COSC 310: SOFTWARE ARCHITECTURE.

Dr. Gema Rodriguez-Perez

University of British Columbia

gema.rodriguezperez@ubc.ca

ARCHITECTURAL DESIGN

SOFTWARE DESIGN

- The ***design process*** determines *how* to construct a software system to meet the system requirements.

- Design involves determining:
  - the system architecture
  - the modules, classes, methods, and interfaces
  - the data structures and algorithms used
  - the interaction protocols/interfaces with users and other systems
  - implementation language

ARCHITECTURAL DESIGN AND AGILITY

Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system

- Critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them.
- The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components.
- An early stage of agile processes is to design an overall systems architecture.

ARCHITECTURAL DESIGN ISSUES

While Individual components implement the functional system requirements, the dominant influence on the **non-functional system characteristic** is the system's architecture.

- *Users*: How many simultaneous users must the system support? Scalable?
- *Distribution*: Are the users, data, or system components physically distributed?
- *Performance*: Are there stringent real-time or interactive performance requirements that must be satisfied?
- *Maintainability*: How will the software be maintained?
- *Security*: Must the software need to handle private data?

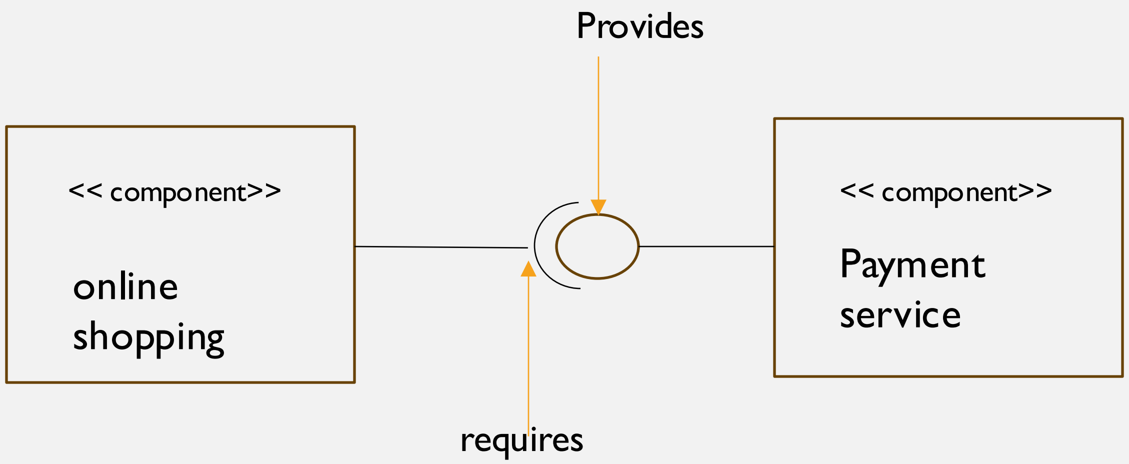
Where to start?? Systems in the same domain often have similar architectures that reflect domain concepts

HOW TO REPRESENT THE ARCHITECTURE?

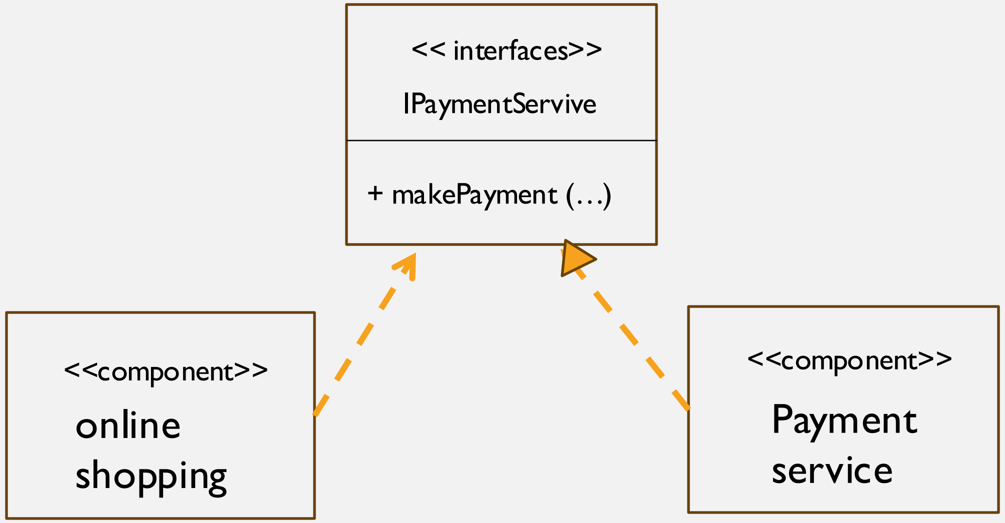
- UML Component diagram – It's a high level view of component and interfaces.
- What it is a component? – It's a piece of your system, a "unit of composition"
  - It contributes to the overall functionality of the system, essentially acting as an individual element within a larger structure
    - For example, in a web service, the online shopping is a component that depends on another component, the payment system. But both systems contributes to the overall functionality of the system.
  - A component can have explicit dependencies

UML COMPONENT DIAGRAM

- UML syntax



- Basic



- Detailed

ARCHITECTURAL PATTERNS

ARCHITECTURAL STYLES/PATTERNS

- They are good practices; they are generic styles that have been used a lot and have been proven to work.
- We know their benefits and limitations.
- The architecture of your system will have big implications on your quality requirements
- There is no perfect architecture, but some will help with safety, security, performance, availability, maintainability, etc.

ARCHITECTURE: MONOLITHIC

- **Monolithic design** an architecture in which all components of an application are tightly integrated and interdependent
  - Monolithic application is built as a single, unified unit, with a single codebase and a single deployment package
  - all components of the application are tightly coupled and communicate with each other directly
  - Any changes to one component of the application can potentially affect other components, making it more difficult to maintain and update the system
  - They can be useful for smaller applications or for teams that are just starting out and need to quickly build a prototype or proof-of-concept
    - However, as applications grow and become more complex, monolithic designs can become difficult to maintain and scale.

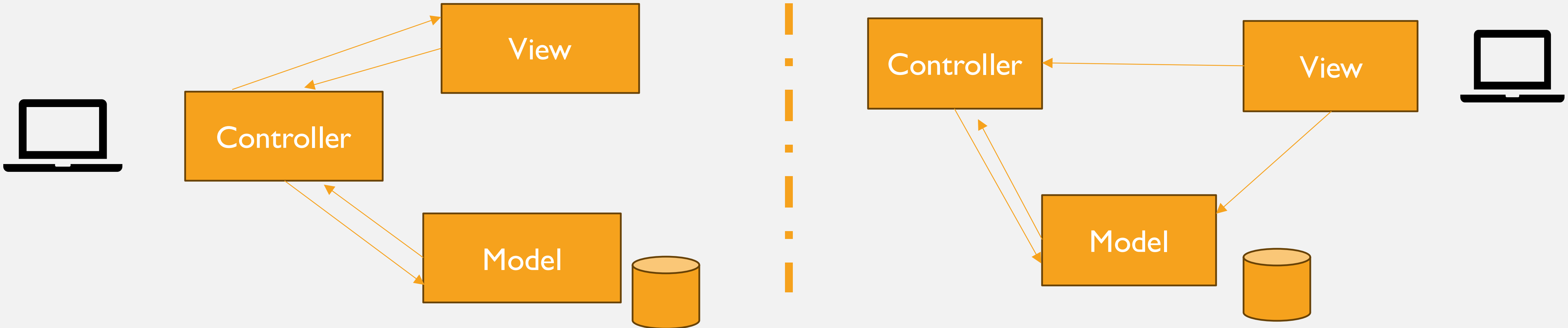
ARCHITECTURAL STYLES/PATTERNS

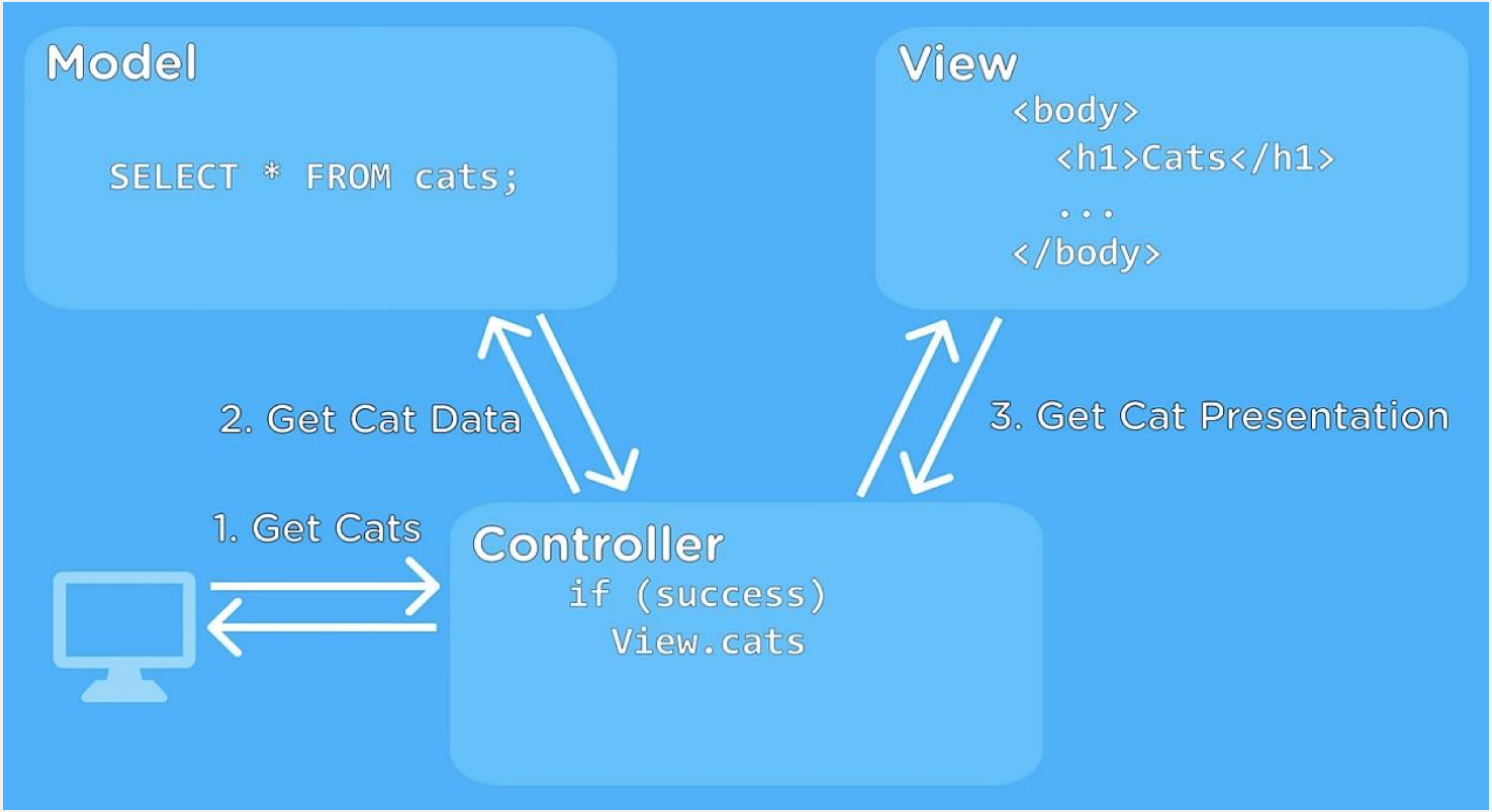
MVC (Model/View/Controller): The state of the data should be independent of the representation and the way to control it.

Model: Handle data logically so it basically deals with data.

View: Data Representation (UI)

Controller: Handles requests flow





Model

```
SELECT * FROM cats;
```

View

```
<body>
  <h1>Cats</h1>
  ...
</body>
```

2. Get Cat Data

3. Get Cat Presentation

1. Get Cats

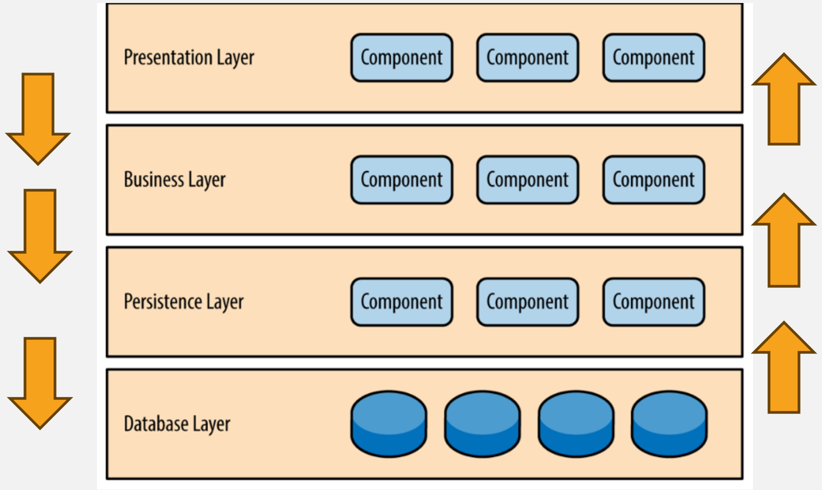
Controller

```
if (success)
  View.cats
```

ARCHITECTURAL STYLES/PATTERNS

Layered architecture: We have multiple layers, like Presentation (UI), Business logic, etc…

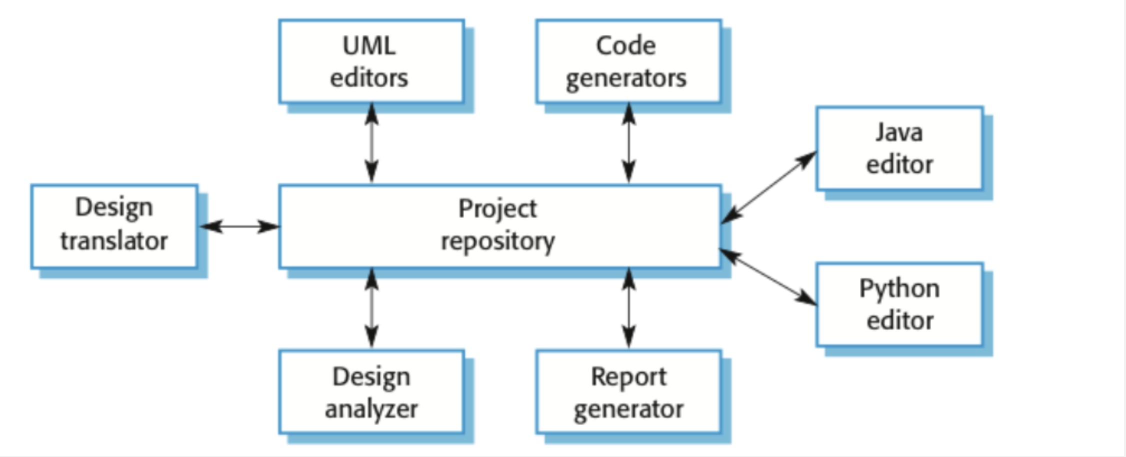
- The upper layers only talked to the layer below them. And the lower layers never called up, only reply. They are only reacting.
- Common why to organize teams.
- Advantages:
  - Security at several layers
  - Build on top
  - Replacement
- Limitations:
  - Performance
  - Difficult to implement in practice



ARCHITECTURAL STYLES/PATTERNS

Repository style: You have a central component (repository) and a number of component that access the repository.

- The repository contains all the data
- The other components do not talk to each other, they only access the main repo.
- Advantage:
  - Component do not have interfaces with other components
- Disadvantage
  - Central component have many interfaces
  - Failure of the central component



ARCHITECTURAL STYLES/PATTERNS

Client/Service architecture: You have some network and clients talk to the network and then servers send what the clients wants from the service.

It is distributed

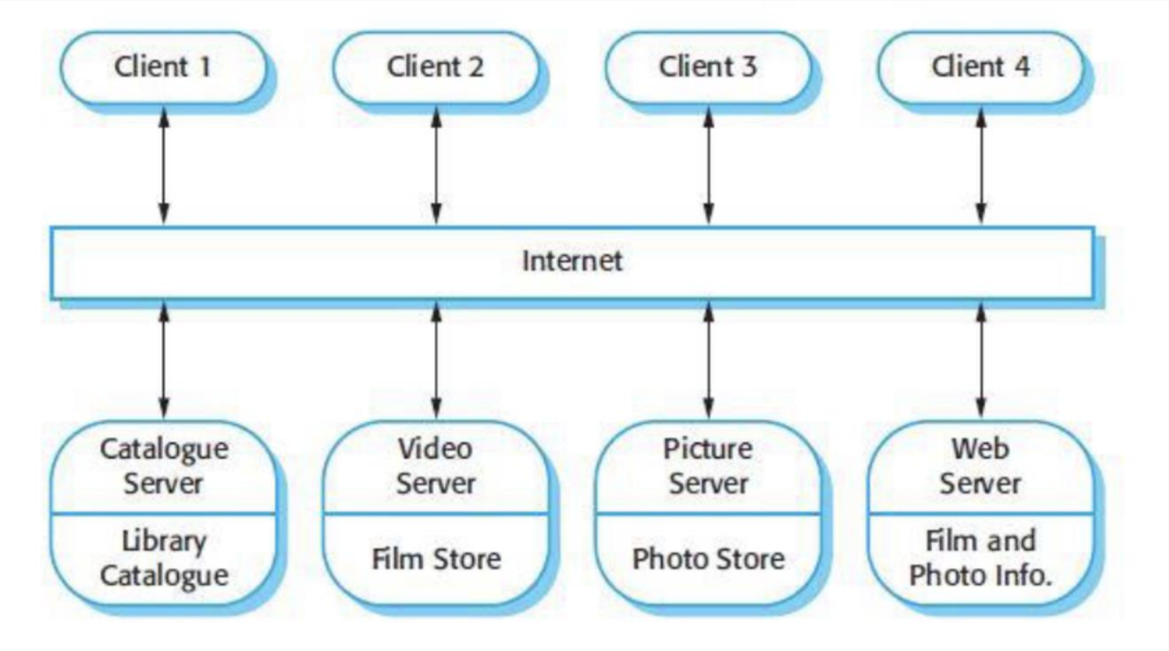
Advantages:

- Performance

Disadvantages:

- Prediction of performance is hard

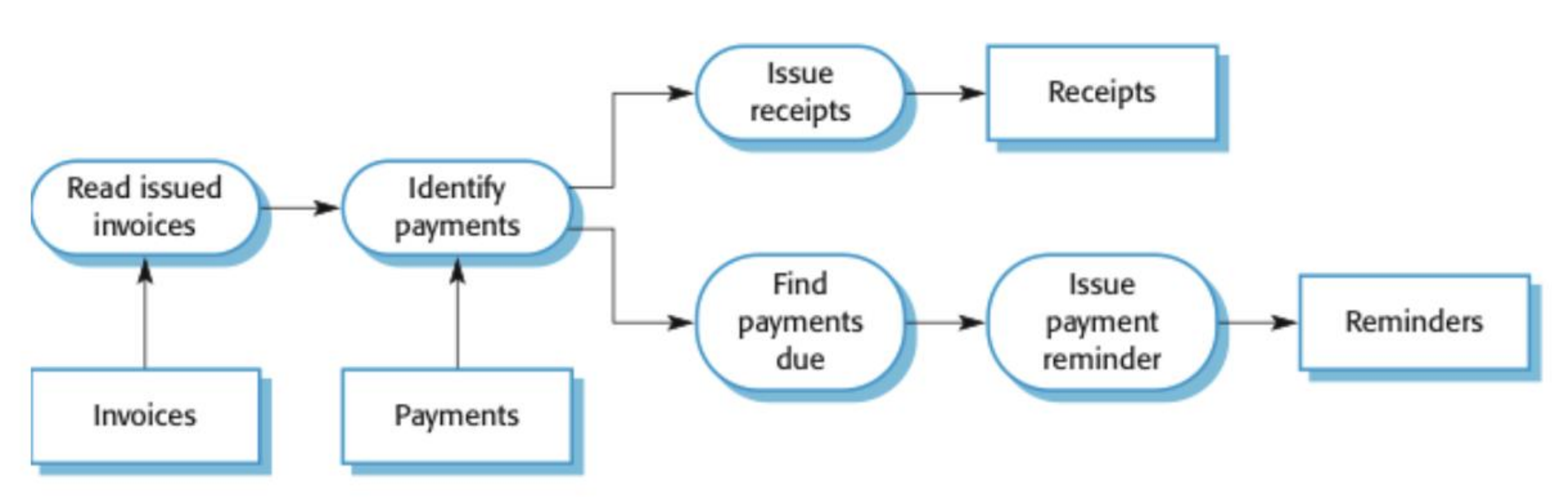
- Single point of failure



ARCHITECTURAL STYLES/PATTERNS

Pipe and Filter: Common for data processing applications (i.e, graphic processing, invoicing)

- There can be concurrent types
- There are transformation steps and the data flows
- Advantages:
  - Reuse
  - Workflows
  - Concurrency
  - Modification
- Disadvantages:
  - Agreed on the pipe



GENERIC APPLICATION ARCHITECTURES

- There are common architectures for similar business
- ERP (enterprise resource planning) system: easy to tailor to different business context. For example, Oracle, SAP
- Data processing applications
- Transactions processing
- Event processing applications
- Language processing applications

SERVICE-ORIENTED ARCHITECTURES (SOA)

- SOA are based around the notion of externally provided services.
  - Focuses on delivery of a given service using a communications protocol over the network
- A service has four properties according to one of many definitions of SOA
  - It logically represents a repeatable business activity with a specified outcome
  - It is self-contained
  - It is a black box for its consumers; consumer does not have to be aware of the service's inner workings
  - It may be composed of other services

ARCHITECTURE TYPES: SOA/MICROSERVICES

- **Microservices architecture:** modern approach to software architecture where the application is broken down into smaller, independent services that communicate with each other via lightweight mechanisms such as APIs
  - Each service is designed to perform a specific business function and can be developed, tested, and deployed independently of the others.
  - Each service is responsible for a specific task (i.e. authentication, payment processing)
    - Can be developed using different technologies and can be deployed independently, allowing teams to work on different parts of the application simultaneously
- Can introduce some complexities, such as managing communication between services, ensuring consistency across services, and monitoring and debugging distributed systems.
  - Choice to use microservices should be made based on the specific needs of the application and the organization.

ARCHITECTURE TYPES: SOA/MICROSERVICES - BENEFITS

- **Scalability** Microservices can be scaled independently of each other, allowing for greater flexibility and the ability to handle changes in traffic or workload.
- **Resilience** If one service fails, it does not necessarily affect the entire system, as other services can continue to function.
- **Flexibility** Teams can work on different parts of the application independently, allowing for faster development and deployment of new features.
- **Technology diversity** Different services can be developed using different programming languages or technologies, allowing teams to choose the best tool for the job.

CHALLENGES OF THE DESIGN PHASE

- The design team should not do too much.
  - The detailed design should not become code.

- The design team should not do too little.
  - It is essential for the design team to produce a complete detailed design.

- Good design requires experience in addition to education.
  - Experience is only learned through practice.
  - Best to learn good design practice from experienced designers while working in teams.

CONCLUSION

- A ***software architecture*** is the fundamental framework for structuring a system. It is critical during design to determine the correct architecture to satisfy the system requirements.
- Architectures have different performance, scalability, security, and availability. Complex architectures are not always better.