河北师范大学软件学院
Software College of Hebei Normal University

第十三章 管道过滤器

任课教师：武永亮
wuyongliang@edu2act.org

课程内容

- 体系结构的概念
- 体系结构的风格（分类）
- 数据流风格

课程内容

- 体系结构的概念
- 体系结构的风格（分类）
- 数据流风格

理解软件体系结构

- 软件体系结构= 软件 的 体系结构

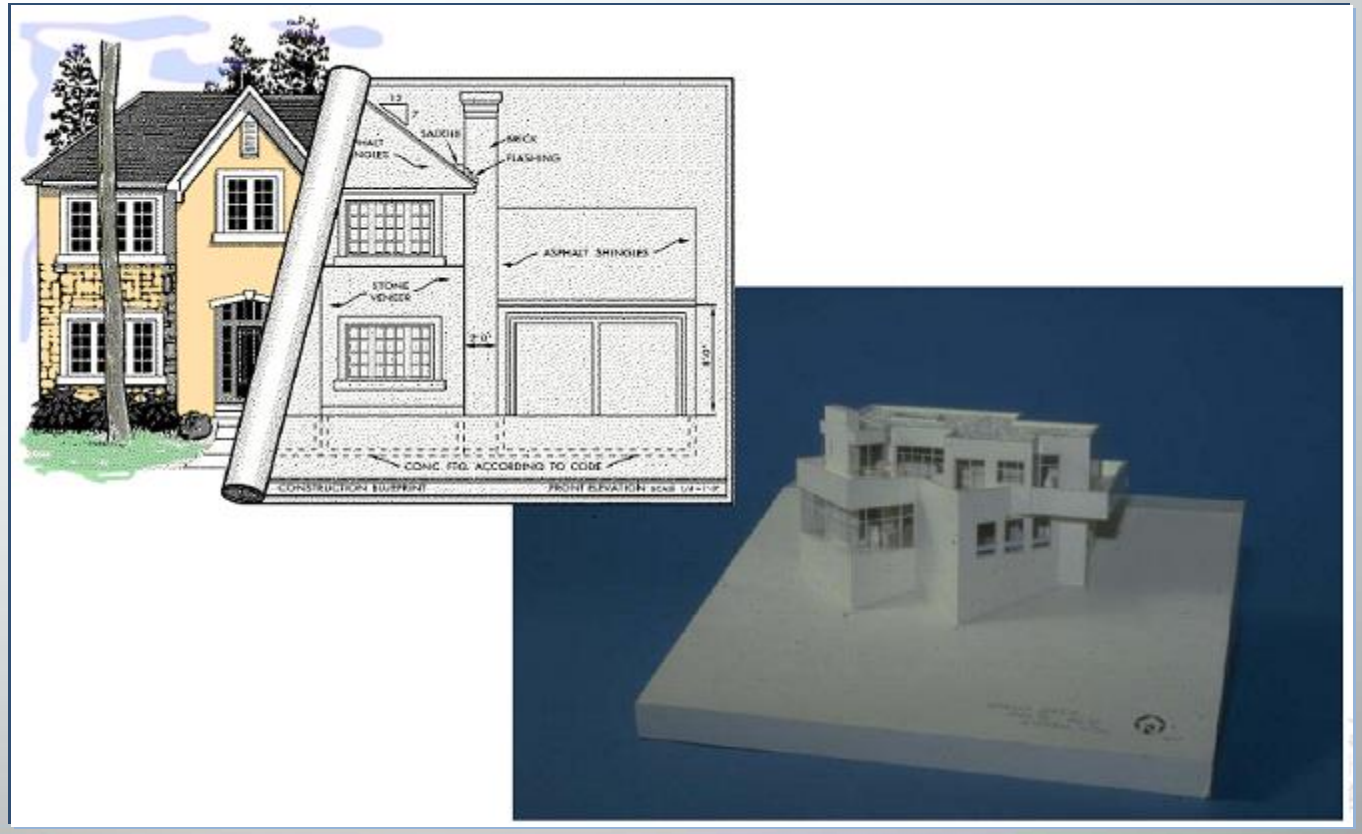
Software Architecture (SA) = Software’s Architecture (S’A)

学习软件体系结构，应首先弄清楚两个问题：
1 什么是“软件” (Software)？
2 什么是“体系结构” (Architecture)？

然后方可回答：
什么是“软件的体系结构” ？

什么是“体系结构”（Architecture）？

- 软件体系结构起源——建筑业
- 建立模型



优秀的体系结构设计

鸟巢

玛雅阿兹特克金字塔

瑞士保险公司大楼

什么是“体系结构”（Architecture）？

- 如何使用基本的建筑模块构造一座完整的建筑？
- 包含两个因素：
  - 基本的建筑模块
  - 建筑模块之间的粘接关系

结构设计师：
设计图纸

管理人员：
施工计划

施工人员：
建造建筑物

“体系结构”的共性

- 一组基本的构成元素—构件
- 这些要素之间的连接关系—连接件
- 这些要素连接之后形成的拓扑结构—物理分布
- 作用于这些要素或连接关系上的限制条件—约束
- 质量—性能

component

connector

deployment

constraint

performance

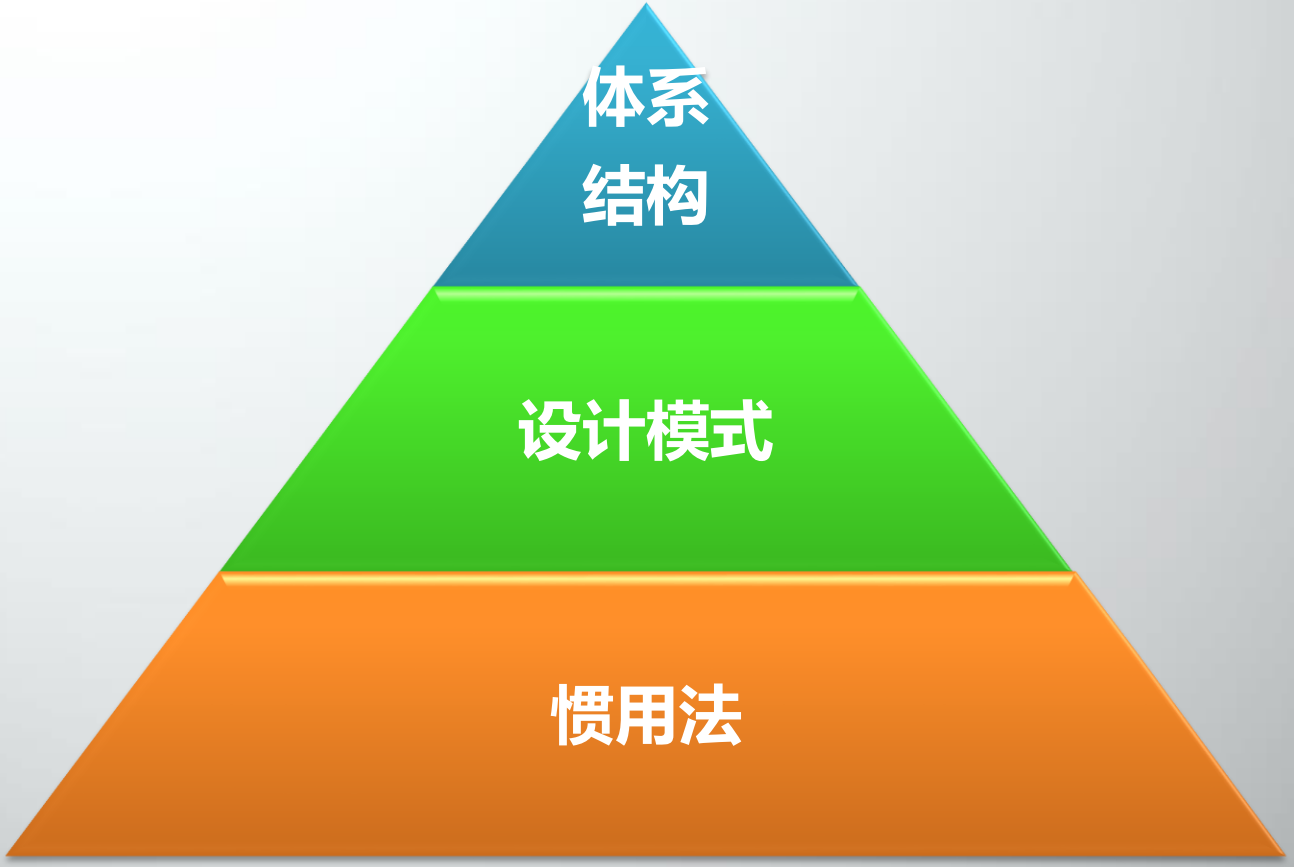
类推“软件体系结构”

- 构件：各种基本的软件构造模块(函数、对象、模式 等)；
- 连接件：将它们组合起来形成完整的软件系统
- 物理分布
- 约束
- 性能

体系结构 = 构件 + 连接件 + 约束

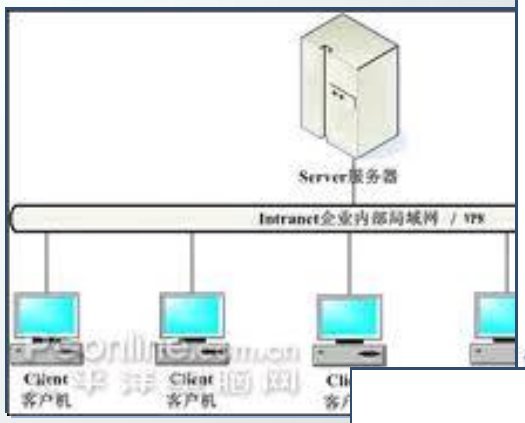
Architecture = Components + Connectors + Constraints

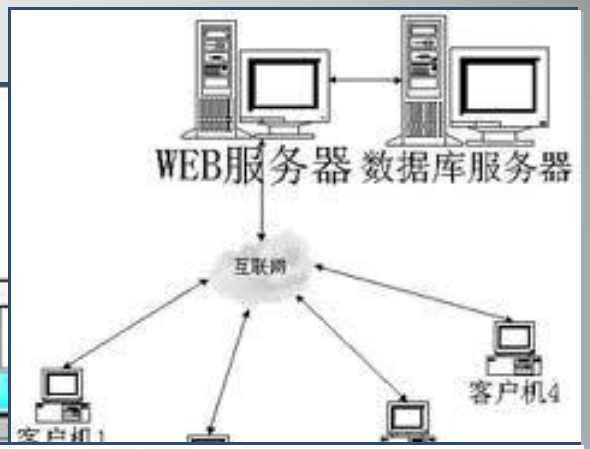
体系结构与设计模式的关系

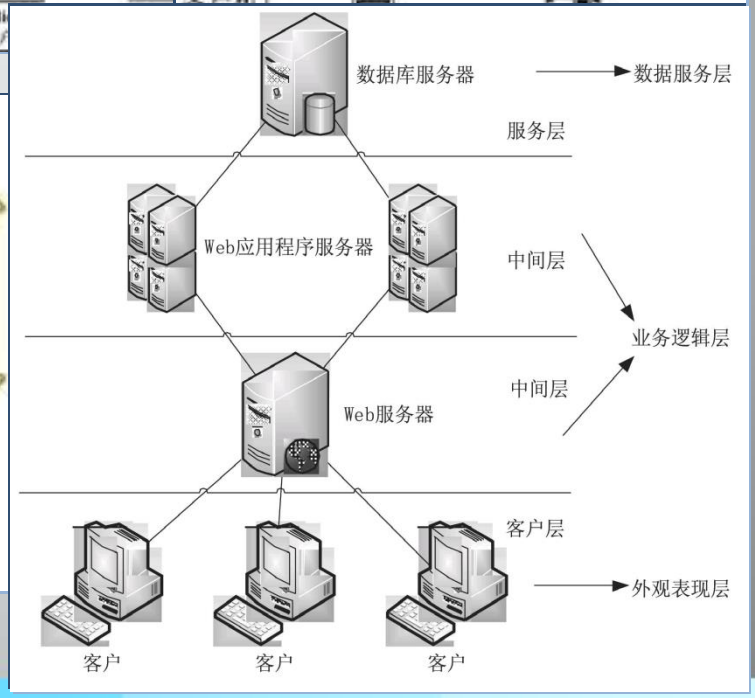


我们熟悉的软件体系结构

- Client/Server
- Browse/Server
- Three-tier
- Distributed
- ……







课程内容

- 体系结构的概念
- 体系结构的风格（分类）
- 数据流风格

软件体系结构风格分类

- 经典SA风格
- 其他常用SA风格
- 异构（复合）SA风格

经典SA风格

分类	典型风格
数据流风格	批处理序列,管道和过滤器
调用/返回风格	主程序/子程序,面向对象,层次结构
独立构件风格	进程通信,事件系统
虚拟机风格	解释器,基于规则系统
仓库风格	数据库系统,黑板系统

其他常用SA风格

分类	典型风格
通讯类	SOA, Message Bus
部署类	Client/Server,3-Tier,N-tier,Peer-to-peer(P2P),Grid Computing, Cloud Computing
领域类	Search-oriented architecture,Web2.0
交互类	MVC, Separated Presentation
结构类	Plugin, Shared nothing architecture

课程内容

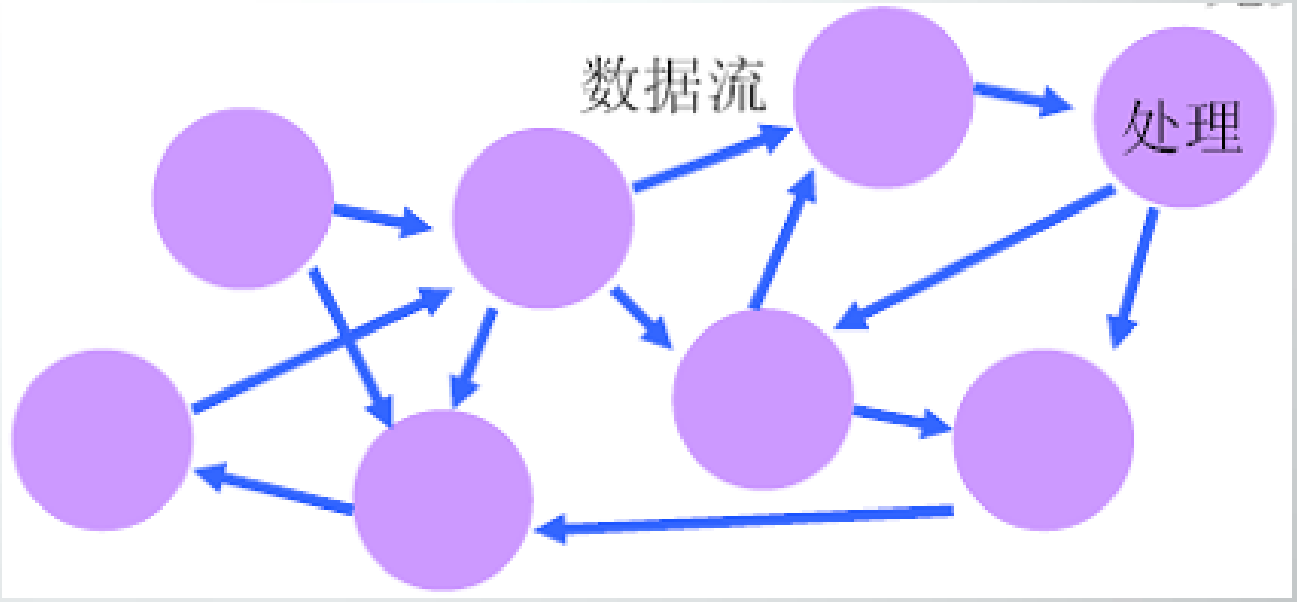
- 体系结构的概念
- 体系结构的风格（分类）
- 数据流风格

什么是数据流风格？

- 数据从一个处理单元流入到另一个处理单元，每经过一个单元就做一次转换。

注意：数据流风格不是某个过程的数据流图，它描述的是系统体系结构级别的设计。

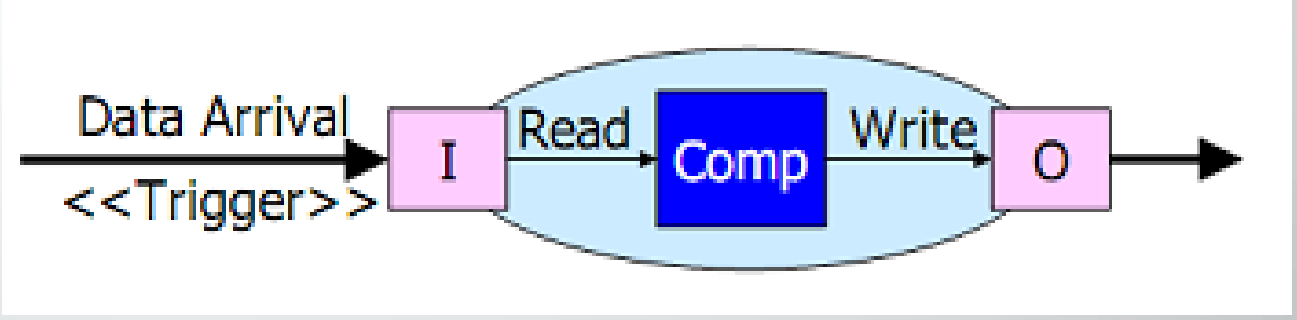
数据流风格的直观理解



处理：数据到达即被激活，无数据时不工作。

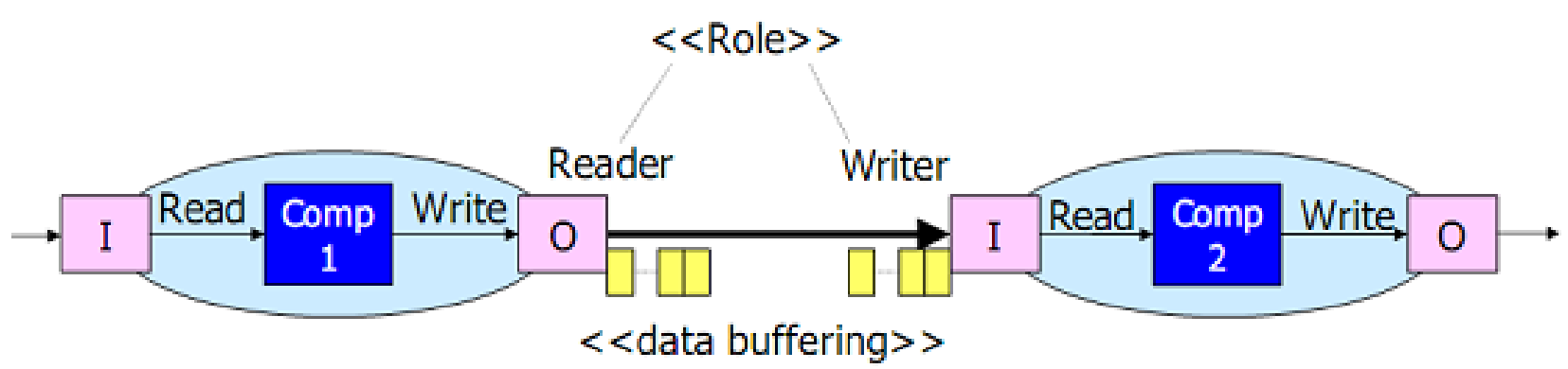
数据流风格的基本构件（Component）

- 基本构件：数据处理
- 构件接口：输入端口和输出端口



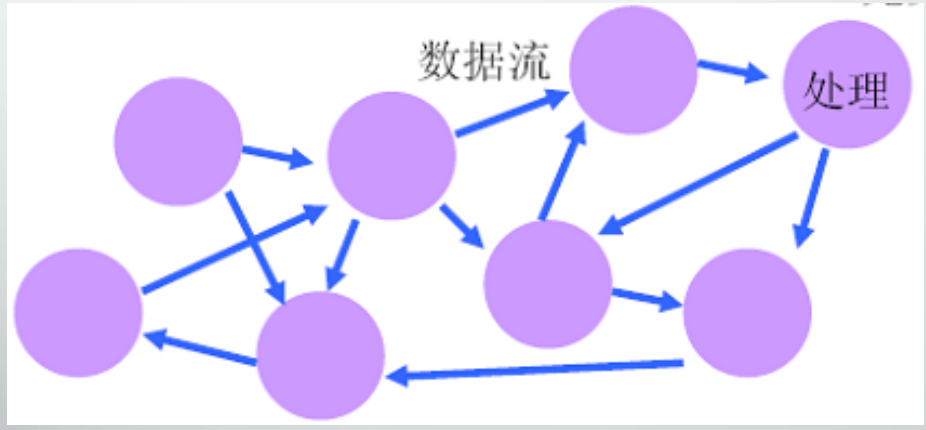
数据流风格的连接件（Connector）

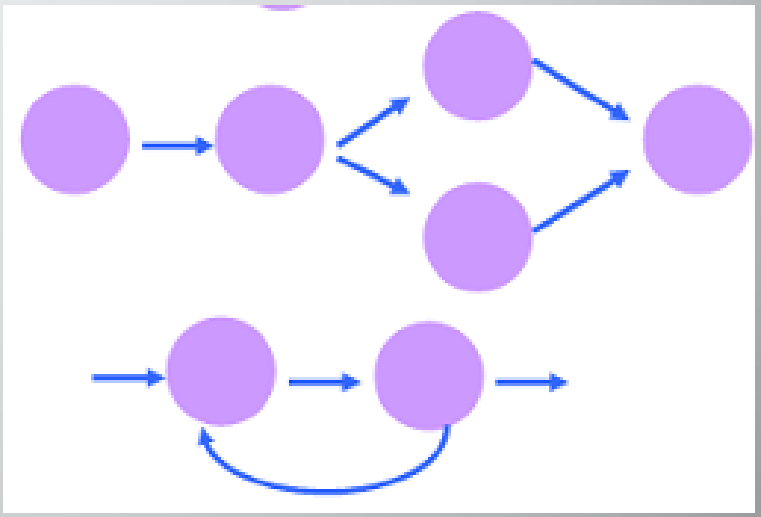
- 连接件：数据流



数据流风格的拓扑结构（Topology）

- 任意拓扑结构的图
  - 一般来说，数据的流向是无序的
  - 我们主要关注近似线性的数据流
  - 或在限度内的循环数据流

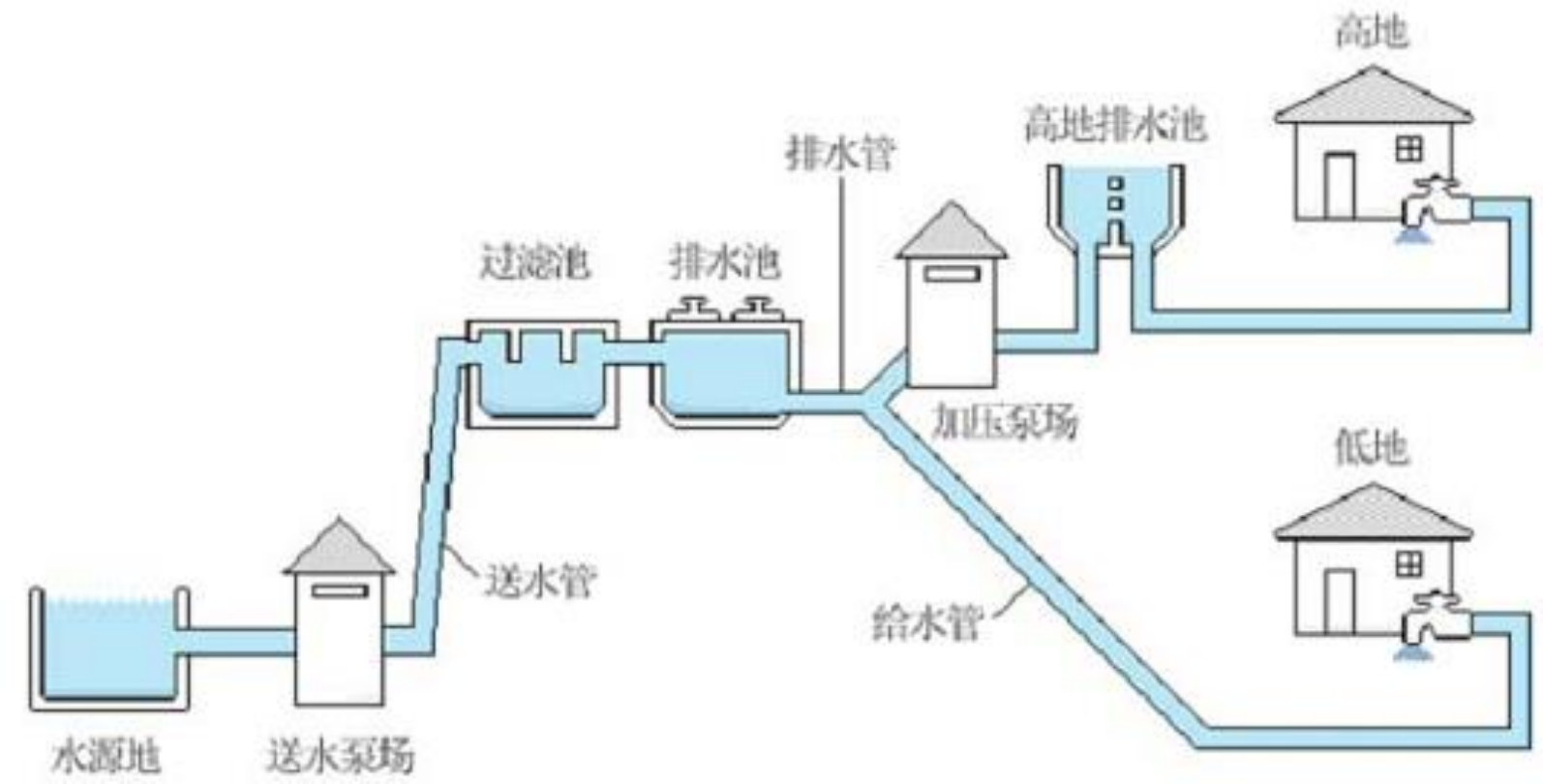




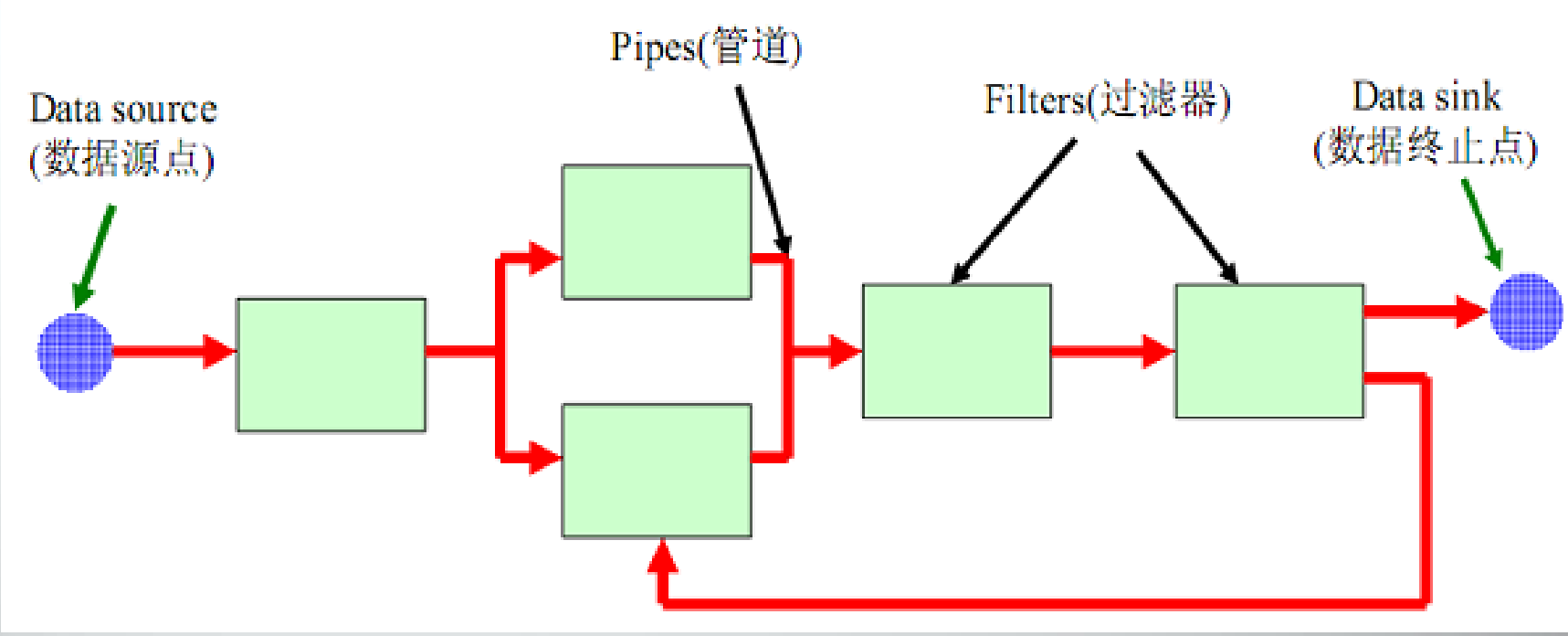
三种典型的数据流风格

- 管道/过滤器（Pipe-and-Filter）
- 批处理（Batch Sequential）
- 过程控制（Process Control）

管道与过滤器（Pipe-and-Filter）



管道-过滤器风格的直观结构

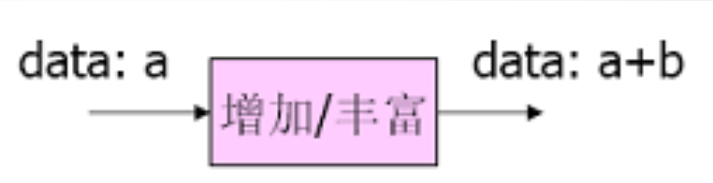


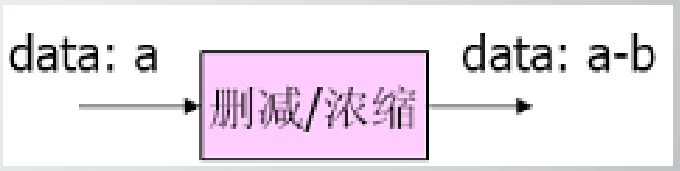
管道-过滤器风格的基本构成

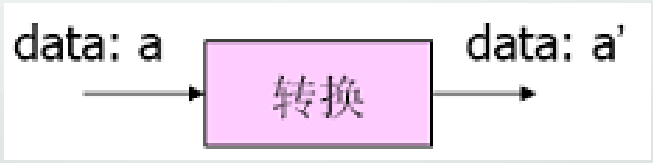
- 构件：过滤器，处理数据流
  - 一个过滤器封装了一个处理步骤
  - 数据源点和数据终止点可以看作是特殊的过滤器
- 连接件：管道，连接一个源和一个目的过滤器
  - 转发数据流
- 连接器定义了数据流图，形成拓扑结构

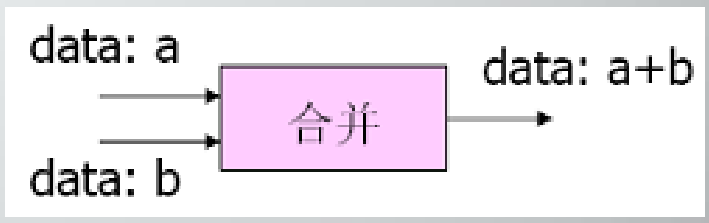
过滤器(Filter)

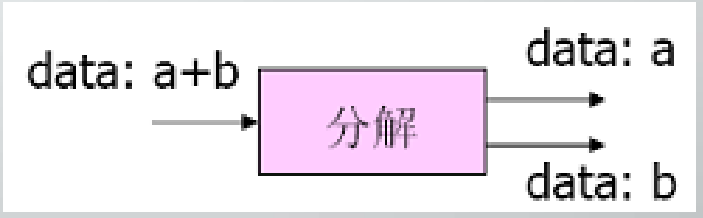
- 目标：将源数据变换成目标数据
- 过滤器对数据流的五种变换类型





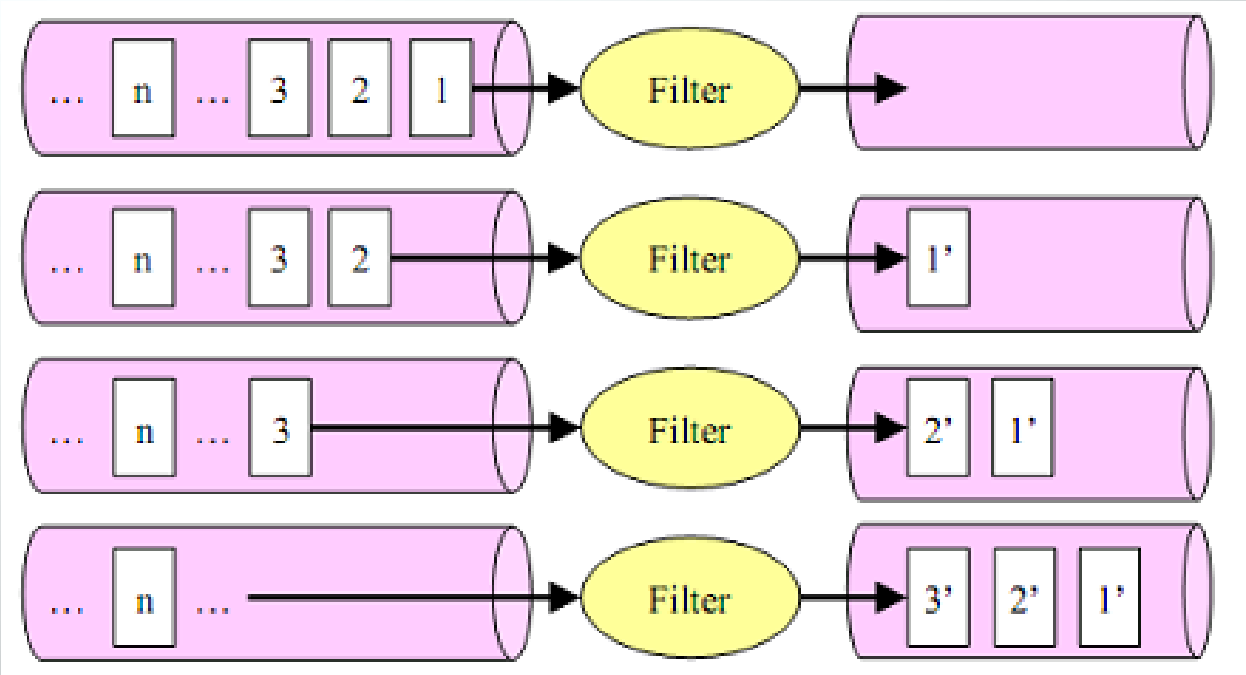






过滤器读取与处理数据流的方式

- 递增的读取和消费数据流
- 在输入被完全消费之前，输出便产生了。



过滤器的一些基本特征

- 无上下文信息
- 不保留状态
- 对其他过滤器无任何了解
- 可使用数据缓冲区临时保存数据流
  - 蓄水池

管道（Pipe）

- 作用：在过滤器之间传送数据
  - 单向流
  - 可能具有缓冲区
- 不同的管道中流动的数据流，具有不同的数据格式(Data format)
- 原因：数据在流过每一个过滤器时，被过滤器进行了丰富、精练、转换、融合、分解等操作，因而 发生了变化。

思考：是什么力量推动数据在管道中流动

- 数据流的分类：推式与拉式
  - 推式：前面的过滤器把新产生的数据推入管道
  - 拉式：随后的过滤器从管道中拉出所需数据
  - 推拉式：过滤器以循环的方式，从管道中拉出其输入数据，并将其处理产生的数据压入后续管道

过滤器的分类：主动与被动

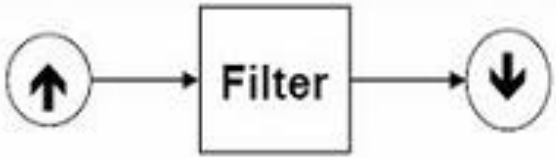
- 主动过滤器：
  - 具有pull/push类型的过滤器
- 被动过滤器：
  - 拉式策略
  - 推式策略
- 系统中至少有一个主动过滤器

主动过滤器

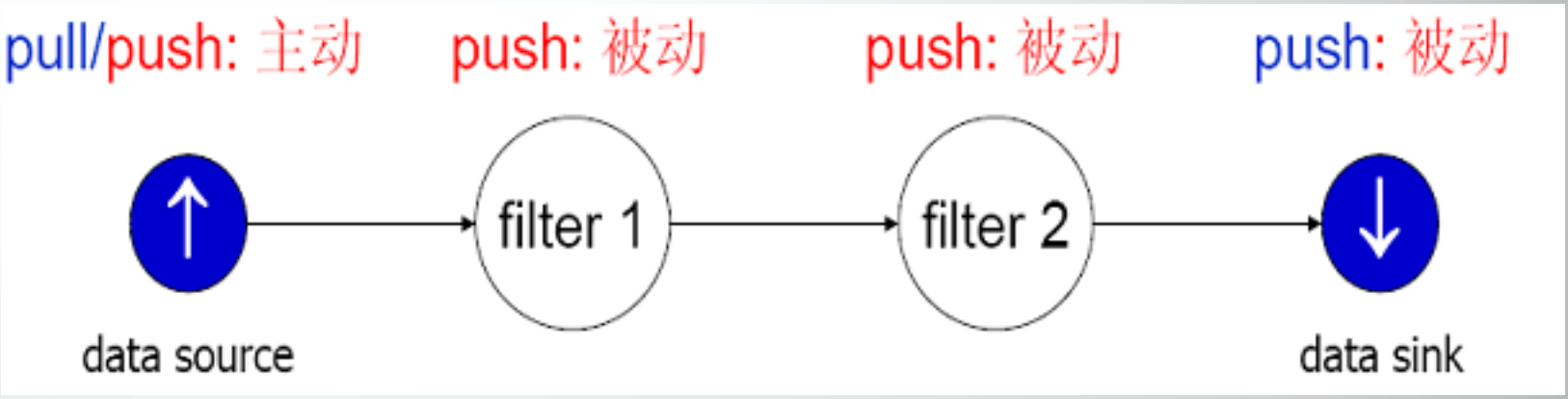
```
while (true) {
  Element x = inputPipe.read (...) ;
  outputPipe.write (f (x)) ;
}
```

Blocking read

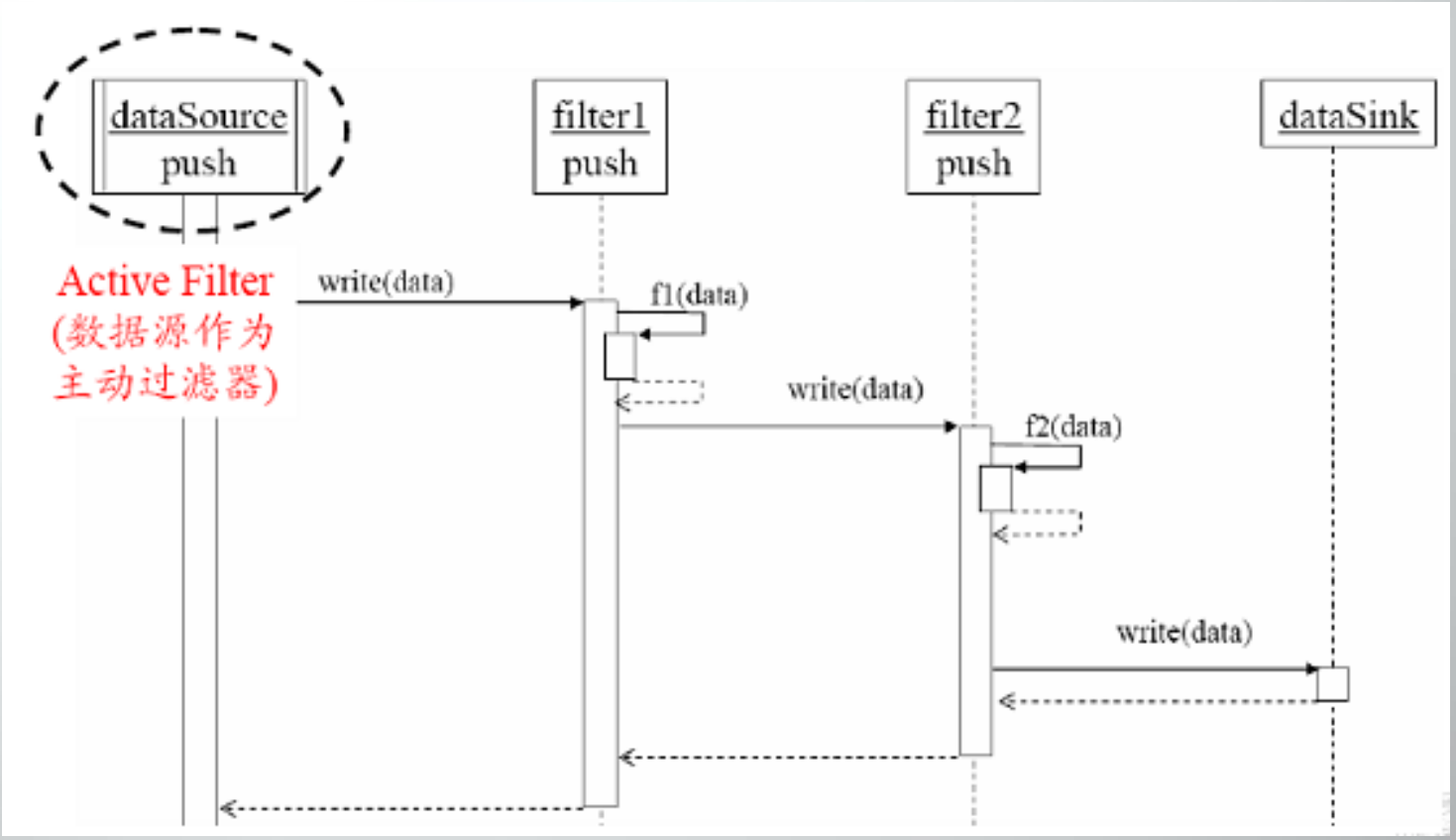
Blocking write



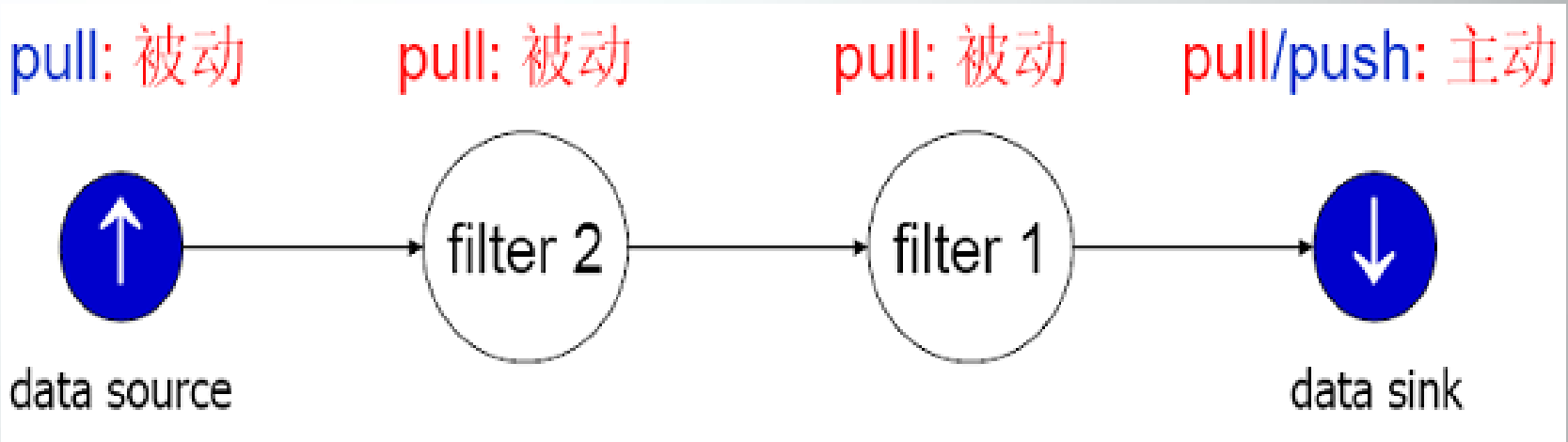
具有推式策略的被动过滤器



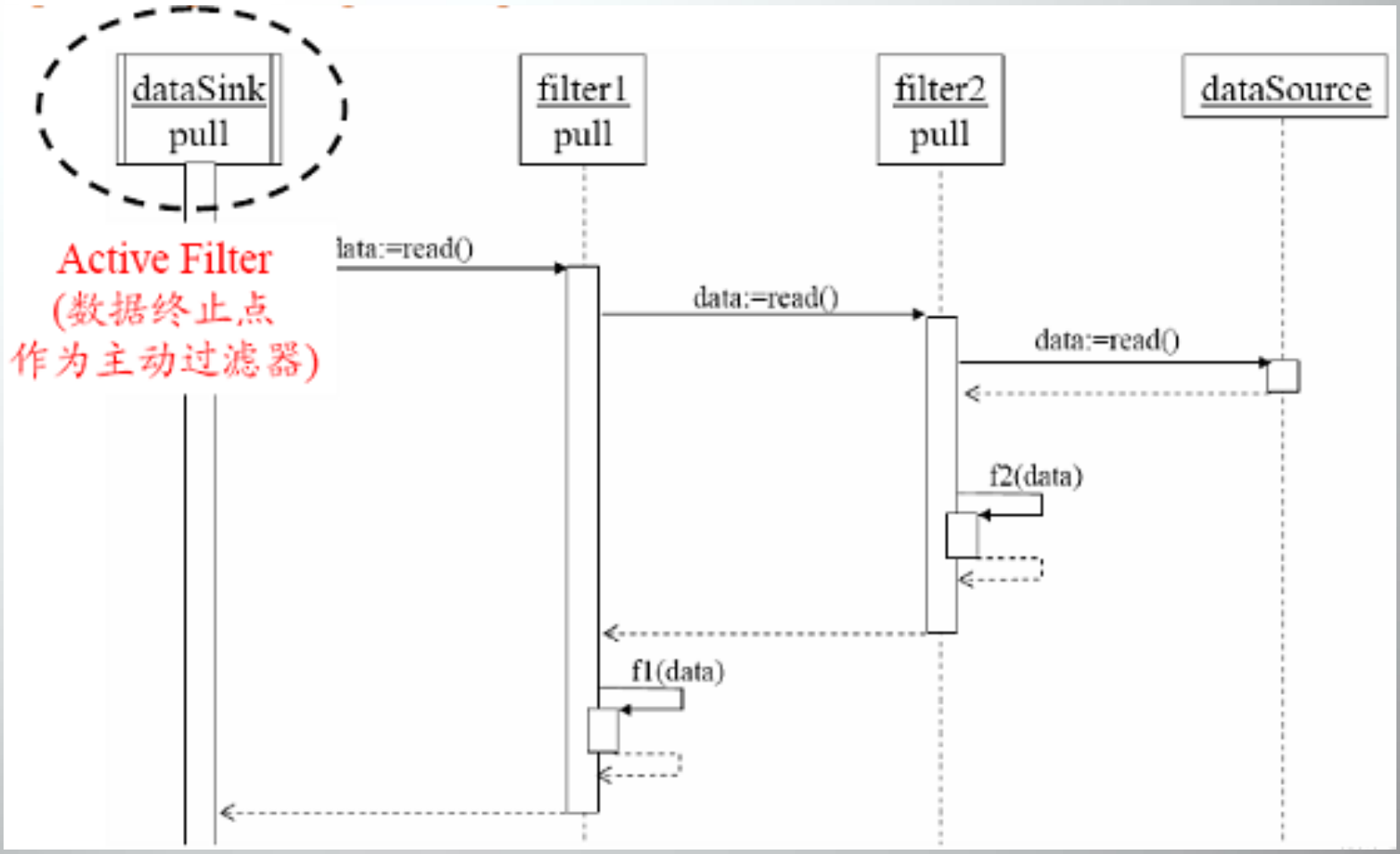
具有推式策略的被动过滤器



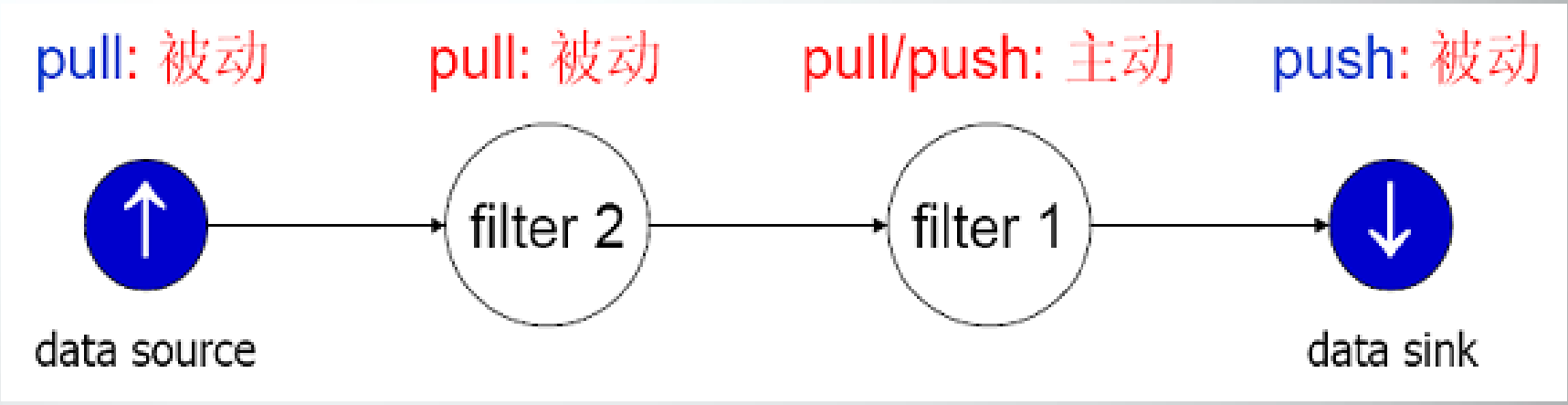
具有拉式策略的被动过滤器



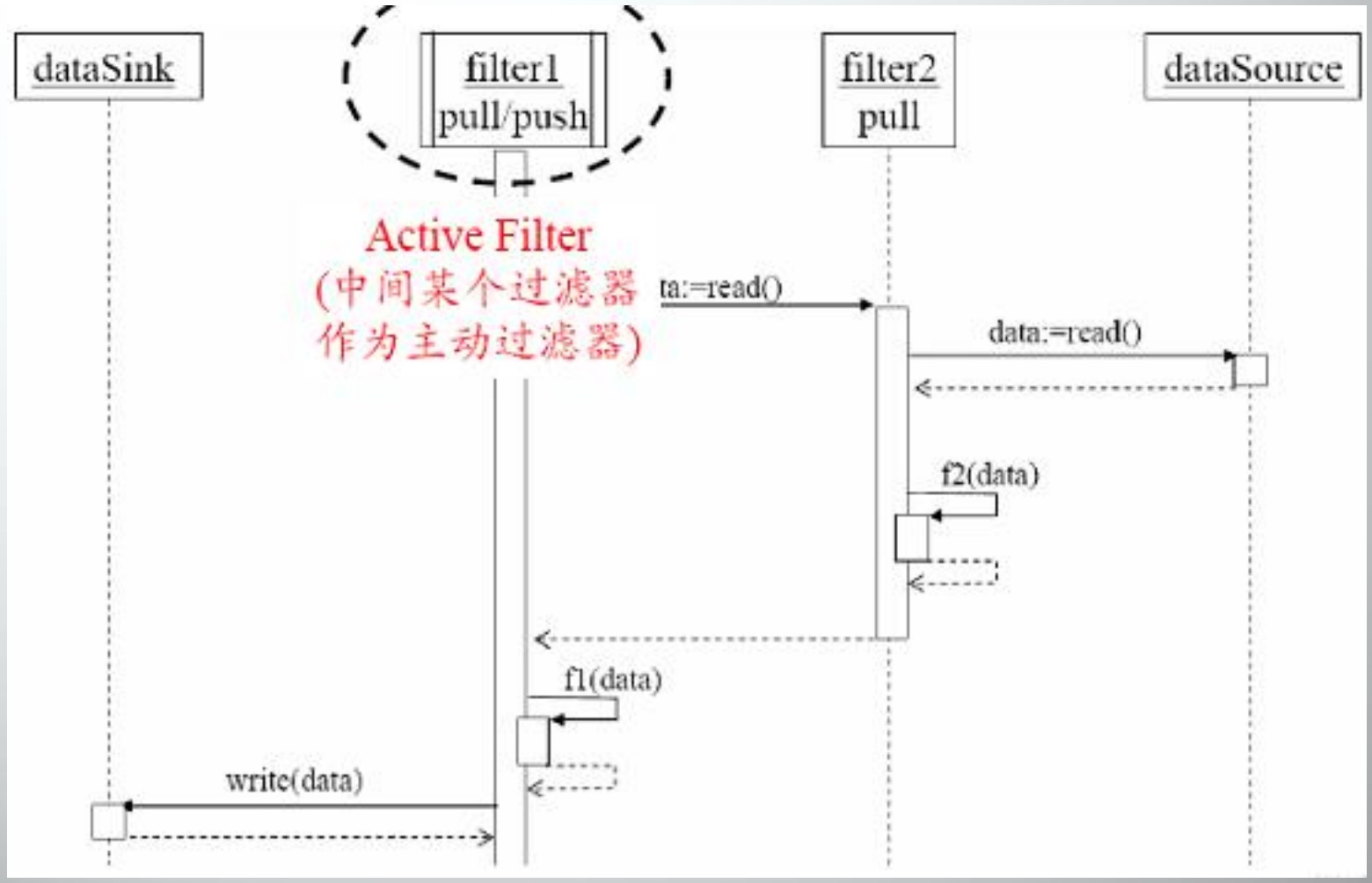
具有拉式策略的被动过滤器



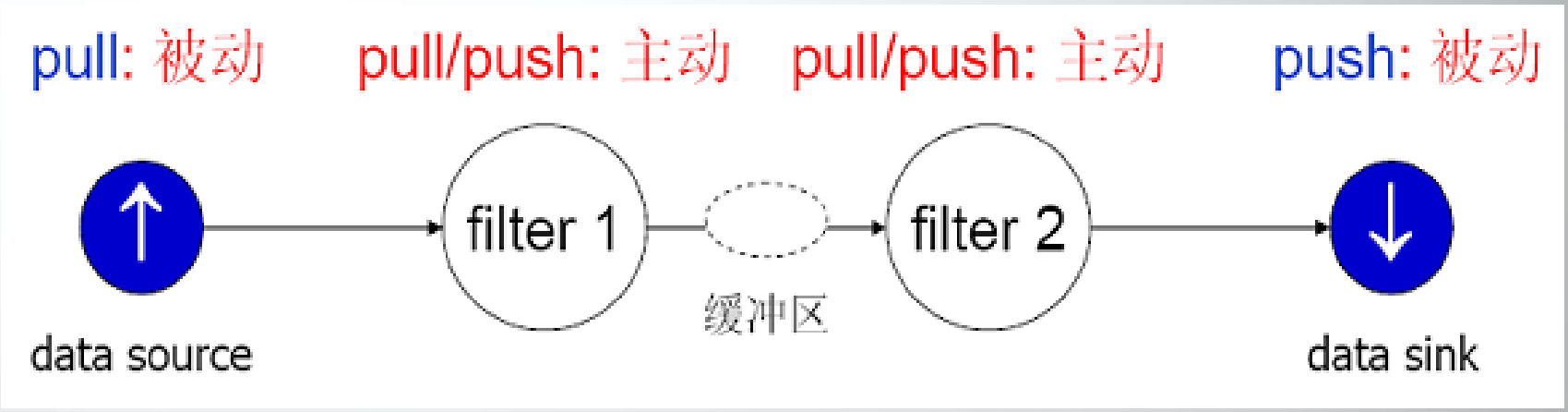
一个混合型的管道-过滤器系统



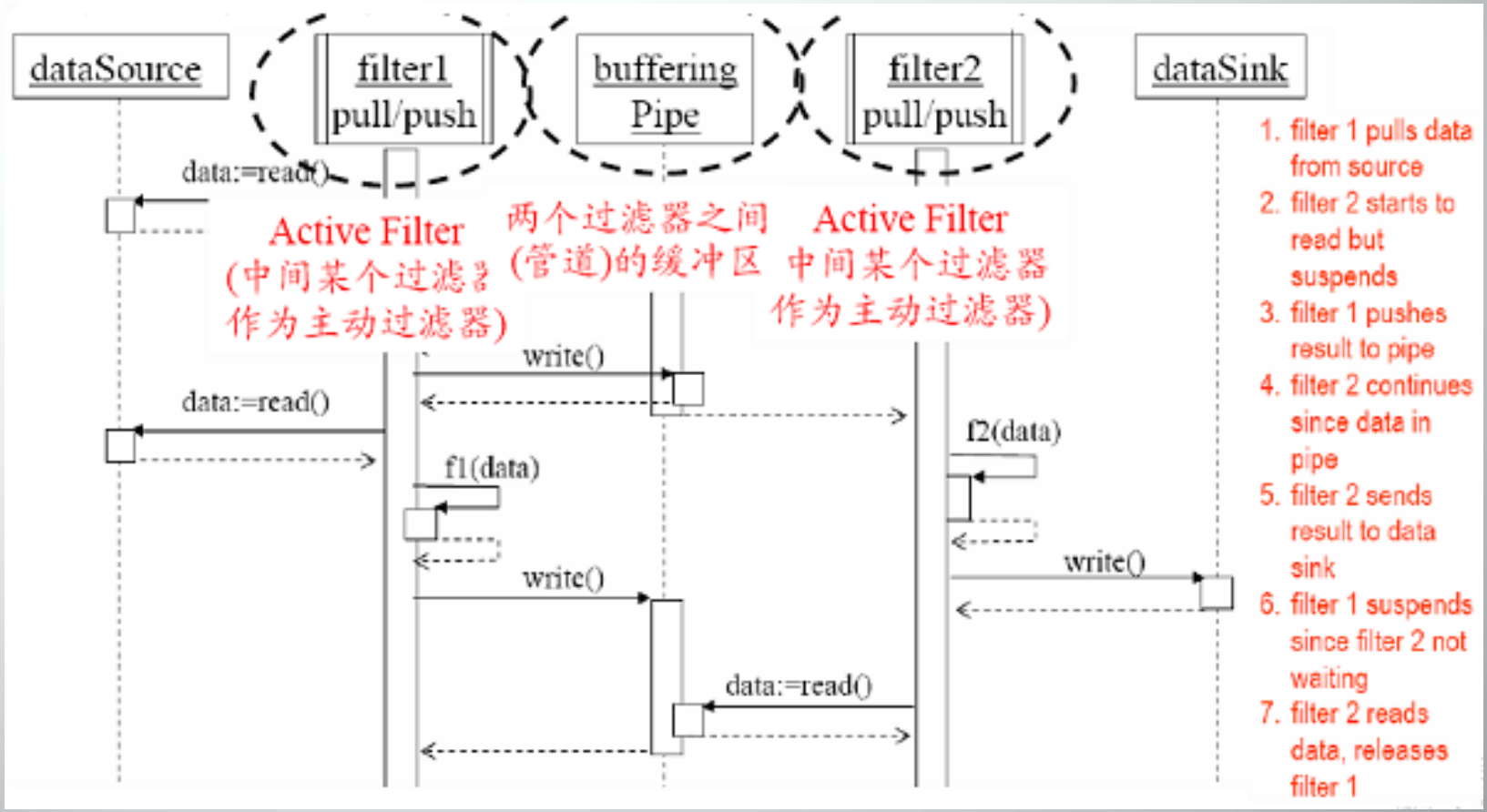
一个混合型的管道-过滤器系统



带有缓冲区的混合型管道-过滤器系统



带有缓冲区的混合型管道-过滤器系统



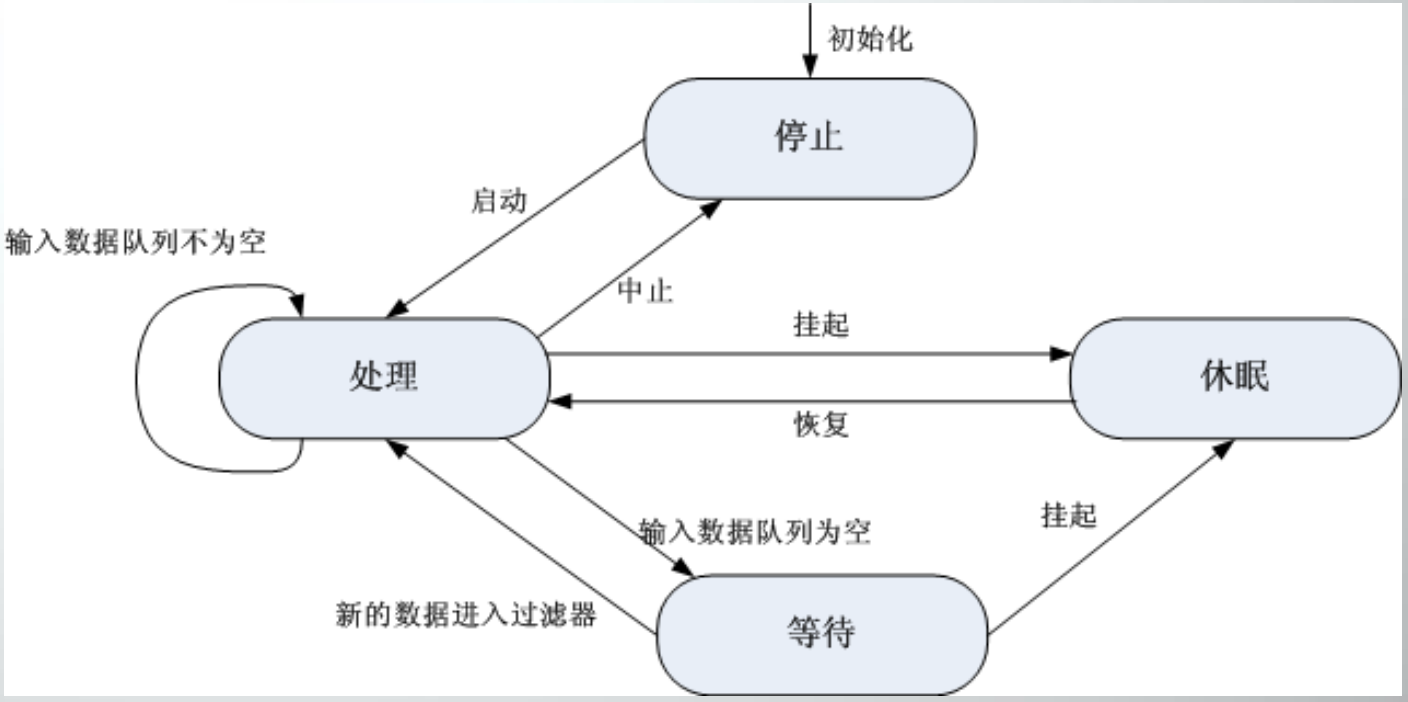
过滤器的设计

- 过滤器有如下状态：
  - 停止状态，工作状态，等待状态，休眠状态。
- 可以将过滤器用状态转换图表示。

过滤器的设计

- 停止状态：表示过滤器处于待启动状态，当外部启动过滤器后，过滤器处于处理状态；
- 处理状态：表示过滤器正在处理输入数据队列中的数据；
- 等待状态：表示过滤器的输入数据队列为空，此时过滤器等待，当有新的数据输入时，过滤器处于处理状态；
- 休眠状态：表示过滤器已经启动，但被挂起。

过滤器状态转换图



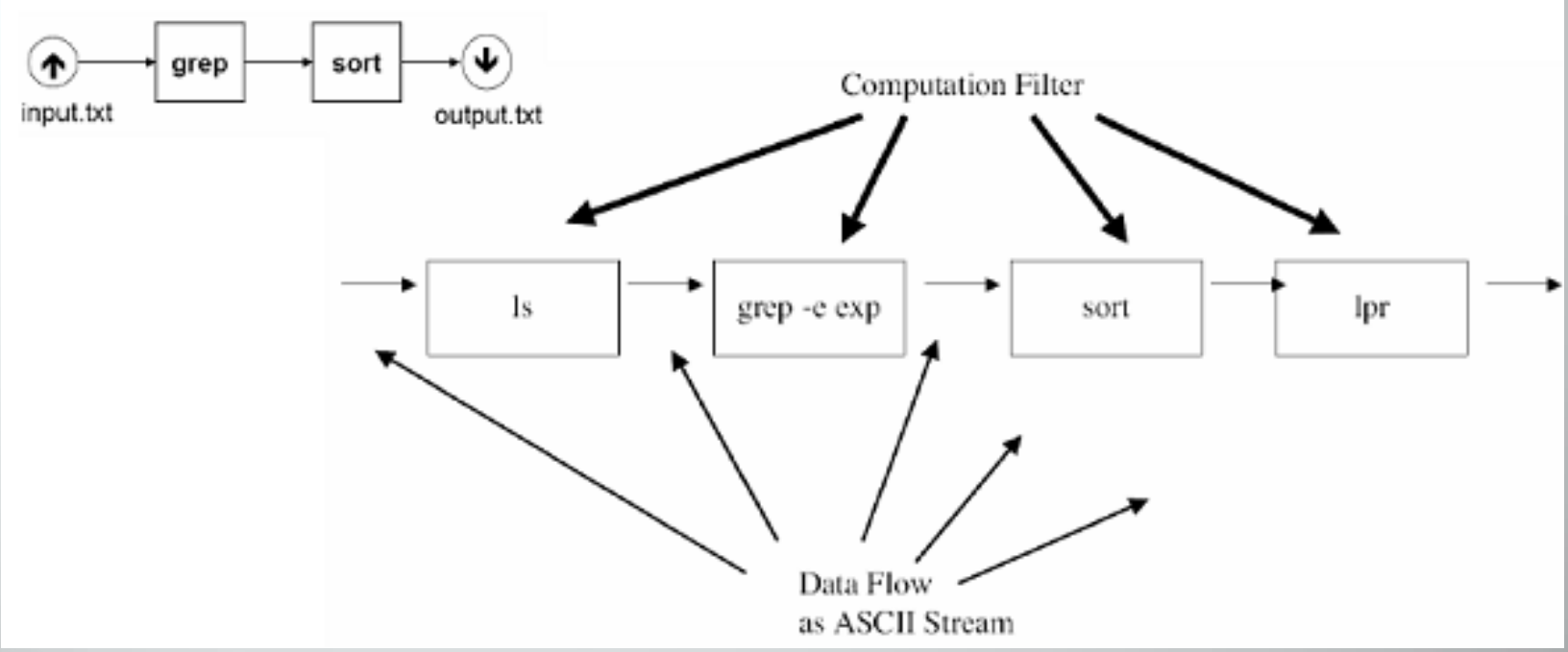
过滤器状态转换图

管道过滤器风格的典型应用

- Unix pipes (Unix管道)
- DOS管道命令
- Complier(编译器)
- Image processing (图像处理)
- Signal processing (信号处理)
- Voice and video streaming (声音与图像处理)

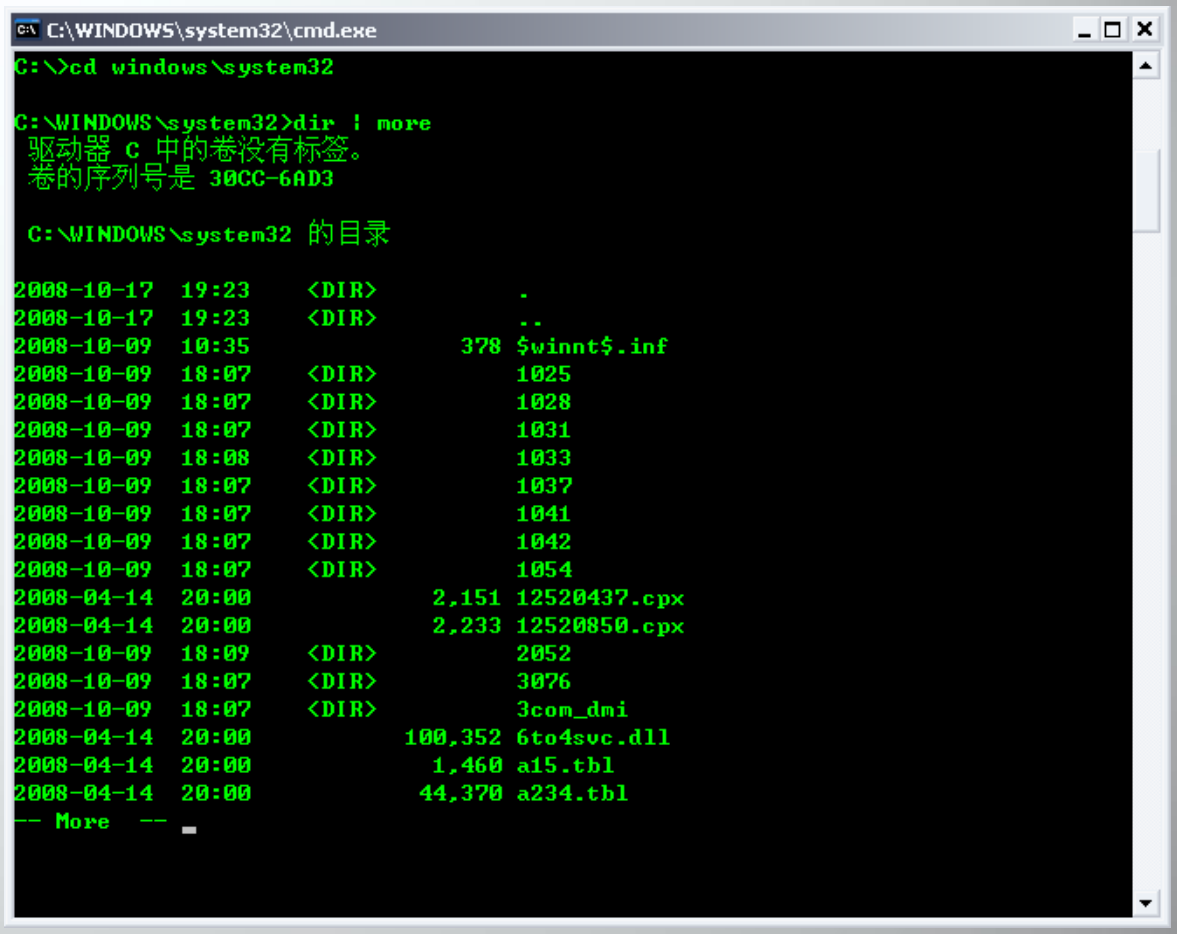
Unix系统中的管道过滤器结构

- cat input.txt | grep “text” | sort > output.txt

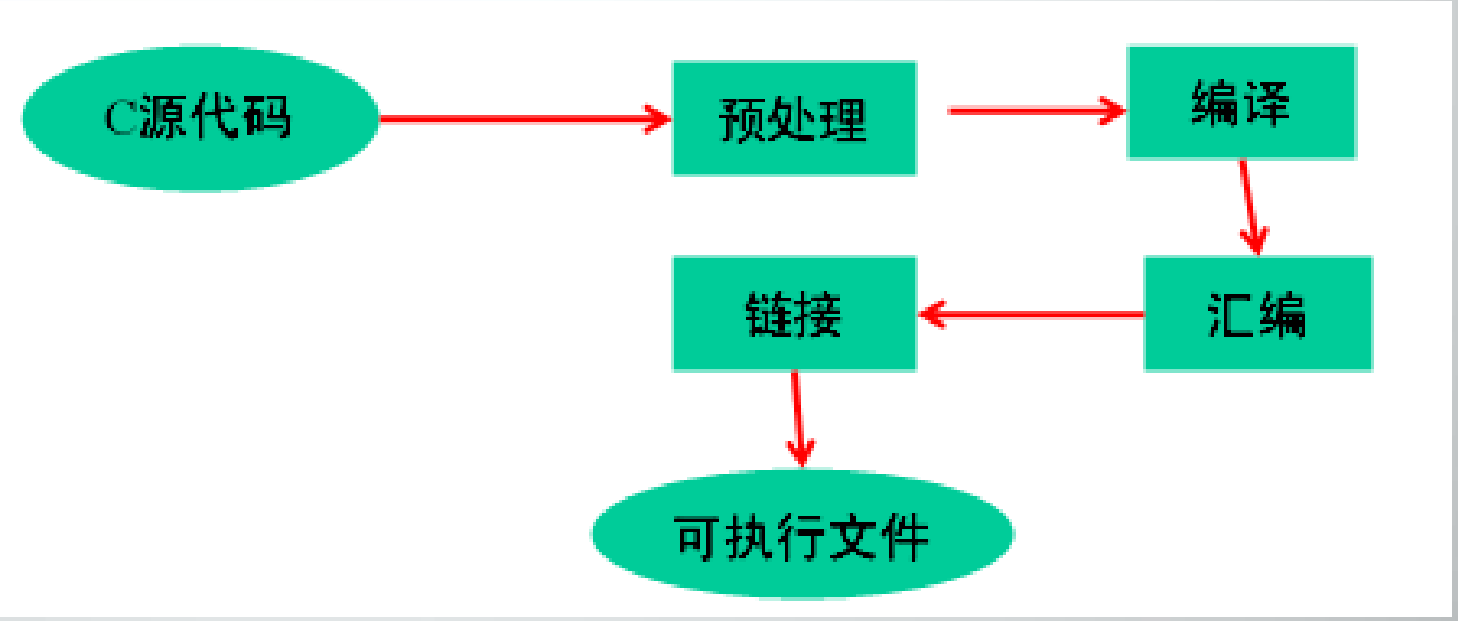


DOS 中的管道命令

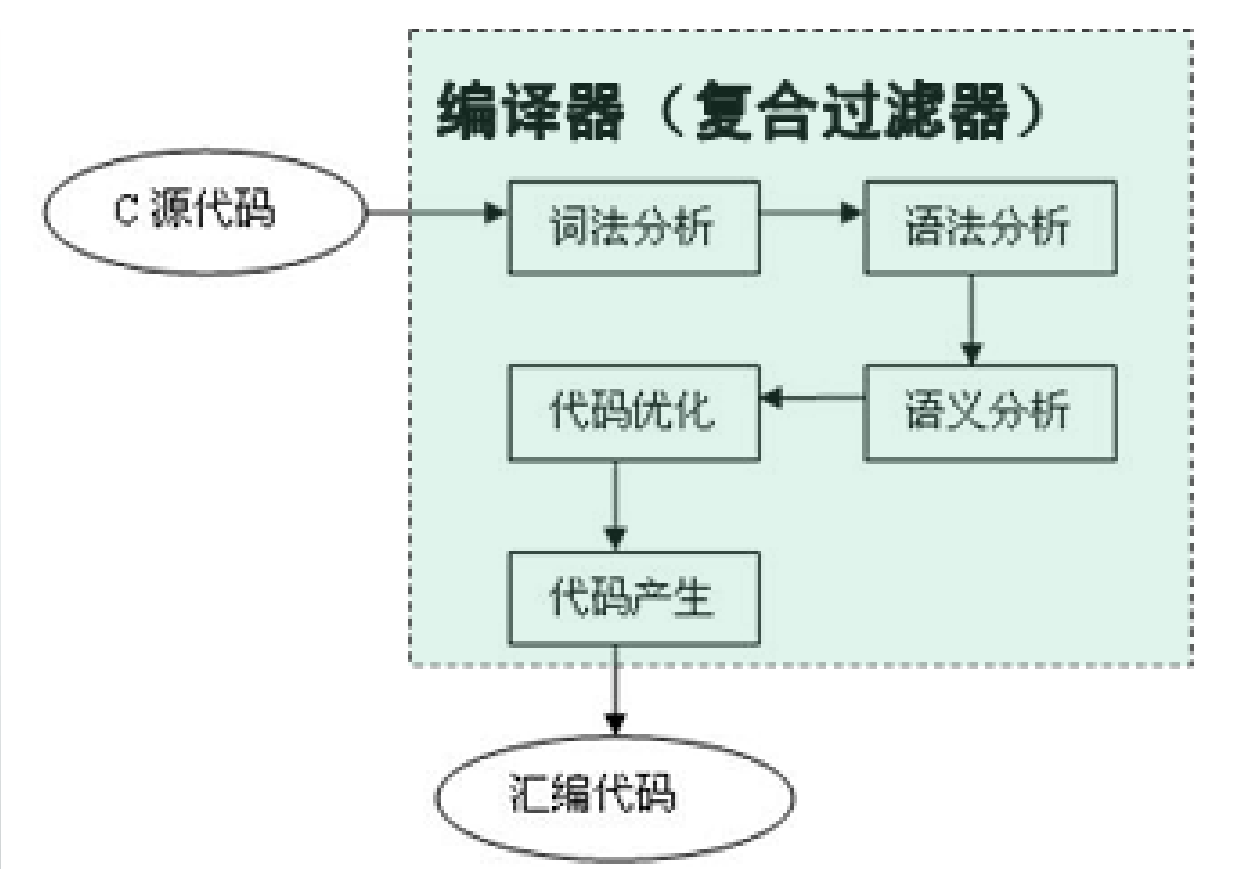
- dir | more



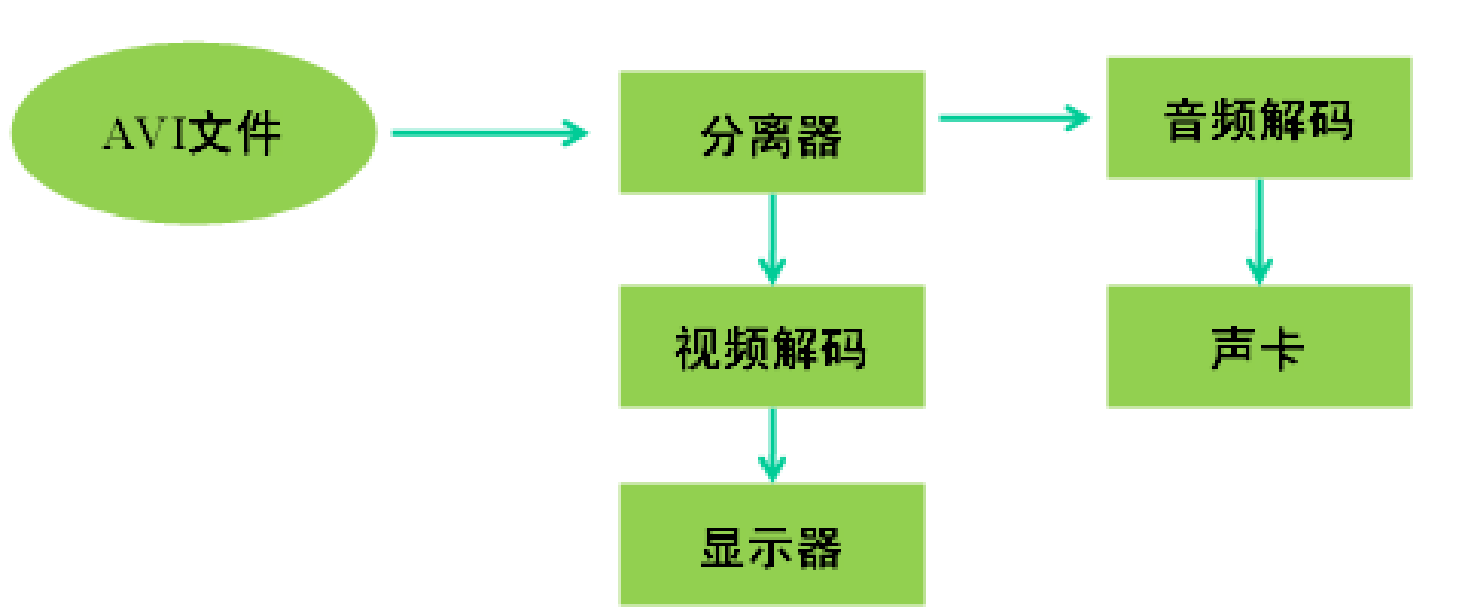
管道-过滤器风格的例子 - 编译器



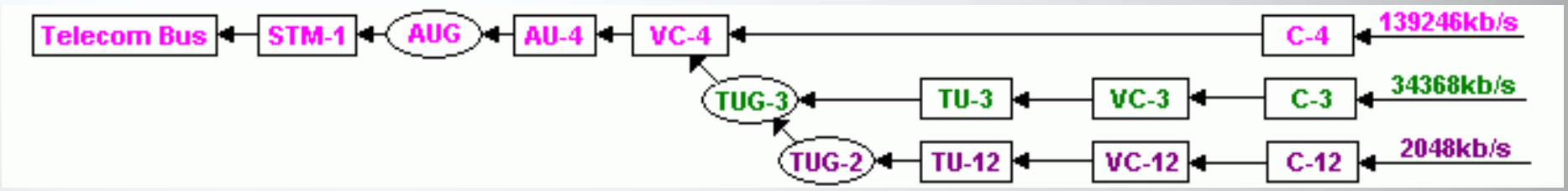
管道-过滤器风格的例子 - 编译器



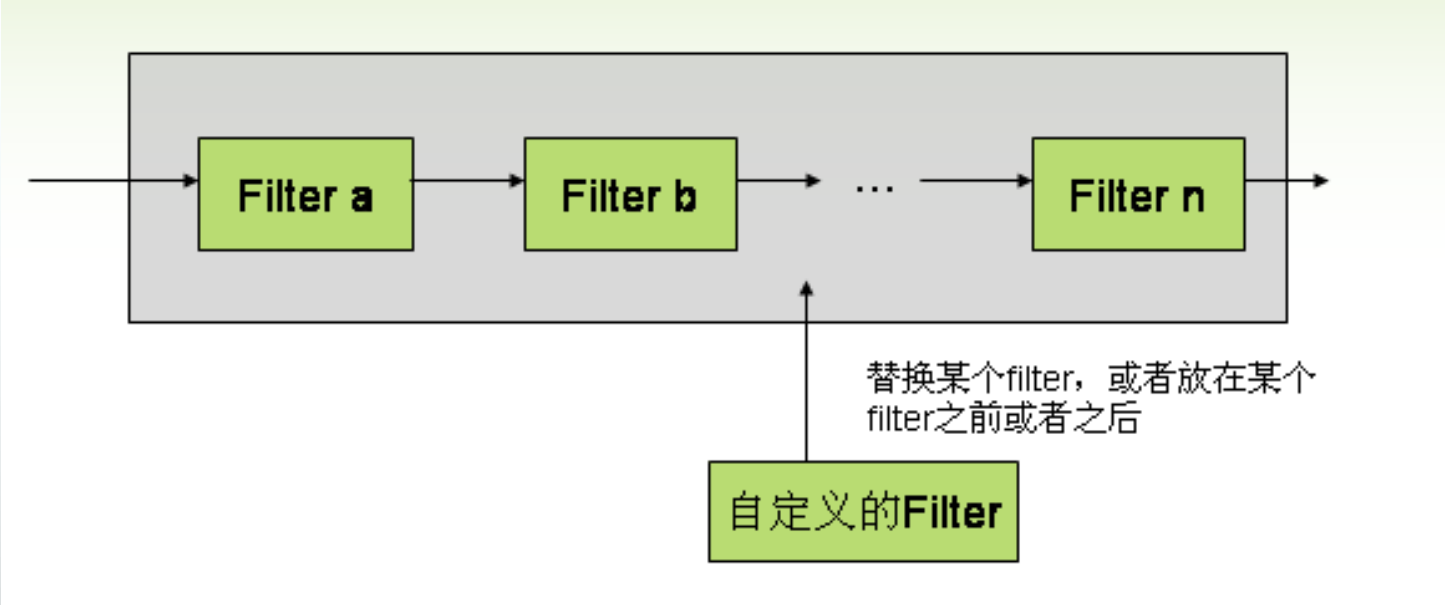
管道-过滤器风格的例子—媒体播放器



通讯协议的信息封装(e.g. SDH)



Spring——Filter

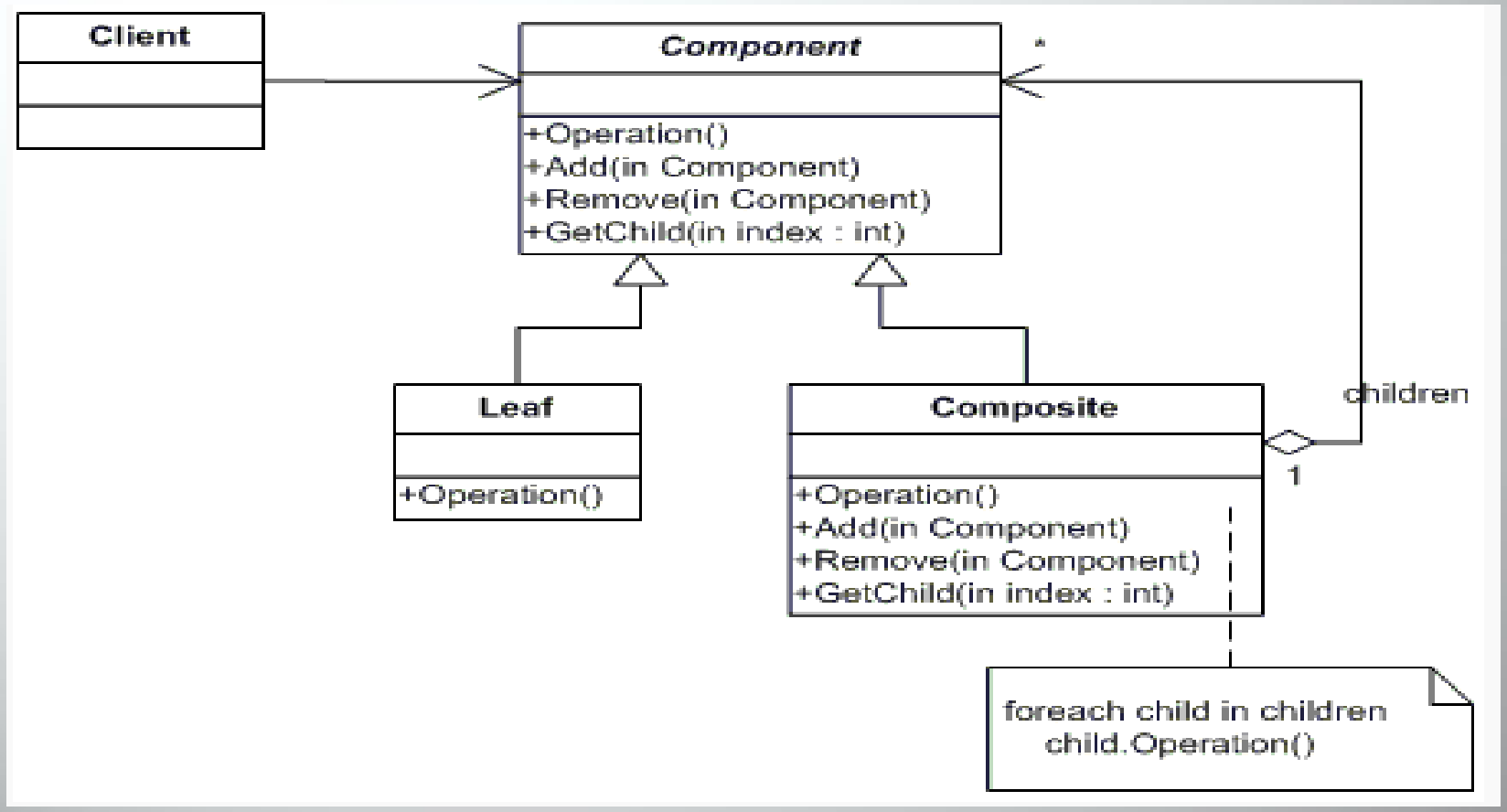


管道过滤器风格实现案例

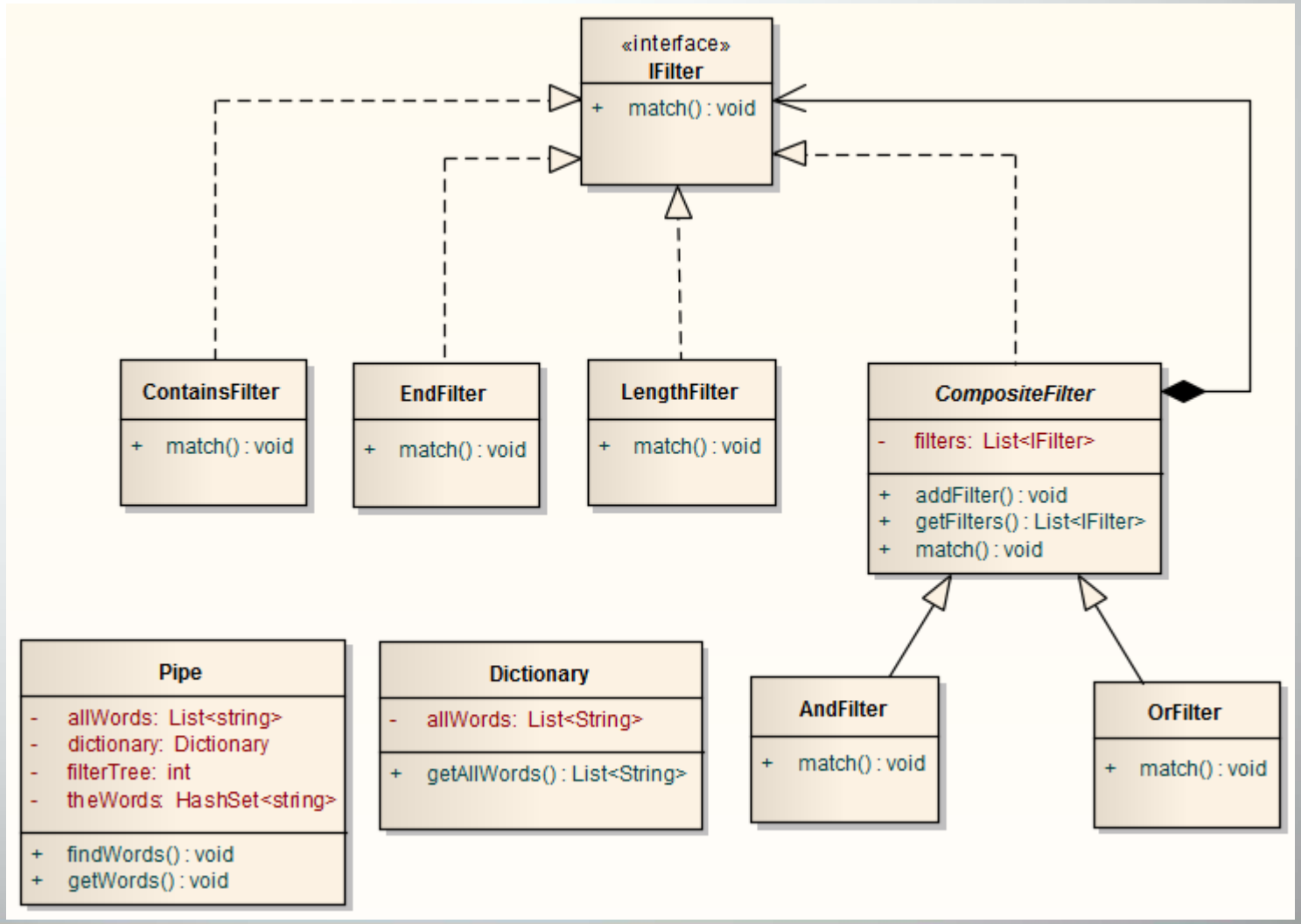
- 查询出字典中包含a、b、cd子串，以end/END结尾，长度>7的单词。

管道过滤器风格实现案例

组合模式回顾



管道过滤器风格实现案例



管道过滤器风格的其他应用

- 网络监听和流量控制(Network monitoring and traffic engineering)
- 电话通信(Telecom call records)
- 网络安全 (Network security)
- 金融领域(Financial Application)
- 工业生产(Manufacturing Processes)
- 网页日志与点击流(Web logs and click streams)

管道-过滤器风格优点

- 使得系统中的构件具有良好的隐蔽性和高内聚、低耦合的特点
- 设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成
- 支持功能模块的复用
- 较强的可维护性和可扩展性
- 支持一些特定的分析，如吞吐量计算和死锁检测等
- 具有并发性

管道-过滤器风格缺点

- 交互式处理能力弱
- 设计者也许不得不花费精力协调两个相对独立但又存在某种关系的数据流之间的关系
- 过滤器具体实现的复杂性
- 往往导致系统处理过程的成批操作

管道和连接器风格的参考文献

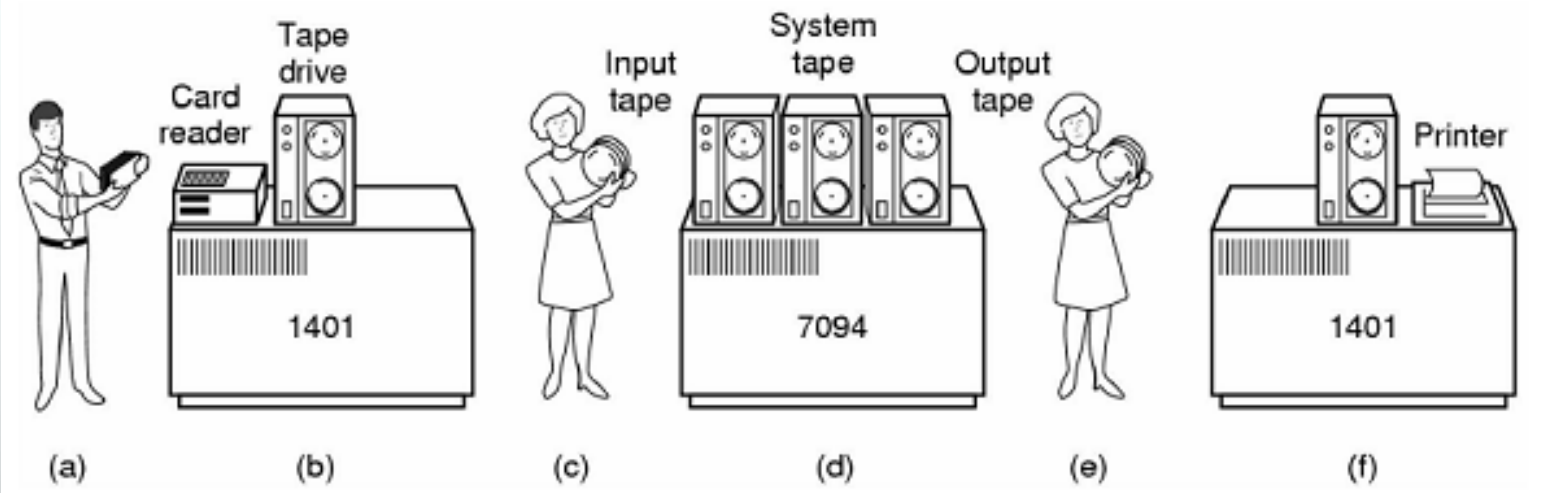
- Maurice J. Bach. *The Design of the UNIX Operating System*, chap. 5, pp. 11-119. Software Series. Prentice Hall, 1986
- Norman Delisle and David Garlan. *Applying formal specification to industrial problems: A specification of an oscilloscope*. IEEE Software, 7(5):29-37, Sept. 1990
- J. C. Browne, M. Azam, and S. Sobek. *Code: A unified approach to parallel programming*. IEEE Software, July 1989.
- G. Kahn. *The semantics of a simple language for parallel programming*. Information Processing, 1974
- David Barstow and Alex Wolf. *Design methods and software architectures track.* In Proceedings of the 7th International Workshop in Software Specification and Design. IEEE Press, 1993

三种典型的数据流风格

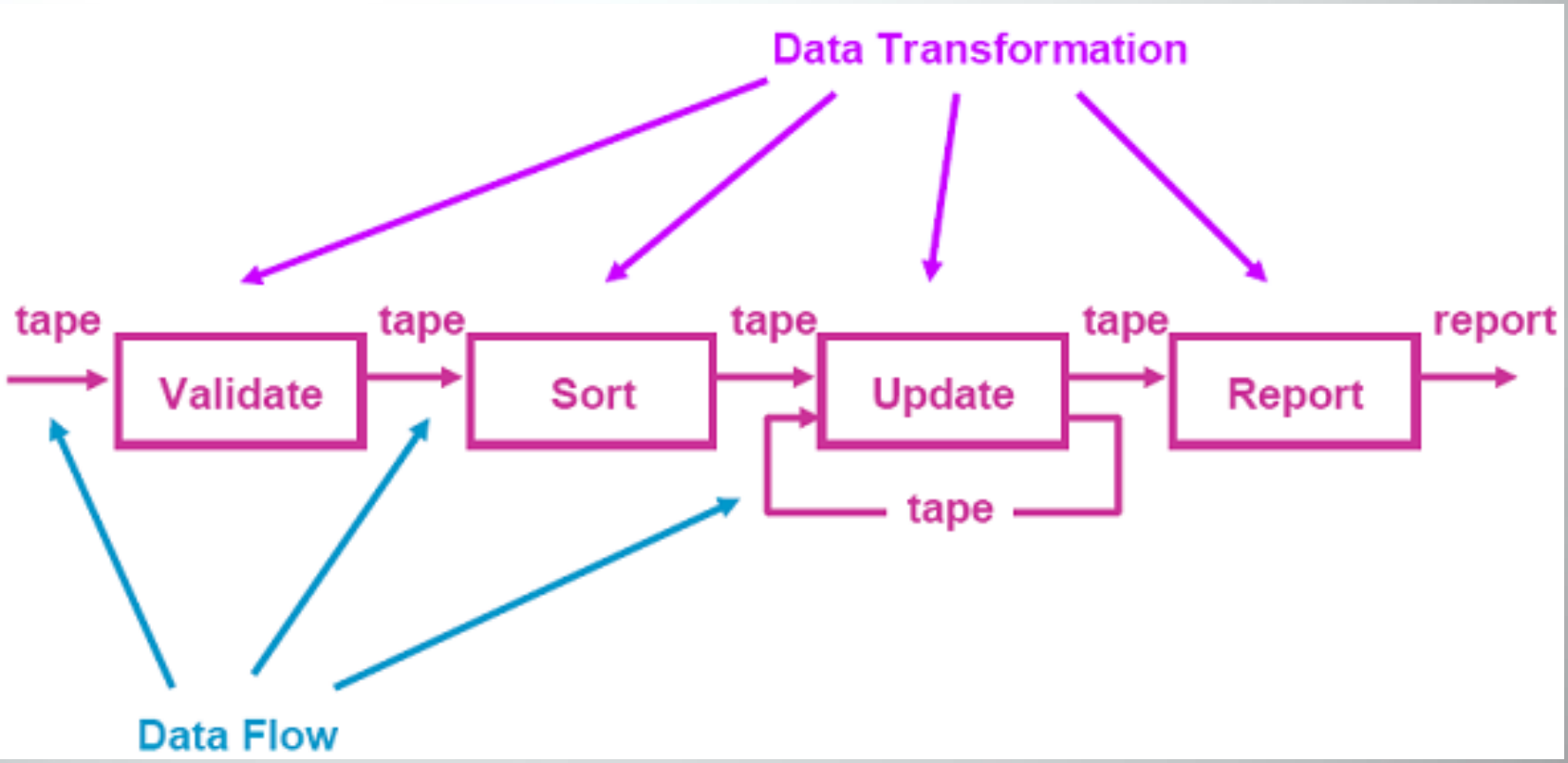
- 管道/过滤器（Pipe-and-Filter）
- 批处理（Batch Sequential）
- 过程控制（Process Control）

批处理风格

- 数据流风格的另外一种模型
- 数据是逐次输入、累积保存、成批处理的
- 输入大量的记录，处理后产生汇总性的输出



批处理风格


Data Transformation
tape
Validate
tape
Sort
tape
Update
tape
Report
report
tape
Data Flow

批处理风格

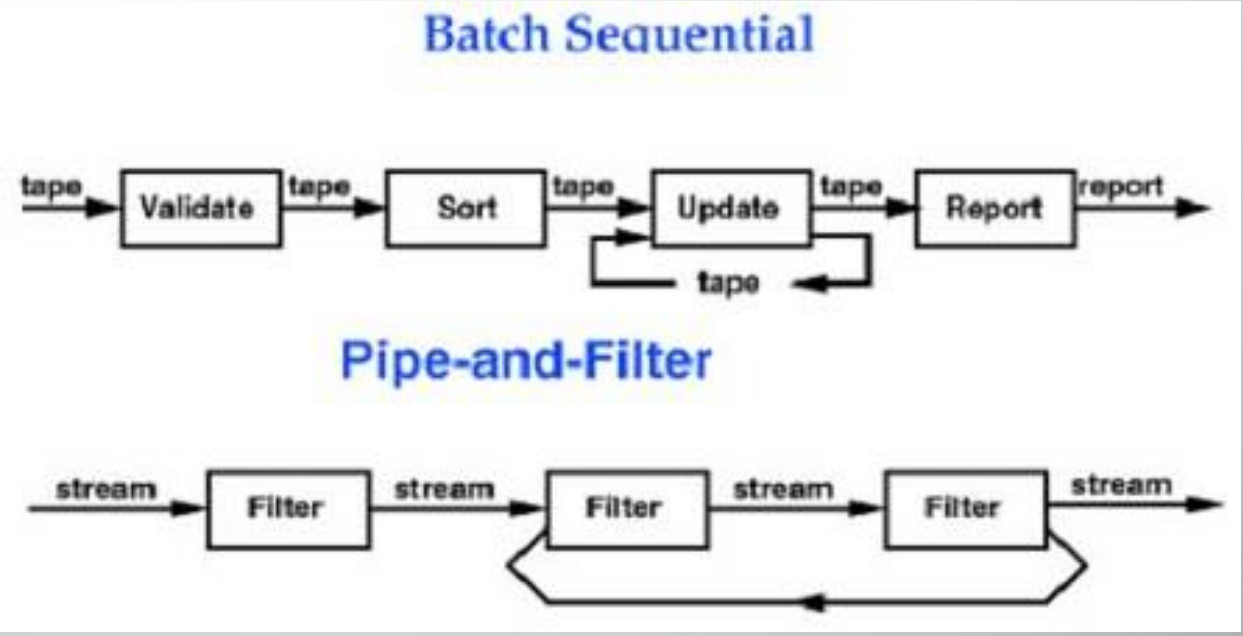
- 每个处理步骤是一个独立的程序
- 每一步必须在前一步结束后才能开始
- 数据必须是完整的，以整体的方式传递

批处理风格基本构成

- 基本构件：独立的应用程序
- 连接件：某种类型的媒质
  - 连接件定义了相应的数据流图，表达拓扑结构
  - 每一步骤必须在前一步骤完全结束之后方能开始

批处理与管道-过滤器的比较

- 相似点
  - 把任务分解成为一系列固定顺序的计算单元
  - 彼此间只通过数据传递交互



批处理与管道-过滤器的比较

不同点

Batch Sequential	Pipe-and-Filter
* total(整体传递数据) * coarse grained(构件粒度较大) * high latency (延迟高，实 时性差) * no concurrency (无并发)	* incremental(增量) * fine grained (构件粒度较小) * results starts processing (实时性) * concurrency possible (可并发)

批处理风格

票据处理系统

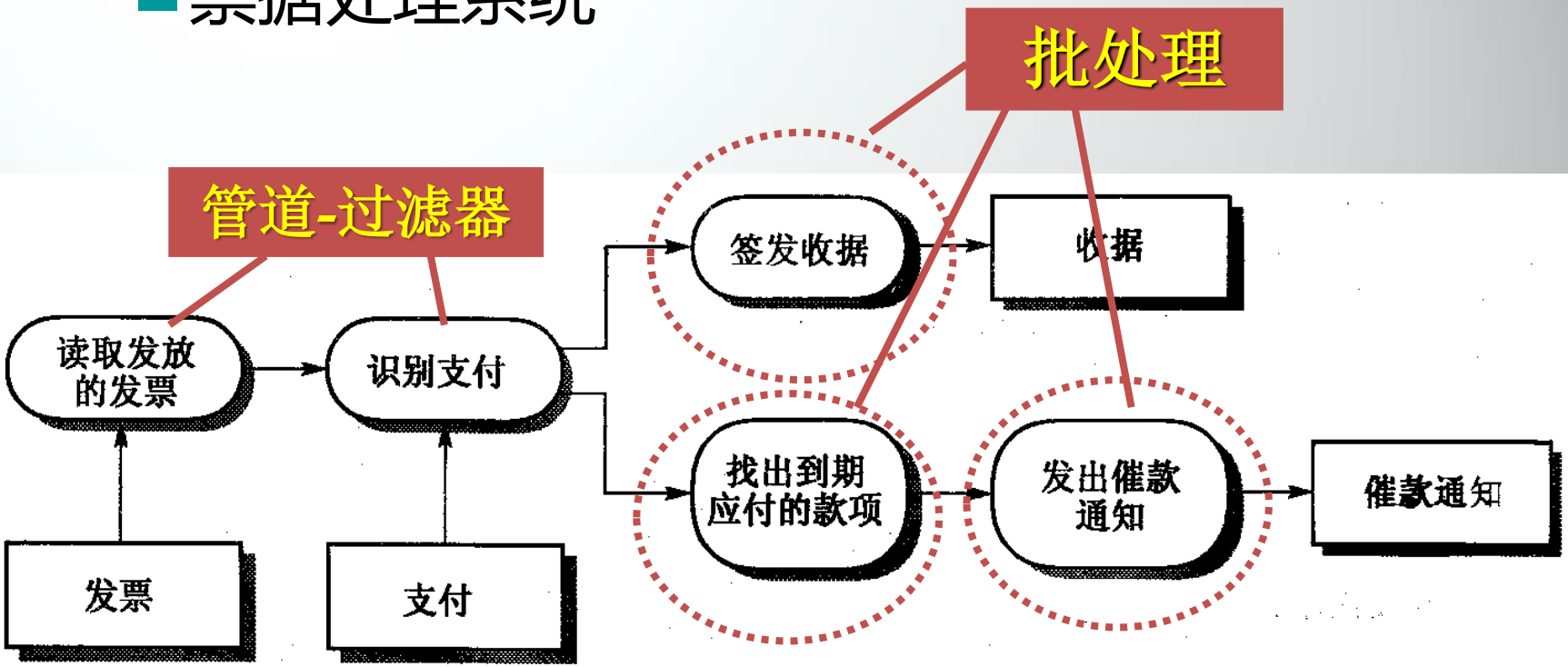


图 10-10　发票处理系统的数据流模型

批处理体系结构优点

- 支持转换的复用，顺序的或是并发的处理。
- 很直观，许多人都能将它们的工作理解成输入和输出处理。

批处理体系结构缺点

- 需要一种适合于所有转换的通用格式。
- 若要求转换是独立的且可复用的，需要定义一种对所有数据都通用的标准格式。
- 可能无法集成那些不兼容数据格式的转换。

过程控制风格（略）

小结

Thank You，谢谢！