Software Engineering

Software Architecture:
Architectural design, System decomposition, and Distribution architecture

1142SE05
MBA, IM, NTPU (M5010) (Spring 2026)
Wed 2, 3, 4 (9:10-12:00) (B3F17)

https://meet.google.com/ish-gzmy-pmo

Min-Yuh Day, Ph.D,
Professor and Director

Institute of Information Management, National Taipei University

https://web.ntpu.edu.tw/~myday

2026-04-01

Syllabus

Week	Date	Subject/Topics
1	2026/02/25	Introduction to Software Engineering
2	2026/03/04	Software Products and Project Management: Software product management and prototyping with Generative AI and Agentic AI
3	2026/03/11	Agile Software Engineering: Agile methods, Scrum, and Extreme Programming
4	2026/03/18	Case Study on Software Engineering I
5	2026/03/25	Features, Scenarios, and Stories
6	2026/04/01	Software Architecture: Architectural design, System decomposition, and Distribution architecture

Syllabus

Week	Date	Subject/Topics
7	2026/04/08	Cloud-Based Software: Virtualization and containers, Everything as a service, Software as a service
8	2026/04/15	Midterm Project Report
9	2026/04/22	Cloud Computing and Cloud Software Architecture
10	2026/04/29	Microservices Architecture, RESTful services, Service deployment
11	2026/05/06	Case Study on Software Engineering II
12	2026/05/13	Security and Privacy; Reliable Programming; Testing: Functional testing, Test automation, Test-driven development, and Code reviews

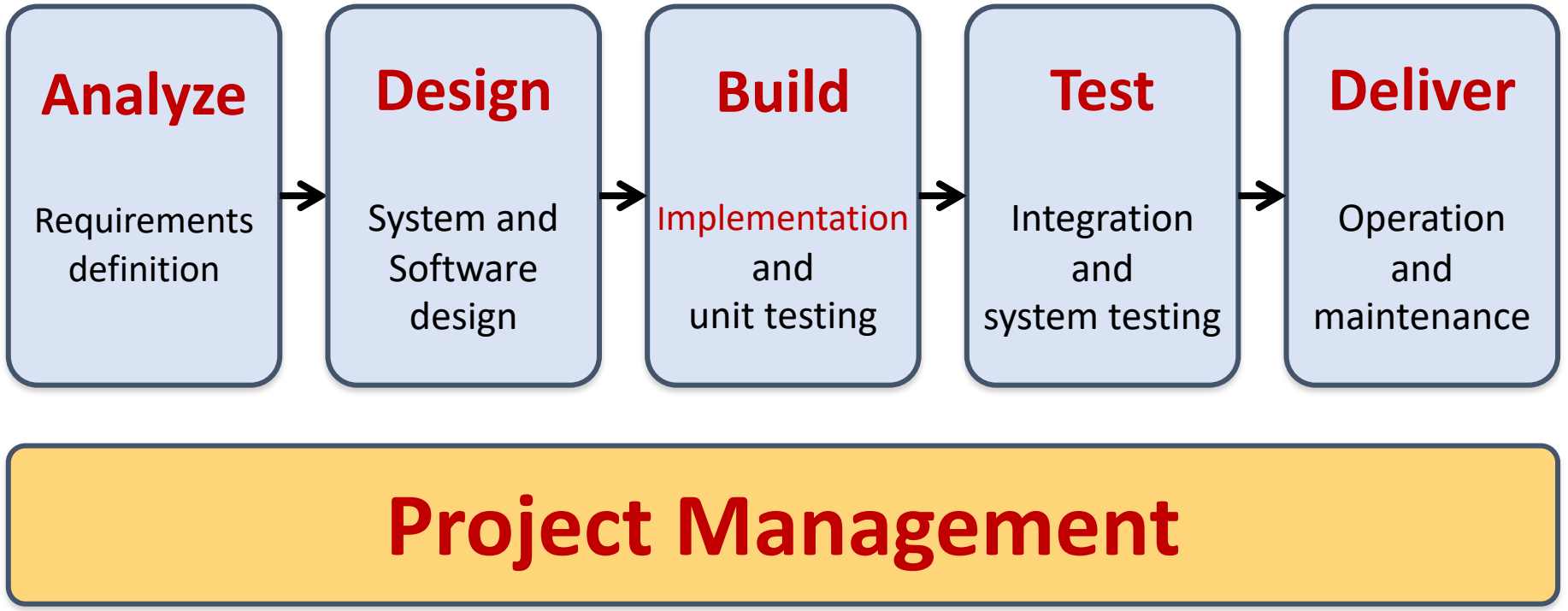
Syllabus

Week	Date	Subject/Topics
13	2026/05/20	Industry Practices of Software Engineering
14	2026/05/27	DevOps and Code Management: Code management and DevOps automation
15	2026/06/03	Final Project Report I
16	2026/06/10	Final Project Report II

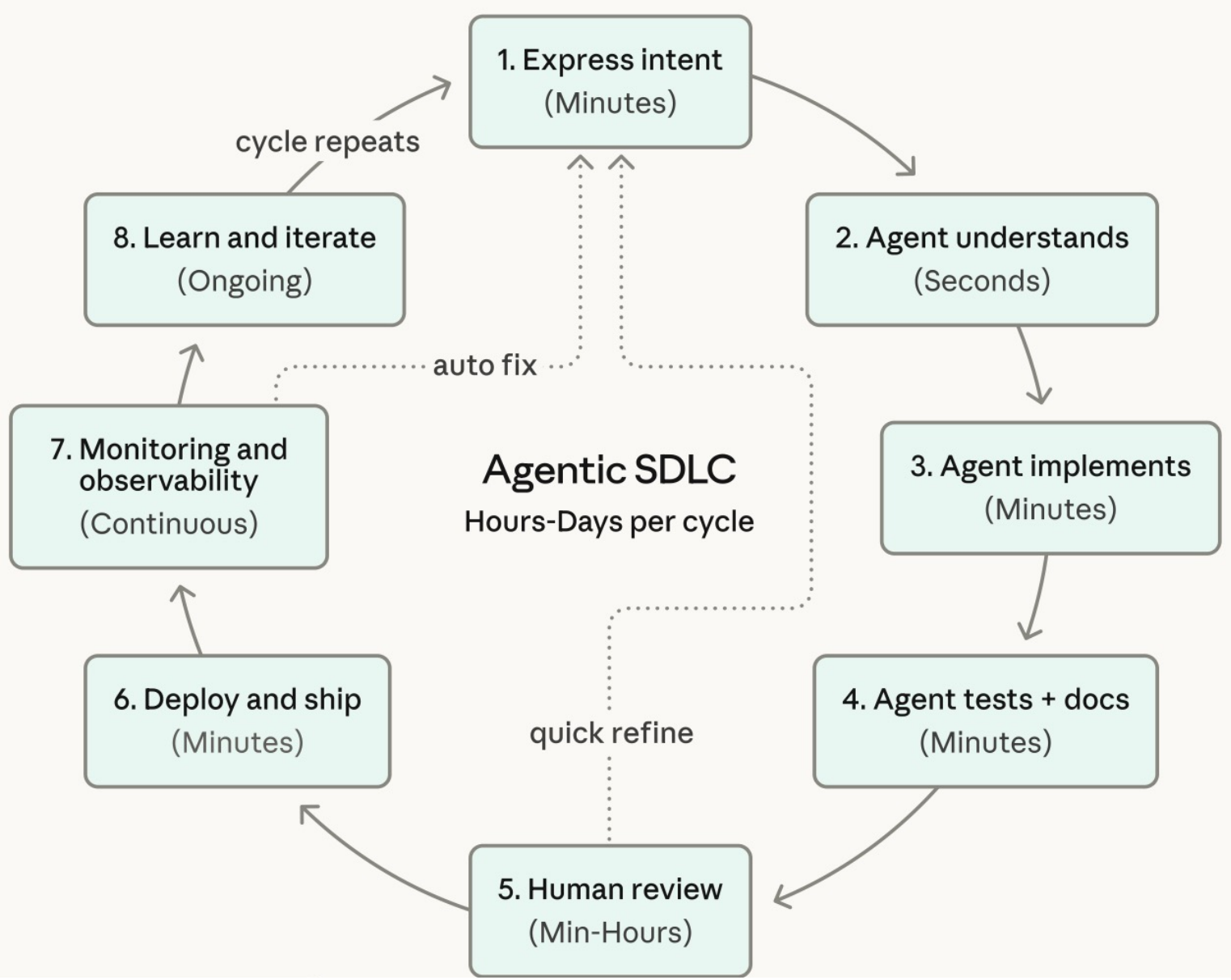
Software Architecture:

Architectural design, System decomposition, and Distribution architecture

Software Engineering
and
Project Management



Agentic Coding Software Development Lifecycle



Source: Anthropic (2026), 2026 Agentic CodingTrends Report: How coding agents are reshaping software development, https://resources.anthropic.com/hubfs/2026%20Agentic%20Coding%20Trends%20Report.pdf

Information Management

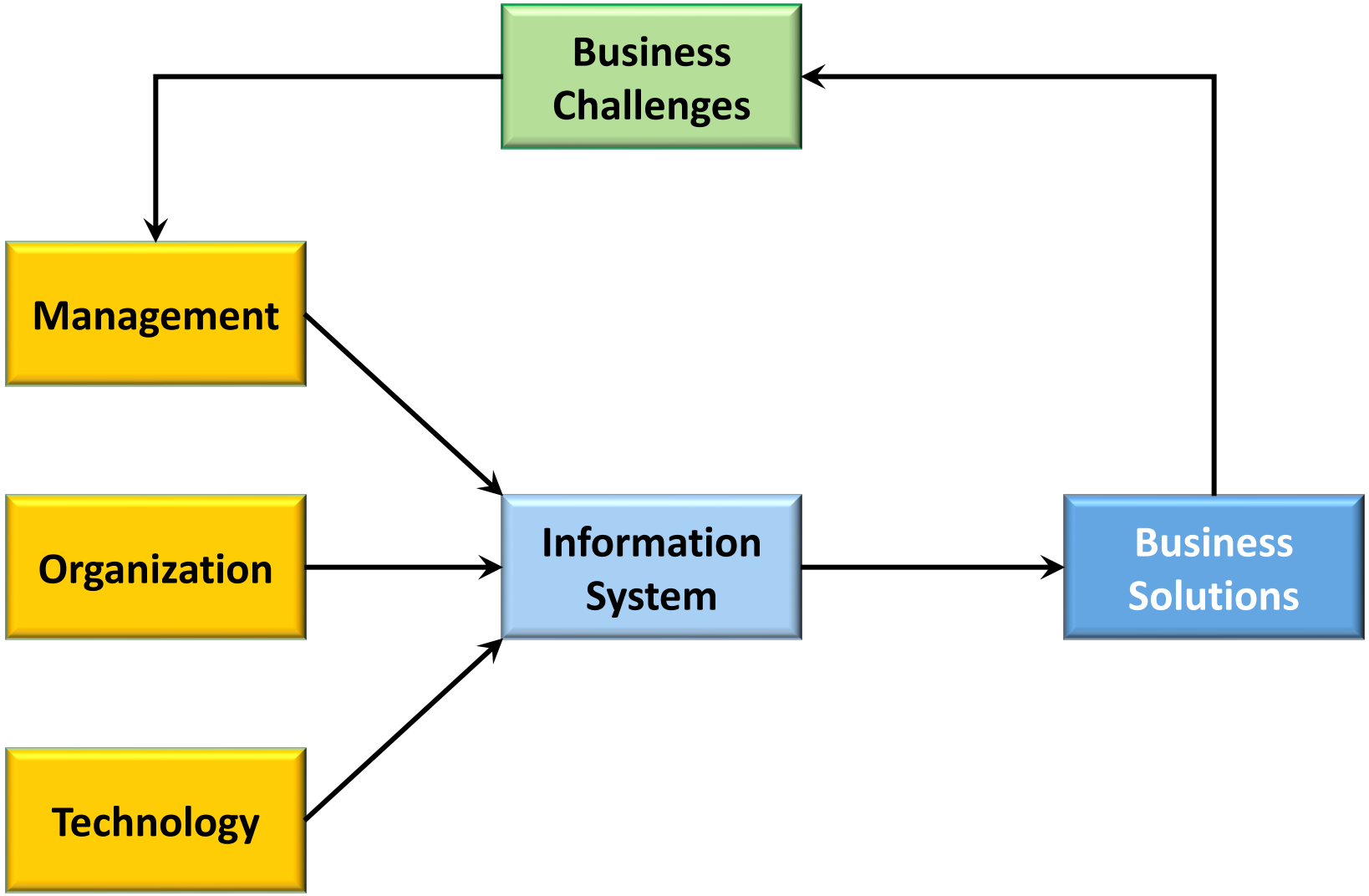
Management Information Systems (MIS)

Information Systems

Information Management (MIS)
Information Systems

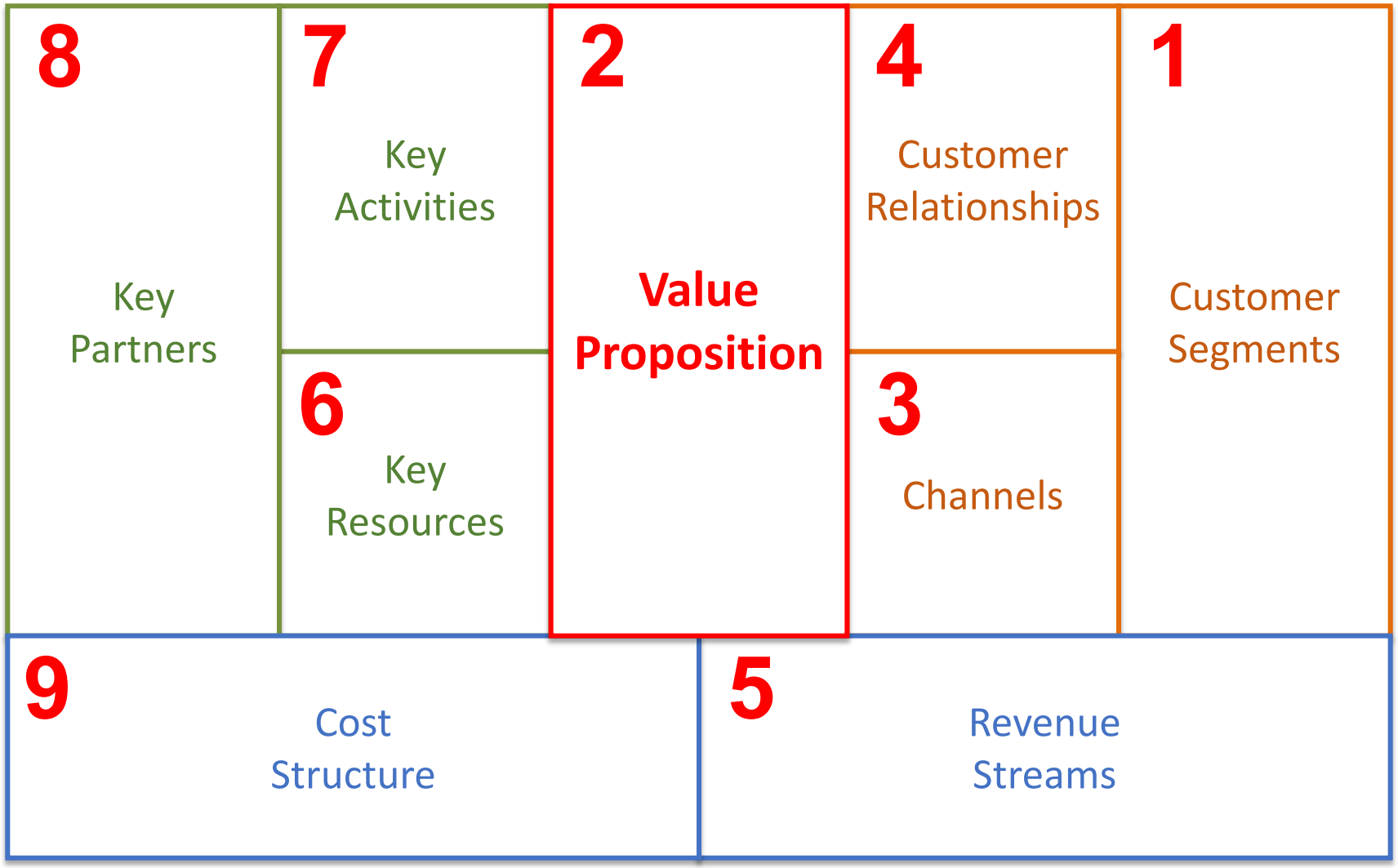
Source: Kenneth C. Laudon & Jane P. Laudon (2014), Management Information Systems: Managing the Digital Firm, Thirteenth Edition, Pearson.

Fundamental MIS Concepts



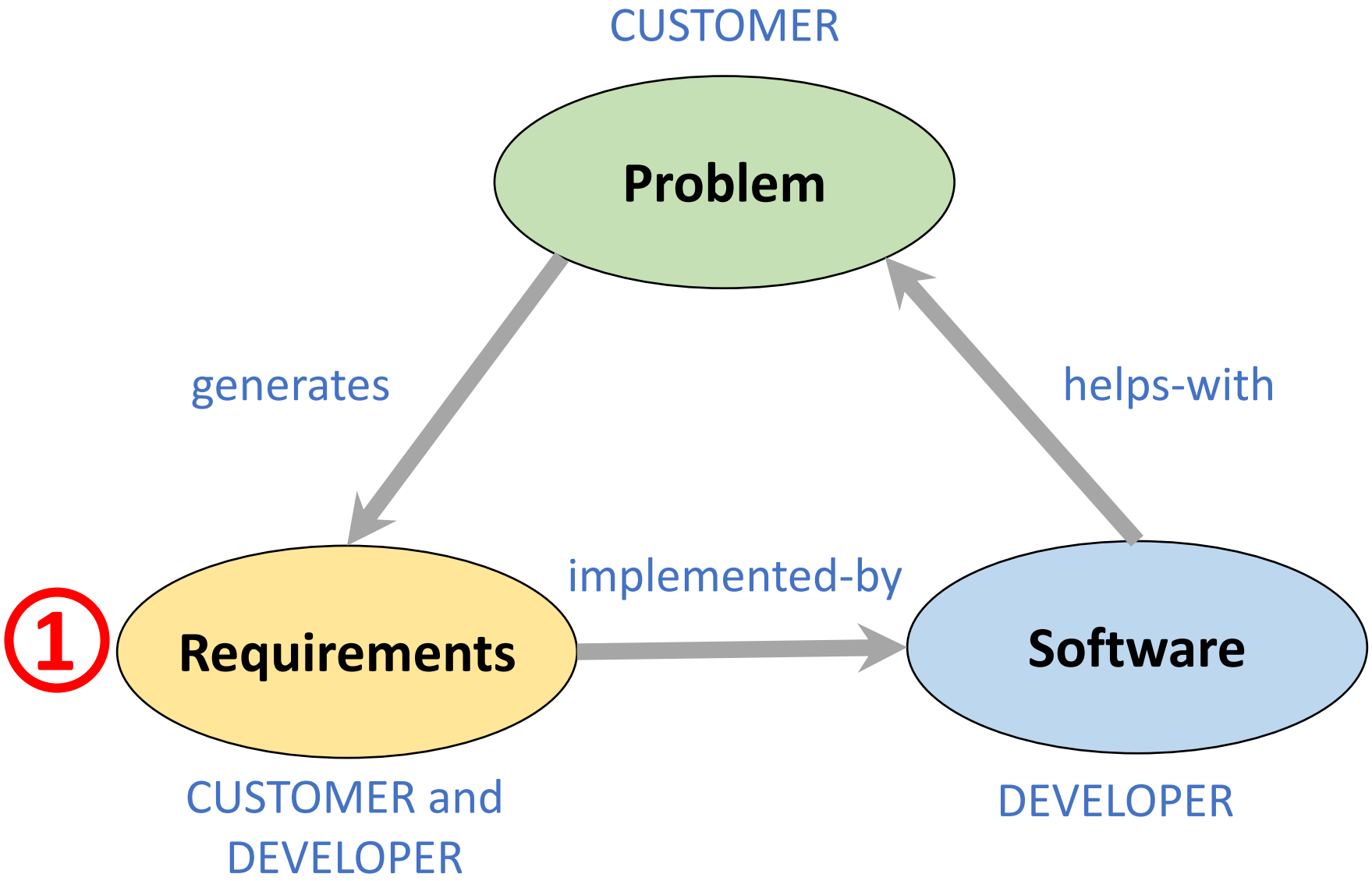
Source: Kenneth C. Laudon & Jane P. Laudon (2014), Management Information Systems: Managing the Digital Firm, Thirteenth Edition, Pearson.

Business Model



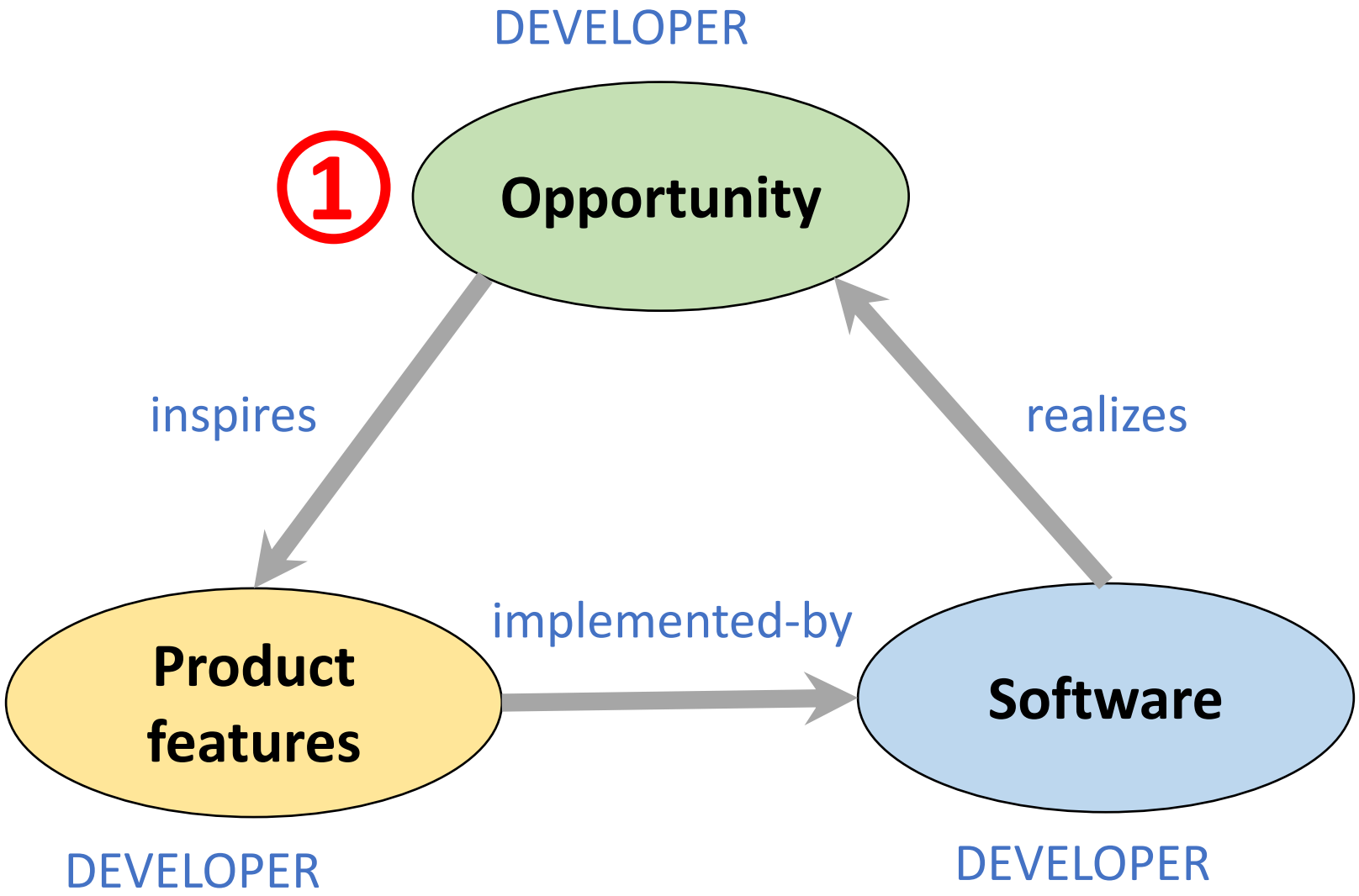
Source: Alexander Osterwalder & Yves Pigneur, Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers, Wiley, 2010.

Project-based software engineering



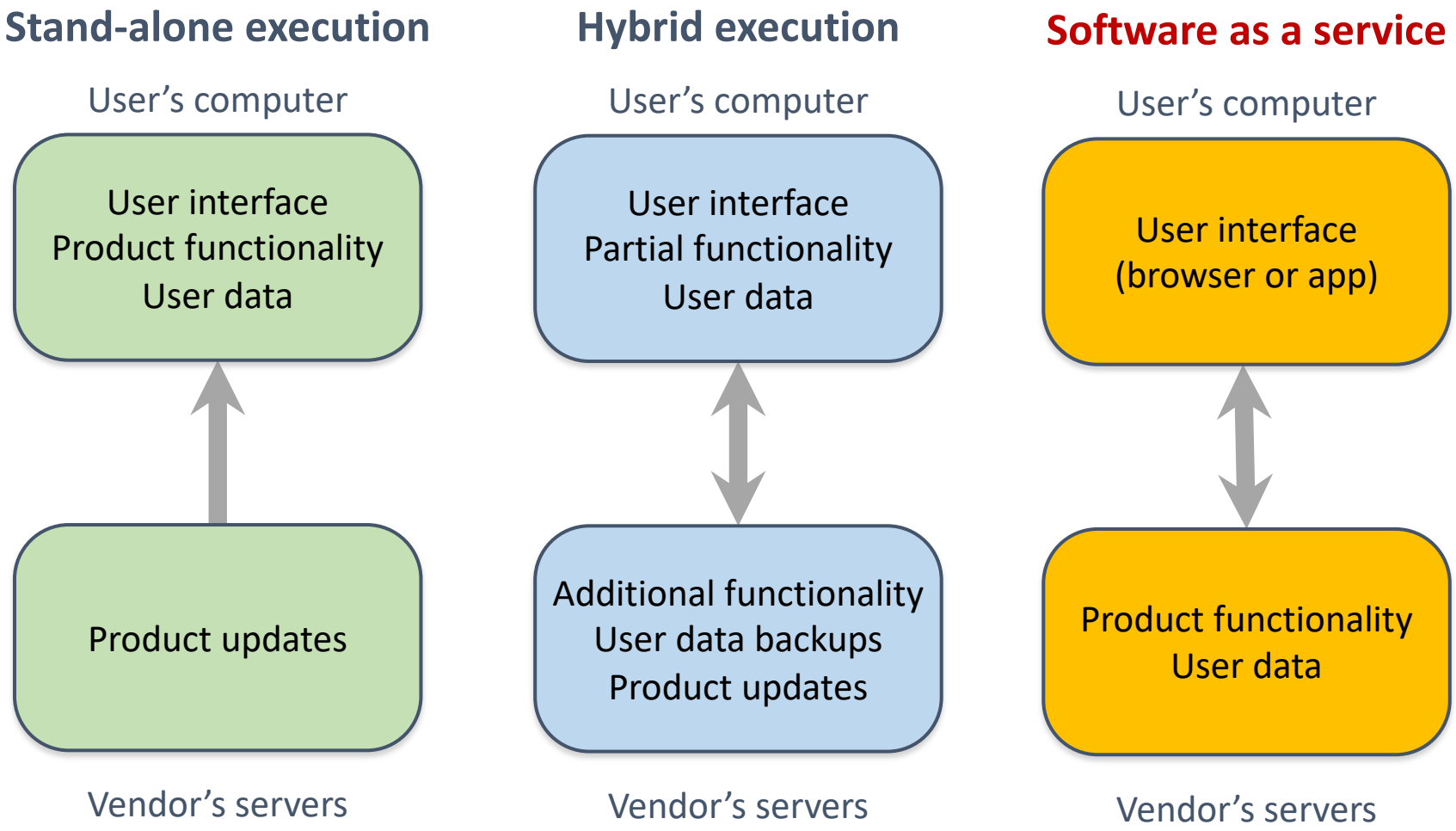
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Product software engineering



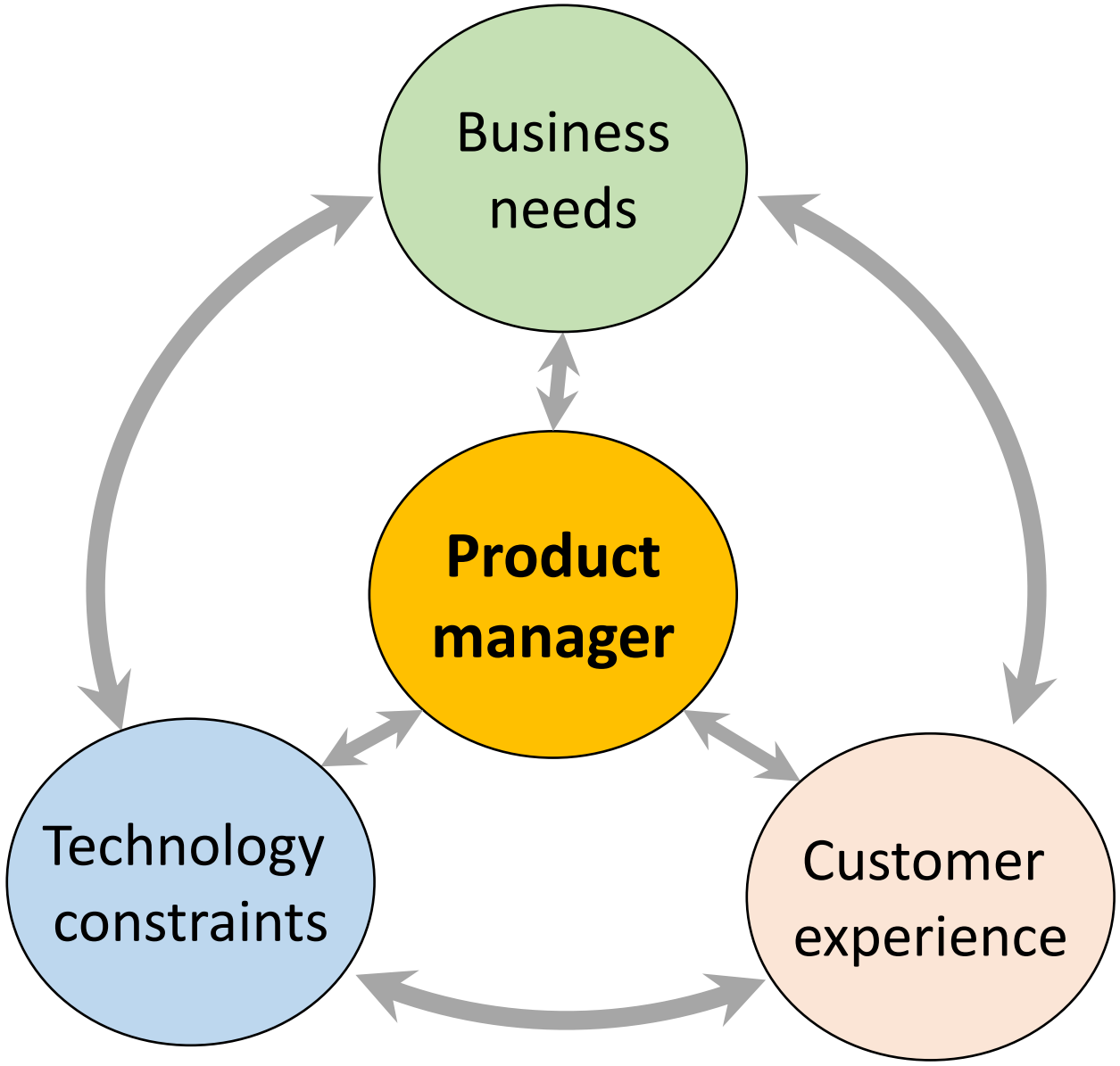
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software execution models



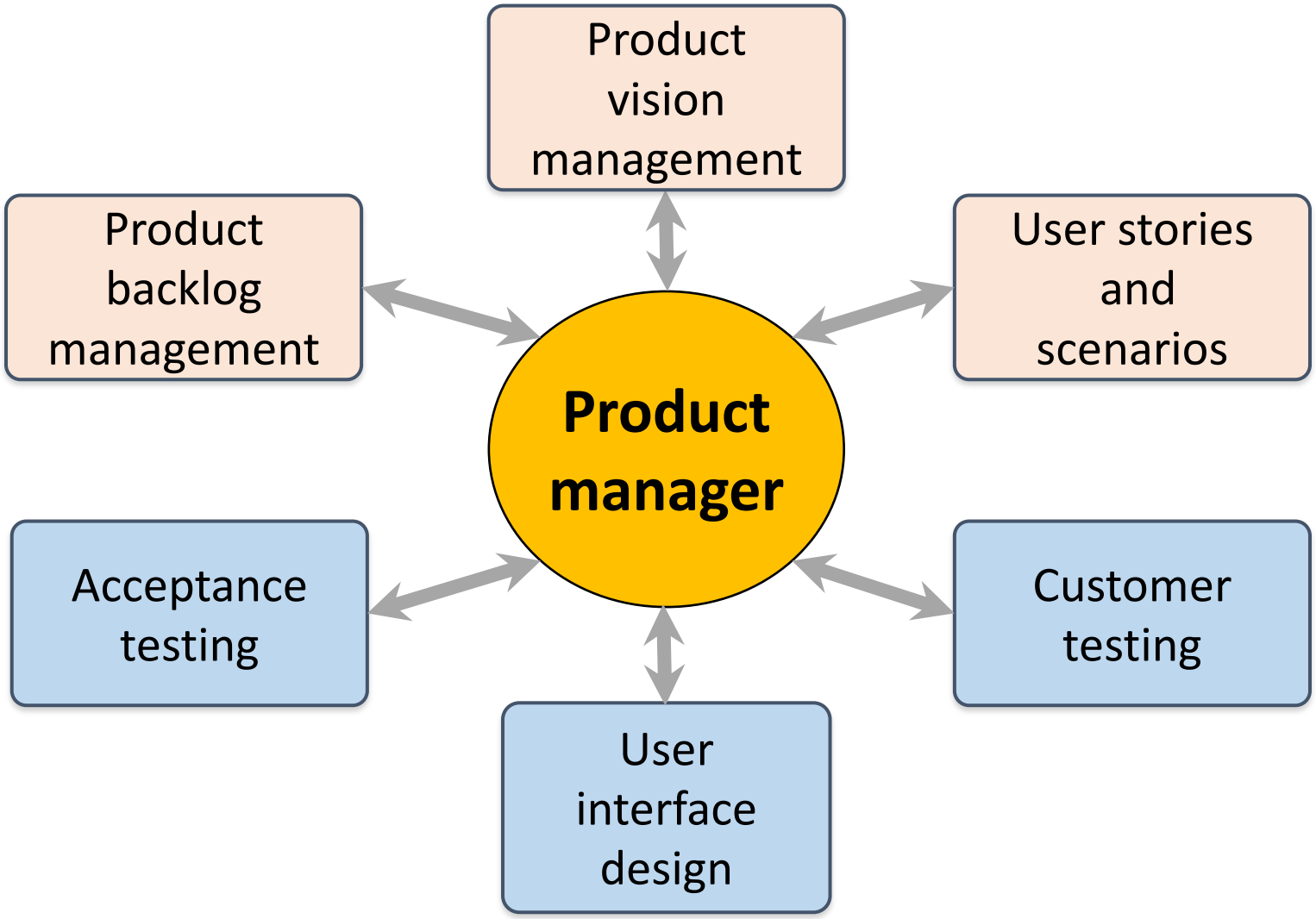
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Product management concerns



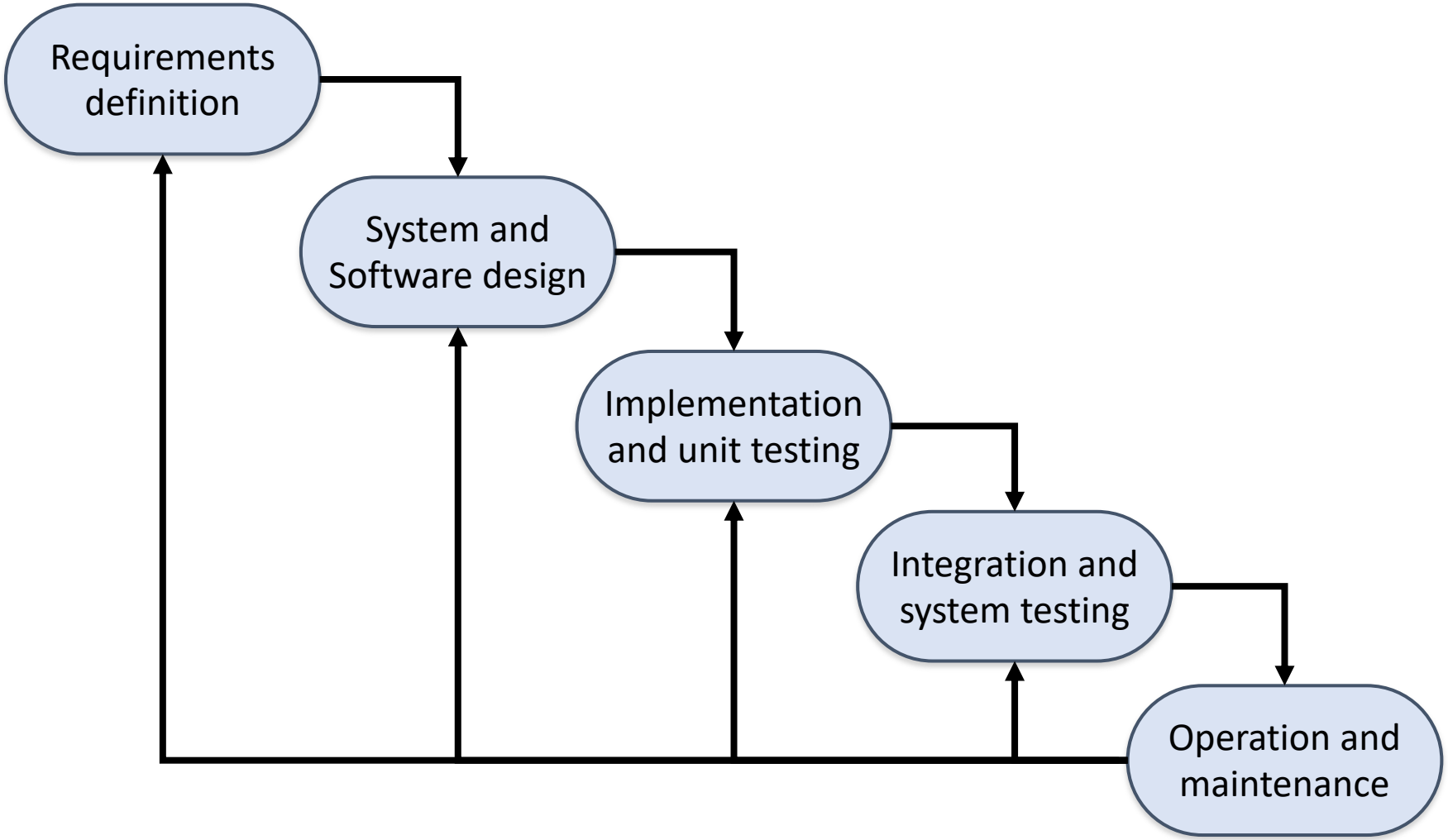
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Technical interactions of product managers



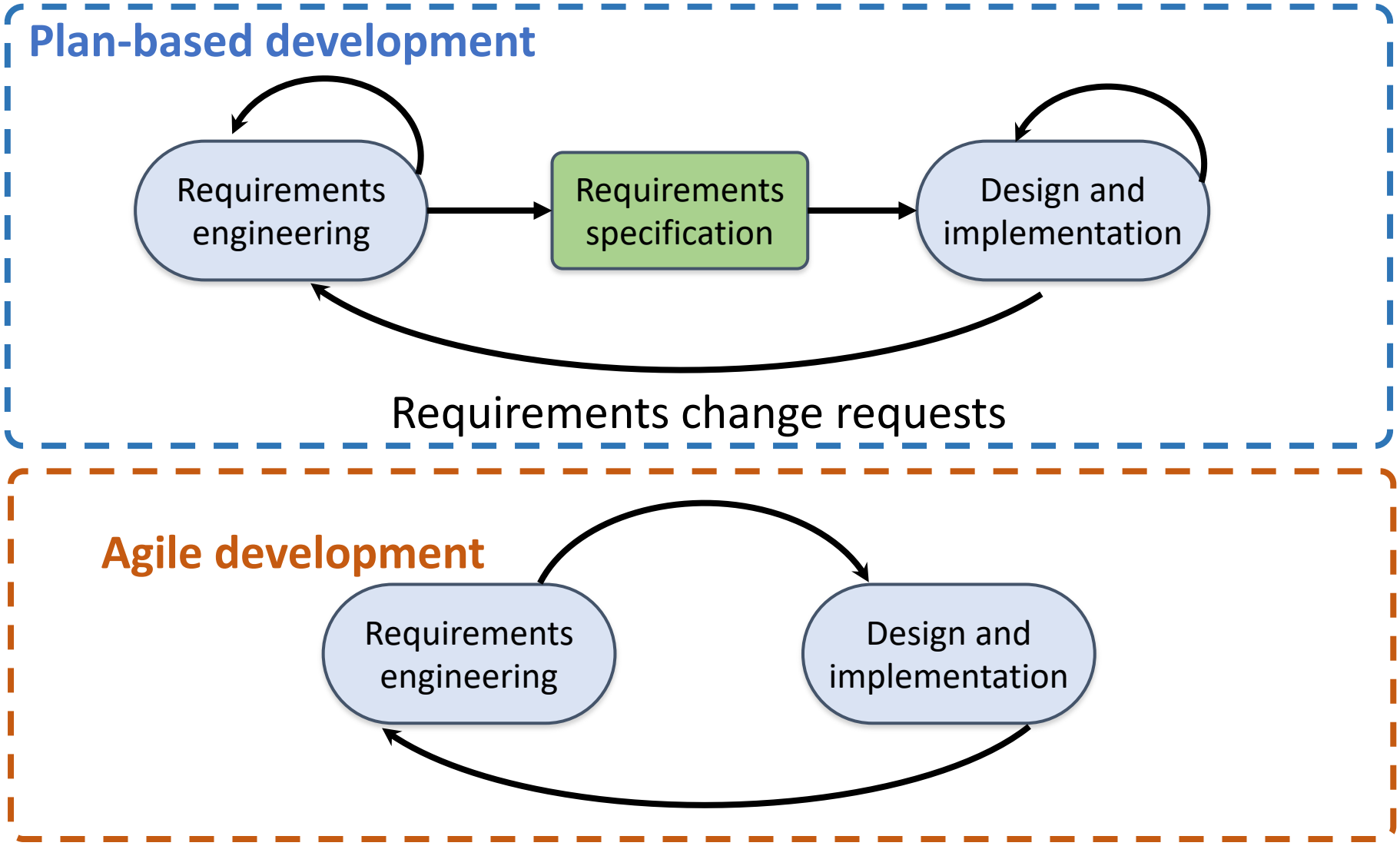
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software Development Life Cycle (SDLC)
The waterfall model



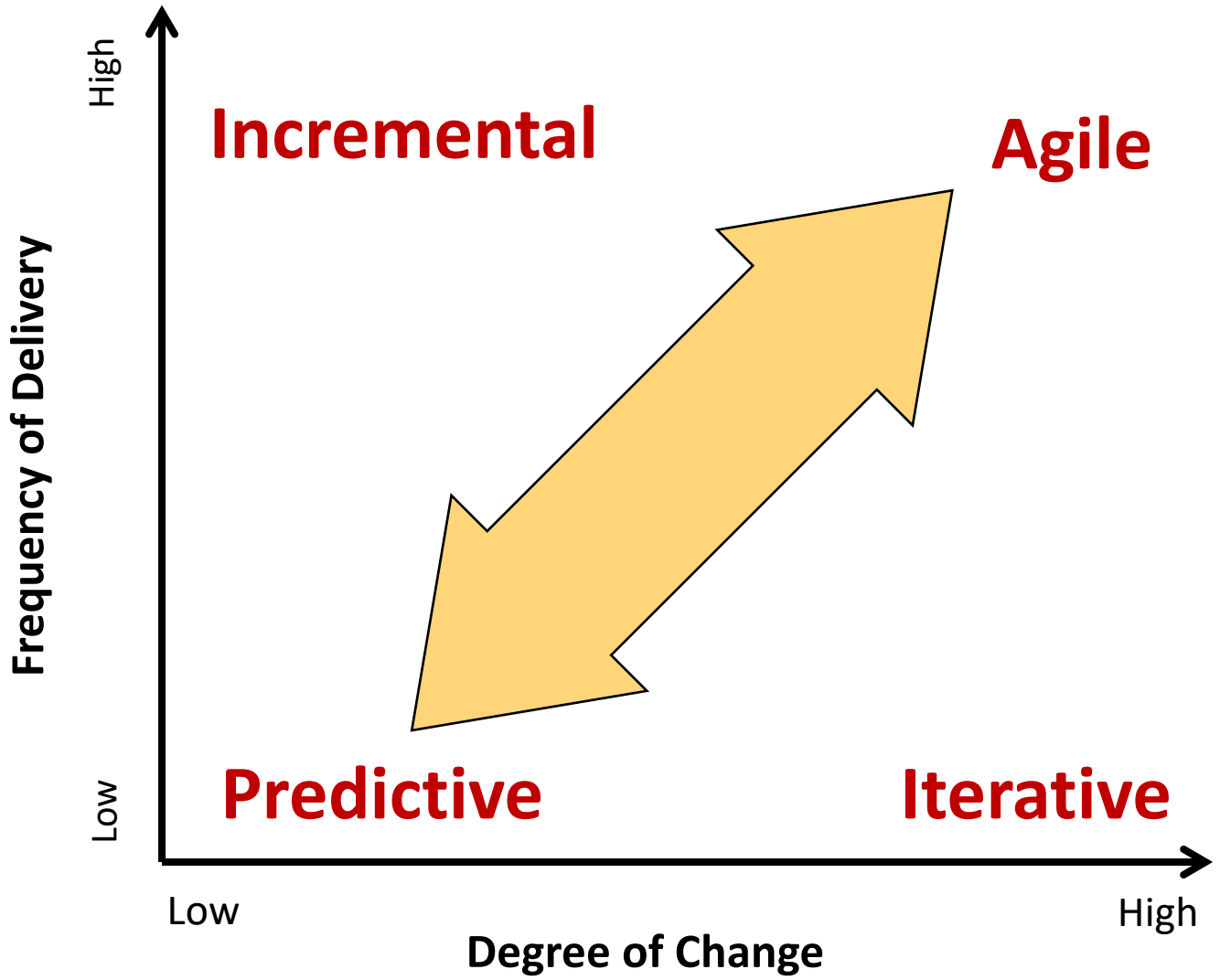
Source: Ian Sommerville (2015), Software Engineering, 10th Edition, Pearson.

Plan-based and Agile development



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

The Continuum of Life Cycles

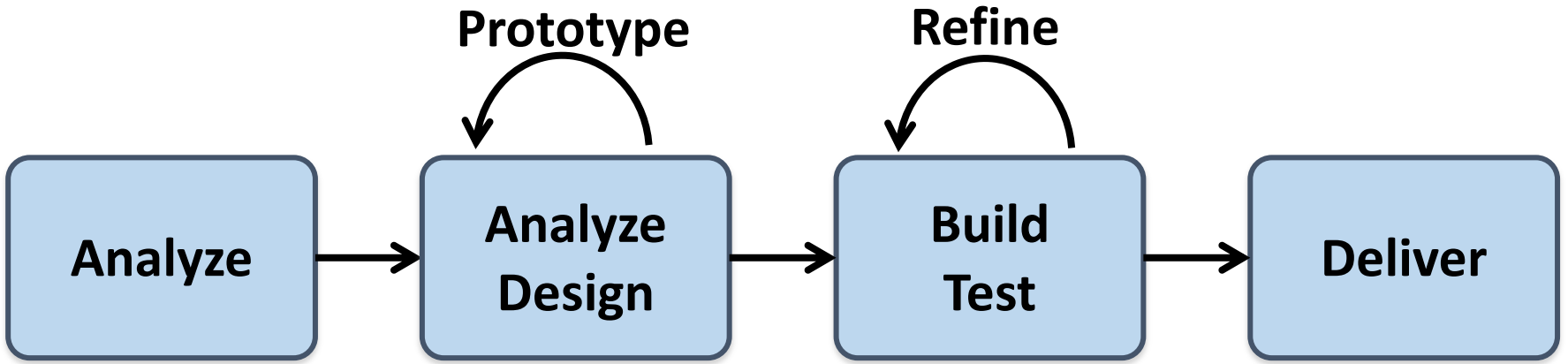


Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Predictive Life Cycle

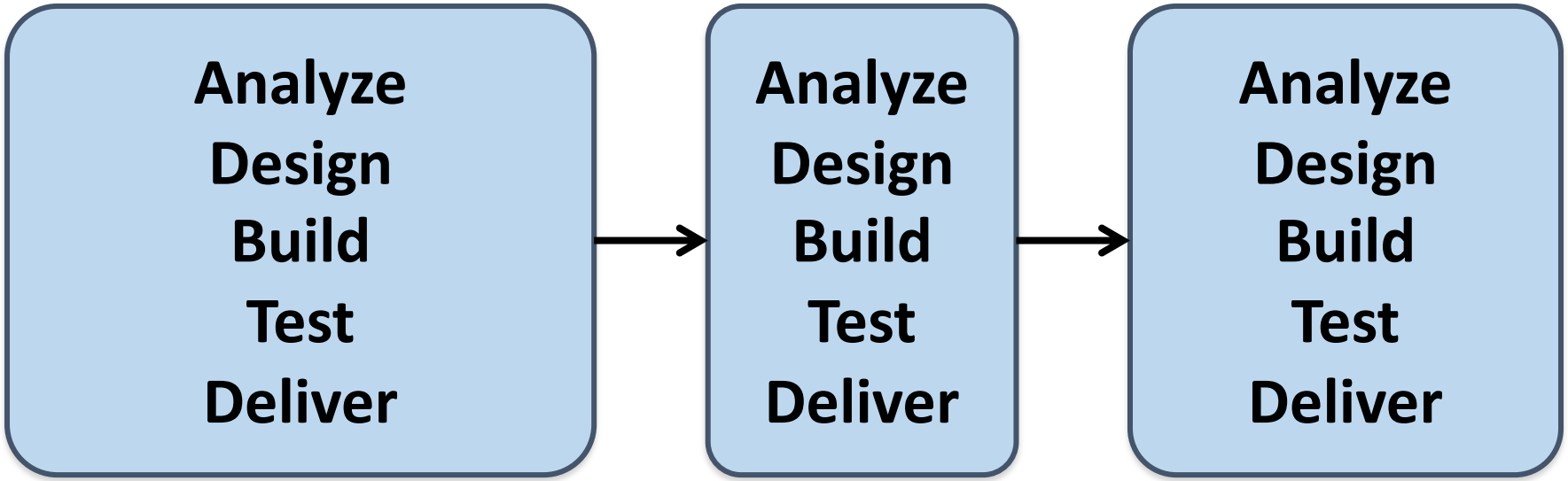
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Iterative Life Cycle



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

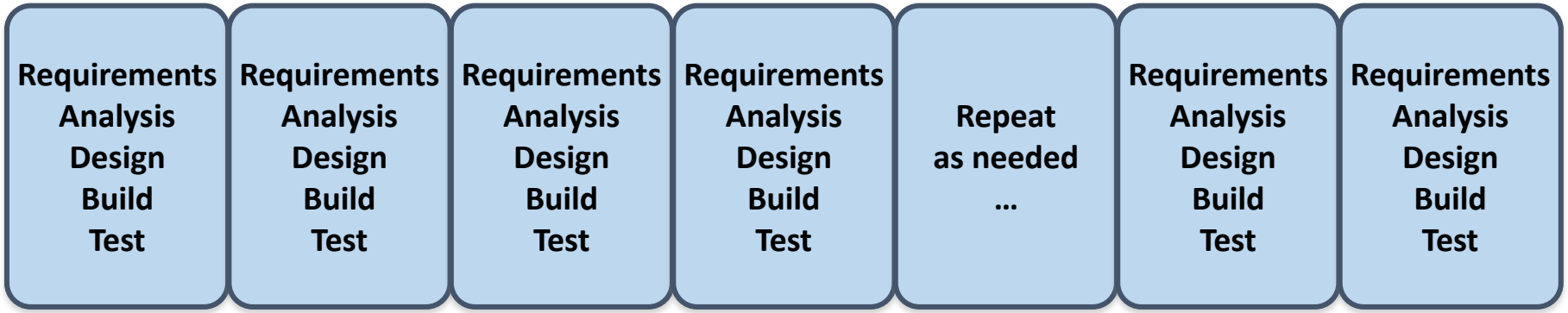
A Life Cycle of Varying-Sized Increments



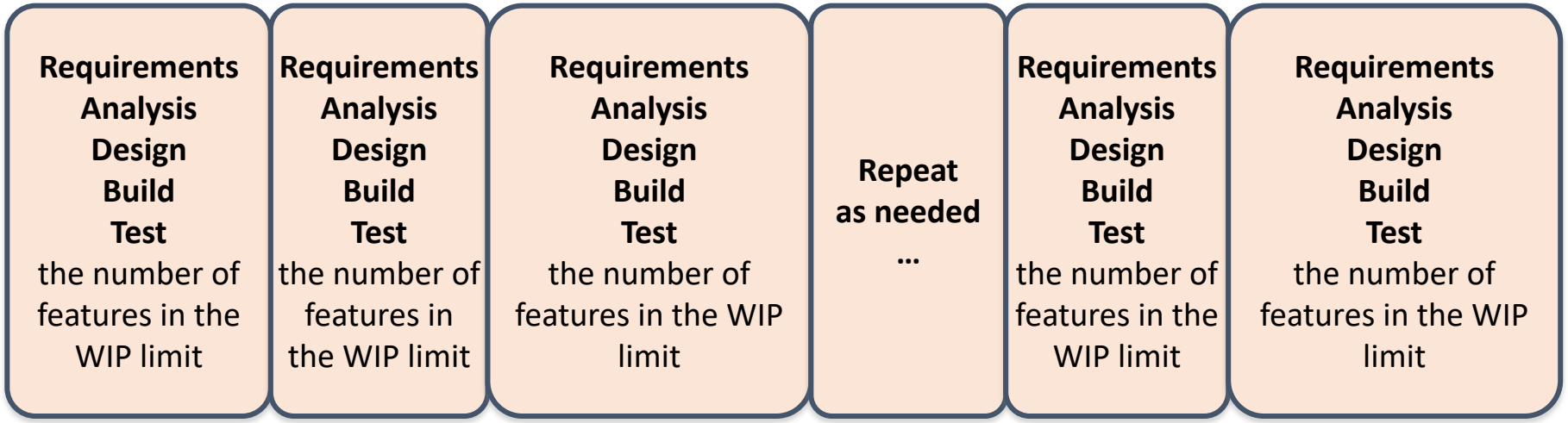
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

Iteration-Based and Flow-Based Agile Life Cycles

Iteration-Based Agile

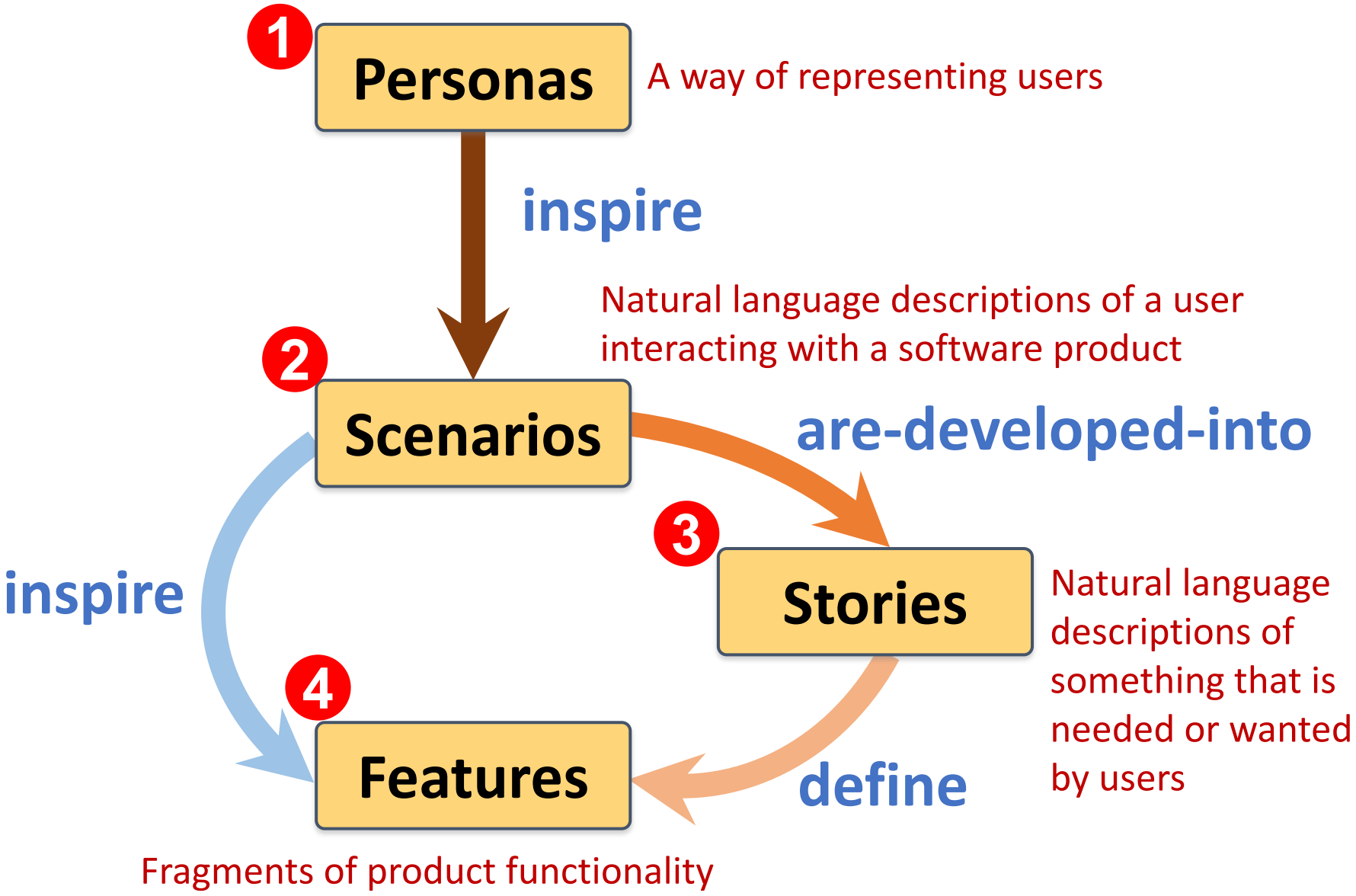


Flow-Based Agile



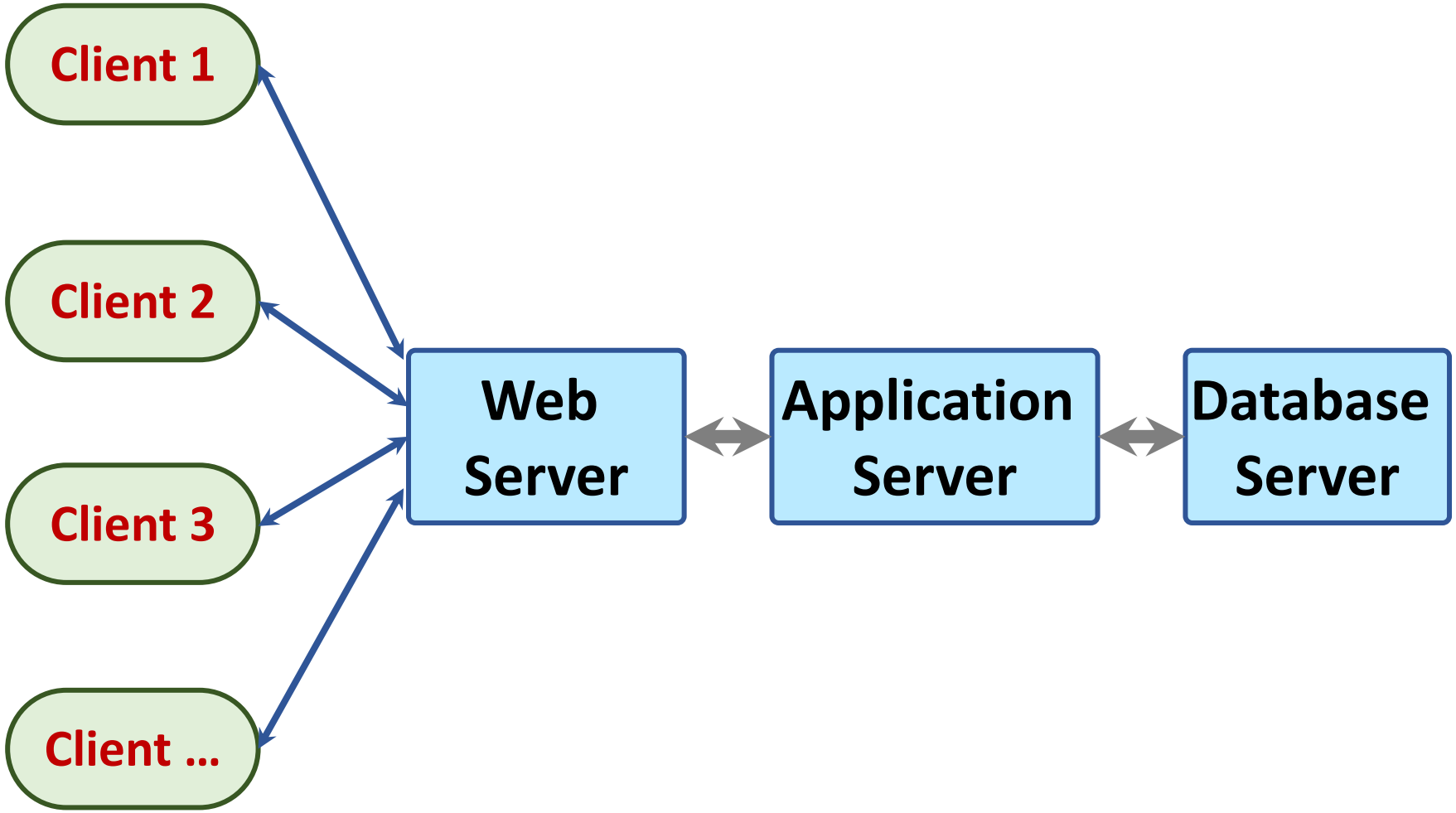
Source: Project Management Institute (2017), Agile Practice Guide, Project Management Institute

From personas to features



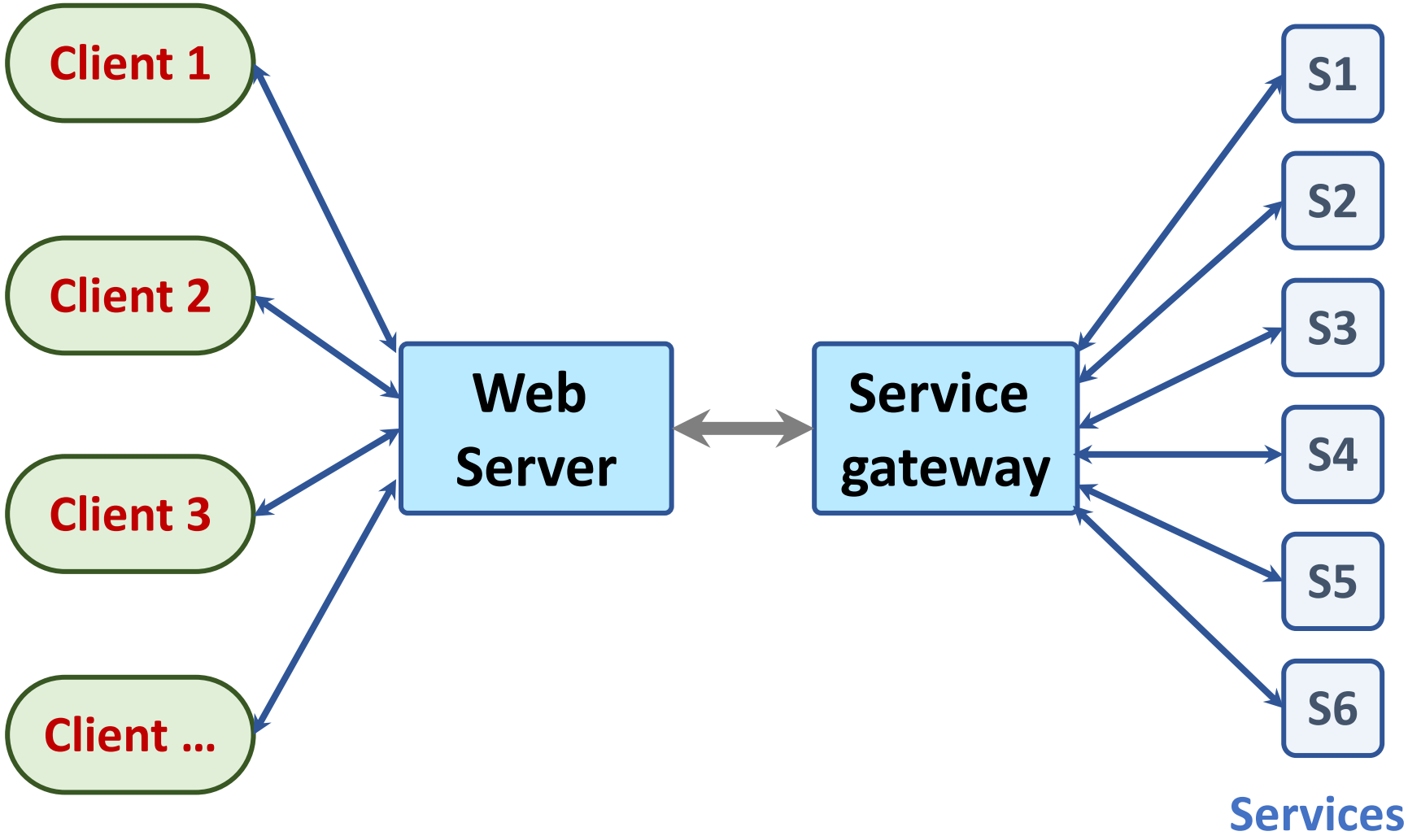
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Multi-tier client-server architecture

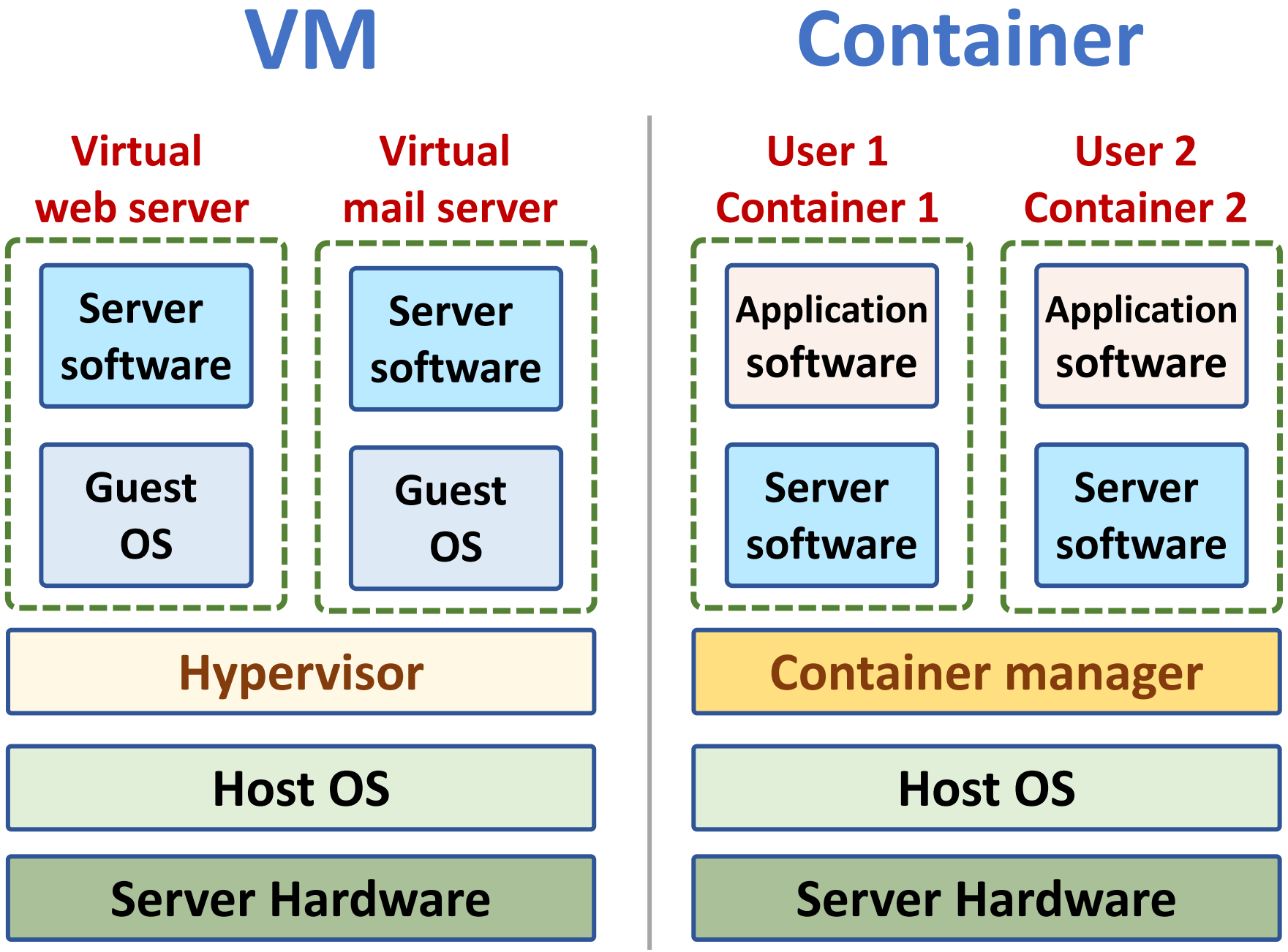


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Service-oriented Architecture

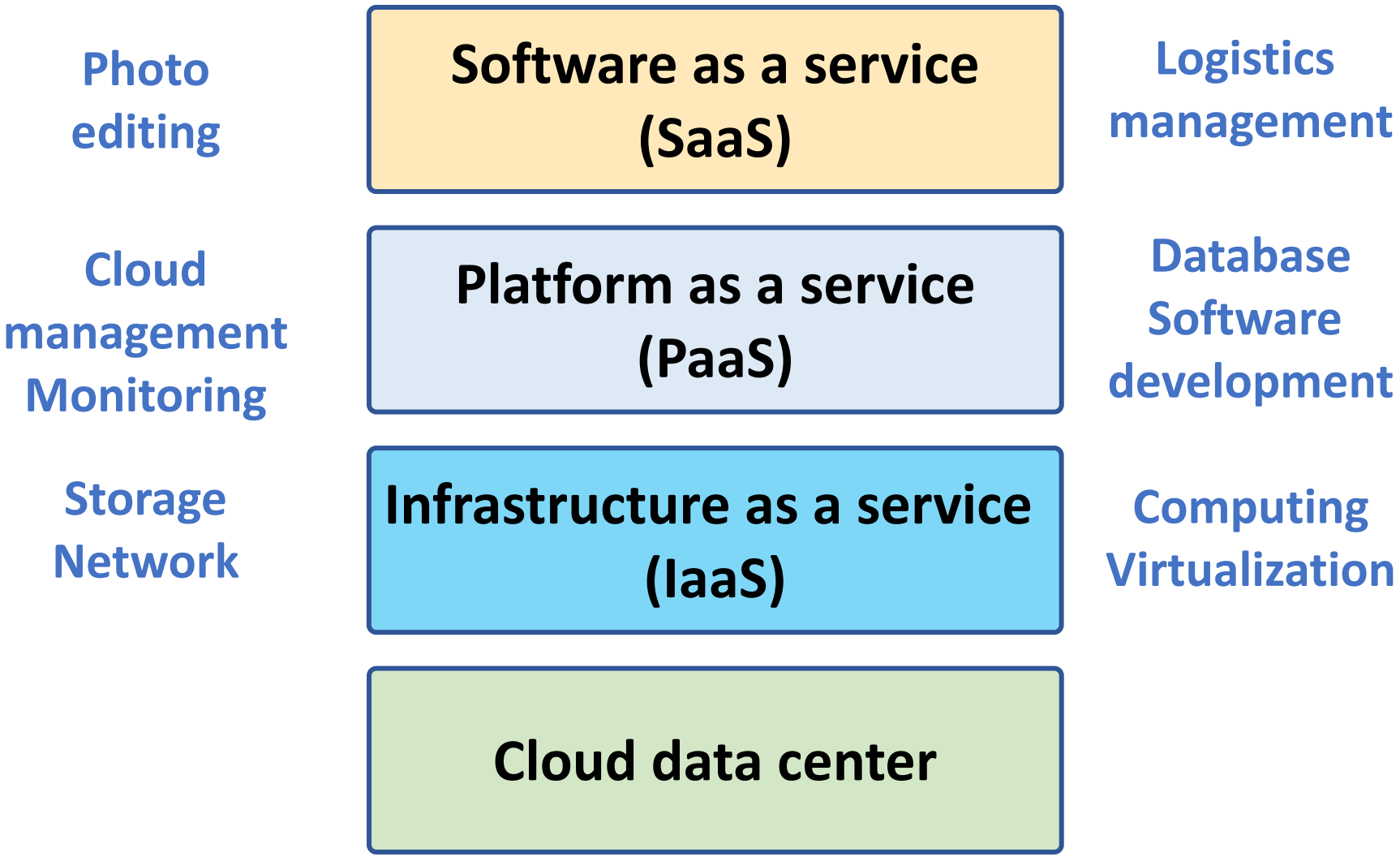


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.



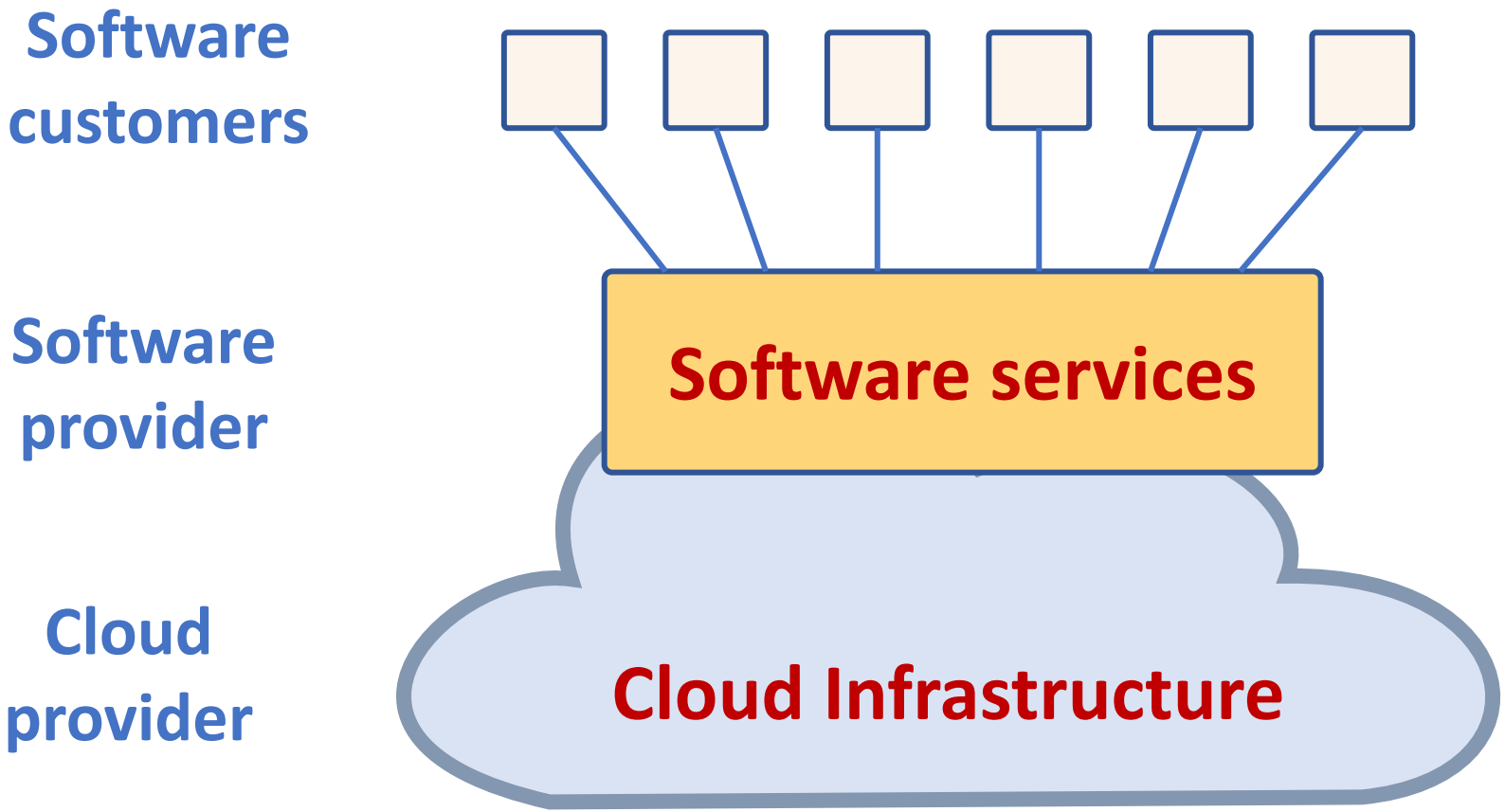
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Everything as a service



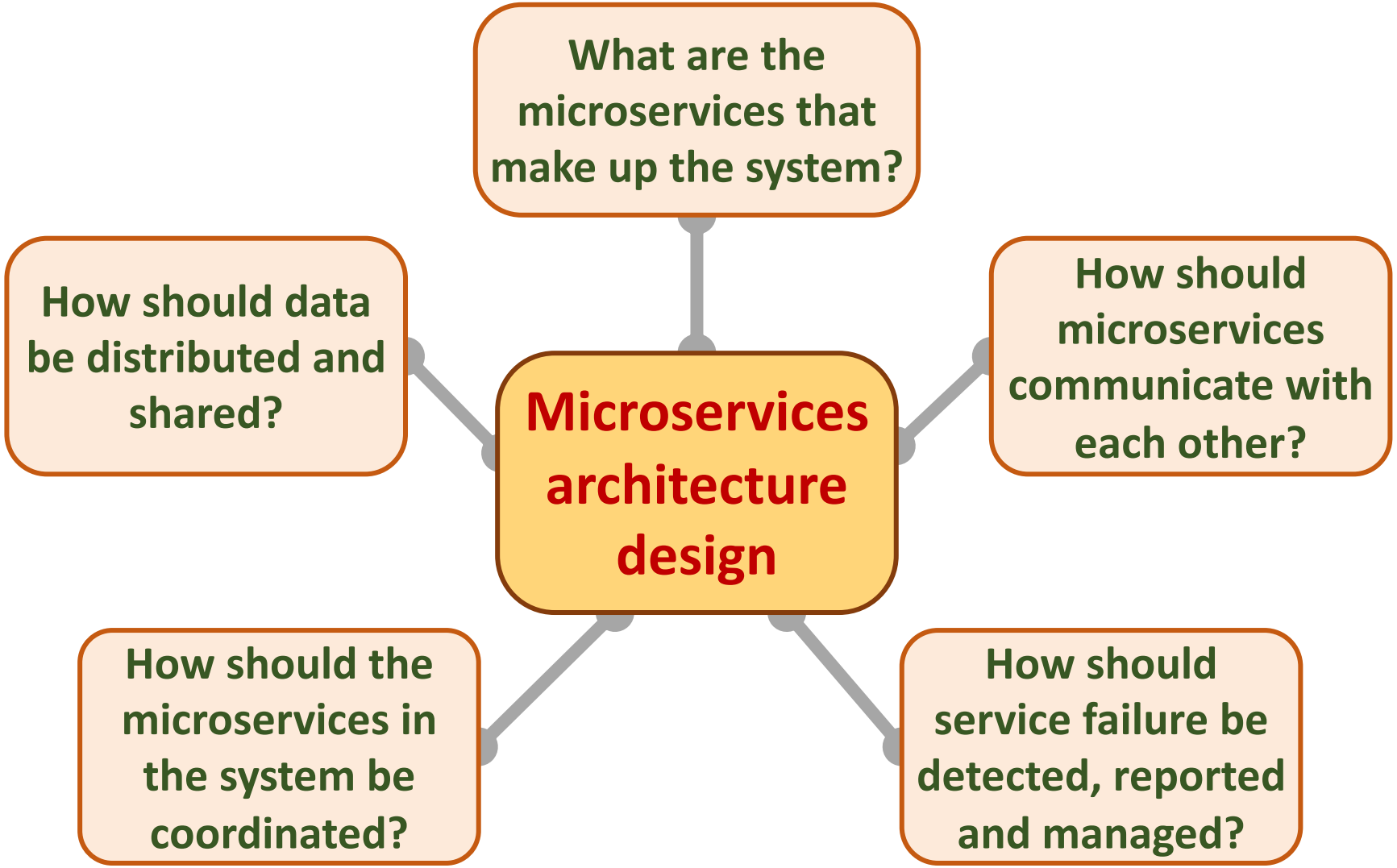
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software as a service



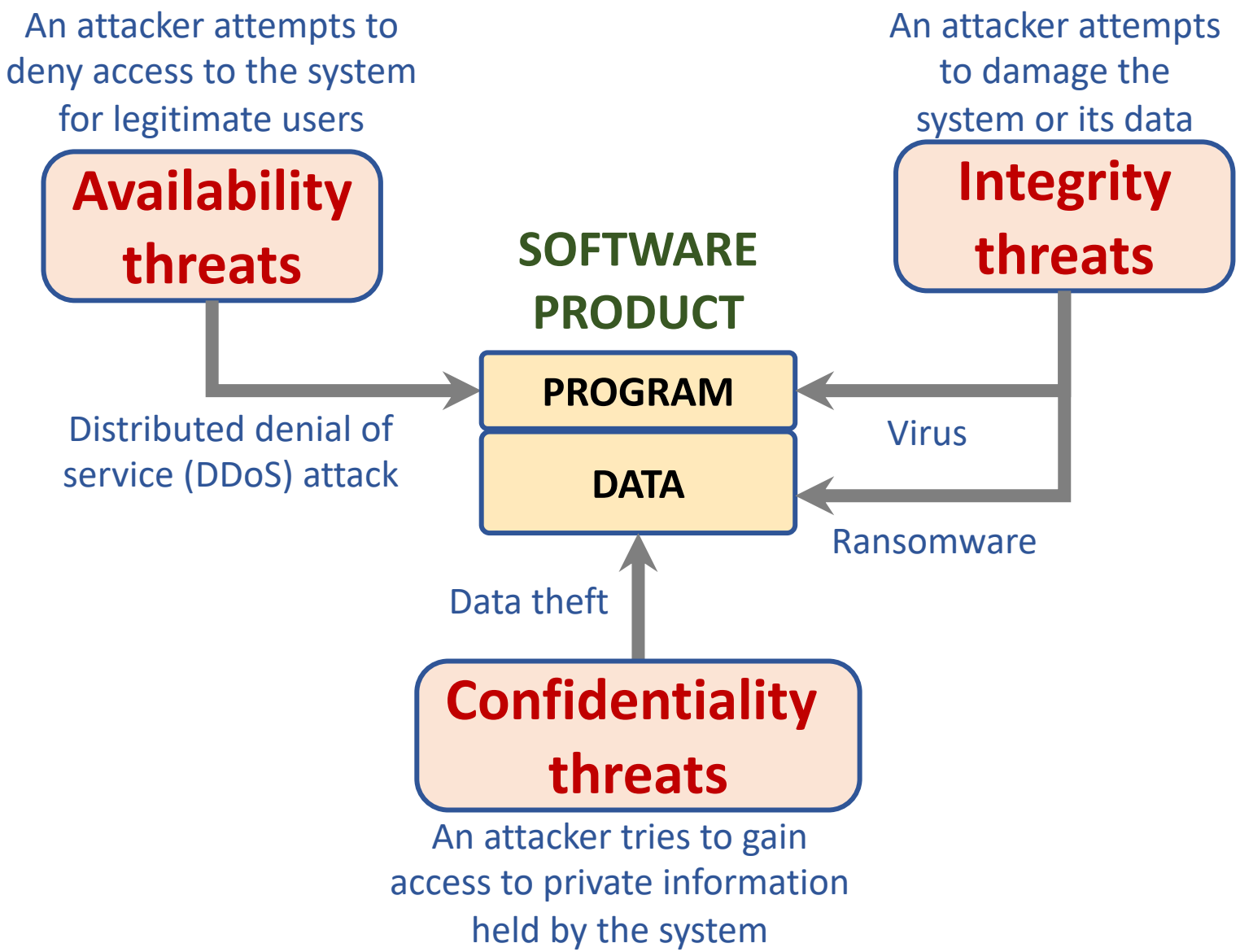
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Microservices architecture – key design questions



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

Types of security threat

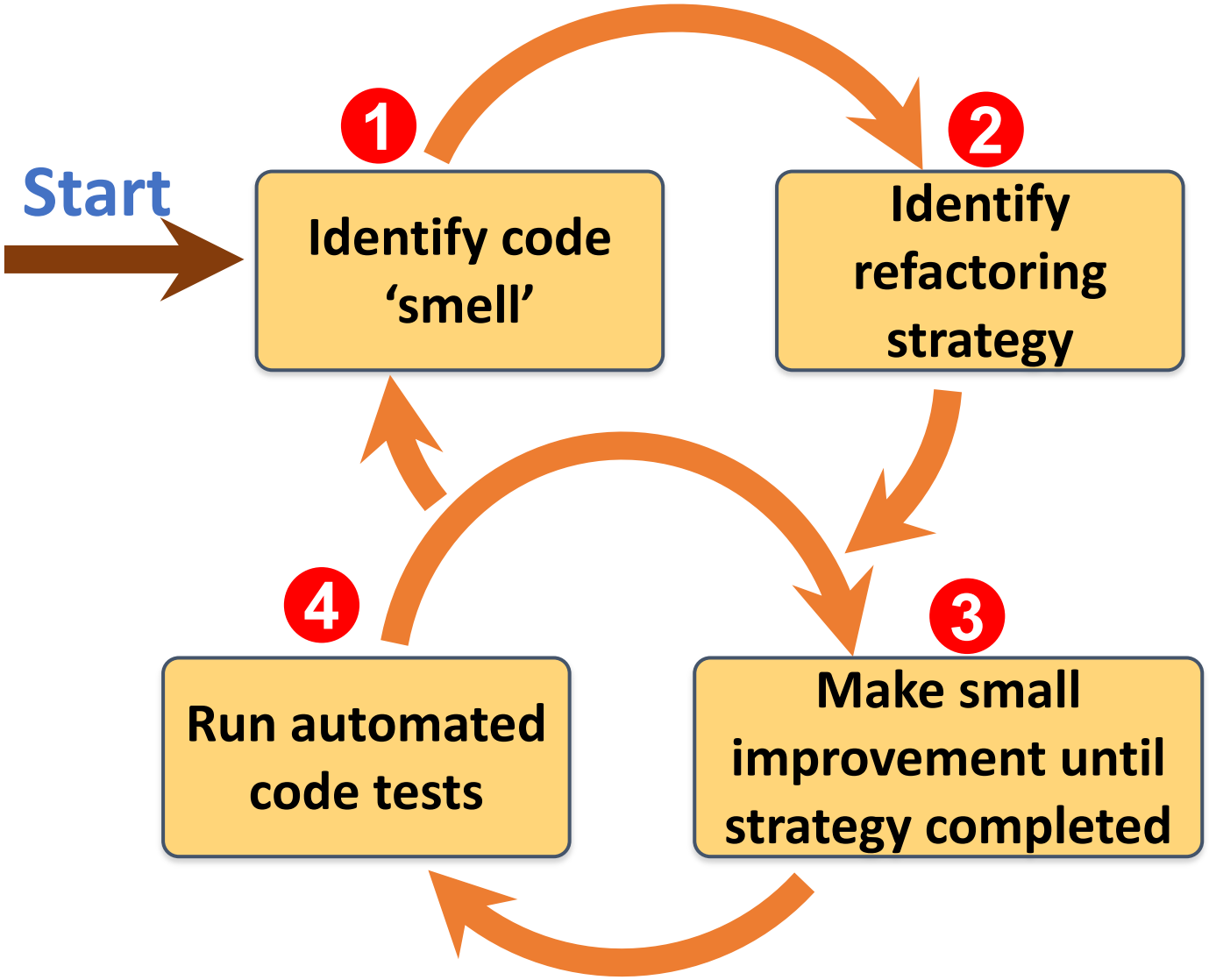


Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Software product quality attributes

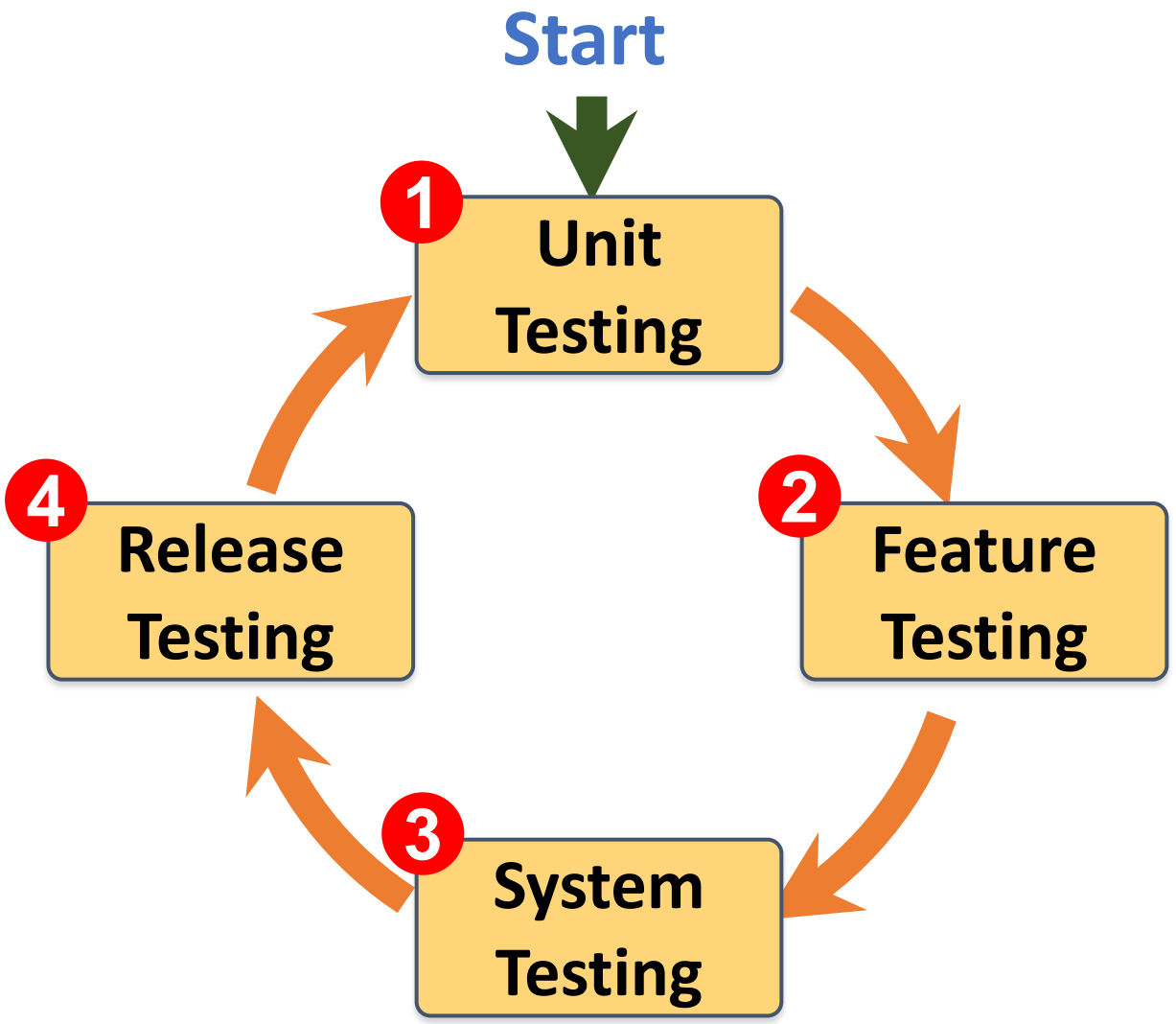
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

A refactoring process



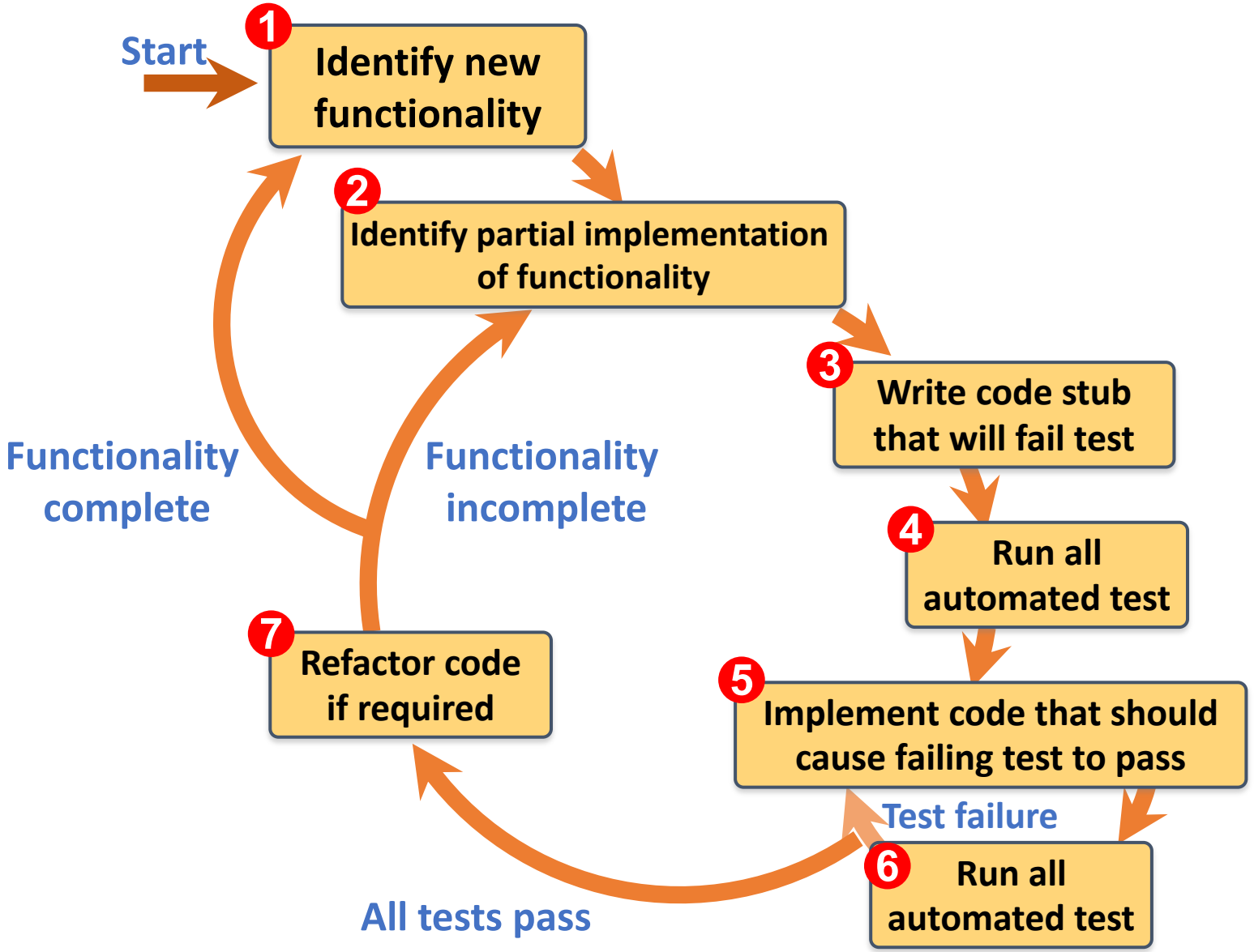
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Functional testing



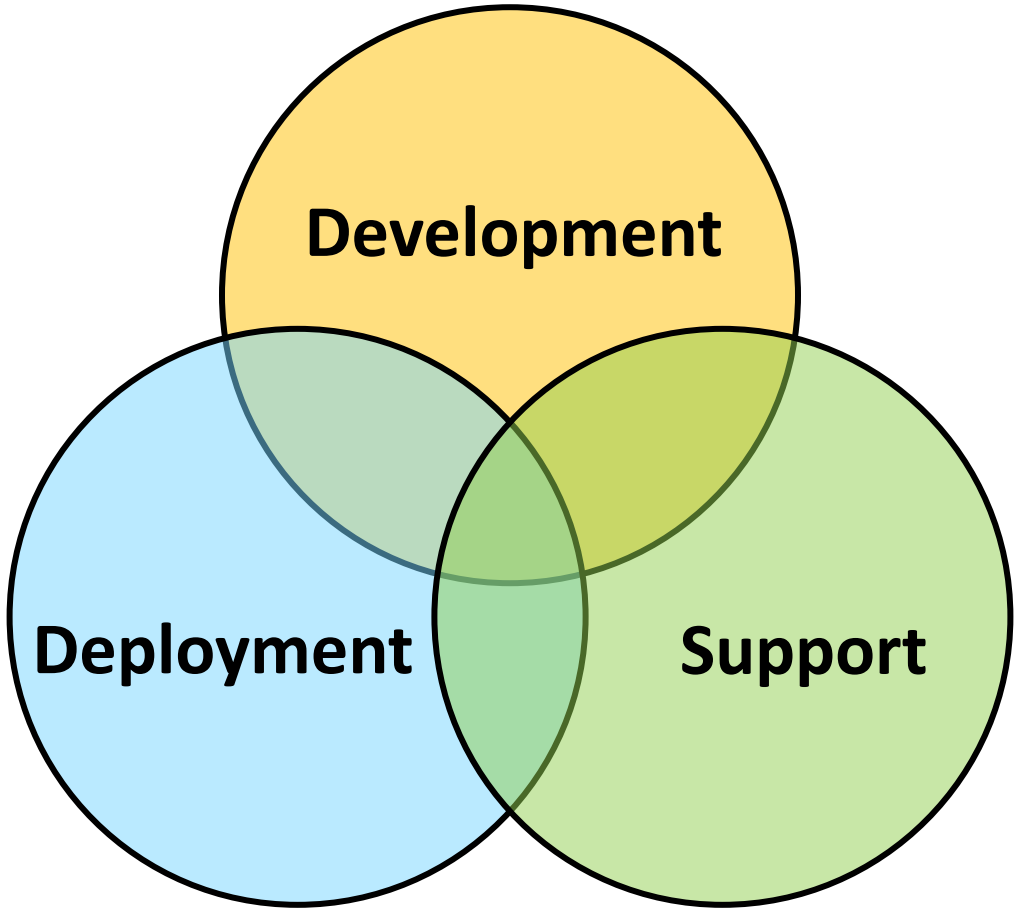
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Test-driven development (TDD)



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

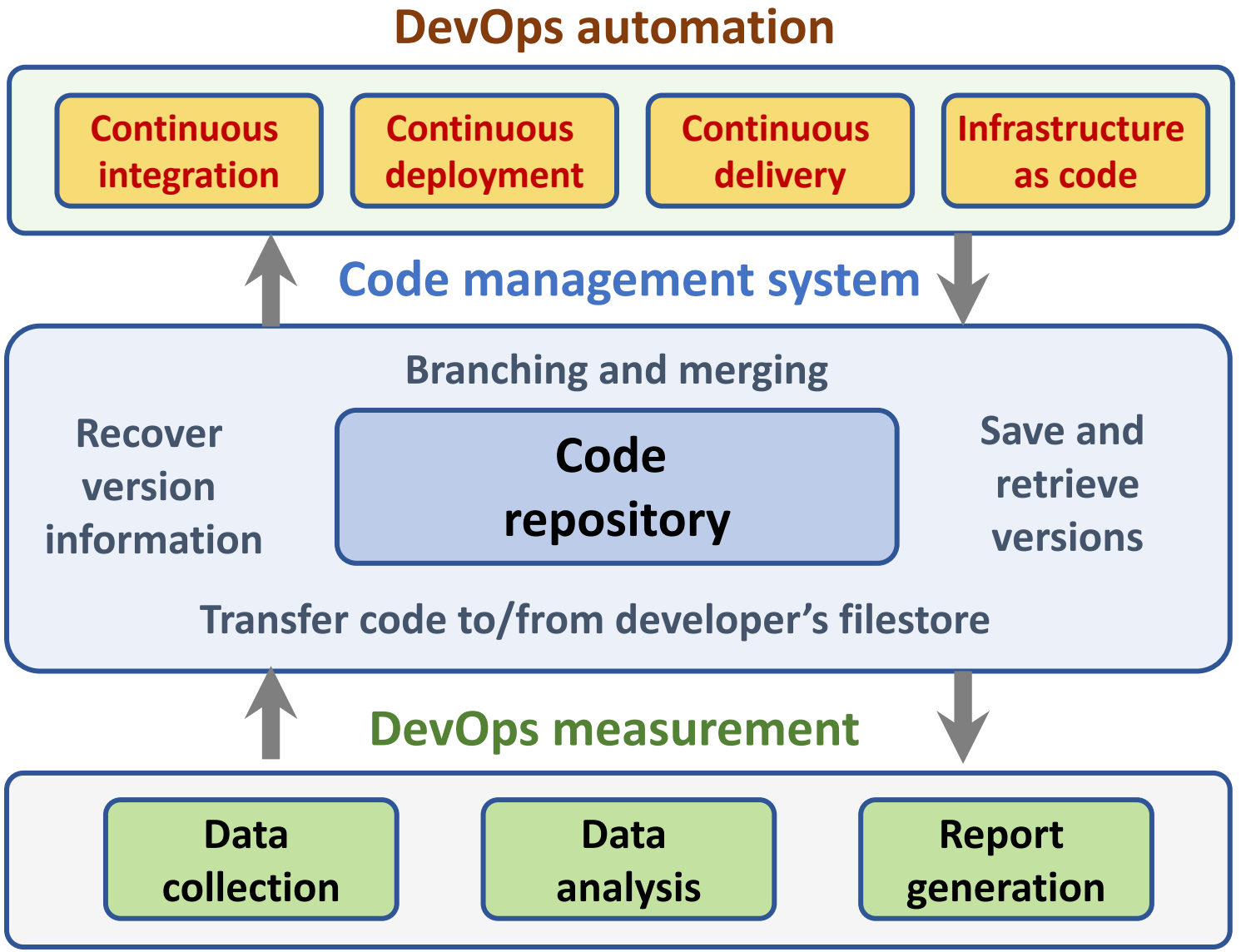
DevOps



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.

Agentic AI for Software Engineering Architecture

Architectural Design

Design Pattern Suggestions

Technology Stack Recommendation

Diagram Generation

Trade-Off Analysis

System Decomposition

Modulization Based on Function

Automated Component Design

Bottleneck Optimization

Distribution Architecture

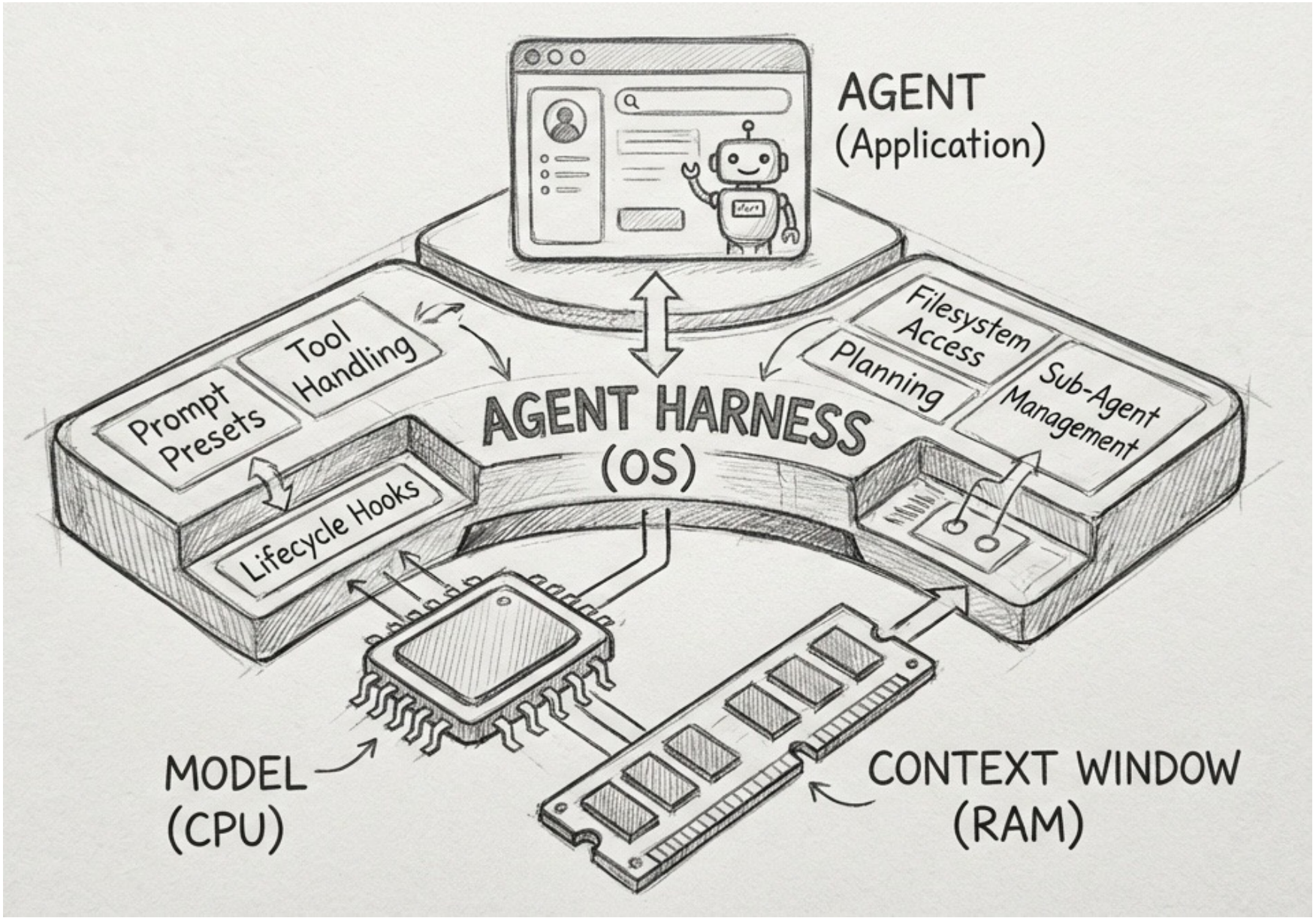
Designing Scalable Systems

Supporting Dynamic Scaling

Optimizing Performance

Key Benefits: Enhanced Efficiency, Data-Driven Decisions, Scalability

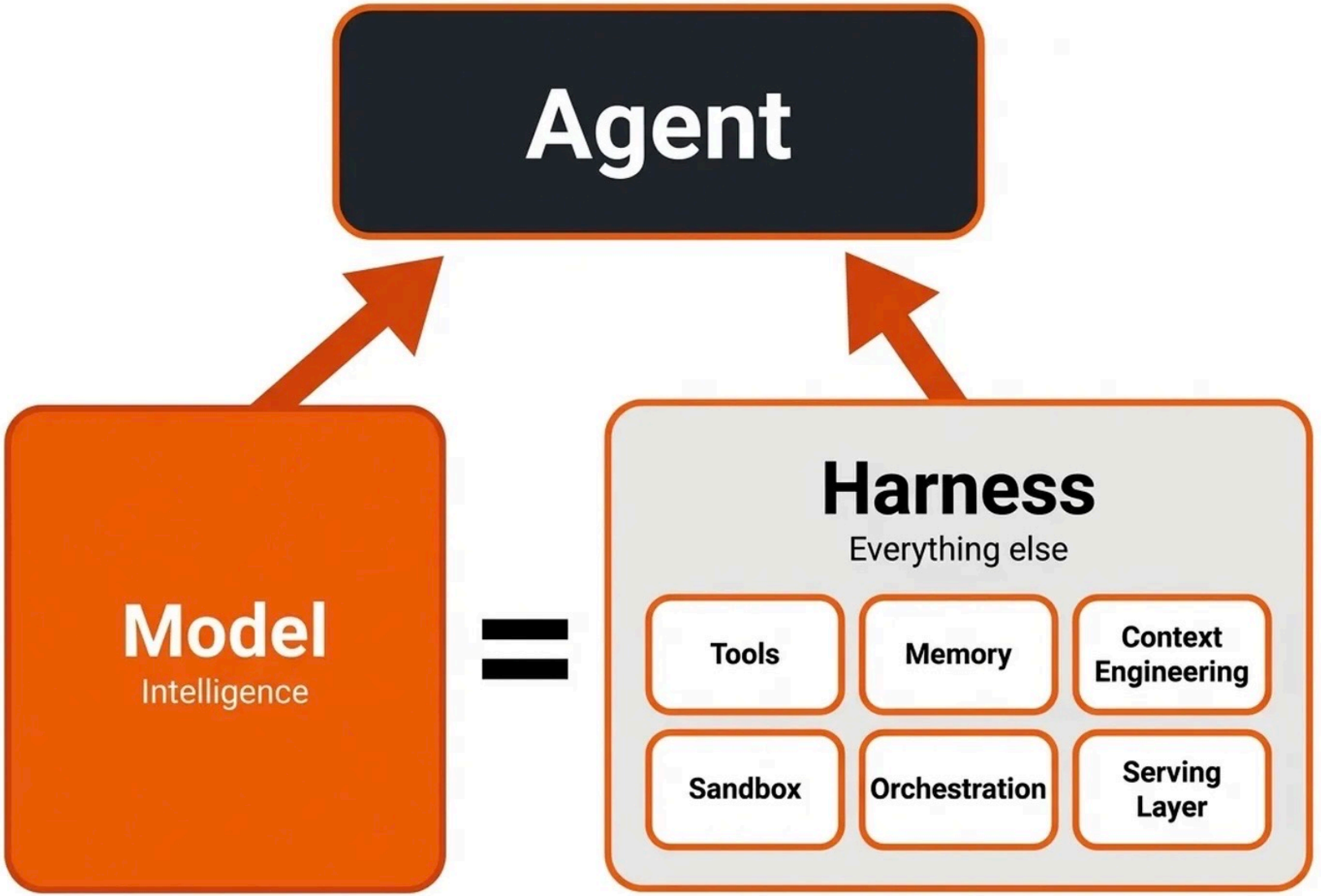
Harness Engineering



Source: https://www.philschmid.de/agent-harness-2026

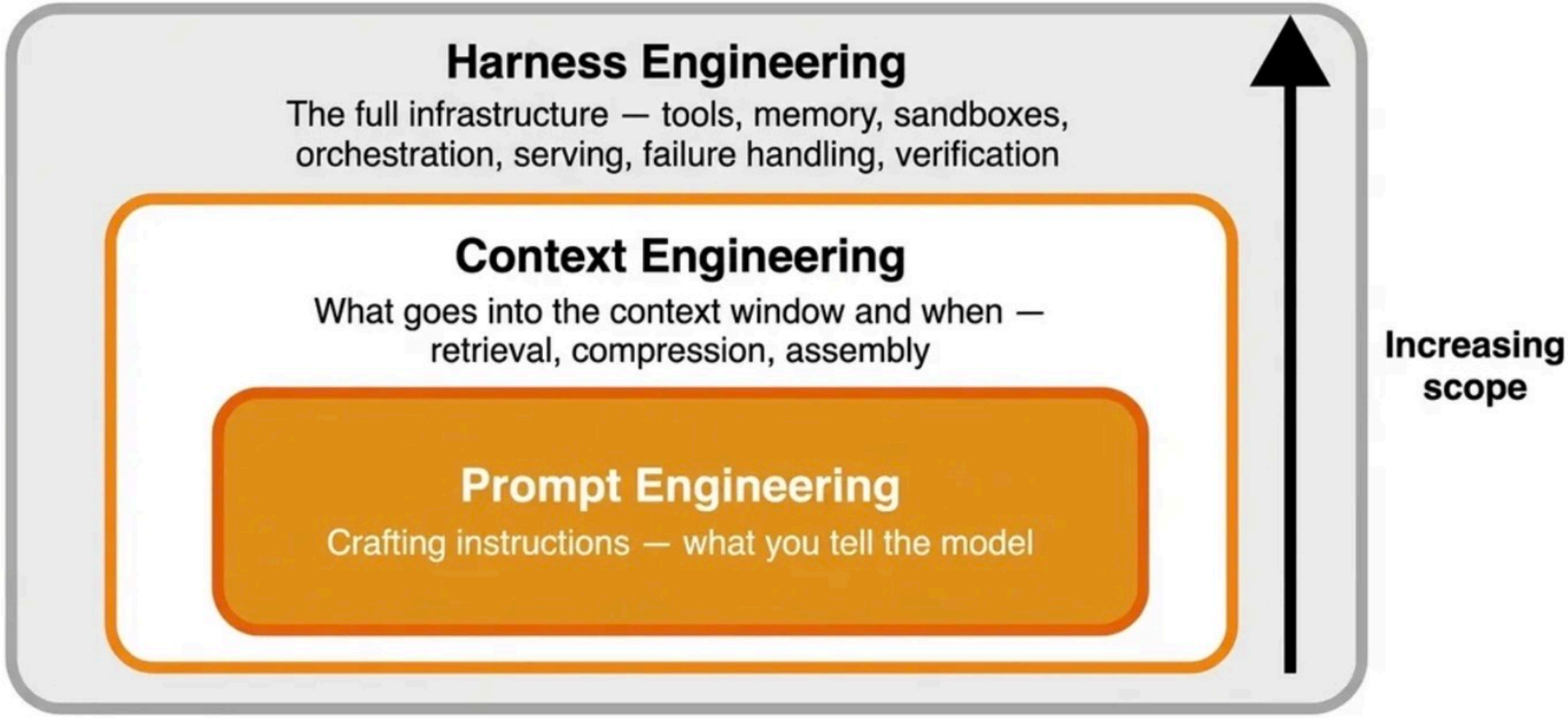
Agentic AI Harness Engineering

Building systems that transform the LLM into the new operating system



Source: https://www.decodingai.com/p/agentic-harness-engineering

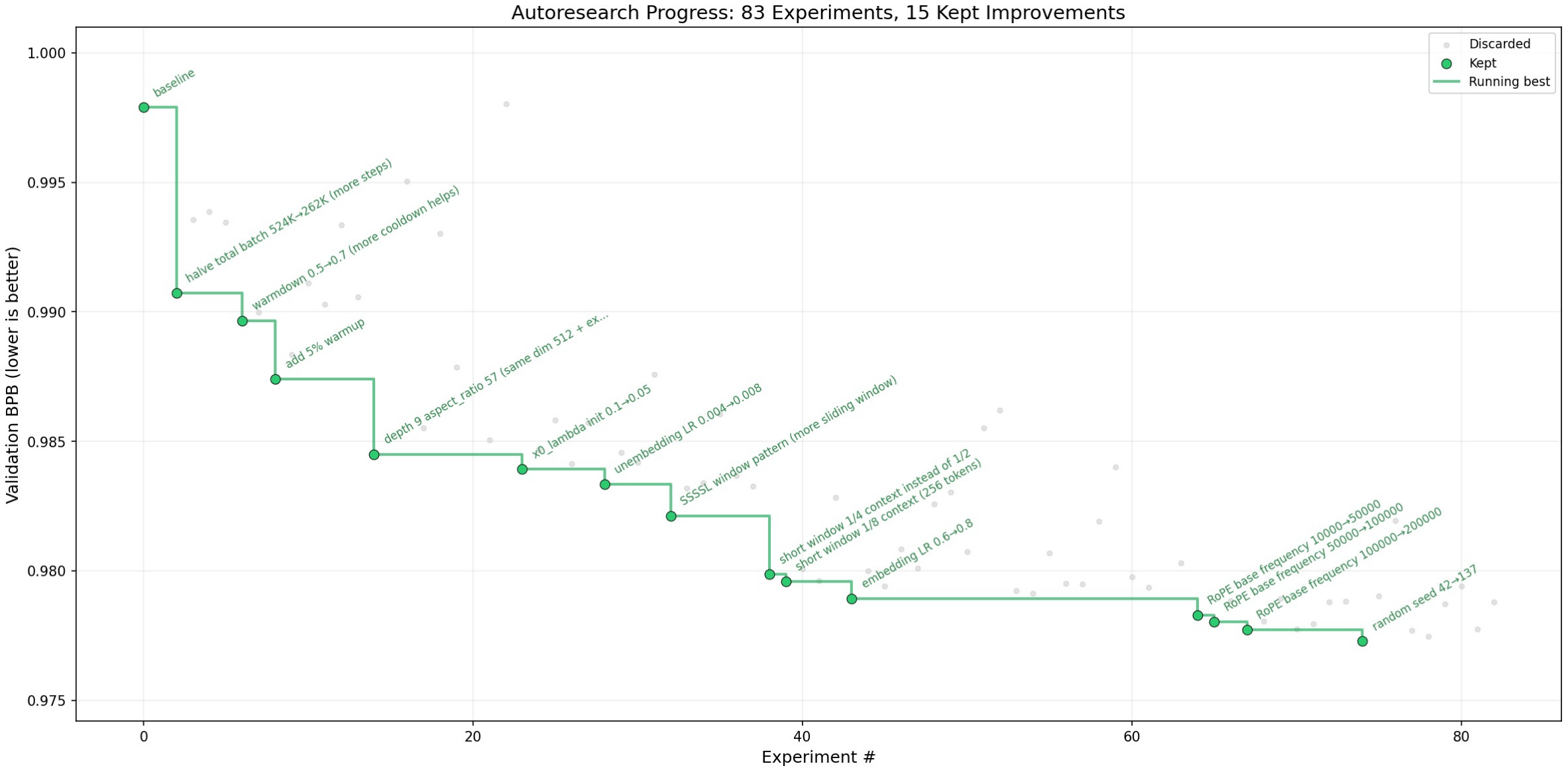
Harness Engineering
Context Engineering, Prompt Engineering



Each level encompasses the previous one.

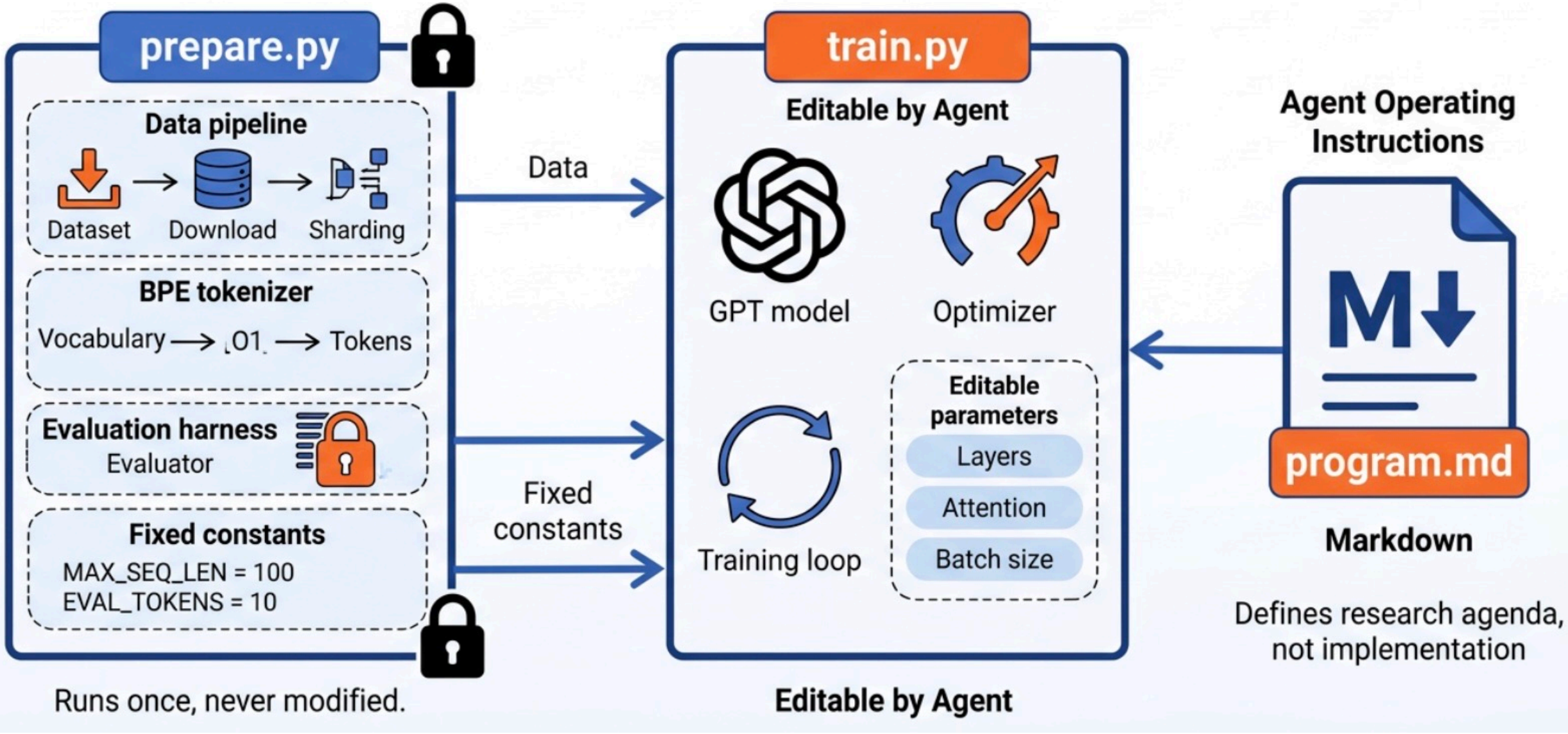
Source: https://www.decodingai.com/p/agentic-harness-engineering

Karpathy Autoresearch



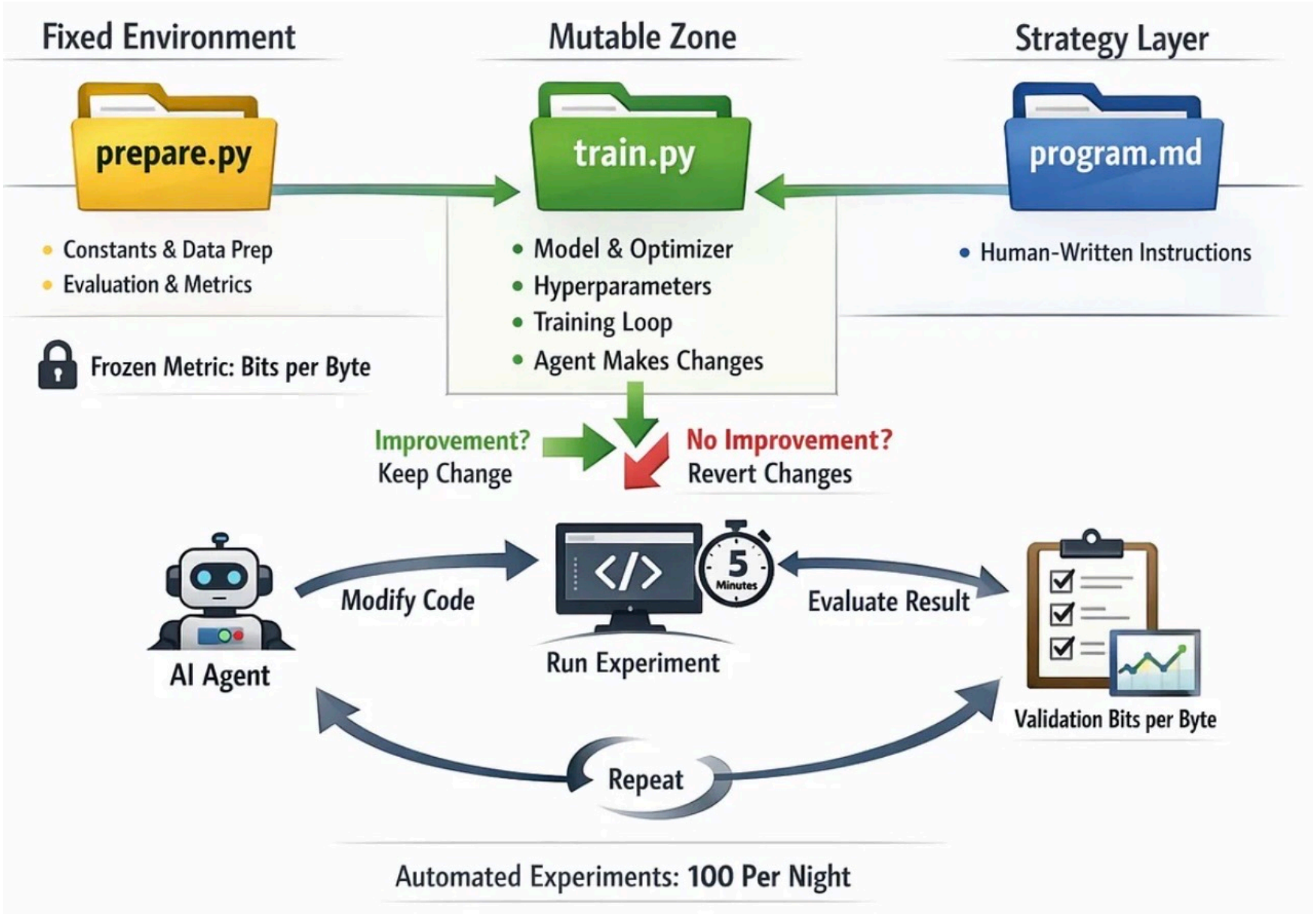
Source: https://github.com/karpathy/autoresearch

Karpathy Autoresearch Architecture



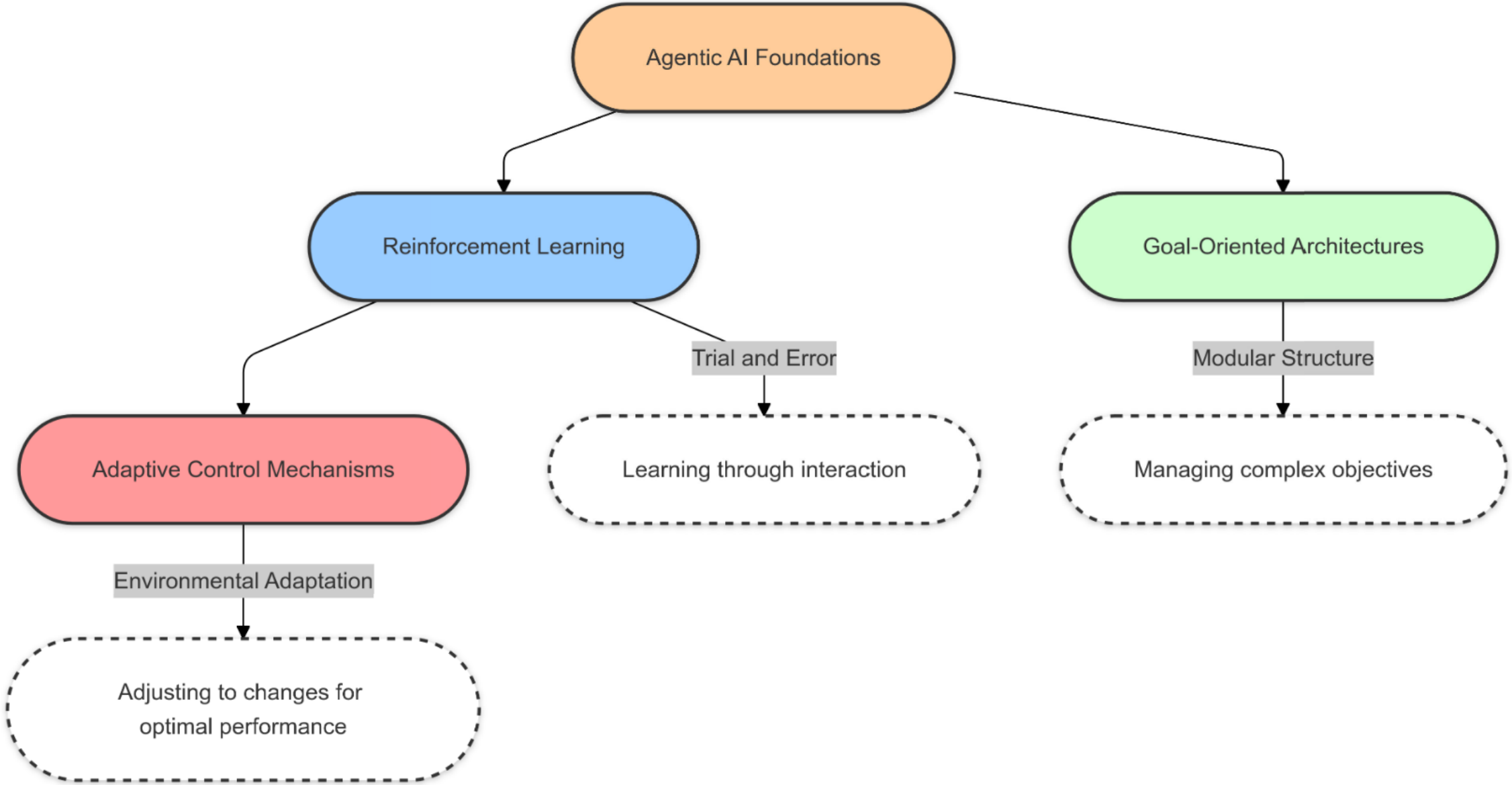
Source: https://datasciencedojo.com/blog/karpathy-autoresearch-explained/

Karpathy Autoresearch: The Loop That Improves Your Work



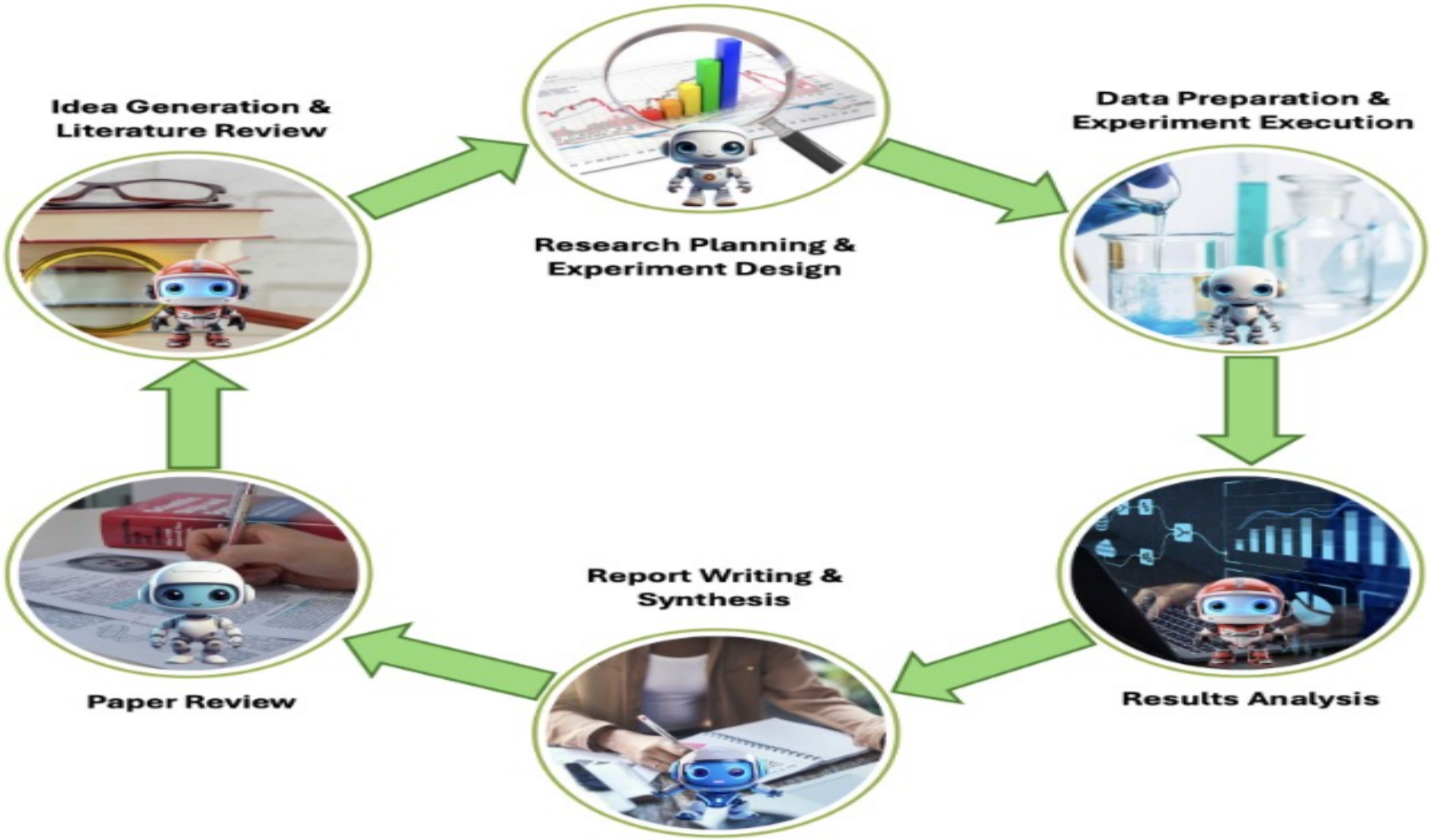
Source: https://thecreatorsai.com/p/autoresearch-the-loop-that-improves

Agentic AI Foundations



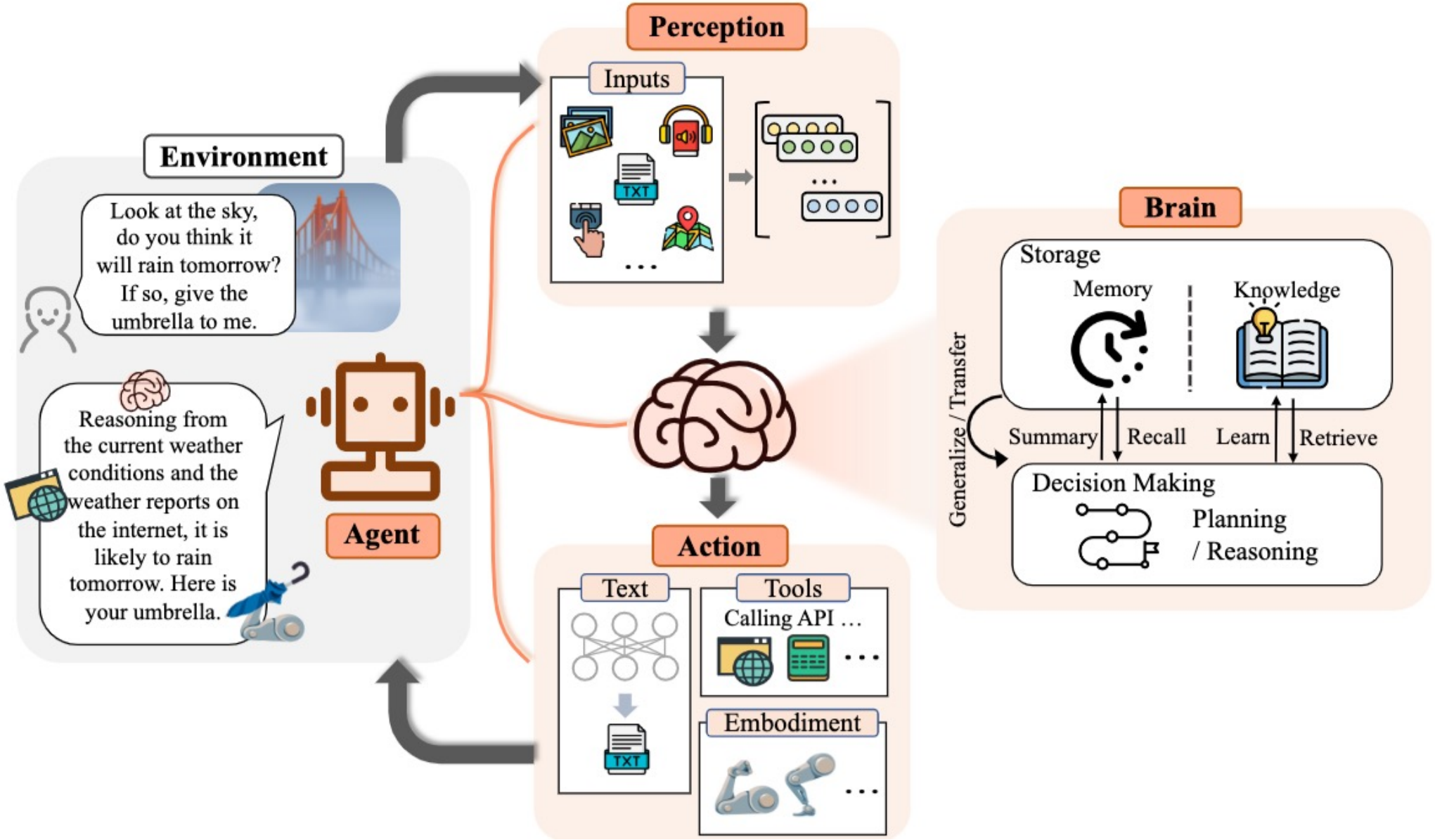
Source: Acharya, Deepak Bhaskar, Karthigeyan Kuppan, and B. Divya. "Agentic AI: Autonomous Intelligence for Complex Goals–A Comprehensive Survey." IEEE Access (2025).

Agentic AI Workflow for Scientific Discovery



Source: Gridach, Mourad, Jay Nanavati, Khaldoun Zine El Abidine, Lenon Mendes, and Christina Mack. "Agentic AI for Scientific Discovery: A Survey of Progress, Challenges, and Future Directions." arXiv preprint arXiv:2503.08979 (2025).

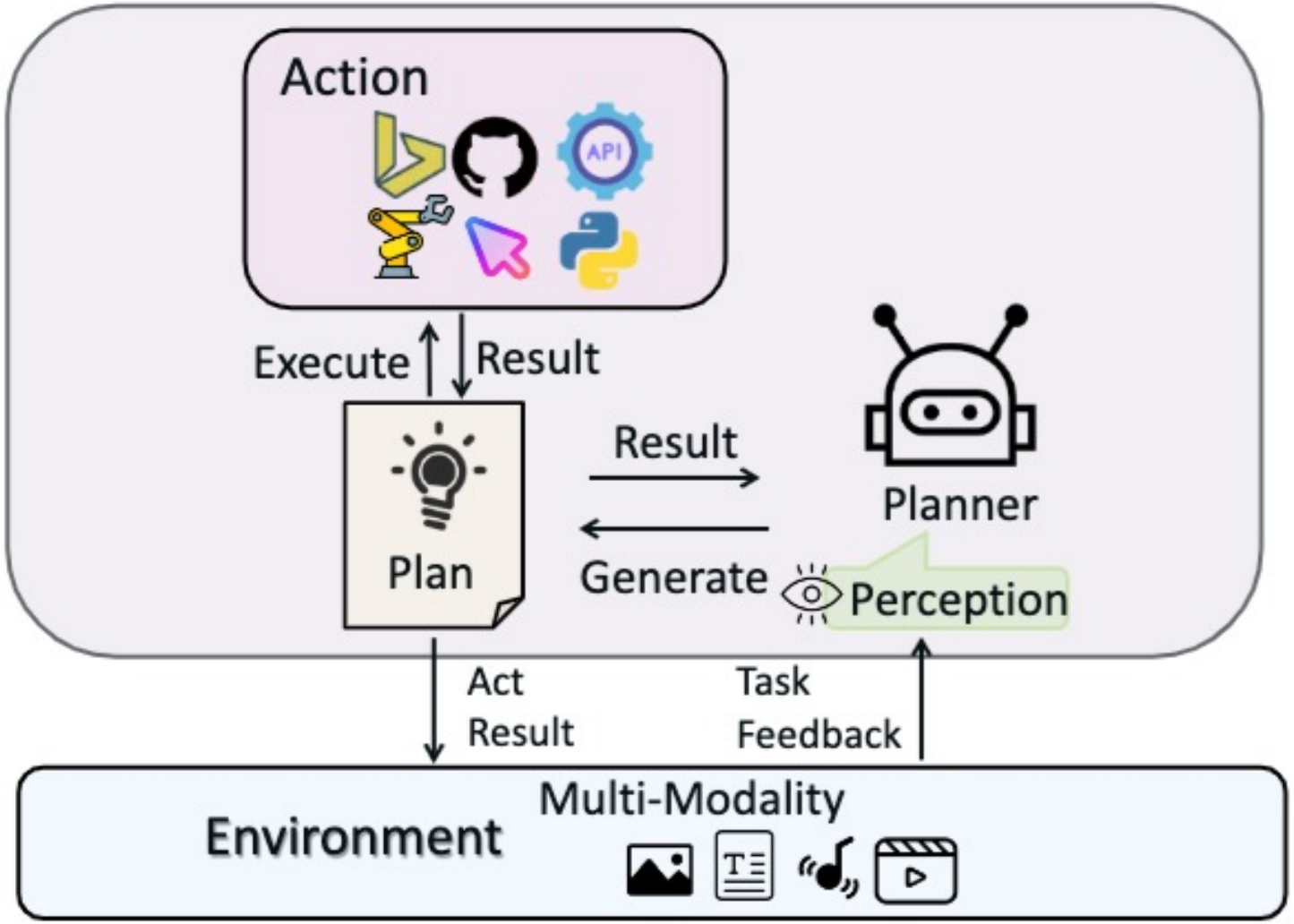
Large Language Model (LLM) Based Agents



Source: Guo, Taicheng, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. "Large language model based multi-agents: A survey of progress and challenges." arXiv preprint arXiv:2402.01680 (2024).

Large Multimodal Agents (LMAs)

(a) Type I: Closed-source LLMs as Planners w/o Longterm Memory.

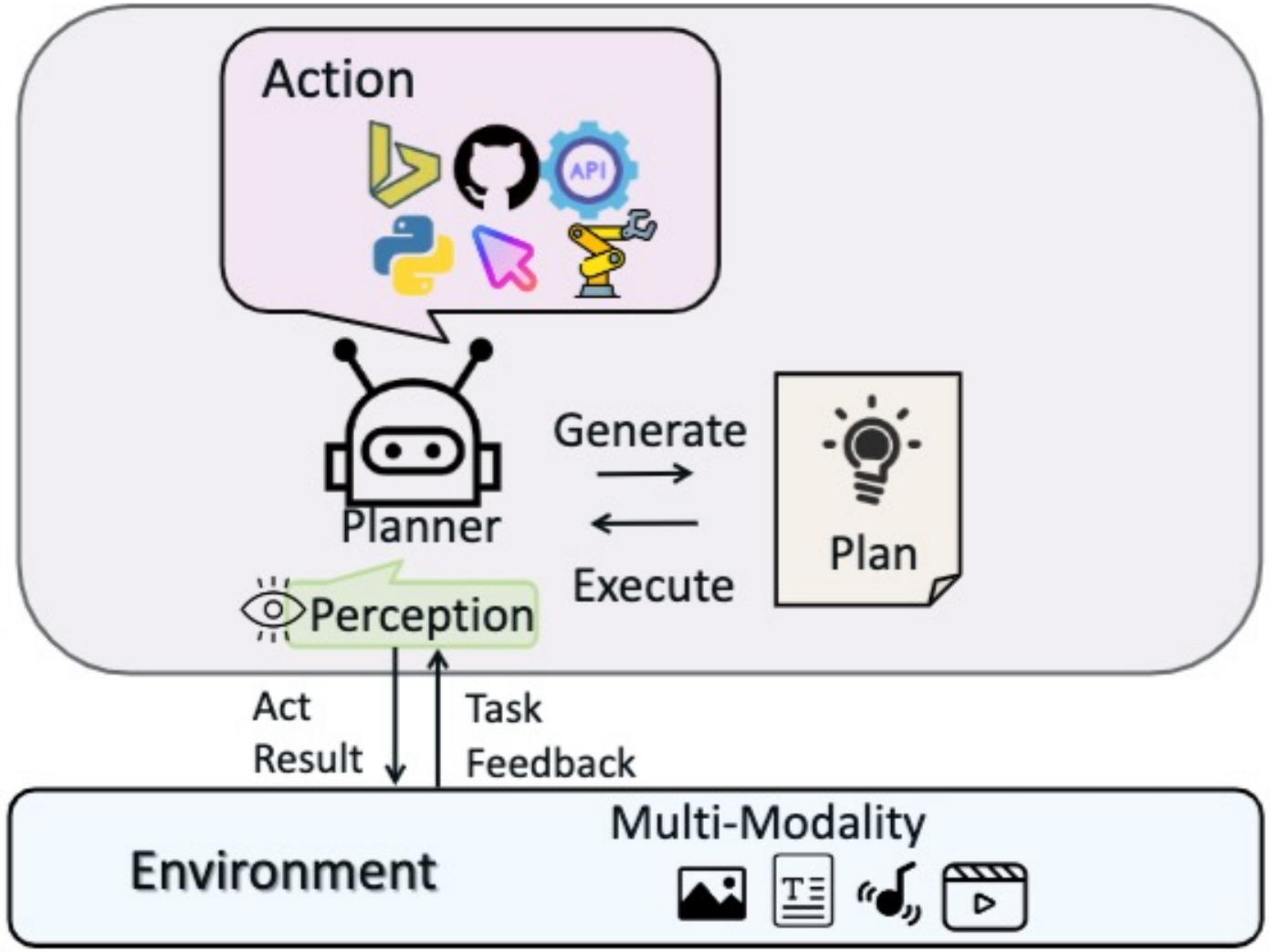


Use prompt techniques to guide closed-source LLMs in decision-making
and planning to complete tasks without long memory.

Source: Xie, Junlin, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. "Large Multimodal Agents: A Survey." arXiv preprint arXiv:2402.15116 (2024).

Large Multimodal Agents (LMAs)

(b) Type II: **Finetuned LLMs** as Planners w/o long-term Memory.

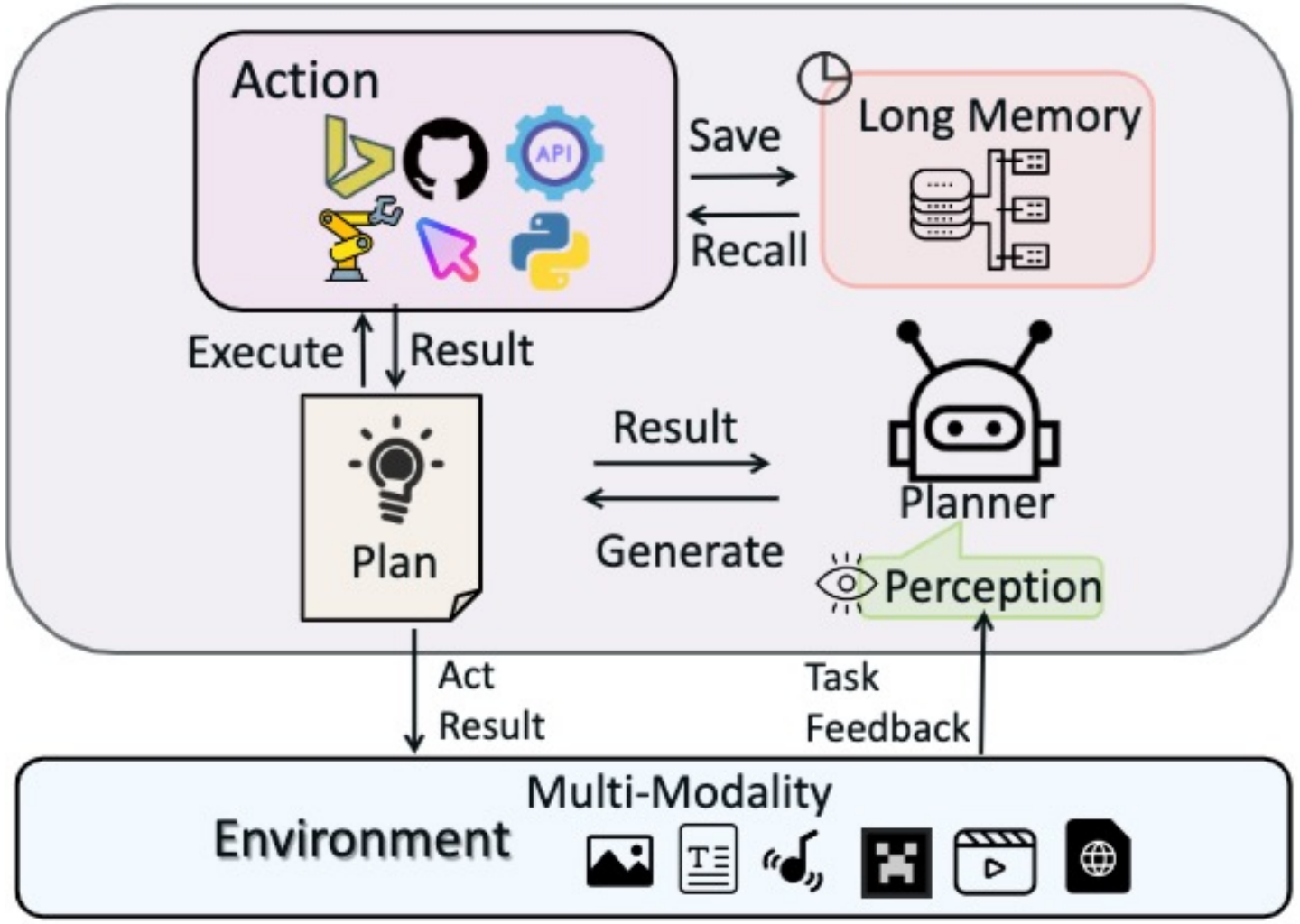


Use action-related data to finetune existing open-source large models,
enabling them to achieve decision-making, planning, and tool invocation capabilities comparable to closed-source LLMs

Source: Xie, Junlin, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. "Large Multimodal Agents: A Survey." arXiv preprint arXiv:2402.15116 (2024).

Large Multimodal Agents (LMAs)

(c) Type III: Planners with **Indirect Long-term Memory**

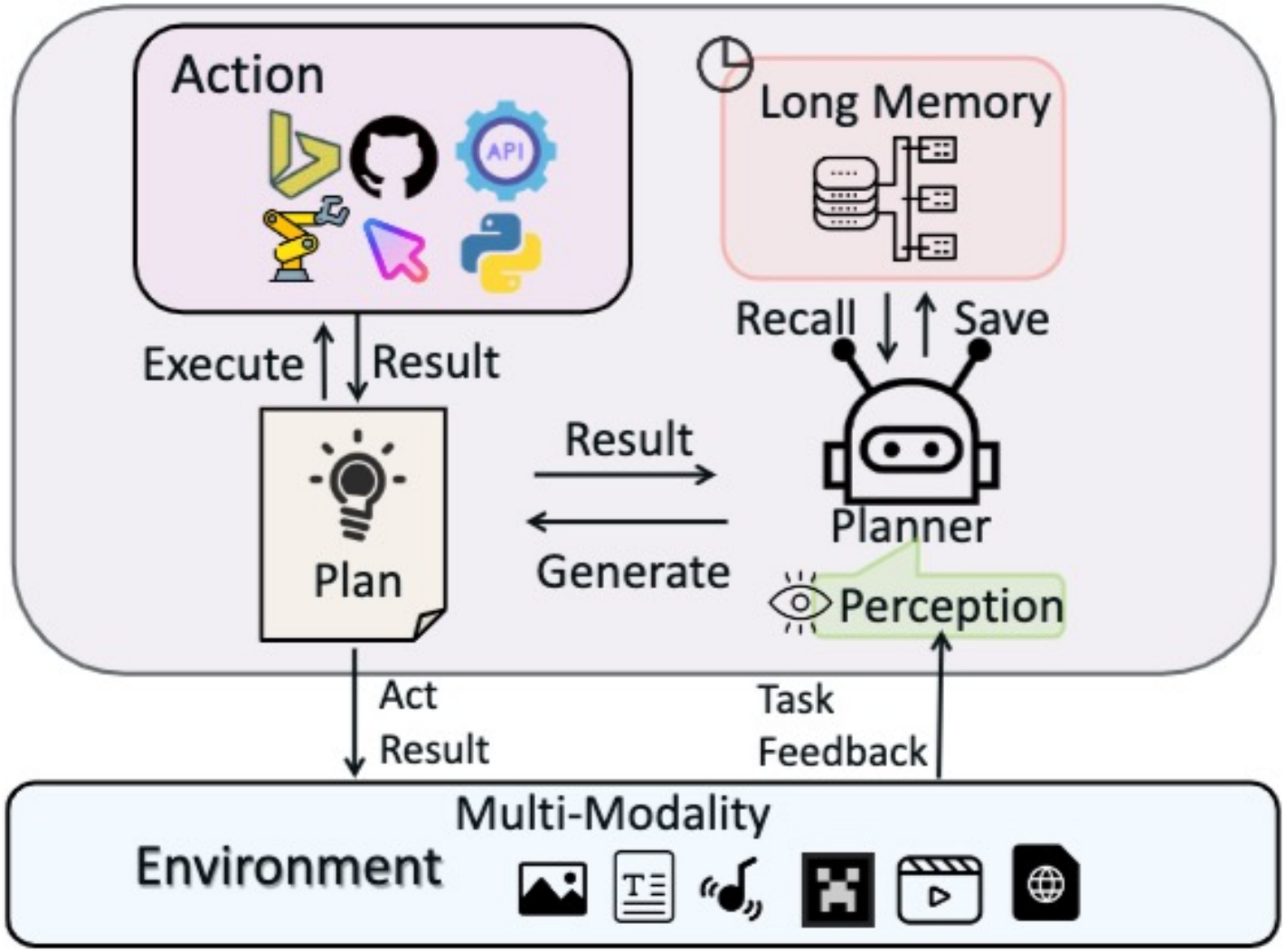


Introduce indirect long-term memory functions,
further enhancing their generalization and adaptation abilities in environments closer to the real world.

Source: Xie, Junlin, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. "Large Multimodal Agents: A Survey." arXiv preprint arXiv:2402.15116 (2024).

Large Multimodal Agents (LMAs)

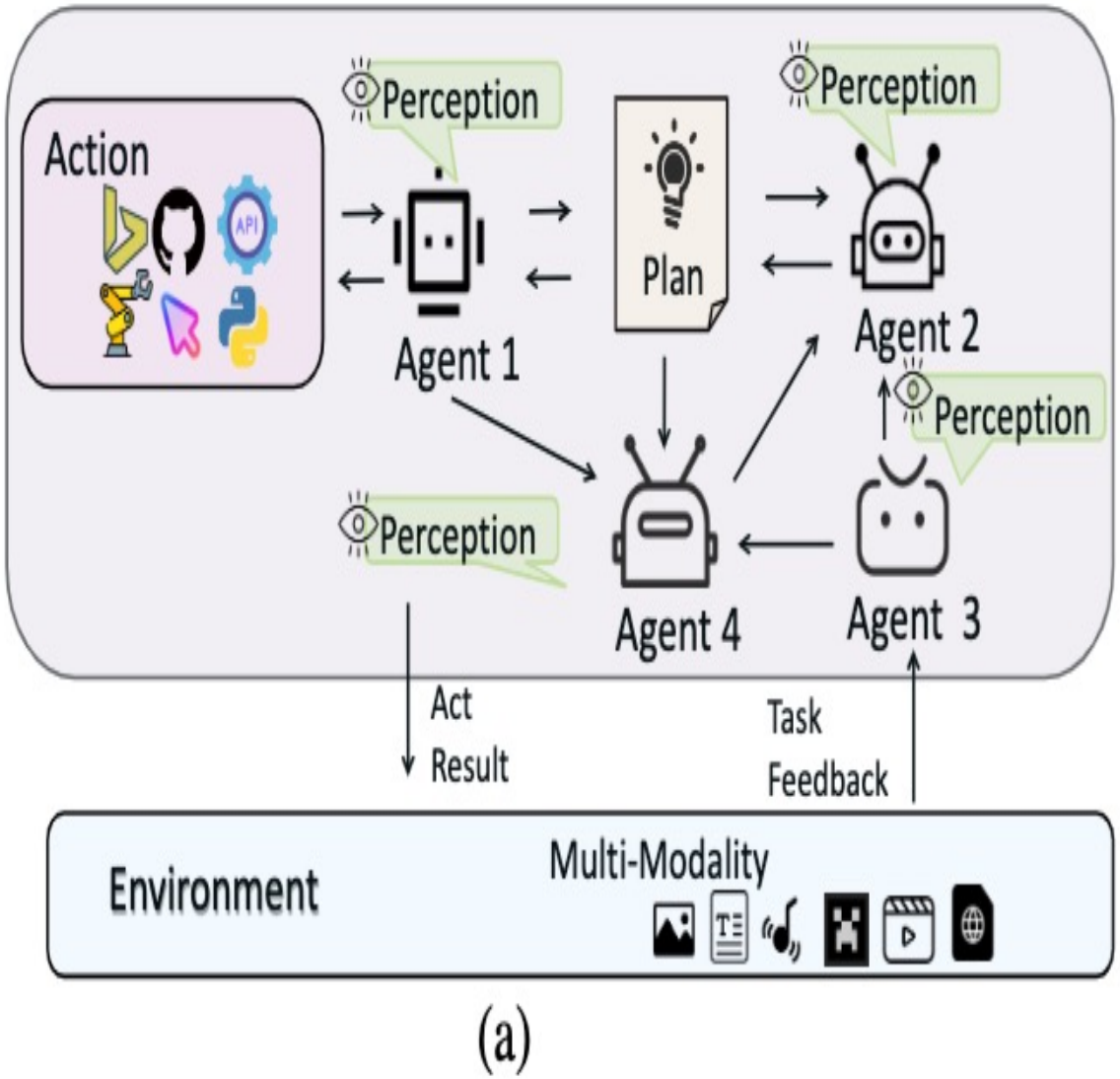
(d) Type IV: Planners with Native Long-term Memory

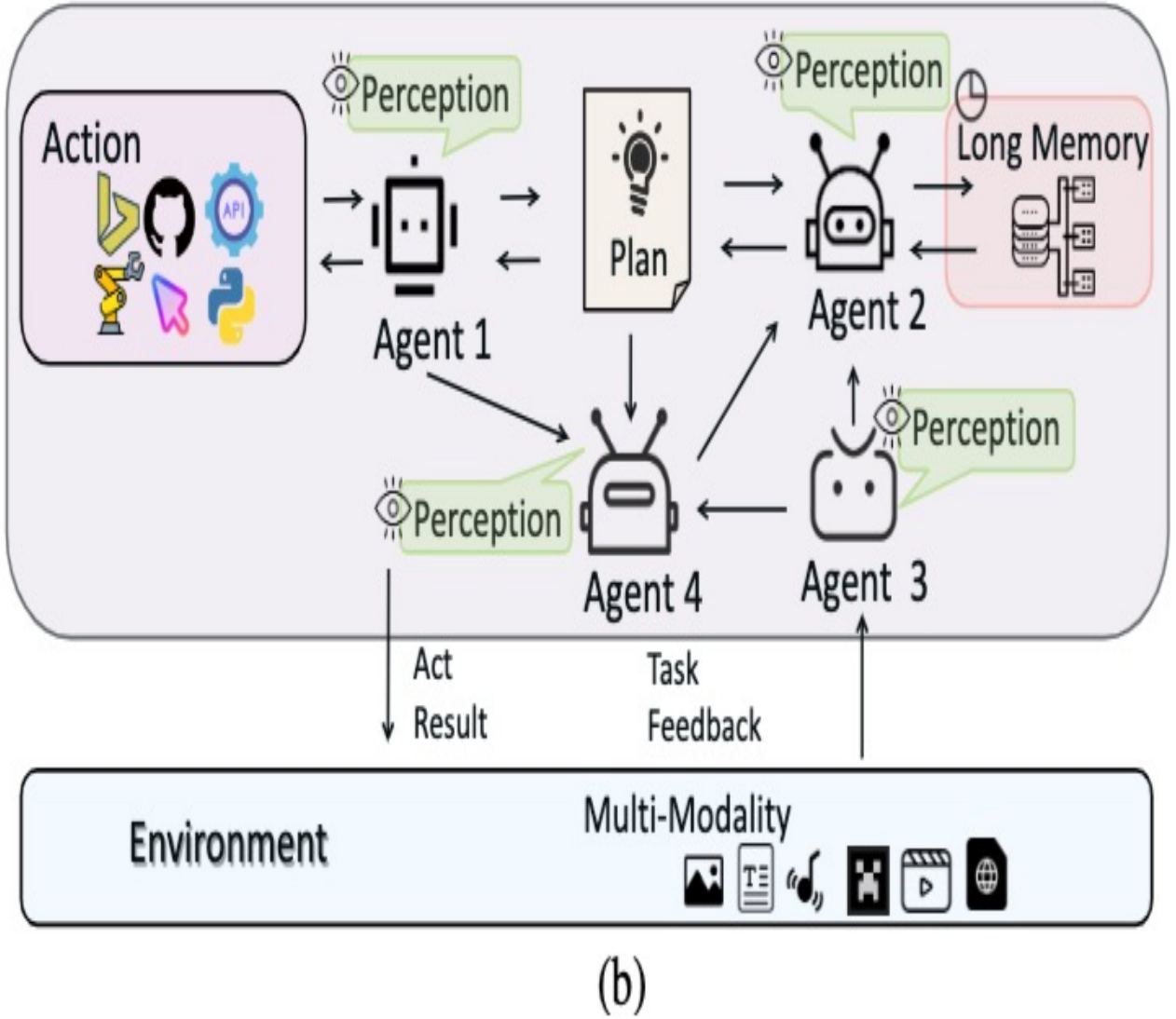


Introduce native long-term memory functions,
further enhancing their generalization and adaptation abilities in environments closer to the real world.

Source: Xie, Junlin, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. "Large Multimodal Agents: A Survey." arXiv preprint arXiv:2402.15116 (2024).

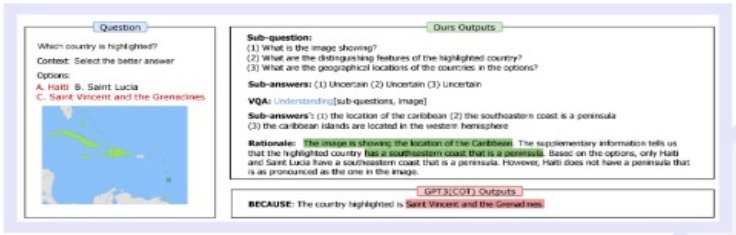
Multi-Agent Frameworks

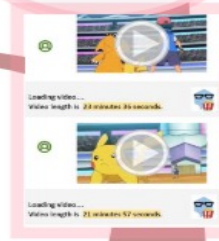




Source: Xie, Junlin, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. "Large Multimodal Agents: A Survey." arXiv preprint arXiv:2402.15116 (2024).

Applications of Large Multimodal Agents (LMAs)





Audio Editing&Generation
MusicAgent
WavJourney
Copilot
AudioGPT

Complex Visual Reasoning Tasks
VisProgram
Visual Chatgpt
HuggingGPT
MM-ReAct
Chameleon
MLLM-Tool
CRAFT
Avis
ViperGPT
GPT4Tools
Llava-plus
AssistGPT
M3
DDCoT
CLOVA

Embodied AI &Robotics
STEVE EMMA
Octopus GRID
JARVIS-1 MP5 DEPS
DiscussNav

Game-developer
SMARTPLAY
VisualWebArena

Video Understanding
AssistGPT
DORAEMONGPT
ChatVideo

Visual Generation & Editing
LLaVA-Interactive
MM-ReAct
Mulan

Autonomous Driving
GPT-Driver
DLAH

UI-assistants
Mobile-Agent
MM-Navigator
ASSISTGUI
WebWISE
DroidBot-GPT
AutoDroid
Auto-UI
AppAgent
OS-Copilo
MemoDroid
Openagents
GPT-4V-Act

Source: Xie, Junlin, Zhihong Chen, Ruifei Zhang, Xiang Wan, and Guanbin Li. "Large Multimodal Agents: A Survey." arXiv preprint arXiv:2402.15116 (2024).

CodeActAgent:
General Agent Multi-turn Interaction Framework

Example

User:

① Find the sum of the reciprocals of the roots of $x^2-13x+4=0$.

Assistant:

② To find the sum of the reciprocals of the roots of the quadratic equation $x^2-13x+4=0$ [...]

③
```
<execute>
import sympy
x = sympy.Symbol('x')
roots = sympy.solve(x**2 - 13*x + 4)
print(1/roots[0] + 1/roots[1])
</execute>
```

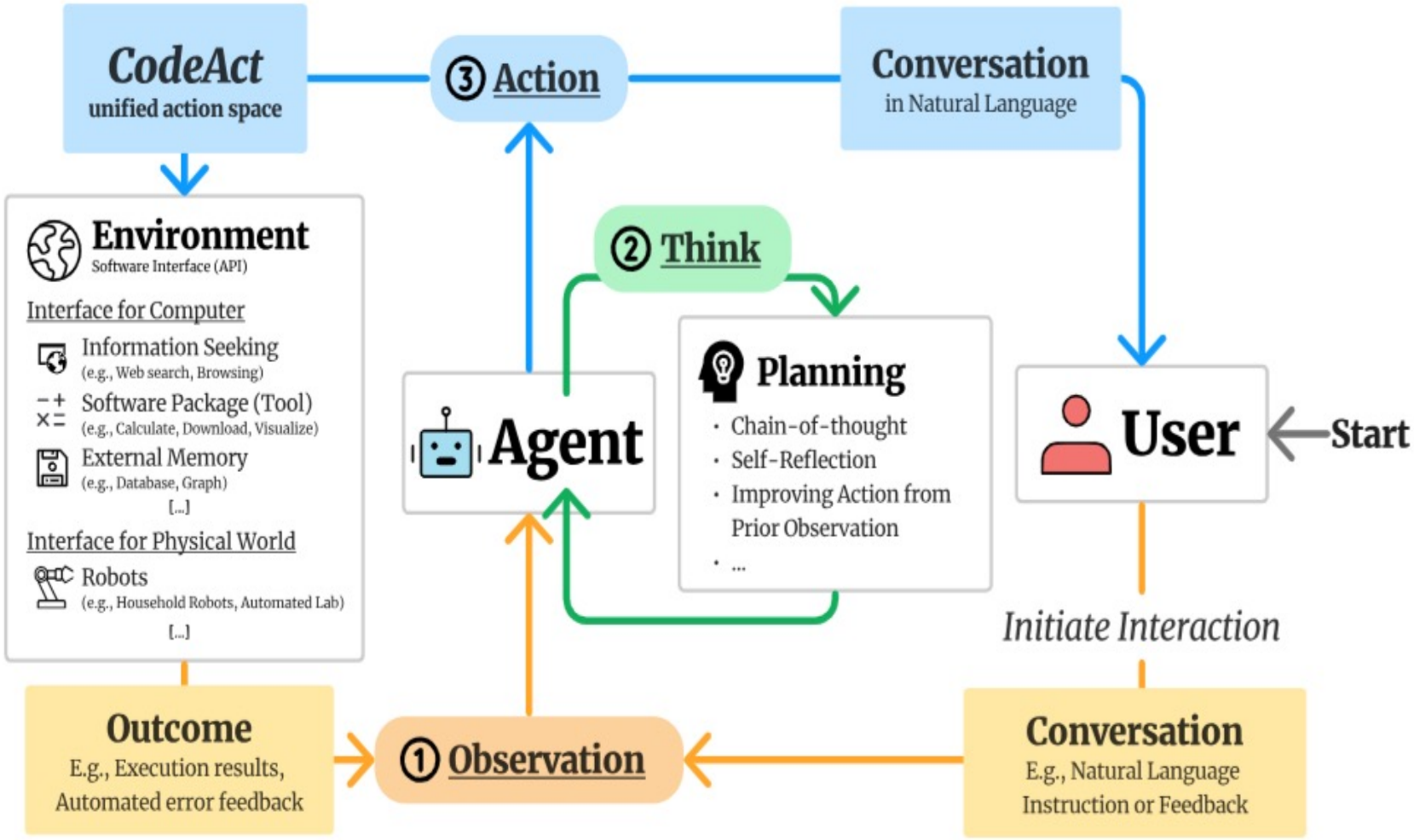
Env:

① 1/(3*sqrt(17)/2 + 13/2) + 1/(13/2 - 3*sqrt(17)/2)

Assistant:

② The roots obtained are in symbolic form, which makes it difficult to evaluate the sum directly. Let's use the quadratic formula to explicitly find the numerical values of the roots first.

[...]



Source: Wang, Xingyao, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. "Executable code actions elicit better llm agents." arXiv preprint arXiv:2402.01030 (2024).

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.

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

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.

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

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.**

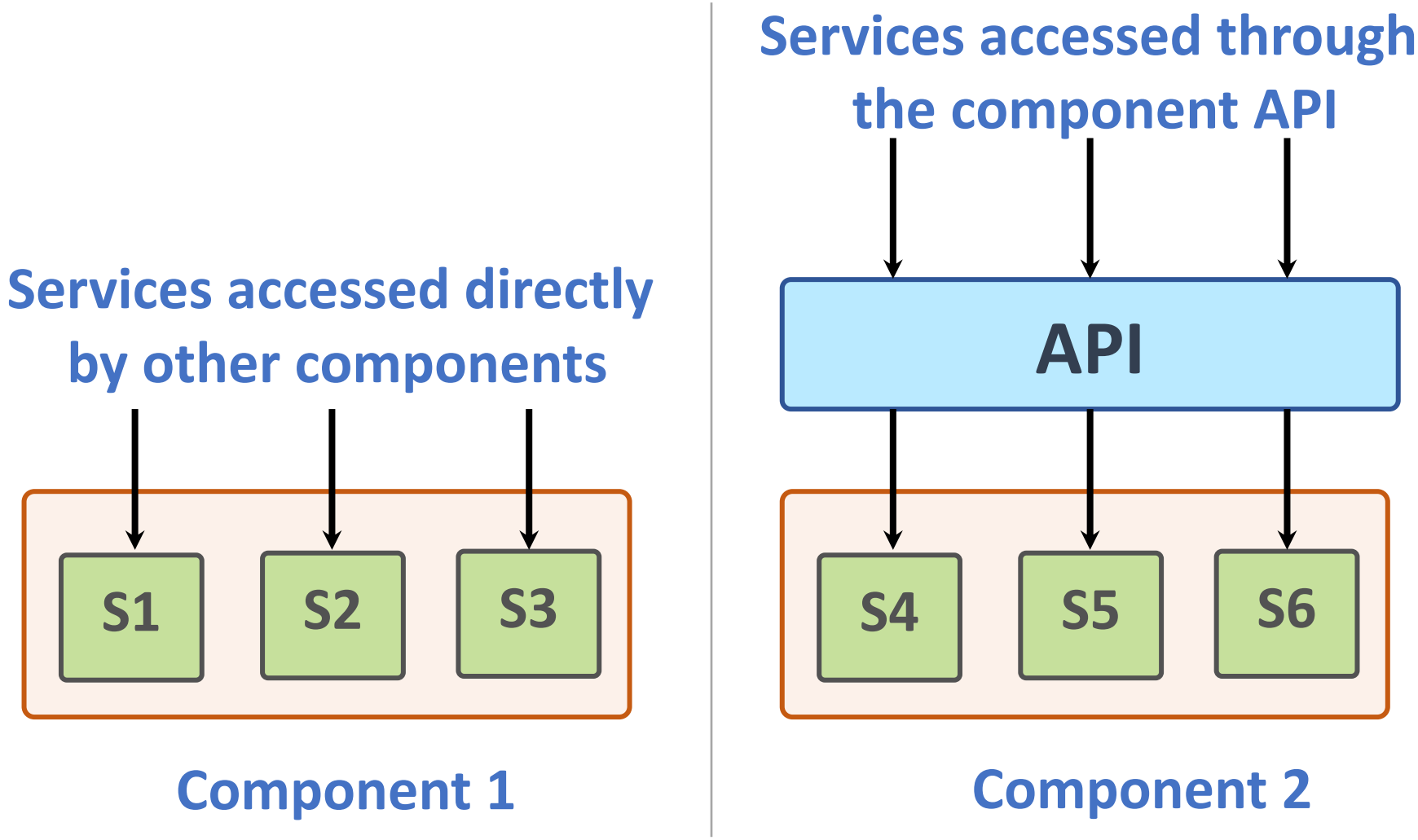
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.**

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

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?

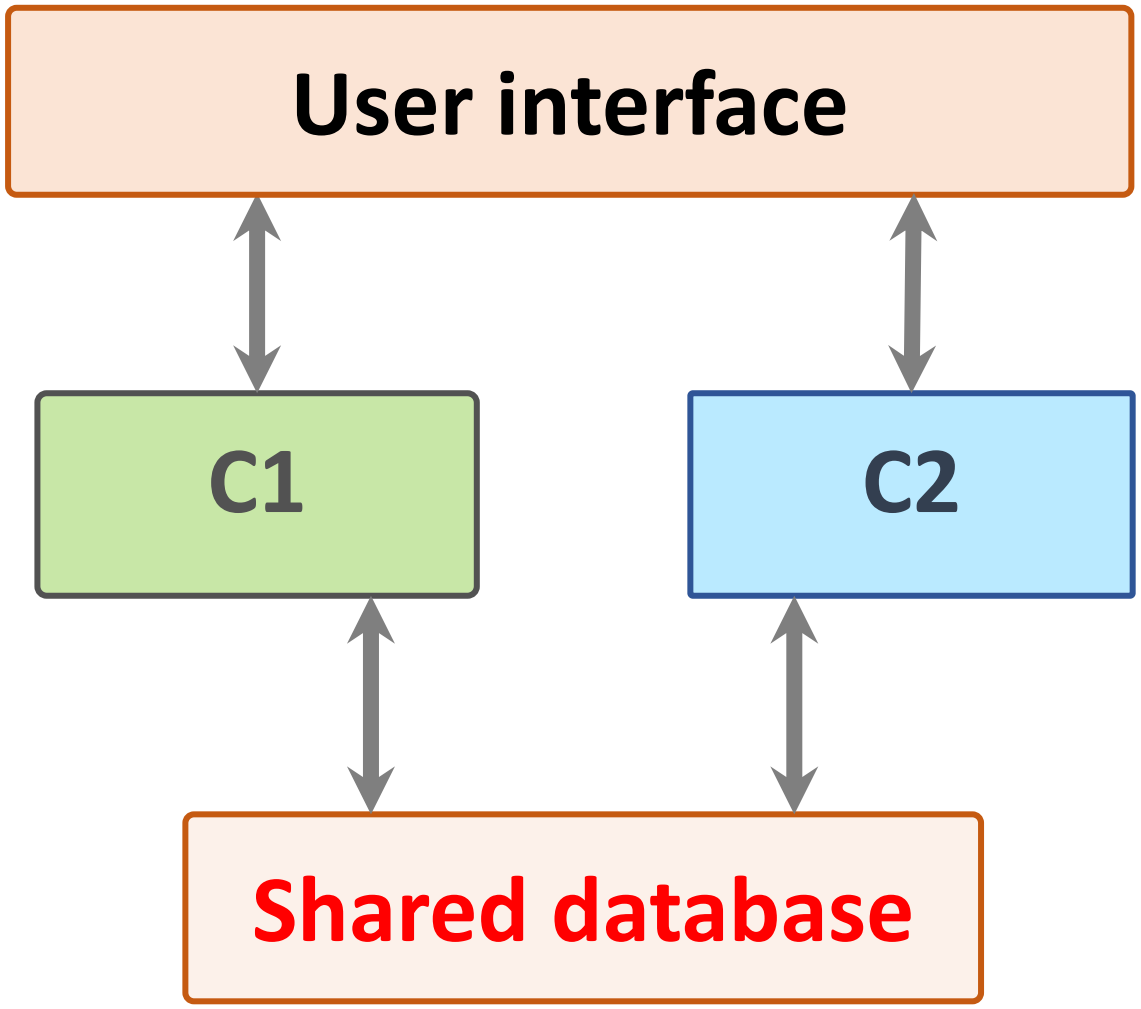
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

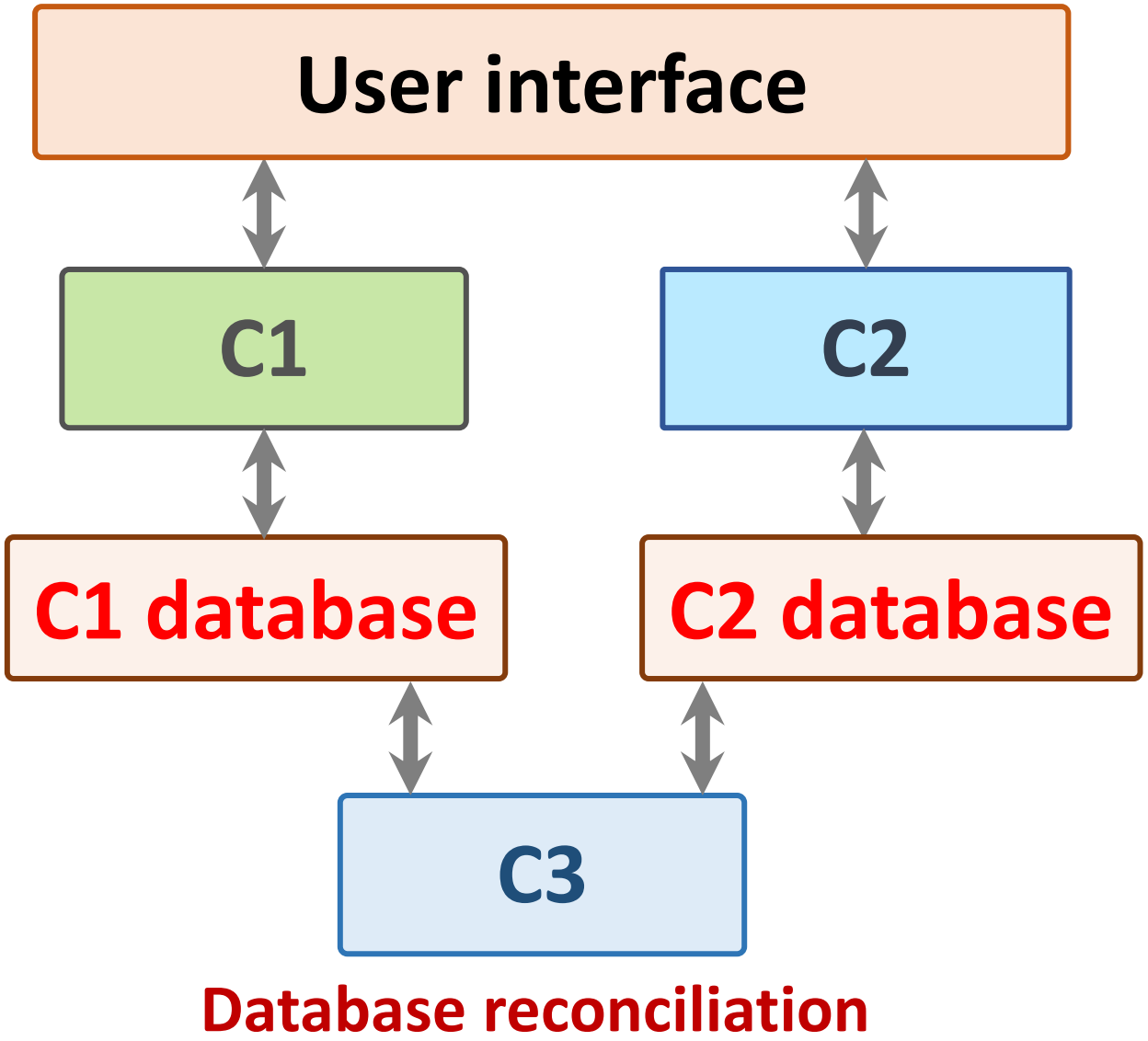
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Shared database architecture



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

Multiple database architecture



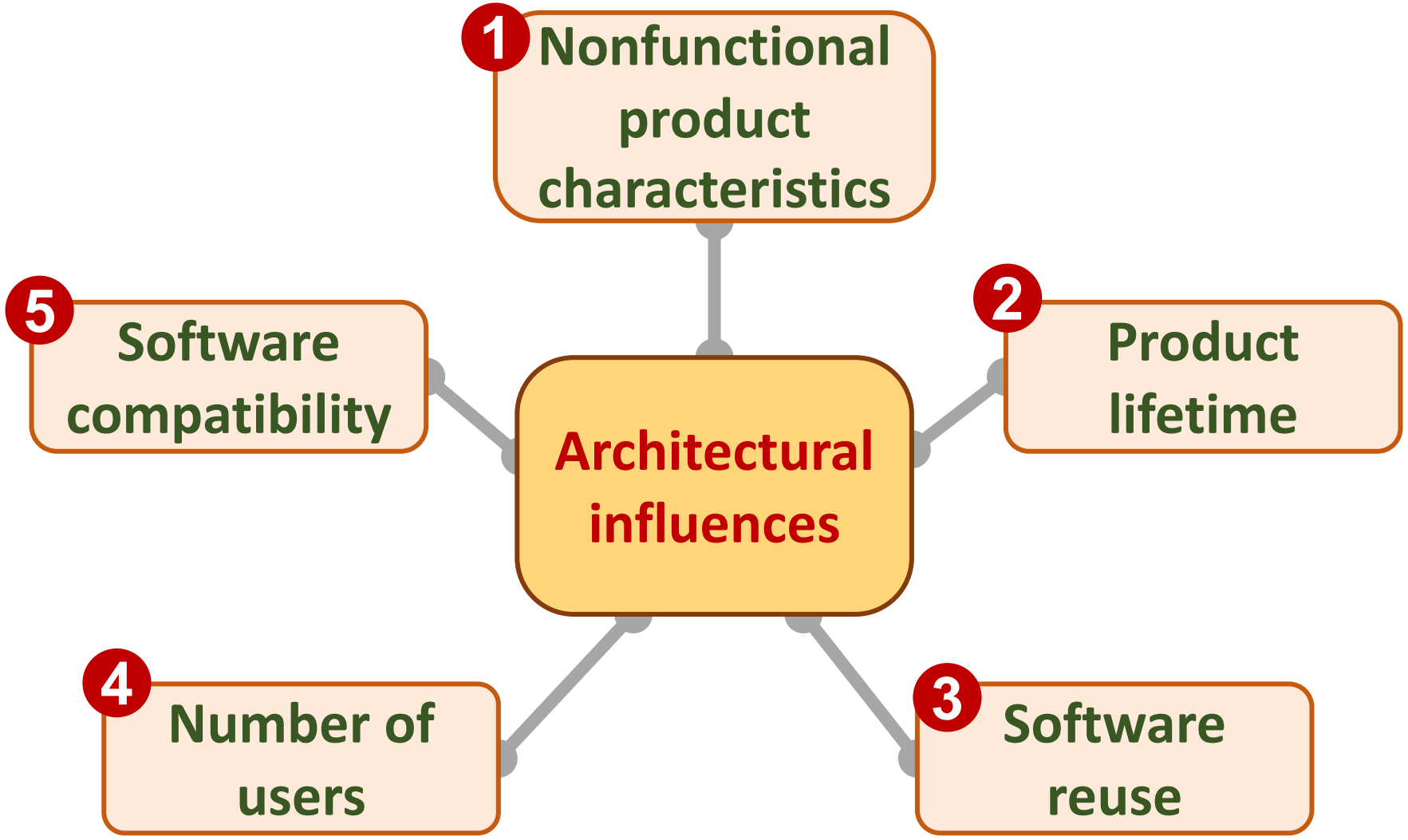
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

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

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

❶

- **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.

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

The importance of architectural design issues

❷

- **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.

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

The importance of architectural design issues

❸

- **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.

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

The importance of architectural design issues

4

- **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.

The importance of architectural design issues

❺

- **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.

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

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.

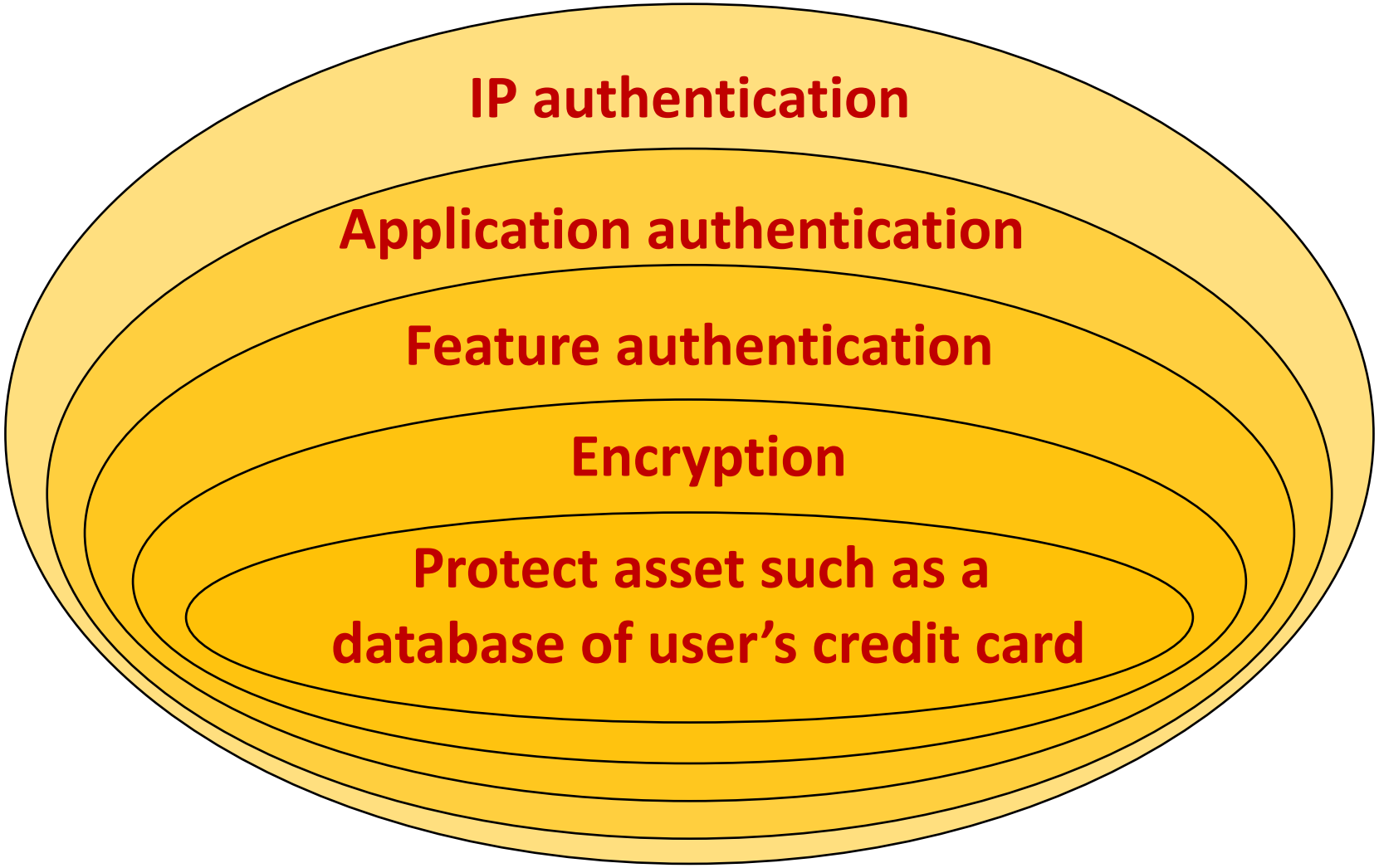
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

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

Authentication layers



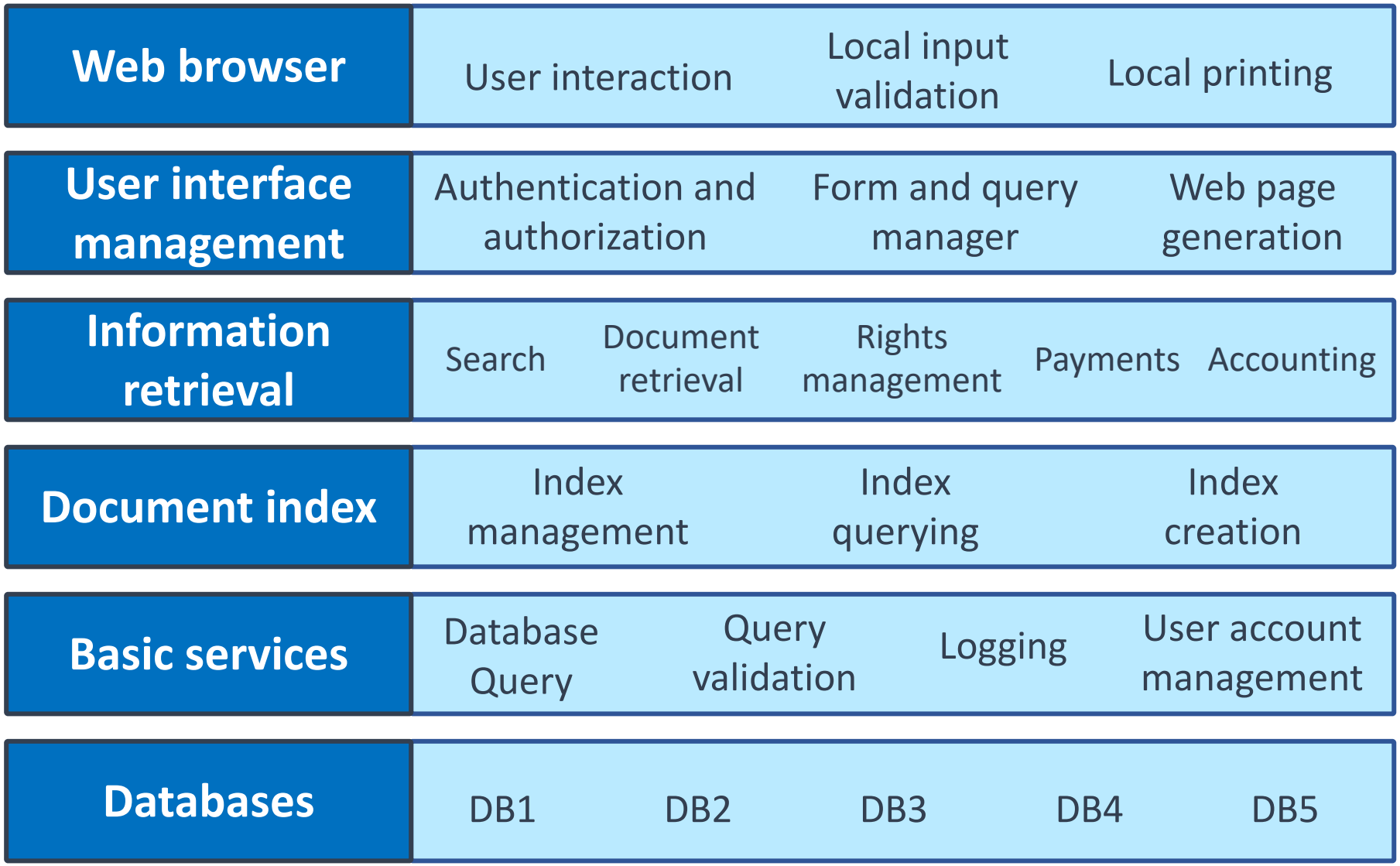
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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

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

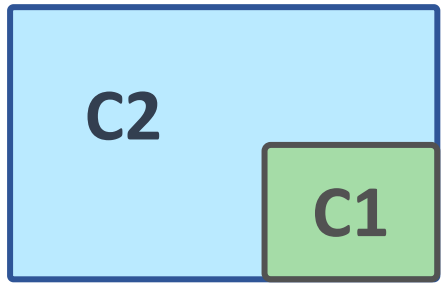
An architectural model of a document retrieval system



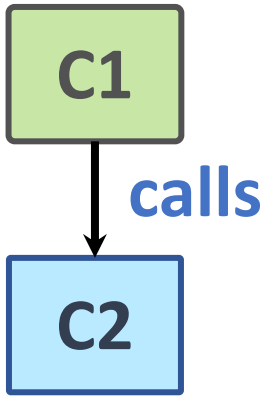
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Examples of component relationships

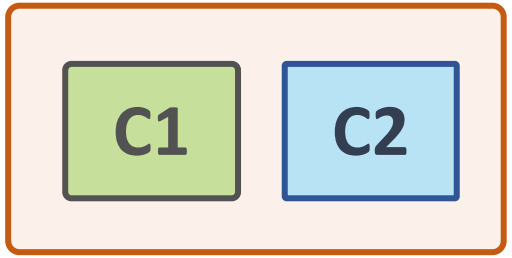
C1 is part of C2



C1 uses C2



C1 is-located-with C2



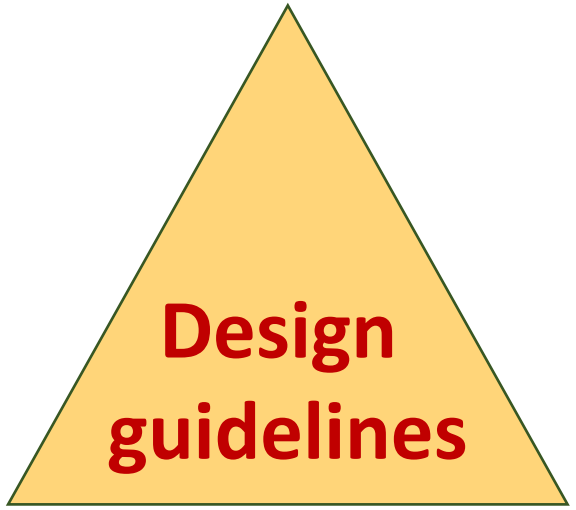
C1 shared-data-with C2

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

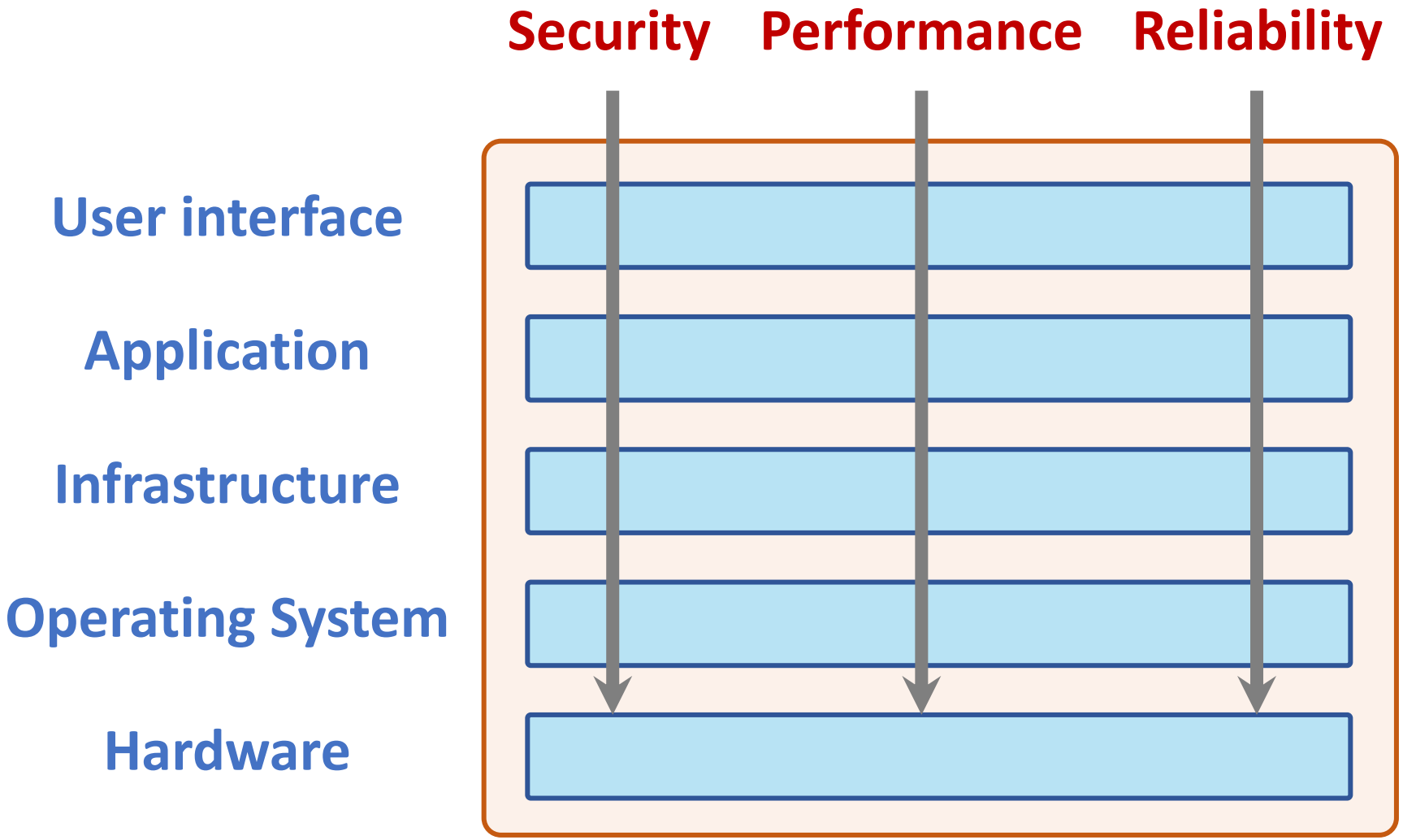
Design component interfaces that are coherent and that changes slowly

Implement once

Avoid duplicating functionality at different places in your architecture

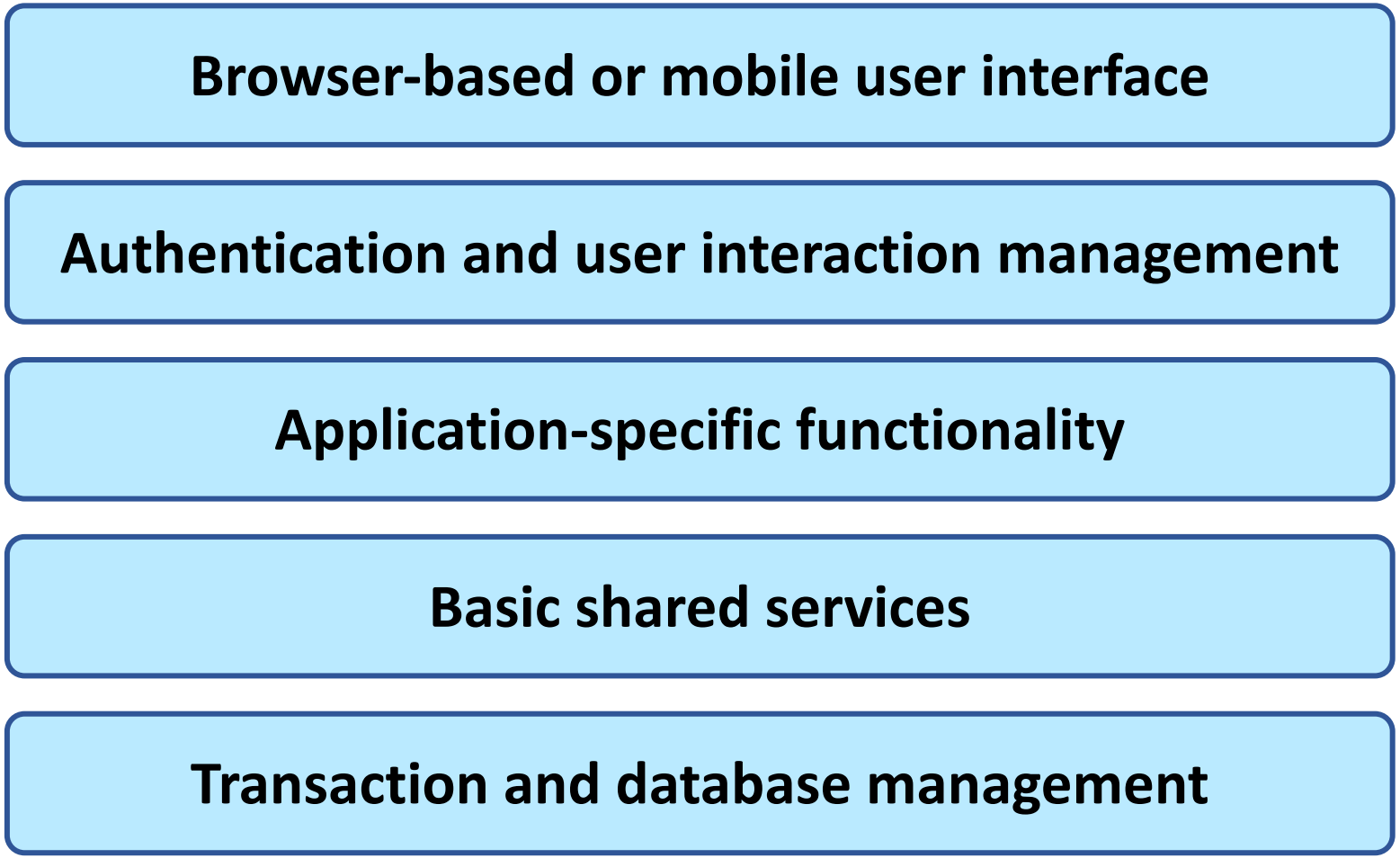
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Cross-cutting concerns



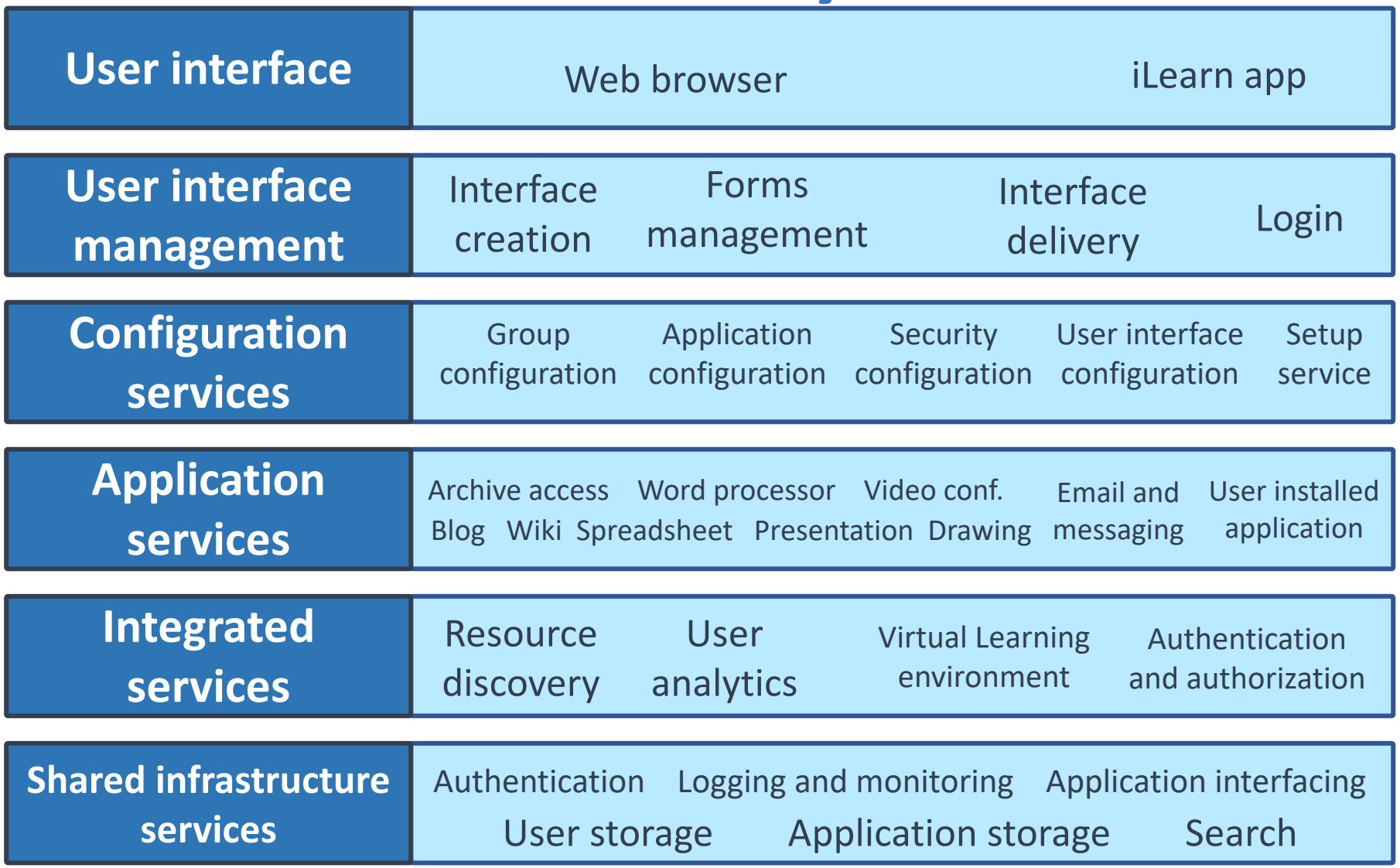
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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



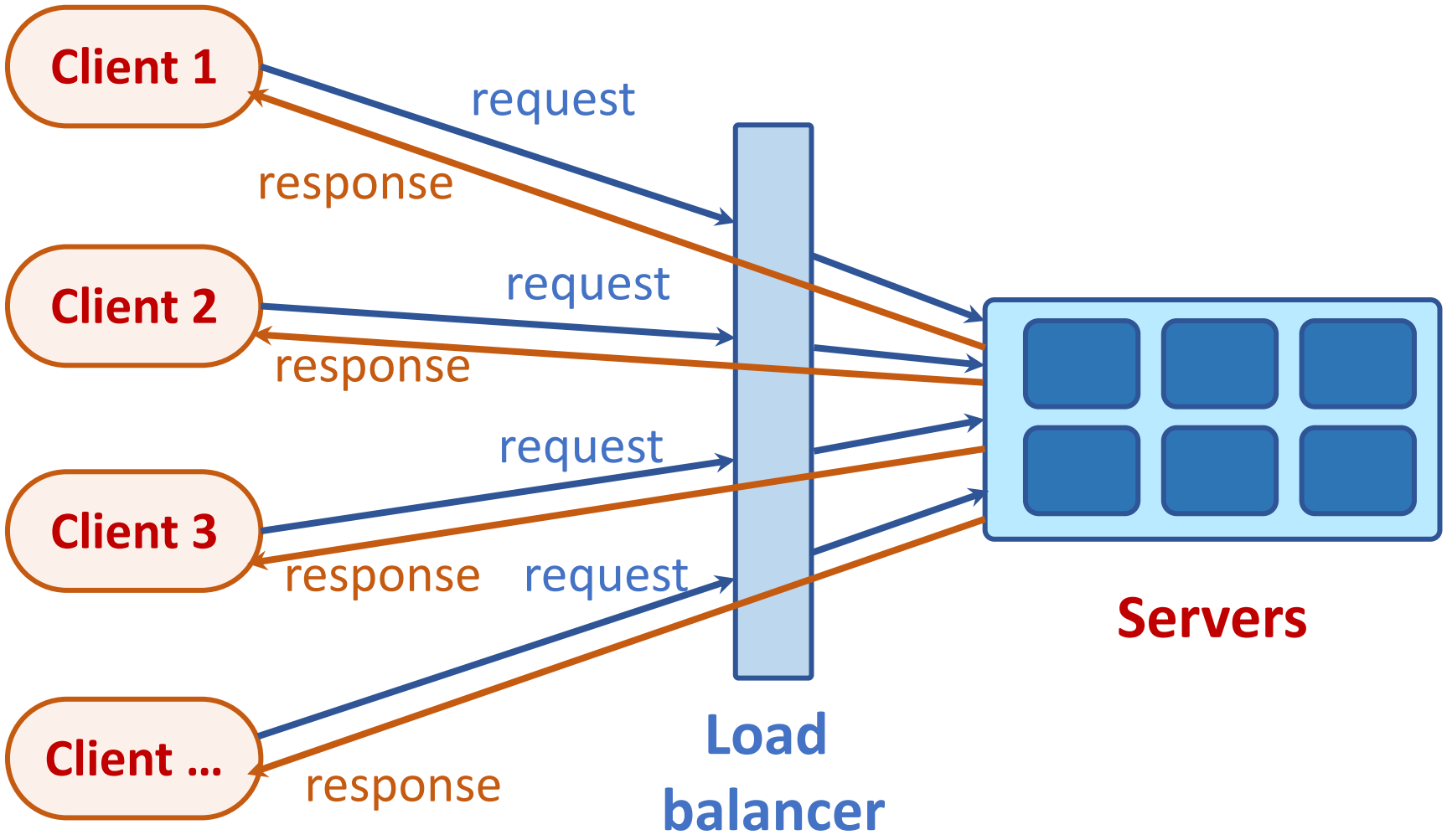
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

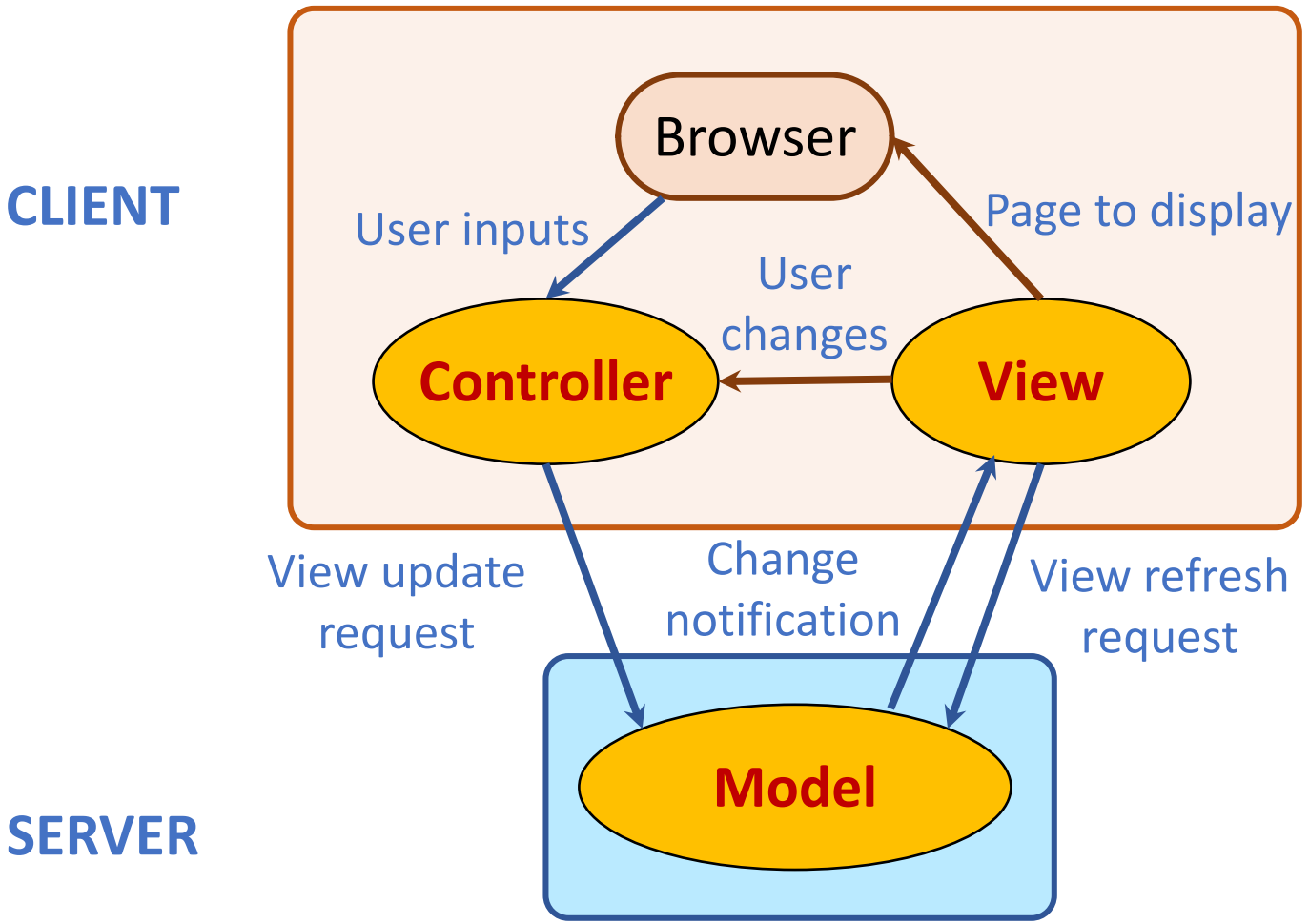
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Client-server architecture



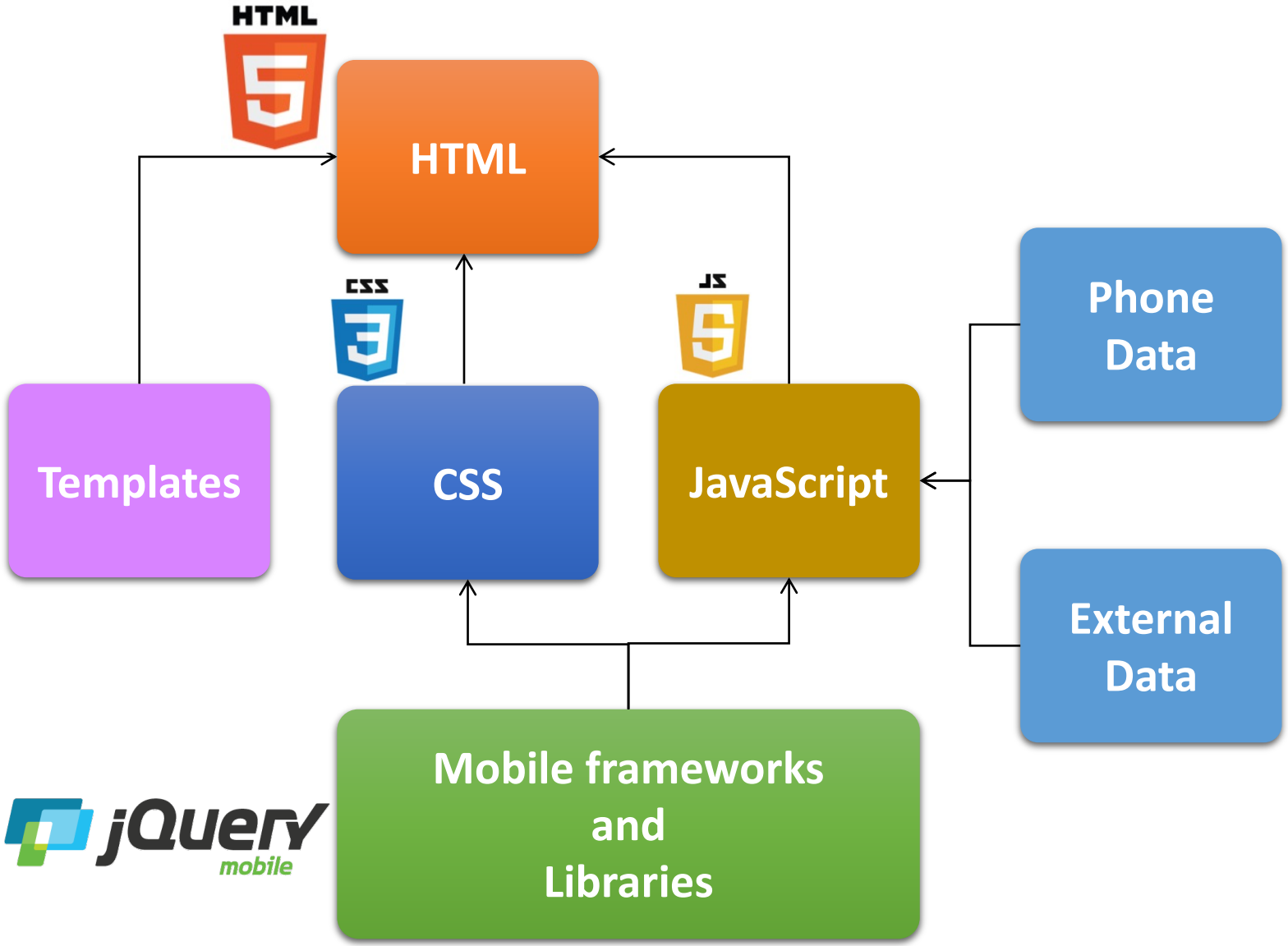
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

The Model-View-Controller (MVC) pattern



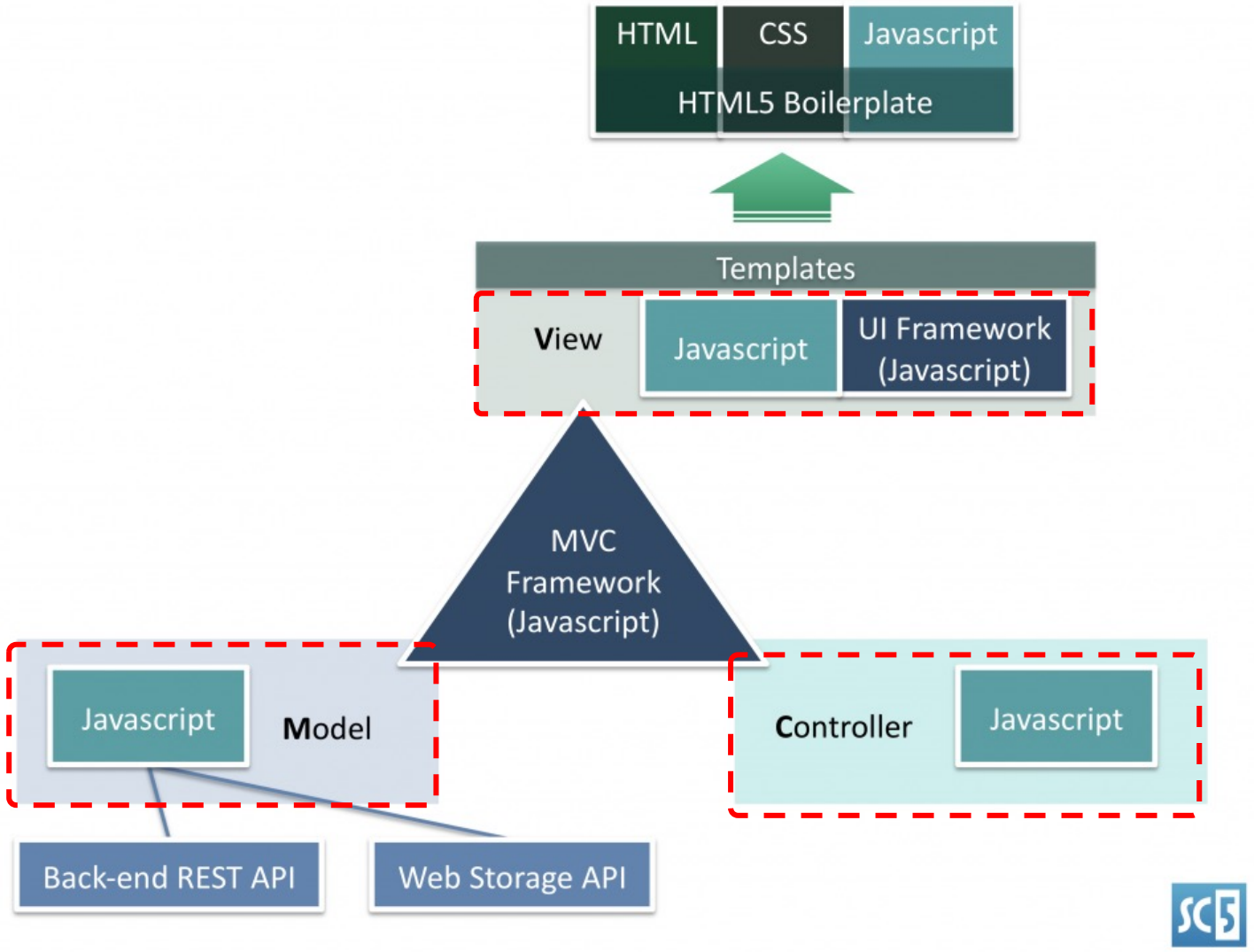
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Mobile Web App



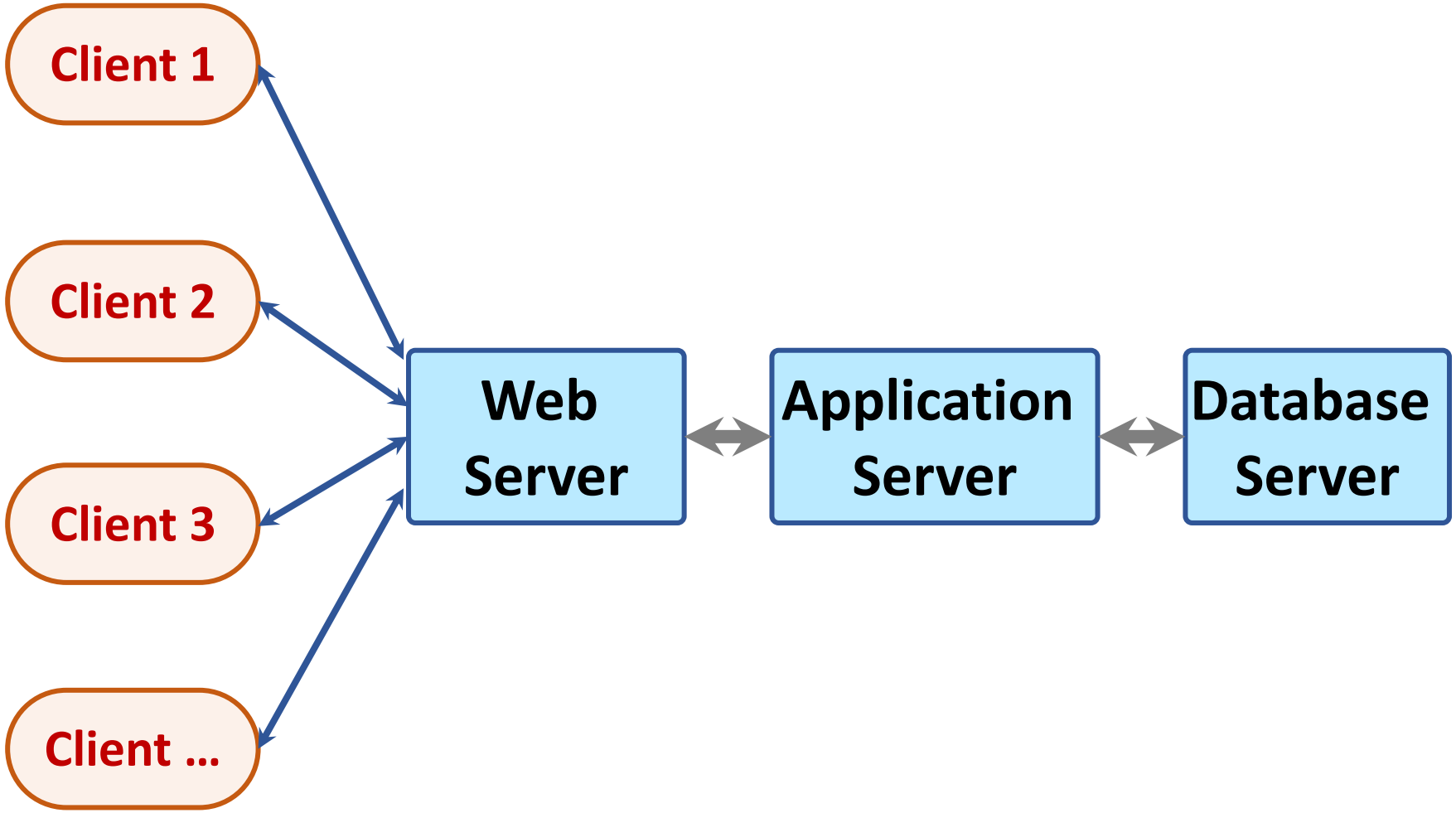
Source: Scott Preston, Learn HTML5 and JavaScript for iOS: Web Standards-based Apps for iPhone, iPad, and iPod touch, Apress, 2012

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



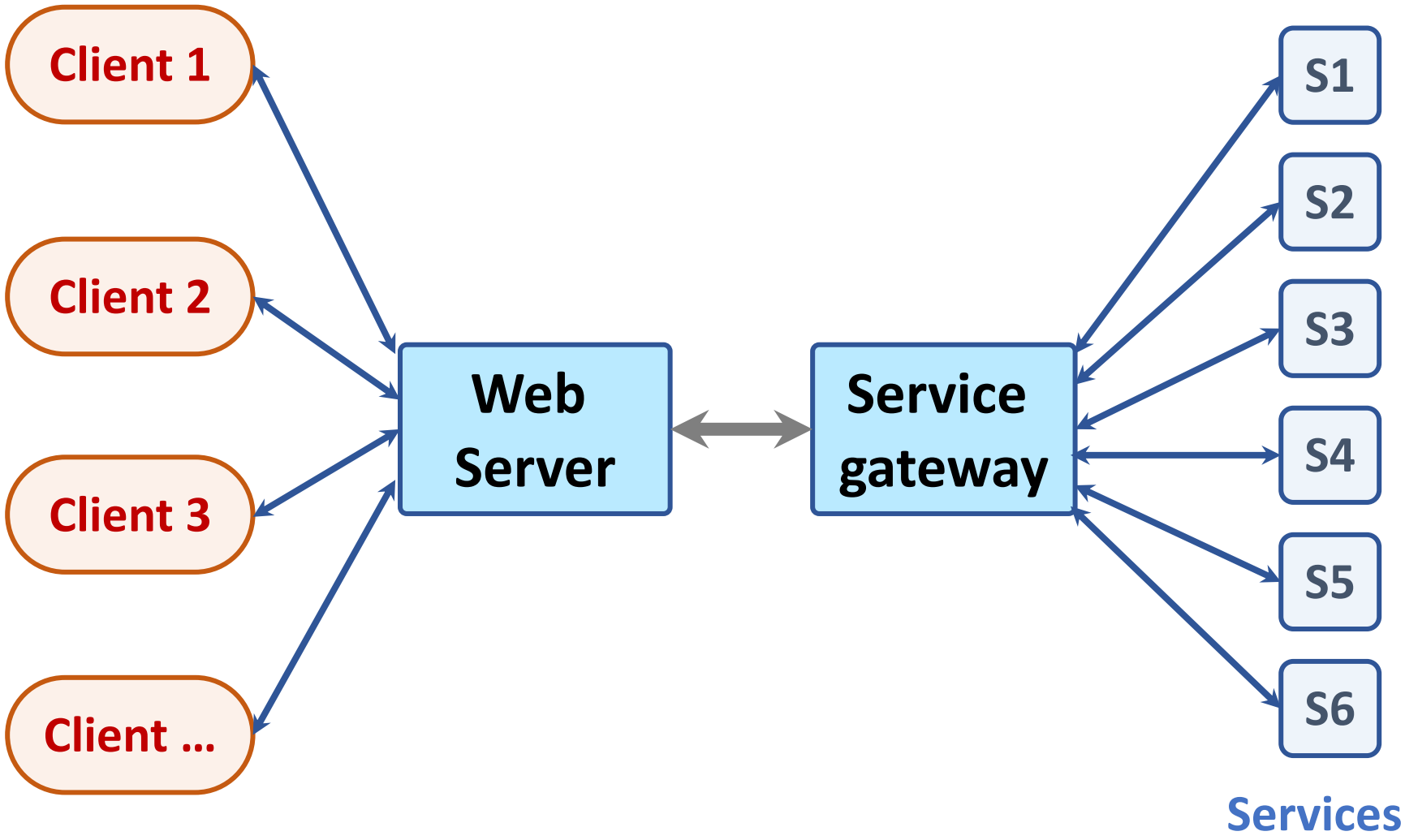
Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

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.

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.

Issues in architectural choice

- **Data type and data updates**
- **Change frequency**
- **The system execution platform**

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.**

Issues in architectural choice

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

Issues in architectural choice

- **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.

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

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.
- Thomas R. Caldwell (2025), The Agentic AI Bible: The Complete and Up-to-Date Guide to Design, Build, and Scale Goal-Driven, LLM-Powered Agents that Think, Execute and Evolve, Independently published
- Tucker J. Marion, Mahdi Srour, and Frank Piller (2024), "When Generative AI meets product development." MIT Sloan Management Review 66, no. 1 : 14-15.
- Yilei Zhao, Wentao Zhang, Xiao Lei, Yandan Zheng, Mengpu Liu, and Wei Yang Bryan Lim. (2026) "Advancing ESG Intelligence: An Expert-level Agent and Comprehensive Benchmark for Sustainable Finance." arXiv preprint arXiv:2601.08676 (2026).
- Nesreen Otoum and Nuha Elkhalili.(2026) "Methods and Techniques of Agentic Software Engineering: A Systematic Literature Review." IEEE Access 14 (2026): 7443-7465.
- Lakshana Iruni Assalaarachchi, Zainab Masood, Rashina Hoda, and John Grundy. (2026) "Toward Agentic Software Project Management: A Vision and Roadmap." arXiv preprint arXiv:2601.16392 (2026).
- NVIDIA DLI (2026), Building RAG Agents with LLMs, https://learn.nvidia.com/courses/course-detail?course_id=course-v1:DLI+S-FX-15+V1
- NVIDIA DLI (2026), Generative AI with Diffusion Models, https://learn.nvidia.com/courses/course-detail?course_id=course-v1:DLI+S-FX-14+V1
- NVIDIA DLI (2026), Building Agentic AI Applications with LLMs, https://learn.nvidia.com/courses/course-detail?course_id=course-v1:DLI+S-FX-41+V1