架构设计

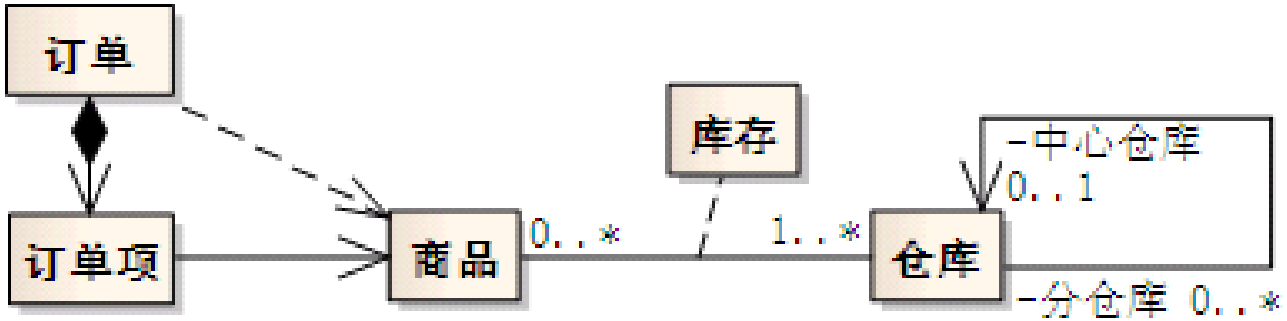
Chapter 6

北京航空航天大學
SCHOOL OF SOFTWARE BEIHANG UNIVERSITY 软件学院

单选题 1分

设置

针对下面的类图，论述错误的是：



A 订单项不能脱离订单独立存在

B 一件商品至少存储在1个仓库中

C 一个中心仓库可能没有也可能有多个分仓库

D 仓库类的自反关联意味着每个仓库对象之间存在着自反链接

提交

Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Architecting a dog house

Can be built by one person
Requires
- Minimal modeling
- Simple process
- Simple tools

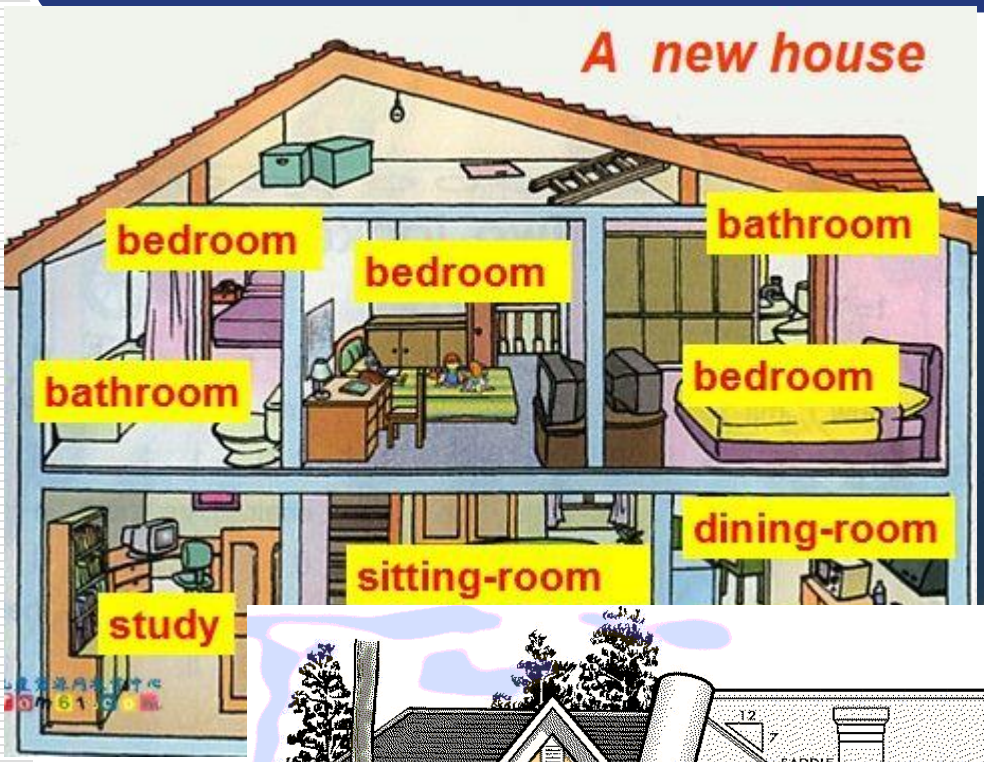
Architecting a house

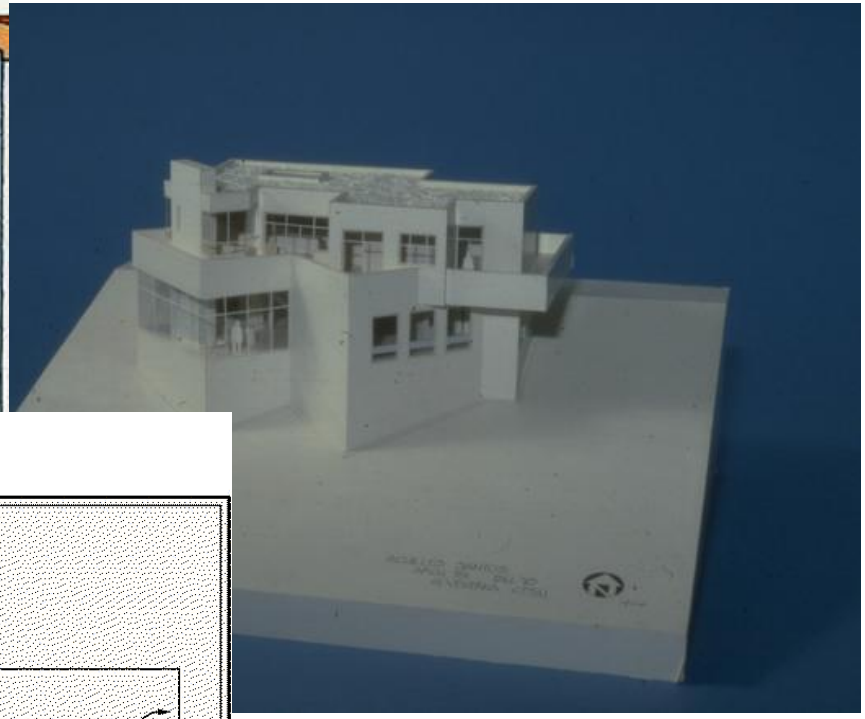
Built most efficiently and timely by a team

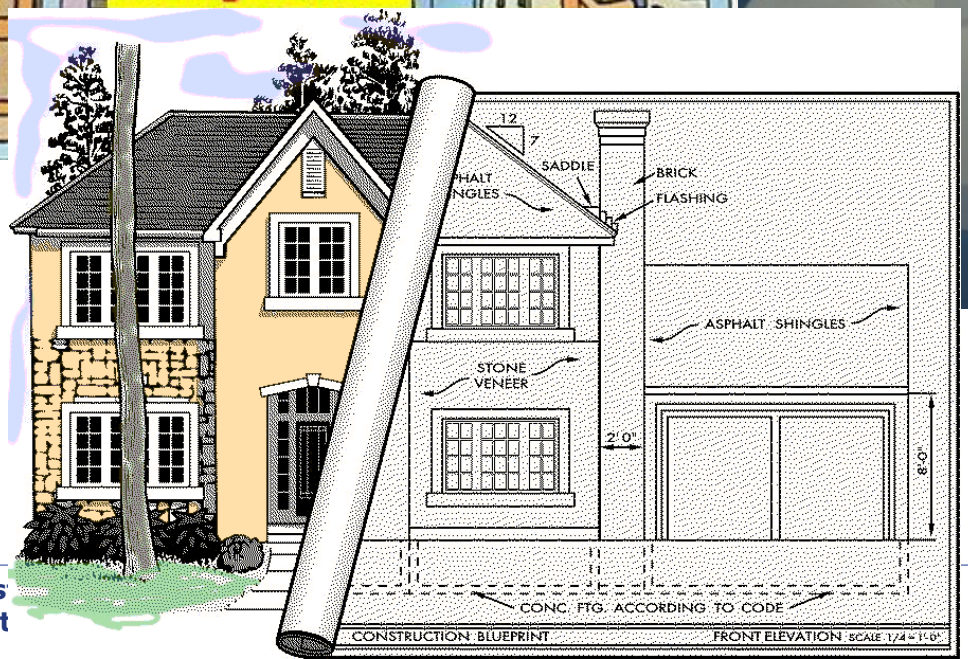
Requires

- Modeling
- Well-defined process
- Power tools

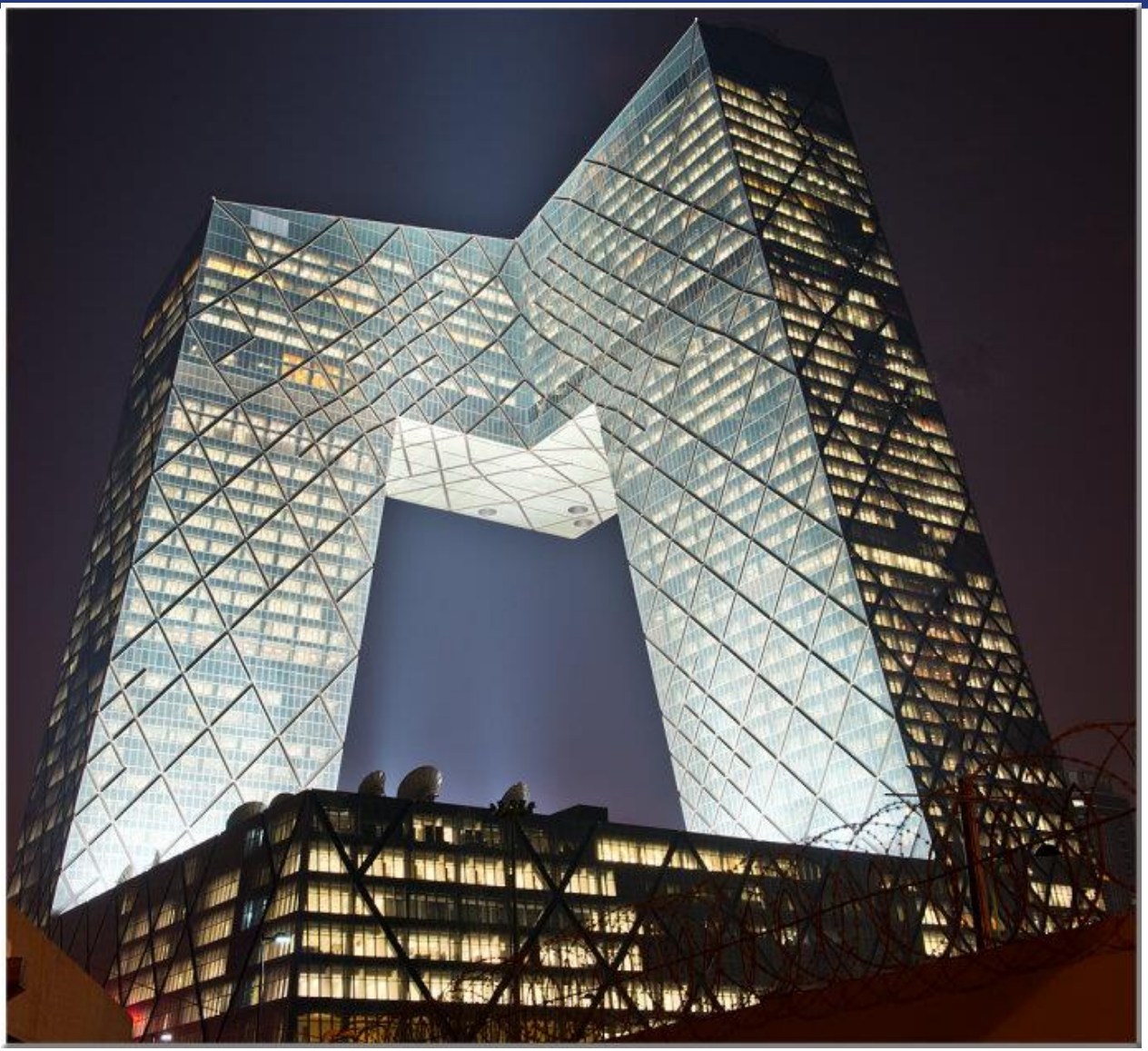
Modeling a house







Architecting a high rise



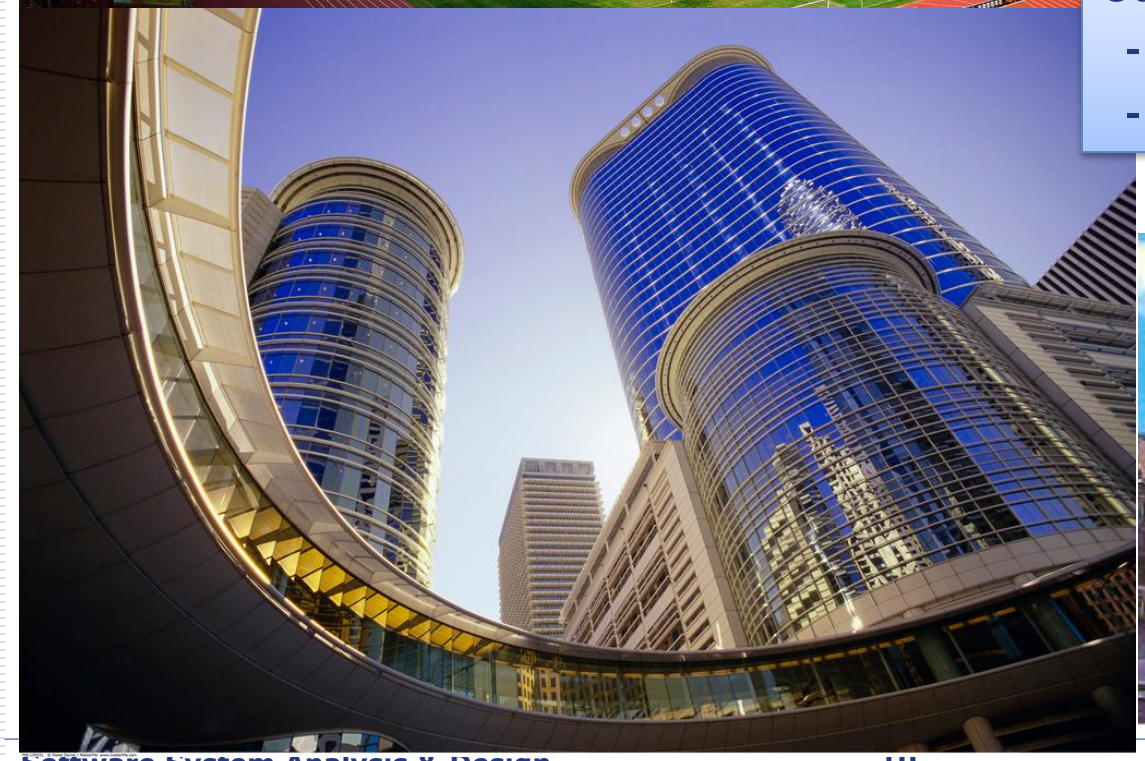
Early architecture

Progress
- Limited knowledge of theory

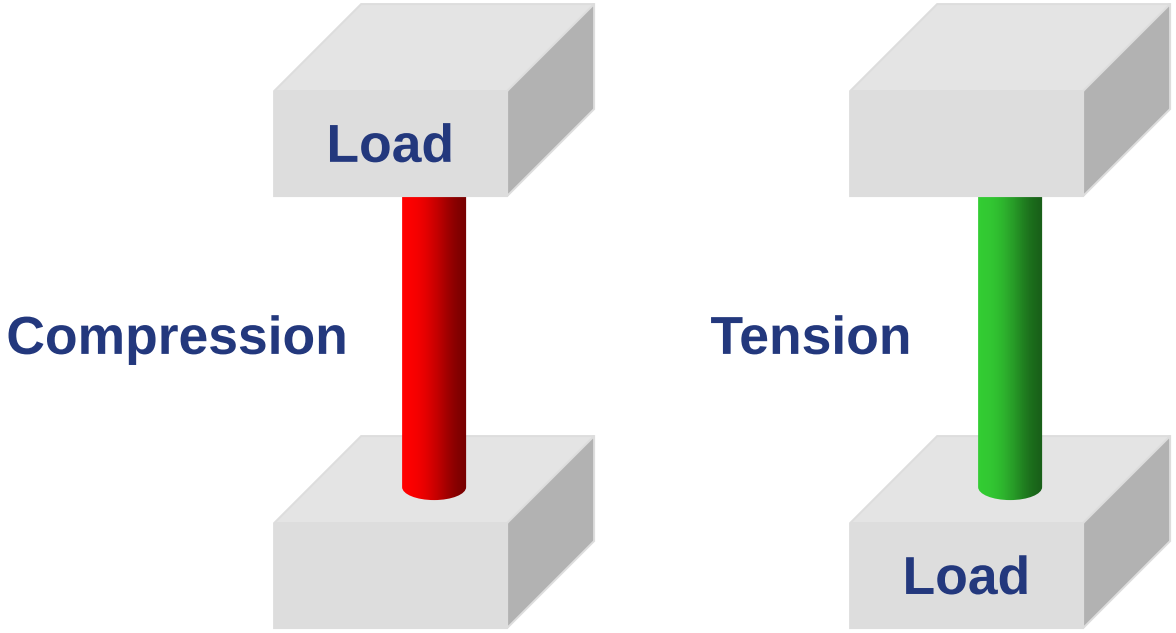
Modern architecture

Progress
- Advances in materials
- Advances in analysis

Scale
- 5 times the span of the Pantheon
- 3 times the height of Cheops



Forces in civil architecture



Kinds of loads

- Dead loads
- Live loads
- Dynamic loads

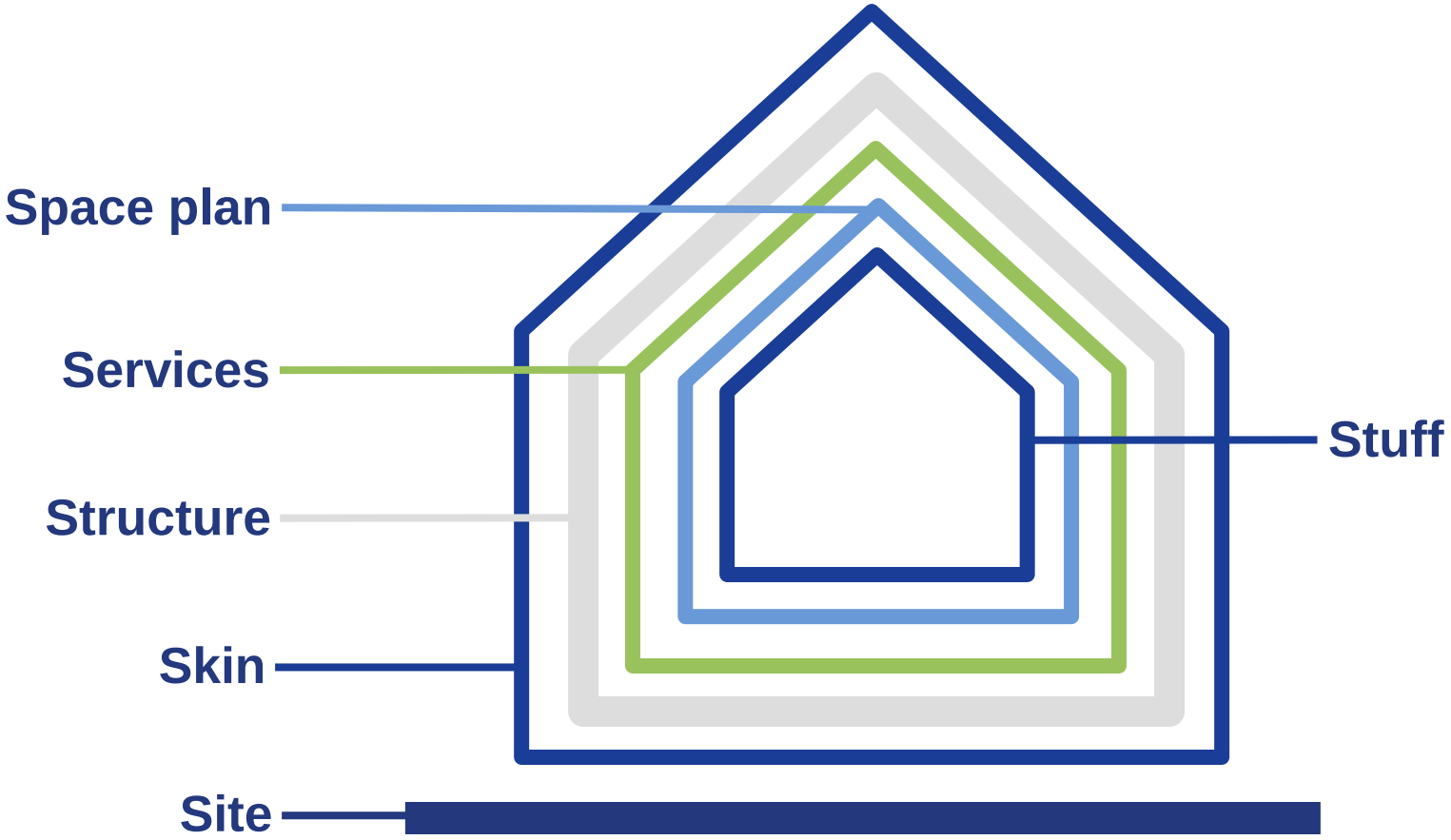
Avoiding failure

- Safety factors
- Redundancy
- Equilibrium

Any time you depart from established practice, make ten times the effort, ten times the investigation. Especially on a very large project.

- LeMessuier

Shearing layers of change

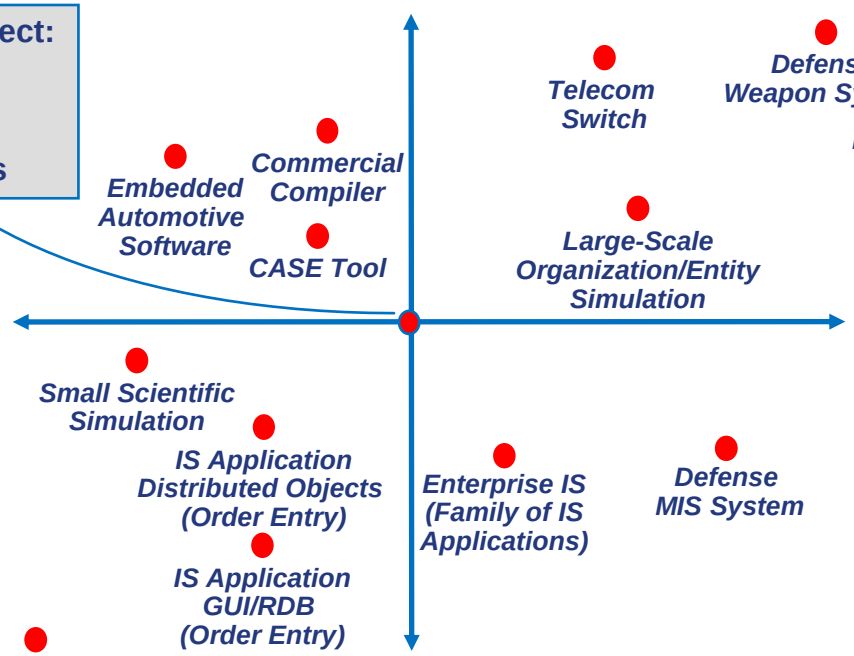


Dimensions of software complexity

Higher technical complexity

- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance

An average software project:
- 5-10 people
- 10-15 month duration
- 3-5 external interfaces
- Some unknowns & risks



Lower management complexity

- Small scale
- Informal
- Single stakeholder
- "Products"

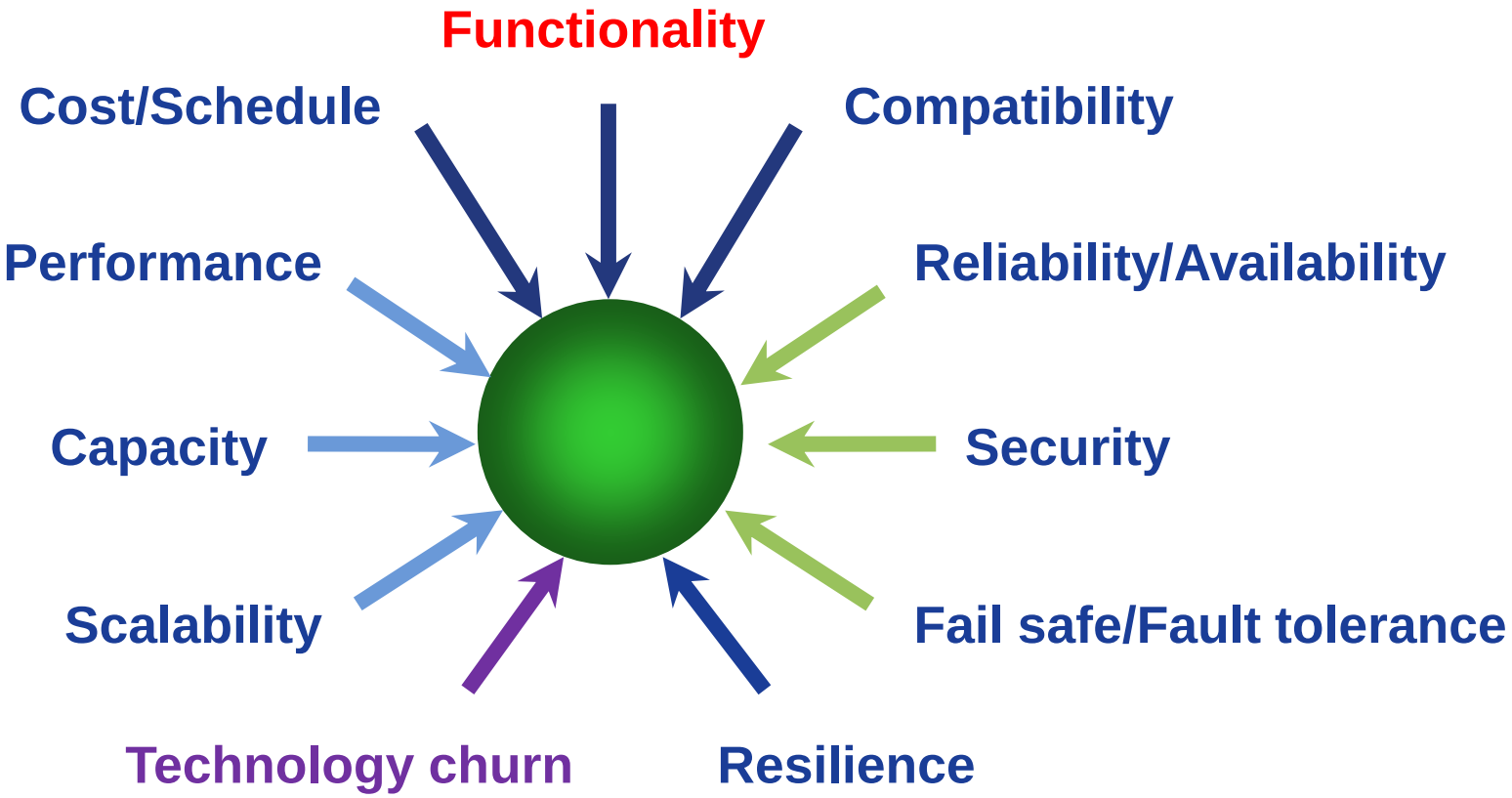
Higher management complexity

- Large scale
- Contractual
- Many stake holders
- "Projects"

Lower technical complexity

- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Forces In Software



Software Architecture – Definition

- Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability.
- It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.
  - *From Microsoft Application Architecture Guide, 2nd Edition*

Software Architecture – Definition

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.
  - *From Software Architecture in Practice (2nd edition)*

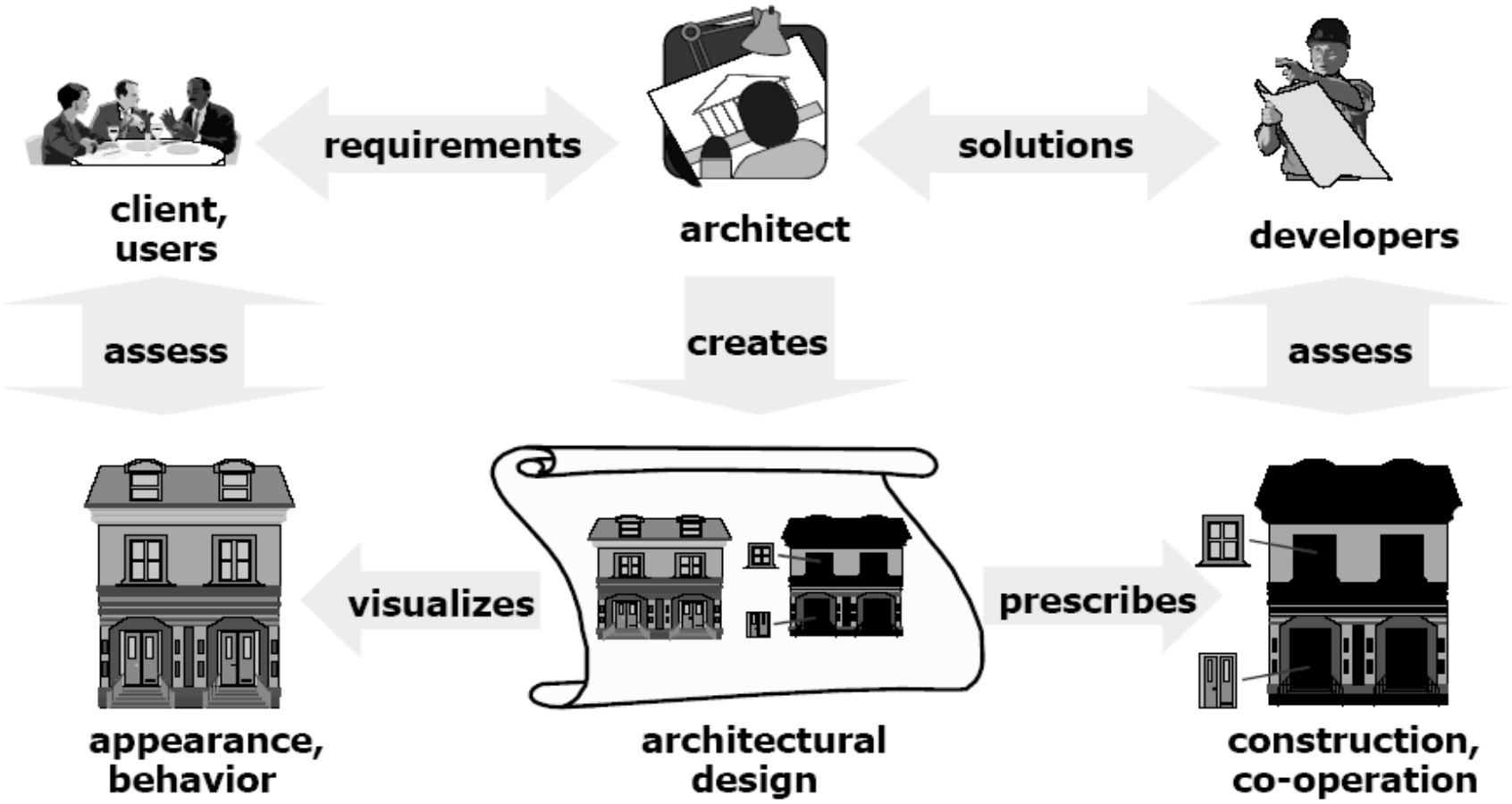
Architecture is Important (1)

- Like any other complex structure, software must be built on a solid foundation
  - Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk
  - Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements.

Architecture is Important (2)

- The risks exposed by poor architecture include
  - software that is unstable
  - is unable to support existing or future business requirements
  - or is difficult to deploy or manage in a production environment.

Architecture is Important (3)



The Architectural Landscape

- The key forces that are shaping architectural decisions are driven by user demand, as well as by business demand for faster results, better support for varying work styles and workflows, and improved adaptability of software design
  - User empowerment
  - Market maturity
  - Flexible design
  - Future trends

The Architectural Landscape (1)

- User empowerment: A design that supports user empowerment is flexible, configurable, and focused on the user experience
  - Design your application with appropriate levels of user personalization and options in mind.
  - Allow the user to define how they interact with your application instead of dictating to them, but do not overload them with unnecessary options and settings that can lead to confusion.
  - Understand the key scenarios and make them as simple as possible; make it easy to find information and use the application.

The Architectural Landscape (2)

- Market maturity: Take advantage of market maturity by taking advantage of existing platform and technology options
  - Build on higher level application frameworks where it makes sense, so that you can focus on what is uniquely valuable in your application rather than recreating something that already exists and can be reused
  - Use patterns that provide rich sources of proven solutions for common problems.

The Architectural Landscape (3)

- Flexible design: Increasingly, flexible designs take advantage of loose coupling to allow reuse and to improve maintainability
  - Pluggable designs allow you to provide post-deployment extensibility
  - You can also take advantage of service orientation techniques such as SOA to provide interoperability with other systems.

The Architectural Landscape (4)

- Future trends: When building your architecture, understand the future trends that might affect your design after deployment.
  - For example, consider trends in rich UI and media, composition models such as mashups, increasing network bandwidth and availability, increasing use of mobile devices, continued improvement in hardware performance, interest in community and personal publishing models, the rise of cloud-based computing, and remote operation

Key Design Principles

- Separate the areas of concern
- Keep design patterns consistent within each layer
- Do not duplicate functionality within an application
- Prefer composition to inheritance
- Establish a coding style and naming convention for development
- Maintain system quality using automated QA techniques during development
- Consider the operation of your application

Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Software Architecture Pattern

- Software Architecture Patterns help define the basic characteristics and behavior of a software system
  - Does the architecture scale?
  - What are the performance characteristics of the application?
  - How easily does the application respond to change?
  - What are the deployment characteristics of the application?
  - How responsive is the architecture?

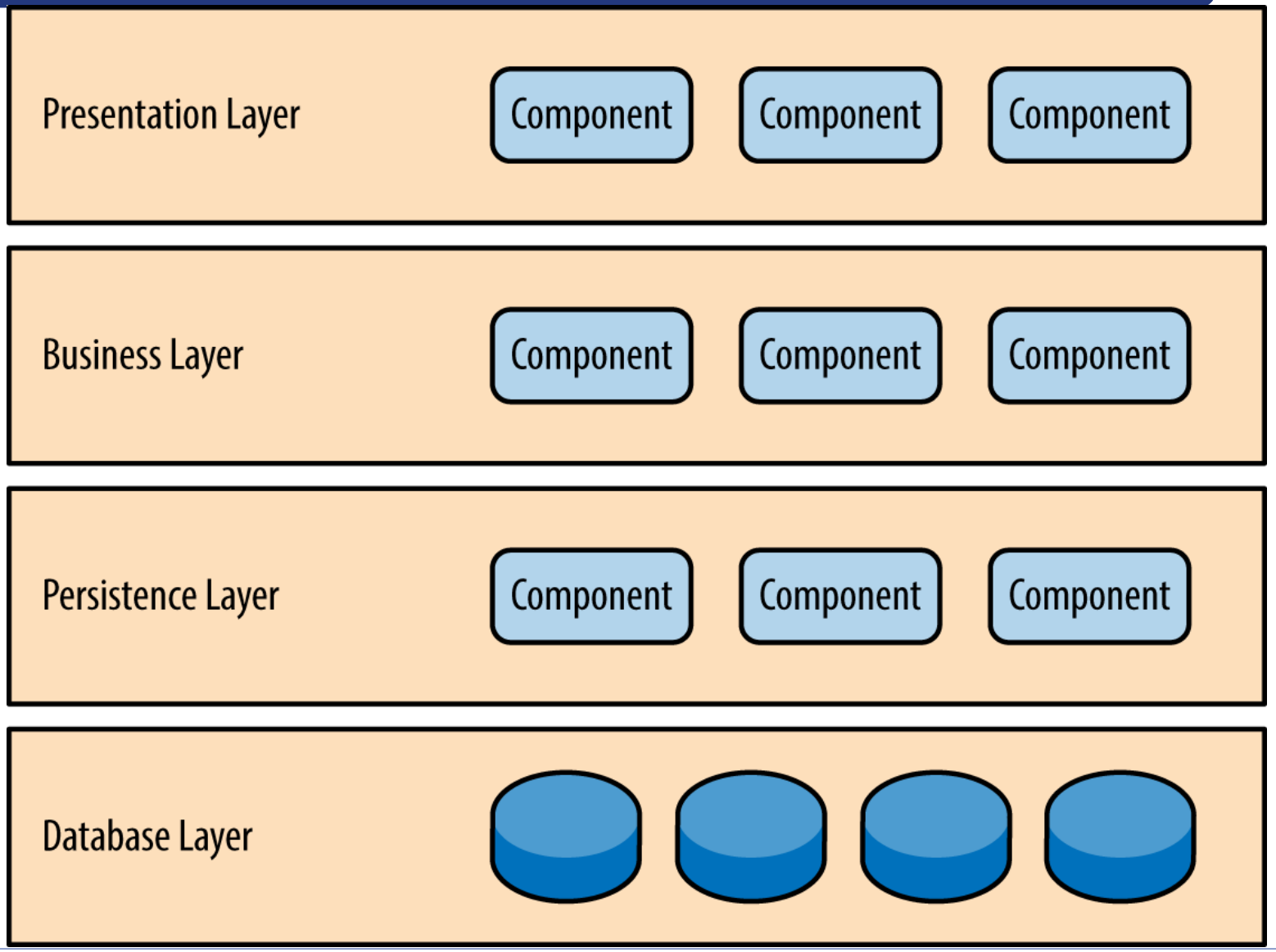
Typical Architecture Pattern

- Layered Architecture (分层)
- Event-Driven Architecture (事件驱动)
- Microkernel Architecture (微内核)
- Microservices Architecture Pattern (微服务)
- Space/Cloud-Based Architecture (云)

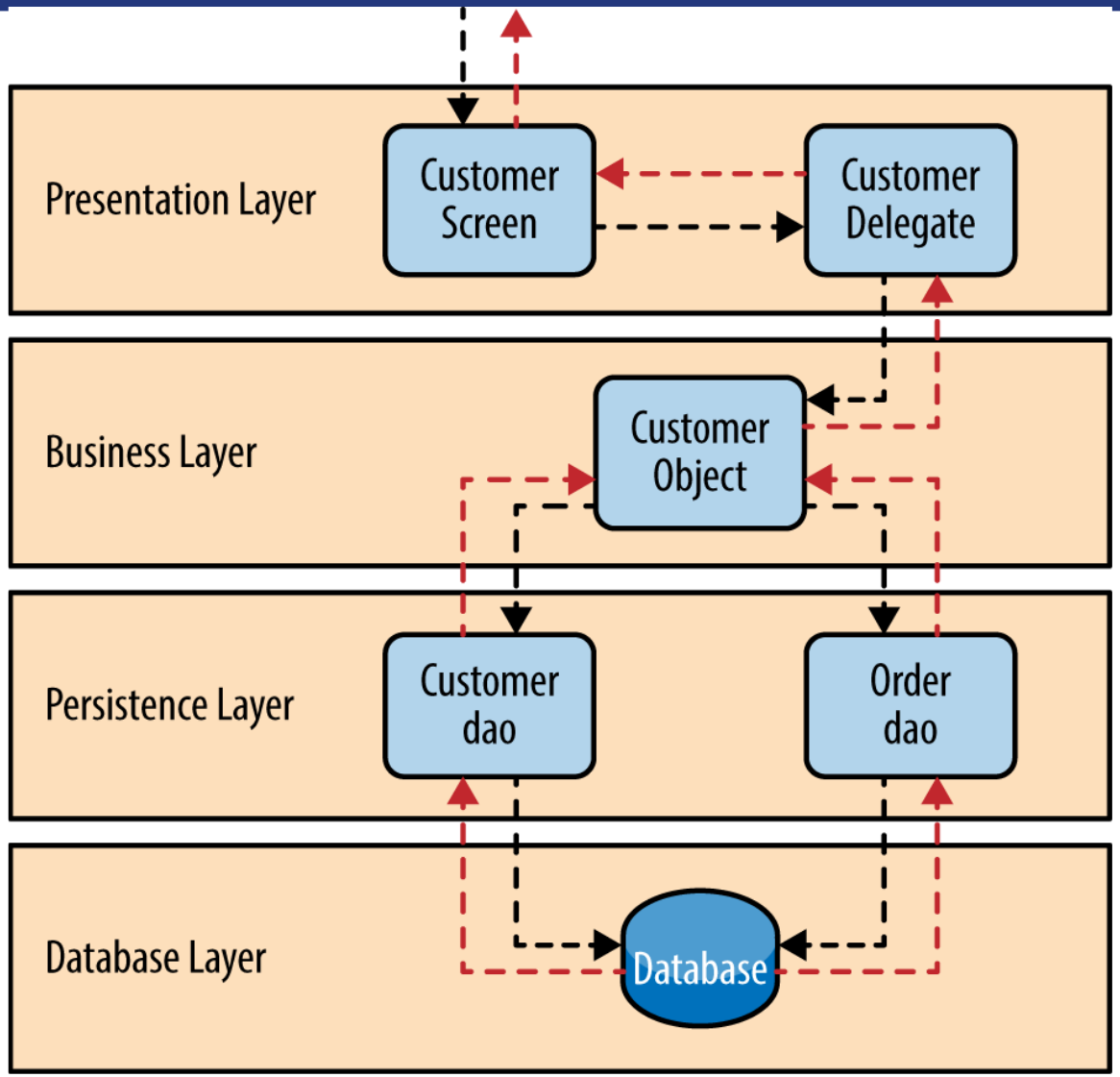
Layered Architecture

- The most common architecture pattern is the layered architecture pattern
  - known as the n-tier architecture pattern.
  - This pattern is the de facto standard for most Java EE applications
- The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.

Pattern Description



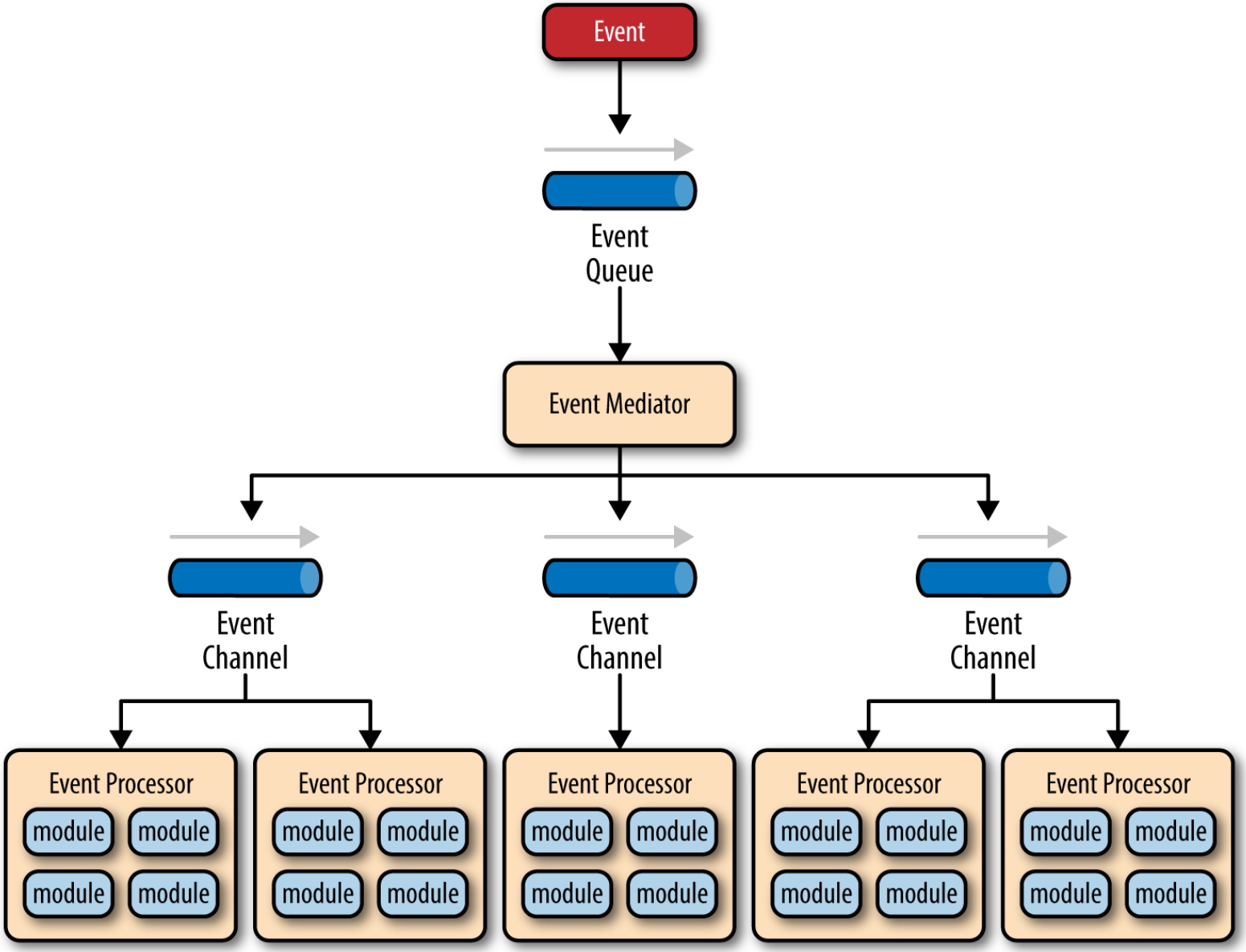
Example


Presentation Layer
Customer Screen
Customer Delegate
Business Layer
Customer Object
Persistence Layer
Customer dao
Order dao
Database Layer
Database

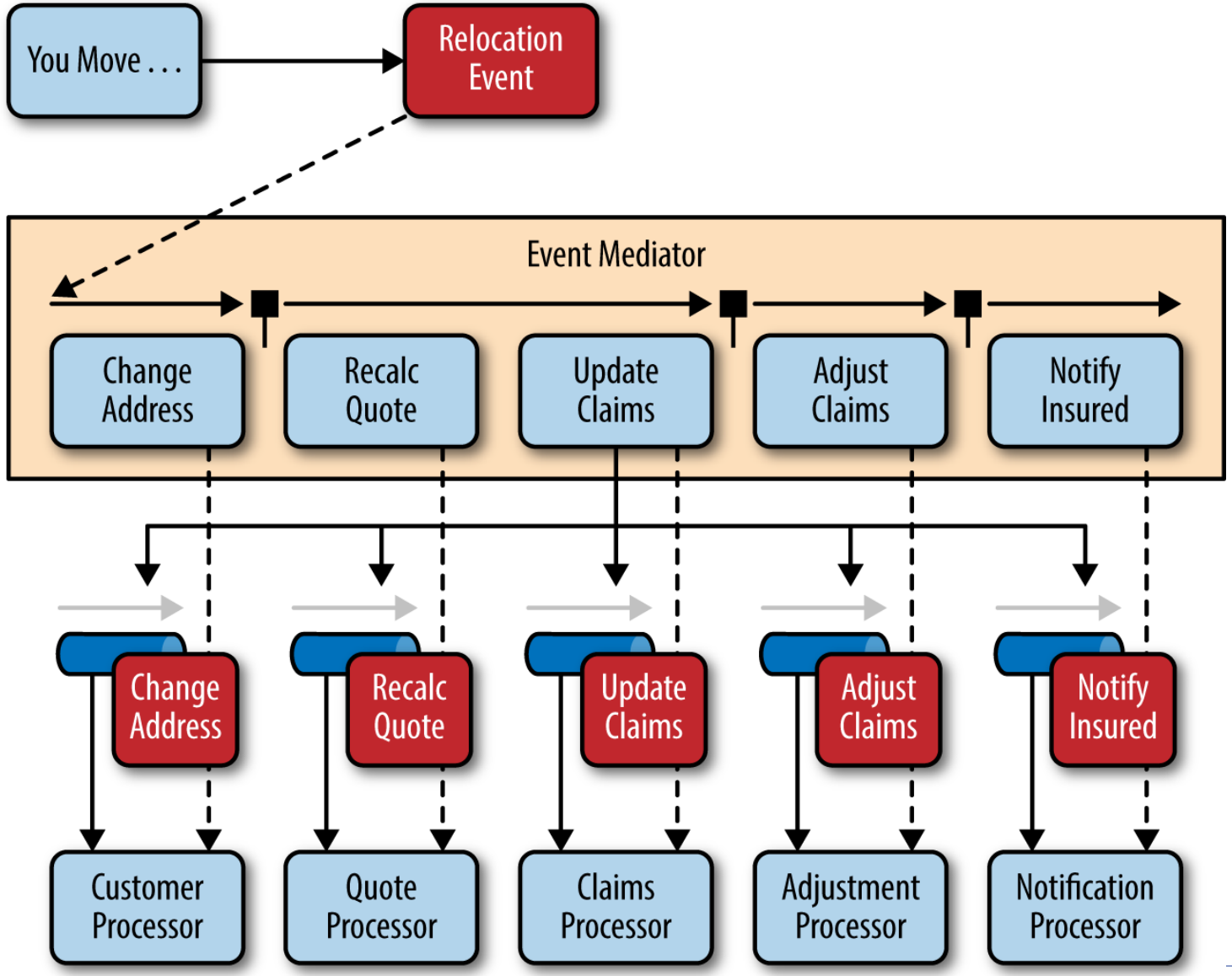
Event-Driven Architecture

- The event-driven architecture pattern is a popular distributed asynchronous architecture pattern used to produce highly scalable applications.
  - It is also highly adaptable and can be used for small applications and as well as large, complex ones.
  - It is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events.
  - It consists of two main topologies, the mediator(中介者) and the broker (代理)

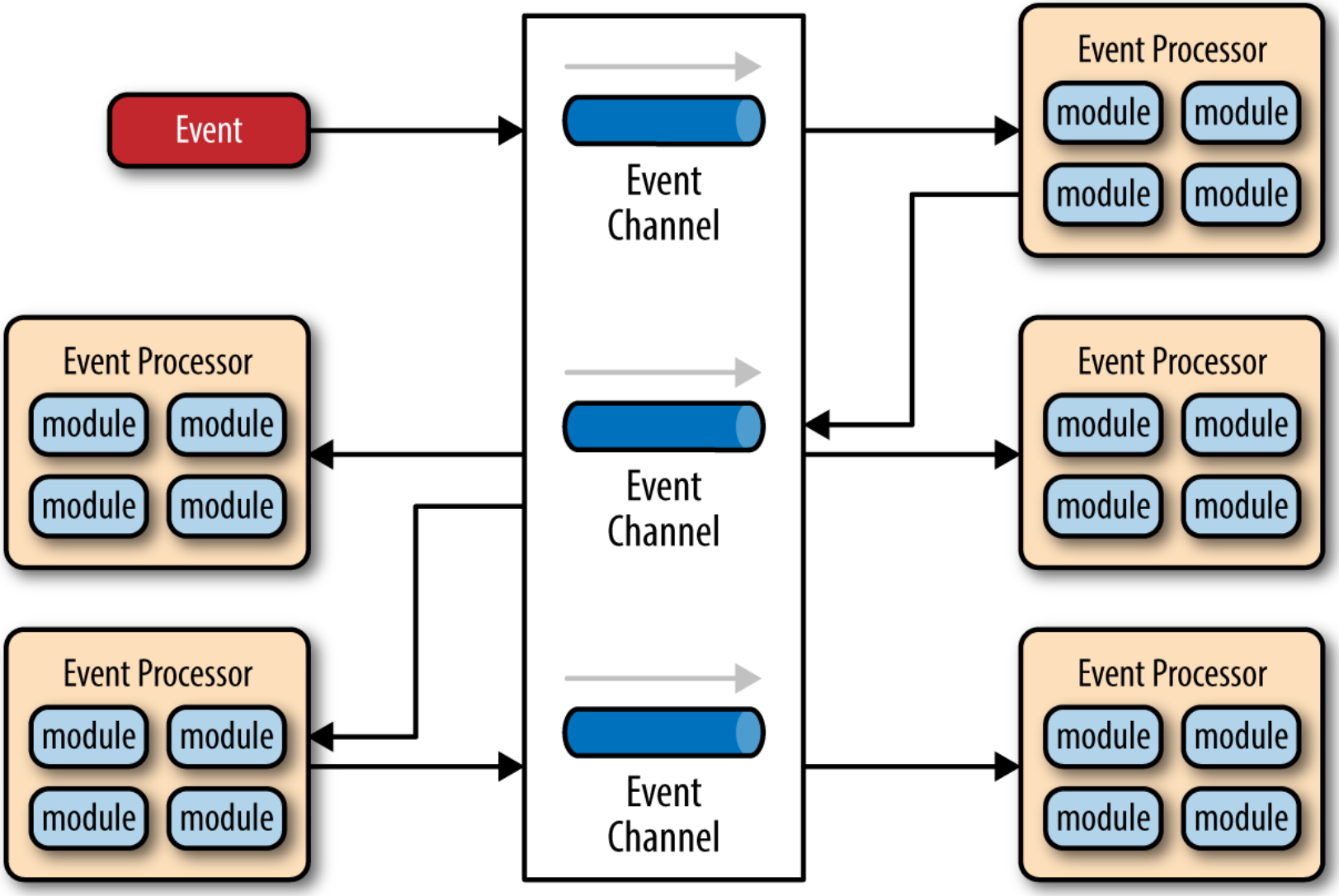
Pattern Description (Mediator)



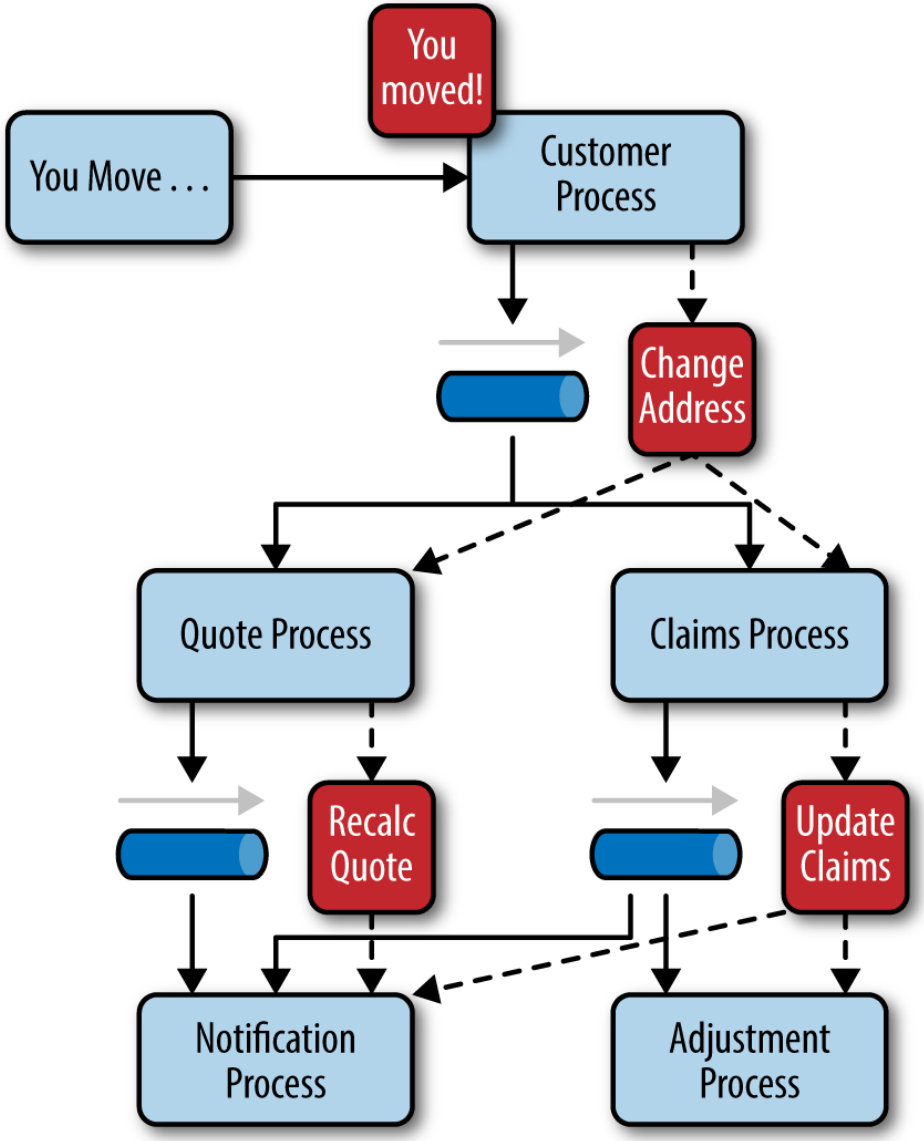
Example



Pattern Description (Broker)



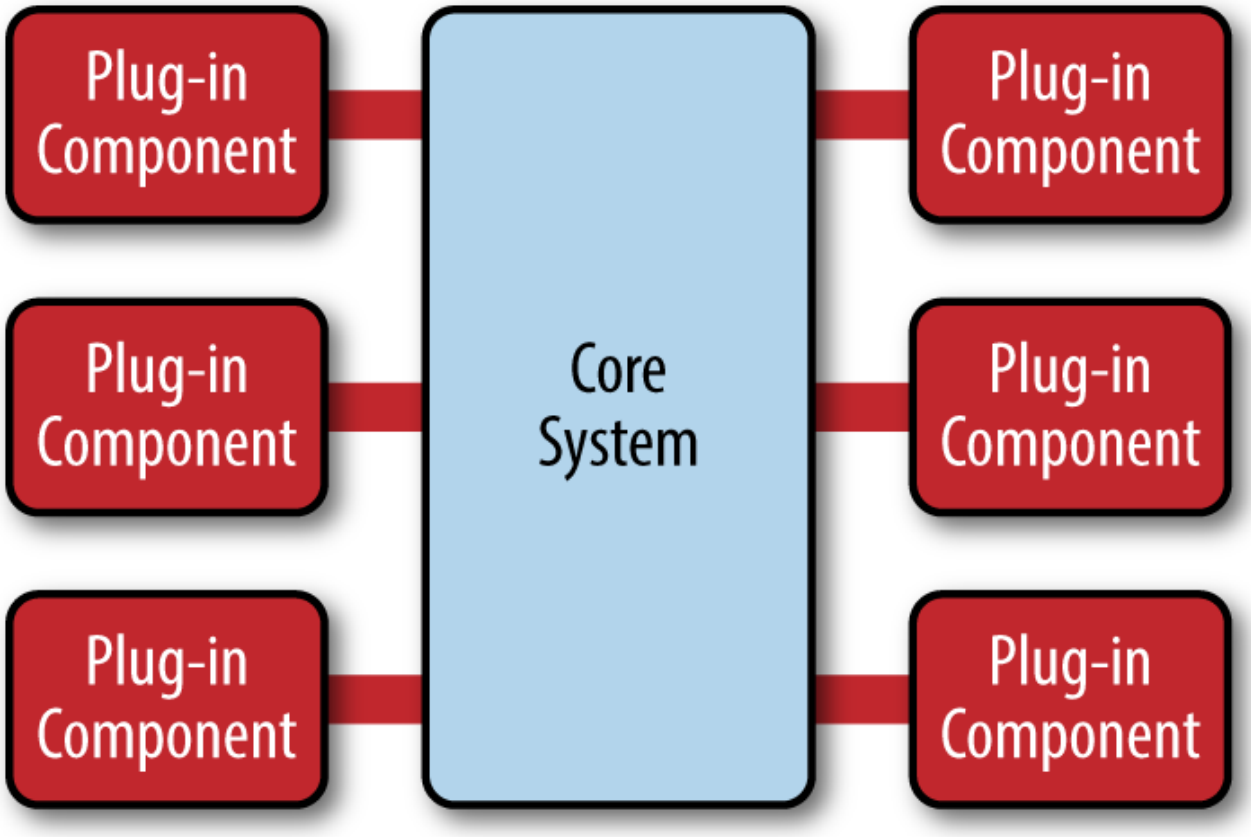
Example



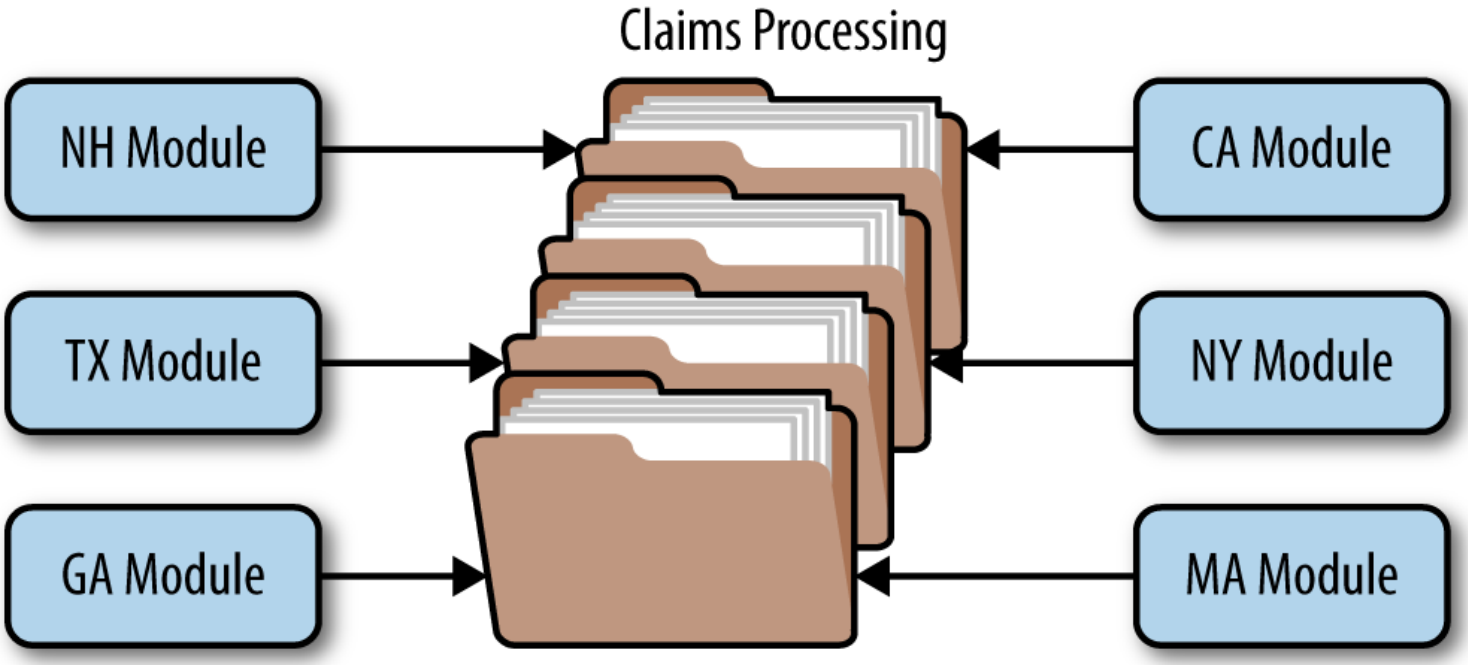
Microkernel Architecture

- The microkernel architecture pattern (plug-in architecture pattern) is a natural pattern for implementing product-based applications
  - A product-based application is one that is packaged and made available for download in versions as a typical third-party product
  - The microkernel architecture pattern allows you to add additional application features as plug-ins to the core application, providing extensibility as well as feature separation and isolation.

Pattern Description



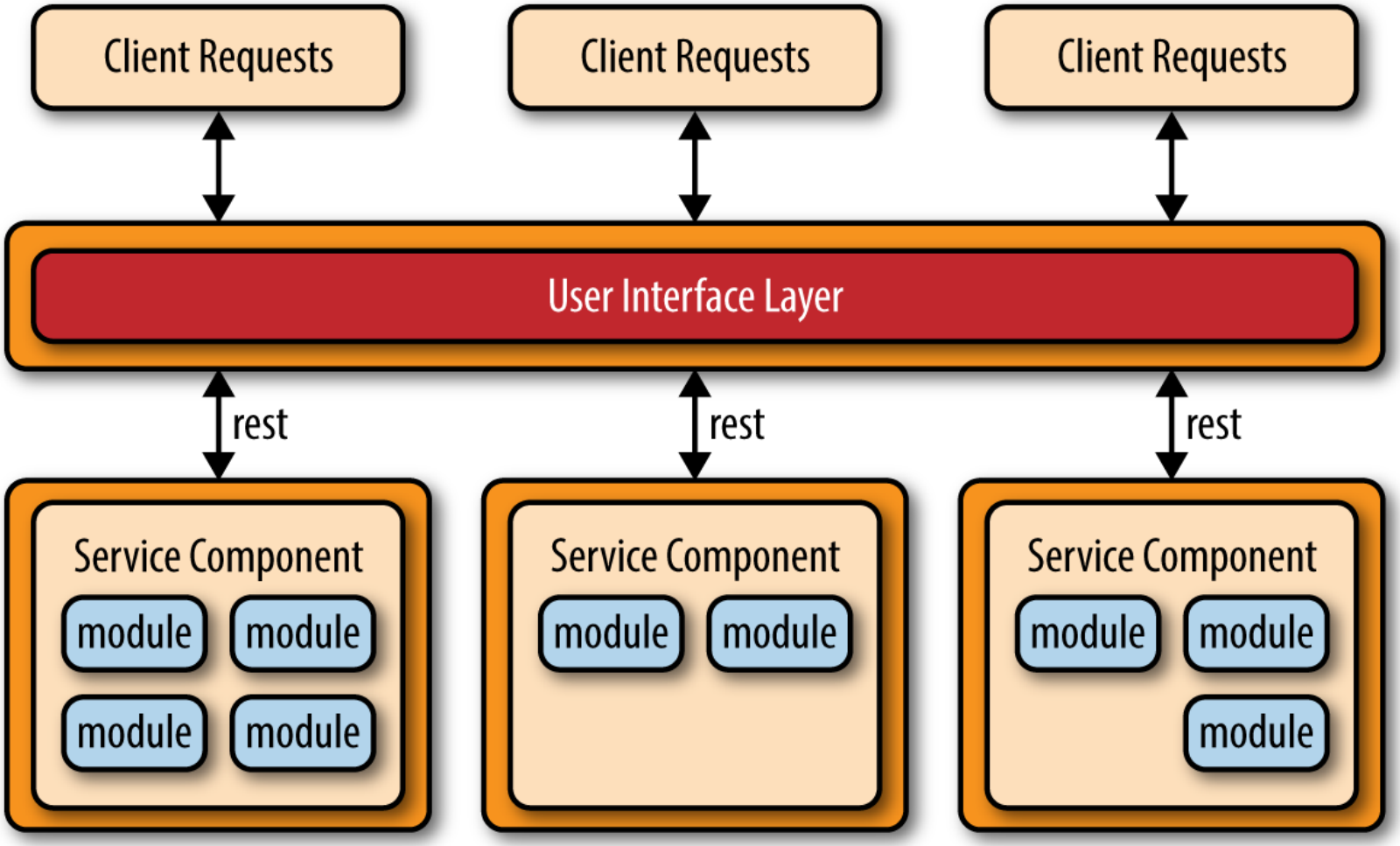
Example: Insurance Claims Processing



Microservices Architecture Pattern

- The microservices architecture pattern is quickly gaining ground in the industry as a viable alternative to monolithic applications and service-oriented architectures.
  - Because this architecture pattern is still evolving, there's a lot of confusion in the industry about what this pattern is all about and how it is implemented.
  - Designing the right level of service component granularity is one of the biggest challenges within a microservices architecture.

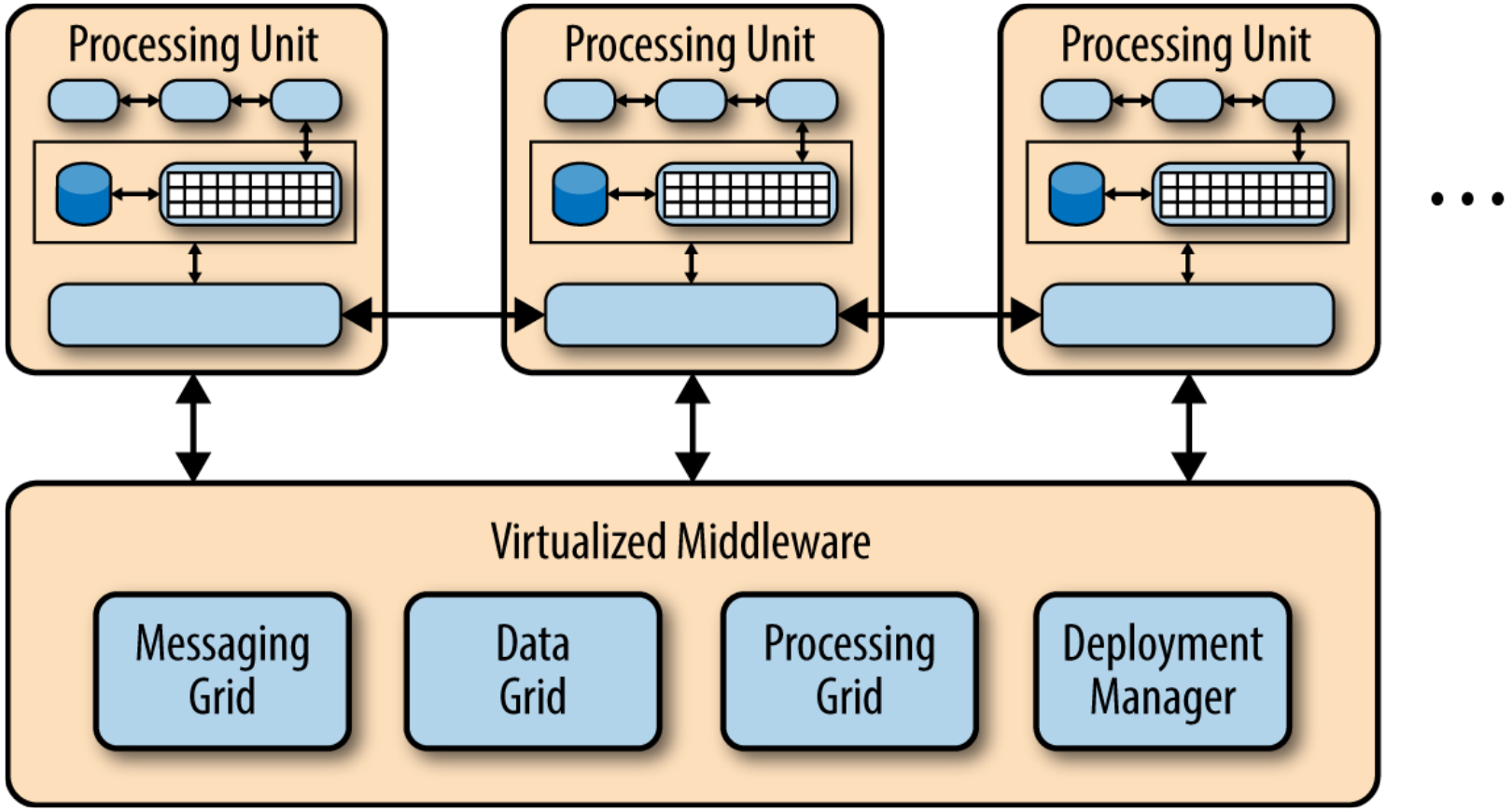
Pattern Description



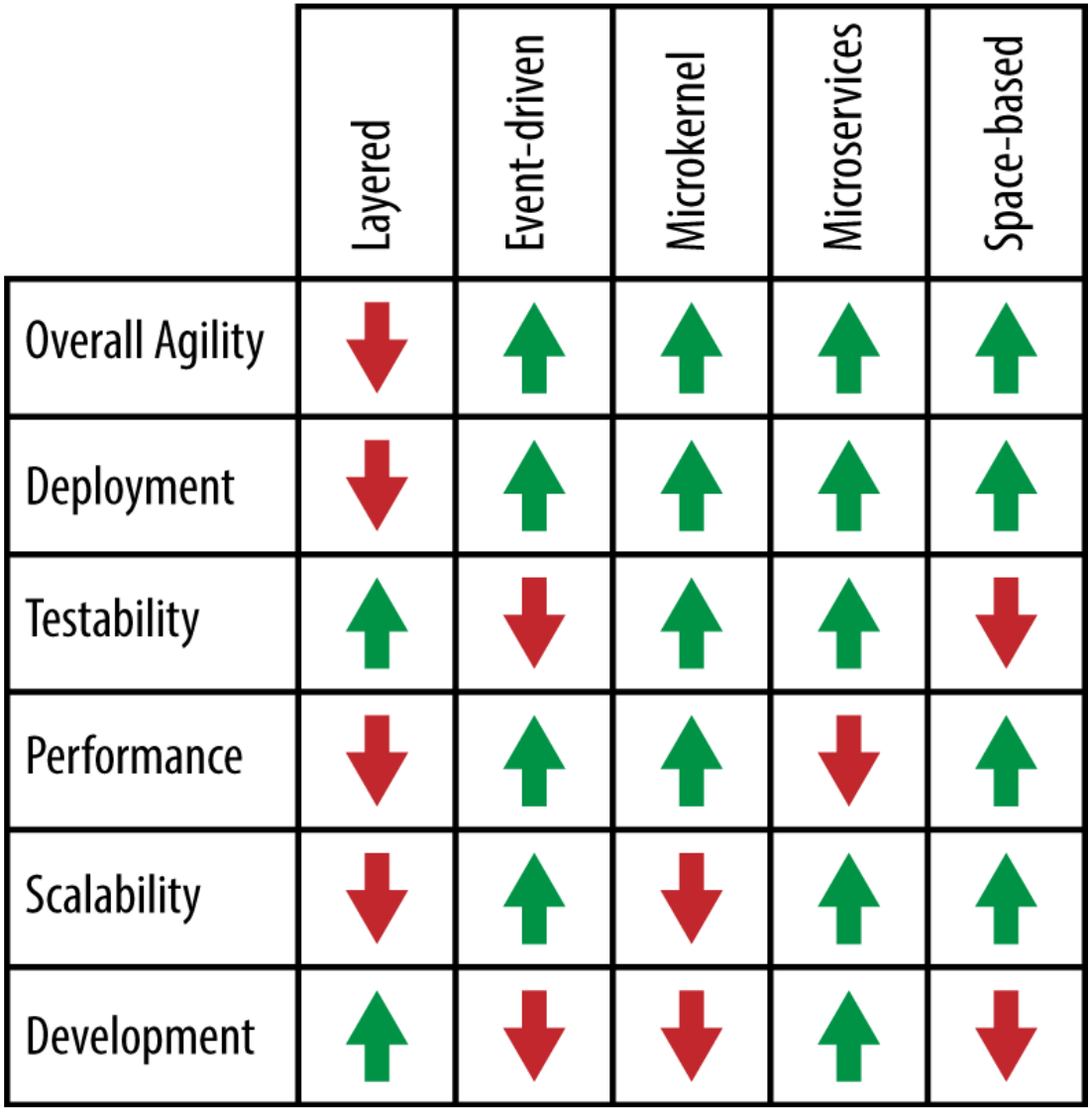
Space/Cloud-Based Architecture

- The space-based architecture pattern is specifically designed to address and solve scalability and concurrency issues.
  - It is also a useful architecture pattern for applications that have variable and unpredictable concurrent user volumes.
  - Solving the extreme and variable scalability issue architecturally is often a better approach than trying to scale out a database or retrofit caching technologies into a non-scalable architecture.

Pattern Description



Pattern Analysis Summary

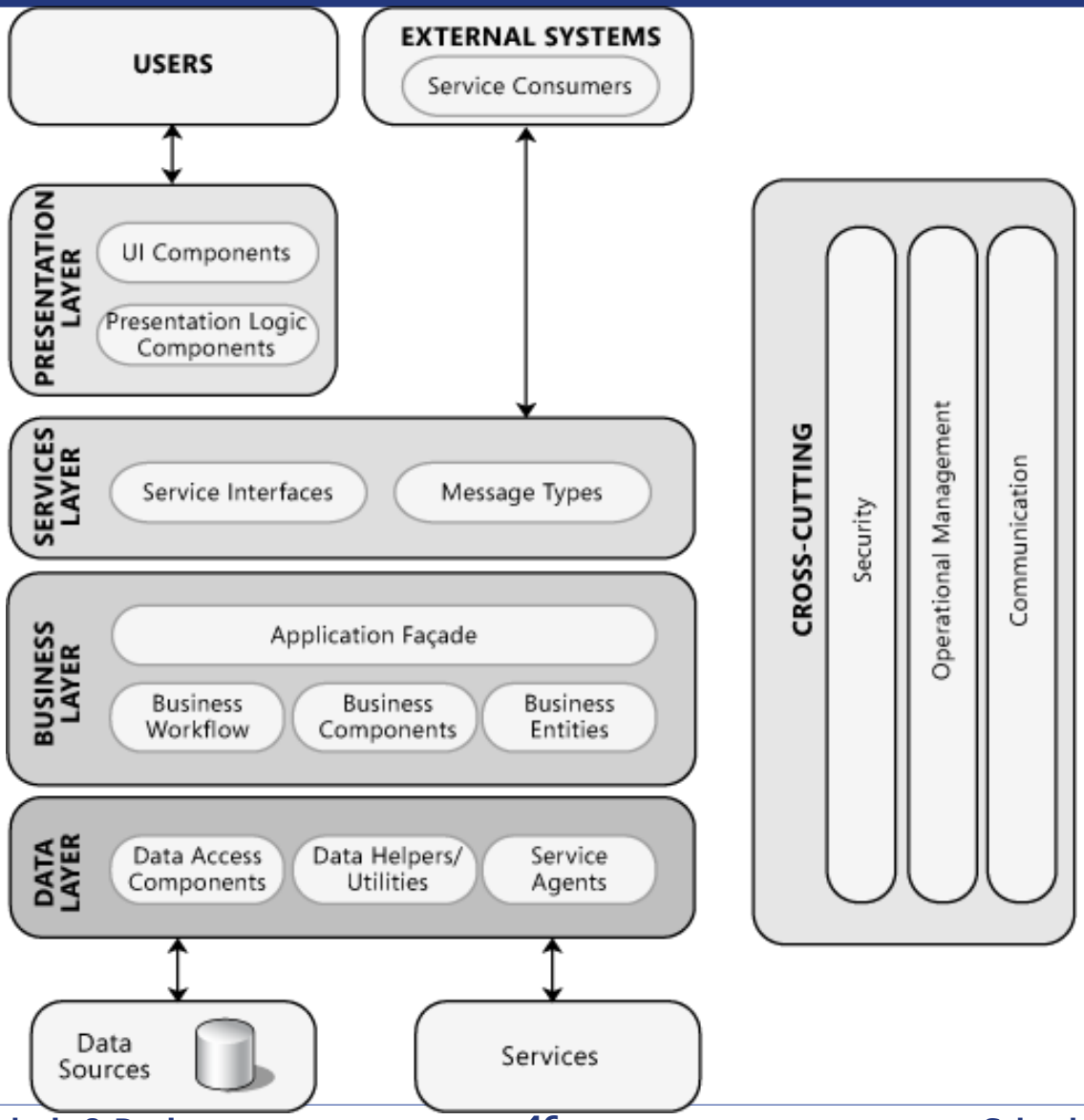


	Layered	Event-driven	Microkernel	Microservices	Space-based
Overall Agility	↓	↑	↑	↑	↑
Deployment	↓	↑	↑	↑	↑
Testability	↑	↓	↑	↑	↓
Performance	↓	↑	↑	↓	↑
Scalability	↓	↑	↓	↑	↑
Development	↑	↓	↓	↑	↓

Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Common application architecture



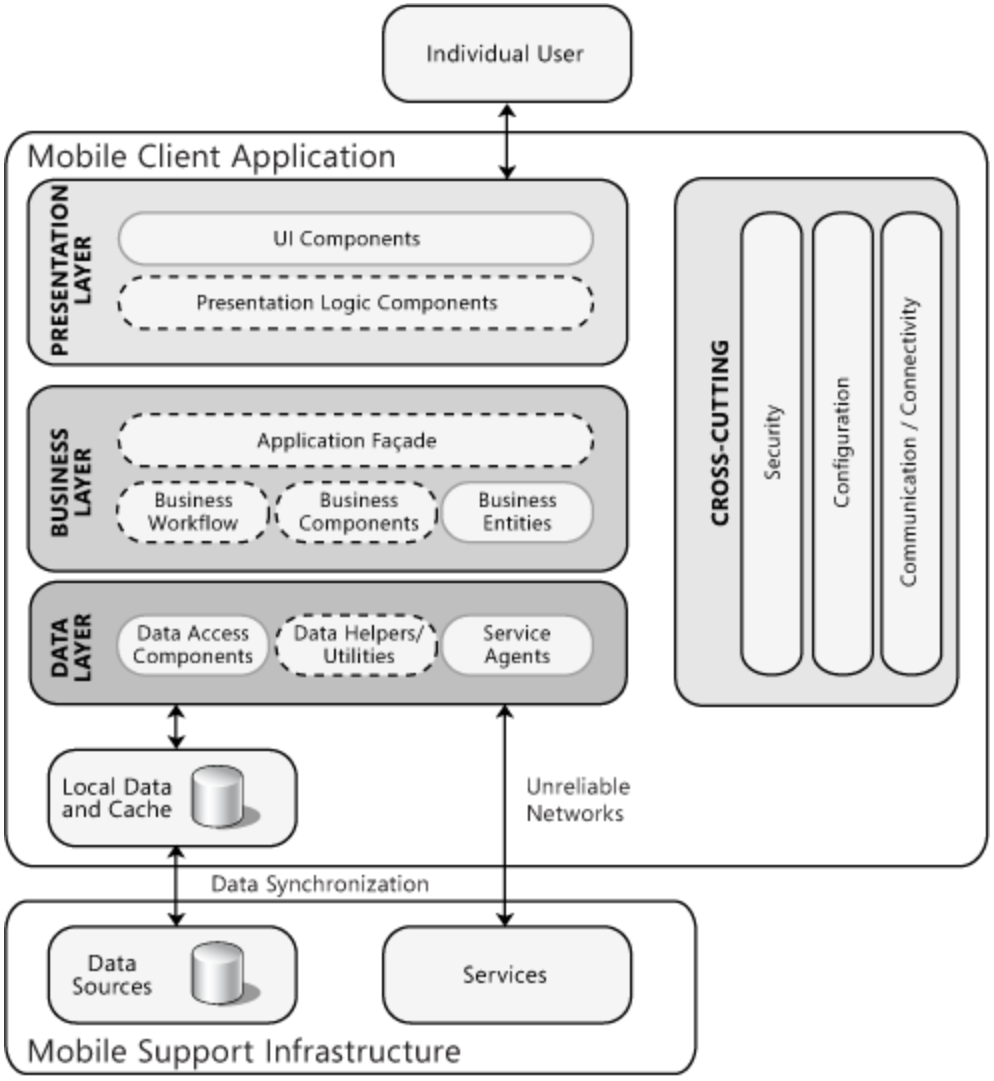
Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

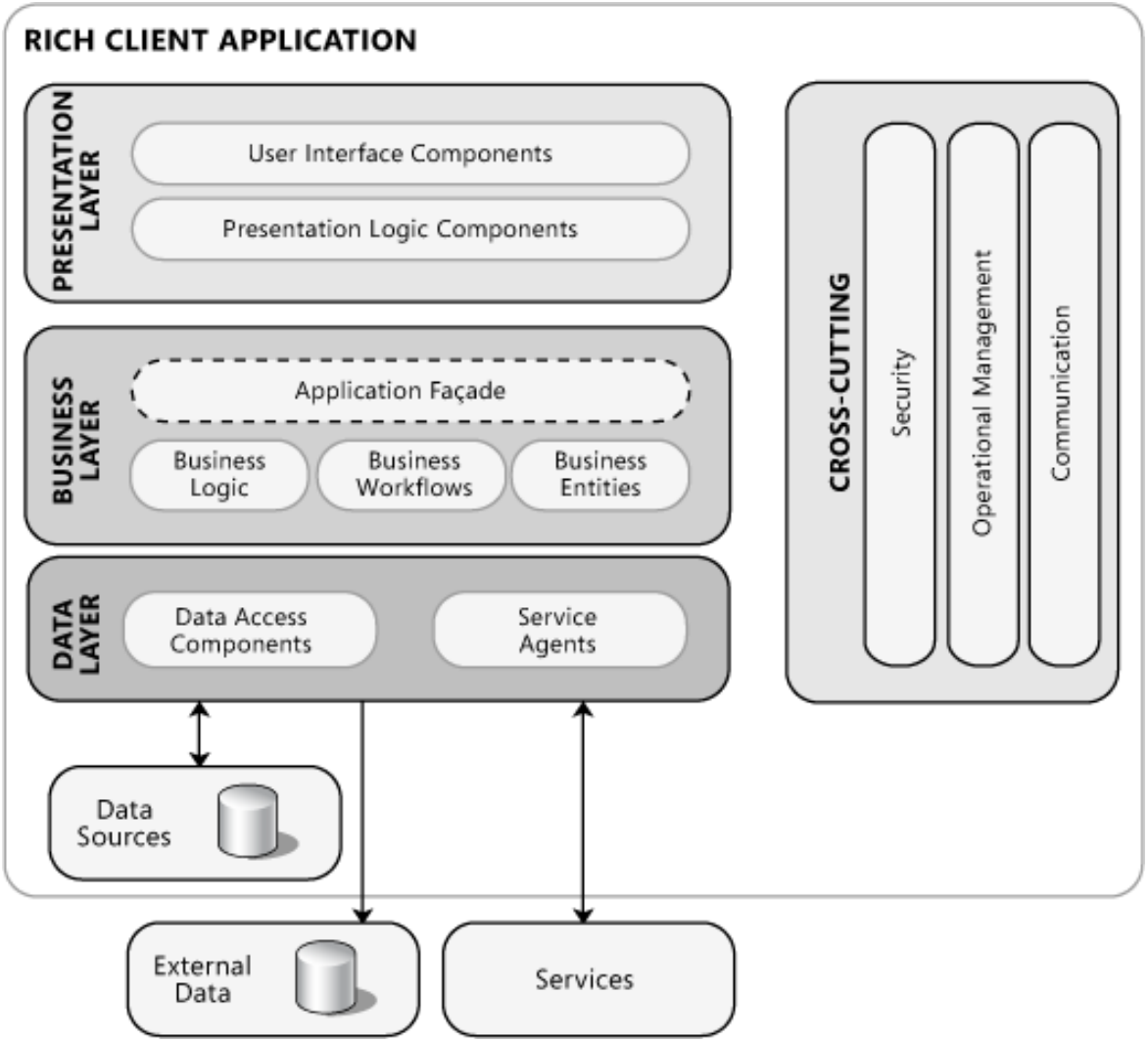
Determine the Application Type

- The following are the common basic types of applications that you may decide to build:
  - Mobile Applications
  - Rich Client Applicatios
  - Rich Internet Applications (RIA)
  - Service Applications
  - Web Applications

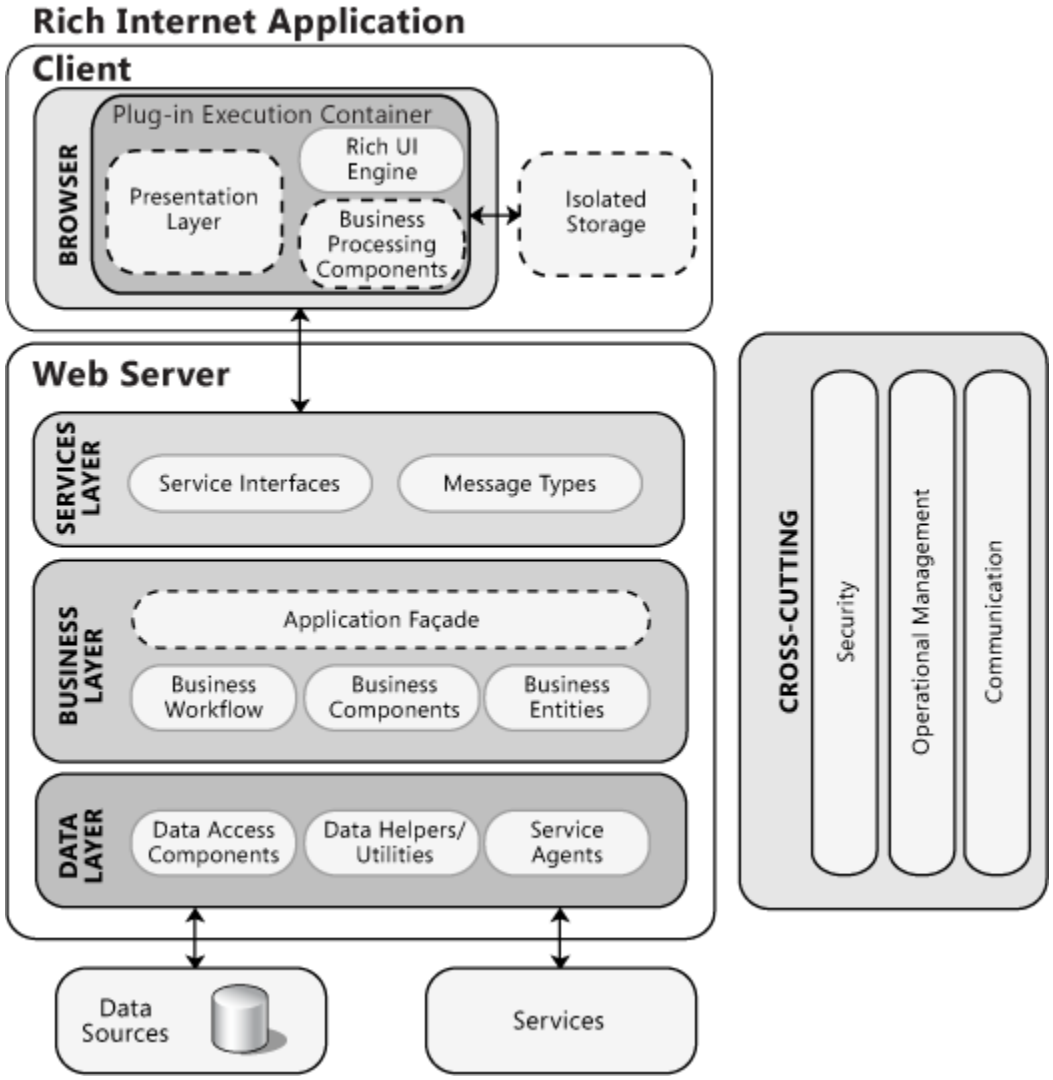
Mobile Application Archetype



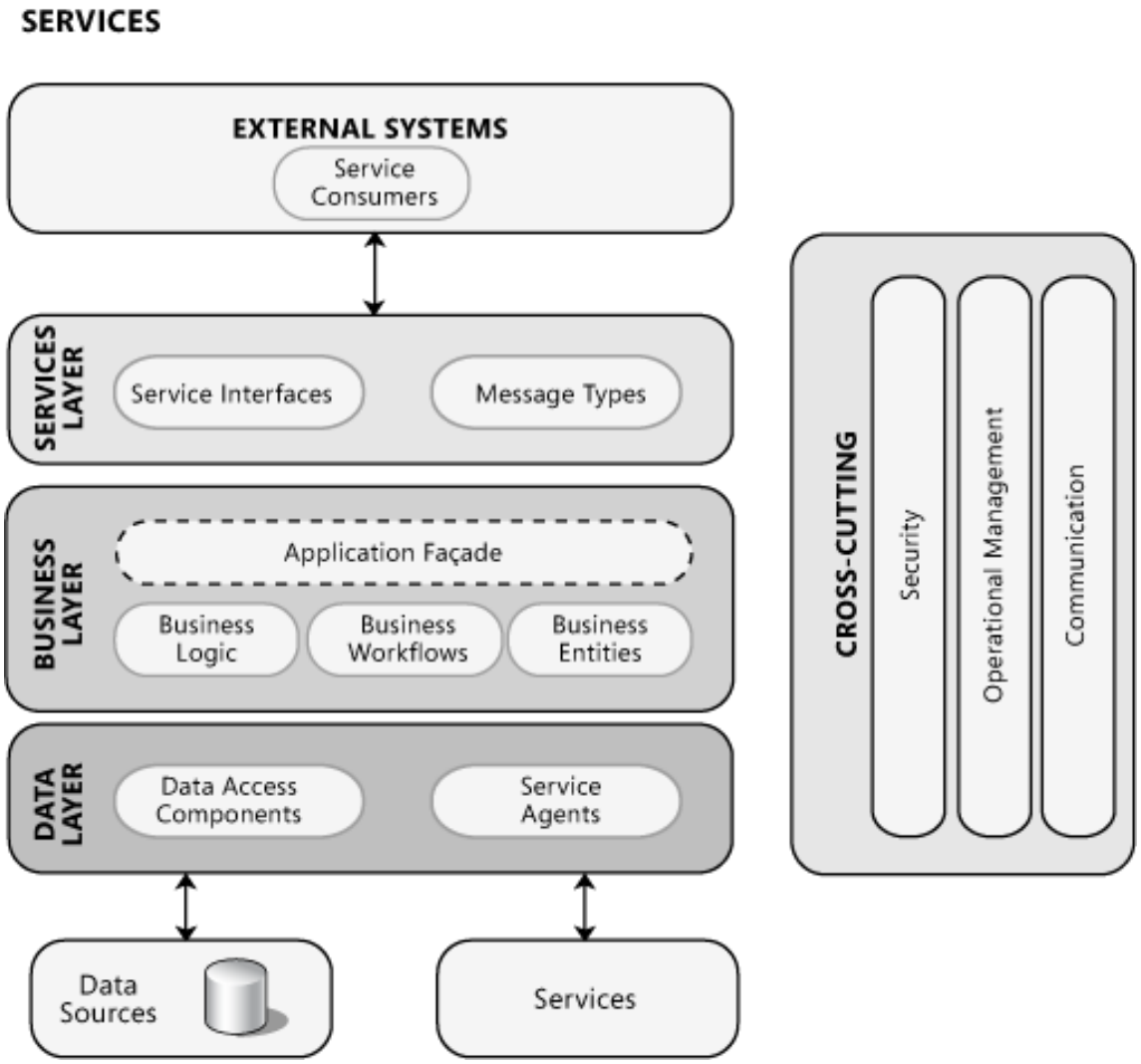
Rich Client Application Archetype



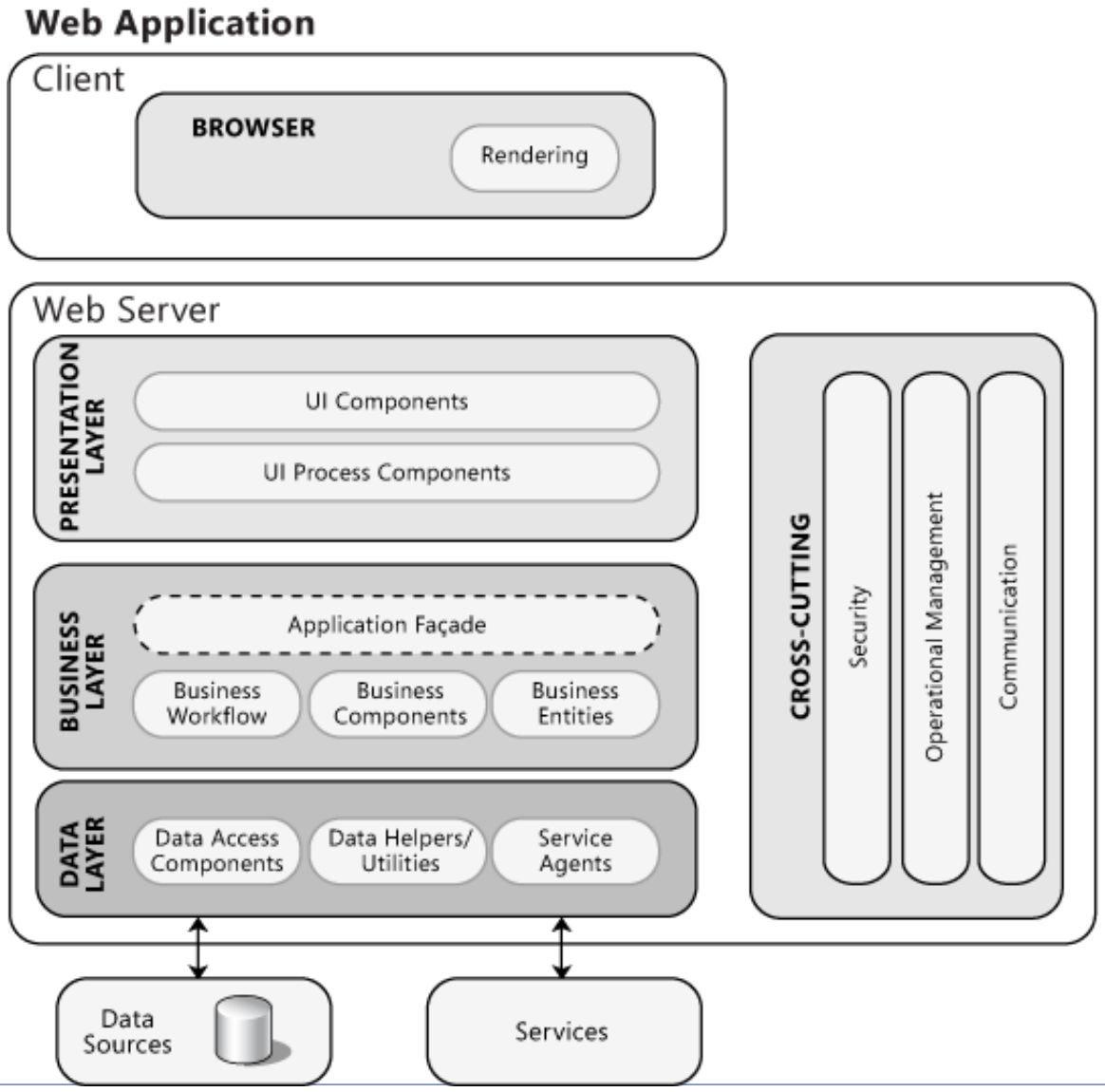
Rich Internet Application Archetype



Service Archetype



Web Application Archetype



Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Determine the Deployment Strategy

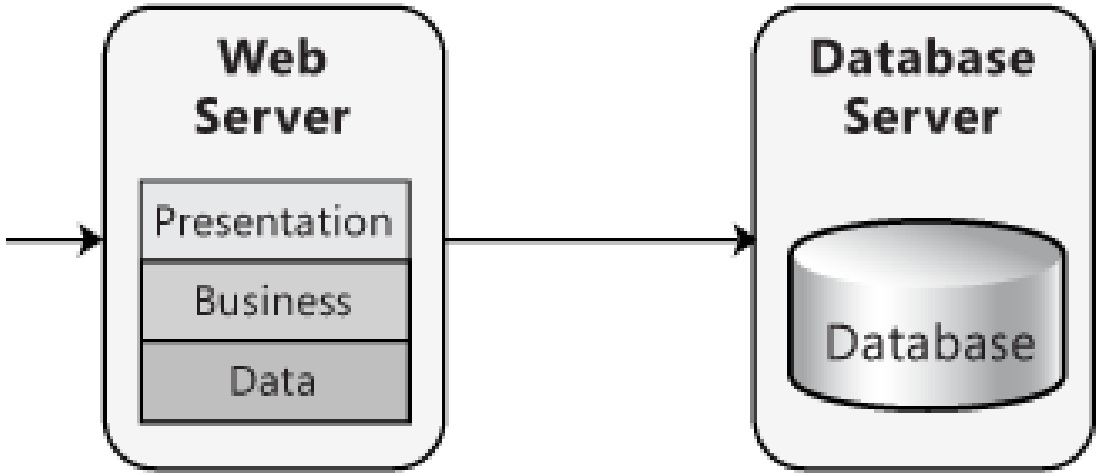
- Applications must be deployed into a physical environment where infrastructure limitations may negate some of the architectural decisions
- You must consider the proposed deployment scenario and the infrastructure as part of your application design process
  - Understand the target physical environment for deployment
  - Understand the architectural and design constraints based on the deployment environment
  - Understand the security and performance impacts of your deployment environment

Distributed and Nondistributed Deployment

- When creating your deployment strategy, first determine if you will use a distributed or a nondistributed deployment model
- If you are building a simple intranet application for your organization, which will be accessed by finite set of users, consider nondistributed deployment
- If you are building a more complex application, which you must optimize for scalability and maintainability, consider a distributed deployment.

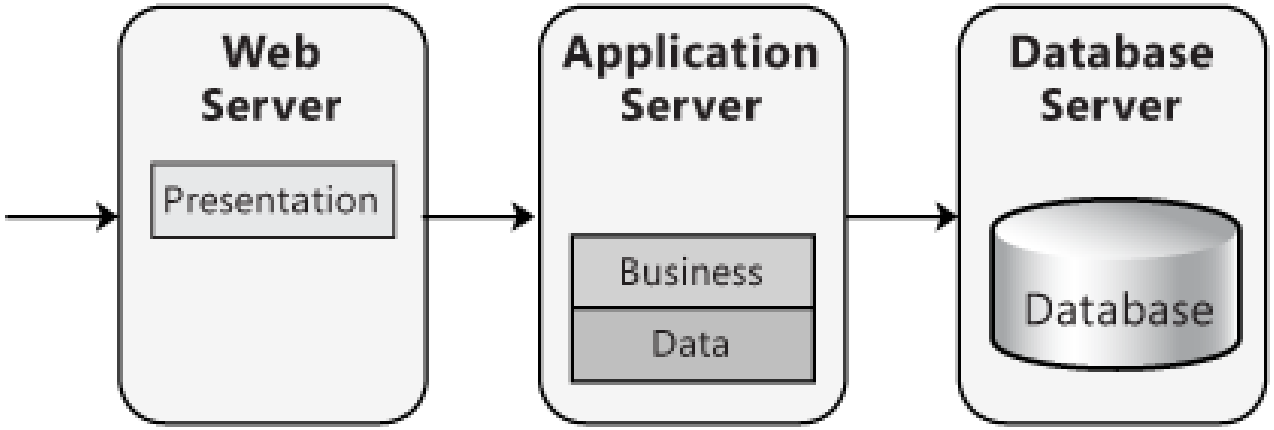
Nondistributed Deployment

- In a nondistributed deployment, all of the functionality and layers reside on a single server except for data storage functionality
- This approach has the advantage of simplicity and minimizes the number of physical servers required



Distributed Deployment

- In a distributed deployment, the layers of the application reside on separate physical tiers
- Tiered distribution organizes the system infrastructure into a set of physical tiers to provide specific server environments optimized for specific operational requirements and system resource usage

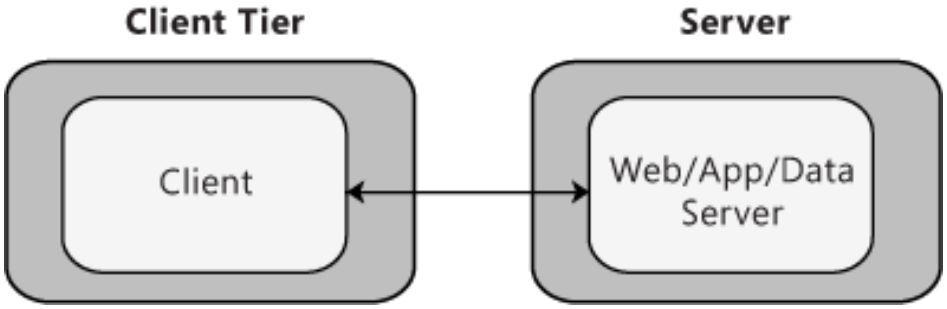


Performance and Design Considerations for Distributed Environments

- Distributing components can reduce performance because of the cost of remote calls across physical boundaries. However, distributing components can improve scalability opportunities, improve manageability, and reduce costs
  - Choose communication paths and protocols between tiers to ensure that components can securely interact with minimum performance degradation
  - Consider using services and operating system features such as distributed transaction support and authentication that can simplify your design and improve interoperability
  - When layers communicate across physical boundaries, you must consider how you will manage state across tiers

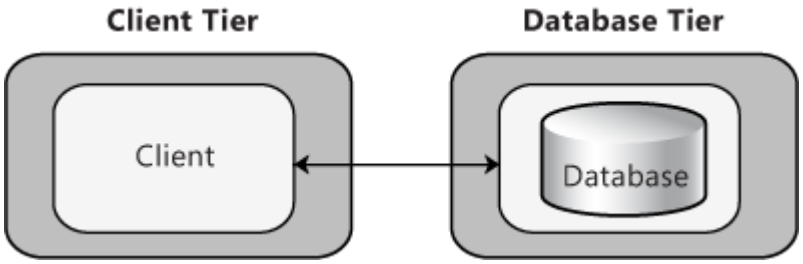
Distributed Deployment Patterns

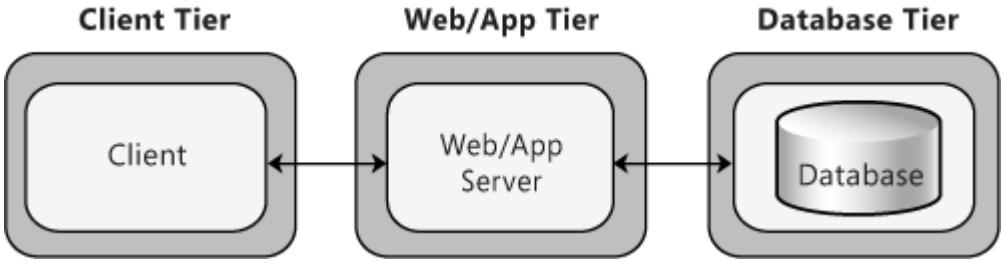
- Several common patterns represent application deployment structures found in most solutions
  - Client-Server Deployment

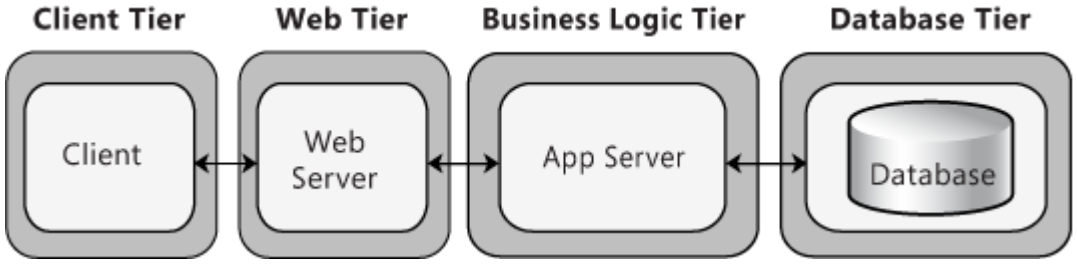


  - *n*-Tier Deployment

n-Tier Deployment







Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Determine the Appropriate Technologies

- Determine the Appropriate Technologies
  - Application Infrastructure / Layer
  - Development Tools
  - Database Server
  - Data Access
  - Collaboration/Integration/Workflow
  - Mobile
  - Rich Client
  - Rich Internet Application (RIA)
  - Services
  - Web
  - Web Server

Microsoft Application Platform

Application Infrastructure	.NET Framework
Application Layer	Asp.Net MVC
Collaboration/Integration /Workflow	Windows Workflow Foundation (WF)
	Microsoft Office SharePoint Server (MOSS)
	Microsoft BizTalk Server
Data Access	ADO.NET Core
	ADO.NET Data Services Framework
	ADO.NET Entity Framework
	ADO.NET Sync Services
	Language Integrated Query (LINQ)
Database Server Development Tools	Microsoft SQL Server
	Microsoft Visual Studio
	Microsoft Expression® Studio design software
Mobile	.NET Compact Framework
	ASP.NET Mobile
	Silverlight Mobile
Rich Client	Windows Forms
	Windows Presentation Foundation (WPF)
Rich Internet Application (RIA) Services	Microsoft Silverlight
	ASP.NET Web Services (ASMX)
	Windows Communication Foundation (WCF)
Web	ASP.NET
Web Server	Internet Information Services (IIS)

Java Platform

Application Infrastructure	JDK, Java EE
Application Layer	SSH/ Spring MVC
Collaboration/Integration/Workflow	JBPM
	IBM WebSphere Application Server
	Oracle WebLogic Server
Data Access	JDBC
	Hibernate
	iBATIS
Database Server	MySql
	Oracle
	DB2
Development Tools	Eclipse
	Netbeans
Mobile	Android (Java)
	HTML5
Rich Client	Swing
	JavaFX
Rich Internet Application (RIA)	JavaScript
	Flash
Services	JWS (Java WebService)
	Apache CXF
Web	JSP
Web Server	Tomcat, JBoss
	IBM WebSphere Application Server
	Oracle WebLogic Server

Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

Determine the Quality Attributes

- Quality attributes are the overall factors that affect run-time behavior, system design, and user experience
- They represent areas of concern that have the potential for application wide impact across layers and tiers
- Some of these attributes are related to the overall system design, while others are specific to run time, design time, or user centric issues

Common Quality Attributes (1)

Category	Quality attribute	Description
Design Qualities	Conceptual Integrity	Conceptual integrity defines the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as factors such as coding style and variable naming.
	Maintainability	Maintainability is the ability of the system to undergo changes with a degree of ease. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements.
	Reusability	Reusability defines the capability for components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components and also the implementation time.
Run-time Qualities	Availability	Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load.
	Interoperability	Interoperability is the ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. An interoperable system makes it easier to exchange and reuse information internally as well as externally.
	Manageability	Manageability defines how easy it is for system administrators to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning.

Common Quality Attributes (2)

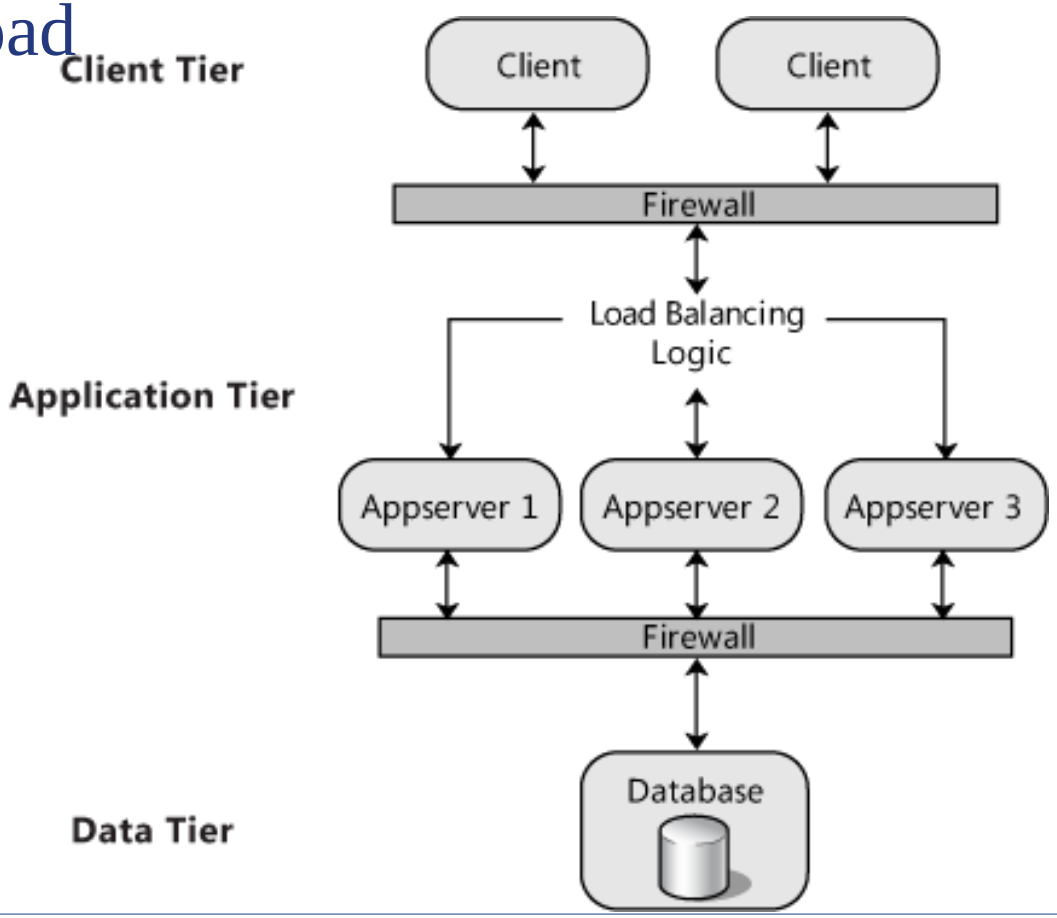
Run-time Qualities	Performance	Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time.
	Reliability	Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.
	Scalability	Scalability is ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
	Security	Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. A secure system aims to protect assets and prevent unauthorized modification of information.
System Qualities	Supportability	Supportability is the ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner.
User Qualities	Usability	Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience.

Performance Patterns

- Performance deployment patterns represent proven design solutions to common performance problems
- When considering a high performance deployment, you can scale up or scale out
  - Scaling up entails improvements to the hardware on which you are already running
  - Scaling out entails distributing your application across multiple physical servers to distribute the load

Load-balanced Cluster

- You can install your service or application onto multiple servers that are configured to share the workload

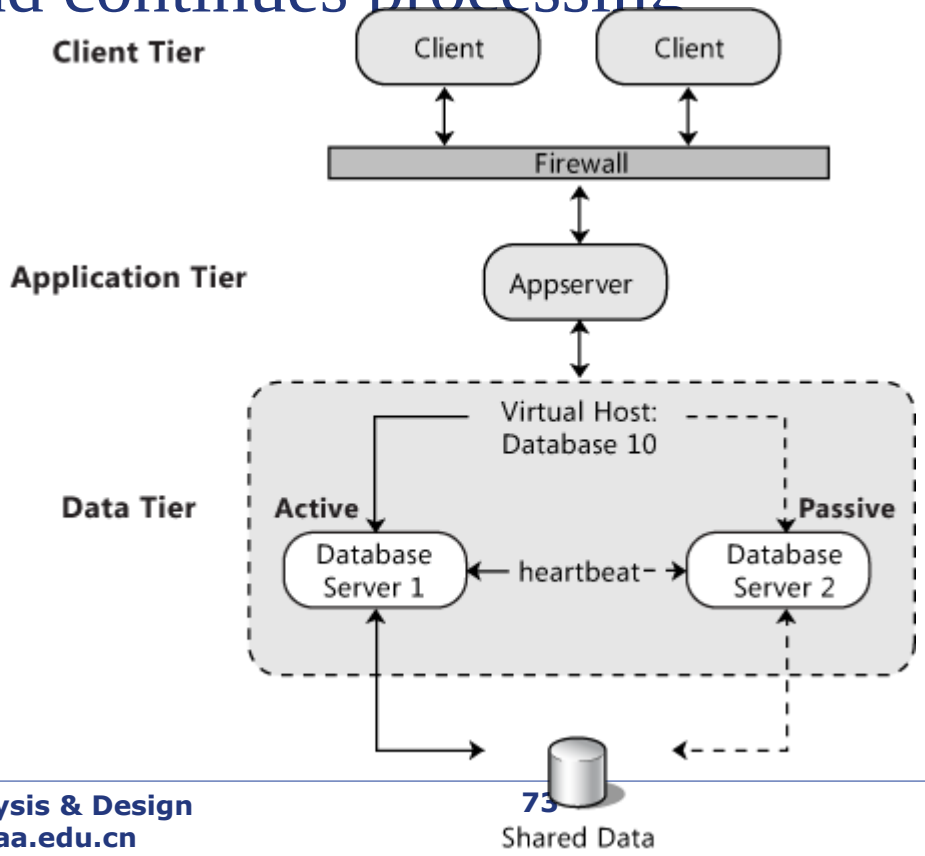


Reliability Patterns

- Reliability deployment patterns represent proven design solutions to common reliability problems
- The most common approach to improving the reliability of your deployment is to use a failover cluster to ensure the availability of your application, even if a server fails

A failover cluster

- A failover cluster is a set of servers that are configured in such a way that if one server becomes unavailable, another server automatically takes over for the failed server and continues processing



Contents

- Software Architecture
- Software Architecture Pattern
- Software Architecture Design
  - Determine the Application Type
  - Determine the Deployment Strategy
  - Determine the Appropriate Technologies
  - Determine the Quality Attributes
  - Determine the Crosscutting Concerns

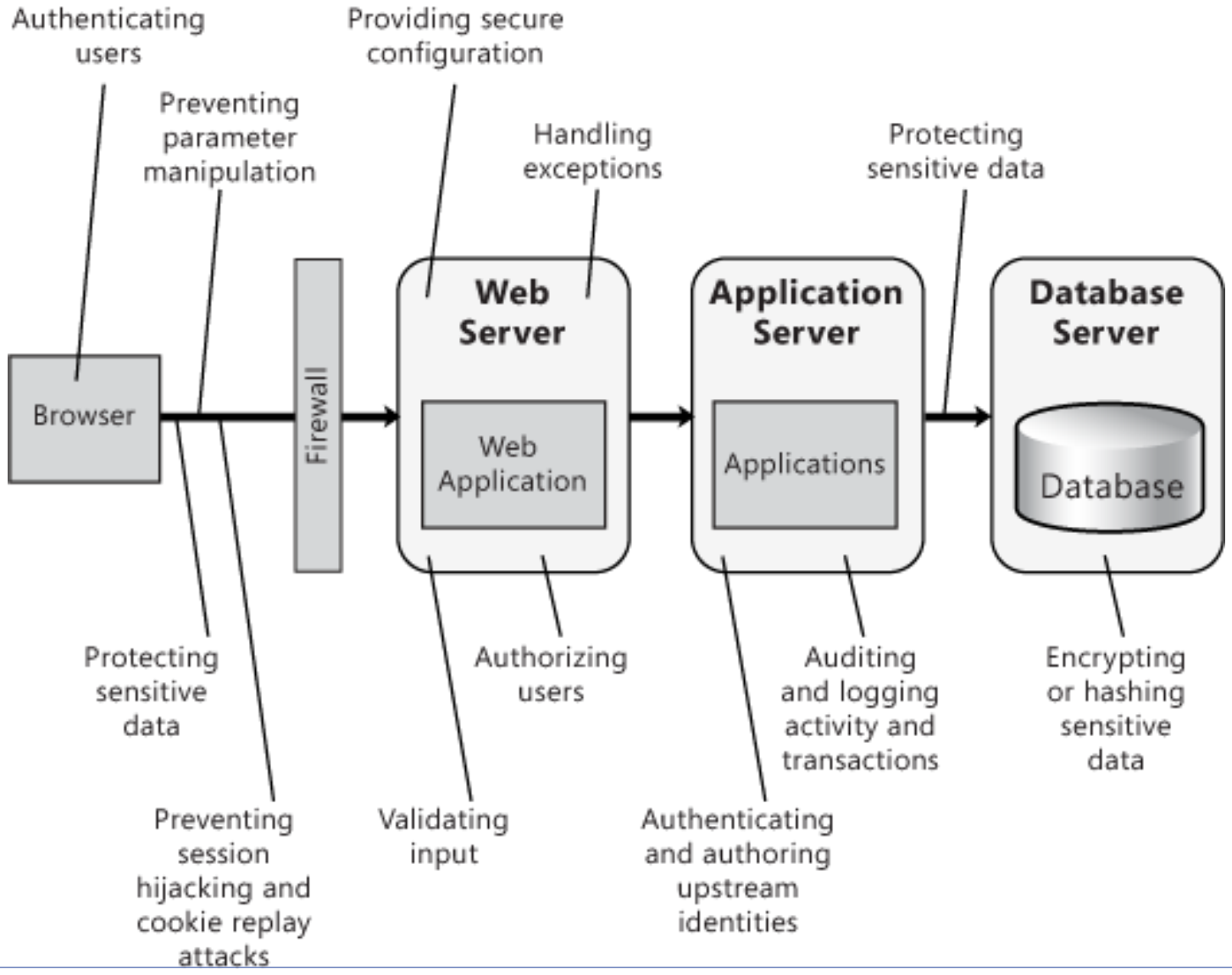
Determine the Crosscutting Concerns

- Crosscutting concerns represent key areas of your design that are not related to a specific layer in your application
- Some of the key crosscutting concerns
  - Security: Authentication & Authorization & Validation
  - Configuration Management
  - Exception Management
  - State Management
  - Logging
  - Caching
  - Communication

Security

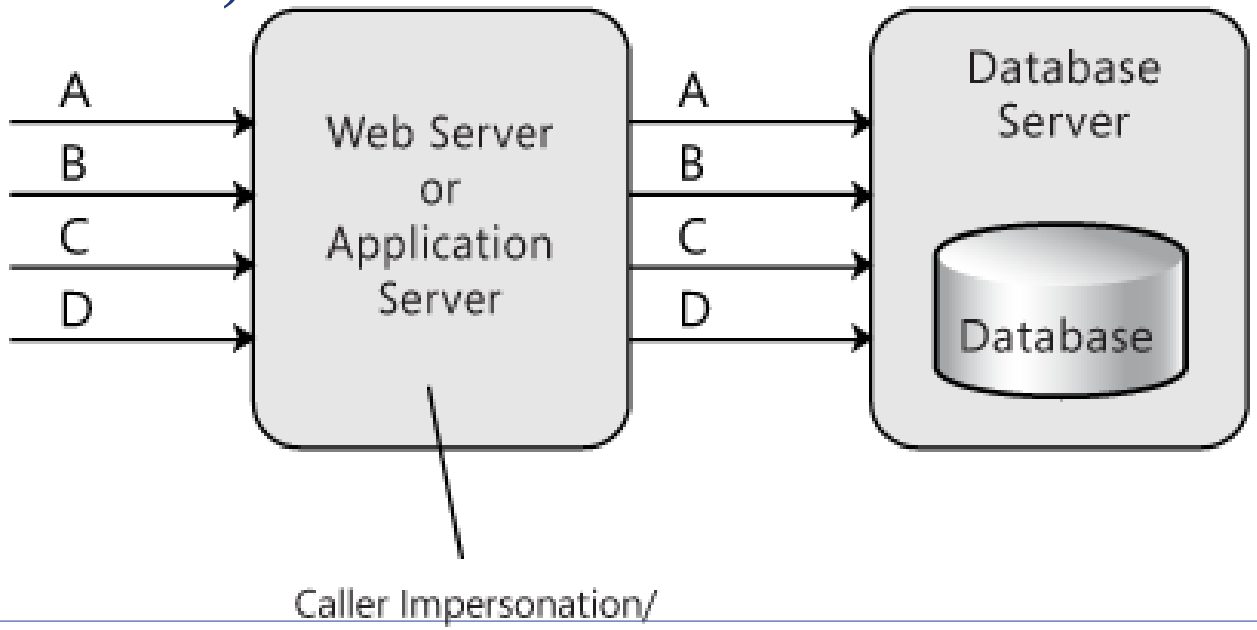
- Questions you might consider when examining the crosscutting concerns for security are
  - Authentication
  - Authorization
  - Input and Data Validation
  - Sensitive data
  - Configuration Management
  - Cryptography
  - Session Management
  - Auditing and Logging
  - Exception Management

Security issues identified in a typical Web application architecture



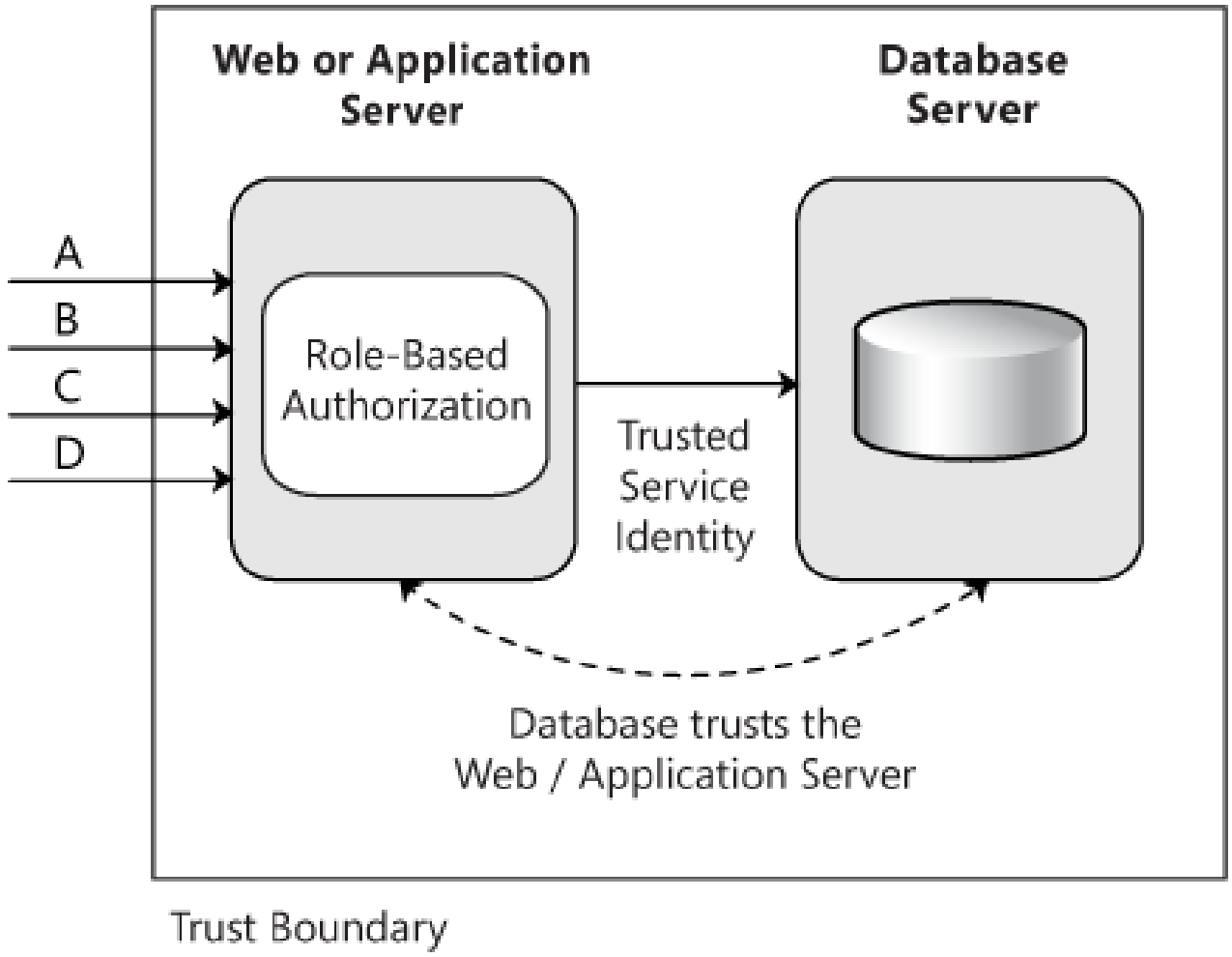
Security Patterns (1)

- The impersonation/delegation authorization model
  - resources and the types of operations (such as read, write, and delete) permitted for each one are secured using Access Control Lists (ACLs) or the equivalent security features of the targeted resource (such as tables and procedures in database)



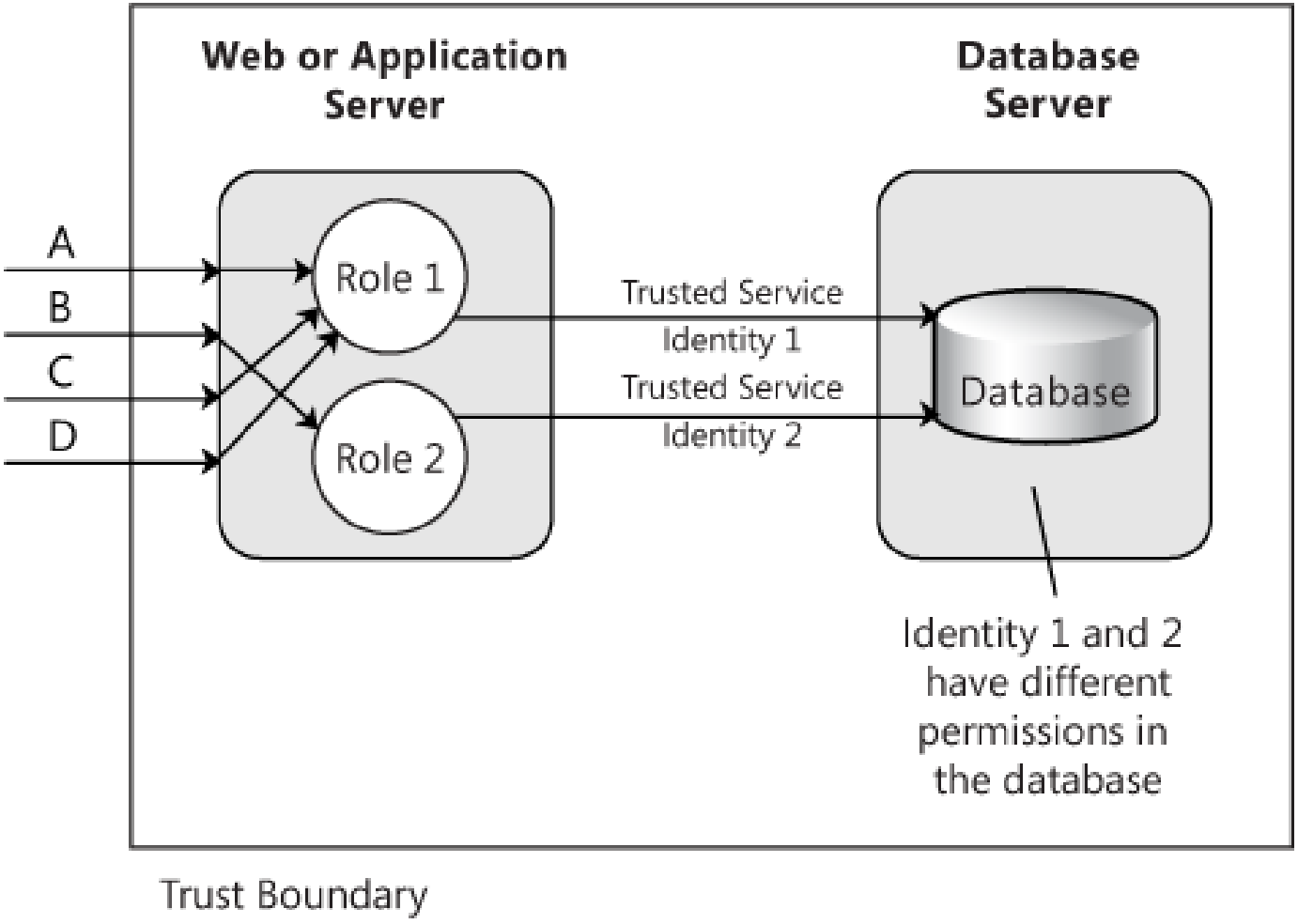
Security Patterns (2)

- The trusted subsystem (or trusted server) model



Security Patterns (3)

- The multiple trusted service identities model



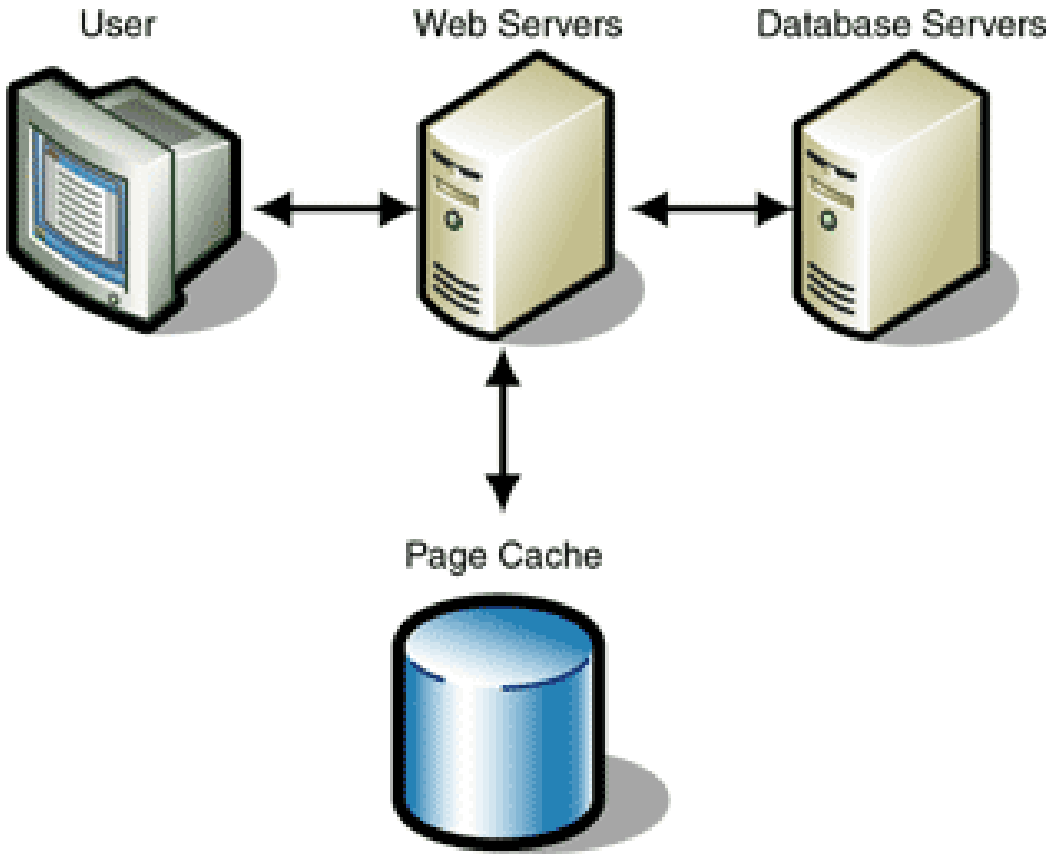
Caching

- Caching can play a vital role in maximizing performance
  - Determine the Data to Cache
  - Determine Where to Cache Data
  - Determine the Format of Your Data to Cache
  - Determine a Suitable Cache Management Strategy
  - Determine How to Load the Cache Data

Caching Patterns

- Cache Dependency
  - Use external information to determine the state of data stored in a cache
  - Technology Options
    - Asp .Net: Cache, Session, Application
    - Java: EhCache, Hibernate, Spring
    - Redis
- Page Cache
  - Improve the response time for dynamic Web pages that are accessed frequently, but that change less often and consume a large amount of system resources to construct

Basic page cache configuration




```
<%@ OutputCache Duration="60" VaryByParam="none" %>
```

Communication and Messaging

- When components are located on the same physical tier, you can often rely on direct communication between these components
- However, if you deploy components and layers on physically separate servers and client machines—as is likely in most scenarios—you must consider how the components in these layers will communicate with each other efficiently and reliably
- In general, you must choose between direct communication (such as method calls between components) and message-based communication.

Communication issues

- Asynchronous vs. Synchronous Communication
- Coupling and Cohesion
- Data Formats
- Interoperability
- Performance
- State Management

Technology Options

- On the Microsoft platform, you can choose between two messaging technologies
  - Windows Communication Foundation (WCF)
  - ASP.NET Web Services (ASMX)
- On the Java platform
  - Java Web Services (JWS)
  - Java Message Service (JMS)
  - Enterprise Service Bus (ESB)

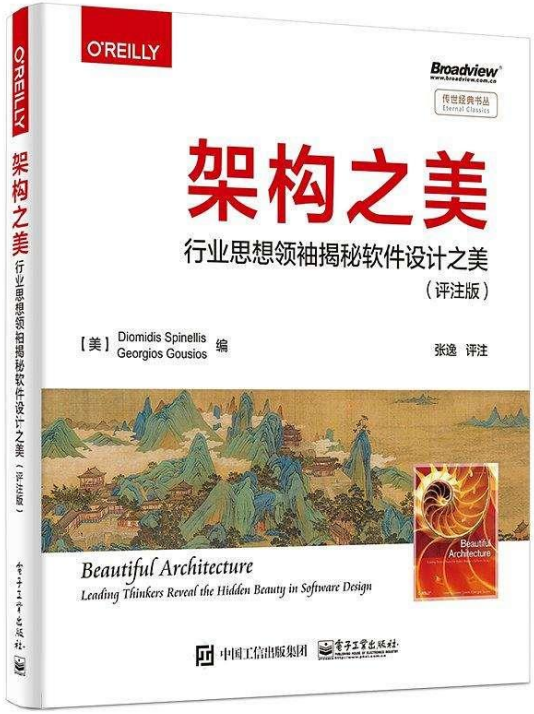
作业

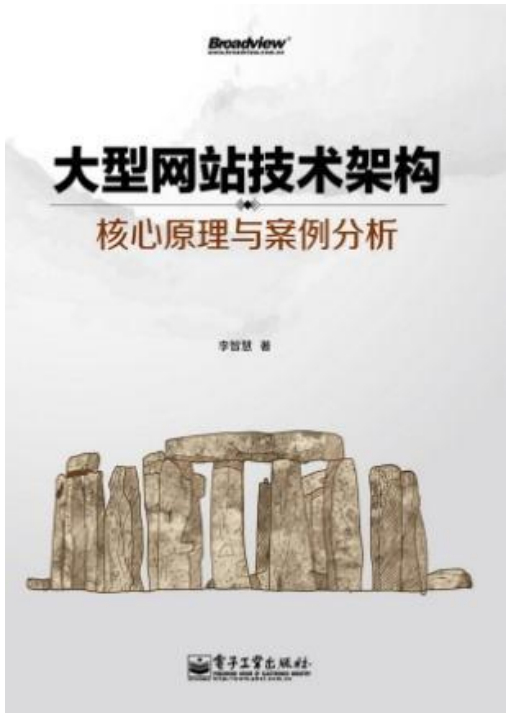
- 作业4: 架构设计 (15分)
  - 作业任务: 架构调研分析和系统架构设计
    - 任务1（5分）：自主选择市面上成熟的大规模软件系统架构，完成架构分析报告，可从以下任意方面选题
      - 调研经典的系统软件架构，如：Linux操作系统、Eclipse开发环境、JavaEE企业级开发框架
      - 调研一些大规模Web系统架构，如：12306、银行系统、淘宝/京东、视频直播网站
      - 调研某一个关键问题的解决方案，比如性能、可靠性、安全、大数据….
      - ……
    - 任务2（10分）：结合课堂讲解的架构内容和调研情况，设计目标系统的初步架构，完成架构设计说明书
  - 交付物：
    - 某某软件架构调研分析报告和PPT
    - 系统架构设计说明书和PPT

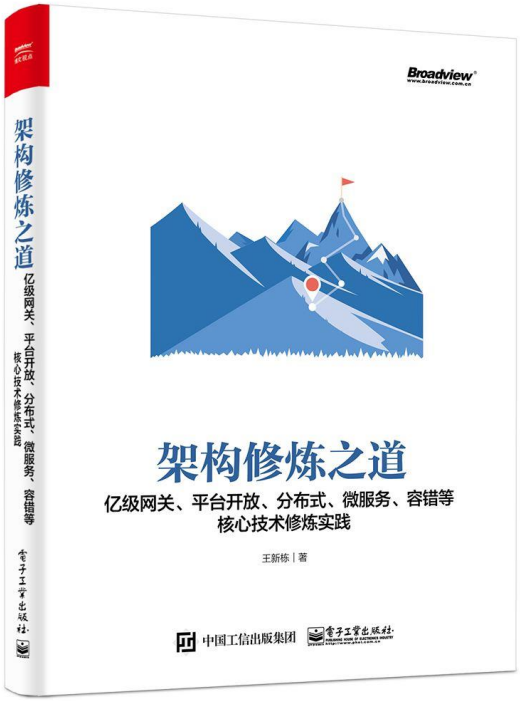
作业（续）

- 作业4：架构设计
  - 时间安排
    - 本次架构设计所有的组都需要进行讲解交流，但每个组至少选择一个内容讲解：架构调研或架构设计
    - 10月18日 (周二) ：安排各小组答辩情况
    - 10月20日&25日：讲解和交流架构设计方案
  - 终稿作业提交：2022-10-29，周六, 22:00
    - 某某软件架构调研分析报告和讲解PPT
    - 系统架构设计说明书和讲解PPT

推荐阅读







- 网上找一些相关的文章，如：
  - 中国银联跨中心，异构数据同步技术与实践
  - 技术揭秘12306改造
  - 淘宝技术架构演进之路