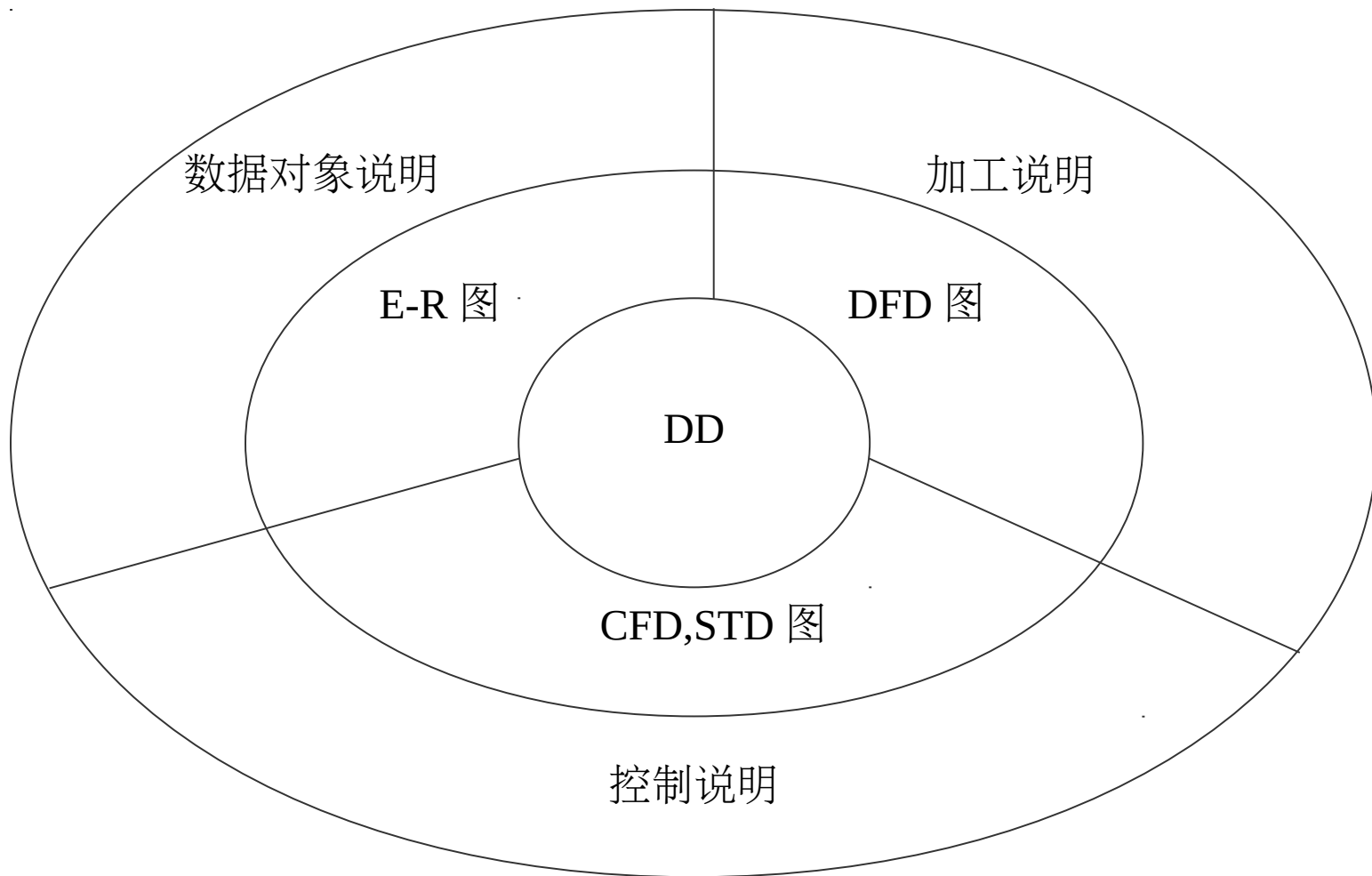


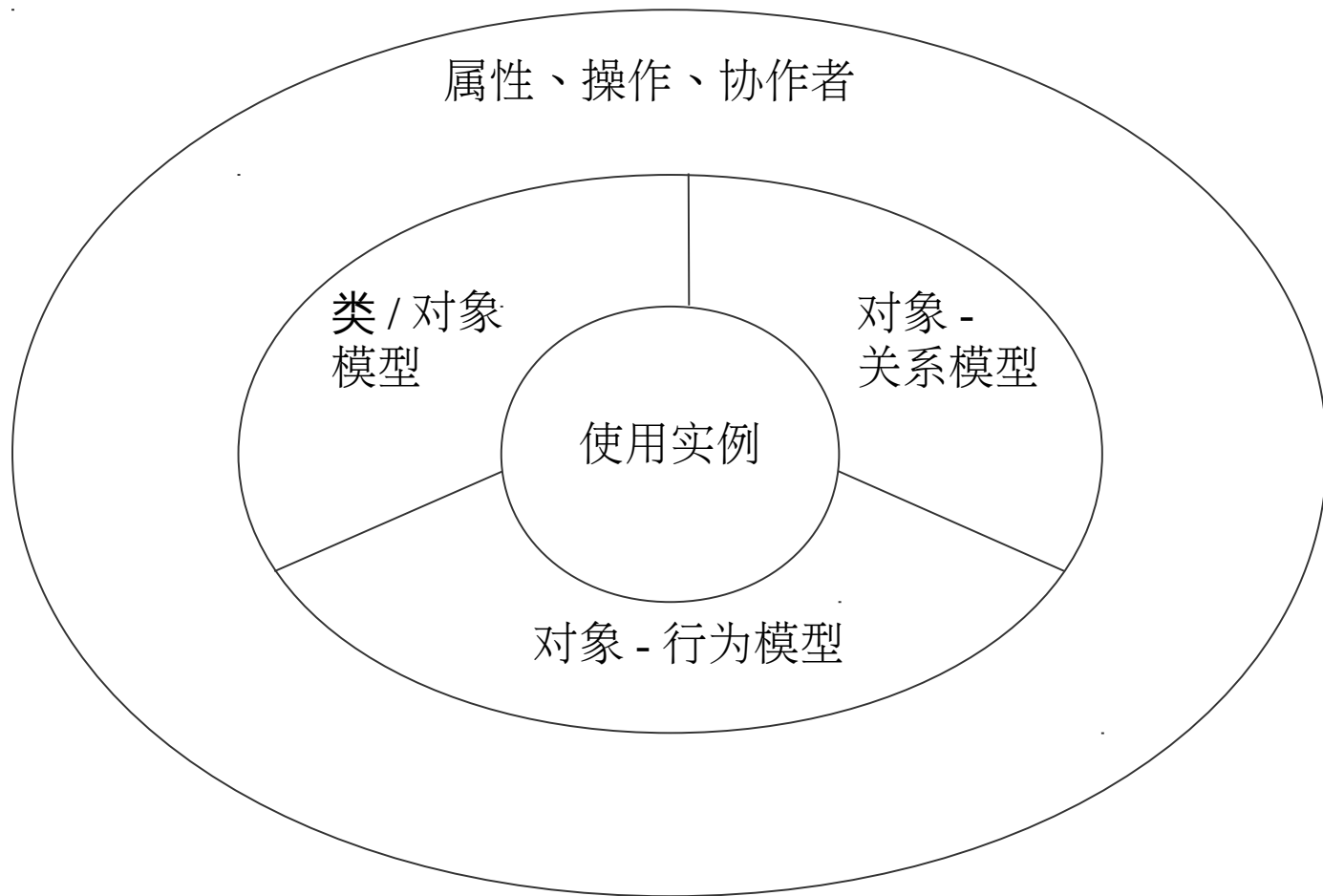
面向数据流的软件分析与设计

上海海事大学物流研究中心

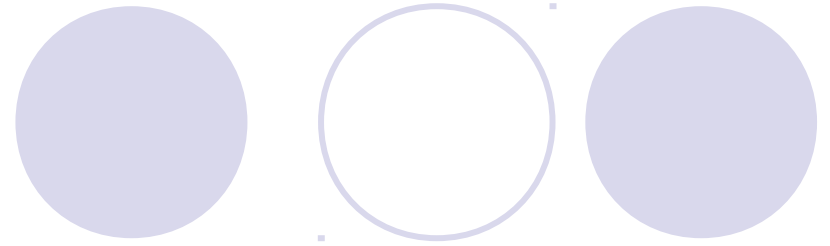
结构化分析模型



面向对象分析模型



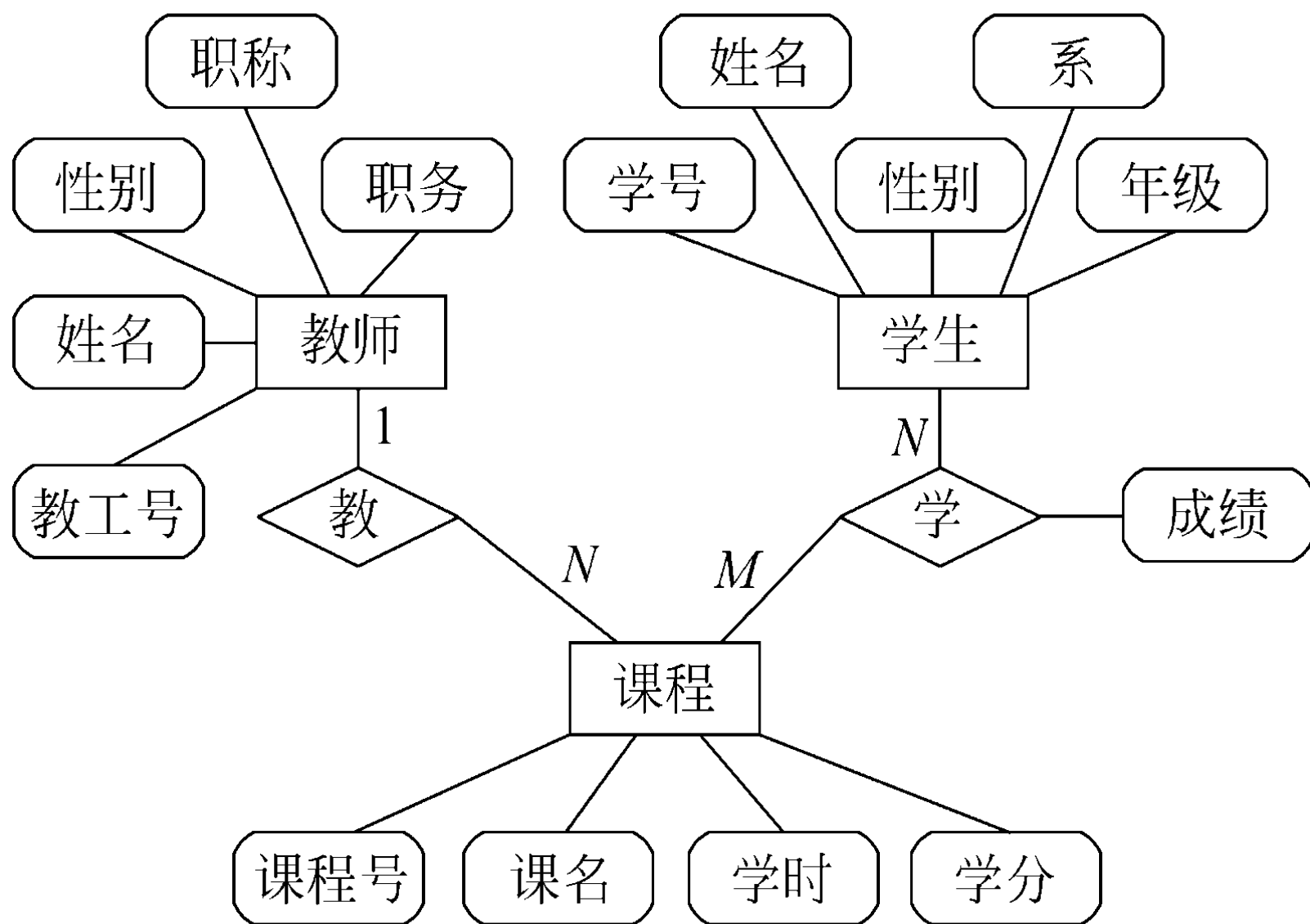
分析模型描述工具



- 结构化分析工具
 - DFD、DD 和 PSPEC
 - CFD、CSPEC 和 STD
 - E-R 图
- 面向对象分析工具
 - 用例图，类对象图
 - 对象 - 关系图
 - 对象 - 行为图

实体 - 联系图

- 实体 - 联系图是一种概念性的数据模型，包含 **3** 种相互关联的信息：
 - **数据对象**是可以由一组属性来定义的实体。
 - **属性**定义了数据对象的性质。
 - 数据对象彼此之间相互连接的方式称为**联系**，也称为**关系**。
 - (1) 一对一联系 (1:1)
 - (2) 一对多联系 (1:N)
 - (3) 多对多联系 (M:N)



某校教学管理 ER 图

- 联系也可能有属性。例如，学生“学”某门课程所取得的成绩，既不是学生的属性也不是课程的属性。由于“成绩”既依赖于某名特定的学生又依赖于某门特定的课程，所以它是学生与课程之间的联系“学”的属性。

实体 - 联系图的符号

- **ER** 图中包含了实体（即数据对象）、关系和属性等 **3** 种基本成分，用矩形框代表实体，用连接相关实体的菱形框表示关系，用椭圆形或圆角矩形表示实体（或关系）的属性，并用直线把实体（或关系）与其属性连接起来。

数据规范化

- 软件系统经常使用各种长期保存的信息，这些信息通常以一定方式组织并存储在数据库或文件中，为减少数据冗余，避免出现插入异常或删除异常，简化修改数据的过程，通常需把**数据结构规范化**。
- 通常用“**范式 (normal forms)**”定义消除数据冗余的程度。第一范式 (**1 NF**) 数据冗余程度最大，第五范式 (**5 NF**) 数据冗余程度最小。

- 范式并非越高越好

- 第一，范式级别越高，存储同样数据就需要分解成更多张表，因此，“存储自身”的过程也就越复杂。

- 第二，随着范式级别的提高，数据的存储结构与基于问题域的结构间的匹配程度也随之下降，因此，在需求变化时数据的稳定性较差。

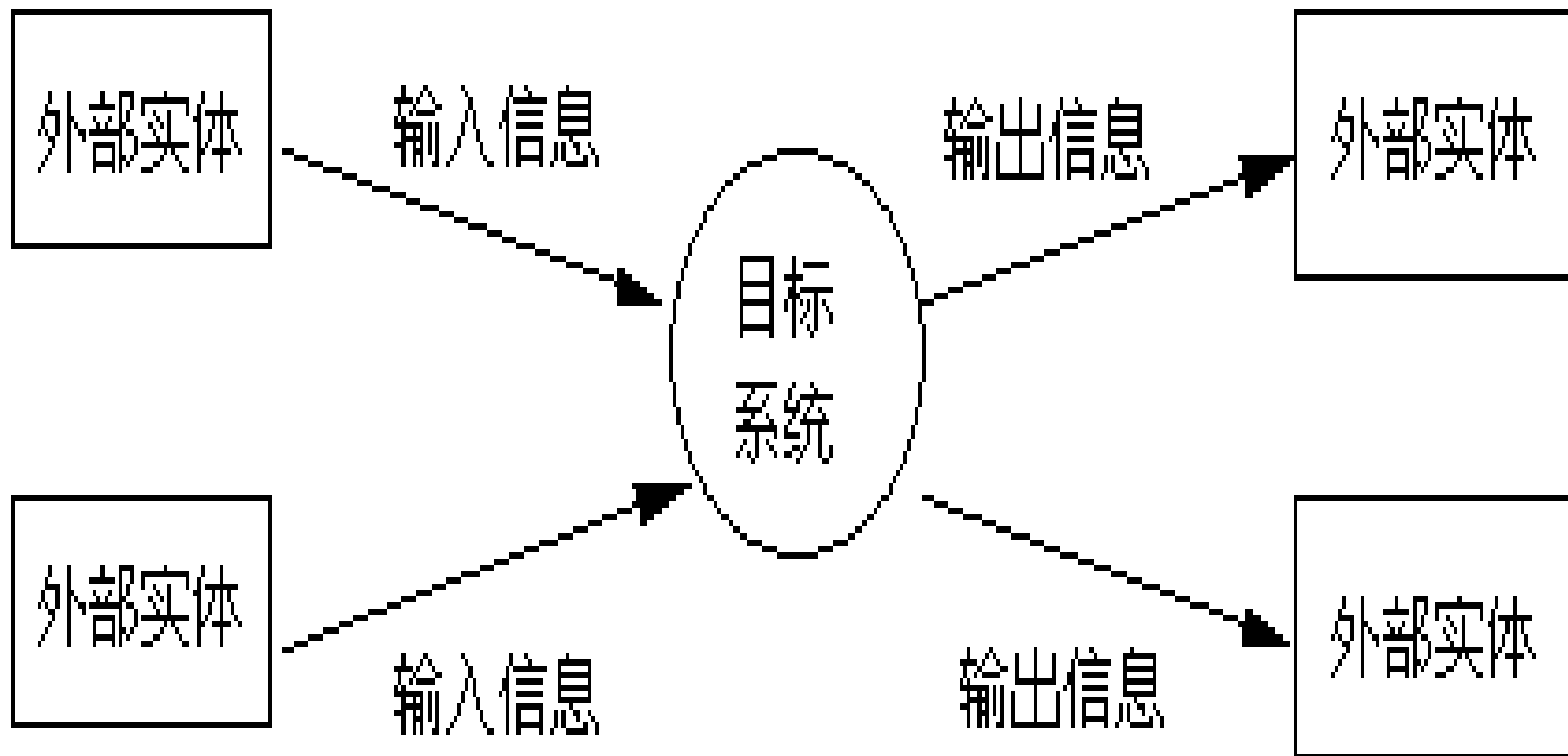
- 第三，范式级别提高则需要访问的表增多，因此性能（速度）将下降。

- 从实用角度来看，在大多数场合选用第三范式比较恰当。

数据流图

- **数据流图 (DFD)** 是一种图形化技术，它描绘信息流和数据从输入移动到输出的过程中所经受的变换。
- 在数据流图中没有任何具体的物理部件，它只是描绘数据在软件中流动和被处理的逻辑过程。
- 数据流图是**系统逻辑功能的图形表示**，即使不是专业的计算机技术人员也容易理解它，因此是分析员与用户之间极好的通信工具。
- 此外，设计数据流图时只需考虑系统必须完成的基本逻辑功能，完全不需要考虑怎样具体地实现这些功能。

数据流图 (DFD)



符号

- 数据流图有四种基本符号：
 - 正方形（或立方体）表示数据的源点或终点；
 - 圆角矩形（或圆形）代表变换数据的处理；
 - 开口矩形（或两条平行横线）代表数据存储；
 - 箭头表示数据流，即特定数据的流动方向。

例子

假设一家工厂的采购部每天需要一张定货报表，报表按零件编号排序，表中列出所有需要再次定货的零件。对于每个需要再次定货的零件应该列出下述数据：零件编号，零件名称，定货数量，目前价格，主要供应者，次要供应者。零件入库或出库称为事务，通过放在仓库中的终端把事务报告给定货系统。当某种零件的库存数量少于库存量临界值时就应该再次定货。

- 第一步可以从问题描述中提取数据流图的4种成分：

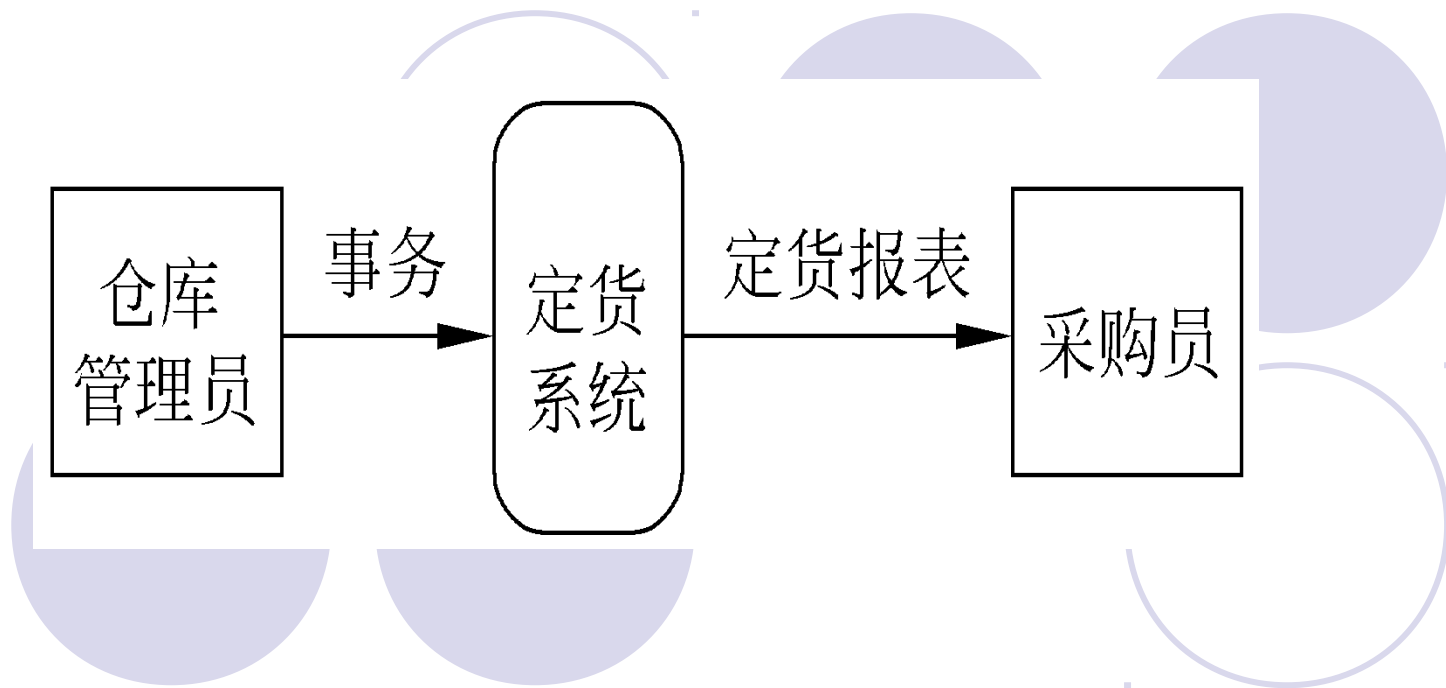
- 首先考虑数据的源点和终点，从上面对系统的描述可以知道“采购部每天需要一张定货报表”，“通过放在仓库中的终端把事务报告给定货系统”，所以采购员是数据终点，而仓库管理员是数据源点。

- 接下来考虑处理，再一次阅读问题描述，“采购部需要报表”，显然他们还没有这种报表，因此必须有一个用于产生报表的处理。事务的后果是改变零件库存量，然而任何改变数据的操作都是处理，因此对事务进行的加工是另一个处理。

○最后，考虑数据流和数据存储：系统把定货报表送给采购部，因此定货报表是一个数据流；事务需要从仓库送到系统中，显然事务是另一个数据流。产生报表和处理事务这两个处理在时间上明显不匹配——每当有一个事务发生时立即处理它，然而每天只产生一次定货报表。因此，用来产生定货报表的数据必须存放一段时间，也就是应该有一个数据存储。

- 数据流图是系统的逻辑模型，然而任何计算机系统实质上都是信息处理系统，也就是说计算机系统本质上都是把输入数据变换成输出数据。

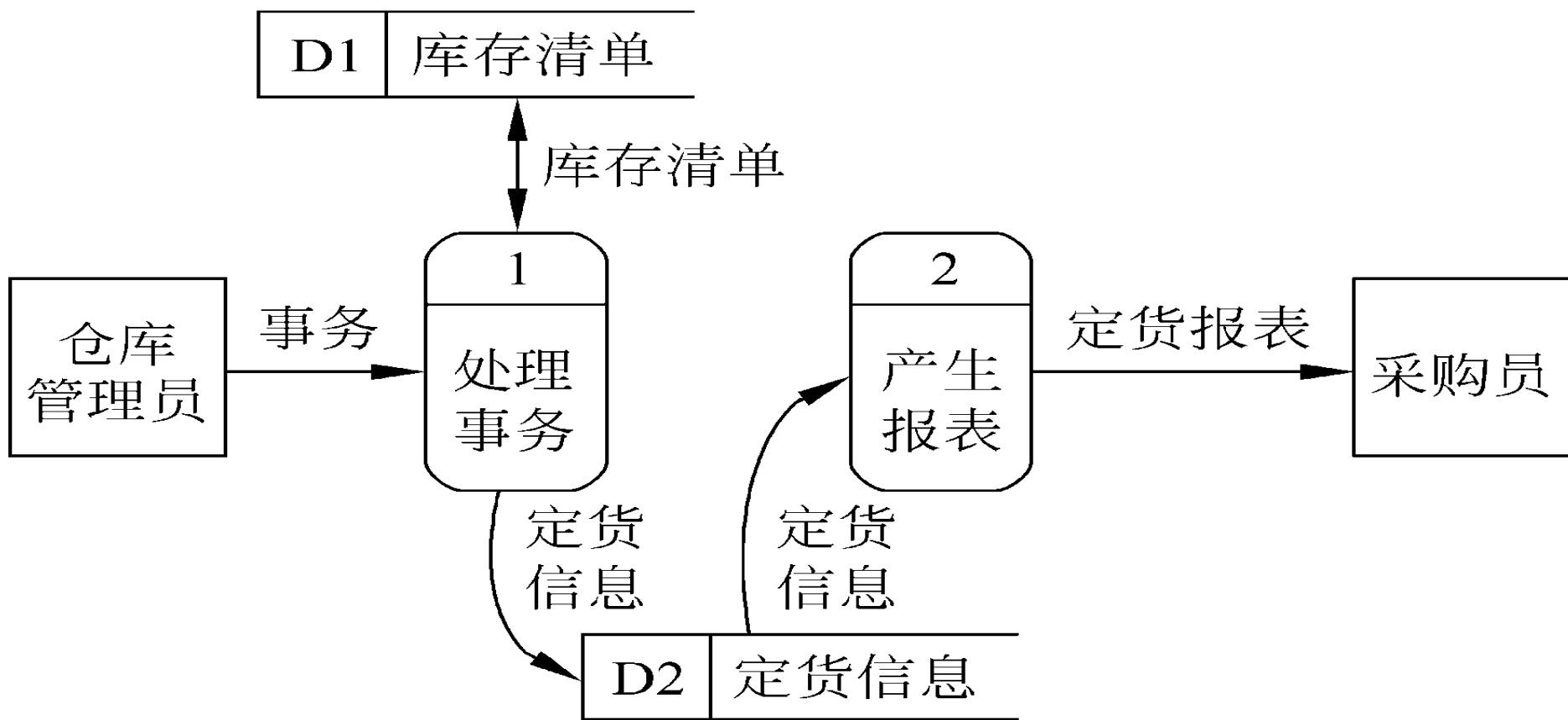
- 因此，任何系统的基本模型都由若干个数据源点 / 终点以及一个处理组成，这个处理就代表了系统对数据加工变换的基本功能。对于上述的定货系统可以画出基本系统模型。



定货系统的基本系统模型

从基本系统模型这样非常高的层次开始画数据流图是一个好办法。在这个高层次的数据流图上是否列出了所有给定的数据源点 / 终点是一目了然的，因此它是很有价值的通信工具。

下一步应该把基本系统模型细化，描绘系统的主要功能。“产生报表”和“处理事务”是系统必须完成的两个主要功能，它们将代替图的“定货系统”。

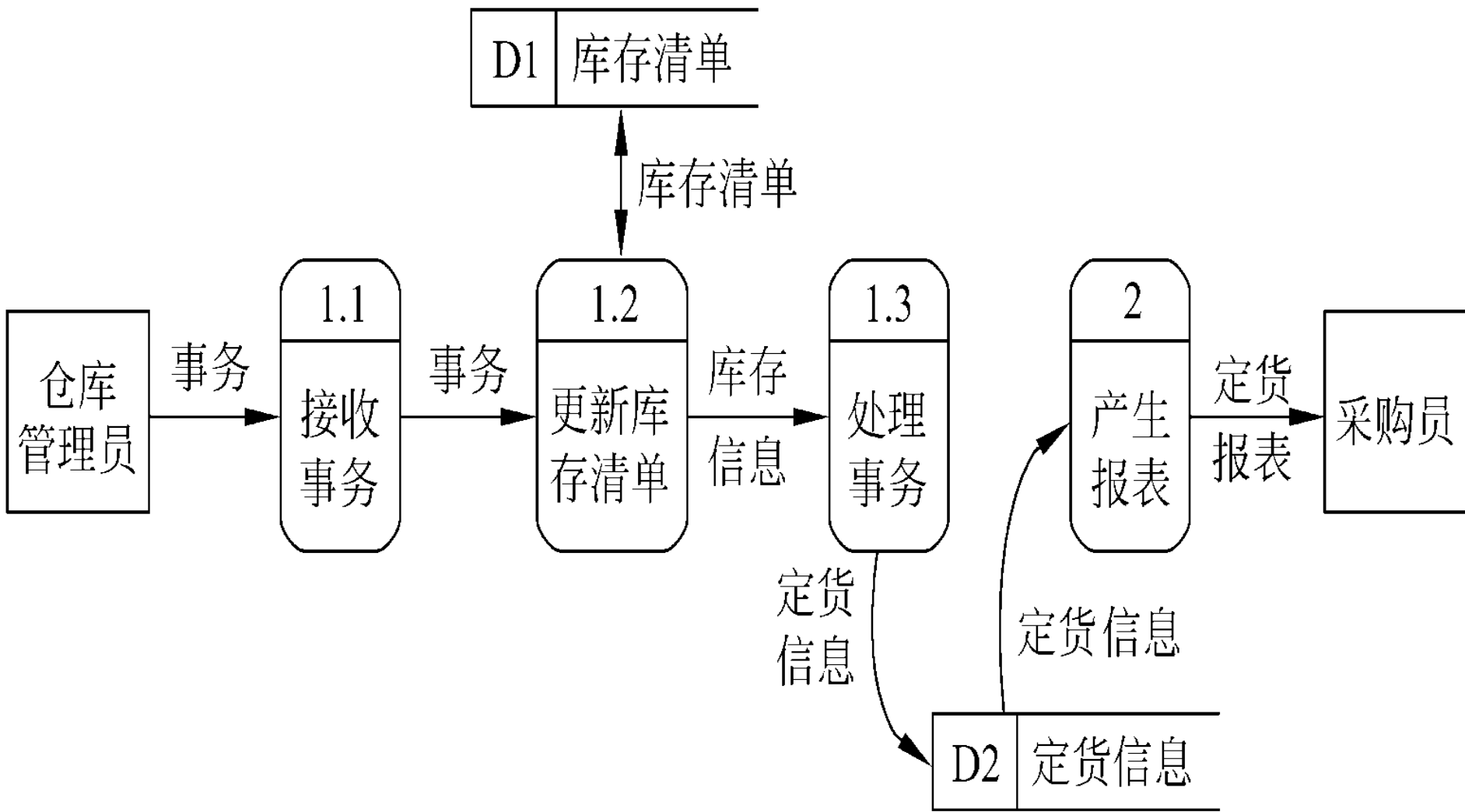


定货系统的功能级数据流图

- 接下来应该对功能级数据流图中描绘的系统主要功能进一步细化。

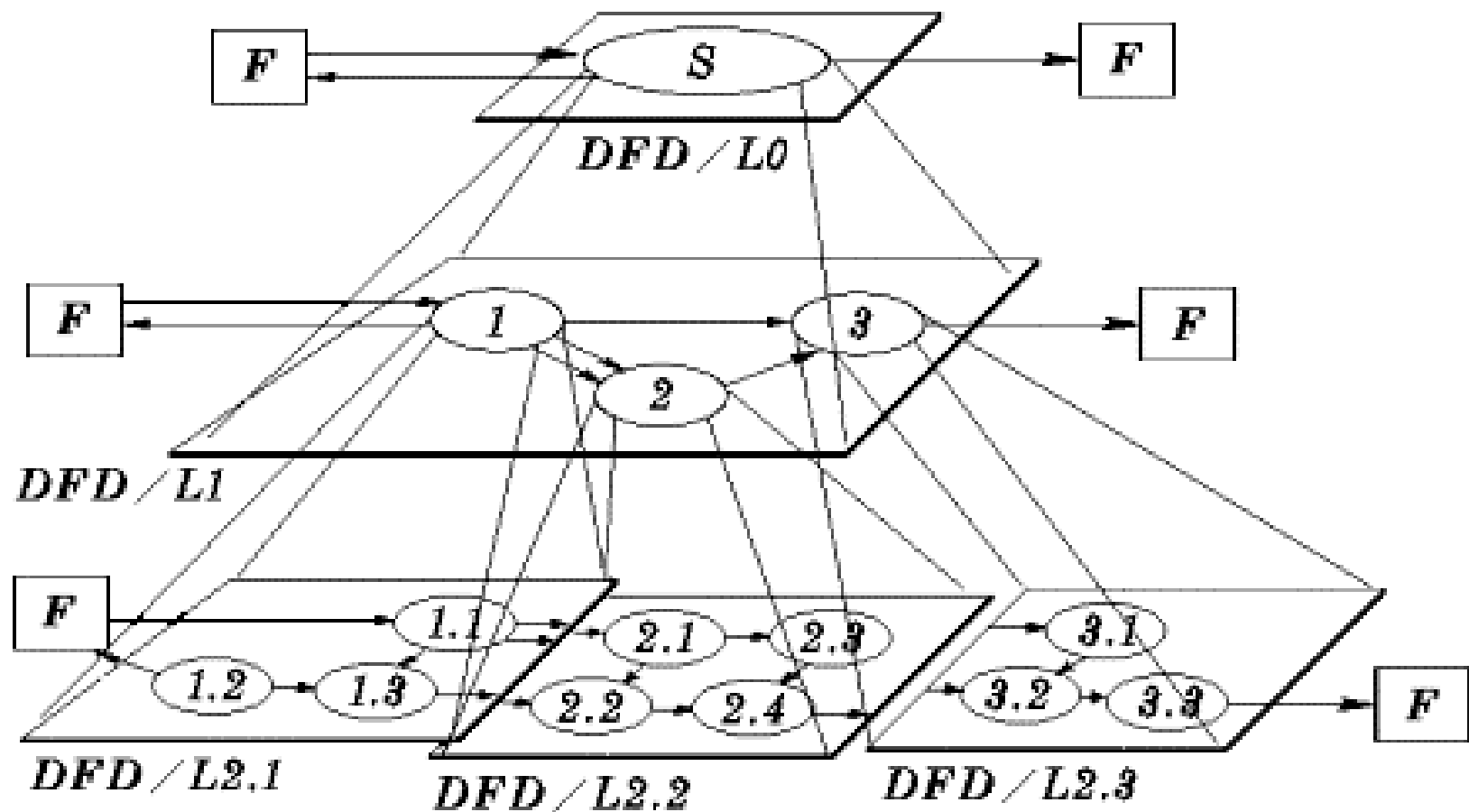
- 考虑通过系统的逻辑数据流：当发生一个事务时必须首先接收它；随后按照事务的内容修改库存清单；最后如果更新后的库存量少于库存量临界值时，则应该再次定货，也就是需要处理定货信息。

- 因此，把“处理事务”这个功能分解为下述 **3** 个步骤，这在逻辑上是合理的：“接收事务”、“更新库存清单”和“处理定货”。



把处理事务的功能进一步分解后的数据流图

分层数据流图



结构化系统分析方法

- 定义：结构化方法的基本思想和主要原则在系统分析中的应用所形成的一系列具体方法和有关工具的总称。

结构化方法的主要思路

- “结构化”的含义：用一组规范的步骤、准则和工具来进行某项工作。
- 基本思路：把整个系统开发过程分成若干阶段，每个阶段进行若干活动，每项活动应用一系列标准、规范、方法和技术，完成一个或者多个任务，形成符合给定规范的产品。

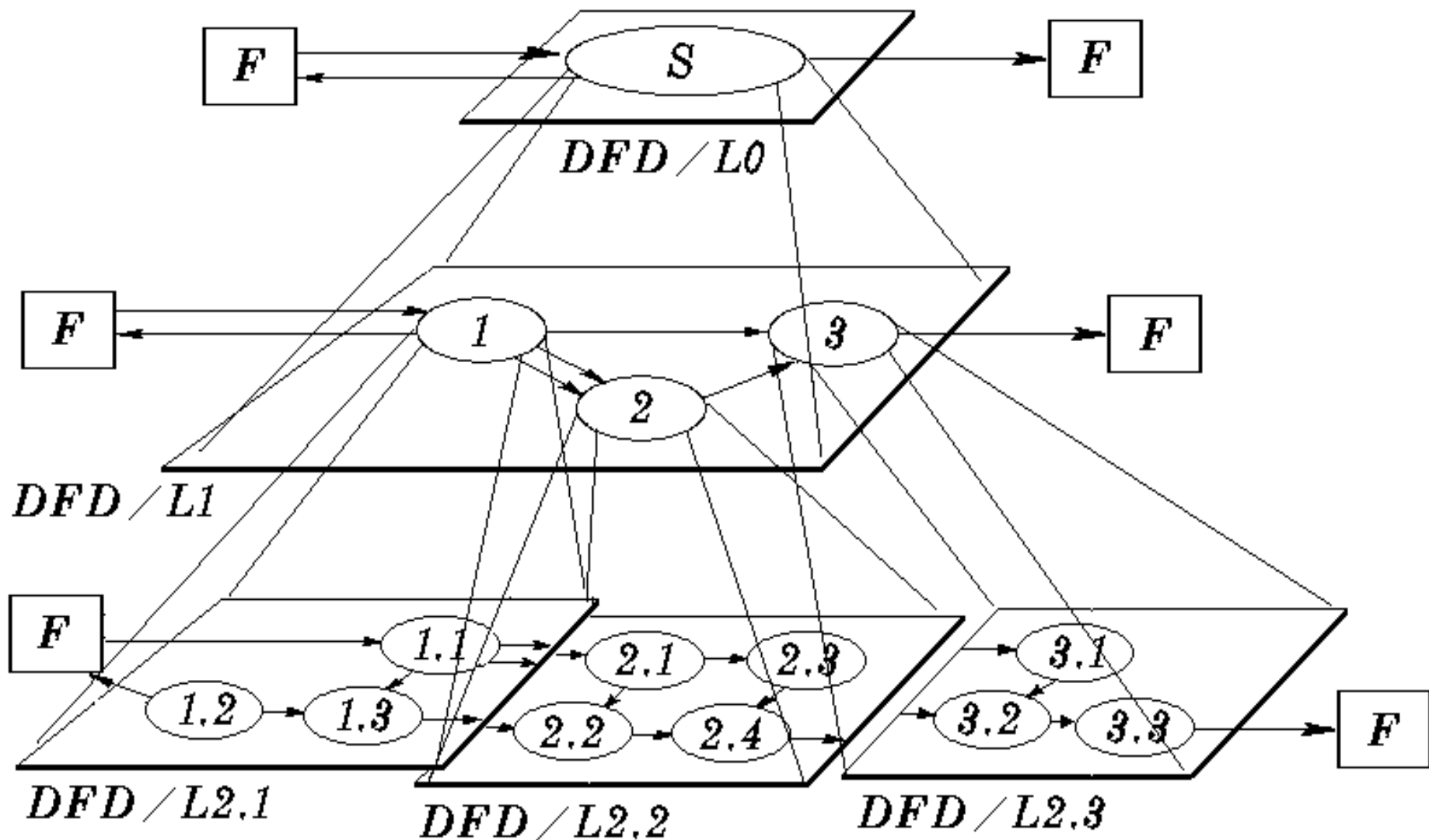
结构化方法的主要原则

- 用户参与。
- “先逻辑、后物理”：重视调查、分析、论证，逻辑方案设计。
- “自顶向下”：“自顶向下”是主导原则，“自底向上”是辅助原则。
- 工作成果描述标准化：文档化、图形化。

数据流图

- **DFD : Data Flow Diagram**
- 概念：一种能全面描述信息系统逻辑模型的主要工具，能综合反映出信息在系统中流动、处理和存储的情况。
- 使用图形以及相关的注释来表示系统的逻辑功能。
- **DFD+DD** 构成系统的完整的逻辑模型。

分层数据流图



数据流图的绘制步骤——详解

概括地说：自外向内，自顶向下，逐层细化，完善求精

- (1) 首先确定系统的输入和输出，以反映系统与外界环境的接口。
- (2) 第 0 层数据流图将软件系统描述为一个加工，以反映最主要业务处理流程，它代表系统本身。但它并未明确表达数据加工的要求。
- (3) 从输入端开始，根据系统业务工作流程，画出数据流流经的各加工框，以反映数据的实际处理过程，逐步画到输出端，得到第一层数据流图。图中的加工分别加以编号。
- (4) 细化每一个加工框。如果加工框内还有数据流，可将这个加工框再细分成几个“子加工框”，并在各子加工框之间画出数据流。
- (5) 一次细化一个加工。
- 数据流图的细化可以连续进行，直到每一个加工只执行一个简单操作为止。就是说，直到每一个加工执行一个可以用程序实现的功能为止。

数据流图的绘制步骤——概说

- 确定系统的外部项
- 画出顶层图
- 自顶向下逐层分解直到基本加工
- 检查
- 征求用户意见
- 定稿
- 复审

绘制原则

- 明确系统界面（确定外部项）
- 自顶向下逐层扩展（顶层图、第一层图。。。。。。）
- 合理布局
- 只从系统逻辑功能上讨论问题
- 用户参与

DFD 练习一分房管理

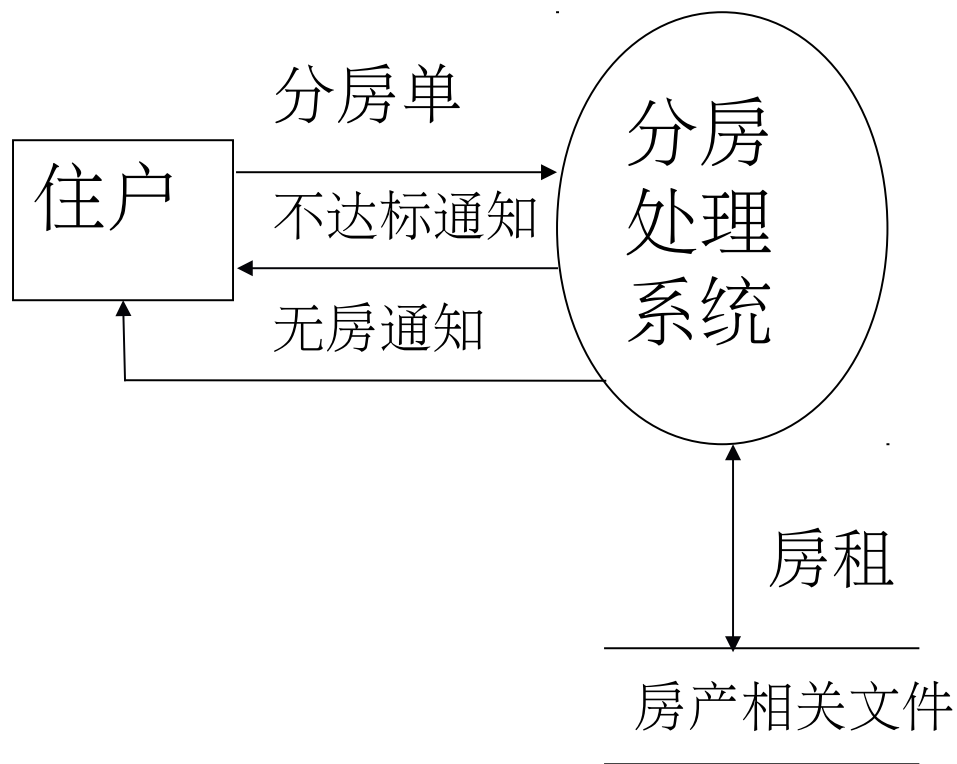
- 问题描述：住户把分房单交给房产管理员，管理员要先根据住房标准文件核准住户的住房条件，如果够标准再根据房产文件察看有无空房可以分配，如果有则分配住房并且计算房租，记录入房租文件。如果不够标准或者无房可分，则不予分房，并对住户下发通知。

DFD 练习一分房管理

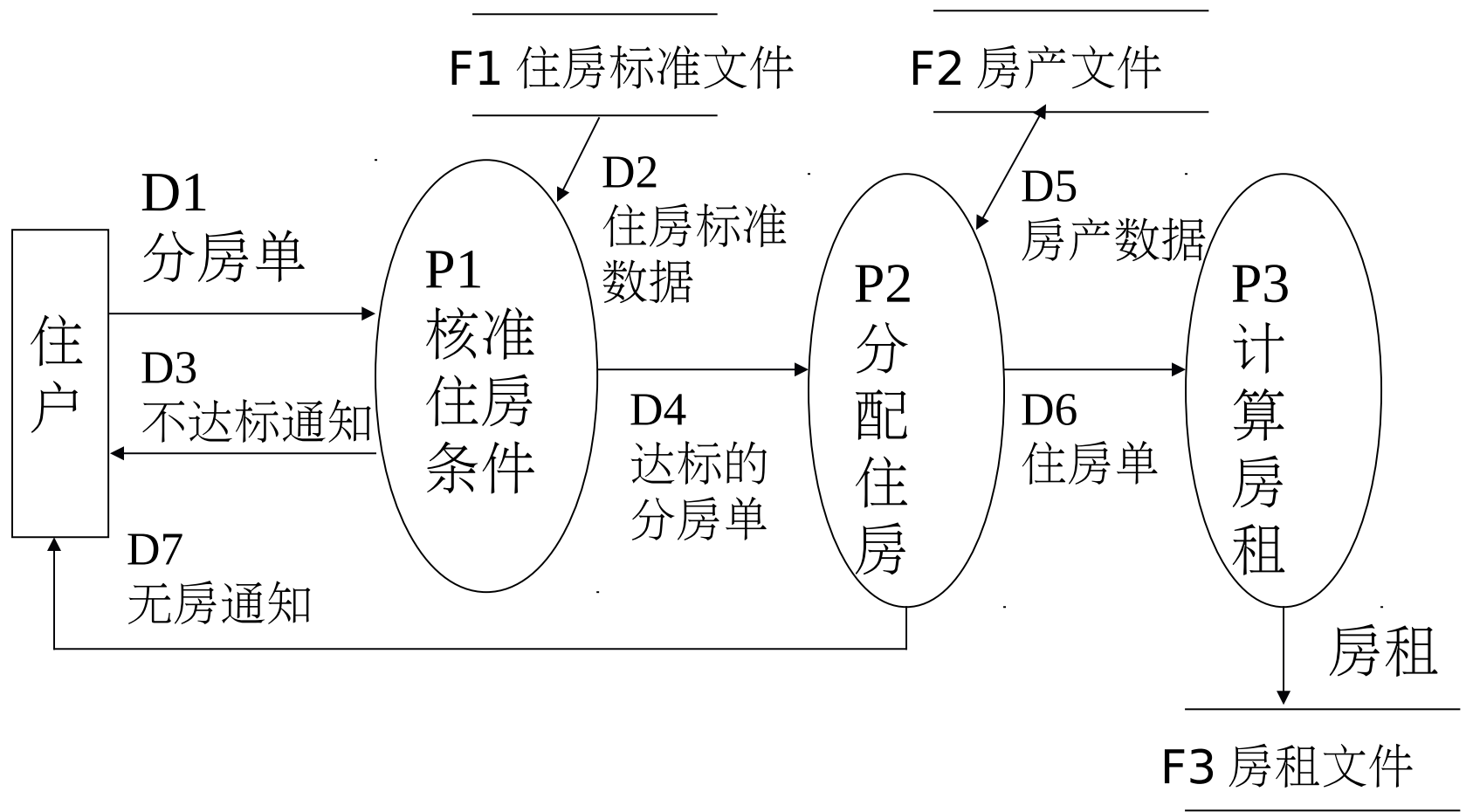
- 语法分析

- 名词：潜在的外部项、数据流、数据存储。
- 动词：潜在的加工、数据流。

DFD 练习一分房管理 0



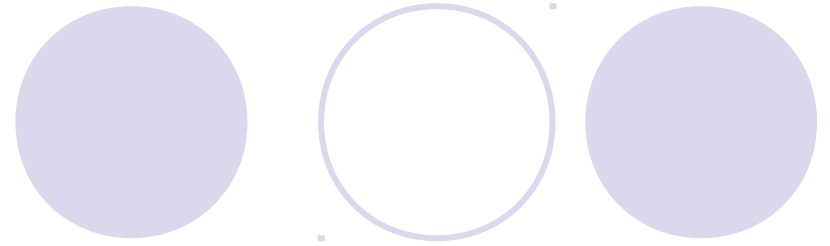
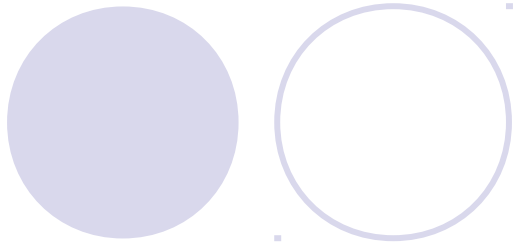
DFD 练习一分房管理 1



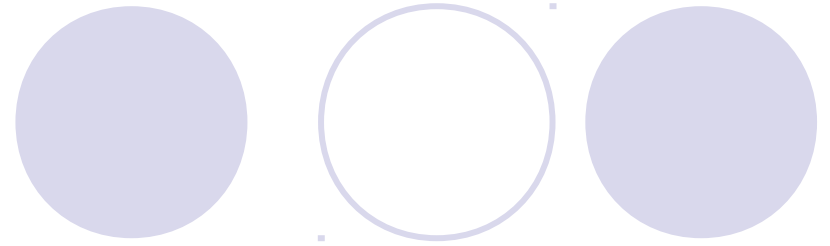
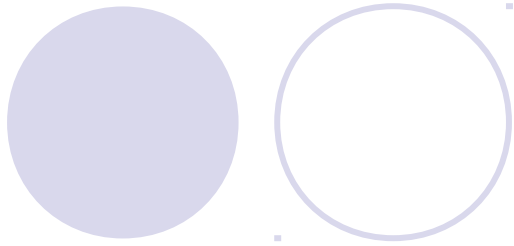
注意事项



- 始终把握住系统的总体目标与功能
- 减少图层，每层的加工项目不超过 8 个
- 保持各层成分（主要是加工）的完整性与一致性
- 加工的分解可能导致数据流、数据存储、外部项的分解



- 可以出现不属于上一层数据存储环节子项的新的数据存储环节
- 数据流必须经过加工
- 数据存储环节一般作为两个加工环节的界面来安排



- 命名（统一使用动词 / 名词）
- 编号（图号、外部项号 S 、加工号 P 、数据流号 **D** 、数据存储号 **F** ）
 - **P2.3**
 - 第一层图的第 2 个加工；第二层图的第 3 个加工
 - **D1.1 / D1.2**
 - 第一层图的第 1 个数据流，第二层图的第 **1/2** 个数据流

数据流图的局限性



- 不详细不具体（需要数据字典）
- 反映的是系统处于静态的逻辑模型，不能反映系统动态模型
- 不能反映系统中的决策与控制过程

基本加工



- 概念：数据流图中所有不进一步分解的加工。
- 识别方法：有父项、无子项。

基本加工说明

- 加工说明 **PSPEC**
 - 说明 **DFD** 中的每个基本加工
- 描述工具
 - 结构化语言（过程设计语言）
 - 决策树
 - 决策表
 - 盒图
 - 数学公式

结构化语言

- 结构化语言 / 伪代码 / 过程设计语言 **PDL**
- 结构化语言介于自然语言与计算机语言之间。
- 语句类型：顺序语句、条件语句、循环语句。

结构化语言举例——商店业务处理系统中 " 检查发货单 " 的例子

- ```
IF 发货单金额超过 $500 THEN
 IF 欠款超过 60 天 THEN
 在偿还欠款前不予批准
 ELSE (欠款未超期)
 发批准书及发货单
 ENDIF
ELSE (发货单金额未超过 $500)
 IF 欠款超过 60 天 THEN
 发批准书, 发货单及催款通知
 ELSE (欠款未超期)
 发批准书及发货单
 ENDIF
ENDIF
ENDIF
```

# 决策树

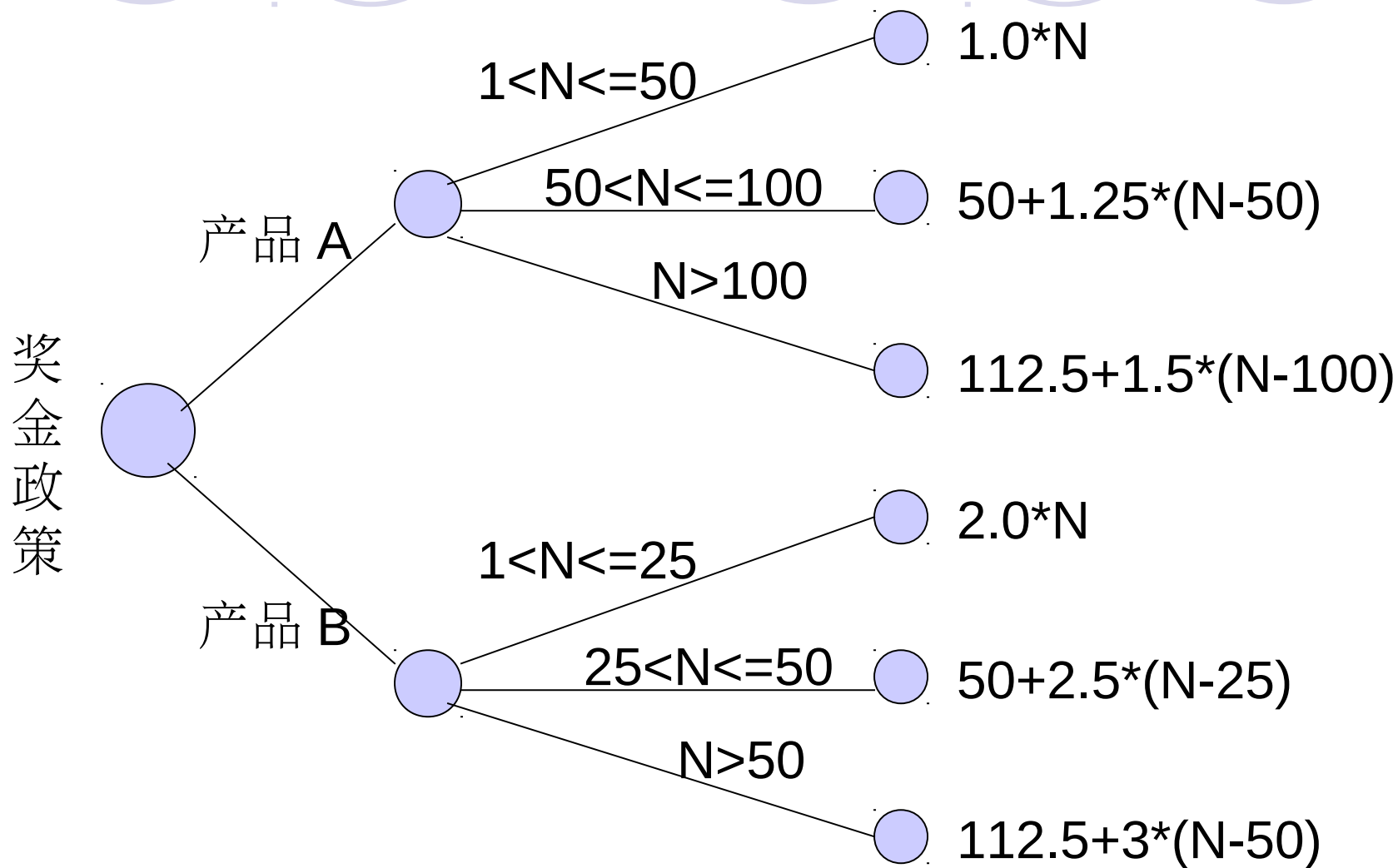


- 决策树是一种图形工具，适合于描述加工中具有多个策略，而且每个策略和若干条件有关的逻辑功能。

# 决策树举例 1：“检查发货单”的判定树



## 决策树举例 2：超产奖决策树



# 决策表



- 如果判断的条件较多，各条件又相互组合，则使用决策表比较好。

# 决策表举例 1 ： " 检查发货单 " 的判定表

|        |        | 1      | 2      | 3      | 4      |
|--------|--------|--------|--------|--------|--------|
| 条<br>件 | 发货单金额  | >\$500 | >\$500 | ≤\$500 | ≤\$500 |
|        | 赊欠情况   | >60天   | ≤60天   | >60天   | ≤60天   |
| 操<br>作 | 不发出批准书 | ✓      |        |        |        |
|        | 发出批准书  |        | ✓      | ✓      | ✓      |
|        | 发出发货单  |        | ✓      | ✓      | ✓      |
|        | 发出赊欠报告 |        |        | ✓      |        |



## 决策表举例 2：超产奖决策表

|                |                           | 1 | 2 | 3 |      |
|----------------|---------------------------|---|---|---|------|
| 超产量<br>(条件)    | $1 < n \leq 50$           | Y | N | N | 状态   |
|                | $50 < n \leq 100$         | N | Y | N |      |
|                | $n > 100$                 | N | N | Y |      |
| 奖金方案<br>(决策方案) | $1.0 * n$                 | X |   |   | 决策规则 |
|                | $50 + 1.25 * (n - 50)$    |   | X |   |      |
|                | $112.5 + 1.5 * (n - 100)$ |   |   | X |      |

- 在对加工的描述中，如果条件比较多，而且又互相组合，则比较适合使用决策表。
- 决策表的田字型结构：条件、状态、决策方案、决策规则。
- 决策表的读表方法：顺时针方向。

# 决策表的绘制步骤

- 分析决策问题涉及几个条件。
- 分析每个条件有几个取值区间。
- 画出条件取值分析表，分析条件的各种可能组合。
- 分析决策问题涉及几个决策方案。
- 画出有条件组合的决策表。
- 决定各种条件组合的决策方案，即填写决策规则。
- 合并化简决策表，即相同决策方案所对应的各个条件组合是否存在无需判断的条件。

# 决策表绘制方法举例

(1) 问题说明：“某货运站收费标准如下：如果收件地点在本省，则快件每公斤 **5** 元，慢件每公斤 **3** 元；如果收件地点在外省，则在 **20** 公斤以内（含 **20** 公斤）快件每公斤 **7** 元，慢件每公斤 **5** 元，而超过 **20** 公斤时，快件每公斤 **9** 元，慢件每公斤 **7** 元。”

# 决策表绘制方法举例

## (2) 条件取值分析表

| 条件           | 取值 | 含义 |
|--------------|----|----|
| 收件地址在本省?     | Y  | 是  |
|              | N  | 否  |
| 邮件重量 <20 公斤? | Y  | 是  |
|              | N  | 否  |
| 快慢件?         | Y  | 快件 |
|              | N  | 慢件 |

# 决策表绘制方法举例

## (3) 决策表

|      |              | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |      |
|------|--------------|---|---|---|---|---|---|---|---|------|
| 条件   | 收件地址在本省?     | Y | Y | Y | Y | N | N | N | N | 状态   |
|      | 邮件重量 <20 公斤? | Y | Y | N | N | Y | Y | N | N |      |
|      | 快慢件?         | Y | N | Y | N | Y | N | Y | N |      |
| 决策方案 | 3 元 / 公斤     |   | X |   | X |   |   |   |   | 决策规则 |
|      | 5 元 / 公斤     | X |   | X |   |   | X |   |   |      |
|      | 7 元 / 公斤     |   |   |   |   | X |   |   | X |      |
|      | 9 元 / 公斤     |   |   |   |   |   |   | X |   |      |

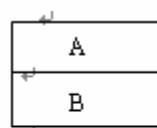
# 决策表绘制方法举例

## (4) 化简后的决策表

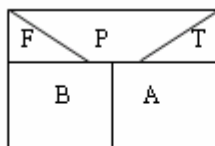
|      |              | 1 | 2 | 3 | 4 | 5 | 6 |      |
|------|--------------|---|---|---|---|---|---|------|
| 条件   | 收件地址在本省?     | Y | Y | N | N | N | N | 状态   |
|      | 邮件重量 <20 公斤? | — | — | Y | Y | N | N |      |
|      | 快慢件?         | Y | N | Y | N | Y | N |      |
| 决策方案 | 3 元 / 公斤     |   | X |   |   |   |   | 决策规则 |
|      | 5 元 / 公斤     | X |   |   | X |   |   |      |
|      | 7 元 / 公斤     |   |   | X |   |   | X |      |
|      | 9 元 / 公斤     |   |   |   |   | X |   |      |

# 盒图（N-S图）

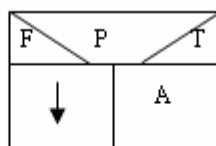
- 三种基本的结构化控制构件都可以用盒图表示。盒图的优点
  - 结构化功能域清晰
  - 控制不能随意转移
  - 数据的作用域清晰



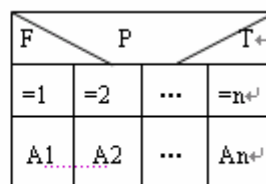
(a)顺序型



(b)选择型



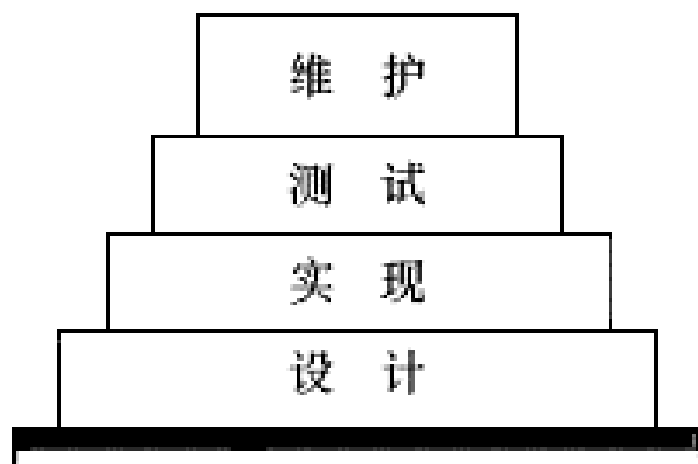
(c)先判定型循环



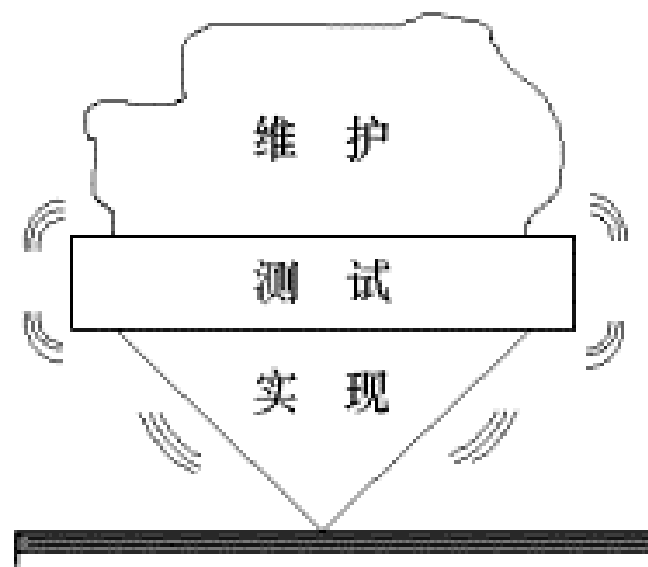
(d)后判定型循环

(e)多分支选择型

# 软件设计的重要性



有软件设计



没有软件设计



# 软件设计的任务

- 数据设计

- 信息模型 →

软件数据结构

- 体系结构设计

- 定义软件部件间的关系

- 过程设计

- 软件组件的过程性描述

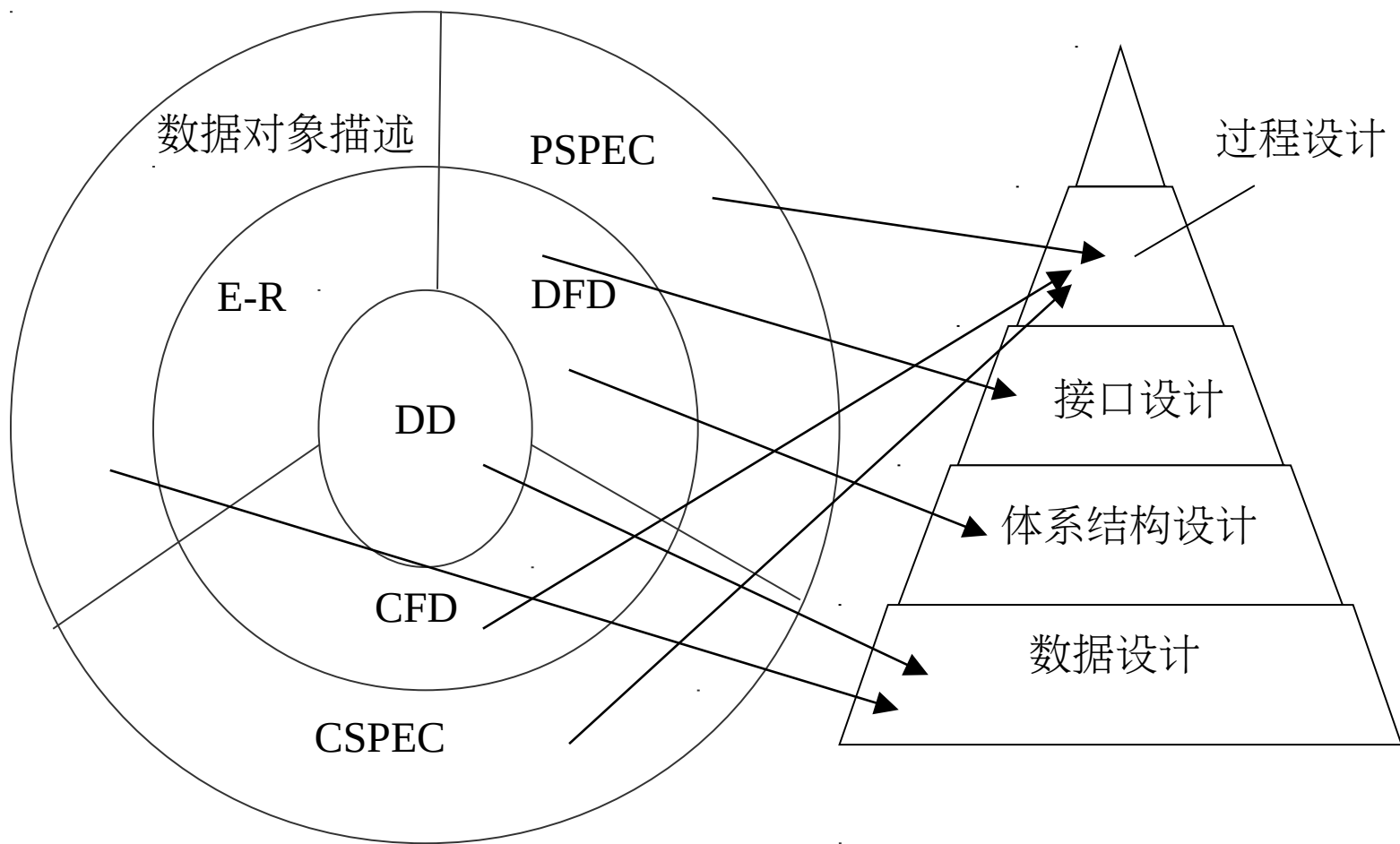
- 接口设计

- 软件内部、外部及与人之间的通信

# 软件设计的基本概念

- 模块（ **module** ） 与构件（ **component** ）
  - 模块：定义输入、输出和特性的程序实体
  - 构件：可重复使用的软件组件
- 抽象（ **abstract** ） 与细化（ **refinement** ）
  - 抽象：分层次考虑和处理问题（数据和过程）
  - 细化：从高到低的逐步分解过程
- 信息隐藏
  - 对其它模块隐藏模块内部的数据和过程
- 软件复用
  - **Design with reuse, design for reuse**

# 结构化设计的内容



# 结构化设计的内容

- 结构设计—概要设计

- 体系结构设计

- SC 图（系统结构图）

- 接口设计

- SC 图（系统结构图）

- 数据库设计

- 物理数据模型

- 过程设计—详细设计

- 模块的处理过程

- N-S（盒图）， PAD（问题分析图）， PDL 等

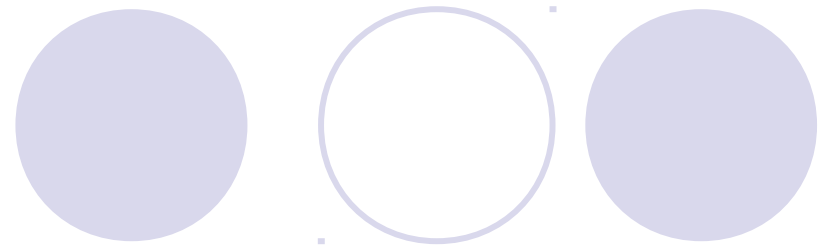
# 体系结构设计

- 体系结构设计的主要目标是开发一个模块化的程序结构，并表示出模块间的控制关系。
- 体系结构设计将程序结构和数据结构相结合，为数据在程序中流动定义了接口。

# 体系结构设计

- 面向数据流的设计是一种体系结构设计方法，它通过如下步骤将分析模型转换到程序结构的设计描述：
  - 1) 确定数据流的类型（变换型、事务型）；
  - 2) 指明流的边界；
  - 3) 将 **DFD** 映射到程序结构；
  - 4) 用”因子化”的方法定义控制的层次结构；
  - 5) 通过设计复审和启发策略对结构进行求精。

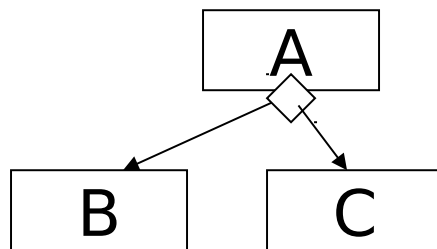
# 描述工具— SC 图



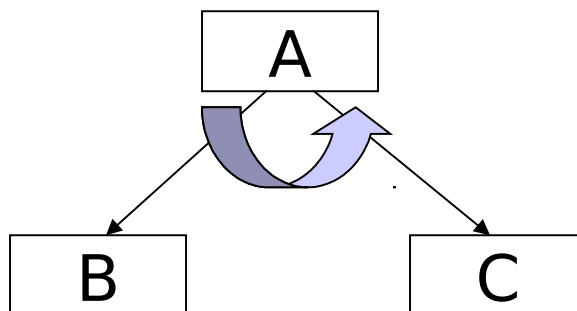
- SC 图的组成符号
  - 矩形框来表示模块
  - 带箭头的连线表示模块间的调用关系
  - 输入和输出模块的数据 / 控制流
- SC 图中的模块符号
  - 输入
  - 输出
  - 变换
  - 控制信息
  - 数据信息

# SC 图中的模块调用

- 简单调用
- 选择调用



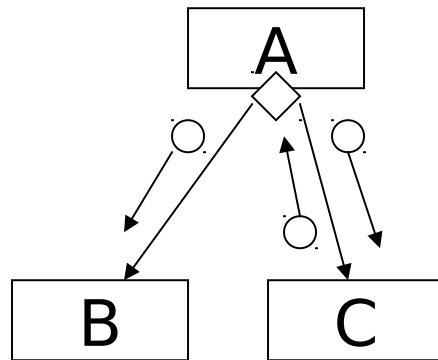
- 循环调用



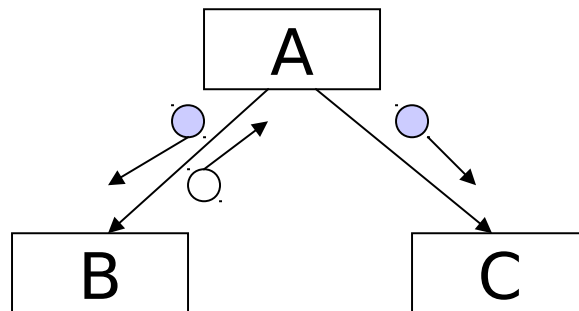


# SC 图中的数据 / 控制信息

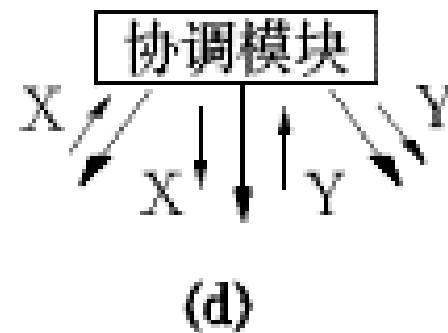
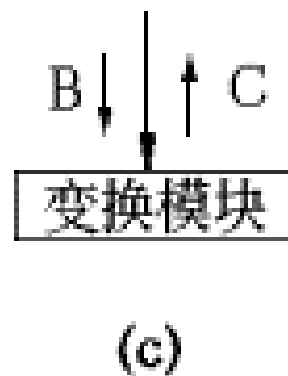
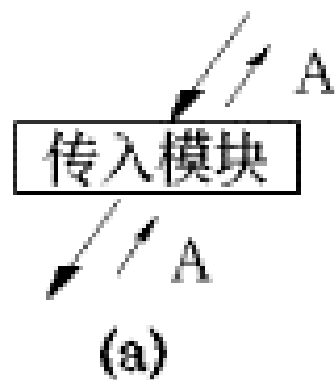
- 数据信息



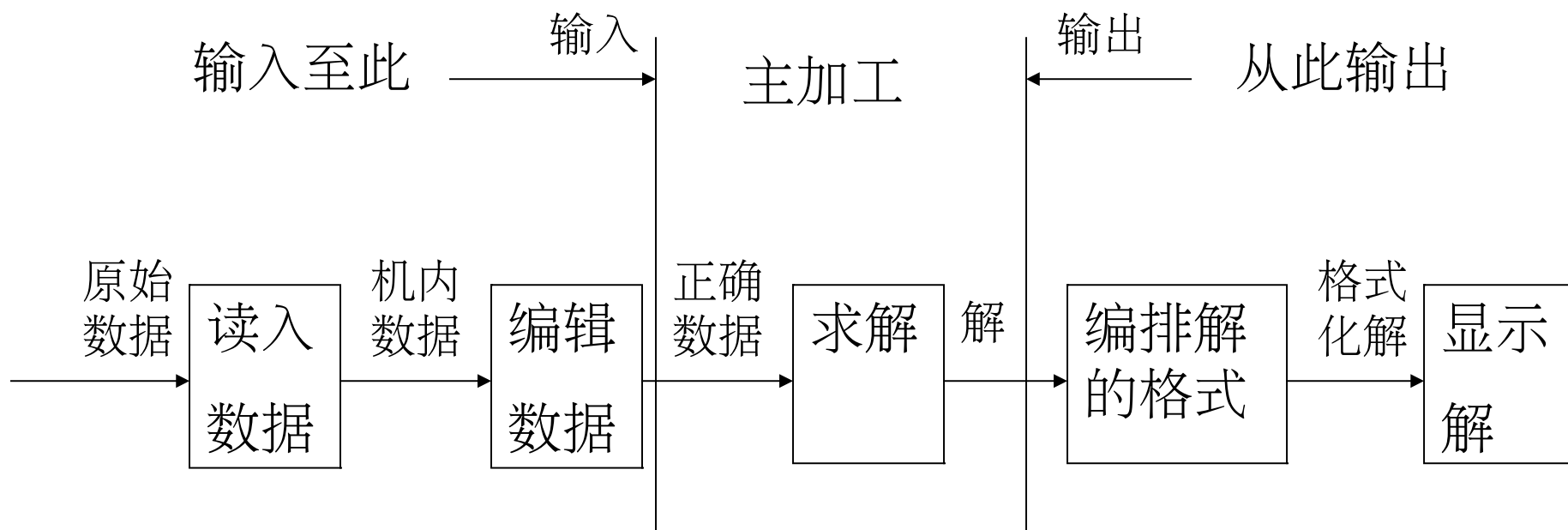
- 控制信息



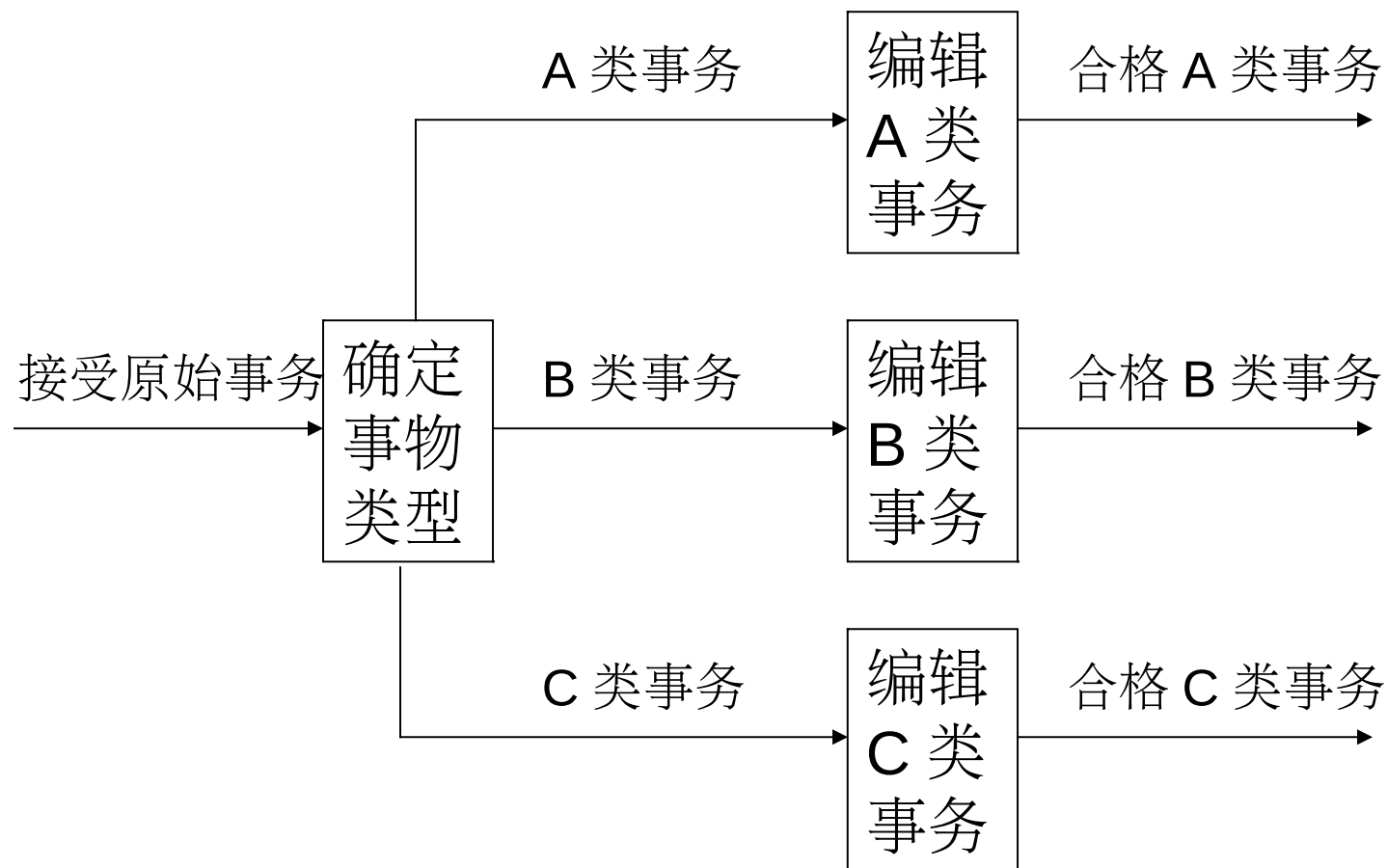
# SC 图的4种模块类型



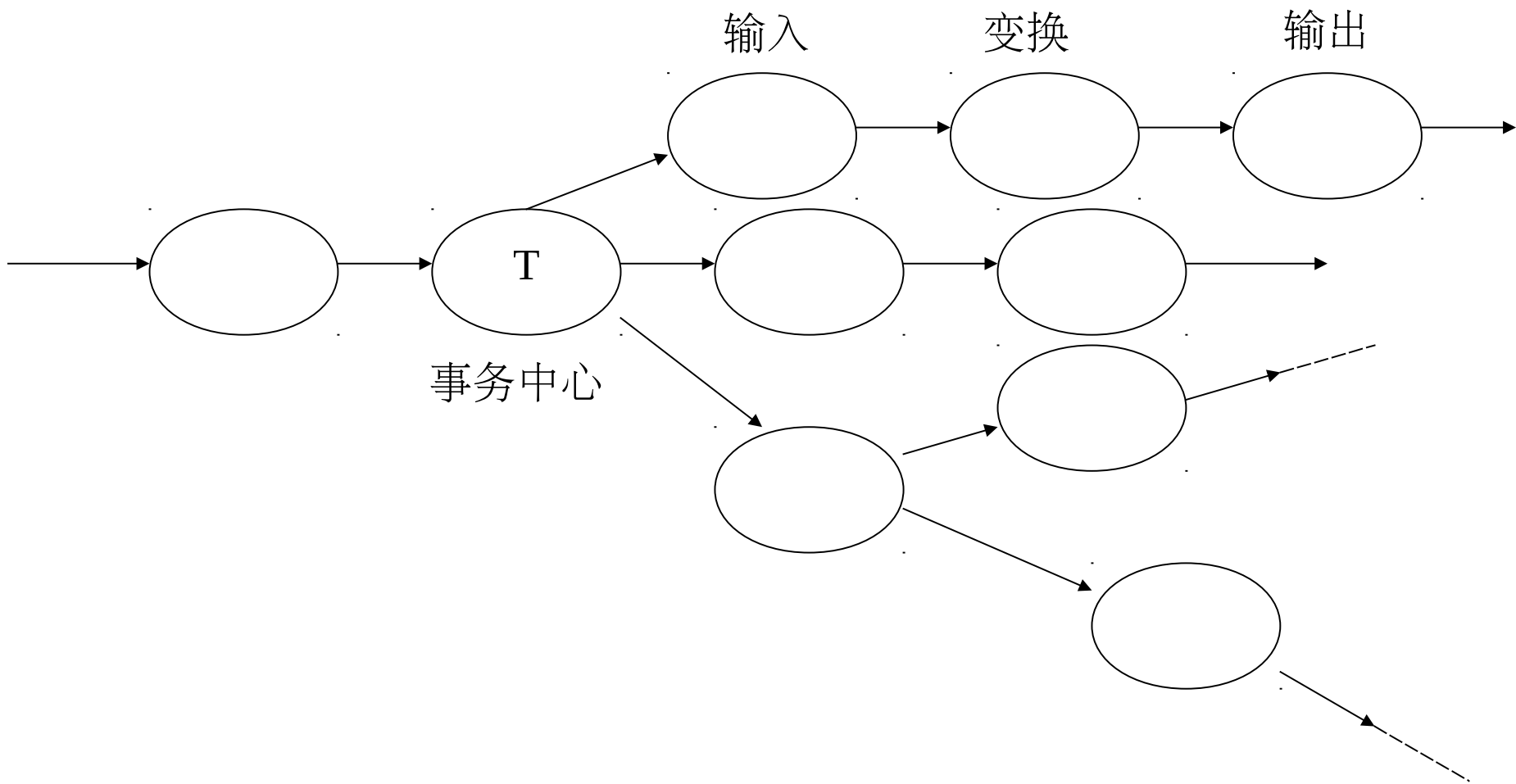
# 变换结构的 DFD 举例



# 事务型结构 DFD 举例



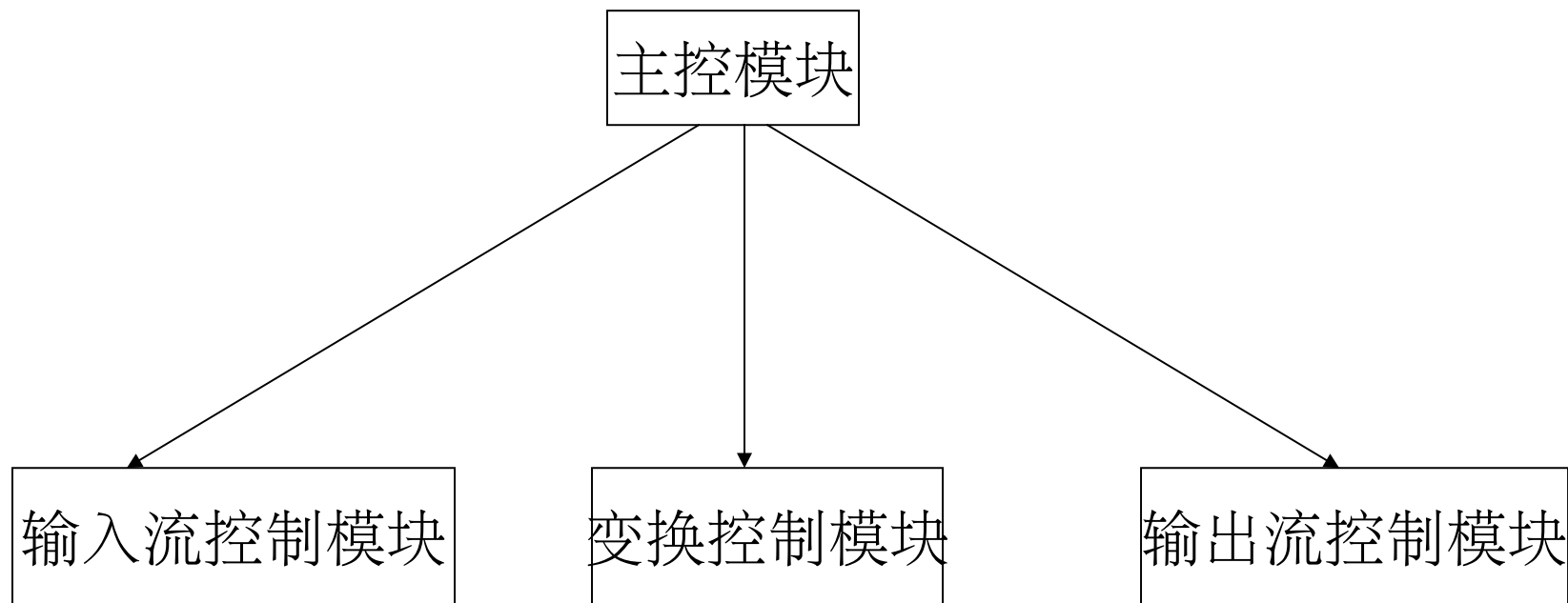
# 同时存在两类结构



# 变换分析

- 划分 **DFD** 图的边界
- 建立初始 **SC** 图的框架
  - 顶层都只含一个用于控制的主模块
  - 第一层包括输入、输出和中心变换三个模块
- 分解 **SC** 图的各个分支
  - 分解实质上是“映射”

# 变换分析初始 SC 图框架

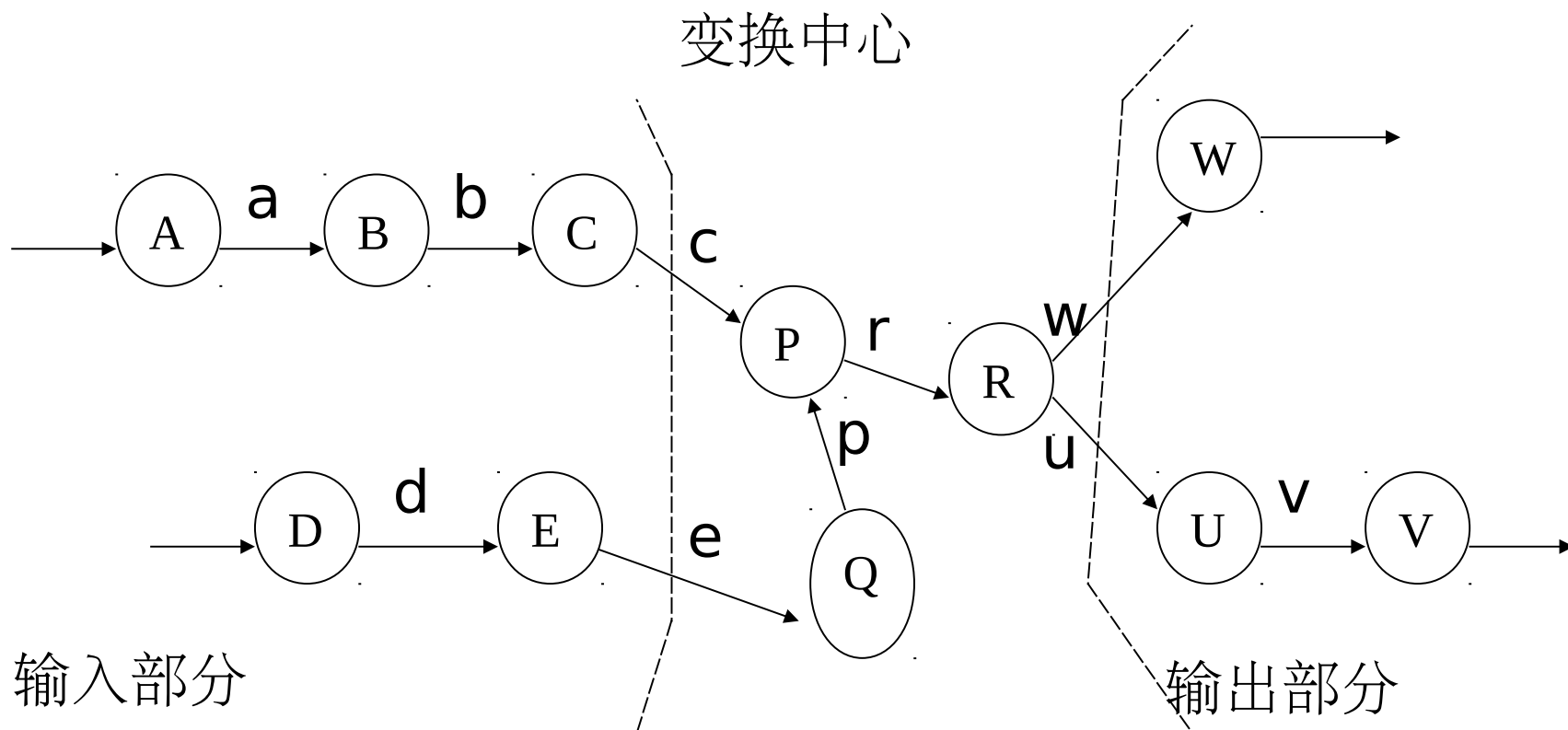


# 变换分析具体步骤

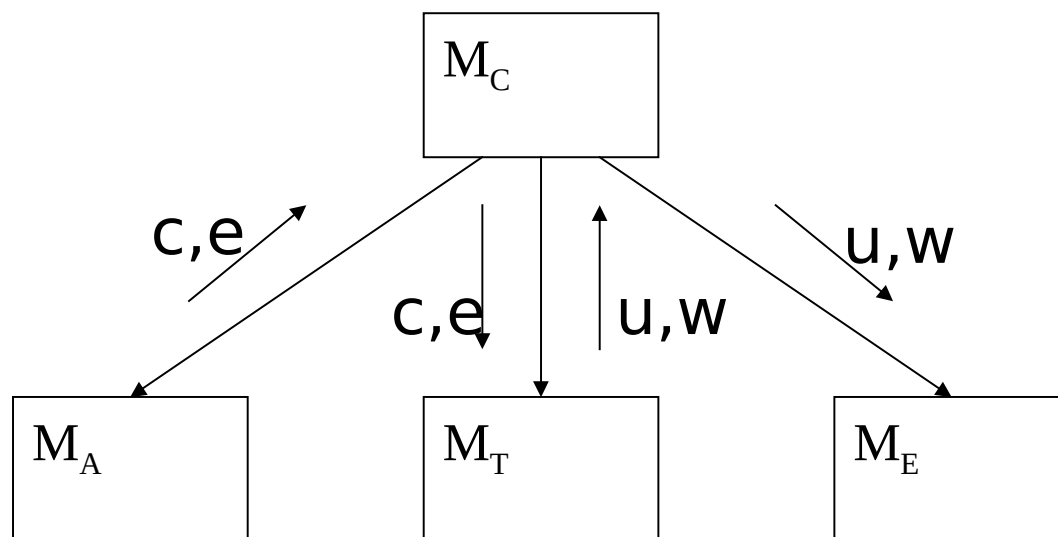
- 划分 **DFD**：划定输入、输出流边界，孤立变换中心。
- 一级分解：形成初始的 **SC** 图，顶层为主控模块，底层为输入、变换、输出模块。
- 二级分解：把 **DFD** 中的处理映射为程序结构中的适当的模块；从变换中心的边界开始沿输入、输出通道向外移动。
- 精化程序结构：以“模块独立”为指导思想，对模块或分或合，优化程序结构。



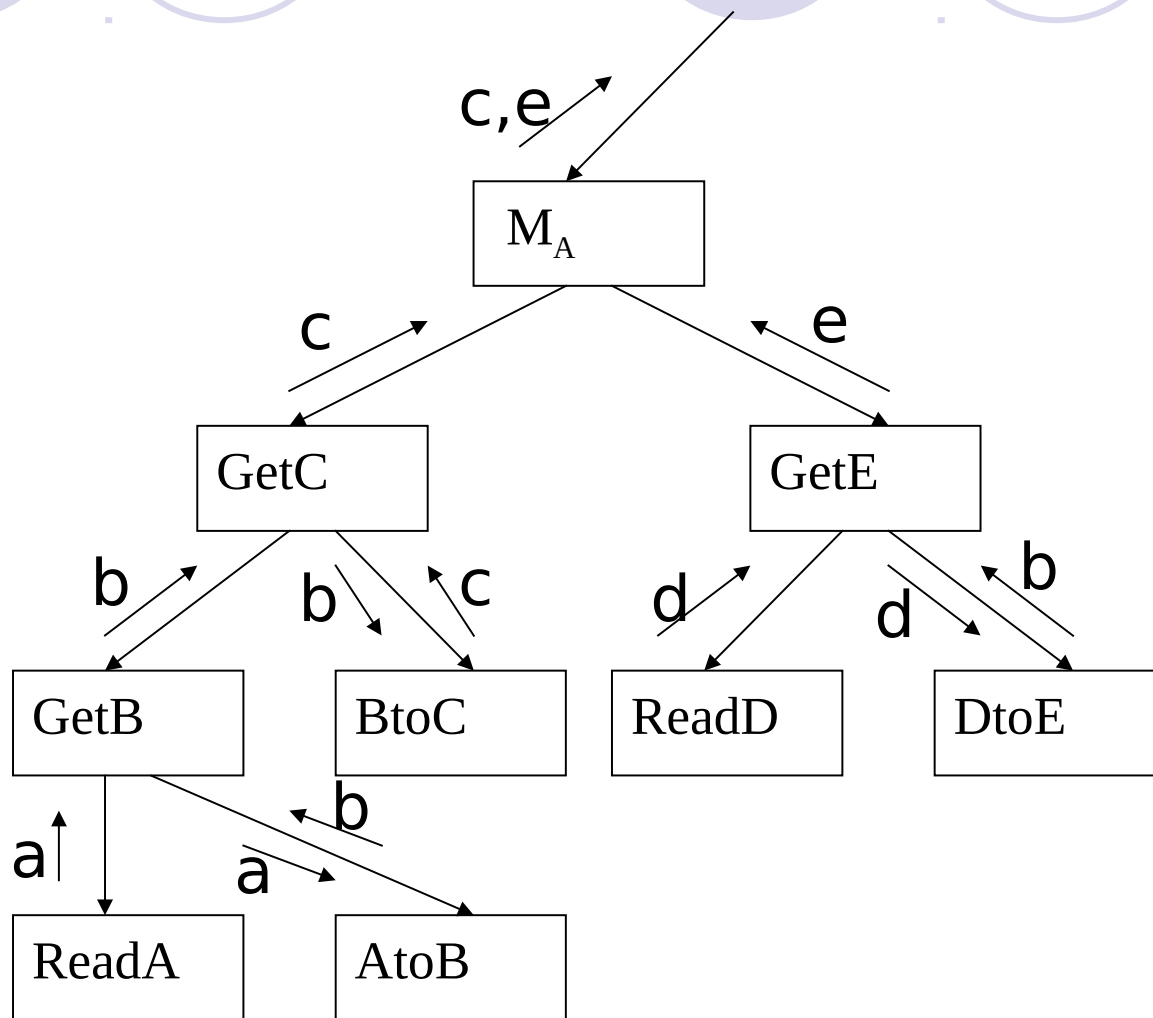
# 划分 DFD



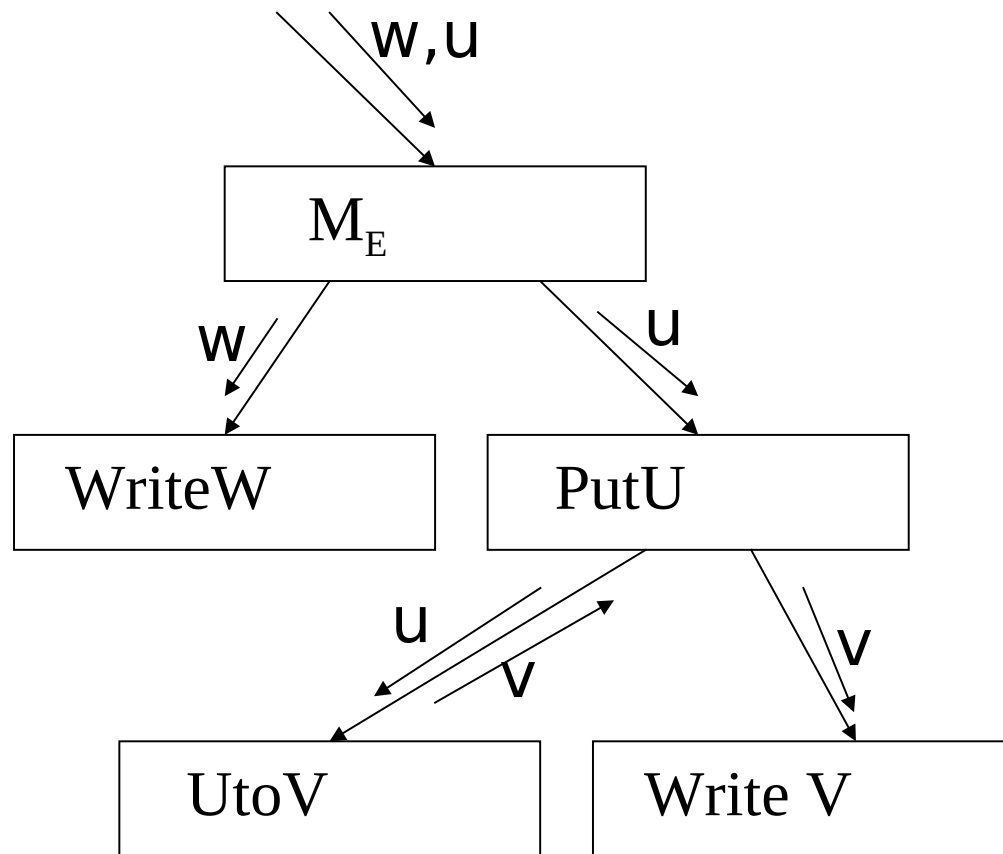
# 一级分解



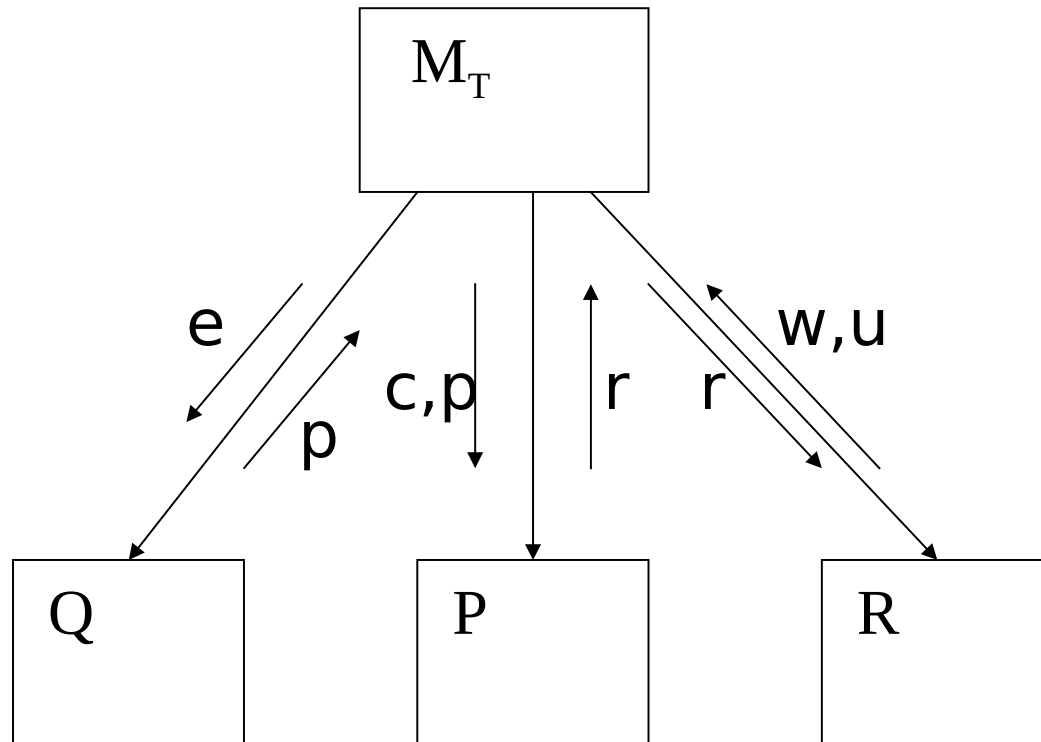
# 输入分支的分解



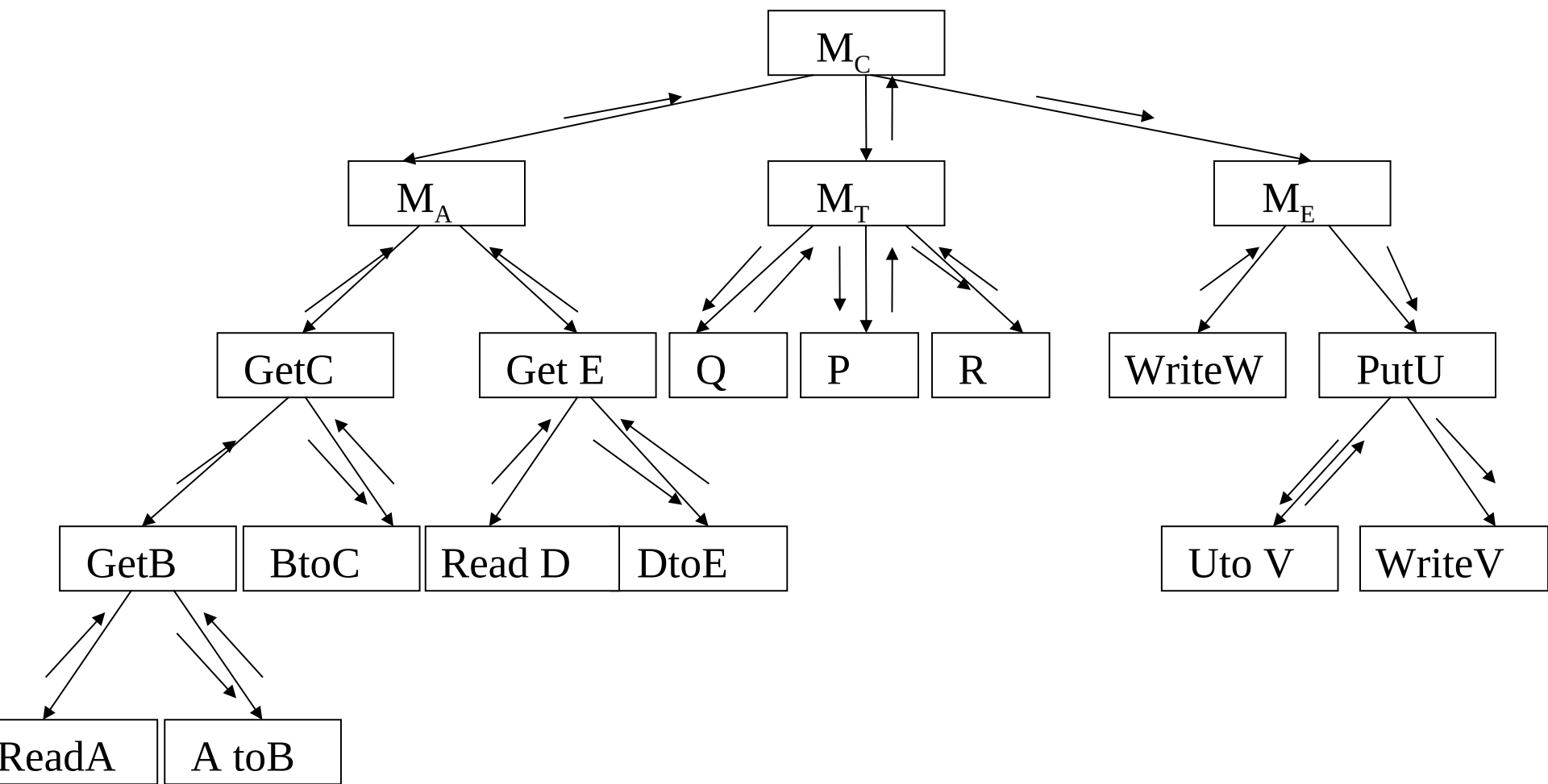
# 输出分支的分解



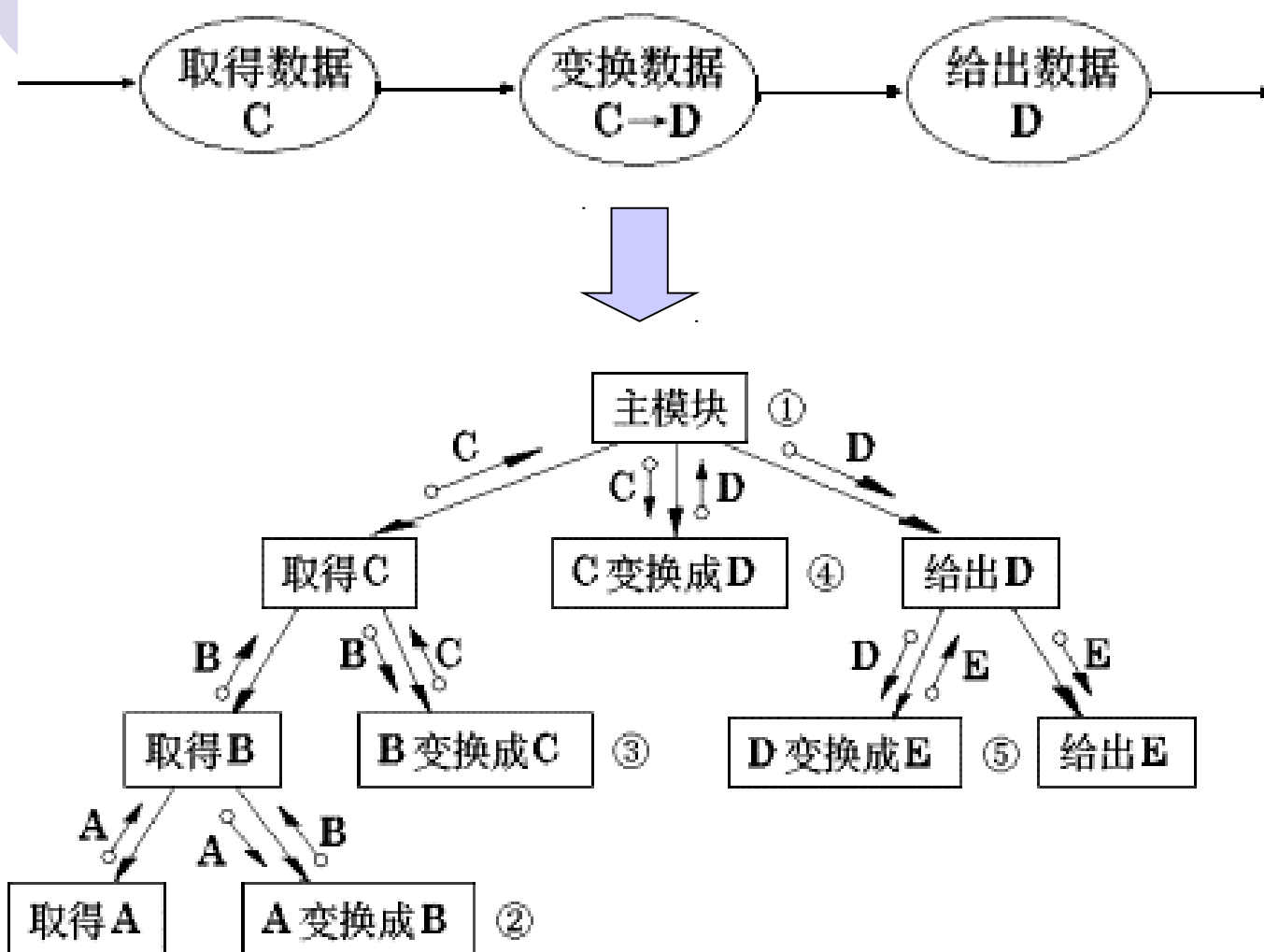
# 变换中心的分解



# 生成的 SC 图



# 变换分析——小结

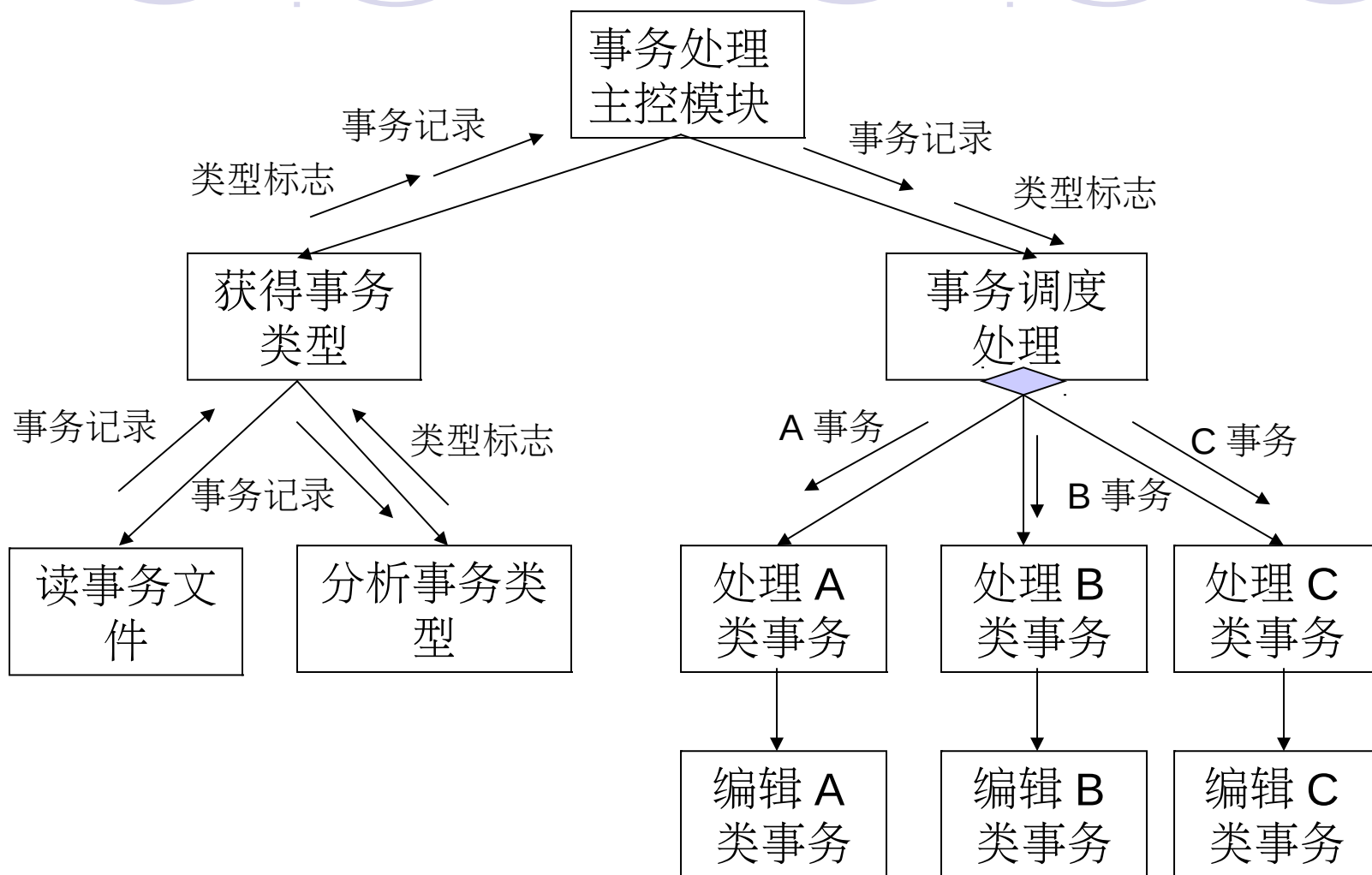


# 事务分析

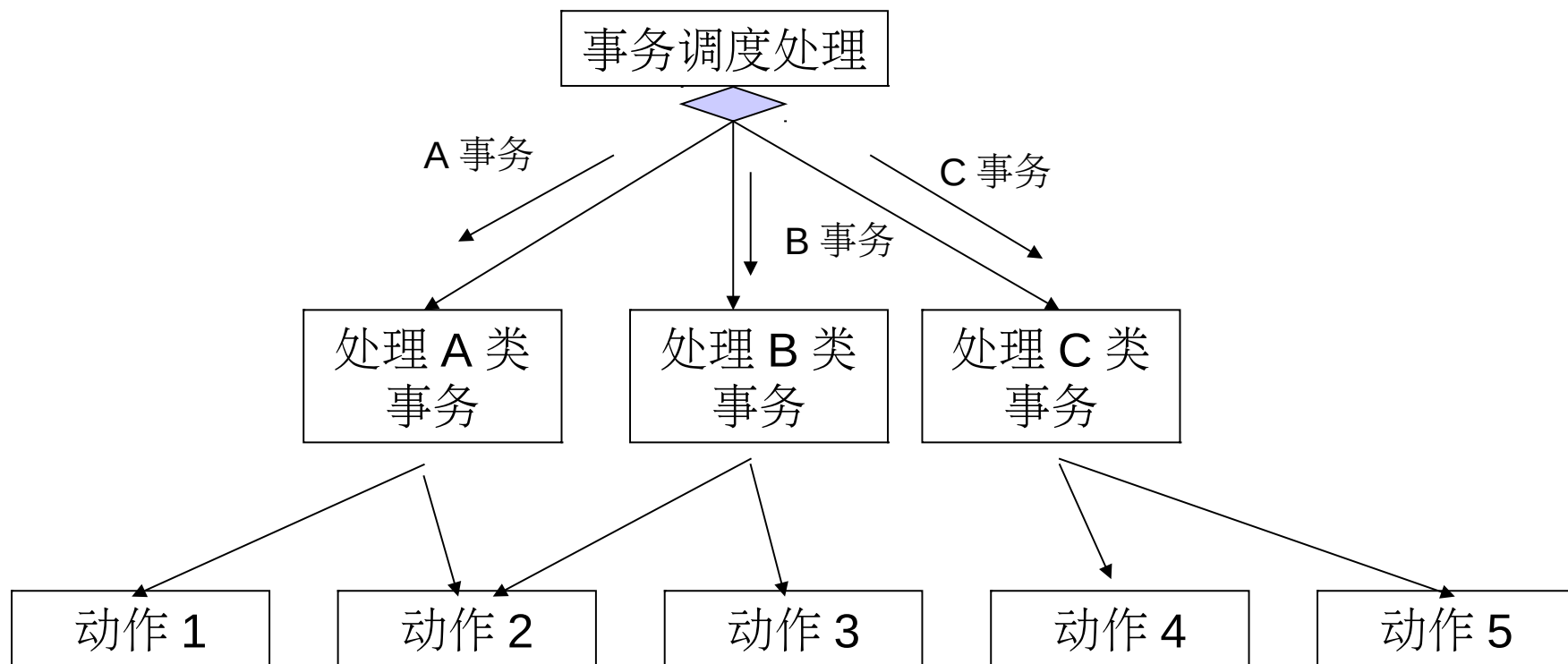
- 在 DFD 图上确定边界
  - 事务中心
  - 接受部分（包括接受路径）
  - 发送部分（包括全部动作路径）
- 画出 SC 图框架
  - DFD 图的三个部分分别映射为事务控制模块，接受模块和动作发送模块
- 分解和细化接受分支和发送分支



# 典型的事务分析后的 SC 图



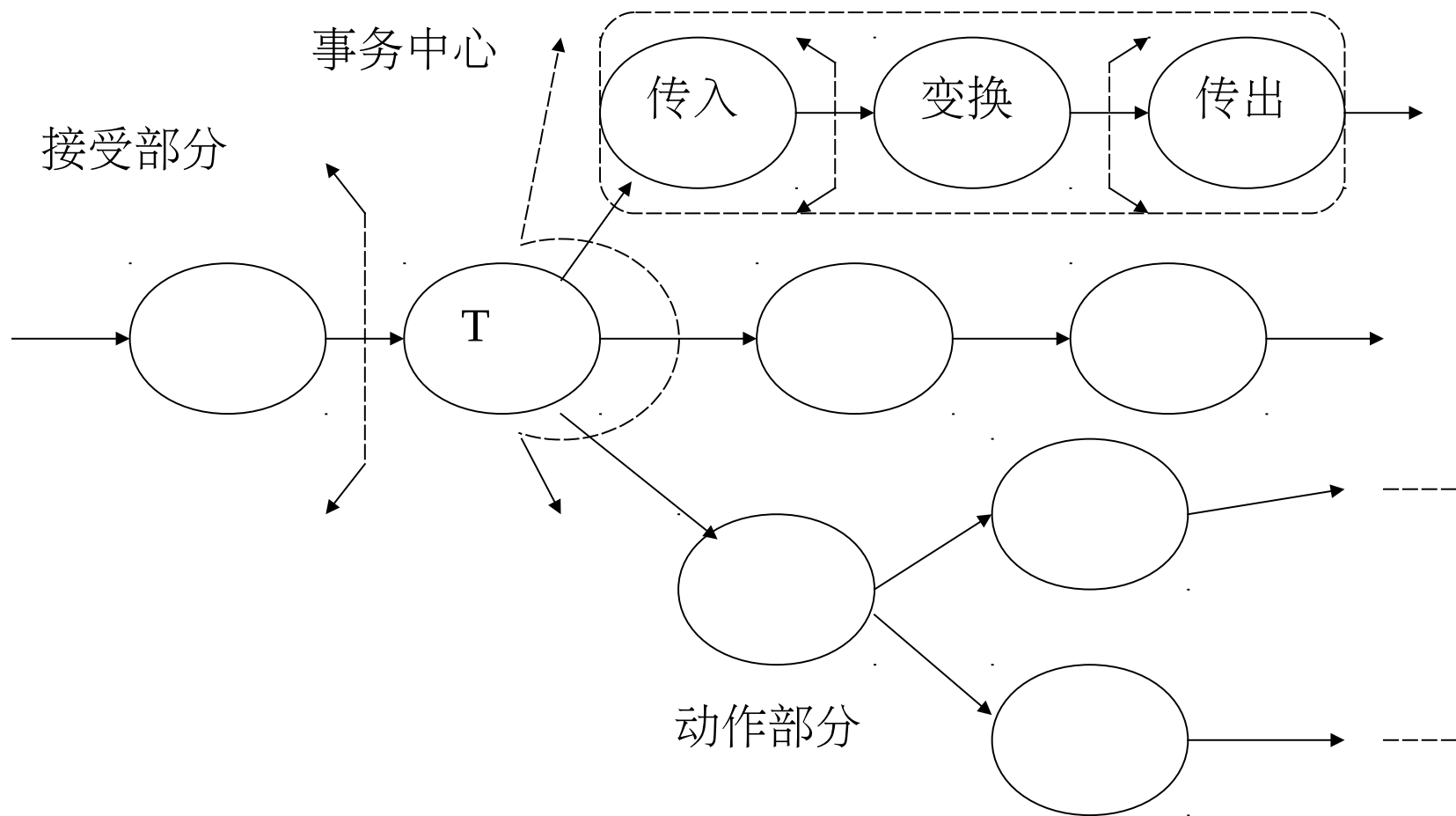
# 事务调度处理模块结构图



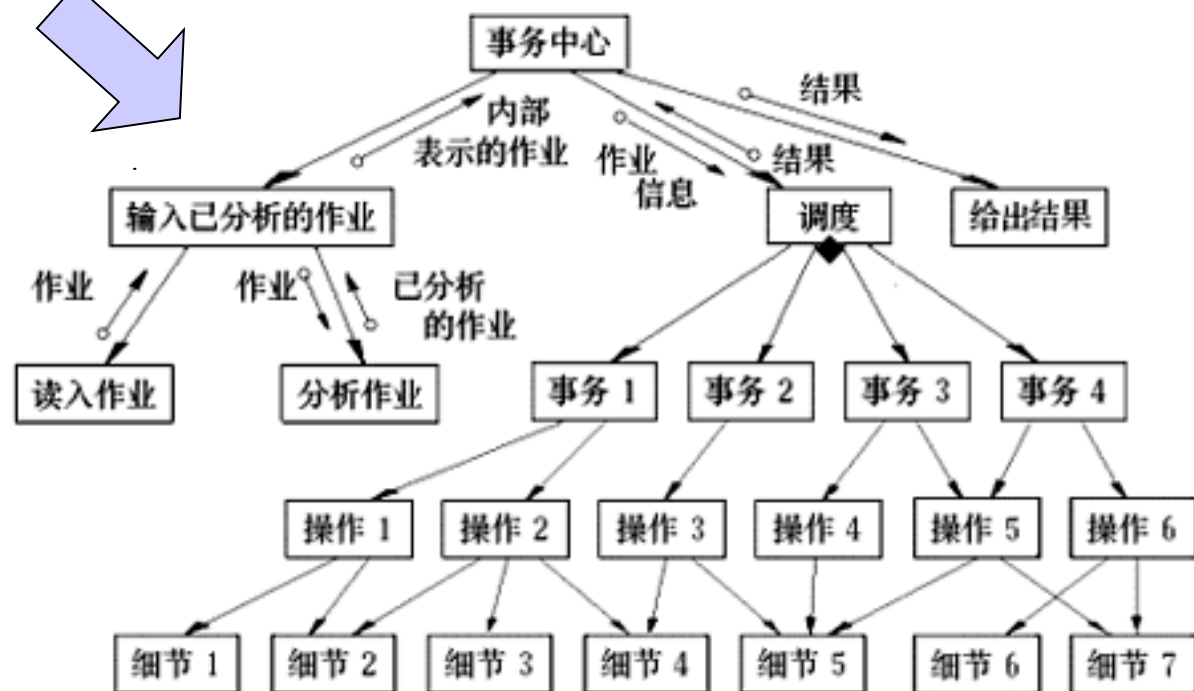
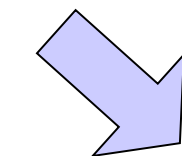
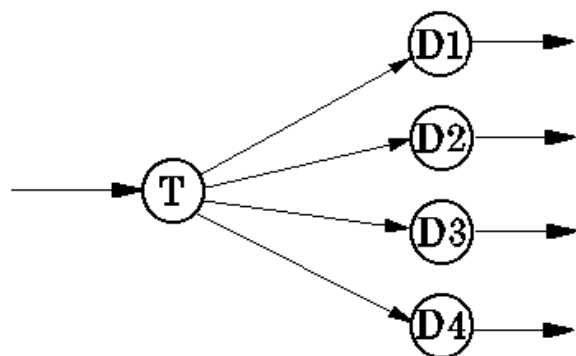
# 事务分析具体步骤

- 划分 **DFD**：指出事务中心，确定从事务中心出发的所有动作路径。
- 一级分解：形成初始的 **SC** 图，顶层为主控模块，底层为输入、散转模块。
- 二级分解：把输入路径和事务处理动作路径中的处理映射为程序结构中的适当的模块。
- 精化程序结构：以“模块独立”为指导思想，对模块或分或合，优化程序结构。

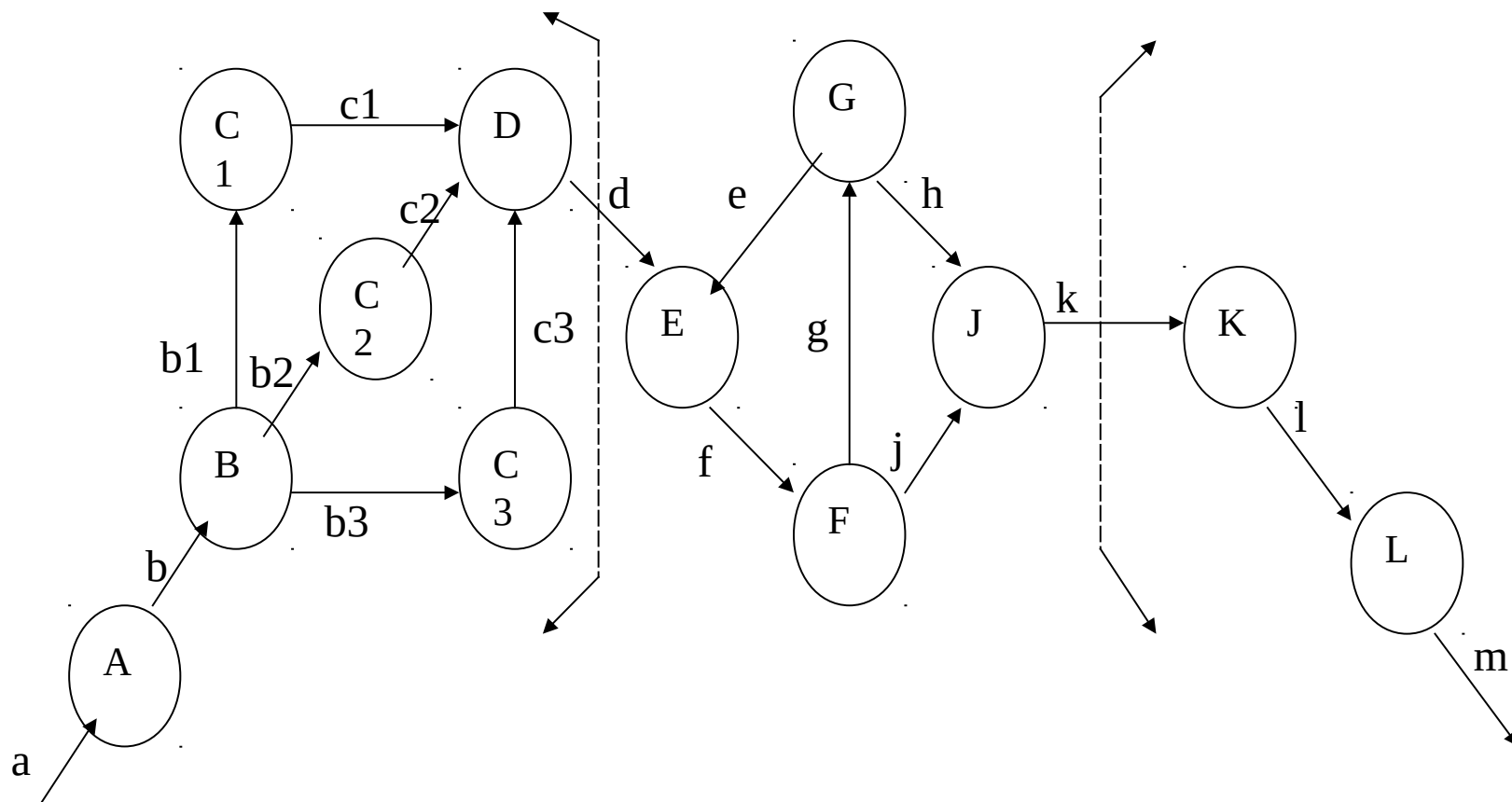
# 划分 DFD



# 事务分析——小结



# 混合结构



# 模块化设计 (modular design)

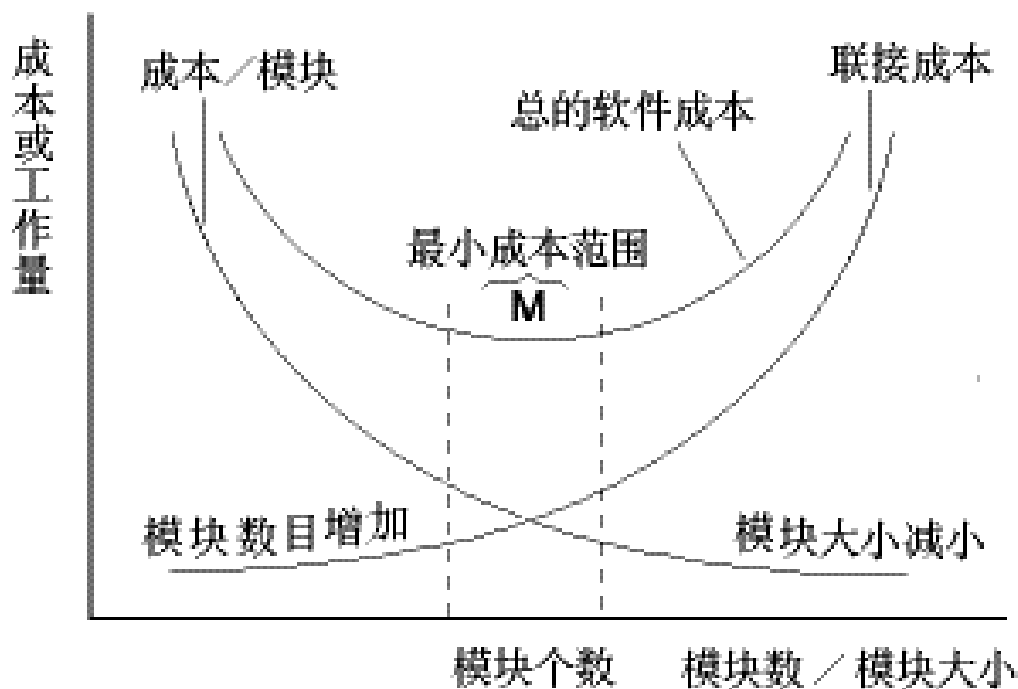
- 分解 ( decomposition )
- 关于模块的一些概念
- 模块独立性 ( module independence )

# 分解 (decomposition)

$$C(P_1 + P_2) > C(P_1) + C(P_2)$$

$$E(P_1 + P_2) > E(P_1) + E(P_2)$$

C 为问题的复杂度， E 为解题需要的工作量

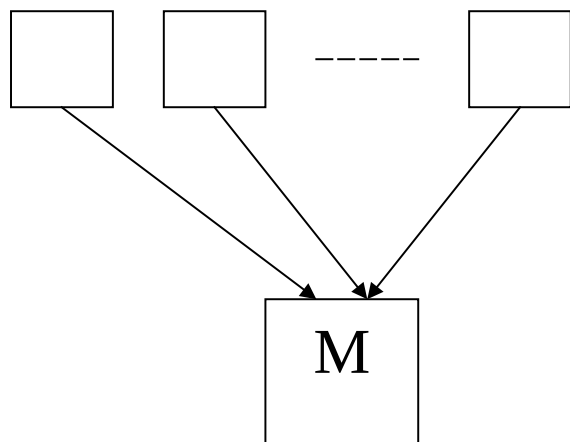
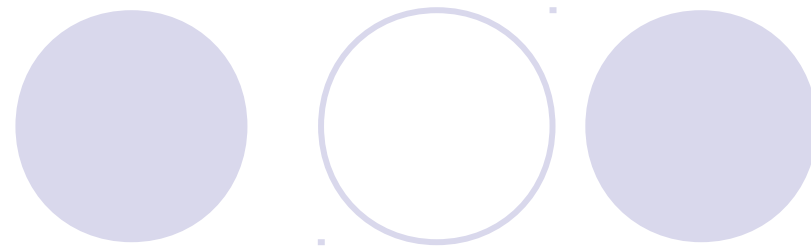




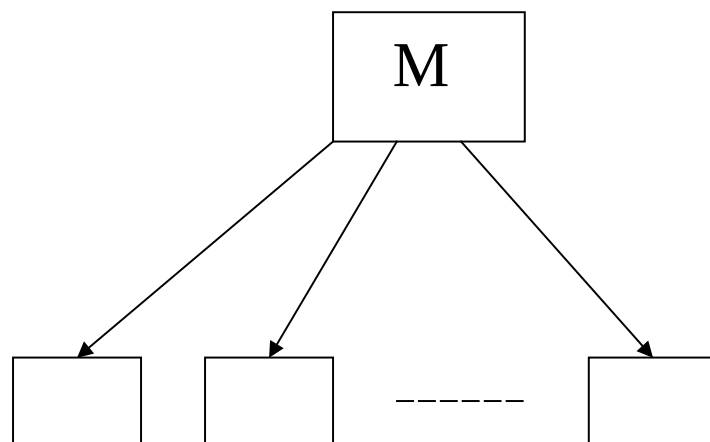
# 系统（模块结构）的深度与宽度

- 深度
  - 系统结构中的控制层数。
- 宽度
  - 同一层次的模块总数的最大值。

# 模块的扇入和扇出



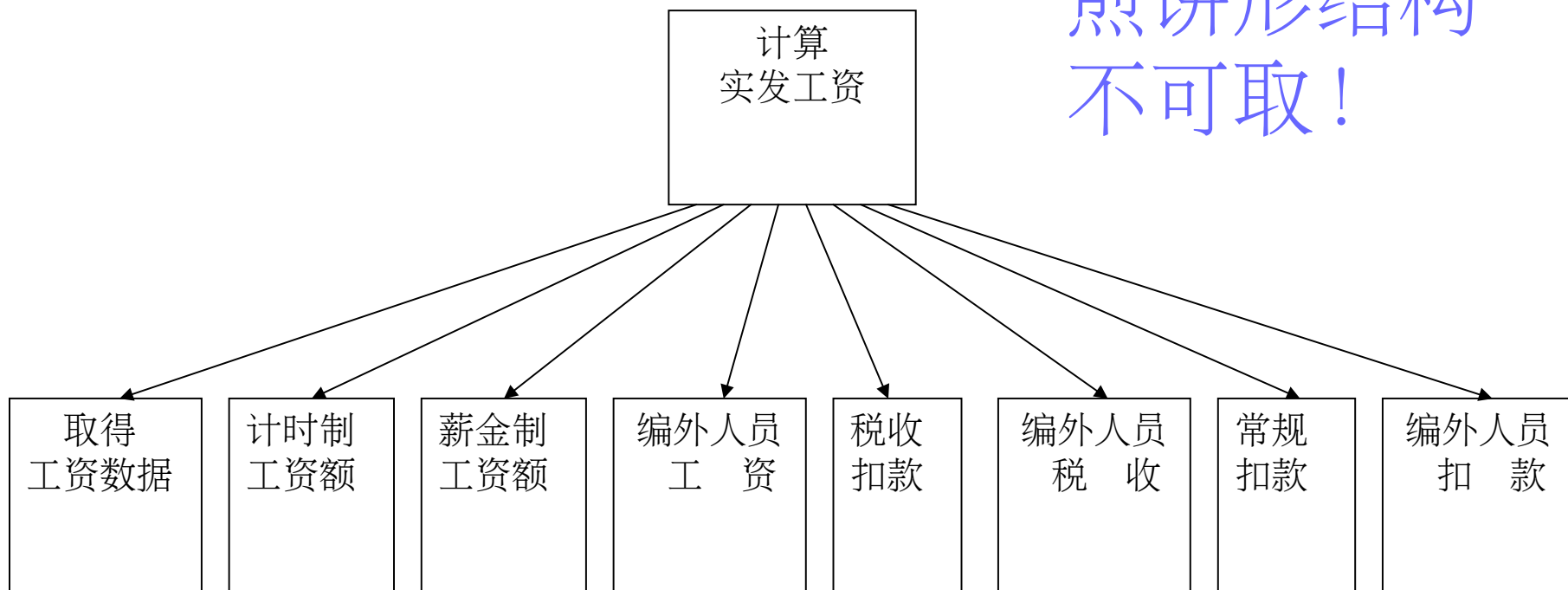
M 的扇入



M 的扇出

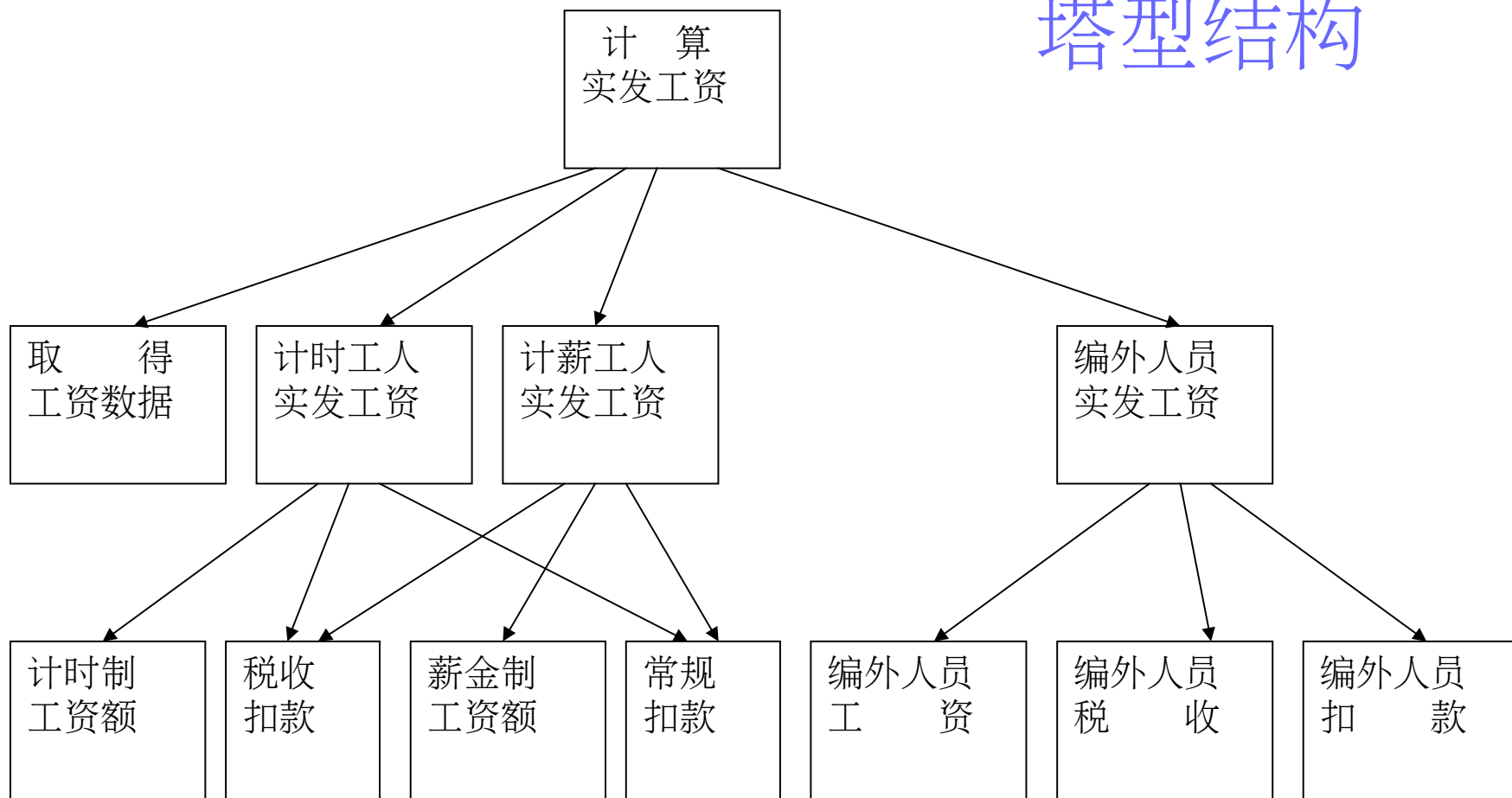
# 例子：扇出

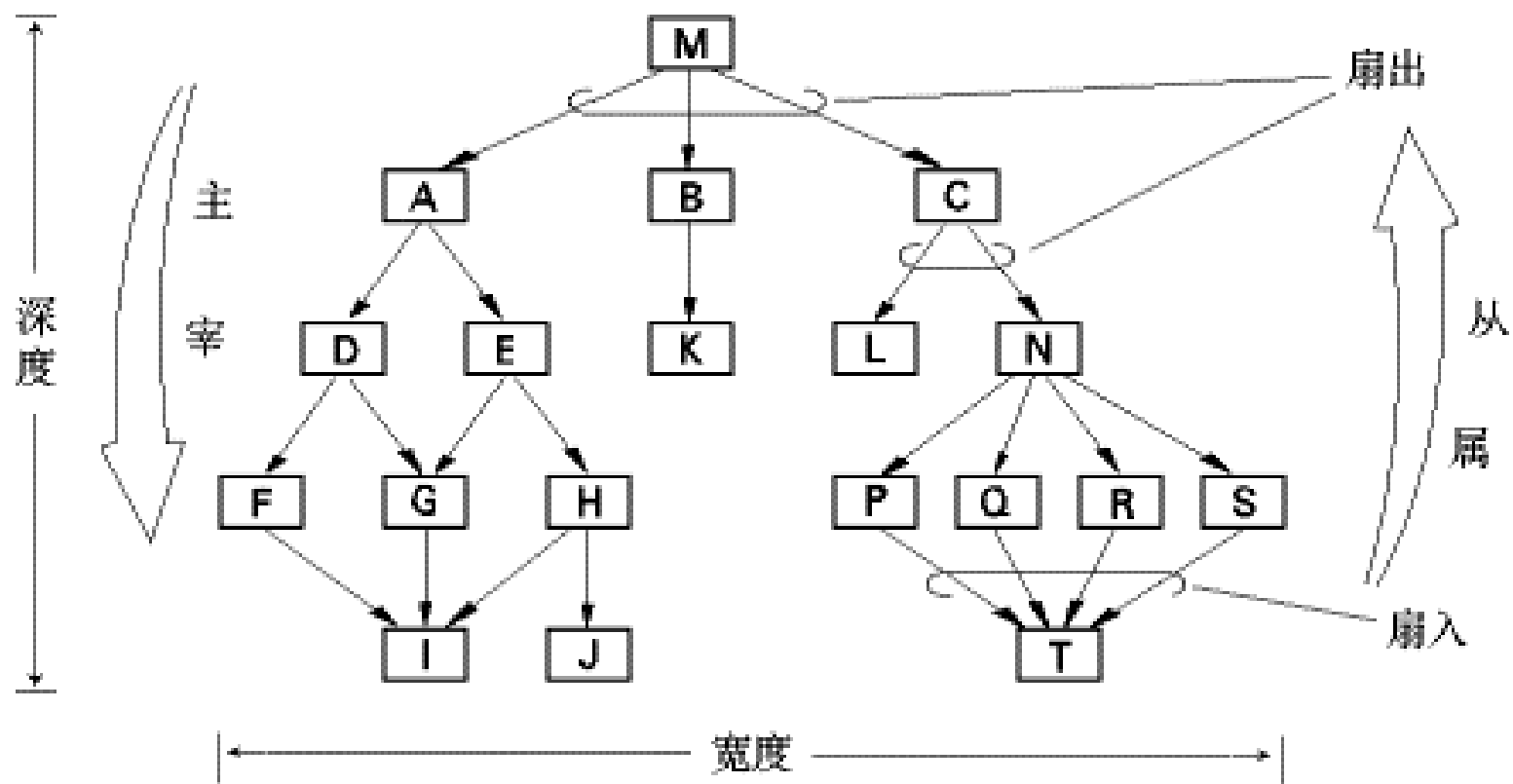
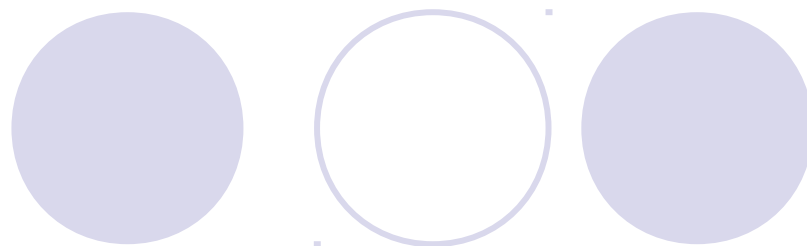
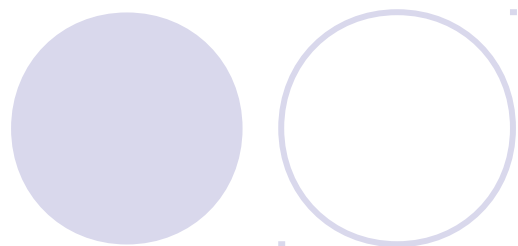
煎饼形结构  
不可取！



# 例子：扇出

## 塔型结构





# 模块的作用范围与控制范围

- 作用范围

- 受到该模块内部一个判定影响的所有模块的集合。

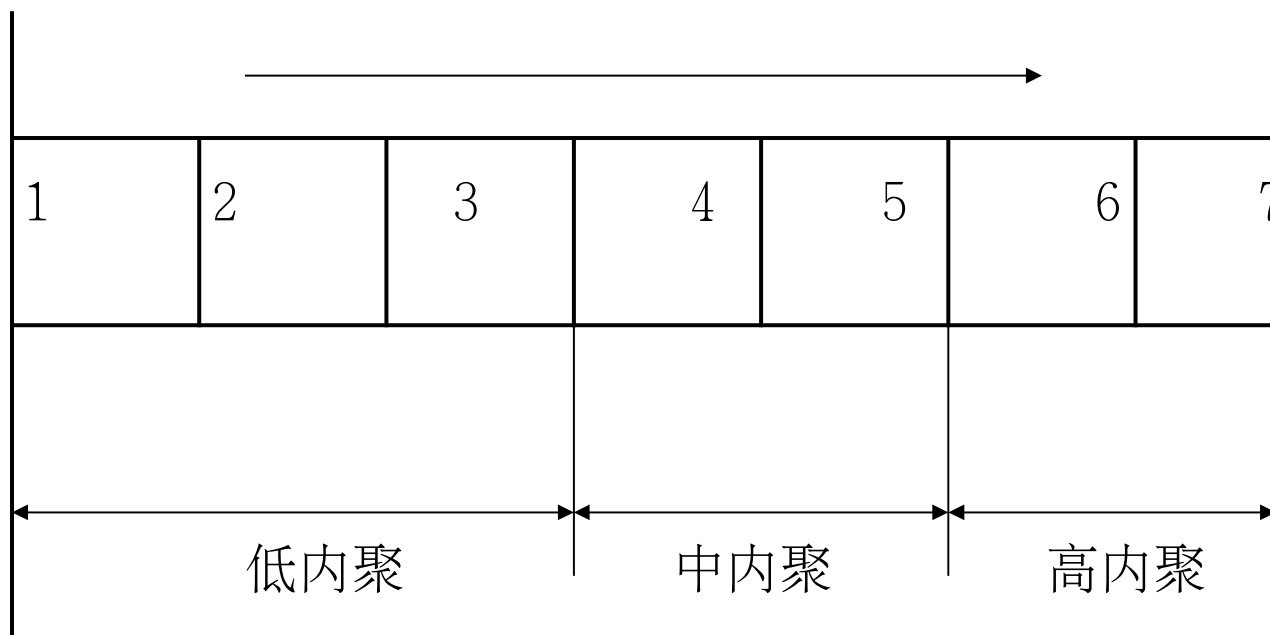
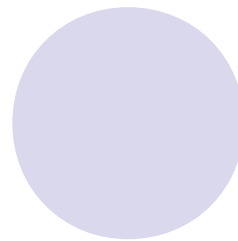
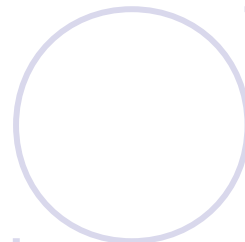
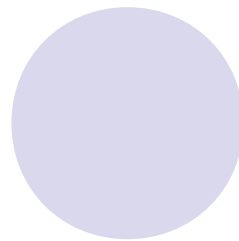
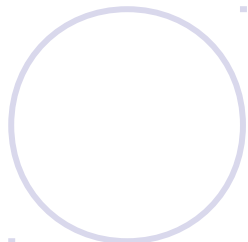
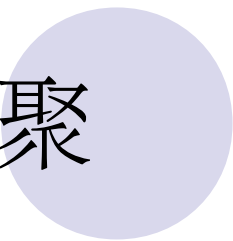
- 控制范围

- 包括该模块本身及所有下属模块的集合。

# 模块独立性 ( module independence )

- 内聚 ( **cohesion** )
  - 模块内部各成分之间
- 耦合 ( **coupling** )
  - 一个模块与其它模块之间
- 模块的独立性高
  - 块内联系强 / 块内聚合大
  - 块间联系弱 / 块间耦合小

# 内聚





# 内聚 cohesion

- 1 . 偶然性内聚
- 2 . 逻辑性内聚
- 3 . 时间性内聚
- 4 . 过程性内聚
- 5 . 通信性内聚
- 6 . 顺序性内聚
- 7 . 功能性内聚

coincidental cohesion

logical cohesion

temporal cohesion

procedural cohesion

communicational

cohesion

sequential cohesion

functional cohesion

# 偶然性内聚

- 模块中所要完成的各个动作之间没有任何关系。
- 常见：发现多个模块中有一组语句重复出现，就把这组语句抽出单独成为一个模块。

# 逻辑性内聚

- 模块中的各个组成部分在逻辑上具有相似的处理动作，但功能上、用途上却彼此无关。
- 常见：把多个逻辑上类似的模块（比如输出报表模块）合并到一个模块中。

# 时间性内聚



- 模块中所包含的各个处理动作必须在同一时间内执行。
- 常见：初始化模块。

# 过程性内聚



- 模块内各个成分彼此相关，并且按照特定的次序执行。
- 常见：由程序流程图直接演变而来的模块。

# 通信性内聚

- 模块中的各个成分都对数据结构的同一区域进行操作，以达到通信的目的。
- 常见：模块内各个动作都使用同一个输入数据或者产生同一个输出数据。

# 顺序性内聚



- 模块内各个处理成分都与同一个功能相关，并且这些处理必须顺序执行。
- 常见：前一部分处理动作的输出是后一部分处理动作的输入。

# 功能性内聚



- 模块内所有成分形成一个整体，完成单个的功能。
- 常见：单一目的，单一功能，界面清楚。  
(比如求平方根模块)



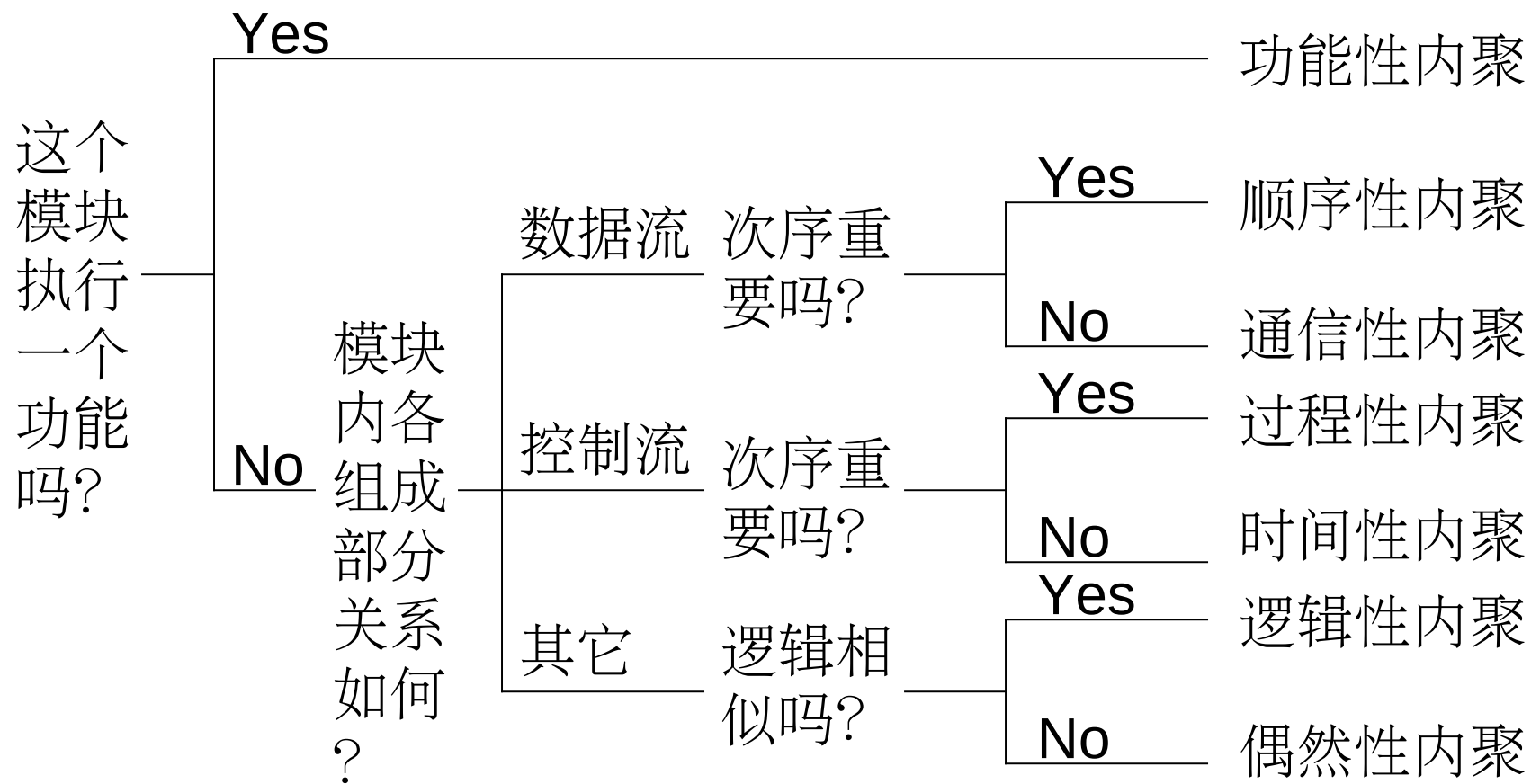
# 块内七种聚合形式的比较

| 块内聚合  | 联接形式 | 可修改性 | 可读性 | 通用性 | 联系程度 |
|-------|------|------|-----|-----|------|
| 功能性聚合 | 好    | 好    | 好   | 好   | 高    |
| 顺序性聚合 | 好    | 好    | 好   | 中   | ↑    |
| 通信性聚合 | 中    | 中    | 中   | 不好  |      |
| 过程性聚合 | 中    | 中    | 中   | 不好  |      |
| 时间性聚合 | 不好   | 不好   | 中   | 最坏  |      |
| 逻辑性聚合 | 最坏   | 最坏   | 不好  | 最坏  | ↓    |
| 偶然性聚合 | 最坏   | 最坏   | 最坏  | 最坏  | 低    |

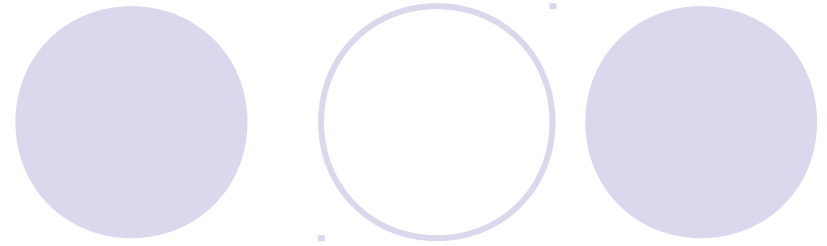
# 模块的内聚



# 块内聚合形式的判定



# 耦合 coupling




- 1. 非直接耦合 no direct coupling
- 2. 数据耦合 data coupling
- 3. 特征耦合 stamp coupling
- 4. 控制耦合 control coupling
- 5. 外部耦合 external coupling
- 6. 公共耦合 common coupling
- 7. 内容耦合 content coupling

# 非直接耦合

- 两个模块中的任何一个都不依赖于对方能够独立工作。

# 数据耦合



- 两个模块通过参数传递信息，信息仅限于数据。

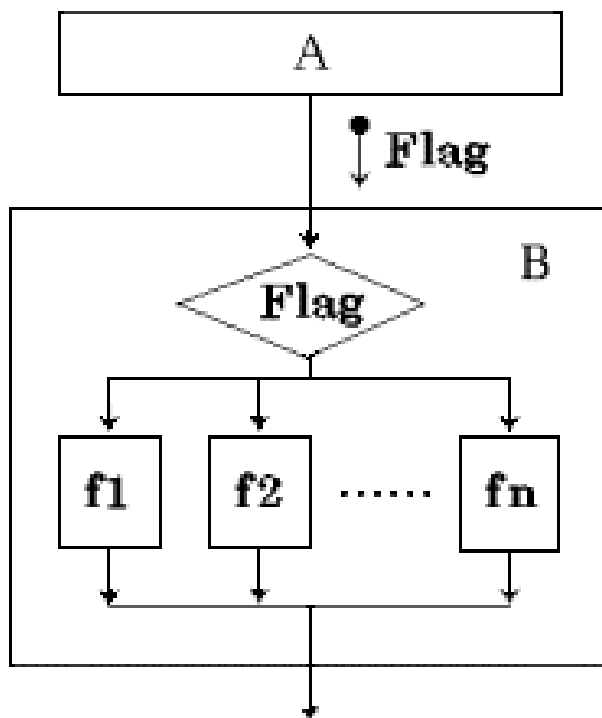
# 特征耦合



- 模块间通过参数传递复杂的数据结构，此数据结构的变化将使相关的模块发生变化。这种耦合介于数据耦合与控制耦合之间。

# 控制耦合

- 两个模块通过参数传递信息，信息中包含控制。





# 外部耦合



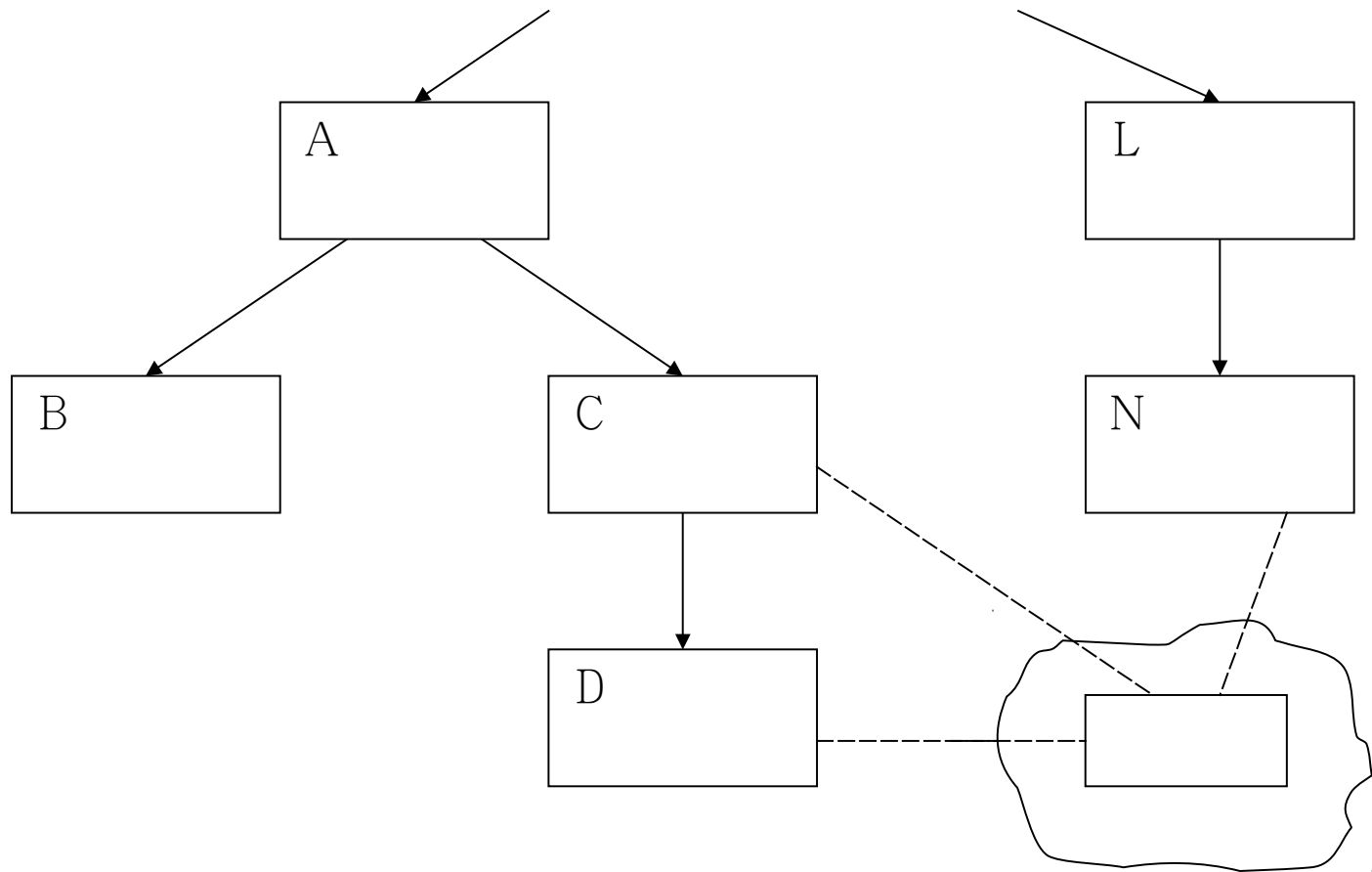
- 多个模块与同一个外部环境关联。
- 常见：多个 I/O 模块与特定的设备、格式和通信协议相关联。

# 公共耦合



- 多个模块通过全局的数据环境相互作用。
- 全局数据环境包括：全局变量、公共区、内存公共覆盖区、任何存储介质上的文件、物理设备等。

# 公共耦合

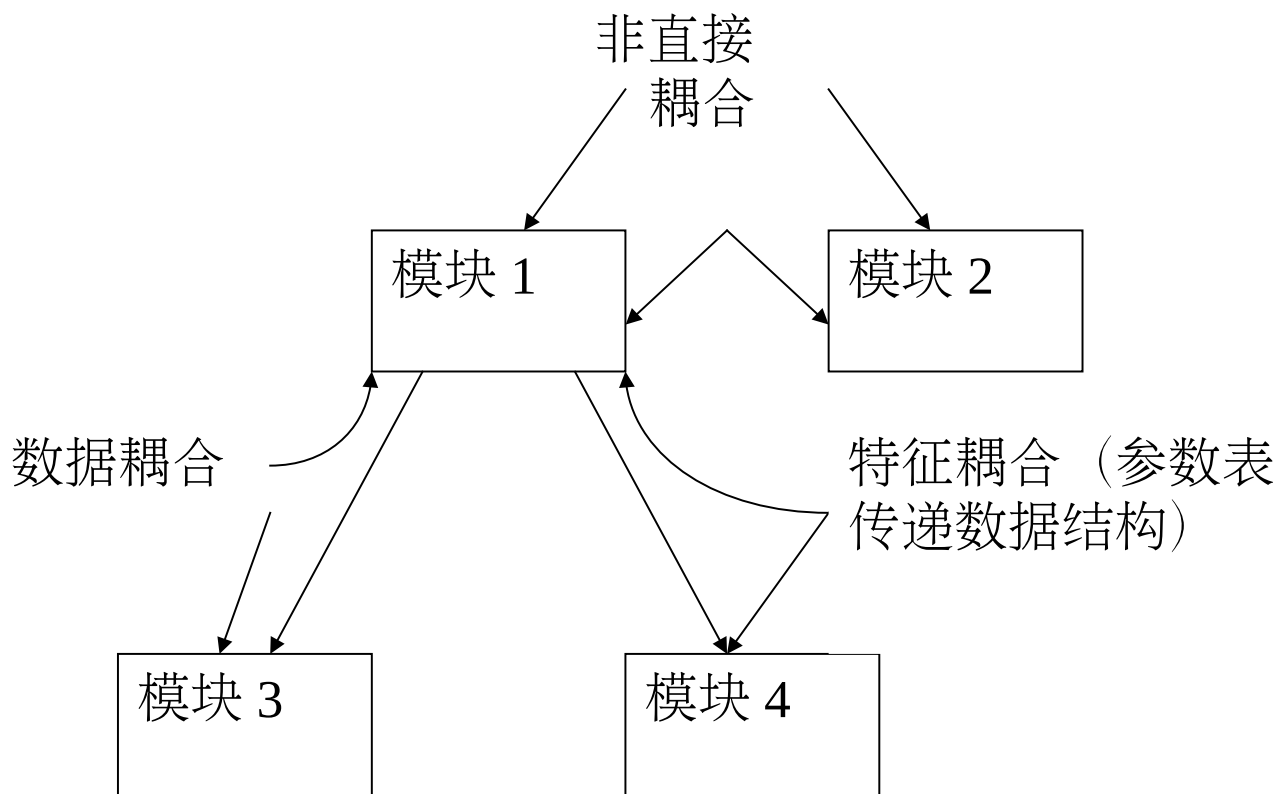


# 内容耦合

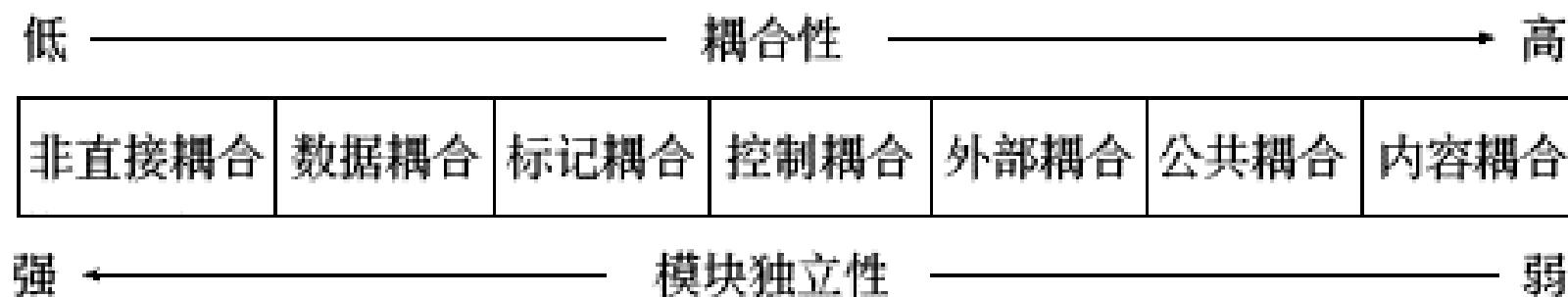


- 一个模块使用另外一个模块的内部数据或者控制信息；一个模块直接转移到另外一个模块内部。

# 弱耦合



# 模块的耦合



# 基于块间耦合的设计原则

- 尽量使用数据耦合
- 减少使用控制耦合
- 限制外部环境耦合和公共耦合
- 杜绝内容耦合

# 体系结构设计优化的指导规则

- 设计阶段早期应该对程序结构多做精化和评估，以达到最好。
- 便于优化是开发软件体系结构表示的一个重要因素。
- 结构上的简单往往反映出程序的优雅和高效。
- 设计优化应在满足模块化要求的前提下尽量减少模块数量。
- 在满足信息需求的前提下尽量减少复杂的数据结构。



# 体系结构设计优化的指导规则

- 对模块分割、合并和变动调用关系的指导规则
  - 提高内聚，降低耦合后
  - 简化模块接口
  - 少用全局性数据和控制型信息
- 保持高扇入 / 低扇出的原则
- 作用域 / 控制域规则
  - 作用域不要超出控制域的范围
  - 位置离受它控制的模块越近越好

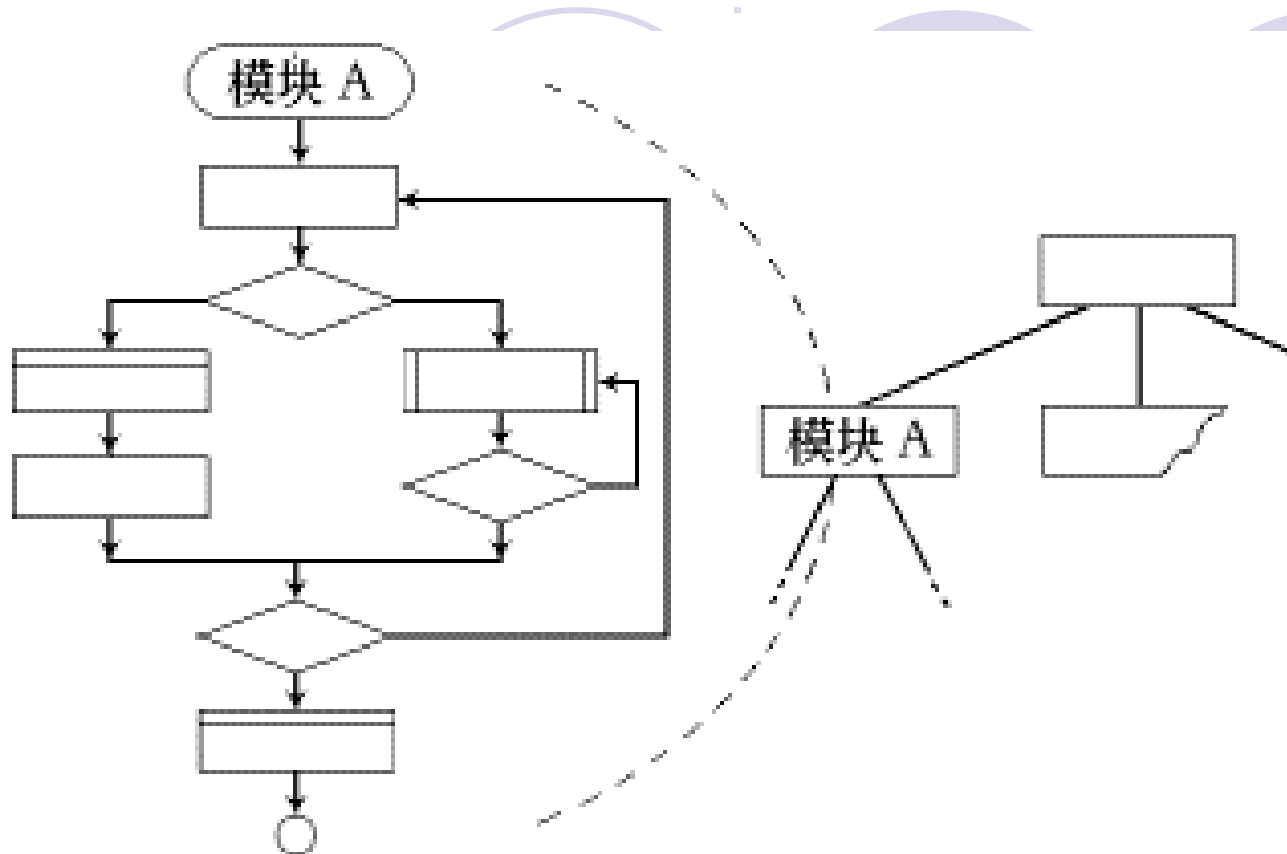
# 体系结构设计优化的指导规则

- 对于性能要求高的应用，有时需要进行如下的工作：
  - 开发和精化程序结构，且不考虑关键性能而进行的优化。
  - 使用可以提高运行性能的 **CASE** 工具来孤立低效的部分。
  - 在后期设计中，对有可能最消耗时间的模块的算法进行时间优化。
  - 用适当的程序设计语言编码。
  - 孤立出那些大量占用处理器时间的模块。
  - 如果需要，用依赖机器的语言重新设计和编码，以提高效率。
- 先使其工作起来，再设法使其更好地工作。

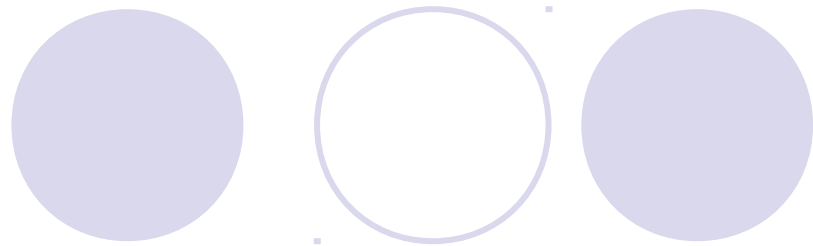
# 过程设计

- 目的
  - 确定模块采用的算法和块内数据结构
- 任务：编写软件的“过程设计说明书”
  - 为每个模块确定采用的算法
  - 确定每一模块使用的数据结构
  - 确定模块接口的细节

# 一个模块内的软件过程

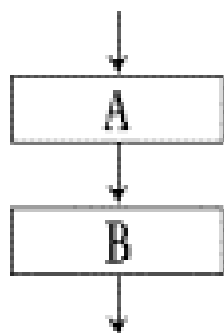


# 过程设计工具

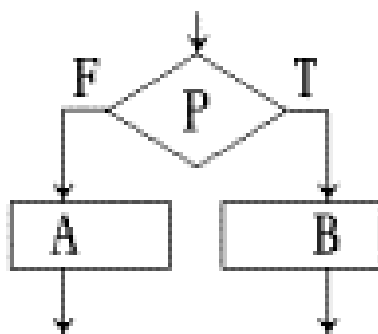


- 流程图
- N-S 图
- 伪代码
- PDL 语言

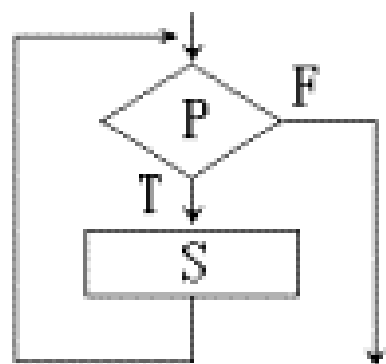
# 程序流程图



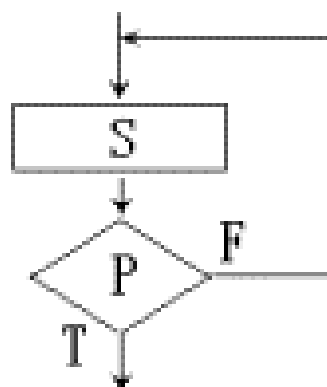
① 顺序型



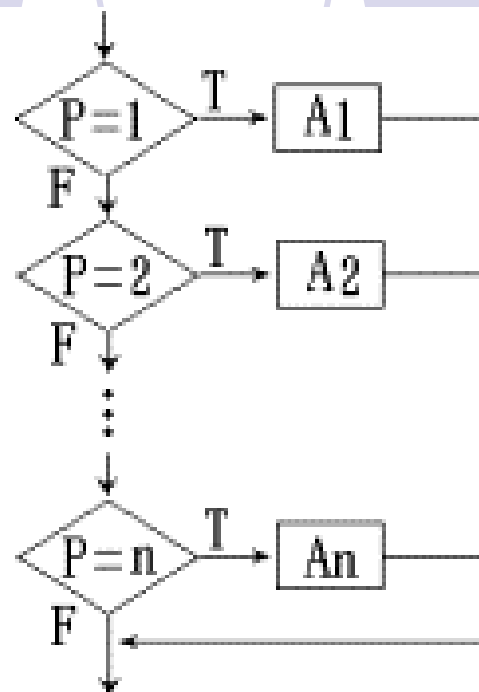
② 选择型



③ 先判定型循环  
(DO-WHILE)

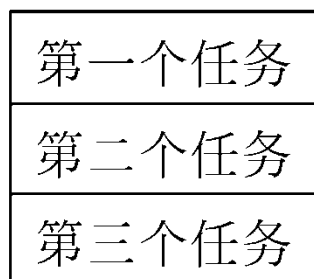


④ 后判定型循环  
(DO-UNTIL)

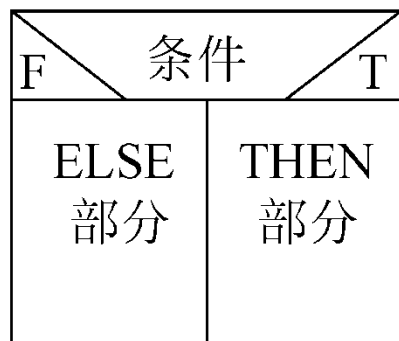


⑤ 多情况选择型  
(CASE 型)

# N-S 图



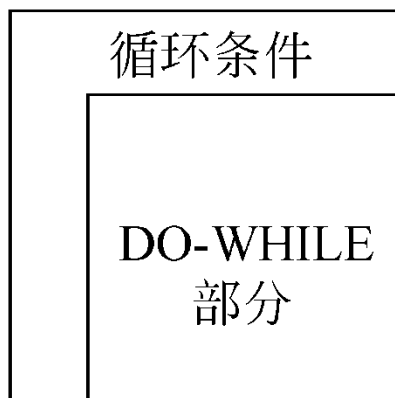
(a)



(b)



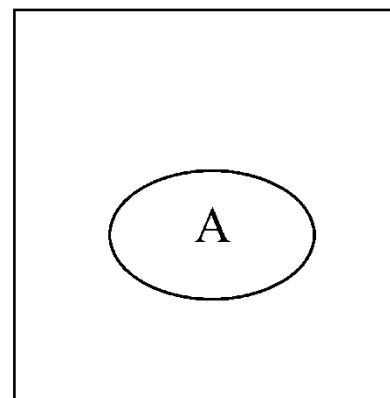
(c)



(d)



(e)



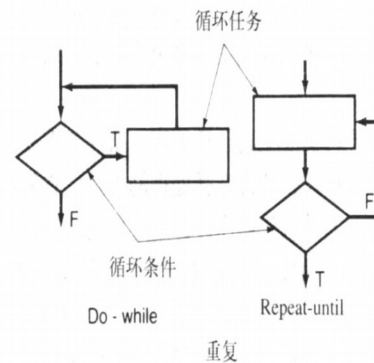
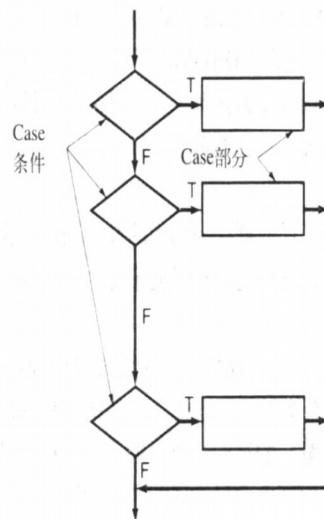
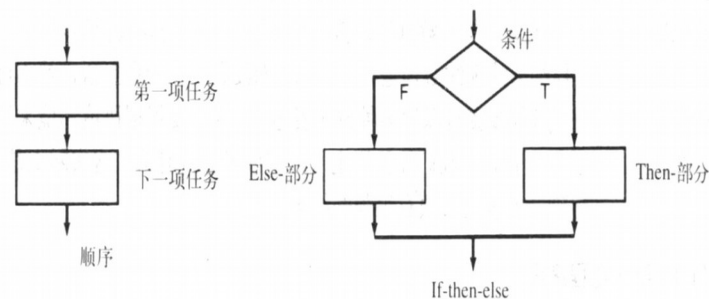
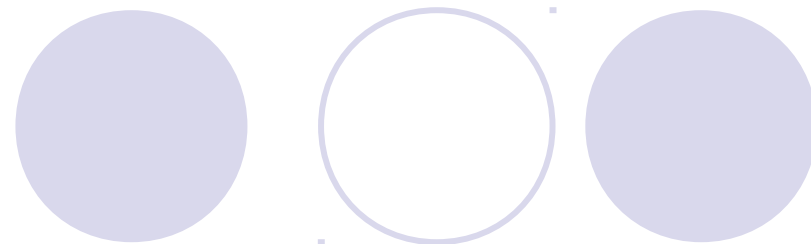




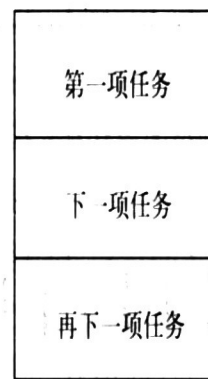
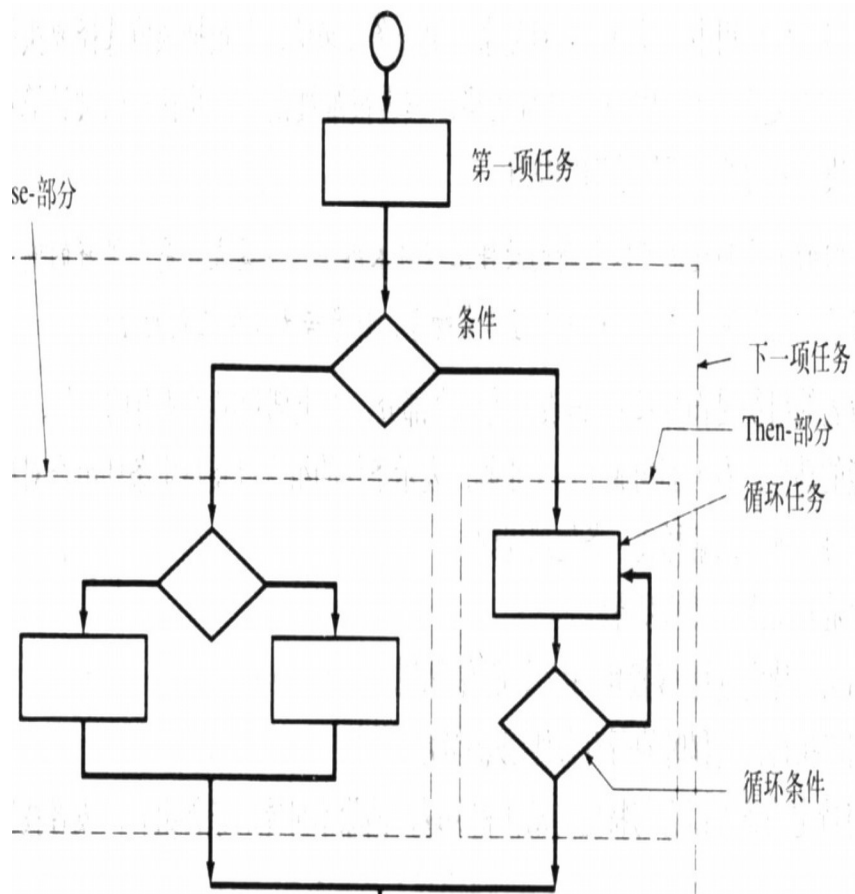
# 过程设计

- 过程设计应该在数据、体系结构和界面设计完成后进行。

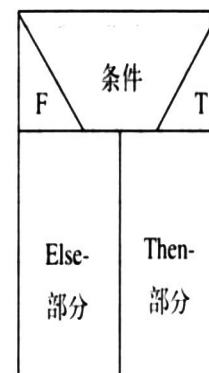
- 结构化程序设计



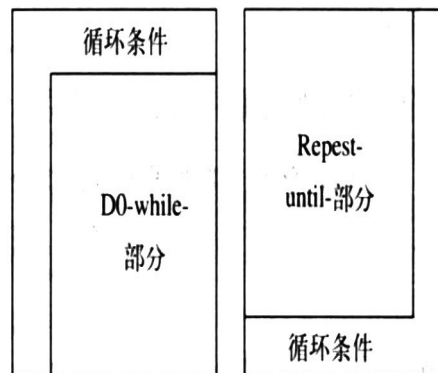
# 过程设计



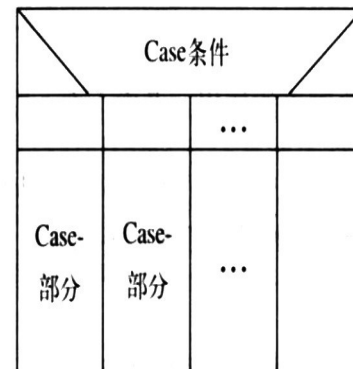
顺序



if-then-else



重复



选择