HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

SOICT

School of Information and Communication Technology

ONE LOVE. ONE FUTURE.

TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

IT3180 – Introduction to Software Engineering

12 – System Architecture

ONE LOVE. ONE FUTURE.

Design Phase

For a given set of requirements, the software development team must design a system that will meet those requirements

The design of a system has many aspects:

- System architecture
- Program design
- Security
- Performance

- In practice, requirements and design are interrelated, working on the design often **clarifies** the requirements.

Creativity and Design

Creativity

- System and program design are a particular part of creativity in the software development, as user interfaces

Software development as a craft

- Software developers have a variety of tools that can be used in design
- We have to select the appropriate tool for a given implementation

System architecture

System architecture is the overall design of a system:

- **Computers and networks** (e.g., LAN, Internet, cloud services)
- **Interfaces and protocols** (e.g., HTTP, IMAP, ODBC)
- **Databases** (e.g., relational, distributed)
- **Security**
- **Operations** (e.g., backup, archiving, audit trails)

At this stage of the development process, we should select:

- Software environment (e.g., language, database system, framework)
- Testing framework

Models for System Architecture

Our models for system architecture are based on UML

- For every system, there is a choice of models
- The goal is to choose models that best model the system, which are also clear to everybody

In UML, every model must have both a diagram and a supporting specification

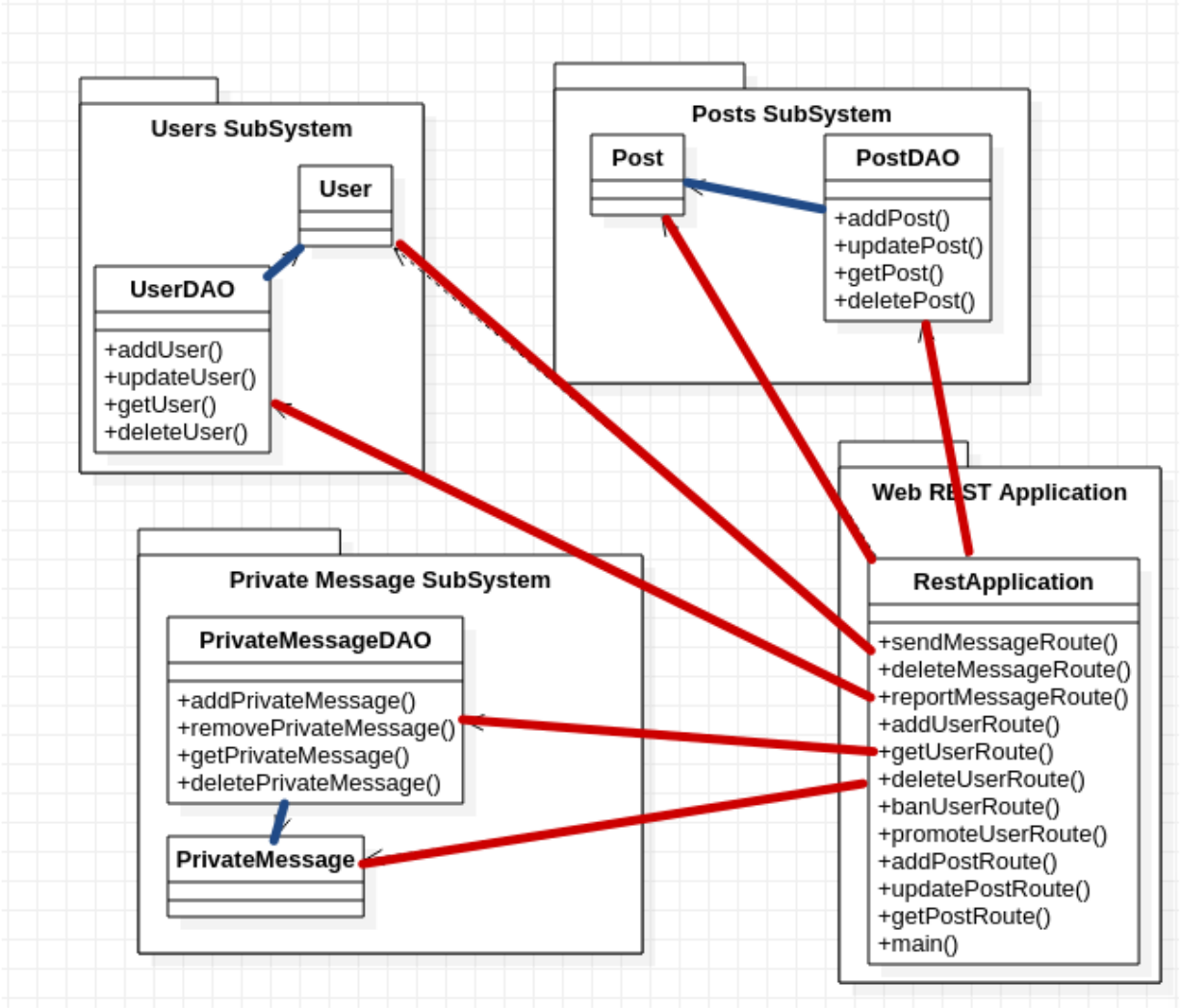
- The lectures provide diagrams that give an outline of the system, without supporting specifications
- The diagrams show the relationship among parts of the system, but much more details are needed to specify the system

Subsystems

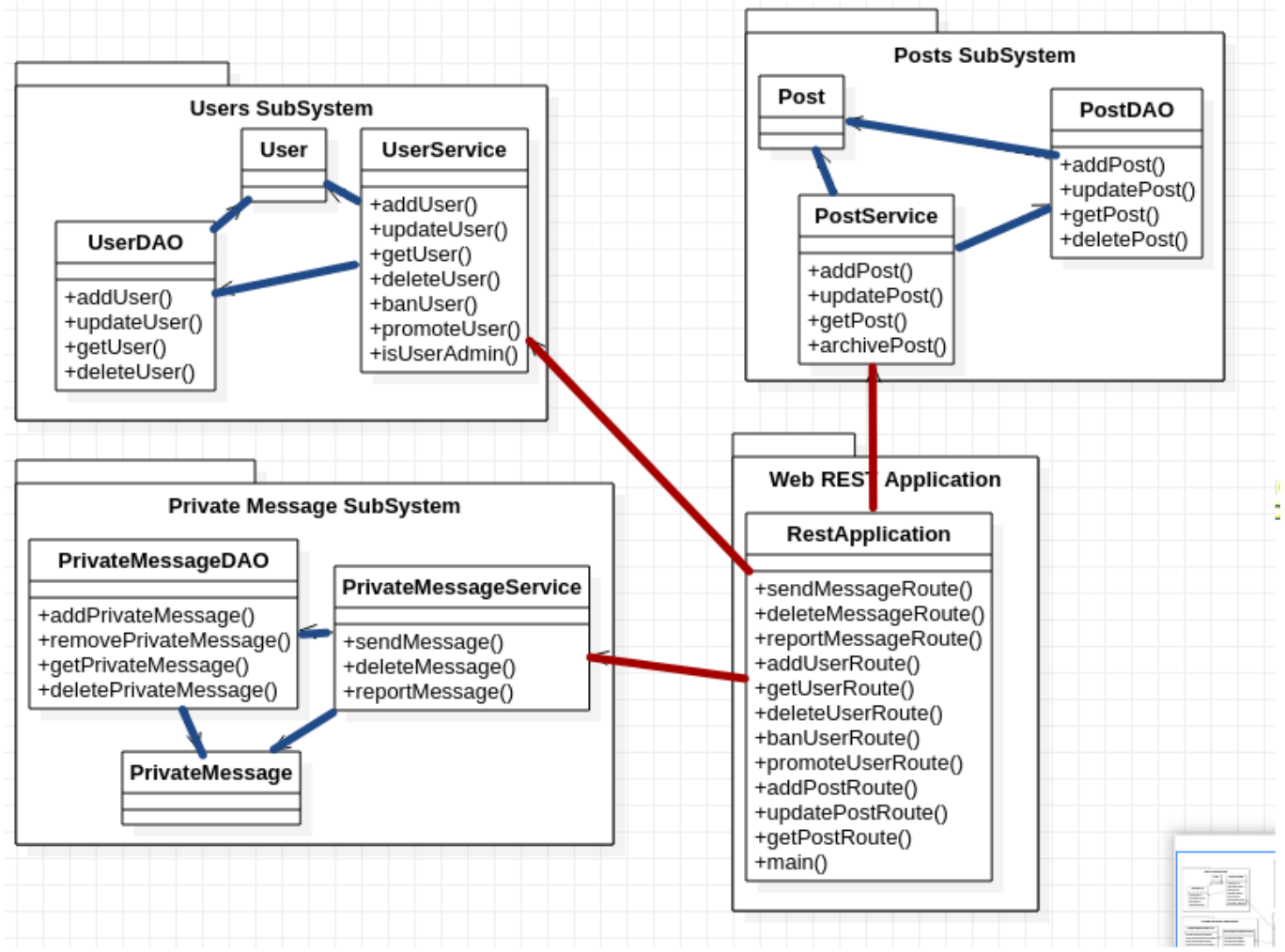
Subsystem is a group of elements that form part of a system

- Coupling is a measure of the dependencies **between** two subsystems
  - If two systems are strongly coupled, it is hard to modify one without modifying the other
- Cohesion is a measure of dependencies **within** a subsystem
  - If a subsystem contains many closely related functions, its cohesion is high
- An **ideal division** of a complex system into subsystems has **low coupling** between subsystems and **high cohesion** within subsystems

Example: Low cohesion (blue), High coupling (red)

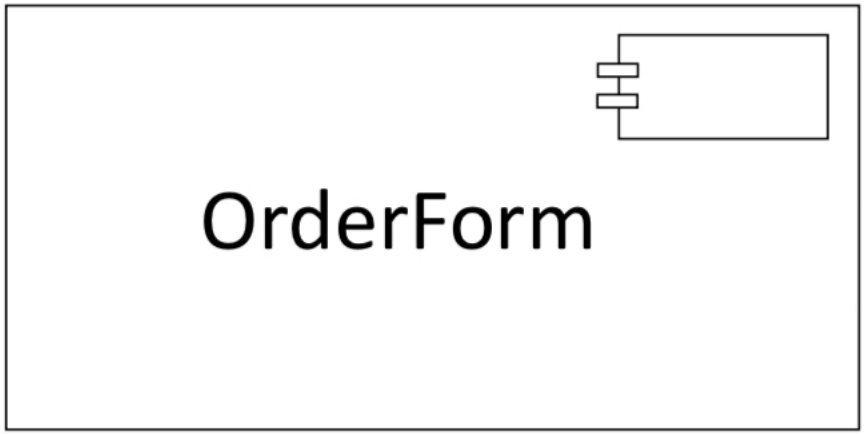


Example: High cohesion (blue), Low coupling (red)



BETTER!

Component

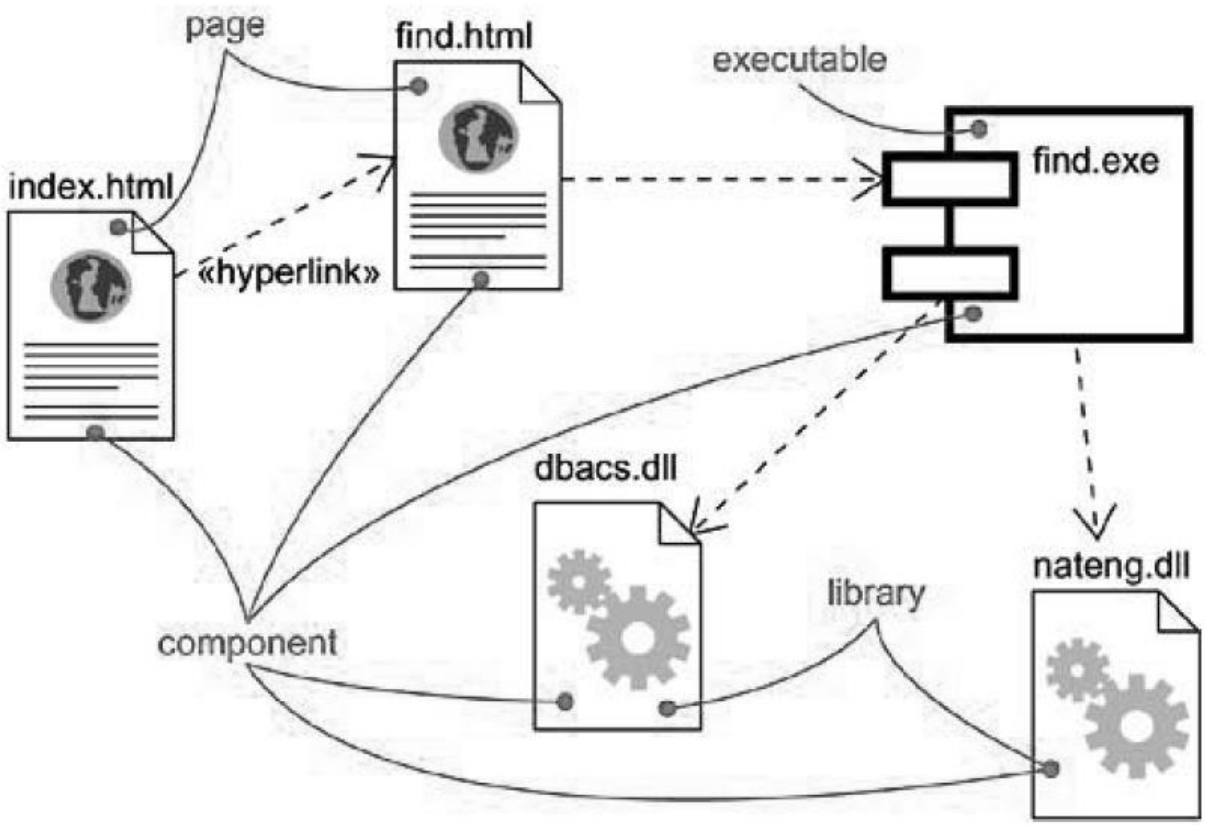


- A **component** is a replaceable part of a system that conforms to and provides the realization of a set of interfaces
- A **component** can be thought of as an implementation of a **subsystem**
- **UML definition** of a component
  - A distributable piece of implementation of a system, including software code (source, binary, executable) but also including business documents

Components as Replaceable Elements

Components allow systems to be assembled from **binary replaceable elements**

- A component can be **replaced** by any other component(s) that conforms to the **interfaces**
- A component is **part of a system**
- A component provides the **realization** of a set of **interfaces**



Components and Classes

- Classes represent **logical abstractions**
  - They have **attributes** (data) and **operations** (methods)
  - Classes can be combined to form programs

- Components represent **physical** things that live in the world of **bits**
- Components may be at different levels of abstraction
- Components have **operations** that are reachable only through **interfaces**
- Components can be combined to form systems

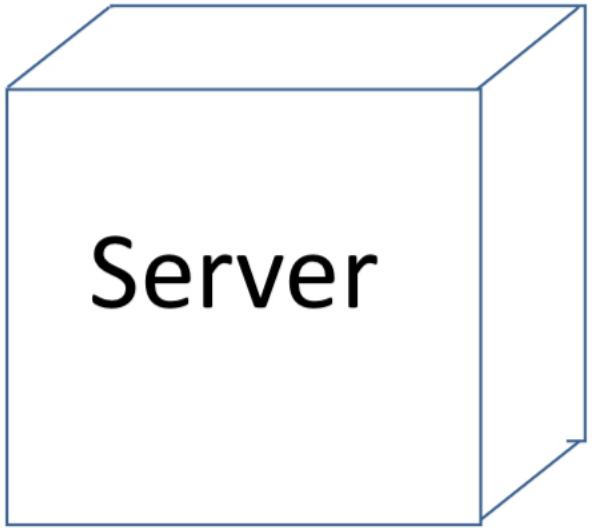
Kinds of Components

- Three kinds of components may be distinguished
- Deployment components
  - Components are necessary and sufficient to form an executable system
  - Dynamic libraries (.dll) or executable files (.exe) are two examples of deployment components
- Work product components
  - These components are generally the residue of the development process
  - Source code files, data files are examples of work product components from which deployment components are created
- Execution components
  - These components are created as a consequence of an executing system, such as .obj, .class files

Package

A **package** is a general purpose mechanism for organizing elements into groups

Node



- A **node** is a **physical element** that exists at a run time and provides a computational resource, e.g., a computer, a smartphone, a router etc.
- **Components** may live on **nodes**

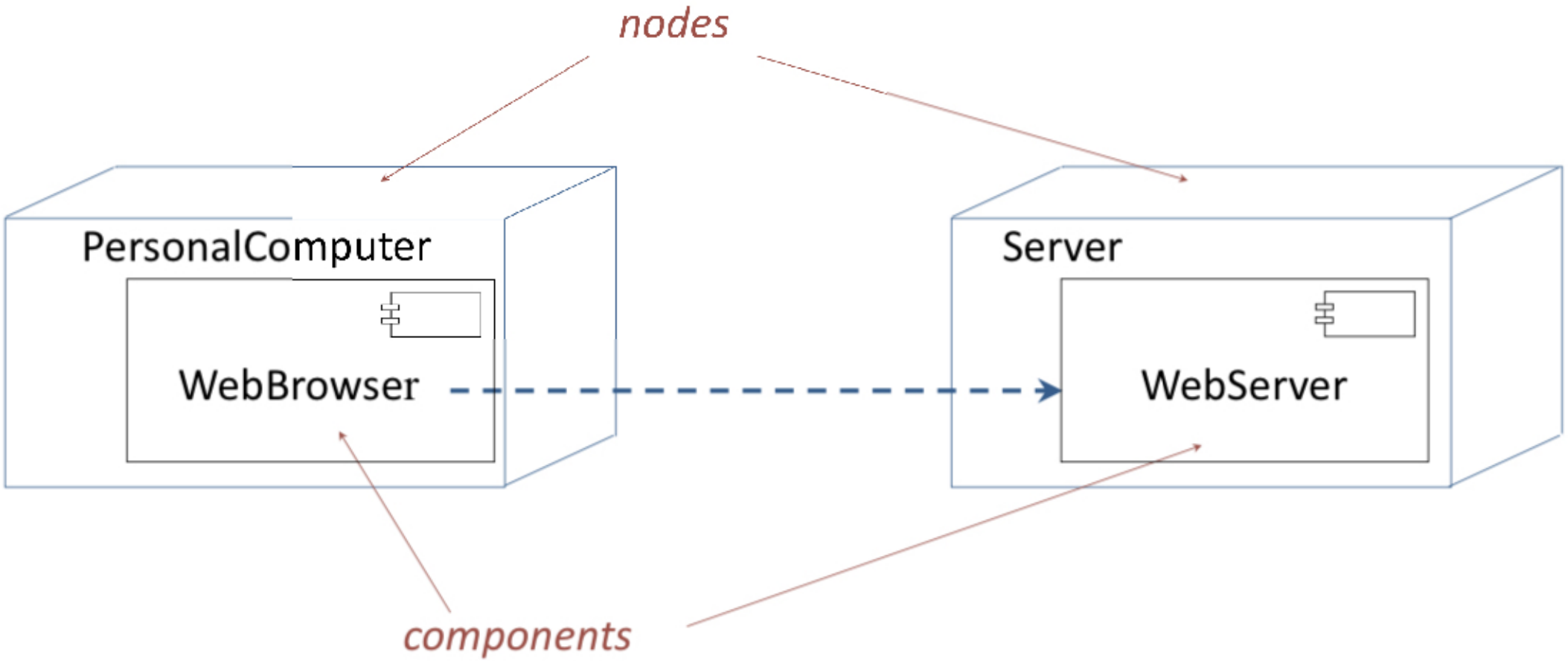
Example: Simple Web System

- Static pages from server
- All interactions require communication with the server

Deployment Diagram

- One of the two kinds of diagrams used in modeling the physical aspects of an object-oriented system
- Show the configuration of run time **processing nodes** and the **components** that live on them
- We use deployment diagrams to model the static deployment view of a system
- A deployment diagram commonly contains
  - Nodes
  - Dependency and association relationships

Deployment Diagram (2)



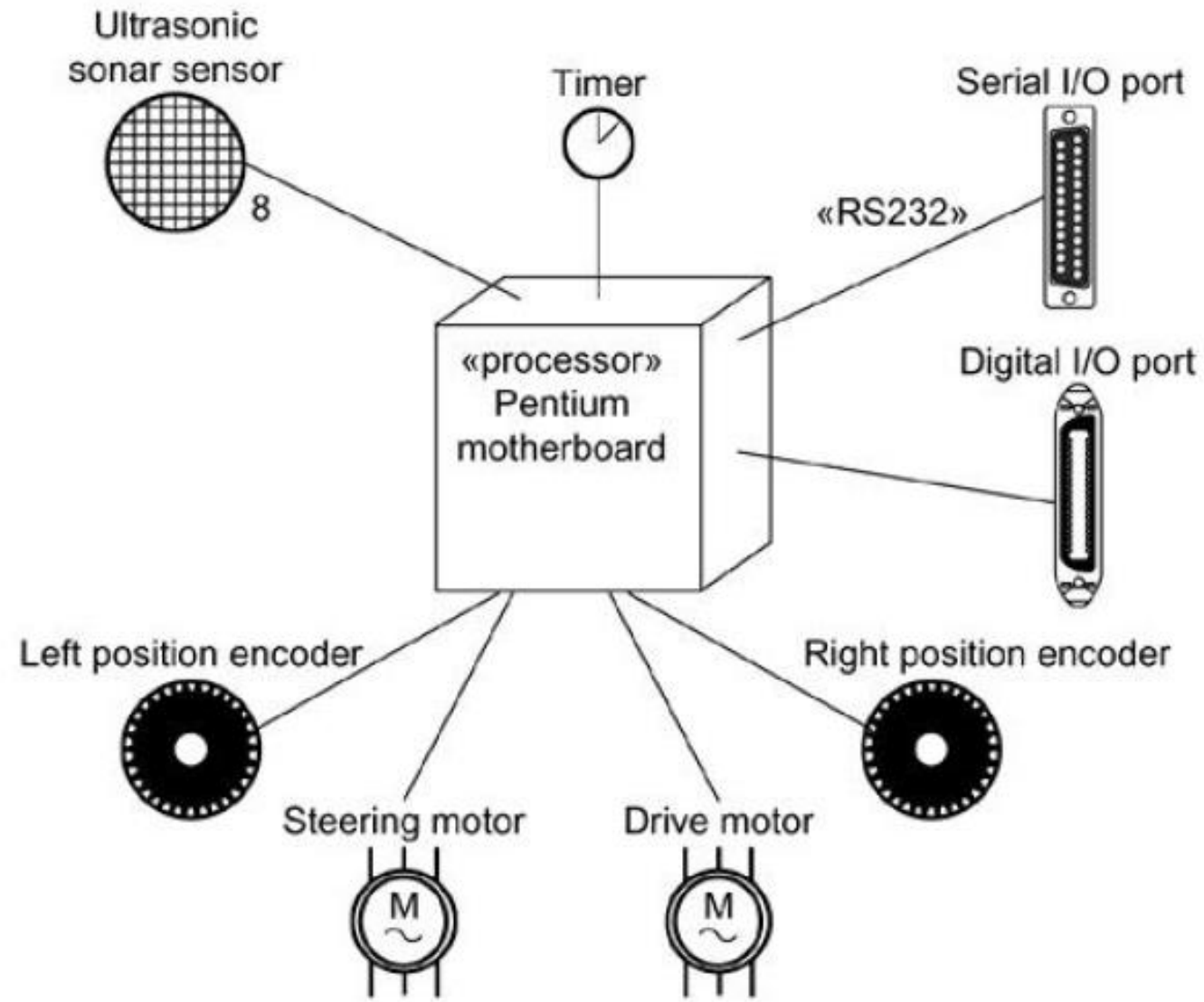
Deployment Diagram (3) – Common Uses

When modeling the **static deployment view** of a system, we'll typically use deployment diagrams in one of the three ways

- To model **embedded** systems
- To model **client/server** systems
- To model fully **distributed** systems

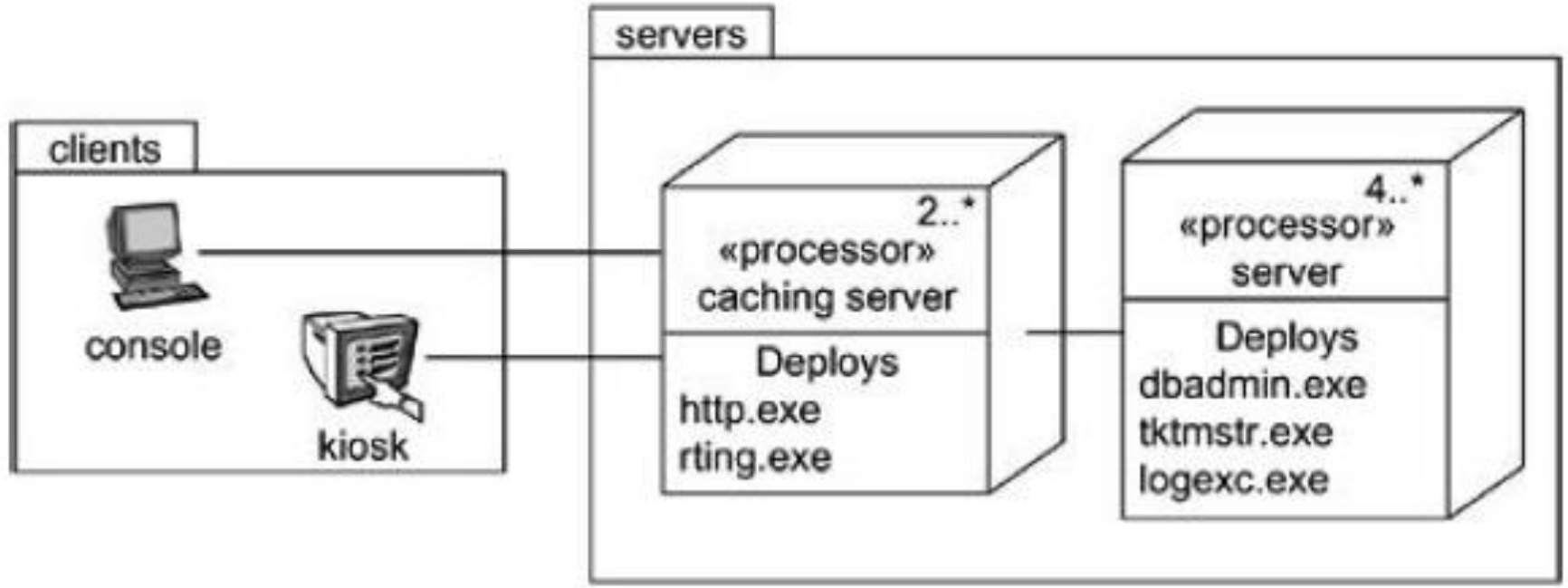
Model embedded system

- Identify the devices and nodes that are unique to your system
- Model the relationships among these processors and devices

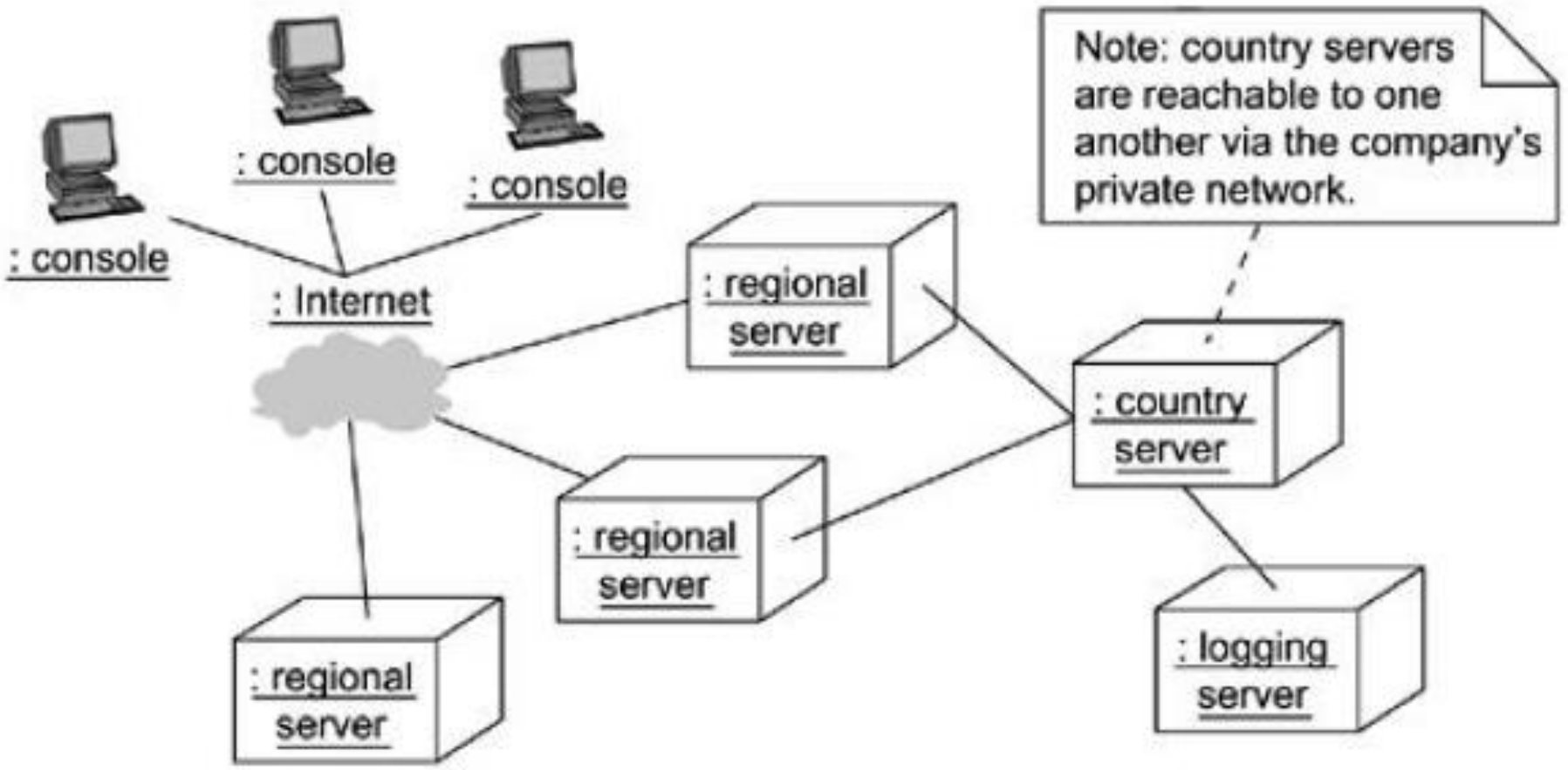


Model client/server system

- Identify the nodes that represent your system's client and server processors
- Highlight those devices that are relevant to the behavior of the system
- Model the topology of these nodes in a deployment diagram

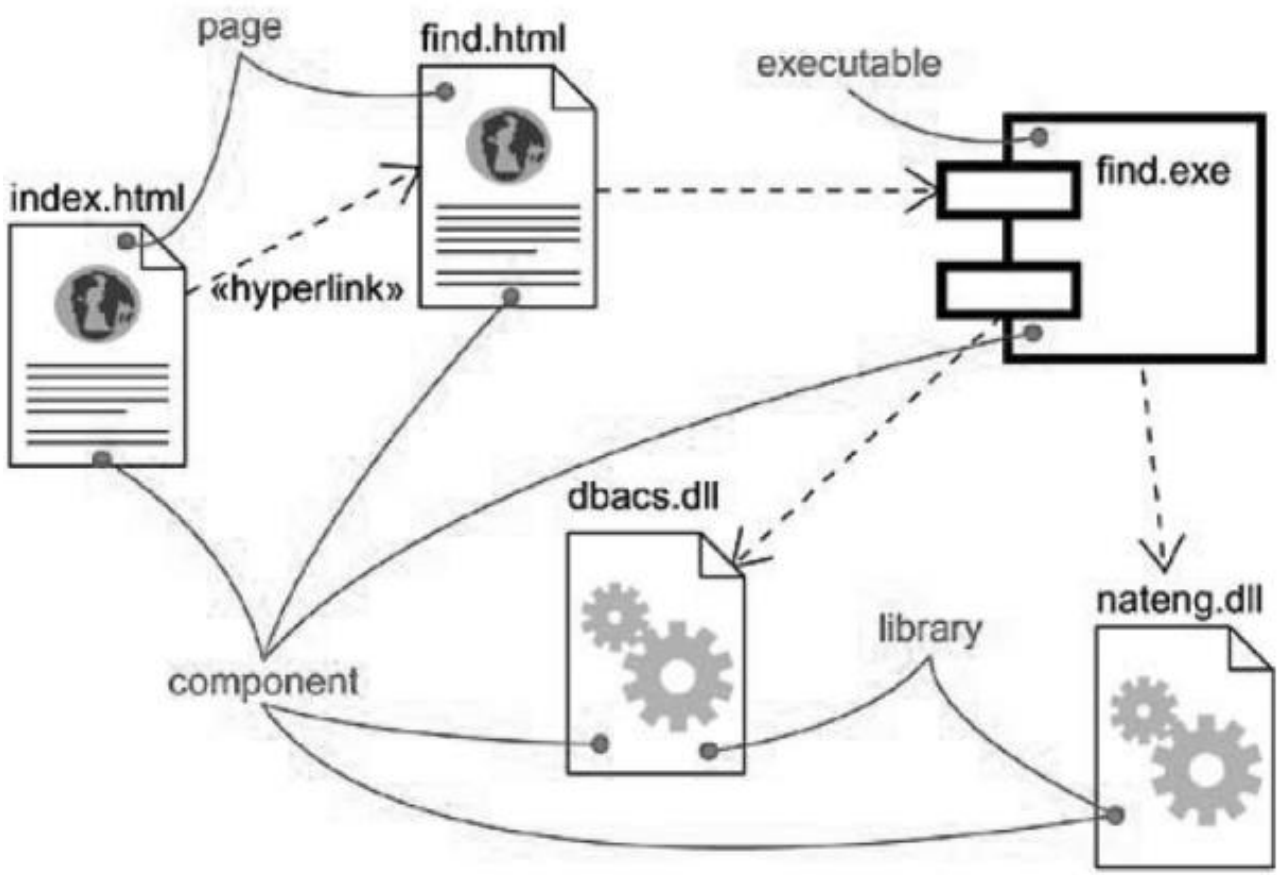


Model a fully distributed system



Component Diagram

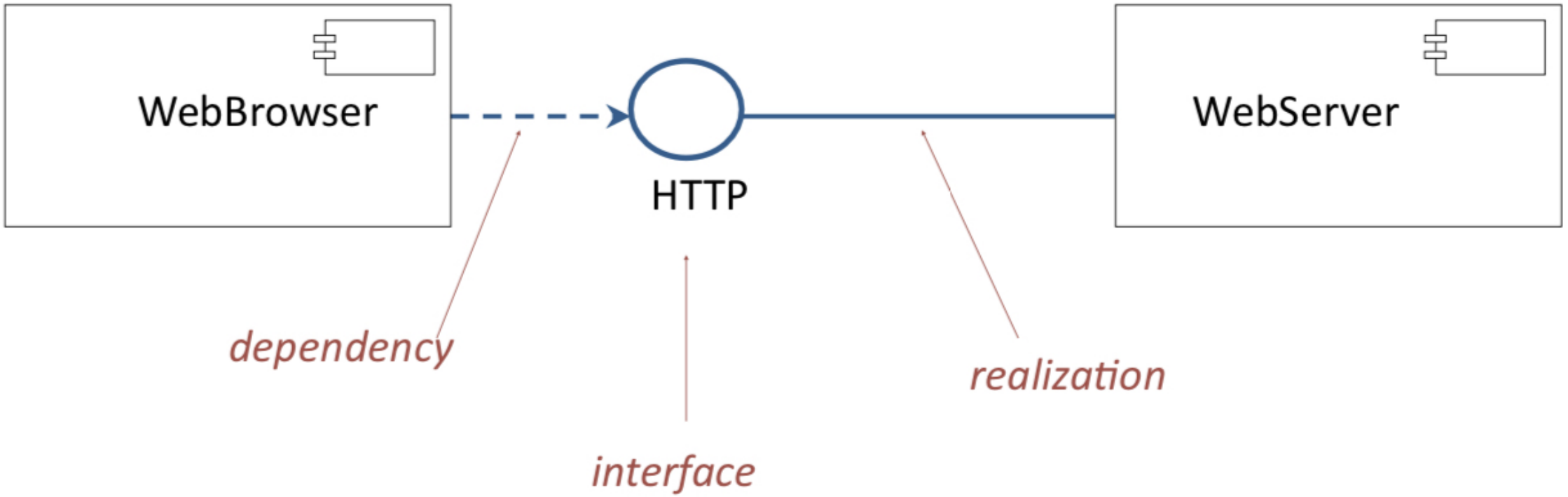
- Component diagrams are one of the two kinds of diagrams for modeling the physical aspects of object-oriented software system
- Show the **organization** and **dependencies** among a set of components
- We use component diagrams to model the **static implementation view** of a software system



Component diagram - content

Component diagram commonly contains:

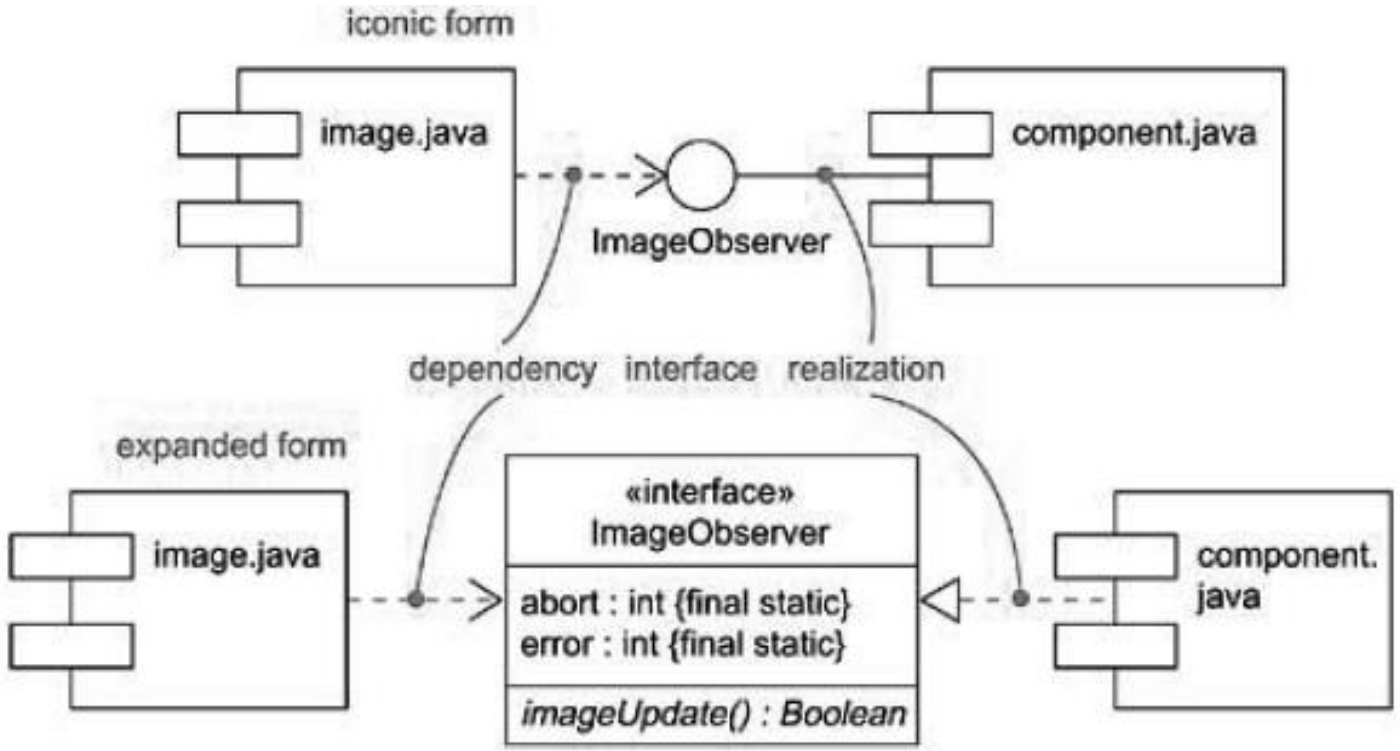
- **Components**
- **Interfaces**
- **Dependency, generalization, association and realization relationships**



Component Diagram: Interfaces

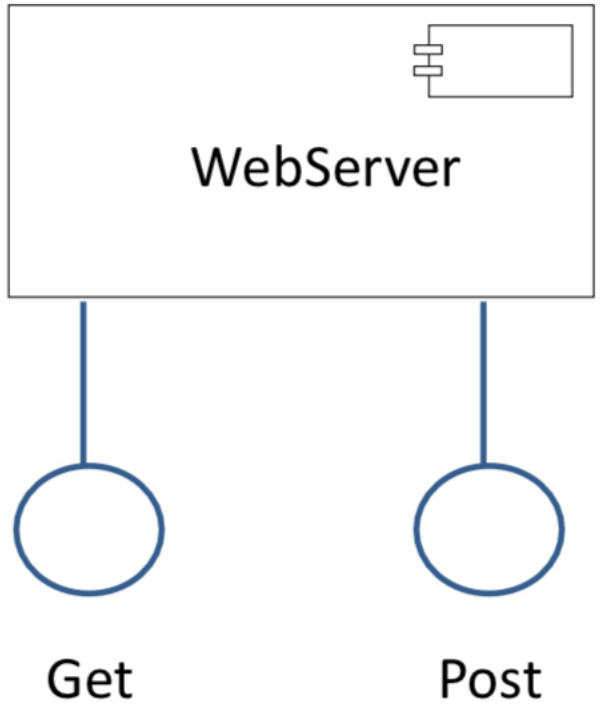
An interface is a **collection of operations** that are used to specify a **service** of a class or a component

- Relationship between a component and its interface can be in two ways
  - Iconic form of **realization**
  - Expanded form (reveal operations) of **realization**
- The component that **accesses** the services of the other component through the interfaces using **dependency**

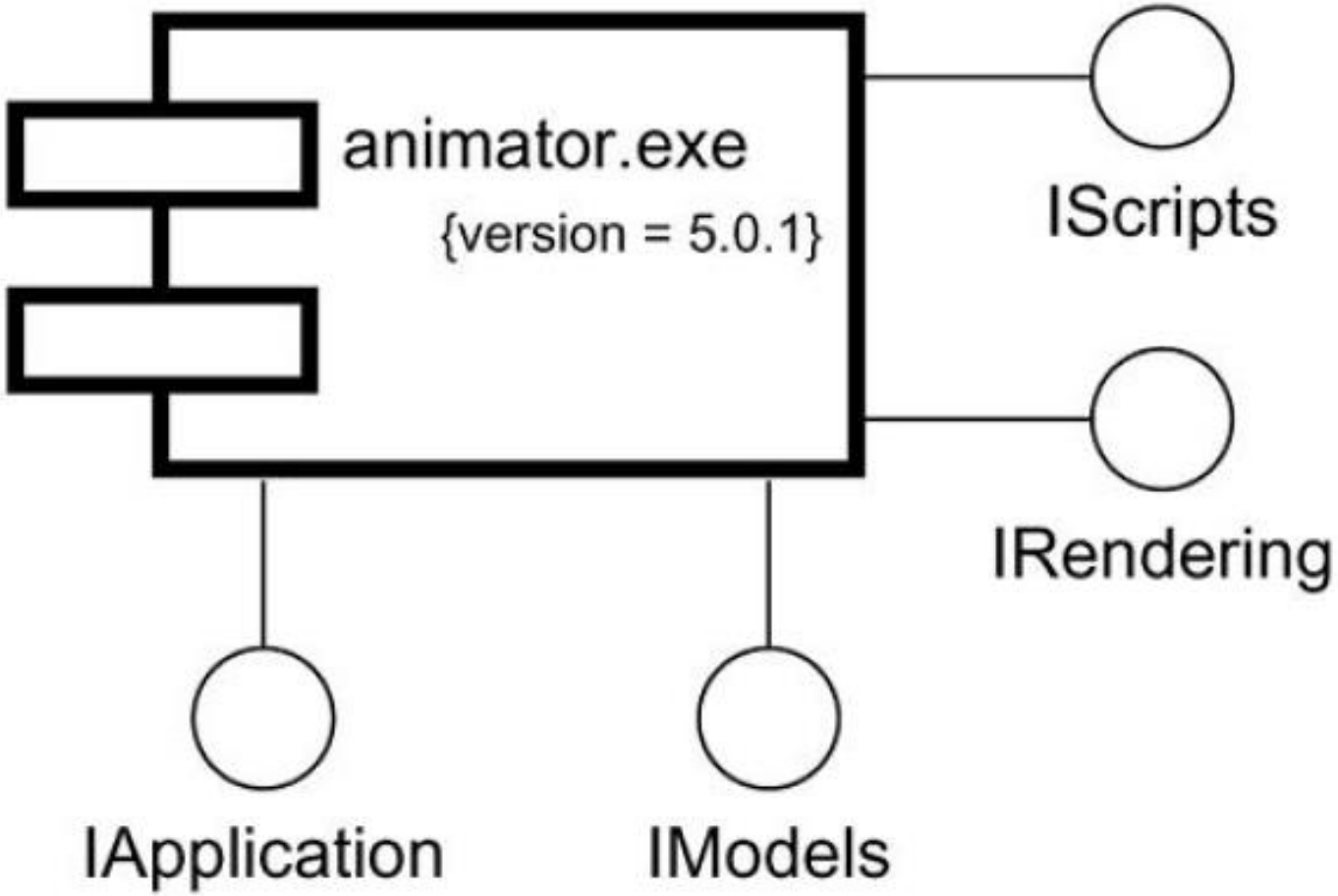


Application Programming Interface (API)

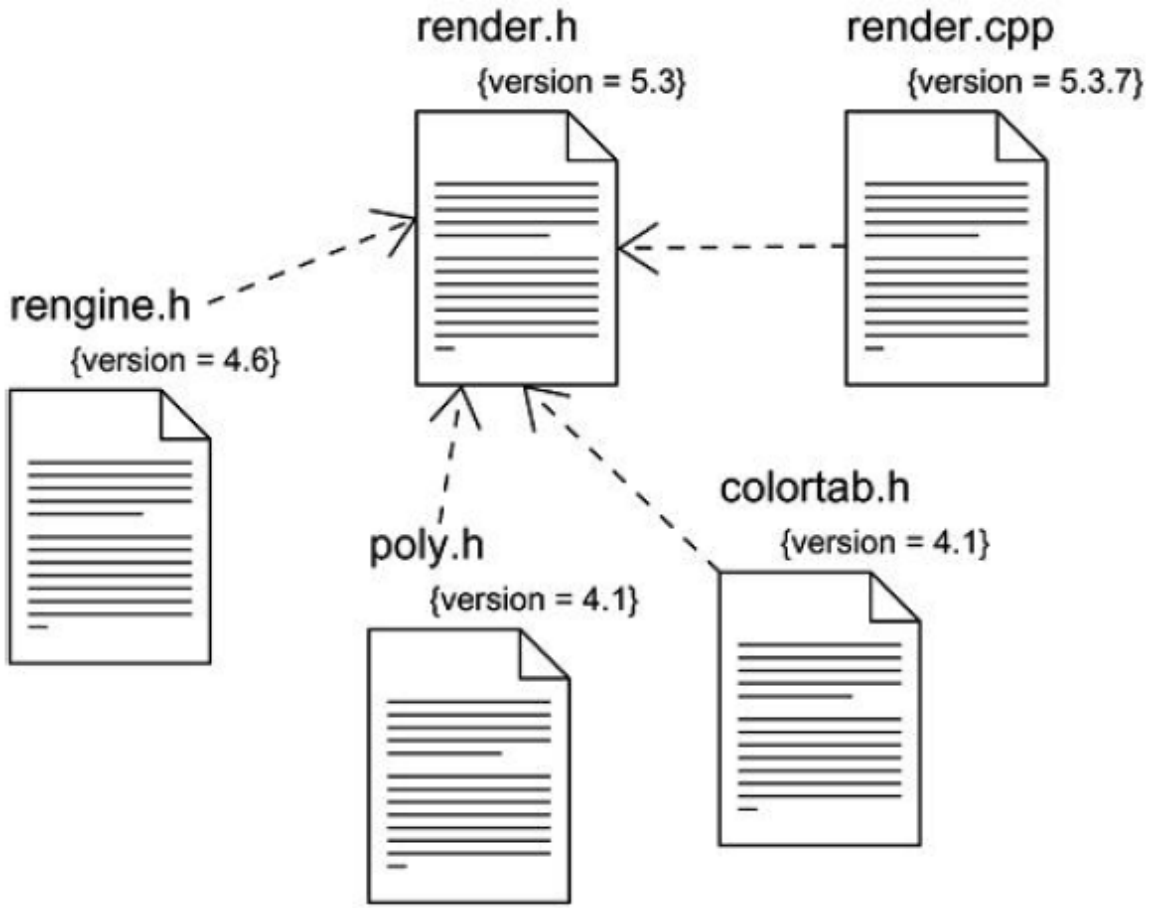
An API is an interface that is realized by one or more components



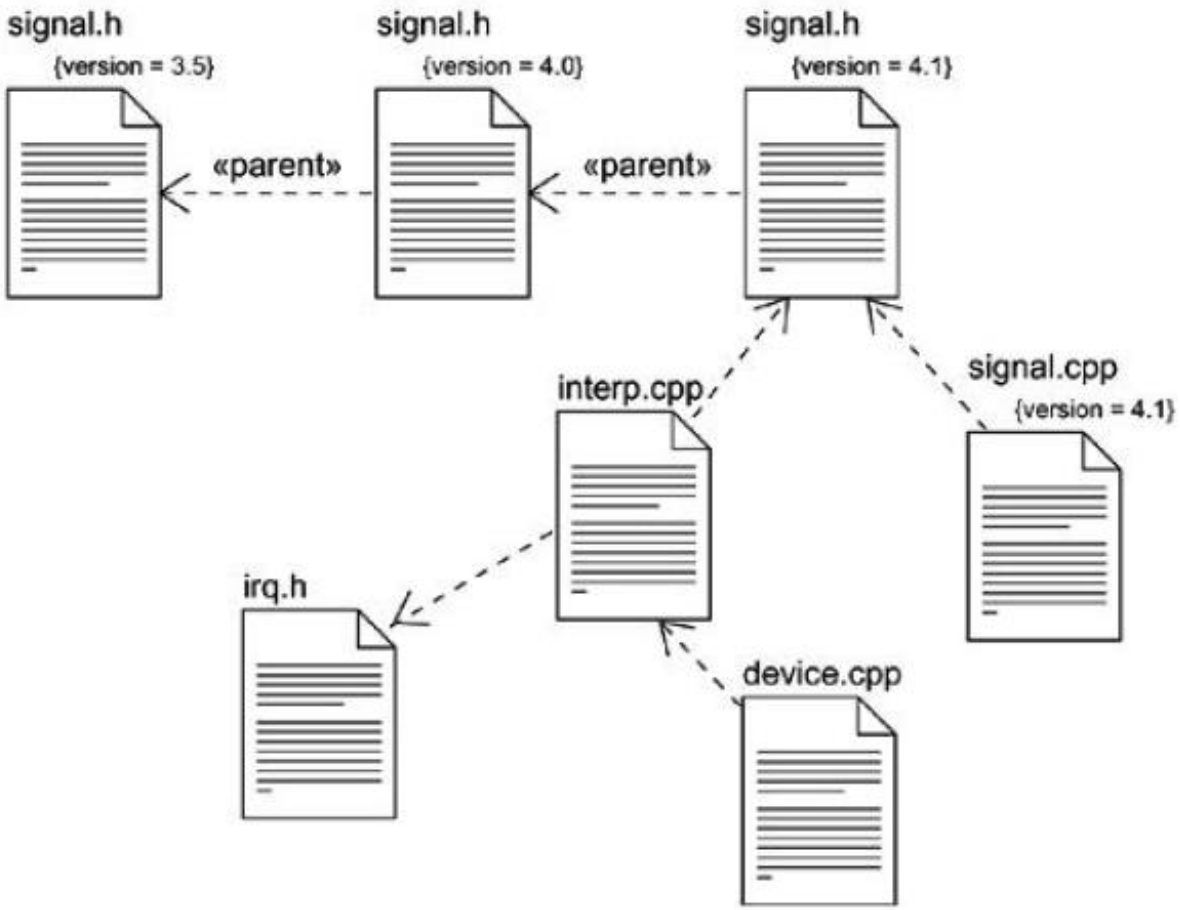
Component diagaram to Model an API



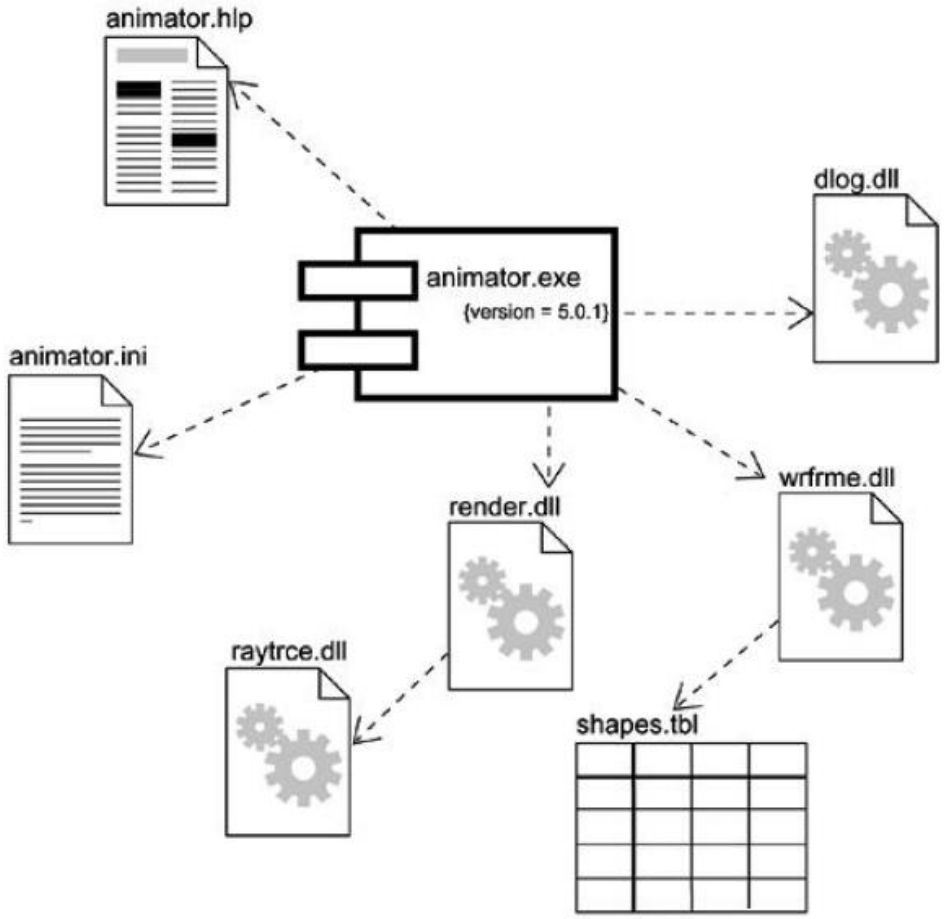
Component Diagram to Model Source Code



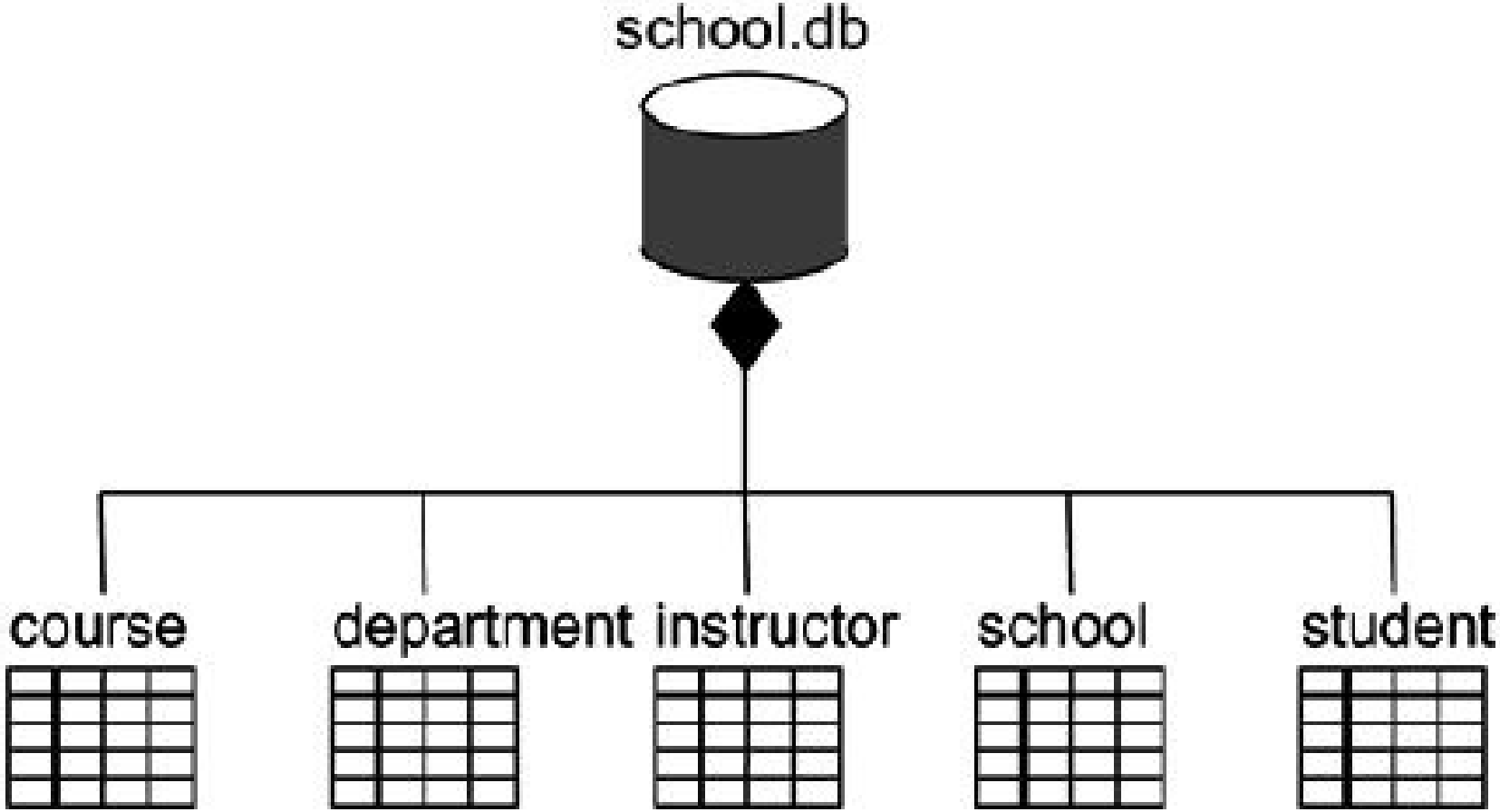
Component Diagram to Model an executable release



Component Diagram to Model Tables, Files, Documents



Component Diagram to Model a physical database



Architectural Styles

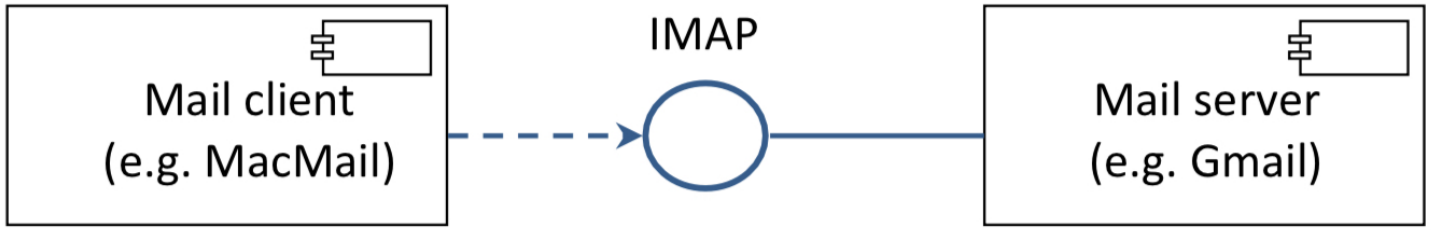
- An **architectural style** is system architecture that recurs in many different applications
- See:
  - Mary Shaw and David Garlan, ***Software architecture: perspective on an emerging discipline***. Prentice Hall, 1996.

Architecture Style: Pipe

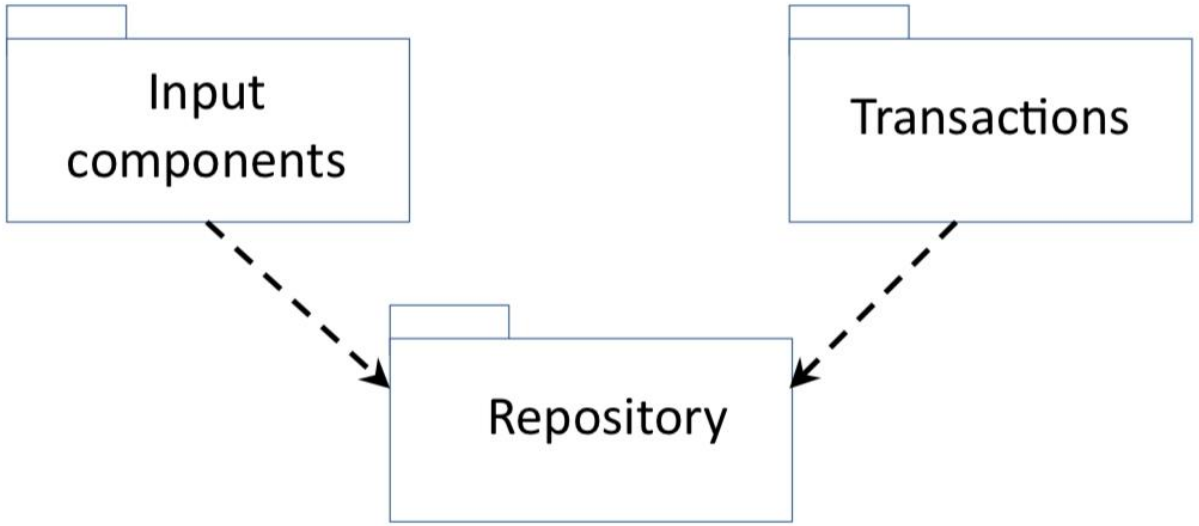
- Example: A three-pass compiler

Output from one subsystem is the input to the next.

Architectural Style: Client/Server



- Example: A mail system
- The control flows in the client and the server are independent
- Communication between client and server follows a protocol
- Because components are binary replaceable elements, either the client or the server can be replaced by another component that implements the protocol
- In a peer-to-peer architecture, the same components act as both client and server

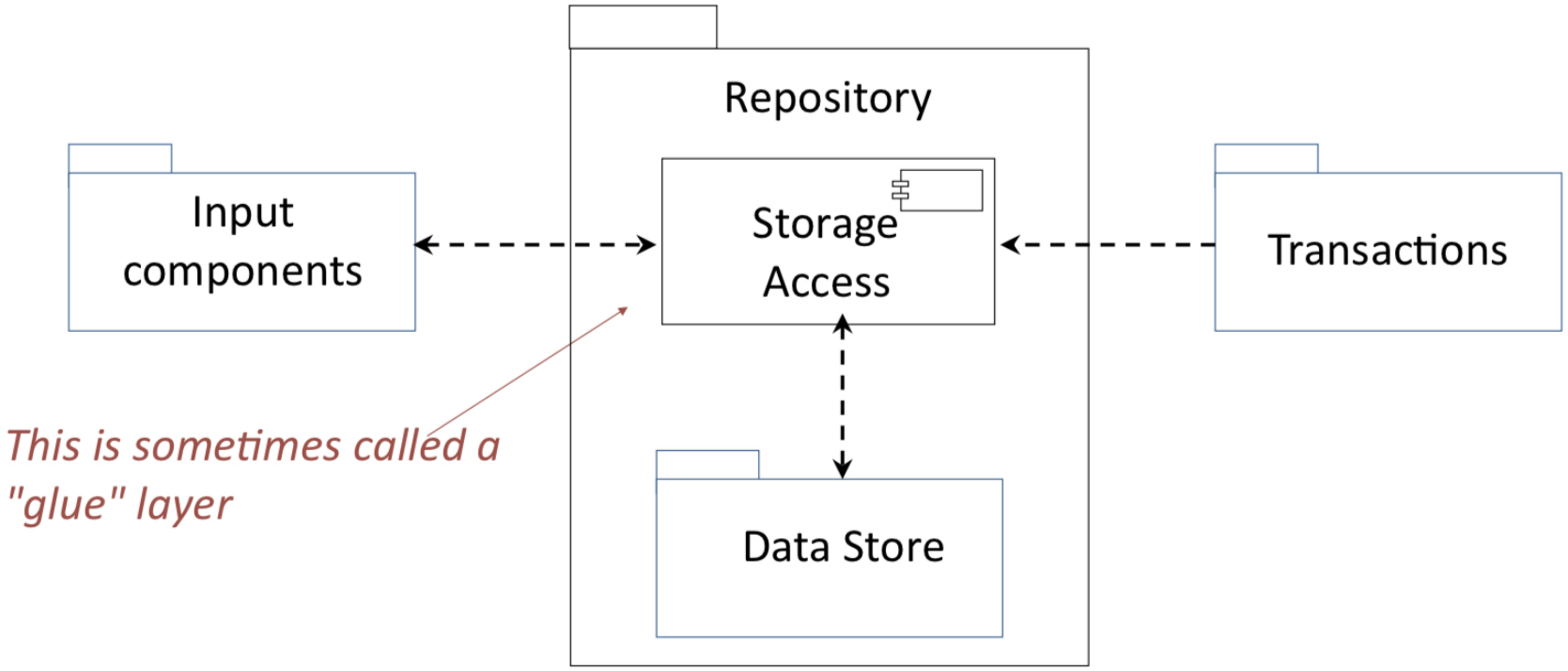


Advantages: Flexible architecture for data-intensive systems.

Disadvantages: Difficult to modify repository since all other components are coupled to it.

Architectural Style: Repository

Architectural Style: Repository with Storage Access Layer



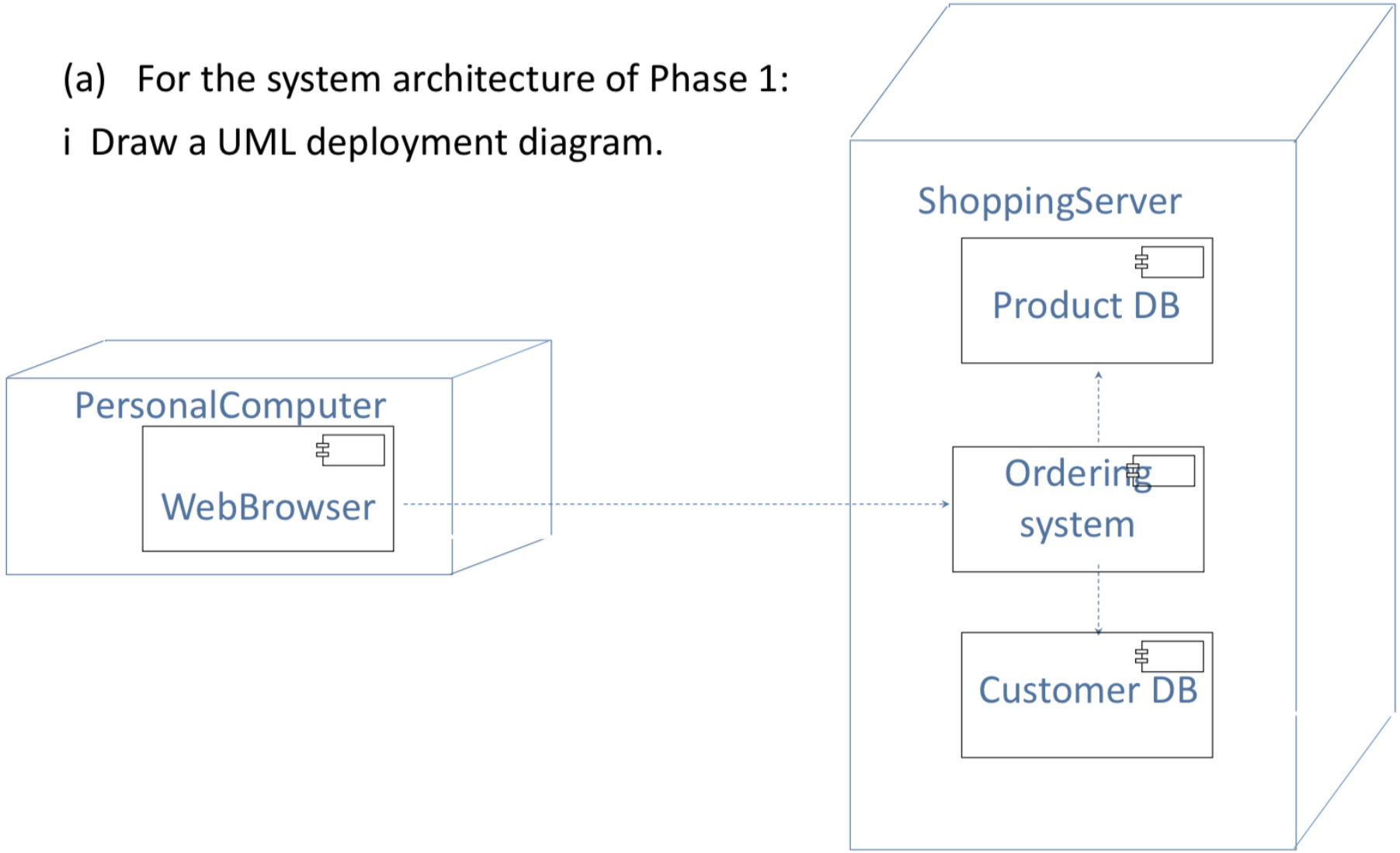
Exercise (Old Exam Question)

- A company that makes sport equipments decides to create a system for selling online. The company already has a **product database** with description, marketing information, and prices of the equipment that it manufactures
- To sell equipment online the company will need to create: a **customer database** and an **ordering system** for online customers
- The plan is to develop the system in two phases. During phase 1, simple versions of the customer database and ordering system will be brought into production. In Phase 2, major enhancements will be made to these components

Solution for Phase 1

(a) For the system architecture of Phase 1:

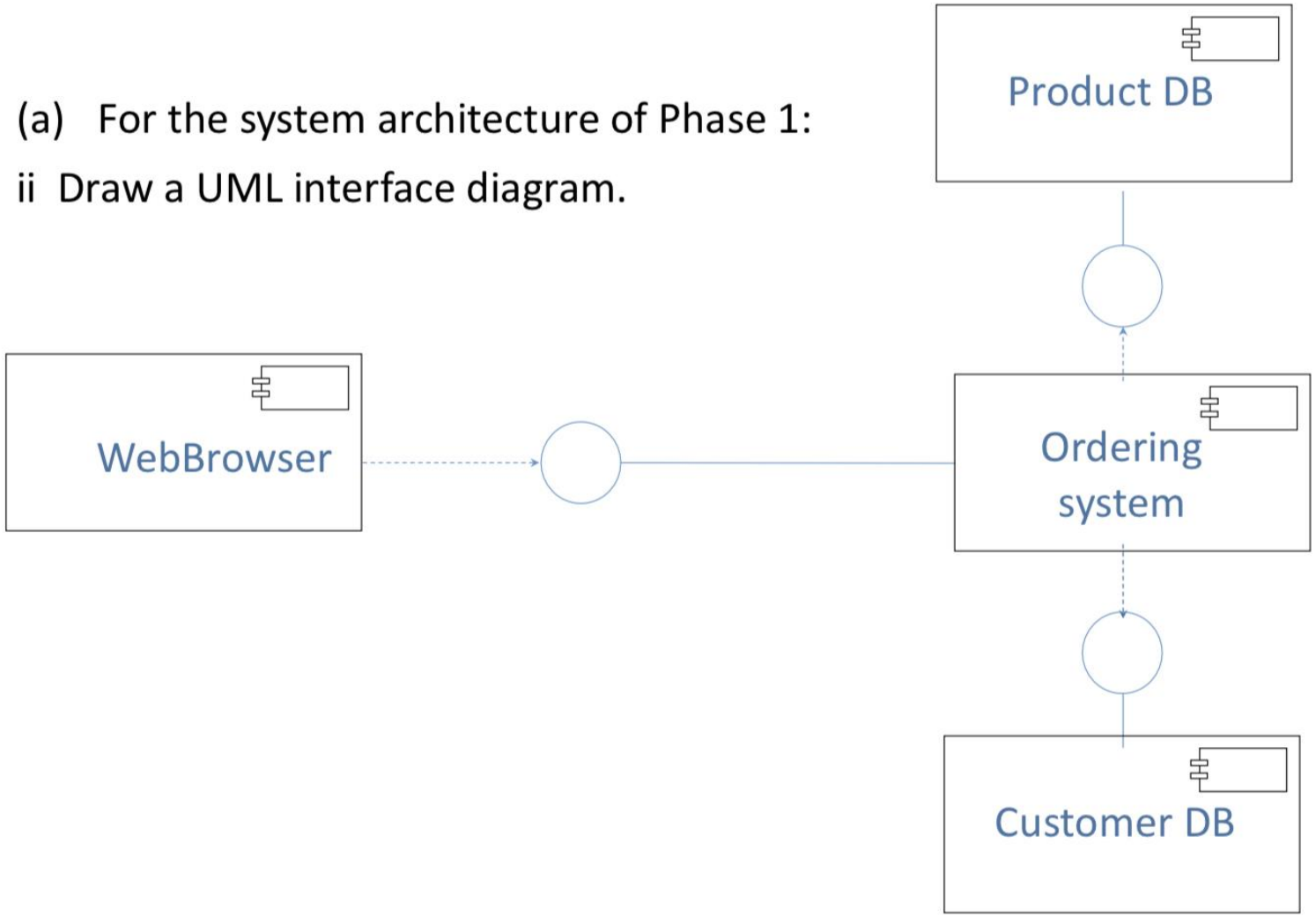
i Draw a UML deployment diagram.



Solution for Phase 1

(a) For the system architecture of Phase 1:

ii Draw a UML interface diagram.



Solution for Phase 2

- (b) For Phase 2:
  - What architectural style would you use for the customer database?

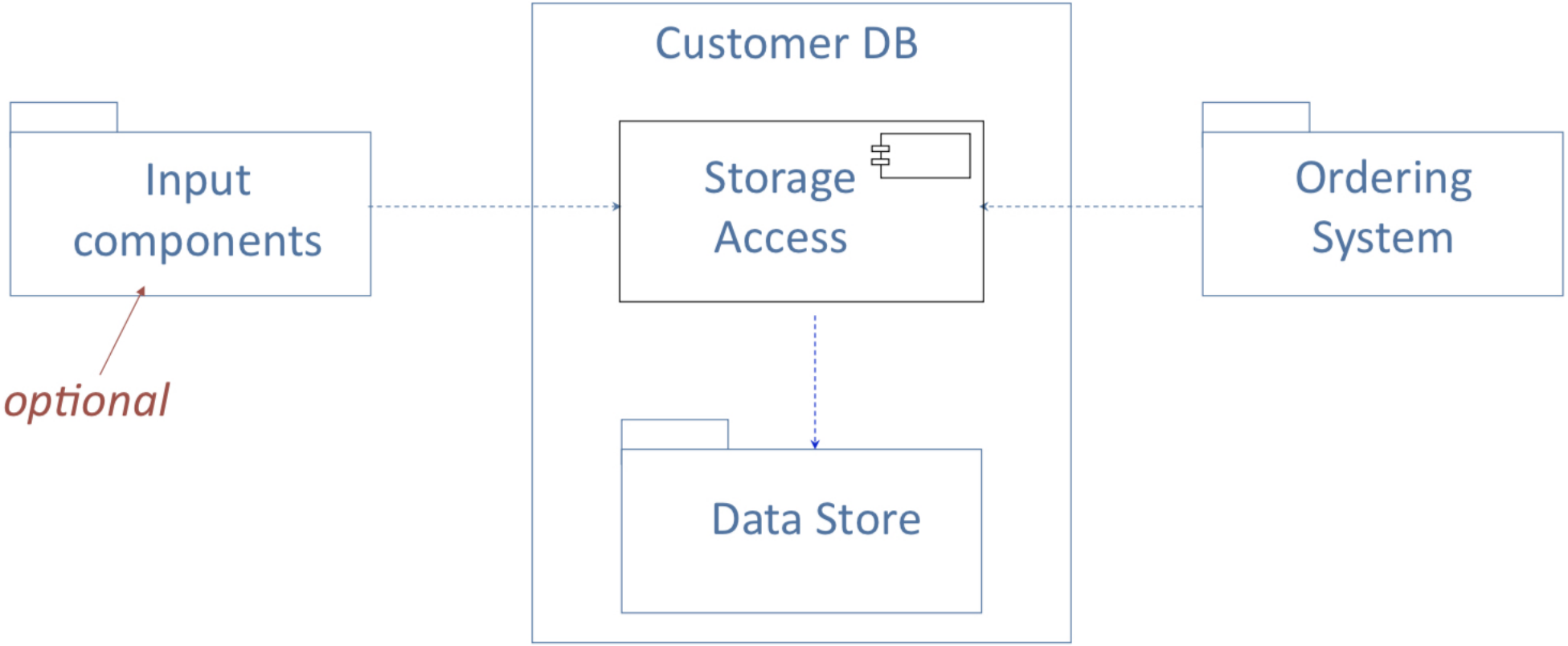
Repository with Storage Access Layer

  - Why would you choose this style?

It allows the database to be replaced without changing the applications that use the database

Solution for Phase 2

- (b) For Phase 2:
  - Draw an UML diagram for this architectural style showing its use in this application



12. System Architecture

(end of lecture)

ONE LOVE. ONE FUTURE.