HOCHSCHULE
KONSTANZ
TECHNIK, WIRTSCHAFT
UND GESTALTUNG

Verteilte Systeme
Vorlesung 04

12. Apr. 2017

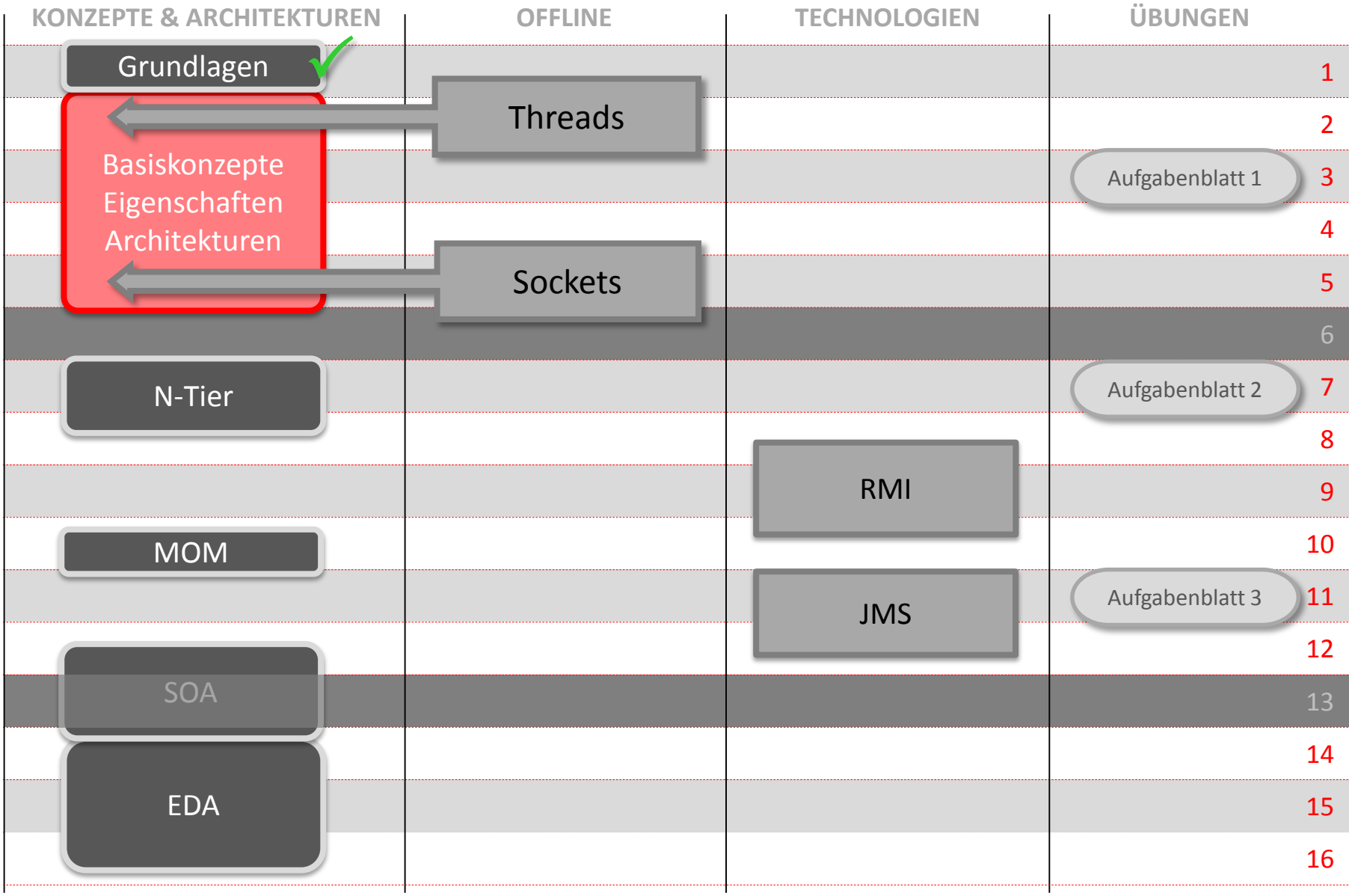
Kapitel 3: Basiskonzepte, Eigenschaften, Architekturen

Prof. Dr. Rainer Mueller

SS 2017

Hochschule Konstanz | Brauneggerstr. 55 | 78462 Konstanz | www.htwg-konstanz.de

Vorlesung: Übersicht



Übersicht

K A P I T E L 3

Vom zentralen Fall übertragbare Basiskonzepte, daraus ableitbare grundlegende Eigenschaften, erste teilweise auch zentral gültige Strukturen und Architekturelemente

Übersicht

E I G E N S C H A F T E N

Qualitätsmerkmale

Für die Güte eines verteilten Systems

Entwurfskriterien

Bei der Architektur und Entwicklung eines verteilten Systems

- Transparenz
- Skalierbarkeit
- Nebenläufigkeitsgrad
- Kommunikationsgrad

3.7 Kommunikationsgrad

zwei Perspektiven
- Architektur
- technische Perspektive

Wer kommuniziert?

Wer hat welche Kommunikationsrolle?

Architektur

→ 3.10 Rollenauflösung von Client und Server

Kommunikationskonzepte:

Rollen, Dienste, Architekturen

Technische Perspektive

→ 3.7 Kommunikationsgrad

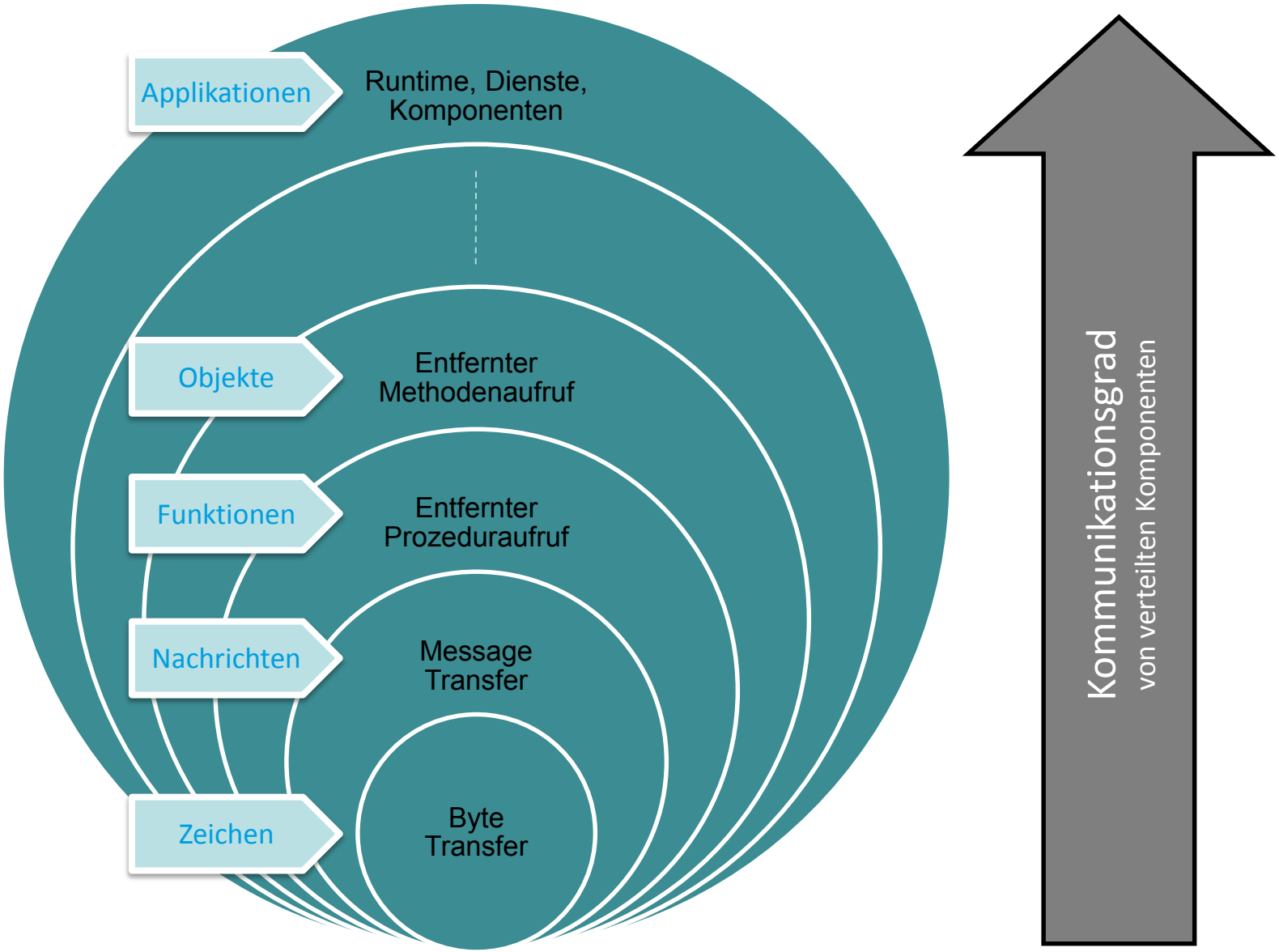
Kommunikationselemente:

Pakete, Nachrichten, Aufrufe

Wie und **was** kommunizieren wir?

Wie reagieren wir und mit **was?**

3.7.1 Schalenmodell



3.7.2 Sockets

Sockets für uns Kommunikations-Programmierschnittstellen

SYSTEM- UND PLATTFORMÜBERGREIFENDE API FÜR IPC

- Industrie-Standard → Kompatible Socket-Implementierungen auf verschiedenen Plattformen
  - *Beispiel: Windows SDK (Winsock), Java, . NET*
- Herkunft: Berkeley UNIX
- Die API für TCP-UDP-/IP-Protokolle
  - Adressierung, Management und Sicherung der Datenübertragung nach den TCP-UDP-/IP-Regeln
- Protokolle: Schicht 4 des OSI-Referenzmodells
  - **Stream**: Verbindungsorientierte Kommunikation → TCP
  - **Datagram**: Verbindungslose Kommunikation → UDP
- Duplex-Datenübertragung (ggf. gesichert) **ein und ausgehende Übertragung**
- Datenfluss: Byte-weise, nicht block-weise, nicht nachrichten-basiert
- Konzept und Begrifflichkeit: Kommunikationsendpunkte für Applikationen im Sinne von Steckdosen

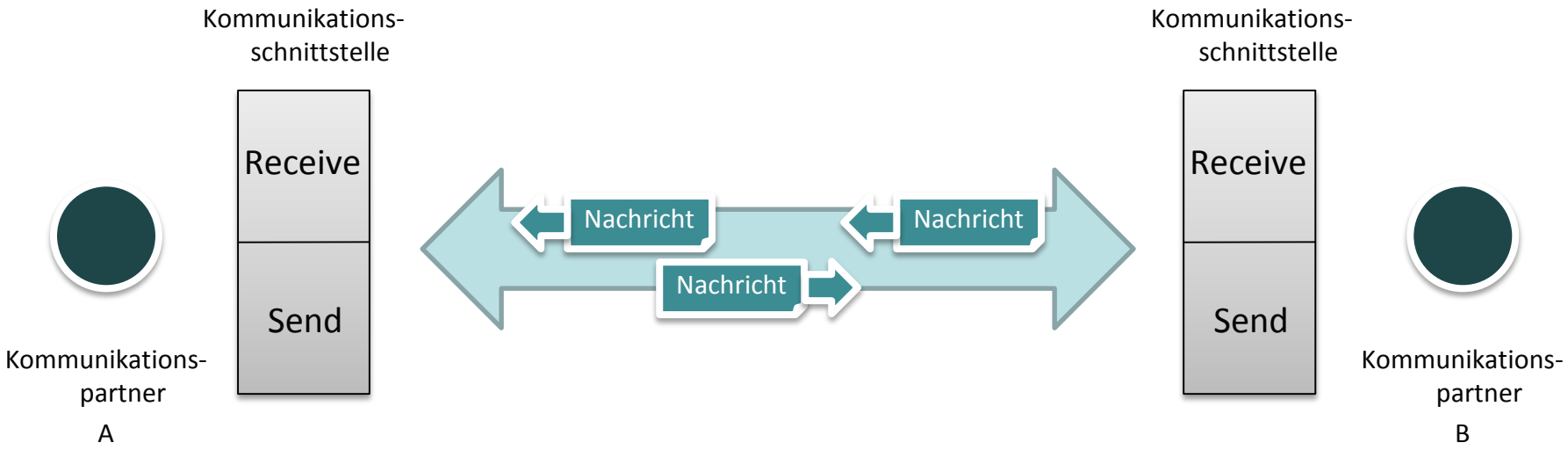
3.7.3 Nachrichten

ABSTRAKTIONSSCHICHT OBERHALB VON SOCKETS

- Basis: Dienstprimitive **Send** (receiverAddress, message) und **Receive** (senderAddress, message)
- Mögliches Schicht 4-Protokoll: TCP/UDP
- Keine Initialisierung oder Aufräumsequenz
- Verbindungslose, paketorientierte Kommunikation

Nachricht: ein Stück Information das von A nach B übertragen wird

Entkopplung: Kommunikationspartner sind unabhändig von einander z.B. können Kommunikationspartner Offline sein

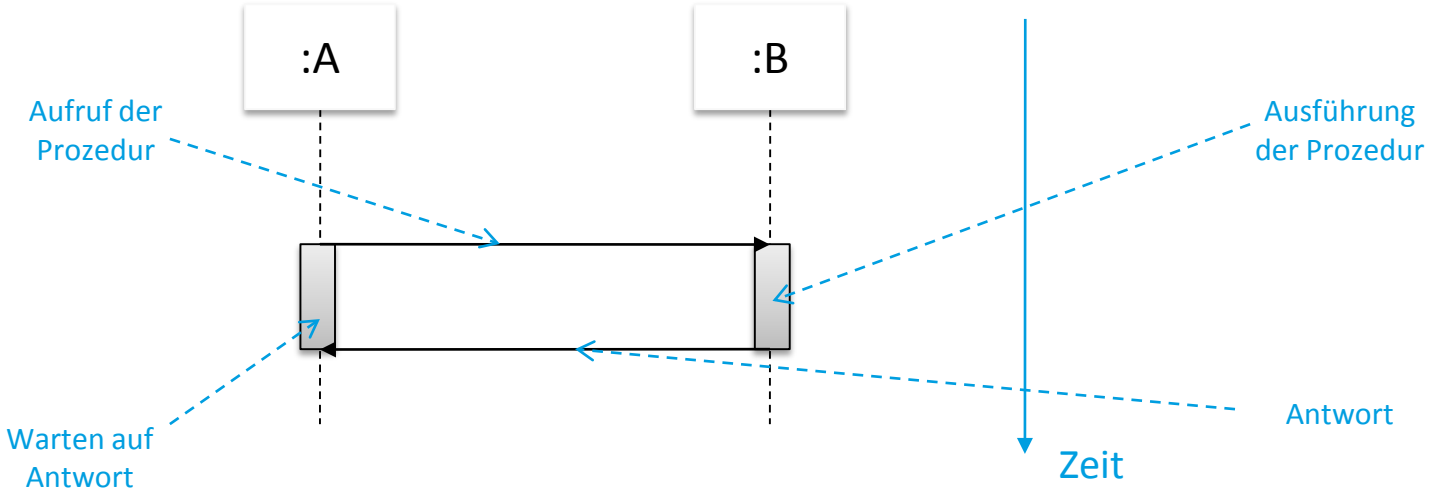


3.7.4 Entfernter Prozeduraufruf

NACHAHMUNG DES LOKALEN PROZEDURAUFRUFS

Request For Command: theoritische Vorschläge oder Definitionen werden von Fachkundigen kritisiert (Standard festlegen)

- Bezeichnung: RPC (Remote Procedure Call)
- Ziel: Aufruf von Funktionen in fremden Adressräumen
- Herkunft: Sun Microsystems (entwickelt für NFS)
- Standard: IETF (RFC 1057, RFC 5531)
- Grundlage für: Java RMI, CORBA, DCOM, XML-RPC, RPyC (für Python)
- Verwendung in prozeduralen Programmiersprachen
- Kommunikationsprinzip: Synchron mit Handshake (→ Aufrufer wartet)
  - Alternative: Asynchron ohne Warten (→ Antwort über Exception, Callback oder Polling)



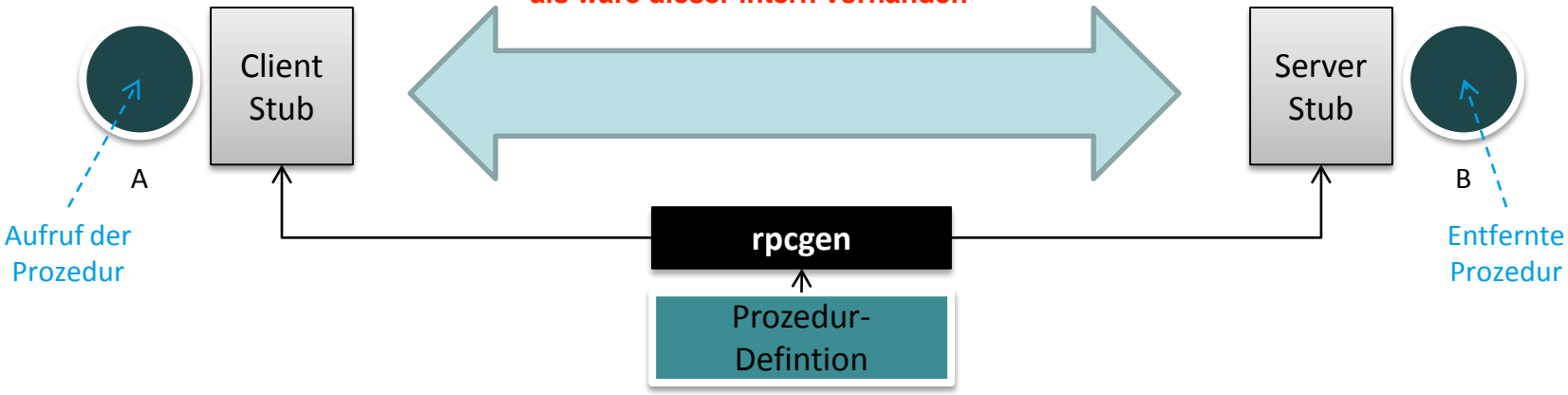
3.7.4 Entfernter Prozeduraufruf

RPC-MIDDLEWARE MIT STUBS

- Definition der entfernten Prozedur
  - Spezifikation in Format ähnlich zu C-Header-Datei
  - Implementierung als C-Funktion
- RPC-Generator **rpcgen**
  - Erstellt **Client Stub** und **Server Stub** aus entfernter Prozedur
- **Parameter-Marshalling**
  - Kommunikationspartner ruft RPC-Prozedur auf
    - → Aufruf des zugehörigen Client Stub
    - → Konvertierung in **XDR** (External Data Representation): Plattformunabhängiges Format
    - → Versand der XDR-Nachricht an Server Stub
    - → Aufruf der eigentlichen Prozedur auf dem Server
  - Rückweg (Antwort) analog

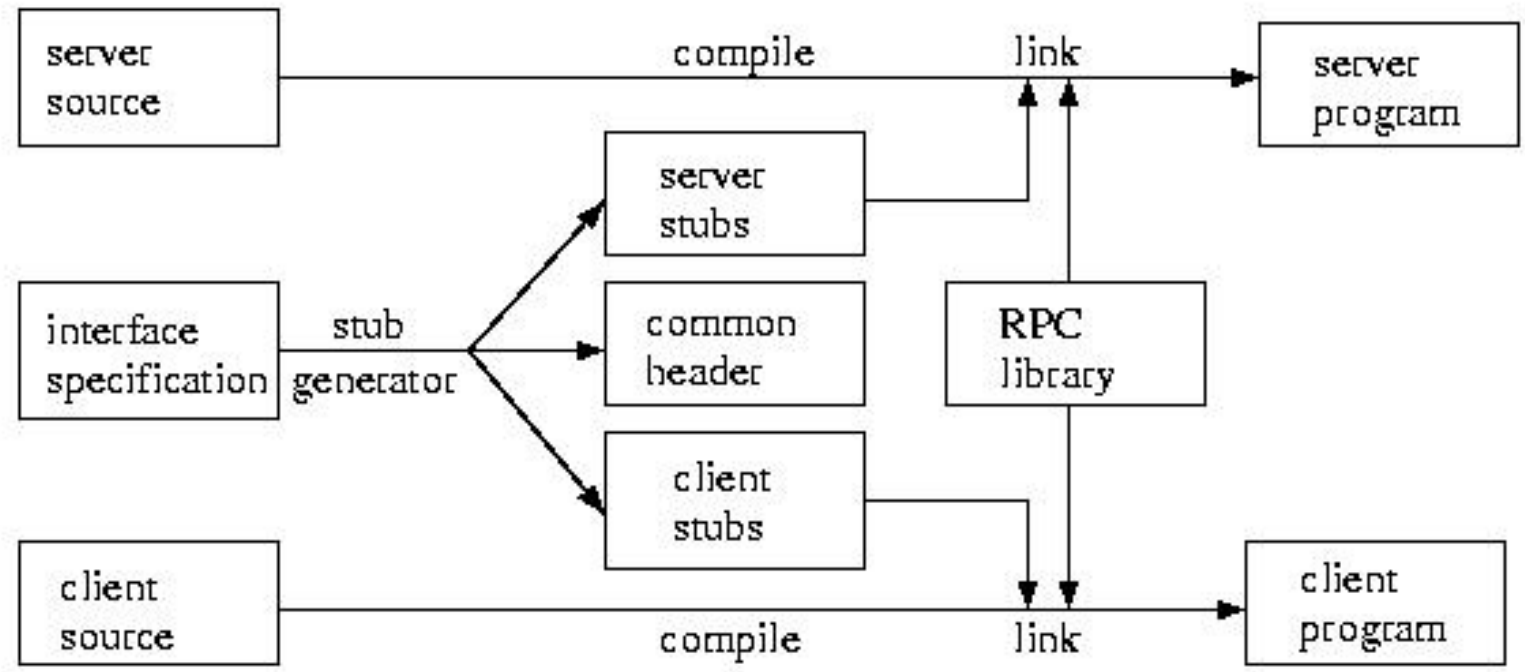
Signatur: Rückgabe der Funktion

Stub: automatisch generiert, ermöglicht kommunikation eines extnernes rechners als wäre dieser intern vorhanden



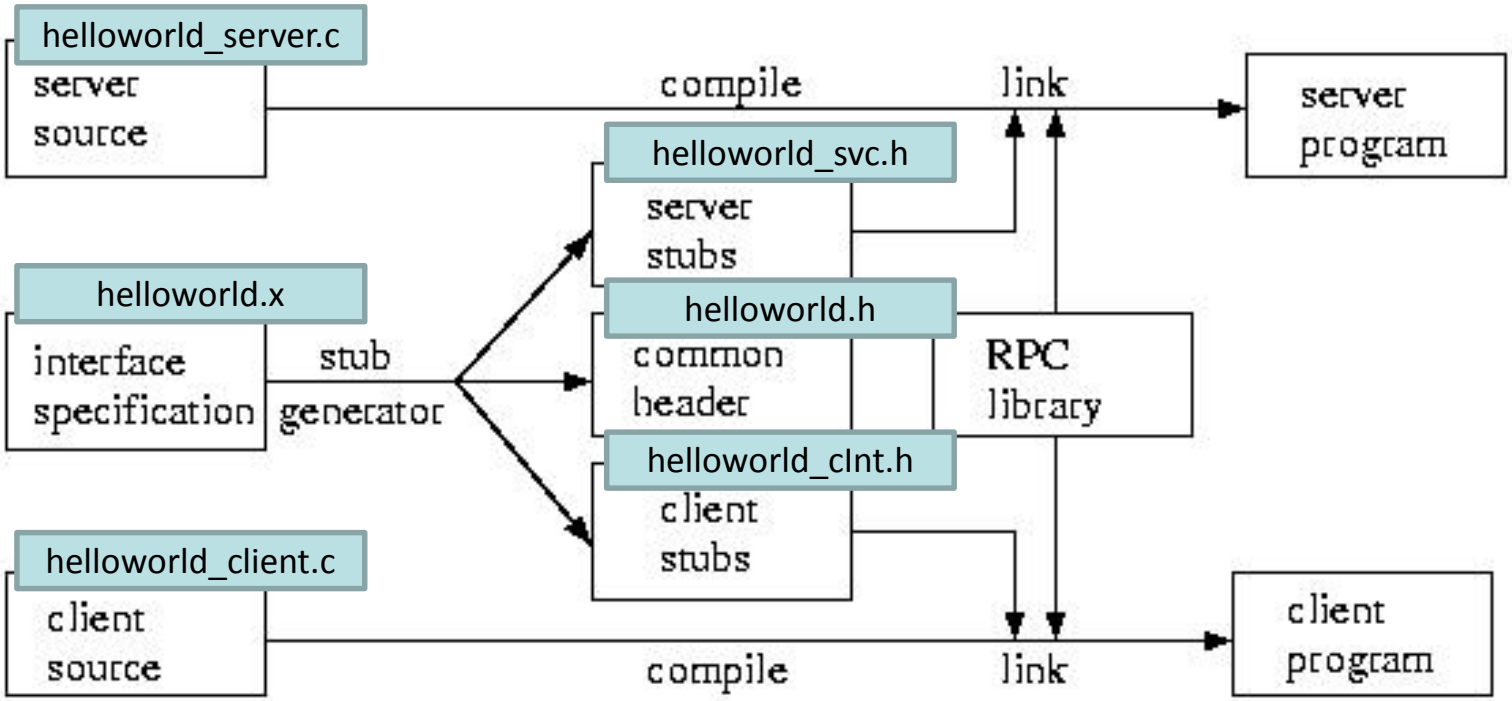
3.7.4 Entfernter Prozeduraufruf

RPC: ERSTELLUNG



3.7.4 Entfernter Prozeduraufruf

RPC-BEISPIEL: DATEIEN MIT RPCGEN



3.7.4 Entfernter Prozeduraufruf

RPC-BEISPIEL (ONC): INTERFACE SPEZIFIKATION IN RPC-QUELLCODE

helloworld.x

```
program HELLOWORLDPROG {
        version HELLOWORLDVERS {
                string HELLOWORLD(void) = 1;
        } = 1;
} = 0x30000498;
```

RPCGEN

ONC (Open Network Computing): RPC-Typ, manchmal „SUN RPC“

RPC-BEISPIEL: COMMON HEADER

helloworld.h

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#ifndef _HELLOWORLD_H_RPCGEN
#define _HELLOWORLD_H_RPCGEN

#include <rpc/rpc.h>

#ifdef __cplusplus
extern "C" {
#endif

#define HELLOWORLDPROG 0x30000498
#define HELLOWORLDVERS 1
```

3.7.4 Entfernter Prozeduraufruf

… RPC-BEISPIEL: COMMON HEADER

helloworld.h

```
#if defined(__STDC__) || defined(__cplusplus)
#define HELLOWORLD 1
extern char ** helloworld_1(void *, CLIENT *);
extern char ** helloworld_1_svc(void *, struct svc_req *);
extern int helloworldprog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define HELLOWORLD 1
extern  char ** helloworld_1();
extern  char ** helloworld_1_svc();
extern int helloworldprog_1_freeresult ();
#endif /* K&R C */

#ifdef __cplusplus
}
#endif

#endif /* !_HELLOWORLD_H_RPCGEN */
```

Funktionsdeklaration zu „Client Stub“-Funktion für Client

Funktionsdeklaration zu „Server Stub“-Funktion für Server

3.7.4 Entfernter Prozeduraufruf

RPC-BEISPIEL: CLIENT

helloworld_client.c

```
#include "helloworld.h"

void helloworldprog_1(char *host)
{
    CLIENT *clnt;
    char **result_1;
    char *helloworld_1_arg;

#ifndef  DEBUG
    clnt = clnt_create (host, HELLOWORLDPROG, HELLOWORLDVERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif
```

3.7.4 Entfernter Prozeduraufruf

… RPC-BEISPIEL: CLIENT

Aufruf „Client Stub“-Funktion

```
    result_1 = helloworld_1((void*)&helloworld_1_arg, clnt);
    if (result_1 == (char **) NULL) {
        clnt_perror (clnt, "call failed");
    }
#ifndef  DEBUG
    clnt_destroy (clnt);
#endif

    printf ("Got \"%s\" from the server.\n", *result_1);
}
```

ist praktisch ein lokaler Aufruf

3.7.4 Entfernter Prozeduraufruf

… RPC-BEISPIEL: CLIENT

```
int main (int argc, char *argv[])
{
     char *host;
     if (argc < 2) {
          printf ("usage: %s server_host\n", argv[0]);
          exit (1);
     }
     host = argv[1];
     helloworldprog_1 (host);
     exit (0);
}
```

Übersicht

BASISKONZEPTE

- 3.1 RESSOURCEN UND DIENSTE
  - 3.1.1 Prinzipien und Phasen für Verteilung
  - 3.1.2 Ressource
  - 3.1.3 Dienst
  - 3.1.4 Anbieter und Konsument
- 3.2 CLIENT UND SERVER
  - 3.2.1 Fundamentalverteilung: Client-Server
  - 3.2.2 Netzwerkebene
- 3.3 INTERAKTIONSMODELLE
- 3.4 (A)SYNCHRONITÄT
  - 3.4.1 Anfrage und Antwort
  - 3.4.2 Multiple Anfragen

EIGENSCHAFTEN

- 3.4 KOHÄRENZ UND TRANSPARENZ
  - 3.4.1 Definition
  - 3.4.2 Transparenz von Verteilungseigenschaften
  - 3.4.3 Bedeutung und Realisierbarkeit
- 3.5 SKALIERBARKEIT
  - 3.5.1 Definition und Typen
  - 3.5.2 Typ: Größe
  - 3.5.3 Typ: Geographie
  - 3.5.4 Typ: Administration
- 3.6 NEBENLÄUFIGKEITS-GRAD
  - 3.6.1 Bedeutung und Konsequenz
  - 3.6.2 Ausschlussverfahren
  - 3.6.3 Grad und Auswirkung
- 3.7 KOMMUNIKATIONS-GRAD
  - 3.7.1 Schalenmodell
  - 3.7.2 Sockets
  - 3.7.3 Nachrichten
  - 3.7.4 Entfernter Prozeduraufruf
  - 3.7.5 Entfernter Methodenaufruf
  - 3.7.6 Runtime, Dienste, Komponenten

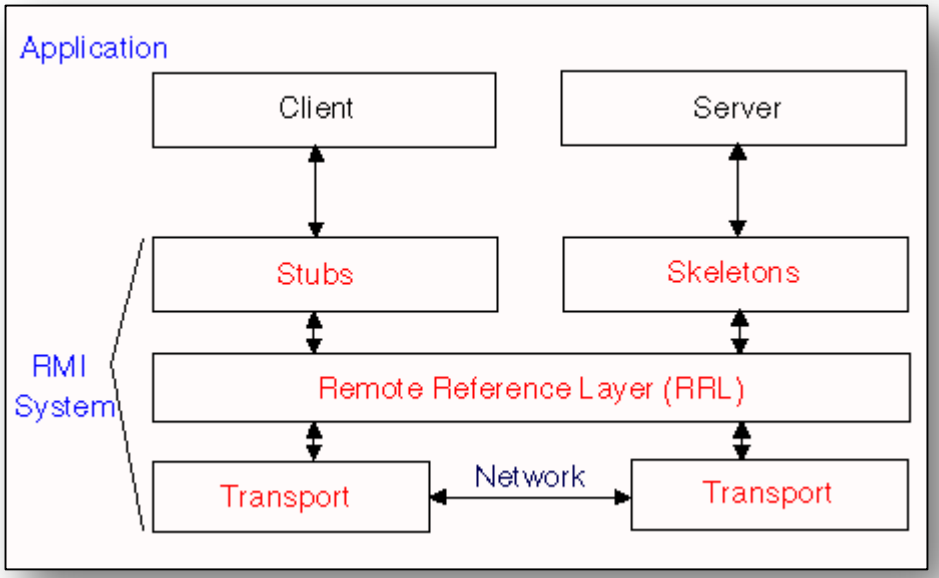
ARCHITEKTUREN

- 3.8 SOFTWARE- UND SYSTEM-ARCHITEKTUR
  - 3.8.1 Software-Architektur

3.7.5 Entfernter Methodenaufruf

ÜBERTRAGUNG VON RPC AUF OBJEKTORIENTIERUNG

- Entfernte Objekte statt entfernten Prozeduren/Funktionen
- Anwendung: Aufruf der öffentlichen Methoden eines entfernten Objekts
  - Ziel: Verwendung von „entfernten“ Methoden wie „lokalen“ Methoden
- Parameter-Marshalling wie bei RPC mit Server- und Client Stubs
- *Beispiel: JAVA RMI, DCOM*

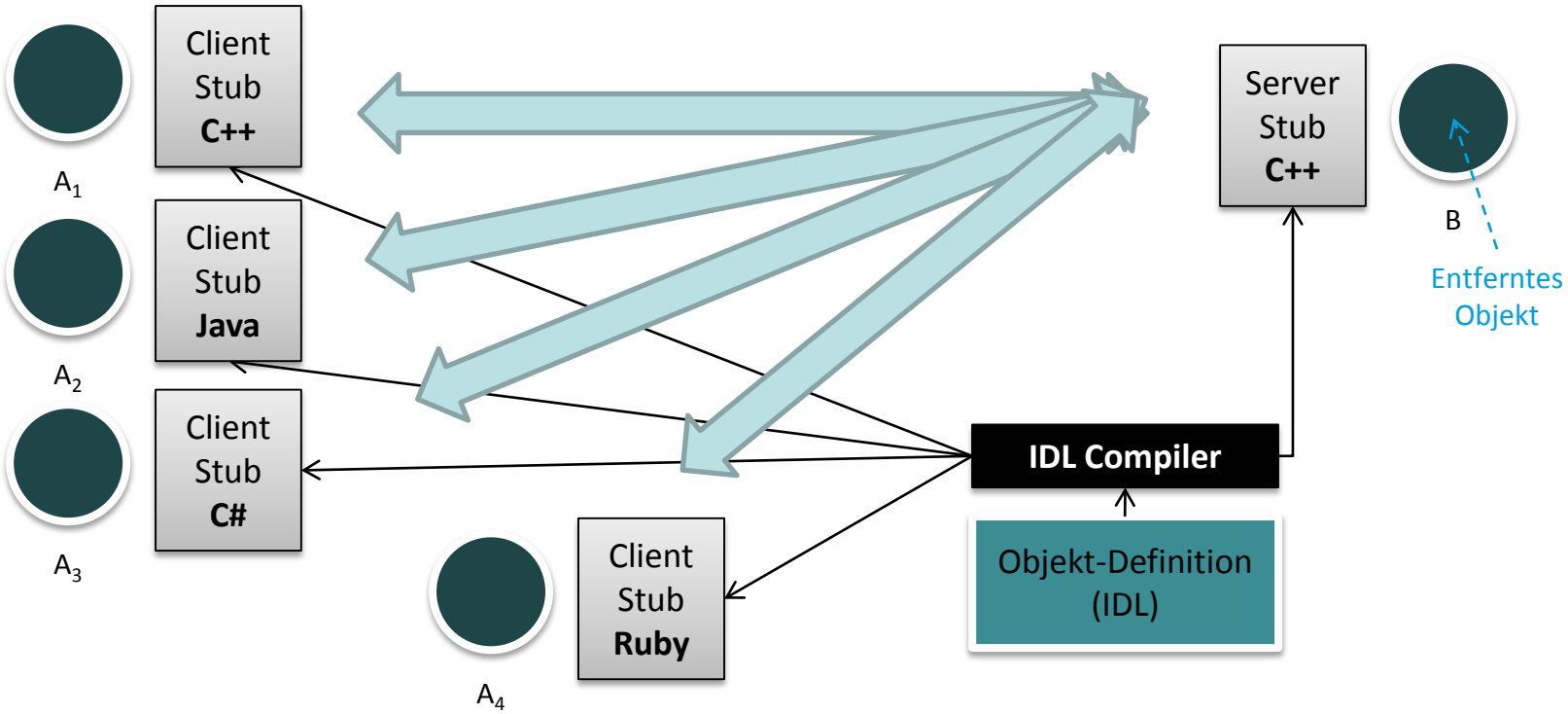


Quelle: Chris Matthews – Introduction to RMI

3.7.5 Entfernter Methodenaufruf

VORTEIL SPRACHUNABHÄNGIGKEIT

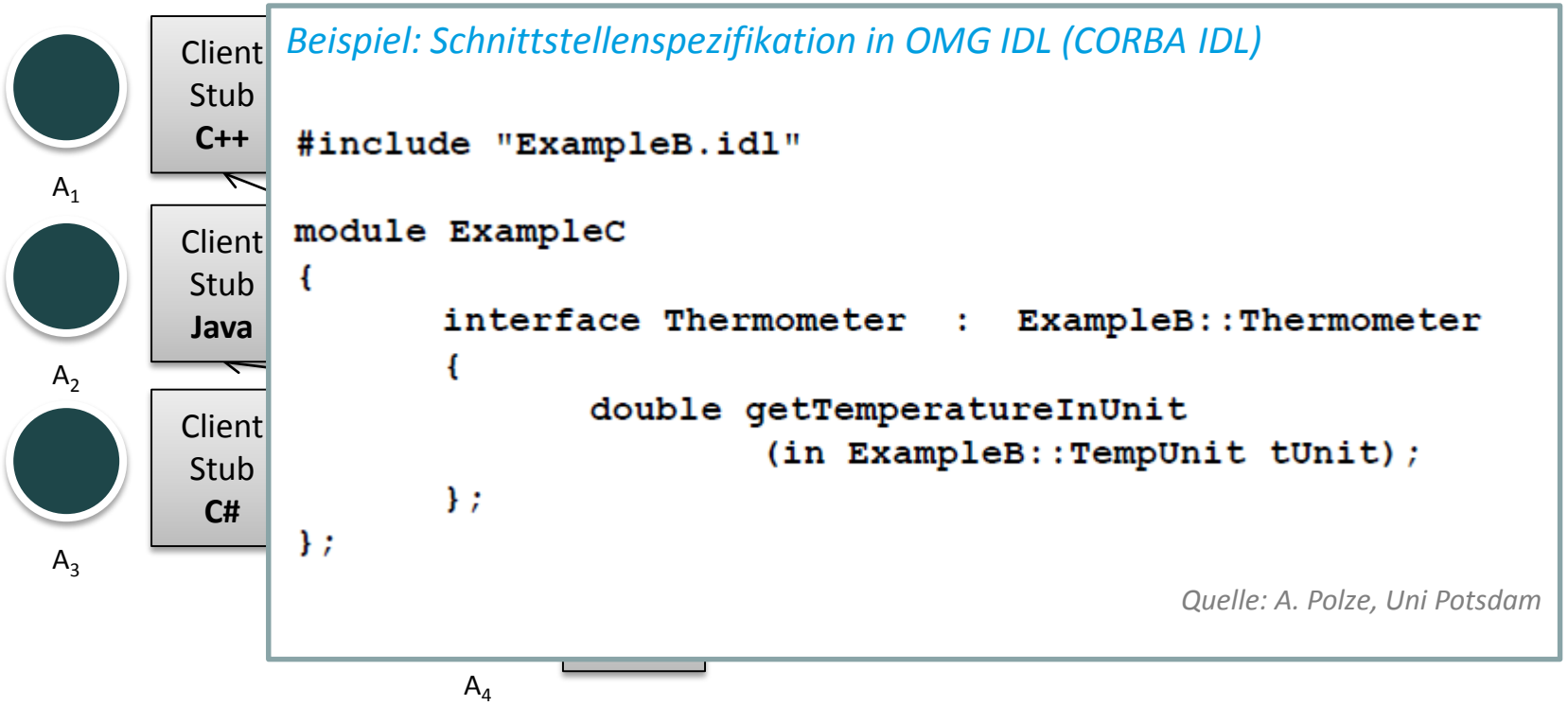
- Erster Schritt: Verwendung von XDR wie bei RPC
  - Sprachtransparenz: Sender benötigt keine Kenntnis von verwendeter Empfänger-Sprache
- Zweiter Schritt: Beschreibung des entfernten Objekts in **IDL** (Interface Definition Language)
  - *Beispiel: OMG IDL (CORBA IDL), AIDL* **OMG = Object Management Group**
- Dritter Schritt: Stub-Generierung für alle relevanten Prog.-Sprachen auf Client- und Server-Seite
- *Beispiel: CORBA*



3.7.5 Entfernter Methodenaufruf

VORTEIL SPRACHUNABHÄNGIGKEIT

- Erster Schritt: Verwendung von XDR wie bei RPC
  - Sprachtransparenz: Sender benötigt keine Kenntnis von verwendeter Empfänger-Sprache
- Zweiter Schritt: Beschreibung des entfernten Objekts in **IDL** (Interface Definition Language)
  - *Beispiel: OMG IDL (CORBA IDL), AIDL*
- Dritter Schritt: Stub-Generierung für alle relevanten Prog.-Sprachen auf Client- und Server-Seite
- *Beispiel: CORBA*

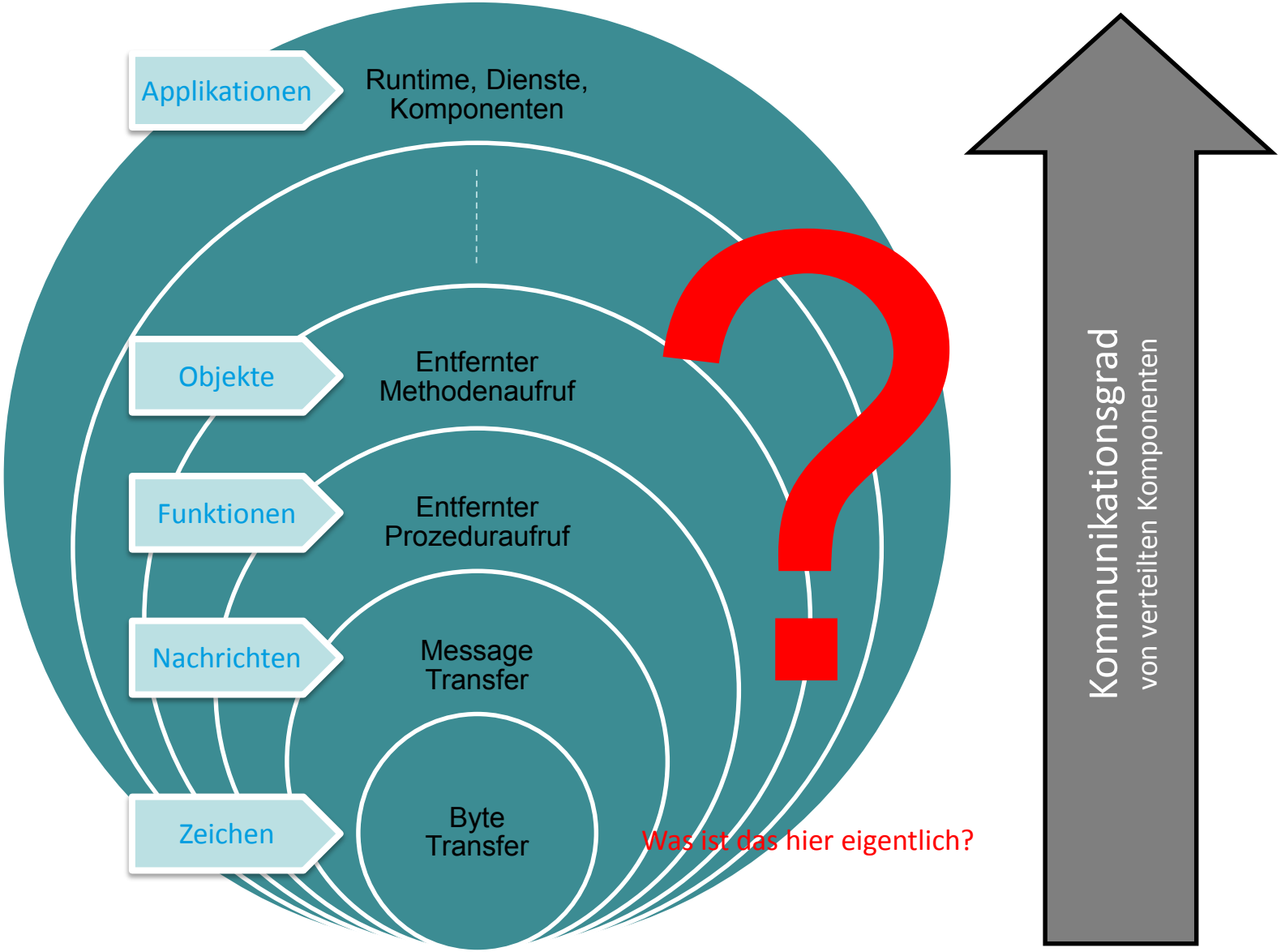


Beispiel: Schnittstellenspezifikation in OMG IDL (CORBA IDL)

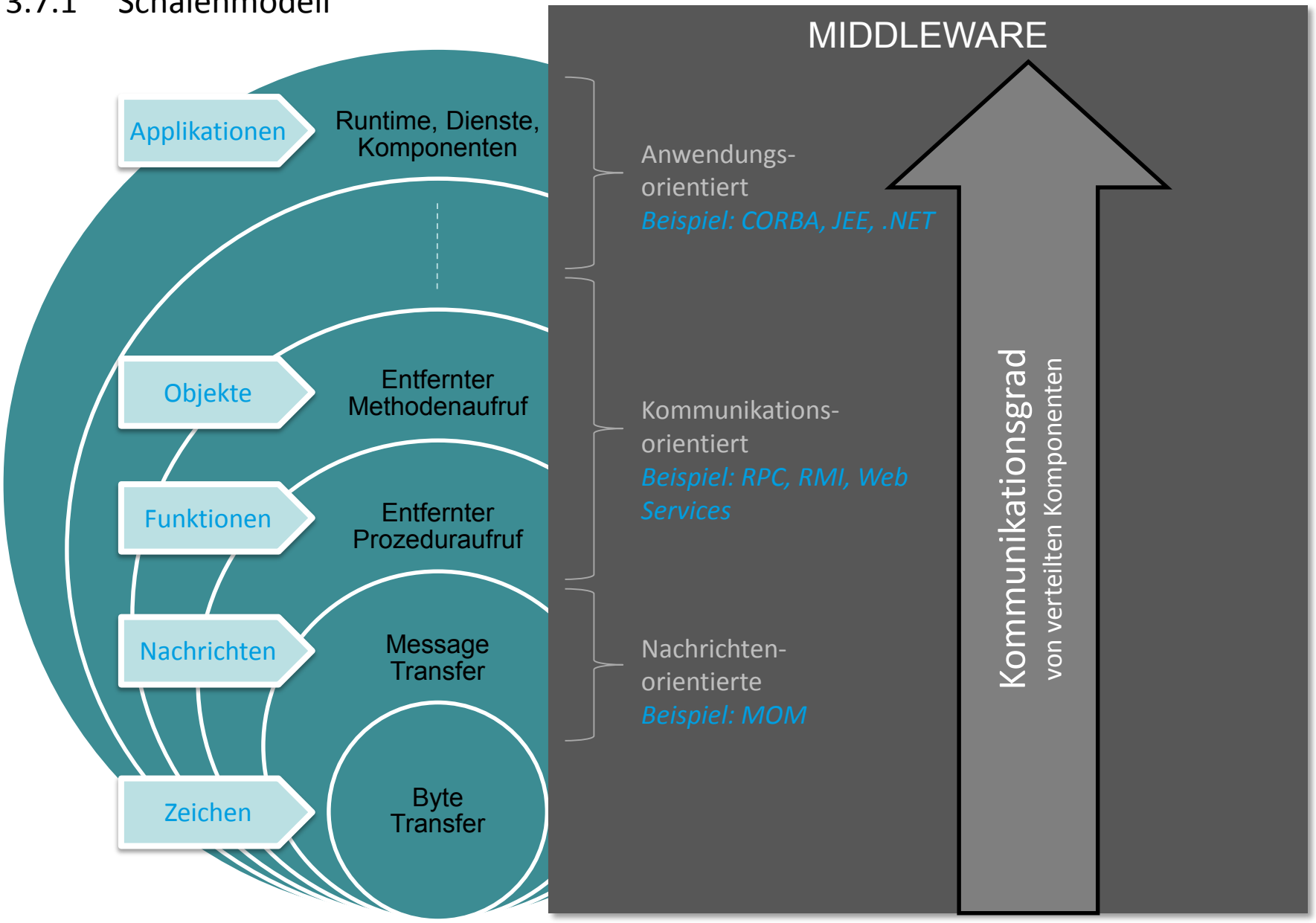
```
#include "ExampleB.idl"

module ExampleC
{
        interface Thermometer   :   ExampleB::Thermometer
        {
                double getTemperatureInUnit
                        (in ExampleB::TempUnit tUnit);
        };
};
```

Quelle: A. Polze, Uni Potsdam

3.7.1 Schalenmodell



3.7.1 Schalenmodell



3.7.6 Runtime, Dienste, Komponenten

ANWENDUNGSORIENTIERTE MIDDLEWARE

- Anwendungsneutrale Vermittlungssoftware zwischen verteilten Anwendungen
  - Vermittelt so, dass Komplexität, Plattform und Infrastruktur der Anwendungen verborgen bleibt
  - Vermittelt so, dass Netzwerk für Anwendungen transparent wird
- Baut auf kommunikationsorientierter Middleware auf (RMI, RPC, Web Services)
  - Erweiterung um
    - Laufzeitumgebung
      - Ressourcenverwaltung (Nebenläufigkeit, Verbindungsverwaltung)
      - Verfügbarkeit (Replikation, Clustering, Balancing)
      - Sicherheit (Authentifizierung, Authorisierung, Verschlüsselung, Integrität)
    - Dienste
      - Sitzungsverwaltung
      - Namensdienste
      - Transaktionsverwaltung
      - Persistierung
    - Komponentenmodell
      - Komponentenbegriff (mit Struktur und Eigenschaften)
      - Schnittstellen mit Verträgen
      - Komponentenlaufzeit
- *Beispiele: Application Server, ORBs, Plattformen (JEE, .NET, CORBA)*

3.7.6 Runtime, Dienste, Komponenten

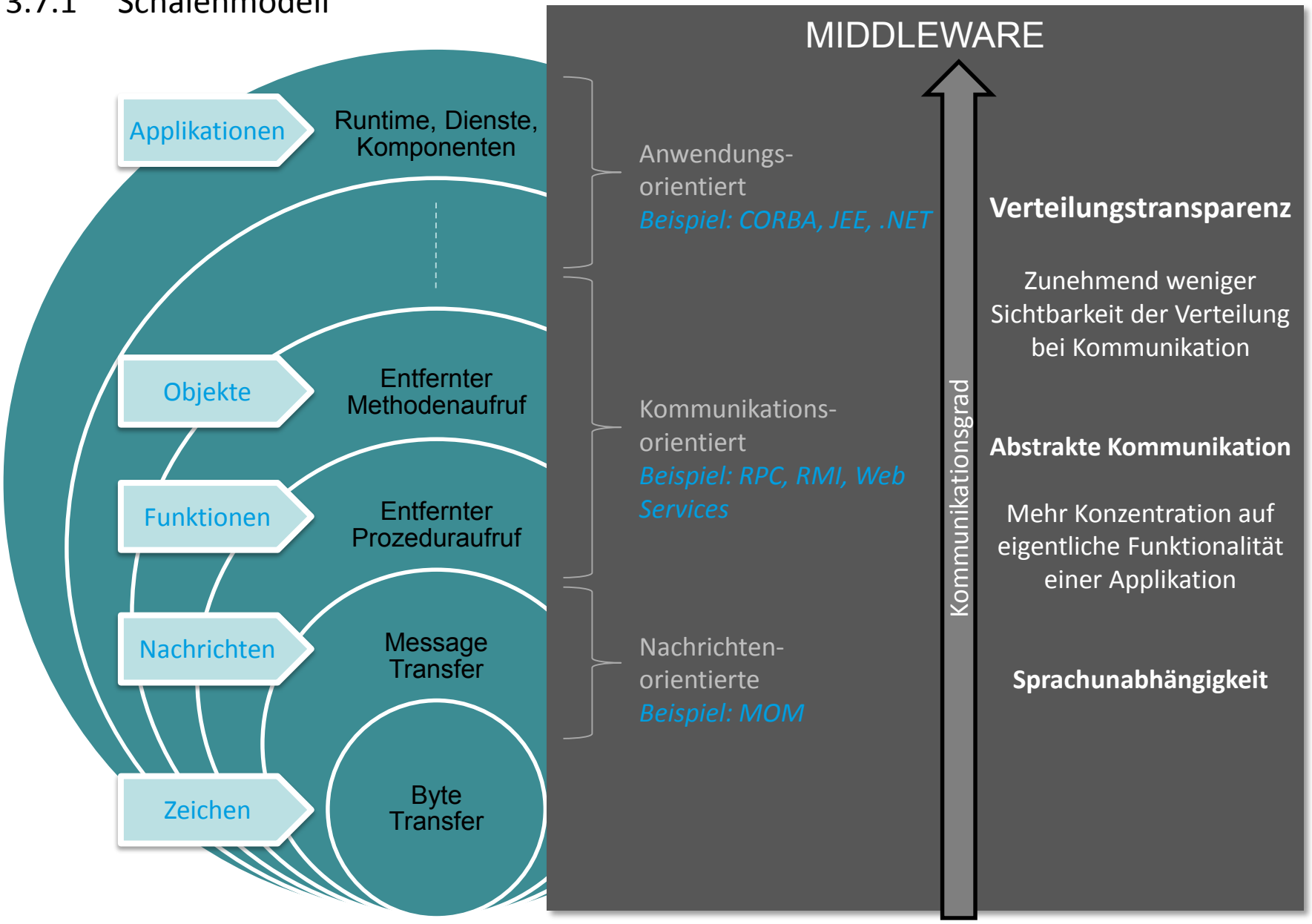
BEISPIEL: AD-HOC NETWORKING

- Neue Stufe des Kommunikationsgrads: Automatische Paarung von Client und Server in unbekannten Netzwerken (Sun: **Spontaneous Networking**)
  - (1) Verwendung von Nachschlagediensten
  - (2) Automatisches Herunterladen von Stubs
- *Beispiel: Apache River (früher Jini)*
  - Herkunft: Sun Microsystems
  - Nachschlagedienst: **Lookup Service**
  - Stub zum Herunterladen: **Service Proxy**
    - Realisierung offen: Lokal oder entfernt; Verwendung von Sockets, RPC, RMI
    - Einfachster Fall: Service Proxy ist als RMI-Stub realisiert
  - Dienstanfrage (Clients): Auffinden des Lookup Services über **Discovery Protocol**
    - Versenden einer Multicast-Nachricht im Netz: Dienstbeschreibung über Java Interface + Metadaten
  - Dienstanmeldung **JOIN** (Server) bei Lookup Service: Java Interface + Metadaten + Service Proxy
- *Anwendungsbeispiel: Auffinden von Druckern/Druckdiensten in unbekannten Netzen*

Namensdienst: schaut nach ob er Anfrage bedienen kann und liefert den registrierten Dienst nach ein gewissen Zeit zurück

Probleme:
-> Namensdienst ist weg

3.7.1 Schalenmodell



Übersicht

BASISKONZEPTE

- 3.1 RESSOURCEN UND DIENSTE
  - 3.1.1 Prinzipien und Phasen für Verteilung
  - 3.1.2 Ressource
  - 3.1.3 Dienst
  - 3.1.4 Anbieter und Konsument
- 3.2 CLIENT UND SERVER
  - 3.2.1 Fundamentalverteilung: Client-Server
  - 3.2.2 Netzwerkebene
- 3.3 INTERAKTIONSMODELLE
- 3.4 (A)SYNCHRONITÄT
  - 3.4.1 Anfrage und Antwort
  - 3.4.2 Multiple Anfragen

EIGENSCHAFTEN

- 3.4 KOHÄRENZ UND TRANSPARENZ
  - 3.4.1 Definition
  - 3.4.2 Transparenz von Verteilungseigenschaften
  - 3.4.3 Bedeutung und Realisierbarkeit
- 3.5 SKALIERBARKEIT
  - 3.5.1 Definition und Typen
  - 3.5.2 Typ: Größe
  - 3.5.3 Typ: Geographie
  - 3.5.4 Typ: Administration
- 3.6 NEBENLÄUFIGKEITS-GRAD
  - 3.6.1 Bedeutung und Konsequenz
  - 3.6.2 Ausschlussverfahren
  - 3.6.3 Grad und Auswirkung
- 3.7 KOMMUNIKATIONS-GRAD
  - 3.7.1 Schalenmodell
  - 3.7.2 Sockets
  - 3.7.3 Nachrichten
  - 3.7.4 Entfernter Prozeduraufruf
  - 3.7.5 Entfernter Methodenaufruf
  - 3.7.6 Runtime, Dienste, Komponenten

ARCHITEKTUREN

- 3.8 SOFTWARE- UND SYSTEM-ARCHITEKTUR
  - 3.8.1 Software-Architektur

Übersicht

- 3.8.2 System-Architektur
- 3.8.3 Software-Architektur → Systemarchitektur

3.9 ZENTRALISIERTE SOFTWARE-ARCHITEKTUREN

- 3.9.1 Architekturstile
- 3.9.2 Zugehörige System-Architekturen

3.10 ROLLENAUFLÖSUNG VON CLIENT-SERVER

- 3.10.1 Funktionen eines VSYS und Client-Server
- 3.10.2 Client-Server: Aufgabenverteilung
- 3.10.3 Von Client-Server zu Mehrschicht-Architekturen
- 3.10.4 Vom Server zum Service
- 3.10.5 Von der Web-Anwendung zur RIA
- 3.10.6 Von der Dienstverteilung zur Komponentenverteilung
- 3.10.7 Vertauschbare Client-/Server-Rollen
- 3.10.8 Schrittweise Aufweichung des C-/S-Prinzips

Übersicht

E I G E N S C H A F T E N

Qualitätsmerkmale

Für die Güte eines verteilten Systems

Entwurfskriterien

Bei der Architektur und Entwicklung eines verteilten Systems

- Transparenz ✓
- Skalierbarkeit ✓
- Nebenläufigkeitsgrad ✓
- Kommunikationsgrad ✓

Übersicht

K A P I T E L 3

Vom zentralen Fall übertragbare Basiskonzepte, daraus ableitbare grundlegende Eigenschaften, erste teilweise auch zentral gültige Strukturen und Architekturelemente

Übersicht

K A P I T E L 3

Vom zentralen Fall übertragbare Basiskonzepte, daraus ableitbare grundlegende Eigenschaften, erste teilweise auch zentral gültige Strukturen und Architekturelemente

3.8.1 Software-Architektur

SOFTWARE-ARCHITEKTUR: DEFINITIONEN

- „The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them“

  Len Bass et. al, Software Architecture in Practice, Addison-Wesley, 2003

- Eine Softwarearchitektur besteht aus einer Menge von Komponenten und deren Beziehungen untereinander

  Manfred Broy, TUM, 2002

- Software-Architektur ist „eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen“

  Helmut Balzert, Lehrbuch der Software-Technik, Spekturm 1996

- **Definition Software-Architektur**: Zerlegung der Gesamtfunktionalität in Komponenten mit eindeutig spezifizierten Schnittstellen und Beziehungen

3.8.1 Software-Architektur

SOFTWARE-ARCHITEKTUR: EIGENSCHAFTEN UND ZIELE

- **Abstraktionsprinzip:** Separation von nicht systemrelevanten Details und Kernfunktionalität
  - → Abstraktion der Gesamtfunktionalität → Strukturierung
- **Zerlegungsprinzip:** Zerlegung der Funktionalität erlaubt Beherrschung der Komplexität
  - → Menge von Komponenten
- **Kapselungsprinzip** (Information Hiding): Trennung von öffentlichen und privaten Eigenschaften
  - → Schnittstellen
- **Wiederverwendbarkeitsprinzip:** Ausgestaltung von komponentenbasierter Funktionalität für spätere Nutzung
  - → Komponenten sind wiederverwendbar
  - → Komponenten sind ersetzbar

3.8.2 System-Architektur

SYSTEM-ARCHITEKTUR: DEFINITON

- „Entsprechend der gewählten Entwurfsalternative wird die Zerlegung (Dekomposition) des Systems in Segmente, HW-, SW- und externe Einheiten beschrieben.“

  Spezifikation V-Modell XT

- A representation of a system, including a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.

  Software Engineering Institute, Carnegie Mellon Univerity, Pittsburgh, PA

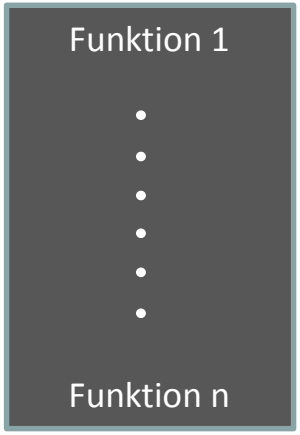
- Aber: „Eine Systemarchitektur beschreibt die Struktur des Systems durch Systemkomponenten und ihre Beziehungen untereinander.“

  Helmut Balzert, Lehrbuch der Software-Technik, Spekturm 1996

→ Entspricht im Wesentlichen der Definition „Software-Architektur“

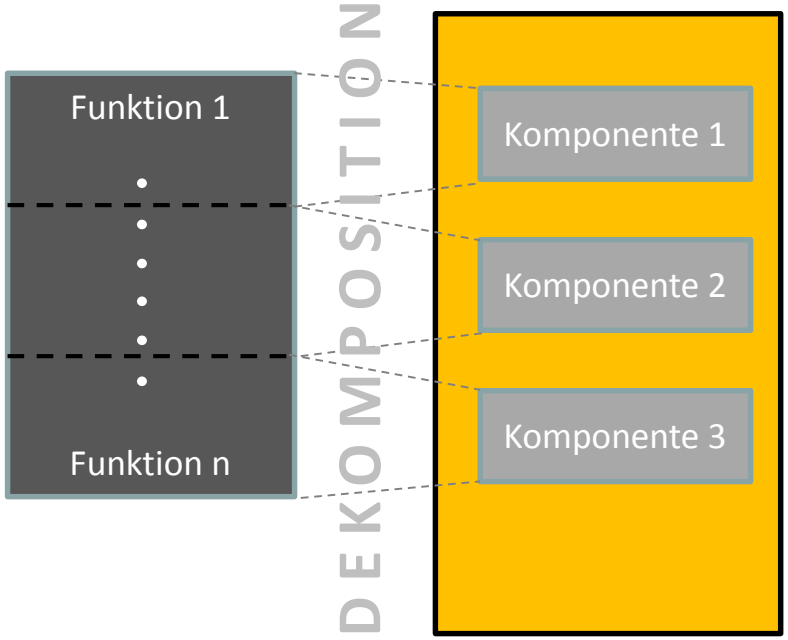
3.8.3 Software-Architektur → System-Architektur

Funktionalität



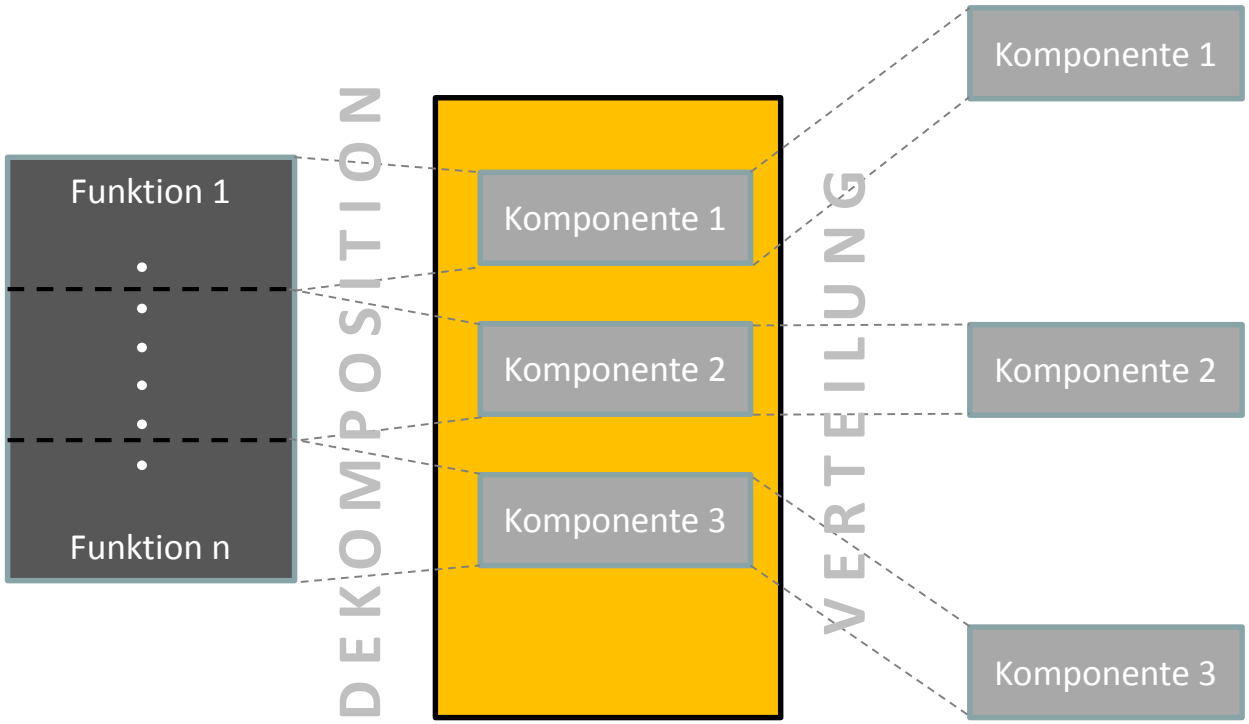
3.8.3 Software-Architektur → System-Architektur

Funktionalität Software-Architektur

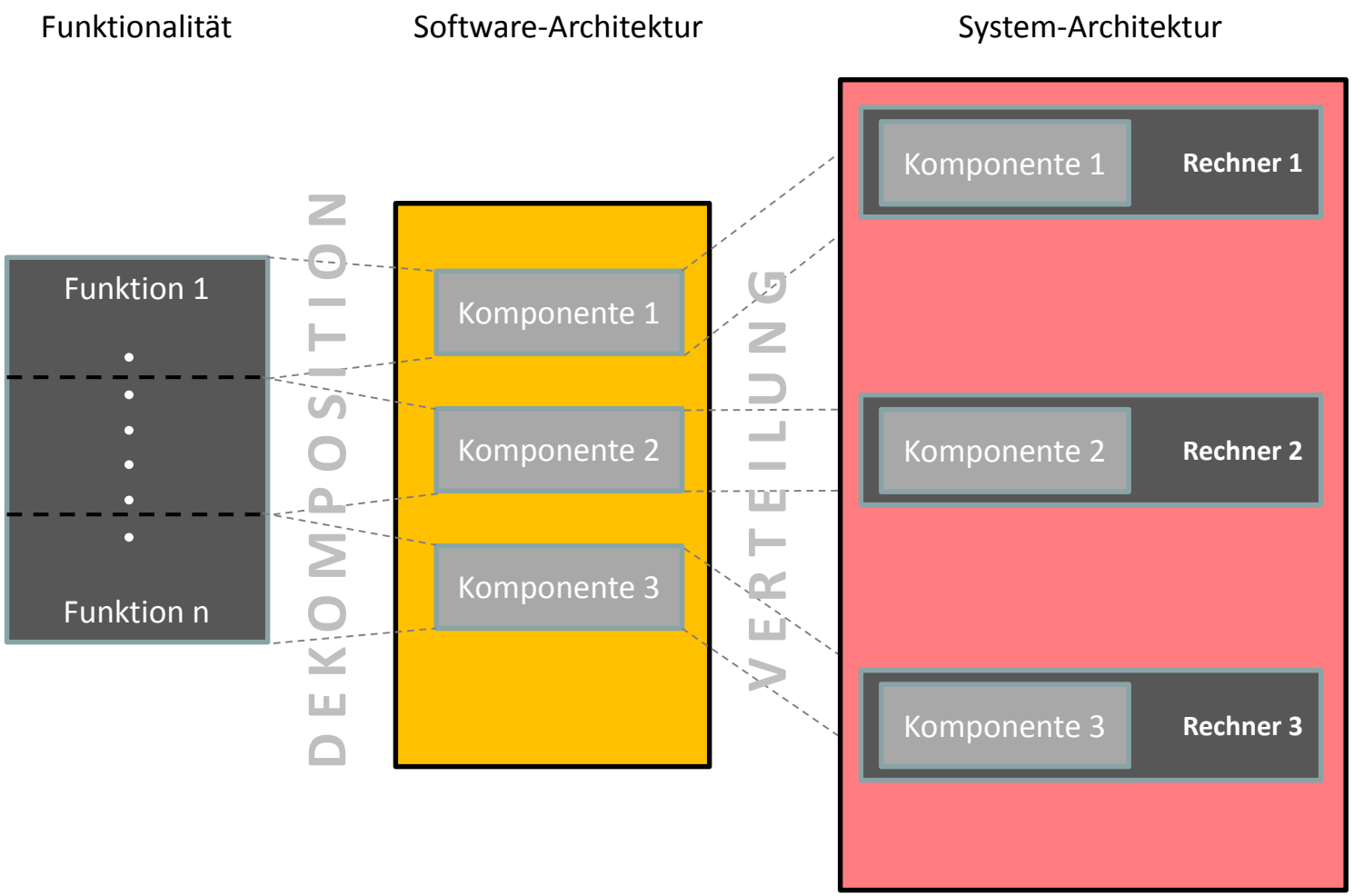


3.8.3 Software-Architektur → System-Architektur

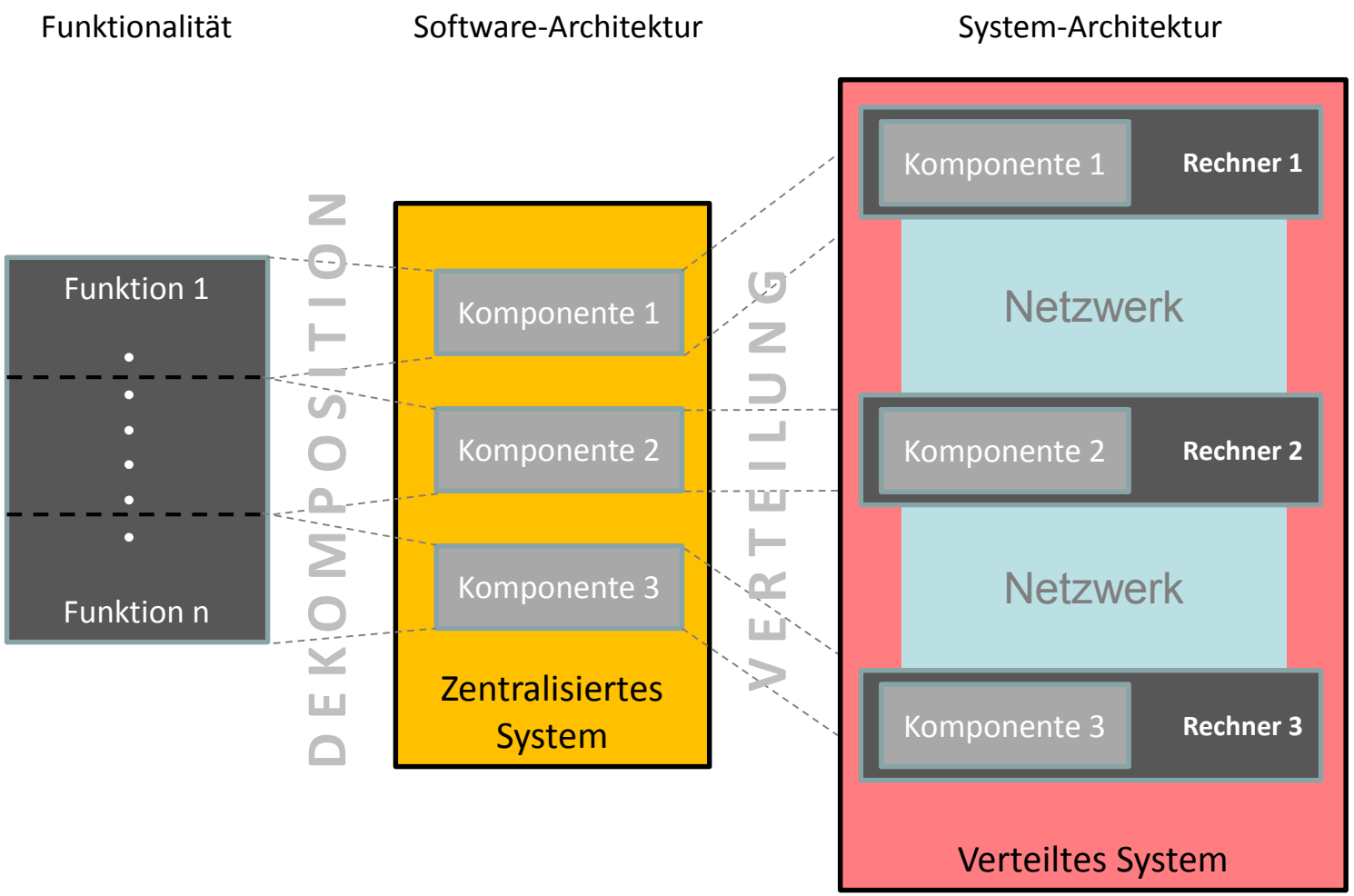
Funktionalität Software-Architektur



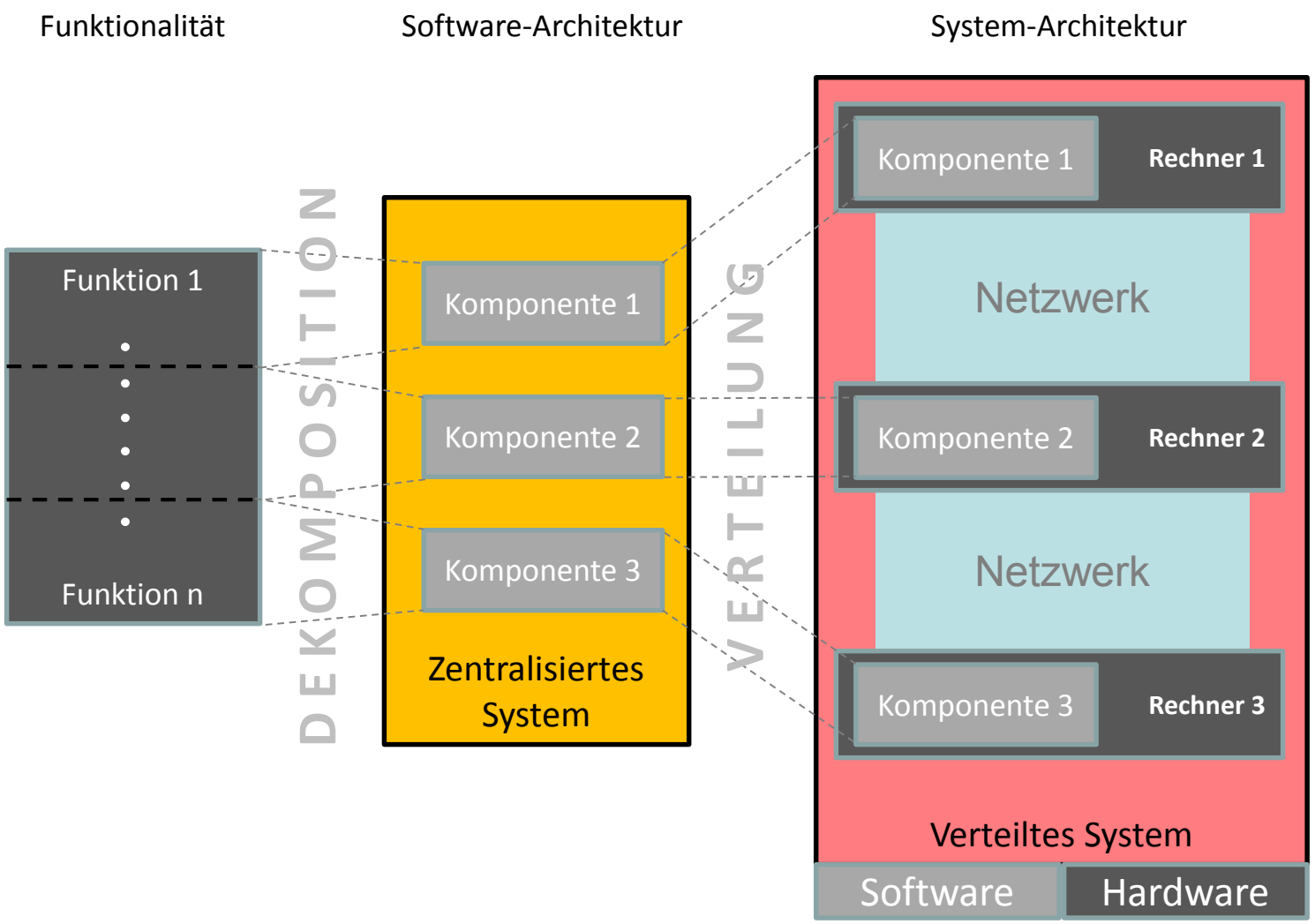
3.8.3 Software-Architektur → System-Architektur



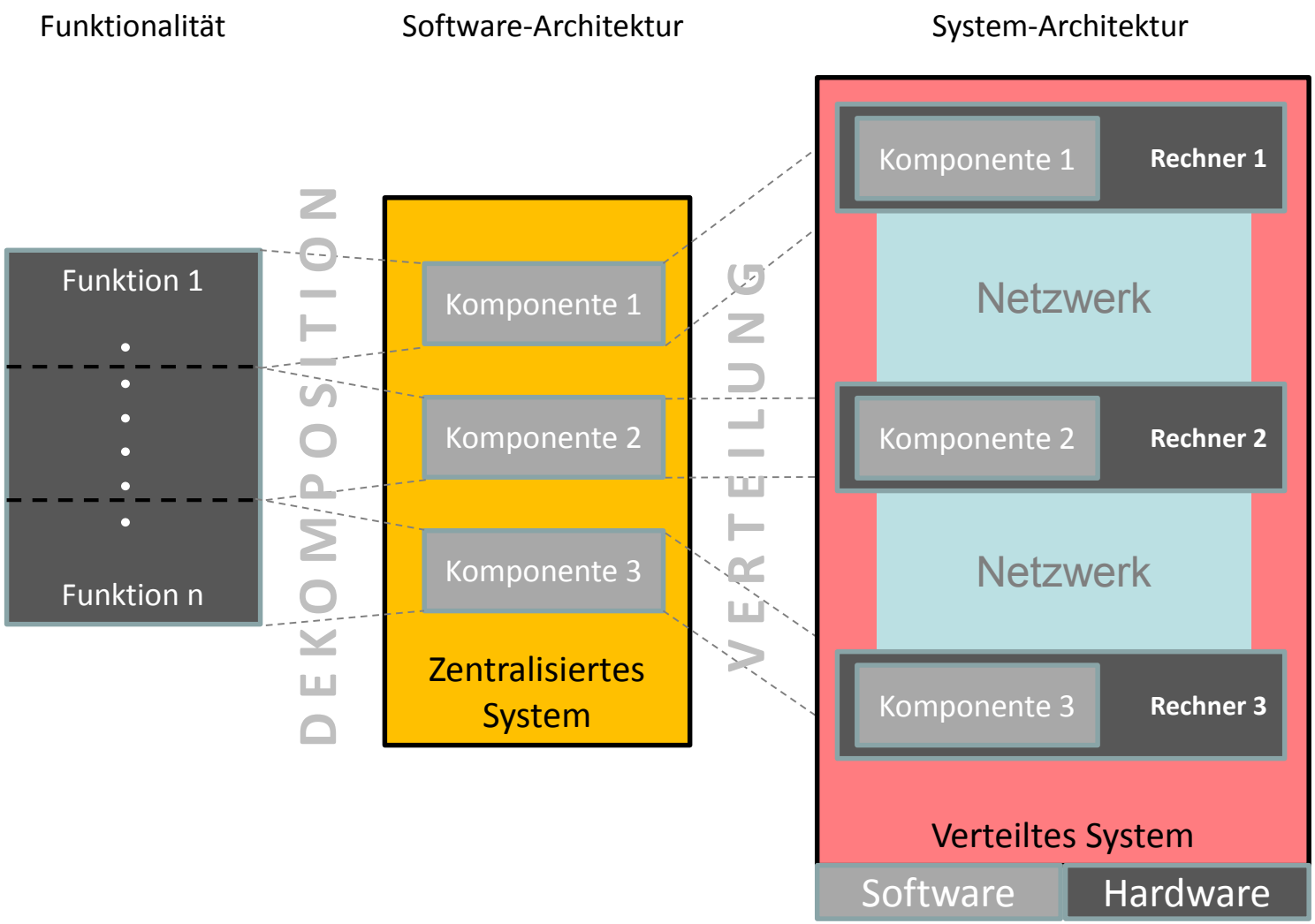
3.8.3 Software-Architektur → System-Architektur



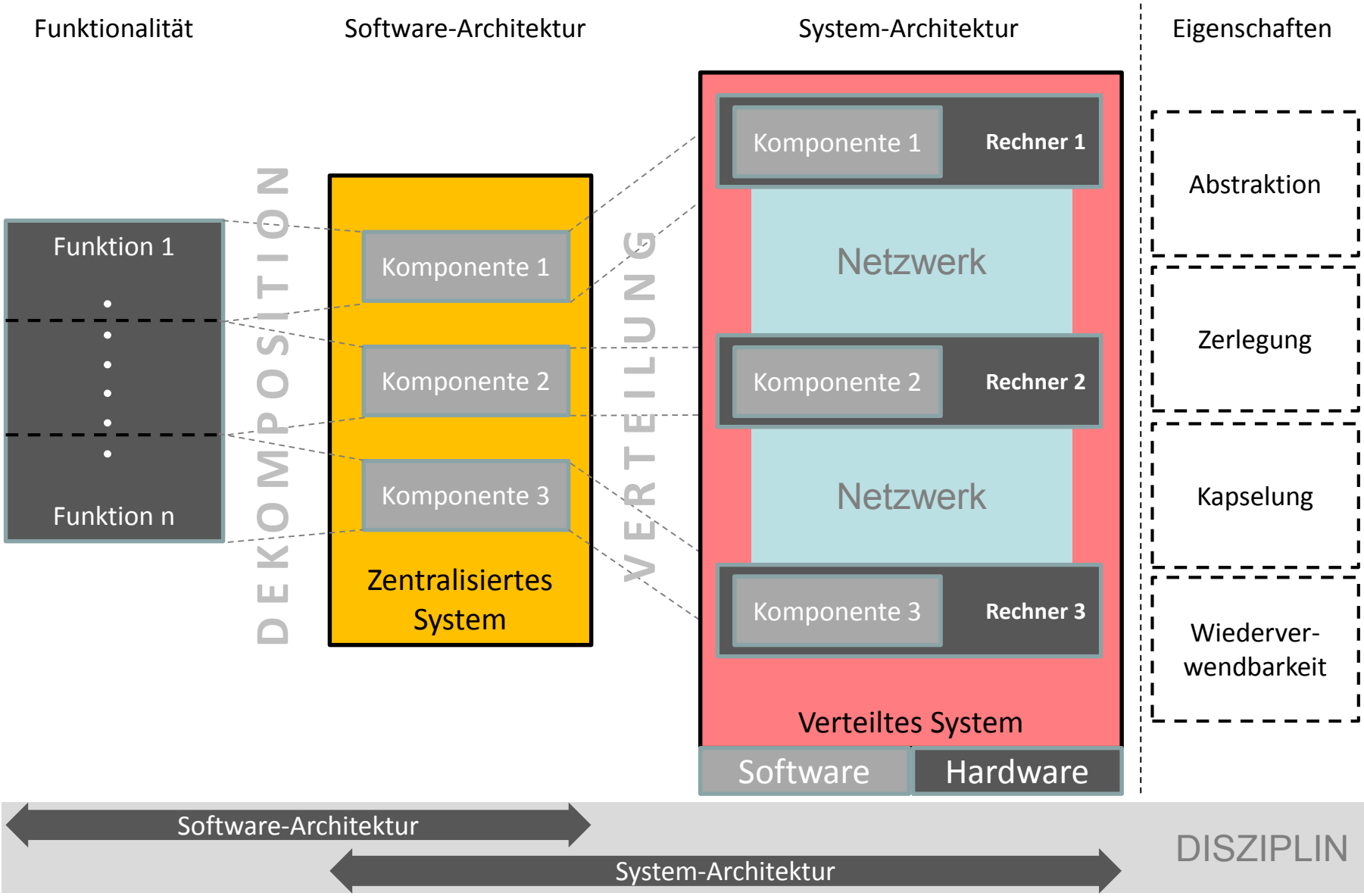
3.8.3 Software-Architektur → System-Architektur



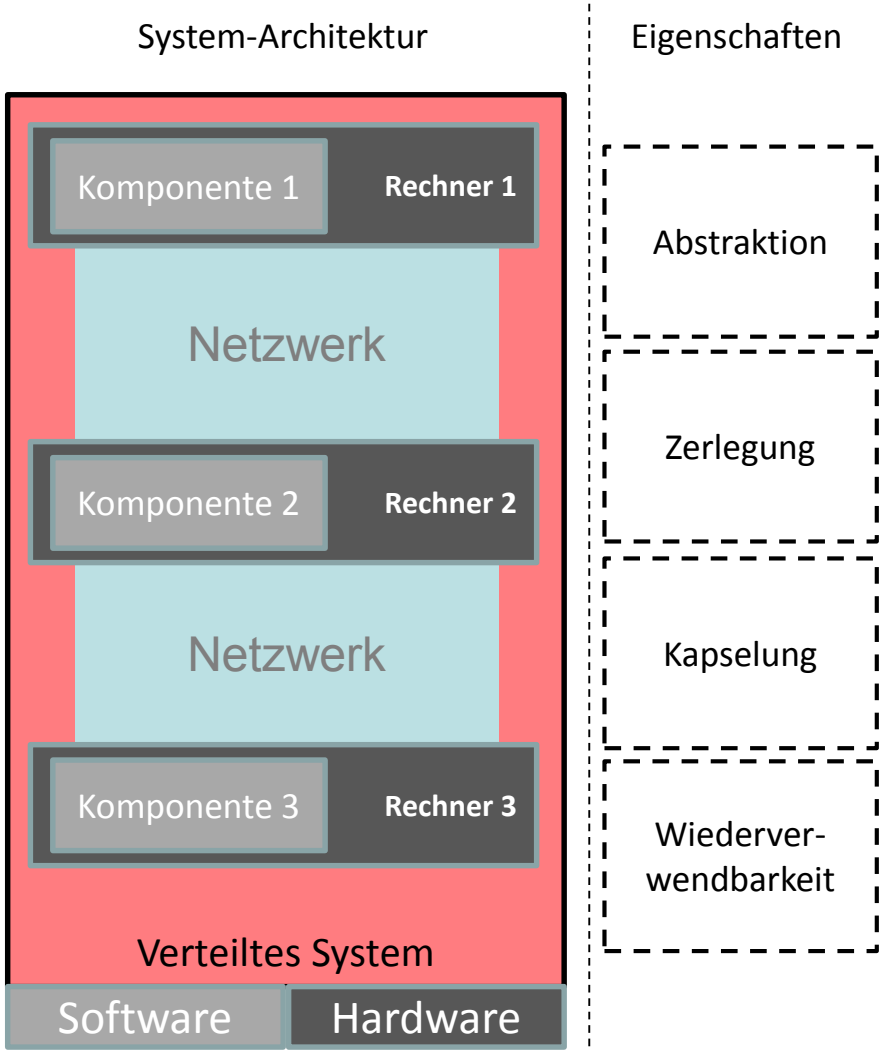
3.8.3 Software-Architektur → System-Architektur



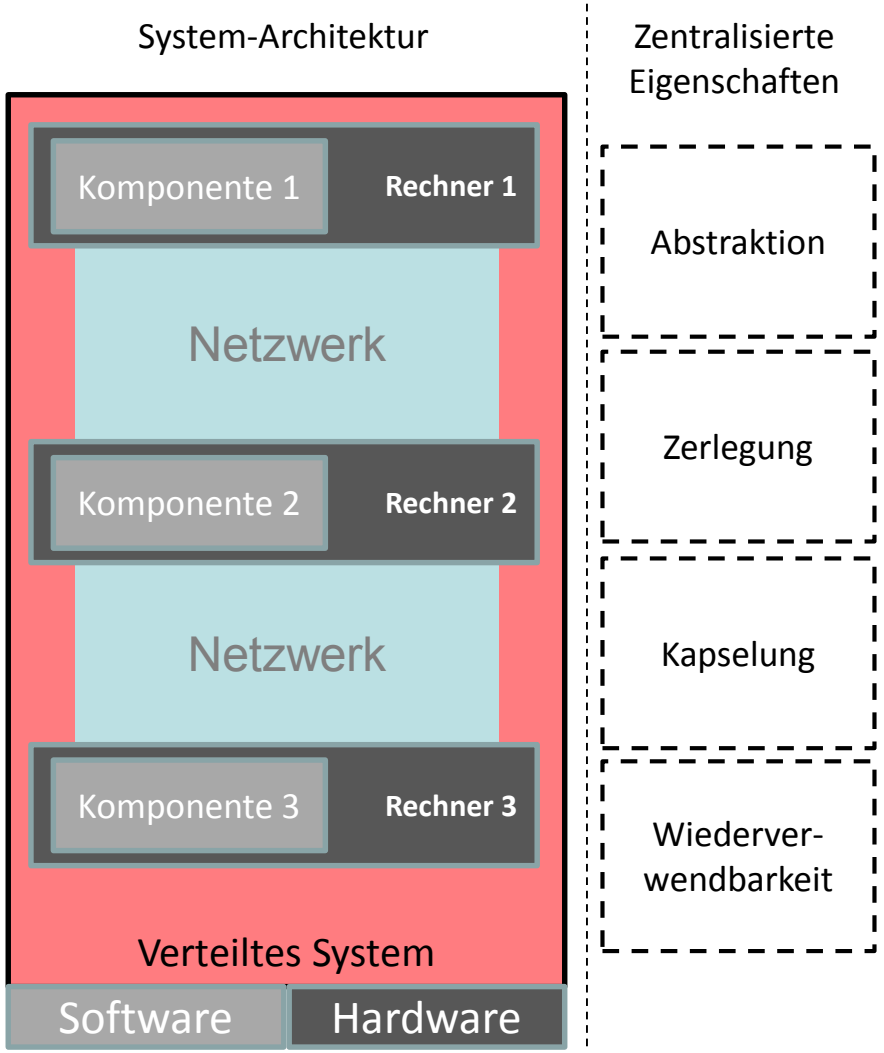
3.8.3 Software-Architektur → System-Architektur



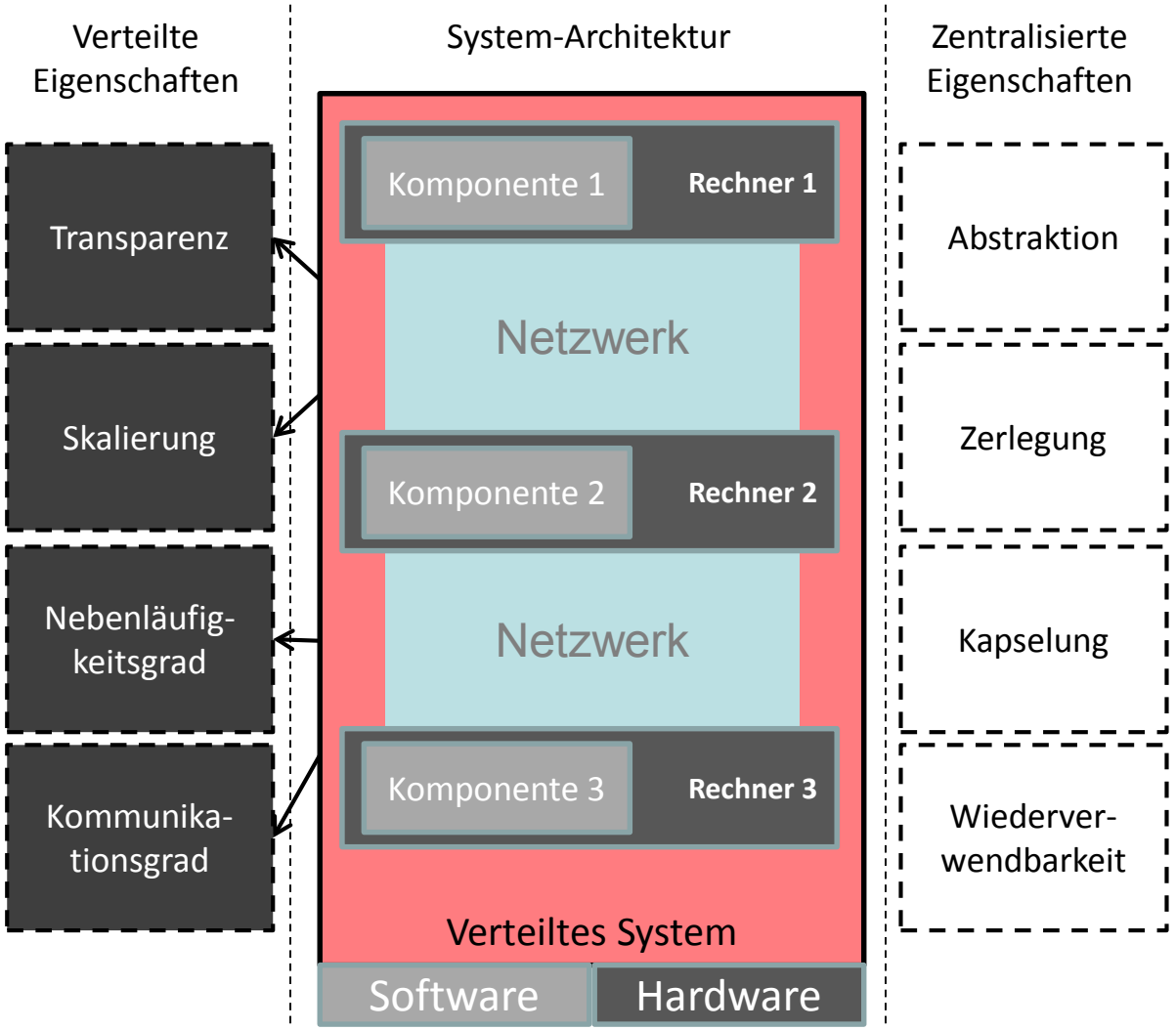
3.8.3 Software-Architektur → System-Architektur



3.8.3 Software-Architektur → System-Architektur



3.8.3 Software-Architektur → System-Architektur



Übersicht

BASISKONZEPTE

- **3.1 RESSOURCEN UND DIENSTE**
  - 3.1.1 Prinzipien und Phasen für Verteilung
  - 3.1.2 Ressource
  - 3.1.3 Dienst
  - 3.1.4 Anbieter und Konsument
- **3.2 CLIENT UND SERVER**
  - 3.2.1 Fundamentalverteilung: Client-Server
  - 3.2.2 Netzwerkebene
- **3.3 INTERAKTIONSMODELLE**
- **3.4 (A)SYNCHRONITÄT**
  - 3.4.1 Anfrage und Antwort
  - 3.4.2 Multiple Anfragen

EIGENSCHAFTEN

- **3.4 KOHÄRENZ UND TRANSPARENZ**
  - 3.4.1 Definition
  - 3.4.2 Transparenz von Verteilungseigenschaften
  - 3.4.3 Bedeutung und Realisierbarkeit
- **3.5 SKALIERBARKEIT**
  - 3.5.1 Definition und Typen
  - 3.5.2 Typ: Größe
  - 3.5.3 Typ: Geographie
  - 3.5.4 Typ: Administration
- **3.6 NEBENLÄUFIGKEITS-GRAD**
  - 3.6.1 Bedeutung und Konsequenz
  - 3.6.2 Ausschlussverfahren
  - 3.6.3 Grad und Auswirkung
- **3.7 KOMMUNIKATIONS-GRAD**
  - 3.7.1 Schalenmodell
  - 3.7.2 Sockets
  - 3.7.3 Nachrichten
  - 3.7.4 Entfernter Prozeduraufruf
  - 3.7.5 Entfernter Methodenaufruf
  - 3.7.6 Runtime, Dienste, Komponenten

ARCHITEKTUREN

- **3.8 SOFTWARE- UND SYSTEM-ARCHITEKTUR**
  - 3.8.1 Software-Architektur

Übersicht

- ✓ 3.8.2 System-Architektur
- ✓ 3.8.3 Software-Architektur → Systemarchitektur

3.9 ZENTRALISIERTE SOFTWARE-ARCHITEKTUREN

- 3.9.1 Architekturstile
- 3.9.2 Zugehörige System-Architekturen

3.10 ROLLENAUFLÖSUNG VON CLIENT-SERVER

- 3.10.1 Funktionen eines VSYS und Client-Server
- 3.10.2 Client-Server: Aufgabenverteilung
- 3.10.3 Von Client-Server zu Mehrschicht-Architekturen
- 3.10.4 Vom Server zum Service
- 3.10.5 Von der Web-Anwendung zur RIA
- 3.10.6 Von der Dienstverteilung zur Komponentenverteilung
- 3.10.7 Vertauschbare Client-/Server-Rollen
- 3.10.8 Schrittweise Aufweichung des C-/S-Prinzips

3.9.1 Architekturstile

ZENTRALISIERTE SOFTWARE-ARCHITEKTURSTILE

- **Definition Architekturstil:** Kategorien inhaltlich zusammengehöriger Architekturen
- Architekturstile

Datenzentrierte Architektur

Geschichtete Architektur

Objektorientierte Architektur

Ereignisbasierte Architektur

3.9.1 Architekturstile

… ZENTRALISIERTE SOFTWARE-ARCHITEKTURSTILE

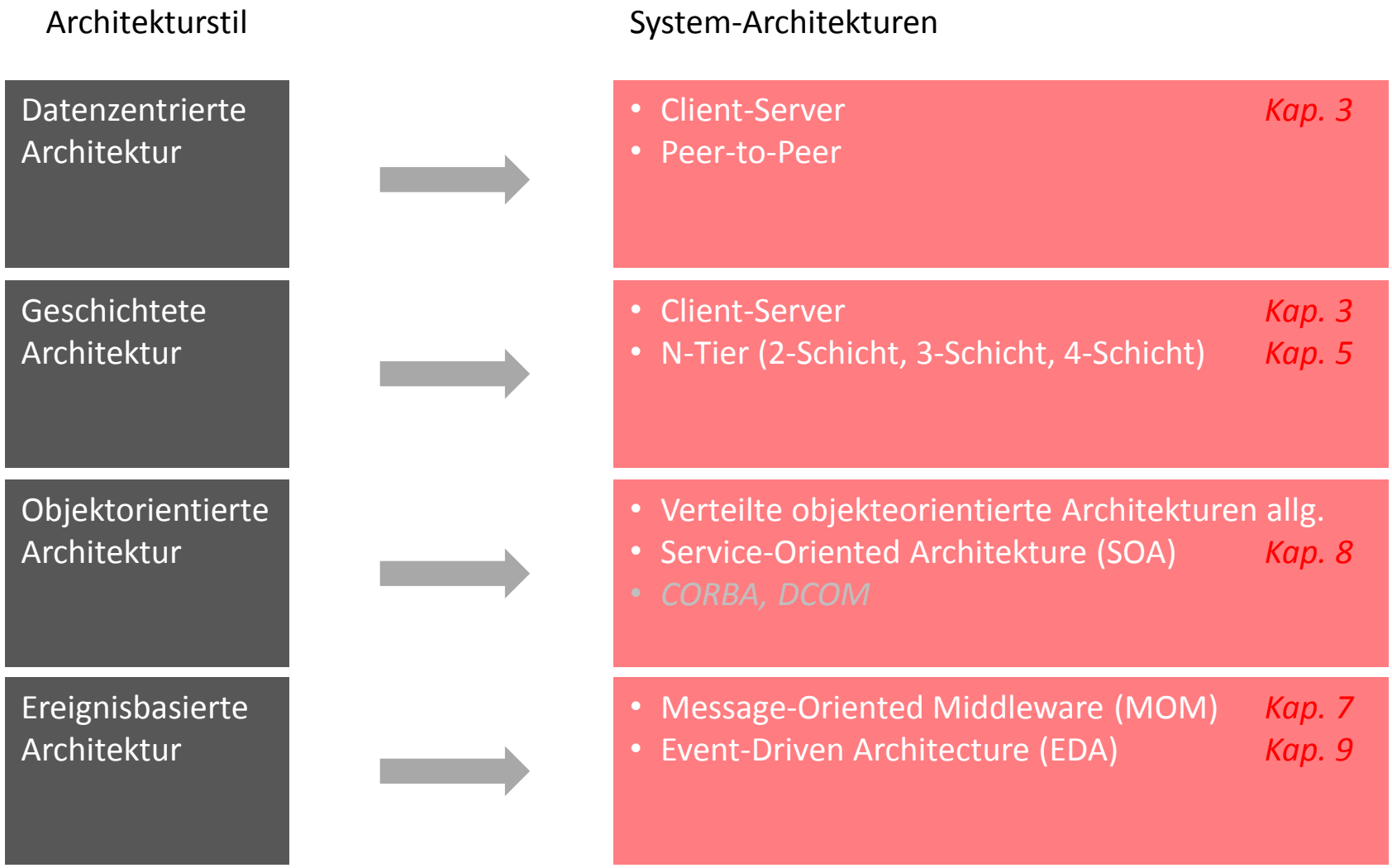
- Datenzentrierte Architektur
- Geschichtete Architektur
- Objektorientierte Architektur
- Ereignisbasierte Architektur

3.9.1 Architekturstile

Datenzentrierte Architektur	• Komponenten kommunizieren über gemeinsamen Datenspeicher (Repository) miteinander • Zeitliche Entkopplung der Komponenten voneinander
Geschichtete Architektur	• Schichtweise Anordnung der Komponenten • Höhere Schicht → komplexere Dienste • Komponente der Schicht S(k+1) darf nur Komponenten der Schicht S(k) verwenden • Anforderungen von oben nach unten, Antworten von unten nach oben
Objektorientierte Architektur	• Objekte entsprechen Komponenten • Interaktion in Form von Methodenaufrufen • Häufig Kombination von Schichten- und objektorientierter Architektur ○ Objekte sind Schichten zugeordnet, Aufrufe nur innerhalb derselben oder in nächstniedrigerer Schicht.
Ereignisbasierte Architektur	• Indirekte Kommunikation durch Ereignissen (Middleware, Ereignisbus) • Beispiel: Publish/Subscribe • Kombination mit datenzentrierter Architektur → gemeinsame Datenräume

3.9.2 Zugehörige System-Architekturen

ZUORDNUNG ARCHITEKTURSTIL → SYSTEM-ARCHITEKTUREN



Übersicht

- ✓ 3.8.2 System-Architektur
- ✓ 3.8.3 Software-Architektur → Systemarchitektur

✓ 3.9 ZENTRALISIERTE SOFTWARE-ARCHITEKTUREN

- ✓ 3.9.1 Architekturstile
- ✓ 3.9.2 Zugehörige System-Architekturen

3.10 ROLLENAUFLÖSUNG VON CLIENT-SERVER

- 3.10.1 Funktionen eines VSYS und Client-Server
- 3.10.2 Client-Server: Aufgabenverteilung
- 3.10.3 Von Client-Server zu Mehrschicht-Architekturen
- 3.10.4 Vom Server zum Service
- 3.10.5 Von der Web-Anwendung zur RIA
- 3.10.6 Von der Dienstverteilung zur Komponentenverteilung
- 3.10.7 Vertauschbare Client-/Server-Rollen
- 3.10.8 Schrittweise Aufweichung des C-/S-Prinzips