

Software Engineering - Design

Summer Term 2024
02 Architectures

Prof. Dr. Robert Lokaiczyk
Frankfurt University of Applied Sciences
Faculty of Computer Science and Engineering
robert.lokaiczyk@fb2.fra-uas.de

Announcement

Please register in HIS

<https://his.frankfurt-university.de/>

for the pre-qualification for the SWED exam.

Deadline: 2024-05-31

Without registration and and pre-qualification you will not be able to take part in the final exam.

Agenda for today

What is an architecture?

Why do we need an architecture?

Which architectural styles are commonly used?

Examples

- Pipes and Filters

- MVC

- Client Server

- Blackboard

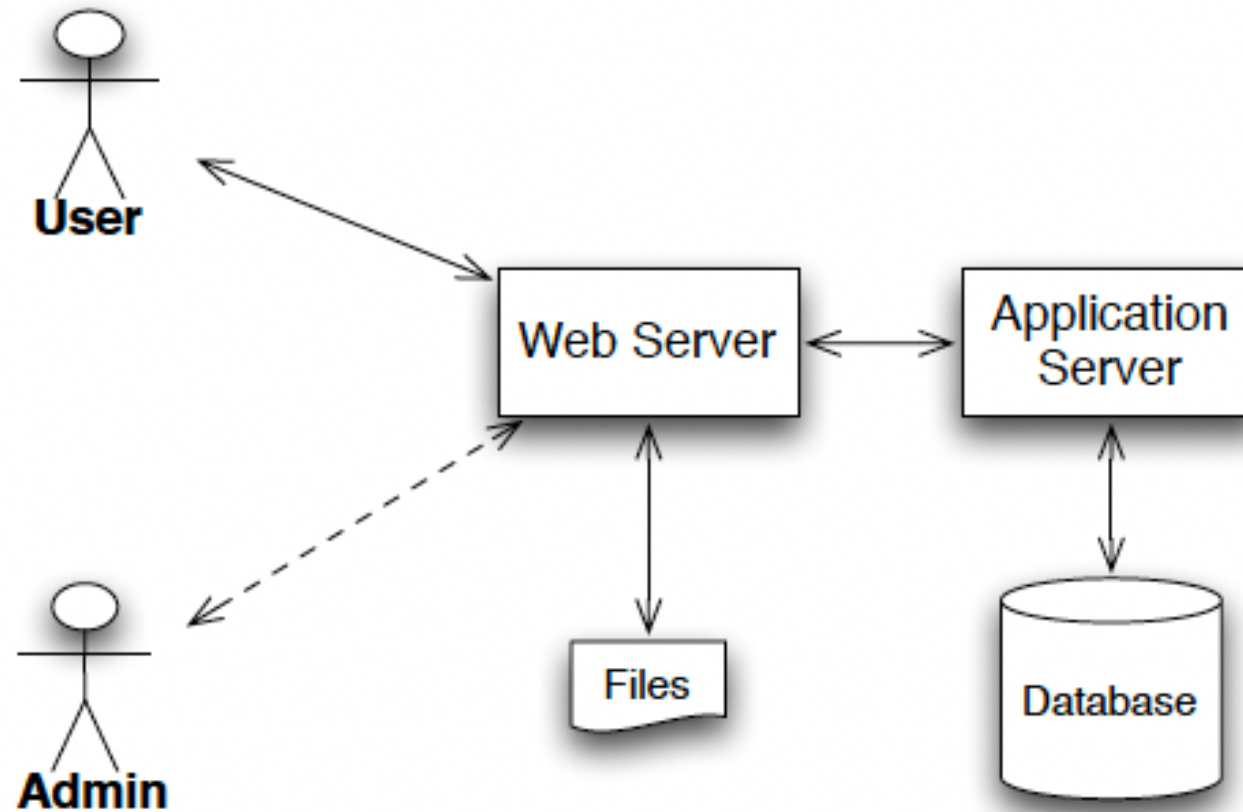
- Pub/Sub

- Microservices

The 4 views model

Example 1

Web application



Example 2

Car2Car communication

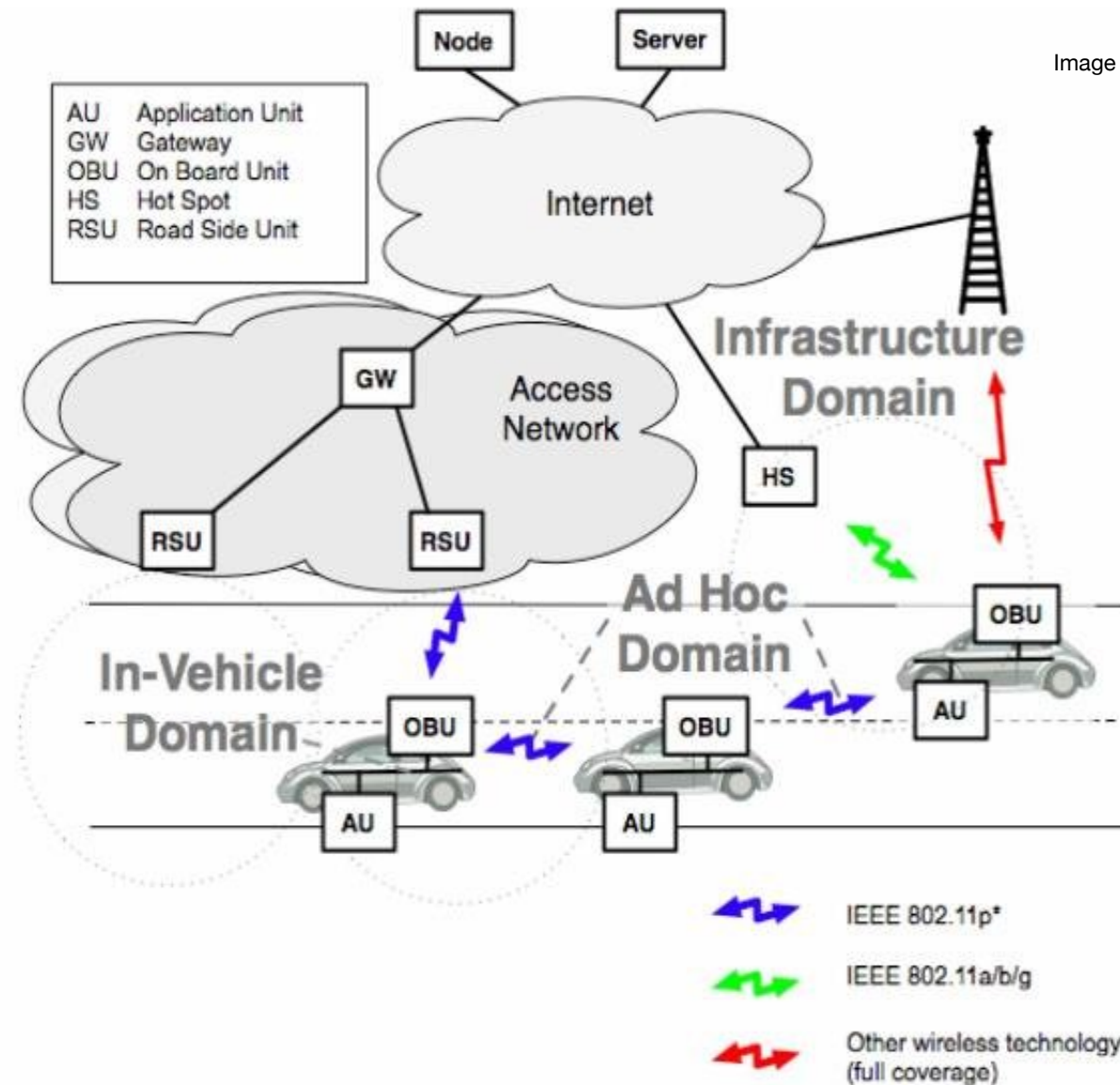


Image source: Taoufik Yeferny, Sofian Hamad CC-NC-ND 4.0

Questions

- What is the semantics (meaning) of the boxes?
- What is the semantics of the arrows?
- What is the semantics of the arrow tail and heads?
- Why are some lines dashed, while others are not?
- What ***is*** architecture?

Definition

software architecture.

(1) The **organizational structure** of a system or component.

See also component, module, subprogram, routine.

component.

(1) **One of the parts** that make up a **system**. A component may be **hardware or software** and **may be subdivided** into other components. Note: The term „component“, „module“ and „unit“ are often used interchangeably or defined sub elements of one another in different ways depending on the context. The relationship of these terms is not yet standardized.

IEEE Standard Glossary of Software Engineering Terminology, 1990

Risks without an architecture

- Result does **not fulfill** non-functional **requirements**
- System does not meet expectations
- **Maintenance** is **difficult** and error-prone
- New functionality is hard to integrate
- System **disliked** by developers
- Responsibilities by developers are not defined
- System disliked by users
- Software project is difficult to manage

Purpose of an architecture

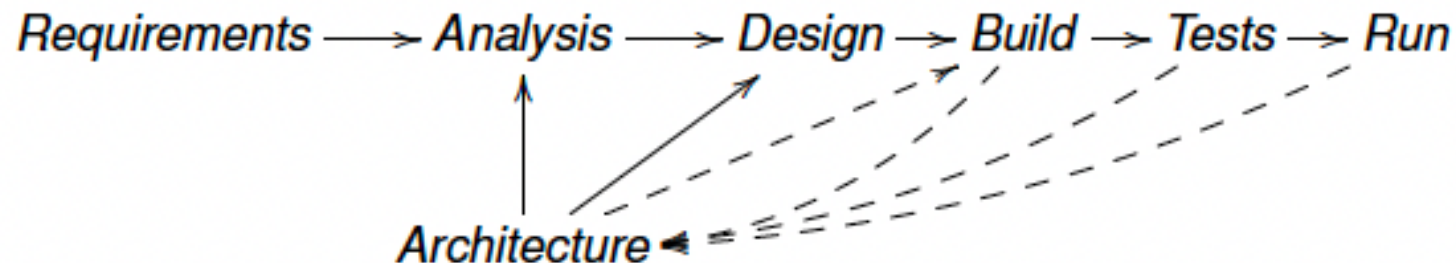
- Provide **overview** of the whole **system**
- **Visualize** the structure
- Separate **responsibilities**
- Assign software modules to **devices**
- Align **functionality** with the customer
- **Manage** the project

Each architecture has to encompass

- **Components**
 - elements
 - modules
- **Relationships**
 - communication links
 - data exchange
- **Explanations** for each element and relationships
- **Legend**

The feedback loop

- An architecture typically is **not final after the design phase**. Programming phase, testing phase and production phase deliver feedback for improvements.



- Software architecture can be considered as a form of art where aesthetics matters. A good architect will learn from pieces of other masters.

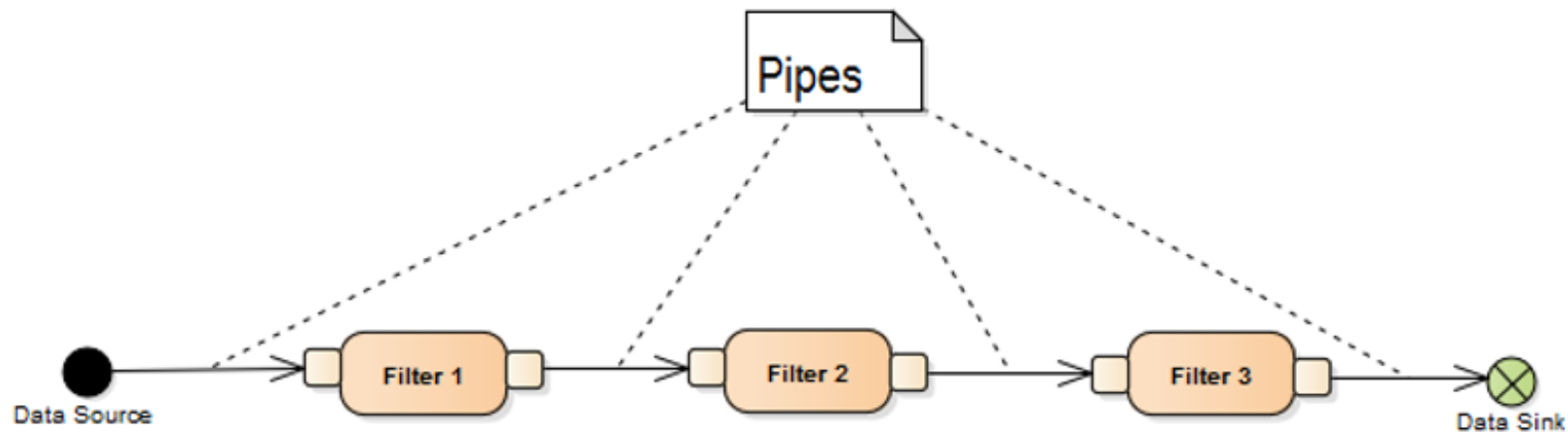
Architectural styles

Type	Description	Instances
Dataflow-centric	Consist of a sequence of data and operations	Batch-Sequential Pipes and Filters
Data-centric	Shared, central data source	Repository Blackboard
Hierarchical	Consists of ordered parts in different hierarchical layer	Master-Slave Layered Ports and Adapter
Distributed systems	Consists of storage and processing units that communicate through networks	Client-Server Broker Peer to Peer
Event-based	Independent elements that communicate and call each other via events	Publish-Subscriber Message Queue
Service-oriented	Divides app into small, independent services that communicate through standard protocols	Broker Microservices „Serverless“

...

Pipes and Filters

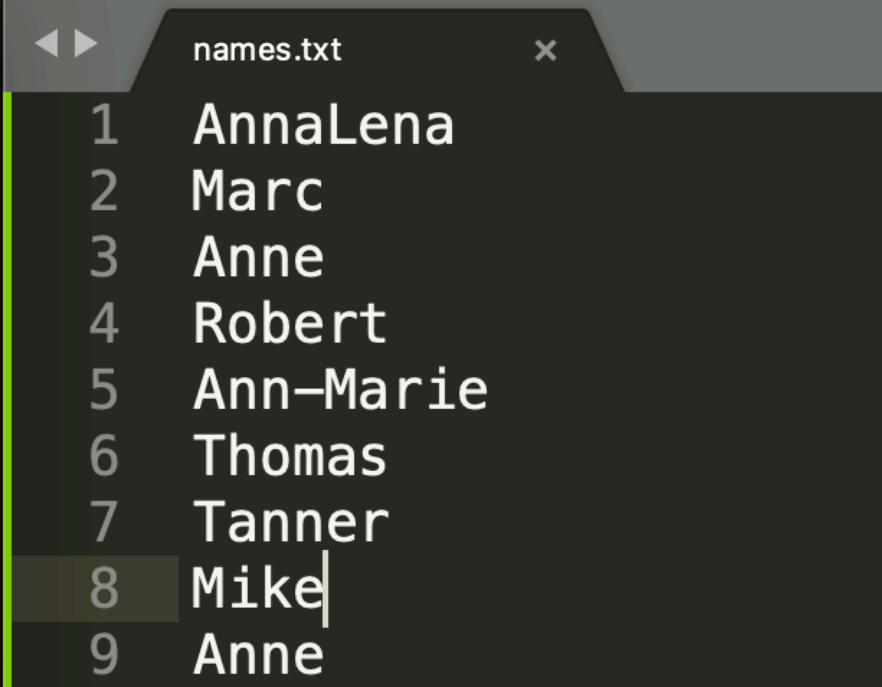
- A **filter** is a processing step that **transforms data**: add, removes or replaces elements
- A **pipe** is a **connection between filters**
- Each output of a filter is the input of another filter.



Pipes and Filters

Example: the Linux command line

```
[bash-3.2$ cat names.txt
AnnaLena
Marc
Anne
Robert
Ann-Marie
Thomas
Tanner
Mike
Anne
[bash-3.2$ cat names.txt | grep Ann
AnnaLena
Anne
Ann-Marie
Anne
[bash-3.2$ cat names.txt | grep Ann | sort
Ann-Marie
AnnaLena
Anne
Anne
[bash-3.2$ cat names.txt | grep Ann | sort | uniq
Ann-Marie
AnnaLena
Anne
[bash-3.2$ cat names.txt | grep Ann | sort | uniq | wc -l
3
bash-3.2$
```



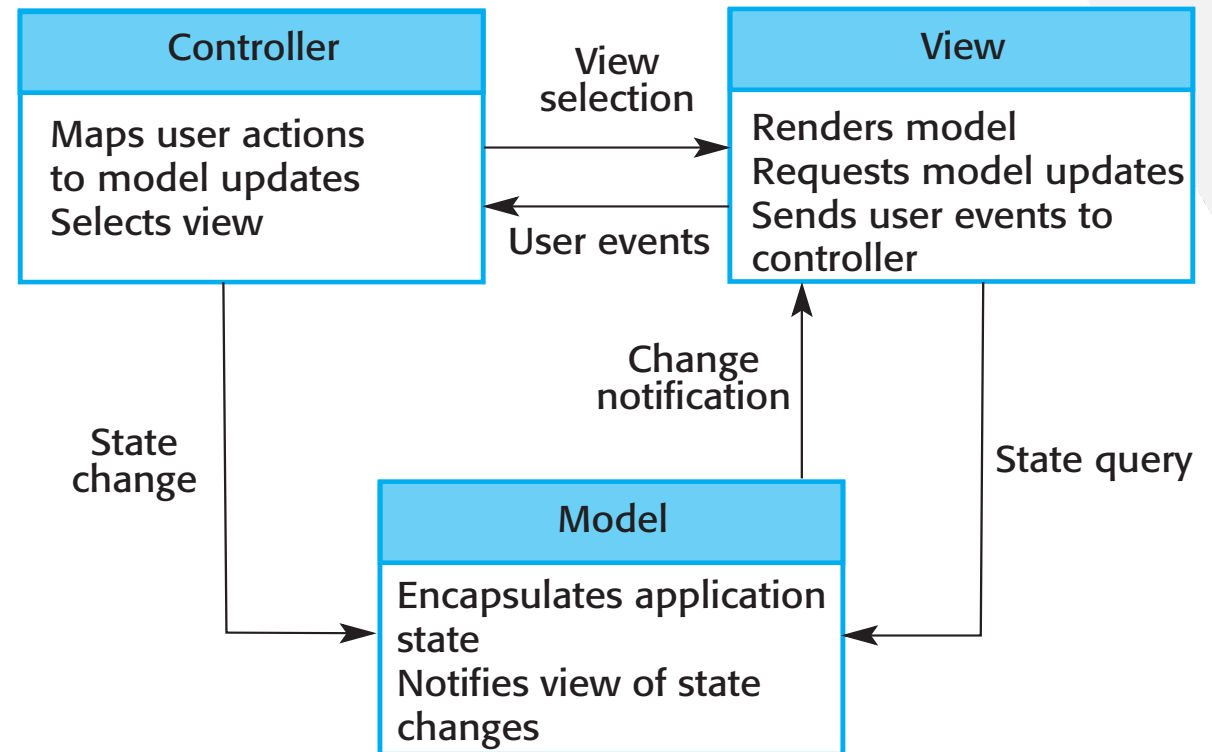
```
names.txt
1 AnnaLena
2 Marc
3 Anne
4 Robert
5 Ann-Marie
6 Thomas
7 Tanner
8 Mike
9 Anne
```

Model View Controller

MVC is a design pattern widely used in software engineering for developing **user interfaces**.

It separates an application into **three** interconnected **components**: Model, View, and Controller.

Each of these components has a specific role and responsibility within the system.



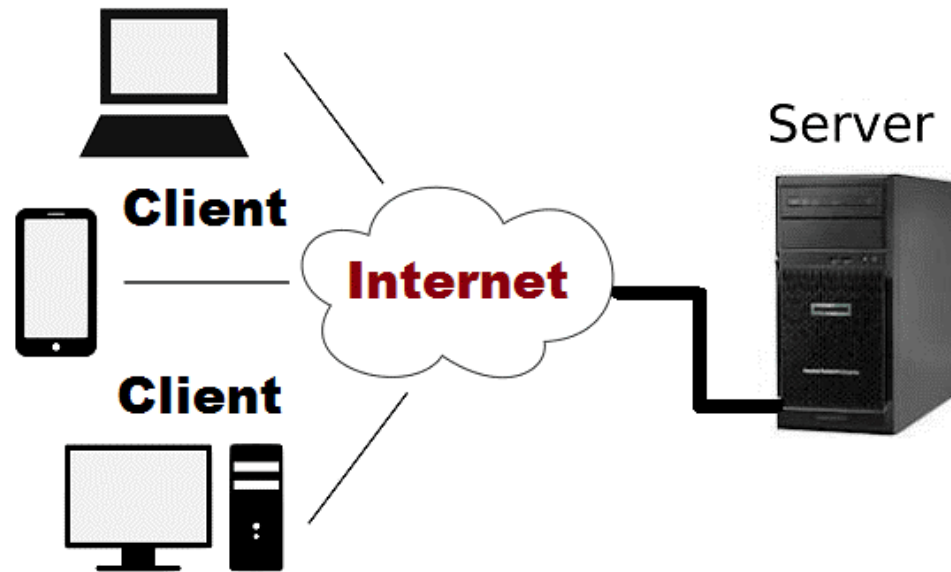
Responsibilities

Model	View	Controller
<p>represents the application's data and business logic.</p> <p>Encapsulates the data and provides methods to manipulate that data.</p> <p>Notifies the View and Controller when the data changes, typically using an observer pattern or similar mechanisms.</p> <p>should be independent of the user interface and any specific presentation or interaction logic.</p>	<p>represents the presentation layer of the application.</p> <p>displays the data from the Model to the user and provides the necessary interface elements for user interaction.</p> <p>responsible for rendering the user interface elements and responding to user input.</p> <p>observe changes in the Model and update themselves accordingly to reflect any changes in the underlying data.</p>	<p>acts as an intermediary between the Model and the View.</p> <p>receives user input (mouse or keyboard) from the View, processes it (e.g., validation, formatting), and updates the Model accordingly.</p> <p>Handles user interactions and translate them into actions on the Model or the View.</p> <p>orchestrate the flow of data and control the application's behavior.</p>

Advantages

- **Separation of concerns:** Each component has a distinct role, promoting code organization and maintainability.
- **Reusability:** Components can be reused across different parts of the application or even in different applications. You can have multiple Views on one Model.
- **Testability:** Components can be tested independently, facilitating unit testing and ensuring code quality.
- **Scalability:** The modular structure allows for easier scaling of the application as it grows in complexity.
- MVC is a **widely used** in web development frameworks, in desktop and mobile applications.

Client Server architecture



The client-server style is the most frequently encountered of the architectural styles for **network-based applications**. A **server** component, offering a set of services, **listens for requests** upon those services. A **client** component, desiring that a service be performed, **sends a request** to the server via a connector, mostly HTTP. The server either rejects or performs the request and **sends a response** back to the client.

Discussion

Advantages

- **Scalability:** Upgrading the central server instance allows better performance as the system grows.
- **Centralized Data Management:** Facilitates efficient data management and maintenance by storing and managing data in a central location.
- **Resource Sharing:** Enables efficient utilization of shared resources among multiple clients, reducing redundancy and improving overall efficiency.
- **Security:** Centralized servers can implement robust security measures to protect sensitive data and resources, ensuring better security and compliance.
- **Reliability and Fault Tolerance:** Implementing redundancy and failover mechanisms at the server level ensures high availability and minimizes downtime in case of failures.

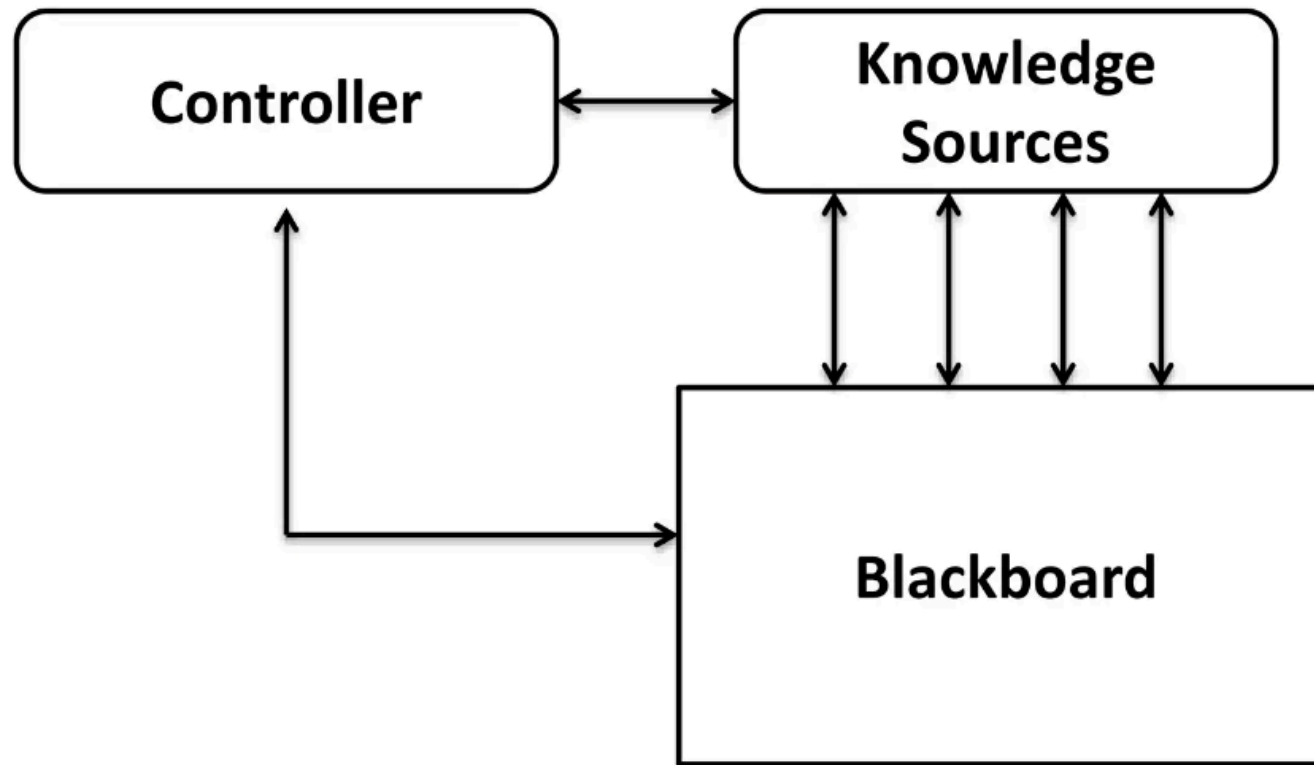
Disadvantages

- **Single Point of Failure:** Dependency on a central server poses a risk of system-wide failure if the server malfunctions or becomes unavailable.
- **Network Dependency:** Performance and availability are dependent on the quality and reliability of the network infrastructure, which can be prone to issues.
- **Scalability Challenges:** Scaling may face limitations and complexities, especially for large-scale deployments, requiring additional resources (costs) and management efforts (dynamic scaling).

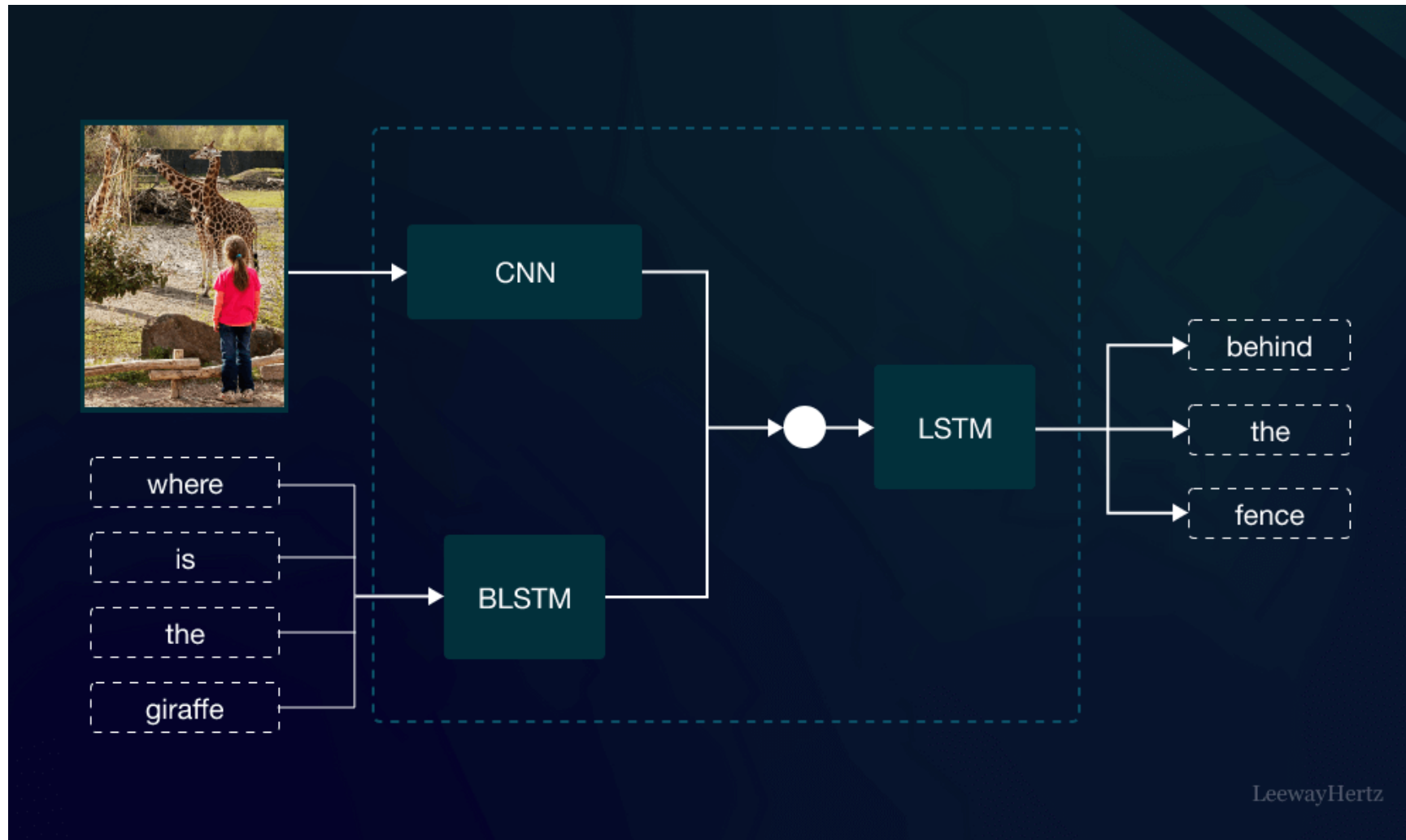
Blackboard architecture

- One common data structure stored on the blackboard
- Components act independently on this structure and not with each other
- Alerts upon data-store change triggers notifications to the subscribed clients
- Analogy: Students in a classroom
 - Announcement of the problem
 - Students watch blackboard for opportunity to make a contribution
 - Recording of contribution on the blackboard
 - Others can use new information to continue
 - Teacher manages
- When to apply? When a single expert is not able to solve the problem

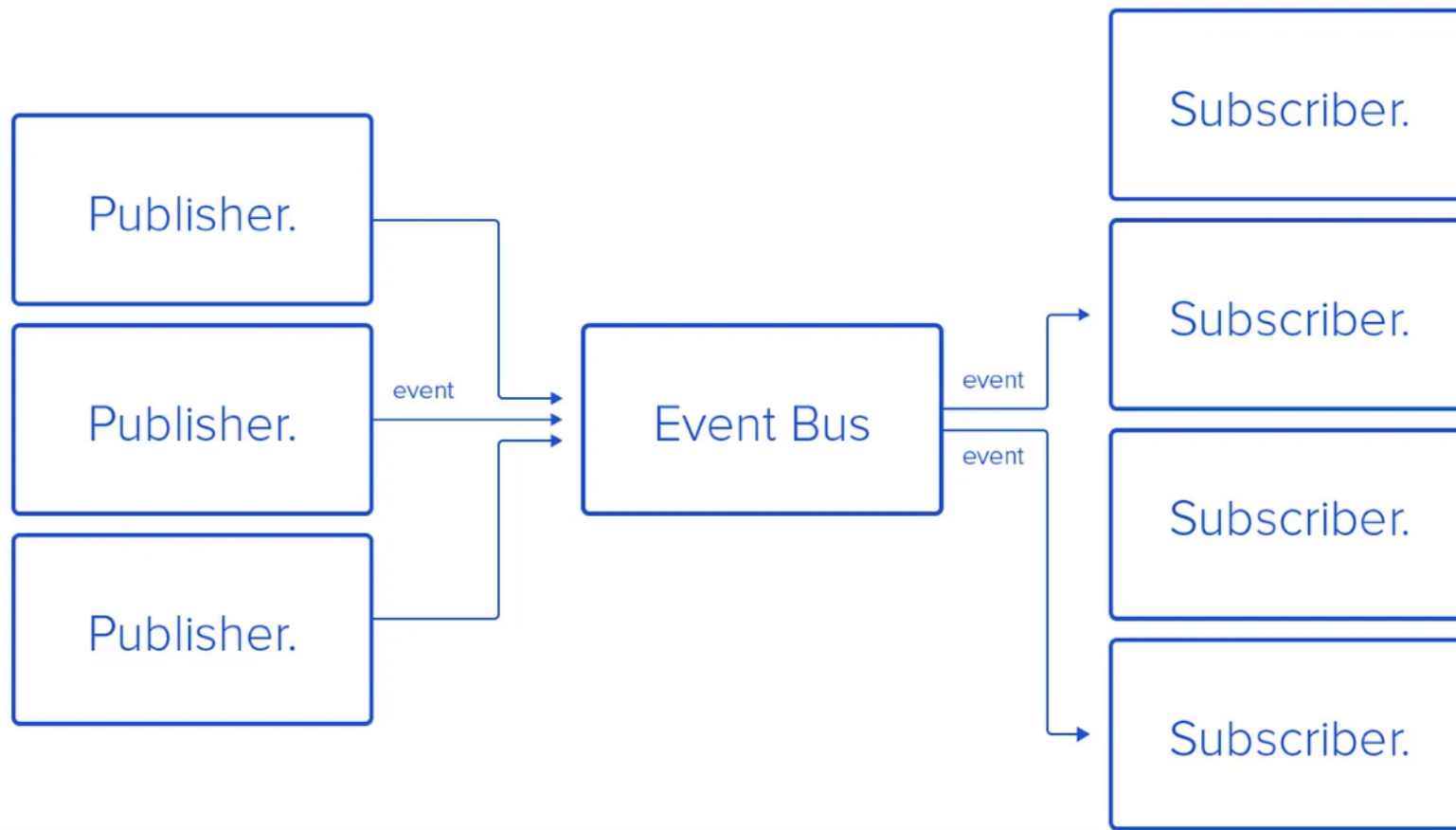
Blackboard architecture



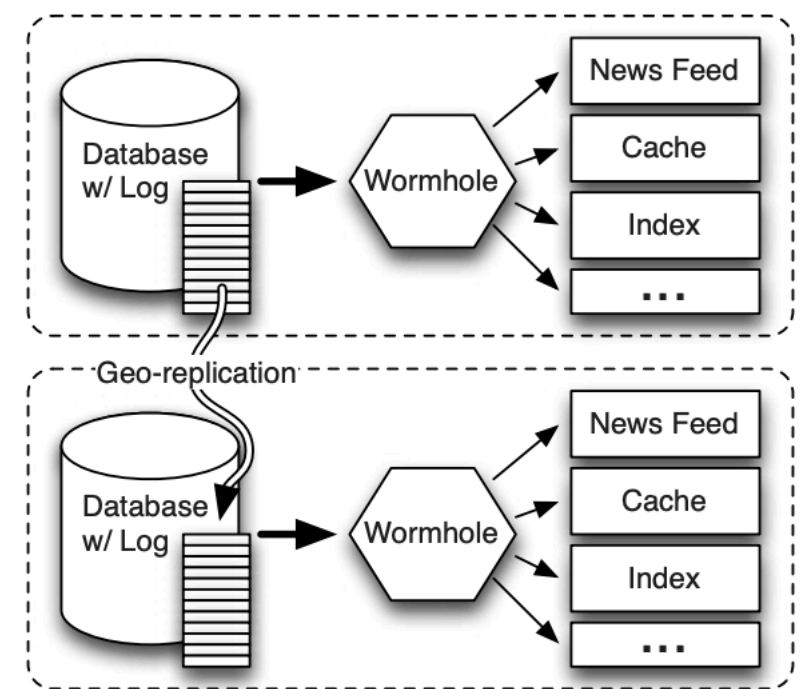
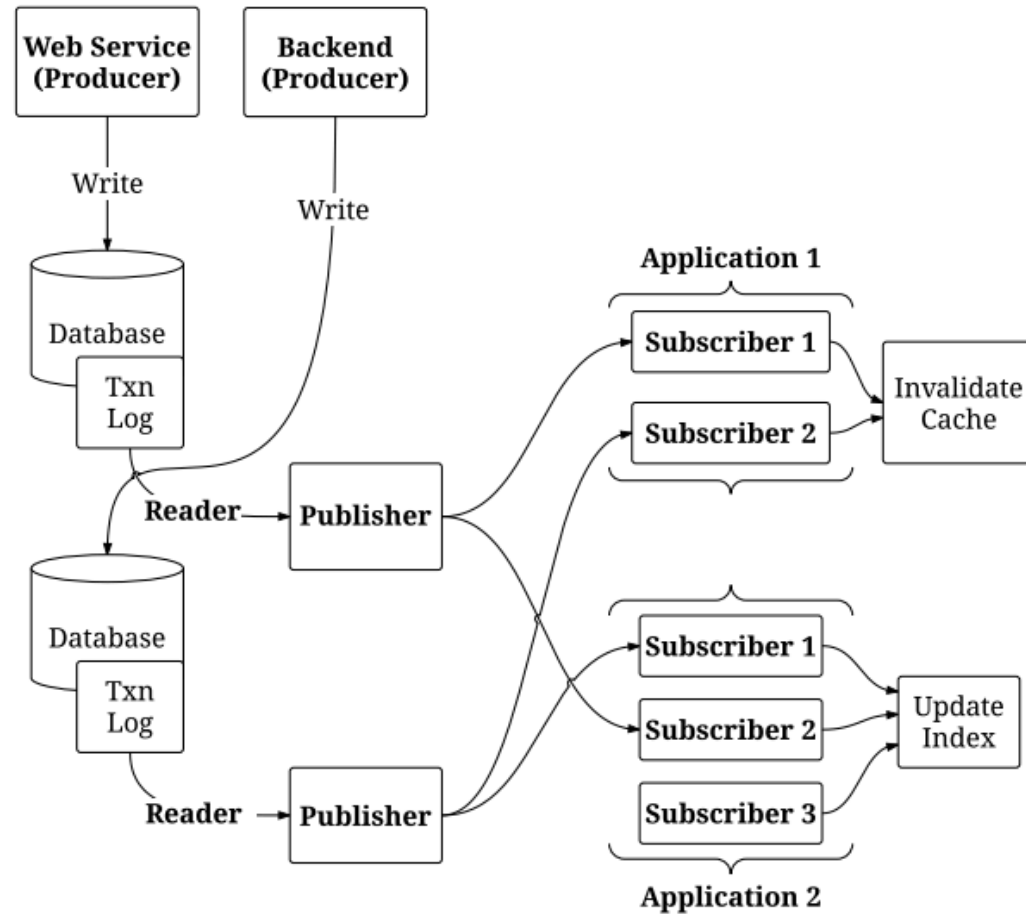
Usage in Multimodal Large Language Models (M-LLMs)



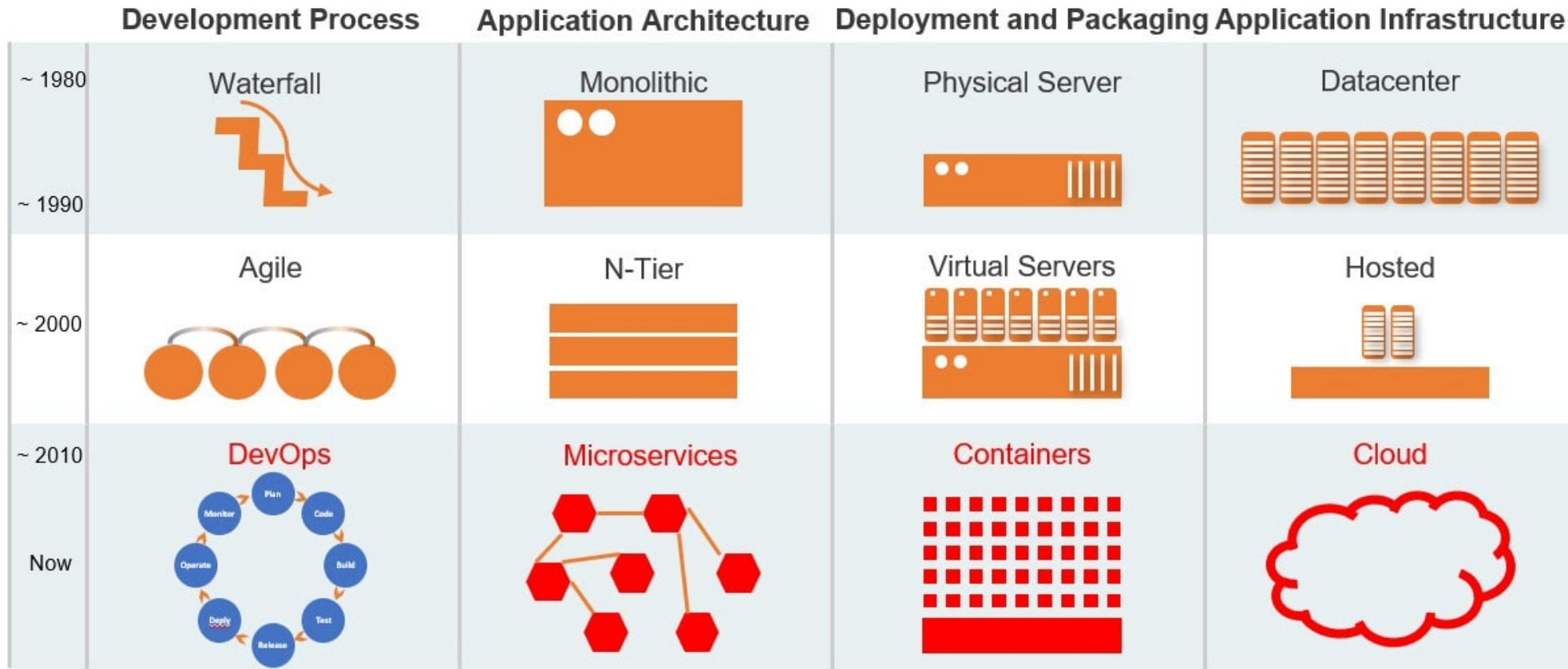
Pub/Sub architecture



Example: Facebook Wormhole



The evolution of architectures

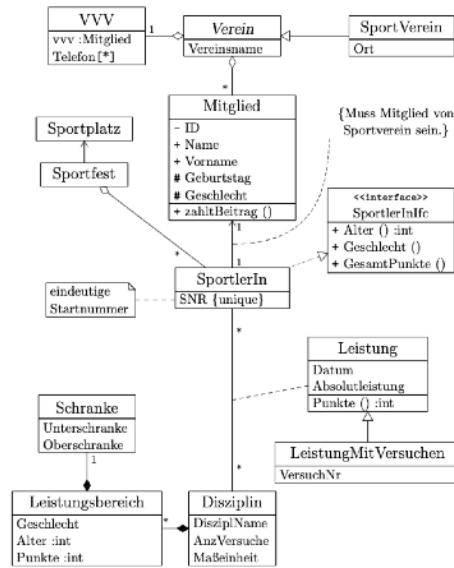


The 4+1 view model of software architecture

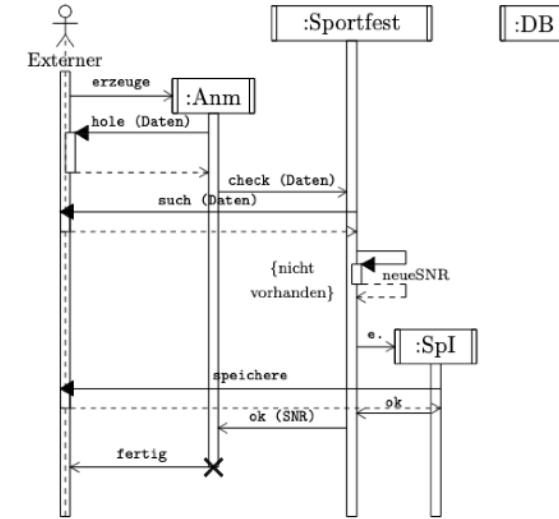
- A **logical view**, which shows the key abstractions in the system as objects or object classes.
- A **process view**, which shows how, at run-time, the system is composed of interacting processes.
- A **development view**, which shows how the software is decomposed for development.
- A **physical view**, which shows the system hardware and how software components are distributed across the processors in the system.

The views correspond to UML diagram types

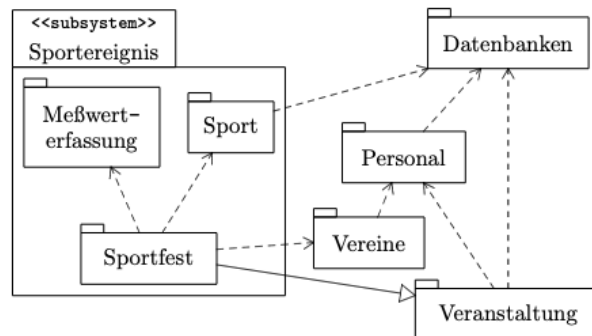
Logical view



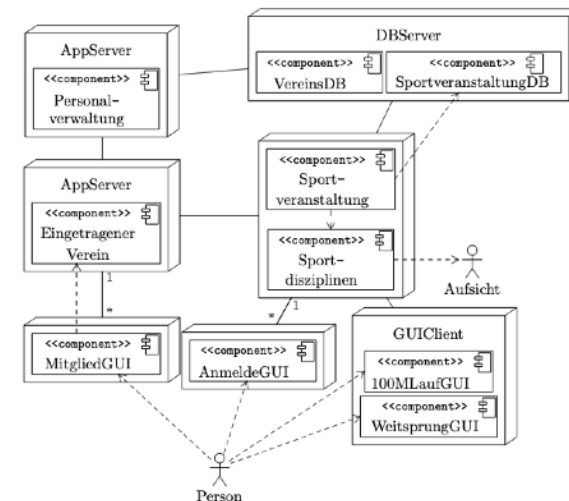
Process view



Development view



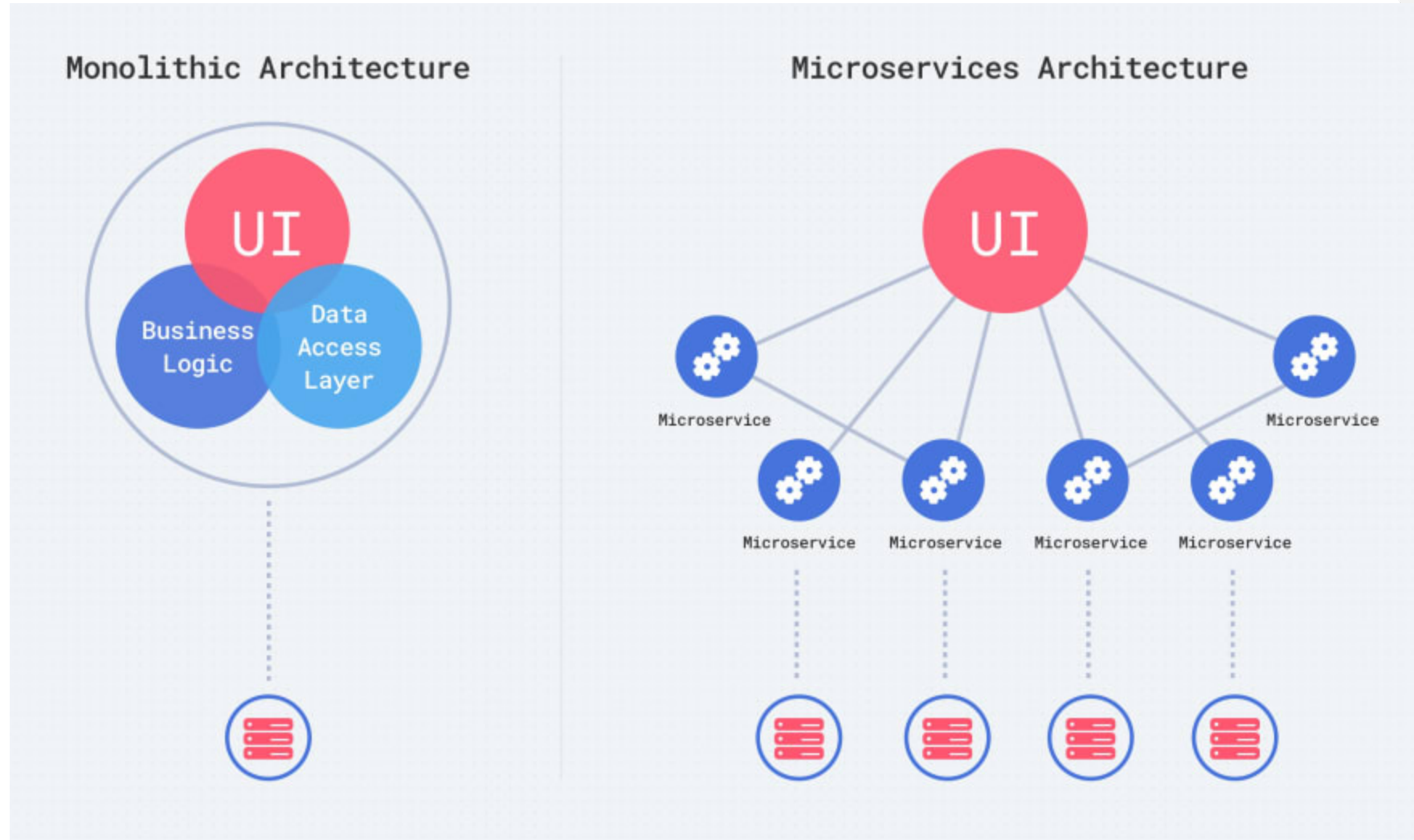
Deployment view



Microservices

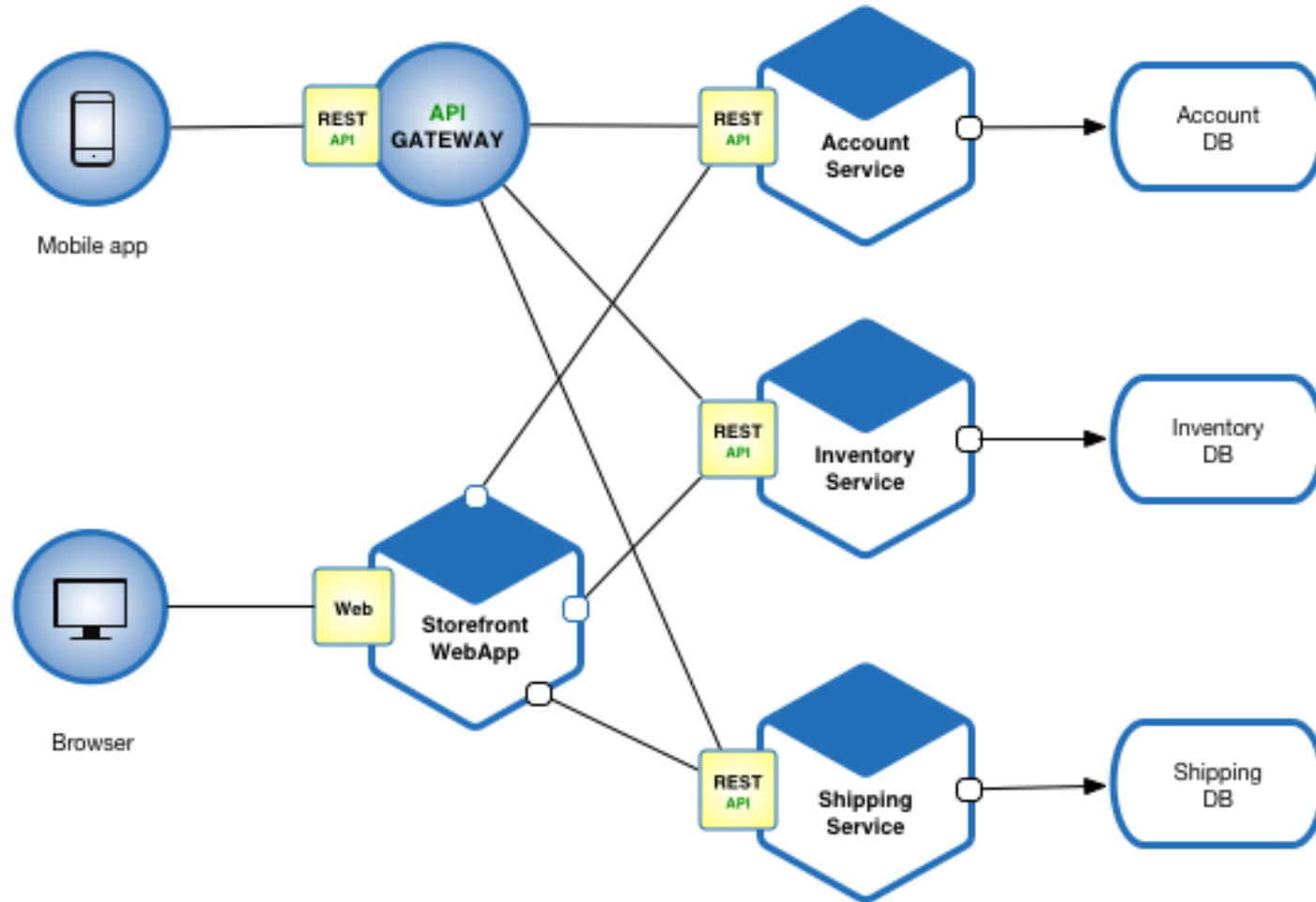
- A microservices architecture is a software development approach where an application is **composed of loosely coupled, independently deployable services**.
- Each service is **focused on a specific business capability** and can be **developed, deployed, and scaled independently**.
- **Communication** between services typically occurs through **lightweight protocols such as HTTP/REST** or messaging queues.
- Idea: scale applications also in changing business requirements, Stay **flexible, agile and scalable**

Microservices



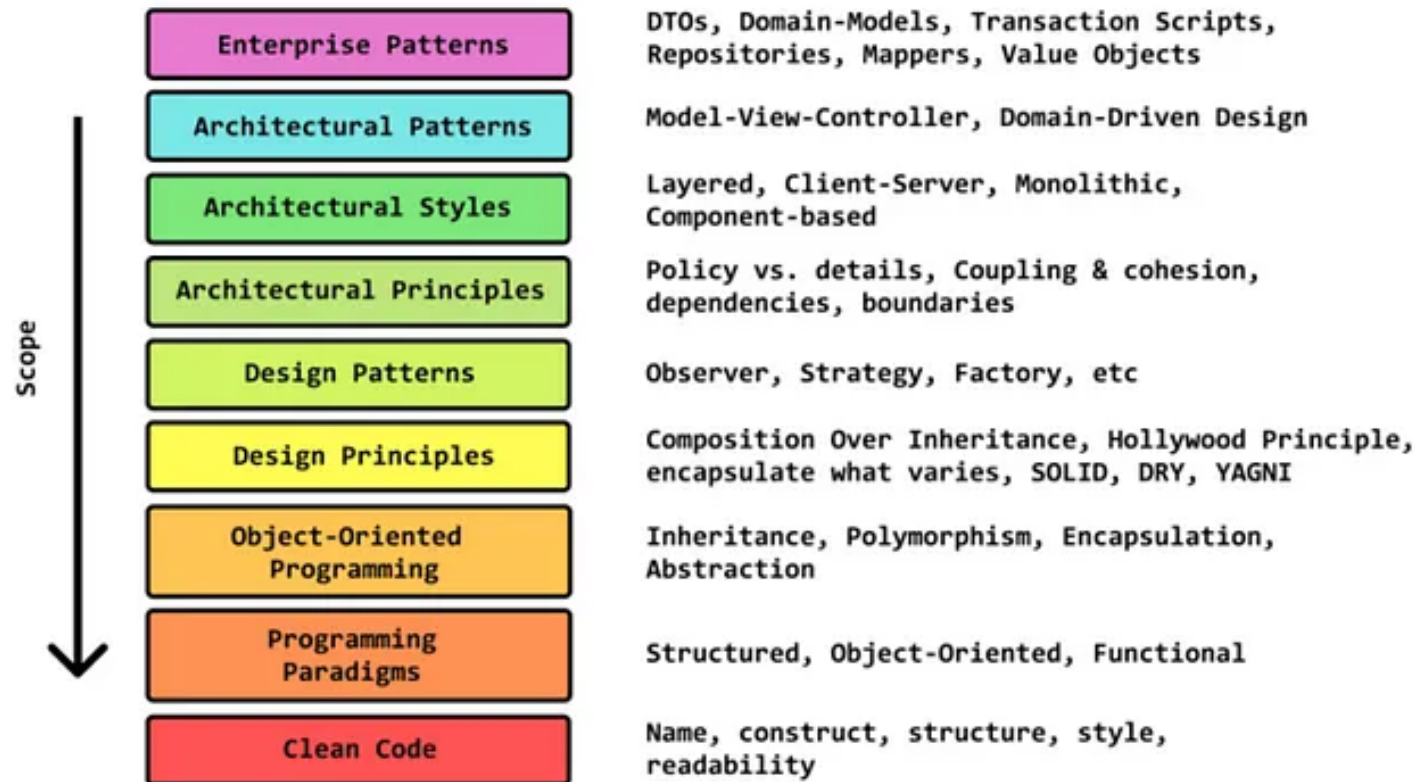
Example Microservices

E-commerce application

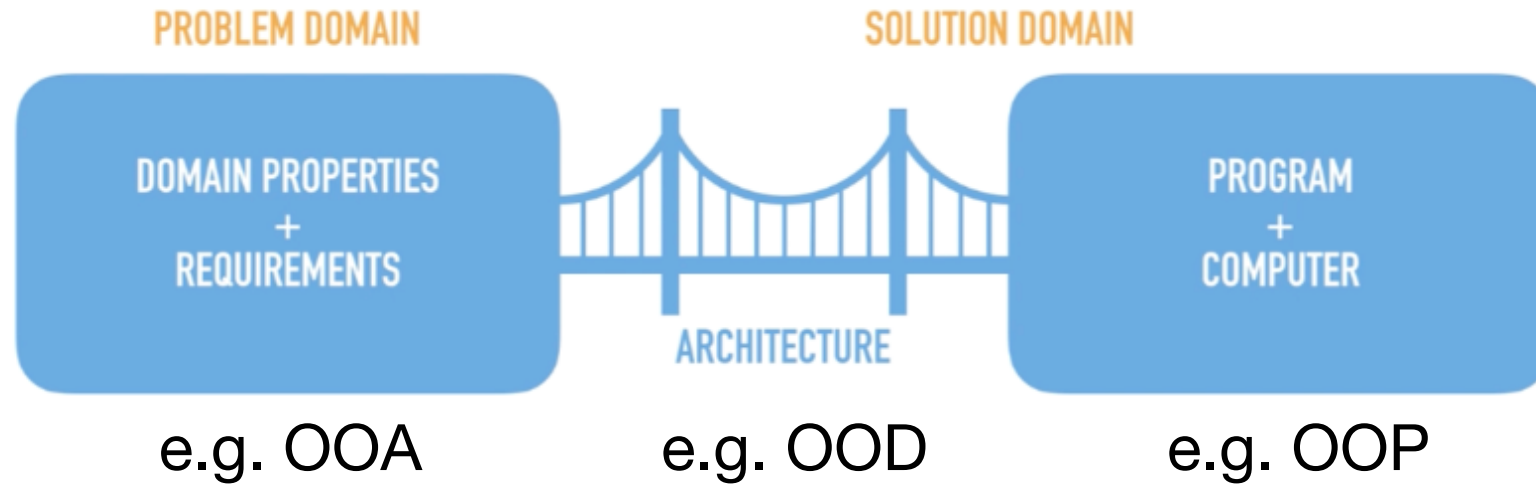


The architecture stack

The Software Design & Architecture Stack



Freedom of choice?



- **Situation A:** You create a new architecture from scratch with freedom in choice of components and style
- **Situation B:** You have to integrate elements from an existing system landscape
- **Situation C:** You develop a system/component according to an existing, domain specific software architecture

Final remarks

- **Summary:**
 - You know why it is good to have an architecture.
 - You can define the term architecture.
 - You know different architectural styles.
 - You can explain common architectures and visualize them.
- **Next week:** Designing classes
- **Please read:** UML notation summary, UML cheatsheet