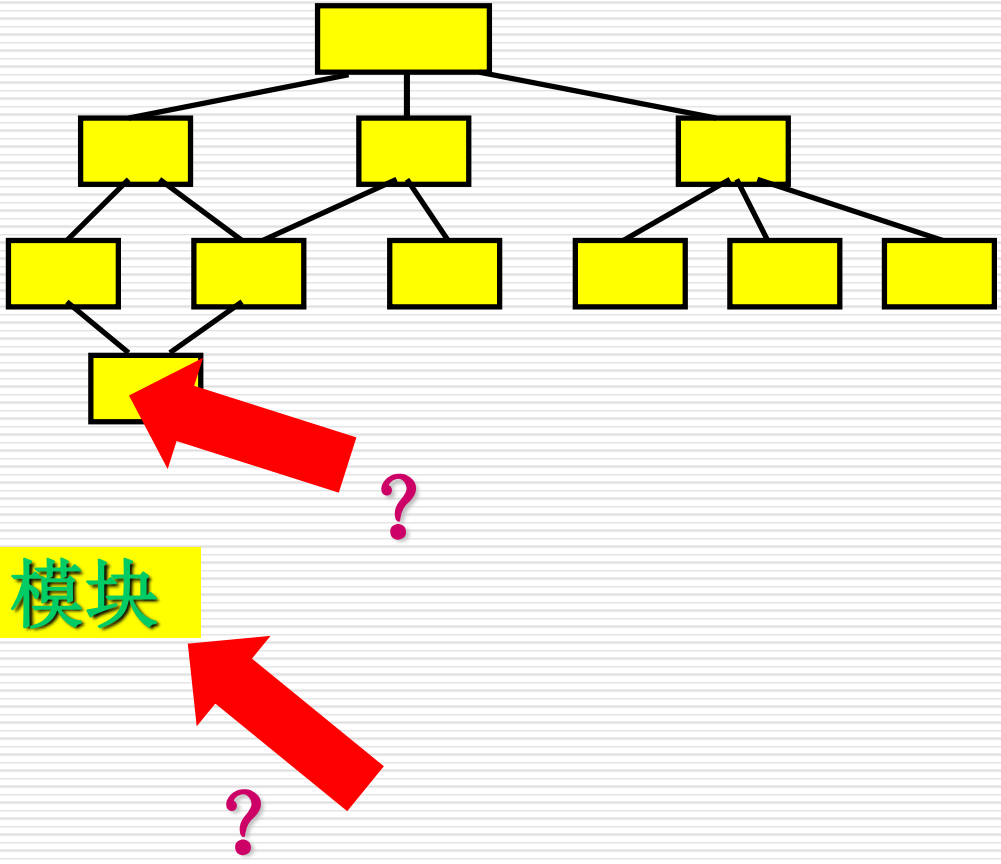
CHAPTER 5

Detailed Design

软件设计的任务

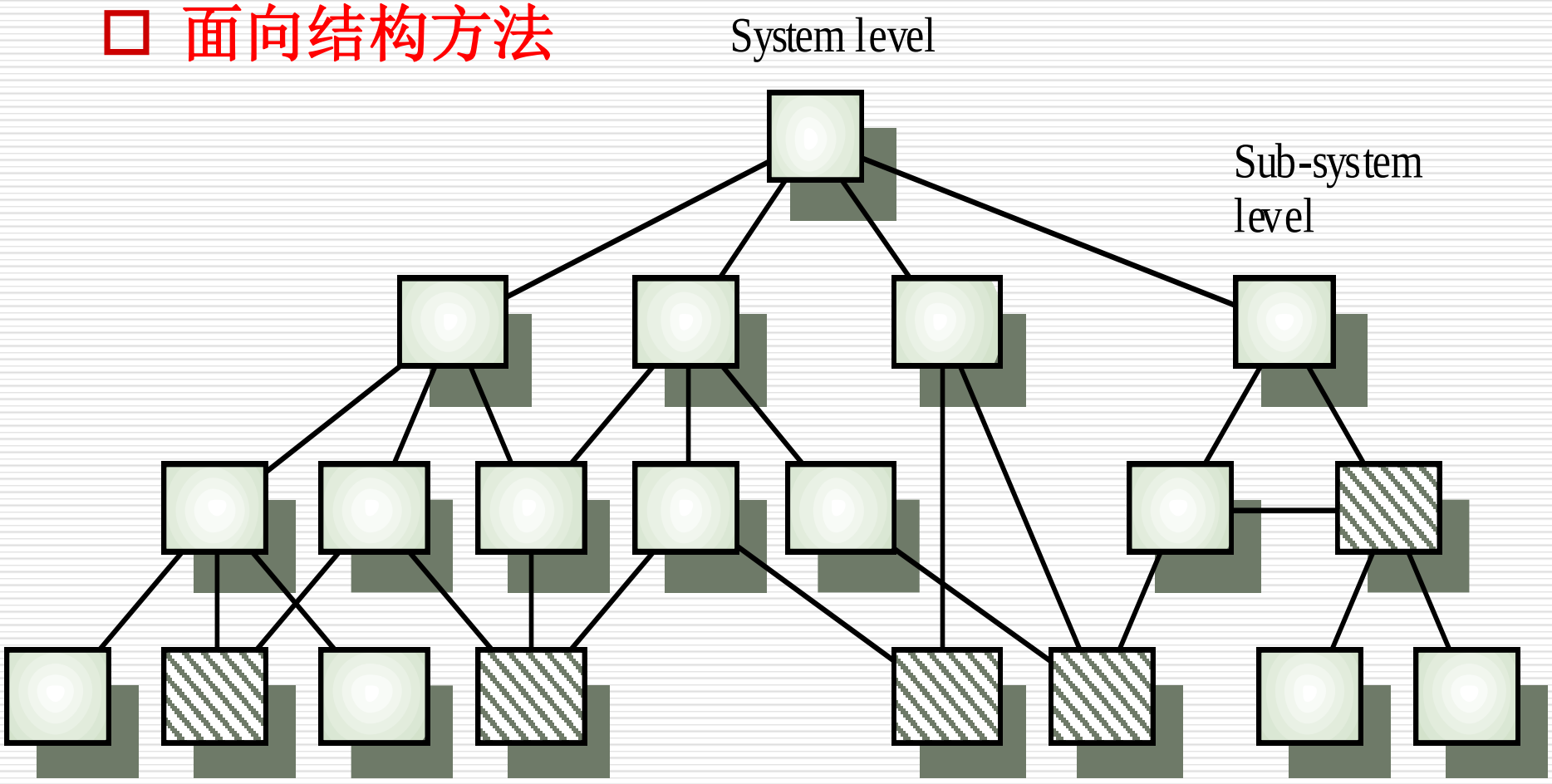
总体设计



详细设计

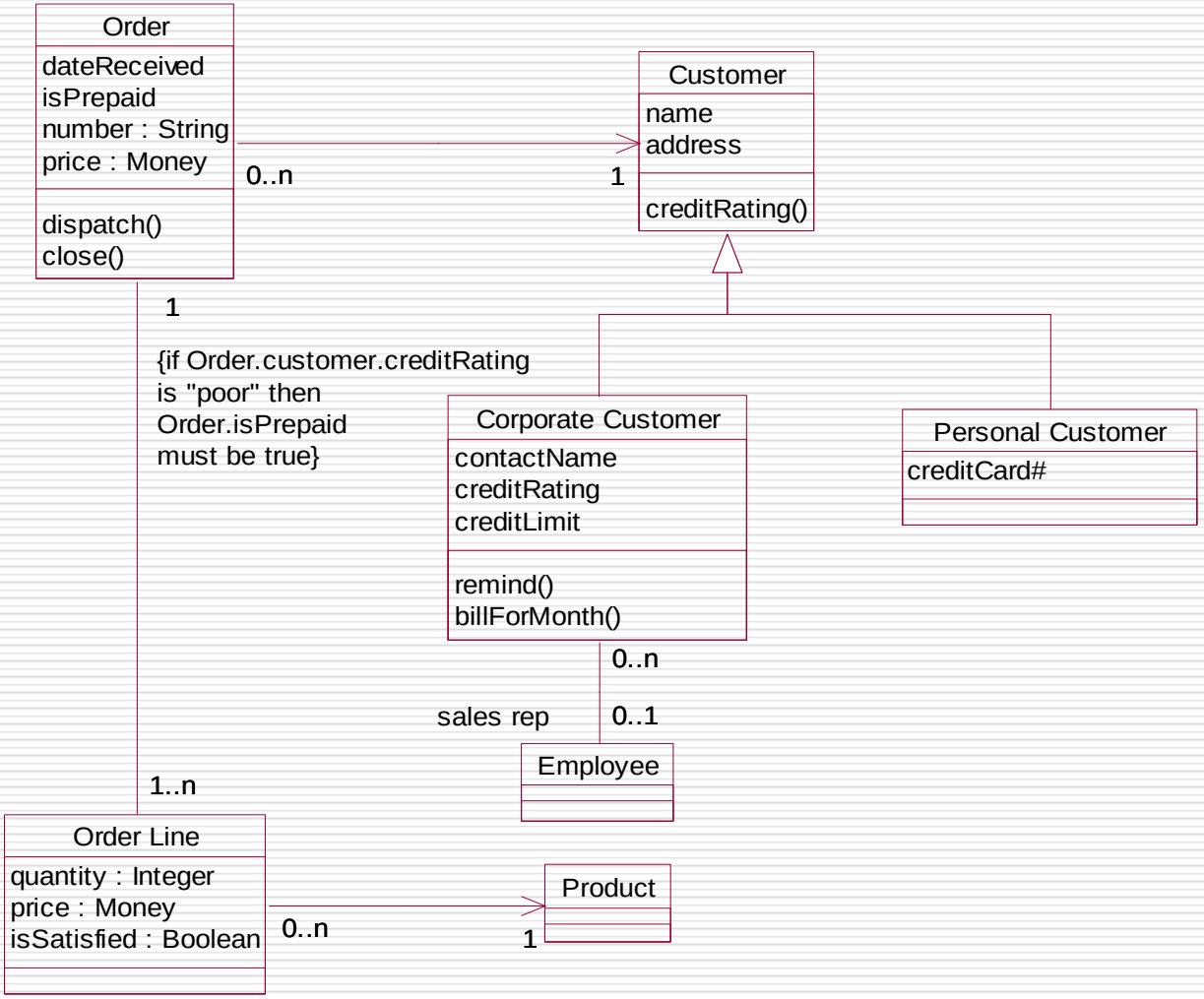
Architecture

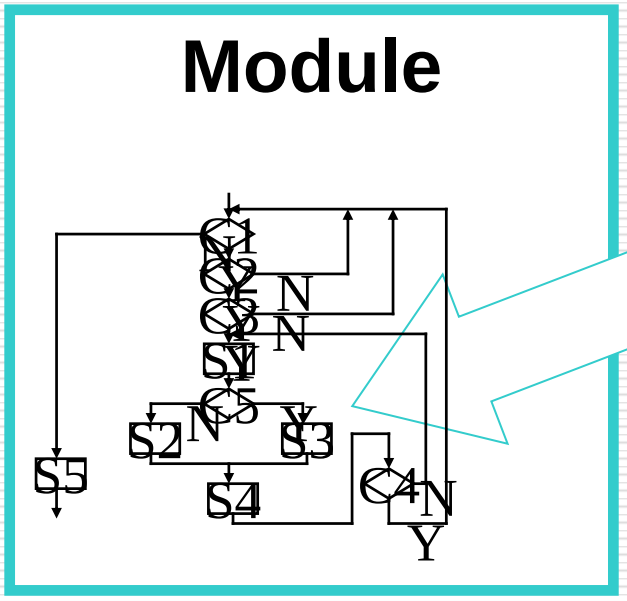
- 面向结构方法



Architecture

- 面向对象方法



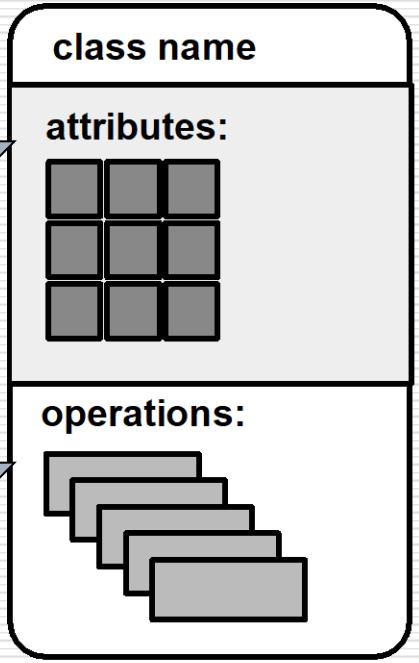


?

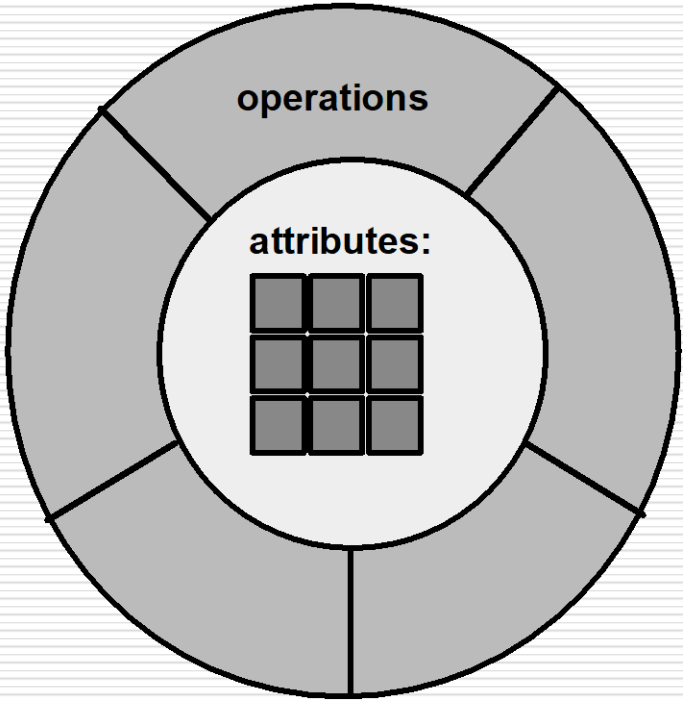
模块设计

形象理解

Class



?



本章学习目的和内容

(1) Goal of detailed design

(2) Remark on structure design

(3) Remark on object oriented design

(4) Notation in detailed design

(5) 举例

Detailed design

- ✓ Procedural design
- ✓ Algorithm design
- ✓ Program design
- ✓ Module inner design

详细设计初认识

前一章介绍了软件的总体设计，本章将在总体设计的基础上进行详细的展开、精确的描述。

详细设计阶段的根本目标是如何实现所要求的系统，也即要对目标系统进行精确描述，为编码阶段的程序书写做准备。

详细设计阶段的任务还不是具体地编写程序，而是要设计出程序的“蓝图”，程序员根据这个蓝图写出实际的程序代码。因此，详细设计时应该考虑程序代码的质量。即衡量程序的质量不仅要看它的逻辑是否正确，性能是否满足要求，更主要的是要看它是否容易阅读和理解。

详细设计

软件定义阶段定义了问题结构，叫作软件设计的一级蓝图。可用系统流程图表示、或数据流图表示、或用结构化语言表示、或以形式化软件设计语言表示。

软件总体设计确定了软件结构，即确定模块的划分、模块间的接口。可称作软件设计二级蓝图。用结构图、Jackson结构图、Warnier图来表示、或用HIPO图来表示。

软件详细设计（也称软件算法设计、软件过程设计、软件逻辑设计）确定每个软件模块的实现算法，可称软件设计的三级蓝图。可用程序流程图描述、或用伪码描述。

详细设计 (summary)

- 详细设计是给出软件结构中各模块的内部过程描述。
- 确定软件各个组成部分内的算法以及各部分的内部数据组织。
- 选定某种过程的表达形式来描述各种算法。

KEY:

Detailed design = data structure+ algorithm

软件设计的步骤

（1）细化总体设计的结构图。

（2）为每一个模块选定算法，选择描述工具详细描述。

（3）为每一个模块确定使用数据组织。

（4）确定模块接口细节。

（※内部接口—模块间接口。※外部接口—用户界面，外部软件接口，与硬件接口。※输入输出数据。※局部数据）

（5）为模块写出测试用例

（6）编写详细设计说明书。

详细设计学习掌握要点

目的： 对于软件结构图（SC图或HC图）中的每一个*模块*，确定使用的算法和块内数据结构，并用某种选定的表达工具给出清晰的描述。

原则： （1）模块的逻辑描述要清晰易读，正确可靠

（2）采用结构化设计方法，改善控制结构，降低程序的复杂程度，从而提高程序的可读性，可测试性，可靠维护性。

描述工具： (1) 图形工具

(2) 表格工具

(3) 语言工具

详细设计工具

- 图形工具

  将过程细节用图来表示，在图中，逻辑结构用具体的图形表示

- 列表工具

  利用表来表示过程细节，表列出了各种操作和相应的条件

- 语言工具

  用类语言（伪码）表示过程的细节，很接近编程语言

结构程序设计(再评述)

结构程序设计的概念最早有E.W.Dijkstra提出。

- 结构程序设计三种基本的控制结构是“顺序”、“选择”和“循环”。流程图略。

- 结构程序设计比较统一、普遍接受的定义是：

结构程序设计是一种设计程序的技术，它采用“自顶向下，逐步求精”的设计方法，以及单入口单出口的控制结构”。

结构化设计

- 传统的设计技术和旧观念：
  - 强调设计的随意性，具有浓厚的个人色彩.
  - 追求程序效率和个人设计技巧
- 新的设计思想和风格：
  - 清晰第一
  - 使用标准的、规范的控制结构
  - 逐步细化

结构程序设计(再评述)

在总体设计阶段采用“自顶向下，逐步求精”的方法，可以把一个复杂问题的解法分解和细化成一个由许多模块组成的层次结构的软件系统。在详细设计或编码阶段采用“自顶向下，逐步求精”的方法，可以把一个模块的功能逐步分解细化为一系列具体的处理步骤或某种高级语言的语句 。

结构程序设计的优点：

（1）可以显著提高软件开发工程的成功率和生产率。

（2）用先全局后局部、先整体后细节、先抽象后具体的逐步求精过程开发出的程序有清晰的层次结构，因此容易阅读和理解。

结构程序设计(再评述)

（3）不使用GO TO 语句。仅使用单入口单出口的控制结构，使得程序的静态结构和它的动态执行情况比较一致。因此，程序容易阅读和理解，开发时也比较容易保证程序的正确性，即使出现错误也比较容易诊断和纠正。

（4）控制结构有确定的逻辑模式，编写程序代码只限于使用很少几种直截了当的方式，因此源程序清晰流畅，易读易懂而且容易测试。

（5）程序清晰和模块化，使得在修改和重新设计一个软件时，可以重用的代码量最大。

（6）程序的逻辑结构清晰，有利于程序正确性证明。

结构程序设计(再评述)

结构程序设计的缺点：

需要的存储容量和运行时间都有一些增加。此外，现有的许多程序设计语言是非结构化的语言，并不提供上述的单入口单出口的基本控制结构。但是，由于硬件技术的飞速进步，程序需要的存储容量和运行时间稍有增加，在今天对绝大多数应用领域已经不是严重问题。如果使用非结构化语言编写程序，则可以利用GO TO 语句实现上述基本控制结构，虽然形式上程序中有GO TO 语句，却仍然能够体现出结构程序设计的基本精神。

Why structure design ?

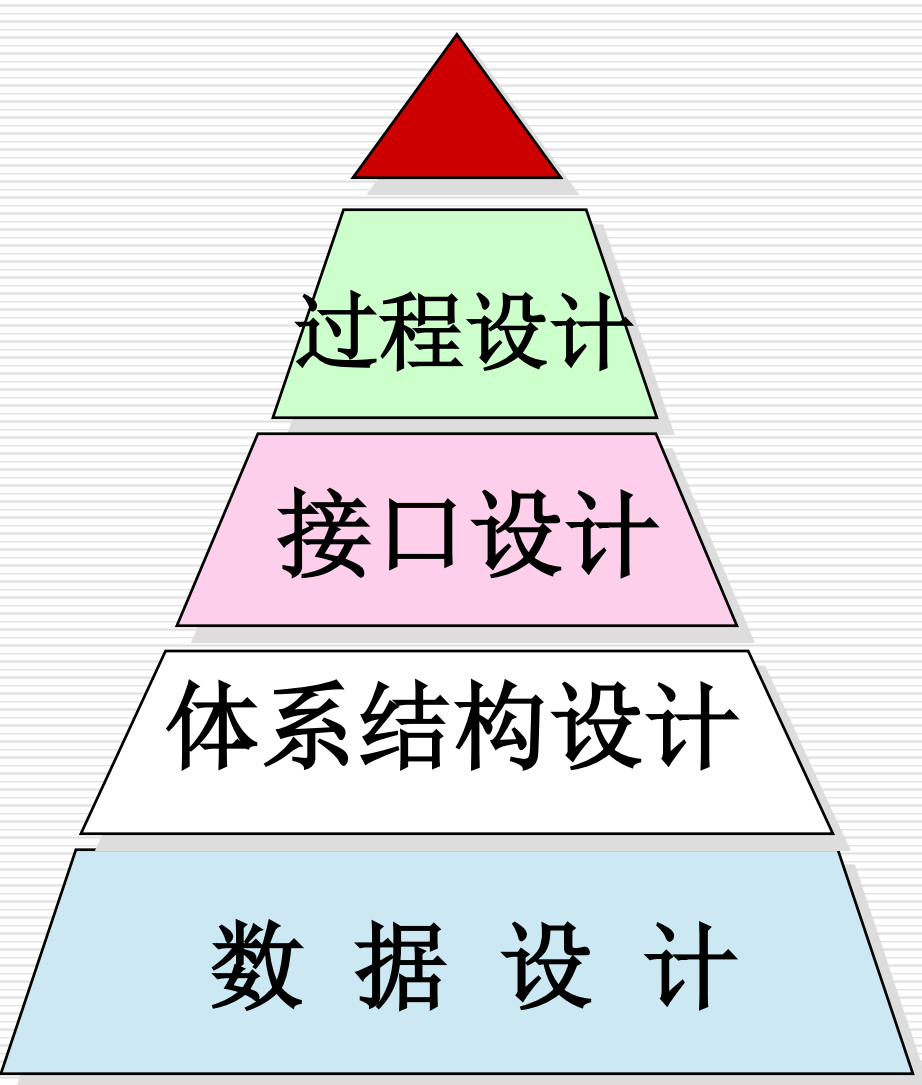
- Practiced informally since programming began
- Thousands of systems have been developed using this approach
- Supported directly by most programming languages
- Most design methods are functional in their approach
- CASE tools are available for design support

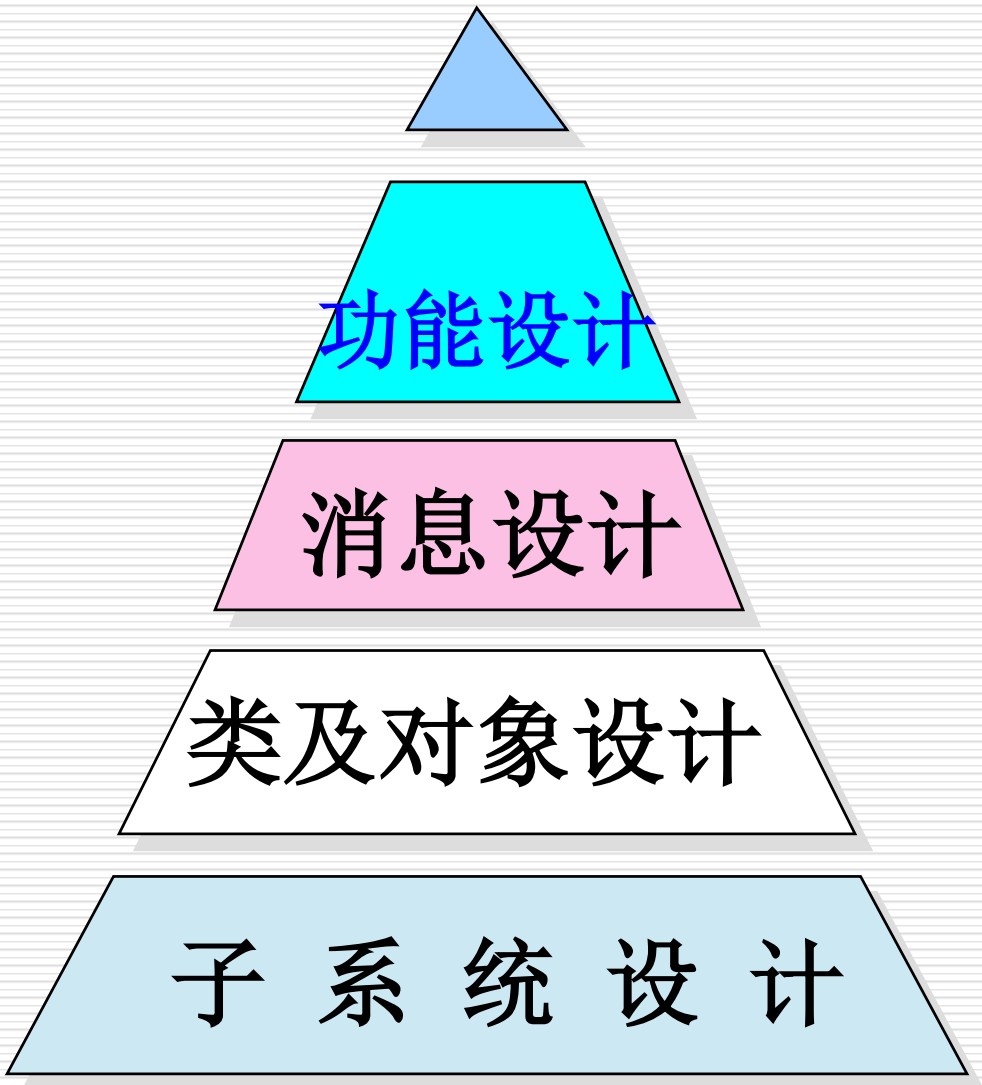
Method deficiencies

- They are guidelines rather than methods in the mathematical sense. Different designers create quite different system designs

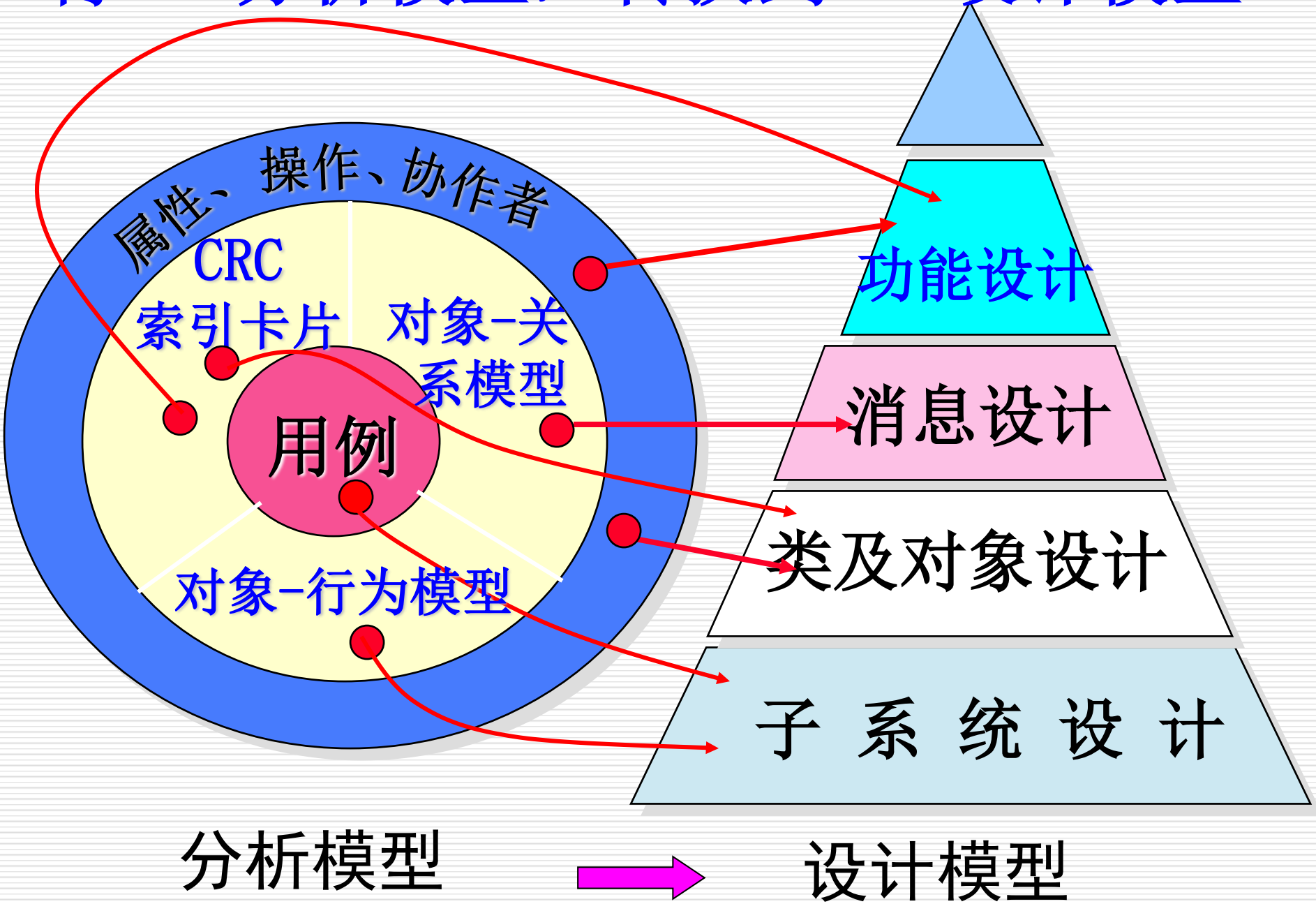
- They do not help much with the early, creative phase of design. Rather, they help the designer to structure and document his or her design ideas

Remark on Object Oriented Design

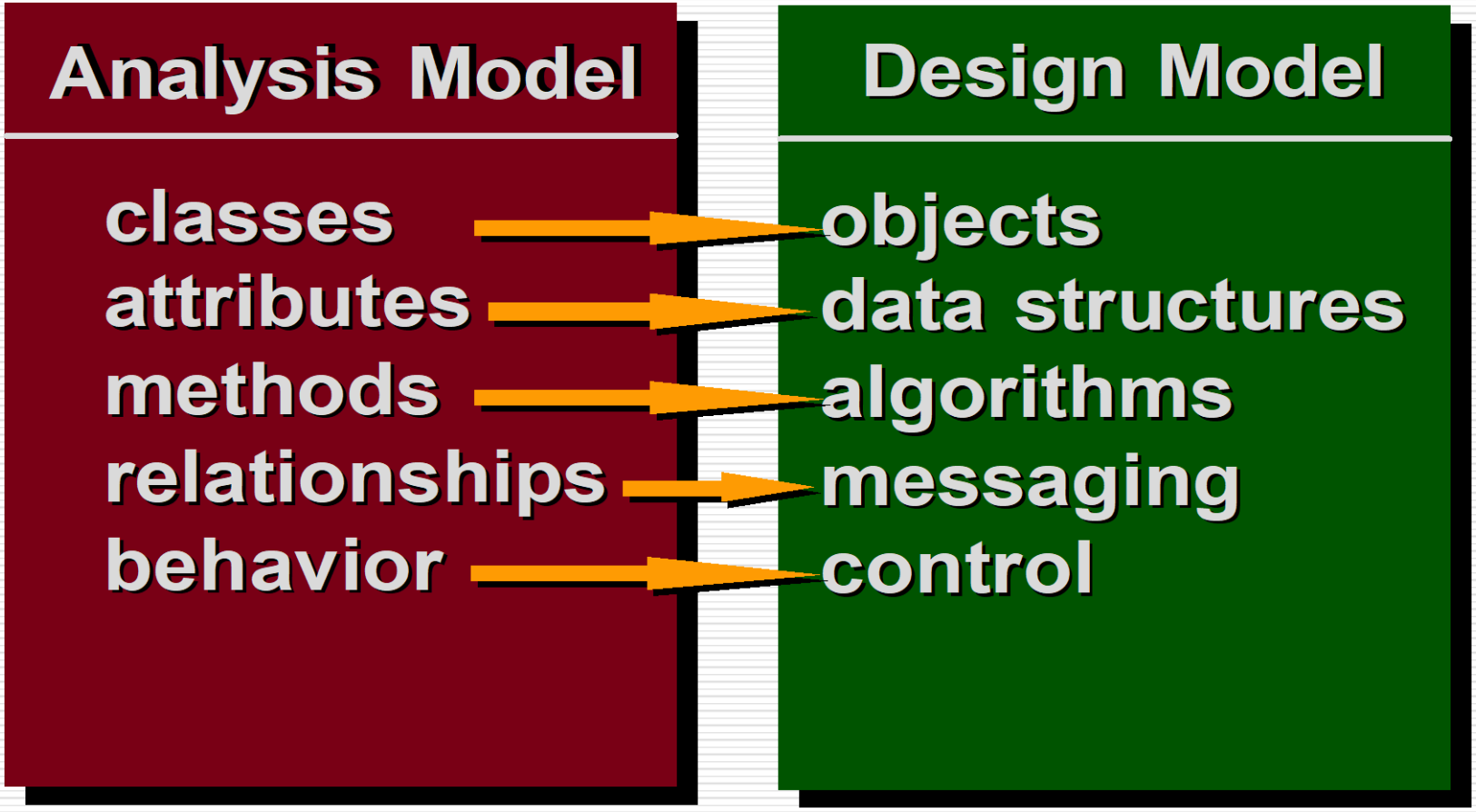




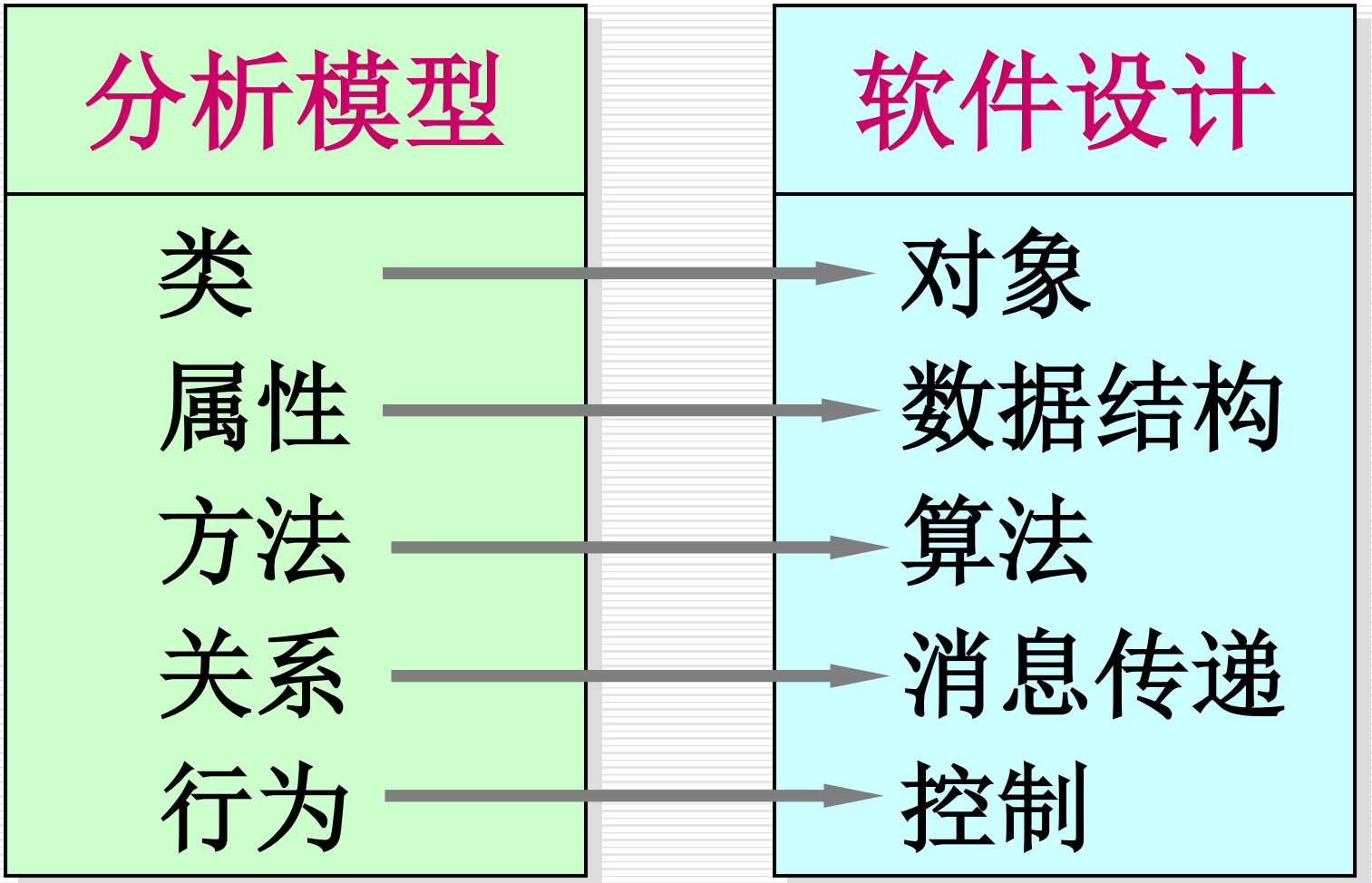
将OO分析模型，转换到OO设计模型



OOA and OOD


Analysis Model
classes
attributes
methods
relationships
behavior
Design Model
objects
data structures
algorithms
messaging
control

分析模型 —> 设计模型



Design in order to

- ✓ find a good object
- ✓ abstraction
- ✓ encapsulation
- ✓ association
- ✓ interaction
- ✓ inheritance
- ✓ reuse

Architecture phase

Detailed phase

不刻意分总体设计和详细设计（**UML)**

面向对象详细设计技术是什么呢？

软件重用（复用）Reuse

软件重用的层次

(1) 知识重用

(2) 方法和标准重用

(3) 软件成分重用

可重用的软件成分

- 项目计划
- 成本估算
- 体系结构
- 需求模型和规格说明
- 设计方案
- 源代码
- 用户文档和技术文档
- 用户界面
- 数据

面向对象设计的重用机制

传统上：

- 内部函数

面向对象：

- 内部类（类构件）
  - 实例重用
  - 继承重用
  - 多态重用

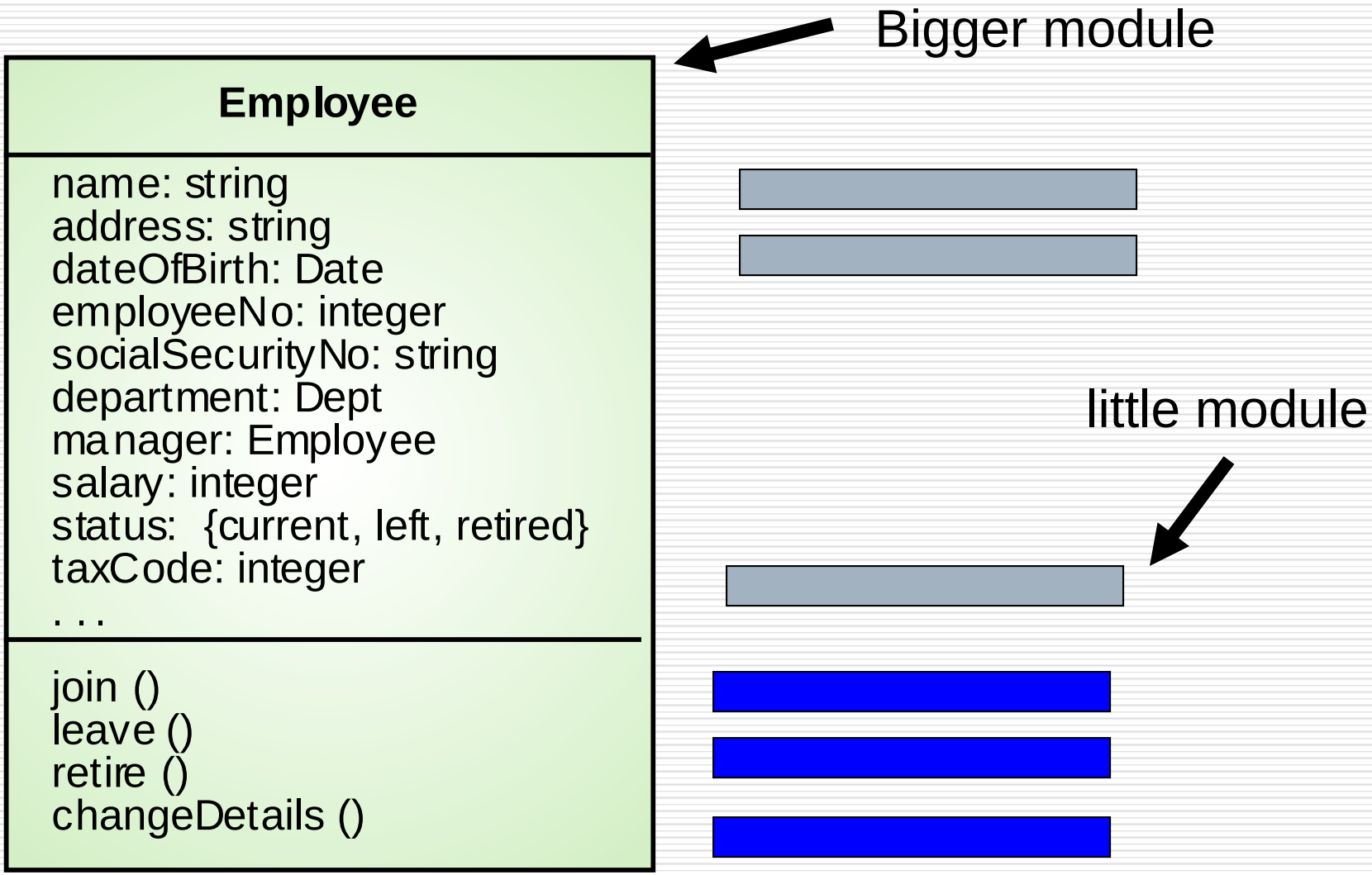
举例

classe student {…}
student one_student;

class x { … };
class y {… };
class z: x,y ;

int max(x, y);
float max(x,y);
char max(x,y);

Module 的再认识

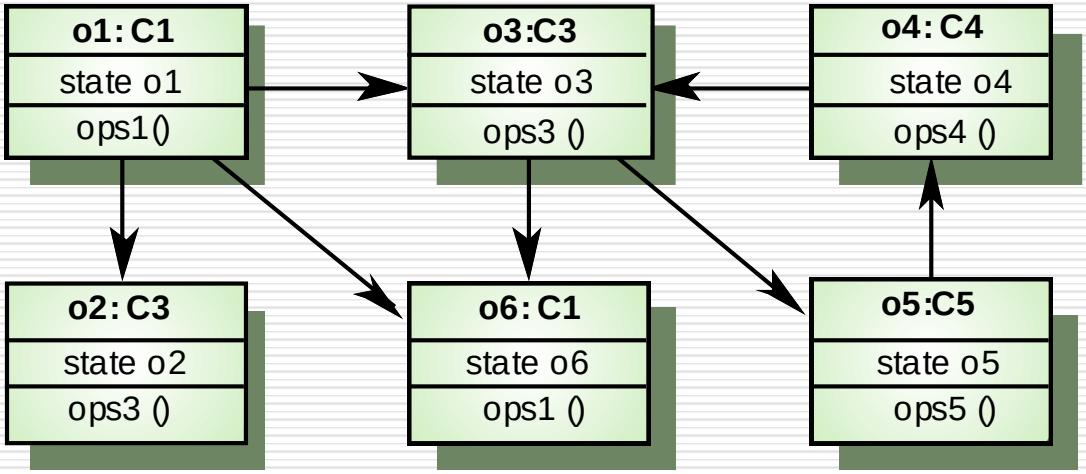

Bigger module
Employee
name: string
address: string
dateOfBirth: Date
employeeNo: integer
socialSecurityNo: string
department: Dept
manager: Employee
salary: integer
status: {current, left, retired}
taxCode: integer
. . .
join ()
leave ()
retire ()
changeDetails ()
little module

The Challenge of OO Design

- To create re-usable OO software a designer must:
  - Find pertinent objects
  - Factor them into classes of the right granularity
  - Define class interfaces and inheritance hierarchies
- The design should be specific to the problem at hand but general enough to address future problems and requirements.
- All design methods strive for:
  - Abstraction
  - Information hiding
  - High cohesion; Low coupling
  - Modularity
- But only OOD can achieve all four without complexity or compromise.

Characteristics of OOD

- Objects are abstractions of real-world/system entities
- Objects manage themselves
- Objects are independent and encapsulate state and representation
- System functionality is expressed in terms of object services
- Shared data areas are eliminated. Objects communicate by message passing
- Objects may be distributed and may execute sequentially or in parallel



Mixed-strategy design

- Although it is sometimes suggested that one approach to design is superior, in practice, an object-oriented and a functional-oriented approach to design are complementary

- Good software engineers should select the most appropriate approach for whatever sub-system is being designed

详细设计的描述

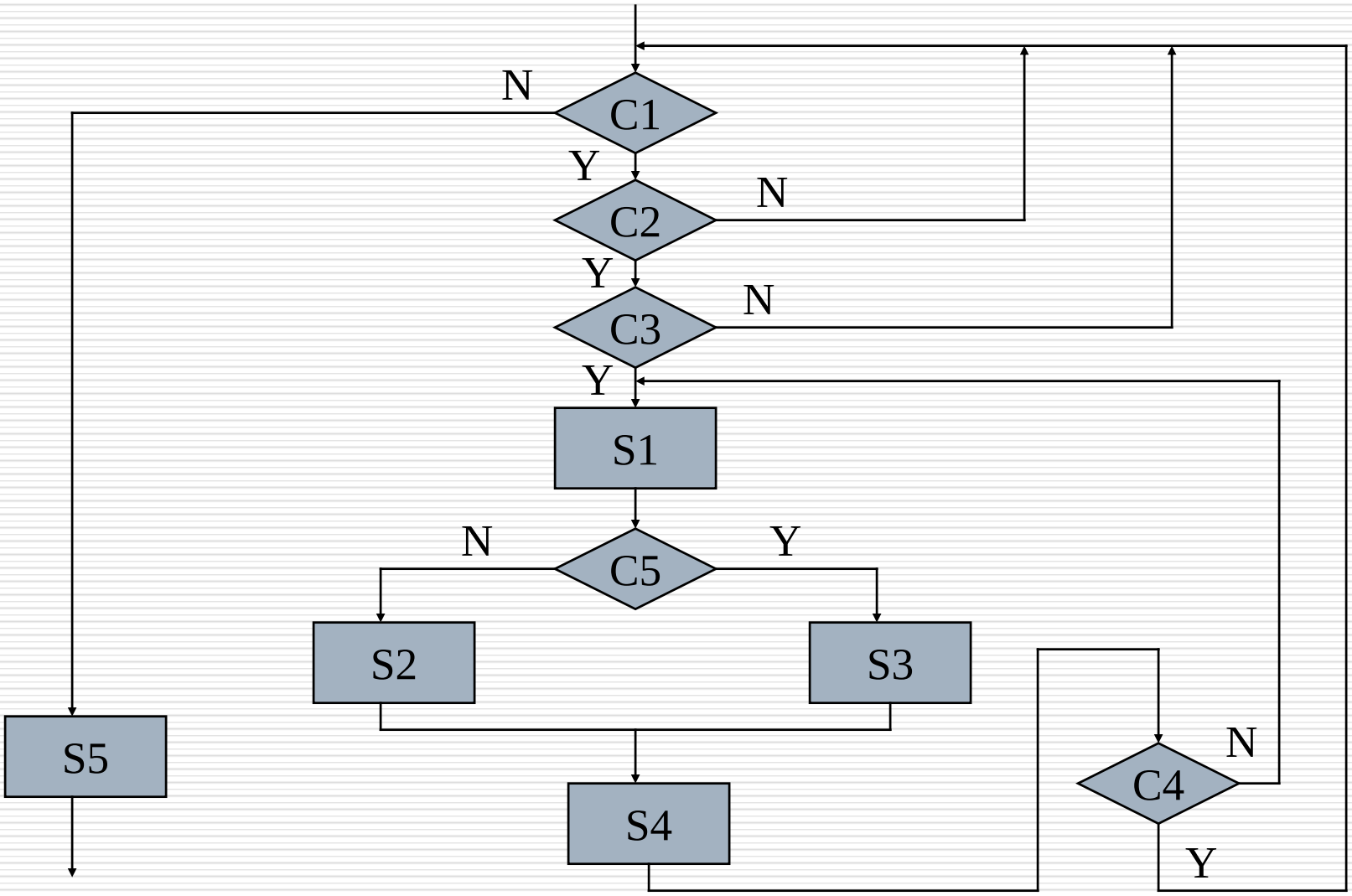
tools：

(1) Graphical notation

(2) Tabular notation

(3) Language notation

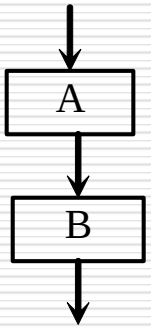
Program Flow chart



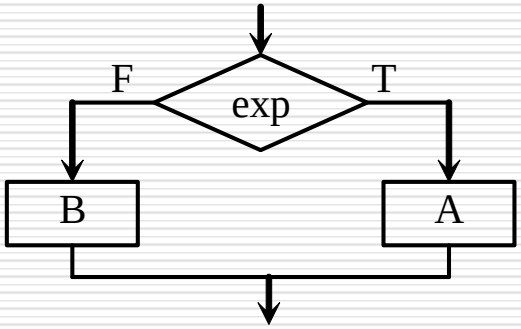
程序流程图

- 程序流程图也称为程序框图，程序流程图使用三种基本控制结构是：
  - ✓ 顺序
  - ✓ 选择
  - ✓ 循环

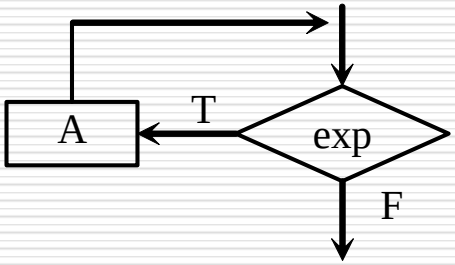
Three kinds of control structure



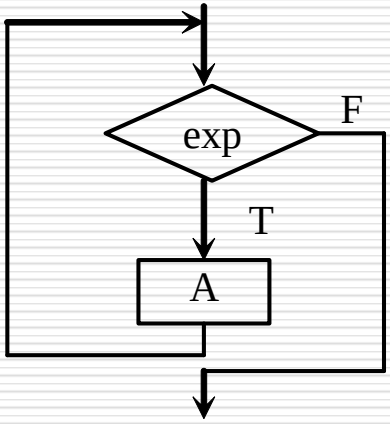
(a) sequence



(b) (b) choose



or



(c) loop

程序流程图

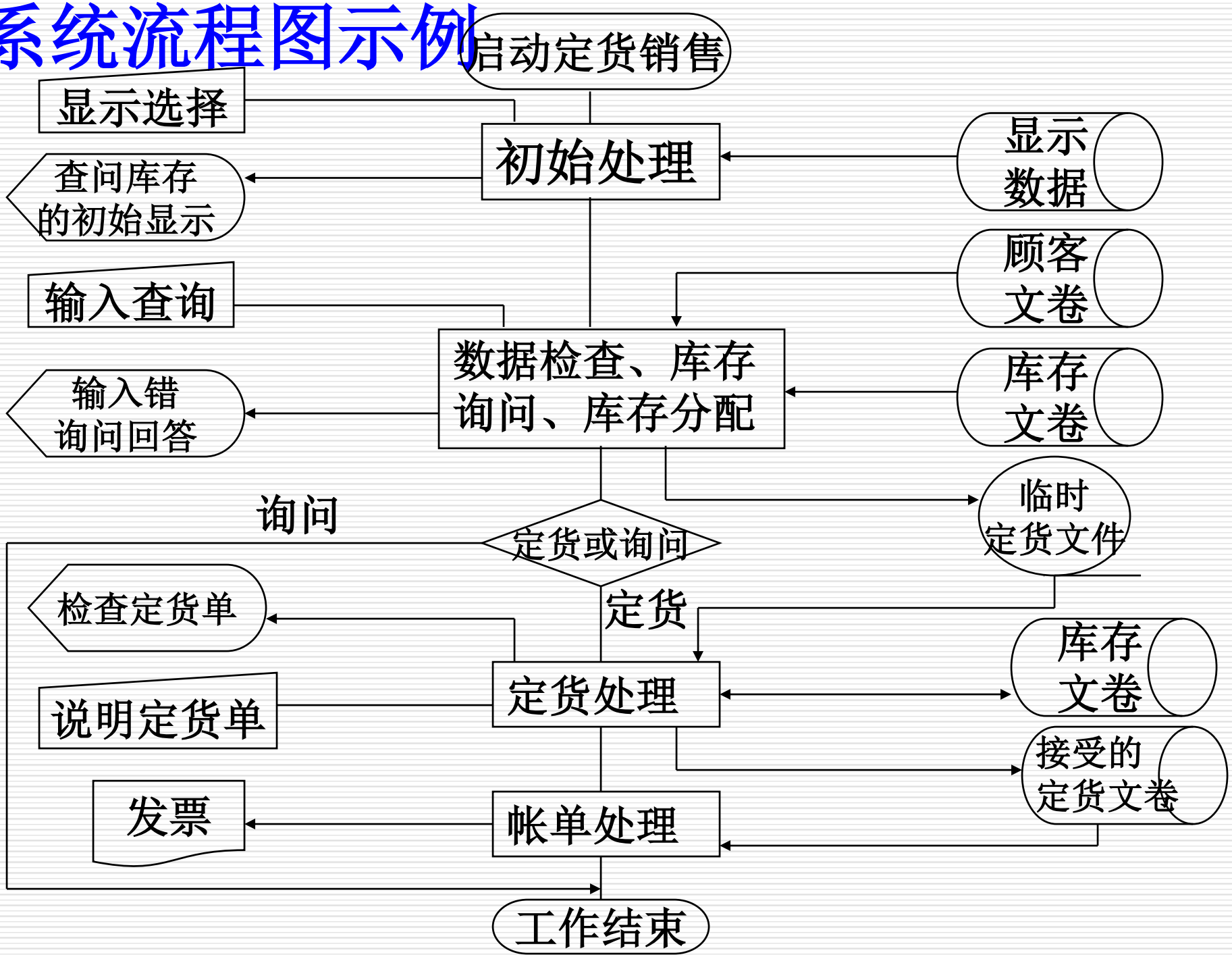
- **流程图的优点：**

  直观、简单、易学、普及

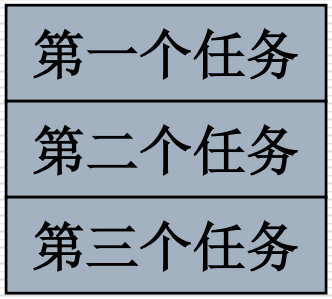
- **流程图的主要缺点：**

  - 流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。
  - 流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制。
  - 流程图不易表示数据结构。

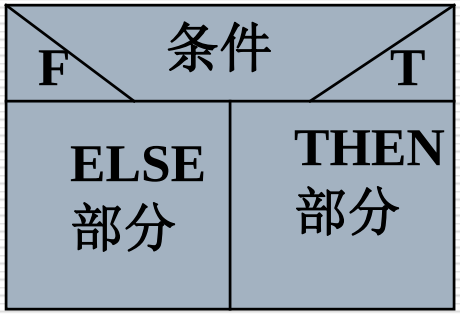
系统流程图示例


启动定货销售
显示选择
初始处理
显示
数据
查问库存
的初始显示
顾客
文卷
输入查询
数据检查、库存
询问、库存分配
库存
文卷
输入错
询问回答
临时
定货文件
询问
定货或询问
检查定货单
定货
库存
文卷
说明定货单
定货处理
接受的
定货文卷
发票
帐单处理
工作结束

N-S Diagram （N-S盒图）



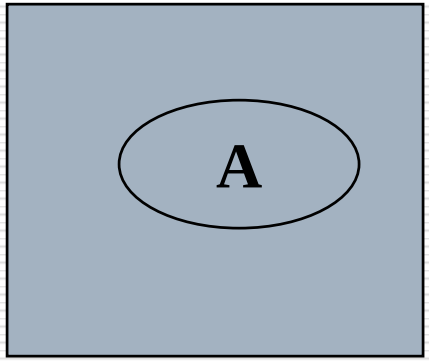
顺序



IF-THEN-ELSE分支

CASE分支

循环



调用子程序A

N-S盒图

- 研制方块图的目的是：既要制定一种图形工具，又不允许它违反结构化原则。

- 盒图具有以下特点：

（1）功能域（即某一具体构造的功能范围）有明确的规定，并且很只观地从图形表示中看出来；

（2）想随意分支或转移是不可能的；

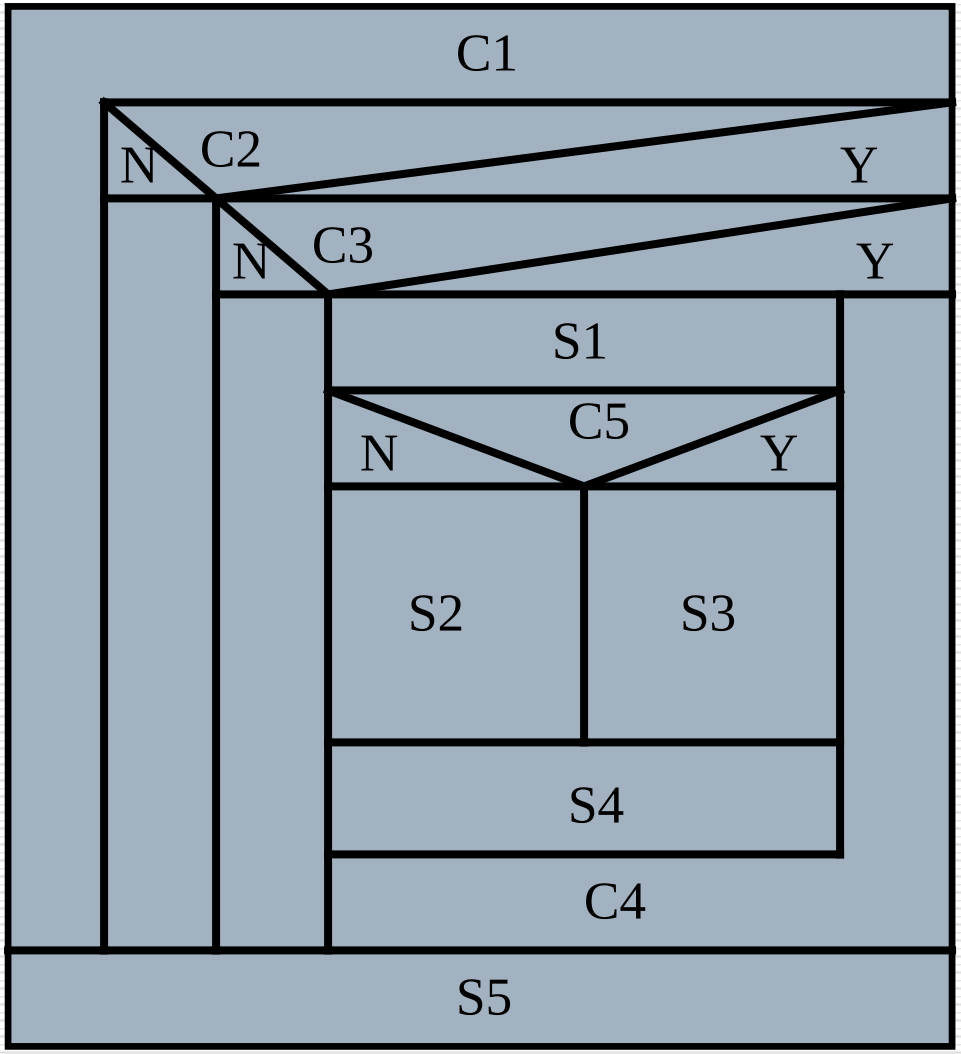
（3）局部数据和全程数据的作用域可以很容易确定；

（4）容易表示出递归结构。

结构化原则（回顾）

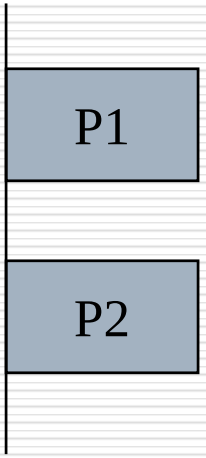
- ✓ **from top to bottom** 自顶向下
- ✓ **refinement** 逐步求精
- ✓ **single entry, single exit** 单入口单出口

An example

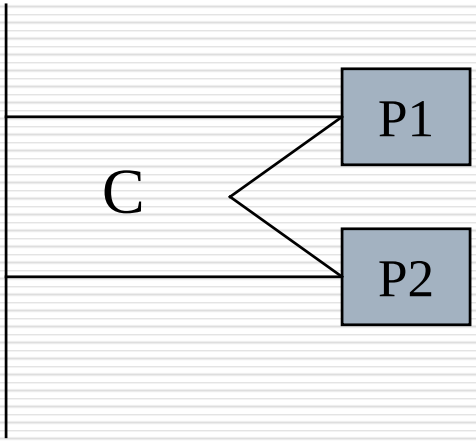


```
Do while C1
    if C2
        if C3
            Do
                S1;
                if C5
                    S3;
                else
                    S2;
                endif
                S4;
            until C4
            S5;
        endif
    endif
enddo
```

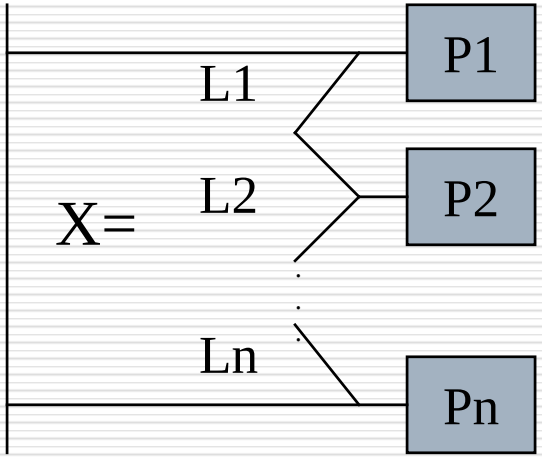
Problem Analysis Diagram (PAD图)



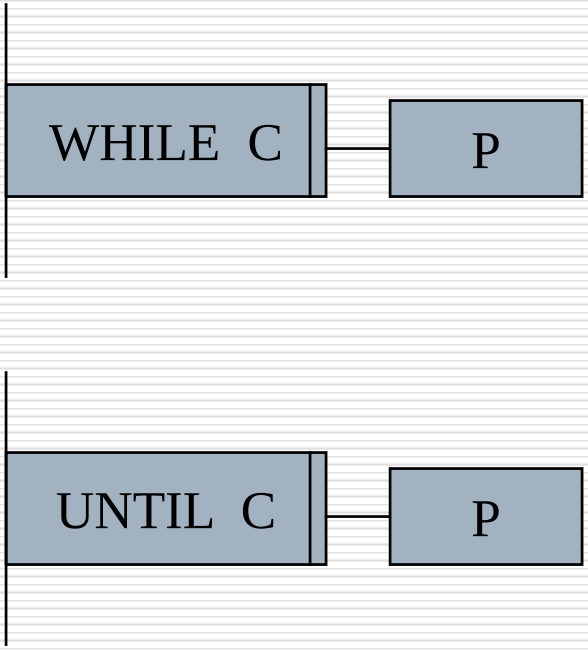
顺序



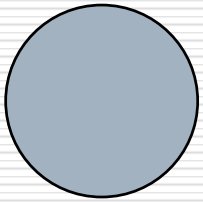
选择



CASE型选择



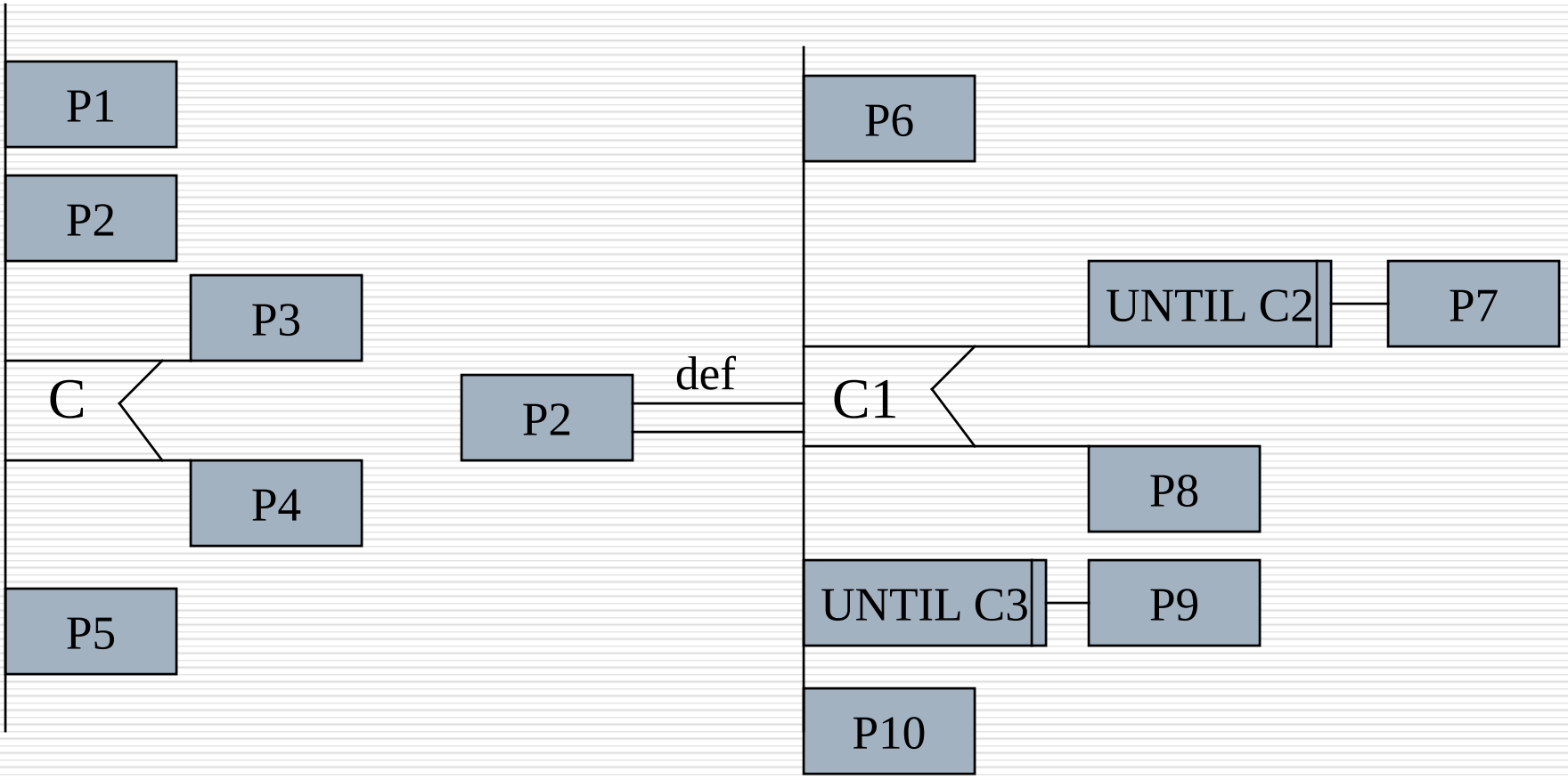
循环

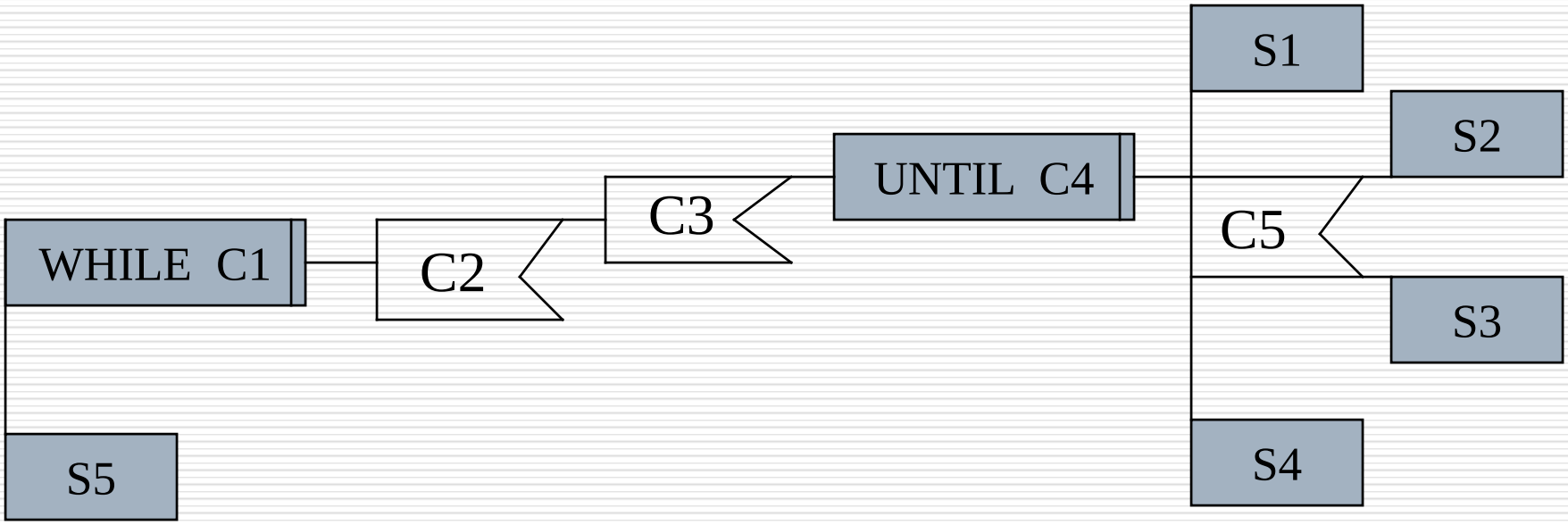


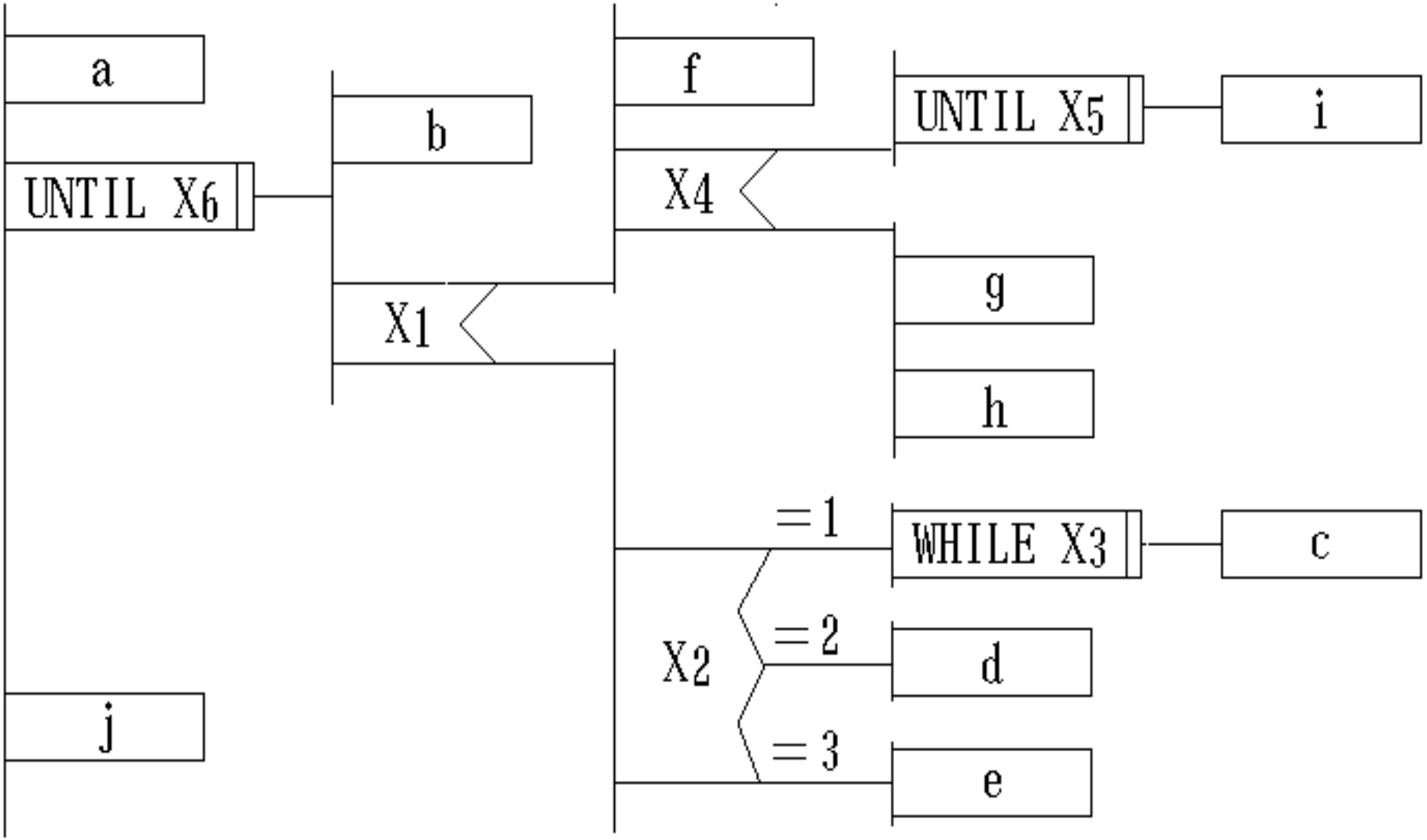
语句标号

def

定义


P1
P2
P3
C
P4
P5
P6
UNTIL C2
P7
def
P2
C1
P8
UNTIL C3
P9
P10


WHILE C1
C2
C3
UNTIL C4
C5
S1
S2
S3
S4
S5



PAD描述的示例

```
a;
do until X6
    b;
    if  X1
        f;
        if X4
            do until X5
                i;
            enddo
        else
            g;
            h;
        endif
    else
        case  X2=1   do while X3 c;
        case  X2=2   d;
        case  X2=3   e;
    endif
enddo
j;
```

PAD图的特点

- ✓ 使用表示结构化控制结构的PAD符号所设计出的程序必然是结构化程序。
- ✓ PAD图所描述的程序结构十分清晰，图中最左面的竖线是程序的主线，即第一层结构，随着程序层次的增加，PAD图逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线，PAD图中的竖线的总条数就是程序的层次数。
- ✓ PAD是二维树层次结构、自顶向下、自左到右遍历所有节点，易读、易记、易懂。

PAD图的特点

- ✓ 既可用于表示程序逻辑，也可用于描述数据结构。
- ✓ PAD描述的算法易译为高级程序语言。
- ✓ PAD图的符号具有支持自顶向下、逐步求精方法的作用。开始时设计者可以定义一个抽象的程序，随着设计工作的深入而用def符号逐步增加细节，直至完成详细设计。

Decision Table

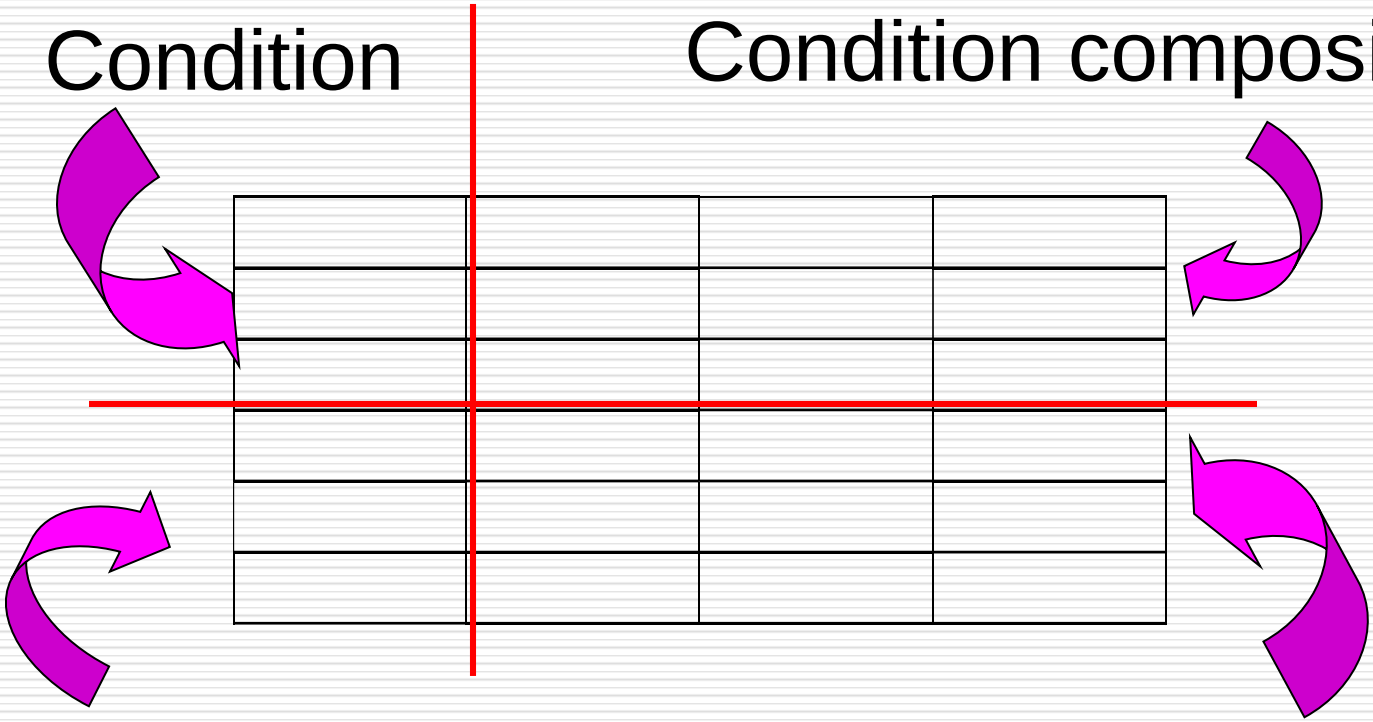
why?

复杂的条件组合与应做的动作之间的对应关系。

组成：

1. 左上部列出所有条件；
2. 左下部是所有可能做的动作；
3. 右上部为各种可能组合条件，其中每一列表示一种可能组合；
4. 右下部的每一列是和每一种条件组合所对应的应做的工作。

判断表


Condition
Condition composition
Action
Relative Action under one condition composition

Example

例：某校制定了教师的讲课课时津贴标准。对于各种性质的讲座，无论教师是什么职称，每课时津贴费一律是50元；而对于一般的授课，则根据教师的职称来决定每课时津贴费：教授30元，副教授25元，讲师20元，助教15元。

	1	2	3	4	5
教授		T	F	F	F
副教授		F	T	F	F
讲师		F	F	T	F
助教		F	F	F	T
讲座	T	F	F	F	F
50	√				
30		√			
25			√		
20				√	
15					√

判定表特点

优点：简洁

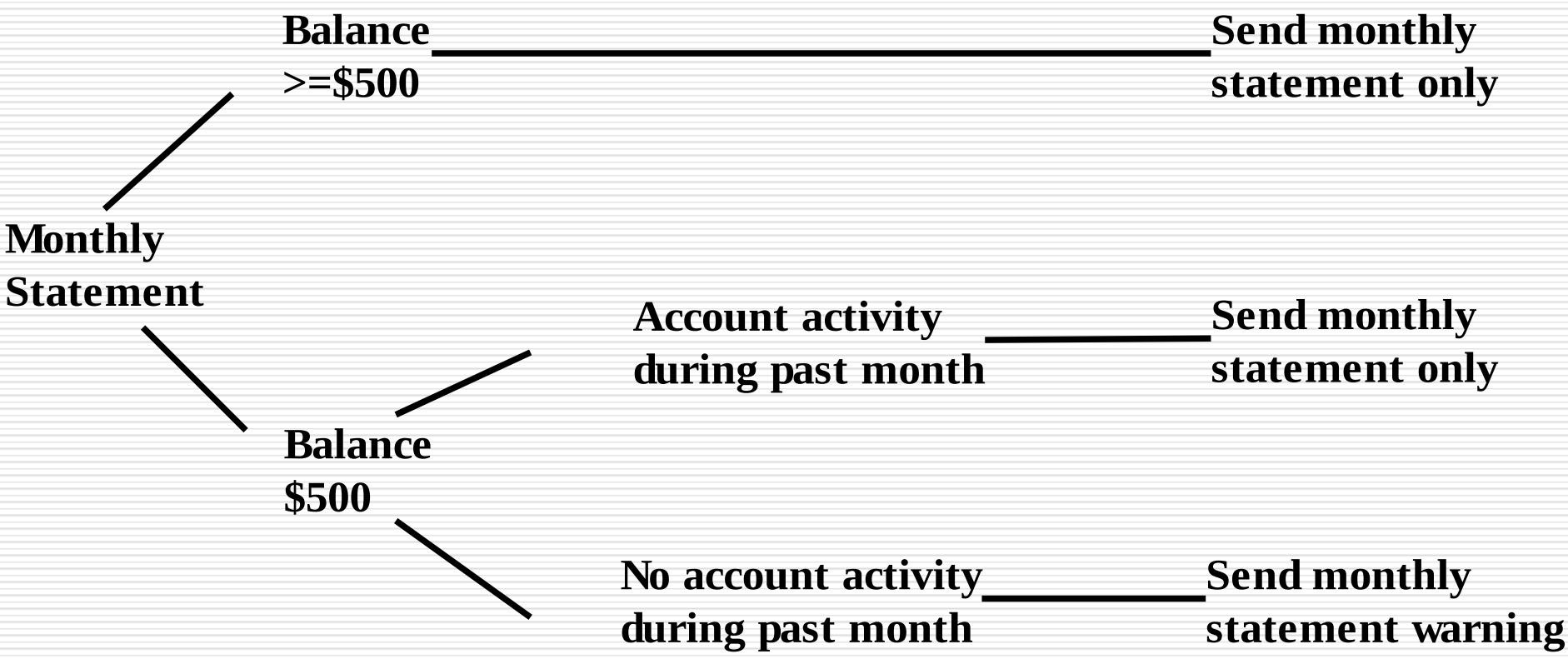
无歧义

类似卡若图

缺点：

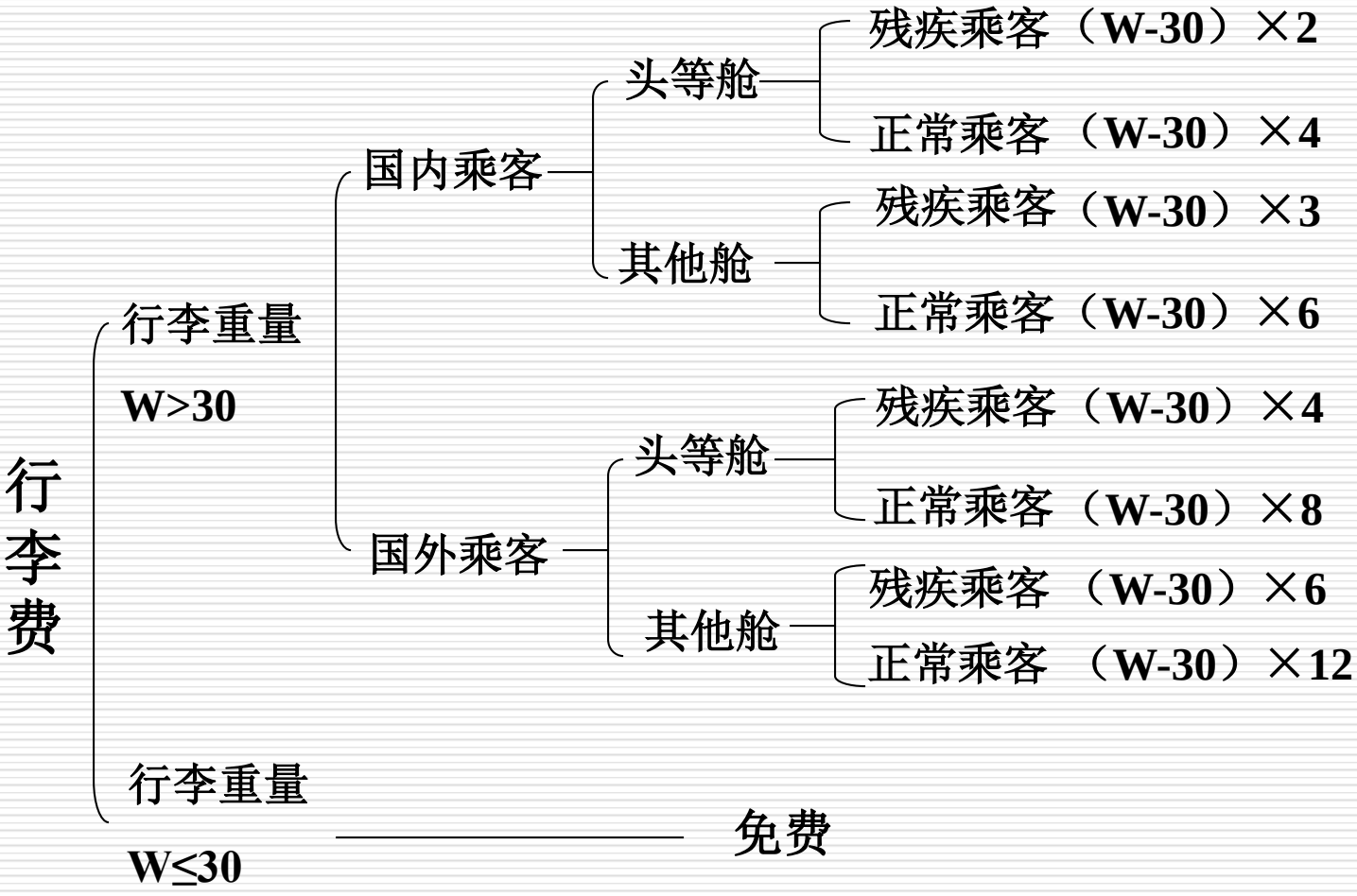
不能表示顺序和重复的处理特性。

Decision Tree

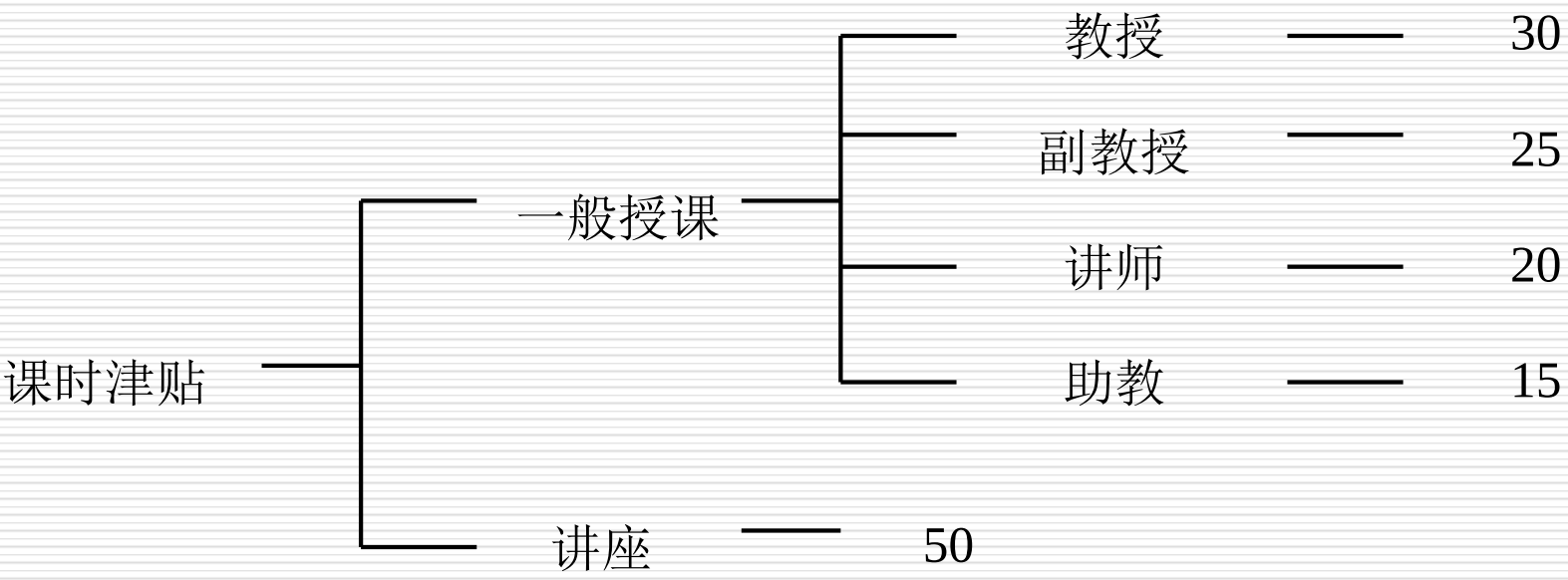


判定树

判定树是判定表的变种，形式简单易懂，更容易表示层次结构关系。但不如判定表简洁，重复次数多。



教师课时津贴判定树



简洁，与书写顺序有关树

分支选择嵌套程序的编程实现方式

方式1：多重嵌套

```
If  条件1
    操作1;
else
    if 条件2
       操作2;
    else
       if 条件3
          操作3;
       else
          if 条件4
             操作4;
          else
             if 条件5
                操作5;
             else
                if 条件6
                   操作5;
                else
                   ……
```

缩进嵌套

分支选择嵌套程序的编程实现方式

方式2：条件逻辑组合， ∧，∨，¬

If 条件1 ∧ 条件2 ∧ 条件3 ∧ 条件4 ∧ 条件5 ∧ 条件6 ∧ ……
动作1；
If 条件1 ∨ 条件2 ∨ 条件3 ∨ 条件4 ∨ 条件5 ∨ 条件6 ∨ ……
动作2；
If ¬条件1 ∧ ¬条件2 ∧ ¬条件3 ∧ ¬条件4 ∧ ¬条件5 ∧ ¬条件6 ∧ ……
动作3；
If ¬条件1 ∨ ¬条件2 ∨ ¬条件3 ∨ ¬条件4 ∨ ¬条件5 ∨ ¬条件6 ∨ ……
动作4；
If 条件1 ∧ 条件2 ∧ 条件3 ∧ ¬条件4 ∧ ¬条件5 ∧ ¬条件6 ∧ ……
动作5；
If 条件1 ∨ 条件2 ∨ 条件3 ∨ ¬条件4 ∨ ¬条件5 ∨ ¬条件6 ∨ ……
动作6；
If ……

执行效率，软件工程提倡哪个？

PDL (Program Design Language)

- PDL是一种用于描述功能模块的算法设计和加工细节的语言。称为设计程序用语言。伪码语言。
- 伪码的语法规则分为“外语法”和“内语法”。
- PDL具有严格的关键字外语法，用于定义控制结构和数据结构，同时它的表示实际操作和条件的内语法可使用自然语言的词汇。

示例：拼词检查程序

```
PROCEDURE spellcheck IS
    BEGIN
        split document into single words;
        look up words in dictionary;
        display words which are not in dictionary;
        create a new dictionary;
    END spellcheck
```

划分算法Partition_Algorithm( )

输入：划分映射数组P
输出：已精化的划分映射数组P

```
while
   p←GetUnrefinedPartition();          /*得到未精化的划分块的块号p*/
   if |w(p)-W(p)| is in the range of error
                                       /*w(p)为p的实际权值，W(p)为p的所需权值*/
      Set p to be refined;             /*将p设为已精化*/
   else if w(p)<W(p)
      Find unrefined q∈AdjPartition(p) whose w(q)-W(q) is the largest;
                                       /*在p的未精化的邻接块中找q满足w(q)-W(q)最大*/
      Find v∈BoundaryVeterx(q) which gain(v)p is the largest;
                                       /*在q的边界点中找一点v满足gain(v)p 最大*/
      P[v]←p;
      w(p)←w(p)+w(v);
      w(q)←w(q)-w(v);
   else if w(p)>W(p)
      Find unrefined q∈AdjPartition(p) whose w(q)-W(q) is the smallest;
                                       /*在p的未精化的邻接块中找q满足w(q)-W(q)最小*/
      Find v∈BoundaryVeterx(p) which gain(v)q is the largest;
      P[v]←q;
      w(p)←w(p)-w(v);
      w(q)←w(q)+w(v);
   endif
endwhile
```

并行程序任务图DAG图调度算法Scheduling_Algorithm( )

输入：DAG图，处理器数

输出：总执行时间total_time, 调度策略

*L*1: L←{i | indegree(i)=0, 1≤ i ≤ N }

*L*2: ts(i)=0, te(i)=0, 1≤ i ≤ N, τ_pidle(x)=0, 0≤ x ≤ P

*L*3: ε←ϕ

*L*4: Do until L empty {

*L*5: (1) For each idle processor p(x)

*L*6: i←Sched(L, p(x))

*L*7: if i > 0

*L*8: L←L–{i}

*L*9: ε←ε+{i}

*L*10: ts(i)← max(ts(i), τ_pidle(x))

*L*11: (2) For each executable task j∈ ε

*L*12: ε←ε – {j}

*L*13: te(j)← ts(j)+ T comp(j)+ T comm(j)

*L*14: τ_pidle(x) ← te(j)

*L*15: For each immediate successor k of task j

*L*16: ts(k)← max(ts(k), τ_pidle(j))

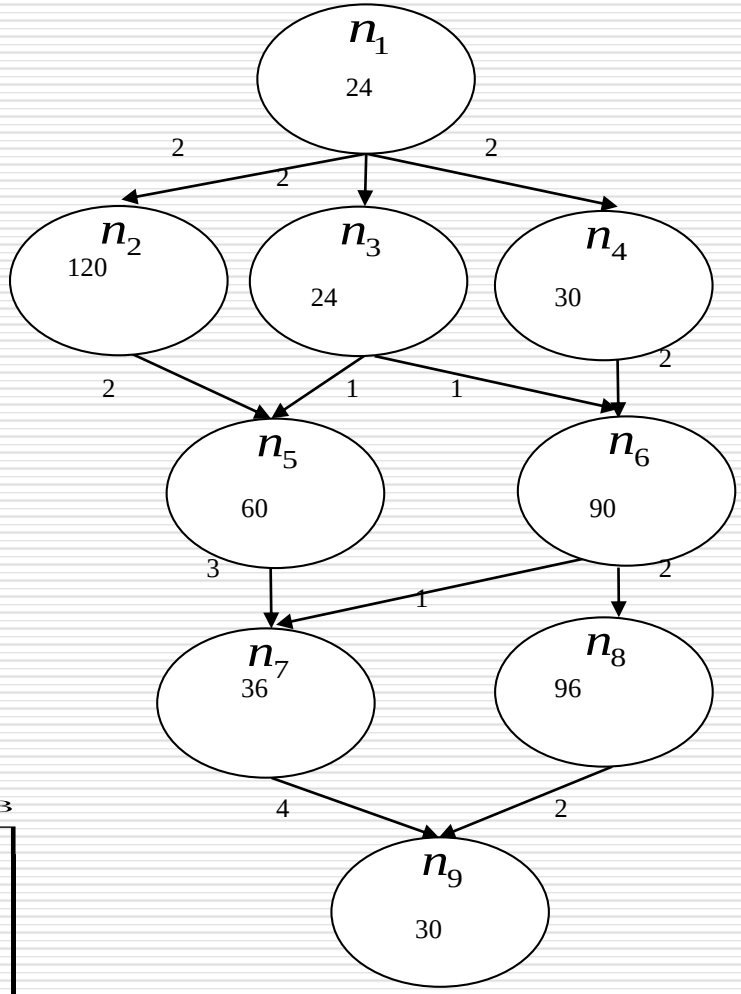
*L*17: indegree(k) ← indegree(k)–1

*L*18: if indegree(k)=0

*L*19: L←L+{k}

*L*20: }

*L*21: total_time ← max{ te(j), 1≤ i ≤ N }



	p_1	p_2	p_3
n_1	3	6	8
n_2	15	12	20
n_3	4	6	3
n_4	10	6	15
n_5	12	6	5
n_6	9	10	6
n_7	6	3	12
n_8	6	16	8
n_9	5	3	6

total_time=33

HLD Algorithm
Begin
Step 1: Construct a priority based task sequence ξ;
do
Step 2: $t_i \leftarrow$ first unscheduled task in ξ;
Step 3: **for** (all p_k in P)
Call Get_EFT (t_i , p_k) and record the $F_{i,k}$;
Step 4: Schedule t_i on processor p_k with minimum $F_{i,k}$;
while (there exists an unscheduled task in ξ)
End

Get_EFT (t_i , p_k)
Begin
Repeat
{
Step s1: $S_{i,k} \leftarrow$ Find start time of t_i on p_k;
Step s2: $F_{i,k} \leftarrow S_{i,k} + w_{i,k}$;
Step s3: $M_i \leftarrow$ Find MIIP of t_i for p_k ;
Step s4: **if** (M_i does not exist or is already scheduled on p_k)
Return $F_{i,k}$;
else if (suitable slot exists for M_i on p_k)
{ $F_{j,k} \leftarrow$ Find the finish time of $M_i (= t_j$ say) on p_k;
if ($F_{j,k}$ is less than $S_{i,k}$)
Duplicate M_i on p_k ; // duplication successful, repeat the loop for next MIIP
else Return $F_{i,k}$;
}
else Return $F_{i,k}$;
}
End

Fig. 2. Pseudo code for HLD algorithm.

PDL语言具有下述特点

（1）PDL虽然不是程序设计语言，但是它与高级程序设计语言非常类似，只要对PDL描述稍加变换就可变成源程序代码。因此，它是详细设计阶段很受欢迎的表达工具。

（2）用PDL写出的程序，既可以很抽象，又可以很具体。因此，容易实现自顶向下逐步求精的设计原则。

（3）PDL描述同自然语言很接近，易于理解。

（4）PDL描述可以直接作为注释插在源程序中，成为程序的内部文档。这对提高程序的可读性是非常有益的。

（5）PDL一种半结构化的描述，与程序结构相似，因此自动产生程序比较容易。

PDL的缺点是不如图形描述形象直观，因此人们常常将PDL描述与一种图形描述结合起来使用。

一个详细设计举例

文档编号：NameAdd’98_Development_02

版本号：1.0

文档名称：详细设计说明书

项目名称：“通信录软件”

负 责 人：谭刚

编　　写：谭刚

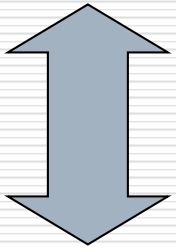
校　　对：王磊

审　　核：王虎

批　　准：王虎

开发单位：清华大学计算机系计45班软件开发小组

UML notation



?

Detailed design

Rose设计举例

□ Rose UML逻辑模型设计

结合之后的“课程实习”，各自自愿实践

(1)共享单车手机APP的需求描述

学号 <= 2251022

(2)手机私家车拼车软件系统的需求描述

251024 <= 学号 <=2251753

(3)手机公园导游软件系统的需要描述

2251762 <= 学号 <= 2252537

(4)城市公交车无人驾驶系统的需求描述

2252538 <= 学号 <= 2253331

（5）智能衣服电子系统的需求描述

2253334 <= 学号 <2299999

Homework 2024-10-28/2024-11-04

Page 140-141

T2
T3
T4
T5