Course review

- Poor software design and engineering are the root causes of most security vulnerabilities in deployed systems today.

- This course takes a close look at software as a mechanism for an attack, a tool for protecting resources, and as a resource to be defended

Lecture outline

A. Architectural design patterns and product lines.
B. Design patterns.
C. Service-oriented architectures.

A. Architectural design patterns and product lines

- **Architectural patterns** (also known as **architectural styles**) codify recurrent software architectures, by describing the key elements of the architecture and how they fit together.

Software architectures

- A **software architecture** refers to the broad structure of a software system, it describes its major parts, and how they are put together and interact.

Software architectures

- **Some of the architectural views are:**

1. The functional (or logical view), it describes:

- the system's main basic services/functional elements,
- their responsibilities,
- interfaces,
- and primary interactions.

2. The process view, addresses concurrent aspects of the system at run-time, all system processes, start-up and shut-down.

3. The deployment view, maps the software elements onto the run-time, it describes the environment into which the system will be deployed, including processing nodes, network interconnections and disk storage facilities required;

Summary of Architecture & Design Pattern Types

Architecture Patterns	Design Patterns
Patterns for the skeleton and abstract view of the software.	Patterns for solving specific code scenarios
Layers Architecture	
Model View Controller (MVC)	Observer Behavioral pattern
Components Architecture	Singletons **Creational** pattern
Service Oriented Architecture	Factory **Creational** pattern

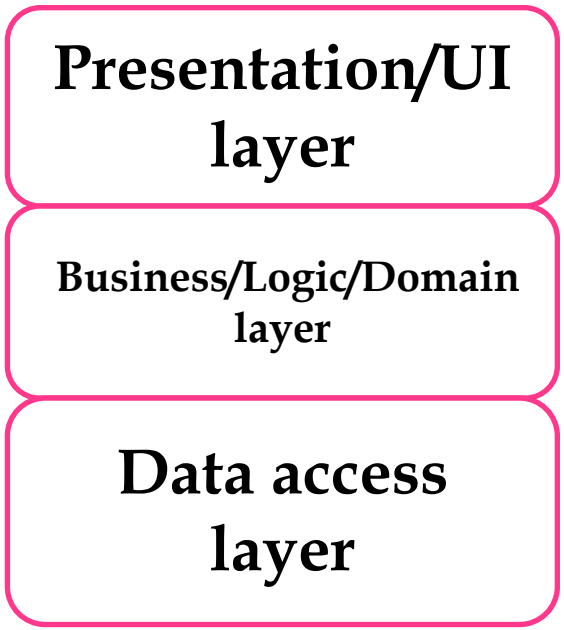
Design Patterns

- **These design patterns divided mainly into three Groups:**

1. Creational Patterns
2. Structural Patterns
3. Behavioral Patterns

The Layers pattern

- **The Layers pattern**, help to structure the architecture of a system into groups of basic services/functionalities, each at a particular level of abstraction.
- How it works: The system is structured into an appropriate number of layers, with each layer making use only of services provided by the layers below it.



The Layers pattern cont.

- When to use it: Applies in the development of large complex systems, which require decomposition.
- Example: Networking protocols (e.g. the ISO OSI seven-layer model) that describe how computers communicate across machine boundaries and over networks. ranging from the physical layer (dealing with transferring electrical signals on a physical network) up to the application layer (dealing with transferring data between remote applications).

The Layers pattern

Advantages:

- separation of concerns: each layer addresses a particular group of related services/functionalities, so that all the groups of functionalities are kept separate.
- lowered coupling: the dependency between layers is kept low by following clearly stated rules, typically from high- to low-level layers;
- reusability: general services (for instance access to remote devices over a network) are kept separate from specific application logic and hence are potentially reusable.
- Maintainability & flexibility: because of low coupling and separation of concerns, changes in elements of the system are localized and hence easier to achieve; entire layers may be replaced with alternative implementations of the same basic services.

Disadvantages :

- replication: there may be a need to replicate information across layers;
- performance: there may be a need to perform extra transformations at each layer.

The Layers pattern

SAQ 1

(a) What are the main issues addressed by the Layers pattern and what are the main mechanisms used to address them?

(b) How does the adoption of the pattern facilitate reuse and flexibility?

ANSWER.....

(a) The main issues addressed by the pattern are achieving low coupling between functionalities and separating concerns. This is achieved by organizing the system elements in layers/tiers/folders, each layer addressing a particular concern.

(b) The implementation of general services can be separated from that of other parts of the system. Hence services can be potentially reused in other applications, or their implementation replaced while still providing the same basic services.

The Layers pattern

- From a design viewpoint, the main difficulty in applying the Layers pattern is to decide:
  - how the system should be separated into layers and
  - how to assign **responsibility** to each layer.
- **The basic Layers pattern includes three principal layers:**
  1. The presentation layer, handles the interaction between the users and the system; it captures user events and the system's user interface.
  2. The domain (or business) layer, defines the work of the system; it contains domain logic and implements domain rules.
  3. The data source layer, handles the storage of persistent data, and networking with other systems.
- **There are two extra mediating layers between the principal layers:**
  4. The application layer, mediates between the presentation and domain layers; it may maintain session states, or perform transformations to consolidate disparate data for presentation.
  5. The technical services layer, mediates between the domain and data source layers. It may contain services for persistence, transactions, load balancing, or security; or interfaces to external systems.

The Layers pattern

- **There are two styles (flavours) of Layers pattern:**
  - **Three-layer pattern:** Principle layers (1, 2, 3)
  - **Five-layer pattern:** involves layers 1-5
- We can represent a **layered architecture using the UML notation for a package**.
- **A UML package** is defined to represent each layer in the two flavours of Layers pattern as shown in **figure 1**.

- Figure 2 shows a layered view of the design for the hotel system using a UML package. The HotelChainBusiness package is the domain layer of the system and contains the whole class diagram for the hotel system.

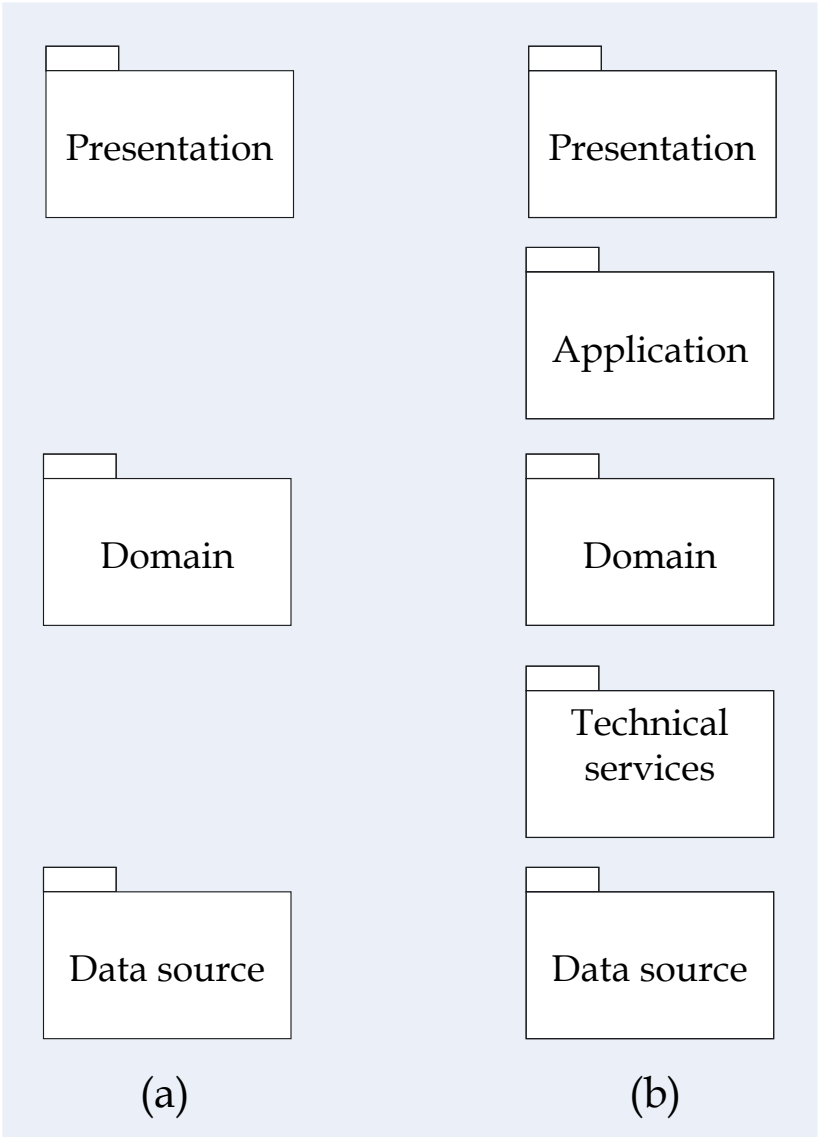


Figure 1 Two flavours of the Layers pattern, represented as UML packages

The Layers pattern

SAQ 2

(a) Describe the relationship between the two styles of Layers pattern illustrated in Figure 1.

(b) What do you think may determine the flavour of the patterns that should be adopted for the architecture of a particular enterprise system?

ANSWER.....

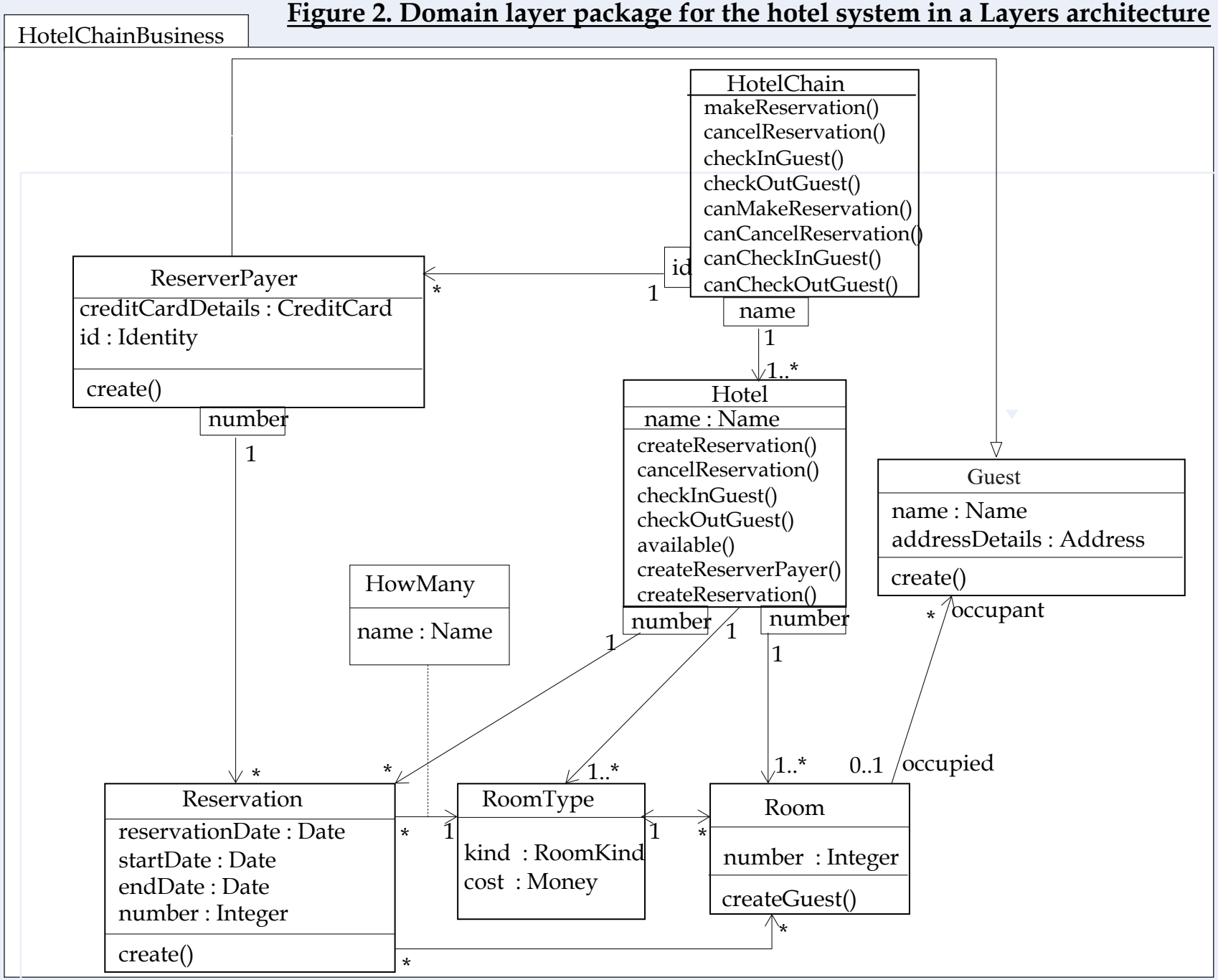
(a) The five-layered version (Figure 1(b)) introduces two mediating layers between the fundamental layers of the three-layered version (Figure 1(a)).

(b) Complexity and flexibility are determining factors. For instance, if the relationship between the presentation and domain layers is straight forward and can be easily modified and tested, then it might be more efficient to distribute the logic between these two layers rather than introducing a third one. On the other hand, in a system that allows complex interactions to occur with a large variety of users and where changes are frequent, the introduction of an application layer might allow the logic in the two other layers to be independently modified, tested and validated. A similar argument applies to the technical services layer.

The Layers pattern

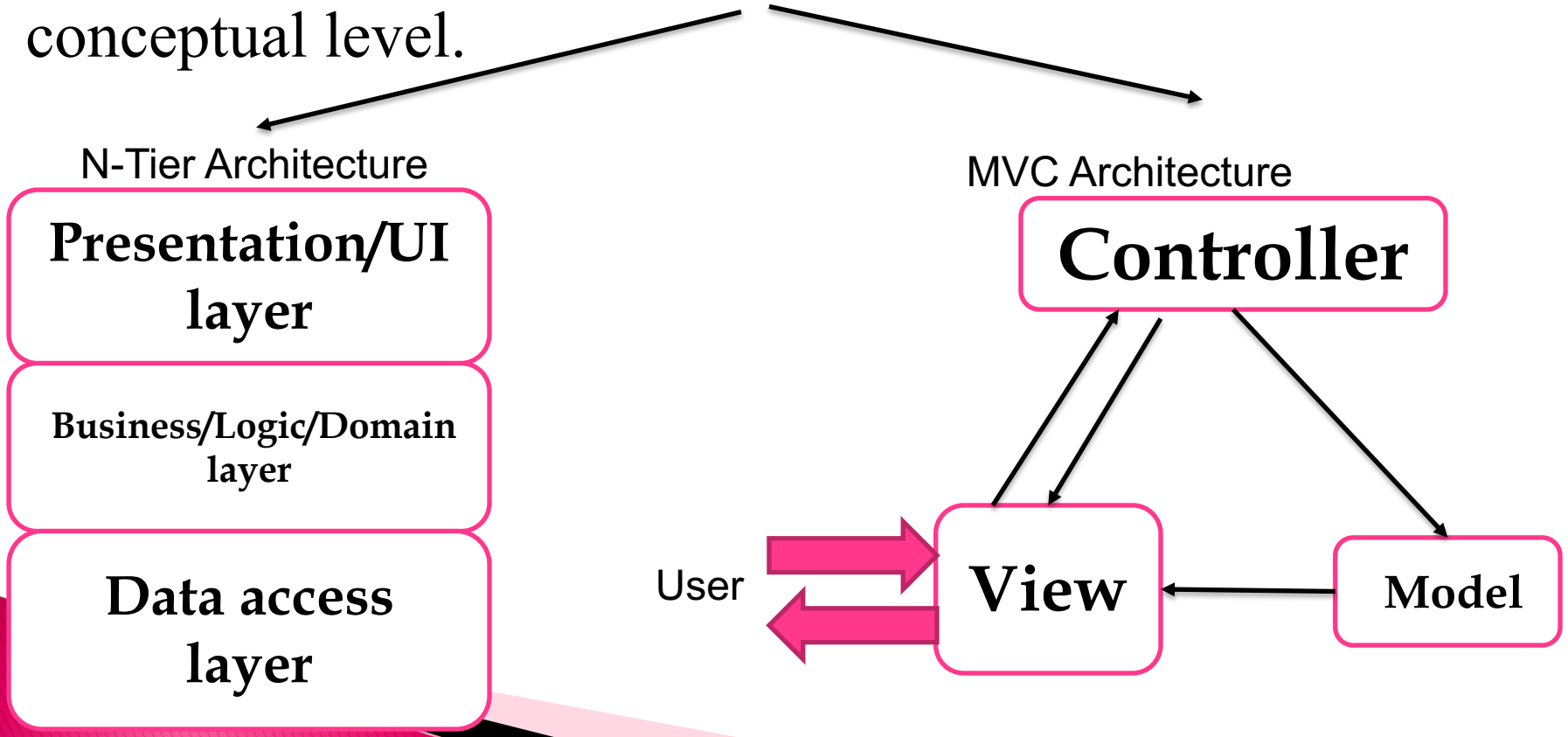
- Figure 2 shows a layered view of the design for the hotel system using a UML package.
- The HotelChainBusiness package is the domain layer of the system and contains the whole class diagram for the hotel system.

Figure 2. Domain layer package for the hotel system in a Layers architecture



Architecture Patterns (Cont.)

3. Multi-Tier/Layer: Objects must be grouped into tiers/layers (folders), where each tier is responsible for a common functionality of the software at a very high conceptual level.



1. Model–View–Controller (MVC)

- **Intent** To split user interface interaction into three distinct roles.
- **How it works** It identifies three roles:

a) **The model**, corresponding to an object with some information about the domain. The model contains data and behaviour and is not a visual object. In a layered architecture, the model resides in the business layer.

b) **The view** is the graphical manifestation of the model in the user interface: it is what the user perceives of the model. Views reside in the presentation layer of a layered application.

c) **Controllers** handles all user inputs which affect the model, and are also presentation-layer objects.

1. Model–View–Controller (MVC)

- The pattern works in such a way that user inputs to the controller translate into changes to the model's state, which in turn are reflected in the view.
- the view and the controller are tightly coupled to a point that they are often integrated into the same frameworks, for example, in the user interface (UI) components of Java's Swing library. (OLD…..)
- In web apps the view is the front end and usually uses Angular, VUE, or react frameworks.

1. Model–View–Controller (MVC) (Cont.)

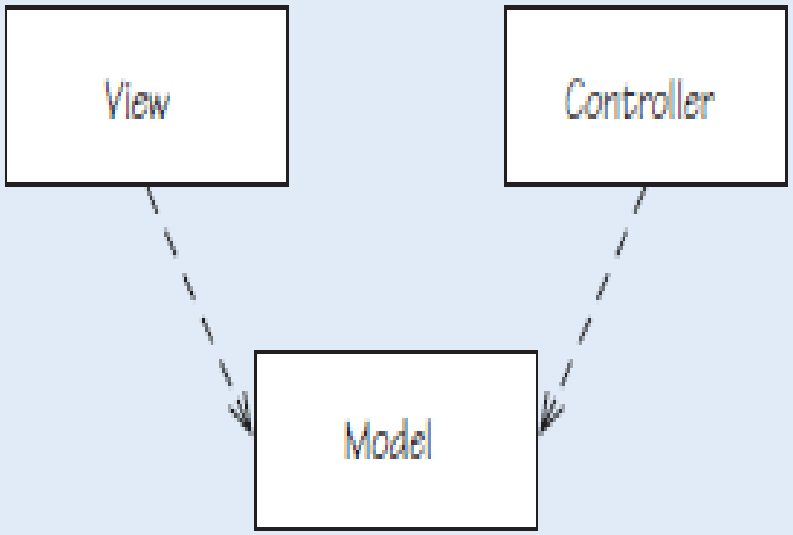


Figure 5. Dependencies in Model-View-Controller

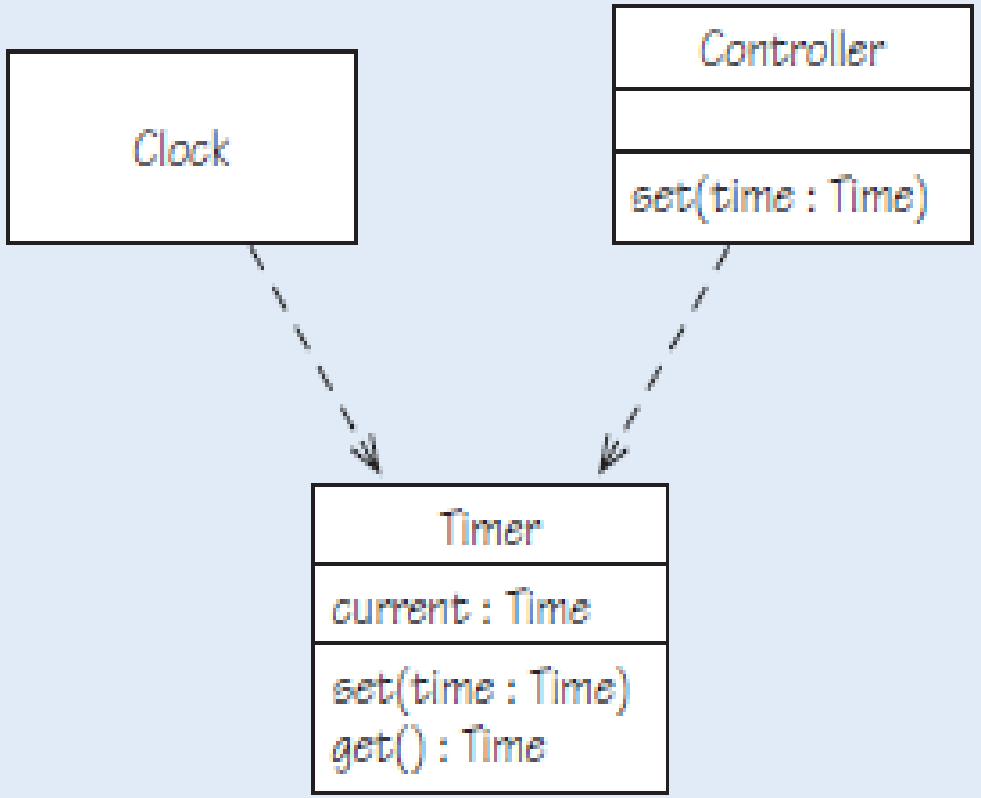


Figure 6. MVC structure for a clock system

1. Model–View–Controller (MVC) (Cont.)

When to use it Use this pattern to separate the presentation from the model. **The advantages** of this separation are many, including:

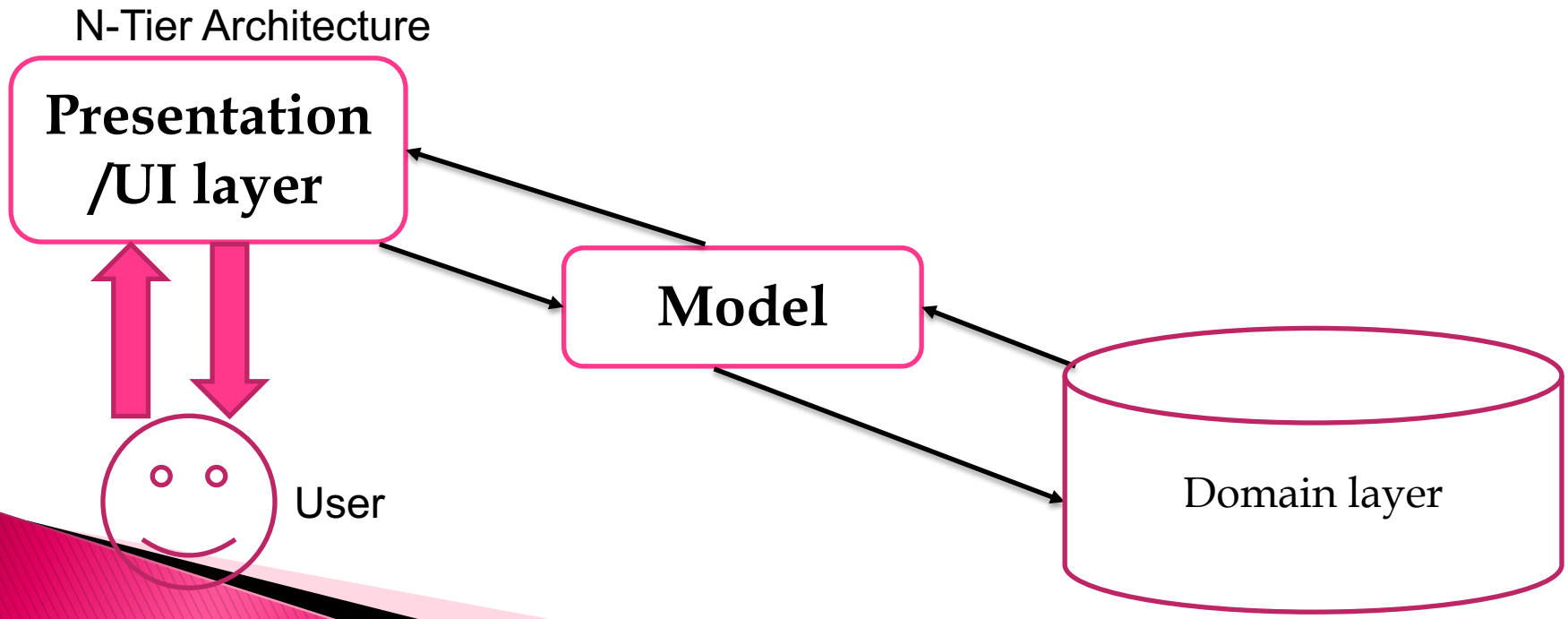
a. **separation of concerns**
b. **ease of testing**
c. **Maintainability**

- **The view and the controller are both dependent on the model**, as shown in Figure 5.
- **Example** Figure 6 shows an example of a Clock (the view) showing the current time of Timer (the model), with a Controller which allows the user to set the current time to an arbitrary value.

New 8-10-2022

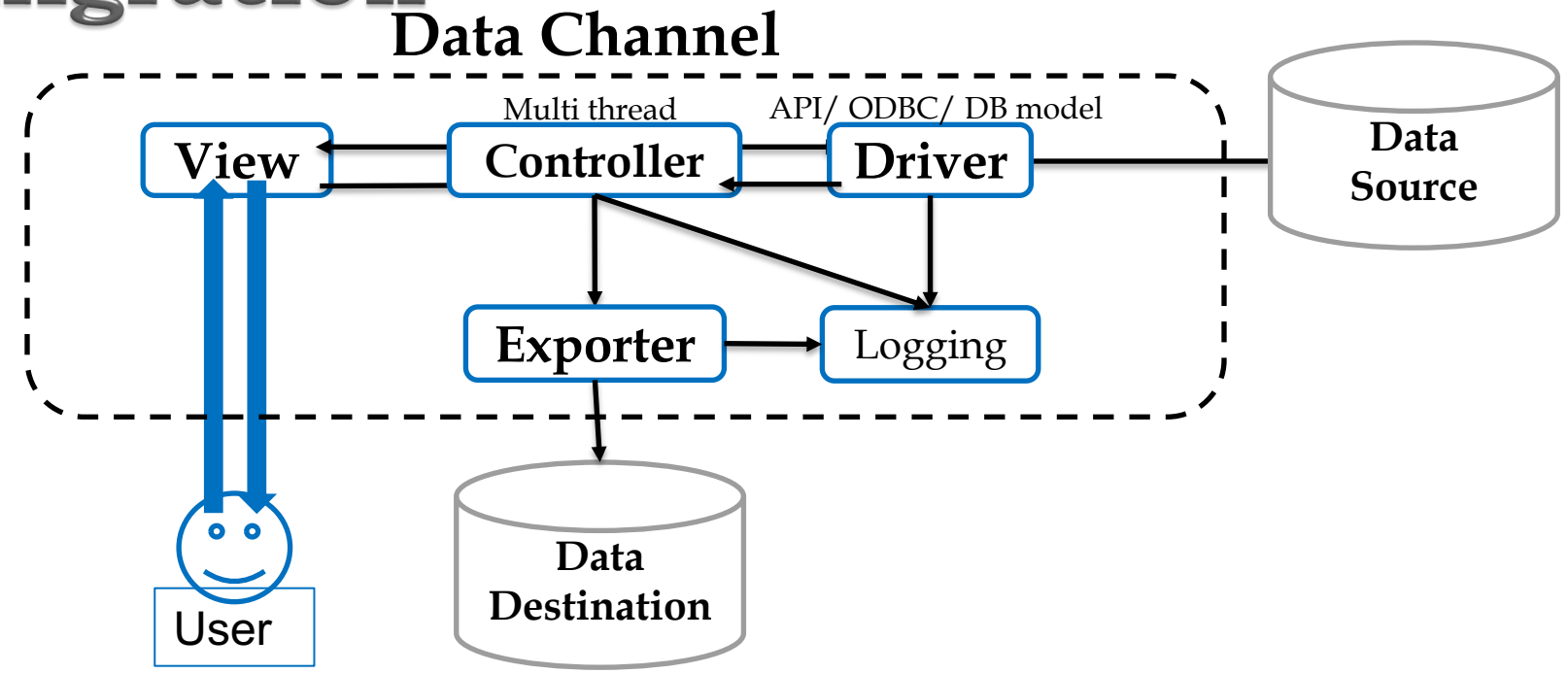
2. Customized architecture

Logic/Domain layer: is a collection of dynamic stored procedures that can rebuild their behavior at run time based on parameters passed to them. Each business entity has one stored procedure that gathers all its operations.



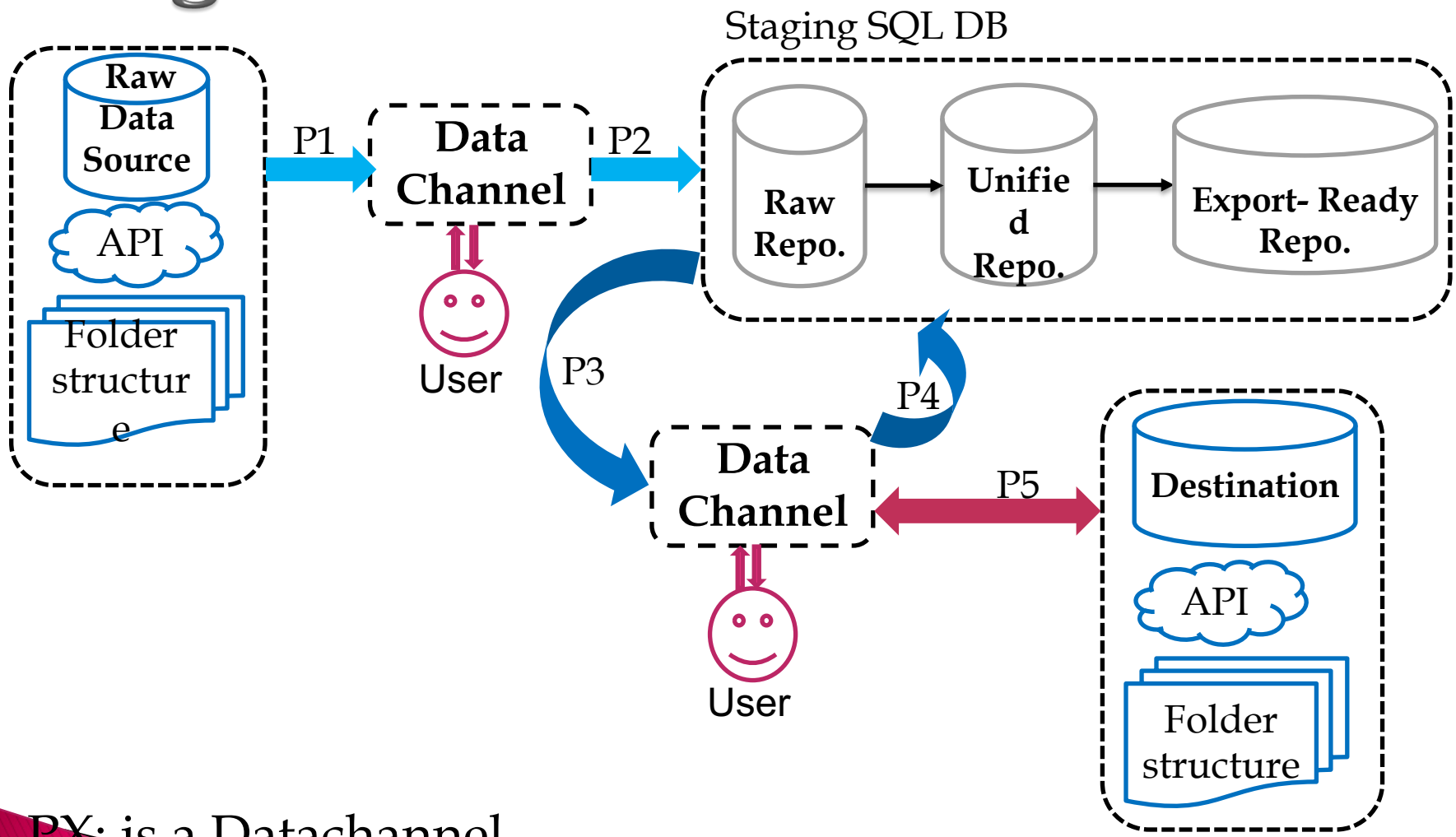
modularized architecture for Data migration

5-Nov-2022



modularized architecture for Data migration

5-Nov-2022



PX: is a Datachannel profile

Service-oriented architectures.

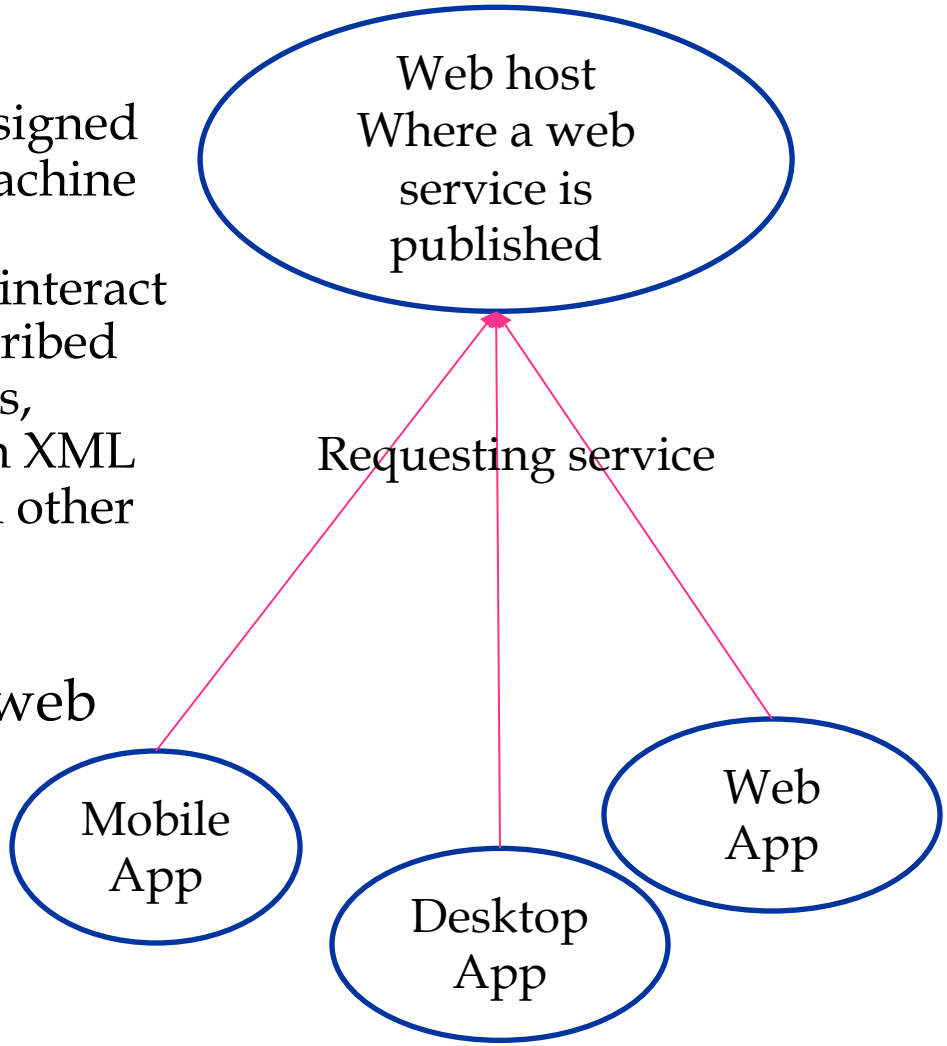
- **Service-oriented architecture (SOA)** is an approach to build enterprise systems that deliver application functionality either as services to end-user applications or to build other services.
- **A service** is an application function which is packaged as a reusable component for use in a business process.
- **One important characteristic of services within SOA** is that they are **discoverable**, that is to say the consumer does not need to know about the service or where is located.

Web Service

Definition

A Web service is a software module designed to support interoperable machine-to-machine interaction over a network. It has a programming interface. Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML or json serialization in conjunction with other Web-related standards [https://www.w3.org/TR/ws-gloss/]

Web API is the another form of a web application.



Service-oriented architectures.

The characteristics of a service in SOA:

1. A service is a self-contained and modular software entity.
2. It interacts with applications and other services through a loosely coupled, message-based communication model.
3. It supports interoperability through reliance on communication standards.
4. It can be found by its consumers through a registry; hence its location is transparent to them.

Enterprise components and architectures (Cont.)

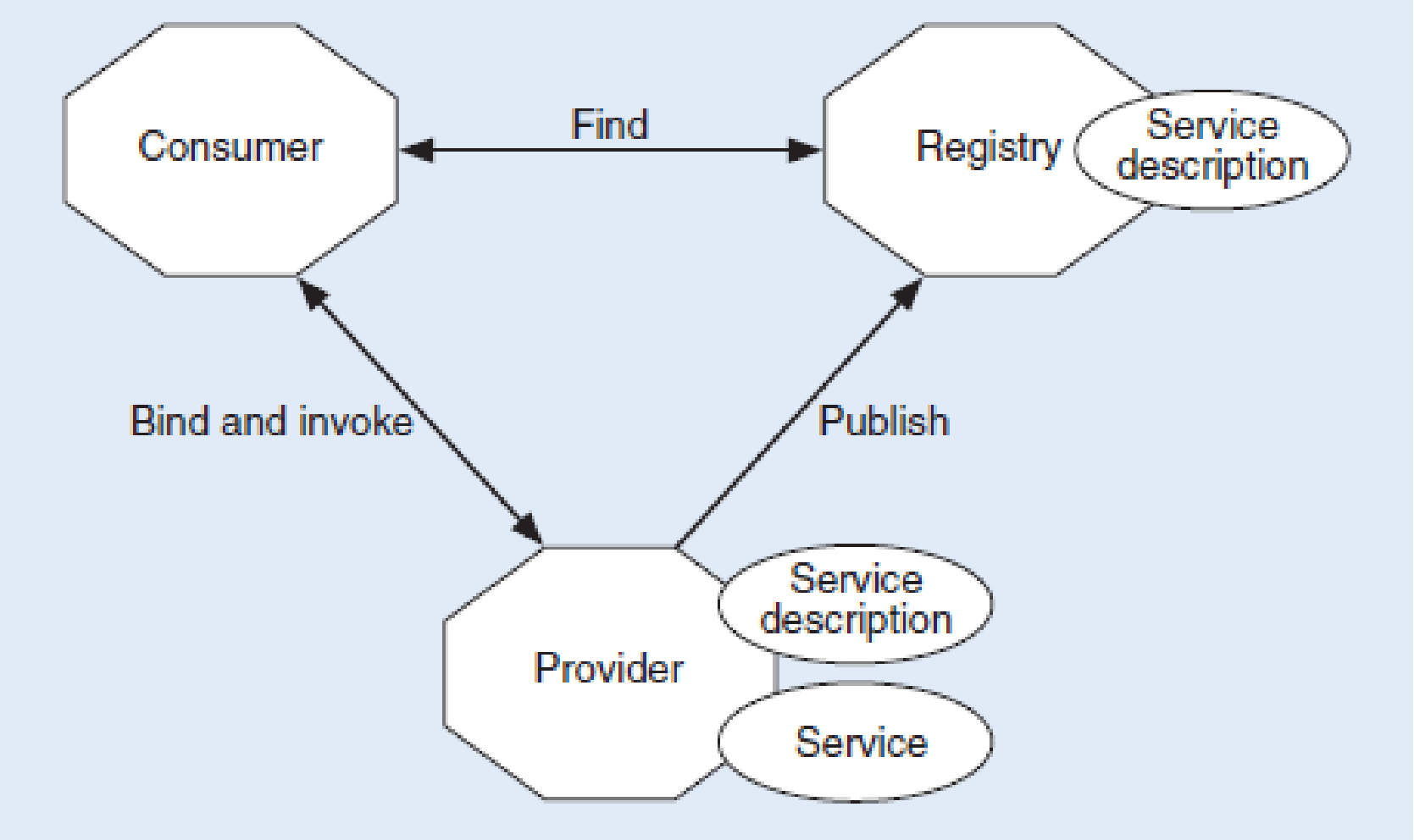


Figure 12. Service collaboration in SOA: the 'find, bind and invoke' cycle