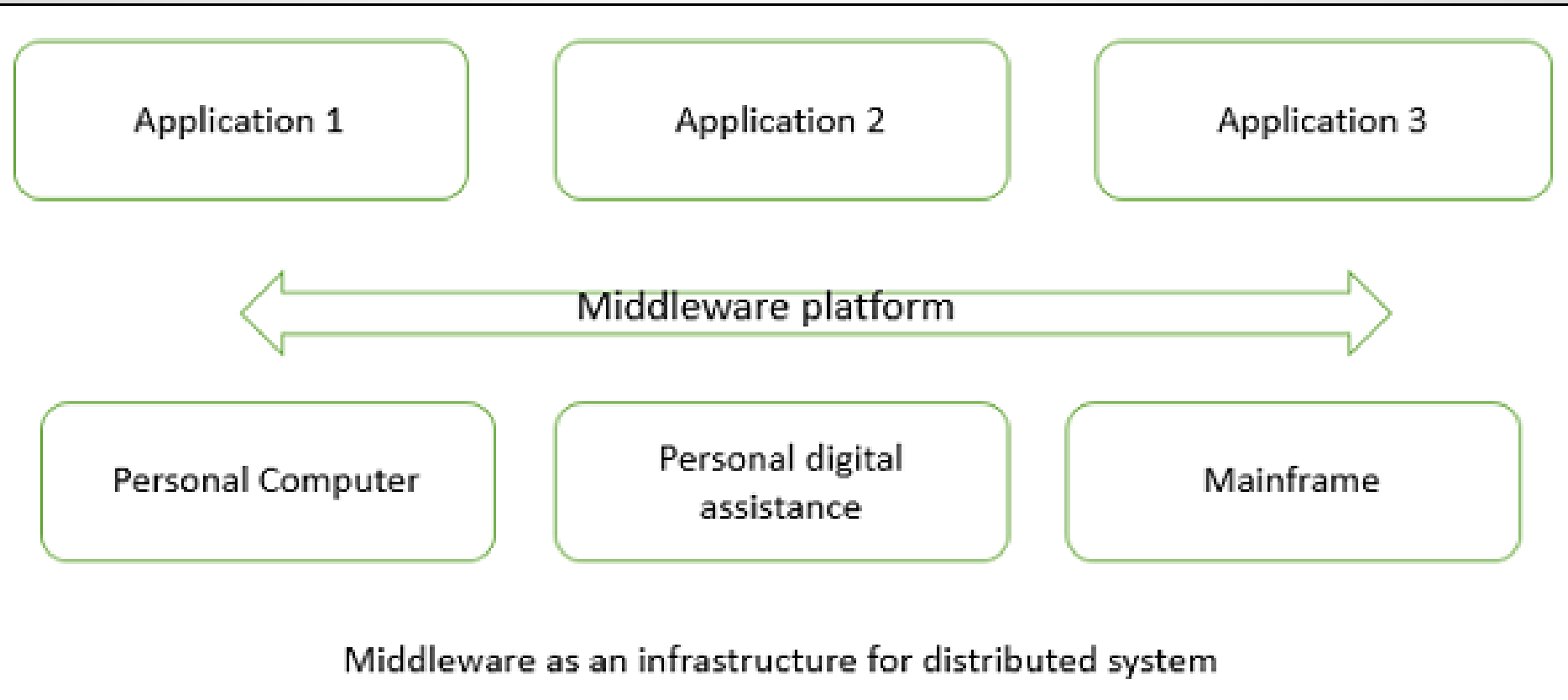
Distributed Architecture

- A distributed system can be demonstrated by the client-server architecture which forms the base for multi-tier architectures; alternatives are the broker architecture such as CORBA, and the Service-Oriented Architecture (SOA).
- There are several technology frameworks to support distributed architectures, including .NET, J2EE, CORBA, .NET Web services, AXIS Java Web services, and Globus Grid services.

Distributed Architecture

- **Middleware** is an infrastructure that appropriately supports the development and execution of distributed applications.
- It provides a **buffer** between the **applications and the network**.
- It sits in the middle of system and **manages or supports the different components** of a distributed system.
- Examples are transaction processing monitors, data convertors and communication controllers etc.

Distributed System



Distributed System

Advantages

- **Resource sharing** – Sharing of hardware and software resources.
- **Openness** – Flexibility of using hardware and software of different vendors.
- **Concurrency** – Concurrent processing to enhance performance.
- **Scalability** – Increased throughput by adding new resources.
- **Fault tolerance** – The ability to continue in operation after a fault has occurred.

Disadvantages

- **Complexity** – They are more complex than centralized systems.
- **Security** – More susceptible to external attack.
- **Manageability** – More effort required for system management.
- **Unpredictability** – Unpredictable responses depending on the system organization and network load.

Centralized System vs. Distributed System

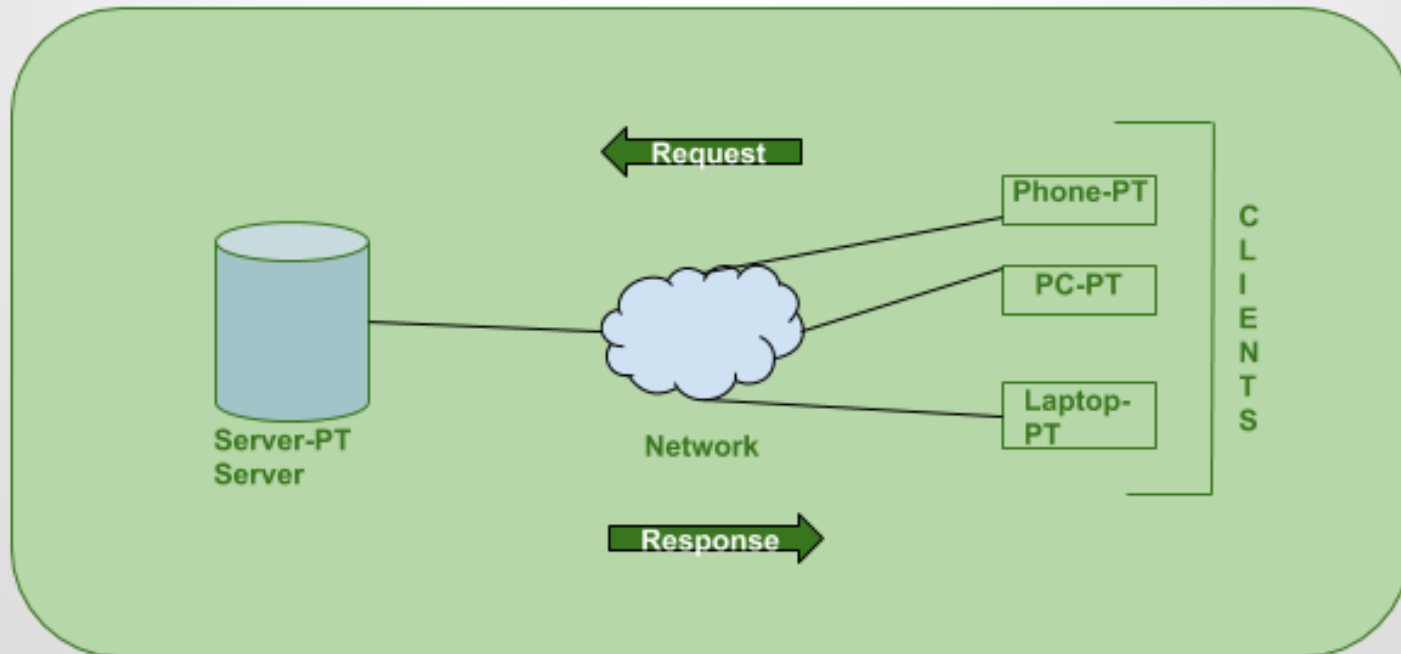
Criteria	Centralized system	Distributed System
Economics	Low	High
Availability	Low	High
Complexity	Low	High
Consistency	Simple	High
Scalability	Poor	Good
Technology	Homogeneous	Heterogeneous
Security	High	Low

Client-Server Architecture

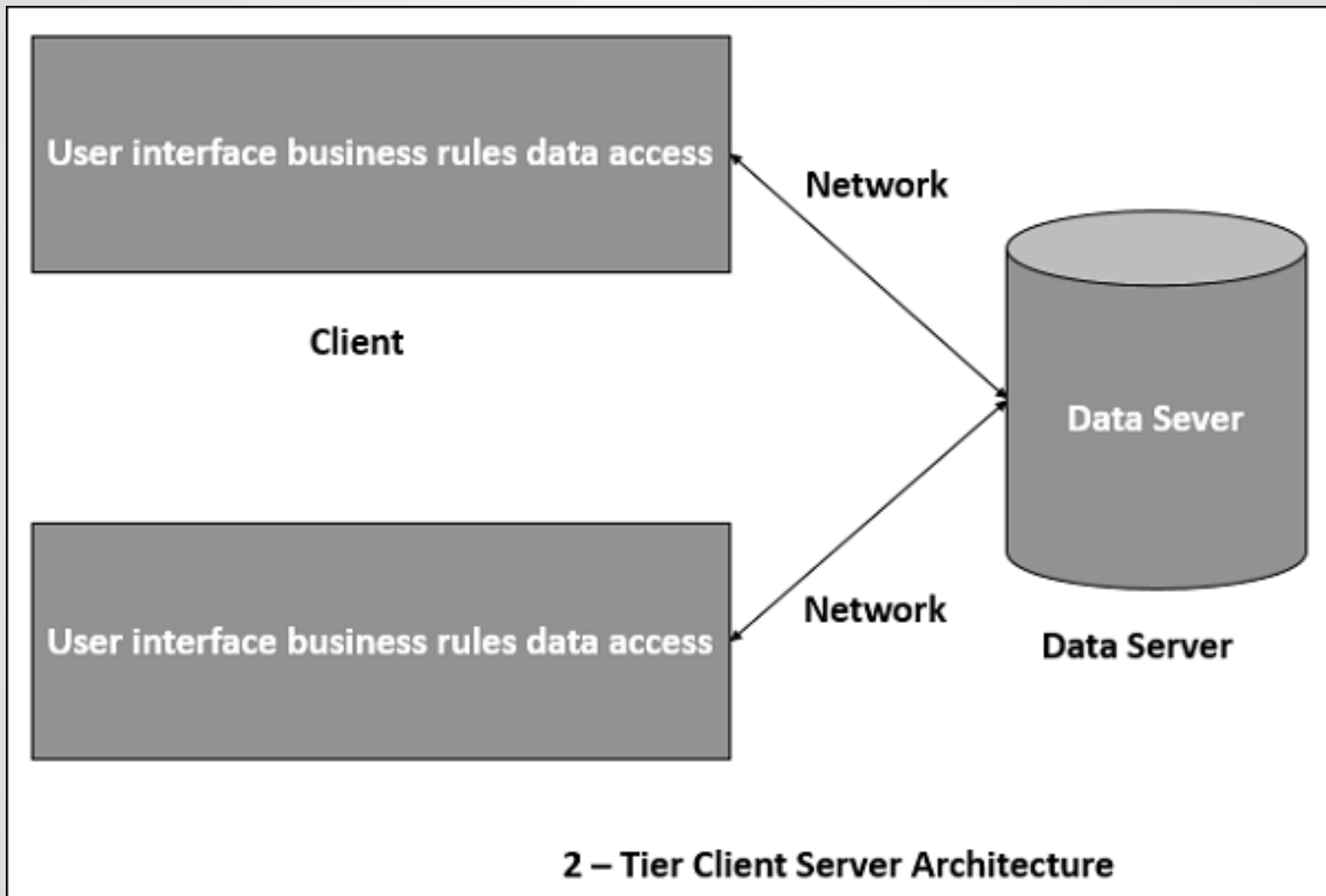
The client-server architecture is the most common distributed system architecture which decomposes the system into two major subsystems or logical processes –

Client – This is the first process that issues a **request to** the second process i.e. the server.

Server – This is the second process that **receives the request**, carries it out, and sends a reply to the client.



Client-Server Architecture

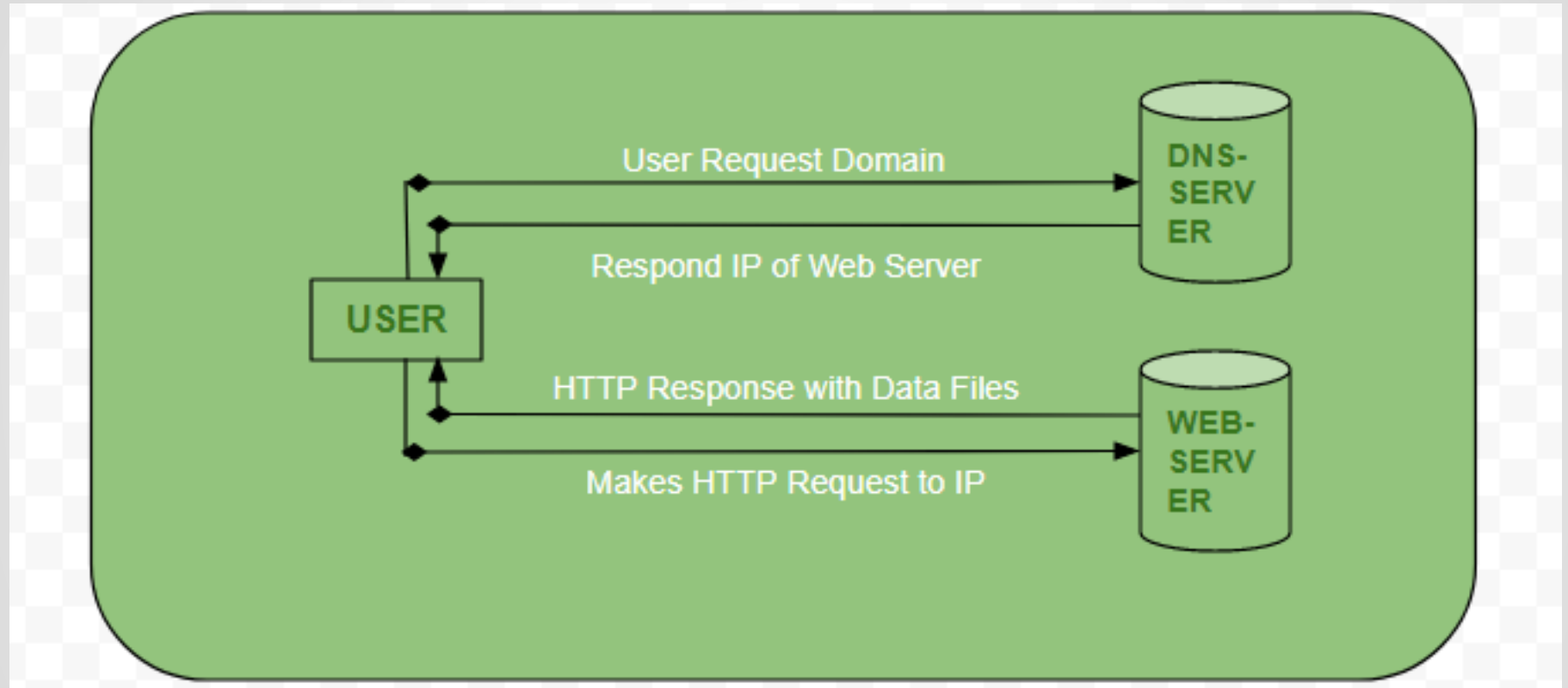


How the Client-Server Model works ?

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.

Client-Server Architecture

How the Client-Server Model works ?



Client-Server Architecture

Examples of Client Server Architecture

Mail Servers – Email servers help to send and receive all emails. Some softwares are run on the mail server which allow administrator to create and handle all email accounts for any domain that is hosted on the server.

Mail servers use the some protocols for sending and receiving emails such as SMTP, IMAP, and POP3. **SMTP protocol** helps to fire messages and manages all outgoing email requests. **IMAP** and **POP3** help to receive all messages and handle all incoming mails.

File Servers – File server is dedicated systems that allow users to access for all files. It works like as centralized file storage location, and it can be accessed by several terminal systems.

Client-Server Architecture

Advantages

- Distribution** of data is straightforward;
- Makes effective use of **networked** systems. May require cheaper hardware;
- Easy to add **new servers or upgrade existing servers**.

Disadvantages

- No shared data model so sub-systems use **different data organisation**. **Data interchange** may be **inefficient**;
- Redundant management** in each server;
- No central register of names and services** - it may be hard to find out what servers and services are available.

Here are some real-life situations that this would be useful for:

- Enterprise resource planning (ERP) system
- SAP (sap.com).
- Oracle Business Suite (oracle.com).
- Microsoft Dynamics (microsoft.com).
- Infor (infor.com).
- Epicor (epicor.com).

Case-Study: Solving a Business Problem With Client-Server Architecture

IrisGold is a gold mining company. Here are some **quick facts**:

- IrisGold operates on **three continents**, with more than **21,000 employees**.
- The company's mines are mostly **located in remote places** like the Amazonas in Brazil, the Andes mountain range, the Ural mountains in Russia, and eastern South Africa.
- The company is selecting an Enterprise Resource Planning (ERP) system package. How does this inform your decision?

What's the Business Problem?

IrisGold wants to deploy and operate its new ERP package securely. But they have **two main constraints**:

- Its users are in **remote places** in the world with different kinds of devices (laptops, notebooks, phones, tablets).
- **Client devices must be light**: they must be a simple notebook computer, tablet, or phone with few processing and storage capabilities, **able to communicate to a remote server**.

Case-Study: Solving a Business Problem With Client-Server Architecture

Let's start with the facts!

1. We know that users are scattered over the world and use different devices that need to be lightweight.
2. We know that users have deficient infrastructure capabilities and work in remote places.
3. Clients need only to be able to connect with a central server.
4. The ERP system installed in the central server must cover all the business rules. In essence, it must manage all modules for all countries internally, and it must answer client requests by communicating with a central database.

Case-Study: Solving a Business Problem With Client-Server Architecture

So to sum up:

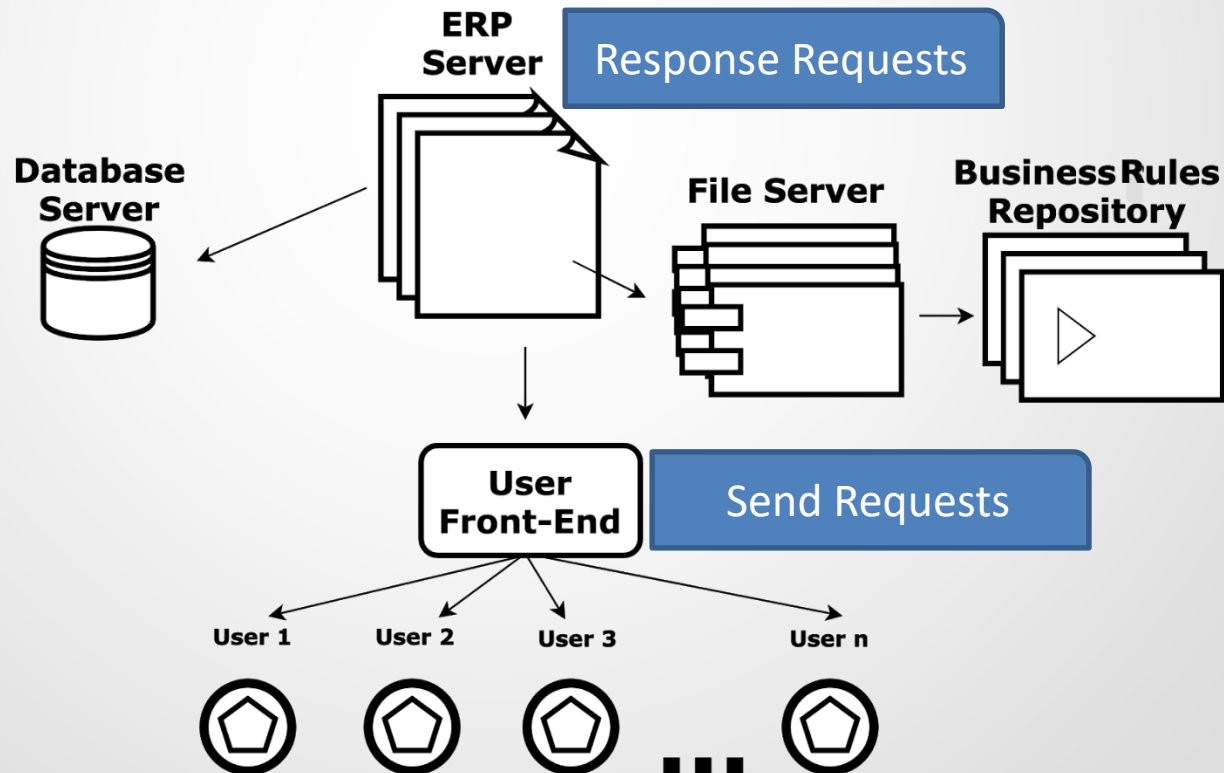
1. We know that users are scattered over the world and use different devices that need to be lightweight.
2. We know that users have deficient infrastructure capabilities and work in remote places.
3. Clients need only to be able to connect with a central server.
4. The ERP system installed in the central server must cover all the business rules. In essence, it must manage all modules for all countries internally, and it must answer client requests by communicating with a central database.

Case-Study: Solving a Business Problem With Client-Server Architecture

What's the Solution?

All heavy processing is done at **IrisGold's headquarters**. Clients are thin; that's why they are called "**thin clients**." They do not process or store large amounts of data. They just speak with the server.

Client-Server Architecture IrisGold



Case-Study: Solving a Business Problem With Client-Server Architecture

Front-End: This is the piece of software that interacts with ERP users, even if they are in different countries.

ERP server: This is the server where the ERP software is installed.

File server: The ERP server requests files from the file server to fulfill user requests. Examples: an invoice printed in PDF format, a report, a data file that the user needs. It is good practice to install a file server when the application has many users that read, update, or write files frequently.

Business rules repository: A repository of business procedures, methods, and regulations for every country (all different), to make the ERP work according to each country's needs. This repository is often separated from the software to make customizations more agile and secure. It is a good practice introduced by ERPs in the '90s that has spread over many kinds of non-ERP applications.

Database server: This server contains the tables, indexes, and data managed by the ERP system. Examples: customer table, provider table, invoice list, stock tables, product IDs, etc.

Sec. A and B

Class Test-03: 19-March-2023

Lecture: 12, 13

Class Test-04: 3-April-2023

Lecture:



Thanks to All