Server side Web Development

Lecture 4

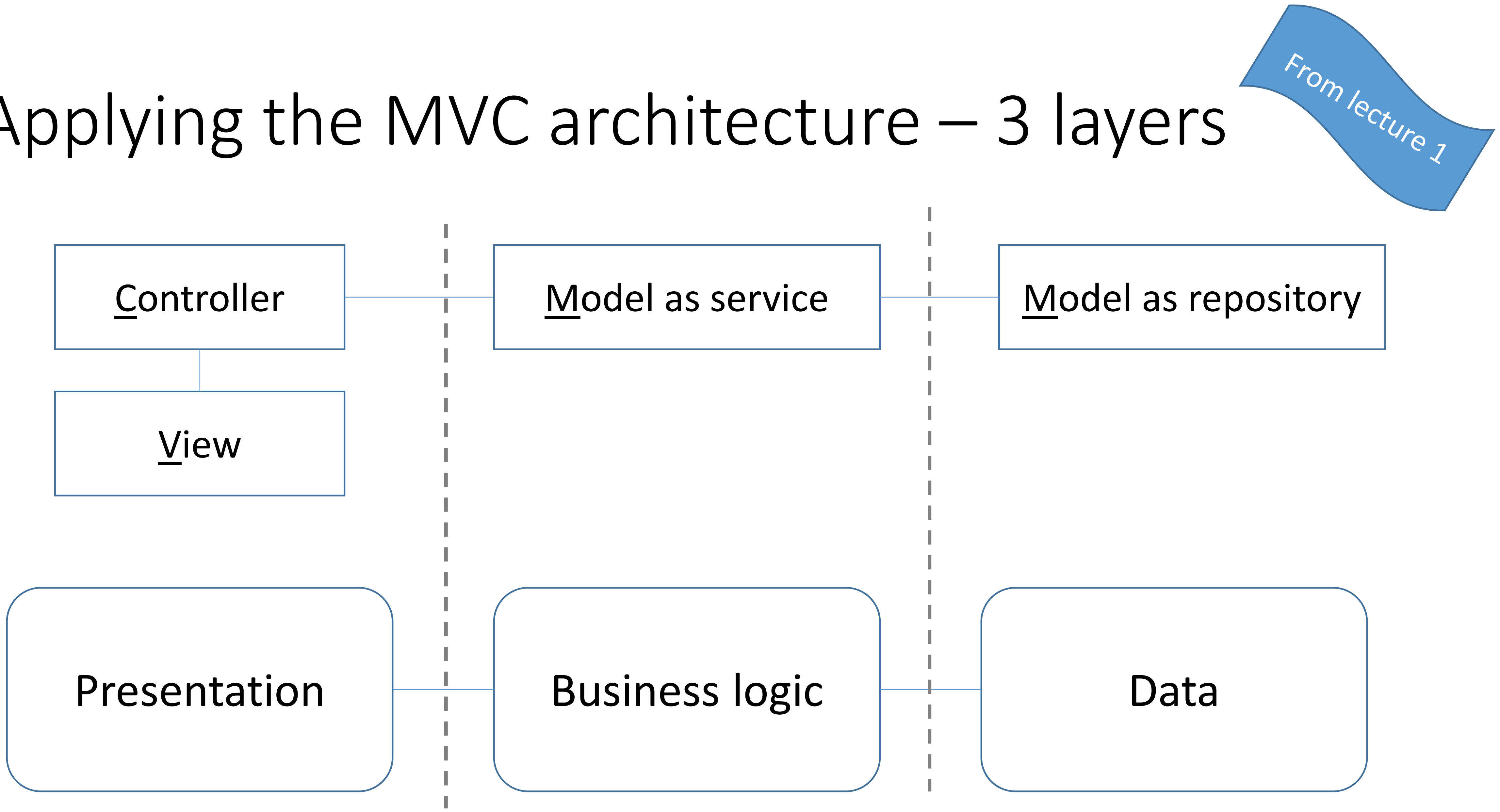
Web architectures
Introduction of the Library database

Outline

- About web architectures
- Database access the traditional way
- A little about design patterns

- Library database (In SQL Server Management Studio)
  - ER-model
  - Script

Applying the MVC architecture – 3 layers



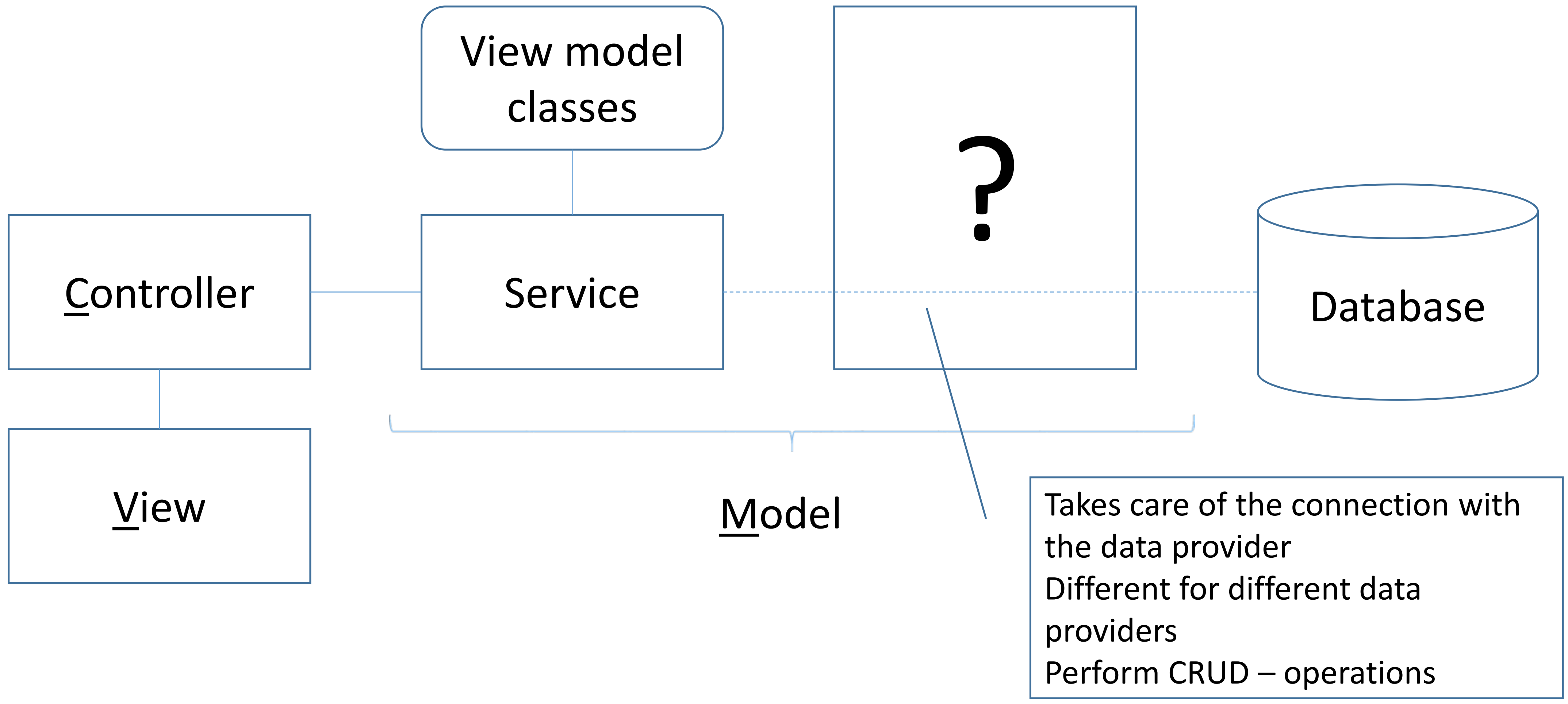
Why divide in three layers?

- Single Responsibility Principle (SRP)
  - A design pattern
  - Each layer should do one and only one thing
  - Independent layers make it easier to:
    - Extend or further develop an application
    - Exchange data providers
    - Connect to multiple data providers (provider model design pattern)
- Requires more initial thinking / work
  - Pays off in large projects
  - Might seem unnecessary in small projects
  - Projects might start small and end up large

From model to data

- The service layer
  - Holds the business logic
    - Rules and operations that regulate how data is used by the users
  - Consists of View model classes in order to represent data for presentation
    - Properties and methods
- The repository layer
  - Has responsibility to connect to a data provider
    - A relational or other type of database, data files etc.
  - Has Entity classes
    - Matches the data in the data provider (Relations and interconnecting relationships)
  - Performs load and mapping of data from data provider to the entity classes

Applying the MVC architecture – 3 layers

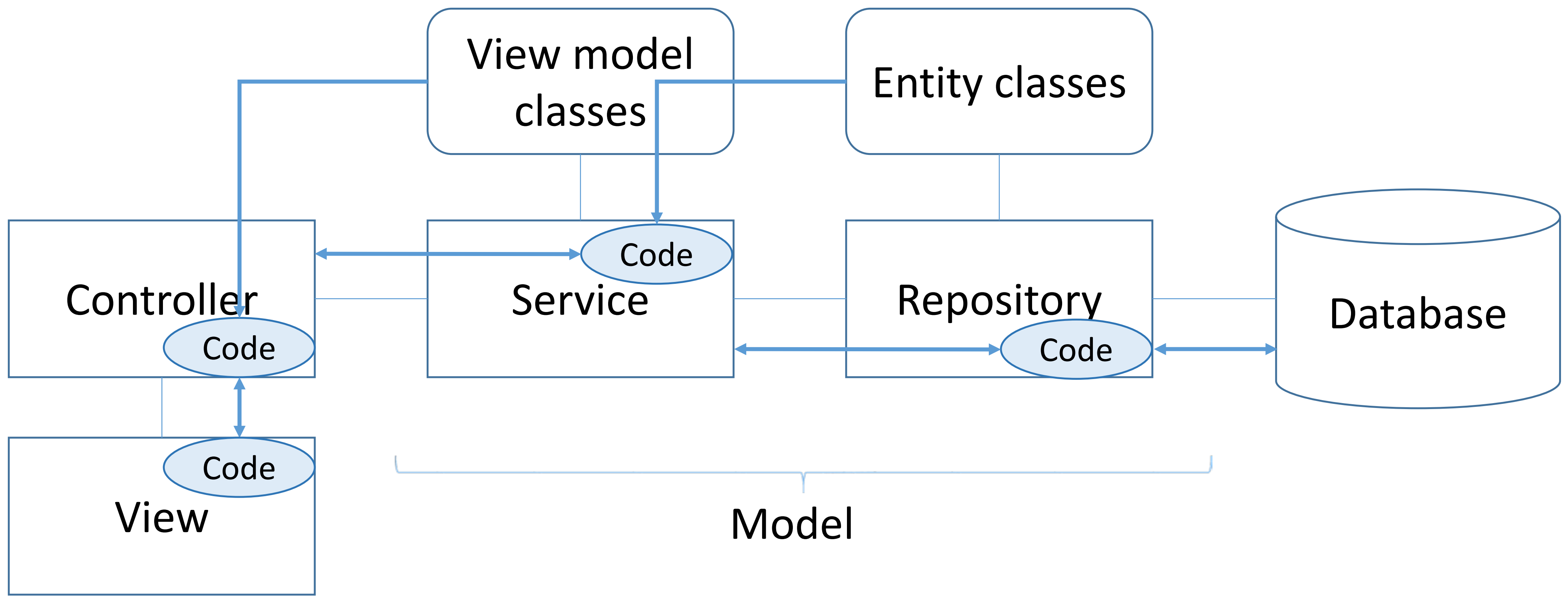


Takes care of the connection with the data provider
Different for different data providers
Perform CRUD – operations

Connecting to the Repository/Data provider

- Pure ADO.NET
    - Traditional old style way, embedding SQL-queries in code and writing the mapping to entity classes
- Object Relational Mapping (ORM) technology
    - Entity Framework (MS) – The dominating technique used today
    - LINQ to SQL (MS)
    - nHibernate (Open Source)

Web MVC architecture

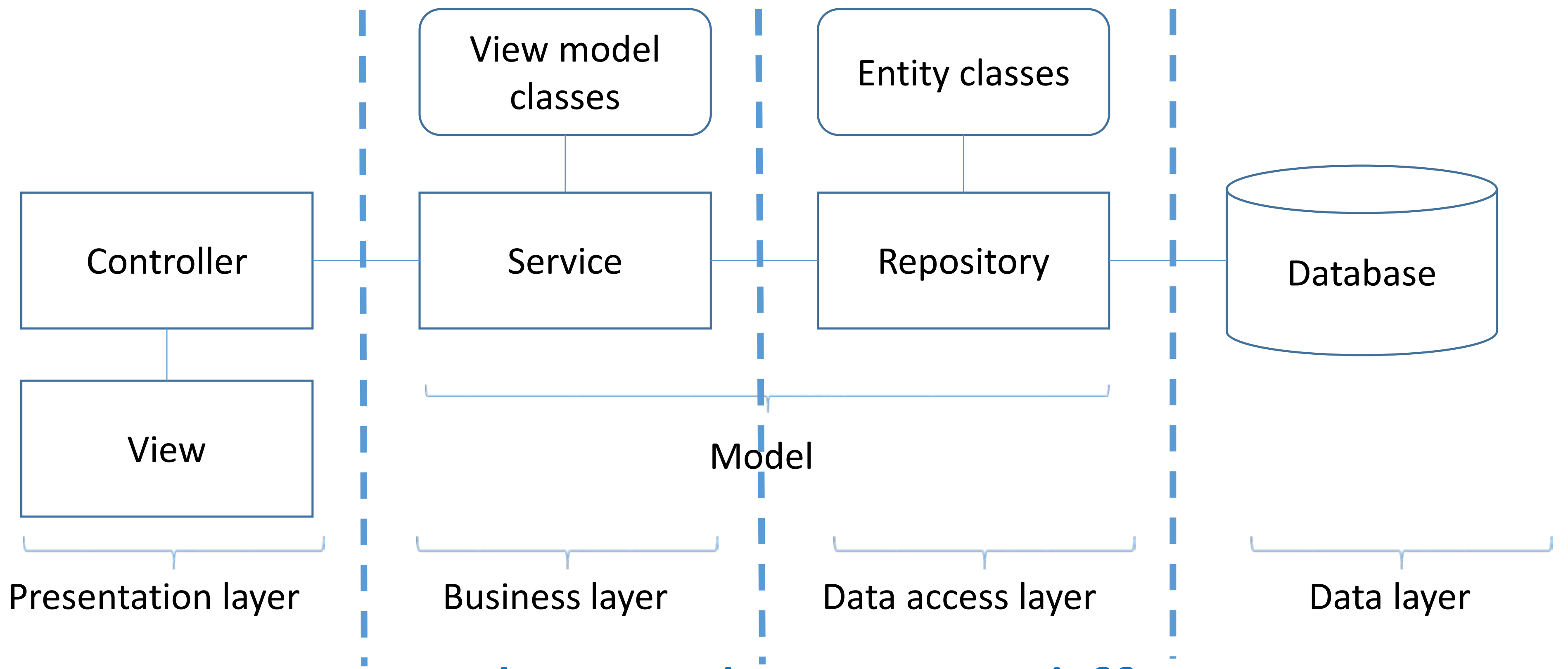


Which code to write where?

- Presentation Layer
    - The controller (is the conductor)
        - Calls the appropriate domain service methods (for retrieving needed data)
        - Passes retrieved data to the correct view
    - The View
        - Prepares the HTML
        - Using Razor syntax to create the dynamic HTML
        - Can do an Action-call in order to (via the controller) call on methods in the service layer
- Service Layer
    - Performs domain logic (business logic)
        - Classes for representing data (e.g. for presentation)
        - Methods for doing various calculations on entered and stored data, validation of data, find the correct retrieval of data from data provider(s)
- Repository layer
    - Connects to the dataprovider
    - Performs Create, Read, Update and Delete (CRUD) operations

What is the difference between layers and tiers?

Layers – logical separation of code



Tiers – Distributing layers on different servers

Does not mean that each layer need to be placed in a separate tier

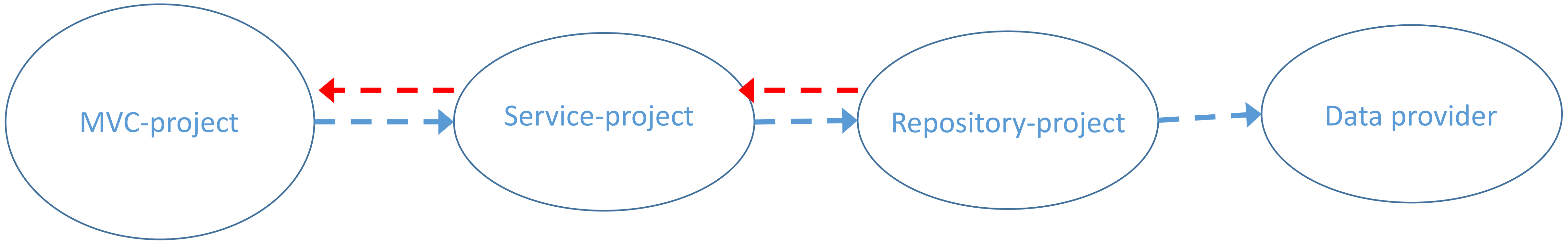
Implementing the architecture

- The layered structure can be implemented within different projects
    - This makes it easier to make the layers independent and to separate into different tiers if required
- The projects need to reference each other in such away as to avoid circular references
    - The presentation layer need to reference the Service layer
    - The Service layer need to reference the Repository (Data Access) Layer (DAL)
    - The Repository need to reference the Data layer

References between projects

When separating layers into different projects

It is necessary to set up the references between the projects



Must avoid circular references

Cannot place part of the domain model in the MVC project

Cannot place mapping of data, from entity classes to view model classes, in repository

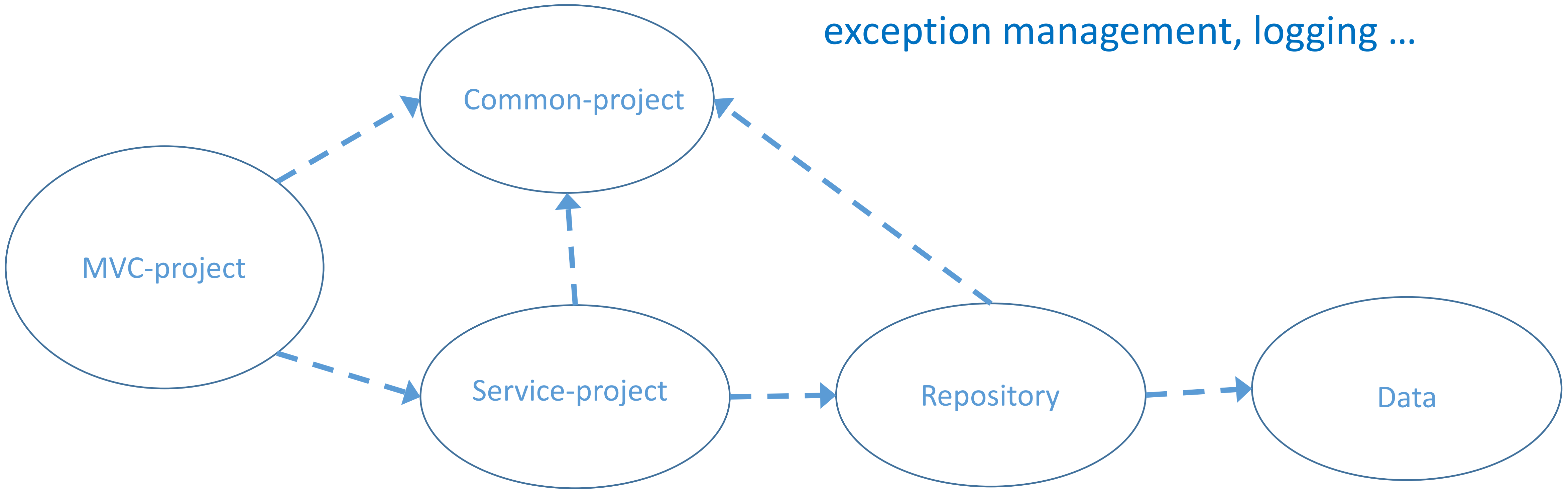
If we need circular references

- We can’t create circular references
- But we can create a common layer which can be referenced from all layers
    - Common features for all or several layers
        - Mapping of data - placed in the repository layer
        - Communication - Often used for Data Transfer Objects (DTO’s)
        - Authorization and authentication
        - Logging
        - Exception management
        - Etc.

Using a common layer

avoiding circular references

There might be certain features that need to be common between layers. Such as mapping of data, authentication, exception management, logging …



Adding a new project

- In VS select FILE -> New -> Project
- In window “New Project”
  - Select Visual C# and Class Library
  - Name the new project [Service, Repository or Common]
  - For “Solution:” select “Add to solution”
  - Click OK
- Add the required reference(s)
  - Right click “References” and select “Add a reference”
  - In “Solution” – “Projects” click the projects for which a reference is required
  - E.g. in the main project (with controllers and views) add a reference to “Service” [and to “Common”]

Example with the “Project Manager” application…

- Further develop the example in two phases
- Phase 1
  - Add the service layer as a separate project
  - Implement the mockup in the service layer
  - Implement appropriate methods in the Department and Employee classes
    - (Using the Domain Model design pattern)
- Phase 2
  - Add the repository layer as a separate project
  - Implement the Entity classes
  - Implement the connection to the database
  - Implement the mapping of Entity classes to View model classes

…Example cont. – Phase 1 (1)…

- Add Service project
  - Will not use the Common project at the moment
- Add a “Models” folder to the “Service” project
  - Create the classes, from the “ProjectManager” – Models folder, in the “Service”-Models folder
    - Note they have to have the namespace of the “Service”-project
- In “ProjectManager” project add reference to “Service” project
- In the controller classes set using statement to the Service.Models

…Example cont. – Phase 1 (2)…

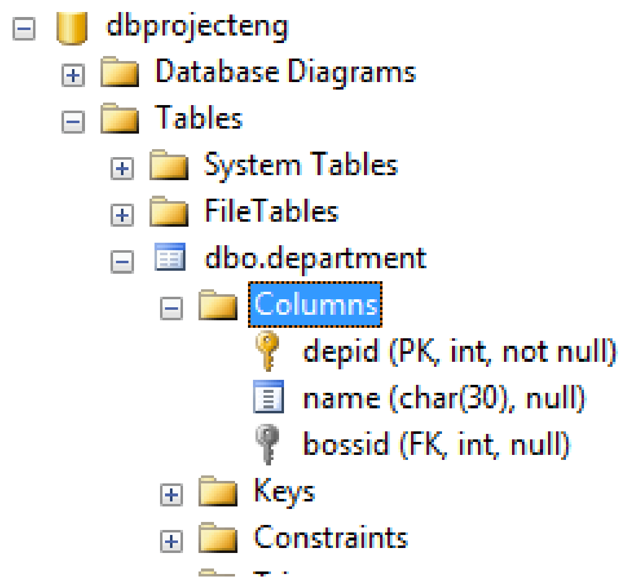
- Add a “Mockups” folder to the “Service” project
  - Instead of doing the mockup directly in the Controllers
- In the “Mockups” folder add
  - A class for “DepartmentMockup”
  - A class for “EmployeeMockup”
- Move the mockup code for Departments to the DepartmentMockup and the code for Employees to the EmployeeMockup
  - Make the objects public
  - Add using statement to the Service.Models

…Example cont. – Phase 1 (3)…

- Now, instead of the mockup code in the “ProjectManager” controls, we will create the lists for departments and employees by calling methods in the service classes for Department resp. Employee
- Add a static public method getDepartments to the Department class
  - Returns a List<Department>
- Add a static public method getEmployees to the Employee class
  - Returns a List<Employee>
- Make appropriate calls to these methods in the “ProjectManager” control class(es)

…Example cont. – Phase 2 (1)…

- Add a project for the Repository
- In “Service” project add reference to “Repository” project
- In Repository, add a folder “EntityModel” for the entity classes
- Add an entity class for each table in the database
  - The entity class matches exactly the table, using appropriate datatypes



```
dbprojecteng
    Database Diagrams
    Tables
        System Tables
        FileTables
        dbo.department
            Columns
                depid (PK, int, not null)
                name (char(30), null)
                bossid (FK, int, null)
            Keys
            Constraints
```

```
namespace Repository.EntityModel
{
    public class department
    {
        private int _depid;
        private string _name;
        private int _bossid;

        public int DepId { get { return _depid; } set { _depid = value; } }
        public string Name { get { return _name; } set { _name = value; } }
        public int BossId { get { return _bossid; } set { _bossid = value; } }
    }
}
```

…Example cont. – Phase 2 (2)…

- Assume there is a database to connect to
- Add a connection string to the database in the Web.config file

```
<connectionStrings>
    <add name="projectmanager“ connectionString="Server=[servername];
      Database=[databasename];user     Id=[user id];Password=[password];"
      providerName="System.Data.SqlClient"/>
</connectionStrings>
```

- Formatting of connectionString property (for the data provider to use) see https://www.connectionstrings.com/
- The name property provides the name by which we can recall the connection-string we want to use
  - It is possible to have several
- The providerName property is the using statement for adding the namespace with the appropriate provider class library

…Example cont. – Phase 2 (3)…

- In Repository project, add a folder “Repositories”
- Add classes that will provide methods for performing queries
  - SQL queries in case of a relational database
  - There will be such a repository class for each table
- A SQL-query need typically to:
  - [1] Fetch the connection string
  - [2] Create a connection object
  - [3] Attach the query to the connection object
  - [4] Open the connection
  - [5] Run the query
  - [6] Read the data from the obtained dataset
  - Close the connection object

See the numbered rows in the example on next slide

…Example cont. – Phase 2 (4)…

```
    static public department dbGetDepartment(int id)
    {
       department _depObj = null;
[1]    string _connectionString = DataSource.getConnectionString("projectmanager");
[2]    SqlConnection con = new SqlConnection(_connectionString);
[3]    SqlCommand cmd = new SqlCommand("SELECT * FROM department WHERE depid = " +
            Convert.ToString(id) + ";", con);
       try
       {
[4]        con.Open();
[5]        SqlDataReader dar = cmd.ExecuteReader();
           if(dar.Read())
           {
               _depObj = new department();
[6]            _depObj.DepId = Convert.ToInt32(dar["depid"]);
               _depObj.Name = dar["name"] as string;
               _depObj.BossId = Convert.ToInt32(dar["bossid"]);
           }
       }
        ...
        return _depObj;
    }
```

…Example cont. – Phase 2 (5)

- Fetching the connection string can be done using a built in property

```
System.Web.Configuration.WebConfigurationManager.
ConnectionStrings[name].ConnectionString;
```

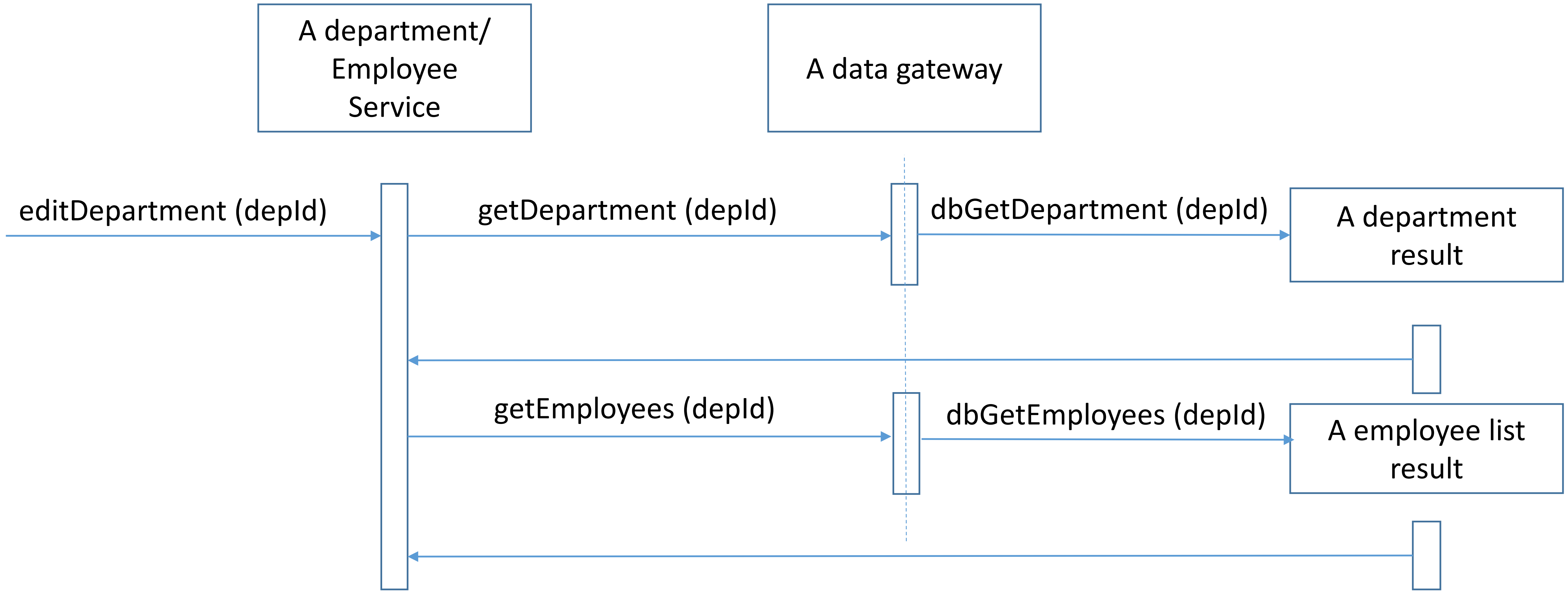
- This is what is done by calling the getConnectionString method and passing the name for the connection string that was set up in Web.config
- This make it possible to rather easily handle different data providers requiring different connection strings
- Thereby you also need not expose username and password directly in the written code

Design patterns for organizing domain logic

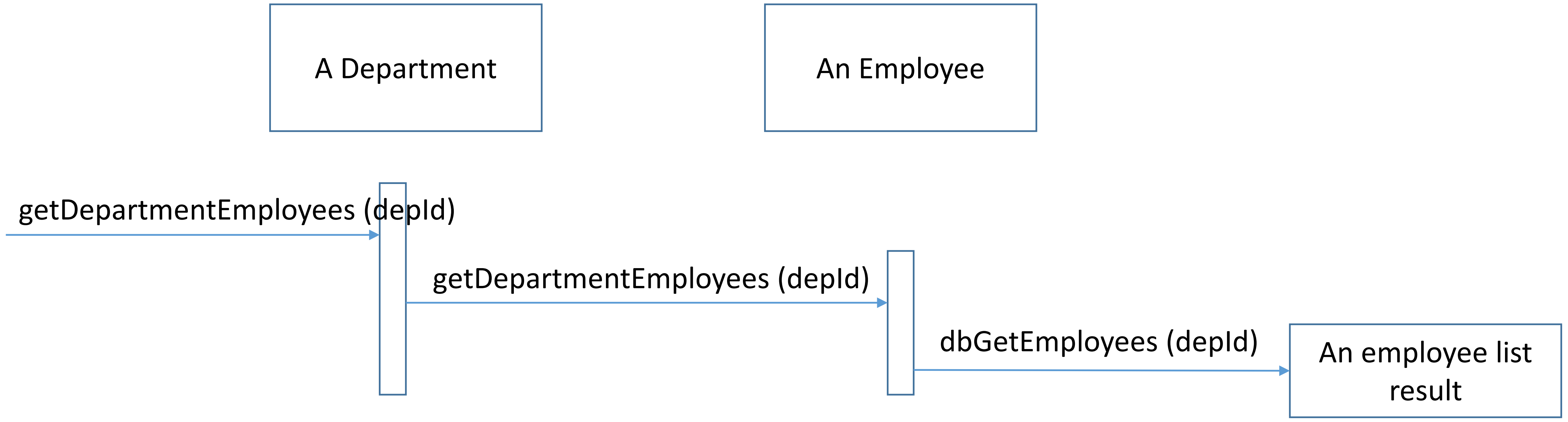
- Two out of several existing patterns:
    - *Transaction script pattern*
        - One procedure for each action
        - Collecting data using a data gateway
        - Summarizing to an answer
        - Easy to use
        - Leads to duplication of code
    - *Domain model pattern*
        - A more object-oriented approach.
        - The action is forwarded between objects
        - Each object takes responsibility for their part
        - Handles complex logic in a well organized way
- Ref. Fowler M. *Patterns of Enterprise Application Architecture*

Example of Transaction Script Pattern

Retrieving the list of employees working at a department



Example of Domain Model pattern



Reading

- Pro ASP.NET MVC 5
  - Chapter X – yyy
- Links:
  - http://stackoverflow.com/questions/120438/whats-the-difference-between-layers-and-tiers
  - https://msdn.microsoft.com/en-us/library/ff650706.aspx
    - Especially the chapter about Design fundamentals
  -