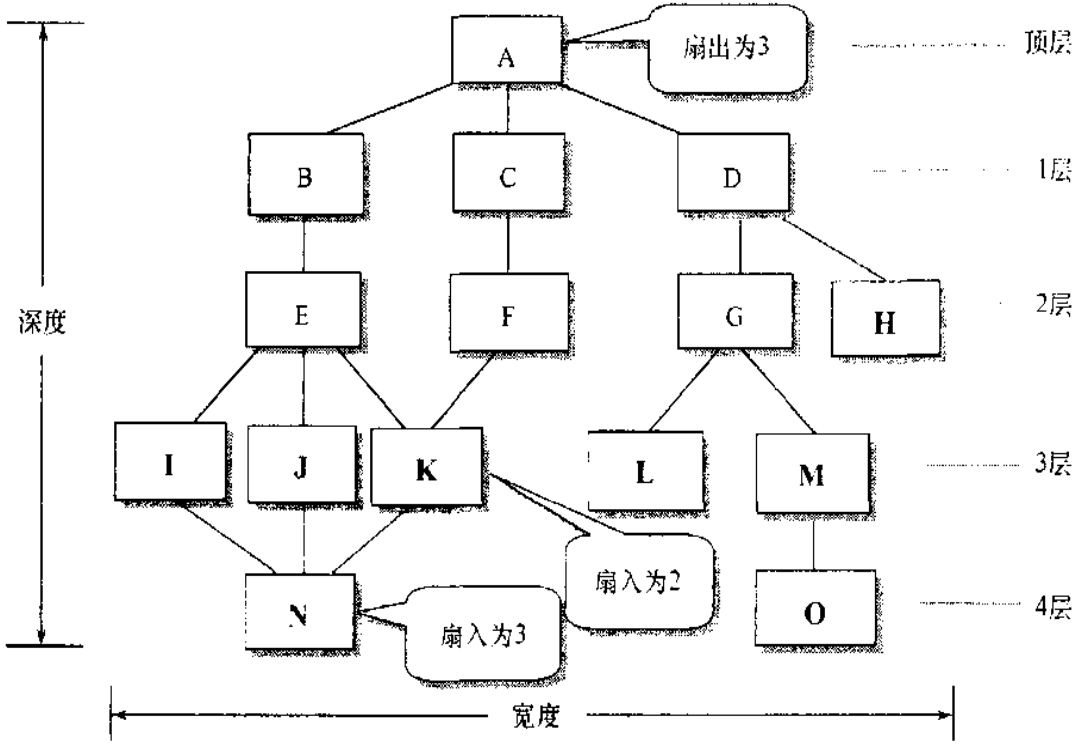
第8讲 软件结构和结构化设计

网络空间安全学院

芦效峰

1. 软件结构

- 1.1软件结构就是构成软件的模块的组织方式。模块之间可以有各种关系。一般可表示为层次结构和网状结构两种。



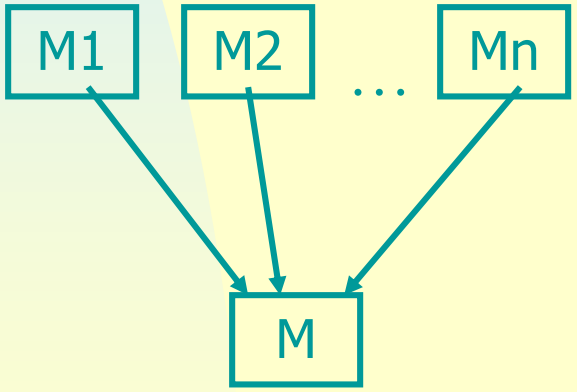
衡量模块层次结构的有关指标

深度—软件结构中控制的层数，粗略标志系统的大小和复杂程度；

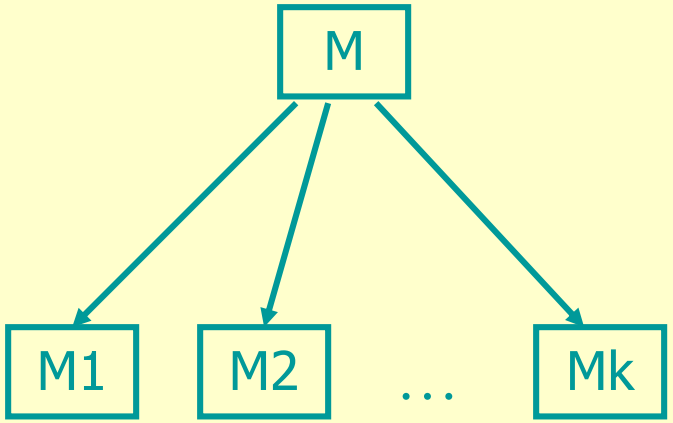
宽度—软件结构内同一层次上的模块总数的最大值；

扇出—一个模块直接控制的模块数目；典型系统的平均扇出为3或4。

扇入—有多少个上层模块直接调用它；



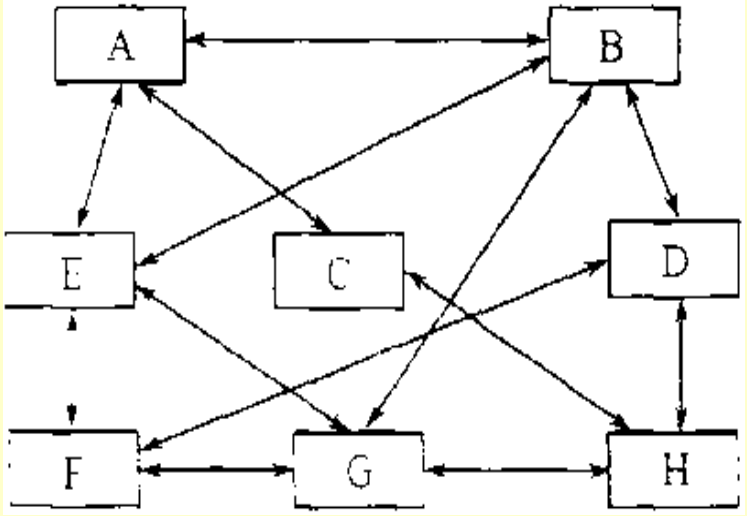
模块的扇入



模块的扇出

（2）模块的网状结构

- 在网状结构中，任何两个模块间都可以有双向的关系。由于不存在上级模块和下属模块的关系，也就分不出层次来。



- 对于不加限制的网状结构，由于模块间相互关系的任意性，使得整个结构十分复杂，处理起来势必引起许多麻烦，这与原来划分模块为便于处理的意图相矛盾。

1.2 模块设计优化

1） 改进软件结构提高模块独立性

通过模块的分解和合并，力求降低耦合提高内聚。

2） 模块规模应该适中

一个模块的规模不应过大，最好能写在一页纸内。不要超过60行。

过大的模块往往分解不充分，但进一步分解不应该降低模块独立性。

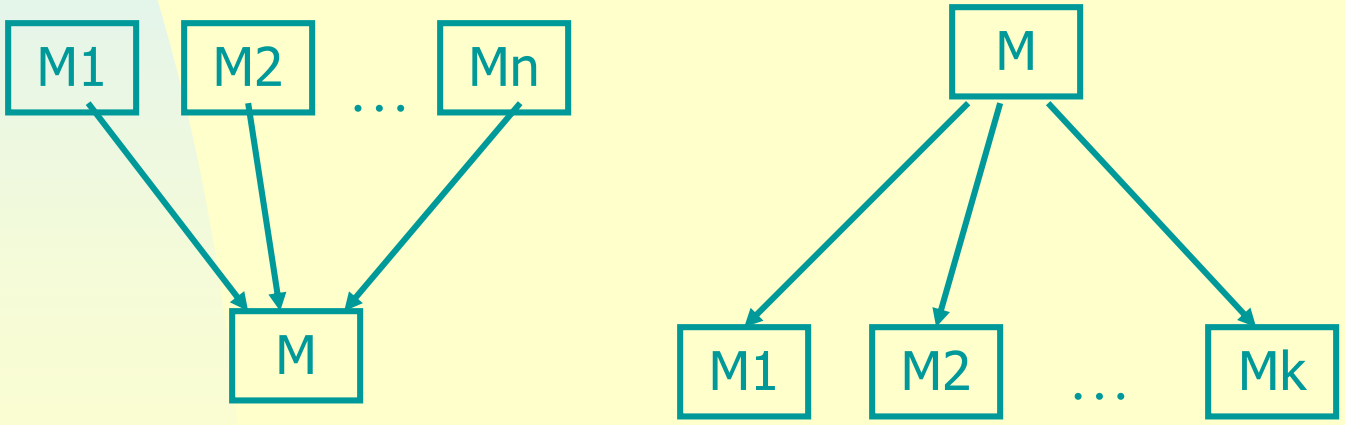
过小的模块开销大于有效操作，模块数目过多将使系统接口复杂。

防止模块功能过分局限

- 功能单一的模块具有高内聚。
- 但如任意限制局部数据结构的大小，过分限制控制流中可做的选择或外部接口的模式，模块功能就过分局限，使用范围过分狭窄，缺乏灵活性和可扩充性。

3） 深度、宽度、扇出和扇入都应适当

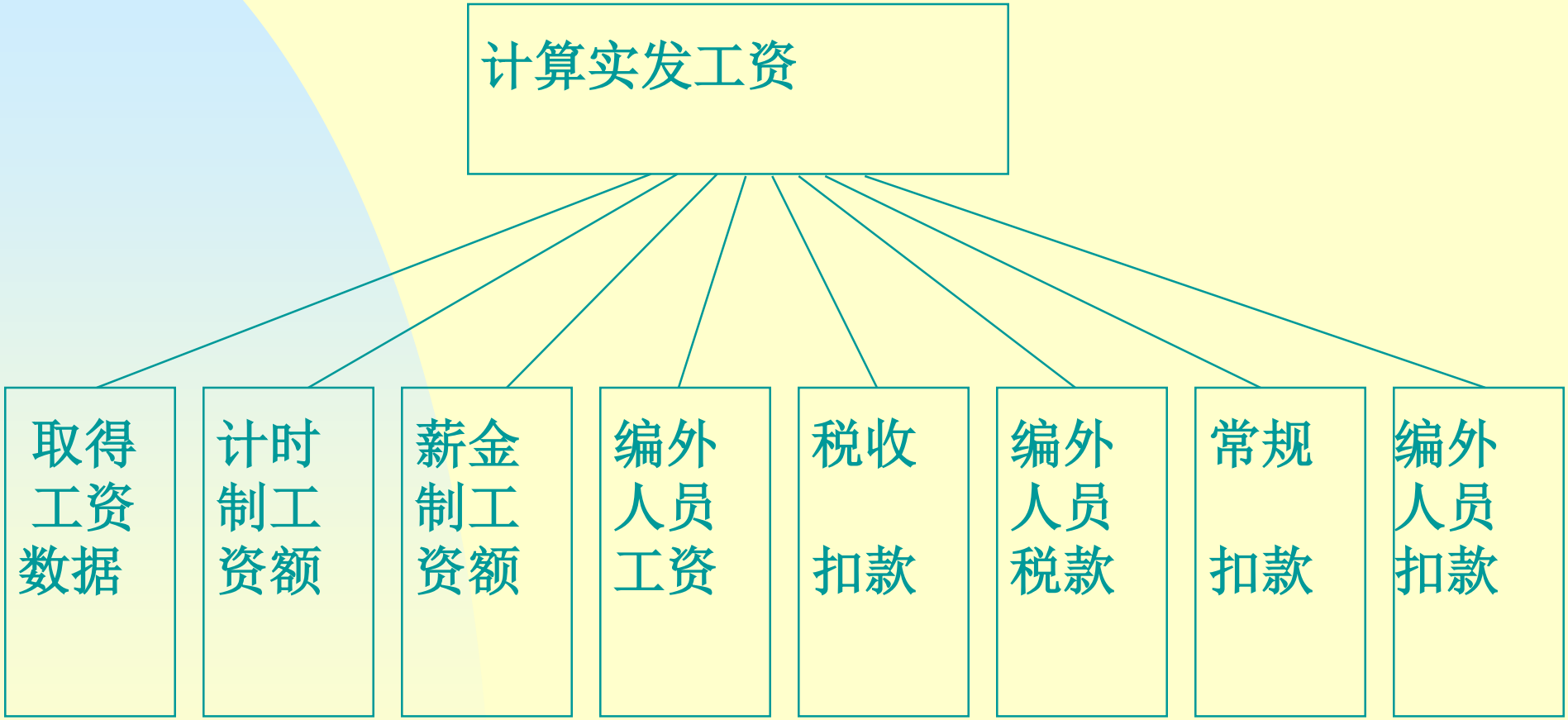
- 宽度数量应控制在7±2 (普遍认为的人类"智力"限度值)。
- 一般来讲，平均扇出数是3~5



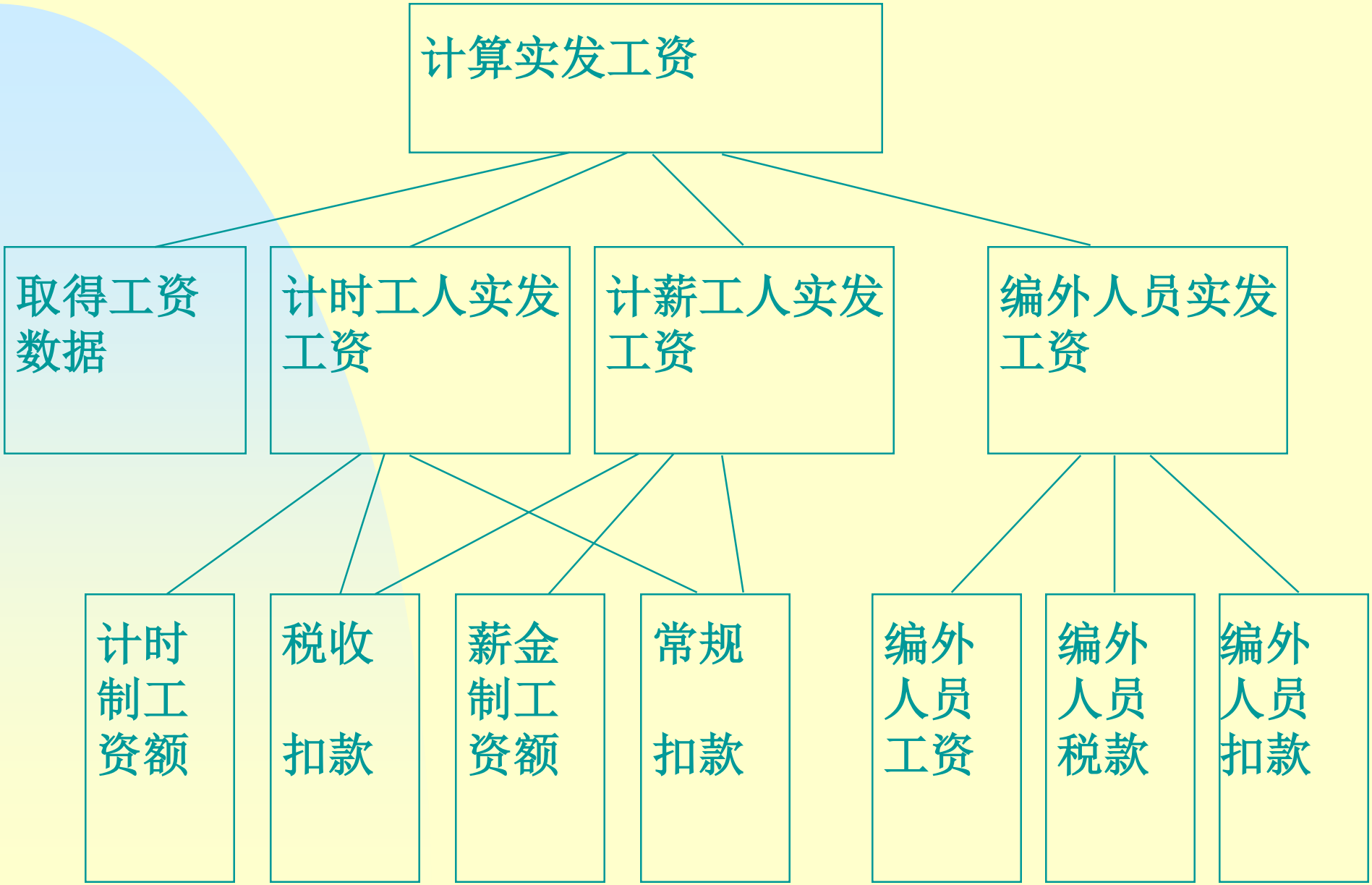
模块的扇入　　模块的扇出

原则：减少高扇出，争取高扇入

高扇出的模块结构举例：



增加中间层降低扇出


计算实发工资
取得工资数据
计时工人实发工资
计薪工人实发工资
编外人员实发工资
计时制工资额
税收扣款
薪金制工资额
常规扣款
编外人员工资
编外人员税款
编外人员扣款

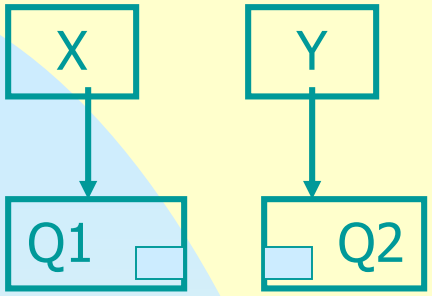


图1

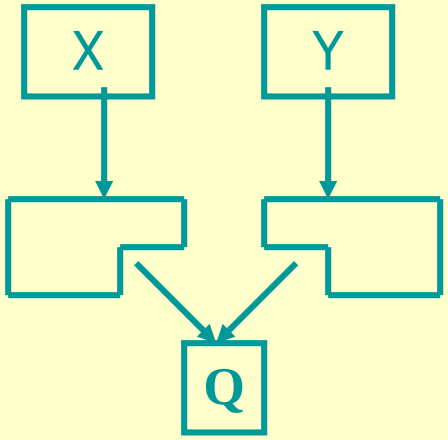


图2

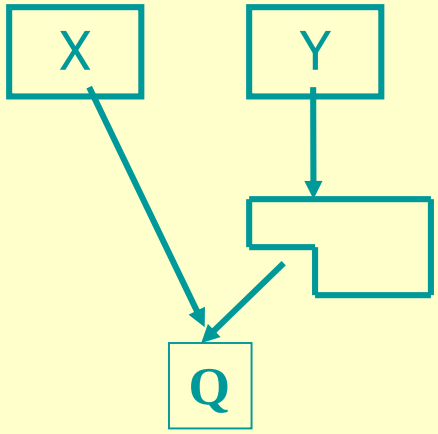


图3

图1，Q2中与已有的Q1相似；

图2，把Q1和Q2中相同的部分Q分离出来；

图3，若Q’1很小，可并入X中；

图4，若Q’2也很小，也可并入Y中；

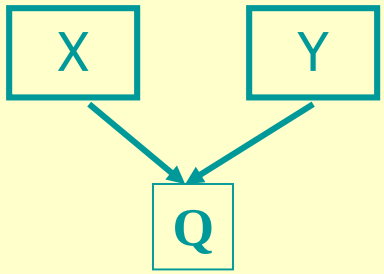


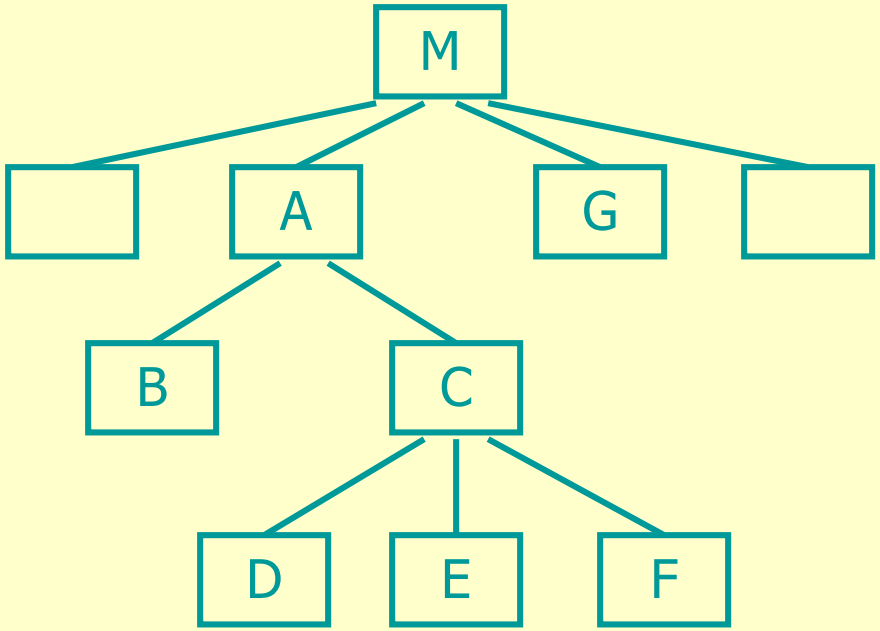
图4

分解模块

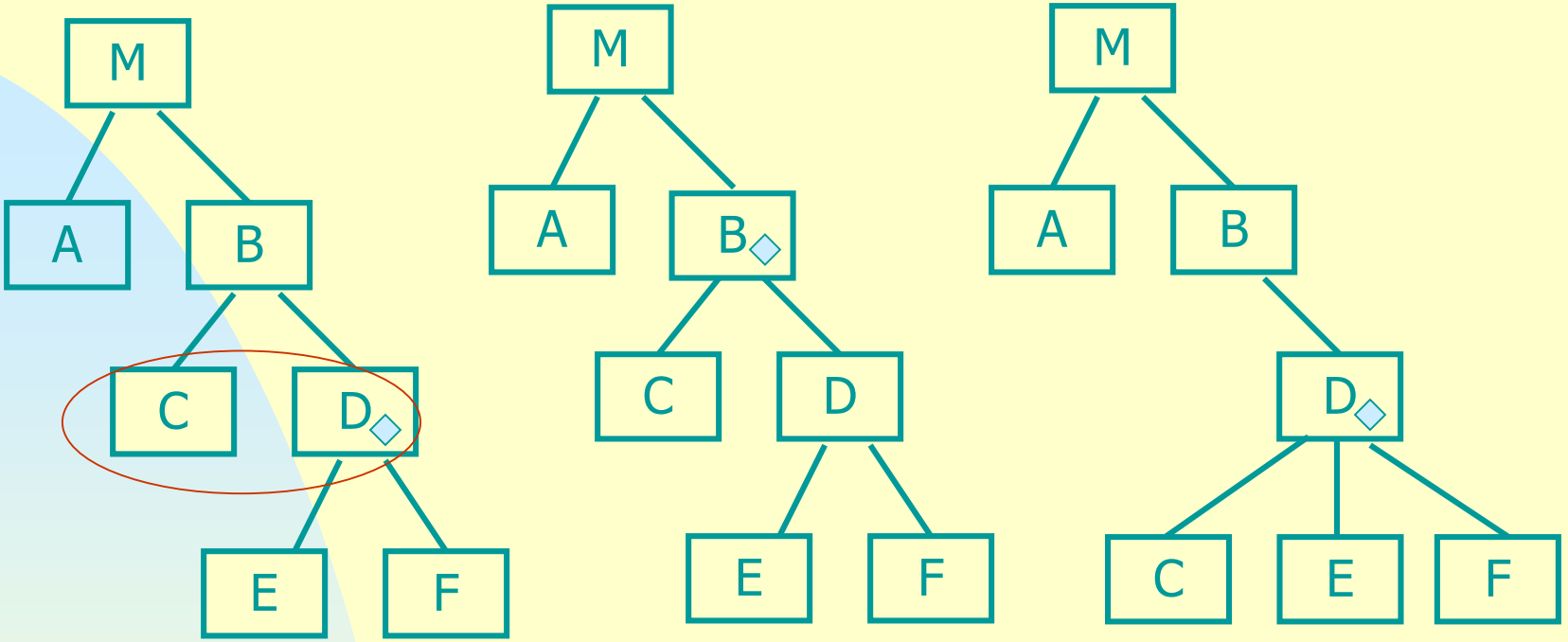
4） 模块的作用域应该在控制域之内

模块的作用域——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——模块本身以及所有直接或间接从属于它的模块的集合。



模块的作用域和控制域



假设C受D中判断的影响

在设计过程中，发现模块的作用范围不在其控制范围之内，可以用以下方法改进：

①上移判定点；　②下移受判定影响的模块

5） 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重要原因。

如：求一元二次方程 的根的模块：

root（tbl，x）

说明： 数组tbl是方程的系数，数组x回送根；

但是，以数组tbl作为参数增加了接口复杂度！

root（a,，b，c，r1，r2）

说明： a,，b，c是方程的系数，

r1，r2是计算的两个根；

6）设计单入口单出口的模块

强调模块间不要出现内容耦合，即病态连接。

7） 模块功能应该可以预测

把模块当作一个黑盒子，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以预测的。

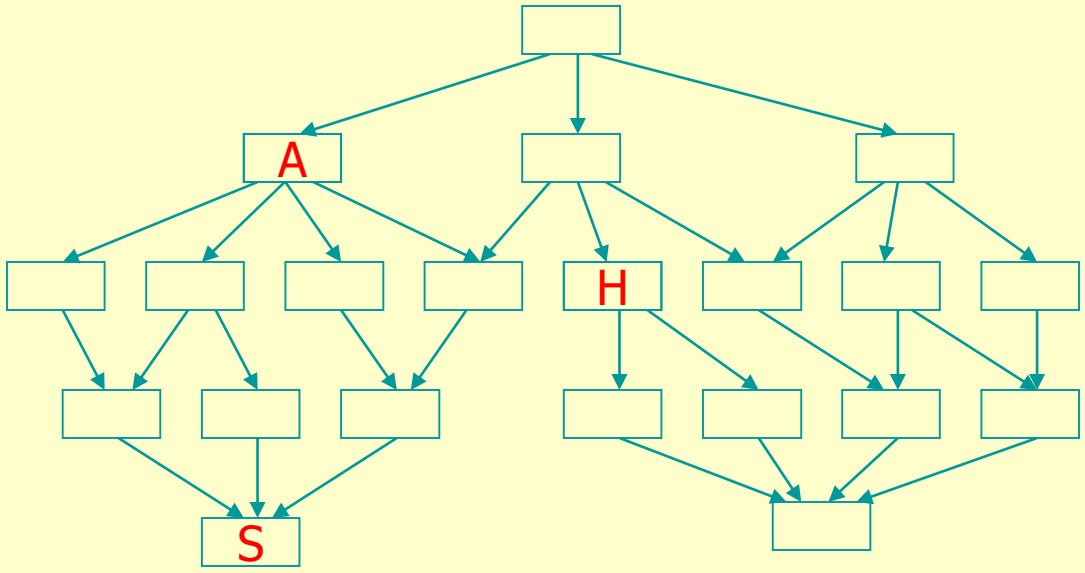
思考题：

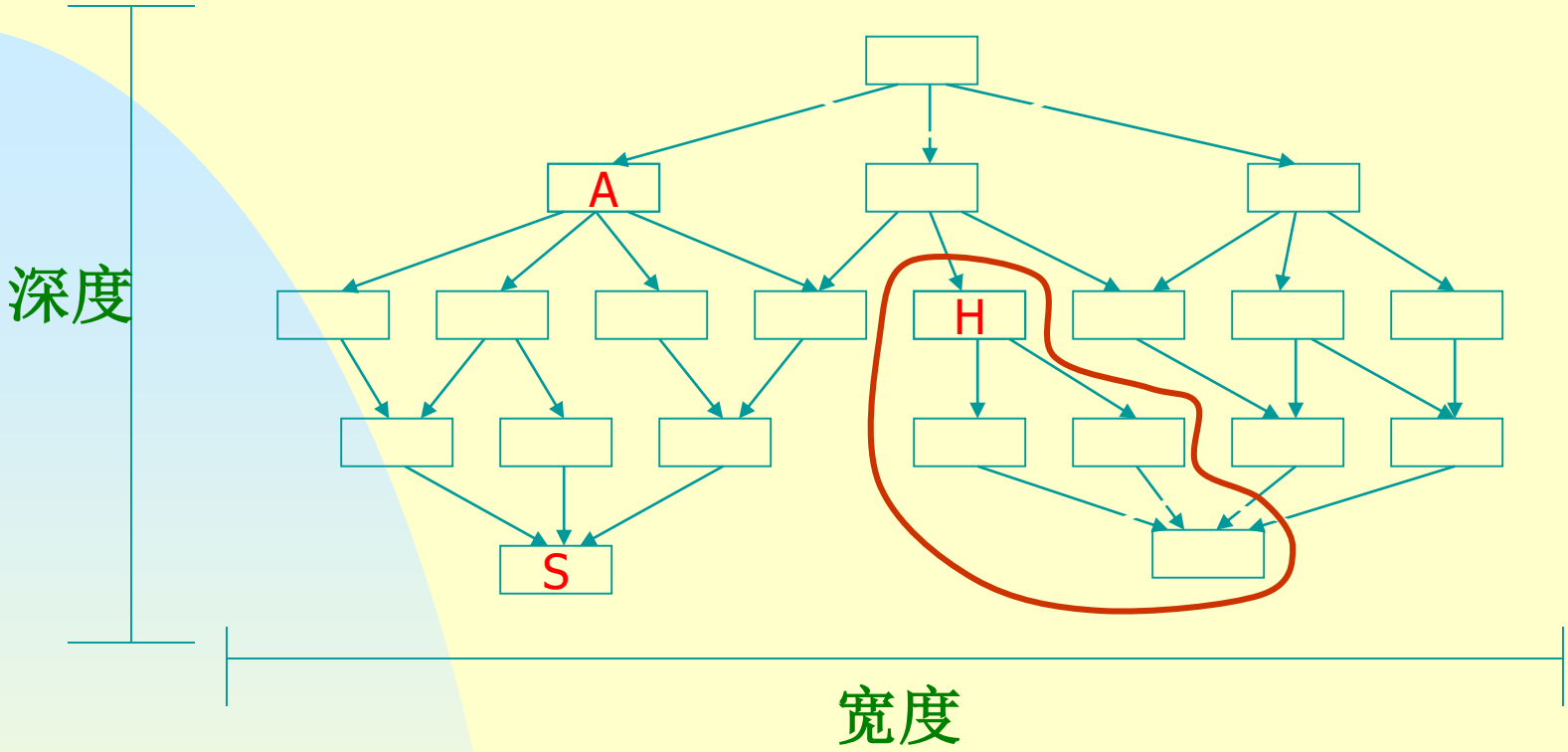
在软件结构图中：

1、软件结构的深度？宽度？

2、模块S的扇入？模块A的扇出？

3、模块H的控制域？



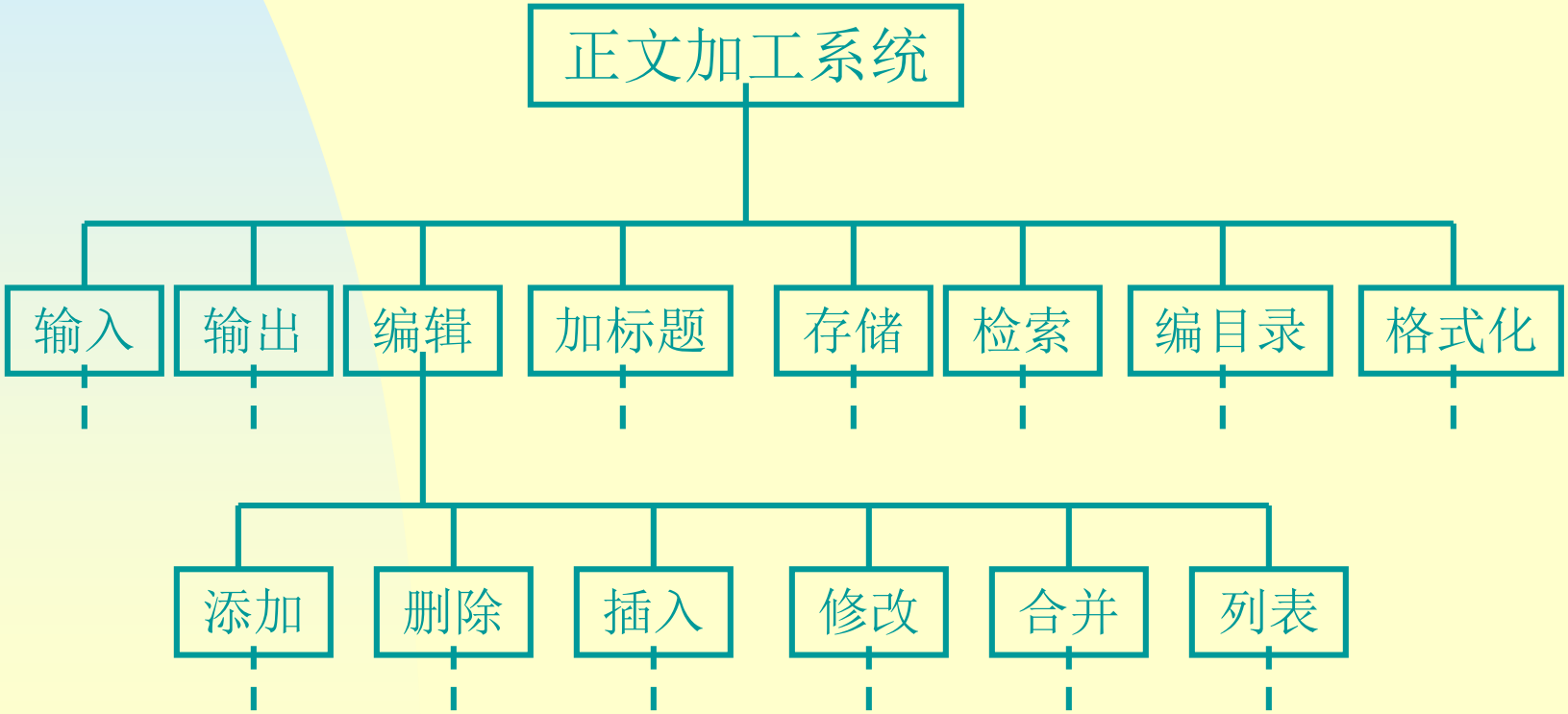


1、软件结构的深度为5。宽度为8。
2、模块S的扇入为3。模块A的扇出为4。
3、模块H的控制域：H本身及下层3个模块。

3. 概要设计阶段使用的工具

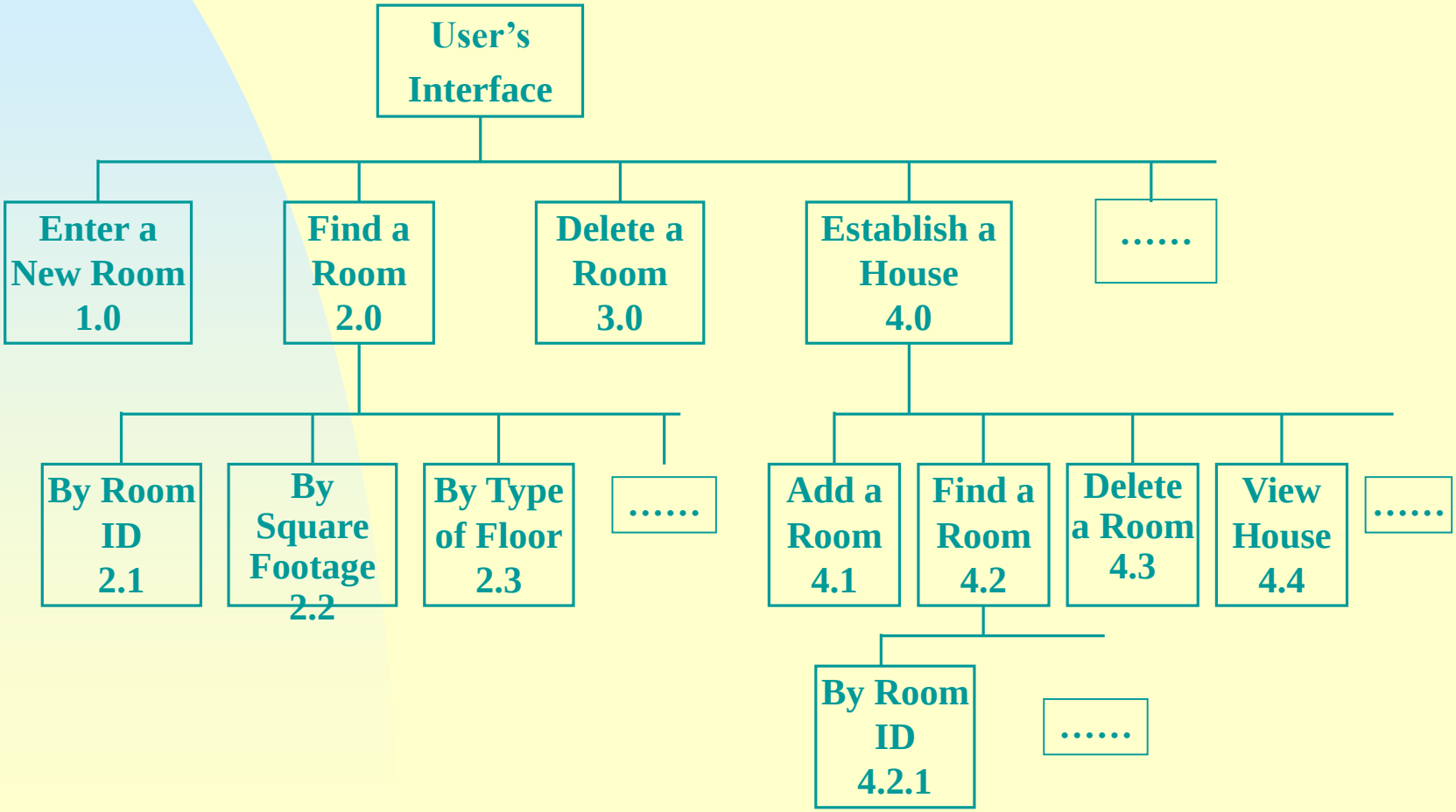
3.1 层次图

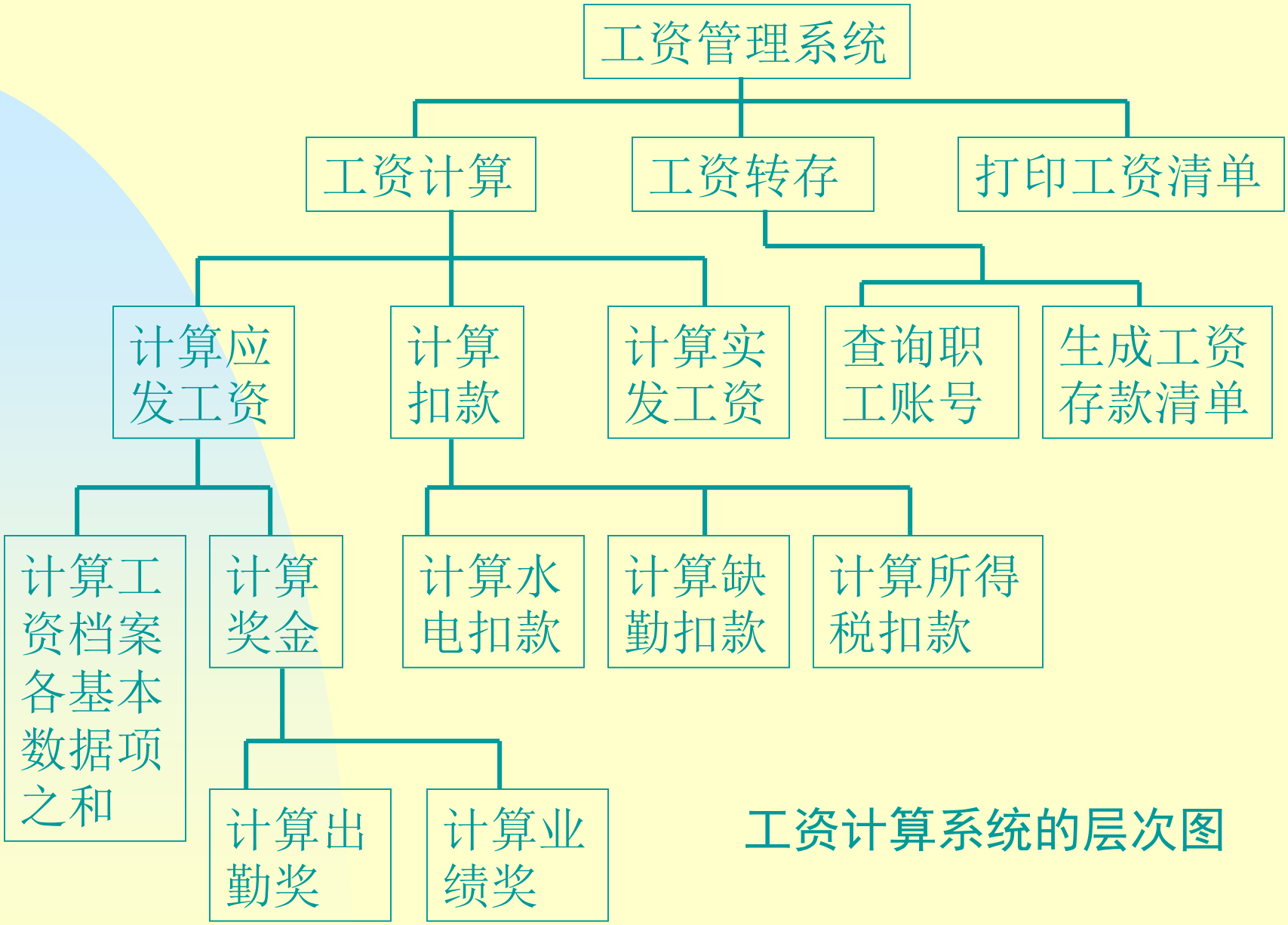
- 描绘软件的层次结构，而非数据结构
- 矩形代表模块，连线代表调用



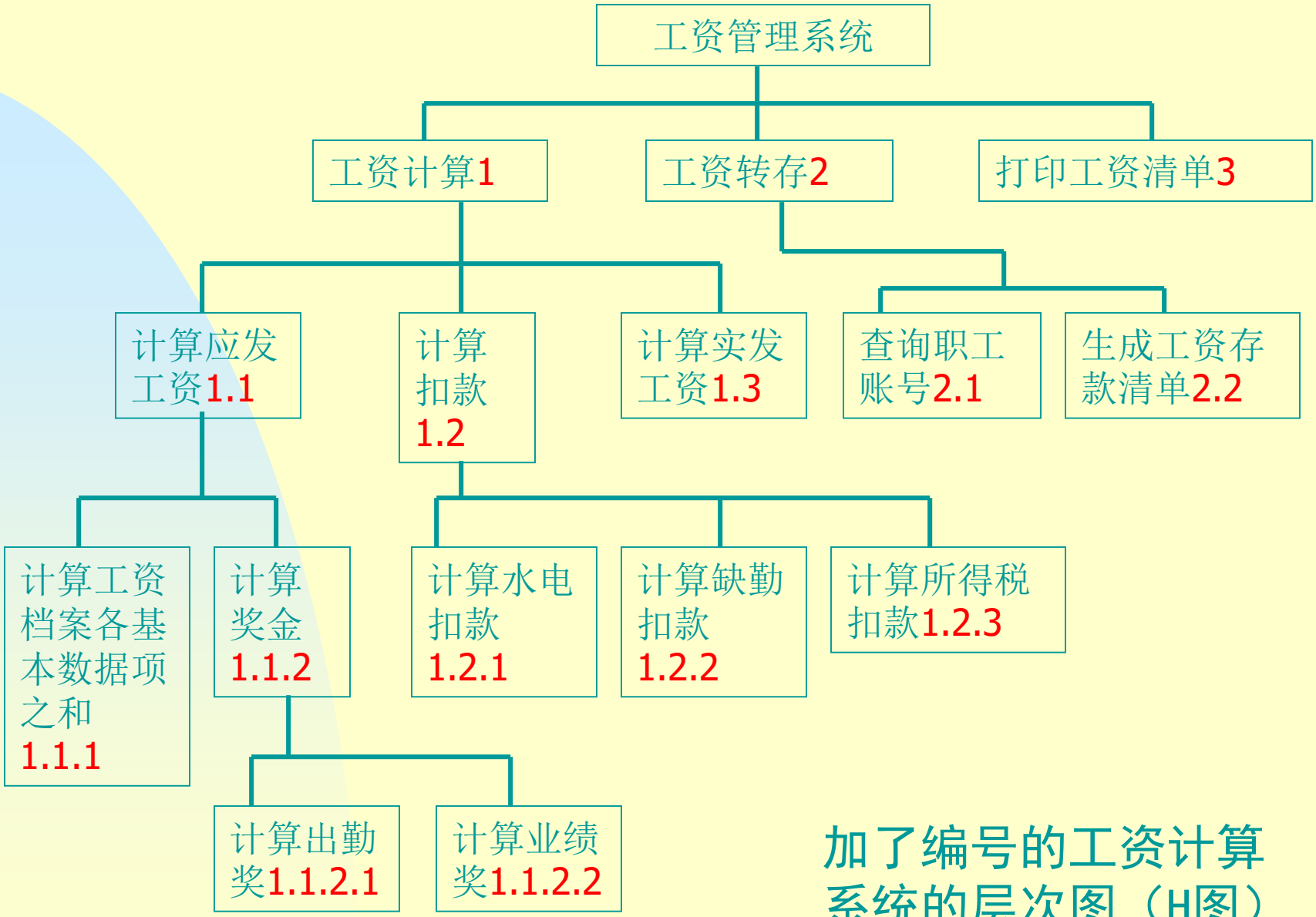
3.2 HIPO图

层次图加IPO图,对除最顶层的方框外的所有方框加编号（编号规则同数据流图）

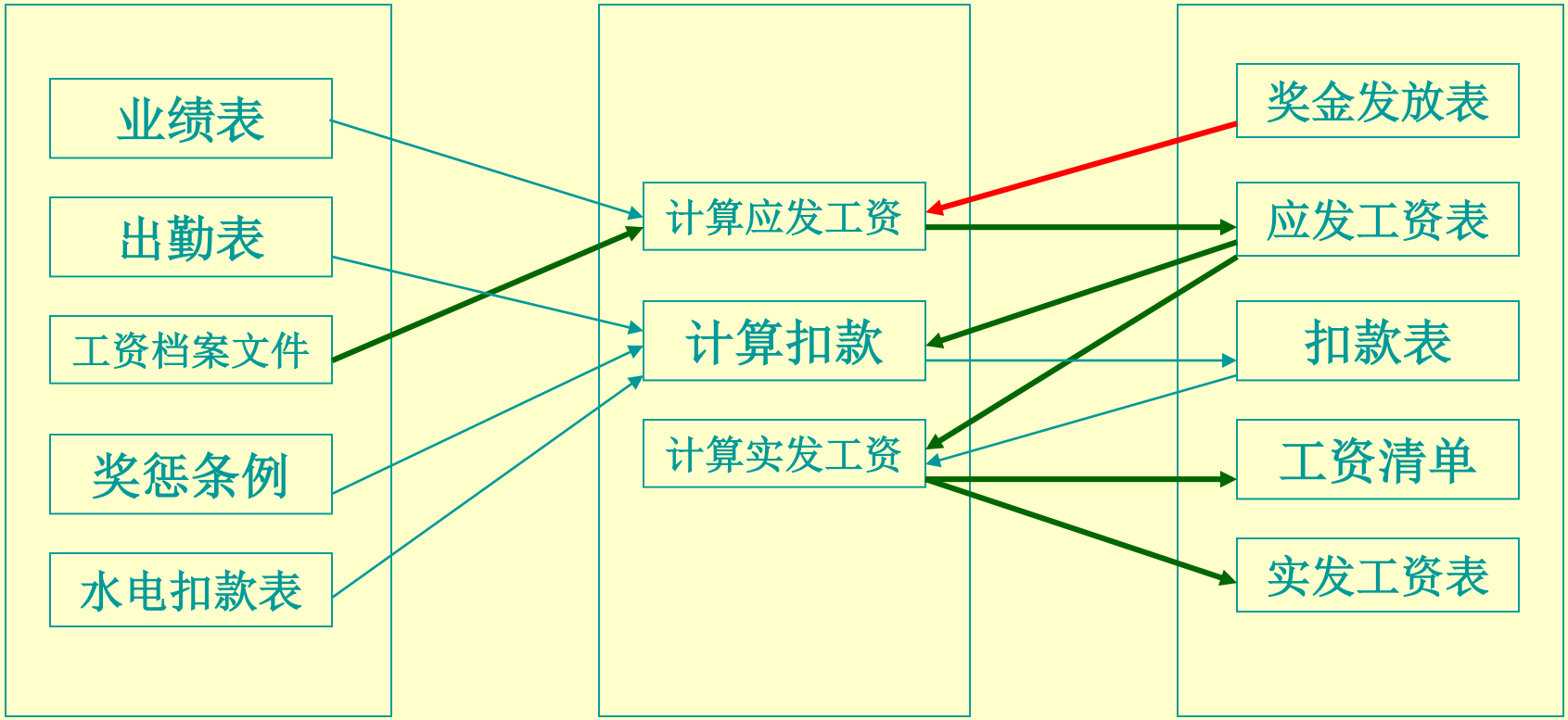



工资管理系统
工资计算
工资转存
打印工资清单
计算应发工资
计算扣款
计算实发工资
查询职工账号
生成工资存款清单
计算工资档案各基本数据项之和
计算奖金
计算水电扣款
计算缺勤扣款
计算所得税扣款
计算出勤奖
计算业绩奖

工资计算系统的层次图



加了编号的工资计算系统的层次图（H图）



计算工资模块的IPO图

改进的IPO图的形式

（IPO表）

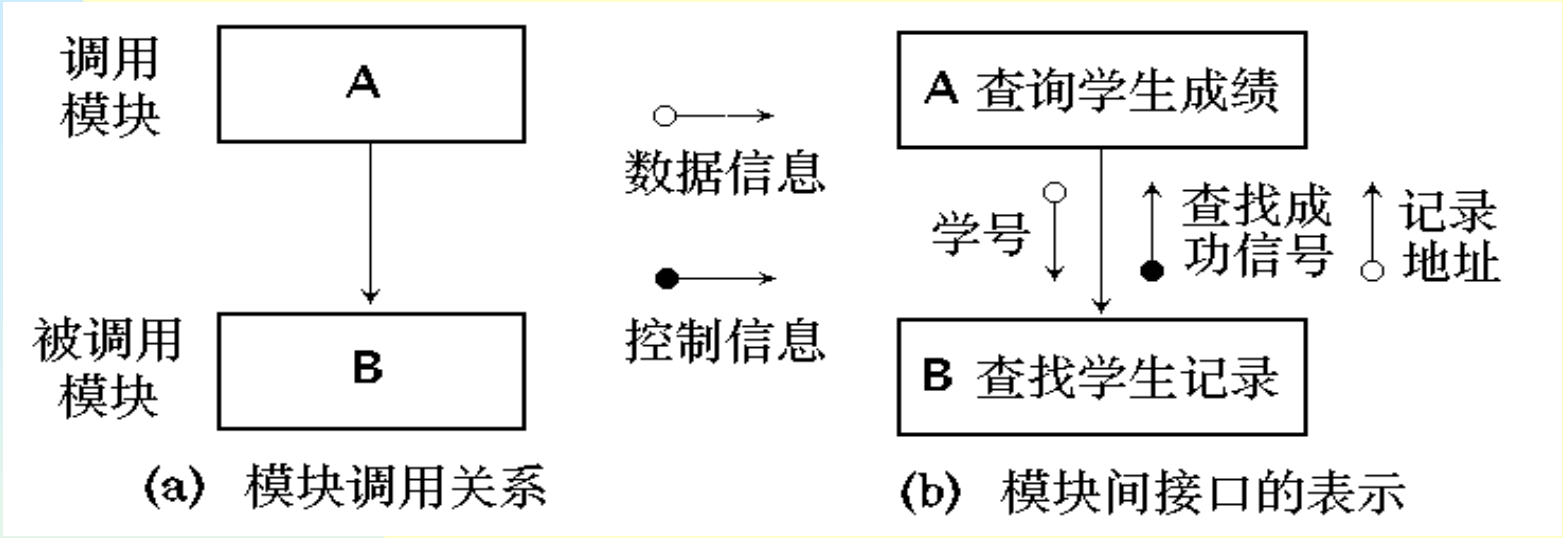
IPO表

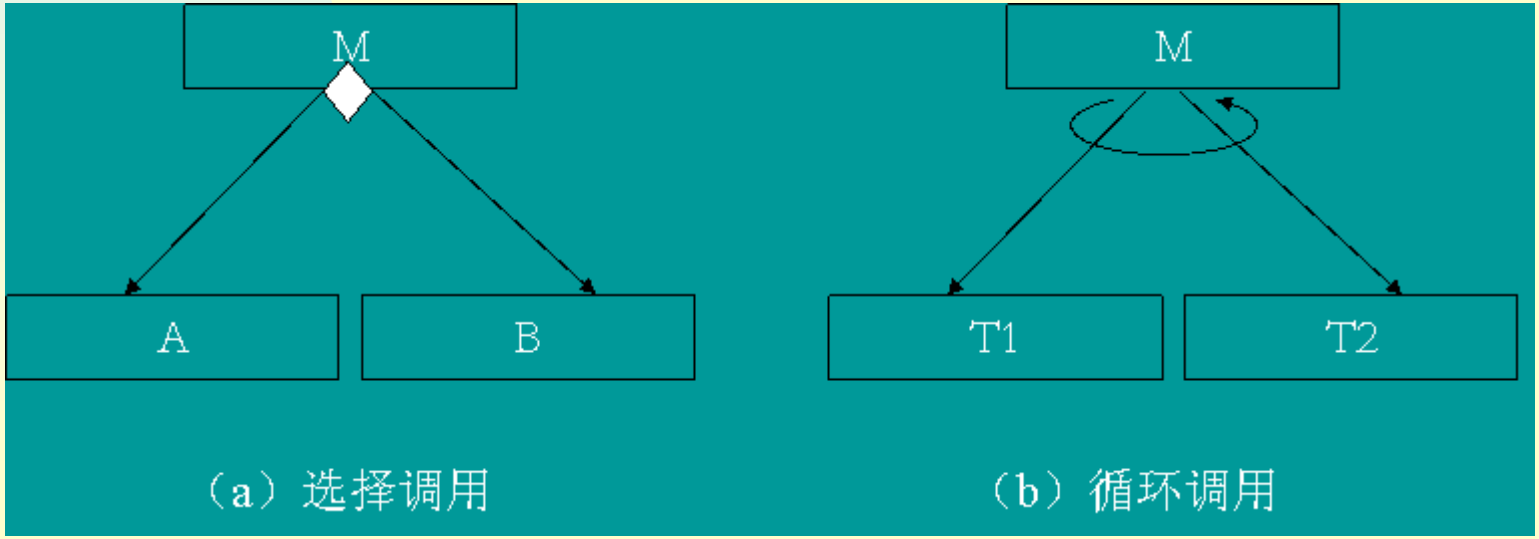
系统：———— 作者：————

模块：————日期：————编号：————

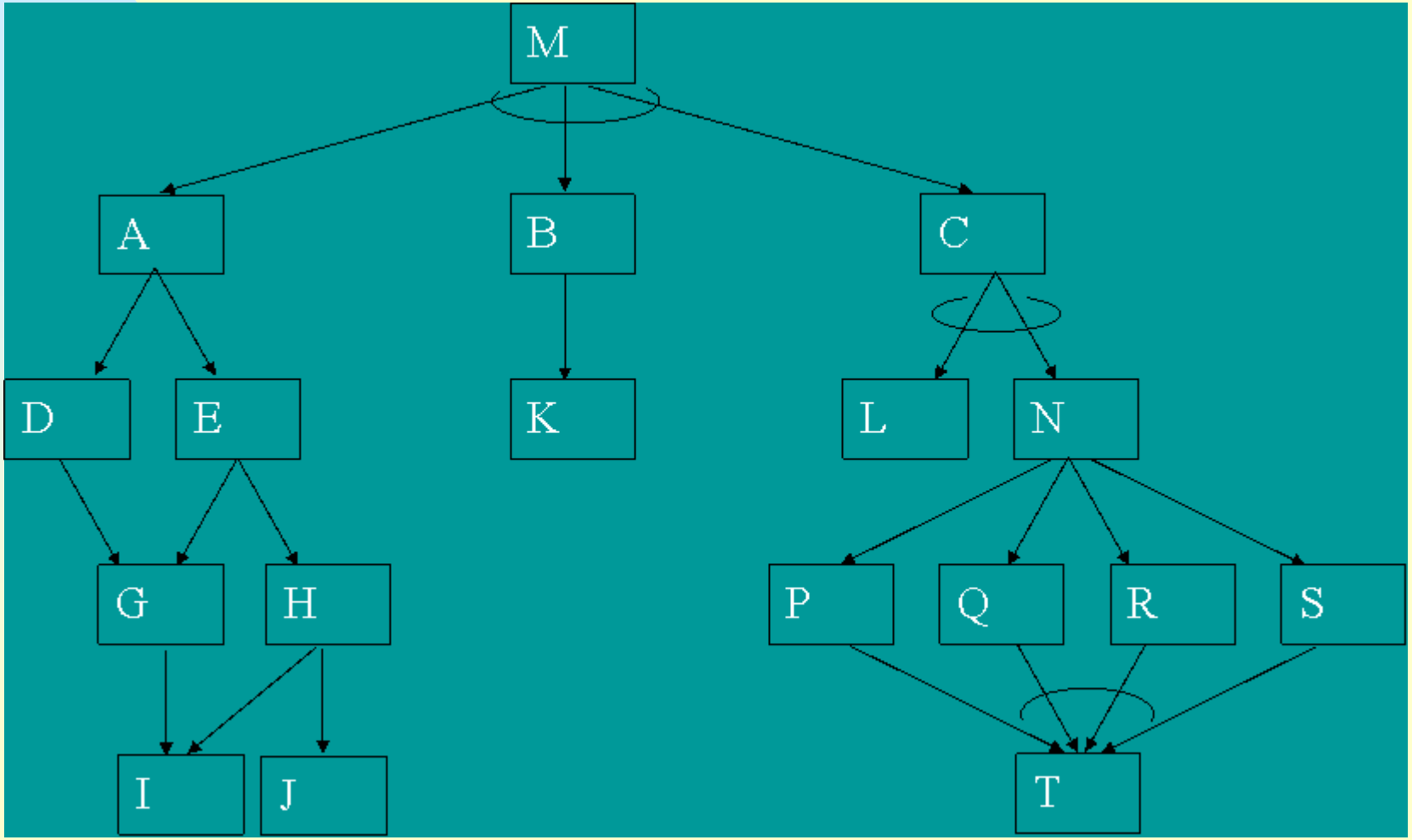
被调用：	调用：
输入：	输出：
处理：	
局部数据元素：	注释：

3.3 程序结构图 反映程序中模块之间的层次调用关系和联系





程序结构图



4. 结构化设计方法

一、数据流类型

面向数据流的设计方法。把信息流映射成软件结构，信息流的类型决定了映射的方法。

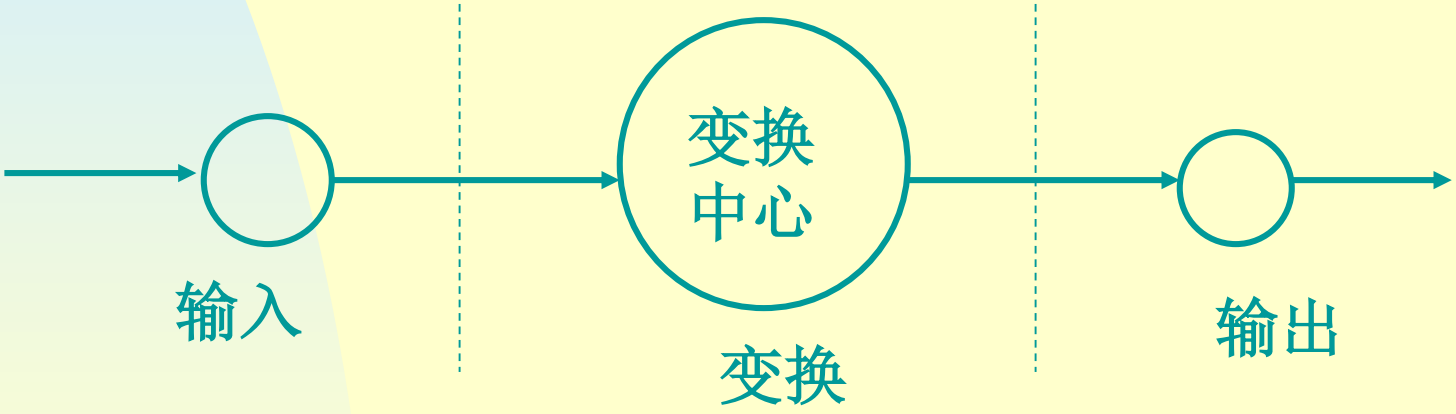
信息流具有下述两种类型：

变换流和事务流

4.1基本概念

1、变换型数据流图

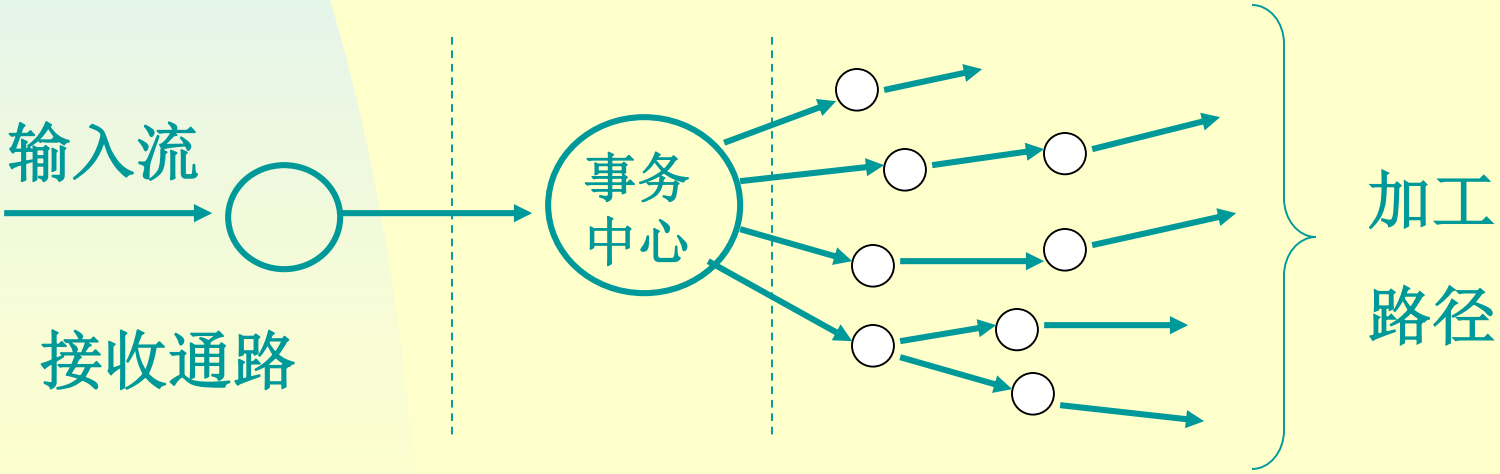
信息通常以“外部世界”的形式进入软件系统，经过处理以后再以“外部世界”的形式离开系统。由逻辑输入、变换中心、逻辑输出三部分组成。



2、事务流

它完成以下任务：

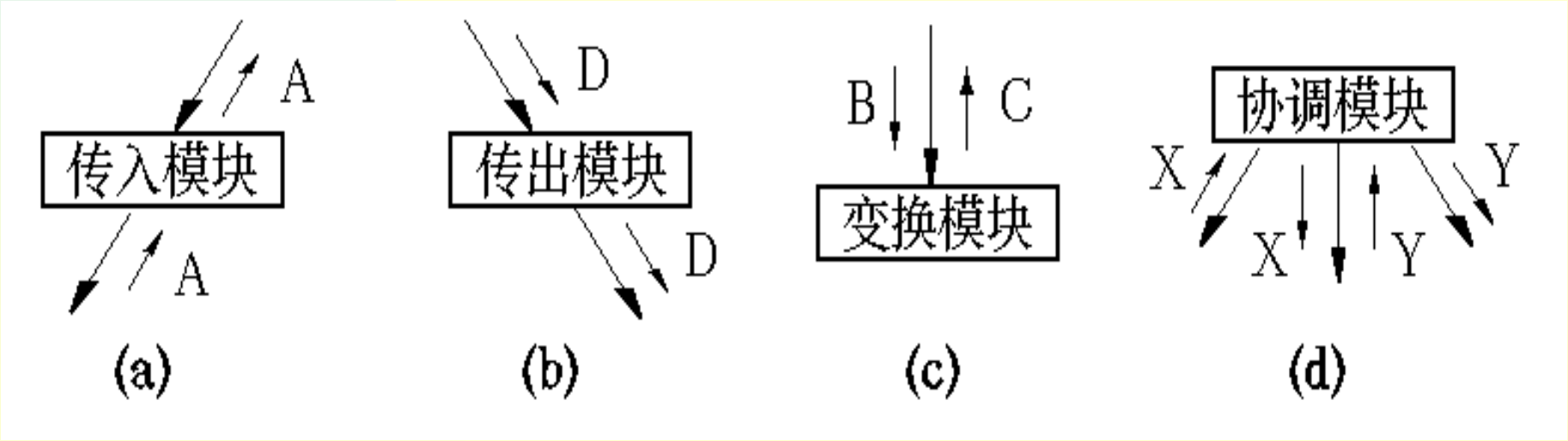
- 1）接收输入数据（事务）
- 2）分析每个事务以确定它的类型；
- 3）根据事务类型选取一条活动通路。



4.2 系统结构图中的模块

- 传入模块 — 从下属模块取得数据，经过某些处理，再将其传送给上级模块。它传送的数据流叫做逻辑输入数据流。
- 传出模块 — 从上级模块获得数据，进行某些处理，再将其传送给下属模块。它传送的数据流叫做逻辑输出数据流。

- 变换模块 — 它从上级模块取得数据，进行特定的处理，转换成其它形式，再传送回上级模块。它加工的数据流叫做变换数据流。
- 协调模块 — 对所有下属模块进行协调和管理的模块。



4.3 结构化设计过程

（1）首先研究、分析和审查数据流图。从软件的需求规格说明中弄清数据流加工的过程，检查有无遗漏或不合理之处，对于发现的问题及时解决并修改。

（2）根据数据流图决定问题的类型。数据处理问题典型的类型有两种，分别是变换型和事务型，针对两种不同的类型分别进行分析处理。

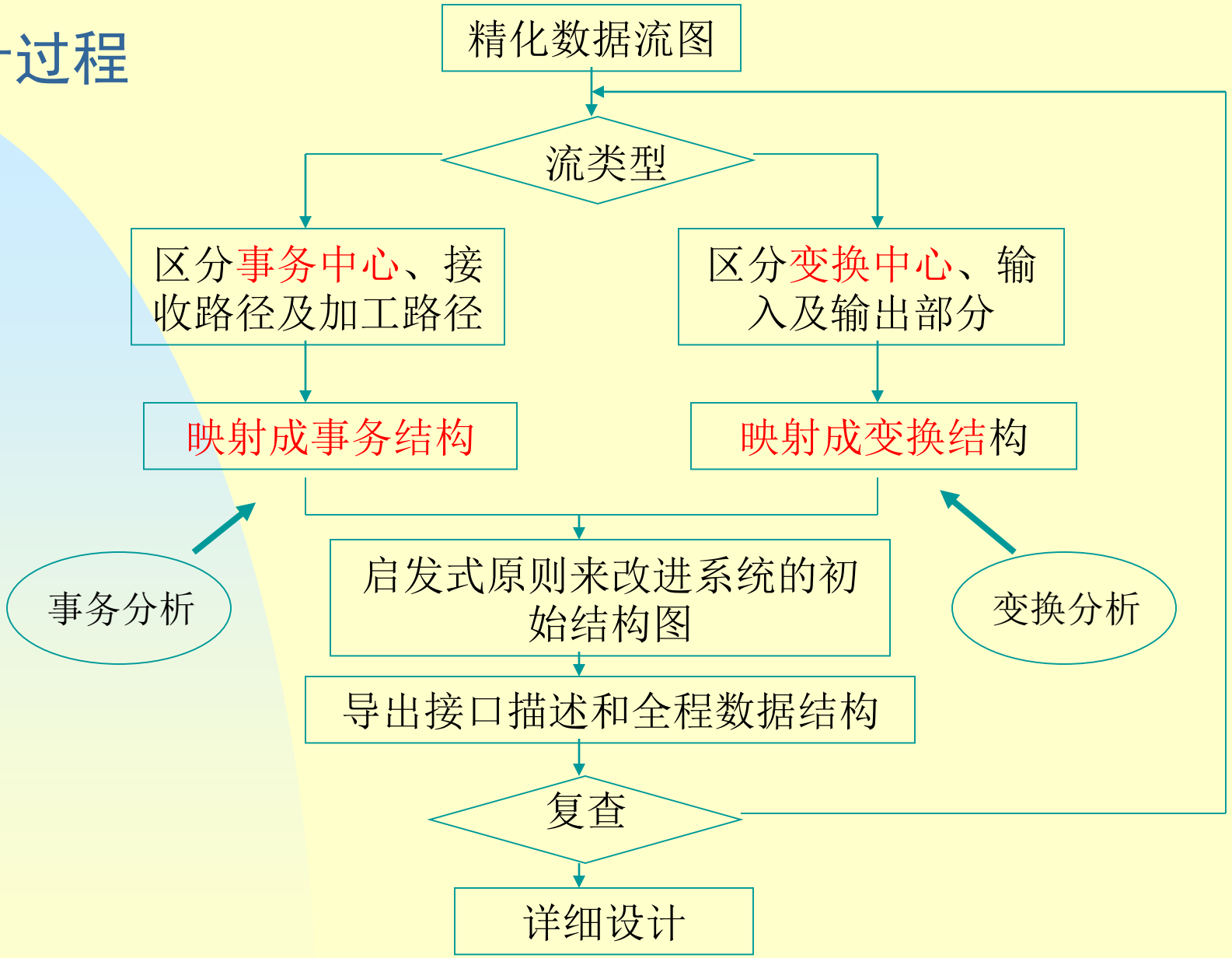
（3）由数据流图推导出系统的初始结构图。

（4）利用一些启发式原则来改进系统的初始结构图，直到得到符合要求的结构图为止。

（5）描述模块功能、接口及全局数据结构，修改和补充数据字典。

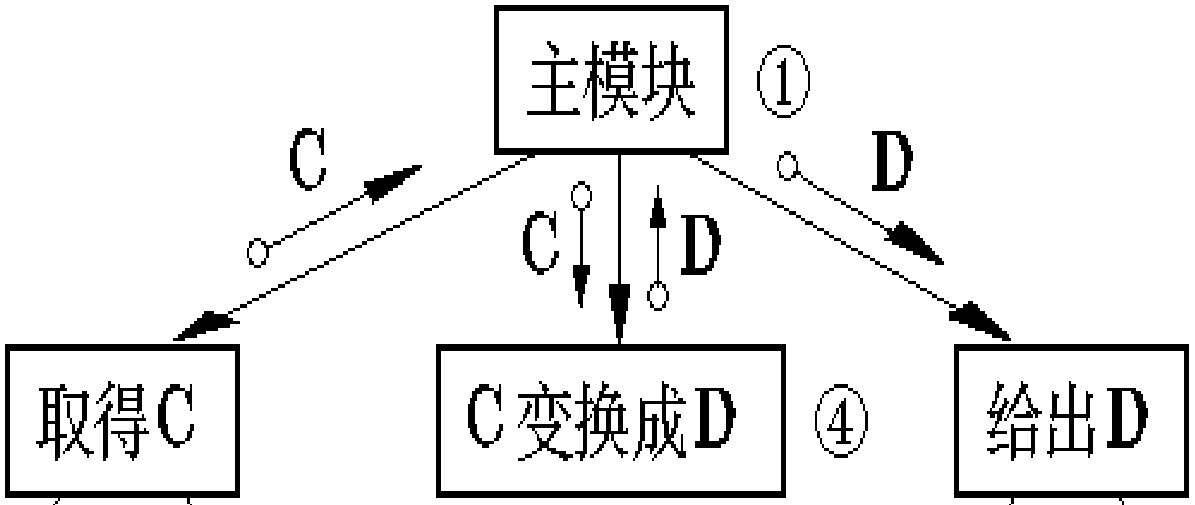
（6）复查，如果出现错误，转入第二步修改完善，否则进入详细设计，同时制定测试计划。

设计过程


精化数据流图
流类型
区分事务中心、接收路径及加工路径
区分变换中心、输入及输出部分
映射成事务结构
映射成变换结构
事务分析
变换分析
启发式原则来改进系统的初始结构图
导出接口描述和全程数据结构
复查
详细设计

4.4 变换分析

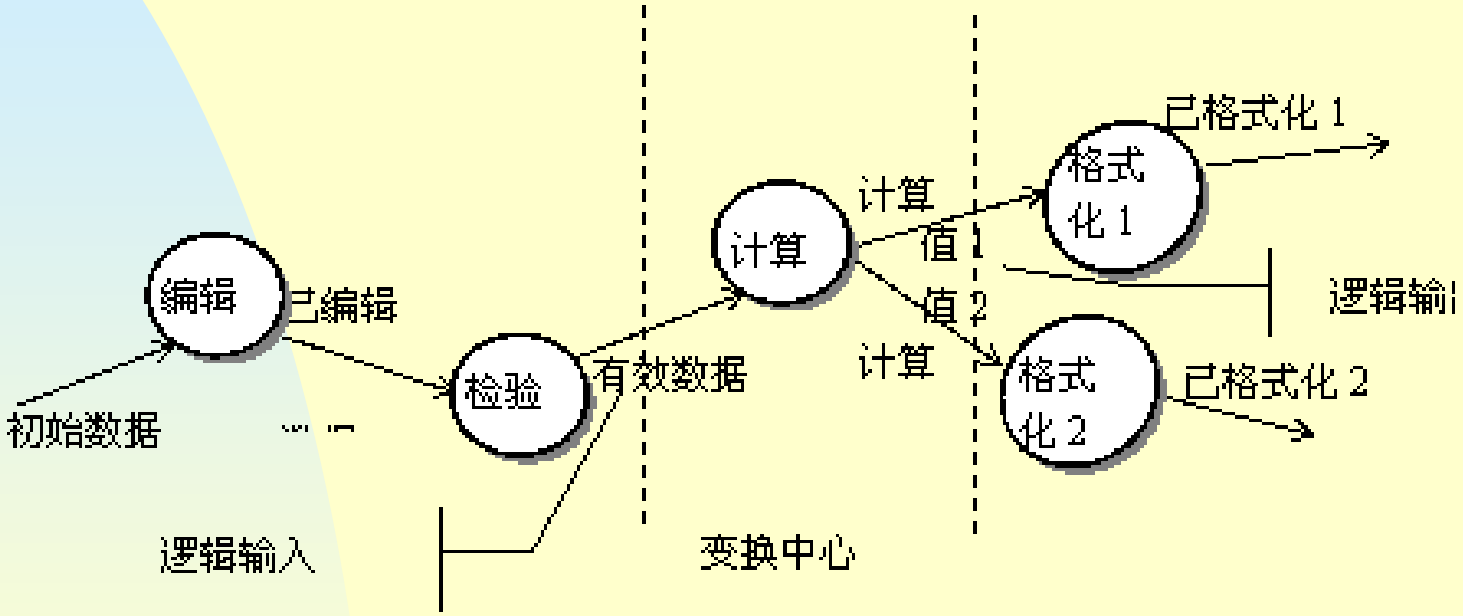
- 变换型数据处理问题的工作过程大致分为三步，即取得数据，变换数据和给出数据。
- 相应于取得数据、变换数据、给出数据，变换型系统结构图由输入、中心变换和输出等三部分组成。


主模块
①
C
D
C
D
取得C
C变换成D
④
给出D

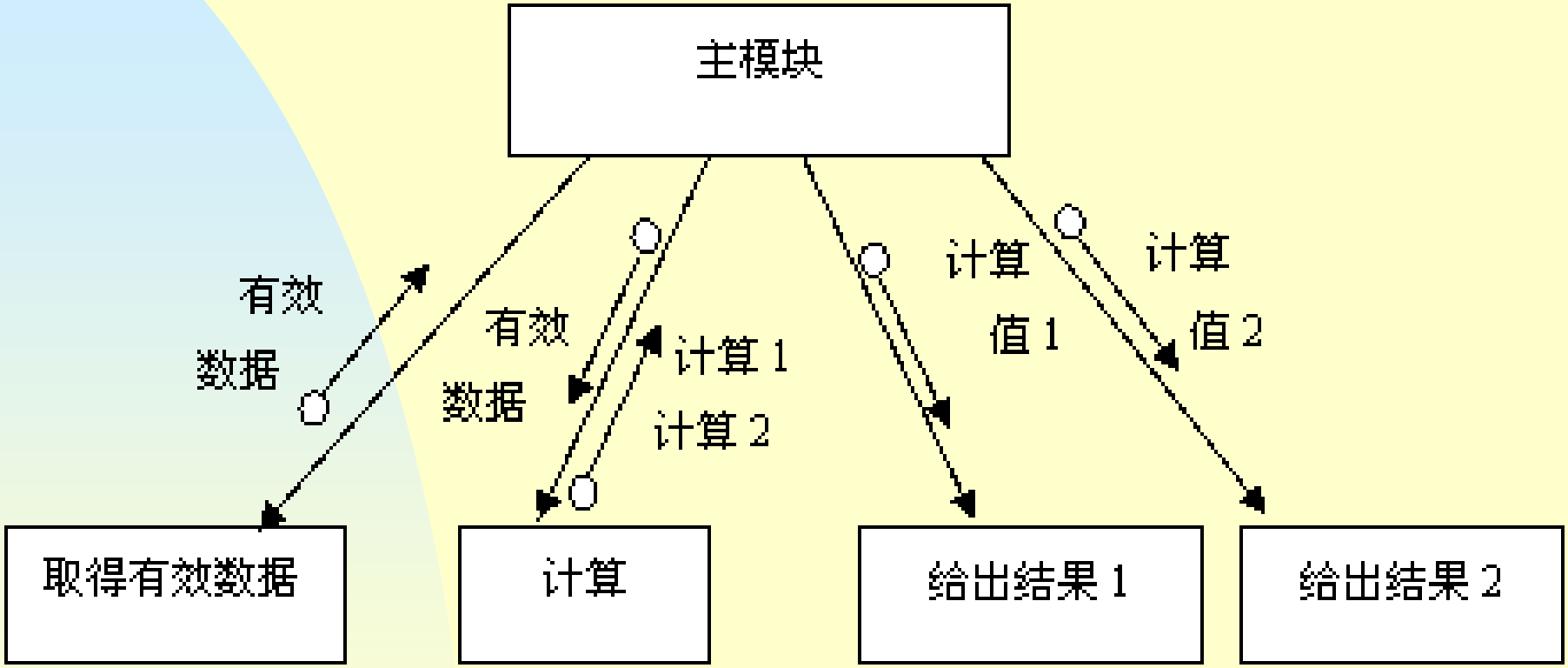
变换分析步骤

- 变换分析方法由以下四步组成：
  - 重画数据流图；
  - 区分有效(逻辑)输入、有效(逻辑)输出和中心变换部分；
  - 进行一级分解，设计上层模块；
  - 进行二级分解，设计输入、输出和中心变换部分的中、下层模块。

- 1．确定数据流图（DFD）中的变换中心，逻辑输入、逻辑输出

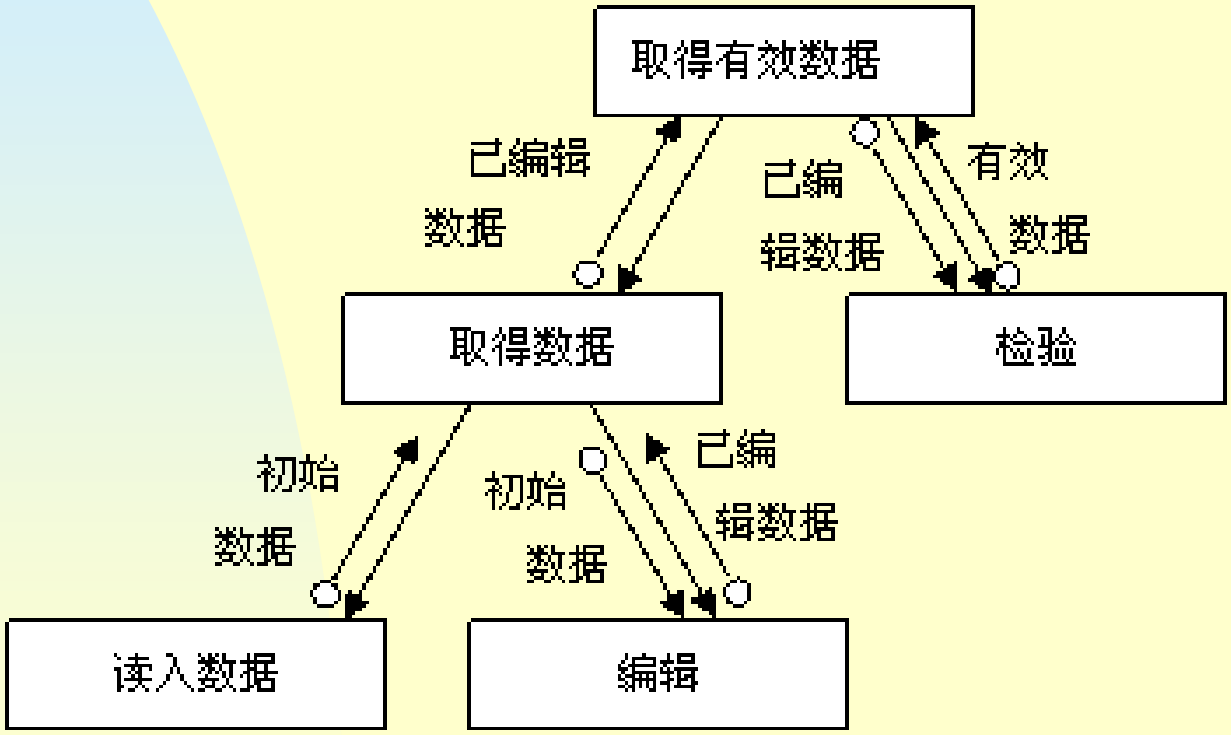


- 2．设计软件结构的顶层和第一层——变换结构



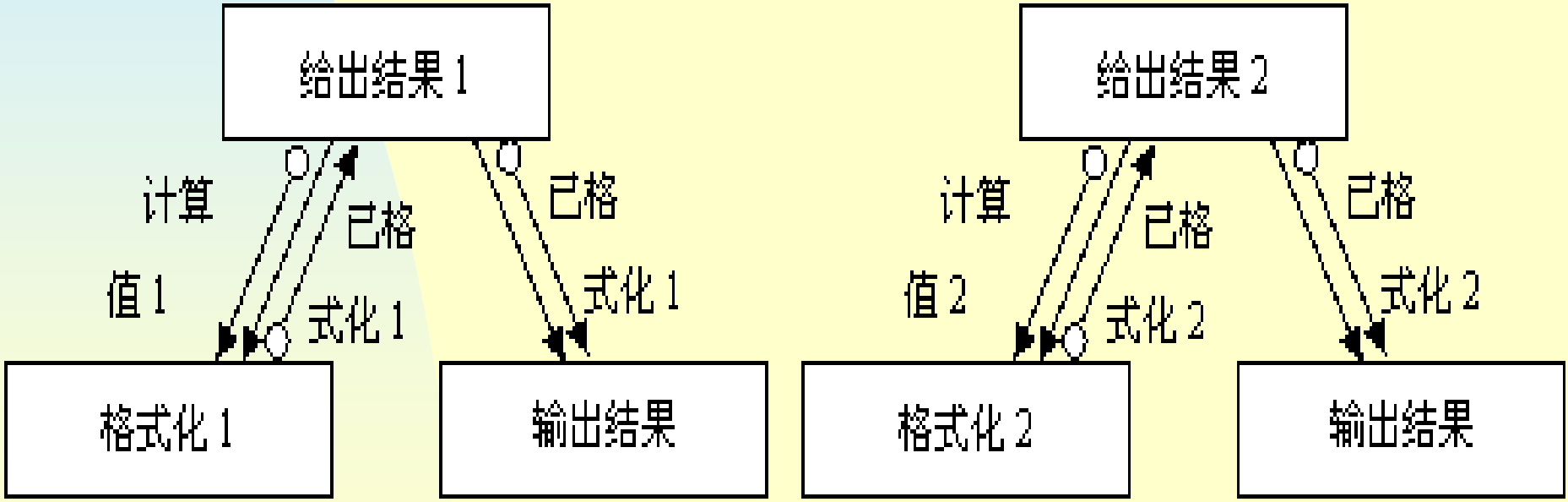
- 3．设计中、下层模块

（1）输入模块下属模块的设计。



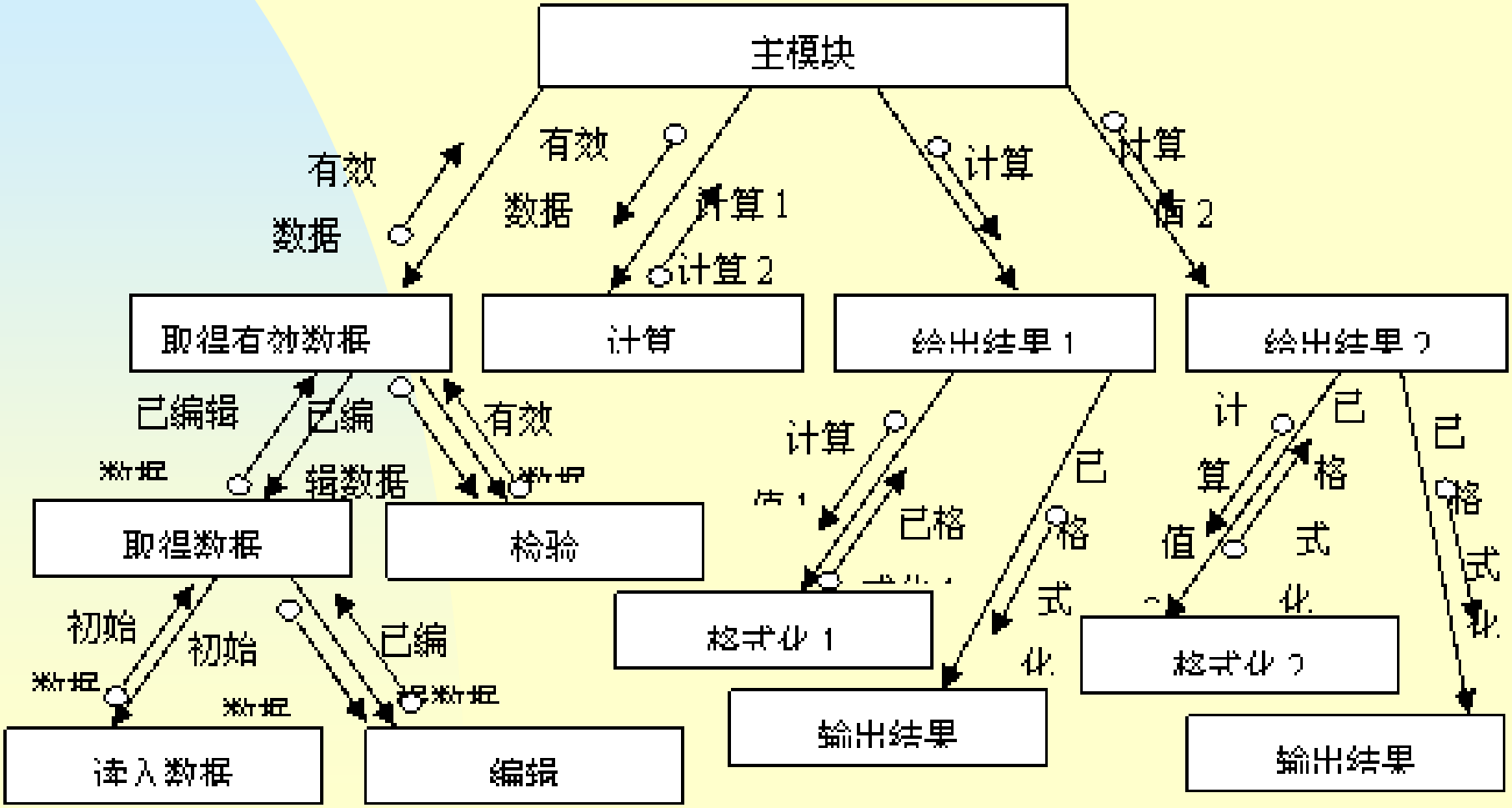
- 3．设计中、下层模块

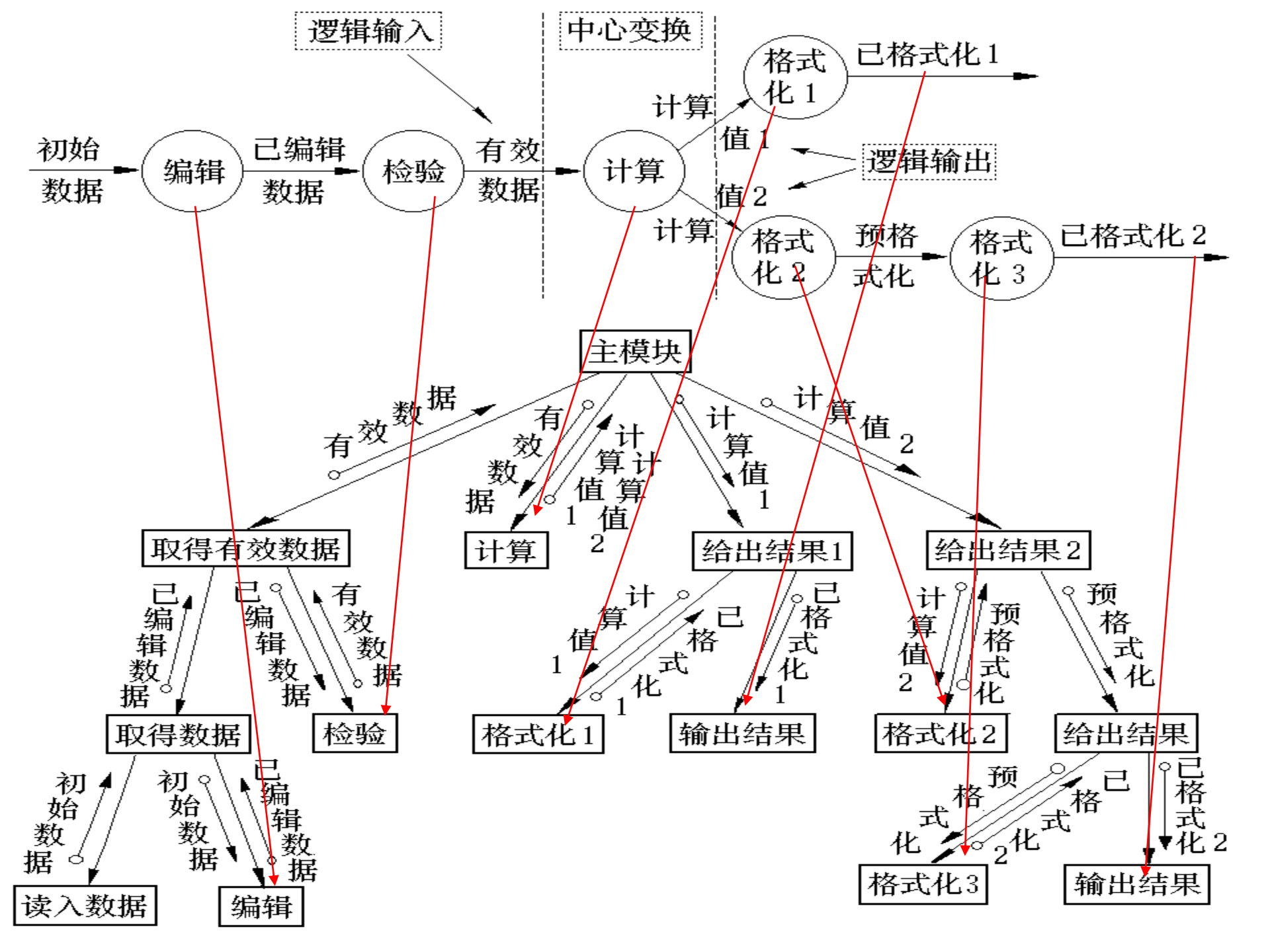
（2）输出模块下属模块的设计。



- 3．设计中、下层模块

（3）变换模块下属模块的设计




逻辑输入
中心变换
初始数据
编辑
已编辑数据
检验
有效数据
计算
格式化1
已格式化1
值1
值2
逻辑输出
格式化2
预格式化
格式化3
已格式化2
主模块
取得有效数据
计算
给出结果1
给出结果2
取得数据
检验
格式化1
输出结果
格式化2
给出结果
读入数据
编辑
格式化3
输出结果

4. 设计优化

（1）输入部分优化。对每个物理输入设置专门模块，以体现系统的外部接口，其它输入模块并非真正输入，当它与转换数据的模块都很简单时，可将它们合并成一个模块。

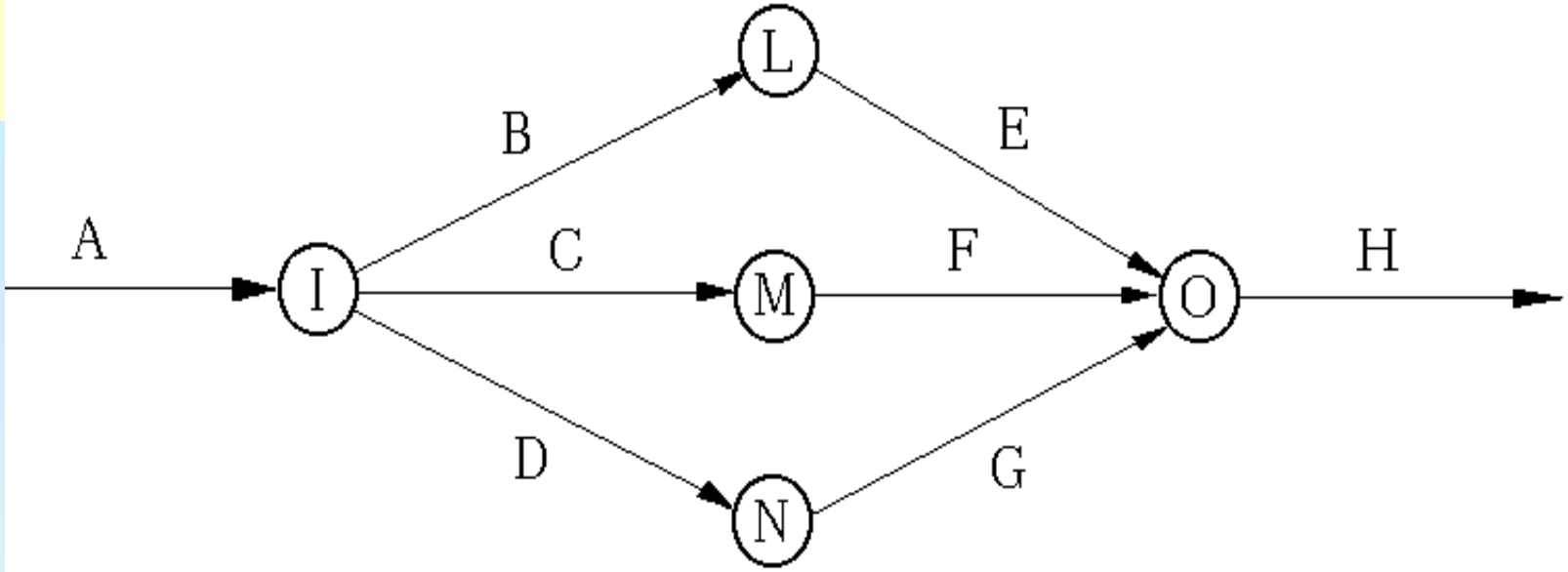
（2）输出部分优化。为每个物理输出设置专门模块，同时注意把相同或类似的物理输出模块合并在一起，以降低耦合度，提高初始结构图的质量。

（3）变换部分优化。根据设计准则，对模块进行合并或调整。

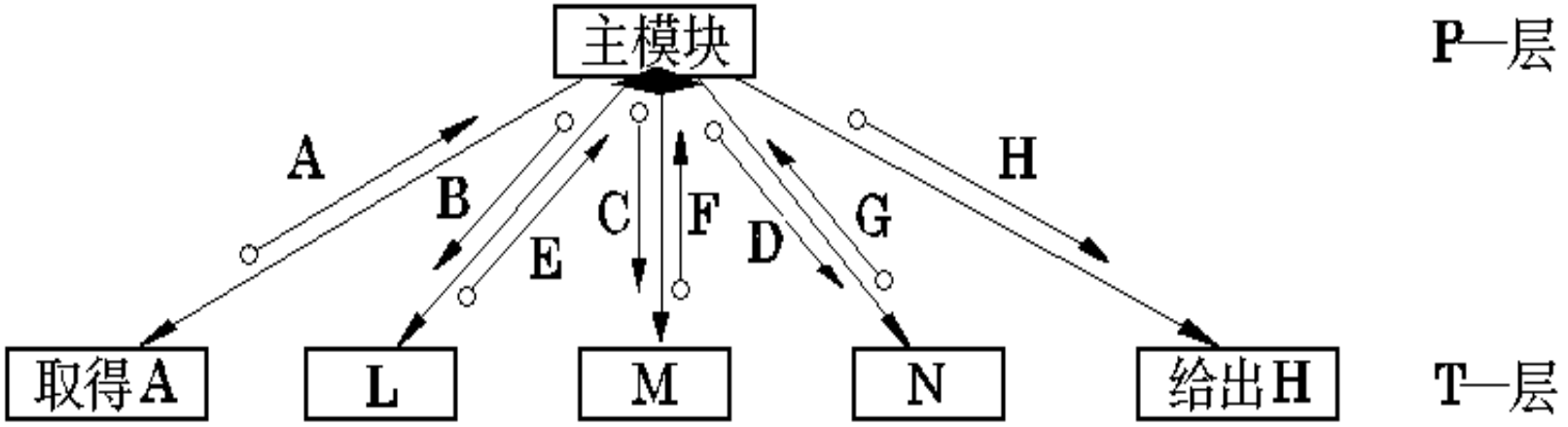
4.5 事务分析

- 在很多软件应用中，存在某种作业数据流，它可以引发一个或多个处理，这些处理能够完成该作业要求的功能。这种数据流就叫做事务。
- 与变换分析一样，事务分析也是从分析数据流图开始，自顶向下，逐步分解，建立系统到结构图。

- 在事务型系统结构图中，存在一个事务中心模块，它接受一项事务，根据事务处理的特点和性质，选择分派一个适当的处理单元。各事务处理模块并列。每个事务处理模块可能要调用若干个操作模块，而操作模块又可能调用若干个细节模块。



(a)



(b)

事务分析过程

① 识别事务源

利用数据流图和数据词典，从问题定义和需求分析的结果中，找出各种需要处理的事务。通常，事务来自物理输入装置。有时，设计人员还必须区别系统的输入、中心加工和输出中产生的事务。

②注意利用公用模块

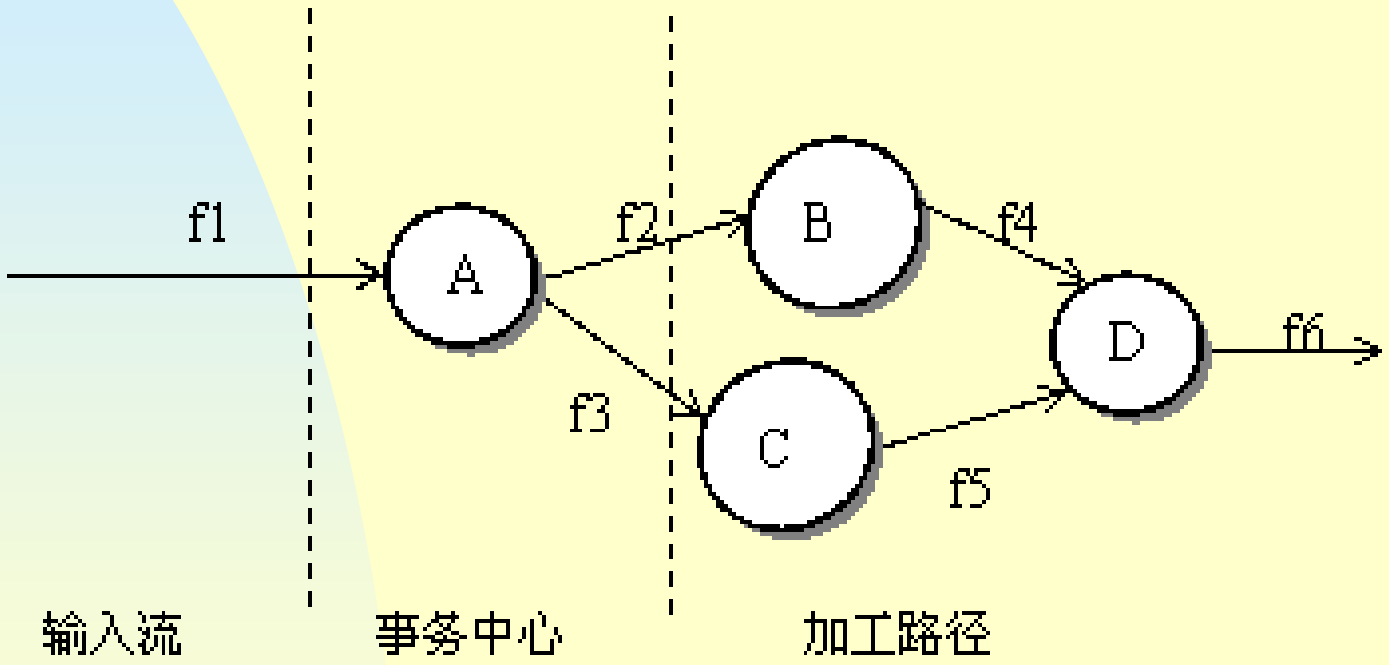
在事务分析的过程中，如果不同事务的一些中间模块可由具有类似的语法和语义的若干个低层模块组成，则可以把这些低层模块构造成公用模块。

③对每一事务，或对联系密切的一组事务，建立一个事务处理模块；

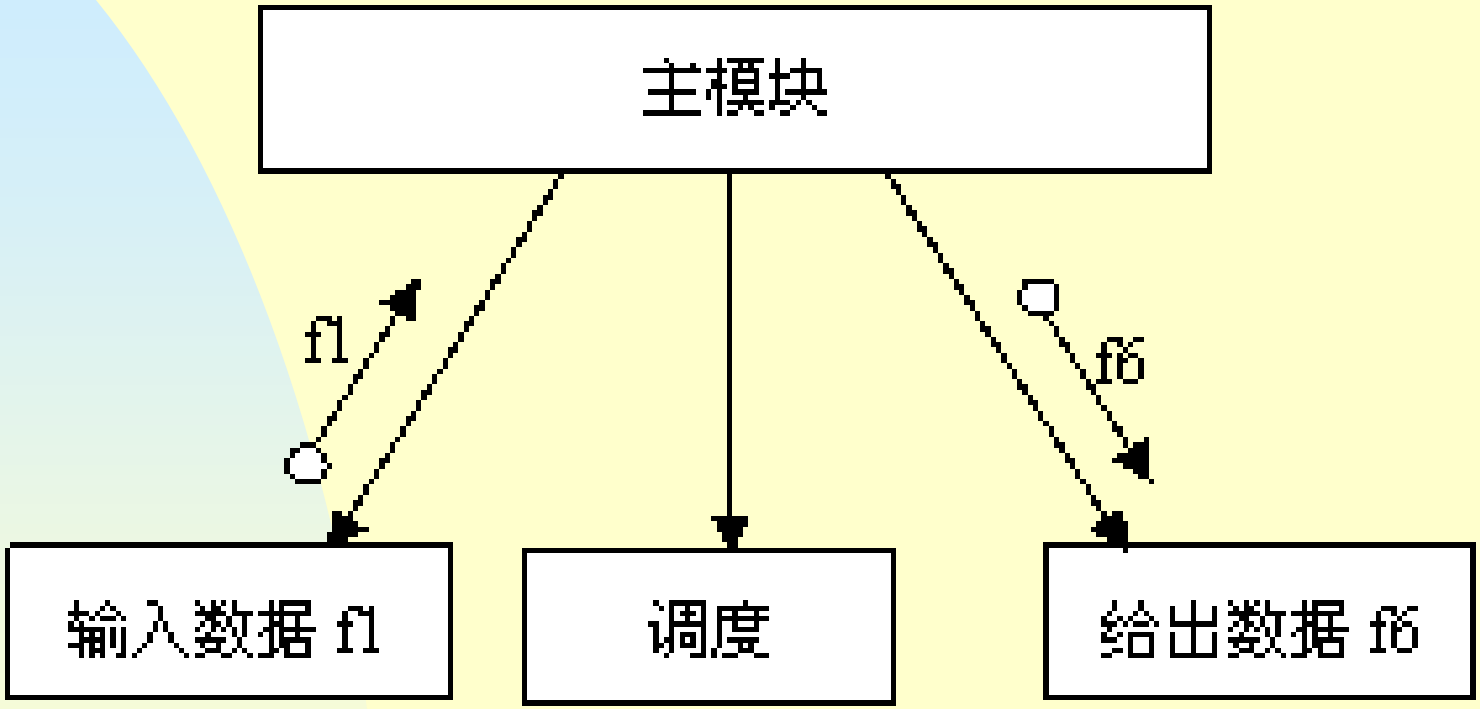
如果发现在系统中有类似的事务，可以把它们组成一个事务处理模块。

④下层操作模块和细节模块的共享

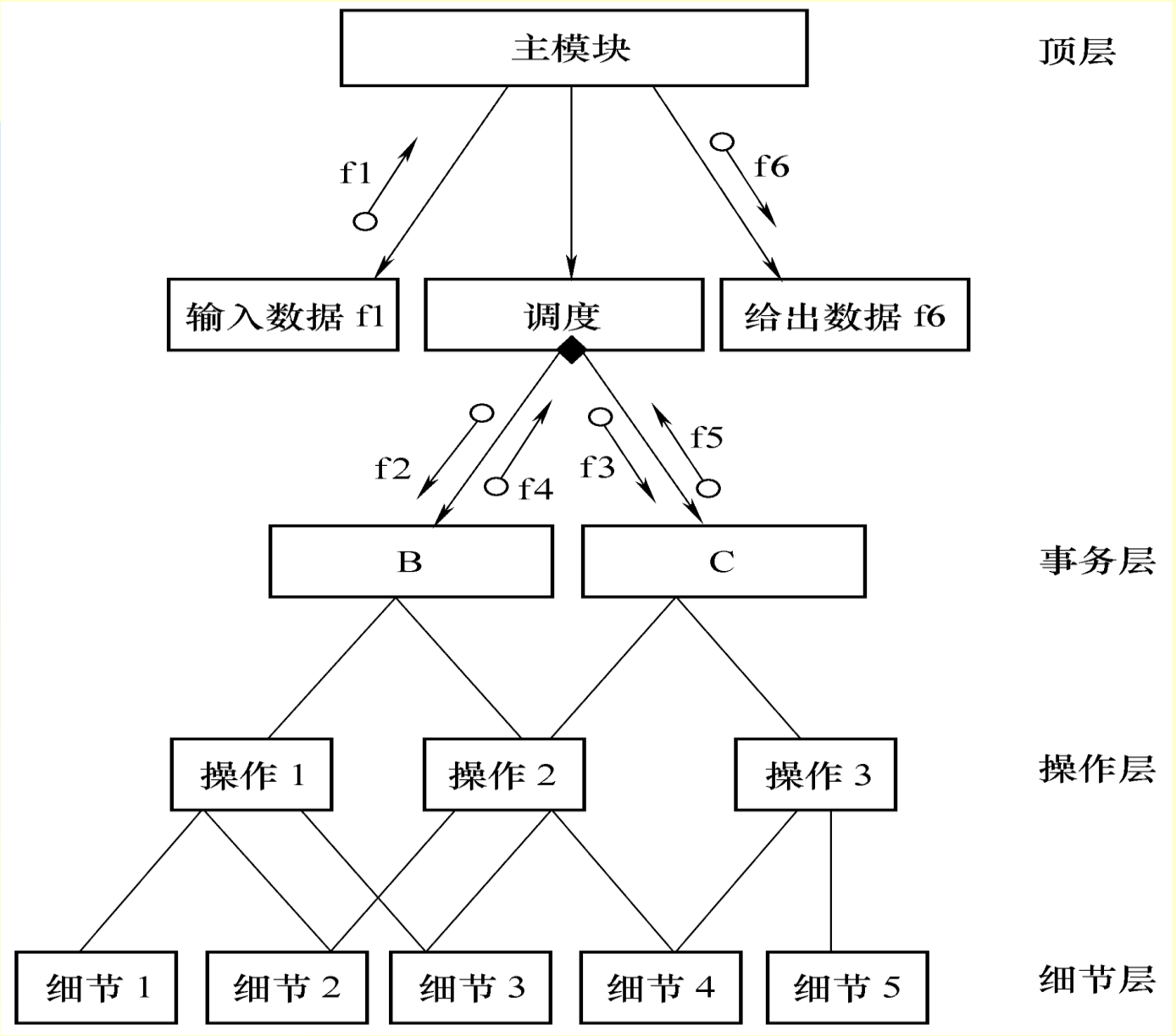
1. 确定数据流图中的事物中心和加工路径

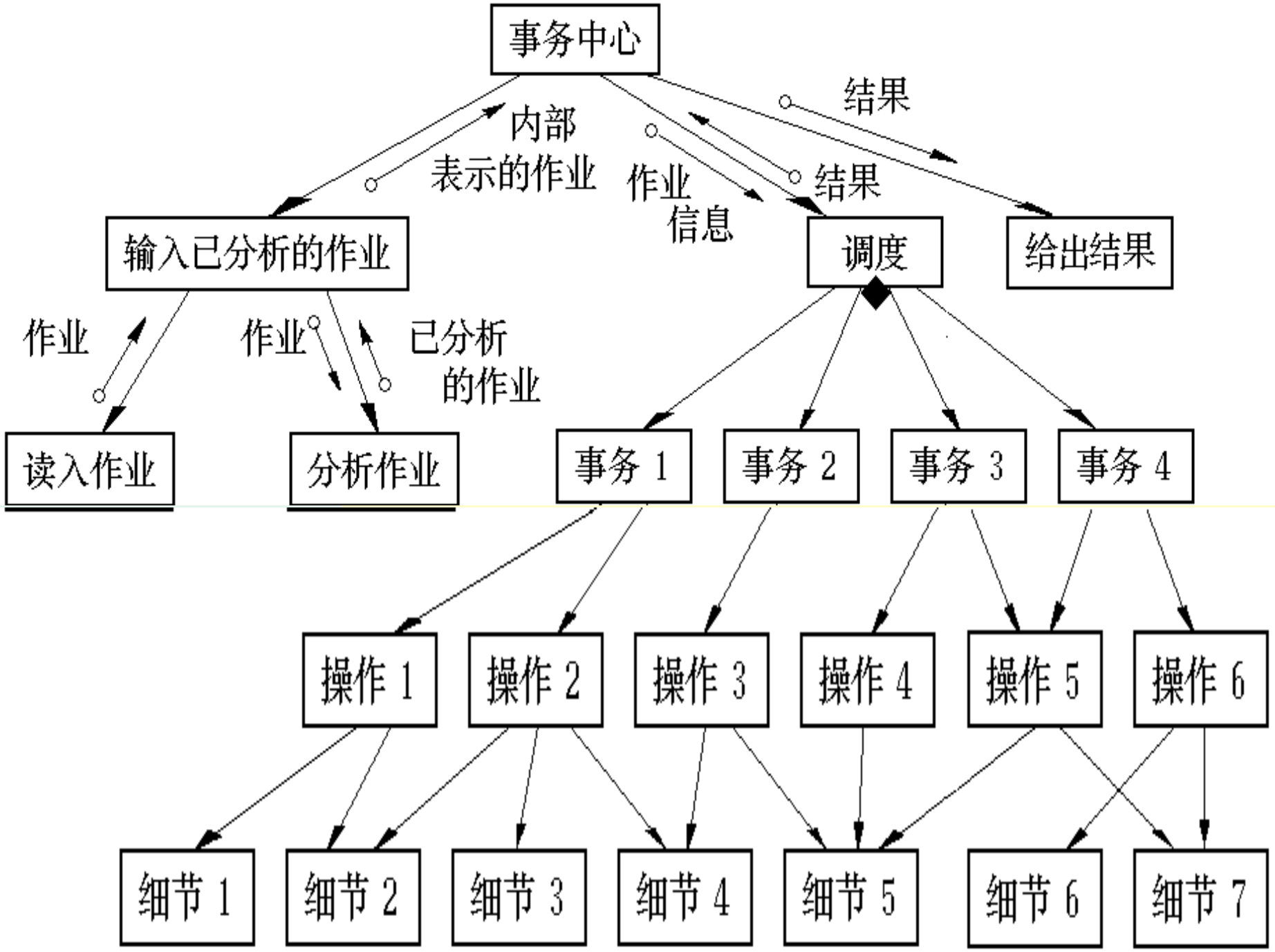


2. 设计软件结构的顶层和一层——事务结构



3. 设计中、下层模块并优化

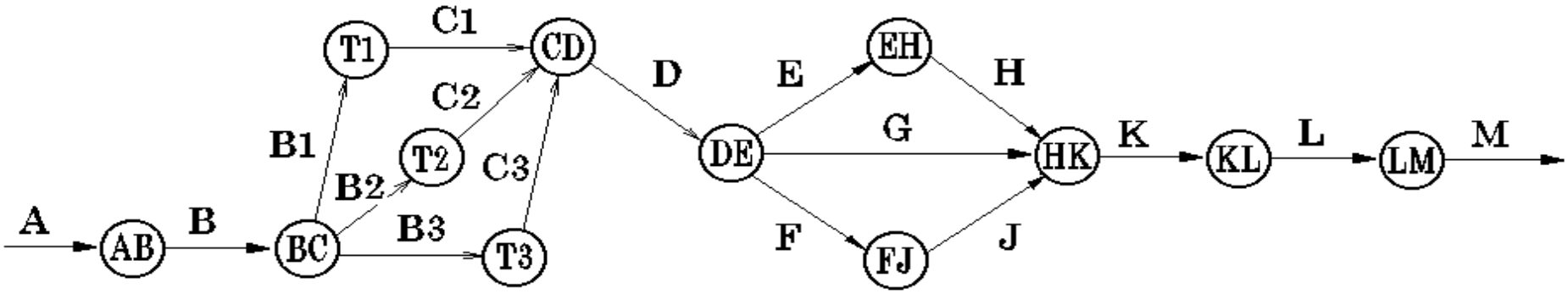



事务中心
结果
内部
表示的作业
作业
信息
结果
输入已分析的作业
调度
给出结果
作业
作业
已分析
的作业
读入作业
分析作业
事务 1
事务 2
事务 3
事务 4
操作 1
操作 2
操作 3
操作 4
操作 5
操作 6
细节 1
细节 2
细节 3
细节 4
细节 5
细节 6
细节 7

4.6 混合结构分析

- 一般而言，一个大型的软件系统是变换型结构和事务型结构的混合结构。通常利用以变换分析为主，事务分析为辅的方式进行软件结构设计。
- 在系统结构设计时，首先利用变换分析方法把软件系统分为输入、中心变换和输出三个部分，设计上层模块，即主模块和第一层模块，然后，根据数据流图各部分的结构特点，适当地利用变换分析或事务分析，即得到初始系统结构图的一个方案。

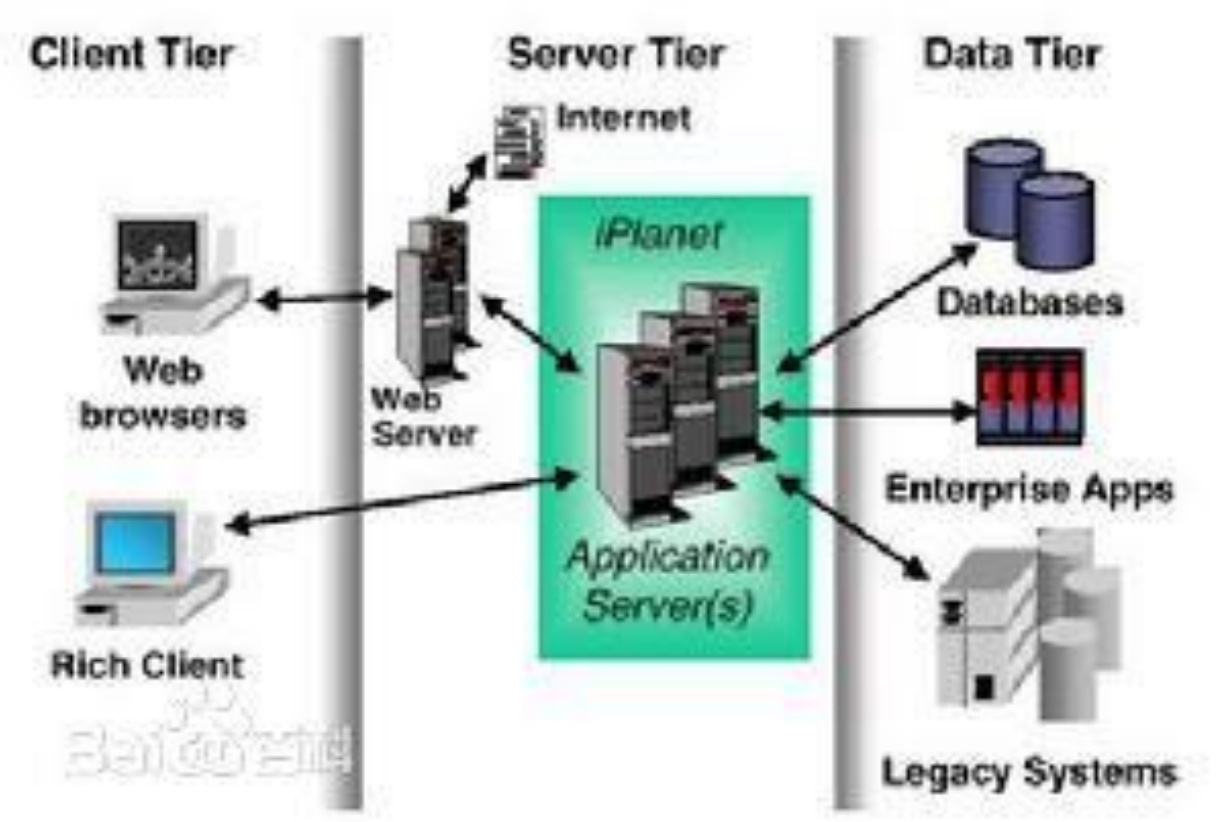
练习：画出结构图



软件体系结构

- 软件体系结构一词在软件工程领域中有着广泛的应用，但迄今为止还没有一个被大家所公认的定义。
- **Mary Shaw和David Garlan认为软件体系结构是软件设计过程中的一个层次，这一层次超越计算过程中的算法设计和数据结构设计。软件体系结构问题包括总体组织和全局控制、通讯协议、同步、数据存取，给设计元素分配特定功能，设计元素的组织，规模和性能，在各设计方案间进行选择等。**

- 软件体系结构不是软件结构。之所以称之为体系结构，表明其描述的不是一个软件元素，而是一组软件元素之间的组织和协作关系，并且这些软件元素可分布在不同物理位置上，通过网络协议相互通信。



软件体系结构的概念

软件体系结构和软件架构对应的英文单词都是“Software architecture”，即它们是意义完全相同的两个中文单词用语。在使用它们时往往带有一种习惯上的差异，通常学术上用“软件体系结构”较多，在软件系统设计上用“软件架构”较多，如在软件公司里，有“软件架构师”的职位，但很难听到“软件体系结构师”的说法。

软件开发重点的转移

面向对象技术、可视化编成技术、组件技术，以及强大的软件工具的支持，使得原来实现起来很复杂、很耗费精力的软件实现技术变得非常简单，并且在某些环境下，可以不编程序代码，或编很少的程序代码，就可实现一个界面美观、可靠性高的软件。

另一方面，软件的规模和复杂性越来越大，软件运行环境越来越复杂，使得**软件开发的主要精力放在了软件运行基础环境的考虑、软件开发环境的配置、软件开发总体方案的制定、软件架构模式的选择、不同环境中软件元素间通信的实现、软件安装及运行环境的配置等事关全局的问题上。**

即软件设计已经从关心基本的结构和算法转移到了对宏观结构的认识上；从关注功能的实现转移到了对综合性能的要求上，软件架构的设计成了关注的重点。

与软件架构相关的概念—构件

广义上，构件是软件系统的结构块单元，是软件功能和承载体。所以，从系统的构成上看，任何在系统中承担一定功能、发挥一定作用的软件体都可以看成是构件。构件可以分为计算构件、数据构件和连接构件。

狭义上，把构成软件系统的结构块分为构件和连接件，而构件与连接件在一个软件系统中扮演的角色是可区分的。因此，狭义的构件指与系统设计目标（业务功能需求）对应的结构块（处理单元和数据单元），为狭义的构件之间协同运行穿针引线的结构块称为连接件。

构件与连接件的根本区别是：构件有领域业务处理的功能，连接件只起业务构件之间的中介作用

与软件架构相关的概念—构件

构件有不同粒度。一个构件可以小到只有一个过程（或函数），也可以大到包含整个应用程序。函数、例程、对象、类库、数据包、服务器、文件等都可以作为一个构件。

对应一个软件架构实例来说，用什么粒度的构件来描述软件架构呢？应该从软件“系统”层面上可以把软件分成多少个逻辑上相对独立的计算单元（子系统）。每一个相对独立的计算单元就是一个构件。如果每个子系统又是一个高度独立的系统，又可以以它作为观测的“系统”对象，将其分解为若干个逻辑上相对独立的计算单元，每个计算单元就是一个构件。

在不同设计环境中，构件可表现为控件、组件、表、实体、包、设计模式、框架等。

与软件架构相关的概念—连接件

连接是构件和构件之间建立和维持行为关联和信息传递的途径。连接包括：实现机制和信息交换协议。

1、实现机制

硬件层：过程调用、中断、存储、栈等

基础控制描述层：过程调用、中断/事件、流、文件、网络等。

资源管理调度层：进程、线程、共享、同步、并行、事件、异常、远程调用等。

高级抽象层：管道、解释器、转换器、浏览器、组件/中间件、C/S、B/S、ODBC等。

2、信息交换协议

信息交换协议是连接的规约，是实现有意义连接的保证。最简单的连接是过程调用，过程调用的实现也有协议，如接口参数的顺序和类型、如何把参数放到获得数据的一方能够取得到的地方等。

与软件架构相关的概念—连接件

连接件的设计已经成为软件开发的关键技术。

连接件的设计要考虑的因素包括不同涉众的需求、构件的形式、系统的性能、系统的可维护性、系统的安全性、系统的可靠性等。构件可以以多种形式出现。对于集中数据管理的分布式应用系统，共享后台数据库的不同构件之间的连接件。相互独立的应用系统之间的交互可以用Web Service组件、消息中间件实现连接是目前广泛采用的技术。分布式系统采用选定的机制和协议实现连接。

与第三方认证的连接、与银行交费系统的连接、与税务系统的连接，都与系统的安全性、涉众利益直接相关，技术非常复杂。

C/S、B/S模式的构件之间的连接，都是通过中间件和协议实现。

软件架构设计的关注点

软件设计的首要任务是实现系统的功能需求，功能不能满足用户需求的软件无论如何也算不上好软件。同时软件设计必须对软件的非功能要求提出解决方案，使开发的软件满足那些非功能要求，不能满足非功能要求的软件也不是一个好软件。简单地说，不能满足功能要求的软件是一个不能用的软件（不能解决用户关注的业务问题），不能满足非功能要求的软件是一个不好用的软件。

在以计算机网络和**Internet**为计算环境的今天，仅仅满足软件的功能要求是远远不够的。可以设想，如果学生学籍管理系统只由一台微机来实现，或者只能由连接相邻几个房间的局域网来实现，将是什么场景？

软件体系结构设计的关注点是用户的非功能需求。软件的非功能需求是软件结构设计的驱动力。

关键需求决定架构

关键需求决定架构，其余需求验证架构，是架构设计应遵循的基本策略。

影响软件架构设计的关键需求包括功能需求、质量（属性）需求、商业需求三类。

关键的功能指影响系统成败的功能。如考试系统的在线考试功能是最关键的功能，其它功能无论多么完善，如果在线考试功能过不了关，就是一个不合格的考试系统。

有的系统围绕关键性能进行架构设计。电信系统、考试系统的性能是最关键的质量特性。**7*24**小时的不间断运行的系统，短暂停机就可能对生产造成重大影响。

商业需求指软件系统开发和应用方面的商业考虑，它关注从客户群、企业现状、未来发展、预算、立项、开发、运营、维护在内的整个软件生命周期涉及到的商业因素，包括了商业层面的目标、期望和限制。系统开发预算会影响对技术的选择。客户群对架构的影响，法律法规会影响系统的可扩展性、可修改性、可维护性。