

# Technologies and Web Programming

## 2019/2020

### Web Application Architectures



# Overview

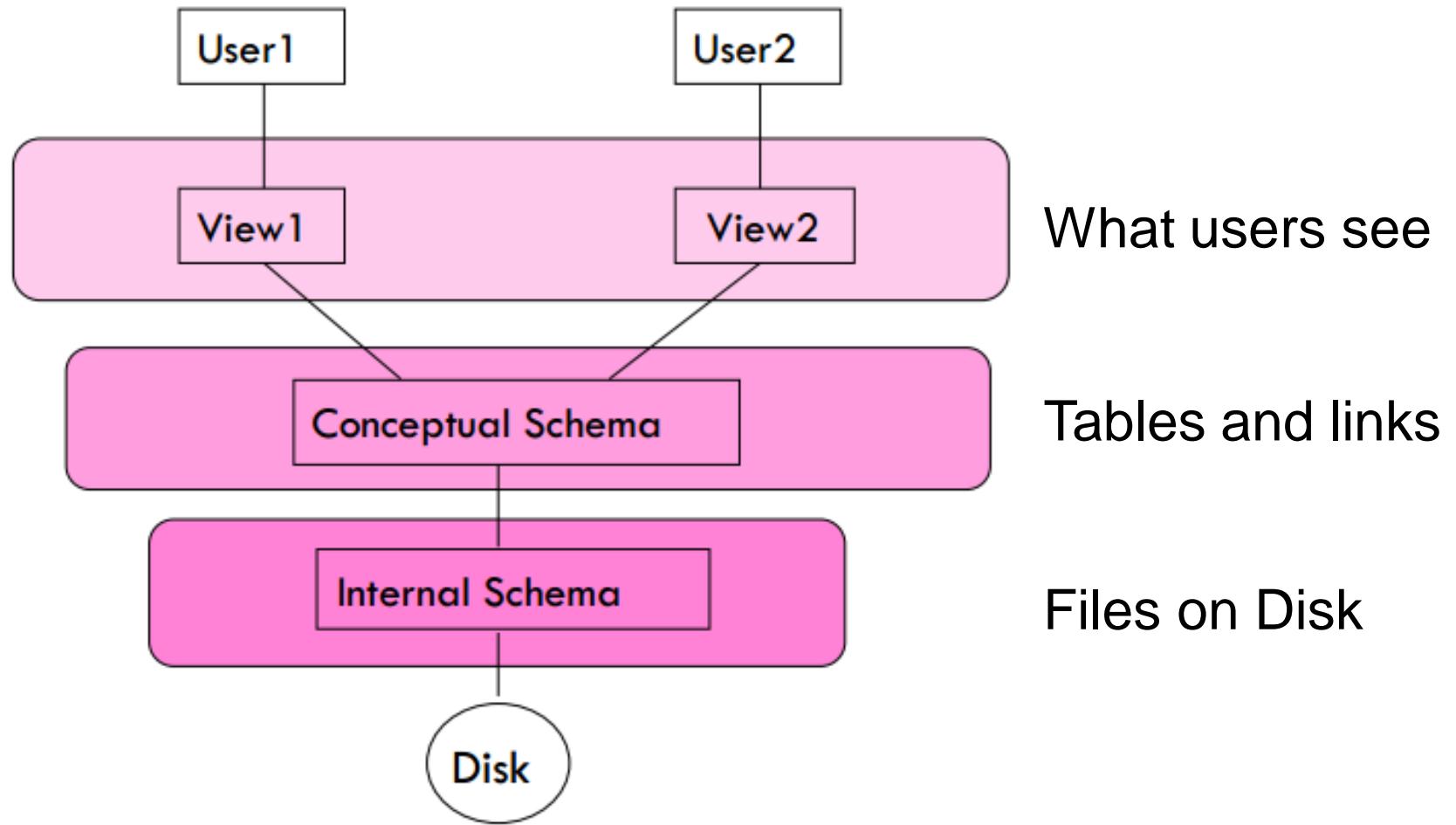
- Data Independence in Databases
- N-Tier Architectures
- Design Patterns
  - The MVC Design Pattern



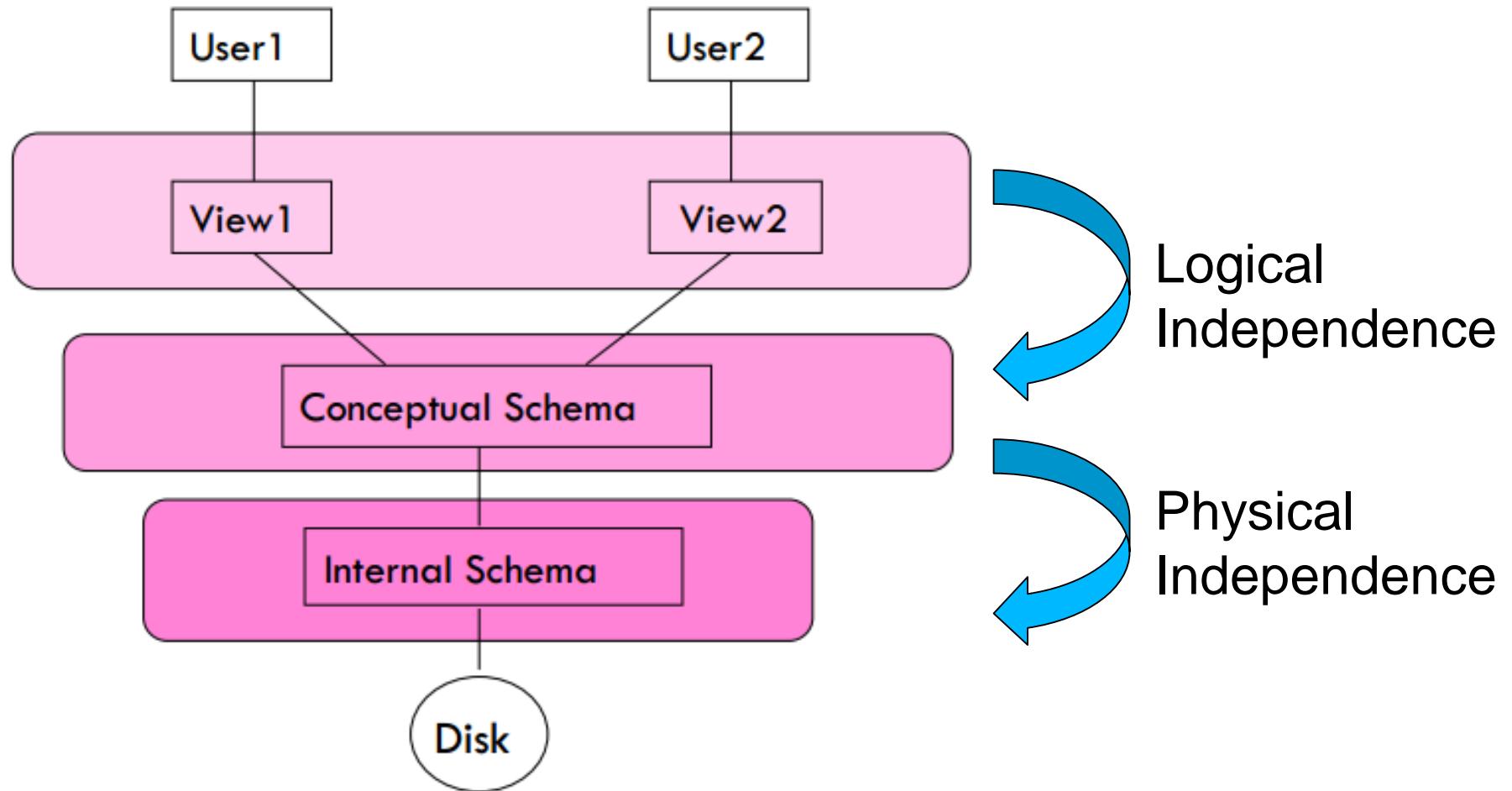
# Web Application Architectures

## Data Independence in Databases

# Database Architecture With Views



# Logical and Physical Independence



- Each level is independent of the levels below



# Data Independence

- **Logical Independence**
  - The ability to change the logical schema without changing the external schema or application programs
    - Can add new fields, new tables without changing views
    - Can change structure of tables without changing view
- **Physical Independence**
  - The ability to change the physical schema without changing the logical schema
    - Storage space can change
    - Type of some data can change for reasons of optimization



# Data Independence

- **Main Lesson to Learn**
  - Keep the VIEW
    - (what the user sees )
  - Independent of the MODEL
    - (domain knowledge)



# Web Application Architectures

## N-Tier Architectures

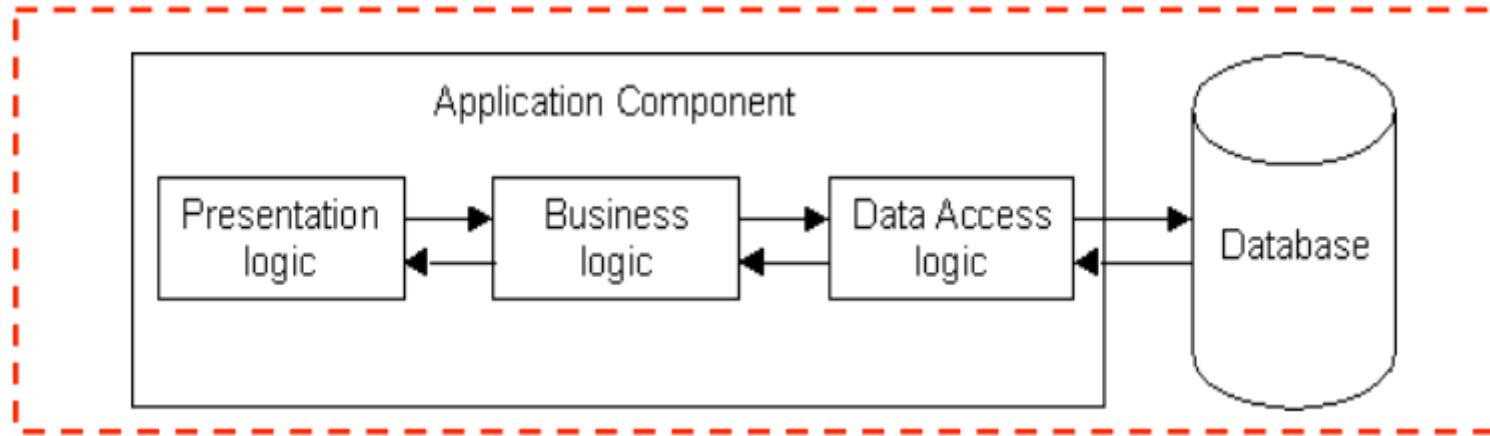


# Significance of “Tiers”

- N-Tier architectures have the same components
  - Presentation
  - Business/Logic
  - Data
- N-Tier architectures try to separate the components into different tiers/layers
  - Tier: physical separation
  - Layer: logical separation



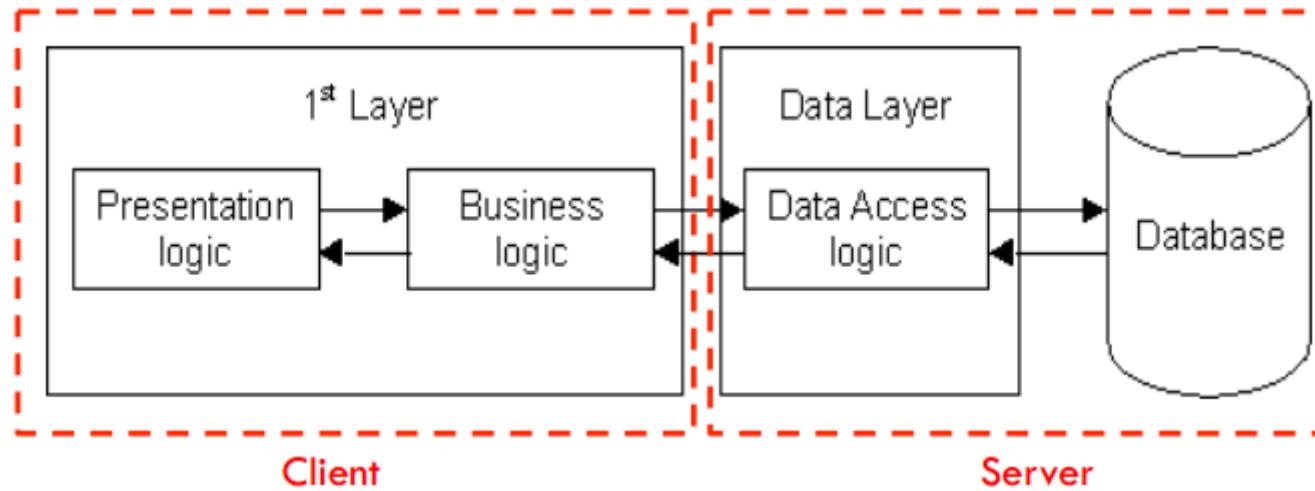
# 1-Tier Architecture



- All 3 layers are on the same machine
  - All code and processing kept on a single machine
- Presentation, Logic, Data layers are tightly connected
  - Scalability: Single processor means hard to increase volume of processing
  - Portability: Moving to a new machine may mean rewriting everything
  - Maintenance: Changing one layer requires changing other layer



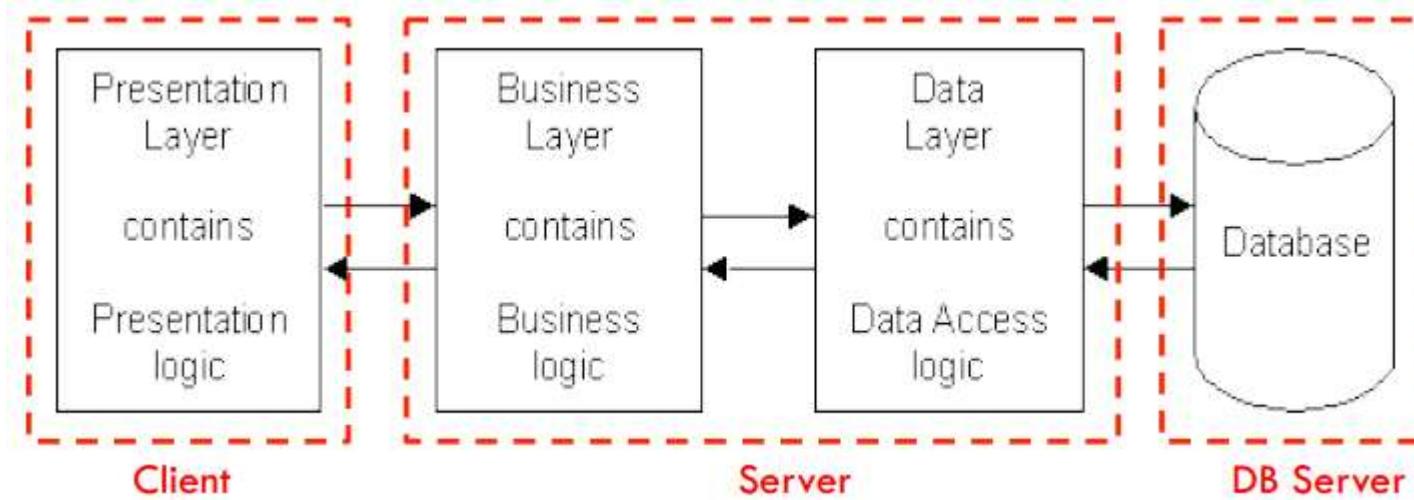
# 2-Tier Architecture



- Database runs on Server
  - Separated from client
  - Easy to switch to a different database
- Presentation and logic layers still tightly connected
  - Heavy load on server
  - Potential congestion on network
  - Presentation still tied to business logic



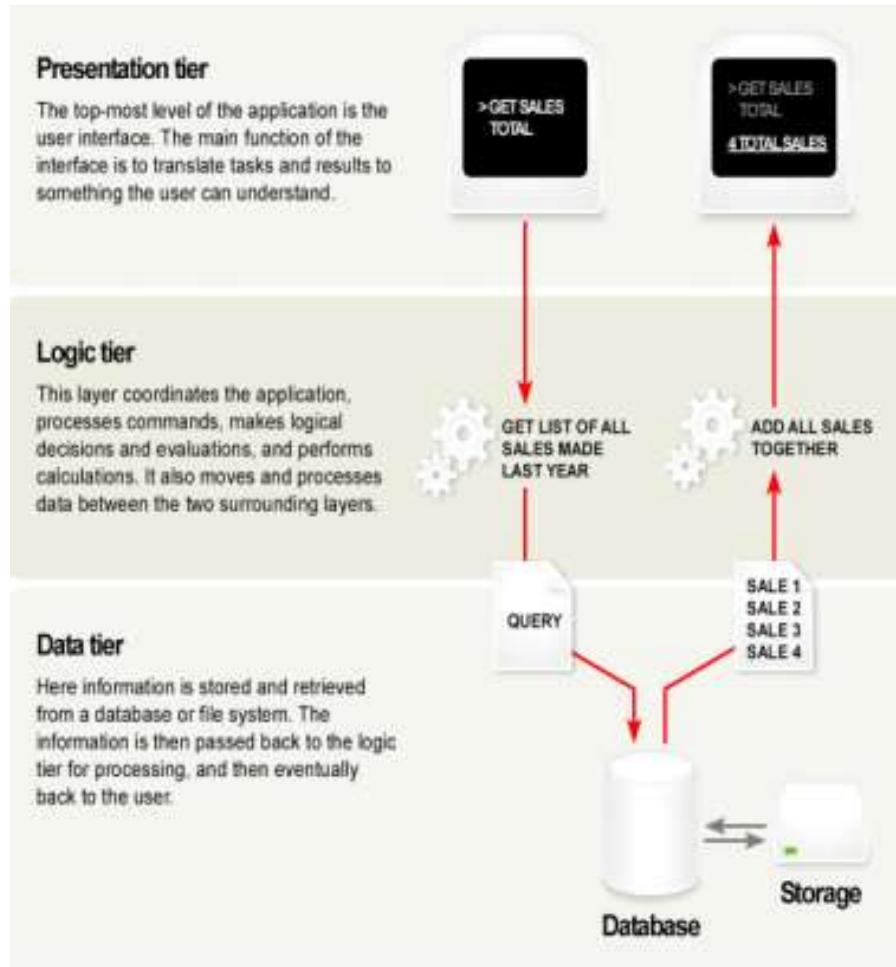
# 3-Tier Architecture



- Each layer can potentially run on a different machine
- Presentation, logic, and data layers are disconnected



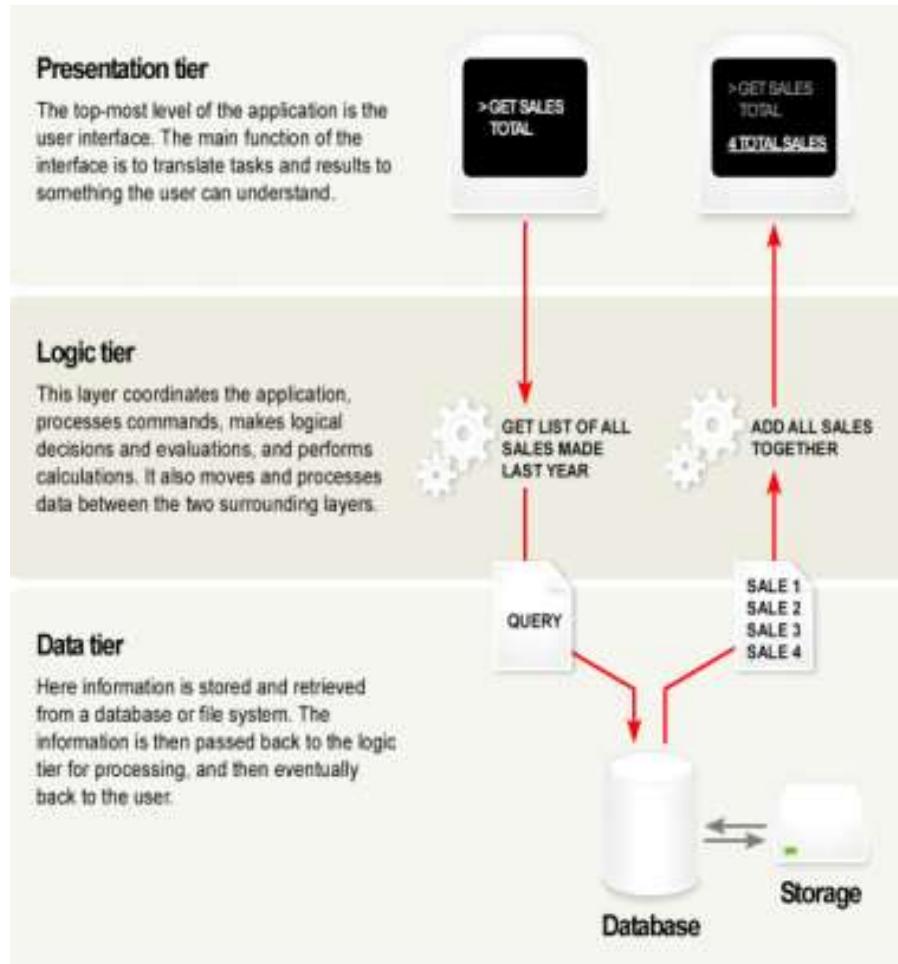
# A Typical 3-tier Architecture



## Architecture Principles

- Client-server architecture
- Each layer (Presentation, Logic, Data) should be independent and should not expose dependencies related to the implementation
- Unconnected layers should not communicate
- Changes in a platform affects only the layer running on that particular platform

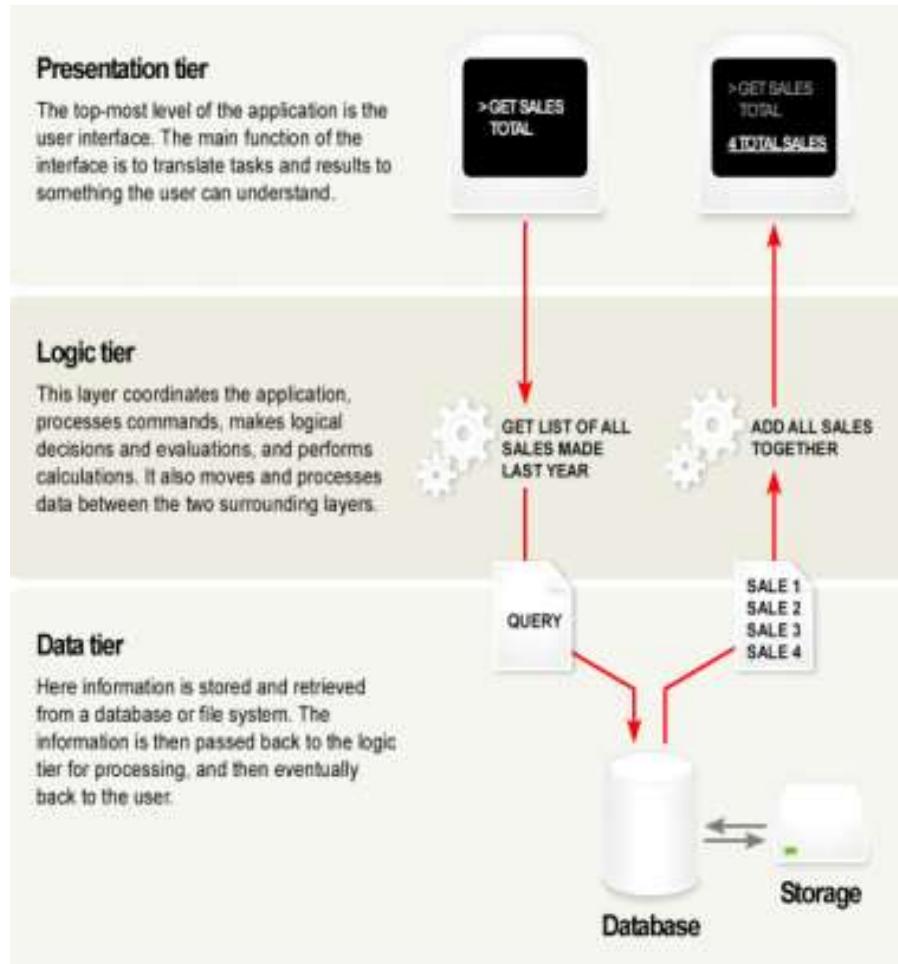
# A Typical 3-tier Architecture (ii)



## Presentation Layer

- Provides user interface
- Handles the interaction with the user
- Sometimes called the GUI, client view or **front-end**
- Should not contain business logic or data access code

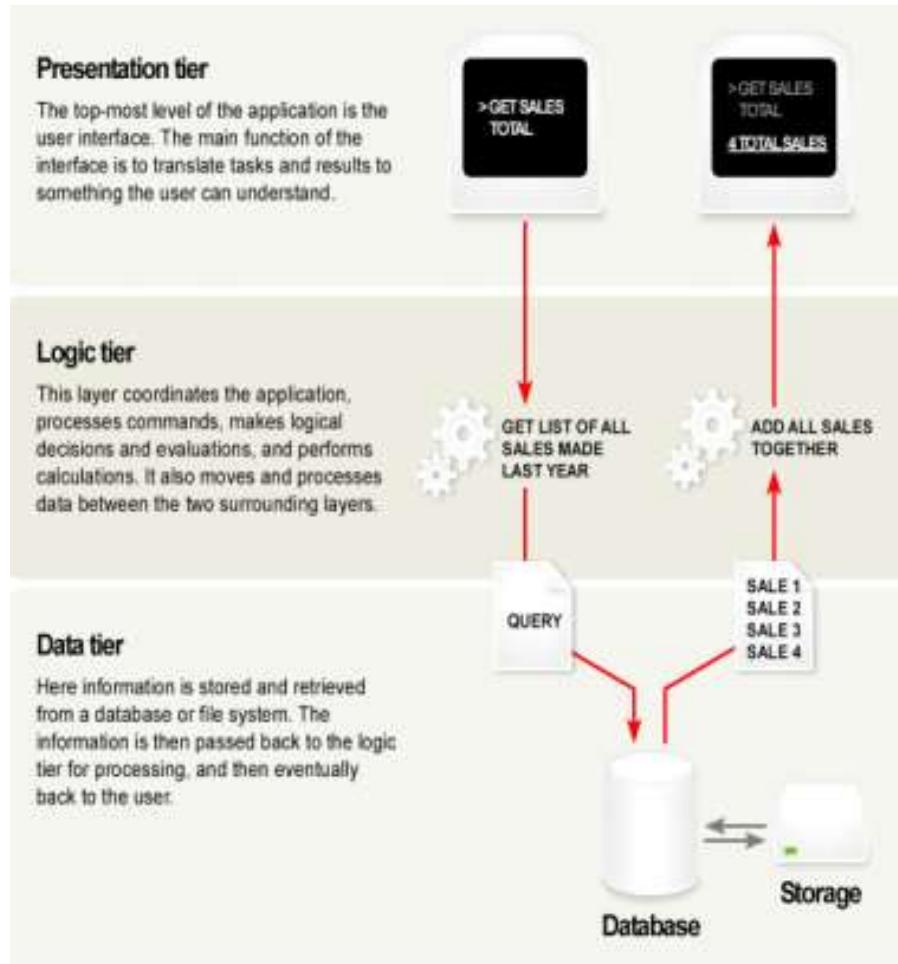
# A Typical 3-tier Architecture (iii)



## Logic Layer

- The set of rules for processing information
- Can accommodate many users
- Sometimes called **middleware/back-end**
- Should not contain presentation or data access code

# A Typical 3-tier Architecture (iv)



## Data Layer

- The physical storage layer for data persistence
- Manages access to DB or file system
- Sometimes called **back-end**
- Should not contain presentation or business logic code

# The 3-Tier Architecture for Web Apps



- Presentation Layer
  - Static or dynamically generated content rendered by the browser (**front-end**)
- Logic Layer
  - A dynamic content processing and generation level application server, e.g., Java EE, ASP.NET, Python Django Framework (**middleware**)
- Data Layer
  - A database, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data (**back-end**)

# 3-Tier Architecture – Advantages



## Independence of Layers

- Easier to maintain
- Components are reusable
- Faster development (division of work)
  - Web designer does presentation
  - Software engineer does logic
  - DB admin does data model



# Web Application Architectures

## Design Patterns

# Design Problems & Decisions



- Construction and testing
  - how do we build a web application?
  - what technology should we choose?
- Re-use
  - can we use standard components?
- Scalability
  - how will our web application cope with large numbers of requests?
- Security
  - how do we protect against attack, viruses, malicious data access, denial of service?
- Different data views
  - user types, individual accounts, data protection

Need for general and reusable solution: **Design Patterns.**

# What is a Design Pattern?

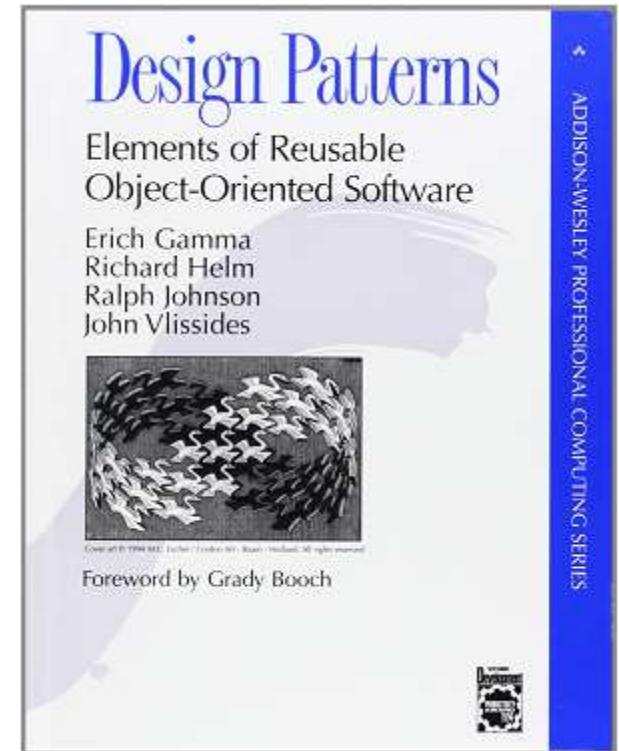


- A general and reusable solution to a commonly occurring problem in the design of software
- A template for how to solve a problem that has been used in many different situations
- NOT a finished design
  - The pattern must be adapted to the application
  - Cannot simply translate into code

# Origin of Design Pattern?



- Architectural concept by Christopher Alexander (1977/79)
- Adapted to OO Programming by Beck and Cunningham (1987)
- Popularity in CS after the book:  
“Design Patterns: Elements of Re-useable Object Oriented Software”, 1994.  
Erich Gamma, Richard Helm,  
Ralph Johnson, John Vlissides
- Now widely-used in software engineering





# Web Application Architectures

## The MVC Design Pattern

# Design Problem



- Need to change the look-and-feel without changing the core/logic
- Need to present data under different contexts (e.g., powerful desktop, web, mobile device).
- Need to interact with/access data under different contexts (e.g., touch screen on a mobile device, keyboard on a computer)
- Need to maintain multiple views of the same data (list, thumbnails, detailed, etc.)



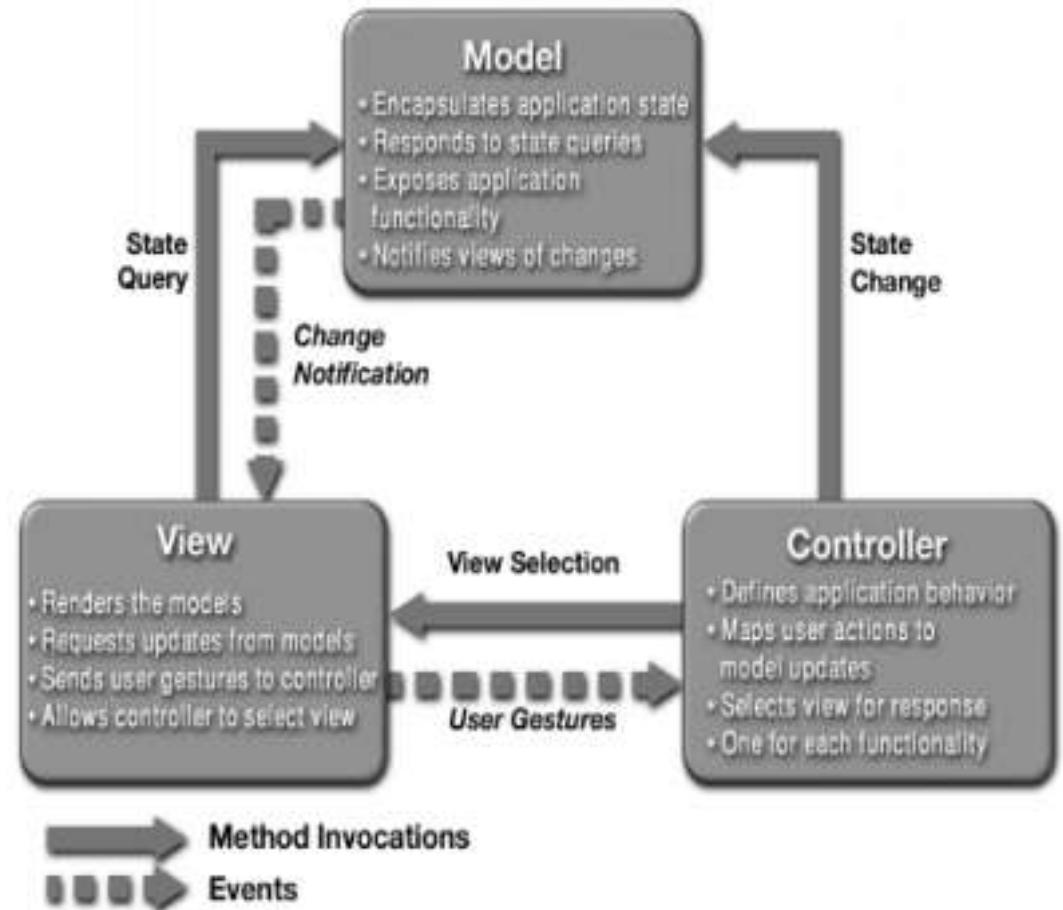
# Design Solution

- Separate core functionality from the presentation and control logic that uses this functionality
- Allow multiple views to share the same data model
- Make supporting multiple clients easier to implement, test, and maintain

# The Model-View-Controller Pattern



- Design pattern for graphical systems that **promotes separation between model and view**
- With this pattern the logic required for data maintenance (database, text file) **is separated** from how the data is viewed (graph, numerical) and how the data can be interacted with (GUI, command line, touch)





# The MVC Pattern

## Model

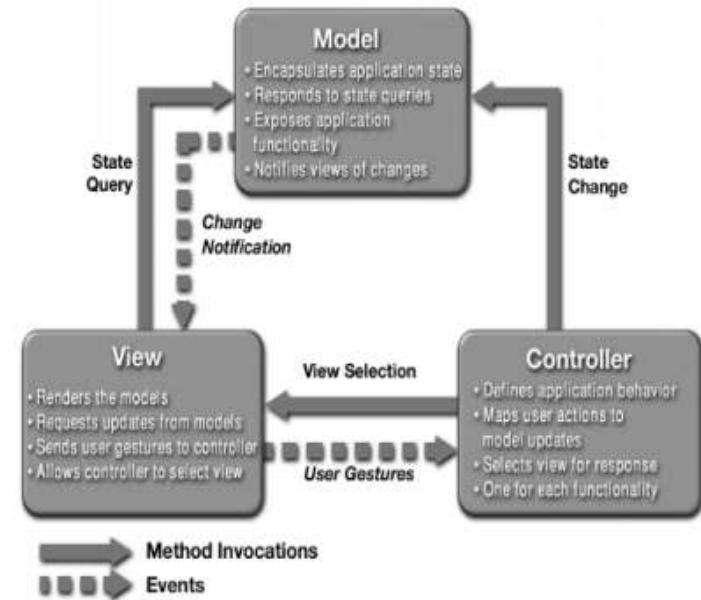
- manages the behavior and data of the application domain
- responds to requests for information about its state (usually from the view) follows instructions to change state (usually from the controller)

## View

- renders the model into a form suitable for interaction, typically a user interface (multiple views can exist for a single model for different purposes)

## Controller

- receives user input and initiates a response by making calls on model objects
- accepts input from the user and instructs the model and viewport to perform actions based on that input





# The MVC Pattern (in practice)

## Model

- Contains domain-specific knowledge
- Records the state of the application
  - E.g., what items are in a shopping cart
- Often linked to a database
- Independent of view
  - One model can link to different views

## View

- Presents data to the user
- Allows user interaction
- Does no processing

## Controller

- defines how user interface reacts to user input (events)
- receives messages from view (where events come from)
- sends messages to model (tells what data to display)

# The MVC for Web Applications



## Model

- database tables (persistent data)
- session information (current system state data)
- rules governing transactions

## View

- (X)HTML
- CSS style sheets
- server-side templates

## Controller

- client-side scripting
- http request processing
- business logic/preprocessing



# MVC Advantages

- Clarity of Design
  - model methods give an API for data and state
  - view and controller design is easier
- Efficient Modularity
  - any of the components can be easily replaced
- Multiple Views
  - many views can be developed as appropriate
  - each uses the same API for the model
- Easier to Construct and Maintain
  - simple (text-based) views while constructing
  - more views and controllers can be added
  - stable interfaces ease development
- Distributable
  - natural fit with a distributed environment

# 3-Tier Architecture vs. MVC Pattern



- Communication
  - **3-Tier:** The presentation layer never communicates directly with the data layer – but, only through the logic layer (linear topology)
  - **MVC:** All layers communicate directly (triangle topology)
- Usage
  - **3-Tier:** Is used in web applications where the client, middleware and data tiers ran on physically separate platforms
  - **MVC:** Historically used on applications that run on a single graphical workstation
    - In a Web Application context, however, its logical components can be decoupled to comply with a true 3-tier architecture