OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

FACULTY OF
COMPUTER SCIENCE

SwarmLab

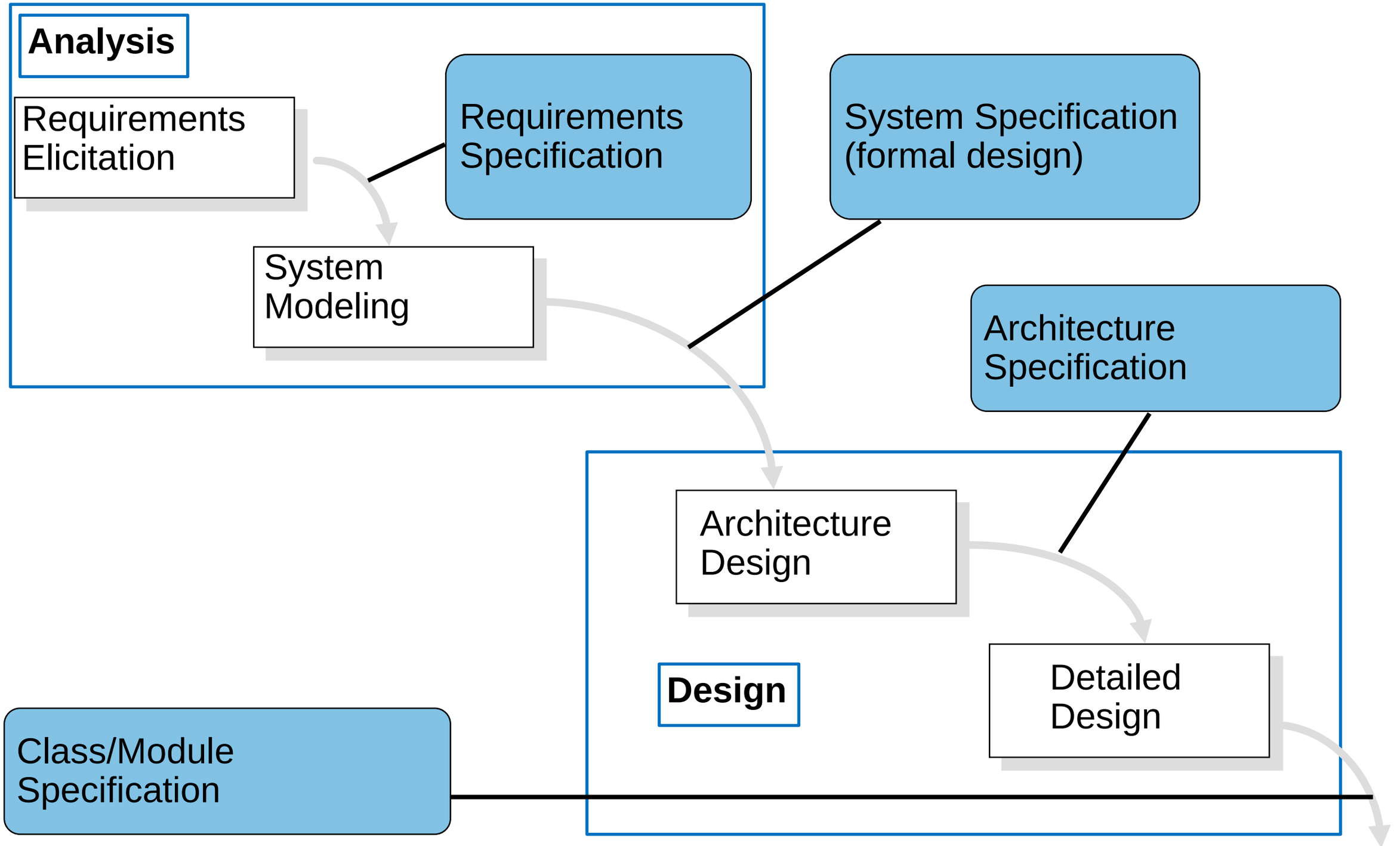
Introduction to Software Engineering for Engineers L-06: Software Architectures
Part 1: Motivation

Dr.-Ing. Christoph Steup

Content

- Motivation
- Foundations of Software Architecture
- Views in Architectures
- Architecture styles and patterns
- Common/Popular Architectures

Design Phases



Why an Software Architecture?

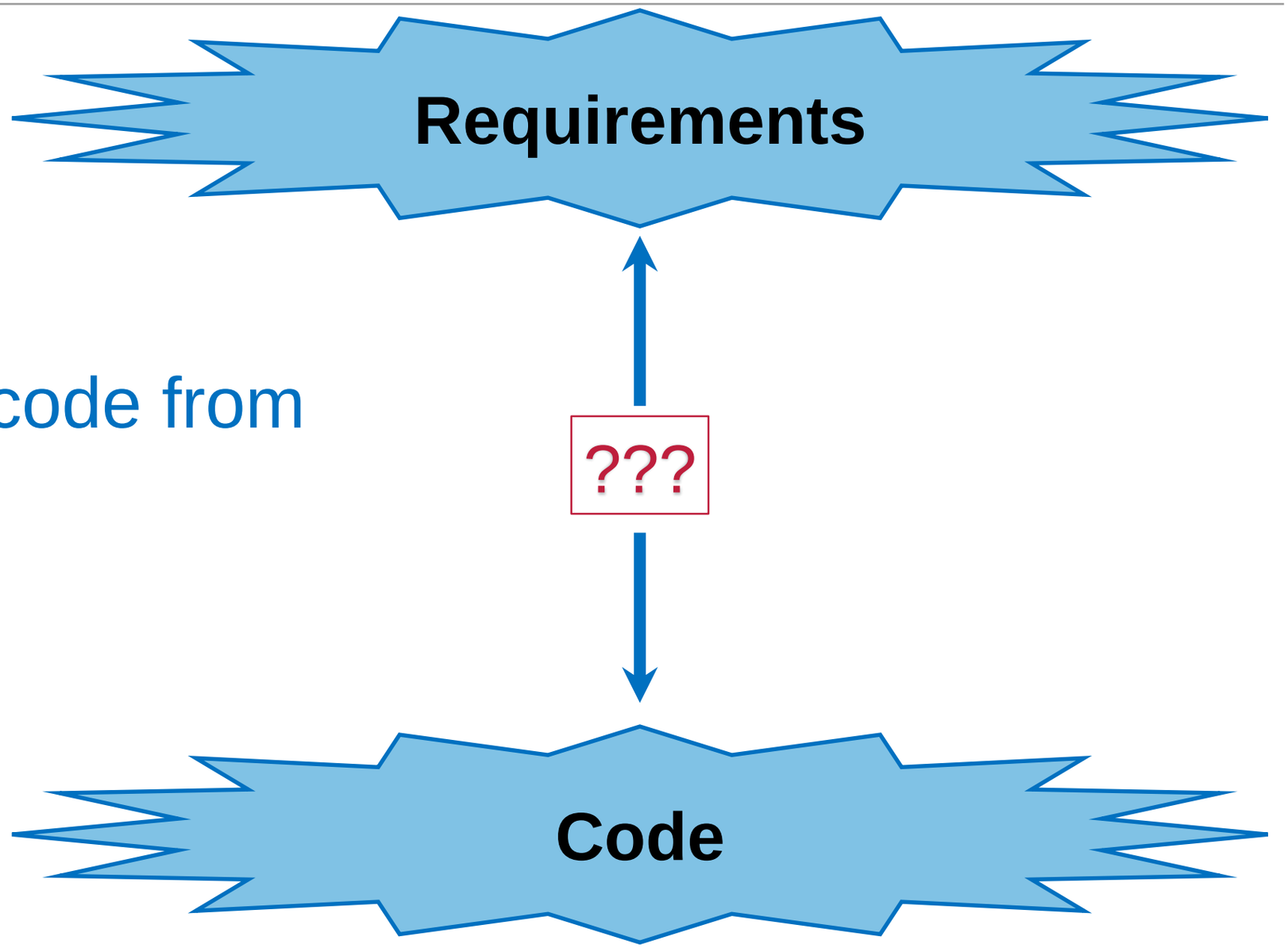
Structured procedure in software development

Reliable foundation for software (“statics”)

Well-defined points for extension

No uncontrolled “balcony additions”

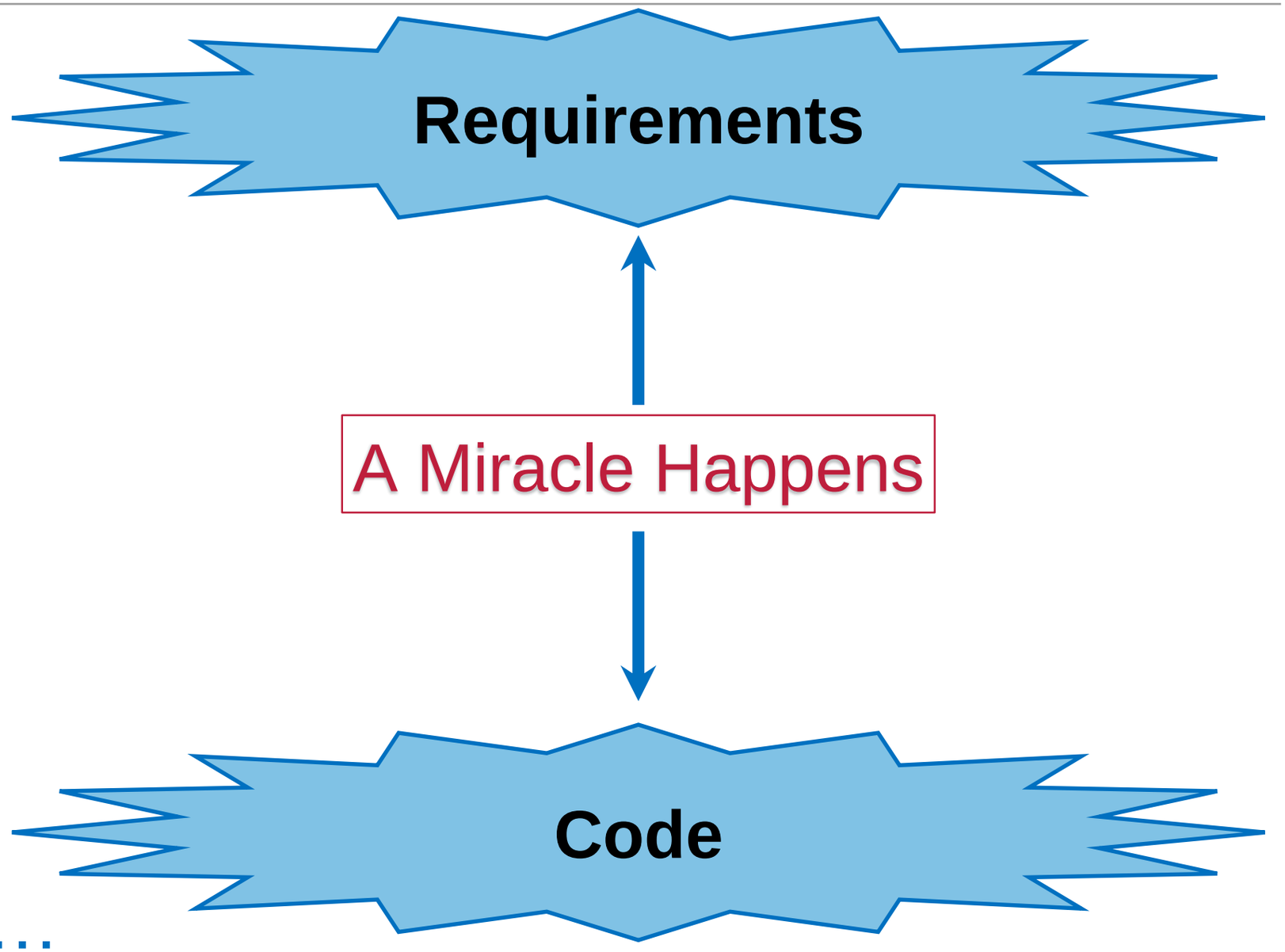
Fundamental Problem in Software Development



- How to obtain code from requirements?

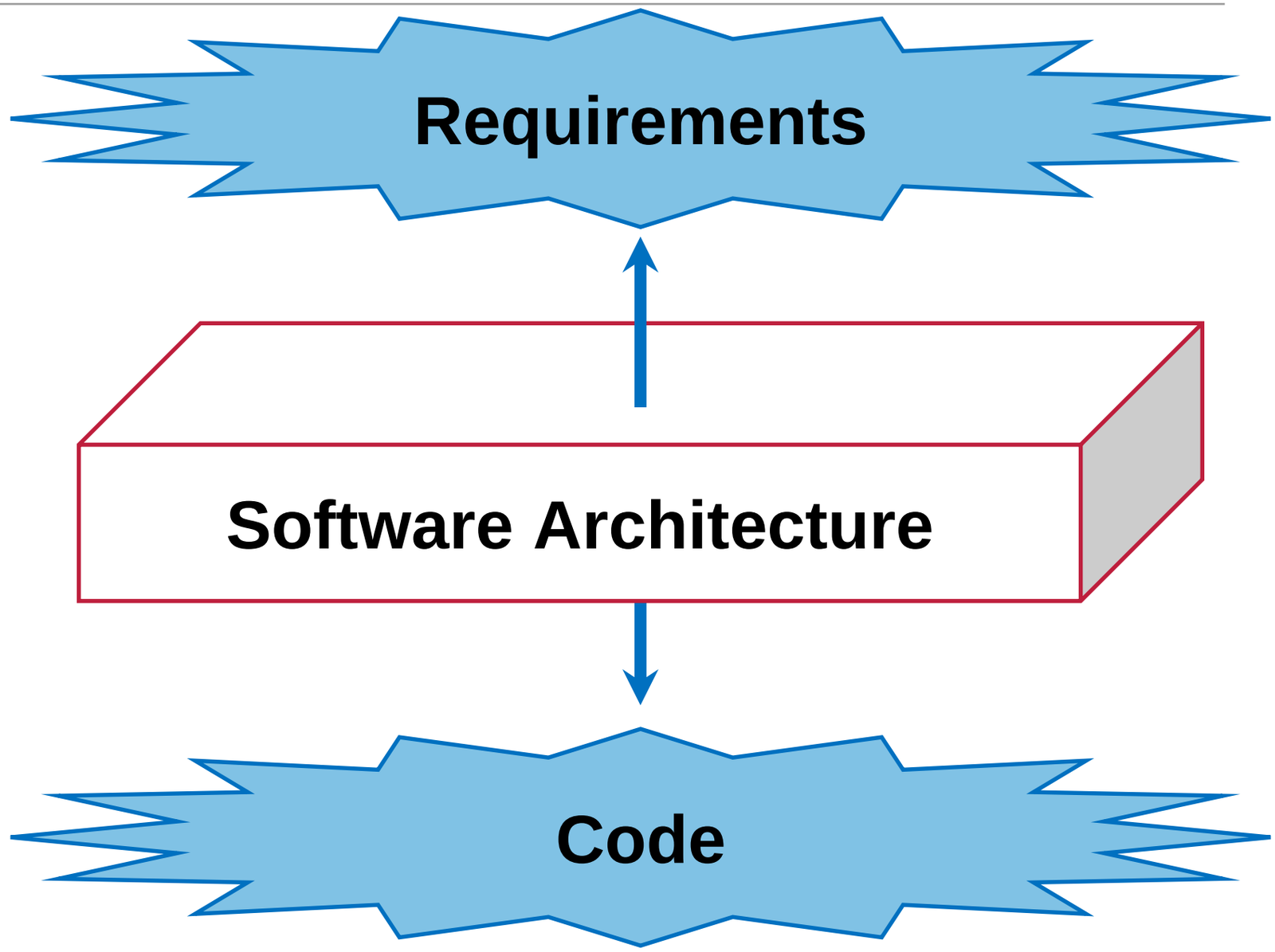
Traditional Answer

- Ad hoc
- Experiences
- Unpredictable
- cost-intensive…



Role of Software Architecture

- Composition of large, complex components
- Abstraction
- Reuse



[Sommerville 04]

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

FACULTY OF COMPUTER SCIENCE

SwarmLab

Introduction to Software Engineering for Engineers L-06: Software Architectures Part 2: Foundations & Views

Dr.-Ing. Christoph Steup

Software Architecture: An own Definition

Software Architecture (SA) has two aspects:

- ➢ SA constitutes the structural type of a system, reduced to its essence (abstraction).
- ➢ SA of a program or system describes the structure of a system.

SA precisely describes the most important structural properties of a system, but at the same time it is rather compact.

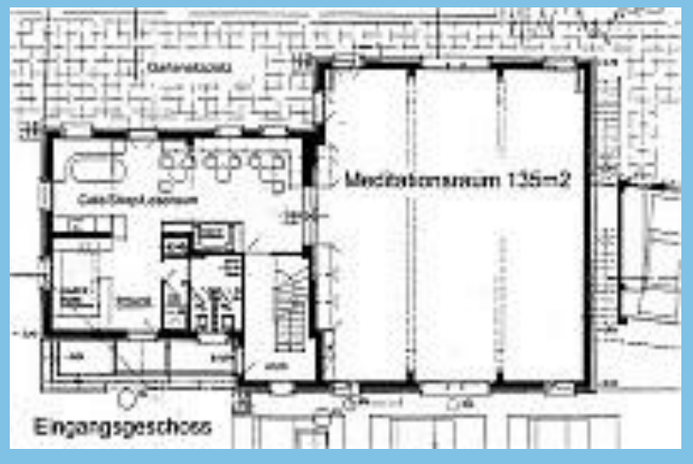
SA contains the properties of a system that can be described and analyzed on the entire system view

Features of a Software Architecture

The description contains

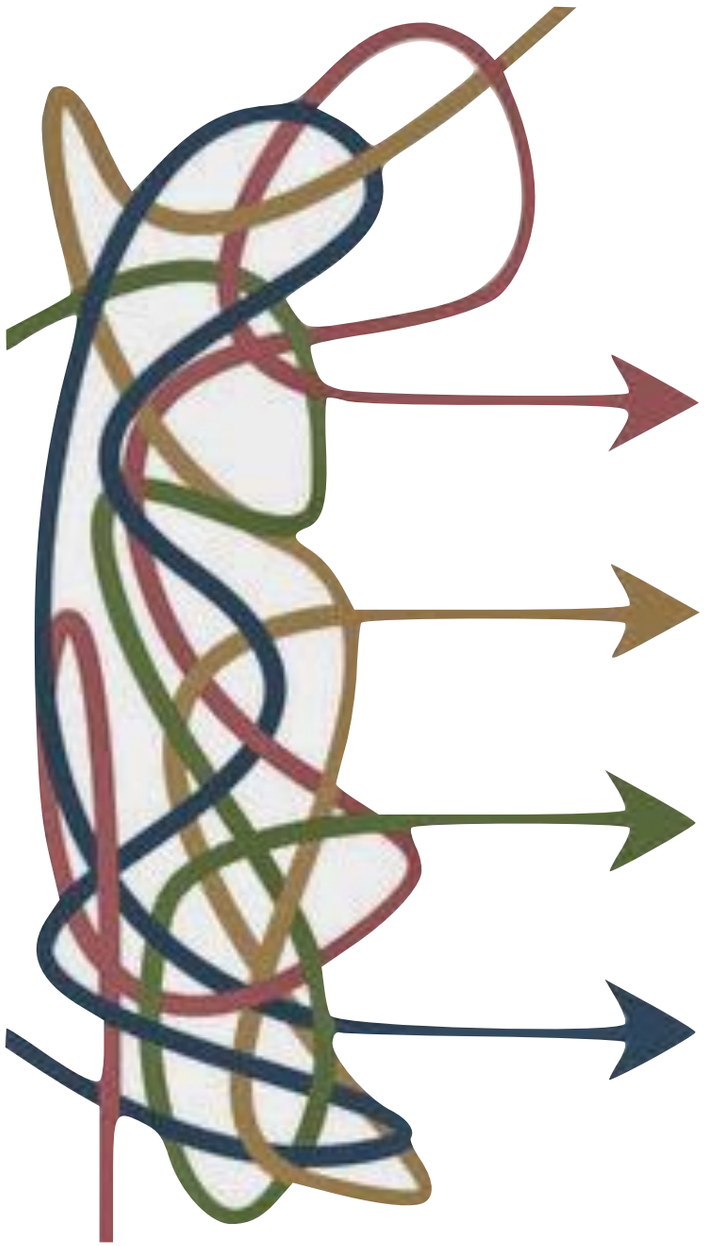
- the (architecture) building blocks (known as components),
- their interfaces
- and relations and interactions among them
- Behavior of components (as far as visible from outside)

For an architecture, different views and levels of abstraction exist, which allow to focus on different aspects.



nach [Bass, 1998]

Goals of Software Architecture



- Efficient development
  - frame for integration
  - foundation for project planning
- Minimizing risk
  - Assessment
  - Determining influence factors
- Communication between stakeholders
- Preservation of knowledge
  - Reuse

[Posch et al., 2011]

Documentation of Software Architecture

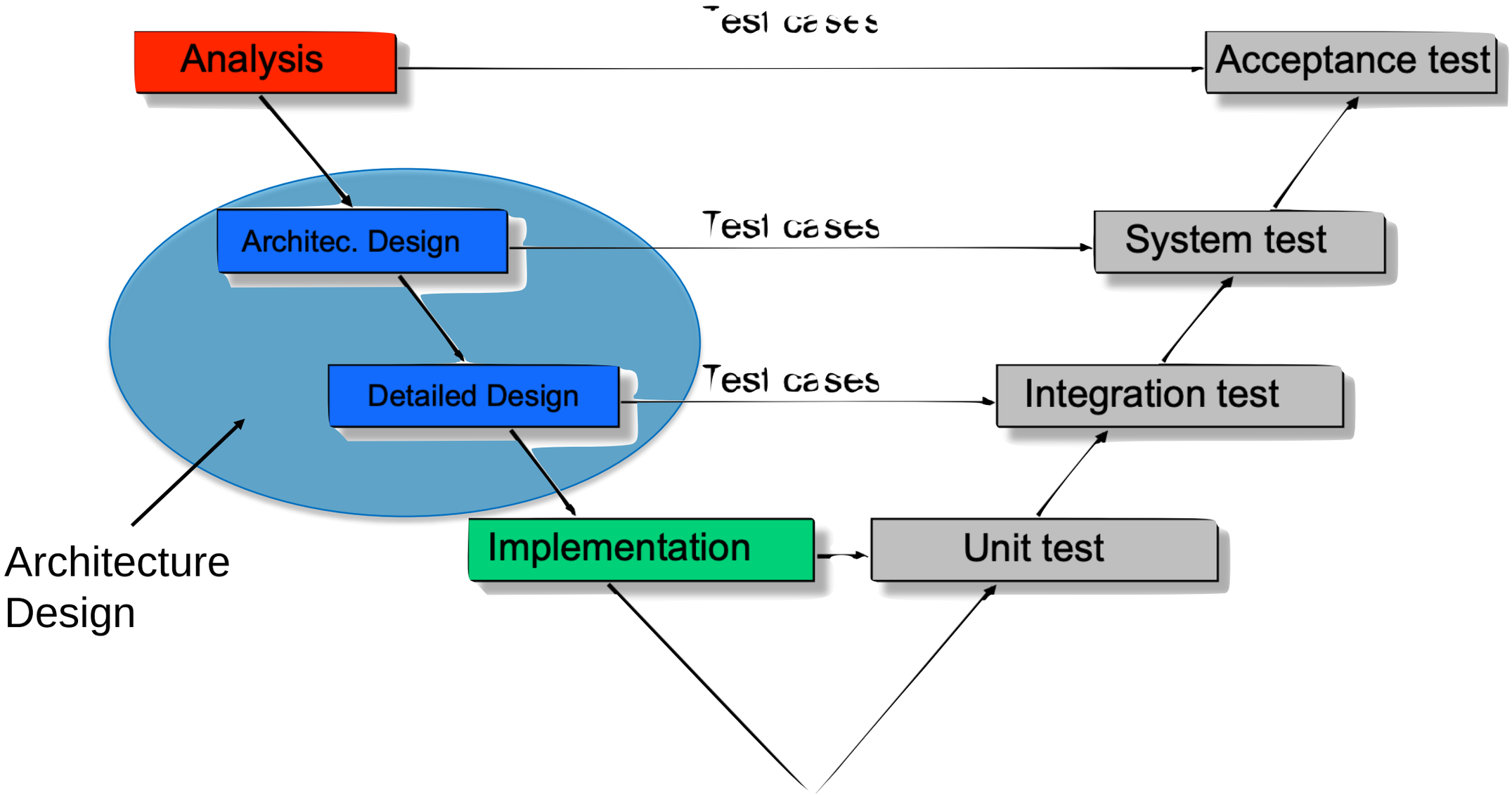
An explicitly modelled software architecture

- ➢ structures the system („divide and conquer“)
- ➢ brings stability of the development
- ➢ improves the evolution (i.e., future development)

As a result, a software architecture determines fundamental rules for project organization, design, implementation, and evolution.

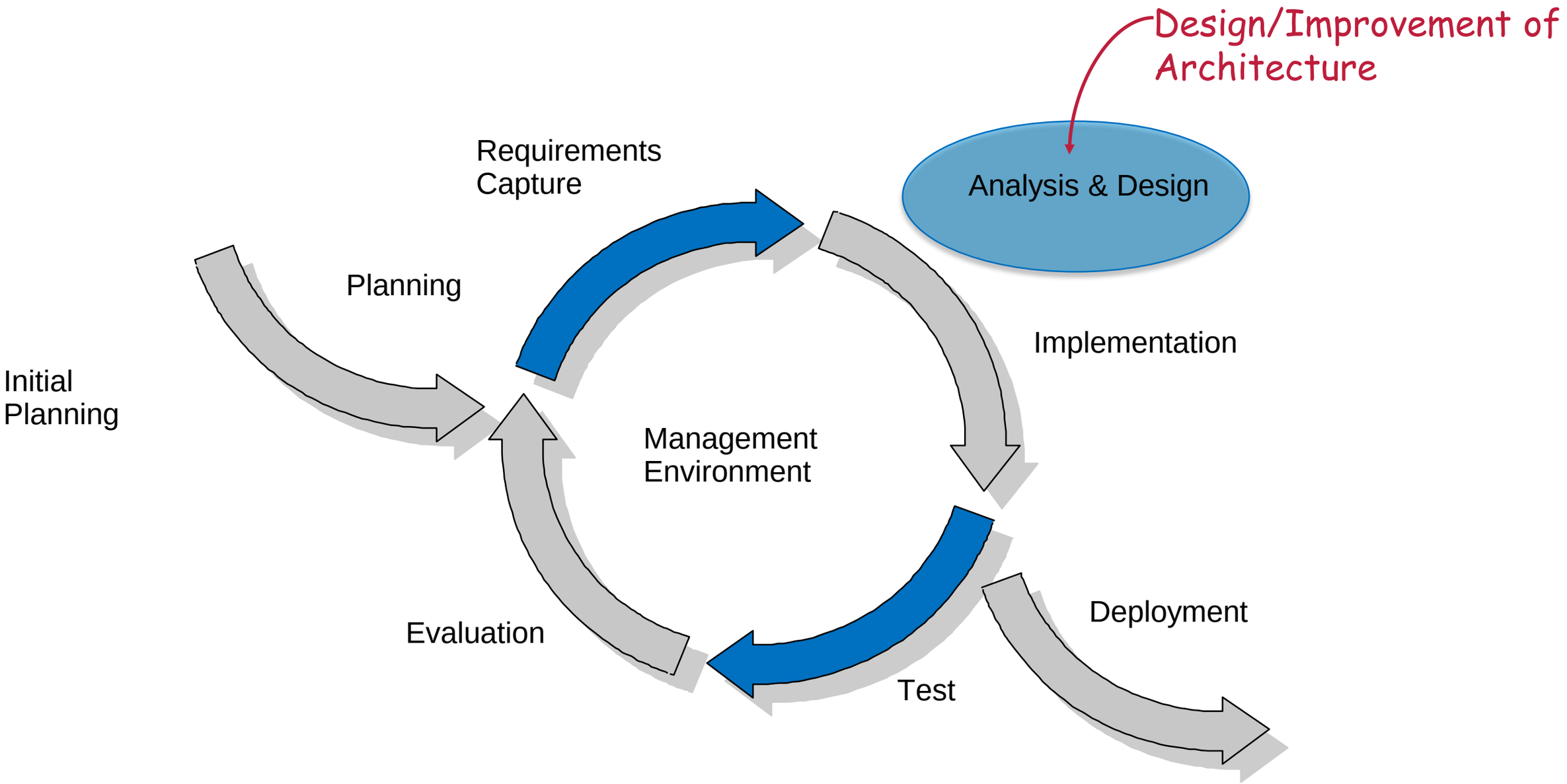
Note: Software architecture considerably influences the (further) process!

Software Architecture in the Development Process (V-Model)



Software architecture influences **all further** development phases!

Software Architecture in the Iterative Model



Influence of Stakeholders on Software Architecture

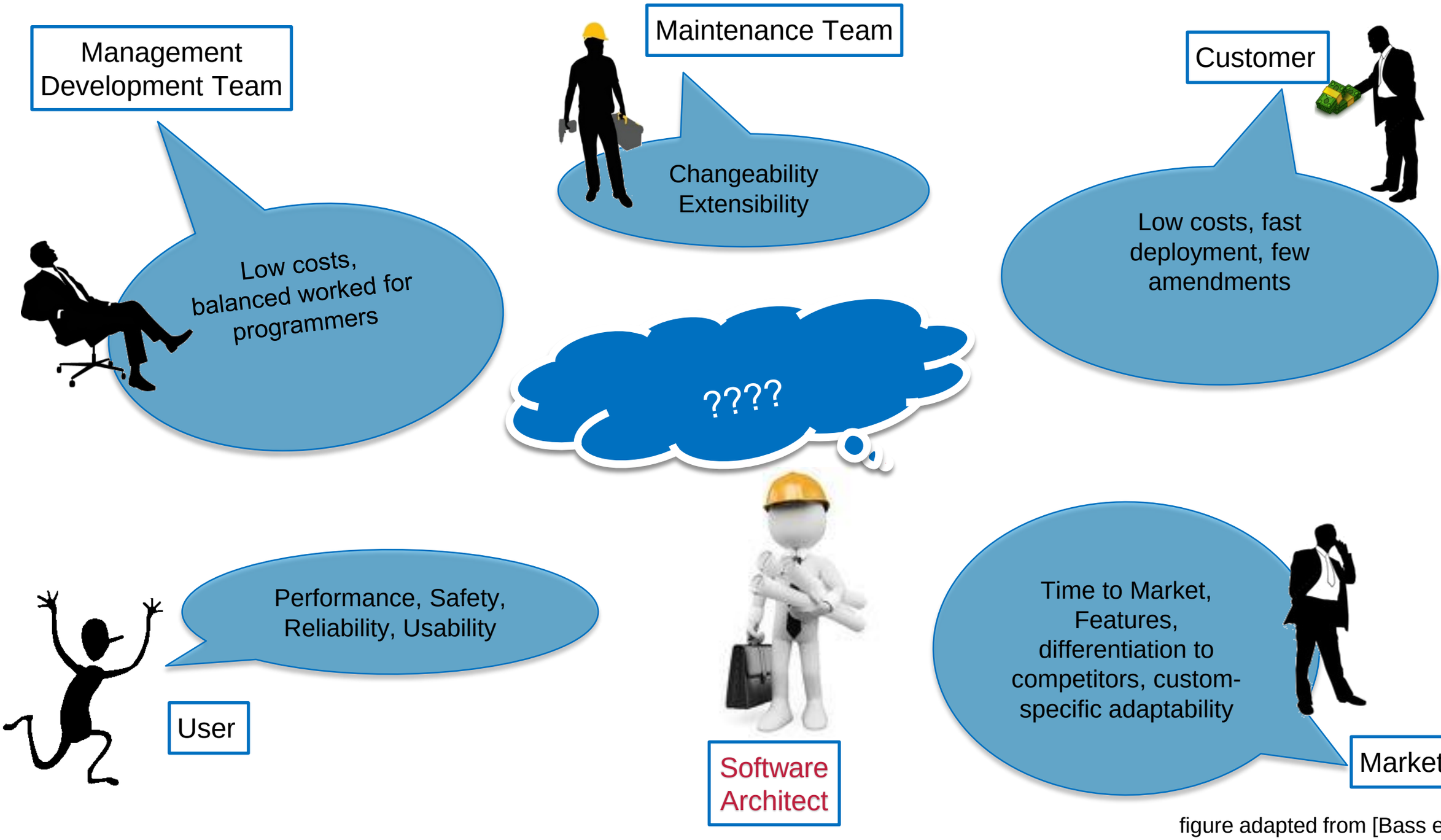


figure adapted from [Bass et al., 03]

Architecture Description

Definition according to IEEE 1471

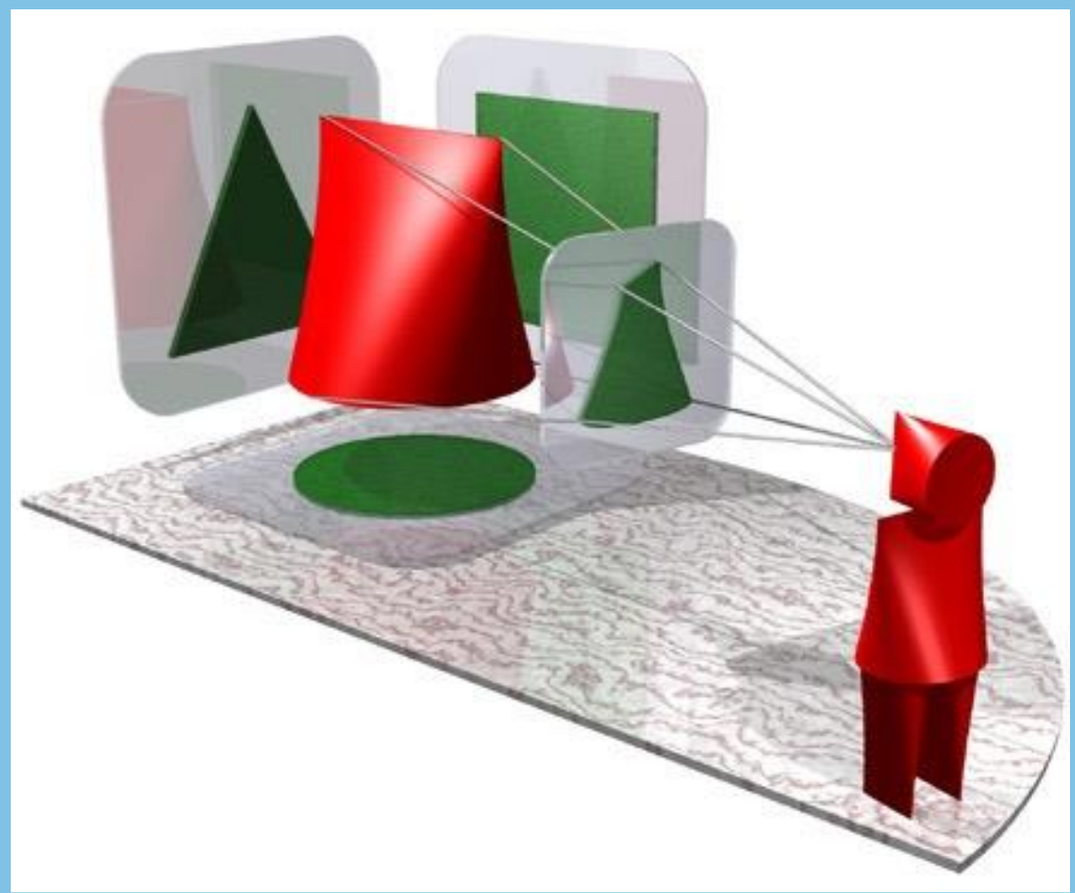
An **architecture description** is a set of models (textual specification or graphical diagrams) that document the software architecture.

Views/Viewpoint

Different stakeholders require different information:

- ➢ appropriate description for different target groups desired
- ➢ focussing on core pieces of an aspect

Representation of a concrete system from a particular viewpoint is called view.

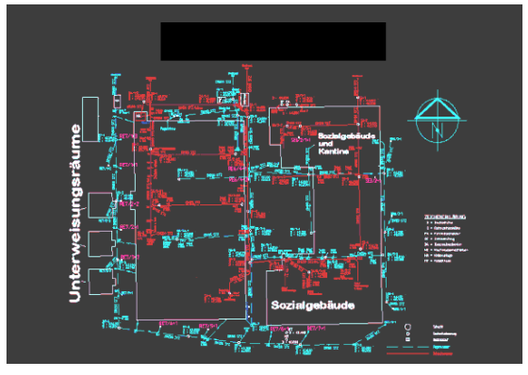


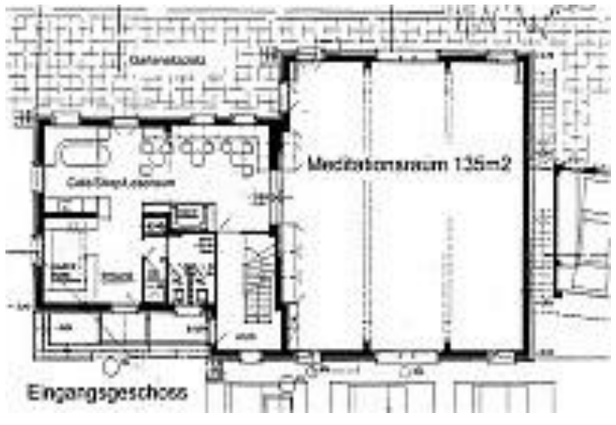
aus [AC98]

Analogy: House Building

A house can be described by several, different plans:

- Ground view
- Drawing
- Location plan
- Electrical connection plan
- Sewer (canalization)





Every plan

- serves a particular purpose
- represents detail of the house
- has a different target group

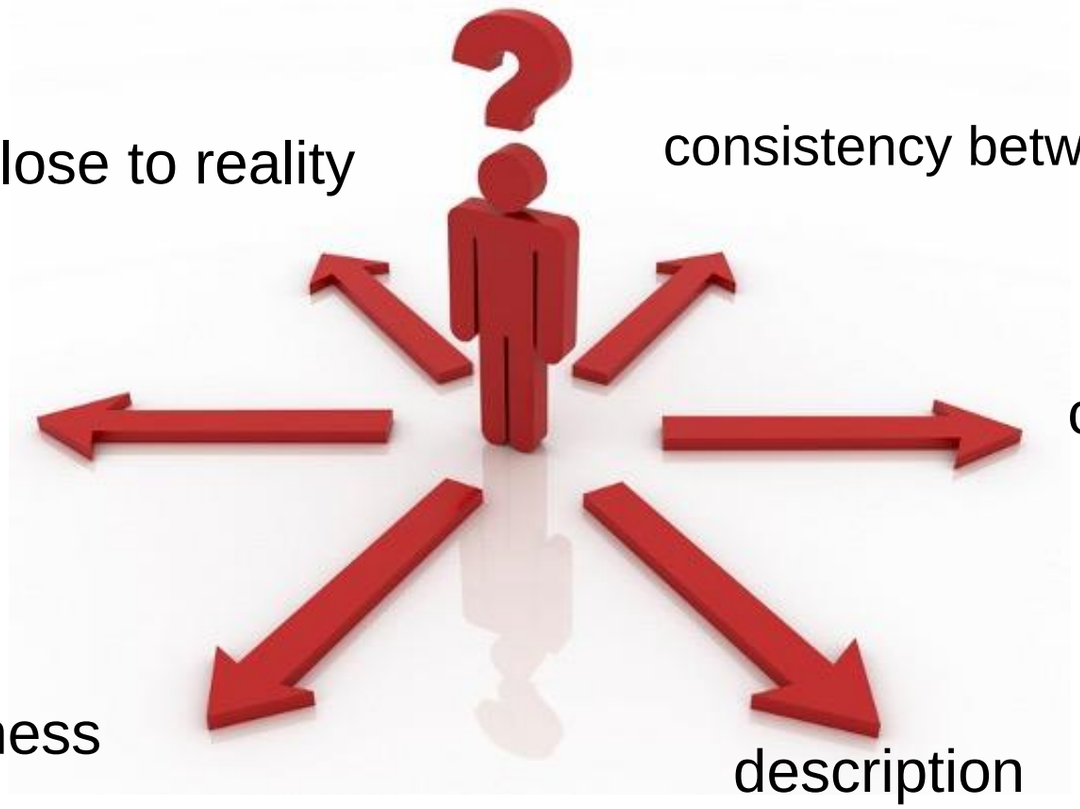
View

A View is a representation of a system, which contains only elements and relations that are relevant for a particular perspektive.

Challenges:

view should be close to reality

consistency between views



possibility for abstraction

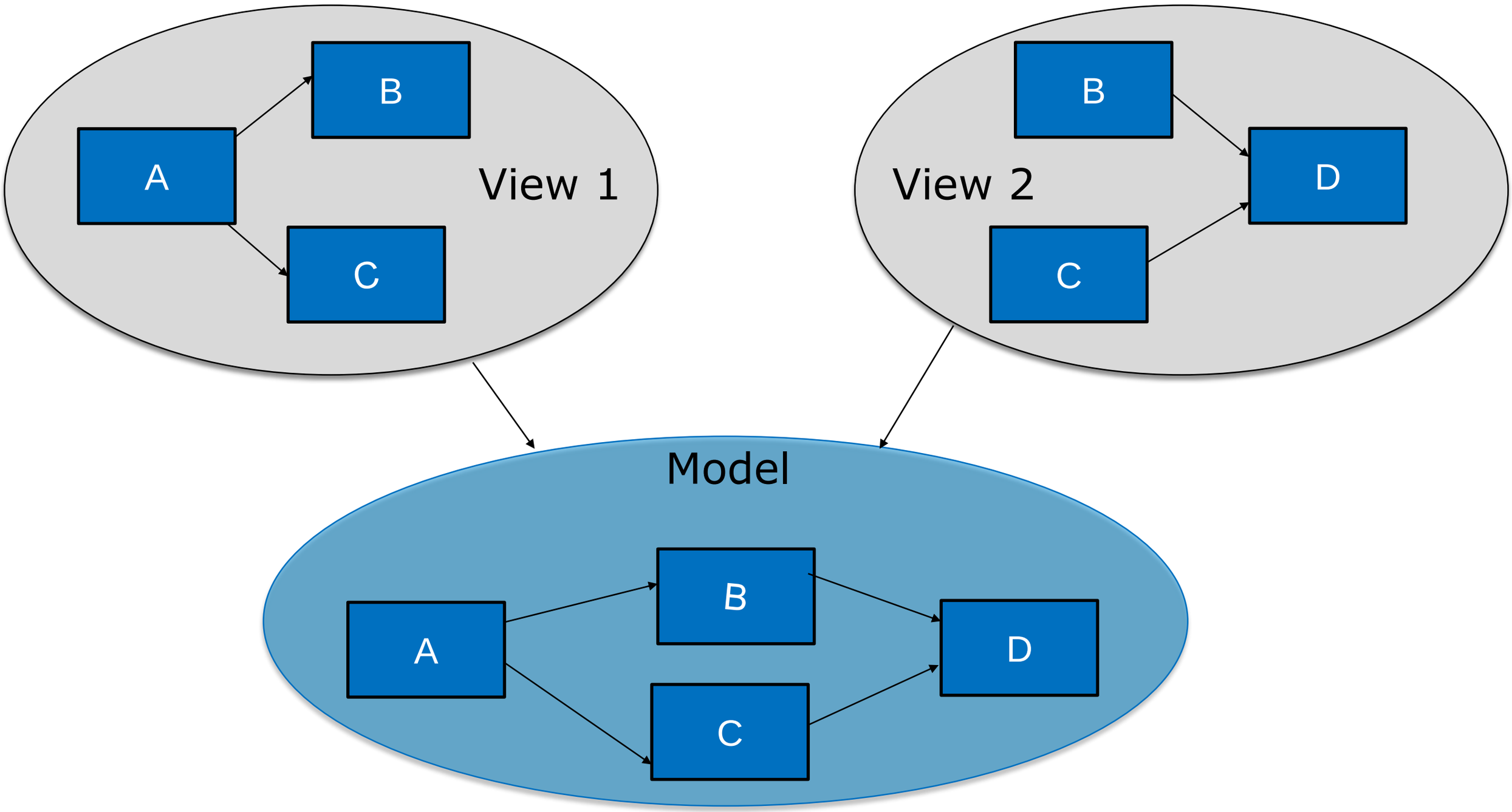
clarity of representation

completeness

description

Views

Example



Fundamental Views for Architectures

Static structural model, representing the fundamental System components and their interfaces.

Dynamic process model, representing the process behavior in a system.

Deployment model, representing the distribution of resources (processors, network connections etc.).

OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

FACULTY OF
COMPUTER SCIENCE

SwarmLab

Introduction to Software Engineering for Engineers L-06: Software Architectures
Part 3: Component & Deployment Diagrams

Dr.-Ing. Christoph Steup

Views in UML

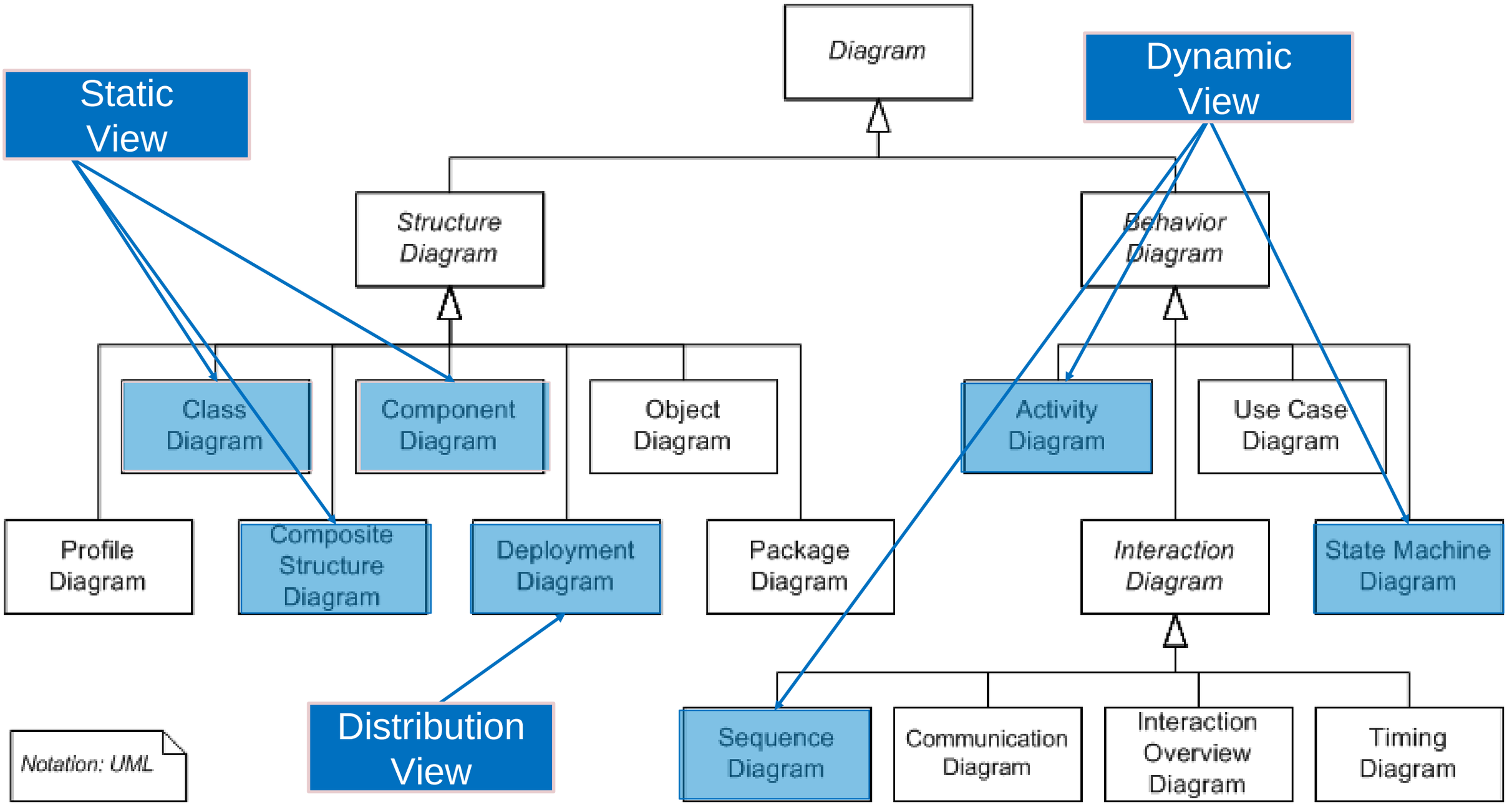
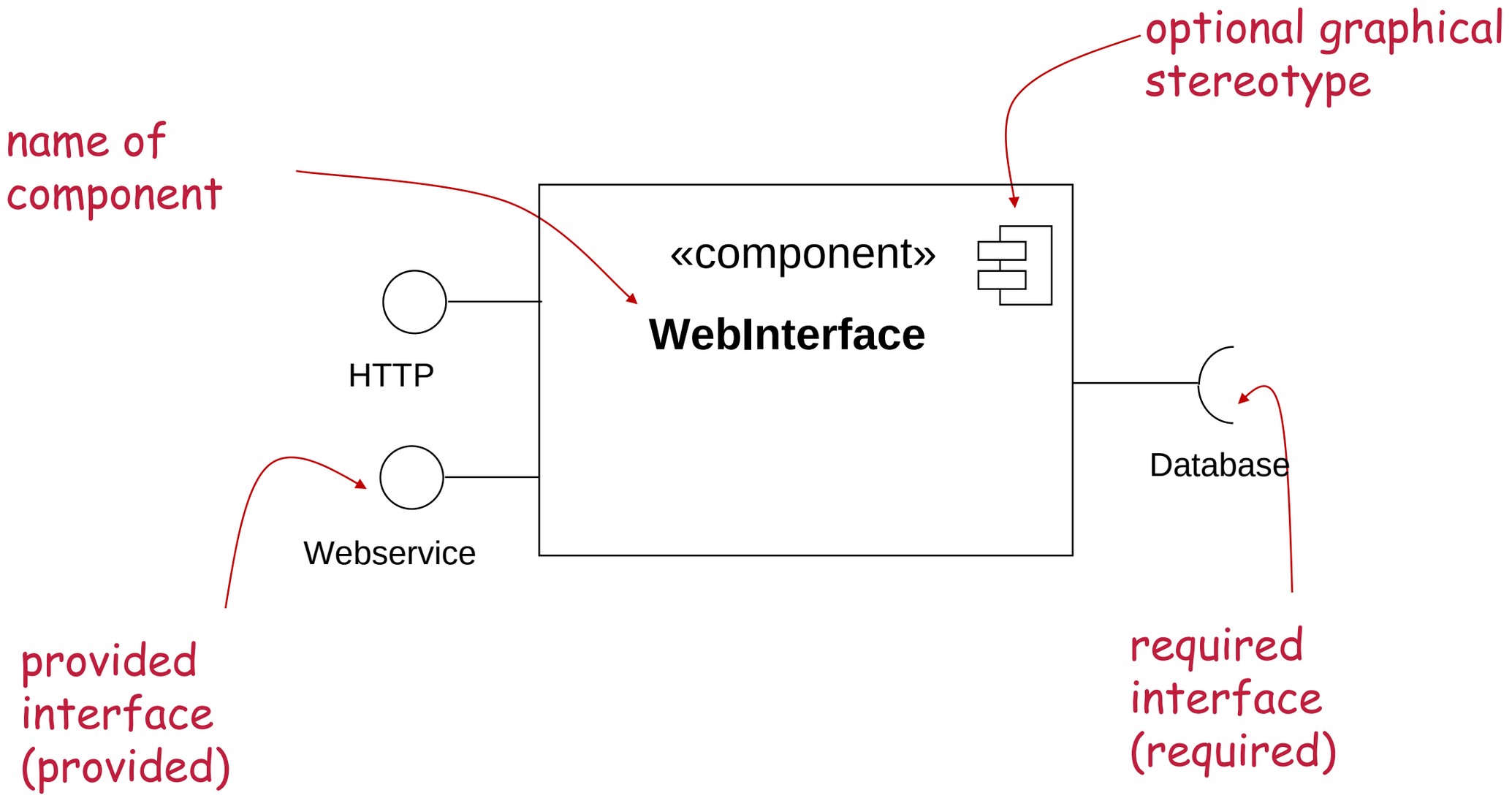


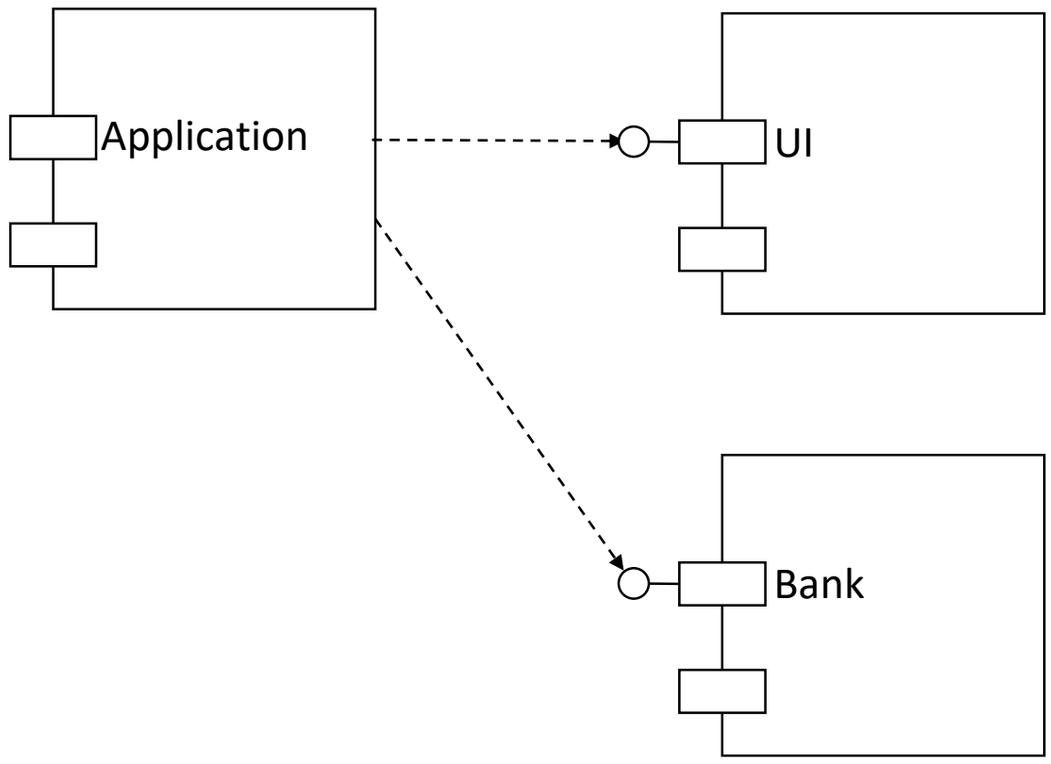
Diagram
Static View
Dynamic View
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Activity Diagram
Use Case Diagram
Profile Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Interaction Diagram
State Machine Diagram
Distribution View
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram
Notation: UML

Components


optional graphical stereotype
name of component
«component»
WebInterface
HTTP
Webservice
Database
provided interface (provided)
required interface (required)

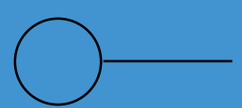
UML Component Diagrams

The component diagram represents the (logical) components of the system together with their interfaces (ports).



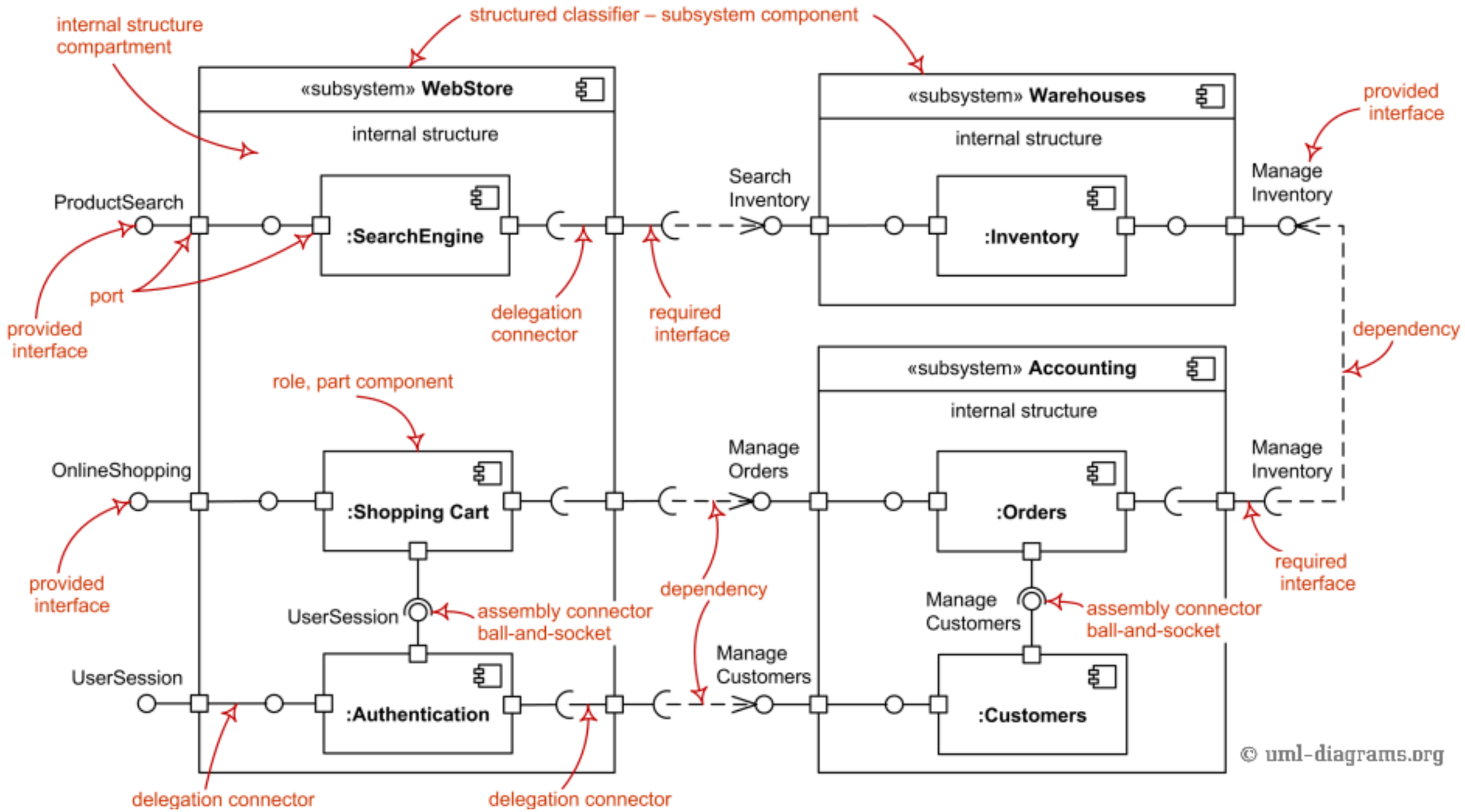
UI = User Interface
GUI = Graphical User Interface

Provided & Required Interfaces

Components exhibit two different interfaces:		
Provided interfaces provide information about how to use the component.		In Java explicitly ➢ **`implements`** as keyword
Required interfaces specify which functionalities external components have to provide.		In Java implicitly ➢ defined by the interfaces used

Component Diagrams

Example

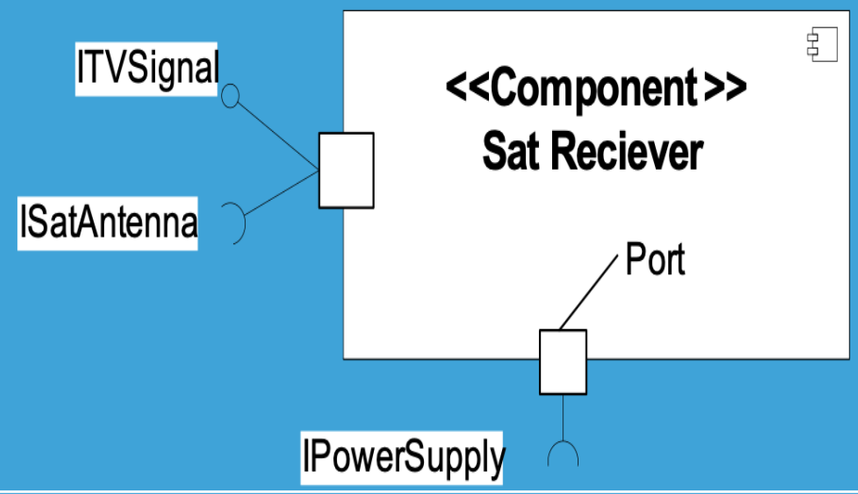


External Interfaces – Ports

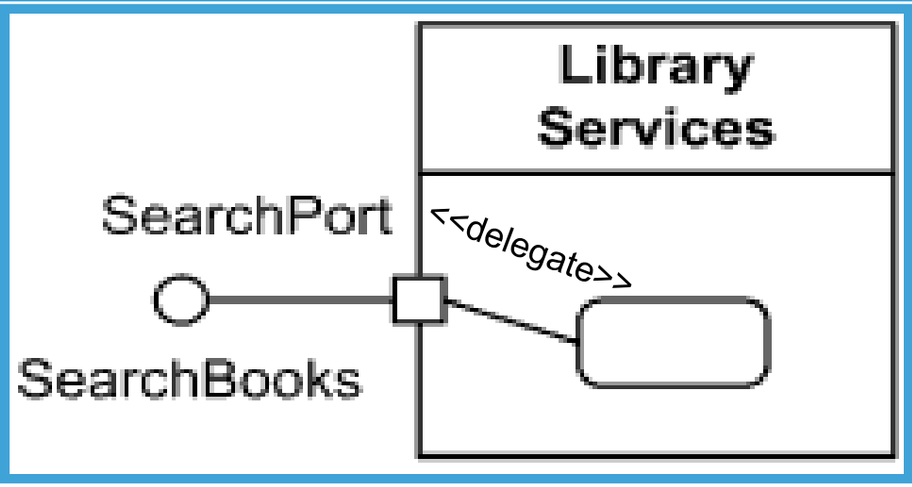
Ports are used to model points of communication.

They encompass provided and required interfaces.

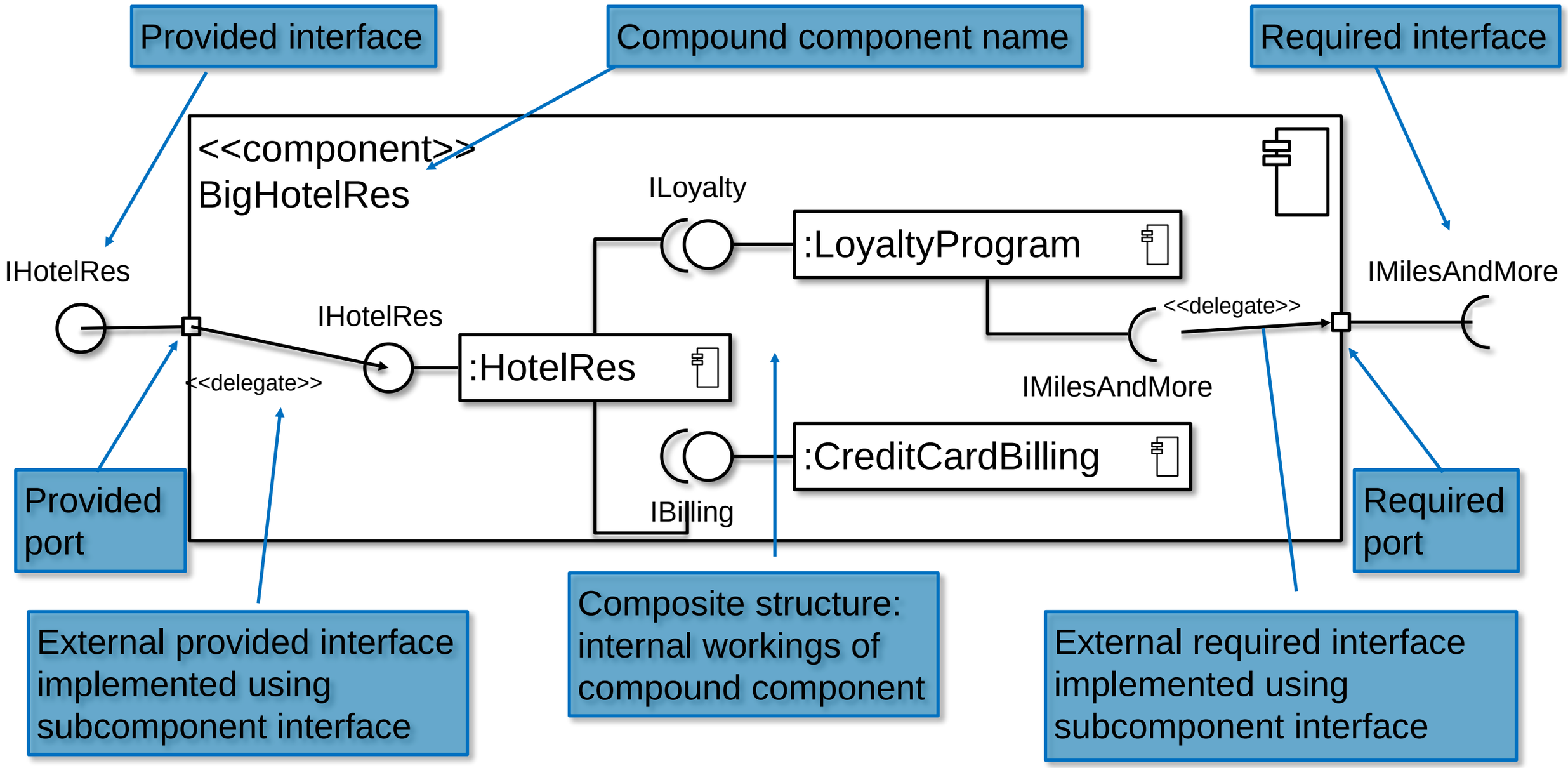
All communication is managed over the port and not directly between components.



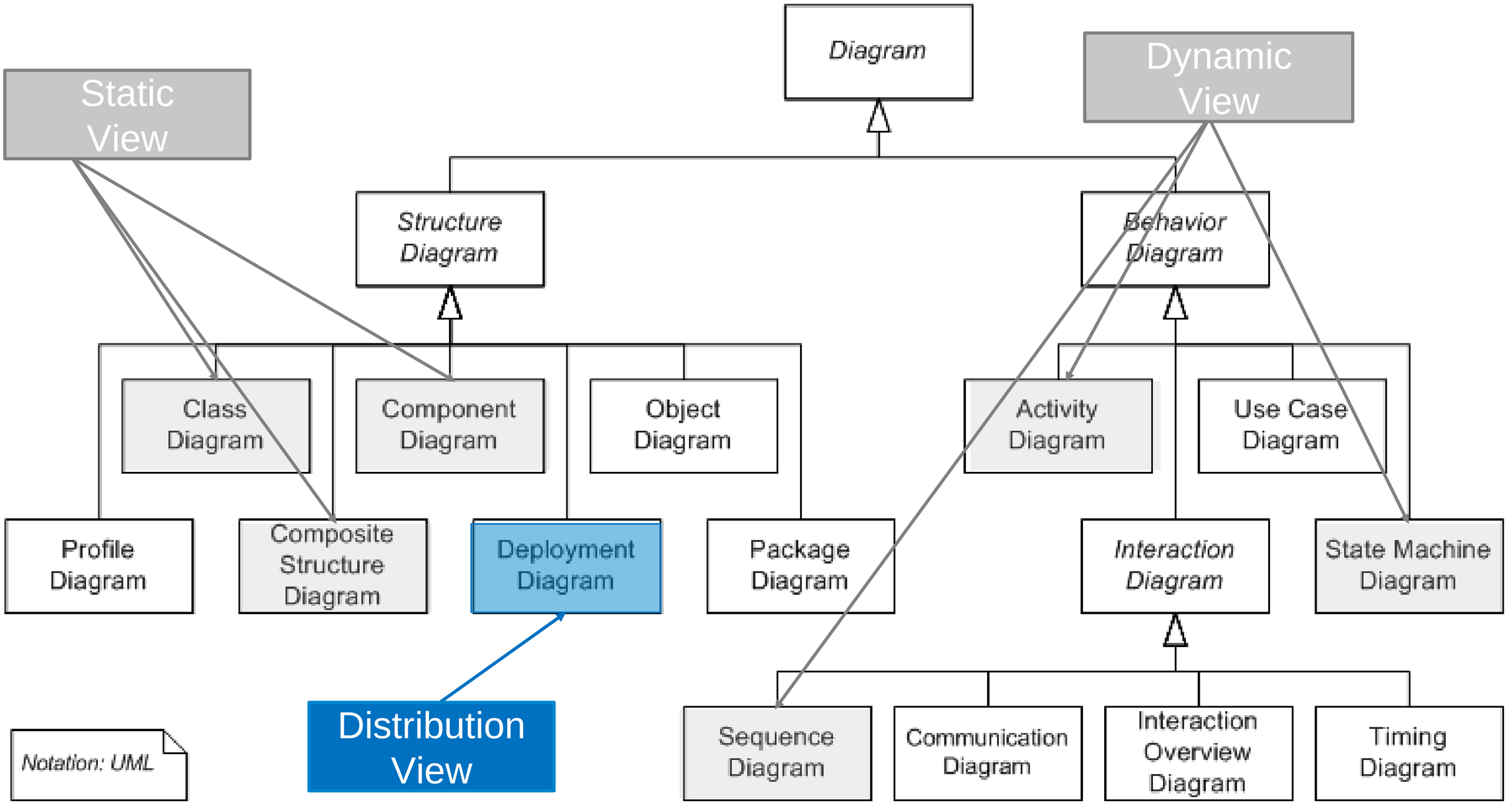
“Delegate” connectors are used to connect ports with internal interfaces.



UML2 Composite Components



Distribution View



Distribution

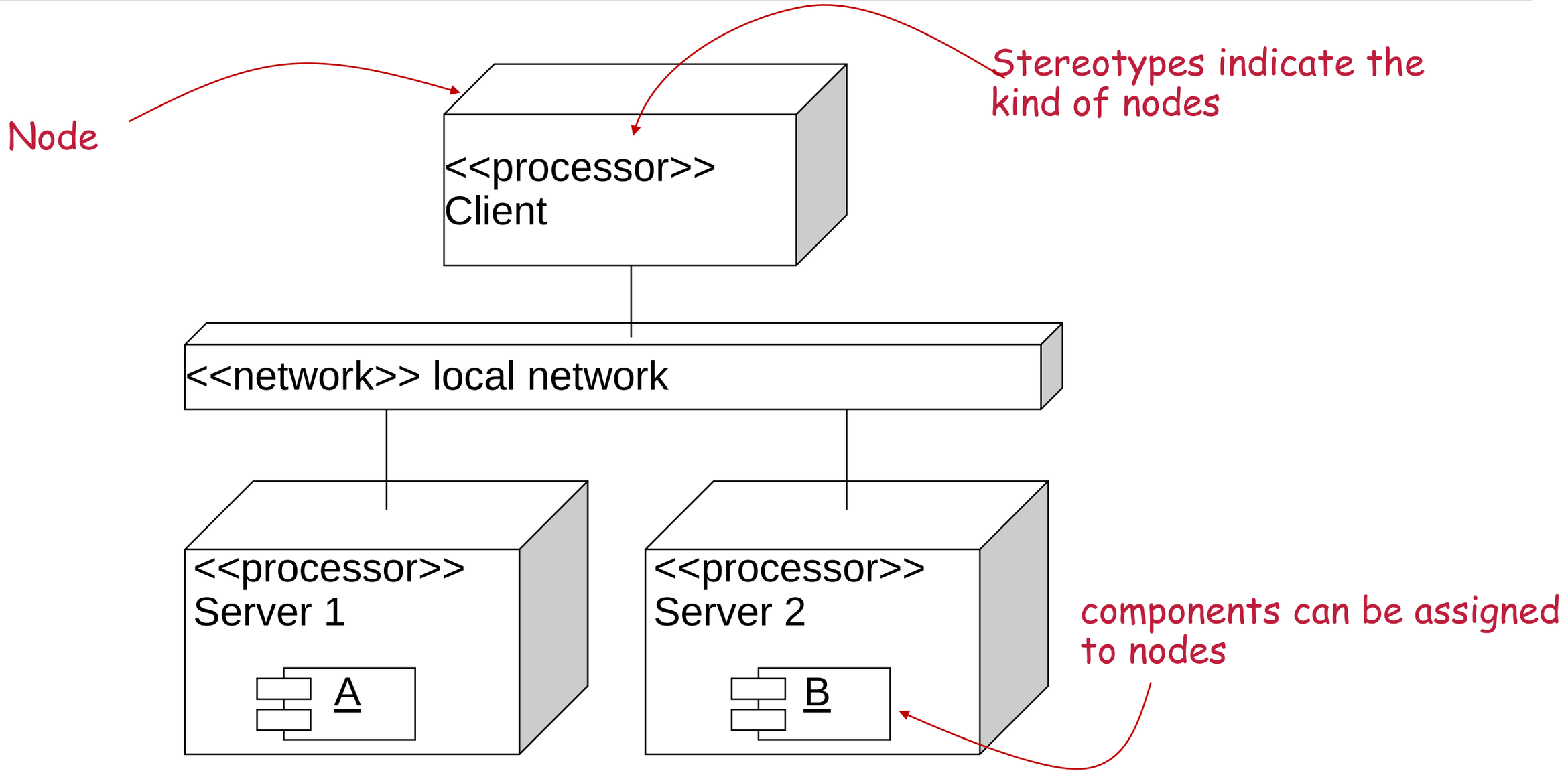
Shows physical distribution of the system and its code

Focus on physical resources (nodes, network, application server, etc.)

Artifacts represent executable files such as libraries, archives, database schemas, configuration files, etc.

No reference to particular instances of artefacts and nodes.

Deployment Diagram



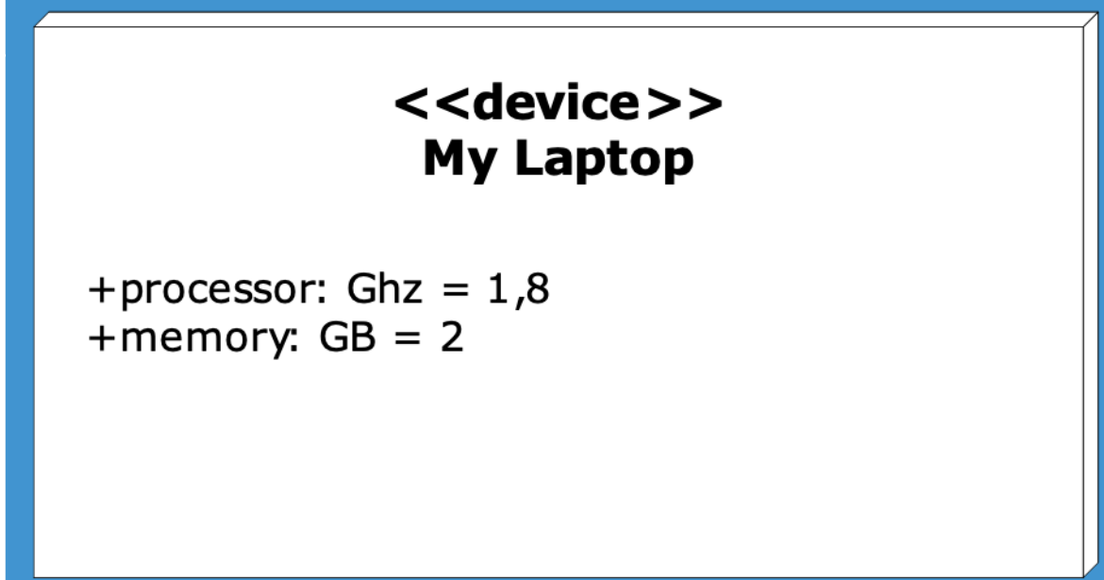
Nodes

Physics or logical deployment unit

Typical nodes
Hardware:
- Device
- Client workstation
- Mobile device
- Embedded device

Software:
- Execution environment
- Application server

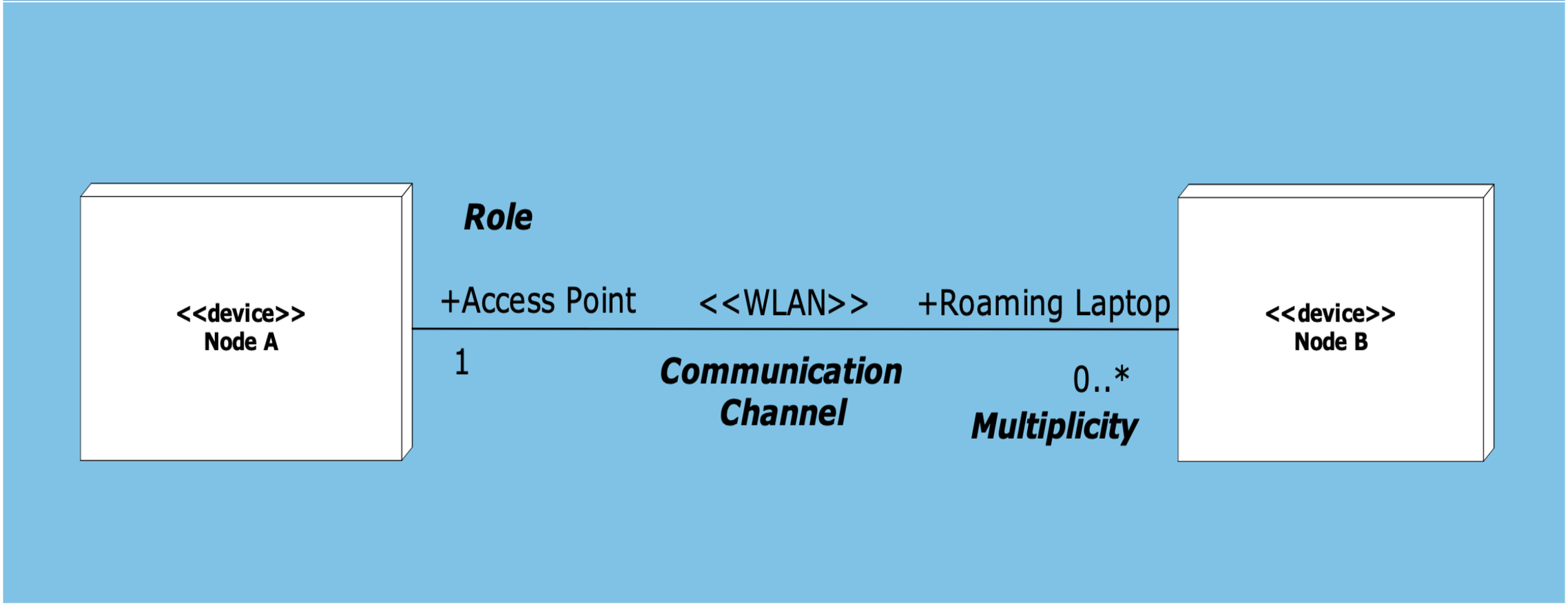
Can contain descriptive atrtibutes (e.g. memory, processor, …)



Communication Channel

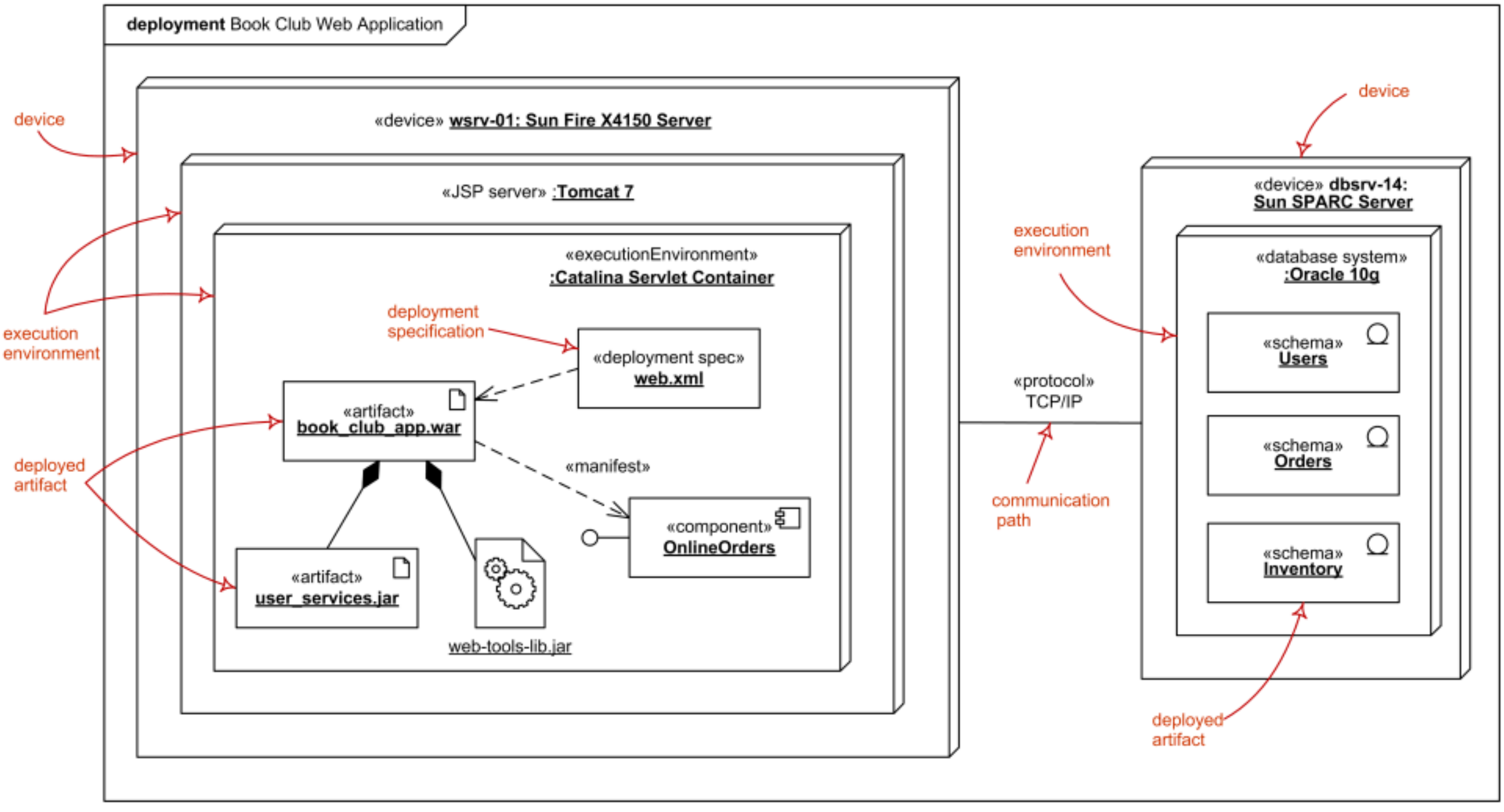
Communication Channel

- connection of nodes
- representing the infrastructure



Deployment Diagram

Example



OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

FACULTY OF COMPUTER SCIENCE

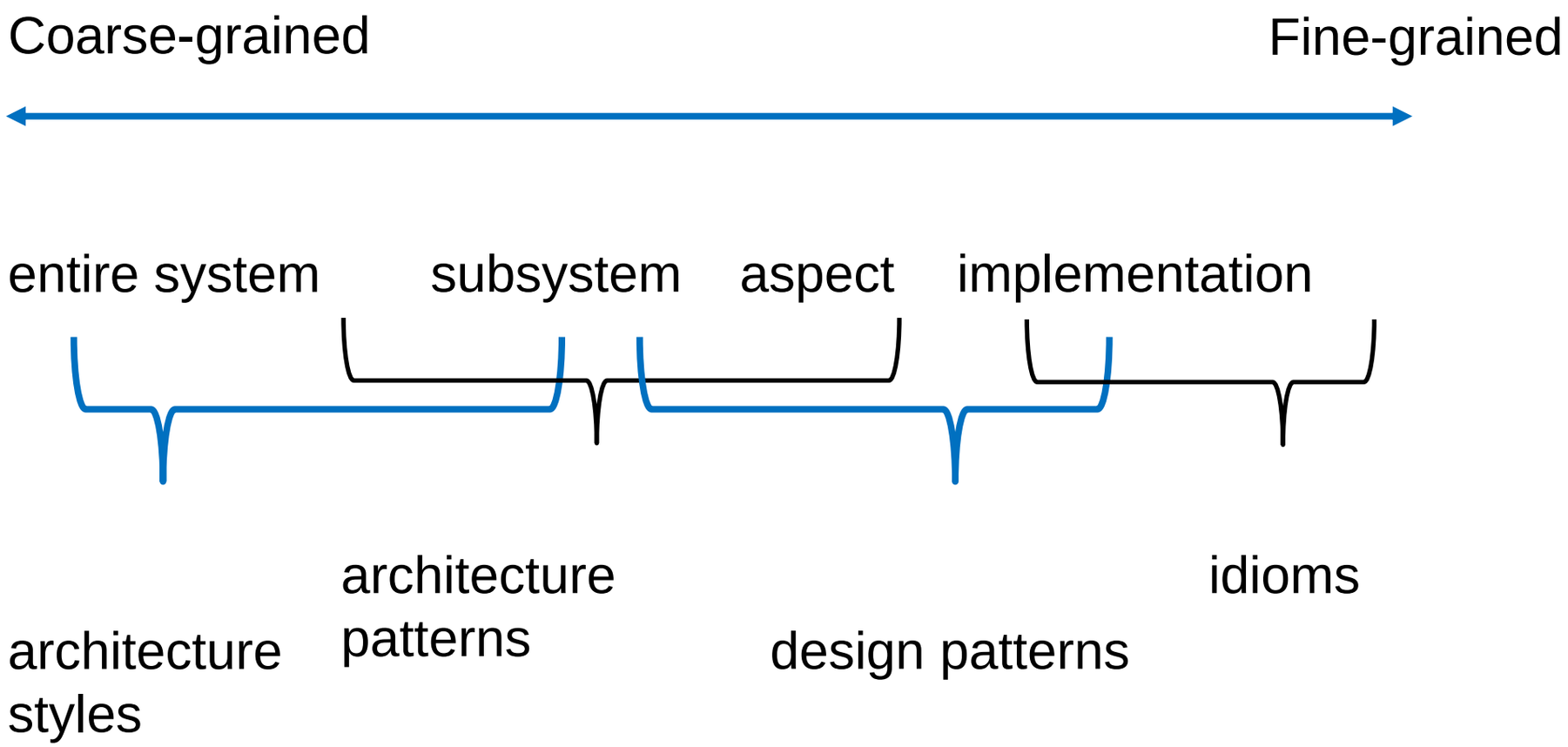
SwarmLab

Introduction to Software Engineering for Engineers L-06: Software Architectures

Part 4: Software Architecture Styles & Patterns

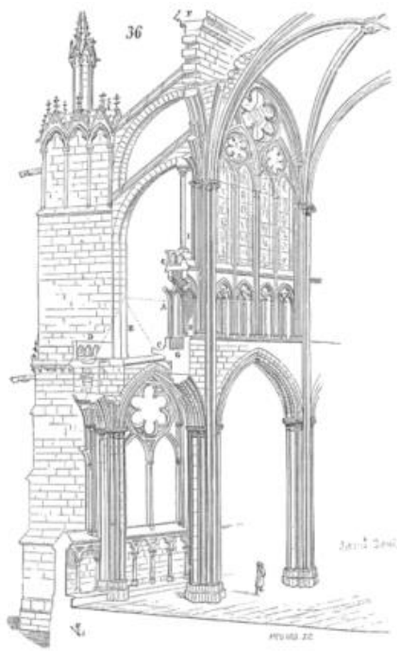
Dr.-Ing. Christoph Steup

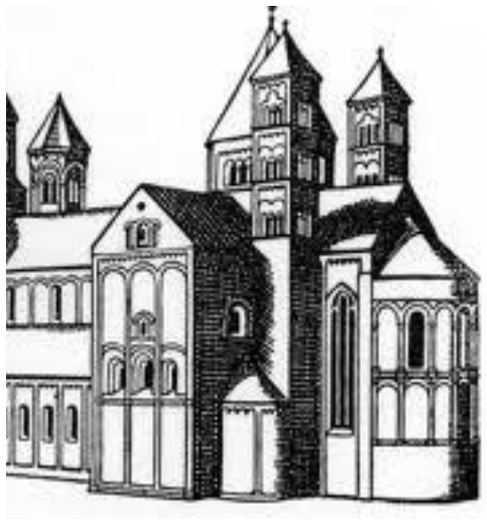
Overview


Coarse-grained
Fine-grained
entire system
subsystem
aspect
implementation
architecture styles
architecture patterns
design patterns
idioms

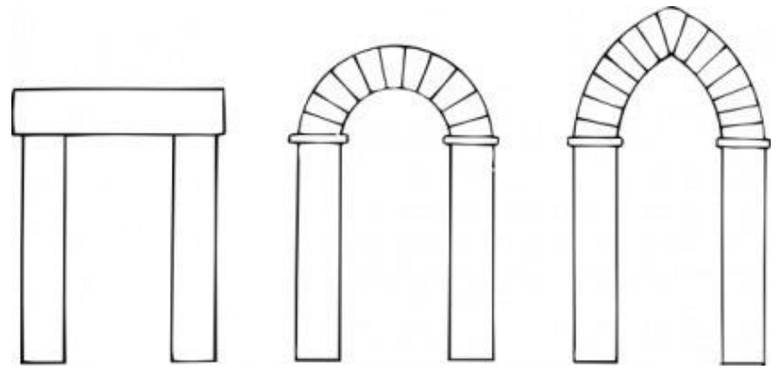
Overview

- Architecture styles
  - Layers
  - Pipes and Filters
  - Blackboard
  - Client-Server / Peer-to-Peer
  - Service-orientierte Architectures (SOA)





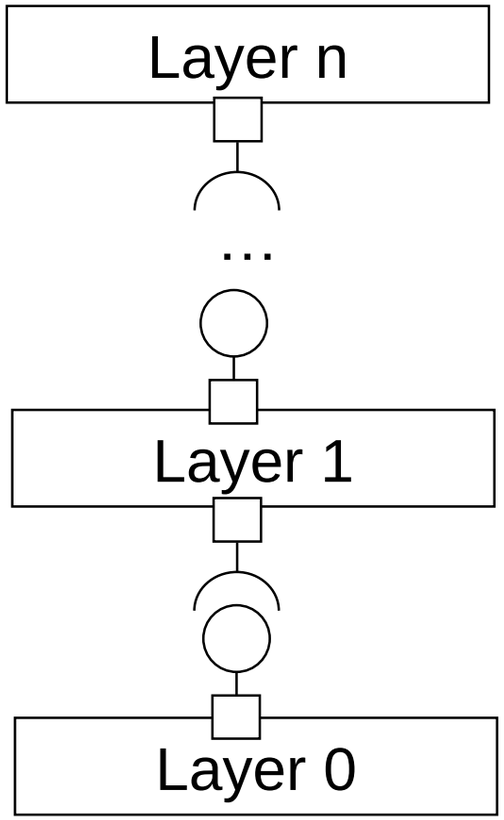
- Architecture patterns (excerpt)
  - Model-View-Controller
  - Proxy
  - Broker
  - Client-Dispatcher-Server
  - Active Objects

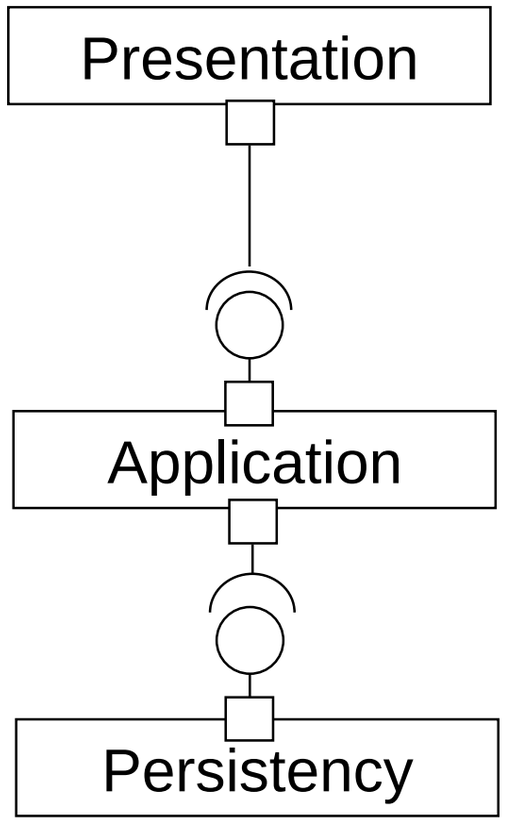


Layers

- Services are offered from bottom to top
- sub tasks are delegated from top to bottom
- 1:1-relation between layers
- layers can be skipped (Relaxed Layered System)

component diagram:



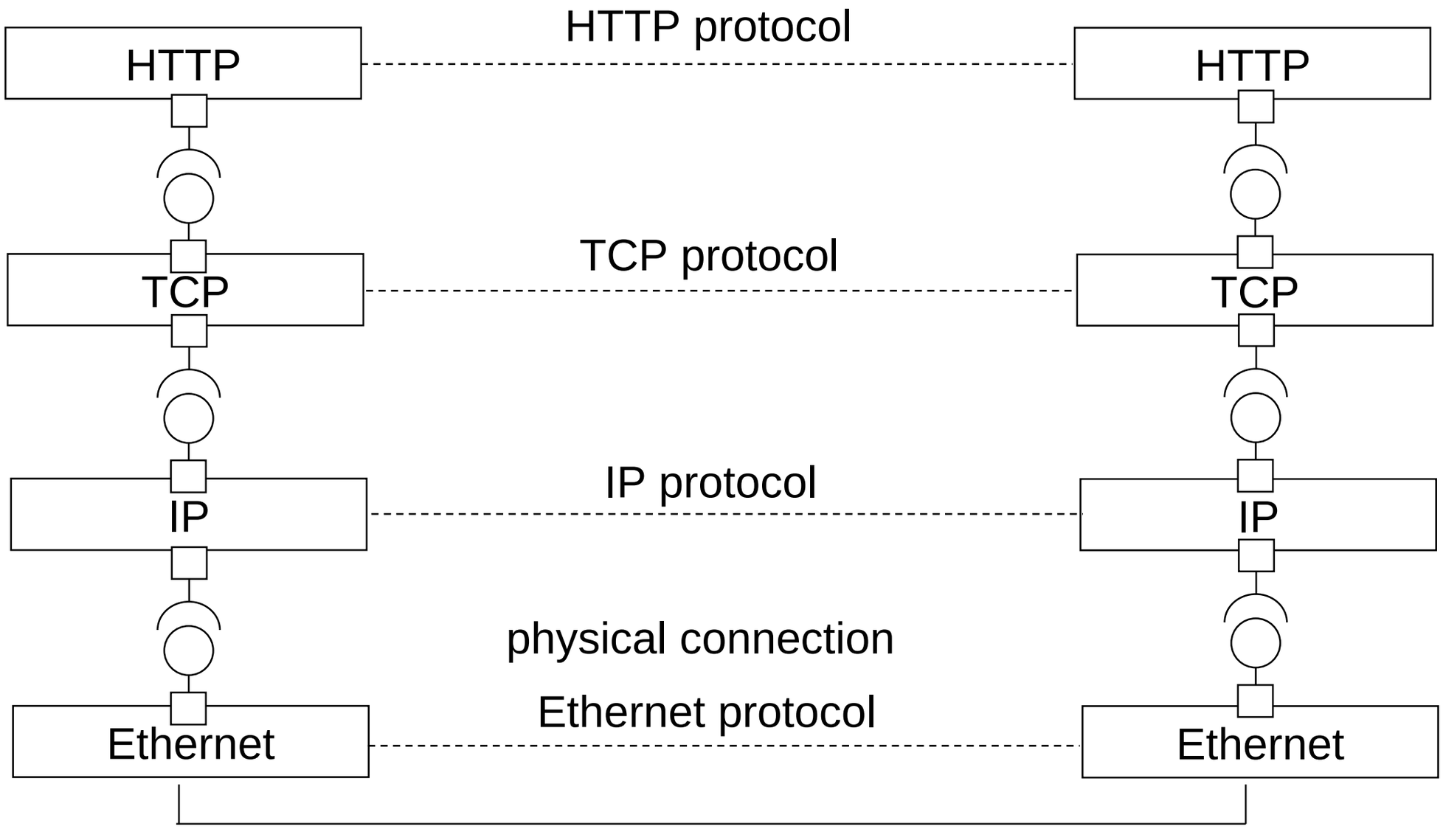


Example for Layers - TCP/IP

Example

Logical communication between services

Actual communication only via physical connection



Pipes and Filters

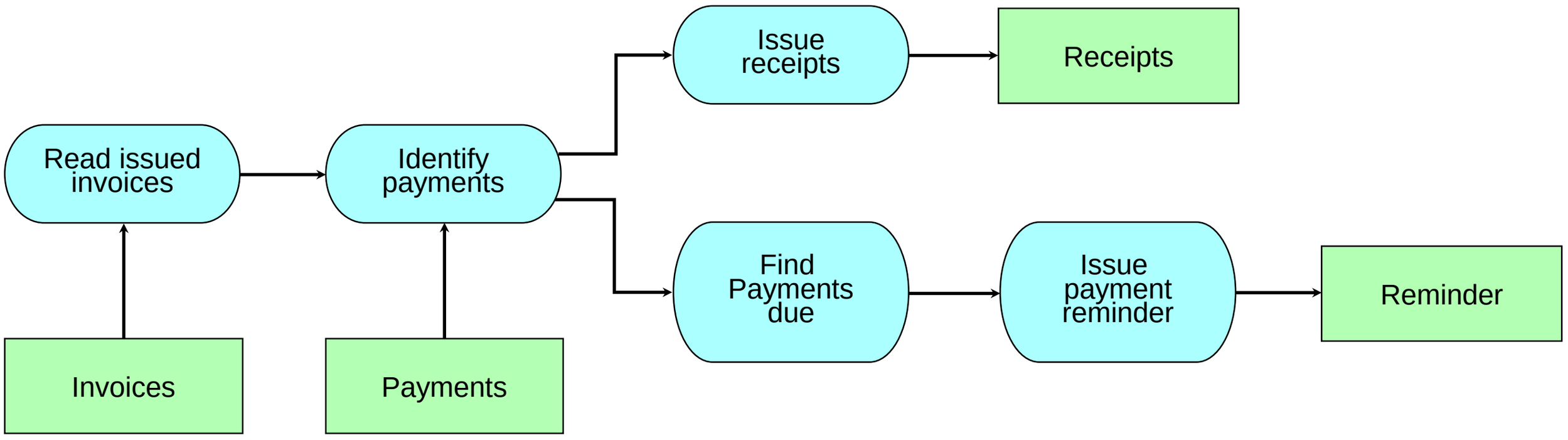
- Goal:
  In a system, data streams pass through different processing phases.
  - data flow should be easy to change.
  - processing phases should be exchangeable, extensible and reusable.

- Style:
  Description of processing phases by filters and data streams by pipes.
  - filter components have input/output and process a data stream.
  - Pipes: data stream from a filter input to a filter output.
  - Filter process input data continuously.

Exampe for Pipes und Filters

Example

Accounting Management System



[nach Reussner]

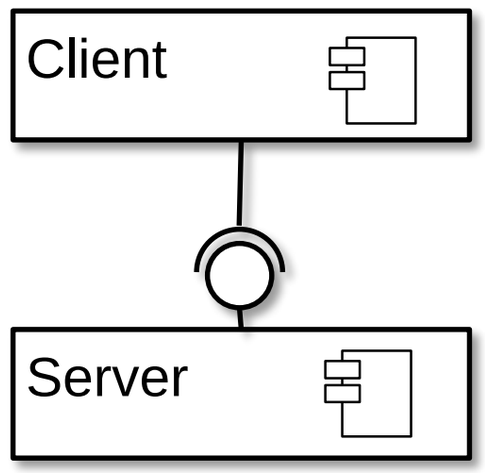
2-Tier (Client-Server)

Problem:

- Several clients should work on the same data.

Solution:

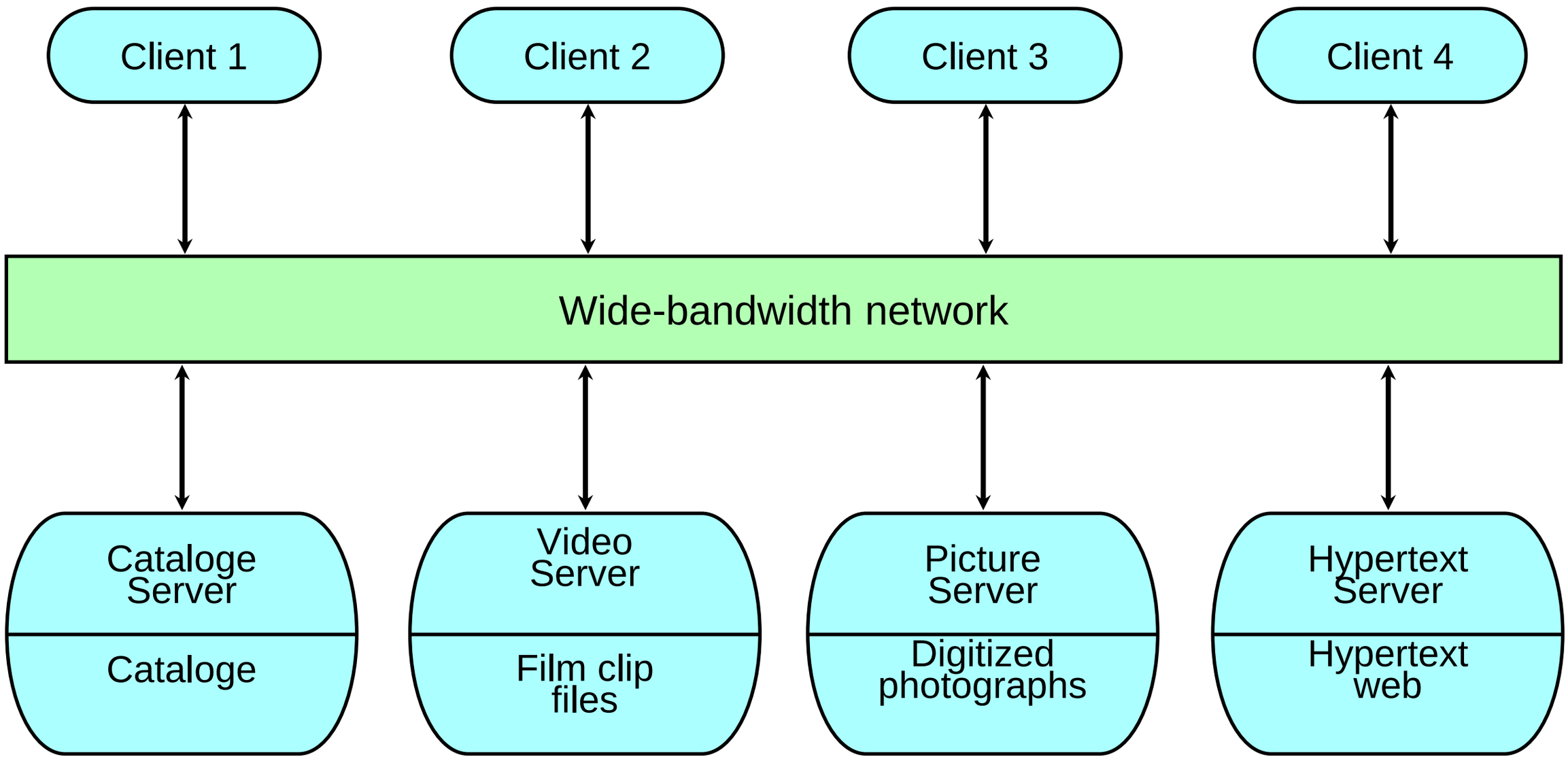
- Separation of application and data management



Difference to layered architectures?

Client-Server - Example

Example



[nach Reussner]

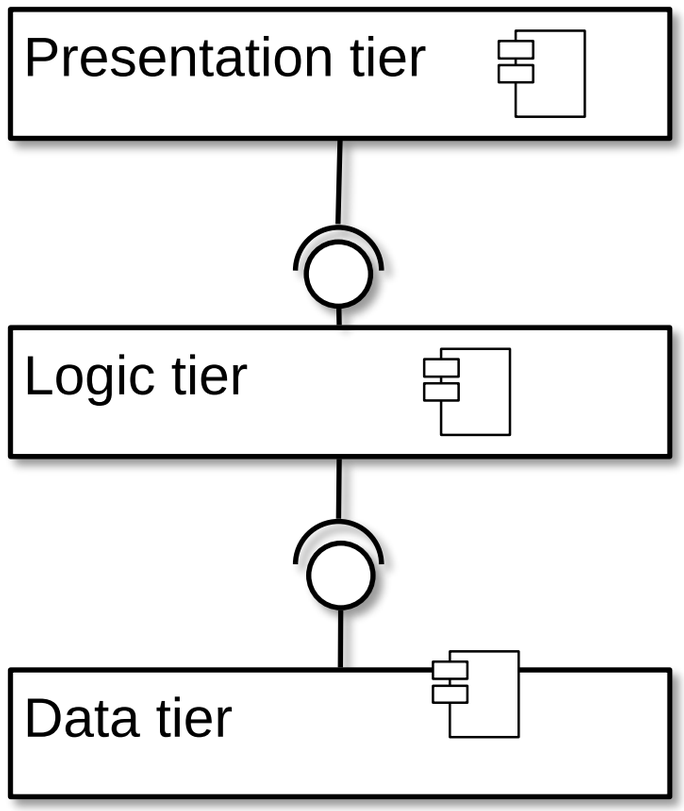
3-Tier Architecture

Problem:

- Different clients should provide the same functionality
- Presentation of data should be exchangeable

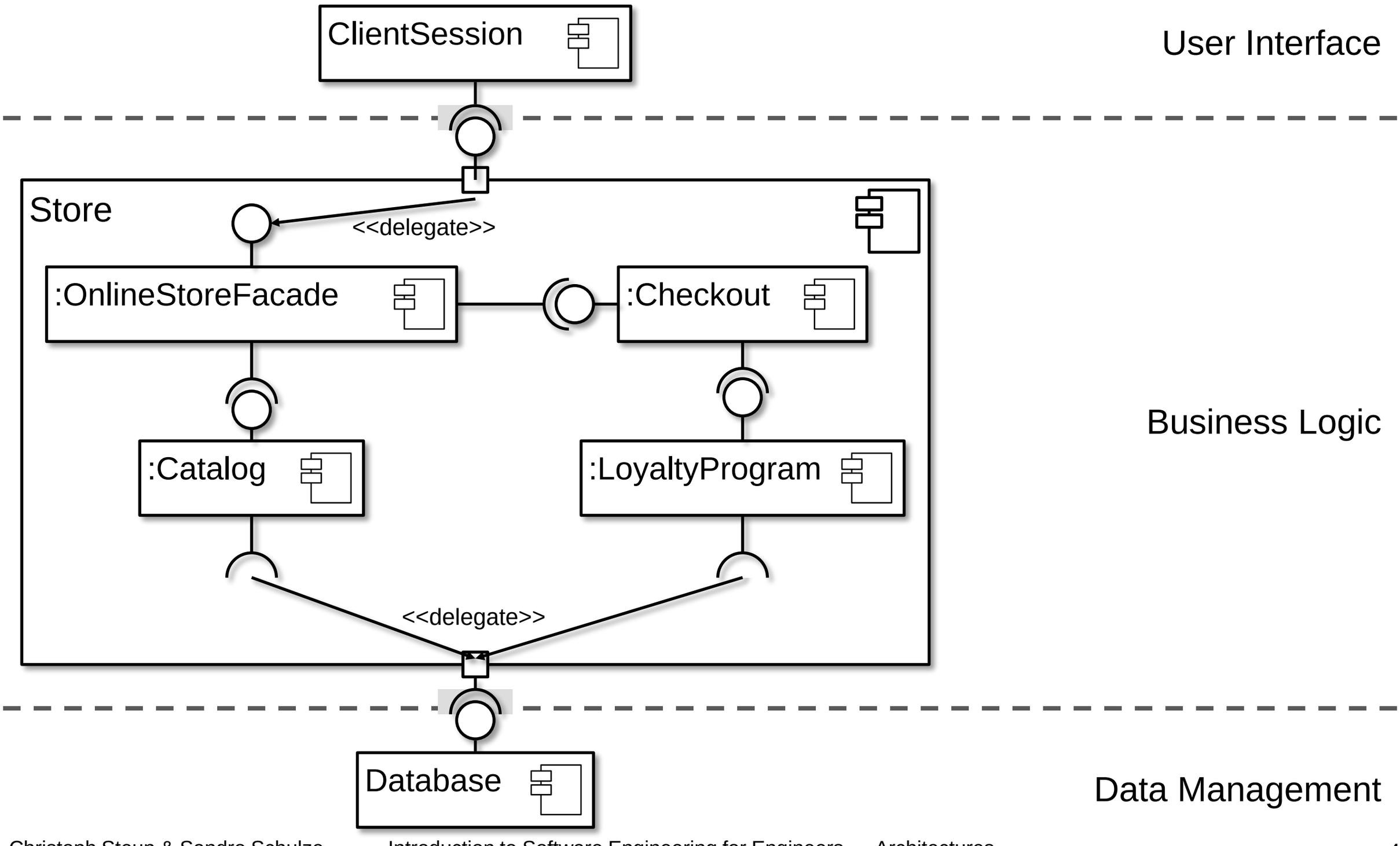
Solution:

- Implementation of presentation in separate layer



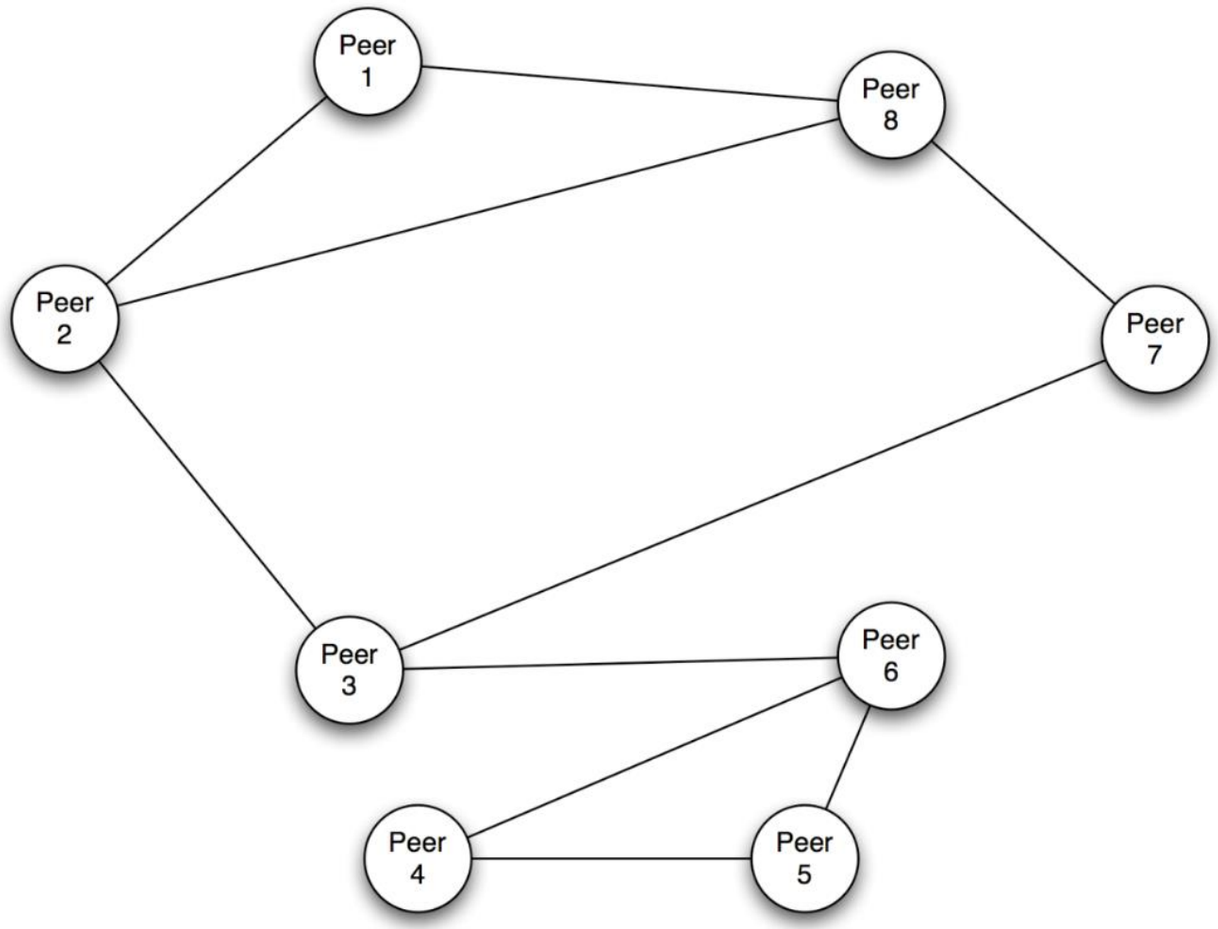
3-Tier Architecture - Example

Example



Peer-to-Peer

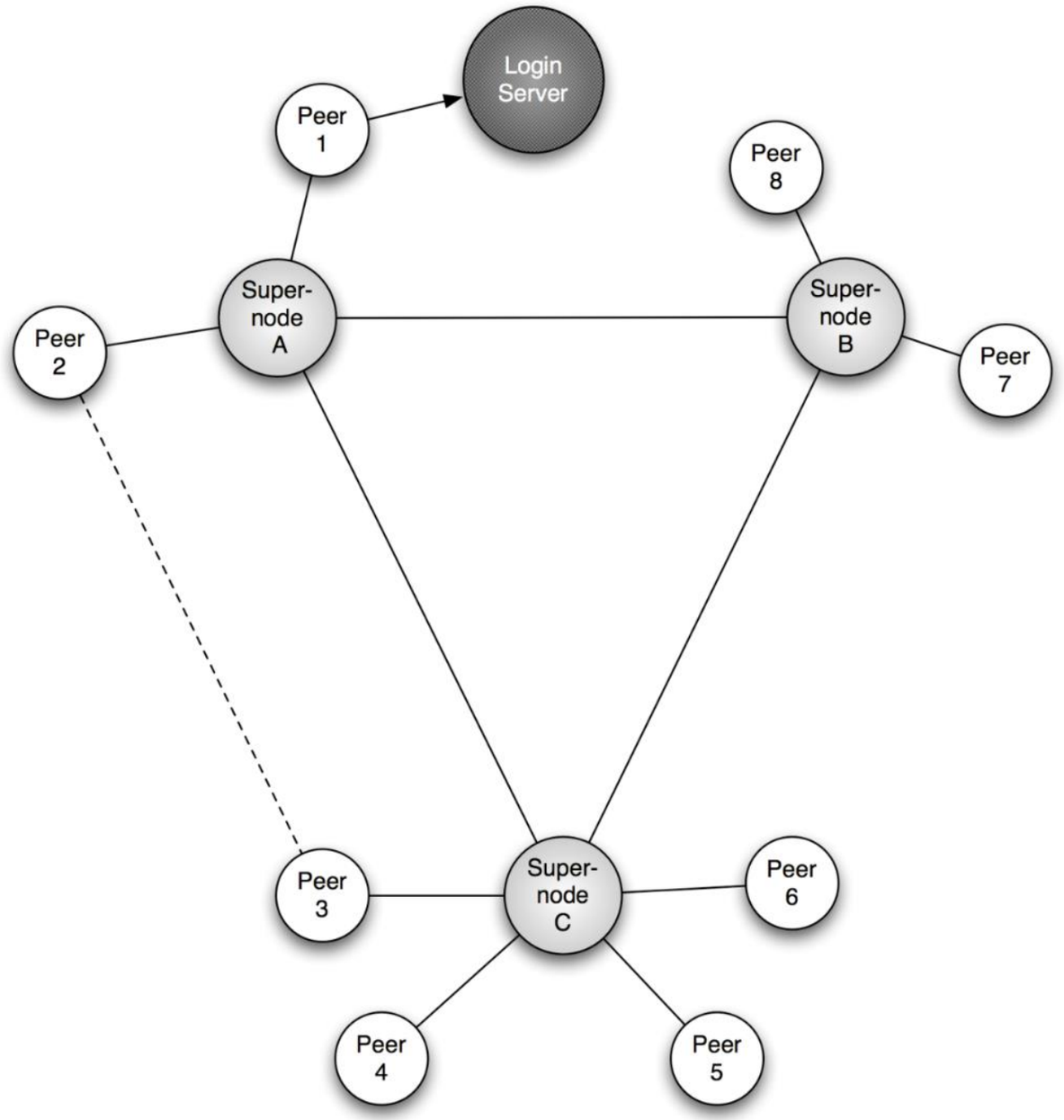
- Peers can be Clients and/or Servers.
- Arbitrary topology
- Decentralized computation with control and data flow between Peers
- Robust against breakdown of particular nodes
- scalable with respect to resources and machine capacity
- Examples: BitTorrent, Napster, Gnutella, …



Example - Skype

Example

- Mixed Client-Server and Peer-to-Peer architecture for discovery problem
- Replication and distribution of directories (supernodes) for scalability and robustness.
- Upgrade of ordinary peers to supernodes based on network and computing performance
- Encryption for routed calls



Model-View-Controller (MVC)

Conditions for application:

- An application contains several views on the same data.
- User can change that data in the views.
- Views need to be kept up-to-date and changes of data have to be depicted.

Solution:

Decomposing the system into three components

- Model
  - data management of the system
- View
  - view on data
- Controller
  - user interface and modification of data

usually: additional application of Observer pattern for messaging

Model-View-Controller - Structure

Model

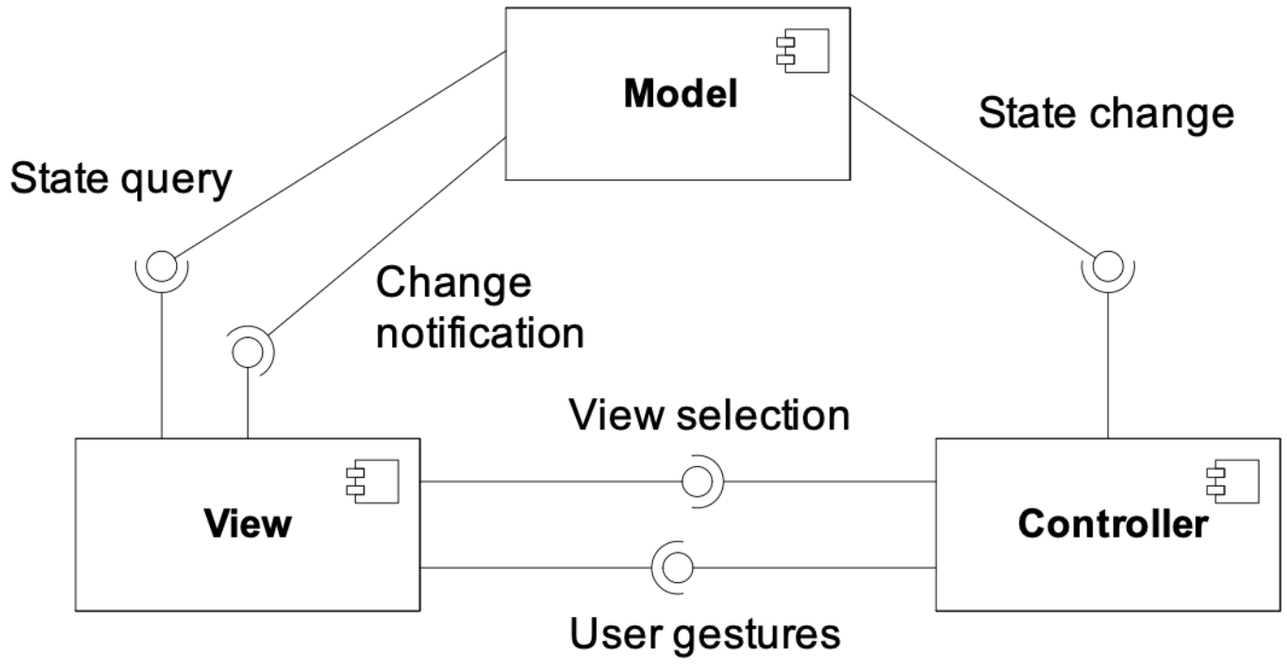
- captures state of application and functionality
- notifies Views regarding changes

View

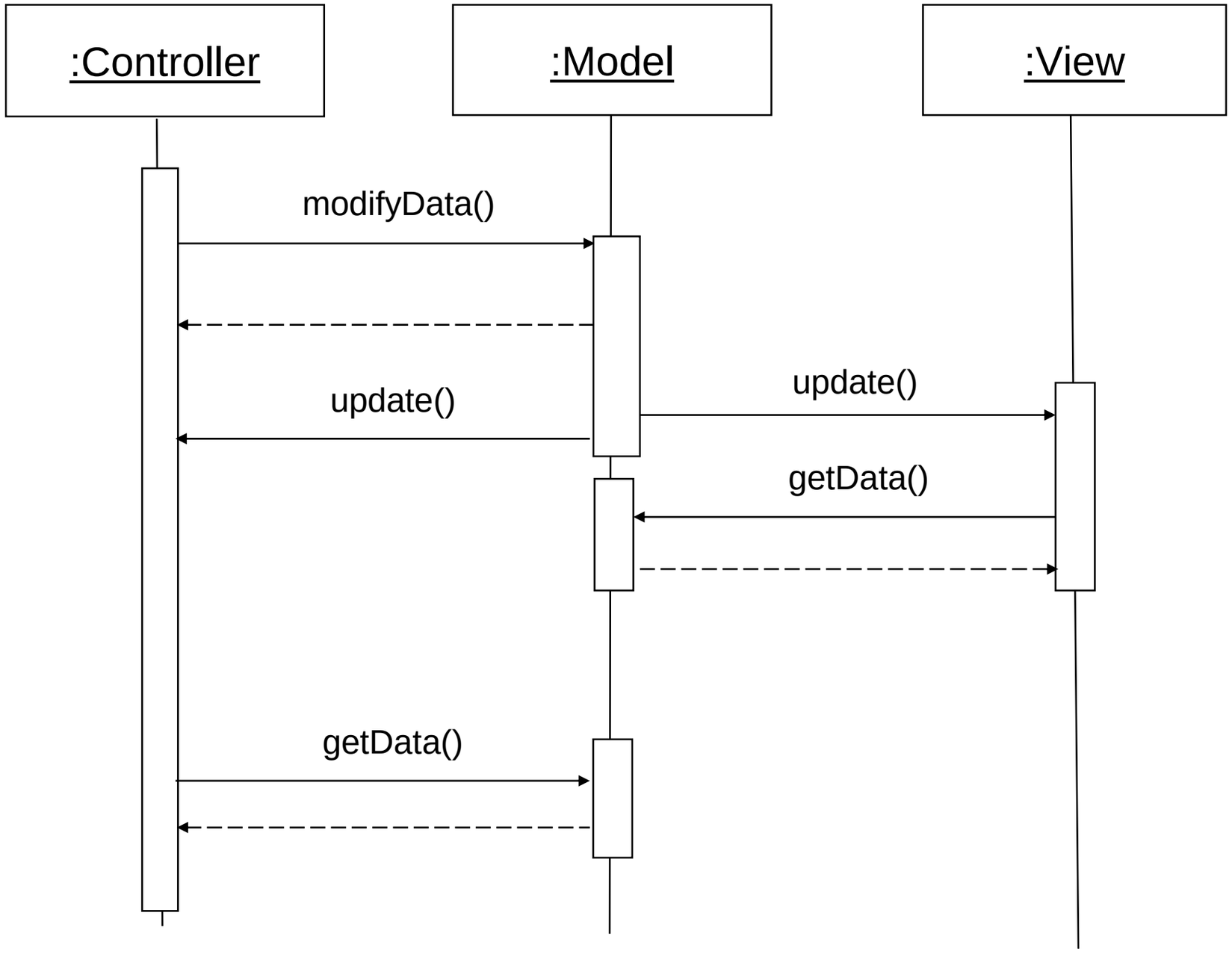
- queries updates from model(s)
- Mapping of user inputs and controller commands

Controller

- Mapping of commands to model updates

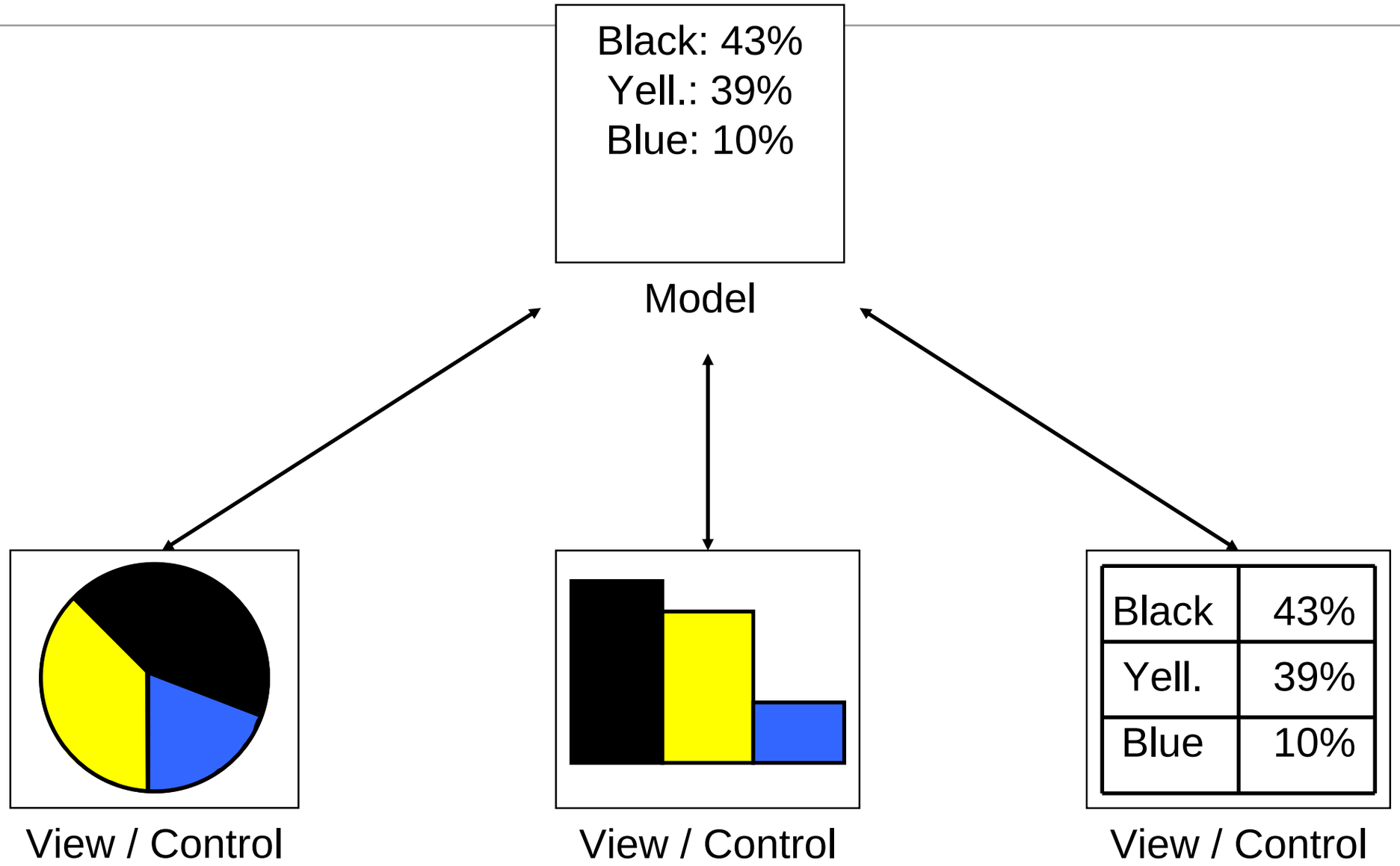


MVC: Process of Data Update



MVC - Example

Example



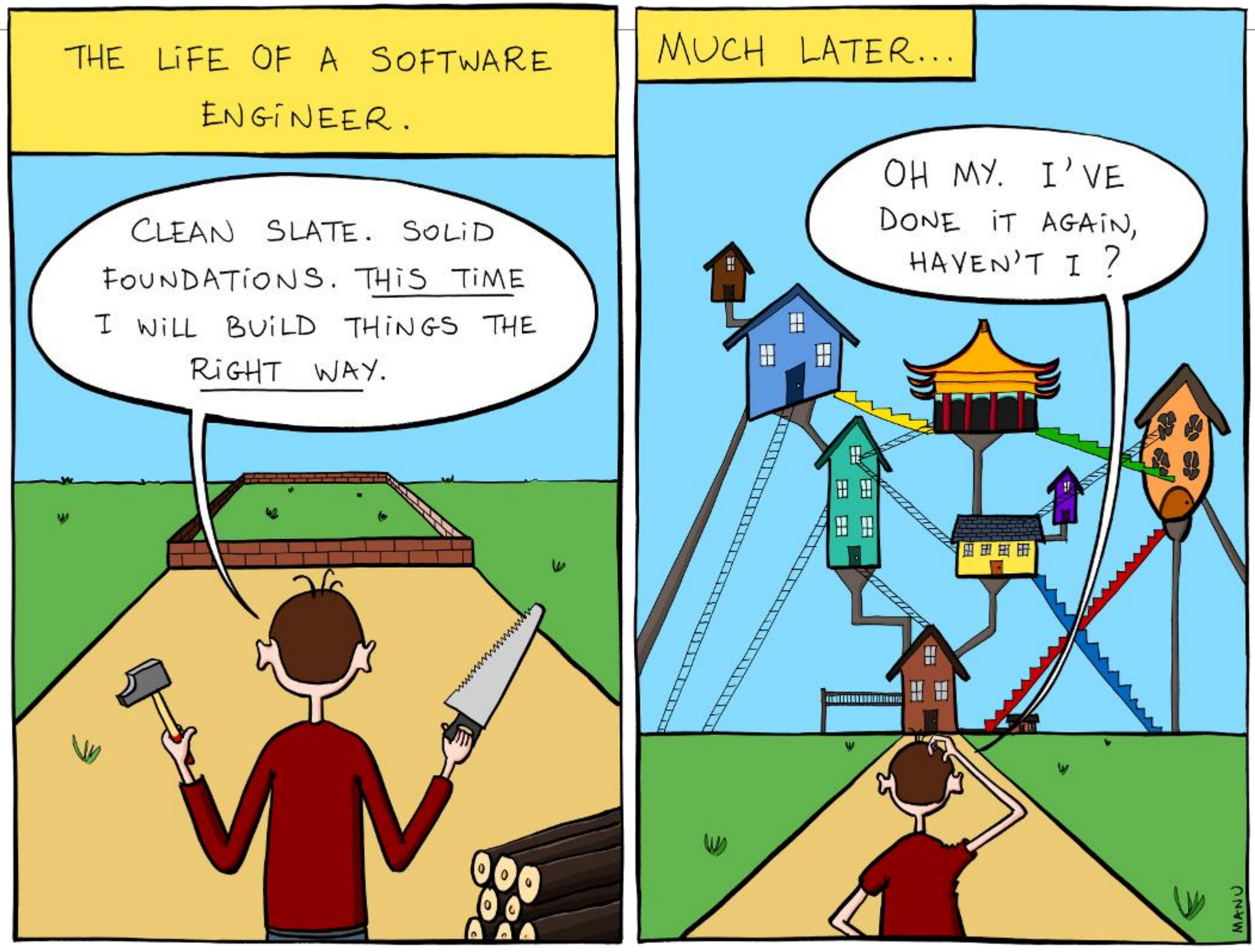
Summary

Software architecture depends on the perspective:

- Structural diagrams
- Behavioral diagrams
- Deployment diagrams

well-known architectures as reference.

The Reality


THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.
MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
MANU

Literature

- R. Reussner, W. Hasselbring (Hrsg.), Handbuch Softwarearchitektur, dpunkt Verlag, 2006 (**in German**)

- R. Taylor, N. Medvidovic, and E. Dashofy: Software Architecture: Foundations, Theory, and Practice", Wiley, Hoboken, 2009

- Clements et al. "Documenting Software Architectures", Addison Wesley, 2003

- T. Posch, K. Birken, M. Gerdom, Basiswissen Softwarearchitektur, dpunkt Verlag, 2011 http://www.dpunkt.de/buecher/3527/basiswissen-softwarearchitektur.html
(**in German**)