HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT UND GESTALTUNG

Verteilte Systeme
Vorlesung 03

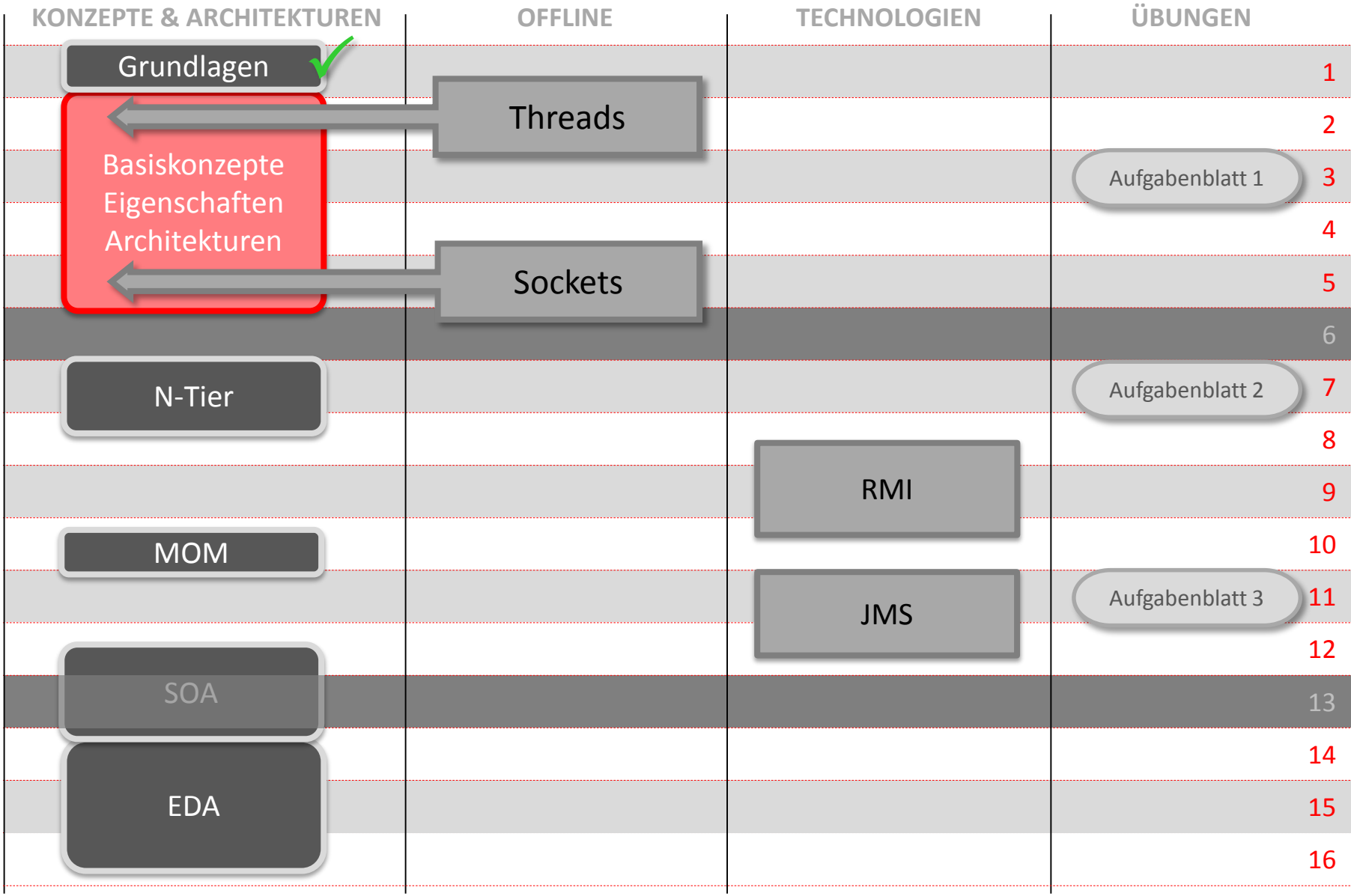
29. März 2017

Kapitel 3: Basiskonzepte, Eigenschaften, Architekturen

Prof. Dr. Rainer Mueller

SS 2017

Vorlesung: Übersicht



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

3.5.1 Definition und Typen

DEFINITION

- **Skalierbares System**: Existiert in beliebiger Ausdehnung ohne Anpassungsnotwendigkeit für Sprachelemente wie Algorithmen, Protokolle, etc.
  (Art der Ausdehnung ergibt sich durch die Typen der Skalierbarkeit)

BEDEUTUNG

- Skalierbarkeit ist wichtiges Gütemaß für verteilte Systeme (auch Entwurfsziel)
- *Beispiel Web: Gut skalierendes verteiltes System ohne zentrale Kontrolle*

TYPEN DER SKALIERBARKEIT

- Mögliche Typen
  - Größe: Erweiterung durch Komponenten oder Teilnehmer
  - Geographie: Erweiterung durch Standorte (in Institutionen, Städten, Ländern, Kontinenten)
  - Administration: Erweiterung durch administrative Domänen
- Ausdehnung ohne Anpassungsnotwendigkeit (vgl. Definition) relativ zu einem Typ
- Problemquellen je nach Typ unterschiedlich

  → Vorgehensweise oder Lösungsstrategien je nach Typ unterschiedlich

3.5.2 Typ: Größe

ZENTRALISIERUNG ALS GRUNDPROBLEM

- Zentralisierte Elemente in verteilten Systemen reduzieren Größenskalierbarkeit
  - Probleme bei zentralen Daten: Hoher lokaler Speicherbedarf, lange Bearbeitungsdauer (**Antwortzeiten)** bei Anfragen, lokaler Flaschenhals bei Kommunikation
  - Probleme bei zentralen Diensten: Überlastung bei gleichzeitigem Zugriff (CCU: concurrent user)
  - Probleme bei zentralen Funktionalitäten: Fehlendes Wissen über Gesamtsystem

LÖSUNG UND UMSETZUNG

- Lösung für reduzierte Größenskalierbarkeit: Dezentralisierung der zentralen Elemente
  - **Partitionierung**
    - Prinzip: Aufteilung der Ressourcen auf verschiedene Rechner
      → Aufteilung der Anfragen zu den Ressourcen
    - Vorgehensweise: Physische Bearbeitung aller Anfragen durch Einzelrechner
      → Weiterleitung der Anfragen an andere Rechner je nach Ressource
  - **Replikation**
    - Prinzip: Replikation aller Ressourcen auf alle Rechner
    - Vorgehensweise: Jeder Rechner kann jede Anfrage beantworten
      - Verteilung der Anfragen kann durch eine Zentralinstanz erfolgen
        - *Beispiel: **SLB (Server Load Balancing)***
          - *Zentralinstanz ist **Lastenverteiler** (Load Balancer)*
          - *Varianten: NAT-basiertes SLB, Flat SLB, DNS-basiertes SLB*

3.5.2 Typ: Größe

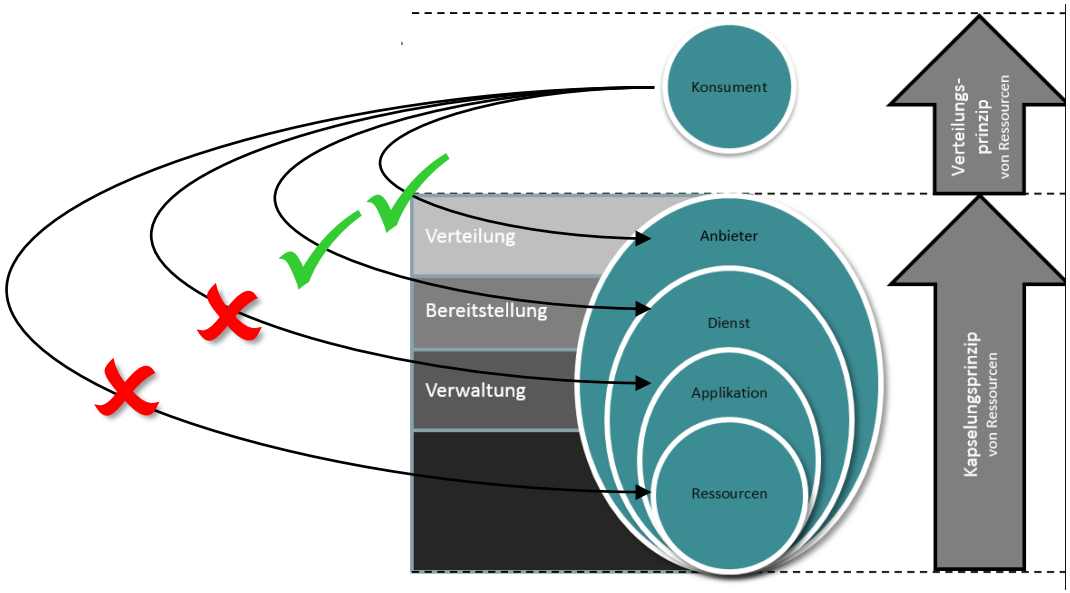
… LÖSUNG

- Umsetzung von „Partitionierung“ und „Replikation“ bei zentralen Elementen
  - Daten
    - Replikation: Redundante Speicherung von Daten
      - *Beispiel Web: Caching im Web-Browser für Read-only Daten*
    - Partitionierung: Partitionierung der Daten auf verschiedenen Rechnern
      → Lastverteilung und Datenspeicherung nahe am Zugriffsort (**Datenlokalität**)
      - *Beispiel Web*
        - *Weiterleitung einer Anfrage zu bestimmter Web-Seite nach Auflösung der IP-Adresse an passenden Server*
        - *Anfrage nach gleicher Web-Seite immer an gleichen Server*
        - *Nicht jeder Server kann jede Anfrage beantworten*
  - Dienste
    - Replikation: Mehrfachangebot von Diensten (beinhaltet idR Daten-Replikation)
      - *Beispiel Web: Web-Server (interne Verteilung der Anfragen od. mehrere DNS-Einträge)*
    - Partitionierung: Dienstaufteilung auf verschiedene Rechner (beinhaltet idR Daten-Verteilung)
      - *Beispiel Web: DNS (Dienst 1: .de → Dienst 2: htwg-konstanz.de → Dienst 3: in.htwg-konstanz)*
        - *Partitionierung des DNS-Netzes in Top-Level-Domains (Beispiel: .de, .edu, .com)*
        - *Vielstufige Partitionierung mit Root-Server an der Wurzel*
        - *Replikation durch Weiterleitung bekannter Adressen an andere Server*
          → Ermöglicht direkte Beantwortung von DNS-Anfragen für diese Adressen

3.5.2 Typ: Größe

… LÖSUNG

- … Umsetzung bei zentralen Elementen
  - Funktionalitäten
    - Replikation: Keine Lösung, da replizierte Funktionen i.A. nicht adressierbar oder auffindbar



    - Partitionierung: Einzelne Funktionskomponenten auf verschiedenen Rechnern haben nur eingeschränkte Sicht

      → Nur wenig bis kein neues Wissen bei Systemwachstum erforderlich

      - *Beispiel: Routing-Algorithmen*
        - *Zentrale Verfahren kennen gesamtes Netz, dezentrale Verfahren kennen nur Nachbarentfernung*

3.5.3 Typ: Geographie

LATENZ ODER NACHRICHTENREPLIKATION ALS GRUNDPROBLEM

- Latenz: Verlängerung der Übertragungswege erhöht Kommunikationsdauer
- Nachrichtenreplikation: Multicast oder Broadcast überflutet immer mehr Teilnehmer und überlastet Gesamtsystem → Erhöht Kommunikationsdauer indirekt

LÖSUNG: VORVERARBEITUNG UND ASYNCHRONITÄT GEGEN KOMMUNIKATIONSOVERHEAD

- Vorverarbeitung
  - Gemeinsamer Versand mehrerer Anfragen
  - Kein Versand inkorrekter Anfragen → Validierung vor Versand
    - *Beispiel: Syntaxcheck*
  - Nachrichtenreplikation nur im lokalen Umfeld
    - *Beispiel: Applikationsbasierte Reduzierung von Broadcast oder Beschränkung auf LANs*
- Ausnutzen von Wartezeiten: Asynchrone Kommunikation reduziert Wartezeiten auf Antworten
  - Wartezeit sinnvoll nutzbar?

3.5.4 Typ: Administration

SICHERHEITSRICHTLINIEN ALS GRUNDPROBLEM

- Administrative Domäne implementiert Vertrauensbereich (Trust Domain): Implementierung von Sicherheitsrichtlinien gegen externe Angriffe
  - Intern: Uneingeschränkte Kommunikation und Sichtbarkeit des gesamten Netzes
  - Extern: Sichtbarkeit und Erreichbarkeit weniger Rechner

LÖSUNG: GRADUELLE AUSNUTZUNG DER SICHERHEITSRICHTLINIEN FÜR MEHR SKALIERBARKEIT

- Beispiel Firewall
  - Keine Firewalls: Alle internen Teilnehmer von außen sichtbar und erreichbar
    → Domänengrenze hat kein Einfluss auf Skalierbarkeit
  - Öffnung der Firewall für dedizierte Anwendungen/Ports
    - *Beispiel: HTTP-Tunneling (CGI-SKripte od. Servlets erforderlich)*
    → Einfluss der Domänengrenze überwindbar
  - Umgehen einer geschlossenen Firewall durch IP-Hole-Punching
    - Ausgangslage: Einer oder beide Komponenten hinter einer Firewall
    - Lösung: Aufbau einer Verbindung beider Komponenten zu einem Mittlerknoten ausserhalb der Domänen (Firewalls)
    → Kein Einfluss der Domänengrenze auf Skalierbarkeit
    → Keine Veränderung der Firewall-Einstellungen erforderlich

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

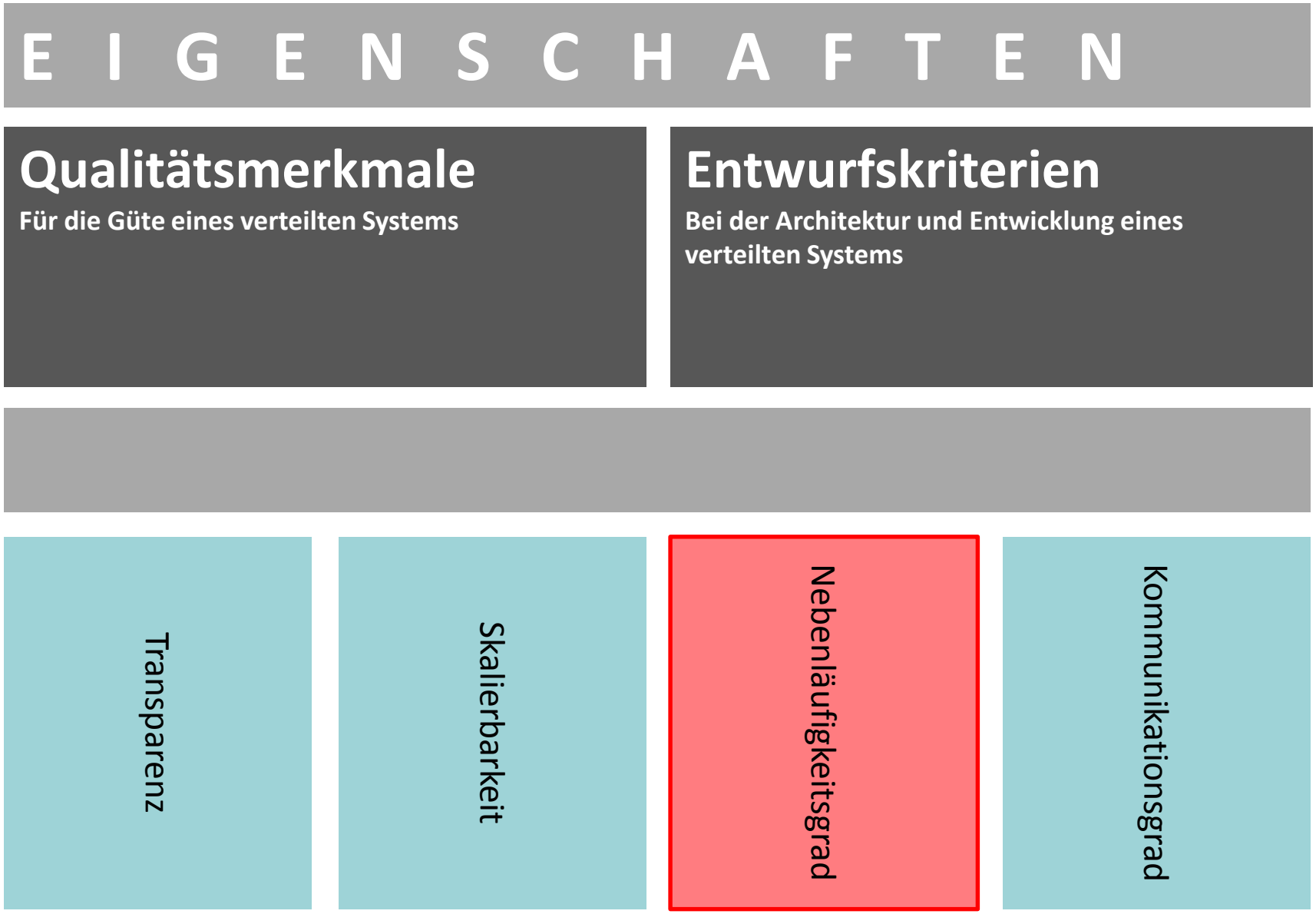
EIGENSCHAFTEN

- 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


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

3.6.1 Bedeutung und Konsequenz

BEDEUTUNG DER NEBENLÄUFIGKEIT FÜR VSYS

- Parallele Verarbeitung wesentlicher Geschwindigkeitsvorteil von VSYSen
  - In diesem Fall wirklich parallel und nicht nur quasi-parallel im Vergleich zu zentralisiertem System auf Einprozessor-Maschinen
- Rein sequentielle Verarbeitung ist Geschwindigkeitsverlust gegenüber zentralisiertem System
  - Ursache: Kommunikationsoverhead

    → Sequentielles VSYS langsamer als zentralisiertes System

KONSEQUENZEN DER NEBENLÄUFIGKEIT

- Problem: Höhere Wahrscheinlichkeit für inkonsistentes System unter gleichzeitigem Zugriff
- Anforderung: Erhöhter Koordinationsbedarf
- *Beispiel: Gleichzeitiger Zugriff auf Ressourcen (Dateien, Datenbanken, Funktionen, etc.)*

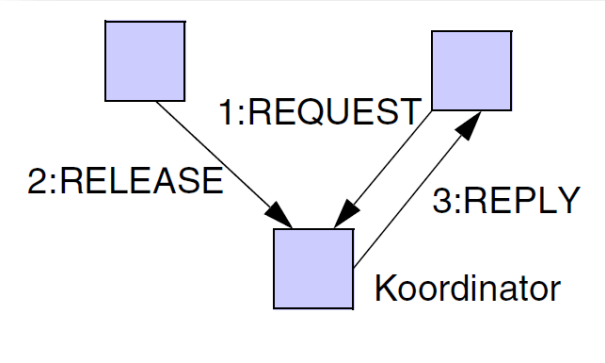
AUSSCHLUSSVERFAHREN: ALLGEMEIN

- Zentrale Verfahren: Kritische Bereiche mit Sperrobjekten; Monitore; Semaphore
- Verteilte Verfahren: Aufgeteilte Koordination (*Bsp. Alg. Ricard/Agrawala: Gemeinsame Zustimmung gemäß relativer Anfragezeit; Token-Alg.*)

3.6.2 Ausschlussverfahren

AUSSCHLUSSVERFAHREN: ZENTRALER ALGORITHMUS

- Zentraler Koordinator
  - Verwaltet Zugriff auf Ressource
  - Kennt den aktuellen Nutzer der Ressource (Komponente des VSYS)
  - Verzögert Antwort auf Ressourcenanfrage bis zu Release-Nachricht des vorherigen Nutzers
    - Variante: Mit Warteschlage für Anfragen
- *Beispiel: Kritische Bereiche mit Sperrobjekten, Monitoren, Semaphoren*
- Vorteile +
  - Einfache Realisierung
  - Fairness (wenn mit Warteschlange)
  - Anzahl der Nachrichten: 3 Nachrichten für Ressourcen-zugang (Request, Reply, Release)
- Nachteile –
  - Koordinator ist Flaschenhals
  - Zentraler Fehlerpunkt (Koordinator fällt aus)
  - Unklare Ursache bei verzögertem Reply



Quelle:
F. Hauck, Verteilte Betriebssysteme, Uni Ulm

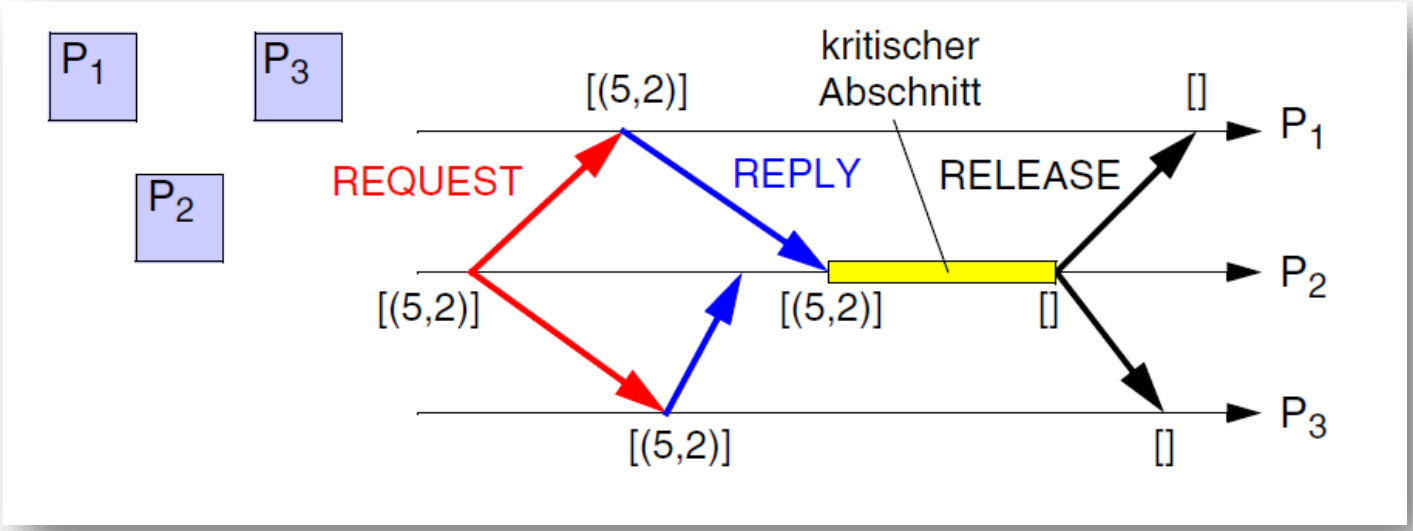
3.6.2 Ausschlussverfahren

AUSSCHLUSSVERFAHREN: VERTEILTER ALGORITHMUS NACH LAMPORT

- Warteschlangen für alle Komponenten
- Ressourcenanfrage
  - Komponente sendet Request-Nachricht an alle Komponenten
    - Verwendung eines relativen Zeitstempels (Beispiel: Lamports logische Uhr)
  - Einfügen des Requests in nach Zeitstempeln sortierte Warteschlange (ältester zuerst)
- Bearbeitung einer Request-Nachricht
  - Senden einer Reply-Nachricht an anfragende Komponente
- Voraussetzungen für Ressourcennutzung
  - Reply-Nachricht von allen Komponenten
  - Eigener Request am Warteschlangenkopf
- Freigabe der Ressource
  - Entfernen des eigenes Request aus der eigenen Warteschlange
  - Senden eines Release-Nachricht an alle Knoten → Entfernen Request aus ihrer Warteschlange

3.6.2 Ausschlussverfahren

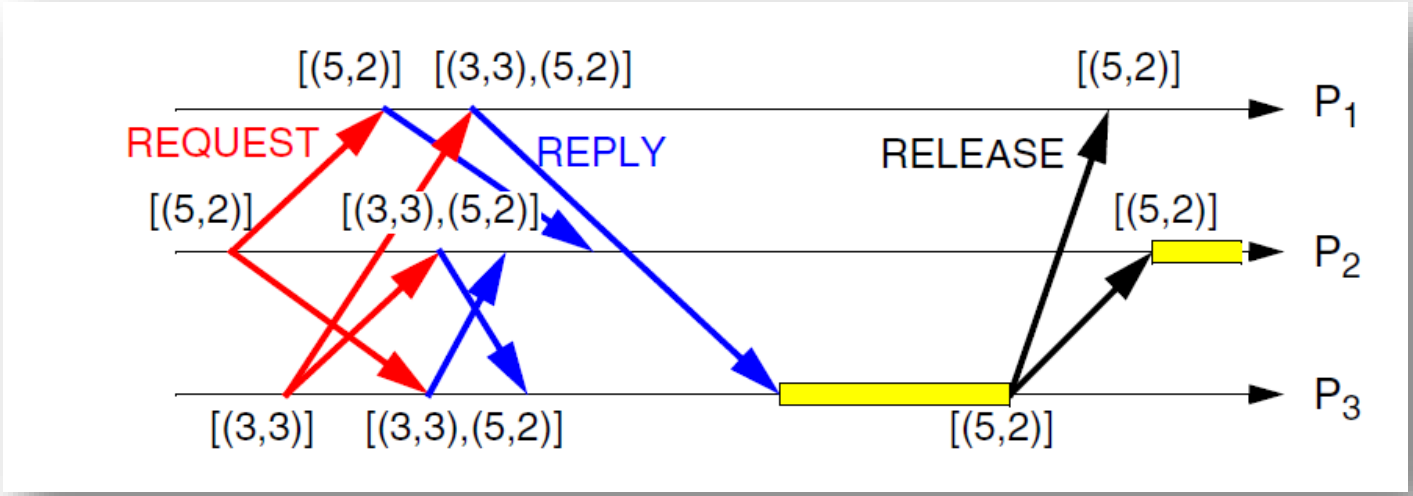
… AUSSCHLUSSVERFAHREN: VERTEILTER ALGORITHMUS NACH LAMPORT



Request-Nachricht von Komponente 2

(<relativer Zeitpunkt>, <anfragende Komponente>)

Quelle: F. Hauck, Verteilte Betriebssysteme, Uni Ulm



Request-Nachricht von Komponente 2 und 3

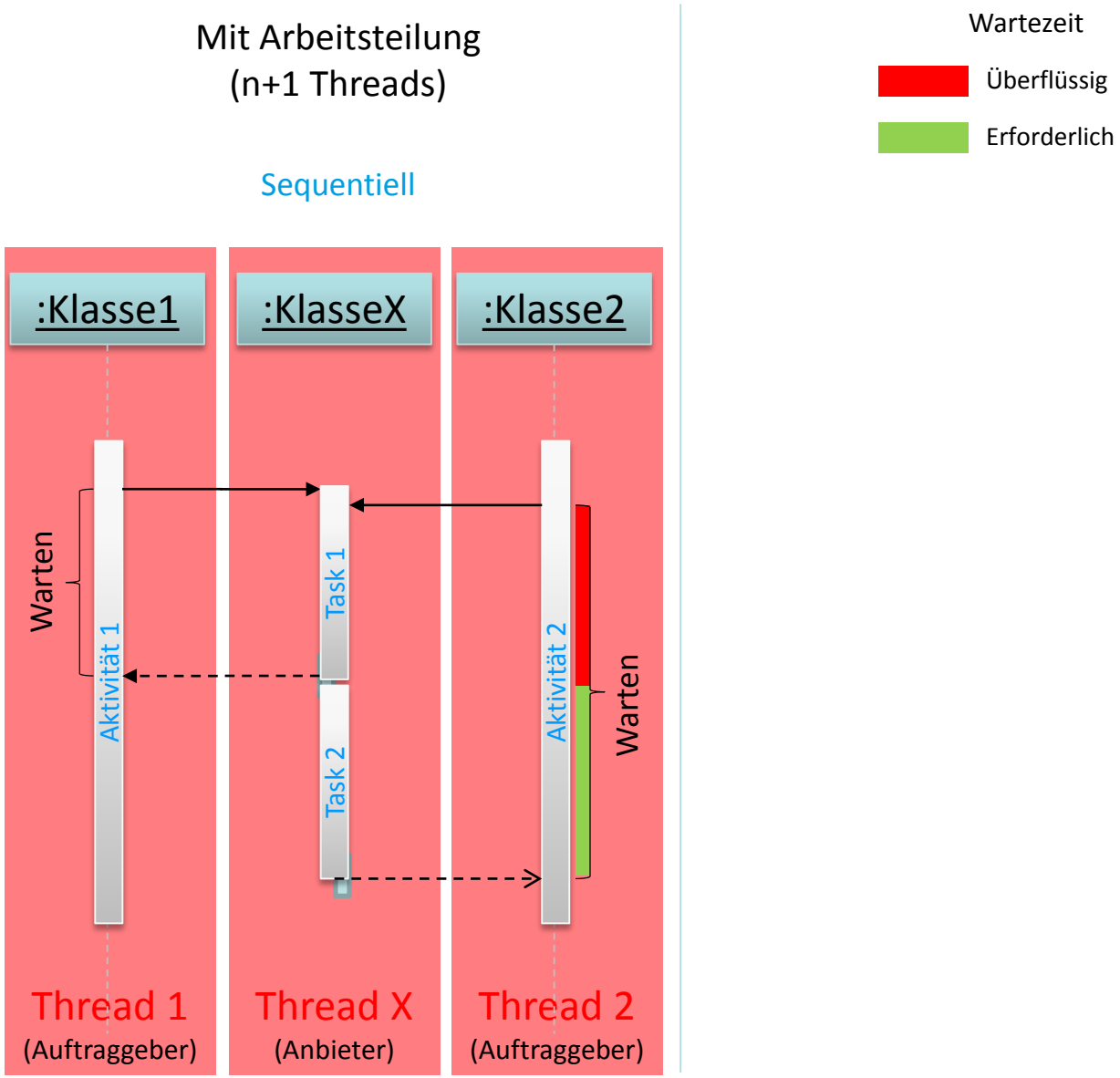
Quelle: F. Hauck, Verteilte Betriebssysteme, Uni Ulm

3.6.2 Ausschlussverfahren

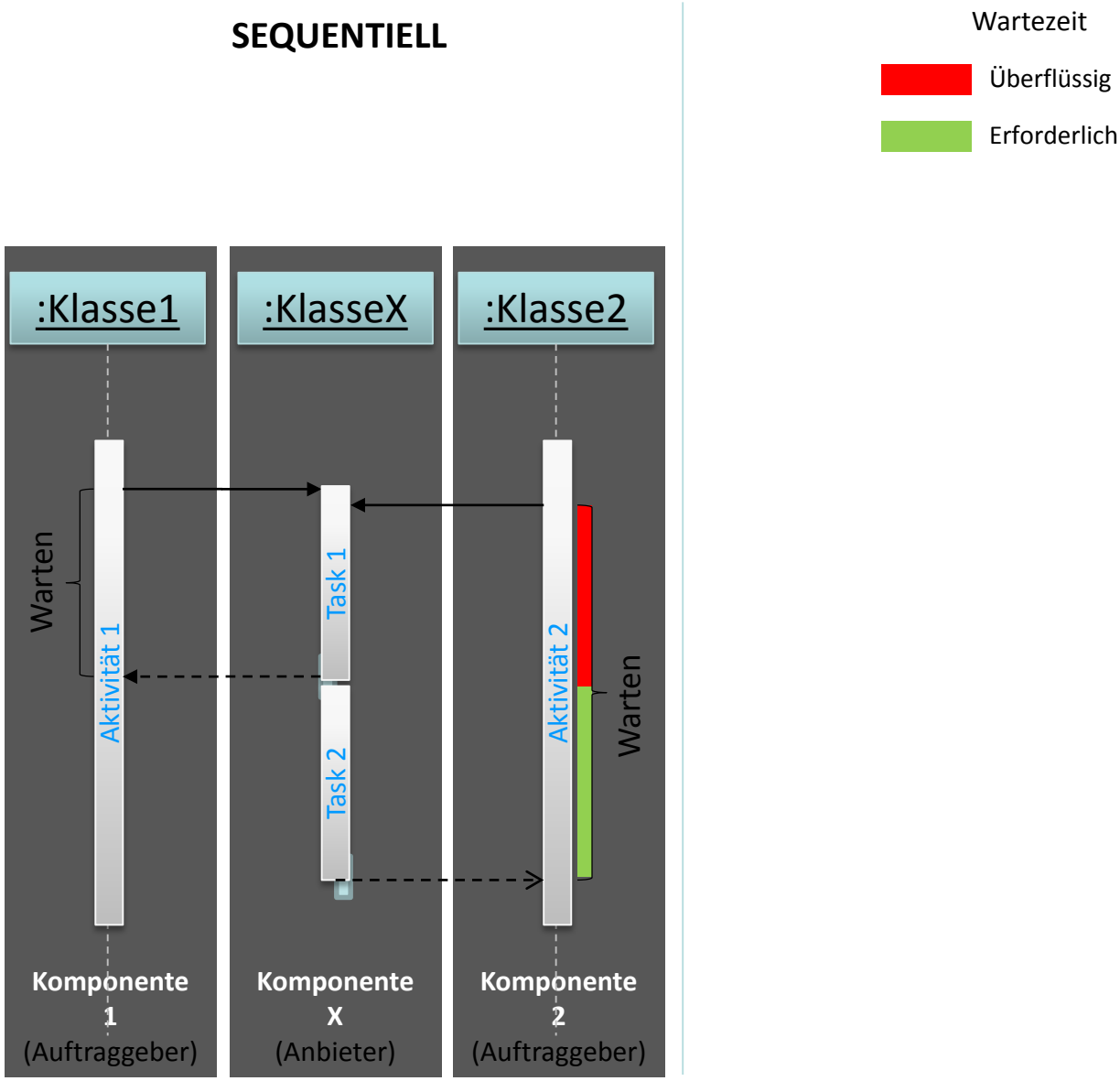
… AUSSCHLUSSVERFAHREN: VERTEILTER ALGORITHMUS NACH LAMPORT

- Vorteile +
  - Verteilte Ausschlussverfahren sind möglich
- Nachteile –
  - Zentraler Fehlerpunkt
  - Skalierbarkeit: Jede Komponente ist für Ressourcennutzung erforderlich (auch ohne Ressourcen-Mitbewerb)
  - 3(N-1) Nachrichten pro Ressourcenanfrage (Request, Reply, Release an alle)
- Alternative verteilte Verfahren
  - Ricard/Agrawala (1981) beseitigt Release-Nachricht
  - Token-Ring

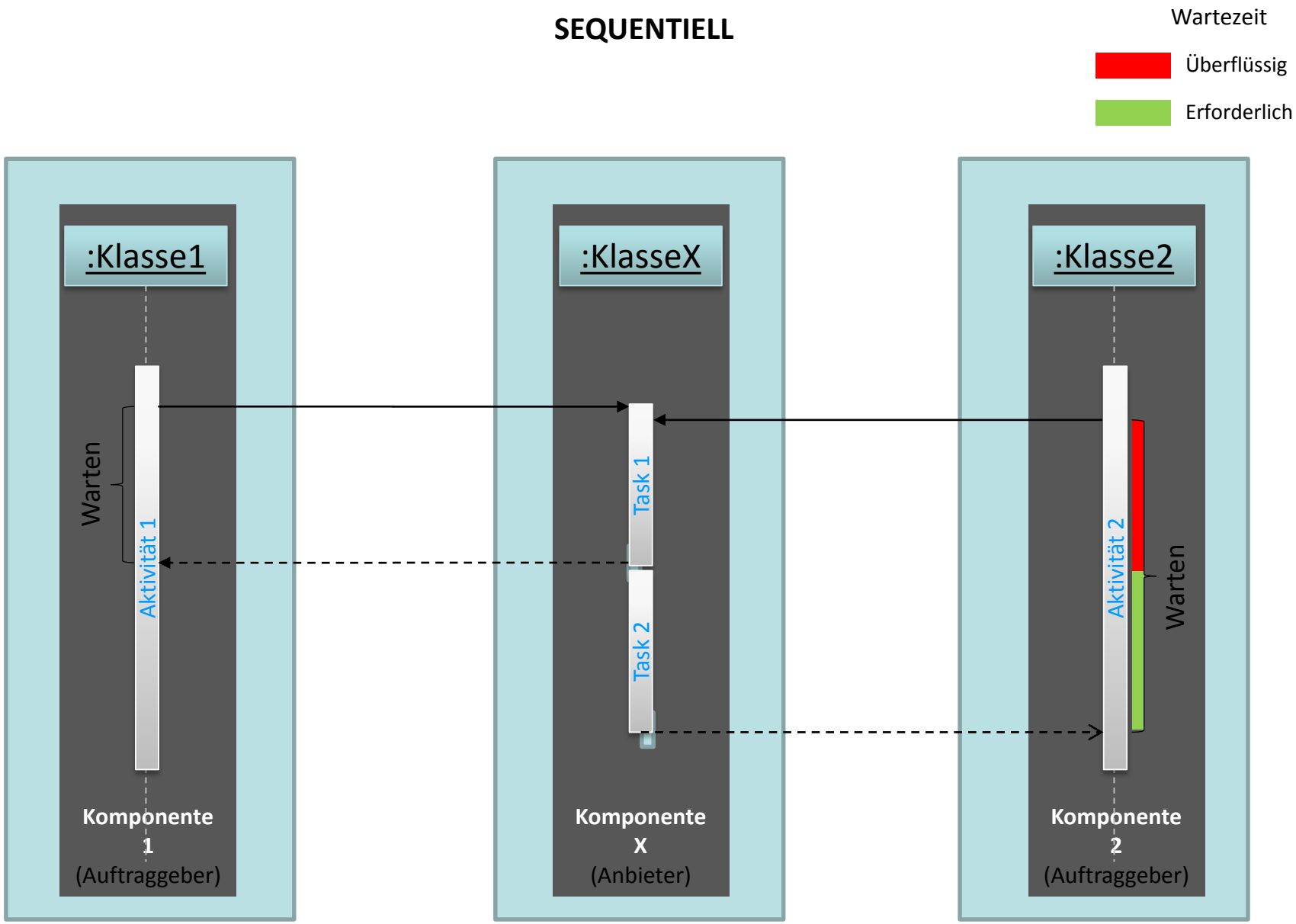
3.6.3 Grad und Auswirkung



3.6.3 Grad und Auswirkung

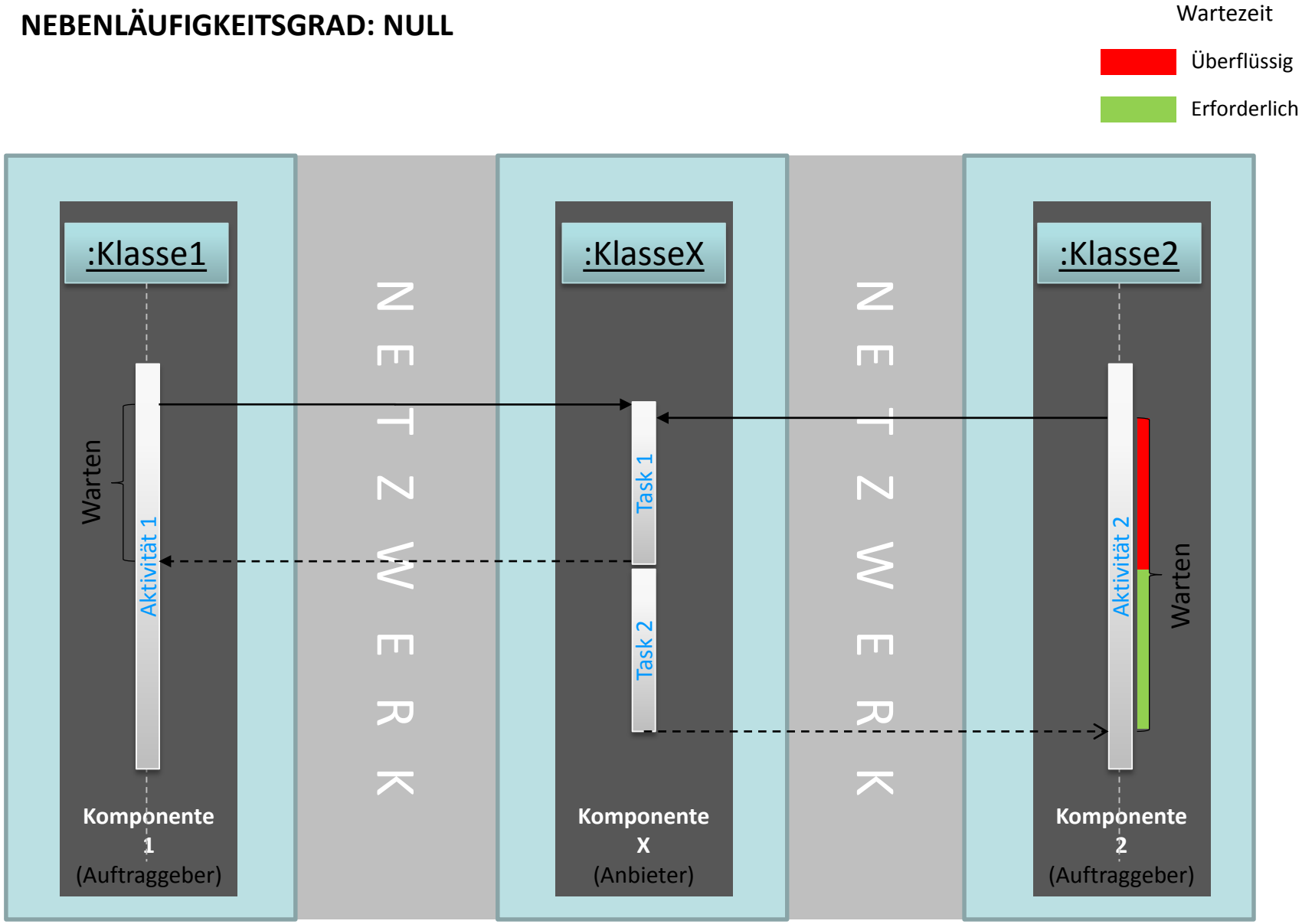


3.6.3 Grad und Auswirkung



3.6.3 Grad und Auswirkung

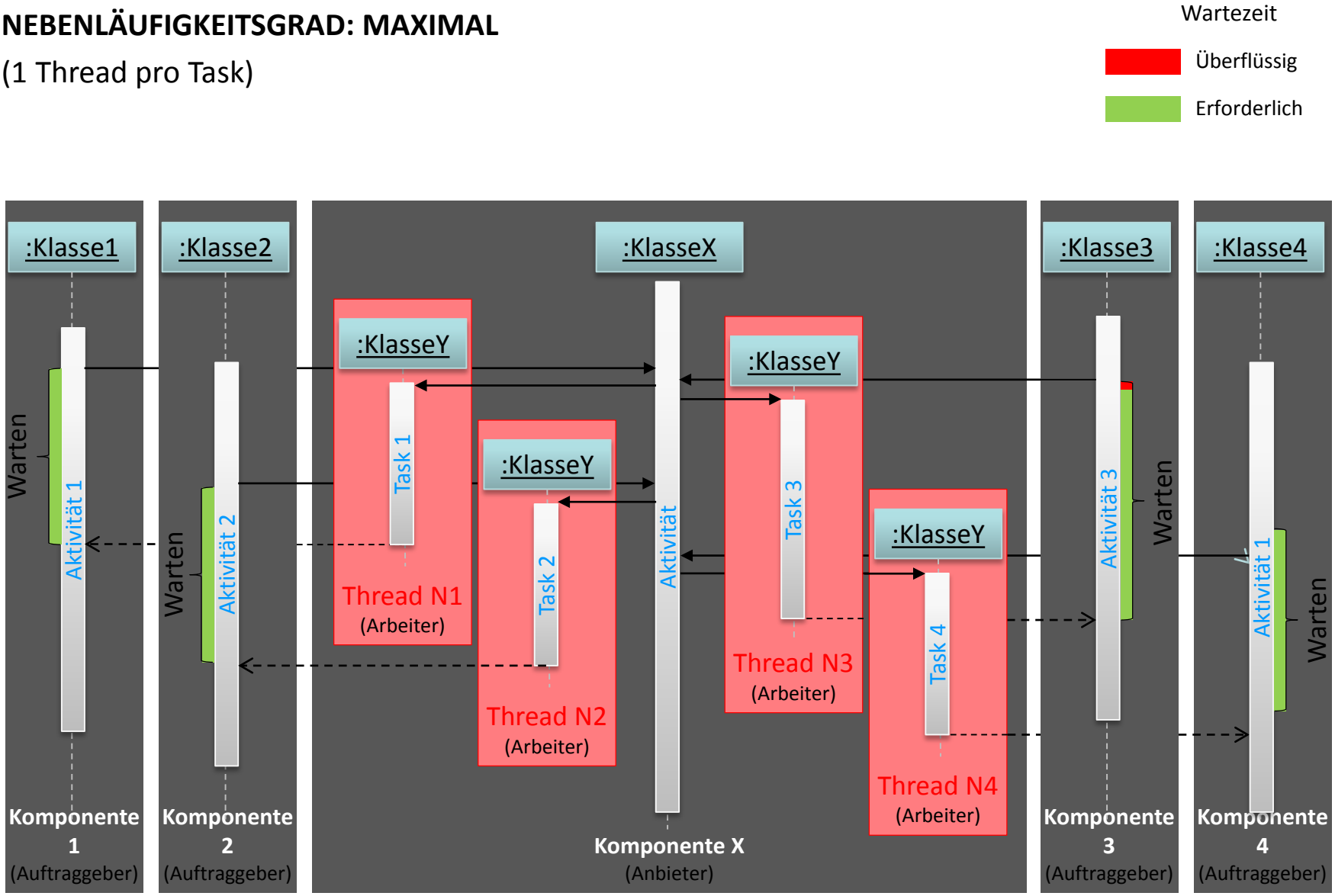
NEBENLÄUFIGKEITSGRAD: NULL



3.6.3 Grad und Auswirkung

NEBENLÄUFIGKEITSGRAD: MAXIMAL

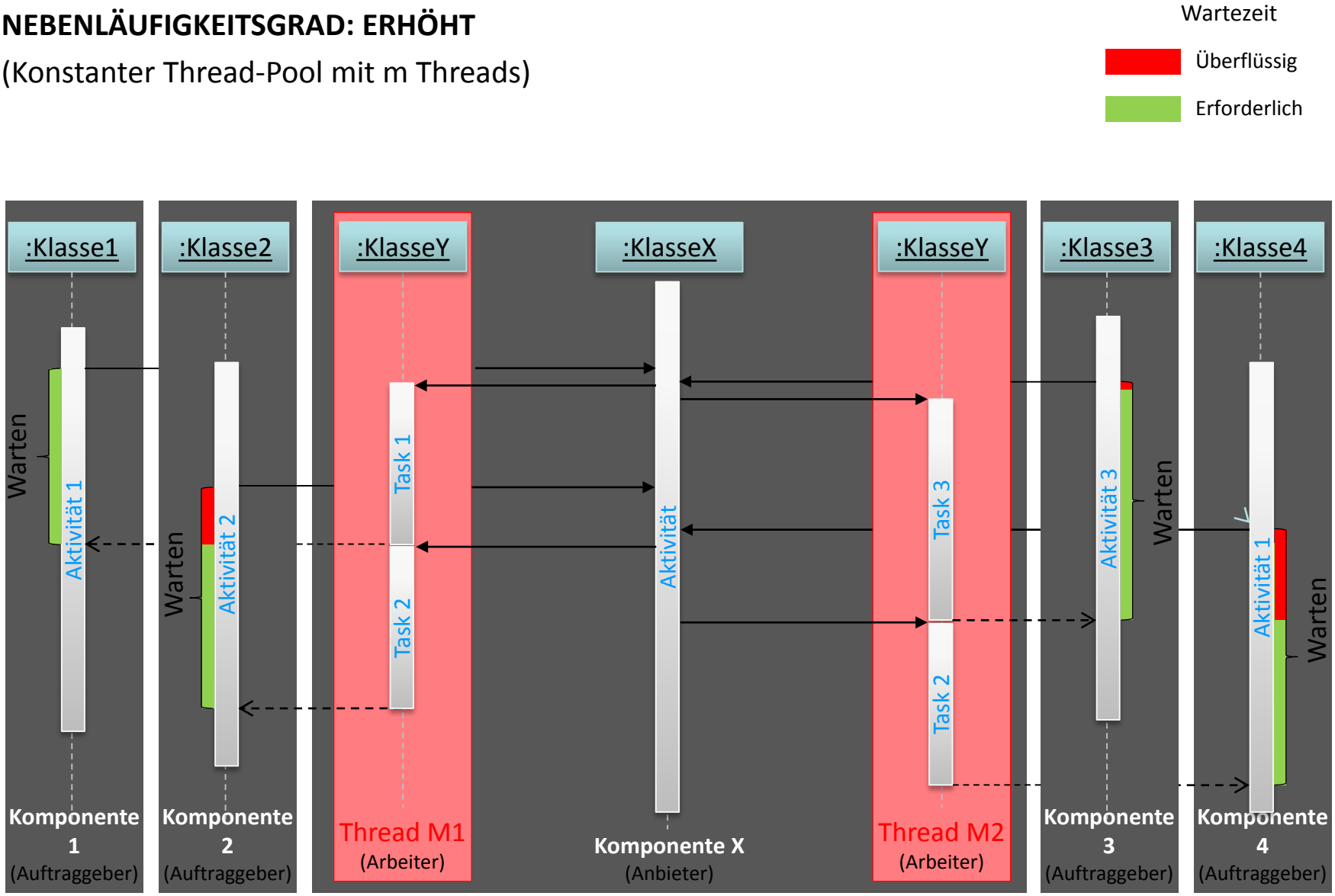
(1 Thread pro Task)



3.6.3 Grad und Auswirkung

NEBENLÄUFIGKEITSGRAD: ERHÖHT

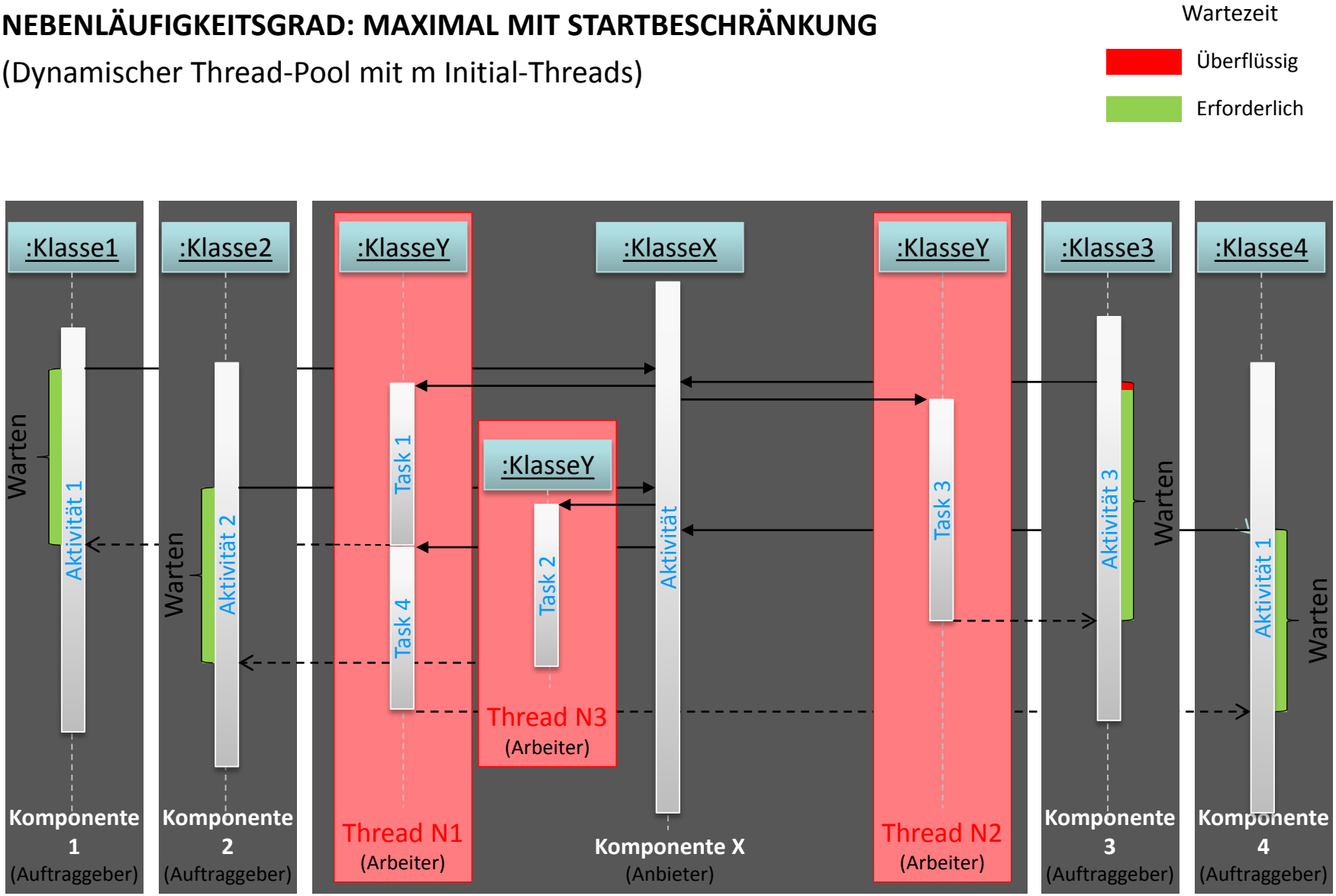
(Konstanter Thread-Pool mit m Threads)



3.6.3 Grad und Auswirkung

NEBENLÄUFIGKEITSGRAD: MAXIMAL MIT STARTBESCHRÄNKUNG

(Dynamischer Thread-Pool mit m Initial-Threads)



3.6.3 Grad und Auswirkung

LAUFZEITEN, WARTEZEITEN, LASTVERHALTEN UND RESSOURCENAUSNUTZUNG

Pool		Konsequenzen großer Anfragemengen für				
Verhalten	Größe	Tasks			Gesamtsystem	
		Neu		Laufend	Last	Ressourcen-ausnutzung
		Wartezeit	Laufzeit	Laufzeit		
Konstant	1	hoch	schnell	gleich	niedrig	schlecht
Konstant	m	mittel-hoch	langsam -schnell	gleich-langsamer	niedrig-mittel	schlecht-optimal
Dynamisch	m + ∞	keine	langsam	langsamer	(zu) hoch	optimal-hoch
Dynamisch	∞	keine	langsam	langsamer	(zu) hoch	optimal-hoch

NEBENLÄUFIGKEITSGRAD

- Mit zunehmendem Nebenläufigkeitsgrad …
  - sinkt die Wartezeit für neue Anfragen +
  - steigt die Laufzeit von Anfragen (neu und alt) +-
  - steigt die Last der Anbieterkomponenten -
  - steigt die Ressourcenausnutzung +
  - **steigt die Komplexität und Fehleranfälligkeit des verteilten Systems** --

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

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

Wer kommuniziert?

Wer hat welche Kommunikationsrolle?

Architektur

→ 3.10 Rollenauflösung von Client und Server

Kommunikationskonzepte:

Rollen, Dienste, Architekturen

Technische Perspektive

→ 3.7 Kommunikationsgrad

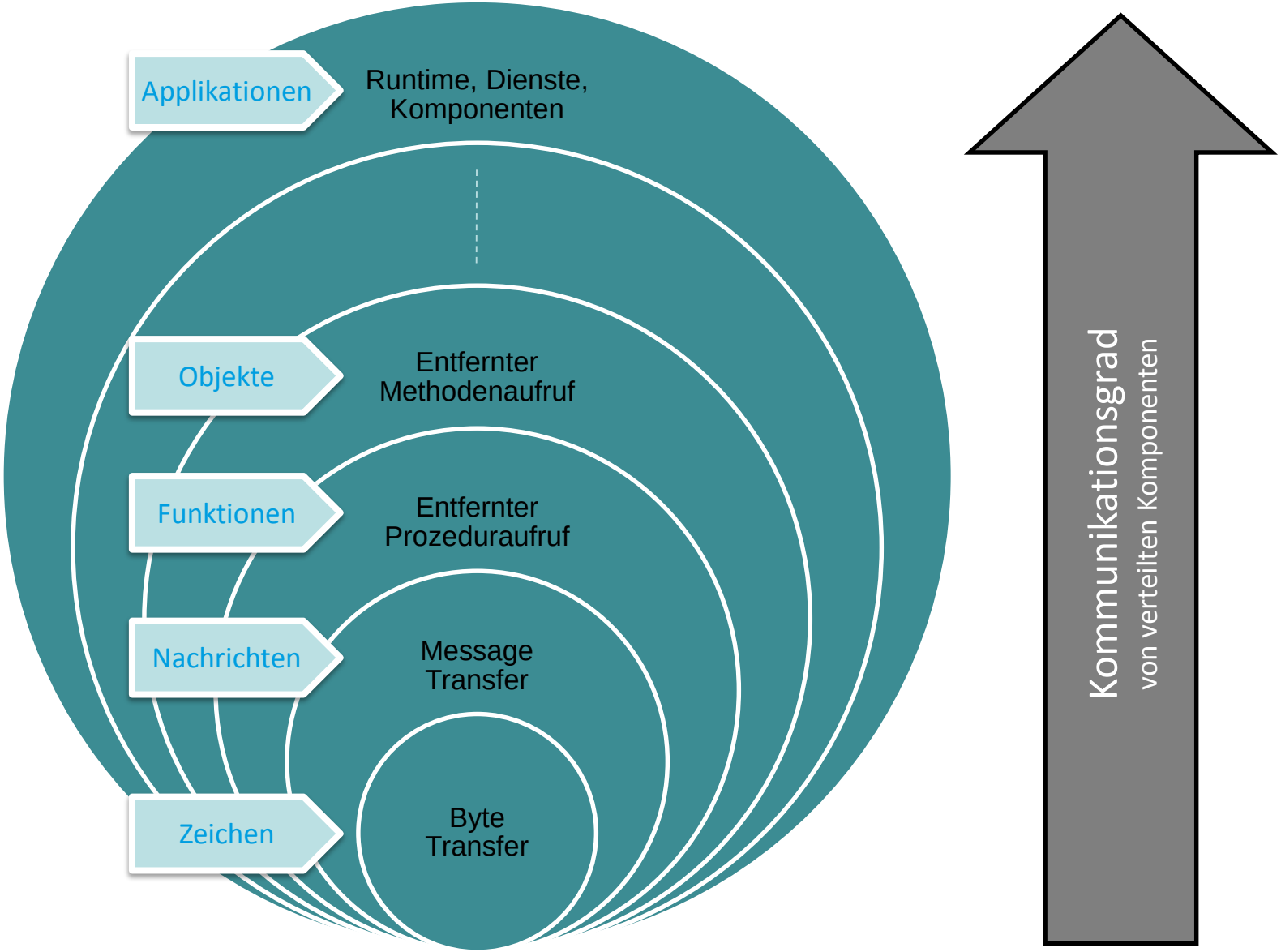
Kommunikationselemente:

Pakete, Nachrichten, Aufrufe

Wie und **was** kommunizieren wir?

Wie reagieren wir und mit **was?**

3.7.1 Schalenmodell



3.7.2 Sockets

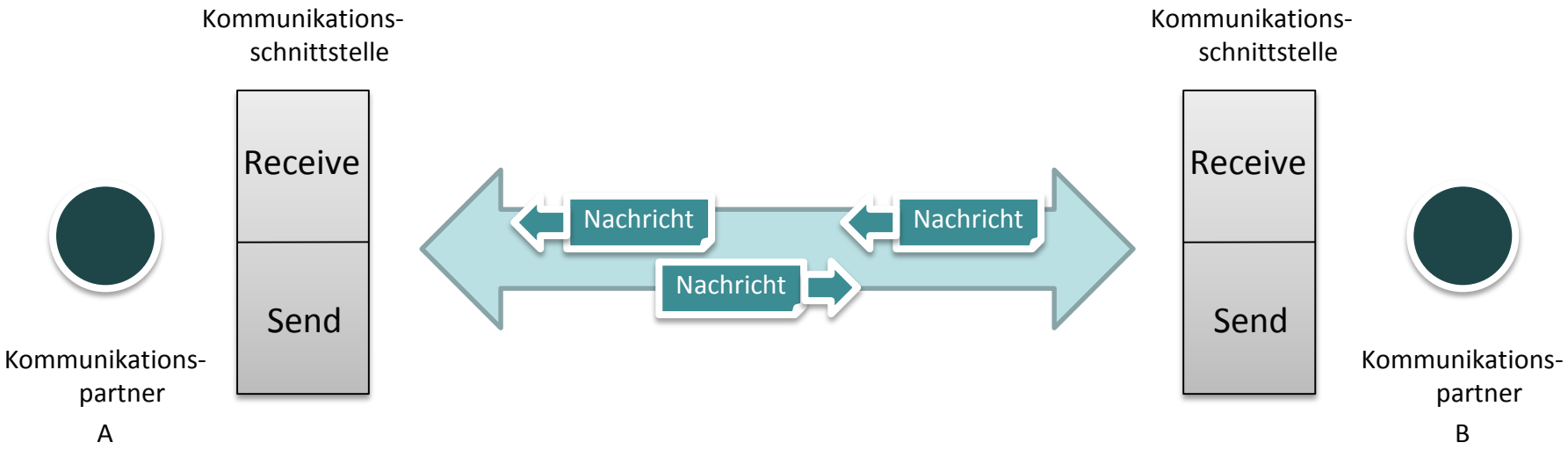
SYSTEM- UND PLATTFORMÜBERGREIFENDE API FÜR IPC

- 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)
- 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

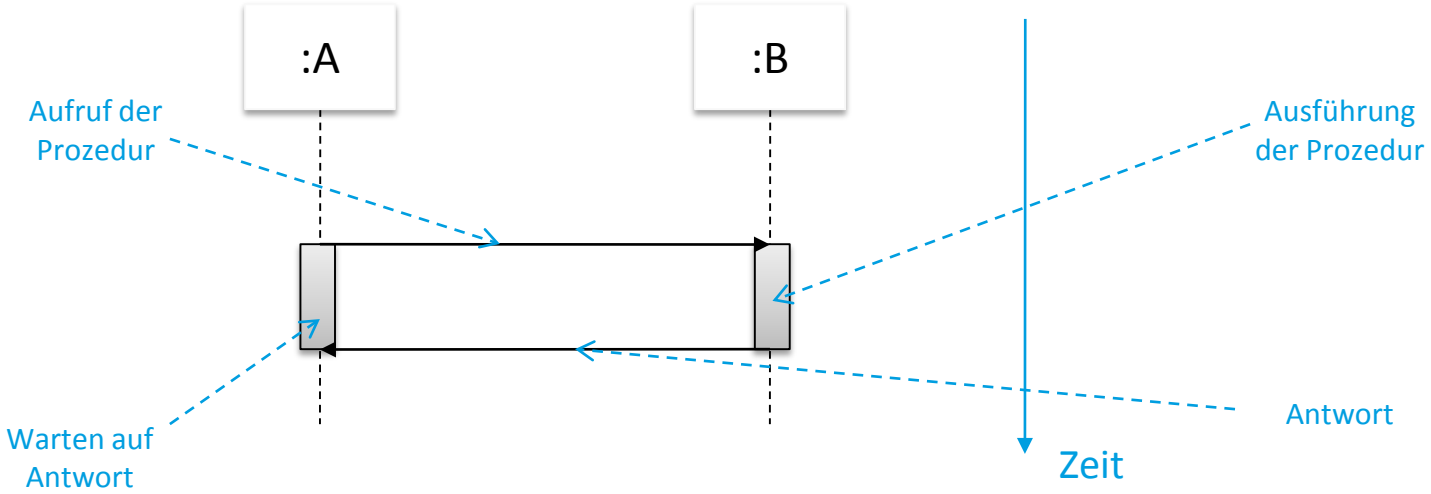
- Basis: Dienstprimitive **Send** (receiverAddress, message) und **Receive** (senderAddress, message)
- Mögliches Schicht 4-Protokoll: TCP/UDP
- Keine Initialisierung oder Aufräumsequenz
- Verbindungslose, paketorientierte Kommunikation



3.7.4 Entfernter Prozeduraufruf

NACHAHMUNG DES LOKALEN PROZEDURAUFRUFS

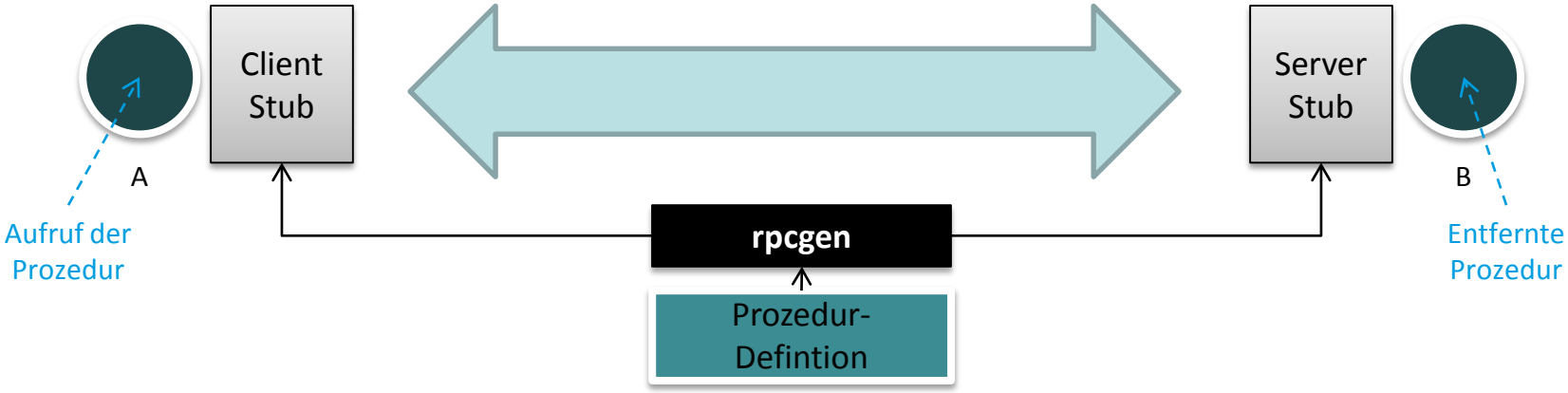
- Bezeichnung: RPC (Remote Procedure Call)
- Ziel: Aufruf von Funktionen in fremden Adressräumen
- Herkunft: Sun Microsystems (entwickelt für NFS)
- Standard: IETF (RFC 1057, RFC 5531)
- Grundlage für: Java RMI, CORBA, DCOM, XML-RPC, RPyC (für Python)
- Verwendung in prozeduralen Programmiersprachen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



3.7.4 Entfernter Prozeduraufruf

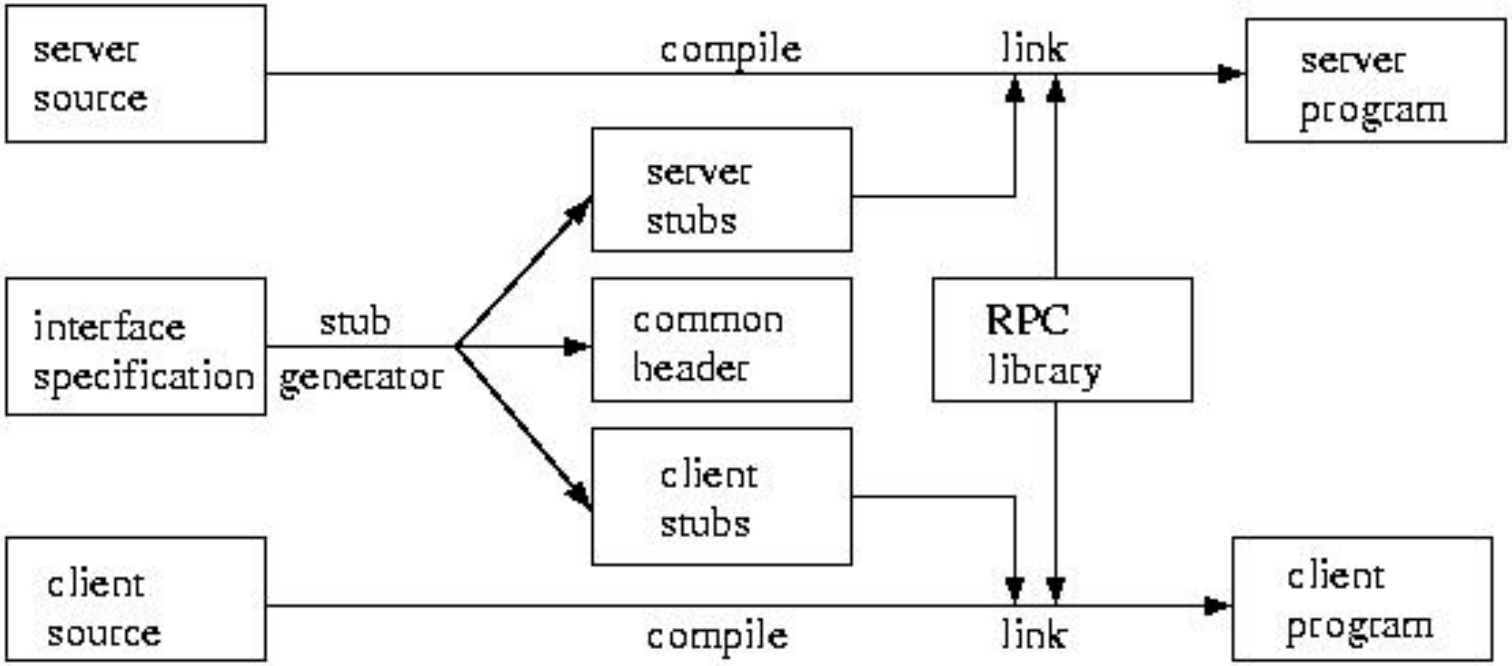
RPC-MIDDLEWARE MIT STUBS

- 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



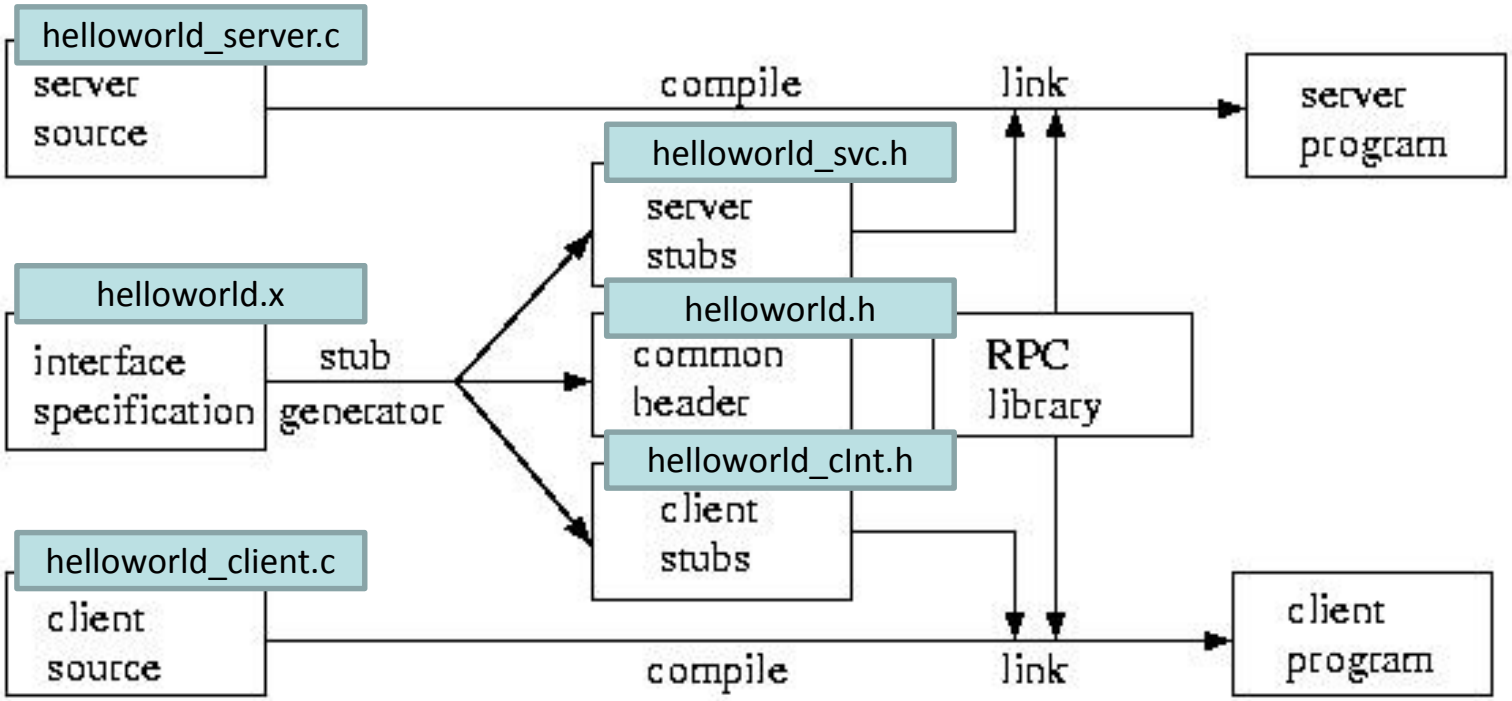
3.7.4 Entfernter Prozeduraufruf

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

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);
}
```

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

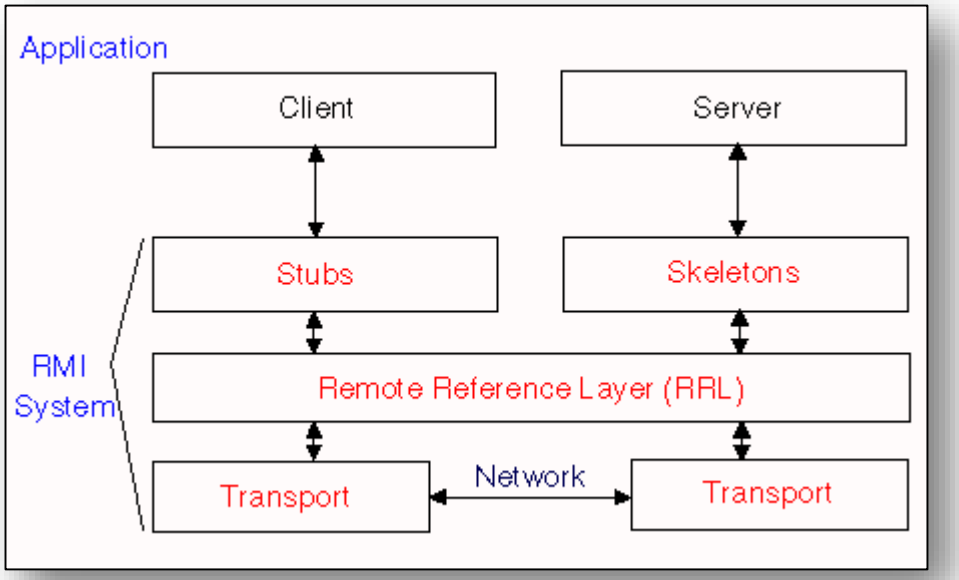
ARCHITEKTUREN

- 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*

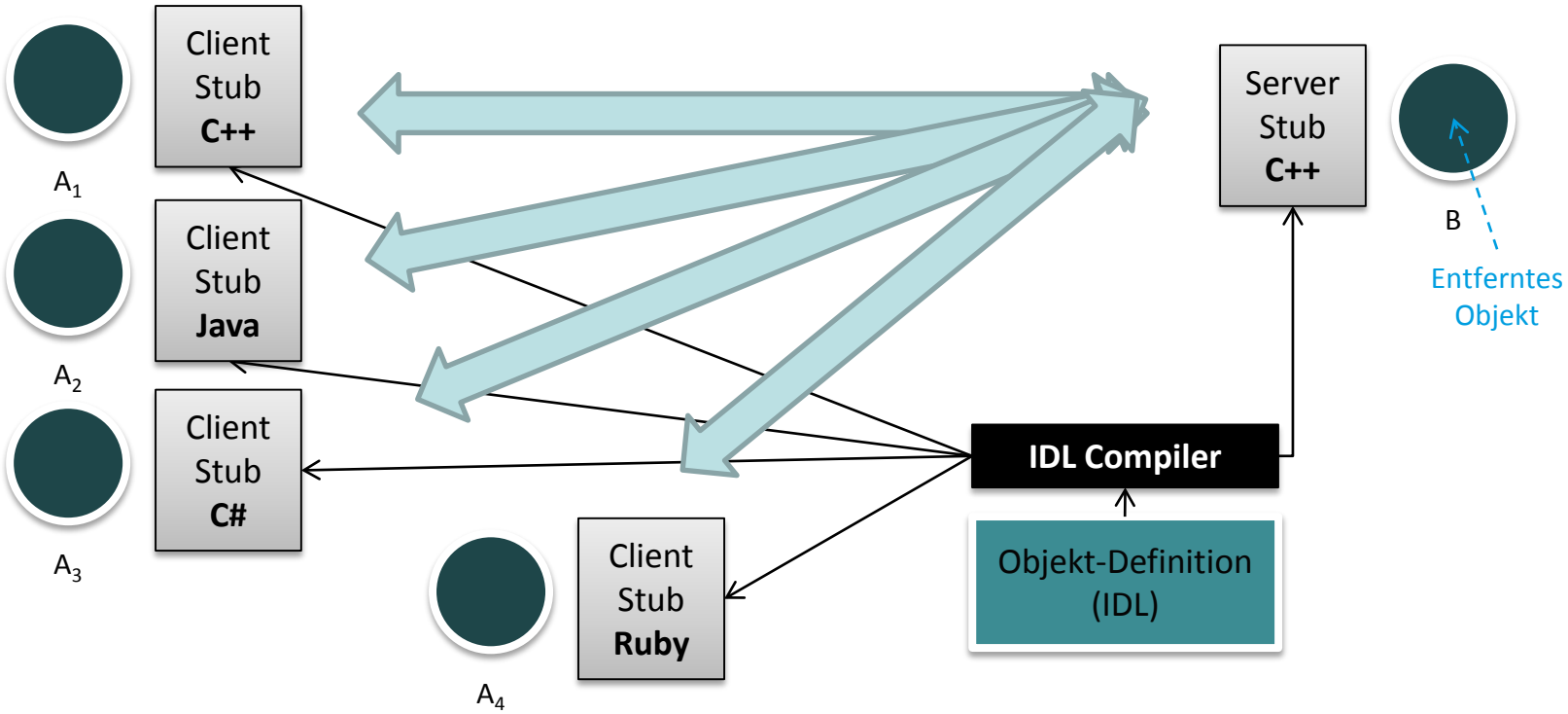


Quelle: Chris Matthews – Introduction to RMI

3.7.5 Entfernter Methodenaufruf

VORTEIL SPRACHUNABHÄNGIGKEIT

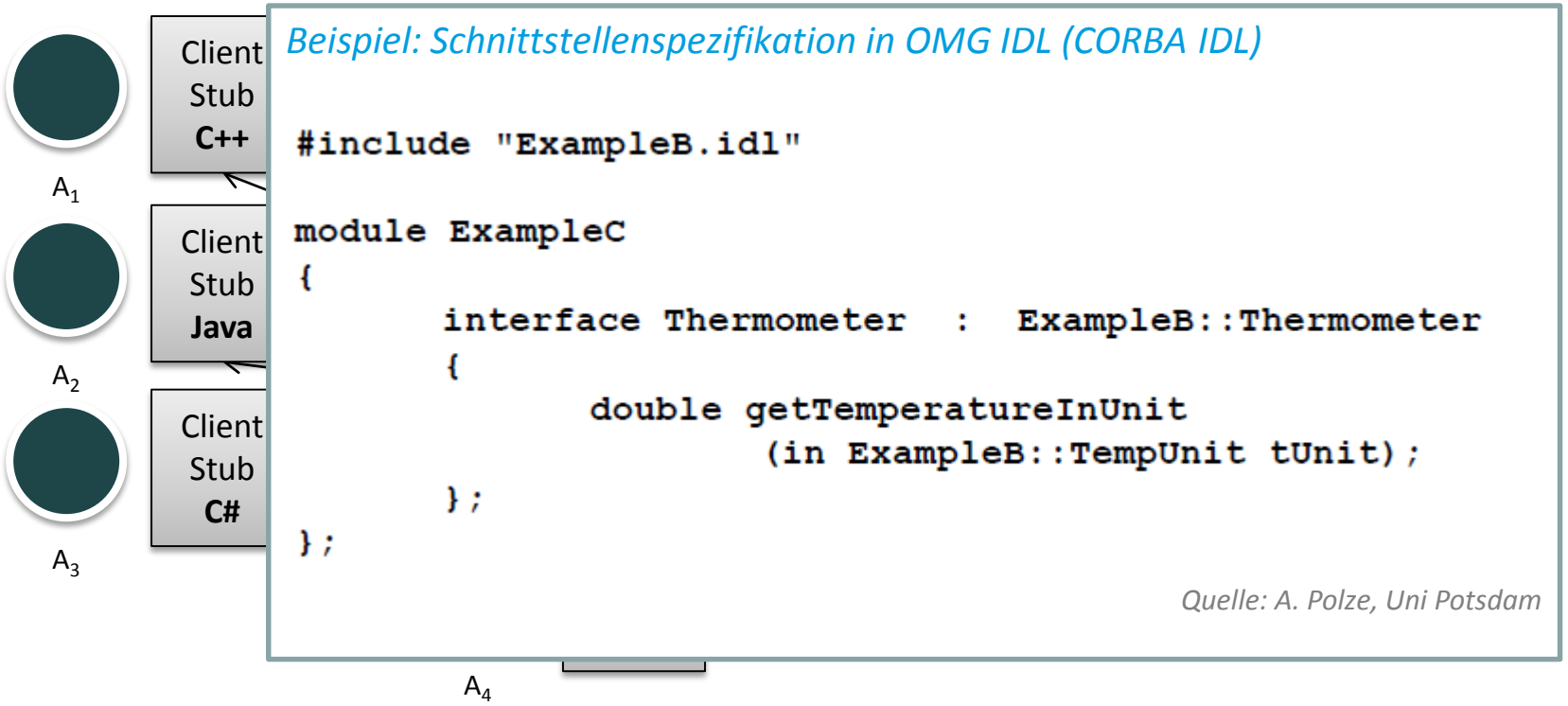
- Erster Schritt: Verwendung von XDR wie bei RPC
  - Sprachtransparenz: Sender benötigt keine Kenntnis von verwendeter Empfänger-Sprache
- Zweiter Schritt: Beschreibung des entfernten Objekts in **IDL** (Interface Definition Language)
  - *Beispiel: OMG IDL (CORBA IDL), AIDL*
- Dritter Schritt: Stub-Generierung 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*

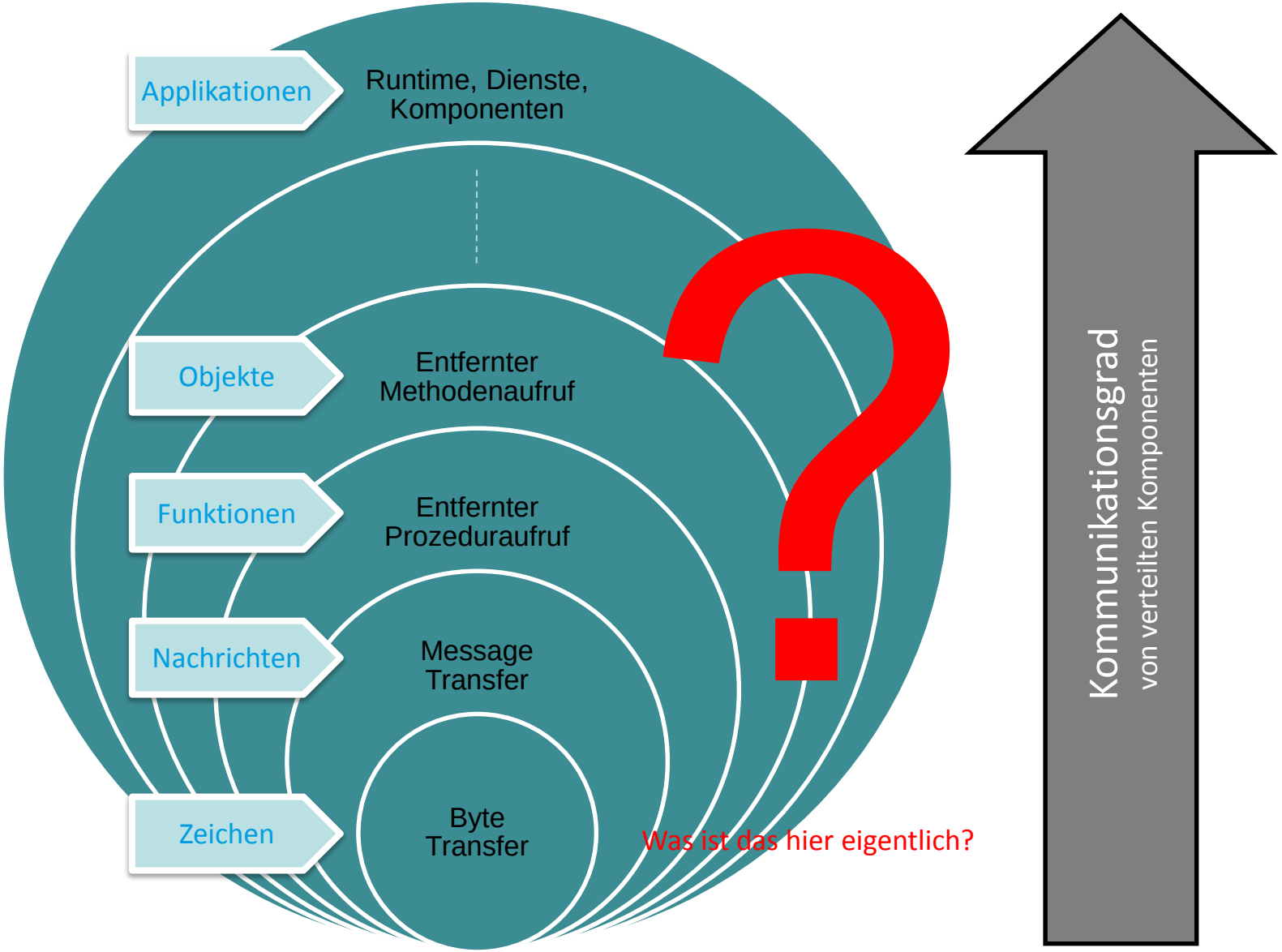


Beispiel: Schnittstellenspezifikation in OMG IDL (CORBA IDL)

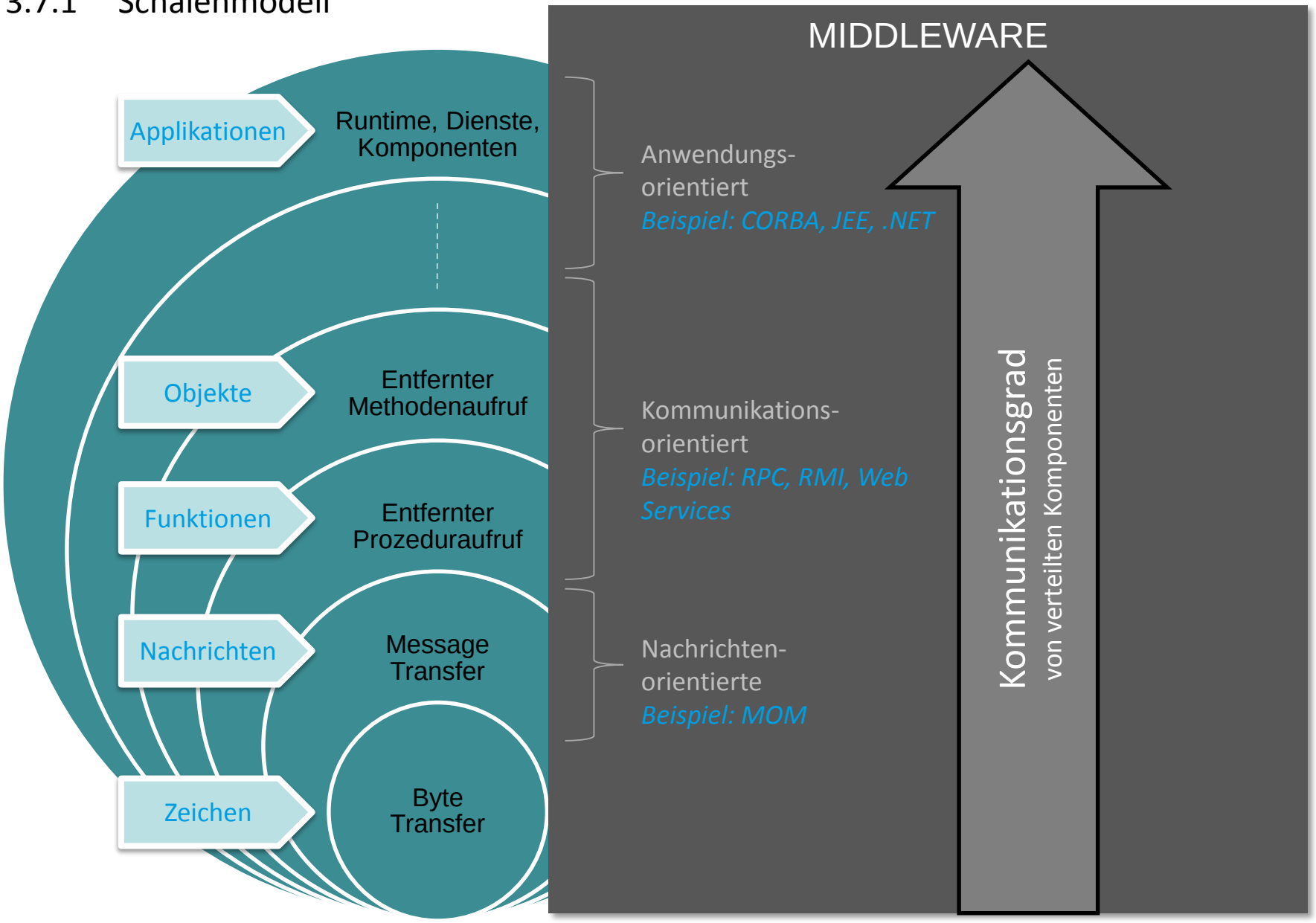
```
#include "ExampleB.idl"

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

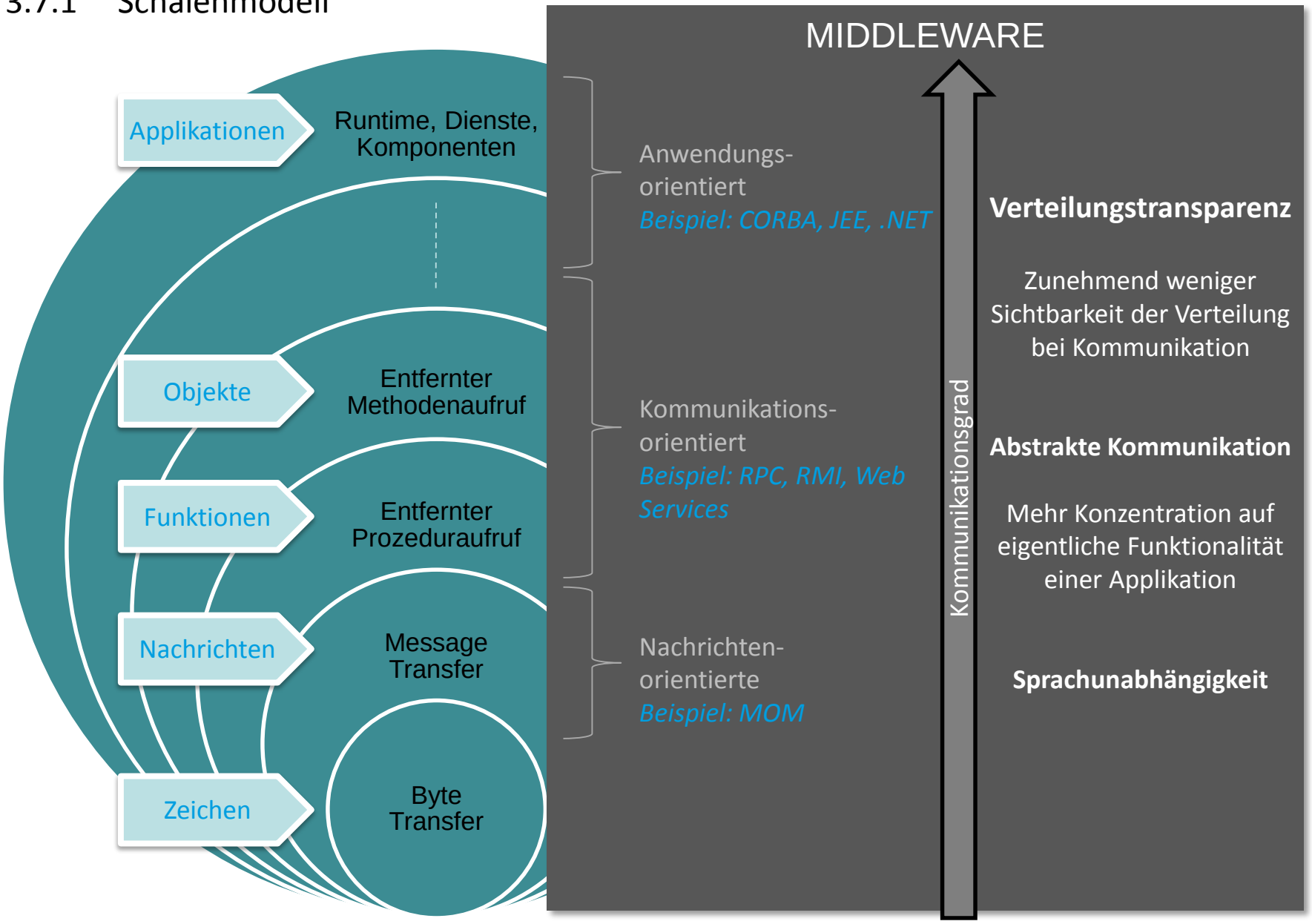
- Anwendungsneutrale Vermittlungssoftware zwischen verteilten Anwendungen
  - Vermittelt so, dass Komplexität, Plattform und Infrastruktur der Anwendungen verborgen bleibt
  - Vermittelt so, dass Netzwerk für Anwendungen transparent wird
- Baut auf kommunikationsorientierter Middleware auf (RMI, RPC, 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*

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