软件工程概论

Software Engineering

刘伟

liuwei@xidian.edu.cn

88204608

CH5. Design the System

Content

- Conceptual design and technical
- Design styles, techniques and tools
  - Decomposition and Modularity
  - Architectural Styles and Strategies
  - Important design issues
- Characteristics of good design
- Techniques for Improving Design
- Evaluating and Validating Design
- Documenting the design

5.1 Conceptual design and technical

- ***Design*** is the creative process of transforming the problem into a solution. The description of a solution.
- The new house's requirements that Chuck and Betsy Howell want. **P195**
- There are many proposed designs to solve the problem. In many cases, the number of possible solutions is limitless (无限的).
- The nature(特征) of the solution may change as the solution is described or implemented.

5.1 Conceptual design and technical

- To transform requirements into a working system, designers must satisfy both customers and the system builders on our development team.
- Two stage of design
  - Conceptual design or System design
  - Technical design

5.1 Conceptual design and technical

Conceptual design

- Tells the customer *exactly* what the system will do.
- Answers questions such as the following:
  - Where will the data come from?
  - What will happen to the data in the system?
  - What will the system look like to users?
  - What choices will be offered to users?
  - What is the timing of events?
  - What will the reports and screens look like?

5.1 Conceptual design and technical

Conceptual design

- A good conceptual design should have the following characteristics:
  - in customer language with no technical jargon(行话)
  - describes system functions
  - independent of implementation
  - linked to requirements (definition)

5.1 Conceptual design and technical

Conceptual design

概念设计确定：

- 软件系统的结构
- 各模块功能及模块间联系(接口)

概念设计的过程 ：

(1)设想可能的方案

(2)选取合理的方案

(3)推荐最佳方案

(4)功能分解

(5)设计软件结构

(6)数据库设计

(7)制定测试计划

(8)编写文档

(9)审查与复审

5.1 Conceptual design and technical

Technical design

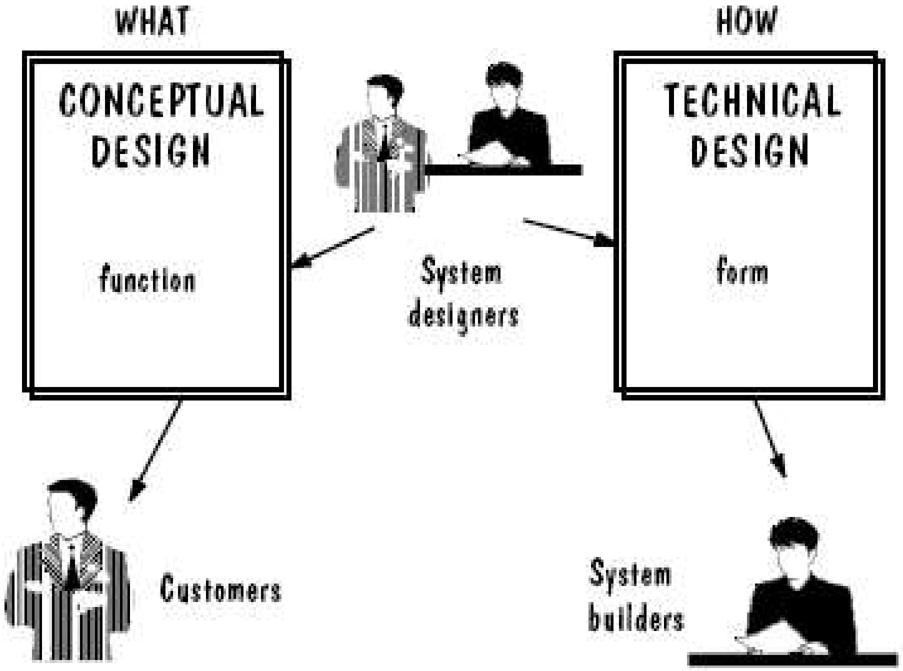
- Allows system builders to understand the actual hardware and software needed to solve the customers' problem.
- Includes:
  - major hardware components and their function
  - hierarchy and function of software components
  - data structures
  - data flow

5.1 Conceptual design and technical

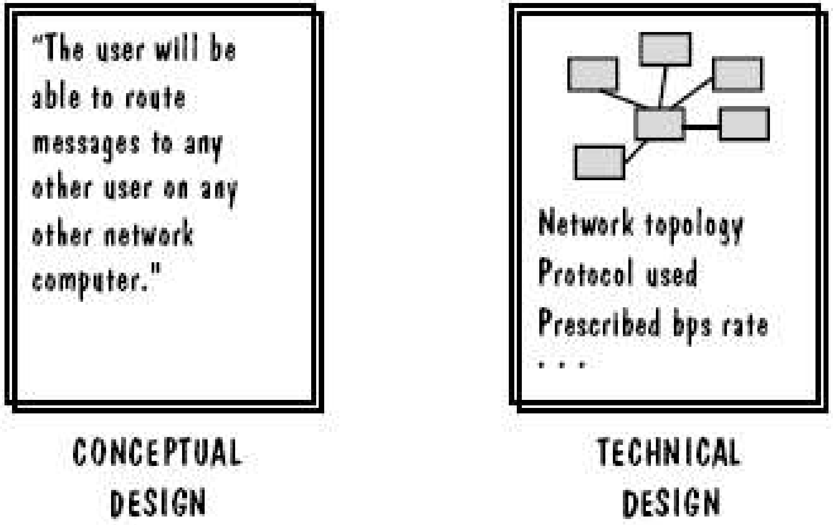
Technical design

- 编写技术设计说明书：
  - 确定每个模块的算法，用工具表达算法的过程，写出模块的详细过程性描述。
  - 确定每一模块的数据结构。
  - 确定模块接口细节。
- 技术（详细）设计是编码的先导。

5.1 Conceptual design and technical



概念和技术设计


"The user will be able to route messages to any other user on any other network computer."

设计文档间的差异

5.2 Design styles, techniques and tools

- Decomposition and Modularity
- Architectural Styles and Strategies
- Important Design Issues

5.2.1 Decomposition and Modularity

- To design a system is to determine a set of ***components*** and ***inter-component interfaces*** that satisfy a specified set of requirements.
- Five ways to create designs. **P199**
  - Modular decomposition
  - Data-oriented decomposition
  - Event-oriented decomposition
  - Outside-in design
  - Object-oriented design

5.2.1 Decomposition and Modularity

- Every design method involves some kind of ***decomposition***: starting with a high-level depiction of the system's key elements and creating lower-level looks at how the system's features and functions will fit together.

- A design can be derived by working from system data descriptions, events, user inputs, high-level functional descriptions, or a combination, and creating a ***hierarchy*** of information with increasing detail.

5.2.1 Decomposition and Modularity

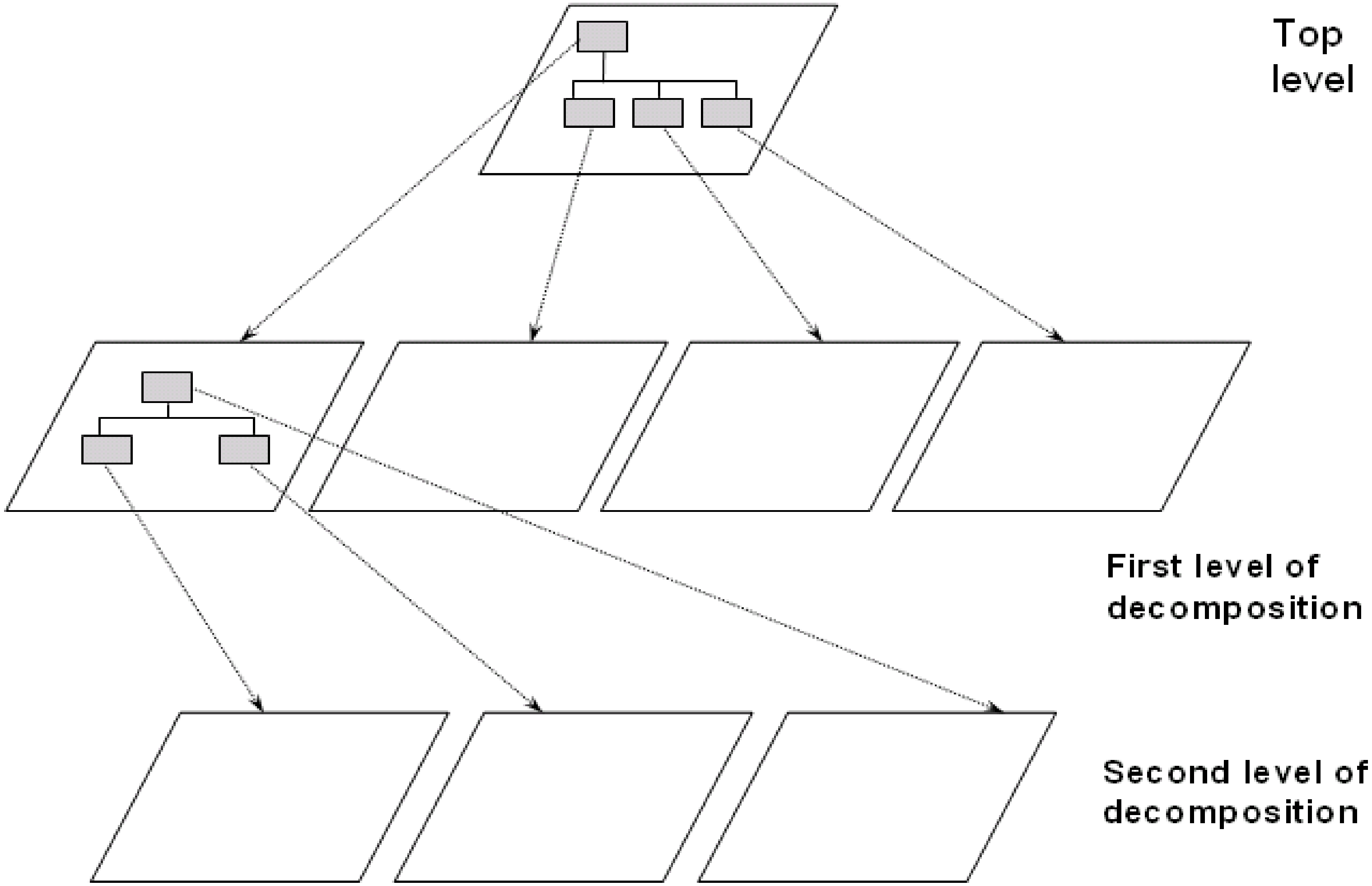


Fig. 5.3 Levels of decomposition

5.2.1 Decomposition and Modularity

- Each kind of decomposition separates the design into its composite parts, called ***modules***(模块) or ***components*** (组件/构件).
- We say that a system is ***modular***(模块化) when each activity of the system is performed by exactly one component, and when the inputs and outputs of each component are well-defined.
- A well-defined component is
  - All inputs to it are essential(必需的) to its function.
  - All outputs are produced by one of its actions.

5.2.2 Architectural Styles and Strategies

- Like the house design, software architecture is also the first step in producing a software.
- Three design levels:
  - ***Architecture***: associates system components with capabilities;
  - ***Code design***: specifies algorithms and data structures for each component;
  - ***Executable design***: lowest level of design, including memory allocation, data formats, bit patterns.

5.2.2 Architectural Styles and Strategies

- It is useful to work from the top down, designing an architecture, then the code design, and finally the executable design.
- It is important for the architecture to provide a cohesive (内聚) "big picture" to guide further design and development.
- Any back-and-forth among design components should preserve this cohesiveness.

5.2.2 Architectural Styles and Strategies

- Just as building reflect a particular architectural style, so, too, can we characterize software architectural style.
  - House: 平房、高楼、板式小高层，哥特、罗马、中国等
  - Software: Client/Server、Browser/Server，Layering等
- Three aspects that a Style involves:
  - Components
  - Connectors
  - Constraints on combining components

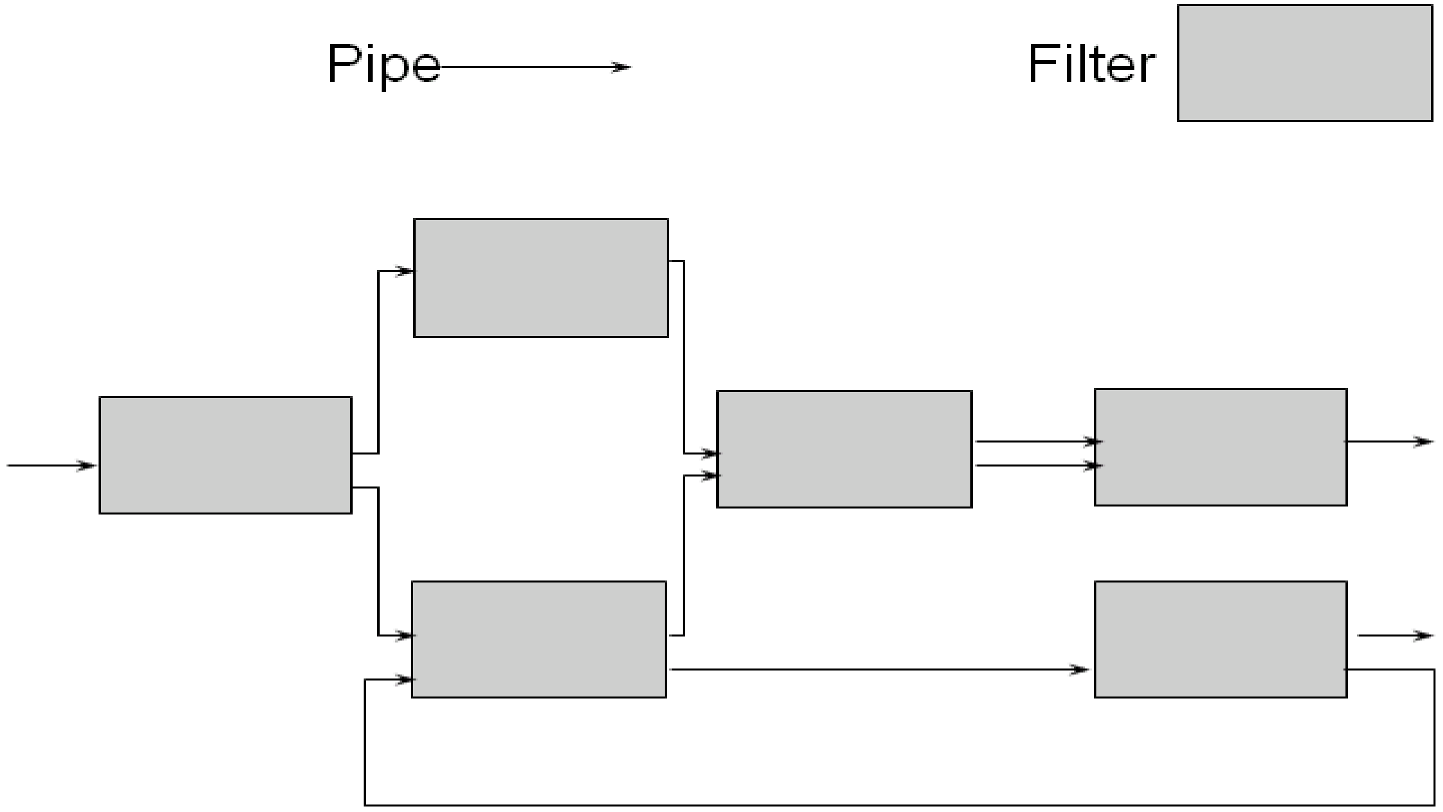
5.2.2 Architectural Styles and Strategies

Commonly used architectural styles

- Pipes and filters (管道和过虑器)
- Object-oriented design (OO设计)
- Implicit invocation (隐含调用)
- Layering (分层)
- Repositories (仓库)
- Interpreters (解释器)
- Process control (过程控制)
- Client-server (客户机/服务器)

5.2.2 Architectural Styles and Strategies

Pipe and filters



5.2.2 Architectural Styles and Strategies

Pipe and filters

- Properties性质
  - Explicit representation of input/output relation I/O关系表示明确
  - Easy for reuse, modification (evolution) and simulation复用、修改、仿真容易
  - Concurrent executing of filters允许并发执行过滤器

- Limitations局限性
  - More appropriate for batch processing更适合批处理（不适合交互式处理）
  - Require correspondence between data streams数据流之间需要对应
  - Potential duplicate operations performed by parallel filters类似过滤器潜在的重复操作执行

5.2.2 Architectural Styles and Strategies

Object-oriented design

- Component = ADT（抽象数据类型）
- Connector = message passing (信息传送)
- Characteristic特征
  - Preserving the integrity of data representation (persistent objects) 保证数据表示的完整性
  - Information hiding (encapsulation) 信息隐藏（封装）
  - Combining functions with data 功能与数据结合
- More depth in Chapter 6.

5.2.2 Architectural Styles and Strategies

Implicit invocation

- The implicit invocation is event-driven, based on the notion of broadcasting (广播).
- Instead of invoking a procedure directly, a component announces (通知) that one or more events have taken place.
- ***Registering*** the procedure
- Data exchange in this type of system must be done through ***shared data*** in a repository.

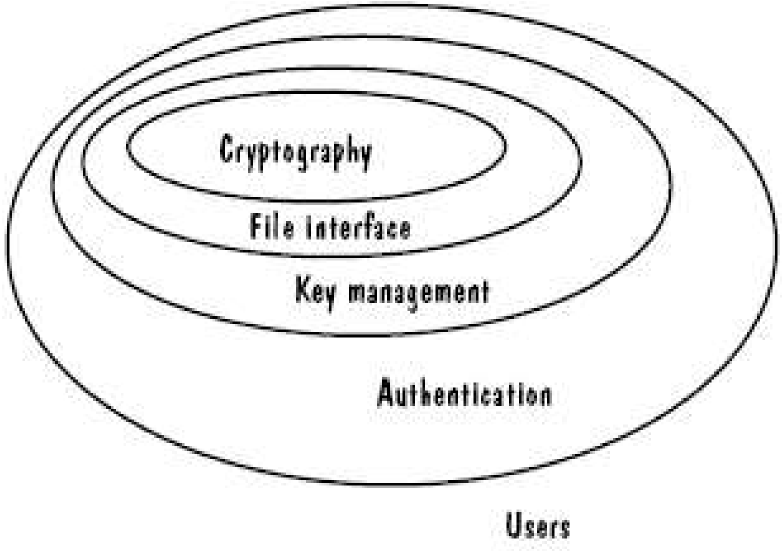
5.2.2 Architectural Styles and Strategies

Implicit invocation

- Advantages
  - Easy reuse of components from other systems
  - Especially useful for user interfaces
- Drawbacks
  - The response to an event is not certain.
  - Difficulty to test the system for all possible sequences of events.
- Combine implicit invocation with explicit invocation.

5.2.2 Architectural Styles and Strategies

Layering



- Component = Layer
- Connector = proc call or protocol
- Layer relationships
  - one layer has access only to adjacent layers
  - one layer has access to some or all other layers

5.2.2 Architectural Styles and Strategies

Layering

- Advantages优点
  - Representing different levels of abstraction表示不同的抽象层次
  - Layers modification usually affects only the two adjacent layers对层的修改通常只影响相邻的两层

- Drawbacks缺点
  - Difficulty In defining the multiple levels of abstraction during the requirement stages需求阶段定义多层抽象很困难
  - Performance issue性能问题?

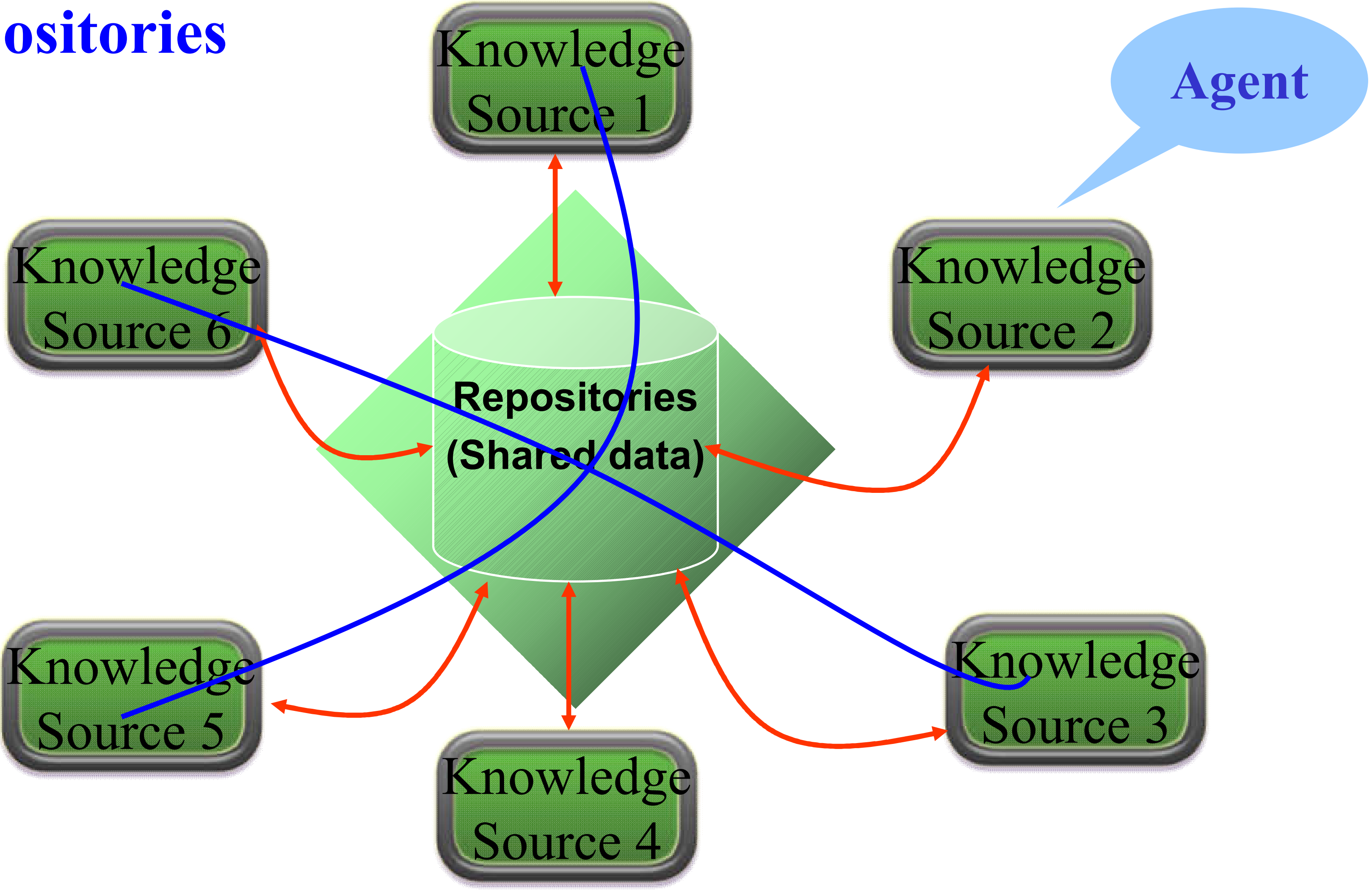
5.2.2 Architectural Styles and Strategies

Repositories

- Two types of components
  - A central data store
  - A collection of independent processes
- Two interaction schemes
  - Traditional database
    - Input stream trigger process execution
  - Blackboard
    - Central store controls the triggering of processes
    - Comes from the artificial intelligence community

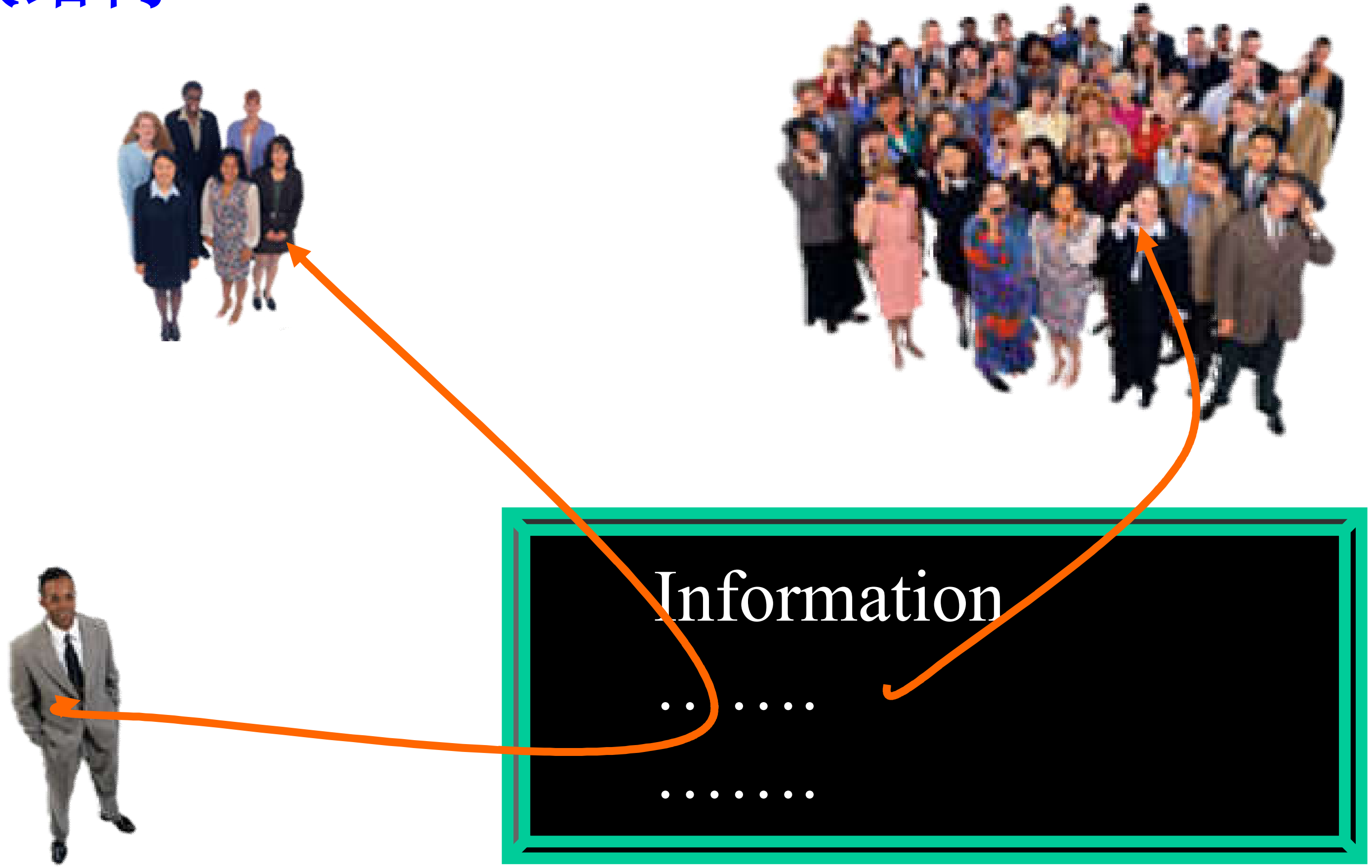
5.2.2 Architectural Styles and Strategies

Repositories



5.2.2 Architectural Styles and Strategies

Repositories有时也被称为黑板结构



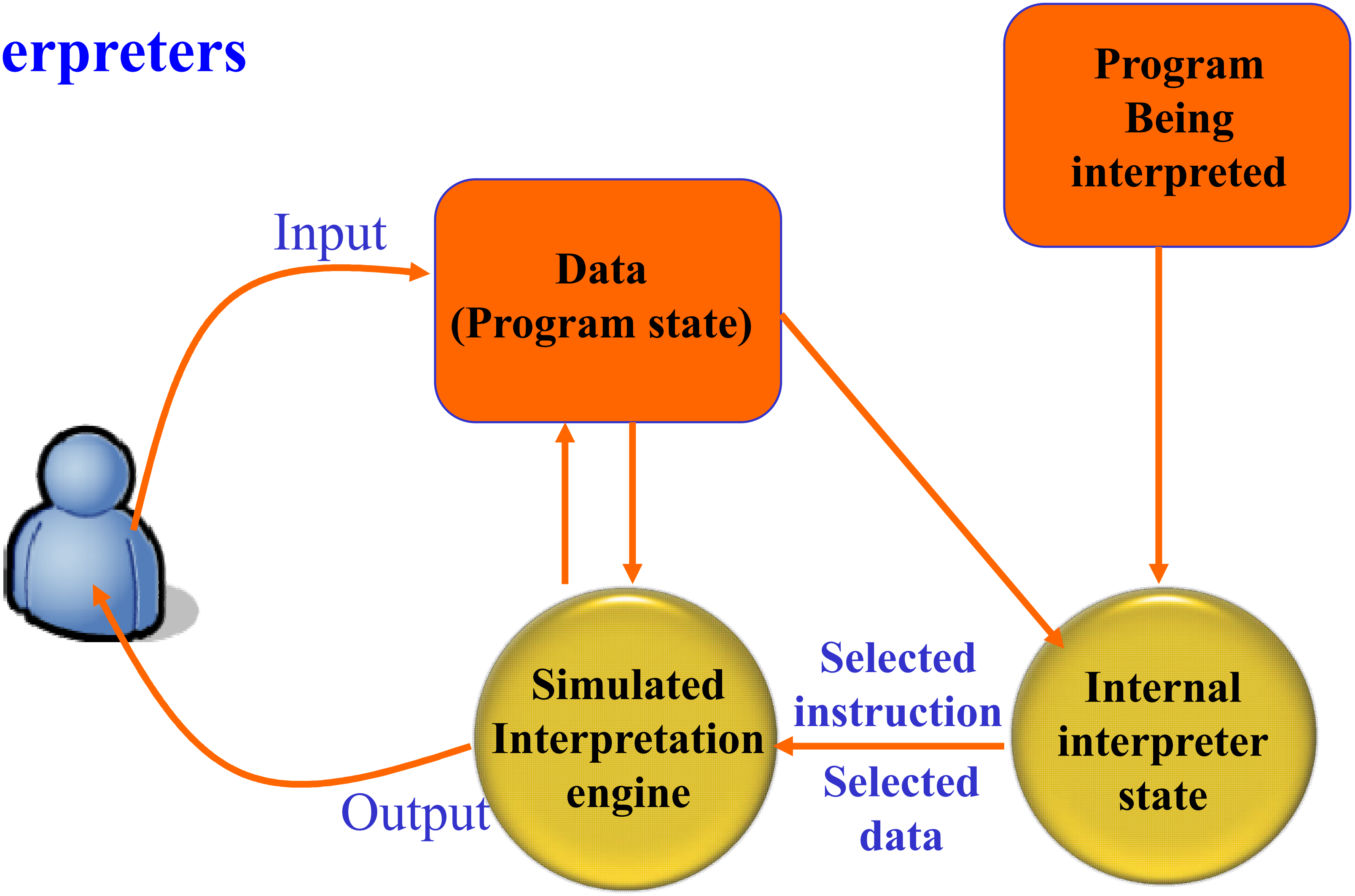
5.2.2 Architectural Styles and Strategies

Repositories

- Examples: Libraries; Large database; Search Engines.
- Advantage:
  - Modularity
  - Openness – the data representation is often made available.
- Disadvantage:
  - Openness – shared data must be in a form acceptable to knowledge sources.

5.2.2 Architectural Styles and Strategies

Interpreters



5.2.2 Architectural Styles and Strategies

Interpreters

- □ An interpreter takes a string of characters, and converts it into actual code that is then executed
- □ Always used to build virtual machine
  - □ Basic
  - □ Java virtual machine
  - □ 目前基于Web开发的各种脚本语言

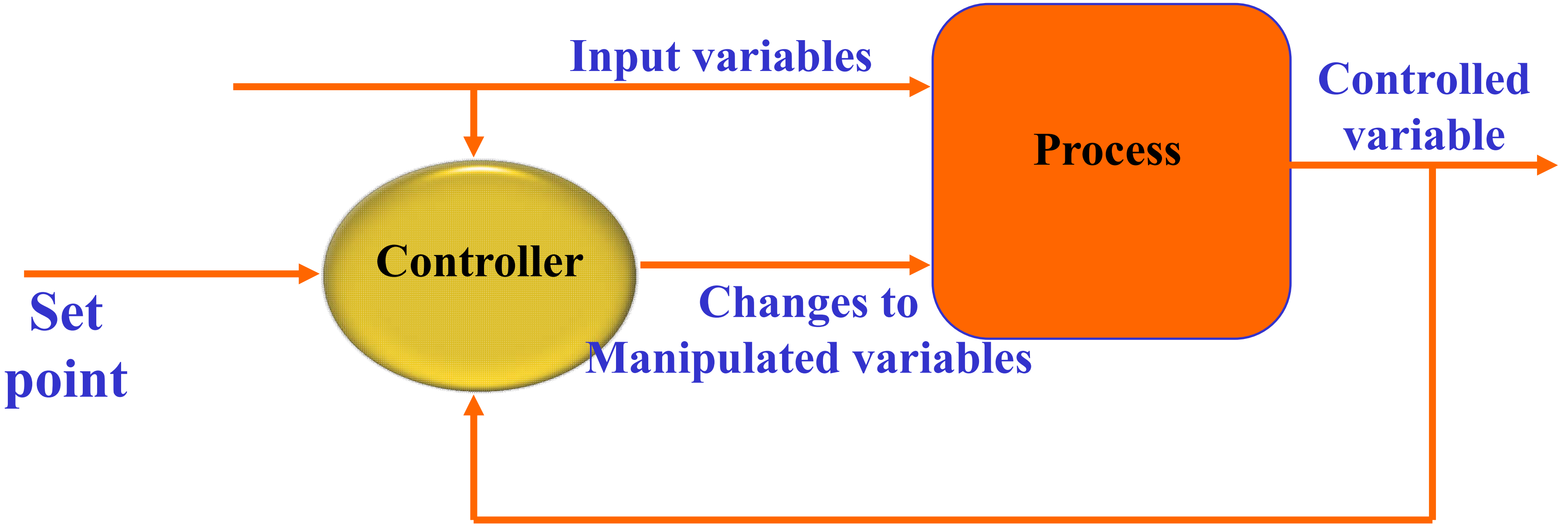
5.2.2 Architectural Styles and Strategies

Process control

- Process Control systems are very different from function- or object-based designs.
- Process control system are characterized not only by the ***type of component***, but also by the ***relationships*** that hold among them.
  - 核动力系统中对核燃料棒中原子分裂状态的控制；
  - 空调系统的温度控制
- The most common software-based control system involves a closed loop in one of two forms, feedback and feedforward.

5.2.2 Architectural Styles and Strategies

Process control

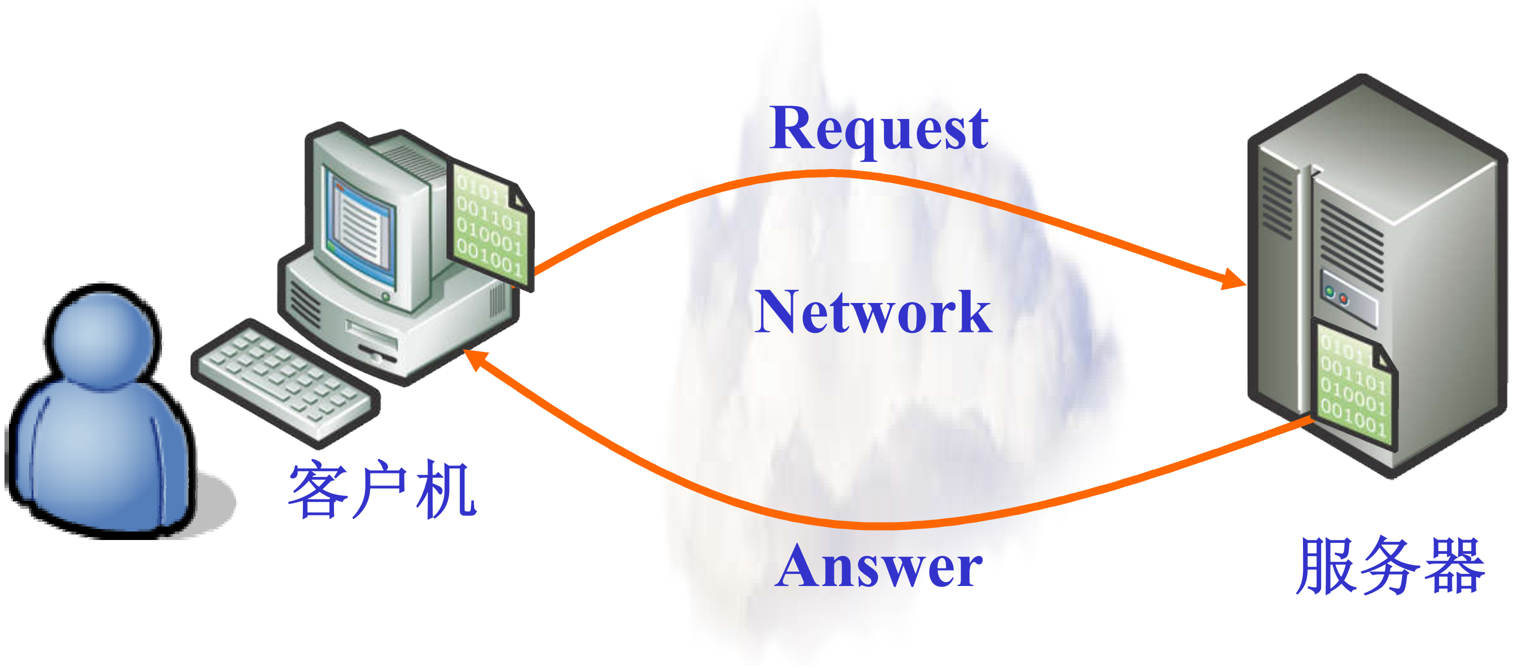


Feedback Loop

Feedforward Loop

5.2.2 Architectural Styles and Strategies

Client/Server → Browser/Server



当客户端软件变成为浏览器后，称之为B/S结构

5.2.2 Architectural Styles and Strategies

Client/Server

- Advantages
  - Most of data and computational power reside on the server
  - Multiple views of the same data
  - 软件部署难度降低
- Drawbacks
  - More sophisticated security, systems management and application development
  - 对网络流量有很强的依赖性

5.2.3 Important Design Issues

- There are many issues(问题) involved in creating a design:
  - What is best for the application
  - What is comfortable for the designer
  - What makes sense for the overall architecture
- Thus, no one style or method is best for every situation.

5.2.3 Important Design Issues

- □ Modularity and levels of abstraction模块性与抽象层次
- □ Collaborative design协作设计
- □ Designing the user interface设计用户界面
  - □ metaphors, mental model, navigation rules, look and feel比喻、智力模型、导航规则、外表与感知
  - □ cultural issues文化问题
  - □ user preferences用户爱好
- □ Concurrency并发
- □ Design patterns and reuse设计模式与复用

5.2.3 Important Design Issues

- We noted earlier that modularity is a characteristic of good design.
- In a modular design, the components have clearly defined inputs and outputs, and each component has a clearly stated purpose. So,
  - It is easy to examine each component separately from the others to determine whether the component implements it required tasks.
  - Modular components are organized in a hierarchy, as the result of decomposition or abstraction, so that we can investigate the system one level at a time
  - Modularity hides detail—***information hiding(信息隐藏).***

5.2.3 Important Design Issues

- We consider the top level to be the most abstract, and components are said to be arranged in ***levels of abstraction.***
- The levels of abstraction help us to understand the problem addressed by the system and the solution proposed by the design.
- By combining modular components with several levels of abstraction,
  - We can get several different views of the system.
  - Modularity provides the flexibility we need to understand what the system is to do.
  - It can allow us to understand that problem at increasing levels of detail.

5.2.3 Important Design Issues

Example of abstraction P211

1

```
Rearrange L in non-decreasing order
```

2

```
DO WHILE I is between 1 and (length of L)-1
  Set Low to index of smallest value in L(I), …, L (length of L)
  Interchange L(I) and L(LOW)
ENDDO
```

3

```
DO WHILE I is between 1 and (length of L) – 1
  Set LOW to current value of I
    DO WHILE J is between I+1 and (length of L) – 1
      if L(LOW) is greater than L(J)
        THEN set LOW to current value of J
      ENDIF
    ENDDO
  Set temp to L(LOW)
  Set L(LOW) to L(I)
  Set L(I) to TEMP
ENDDO
```

5.2.3 Important Design Issues - Collaborative design

- On most projects, the design is not created by one person. Rather, a team works collaboratively to produce a design, often by assigning different parts of the design on different people.
- Several issues must be addressed by the team, including:
  - ***Who*** is best suited to design each aspect of the system.
  - ***How*** to document the design so each team member understands the designs of others.
  - ***How*** to coordinate the design components so they work well as a unified whole.

5.2.3 Important Design Issues - Collaborative design

- The major problems in performing collaborative design are:
  - ***Differences*** in personal experience, understanding, and preference.
  - Behavior ***differences*** in groups from the way they would behave individually.
- It is important to view the group interaction in its cultural(文化) and ethical(伦理) contexts.
- Sidebar 5.3 The causes of design breakdown. **P212**

5.2.3 Important Design Issues - Collaborative design

□ Software design is both a collaborative and iterative (迭代) process.

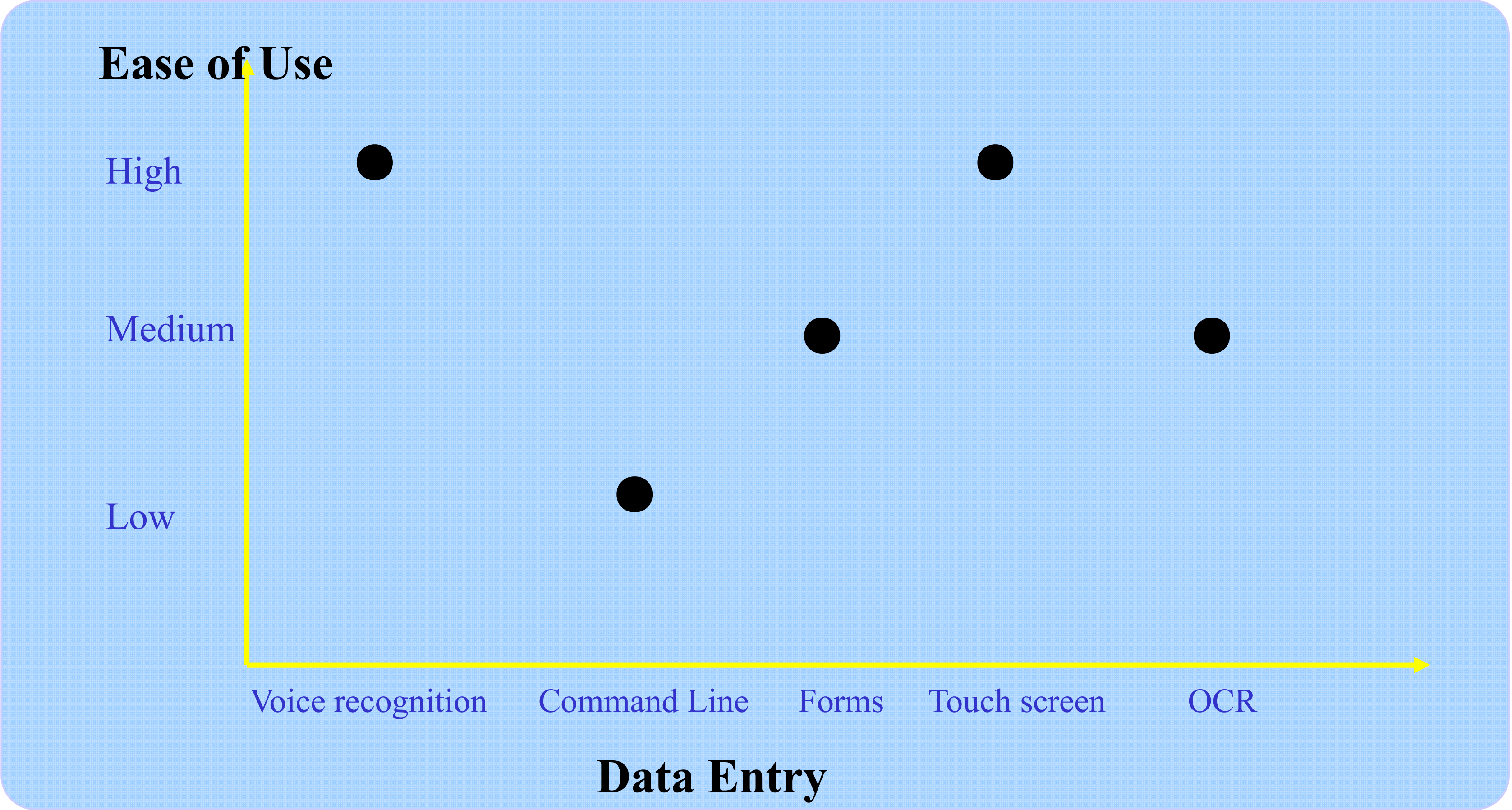
□ In building a software system, we are not just building a product; we are also building a shared understanding of the customers, the users, the application domain, the environment, and more.

□ The focus of our design efforts should be on revealing(揭示) as much about all of these aspects as we can.

5.2.3 Important Design Issues – User Interface design

- User interfaces can be tricky things to design, because different people have different styles of perceiving(感知), understanding, and working.
- Marcus(1993) points out that an interface should address several key elements(关键因素):
  - Metaphors(比喻)
  - A mental model(智力模型)
  - The navigation rules(导航规则) for the model
  - Look
  - feel

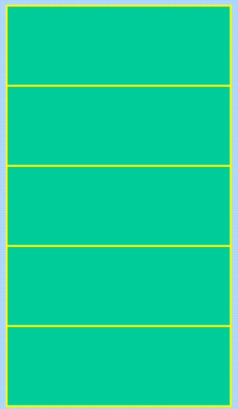
5.2.3 Important Design Issues – User Interface design



5.2.3 Important Design Issues – Concurrency

- In many systems, actions must take place concurrently (并发)rather than sequentially(串行).
  - 计算机、手机、**P217**
- One of the biggest problems with concurrent systems is to need to assure the ***consistency of the data shared*** among components that execute at the same time.

5.2.3 Important Design Issues – Concurrency



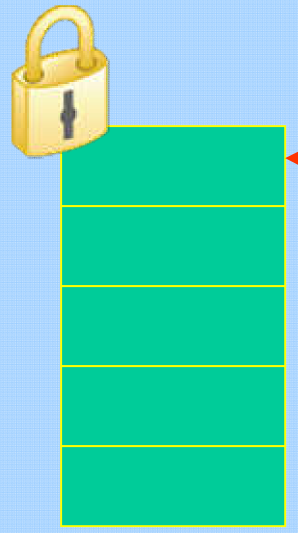
堆栈 X

Component 1	Component 2
1. Asks	
2. Informed	1. Asks
	2. Informed
3. Pops	
	3. Pops ✗

5.2.3 Important Design Issues – Concurrency

- Address(解决) concurrency conflicts(冲突) methods:
  - Timing
  - Synchronization
  - Process priority(优先级) schemes
- Synchronization同步: a method for allowing two activities to take place concurrently without their interfering with each other.
- Mutual exclusion互斥: a popular way to synchronize processes; it makes sure that when one process is accessing a data element, no other process can effect that element.

5.2.3 Important Design Issues – Concurrency



堆栈 X

Component 1	Component 2
1. Asks 2. Informed 3. Pops and Lock	1. Asks 2. Informed 3. Pops ? **Waiting** 4. Notified 5. Pops

Another Component

5.2.3 Important Design Issues - design pattern & reuse

- □ We want to take advantage of the commonality (共性) among systems, so we need not develop each “from scratch”. (从零开始的开发)
- □ A popular way of identifying the commonalities is to look for design patterns.
- □ We can reuse the patterns, as well as code, tests, and documents related to them, when we build the next similar system.
- □ ***Design pattern*** definition: **P219**

5.3 Characteristics of good design

- Characteristics of good design:
  - Ease of understanding
  - Ease of implementation
  - Ease of testing
  - Ease of modification
  - Correct translation from the requirements specification

5.3 Characteristics of good design

- Component independence
  - Coupling(耦合)
  - Cohesion(内聚)
- Exception identification and handling
- Fault prevention and tolerance(容错)
  - Active(主动的)
  - Passive(被动的)

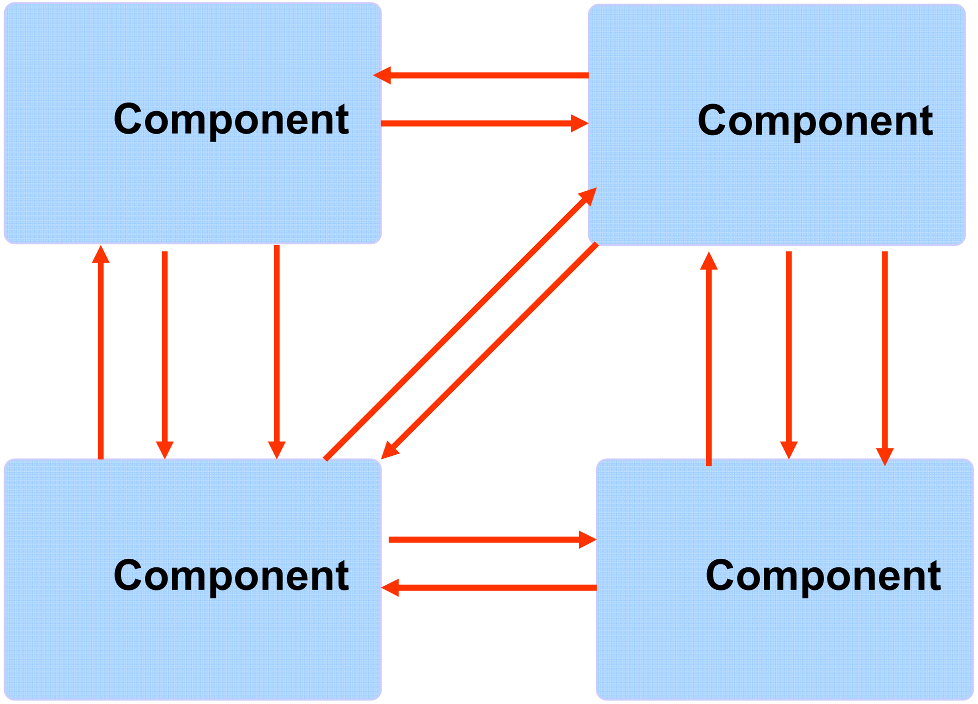
5.3 Characteristics of good design

- 组件独立是模块化、抽象、信息隐蔽和局部化的直接结果。
- 含义：一个模块具有独立功能而且和其它模块之间没有过多的相互作用
- 意义：独立的模块容易开发（规模小，接口简单）；独立的模块容易测试和维护；有效阻断错误传播（ Ripple effect“涟漪效应”）
- 度量标准：内聚和耦合（由C. Myers，Constantine和Yourdon等人提出）

5.3 Characteristics of good design

Component independence

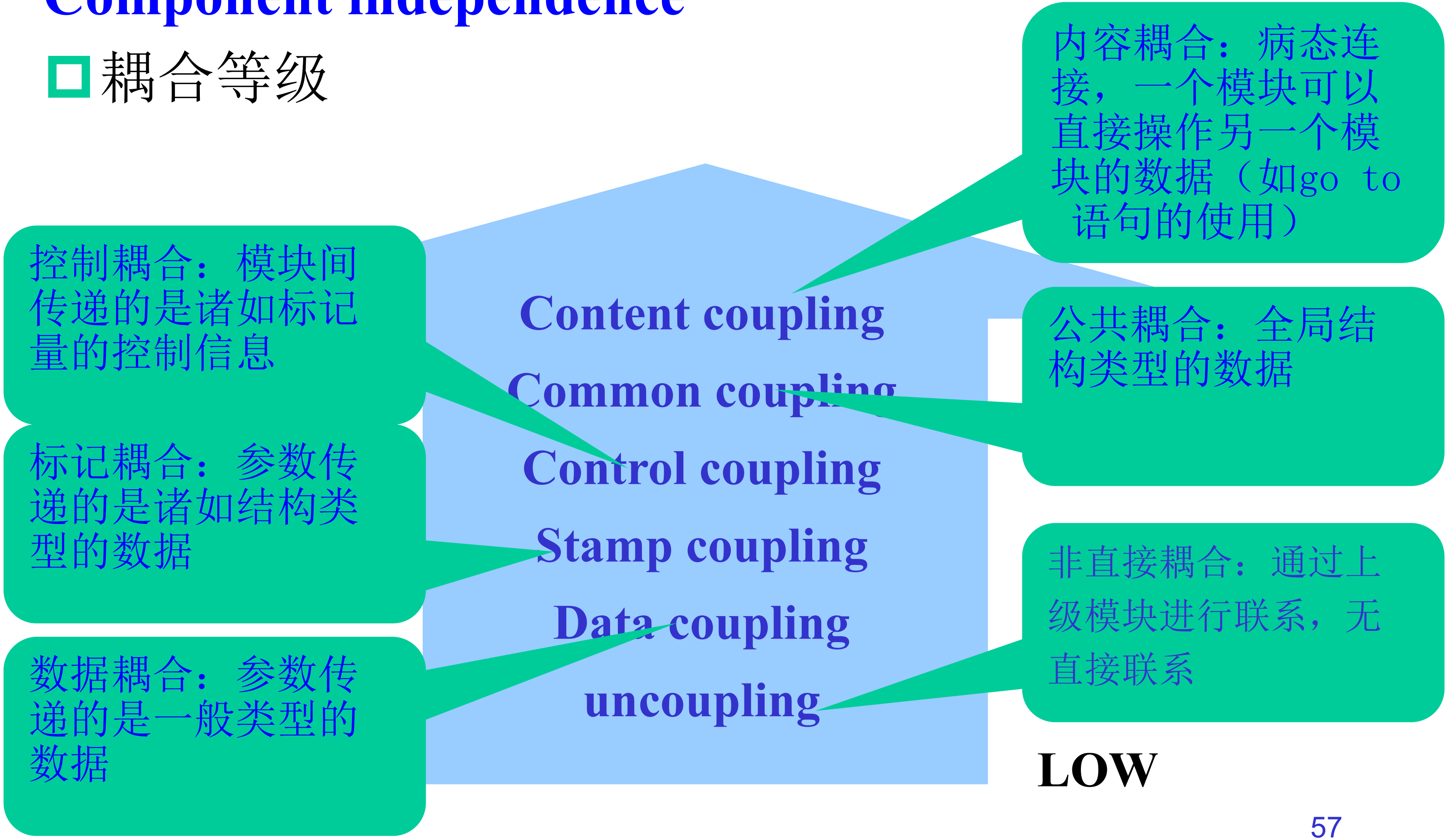
- Coupling (耦合)
  - Uncoupled
  - Loosely coupled
  - Highly coupled



5.3 Characteristics of good design

Component independence

- 耦合等级



内容耦合：病态连接，一个模块可以直接操作另一个模块的数据（如go to 语句的使用）

控制耦合：模块间传递的是诸如标记量的控制信息

公共耦合：全局结构类型的数据

标记耦合：参数传递的是诸如结构类型的数据

数据耦合：参数传递的是一般类型的数据

非直接耦合：通过上级模块进行联系，无直接联系

Content coupling

Common coupling

Control coupling

Stamp coupling

Data coupling

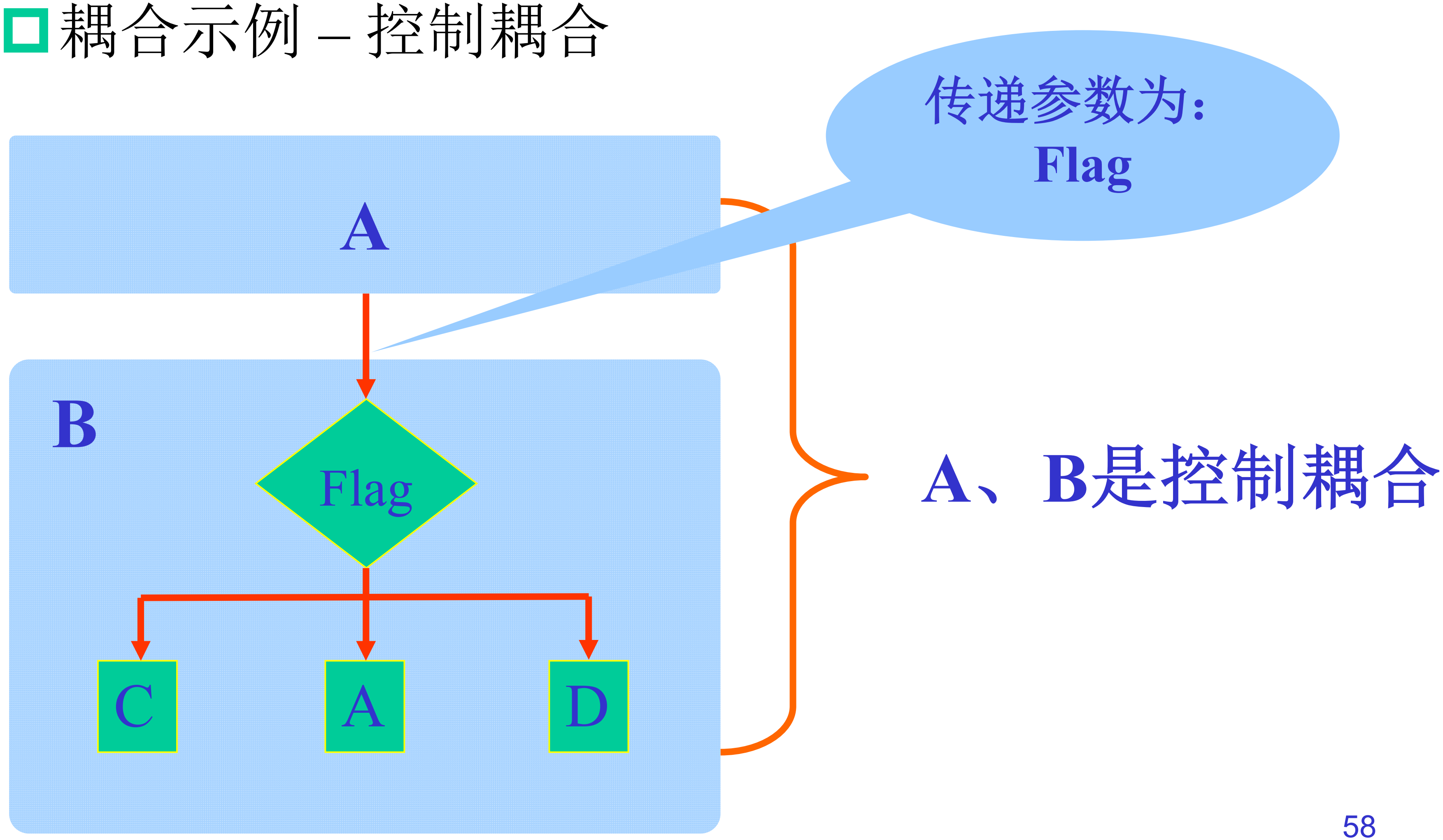
uncoupling

LOW

5.3 Characteristics of good design

Component independence

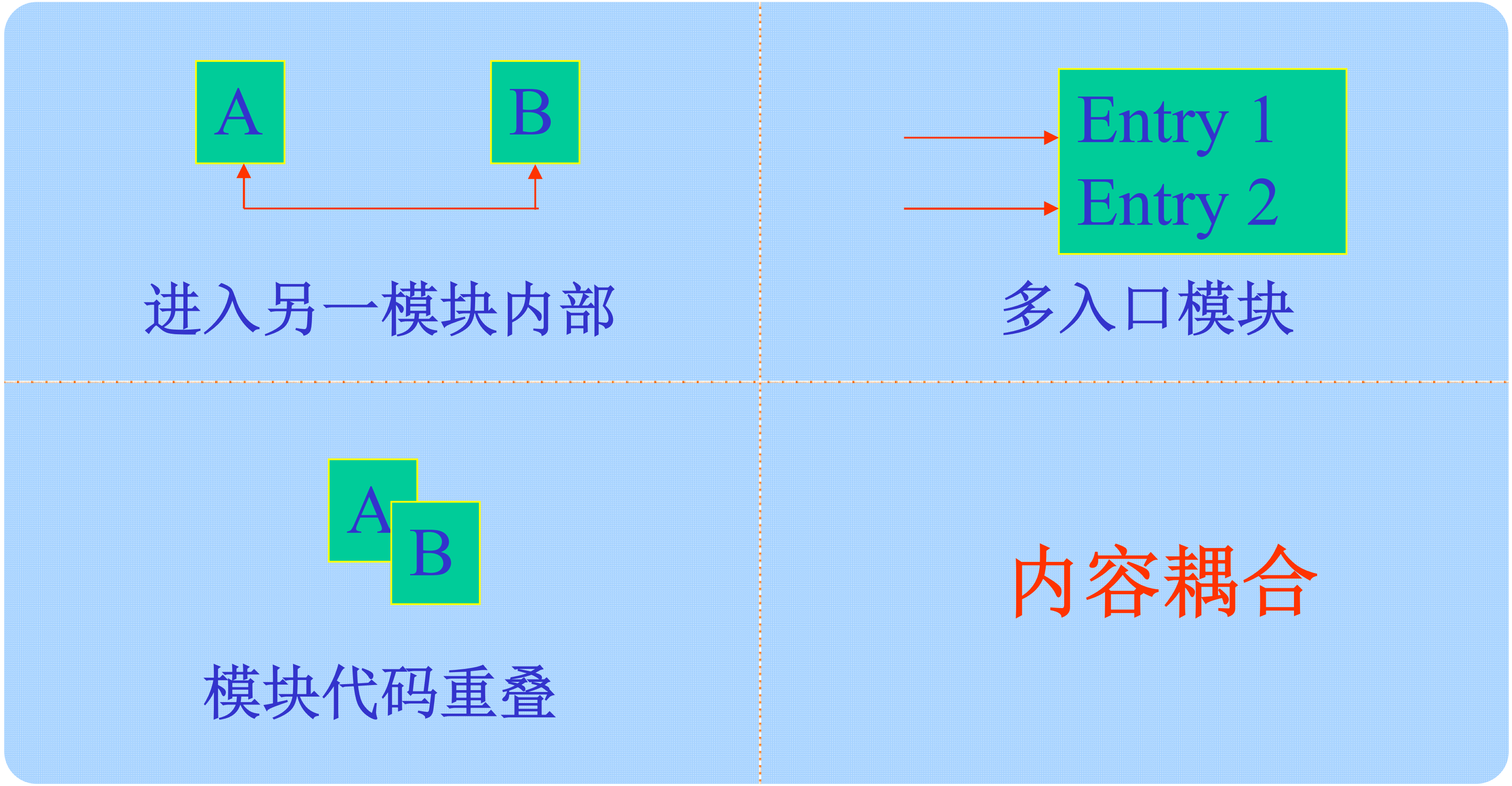
- 耦合示例 – 控制耦合



5.3 Characteristics of good design

Component independence

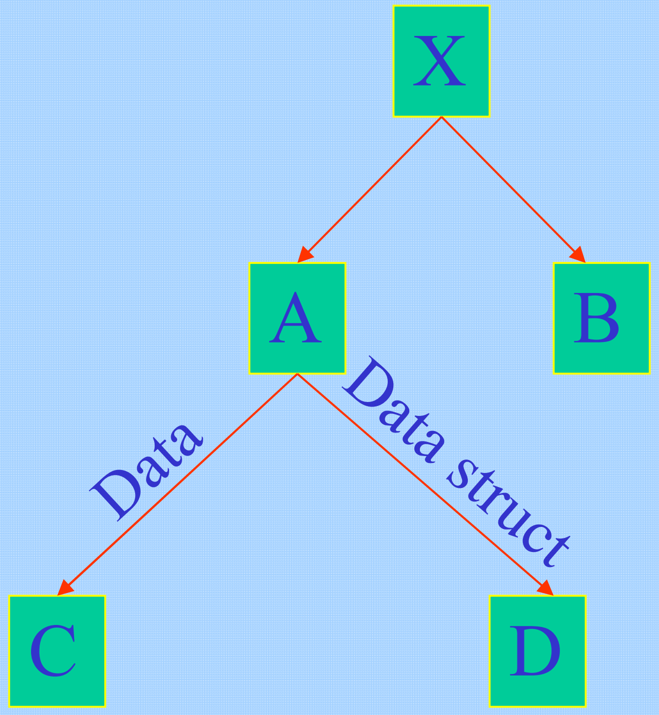
□耦合示例 – 内容耦合



5.3 Characteristics of good design

Component independence

□ 耦合示例



AB为非直接耦合

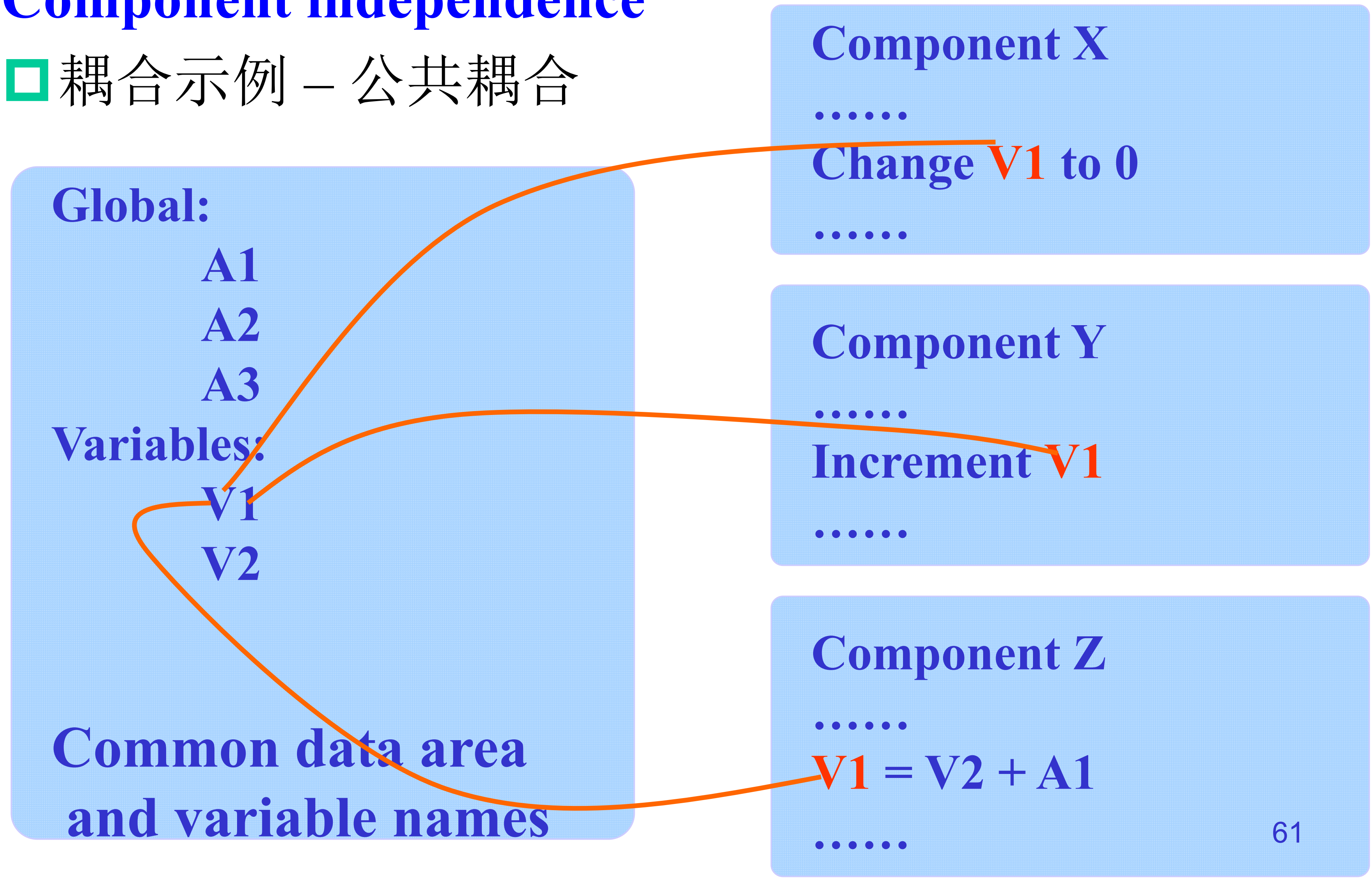
AC为数据耦合

AD为标记耦合

5.3 Characteristics of good design

Component independence

- 耦合示例 – 公共耦合



Global:
A1
A2
A3
Variables:
V1
V2

Common data area and variable names

Component X
……
Change V1 to 0
……

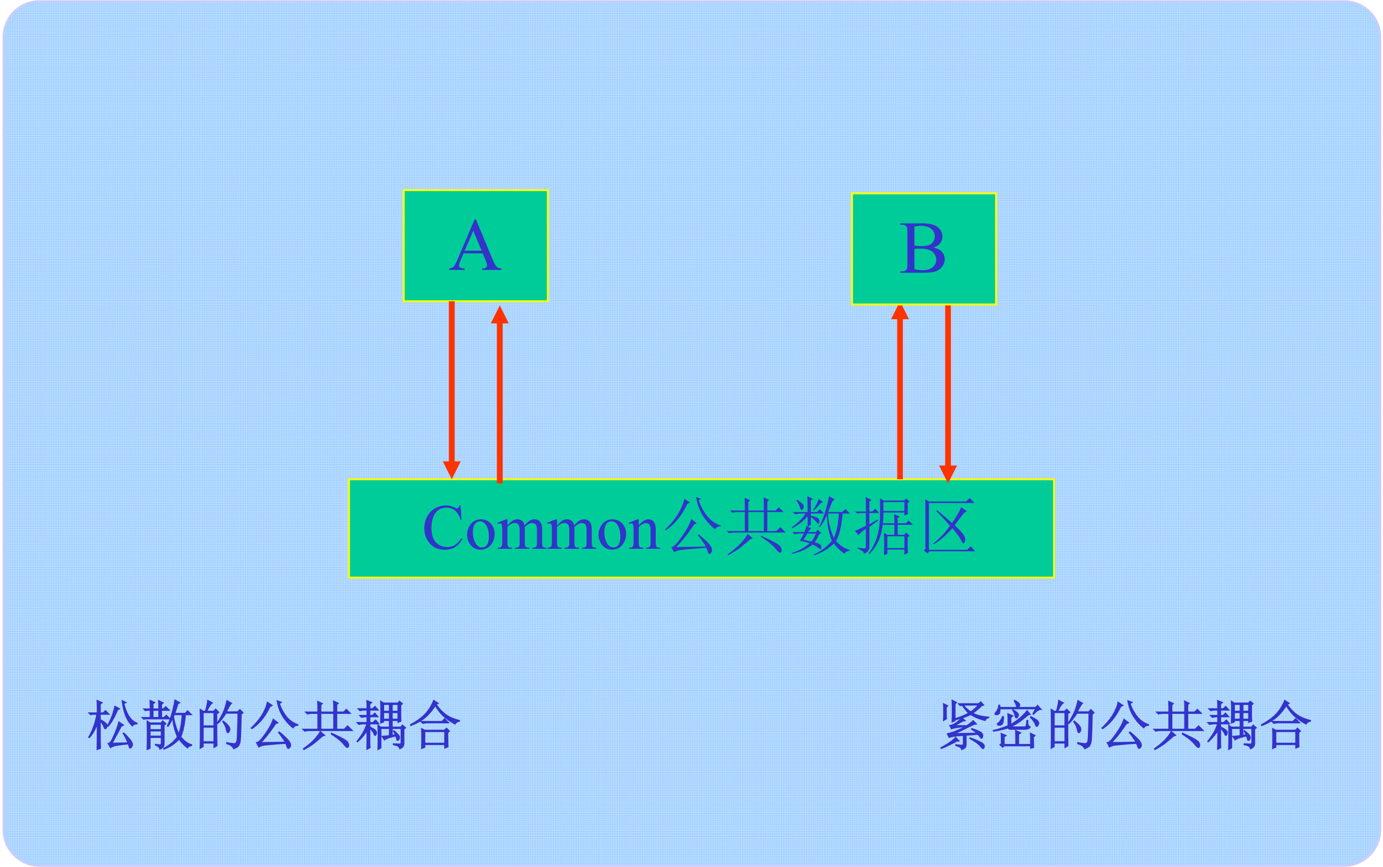
Component Y
……
Increment V1
……

Component Z
……
V1 = V2 + A1
……

5.3 Characteristics of good design

Component independence

- 耦合示例 – 公共耦合



5.3 Characteristics of good design

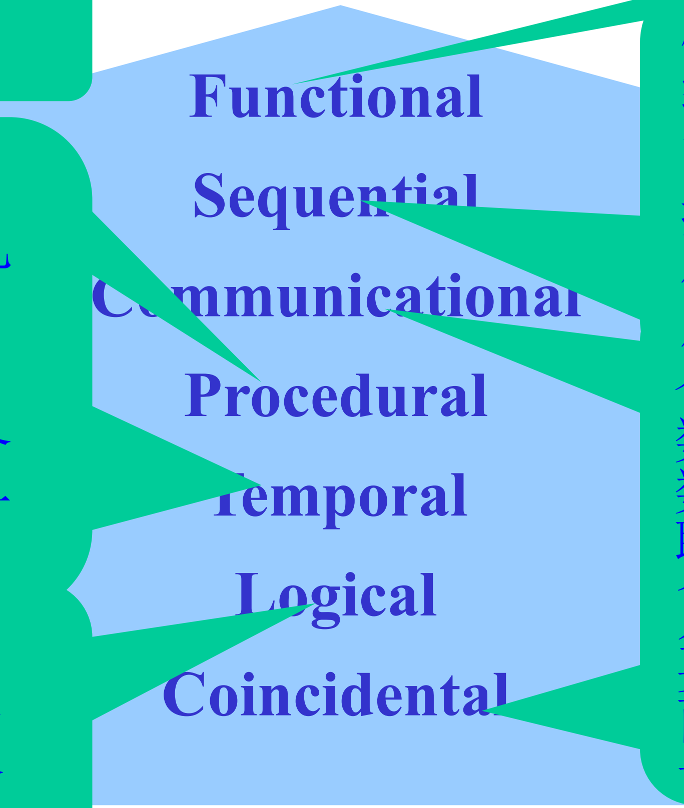
Component independence

- Cohesion(内聚)
  - 块内联系或模块强度，指模块内各个成分（元素）彼此结合的紧密程度，即模块内部的聚合能力。
  - “理想的模块仅仅做一件事”。

5.3 Characteristics of good design

Component independence

□ 内聚等级



功能性内聚：一个功能一个模块，块内各成分属于一个整体

过程内聚：块内成份必须按照特定次序执行

顺序内聚：模块内各个组成部分都是与一个功能密切相关，并是顺序执行的。一般是一个成份的输出就是下一个成份的输出

时间内聚：因执行时，一样或顺序排列而把几个任务安排一个模块，如把“变量赋初值”、“打开文件”等完成各种初始化任务安排在一个模块

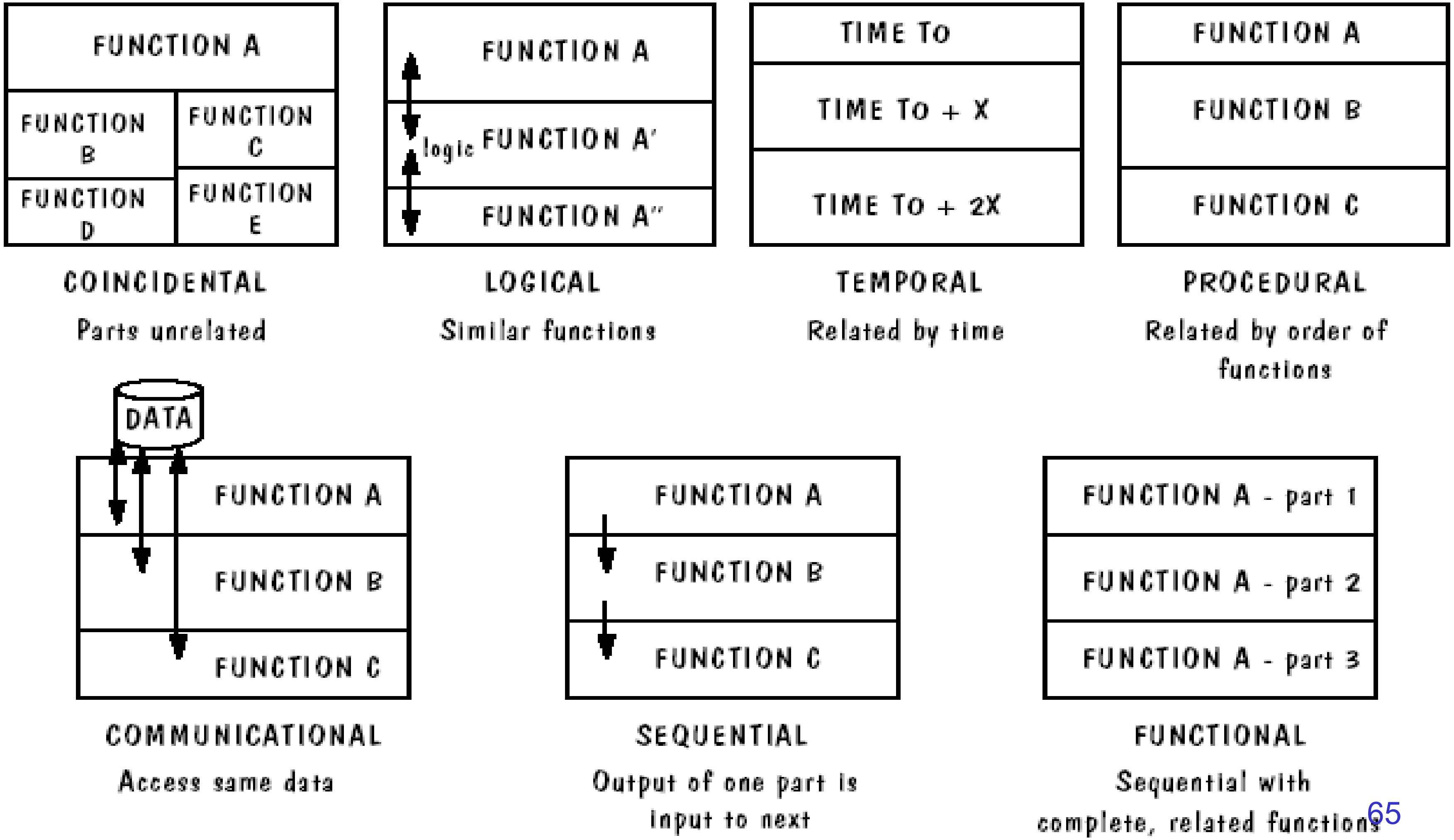
个成份都使用同一输入数据，或产生同一输出数据，即借公用数据而联系在一起

逻辑内聚：块内仕务间在逻辑上相似或相同，例如求某班的平均分和最高分，因其输入和输出相同而安排在一个模块内完成。

[illegible]系松散，互不相关。主要是为了避免重复书写而把重复的代码集成到一个模块内。

5.3 Characteristics of good design

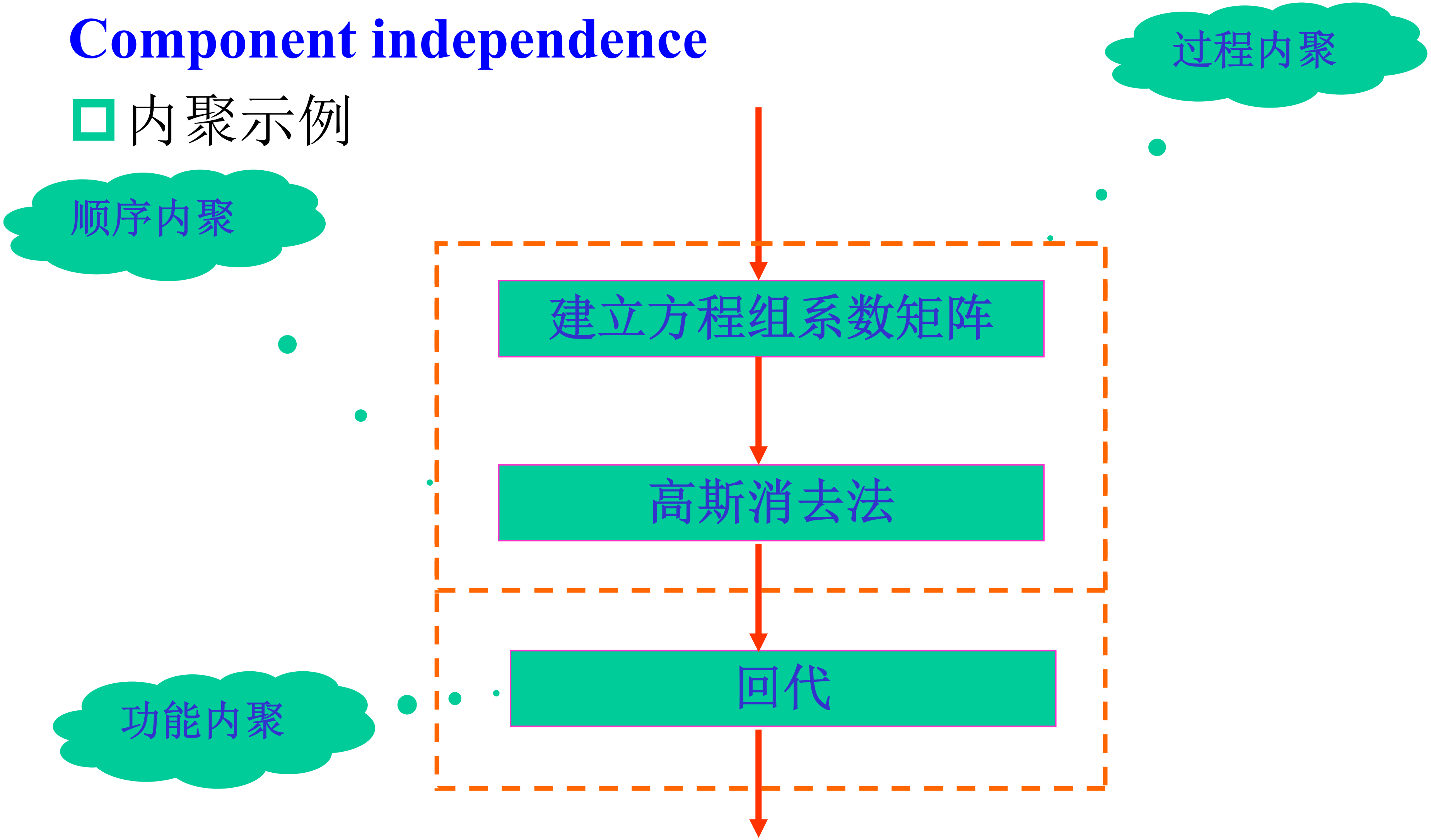
Component independence 内聚等级



5.3 Characteristics of good design

Component independence

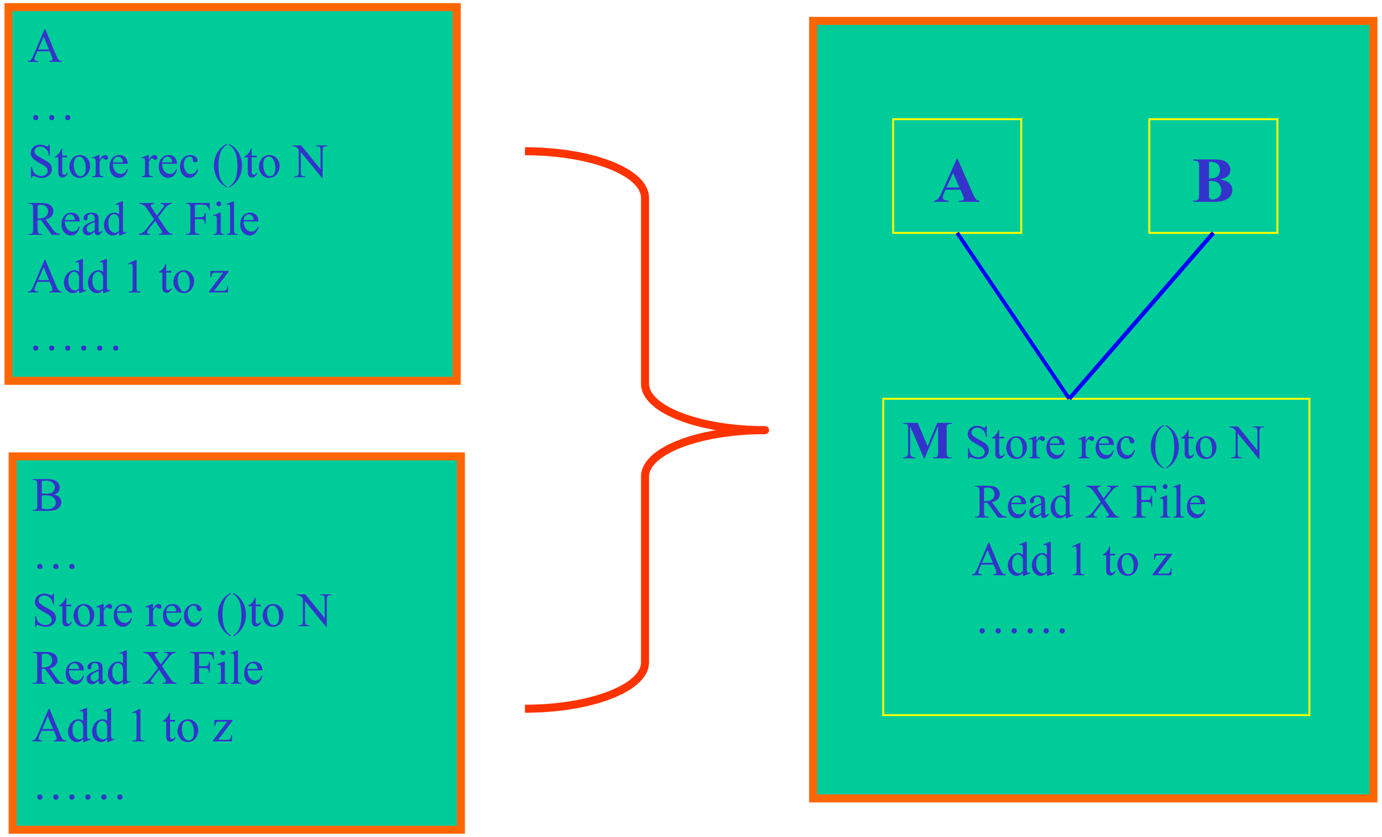
- 内聚示例



5.3 Characteristics of good design

Component independence

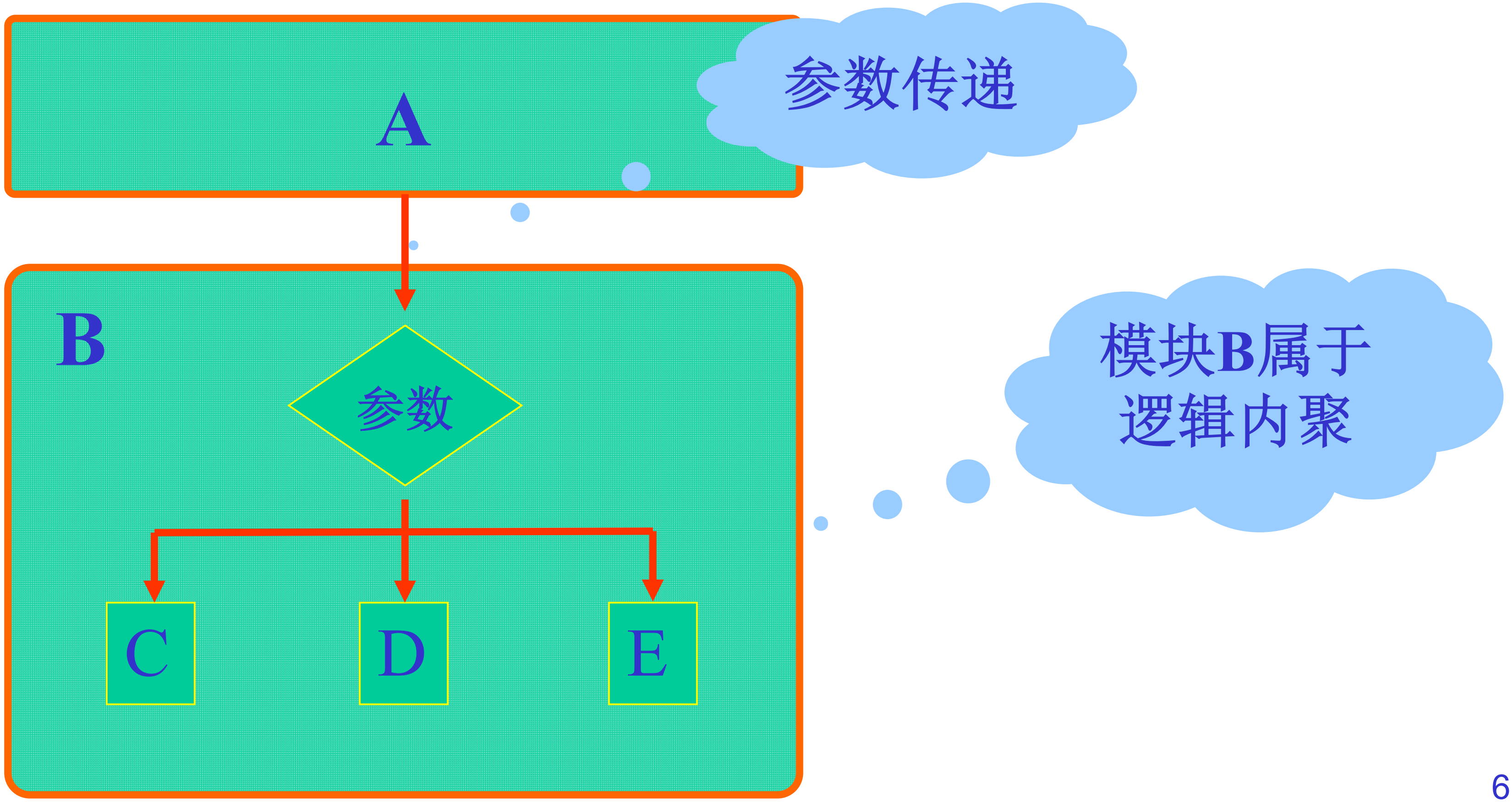
- 内聚示例 – 偶然性内聚



5.3 Characteristics of good design

Component independence

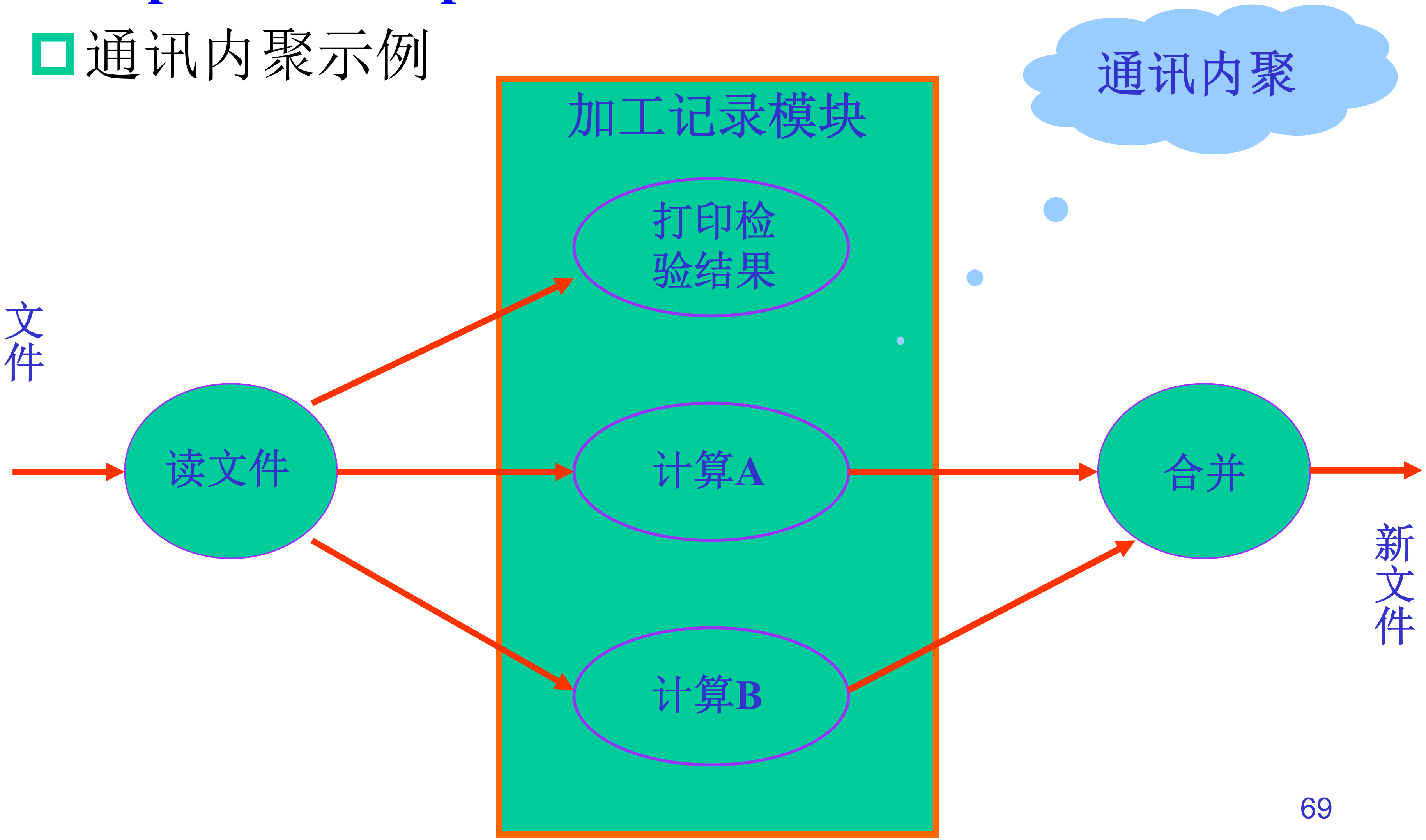
□ 内聚示例 – 逻辑内聚



5.3 Characteristics of good design

Component independence

- 通讯内聚示例



5.3 Characteristics of good design

Component independence

- 启发式规则
  - 提高模块独立性
  - 设计规模适中的模块
  - *深度*、*宽度*、*扇入*、*扇出*适中
  - 模块的作用域应该在控制域之内
  - 降低接口复杂性
  - 设计单入口和单出口的模块
  - 设计功能可以预测的模块

说明：启发式规则是一种经验规律，对改进设计和提高软件质量具有重要的参考价值，但不要过分拘泥于这些规则。

5.3 Characteristics of good design

Component independence

- 启发式规则 – 提高模块独立性
  - 模块独立性是划分模块的最高准则。
  - 高内聚，尽量一个模块一个功能；
  - 低耦合，避免“病态连接”；
  - 降低接口的复杂程度；
  - 综合考虑模块可分解性、模块可组装性、模块可理解性、模块连续性和模块保护（因修改错误而引起的副作用被控制在模块的内部）等。

5.3 Characteristics of good design

Component independence

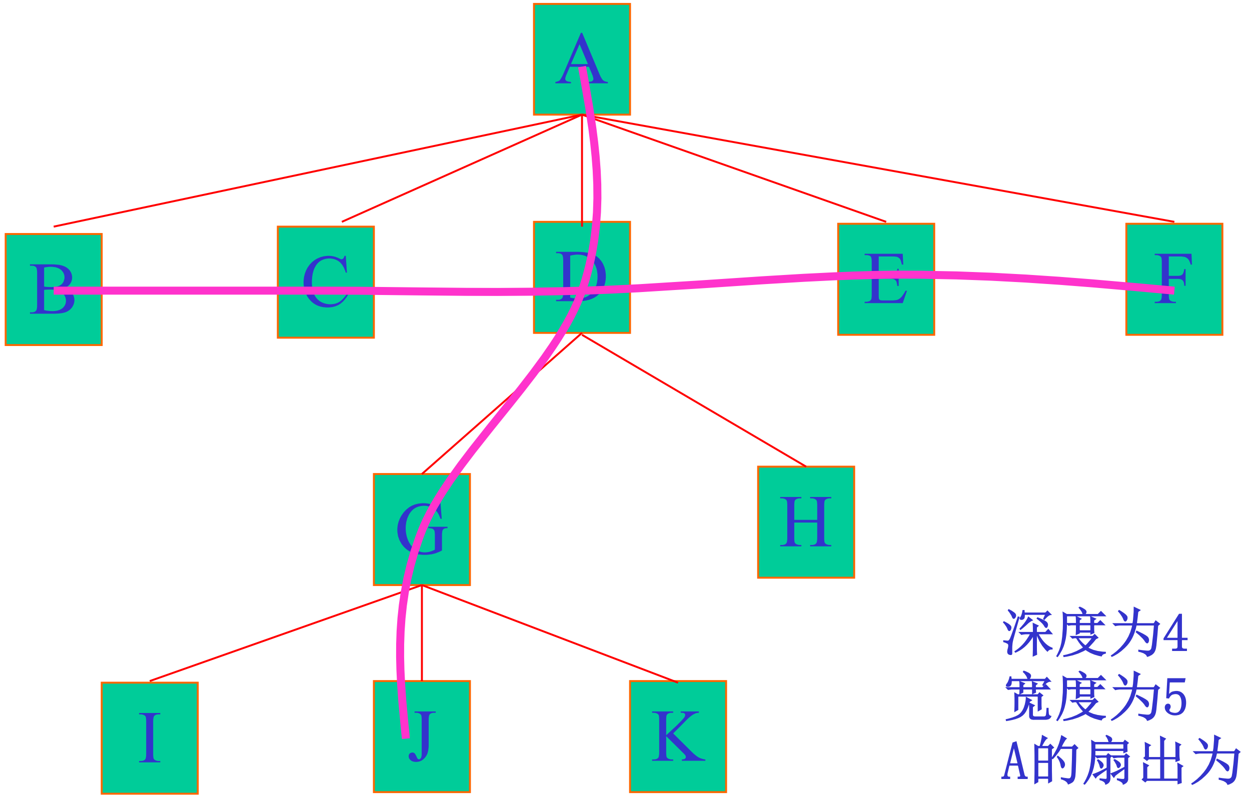
- 启发式规则 – 设计规模适中的模块
  - W. M. Weinberg的研究表明：如果一个模块长度超过30条语句，其可理解性将迅速下降；
  - F. T. Baker : 最好控制在50行左右，能够打印在一张纸上。
  - 由于模块独立性是最高原则，对于一个设计合理的功能性模块，即使长达千句或小到几行，也是允许的。
  - 分解模块不应该降低模块独立性。

For example

5.3 Characteristics of good design

Component independence

- 启发式规则 – 深度、宽度、扇入、扇出适中



深度为4
宽度为5
A的扇出为5，扇入为0
K的扇出为0，扇入为1

5.3 Characteristics of good design

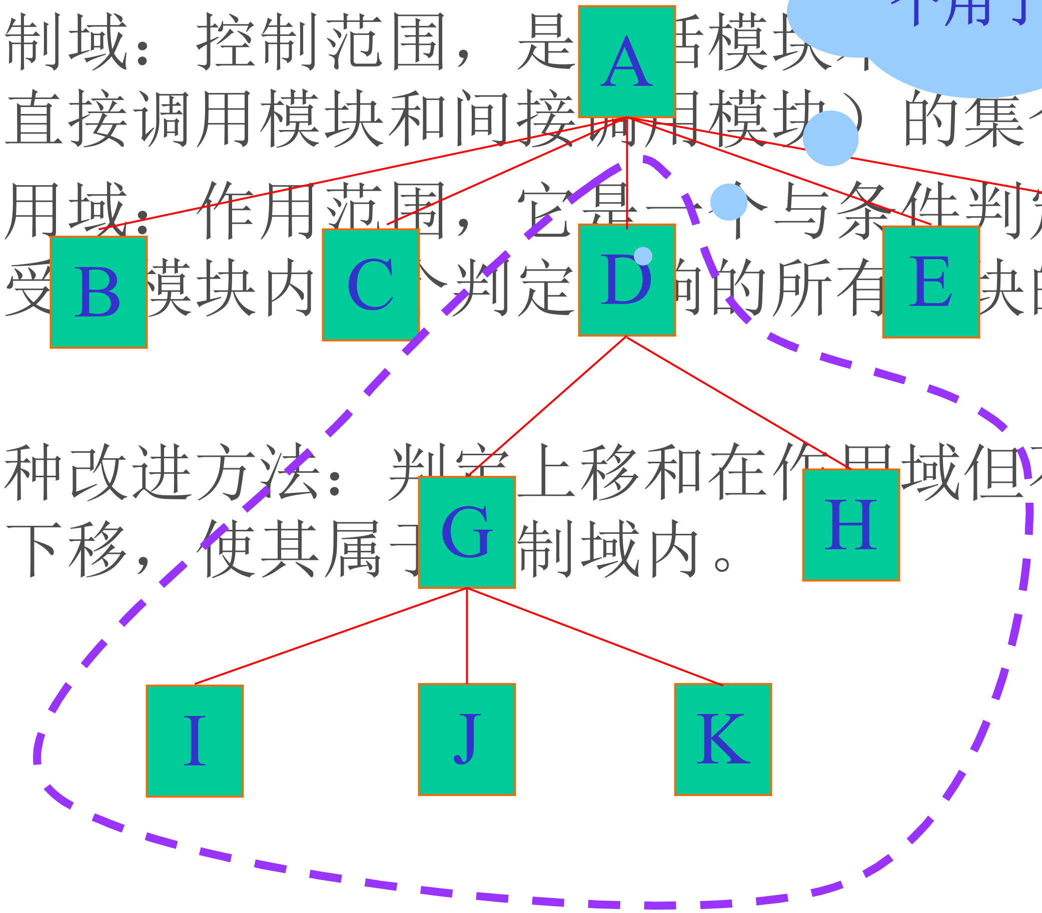
Component independence

- 启发式规则 – 深度、宽度、扇入、扇出适中
  - 深度：软件结构中控制的层数。一般而言它与系统的复杂度和系统大小直接对应。
  - 宽度：软件结构中同一个层次上的模块总数的最大数。
  - 扇出：一个模块直接控制（调用）的模块数目。 扇出过大说明模块过分复杂；过小也不好，不利于系统平衡分解，3到9为宜。
  - 扇入：一个模块的扇入是指直接控制该模块的模块数目。扇入越大说明共享该模块的上级模块越多。
  - 整个系统结构呈现“椭圆外型”。

5.3 Characteristics of good design

Component independence

- 启发式规则 – 作用域应在控制
  - 控制域：控制范围，是 括模块 模块（直接接调用模块和间接调用模块）的集合。
  - 作用域：作用范围，它是一个与条件判定相联系的概念。是受 模块内 个判定 响的所有 块的集
  - 两种改进方法：判定上移和在作用域但不在控制域的模块下移，使其属于控制域内。



D中有判定条件影响到E。通常是一个用于判定的变量

5.3 Characteristics of good design

Component independence

- □ 启发式规则 – 降低模块间接口复杂性
  - □ 尽量少使用go to语句，避免病态连接和内容耦合。
  - □ 注意全局变量的使用，控制外部耦合和公共耦合的使用。
  - □ 将数据结构的传递改成数据传递，例如：求一元二次方程根的模块quad_root(table, x)中，利用系数数组table和根数组x进行参数传递。如果将其改为直接的系数和根传递，即quad_root(a, b, c, x)，则特征耦合→数据耦合。

5.3 Characteristics of good design

Component independence

- 启发式规则 – 设计单入口和单出口的模块
  - 符合结构化程序设计的思想
  - 应避免病态连接和内容耦合。
  - “一个功能一个模块”→提高软件的可读性和可理解性。
  - 有效阻断“涟漪效应(ripple effect)”→提高软件的可靠性和可维护性。

5.3 Characteristics of good design

Component independence

- 启发式规则 – 设计功能可预测的模块
  - “可以预测”—模块的输入和输出之间的关系比较简单。
  - 功能可以预测的模块：如果一个模块可以当作一个“黑盒子(Black Box)”来对待，对于该模块的输入数据来说，可以在不考虑其内部处理细节的情况下生成输出数据。
  - 模块功能应该可以预测，但不要过分受其局限。

5.3 Characteristics of good design

Exception identification and handling

- □ Typical exceptions 典型的异常
  - □ Failure to provide a service 无法提供某种服务
  - □ Providing the wrong service or data 提供了错误的服务或数据
  - □ Corrupting data 破坏性的数据
- □ Three ways of handle exceptions 三种方法
  - □ Restoring and retrying 恢复系统＋重试
  - □ Restoring and correcting 恢复系统＋改正
  - □ Restoring and reporting 恢复系统＋报告
- □ Defensive designing is not easy

5.3 Characteristics of good design

Fault prevention and tolerance(容错)

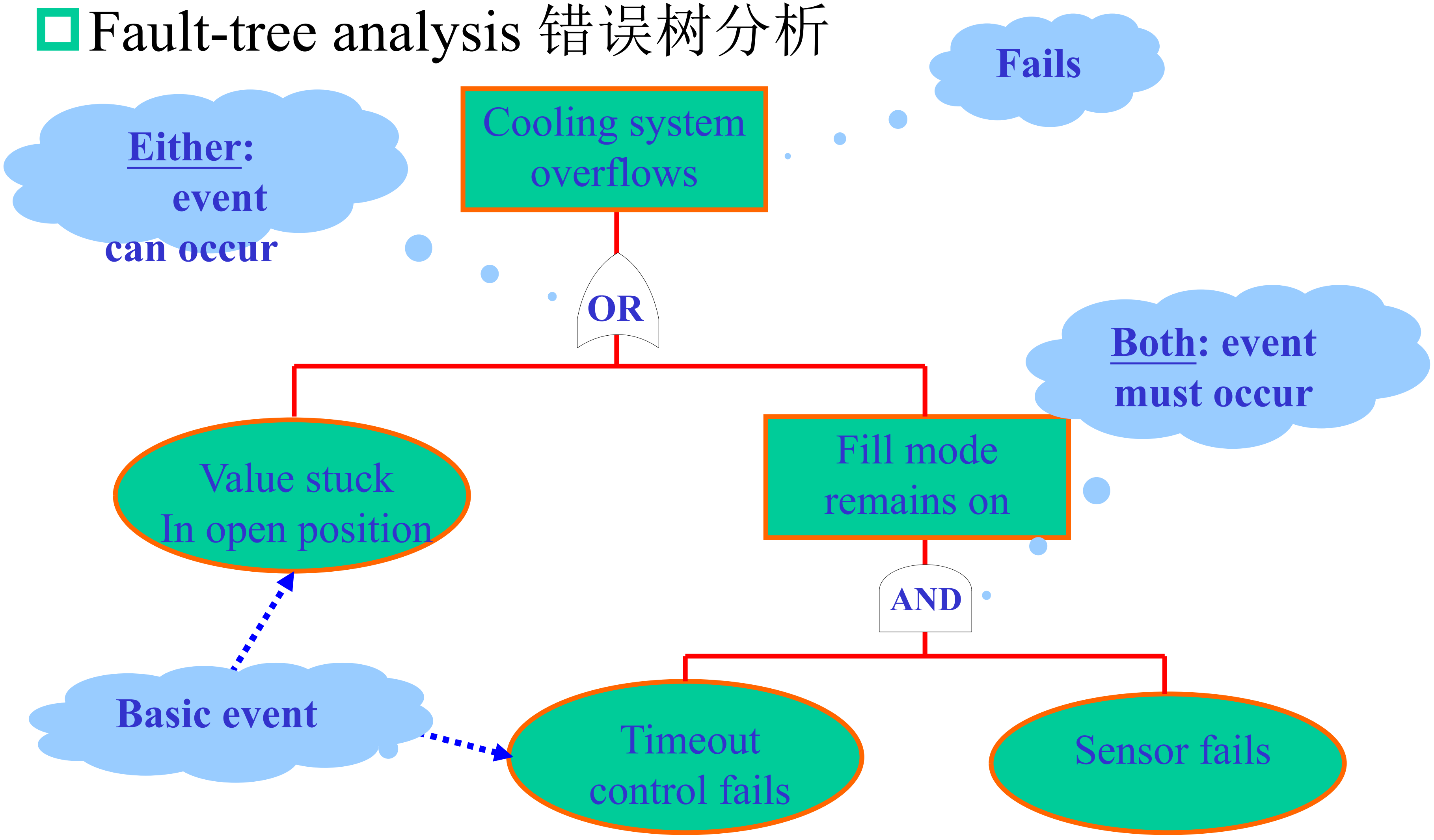
- Fault prevention and tolerance(容错)
  - Active Fault Detection – Mutual Suspicion Policy
  - Fault Correction – Windows的错误报告
  - Fault Tolerance

5.4 Techniques for Improving Design

- □ Reducing complexity 降低复杂度 P231
- □ Design by contract 通过契约设计 P233
- □ Prototyping design 原型化设计 P235
- □ Fault-tree analysis 错误树分析 P236

5.4 Techniques for Improving Design

- Fault-tree analysis 错误树分析



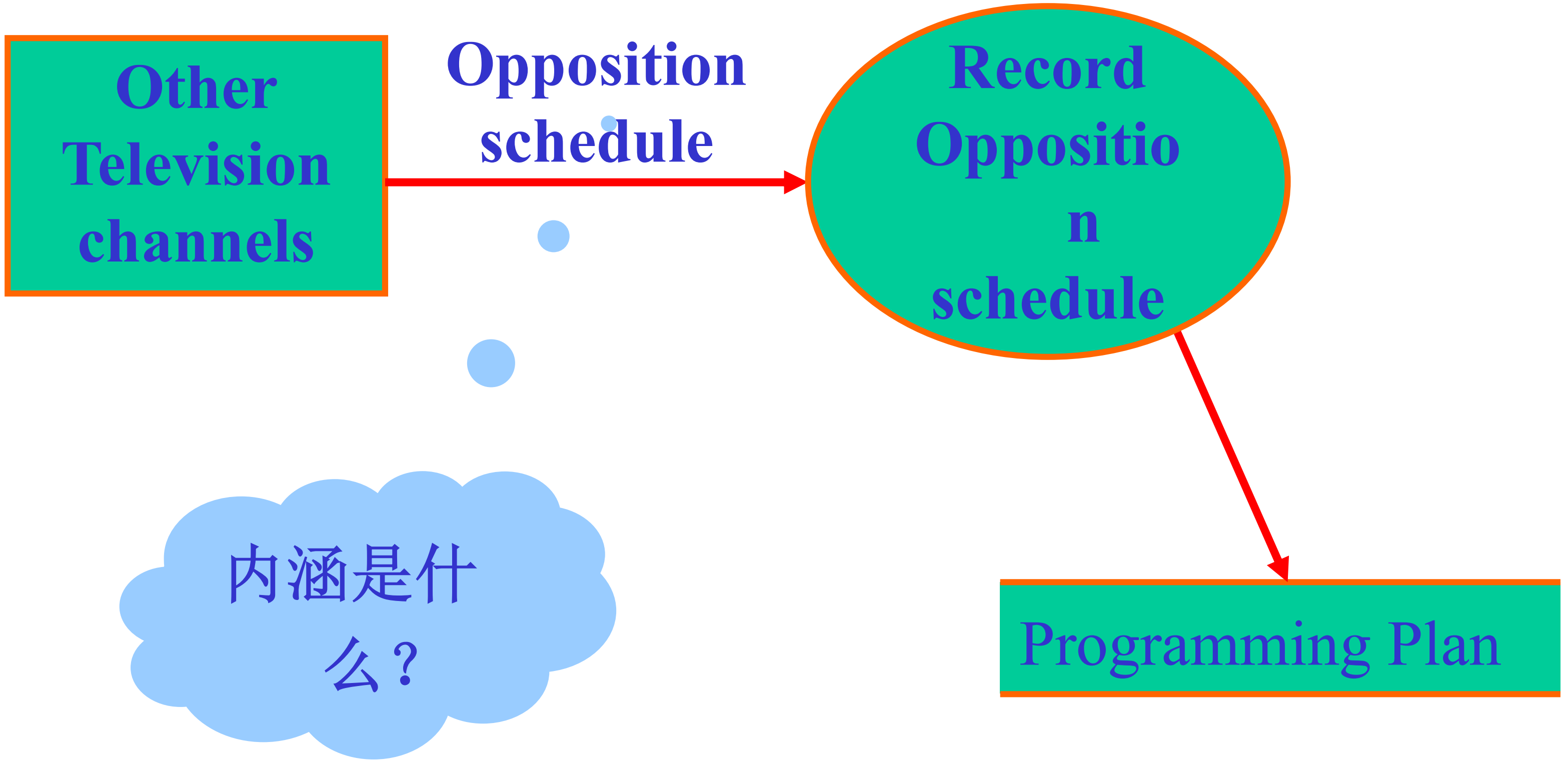
5.5 Evaluating and Validating Design

- □ Mathematical validation 数学确认
- □ Measuring design quality 测量设计质量
- □ Comparing designs 比较设计 P241
  - □ one specification, many designs一个规格说明，多个设计
  - □ comparison table比较表
- □ Design reviews设计评审

5.6 Information system example

- Design can be documented in a variety of ways:
  - Formal languages
  - State machine
  - Data flow diagrams
  - Data dictionaries
  - Object-oriented approaches
  - Many other available notations and techniques
- It is important to choose the technique or notation.

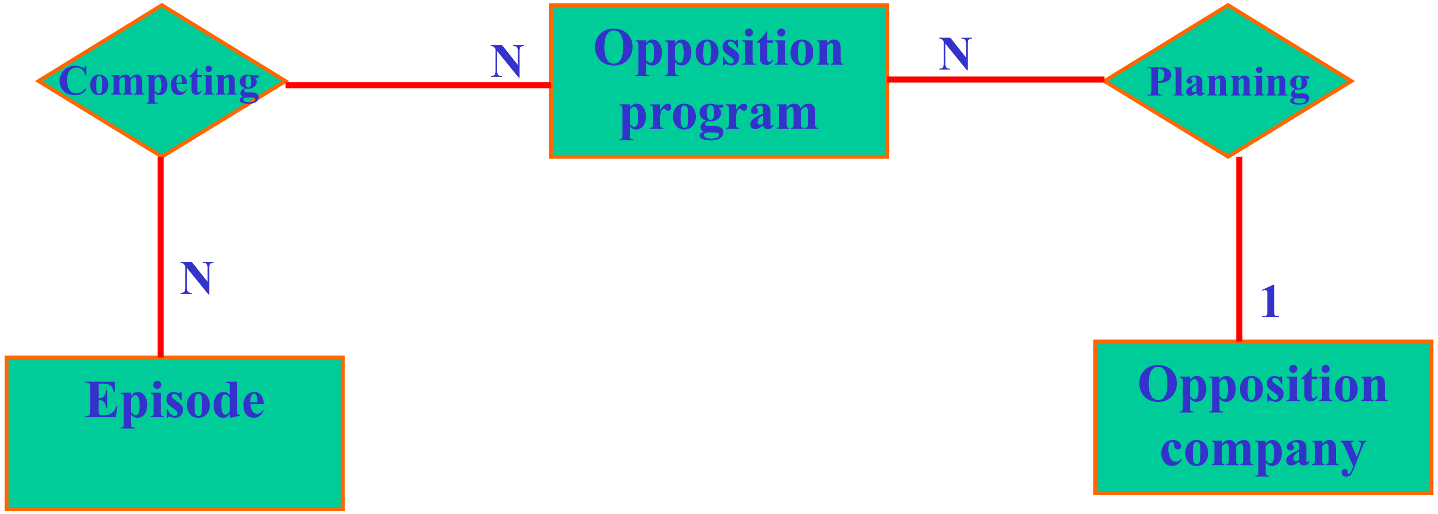
5.6 Information system example



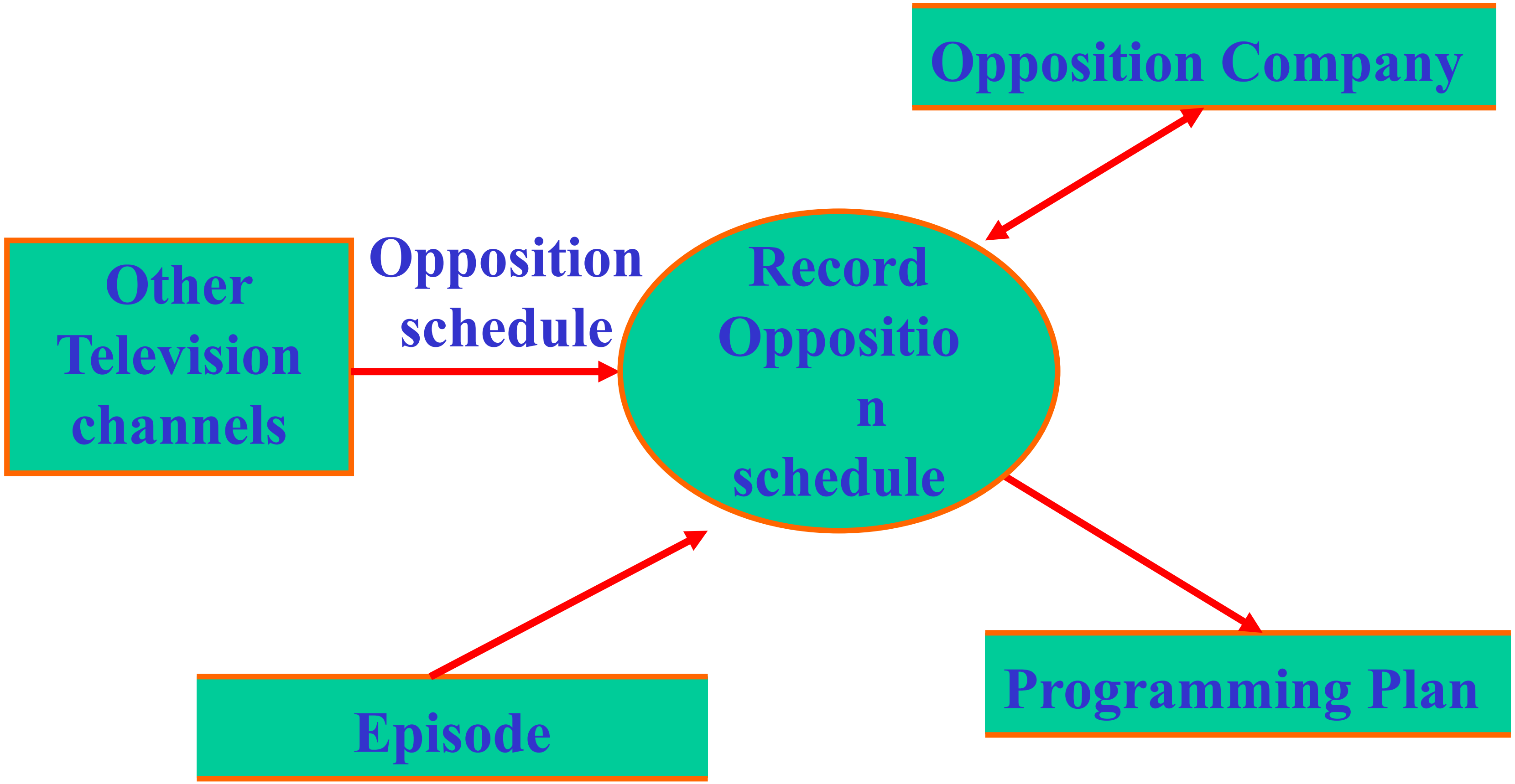
5.6 Information system example

Opposition schedule

Opposition schedule = * Data flow *
Television company name
+ {Opposition transmission date
+ Opposition transmission time
+ Opposition program name
+ (Opposition predicted rating) }



5.6 Information system example



5.6 Information system example

```
Input: Opposition schedule
For each television company name, create Opposition company.
    For each Opposition schedule,
        Locate the episode where
          Episode schedule date = Opposition transmission date
         AND Episode start time = Opposition transmission time
        Create instance of Opposition program
    Create the relationships Planning and Competing
Output: List of Opposition programs
```

下节课内容

第六章 面向对象方法

- □该章内容讲贯穿一个完整的例子。P258～P259
- □大约需要两次课
- □下节课之前请大家阅读6.1~6.5小节