INFORMATIONSINTEGRATION UND WEBPORTALE

Mehrschichtenarchitekturen und Enterprise Java Beans
Dr. Simone Braun

Klick-And-Bau

Informationsintegration und Webportale

ZU MEINER PERSON

Dr. Simone Braun (CAS Software AG, früher FZI)

- **Ausbildung**
  - Promotion über kollaborative Ontologieerstellung in der Anwendung von Social Semantic Tagging, Prof. Studer, AIFB
- **Aktuelle Tätigkeit**
  - Manager Innovation & Business Design, CAS Software AG
- **Interessen**
  - CRM, xRM, Wissensmanagement, Semantic Web, E-Learning, Cloud Computing

- **Aktuelle Projekte**
  - Broker@Cloud – Continuous Quality Assurance and Optimisation for Cloud Brokers
  - MAC4U - Mass Customization für individualisierte Produkterweiterungen
  - OSMOSE – OSMOsis applications for the Sensing Enterprise
- **Lehre**
  - Seit 2008 TGL-Seminar im SoSe: http://tgl.fzi.de
  - Seit 2011 IIWP
    - *Mehrschichtenarchitekturen* für skalierbare, zuverlässige, wartbare Anwendungen
    - Konzepte zur *Informationsintegration*, d.h. Einbindung externer Inhalte & Quellen

CAS Software AG

Deutscher CRM-Marktführer für KMU

43 Mio.

Umsatz 2013
der CAS-Gruppe*:
> 43 Mio. Euro

450+

Mitarbeiter:
ca. 450 CAS-Gruppe*

200

Vertriebs- und
Lösungspartner: 200

45%

Eigenkapitalquote:
> 45 Prozent

20%

Investitionen in
Innovationen: 20 - 30
Prozent vom Umsatz

250 Tsd.

Nutzer von
CAS-Produkten:
250.000 Menschen

30

International:
aktiv in über
30 Ländern

10

Sprachversionen:
CAS genesisWorld
in 10 Sprachen

*CAS Software AG und Beteiligungen anteilig

CAS SmartCcompanies & SmartEnterprise

„To each customer their own CAS“

Car Dealerships	Educational Institutions	SMEs	Aviation Industry	Multi-variant Production / Services	Business Ecosystems	Associations Networks

Successful Relationships

CAS Drive
Efficient & professional customer management tailored to the car trade

CAS Education
Flexible solutions for hihger education and schools to suit individually defined processes

CAS Mittelstand
Multi-awarded market-leading xRM solution for SMEs

CAS Aviation
Individual solutions for product configuration management and sales processes for aviation industries

CAS Merlin
Software solution and expert knowledge for configuration and sales support of complex products

CAS Ecosystems
Smart solutions for strategic cooperation and efficient cross-company processes

CAS Communities
Innovative software solution for networks with integrated portal for communication purposes

CAS SmartEnterprise Plattform

IT Platform	Devolpment / Production	Services
xRM Plattform CAS SmartDesign	Development, PM, Test, QM, Support, Innovation Management, Design Thinking	Marketing, Communication, Finance, HR, IT, Acedemy

MOTIVATION MEHRSCHICHTEN-ARCHITEKTUREN

Herausforderungen

- Wie baut man moderne Shop-Anwendungen?
- Wie baut man sie
  - skalierbar
  - wartbar
  - zuverlässig?

Beispielszenario Baumärkteportal

- Für einen Kunden soll ein **Konsumenten-Portal Klick-Und-Bau.com** erstellt werden

Warenkatalog

Angebote verschiedener Baumärkte

Onlinebestellung

Redaktioneller Teil

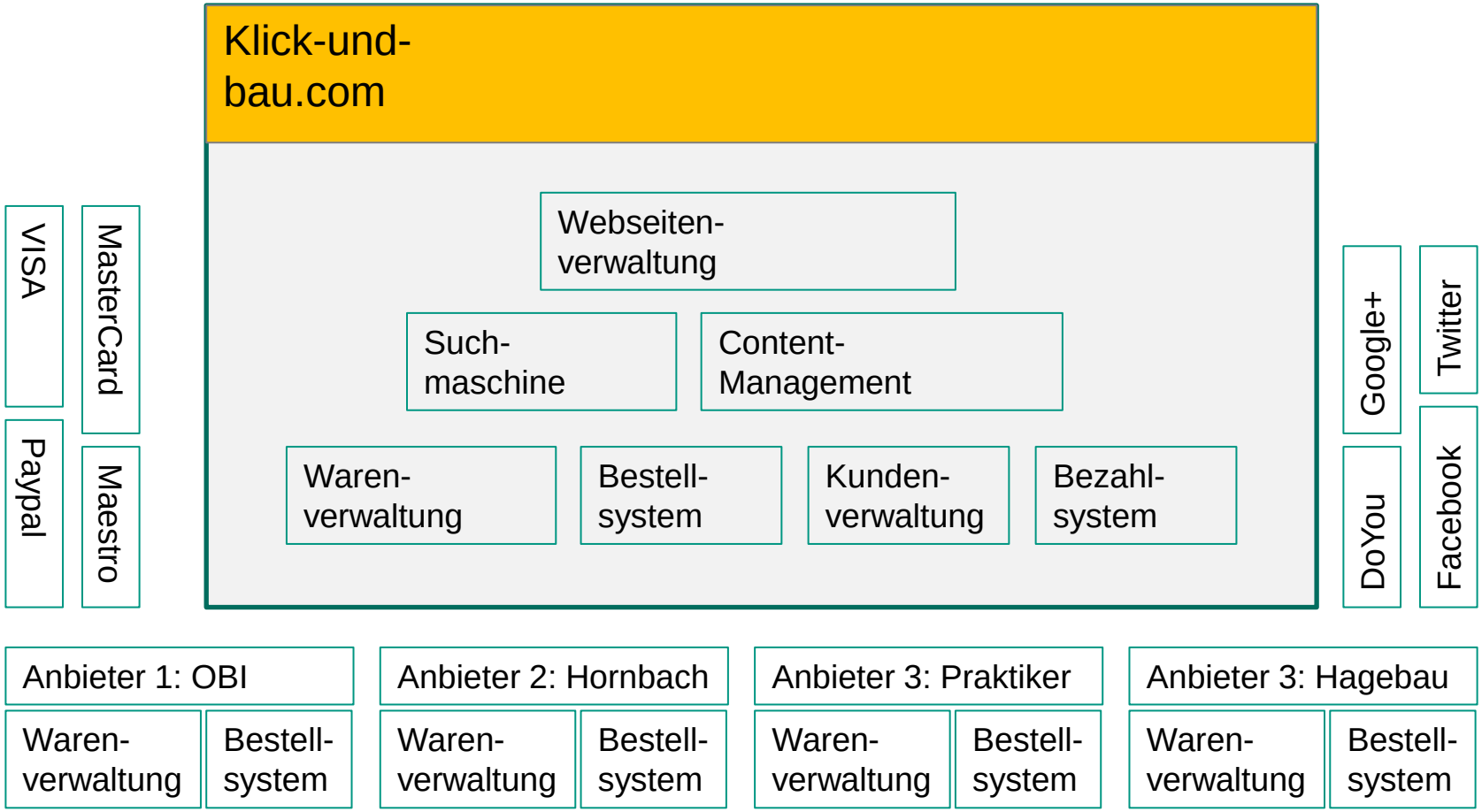
Anbindung an Bezahlsysteme

Community: Produktbewertungen

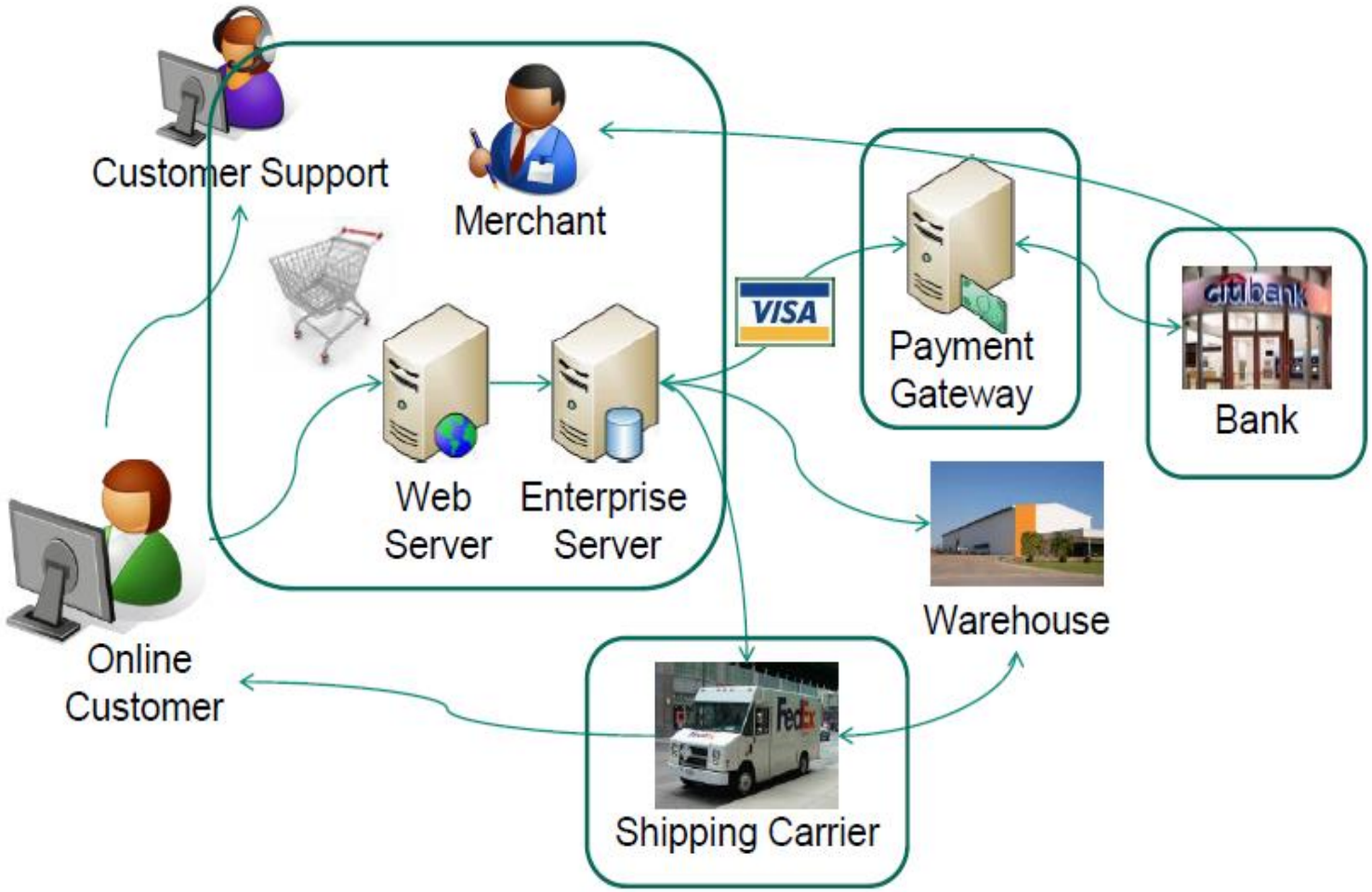
Personalisierte Angebote

Baumärkteportal: Inhalte und Quellen

Eine Analyse der Inhalte und Quellen führte zur folgenden Übersicht

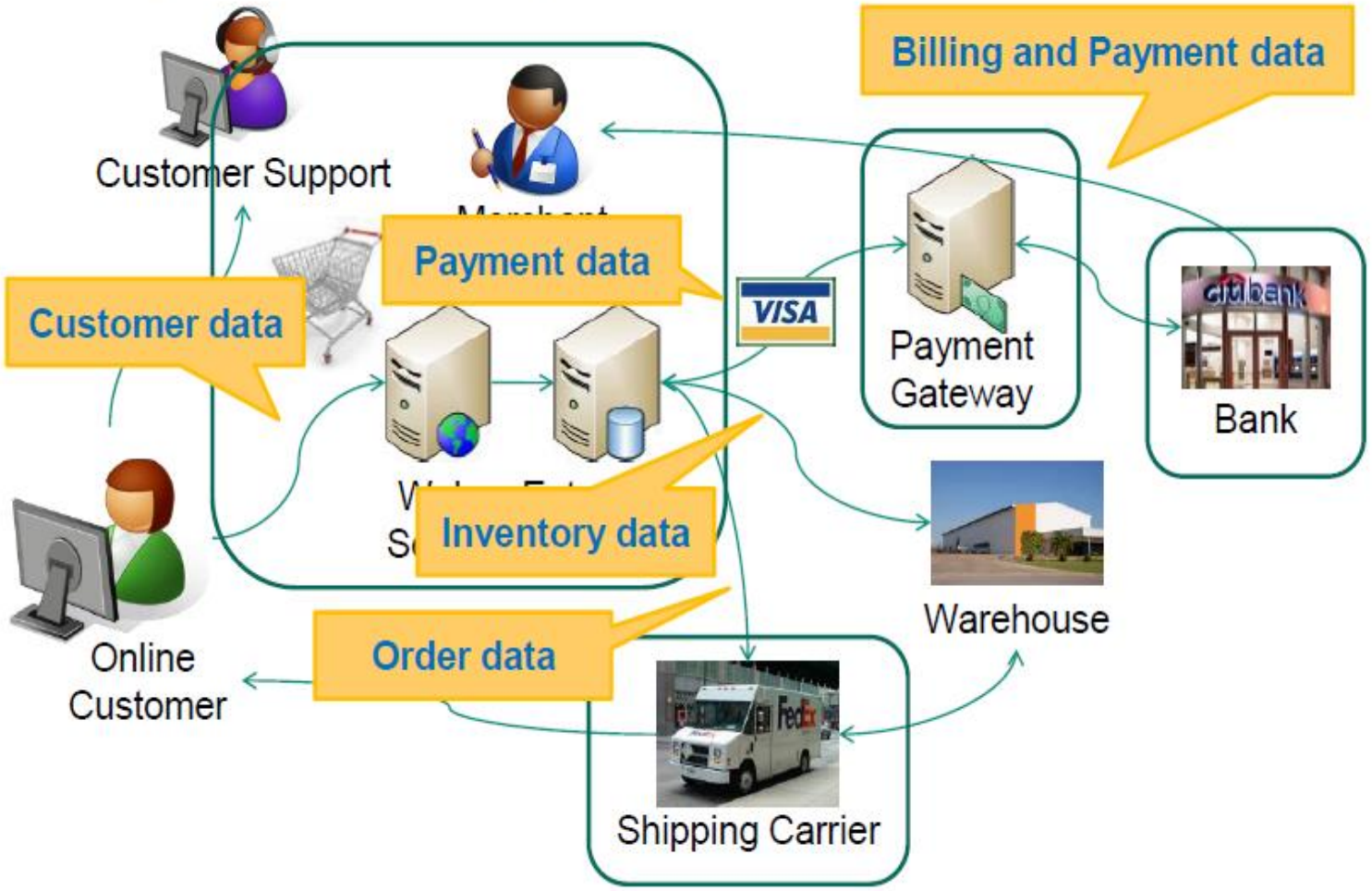


Beispiel für eine verteilte Shop-Anwendung



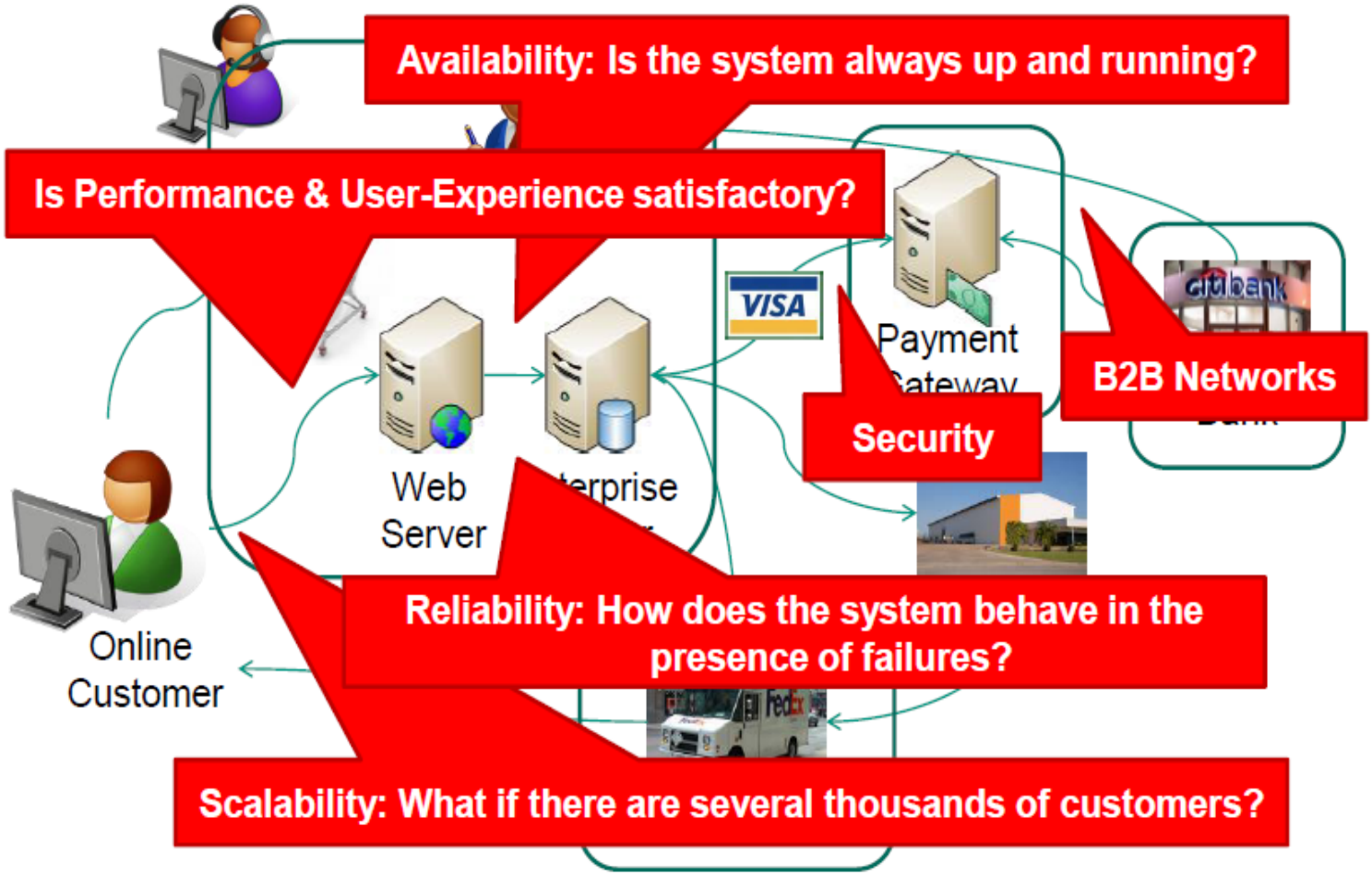
Quelle: Prof. Thai

Viele Daten & viel Datenaustausch



Quelle: Prof. Thai

Viele Anforderungen



Quelle: Prof. Thai

Herausforderungen (2)

- **Skalierbarkeit**
  - Wie können wir wachsende Besucherzahlen bewältigen?
  - Wie können wir ein wachsendes Produkt- und Dienstleistungsangebot bewältigen?
- **Zuverlässigkeit**
  - Wie können wir unser System robust gegen Ausfälle gestalten?
- **Wartbarkeit und Agilität**
  - Wie können wir neue Versionen ohne Ausfallzeiten online bringen?
  - Wie können wir das System so gestalten, dass wir möglichst schnell Änderungen ohne Seiteneffekte umsetzen können?

Technische Antworten

- **Komponentenorientierung**
  - Ermöglichen modularen Aufbau
  - Komponenten sind austauschbar, wiederverwendbar und möglichst isoliert wartbar
- **Verteilung**
  - Lastausgleich
  - Ausfallsicherheit durch Redundanz
- **Hot Deployment**
  - Aktualisieren von Enterprise Anwendungen im laufenden Betrieb
  - Kein Stoppen oder Neustarten des Servers
- **Transaktionen**
  - Folge von Verarbeitungsschritten, die nur gemeinsam oder gar nicht durchgeführt werden

MEHRSCHICHTEN-ARCHITEKTUREN

Software-Architektur

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

Quelle: Bass, Clemens, Kazman. Software Architecture in Practice. Addison-Weslay 1998

„eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Beziehungen“

Quelle: Helmut Balzert: Lehrbuch der Softwaretechnik. Spektrum Akademischer Verlag, 2011

Schichtenarchitekturen

- Schichtung ist weit verbreitetes Architekturmuster
  - Definieren Prinzipien zur Strukturierung von Software-Architekturen
  - Häufig angewandt: hierarchische Strukturierung
  - Jeder Schicht wird ein bestimmter Aspekt zugeordnet, z.B. Teil-Funktionalität, **Komponente** oder auch Klasse

    → Ordnet die Abhängigkeit von Komponenten in Schichten, die sich nur begrenzt gegenseitig nutzen dürfen, z.B. „höhere“ Schicht darf nur „tiefere“ Schichten verwenden

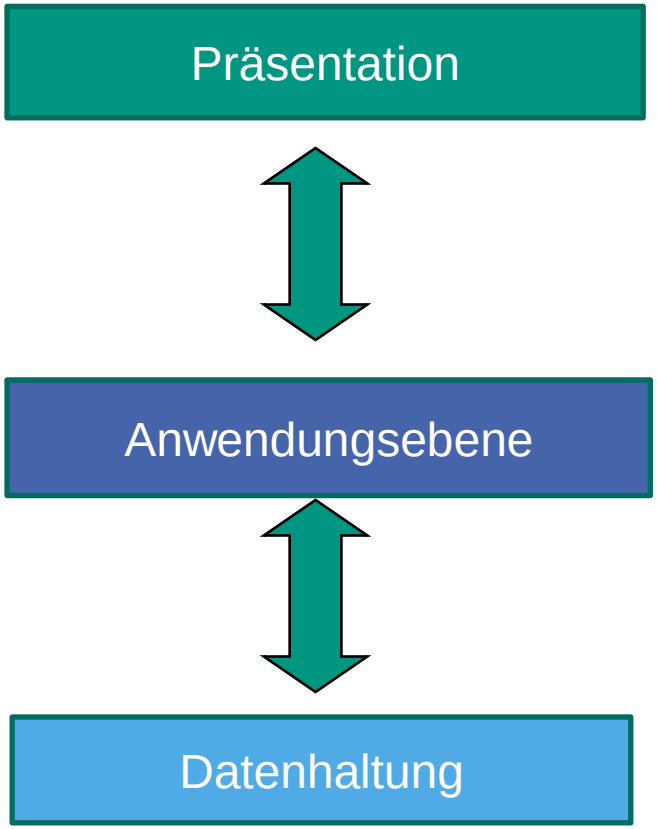
    → Beziehung der Komponenten untereinander ist zu beschreiben
  - Ziel: Bessere Strukturierung und Reduktion von Komplexität

    → durch wechselseitige Abhängigkeiten

    → vereinfacht und abstrahiert die Funktionen der individuellen Komponenten

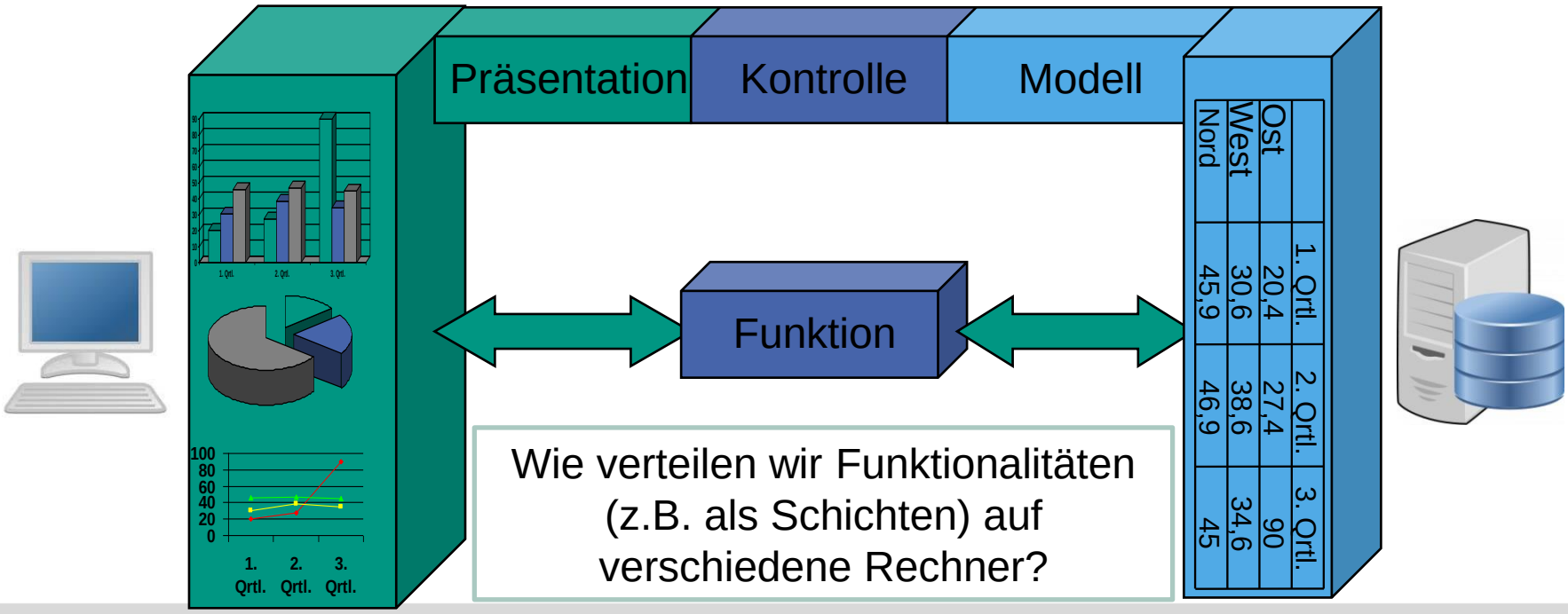
Schichtenarchitekturen

- Die Komponenten einer Softwarearchitektur können in logische Ebenen (Layer) angeordnet werden, z.B. in
  - Ebene mit Komponenten zur Erzeugung der GUI und Kommunikation mit Nutzer
  - Ebene mit Komponenten mit Anwendungslogik
  - Ebene mit Komponenten zur Daten-/Ressourcenhaltung-, -management und -zugriff



Schichtenarchitekturen

- Unterschiedliche Schichtenarchitekturen je nach dem wie und wo Präsentation, Anwendungslogik und Datenhaltung einer Anwendung implementiert sind
- Tiers = physische Verteilung von Schichten auf Rechner
- Layers = logische Verteilung

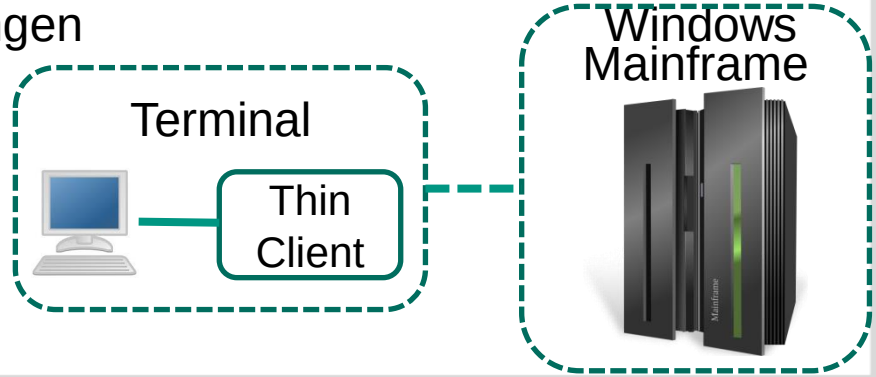


1-Schichtenverteilung

- Präsentation, Anwendungslogik und Datenhaltung in einer Schicht
  - Management der Ressourcen erfolgt zentral
  - Software selbst kann hoch-optimiert werden (Trennung zwischen Schichten hier nicht zwingend notwendig)

- Nutzer arbeiten mit einer monolithischen Anwendung
  - z.B. Textverarbeitung für einzelnen Nutzer; ggf. geteiltes Dateisystem

- Mehrere Nutzer: Mehrere Rechner mit monolithischer Anwendung über (grafische) Terminals verbunden
  - z.B. typisch bei Mainframeanwendungen oder Terminal Server (WTS)

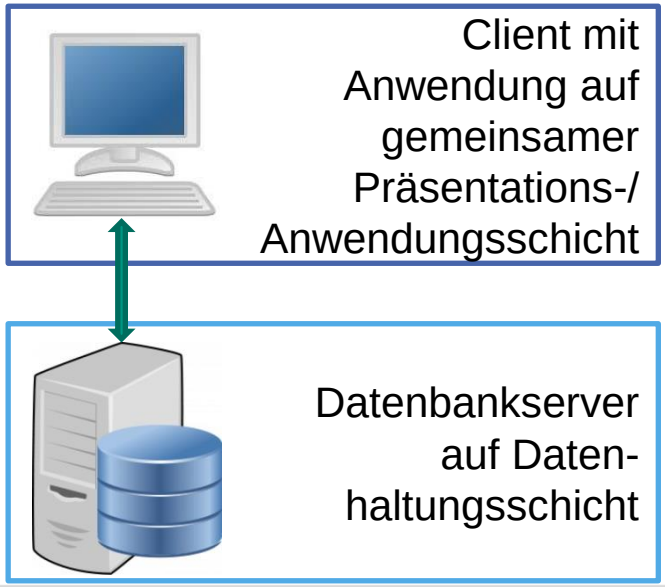


2-Schichtenverteilung

- aka. Client-Server-Systeme

a) Trennung von Präsentation und dem Rest (Anwendungslogik und Datenhaltung)
   - Client enthält Präsentation mit GUI und behandelt Interaktion mit Nutzer → Thin Client
   - Server kapselt Anwendungslogik und Datenhaltung

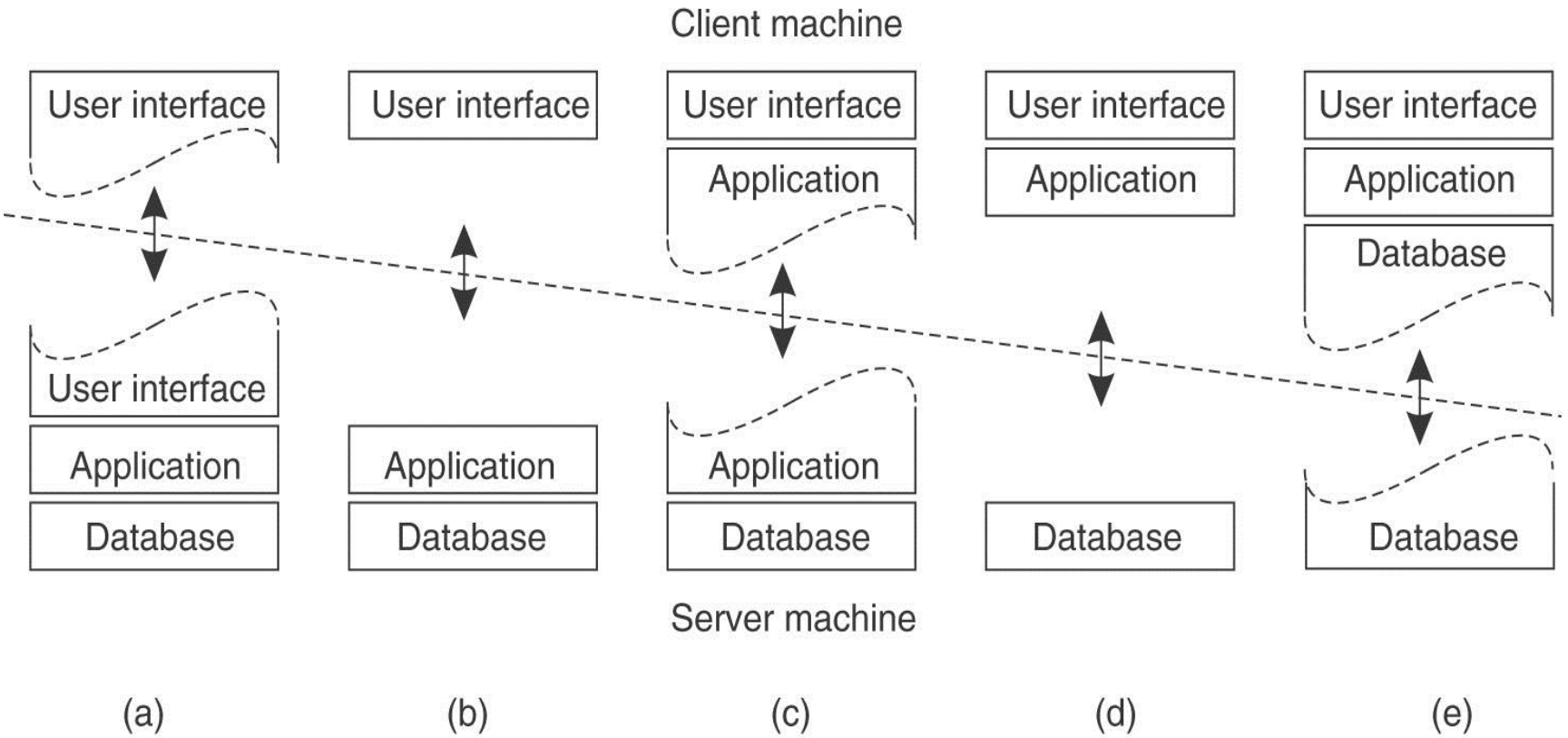
b) Trennung von Datenhaltung und dem Rest (Präsentation und Anwendungslogik)
   - Client enthält Anwendung mit GUI und Logik → Fat Client
   - Server kümmert sich um Datenmanagement und -haltung



2-Schichtenverteilung

- Clients sind (weitgehend) unabhängig voneinander
  - Es kann auch verschiedene Clients für verschiedene (Teil-) Funktionalitäten geben
- Erlaubt die Nutzung komplexerer GUI's mit intensiverer CPU-Nutzung, da Clients verteilt sind
- Universelle Kommunikationsschnittstellen zwischen Client und Server müssen bereitgestellt werden

Thin Client / Fat Client



2-Schichtenverteilung

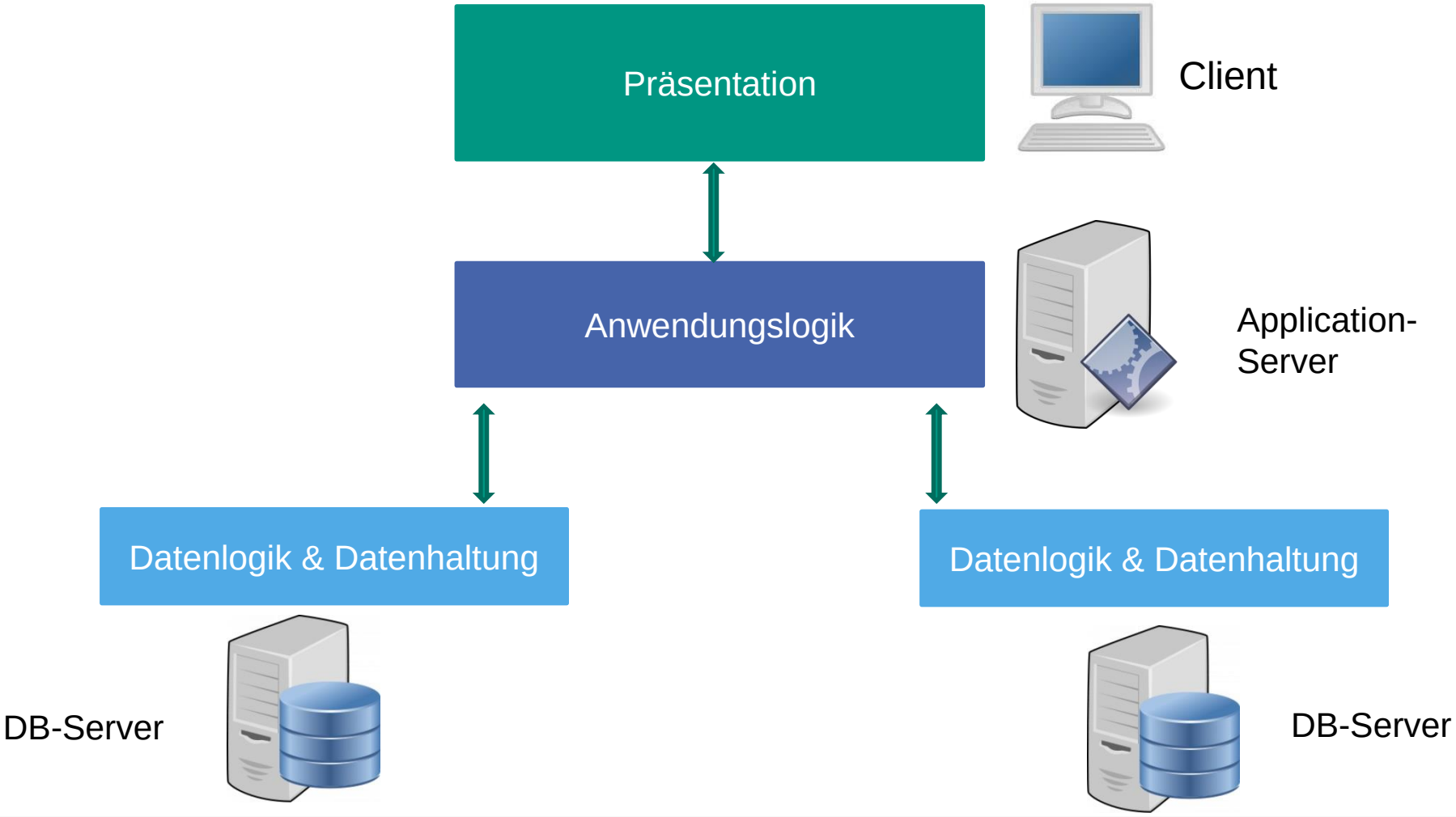
- Nachteile bei a)
  - Ein Server muss in der Regel alle Clients bedienen
- Nachteile bei b)
  - Mehrere Server, die bzgl. dem Ressourcenzugriff nicht synchronisiert sind
    - da gemeinsam genutzte Anwendungslogik fehlt (wo diese stattfinden kann)
  - → Client ist Integrator beim Zugriff auf Server mit verschiedenen Funktionalitäten
  - → Client muss wissen, wo was ist, wie er darauf zugreifen kann, wie Konsistenz garantiert werden kann, etc.
- Ineffizienz nur durch Erweiterung der Schichtverteilung zu lösen

3-Schichtenverteilung

- Trennung der Präsentation, Anwendungslogik und Datenhaltung
  - voll modularisiertes System
  - Verschiedene Schichten können über Netzwerktechnologien miteinander kommunizieren
- Die Schichten werden dabei oft auch auf verschiedene Rechnersysteme verteilt
  - Dadurch höhere Skalierbarkeit
  - Ggf. Performanzverlust durch Verteilung
- Middleware wird als universelle Kommunikations-"Brücke" zwischen den einzelnen Schichten eingesetzt
  - trennt Anwendungslogik zu einem gewissen Teil von der Kommunikationslogik
  - erhöht die Interoperabilität von Softwaresystemen über Rechnergrenzen hinweg

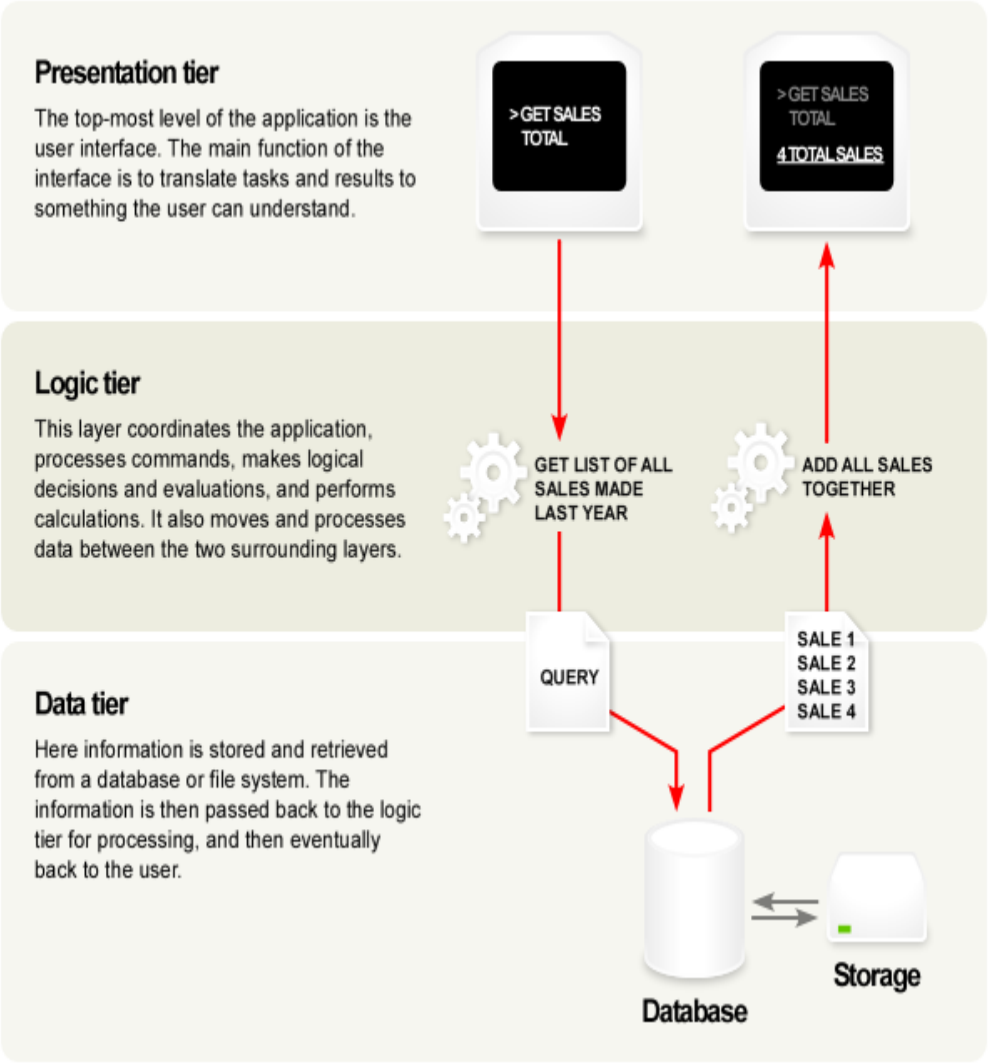
3-Schichtenverteilung

- Typisches Beispiel



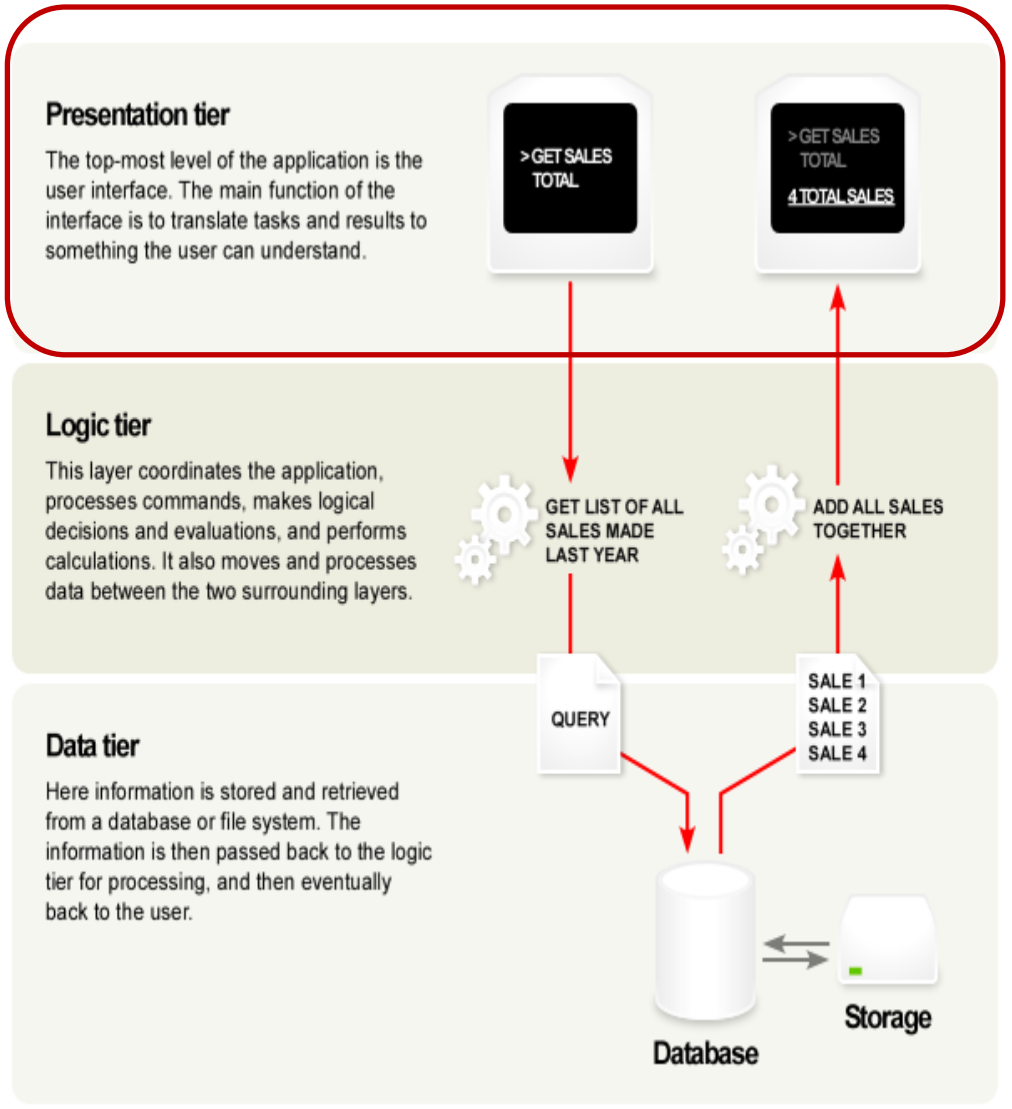
3-Schichtenverteilung

- Prinzipien:
  - Client-Server Architektur
  - Jede Schicht (Präsentation, Anwendungslogik, Datenhaltung) sollte unabhängig sein und keine Abhängigkeiten zur Implementierung aufzeigen
  - Nicht direkt verbundene Schichten sollten nicht miteinander kommunizieren
  - Änderungen einer Plattform betreffen nur die darauf laufende Schicht



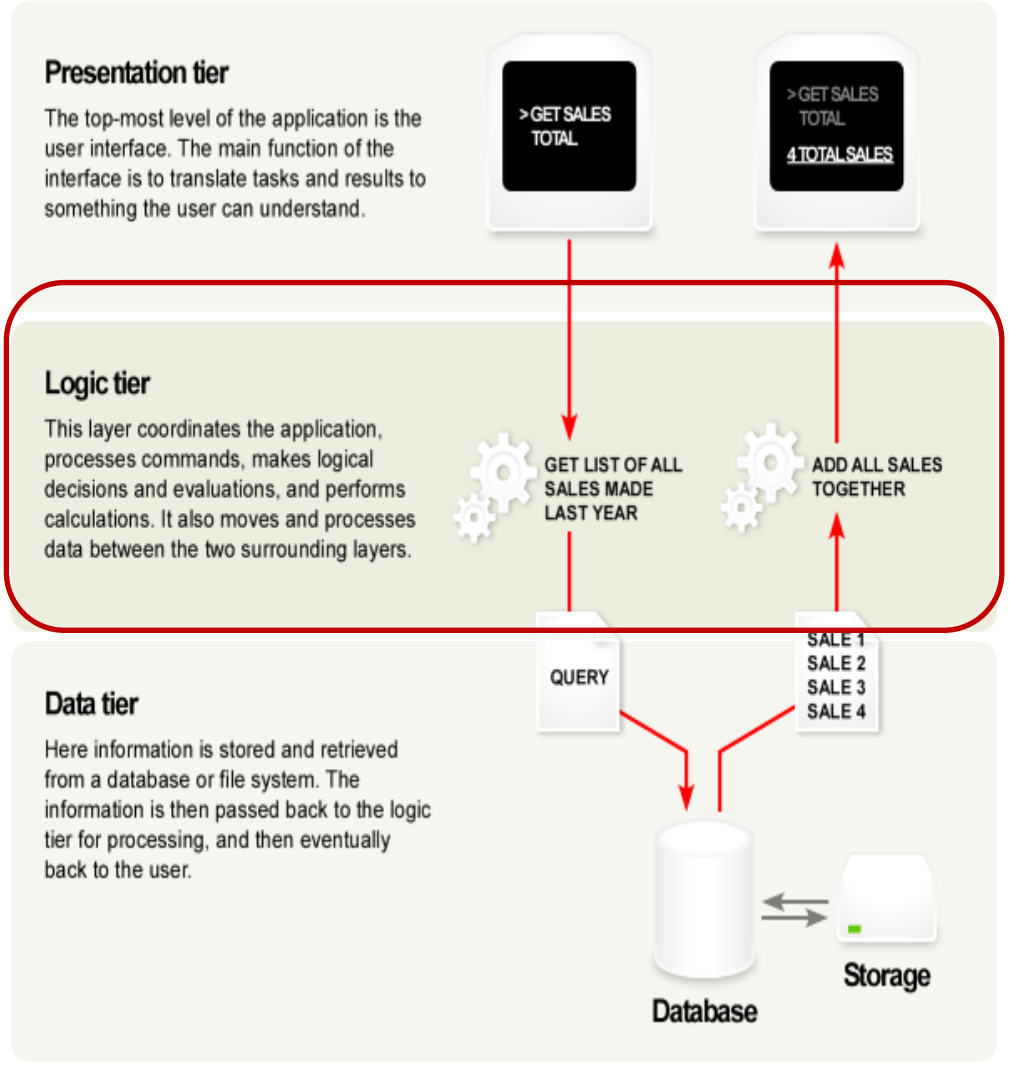
3-Schichtenverteilung

- **Präsentation**
- Stellt UI zur Verfügung
- Behandelt Nutzerinteraktionen
- auch GUI oder Client View oder Front-end
- Sollte keine Anwendungslogik oder Datenzugriffslogik enthalten



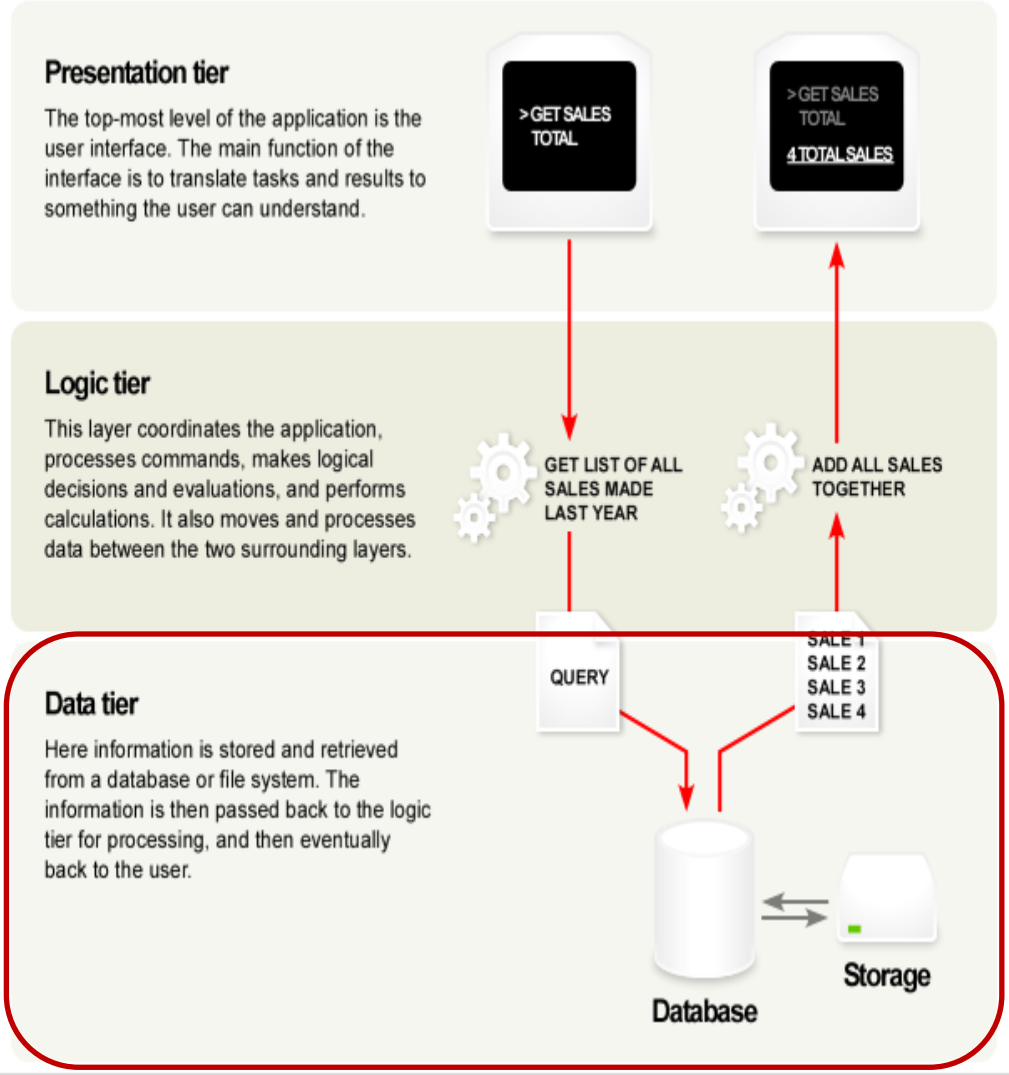
3-Schichtenverteilung

- **Anwendungslogik**
- Regelset zur Informationsverarbeitung
- Kann mehrere Nutzer verwalten
- Manchmal auch als Middleware oder Back-end bezeichnet
- Sollte keine Präsentations- oder Datenzugriffslogik enthalten



3-Schichtenverteilung

- **Datenhaltung**
- physische Speicherspeicherung zur Datenpersistenz
- Managt Zugriff auf die DB
- Manchmal auch Back-end genannt
- Sollte keine Präsentations- oder Anwendungslogik enthalten



N-Schichtenverteilung

- Durch Hinzufügen weiterer Schichten
  - z.B. bei Web-Anwendungen durch zusätzlichen Web-Tier
  - → Einsatz von komponenten-orientierten Frameworks, die auf Application Server basieren
  - Komplette Frameworks für Mehrschicht-Anwendungen:
    - .NET Framework
    - Java J2EE

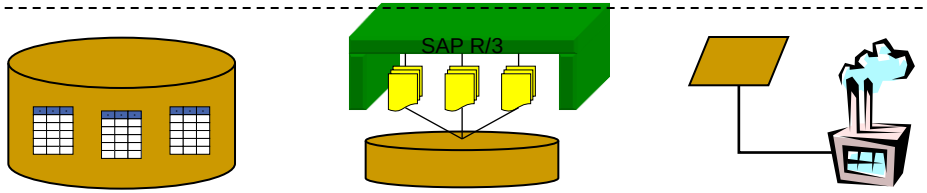
- Durch Verknüpfungen mehrerer Mehrschicht-Anwendungen über zusätzliche Integrationsschicht

Ausgangslage Shop-Anwendung

Client mit unterschiedlichen Fähigkeiten

„Enterprise Application“

Rohdaten und -dienste (nicht integriert)



Mehrschichtenarchitektur

Client mit unterschiedlichen Fähigkeiten



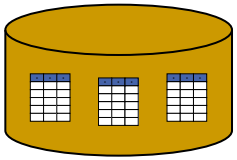
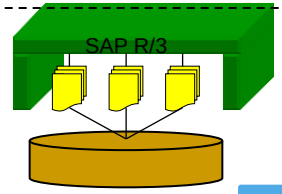
Web-Server mit Personalisierungsschicht

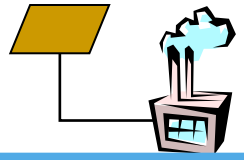
Präsentation

Aufeinander aufbauende Applikationen

Anwendungslogik

Rohdaten und -dienste (nicht integriert)



Datenlogik & Datenhaltung

Mehrschichtenarchitekturen

Client mit unterschiedlichen Fähigkeiten

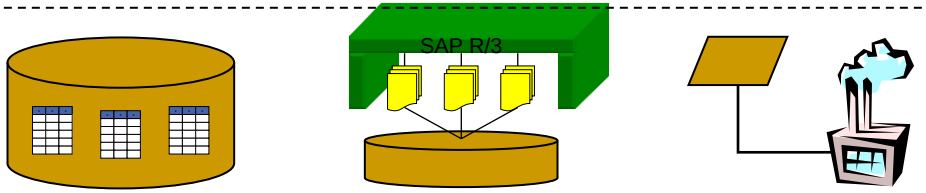
Interaktion

Geschäftsprozesse (auf Basis der Geschäftsobjekte)

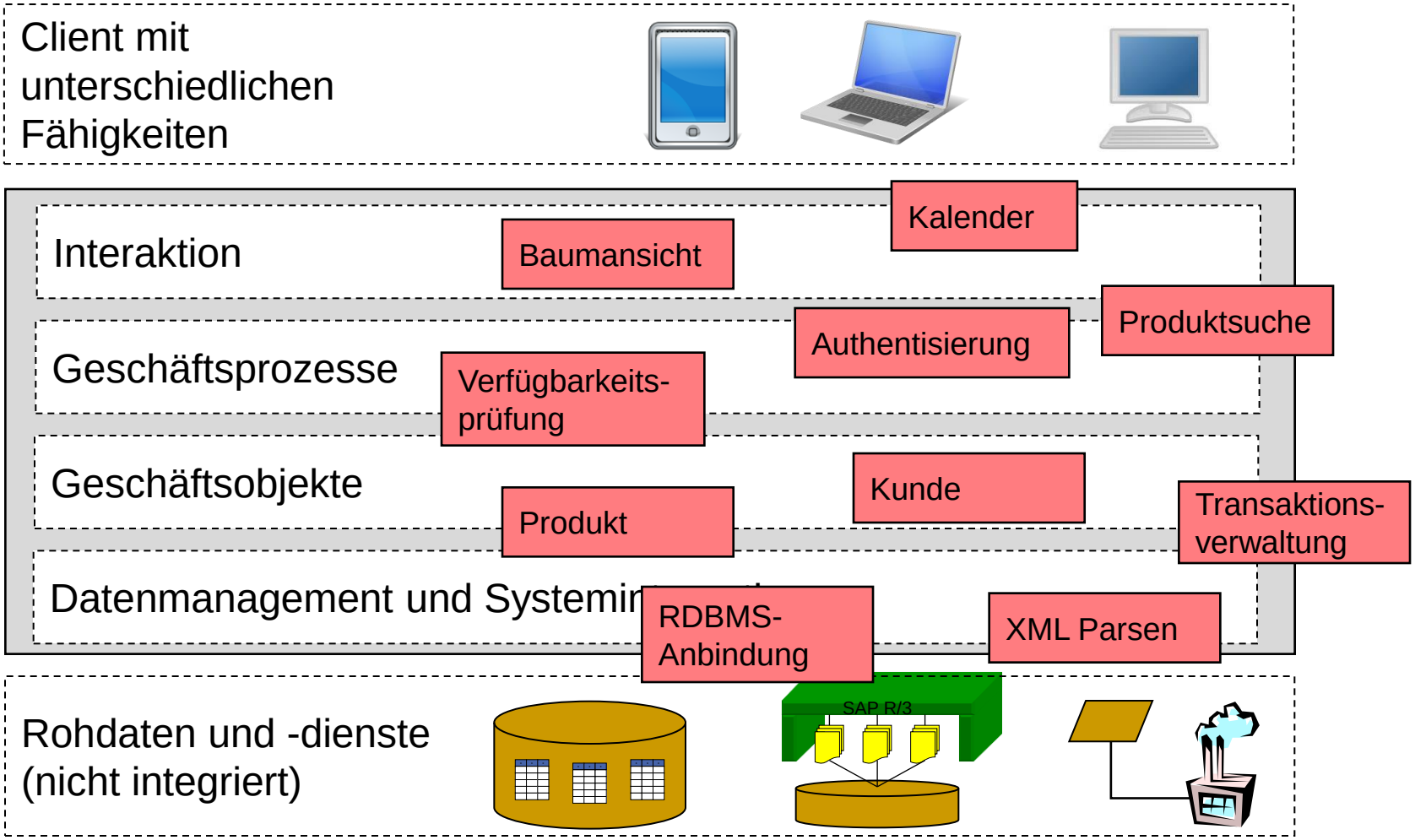
Geschäftsobjekte (technisch und inhaltlich integrierte Daten)

Datenmanagement und Systemintegration

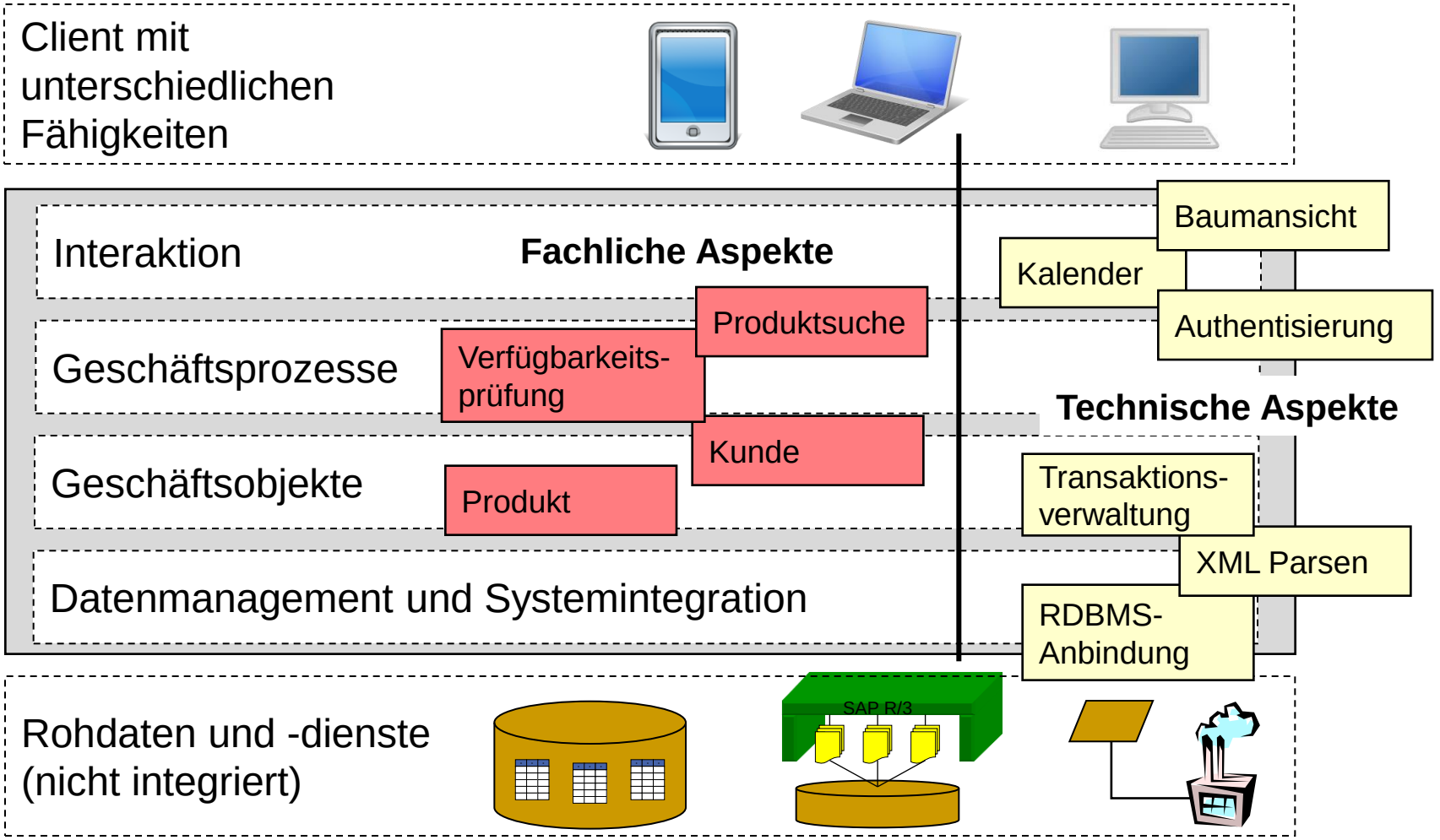
Rohdaten und -dienste (nicht integriert)



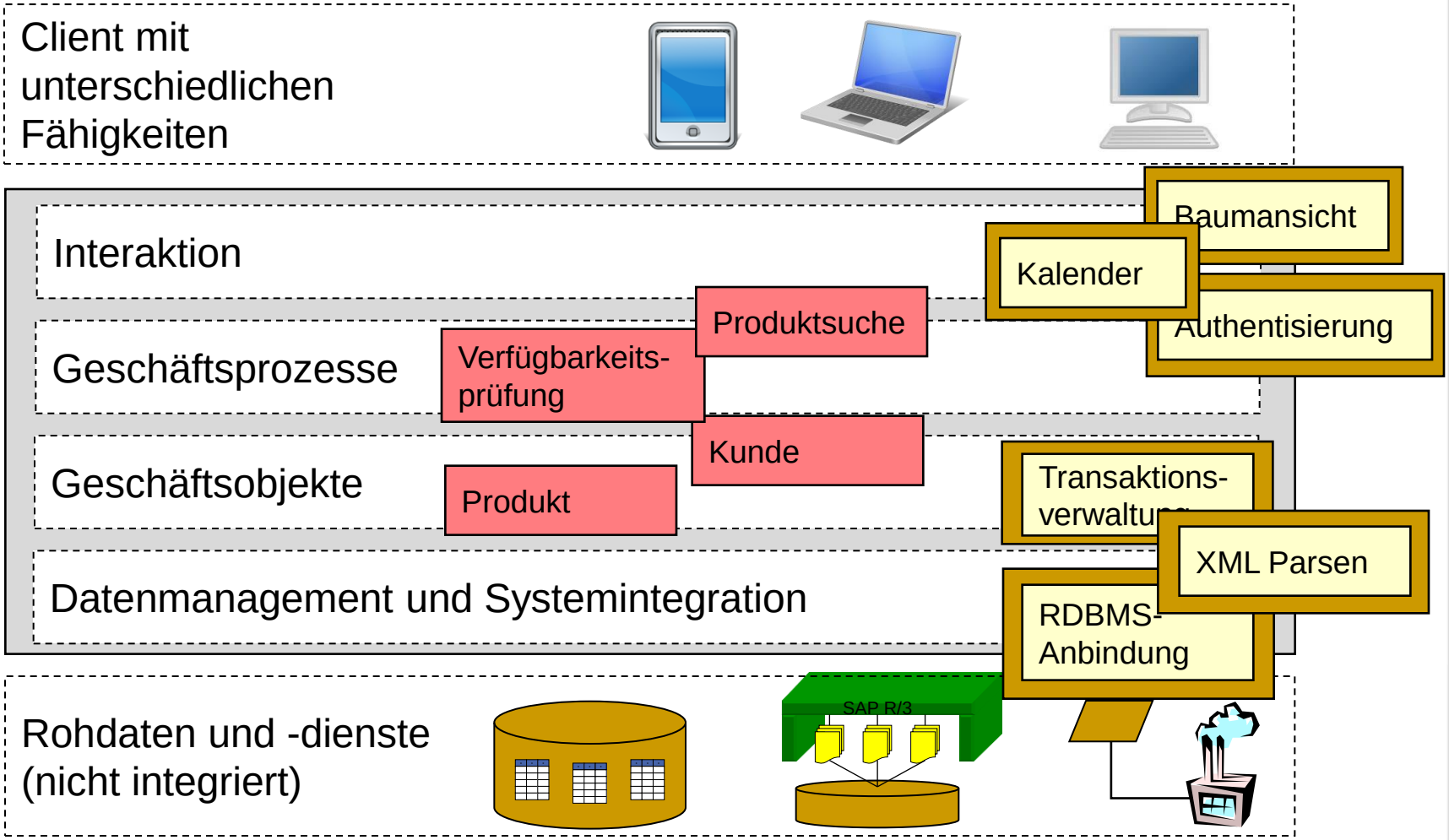
Modularisierung



„Separation of Concerns“ – Trennung von Aufgabenbereichen/Belangen



Standardisierung



Die Vorteile auf einen Blick

- Mehrschichtenarchitektur
  - Austauschbarkeit einzelner Schichten
  - 1:N-Lösungen (z.B. ein Prozess – mehrere Oberflächen)
- Modularisierung
  - Detaillierte Strukturierung des Entwicklungsprozesses
  - Eine gewisse Wiederverwendbarkeit von Teillösungen
- „Separation of Concerns“
  - Hohe Wiederverwendbarkeit und Austauschbarkeit technischer Module (hier noch unternehmensintern!)

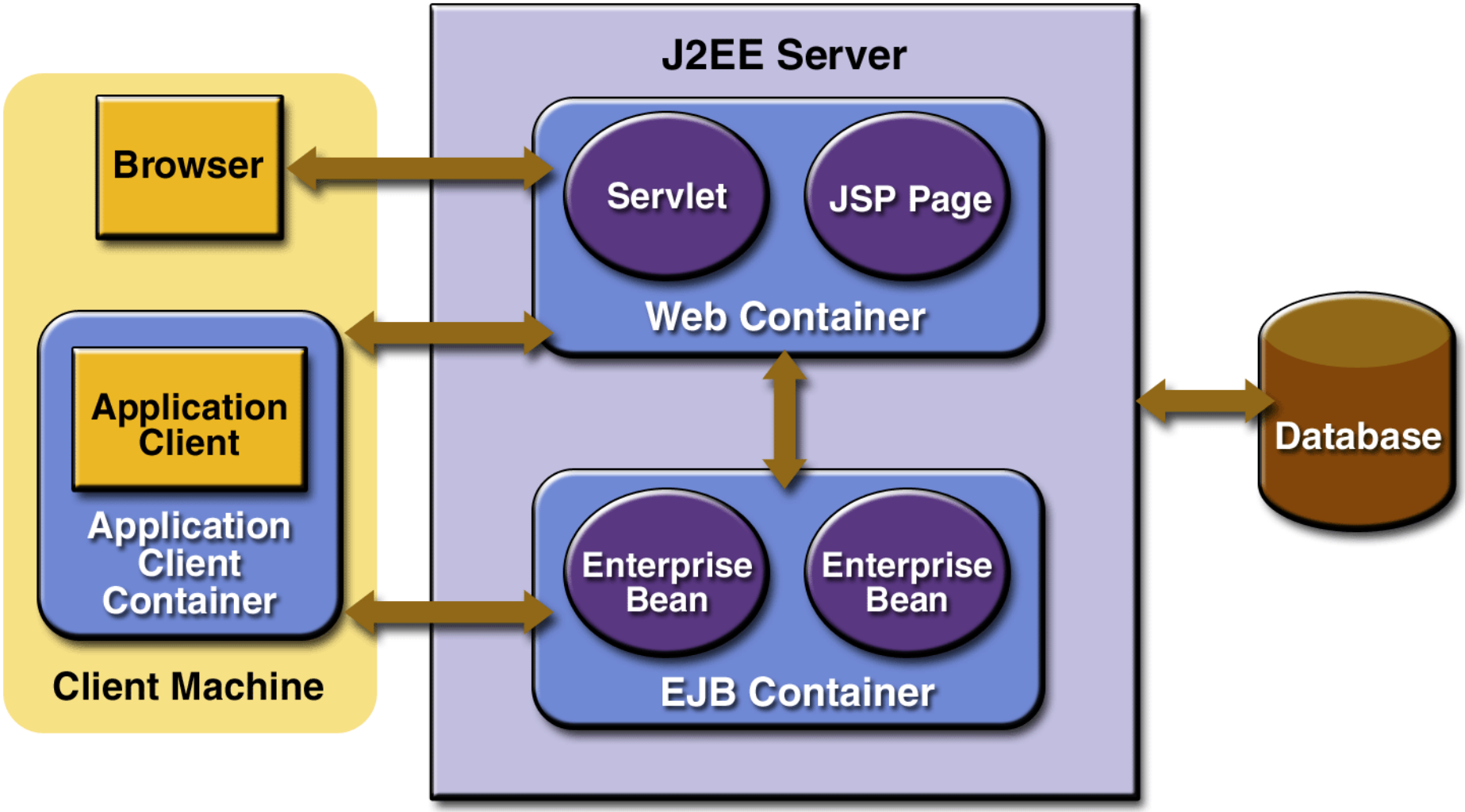
Framework für Mehrschicht-Anwendungen

JAVA EE

Java EE

- Framework für Mehrschicht-Anwendungen
- Basierend auf der Java-Technologie
- Komponenten laufen typischerweise auf unterschiedlichen Maschinen
  - Client-Schicht-Komponenten auf dem Client-Rechner
  - Web-Schicht-Komponenten auf dem Java EE Server
  - Geschäftsschicht-Komponenten auf dem Java EE Server
  - Enterprise Information System (EIS) Schicht Software (z.B. Legacy Anwendung, ERP System etc.) auf EIS Server

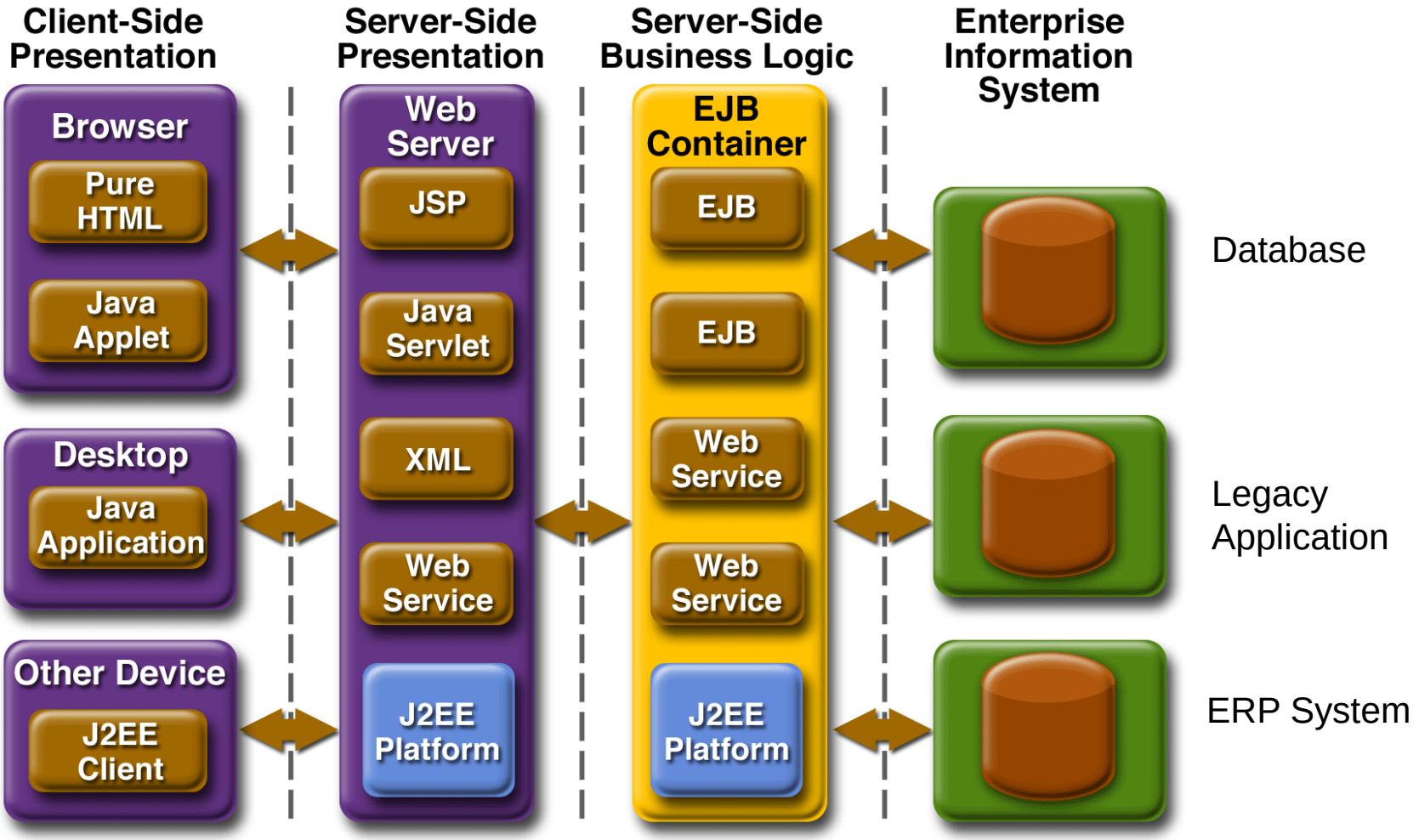
Anatomie einer Java EE-Anwendung



Container-Konzept

- Laufzeitumgebung
- Kommunikation mit den Anwendungskomponenten über standardisierte APIs
- Bereitstellung von generischen Diensten
  - Verzeichnis-/Namensdienste
  - Datenbankverbindungen incl. Pooling
  - Transaktionen
  - Nachrichtendienste
  - Authentisierung, Autorisierung
  - …

Anatomie einer Java EE-Anwendung (2)



EJB 3.x

- Die ersten Versionen von EJB (1998-2003) galten Entwicklern als zu umständlich
  - z.B. Deployment-Deskriptoren
- Deshalb hatten sich immer stärker leichtgewichtige Alternativen durchgesetzt, die mit POJOs und Frameworks wie Spring arbeiteten
- Mit EJB 3.0 (2006) wurde eine annotationsbasierte Entwicklung ermöglicht, die den Overhead deutlich reduziert
- Aktuelle Version EJB 3.1 (2009)

Java Beans

- Sind Software-Komponenten für Java mit Felder und Methoden, die Module der Geschäftslogik implementieren
- Werden als Container zur Datenübertragung verwendet
  - D.h. Komponente implementier Standard-Interface
  - Dadurch wird das Bean an einen Bean Container gekoppelt (= Runtime Object für ein Bean)
  - → Java Klasse, die ein Standard-Bean-Interface implementiert ist ein Bean
- Haben parameterlosen Konstruktur
- Attribute (properties) zur Speicherung persistenter Daten, bestehend aus
  - Private Instanzvariablen & Getter- & Setter-Methoden für öffentlichen Zugriff
- Sind serialisierbar
- Java Bean kann aus anderen JavaBeans aufgebaut sein

3 Arten von Beans

- **Session Beans**
  - Bilden die wesentliche Applikationslogik ab
  - Stateless
    - Sind unabhängig vom konkreten Aufrufkontext
    - Können gepoolt werden
  - Stateful
    - Hängen z.B. von Benutzer-Session ab
- **Message-Driven Beans**
  - Asynchron aufrufbare Komponenten (= stateless session bean + message listener)
- **Persistenz-Beans** (früher Entity Beans)
  - Objektorientierte Repräsentationen der zu speichernden Daten
  - Nutzen JPA (vgl. Vorlesung zu Objektrelationalem Mapping)
  - „Container-Managed Persistence“: Persistenz wird durch Bean-Container gemanaged

Annotationen

- Annotationen markieren Klassen als Session Beans und definieren deren Eigenschaften
  - @Stateless macht aus beliebiger Klasse ein Stateless Session Bean
  - @Remote macht aus einem implementierten Business-Interface ein Remote-Interface; @Local ein Local-Interface
  - @TransactionAttribute definiert die Transaktionalität der gesamten Klasse oder einzelner Methoden
  - @RolesAllowed, @PermitAll, und @DenyAll bestimmen die erforderlichen Zugriffsrechte für den Aufruf
  - Über @PostConstruct und @PreDestroy lässt sich der Lebenszyklus der Bean verfolgen
  - Dependency Injection über @Resource, @EJB, @PersistenceUnit,
  - @PersistenceContext ermöglicht die deklarative Verknüpfung von Session Beans und den von ihnen benötigten Ressourcen

Ein sehr einfaches Beispiel

- 1. Schritt: Definition des Bean-Interfaces

```
package foo;
import javax.ejb.*;

@Remote
public interface FooRemote {
    public String echo(String s);
}
```

Ein sehr einfaches Beispiel (2)

- 2. Schritt: Bean-Implementierung

```
package foo;
import javax.ejb.*;

@Stateless
public class FooBean implements FooRemote {
    public String echo(String s) {
      return s;
    }
}
```

Ein sehr einfaches Beispiel (3)

```
Context ic = new InitialContext();
Object obj = ic.lookup(FooRemote.class.getName());

FooRemote foo = (FooRemote) obj;
String s = foo.echo(„test“);
```

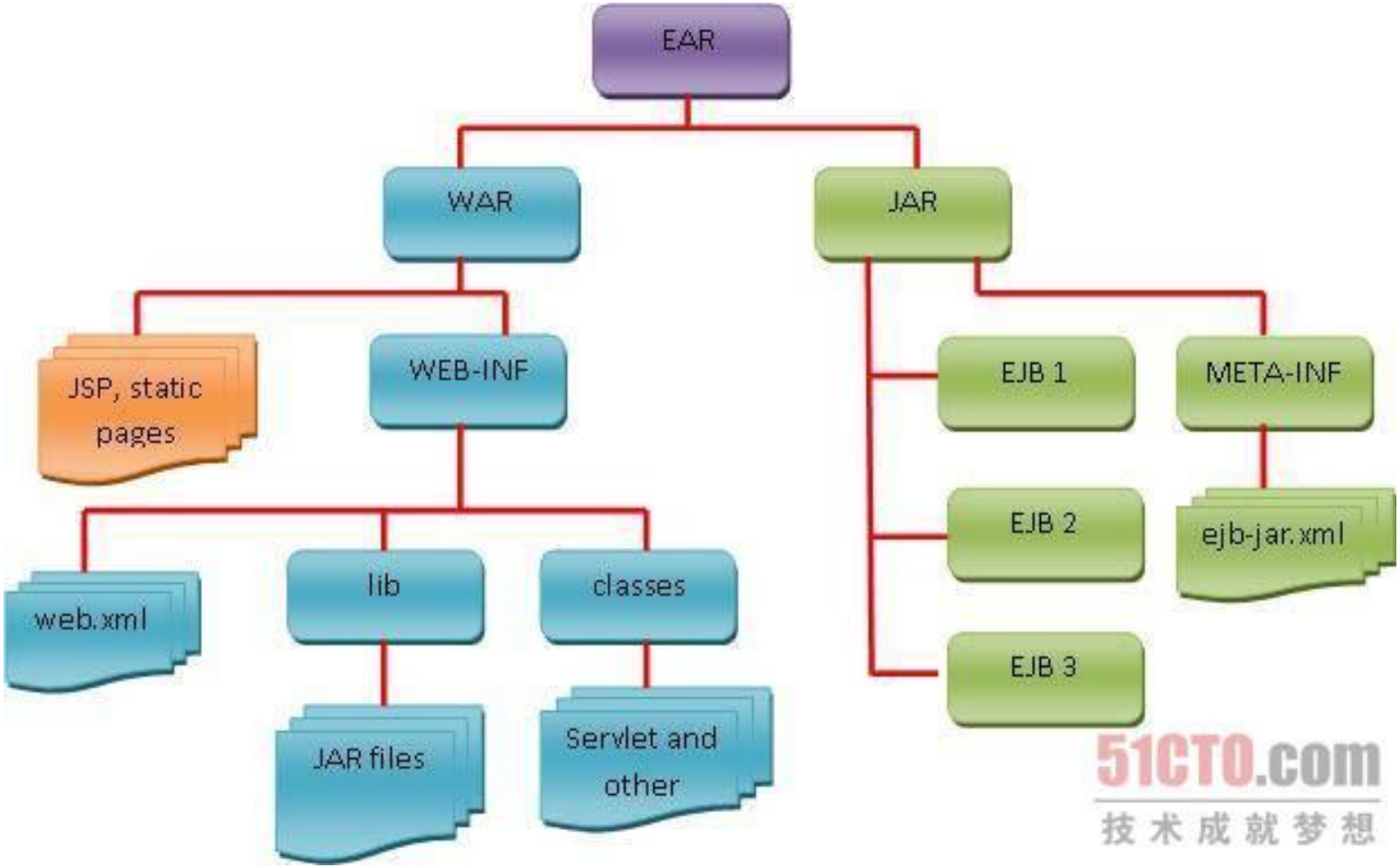
Mit Dependency Injection geht es noch einfacher:

```
Class TestDI
{
   @EJB(name=„ejb/FooBean“) bean;
   public void test() { bean.echo(„test“) }
}
```

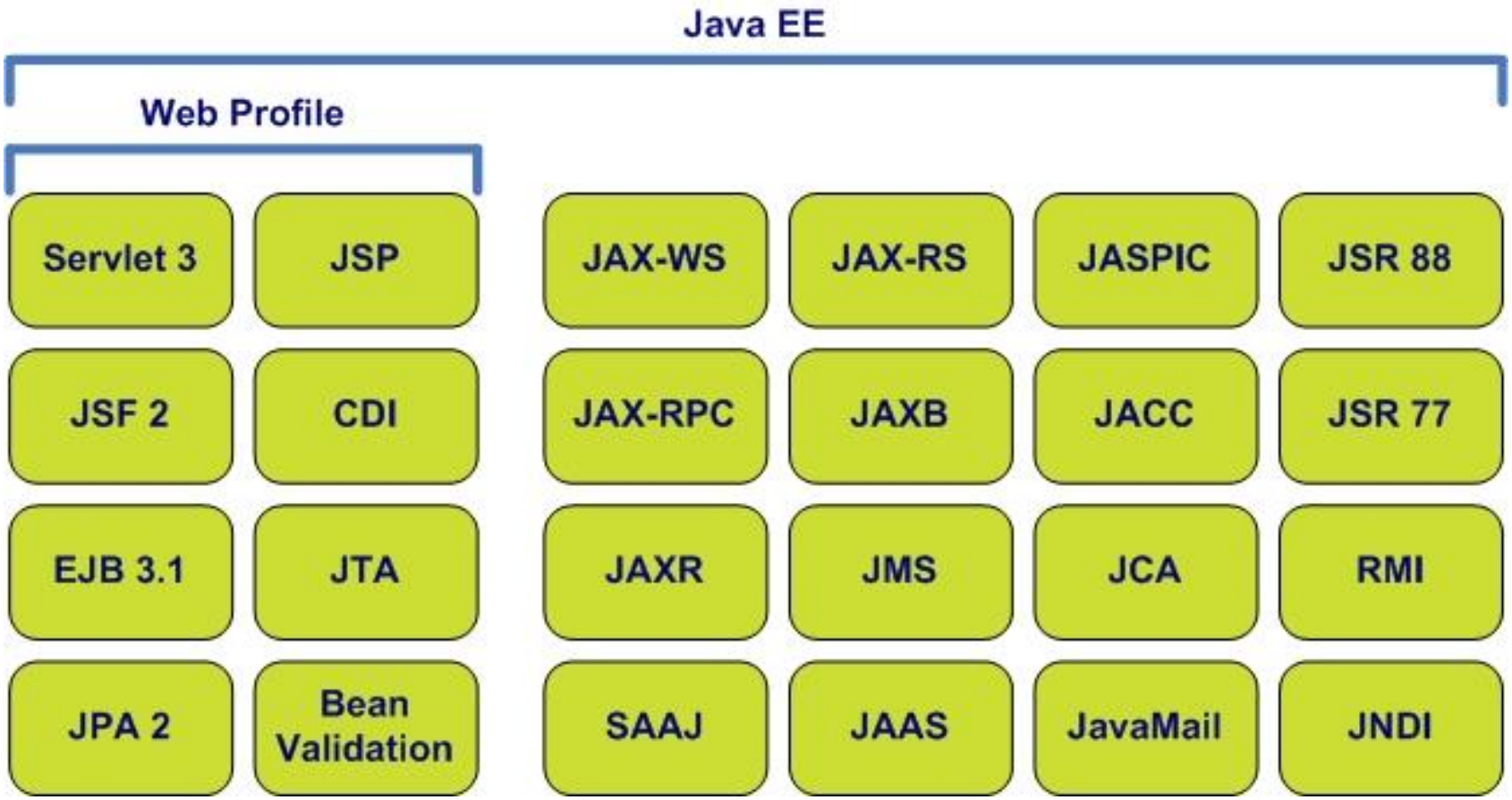
Elemente einer Java EE-Anwendungen

- **Interaktion**
  - Java Server Pages (JSP), Servlets als Controller
  - Java Server Faces (JSF) – Komponentenframework
  - + CSS, Bilder etc.
- **Geschäftsprozesse**
  - Umsetzung als SessionBeans
- **Geschäftsobjekte**
  - Objektmodell mit Abbildung über JPA
- **Datenbanksystem**
  - Relationale oder objektorientierte Datenbank

EJB Packaging



Was gehört noch alles dazu?



Weitere Elemente

- **Präsentationsebene**: Java Server Pages (JSP) und Java Server Faces (JSF)
- **Applikationsebene**
  - Transaktionale Semantik für Komponentenaufrufe
- **Dienstebene**:
  - Anbindung als Web Services (JAX-WS)
- **Datenebene**:
  - Java Persistence API (JPA) für die Anbindung von Datenbanken (vor allem relationale Datenbanken)
  - Java Connector Architecture (JCA) für die Anbindung von Enterprise-Systemen
- Infrastruktur
  - JNDI als Namensdienst
  - JavaMail für die Verarbeitung von Emails
  - Management-Funktionalitäten

Declarative Security: Zugriffskontrolle auf Anwendungsebene

- Zugriffsschutz beim Aufruf von EJBs wird über Annotations definiert
- @DeclareRoles: Definiert Rollen, gegen die beim Aufruf geprüft werden kann
- @RolesAllowed: Definiert eine Liste von Rollen, die ein angemeldeter Benutzer haben muss, um Methoden eines EJBs aufzurufen
- @DenyAll: Weist den Aufruf von Methoden eines EJBs für alle angemeldeten Benutzer ab

Declarative Security: Zugriffskontrolle auf Anwendungsebene

```
@RolesAllowed("admin")
public class SomeClass {
    public void aMethod () {...}
    public void bMethod () {...}
    ...
}

@Stateless public class MyBean implements A extends
SomeClass
{
    @RolesAllowed("HR")
    public void aMethod () {...}

    public void cMethod () {...}
    ...
}
```

JCA - Java Connector Architecture

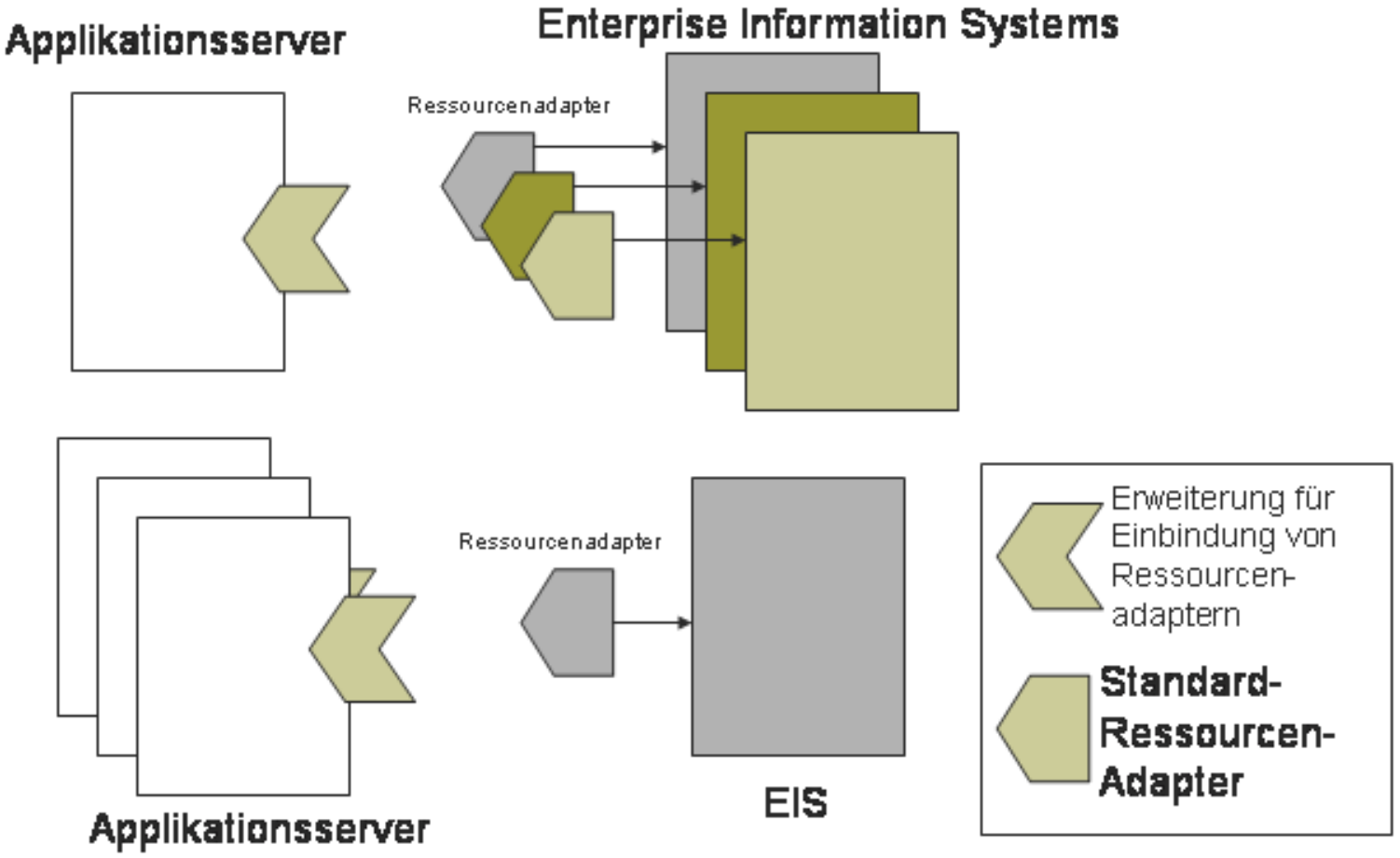
- Java Connector Architecture (JCA) ist Software-Architektur und Programmierschnittstelle (API) zur Integration von heterogenen Anwendungen und EIS
  - JCA Resource Adapter: Library, die vom Applikationsserver/Client zum Verbinden mit dem Resource Manager verwendet wird (z.B. JDBC)
  - Common Client Interface: universelle API für die Interaktion mit beliebigen Resource Adaptoren

- Reduzieren anwendungsspezifische Verbindungen, da keine Anpassung pro Applikations-Server mehr notwendig ist

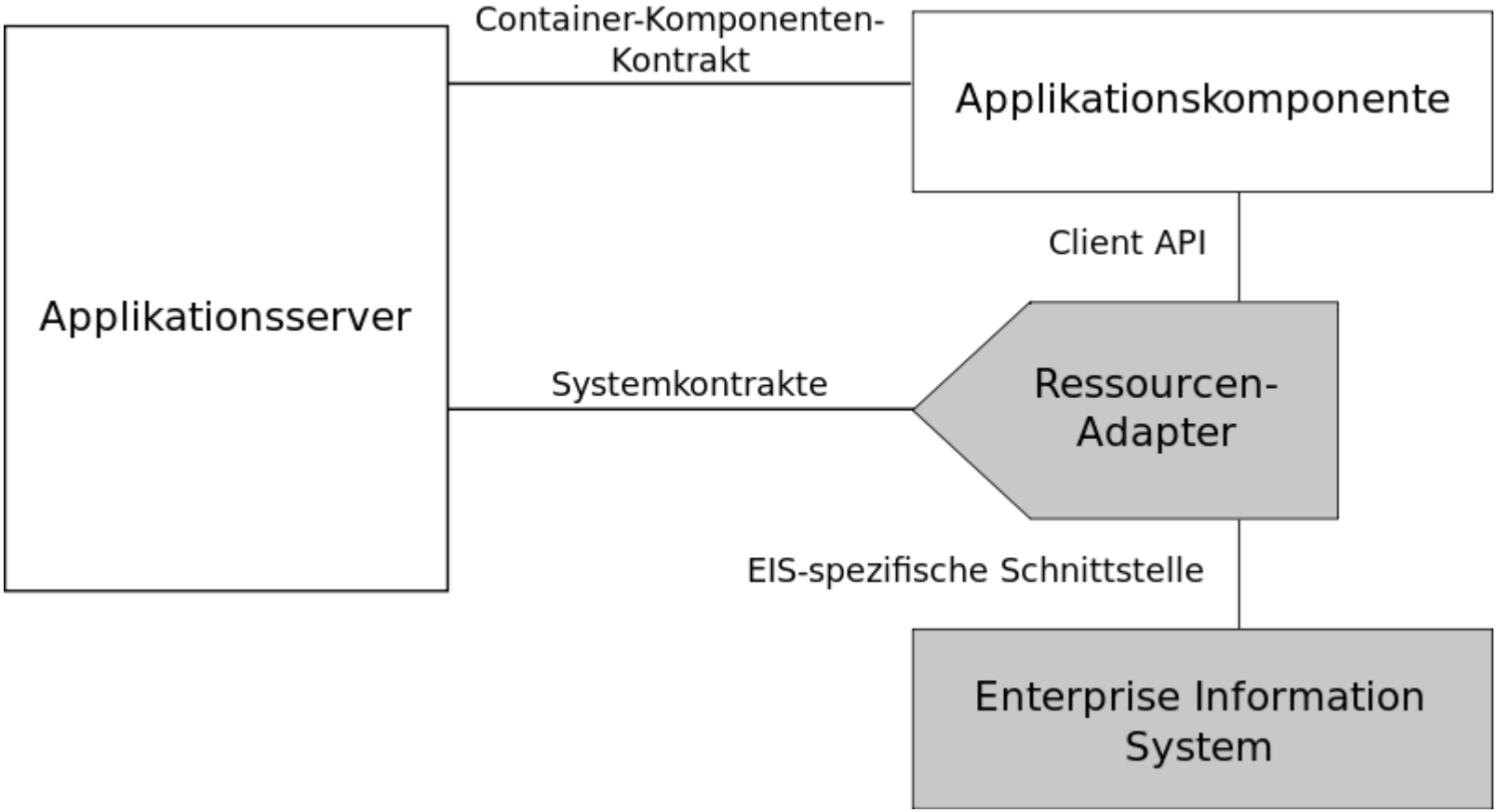
JCA - Java Connector Architecture

- Resource Adapter sind die Implementierungen einer Schnittstelle, die mehrere Server-Laufzeitkontrakte gleichzeitig erfüllt
  - Verbindungsmanagement: Verwaltung des Verbindungspools zum EIS; sprich Connection Pooling
  - Transaktionsmanagement: Einbindung in einen transaktionalen Kontext und Connection Sharing
  - Sicherheitsmanagement: sicherer Zugriff auf EIS, d.h. Übergabe von Credentials

JCA - Java Connector Architecture



JCA (2)


Container-Komponenten-Kontrakt
Applikationskomponente
Client API
Applikationsserver
Systemkontrakte
Ressourcen-Adapter
EIS-spezifische Schnittstelle
Enterprise Information System

EJB vs. Spring

- Lange Zeit umstritten, ob EJB oder POJO mit Frameworks wie Spring der bessere Weg ist

- Inzwischen haben sich die beiden Lösungen angenähert
  - EJB leichter zu benutzen: weniger Konfigurationsdateien, mehr Annotationen, Dependency Injection, Nutzung von JPA, …
  - Spring hat einen kompletten Stack an Funktionalität entwickelt, der ähnliche Aufgabenstellungen wie EJB abdeckt

- Hauptunterschied
  - EJB lokalisiert das Framework im Application Server (Container)
    - Dadurch schwergewichtigere AS, z.B. JBoss, Websphere, etc.
  - Spring ist ein Framework oberhalb des Application Servers
    - Dadurch lassen sich auch leichtgewichtige Application Server einsetzen, wie z.B. Tomcat, Jetty oder Resin

Fazit

- Mehrschichtenarchitekturen de facto Standard für die Entwicklung von web-zentrierten Anwendungen
- Java EE / Enterprise Java Beans sind ein Technologiestack für Mehrschichtenarchitekturen, der im Zentrum auf einen Application Server aufsetzt
- Ab v3.0 deutlich einfacher und wettbewerbsfähiger
- Alternative Ansätze bleiben relevant (z.B. Spring)

Weiterführende Literatur

- EJB 3.1
  - Ihns et. Al: EJB 3.1 professionell – Grundlagen und Expertenwissen zu Enterprise JavaBeans 3.1 inkl JPA 2.0, dpunkt, 2011

- Spring Framework http://www.springframework.org

SONSTIGES


CAS
CAMPUS

Abschlussarbeiten/Praktikum/WerkstudentIn

- CAS Merlin (ab sofort):
  - Industriepraktikum/Master-/Bachelorarbeit in den Bereichen Mobile, SmartDesign, HTML5/Vaadin
    - Prototypische Implementierung anhand CAS M.Model/M.Sales
    - Einsatz von Mobile- und Webtechnologien
  - Industriepraktikum/Master-/Bachelorarbeit zur Textuellen Regeldefinition im Configuration Management (Prototyping anhand CAS M.Model)
    - Prototypische Umsetzung einer 'Domain Specific Language' für das Configuration Management.
    - Basis ist CAS M.Model (Produktkonfigurator).
- CAS Aviation (ab Februar):
  - Master-/Bachelorarbeit im Bereich Supply-Chain Knowledge Management und Configuration Management
    - Wissensaustausch zwischen Unternehmen mit unterschiedlichen Wissensdomänen.
    - Konfiguration von Wissensverknüpfungen zwischen semantisch formalisiertem Wissen
    - Konfigurationsregeln auf Basis von CAS M.Model

Abschlussarbeiten/Praktikum/WerkstudentIn

- Praktikum/WerkstudentIn IBD
  - Unterstützung bei der Beantragung & Durchführung inter-/nationaler Forschungsprojekte

- Abschlussarbeiten/Praktikum im Campus-Management Umfeld
  - Konzepte, Softwareentwicklung, Usability etc.

http://www.cas-selbst-erleben.de