第3章 软件体系结构风格

内容

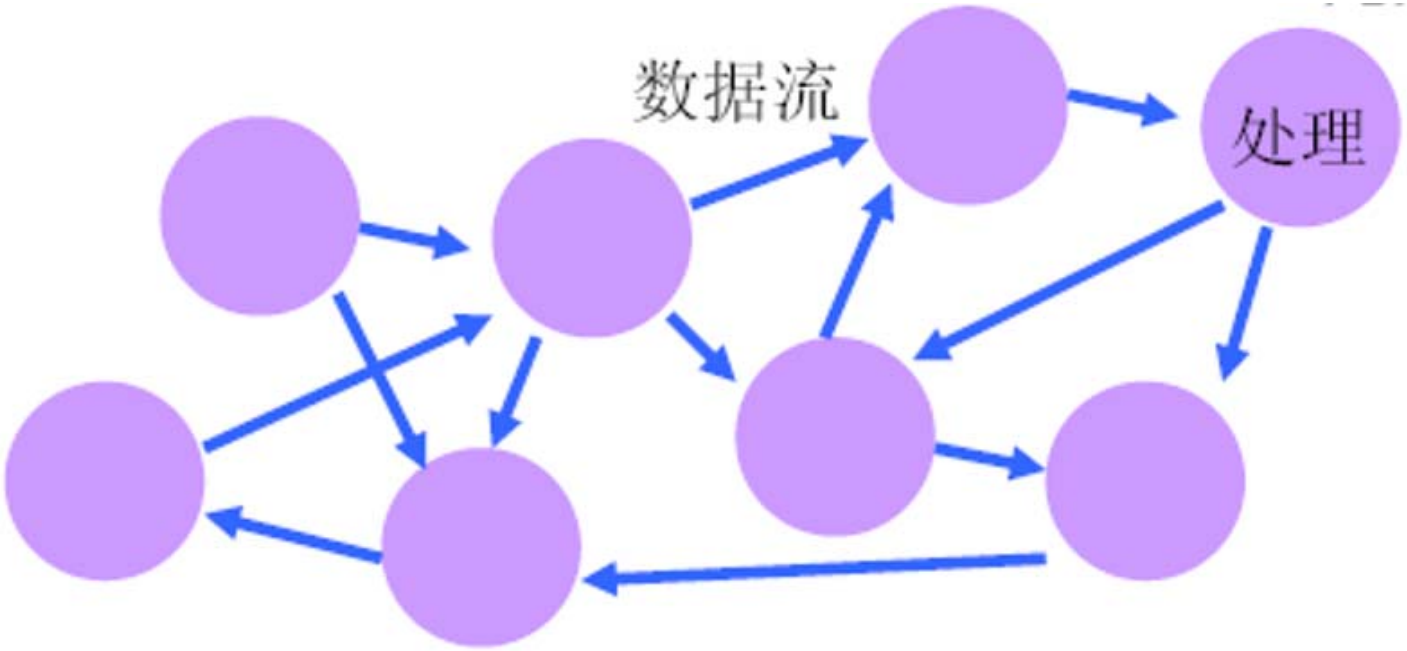
软件架构风格

- 软件架构风格是一个面向一类给定环境的架构设计决策的集合，这些通用的设计决策形成了一种特定的模式，为一族系统提供粗粒度的抽象框架。
- 每一个软件系统都有其占主导地位的软件架构风格。
- “从软件中来，到软件中去”
- 架构风格通过为常见的问题提供解决方案，增强了对问题的分解能力、提升了设计重用的水平。

内容

- **3.1 概述**
- **3.2 数据流风格**
- 3.3 过程调用风格
- 3.4 独立构件风格
- 3.5 层次风格
- 3.6 虚拟机风格
- 3.7 客户/服务器风格
- 3.8 表示分离风格
- 3.9 插件风格
- 3.10 微内核风格
- 3.11 SOA风格

数据流风格（Data flow style）

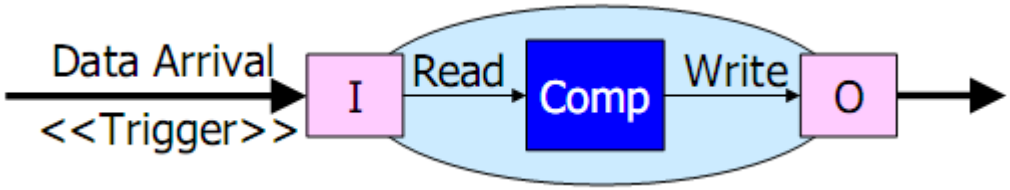


数据流风格的特征

- **A data flow system is one in which**
  - the availability of data controls the computation (数据的可用性决定着处理<计算单元>是否执行)
  - the structure of the design is dominated by orderly motion of data from process to process (系统结构：数据在各处理之间的有序移动)
  - in a pure data flow system, there is no other interaction between processes (在纯数据流系统中，处理之间除了数据交换，没有任何其他的交互)

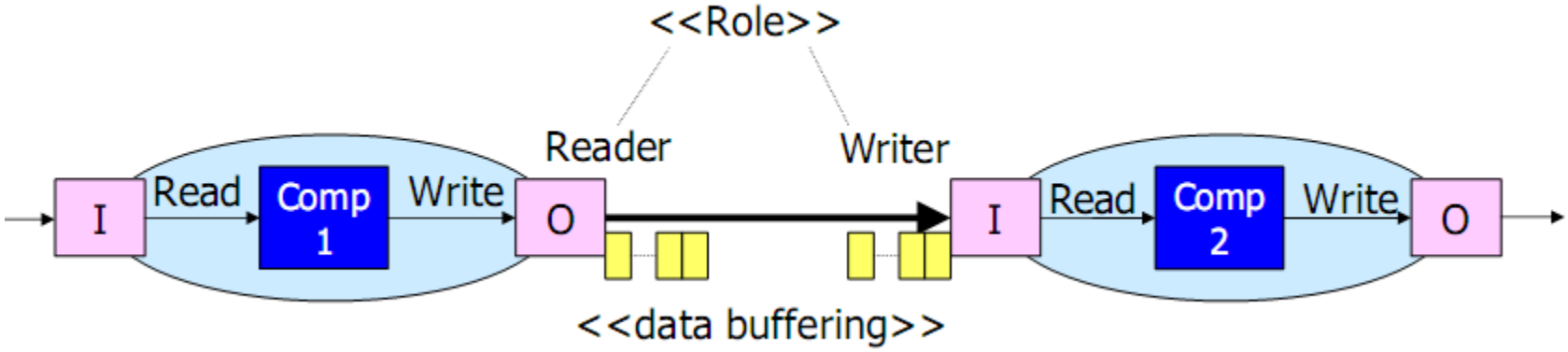
数据流风格的基本构件

- **Components: data processing components**(基本构件：数据处理)
  - Interfaces are input ports and output ports (构件接口：输入端口和输出端口)
  - Input ports read data; output ports write data (从输入端口读取数据，向输出端口写入数据)
  - Computational model: read data from input ports, compute, write data to output ports (计算模型：从输入端口读数，经过计算/处理，然后写到输出端口)



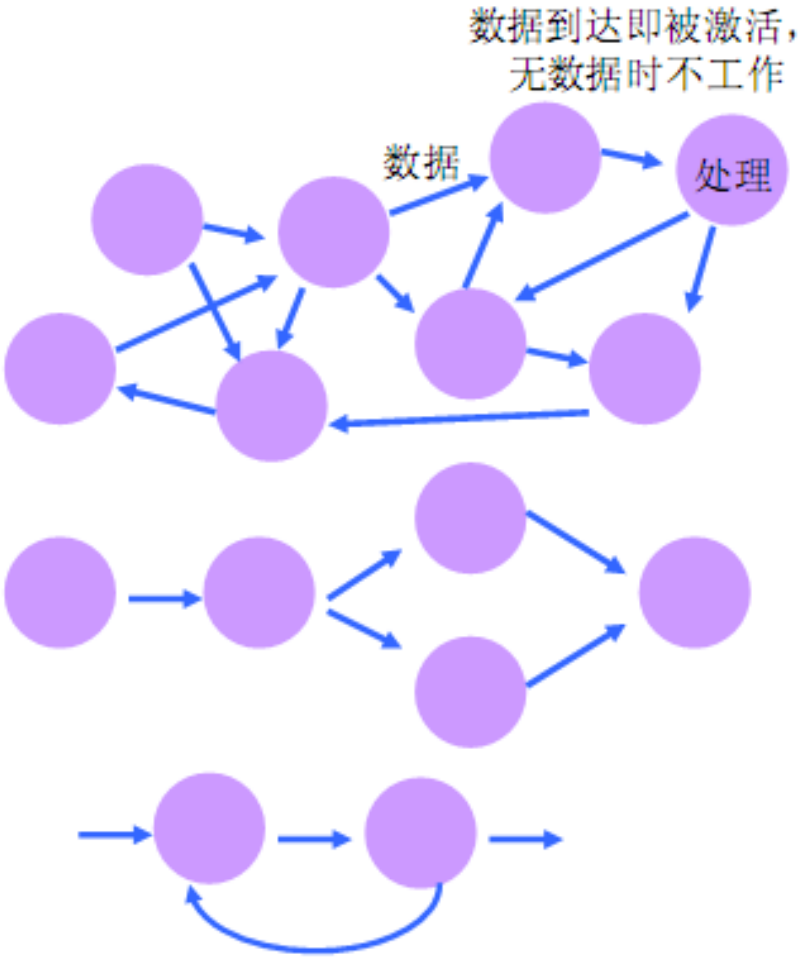
数据流风格的连接件

- **Connectors: data flow (data stream)**（连接件：数据流）
- Uni-directional, usually asynchronous, buffered (单向、通常是异步、有缓冲)
- Interfaces are reader and writer roles (接口角色：reader和writer)
- Computational model (计算模型: 把数据从一个处理的输出端口传送到另一个处理的输入端口)



数据流风格的拓扑结构

- Arbitrary graphs (任意拓扑结构的图)



In general, data can flow in Arbitrary patterns (一般来说，数据的流向是**无序的**)

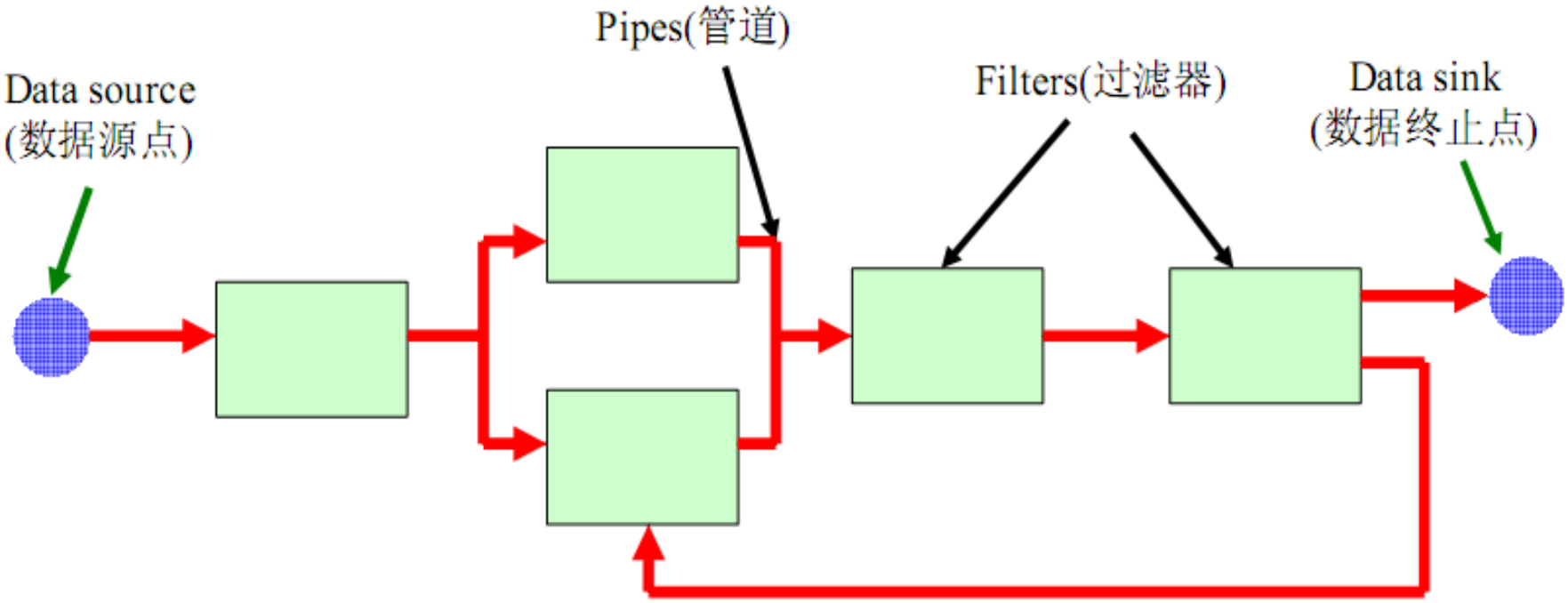
Often we are primarily Interested in nearly linear Data flow systems (我们主要关注近似**线性的数据流**)

Or in very simple, highly constrained cyclic structures(或在限度内的循环数据流)

两种典型的数据流风格

- Pipe-and-Filter (管道-过滤器)
- Batch Sequential (批处理)

Pipe-and-Filter风格的直观结构



Pipe-and-Filter风格的基本构成

- Components: Filters — process data streams (构件：过滤器，处理数据流)
  - A filter encapsulates a processing step (algorithm or computation)（一个过滤器封装了一个处理步骤)
  - Data source and data sink are particular filters (数据源点和数据终止点可以看作是特殊的过滤器)
- Connectors: A pipe connects a source and a sink filter (连接件：管道，连接一个源和一个目的过滤器)
  - Pipes move data from a filter output to a filter input (转发数据流)
  - Data is a stream of ASCII characters (数据是ASCII字符形成的流)
- Topology: Connectors define data flow graph (连接器定义了数据流图，形成拓扑结构)

过滤器（Filter）的一些基本特征

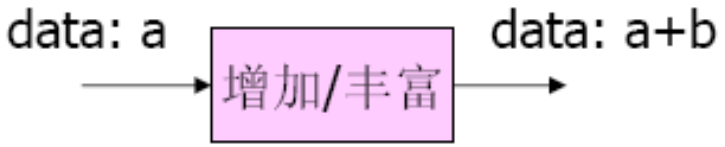
- Filters are independent entities, i.e.,
  - no context in processing streams (无上下文信息)
  - no state preservation between instantiations (不保留状态)
  - no knowledge of upstream/downstream filters (对其他过滤器无任何了解)
  - collections can be used to buffer the data passed through pipes: files, arrays, dictionaries, trees, etc. (可使用数据缓冲区临时保存数据流)
    - 蓄水池

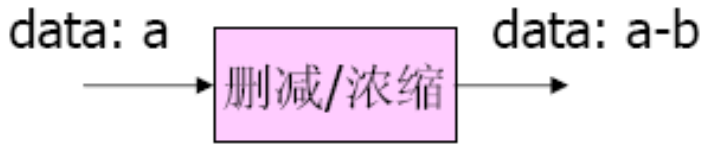
管道（Pipe）

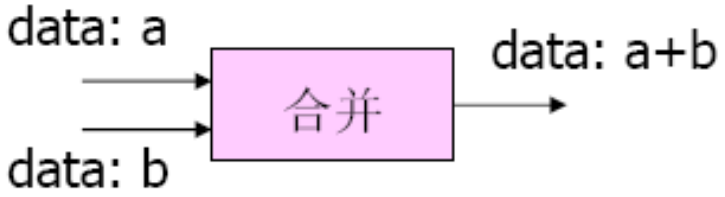
- Move data from a filter's output to a filter's input (or to a device or file) (作用：在过滤器之间传送数据)
  - One way flow from one data source to one data sink (单向流)
  - A pipe may implement a buffer (可能具有缓冲区)
  - Pipes form data transmission graph (管道形成传输图)

- 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。

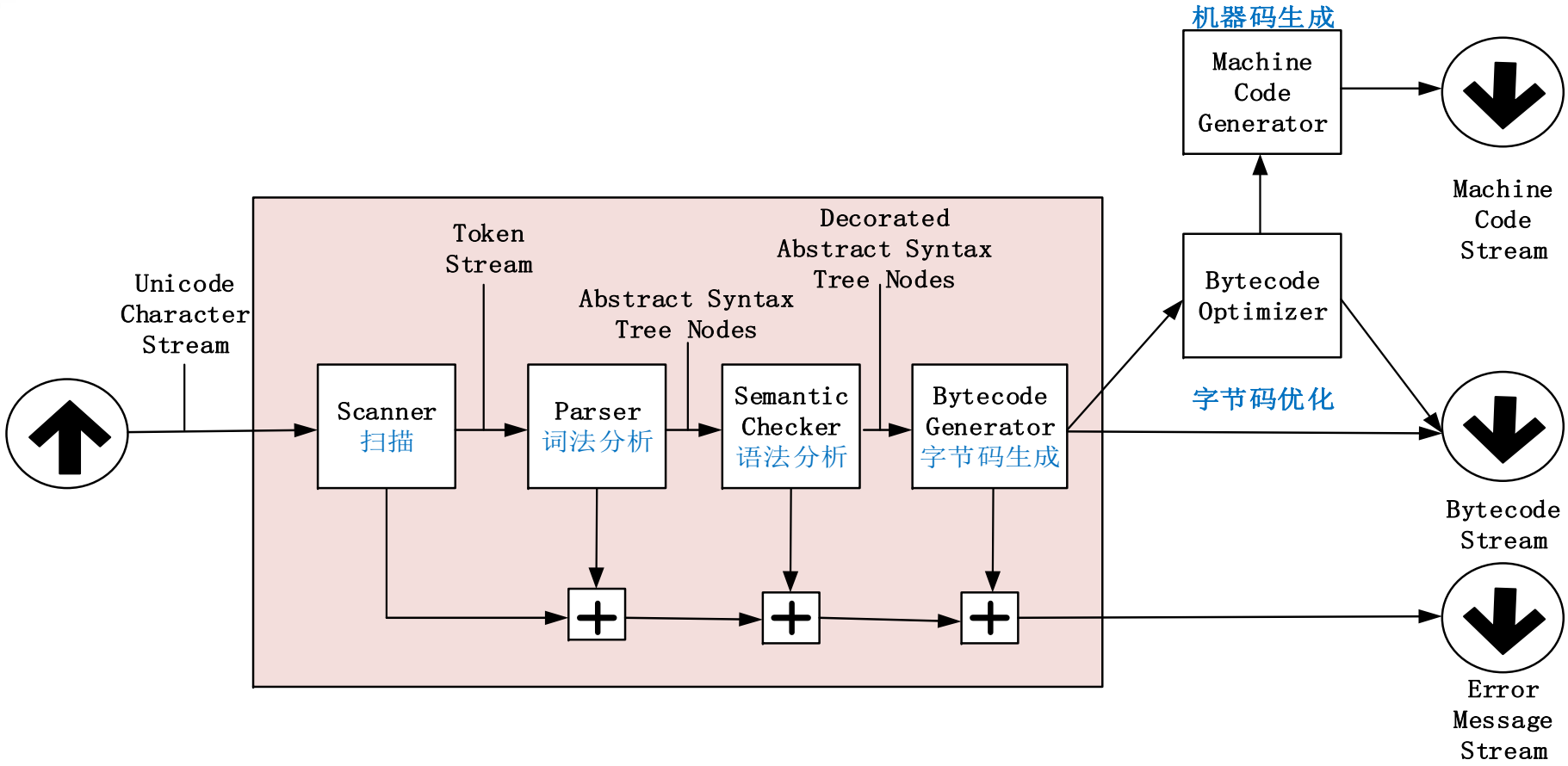
过滤器对数据流的五种变换类型







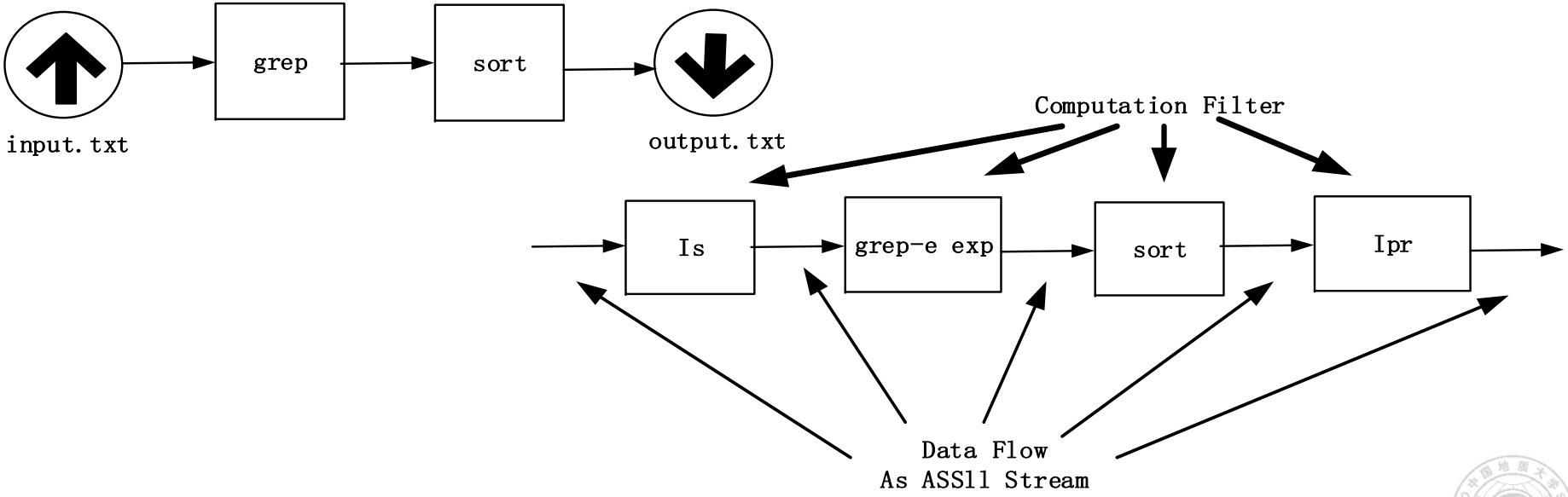
管道-过滤器风格的例子—编译器



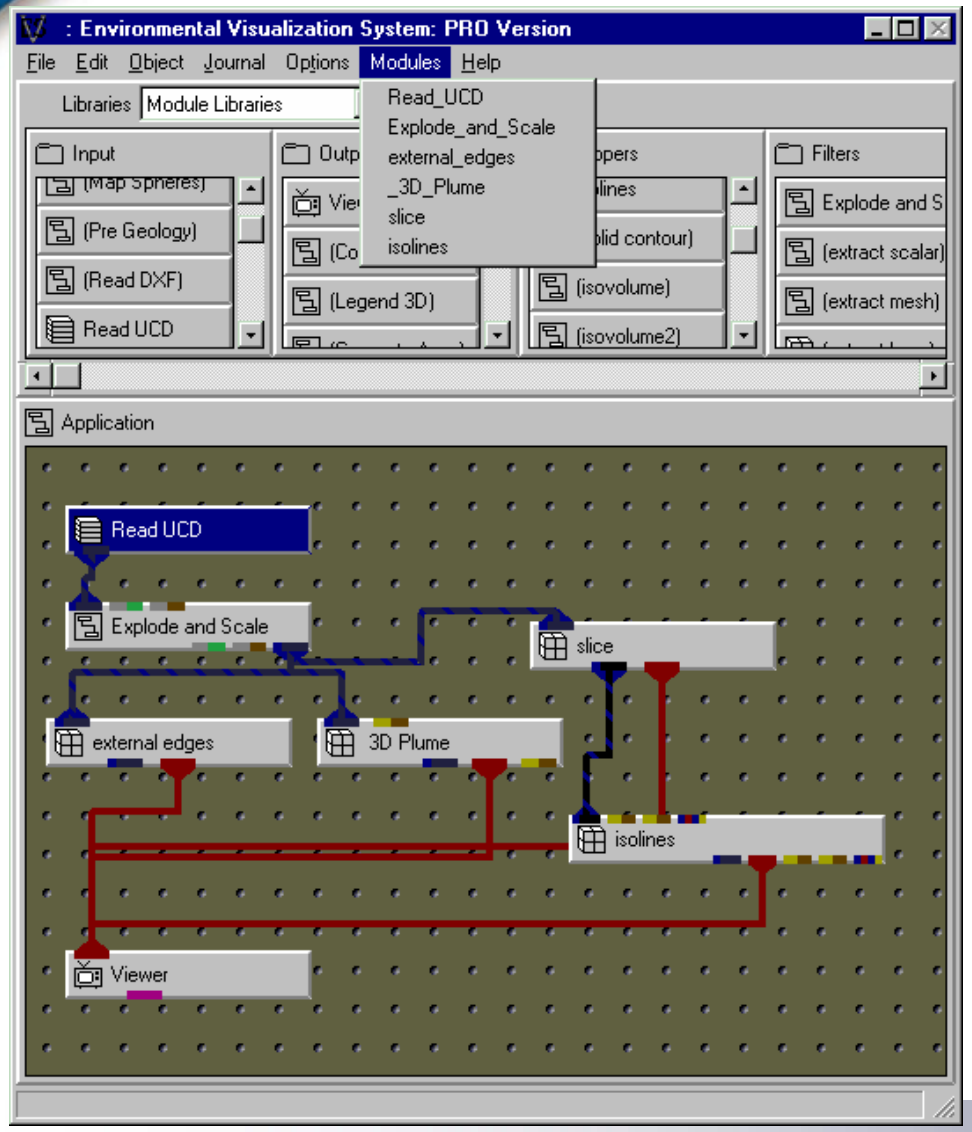
管道-过滤器风格的例子—Unix Shell

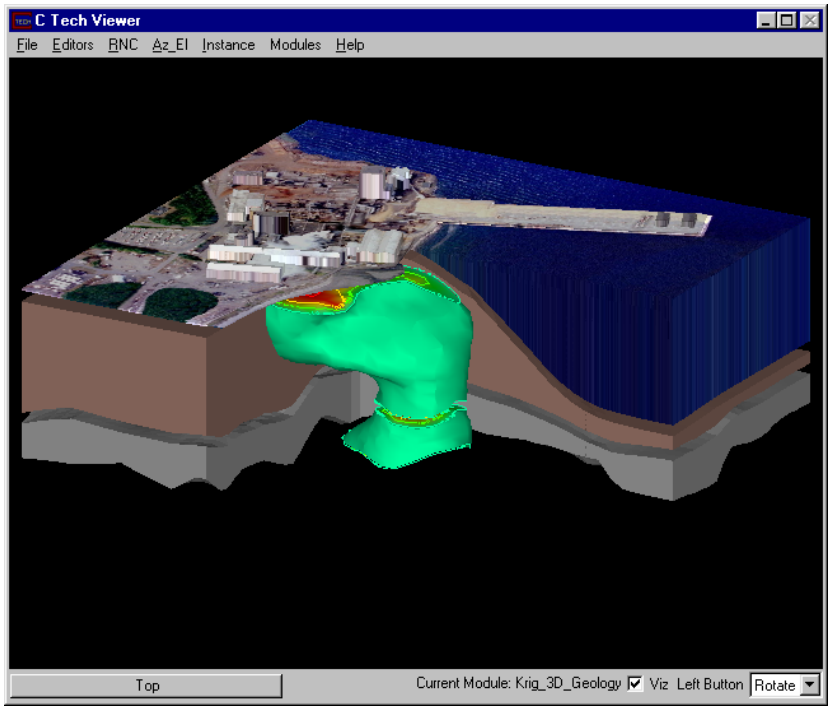
- Unix中的“管道与过滤器”

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

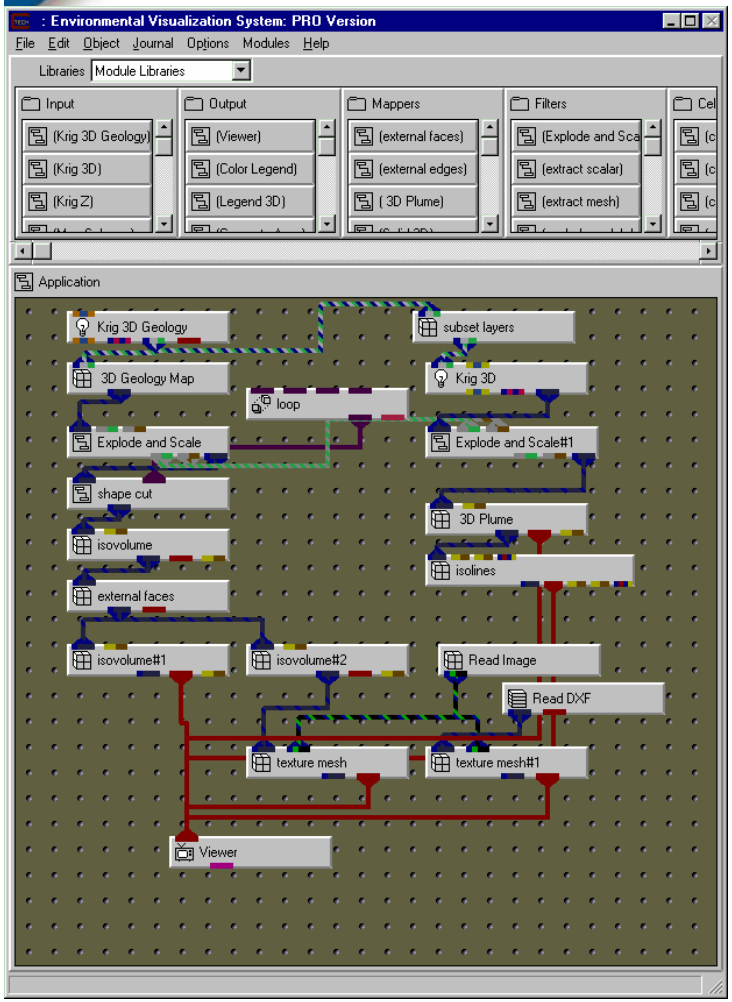


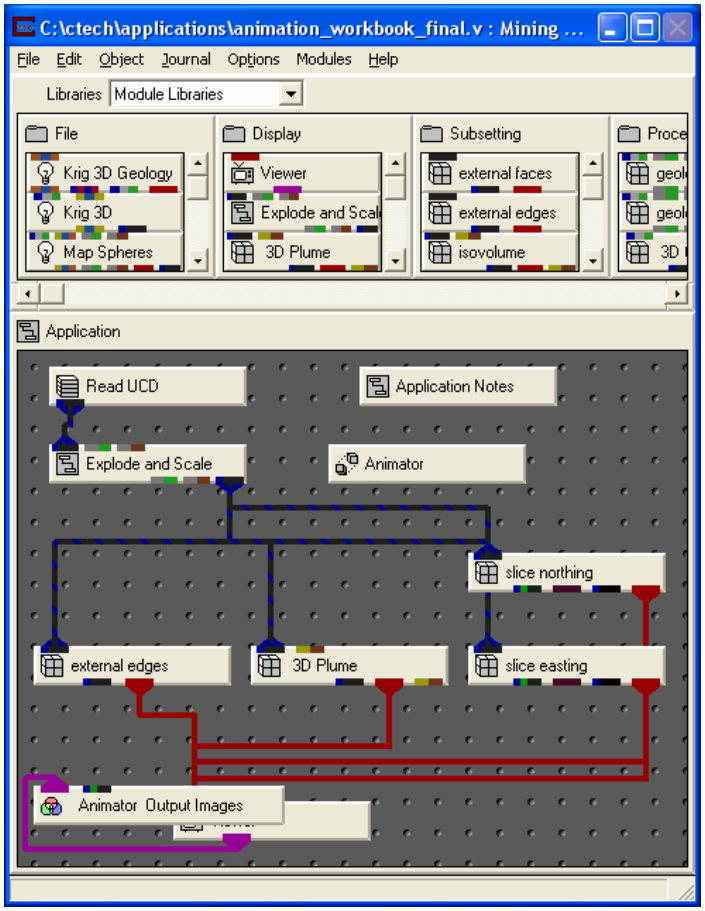
管道-过滤器风格的例子——三维地质建模软件 CTECH

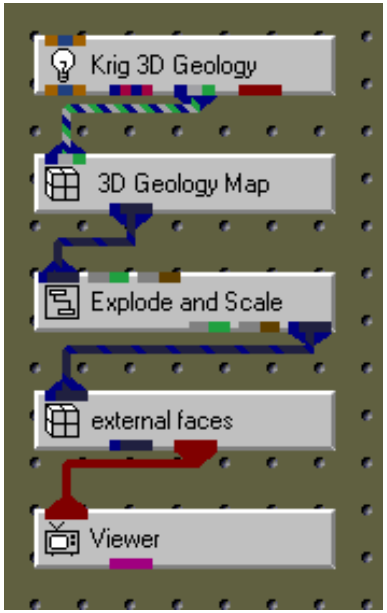




管道-过滤器风格的例子—CTECH

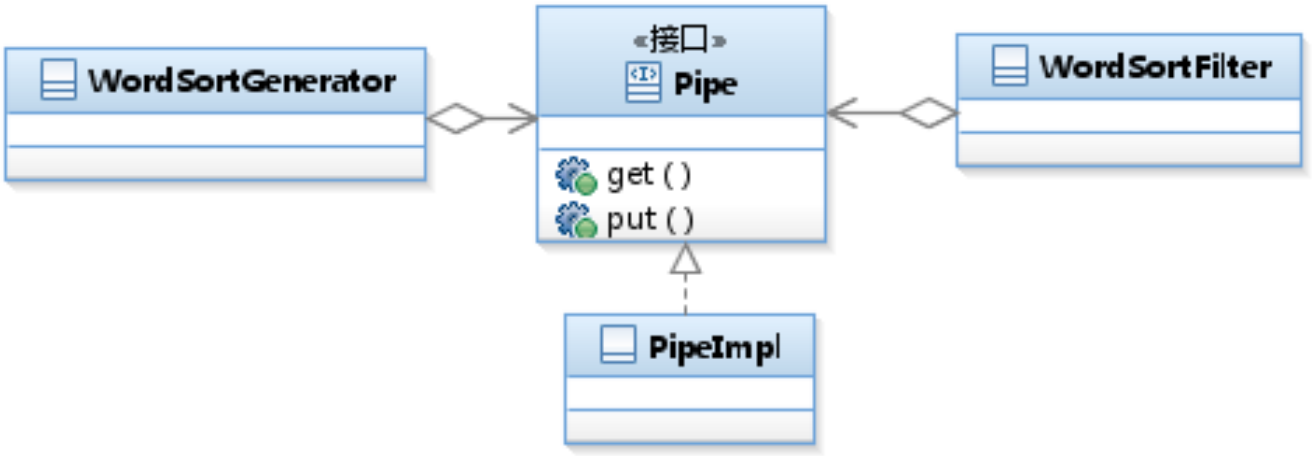




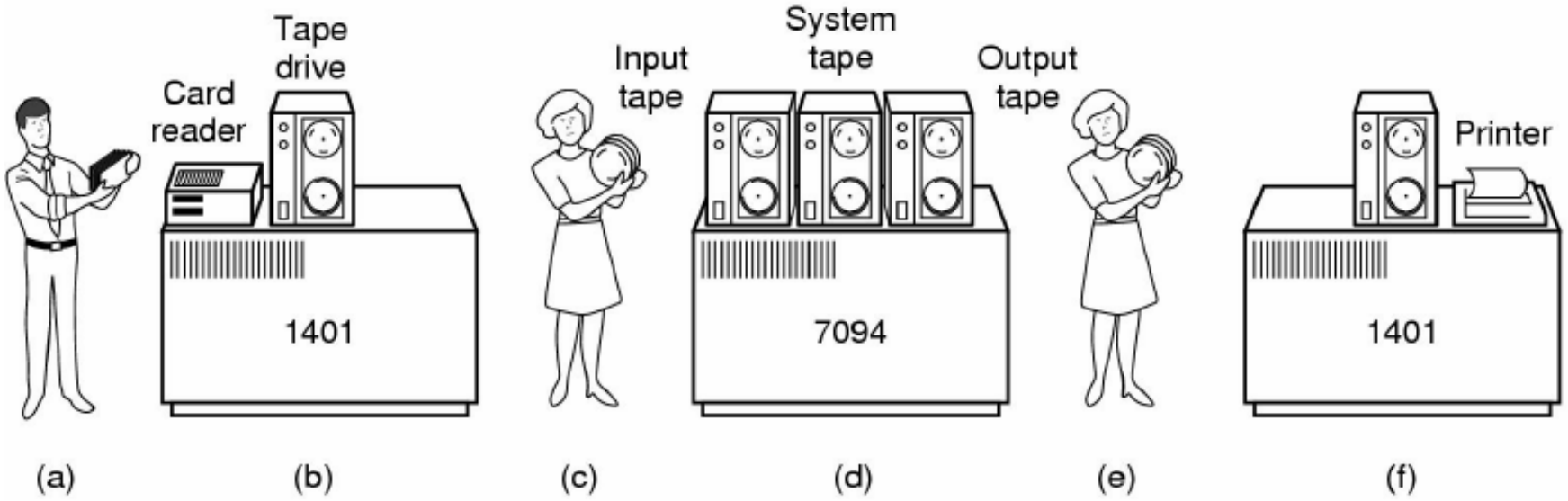


案例分析：一个单词排序程序

- **WordSortGenerator**：单词产生器负责从磁盘中读取文件
- **Pipe**：管道，负责数据的传输工作，将数据传送到单词排序过滤器WordSortFilter进行处理
- **WordSortFilter**：过滤器，能够对传入的数据流进行排序然后将结果写入文件



批处理风格（Batch Sequential Style）



将用户输入的纸带上的数据写入磁带

将磁带作为计算设备的输入，进行计算，得到输出结果

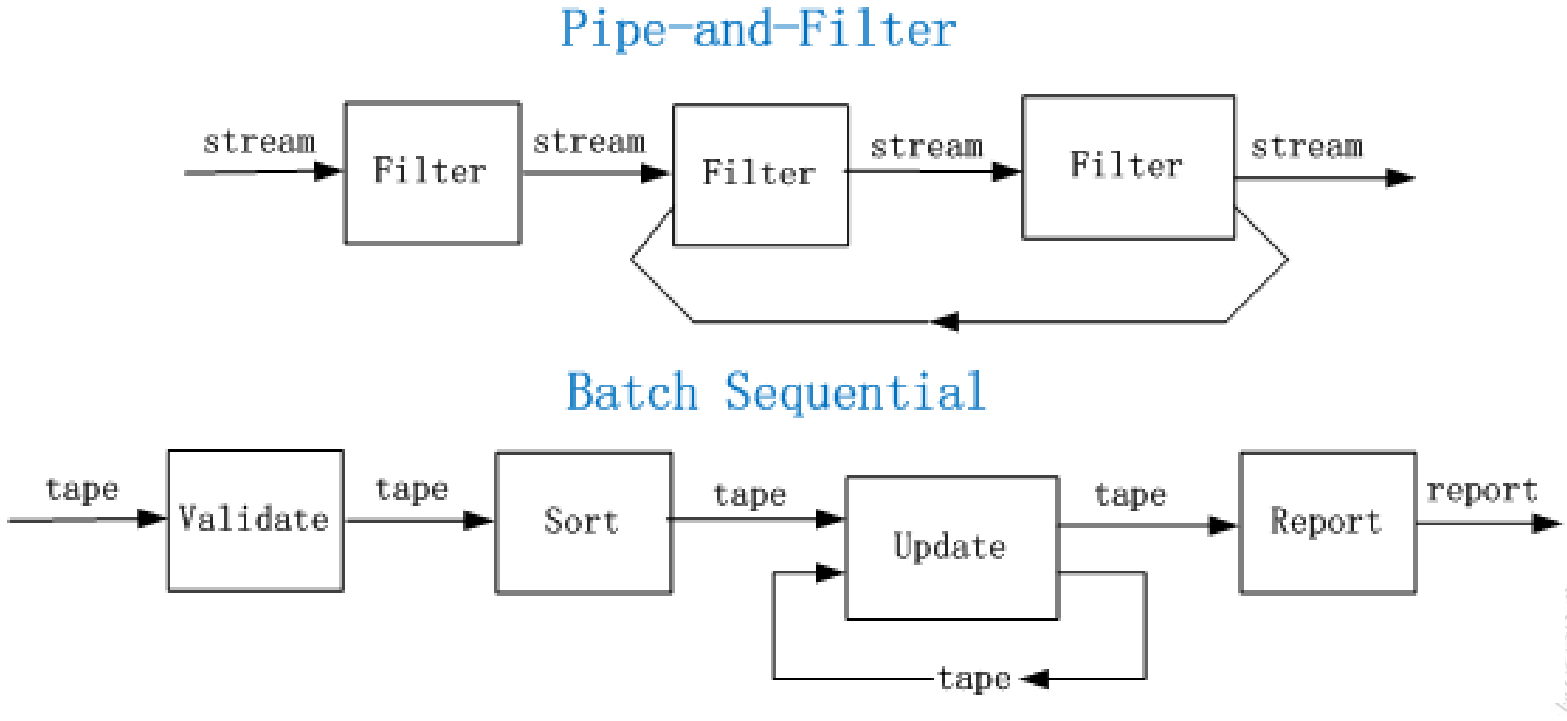
打印计算结果

管道-过滤器与批处理的比较

- **相似点：**

  Decompose task into fixed sequence of computations (把任务分解成为一系列固定顺序的计算单元)

  Interact only through data passed from one to another (彼此间只通过数据传递交互)



管道-过滤器与批处理的比较

- 不同点：

Pipe-and-Filter	Batch Sequential
• incremental(增量) • fine grained (构件粒度较小) • results starts processing (实时性) • concurrency possible (可并发)	• 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)

数据流风格的特点

- 优点：
  - 构件间仅通过数据传递进行交互，使得构件设计具有良好的隐蔽性和高内聚、低耦合的特点；
  - 只要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来，从而支持良好的软件复用特性（为什么？）；
  - 新的过滤器可以很方便地添加到现有系统中来，替换掉旧的过滤器，系统维护和增强系统性能相对简单；
  - 每个过滤器负责完成一个单独的任务，因此可与其它任务并行执行。
  - 此外，允许对一些如吞吐量、死锁等特性的分析。
- 缺点：
  - 通常导致系统进程成为批处理的结构，整体性能不高；
  - 不适合交互应用；
  - 每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性。

选择数据流风格的一些原则

- Task is dominated by the availability of data (任务由数据的可用性来主导)

- Data can be moved predictably from process to
- process (事先知道数据的确切流向)

- There may be a performance penalty in data flow styles, however this depends on many factors (数据的流动带来性能损耗)

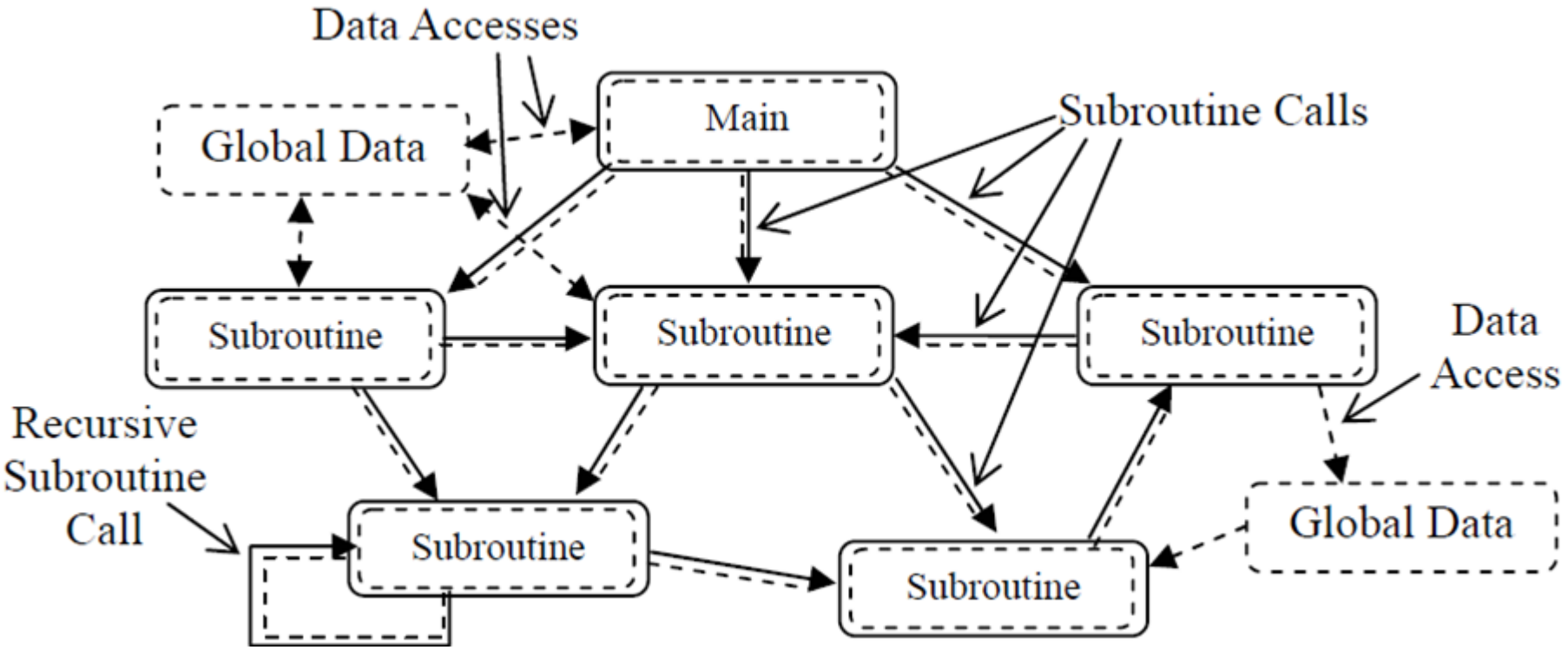
内容

- **3.1 概述**
- **3.2 数据流风格**
- **3.3 过程调用风格**
- 3.4 独立构件风格
- 3.5 层次风格
- 3.6 虚拟机风格
- 3.7 客户/服务器风格
- 3.8 表示分离风格
- 3.9 插件风格
- 3.10 微内核风格
- 3.11 SOA风格

常见过程调用风格

- 主程序-子过程风格
- 面向对象风格
- 层次风格（3.5节会专门讲解）

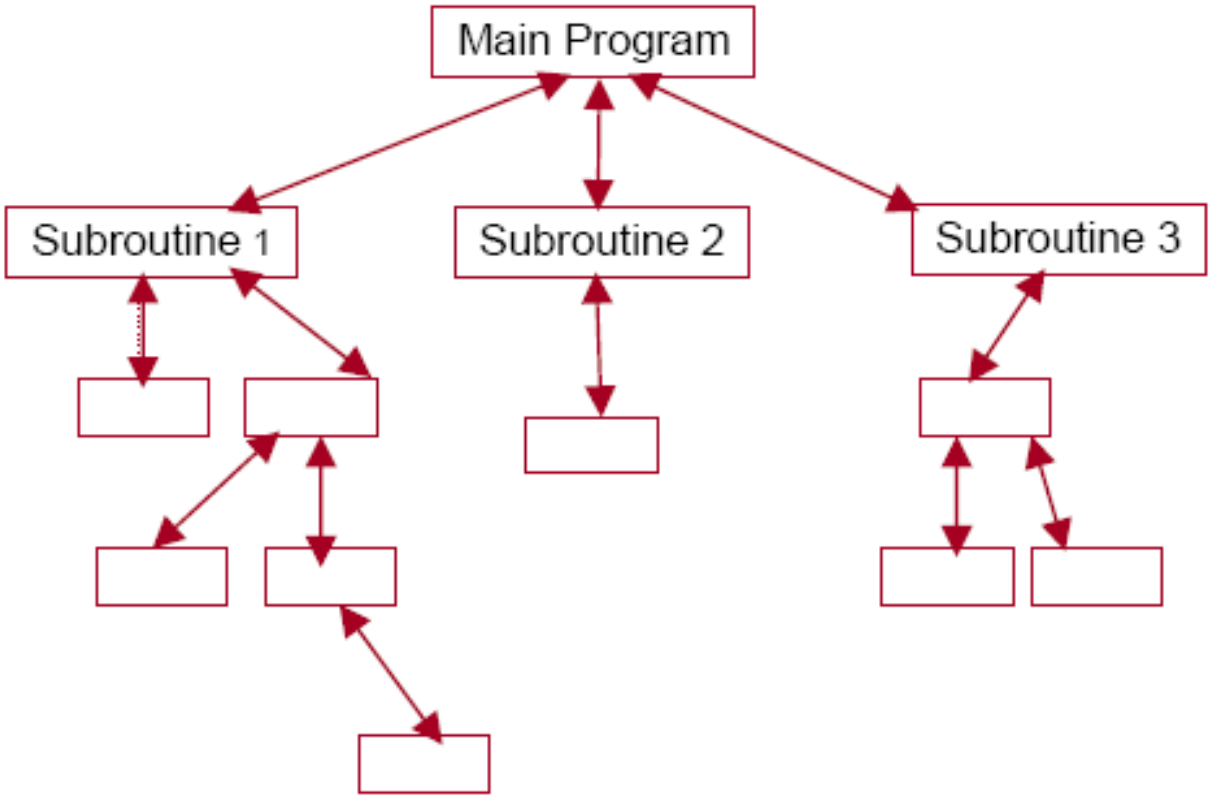
主程序-子过程风格


Data Accesses
Global Data
Main
Subroutine Calls
Subroutine
Subroutine
Subroutine
Data
Access
Recursive
Subroutine
Call
Subroutine
Subroutine
Global Data

回顾：非结构化的程序

- All the program code written in a single continuous main program (所有的程序代码均包含在一个主程序文件中).
- Disadvantages for large programs (缺陷):
  - difficult to follow logic (逻辑不清)
  - if something needs to be done more than once must be re-typed (无法复用)
  - hard to incorporate other code (难以与其他代码合并)
  - not easily modified (难于修改)
  - difficult to test particular portions of the code (难以测试特定部分的代码)

结构化程序：自上而下的功能设计
(Top-down functional design)

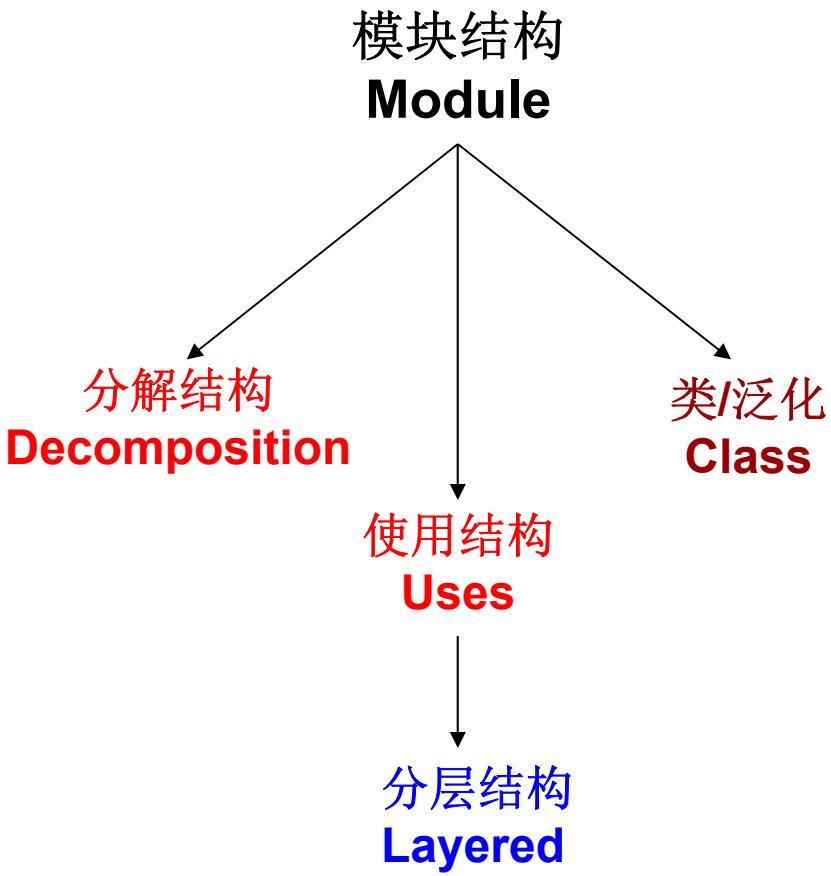


high-level subtasks

more primitive subtasks

individual HLL statements

回顾：模块结构


模块结构
Module
分解结构
Decomposition
使用结构
Uses
类/泛化
Class
分层结构
Layered

结构化程序：逐层分解
Hierarchical decomposition

- The system is designed from a functional viewpoint, starting with a high-level view and progressively refining this into a more detailed design (从功能的观点设计系统)
- This methodology is exemplified by Structured Design and step-wise refinement (结构化设计与逐步细化)
- Hierarchical decomposition (逐步分解):
  - Based on definition-use relationship (基于“定义—使用”关系)
  - Uses procedure call as interaction mechanism (用过程调用作为交互机制)
  - Correctness of a subroutine depend on the correctness of the subroutines it calls (主程序的正确性依赖于它所调用的子程序的正确性)

主程序-子过程风格的基本构成

- Component (构件：主程序、子程序)
  - Main program and subroutines, hierarchically decomposing a program
- Connector(连接器：调用-返回机制)
  - Call-return mechanism, each component get control and data from its parent and pass it to children.
- Topology(拓扑结构：层次化结构)
  - Hierarchical structure
- 本质：将大系统分解为若干模块(模块化)，主程序调用这些模块实现完整的系统功能。

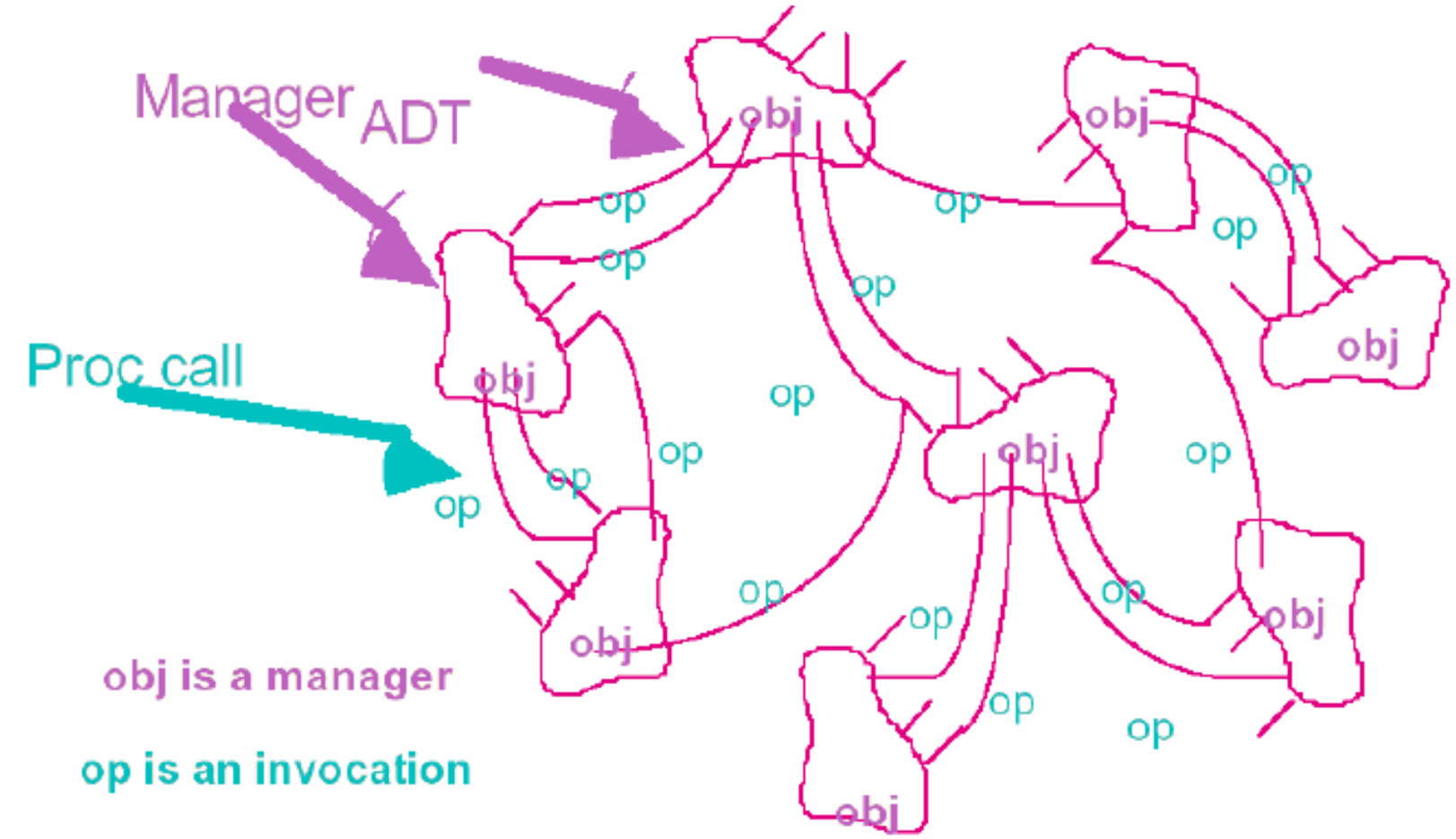
主程序-子过程风格的优点与缺点

- 优点:
  - This has proved to be a highly successful design methodology. It has allowed the development of large programs (100,000 lines or more of source code) (已被证明是成功的设计方法，可以被用于较大程序)
- 缺点:
  - However, as program size increases beyond this point (100,000 lines), the approach performs poorly. (程序超过**10**万行，表现不好)
  - We observe that code development becomes too slow and that it becomes increasingly difficult to test the software and guarantee its reliability. (程序太大，开发太慢，测试越来越困难)

面向对象风格
（Object-Oriented Style）

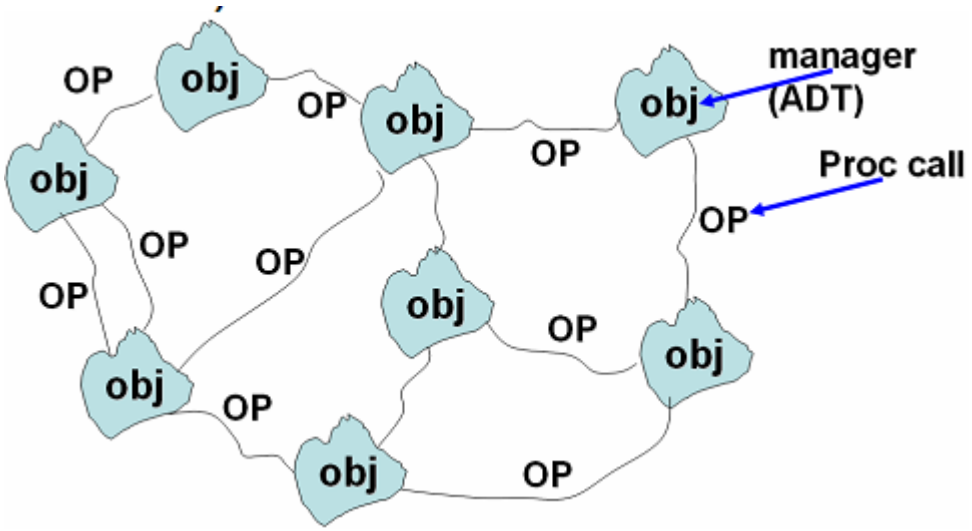
- The system is viewed as a collection of objects rather than as functions with messages passed from object to object. (系统被看作对象的集合)
- Each object has its own set of associated operations (每个对象都有一个它自己的功能集合。数据及作用在数据上的操作被封装成抽象数据类型——对象)

Data Abstraction or Object-Oriented



Object-Oriented风格基本构成

- Component: classes and objects (OO风格的构件是：类和对象)
- Connectors: objects interact through function and procedure invocations. (连接件：对象之间通过功能与函数调用实现交互)



面向对象架构风格特点

- 优点：面向对象的优点（大家很熟悉，略）
- 缺点：
  - Management of many objects: Need structure on large set of definitions (管理大量的对象：怎样确立大量对象的结构)
  - In order for one object to interact with another (via procedure call) it must know the identity of that object. (必须知道对象的身份)
  - 对比：在管道-过滤器系统中，一个过滤器无需知道 其他过滤器的任何信息
  - Inheritance introduces complexity and this is undesirable, especially in critical systems (继承引起复杂度，关键系统中慎用)

数据流风格与调用返回风格的对比分析

——以“管道-过滤器”和“主程序-子过程”风格为例

	管道-过滤器	主程序-子过程
Control	Asynchronous, data-driven (异步/数据驱动)	Synchronous, Blocking (同步/阻塞)
Semantics	Functional (按功能的扁平分解)	Hierarchical (层次分解)
Data	Streamed (流)	Parameter / return value (参数/返回值)

思考题

- 结合所给的单词排序程序，分析数据流风格的特点，给出可能的其他应用。
- 为什么说管道-过滤器风格可以支持良好的软件复用特性？