Software Engineering Essentials

# Architectural Patterns

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

# Learning goals

1) Understand the concept of architectural patterns and how to apply them during system design
2) Apply the Layer pattern
3) Understand the Client-Dispatcher-Server pattern
4) Understand the MVC architectural style

# Architecture is an Art

Inventing a novel architecture is a highly creative act

Requires:

- Knowledge of existing work
- Experience

Sydney Opera House Sails" (CC-BY-SA 3.0) by Enoch Lau

Washington Monument

# Architectural Styles for Buildings

Ranch Style

T-Ranch Style

Raised Ranch Style

- Style defines main components/layout
- Architect picks an architectural style based on customer requirements
- Architect changes details according to customer requirements

→ Enables reuse of established engineering knowledge

We apply the same approach to software architecture

# Terminology: Architectural Pattern vs. Architectural Style

Buschmann and his colleagues see the following differences between these terms

- Architectural styles only describe the overall structural frameworks for applications
- Architectural patterns define the basic structure of an application
- Architectural styles are independent from each other, a pattern depends on smaller patterns
- Patterns are more problem oriented than architectural styles

We use the terms interchangeably

- **Architectural Style = Architectural Pattern**: A pattern for a subsystem decomposition
- **Software Architecture: Instance of an architectural style** (architectural pattern)

# Elements of Software Architecture

## **Components** (Subsystems)

- Computational units with specified interfaces
- Examples: filters, databases, layers, objects

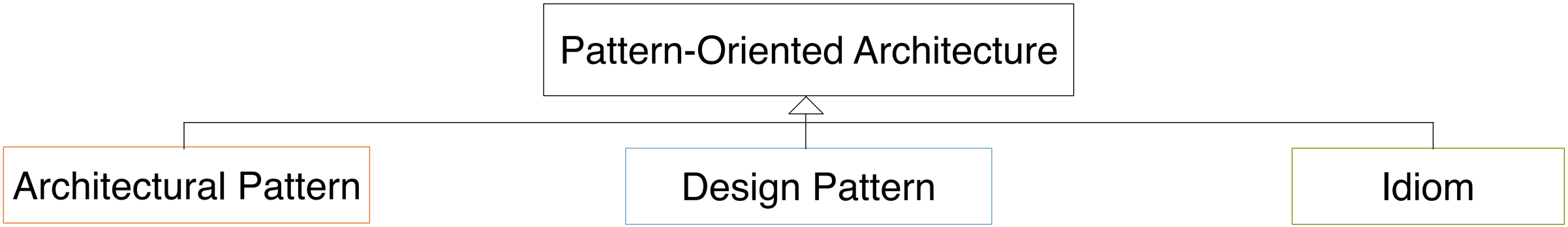
## **Connectors** (Communication)

- Interaction between the components (subsystems)
- Protocols that define how the components interact
- Examples: method calls, pipes, event broadcasts, shared data

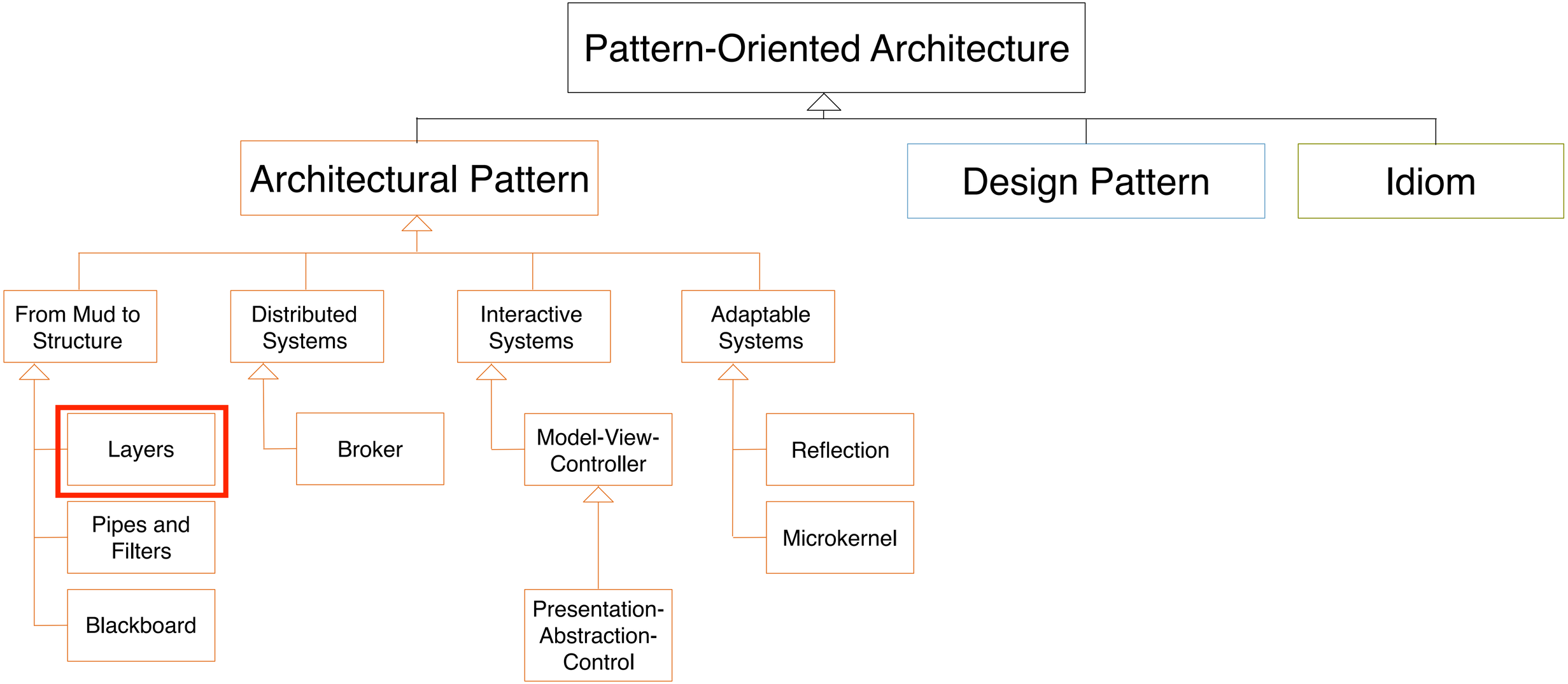
M. Shaw, D. Garlan: *Software Architecture*, Prentice Hall, 1996

# Pattern-Oriented Architecture

"Gang of Five" Taxonomy

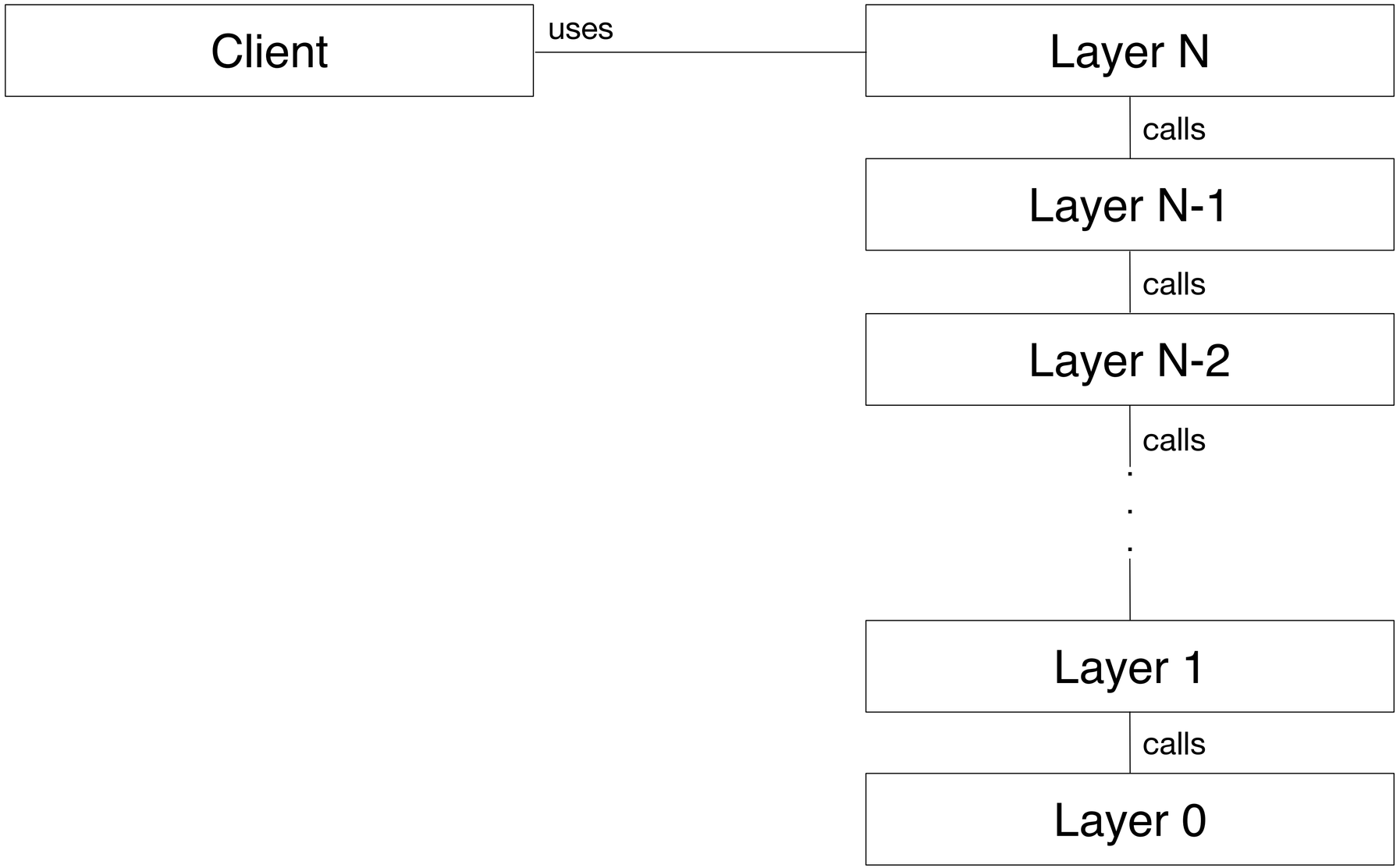


# Pattern-Oriented Architecture



# Layer Pattern

Example: 7-Layer of the OSI Model



# The Layers of the T.H.E. System

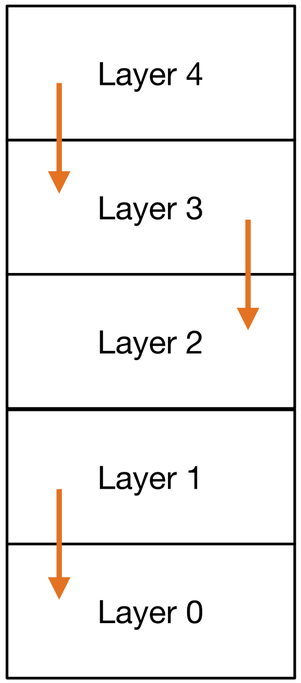
“An operating systems is a hierarchy of layers, each layer using services offered by the lower layers”

| |
|---|
| Layer 4: User Programs |
| Layer 3: I/O Device Manager |
| Layer 2: Communication between OS and Console |
| Layer 1: Pager |
| Layer 0: Scheduler |

# Closed (opaque) vs. Open (transparent) Architecture

## Closed Architecture

- Each layer can only call operations from the layer below

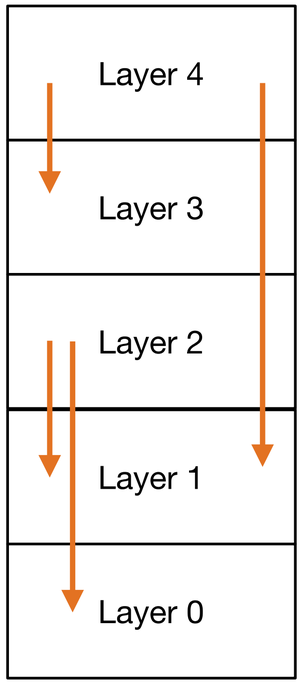



Design Goals:

- Flexibility
- Testability
- Maintainability

## Open Architecture

- Each layer can call operations from any layer




Design Goals:

- Runtime efficiency

# Pros and Cons of the Layer Pattern

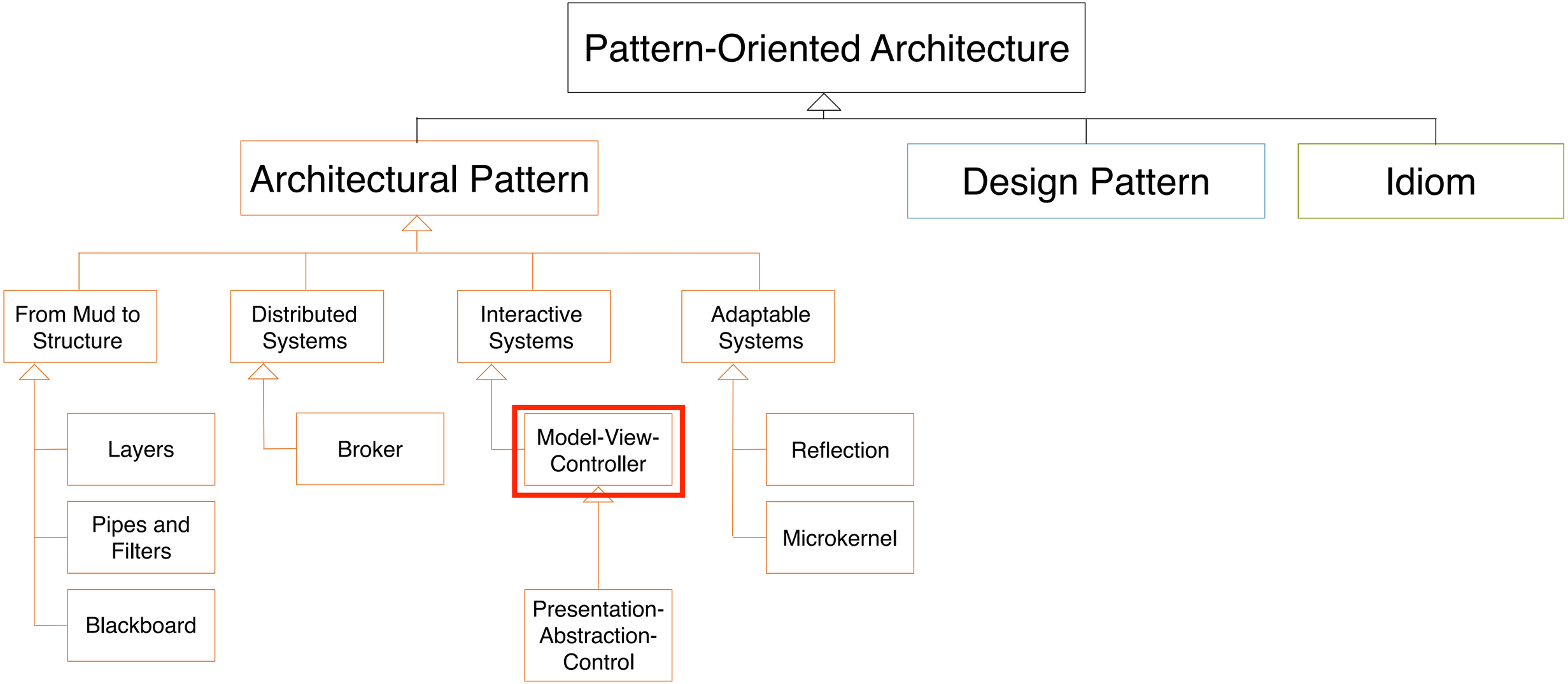
**Pros (Benefits)**

+ Reusability of Layers, especially in a closed architecture
+ Support for standardization
+ Low coupling
+ Improves testability

**Cons (Liabilities)**

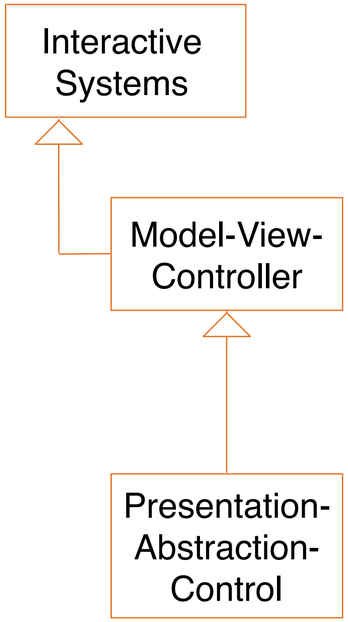
- A local change in a lower layer may require rework in higher layers
- Lower efficiency

# Pattern-Oriented Architecture



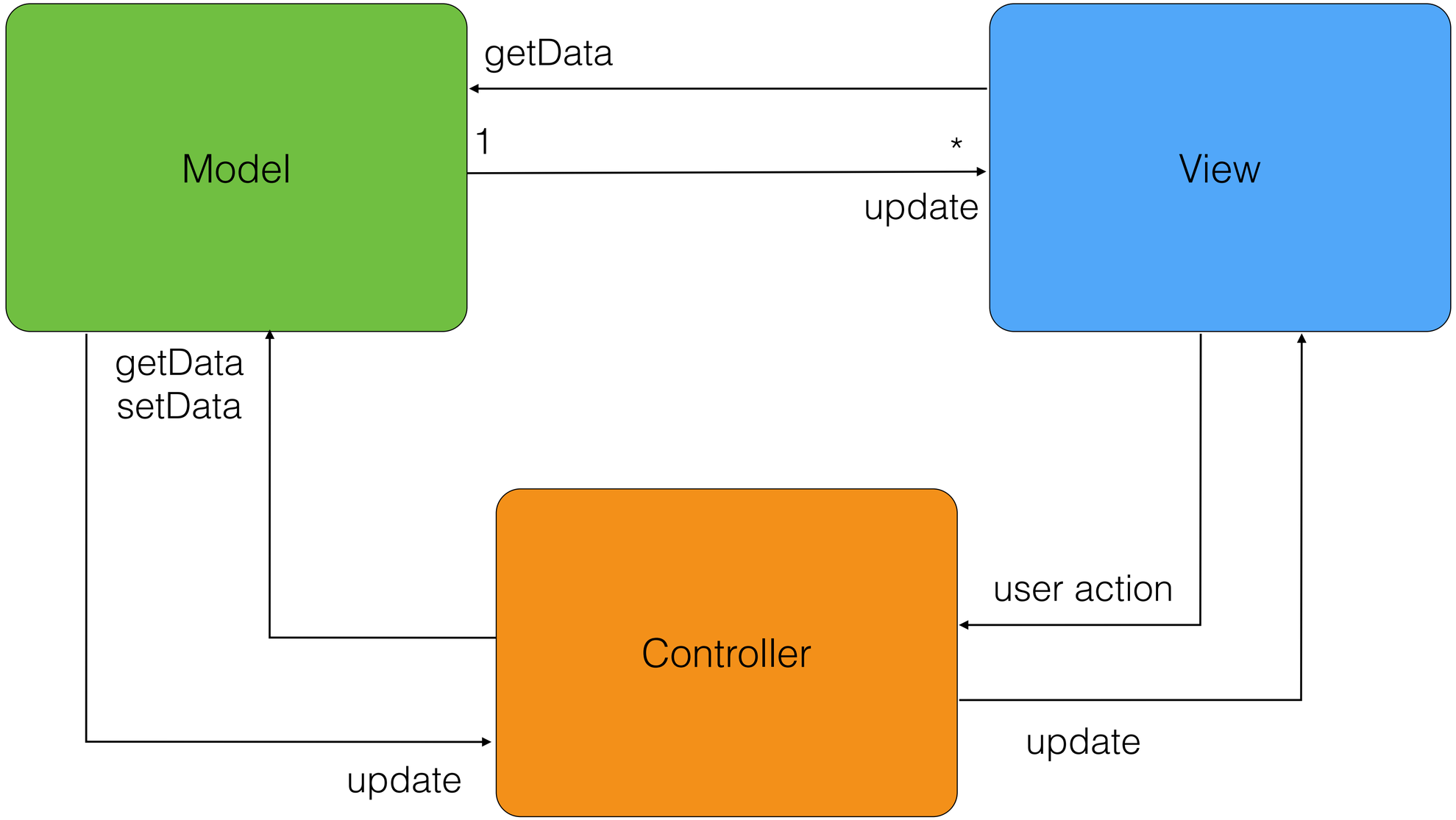
# Model view controller

The MVC architectural style decouples data access and data presentation.

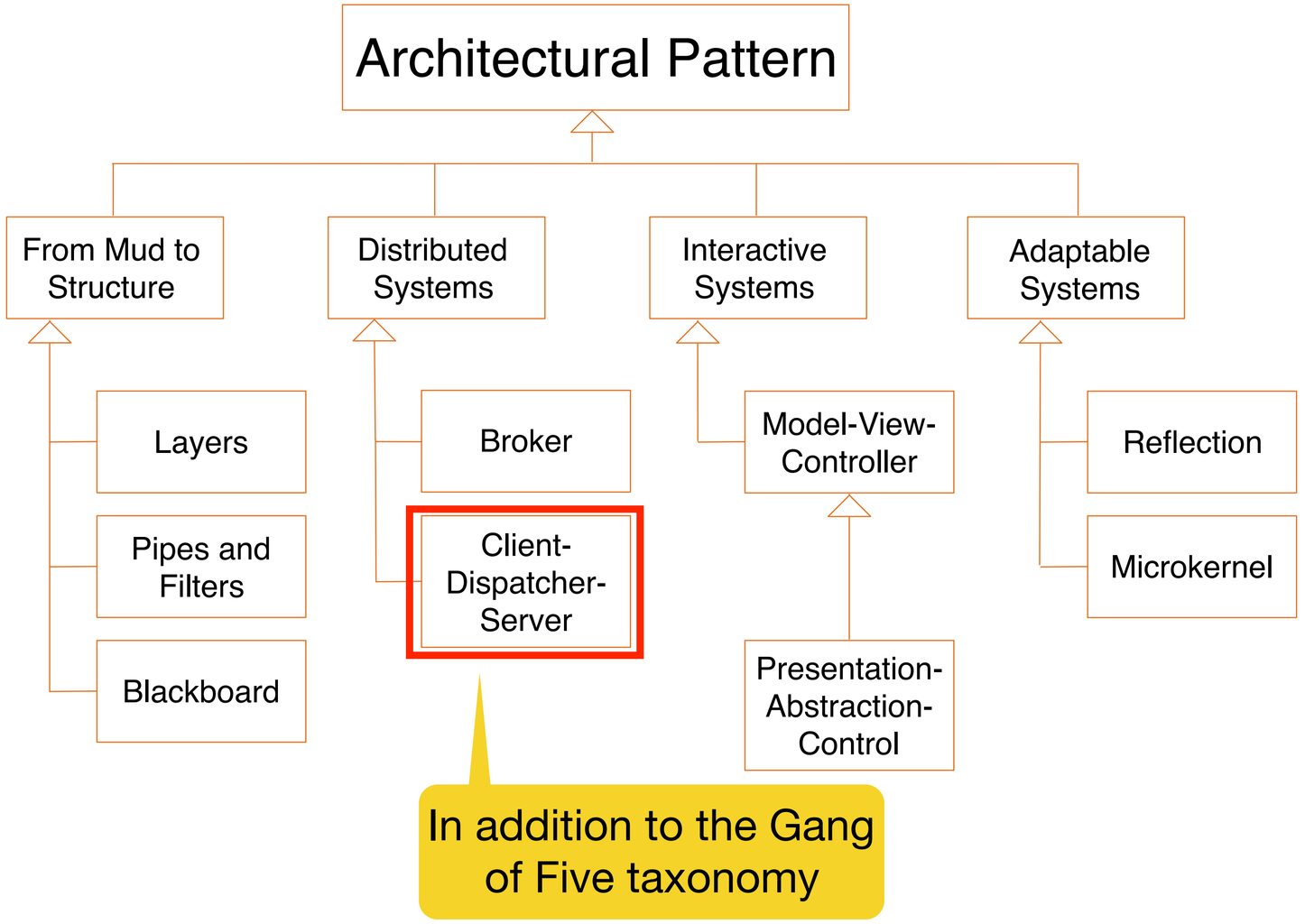



| | |
|---|---|
| **Model** | • Data access subsystem<br>• Includes objects that represent the application domain knowledge |
| **View** | • Data presentation subsystem<br>• Includes objects that are a visual representation of the model |
| **Controller** | • Mediates between view and model<br>• Includes objects that link model objects with their views |

# Model view controller diagram



# Pattern-Oriented Architecture



# Challenges in Distributed Systems

## Connectivity

- Systems are distributed and have to find and communicate with each other
- Pattern: Client-Dispatcher-Server

## Heterogeneity

Not covered in this course

- Systems use different languages, platforms and protocols
- Pattern: Broker, Microservices

## Security

Not covered in this course

- Access Control, Authentication and Encryption

# Distributed Systems must communicate with each other

Designing distributed systems requires a communication mechanism

- Solution A — Client-Server
  - The components implement the communication mechanism themselves
  - Disadvantage: Clients need to know the location of the server, tight coupling, bad design
- Solution B — Client-Dispatcher-Server
  - The communication mechanism is placed in a name server, thereby decoupling clients and servers
  - Advantage: The name server provides location transparency for the server and can set up the communication path between client and server.

Software Engineering Essentials

# Architectural Patterns

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics