体系结构的设计

- 建立一个软件系统的总体结构

目标

- 软件的体系结构设计为什么很重要
- 可能用于系统体系结构的不同模型
- 一系列不同类型的软件体系结构
- 领域相关的体系结构模型怎样成为产品线体系结构的基础

内容

- 系统构成
- 控制模型
- 模块化分解
- 领域相关的体系结构

软件结构

- 识别出组成系统的子系统，并建立子系统控制和通信的框架的过程，叫做体系结构设计。
- 该设计过程的输出结果是软件体系结构描述文档

体系结构设计

- 系统设计过程的早期阶段
- 描述和设计过程之间的连接
- 经常与一些描述活动并行完成
- 包括识别出主要的系统组件和它们之间的通信

清晰的体系结构的好处

- 项目相关人员之间的沟通
  - 可以作为项目相关人员之间讨论的焦点
- 系统分析
  - 使得分析系统能否满足其非功能需求成为可能
- 大规模复用
  - 体系结构能在具有相似需求的系统之间互用

体系结构设计过程

- 系统结构化
  - 将系统分解成一系列基本子系统，并识别出子系统之间的通信
- 控制建模
  - 建立系统各部分之间控制关系的模型
- 模块分解
  - 把每个识别出来的子系统进一步分解成模块

子系统和模块

- 一个子系统独立构成系统，不依赖其他子系统提供的服务
- 一个模块通常是一个能提供服务给其他组件的系统组件。通常不被看成是一个独立的系统。

体系结构模型

- 在设计过程中会产生不同的体系结构模型
- 每个模型代表了体系结构的不同观察角度

体系结构模型

- 静态结构模型表示主要的系统组件
- 动态过程模型表示了系统的过程结构
- 接口模型定义了子系统接口
- 关系模型给出如组件间数据流这样的关系

体系结构样式

- 体系结构模型应符合通用的体系结构模型或样式
- 通晓这些样式可以使得系统体系结构定义变得简单
- 然而，多数大型系统是异构的，无法遵循单一的体系结构样式

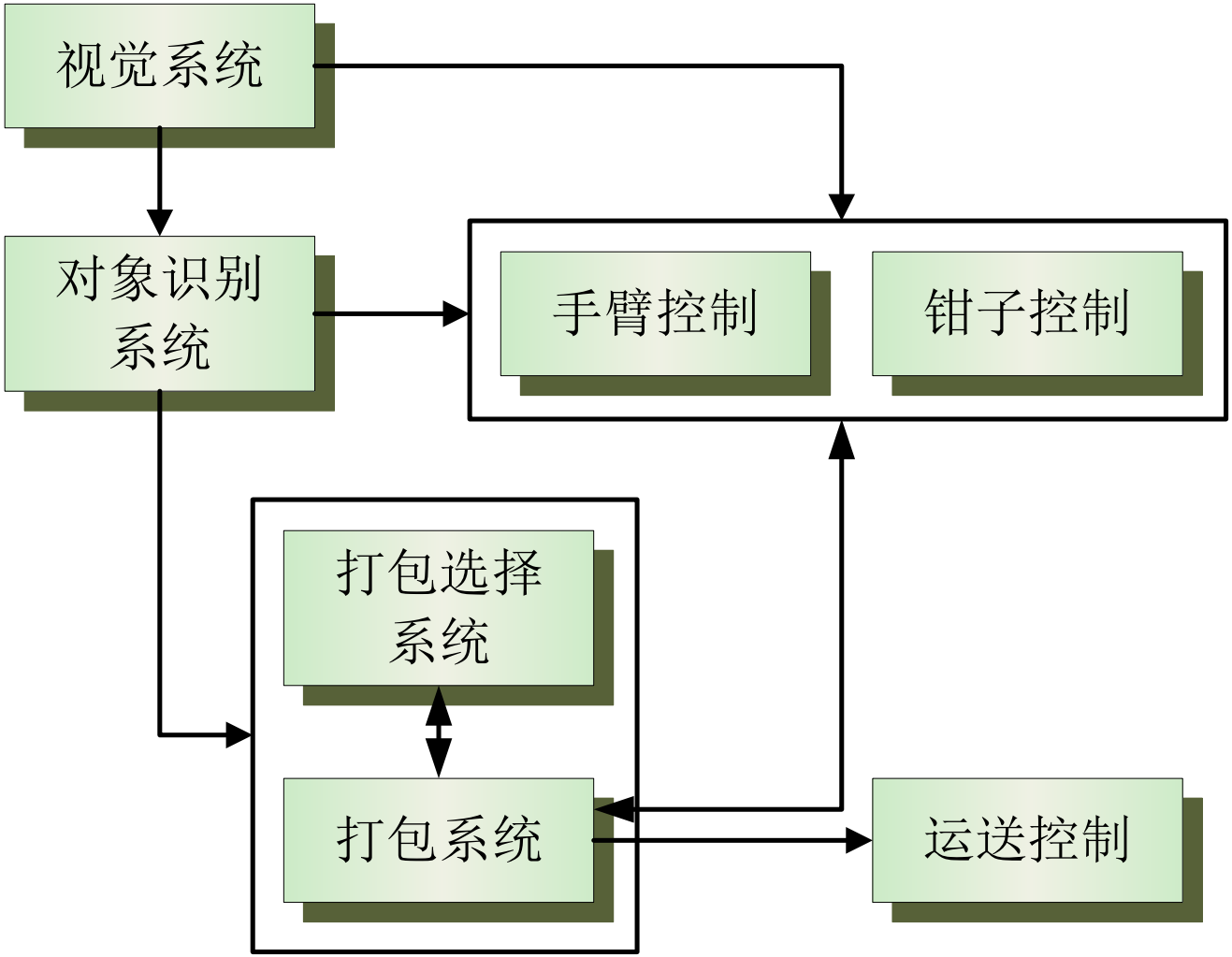
体系结构的特征

- 性能
  - 定位操作以尽量减少子系统间的通信
- 保密性
  - 使用分层结构，最关键的资源放在内层
- 安全性
  - 隔离安全性要求的组件
- 可用性
  - 在体系结构中采用冗余组件
- 可维护性
  - 使用小粒度、独立的组件

系统构成

- 将系统分解成互相作用的子系统
- 体系结构设计通常用一个方块图表达，代表了系统结构的概貌
- 还可以提出更专门化的模型用来描述子系统是如何共享数据、如何分布以及如何彼此交互的

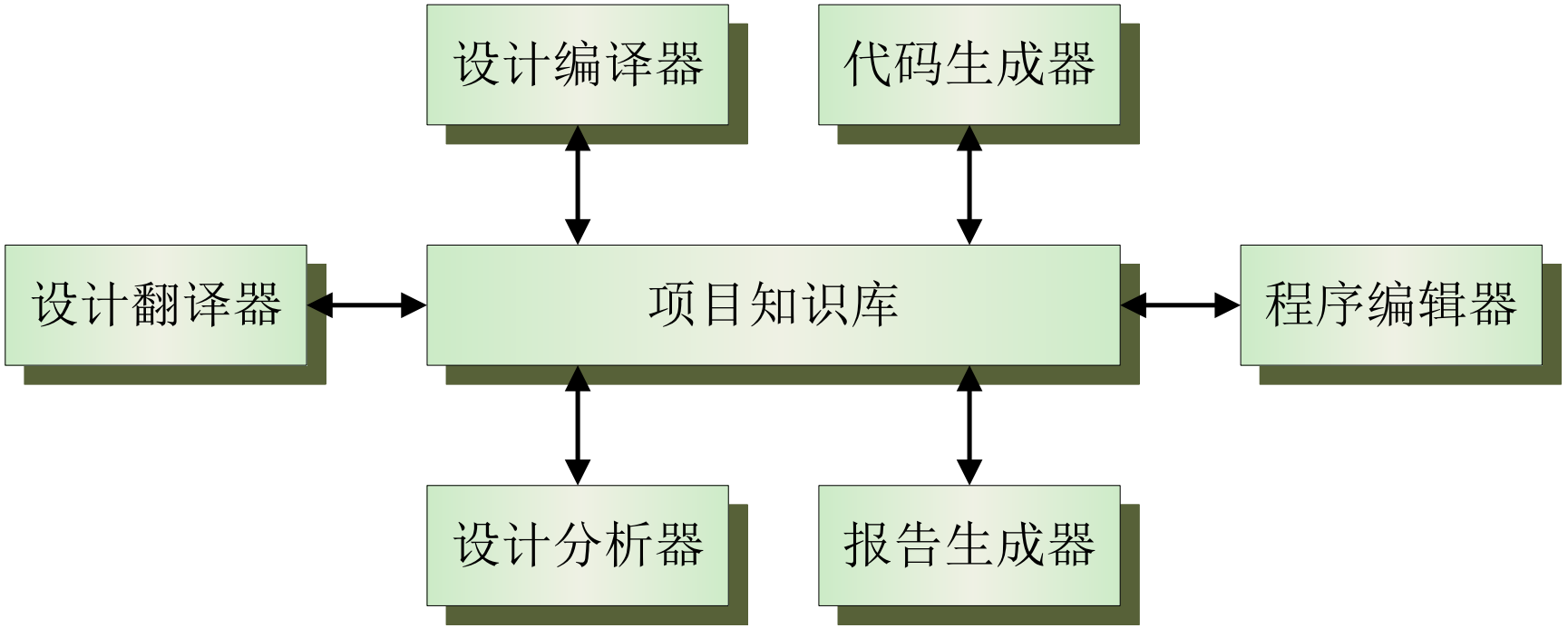
打包机器人控制系统



容器模型

- 子系统要交换数据，这可以有两种方法:
  - 共享数据存放在一个中央数据库或者是容器中，可以被所有子系统访问
  - 每个子系统维护自己的数据库，显式地将数据传送给其他子系统
- 当共享大量的数据时，容器模型是最常用的

CASE工具集体系结构



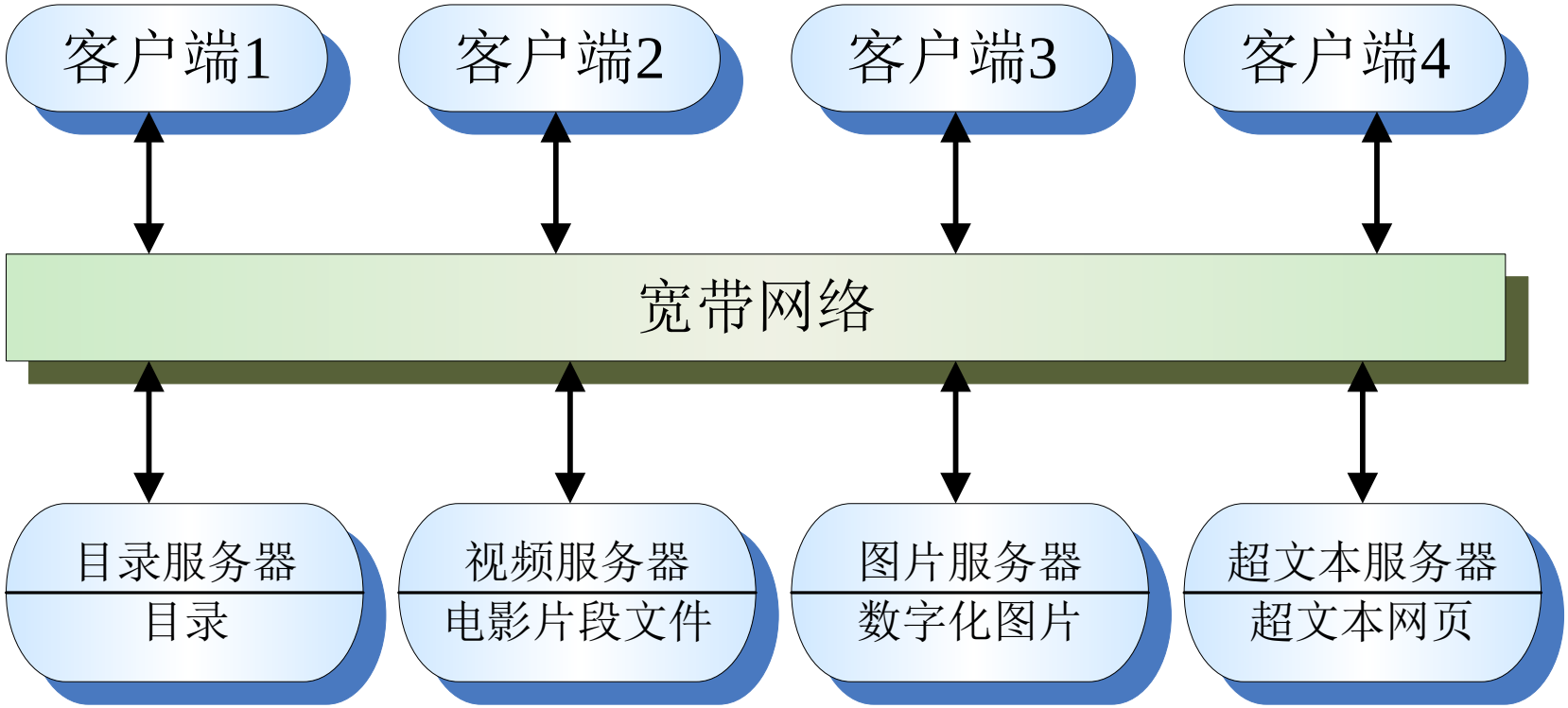
容器模型的特点

- 优点
  - 共享大量数据的有效方法
  - 子系统不需关心数据是如何进行集中管理的，如备份、加密等
  - 某些活动（备份、保密型、访问控制、恢复）等集中进行
  - 通过容器模型可以清晰的看出共享模型
- 缺点
  - 子系统要与容器数据模型一致。不可避免的需要妥协。
  - 数据进化比较困难和昂贵
  - 对特定的管理政策缺乏不同的范围
  - 数据分布比较困难

客户端-服务器体系结构

- 说明数据和处理是如何在一个范围内的组件间分布的分布式系统模型
- 一组提供特定服务的单机服务器，如打印服务、数据管理服务等。
- 一组向服务器请求服务的客户机
- 一个连接客户机和服务器的网络

电影和图片库系统的体系结构



客户-服务器模型的特点

- 优点
  - 数据的分发简单明了
  - 有效利用网络系统。可以使用更低廉的硬件。
  - 容易增加新的服务器或升级已有服务器
- 缺点
  - 没有共享数据模型，所以子系统使用不同的数据组织。数据交换可能效率不高
  - 各个服务器存在冗余的管理
  - 没有名字和服务的集中登记，难于发现都有哪些服务器以及服务

抽象机模型

- 用来建立子系统的接口模型
- 将系统组织成一系列的层次（或者叫抽象机），每一层提供一组服务
- 支持不同层中的子系统的增量开发。当一个层的接口改变时，只是相邻层受到影响。
- 然而，用这种方式构建系统通常比较困难

版本管理系统

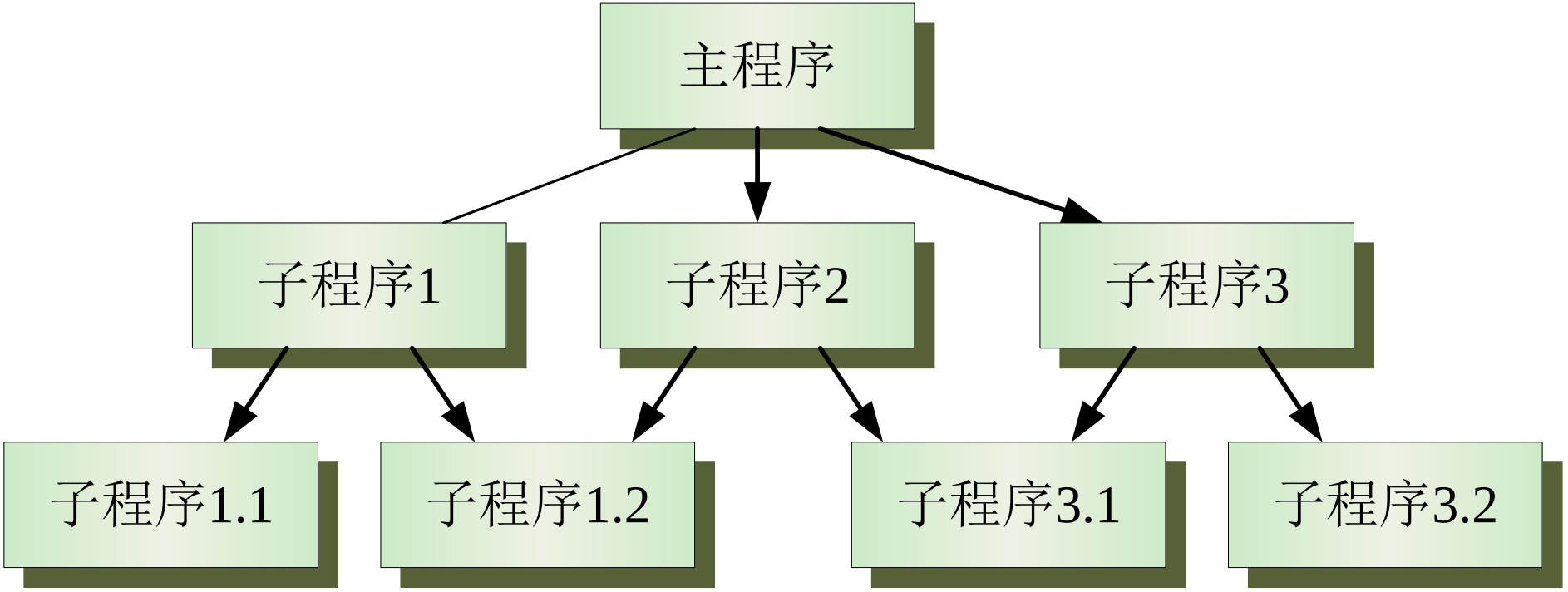
控制模型

- 关注子系统间的控制流。不同于系统分解模型。
- 集中式控制
  - 一个子系统全面负责控制，负责启动和终止其它子系统。
- 基于事件的控制
  - 每个子系统都能对来自别的子系统或系统环境的外部事件，作出响应。

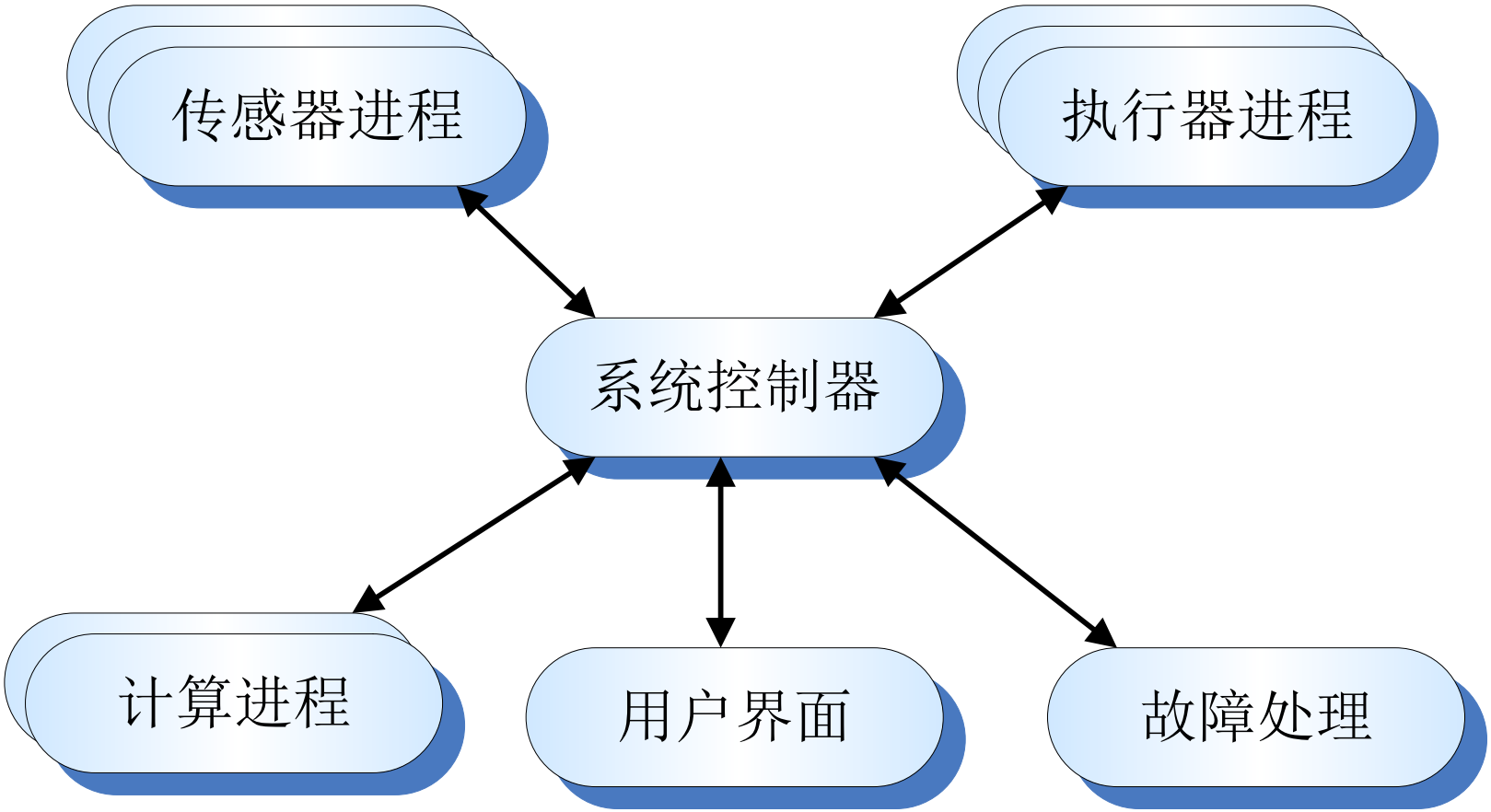
集中式控制

- 一个控制子系统负责管理其它子系统的执行
- 调用-返回模型
  - 自上而下的子过程模型，控制从子程序层的顶端开始，向下移动。适用于顺序系统。
- 管理器（者）模型
  - 适用于并发系统。一个系统组件控制其它系统过程的停止、开始和协调。可以在顺序系统中用case语句实现。

调用-返回模型



实时系统控制



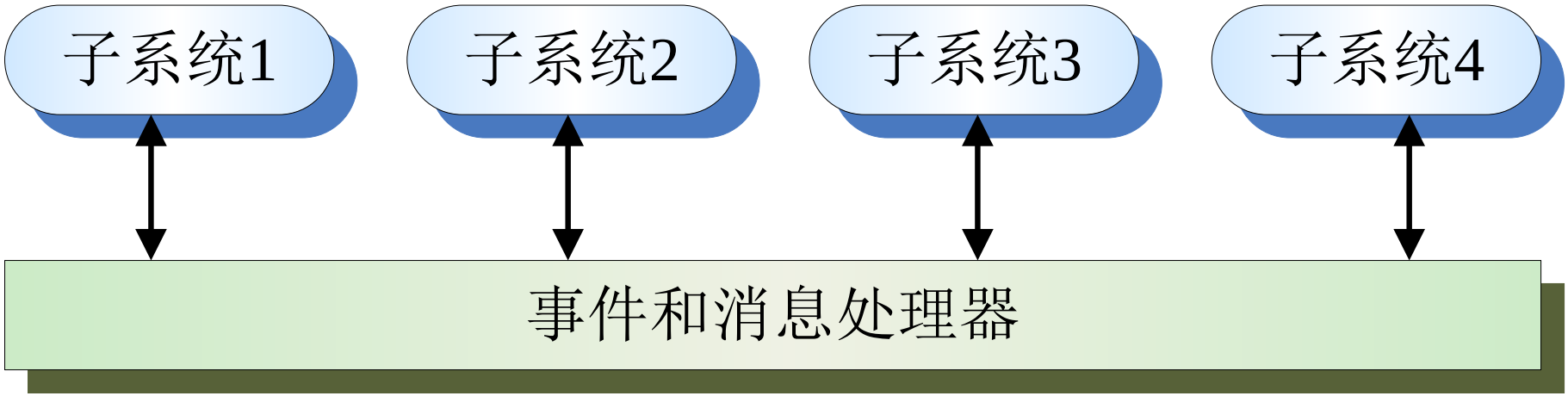
事件驱动系统

- 受外部产生事件的驱动，产生事件的时刻与处理该事件的子系统无关
- 两个主要的事件驱动模型
  - 广播模型. 一个事件向所有子系统广播，包括任何一个能够处理该事件的子系统
  - 中断驱动模型. 应用于实时系统，中断被中断处理器探测到，再被传递给其它负责处理中断的组件
- 其它事件驱动模型包括电子数据表和产品系统

广播模型

- 对于将子系统集成到网络中不同的计算机是非常有效的。
- 子系统注册了特定的事件。当事件发生时，控制传递到能够处理该事件的子系统。
- 控制策略不嵌入到事件和消息处理器中。由子系统根据事件来决定。
- 然而，子系统不知道是否以及什么时候要处理一个事件

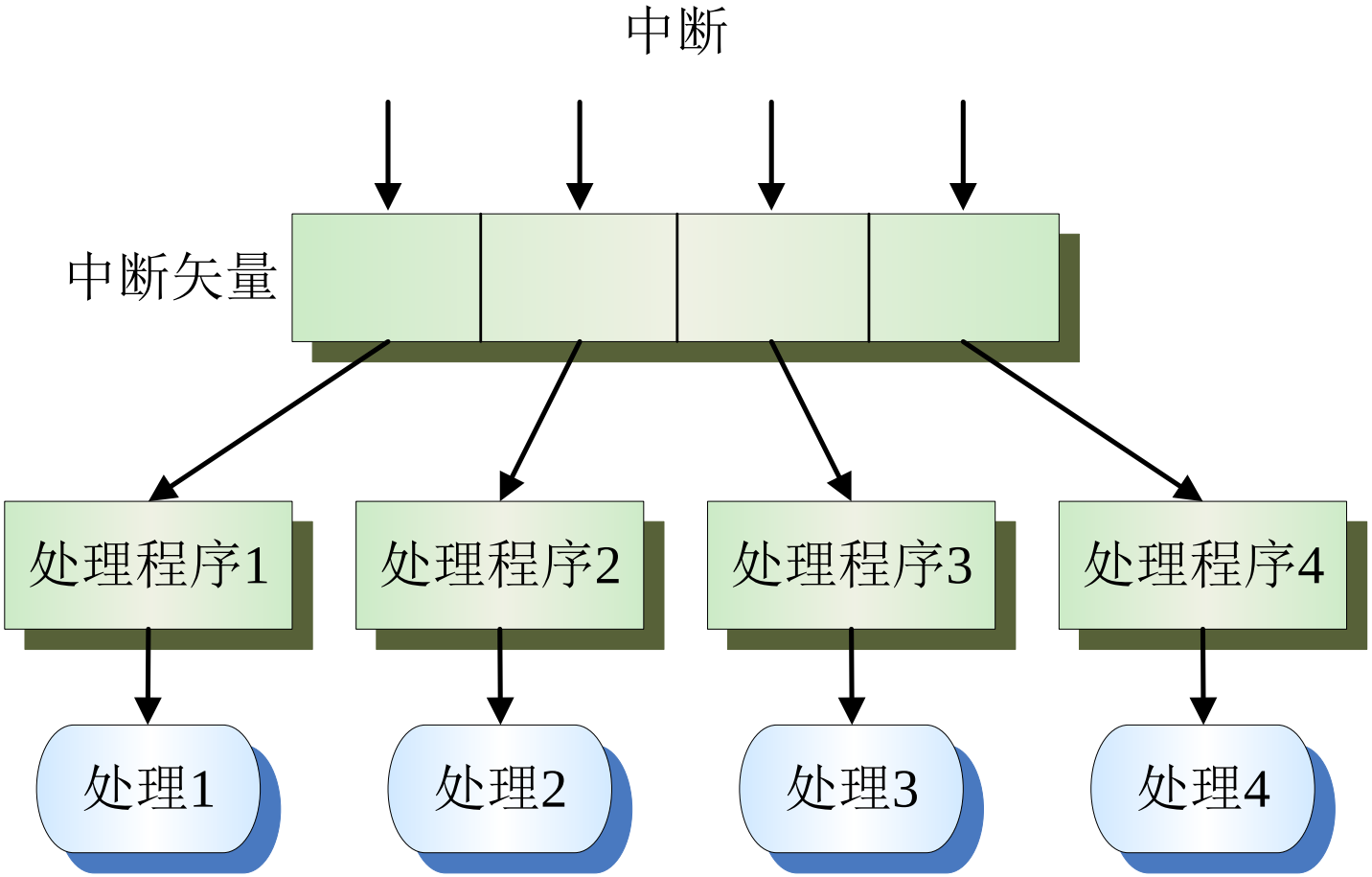
选择的广播



中断驱动系统

- 用于实时系统，需要对事件作出快速响应
- 有已知的中断类型，以及为每个中断类型定义的中断处理
- 每种类型都关联到一个存储单元，一个硬件开关将中断转到它的处理程序
- 可以快速响应，但是编程复杂且难于验证

中断驱动的控制



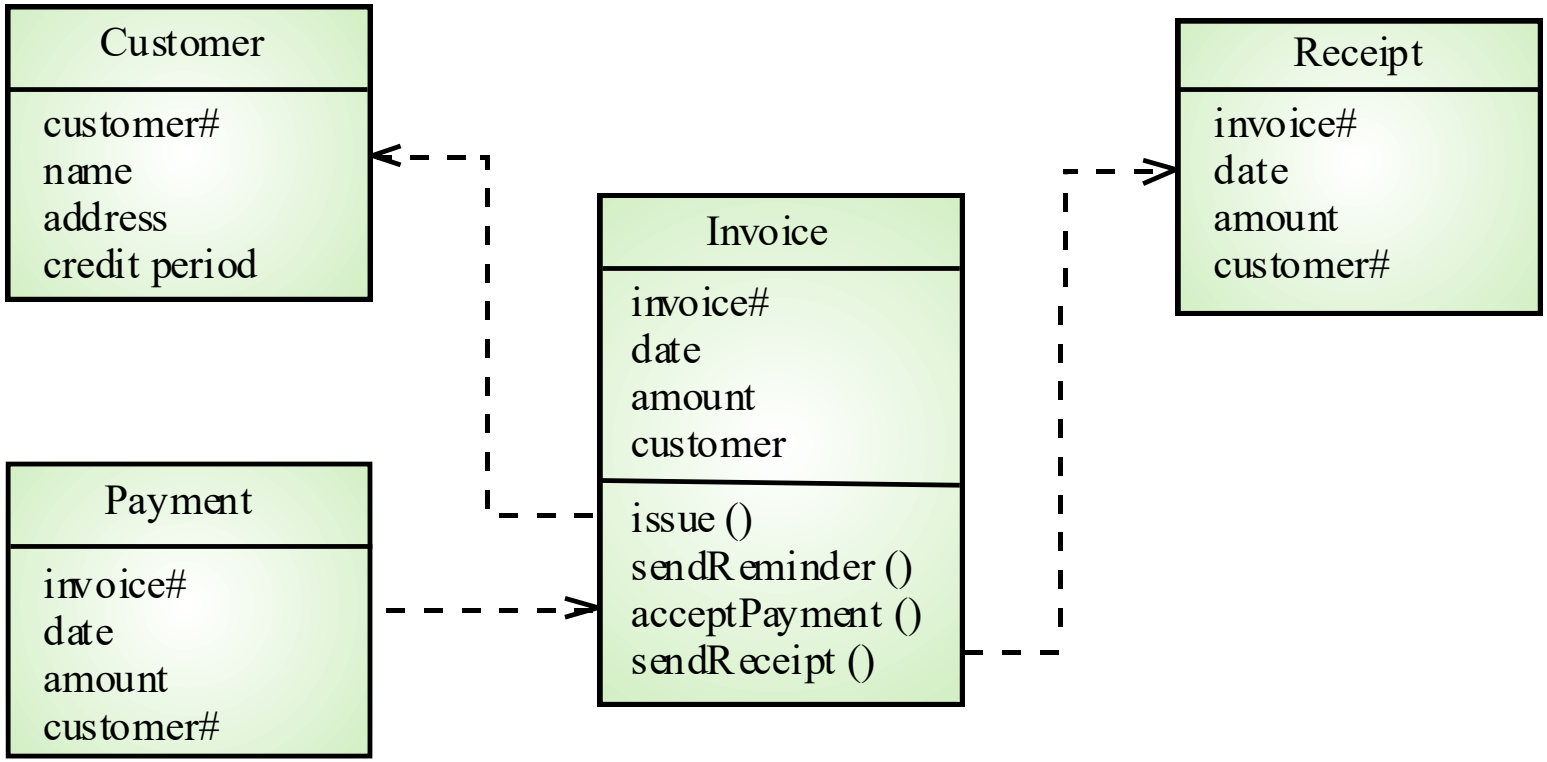
模块分解

- 另一个结构层次上，子系统分解成模块
- 两个模块分解模型
  - 对象模型。系统分解成互相作用的对象。
  - 数据流模型。系统分解成功能模块，这些功能模块将输入转化为输出。也叫管道模型。
- 如果可能，设计者应避免不太成熟的并发设计。可先做模块分解，关于是否执行并发可以延迟到模块开发时决定。

对象模型

- 将系统分解成一组松散的对象，以及良好定义的接口
- 面向对象分解关系到识别对象类，它们的属性和操作
- 当实现的时候，对象从这些对象类产生，用一些控制模型协调对象的操作

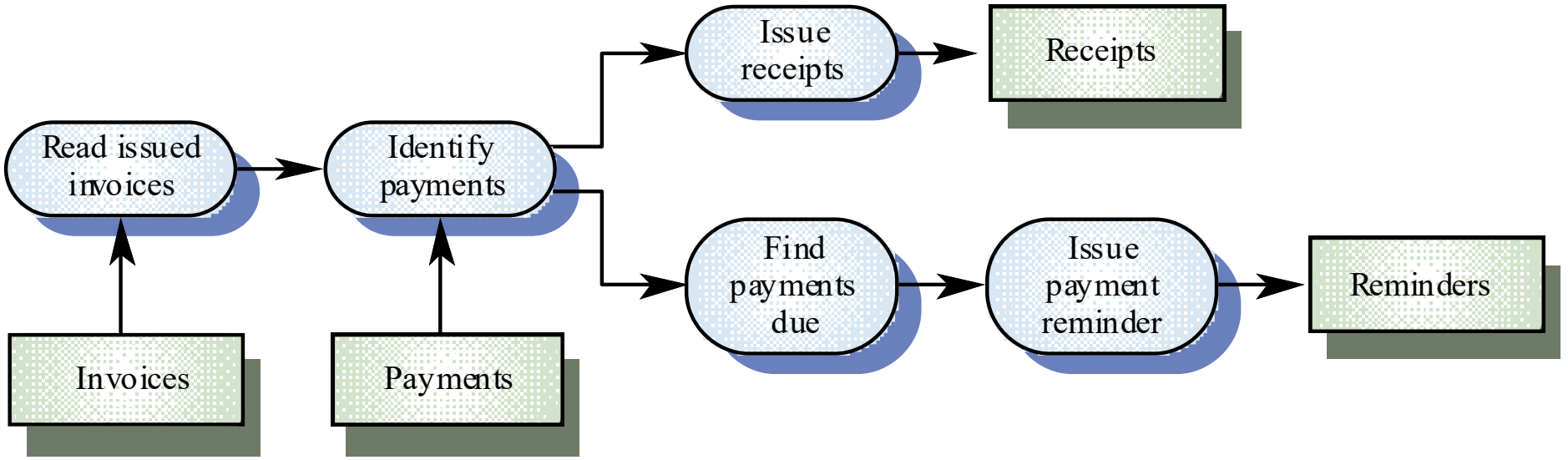
发票处理系统


Customer
customer#
name
address
credit period
Receipt
invoice#
date
amount
customer#
Invoice
invoice#
date
amount
customer
issue ()
sendReminder ()
acceptPayment ()
sendReceipt ()
Payment
invoice#
date
amount
customer#

数据流模型

- 功能转换处理其输入，产生其输出
- 也称为管道或过滤器模型 (在UNIX命令行环境中)
- 当转换是顺序的，这个方法就是批处理顺序模型，广泛应用于数据处理系统
- 对交互式系统不合适

发票处理系统


Read issued invoices
Identify payments
Issue receipts
Receipts
Find payments due
Issue payment reminder
Reminders
Invoices
Payments

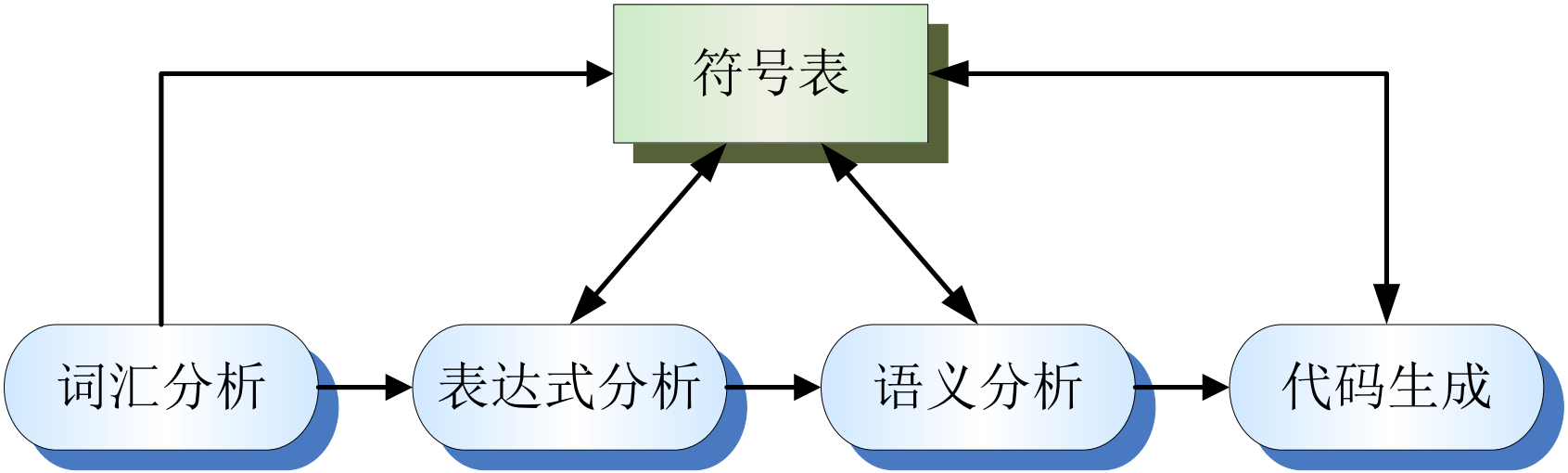
领域相关的体系结构

- 特定于某些应用领域的体系结构模型
- 两种领域相关的模型
  - 类模型，从许多真实系统中抽象出来，封装了这些系统的主要特征
  - 参考模型，更加抽象的理想化的模型，提供了关于系统类型的信息以及比较不同体系结构的手段
- 类模型通常是从下往上的模型；参考模型是从上往下的模型。

类模型

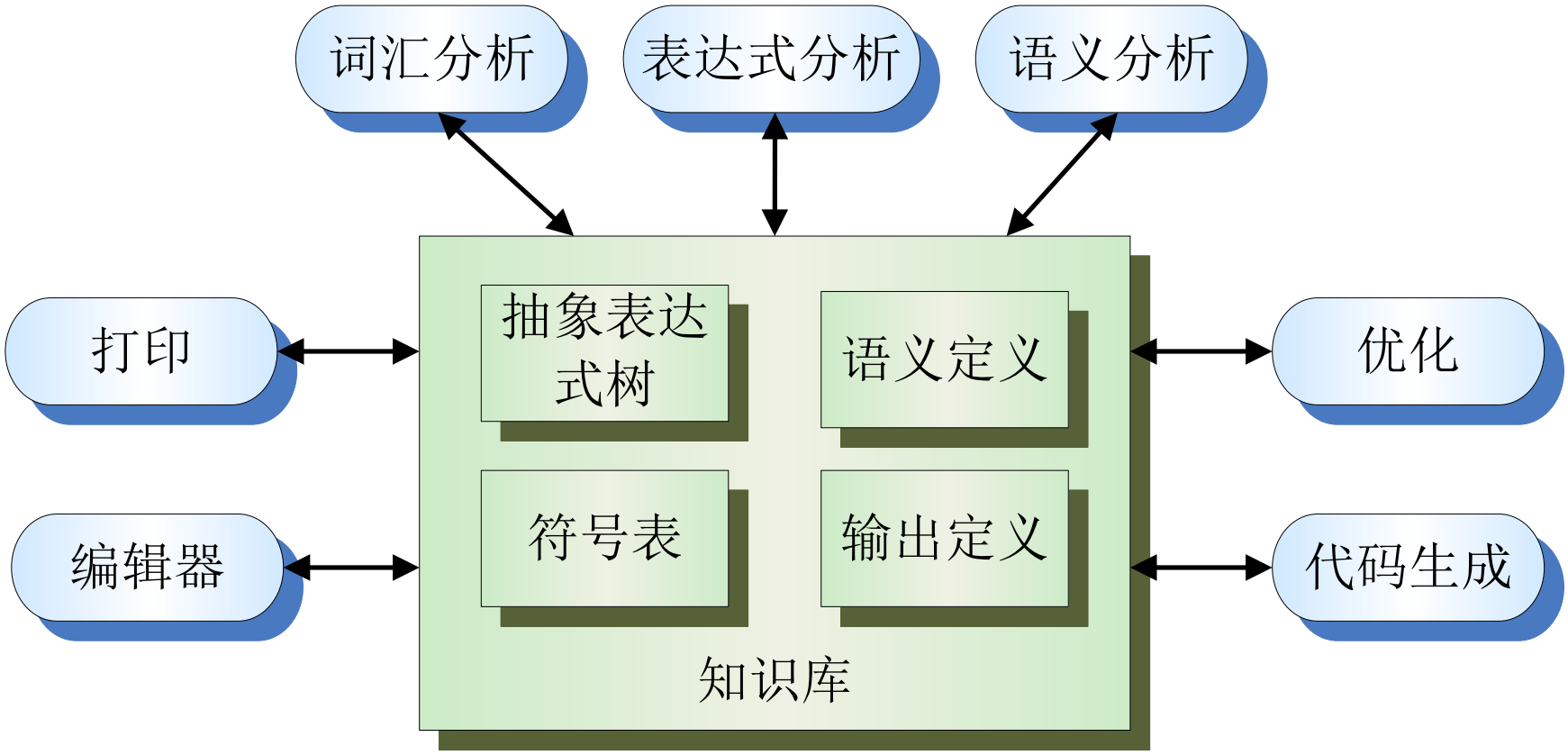
- 编译器模型是一个众所周知的例子，尽管在其它更加专门的应用领域里还有其它模型
  - 词汇分析器
  - 符号表
  - 语法分析器
  - 语法树
  - 语义分析器
  - 代码生成器
- 通用编译器模型可根据不同的梯形结构模型进行组织

编译器模型



编译器的数据流模型

语言处理系统

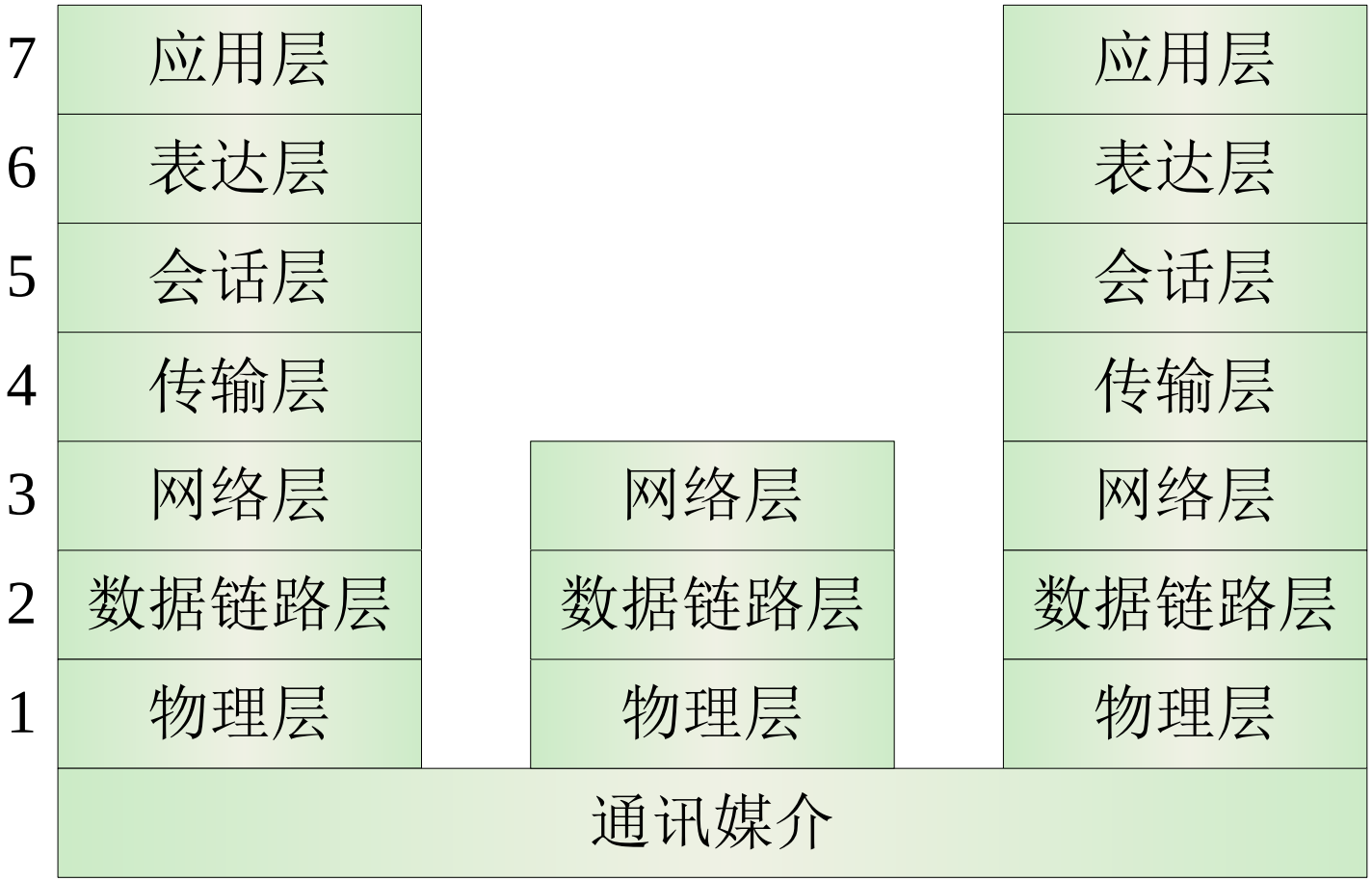


语言处理系统的容器模型

参考体系结构

- 参考模型源自对应用领域而不是一个存在系统的研究
- 用来作为一个系统实现的基础，或者不同系统的比较。它可以作为一个标准，系统可以用它来评估
- OSI模型是一个通信系统的分层模型

OSI参考模型


7
6
5
4
3
2
1
应用层
表达层
会话层
传输层
网络层
数据链路层
物理层
网络层
数据链路层
物理层
应用层
表达层
会话层
传输层
网络层
数据链路层
物理层
通讯媒介

要点

- 软件体系结构是负责导出结构化系统模型，控制模型和子系统分解模型
- 很少有大型系统遵从一个单一的体系结构模型
- 系统分解模型包括容器模型、客户机-服务器模型和抽象机模型
- 控制模型包括集中式控制模型和事件驱动模型

要点

- 模块化分解模型包括数据流模型和对象模型
- 领域相关的体系结构是对应用领域的抽象。领域相关模型可以是类模型，也可以是参考模型。