軟體工程
(Software Engineering)
軟體架構：
架構設計、系統分解、分散式架構
(Software Architecture: Architectural design, System decomposition, and Distribution architecture)

1101SE05
MBA, IM, NTPU (M6131) (Fall 2021)
Thu 11, 12, 13 (19:25-22:10) (209)

Min-Yuh Day
戴敏育
Associate Professor
副教授
Institute of Information Management, National Taipei University
國立臺北大學 資訊管理研究所
https://web.ntpu.edu.tw/~myday

2021-10-28

課程大綱 (Syllabus)

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
1	2021/09/23	軟體工程概論 (Introduction to Software Engineering)
2	2021/09/30	軟體產品與專案管理：軟體產品管理，原型設計 (Software Products and Project Management: Software product management and prototyping)
3	2021/10/07	敏捷軟體工程：敏捷方法、Scrum、極限程式設計 (Agile Software Engineering: Agile methods, Scrum, and Extreme Programming)
4	2021/10/14	功能、場景和故事 (Features, Scenarios, and Stories)
5	2021/10/21	軟體工程個案研究 I (Case Study on Software Engineering I)
6	2021/10/28	軟體架構：架構設計、系統分解、分散式架構 (Software Architecture: Architectural design, System decomposition, and Distribution architecture)

課程大綱 (Syllabus)

國立臺北大學 National Taipei University

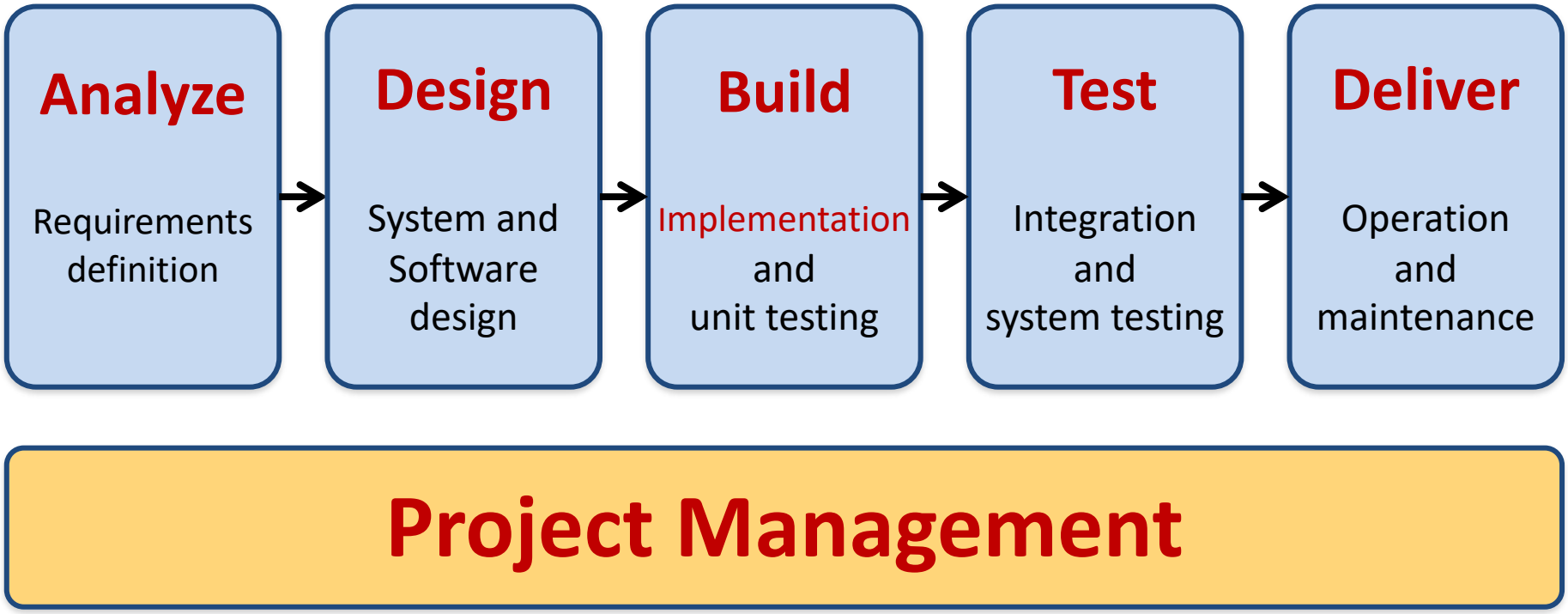
週次 (Week)	日期 (Date)	內容 (Subject/Topics)
7	2021/11/04	基於雲的軟體：虛擬化和容器、軟體即服務 (Cloud-Based Software: Virtualization and containers, Everything as a service, Software as a service)
8	2021/11/11	期中報告 (Midterm Project Report)
9	2021/11/18	雲端運算與雲軟體架構 (Cloud Computing and Cloud Software Architecture)
10	2021/11/25	微服務架構：RESTful服務、服務部署 (Microservices Architecture, RESTful services, Service deployment)
11	2021/12/02	軟體工程產業實務 (Industry Practices of Software Engineering)
12	2021/12/09	軟體工程個案研究 II (Case Study on Software Engineering II)

課程大綱 (Syllabus)

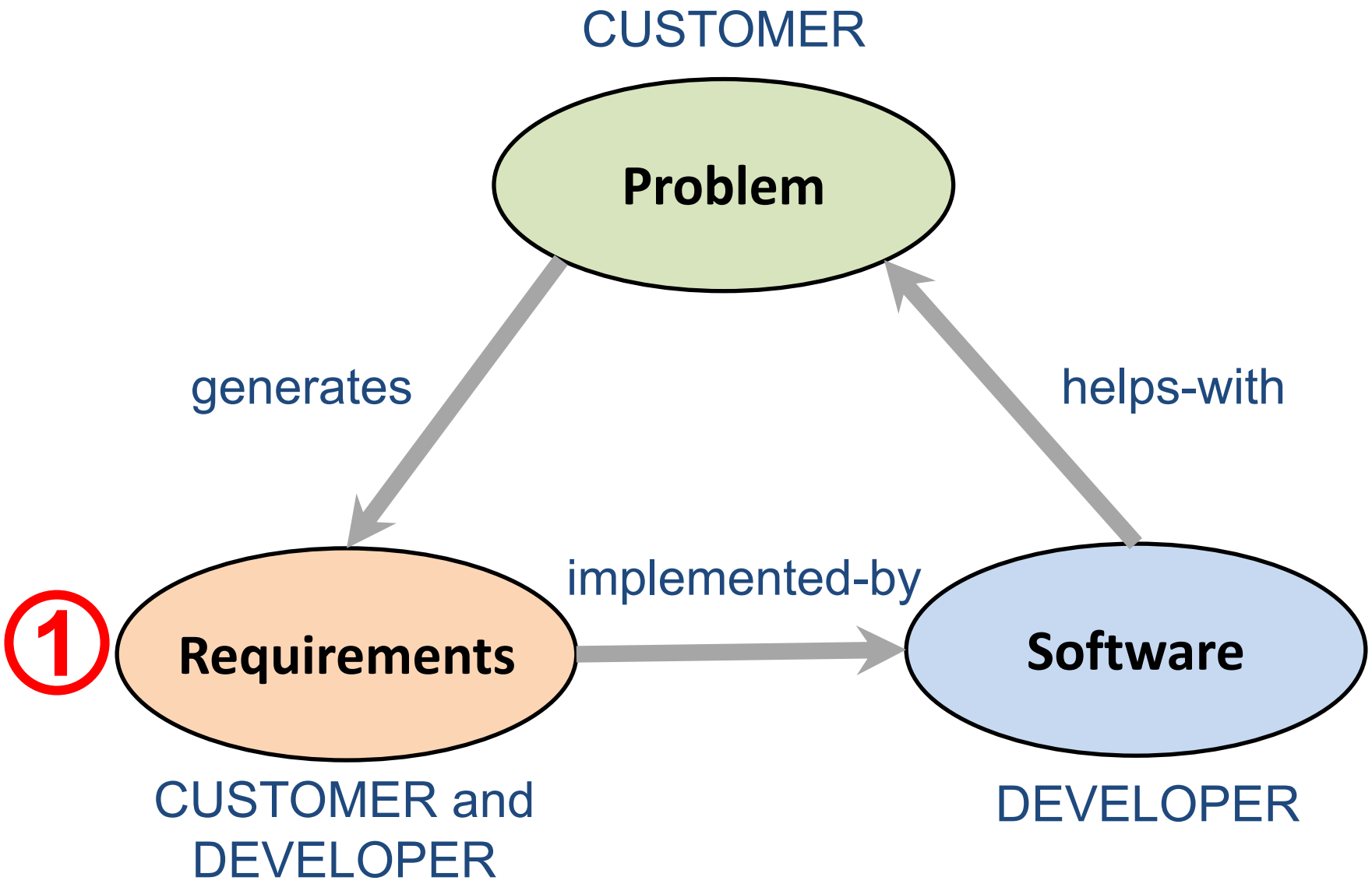
國立臺北大學 National Taipei University

週次 (Week)	日期 (Date)	內容 (Subject/Topics)
13	2021/12/16	安全和隱私 (Security and Privacy); 可靠的程式設計 (Reliable Programming)
14	2021/12/23	測試：功能測試、測試自動化、 測試驅動的開發、程式碼審查 (Testing: Functional testing, Test automation, Test-driven development, and Code reviews); DevOps和程式碼管理：程式碼管理和DevOps自動化 (DevOps and Code Management: Code management and DevOps automation)
15	2021/12/30	期末報告 I (Final Project Report I)
16	2022/01/06	期末報告 II (Final Project Report II)
17	2022/01/13	學生自主學習 (Self-learning)
18	2022/01/20	學生自主學習 (Self-learning)

Software Engineering and Project Management

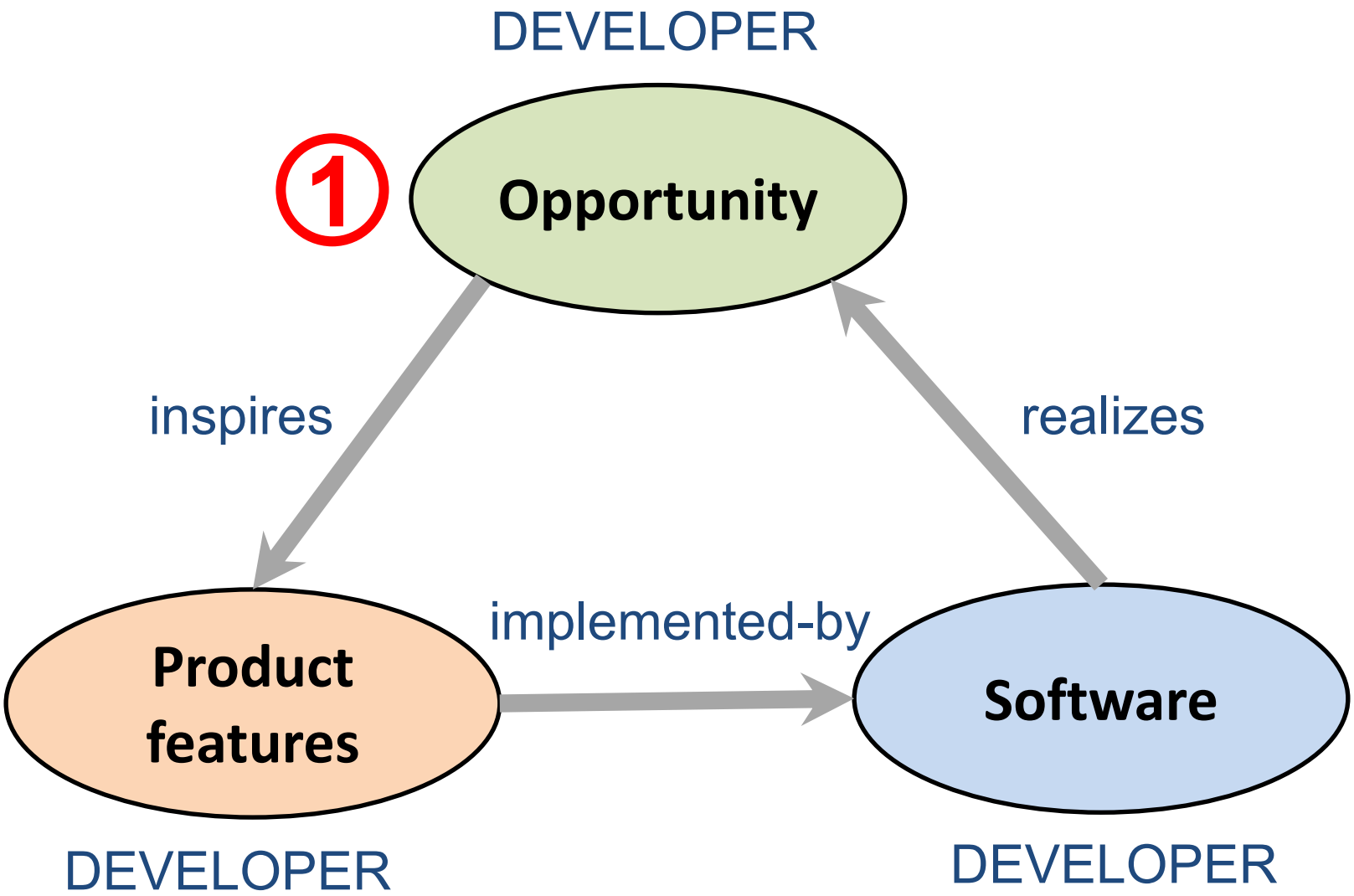


Project-based software engineering



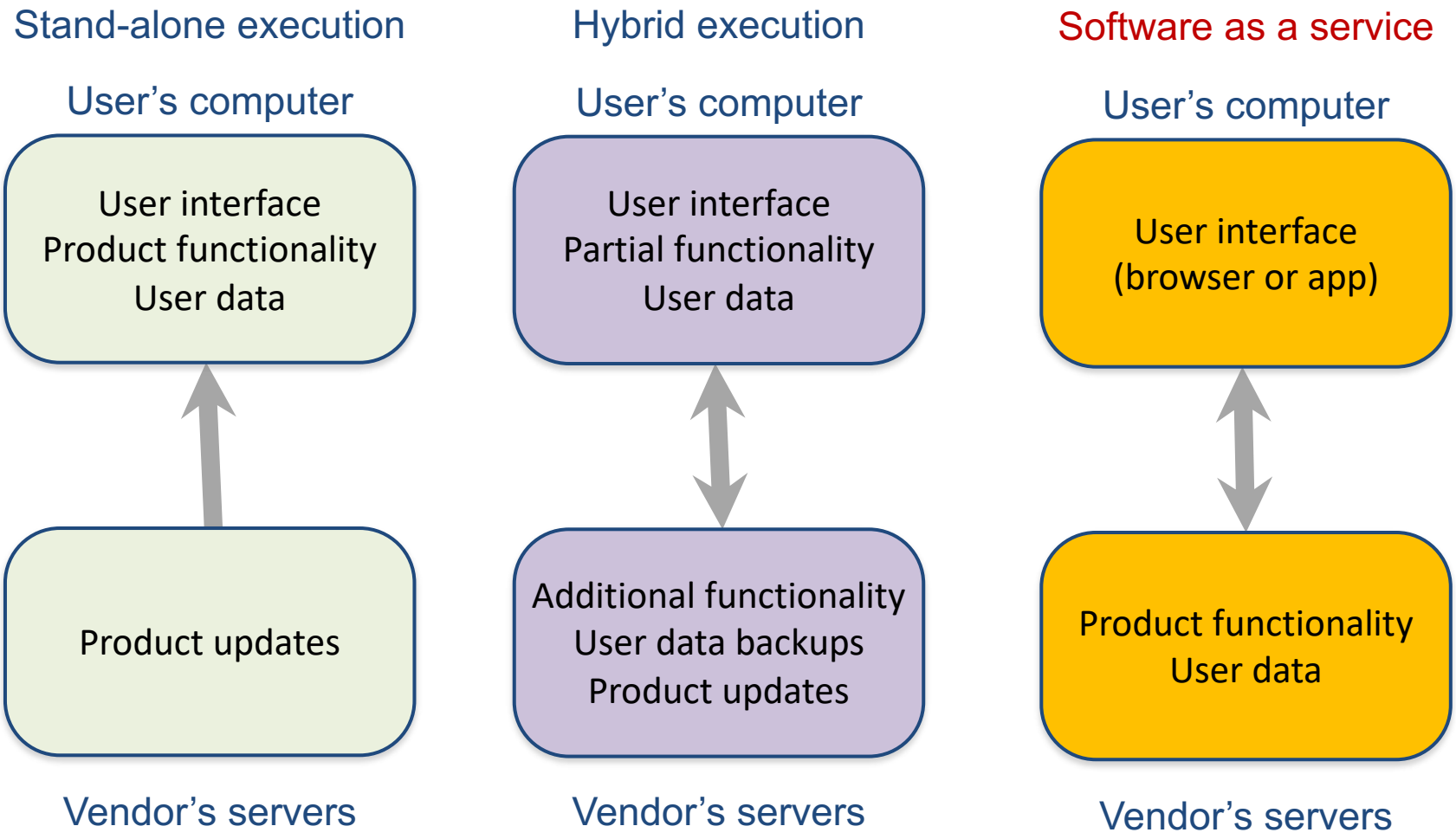
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Product software engineering



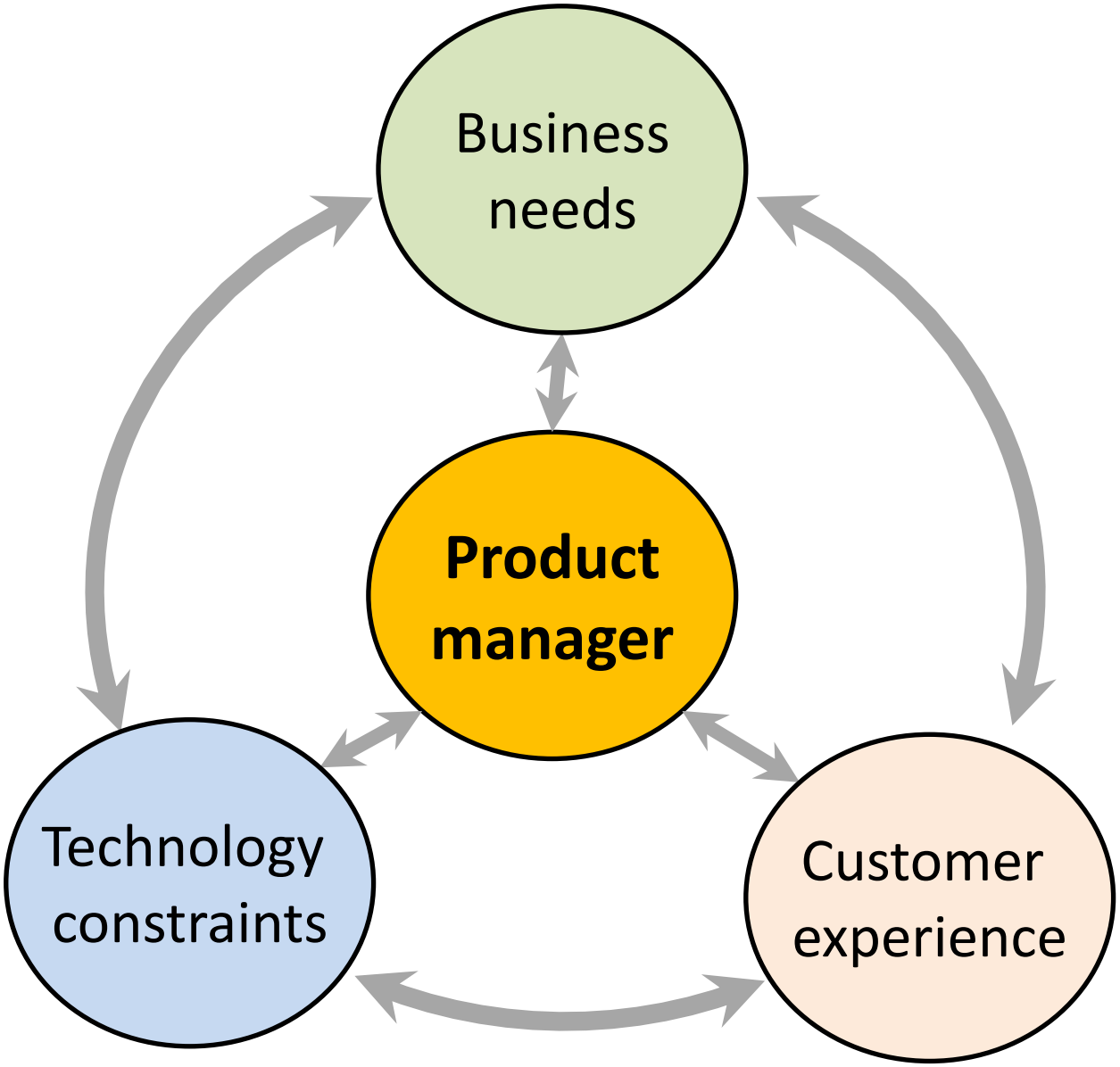
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software execution models



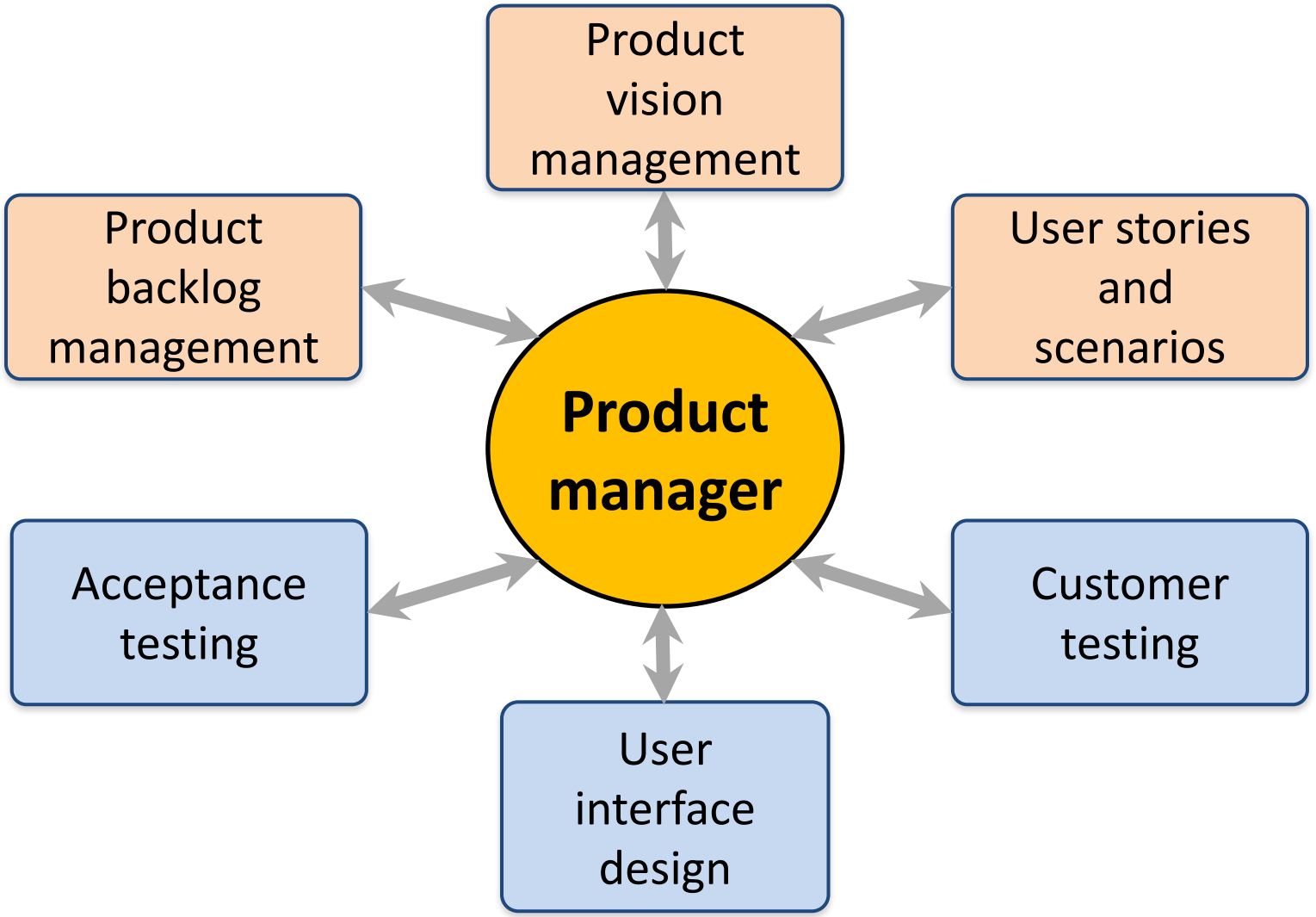
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Product management concerns



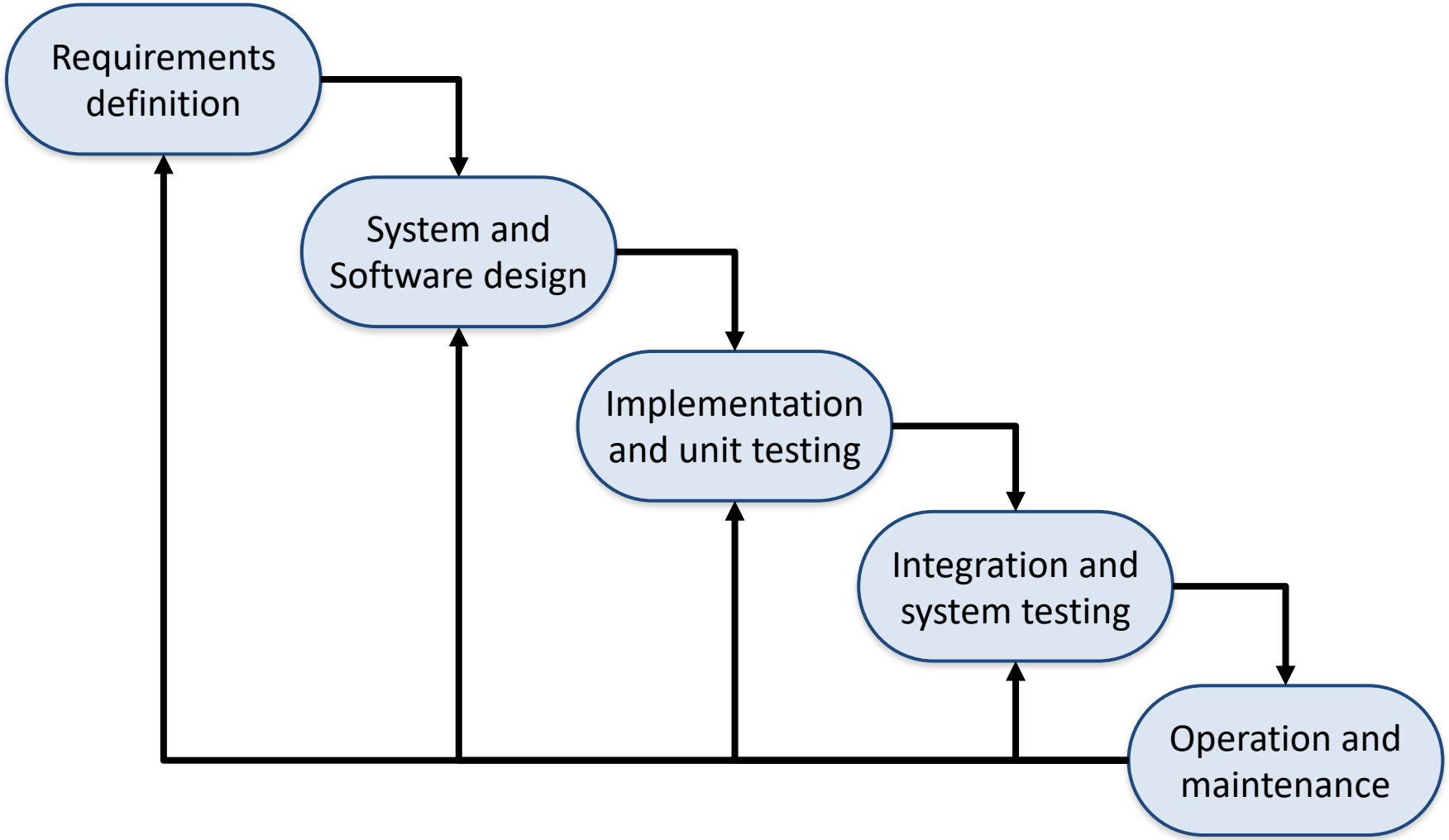
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Technical interactions of product managers



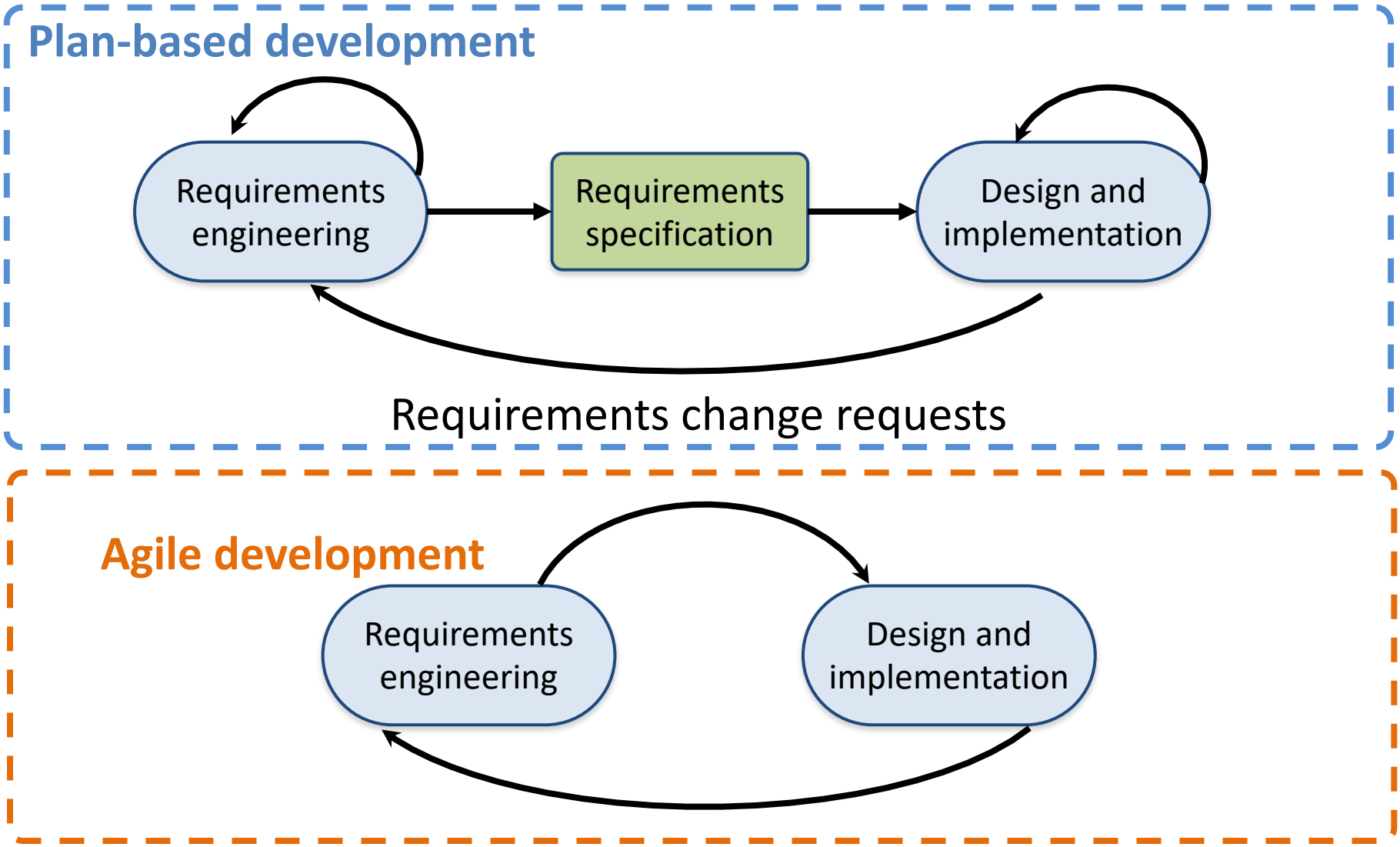
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software Development Life Cycle (SDLC)
The waterfall model



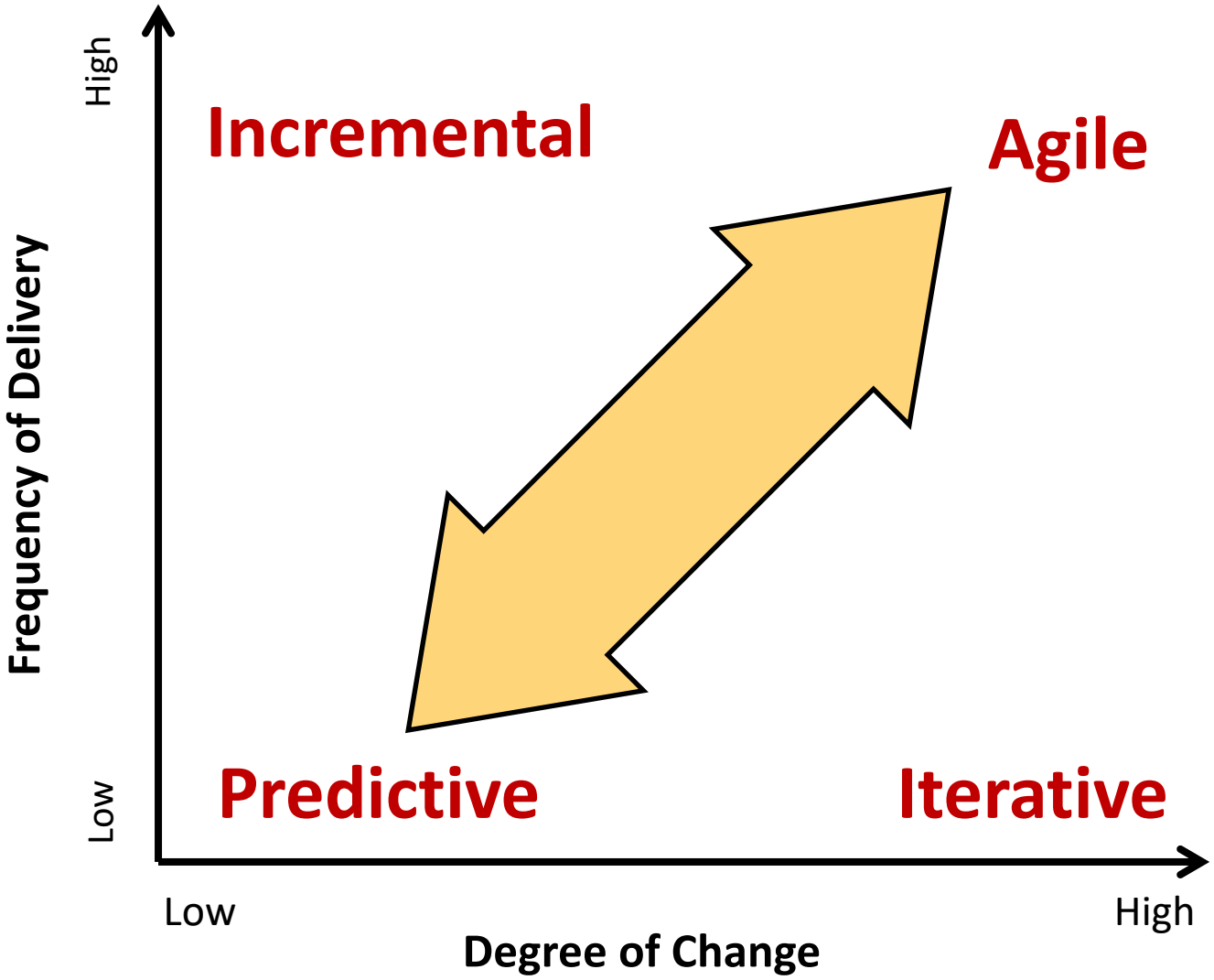
Source: Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.

Plan-based and Agile development



Source: Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.

The Continuum of Life Cycles

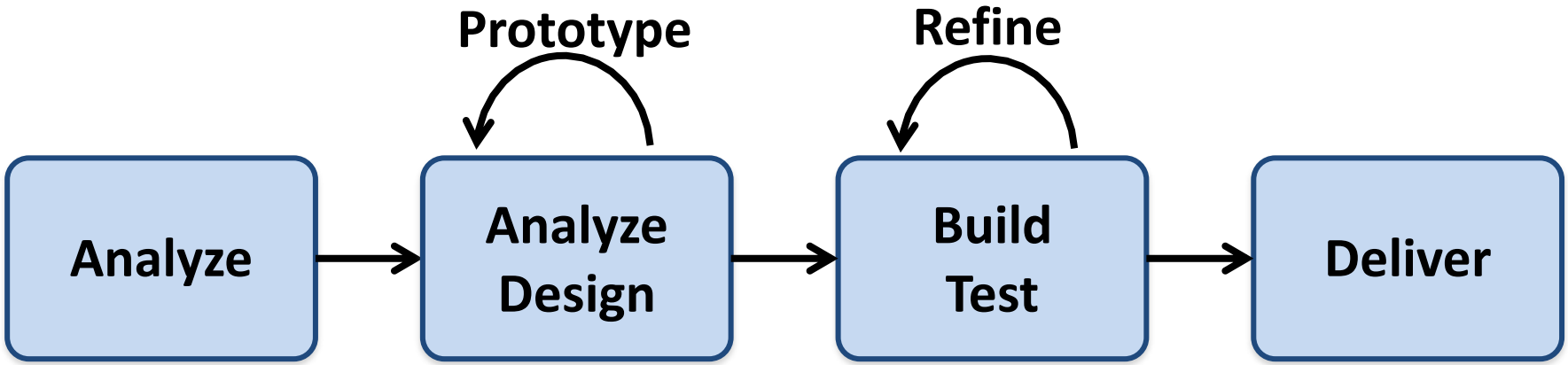


Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Predictive Life Cycle

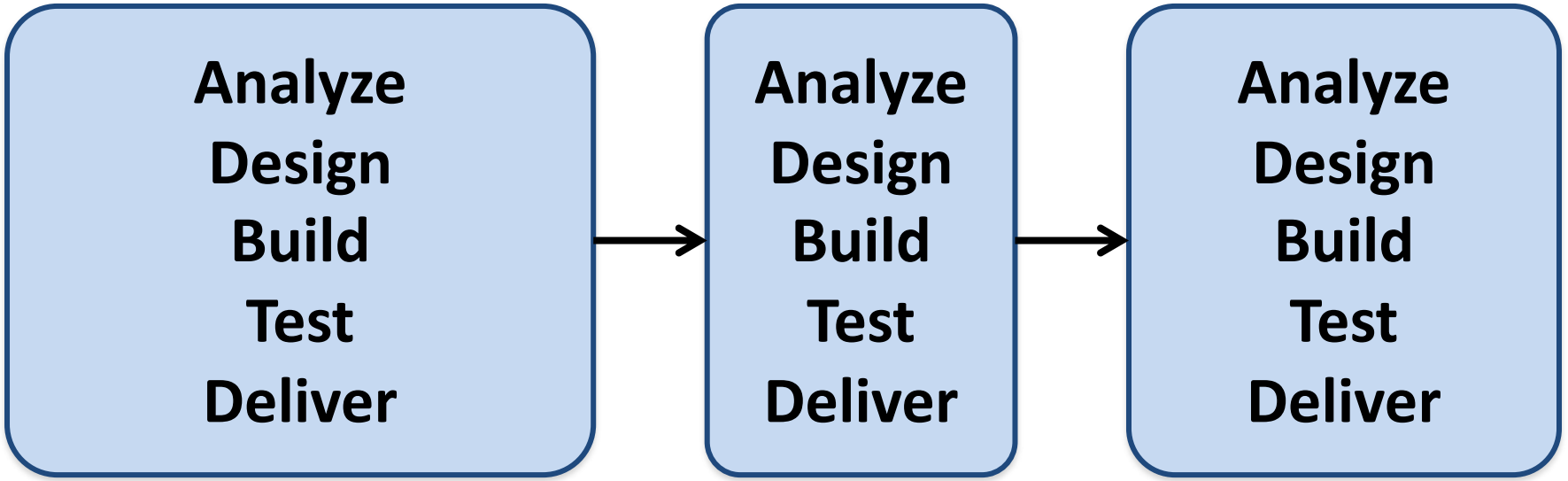
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Iterative Life Cycle



Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

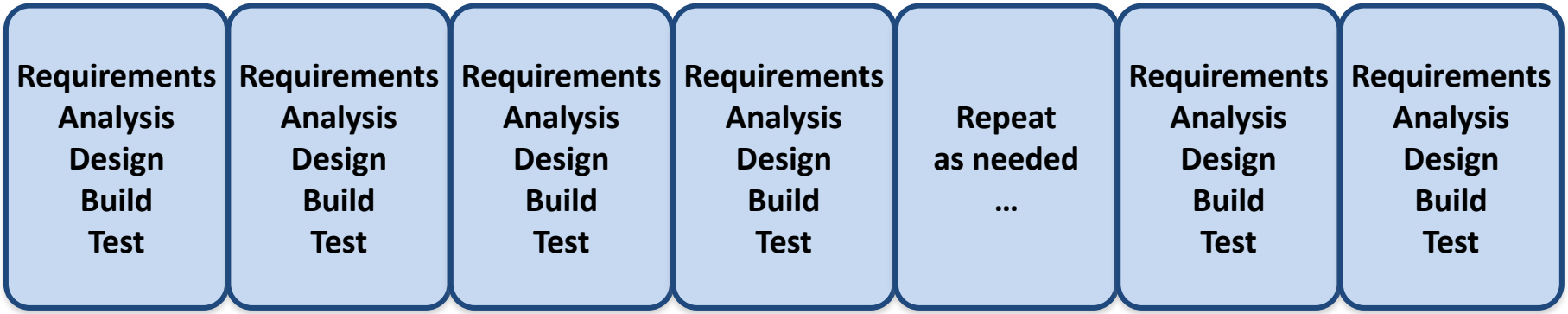
A Life Cycle of Varying-Sized Increments



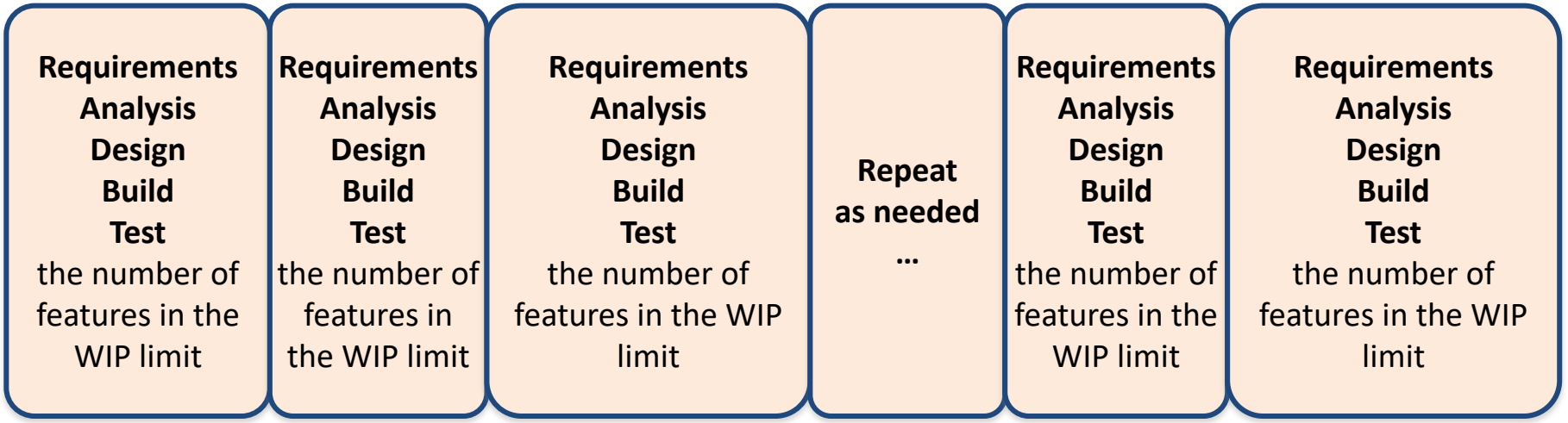
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Iteration-Based and Flow-Based Agile Life Cycles

Iteration-Based Agile

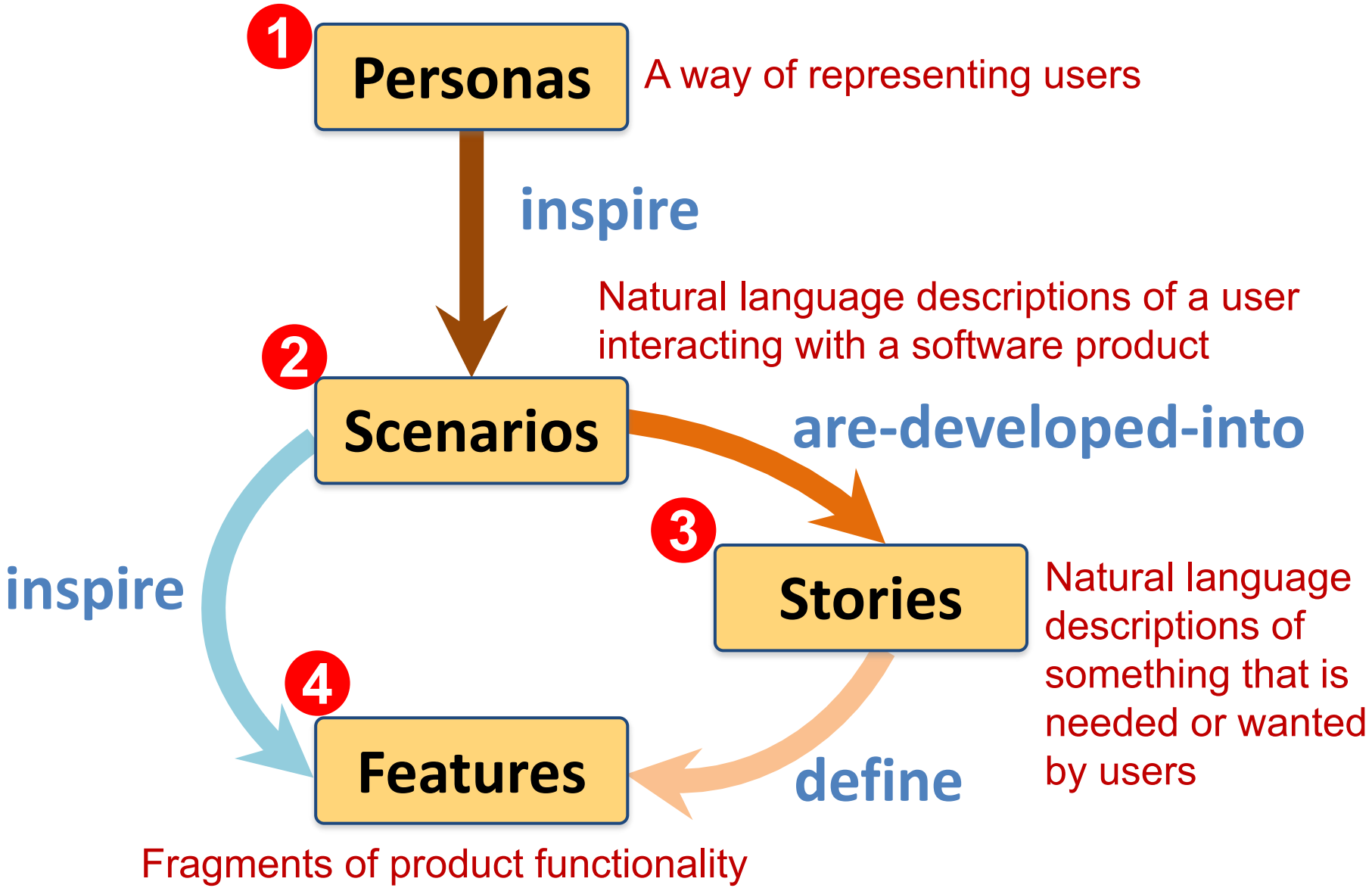


Flow-Based Agile



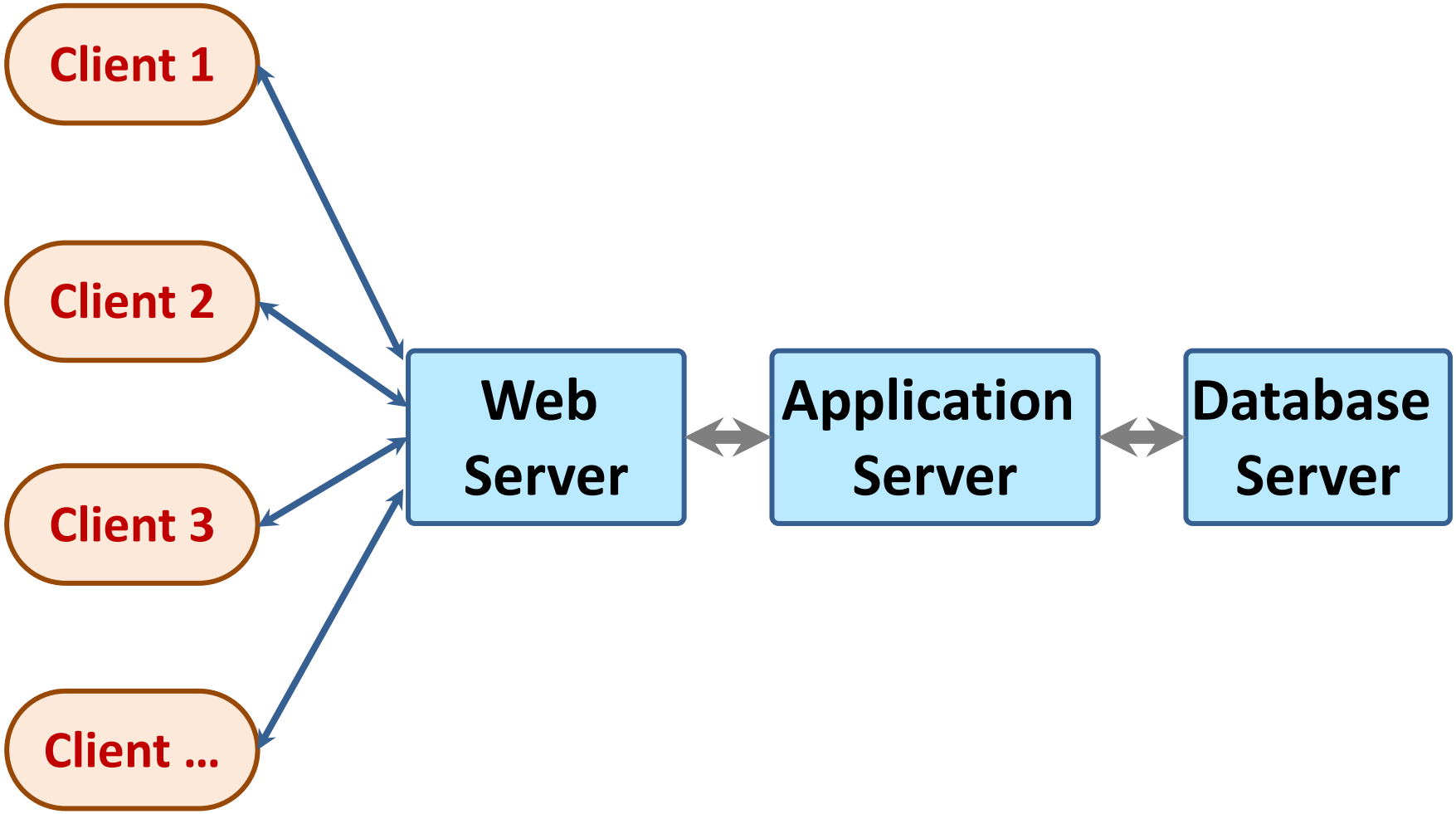
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

From personas to features



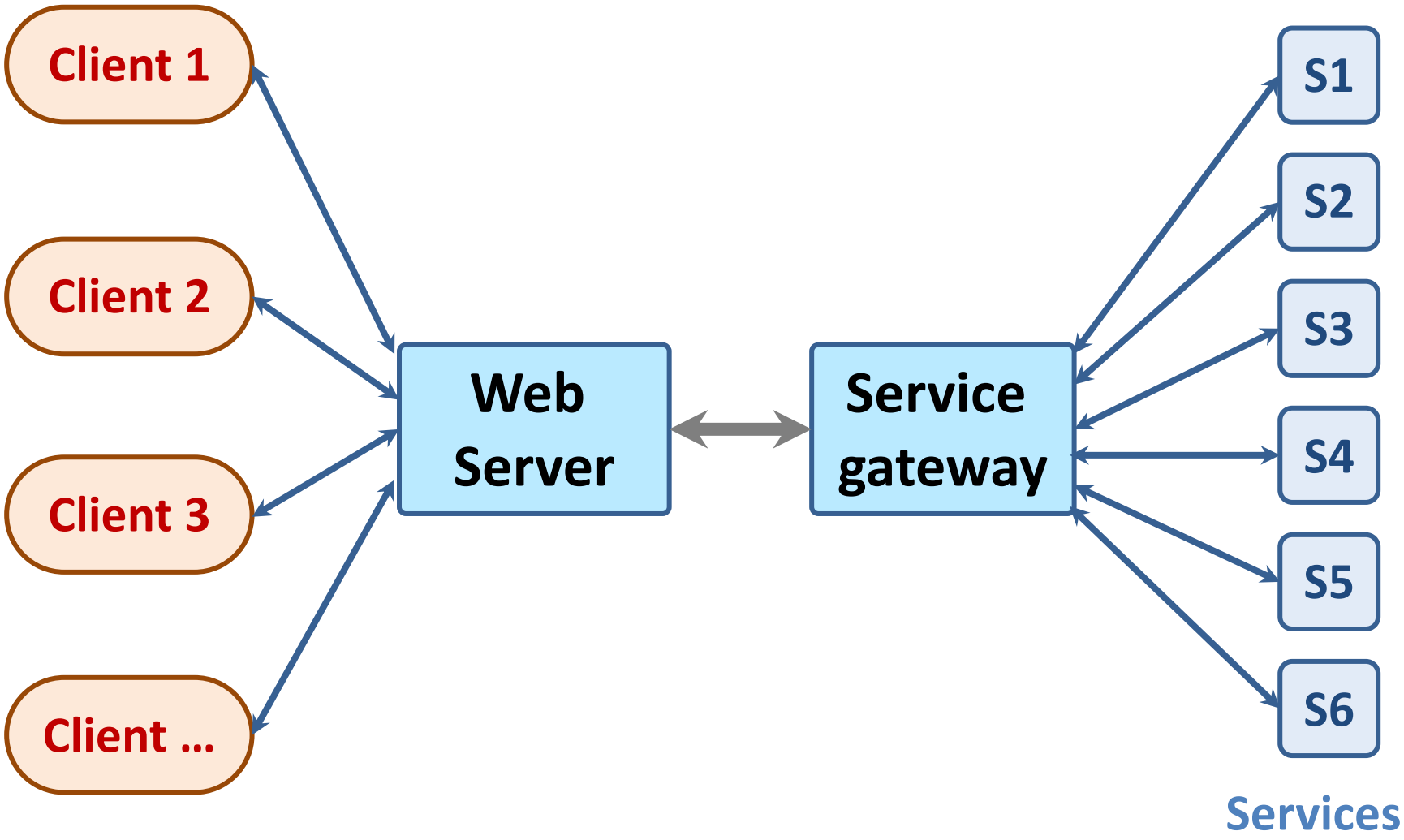
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Multi-tier client-server architecture

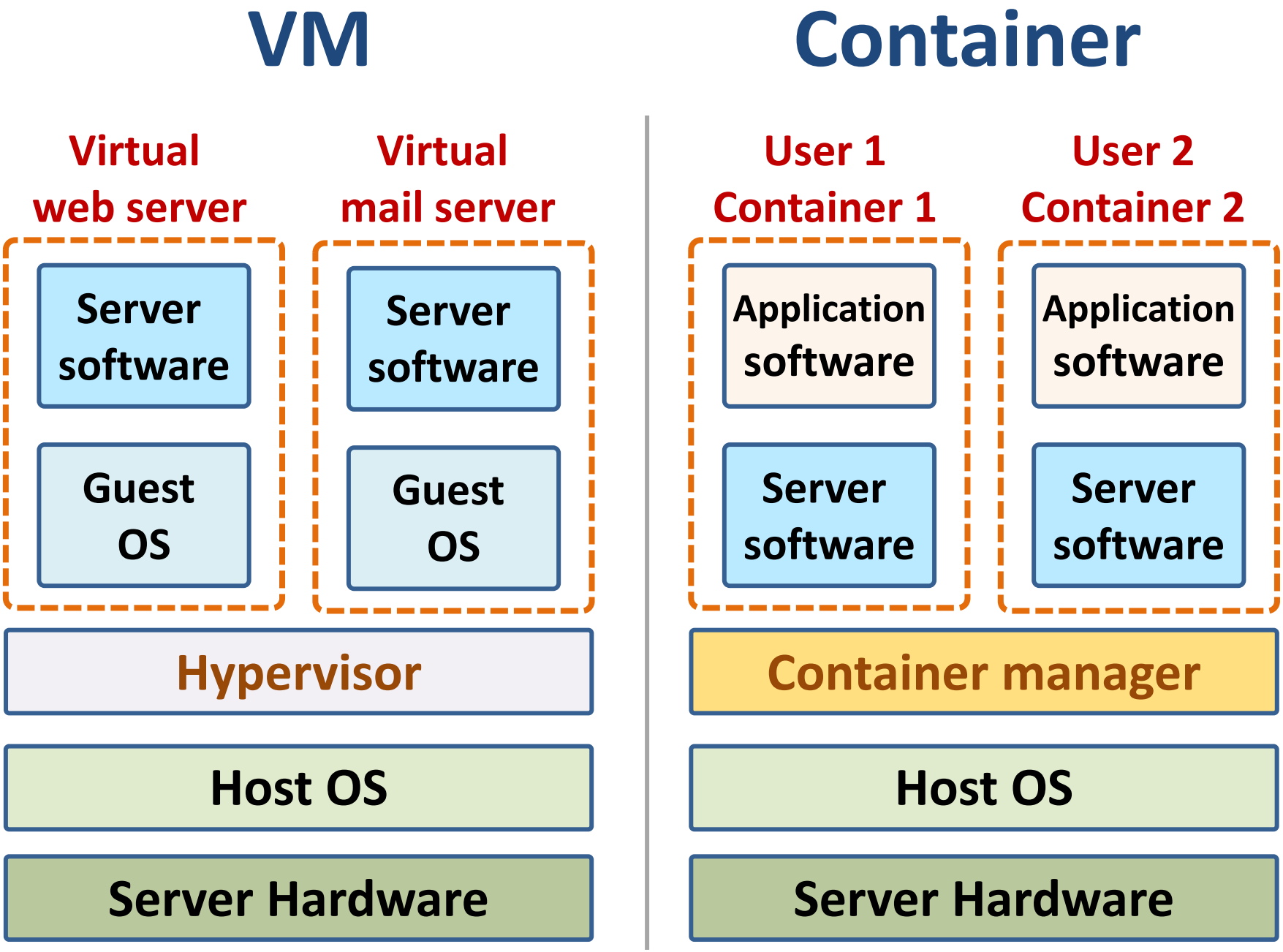


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Service-oriented Architecture

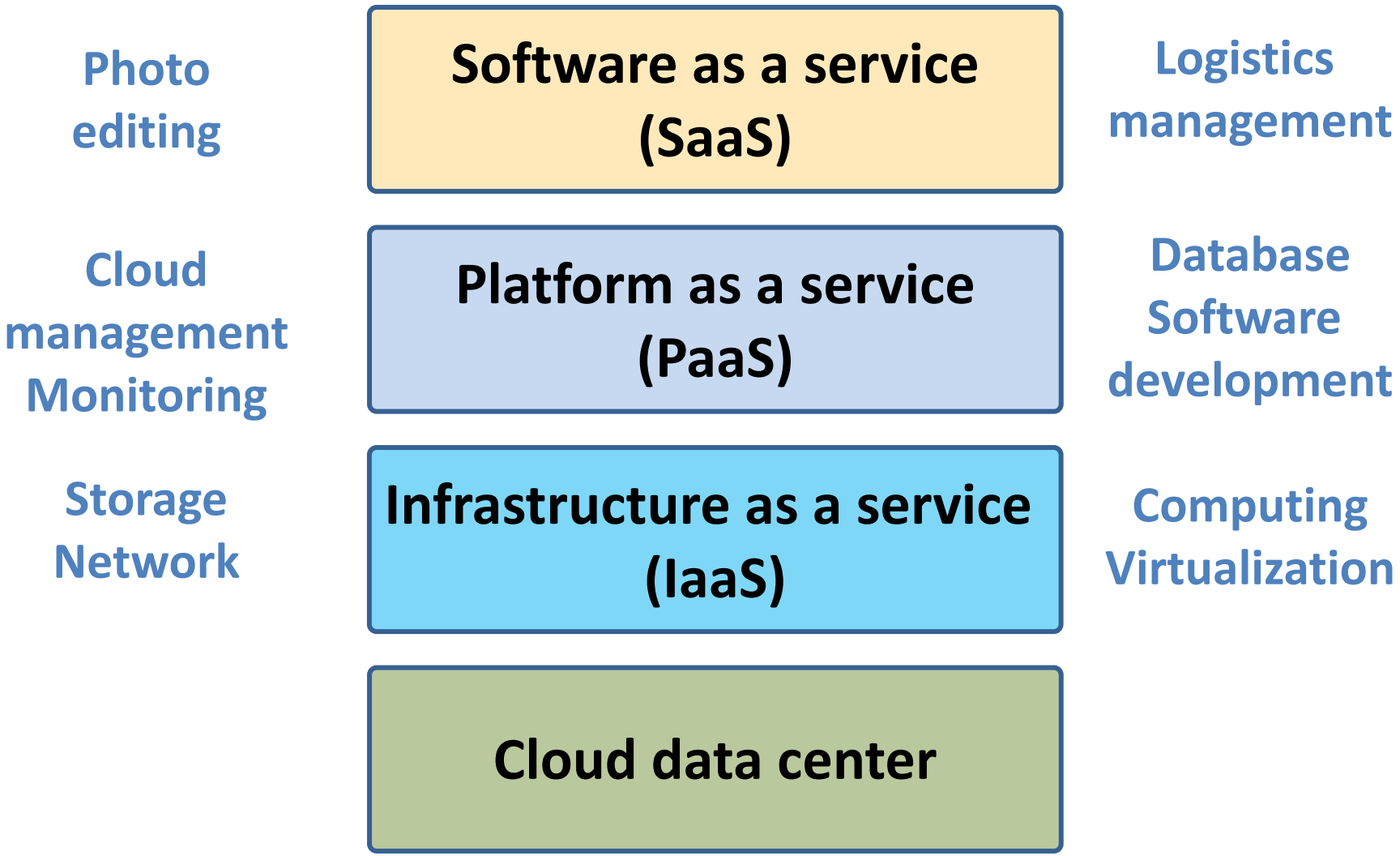


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.



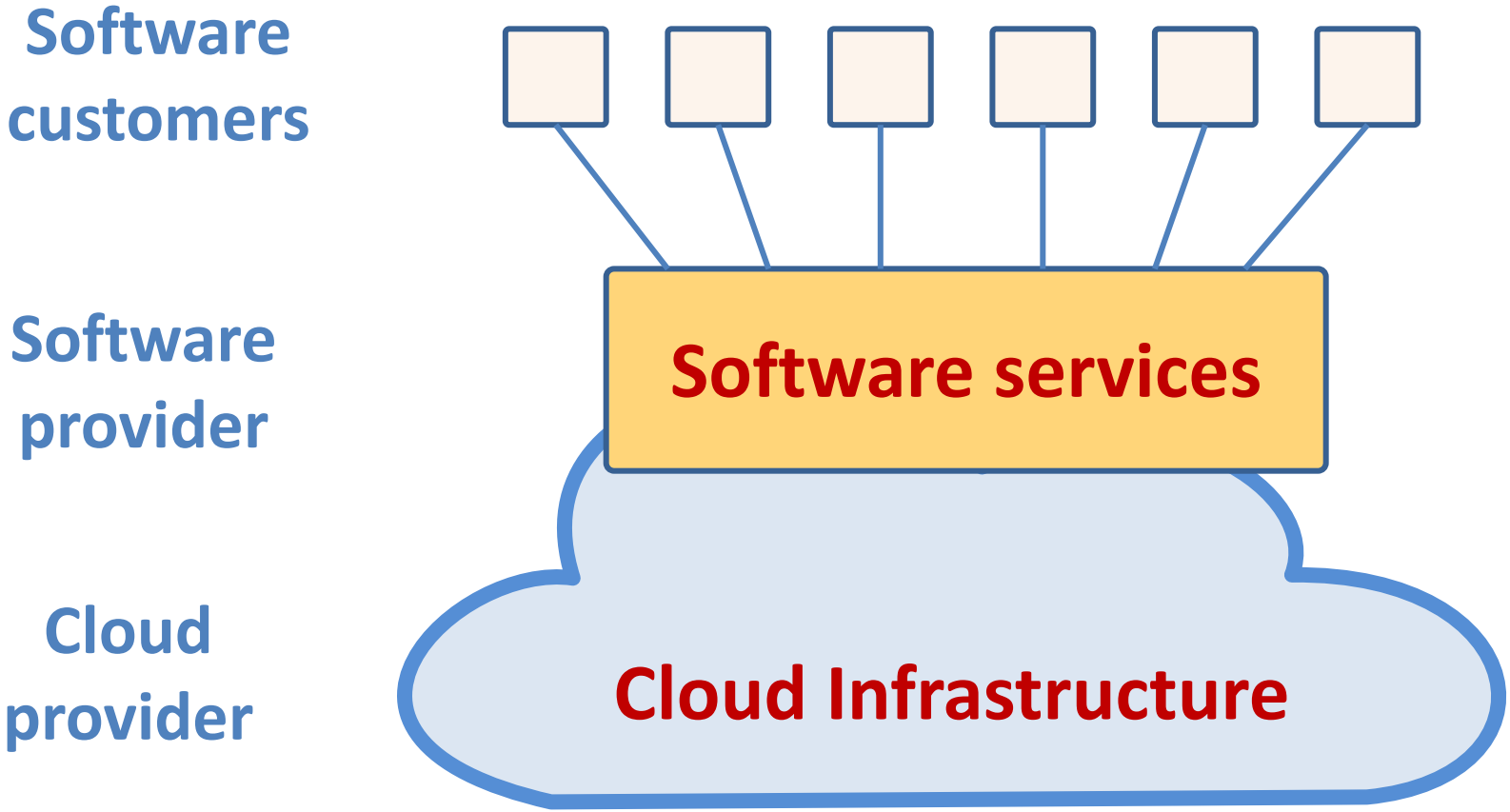
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Everything as a service



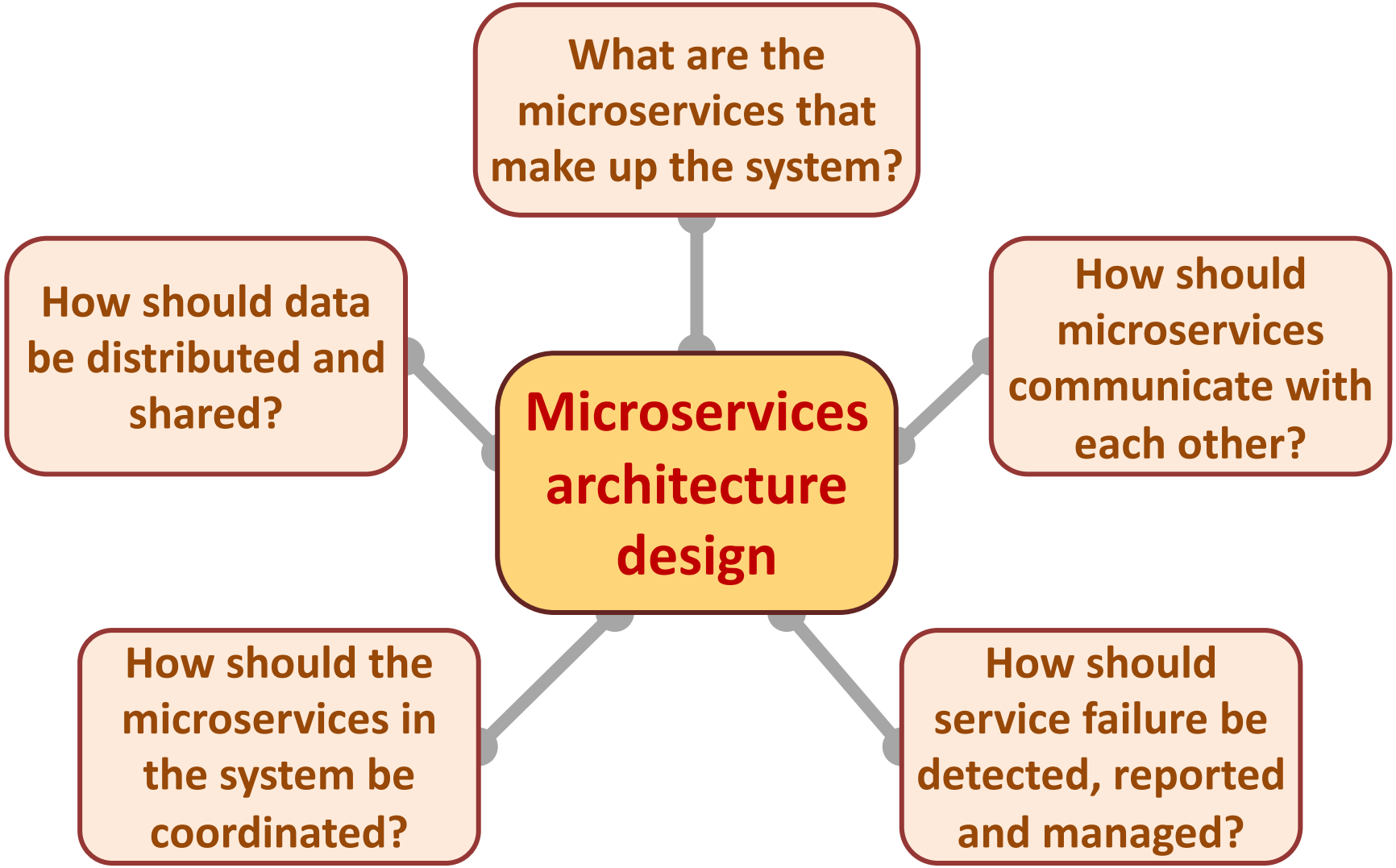
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software as a service



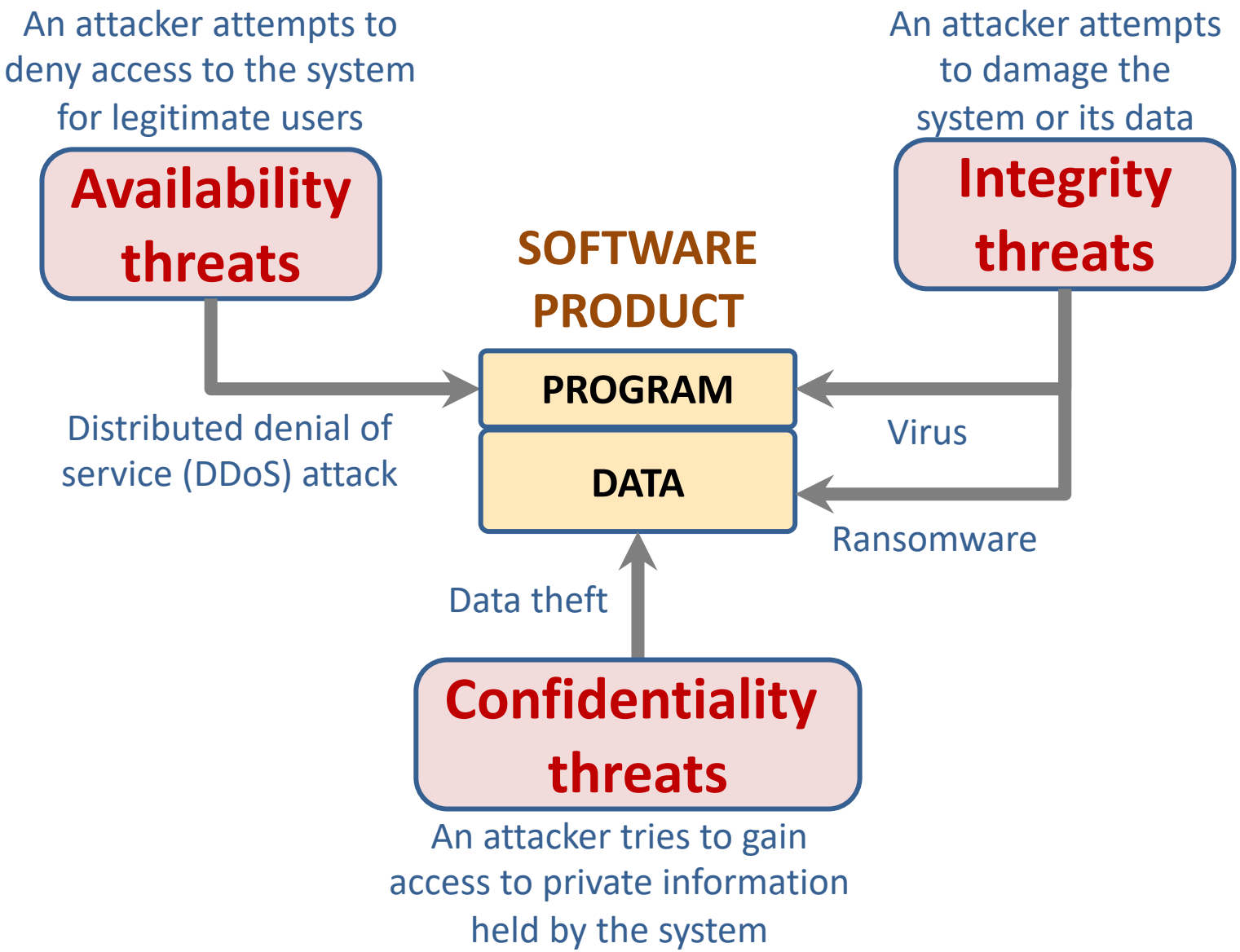
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Microservices architecture – key design questions



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Types of security threat

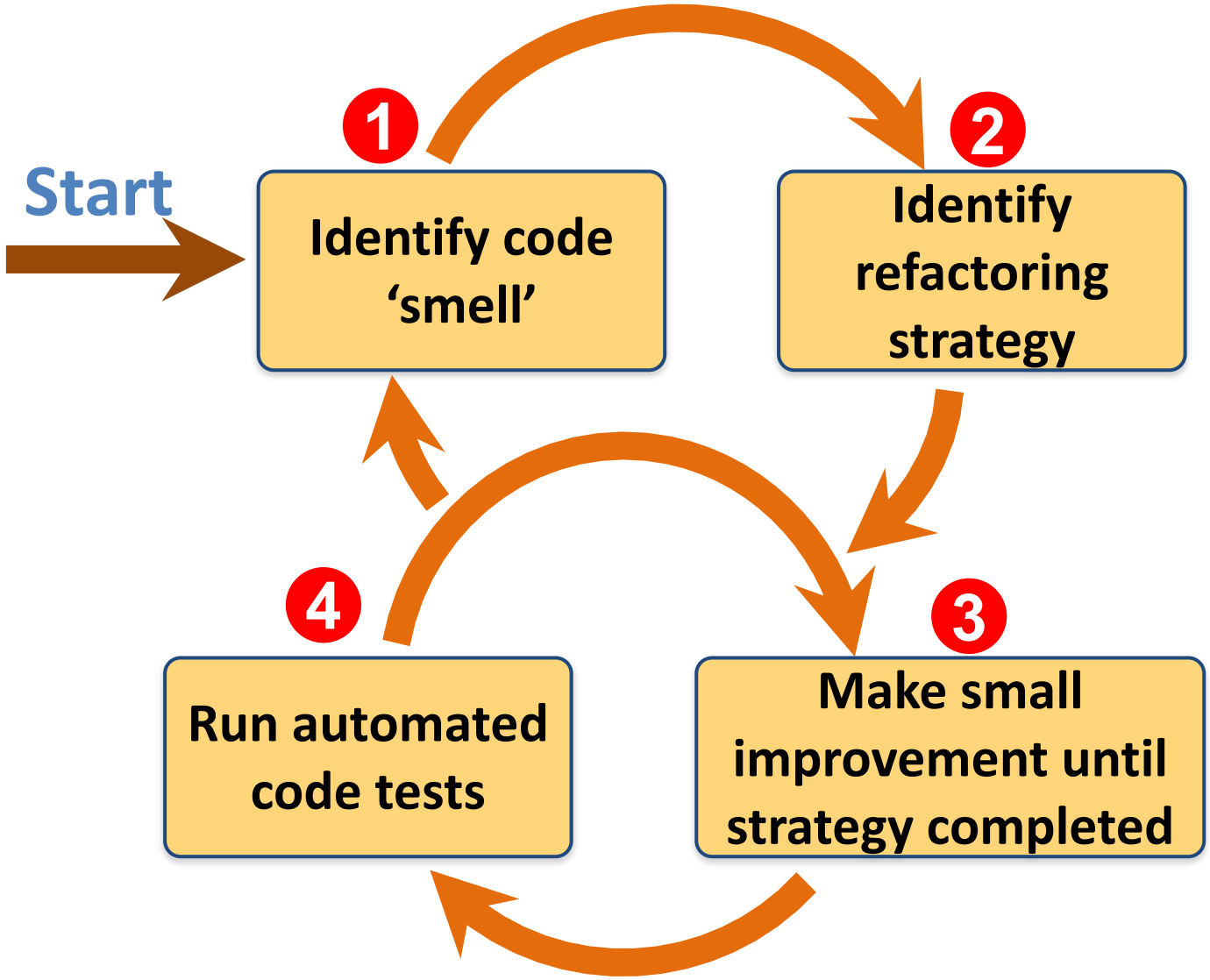


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software product quality attributes

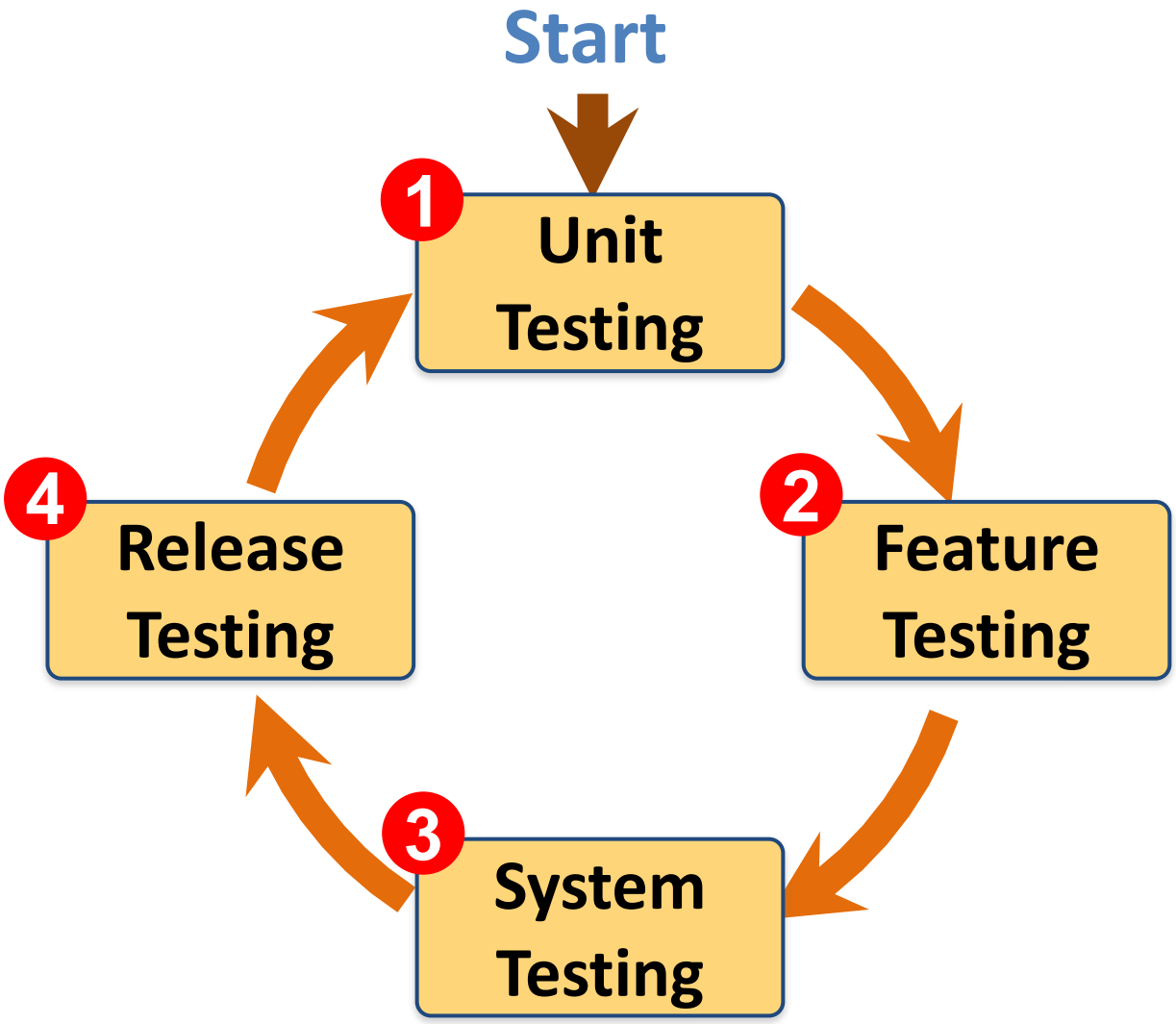
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

A refactoring process



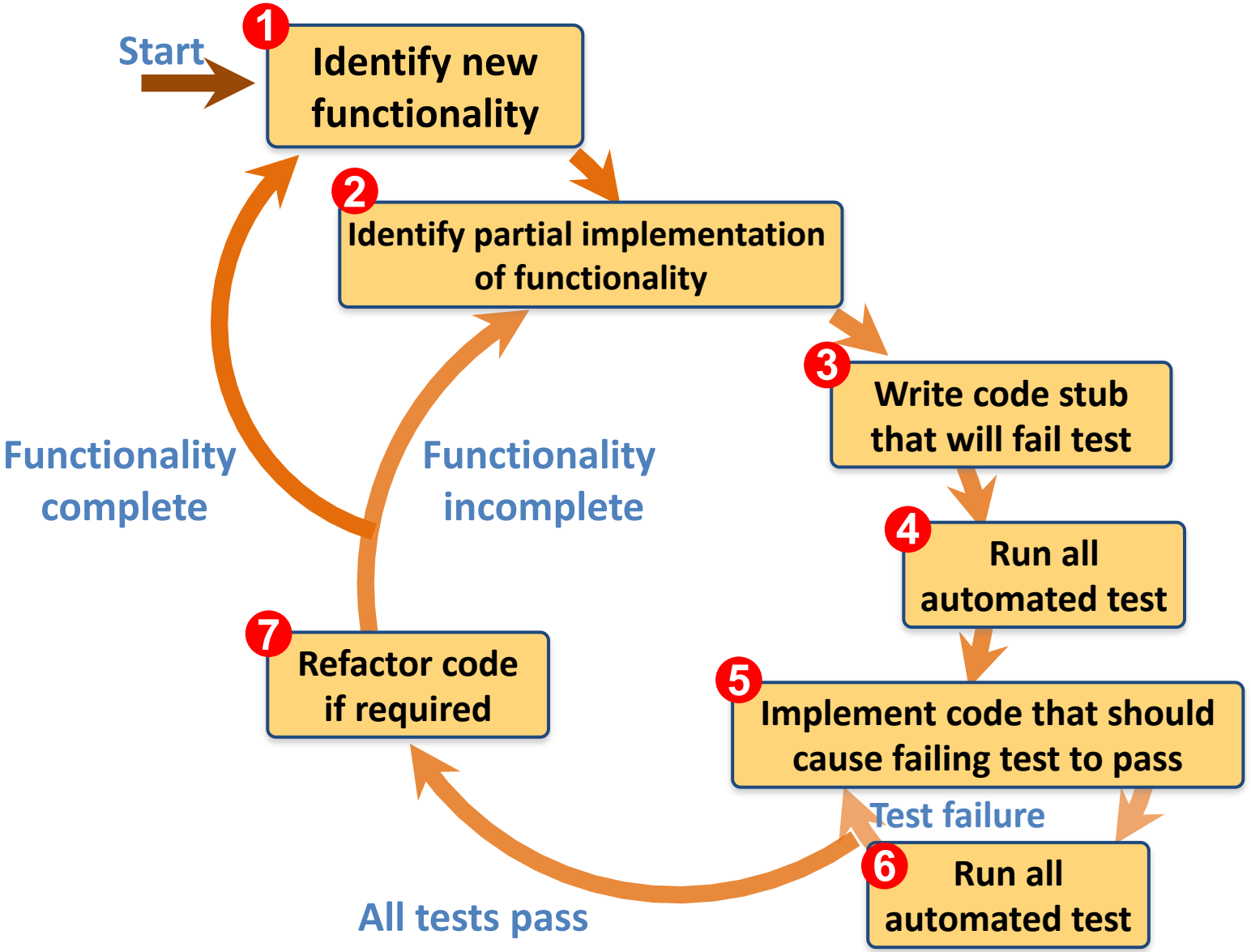
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Functional testing



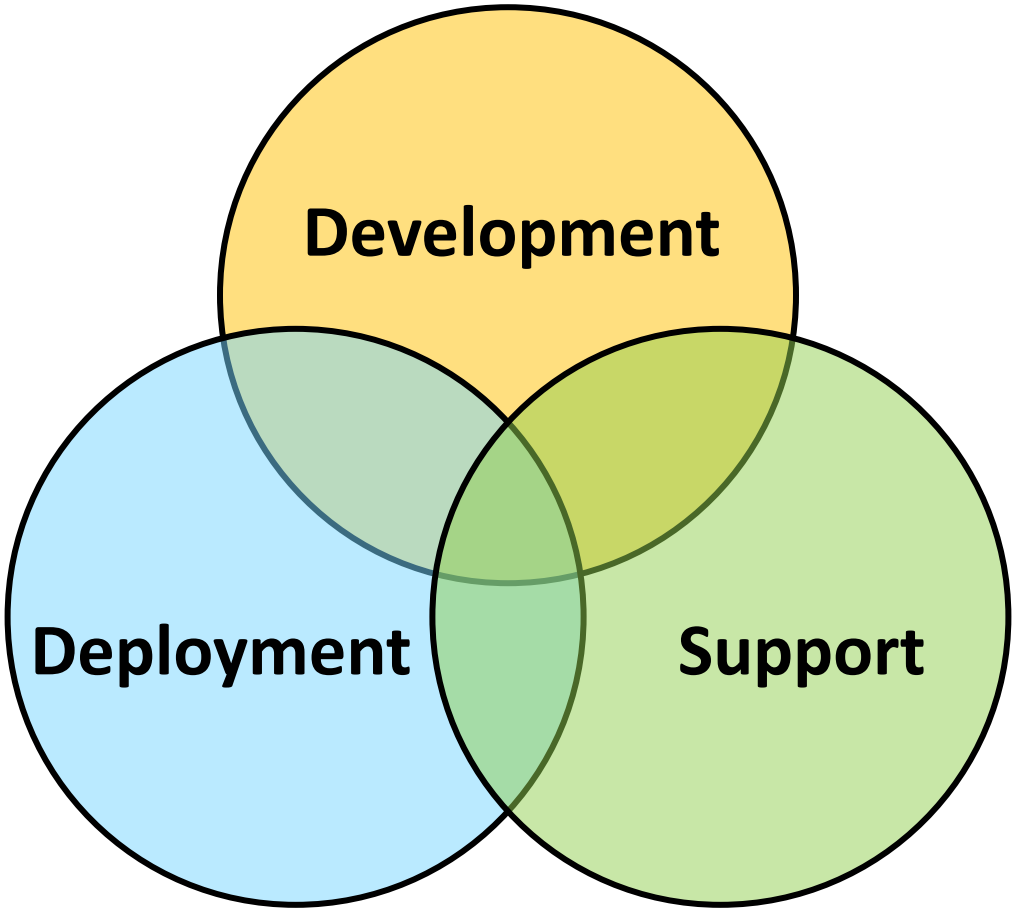
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test-driven development (TDD)



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

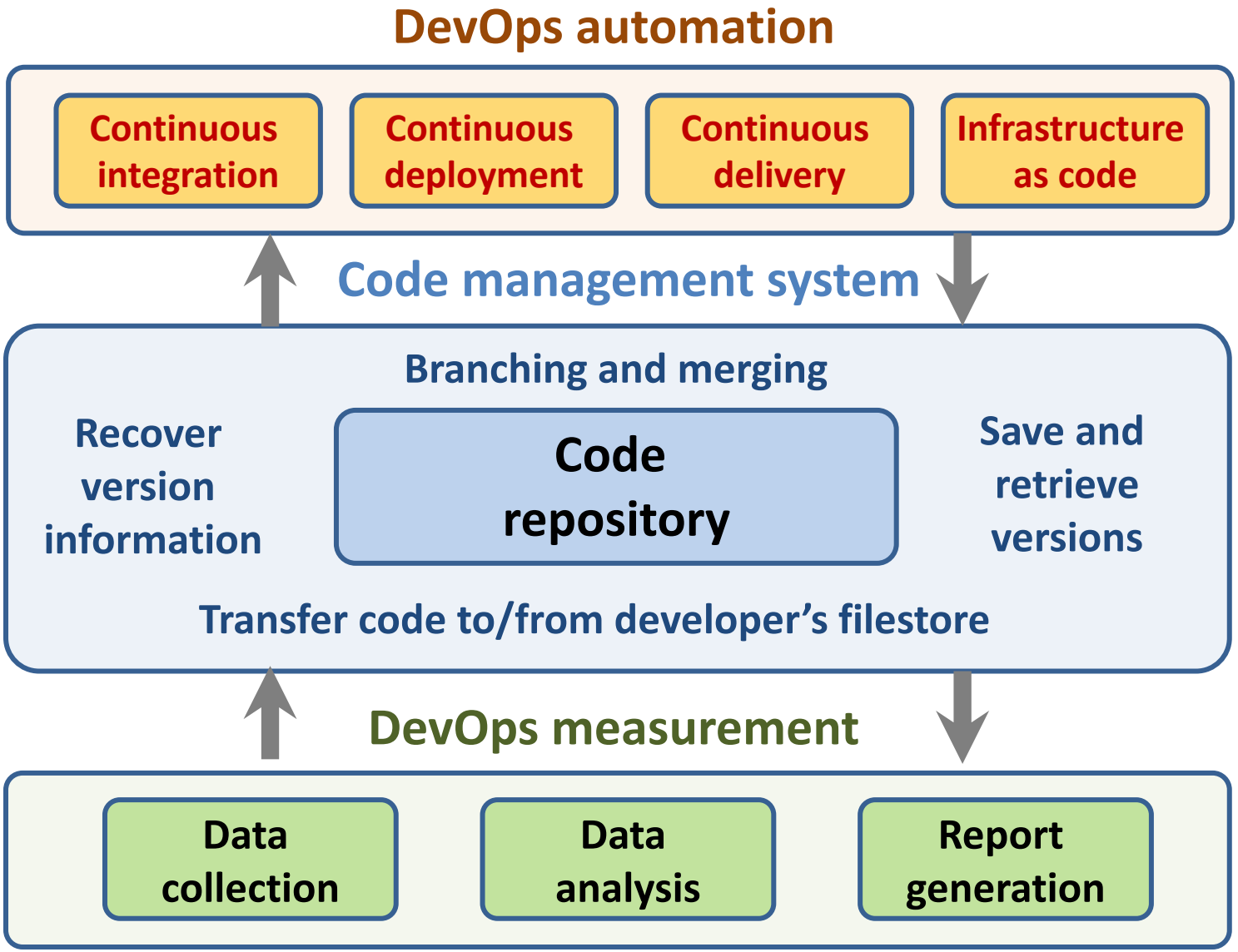
DevOps



Multi-skilled DevOps team

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Code management and DevOps



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software Architecture:

Architectural design, System decomposition, and Distribution architecture

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.

Software architecture

- 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.

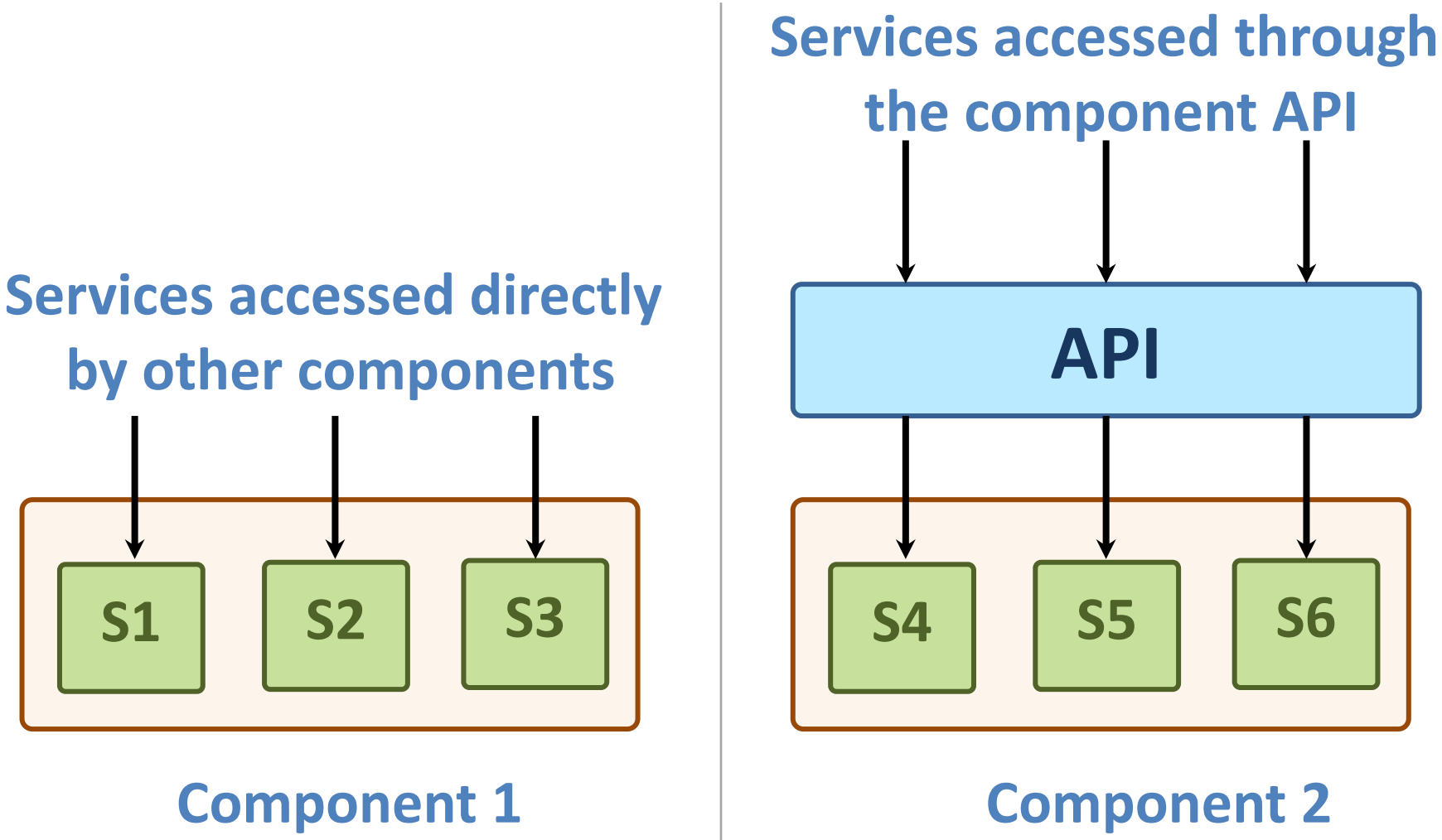
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Access to services provided by software components



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

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 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?

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Non-functional system quality attributes

- Usability

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

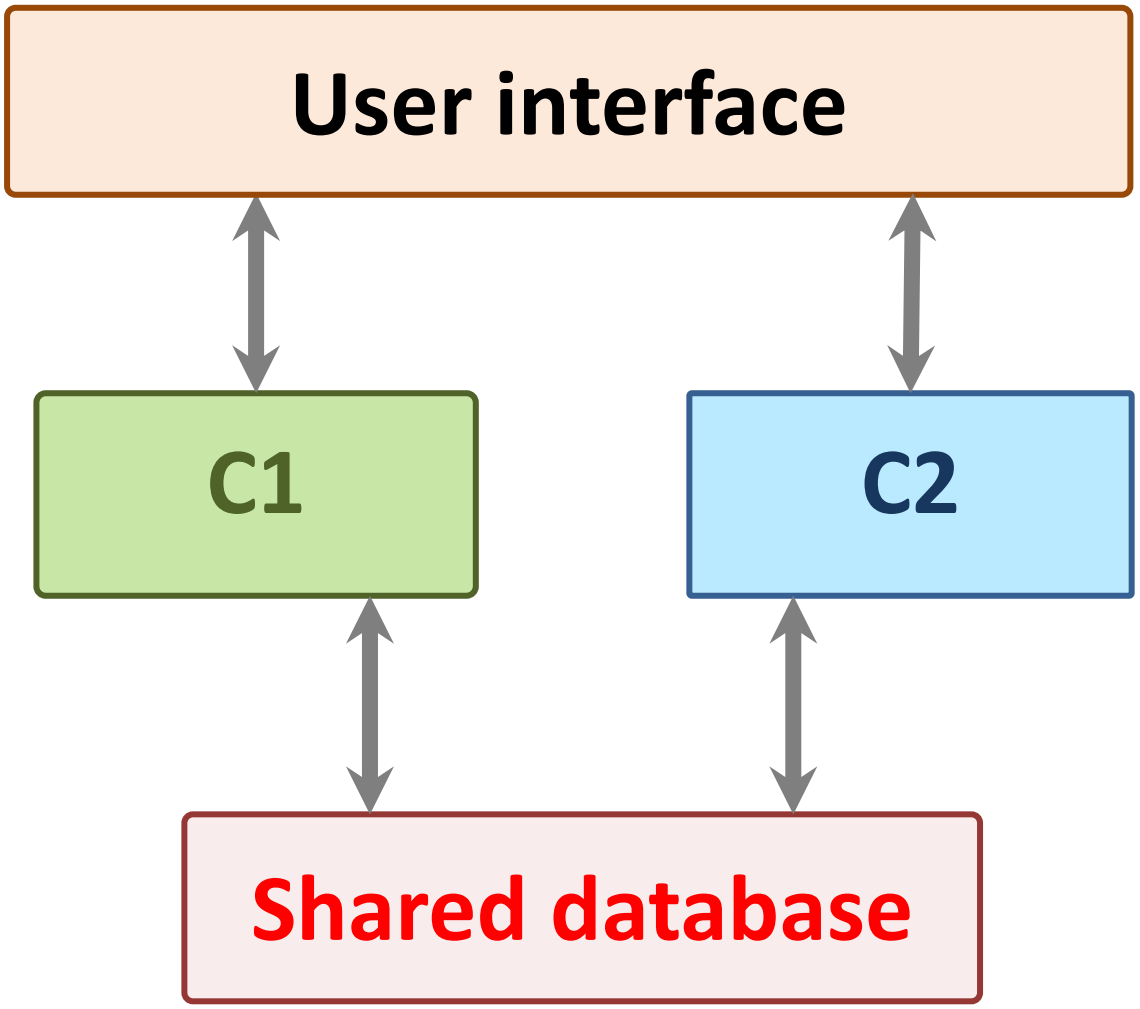
  Can the system be readily updated and new features added without undue costs?
- Resilience

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

Centralized security architectures

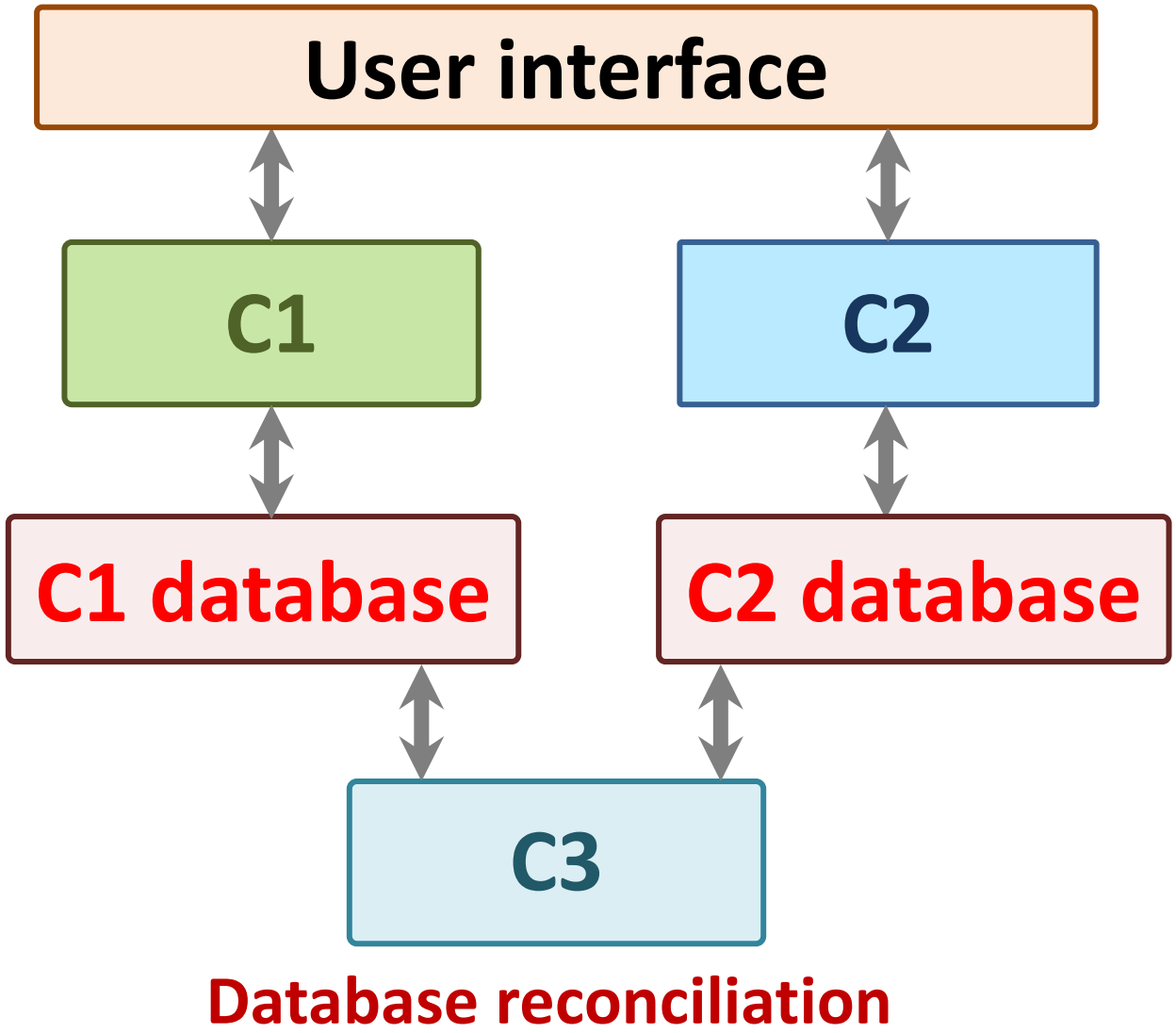
- The benefits of a centralized security architecture are that it is easier to design and build protection and that the protected information can be accessed more efficiently.
- However, if your security is breached, you lose everything.
- If you distribute information, it takes longer to access all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

Shared database architecture



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Multiple database architecture

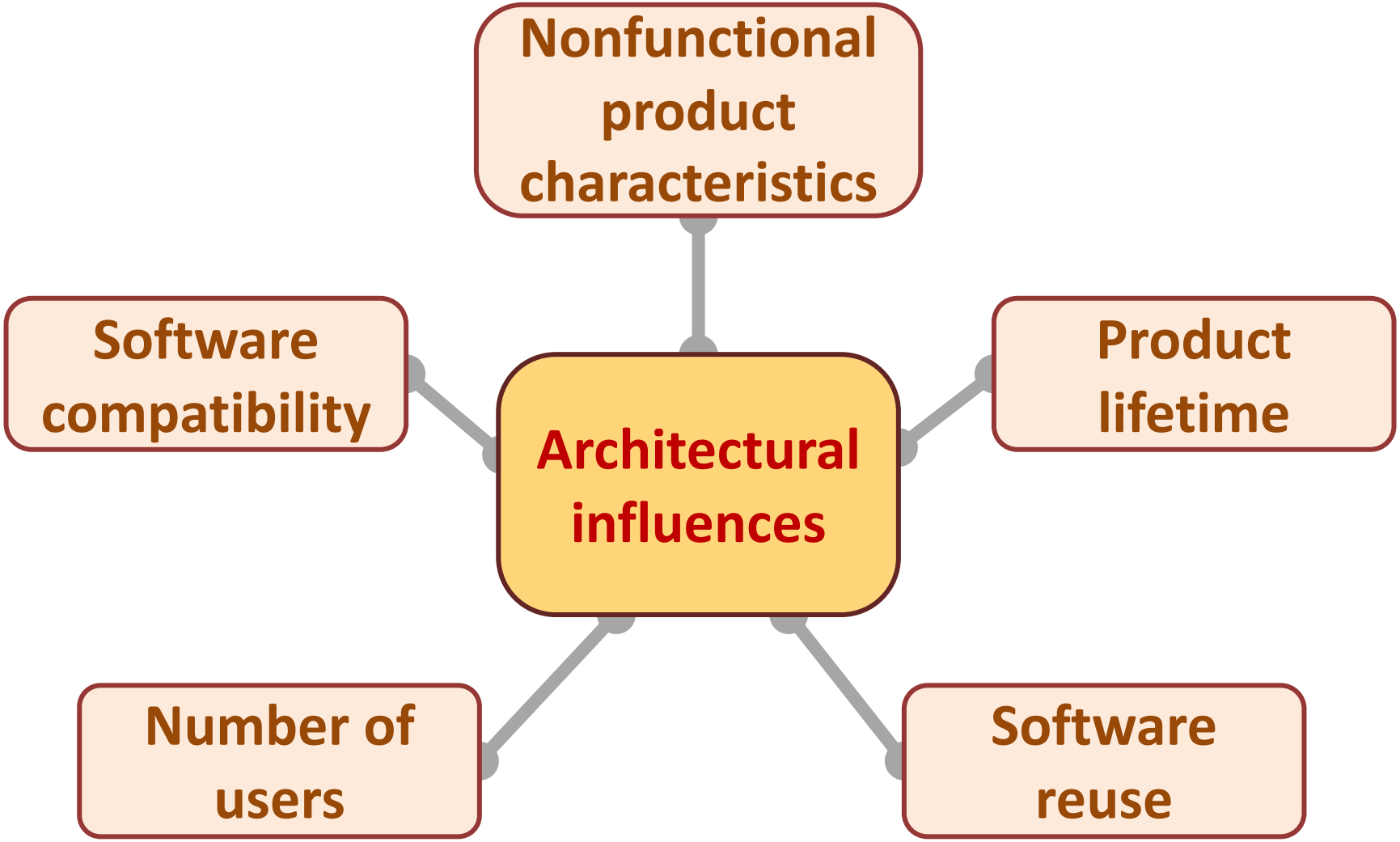


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Maintainability and performance

- **Shared database architecture**:
  - system with two components (C1 and C2) that share a common database.
- **Multiple database architecture**:
  - 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.
- 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.

Issues that influence architectural decisions



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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 is 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.

The importance of architectural design issues

- **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.

Trade off: Maintainability vs performance

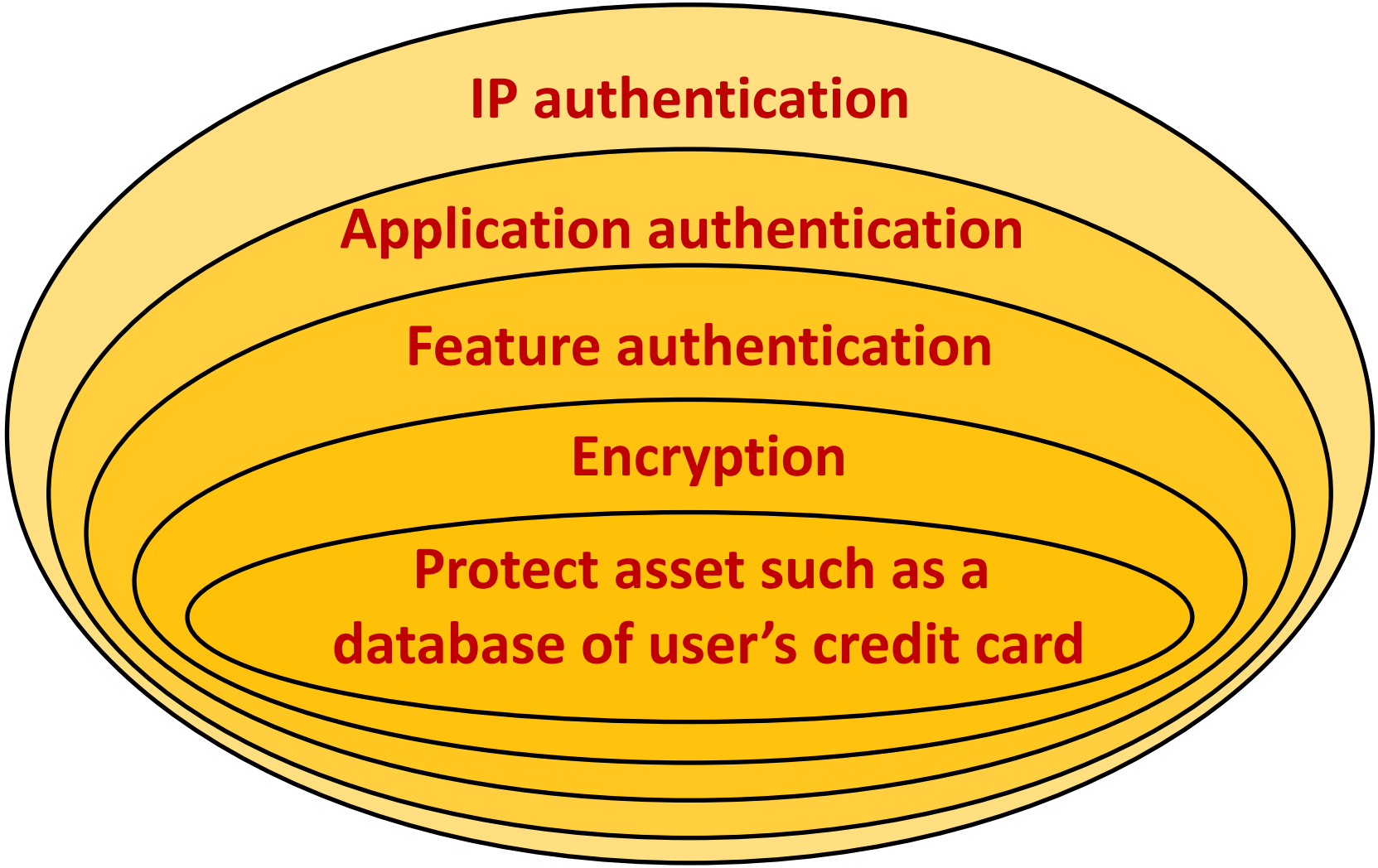
- System maintainability is an attribute that reflects how difficult and expensive it is to make changes to a system after it has been released to customers.
  - You improve maintainability by building a system from small self-contained parts, each of which can be replaced or enhanced if changes are required.
- In architectural terms, this means that the system should be decomposed into fine-grain components, each of which does one thing and one thing only.
  - However, it takes time for components to communicate with each other. Consequently, if many components are involved in implementing a product feature, the software will be slower.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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

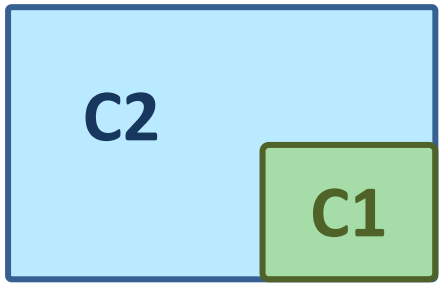
An architectural model of a document retrieval system

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

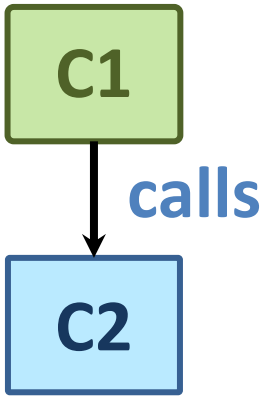
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Examples of component relationships

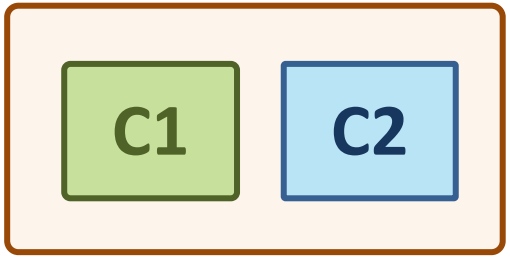
C1 is part of C2



C1 uses C2



C1 is-located-with C2



C1 shared-data-with C2

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Architectural design guidelines

Separation of concerns

Organize your architecture into components that focus on a single concern

Stable interfaces

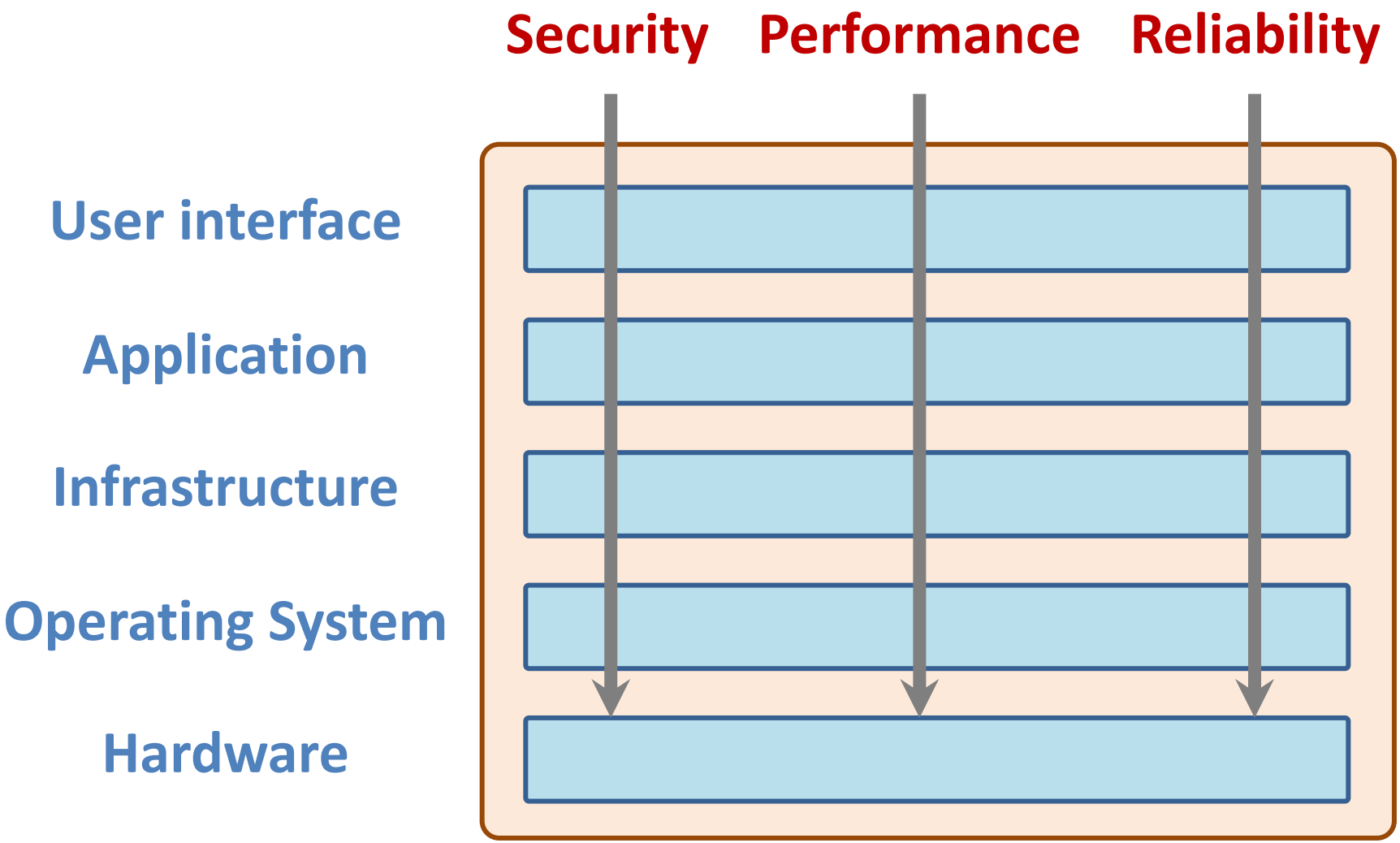
Design component interfaces that are coherent and that changes slowly

Implement once

Avoid duplicating functionality at different places in your architecture

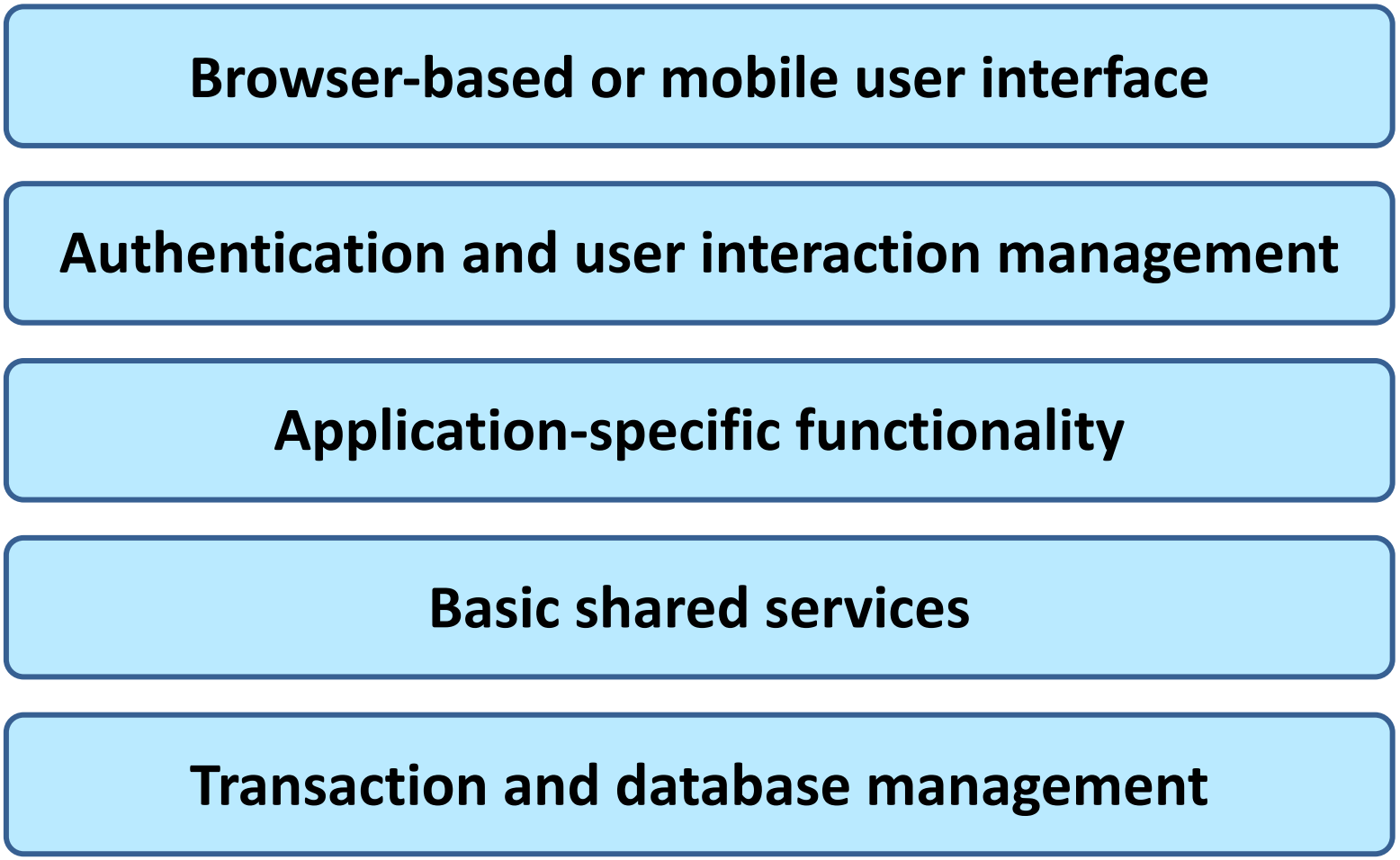
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Cross-cutting concerns



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

A generic layered architecture for a web-based application



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

A layered architectural model of the iLearn system

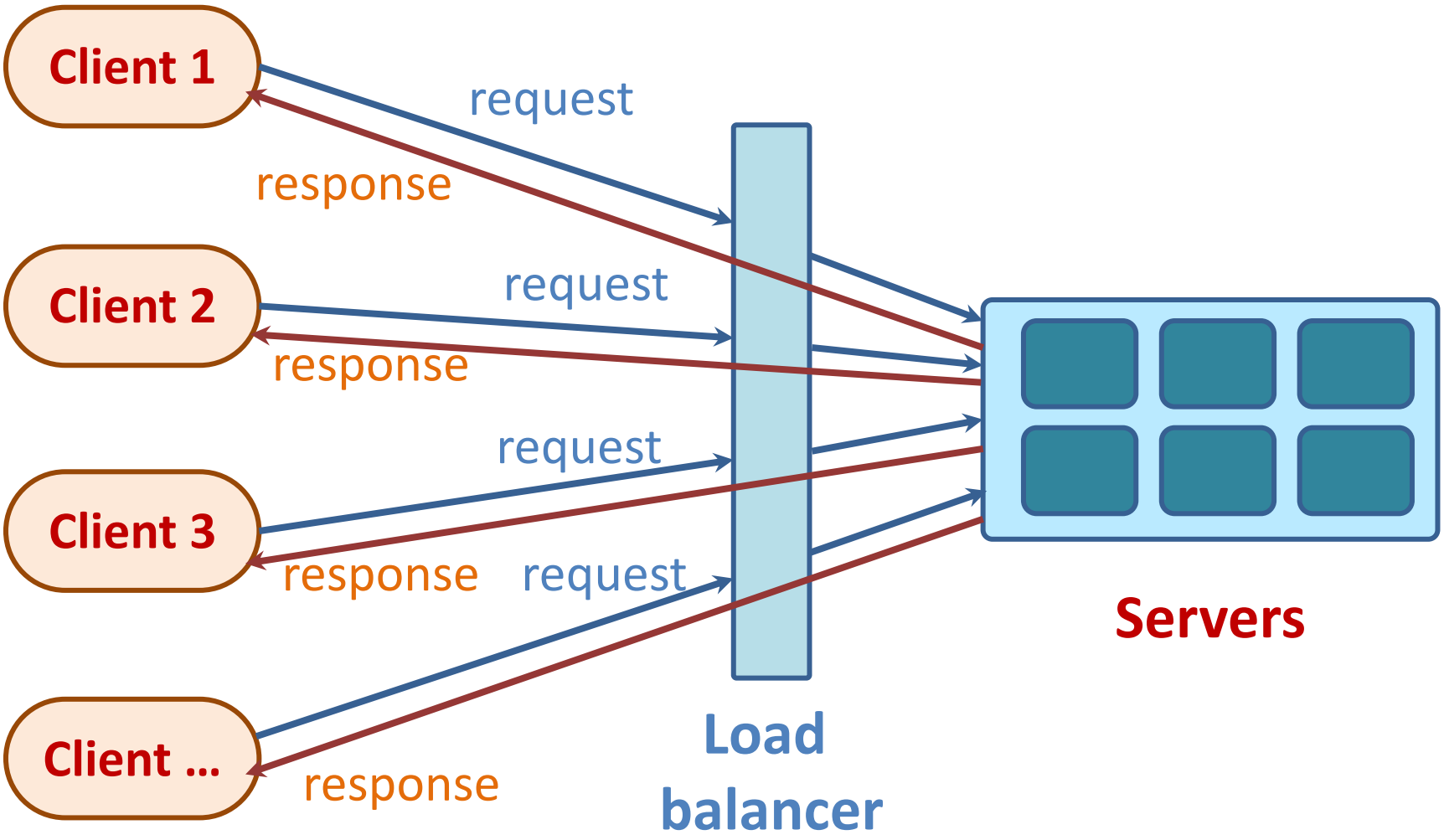
User interface	Web browser iLearn app
User interface management	Interface creation Forms management Interface delivery Login
Configuration services	Group configuration Application configuration Security configuration User interface configuration Setup service
Application services	Archive access Word processor Video conf. Email and messaging User installed application Blog Wiki Spreadsheet Presentation Drawing
Integrated services	Resource discovery User analytics Virtual Learning environment Authentication and authorization
Shared infrastructure services	Authentication Logging and monitoring Application interfacing User storage Application storage Search

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Distribution architecture

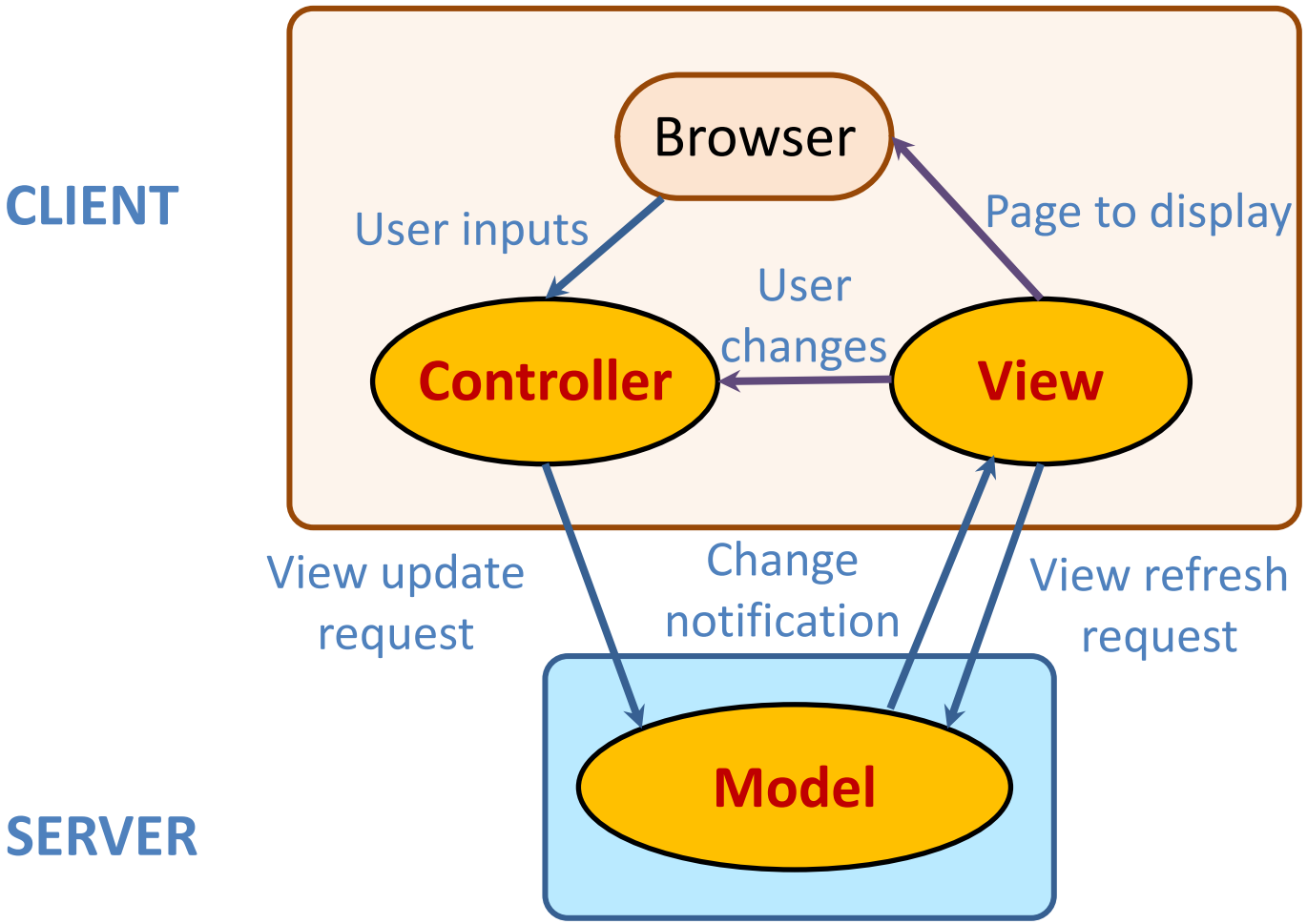
- The distribution architecture of a software system defines the servers in the system and the allocation of components to these servers.
- Client-server architectures are a type of distribution architecture that is suited to applications where clients access a shared database and business logic operations 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



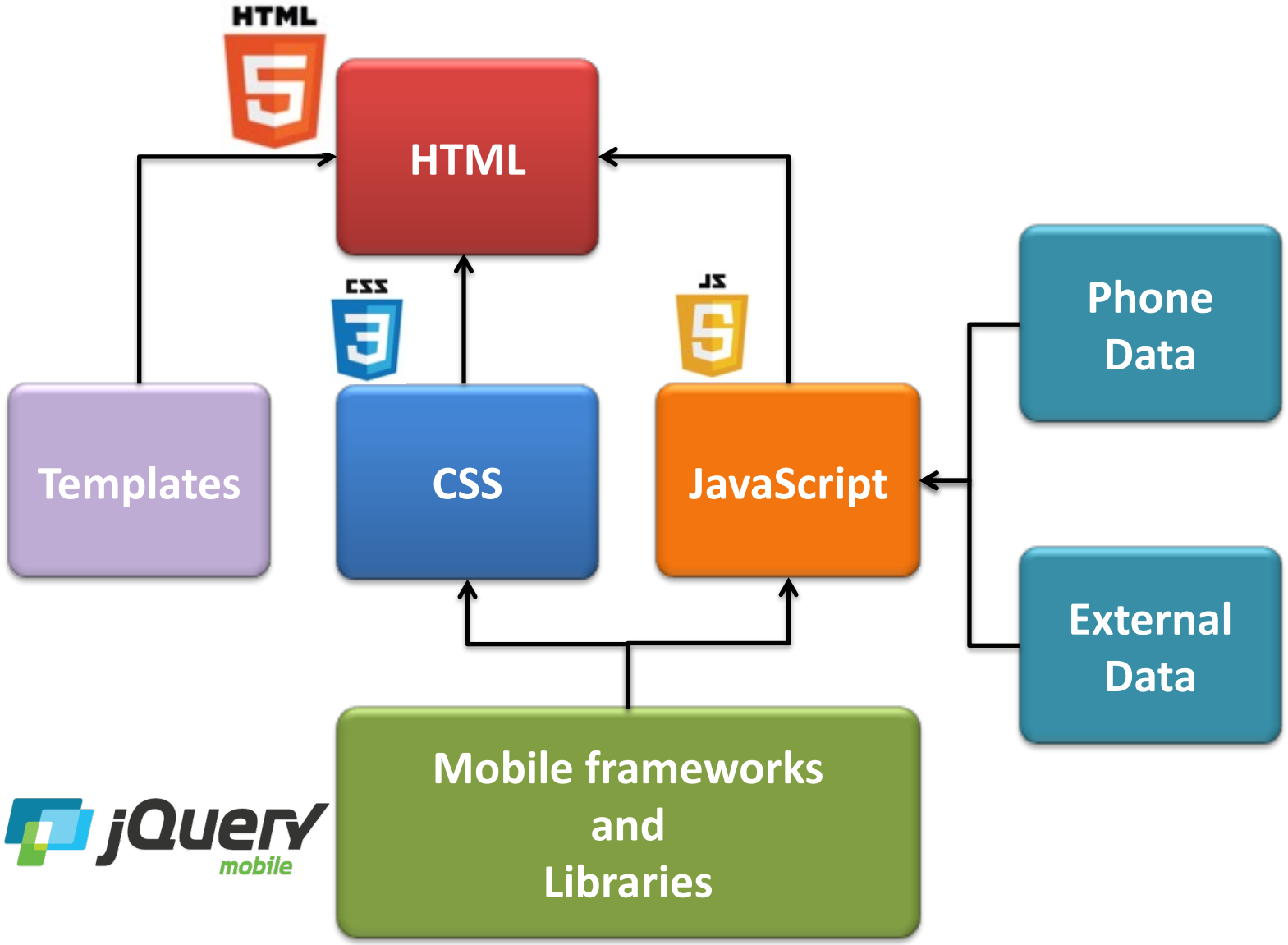
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

The Model-View-Controller (MVC) pattern



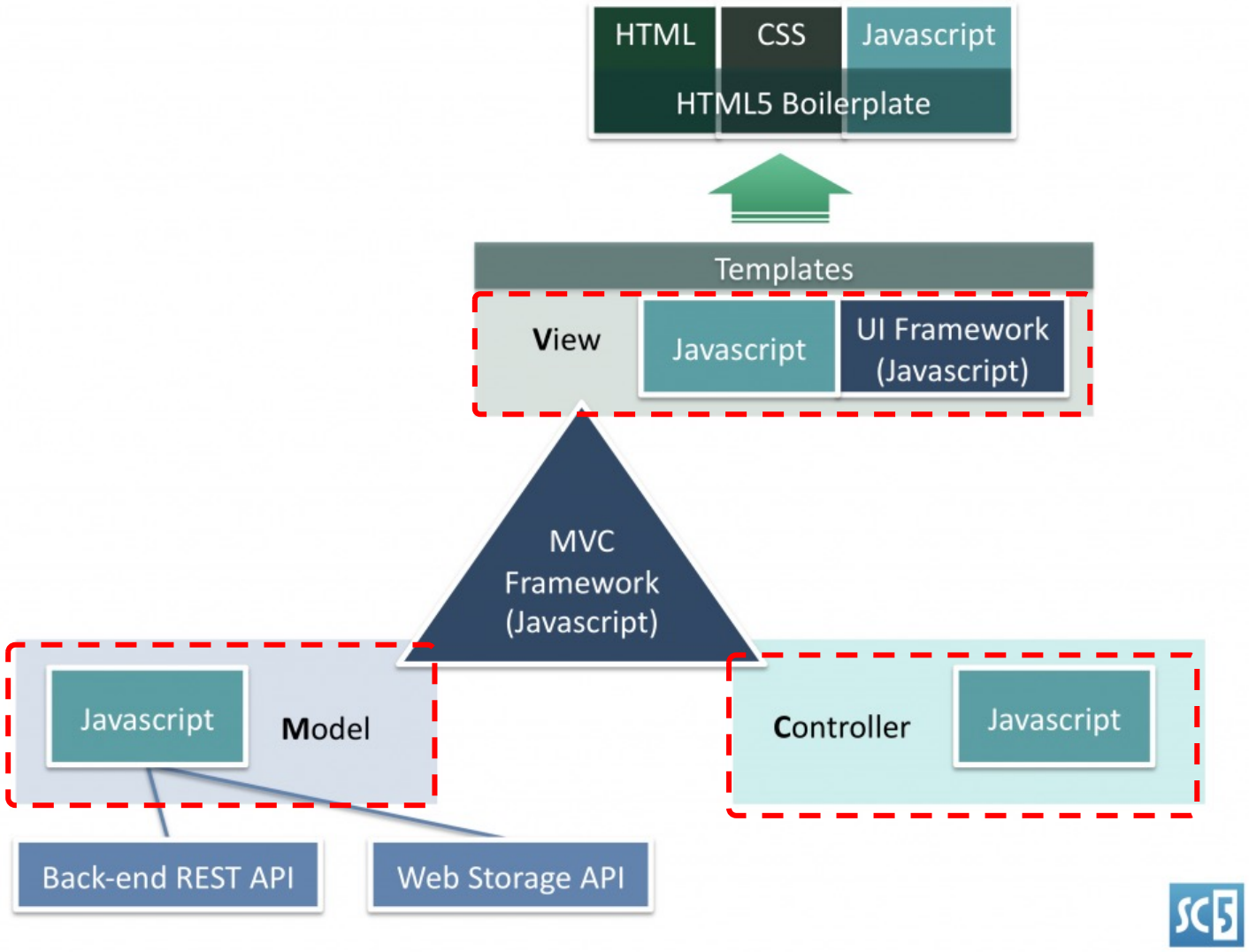
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Mobile Web App



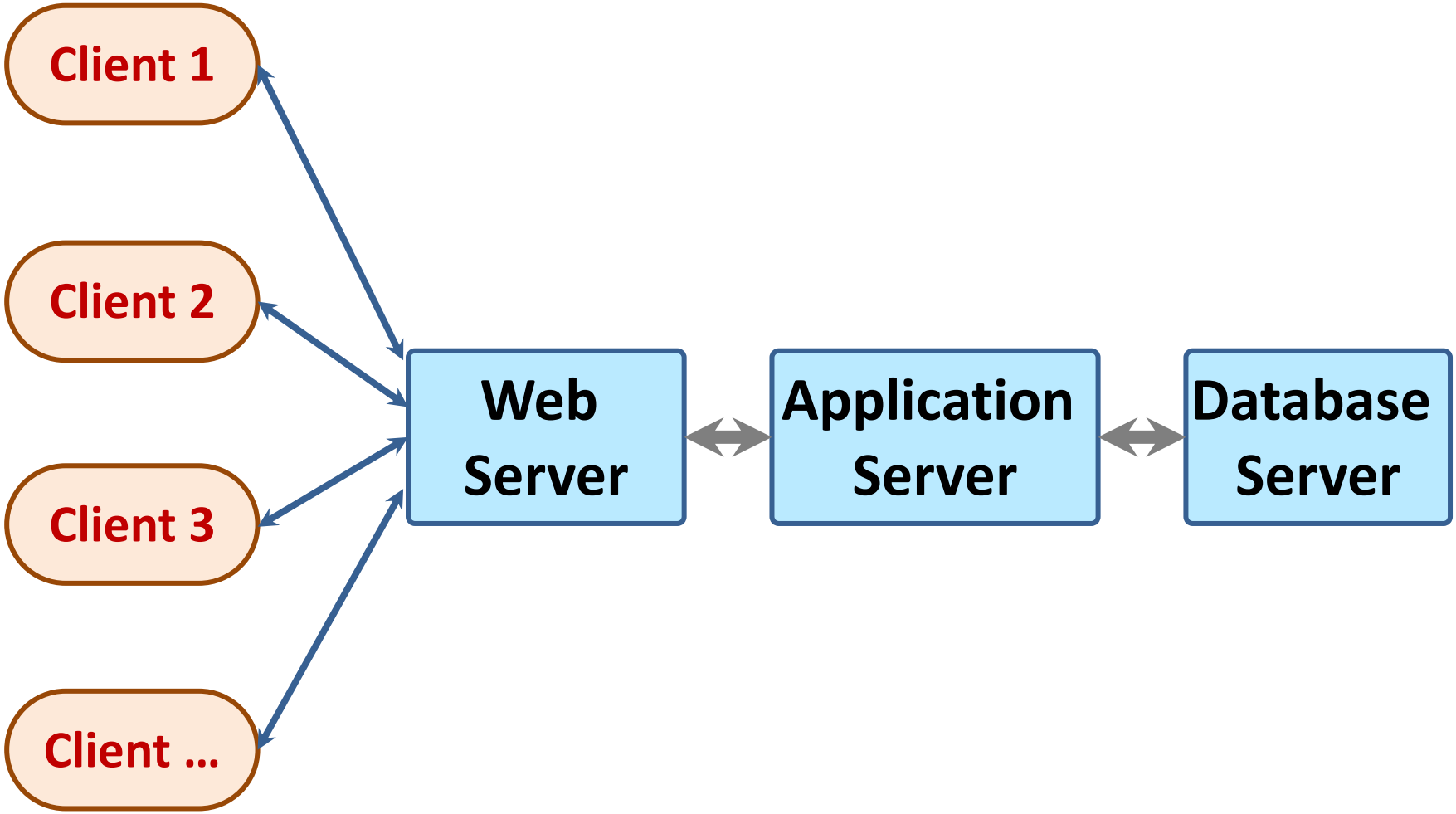
Source: Scott Preston, Learn HTML5 and JavaScript for iOS: Web Standards-based Apps for iPhone, iPad, and iPod touch, Apress, 2012

MVC Framework of Mobile Apps (HTML5, CSS3, JavaScript)



Source: http://sc5.io/blog/2012/02/anatomy-of-a-html5-app/

Multi-tier client-server architecture

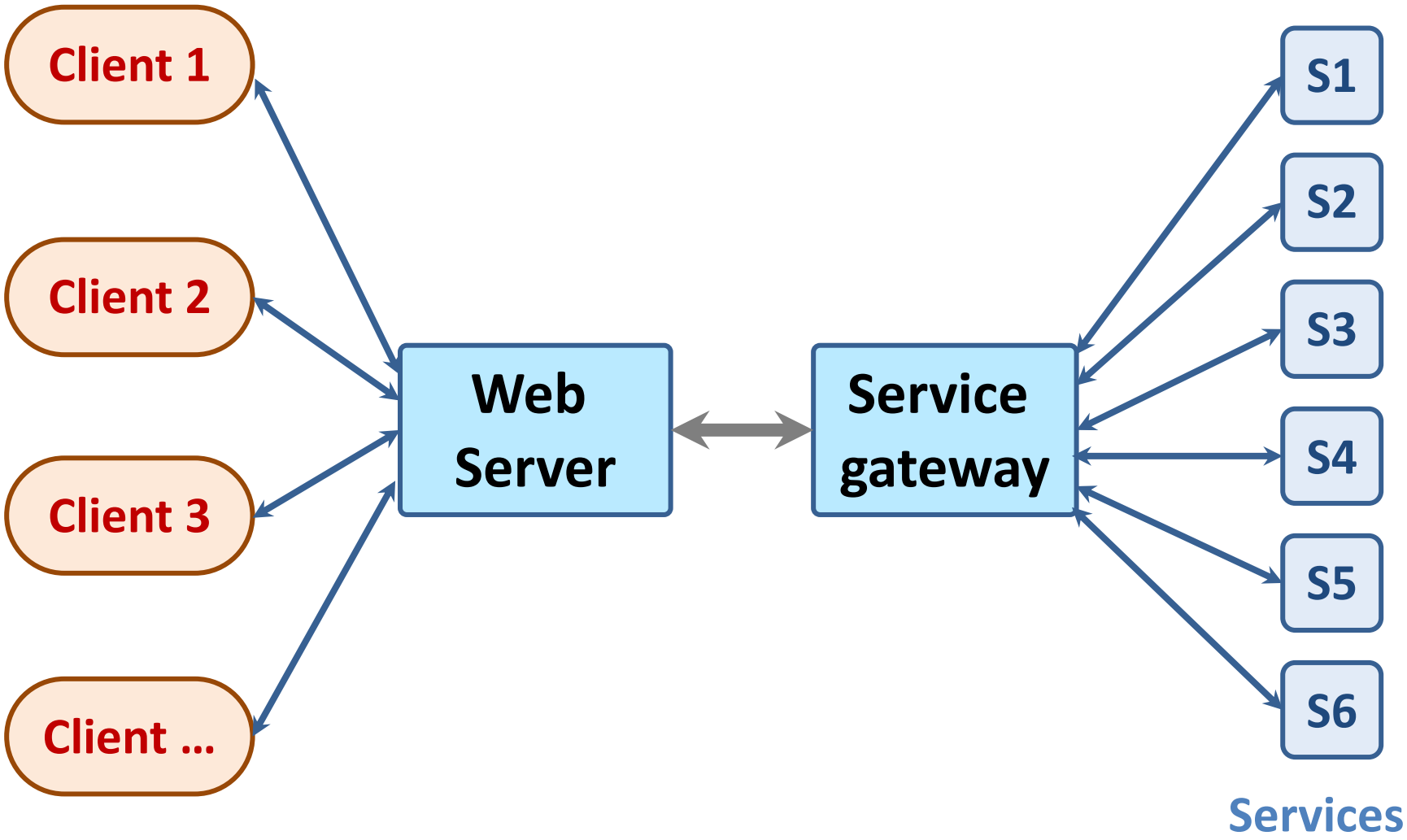


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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 users accessing it over the Internet, it is usually best 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

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Summary

- 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.

Summary

- 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.

Summary

- 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.

Summary

- 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.

References

- Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.
- Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.
- Titus Winters, Tom Manshreck, and Hyrum Wright (2020), Software Engineering at Google: Lessons Learned from Programming Over Time, O'Reilly Media.
- Project Management Institute (2021), A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Seventh Edition and The Standard for Project Management, PMI
- Project Management Institute (2017), A Guide to the Project Management Body of Knowledge (PMBOK Guide), Sixth Edition, Project Management Institute
- Project Management Institute (2017), Agile Practice Guide, Project Management Institute