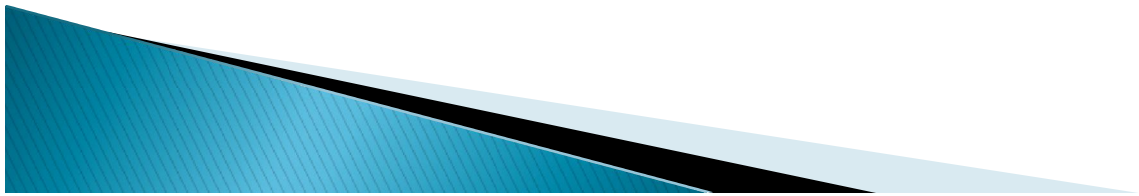
Chapter 4

典型软件体系结构风格及其应用模式
Classic Software Architecture Styles&Patterns

Highlights:

- What's Software Architecture Styles/Patterns?
- Pipe/Filter Style/Pattern
- Layer Style/Pattern
- Event-based, Implicate Innovation Style/Pattern
- Repository Pattern/Pattern



4.1 软件体系结构风格？

- **软件体系结构风格**

  经过工程实践提炼总结得到，对某一类型或某一应用领域的软件系统具有普遍适用性或借鉴性的体系结构风格。

  - 每种体系结构风格包含对系统结构组成元素即构件的描述，以及构件间的组合机制的描述即连接件。
  - 体系结构风格可以帮助我们快速分析与设计一个系统的体系结构，体现了一种系统体系结构的重用。
  - 设计人员在选择一种风格时，可以根据每种风格的特性来进行抉择需要参照的系统体系结构。

在实际中，风格一般用于理论界，而工程界则偏爱于使用模式这一词。本课程中，我们用风格专指具有一般意义的体系结构，而用模式指一种体系结构风格在工程实践中的不同应用形式。

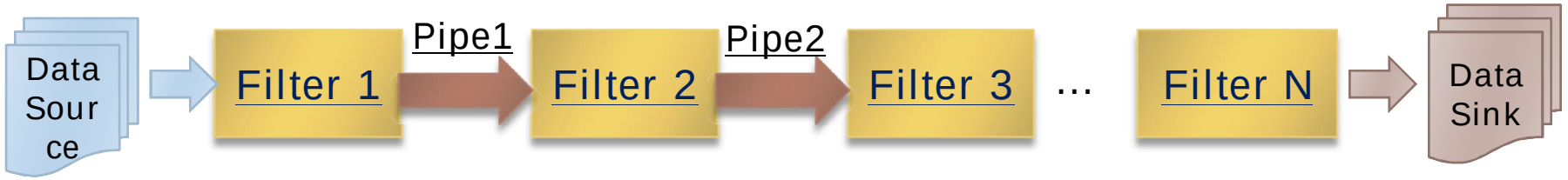
4.1 软件体系结构风格?

- **典型的软件体系结构风格**
  - 管道-过滤器体系结构风格
  - 基于事件的隐式调用体系结构风格
  - 分层体系结构风格
  - 仓库风格
  - 异构系统合成风格

4.2 管道-过滤器风格及其应用模式

● 管道—过滤器（Pipe-Filter）体系结构风格

管道—过滤器体系结构风格为处理数据流的软件系统架构提供了一种参考结构。它是由过滤器和管道组成的。每个处理步骤都被封装在一个过滤器组件中，数据流通过相邻过滤器之间的管道进行传输。每个过滤器可以单独修改，功能单一，并且它们之间的顺序可以进行配置。



Fig：Pipe/Filter 体系结构风格示例

- **构件类型： 过滤器（Filter）----数据处理构件**
- **连接件类型：管道（Pipe）----过滤器间的连接件**

4.2 管道-过滤器风格及其应用模式

- Context（适用场景）
  - 数据流处理软件系统的一种普遍体系结构风格
    - 为处理数据流的系统提供了一种结构.
    - 数据流的每个处理步骤，封装实现在一个过滤器(Filter)组件中.
    - 相邻过滤器之间通过管道(Pipe)交互.
    - 重组过滤器可以建立相关的系统族（不同数据处理顺序或功能）.
  - 典型系统案例
    - 高级语言编译器系统
    - Windows CMD/Unix Shell命令解释器系统
    - ...

4.2 管道-过滤器风格及其应用模式

- 典型系统案例

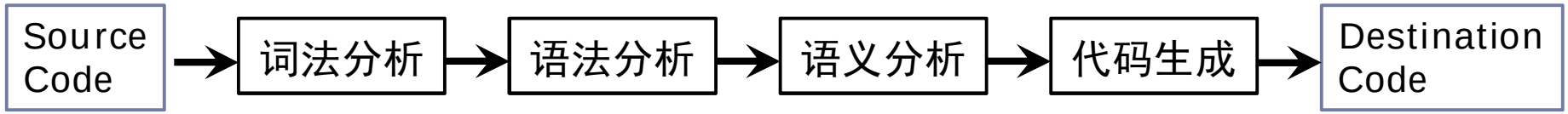


Fig1:基于Pipe—Filter的编译器系统

```
% ps auxwww | grep dutoit | sort | more
dutoit   19737  0.2  1.6 1908 1500 pts/6    O 15:24:36  0:00 -tcsh
dutoit   19858  0.2  0.7  816  580 pts/6    S 15:38:46  0:00 grep dutoit
dutoit   19859  0.2  0.6  812  540 pts/6    O 15:38:47  0:00 sort
```

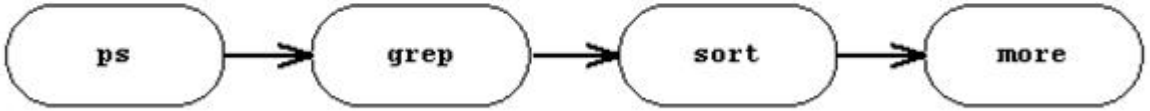
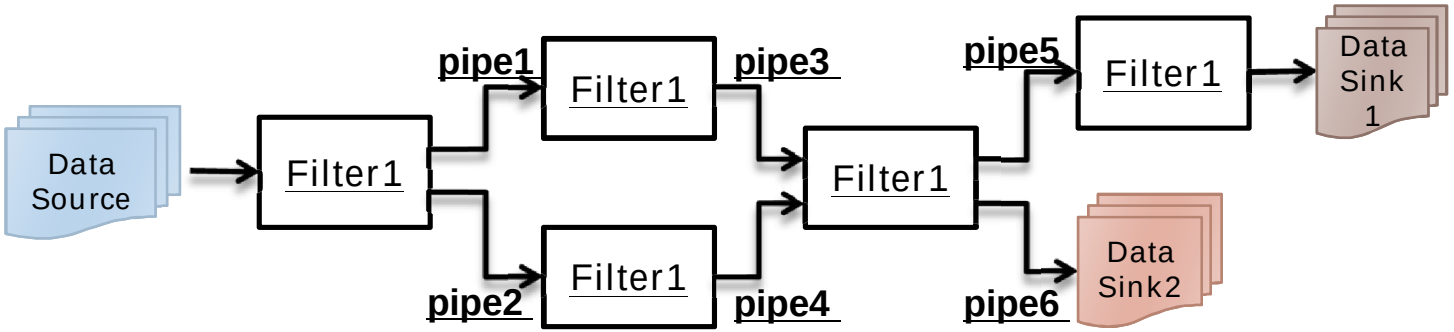


Fig2: Unix Shell命令解释器示例

4.2 管道-过滤器风格及其应用模式

- Pipe-Filter 体系结构风格的特点
  - 功能特性
    - 处理或者转换输入数据流.
    - 对数据流的处理可以容易地分成几个独立的处理步骤.
  - 非功能特性(质量特性)
    - 系统的升级要求可以通过替换/增加/重组过滤器实现. 有时甚至由使用者完成操作（系统运行时配置更新）.
    - 不同的处理步骤不共享信息（过滤器构件间松耦合）.

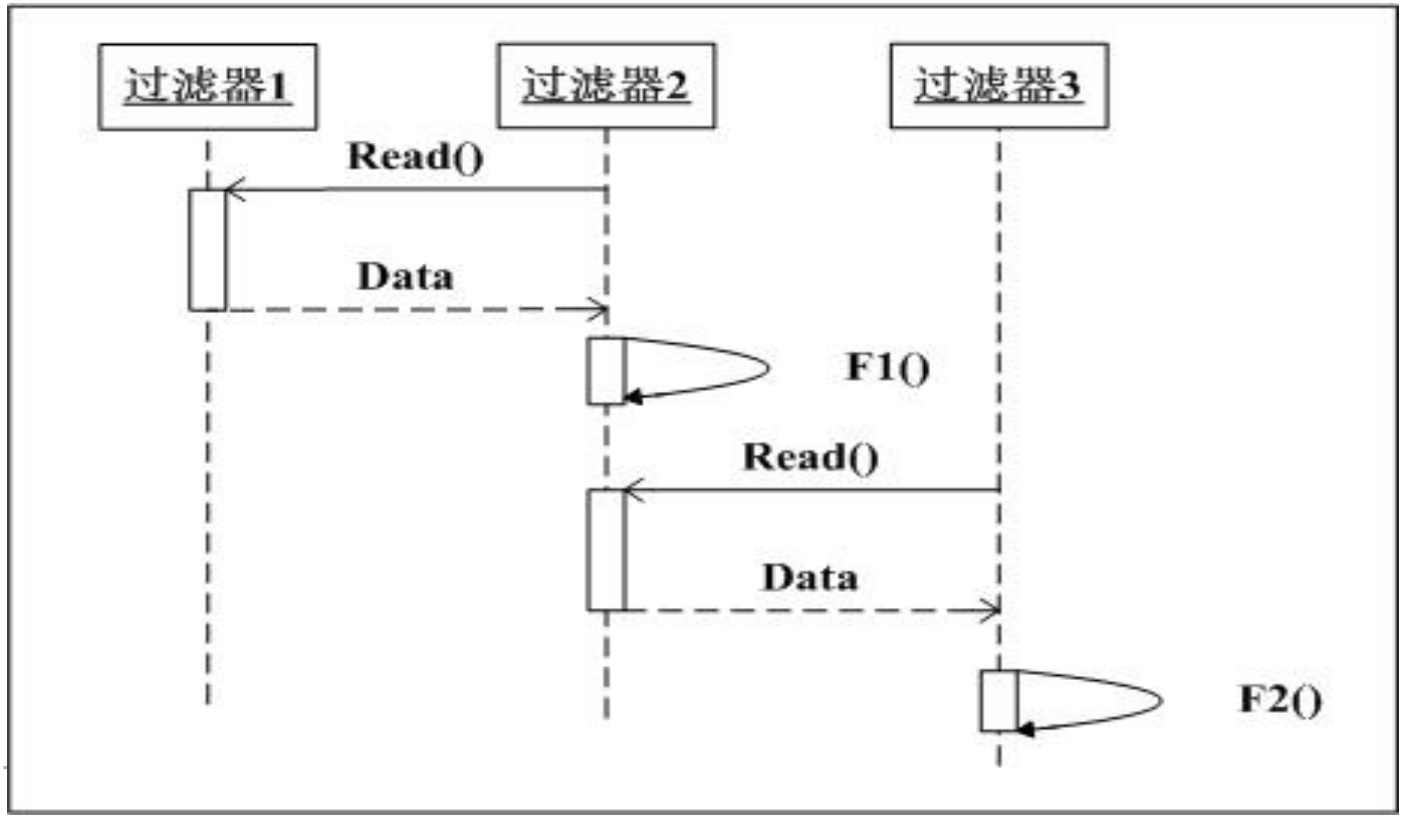


4.2 管道-过滤器风格及其应用模式

- **过滤器(Filter)是数据流的处理功能单元。根据相邻过滤器间交互方式，分为：**
  - 被动过滤器(Passive Filter)
    - 过滤器从上一个相邻过滤器采用拉入(pull)方式读取数据.
    - 过滤器采用压出(push)方式输出数据给下一个相邻过滤器.
  - 主动过滤器(Active Filter)
    - 过滤器同时支持以上两种方式(Pull/Push)读写数据.

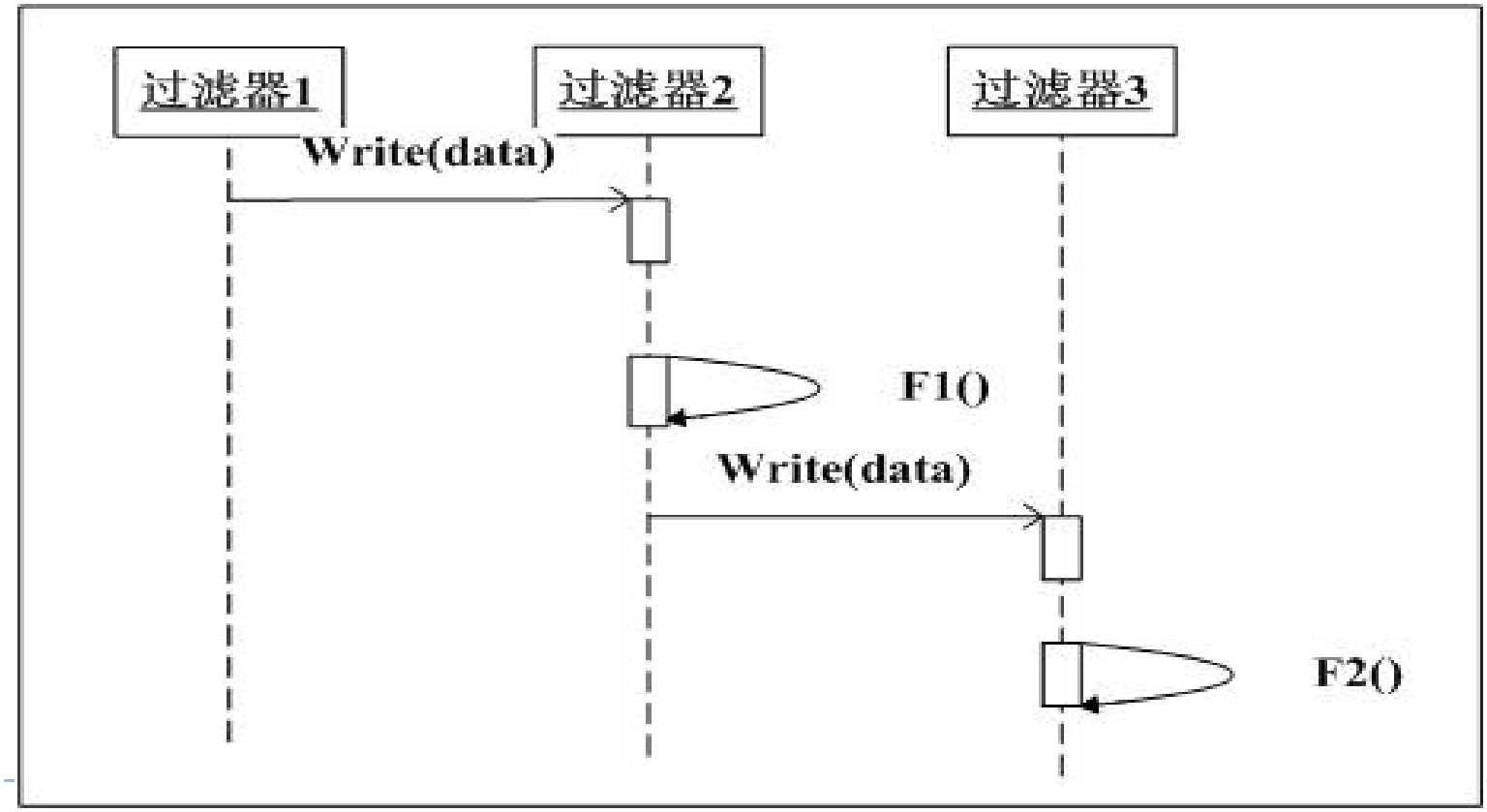
4.2 管道-过滤器风格及其应用模式

- 被动过滤器(Pull)



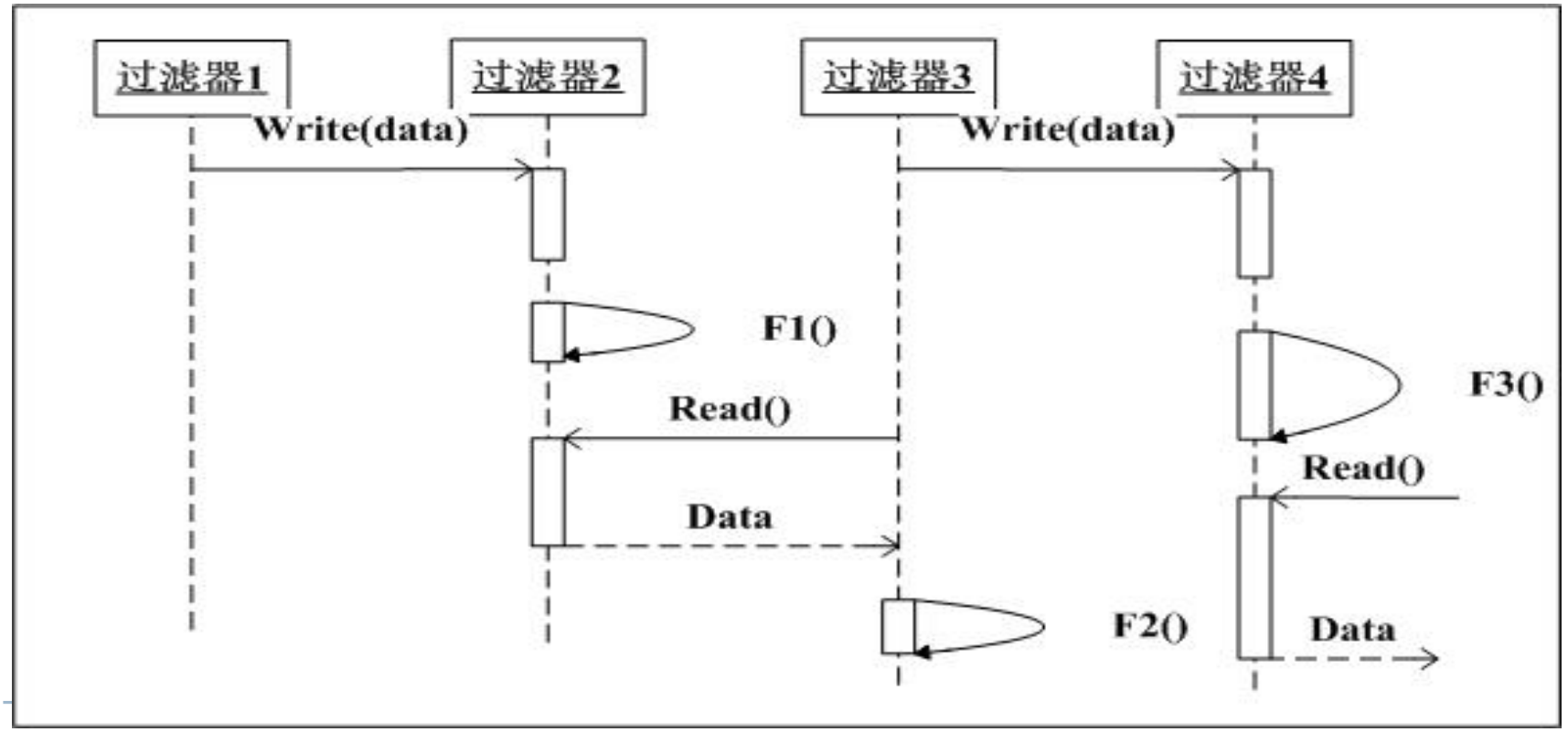
4.2 管道-过滤器风格及其应用模式

- 被动过滤器(Push)



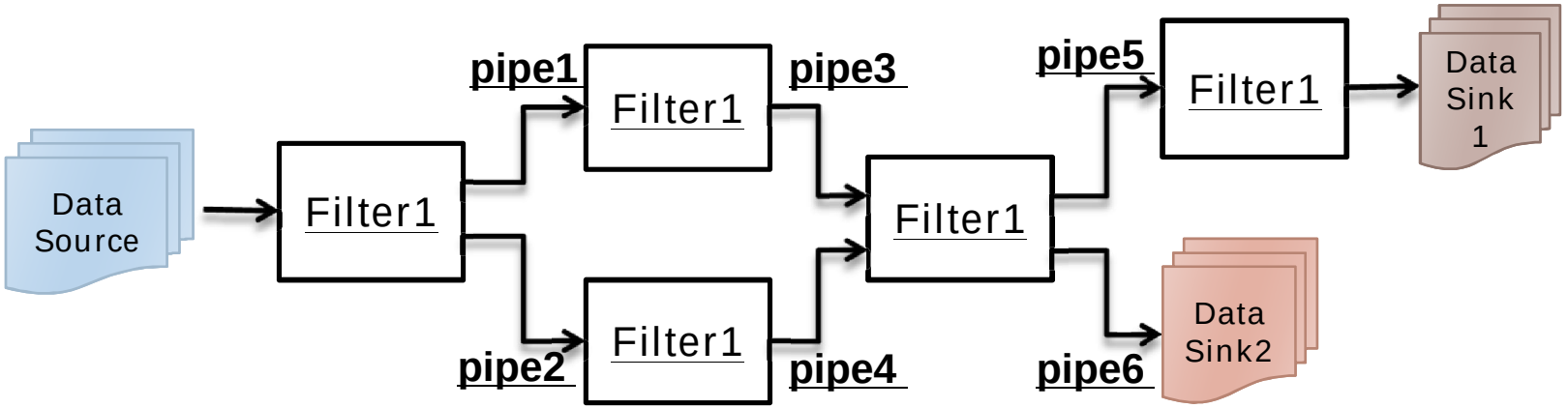
4.2 管道-过滤器风格及其应用模式

- 主动过滤器



4.2 管道-过滤器风格及其应用模式

- **管道(Pipe)：实现过滤器构件间交互的连接件。**
  - 如果管道连接两个主动过滤器，那么管道需要进行缓冲和同步.
  - 管道可以实现数据在过滤器之间的数据转换，将一个过滤器的输出数据格式转换为其后接的过滤器的数据输入格式.



4.2 管道-过滤器风格及其应用模式

- **优点：**
  - **高内聚和低耦合**
    - **高内聚：**过滤器是独立运行的系统构件。除了输入和输出外，每个过滤器不受任何其他过滤器运行的影响。在设计上，过滤器之间不共享任何状态信息。
    - **低耦合：**每个过滤器对其相邻的过滤器是“无知”的。它唯一关心的是其输入数据，然后进行加工处理，最后产生数据输出。
  - **通过过滤器的增加/移除/重组可实现数据流处理系统的灵活性/可扩展性(Scalability)；**
  - **过滤器构件具有可重用性(Reusability)；**
  - **有利于系统的维护与更新（Evolution）；**
  - **可支持局部步骤的并行处理以提高效率；**

4.2 管道-过滤器风格及其应用模式

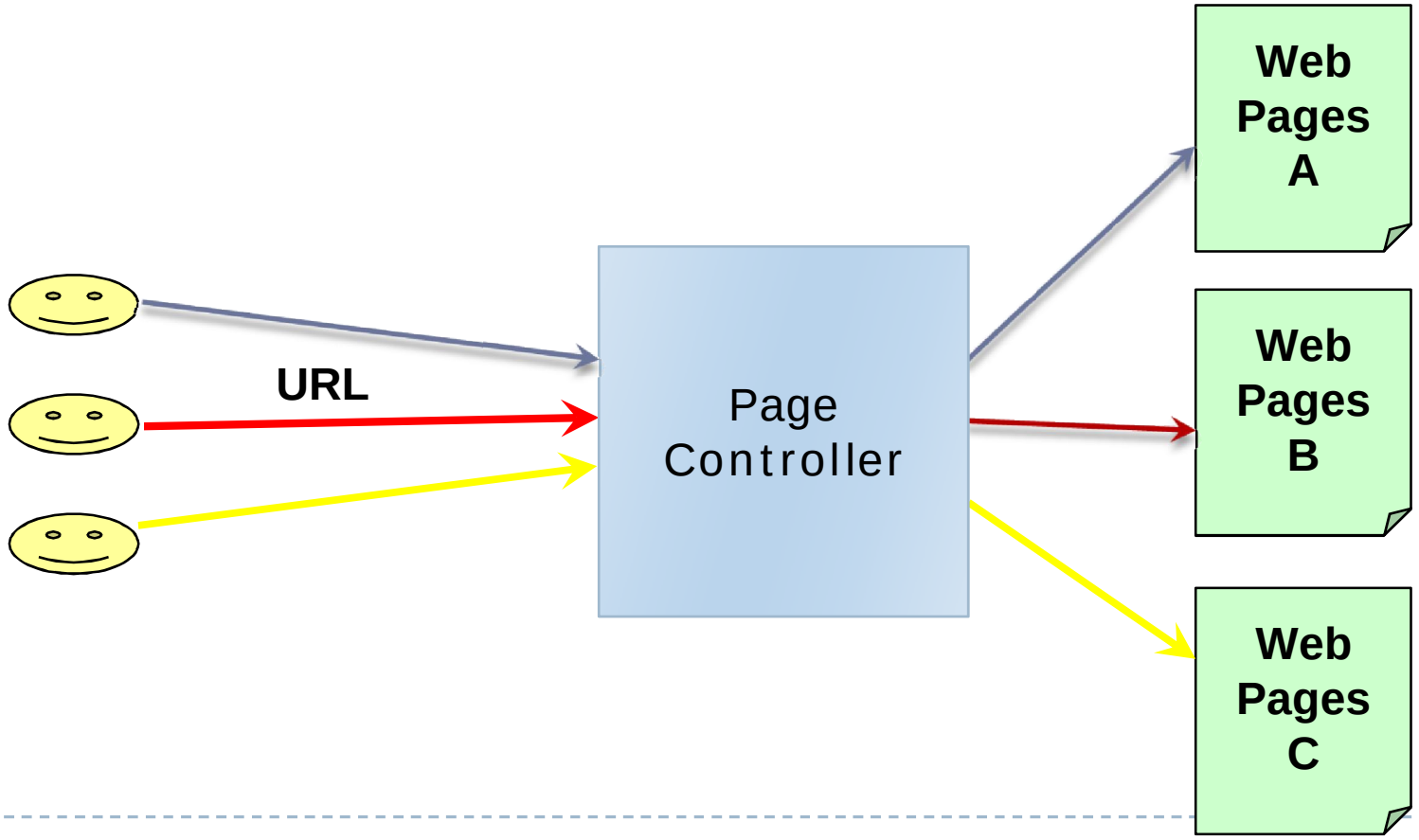
- **缺点：**
  - 增量式处理数据，存在效率问题；

    管道过滤器风格在数据流处理上是串行增量式的处理过程，因此存在一定的数据处理周期和效率上的限制；
  - 数据格式转换的问题：数据转换额外开销；

    过滤器的输出流OUTn经由管道作为下一个过滤器的输入数据流INn+1，如果OUTn与INn+1之间存在数据格式不一致的问题，则管道往往需要完成数据的转换工作。这会引起一定的额外开销。
  - 不适合交互式应用系统；

4.2 管道-过滤器风格及其应用模式

- **应用案例：**PageController



4.2 管道-过滤器风格及其应用模式

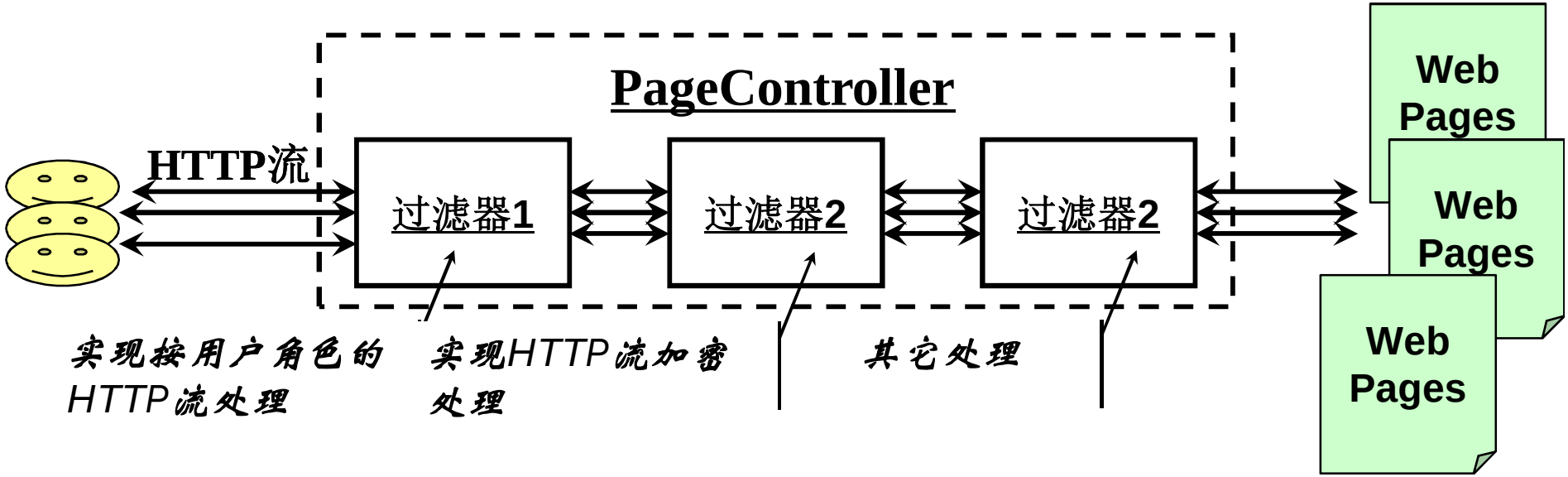
- **应用案例：PageController**
  - **功能需求：**

    不同用户通过一个统一的URL访问某Web应用系统时，要求该系统能根据不同用户的属性（如角色权限、用户类别）与系统配置自动引导显示不同的页面给用户。
  - **非功能需求：**

    系统的引导规则具有不确定性，在系统维护期可能会发生变化。要求系统可通过动态配置定义引导规则；

4.2 管道-过滤器风格及其应用模式

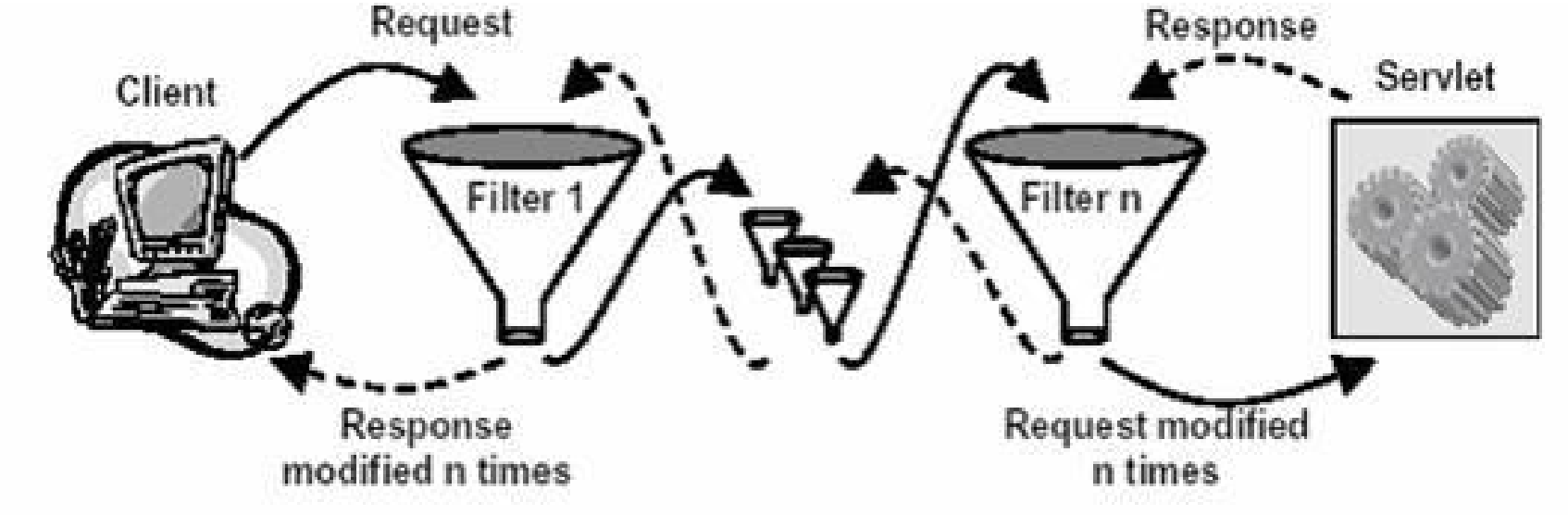
- **应用案例：**PageController



- 使用不同过滤器实现对**HTTP**流的不同控制处理；
- 根据需要可灵活添加新的过滤器实现新的**HTTP**流控制；

4.2 管道-过滤器风格及其应用模式

- **应用案例：**PageController
  - Java Filter



4.2 管道-过滤器风格及其应用模式

- **应用案例：**PageController
  - 实现示例（Java）

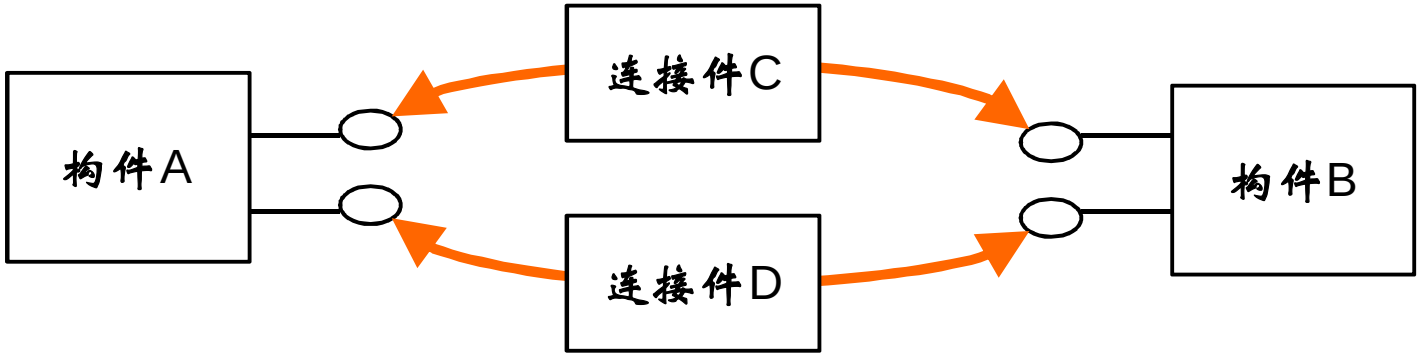

```
public class SecurityFilter implements Filter {
  …
  public void doFilter(ServletRequest req, ServletResponse res, FilterChain
                                chain) throws IOException, ServletException {
    …
  }
  …
}
```

详细参考： *http://www.ibm.com/developerworks/cn/java/l-j2ee-filter/index.html*
http://msdn2.microsoft.com/zh-cn/architecture/ms998532.aspx

4.3 基于事件的隐式调用体系结构风格

- **直接耦合 vs 间接耦合**

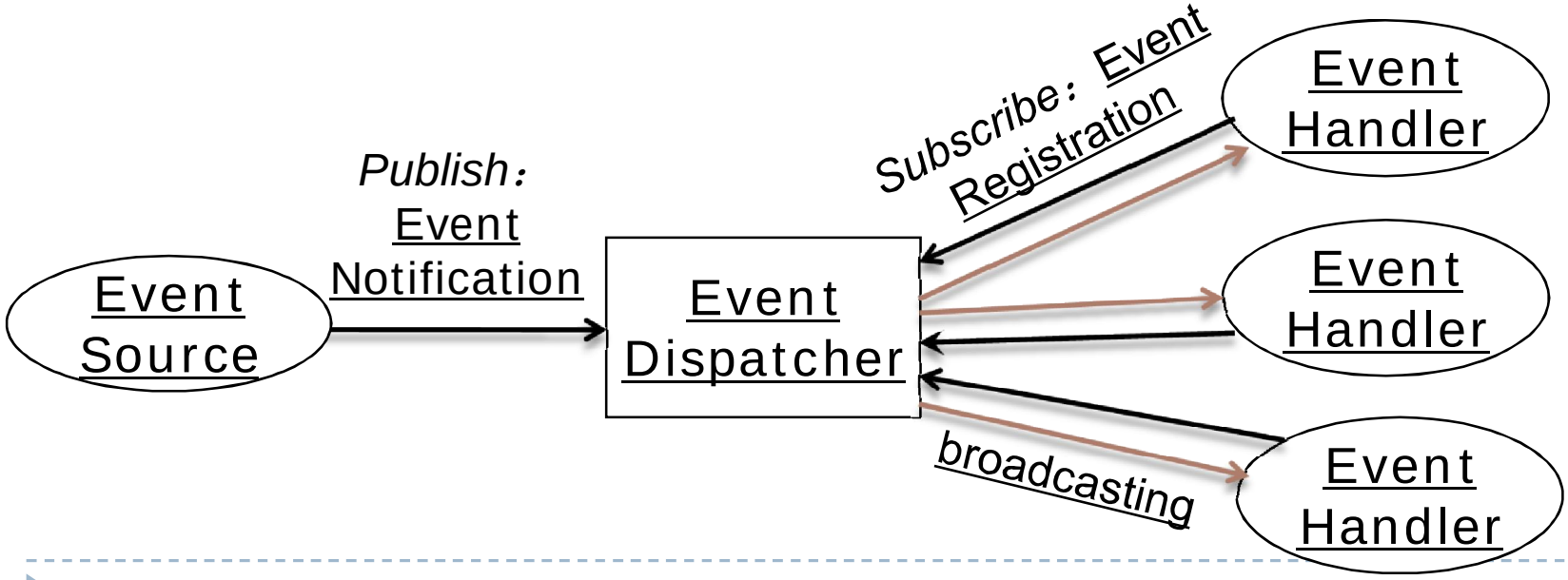
*直接调用：过程调用、**RPC**。。。*



间接调用

4.3 基于事件的隐式调用体系结构风格

- 基于事件的（隐式调用风格）系统结构风格中，构件间不采用直接耦合方式交互，而是通过事件机制交互。系统中的事件处理者构件可以通过事件注册来订阅它所关心的事件相关联。当某一事件发生时，系统会通知所有与这个事件相关联的事件处理者构件，即一个事件的激发导致了事件处理者构件与事件源构件间的隐式地交互。



4.3 基于事件的隐式调用体系结构风格

- **构件类型**:
  - 事件源(Event Source)构件
  - 事件处理者(Event Handler)构件
  - 事件分发者(Event Dispatcher)构件
- **连接件类型**: **事件对象**(Event)
  - An event is a notification that occurs in response to an action, such as a change in state, or as a result of the user clicking the mouse or pressing a key while viewing the document.

4.3 基于事件的隐式调用体系结构风格

- Context（适用场景）
  - 松散耦合的异步运行系统
    - 对事件的处理无顺序要求的系统
    - 事件的处理要求具有很好的灵活性
    - 非集中式控制的实时响应软件系统
  - 典型系统案例
    - 图形用户界面系统
    - 某些监控系统
    - 现代高级语言的异常处理
    - ……

4.3 基于事件的隐式调用体系结构风格

- 典型系统案例：高级语言的异常处理机制

```
public class ProcessFile {
   public static void Main() {
      try {
         StreamReader sr = File.OpenText("data.txt");
         Console.WriteLine("The first line of this file is ",
                           sr.ReadLine());
      }
      catch(IOException e){
        Console.WriteLine(“The file is not found!”);
      }
      catch(Exception e) {
         Console.WriteLine("An error occurred: ", e);
      }
   }
}
```

Event Source

Event Register

Event Handler 1

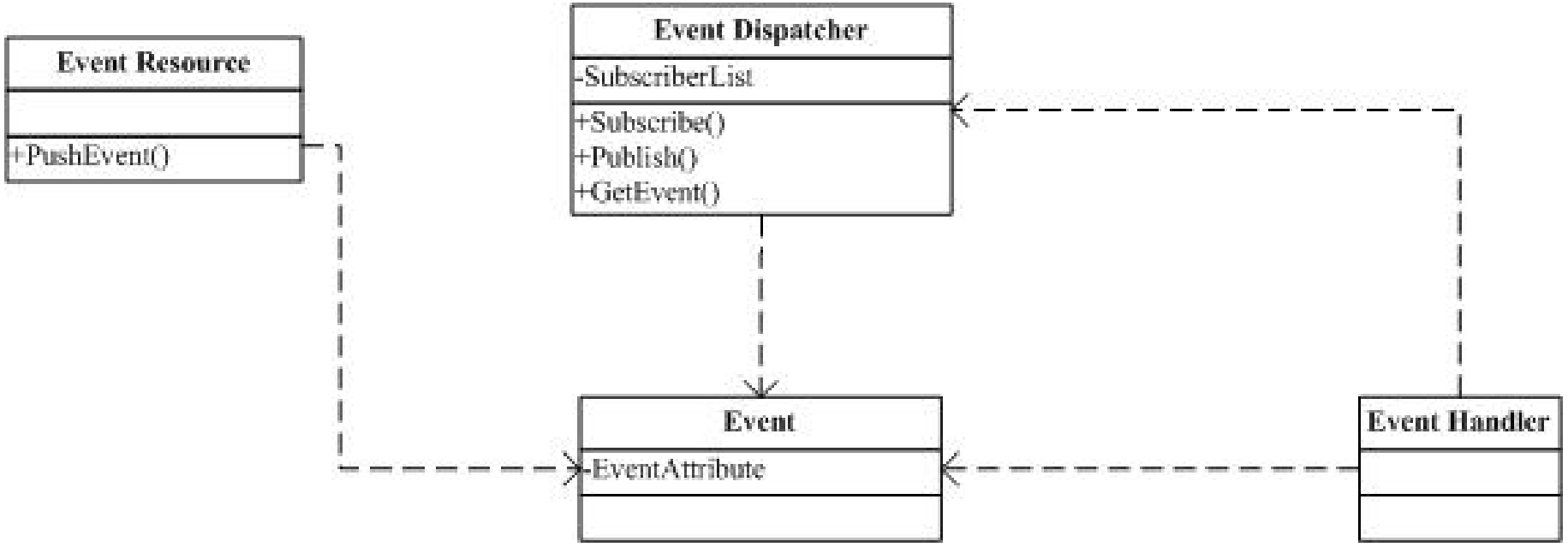
Event Register

Event Handler 2

Where is the Event Dispatcher?

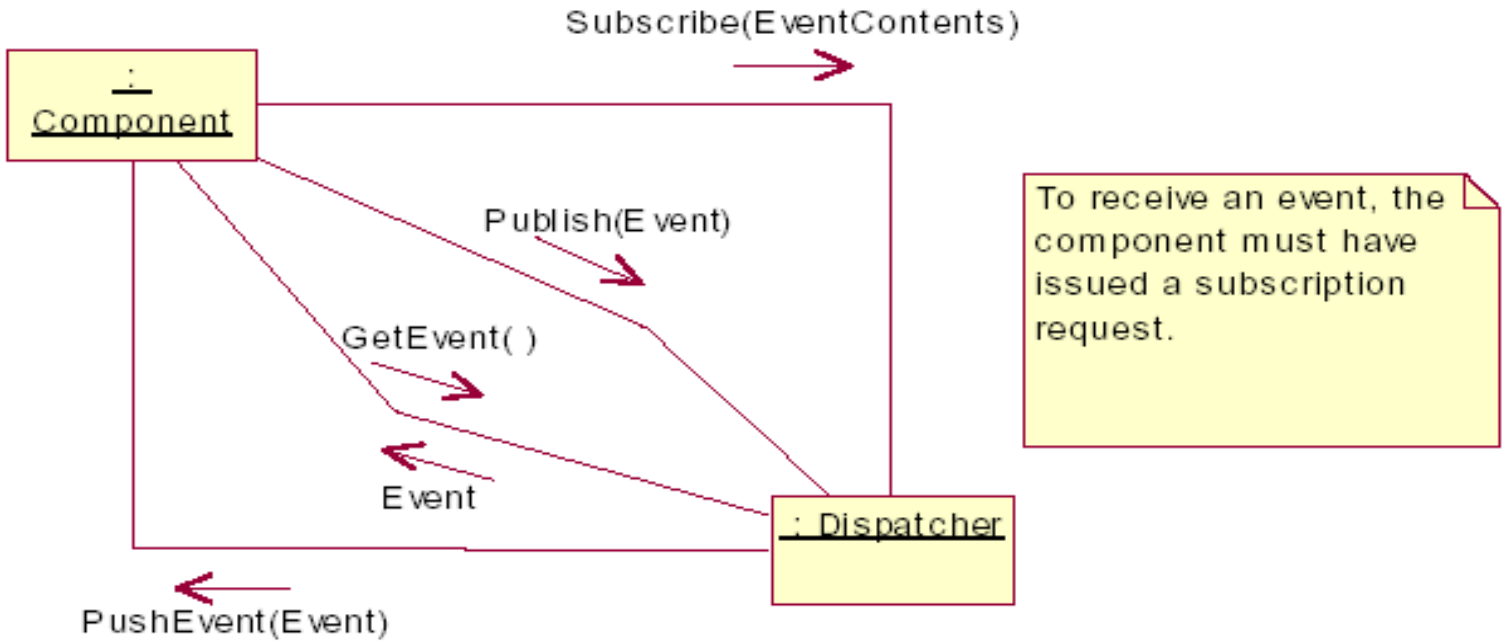
4.3 基于事件的隐式调用体系结构风格

- Model
  - Logic View



4.3 基于事件的隐式调用体系结构风格

- Model
  - Scenario View

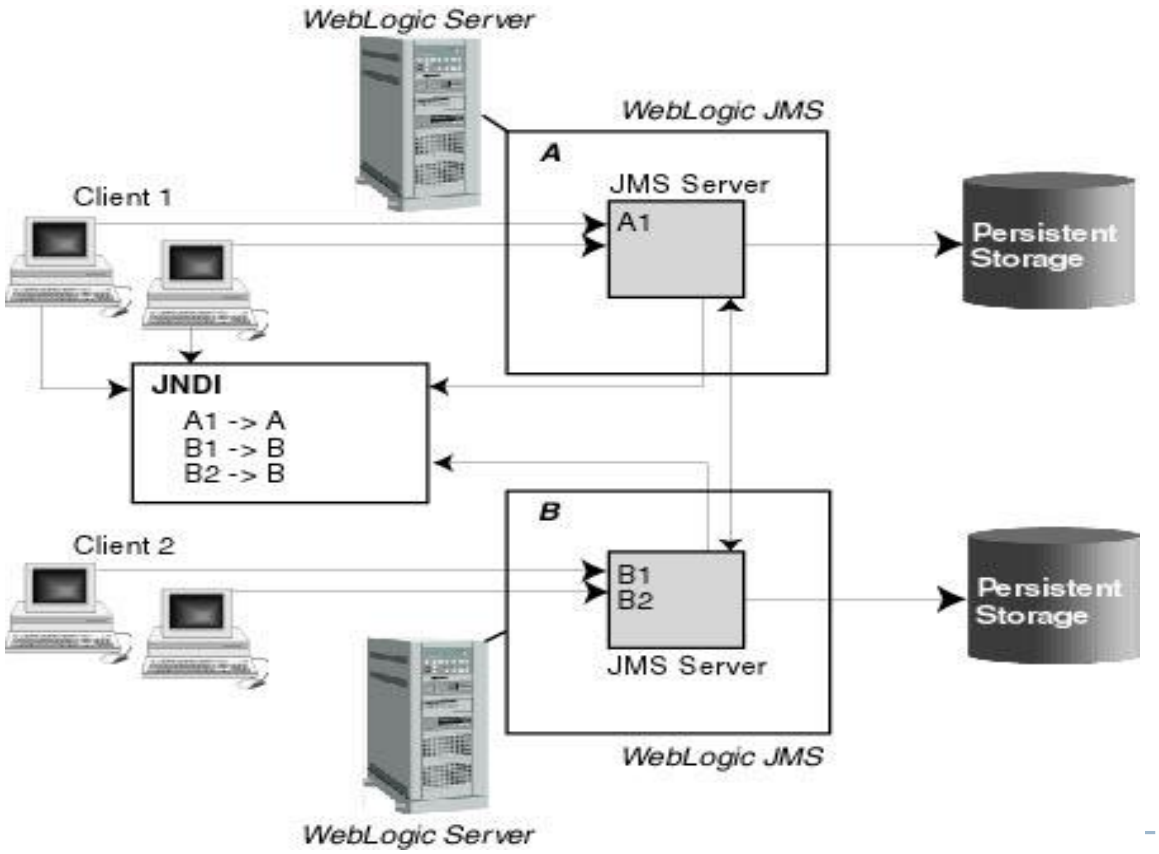


4.3 基于事件的隐式调用体系结构风格

- **应用案例**: Weblogic JMS

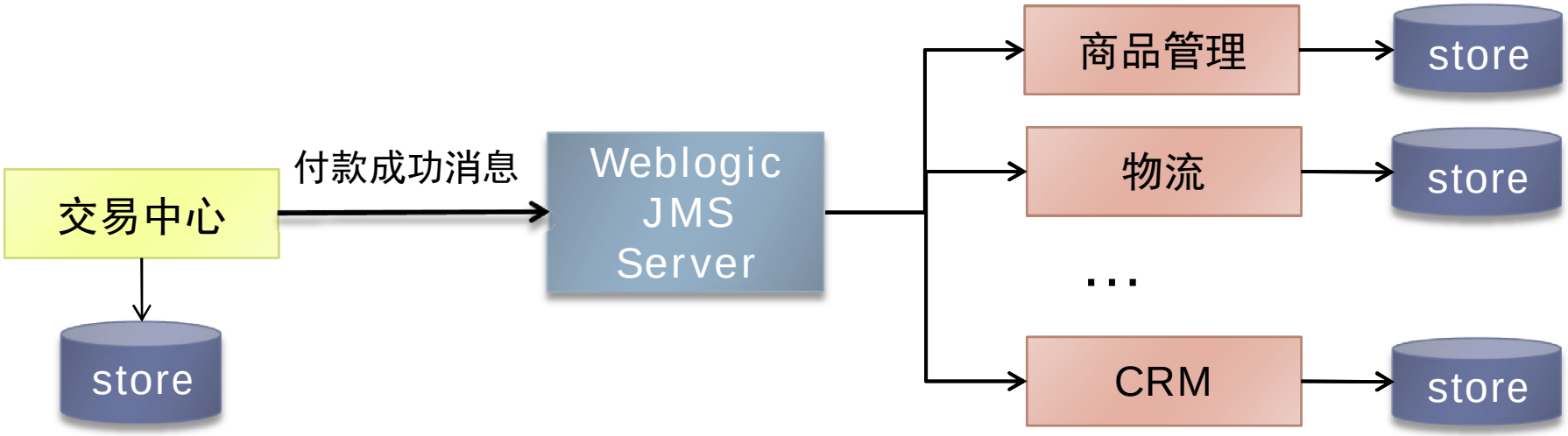
4.3 基于事件的隐式调用体系结构风格

- 应用案例：Weblogic JMS



4.3 基于事件的隐式调用体系结构风格

- 基于Weblogic JMS的多应用的集成
  - 提供分布式事务支持
  - 保证全局数据的一致性



4.3 基于事件的隐式调用体系结构风格

- **优点**
  - **系统具有很好的灵活性，系统易于伸缩扩展**
    - 当用一个构件代替另一个构件时，不会影响到其它构件的接口。
    - 当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中。
- **缺点**
  - **系统控制权的问题**
    - 系统放弃了全局控制：事件源构件触发一个事件时，不能确定其它构件是否会响应它。即使它知道事件注册了哪些事件处理者构件，但它也不能保证事件被处理的顺序和时间。
  - **数据的交换问题**
    - 数据可被一个事件传递，但一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。这也使全局性能和资源管理便成了值得注意的问题。

4.4 分层体系结构风格及应用模式

- Layered System

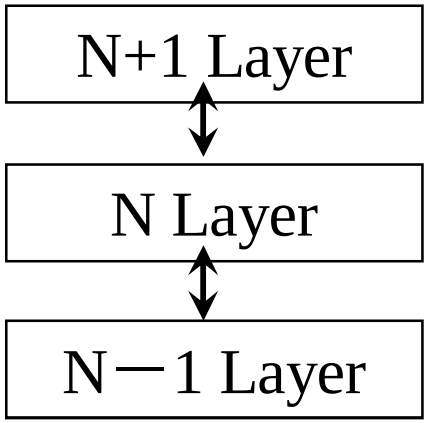
诺贝尔奖获得者赫伯特A. 西蒙曾论述到：“要构造一门关于复杂系统的比较正规的理论，有一条路就是求助于层级理论……我们可以期望，在一个复杂性必然是从简单性进化而来的世界中，复杂系统是层级结构的”。对于软件这样复杂的人造事务，发现层级和运用层级，是分析和构建的基本原则。

4.4 分层体系结构风格及应用模式

- Context（适用场景）
  - 一个需要分解的大系统
    - 如果系统的显著特征是混合了低层与层次问题。
    - 系统的需求本身定义了多个层次上的需求。
  - 系统规格说明描述了高层任务，并希望可移植性。
  - 系统的外部边界要求系统附带功能接口。
  - 高层任务到平台的映射不是直接的。
  - 系统需要满足以下非功能（质量）特性：
    - 后期源代码的改动应该不影响到整个系统。
    - 系统的各个部分应该可以替换。构件应该可以有不同的实现而不影响其他的构件。
    - 提高构件的内聚性。
    - 复杂构件的进一步分解。
    - 设计和开发系统时，工作界限必须清楚。

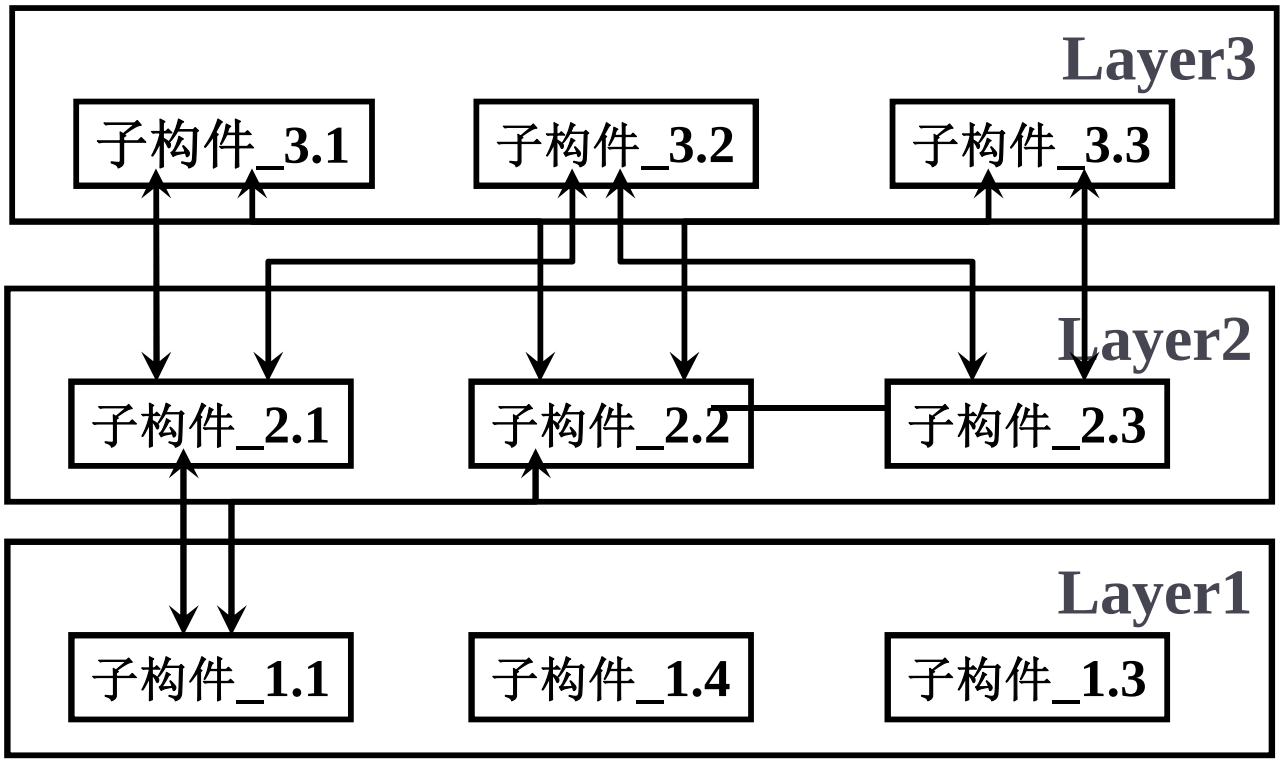
4.4 分层体系结构风格及应用模式

- Model
  - 将系统分成适当层次，按适当次序放置。
  - 从最低抽象层次开始，以梯状把抽象层次n放在n-1层顶部。直到功能顶部。
  - 第n层提供的绝大多数服务由第n-1层提供的服务组成。
  - 每个层次是一个独立的组件。它的责任是：提供了由上层使用的服务，并且委派任务给下一层次。
  - 不允许较高层次直接越级访问较低层次。



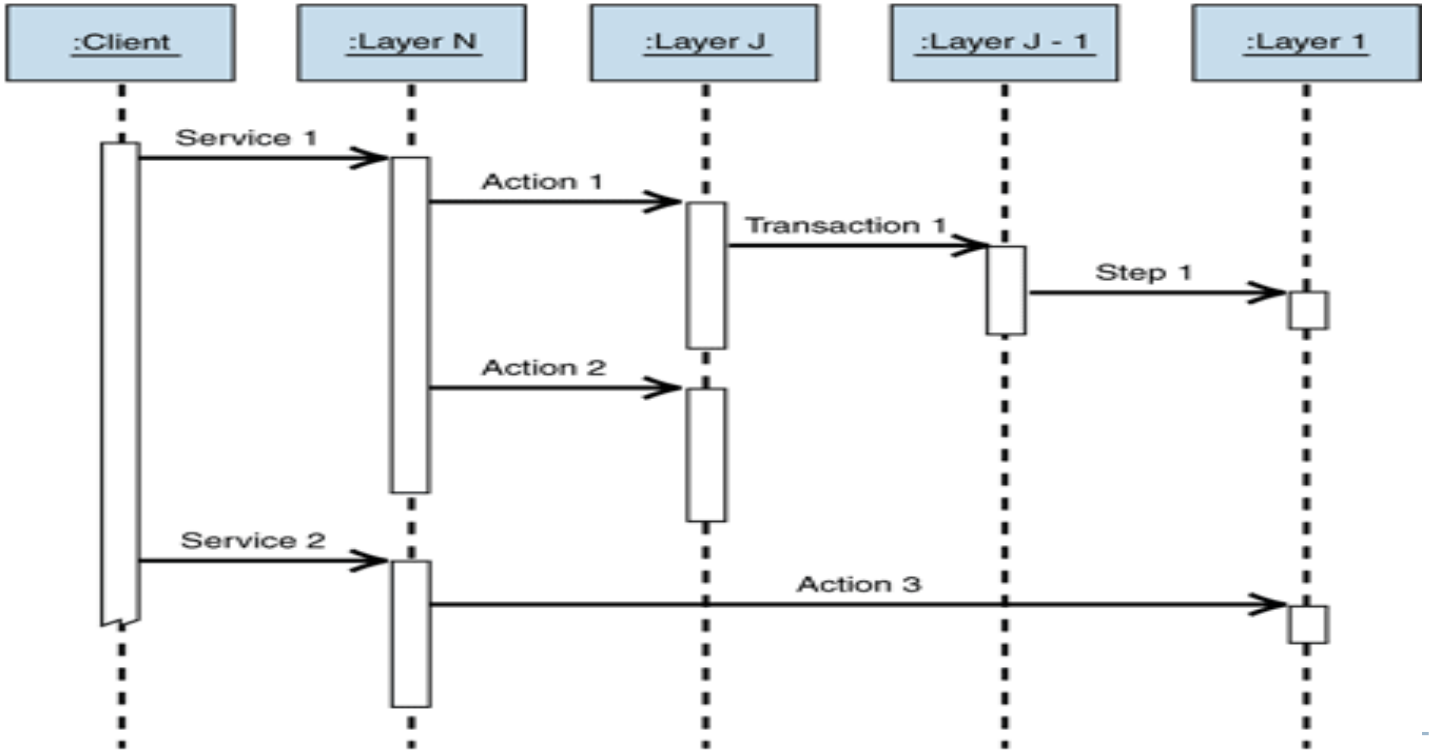
4.4 分层体系结构风格及应用模式

- Model



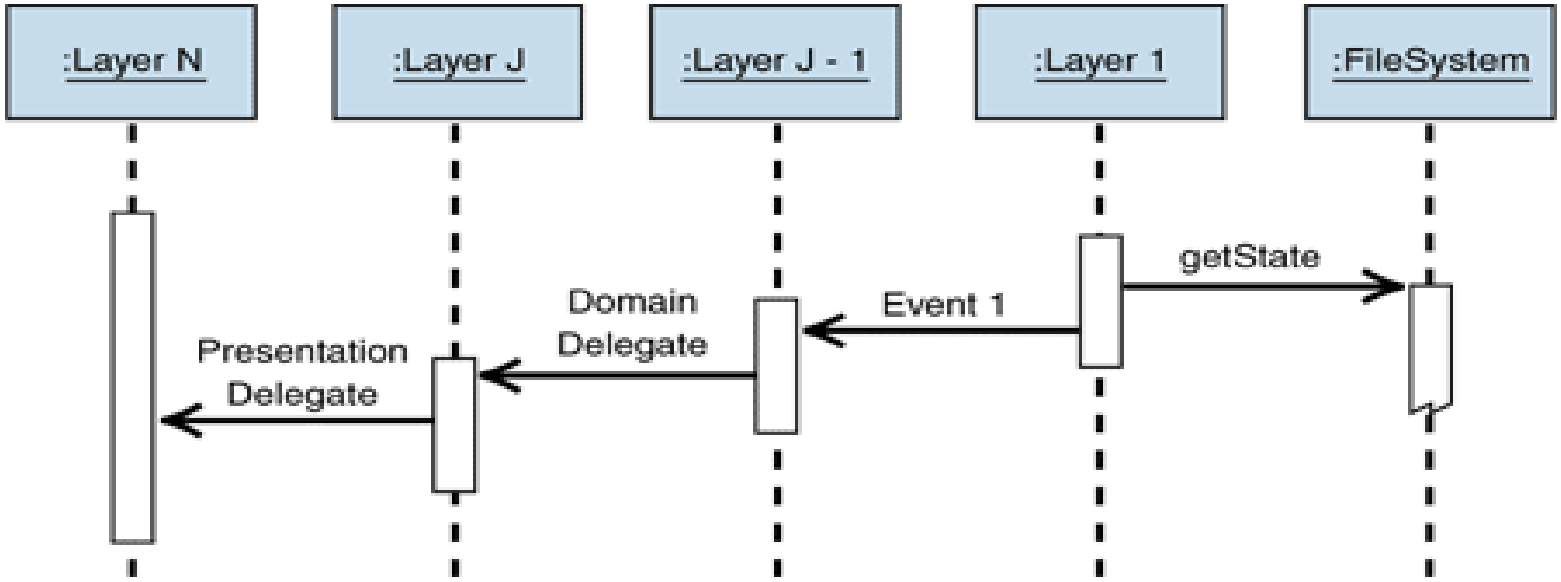
4.4 分层体系结构风格及应用模式

- Scenario View(1)：用户向顶层N发出一个请求。（Top-Down）



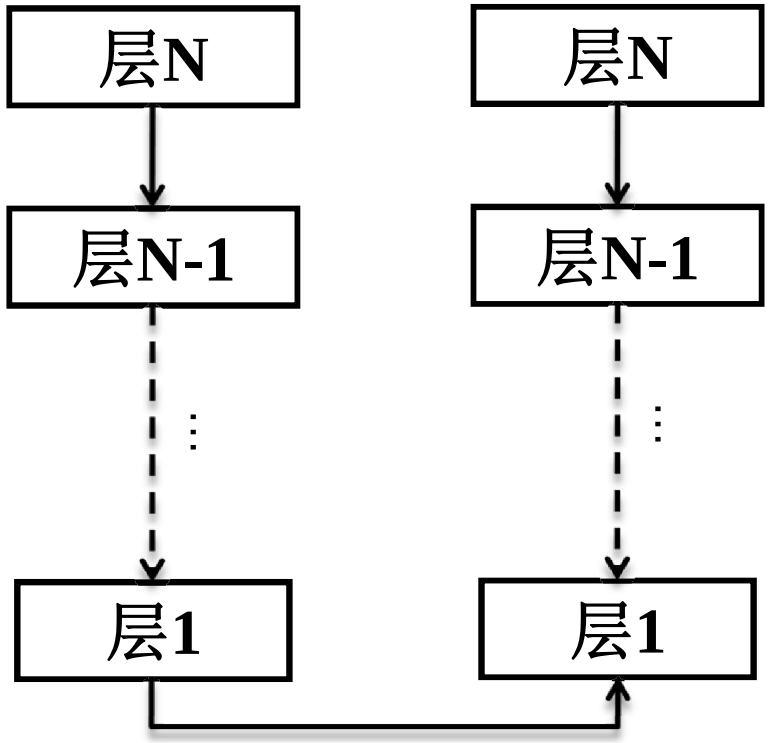
4.4 分层体系结构风格及应用模式

- Scenario View(2)：从底层开始的动作链（Botton-Up）
  - 底层组件探测到某个激发条件，比如某个硬件中断，并且把这个情况组合之后报告给第二层。
  - 层2通知层3，…，层N-1通知层N。



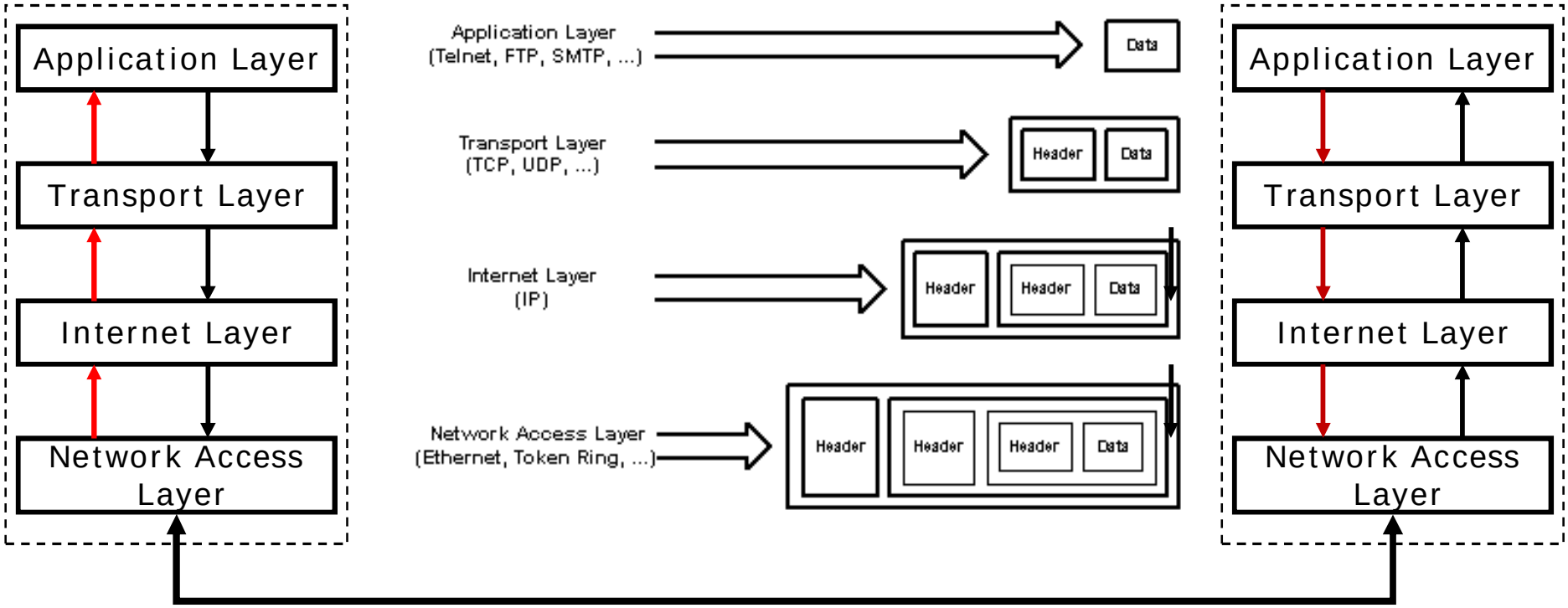
4.4 分层体系结构风格及应用模式

- Scenario View(3)：两个彼此能够通讯的两个层系统
  - 层N接受了一个通信请求后，这个请求通过层向下移动
  - 到达底层之后，数据被传送给另一个堆栈的底层。
  - 另一个堆栈的底层接到数据后，激发
  - 一个自己向上的响应链。
  - 可以理解为第一第二中场景的组合。



4.4 分层体系结构风格及应用模式

- **应用案例：TCP/IP协议系统**

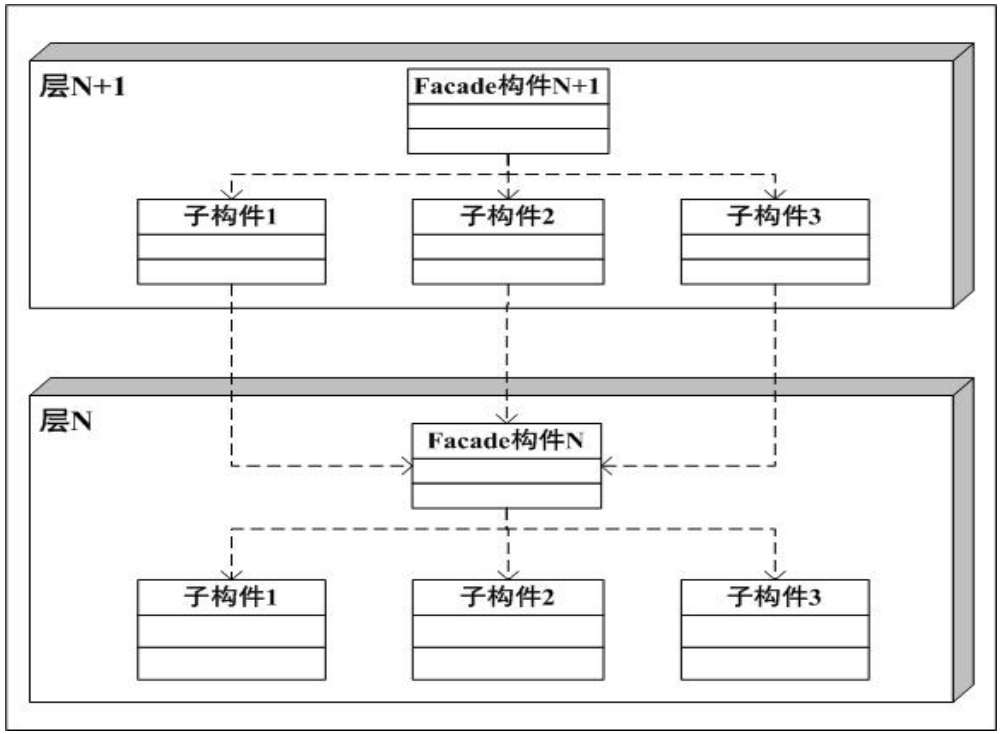


4.4 分层体系结构风格及应用模式

- “层”构件的设计
  - 根据抽象准则定义抽象层次
    - 每个抽象层次对应模式中的一层。
    - 是否把某些功能分在两个层次/或者合并成一个层次时需要权衡利弊
      - 过多的层次增加开销；
      - 层次太少则结构比较差；
  - 给每个层次命名并指定任务
  - 最高层的任务是整个系统的任务。如果较高层完成某个任务的时候，需要使用到下一个抽象层次的细节，那么给下一层次增加相应的任务。
  - 分层的时候还需要考虑到将来的重用。设计底层时需要考虑可能的标准以增加重用的特性。

4.4 分层体系结构风格及应用模式

- 层构件内部子构件端口保护：为每个层构件指定一个单一的端口（Port）
  - 为保证层构件具有良好的封装性，第N层应该对于第N+1层是黑盒。
  - 可以考虑设计一个flat接口，并且把这个接口封装在一个facade（外观）对象中。
  - 灰盒：上层能够知道下层由哪些组件组合而成，但是不知道每个组件的内部结构。
  - 应该尽量使用黑盒的方式，除非有一些特殊的原因：比如效率问题。



4.4 分层体系结构风格及应用模式

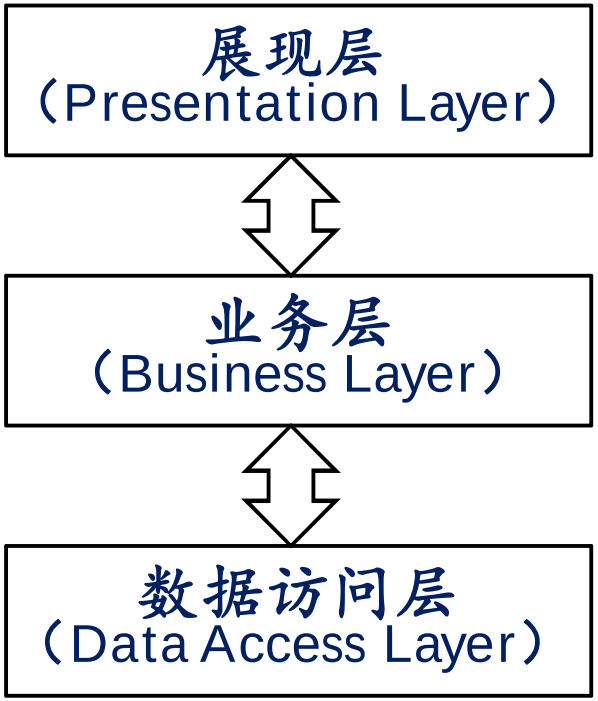
- **指定相邻层之间的交互模式**
  - Push model
  - Pull model
- **指定服务**
  - 在层次之间分配他们需要对上层提供的具体服务。
  - 一般来说：把比较多的服务放在较高层次，而在底层提供的服务应该少，灵活，全面。
- **设计一种错误处理策略**
  - 对于层式体系结构，错误处理在处理时间和编程工作方面的代价比较大。
  - 当错误发生的时候，程序既可以直接处理，也可以把他们发送到更高层次处理。发送到更高层次时，需要使用高层能够理解的错误描述。
  - 应当尽可能在低层处理错误。
  - 至少应该设法将错误归类，避免高层要直接面对众多的难以理解的错误代码。

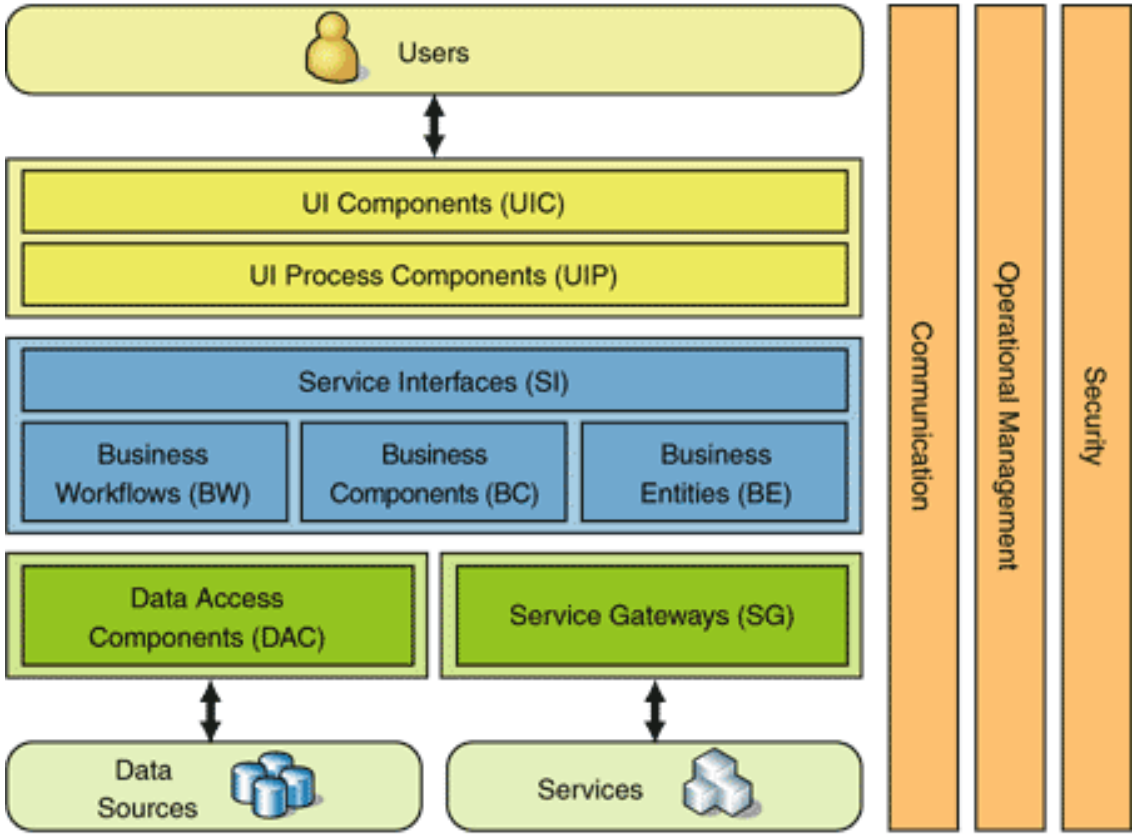
4.4 分层体系结构风格及应用模式

- **特点**
  - **优点：**
    - 层构件的封装性、可重用性、可替换性
    - 系统的局部依赖特性
  - **缺点：**
    - 层构件间的依赖性，特别是当低层构件的修改影响高层构件的时候，可能引起底层之上的多个层构件的修改。
    - 效率问题：顶层构件到底层构件之间需要进行层层的参数传递/转换等。

4.4 分层体系结构风格及应用模式

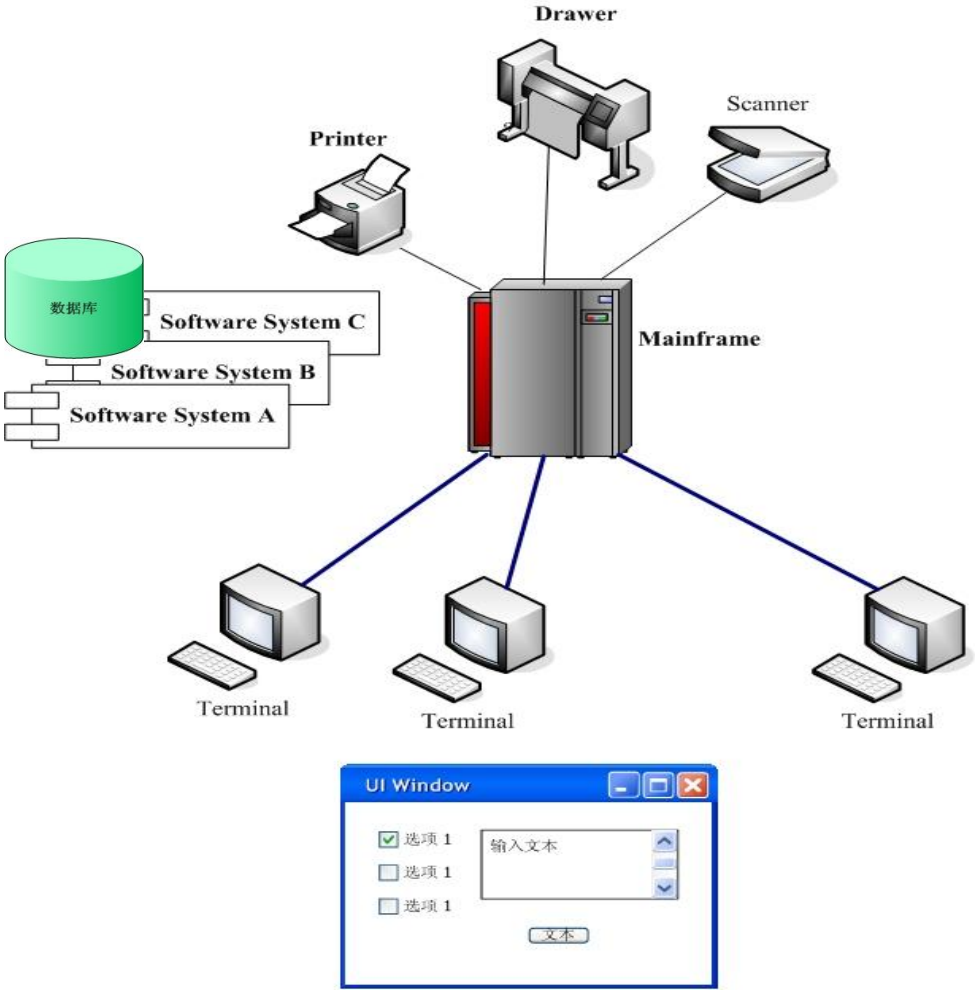
- 应用案例：Web应用系统的分层体系结构





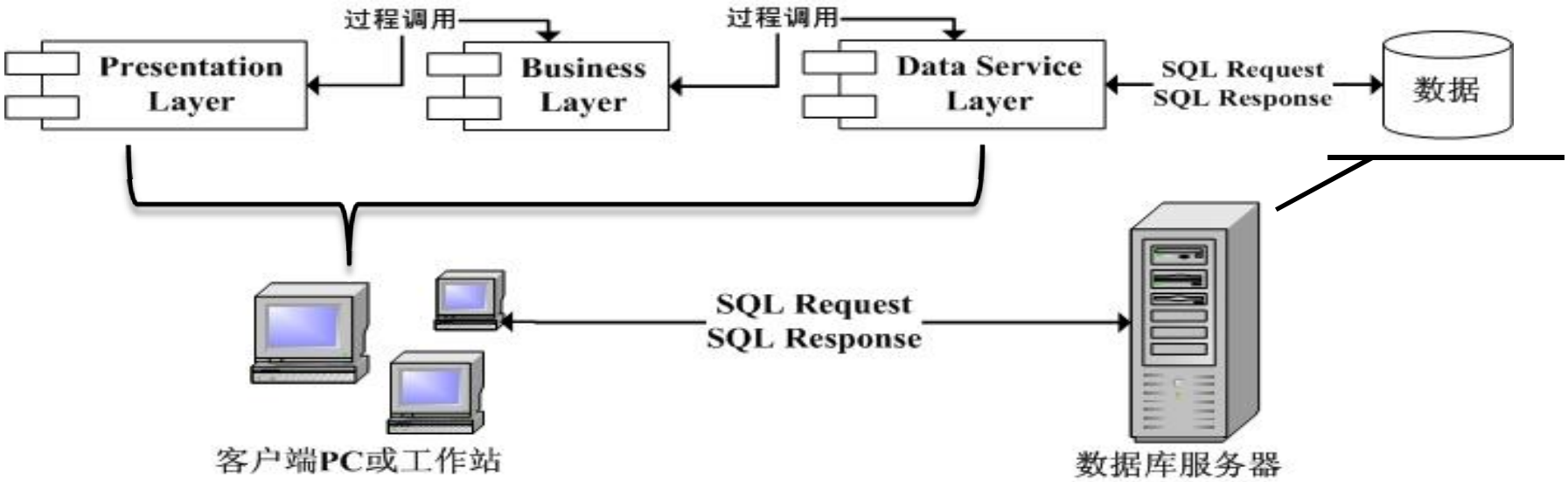
4.4 分层体系结构风格及应用模式

- **企业应用系统的分层体系结构的发展**
  - **集中式物理部署架构（Centralized System Architecture）— 主机时代的企业应用系统体系结构：**
  - Data Layer，Business Layer，Presentation Layer all in one.
  - Software runs on mainframe
  - Users interact at the side of terminals



4.4 分层体系结构风格及应用模式

- 企业应用系统的分层体系结构的发展
  - Two-Tier C/S Pattern



4.4 分层体系结构风格及应用模式

- **企业应用系统的分层体系结构的发展**
  - Two-Tier C/S Pattern

优点:

– DB product independence (compared to single-tier model)

缺点:

– Presentation, Business Logic and Data Service Layers are intertwined (at client side), difficult for updates and maintenance

– Data Acess is “tightly coupled” to every client: If DB Schema changes, all clients break

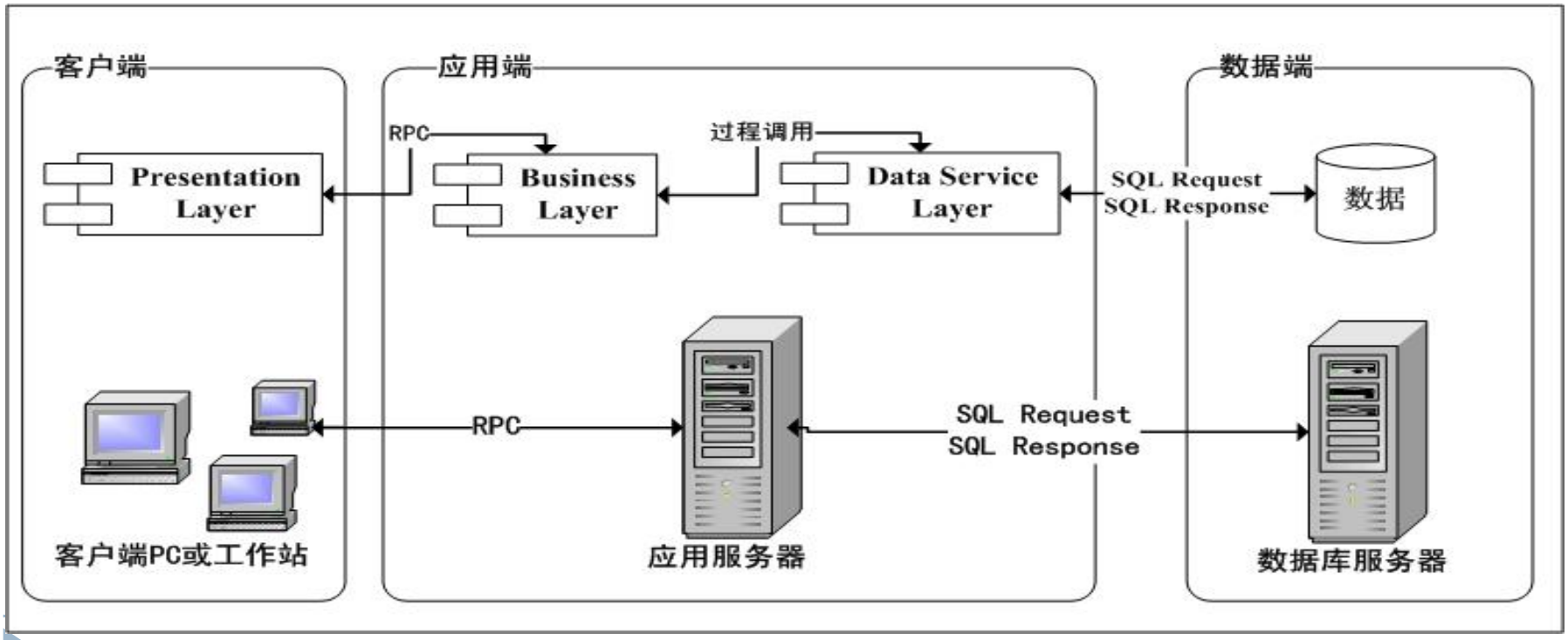
– Updates have to be deployed to all clients making System maintenance nightmare

– DB connection for every client, thus difficult to scale

– Raw data transferred to client for processing causes high network traffic

4.4 分层体系结构风格及应用模式

- 企业应用系统的分层体系结构的发展
  - Three-Tier C/S Pattern



4.4 分层体系结构风格及应用模式

- **企业应用系统的分层体系结构的发展**
  - Three-Tier C/S Pattern

特点:

– Business logic can change more flexibly than 2-tier model

– Most business logic reside in the middle-tier server

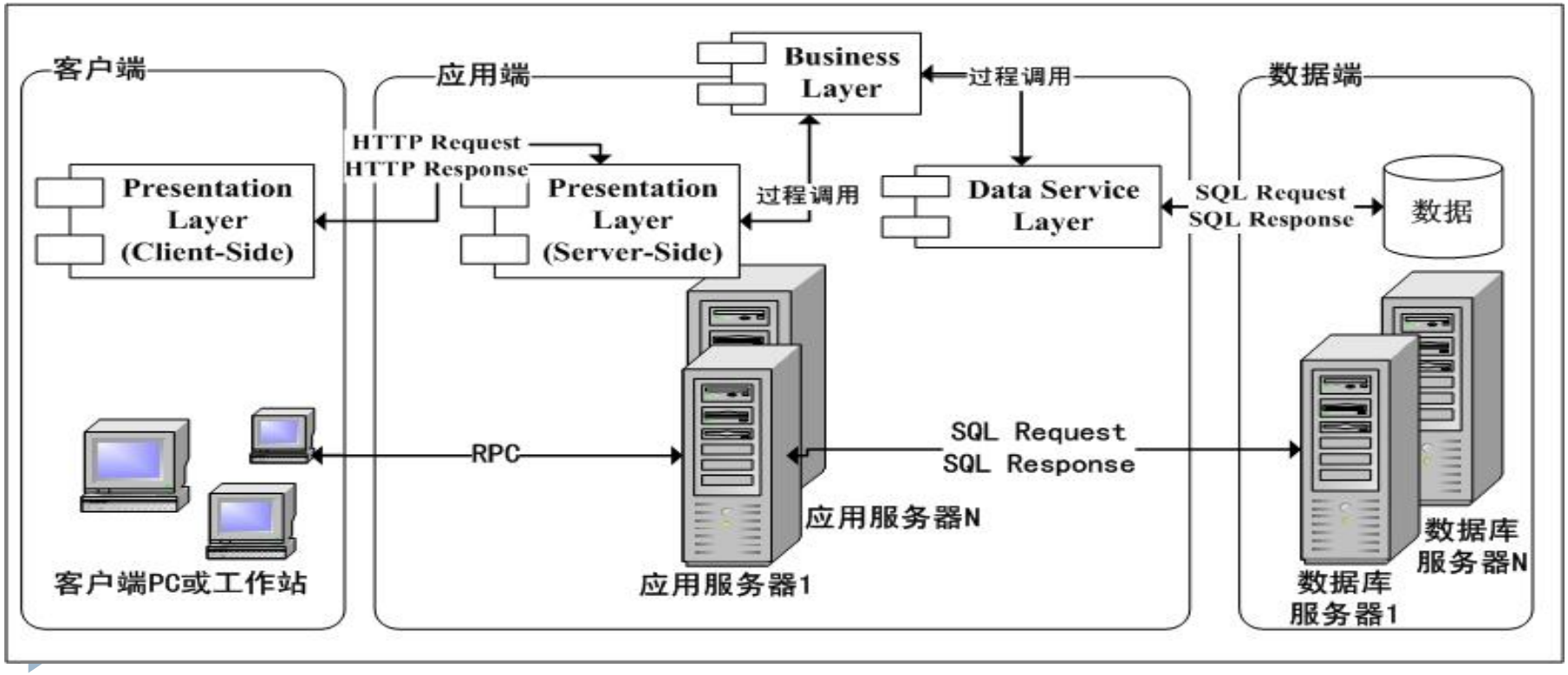
缺点:

– Complexity is introduced in the middle-tier server

– Client and middle-tier server is more tightly coupled (than the three-tier object based model)

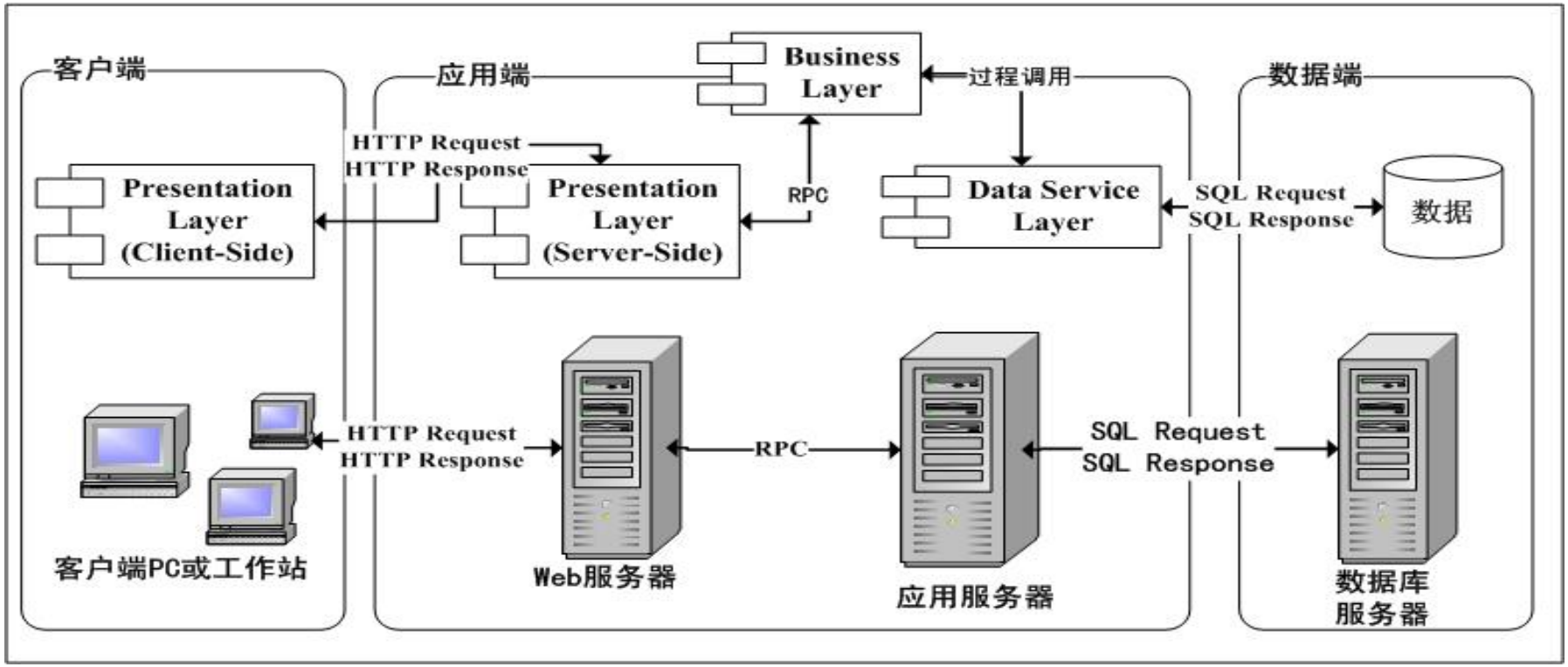
4.4 分层体系结构风格及应用模式

- 企业应用系统的分层体系结构的发展
  - Three-Tier B/S(Browser/Server) Pattern（1）



4.4 分层体系结构风格及应用模式

- 企业应用系统的分层体系结构的发展
  - Three-Tier B/S(Browser/Server) Pattern（2）



4.4 分层体系结构风格及应用模式

- **企业应用系统的分层体系结构的发展**
  - Three-Tier B/S(Browser/Server) Pattern

特点:

– Ubiquitous client types

– Zero client management

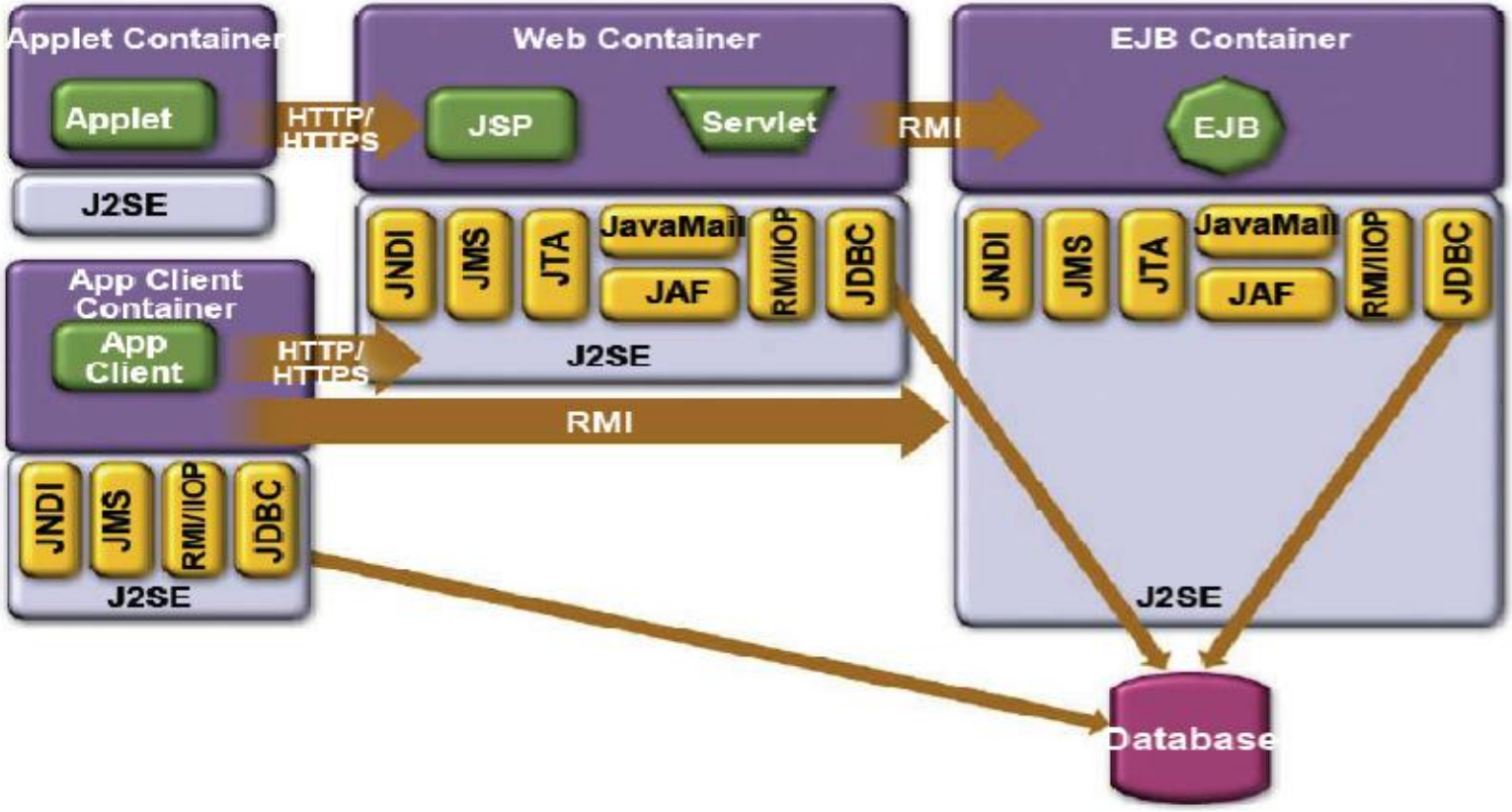
– Support various client devices

缺点:

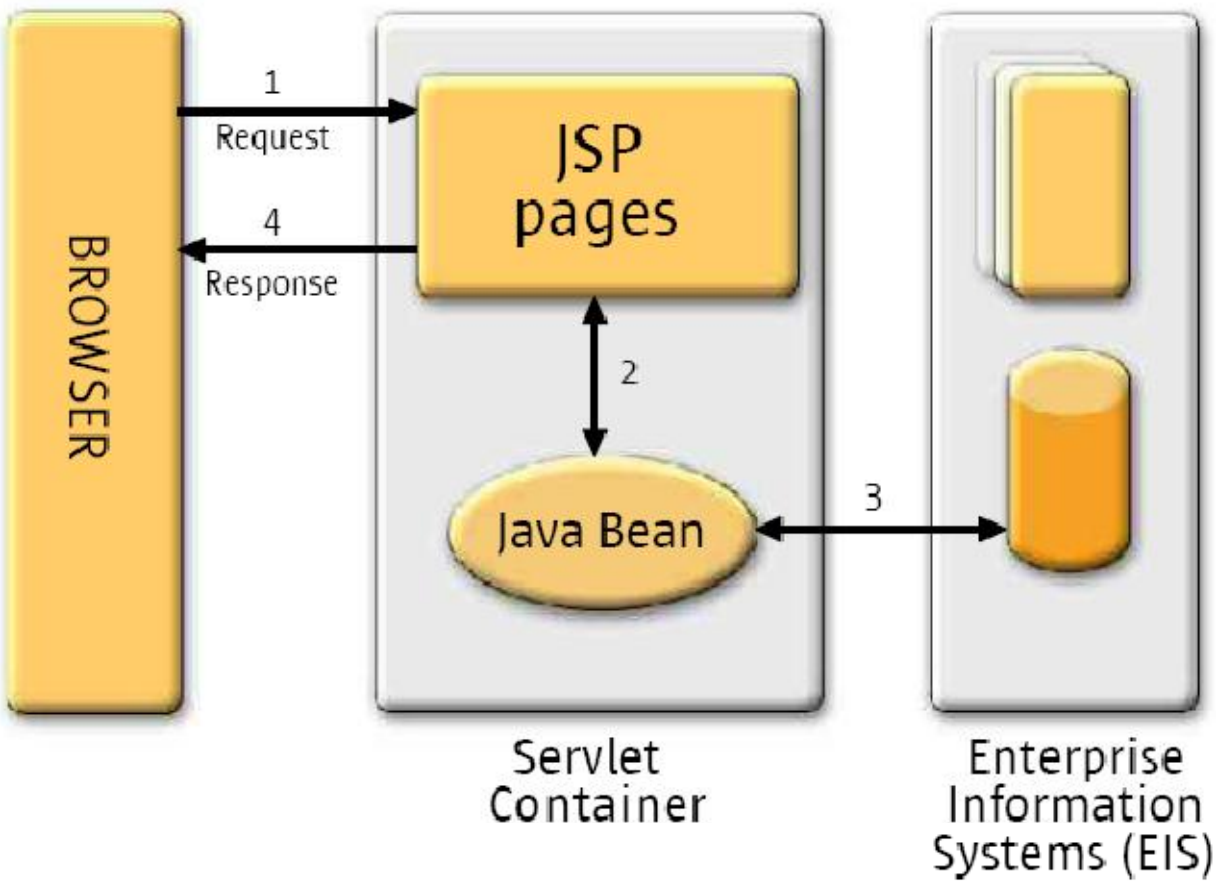
– Complexity is introduced in the middle-tier server

4.4 分层体系结构风格及应用模式

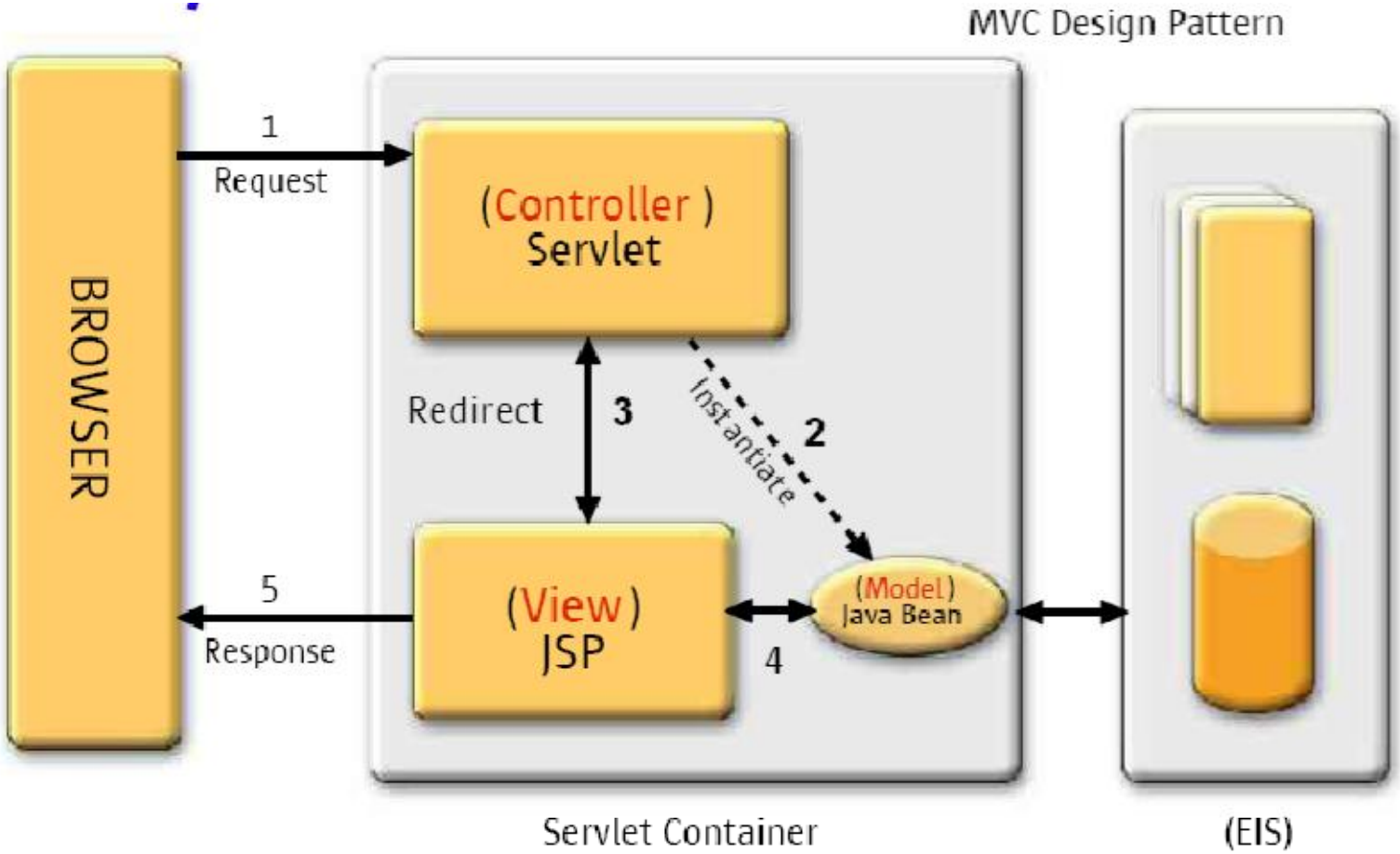
- 基于J2EE的分层企业应用系统架构



4.4 分层体系结构风格及应用模式

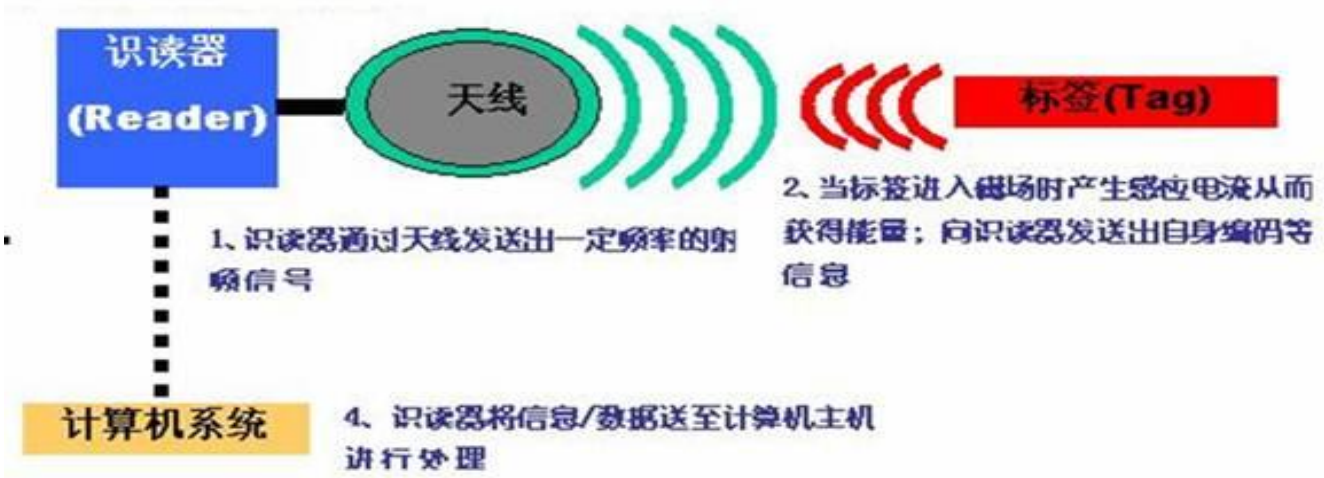


4.4 分层体系结构风格及应用模式



案例分析：一个RFID应用中间件

- RFID (Radio Frequency Identification,**射频识别**) **技术**



读卡器（Reader）：读取（或写入）标签信息的设备，可设计为手持式或固定式。
读卡器对标签的操作有三类：
- 识别(Identify):读取UID;
- 读取(Read):读取用户数据;
- 写入(Write):写入用户数据

标签(Tag)：标签由天线和芯片组成，天线在标签和读卡器间传递射频信号，芯片里面保存每个标签具有的唯一电子编码和用户数据。 每个标签都有一个全球唯一的ID号码—UID，UID是在制作芯片时放在ROM中的，无法修改；用户数据区是供用户存放数据的，可以进行读写、覆盖、增加的操作。

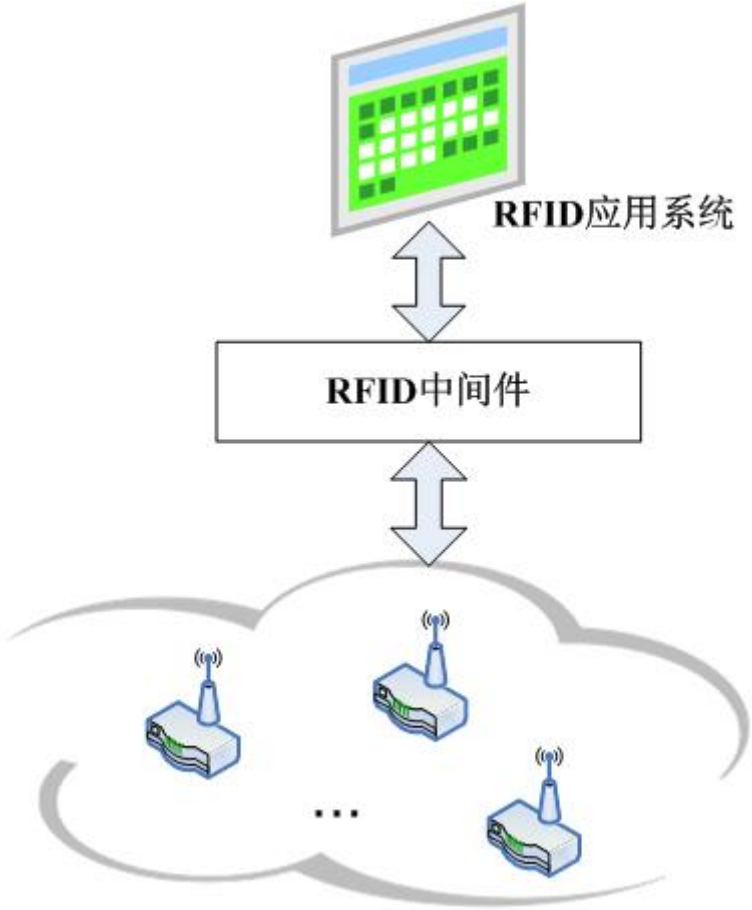
案例分析：一个RFID应用中间件

- **RFID主要应用领域：**
  - 铁路车号自动识别管理。如：北美铁路、中国铁路、瑞士铁路等。
  - 车辆道路交通自动收费管理。如：北美部分收费高速公路的自动收费中国部分高速公路自动收费管理、东南亚国家部分收费公路的自动收费管理。
  - 旅客航空行包的自动识别、分拣、转运管理。如：机场行李分拣。
  - 车辆出入控制。如：停车场、垃圾场、水泥场车辆出入、称重管理等。
  - 校园卡、饭卡、乘车卡、会员卡、驾照卡、健康卡(医疗卡)等
  - 生产线产品加工过程自动控制：主要应用在大型工厂的自动化流水作业线上。
  - 动物识别(养牛、养羊、赛鸽等)：大型养殖厂、家庭牧场、赛鸽比赛。
  - 物流、仓储自动管理。大型物流、仓储企业。
  - 贮气容器的自动识别管理。
  - 汽车遥控门锁、电子门锁等。

案例分析：一个RFID应用中间件

- **RFID中间件系统需求描述**

  作为一个RFID Reader设备网络与应用信息系统间的一个中间系统，其面向下层的RFID Reader设备需提供一个物理设备透明和动态配置的设备读写与管理的适配层，面向上层的各种RFID业务应用系统应提供灵活的RFID数据访问服务与业务场景的适应机制



案例分析：一个RFID应用中间件

- **RFID中间件主要设计目标**
  - 支持标准的RFID物理设备的读写驱动与配置管理，可动态配置新的RFID设备驱动及物理参数；
  - 支持ISO和EPCglobal标签（Tag）标准，支持各RFID标签提供商的Tag扩展的读写；
  - 遵循EPCglobal的Application Level Events (ALE)标准；
  - 提供可灵活配置的面向业务场景需要的ALE事件处理逻辑；
  - 提供面向企业应用信息系统访问与集成的各种API访问接口与集成接口；
  - 提供对RFID设备的远程实时管理与监控；
  - 支持多RFID设备的并发实时读写与ALE事件处理；

案例分析：一个RFID应用中间件

- **系统高层架构视图：分层风格的系统体系结构**
  - **数据集成服务层**

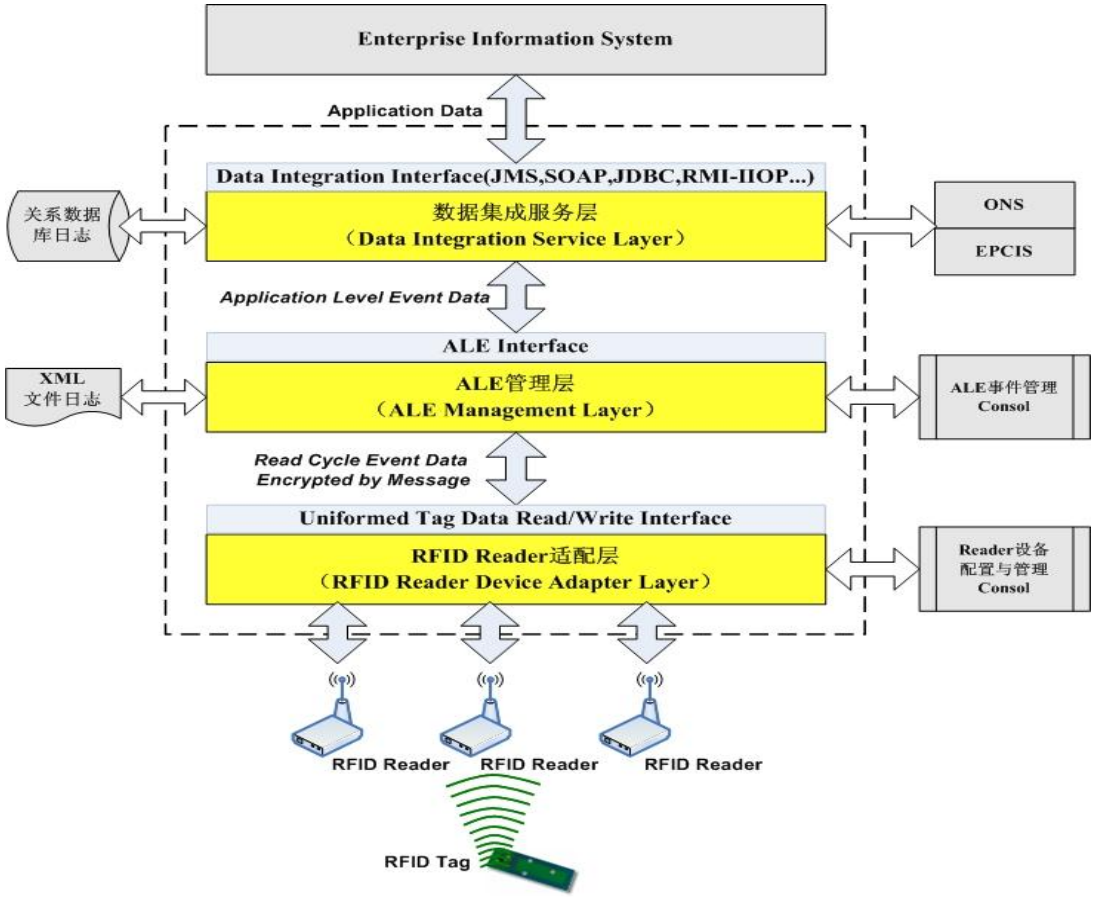
实现与各种企业应用信息系统的数据交换与集成，包括同步与异步的数据访问、基于消息的数据集成、基于Web Service的服务访问与集成以及支持其它各种访问与集成机制的Connector扩展。

  - **ALE管理层**

RFID读写事件的定义、事件过滤与处理及事件持久化的日志记录。

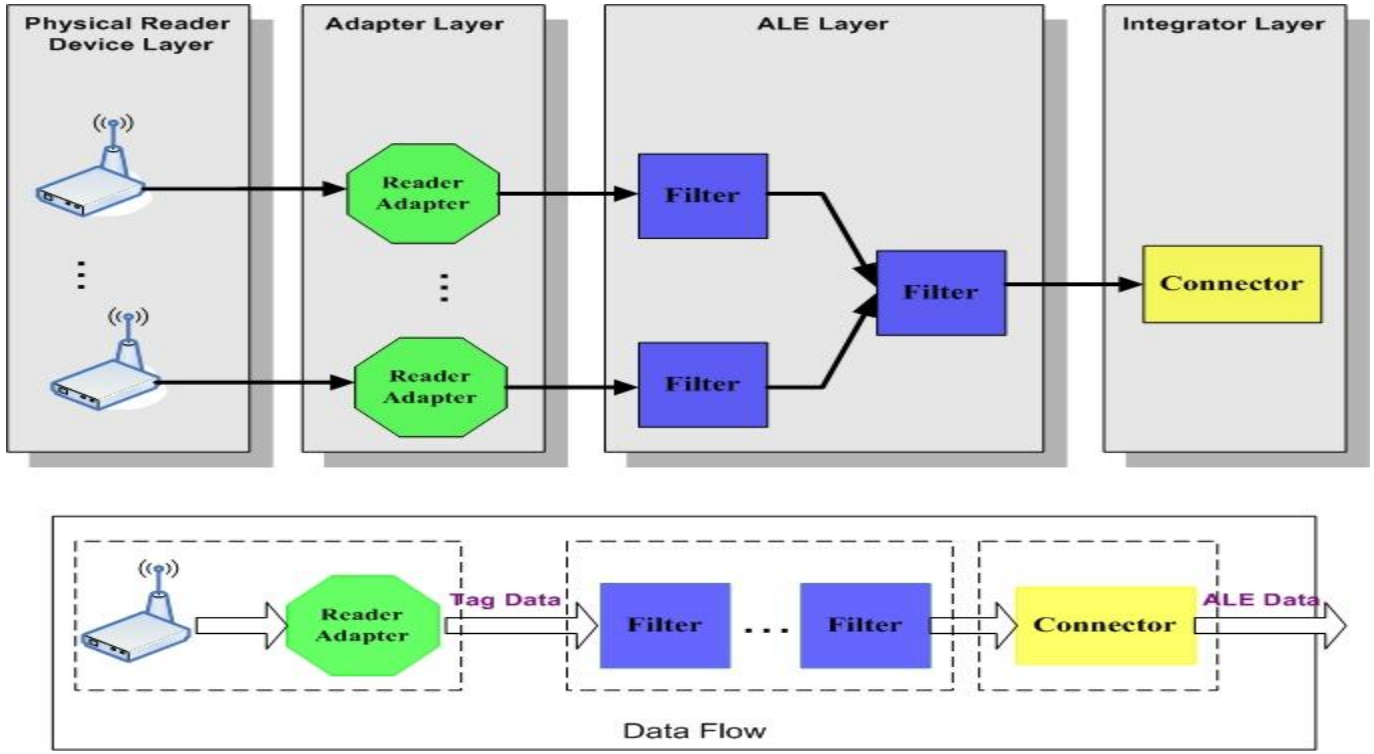
  - **RFID数据适配层**

对接入的RFID Reader物理设备的读写驱动及物理参数的配置与管理。



案例分析：一个RFID应用中间件

- 进一步分解后的架构视图—基于管道-过滤器体系结构风格的数据流处理系统



案例分析：一个RFID应用中间件

- 支持的Filter与Connector类型

表 2-5　支持的过滤器

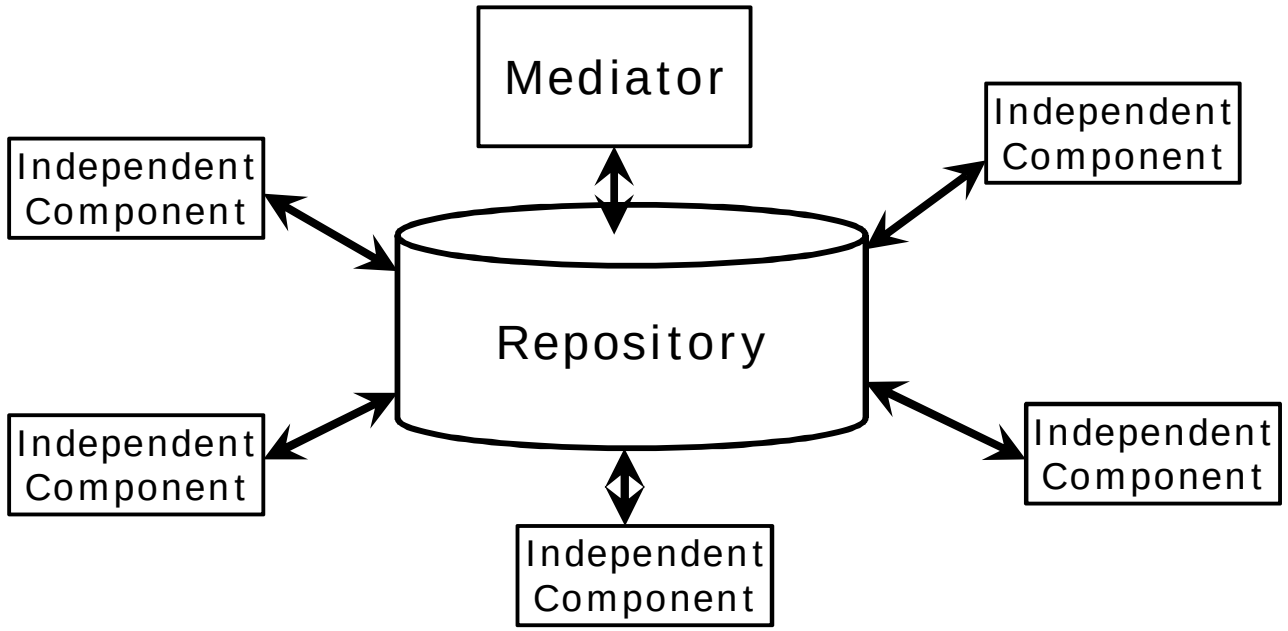
名称	说明	属性
BandPass	对阅读器 EPC 进行带通过滤。与 EPC 掩码匹配的阅读器事件被传递到监听器，不匹配的则不传递。	请参阅表 B-4
EPC	对标签 EPC 进行带通过滤。与 EPC 掩码匹配的 EPC 被传递到监听器，不匹配的则不传递。	请参阅表 B-3
Smoothing	创建在指定 *n* 次循环后发现的 EPC 集合。将报告在 < *n* 次循环后发现的 EPC，如果在最后 *n* 次循环后仍未发现，则不报告该 EPC。由于 RFID 阅读器不能以 100% 的准确度报告标签，此功能很有必要。	请参阅表 B-1

表 2-6　支持的连接器

名称	说明	属性
File Connector	向文件中写入 PML Core 的连接器	请参阅表 B-5
HttpPml Connector	向 HTTP 连接写入 PML Core 的连接器	请参阅表 B-6
JMS Connector	向 JMS 队列或主题写入 PML Core 的连接器	请参阅表 B-7
Socket Connector	创建套接字连接，并开始向连接写入 PML Core。	请参阅表 B-8
ServerSocket Connector	创建服务器套接字并监听连接。建立连接后即开始写入 PML Core。	请参阅表 B-9

4.5 Repository风格及应用模式

- **以数据为中心（Data-Center）的体系结构风格**



  - Component类型: Independent Component（独立构件）、Mediator（仲裁者）
  - Connector类型: Repository

4.5 Repository风格及应用模式

- Context（适用场景）
  - 以数据为中心的分布式系统

Repositories Types:

a. **Passive Repository: Database系统**

Accessed by a set of components.

b. **Active Repository: Blackboard系统**

Sends notification to components when data of interest changes.

4.5 Repository风格及应用模式

- Active Repository----Blackboard 风格
  - Component类型: Knowledge Source(KS)、Mediator;
  - Connector类型：Blackboard;

➢ Blackboard: problem-solving state data, organized into an application-dependent hierarchy. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.

➢ KS: separate, independent parcels of application dependent knowledge. Interaction among knowledge sources takes place solely through the blackboard.

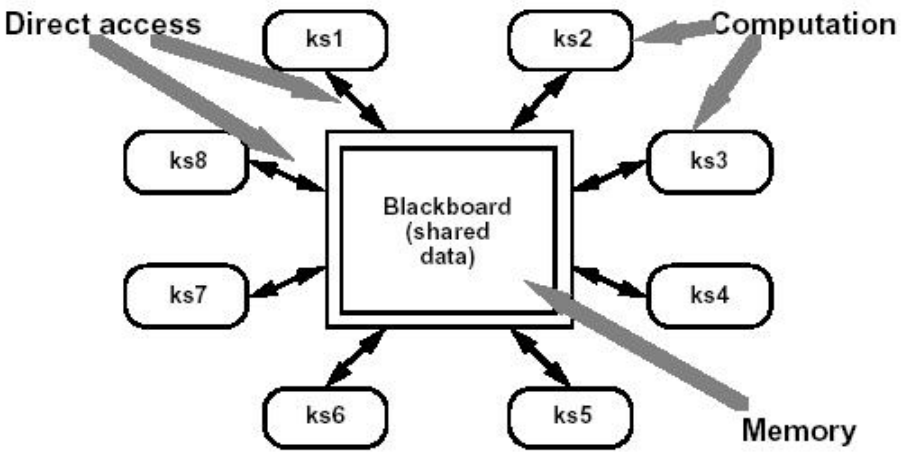
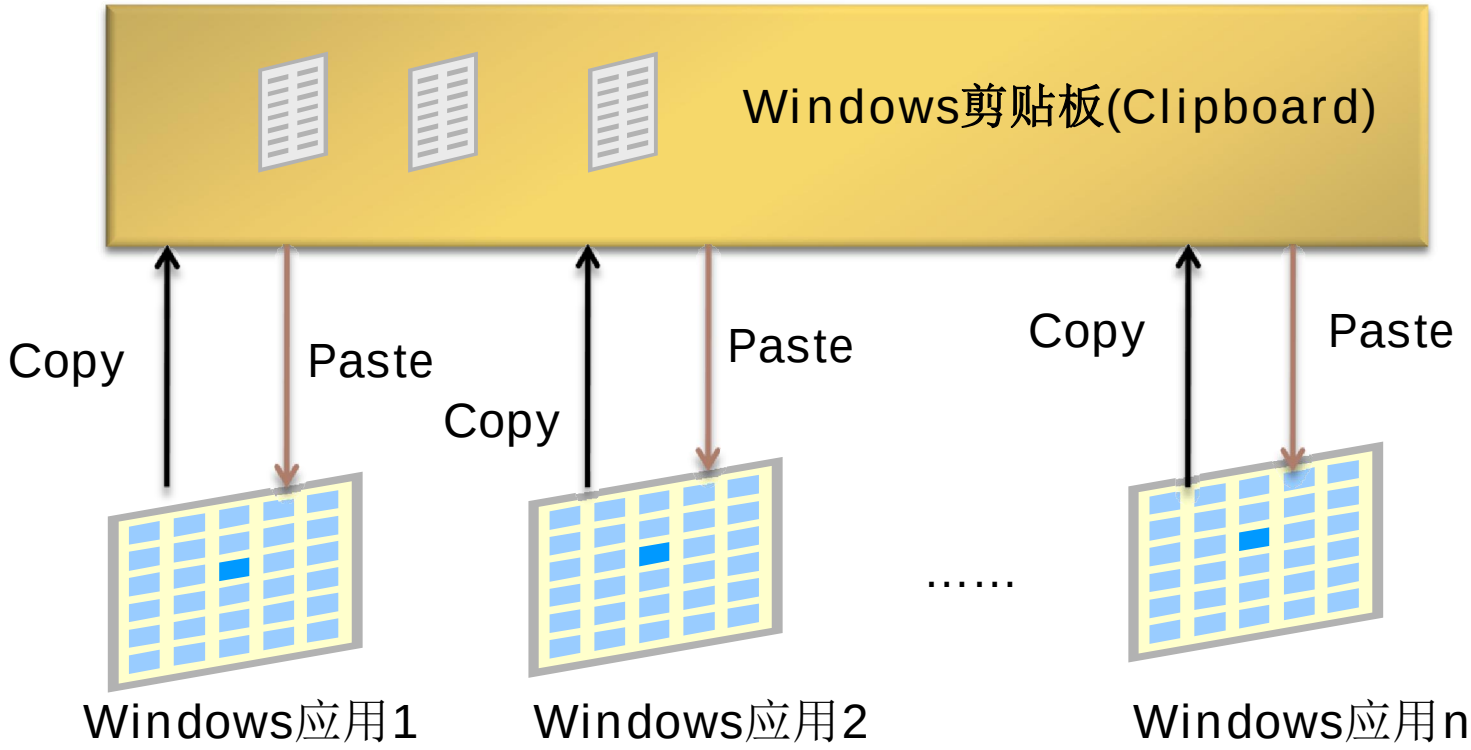


Figure 4: The Blackboard

➢ Mediator: driven entirely by state of blackboard. Knowledge sources respond opportunistically when changes in the blackboard make them applicable.

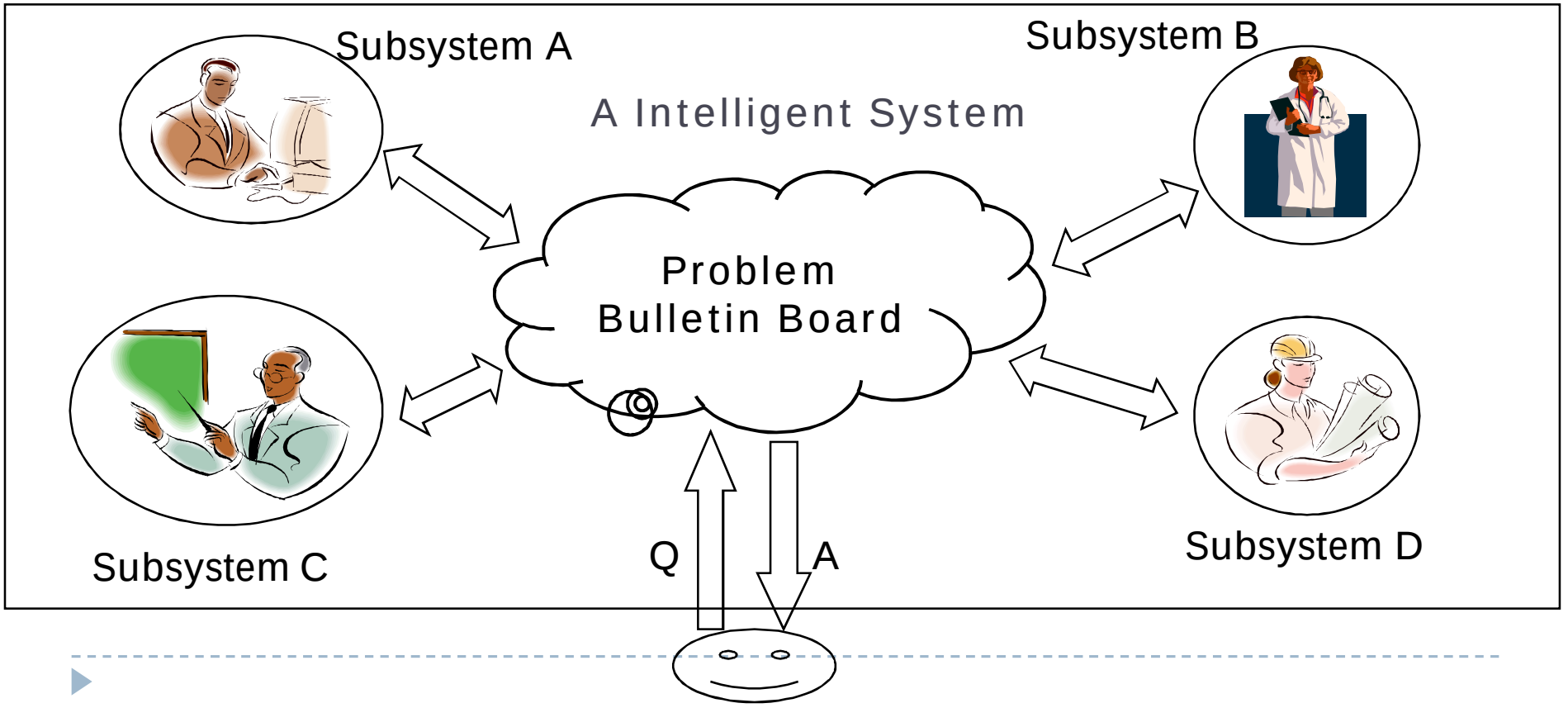
4.5 Repository风格及应用模式

- **典型系统案例：Windows剪贴板（Passive Repository）**



4.5 Repository风格及应用模式

- **典型系统案例: 专家系统（Active Repository）**

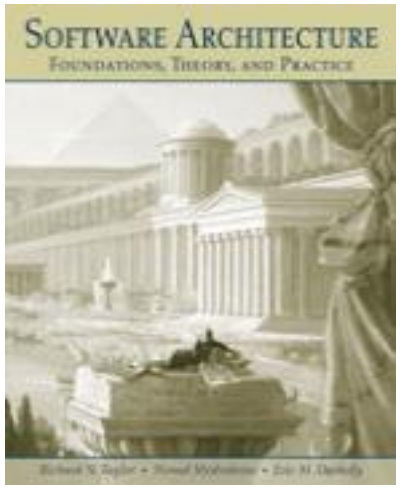


4.5 Repository风格及应用模式

- **特点**
  - 优点
    - 可扩展性(Scalability): components can be easily added.
    - 可维护性(Flexibility): functionality of components can be easily updated.
    - 安全性(Security): all components share the same data, so security measures can be centralized around blackboard.
    - 并行处理性: easily to execute in parallel fashion, but consistency may incur synchronization.
  - 缺点:
    - 单一失败点: Repository is single point of failure.

Continue...

- **课外参考资源**



- The homepage for book *Software Architecture: Foundations, Theory, and Practice*:
http://www.softwarearchitecturebook.com/svn/main/slides/ppt/

- 国外大学软件体系结构课程网站（*加拿大Toronto大学*）：
http://www.cdf.toronto.edu/~csc407h/winter/