University of Science and Technology of China

Software Architecture

SSE USTC Qing Ding
dingqing@ustc.edu.cn

Pattern and Style I

Outline

学习架构就要关注components，connectors和优点，缺点

- Introduction
- Data-flow Architectures
- Data-centered Architectures
- layer architectures
- Notification Architectures

以上4个patterns是比较古老经典的

Introduction

How Do You Design?

Where do architectures come from?

Creativity

1) Fun!
2) Fraught with peril 充满危险
3) May be unnecessary
4) May yield the best

1) Efficient in familiar terrain
2) Not always successful
3) Predictable outcome (+ & -)
4) Quality of methods varies

Method

Identifying a Viable Strategy

●使用基本的设计工具：抽象和模块化。
　◆但如何？
●灵感，灵感是必要的。可预见的技术。
　◆但是哪里需要创造力呢？
●应用自己的经验或别人的经验

- Use fundamental design tools: abstraction and modularity.
  - *But how?*
- Inspiration, where inspiration is needed. Predictable techniques elsewhere.
  - *But where is creativity required?*
- Applying own experience or experience of others.

The Tools of "Software Engineering 101"

- Abstraction
  - Abstraction(1): 查看细节，并将其抽象"上升"到概念 look at details, and abstract "up" to concepts
  - Abstraction(2): 选择概念，然后添加详细的子结构，然后"向下 " 移动 choose concepts, then add detailed substructure, and move "down"
    - Example: design of a stack class

- 关注点分离 Separation of concerns

Learning Objectives

中国科学技术大学
University of Science and Technology of China

描述DSSAs和模式在软件体系结构中的作用，并将公共模式应用于问题

- Delineate the role of DSSAs and Patterns in Software architecture, and apply common patterns to problems

理解架构风格的作用和好处

- Understand the role and benefits of architectural styles

在你的设计中理解和应用常见的风格

- Understand and apply common styles in your designs

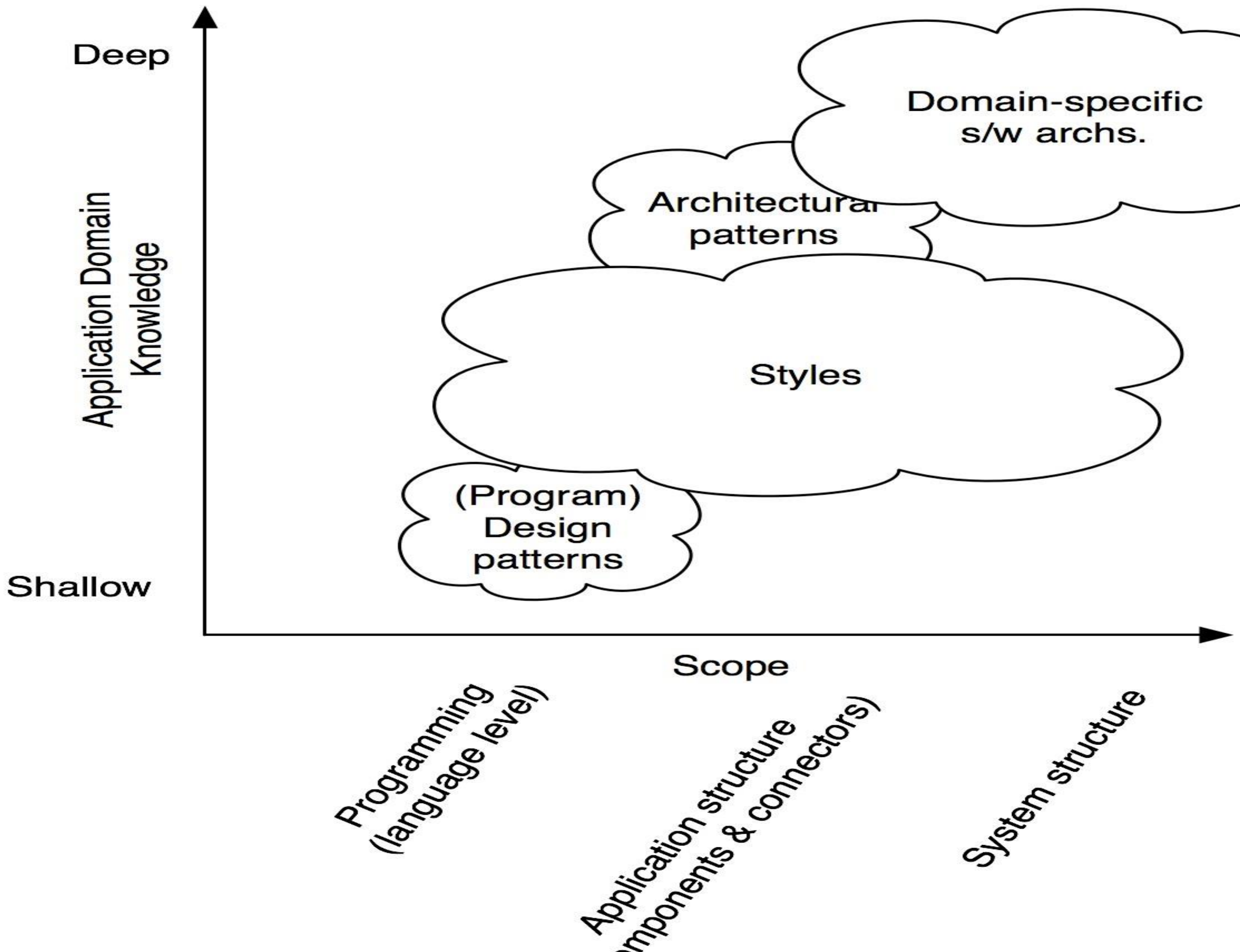
从简单的样式中构造复杂的样式

- Construct complex styles from simpler styles

了解未开发设计的挑战

- Understand the challenges around greenfield design

Patterns, Styles, and DSSAs


Deep
Application Domain Knowledge
Shallow
Domain-specific s/w archs.
Architectural patterns
Styles
(Program) Design patterns
Scope
Programming (language level)
Application structure
mponents & connectors)
System structure

Architecture and Program

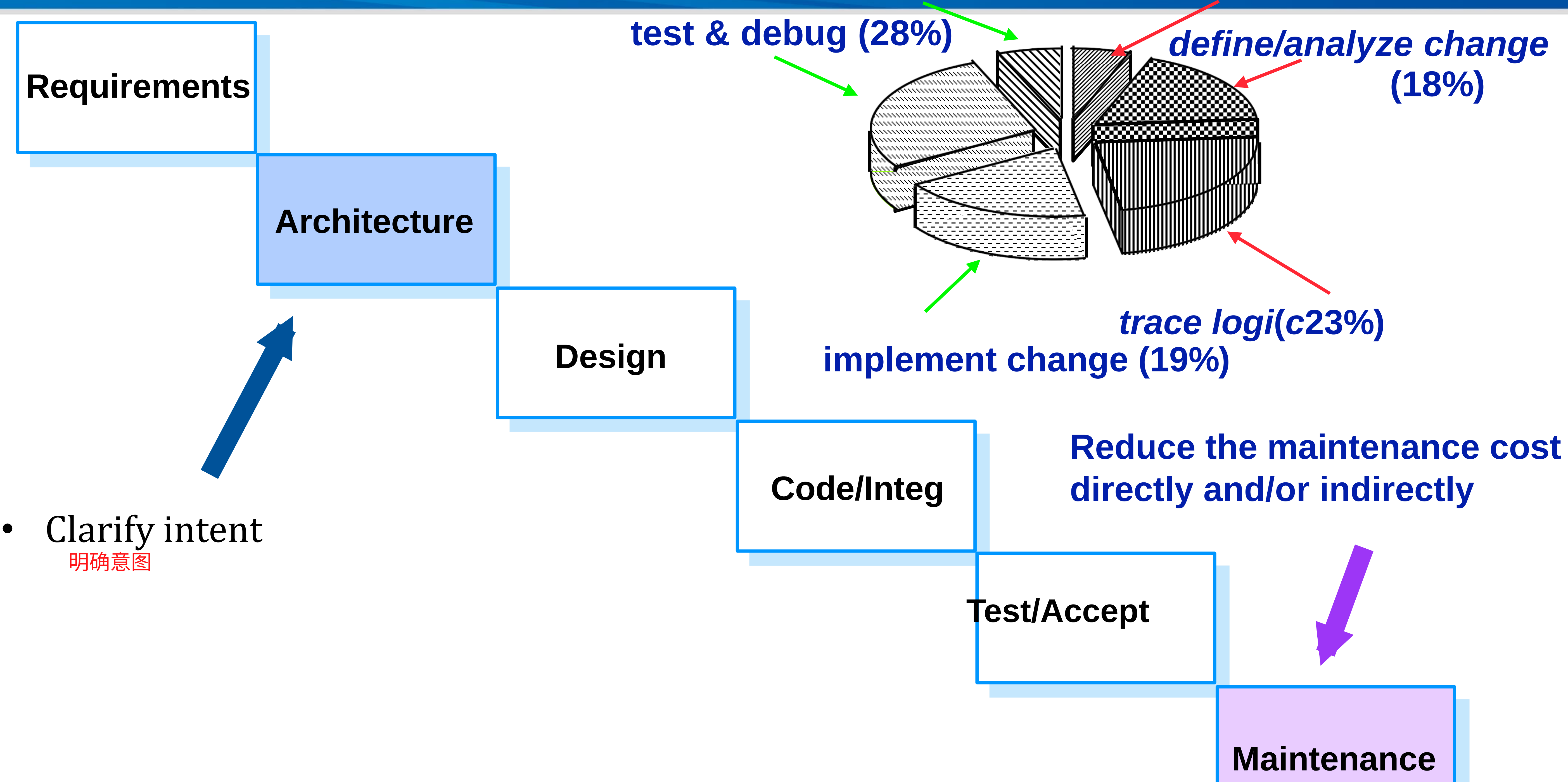
Architecture	Program
interactions among parts	implementations of parts
structural properties	computational properties
declarative	operational
mostly static	mostly dynamic
system–level performance	algorithmic performance
outside module boundary	inside module boundary
composition of subsystems	copy code or call libraries

Architecture Modeling

中国科学技术大学
University of Science and Technology of China

- Clarify Design Intent 明确设计意图
  -设计好的架构经常会丢失。它大多是非正式的，无论如何也很难沟通
  – Intended architecture is often lost. It's mostly informal, it's hard to communicate anyhow.
- Provide Analysis for Design 为设计提供分析
  -工程设计需要性能预测和设计调整。常规的做法
  – Engineering design entails performance prediction and design tuning. Routine practice.
- Improve maintainability 提高可维护性
  -超过一半的维护工作需要弄清楚到底有什么
  – Over half of maintenance effort goes into figuring out just what's there.
- Answer Difficult Questions 回答困难的问题
  -即使没有正式的方法，明确的架构建模也可以揭示模糊的需求、思维和设计方法
  – Even without formal methods, explicit architectural modeling can uncover fuzzy requirements, thinking, and design approaches

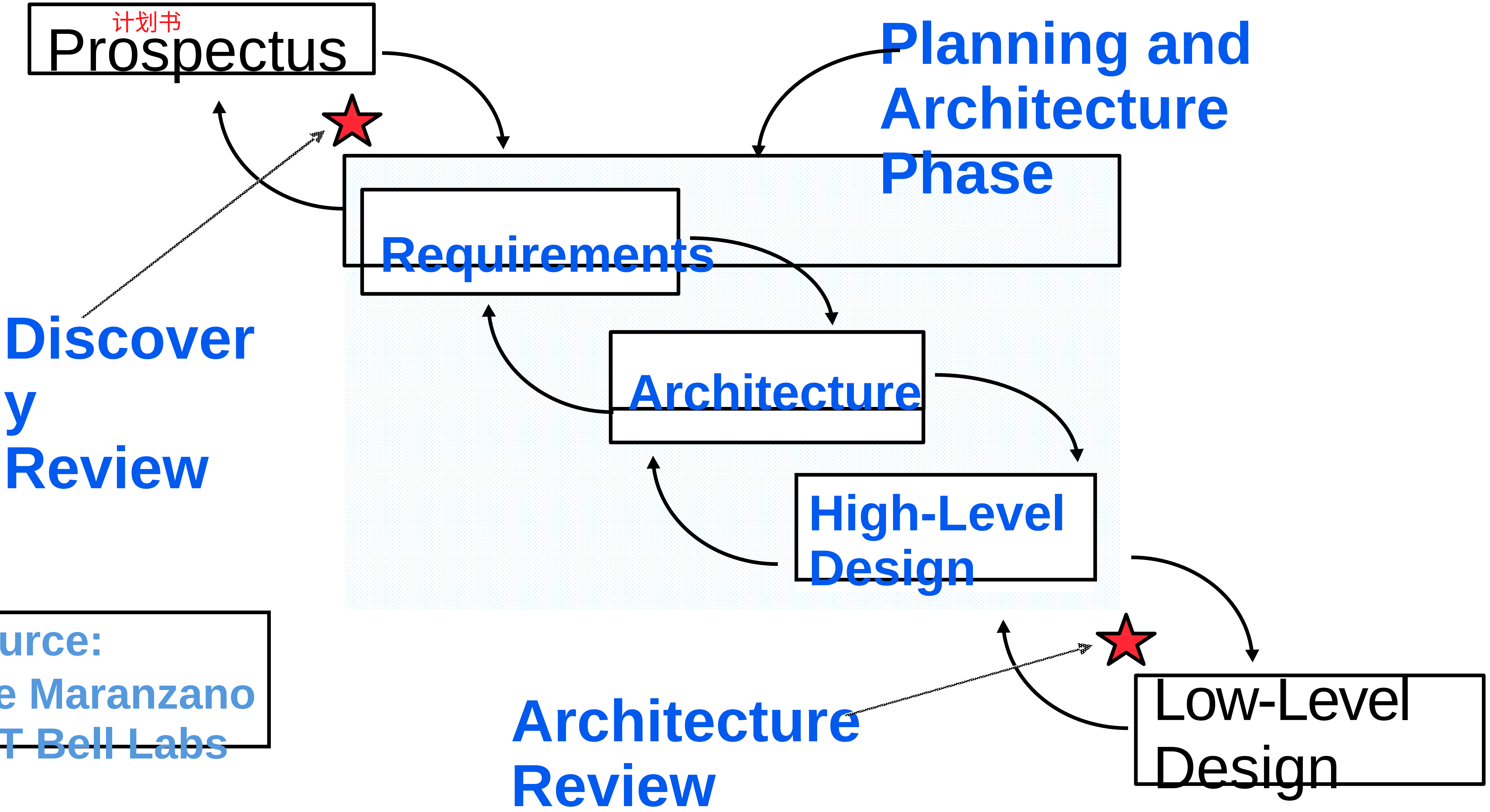
Architecture Modeling



Reduce the maintenance cost directly and/or indirectly

- Clarify intent
  明确意图

Check Architecture



Source:
Joe Maranzano
ATT Bell Labs

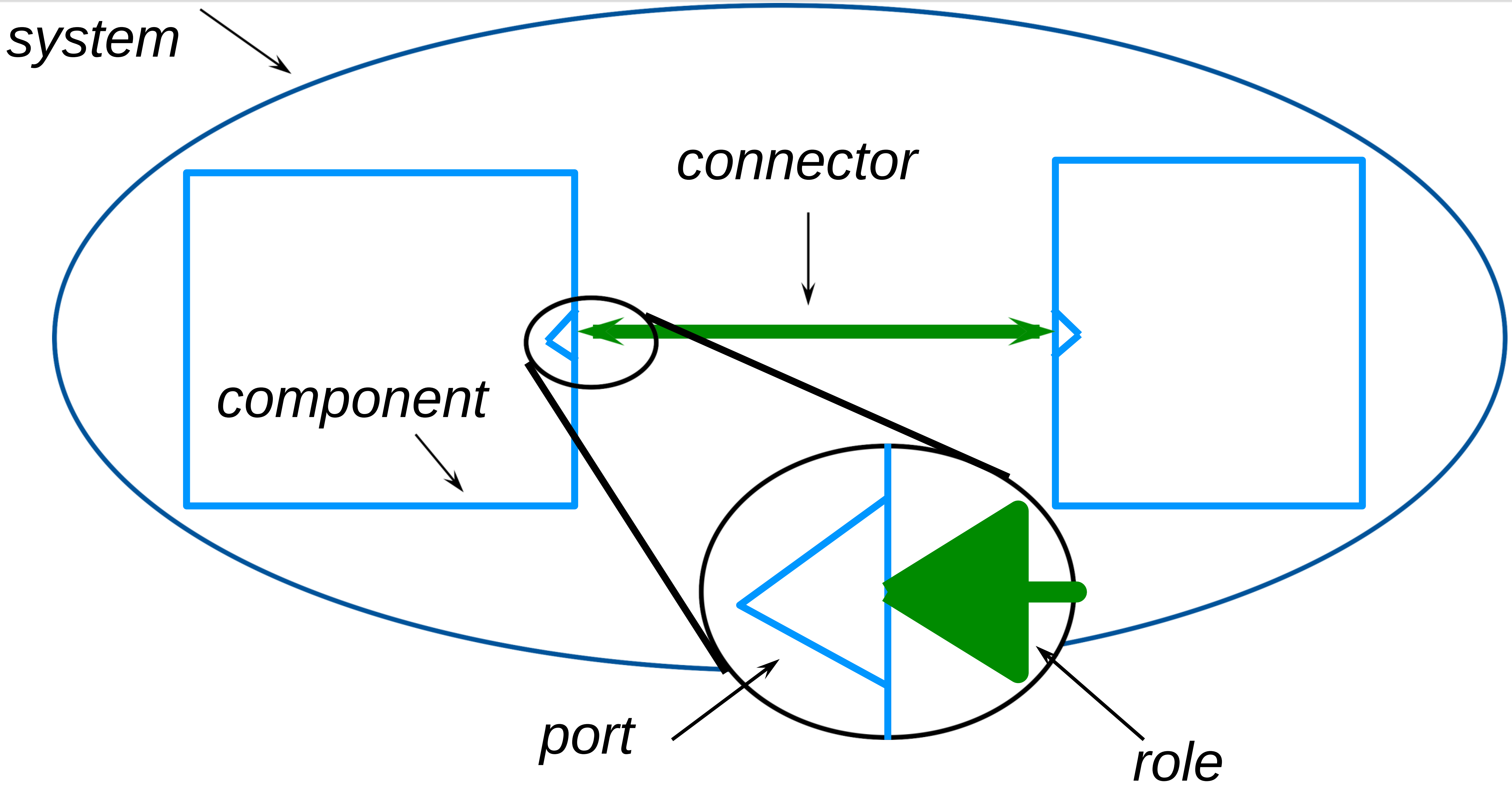
Description of Architecture

- Architecture includes：
  - Components:
    - e.g.: filters, databases, objects, clients/servers
  - Connectors:
    - e.g.: procedure call, pipes, event broadcast
  - Properties:
    - e.g.: signatures, pre/post conds, RT specs

体系结构包括：
- 组件：
  - •例如：过滤器、数据库、对象、客户端/服务器
- 连接器：
  - •例如：程序调用、管道、事件广播
- 属性：
  - •例如：签名，前/后条件，RT规格

Vocabulary of Architecture

中国科学技术大学
University of Science and Technology of China



Architectural Patterns

中国科学技术大学
University of Science and Technology of China

体系结构模式是一组适用于重复出现的设计问题的体系结构设计决策，并且参数化以说明问题出现时的不同软件开发上下文

- An architectural pattern is a set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears.

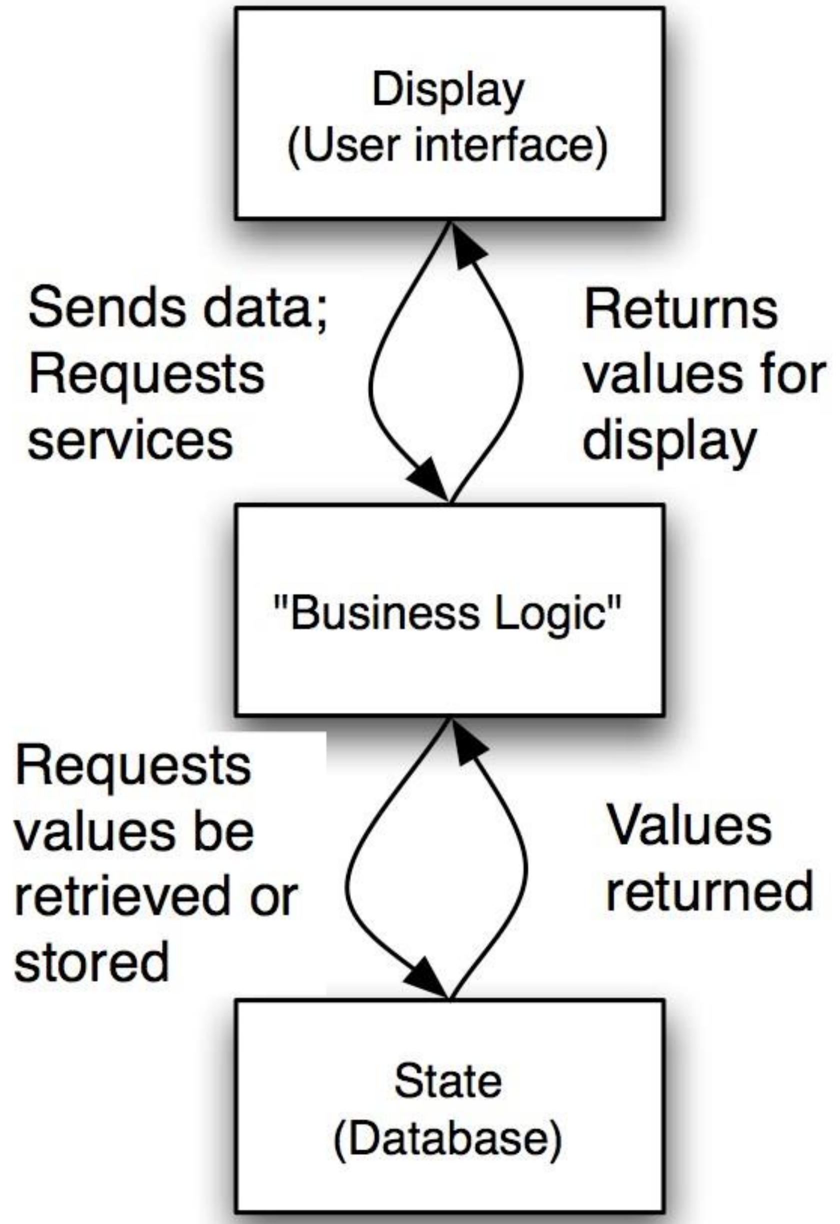
体系结构模式与DSSAs类似，但是应用于“较低的级别”和更小的范围

- Architectural patterns are similar to DSSAs but applied "at a lower level" and within a much narrower scope.

State-Logic-Display: Three-Tiered Pattern

- Application Examples
  - Business applications
  - Multi-player games
  - Web-based applications

应用实例
◆业务应用程序
◆多人游戏
◆基于web的应用程序

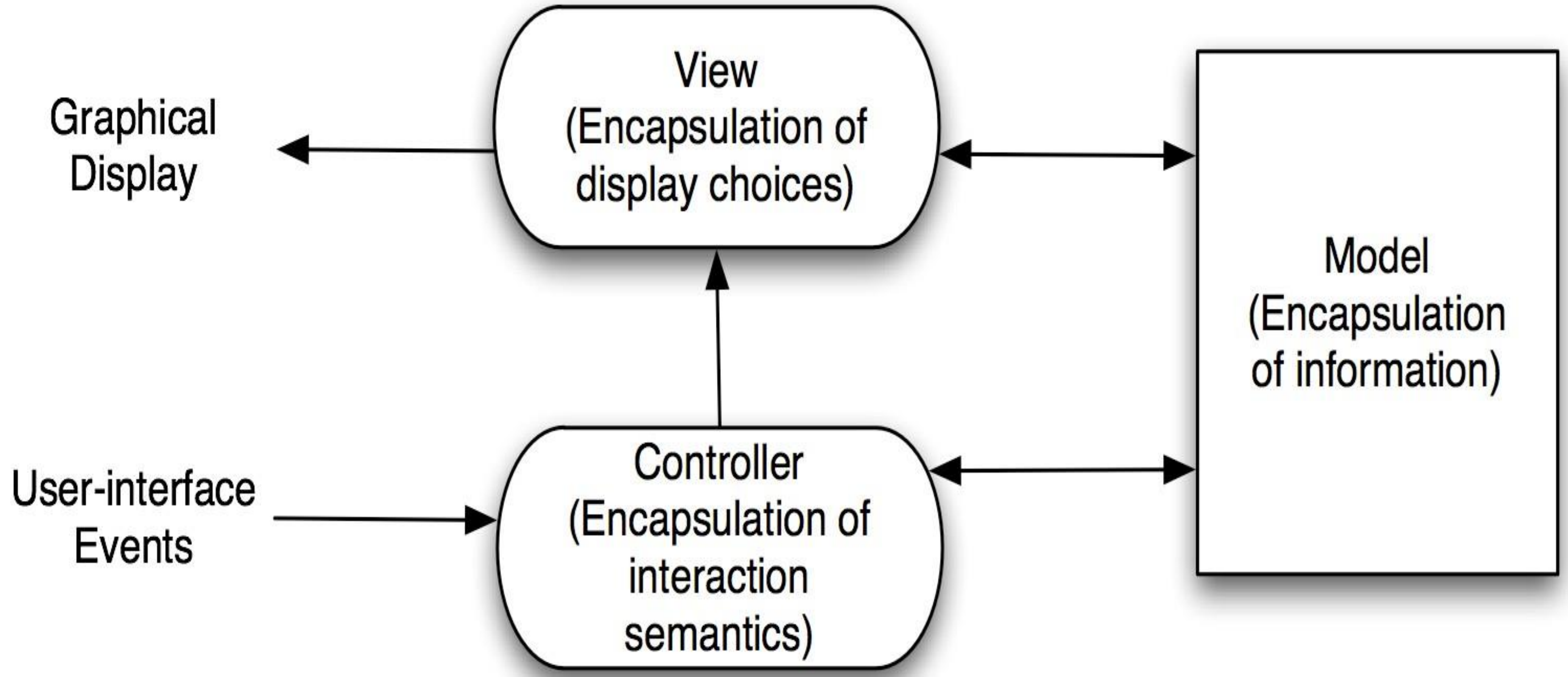


Model-View-Controller (MVC)

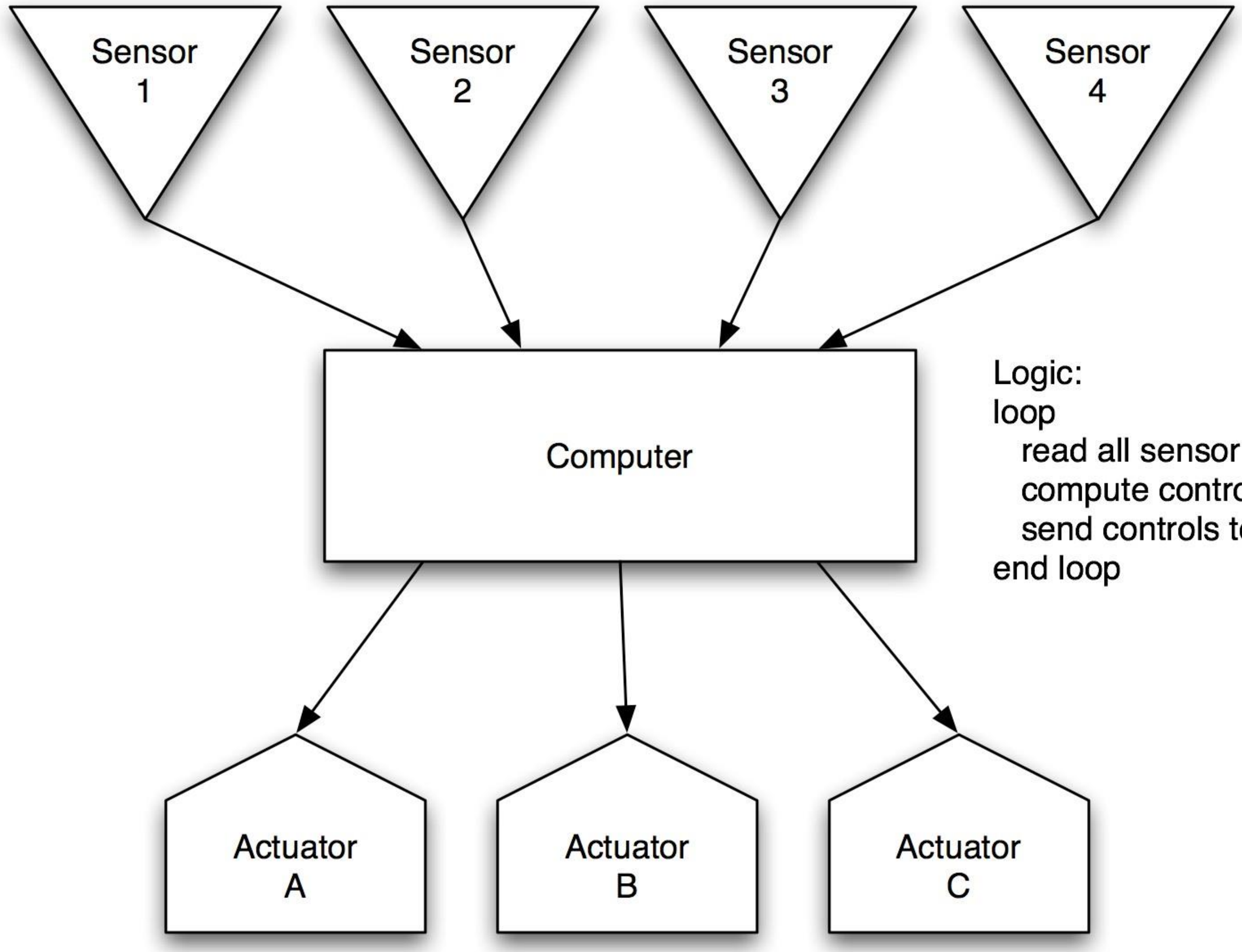
- Objective: Separation between information, presentation and user interaction.
  目标:分离信息、表示和用户交互
- When a model object value changes, a notification is sent to the view and to the controller. So that the view can update itself and the controller can modify the view if its logic so requires.
  当模型对象值发生变化时，一个通知被发送到视图和控制器。这样，视图可以更新自己，控制器可以修改视图，如果它的逻辑需要的话
- When handling input from the user the windowing system sends the user event to the controller; If a change is required, the controller updates the model object.
  当处理来自用户的输入时，窗口系统将用户事件发送给控制器;如果需要更改，控制器将更新模型对象

Model-View-Controller

中国科学技术大学
University of Science and Technology of China


Graphical Display
View (Encapsulation of display choices)
Model (Encapsulation of information)
User-interface Events
Controller (Encapsulation of interaction semantics)

Sense-Compute-Control



```
Logic:
loop
   read all sensor values
   compute control outputs
   send controls to all actuators
end loop
```

目的:构建嵌入式控制应用

Objective: Structuring embedded control applications

The Lunar Lander: A Long-Running Example

月球着陆器:一个长时间运行的例子

University of Science and Technology of China

一个简单的电脑游戏，第一次出现在1960年代

●简单的概念:

◆你(飞行员)控制阿波罗登月飞行器的下降速度

●节流设置控制引擎

●有限的燃料

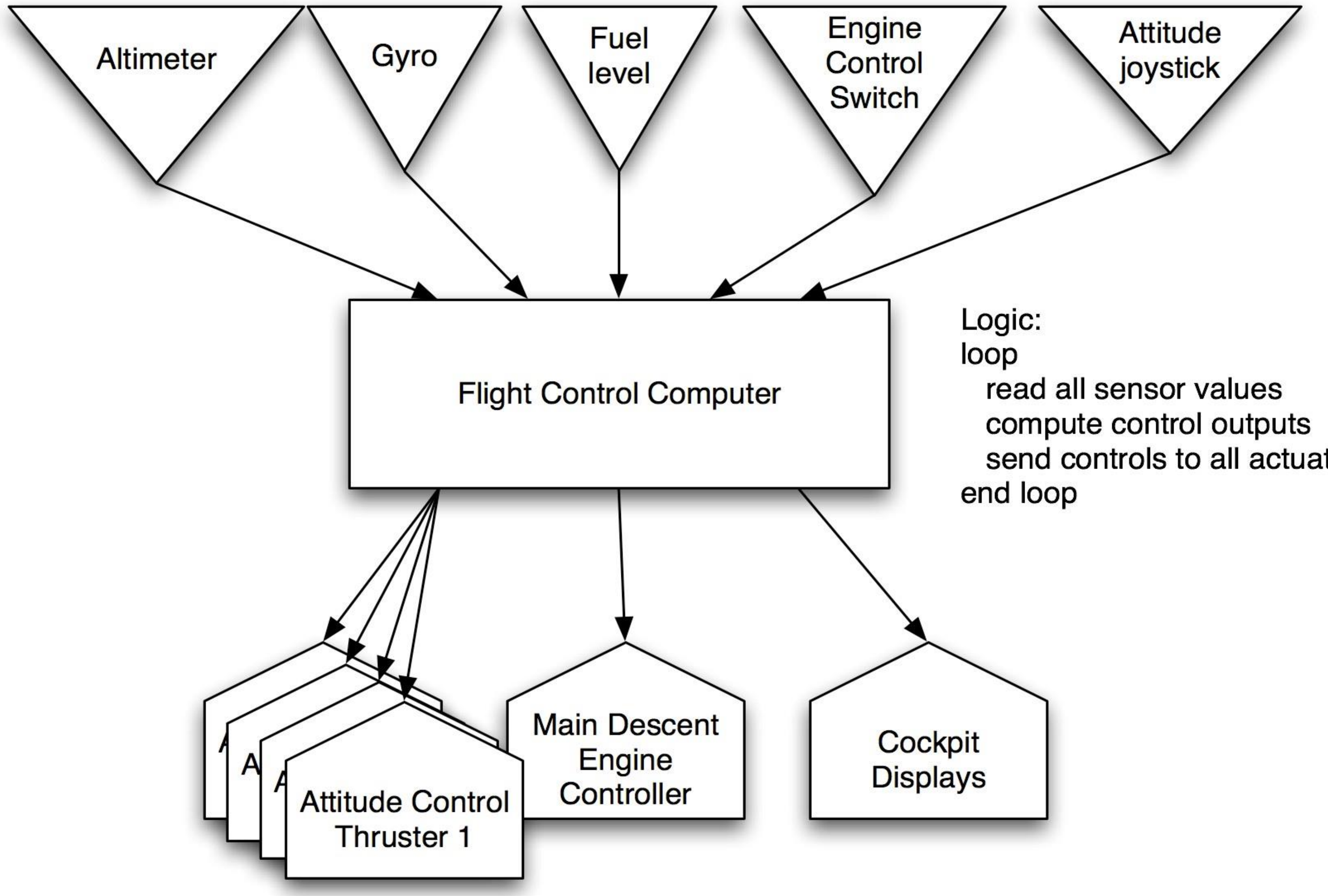
●初始高度和速度预设

●如果你以低于5帧/秒的下降速度着陆，你就赢了(不管有没有剩余的燃料)

◆高级版本:操纵杆控制姿态和水平运动

- A simple computer game that first appeared in the 1960’s
- Simple concept:
  - You (the pilot) control the descent rate of the Apollo-era Lunar Lander
    - Throttle setting controls descent engine
    - Limited fuel
    - Initial altitude and speed preset
    - If you land with a descent rate of < 5 fps: you win(whether there’s fuel left or not)
  - “Advanced” version: joystick controls attitude & horizontal motion

Sense-Compute-Control LL



```
Logic:
loop
   read all sensor values
   compute control outputs
   send controls to all actuators
end loop
```

Architectural Styles: Definition

体系结构风格是体系结构设计决策的命名集合

- An architectural style is a named collection of architectural design decisions that

  适用于给定的开发环境

  - are applicable in a given development context

  约束特定于该上下文中特定系统的架构设计决策

  - constrain architectural design decisions that are specific to a particular system within that context

  在每个产生的系统中引出有益的品质

  - elicit beneficial qualities in each resulting system

描述软件系统设计经验教训的主要方法

- A primary way of characterizing lessons from experience in software system design

styles比patterns反映更少的领域特性

- Reflect less domain specificity than architectural patterns
- Useful in determining everything from subroutine structure to top-level application structure

用于确定从子例程结构到顶级应用程序结构的所有内容

Basic Properties of Styles

中国科学技术大学
University of Science and Technology of China

- A vocabulary of design elements
  - Component and connector types; data elements
  - e.g., pipes, filters, objects, servers
- A set of configuration rules
  - Topological constraints that determine allowed compositions of elements
  - e.g., a component may be connected to at most two other components
- A semantic interpretation
  - Compositions of design elements have well-defined meanings
- Possible analyses of systems built in a style

- 设计元素的词汇
  - 组件和连接器类型;数据元素
  - 例如，管道、过滤器、对象、服务器
- 一组配置规则
  - 决定允许元素组成的拓扑约束
  - 例如，一个部件最多可与其他两个部件连接
- 语义解释
  - 设计元素的组成具有明确的含义
- 以一种风格构建的系统的可能分析

Benefits of Using Styles

中国科学技术大学
University of Science and Technology of China

- Design reuse
  - Well-understood solutions applied to new problems
- Code reuse
  - Shared implementations of invariant aspects of a style
- Understandability of system organization
  - A phrase such as “client-server” conveys a lot of information
- Interoperability
  - Supported by style standardization
- Style-specific analyses
  - Enabled by the constrained design space
- Visualizations
  - Style-specific depictions matching engineers’ mental models

- 设计重用
  能够很好地理解新问题的解决方案
- 代码重用
  风格的不变方面的共享实现
- 系统组织的可理解性
  像“客户-服务器”这样的短语传达了很多信息
- 互操作性
  以风格标准化为支撑
- 特定风格分析
  由受限的设计空间启用
- 可视化
  特定风格的描述与工程师的思维模式相匹配

Style Analysis Dimensions

- 什么是设计词汇?
  组件和连接器类型
- 容许结构模式是什么?
- 底层的计算模型是什么?
- 风格的重要不变量是什么?
- 其使用的常见例子是什么?
- 使用该风格的优（劣）势是什么?
- 风格的专门化是什么?

- What is the design vocabulary?
  - Component and connector types
- What are the allowable structural patterns?
- What is the underlying computational model?
- What are the essential invariants of the style?
- What are common examples of its use?
- What are the (dis)advantages of using the style?
- What are the style’s specializations?

Architectural Styles

中国科学技术大学 University of Science and Technology of China

- Data-flow styles
  - Pipe and filter
  - Batch sequential
- Data-centered architectures
  - blackboard
  - repository
- layer architectures
  - Abstraction layer architectures
  - N-tier architectures
- Notification architectures
  - publish/subscribe
  - point-to-point
  - Event based
- Network-Centred Style
  - Client-Server
  - Peer to peer

- 数据流方式
  - 管道和过滤器
  - 批次顺序
- 数据中心架构
  - 黑板
  - 库
- 层架构
  - 抽象层架构
  - n层体系结构
- 通知架构
  - 发布/订阅
  - 点对点
  - 基于事件的
- 网络中心风格
  - 客户端-服务器
  - 点对点

Architectural Styles

- Remote Invocation Architectures
  - Web services
  - Broker
  - Interpreters
- GUI Architectures (Interactive)
  - Model---View---Controller
  - Presentation---Abstraction---Control
- Adaptable Architectures
  - micro---kernel
  - reflection
- Transaction-Processing Architectures
- Others
  - process control
  - rule---based
- HeterogeneousArchitectures

- 远程调用的架构
  - Web服务
  - 代理
  - 解释器
- GUI架构(交互式)
  - 模型-视图-控制器
  - 演示-抽象-控制
- 具有适应性的体系结构
  - 微内核
  - 反射
- 事务处理架构
- 其他
  - 过程控制
  - 基于规则
- 异构

中国科学技术大学
University of Science and Technology of China

Data-flow Architectures

处理数据流的架构风格

Architecture style dealing with a stream of data.

数据流：与时间绑定的数据

Data-flow architectures

中国科学技术大学
University of Science and Technology of China

这些体系结构的目标是实现可重用性和可修改性的质量

- These architectures have the goal of achieving the quality of reuse and modifiability.

数据流样式的特点是将系统视为对连续输入数据的一系列转换

一段一段的数据输入（可看做链表）

- The data-flow style is characterized by viewing the system as a series of transformations on successive pieces of input data.

- Data enter the system and then flows through the components one at a time

数据进入系统，然后流经各个组件，一次一个

- Finally, the data is assigned to some final destination (output or a data store).

最后，数据被分配到某个最终目的地(输出或数据存储)

Data-flow architectures

中国科学技术大学
University of Science and Technology of China

batch-sequential是pipes和filters的降级版
batch-sequential是串行的，pipes和filters可以并行

数据流架构可以分为批处理顺序架构、管道和过滤器

- Data-flow architectures can be classified into batch-sequential architectures and pipes and filters

在批处理顺序样式中，在开始下一个步骤之前，每个步骤运行到完成

- In the batch sequential style each step runs to completion before the next step starts

UNIX命令行管道

- E.g. UNIX command line pipes

在管道和过滤器样式中，步骤可以并发运行，以增量方式处理数据的各个部分

- In pipe and filters style steps might run concurrently processing parts of data incrementally

并行：1.任务并行（pipes and filters架构，filter处理数据，处理完后通过pipe将数据传输到下个filter。filter之间存在同步问题，因为filter运行速度不同。）
2.数据并行（将大量的数据分成小块，在不同的进程里采用相同的方法处理。Master-Slave程序，Master分配数据，Slave处理数据，Slave处理完数据后通知Master。该程序加速比很高）

The Pipe-and-Filter Architectural Pattern

数据流以一种相对简单的格式通过一系列处理
数据持续输入管道;每个组件以某种方式转换数据

- **Streams of data, in a relatively simple format, passed through series of processes**
  - Data constantly fed into a pipeline; Each component transforms the data in some way.

这些进程可以并发地工作，不断的数据输入和输出

  - The processes work ***concurrently***. Constant data in and coming out.

非常灵活的架构

  - Very flexible architecture.

几乎所有的组件都可以移除

    —Almost all the components could be **removed**.

组件可以添加、更改、删除和重新排序

    —Components may **added**, **changed**, **deleted**, **reordered**…

非常灵活，特别是(例如)在转换数据或过滤(删除)字符或特征等方面

  - Very flexible particularly (for example) as in converting **data** or filtering out (removing) **characters** or '**features**', etc.

  - Sometimes (oftentimes) data might undergo a series of **transformations**…
  - Can also split pipelines or join pipelines together.

有时(通常)数据可能会经历一系列转换...
也可以分裂管道或连接管道

Pipe and Filter Style

- Components are filters
  - Transform input data streams into output data streams
  - Possibly incremental production of output
- Connectors are pipes
  - Conduits for data streams
- Style invariants
  - Filters are independent (no shared state)
  - Filter has no knowledge of up- or down-stream filters
- Examples
  - UNIX shell
  - Distributed systems

signal processing

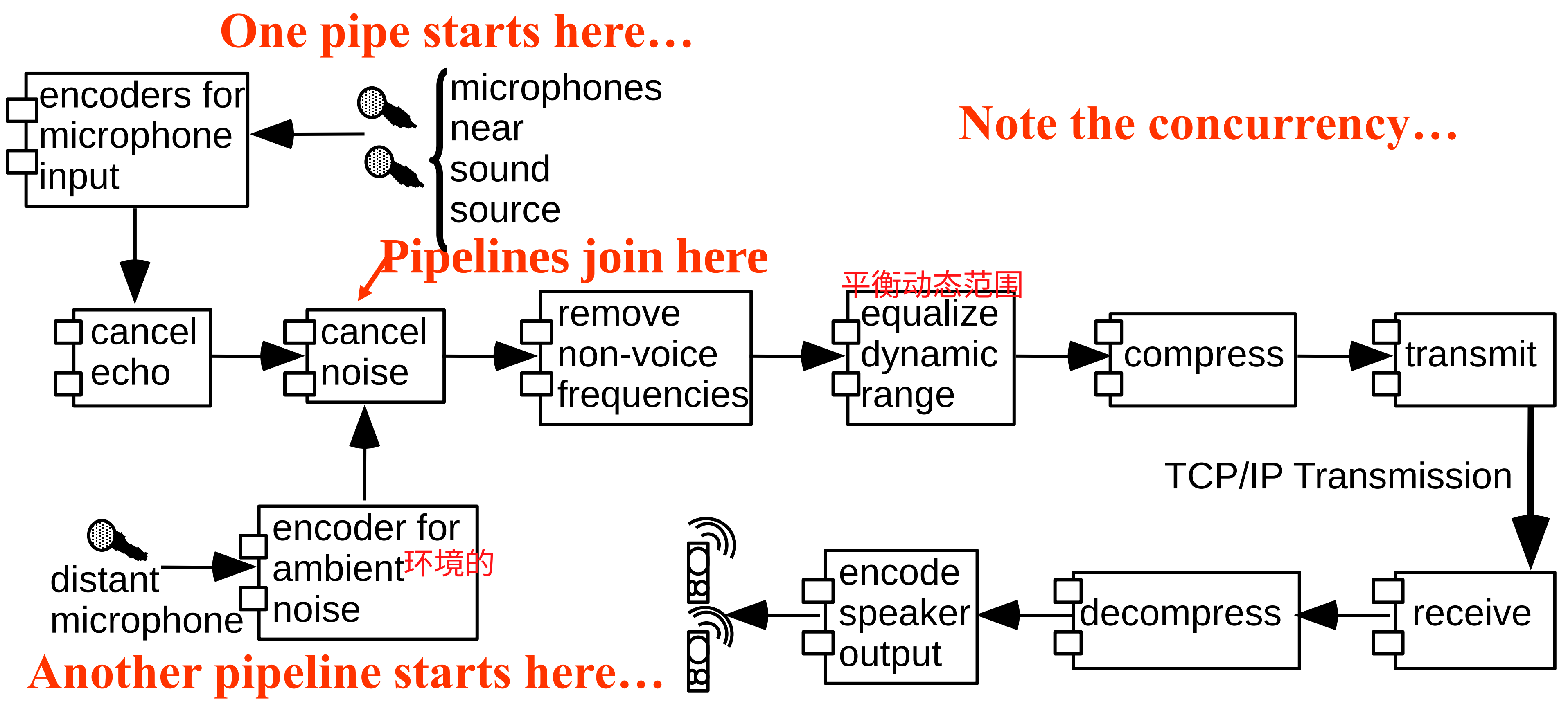
parallel programming

- Example: `ls invoices`

```
| grep -e August
| sort
```

●组件是过滤器
◆将输入数据流转换为输出数据流
◆可能增量产出
●连接器是管道
◆数据流管道
●风格不变量
◆过滤器是独立的(没有共享状态)
◆过滤器不了解上、下游过滤器
●例子
◆UNIX shell
◆分布式系统

Example Of A Pipe-and-filter System



Think in terms of manufacturing processes, process control applications or a GPS system.

Used more frequently in scientific/engineering systems than in information systems applications.

Note: questionable architecture for an information system….

- 想想制造过程，过程控制应用程序或GPS系统。
- 在科学/工程系统中比在信息系统应用中使用得更频繁。
- 注意:有问题的信息系统架构. . .

Pipes and filters

中国科学技术大学
University of Science and Technology of China

- Data flows through pipes: communication channels between filters

  Processing units: filters

· 数据通过管道流动:过滤器之间的通信通道
  处理单位:过滤器
· 根据过滤器驻留的位置，不同类型的执行架构可能适用
· 例如，相同的进程:过滤器在线程中运行
· 例如，同一台机器:过滤器在自己的进程中运行
· 例如，网络:管道使用网络架构

- Depending on where the filters reside different types of execution architectures might apply
- E.g. same process: filters run within threads
- E.g. same machine: filters run within own processes
- E.g. network: pipes use the networking architecture

变体
· 管道-过滤器的线性序列
· 有界管道-管道上的数据量有限
· 类型管道-数据强类型

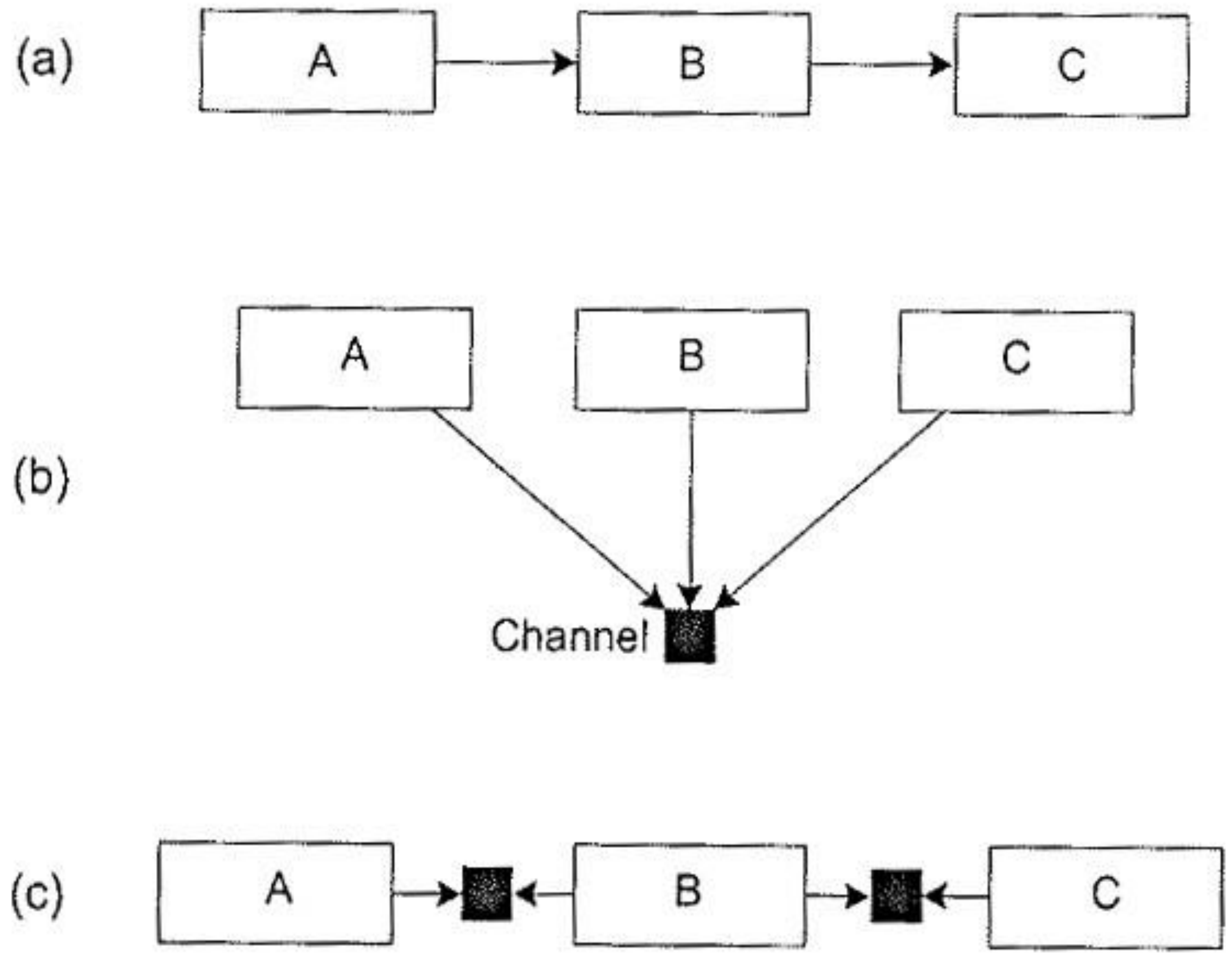
- Variations
  - Pipelines — linear sequences of filters
  - Bounded pipes — limited amount of data on a pipe
  - Typed pipes — data strongly typed

Pipes and filters

- Conceptually filters consume data from input and write data to output Input and output: communication channels, i.e. pipes
- Filters do not know anything about other filters (loose coupling between the components)
- Ideally they are completely independent from each other
- Data flows in streams: good for processing of images, audio, video, ...

· 从概念上过滤器从输入消耗数据并将数据写入到输出，输入和输出:通信通道，即管道
· 过滤器不知道其他过滤器的任何信息(组件之间的松散耦合)
· 理想情况下，它们是完全独立的
· 数据流:适合处理图像、音频、视频等

Pipes and filters



Pipes and filters

中国科学技术大学 University of Science and Technology of China

- Variations: **structural** and **communicational**
- Structural: more complex topologies might be used

· 变化:结构和交流
· 结构:可以使用更复杂的拓扑,例如,循环，分支，多个输入...管道术语用于过滤器的线性序列
· 通信:过滤器是否被阻塞并等待数据?有界管道术语用于管道中有限的数据量

- E.g. loops, branches, more than one input, ... Term **pipeline** used for linear sequence of filters
- Communicational: are filters blocked and wait for data? Term **bounded pipe** for limited amount of data in the pipe

Pipes and filters

中国科学技术大学
University of Science and Technology of China

- 管道中的数据结构是什么?如果数据是结构化的，管道中的所有组件必须同意术语--类型管道
- 数据结构越具体，耦合就越紧密

- What is the data-structure within the pipe? All components in the pipe have to agree Term **typed pipe** if data is structured
- The more specific the data-structures are, the tighter the coupling

Pipes and filters

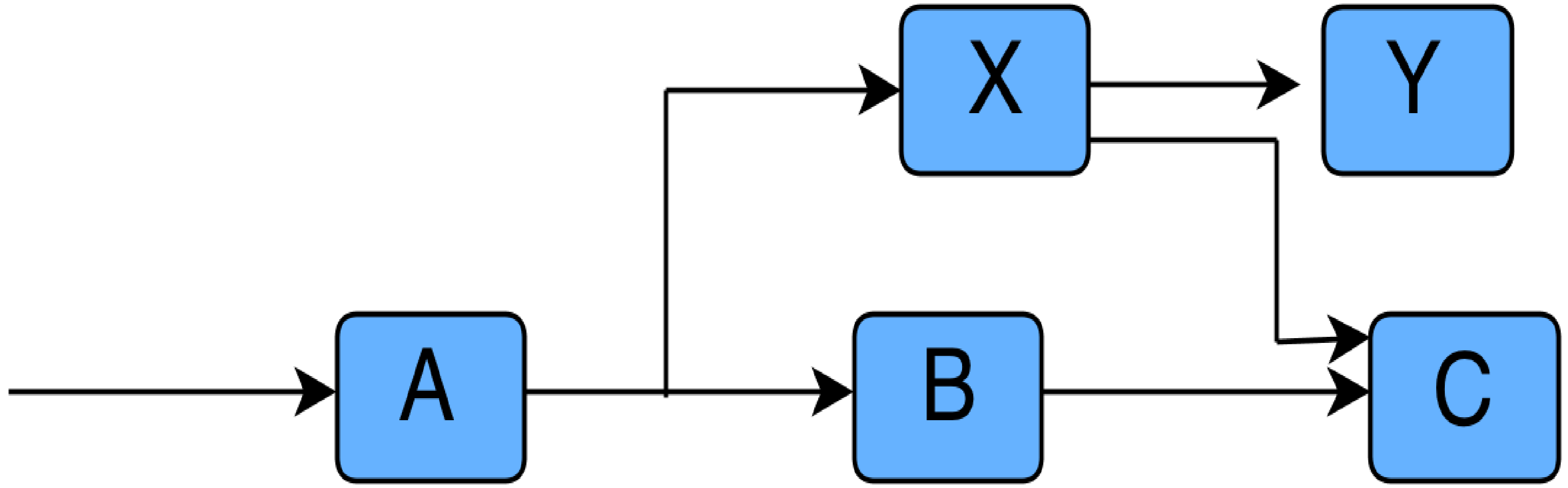
如何使用数据?
流，或一次性全部数据(批处理顺序)，或数据块

- How is the data consumed? Streaming, or
- All data at once (batch sequential), or Chunks of data

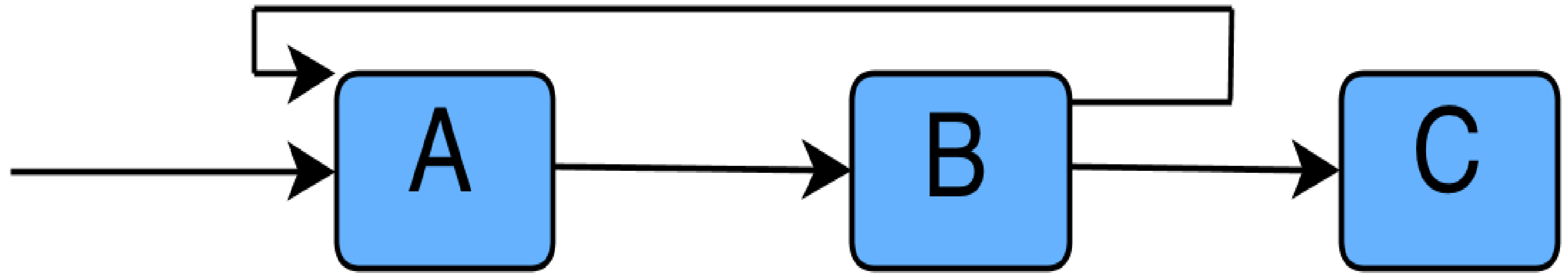
Pipes and filters

中国科学技术大学 University of Science and Technology of China

(a)



(b)



Pipes and filters - advantages

- Pipes are conceptually simple (helps maintainability)

  Components can be reused

- Easy to add and remove components (helps evolvability)
- Allow injection of special components to address cross-cutting concerns
- E.g. monitor throughput, logging, ...
- Allow concurrent/parallel execution (helps scalability)

· 管道在概念上是简单的(有助于维护)
组件可以重用
· 易于添加和删除组件(有助于可进化性)
· 允许注入特殊组件来解决横切问题
· 例如，监控吞吐量，日志记录，…
· 允许并发/并行执行(有助于可伸缩性)

Pipes and filters - disadvantages

- 管道经常导致批量处理
  Pipes often lead to batch processing
- 因此不适合交互式应用程序
  Therefore not well suited for interactive applications
- 很难实现增量更新
  E.g. hard to implement incremental updates
- 数据结构的最小共同特性
  Lowest common denominator for data structure
- 每个过滤器必须解析/打包数据(不利于性能)
  Each filter has to parse/unparse the data (bad for performance)
- 增加每个组件的复杂性
  Adds complexity to each component

Pipe-and-Filter Architecture: Design Principles

- 1. *Divide and conquer*: The separate processes can be **independently** designed. **Excellent for manufacturing / process control / etc. systems.**

- 2. *Increase cohesion*: The processes have ***functional cohesion***. (single input; single output; no side effects…)

- 3. *Reduce coupling*: The processes have only ***one*** input and ***one*** output.

- 4. *Increase abstraction*: The pipeline components are often good abstractions, ***hiding*** their internal details. And components are usually replacable!

- 5. *Increase reusability*: The processes can often be used in **many** different contexts.

- 6. *Increase reuse*: It is often possible to find reusable components to insert into a pipeline.

1. 分而治之:可以独立地设计独立的进程。适用于制造/过程控制等系统。
2. 增加内聚性:进程具有功能内聚性。(单输入;单输出;无副作用)
3. 减少耦合:进程只有一个输入和一个输出。
4. 增加抽象:管道组件通常是很好的抽象，隐藏了它们的内部细节。而且组件通常是可替换的!
5. 增加可重用性:进程通常可以在许多不同的上下文中使用。
6. 增加重用性:经常可以找到可重用的组件插入到管道中

Batch Sequential

批处理
-一个降级版本的管道和过滤器风格。
-批处理模型还强调数据在过程步骤间传递，
-然而，数据是以块传输，
-数据不会转移到下一个过程步骤，直到整个块被处理。
-每一个过程的步骤是独立于任何其他步骤

- Batch
  - A degraded version of Pipers and Filters style.
  - Batch model also emphasizes that the data is transferred among process steps,
  - However, the data is transferred by blocks, and
  - The data is not transferred to next process step until the whole block is processed.
  - Each process step is independent of any other steps

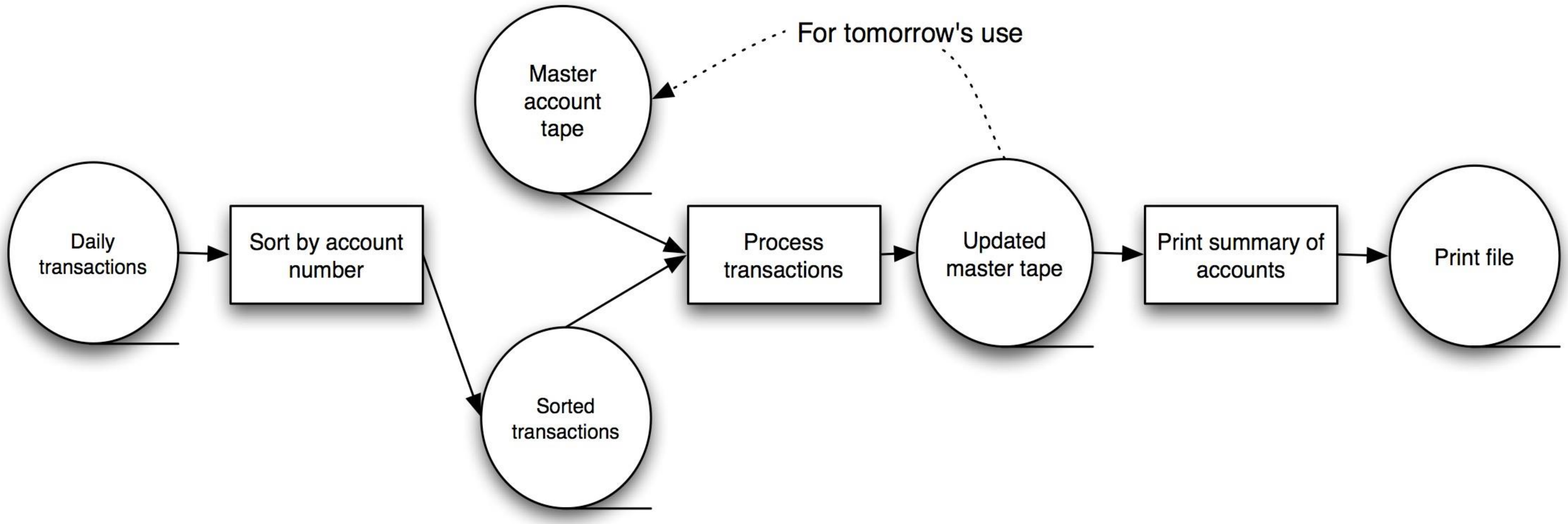
Batch Sequential

中国科学技术大学
University of Science and Technology of China

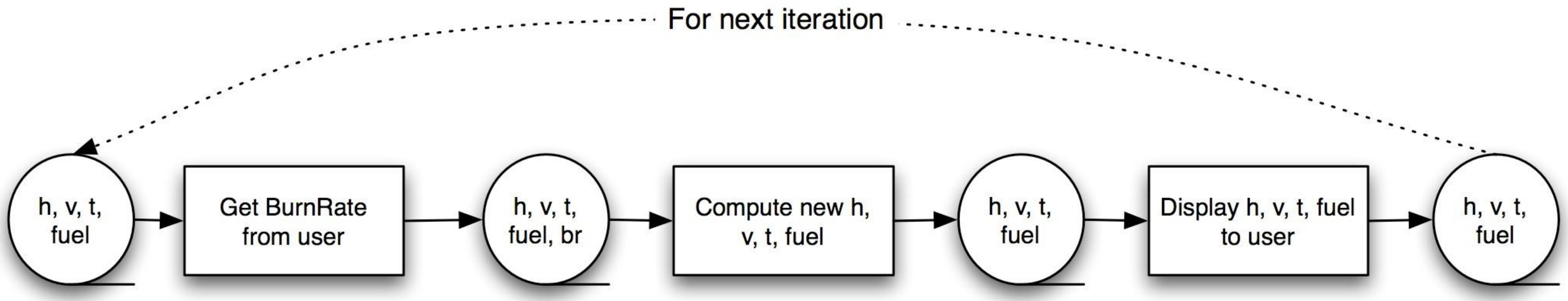
- ◆Separate programs are executed in order; data is passed as an aggregate from one program to the next.
- ◆Connectors: "The human hand" carrying tapes between the programs, a.k.a. "sneaker-net"
- ◆Data Elements: Explicit, aggregate elements passed from one component to the next upon completion of the producing program's execution.
- ●Typical uses: Transaction processing in financial systems. "The Granddaddy of Styles"

· 多个程序按顺序执行;数据以聚合的形式从一个程序传递到下一个程序。
· 连接器:"人类之手"在程序之间携带磁带，又名"运动鞋网"。
· 数据元素:在生产程序执行完成后，从一个组件传递到下一个组件的明确的、聚合的元素。
· 典型用途:金融系统的事务处理。"风格的鼻祖"

Batch-Sequential: A Financial Application



Batch-Sequential LL



这可不是成功登月的秘诀！

Not a recipe for a successful lunar mission!

Data-centered Architecture

Architectures focused on data integrity.

Data-centered architectures

中国科学技术大学
University of Science and Technology of China

这些架构的目标是实现数据的可集成性

- These architectures have the goal of achieving the quality of integrability of data.

这个术语指的是访问和更新广泛访问的数据存储是其主要目标的系统

- The term refers to systems in which the access and update of a widely accessed data store is their primary goal.

基本上，它只不过是一个与许多客户机通信的集中式数据存储

- Basically, it is nothing more than a centralized data store that communicates with a number of clients

对于这种样式，重要的是三种协议:通信协议、数据定义协议和数据操作协议

- Important for this styles are three protocols: communication, data definition and data manipulation protocol

Data-centered architectures

通信手段区分了这两种子类型

- The means of communication distinguishes the two subtypes:
- **repository** and **blackboard**

Repository:客户端向系统发送请求以执行必要的操作(例如插入数据)

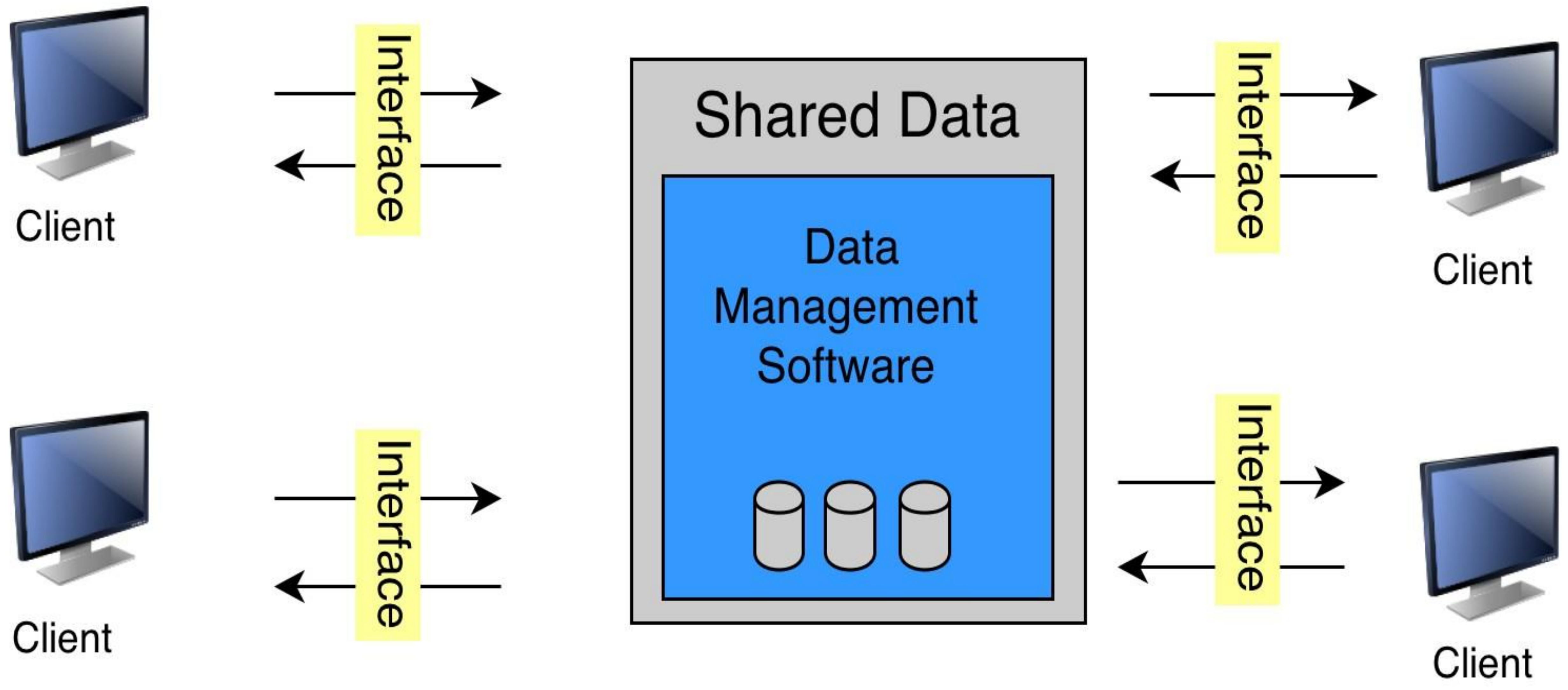
- Repository: a client sends a request to the system to perform a necessary action (e.g. insert data)

Blackboard(需要Notification架构支持):当用户感兴趣的数据发生变化时，系统会向用户发送通知和数据，所以是主动的

- Blackboard: the system sends notification and data to subscribers when data of interest changes, and is thus active

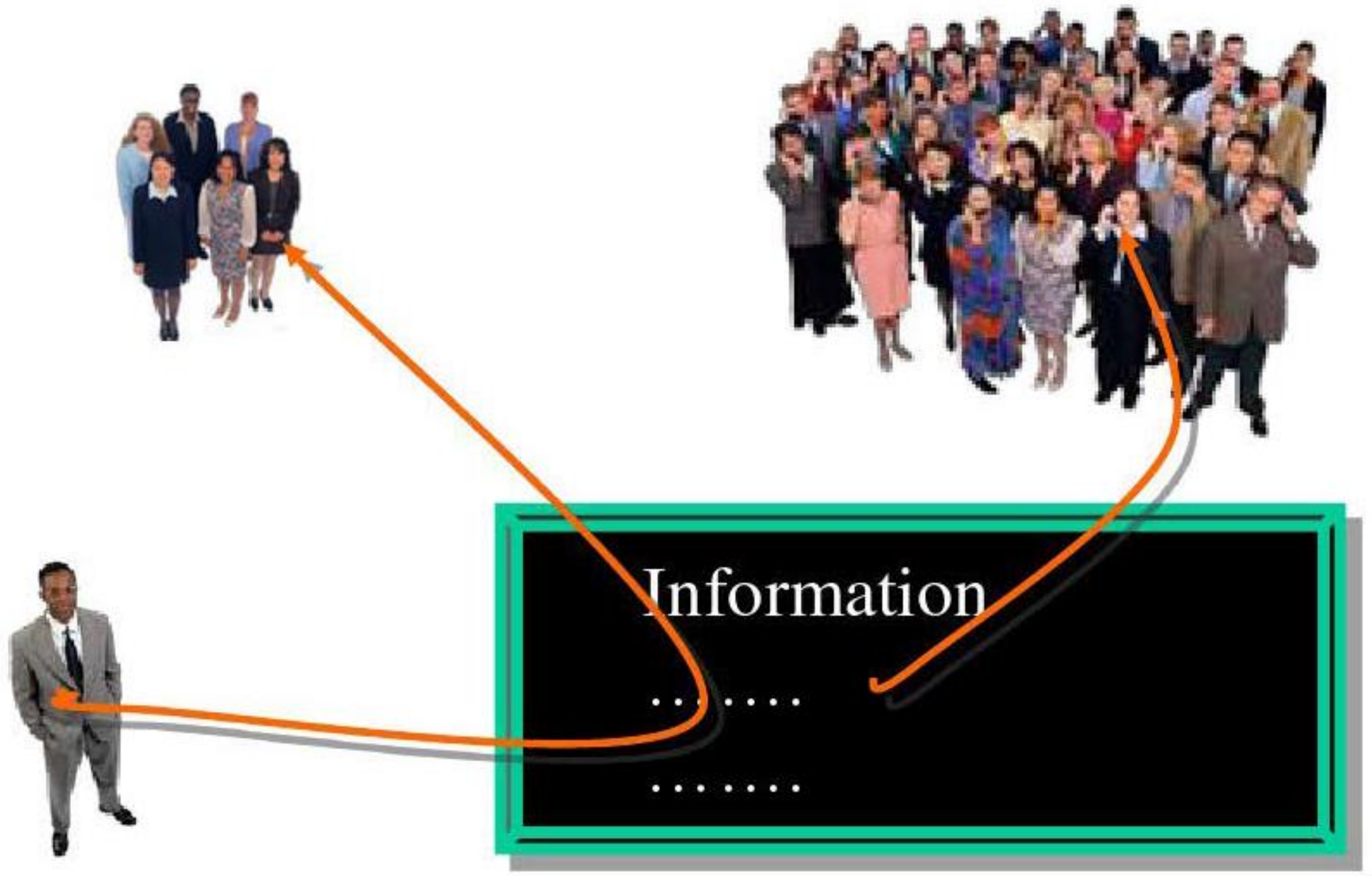
短连接：客户端与服务器传输数据（Repository）后，连接关闭，减轻服务器的压力
长连接：客户端与服务器建立连接后，连接不会断开，服务器可与客户端主动通信（Blackboard），服务器压力较大
服务器不知道客户端的ip地址，只能采用长连接方法

Repository

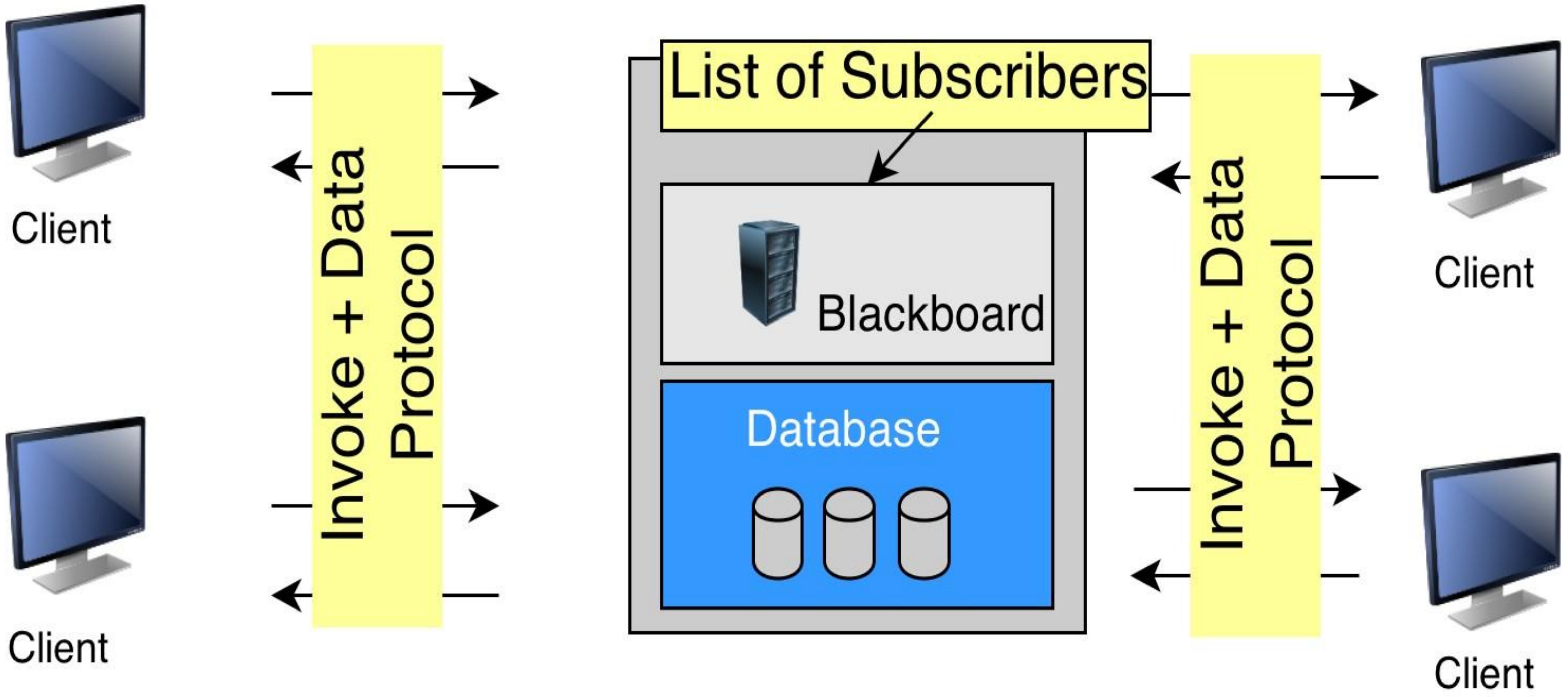


Data-Centered Style

Blackboard



Blackboard



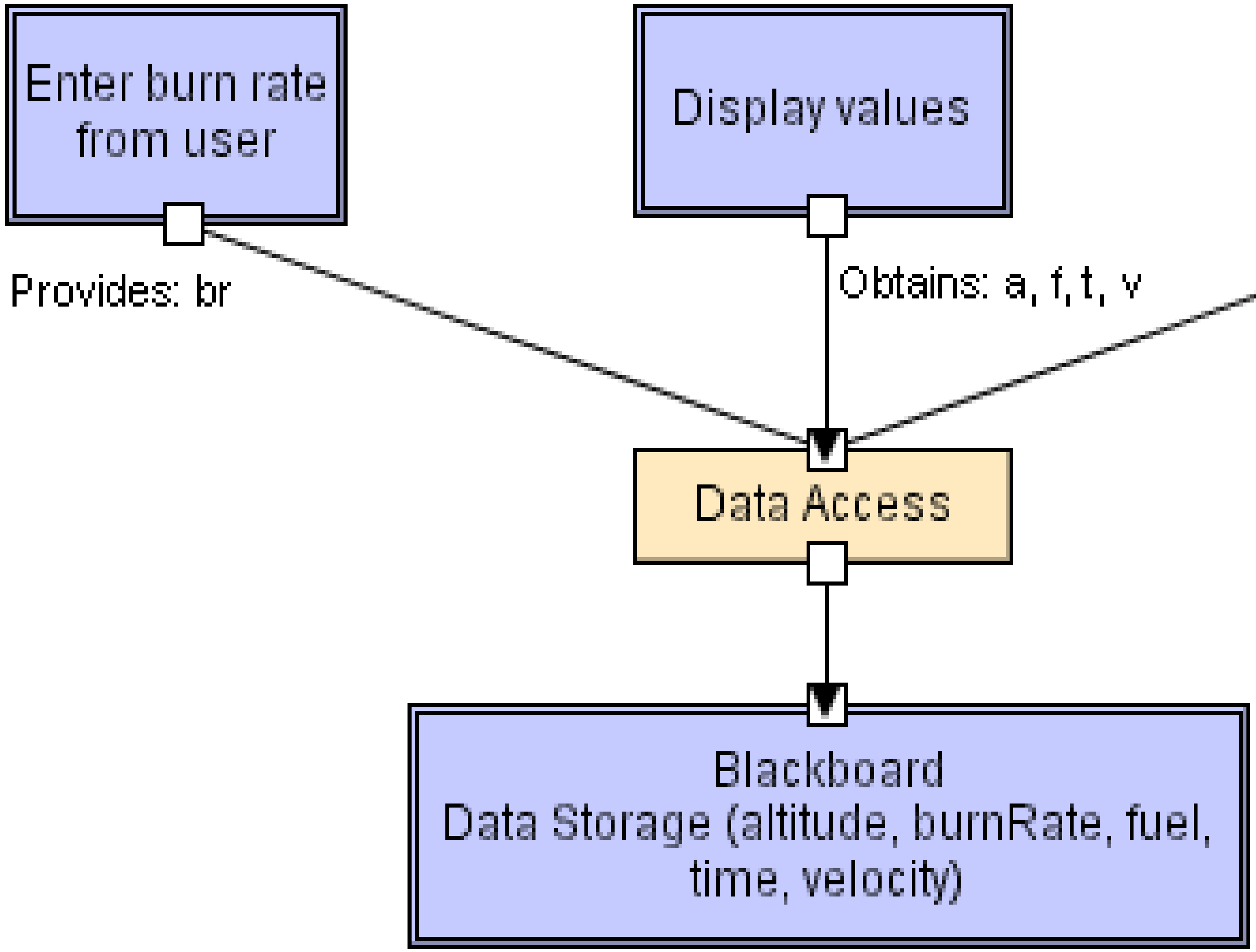
Data-Centered Style

Blackboard Style

- Two kinds of components
  - 核心数据结构（数据库）
  - Central data structure — blackboard
  - 在Blackboard上操作的组件
  - Components operating on the blackboard
- 系统控制完全由Blackboard状态驱动
- System control is entirely driven by the blackboard state
- Examples
  - Typically used for AI systems
  - Integrated software environments (e.g., Interlisp)
  - Compiler architecture

◆通常用于人工智能系统
◆集成软件环境(如Interlisp)
◆编译器架构

Blackboard LL


Enter burn rate from user
Display values
Compute new values and Update
Provides: br
Obtains: a, f, t, v
Obtains: a, br, f, t, v
Provides: a, f, t, v
Data Access
Blackboard
Data Storage (altitude, burnRate, fuel, time, velocity)

Database architecture

中国科学技术大学
University of Science and Technology of China

- One of the most well-known examples of the data-centered
  以数据为中心的体系结构最著名的例子之一是数据库体系结构
  architecture, is a database architecture
- There is a common database schema (i.e. meta-structure of
  有一个通用的数据库模式(即存储库的元结构)--由数据定义协议创建
  the repository) - created with data definition protocol
- E.g. in RDBMS a set of related tables with fields, data types,
  例如，在RDBMS中，一组具有字段、数据类型、键的相关表
  keys, ...

Database architecture

客户端使用数据操作协议来处理数据

- Clients use data manipulation protocol to work with the data

例如，SQL用于插入、选择、删除数据

- E.g. SQL for inserting, selecting, deleting data,…
- Depending on where clients are located communication protocol might be

根据客户机的位置，通信协议可能是进程内部通信

- An inner-process communication
- Communication between components on the same machine Communication over network, e.g. LAN, Internet, etc.

同一台机器上的部件之间的通信，通过网络进行通信，如局域网、Internet等

Database architecture

中国科学技术大学 University of Science and Technology of China

具有共享数据的管道类似于以数据为中心的架构
- Pipes with shared data are similar to data-centered architectures

Blackboard架构传统上用于复杂系统，相对来说Blackboard架构代价较大
- Blackboard architecture traditionally used for complex systems
  语音与模式识别
  - E.g. Speech and pattern recognition

允许松散耦合的组件
- Allows loosely coupled components

Data-centered architectures: advantages

确保数据的完整性
- Ensures data integrity

可靠、安全、可测试性得到保证
- Reliable, secure, testability guaranteed

客户端独立于系统:客户端的性能和可用性通常都很好
- Clients independent from the system: performance and usability on the client side is typically good

Data-centered architectures: disadvantage

可伸缩性、可靠性方面的问题(单点故障)

- Problems with scalability, reliability (single point of failure)

解决方案:共享存储库、复制，但这会增加复杂性

- Solutions: shared repositories, replication but this increases complexity

哪些功能在数据库中，哪些在客户端中，界限不清

- Unclear border which functionality lies in the DB and which in the client

数据分片(将数据分成不同部分存入不同的数据库)只能提高scalability和performance，但不能提高reliability
用户可以并发访问数据库，但数据只有一份
分片规则存储在Gateway，访问数据库前需访问Gateway，这可能会成为性能瓶颈

数据复制(分布式复制数据库)能提高scalability,performance和reliability，但会引起数据一致性问题

Web architecture

中国科学技术大学
University of Science and Technology of China

以数据为中心的架构的另一个例子是Web架构

- Another example of data-centered architectures is the Web architecture

超媒体数据模型后面有一个通用的数据模式(即Web的元结构)

- There is a common data schema (i.e. meta-structure of the Web) Follows hypermedia data model

页面(节点)和节点之间的链接，以及寻址机制

- Pages (nodes) and links between them, as well as addressing mechanism

Web architecture

中国科学技术大学
University of Science and Technology of China

数据操作不是直接在系统中进行的，通常是通过应用程序实现的

- Data manipulation not directly in the system but typically achieved through applications

尽管HTTP协议有数据操作的方法

- Although HTTP protocol has methods for data manipulation

通信协议是HTTP

- Communication protocol is HTTP

分析:不保证完整性(404错误)，但非常可伸缩

- Analysis: integrity not guaranteed (404 error) but extremely scalable

Resource-Oriented Architecture

这样的架构将Web看作是一个巨大的分布式数据库

- Such architectures look onto the Web as a huge distributed database

数据模型:节点可通过URL寻址并相互链接

- Data model: nodes addressable by URL and interlinked

数据操作:HTTP方法(GET, PUT, POST, DELETE)

- Data manipulation: HTTP methods (GET, PUT, POST, DELETE)

可扩展性、良好的性能、可用性等

- Scalable, good performance, usability, etc.

Layered Architectures

具有明确职责层的架构

Architectures with clear layers of responsibilities.

The Multi-Layer Architectural Pattern

- **Layered architecture:**

—layers communicate down!

—Normally immediately below with few ‘skips’

—Is the classical approach.

—The higher layer sees the lower layers as a set of ***services***.

—This notion is **fundamental to good design.**

—Often, a layer communicates **ONLY** with the layer below it

—**not always** - but normally.

分层架构:
· 层向下通信（自顶向下）
· 通常紧接下面，很少有“跳跃”
· 是经典的方法
· 较高层将较低层视为一组服务
· 这个概念是良好设计的基础
· 通常，一个层只与它下面的层通信
· 不总是这样，但通常是这样

Abstract layer architectures

中国科学技术大学
University of Science and Technology of China

分层:系统的结构被组织成一组层，每一层位于另一层的顶部

- Layering: the structure of the system is organized into set of layers Each layer is on the top of another layer

层之间定义良好的接口

- Well-defined interfaces between layers

降低复杂性，提高模块化，可重用性，可维护性，分层的不同标准:最明显的是抽象

- Reduces complexity, improves modularity, reusability, maintainability Different criteria for layering: most notably abstraction

Abstract layer architectures

- **Topological constraints**
- Number of layers?
- Limit access to next lower layer? Access to layer above?
- Sometimes abstraction is not completely opaque

- 拓扑约束
- 层数?
- 限制访问下一层?访问上层?
- 有时抽象不是完全不透明的

Layered Style

- Hierarchical system organization
  - “Multi-level client-server”
  - Each layer exposes an interface (API) to be used by above layers
- Each layer acts as a
  - *Server:* service provider to layers “above”
  - *Client:* service consumer of layer(s) “below”
- Connectors are protocols of layer interaction
- Example: operating systems
- *Virtual machine* style results from fully opaque layers

- 层次系统组织
  “多层次的客户端-服务器”
  每一层都公开一个供上面各层使用的接口(API)
- 每一层充当
  服务器:“上”层的服务提供商
  客户端:“下”层的服务消费者
- 连接器是层交互的协议
- 例子:操作系统
- 虚拟机样式来自完全不透明的层

Multi-Layered Architectural Pattern...

- Built with layers at **increasing levels of abstraction.**
  - —1. **User Interface layer -** normally first for **presentation**
  - —2. **Application Layer is usually** <u>immediately below</u> UI layer and **typically provides** the <u>**application functions**</u> determined by application use-cases. (application layer)
  - —3. **Domain Layer is usually next and** provides **general domain-level services** (business use-cases)
  - —4. **Services / Support (Bottom) layers** provide **general (but essential) services**.
    - » e.g. network communication, database access
    - » operating system services

在不断增加的抽象级别上使用层构建。
1. 用户界面层——通常首先用于表示
2. 应用层通常紧接在UI层之下，通常提供由应用程序用例确定的应用程序功能。(应用层)
3. 域层通常是下一层，它提供一般的域级服务(业务用例)
4. 服务/支持(底层)层提供一般的(但基本的)服务。
   例如，网络通信、数据库访问
   操作系统服务

Extremely Nice Feature of Layered Design

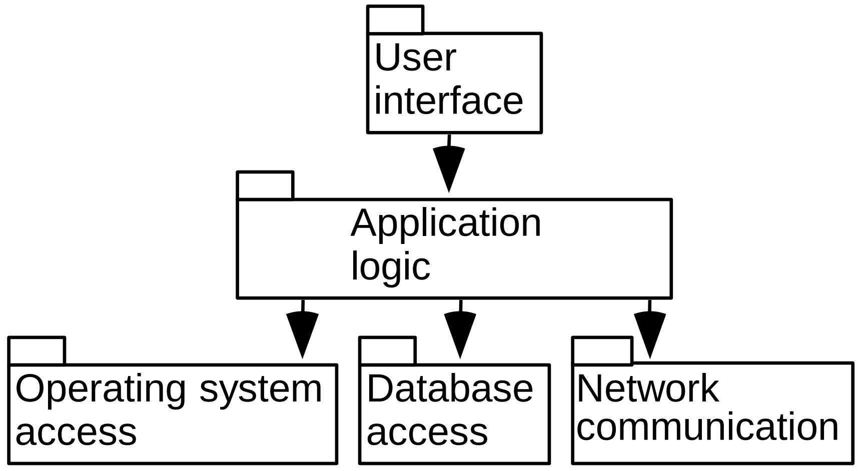
层/层服务是可替换的
-没有影响到其他层和依赖
-如果接口保持不变。
-例子:
当移植到不同的平台或环境时，用户界面层。
升级/增强/优化服务……
-我们有明确的关注点分离
-我们的服务有很好的“内聚”……

— Layers / layer services are **replaceable**

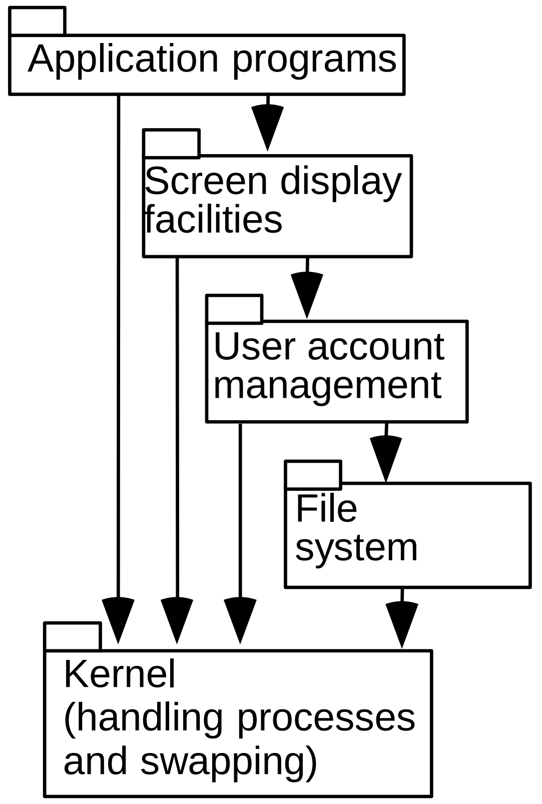
- NO impacting to other layers and dependencies
- **if the interfaces** remains unchanged.
- Examples:
  - » User Interface layer when porting to a different platform or for different environments.
  - » Upgrading / enhancing / optimizing services…
- We have **clear separation of concerns**
- We have very good ‘**cohesion**’ of services…

Example Of Multi-layer Systems

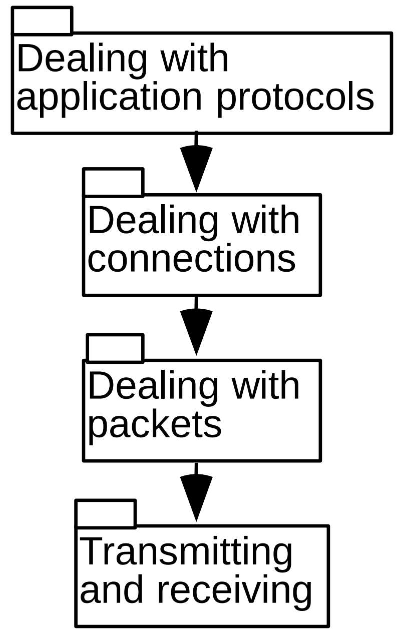
层间通信:通常使用过程调用
上层成为客户，下层成为服务器



a) Typical layers in an application program


Application programs
Screen display facilities
User account management
File system
Kernel (handling processes and swapping)

b) Typical layers in an operating system


Dealing with application protocols
Dealing with connections
Dealing with packets
Transmitting and receiving

c) Simplified view of layers in a communication system

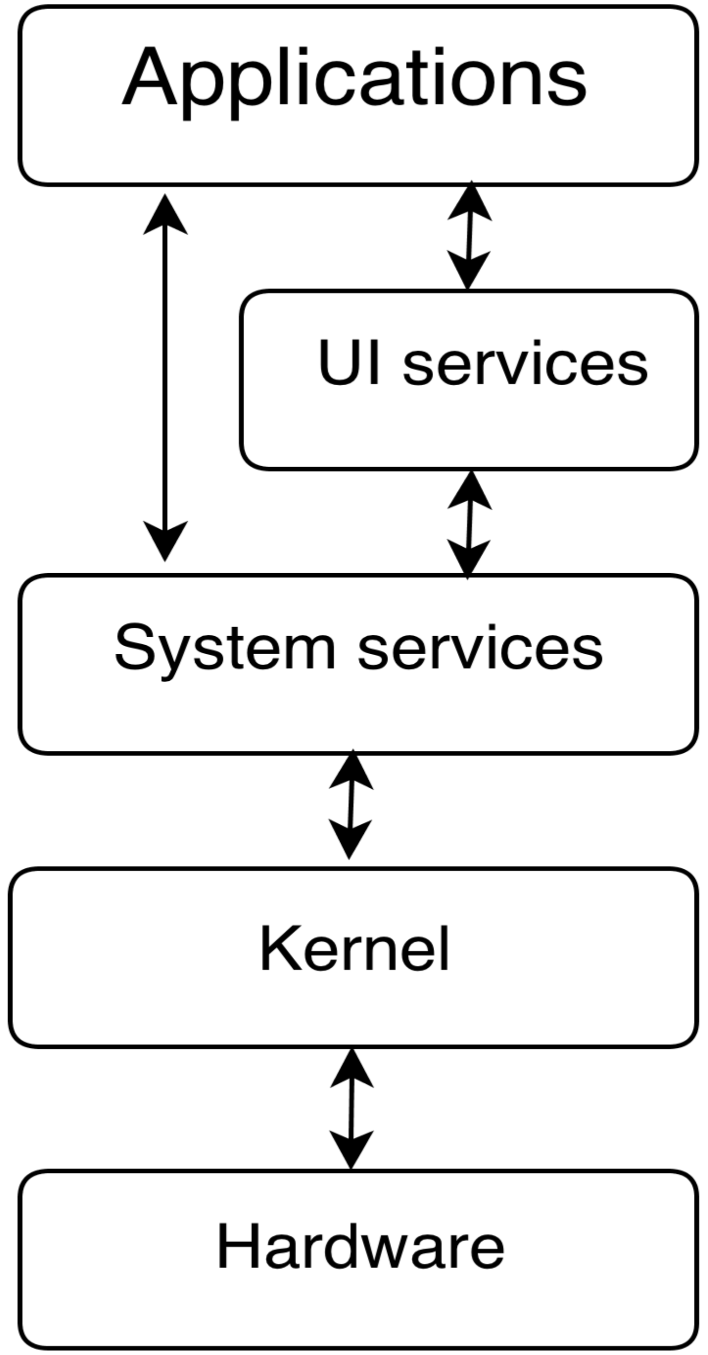
Communications between layers: usually use procedure calls.
Upper layers become clients; lower layers become servers.

Operating systems

- Typically organized into layers
- Successive layers provide more sophisticated services to the layers above them
- Hardware services, kernel services, system services, UI services

· 通常组织成层
· 连续层为上层提供更复杂的服务
· 硬件服务、内核服务、系统服务、UI服务

Figure: Operating system



Peer-to-peer protocol stacks

中国科学技术大学
University of Science and Technology of China

- The lowest layer handles communication between two computers
- The internet layer handles routing of packets accross the network
- The transport layer guarantees that packets are error-free and that they are received ir the same order as they are sent
- The application layer supports application-specific protocols
- E.g. HTTP, SMTP, FTP, ...
- The layers constitute a series of increasingly abstract operations
- Higher layers span a virtual connection between computers on the network

· 最低层处理两台计算机之间的通信
· internet层处理跨网络数据包的路由
· 传输层保证包是没有错误的，并且它们是按照发送的顺序被接收的
· 应用层支持应用程序特定协议，例如，HTTP，SMTP，FTP，…
· 层次构成了一系列日益抽象的操作
· 高层跨越网络上计算机之间的虚拟连接

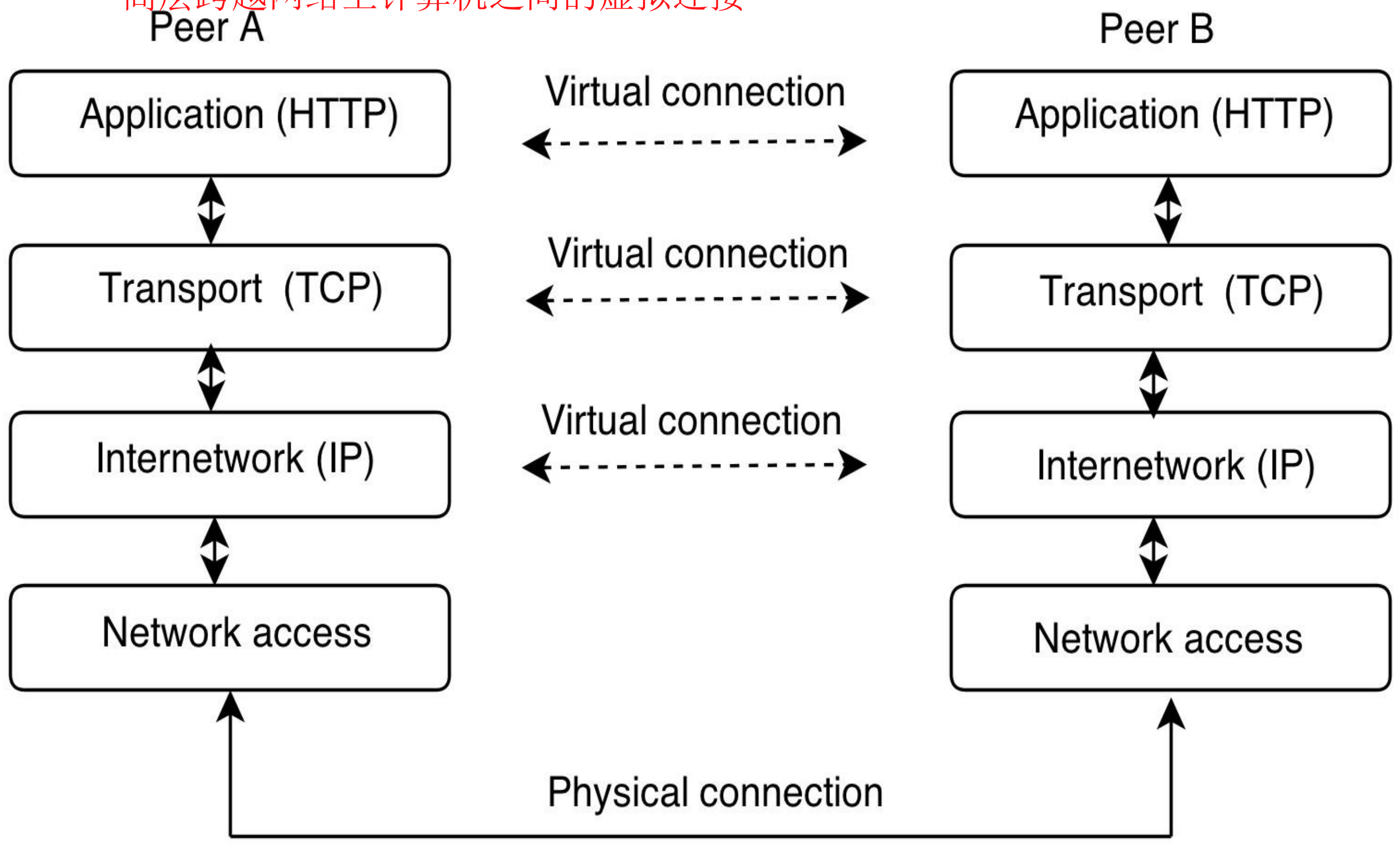


Figure: TCP/IP protocol stack

Virtual machines

中国科学技术大学
University of Science and Technology of China

- A virtual machine implements an instruction set for an imaginery machine
- Often virtual machines are the underlaying mechanism by which a programming language is executed
- E.g. Java virtual machine, different interpreters
- Specifies an interface between compiler and a real machine From conceptual point of view very similar to OS
- Improves portability

· 虚拟机为想象机实现指令集
· 通常，虚拟机是编程语言执行的底层机制
· 例如，Java虚拟机，不同的解释器
· 指定编译器和真实机器之间的接口，从概念上看非常类似于操作系统
· 提高可移植性

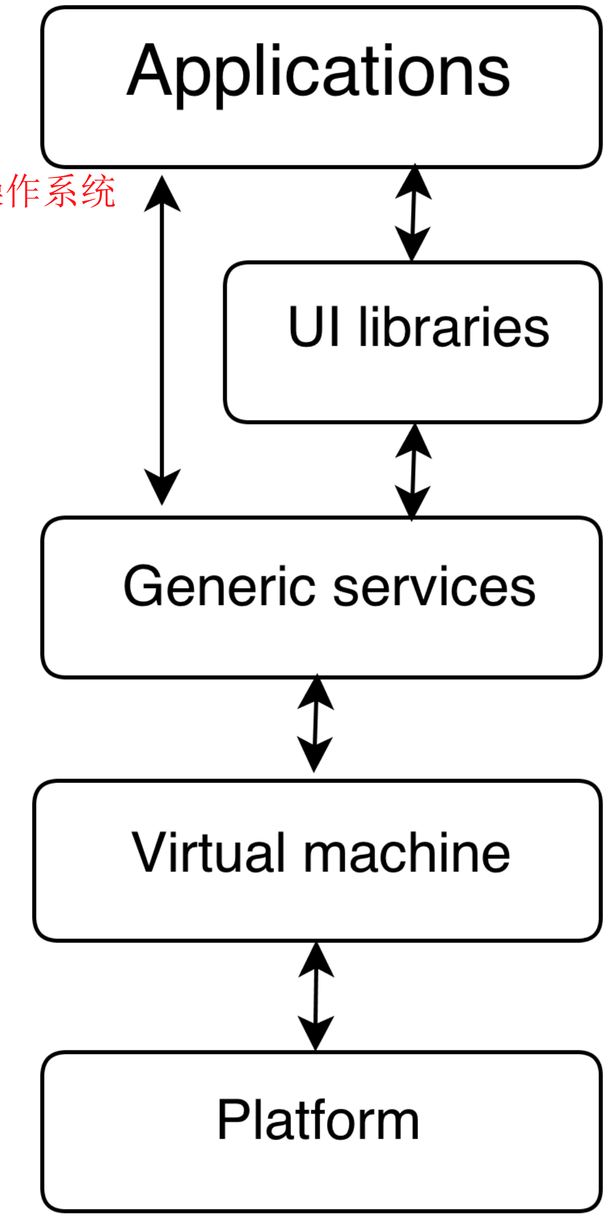
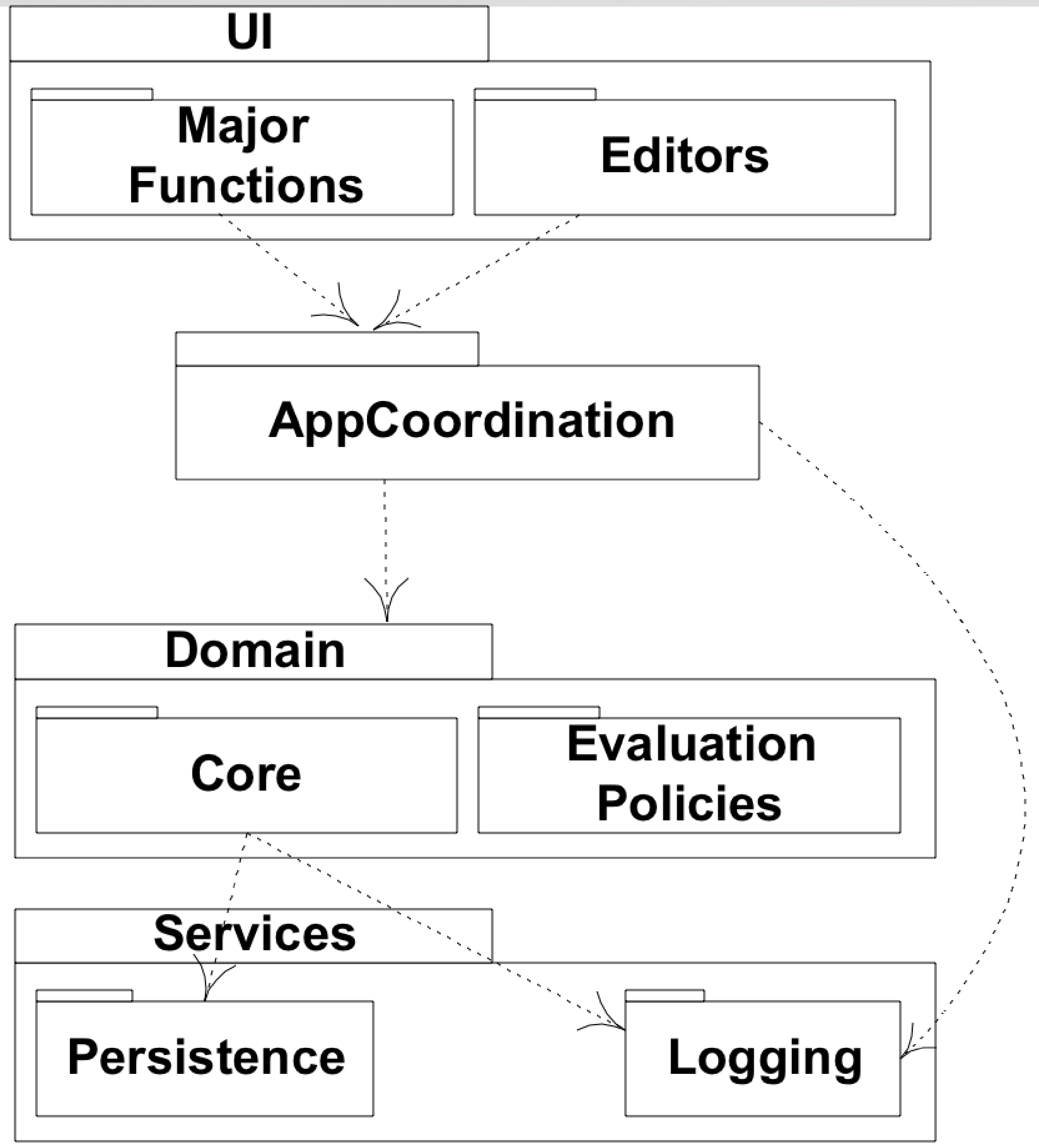
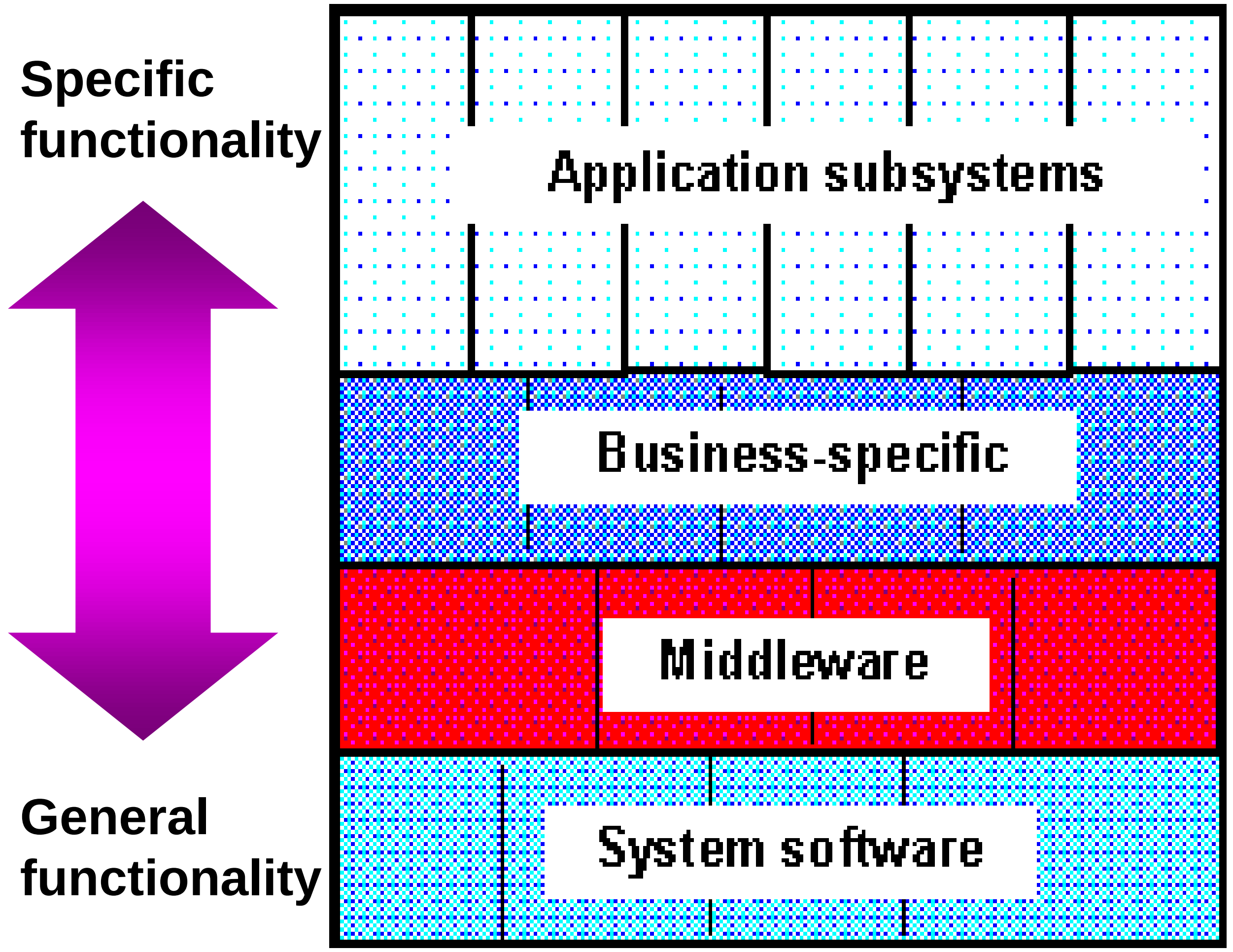


Figure: Virtual machine

Multi-Tier Layered Architecture - Example



Multi-Tier Layered Architecture - Example Layering Approach



Distinct application subsystem that make up an application - contains the value adding software developed by the organization.

Business specific - contains a number of reusable sybsystems specific to the type of business.

Middleware - offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System software - contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.

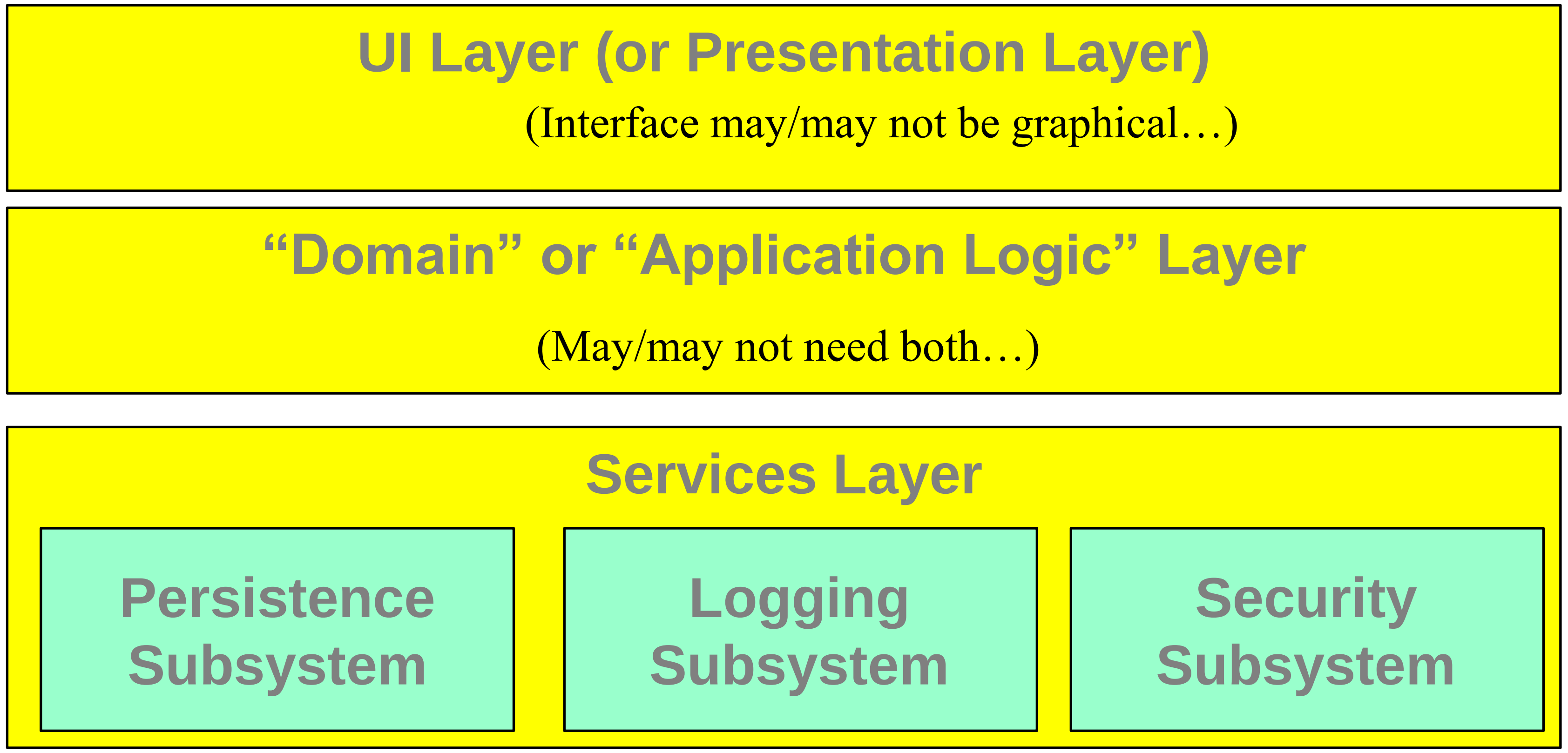
This is a very broad generalization. in practice, things will be considerably different and application dependent in many cases.

Note: this is also a very general view; may/may not include a GUI layer.

Multi-Tier Layered Architecture - Example

分离表示和应用程序逻辑，以及其他需要关注的领域。
考虑一下:不同的名字(在某些情况下)。可以看到主要思想!

- Separate presentation and application logic, and other areas of concern.
- Consider: Different names (in some cases). Can see the main idea!


UI Layer (or Presentation Layer)
(Interface may/may not be graphical…)
“Domain” or “Application Logic” Layer
(May/may not need both…)
Services Layer
Persistence Subsystem
Logging Subsystem
Security Subsystem

Layered architecture - advantages

- 每一层只与邻层耦合
  Each layer is only coupled to the adjacent layers
- Helps evolvability, as one can exchange a single layer and
  有助于可进化性，因为可以交换单个层并限制对邻层的修改
  limit modifications to neighbour layers
- Helps reusability, as layers can by used by multiple
  有助于可重用性，因为层可以被多个系统使用，特别是如果通信是标准化的
  systems Especially, if the communication is standardised
- In practice each layer is often maintained by dedicated
  在实践中，每一层通常由专门的开发团队维护
  development teams

Layered architecture - disadvantages

并不是所有的系统都适合分层组织
- Not all systems lend themselves to be organised in layers

抽象可能会对性能产生负面影响
- Abstraction might have negative effect of performance

例如，不可能找到特定的优化，很难找到正确的抽象级别
- E.g. specific optimisations not possible Hard to find right level of abstraction

在实践中，我们付出了很多努力来实现那些影响多层的特性
- In practice lot of effort to implement features which affect multiple layers

N-tier architectures

- The N-tier architecture is the modern client-server
  n层体系结构是现代的客户机-服务器体系结构
  architecture

  源于业务应用程序
- Originated with business applications
- Through the popularity of the Web today typically
  通过今天Web的普及，典型地与Web应用程序相关
  related with Web applications

  在概念上将架构分为表示层、应用程序层和数据存储层
- Conceptually separate architecture into presentation, application and data storage layers

2-tier architectures

中国科学技术大学
University of Science and Technology of China

客户端通常很丰富(ui+applogic+communication)
- Clients are typically rich (ui+applogic+communication)

服务器存储数据
- Servers store data

每个客户机运行一个完整的应用程序
- Each client runs a complete application

- Drawbacks: each client has to know how to 缺点:每个客户机必须知道如何与所有数据服务器通信 communicate with all data servers

- Scalability is compromised because client are tightly 由于客户端与服务器紧密耦合，可伸缩性受到影响 coupled with servers

2-tier architectures

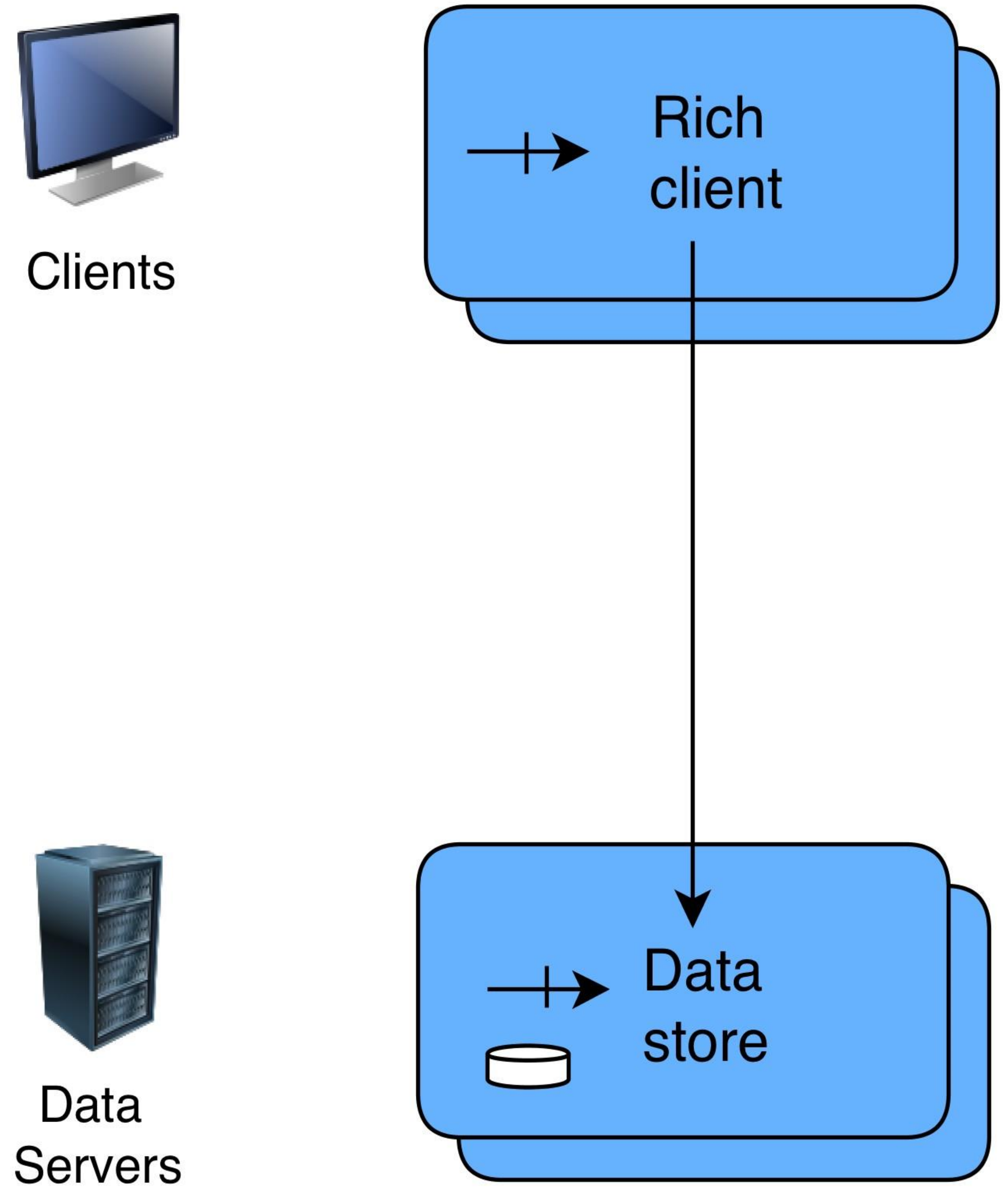


Figure: 2-tier architecture

3-tier architectures

中国科学技术大学
University of Science and Technology of China

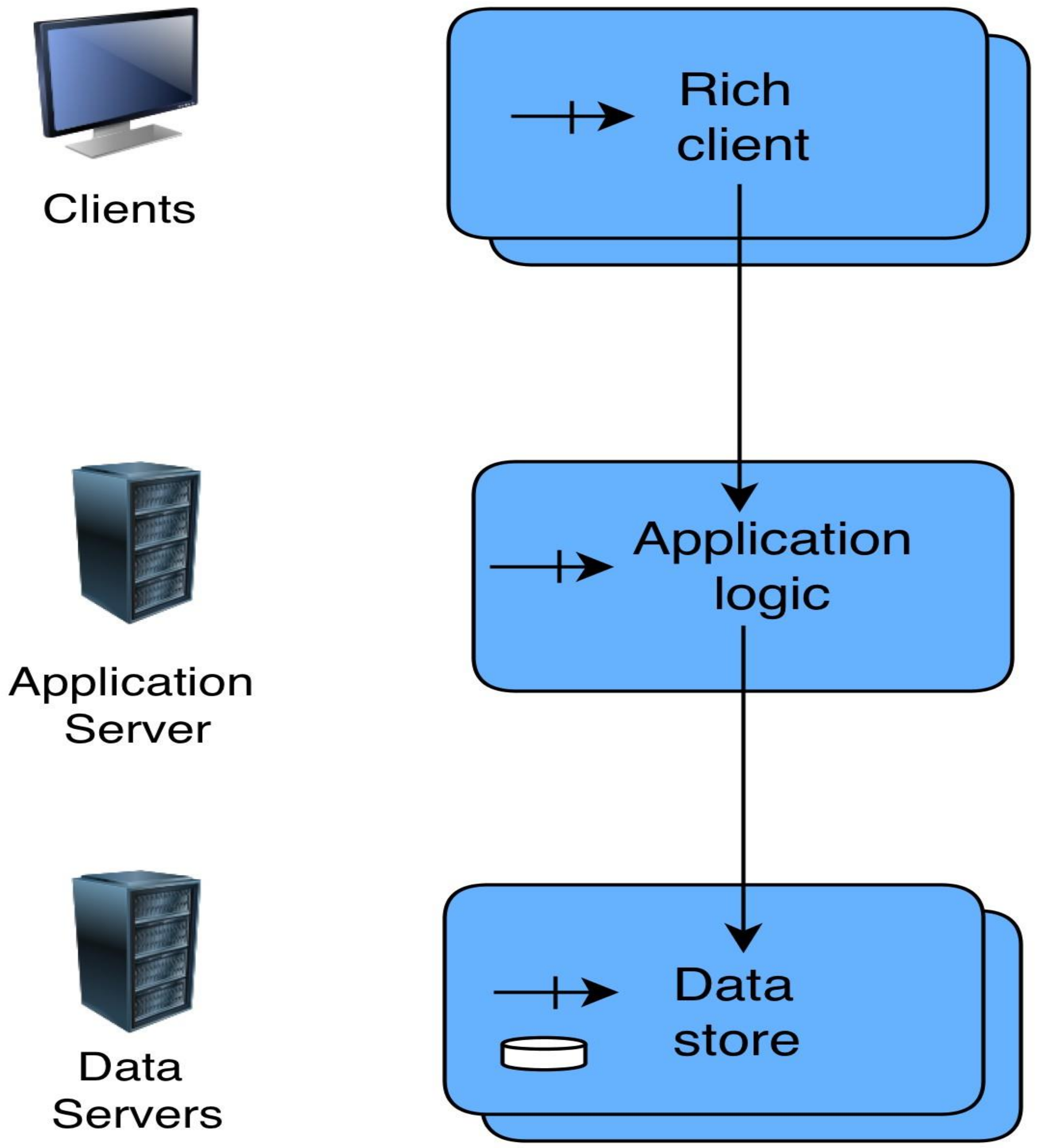
- Evolved from 2-tier architectures to solve their
  从2层架构演变而来，以解决其缺点
  drawbacks

  在客户端和数据服务器之间插入第三层
- A third tier is inserted between clients and data servers

  应用程序或业务逻辑层:中间层
- Application or business logic tier: middle tier
- Typically middle tier is on the server side (but recently
  中间层通常位于服务器端(但最近可能在服务器和客户端之间分离)
  might be split between the server and the client)

3-tier architectures

Figure: 3-tier architecture



3-tier architectures

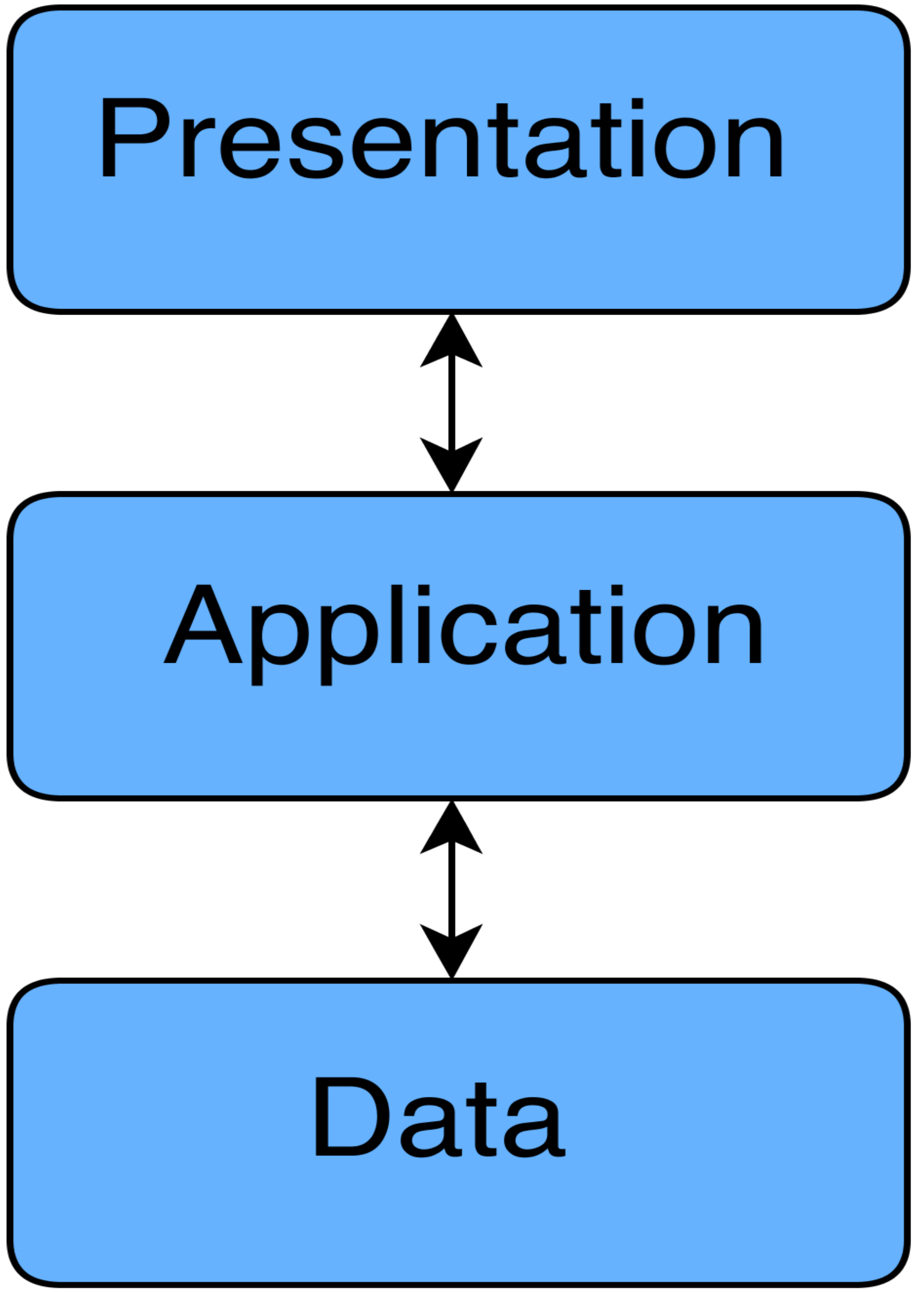
中国科学技术大学
University of Science and Technology of China

另一些优势：
-更容易维护客户端，因为中间层可以独立更新
-更好的隔离数据存储的细节：可扩展性，可配置性
-更佳的网络使用
-在中间层添加额外处理层的可能性
-干净地分离表示，应用逻辑和数据存储

- Another advantages
  - Easier client maintenance because the middle tier can be updated in isolation
  - Better isolation of specifics of the data storage: extensibility, configurability
  - Better network utilization
  - Possibility of adding additional processing layers in the middle layer
  - Cleanly separates presentation, application logic and data storage

3-tier architectures

Figure: Presentation, application, data



Rich clients

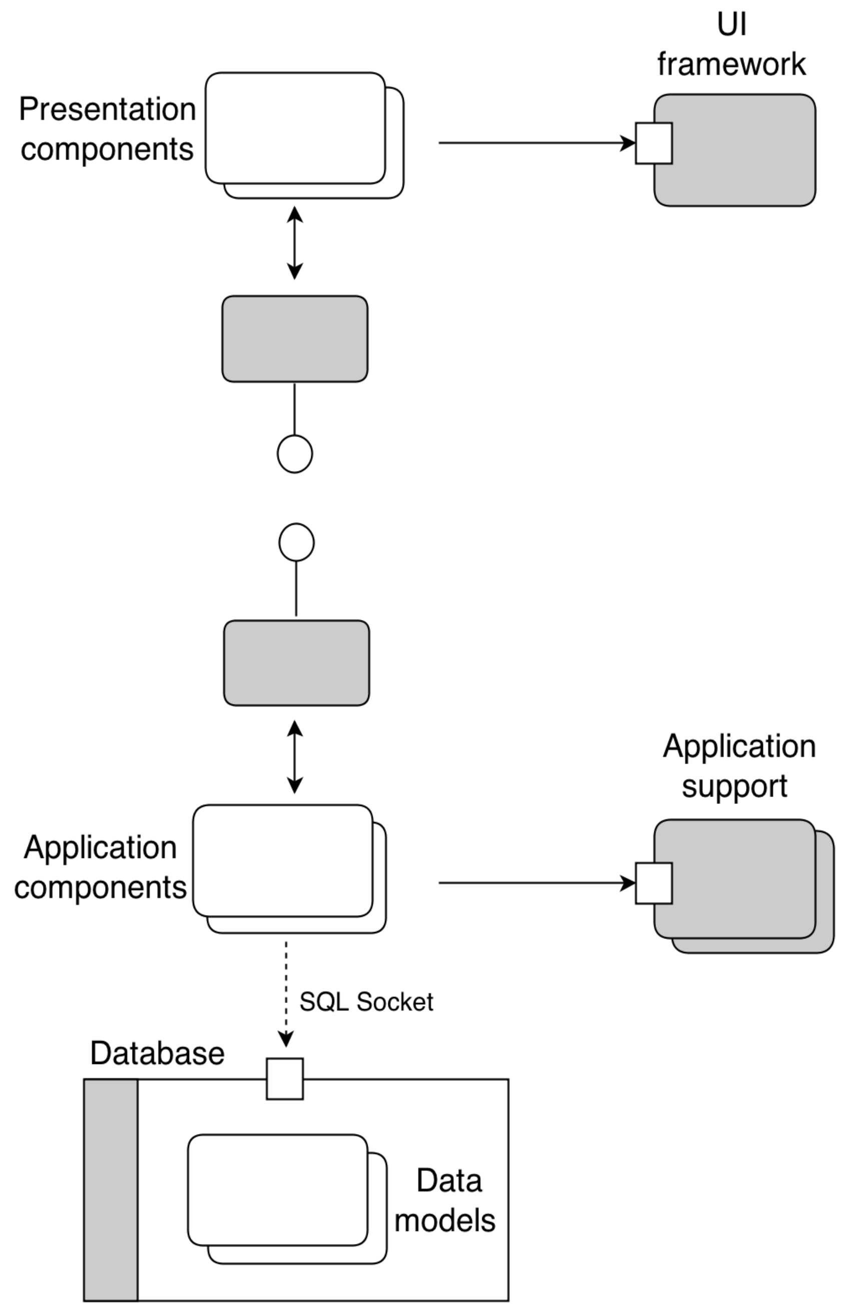
- A rich client contains full knowledge of the application
- It has access to the full suite of UI mechanisms on the platform

· 富客户端包含应用程序的全部知识
· 它可以访问平台上的全套UI机制
· 两种富客户端
–使用标准协议的标准应用程序的客户端(例如，电子邮件客户端)
–使用自定义协议的自定义应用程序的客户端

- Two types of rich clients
  – Clients for standard applications using a standard protocol (e.g. e-mail clients)
  – Clients for custom applications using a custom protocol

Rich clients

Figure: Rich client architecture



Thin clients

- Thin clients have little knowledge of the application

  · 瘦客户机对应用知之甚少
  · 完整的应用知识在服务器上
  · 最明显的是:网络客户端
  · 即使我们使用AJAX:它是由服务器提供的，即应用程序知识在服务器上

- The complete application knowledge is on the server
- Most notably: Web clients
- Even if we use AJAX: it is served by server, i.e. the application knowledge is on the server

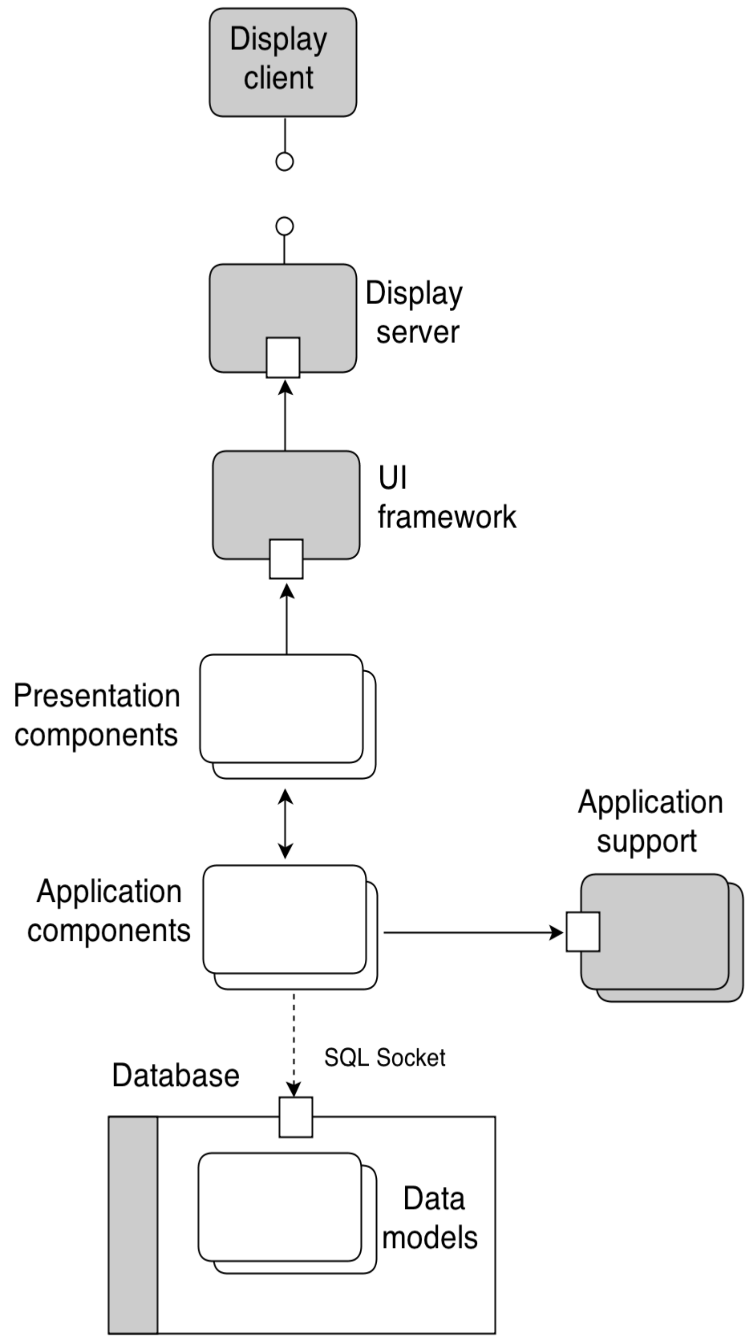
Thin clients

- Another form: a graphical thin client
- Only displays a graphical user interface
- In this architecture we have a so-called display server
- It resides on the server side but acts as a user-interface screen
- It transmits changes in the user-screen to the client
- X-server and X-windows on UNIX systems

· 另一种形式:图形化瘦客户端
· 仅显示图形用户界面
· 在这种架构中，我们有一个所谓的显示服务器
· 它驻留在服务器端，但作为用户界面屏幕
· 它将用户屏幕中的变化传递给客户端
· UNIX系统上的X-server和X-windows

Thin clients

Figure: Thin client architecture



Multi-layered Architecture And Design Principle

Satisfy Eleven Architectural Design Principles)

- 1. *Divide and conquer*: layers can be independently designed.
- 2. *Increase cohesion*: Well-designed layers have layer cohesion.
- 3. *Reduce coupling*: Well-designed lower layers **do not know about the higher layers** and the only connection between layers is through the API.
- 4. *Increase abstraction*: you **do not need to know** the details of how the lower layers are implemented.
- 5. *Increase reusability*: The lower layers can often be designed generically. (e.g. those that handle database access, persistency, etc.) (different databases….)

1. 分而治之:层可以独立设计。
2. 增加内聚性:设计良好的层具有层内聚性。
3. 减少耦合:设计良好的低层不了解高层，层之间的唯一连接是通过API。
4. 增加抽象:您不需要知道底层是如何实现的细节。
5. 增加可重用性:底层通常可以进行通用设计。(例如,那些处理数据库访问、持久性等的数据库)(不同的数据库…)

Multi-layered Architecture And Design Principle

Satisfy Eleven Architectural Design Principles)

- 6. *Increase reuse*: You can often reuse layers built by others that provide the services you need. (think: Domain Layer) 增加重用:您通常可以重用提供所需服务的其他人构建的层。(思考:领域层)
- 7. Increase flexibility: you can add new facilities built on lower-level services, or replace higher-level layers. 增加灵活性:您可以添加构建在底层服务上的新设施，或者替换较高层的层
- 8. *Anticipate obsolescence*: By isolating components in separate layers, the system becomes more resistant to obsolescence. 提前淘汰:通过在单独的层中隔离组件，系统变得更能抵抗淘汰
- 可移植性设计:所有相关的设施都可以隔离在一个较低的层中
  9. *Design for portability*: All the dependent facilities can be isolated in one of the lower layers.
- As we know, some things tend to change over time; more than some other aspects of an application. 我们知道，有些事情会随着时间而改变;比应用程序的其他方面更重要
- 10. *Design for testability*: Layers can be tested independently through the interfaces exercising layer responsibilities. 可测试性设计:可以通过执行层职责的接口独立地测试层
- 11. *Design defensively*: The APIs of layers are natural places to build in rigorous assertion-checking. 防御性设计:层的APIs是构建严格断言检查的自然场所
- **Note that this model appears to satisfy all eleven design principles!** 注意，这个模型似乎满足所有十一个设计原则

Notification Architectures

松散耦合组件的架构

Architectures for loosely coupled components.

Notification architectures

中国科学技术大学
University of Science and Technology of China

- Architectures where information and activity is 通过通知机制传播信息和活动的体系结构 propagated by a notification mechanism
- When there are complex communication patterns 存在在设计时还不知道的复杂通信模式 that are not known at design time
- Basically, components interested in events register 基本上，感兴趣的组件会注册它们感兴趣的事件，当事件发生时，感兴趣的组件会得到通知 their interest When an event occurs the interested components are notified
- Well-known example: event handling in GUIs 众所周知的例子:gui中的事件处理

Notification architectures

中国科学技术大学
University of Science and Technology of China

- 也称为:发布-订阅
  Also called: publish-and-subscribe
- 其他术语:监听和回调
  Other terminology: listeners and callbacks
- Similar to the observer pattern from the field of
  类似于设计模式领域的观察者模式
  design pattern
- 类似于blackboard数据中心架构
  Similar to blackboard data-centric architectures
- Notification architectures are more general because
  通知体系结构更为通用，因为任何类型的事件都可能发生，而不仅仅是与数据相关的事件
  any kind of events might occur not only data-related events

Publish-Subscribe

组件:发布者、订阅者、管理发布的代理
- Components: Publishers, subscribers, proxies for managing distribution

连接器:通常需要一个网络协议
- Connectors: Typically a network protocol is required.

基于内容的订阅需要复杂的连接器
- Content-based subscription requires sophisticated connectors.

数据元素:订阅、通知、发布信息
- Data Elements: Subscriptions, notifications, published information
- Topology: Subscribers connect to publishers either directly or may
  拓扑:订阅者可以直接连接到发布者，也可以通过网络协议从中介接收通知
  receive notifications via a network protocol from intermediaries
- Qualities yielded Highly efficient one-way dissemination of information
  质量产生了高效的信息单向传播，组件的耦合非常低
  with very low-coupling of components

Notification architectures

- 在执行级别有两个变体
  At execution level two variants
- Components interested in events register directly
  对事件感兴趣的组件直接注册到生成这些事件的进程
  with the process that generates these events
- Components register with a dedicated notification
  组件注册到专用通知组件
  component
- 这两个变体中的通信样式都是回调
  The communication style in both variants is callback

Notification architectures

中国科学技术大学 University of Science and Technology of China

额外的变体
- **Additional variants**
- When an event occurs the notifying component may send the
当事件发生时，通知组件可以将事件发送到所有注册的组件(广播)
event to all registered components (broadcast)
- Another possibility is to send the event only to interested
另一种可能是只将事件发送给感兴趣的组件
components

沟通风格的变化
- Variations in communication style

通知可能包含所有相关数据(推送风格)
- Notification might include all the relevant data (push style)
- Notification just signals that something happened: the registered
通知只是表示发生了什么事情:注册的组件提取相关数据
component pulls the relevant data

Notification architectures

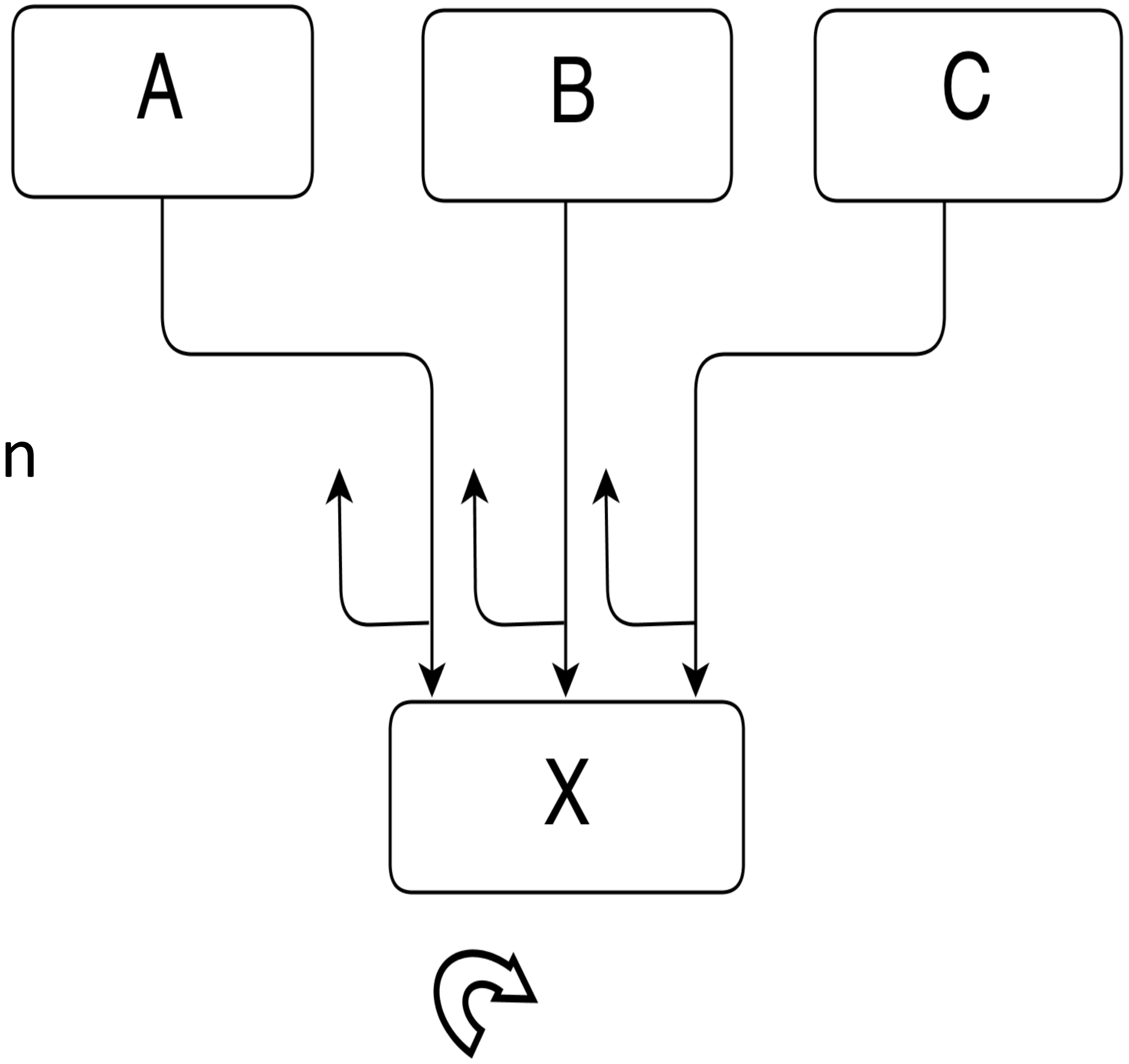


Figure: Notification architecture

Notification architectures

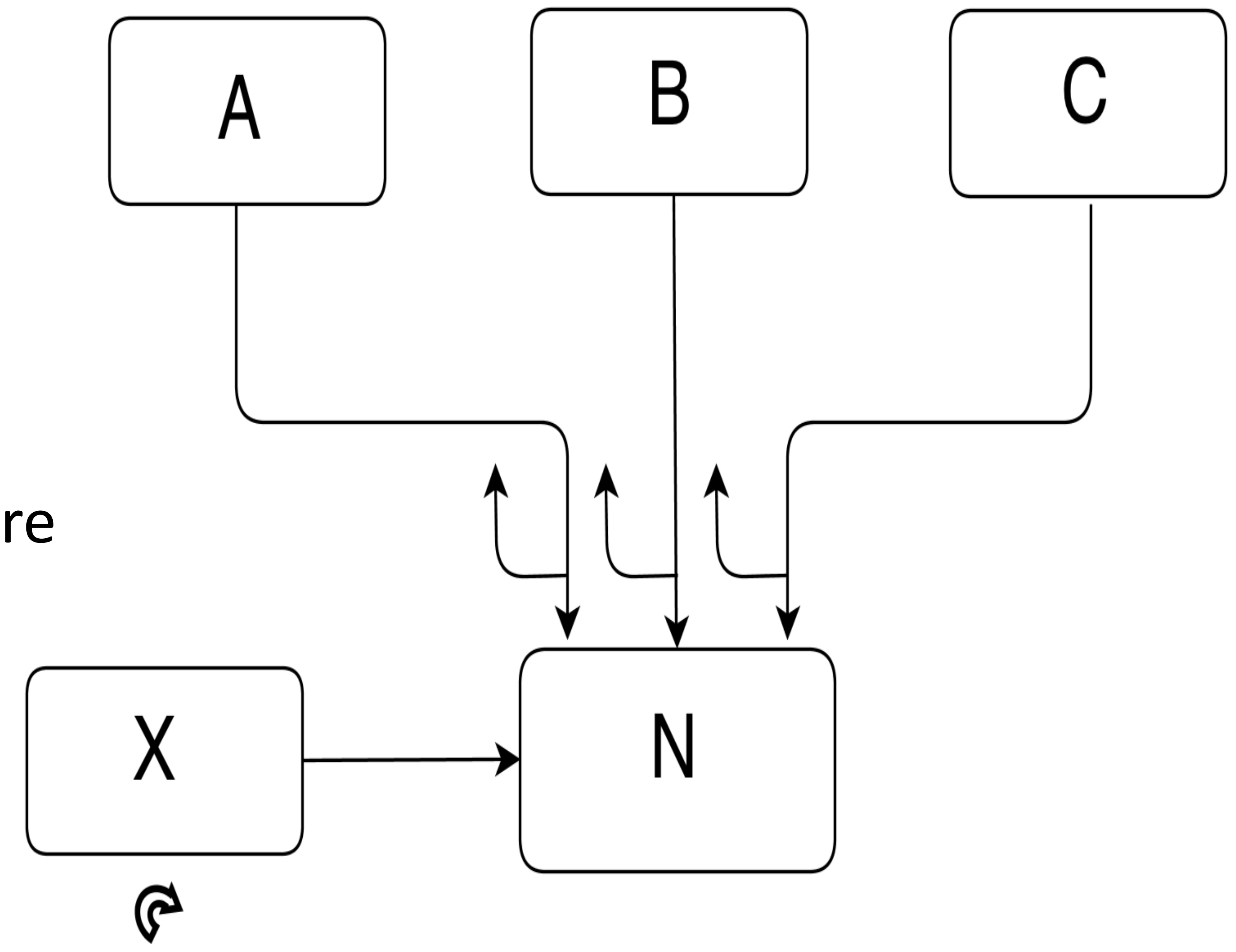


Figure: Notification architecture

Notification architectures

- Similar to notification architecture: message queues
  类似于通知体系结构:消息队列
- Used to control process flow
  用于控制进程流
- Two types: publish/subscribe and point-to-point
  有两种类型:发布/订阅和点对点
- Publish/subscribe also known as topics
  发布/订阅也称为主题

Notification architectures

中国科学技术大学
University of Science and Technology of China

通知风格架构的示例

- **Examples of notification style architectures**
  - Triggers in databases
  - Consistency checks
  - Spell checking in editors
  - Separation of presentation from persistence of data

- 数据库中的触发器
- 一致性检查
- 编辑器中的检查拼写
- 数据的显示与持久化分离

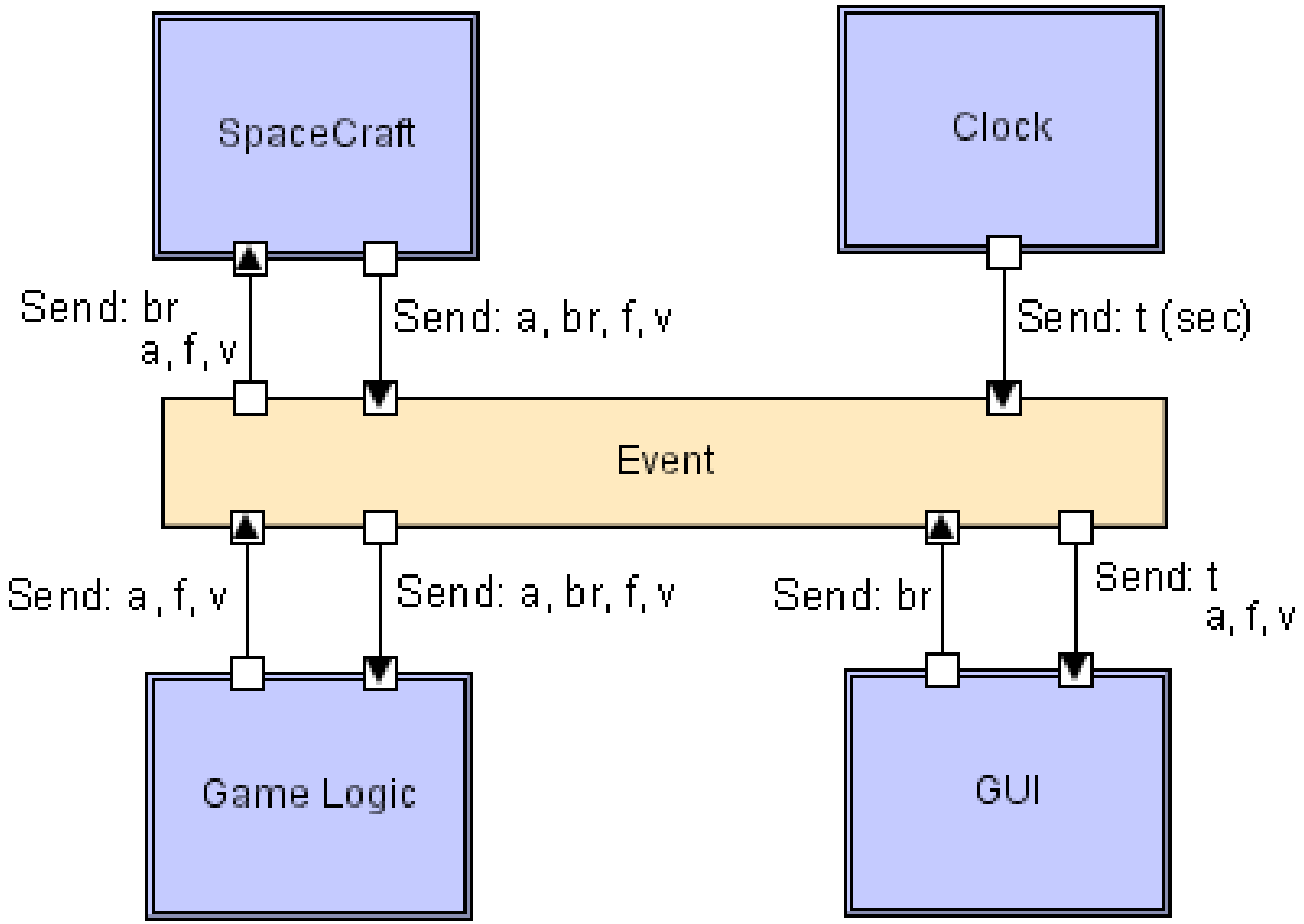
Event-Based Style

中国科学技术大学
University of Science and Technology of China

- Independent components asynchronously emit and receive events
  独立组件异步发送和接收通过事件总线通信的事件
  communicated over event buses
- 组件:独立的、并发的事件生成器和/或使用者
  Components: Independent, concurrent event generators and/ or consumers
- 连接器:事件总线(至少一个)
  Connectors: Event buses (at least one)
- 数据元素:事件——作为头等实体通过事件总线发送的数据
  Data Elements: Events – data sent as a first-class entity over the event bus
- Topology: Components communicate with the event buses, not directly to
  拓扑:组件与事件总线通信，而不是直接彼此通信
  each other.
- Variants: Component communication with the event bus may either be push
  变体:组件与事件总线的通信可以是基于推或基于拉的
  or pull based.
- 高度可伸缩，易于发展，对高度分布式应用程序有效
  Highly scalable, easy to evolve, effective for highly distributed applications

SOA：把系统按照实际业务，拆分成刚刚好大小的、合适的、独立部署的模块，每个模块之间相互独立
缺点：容易形成网状结构，不好管理（引入数据总线，系统便于管理）

Event-based LL



Notification architectures - advantages

- 松散耦合
  Loose coupling
- Notification receivers can easily be changed
  可以很容易地更改和添加通知接收者
  and added
- 帮助可发展性
  Helps evolvability

Notification architectures - disadvantages

- 无法控制进程流
  No control over process flow
- Due to asynchronous nature, might not be deterministic
  由于异步特性，可能不是确定性的(不利于可测试性)
  (bad for testability)
- Unclear which data should be passed with the
  不清楚应随通知传递哪些数据
  notification
- If too little data is passed, the receiver might need to
  如果传递的数据太少，接收方可能需要获取它(增加耦合)
  fetch it (increases coupling)
- Performance issues due to update storm Even worse,
  性能问题由于更新风暴甚至更糟，可能会有无尽的循环
  there might be endless loops