CHAPTER 4

Architecture Design

Outline

(1)模块化原理

(2)模块独立性度量

(3)软件设计经验规则

模块设计原理

- 模块化 (modularization)
  - ✓ 分而治之(Divide-and-conquer)
  - ✓ 逐步求精refinement
  - ✓ 层次化 (hierarchy)
- 抽象（abstraction)
  - ✓ 注意点分散(Separation of Concerns)
- 信息隐蔽(information hiding)
  - ✓ 策略和实现的分离 (separation of police and implementation)
  - ✓ 接口和实现的分离 (separation of interface and implementation)
- 模块独立（module independence)
  - ✓ 耦合 (coupling)
  - ✓ 和内聚 (cohesion)

Modularization

What is a module?

1974 Steven:

A set of one or more contiguous program statements having a name by which other parts of the system can invoke it.

1979 Yourdon:

A module is a lexically contiguous sequence of program statements , bounded by boundary elements, having an aggregate identifier.

procedures, functions, subroutines, object,…

macro, class, data structure, …

模块化的定义

一般意义上的模块化

把一个大的任务划分成若干个小的子任务，子任务继续划分成多个更小的子子任务，一直进行下去，直到原子任务，乃至整个任务容易完成，这样的一个过程。

软件工程意义上的模块化

把程序划分成若干个模块，每个模块完成一个子功能，把这些模块的组成一个整体，可以完成整个软件系统指定的功能，从而满足用户的需求，这个过程称模块化。

procedures, functions, subroutines, lib, package, class, object, component, macro, data structure, …

Why modularization?

suppose P1, P2 be two problems,

function C(x) stands the complexity of problem x,

function E(x) represents the cost of solving problem x,

if C(P1) > C(P2)

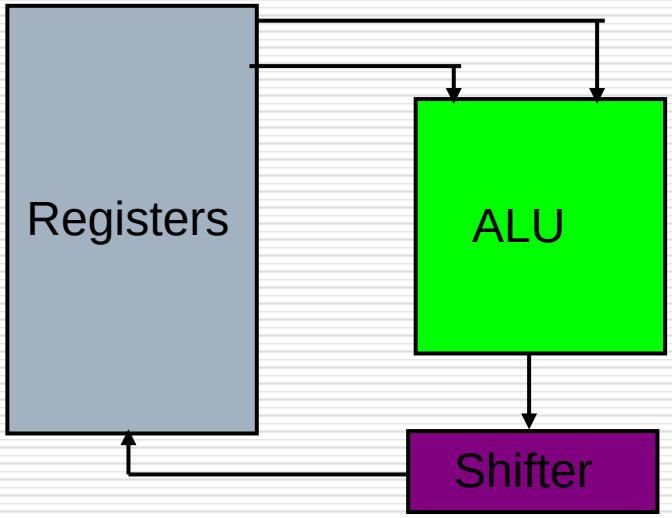
Then E(P1) > E(P2)

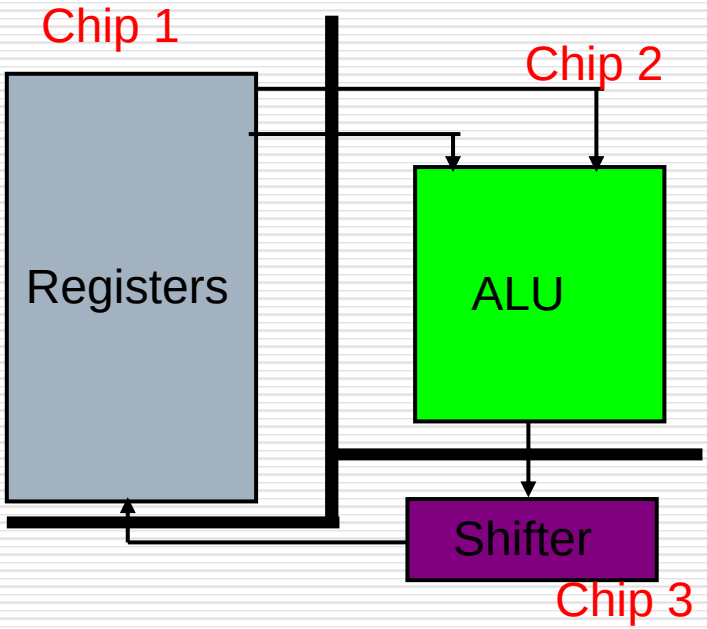
Experiential rule:

$$C(P1+P2) > C(P1) + C(P2)$$

$$E(P1+P2) > E(P1) + E(P2)$$

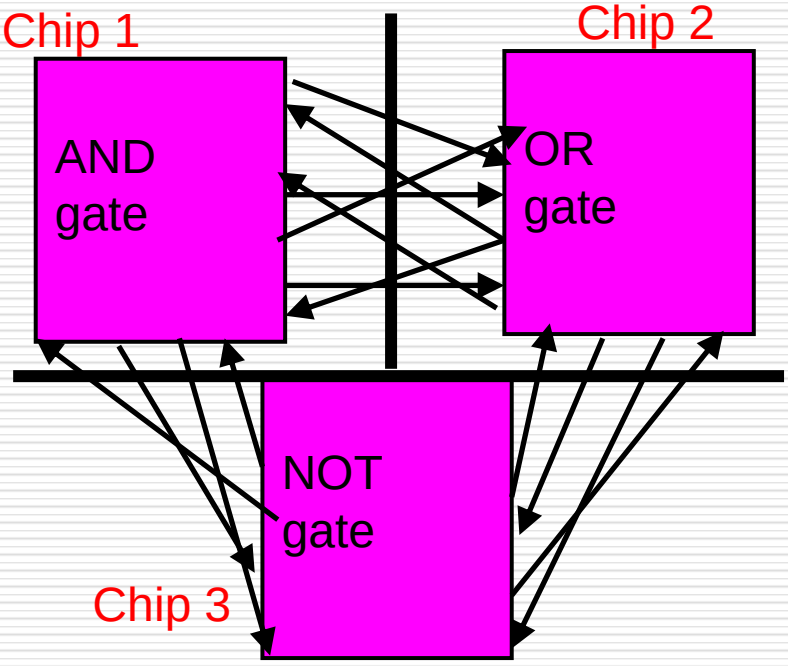
Example: Design of CPU





CPU fabricated on three chips

CPU fabricated on other three chips

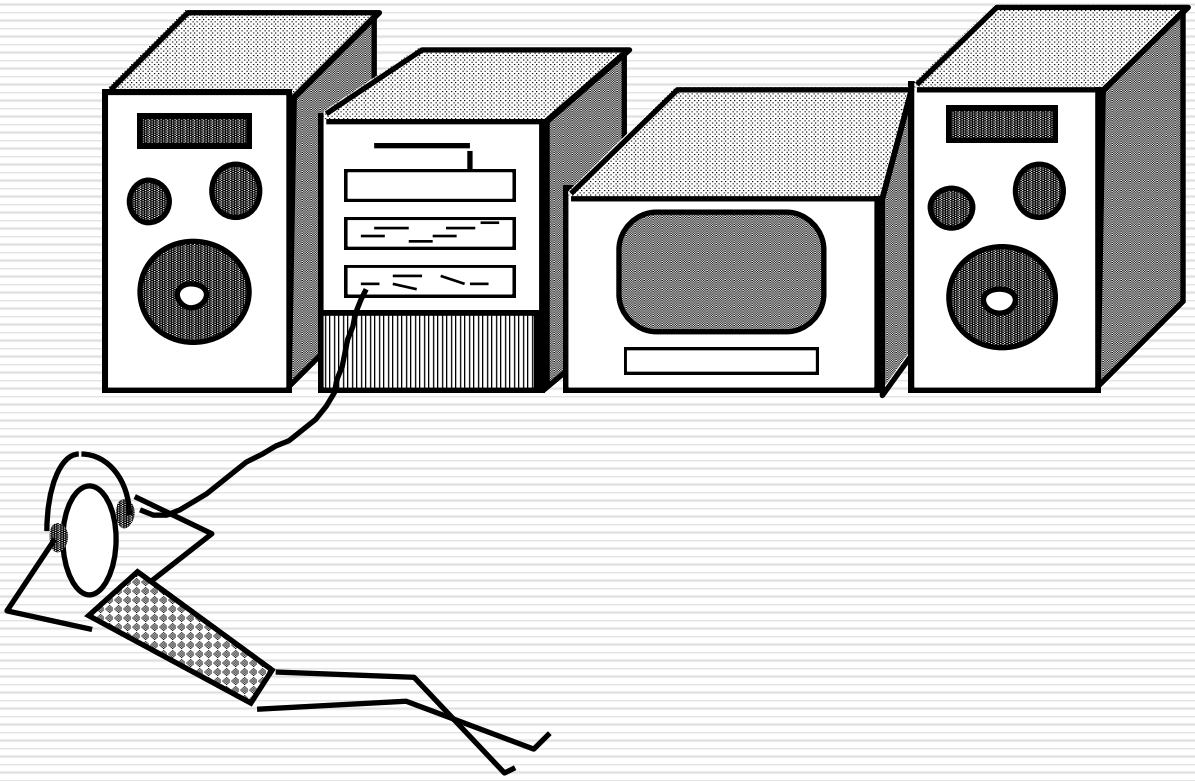


评论

- ✓ The two implementations are functionally equivalent.
- ✓ The right figure is harder to understand.
- ✓ Corrective maintenance of the circuits in right figure is difficult.
- ✓ The right figure is difficult to extend or enhance.
- ✓ The right figure can not reuse.

Benefits of Modularization

easier to build, easier to change, easier to fix ...



模块化的好处

- ✓ **Reduces complexity**
- ✓ **Facilitates change**
- ✓ **Easily understand as a stand-alone unit**
- ✓ **Reuse of existing modules**
- ✓ **Separate people can work on each part**
- ✓ **An individual software engineer can specialize.**
- ✓ **Easier implementation afforded by parallel development**

抽象

考虑问题时，集中考虑和当前问题有关的主要方面，而忽略和当前问题无关的方面，这就是抽象。或者说抽象就是抽出事物的本质特性，而暂时不考虑它们的细节。

软件工程过程的每一步，都是对软件解法的抽象层次的一次细化。在可行性研究阶段，软件被看作是一个完整的系统部分；在需求分析期间，我们使用在问题环境中熟悉的术语来描述软件的解法；当我们由总体设计阶段转入详细设计阶段时，抽象的程度进一步减少；最后，当源程序写出来时，也就达到了抽象的最低层。

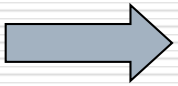
Abstractions 再认识

- Identify important aspects and ignore the details
- Abstractions allow you to understand the essence of a subsystem without having to know unnecessary details
- reducing complexity
- A good abstraction is said to provide information hiding

Modeling

抽象举例

火力发电，
水利发电，
风力发电，
核发电，
……



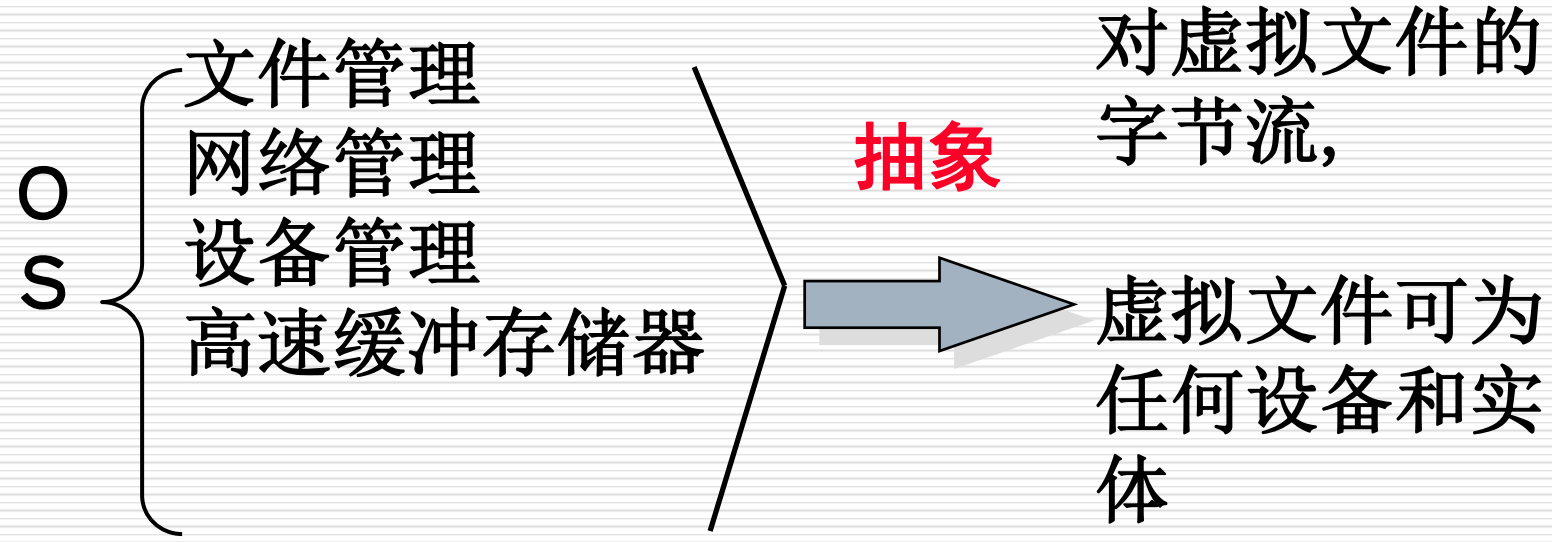
电源
电流

河水
江水
雨水
自来水
……

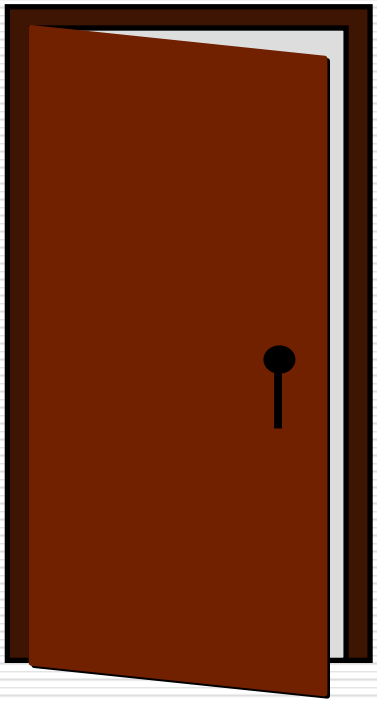
H_2O

抽象举例

Windows NT一体化的I/O系统设计



抽象举例

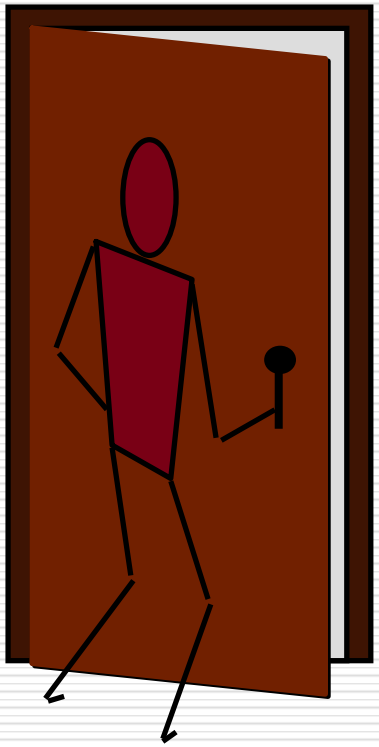


door

manufacturer
model number
type
swing direction
inserts
lights
type
number
weight
opening mechanism

implemented as a data structure

Procedural Abstraction

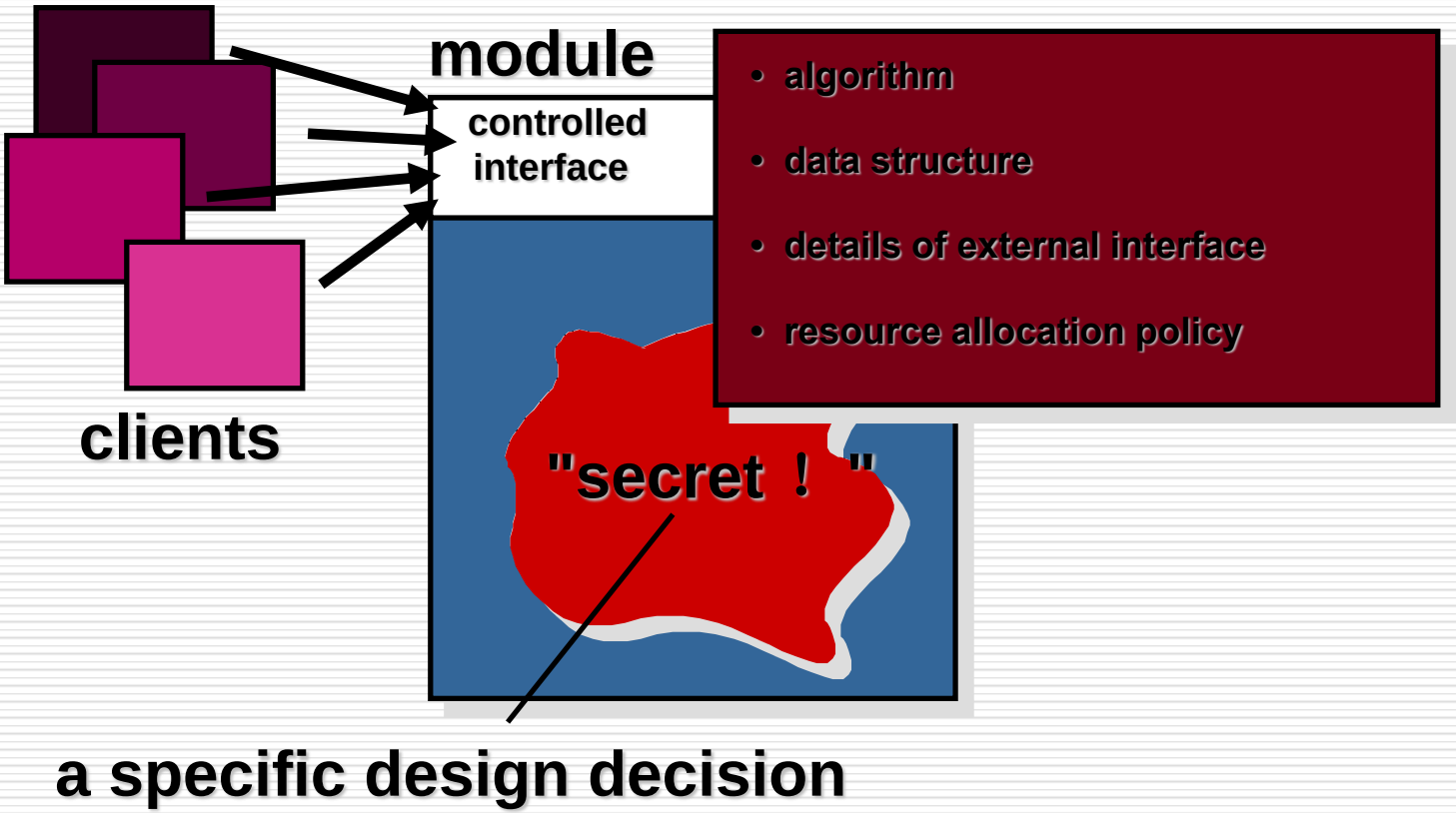


open

details of enter algorithm

implemented with a "knowledge" of the object that is associated with enter

Information Hiding



信息隐藏

- 信息隐蔽是指，每个模块的实现细节对于其它模块来说是隐蔽的。也就是说，模块中所包含的信息（包括数据和过程）不允许其它不需要这些信息的模块使用。

What is Information Hiding ?

- **Modules should be “characterized by design decisions that each hides from all others”**
- **Modules are designed so that information within a module is inaccessible to other modules with no need for the information**
- **Defines and enforces access constraints**
- 设计模块时，应使得一个模块内包含的信息（数据和过程）对于不需要这些信息的模块来说，是不能访问的。

Why Information Hiding?

- ✓ reduces the likelihood of “side effects”
- ✓ limits the global impact of local design decisions
- ✓ emphasizes communication through controlled interfaces
- ✓ discourages the use of global data
- ✓ leads to encapsulation—an attribute of high quality design
- ✓ results in higher quality software
- ✓ They can be changed freely if the change does not affect the interface

信息隐藏举例

日常生活中：

电视机
手机
复读机
……

软件产品中：

MediaPlay.exe
WinRAR.exe
ArobeReader.exe
File, open(), read(),
write()……
TCP/IP, socket(), accept()…
Class { private ……}
Jdbc/odbc中的数据库操作模块

Module Independence 模块独立性

模块独立是指开发具有独立功能而且和其它模块之间没有过多的相互作用的模块。

模块独立性, 是指软件系统中每个模块只涉及软件要求的具体的子功能, 而和软件系统中其它的模块的接口是简单的

例如, 若一个模块只具有单一的功能且与其它模块没有太多的联系, 则称此模块具有模块独立性

不是“你中有我，我中有你”

模块独立性举例

数据库操作

日志操作

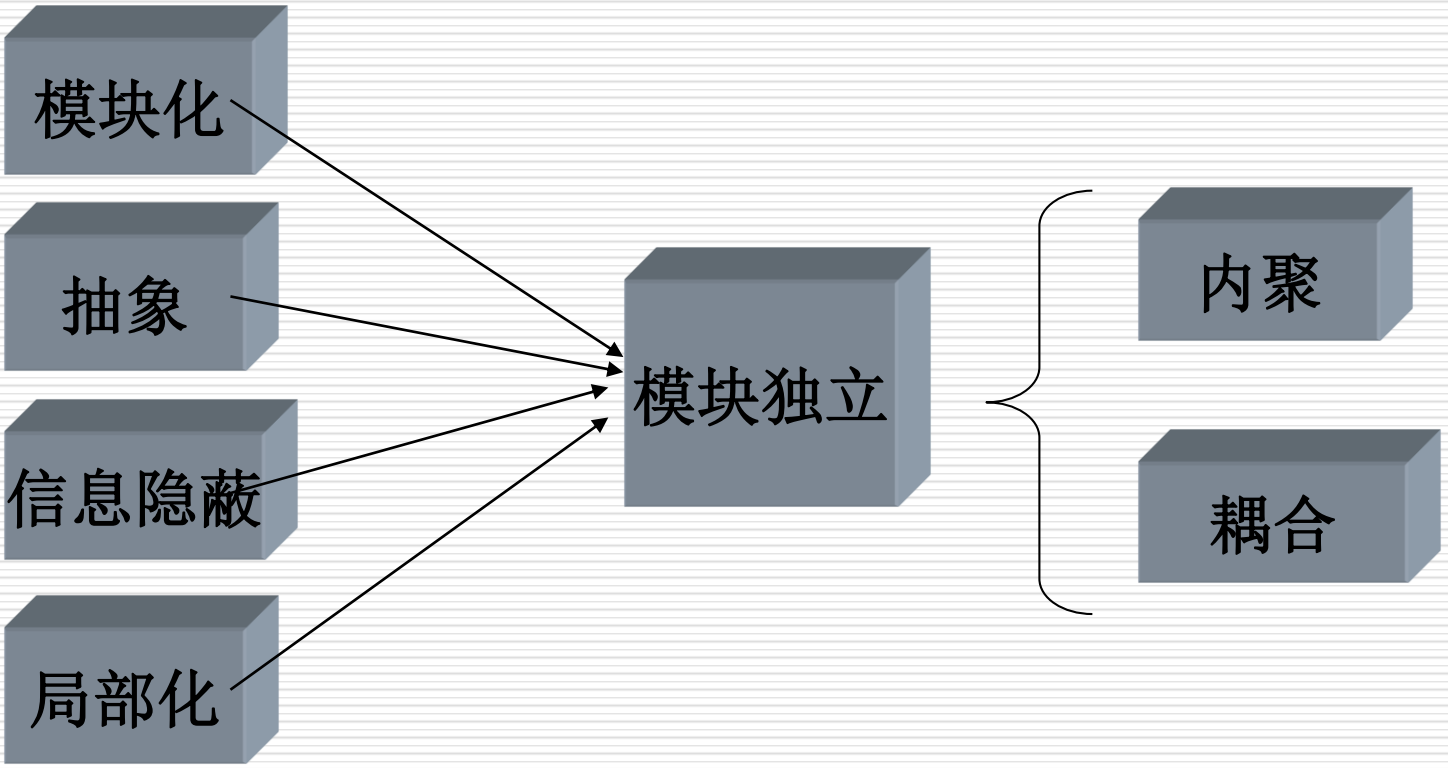
硬件通讯操作

报表打印操作

模块独立的好处

- ✓功能分割清楚，简化接口
- ✓独立的模块修改不会影响其他模块，易于测试和维护
- ✓易于分工，易于多人合作开发同一软件
- ✓符合信息隐蔽和信息局部化原则

如何做到“模块独立”？



追求：高内聚，低耦合

Object oriented remark (again)

Object: module in real life, naturally
class: abstraction to objects
encapsulation: private （inf. hiding）
protected
public
inheritance: coupling
reuse: module

Measurement of module independence

- **Coupling (耦合)**
  - ✓ The degree of interaction between two modules.
  - ✓ The degree to which a module is “connected” to other modules in the system.
- **Cohesion (内聚)**
  - ✓ the degree of interaction within a module.
  - ✓ the degree to which a module performs one and only one function.

模块独立程度的衡量标准

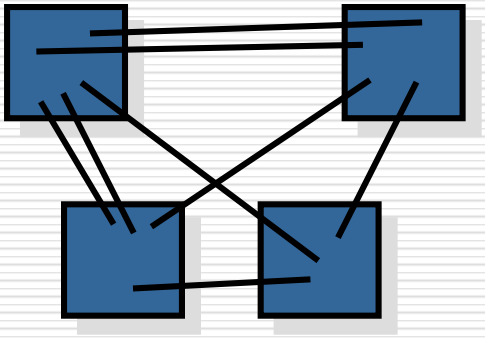
- 耦合性：对一个软件结构内不同模块间互连程度的度量。

- 内聚性：标志一个模块内各个处理元素彼此结合的紧密程度，理想的内聚模块只做一件事情。

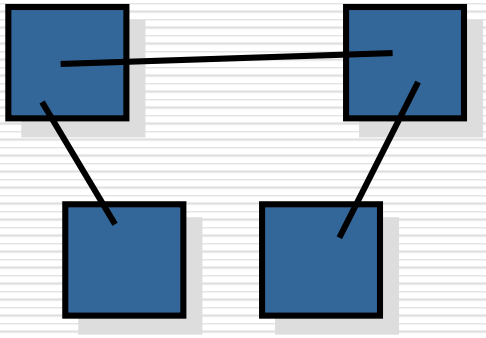
模块独立性比较强的模块应是高内聚，低耦合的模块。

Dependency: Coupling

Coupling is the degree of interdependence between modules



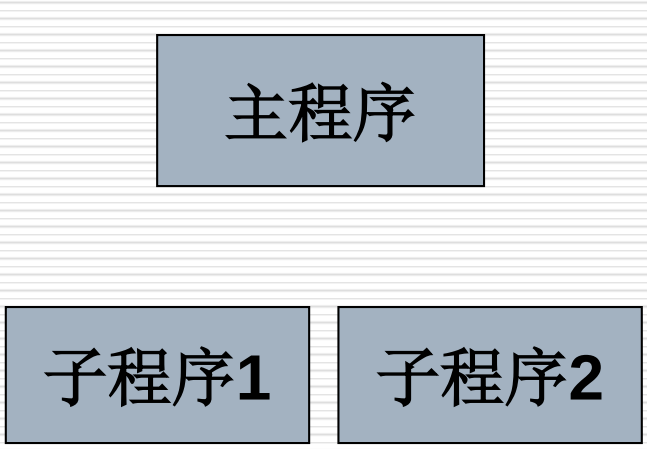
high coupling

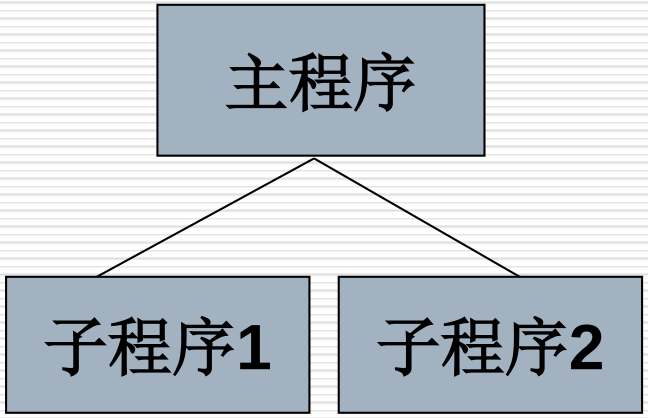


low coupling

耦合的概念

耦合是模块之间的互连程度度量



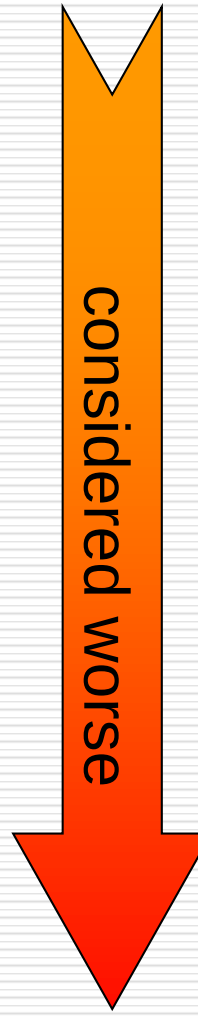


模块之间无连接，则无耦合　　模块之间存在连接，则存在耦合

模块之间的连接有：调用，返回，进入，跳出

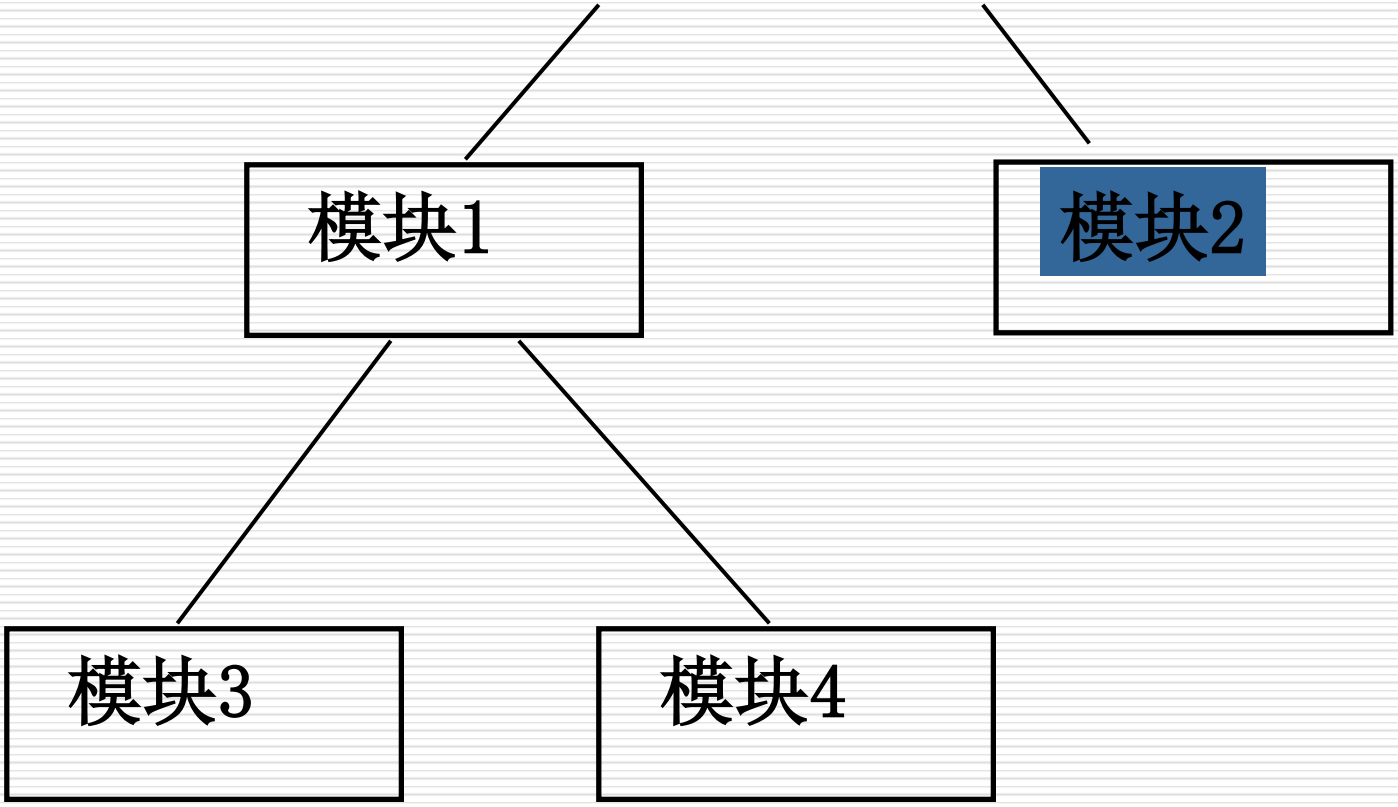
Types of Coupling

- **No coupling**
- **Data coupling**
  data from one module is used in another
- **Control coupling**
  one module may control actions of another module
- **Common coupling**
  two modules use the same environment variables
- **Content coupling**
  a module refers to the internals of another module



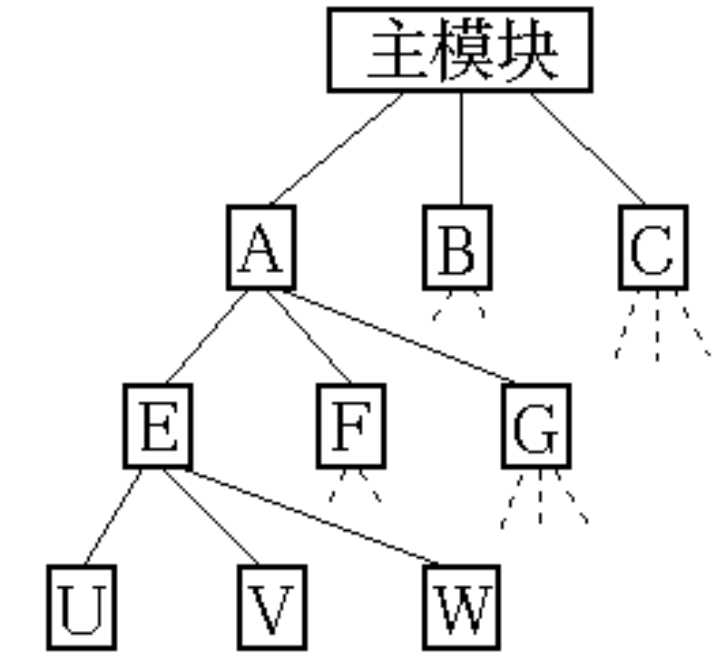
无直接耦合

两个模块没有直接关系(模块1和模块2)，模块独立性最强。



无直接耦合

如果两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的，这就是非直接耦合。这种耦合的模块独立性最强。

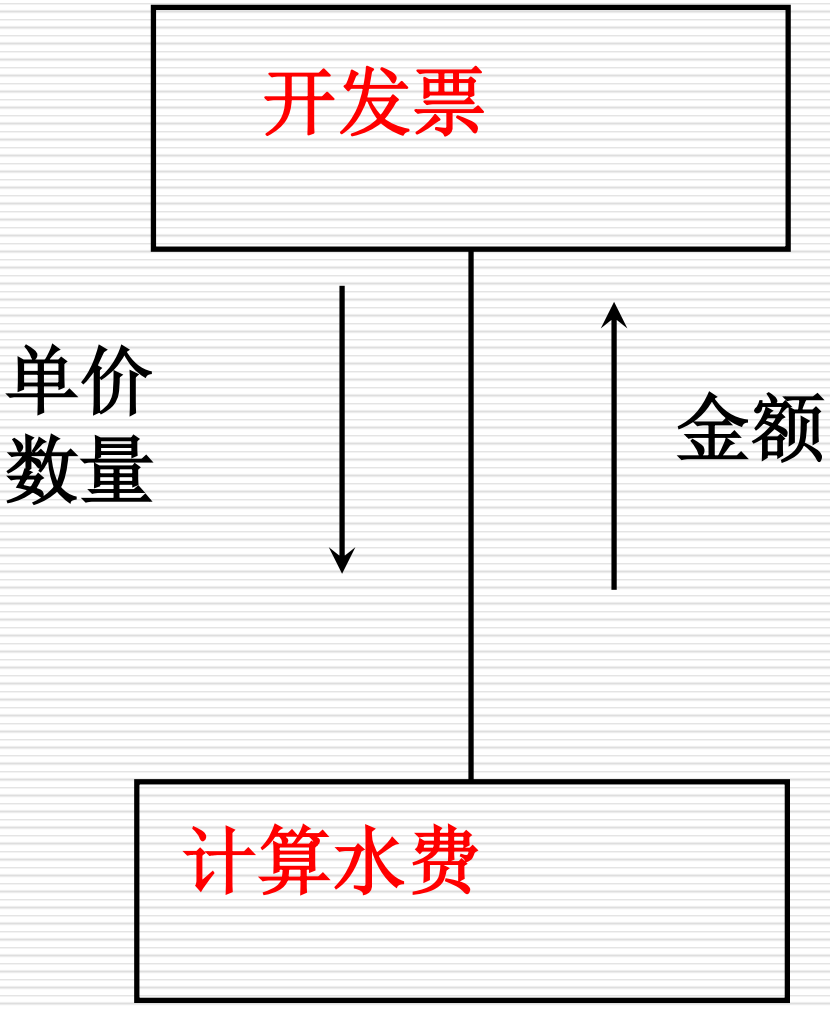


数据耦合

一模块调用另一模块时，被调用模块的输入、输出都是简单的数据(若干参数)。

属松散耦合。

数据耦合举例

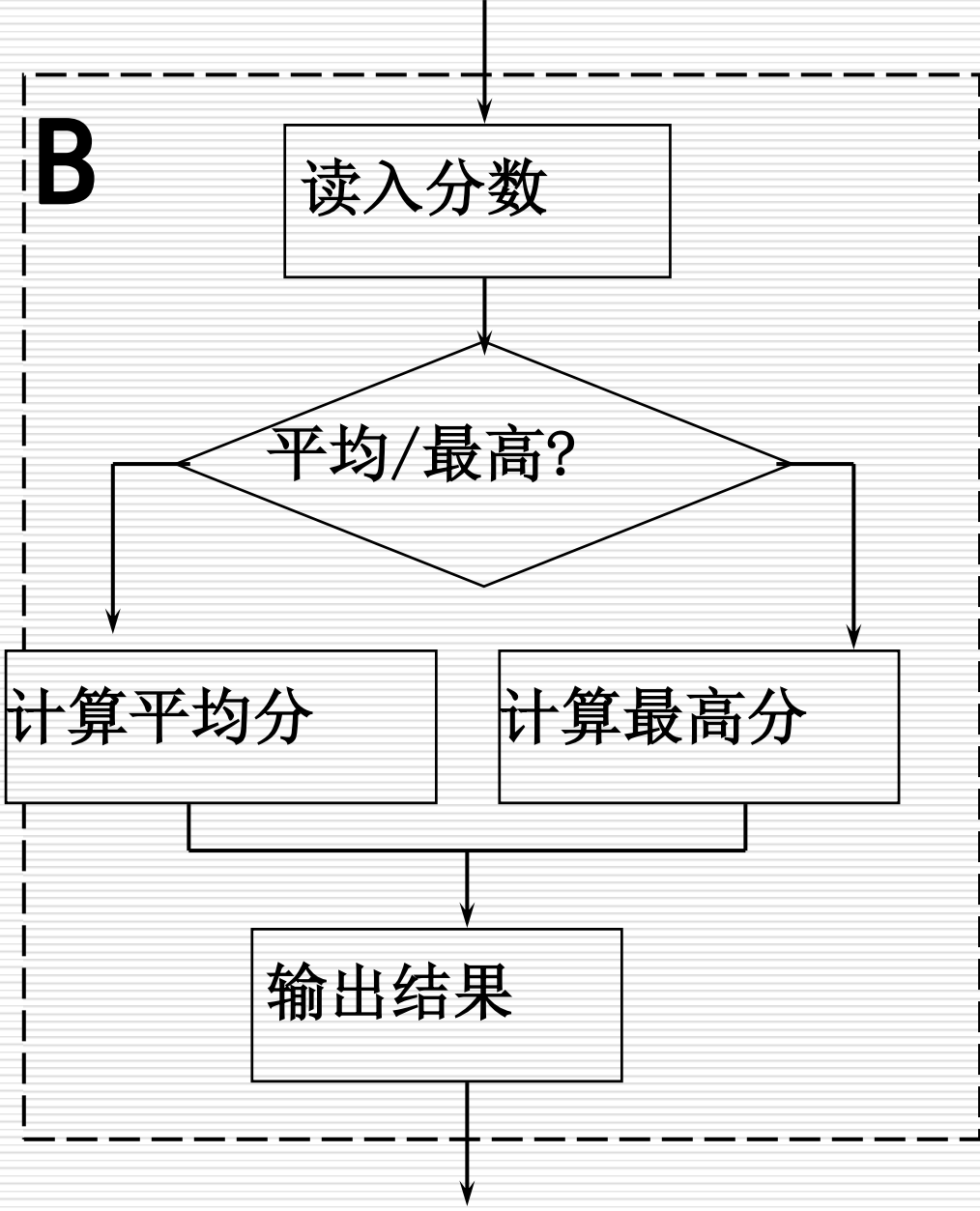


控制耦合

一模块向下属模块传递的信息（开关量、标志等控制被调用模块决策的变量）控制了被调用模块的内部逻辑。

控制耦合举例


A
Call B(flag,mark)
平均/最高
(控制信号
flag)
成绩
B
计算平均分
或最高分


B
读入分数
平均/最高?
计算平均分
计算最高分
输出结果

控制耦合

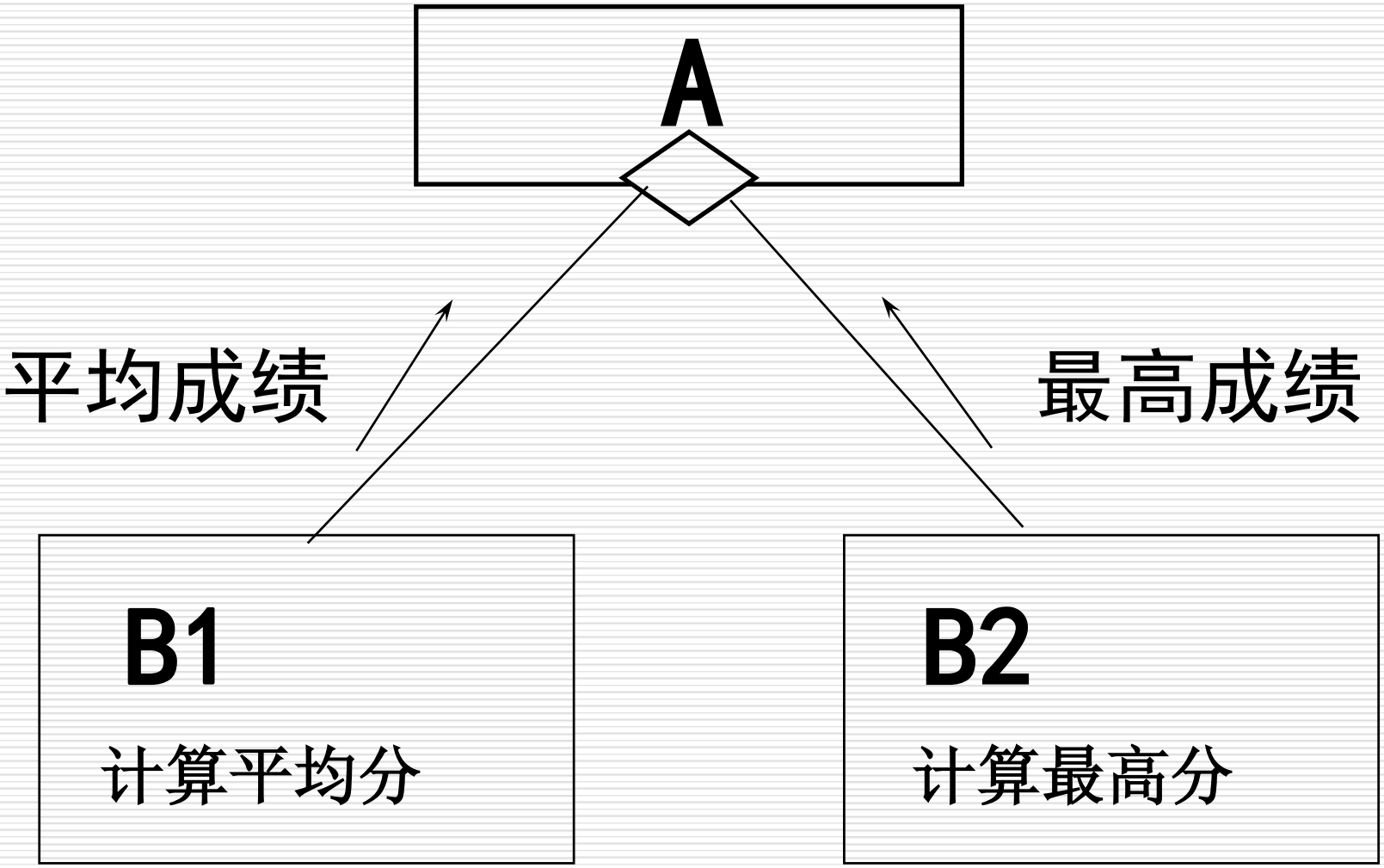
控制耦合增加了理解和编程的复杂性，调用模块必须知道被调模块的内部逻辑，增加了相互依赖

去除模块间控制耦合的方法：

(1)将被调用模块内的判定上移到调用模块中进行

(2)被调用模块分解成若干单一功能模块

改控制耦合为数据耦合举例



公共耦合(公共数据区耦合)

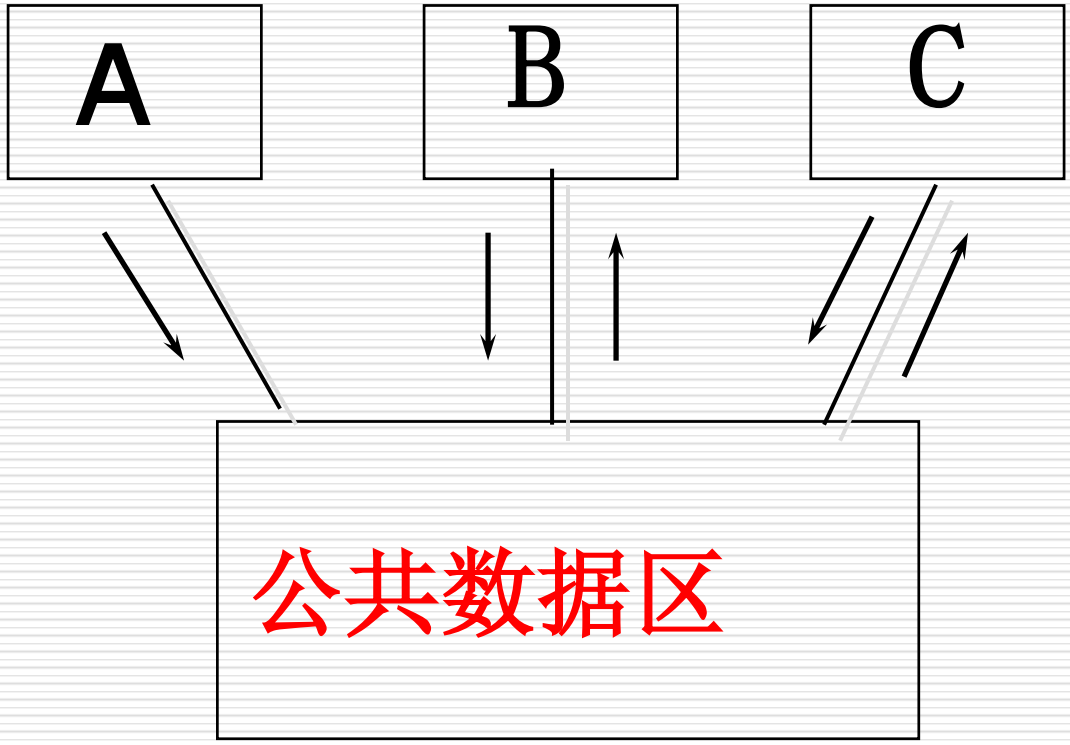
一组模块引用同一个公用数据区

(也称全局数据区、公共数据环境)。

公共数据区指:

- 全局数据结构
- 共享通讯区
- 内存公共覆盖区等

公共耦合举例



模块A、B、C间存在错综复杂的联系

公共耦合存在的问题

(1) 软件可理解性降低

(2) 诊断错误困难

(3) 软件可维护性差

(4) 软件可靠性差

(公共数据区及全程变量无保护措施)

慎用公共数据区和全程变量!!!

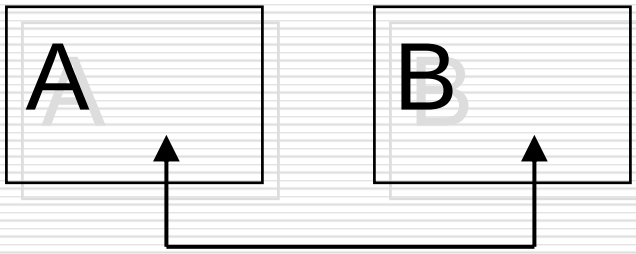
C语言中： **extern**

Fortran: **common**

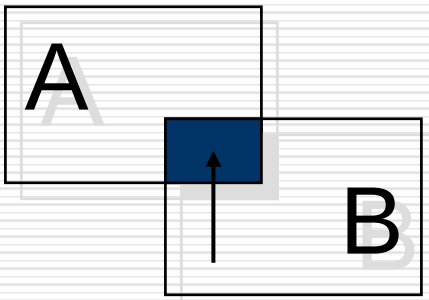
Basic: **data**

数据库： 频繁使用的表

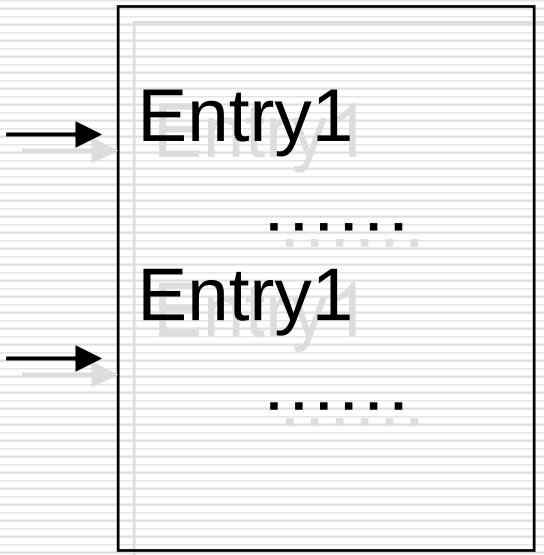
内容耦合



一模块直接访问另一模块的内部信息（程序代码或数据）



模块代码重叠



多入口模块

最不好的耦合形式 !!!

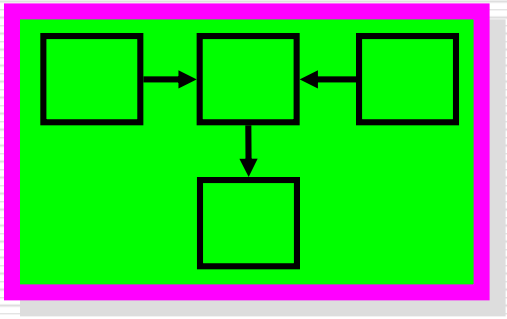
模块化设计，耦合的目标

目标：建立模块间耦合度尽可能松散的系统

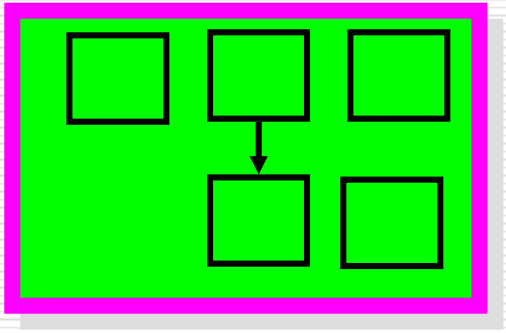
- ✓ 尽量使用数据耦合
- ✓ 少用控制耦合
- ✓ 限制公共耦合的范围
- ✓ 坚决避免使用内容耦合

Dependency: Cohesion 内聚

Cohesion is concerned with the interactions within a module



high cohesion



low cohesion

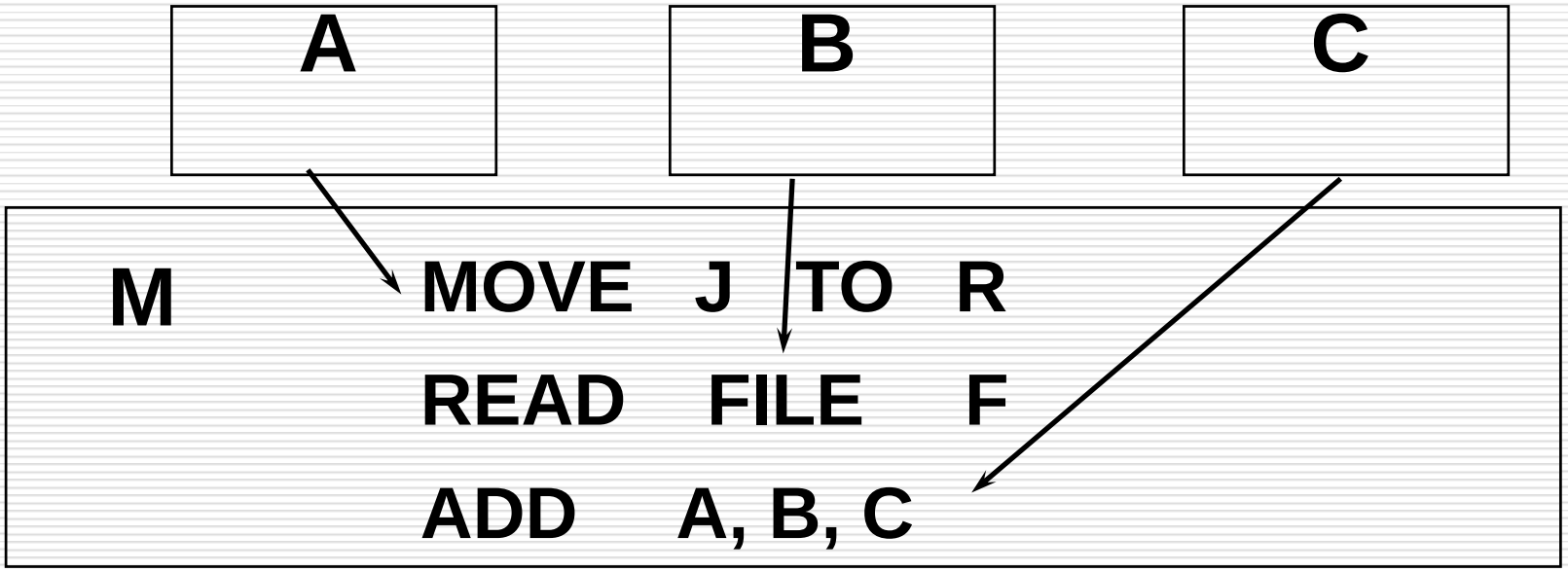
Type of Cohesion

1. coincidental cohesion
2. logical cohesion
3. temporal cohesion
4. procedural cohesion
5. communicational cohesion
6. informational cohesion
7. functional cohesion

模块的内聚性类型

偶然内聚(巧合内聚)

例: 模块内各部分间无联系



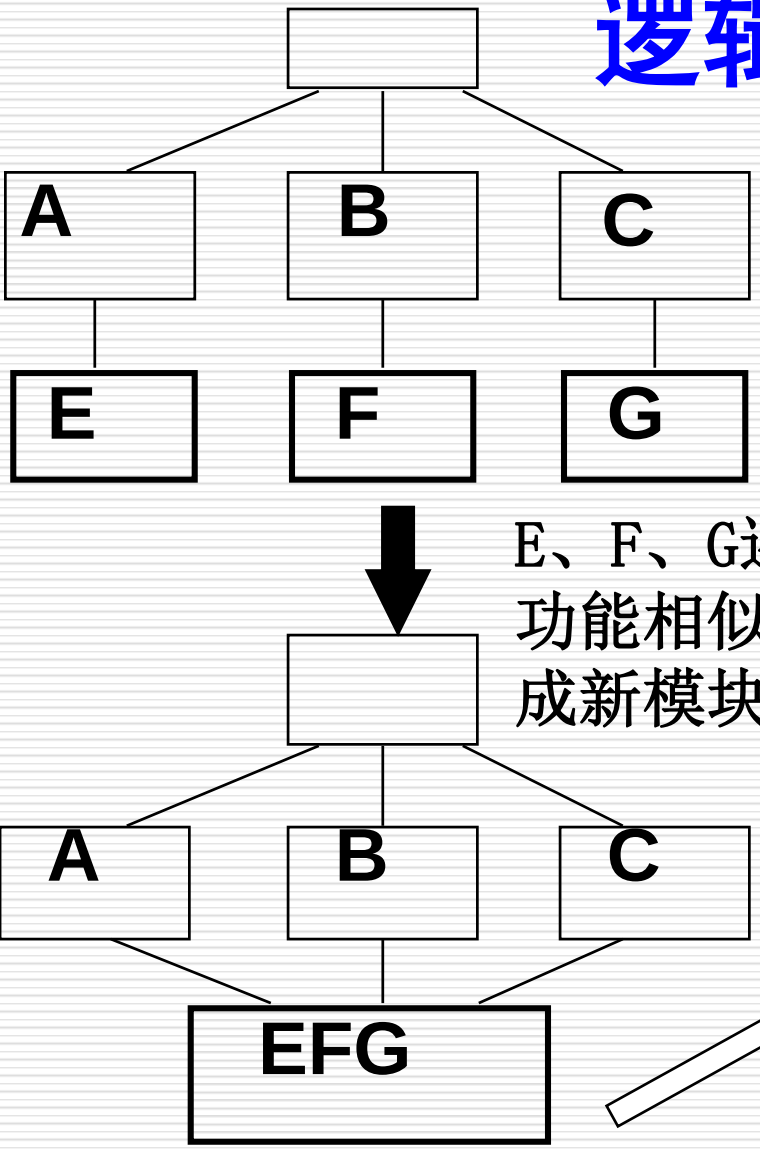
模块M中的三个语句没有任何联系

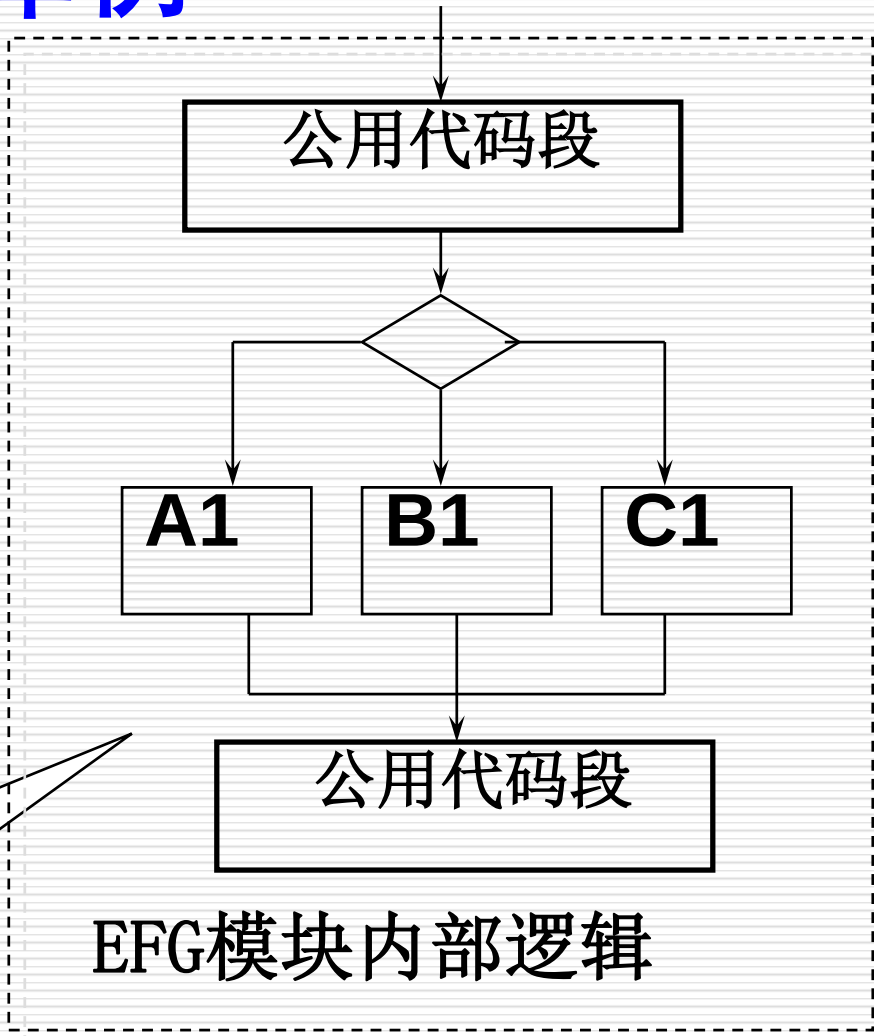
缺点： 可理解性差， 可修改性差

逻辑内聚

把类似功能的语句（逻辑上相似的功能），组合在一模块内，每次调用由控制模块的参数，确定执行哪种具体功能。

逻辑内聚举例





例：直线，曲线，图形等显示

缺点：增强了耦合程度(控制耦合)
不易修改，效率低

时间内聚(经典内聚)

模块完成的功能在同一时间内执行，这些功能只因时间因素关联在一起。

例如:初始化系统模块，

系统结束模块，

中断处理模块，

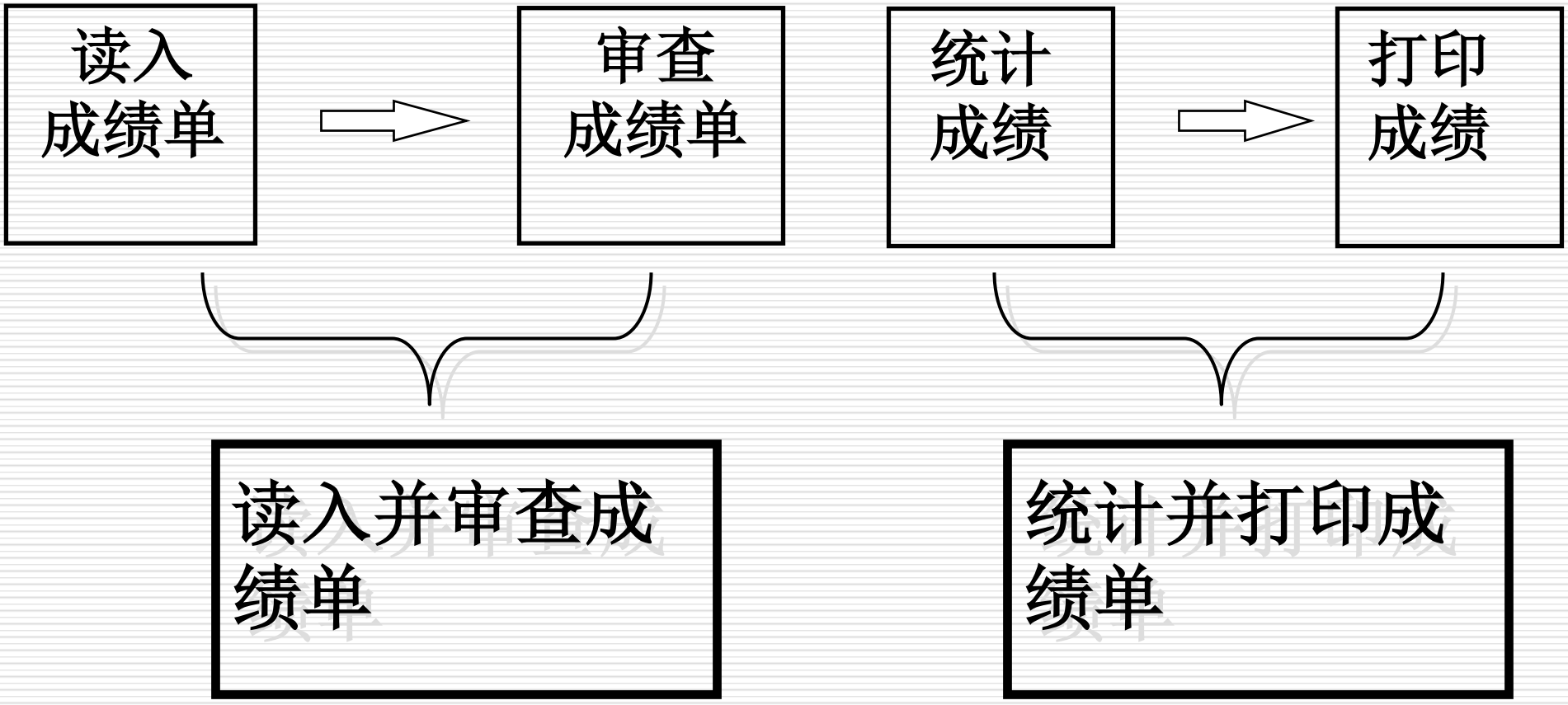
紧急故障处理模块等均是时间性聚合模块

过程内聚（顺序性组合）

模块内各处理成分有数据依赖顺序关系，必须以特定先后次序执行。

书上将其分成：过程内聚，顺序内聚

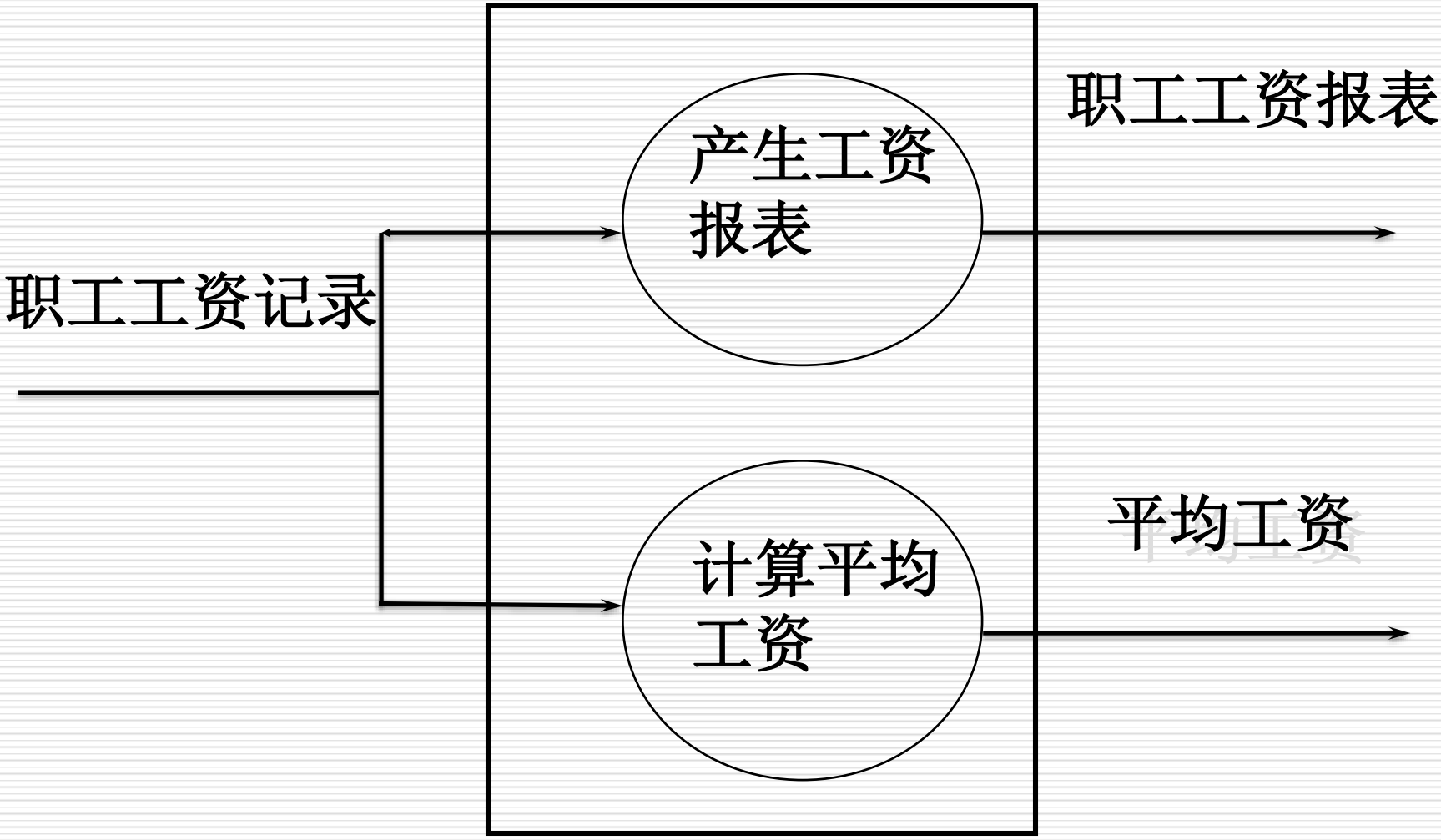
过程内聚模块举例



通信内聚

模块内各部分使用相同的输入数据，或产生相同的输出结果。

通信内聚模块举例

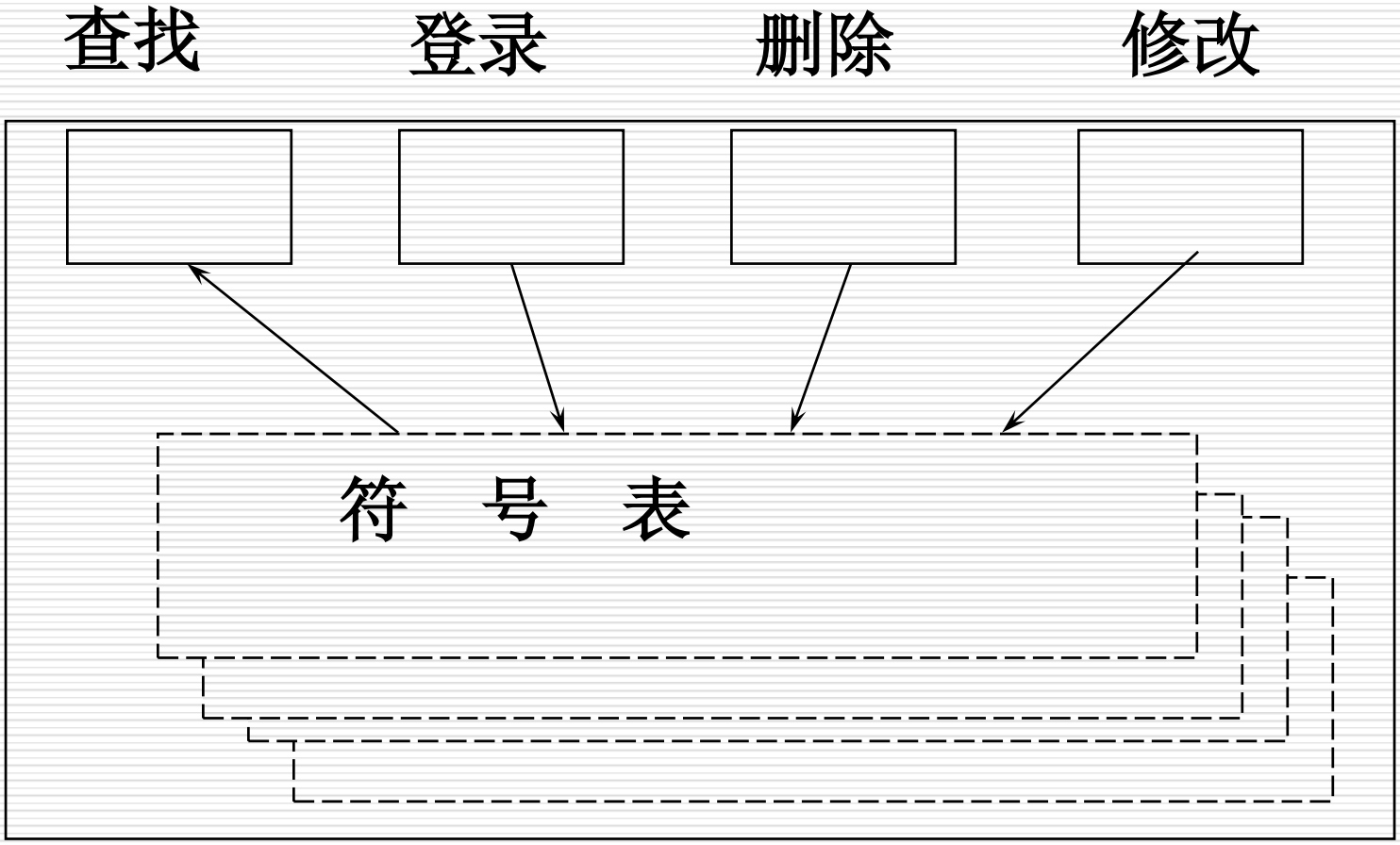


职工工资报表并计算平均工资模块

通信内聚

模块完成多个功能，各个功能都在同一数据结构上操作，每一功能有唯一入口。

通信内聚模块举例



几个操作同时引用一个共同的数据，Database

功能内聚

模块仅包括，为完成某个功能所必须的所有成分。

（模块所有成分共同完成一个功能，缺一不可 ）

例如：求一元二次方程根模块

内聚性最强

模块化设计，内聚的目标

目标：建立各个模块完成独立功能的高效和专一系统

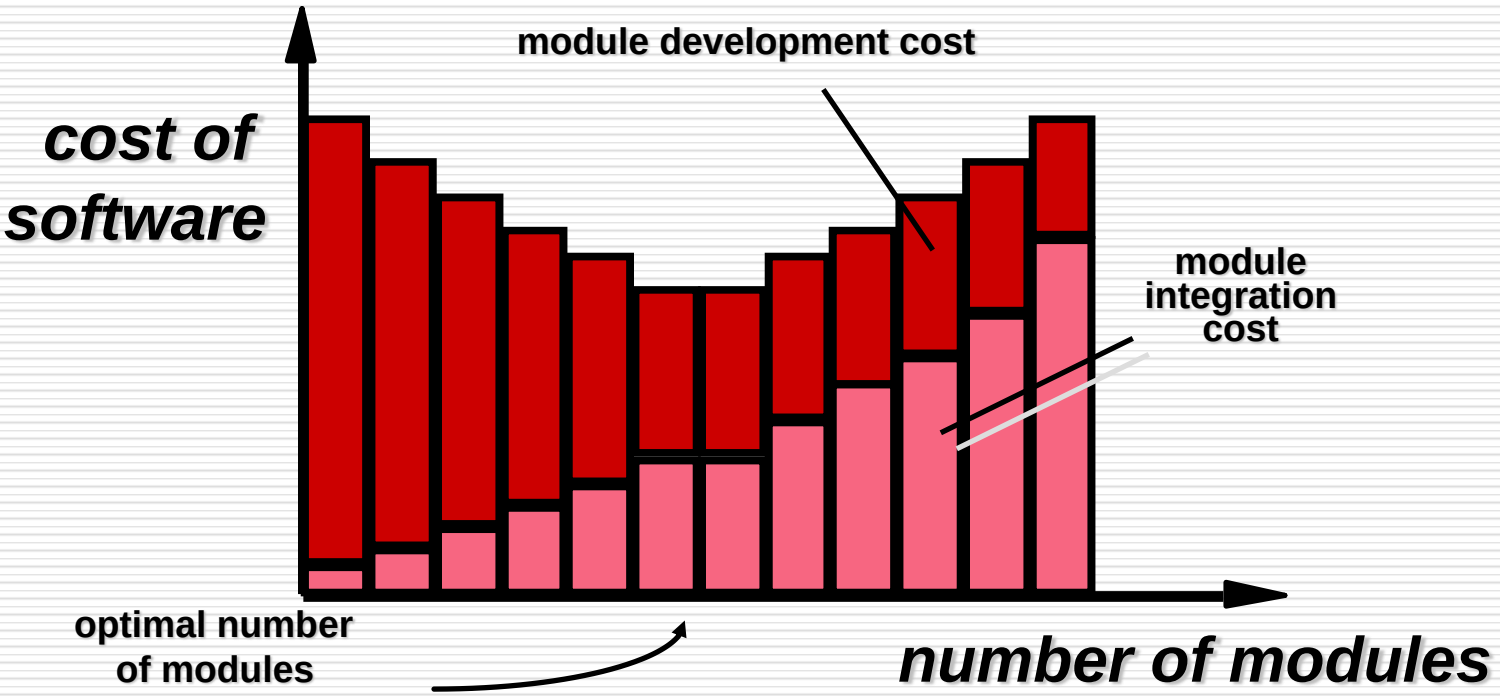
- ✓ 优先使用功能内聚
- ✓ 尽量满足过程内聚（顺序内聚）
- ✓ 少用逻辑内聚
- ✓ 坚决避免偶然内聚

Heuristic rules in module design

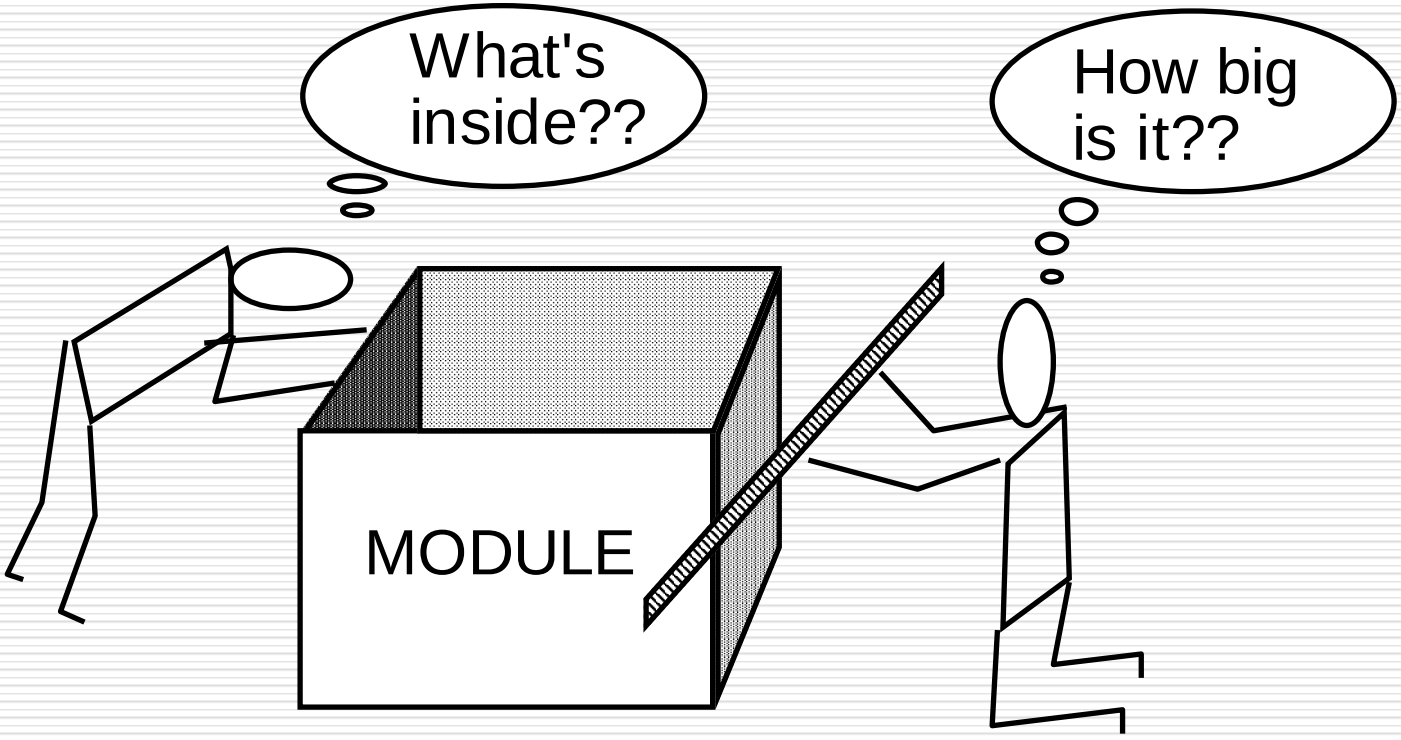
模块设计启发式规则

Modularity: Trade-offs

What is the "right" number of modules for a specific software design?



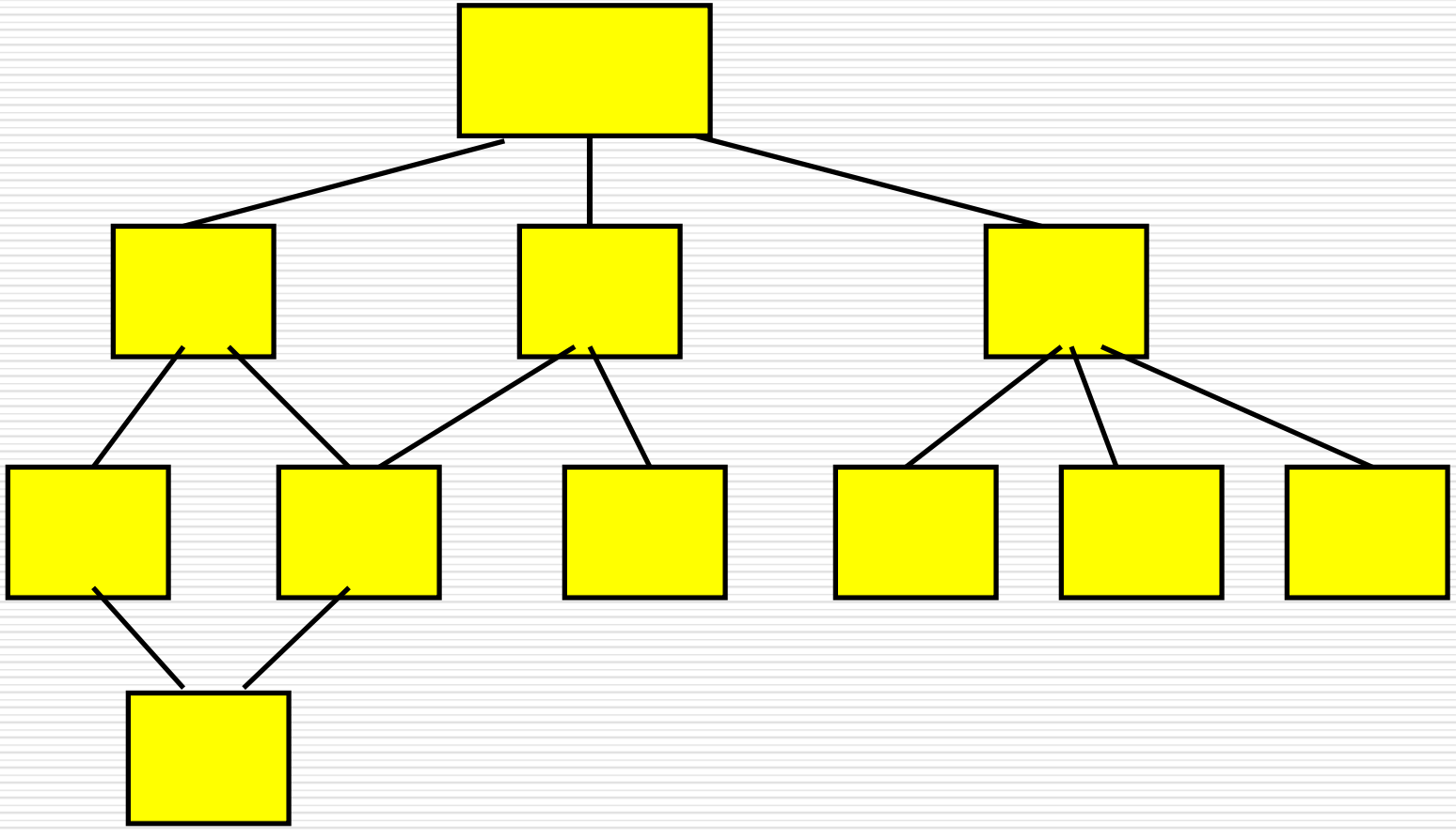
Sizing Modules: Two Views



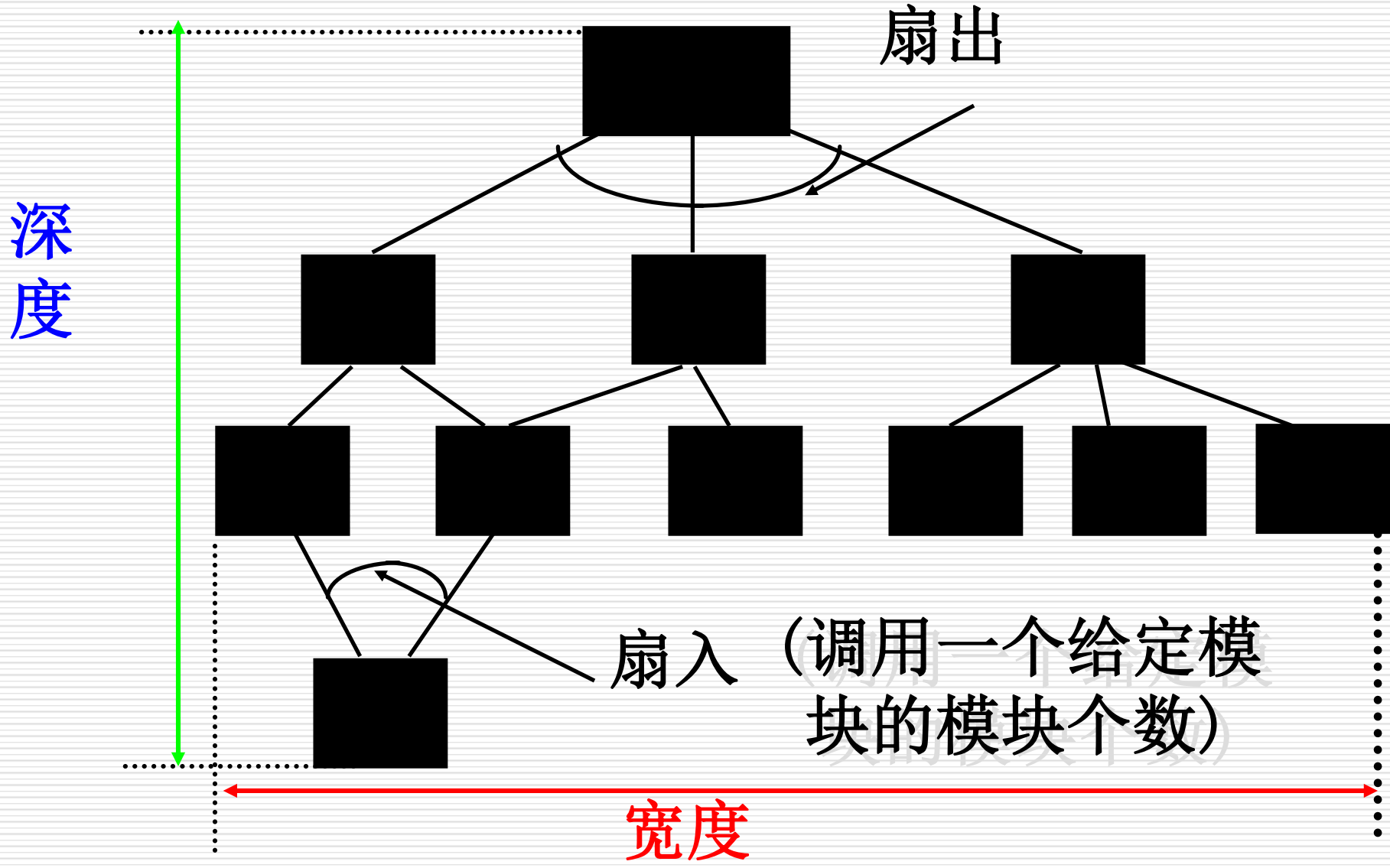
软件结构的度量和术语

- 深度：表示控制的层数。
- 宽度：表示控制（同一层次）总跨度。
- 扇出数：指由一模块直接控制的其他模块的数目。
- 扇入数：指有多少个模块直接控制一个给定的模块。
- 上级模块
- 下级模块

控制结构图示



软件结构度量术语



改进软件结构提高模块独立性

- ✓ 设计出软件的初步结构以后，应通过模块分解或合并审查，力求降低耦合提高内聚。模块的划分要符合独立性原则。
- ✓ 经验表明，一个模块的规模过大，可理解程度迅速下降。要进一步分解，分解后不应该降低模块独立性。
- ✓ 过小的模块，开销过大；模块数目过多将使系统接口复杂。

软件结构的特性参数要适当

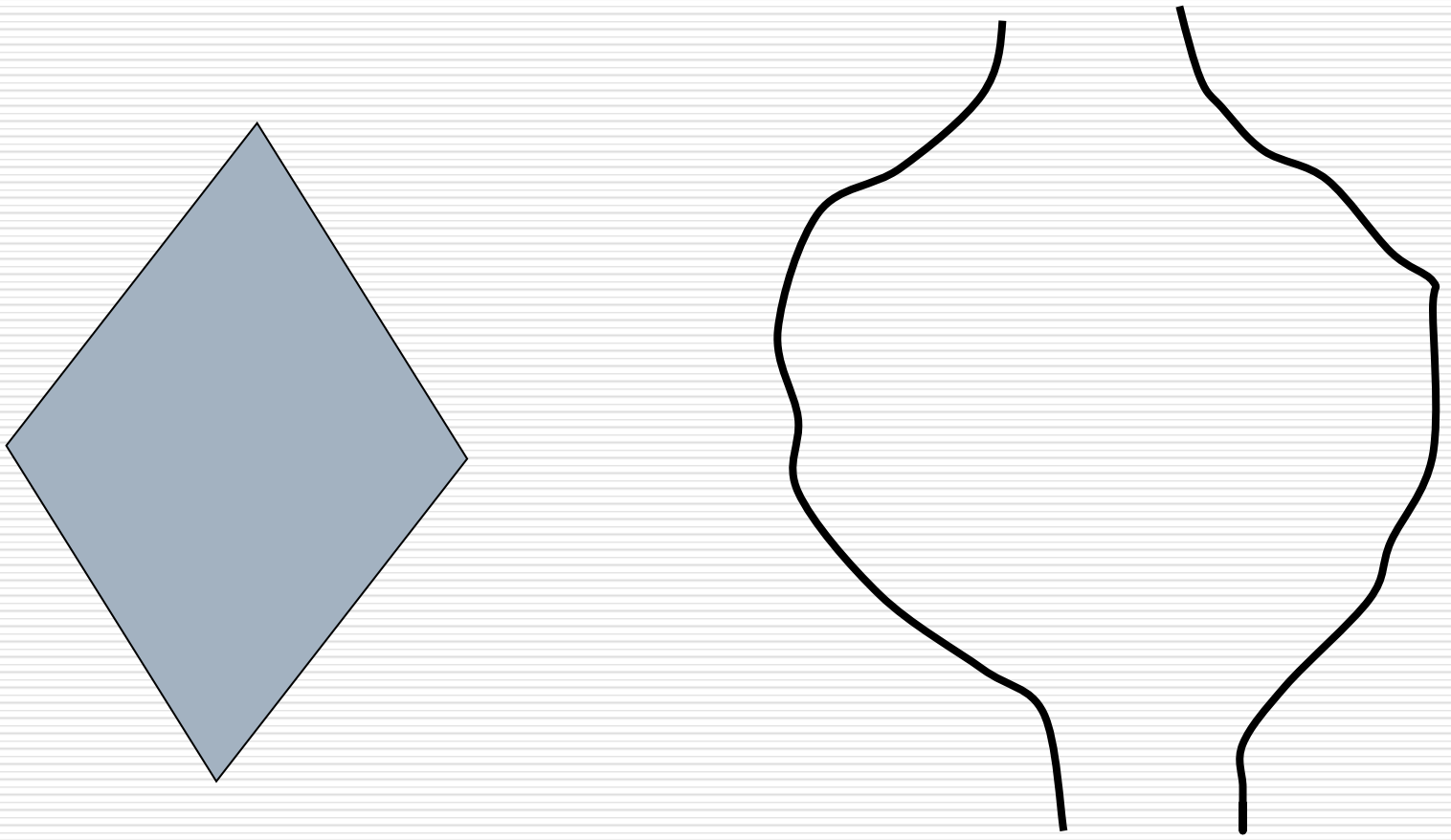
深度：表示软件结构中控制的层数，它能粗略表示软件的复杂程度。

宽度：表示软件结构同一层内的模块总数。宽度越大系统越复杂，对宽度影响最大因素是模块的扇出。宽度太大可增加深度来减少。

扇出：是一个模块直接控制（调用）的模块数目（5-9）。扇出越大模块越复杂。

扇入：是直接调用的上级模块数（3-5）。扇入太大会增加模块接口数，违背模块独立性原则。

观察大量软件系统后发现，设计得很好的软件结构通常顶层扇出比较多，中层扇出较少，底层扇入到公共的实用模块中去（底层模块有高扇入）。



软件结构形态

模块的作用域应该在控制域之内

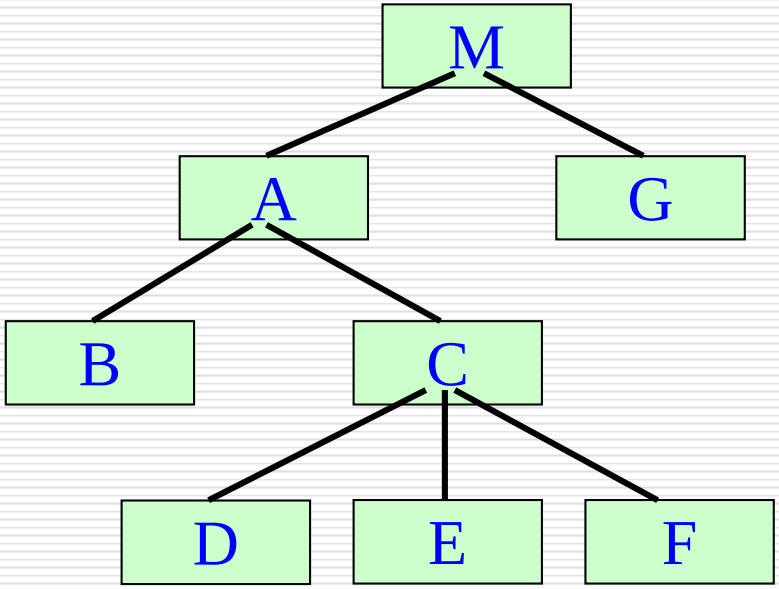
模块的作用域：是指受该模块判定影响的所有模块数。

模块的控制域：是受这个模块直接或间接控制调用的模块数。

模块的控制范围：本身及其所有下级模块。

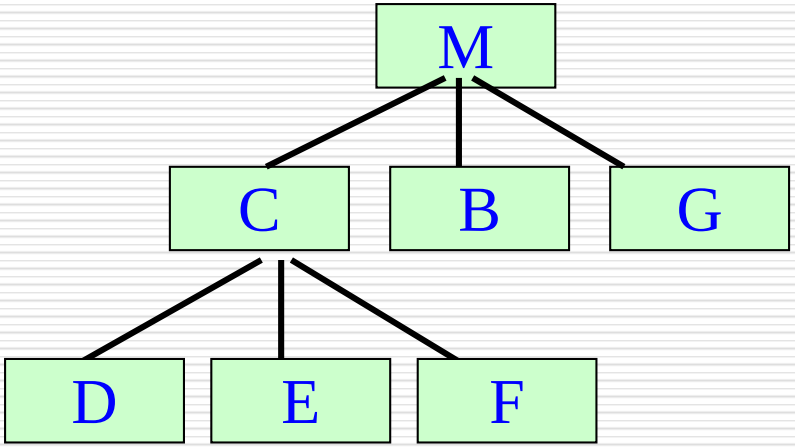
模块的作用范围： 即直接调用的模块 。

举例

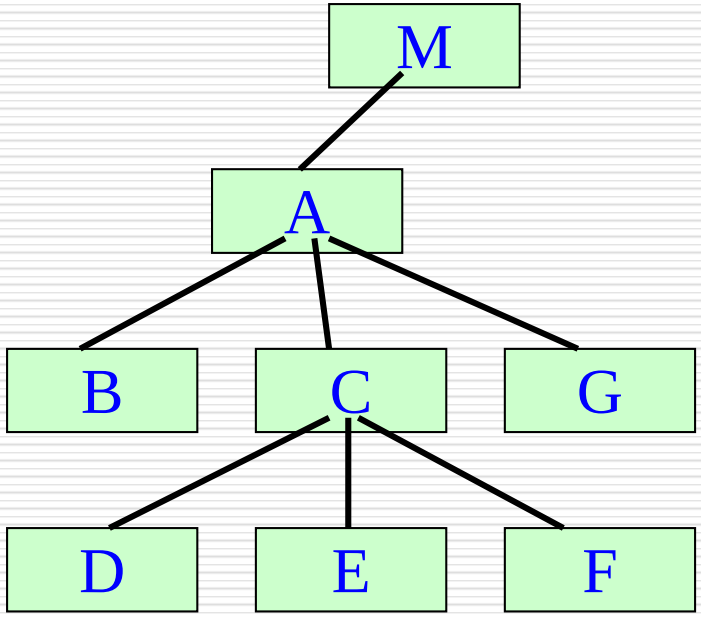


图中：假设模块A的作用域为B、C，模块A的控制域为B、C、D、E、F，则A的控制域包括了作用域，软件结构的划分是合理的。如果模块A的作用域为B、C，G，模块A的控制域为B、C、D、E、F，则A的控制域不包括了作用域，软件结构的划分是不合理的，将使模块间出现控制耦合，应要重新划分调整。

重新调整的方法：①把做判定的点往上移。上例中，把判定从模块A中移到模块 M中，见下图(a) 。②把那些在作用域内但不在控制域内的模块移到控制域内，成为它的直属下级模块。上例中，把模块G移到模块A的下面，见下图(b) 。



图(a) A上调到M中



图(b) G下调由A作用

力争降低模块接口的复杂程度

模块接口复杂是软件发生错误的一个主要原因，应该仔细设计模块接口，使得信息传递简单并且和模块的功能一致。

例如，求一元二次方程的根的模块Q-root(tbl,x) ，其中用tbl传送方程的系数，用数组x回送求得的根。这种传递信息的方法不利于对这个模块的理解，不仅在维护期间容易引起混淆，在开发期间也可能发生错误。下面这种接口可能是比较简单的:

Q-root(A,B,C,root1,root2)其中A,B,C是方程的系数，root1和root2是算出的两个根。

设计单入口单出口的模块

- ✓ 该规则是说模块接口尽量要单入口、单出口，这样可避免出现模块的内容耦合，降低接口成本。

- ✓ 当从顶部进入模块并且从底部退出来时，软件是比较容易理解的，因此也是比较容易维护的。

结构化程序设计极力主张。

模块功能应该可以预测

- ✓ 模块的功能应该能够预测，但也要防止模块功能过分局限。
- ✓ 如果说一个模块可以当做一个黑盒子，也就是说，只要输入的数据相同就产生同样的输出，这个模块的功能就是可能预测的。

以上列出的启发式规则多数是经验规律，对改进设计，提高软件质量，往往有重要的参考价值；但是，它们既不是设计的目标也不是设计时应该普遍遵循的原理。

Some heuristic rues（小结）

(refers to page 99--102)

- **Enhancing module independency**
- **The size of module should be moderate. (30-60 lines)**
- **The depth(3-5), width(5-8), fan-in(3-5), fan-out(5-9) of module are appropriate.**
- **The action domain of module is within its control domain.**
- **Reducing the complexity of module interface.**
- **Module should be design as single entry and single exit.**
- **The function of module can be predictable.**

Homework 2024-10-24

Page 114

T1

T2

T3 (1) (2)