SOFTWARE ENGINEERING PRACTICES

Submitted to :

Dr. Muhammad Asim Rehmat

MENTOR

Highly experienced Technology Consultant with a Ph.D in Computer Science, Automation and Production Inspection and over 15 years of expertise in safety-critical industrial applications.

TEAM MEMBERS

Aiman Malik	[2021-CE-23]
Ansa Aslam	[2021-CE-21]
Ayesha	[2021-CE-18]
Ayesha Saeed	[2021-CE-12]
Ebaa Haq	[2021-CE-22]
Faiza Riaz	[2021-CE-20]
Maham Nadeem	[2021-CE-10]
Maheerah Khalid	[2021-CE-14]
Noor Fatima	[2021-CE-07]
Rabia Batool	[2021-CE-04]
Samreen Razzaq	[2021-CE-13]
Sana Israr	[2021-CE-55]
Urwah Imran	[2021-CE-15]

AGENDA

- Popular Architectural Styles
- Ensuring Security
- Optimizing Performance
- Program Design and Development
- Code reusability
- Conclusion

ARCHITECTURAL STYLES

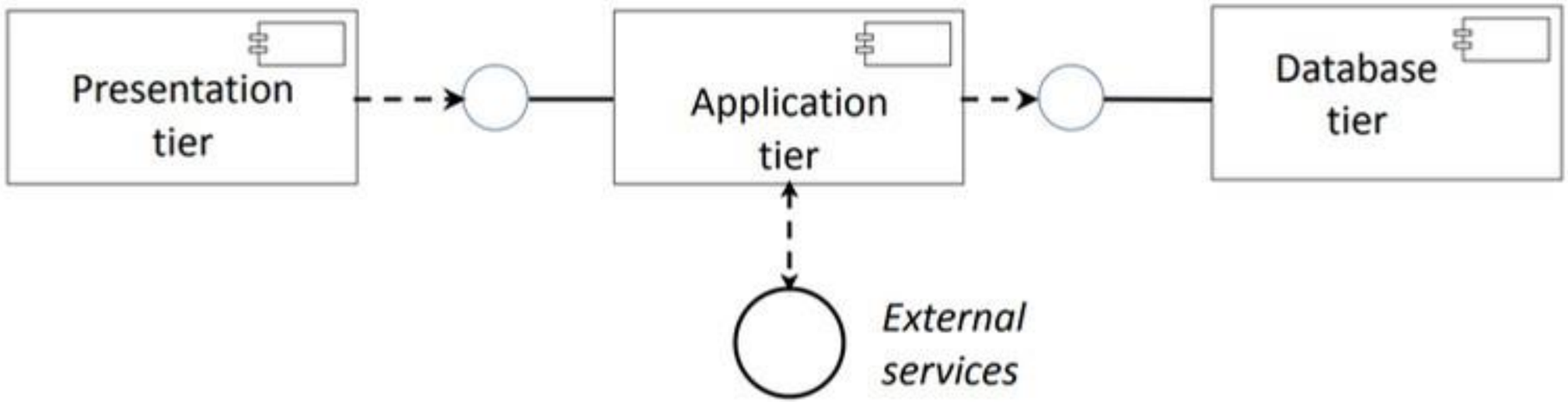
01 Three Tier Architecture

02 Master File Update

03 Model/View/Controller (MVC)

THREE TIER ARCHITECTURE

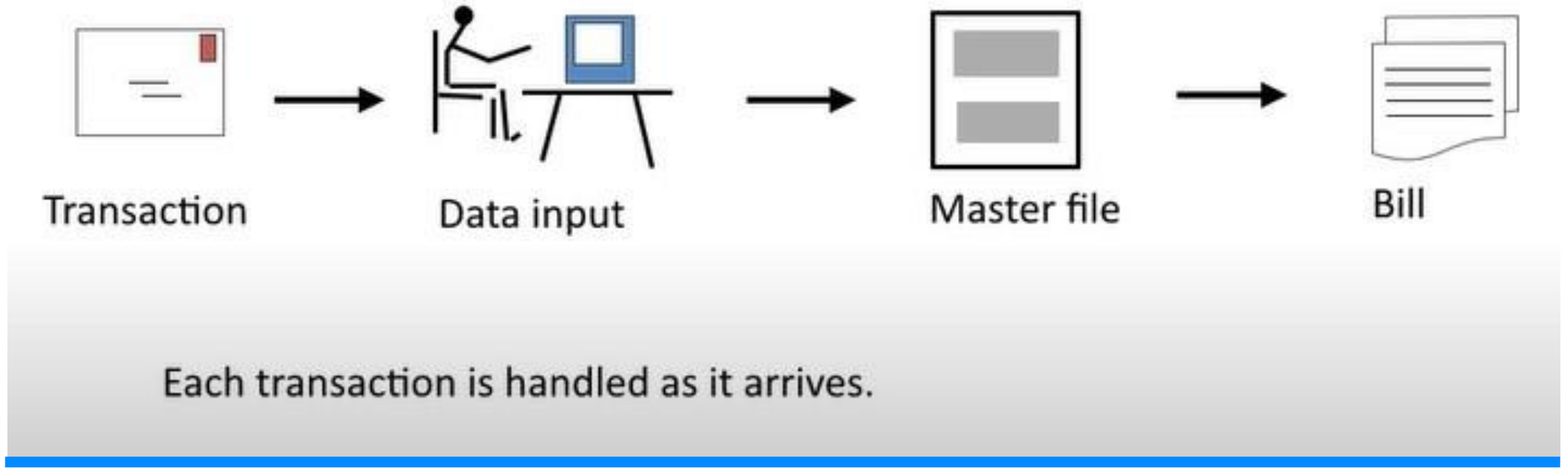
This architecture is an extension of the client/server model. It is the standard architecture for small and medium sized web sites.



MASTER FILE UPDATE

This architecture is an alternative to the repository model.
It is very widely used in data processing systems.

Example: Electricity Utility Billing



MASTER FILE WITH BATCH PROCESSING:

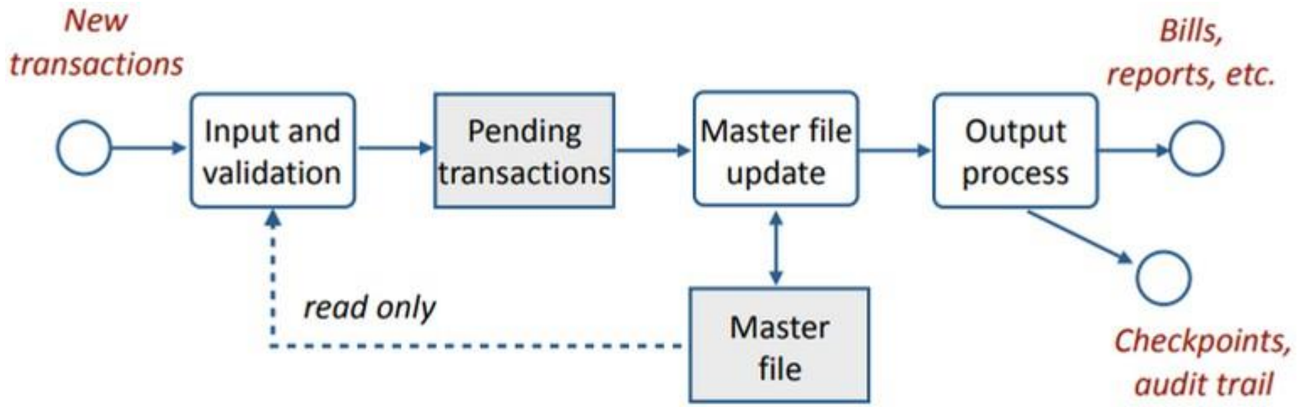
- **Input and validation process**

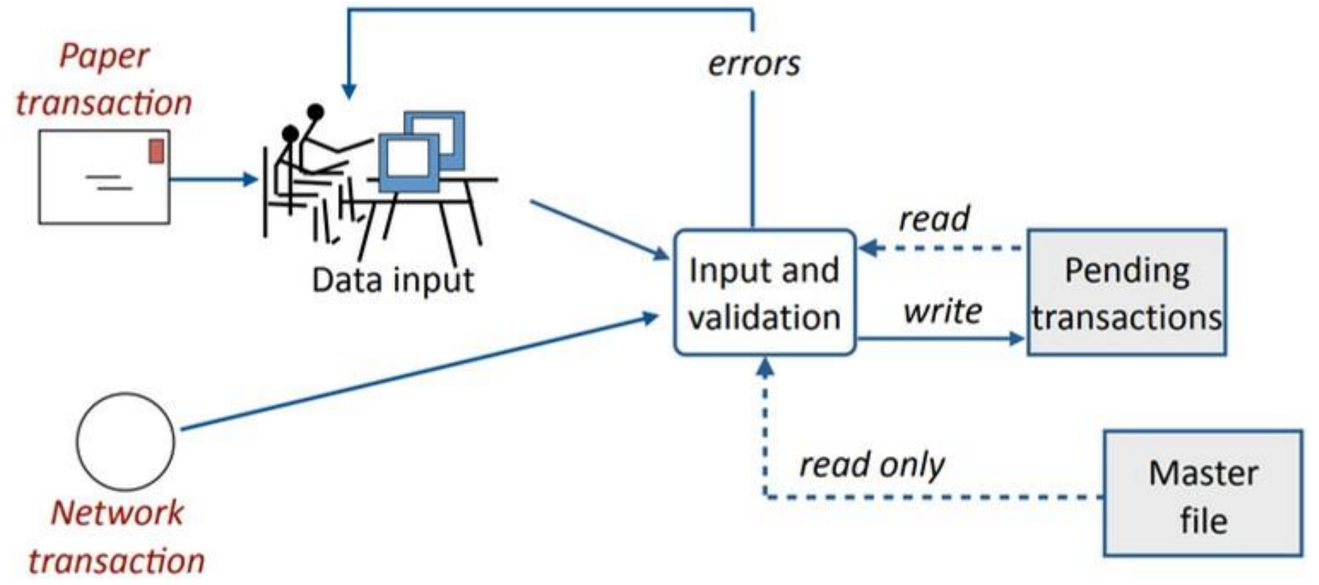
  runs throughout the day.

  processes transactions when they arrive.
- **Master File Update**

  runs once per day (usually at night).
- **Output process**

  runs after the master file update finishes.





BATCH PROCESSING PROS AND CONS

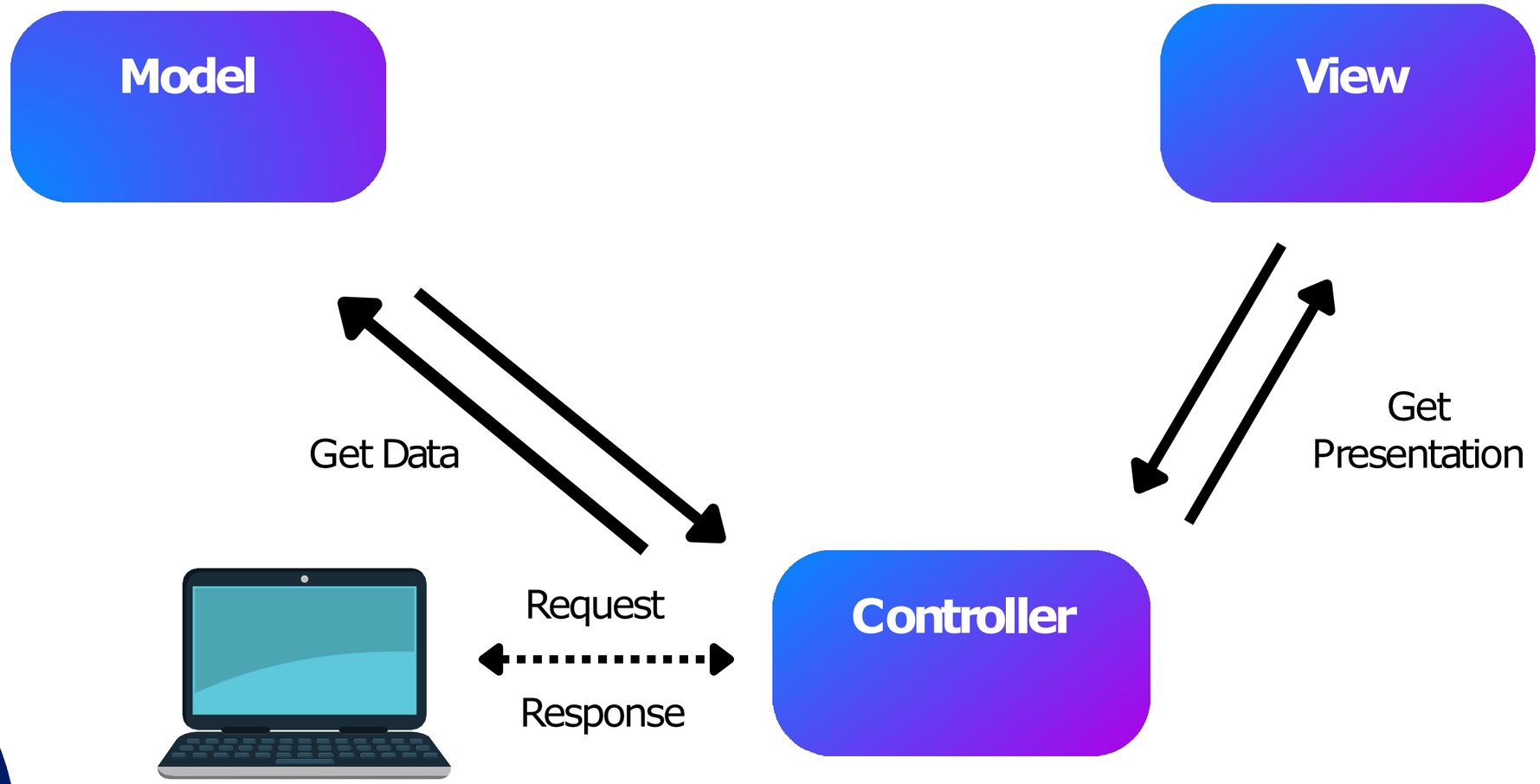
Pros

- Backup and recovery have fixed checkpoints.
- Better management control of operations.
- Efficient use of staff and hardware.
- Error detection and correction is simplified.

Cons

- Information in master file is not updated immediately.
- No good way to answer customer inquiries.

MVC Architecture



Model View Controller

View

State Presentation
Model Subscription
Update Reception
User Input Handling

Model

State Change Handling
Centralized State Management
Validation
Responsibility
Notifications

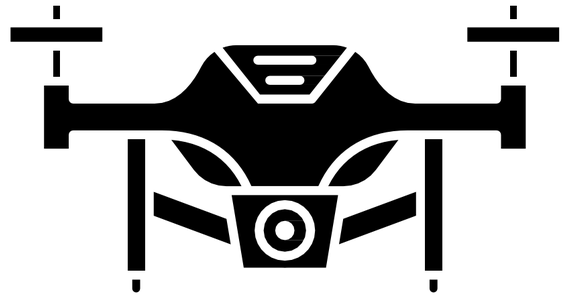
Controller

Application Behavior
User Action Mapping
External Service

Example from a CS 5150 project

Pilot's Request:

Pilot requests a flap setting change to 20 degrees for slower landing speed.



- View to Controller
- Controller to Airplane
- Airplane Confirmation
- Controller to Model
- Model Update
- Model to View


SECURITY
SECURITY
SECURITY
SECURITY
SECURITY
SECURITY
SECURITY
SECURITY

SECURITY

Security in Software Development

Our Goal:

- Security of data
- Security of system

Security Needs:

- Integrity
- Secrecy

Security Dangers:

- Damage to privacy
- Theft of money

The Economics of Security:

- Cost and time in software development.
- Accepting risk as cost of doing business.
- Customer attitude toward security spending.
- Practical security balances cost and risk.

SECURITY WITHIN AN ORGANIZATION

PEOPLE

- Dishonest and disgruntled employees.
- Malicious Intent.

DESIGN FOR SECURITY

- Make it easy for responsible people.
- Make it hard for dishonest or careless people.

External Intruders

Types

The complexity of modern software's constant search for vulnerabilities Government

Dangers

Unauthorized Access
backdoors
denial of service
spoofing

Online data captured by systems
If data is hacked
Damages users
Damage to company

MINIMIZING RISKS IN SOFTWARE DEVELOPMENT

- **System Design:**
  - Secure protocols, authentication, and access barriers.

    Emphasize the importance of a well system architecture.
- **Programming:**

  Defensive programming and rigorous testing.
- **Operating Procedures:**
  - Backup, auditing, vulnerability testing.
  - Training and monitoring personnel for enhanced security awareness.

SYSTEM ARCHITECTURE FOR RISK MINIMIZATION

Secure Protocols:

- HTTPS encryption for data in transit.
- Encryption of stored passwords and two-factor authentication.

Barriers:

- Firewalls, private networks, virtual private networks.
- Data security measures such as encryption

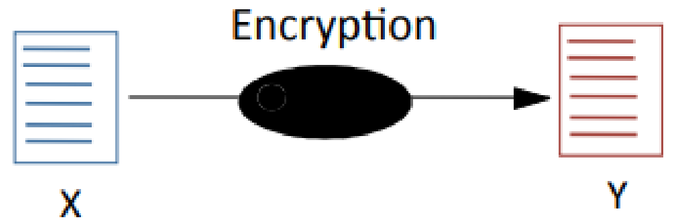
Firewalls

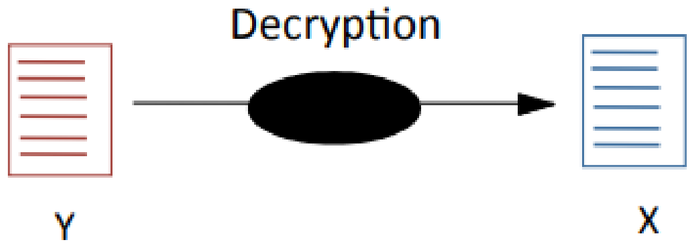
- Use firewalls to inspect and control data flow between network segments.
- Highlight effective isolating components.

Online Data Management:

- Collect essential data only.
- Regularly perform security audits and be prepared for data-related troubles.

SECURITY TECHNIQUES



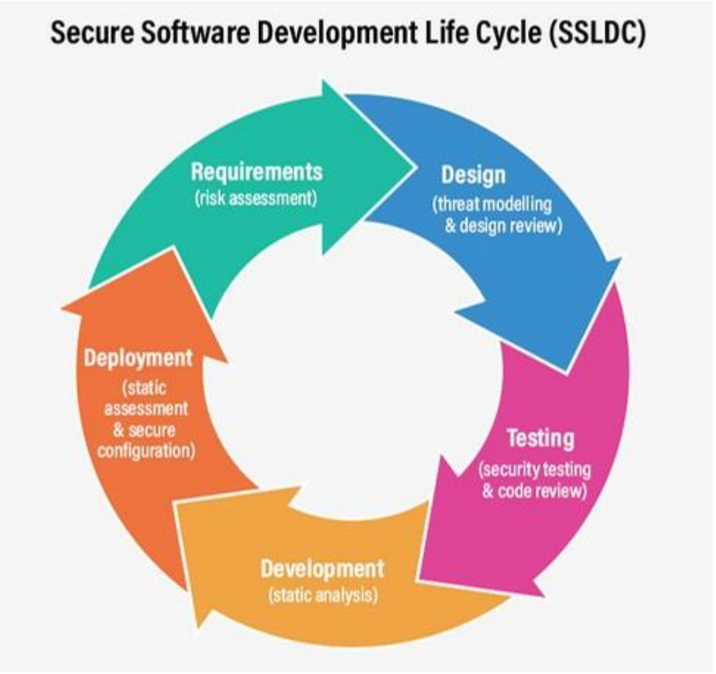


Encryption and Decryption:

- Authentication establishes identity; authorization defines actions.
- Access control lists and group membership as examples.

Programming Secure Software:

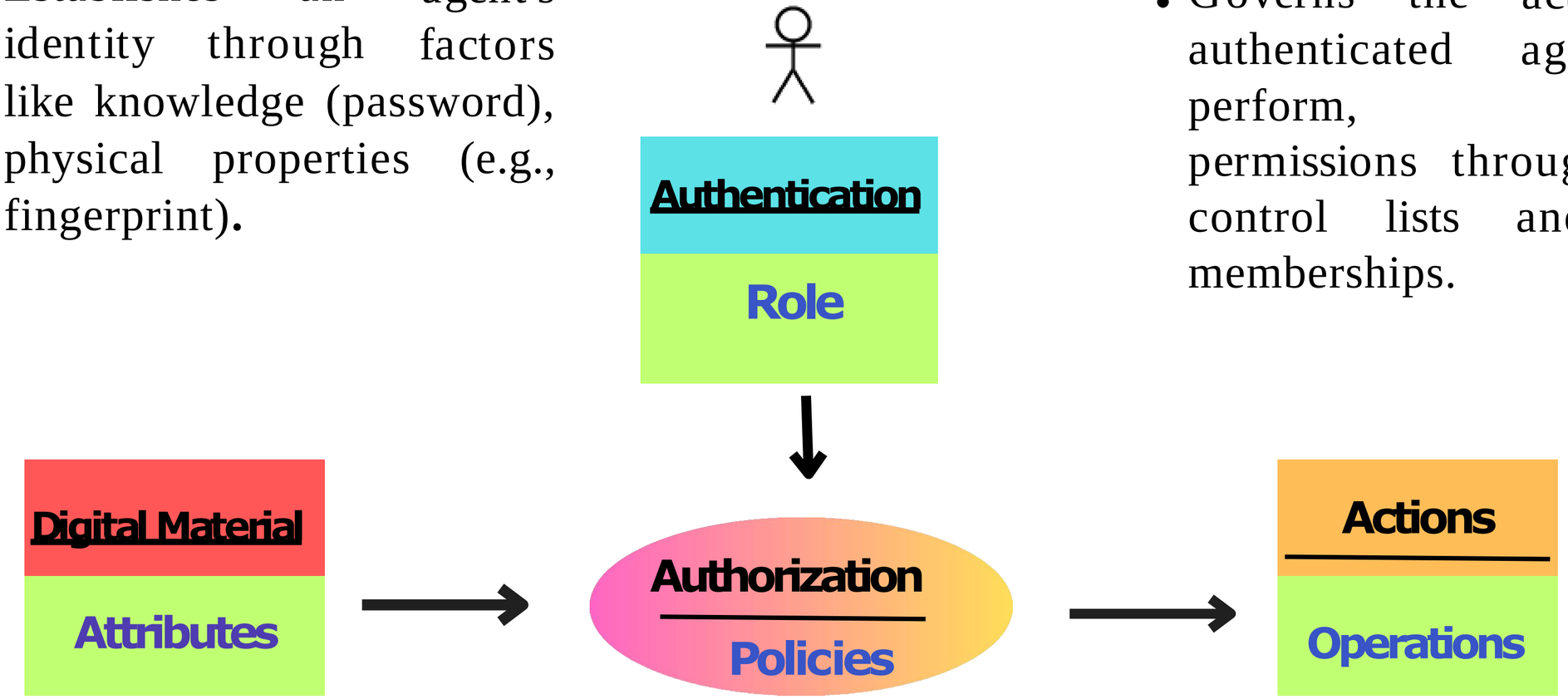
- Emphasize the importance of encryption for secure data transmission.
- Discuss the challenges of developing secure and reliable components in a large system.



AN ACCESS ARCHITECTURE FOR DIGITAL CONTENT

Authentication :

- Establishes an agent's identity through factors like knowledge (password), physical properties (e.g., fingerprint).



Authorization:

- Governs the actions an authenticated agent can perform, defining permissions through access control lists and group memberships.

Performance

Performance of Computer Systems

Introduction

- Despite people costs surpassing hardware expenses, performance is crucial
- Emphasizes the impact of a single bottleneck on the entire system

Importance

- Real time systems
- Very large computations
- User interfaces
- Transaction processing

High-performance computing

- Large data collections
- Huge numbers of users and extensive computations

Performance Challenges and Techniques

- Prediction, Design, and Post-Implementation problem identification.
- Understanding hardware-software interactions
- Calculation of “capacity” and “load”

Example:

Calculations indicate that the capacity of a search system is 1,000 searches per second. What is the anticipated peak demand?

Assume a Growth Rate:

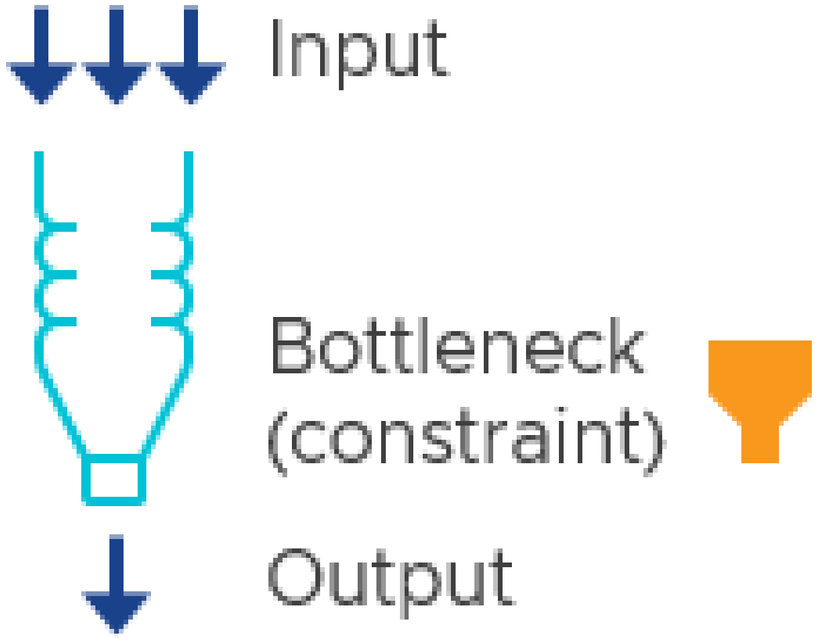
- Let's say you expect a 20% growth in search demand.

Apply the Growth Rate to Capacity:

- Anticipated Peak Demand = Capacity * (1 + Growth Rate)
- Anticipated Peak Demand = 1,000 searches/second * (1 + 0.20)
- Anticipated Peak Demand = 1,000 searches/second * 1.20
- Anticipated Peak Demand = 1,200 searches/second

Identifying Bottlenecks and Utilization

- Hardware Bottlenecks
- Inefficient Software
- CPU Limitations in certain domains
- **Utilization**
  - Proportion of a service's capacity used on average.
  - $$\text{utilization} = \frac{\text{mean service time for a transaction}}{\text{mean inter-arrival time of transactions}}$$



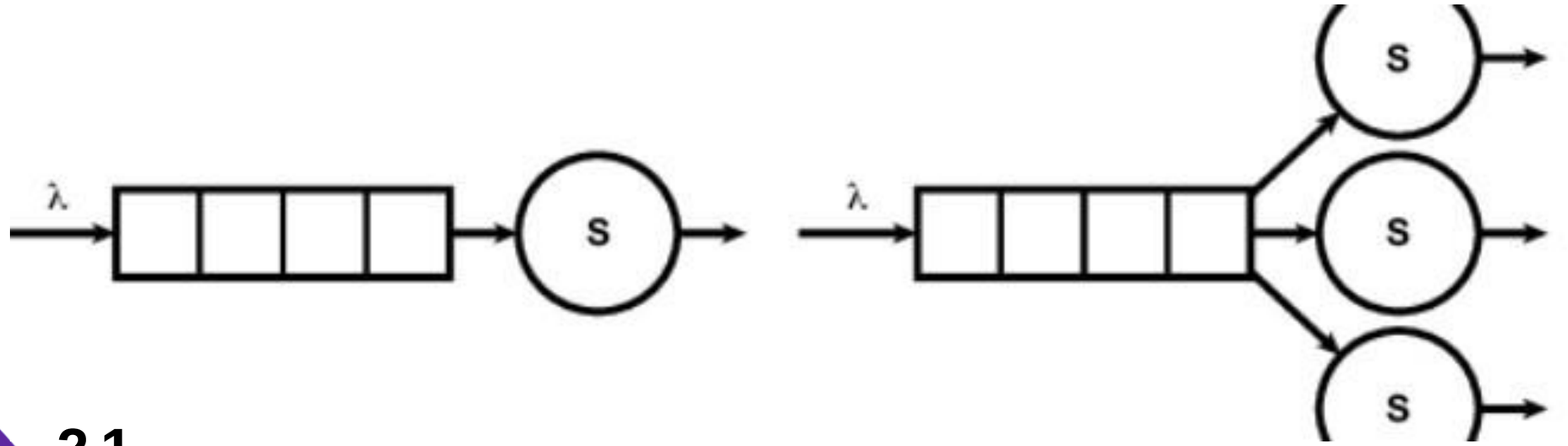
Queues

Single-Server Queue

Tasks being processed on a computer with single processor

Multi-Server Queue

Tasks being processed on a computer with several processors



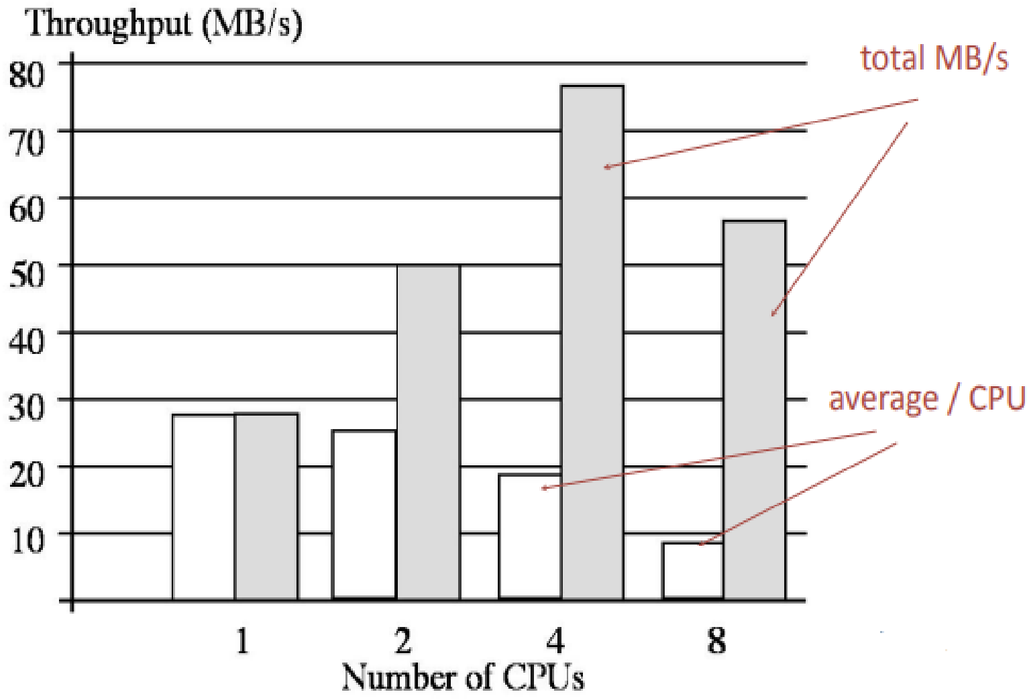
Simulation Techniques

- Introduction to simulation techniques for modeling systems with **states** and **events.**
- Overview of **Discrete Time Simulation** and **Next Event Simulation.**
- Highlight the importance of simulating events using **random variables** or **real data.**

System Performance Analysis

Strategies to understand system performance:

- Emphasize the role of **benchmarks** in running standard problems or simulated loads.
- Discuss the significance of **instrumentation** in tracking specific events with a clock.

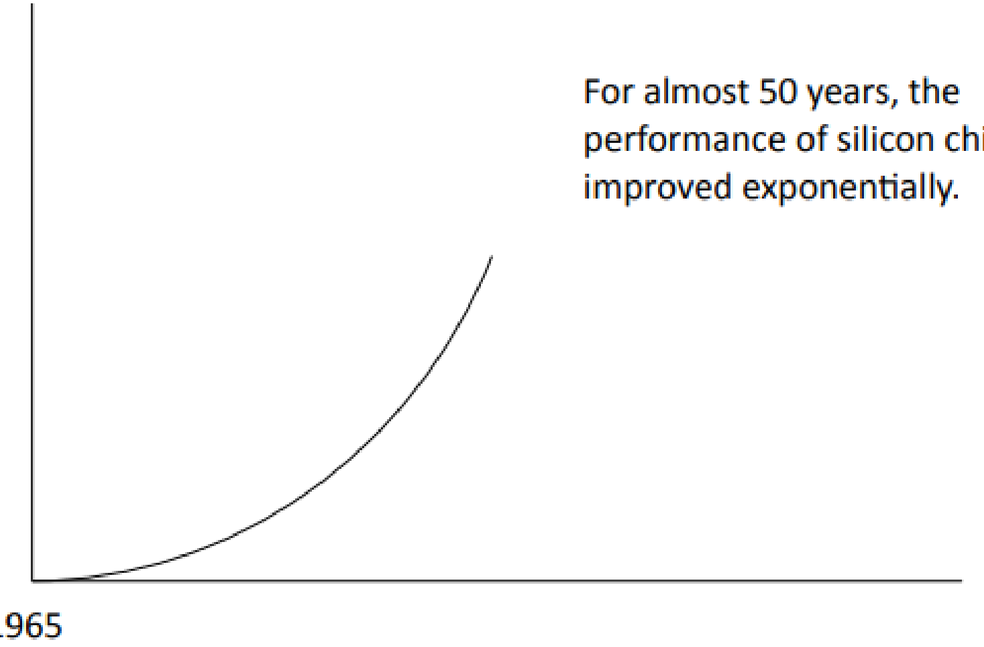


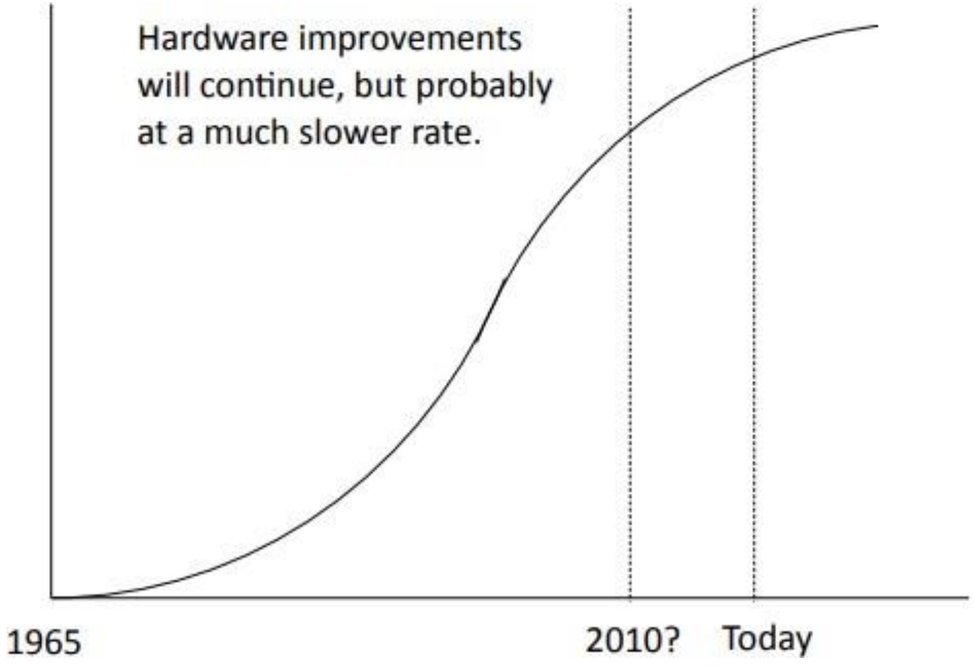
Moore's Law

"The density of transistors in an integrated circuit will double every year."

- From 1965 to around 2010, chip performance, transistor density, and clock speed doubled every two years.
- Power consumption decreased, and costs dropped by about 20% annually.

Address the slowdown in improvements since 2010 due to physical and economic constraints.





Parkinson's Law

"Work expands to fill the time available."

Parkinson's Law applied to software development:

Historical Software Demands:

Demand expanded to utilize all available hardware.

Low prices created new demands.

Future Challenges:

Emerging applications in artificial intelligence, scientific computing, and vision will demand more computing power.

Software developers face continual pressure with hardware improvements slowing down.

Increasing Performance and Future Trends

Strategies for increasing performance:

1. Use **special-purpose hardware**, such as graphics processors.
2. In the past, **parallel processing** occurred within the processor chip or with several CPUs on one chip, while today and in the future, it involves multiple processors, cluster computing, and cloud computing.

Thoughts for the Future:

Future trends emphasizing parallel and distributed computing, network performance, and data center complexity.

Models for Program Design

Approaches to Program Development

Heavyweight Approach

- Design and coding are separate
- UML is suitable for detailed design

Lightweight Approach

- Interwoven design and coding
- Iterative development using IDE

Mixed Approach

- Outline design using models, details worked iteratively during coding

Program Design Overview

“Represent software architecture for implementation.”

Specifies:

- Programs, components, packages, classes, hierarchies.
- Interfaces, protocols (if not in system architecture).
- Algorithms, data structures, operational procedures.

UML Models

- Used for almost all aspects of program design.
- Diagrams provide a general overview.
- Specifications offer details about each design element.
- Heavyweight processes aim to complete specifications before coding.

UML Models for Program Design

Class Diagram:

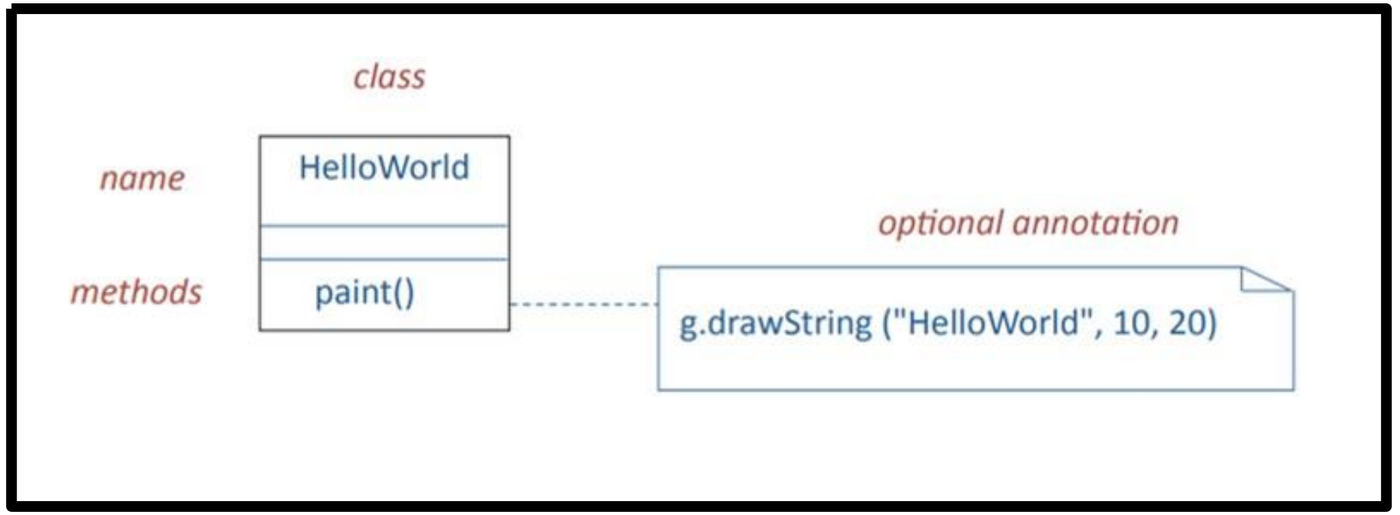
- Describes classes, interfaces, collaborations, and relationships.

Object Diagram or Sequence Diagram:

- Represents objects and their relationships.

Example - "Hello, World!" Applet

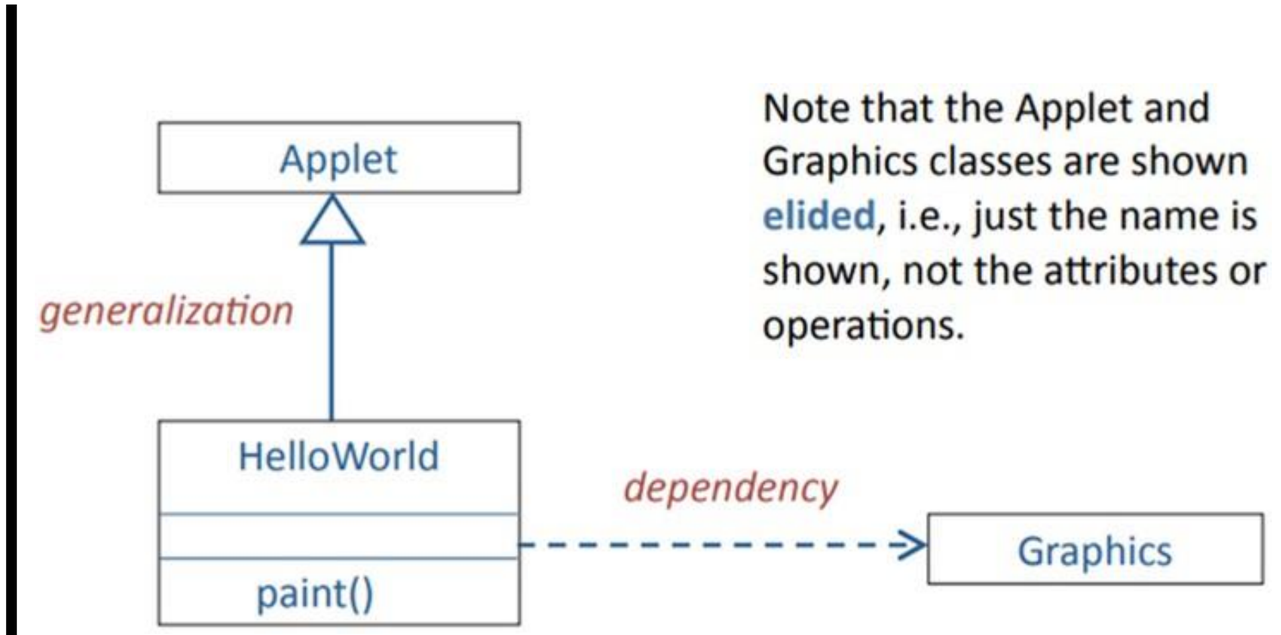
```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet {
    public void paint (Graphics g) {
        g.drawString ("Hello, World!", 10, 20);
    }
}
```



- **Realization**

- **Generalization**

- **Realization**



Noun Identification and Class Diagram

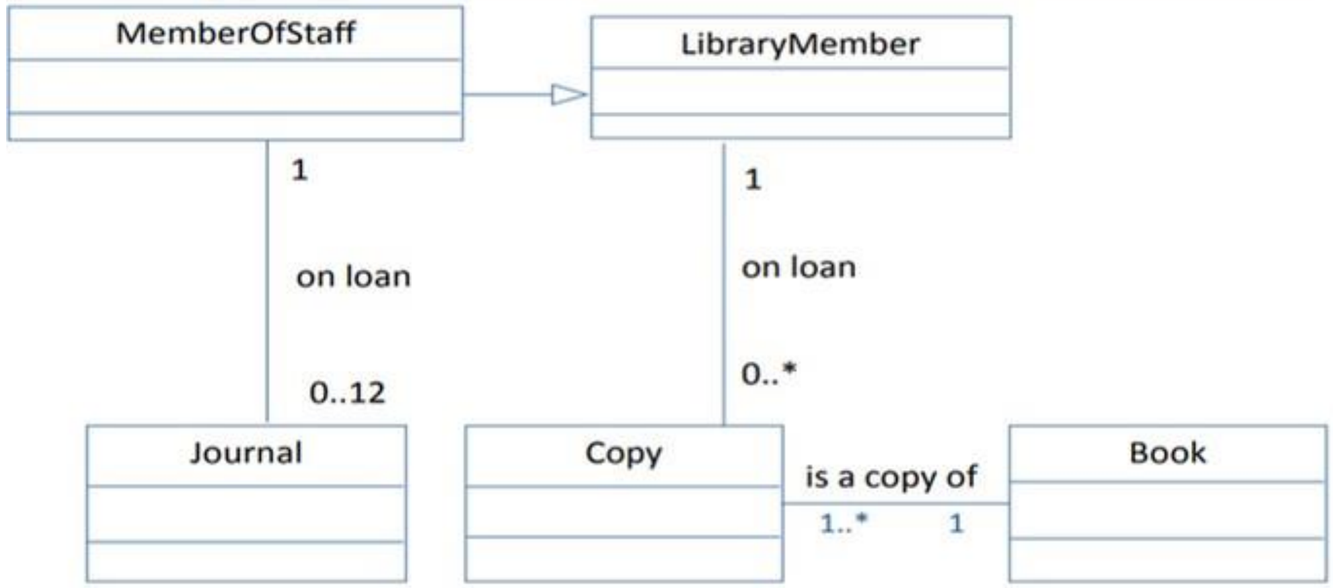
"A good design is often a combination of application classes and solution classes."

- Application classes represent application concepts
- Noun identification is an effective technique to generate candidate application classes.
- Solution classes represent system concepts, e.g., user interface objects, databases, etc

Noun Identification:

A Library Example

A Possible Class Diagram




DEVELOPMENT
render() {
return (
<React.Fragment>

<Title name="our" title= "product

<ProductConsumer>
{(value) => {
console.log(value)
}}
</ProductConsumer>

</React.Fragment>

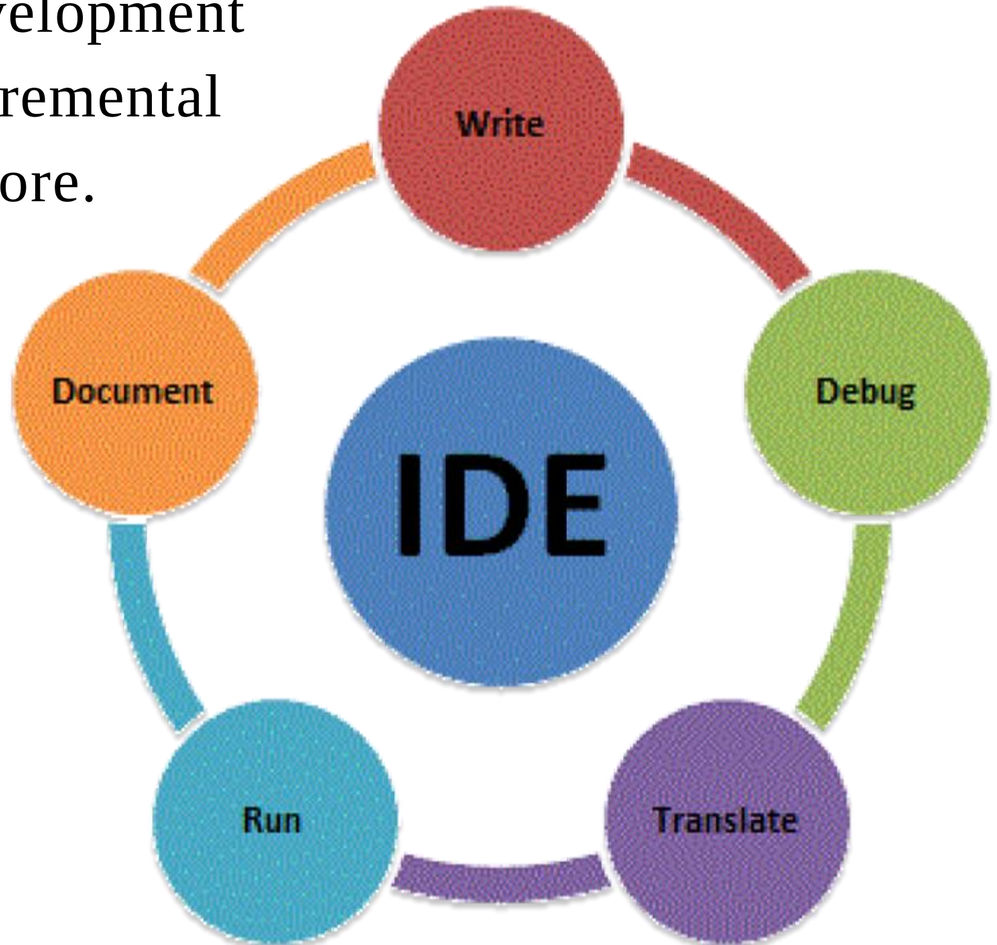
Introduction to Integrated Development Enviroments(IDES)

Essential Components of Software Development:

- IDEs amalgamate several tools for streamlined development
- Integrated components like source code editor, incremental compiler, build automation tools, debugger, and more.

Key Components:

- Text editor (e.g., vi editor for Linux)
- Compiler for individual files
- Build system (e.g., make for Linux)



Eclipse - A Comprehensive Integrated Development Environment

Introduction to Eclipse:

- Eclipse is a contemporary IDE developed by IBM's Rational division.
- Available in versions for various languages, including Java, C/C++, Python, etc

Java System in Eclipse Provides:

- Debugger
- Programming documentation
- Build automation tools
- Version control
- XML editor and tools
- Web development tools

Program Design and Class Hierarchies in IDEs

Program Design in IDEs:

- A pre-existing program design involving classes, methods, data structures, and interfaces.
- Using modeling tools (e.g., UML), design while coding, leveraging existing frameworks.

The Design-Code-Redesign Loop:

- Iterative process for straightforward class structures.
- Create an outline, write code, and modify class structures iteratively.
- Challenges arise with increasing program complexity.

Django Framework for Web Development

Introduction to Django Framework:

A Python-based web development framework

Architecture:

Loosely based on MVC (Model-View-Controller)

Features:

- Web and database server
- Web template system
- Authentication system
- Administration interface
- Integration of web callbacks

Responsive Design with Bootstrap

Challenge:

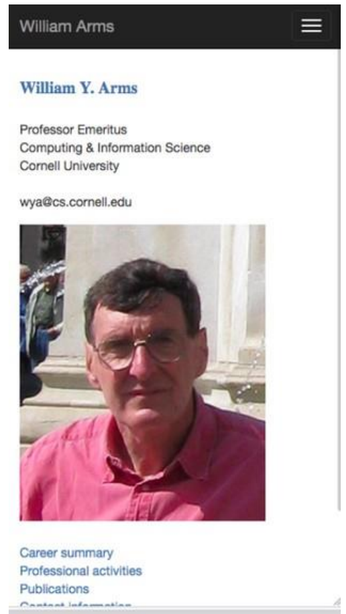
CSS media queries complex for responsive design

Solution:

Bootstrap Framework simplifies responsive website development

Primary Features:

- Simplifies responsive website development
- User-friendly tools for responsive design
- Illustration: Web page on iPhone vs. laptop



Advanced Development Environments

Frameworks with Development Environments:

Example: Django with Eclipse

Components:

IDE, Application Framework, UI Layout Manager

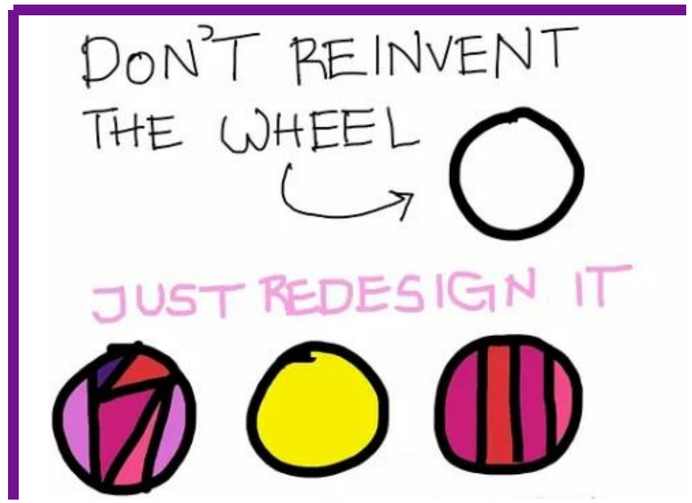
Example: Apple's Xcode for iOS

- **Specific purpose:**
  Mobile apps for Apple devices
- **Features:**
  Special programming language, MVC framework, auto layout

Reuse and Design Patterns

Software Reuse

Developing the new software systems and components by using the existing software artifacts such as code, frameworks etc.



Ways of Reuse Software:

- Open Source software
- Application Packages

Open Source Software

Software with source code that is designed to be publicly accessible.

Application Packages

Pre-built ready to use software solutions that provide specific functionalities

Design for Change

Designing software systems with the intention of making it adaptable and maintainable over time.

Key Components

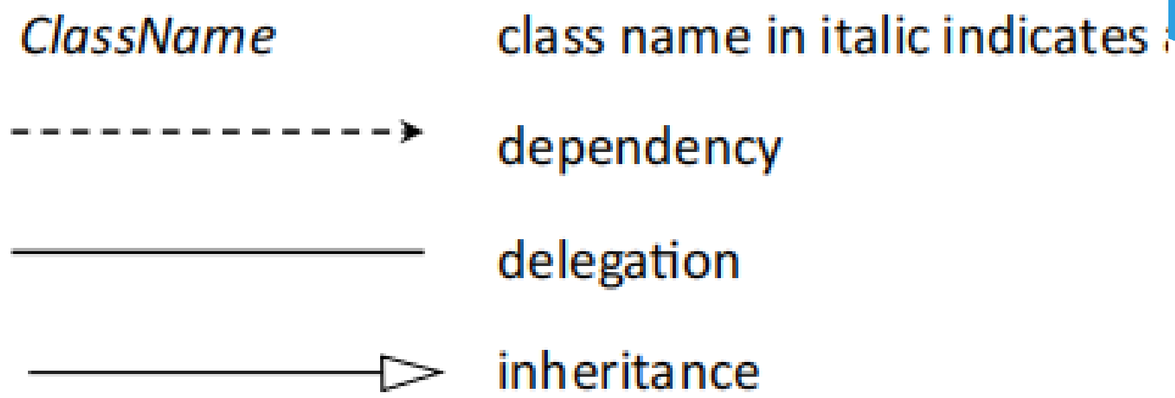
- New vendor or technology
- New implementation
- Addition to the requirements
- Change in the application Domain

Design Patterns

Standardized solutions to common problems in software design, making it easier to create flexible and maintainable systems.

Extensive use

- Inheritance
- Abstract classes
- Delegation



Types of Design patterns

- Adapter design pattern
- Bridge design pattern
- Facade design patteren
- Strategy deign pattern

Adapter Design Pattern

The Structural design pattern and its used so that two unrelated interfaces can work together.

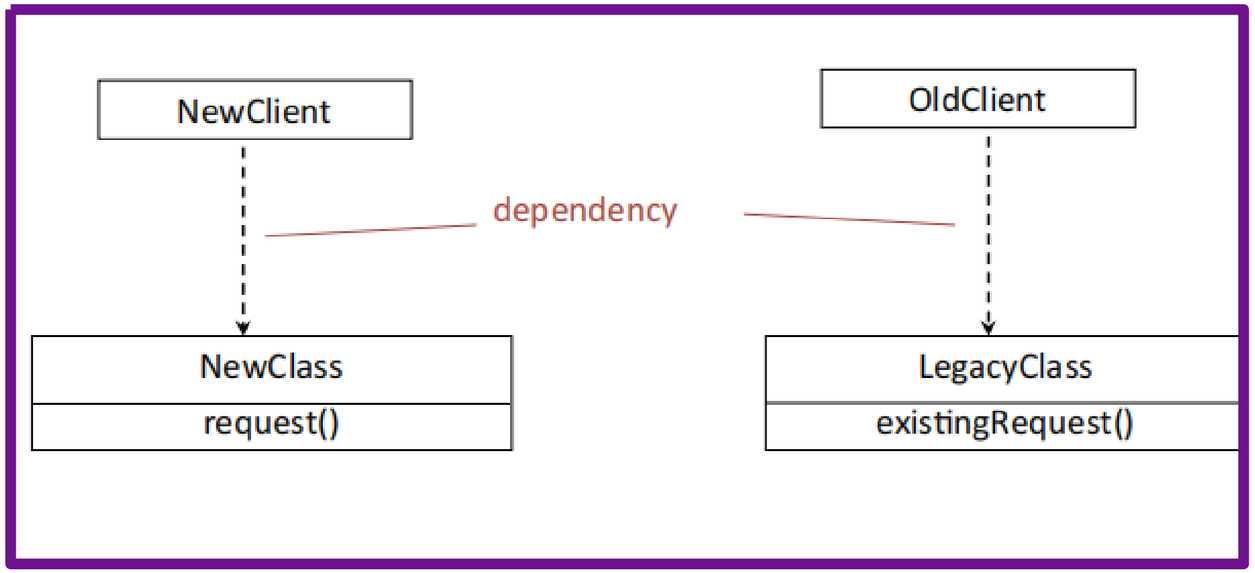


Figure : The Problem

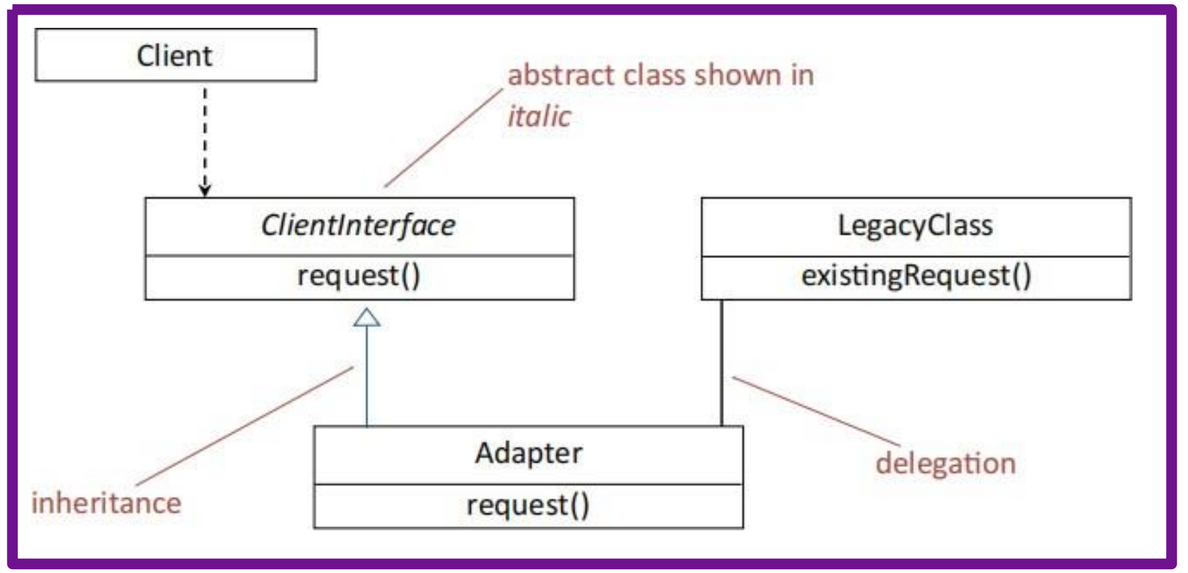


Figure : The Solution

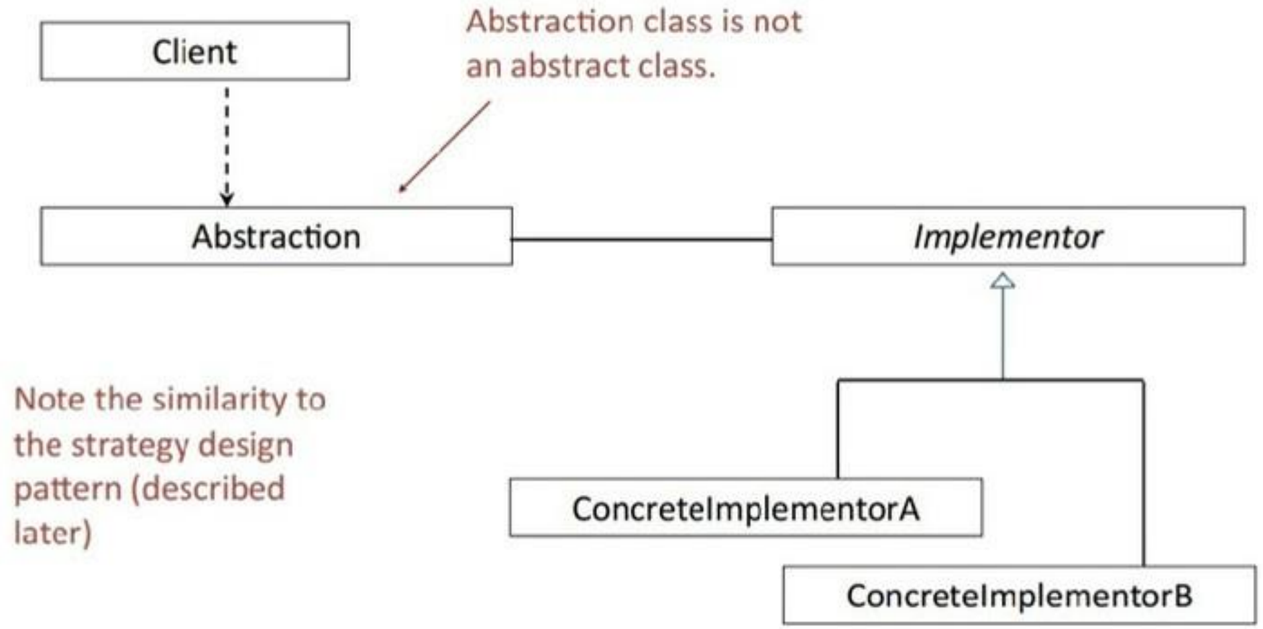
Bridge Design

Problem Description

Solution

Consequences

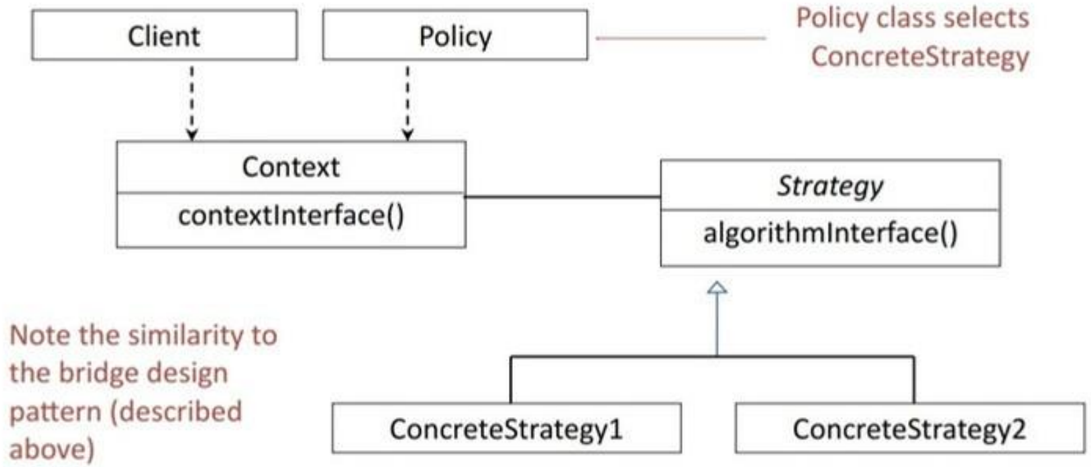
Client is shielded from abstract and concrete implementations
Interfaces and implementations can be tested separately



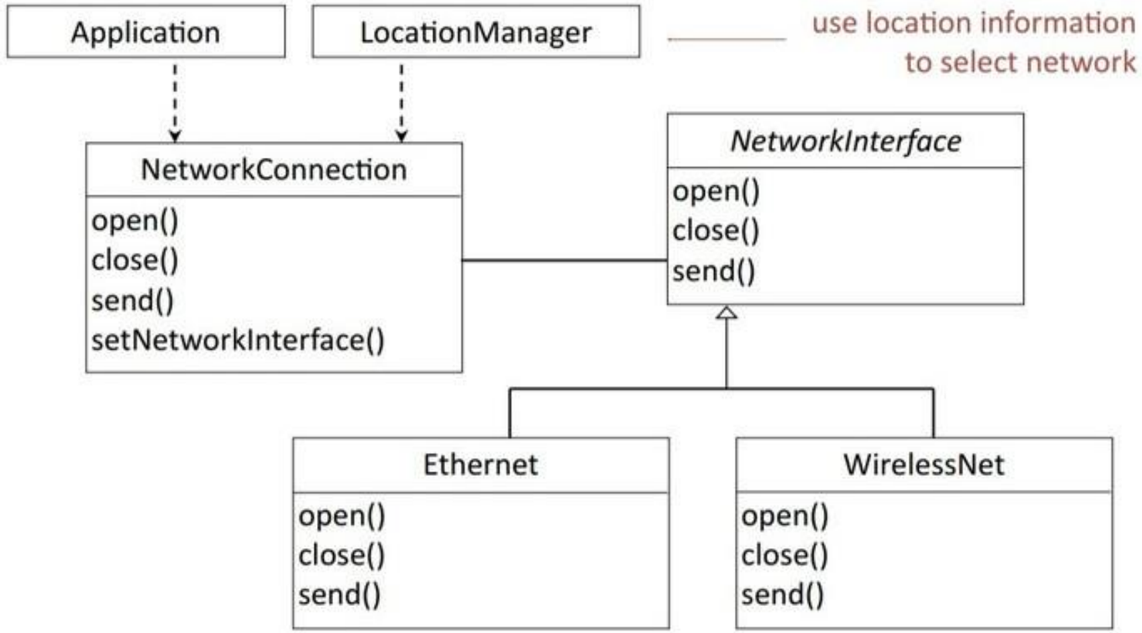
Class Diagram

Strategy Design

- **Problem Description**
- **Solution**
- **Example**
- **Consequences**



Class diagram

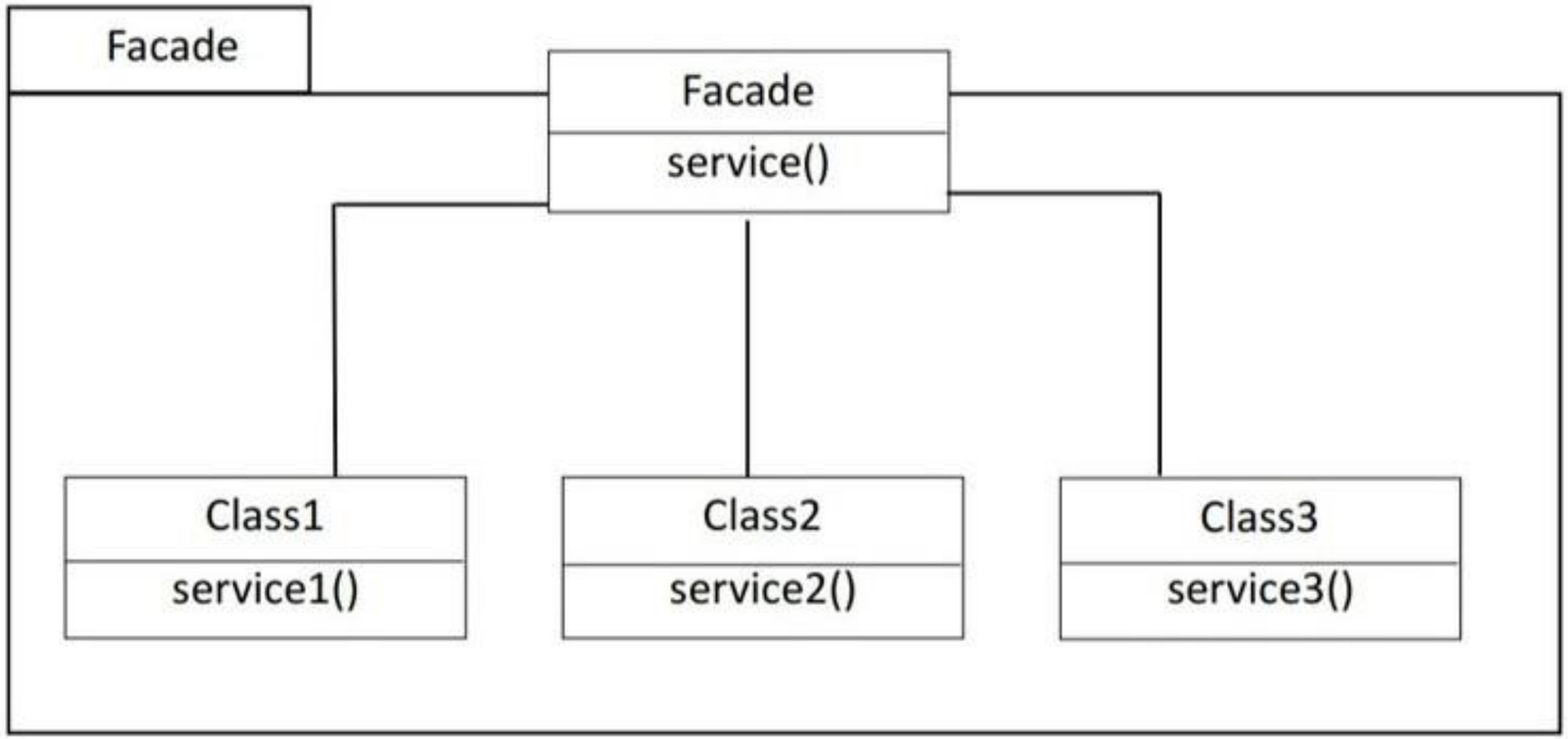


Example

Facade Design

- **Problem Description**
- **Solution**
- **Example**

The Compiler (Facade) class consolidates components (LexicalAnalyzer, Parser, CodeGenerator) into a seamless, single interface for the caller



Class Diagram

Content

- Representing Recursive
- Hierarchies
- Class diagram
- Legacy system
- Legacy code

Representing Recursive

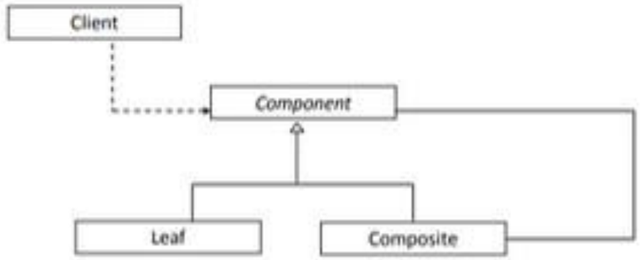
- Refers to how recursion is utilized or represented in a system or within a specific context.

Hierarchies

- Elements are ranked based on their level of importance and relationship to one another.
- Manages complex system in software development.

Class Diagram

- Specifies the services that are shared between Leaf and Composite.
- Implements each service by iterating over each contained Component.
- The Leaf services do the actual work.



Legacy Systems

- Refers to older, often outdated, and established software that has been in use within an organization for a significant period.

Worst Case of Legacy Systems

- Lack of Developer Knowledge.
- Unclear System Functionality
- Incomplete and Outdated Documentation

Legacy Requirements:

- Planning.
- Requirements as seen by the customers and users.
- Requirements as implied by the system design.

Legacy Code

Legacy Code

- Refers to existing software code that has been in use for a significant period
- Often characterized by several key attributes.
- Typically refers to software that has been in existence for a considerable time, often several years or more.

Legacy Code Rebuilding the sotware

- Understand the original systems architecture and program design.
- Move to current versions of programming languages and systems software.
- Develop Missing Subsystems if Source Code is Missing.

Conclusion

- Emphasize the paramount importance of integrating robust security measures
- Highlight the significance of optimizing software performance
- Stress the importance of adhering to develop best practices
- Discuss the benefits of code reuse as a strategy to improve productivity, minimize errors,


Thank You !!!!