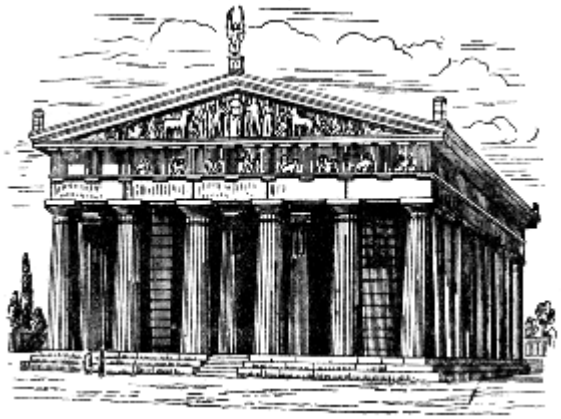
Softwarearchitektur

(Architektur: *αρχή* = Anfang, Ursprung + *tectum* = Haus, Dach)

8. Betriebliche Informationssysteme – Teil 1



Vorlesung Wintersemester 2010 / 11

Technische Universität München
Institut für Informatik
Lehrstuhl von Prof. Dr. Manfred Broy

Dr. Klaus Bergner, Prof. Dr. Manfred Broy, Dr. Marc Sihling

Inhalt

- Betriebliche Informationssysteme
  - Definitionen
  - Charakteristika
- Fachliche Architektur
- Technische Architektur
  - Basismechanismen und Infrastruktur
  - Schichtenarchitekturen und Verteilung
  - Transaktionsverarbeitung
- Datenhaltungsschicht
  - Aufgaben und Charakteristika
  - Realisierungsvarianten
- Zusammenfassung
- Literaturhinweise

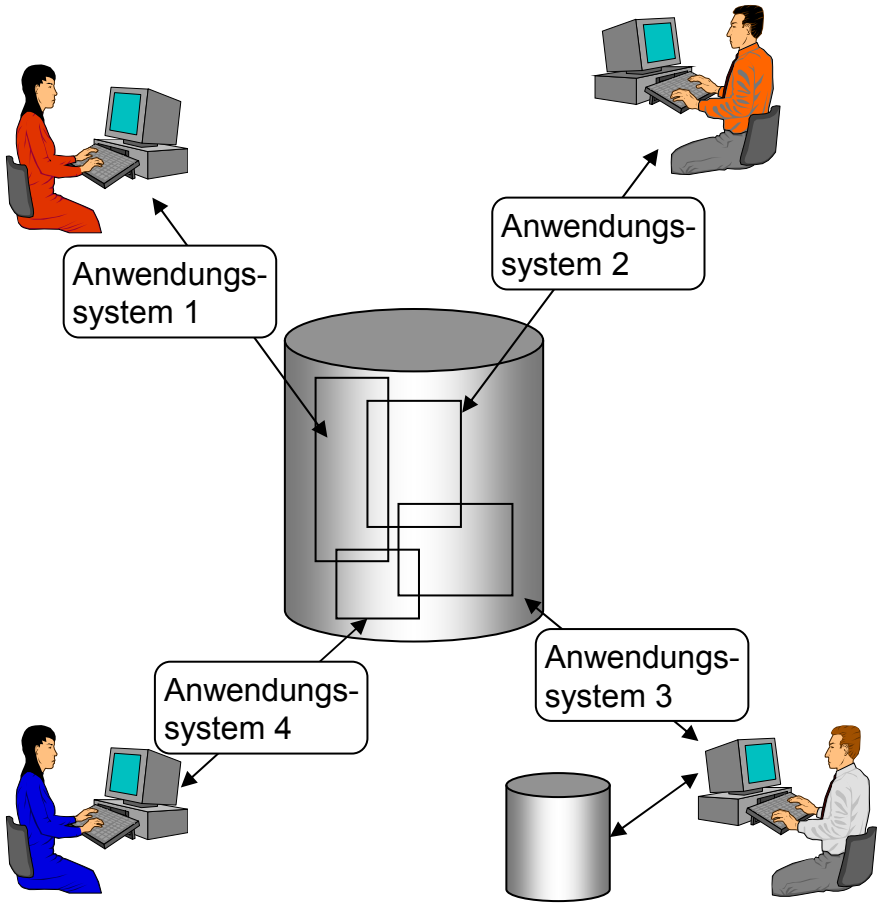
Inhalt

- Betriebliche Informationssysteme
  - Definitionen
  - Charakteristika
- Fachliche Architektur
- Technische Architektur
  - Basismechanismen und Infrastruktur
  - Schichtenarchitekturen und Verteilung
  - Transaktionsverarbeitung
- Datenhaltungsschicht
  - Aufgaben und Charakteristika
  - Realisierungsvarianten
- Zusammenfassung
- Literaturhinweise

Definition: Betriebliches Informationssystem

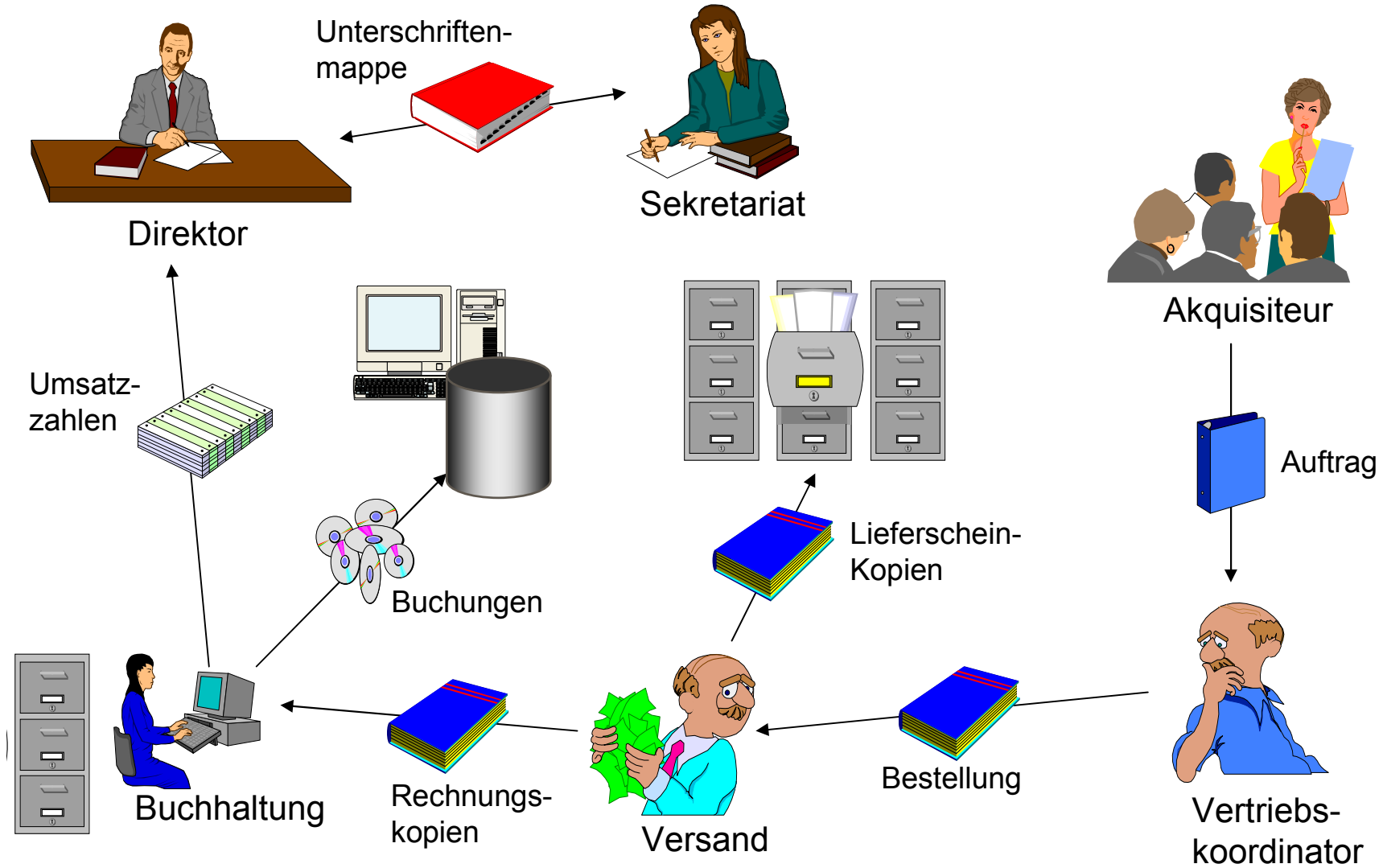
- „Das Informationssystem einer Unternehmung umfasst ihre informationsverarbeitenden Tätigkeiten und Beziehungen."
- „The purpose of an information system is to provide a relatively exact, efficient, unambiguous model of the significant resources of a real-world enterprise."
- „Ein Informationssystem besteht aus Menschen und Maschinen, die Informationen erzeugen und/oder benutzen und die durch Kommunikationsbeziehungen miteinander verbunden sind.
  Ein betriebliches Informationssystem dient zur Abbildung der Leistungsprozesse und Austauschbeziehungen im Betrieb und zwischen dem Betrieb und seiner Umwelt."

Betriebliche Anwendungssysteme



- „... im engeren Sinn die Gesamtheit aller Programme, d.h. die Anwendungssoftware, und die zugehörigen Daten für ein konkretes betriebliches Anwendungsgebiet"
- „ ... im weiteren Sinn zusätzlich die für die Nutzung der Anwendungssoftware benötigte Hardware und Systemsoftware, die erforderlichen Kommunikationseinrichtungen und - je nach Betrachtungsweise - auch die Benutzer ..."

Beispiel: Betriebliches Informationssystem



Typische Konstellation in Großunternehmen

- 10 – 20 Großrechner (Mainframes, Cluster)
- 100 – 500 Datenbankserver
  - 5.000 – 20.000 Tabellen in relationalen Datenbanken
  - 10 bis 20 unterschiedliche Datenbank-Systeme und Versionen
- 30.000 – 100.000 PCs
- 10 – 20 Betriebssysteme und Betriebssystemversionen
- 5.000 – 100.000 Anwendungsprogramme
  - Große betriebswirtschaftliche Standardsoftware mit Anpassungen wie z.B. SAP
  - Maßgeschneiderte SW-Lösungen für spezielle Prozesse (einzelne Batch-Läufe, Excel-Sheets, Abteilungssoftware)
- 500 – 10.000 Mitarbeiter im IT-Bereich
- Weltweit verteilte Standorte

Qualitätskriterien I

- **Reliability**: System liefert stets die erwarteten, richtigen Ergebnisse.
  - Falsche Ergebnisse können geschäftskritisch sein (z.B. Bank, bei der Buchungen verloren gehen).
  - Ergebnisse müssen ggf. nachvollzogen werden können.
- **Availability**: Verfügbarkeit als Verhältnis der Arbeitszeiten eines Systems zu seinen Ausfallzeiten.
  - 60% - 90% aller Aktivitäten in Geschäftsprozessen laufen nicht mehr ohne IT
  - Bei Banken liegt der Überlebenszeitraum beim Totalausfall aller IT-Systeme mittlerweile im Zeitraum von einigen Stunden.
- **Performance**: System erfüllt alle Aufgaben in der erforderlichen Zeit.
  - Bearbeitungsfristen müssen eingehalten werden (Antwortzeiten beim interaktiven Arbeiten, Zeiten für Buchungsläufe im Batch).
  - Aber: Keine harten Echtzeitanforderungen! Durchsatz geht vor Reaktionszeit (z.B. Anzahl der durchgeführten Transaktionen als Kriterium).

Qualitätskriterien II

- **Scalability**: Fähigkeit, einer wachsenden Anzahl von Nutzern zu dienen.
  - Nutzung oft durch Tausende bis Zehntausende Mitarbeiter
  - Bei Internet-Systemen mehrere Millionen Seitenabrufe pro Tag
  - Schnelles Wachstum bzw. hohe Schwankungen der Nutzerzahl bei Internet-Systemen möglich (z.B. bei Web Shops oder Social Media Services)
- **Modifiability**: Notwendige Änderungen können einfach durchgeführt werden. Die Funktionalität des Systems ist leicht zu erweitern.
  - Mischung aus langlebigen und kurzlebigen Anteilen
    - Geschäftsdaten überdauern ca. 20 Jahre
    - Geschäftsprozesse überdauern ca. 2 Jahre
    - Präsentationslogik überdauert ca. 6 Monate
  - Hohe Änderungsfrequenz
    - Pro installierter Programmkopie ca. 2 Wartungen / Jahr.
    - Bei 20 Anwendungsprogrammen, installiert auf 10.000 PCs, und Wartungszyklus von 6 Monaten pro Tag im Schnitt 1.500 Installationen
- **Interoperability**: Anbindung an Daten und Prozesse anderer Systeme.
  - Ein Geschäftsprozess kann bis zu 15 große IT-Systeme umfassen.
  - Fast alle Großunternehmen haben eine heterogene Systemlandschaft.

Inhalt

- Betriebliche Informationssysteme
  - Definitionen
  - Charakteristika
- Fachliche Architektur
- Technische Architektur
  - Basismechanismen und Infrastruktur
  - Schichtenarchitekturen und Verteilung
  - Transaktionsverarbeitung
- Datenhaltungsschicht
  - Aufgaben und Charakteristika
  - Realisierungsvarianten
- Zusammenfassung
- Literaturhinweise

Fachliche Architektur

Daten und Funktionalität

- Finden fachlicher Objekte
- Spezifikation der Funktionen
- Bildung fachlicher Komponenten mit fachlichen Schnittstellen
- Beispiel: eSteuer 2010
  - Spezifikation von fachlichen Objekten mit Klassendiagrammen
  - Komponentendiagramme und Kompositionsstrukturdiagramme zur Spezifikation von Komponenten

→ Schwerpunkte in technischer Software-Architektur:

- Datenhaltungsschicht
- Anwendungsschicht

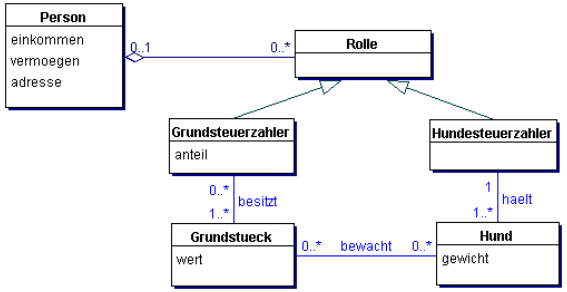
Prozesse und Aktivitäten

- Spezifikation der Geschäftsprozesse
- Spezifikation der Aktivitäten
- Einordnung in Geschäftsprozess-Hierarchie
- Beispiel: folgende Folien
  - Prozesslandkarten als Überblick über alle Prozesse eines Unternehmens
  - UML-Aktivitätsdiagramme zur Spezifikation einzelner Prozesse und Aktivitäten

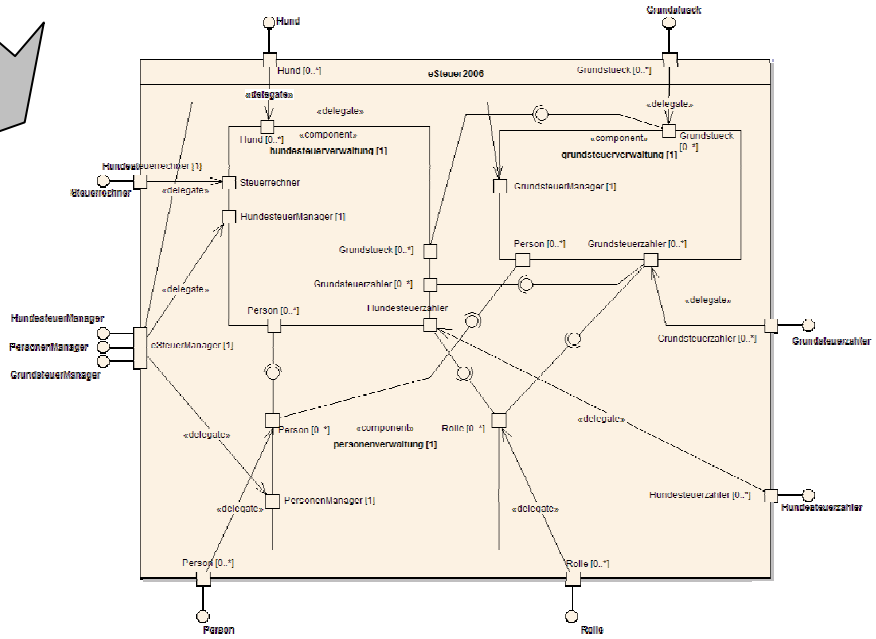
→ Schwerpunkte in technischer Software-Architektur:

- Steuerungsschicht
- Präsentationsschicht

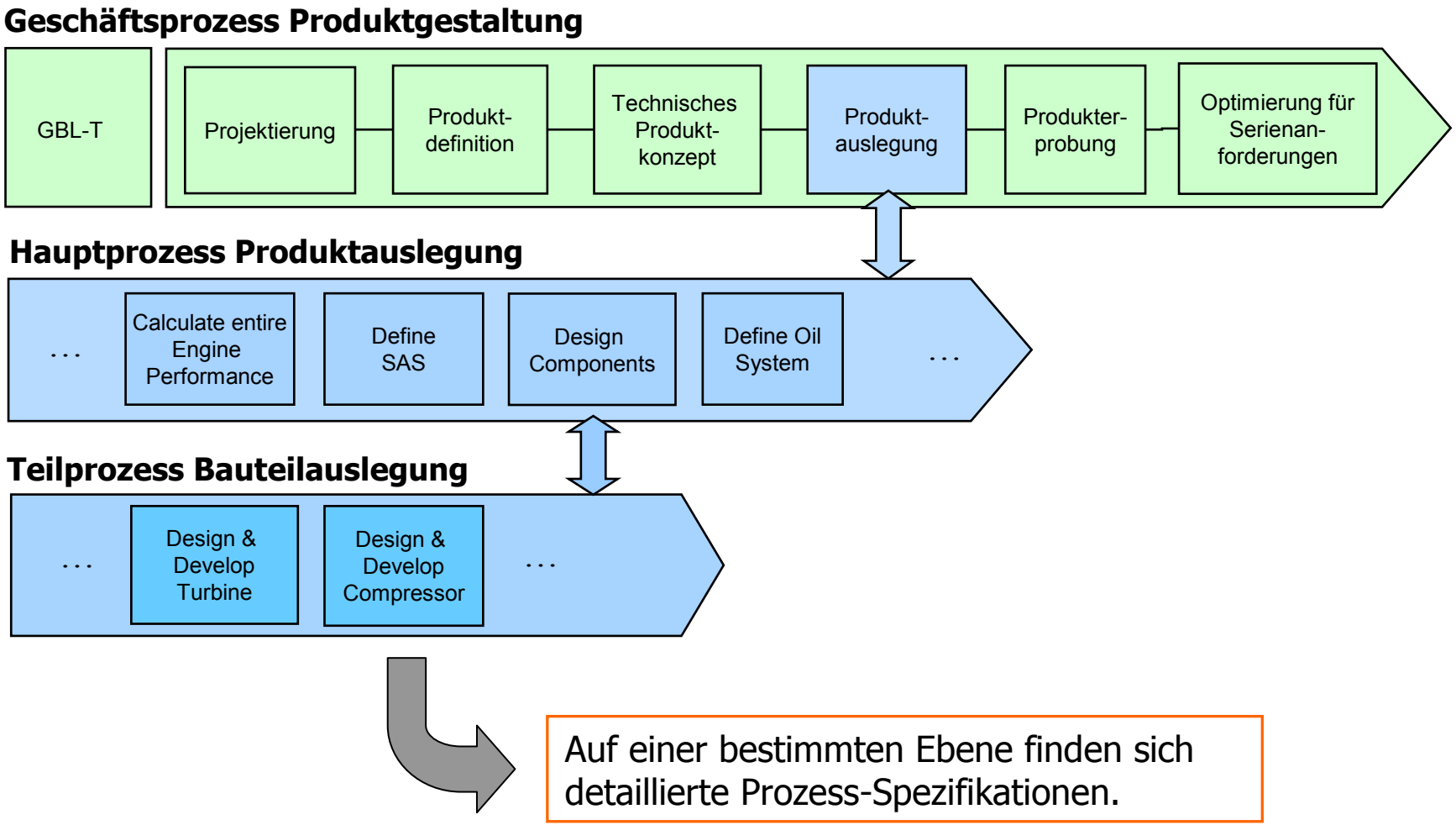
Beispiel: Spezifikation fachlicher Komponenten



eSteuer 2010



Beispiel für Prozesslandkarte

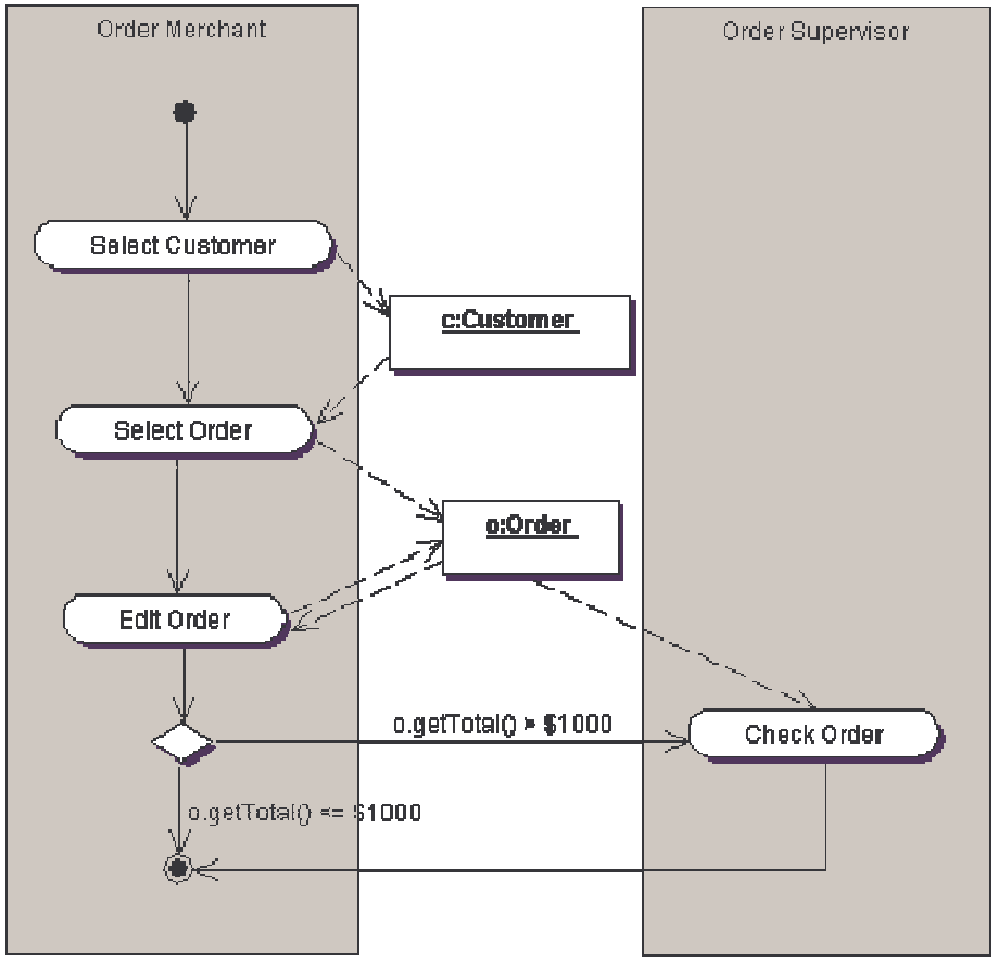


Auf einer bestimmten Ebene finden sich detaillierte Prozess-Spezifikationen.

Beispiel für Prozess-Spezifikation mit Aktivitätsdiagramm

UML-Aktivitätsdiagramm

- Aktoren in Form von Swimlanes
  - Order Merchant
  - Order Supervisor
- Aktivitäten
  - Entsprechen Schritten im Prozess-Ablauf
  - System und/oder Aktoren tun etwas
    - Interaktion
    - Aufruf fachlicher Funktionalität
- Zwischen den Aktivitäten fließen Objekte aus dem fachlichen Objektmodell.



Inhalt

- Betriebliche Informationssysteme
  - Definitionen
  - Charakteristika
- Fachliche Architektur
- Technische Architektur
  - Basismechanismen und Infrastruktur
  - Schichtenarchitekturen und Verteilung
  - Transaktionsverarbeitung
- Datenhaltungsschicht
  - Aufgaben und Charakteristika
  - Realisierungsvarianten
- Zusammenfassung
- Literaturhinweise

Basismechanismen und Infrastruktur

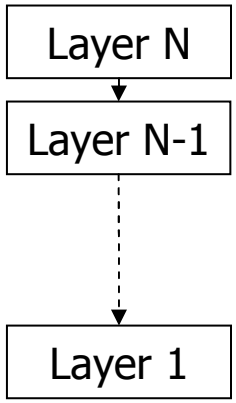
- Einige Techniken und Standards haben sich allgemein etabliert, z.B.
  - Komponenten und Standards wie Datenbanken, SQL
  - Protokolle und Formate wie HTTP, XML, SOAP
- Auf den meisten Gebieten sehr schnelle Entwicklung
  - Ansteuerung relationaler Datenbanken in JEE: EJB1 → EJB2 → EJB3
  - Neue Techniken für Web-GUIs: Servlets, Applets, JSPs, AJAX, Comet
- Neben Basiskomponenten finden sich in allen Bereichen auch Frameworks und Standard-Architekturen.
  - z.B. Sun PetStore Demo für EJB und andere Demo-Anwendungen

- Anforderungen an den Architekten
  - Bewährte, im Unternehmen gesetzte Techniken und Systeme einbinden
  - Neue Techniken evaluieren und in die Architektur integrieren

Schichtenarchitekturen - Rekapitulation

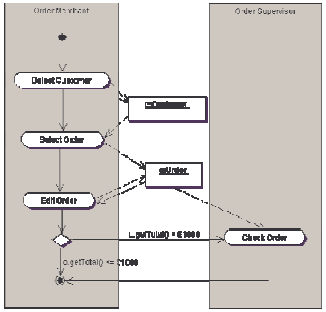
- Im Bereich betrieblicher Informationssysteme haben sich mehrschichtige Architekturen als Standard durchgesetzt.
- Rekapitulation: Schichten-Architekturmuster
  - **Lösung:** Strukturiere das System in Schichten aus Komponenten, die auf der gleichen Abstraktionsebene angesiedelt sind. Komponenten einer höheren Schicht können zur Erledigung ihrer Aufgabe auf Komponenten der gleichen und der unmittelbar darunter liegenden Schicht zurückgreifen.
  - **Struktur:**

Class Schicht J	**Collaboration** Schicht J-1
Responsibility • stellt Dienste für Schicht J+1 zur Verfügung • ruft Dienste von Schicht J-1 auf	

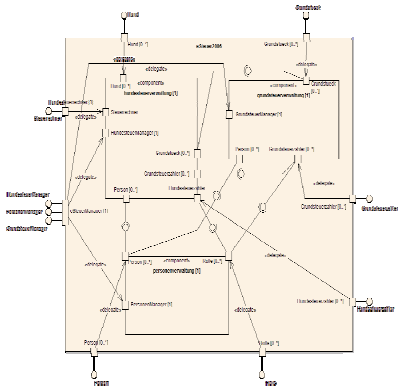


Logische Schichtenarchitektur

Die Aspekte der fachlichen und technischen Architektur werden vier Schichten zugeordnet. Die Unterstützungsschicht bietet querschnittliche Funktionalität.



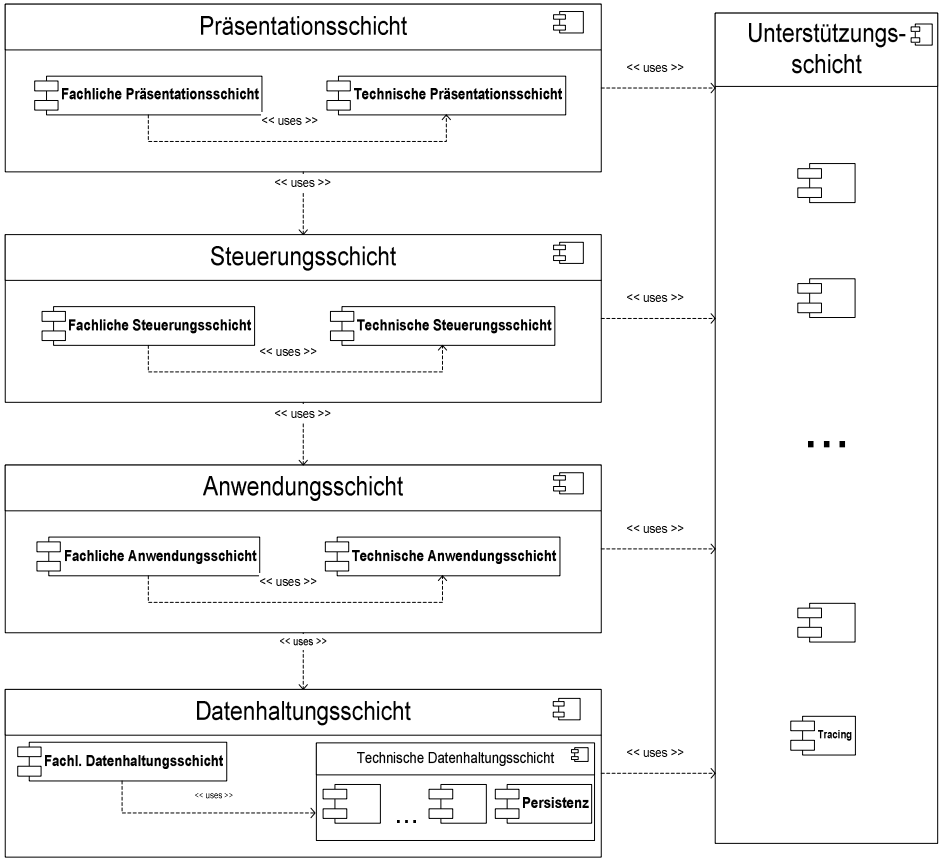
GUIs zu Aktivitäten



Workflows zu Prozessen

Fachliche Komponenten

Datenlayout zu Objekten



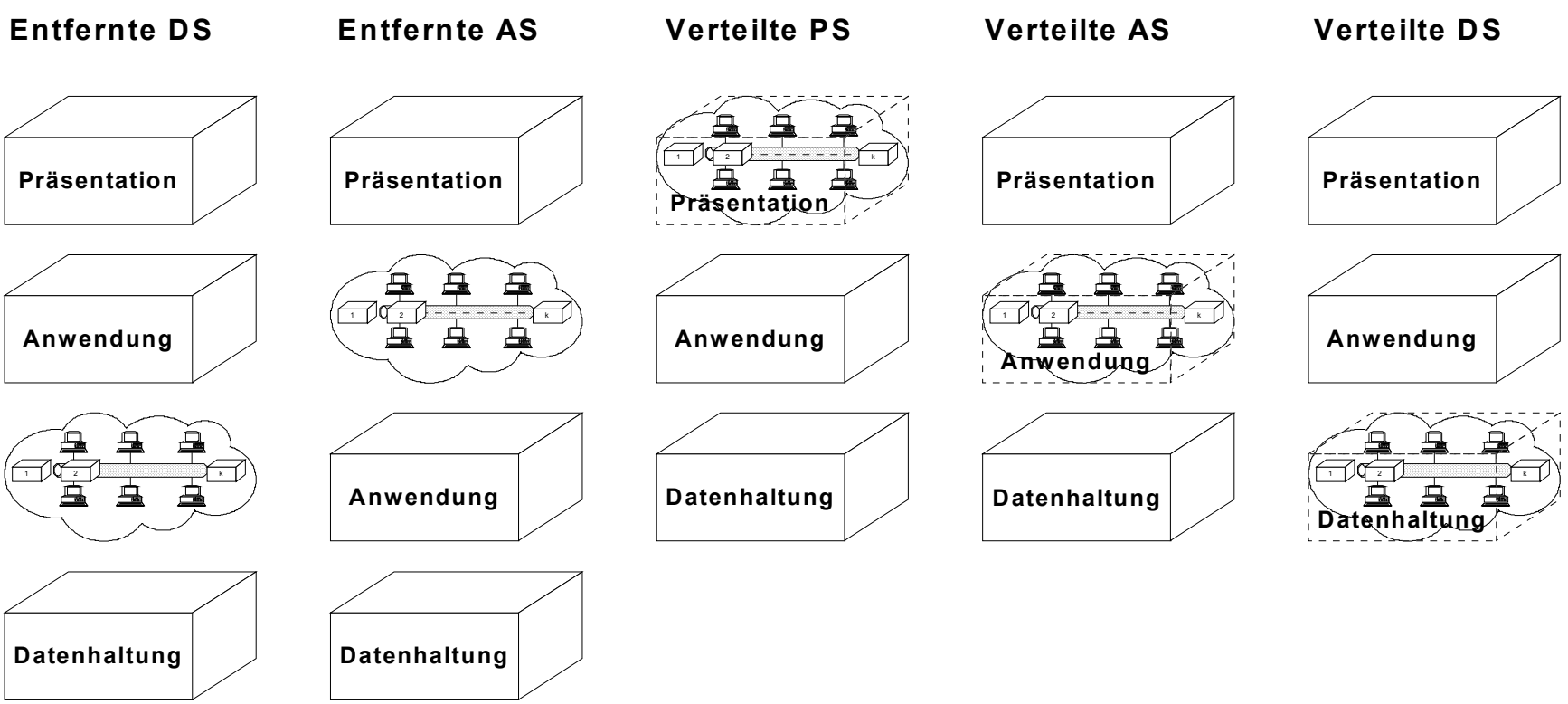
Überblick über die logischen Schichten

- Präsentationsschicht
  - Steuert die Interaktion mit dem Benutzer.
  - Führt selbständig einfache Prüfungen durch.
- Steuerungsschicht (oft mit Anwendungsschicht zusammengefasst)
  - Bietet Ablaufumgebung für die spezifizierten Geschäftsprozesse.
  - Steuert den Ablauf der Aktivitäten (inklusive fachlicher Funktionalität und Dialogen der Präsentationsschicht).
- Anwendungsschicht
  - Bietet Zugriff auf Geschäftsobjekte und deren Funktionalität.
  - Kapselt proprietäre technische Dienste hinter Schnittstellen und bietet damit einfach anzusteuernde Anwendungsdienste.
- Datenhaltungsschicht
  - Speichert und verwaltet die Geschäftsobjekte der Anwendung.
  - Prüft und überwacht selbständig gewisse Konsistenzkriterien.
  - Abstrahiert von der zugrunde liegenden Datenbanktechnologie.

Verteilung und physische Schichtung

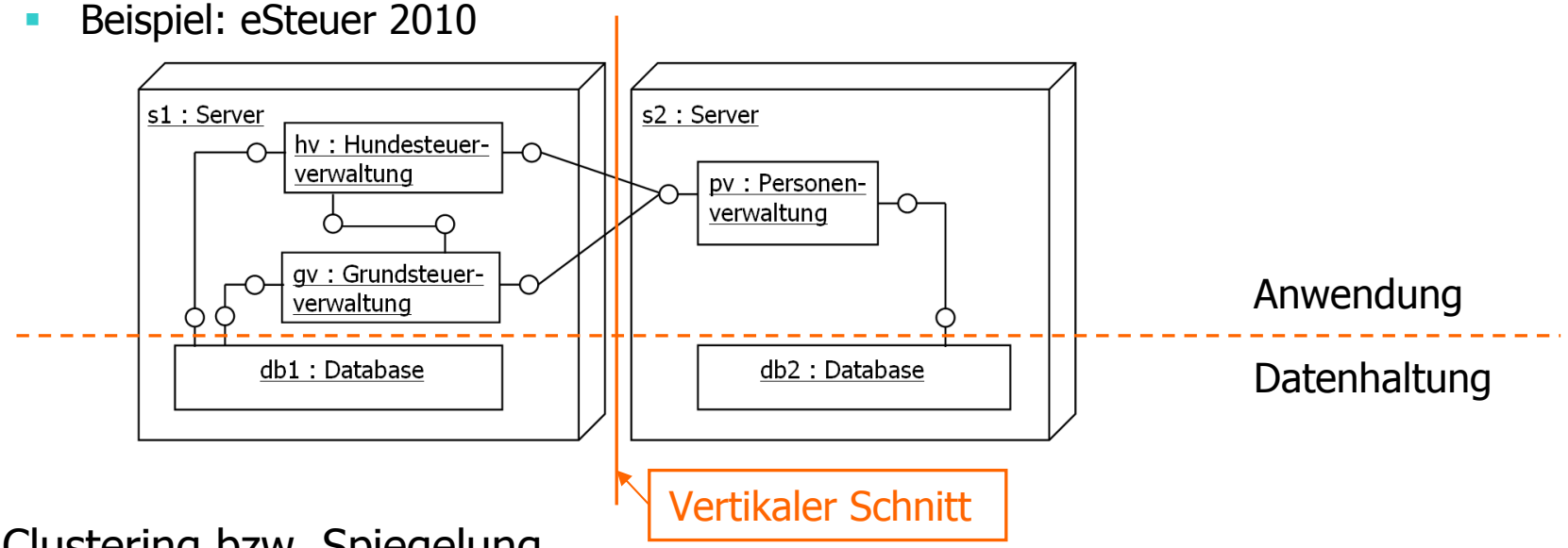
- Die logischen Schichten können unterschiedlich auf mehrere Rechner verteilt werden. Optionen sind:
  - Horizontaler Schnitt zwischen zwei Schichten
  - Horizontaler Schnitt innerhalb einer Schicht
  - Vertikaler Schnitt innerhalb einer Schicht
- Gründe für … und … gegen Verteilung
  - \+ Ortsgebundene Funktionalität
    - \+ Präsentation
    - \+ Rechtliche Aspekte
  - \+ Performanzgewinn durch Parallelisierung
  - \+ Erhöhte Zuverlässigkeit
    - \+ Spiegelung, Clustering
    - \+ Weiterarbeiten nach Teil-Ausfällen
  - \+ Integration der Funktionalität vorhandener Systeme
  - \- Performanzverlust durch Kommunikation
    - \- Erhöhung der Antwortzeiten
    - \- Netzwerk als Flaschenhals
  - \- Verringerte Zuverlässigkeit
    - \- Einsatz komplexer Middleware
    - \- Beherrschung verteilter Abläufe
  - \- Test aufwändig
  - \- Installation und Betrieb aufwändig

Horizontaler Schnitt zwischen Client und Server



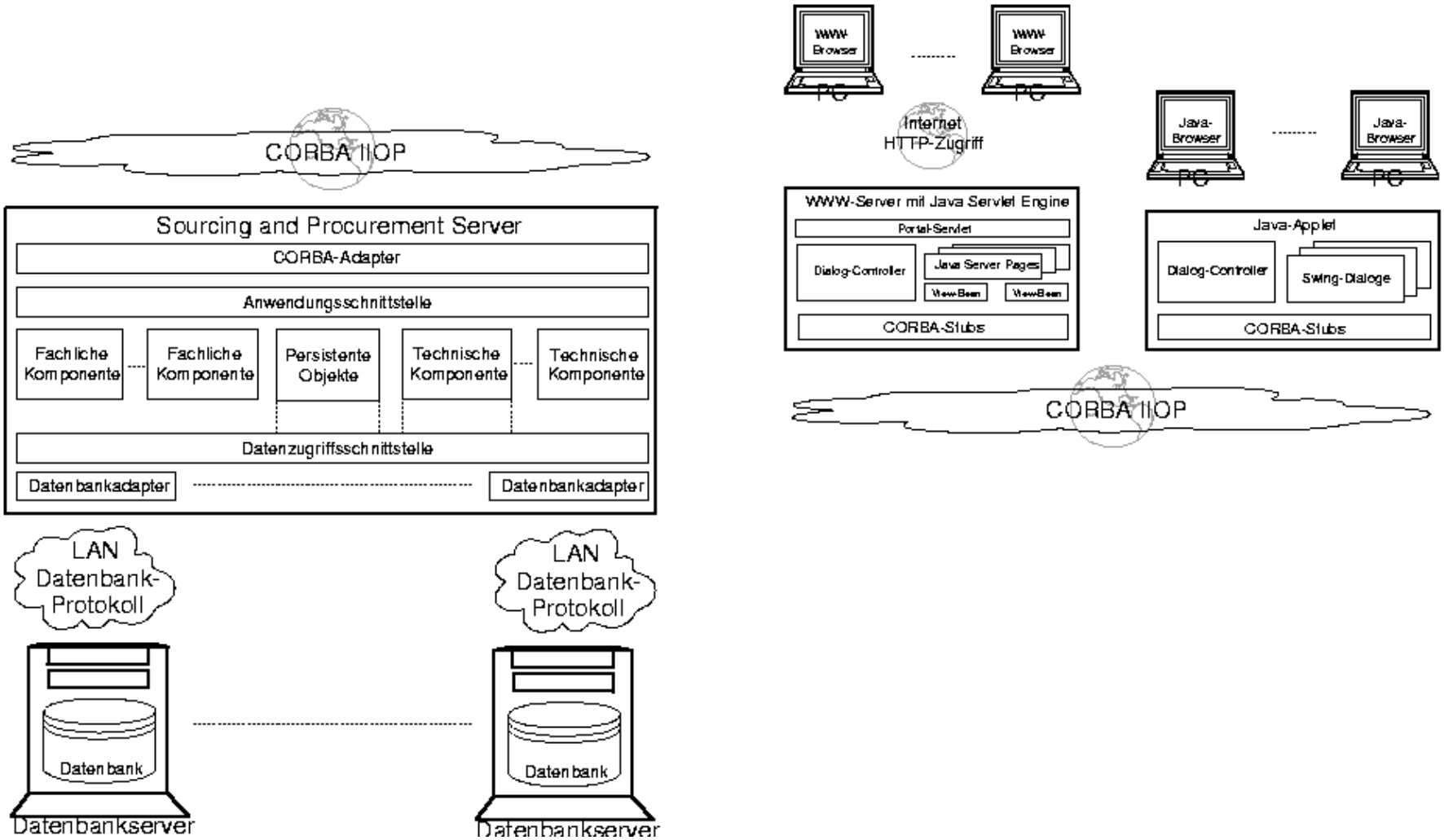
Vertikaler Schnitt innerhalb von Schichten

- Verteilung der Funktionalität
  - Jeder Rechner übernimmt eine bestimmte Aufgabe.
  - Beispiel: eSteuer 2010

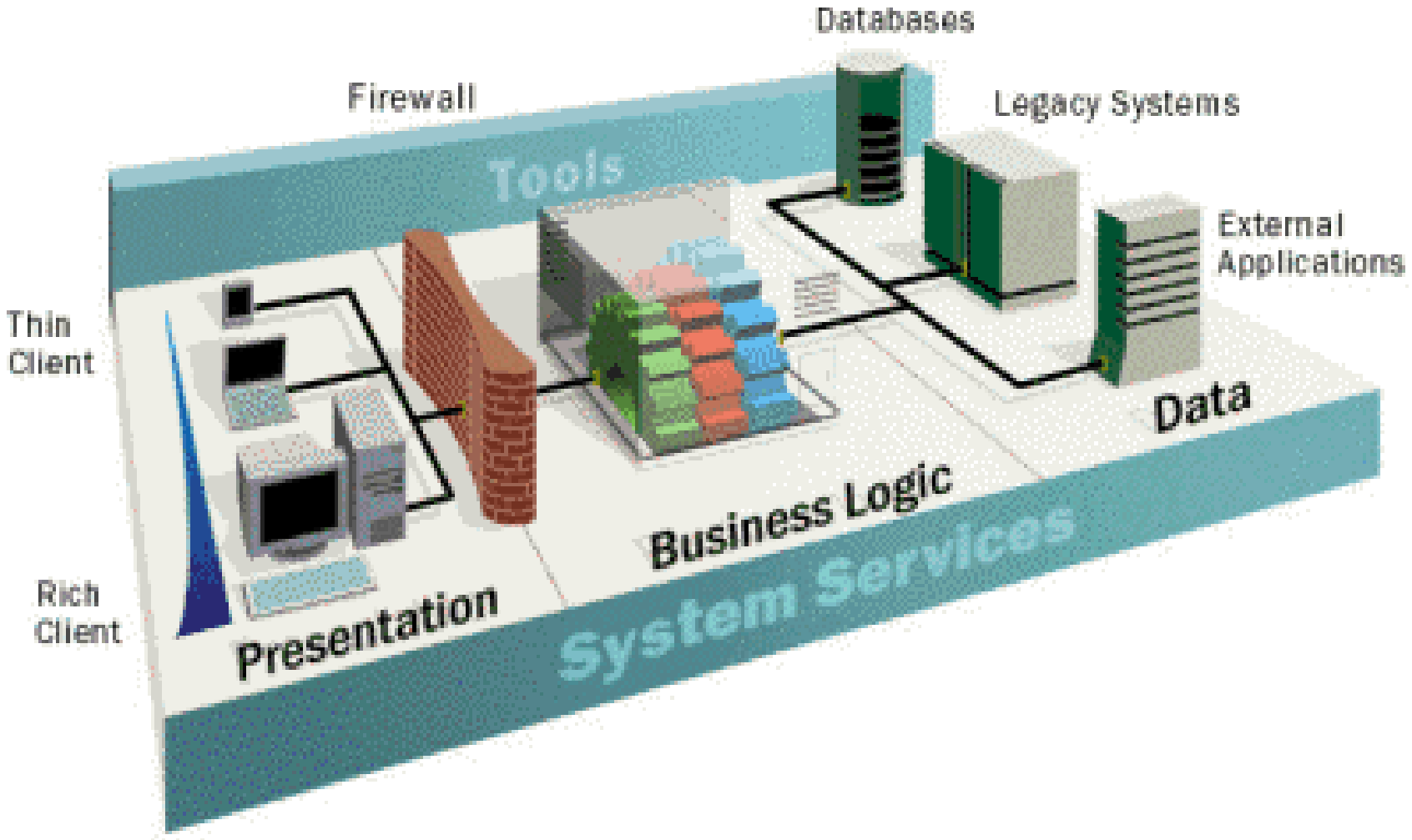


- Clustering bzw. Spiegelung
  - Die selbe Funktionalität wird auf unterschiedlichen Servern bereitgestellt.
  - Bei Änderungen müssen sich die einzelnen Server synchronisieren.
  - Beispiele:
    - Verteiltes Datenbanksystem
    - Clustering bei Applikationsservern

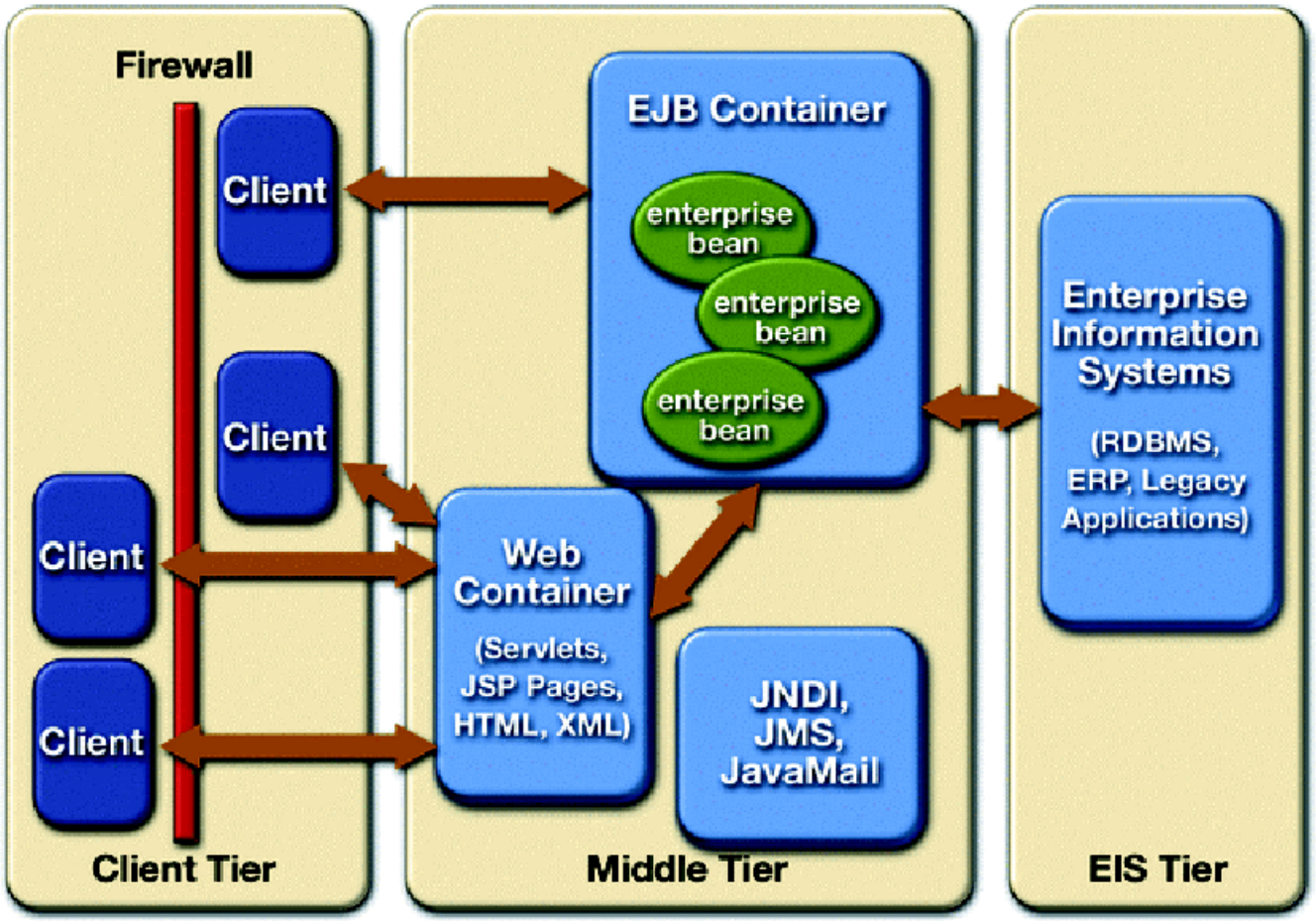
Beispiel: eCommerce Systems



Beispiel : Drei-Schichten-Architektur

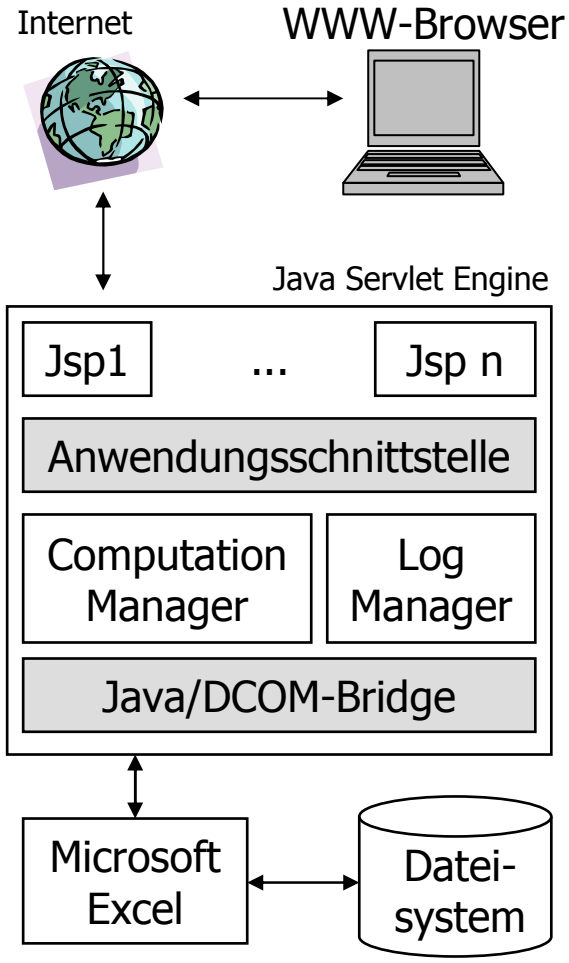


Beispiel: 3-/4-Schichten-Modell in J2EE



Beispiel: Architektur mit Excel-Berechnungsengine

- Anwendungsfall
  - Berechnungen in einem Excel-Sheet sollen über Standard-Browser zur Verfügung gestellt werden
- Architektur
  - Ultra-Thin Client, Generierung von drei unterschiedlichen Web-Seiten über JavaServer Pages
  - Zwei Dienste in der Anwendungs-schicht (Berechnungen durchführen, Berechnungslog führen)
  - Ansteuerung von Excel als Berechnungs-Engine, Datenhaltung in normalen Excel-Dateien



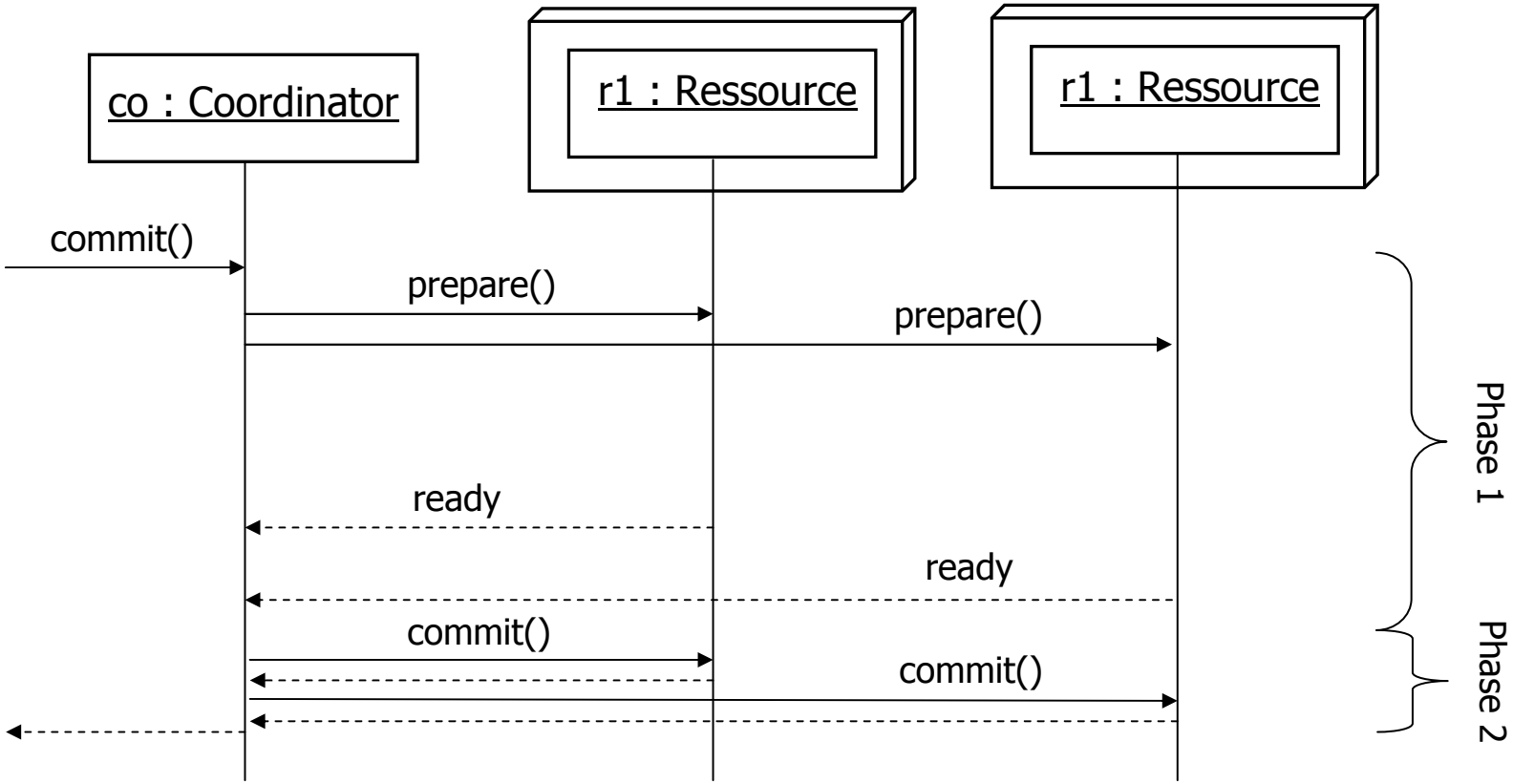
Transaktionsverarbeitung

- Betriebliche Informationssysteme basieren auf Transaktionen als Standardmechanismus zur Verwaltung paralleler Aktivitäten.
- Der Begriff der Transaktion hat eine Doppelbedeutung
  - Geschäftliche Transaktion
  - Technische Transaktion

  allerdings fallen beide Bedeutungen in vielen Systemen zusammen.
- Eigenschaften von Transaktionen:
  - **Atomicity** (Unteilbarkeit): Eine Transaktion wird entweder vollständig oder gar nicht ausgeführt. Bei Fehlern während der Abarbeitung wird jede bereits stattgefundene Aktion der Transaktion rückgängig gemacht.
  - **Consistency** (Konsistenz): Nach Ablauf einer Transaktion ist das System in einem korrekten Zustand, während des Ablaufs einer Transaktion werden möglicherweise inkonsistente Zwischenzustände durchlaufen.
  - **Isolation**: Eine Transaktion läuft so ab, als wäre sie die einzige im System. Parallel ablaufende Transaktionen dürfen einander nicht beeinflussen.
  - **Durability** (Dauerhaftigkeit): Änderungen, die von einer erfolgreichen Transaktion durchgeführt werden, überdauern jeden nachfolgenden Fehlerfall.

Transaktionen und verteilte Systeme: Two-Phase-Commit / 2PC

- Wenn die transaktionalen Ressourcen in mehreren Systemen abgelegt sind, spricht man von globalen Transaktionen (im Gegensatz zu lokalen).
- Mögliche technische Lösung: Globale Transaktion mit Two-Phase-Commit



CAP-Theorem (Eric Brewer, 2000)

- Verteilte Anwendungen können nur zwei von drei der folgenden Anforderungen erfüllen:
  - **Consistency**: Alle Knoten sehen jederzeit die selben Daten.
  - **Availability**: Ausfälle einzelner Knoten beeinträchtigen die Arbeitsfähigkeit der restlichen Knoten nicht.
  - **Partition Tolerance**: Das System arbeitet trotz willkürlicher Verluste von Nachrichten weiter.
- In großen Systemen ist Partition Tolerance essenziell – man kann nicht davon ausgehen, dass alle Systemteile immer fehlerfrei arbeiten. Das System darf dann nicht zu arbeiten aufhören.
- Man wählt also übergreifende Konsistenz **oder** hohe Verfügbarkeit.
- Beispiel Buchbestellung bei Amazon:
  - Ziel ist, alle Bestellungen schnell zu bearbeiten und zu bestätigen (Availability). Man nimmt in Kauf, dass einige Bestellungen fälschlich bestätigt werden (fehlende Konsistenz) und schickt dann später eine Entschuldigungsmail, in der die schon bestätigte Bestellung doch noch storniert wird.

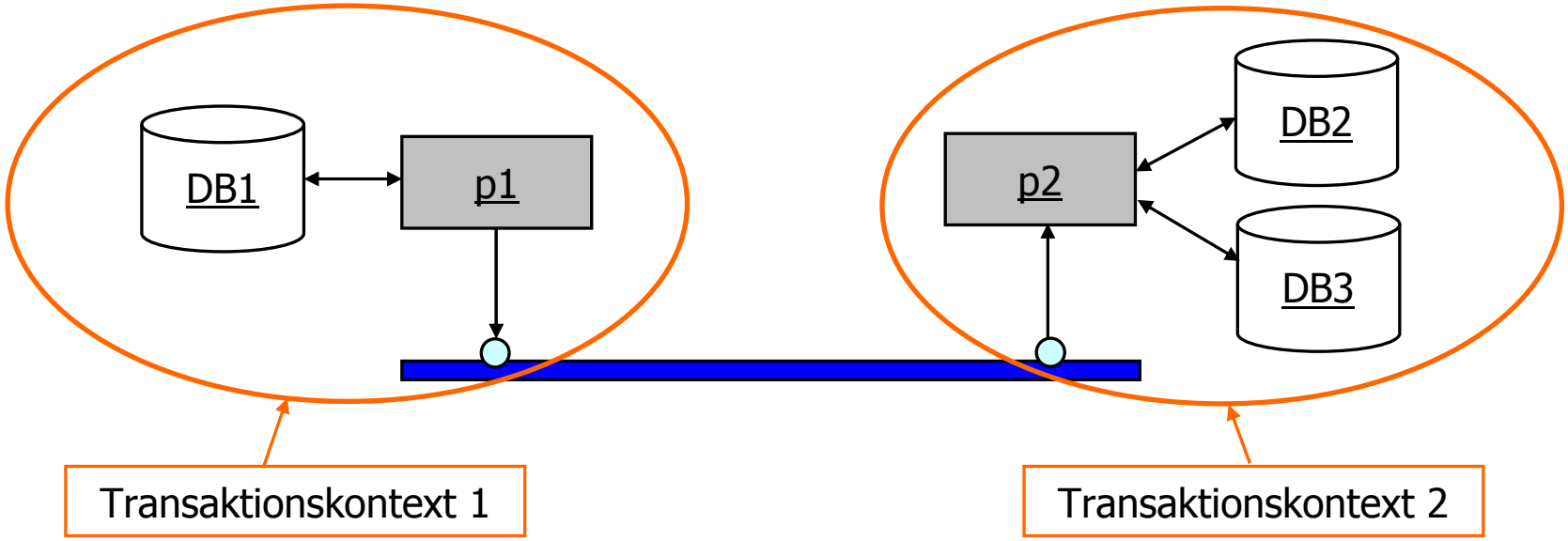
BASE statt ACID

- BASE steht für
  - Basically Available: Basisfunktionalität ist stets verfügbar.
    - Bücherbestellungen funktionieren immer …
  - Soft-State: Es gibt keinen globalen, jederzeit konsistenten Zustand.
    - Nicht alle Knoten wissen immer genau, wie viele Bücher es noch gibt …
  - Eventually Consistent: Irgendwann wird aber die Konsistenz hergestellt.
    - Irgendwann erfährt ein Knoten, ob seine Bestellung möglich war …

- BASE ist damit das „Gegenteil" von ACID:
  - Verfügbarkeit statt Konsistenz
  - Optimistisch statt pessimistisch
  - Best Effort statt exakter Antworten
  - Schnell und einfach statt langsam und komplex

Umsetzung von BASE mit ATM

- Die Unteilbarkeit wird geopfert: Statt einer globalen Transaktion gibt es beim **A**synchronen **T**ransaktionalen **M**essaging mehrere lokale.
- Dabei sind das Ablegen von Nachrichten in einen Nachrichten-Kanal und das Herausnehmen von Nachrichten aus dem Kanal durch lokale Transaktionen geschützt.
- Wenn der Kanal die Daten sicher speichert und weiterleitet, kann man damit in vielen Anwendungen auf globale Transaktionen verzichten.



Globale Transaktionen versus ATM

Globale Transaktionen

+ Einfache Realisierung von Geschäftsvorfällen
+ Global einheitlicher Datenzustand
– Kommunikationsaufwand durch Two-Phase-Commit-Protokoll
– Macht globale Sperren auf Objekten erforderlich
– Mangelnde Skalierbarkeit
– Sämtliche Ressourcen müssen für Erfolg gleichzeitig verfügbar sein
– Nicht von allen Ressourcen unterstützt

ATM

+ Entkopplung der Partner bei der Kommunikation
+ Flexible Konfiguration von Routing, Filtering etc.
+ Performanz und Skalierbarkeit
+ Robustheit bei temporären Ausfällen von Ressourcen
+ Einsatz bewährter Standard-Komponenten (z.B. MQSeries, Tibco, JMS-Server)
– Verteilte Realisierung von Geschäftsvorfällen
– Kein global einheitlicher Datenzustand

In der Praxis: Globale Transaktionen sind nur relativ selten einsetzbar.

Inhalt

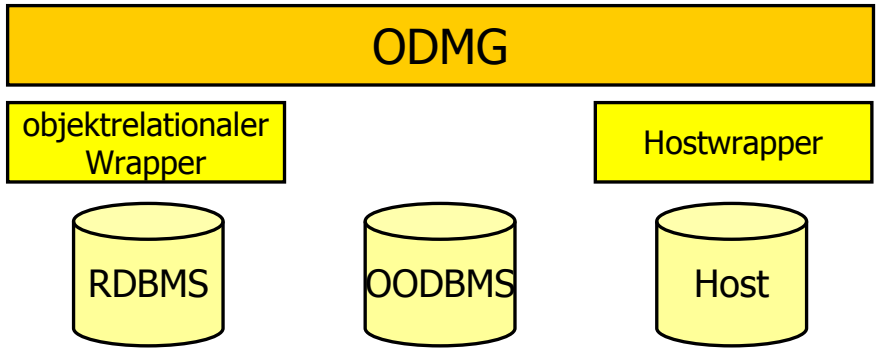
- Betriebliche Informationssysteme
  - Definitionen
  - Charakteristika
- Fachliche Architektur
- Technische Architektur
  - Basismechanismen und Infrastruktur
  - Schichtenarchitekturen und Verteilung
  - Transaktionsverarbeitung
- Datenhaltungsschicht
  - Aufgaben und Charakteristika
  - Realisierungsvarianten
- Zusammenfassung
- Literaturhinweise

Aufgabe der Datenhaltungsschicht

- Speichert und verwaltet die Geschäftsobjekte der Anwendung.
  - Bildet das Anwendungs-Objektmodell auf das Datenschema der zugrunde liegenden Datenbanken ab.
  - Stellt persistente Daten als Objekte in der Anwendungsschicht zur Verfügung.
  - Bietet Dienste für die Verwaltung der Daten wie beispielsweise
    - Ausführung von Datenbank-Abfragen
    - Ausführung von fachlicher Logik (Stored Procedures)
    - Einbindung in Transaktionsverwaltung
    - Schlüsselmanagement (ID-Verwaltung)
- Prüft und überwacht selbständig gewisse Konsistenzkriterien.
  - Referenzielle Integrität von Datenbank-Beziehungen
  - Löschen abhängiger Objekte
- Abstrahiert von der zugrunde liegenden Datenbanktechnologie.
  - Transparente Abbildung auf den verwendeten Persistenzmechanismus

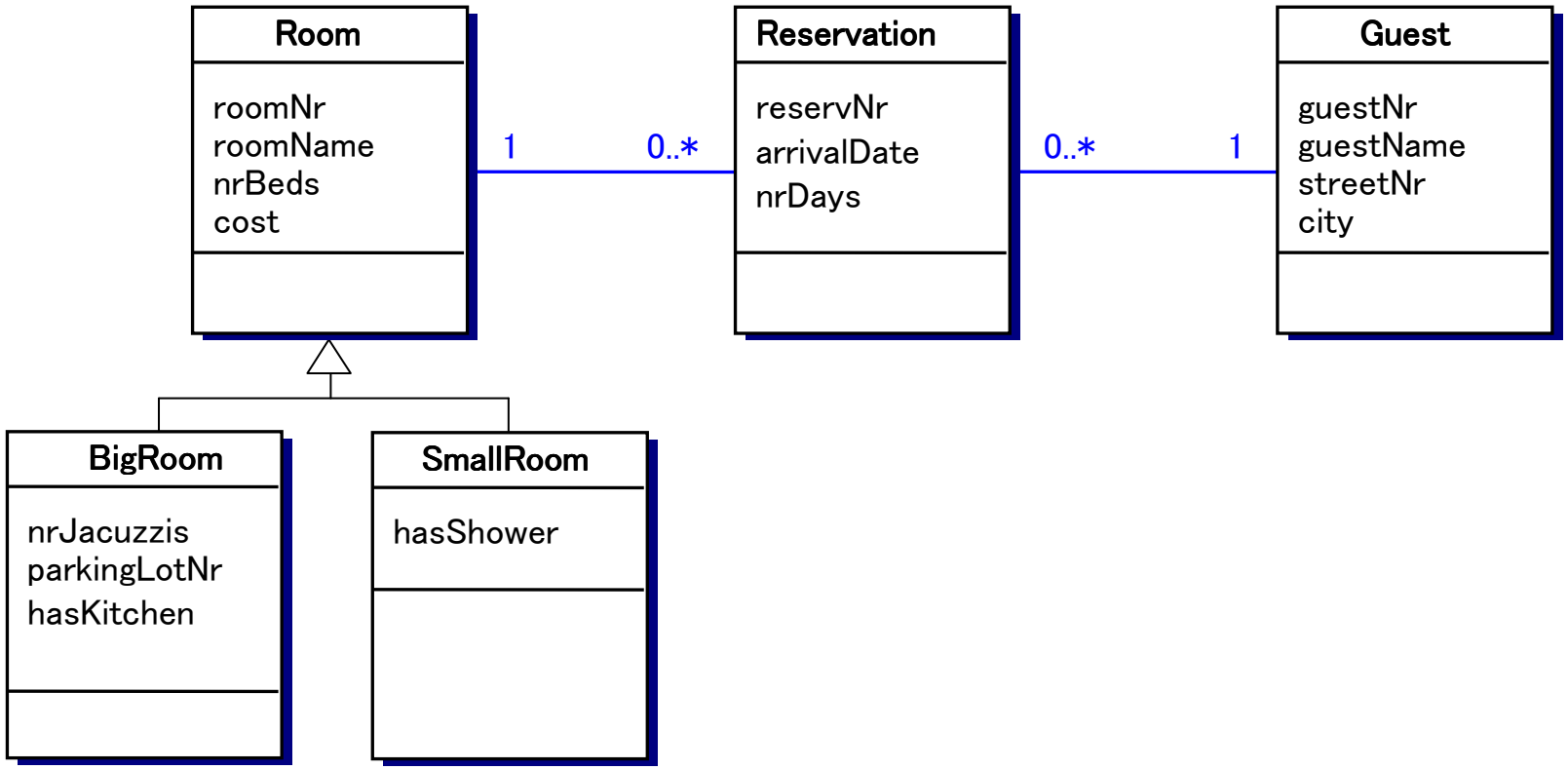
Abstraktion der Datenhaltung

- Die dauerhafte Speicherung der Informationen nutzt häufig unterschiedliche Produkte und Technologien
  - Relationale oder objektorientierte Datenbanksysteme
  - XML-basierte Datenbanken
  - Hierarchische und andere Legacy-DBMS
  - Hostsysteme
  - Dateisysteme
- Eine objektorientierte Verwaltungsschnittstelle erleichtert die Realisierung der Anwendungsschicht und bietet Flexibilität und Erweiterbarkeit.
- Beispiel:



Einfaches Beispiel

Ausschnitt aus dem fachlichen Datenmodell eines Hotels:



Warum objektorientierter Datenzugriff?

- Methodenbruch bei direktem Zugriff auf Datenquelle führt zu unhandlichem und fehlerträchtigem Code.
  - Beispiel: Direkter Zugriff auf SQL-Datenbank

```
ResultSet rs = stmt.executeQuery(
  "SELECT * FROM Reservation WHERE guestNr = " + myGuestNr);
while ( rs.next() ) {
  rnr = rs.getInteger(„reservNr");
  ard = rs.getDate(„arrivalDate");
  nrd = rs.getString(„nrDays");
  System.out.println(rnr + ard + nrd);
}
```

  - Beispiel: Objektorientierter Zugriff

```
Set<Reservation> reservations = myGuest.getReservations();
foreach (r in reservations) {
  System.out.println(r.reservNr + r.arrivalDate + r.nrDays );
}
```

- Typische Objekt-Operationen wie Attribut-Zugriff und Navigation sind auf Objekten im Speicher sehr effizient (Objekte als Cache).

Möglichkeiten zur Realisierung des objektorientierten Zugriffs

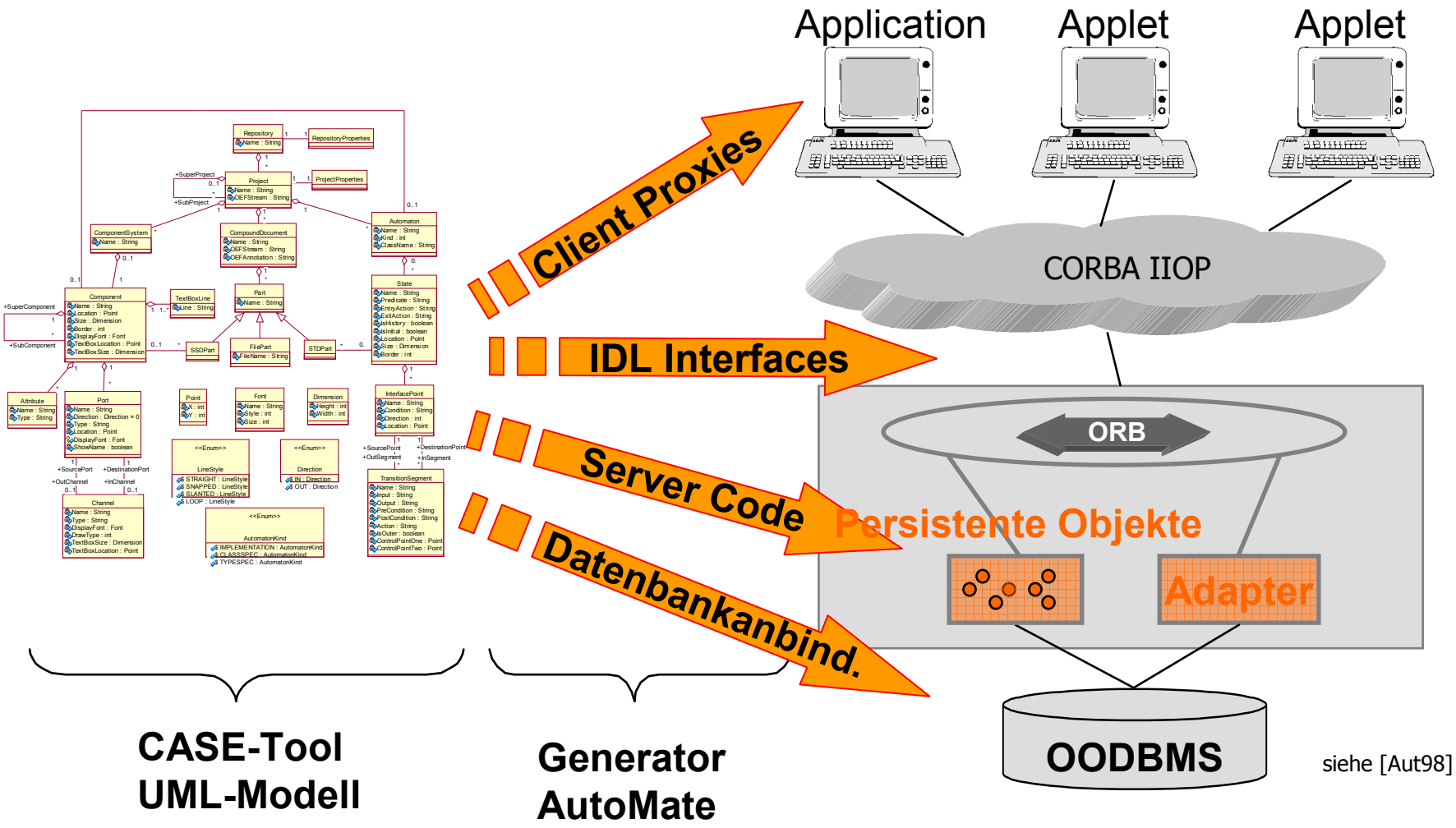
Objektorientierte Datenbank

- Spezielle Datenbank, welche die Speicherung von Objekten nativ unterstützt.
- Standardisierung durch ODMG (Object Data Management Group)
  - Schemabeschreibung: ODL
  - Abfrage-Sprache: OQL
  - Persistenz-Dienste: ODMG Interfaces
- Insgesamt bisher nur in Marktnischen erfolgreich
  - Wenige Anbieter mit teils proprietären Systemen
  - Keine Nutzung existierender Unternehmens-Datenbanken

Objekt-Relationales Mapping

- Zugriffsschicht auf relationale Datenbanken, bieten objektorientierte Schnittstelle.
- Einige Standards, u.a. Java EJB3 Persistence Management, JPA, .NET LINQ, Hibernate.
  - Schemabeschreibung: z.B. über Annotations im Quellcode
  - Abfrage-Sprachen: EJB-QL, Criteria, LINQ
  - Persistenz-Dienste: EntityManager Interface und andere Dienste
- Vorteil: Vorhandene, ausgereifte relationale Datenbank-Systeme können als Basis weiter genutzt werden.

Beispiel: Datenhaltung in ODBMS, Code-Generierung



siehe [Aut98]

Objekt-Schemata: Beispiel **O**bject **D**efinition **L**anguage

Schlüssel als Zusatzkonzept

```
interface Room ( extent Rooms, keys roomNr) : persistent {
   attribute unsigned Long roomNr;
   relationship Set<Reservation> is_reserved_in_Reservation
      inverse Reservation::is_for_Room;
   attribute String roomName;
   attribute unsigned Short nrBeds;
   attribute unsigned Short cost;
}

interface Reservation ( extent Reservations keys reservNr ):
persistent {
   relationship Room is_for_Room
      inverse Room::is_reserved_in_Reservation;
   relationship Guest is_for_Guest
      inverse Guest::has_Reservation;
   attribute String arrivalDate;
   attribute unsigned Short nrDays;
   attribute unsigned Long reservNr;
}
```

Assoziationen als Zusatzkonzept
DB überwacht referenzielle Integrität

Daten-Abfrage: **O**bject **Q**uery **L**anguage

- ODMG-Standard spezifiziert OQL als deklarative Anfragesprache für das ODMG-Objektmodell.
- Weitgehende Kompatibilität mit SQL-95.
- Strenge Typisierung:
  - Alle Ausdrücke haben statisch bestimmbare Typen.
  - Mengenwertige Ausdrücke geben Collection-Typen zurück.
  - Struct-Typen können beliebig zusammengebaut werden.
- Orthogonalität
  - Alle Ausdrücke können, soweit ihre Typen verträglich sind, beliebig kombiniert werden.
  - Keine syntaktischen Ausnahmen und Sonderstellungen für spezielle Typen.
- Wahrung der Objektkapselung
  - Zustandsänderungen nur durch Aufruf entsprechender Methoden.

Beispiel-Abfragen mit OQL

- Gäste mit Reservierungen über mehr als 1 Tag:

```
select x
   from x in Guests, y in x.has_Reservation
   where y.nrDays > 1
```

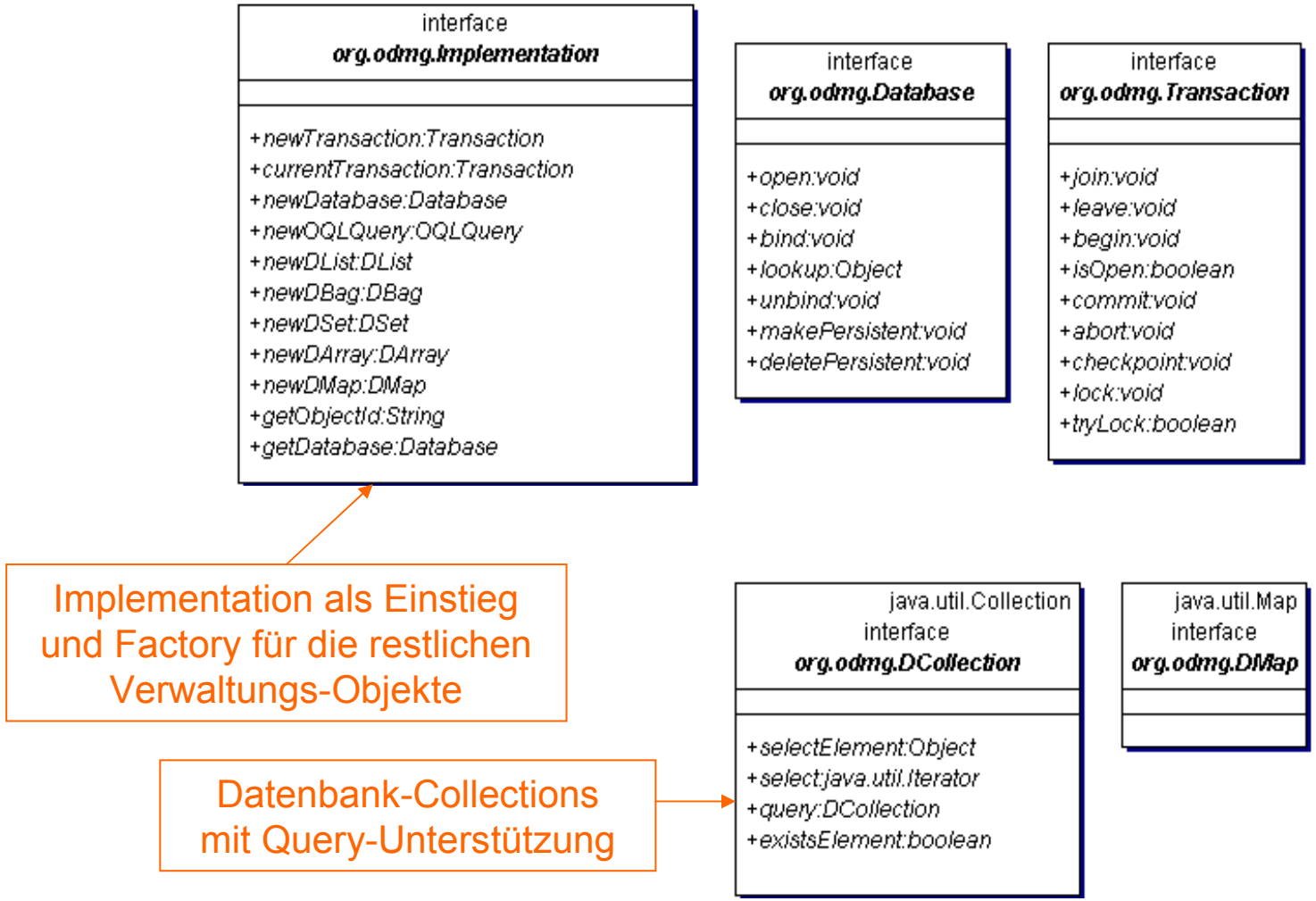
- Besteht eine Reservierung für die Kennedy-Suite am 13. Mai?

```
exists x in Reservations : x.arrivalDate = "13 May"
         and x.is_for_Room.RoomName = "Kennedy"
```

- Raumnamen, Ankunftsdatum und Stadt des Gastes der reservierten Räume:

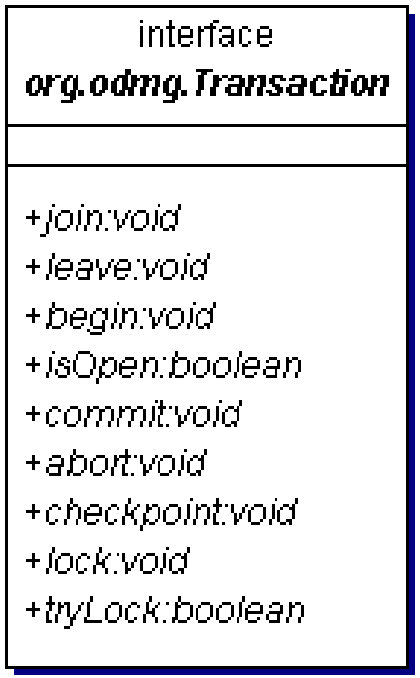
```
select struct(x.roomName,
   (select struct(y.arrivalDate, y.is_for_Guest.City)
      from y in x.is_reserved_in_Reservation))
   from x in Rooms
```

ODMG-Verwaltungsschnittstelle in Java (Ausschnitt)



Transaktionen bei ODMG

- Transaktionen sind selbst Objekte, die man zunächst erzeugen muss, bevor man sie startet, abschließt oder abbricht.
- Erzeugen, Modifizieren, Lesen und Löschen persistenter Objekte ist nur innerhalb von Transaktionen möglich.
- Transaktionen als Kontext
  - Transaktionen müssen nicht explizit bei jeder Operation an persistenten Objekten übergeben werden …
  - … sondern sind implizit als Kontext an den aktuellen Thread gebunden.
  - Threadwechsel über join() und leave() möglich.



OQL-Anbindung an Java

- Generische OQLQuery-Schnittstelle

```
interface OQLQuery {
  public void create(String query);
  public bind(Object parameter);
  public Object execute(); }
```

- *create*() setzt den Abfrage-String, ohne die Abfrage jedoch auszuführen, Parameter haben die Syntax $1, $2, $3 etc.
- *bind*() bindet aktuelle Parameter an die formalen, wobei die Reihenfolge eine Rolle spielt: beim ersten Aufruf wird das Parameterobjekt an $1 gebunden, usw.
- *execute*() führt die Anfrage schließlich aus und liefert eine passende Kollektion in Form eines Objektes zurück.

Beispiel: Objekt-Relationales Mapping mit EJB3

- Insgesamt sind die Mechanismen von EJB3 sehr ähnlich zu den ODMG-Mechanismen für objektorientierte Datenbanken.
- Allerdings gibt es im Detail einige Unterschiede:
  - Zusatzkonzepte wie Schlüssel und Assoziationen werden über Annotationen im Quellcode spezifiziert.
  - Die Abbildung der Klassen auf relationale Tabellen muss festgelegt werden (typischerweise ebenfalls über Quellcode-Annotationen).
  - Als Query-Sprache kommt EJB-QL zum Einsatz.
  - Die Schnittstelle ist im Wesentlichen durch vier Interfaces gegeben
    - EntityManagerFactory: Erzeugung von EntityManagern
    - EntityManager: verwaltet eine Menge zusammengehöriger transaktionaler Objekte, ist Factory für Queries und bietet Zugriff auf Transaktionen
    - EntityTransaction: Transaktionsobjekt
    - Query: repräsentiert EJBQL- oder SQL-Queries
  - EJB3 ist nur für Java definiert – im Gegensatz zu ODMG gibt es keine Abbildung auf andere Sprachen.
- Standard fertiggestellt … aber es gibt weiterhin viele andere Ansätze.

Abbildungsspezifikation über Annotationen in EJB3

```
@Entity @Table(name="TBL_ROOM")
public class Room implements Serializable {
    Long id;
    @Id
    public Long getId() { return id; }
    public setId(Long id) { this.id = id; }

    List<Reservation> reservations;
    @OneToMany(mappedBy="room")
    public List<Reservation> getReservations() { return reservations; }
}

@Entity
public class Reservation implements Serializable {
    Room room;
    @ManyToOne
    public Room getRoom() { return room; }
}
```

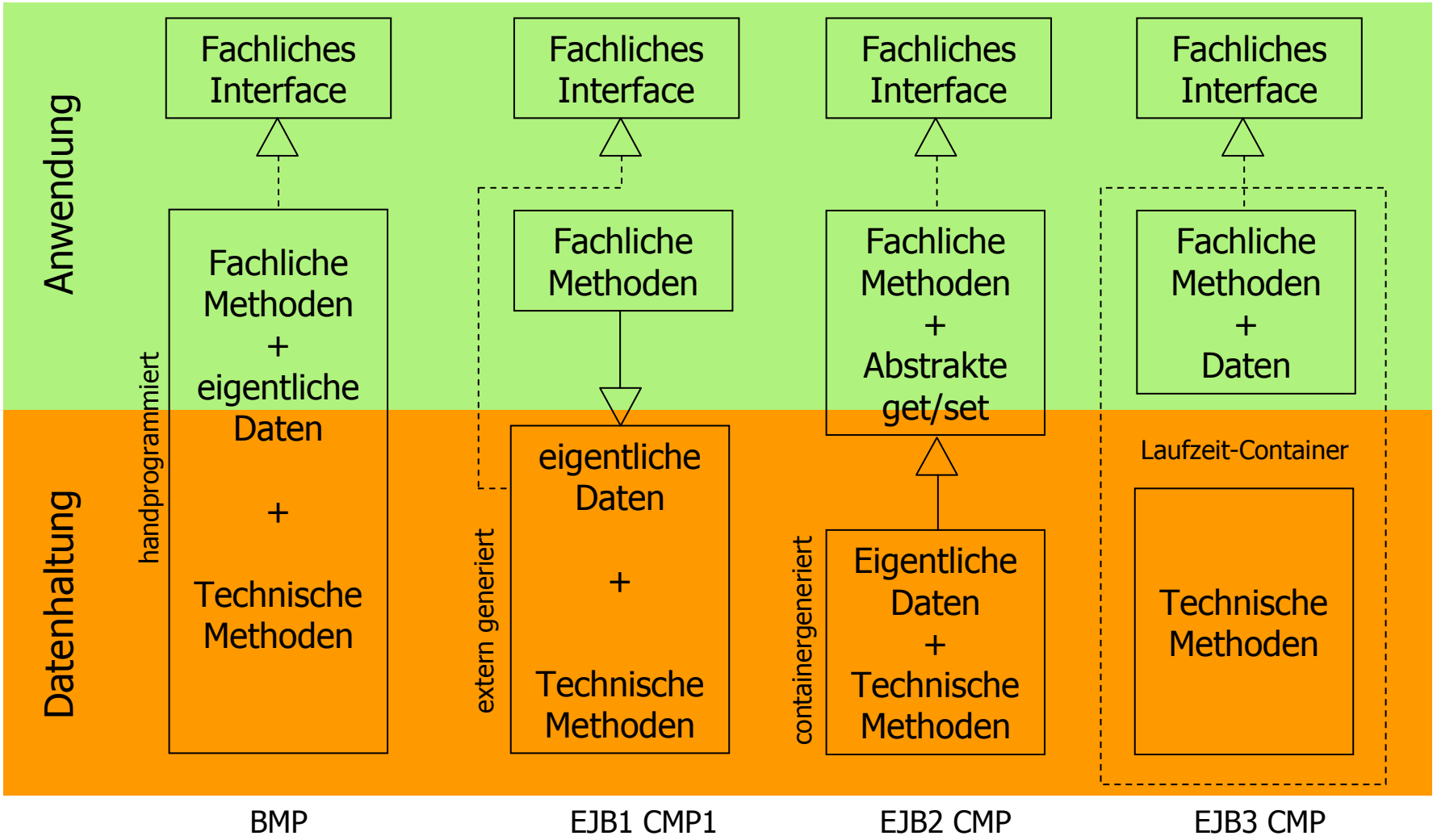
Schlüssel als Zusatzkonzept

Bidirektionale Assoziationen

Abbildung Klassenmodell auf relationales Modell

- Klassen werden zu Tabellen
- Attribute werden zu Spalten
- Primitive Datentypen und Mengentypen
  - Datentypen und Mengentypen der Programmiersprache werden abgebildet auf Datentypen und Mengentypen des DBMS
- Assoziationen werden zu
  - Fremdschlüssel-Attributen (bei ...-zu-1-Assoziationen)
  - Zwischentabellen (bei ...-zu-m-Assoziationen)
- Vererbungsbeziehungen werden aufgelöst zu:
  - Jede Klasse ist aufgeteilt auf mehrere Tabellen entsprechend der Vererbungsstufe, Objekt-ID bezeichnet zusammengehörige Teile
    - Holen eines eines Objekts erfordert Zugriff auf mehrere Tabellen
  - Für jede Klasse eine eigene Tabelle
    - Platzsparend, aber polymorphe Queries erfordern Zugriff auf mehrere Tabellen
  - Alle Klassen eines Vererbungszweiges in einer Tabelle, zusätzliche Typ-Spalte zur Kennzeichnung der Klasse
    - Schnelle polymorphe Queries, aber ggf. Platzvergeudung

Varianten zur Integration der Geschäftsobjekte



Wann kein objektorientierter Datenzugriff?

- Reporting-Anwendungen
  - Datenzugriff ausschließlich lesend über komplexe Queries
  - Keine Ausführung komplexer Anwendungslogik
  - Ergebnis meist in Form von tabellarischen Daten
  - → Relationale Datenbanken und SQL alleine ideal geeignet
- Batch-Läufe
  - Massendatenverarbeitung ohne Benutzerinteraktion
  - Durchführung gleichartiger, einfacher Operationen an vielen Objekten
    - Keine Ausführung komplexer Anwendungslogik
    - Jedes Objekt wird typischerweise nur einmal angefasst
  - → Transaktions- und Objekt-Caches sind eher hinderlich – eventuell sogar spezielle Vorkehrungen nötig, um sie klein zu halten
    - → Transaktionsklammern jeweils nach X Objektbearbeitungen setzen
    - → Objektcache von Zeit zu Zeit programmatisch leeren
  - → Abwägung zwischen Verwendung von OR-Mapper und traditioneller Programmierung mit Hilfe von SQL bzw. Stored Procedures

Inhalt

- Betriebliche Informationssysteme
  - Definitionen
  - Charakteristika
- Fachliche Architektur
- Technische Architektur
  - Basismechanismen und Infrastruktur
  - Schichtenarchitekturen und Verteilung
  - Transaktionsverarbeitung
- Datenhaltungsschicht
  - Aufgaben und Charakteristika
  - Realisierungsvarianten
- Zusammenfassung
- Literaturhinweise

Zusammenfassung

- Betriebliche Informationssysteme sind von zentraler Bedeutung für die Geschäftsprozesse großer Unternehmen.
- Die fachliche Architektur besteht typischerweise aus einem fachlichen Objektmodell (für Daten und Funktionalität) sowie einem hierarchischen Prozessmodell (für Steuerung und Präsentation).
- Basisarchitektur ist die Schichtenarchitektur.
  - Heutige Systeme haben meist drei logische Schichten (Datenhaltung, Anwendung, Präsentation) und ggf. eine Steuerungsschicht.
  - Die logische Schichtenarchitektur kann sich von der physischen Verteilungsarchitektur unterscheiden.
- Nebenläufigkeit und Mehrbenutzerfähigkeit werden im Wesentlichen über Transaktionen realisiert.
- Moderne Systeme bieten einen objektorientierten Datenzugriff, der beispielsweise über eine objekt-relationale Zugriffsschicht realisiert sein kann.

Literaturhinweise

- [Aut98] Klaus Bergner, Karsten Kuhla, Andreas Rausch: *Schnelle Schichten – Transparenter Zugriff auf ODBMS über CORBA,* iX 11/1998.
- [BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal: *Pattern-Oriented Software Architecture , Volume 1: A System of Patterns*. John Wiley & Sons 1996.
- [CBB+00] R.G.G. Cattell, Douglas Barry, Mark Berler et al.: *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann Publishers, 2000.
- [EJB3] Sun Microsystems: *J2EE Website*. http://java.sun.com/j2ee, 2005.
- [Hib] JBoss Inc.: *Hibernate Website*. www.hibernate.org, 2005.
- [Roo02] Robin Roos: *Java Data Objects*, Addison-Wesley, 2002.
- [Sie02] J. Siegel, *An Overview Of CORBA 3.0*, Object Management Group, 2002.