Software Architecture

Lecture 2. Layered Architecture

Professor:
Yushan (Michael) Sun
Fall 2020

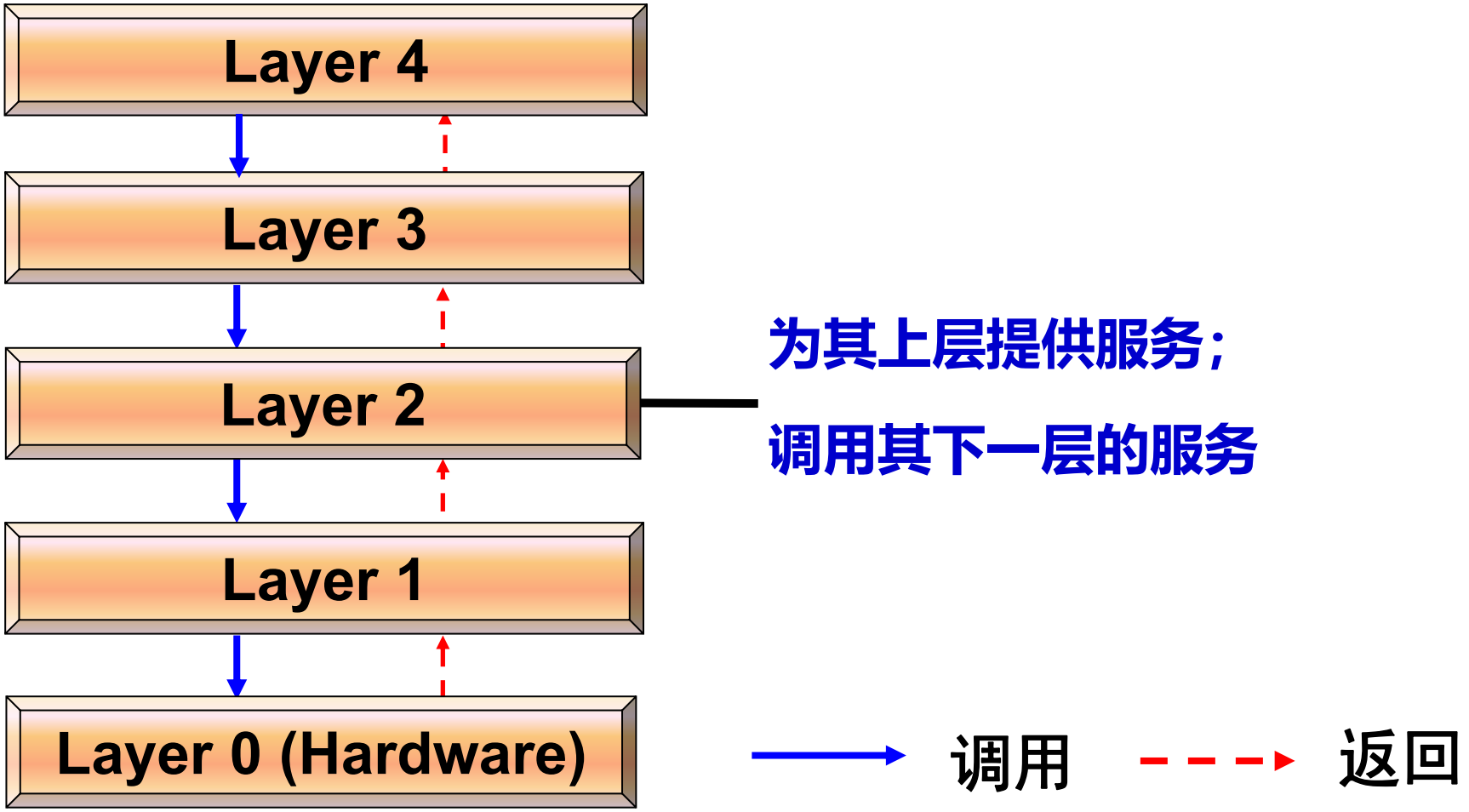
Contents:

Concept of the Layered Architecture

Layered Architecture

- **Definition: A layered system is organized hierarchically. Each layer,**
  - **provides service to the layer above it and**
  - **serves as a client to the layer below.**

Layered Architecture



Layered architecture

Layered Architecture

Note:

- **拓扑限制：交互仅仅在相邻层进行。**
  Topological constraints include limiting interactions to adjacent layers.
  - **不允许隔层调用**
  - **不允许有相反方向的调用**

Layered Architecture

典型应用领域 Typical Application Areas

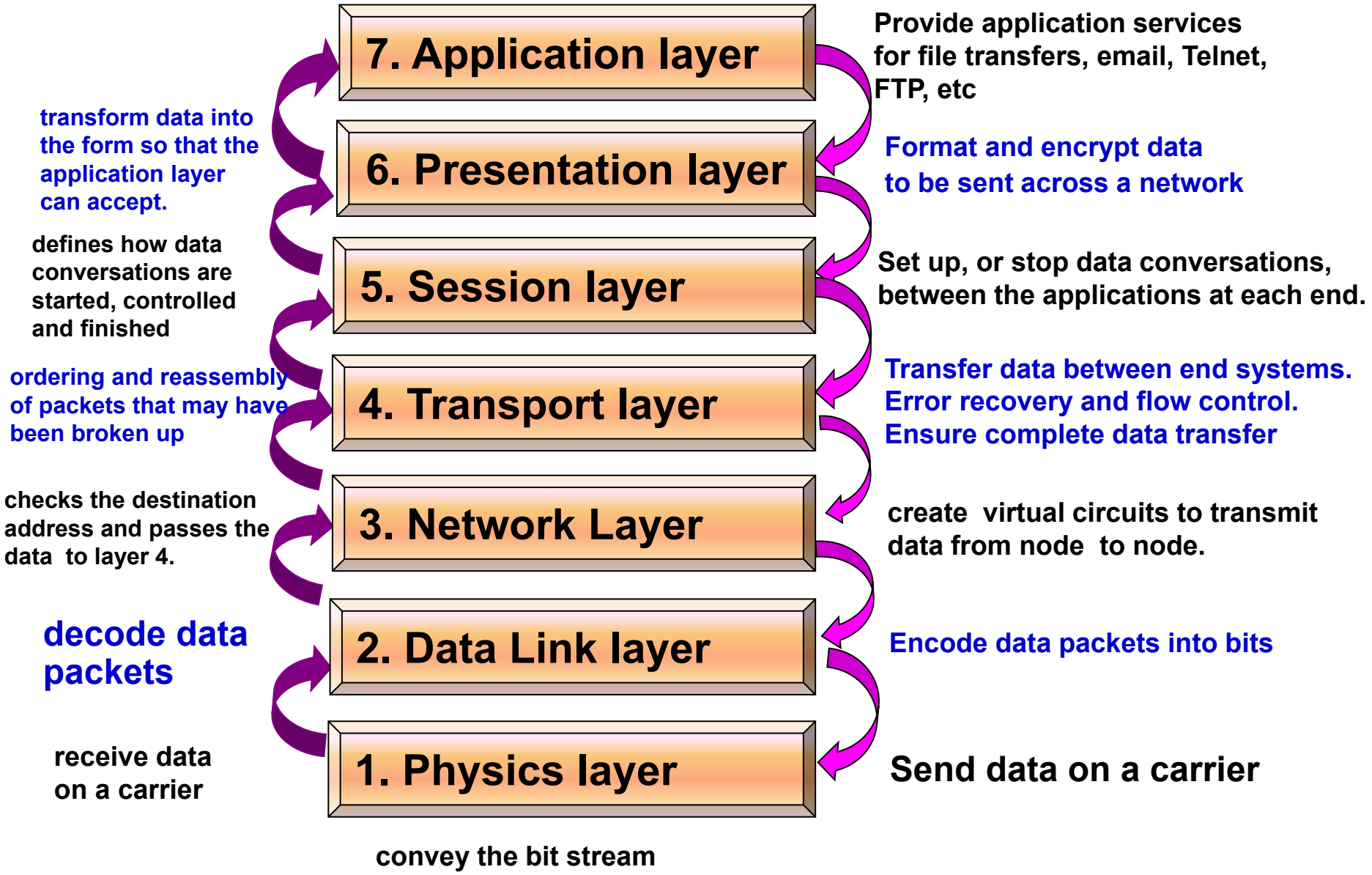
a) 层次通讯协议 Layered Communication Protocols, such as OSI Model, TCP/IP Model
b) 数据库系统 Database systems and
c) 操作系统 Operating systems
d) 应用程序 Application programs (developed in .NET and JavaEE)

Layered Architecture

- 7层层次通讯协议

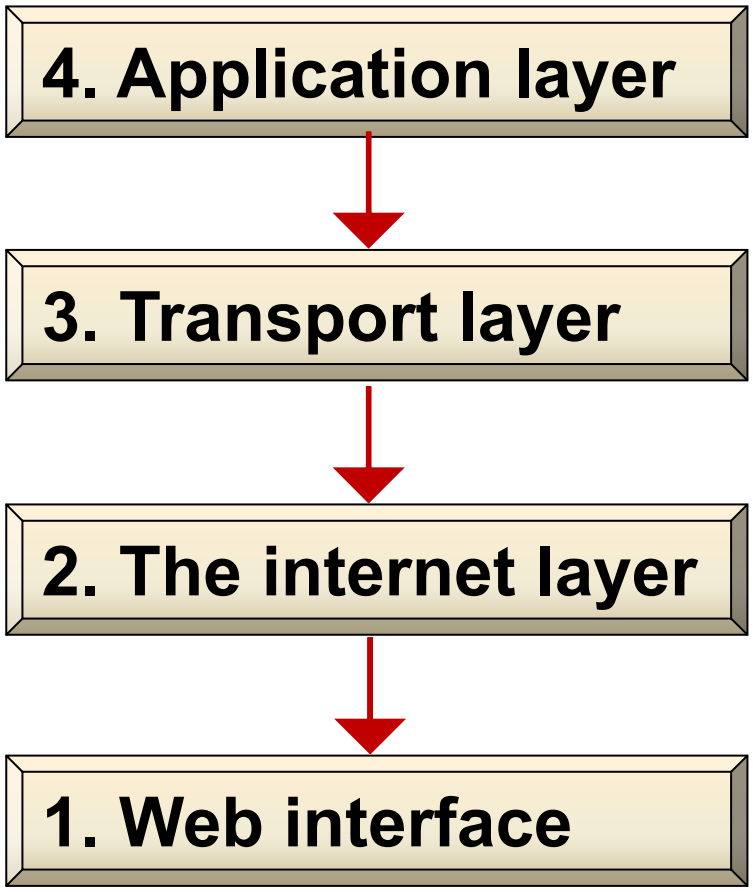
  Layered communication Protocols :
- 每层都提供在一定层次上的通讯模块。Each layer provides a substrate for communication at some level of abstraction.
  - 低层定义低层交互。Lower levels define lower levels of interaction.
  - 最低层通常由硬件连接。The lowest level is typically defined by hardware connections.

Layered Architecture



Layered Communication Protocols- OSI Model

Layered Architecture



telnet, ftp, http

发送端, 中间端,

ip

network card

Layered Communication Protocols- TCP/IP Model

Layered Architecture

Advantages of Layered systems

1. 支持逐层抽象 Layered architecture supports designs based on increasing levels of abstraction.
   - This allows implementers to partition a complex problem into a sequence of incremental steps.
2. 支持更新 Layered architecture supports enhancement.
   - each layer interacts with at most the layers below and above,
   - changes to the function of one layer affect at most two other layers.

Layered Architecture

3. **支持复用 Layered architecture supports reuse**.
   - One can have **different implementations** of the same layer, provided they support the same interfaces to their adjacent layers.
   - With standard layer interfaces, it can be built by different software development teams. **(like OOP)**

Layered Architecture

Disadvantages of layered systems

- Not all systems are easily structured in a layered fashion.
  - Even if a system can logically be structured in layers, considerations of **performance** may require **closer coupling** between
    - logically **high-level functions** and their
    - **lower-level implementations**.
  - It may be quite difficult to find the right levels of abstraction.

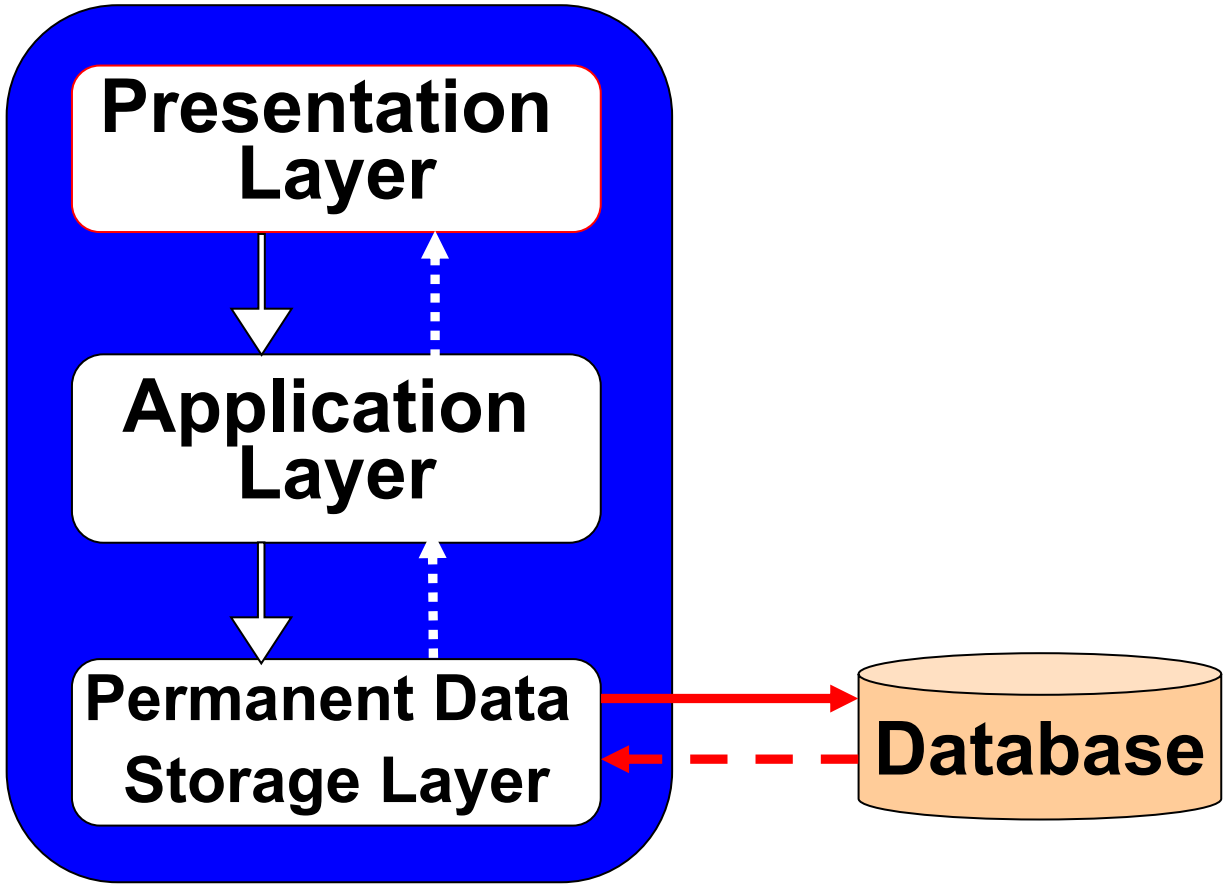
Back

3 Layer Software Architecture

Three Layer Architecture

- 三层架构是近年来经常使用的软件体系结构。该体系结构可以分为通用的3层架构与运行在互联网上的三层架构软件。
  - **运行在互联网上的三层架构**被称为三层客户端-服务器(3-tired Client/server)架构
  - **通用三层架构**。本节要讲述的是通用的三层架构。

Three Layer Architecture



传统的三层的层次体系结构-与Internet无关

Three Layer Architecture

- **表示层(Presentation layer):**
  - 表示层通常包括用户图形界面，用于用户输入、用户请求与显示用户请求的返回结果等。
  - 表示层调用应用层组件的过程，函数或者方法。但是应用层从来不会调用表示层的功能。

Three Layer Architecture

- **应用层(Application layer)**：或者称为业务逻辑(Business Logic)层。主要包括应用的业务逻辑，实现了应用的商业功能。
  - 该层的组件封装了应用的核心数据与功能。
  - 在该层中，如果要访问数据库或者要将程序运行中产生的数据存储到数据库，必须调用永久数据存储层的相应的数据库访问方法，而不能直接访问数据库。

Three Layer Architecture

- **永久数据存储层(Permanent Data Storage layer)**：该层包含数据库的访问与将永久数据存储到数据库中的功能。
  - 在Java实现中，该层包含了利用JDBC写的数据库访问代码。
  - 该层不能调用应用层与显示层，而只能将执行应用层的请求对数据库的访问结果返回给应用层。而应用层有可能将该数据返回给显示层。

Back

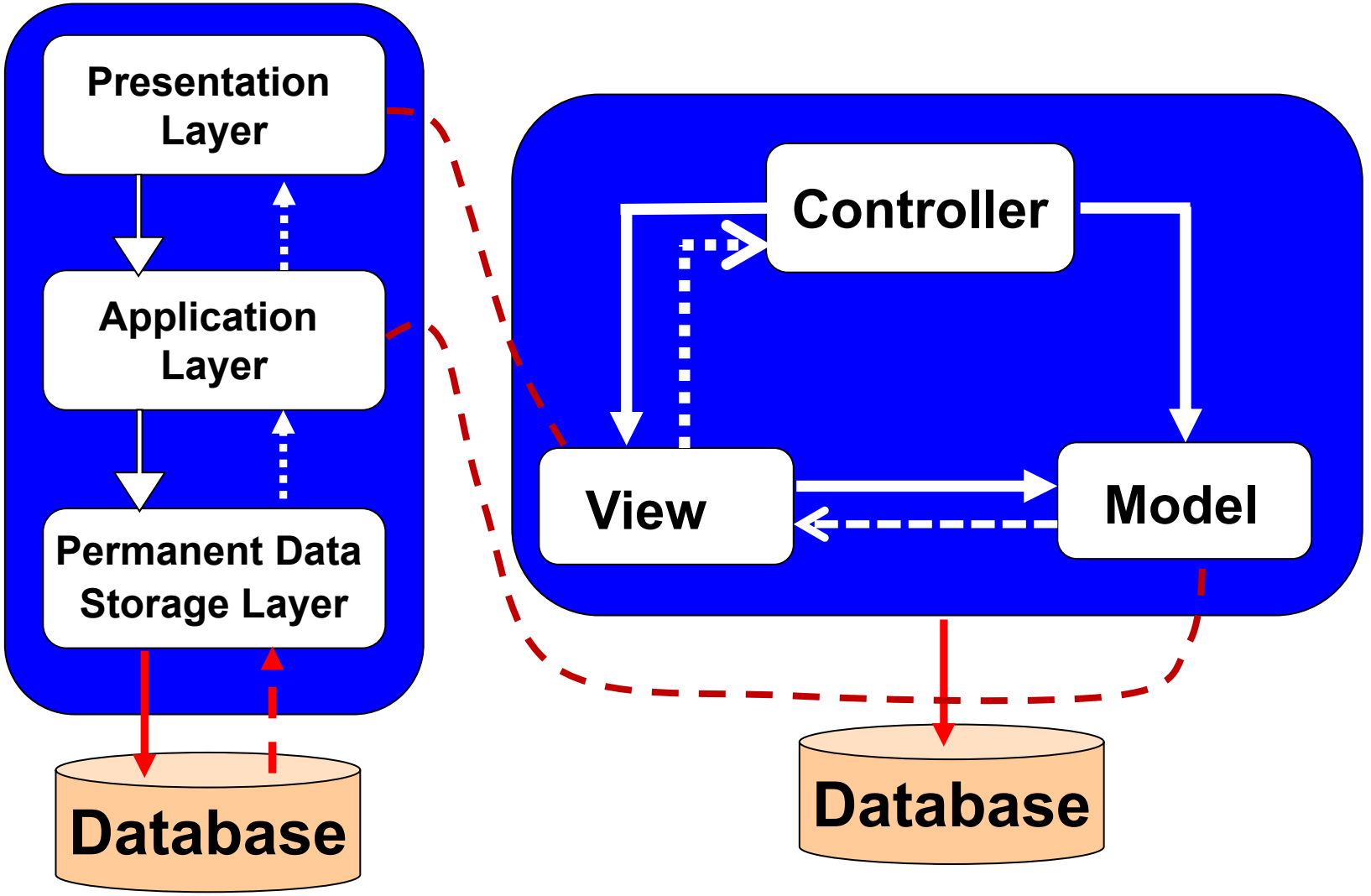
三层层次架构与MVC软件体系结构的比较

三层层次架构与MVC软件体系结构的比较

- **在形式上，三层架构类似于MVC 架构，都是由三部分软件模块组成的，但是实际上它们是不同的。**

- **两种架构相似之处如下**：
  a) 都是由三部分软件模块组成的
  b) 三层架构的显示层与MVC架构的View类似；
  c) 三层架构的应用层与MVC架构的Model类似。

三层层次架构与MVC软件体系结构的比较



3层层次架构与MVC架构

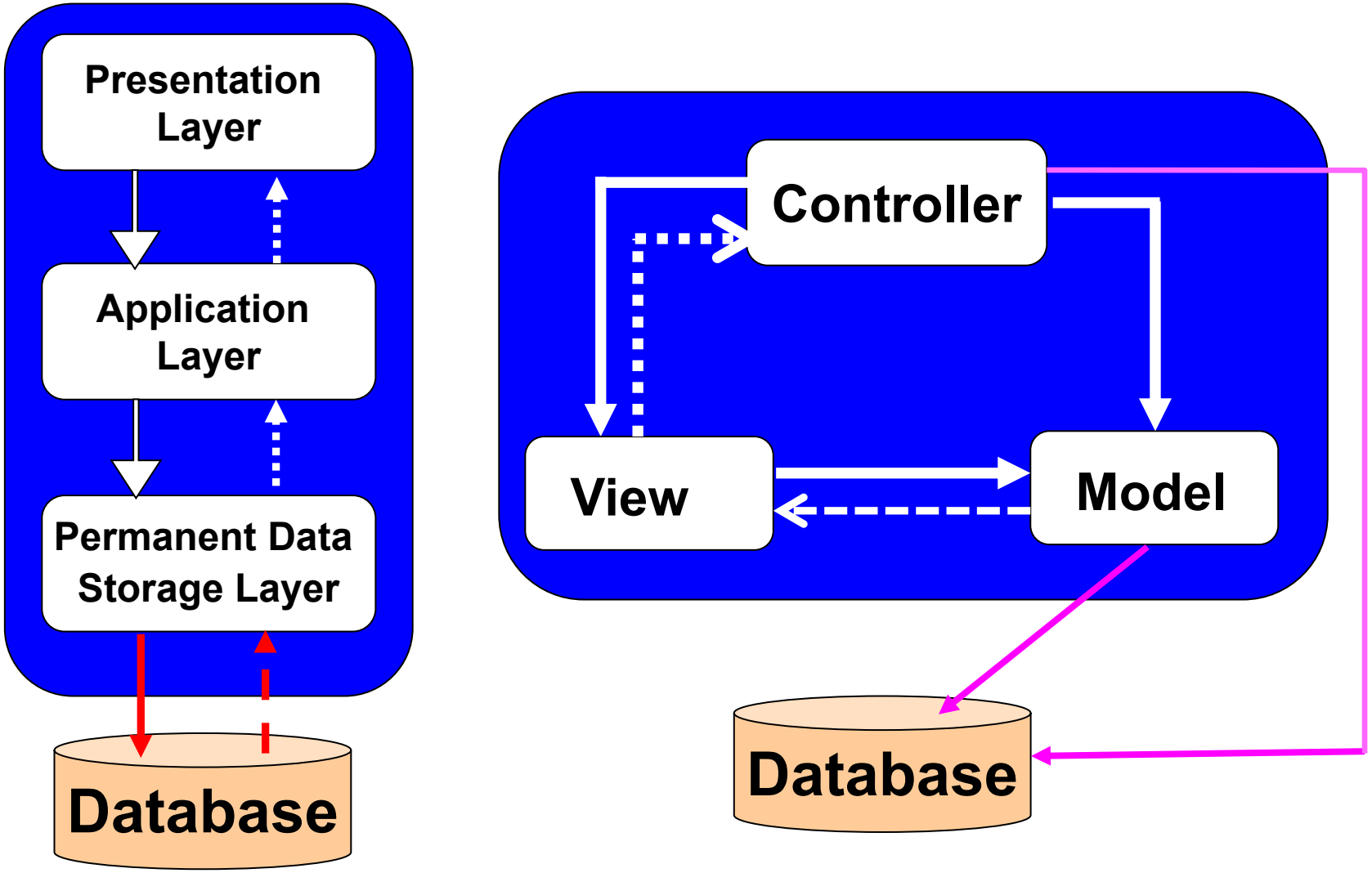
三层层次架构与MVC软件体系结构的比较

- **三层体系结构与MVC体系结构的区别如下：**
- **区别1：各个模块之间的调用关系不同**
  - 三层架构的一个根本原则是显示层不能直接越过应用层直接调用永久数据存储层的代码。首先显示层必须调用应用层，然后再由应用层的相关的方法转而调用永久数据存储层。不允许有相反方向的调用。因此，三层架构的交互是线性的；
  - MVC架构的程序组件之间的交互是三角形的：View对象发送更新给Controller对象，Controller对象更新Model对象，并且View对象直接地从Model对象得到更新。

三层层次架构与MVC软件体系结构的比较

- **区别2：对数据库的访问方式不同。**
  - 三层架构指定一个永久数据访问层，所有对数据库的访问均有此层承担；
  - MVC架构没有指定专门的数据库访问模块，一般的情况下是由Model直接访问数据库，但是也没有排除Controller中直接访问数据库的可能。

三层层次架构与MVC软件体系结构的比较

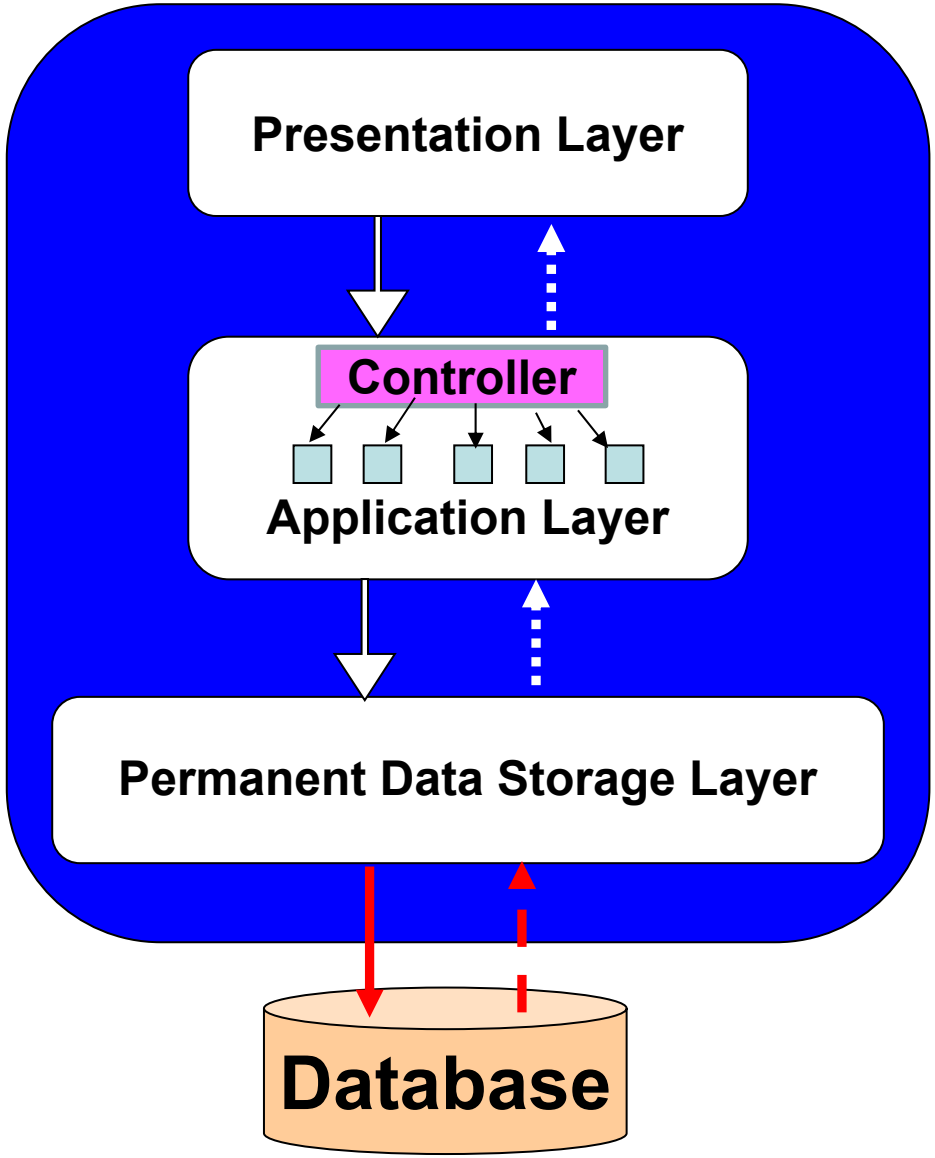


3层层次架构与MVC架构访问数据库的情况

三层层次架构与MVC软件体系结构的比较

- **区别3：关于控制模块(Controller)。**
  - MVC架构中存在专门的Controller模块；
  - 而层次架构中通常没有这样专门的模块。事实上，很多设计者在层次架构中的应用层里面单独地指定一个类似的控制模块。

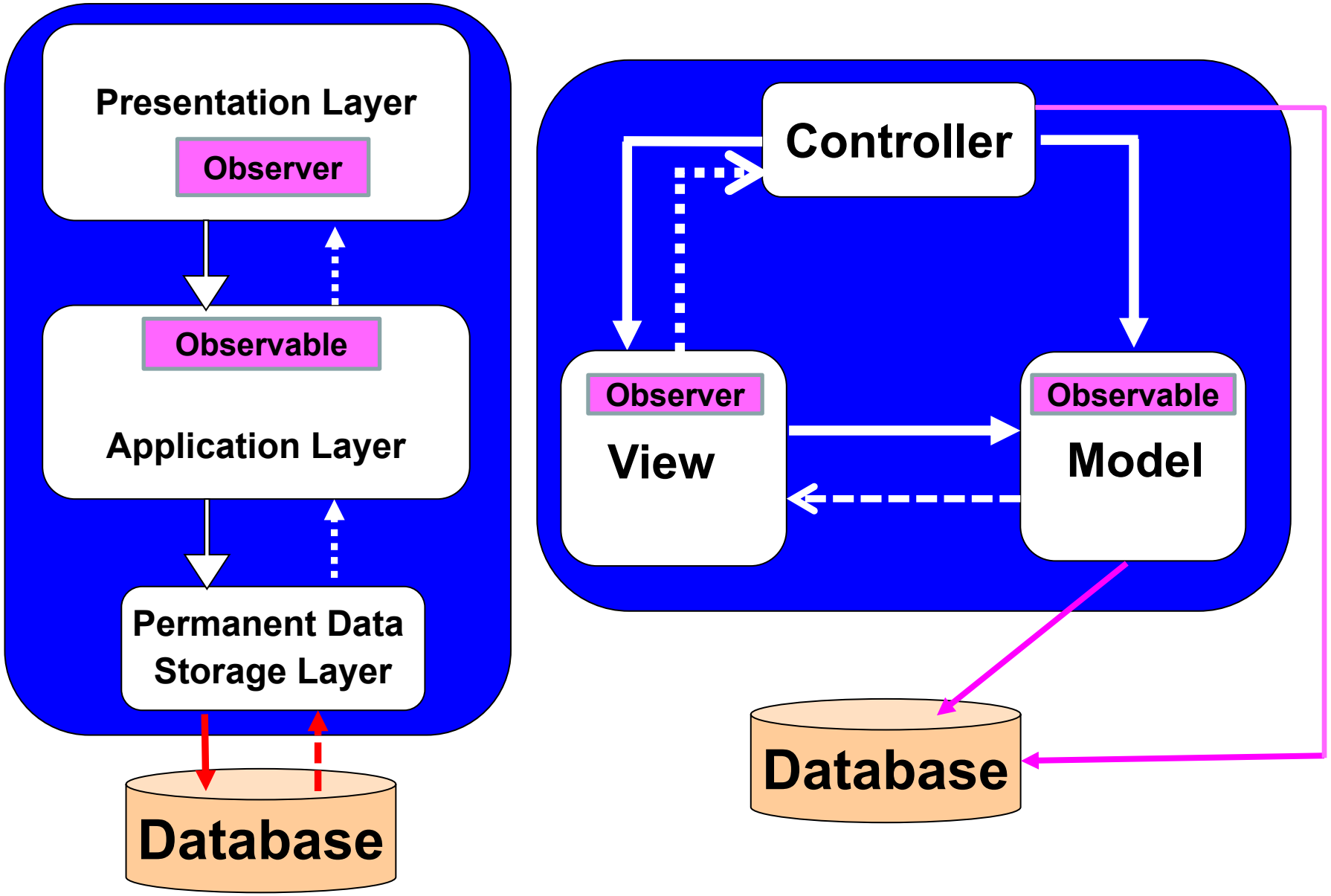
三层层次架构与MVC软件体系结构的比较



很多设计者在层次架构中的应用层里面单独地设计一个Controller模块，其主要责任是接收从显示层输入的用户请求，然后根据请求的类型，决定调用应用层的其它组件。

三层层次架构与MVC软件体系结构的比较

- **关于观察者模式在三层架构与MVC架构中的使用**
  - **在三层架构中，可以在应用层与表示层之间，使用观察者模式**：将应用层作为被观察者，将表示层作为观察者。其目的是一旦应用层的状态改变的时候，及时通知表示层，以便及时刷新用户图像界面。
  - **在MVC架构中，可以在Model与View之间使用观察者模式**： Model作为被观察者， View作为观察者。一旦Model的状态改变的时候，及时通知View，以便及时刷新View。



使用观察者模式的3层层次架构与MVC架构

三层层次架构与MVC软件体系结构的比较

【注】在3层架构中，应用层与表示层之间，使用观察者模式。必然导致应用层对显示层的调用(notifyObservers())。这违反了层次架构的原则。但是考虑到采用观察者机制更新图形界面的效率，以上的小小的“违规”也是值得的。

Back

Case studies

案例分析

2. Case Studies

Case study 1: Banking System

Case study 2: Mobile Robotics (移动机器人技术)

Back

2. Case Studies

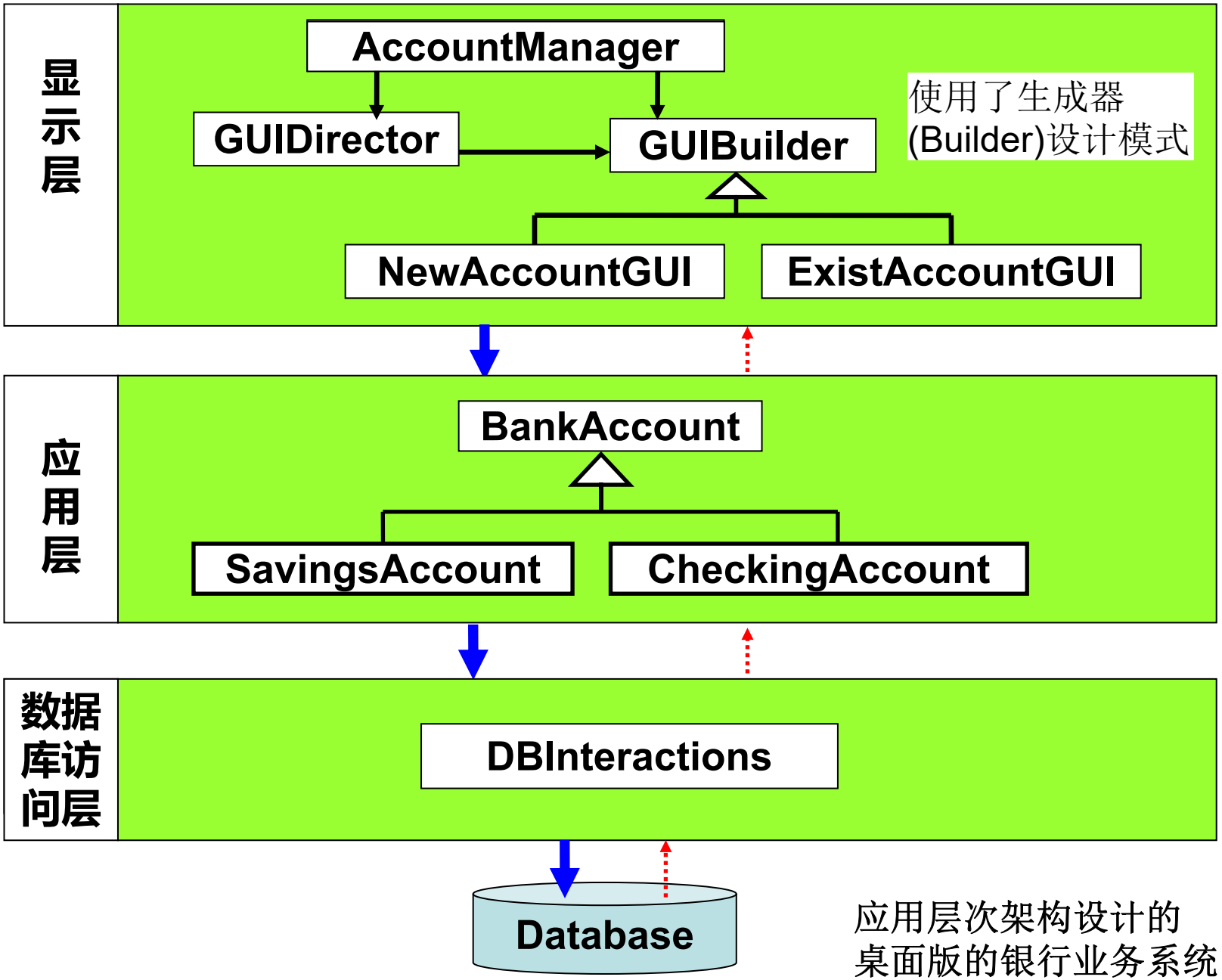
Case 1: Banking System

2. Case Studies

【例1】桌面版银行业务系统。

- 设计并实现一个桌面版的银行业务系统。功能：
  - 存款；
  - 取款；
  - 新开账户；

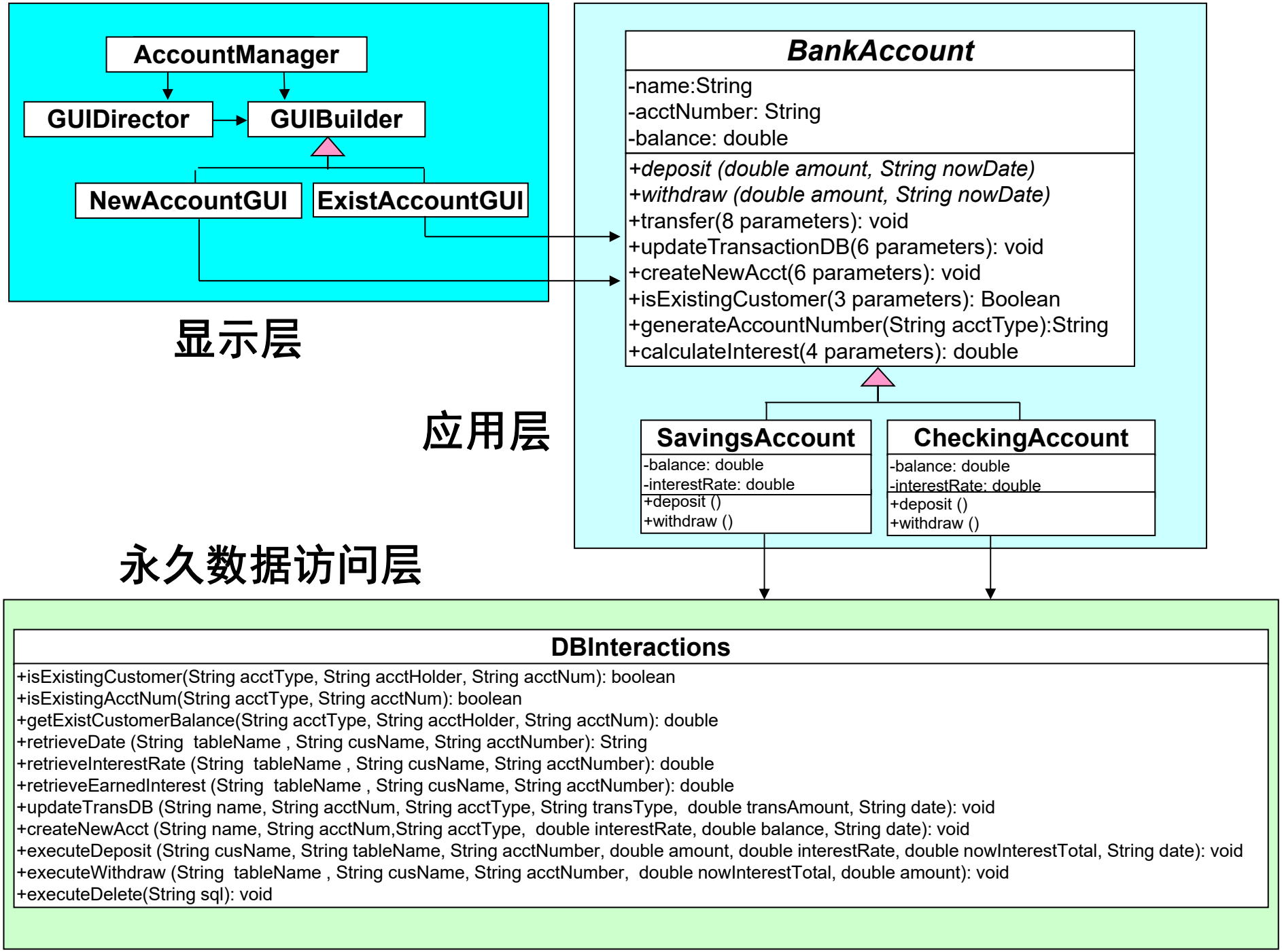
  等等。
- 在账户类型方面，设置了
  - 定期存款账户与
  - 支票账户。
- 该系统采用层次结构设计如下图所示。



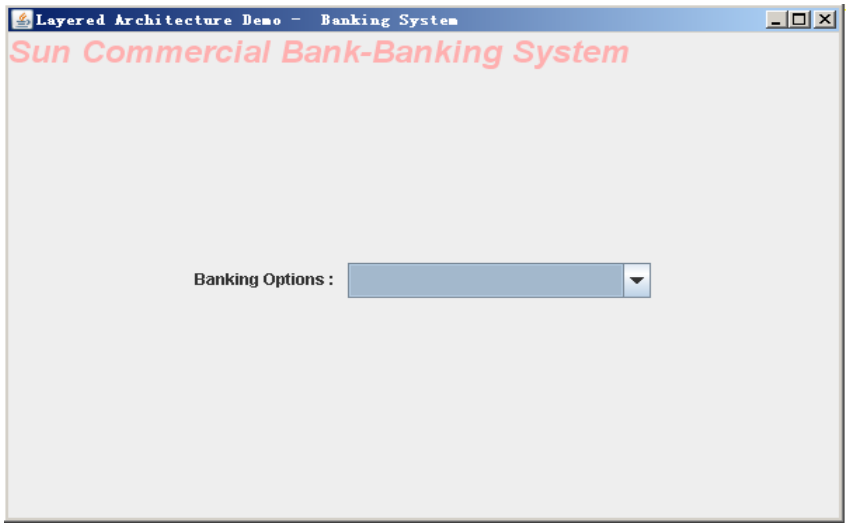
应用层次架构设计的桌面版的银行业务系统

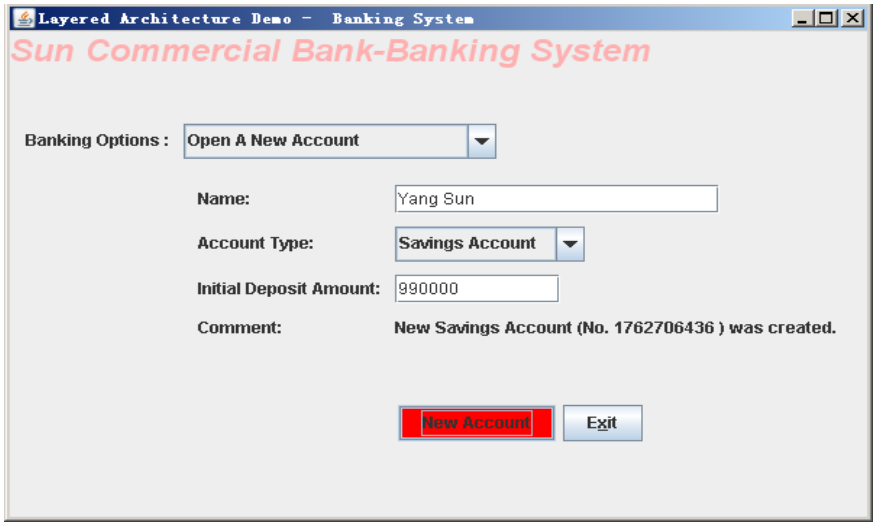
2. Case Studies

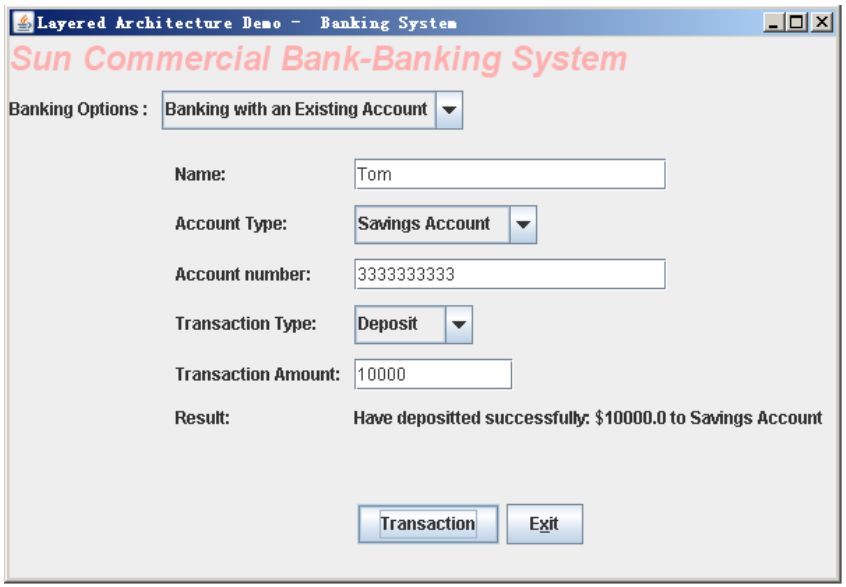
- 该软件被设计为三层架构，包括显示层，应用层，数据库访问层。
  - 在显示层，出于要产生并且在不同的用户界面之间的切换的需要，采用了builder设计模式。
  - 在应用层，封装了银行账户的核心功能部分，包括各种类型的账户与相应的功能。
  - 在永久数据存储层包括用于访问数据库的JDBC代码。在数据库的实现中采用Microsoft Access数据库。
- 桌面版的银行业务系统的设计类图如下图所示。

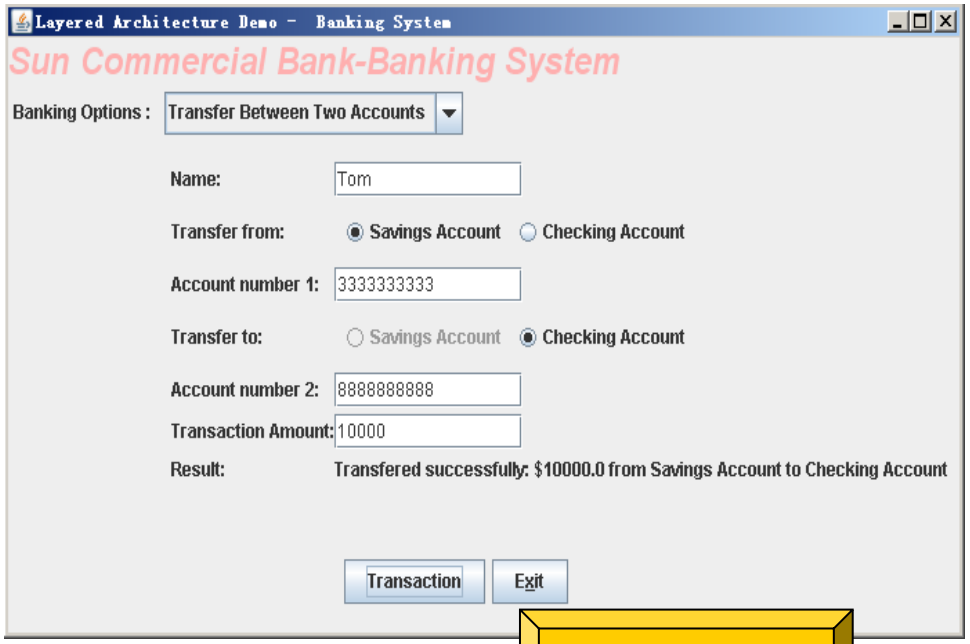

AccountManager
GUIDirector
GUIBuilder
NewAccountGUI
ExistAccountGUI
显示层
BankAccount
-name:String
-acctNumber: String
-balance: double
+deposit (double amount, String nowDate)
+withdraw (double amount, String nowDate)
+transfer(8 parameters): void
+updateTransactionDB(6 parameters): void
+createNewAcct(6 parameters): void
+isExistingCustomer(3 parameters): Boolean
+generateAccountNumber(String acctType):String
+calculateInterest(4 parameters): double
应用层
SavingsAccount
-balance: double
-interestRate: double
+deposit ()
+withdraw ()
CheckingAccount
-balance: double
-interestRate: double
+deposit ()
+withdraw ()
永久数据访问层
DBInteractions
+isExistingCustomer(String acctType, String acctHolder, String acctNum): boolean
+isExistingAcctNum(String acctType, String acctNum): boolean
+getExistCustomerBalance(String acctType, String acctHolder, String acctNum): double
+retrieveDate (String tableName , String cusName, String acctNumber): String
+retrieveInterestRate (String tableName , String cusName, String acctNumber): double
+retrieveEarnedInterest (String tableName , String cusName, String acctNumber): double
+updateTransDB (String name, String acctNum, String acctType, String transType, double transAmount, String date): void
+createNewAcct (String name, String acctNum,String acctType, double interestRate, double balance, String date): void
+executeDeposit (String cusName, String tableName, String acctNumber, double amount, double interestRate, double nowInterestTotal, String date): void
+executeWithdraw (String tableName , String cusName, String acctNumber, double nowInterestTotal, double amount): void
+executeDelete(String sql): void

2. Case Studies









银行系统用户图形界面

Back

2. Case Studies

Case 2: Design of Mobile Robotics System

2. Case Studies

Case study 2: Mobile Robotics

【例2】自动机器人控制系统

- **Mobile Robot Lab, Carnegie Mellon University, Robotics Institute**
- The Mobile Robot Laboratory is engaged in long-term basic research in
  - **Perception (感知, see, feel suroundings)**
  - **control and**
  - **planning**

  for robots that navigate through complex indoor and outdoor spaces.

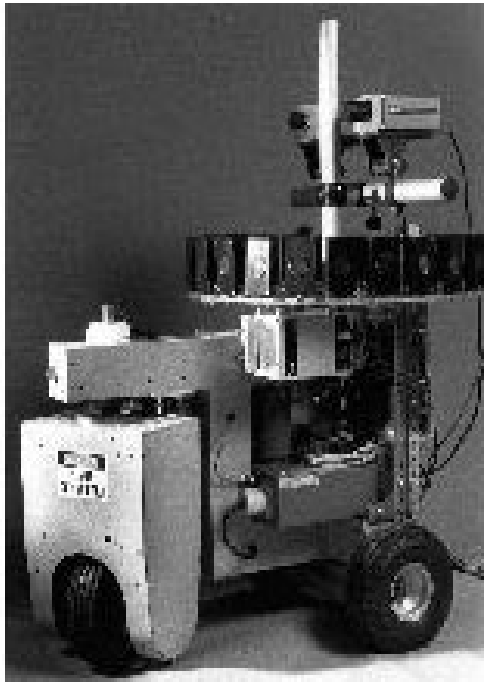
2. Case Studies

- Since 1981, based on work conducted at Stanford University since 1973, the lab has built three different mobile robots
  - Pluto (冥王星)
  - Neptune and (海王星)
  - Uranus (天王星)

  and demonstrated new methods that allow them to navigate cluttered surroundings
  - by stereo and monocular vision (单眼视觉)
  - by broad beam sonar
  - by scanning laser rangefinder and
  - by other sensors

2. Case Studies

Pluto (冥王星)



Neptune (海王星)

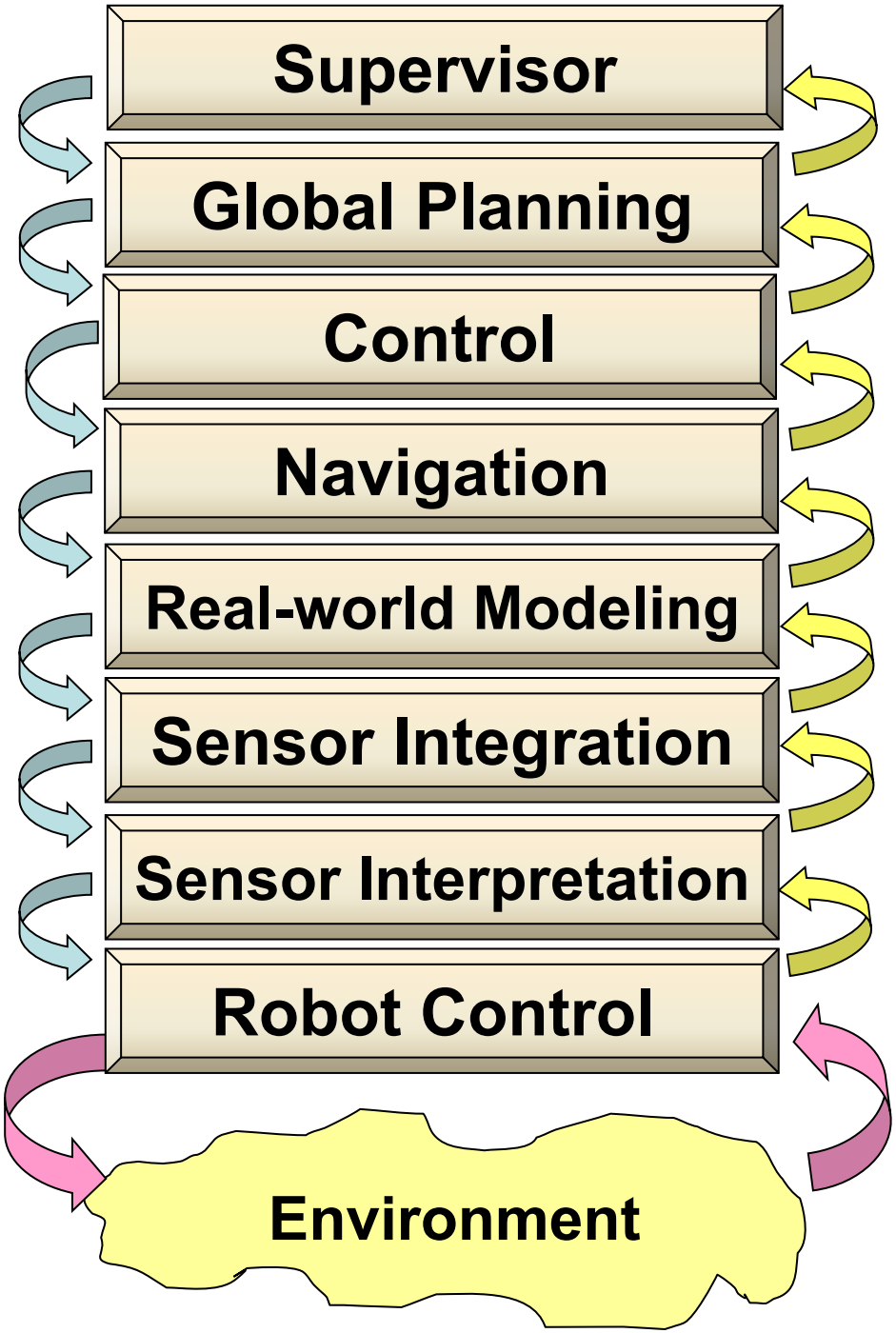
Uranus (天王星)

2. Case Studies

Solution Based on Layered Architecture

- **The following figure** shows Alberto Elfes's definition of the idealized layered architecture, which influenced the design of the Dolphin sonar and navigation system, implemented on
  - **the Terregator and**
  - **Neptune**

  mobile robots.



User level control

Deal with problems, replan
plan the robot's actions

Manage the navigation of the robot

Maintain robot's model of the world.

Analyze different sensor inputs
Comprehensively

Analyze data from one sensor

Control motors, joints,
wheels...

A Layered Solution for Mobile Robots

2. Case Studies

Level 1: the lowest level, reside the robot control routines (motors, joints, etc.).

Levels 2: sensor interpretation, the analysis of the data from one sensor, and

Levels 3: sensor integration, the combined analysis of different sensor inputs

Level 4: is concerned with maintaining the robot's **model** of the **world**.

Level 5: manages the **navigation** of the robot.

Levels 6, 7: schedule and plan the robot's actions. Dealing with problems and replanning is also part of the level-7 responsibilities.

2. Case Studies

- **Problems**:
  - **the layered architecture does not fit actual data and control flow patterns**.
  - **The layers suggest that services and requests are passed between adjacent components. Actually, information passing is less straightforward.**
  - **Example:**
    - ***data requiring fast response may be sent directly from sensor to level 7 to deal with the problem.***
    - ***the corresponding commands may have to skip levels to reach the motors in time.***

2. Case Studies

Summary:

- Abstraction levels defined by the layered architecture
  - provide a framework for organizing components
  - framework succeeds because it is precise about the roles of the different layers
- Major drawbacks
  - model breaks down when taken to the greater level of detail demanded by an actual implementation.
  - Communication patterns in a robot not likely to follow the orderly scheme implied by the architecture.

Back