第5章 总体设计

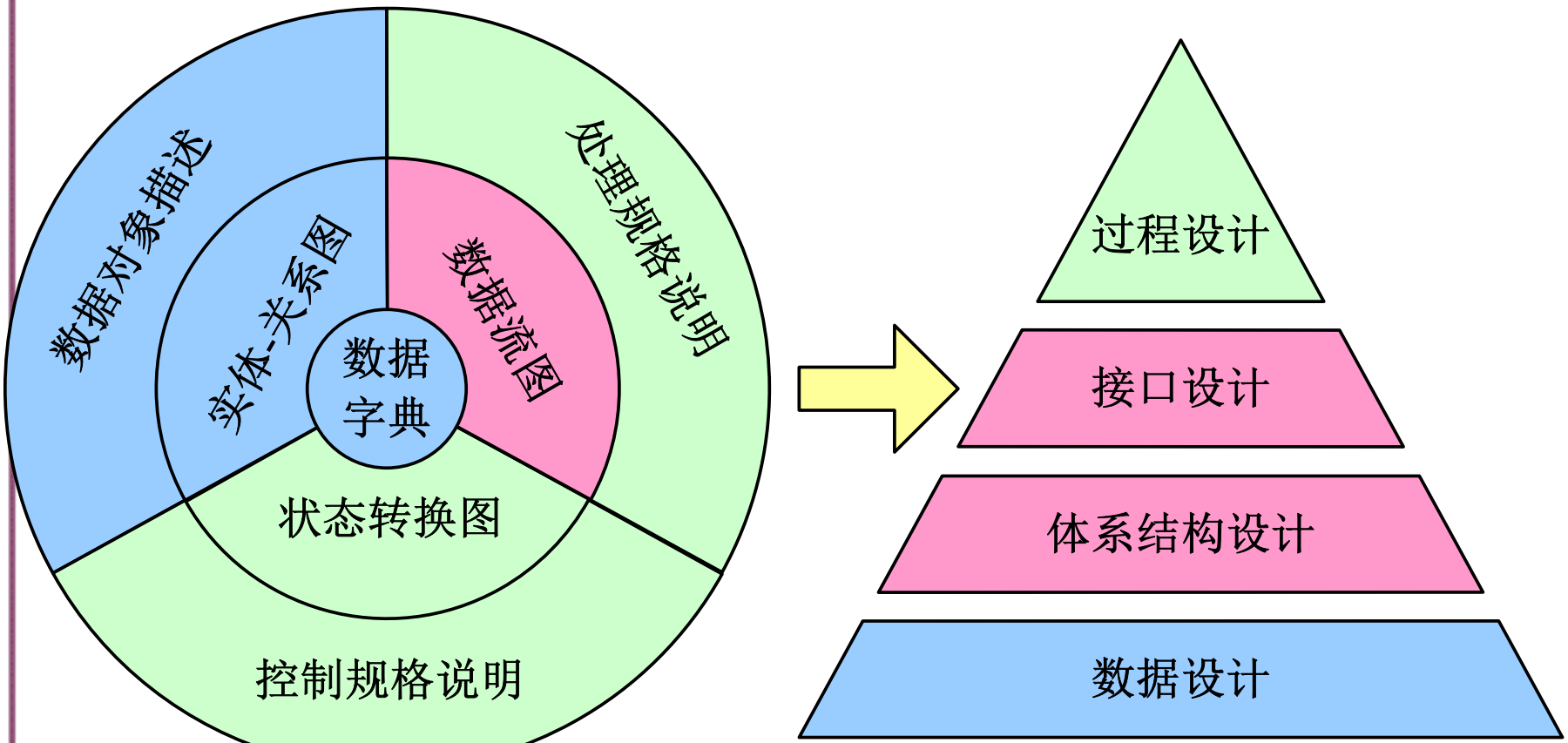
设计原理

启发规则

描绘软件结构的图形工具

面向数据流的设计方法

结构化设计和结构化分析的关系：



5.2 设计原理

5.2.1 模块化

- 模块：是由边界元素限定的相邻程序元素的序列，而且有一个总体标识符代表它。
- 模块化：就是把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。

模块化的根据：

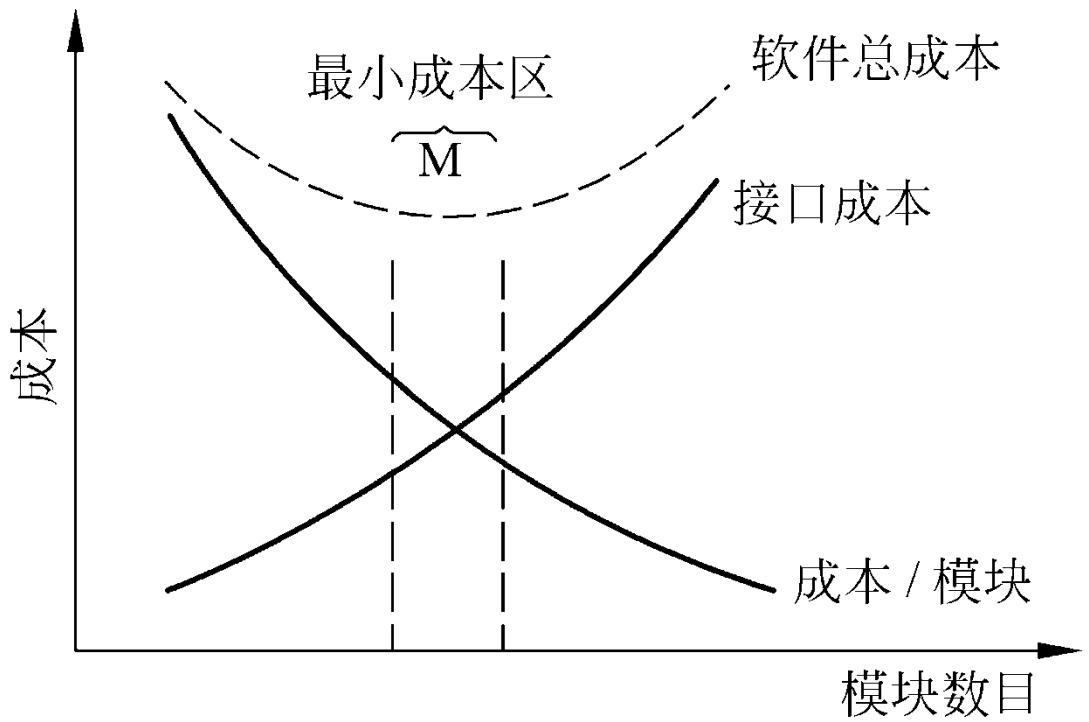
- 如果C(*P*1)>C(*P*2)，显然E(P1)>E(P2)
- 根据人类解决一般问题的经验，

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

- 综上所述，得到下面的不等式

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

- 每个程序都相应地有一个最适当的模块数目*M*，使得系统的开发成本最小。



模块化和软件成本

评价一种设计方法定义模块能力的五条标准：

- 模块可分解性 ■ ■
- 模块可组装性
- 模块可理解性 ■ ■
- 模块连续性
- 模块保护性 ■

5.2.2 抽象

- **抽象：**现实世界中一定事物、状态或过程之间总存在着某些相似的方面(共性)。把这些相似的方面集中和概括起来，暂时忽略它们之间的差异，这就是抽象。
- 抽象就是抽出事物本质特性而暂时不考虑细节。

5.2.3 逐步求精

- 逐步求精：为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。逐步求精是人类解决复杂问题时采用的基本方法，也是许多软件工程技术的基础。
- **Miller**法则：一个人在任何时候都只能把注意力集中在（7±2）个知识块上。

5.2.4 信息隐藏和局部化

- **信息隐藏：**应该这样设计和确定模块，使得一个模块内包含的信息(过程和数据)对于不需要这些信息的模块来说，是不能访问的。
- **局部化：**局部化的概念和信息隐藏概念是密切相关的。所谓局部化是指把一些关系密切的软件元素物理地放得彼此靠近。显然，局部化有助于实现信息隐藏。

5.2.5 模块独立

模块独立：

- 模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。
- 希望这样设计软件结构，使得每个模块完成一个相对独立的特定子功能，并且和其他模块之间的关系很简单。

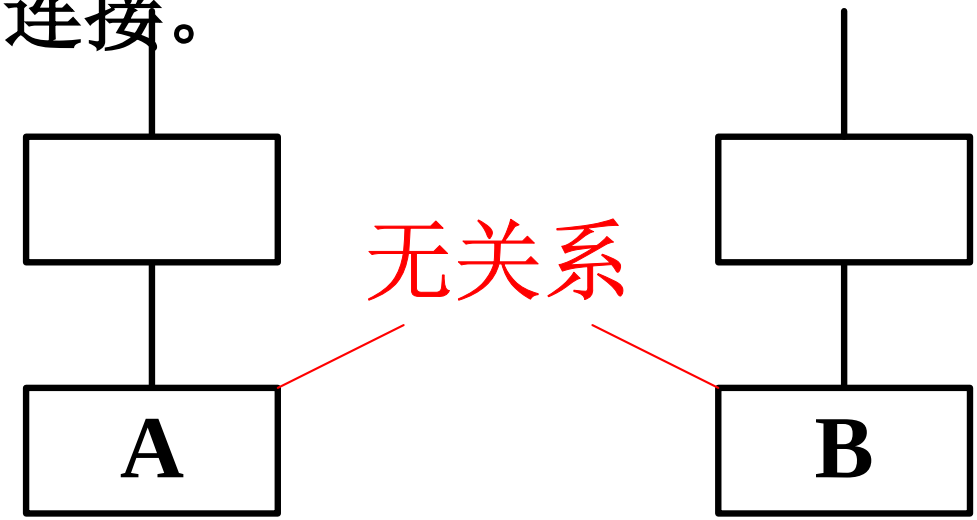
模块独立程度的两个定性标准度量：

- 耦合衡量不同模块彼此间互相依赖(连接)的紧密程度。耦合要低，即每个模块和其他模块之间的关系要简单；
- 内聚衡量一个模块内部各个元素彼此结合的紧密程度。内聚要高，每个模块完成一个相对独立的特定子功能。

耦合程度的度量：

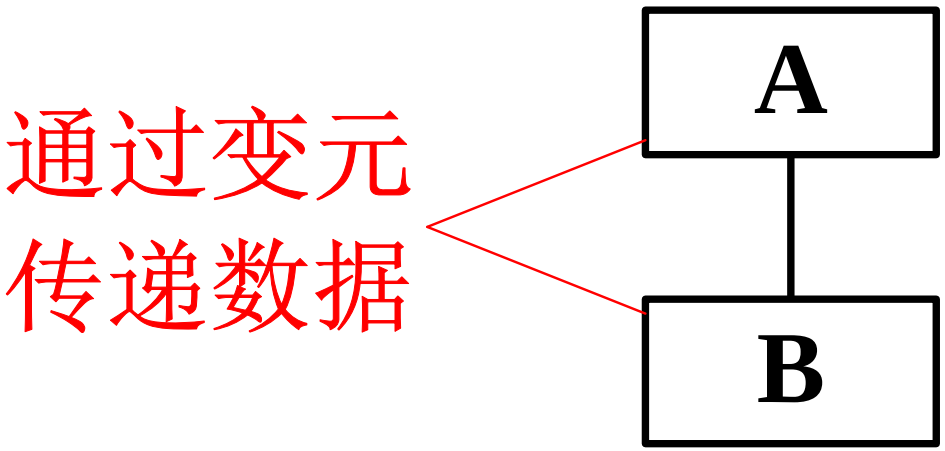
(1) 非直接耦合/完全独立(no direct coupling)

- 如果两个模块中的每一个都能独立地工作而不需要另一个模块的存在，那么它们完全独立。
- 在一个软件系统中不可能所有模块之间都没有任何连接。



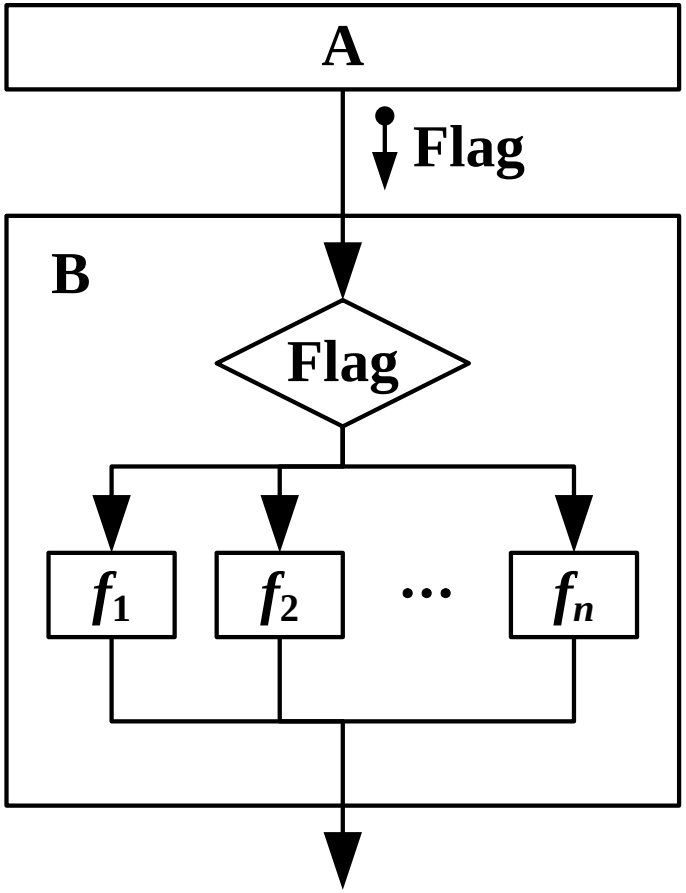
(2) 数据耦合(data coupling)

- 如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合称为数据耦合。



(3) 控制耦合(control coupling)

- 如果两个模块彼此间传递的信息中有控制信息，这种耦合称为控制耦合。



(4) 特征耦合(stamp coupling)

- 当把整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时，就出现了特征耦合。

评价：

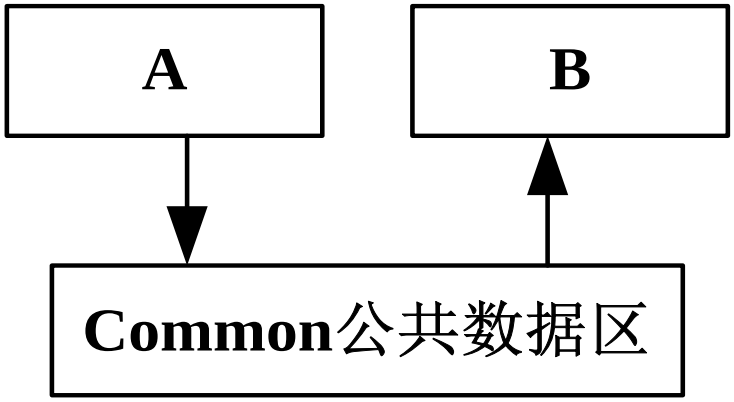
- 被调用的模块可使用的数据多于它确实需要的数据，这将导致对数据的访问失去控制，从而给计算机犯罪提供了机会。
- 无论何时把指针作为参数进行传递，都应该仔细检查该耦合。

(5) 公共环境耦合(common coupling)

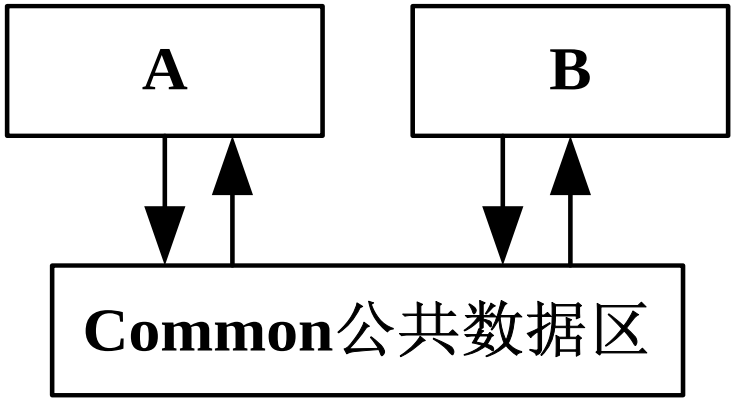
- 当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。公共环境可以是全程变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等等。

公共环境耦合的类型：

- 一个模块往公共环境送数据，另一个模块从公共环境取数据。数据耦合的一种形式，是比较松散的耦合。
- 两个模块都既往公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。



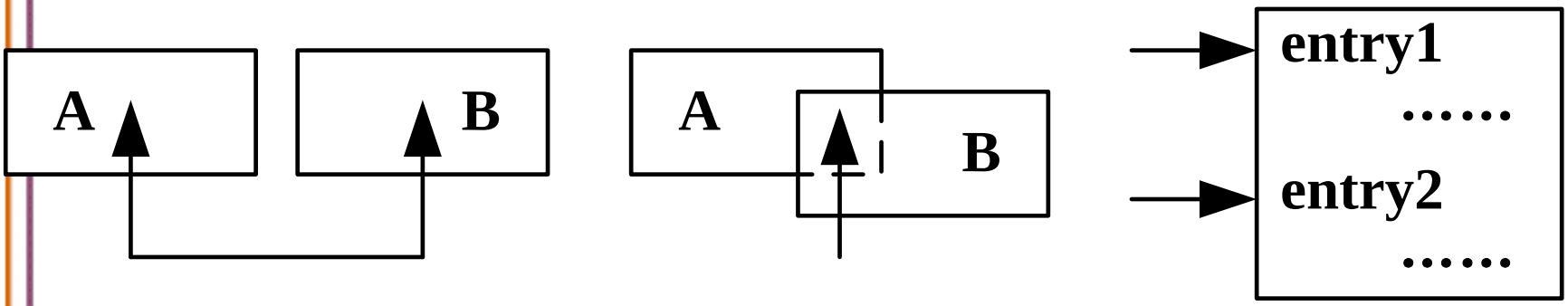
(a) 松散的公共环境耦合



(b) 紧密的公共环境耦合

(6) 内容耦合(content coupling)

- 最高程度的耦合是内容耦合。如果出现下列情况之一，两个模块间就发生了内容耦合：
  - 一个模块访问另一个模块的内部数据；
  - 一个模块不通过正常入口转到另一个模块的内部；
  - 两个模块有一部分程序代码重叠；
  - 一个模块有多个入口。



(a) 进入另一模块内部　(b) 模块代码重叠　(c) 多入口模块

- 耦合是影响软件复杂程度的一个重要因素。
- 应该采取下述设计原则：

  尽量使用数据耦合，

  少用控制耦合和特征耦合，

  限制公共环境耦合的范围，

  完全不用内容耦合。

2. 内聚

- 内聚：标志一个模块内各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然扩展。简单地说，理想内聚的模块只做一件事情。
- 要求：设计时应该力求做到高内聚，通常中等程度的内聚也是可以采用的，而且效果和高内聚相差不多；但是，低内聚不要使用。
- 内聚和耦合是密切相关的，模块内的高内聚往往意味着模块间的松耦合。实践表明内聚更重要，应该把更多注意力集中到提高模块的内聚程度上。

内聚程度的度量：

(1) 偶然内聚(coincidental cohesion)

- 如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。

(2) 逻辑内聚(logical cohesion)

- 如果一个模块完成的任务在逻辑上属于相同或相似的一类，则称为逻辑内聚。

(3) 时间内聚(temporal cohesion)

- 如果一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。

执行初始化
打开旧主文件、新主文件、事务文件和打印文件； 初始化销售地区表； 读第一条事务记录和第一条旧主文件记录；

(4) 过程内聚(procedural cohesion)

- 如果一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。
- 使用程序流程图作为工具设计软件时，常常通过研究流程图确定模块的划分，这样得到的往往是过程内聚的模块。

(5) 通信内聚(communicational cohesion)

- 如果模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚。即在同一个数据结构上操作。

(6) 顺序内聚(sequential cohesion)

- 如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。

(7) 功能内聚(functional cohesion)

- 如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

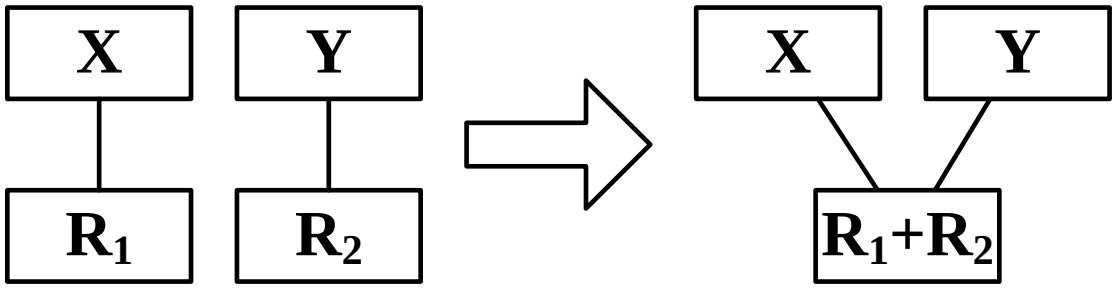
七种内聚的优劣评分结果：

- 高内聚：功能内聚　10分
  顺序内聚　9分
- 中内聚：通信内聚　7分
  过程内聚　5分
- 低内聚：时间内聚　3分
  逻辑内聚　1分
  偶然内聚　0分
- 设计时力争做到高内聚，并且能够辨认出低内聚的模块。

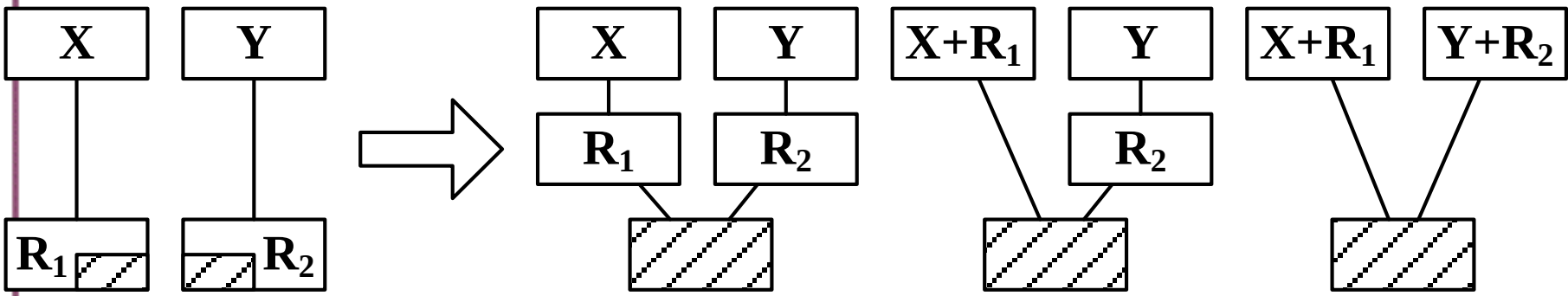
5.3 启发规则

1. 改进软件结构提高模块独立性

- 通过模块分解或合并，降低耦合提高内聚。
- 两个方面：
  - 模块功能完善化。一个完整的模块包含：
    - 执行规定的功能的部分
    - 出错处理的部分
    - 返回一个“结束标志”
  - 消除重复功能，改善软件结构。
    - 完全相似
    - 局部相似



完全相似　　完全合并



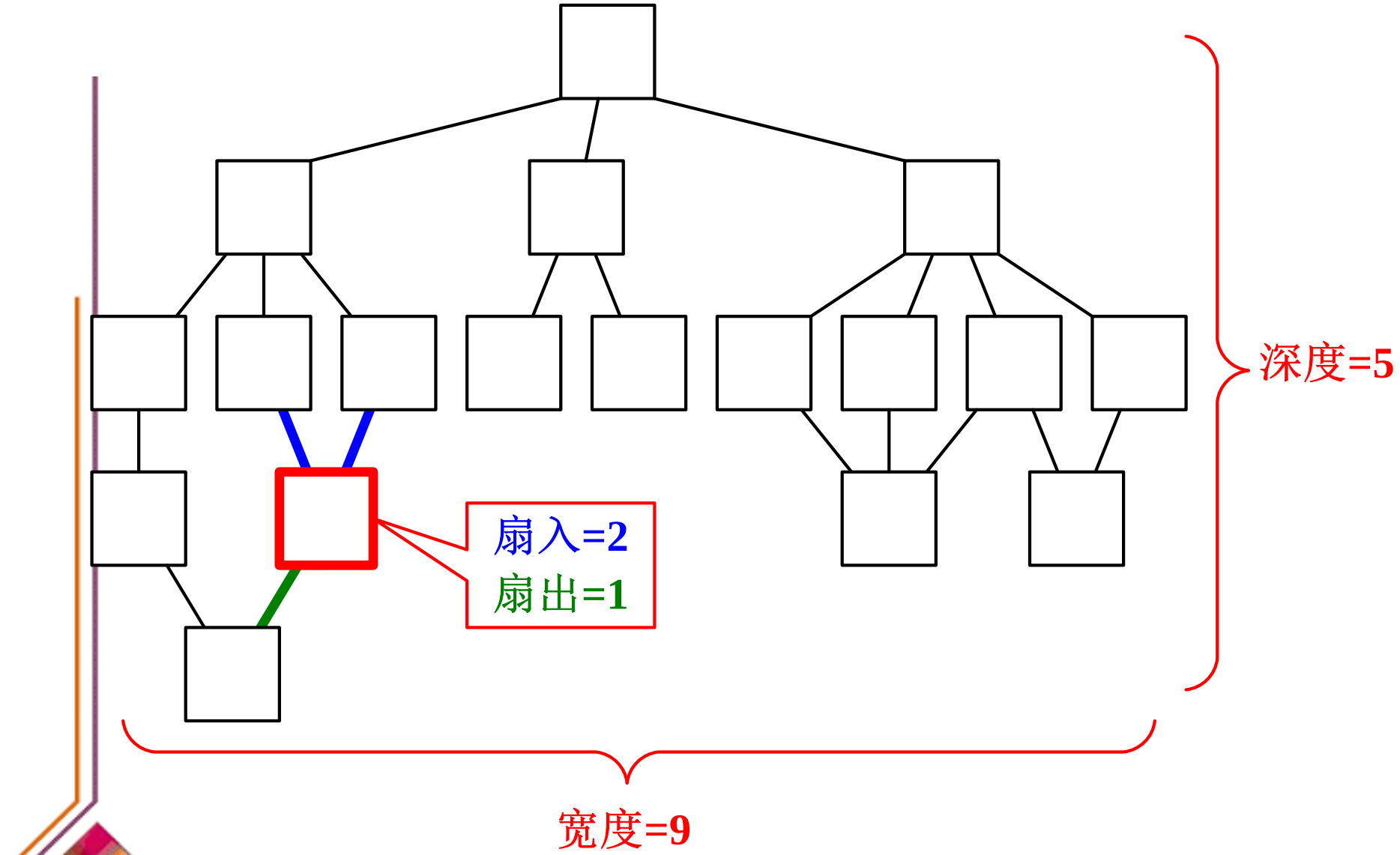
局部相似　　方案一　　方案二　　方案三

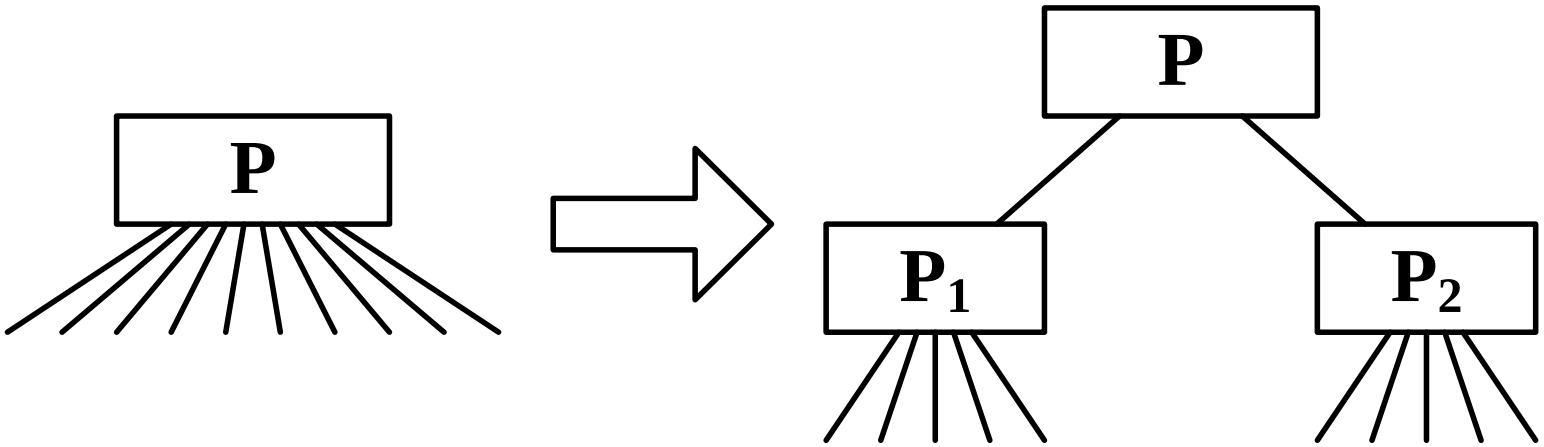
2. 模块规模应该适中

- 经验表明，一个模块的规模不应过大，最好能写在一页纸内。通常规定50~100行语句，最多不超过500行。
- 过大的模块往往是由于分解不充分，但是进一步分解必须符合问题结构，一般说来，分解后不应该降低模块独立性。
- 过小的模块开销大于有效操作，而且模块数目过多将使系统接口复杂。

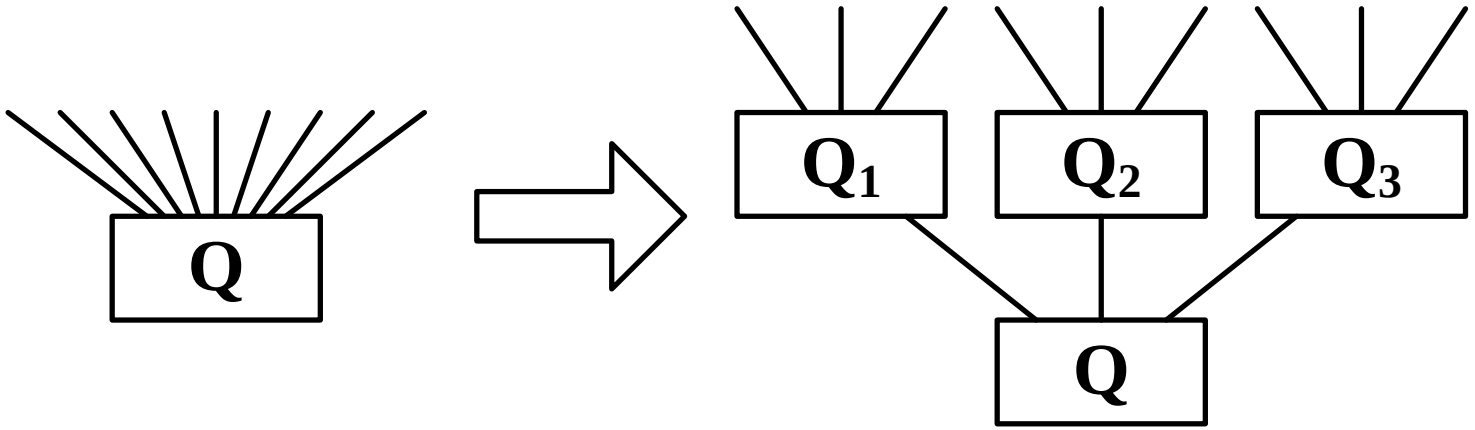
3. 深度、宽度、扇出和扇入都应适当

- 深度：软件结构中控制的层数，它往往能粗略地标志一个系统的大小和复杂程度。
- 宽度：软件结构内同一个层次上的模块总数的最大值。
- 扇出：一个模块直接控制(调用)的模块数目。
- 扇入：有多少个上级模块直接调用它。


深度=5
扇入=2
扇出=1
宽度=9



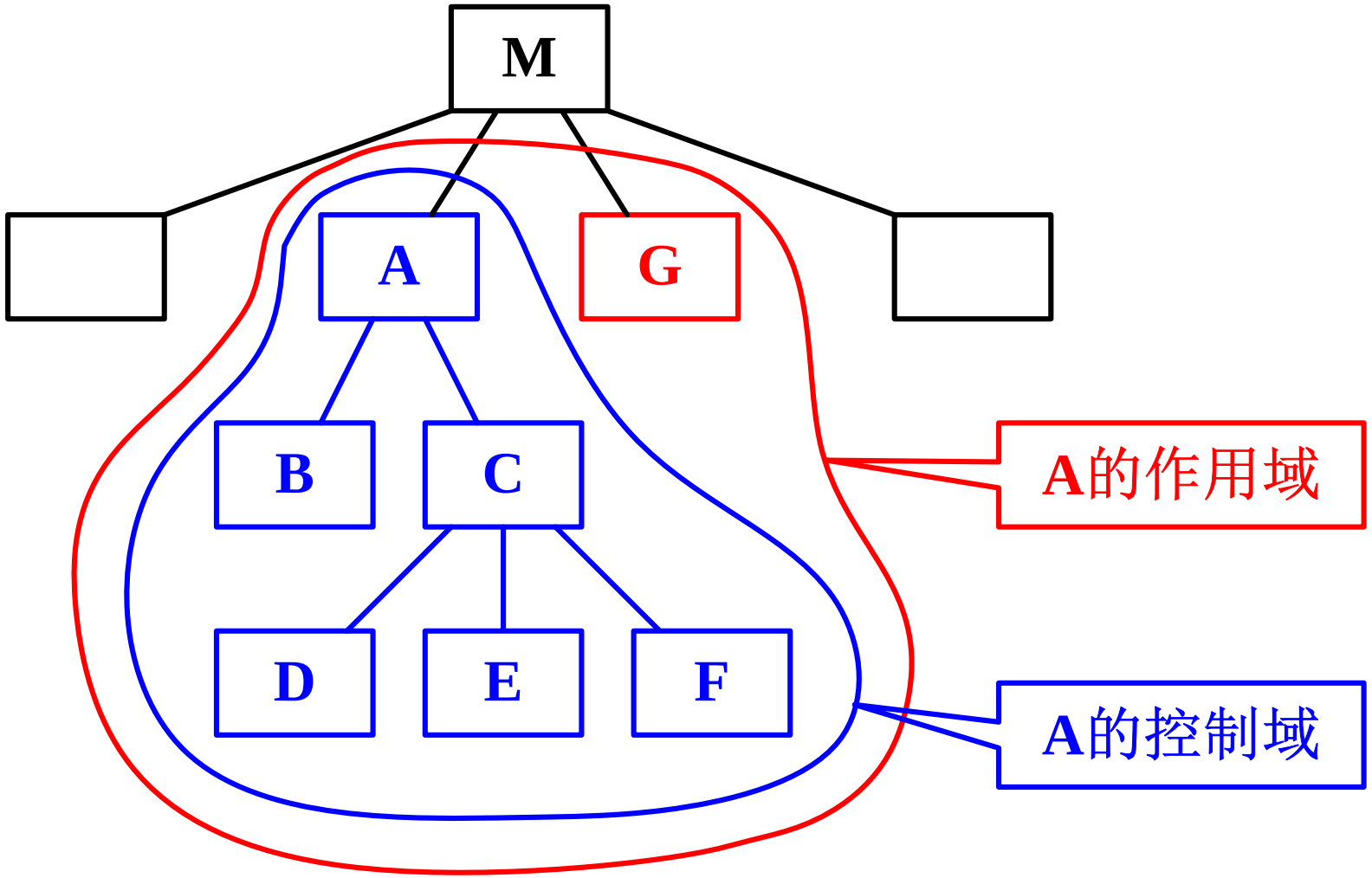
扇出过大



非公共模块扇入过大

4. 模块的作用域应该在控制域之内

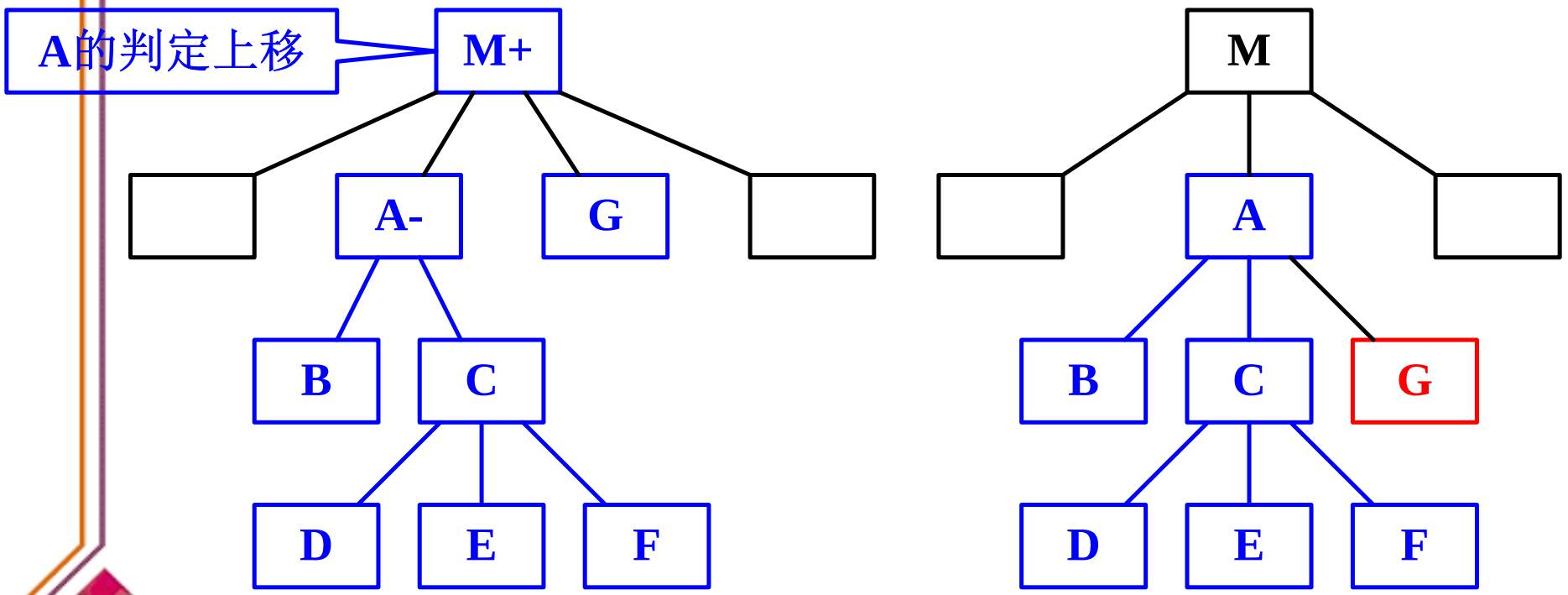
- **模块的作用域**：定义为受该模块内一个判定影响的所有模块的集合。
- **模块的控制域**：是这个模块本身以及所有直接或间接从属于它的模块的集合。
- 在一个设计得很好的系统中，所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块。



违反规则的情况

解决方案：

- 把模块A中的判定移到模块M中；
- 把模块G移到模块A下面，作为他的下级模块。



5.4 描绘软件结构的图形工具

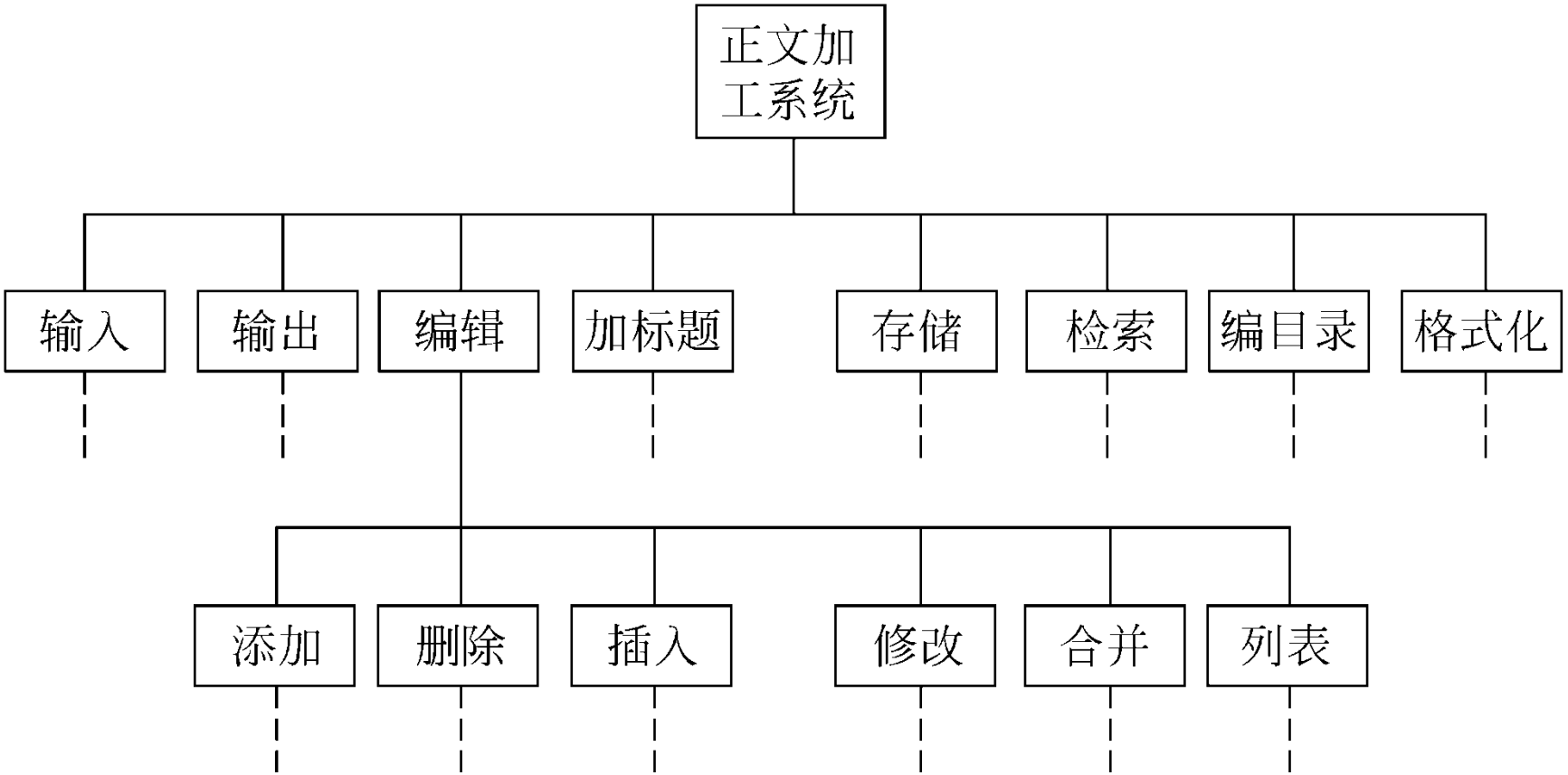
5.4.1 层次图和HIPO图

1. 层次图(H图)

- 层次图用来描绘软件的层次结构。很适于在自顶向下设计软件的过程中使用。

层次图和层次方框图的区别：

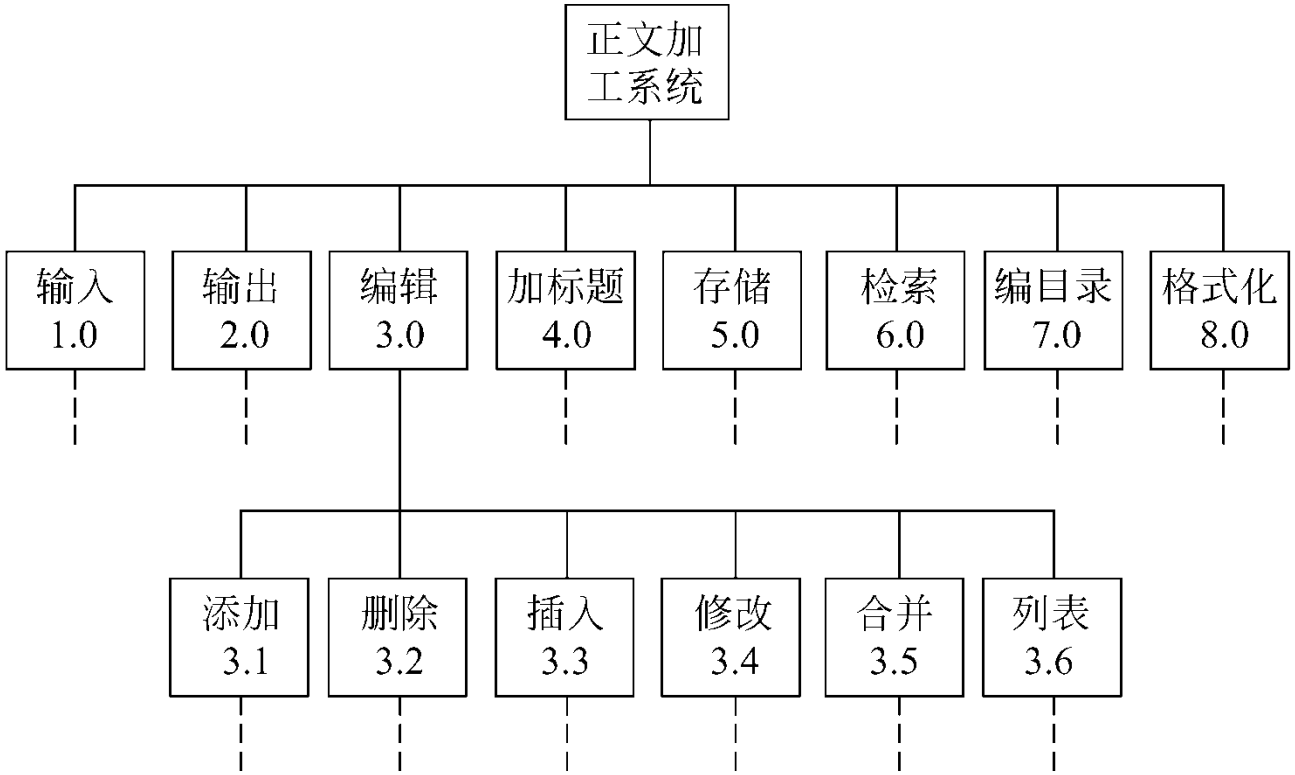
	层次图	层次方框图
作用	描绘软件结构	描绘数据结构
矩形框	模块	数据元素
连线	调用关系	组成关系



正文加工系统的层次图

2. HIPO图

- HIPO图是美国IBM公司发明的“层次图+输入/处理/输出图”的英文缩写。
- 为了能使HIPO图具有可追踪性，在H图(层次图)里除了最顶层的方框之外，每个方框都加了编号。



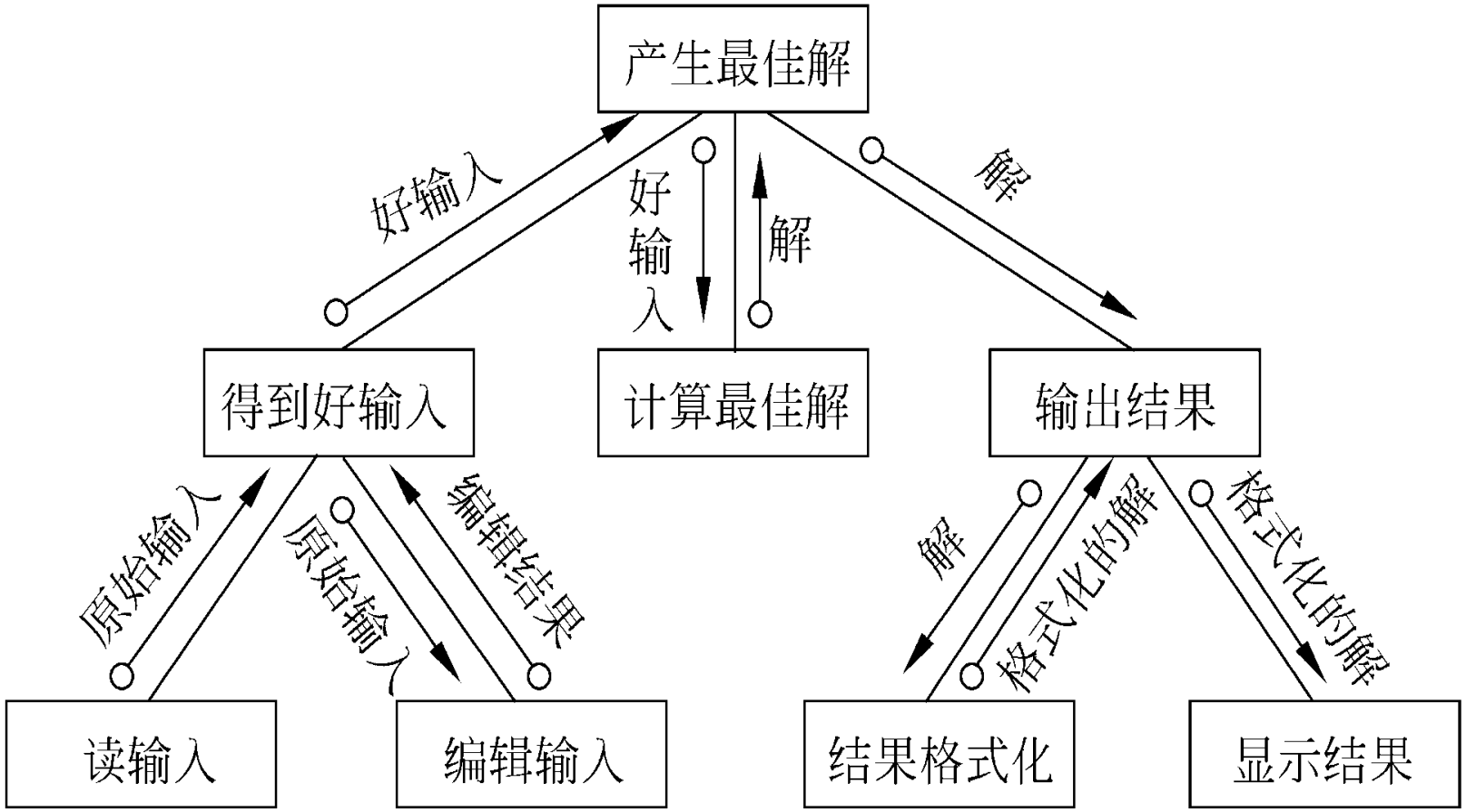
- 和H图中每个方框相对应，应该有一张IPO图描绘这个方框代表的模块的处理过程。模块在H图中的编号便于追踪了解这个模块在软件结构中的位置。

5.4.2 结构图

- Yourdon提出的结构图是进行软件结构设计的另一个有力工具。结构图和层次图类似，也是描绘软件结构的图形工具。

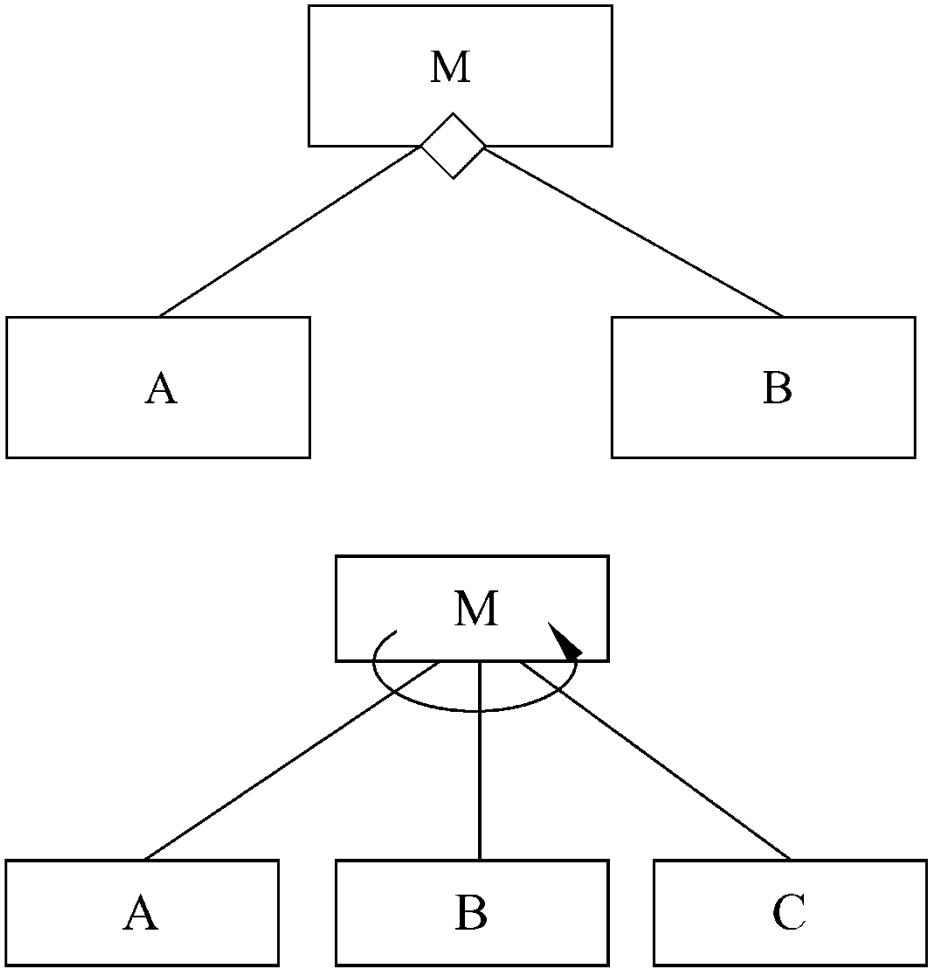
基本符号：

- 方框代表一个模块；
- 方框之间的直线表示模块的调用关系；
- 尾部是空心圆箭头表示传递的是数据；
- 尾部实心圆箭头表示传递的是控制信息。


产生最佳解
好输入
好输入
解
解
得到好输入
计算最佳解
输出结果
原始输入
原始输入
编辑结果
解
格式化的解
格式化的解
读输入
编辑输入
结果格式化
显示结果

附加符号：

- 选择调用：判定为真时调用A，为假时调用B。
- 循环调用：模块M循环调用模块A、B、C。

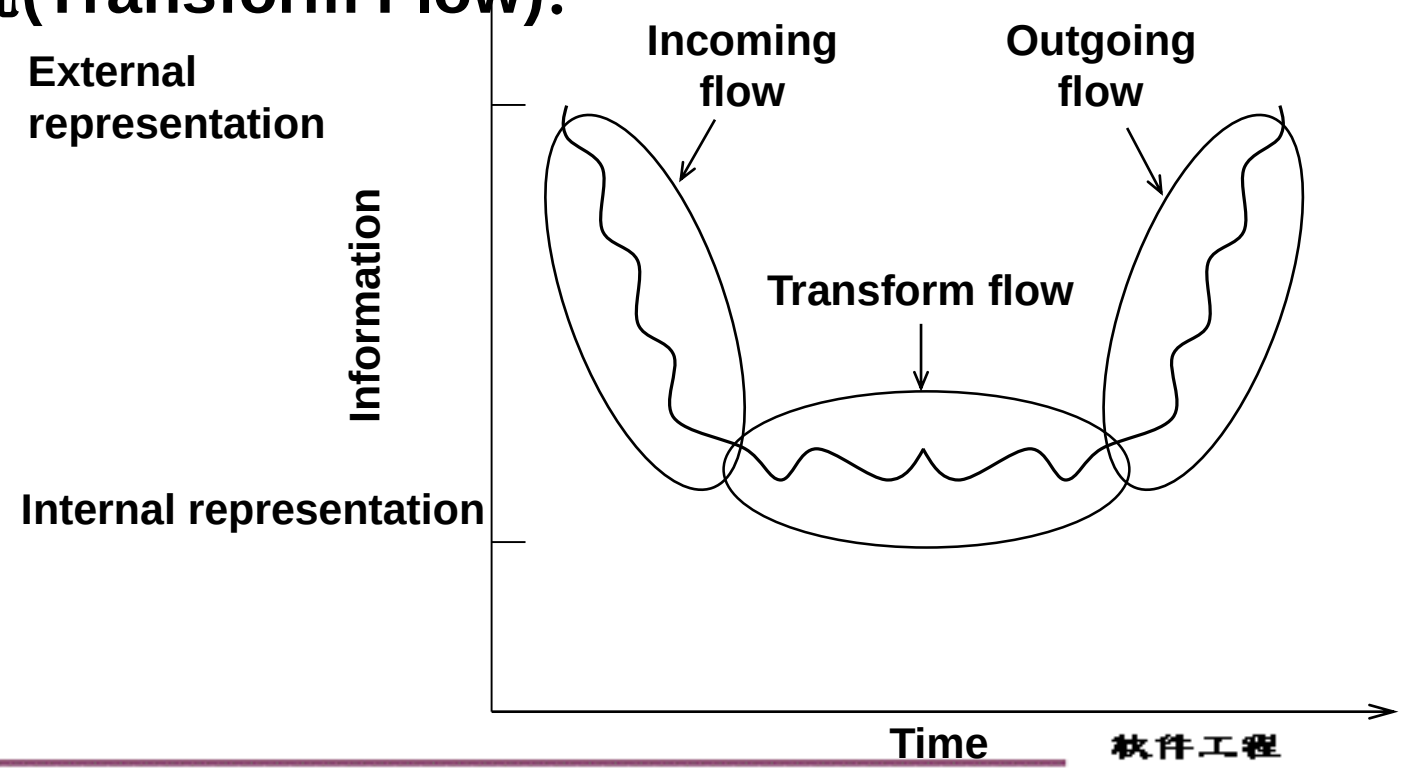


面向数据流的设计方法
（又称为SD：Structural Design）

基本思想：　DFD → System Hierarchy

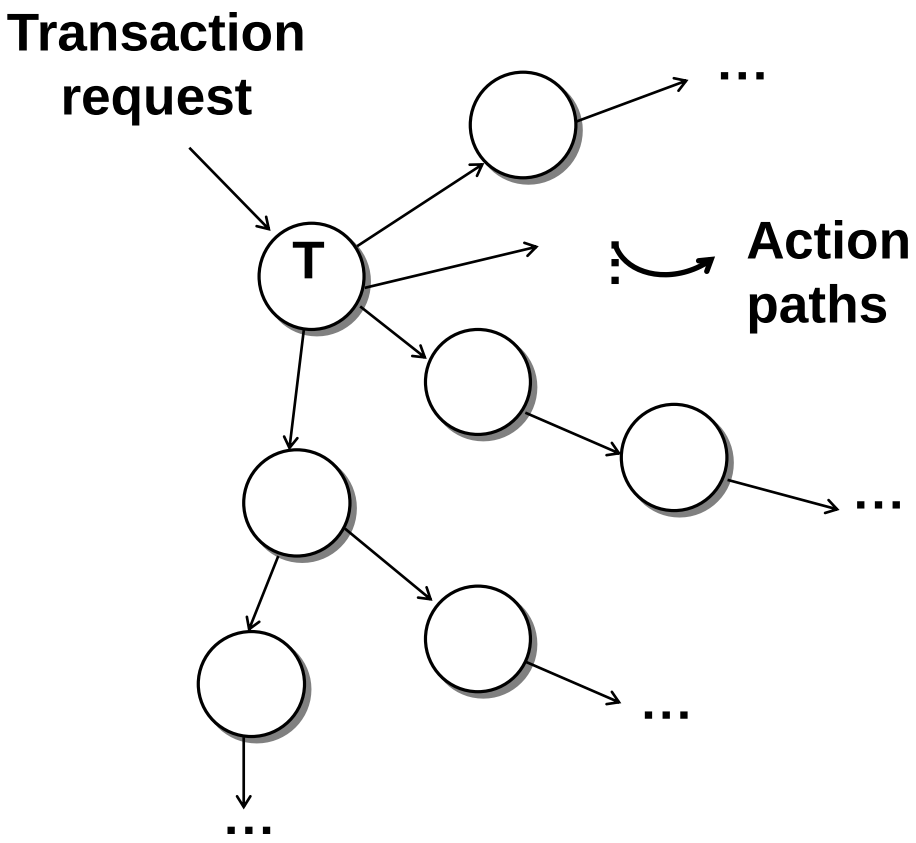
1、Data Flow 的分类

⑴ 变换流(Transform Flow)：



事实上所有信息流都可归结为变换流

(2) 事务流(Transaction Flow)



T = Call one of the several subroutines depending on the type of the incoming transaction request.

当信息流具有明显的“发射中心”时，可归结为事务流。

分析步骤

1. 复审基本系统模型（顶层图），以确保系统的I/O数据流符合规格说明要求
2. 复审和精化DFD图
3. 确定DFD图的类型（变换流/事务流）
4. 划定输入流和输出流的边界，孤立出变换中心
5. 将DFD图映射成相应的程序结构
6. 把DFD图中的每个加工映射成程序结构中的一个适当的模块
7. 采用启发式设计策略，精化所得到的程序结构，改良软件质量

3、分析设计

⑴ 变换分析

汽车数字仪表板的设计

功能：① 通过模 - 数转换实现传感器和微处理机接口；

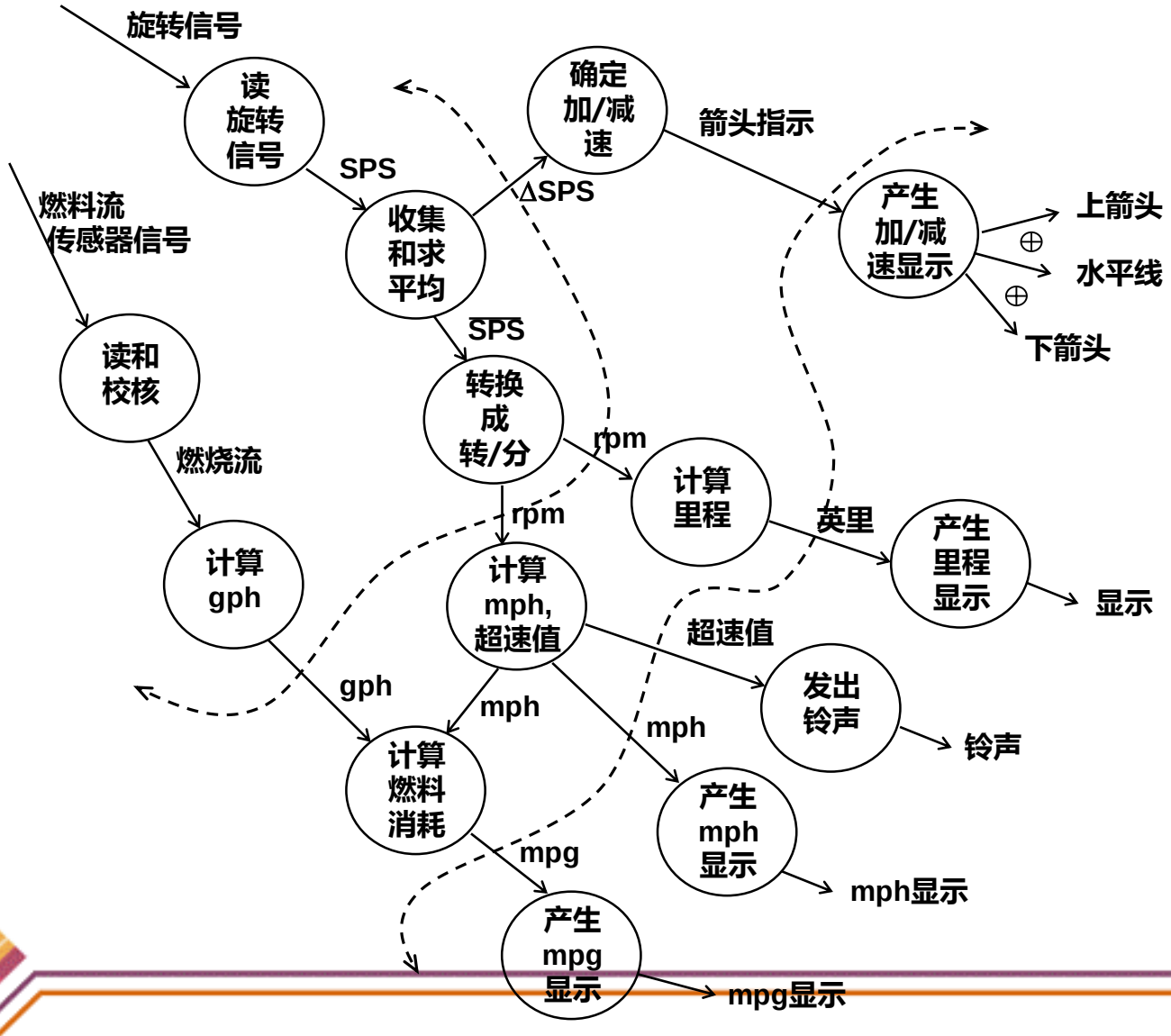
② 在发光二极管面板上显示数据；

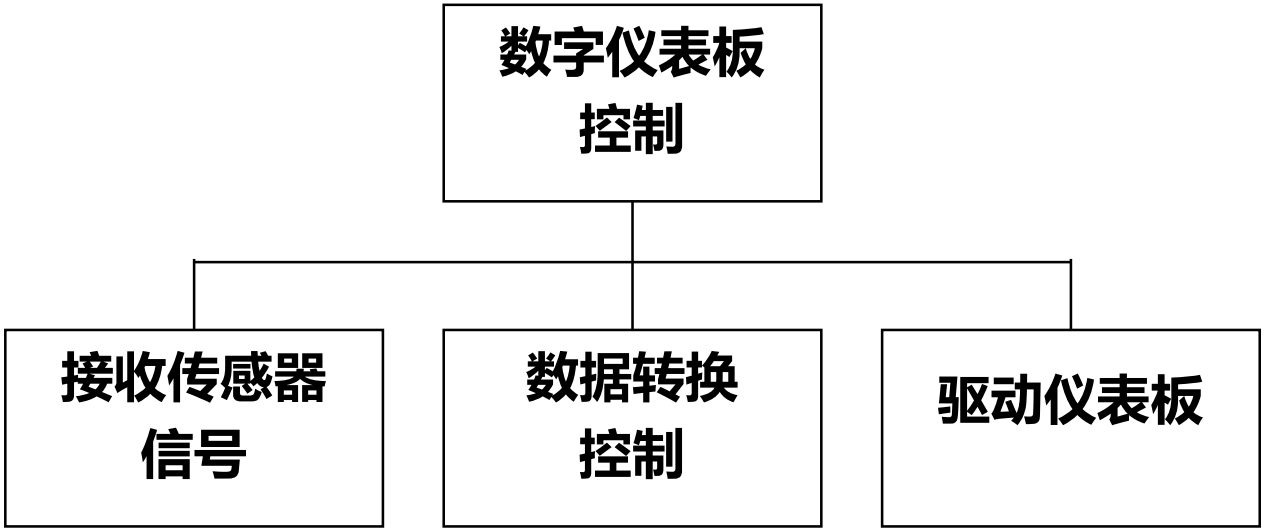
③ 指示每小时英里数(mph),行驶的里程，每加仑油行驶的英里数(mpg)等等；

④ 指示加速或减速；

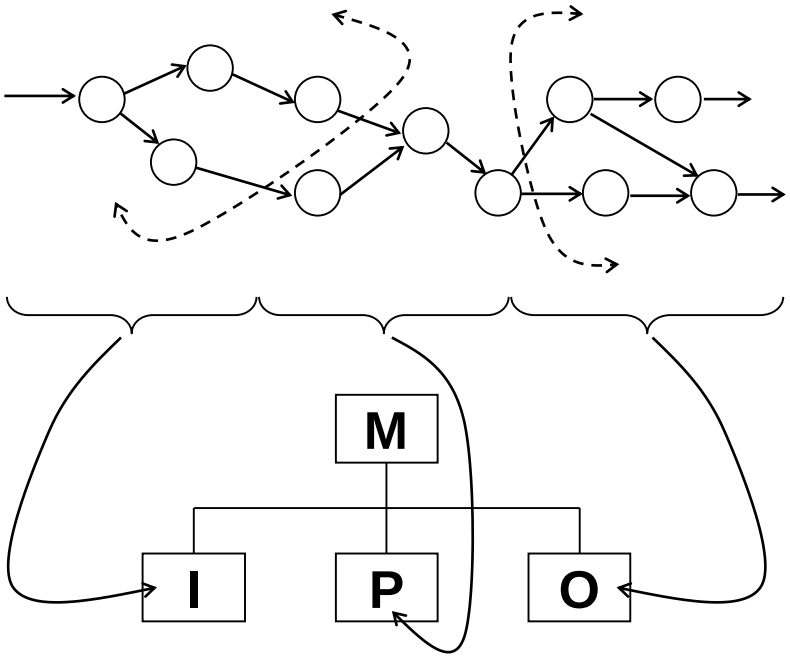
⑤ 如果车速超过55mph, 则发出警告铃声。

DFD的分界，先分出I、P、O三块(SPS, 信号/每秒)



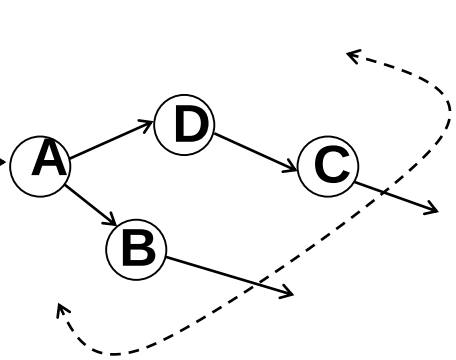


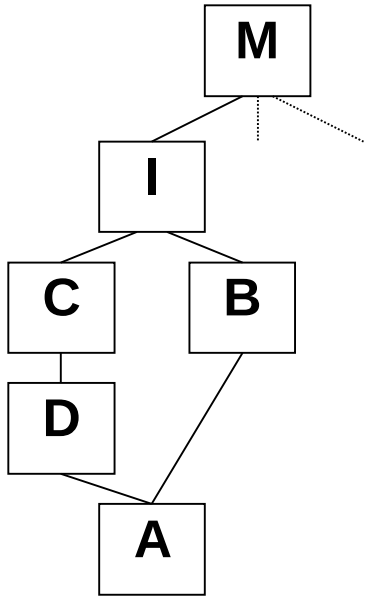
一般问题的一级分解方法：



第二步：映射

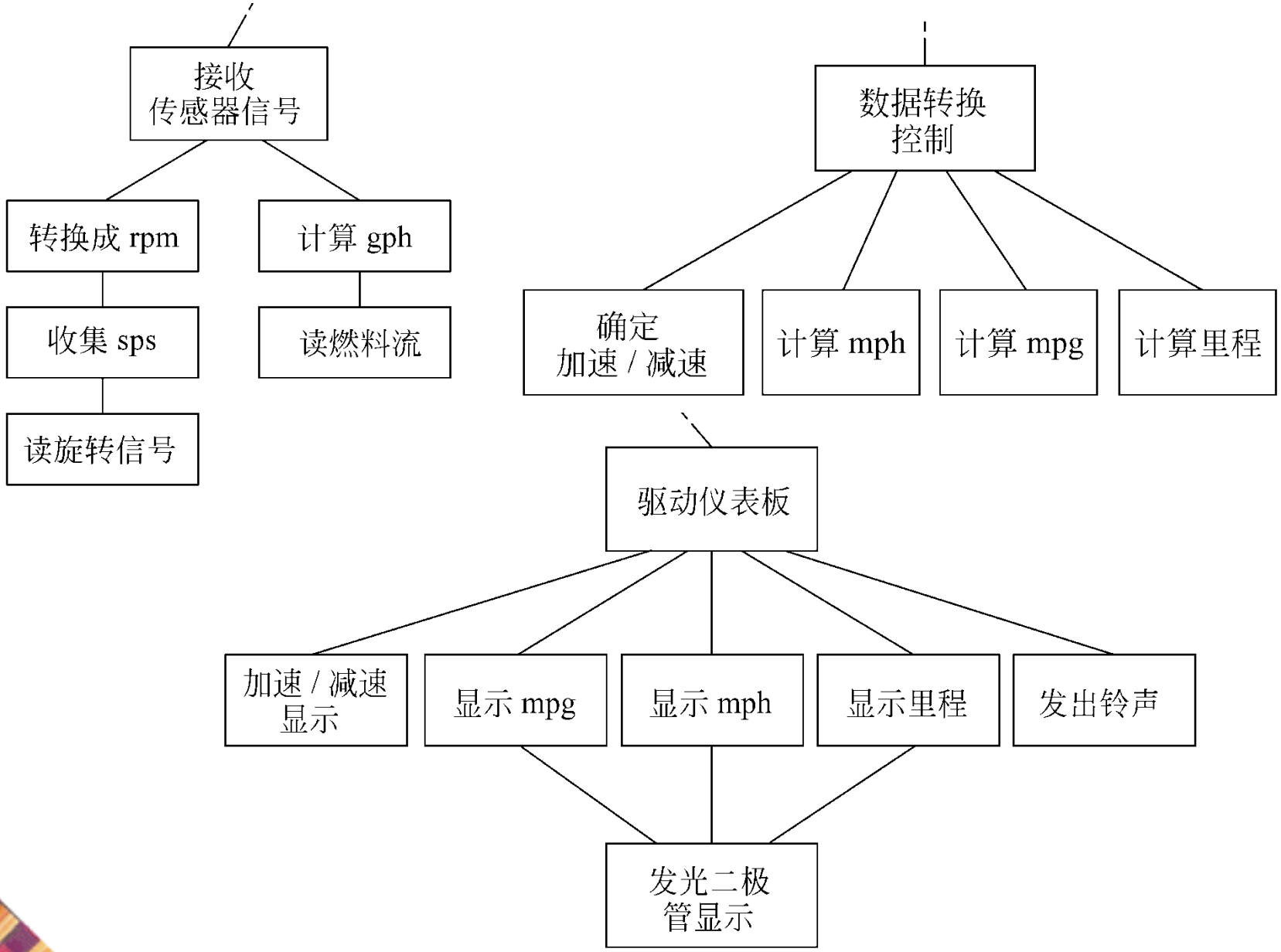
I：由边界向回溯，将每个遇到的处理器映成相应的层模块。





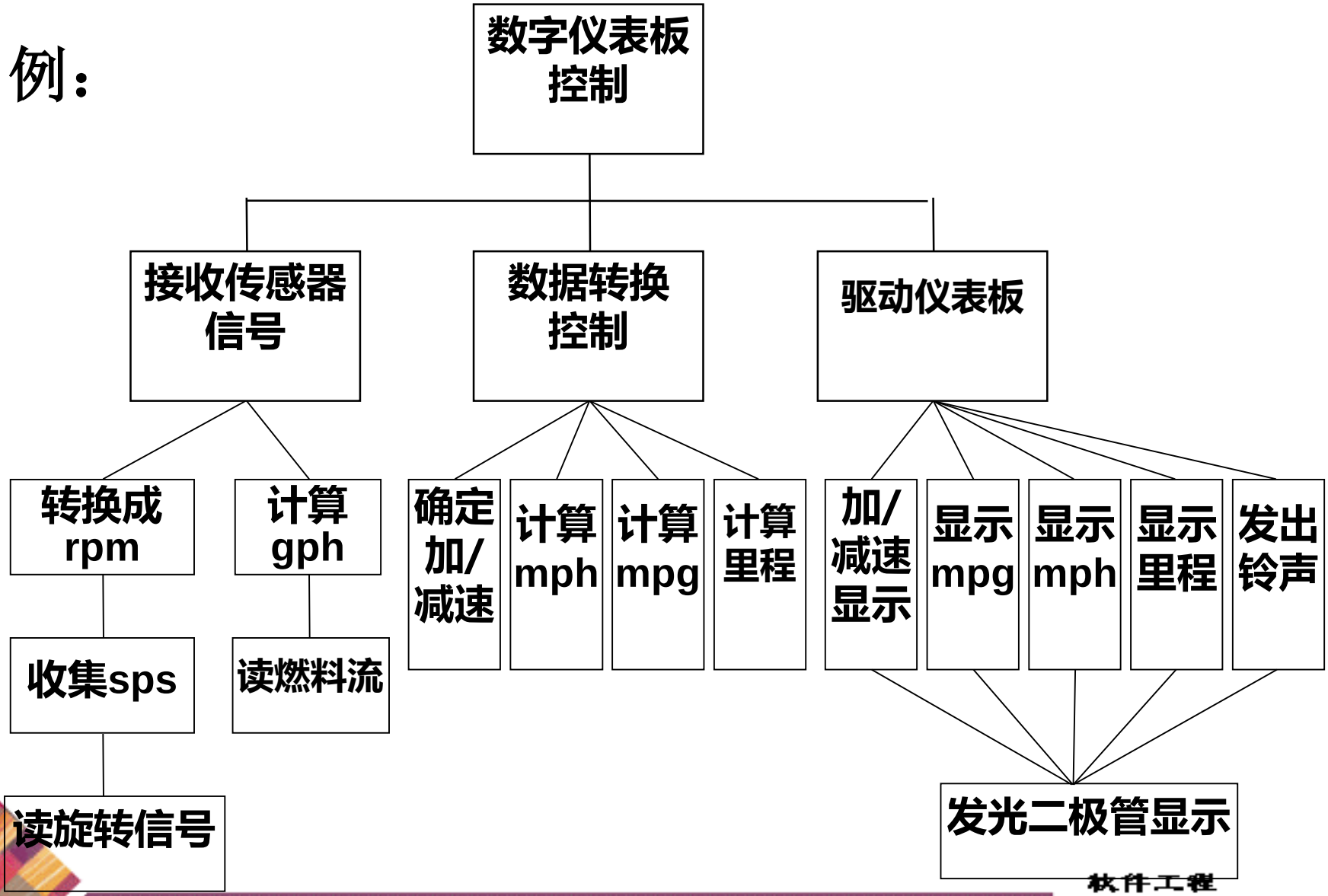
P：每个处理直接对应一个下层模块。

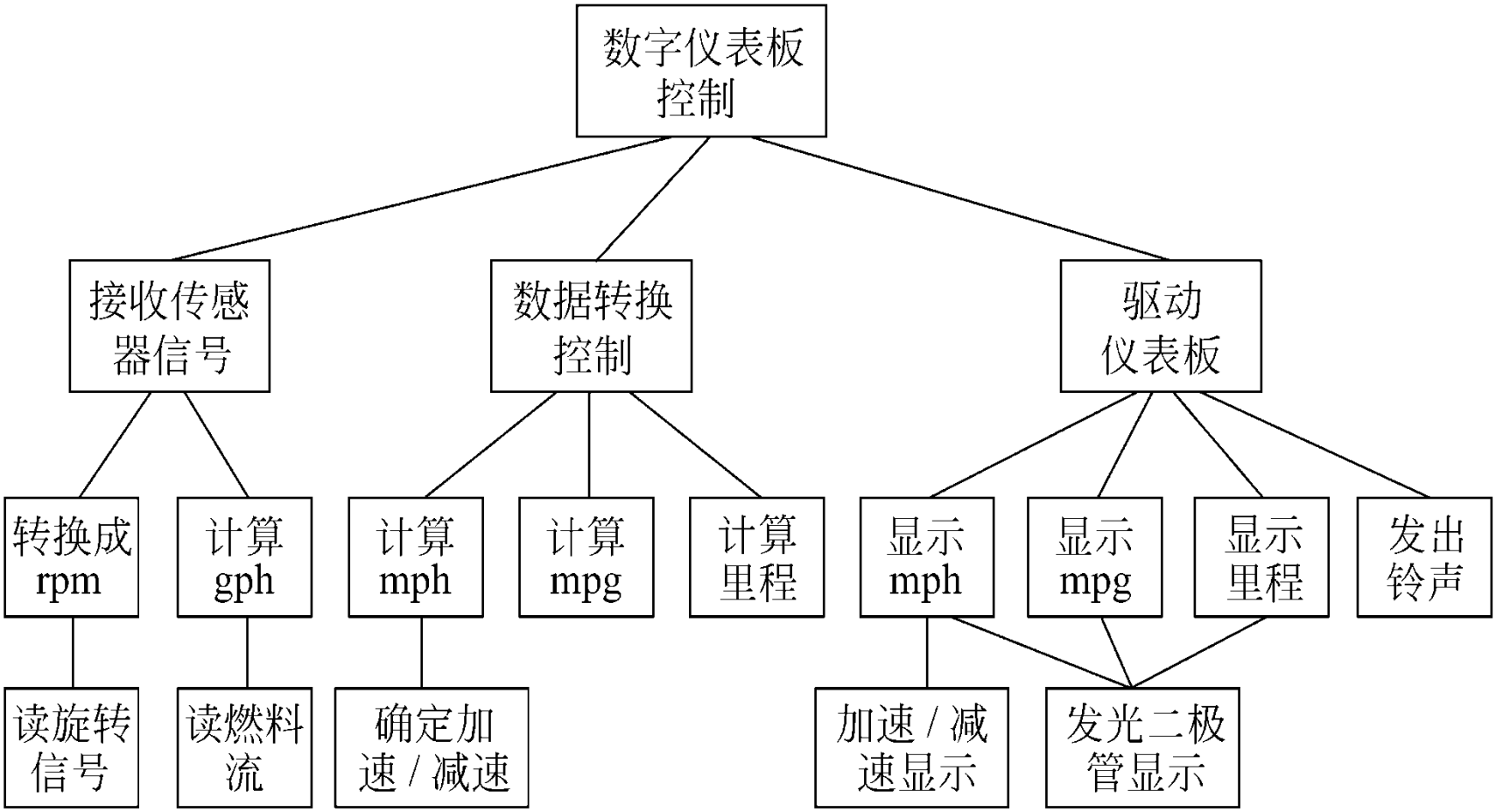
O：由边界向外推，方法与 I 类似


接收
传感器信号
转换成 rpm
计算 gph
收集 sps
读燃料流
读旋转信号
数据转换
控制
确定
加速 / 减速
计算 mph
计算 mpg
计算里程
驱动仪表板
加速 / 减速
显示
显示 mpg
显示 mph
显示里程
发出铃声
发光二极
管显示

第二级分解

例：


数字仪表板控制
接收传感器信号
数据转换控制
驱动仪表板
转换成rpm
计算gph
确定加/减速
计算mph
计算mpg
计算里程
加/减速显示
显示mpg
显示mph
显示里程
发出铃声
收集sps
读燃料流
读旋转信号
发光二极管显示


数字仪表板控制
接收传感器信号
数据转换控制
驱动仪表板
转换成rpm
计算gph
计算mph
计算mpg
计算里程
显示mph
显示mpg
显示里程
发出铃声
读旋转信号
读燃料流
确定加速/减速
加速/减速显示
发光二极管显示

精化

第三步：修改 —— 本着高内聚、低耦合的原则。

注：每个模块应附一简要说明描述

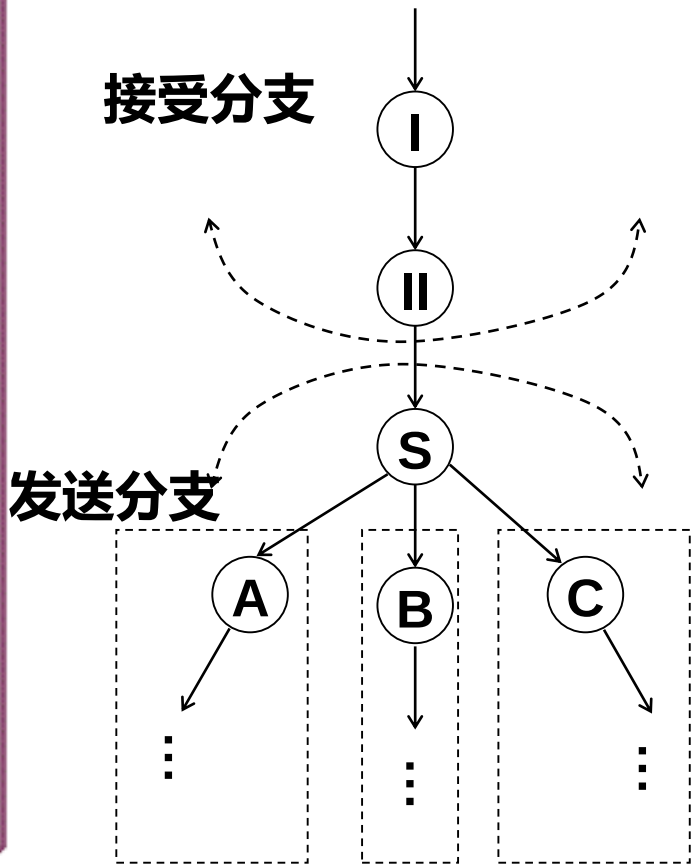
① 进出该模块的信息（接口描述）；

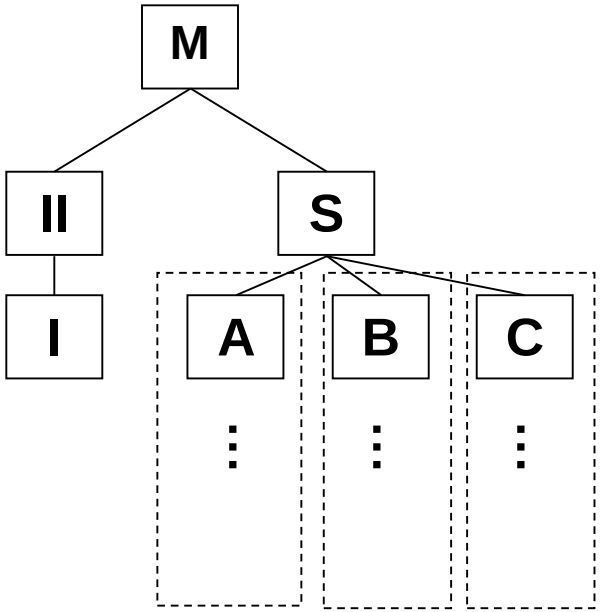
② 模块内部的信息；

③ 过程陈述，包括主要判定点及任务等；

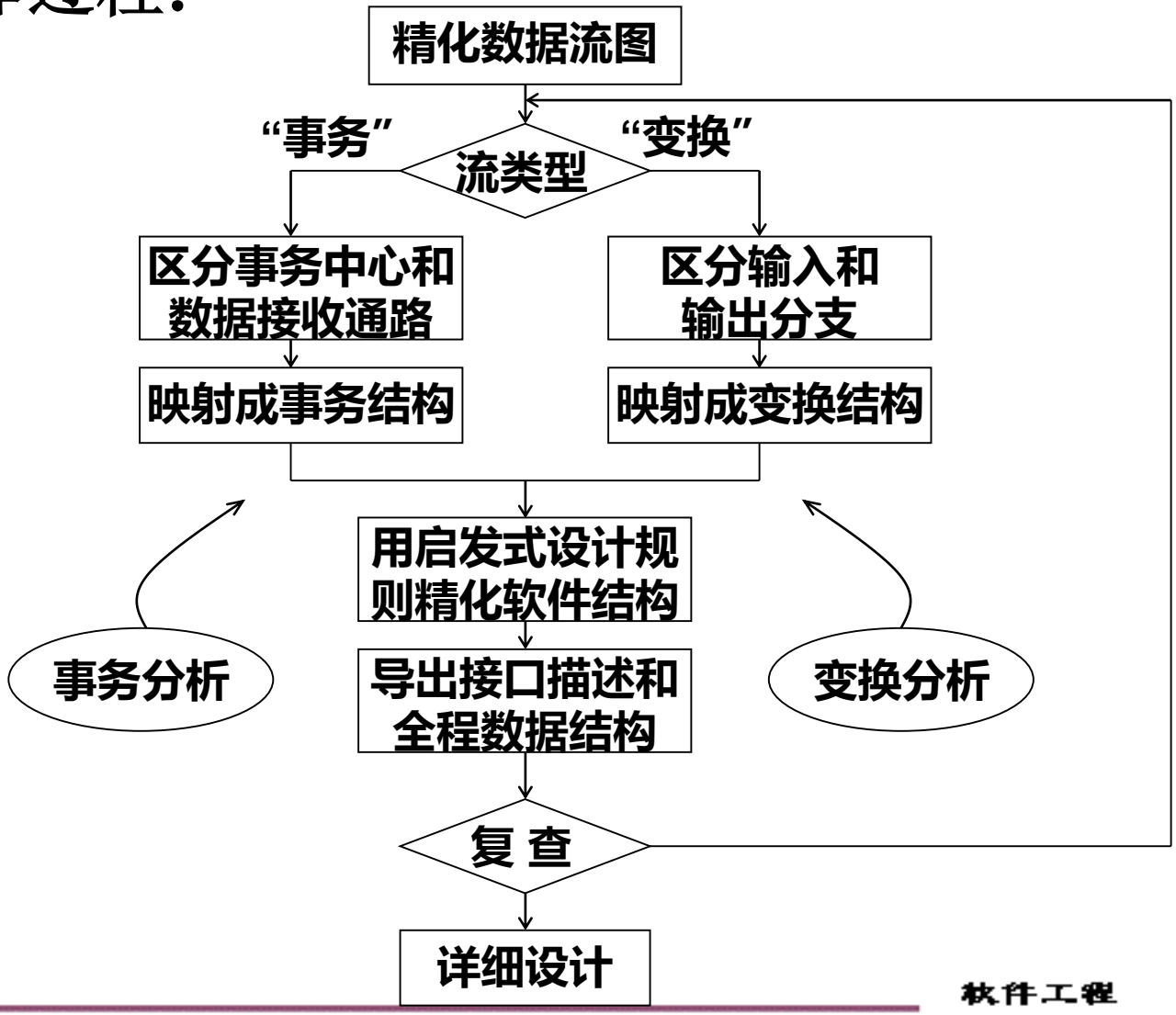
④ 对约束和特殊特点的简短讨论。

(2) 事务分析





3、SD的总体过程：


精化数据流图
“事务”
“变换”
流类型
区分事务中心和
数据接收通路
区分输入和
输出分支
映射成事务结构
映射成变换结构
用启发式设计规
则精化软件结构
事务分析
导出接口描述和
全程数据结构
变换分析
复 查
详细设计

小结

1、设计原理

2、启发规则

3、描绘软件结构的图形工具

4、面向数据流的设计方法