Fachhochschule Dortmund
University of Applied Sciences and Arts

we focus on students

Softwaretechnik 2

Architekturstile II

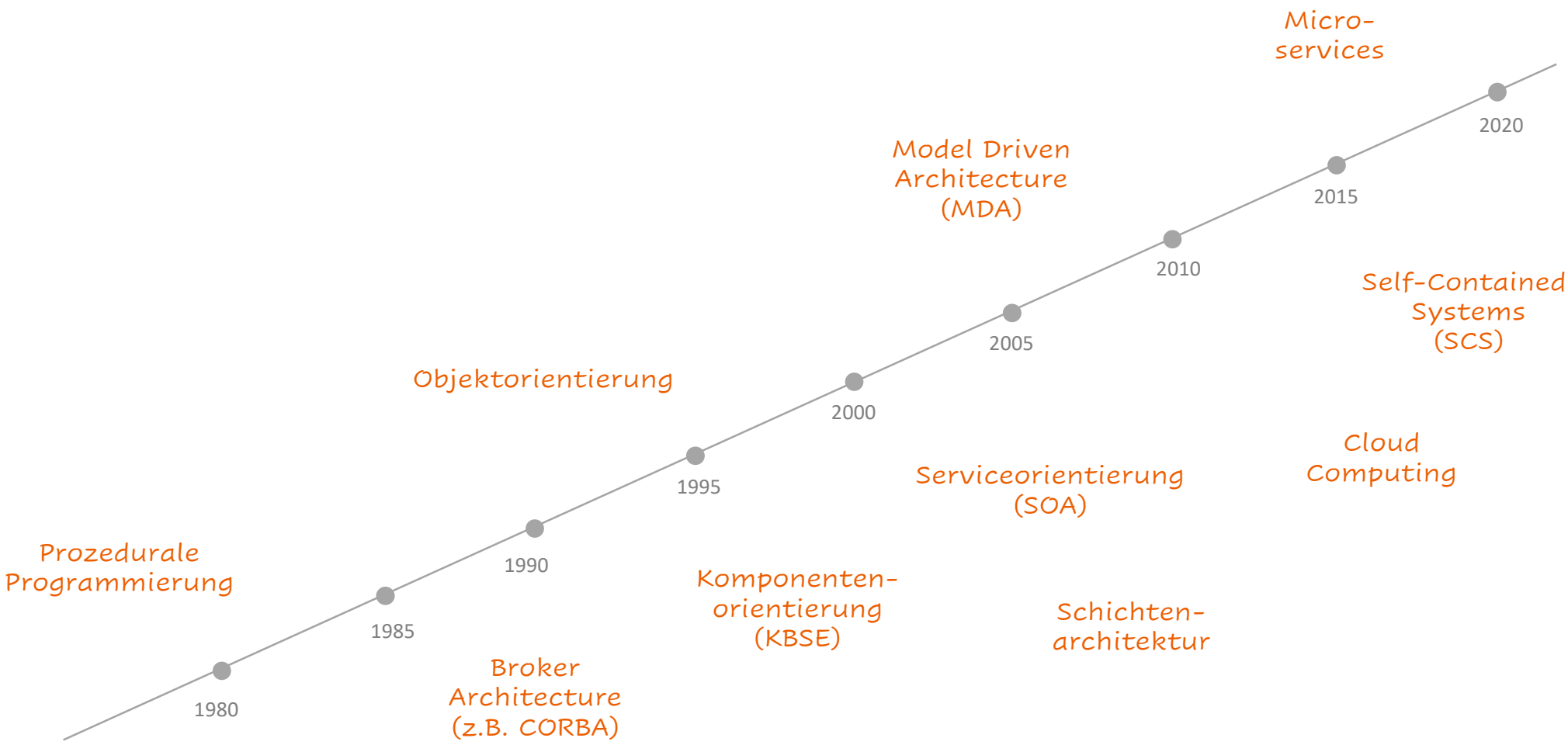
Architekturstile

nach Gernot Starke 2017

Stil	Intention	Beispiele
Datenflusssysteme	Besteht aus Folge (Sequenz) von Operationen auf Daten	**Batch-Sequentiell** **Pipes und Filter** Prozesssteuerung
Datenzentrische Systeme	gemeinsam verwendeter, zentraler Datenbestand	**Repository** **Blackboard**
Hierarchische Systeme	Bestehen aus Bausteinen in unterschiedlichen Ebenen einer Hierarchie	Haupt- und Unterprogramm, Master-Slave, **Schichten** Ports und Adapter virtuelle Maschine
Verteilte Systeme	Bestehen aus Speicher- und Verarbeitungsbausteinen, die über Kommunikationsnetze interagieren	**Client-Server,** CQRS, **Broker-Architekturen,** **serviceorientierte Architekturen** **Peer-to-Peer**
Ereignisbasierte Systeme	Bestehen aus voneinander unabhängigen Bausteinen, die über Ereignisse (Events) kommunizieren, sich implizit aufrufen.	**Ungepufferte Kommunikation:** **Broadcast, Publisher-Subscriber** **Gepufferte Kommunikation:** **Message-Queue,** **Message-Service**
Interaktionsorientierte Systeme	Systeme mit (grafischer) Bedienoberfläche	Model-View-Controller u.a.
Heterogene Systeme	Verwenden mehrere Architekturstile parallel	beispielsweise REST

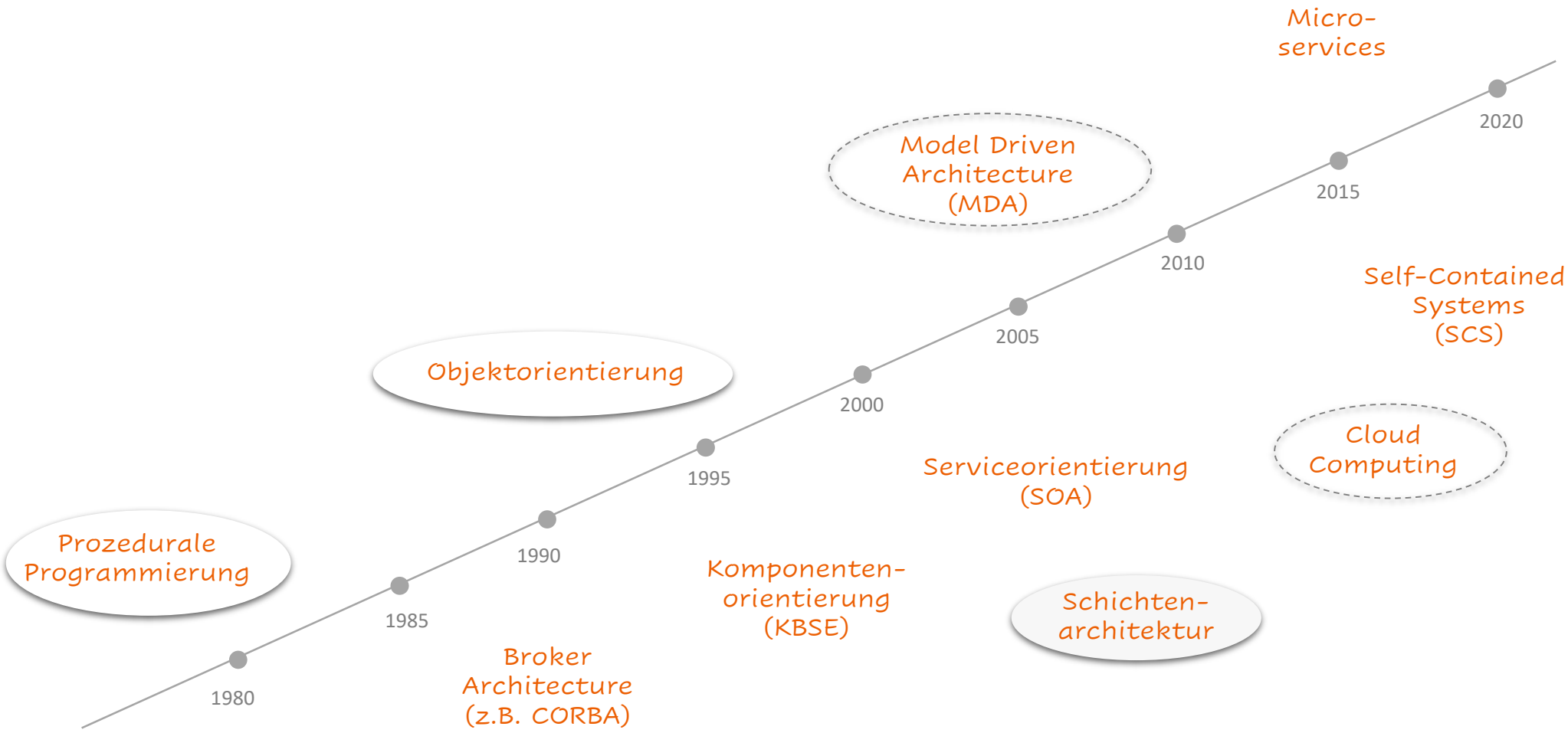
Entwicklungs- und Architekturtrends im Zeitverlauf

Unvollständige Auswahl großer Softwaretechnik-Trends



Entwicklungs- und Architekturtrends im Zeitverlauf

Unvollständige Auswahl großer Softwaretechnik-Trends



Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturstile II

Broker

KBSE

SOA

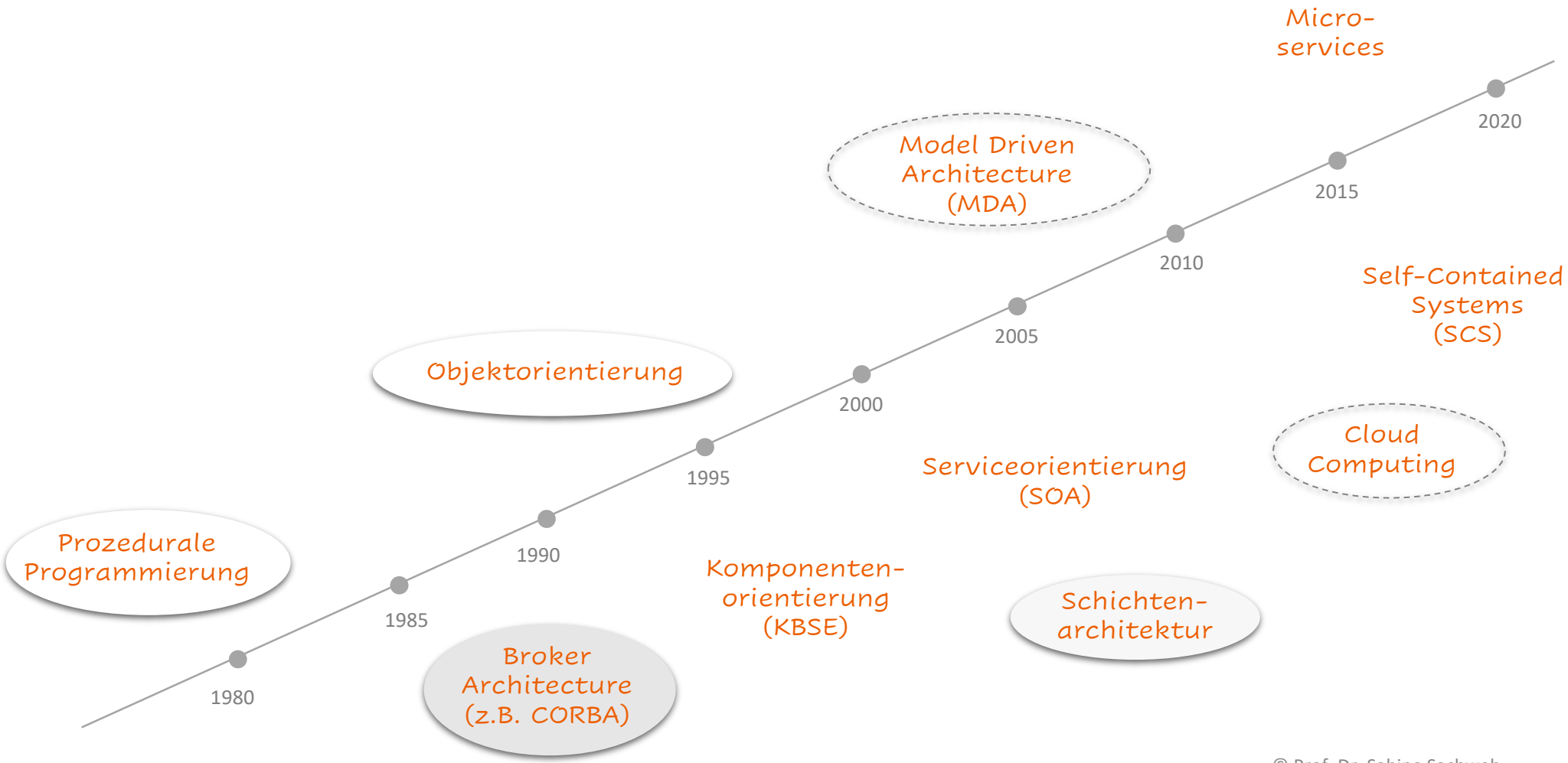
Fachhochschule
Dortmund
University of Applied Sciences and Arts

Broker

am Beispiel CORBA

Entwicklungs- und Architekturtrends im Zeitverlauf

Unvollständige Auswahl großer Softwaretechnik-Trends



Einordnung

- Ende der 80ger Jahre wurden IT-Systeme komplexer und vermehrt eingesetzt
- zu nehmender Bedarf des Datenaustausches zwischen den Systemen
- Anwendungslandschaften „wuchsen langsam zusammen“
- leistungsfähigere Hardware und Betriebssysteme lieferten eine geeignete Basis für Client/Server-Systeme
- zusätzlich stieg der Druck auch ältere Systeme anzubinden und zu warten

⇒ **benötigt wurde eine offene und flexible Kommunikationsinfrastruktur**

Object Management Group (OMG)

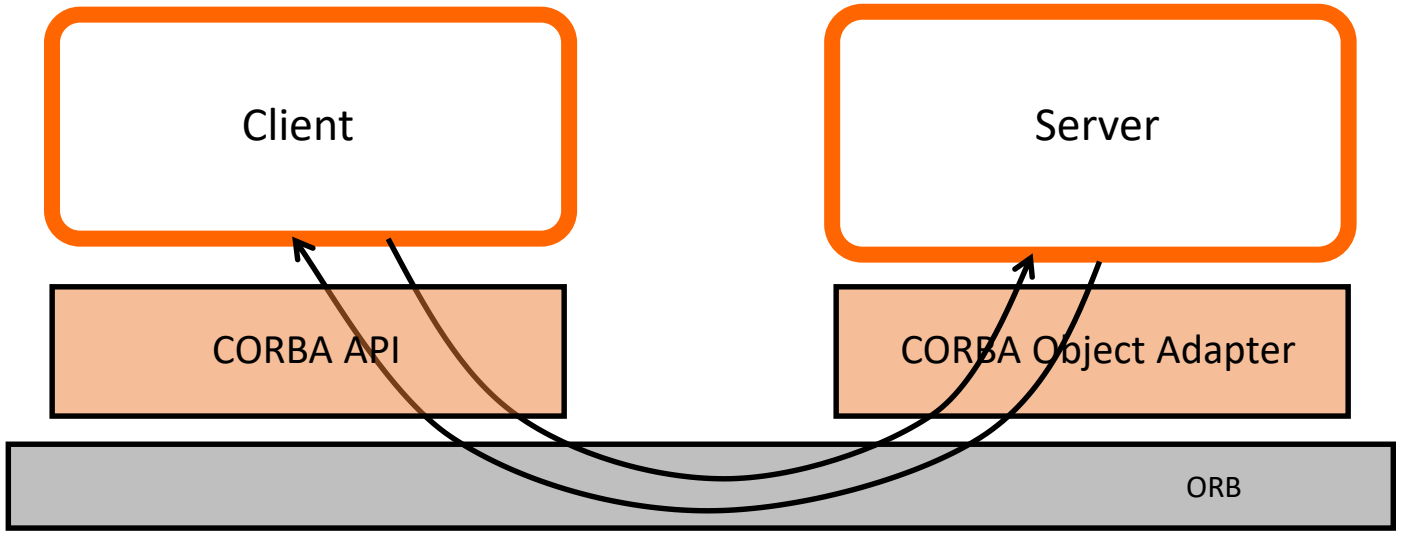
- Object Management Group (OMG)
  Internationaler Zusammenschluss von akademischen Instituten, Hardwareherstellern, Softwareentwickl…,
  Netzwerkbetreibern und kommerziellen Nutzern
- Gründung im Jahre 1989, schnell wachsend, mittlerweile mehr als 800 Mitglieder
- Vision
  Interoperabilität maximieren
- Fokus
  Integration von Systemen und Anwendungen über heterogene Plattformen hinweg
- Ziel
  Sammeln und Aufarbeiten von technologischem Know-how für die Entwicklung verteilter,
  auf Basis verschiedener Plattformen entstandener und laufender Software
- Bekannte Ergebnisse: CORBA (1991) und UML (1997)

CORBA-Eigenschaften

CORBA zeichnet sich vor allem durch folgende Eigenschaften aus:

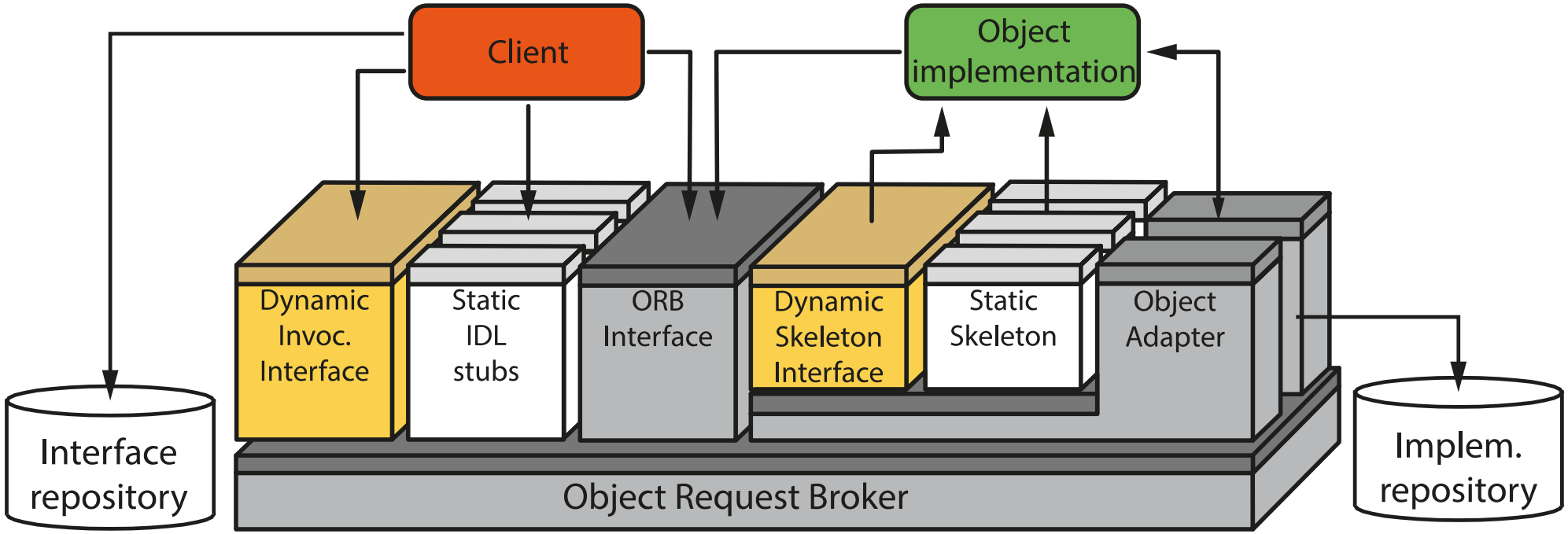
- Grundlage für eine mehrschichtige Client/Server-Architektur
- Sprach- und Systemunabhängigkeit
- Interoperabilität zwischen heterogenen Systemen
- offener Standard
- stabile Basis für verteilte Objekte
- Verfügbarkeit von CORBA auf allen bekannten Betriebssystemen
- CORBA macht Netzwerkkommunikation transparent für Entwickler

CORBA-Grundkonzept (Broker)

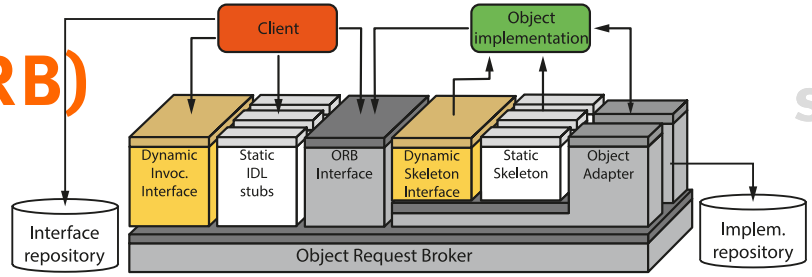


- Literatur: OMG Einführung in CORBA: http://www.omg.org/gettingstarted/corbafaq.htm
  Java-Beispiel: https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

CORBA-Architektur

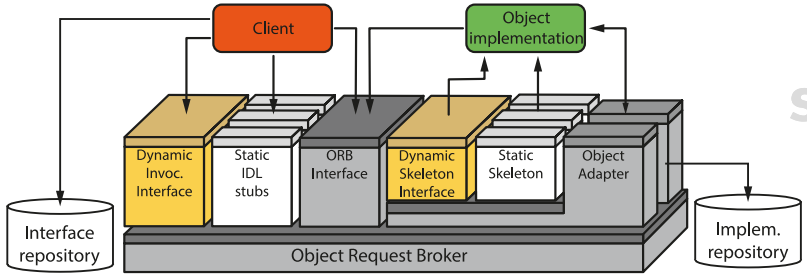


Object Request Broker (ORB)



- Kommunikationsbus zwischen CORBA-Objekten, der einen Mechanismus anbietet, um Anfragen des Clients an die entsprechende Implementierung weiterzuleiten, unabhängig davon, an welchem Ort diese im Netzwerk liegen und unter welchem Betriebssystem sie dort laufen.
- Eine Anfrage des Clients, bestehend aus dem Methodenaufruf und den Parametern, wird serialisiert (marshaling) und über das Netzwerk an den Server geschickt.
- Die Informationen werden auf der Server-Seite wieder umgewandelt (unmarshaling) und die gewünschte Operation ausgeführt.
- Rückgabewerte werden auf die gleiche Weise wieder über den ORB an den Client gesendet.
- Der ORB hat also die Aufgabe, die entsprechende Objektimplementierung zu finden, sie zu starten, falls erforderlich, die Anfrage an das Objekt zu leiten und entsprechende Rückgabewerte wieder an den Client zurückzugeben.

Stubs und Skeletons



- ein Stub stellt die Verbindung zwischen Client und ORB dar, d.h.
  - er stellt dem Client eine Methodenaufrufschnittstelle zur Verfügung und
  - leitet Aufrufe von Methoden serialisiert an den ORB weiter.
- ein Skeleton stellt die Verbindung zwischen ORB und Server dar, d.h.
  - es empfängt die Nachricht (s.o.) auf der Server-Seite vom ORB
  - baut einen „richtigen“ Methodenaufruf zusammen
  - leitet den Aufruf an das Server-Objekt weiter und
  - wartet auf den Rückgabewert, den es entsprechend auf die Reise schickt.
- keine generische Aufgabe, d.h., Stubs und Skeletons sind nicht Teil eines fertig implementierten ORBs sondern werden aus einer Schnittstellen-Beschreibung generiert.
- Schnittstellen-Beschreibung wird in einer Interface Definition Languague (IDL) beschrieben

Interface Definition Languague (IDL)

- Programmiersprachen unabhängig, orientiert sich an der Syntax von C++
- es existieren unterschiedliche Bindungen an Programmiersprachen wie z.B. C, C++, Smalltalk, Ada, COBOL, Java
- eine IDL-Beschreibung enthält
  - den Namen der Schnittstelle
  - den Namen der Methode
  - die erlaubten Typen aller Ein- und Ausgabeparameter
- eine IDL-Beschreibung ist Basis der Stub- und Skeleton- Generierung

```
module <identifier>
{
  <type declarations>;
  <constant declarations>;
  <exception declarations>;
        interface <identifier> [:<inheritance>]
        {
            <type declarations>;
            <constant declarations>;
            <attribute declarations>;
            <exception declarations>;
            [<op_type>]<identifier>(<parameter_list>)
            [raises exception] [context];
                        .
                        .
        }
         interface <identifier> [:<inheritance>]
                .
}
```

Definition eines Namensraums

Definition einer CORBA Klasse

Definition einer Methode

Definition einer CORBA Klasse

Interface Definition Languague (IDL)

- Basisdatentypen
  - **void**
  - **boolean** - true und false
  - **char** - 8Bit
  - **wchar** - implementationsabha¨ngig, normalerweise 16 Bit
  - **float** - IEEE einfache Genauigkeit
  - **double** - IEEE doppelte Genauigkeit
  - **long double** - IEEE erweiterte doppelte Genauigkeit
  - (unsigned) **long** - 32 Bit Integer
  - (unsigned) **long long** - 64 Bit Integer
  - (unsigned) **short** – 16 Bit
  - **octet** – C und C++: char und unsigned char, Java: byte
  - **string** (feste und variable Zeichenla¨nge) – C: Zeichenarray, C++: Cstring, Java: String
- Komplexe Datentypen
  - **enum** (Erzeugung von Typen mit vordefinierte Wertemenge):

    ```
    enum Tage { Montag, Dienstag, ... };
    ```
  - **struct** (Strukturtyp, unterschiedliche Elementwerten, mit call-by-value u¨bergeben):

    ```
    struct Datum { short Jahr, short Monat, short Millisekunde };
    ```

Interface Definition Languague (IDL)

- Beispiel:

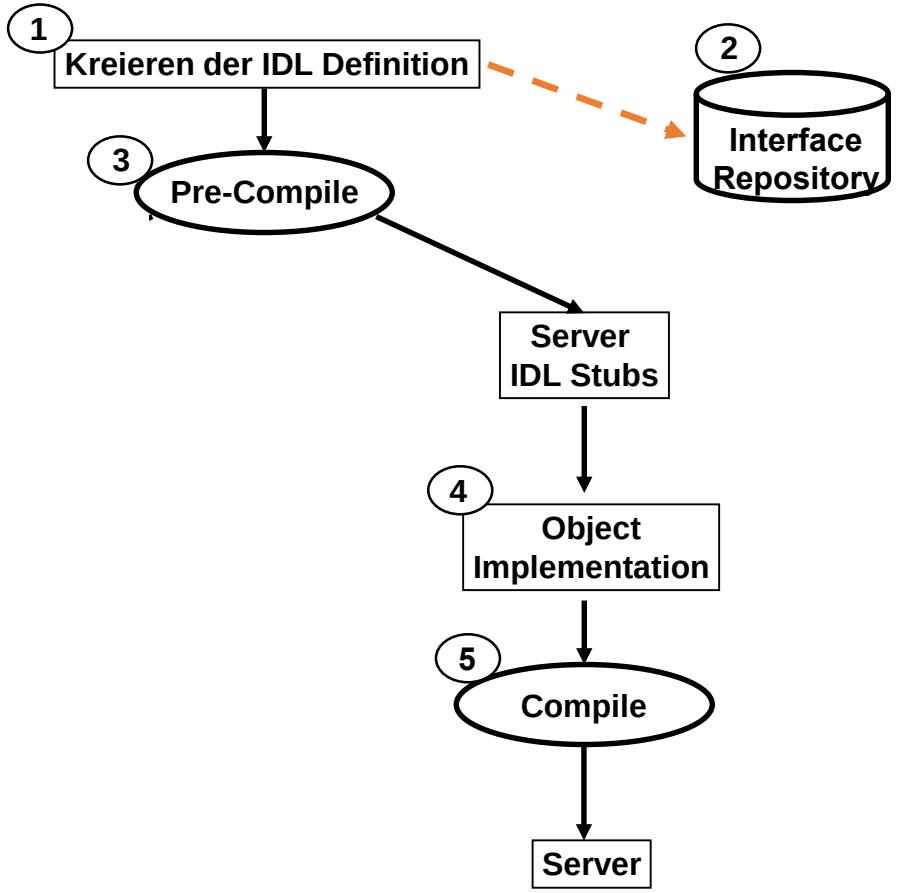
```
module MyAnimals {

        // Class definition of Dog
        interface Dog: Pet, Animal {

                attribute integer age;
                exception notInterested (string explanation);

                void bark(in short how_long)
                        raises (NotInterested);
                void sit(in string where)
                        raises (NotInterested);
                void growl(in string at_whom)
                        raises (NotInterested);
        };

        // class Definition of Cat
        interface Cat: Animal {

                void eat();
                void hereKitty();
        };
}; // end of MyAnimals
```

Definition eines Namensraums

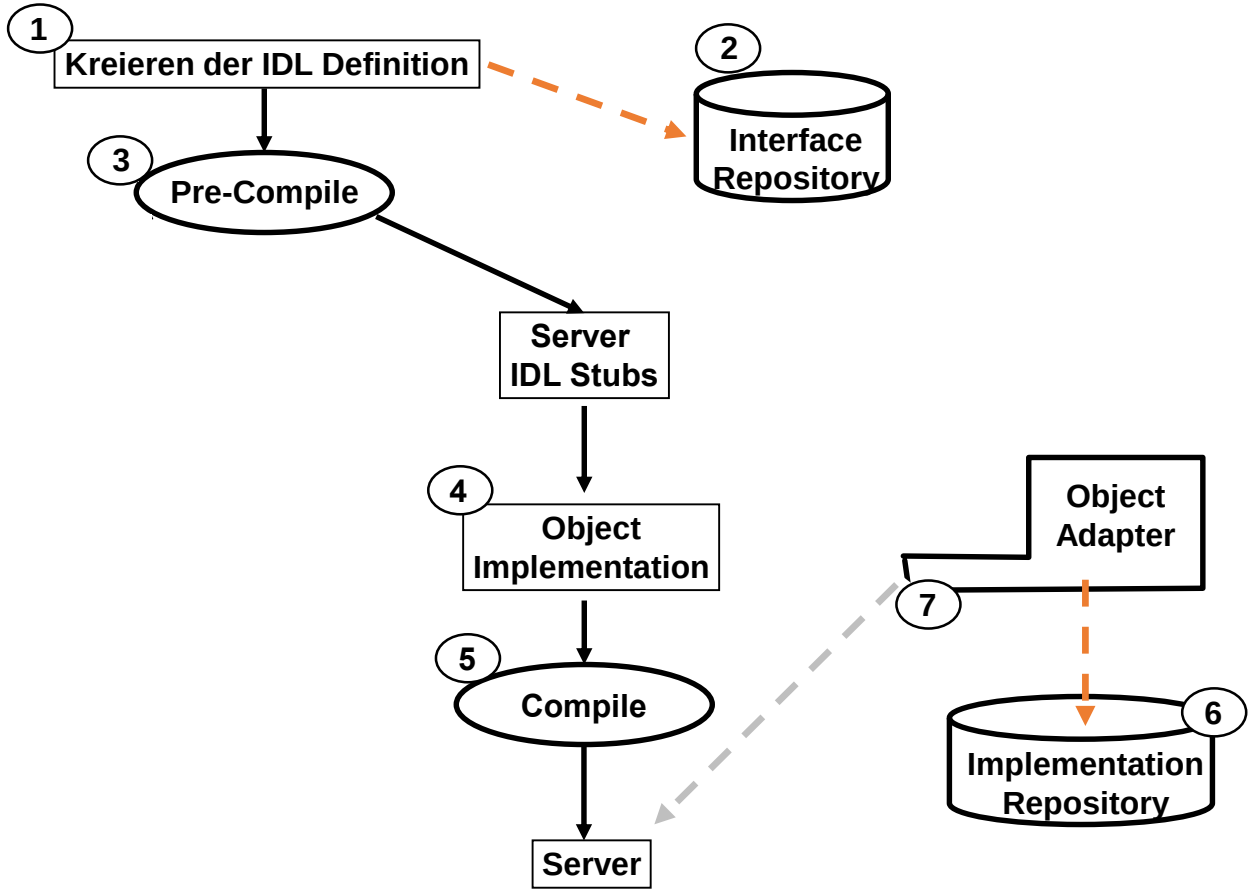
Definition einer Klasse

Definition einer Klasse

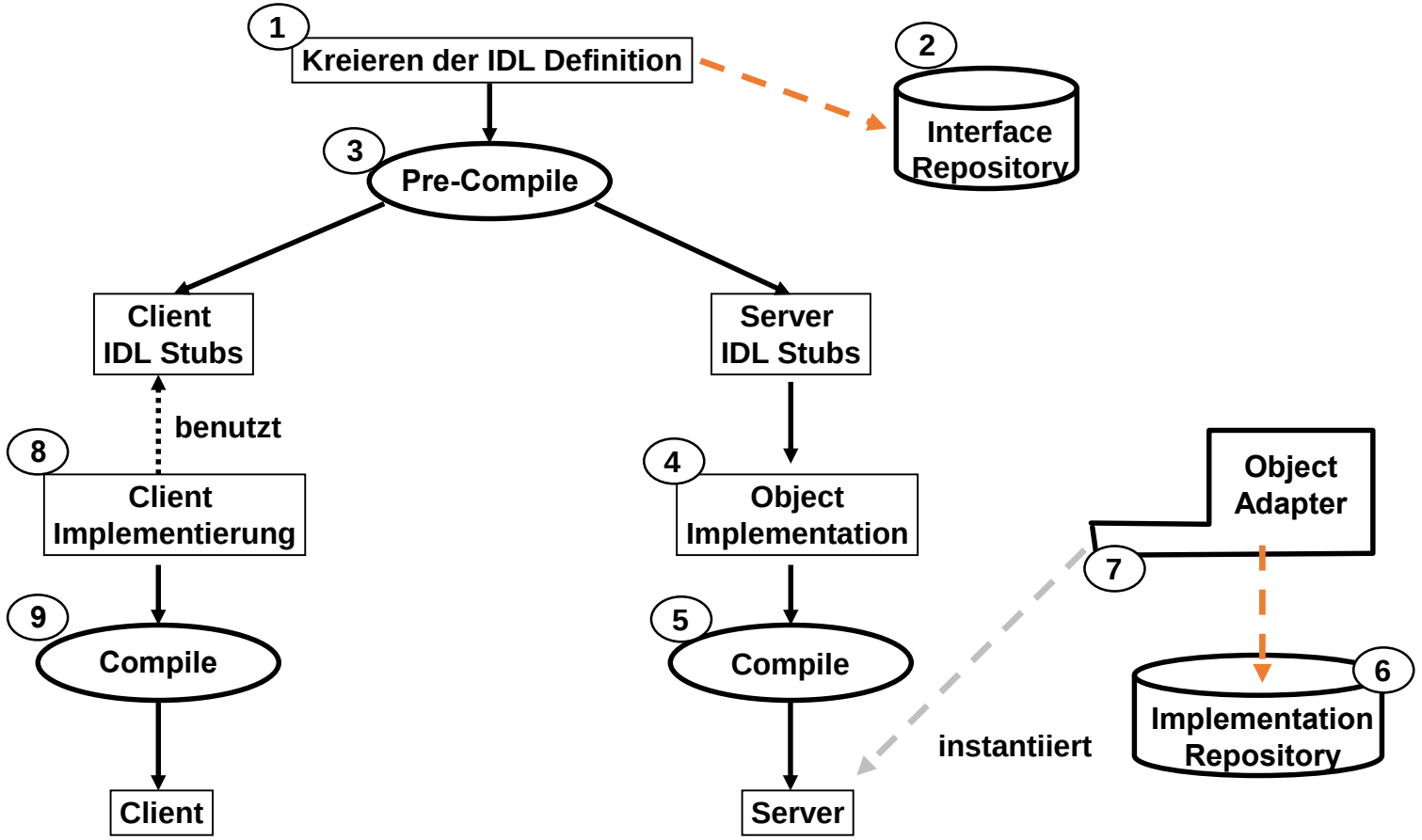
Verwendung einer IDL-Beschreibung



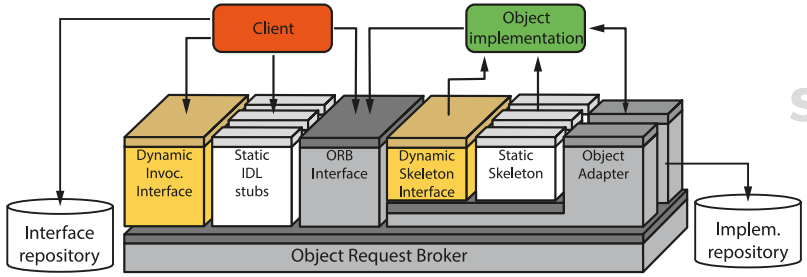
Verwendung einer IDL-Beschreibung



Verwendung einer IDL-Beschreibung

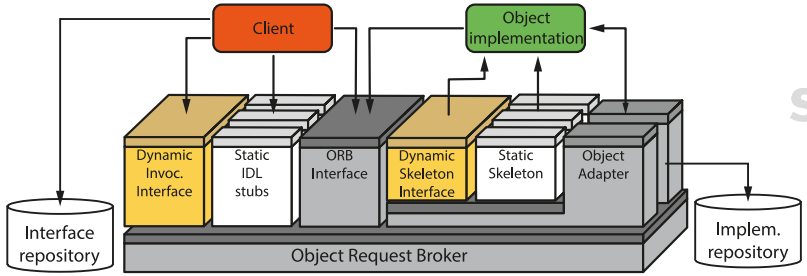


Dynamic Interfaces



- Die Aufgabe der eben besprochenen Stubs und Skeletons ist beschränkt auf die in einer Applikation zur IDL-Übersetzungszeit bekannten (statischen) Schnittstellen.
- CORBA erlaubt aber auch die dynamische Definition und Verwendung entfernter Objekte
  - auf der Client-Seite gibt es dazu das Dynamic Invocation Interface und das Interface Repository und
  - auf der Server-Seite gibt es das Dynamic Skeleton Interface und das Implementation Repository
- in diesen Fällen kann ein Client das Objekt und dessen Schnittstelle, also die Namen und Parameter der Methoden, über das Interface Repository abfragen, falls das Server-Objekt zuvor registriert wurde
- der Aufruf einer konkreten Methoden erfolgt dann über das Dynamic Invocation Interface
- das Dynamic Skeleton Interface unterstützt den Aufruf von Methoden auf Objekten, die keine IDL-Beschreibung besitzen; diese müssen sich zur Laufzeit beim Dynamic Skeleton Interface registrieren, das die Informationen im Implementation Repository ablegt

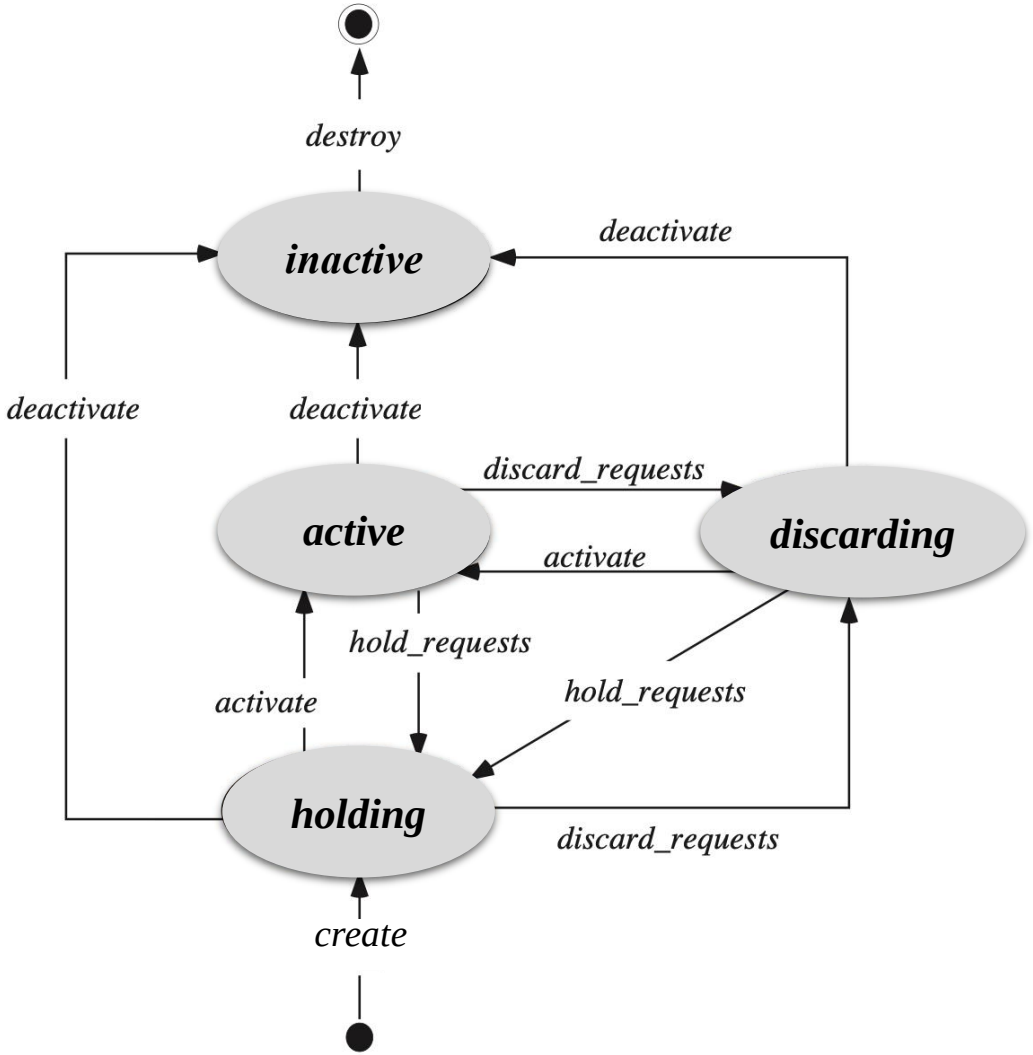
Portable Object Adapter



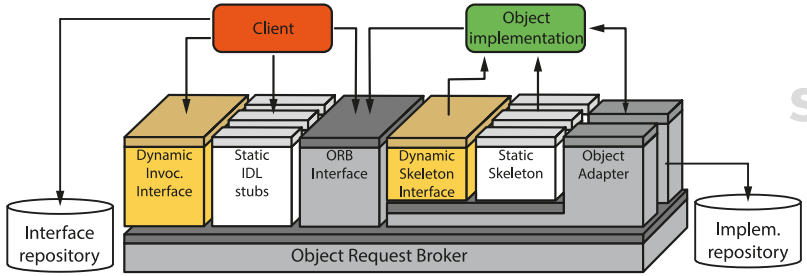
- Der Portable Object Adapter (POA) ist der Teil des ORB, der serverseitige Ressourcen für Skalierbarkeit verwaltet.
- Objektimplementierungen werden deaktiviert, wenn sie nichts zu tun haben und werden erneut aktiviert, sobald sie wieder benötigt werden, so dass die Hardwareressourcen besser ausgenutzt werden können.
- Der POA ist die primäre Schnittstelle, um Dienste des ORBs zu nutzen, zu denen u.a. folgende Dienste zählen:
  - Generieren und Interpretieren von Objektreferenzen,
  - Methodenaufruf,
  - Sicherheit von Aufrufen,
  - Aktivieren und Deaktivieren von Objekten und Implementierungen,
  - Zuordnen von Objektreferenzen zu Implementierungen und
  - Registrieren von Implementierungen
- Obwohl es sich architektonisch um ein separates Teil des ORBs handelt und ORBs potentiell mehr als einen Objektadaptertyp haben können, ist der POA keine Software, die Sie separat vom ORB verfügbar ist, da die Schnittstellen, die den ORB und den POA verbinden, proprietär sind.
- Der POA ist der zweite Objektadapter, den OMG entwickelt hat, da der Vorgänger, der Basic Object Adapter (BOA) nicht alle Unternehmens- und Internetanforderungen erfüllen konnte.

Portable Object Adapter

- **Der Object Adapter** verwaltet quasi den **Lebenzyklus der Objektimplementationen.**
- Er definiert vier **Zustände**, in die ein **Objekt** während seiner Lebensdauer eintreten kann:
  - **holding**: überbrückt nach dem Start die Verzögerung bis Objekt im Zustand active und arbeitsfähig ist. Zwischenzeitliche Anfragen werden in einer Message Queue zwischen gespeichert.
  - **active:** Normalzustand, Objekt beantwortet Anfragen.
  - **discarding**, in diesem Zustand werden Anfragen nicht bearbeitet und eine Exception wird zurückgeschickt. Der Zustand wird beispielsweise bei Überlastung eingesetzt.
  - **inactive**, in diesem Zustand können Objektreferenzen zwar genutzt werden; Methodenaufrufe werden aber vom Object Adapter zurückgehalten und blockiert, bis das Objekt aktiv ist. Dieser Zustand wird üblicherweise für Verwaltungsaufgaben eingesetzt wie z.B. um eine Objektimplementierung auszutauschen oder vor einem Server-Shutdown.



ORB Interface



- das ORB Interface stellt einige nützliche Hilfsfunktionen für lokale Dienste bereit, wie z.B.:
  - Initialisierung des Object Request Broker
  - Initialisierung einer Client/Server-Anwendung
  - Programmierschnittstelle zum Interface Repository
    - Konvertierung einer Objektreferenz in eine Zeichenfolge und umgekehrt
    - Erstellen von Argumentlisten, die für dynamische Methodenaufrufe benötigt werden
  - ...

- OMG Einführung in CORBA: http://www.omg.org/gettingstarted/corbafaq.htm
- Java-Beispiel: https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

Common Object Services

Wichtige (Unterstützungs-)Services

- Naming Service der es Serverobjekten ermöglicht, mittels eines festgelegten Namens angesprochen zu werden.
- Trading Service ermöglicht die Objektsuche zur Laufzeit auf Basis von Objekteigenschaften
- Event Service ermöglicht lose gekoppelte, ereignisbasierte n:m Kommunikation.
- Life Cycle Service stellt Operationen zum Kopieren, Verschieben und Löschen von Objekten zur Verfügung.
- Relationship Service ermöglicht die Modellierung von Beziehungen zwischen Objekten.
- Weitere Dienste: Externalization Service, Persistent Object Service, Concurrency Control Service, Transaction Service, Property Service, Licensing Service, Object Collection Service, Query Service, Time Service, Security Service, Notification Service als Erweiterung zum Event Service.

Common Object Services (Überblick)

Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, stacks, etc.
Query	Facilities for querying collections of objects in a declarative manner; various collections possible
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication (e.g. filtering)
Externalization	Facilities for marshaling and unmarshaling of objects (like Java's serialization)
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services an object has to offer based on constraints
Persistence	Facilities for persistently storing objects; persistence transparency is provided
Relationship	Facilities for expressing relationships between objects (compare database schemes)
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

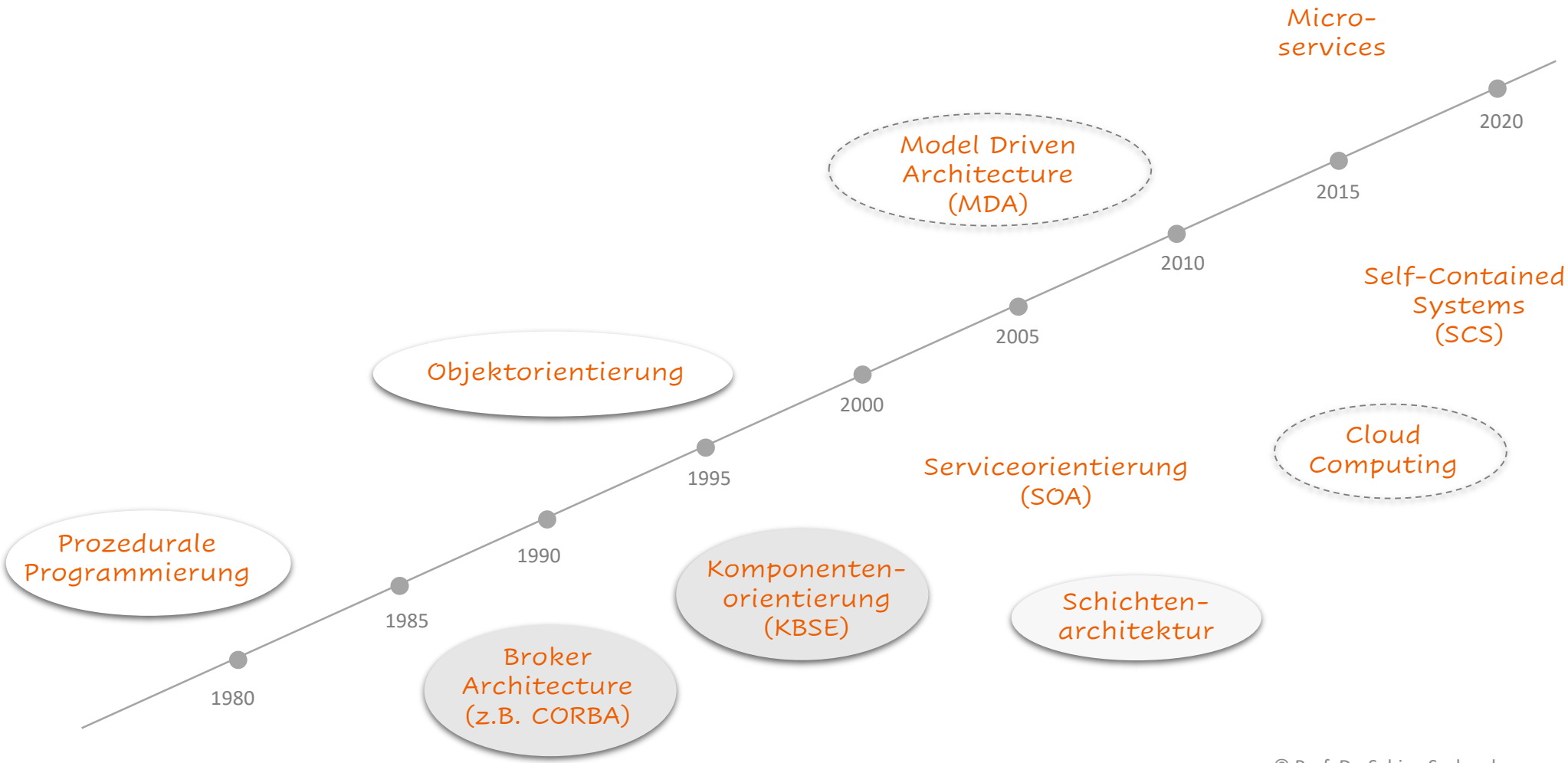
Fachhochschule
Dortmund
University of Applied Sciences and Arts

Komponentenbasierte Systeme (KBSE)

am Beispiel EJB

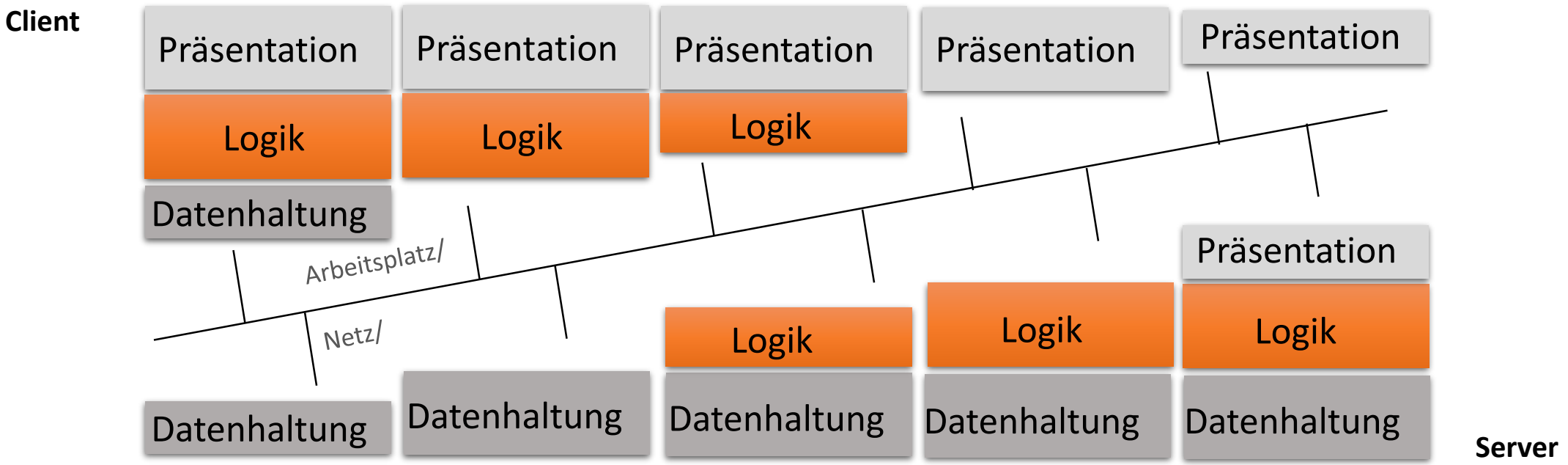
Entwicklungs- und Architekturtrends im Zeitverlauf

Unvollständige Auswahl großer Softwaretechnik-Trends

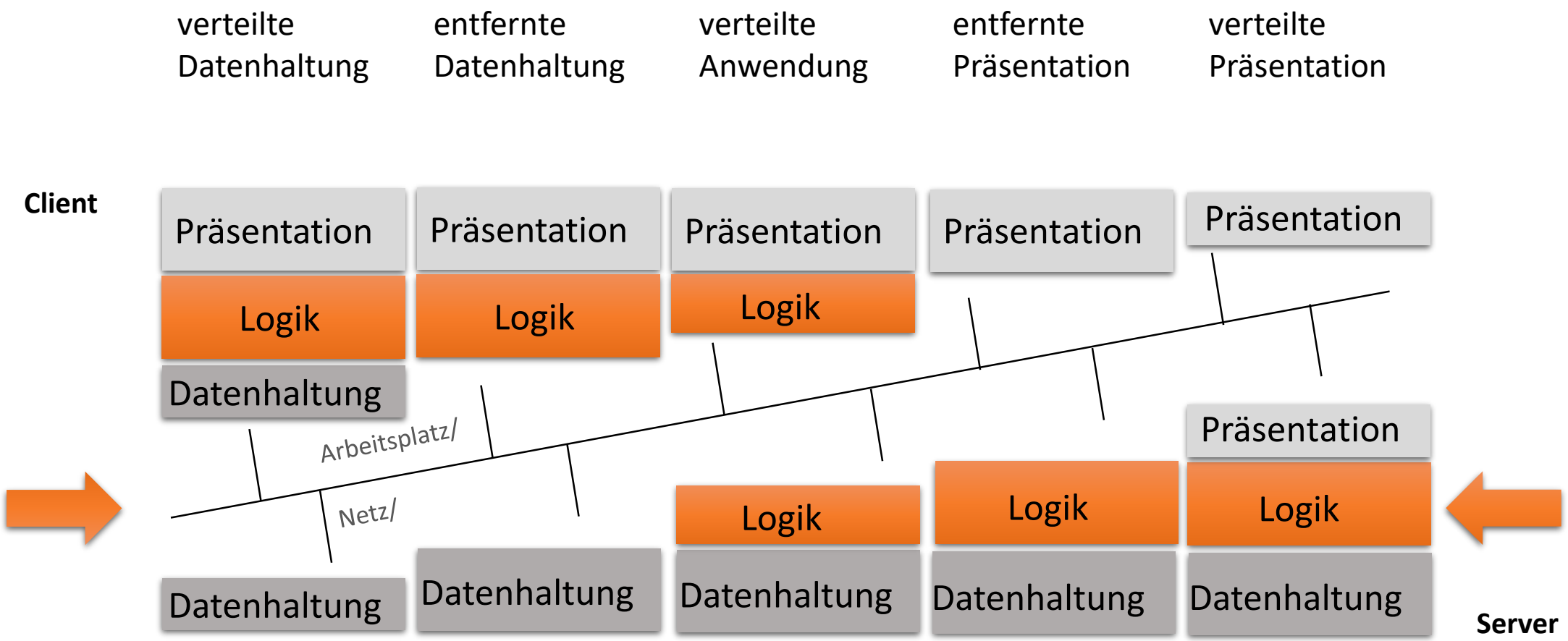


Schichtenmodell (2-Tier bzw. Client/Server)

verteilte Datenhaltung | entfernte Datenhaltung | verteilte Anwendung | entfernte Präsentation | verteilte Präsentation



Schichtenmodell (2-Tier bzw. Client/Server)

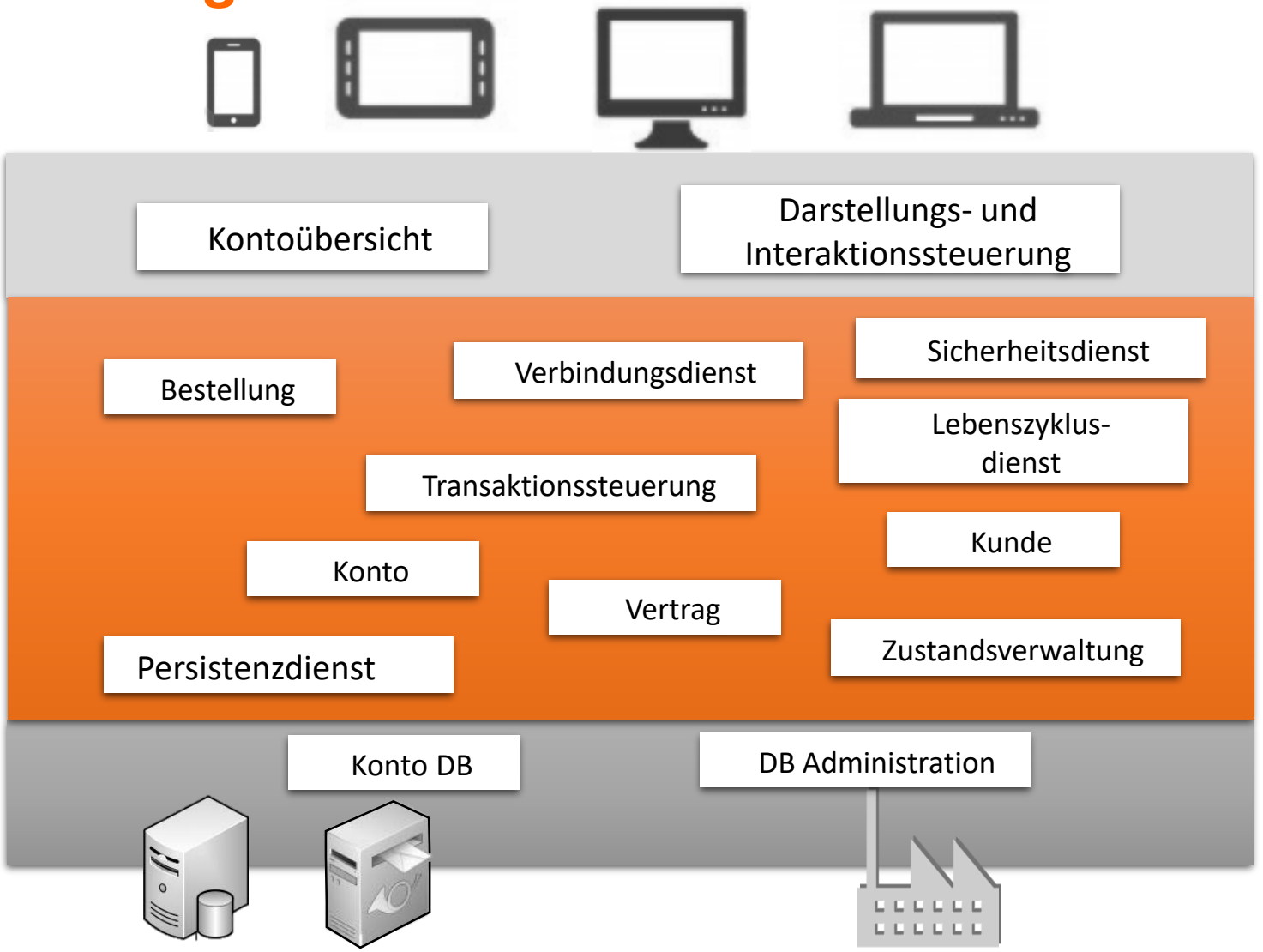


Trennung der Verantwortlichkeiten

viele verschiedene Clients

viele verschiedene Datenquellen

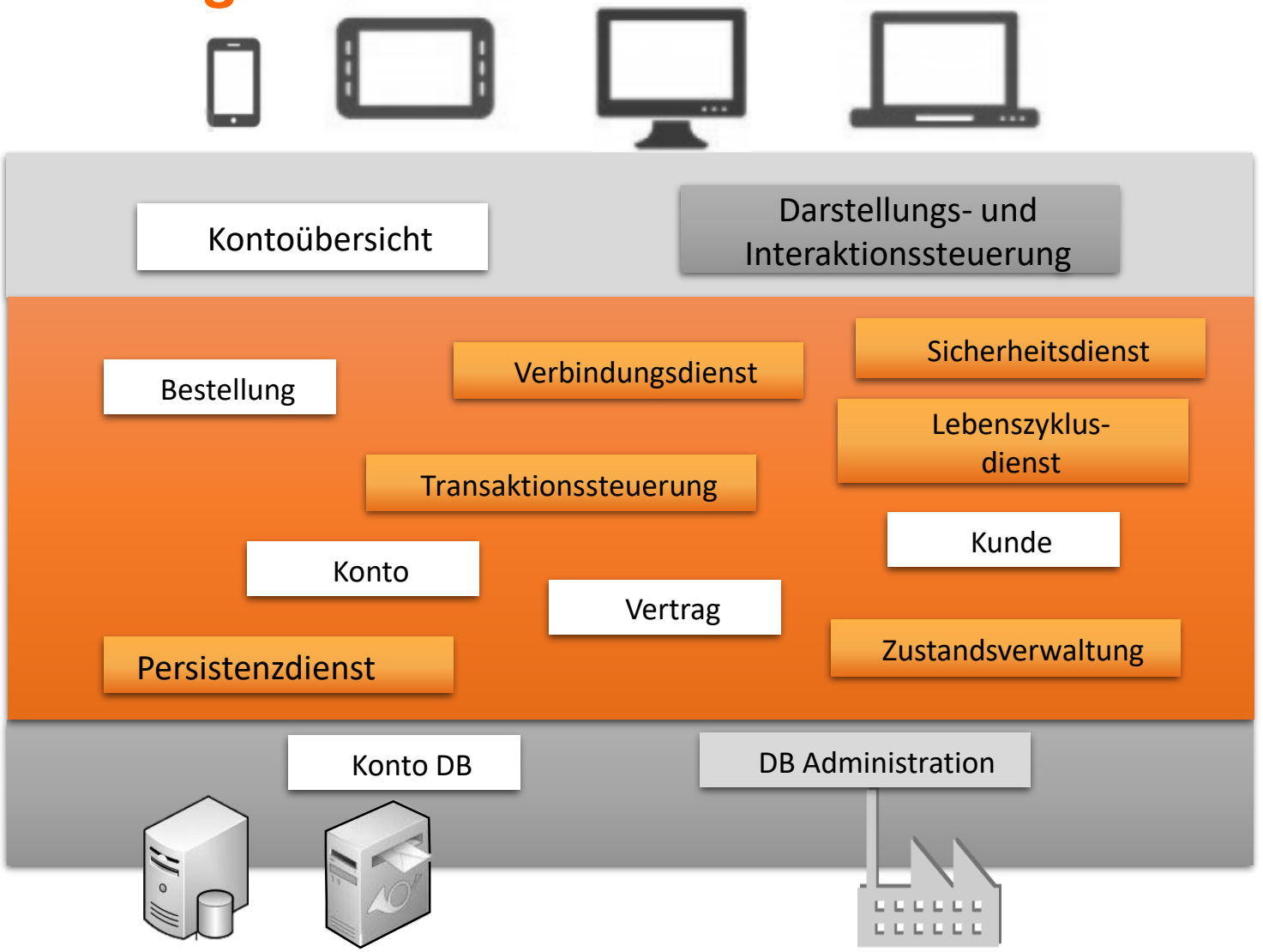
Trennung der Verantwortlichkeiten



viele verschiedene Clients

viele verschiedene Datenquellen

Trennung der Verantwortlichkeiten

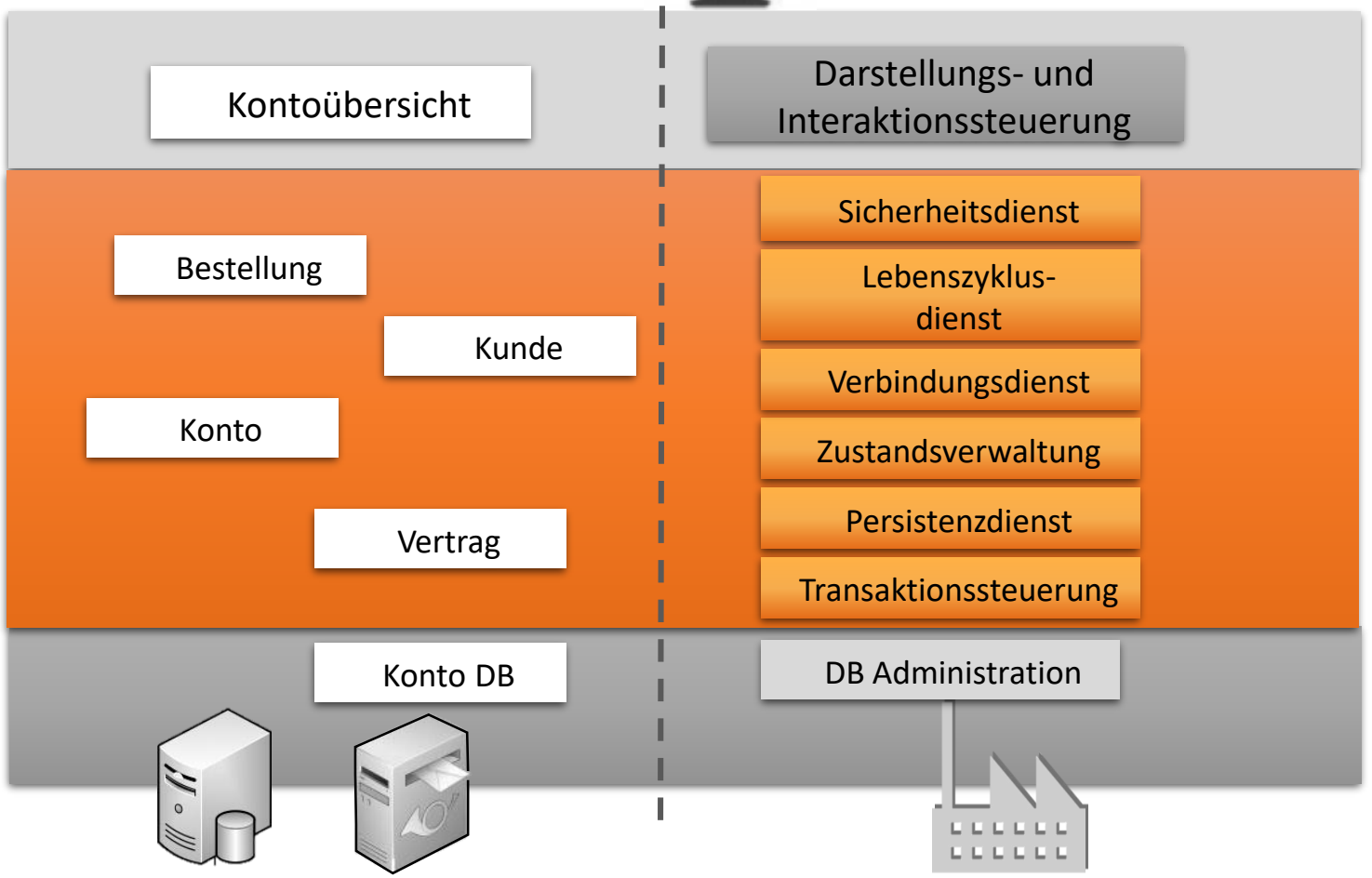


viele verschiedene Clients

viele verschiedene Datenquellen

Trennung der Verantwortlichkeiten

viele verschiedene Clients

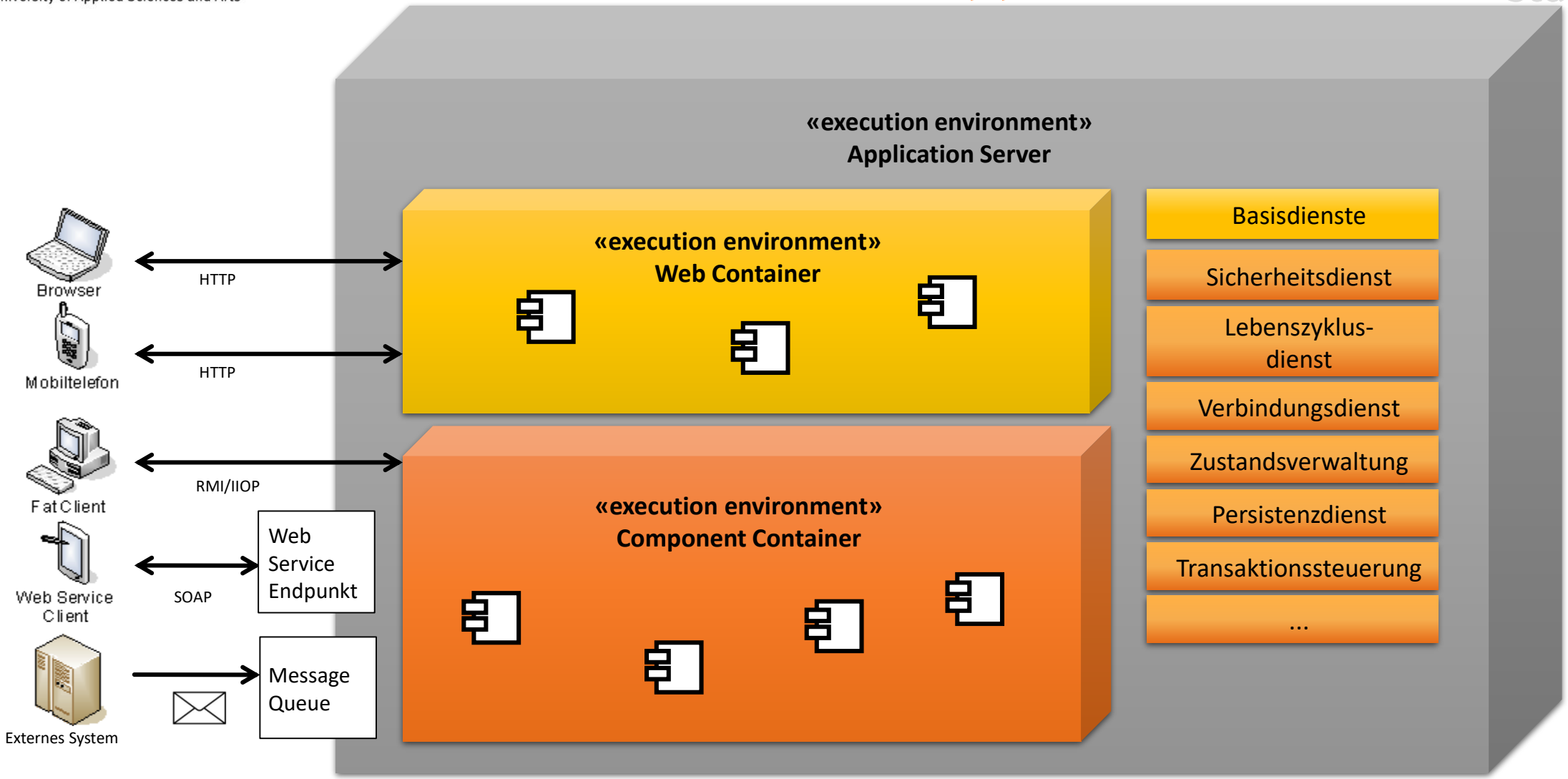


fachlich

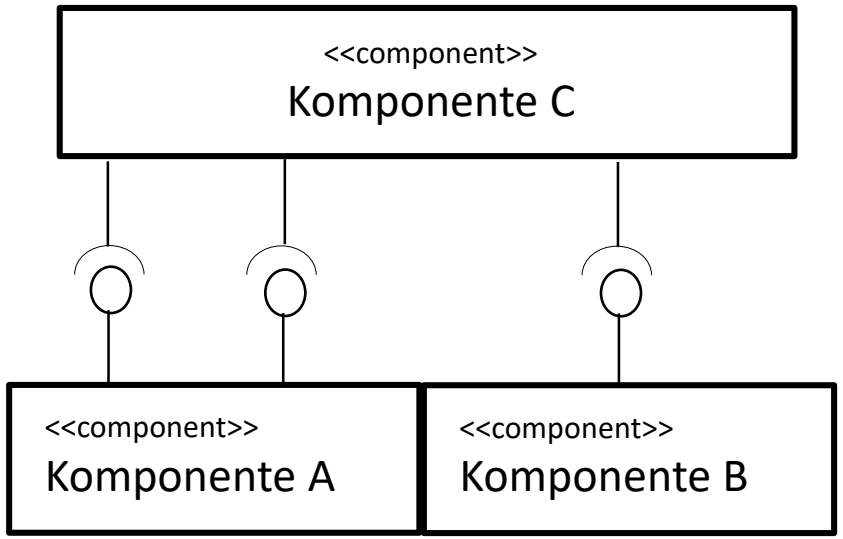
technisch

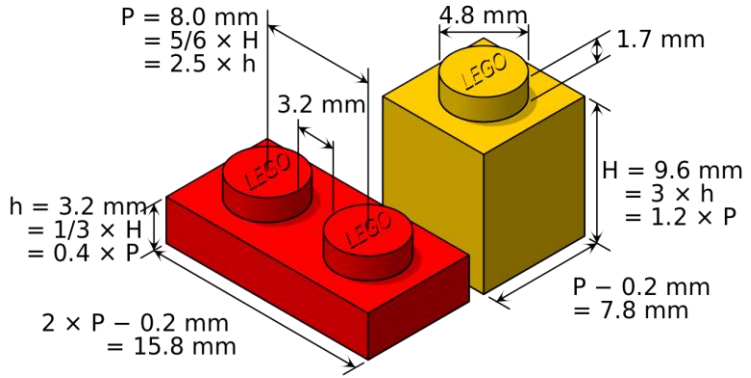
viele verschiedene Datenquellen

Anwendungsserver (engl. Application Server)

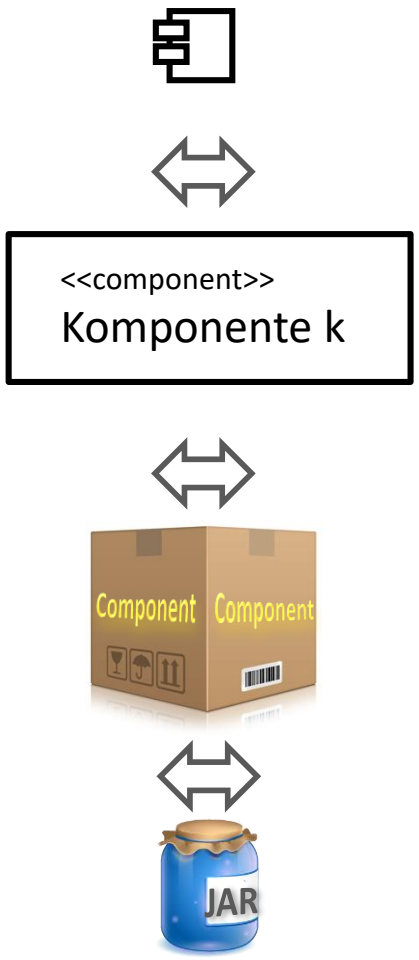


Komponente – Lego Metapher



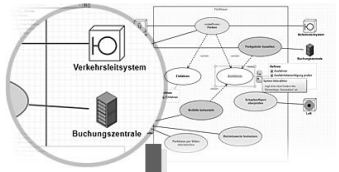


Eine Komponente ist ...

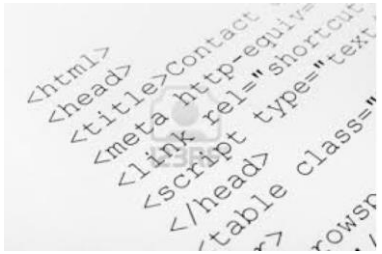


- geschlossener Teil eines Systems
- ohne starke Abhängigkeit
  - von anderen Komponenten
    - geringe Kopplung, hohe Kohäsion
  - von der Umgebung
    - systemunabhängiger Softwarebaustein
    - Interoperabilität:
      eine Komponente kann über Adressbereiche, Netzwerke, Sprachen, Betriebssysteme und verschiedene Tools hinweg aufgerufen werden
- mit expliziten Abhängigkeiten
- Komponenten werden oft als Archive verpackt

Eine Komponente ...

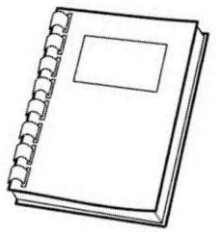


... hat eine Spezifikation



... hat eine Implementierung

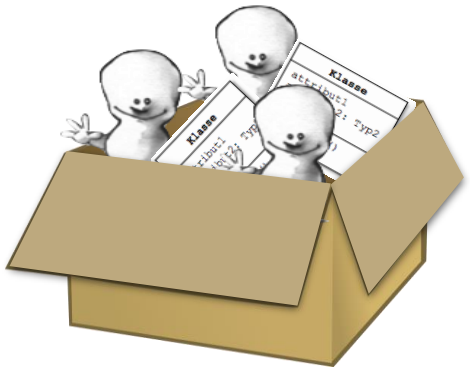
... kann ausgeliefert werden



... entspricht einem Standard
(Corba/CCM, .Net/COM+, Java EE/EJB)

... kann verpackt werden

Komponenten-Frameworks und -standards



- es gibt Komponentenmodelle als Frameworks und Standards in verschiedenen Programmiersprachen und auf verschiedenen Plattformen
- Beispiele für bekannte Vertreter:
  - JavaEE/EJB
  - Microsoft .Net/COM+
  - Corba/CCM
- im weiteren Folgenden wird exemplarisch kurz auf den JavaEE/EJB-Ansatz eingegangen


Singleton

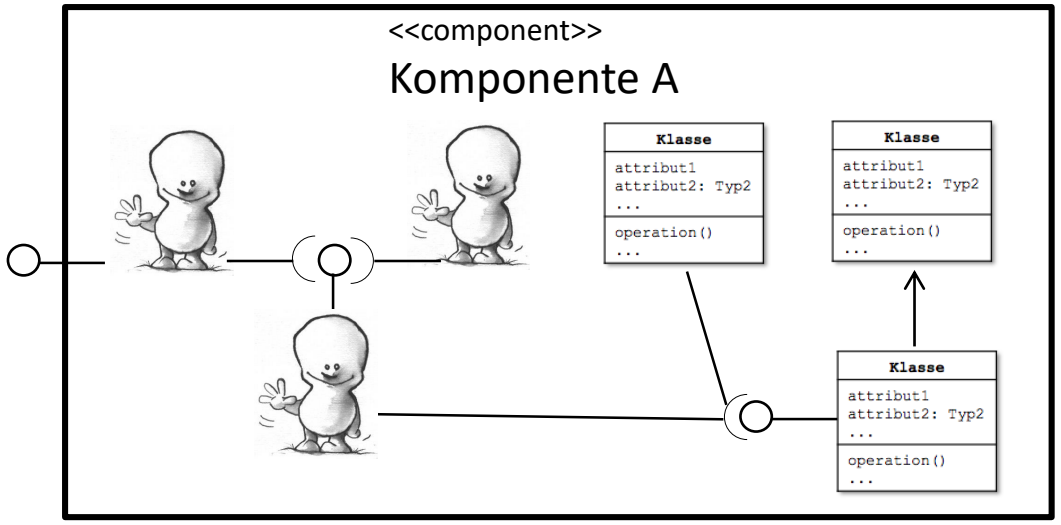

Stateless Session


Stateful Session


Message Driven

Java Enterprise Beans (EJBs)

Komponente (EJB-Modul)



Interfaces (lokal, remote), Beans und (Hilfs-)Klassen

Container

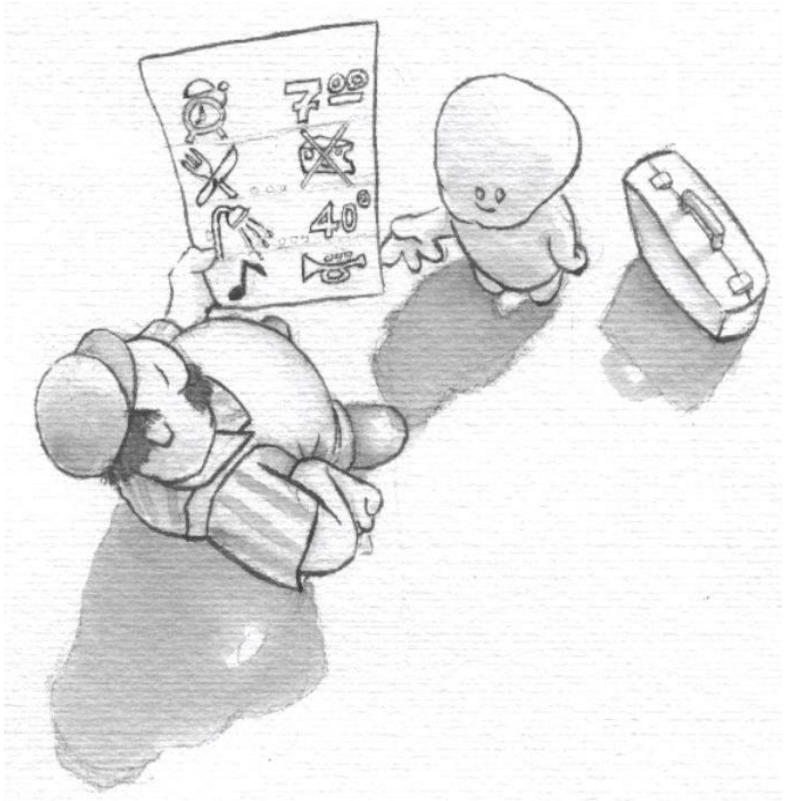
- stellt Dienste bereit
  (Implementierungen übergreifender Anforderungen)

- umgibt Komponenten bzw. die enthaltenen Beans, gibt Clients die Illusion von Komponenten, die eng mit ihrer Umgebung verknüpft sind

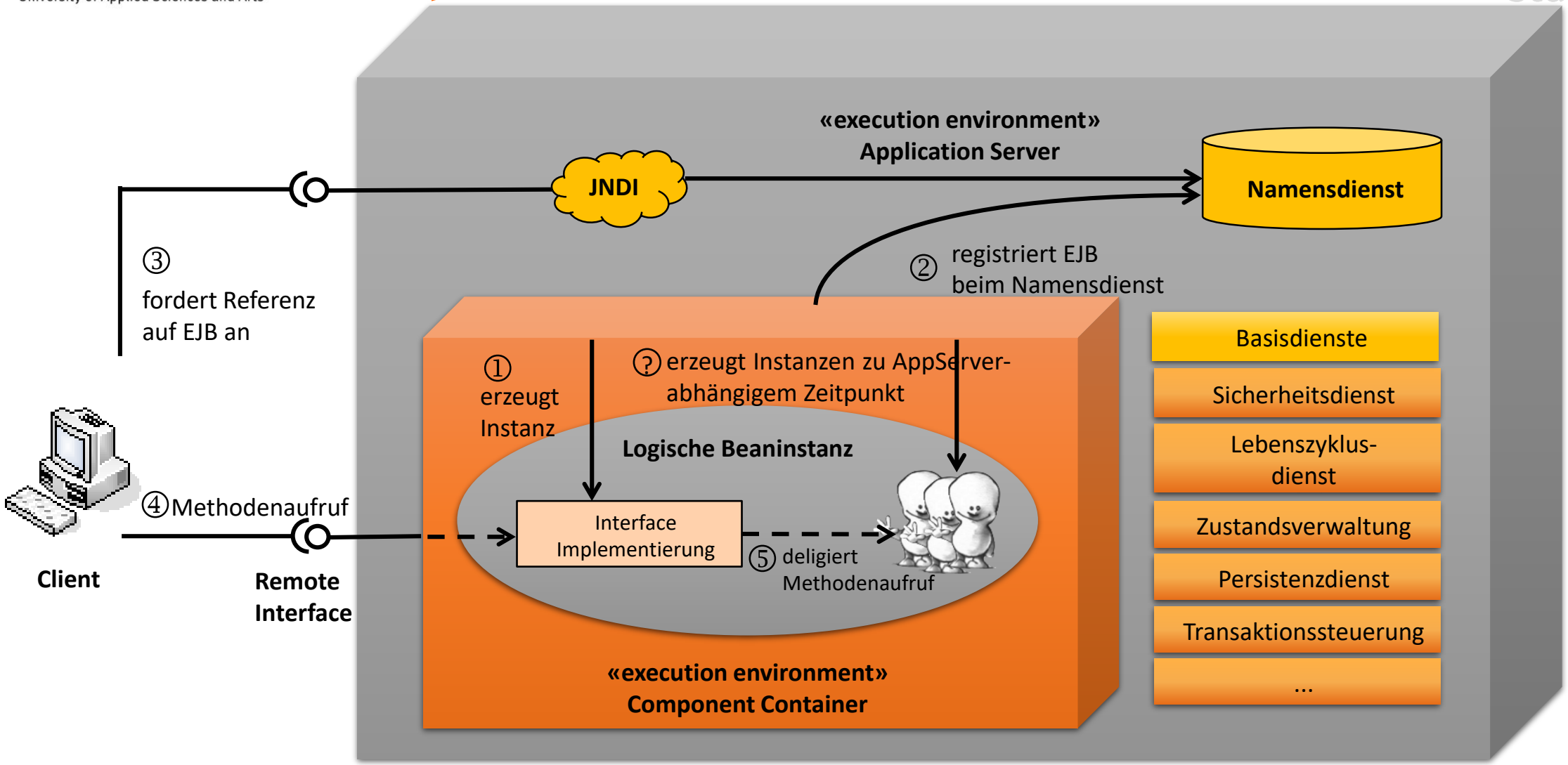
Wo die kleinen Beans wohnen …

Deployment Descriptoren

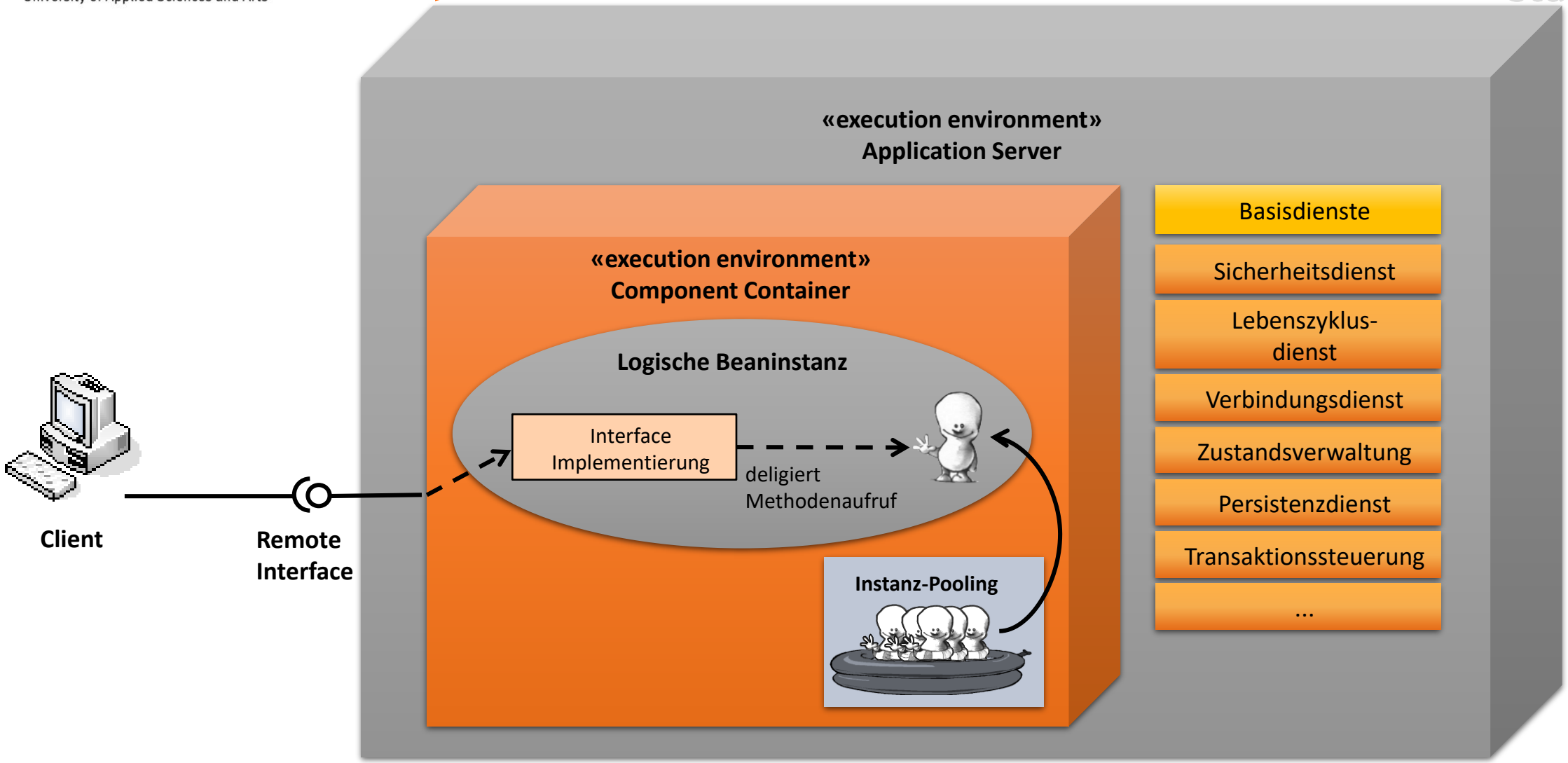
- Der Container ist verantwortlich für die Unterstützung nicht-funktionaler Anforderungen oder zur Verknüpfung von Beans.
  - Er braucht dazu Informationen darüber:
    - was unter Transaktionsschutz laufen soll
    - Sicherheitseinstellungen
    - gesendete Events
    - anpassbare Eigenschaften
- Eine Bean muss auf Nachfrage folgende Informationen über sich liefern, wie z.B:
  Schnittstellen, Eigenschaften, Events, etc.



Beispiel: Aufruf über Remote Interface



Beispiel: Stateless Session Beans - Pooling

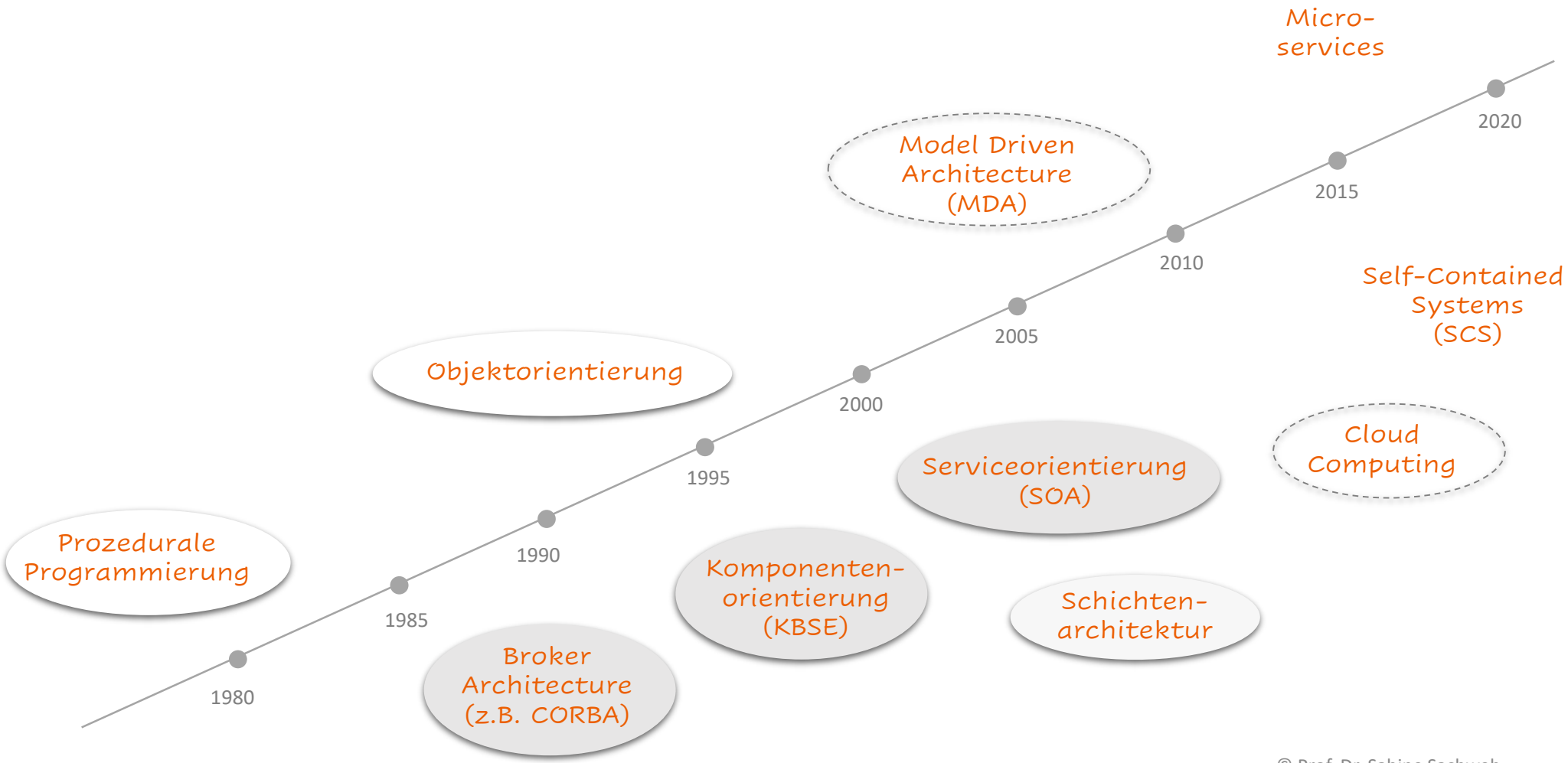


Fachhochschule
Dortmund
University of Applied Sciences and Arts

Service Orientierte Architekturen (SOA)

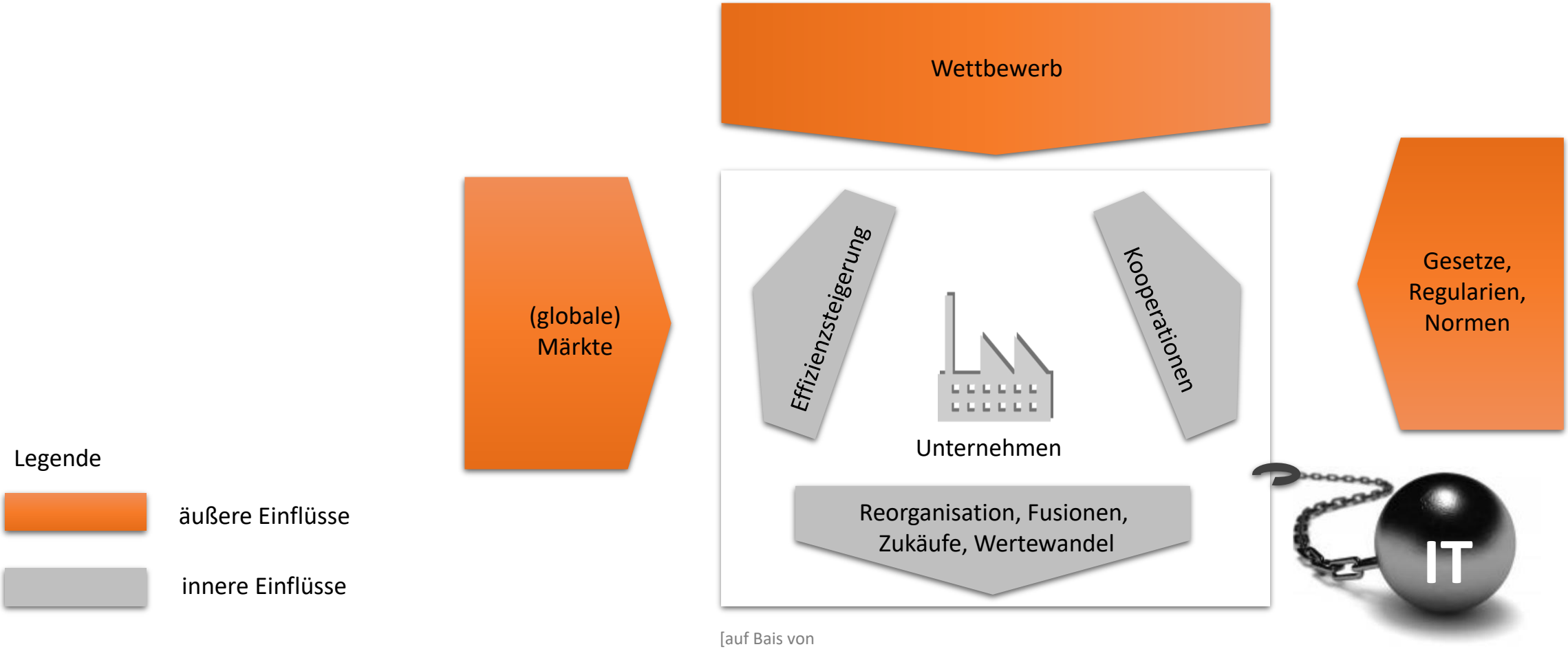
Entwicklungs- und Architekturtrends im Zeitverlauf

Unvollständige Auswahl großer Softwaretechnik-Trends



Motivation

Unternehmen im Spannungsfeld

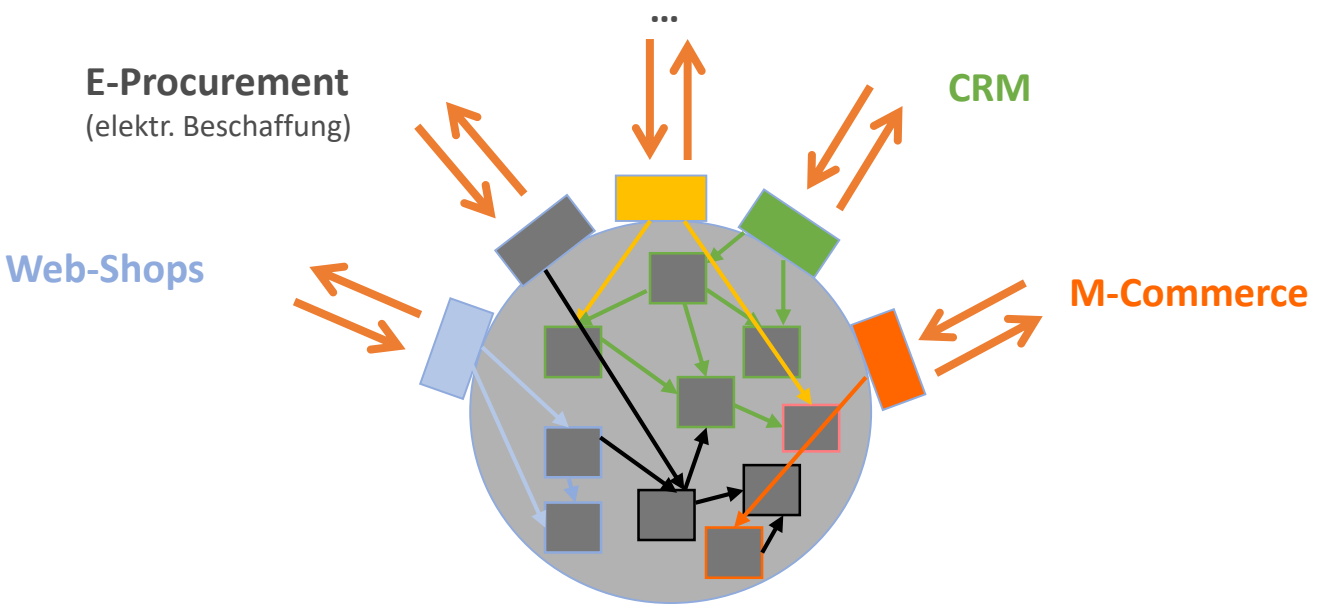


[auf Bais von
http://www.gernotstarke.de/aktuelles/aktuelles-gst_assets/SOATEC06-GStarke-Enterprise-Architecture.pdf]

Motivation

Unternehmensinterne Abläufe und externe Angebote

- Software Systeme müssen den ständig neuen Herausforderungen bzw. ständig wechselnden „Trends“ gewachsen sein



Legende

extern angebotene Dienste

unternehmensinterne Abläufe

Serviceorientierte Architektur (SOA)

Begriffsdefinitionen (viele verschiedene, teils gegensätzliche Definitionen

- Der Begriff „serviceorientierte Architektur“ wurde **1996 von** dem Marktforschungs-unternehmen **Gartner** erstmalig genutzt in der Research Note SPA-401-068.
- **SOA ist ein Paradigma für die Organisation und Nutzung verteilter Funktionalitäten,** die dezentral verwaltet sein können. → Standardisiert [1)]
- SOA ist also ein Software Design Ansatz mit dem Grundgedanken, dass **Geschäftsprozesse im Vordergrund** stehen. [2)]
- SOA soll die **flexible Anpassung von Geschäftsprozessen** an die sich ständig wandelnden Umweltbedingungen der Unternehmens ermöglichen. [2)]
- Dies erfolgt durch eine **Architektur grobkörniger Services** die miteinander **kommunizieren** und **flexibel zusammengestellt** werden können. [2)]

1) [http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf]
2) [http://www.till-rausch.de/assets/baxml/soa_akt.pdf]

Dimensionen einer SOA

1. Ein unternehmensweiter IT-Ansatz

- Fokus auf Business/IT-Alignment
- Modularisierung von Anwendungen
- Von der Anwendungs- zur Service-Landschaft
- Governance-Fokus

2. Eine Softwarearchitektur

- Services mit formal beschriebenen Schnittstellen
- Wiederverwendung durch Aufruf
- Trennung von Schnittstelle und Implementierung
- Realisierbar mit beliebigen Technologien

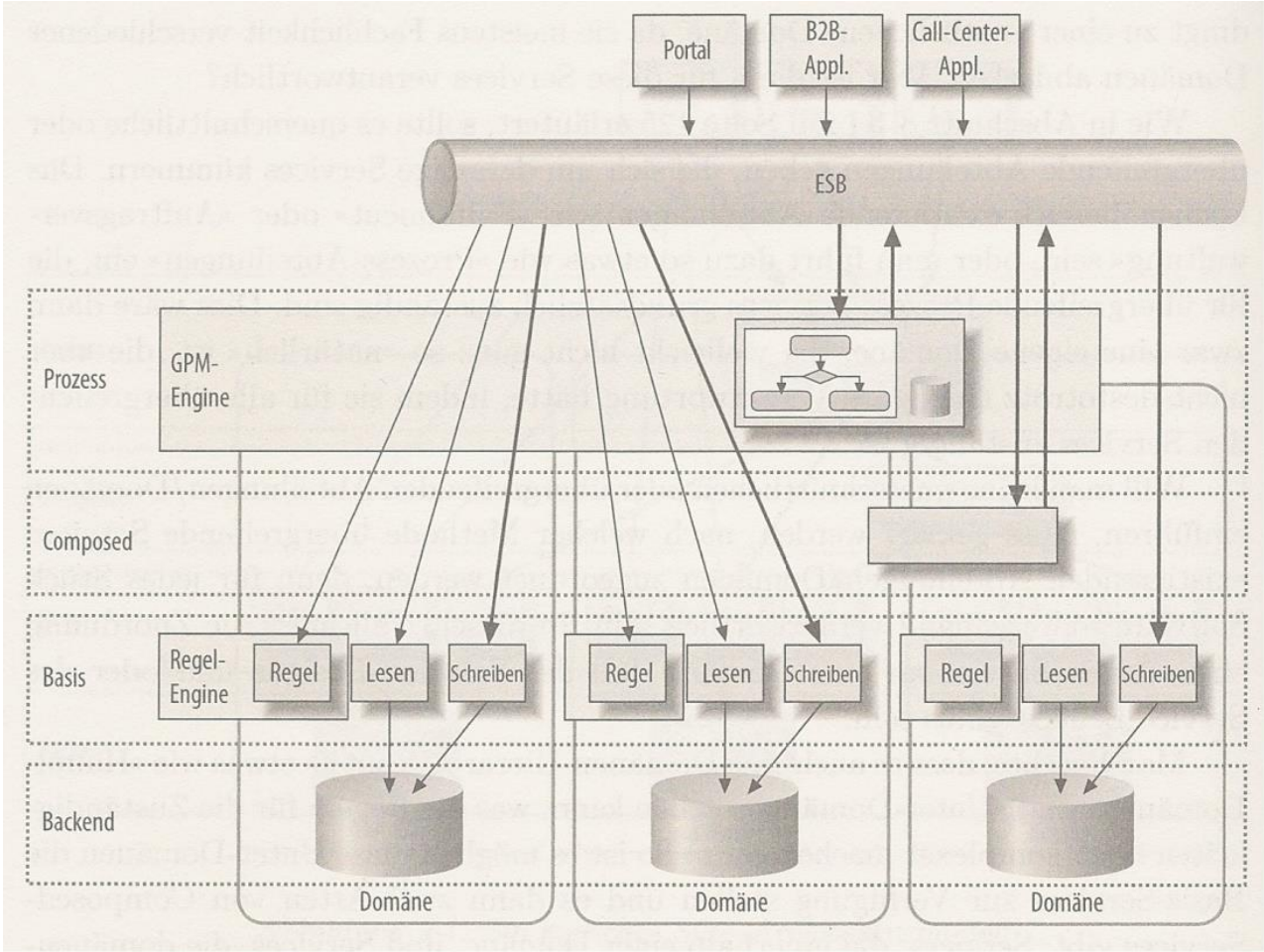
3. Das Konzept hinter SOAP/WSDL/WS-*

- Selbstbeschreibende Nachrichten (XML, XSD)
- Envelope/Header/Body-Konzept
- Dynamic lookup via Registry
- Policy-getriebene Konfiguration
- Anforderungsgetrieben kombinierbare Standards und Spezifikationen

Eine Softwarearchitektur

Zerlegen und Verdrahten von Anwendungen

- Geschäftsobjekte werden von den **Backend-Systemen** der Domänen verwaltet
- **Basis Services** werden von der anbietenden Domäne verwaltet
- (Geschäfts-)Logik auf Ebene der Basis-Services wird mit einer **Rule-Engine** implementiert
- **Composed Services** sind Domänen-übergreifend definiert
- (Geschäfts-)**Prozess-Services** werden von einem GPM-Tool bzw. einer **Process Engine** verwaltet und ausgeführt

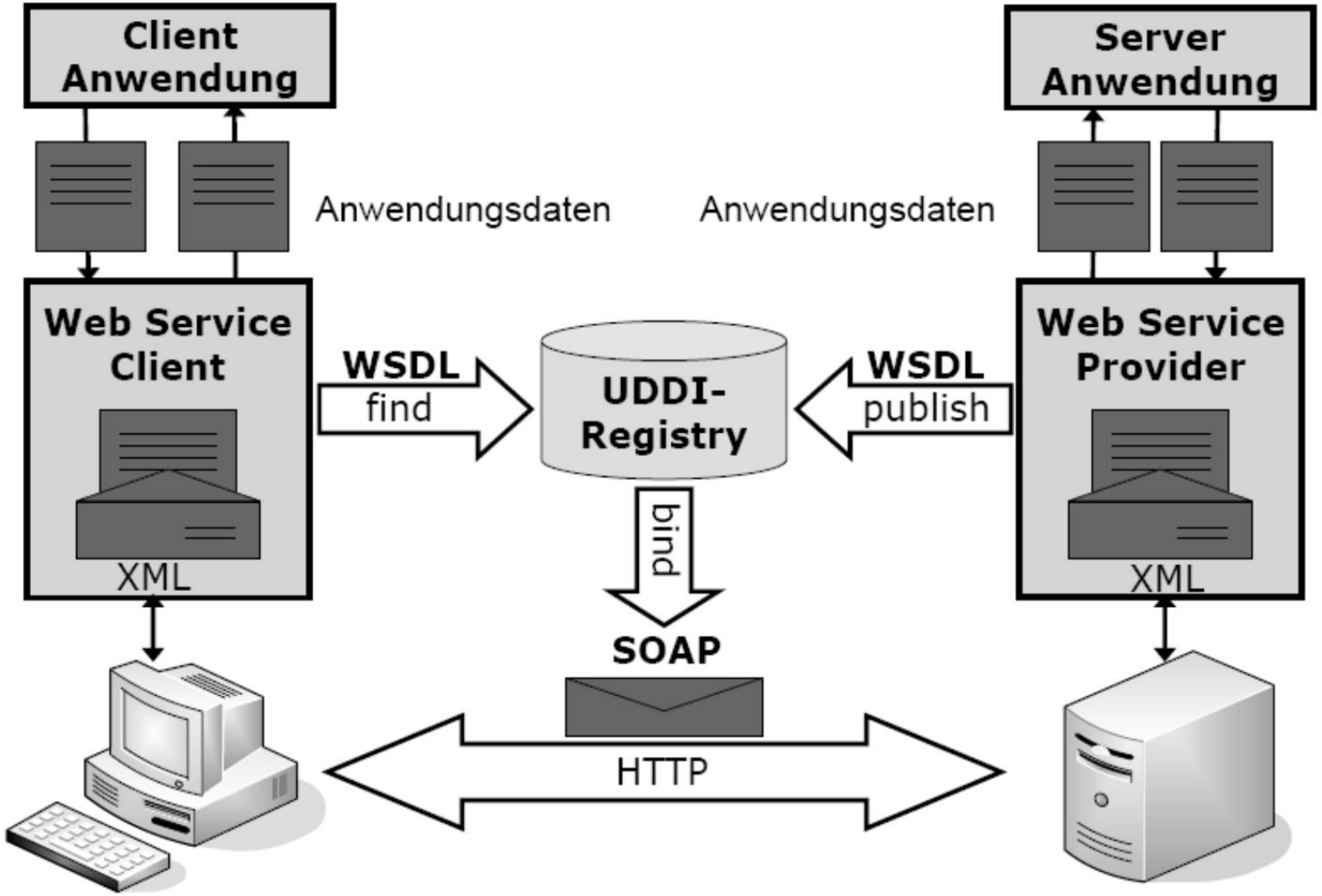


[Nicolai Josuttis (2008) :SOA in der Praxis: System-Design für verteilte Geschäftsprozess]

Das Konzept hinter SOAP/WSDL/WS-*

- Mit WSDL (Web Service Description Language) wird die Schnittstelle eines Web-Services XML-basiert spezifiziert,
- via SOAP (früher: Simple Object Access Protocol) werden Prozedurfernaufrufe übermittelt und
- mit UDDI (Universal Discovery, Description and Integration), einem zentralen Verzeichnisdienst für angebotene Web Services, können andere Web-Services aufgefunden werden.

Zusammenspiel der Standards



SOA und Technologie-Bindung

- **Komplizierte SOA-Modelle** und der **unübersichtliche Wald von Standards**, die in Zusammenhang mit Web Services entwickelt werden, können die **zentralen Vorteile** von SOA verdecken.

- Die **wahren Stärken des SOA-Modells** liegen jedoch
  - in der **einfachen Erweiterung der üblichen Schichtung** von Anwendungen und Architekturen **um eine Service- und Orchestrierungsschicht** und
  - in der Möglichkeit, mit Hilfe von SOA, **Ordnung in eine bestehende heterogene Systemlandschaft zu bringen**.

- Das **SOA-Modell sollte technologieneutral formuliert werden** und gleichzeitig den Einsatz einzelner Produkte oder der gesamten SOA-Produktpalette eines oder mehrerer Hersteller gestatten.

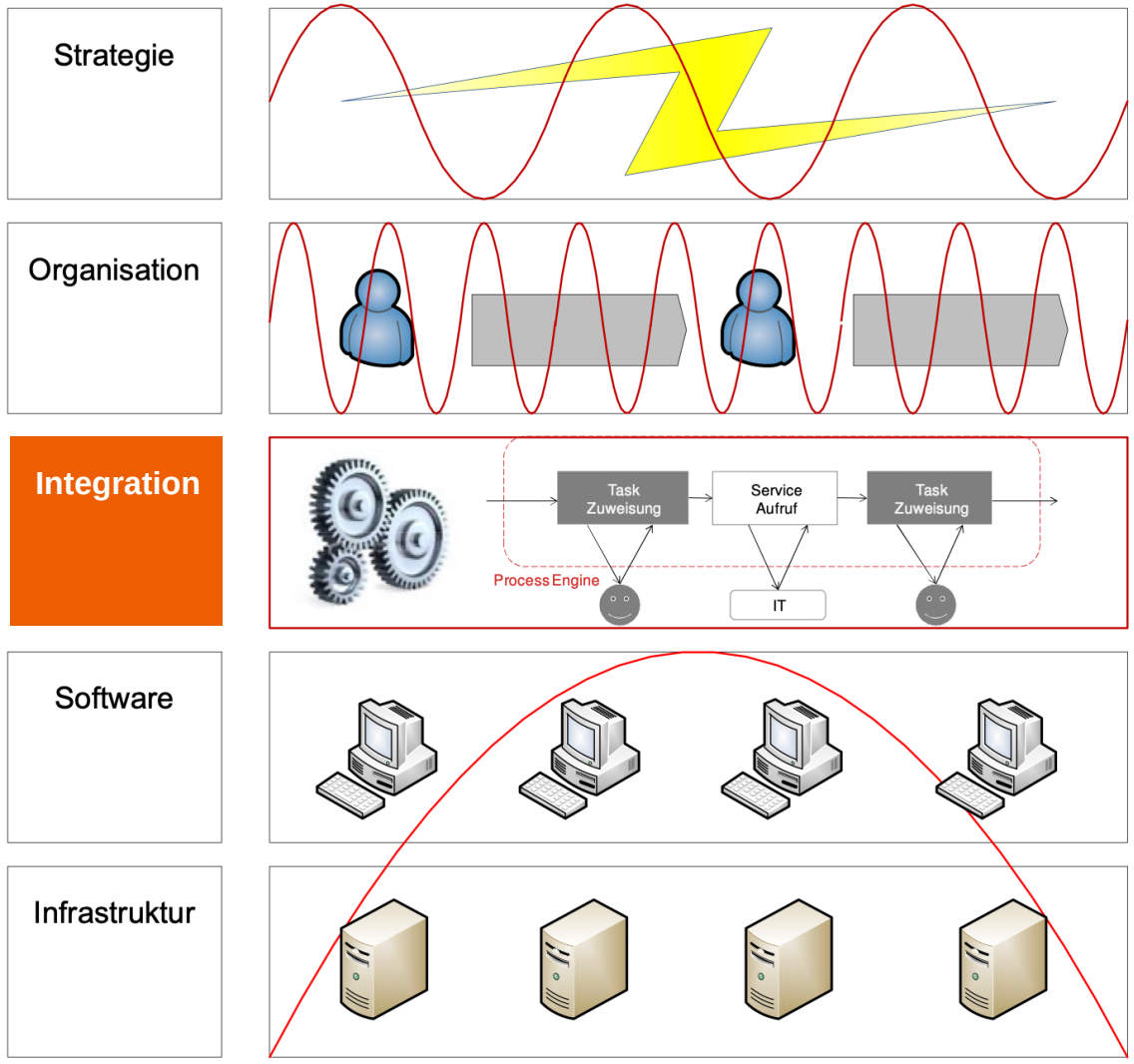
SOA und Technologie-Alternativen

- Die **Ideen der SOA** können auch übernommen werden **ohne WSDL, SOAP und UDDI** zu verwenden.
- Was müssen auch **keine Infrastruktur-Plattformen und ESBs** eingesetzt werden
- Statt **SOAP-Webservices** können auch REST-Webservices verwendet werden, um Anwendungen zu integrieren.
- Somit lassen sich SOAs auch **weniger Technologie-fokussiert** umsetzen.

Service ≠ Service ≠ Service

- **Strategie-Ebene**: 1-2 Jahre
- **Prozess-Ebene**: 3-6 Monate

- **Software-Ebene**: 6-10 Jahre



[In Anlehnung an: Prof. Dr. Robert Winter, Universität St. Gallen]

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturstile II

Broker

KBSE

SOA

Fachhochschule
Dortmund
University of Applied Sciences and Arts

Architekturstile II