MÔ HÌNH PHẦN MỀM PHÂN LỚP

GV: ThS Phạm Thi Vương
ThS Cao Thái Phương Thanh

CONTENT

- Software architecture
- Layers
- Tiers

SOFTWARE ARCHITECTURE

- *Software architecture is an abstraction, or a high-level view of the system. It focuses on aspects of the system that are most helpful in accomplishing major goals, such as* ***reliability****,* ***scalability****, and* ***changeability****. The architecture explains how you go about accomplishing those goals*

Software architecture is a blueprint of your application

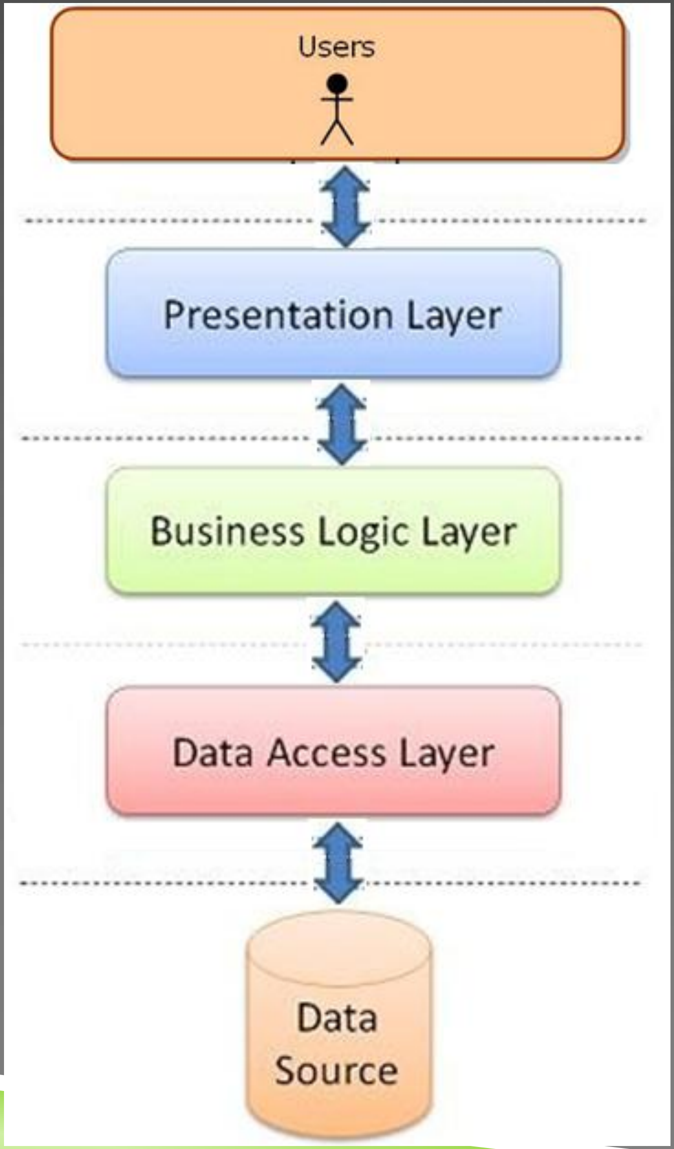
LAYERS AND TIERS

- Logical Separation

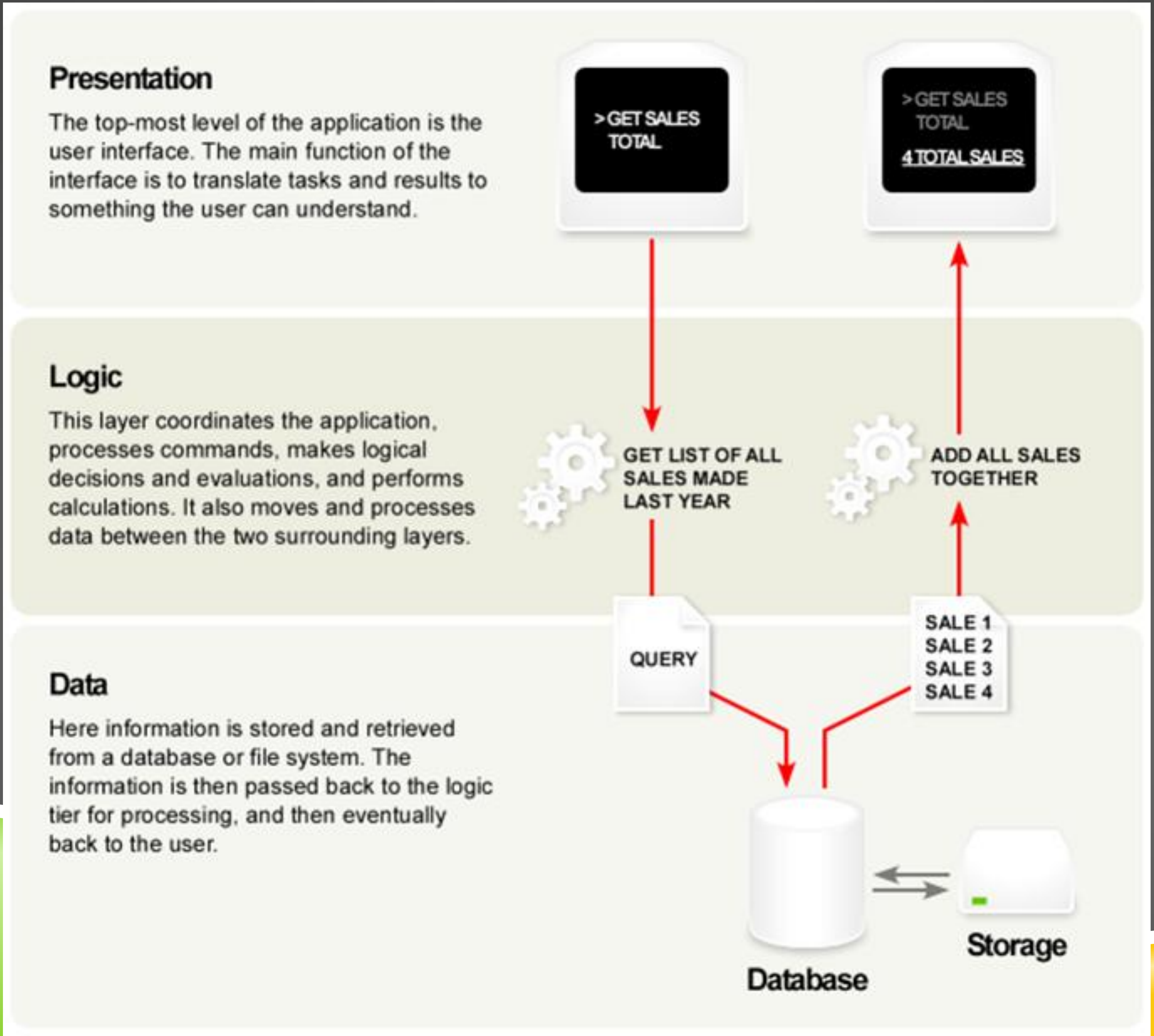
  and

- Physical Separation

MULTI-LAYER ARCHITECTURE


Users
Presentation Layer
Business Logic Layer
Data Access Layer
Data Source

3-TIER ARCHITECTURE



Presentation

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

DATA ACCESS LAYER (DAL)

- A set of classes used to encapsulate data access methods
- CRUD (Create Read Update and Delete) operations as well as any other methods
- Accessing data from a data store (known as Data Layer). DAL's primary job is to communicate with the Data layer, which can be any RDBMS, set of XML files, text files, and so on.
- The DAL layer should not contain any specific logic in its classes, and it should be used like a "utility" or "helper" class to fetch and store data to and from a data store.

DATA ACCESS LAYER

- *Data Access* layer provides an interface between the business logic and the database
- *Data Access* layer interacts with the data management tier to retrieve, update, and remove information
- *Data Access* layer doesn't actually manage or store the data
- *Data Access* logic components may connect to the database directly using a data access API such as ADO.NET

BUSINESS LOGIC LAYER (BLL)

- Contains the business logic and set of operational rules particular to the application and talks to the data access layer (DAL) to:
  - fetch data on which it has to apply rules
  - save updated data after applying rules to it
  - perform operations and validate data
- BLL usually presents the data to the higher Layers (like a GUI layer) after performing business rules on it. This layer may also include error handling, logging, and exception handling strategies, besides encapsulating all the business rules of the project.

BUSINESS LOGIC LAYER (CONT.)

- Some main methods need on a object:
  - Create new data
  - Update data
  - Delete data
  - Check data exists
  - Get all data in table
  - Get detail data with correlative key

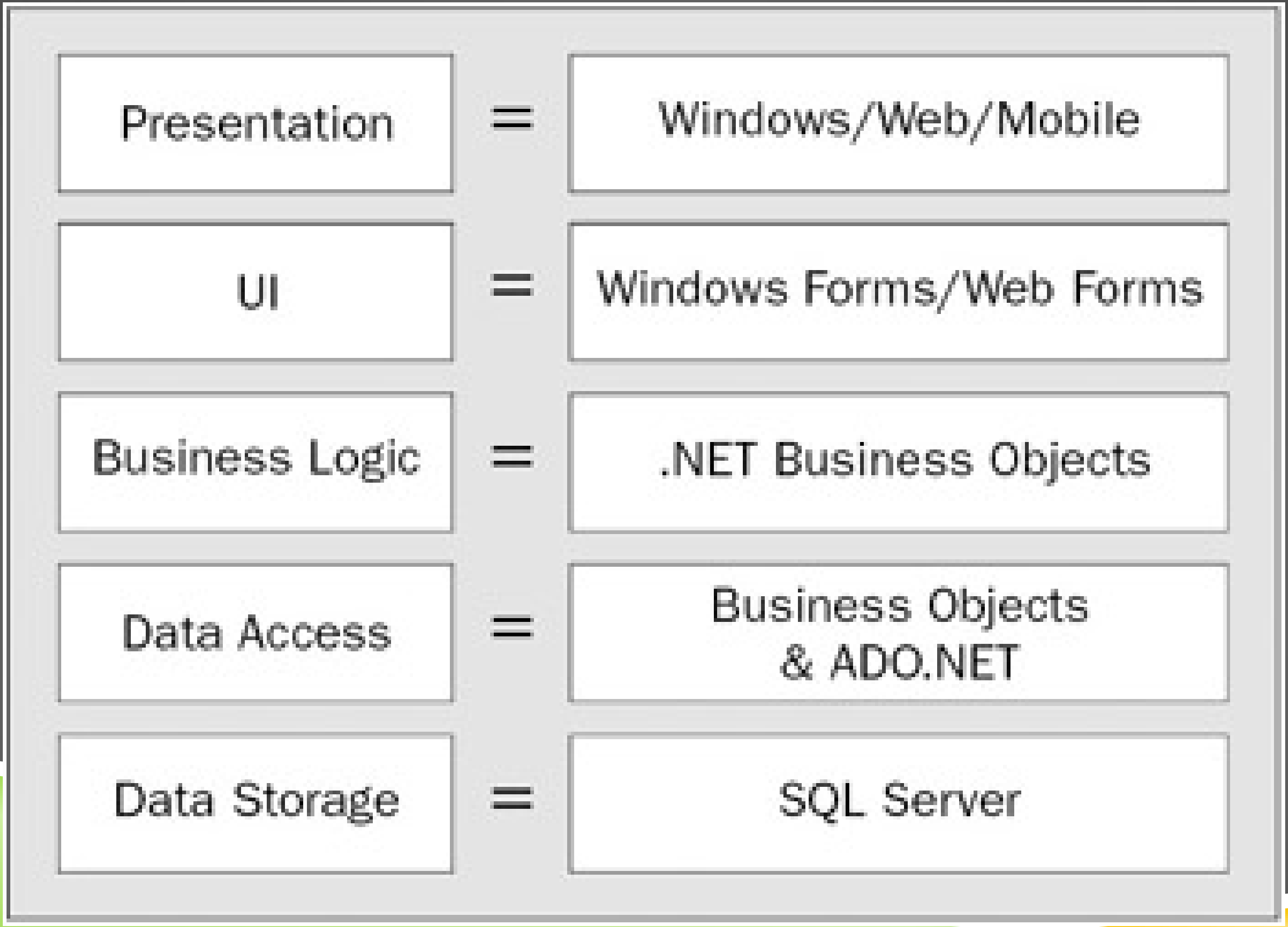
UI LAYER

- **UI layer** contains the graphical display components
- Functionality
  - Restrict the types of input a user can enter
  - Perform data entry validation
  - Perform simple mapping and transformations of the information provided by the user controls to values needed by the underlying components

N-LAYER PROJECT OF **THICK-CLIENT**

- User Interface forms (Window forms or Web forms) as the Presentation layer
- Business logic code as the Business Layer (BL)
- Data access code as the Data Access Layer (DAL)
- The physical database as the Data Layer (DL)

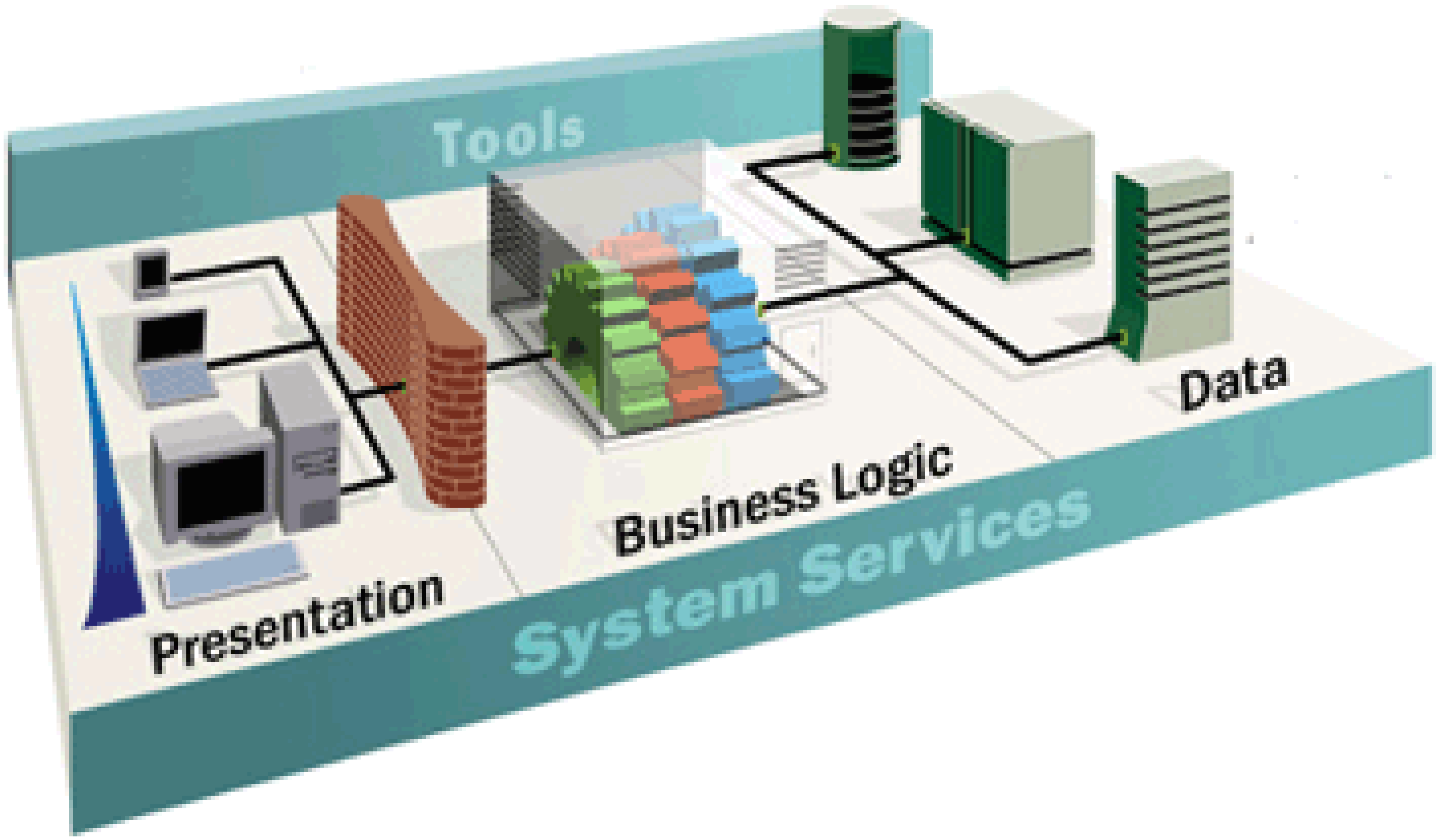
LOGIC MODEL
FRAMEWORK DESIGN



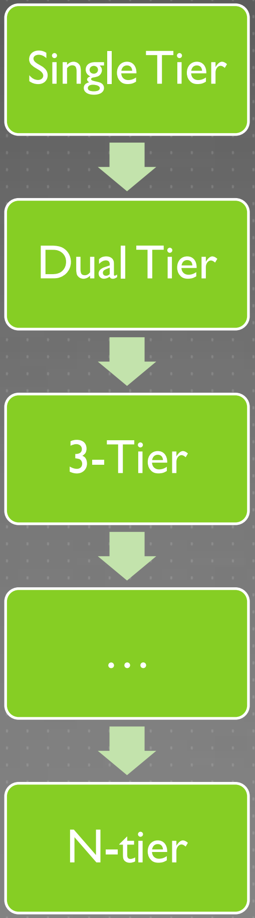
Presentation	=	Windows/Web/Mobile
UI	=	Windows Forms/Web Forms
Business Logic	=	.NET Business Objects
Data Access	=	Business Objects & ADO.NET
Data Storage	=	SQL Server

MULTI-TIER ARCHITECTURE

PHYSICAL SEPARATION

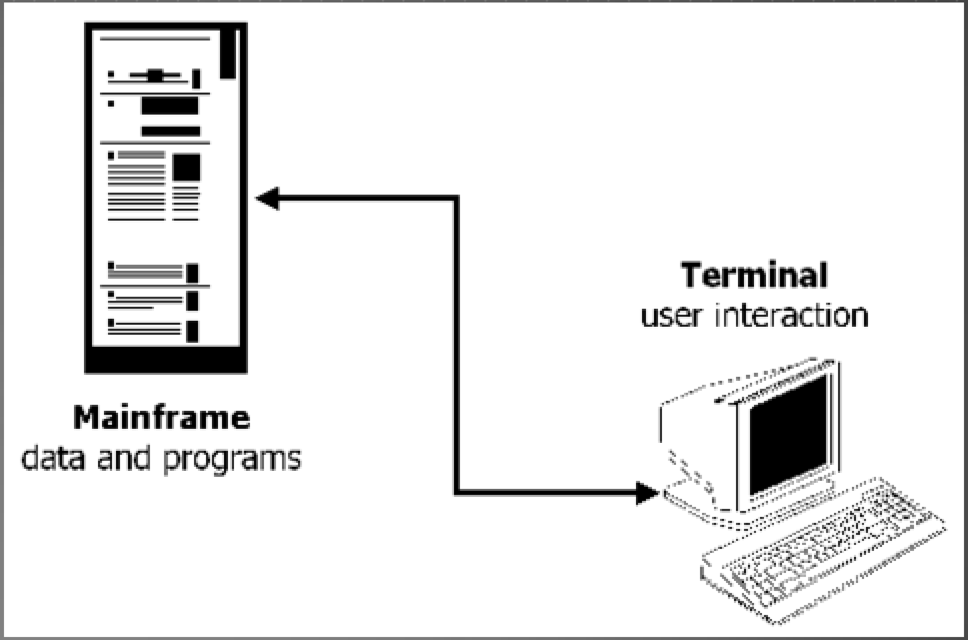


EVOLUTION TO THE 3-TIER ARCHITECTURE



SINGLE TIER ARCHITECTURE

- Time of Huge "Mainframe"
- All Processing in Single Computer
- All Resources Attached to the same Computer
- Access Via Dumb Terminals

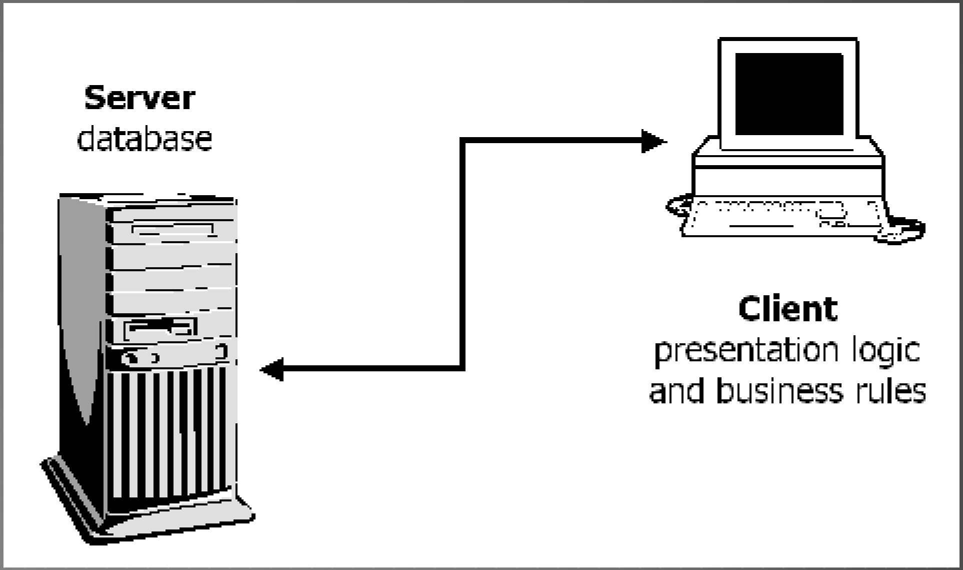


SINGLE TIER – ADVANTAGES & DISADVANTAGES

- Advantages
  - Simple
  - Uncomplicated
  - Quick
- Disadvantages
  - Very Expensive
  - Difficult to maintain
  - Difficult to reuse

DUAL TIER ARCHITECTURE

- The Personal Computer
- Necessity of Providing Personal Software
- The Client Server Model was Born!!
- Logical System Components – Most of which are on the Client

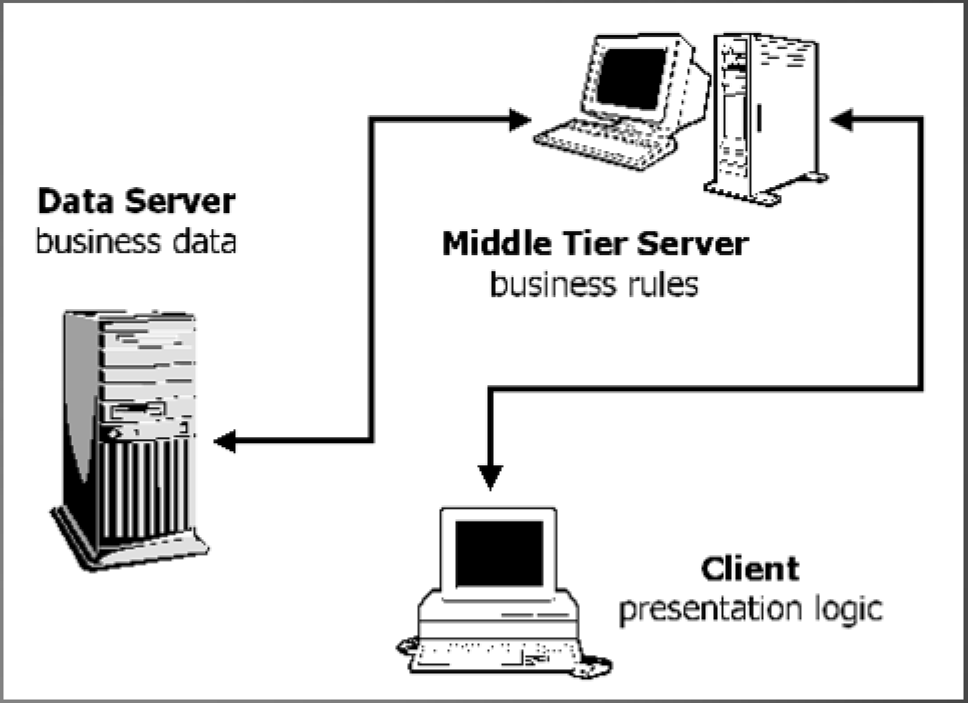


DUAL TIER – ADVANTAGES & DISADVANTAGES

- Advantages
  - Less Expensive than Mainframe
- Disadvantages
  - The Connections to the Database Server are very Expensive
  - One can only connects a limited number of users to a server before Database Server spends more time managing connections than processing requests
  - Cost-ineffective. Many users only use their connections 2-3% of the time.

3-TIER ARCHITECTURE

- These Applications runs on the Traditional Client/Server Model But from a Application server.
- Client only Displays the GUI and data, but has no part in producing results
- Database Server Serves to few Connections



3-TIER ARCHITECTURE

- A three-way interaction in a client/Server environment
  - The User Interface is stored in the Client.
  - The Business Application Logic is Stored in one or more Servers.
  - The Data is Stored in a Database Server.

3-TIER ADVANTAGES

- Scalability
  - The Application Servers can be deployed on many machines
  - The Database no longer requires a connection from every client.
- Reusability
  - If a standard object is employed, the specific language of implementation of middle tier can be made transparent.
- Data Integrity
  - The middle tier can ensure that only valid data is allowed to be updated in the database.

3-TIER ADVANTAGES

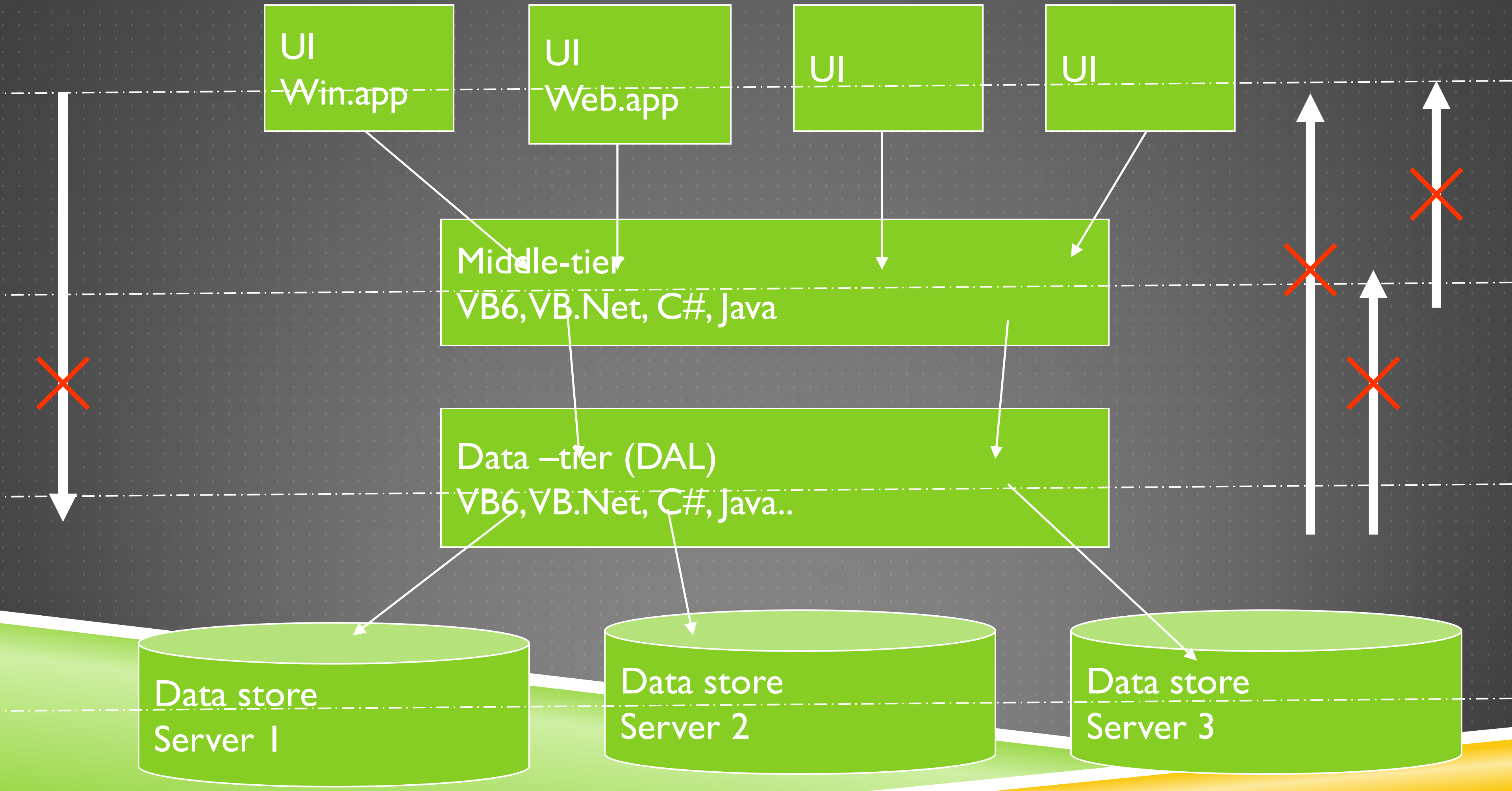
- Improved Security
  - Since the client doesn’t have direct access to the database, Data layer is more secure.
  - Business Logic is generally more secure since it is placed on a secured central server.
- Reduced Distribution
  - Changes to business logic only need to be updated on application servers and need not to distributed on clients
- Improved Availability
  - Mission Critical Applications can make use of redundant application servers and redundant application servers, so it can recover from network of server failures.

3-TIER DISADVANTAGES

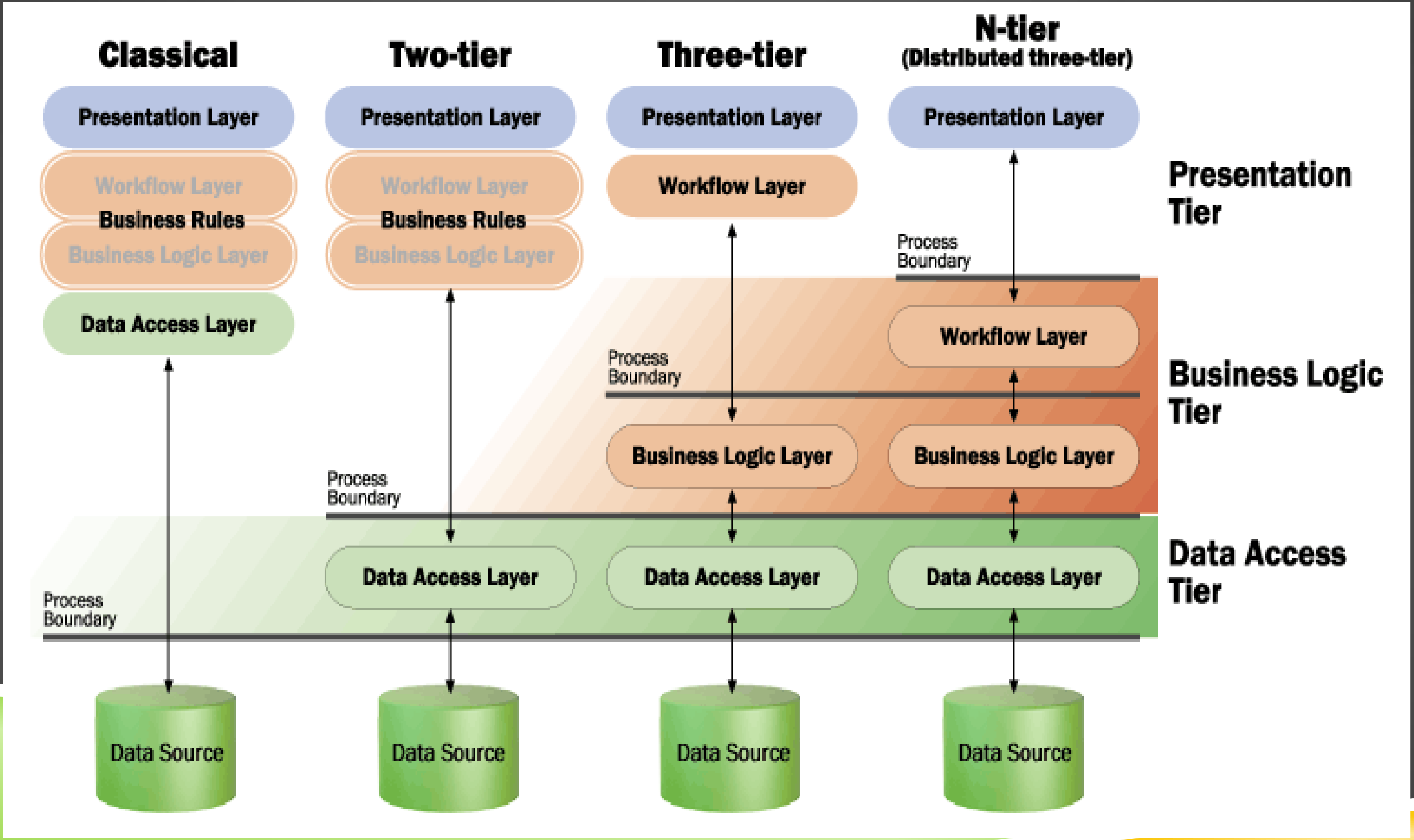
- Increased Complexity / Effort
  - In General 3-tier Architecture is more complex to build compared to 2-tier Architecture.
  - Point of Communication are doubled.

SOFTWARE ARCHITECTURE LAYERS & TIERS

SOFTWARE ARCHITECTURE IN LOGIC AND PHYSIC



LOGICAL & PHYSICAL SEPARATION



WEB BASED APPLICATIONS

- We have a built-in 3-tier architecture by default.
  - The presentation tier is the client-side browser (instead of Windows forms),
  - the code(assuming you have web forms, BL, and DAL in one assembly) is the Application tier,
  - the physical database is the Data tier.

SINGLE TIER—SINGLE LAYER MODEL

- .NET Project (ASPX project) compiling into a DLL in the /bin folder and under a single namespace: MyApp
- No. of project files: 1
- No of namespaces: 1

SINGLE TIER—TWO LAYER MODEL

- The UI code into one namespace, and the BL and DAL into another namespace.
- ASP.NET Web Project that has two folders:
- **Code:** This folder will have class files containing business logic and data
- access code under a single namespace, say MyApp.Code
- **Web:** This folder will have the user controls, ASPX pages, and other presentation-related code under the namespace, say MyApp.Web

SINGLE TIER—THREE LAYER MODEL

- In this model, we logically break BL and DAL in different namespaces, introducing cleaner code separation
- ASP.NET Web Project that has logical separation between presentation, business
- logic and data access code:
- All presentation code will be under the MyApp.Web namespace (Layer 1).
- Furthermore, the single project can have two folders:
- Business (Layer 2): for business logic code, with namespace
- MyApp.Code.Business
- DAL (Layer 3): for data access code, with namespace MyApp.
- Code.DAL

TWO TIER MODEL

- create two projects, one normal web project for UI code, and another class library project for the BL and DAL code.
- The solution will have:
- ASP.NET Web Project having GUI and presentation code (Tier 1)
- A class library project having business logic and data access coding under a
- single namespace, MyApp.Code; no separate namespaces for business logic
- and data access code (Tier 2)

TWO TIER—TWO LAYER MODEL

- The solution will have:
- ASP.NET Web Project having Presentation Layer coding in ASPX and ASCX
- files, under the namespace, MyApp.Web (Tier 1)
- A class library project having two folders (Tier 2):
- Business: for business logic code, with namespace MyApp.
- Code.Business (Layer 1)
- DAL: for data access code, with namespace MyApp.Code.DAL
- (Layer 2)

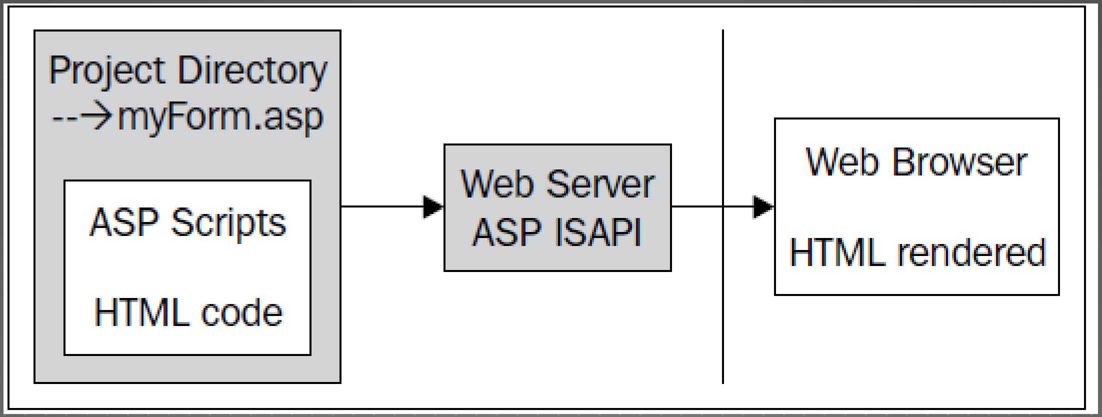
THREE TIER MODEL

- The solution will have:
- ASP.NET Web Project having Presentation Layer coding in ASPX and ASCX files, under namespace MyApp.Web (Tier 1)
- A class library project having business logic code, with namespace, MyApp.Code.Business (Tier 2)
- A class library project DAL for data access code, with namespace, MyApp.Code.DAL (Tier 3)

EXAMPLES

1-TIER 1-LAYER ARCHITECTURE

CLASSIC ASP STYLE: INLINE CODING

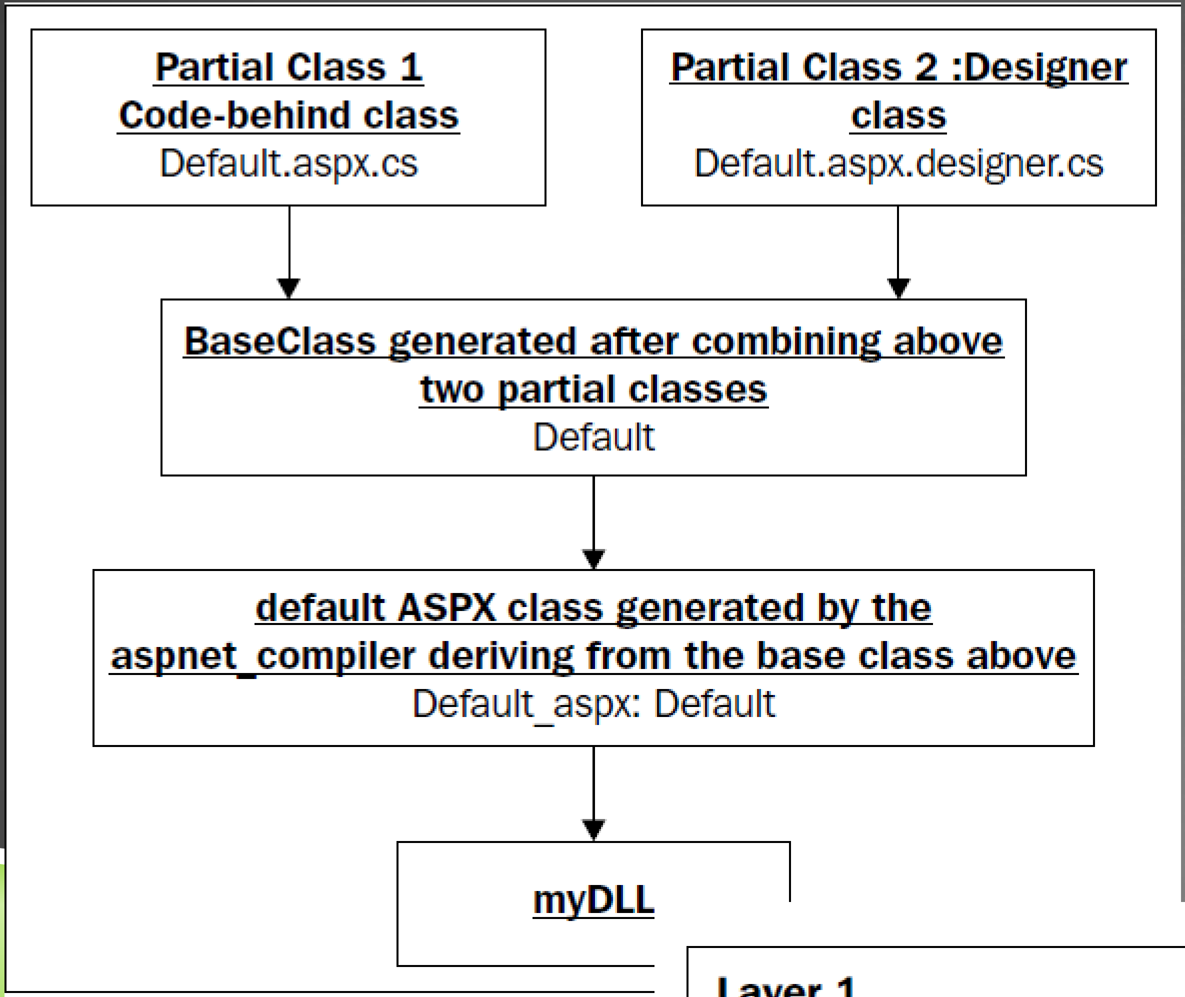


- Demo

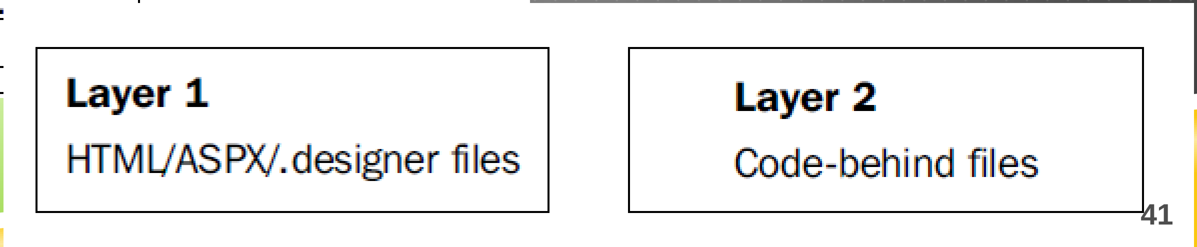
INLINE CODING - DISADVANTAGES

- **No separation of business logic, data access code, and presentation (HTML)**
- **Code re-use**
- **Source Code Control (SCC) problems**
- **Compilation model**
- **Maintenance issue**

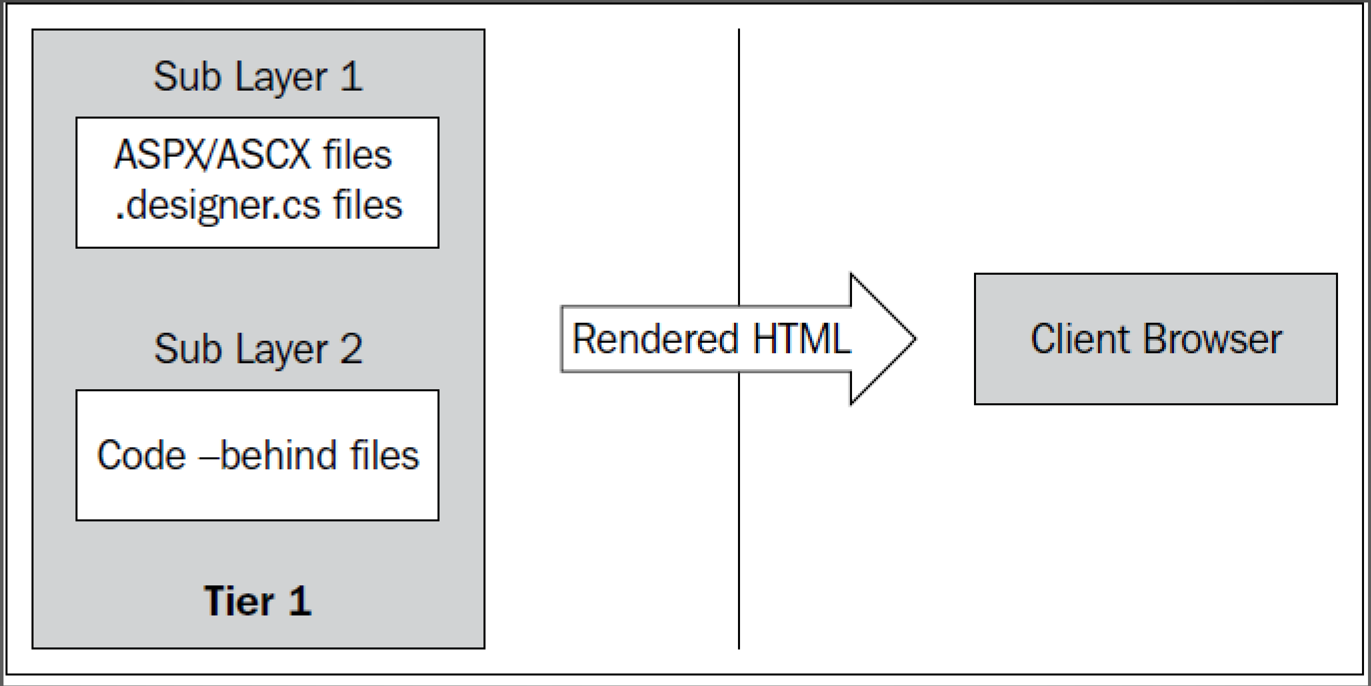
CODE-BEHIND MODEL


Partial Class 1
Code-behind class
Default.aspx.cs
Partial Class 2 :Designer class
Default.aspx.designer.cs
BaseClass generated after combining above two partial classes
Default
default ASPX class generated by the aspnet_compiler deriving from the base class above
Default_aspx: Default
myDLL

- Demo

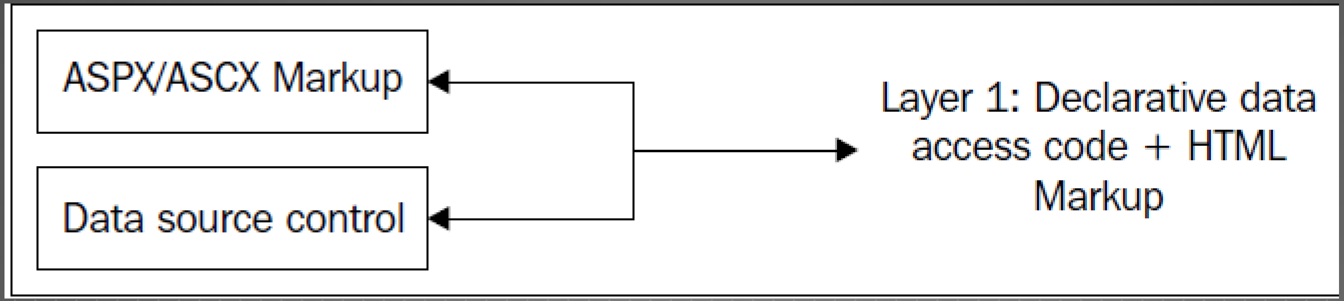


CODE-BEHIND MODEL



- From the given diagram, we can see that we still have one single physical DLL, but the UI code itself is logically separated into two layers — one in the markup code and the other in the code-behind file

DATA SOURCE CONTROLS



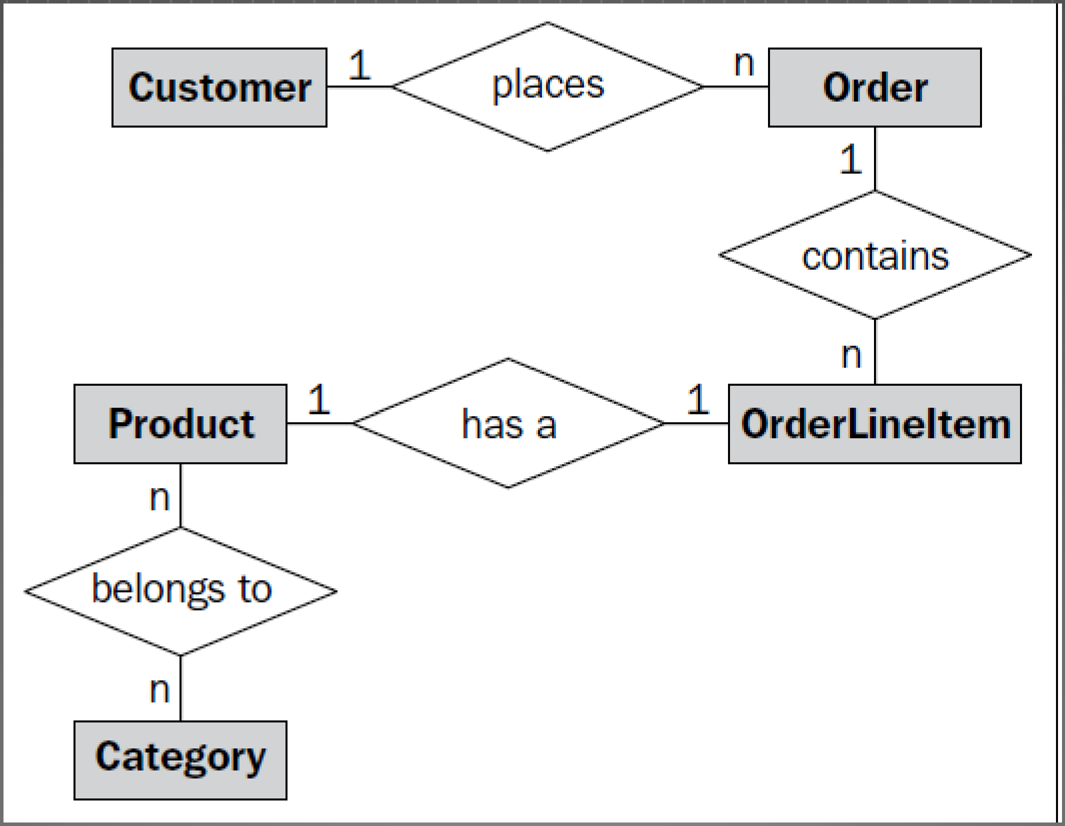
- Demo

DATA SOURCE CONTROLS

- Good for turning out applications quickly
- But we lose finer control over data-access

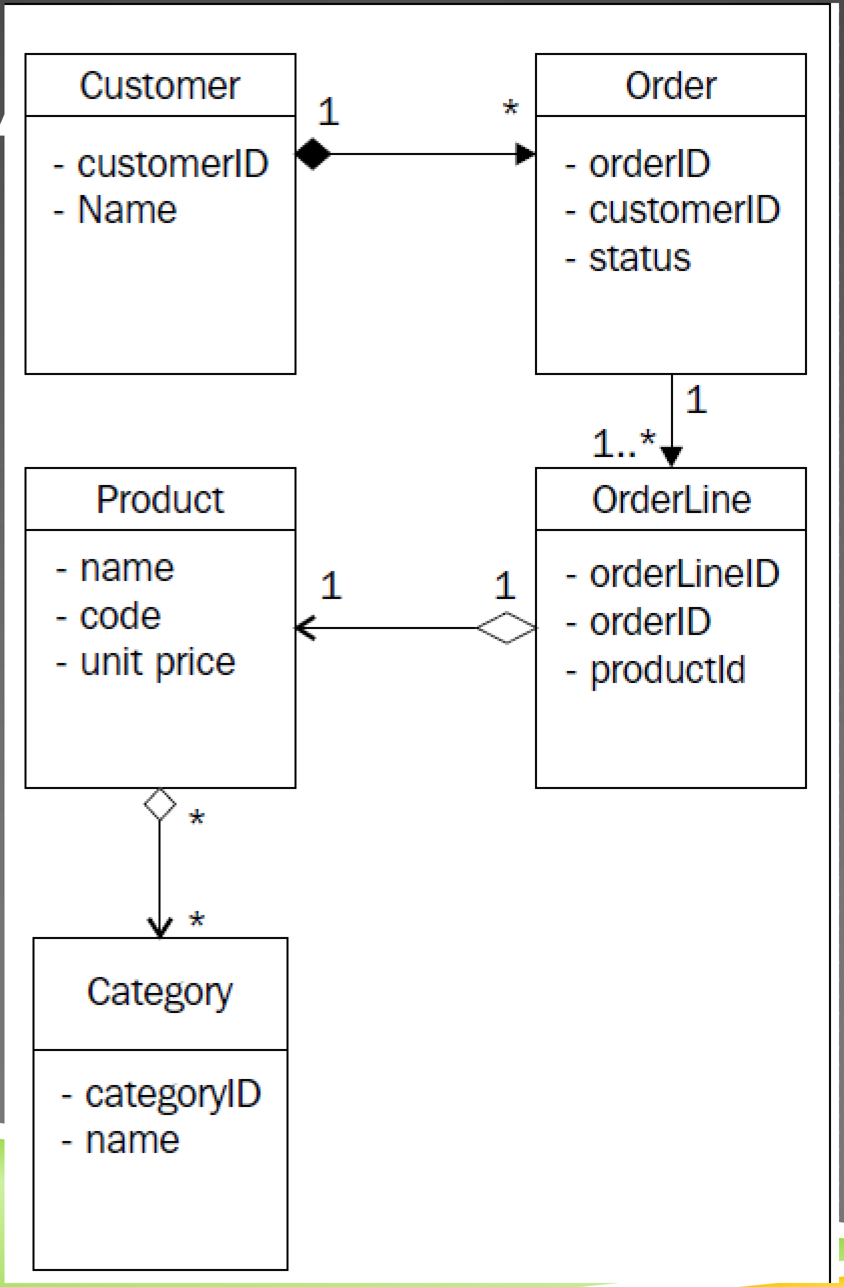
1-TIER N-LAYER ARCHITECTURE

ER - DIAGRAM



Customer
1
places
n
Order
1
contains
n
Product
1
has a
1
OrderLineItem
n
belongs to
n
Category

CLASS DI


Customer
- customerID
- Name
1
*
Order
- orderID
- customerID
- status
1
1..*
Product
- name
- code
- unit price
1
1
OrderLine
- orderLineID
- orderID
- productId
*
*
Category
- categoryID
- name

2-LAYER ARCHITECTURE

- UI-layer with ASPX and code-behind classes
- Data access classes under a different namespace but in the same project (DAL)

- Demo

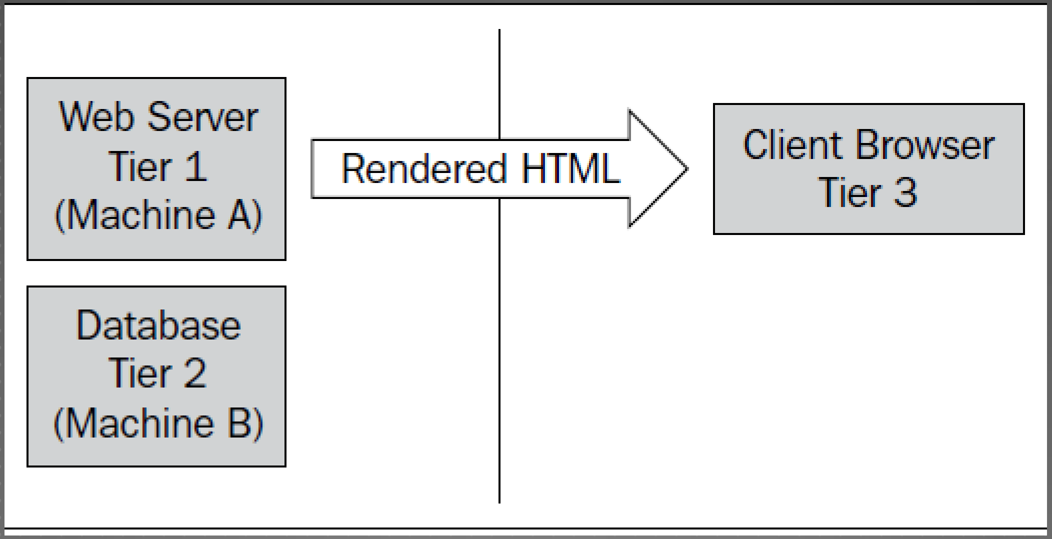
3-LAYER ARCHITECTURE

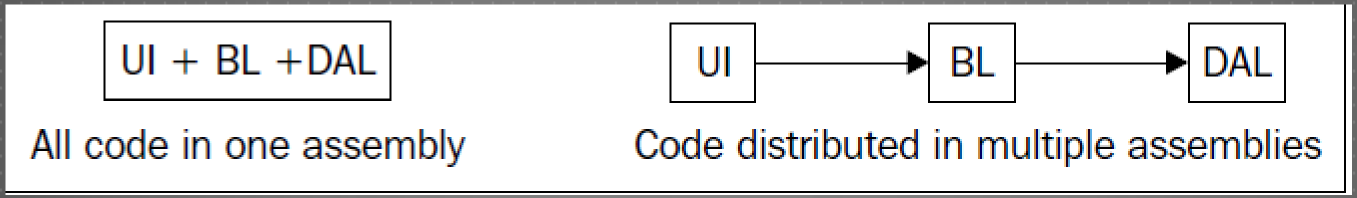
- UI-layer with ASPX and code-behind classes
- Business classes under a different namespace but in the same project (BL)
- Data access classes under a different namespace but in the same project (DAL)

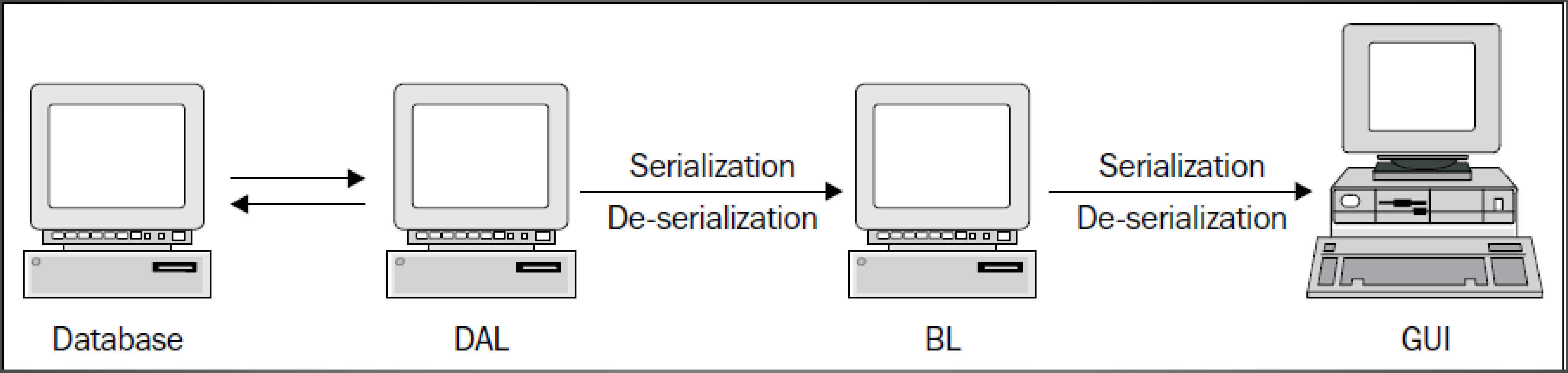
- Demo

N-TIER M-LAYER ARCHITECTURE

3 - TIER


Web Server
Tier 1
(Machine A)
Database
Tier 2
(Machine B)
Rendered HTML
Client Browser
Tier 3


UI + BL +DAL
All code in one assembly
UI
BL
DAL
Code distributed in multiple assemblies


Database
DAL
Serialization
De-serialization
BL
Serialization
De-serialization
GUI

TIERS

- Presentation tier: client-side browser
- UI tier: Web project having ASPX/ASCX, code-behind files
- Data access and business logic tier: in a separate class library project
- [Data tier: the physical database]

DO IT

1. Create a new ASP.NET Web project in Visual Studio 2008 and name it **OMS.WebTier**.

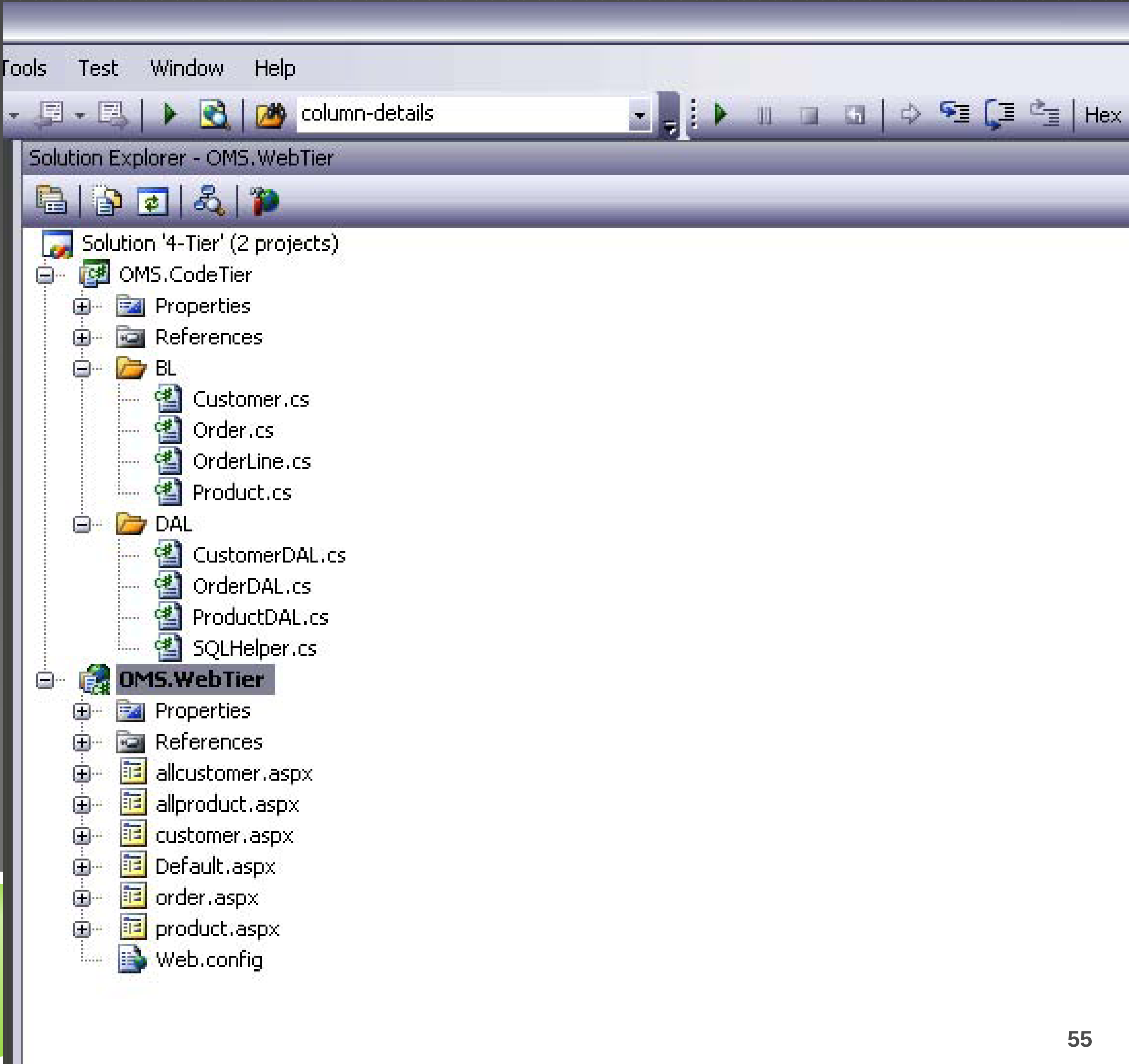
2. This main web project contains UI classes (aspx files) of the 3-layers projects.

3. Create a new class library project named **OMS.CodeTier** that will hold the business logic as well as data access files.

4. Move all of the files from the BL and DAL folders we used in the 3-layers project into this new class library project. We just need to modify the namespaces used (we will see the code to do this soon).

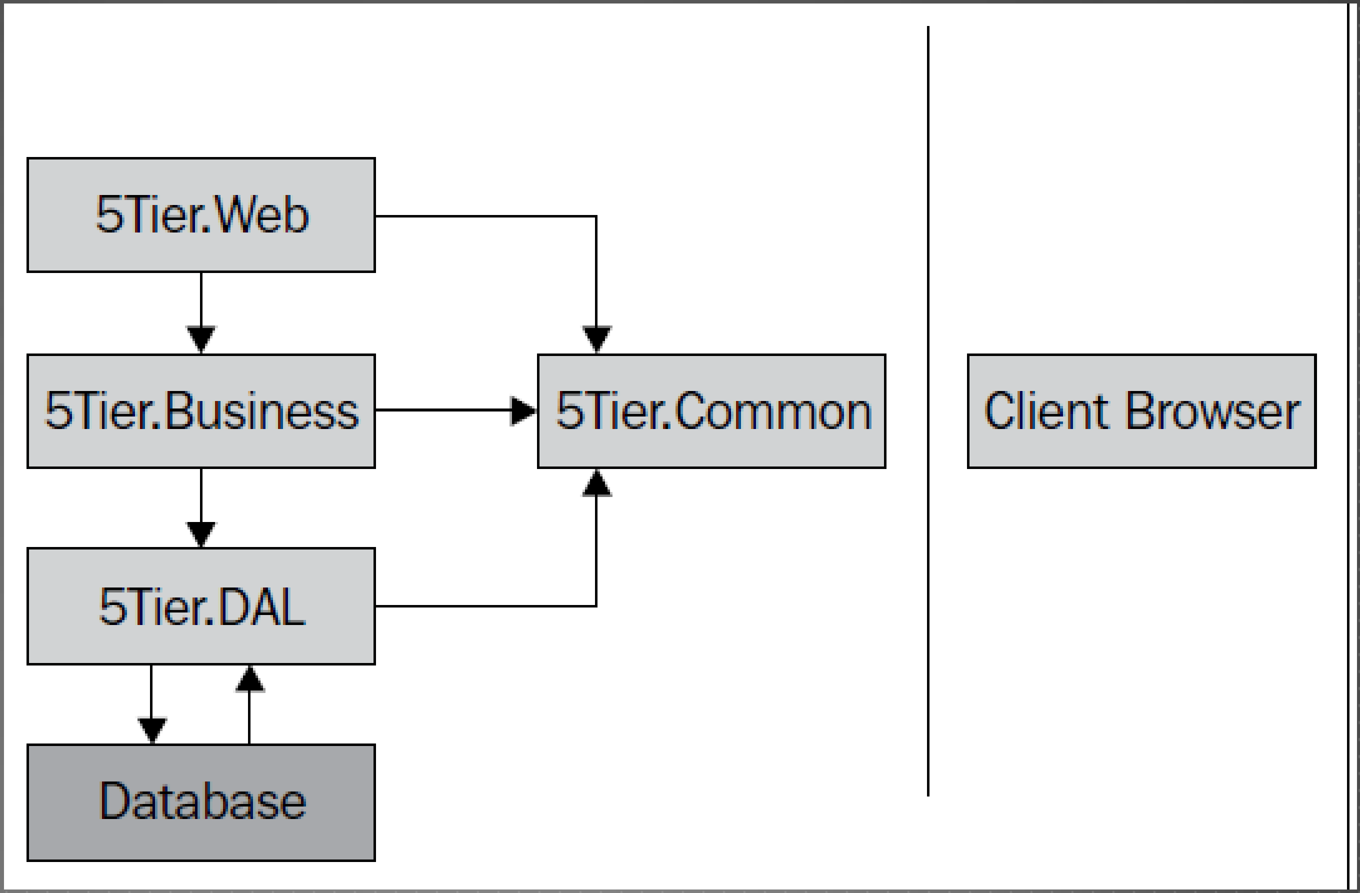
5. Add a reference to this class library project in the main web project.

- Demo



```
Solution '4-Tier' (2 projects)
  OMS.CodeTier
    Properties
    References
    BL
      Customer.cs
      Order.cs
      OrderLine.cs
      Product.cs
    DAL
      CustomerDAL.cs
      OrderDAL.cs
      ProductDAL.cs
      SQLHelper.cs
  OMS.WebTier
    Properties
    References
    allcustomer.aspx
    allproduct.aspx
    customer.aspx
    Default.aspx
    order.aspx
    product.aspx
    Web.config
```

N-TIERS


5Tier.Web
5Tier.Business
5Tier.Common
Client Browser
5Tier.DAL
Database

- Demo