

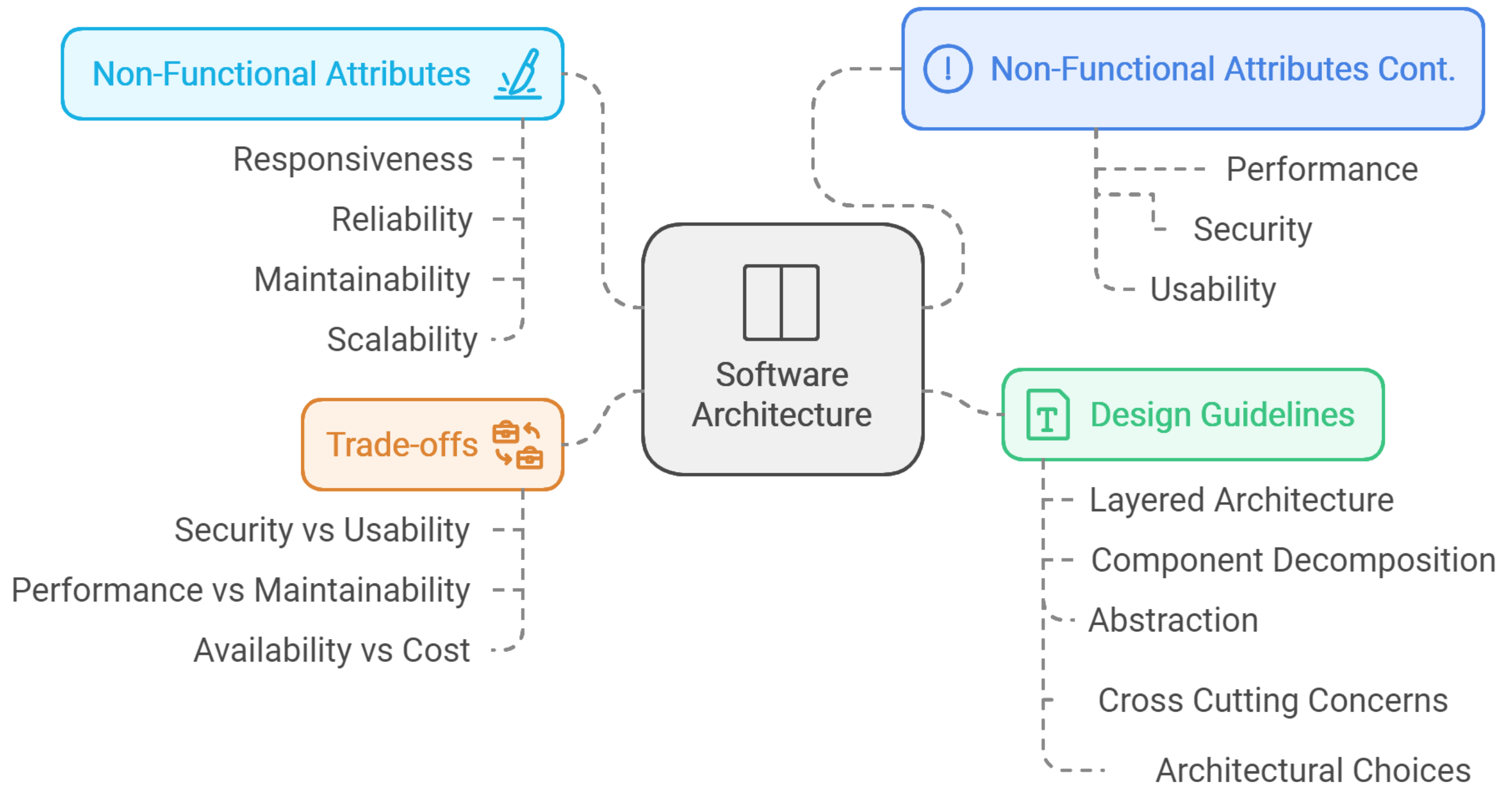
Software architecture

Youssef Ghatas

Acknowledgement

- Presentation is adapted from Ian Sommerville “Engineering Software Products”

Agenda



Software architecture

- To create a reliable, secure and efficient product, you need to pay attention to architectural design which includes:
 - its overall organization,
 - how the software is decomposed into components,
 - the server organization
 - the technologies that you use to build the software. The architecture of a software product affects its performance, usability, security, reliability and maintainability.
- There are many different interpretations of the term 'software architecture'.
 - Some focus on 'architecture' as a noun - the structure of a system and others consider 'architecture' to be a verb - the process of defining these structures.

The IEEE definition of software architecture

Architecture is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

YAGNI (You in't Gonna Need it): a principle relevant to XP and Agile in general.

Doesn't Architecture contradict with Agile?

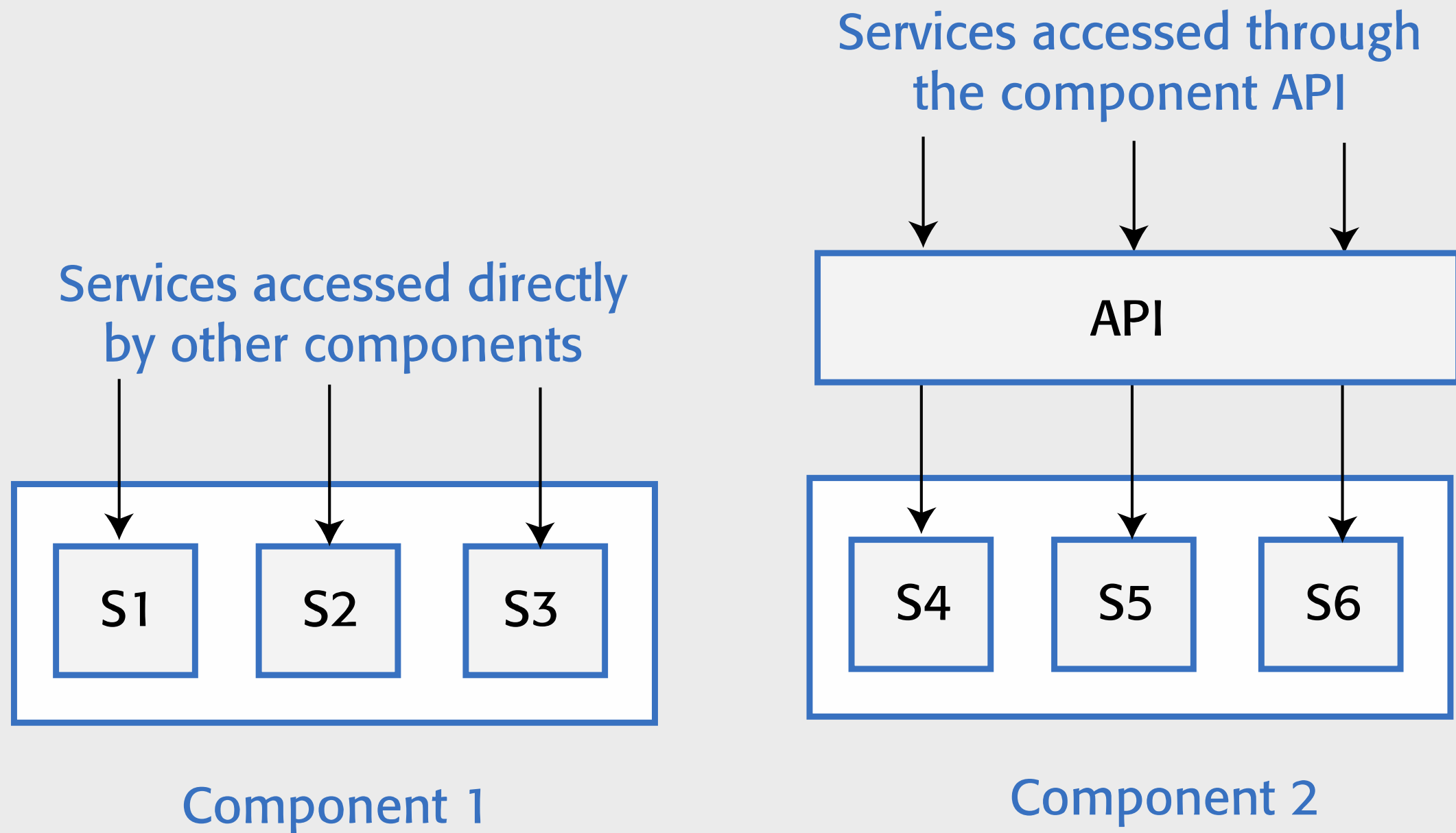
Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.

Antoine De St-Exupery

Software architecture and components

- A **component** is an element that implements a coherent set of functionality or features.
- Software component can be considered as a collection of one or more services that may be used by other components.
- **When designing software architecture, you don't have to decide how an architectural element or component is to be implemented.**
- Rather, you design the **component interface** and leave the implementation of that interface to a later stage of the development process.

Example: Access to services provided by software components



Why is architecture important?

- Architecture is important because the architecture of a system has a fundamental **influence on the non-functional** system properties.
- Architectural design involves understanding the issues that affect the architecture of your product and creating an architectural description that shows the **critical components and their relationships**.
- **Minimizing complexity** should be an important goal for architectural designers.
 - The more complex a system, the more difficult and expensive it is to understand and change.
 - Programmers are more likely to make mistakes and introduce bugs and security vulnerabilities when they are modifying or extending a complex system..

Non-functional system quality attributes



Non-functional system quality attributes

Responsiveness

Does the system return results to users in a reasonable time?

Reliability

Do the system features behave as expected (always) by both developers and users?

Availability

Can the system deliver its services when requested by users?

Security

Does the system protect itself and users' data from unauthorized attacks and intrusions?

Usability

Can system users access the features that they need and use them quickly and without errors?

Maintainability

Can the system be readily updated and new features added without undue costs?

Resilience

Can the system continue to deliver user services in the event of partial failure or external attack?

Example: Centralized vs Distribute security architectures

- The benefits of a centralized security architecture are that it is easier to design and build protection and that the protected information can be accessed more efficiently.
- However, if your security is breached, you lose everything.
- If you distribute information, it takes longer to access all of the information and costs more to protect it.
- If security is breached in one location, you only lose the information that you have stored there.

Architectural Design

- Understand the issues that affect the architecture of your product

Issues that influence architectural decisions

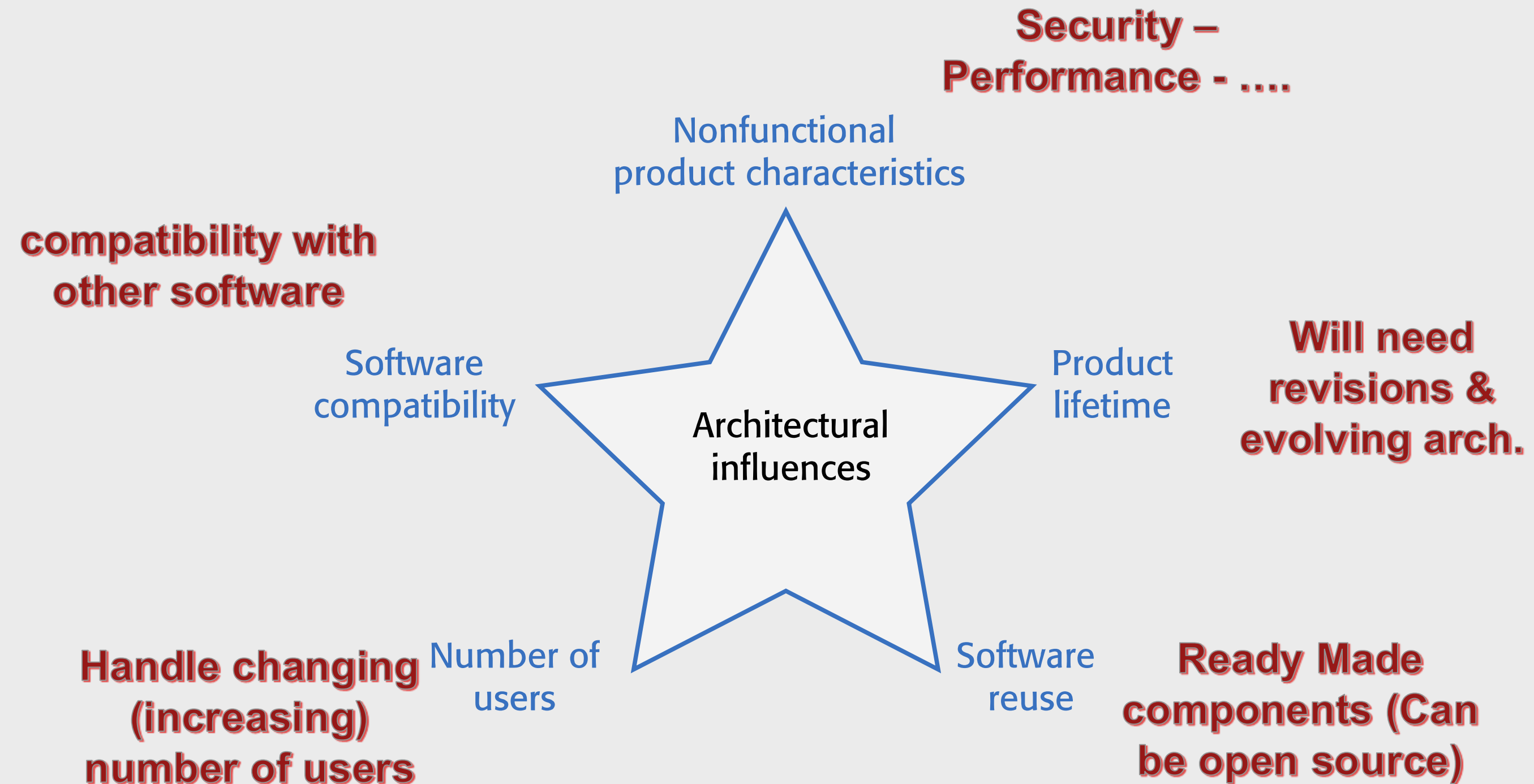


Table 4.4 The importance of architectural design issues

Nonfunctional product characteristics

Nonfunctional product characteristics such as security and performance affect all users. If you get these wrong, your product will be unlikely to be a commercial success.

Unfortunately, some characteristics are opposing, so you can only optimize the most important.

Product lifetime

If you anticipate a long product lifetime, you will need to create regular product revisions. You therefore need an architecture that is evolvable, so that it can be adapted to accommodate new features and technology.

Software reuse

You can save a lot of time and effort, if you can reuse large components from other products or open-source software. However, this constrains your architectural choices because you must fit your design around the software that is being reused.

Number of users

If you are developing consumer software delivered over the Internet, the number of users can change very quickly. This can lead to serious performance degradation unless you design your architecture so that your system can be quickly scaled up and down.

Software compatibility

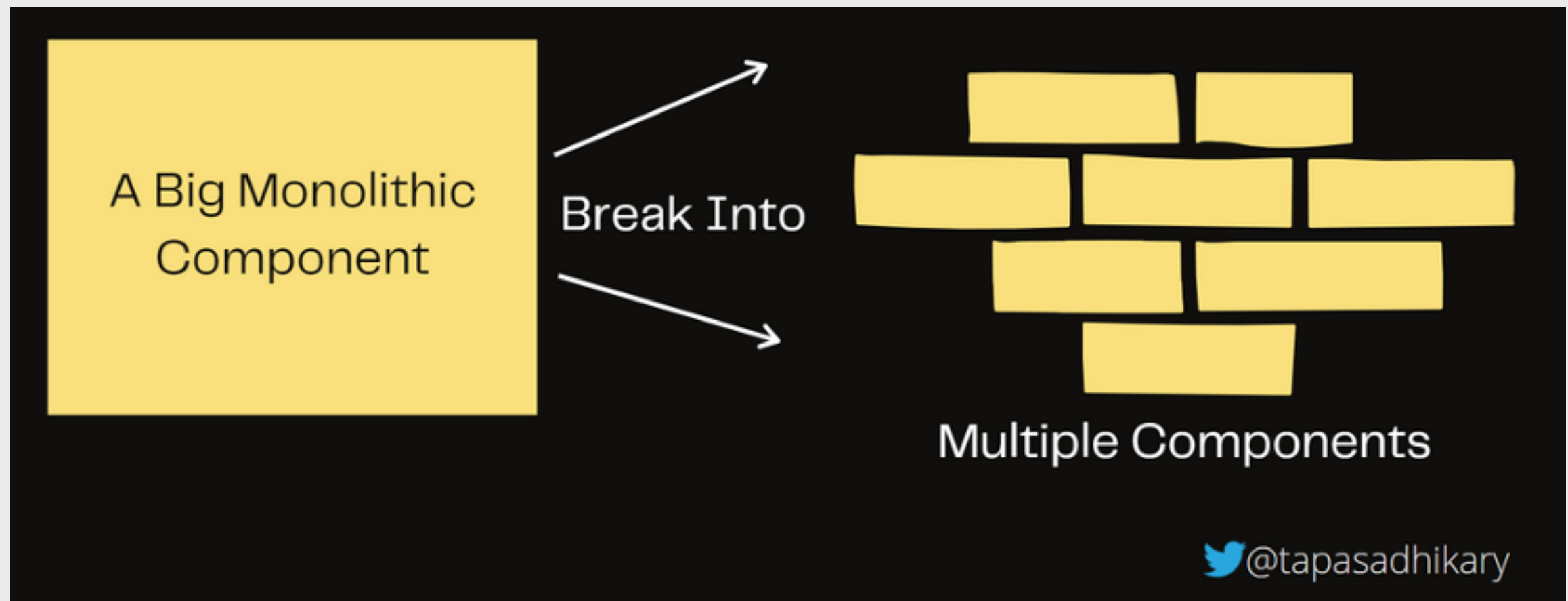
For some products, it is important to maintain compatibility with other software so that users can adopt your product and use data prepared using a different system. This may limit architectural choices, such as the database software that you can use.

Design: Choices & Tradeoffs

Trade off: Maintainability vs performance



Trade off: Maintainability vs performance

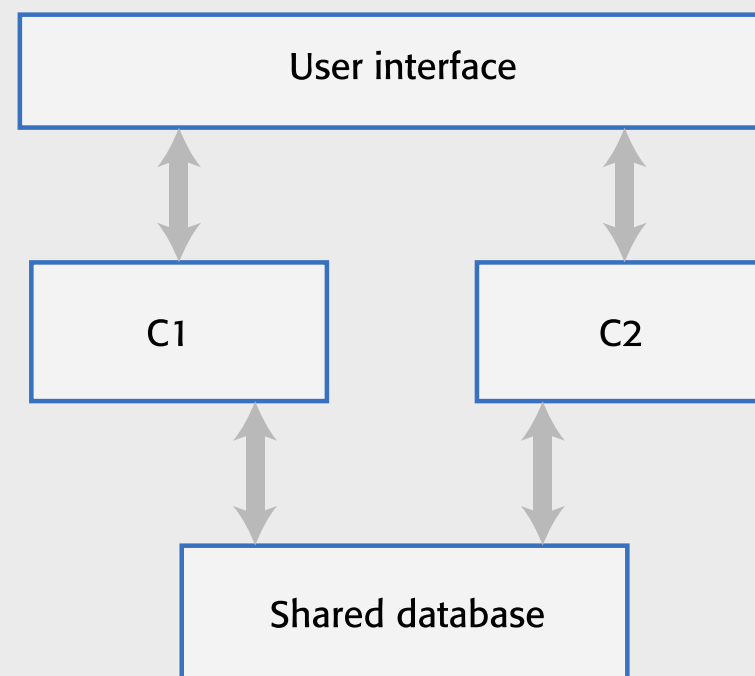


?

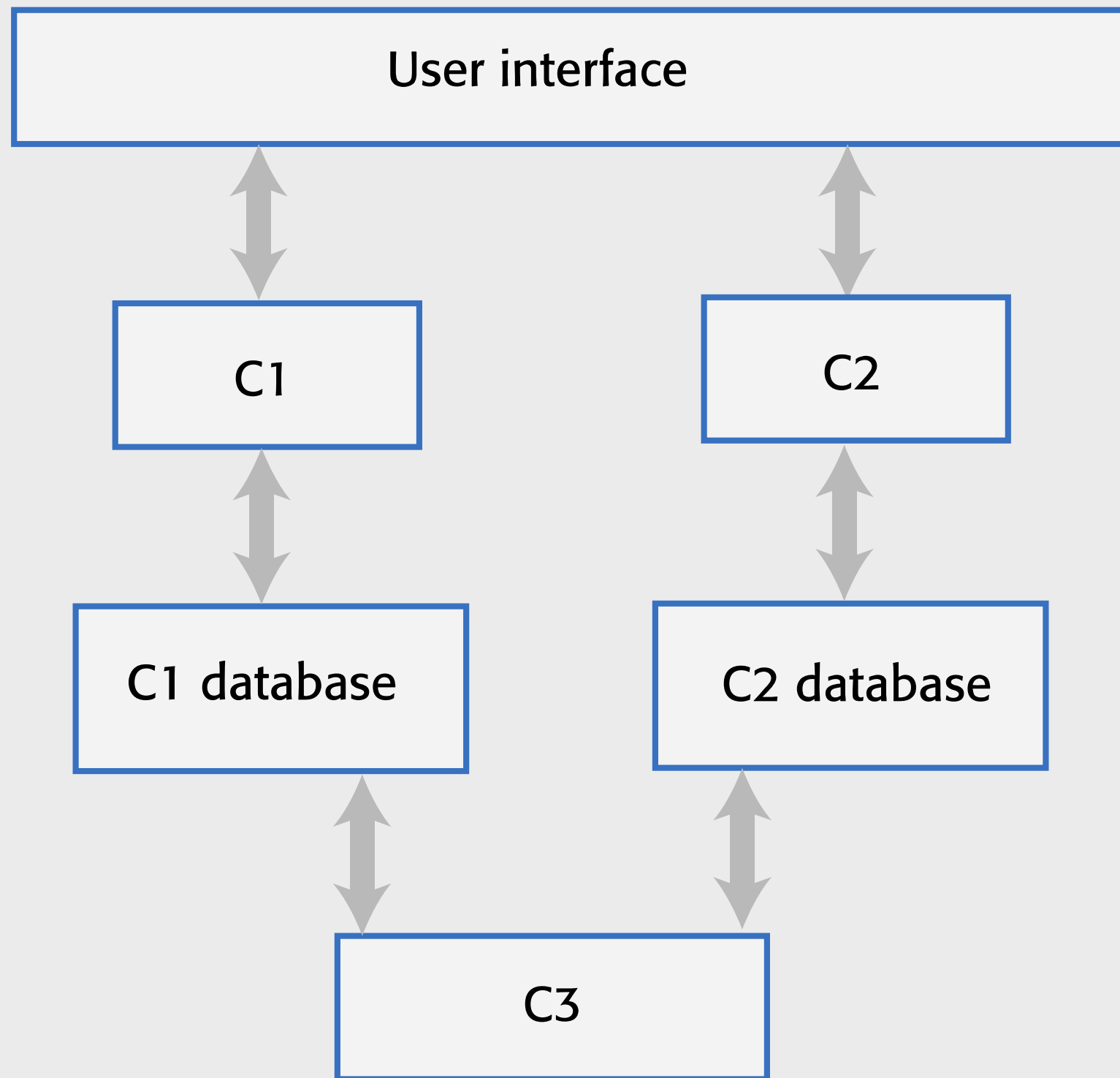
Maintainability and performance: Example

- A system with two components (C1 and C2) that share a common database.
 - What if C1 want to change the DB?

Shared database architecture



Alternative: Multiple database architecture



Database reconciliation

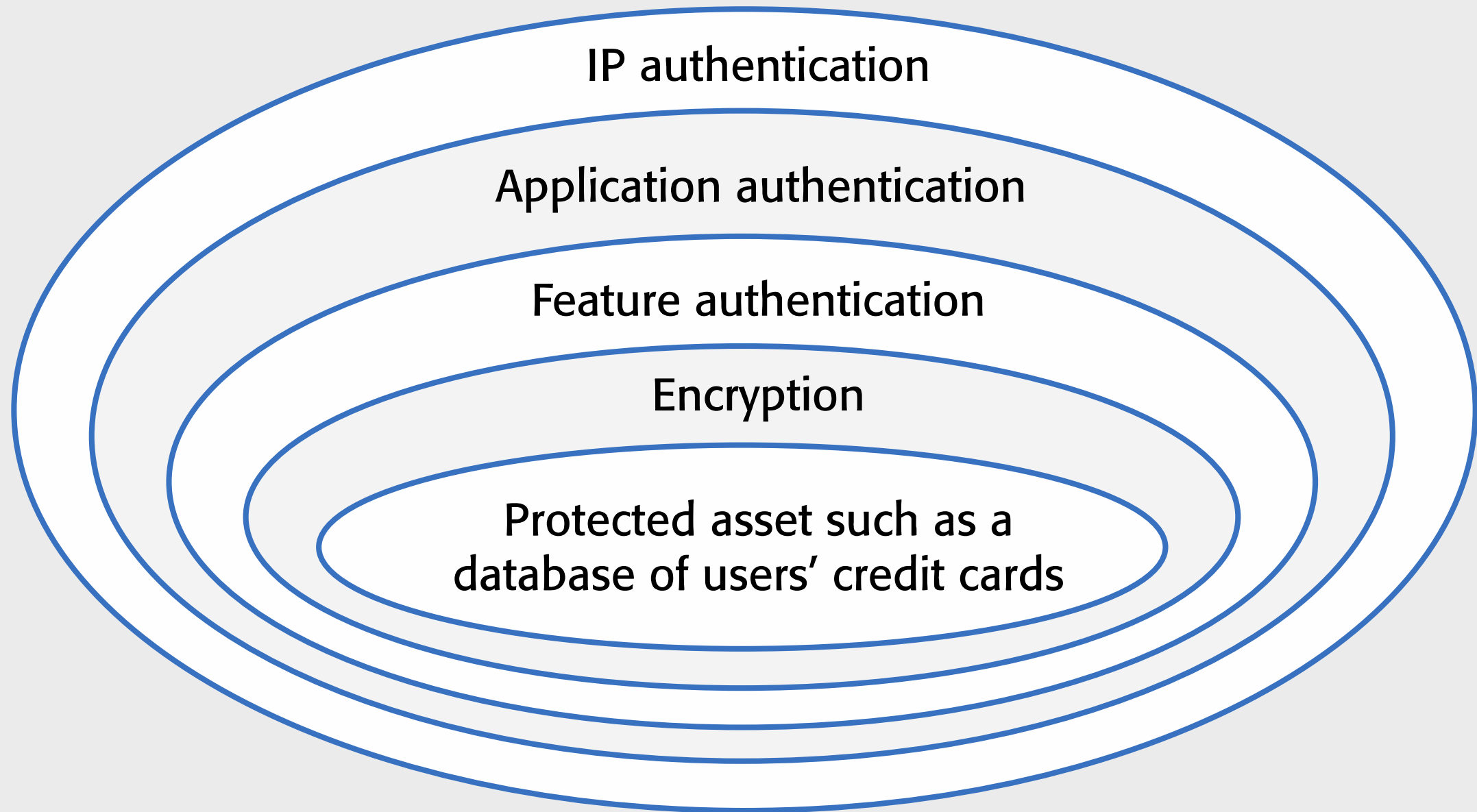
Maintainability and performance: Example

- In the previous Figure, a different architecture is used where each component has its own copy of the parts of the database that it needs.
 - If one component needs to change the database organization, this does not affect the other component.
- However, a multi-database architecture: may run **more slowly** and may **cost more** to implement and change.
 - A multi-database architecture needs a mechanism (component C3) to ensure that the data shared by C1 and C2 is kept consistent when it is changed.

Trade off: Security vs usability

- You can achieve security by designing the system protection as a **series of layers**.
 - An attacker has to penetrate all of those layers before the system is compromised.
- Layers might include system authentication layers, a separate critical feature authentication layer, an encryption layer and so on.
- Architecturally, you can implement each of these layers as separate components so that if one of these components is compromised by an attacker, then the other layers remain intact.

Authentication layers



Usability issues

- A layered approach to security affects the usability of the software.
 - Users have to remember information, like passwords, that is needed to penetrate a security layer. Their interaction with the system is inevitably slowed down by its security features.
 - Many users find this irritating and often look for work-arounds so that they do not have to re-authenticate to access system features or data.
- To avoid this, you need an architecture:
 - that doesn't have too many security layers,
 - that doesn't enforce unnecessary security,
 - that provides helper components that reduce the load on users.

Trade off: Availability vs time-to-market

- Availability is particularly important in enterprise products, such as products for the finance industry, where 24/7 operation is expected.
- The availability of a system is a measure of the amount of 'uptime' of that system.
 - Availability is normally expressed as a percentage of the time that a system is available to deliver user services.
- Architecturally, you achieve availability by having redundant components in a system.
 - To make use of redundancy, you include sensor components that detect failure, and switching components that switch operation to a redundant component when a failure is detected.
- Implementing extra components takes time and increases the cost of system development. It adds complexity to the system and therefore increases the chances of introducing bugs and vulnerabilities.

Architectural design questions

- How should the system be organized as a set of architectural components, where each of these components provides a subset of the overall system functionality?
 - The organization should deliver the system security, reliability and performance that you need.
- How should these architectural components be distributed and communicate with each other?
- What technologies should you use in building the system and what components should be reused?

Component organization

- Abstraction in software design means that you focus on the essential elements of a system or software component without concern for its details.
 - Architectural Level → Large-scale Components
 - Decomposition → From Large-scale components to finer-grain components
- At the architectural level, your concern should be on large-scale architectural components.
- Decomposition involves analysing these large-scale components and representing them as a set of finer-grain components.
- Layered models are often used to illustrate how a system is composed of components.

Reference Terminology

- Service: coherent unit of functionality
- Component: Offers one or more services to other components or end-users.
- Module: Set of components

An architectural model of a document retrieval system

Web browser

User interaction	Local input validation	Local printing
------------------	------------------------	----------------

User interface management

Authentication and authorization	Form and query manager	Web page generation
----------------------------------	------------------------	---------------------

Information retrieval

Search	Document retrieval	Rights management	Payments	Accounting
--------	--------------------	-------------------	----------	------------

Document index

Index management	Index querying	Index creation
------------------	----------------	----------------

Basic services

Database query	Query validation	Logging	User account management
----------------	------------------	---------	-------------------------

Databases

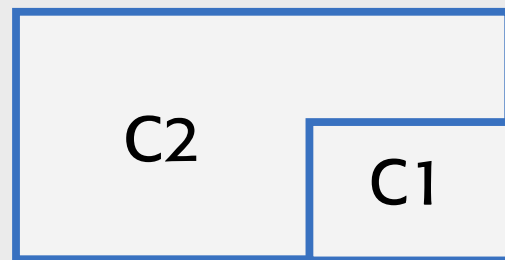
DB1	DB2	DB3	DB4	DB5
-----	-----	-----	-----	-----

Architectural complexity

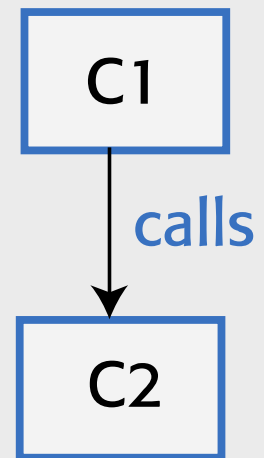
- Complexity in a system architecture arises because of the number and the nature of the relationships between components in that system.
- When decomposing a system into components, you should try to avoid unnecessary software complexity.
 - **Localize relationships**
If there are relationships between components A and B, these are easier to understand if A and B are defined in the same module.
 - **Reduce shared dependencies**
Where components A and B depend on some other component or data, complexity increases because changes to the shared component mean you have to understand how these changes affect both A and B.
- It is always preferable to use local data wherever possible and to avoid sharing data if you can.

Examples of component relationships

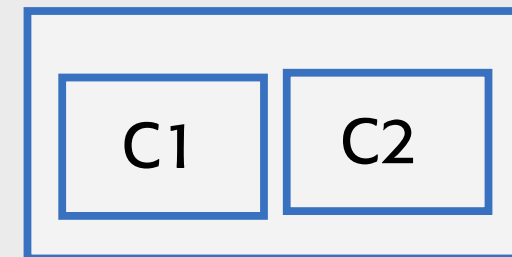
C1 is-part-of C2



C1 uses C2



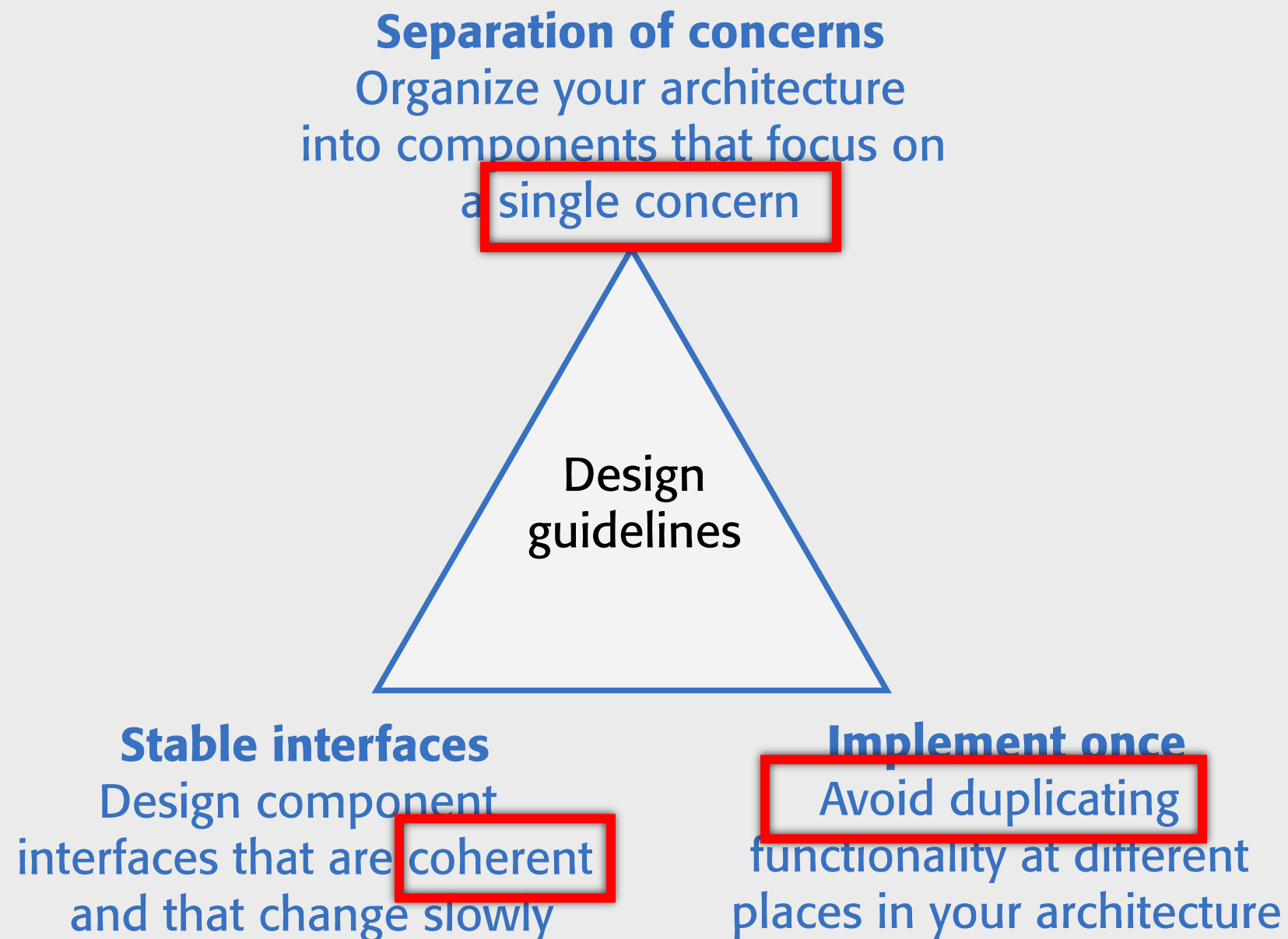
C1 is-located-with C2



C1 shares-data-with C2



Architectural design guidelines (help controlling complexity)



Design guidelines and layered architectures

- Each layer is an area of concern and is considered separately from other layers.
 - The top layer is concerned with user interaction, the next layer down with user interface management, the third layer with information retrieval and so on.
- Within each layer, the components are independent and do not overlap in functionality.
 - The lower layers include components that provide general functionality so there is no need to replicate this in the components in a higher level.
- The architectural model is a high-level model that does not include implementation information.

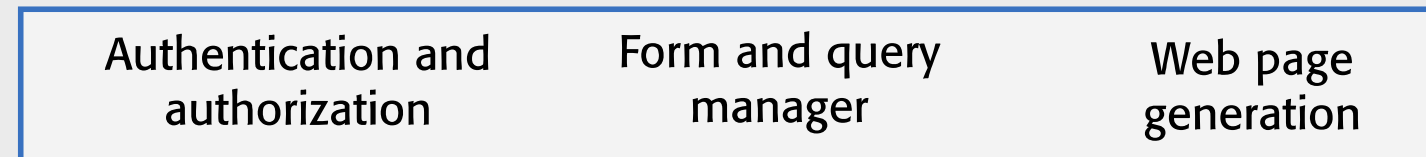
Why redundant ?

An architectural model of a document retrieval system

Web browser



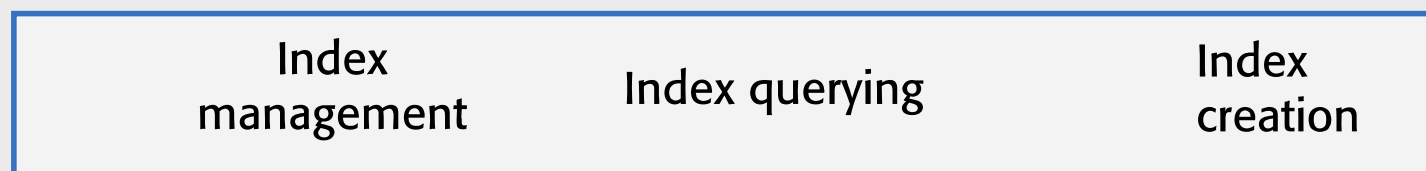
User interface management



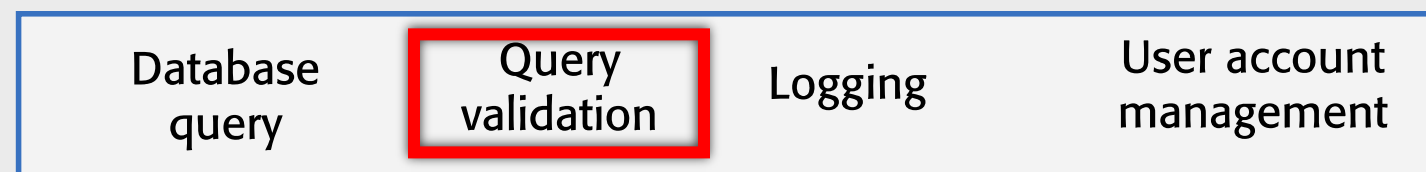
Information retrieval



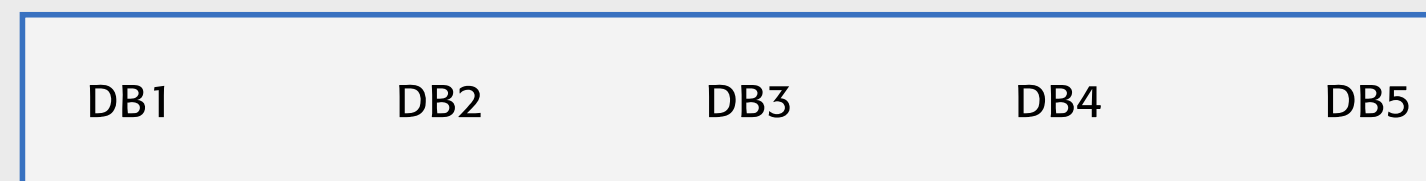
Document index



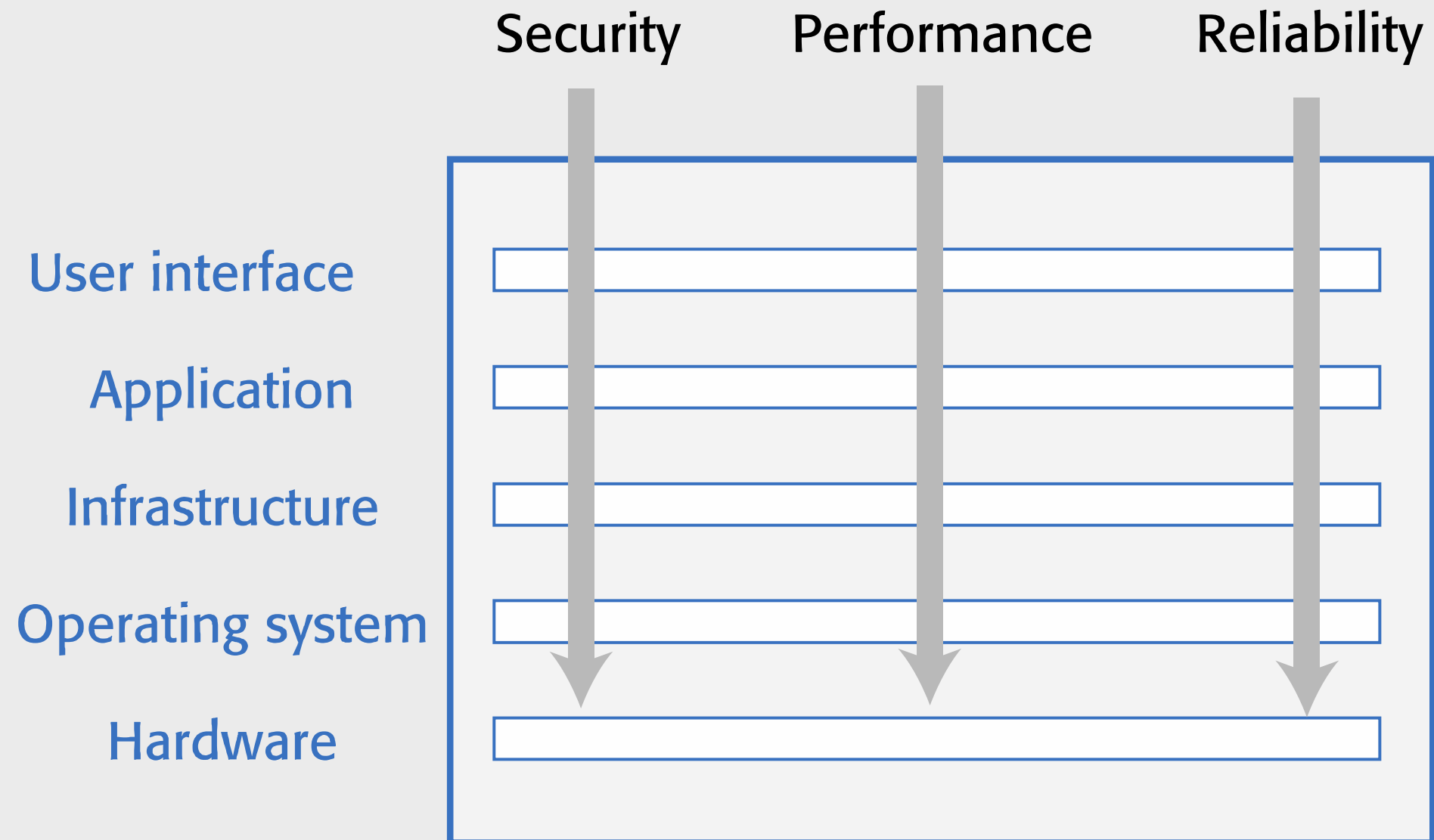
Basic services



Databases



Cross-cutting concerns



Cross-cutting concerns

- Cross-cutting concerns are concerns that are systemic, that is, they **affect the whole system.**
- In a layered architecture, cross-cutting concerns affect all layers in the system as well as the way in which people use the system.
- **Cross-cutting concerns are completely different from the functional concerns represented** by layers in a software architecture.
- Every layer has to take them into account and there are inevitably interactions between the layers because of these concerns.
- The existence of cross-cutting concerns is the reason why modifying a system after it has been designed to **improve its security is often difficult.**

Security as a cross-cutting concern

Security architecture

Different technologies are used in different layers, such as an SQL database or a Firefox browser. Attackers can try to use of vulnerabilities in these technologies to gain access.

Consequently, you need protection from attacks at each layer as well as protection, at lower layers in the system, from successful attacks that have occurred at higher-level layers.

By distributing security across the layers, your system is more resilient to attacks and software failure

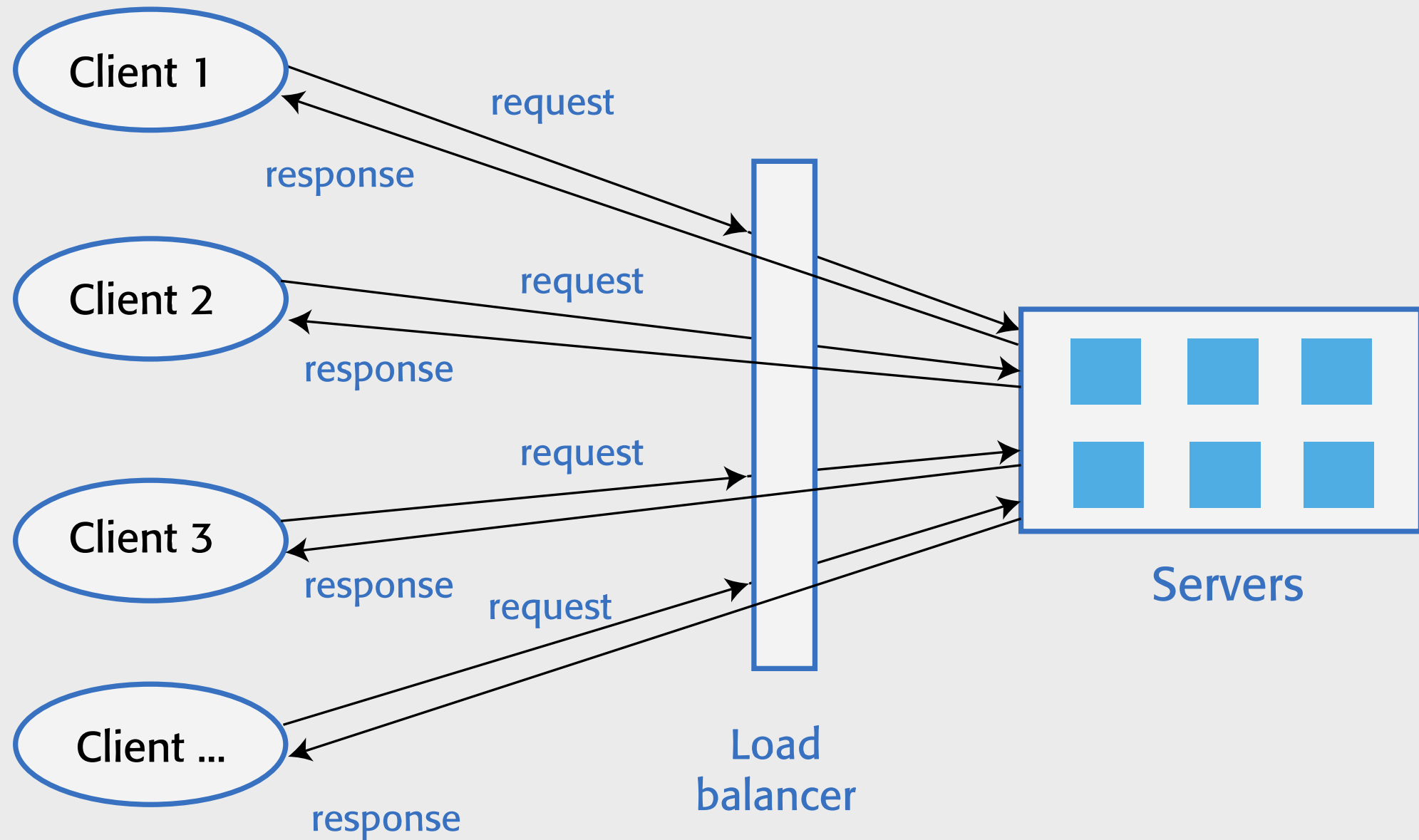
From layers to smaller components (decomposition)

- Software Architect or with the team?
 - Pros & Cons?
- Process:
 - Brain-Storming and Trial-and-error

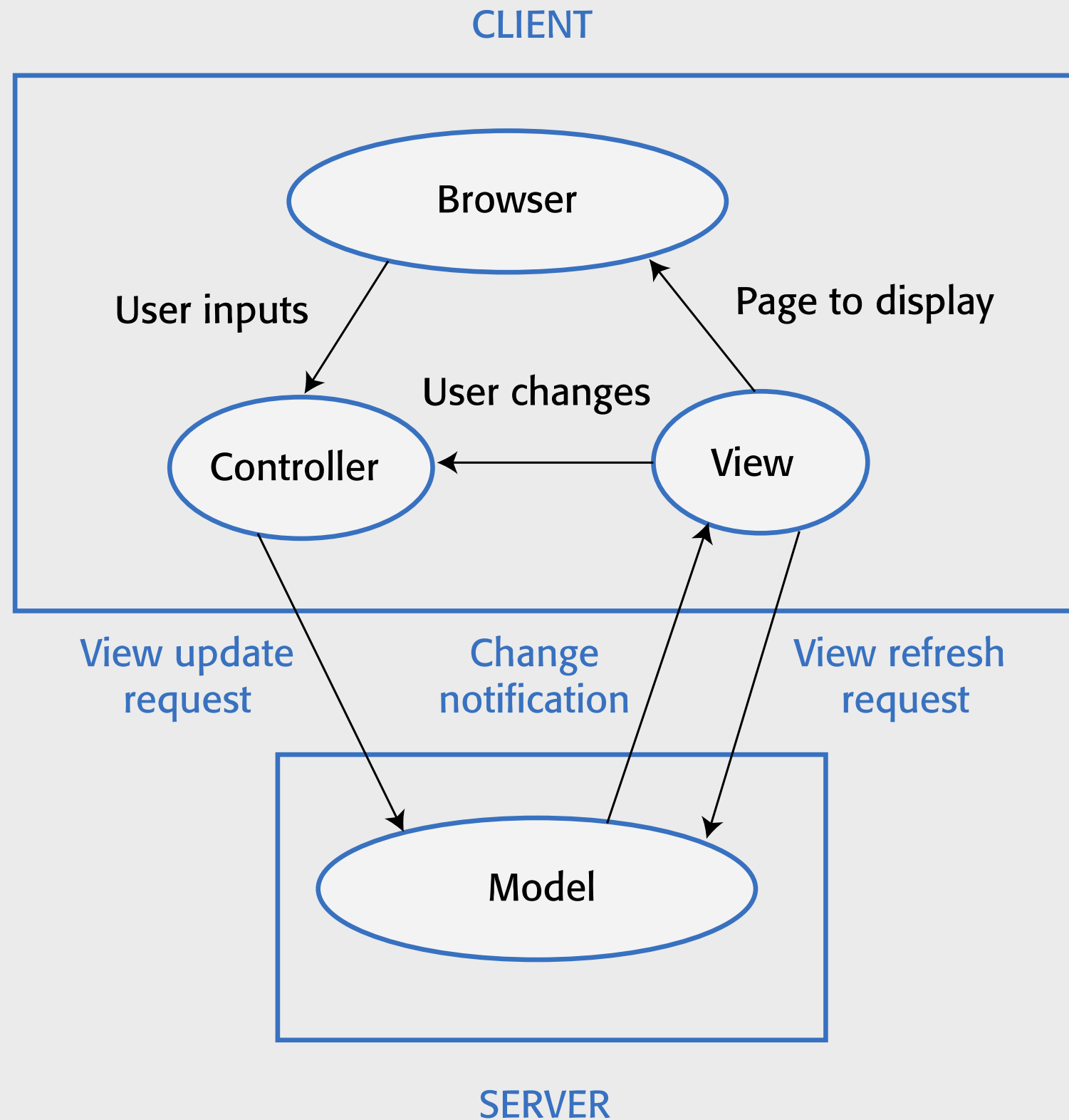
Distribution architecture

- The distribution architecture of a software system defines the servers in the system and the allocation of components to these servers.
- Client-server architectures are a type of distribution architecture that is suited to applications where clients access a shared database and business logic operations on that data.
- In this architecture, the user interface is implemented on the user's own computer or mobile device.
 - Functionality is distributed between the client and one or more server computers.

Client-server architecture



The model-view-controller pattern



Client-server communication

- Client-server communication normally uses the HTTP protocol.
 - The client sends a message to the server that includes an instruction such as **GET** or **POST** along with the **identifier of a resource** (usually a URL) on which that instruction should operate. The message may also include additional information, such as information collected from a form.
- Two ways of representing data are widely used, namely **XML** and **JSON**.
 - XML is a markup language with tags used to identify each data item.
 - JSON is a simpler representation based on the representation of objects in the Javascript language.

Client-server communication

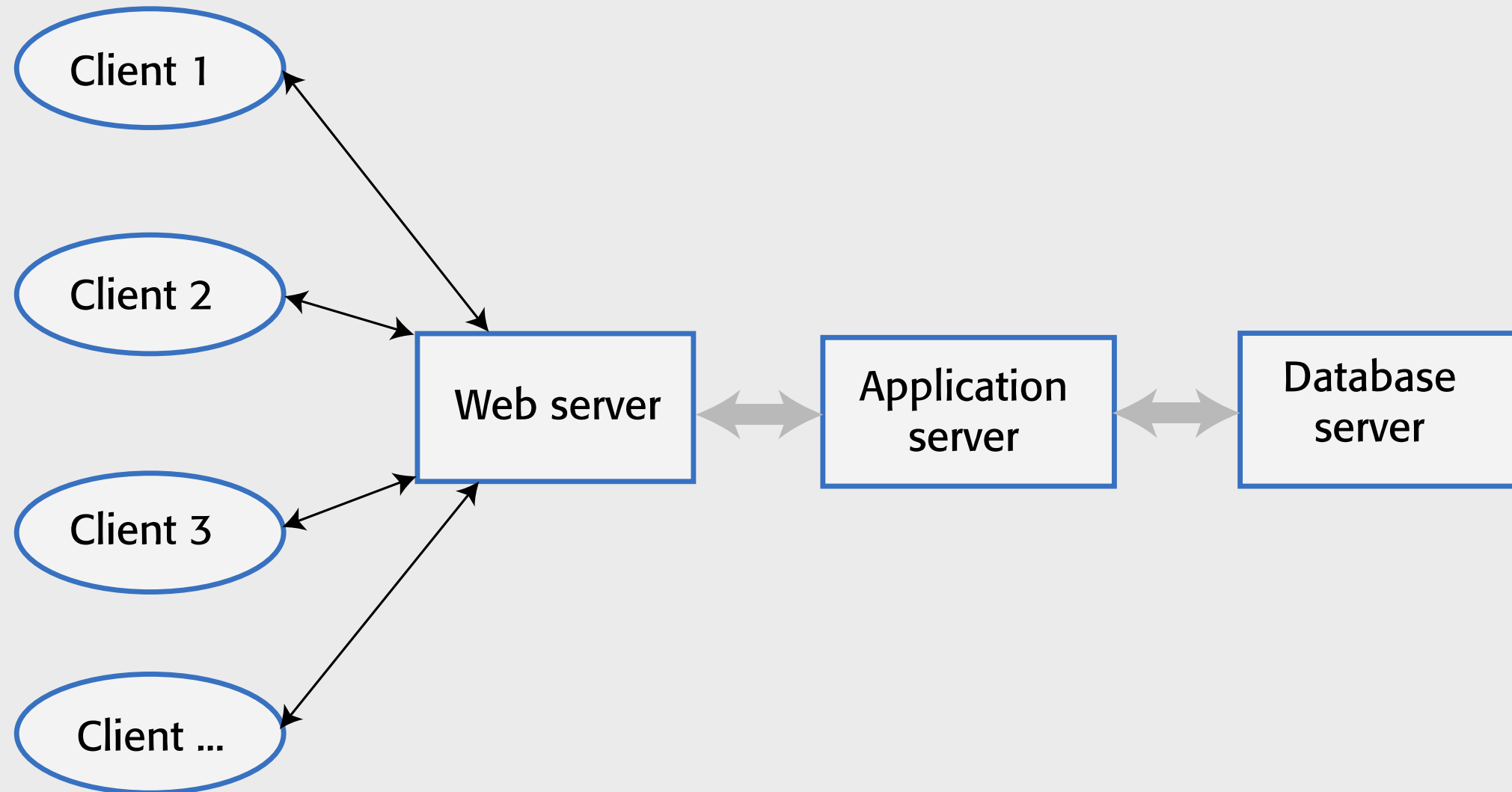
```
{  
  "response": {  
    "status": 200,  
    "message": "Success",  
    "data": {  
      "name": "John Doe",  
      "email": "john@example.com"  
    }  
  }  
}
```

JSON

```
<response>  
  <status>200</status>  
  <message>Success</message>  
  <data>  
    <name>John Doe</name>  
    <email>john@example.com</email>  
  </data>  
</response>
```

XML

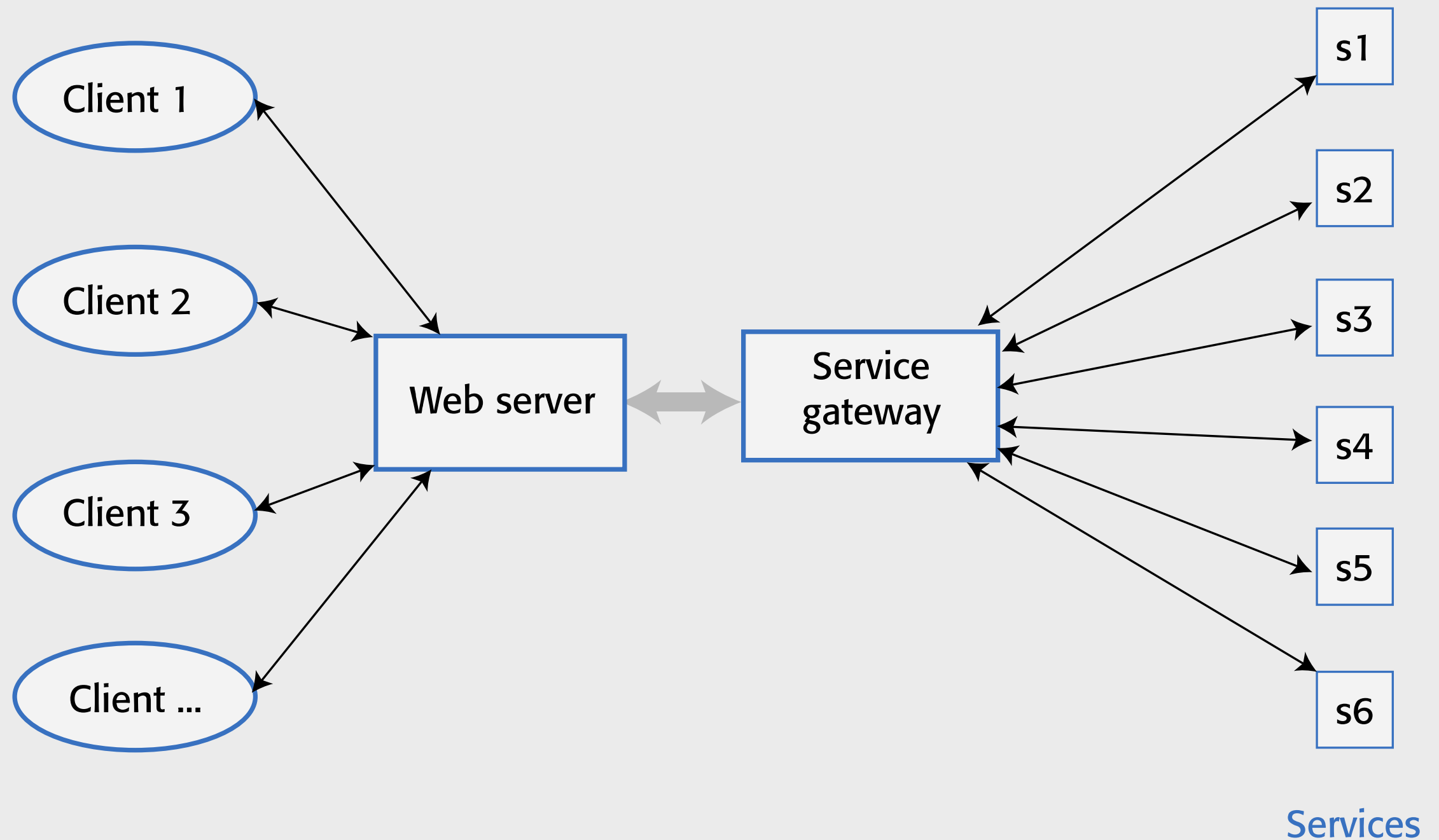
Multi-tier client-server architecture



Service-oriented architecture

- Services in a service-oriented architecture are stateless components, which means that they can be replicated and can migrate from one computer to another.
- Many servers may be involved in providing services
- A service-oriented architecture is usually easier to scale as demand increases and is resilient to failure.

Service-oriented architecture



Issues in architectural choice

- **Data type and data updates**

- If you are mostly using structured data that may be updated by different system features, it is usually best to have a single shared database that provides locking and transaction management.
- If data is distributed across services, you need a way to keep it consistent and this adds overhead to your system.

- **Change frequency**

- If you anticipate that system components will be regularly changed or replaced, then isolating these components as separate services simplifies those changes.

- **The system execution platform**

- If you plan to run your system on the cloud with TOO MANY users accessing it over the Internet, it is maybe better to implement it as a service-oriented architecture because scaling the system is simpler.
- If your product is a business system that runs on local servers, a multi-tier architecture may be more appropriate.

Technology choices

Database

Should you use a relational SQL database or an unstructured NOSQL database?

Platform

Should you deliver your product on a mobile app and/or a web platform?

Server

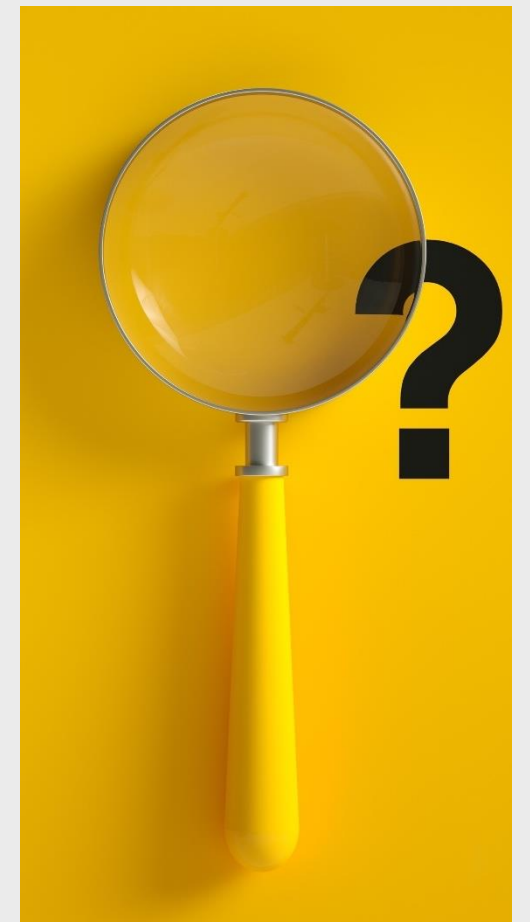
Should you use dedicated in-house servers or design your system to run on a public cloud? If a public cloud, should you use Amazon, Google, Microsoft, or some other option?

Open source

Are there suitable open-source components that you could incorporate into your products?

Development tools

Do your development tools embed architectural assumptions about the software being developed that limit your architectural choices?



Database

- There are two kinds of database that are now commonly used:
 - Relational databases, where the data is organised into structured tables
 - NoSQL databases, in which the data has a more flexible, user-defined organization.
- Relational databases, such as MySQL, are particularly suitable for situations where you need transaction management and the data structures are predictable and fairly simple.
- NoSQL databases, such as MongoDB, are more flexible and potentially more efficient than relational databases for data analysis.
 - NoSQL databases allow data to be organized hierarchically rather than as flat tables and this allows for more efficient concurrent processing of 'big data'.

Delivery platform

- Delivery can be as a web-based or a mobile product or both
- Mobile issues:
 - **Intermittent connectivity** You must be able to provide a limited service without network connectivity.
 - **Processor power** Mobile devices have less powerful processors, so you need to minimize computationally-intensive operations.
 - **Power management** Mobile battery life is limited so you should try to minimize the power used by your application.
 - **On-screen keyboard** On-screen keyboards are slow and error-prone. You should minimize input using the screen keyboard to reduce user frustration.
- To deal with these differences, you usually need separate browser-based and mobile versions of your product front-end.
 - You may need a completely different decomposition architecture in these different versions to ensure that performance and other characteristics are maintained.

Server

- A key decision that you have to make is whether to design your system to run on customer servers or to run on the cloud.
- For consumer products that are not simply mobile apps it maybe better to develop for the cloud.
- For business products, it is a more difficult decision.
 - Some businesses are concerned about cloud security and prefer to run their systems on in-house servers. They may have a predictable pattern of system usage so there is less need to design your system to cope with large changes in demand.
- An important choice you have to make if you are running your software on the cloud is which cloud provider to use.

Open source

- Open source software is software that is available freely, which you can change and modify as you wish.
 - The advantage is that you can reuse rather than implement new software, which reduces development costs and time to market.
 - The disadvantages of using open-source software is that you are constrained by that software and have no control over its evolution.
- Depends on the availability, maturity and continuing support.
- license.
- Your choice of open source software should depend on the type of product that you are developing, your target market and the expertise of your development team.

Development tools

- Development technologies, such as a mobile development toolkit or a web application framework, influence the architecture of your software.
 - These technologies have built-in assumptions about system architectures and you have to conform to these assumptions to use the development system.
- The development technology that you use may also have an indirect influence on the system architecture.
 - Developers usually favour architectural choices that use familiar technologies that they understand. For example, if your team have a lot of experience of relational databases, they may argue for this instead of a NoSQL database.

Question 1

You are designing a database system for an e-commerce platform. The system needs to handle a high volume of transactions for processing orders, managing inventory, and tracking customer interactions. Which type of database would be more appropriate for this scenario:

1. SQL
2. NoSQL

Question 2

You are developing a social media analytics platform that collects and analyzes large volumes of unstructured data from various sources, including text, images, and videos. The system needs to be able to scale horizontally to accommodate increasing data volumes and perform complex analytical queries in real-time. Should you choose SQL or NoSQL database for this application?

1. SQL
2. NoSQL

Question 3

You are tasked with developing a sensor data monitoring system for a smart city project. The system will collect and analyze data from various IoT devices deployed across the city, including temperature sensors, traffic cameras, and air quality monitors. Given the need for scalability, flexibility, and fast data ingestion, would you recommend using an SQL or NoSQL database for this project?

1. SQL
2. NoSQL

Questions

1.Question: In designing a database system for a financial institution's customer management platform, which concept should be prioritized?

1. A) Maintainability
2. B) Security

2.Question: When developing a healthcare information system for a hospital network, which aspect should be given more emphasis?

1. A) Availability
2. B) Maintainability

3.Question: Which aspect should be prioritized for a real-time online gaming platform?

1. A) Performance
2. B) Availability

Key points 1

- Software architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.
- The architecture of a software system has a significant influence on non-functional system properties such as reliability, efficiency and security.
- Architectural design involves understanding the issues that are critical for your product and creating system descriptions that shows components and their relationships.
- The principal role of architectural descriptions is to provide a basis for the development team to discuss the system organization. Informal architectural diagrams are effective in architectural description because they are fast and easy to draw and share.
- System decomposition involves analyzing architectural components and representing them as a set of finer-grain components.

Key points 2

- To minimize complexity, you should separate concerns, avoid functional duplication and focus on component interfaces.
- Web-based systems often have a common layered structure including user interface layers, application-specific layers and a database layer.
- The distribution architecture in a system defines the organization of the servers in that system and the allocation of components to these servers.
- Multi-tier client-server and service-oriented architectures are the most commonly used architectures for web-based systems.
- Making decisions on technologies such as database and cloud technologies are an important part of the architectural design process.