软件工程专业导论

复习

1. 基本的软件过程包括需求分析、概要设计、详细设计、编码、测试、运行和维护等几个阶段。 其中，_____ 阶段对每个模块要完成的工作进行具体描述，为源程序编写打下基础

- A. 需求分析
- B. 概要设计
- C. 详细设计
- D. 编码

复习

1. 基本的软件过程包括需求分析、概要设计、详细设计、编码、测试、运行和维护等几个阶段。 其中，_____ 阶段对每个模块要完成的工作进行具体描述，为源程序编写打下基础

- [ ] A. 需求分析
- [ ] B. 概要设计
- [√] C. 详细设计
- [ ] D. 编码

复习

2. 在基本软件过程的各个阶段中，_____是指对解决现实世界某个问题的软件产品的描述，及对软件产品约束的描述

- A.详细设计
- B.编码
- C.需求分析
- D.概要设计

复习

2. 在基本软件过程的各个阶段中，_____是指对解决现实世界某个问题的软件产品的描述，及对软件产品约束的描述

- [ ] A.详细设计
- [ ] B.编码
- [√] C.需求分析
- [ ] D.概要设计

复习

3. 原型化方法是用户和设计者之间执行的一种交互构成，适用于_____系统

- A.管理信息
- B.需求确定的
- C.实时
- D.需求不确定性高的

复习

3. 原型化方法是用户和设计者之间执行的一种交互构成，适用于_____系统

- [ ] A.管理信息
- [ ] B.需求确定的
- [ ] C.实时
- [√] D.需求不确定性高的

复习

4. 基本的软件过程包括需求分析、概要设计、详细设计、编码、测试、运行和维护等几个阶段。其中，_____阶段的任务是如何改正软件运行过程中发现的缺陷、如何提高软件性能或其他属性、如何使软件产品适应新的环境

- A.软件运行与维护
- B.测试
- C.编码
- D.设计

复习

4. 基本的软件过程包括需求分析、概要设计、详细设计、编码、测试、运行和维护等几个阶段。其中，_____阶段的任务是如何改正软件运行过程中发现的缺陷、如何提高软件性能或其他属性、如何使软件产品适应新的环境

- [√] A.软件运行与维护
- [ ] B.测试
- [ ] C.编码
- [ ] D.设计

复习

5. “软件需求”被定义为软件用于解决真实世界问题而必须展示的特性，指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。以下关于需求的描述中，不正确的是 _____

- A.需求是多样化的
- B.需求是可以量化的
- C.需求是固定的
- D.需求是可验证的

复习

5. “软件需求”被定义为软件用于解决真实世界问题而必须展示的特性，指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。以下关于需求的描述中，不正确的是 _____

- [ ] A.需求是多样化的
- [ ] B.需求是可以量化的
- [√] C.需求是固定的
- [ ] D.需求是可验证的

复习

6. 软件需求是软件生命周期的第一个阶段。成功地开发软件产品，首先需要深入理解待用软件解决的问题----软件需求。需求分析中，开发人员要从用户那里解决的最重要的问题是_____

- A.要给软件提供哪些信息
- B.让软件做什么
- C.要求软件工作效率怎样
- D.让软件具有何种结构

复习

6. 软件需求是软件生命周期的第一个阶段。成功地开发软件产品，首先需要深入理解待用软件解决的问题----软件需求。需求分析中，开发人员要从用户那里解决的最重要的问题是_____

- [ ] A.要给软件提供哪些信息
- [√] B.让软件做什么
- [ ] C.要求软件工作效率怎样
- [ ] D.让软件具有何种结构

复习

7. 需求是使用简单、高阶和抽象的文字叙述来描述使用者需要的系统服务和操作限制，或正式定义系统详细功能的规格书。需求规格说明书的内容不应包括对_____的描述

- A.主要功能
- B.算法的详细过程
- C.用户界面和运行环境
- D.软件性能

复习

7. 需求是使用简单、高阶和抽象的文字叙述来描述使用者需要的系统服务和操作限制，或正式定义系统详细功能的规格书。需求规格说明书的内容不应包括对_____的描述

- [ ] A.主要功能
- [√] B.算法的详细过程
- [ ] C.用户界面和运行环境
- [ ] D.软件性能

软件设计

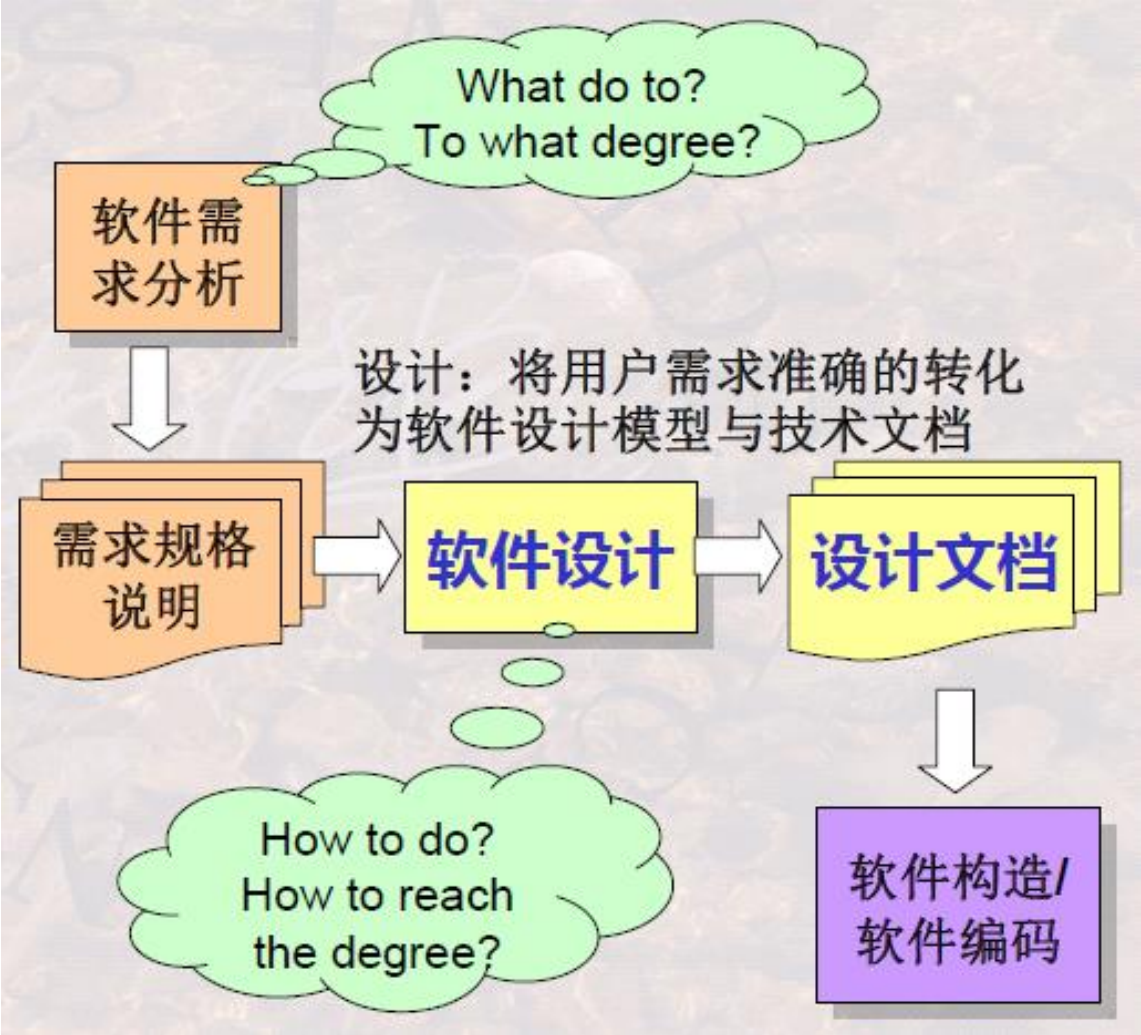
学习目标：

- 什么是软件设计
- 软件体系结构
- 设计方法
- 统一建模语言UML

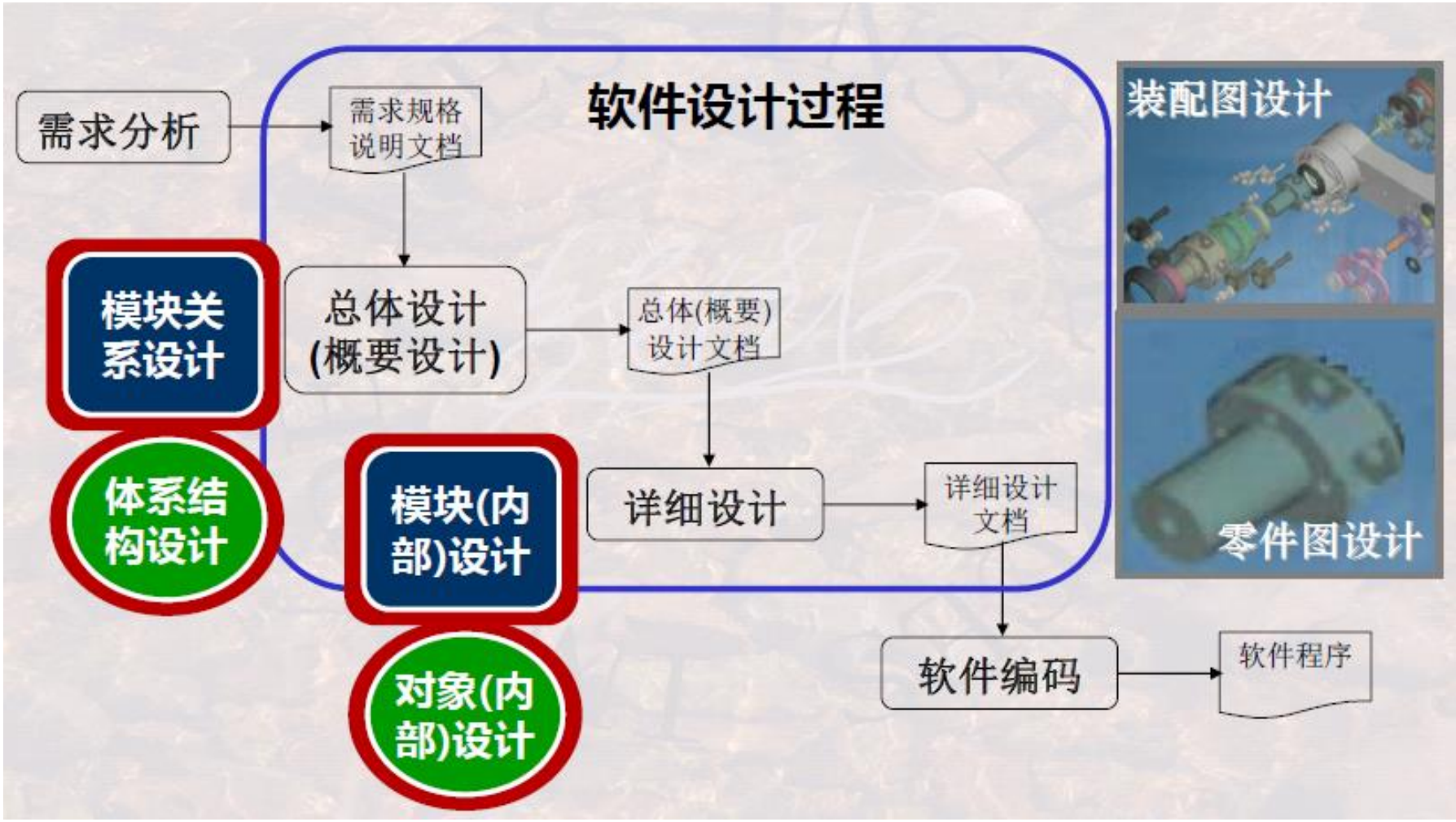
1. 什么是软件设计

- 软件设计是从软件需求规格说明书出发，根据需求分析阶段确定的功能设计软件系统的整体结构、划分功能模块、确定每个模块的实现算法以及编写具体的代码，形成软件的具体设计方案。---- 百度百科
- Software design usually involves problem solving and planning a software solution. This includes both a low-level component and algorithm design and a high-level, architecture design. ---- Wikipedia

1. 什么是软件设计

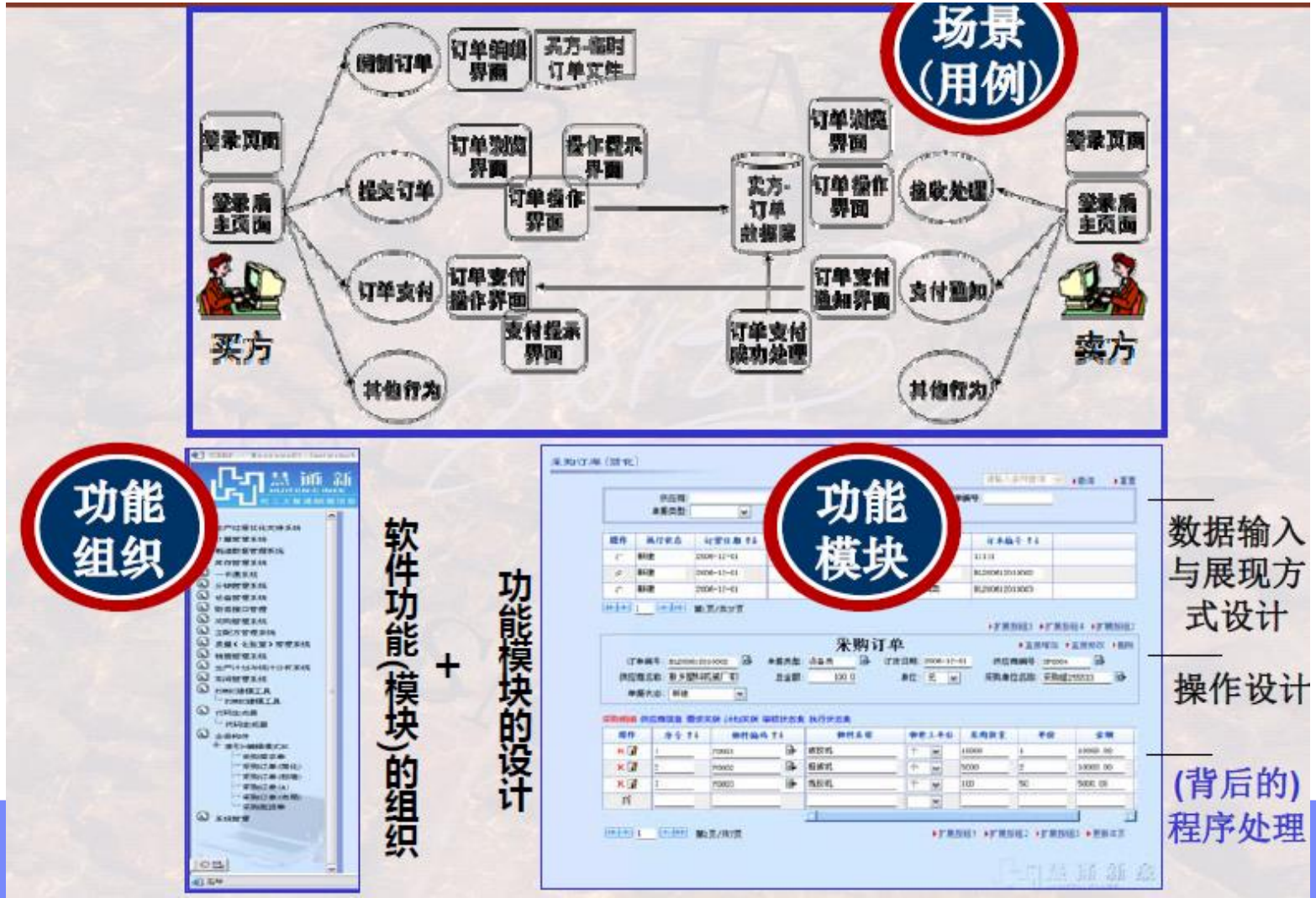

What do to?
To what degree?
软件需求分析
设计：将用户需求准确的转化为软件设计模型与技术文档
需求规格说明
软件设计
设计文档
How to do?
How to reach the degree?
软件构造/软件编码

1. 什么是软件设计


需求分析
需求规格说明文档
软件设计过程
模块关系设计
总体设计(概要设计)
总体(概要)设计文档
体系结构设计
模块(内部)设计
详细设计
详细设计文档
对象(内部)设计
软件编码
软件程序
装配图设计
零件图设计

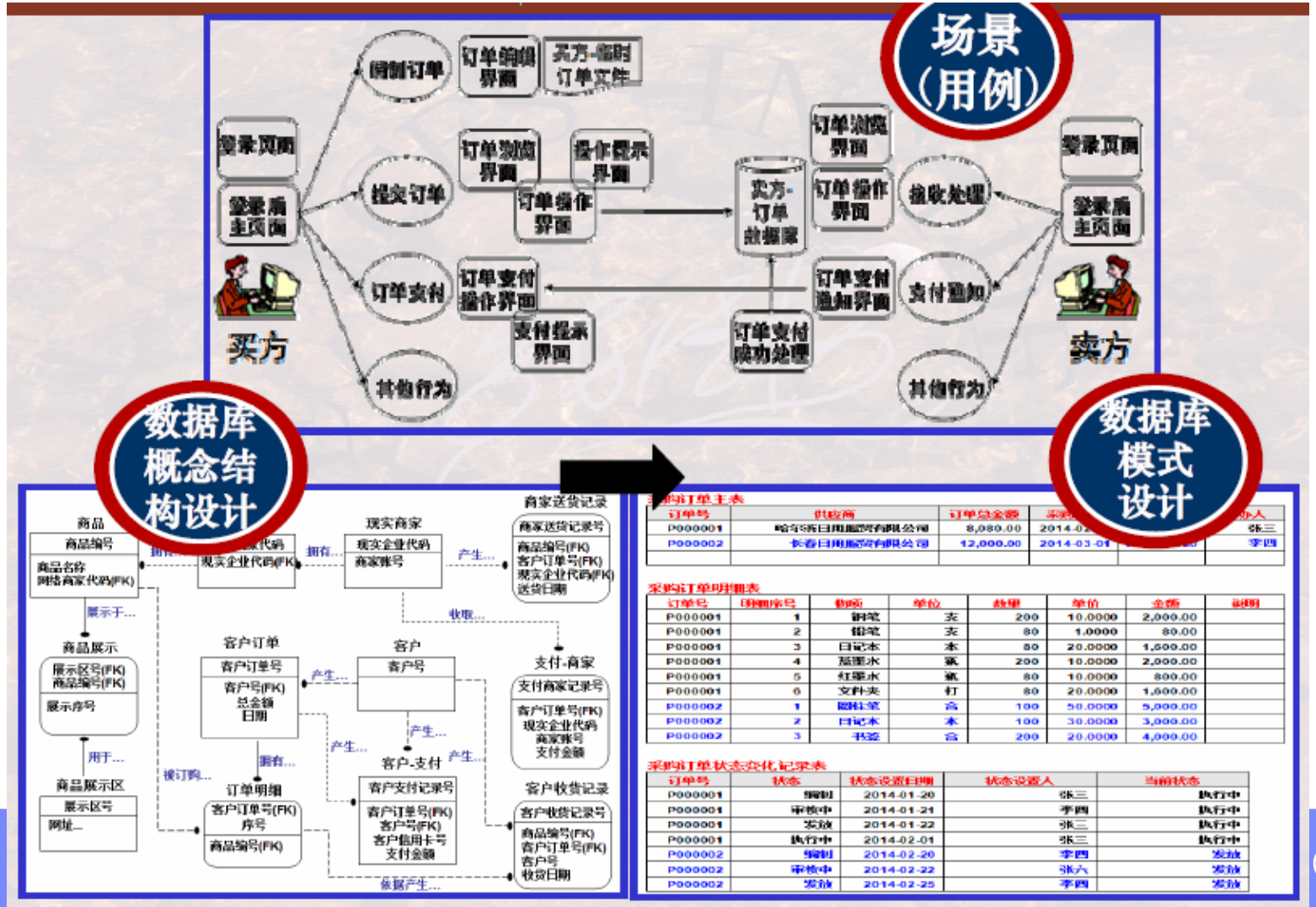
2. 软件设计的范畴

- 从使用者角度，软件 ≈ 功能组织 + 功能模块



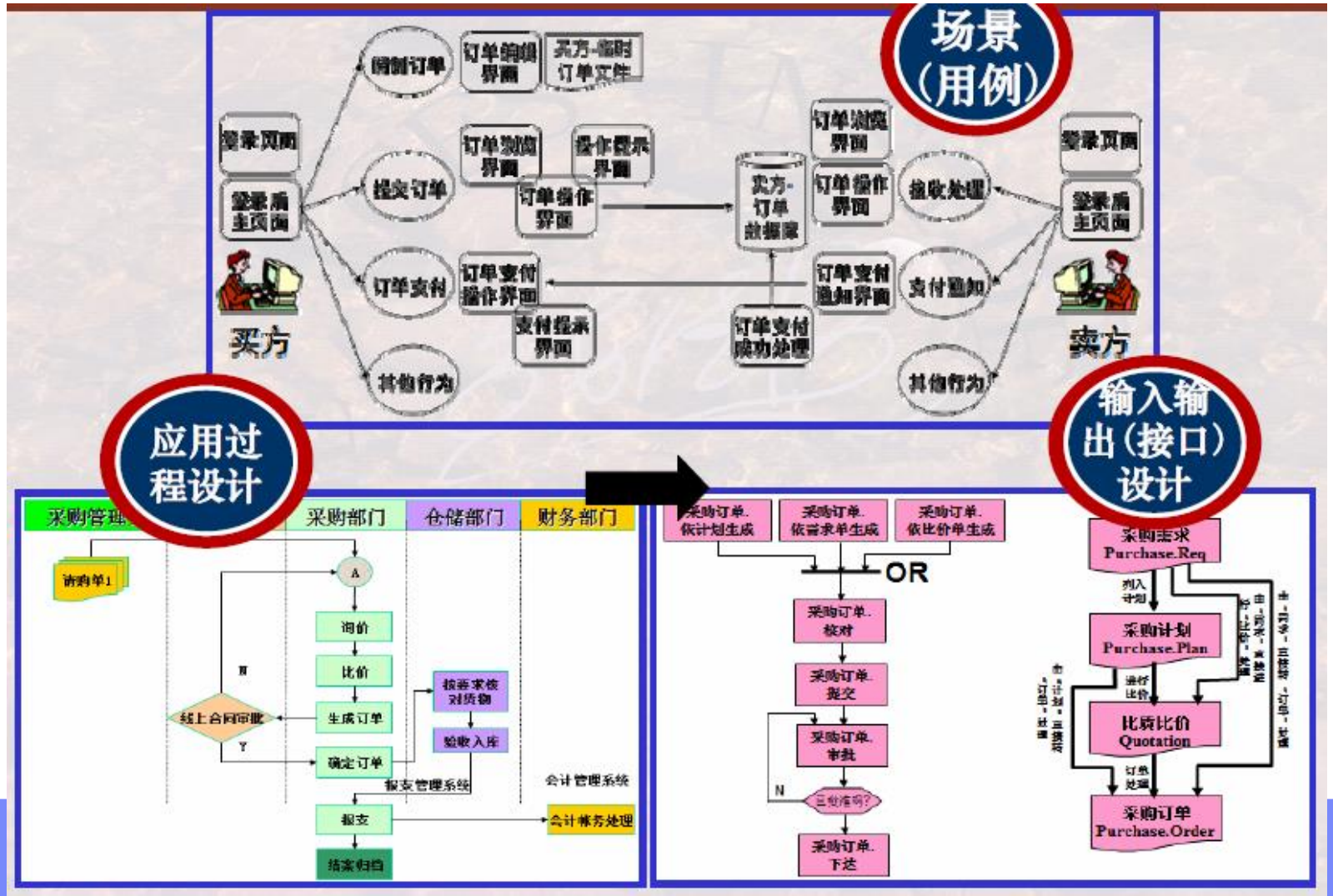
2. 软件设计的范畴

- 从设计者角度



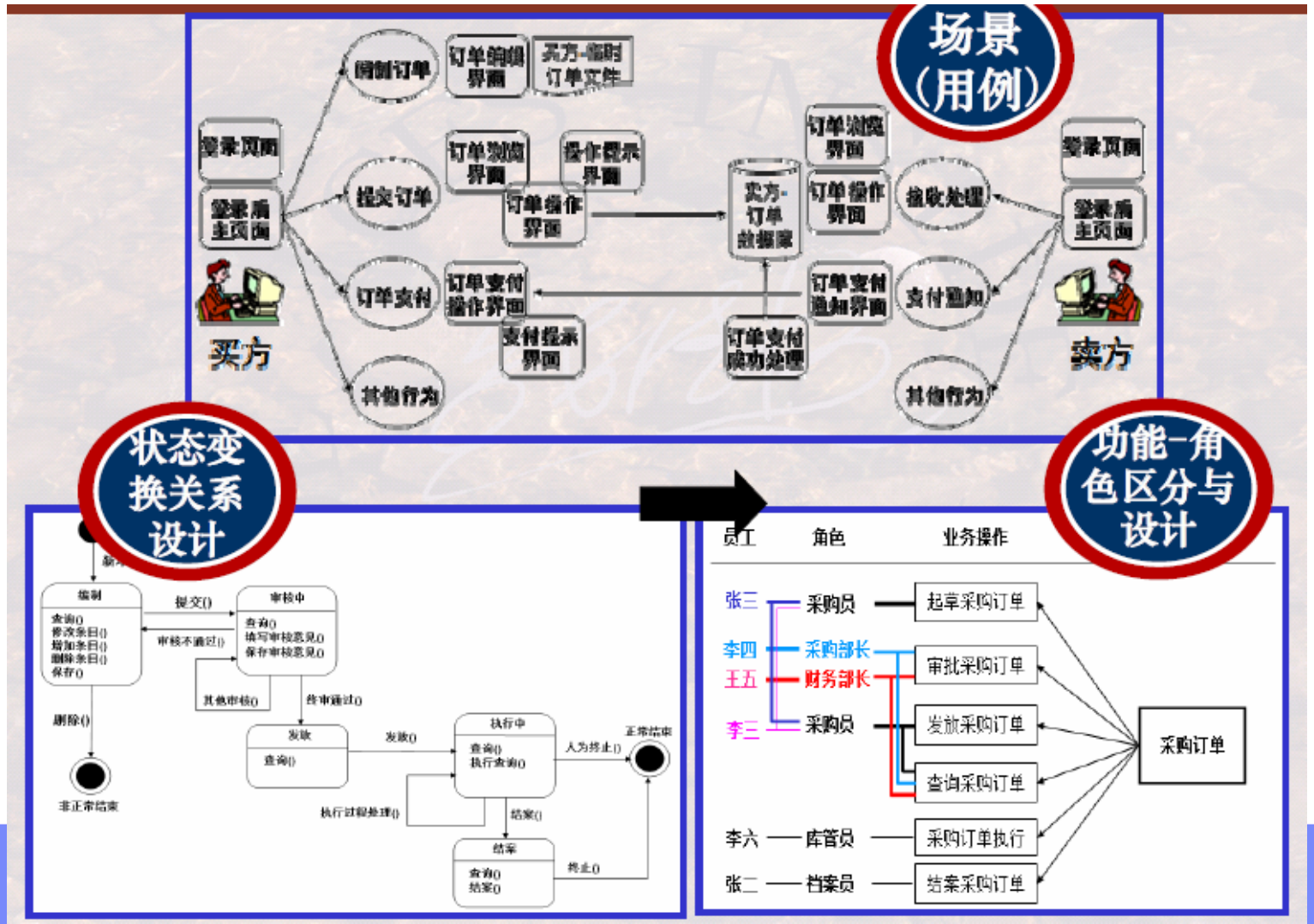
2. 软件设计的范畴

- 从设计者角度



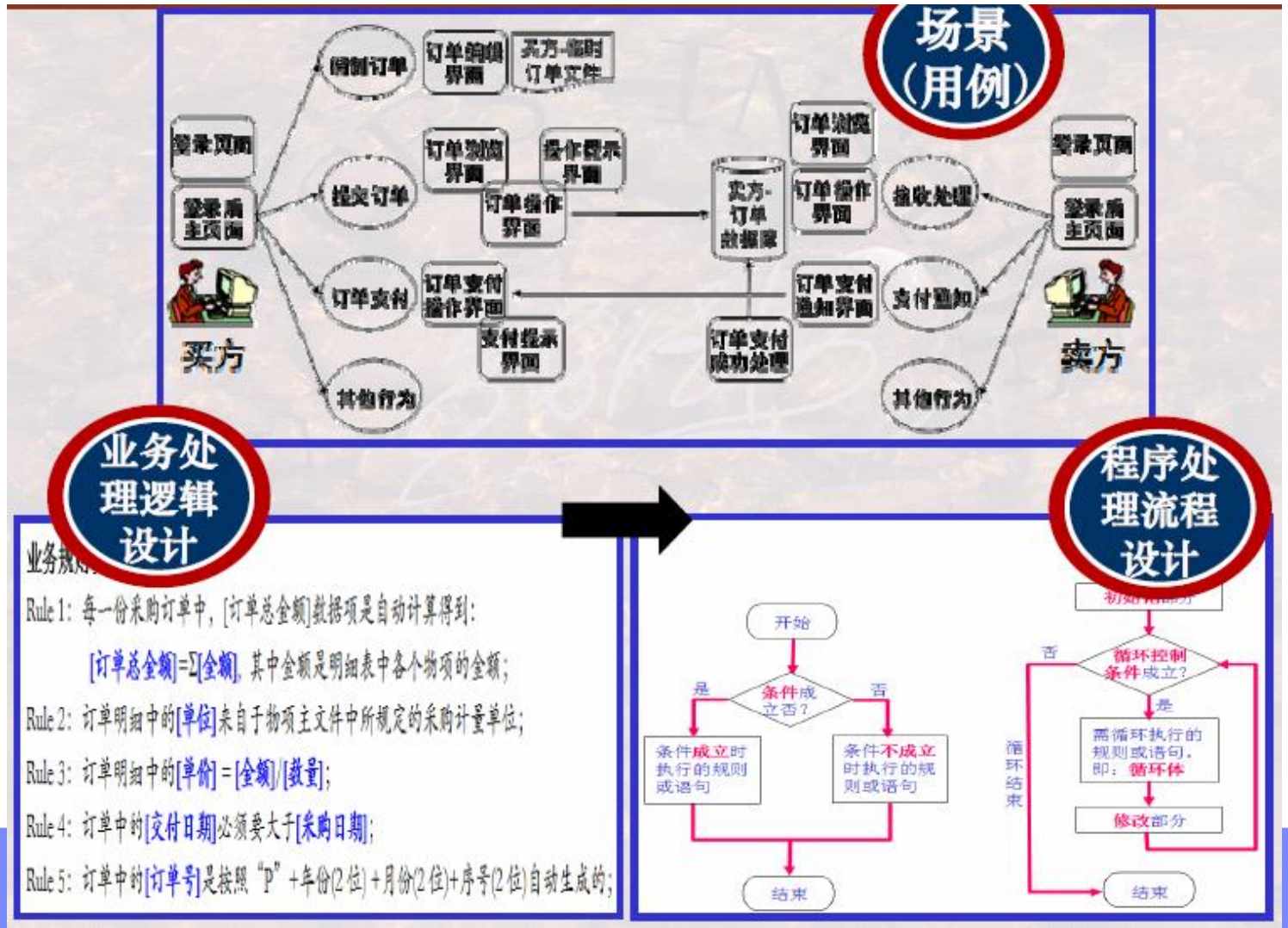
2. 软件设计的范畴

- 从设计者角度



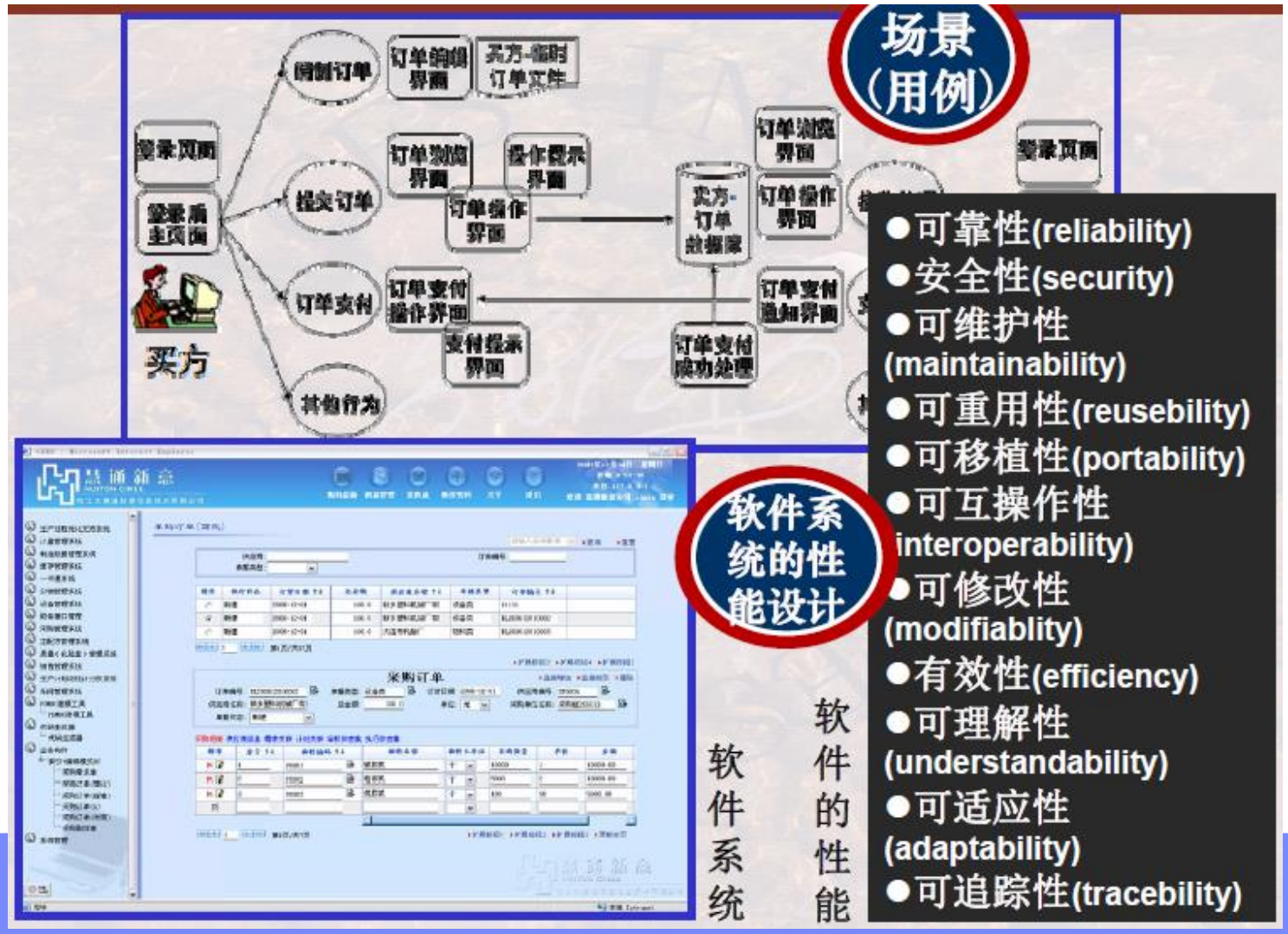
2. 软件设计的范畴

- 从设计者角度



2. 软件设计的范畴

- 从设计者角度



3. 软件体系结构（Software Architecture）

- 什么是软件体系结构（SWA）？

- Perry and Wolf
- SWA = {元素(elements) ----- what
  , 形式(form) ----- how
  , 合理性(rationale) ----- why

- Shaw and Garland
- SWA包括：
  - 描述系统构建的元素
  - 元素之间的交互
  - 知道系统组成的模式（pattern）
  - 这些模式的约束

3. 软件体系结构（Software Architecture）

- 什么是软件体系结构（SWA）？
  - 通用定义：一组系统设计决策（design decision）的准则

- 软件系统的蓝图（blueprint）
  - 结构
  - 行为
  - 交互
  - 非功能特性

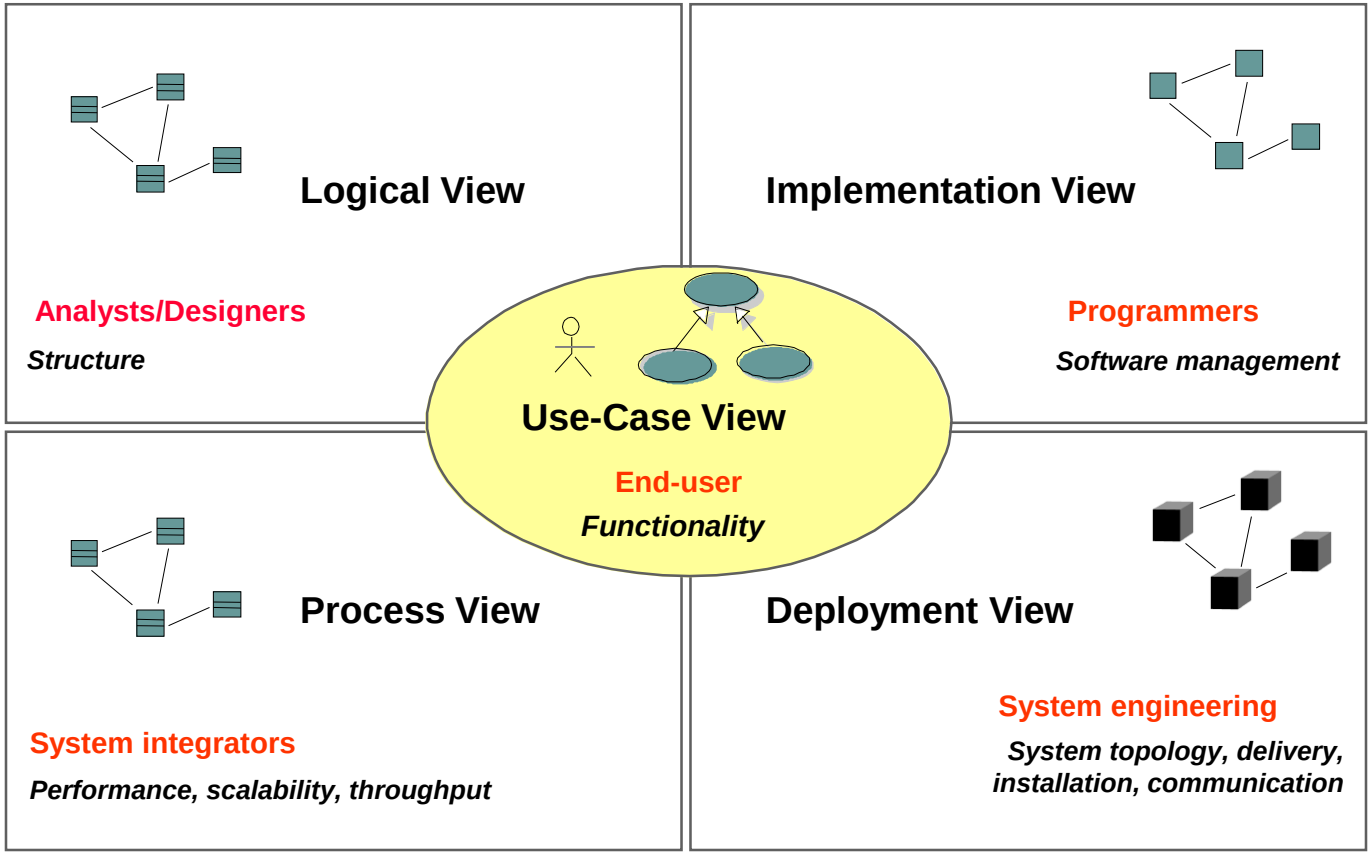
3. 软件体系结构（Software Architecture）

- 什么是软件体系结构（SWA）？
  - 通用定义：一组系统设计决策（design decision）的准则

- SWA是迭代的过程
- 在系统生命周期内，设计决策会不断改变

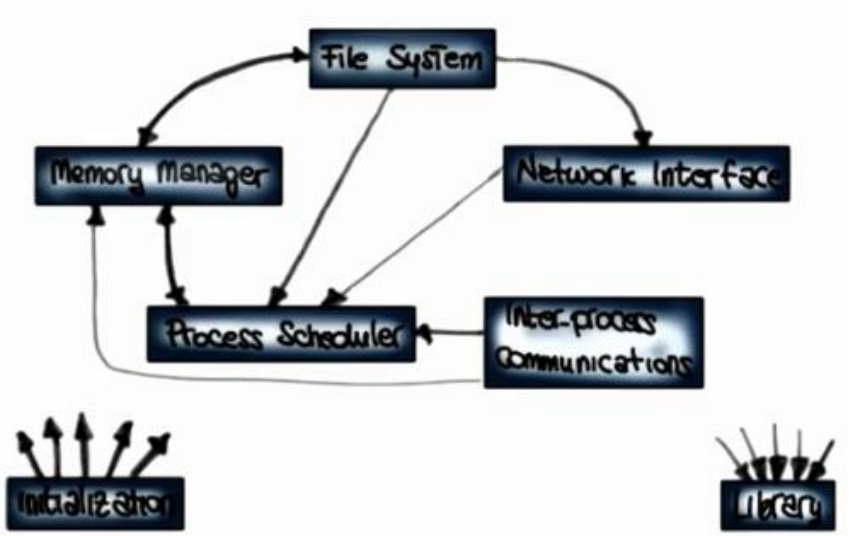
3. 软件体系结构（Software Architecture）

- “4+1视图”模型

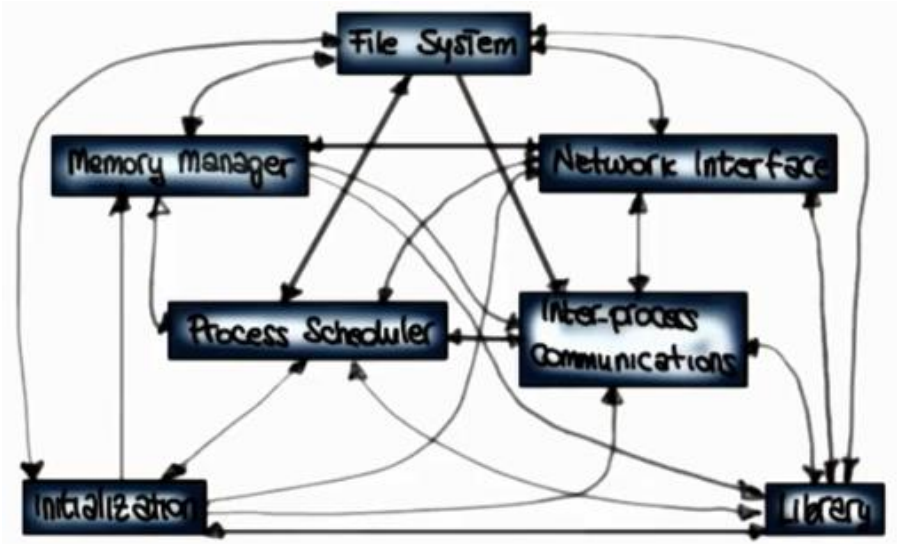


3. 软件体系结构（Software Architecture）

- 现实案例：早期 Linux 内核



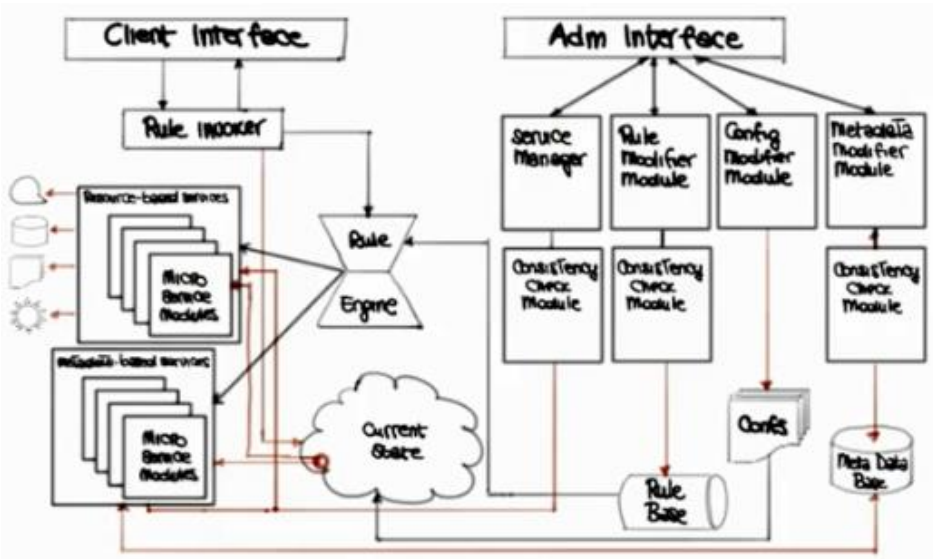
规定的架构（prescriptive architecture）



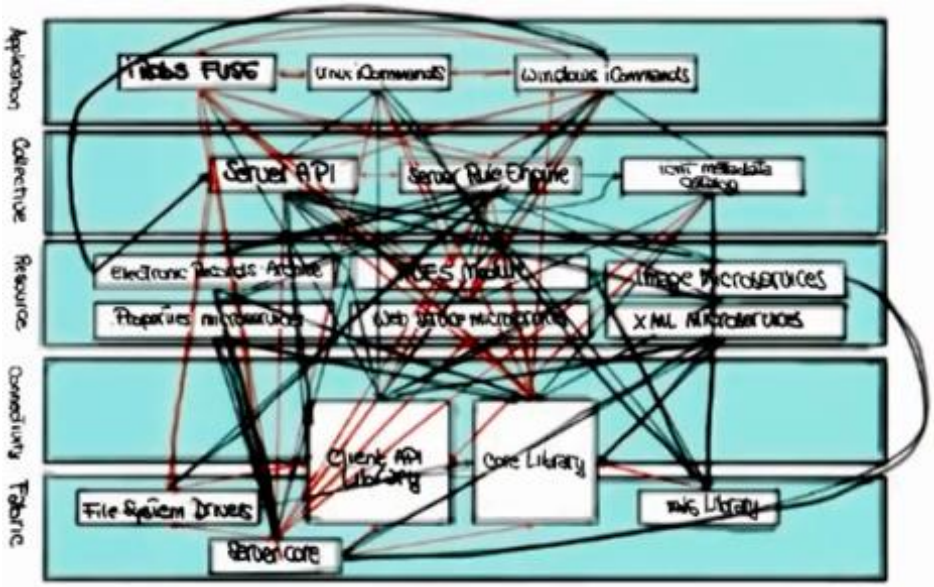
实际完成后的架构描述
（descriptive architecture）

3. 软件体系结构（Software Architecture）

- 现实案例：iRODS



prescriptive architecture



descriptive architecture

3. 软件体系结构（Software Architecture）

- 哪些是架构设计的理想特性？

- ☐ 可扩展性（scalability）
- ☐ 低内聚（low cohesion）
- ☐ 低耦合（low coupling）

3. 软件体系结构（Software Architecture）

- 哪些是架构设计的理想特性？

- [√] 可扩展性（scalability）
- [ ] 低内聚（low cohesion）
- [√] 低耦合（low coupling）

3. 软件体系结构（Software Architecture）

- 软件体系结构的元素（elements）

- 处理元素：实现商业逻辑和数据变换

- 数据元素（信息、状态）

- 交互元素：将架构的不同部分组装到一起

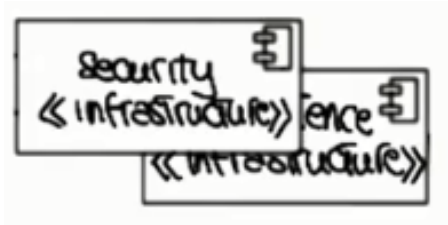
- 组件（components）

- 由系统连接器（connector）维护和控制

配置
(configuration)

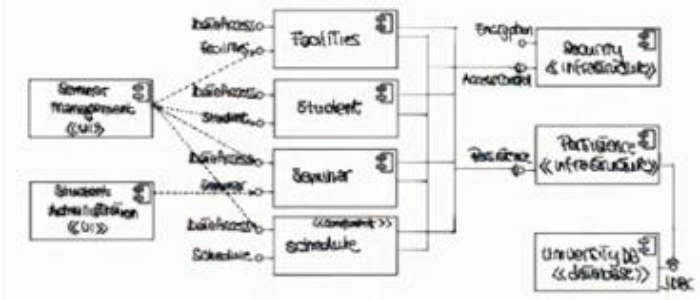
3. 软件体系结构（Software Architecture）

- 组件、连接器和配置



- 组件封装了系统一组功能和/或数据
- 通过接口限制对组件功能的访问

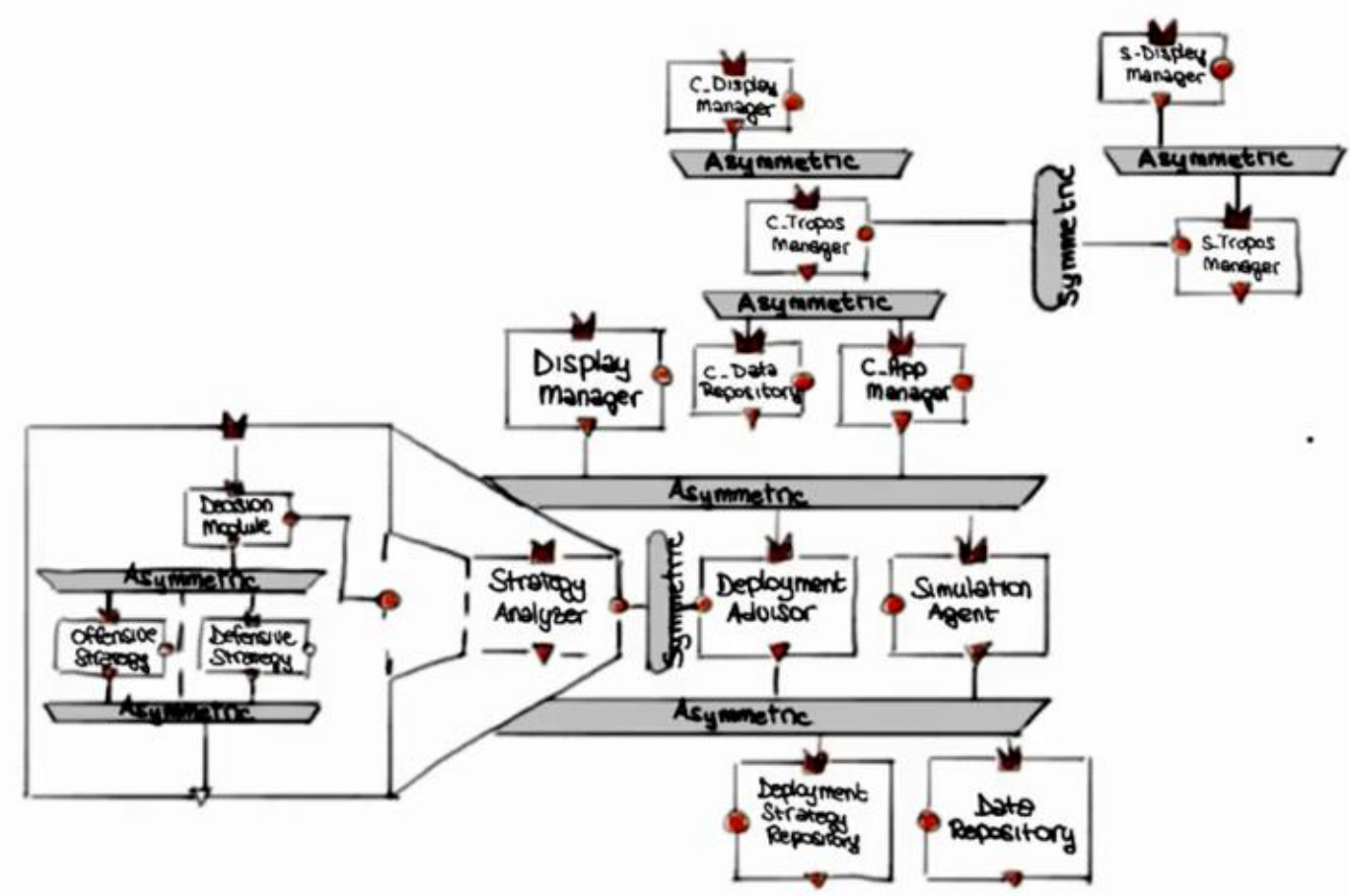
- 连接器管理和控制组件之间的交互



- 配置是组件和连接器之间的关联和组成

3. 软件体系结构（Software Architecture）

- 配置的例子



3. 软件体系结构（Software Architecture）

- 架构的风格（Architecture Styles）

- 管道和过滤器（pipes and filters）

- 事件驱动（event-driven）：GUI

- 订阅和发布（publish-subscribe）

3. 软件体系结构（Software Architecture）

- 架构的风格（Architecture Styles）

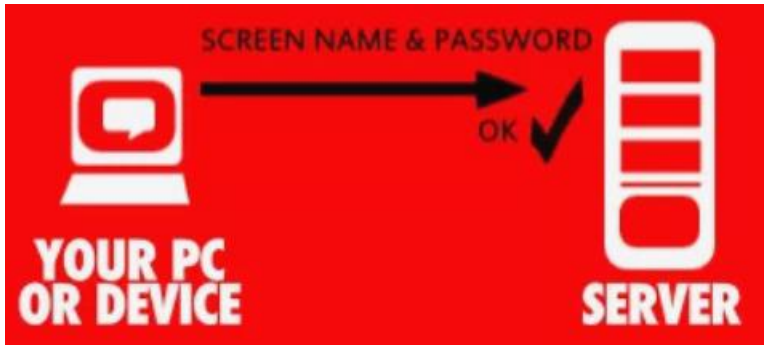
- 客户-服务器（Client-Server）

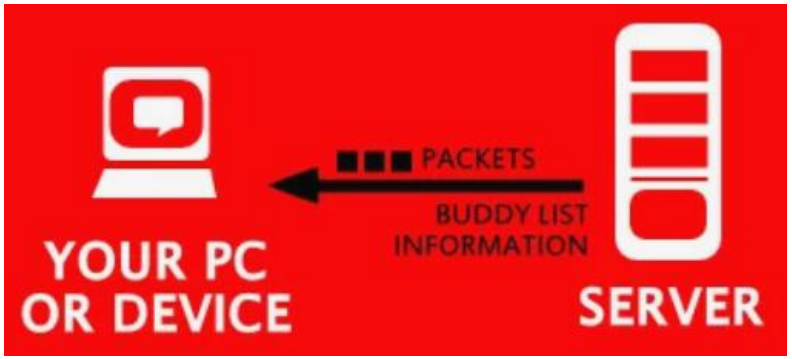
- 对等（Peer-to-Peer）

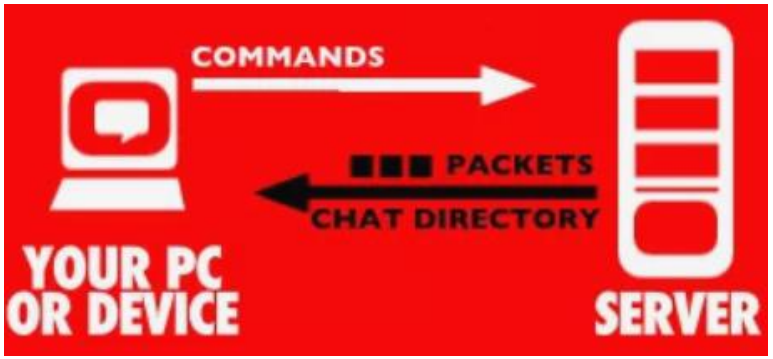
- REST(Representational State Transfer)

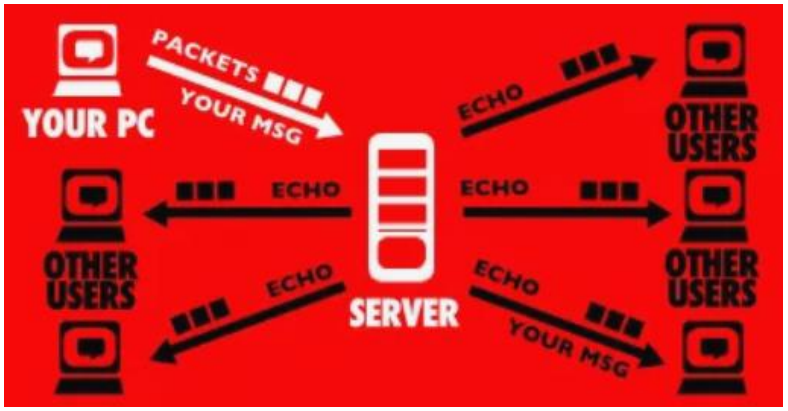
3. 软件体系结构（Software Architecture）

- 客户-服务器（Client-Server）：聊天软件







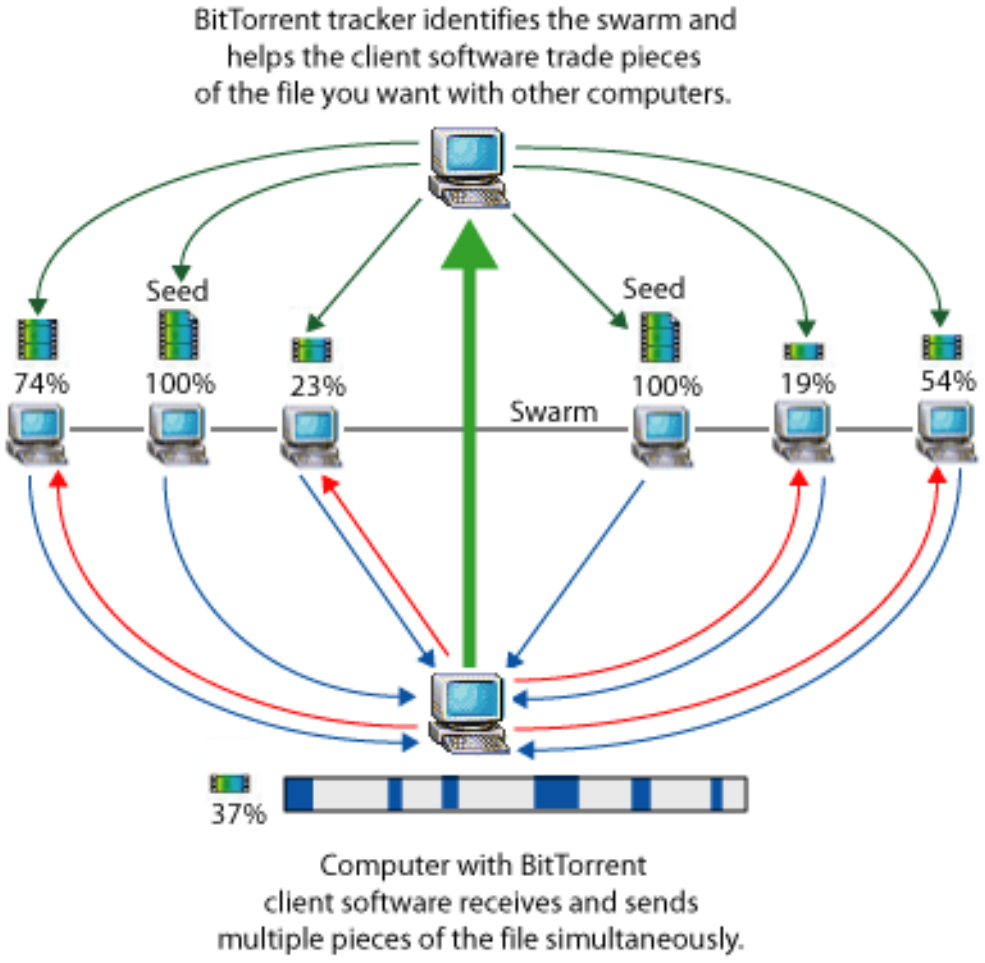


3. 软件体系结构（Software Architecture）

- 对等（Peer-to-Peer）：BitTorrent（BT）

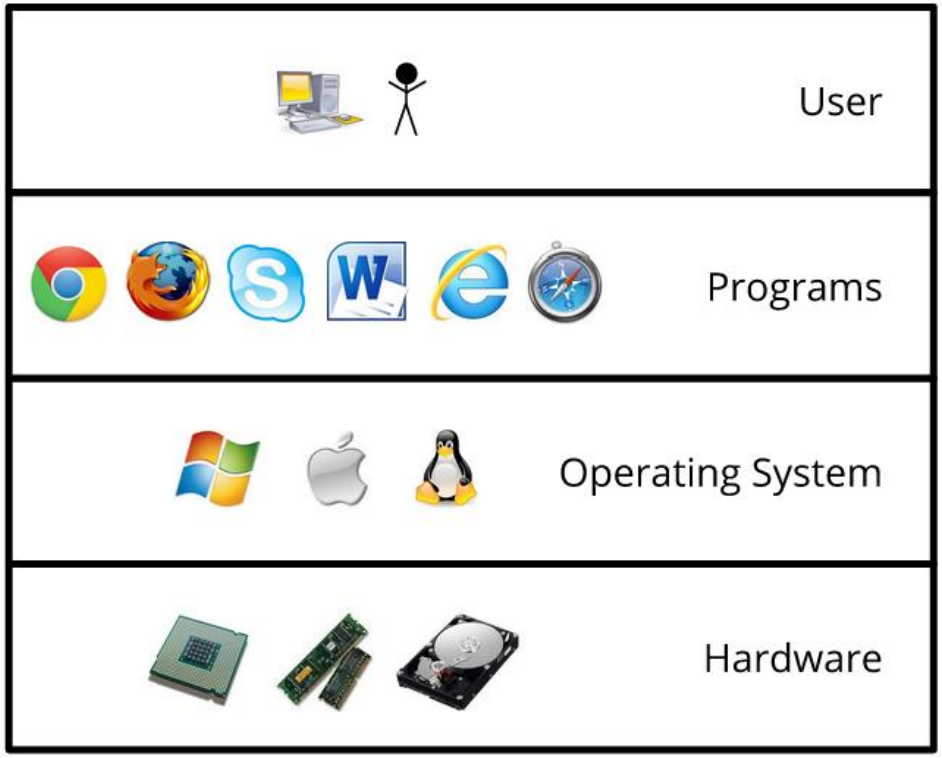
http://mg8.org/processing/bt.html

https://computer.howstuffworks.com/bittorrent.htm

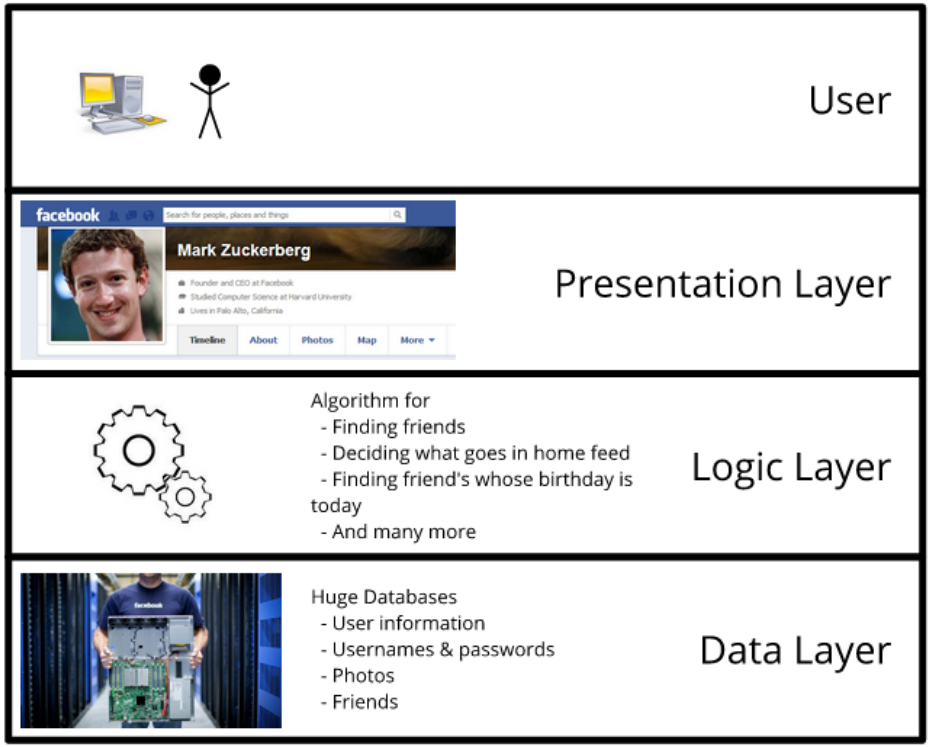


3. 软件体系结构（Software Architecture）

- 层次结构（Layered Architecture）



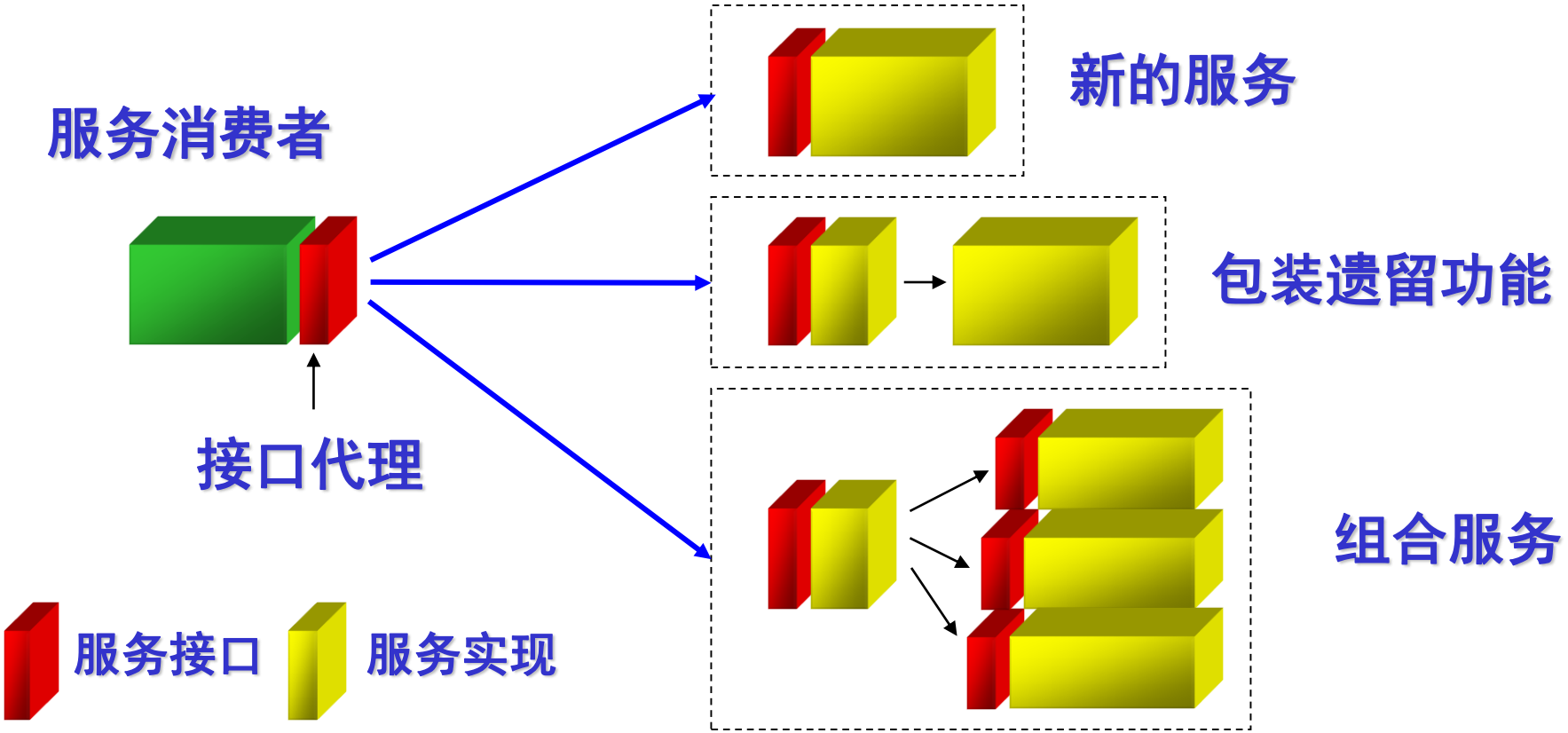
Your computer as a layered system



Facebook as a three-tier system

3. 软件体系结构（Software Architecture）

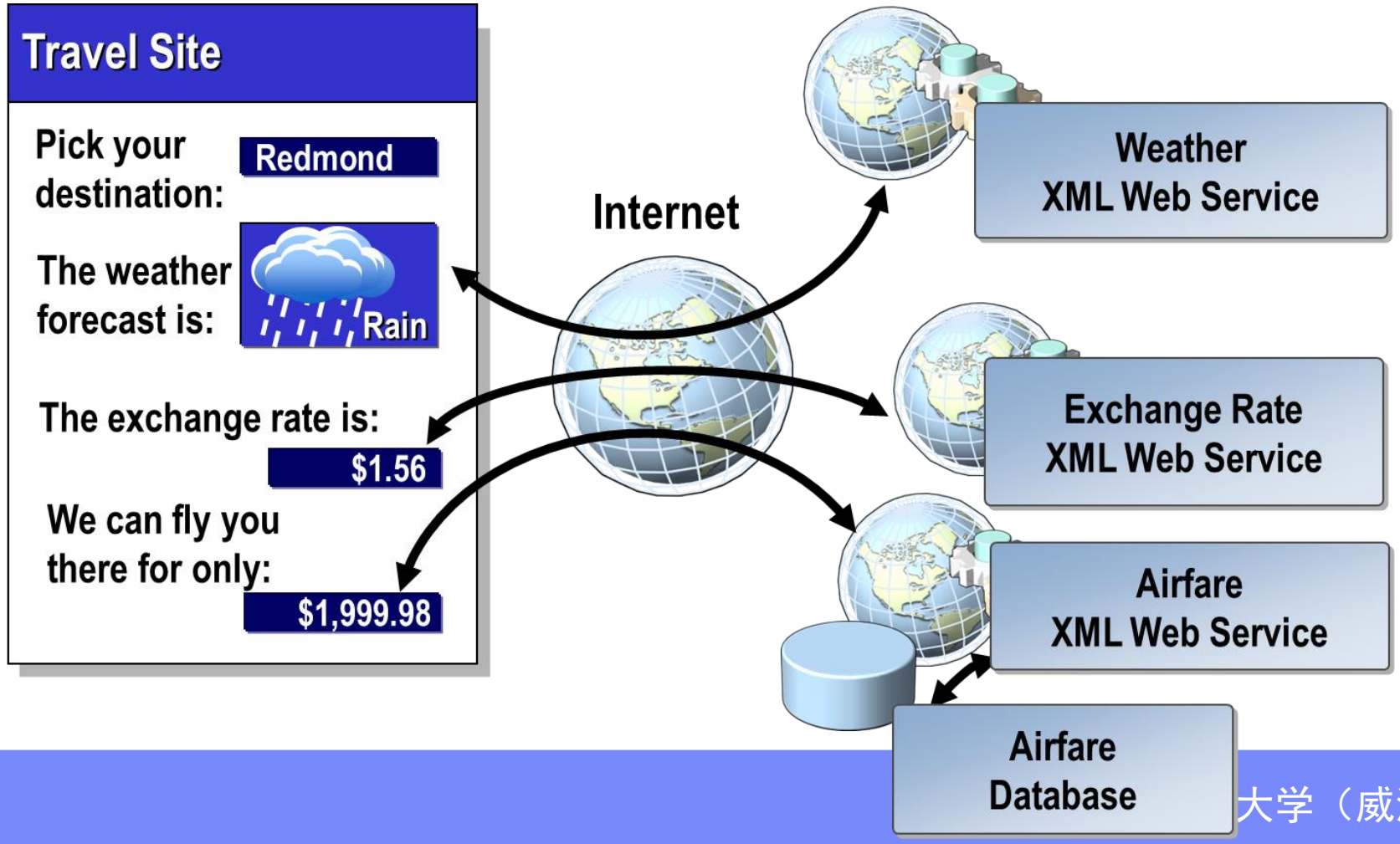
- 架构的风格（Architecture Styles）



基于服务的架构（Service Oriented Architecture - SOA）

3. 软件体系结构（Software Architecture）

- 基于服务的架构（Service Oriented Architecture - SOA）



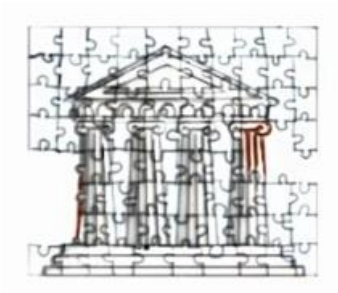
3. 软件体系结构（Software Architecture）

- 基于服务的架构（Service Oriented Architecture - SOA）

Web Service 示例

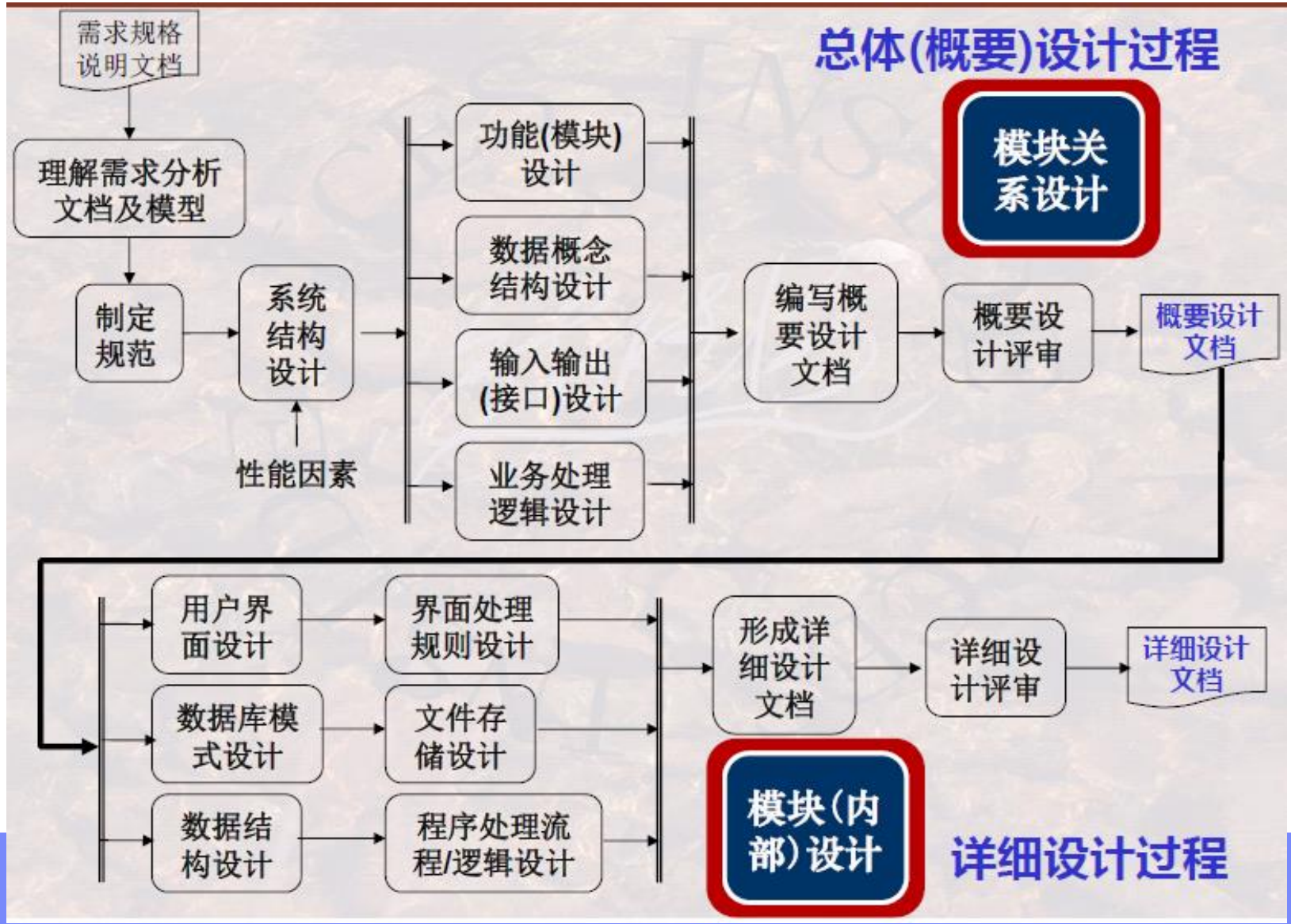
http://blog.csdn.net/qq_20545159/article/details/47903513

3. 软件体系结构（Software Architecture）

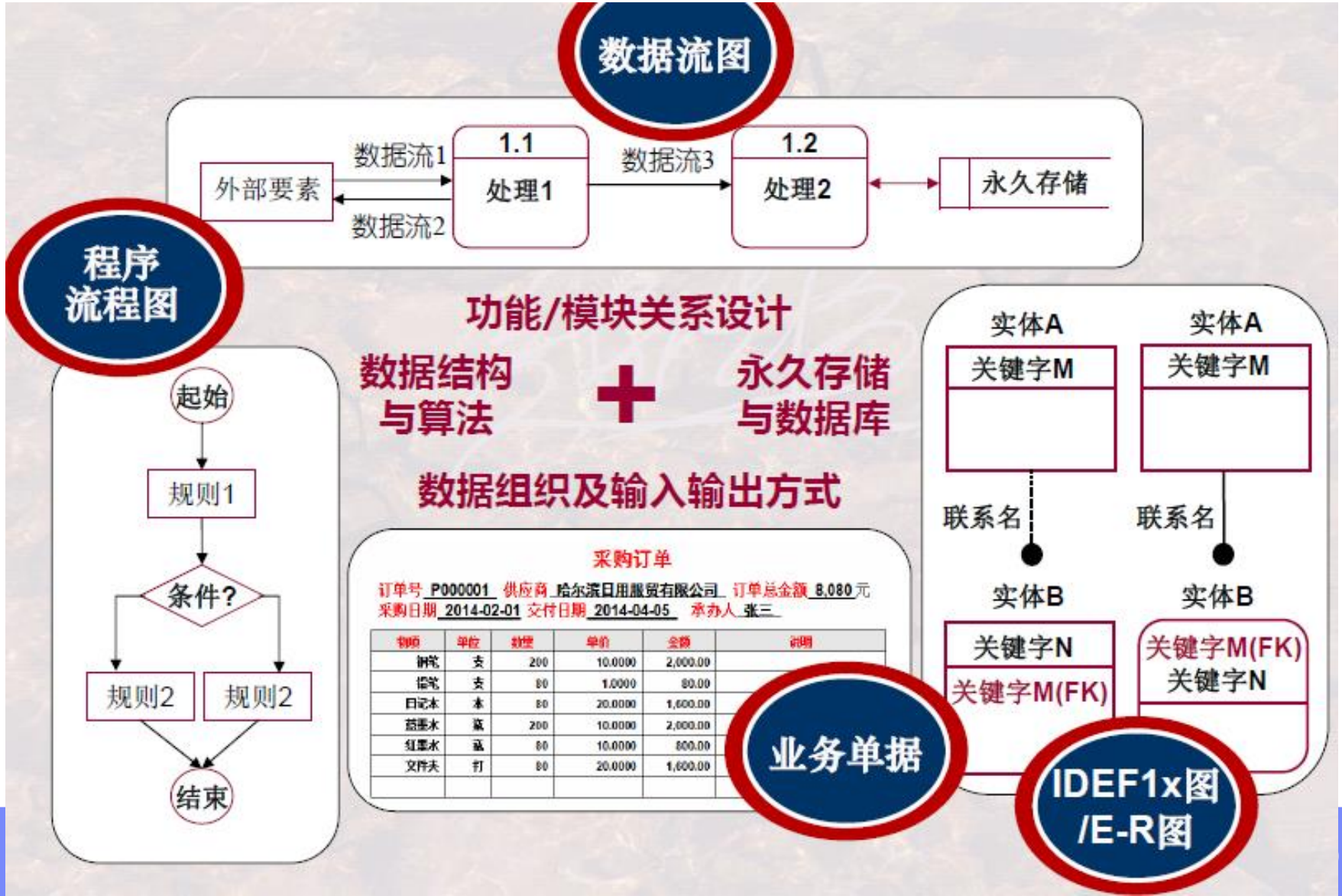


- 好的架构是成功的基础
- 好的架构反映了对问题域的深入理解
- 好的架构通常是简单架构的组合

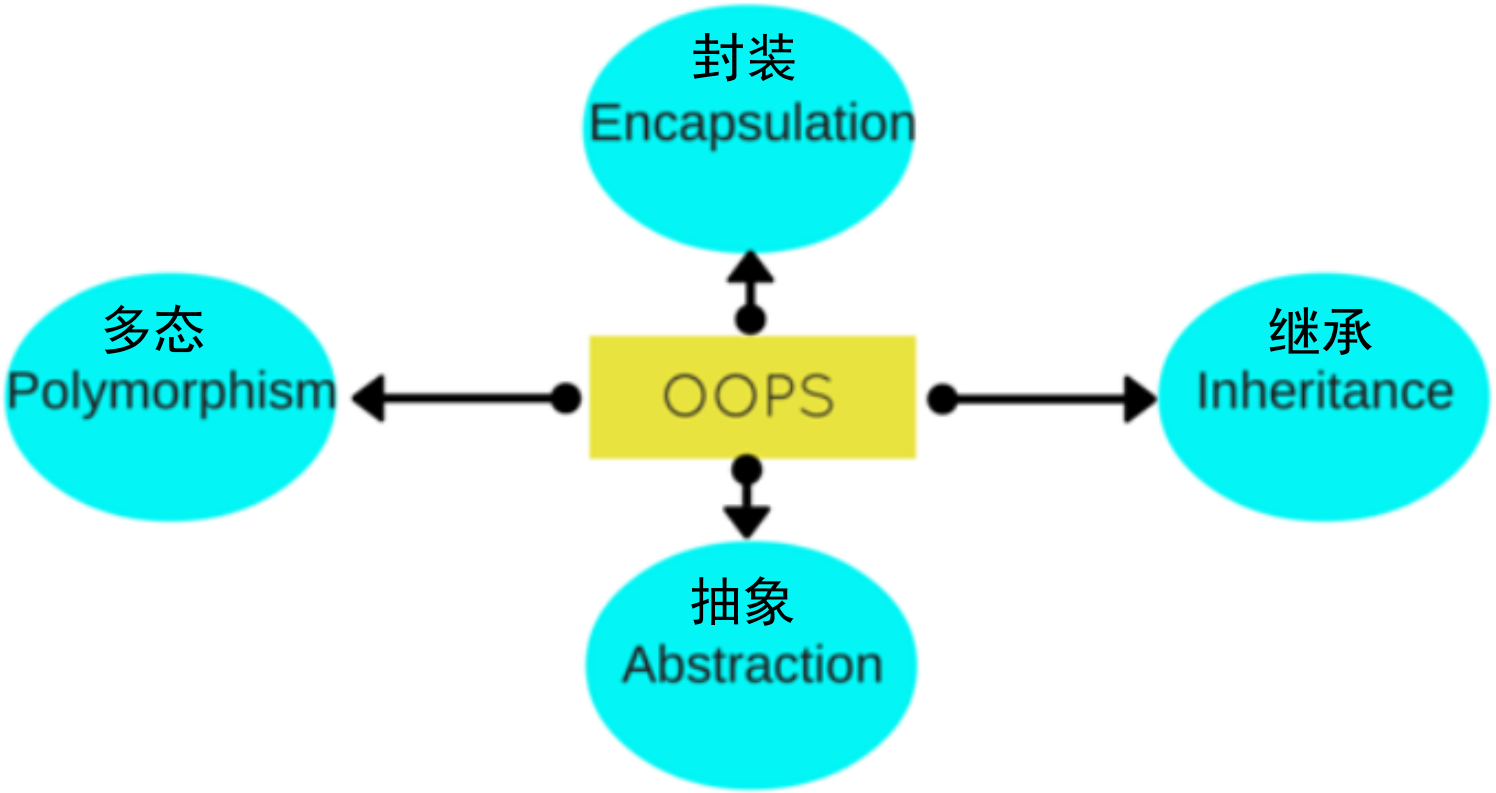
4. 面向功能/过程的结构化软件设计过程



4. 面向功能/过程的结构化软件设计过程



5. 面向对象的分析与设计（OOAD）

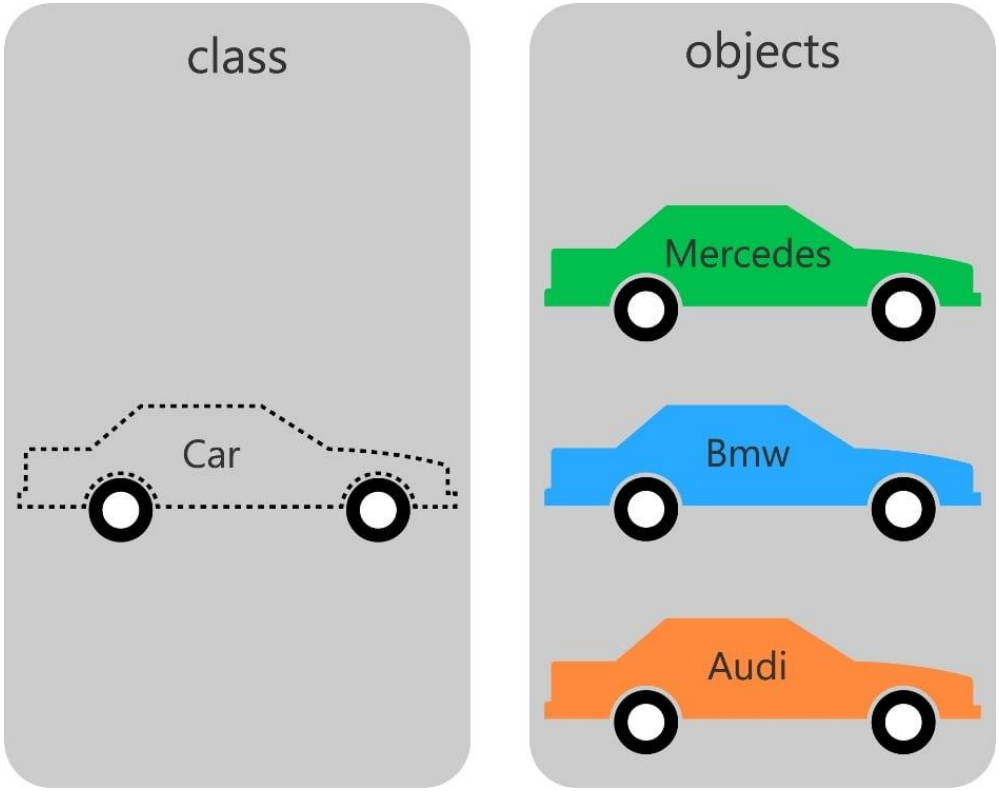


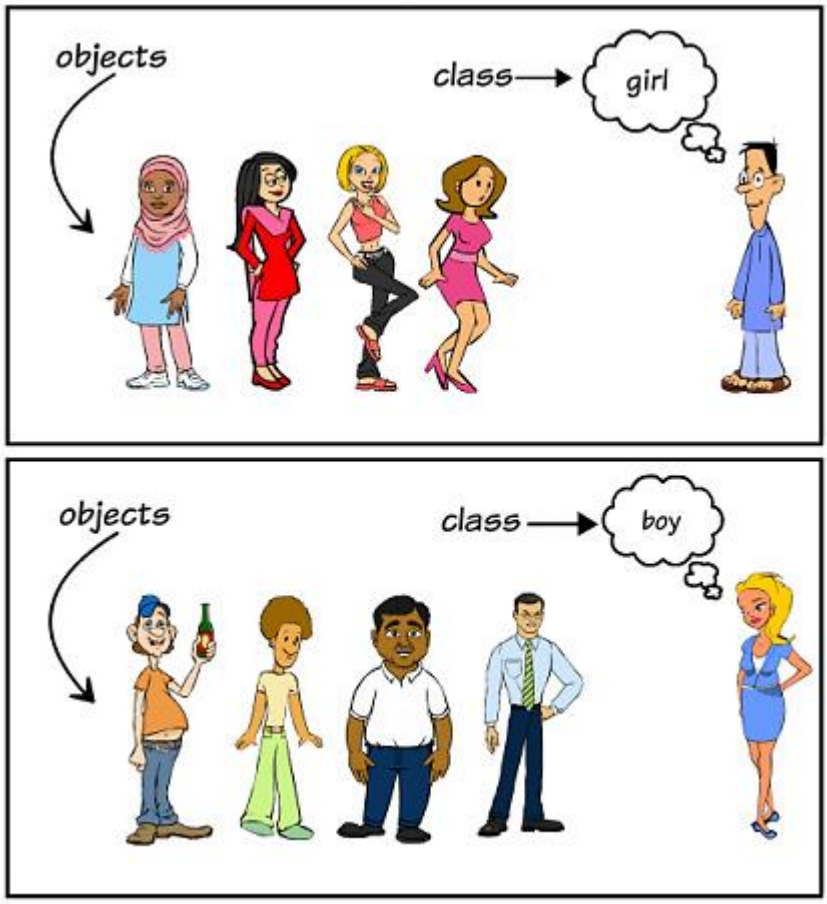
5. 面向对象的分析与设计（OOAD）

- 什么是对象（Object）：实体

5. 面向对象的分析与设计（OOAD）

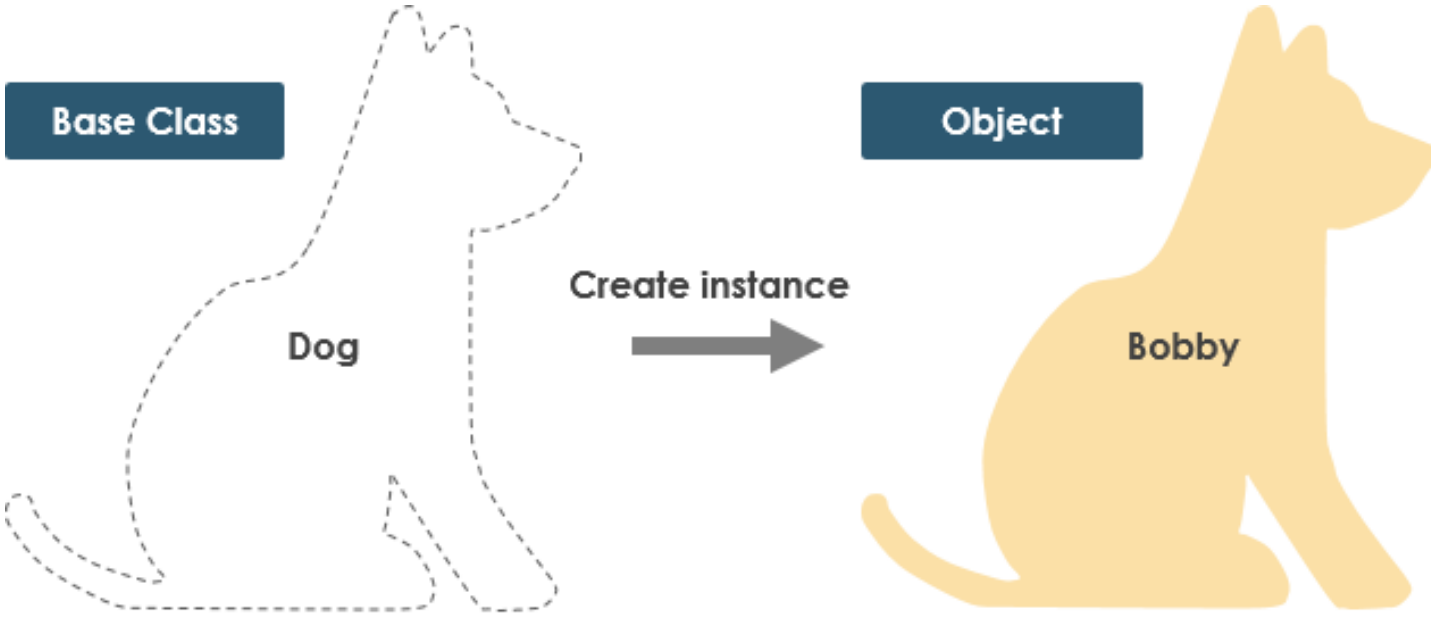
- 什么是类（Class）：对象的集合





5. 面向对象的分析与设计（OOAD）

- 类 vs. 对象

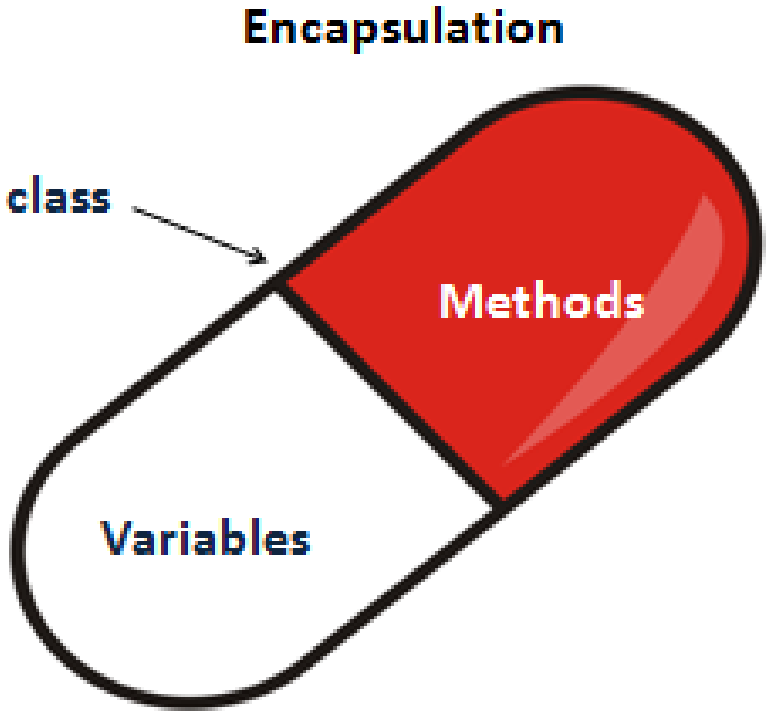


Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

5. 面向对象的分析与设计（OOAD）

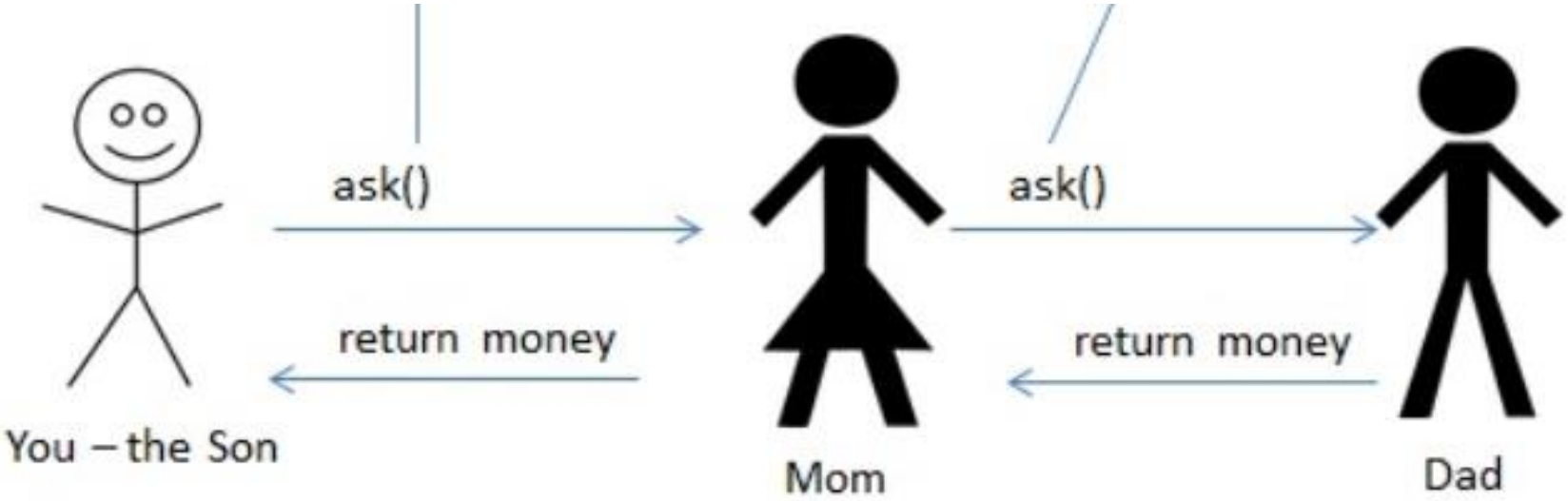
- 封装（Encapsulation）



BankAccount
owner : String balance : Dollars
deposit (amount : Dollars) withdrawal (amount : Dollars)

5. 面向对象的分析与设计（OOAD）

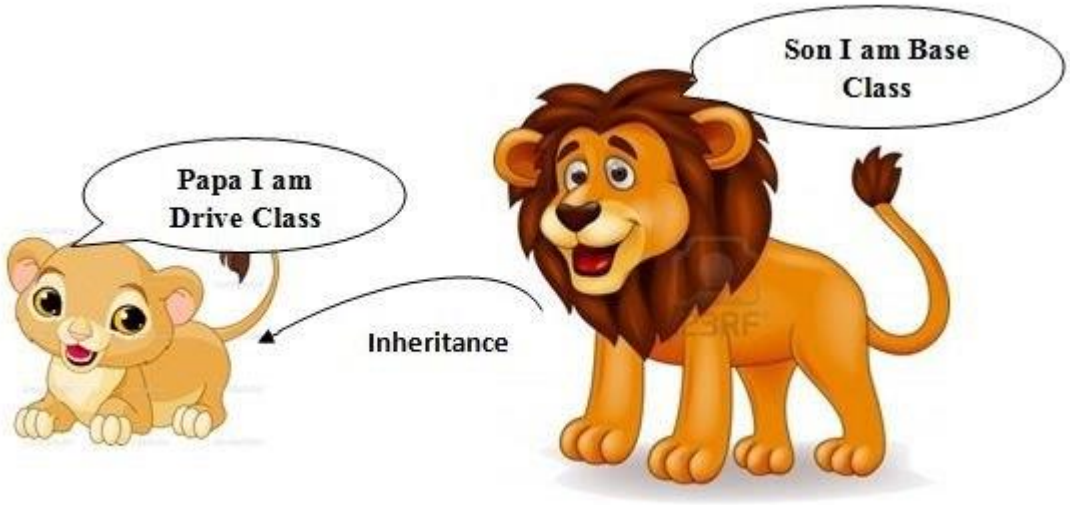
- 封装（Encapsulation）

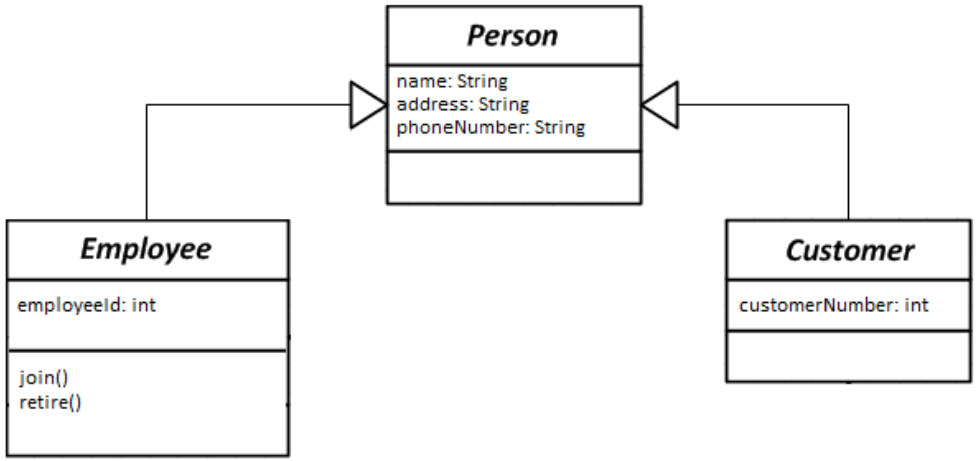


Mom's details are hidden or encapsulated from Son.

5. 面向对象的分析与设计（OOAD）

- 继承（Inheritance）



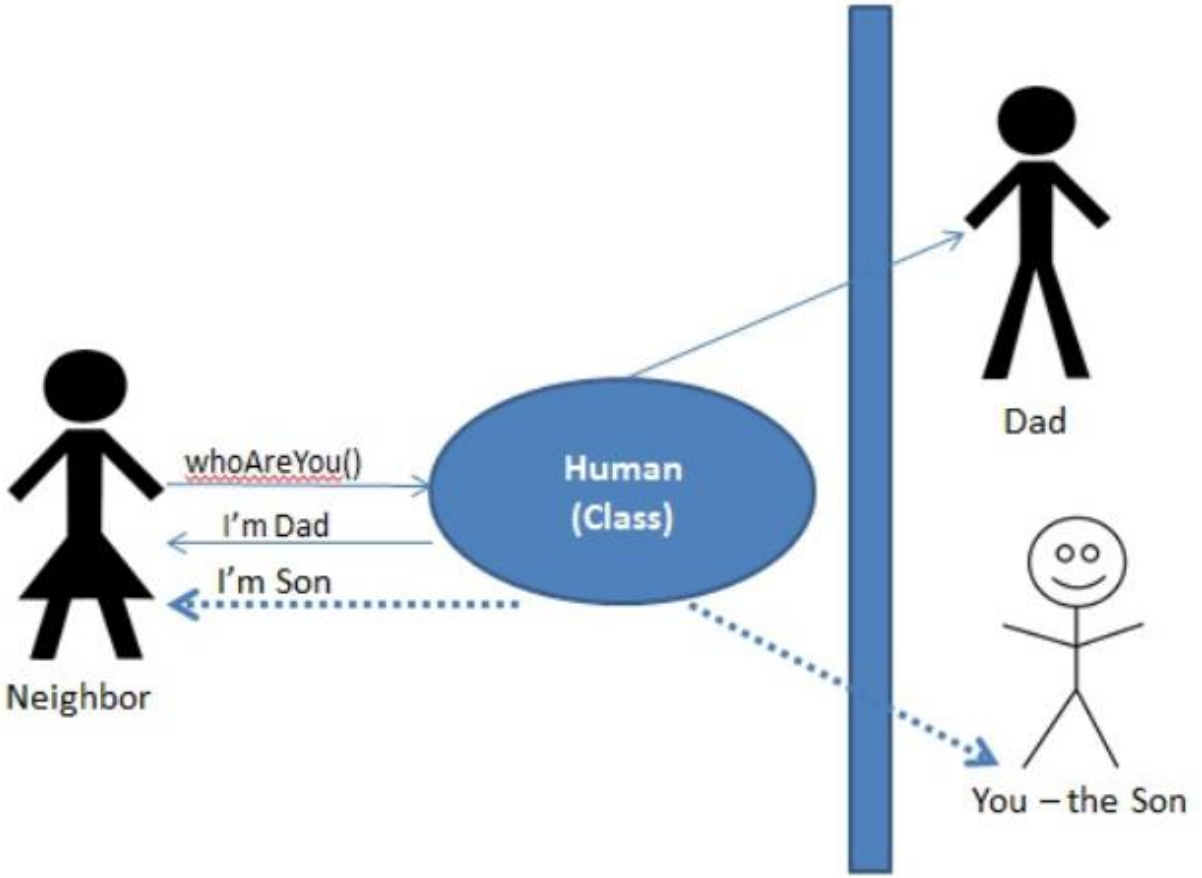


5. 面向对象的分析与设计（OOAD）

- 多态（Polymorphism）

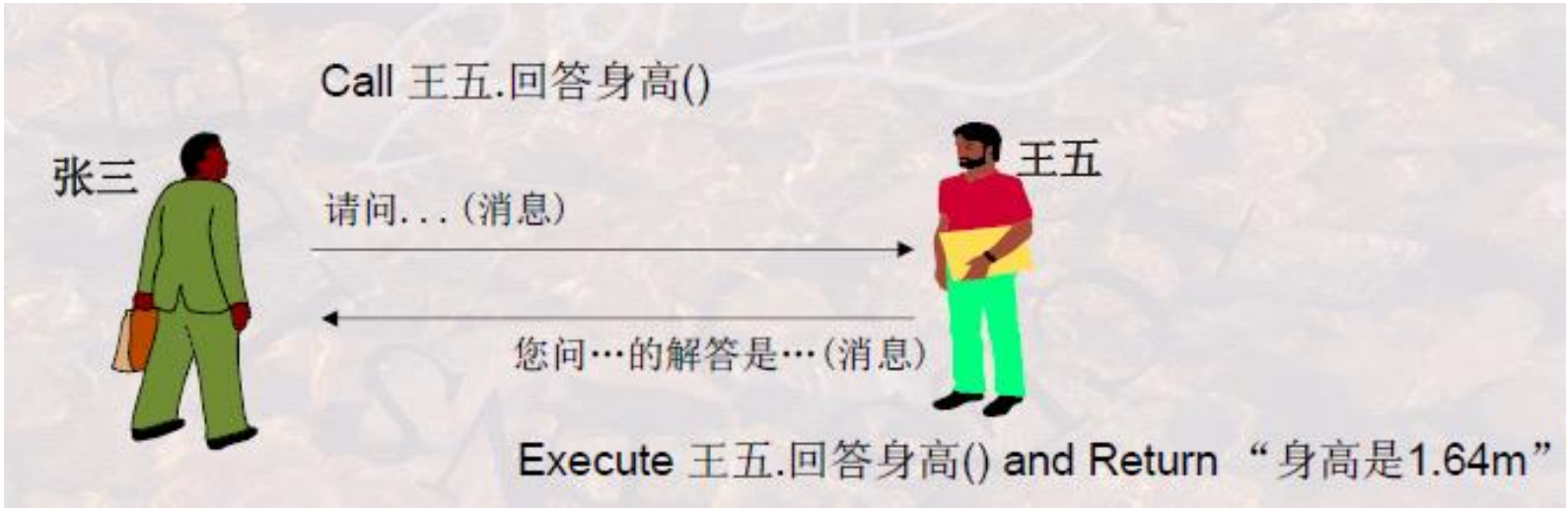
5. 面向对象的分析与设计（OOAD）

- 多态（Polymorphism）



5. 面向对象的分析与设计（OOAD）

- 对象之间通过消息（message）进行交互，消息是“对象.成员函数()”的调用和执行

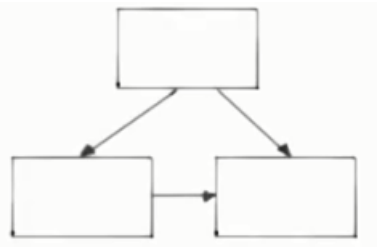


5. 面向对象的分析与设计（OOAD）

- 为什么使用OO?

- 减少维护成本：封装和信息隐藏使得对系统的修改更加容易，而不会影响其他组件

- 改善开发过程：提高代码和设计的重用性



- 促使好的设计：好的设计原则

5. 面向对象的分析与设计（OOAD）

- 某公司决定在其软件开发过程中采用OO方法，他们将从哪些方面获益？

- ☐ 模块化编码的风格将增加代码的重用性
- ☐ 由于系统的设计更容易适应变化，将增加系统的可维护性
- ☐ 可加快系统的运行速度
- ☐ 由于设计对现实世界的实体进行建模，将加深对问题的理解

5. 面向对象的分析与设计（OOAD）

- 某公司决定在其软件开发过程中采用OO方法，他们将从哪些方面获益？

- [√] 模块化编码的风格将增加代码的重用性
- [√] 由于系统的设计更容易适应变化，将增加系统的可维护性
- [ ] 可加快系统的运行速度
- [√] 由于设计对现实世界的实体进行建模，将加深对问题的理解

5. 面向对象的分析与设计（OOAD）

- 考虑在线购物网站的如下需求：“用户可以一次添加多个在售的商品到购物车内”下列哪些元素应该抽象为类？

- [ ] 商品
- [ ] 在售
- [ ] 购物车
- [ ] 用户

5. 面向对象的分析与设计（OOAD）

- 考虑在线购物网站的如下需求：“用户可以一次添加多个在售的商品到购物车内”下列哪些元素应该抽象为类？

√ 商品

□ 在售

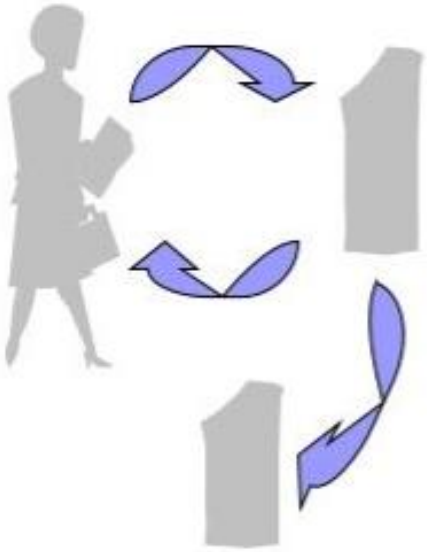
√ 购物车

√ 用户

6. 面向过程 vs. 面向对象

Procedural vs. Object-Oriented

- Procedural



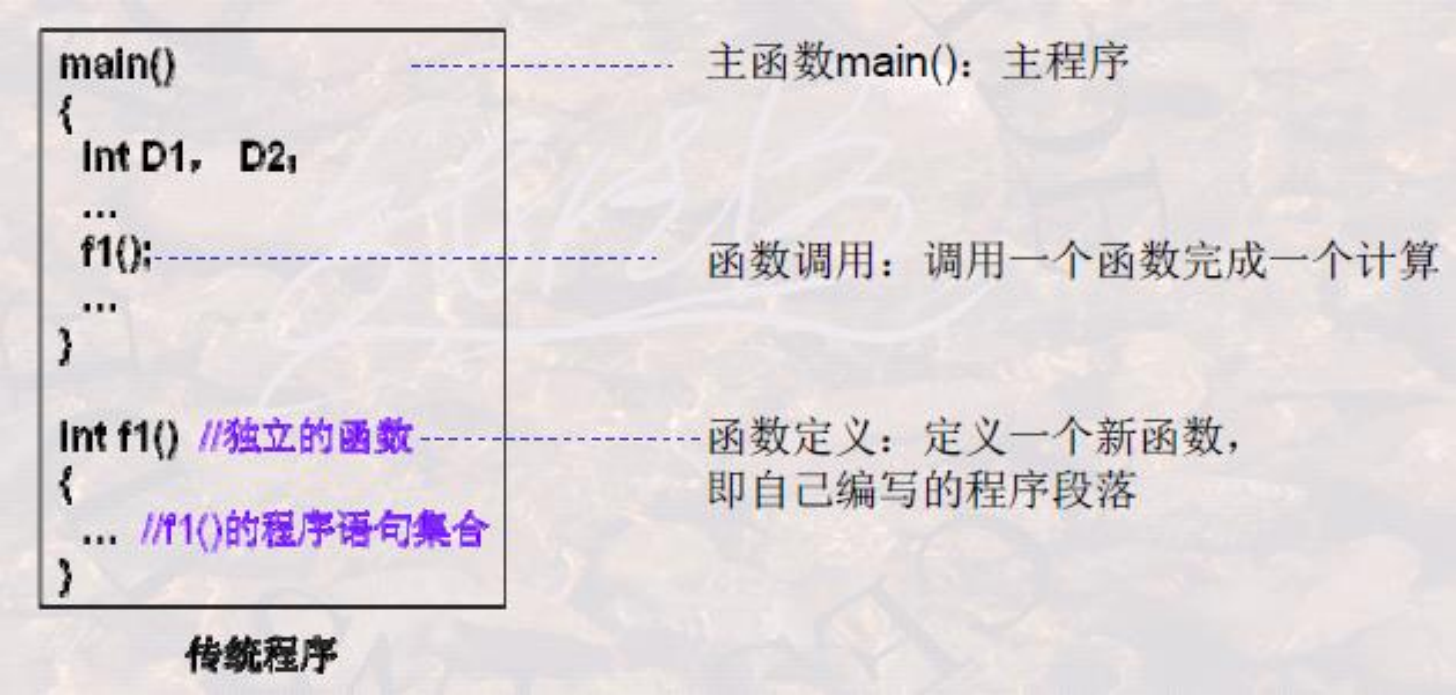
Withdraw, deposit, transfer

- Object Oriented

Customer, money, account

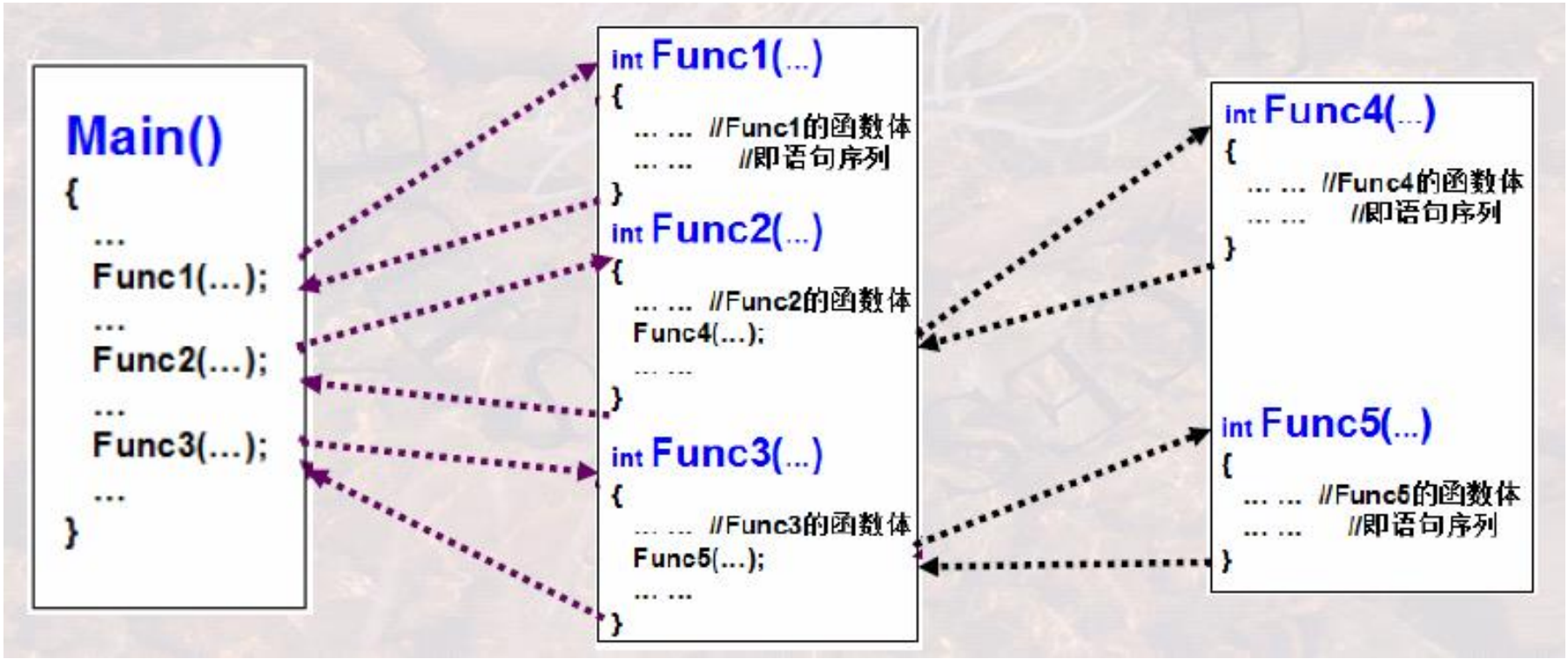
6. 面向过程 vs. 面向对象

- 面向过程的程序的构成：变量与常量、表达式、语句和函数



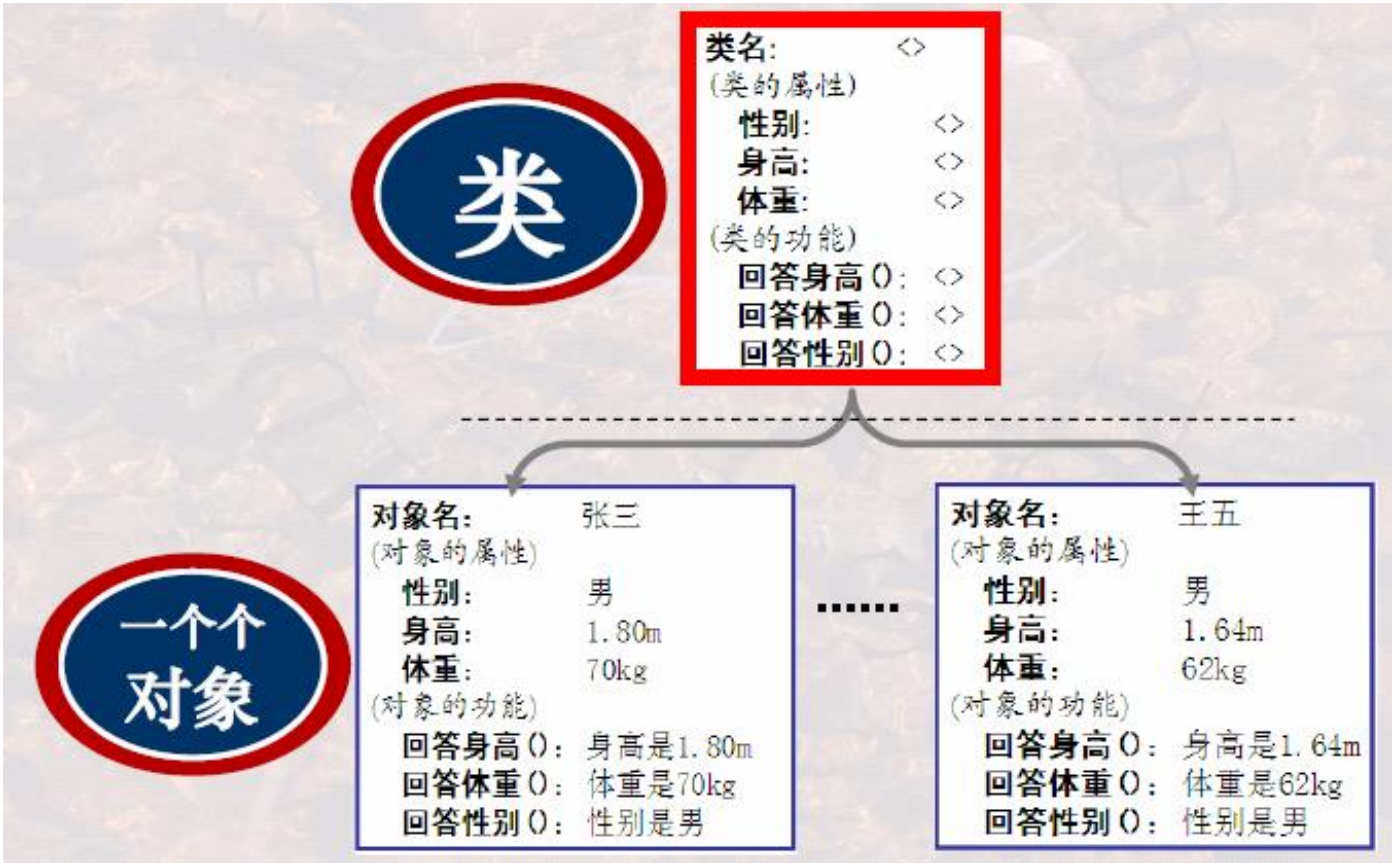
6. 面向过程 vs. 面向对象

- 面向过程的程序：逐步求精



6. 面向过程 vs. 面向对象

- 面向对象的程序的构造方法：



6. 面向过程 vs. 面向对象

- 面向对象的程序：对象之间的交互

体验对象的产生

对象之间的消息交互

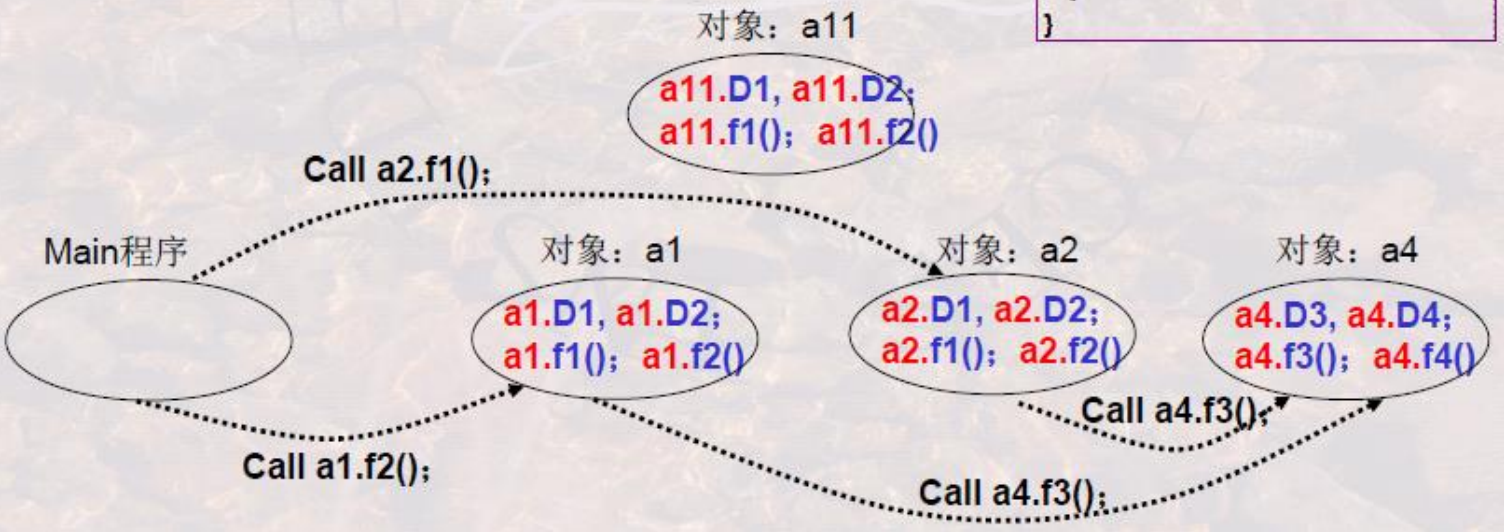
```
Main()
{ int M1, M2;
  a1 = new A1;      //创建类A1的对象a1
  a2 = new A1;      //创建类A1的对象a2
  a11 = new A1;     //创建类A1的对象a11

  M1 = call a1.f2();        //执行对象a1的f2的程序
  M2 = call a2.f1();        //执行对象a2的f1的程序
  a1.D1 = 15;       //对象a1的数据D1被赋值为15
  a2.D1 = 20;       //对象a2的数据D1被赋值为20
  …… ……
}
```

```
Class A1            //定义一个类
{
   int D1, D2       //定义变量
   int f1();        //定义类中的函数
   int f2();        //定义类中的函数

   int f1()         //类A1的函数f1的程序
   { //f1的程序语句块 }

   int f2()         //类A1的函数f2的程序
   { //f2的程序语句块1
     a4 = new A2;   //创建类A2的对象a4
    //f2的程序语句块2
     D1 = call a4.f3(); //调用另一对象的某一函数
    //f2的程序语句块3
   }
}
```



6. 面向过程 vs. 面向对象

```
Main()
{
    int  D1, D2
    int sum; sum=D1;
    call f1(sum);
    call f2();
}

int f1(int qty)
{   int temp;
    temp=qty*qty;
    return(qty);
    //f1的程序语句块 }

int f2()
{ //f2的程序语句块 }
```

全局变量

独立定义的函数

局部变量

```
Class  A1
{     int  D1, D2;
      int  f1(int qty)
      {   int temp; … …//f1的程序语句块 }
}
int A1:: f2()
{  //f2的程序语句块 }
int f2() { //f2的程序语句块 }
Main() {  …
    a1 = new A1;
    a2 = new A1;
    a1.D1 = M1;
    call f2();
    call  a1.f1(M1);
    call  a2.f1(M2);
}
```

定义“类”

类中定义的函数

类中定义的函数

独立定义的函数

用类产生对象

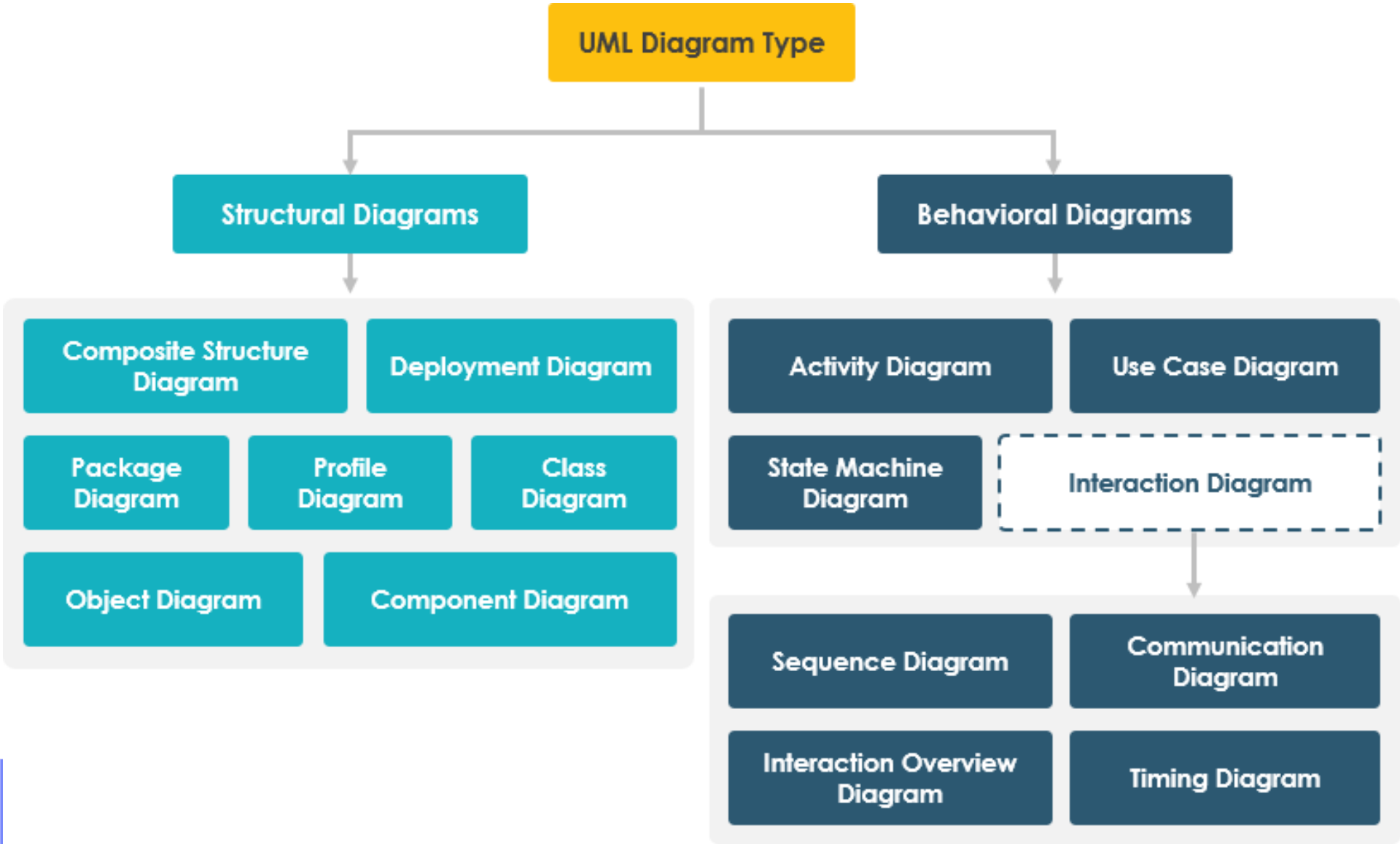
调用独立的函数

调用对象的函数

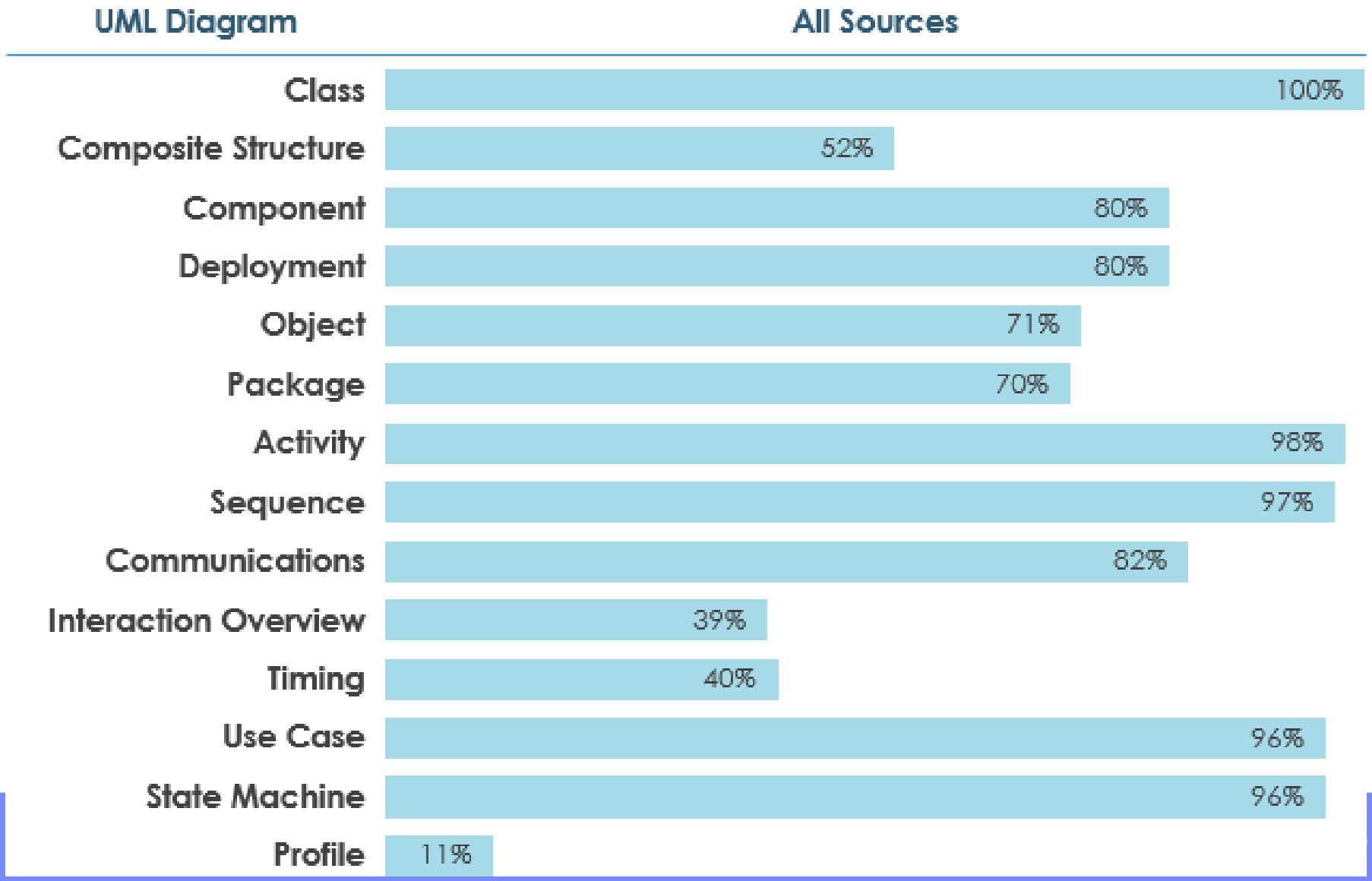
7. UML（统一建模语言）

- 面向对象程序分析、设计与构造的一种表达方法
- 包含了类图、时序图、状态图、用例图等图形化的表达方法
- 便于人们交流分析设计的成果
- 软件工程领域的一种共用的表达方法

7. UML（统一建模语言）

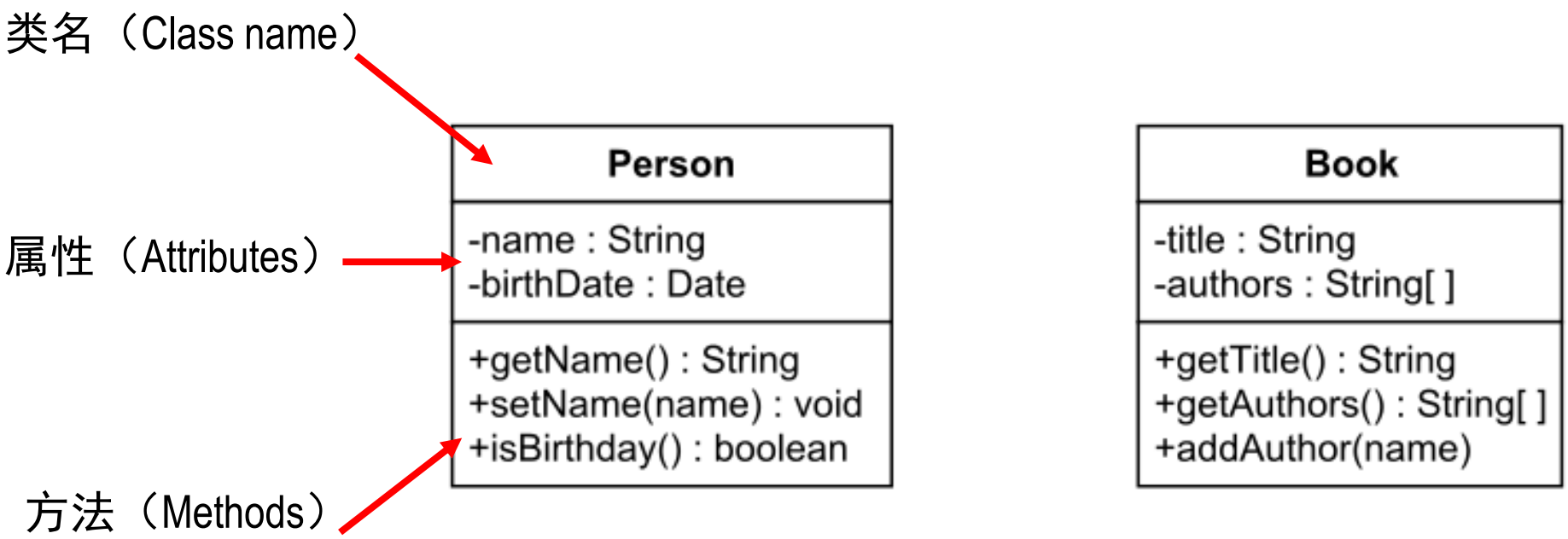

UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

7. UML（统一建模语言）


UML Diagram
All Sources
Class
100%
Composite Structure
52%
Component
80%
Deployment
80%
Object
71%
Package
70%
Activity
98%
Sequence
97%
Communications
82%
Interaction Overview
39%
Timing
40%
Use Case
96%
State Machine
96%
Profile
11%

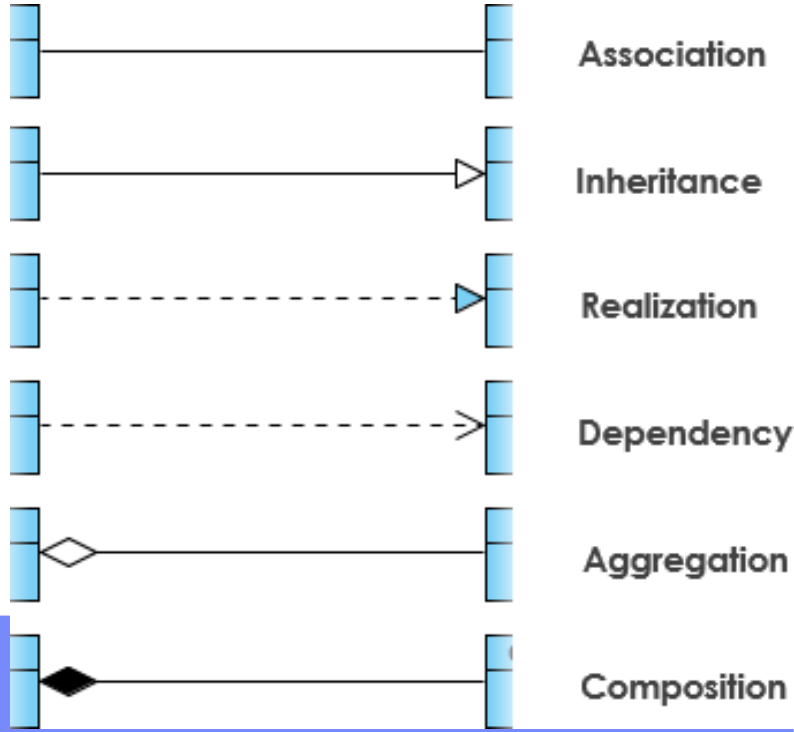
7. UML（统一建模语言）

- 类图：描述类及其之间关系的一种图示化方法



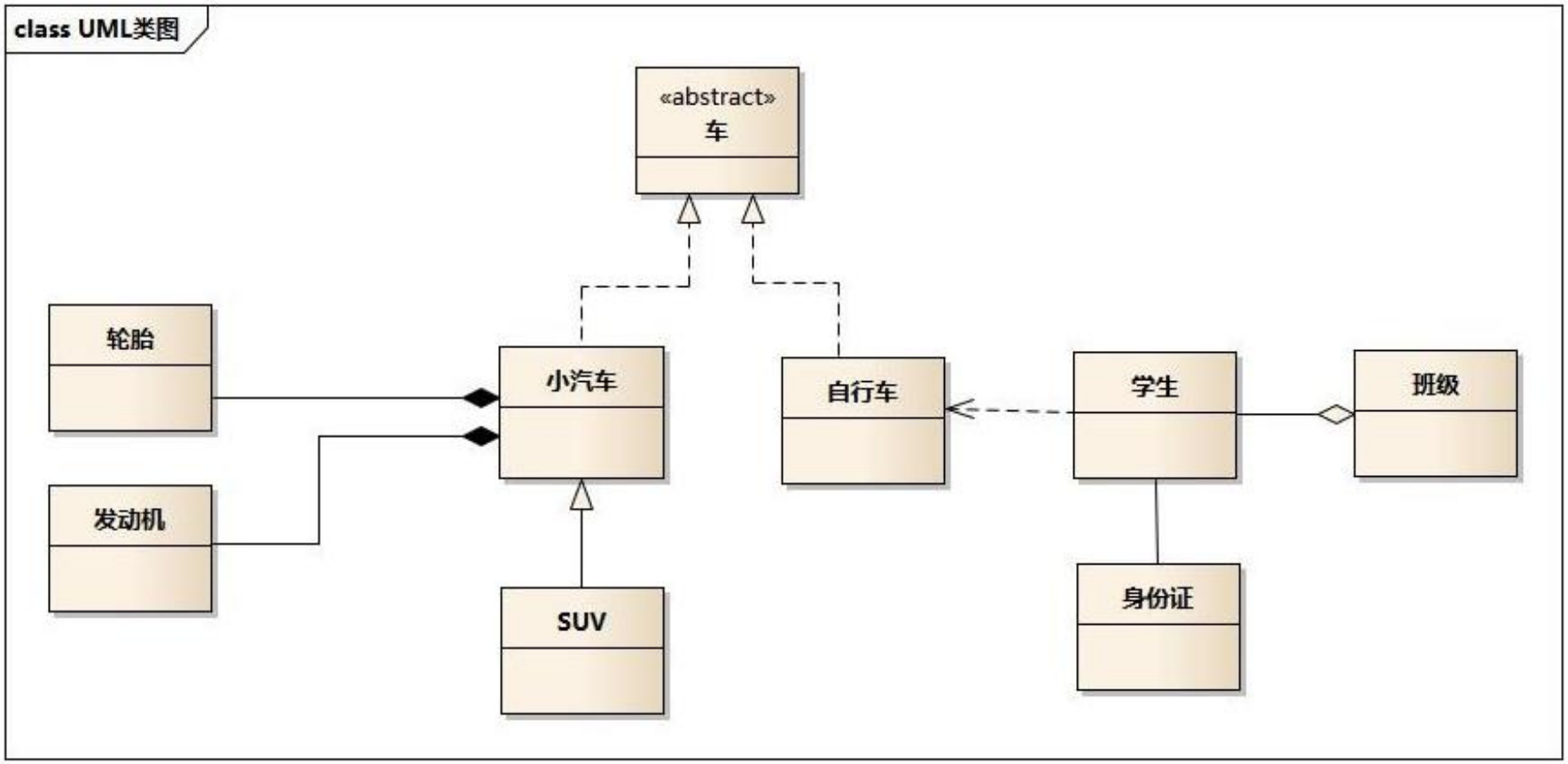
7. UML（统一建模语言）

- 类图：描述类及其之间关系的一种图示化方法
  - 依赖（Dependency）：X uses Y
  - 关联/聚合/组合（Associations/Aggregations/Composition）：X has a Y
  - 继承、泛化（Inheritance/Generalization）：X is a Y



7. UML（统一建模语言）

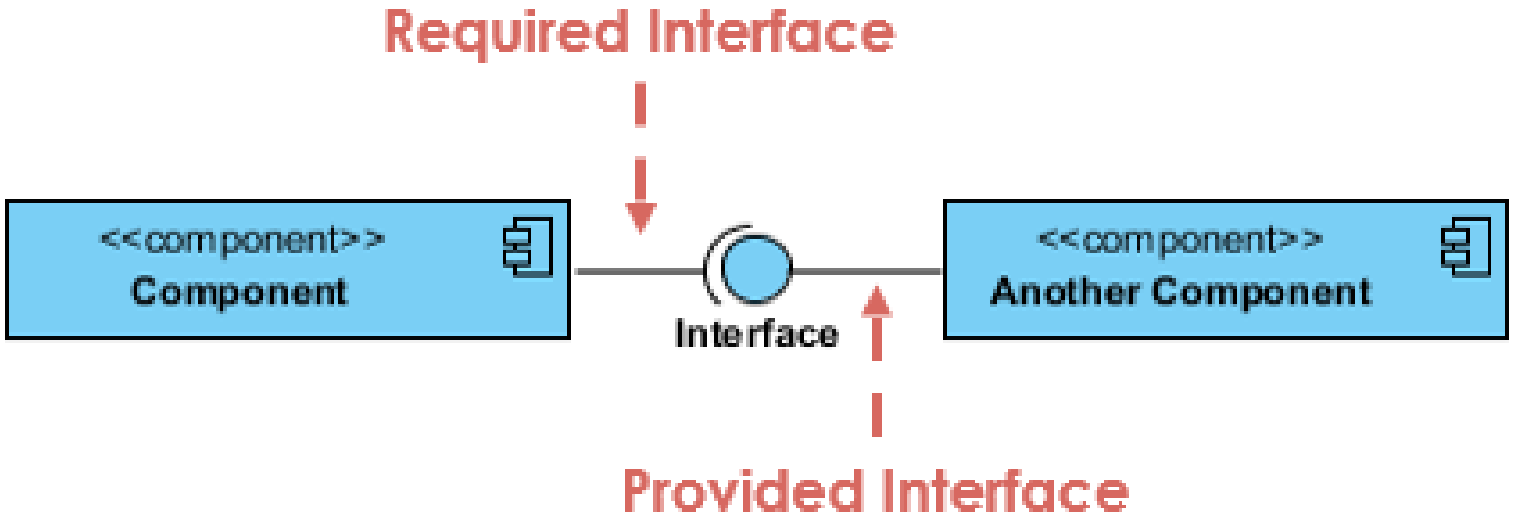
- 类图：描述类及其之间关系的一种图示化方法



7. UML（统一建模语言）

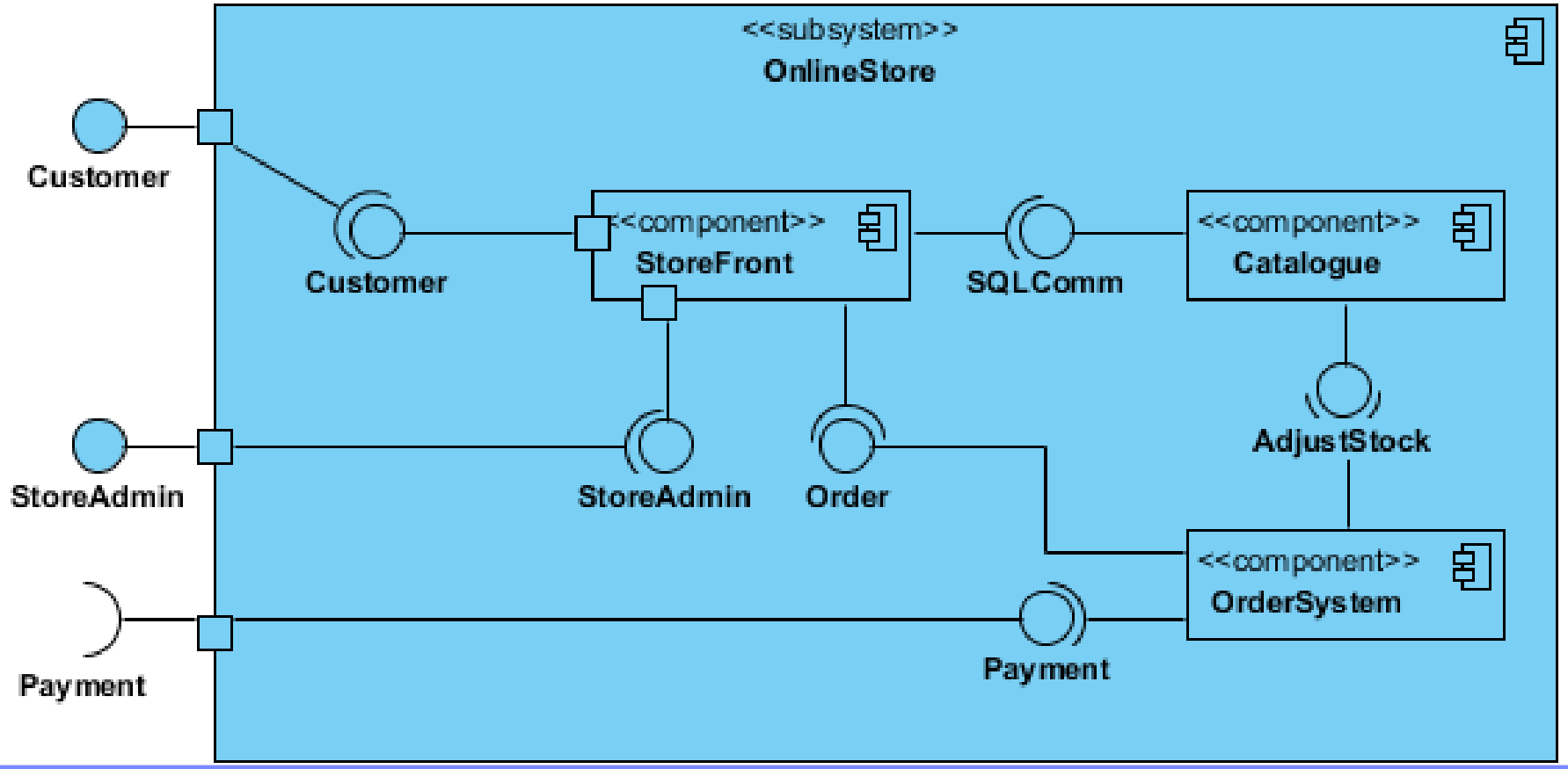
- 组件图（Component Diagram）

  - 组件和组件之间关系的静态视图
  - 节点：组件（具有良好定义接口一组类）
  - 边：关系（提供的服务、需要的服务）



7. UML（统一建模语言）

- 组件图（Component Diagram）

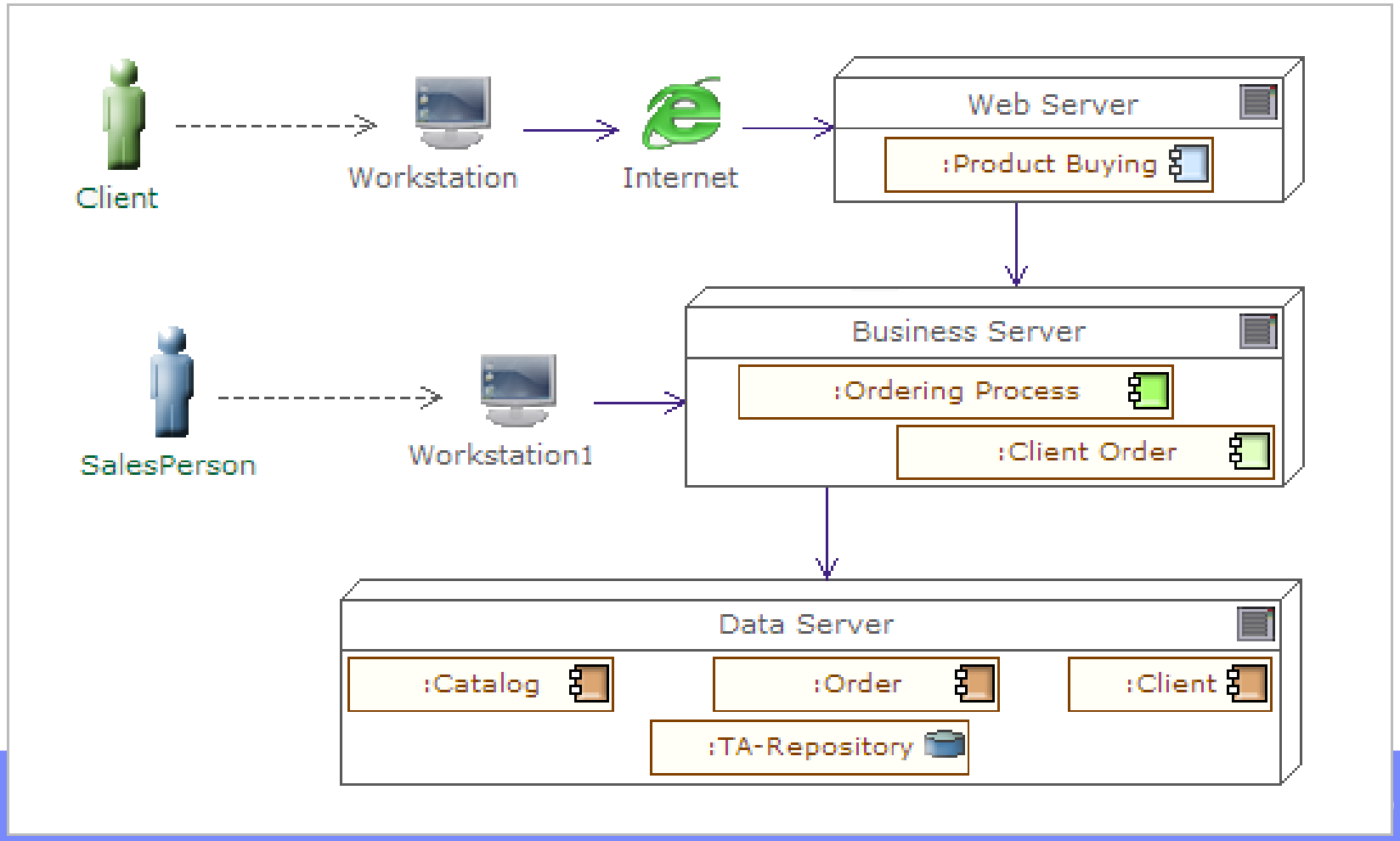


7. UML（统一建模语言）

- 部署图（Deployment Diagram）
  - 系统部署的静态视图
  - 将组件物理分配到计算单元
  - 节点：计算单元
  - 边：计算单元之间的通信

7. UML（统一建模语言）

- 部署图（Deployment Diagram）



7. UML（统一建模语言）

- 下列哪些图是UML结构图（Structural Diagrams）：

  - ☐ 用例图（Use case diagram）
  - ☐ 类图（Class diagram）
  - ☐ 部署图（Deployment diagram）
  - ☐ 序列图（Sequence diagram）

7. UML（统一建模语言）

- 下列哪些图是UML结构图（Structural Diagrams）：

☐ 用例图（Use case diagram）

☑ 类图（Class diagram）

☑ 部署图（Deployment diagram）

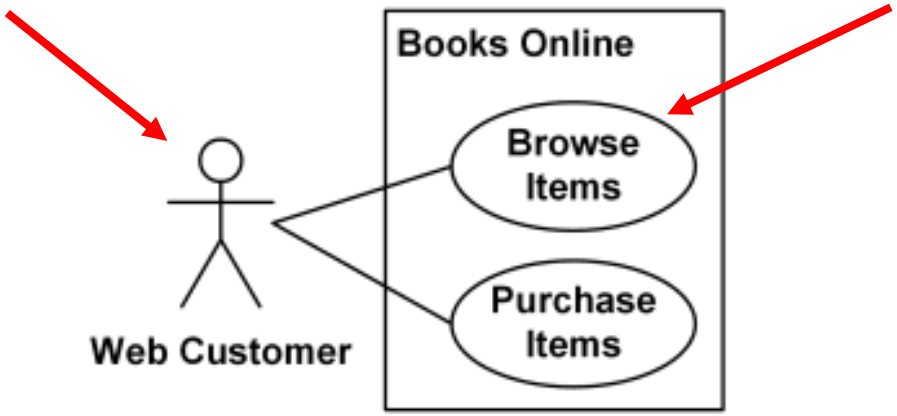
☐ 序列图（Sequence diagram）

7. UML（统一建模语言）

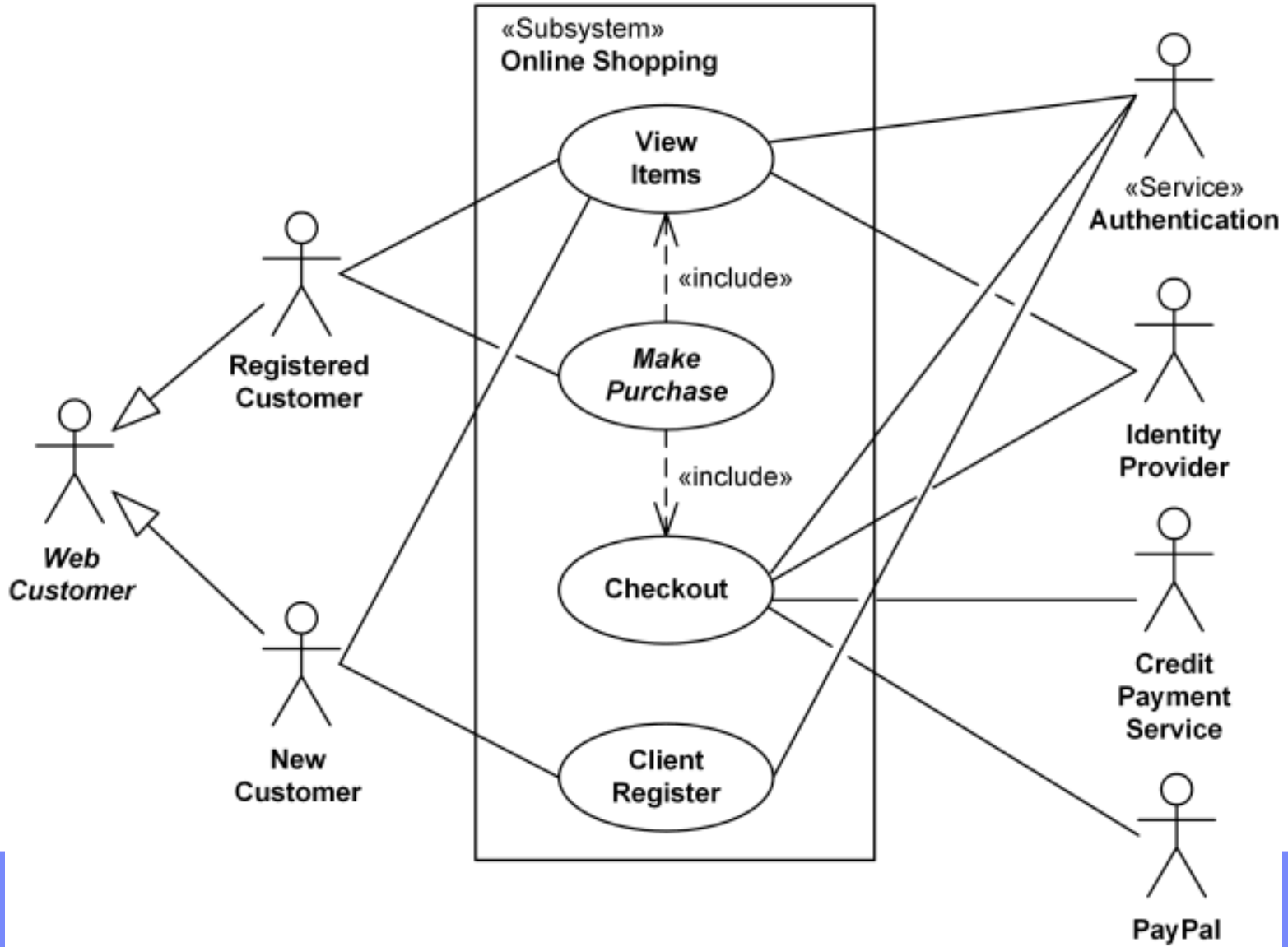
- 用例图（Use Case Diagrams）：从外部角度来描述系统
  - 外部实体（Actor）与系统的交互
  - 系统向外部实体展示可观测的结果
  - 也被称为：场景、脚本、用户故事等

Actor：用户、设备或其他系统

Use Case：系统和Actor之间的交互

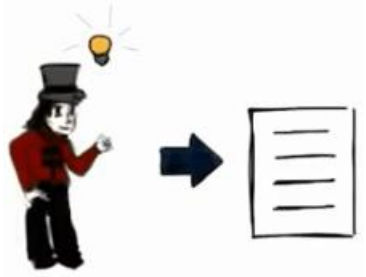


7. UML（统一建模语言）

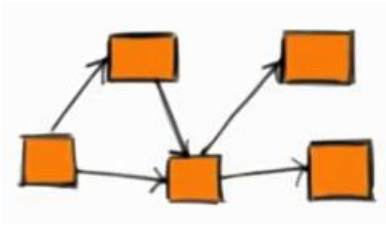

«Subsystem»
Online Shopping
View Items
«include»
Make Purchase
«include»
Checkout
Client Register
Registered Customer
Web Customer
New Customer
«Service»
Authentication
Identity Provider
Credit Payment Service
PayPal

7. UML（统一建模语言）

- 用例图的作用



需求获取（requirement elicitation）



架构分析（architectural analysis）

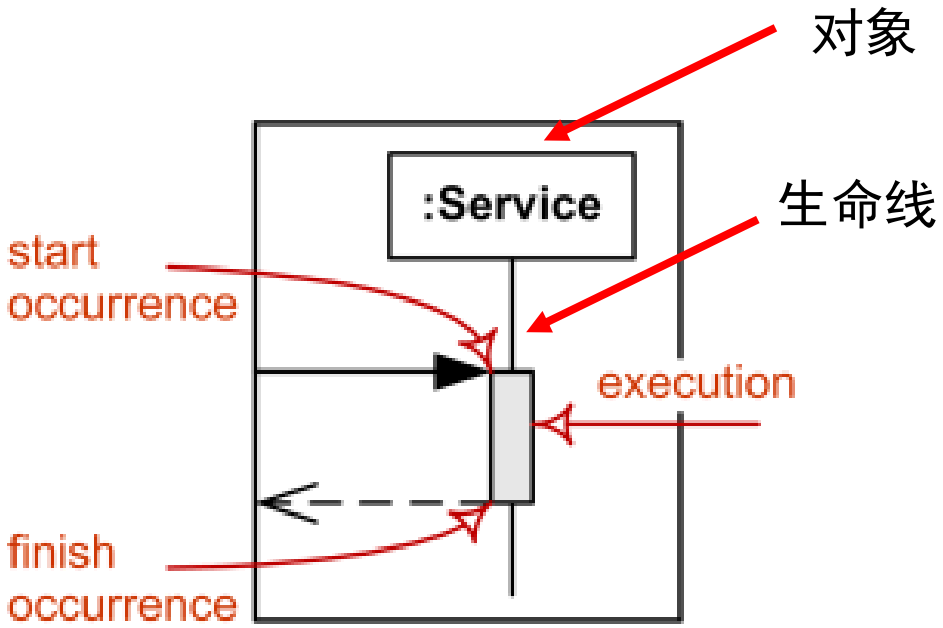
用户优先级（user prioritization）

规划（planning）

测试（testing）

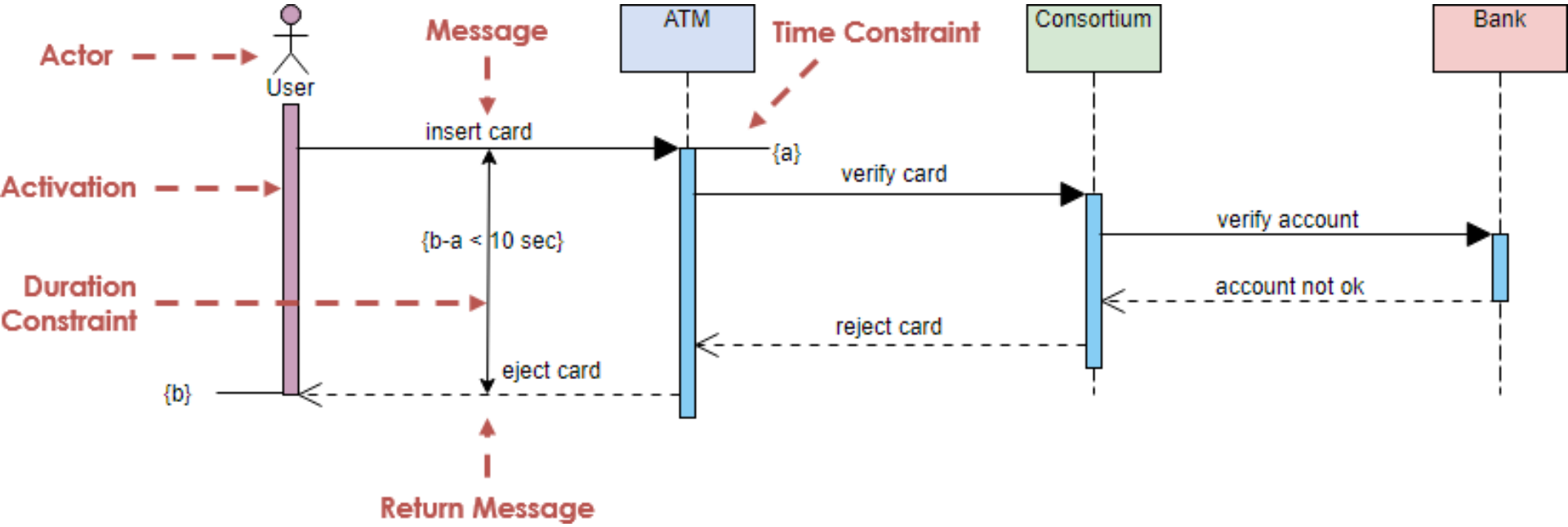
7. UML（统一建模语言）

- 序列图（Sequence Diagram）
  - 强调对象之间消息交互的时间顺序



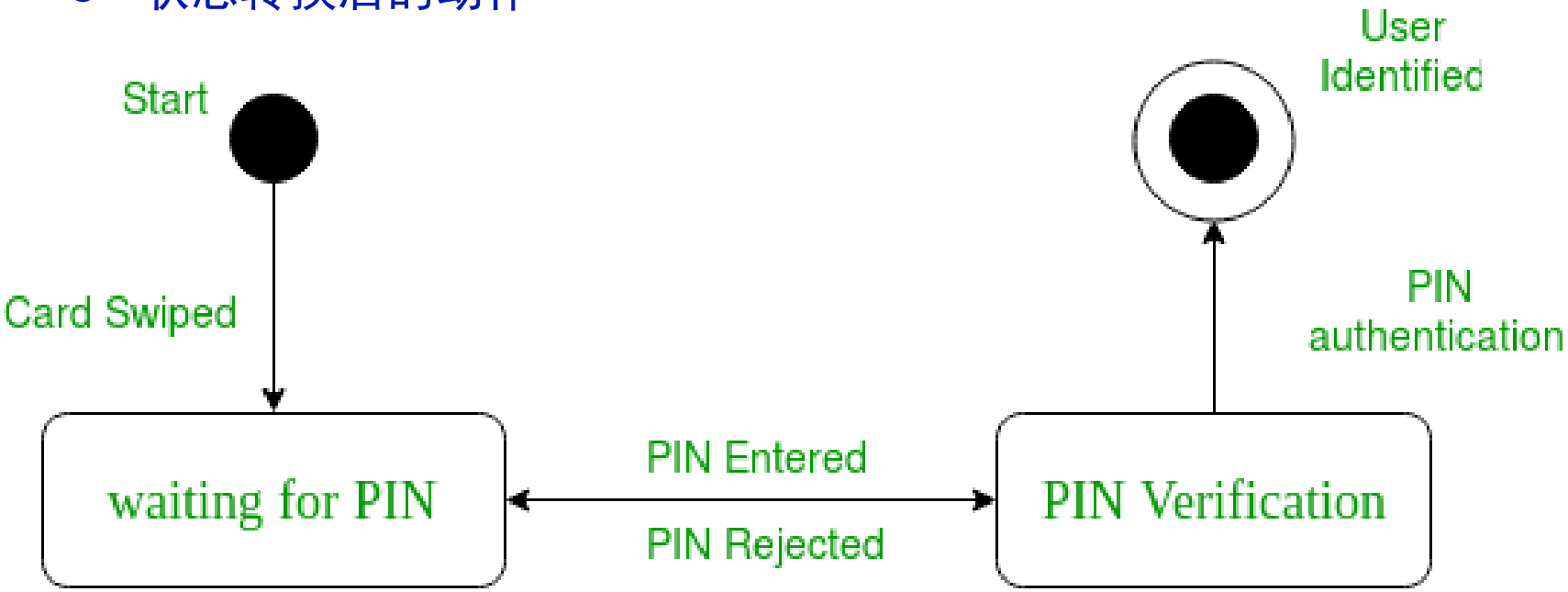
7. UML（统一建模语言）

- 序列图（Sequence Diagram）



7. UML（统一建模语言）

- 状态转换图（State Transition Diagram）
  - 类或对象可能的状态
  - 导致状态之间转换的事件
  - 状态转换后的动作



7. UML（统一建模语言）

- UML状态转换图说明：
  - 一组对象一起工作执行某个动作
  - 导致对象从一个状态转换到另一个状态的事件
  - 系统中组件
  - 状态转换的效果

7. UML（统一建模语言）

- UML状态转换图说明：

□ 一组对象一起工作执行某个动作

√ 导致对象从一个状态转换到另一个状态的事件

□ 系统中组件

√ 状态转换的效果

8. 案例：课程管理系统

- 系统描述

1. 管理员利用调度算法安排一个学期的课程
2. 一门课可有多个课程设置
3. 每个课程设置有编号、地点、时间
4. 学生通过提交注册表选择4门必修课和2门选修课
5. 学生注册后，在一定的时间内可以使用系统来增加、删除课程
6. 教师使用系统获得课程的学生名册
7. 使用注册系统的用户可以被分配密码，用作登录验证

8. 案例：课程管理系统

- 从描述中抽取可能成为类的名词

1. 管理员利用调度算法安排一个学期的课程
2. 一门课可有多个课程设置
3. 每个课程设置有编号、地点、时间
4. 学生通过提交注册表选择4门必修课和2门选修课
5. 学生注册后，在一定的时间内可以使用系统来增加、删除课程
6. 教师使用系统获得课程的学生名册
7. 注册系统的用户可以被分配密码，用作登录验证

8. 案例：课程管理系统

- 类图

8. 案例：课程管理系统

- 类的属性（Attribute）
  - 检查类的定义
  - 研究需求
  - 应用领域知识

8. 案例：课程管理系统

1. 管理员利用调度算法安排一个学期的课程
2. 一门课可有多个课程设置
3. 每个课程设置有编号、地点、时间
4. 学生通过提交注册表选择4门必修课和2门选修课
5. 学生注册后，在一定的时间内可以使用系统来增加、删除课程
6. 教师使用系统获得课程的学生名册
7. 使用注册系统的用户可以被分配密码，用作登录验证

8. 案例：课程管理系统

- 类的属性（Attribute）
  - 检查类的定义
  - 研究需求
  - 应用领域知识

每个课程设置有编号、地点、时间

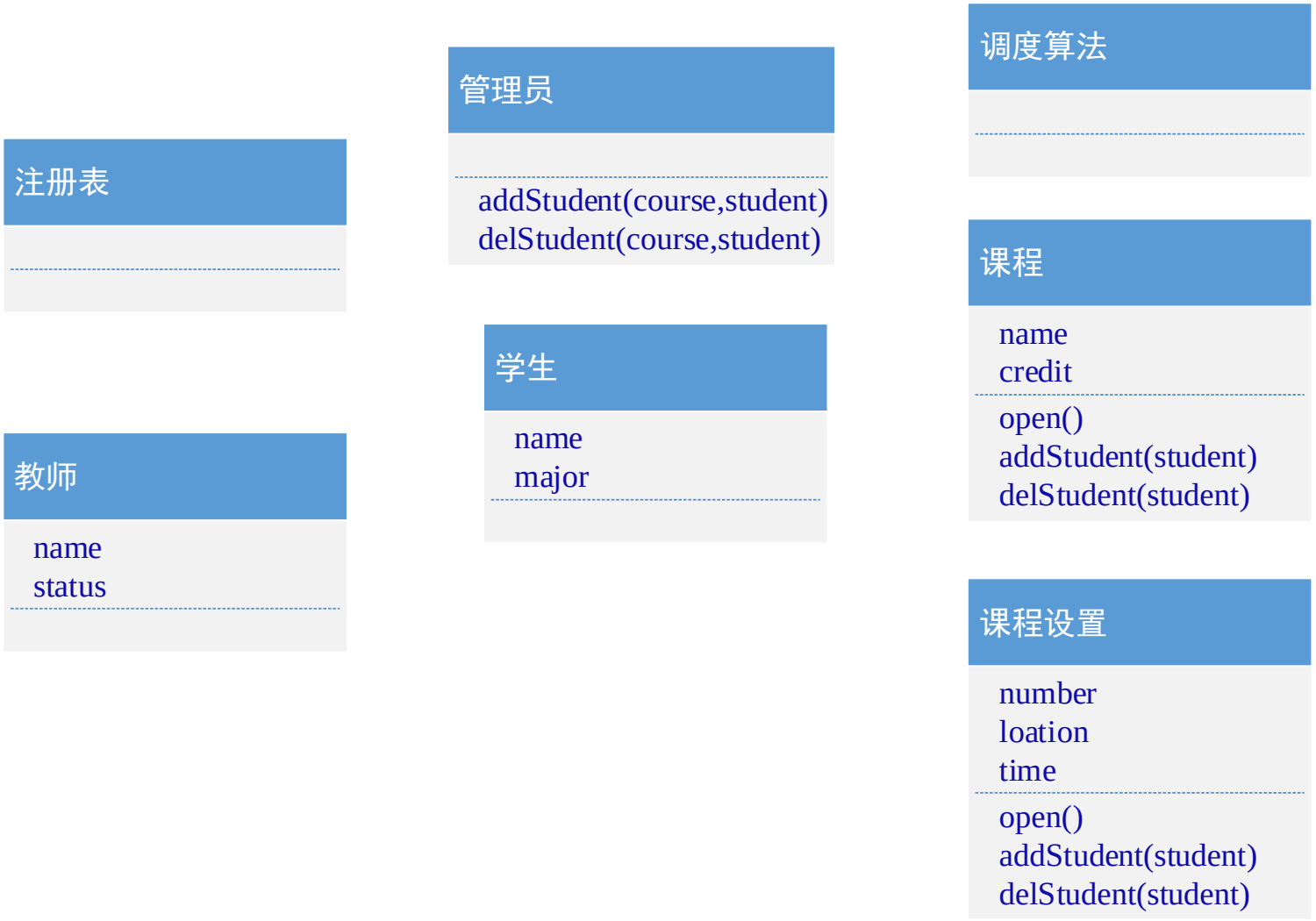
8. 案例：课程管理系统

- 类的操作（Operation）
  - 表示类的行为
  - 可通过检查实体之间的交互来发现

8. 案例：课程管理系统

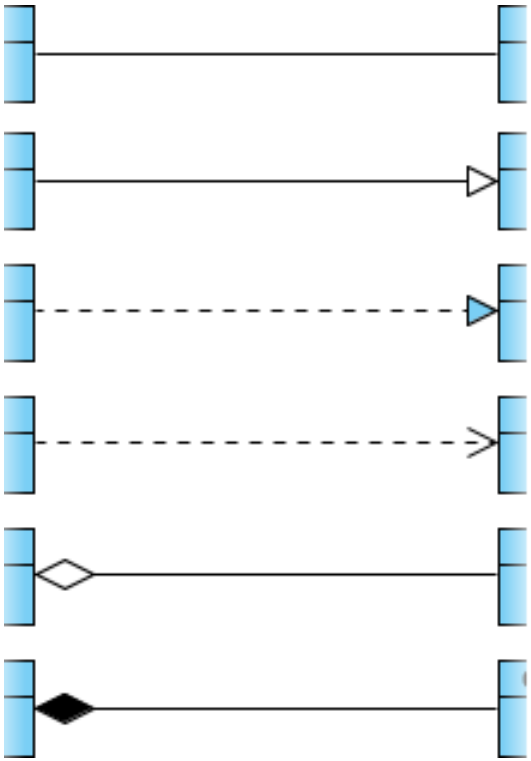
1. 管理员利用调度算法安排一个学期的课程
2. 一门课可有多个课程设置
3. 每个课程设置有编号、地点、时间
4. 学生通过提交注册表选择4门必修课和2门选修课
5. 学生注册后，在一定的时间内可以使用系统来增加、删除课程
6. 教师使用系统获得课程的学生名册
7. 使用注册系统的用户可以被分配密码，用作登录验证

8. 案例：课程管理系统


注册表
管理员
addStudent(course,student)
delStudent(course,student)
调度算法
课程
name
credit
open()
addStudent(student)
delStudent(student)
学生
name
major
教师
name
status
课程设置
number
loation
time
open()
addStudent(student)
delStudent(student)

8. 案例：课程管理系统

- 类之间的关系（Relationship）
  - 描述对象之间的交互



8. 案例：课程管理系统

- 下列哪些关系与模型相符？

  - ☐ 管理员使用调度算法（依赖）
  - ☐ 学生使用管理员（依赖）
  - ☐ 学生注册课程（关联）
  - ☐ 学生包含课程（聚合）
  - ☐ 课程包含课程设置（聚合）
  - ☐ 课程设置是课程（继承）
  - ☐ 学生是一个注册用户（继承）
  - ☐ 教师是一个注册用户（继承）

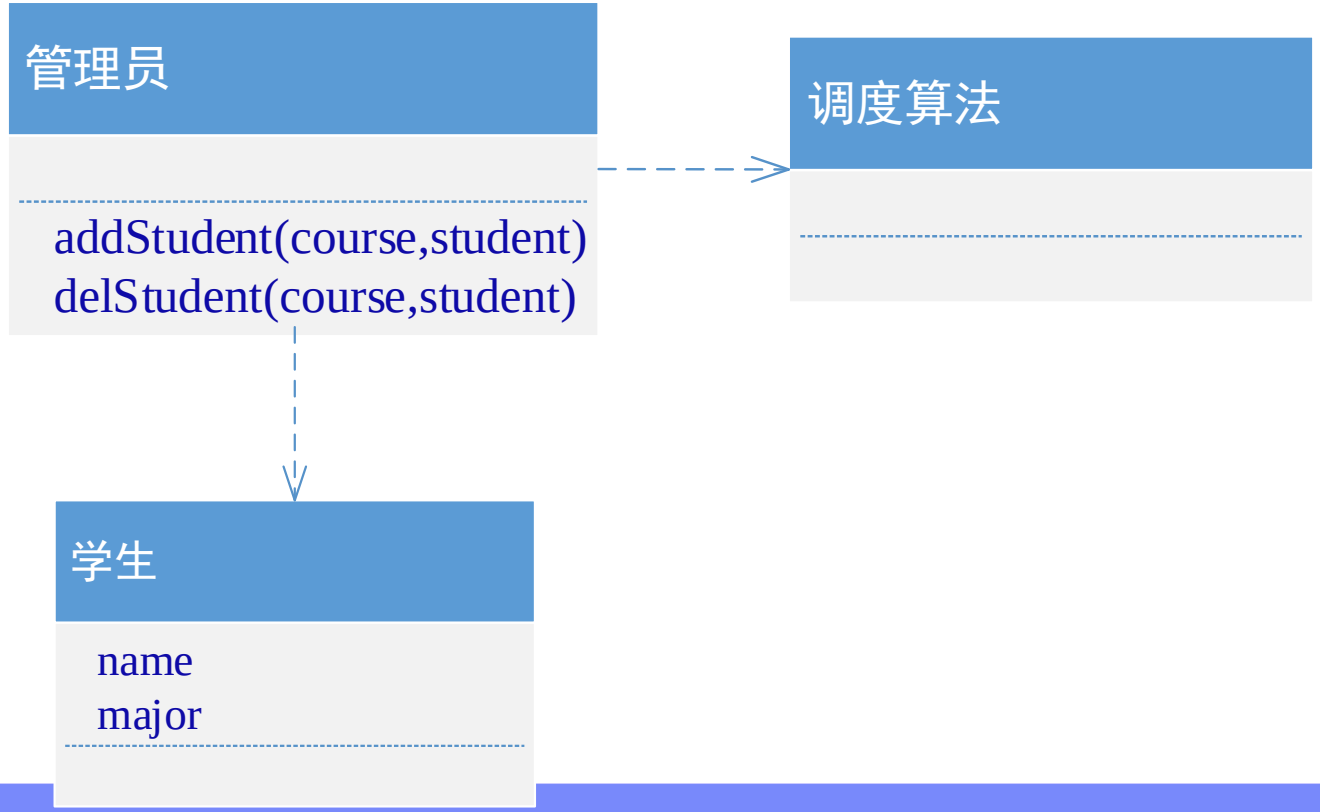
8. 案例：课程管理系统

- 下列哪些关系与模型相符？

- [√] 管理员使用调度算法（依赖）
- [ ] 学生使用管理员（依赖）
- [√] 学生注册课程（关联）
- [ ] 学生包含课程（聚合）
- [√] 课程包含课程设置（聚合）
- [ ] 课程设置是课程（继承）
- [√] 学生是一个注册用户（继承）
- [√] 教师是一个注册用户（继承）

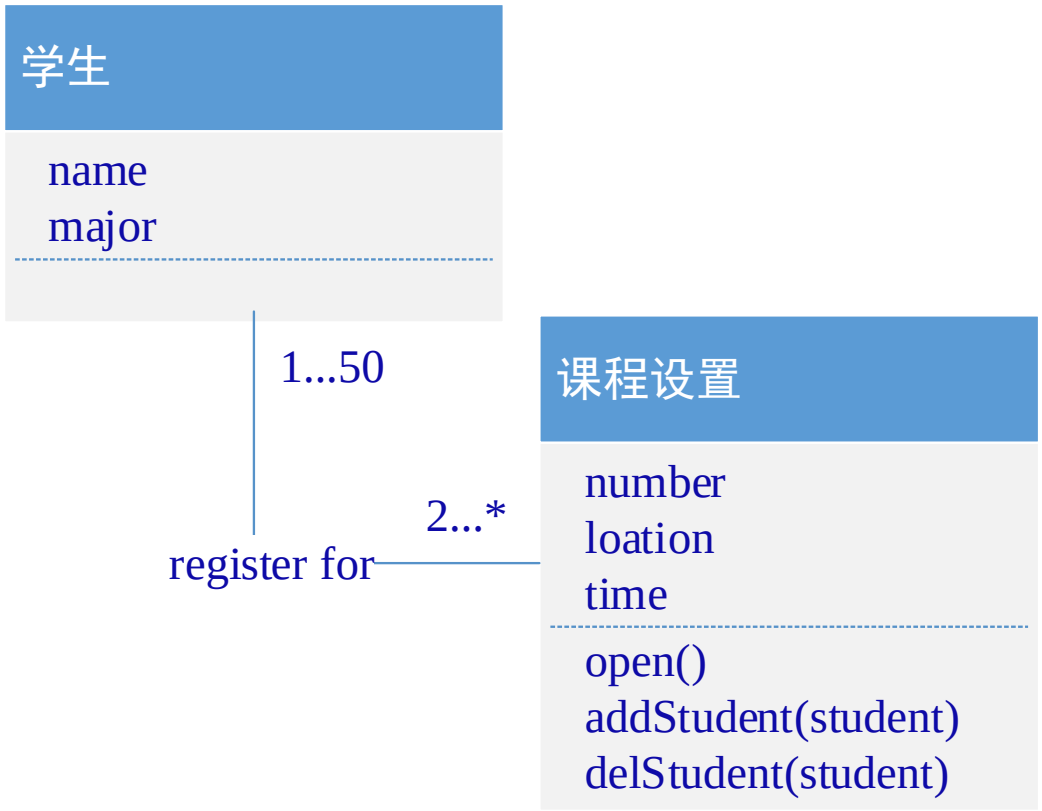
8. 案例：课程管理系统

- Dependency（依赖）：客户与供应者之间的关系
  - 供应者（supplier）的改变会导致客户的改变



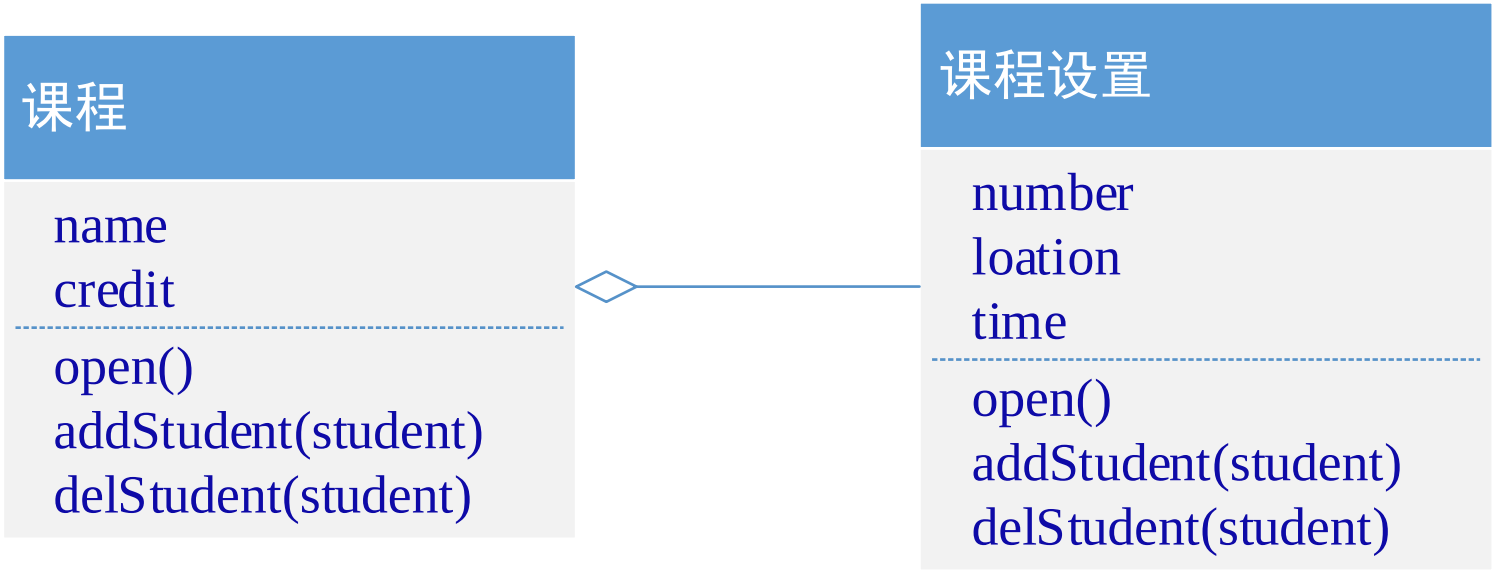
8. 案例：课程管理系统

- Association（关联）：一个类与另一个类的对象的联系（has-a）



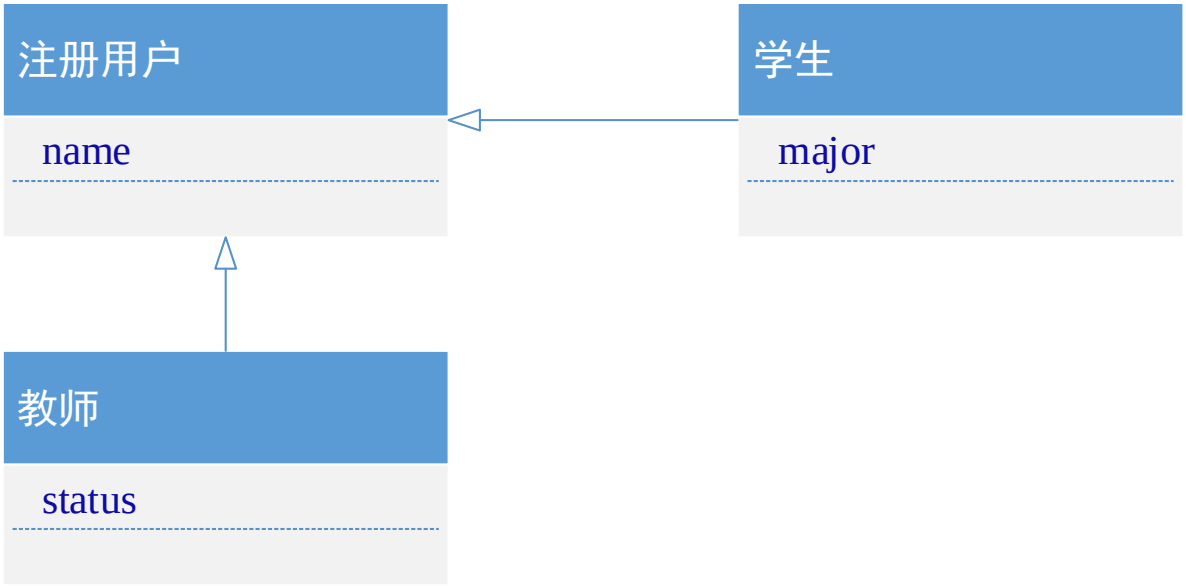
8. 案例：课程管理系统

- Aggregation（聚合）：整体和部分的关系



8. 案例：课程管理系统

- Inheritance/Generalization（继承/泛化）：is-a关系



8. 案例：课程管理系统

- 类图创建的提示
  - 理解问题
  - 选择良好的类名
  - 关注 what 而不是 how
  - 从简单设计开始
  - 细化设计直到感觉完整

8. 案例：课程管理系统

- 用例图

1. 管理员利用调度算法安排一个学期的课程
2. 一门课可有多个课程设置
3. 每个课程设置有编号、地点、时间
4. 学生通过提交注册表选择4门必修课和2门选修课
5. 学生注册后，在一定的时间内可以使用系统来增加、删除课程
6. 教师使用系统获得课程的学生名册
7. 使用注册系统的用户可以被分配密码，用作登录验证

8. 案例：课程管理系统

- 用例图：Actor

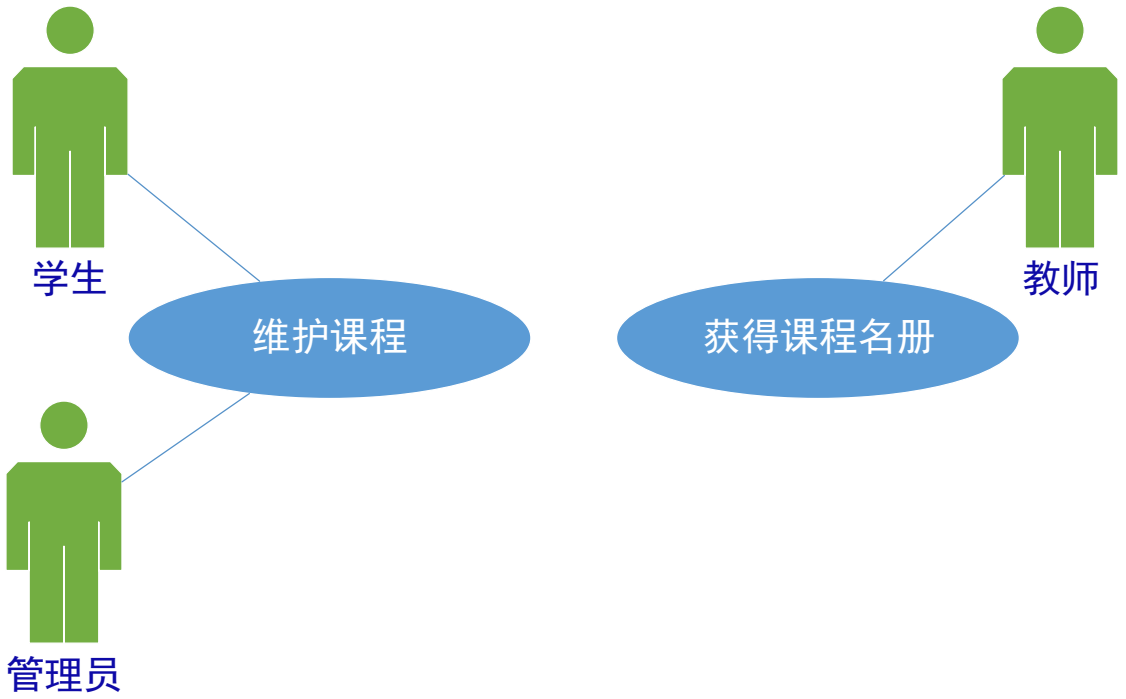
1. 管理员利用调度算法安排一个学期的课程
2. 一门课可有多个课程设置
3. 每个课程设置有编号、地点、时间
4. 学生通过提交注册表选择4门必修课和2门选修课
5. 学生注册后，在一定的时间内可以使用系统来增加、删除课程
6. 教师使用系统获得课程的学生名册
7. 使用注册系统的用户可以被分配密码，用作登录验证

8. 案例：课程管理系统

- 用例图：Actor

8. 案例：课程管理系统

- 用例图



8. 案例：课程管理系统

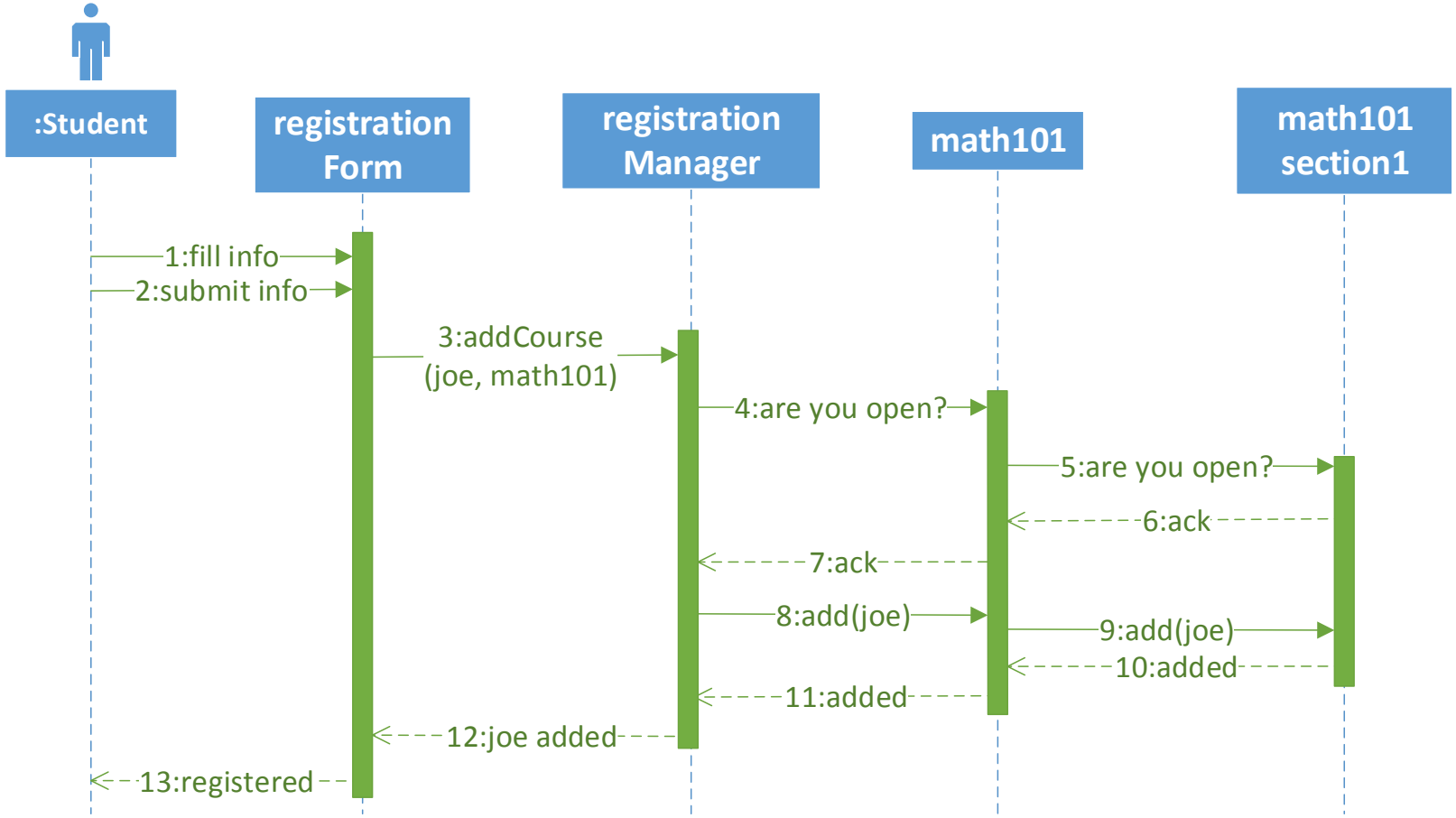
- 用例描述：从actor的角度来描述用例的行为（正式或非正式描述）
  - 用例如何开始和结束
  - 正常的事件流
  - 可选的事件流
  - 异常的事件流

8. 案例：课程管理系统

- 用例描述：课程维护
  - 管理员输入密码登录系统
  - 如果密码正确，系统提示管理员选定某一个学期
  - 管理员选定一个学期
  - 系统提示管理员可选的操作：添加、删除、查看或退出
  - 如果管理员选择添加操作，系统允许管理员在制定学期课程列表中添加课程
  - 如果管理员选择删除操作，系统允许管理员在制定学期课程列表中删除课程
  - 如果管理员选择查看操作，系统允许管理员查看制定学期课程
  - 如果管理员选择退出操作，退出系统（用例结束）

8. 案例：课程管理系统

- 序列图：强调信息交互的时间顺序



小结

- 软件设计是软件成功的关键
- 借鉴类似问题的解决方案（架构设计、设计模式）
- 面向对象分析与设计的描述 ---- UML