

---

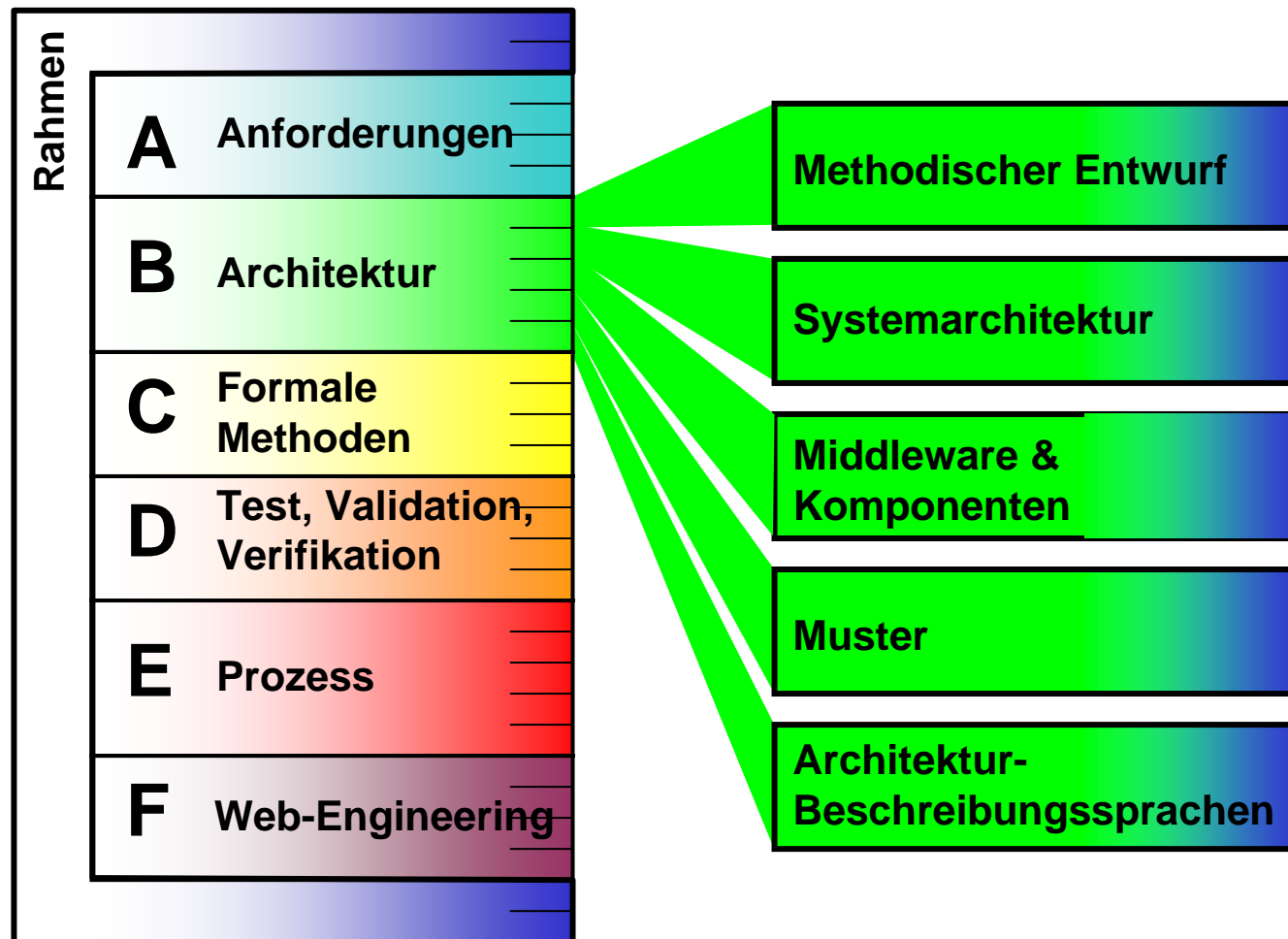
# Methoden des Software-Engineering

## Software Architektur

**Martin Wirsing, Harald Störrle**

**Block B, Teil 3: Middleware  
WS 2006/07, LMU München**

# Einbettung von B.3: Middleware & Komponenten

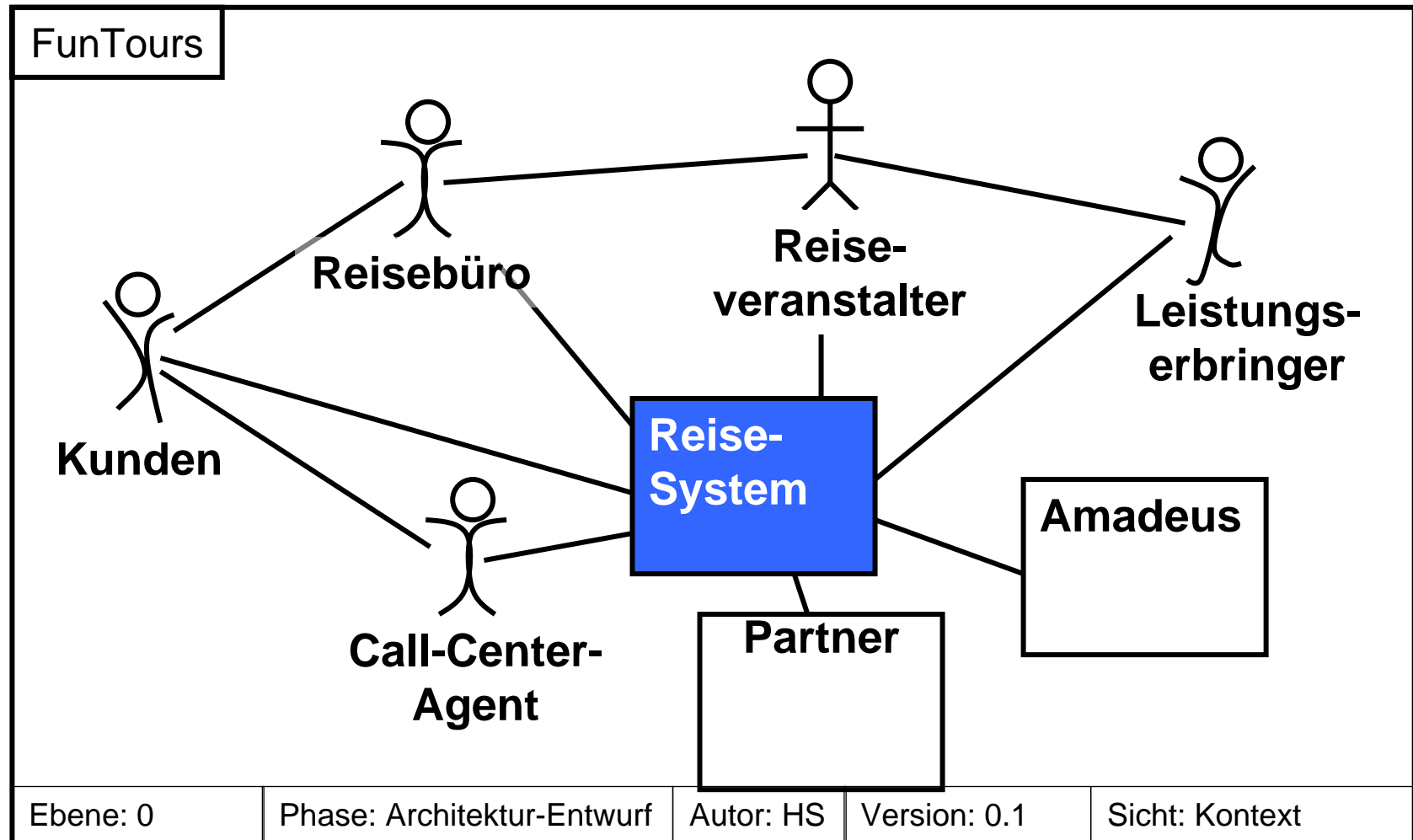


# Kernpunkte heute

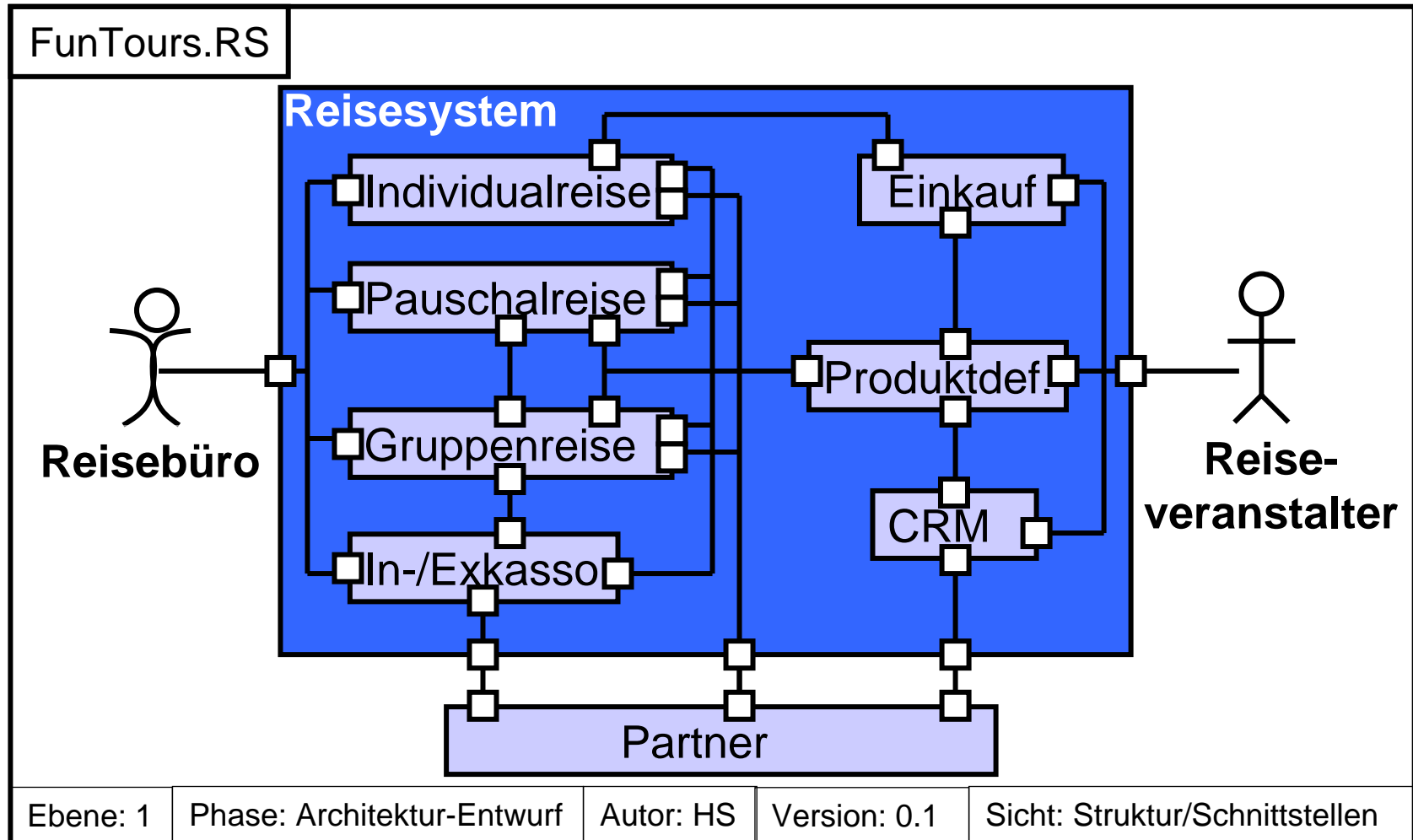
---

- **Middleware ist die „Klebe“ zwischen Komponenten.**
  - Man spricht von „architectural glue“, d.h. die Infrastruktur für/zwischen Komponenten.
- **Es gibt verschiedene Kategorien von Middleware, je nach Granularität und Art der Komponenten.**
  - Typische Beispiele sind Applikationsserver und Transaktionsmonitore, Corba, MQ-Series, RPC/RMI, und O2R-Mapping-Tools.
- **Beispiele für verschiedene Arten von Middleware.**

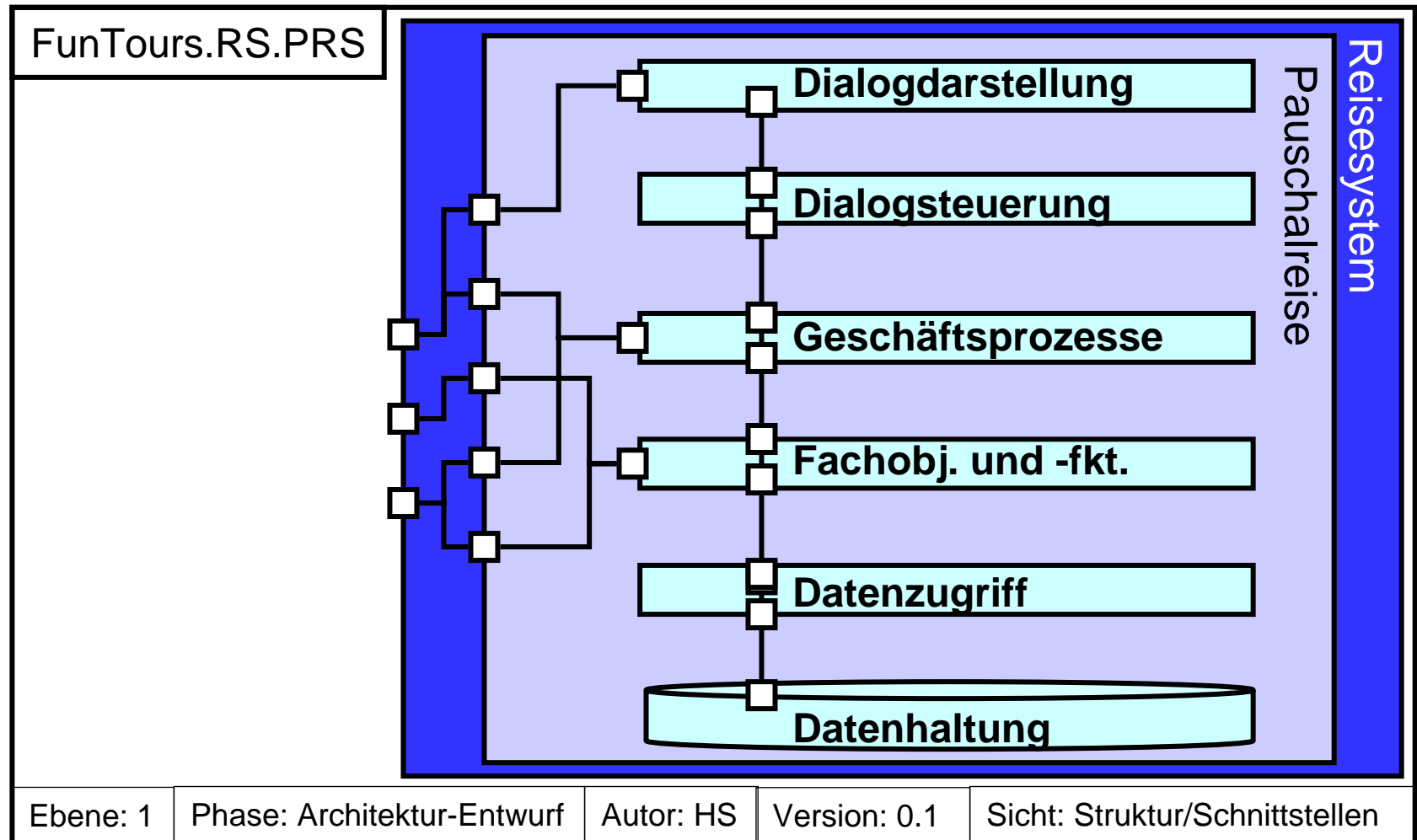
# Ebene 0: Systemlandschaft



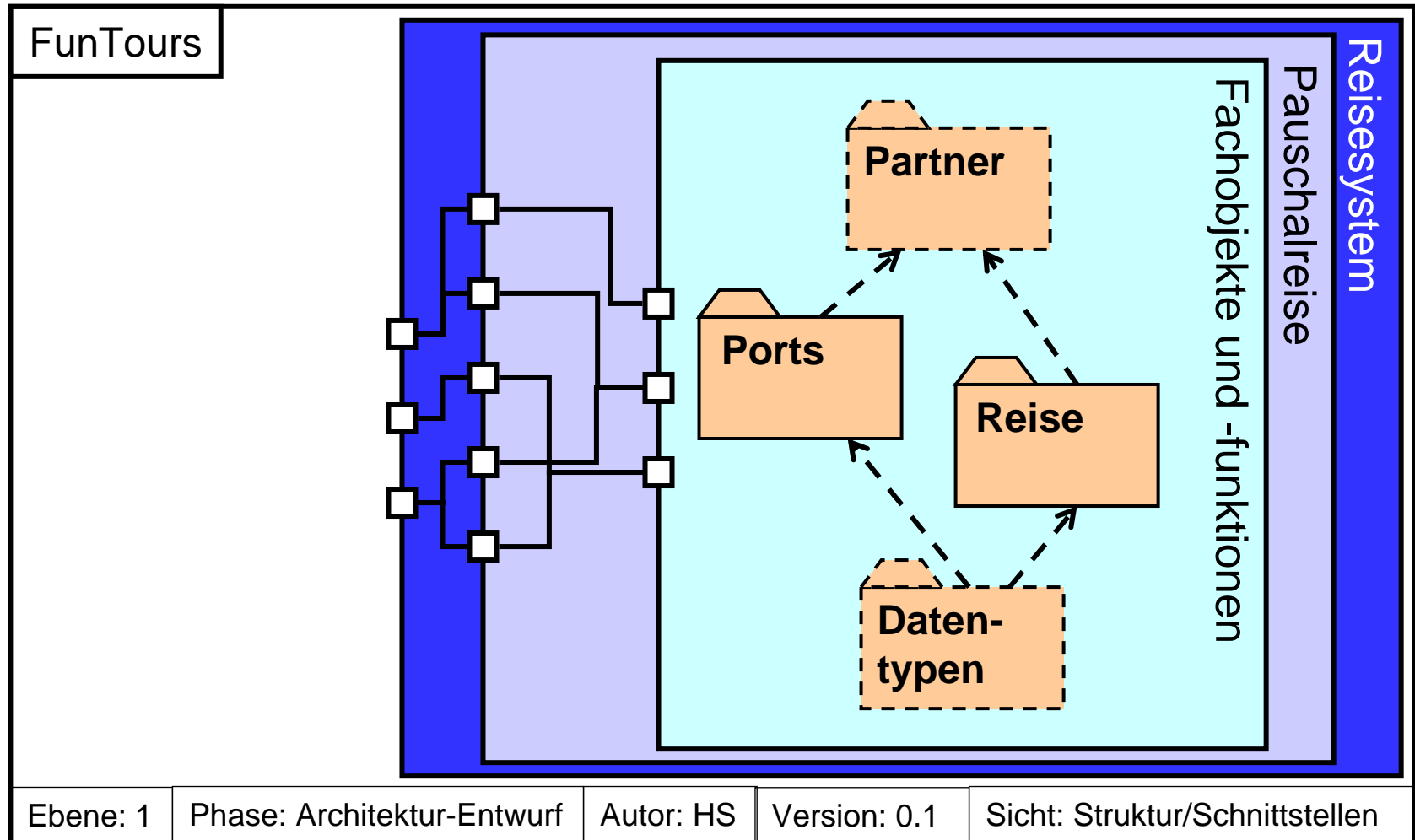
# Ebene 1: System



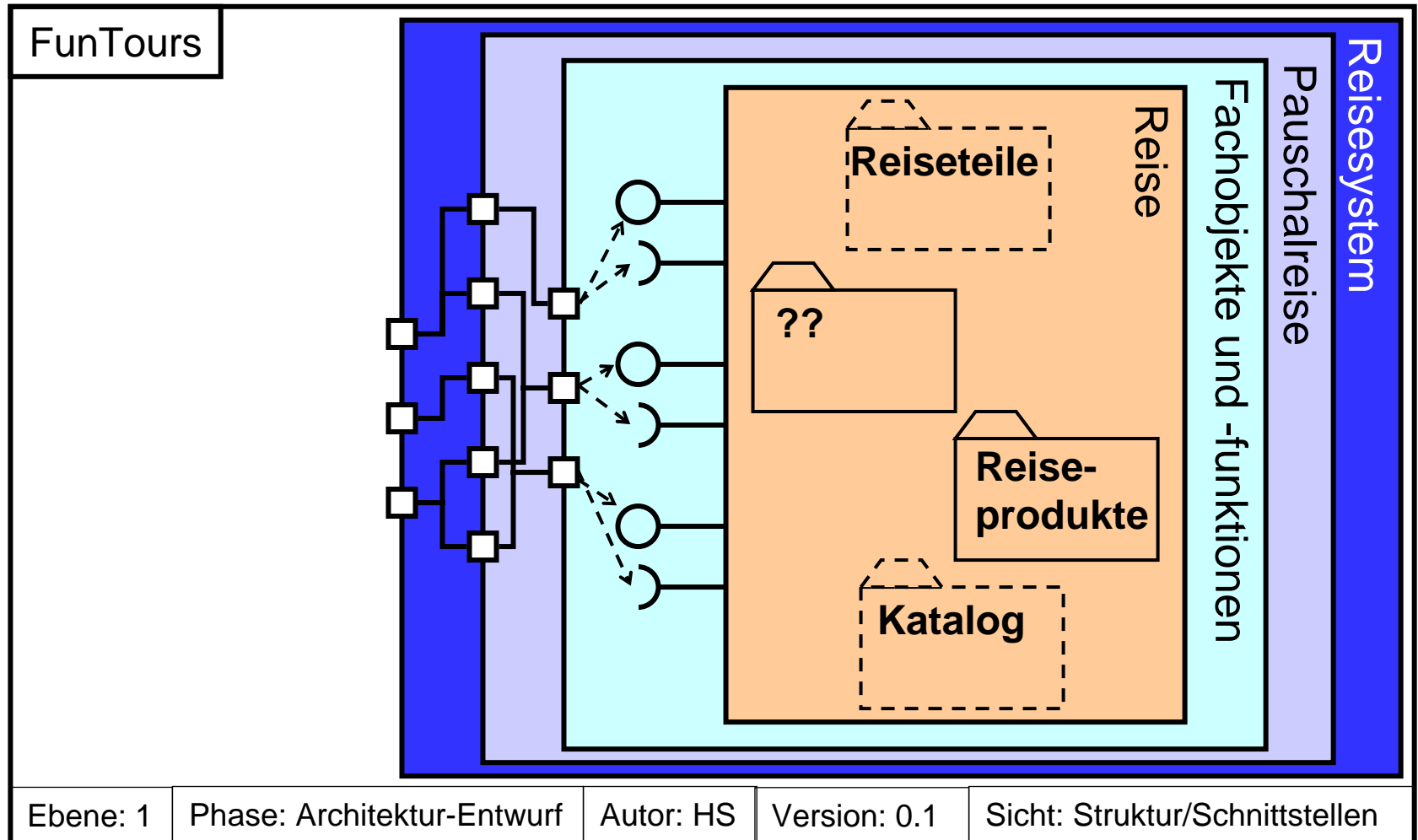
# Ebene 2: Subsystem



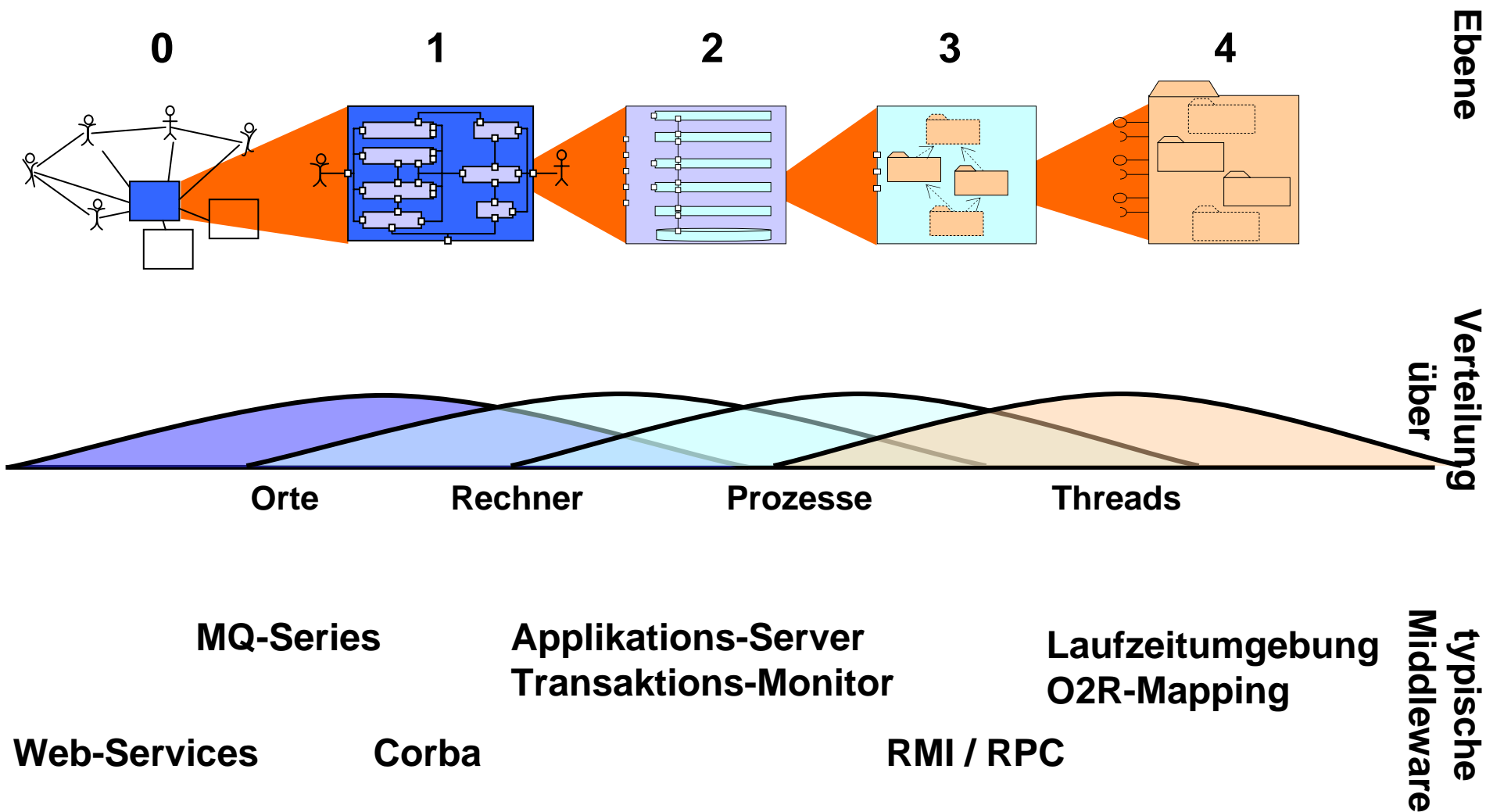
# Ebene 3: Schicht



# Ebene 4: Modul

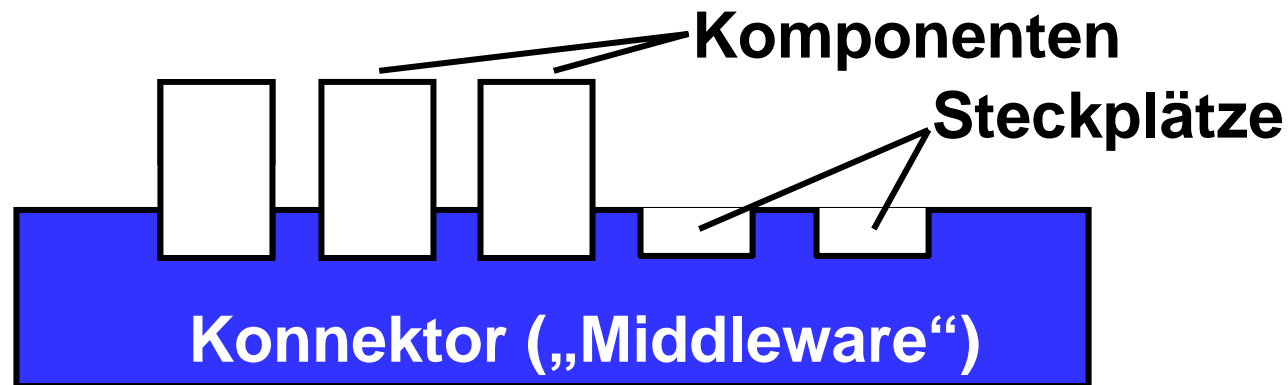


# Arten von Middleware je Ebene / Verteilungsraum



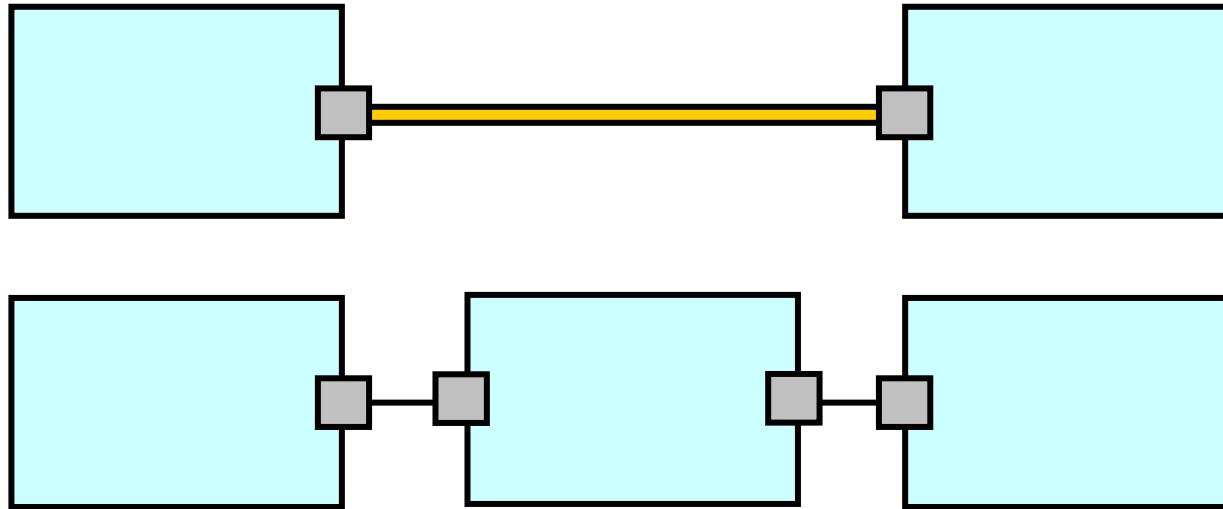
# Die Stecker-und-Steckdose-Metapher

- Die ursprüngliche Idee kommt in der Stecker-und-Steckdose-Metapher zum Ausdruck (McIlroy, 1968). Die Idee ist heute noch fesselnd, der Anspruch ist aber bis heute nicht wirklich eingelöst.



- Im Vordergrund standen zunächst die zu verbindenden Komponenten, das Verbinden und die Verbinder hielt man – in Anlehnung an das Vorbild Hardware – für trivial.
- Aber die Ports und Konnektoren können auch komplex sein. Dann ist die Middleware eher als eine Infrastruktur aufzufassen.

# Komplexe Konnektoren als Abkürzung



# Middleware-Definition

---

- Konzeptuell steht Middleware zwischen einer Plattform (Hardware und Software) und der Applikation, daher der Name.
- Als Middleware bezeichnet man im Wesentlichen alle Infrastruktur-Komponenten, Produkte, Standards, Werkzeuge und Technologien, die die Herstellung und den Betrieb verteilter Anwendungen in heterogenen Umgebungen erleichtern.

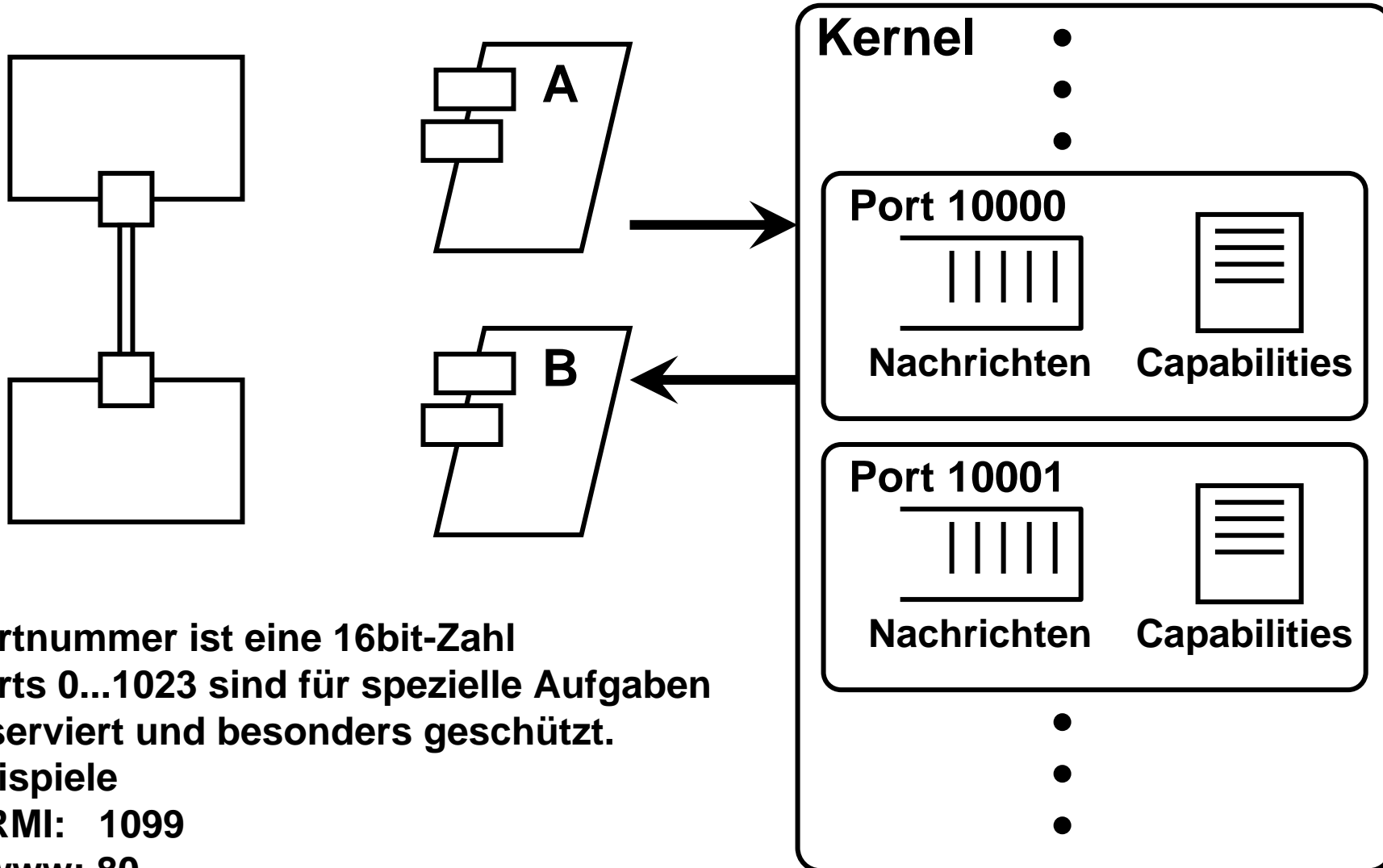
# Beispiele für Middleware

Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
<b>OS-Ports</b>	DB	O2R-mapping eSQL	CICS
Web-Services	MQSeries	Tuple-spaces in Linda	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# Interprozesskommunikation mit Ports



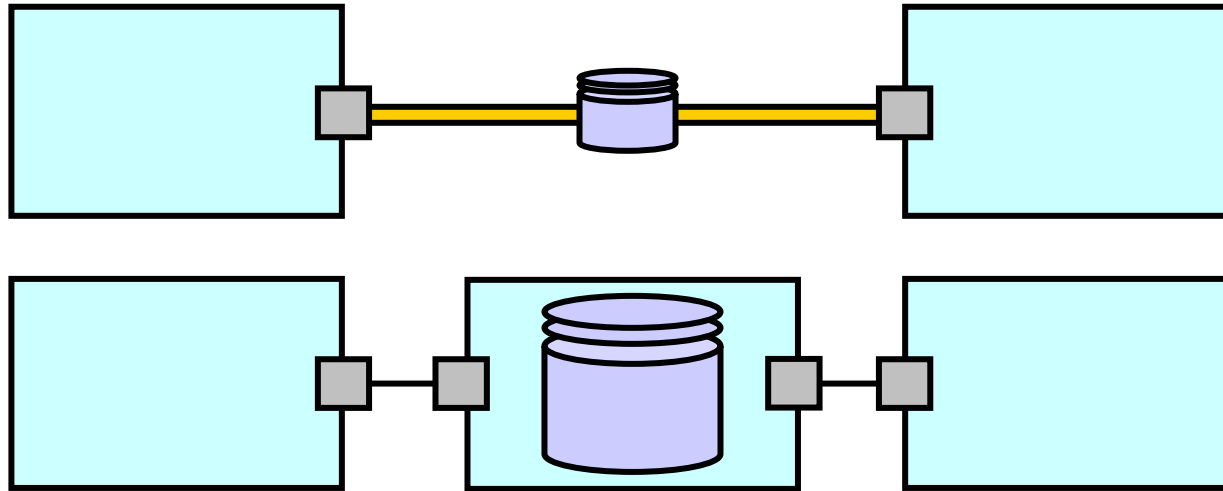
# Beispiele für Middleware

Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	<b>DB</b>	O2R-mapping eSQL	CICS
Web-Services	MQSeries	Tuple-spaces in Linda	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# DB/Datei als Puffer



**...oder auch in eine Datei oder ein ähnliches Konstrukt  
(z.B. in der Host-Welt).**

# Beispiele für Middleware

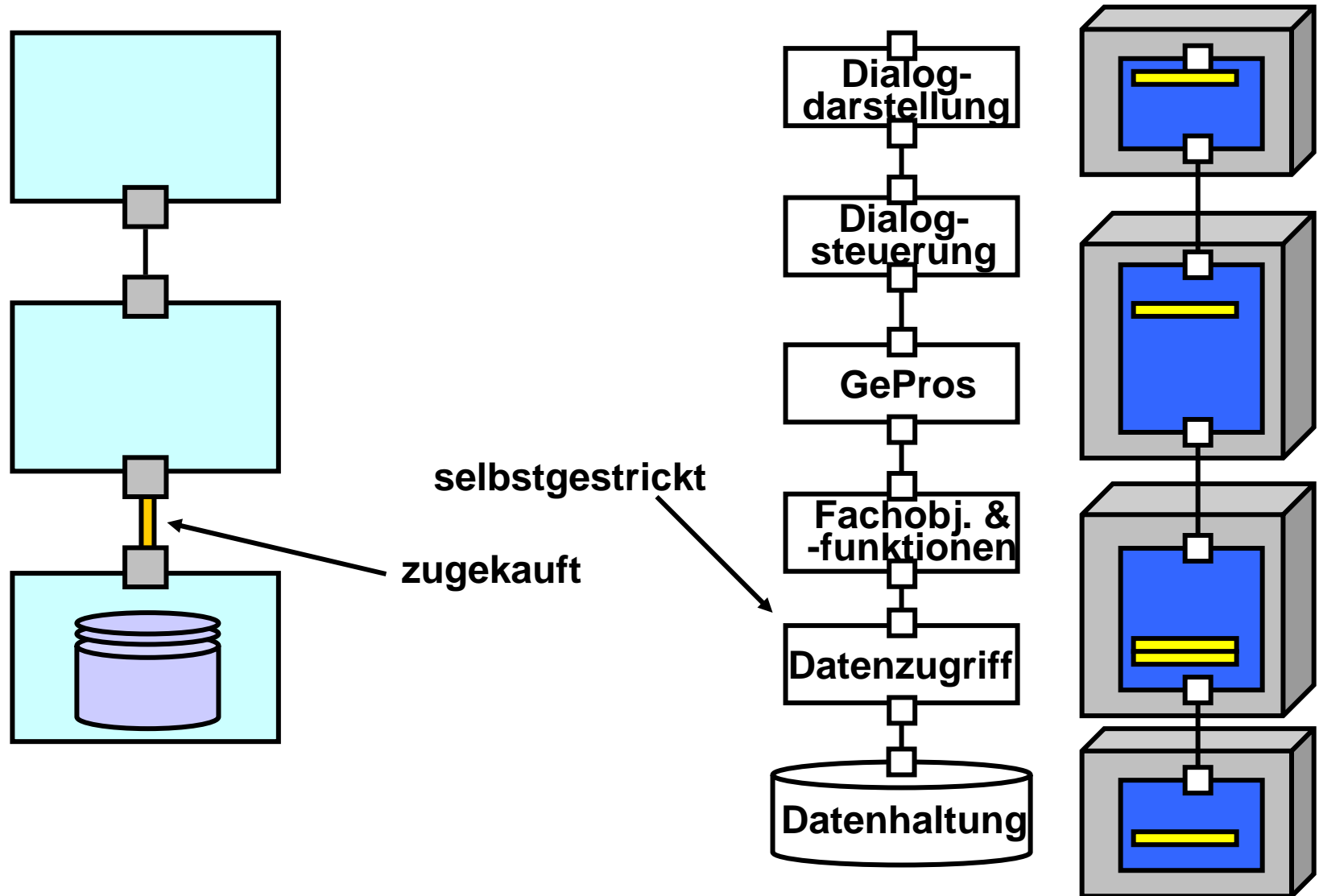
Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	DB	O2R-mapping eSQL	CICS
Web-Services	MQSeries	Tuple-spaces in Linda	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

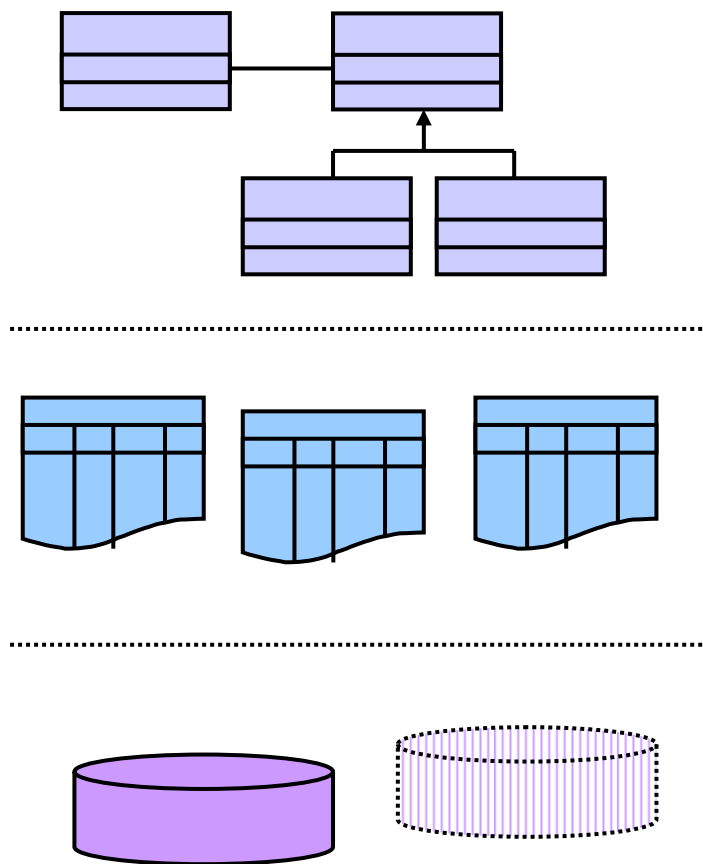
# Middleware in der Persistenzschicht

## O2R-Mapping



# Middleware in der Persistenzschicht

## O2R-Mapping



### Objektmodell

Konzepte und Beziehungen (im wesentlichen fachlich)



**O2R-Mapping, De-/Normalisierung**

### Relationenmodell

Relationen, Primär- und Fremdschlüssel,  
Attribute, Typen; Normalformen



**DB-Tuning**

### physisches Datenmodell

Tablespaces, Indizes, DB-Schema, Optimierungen

# O2R-Mapping

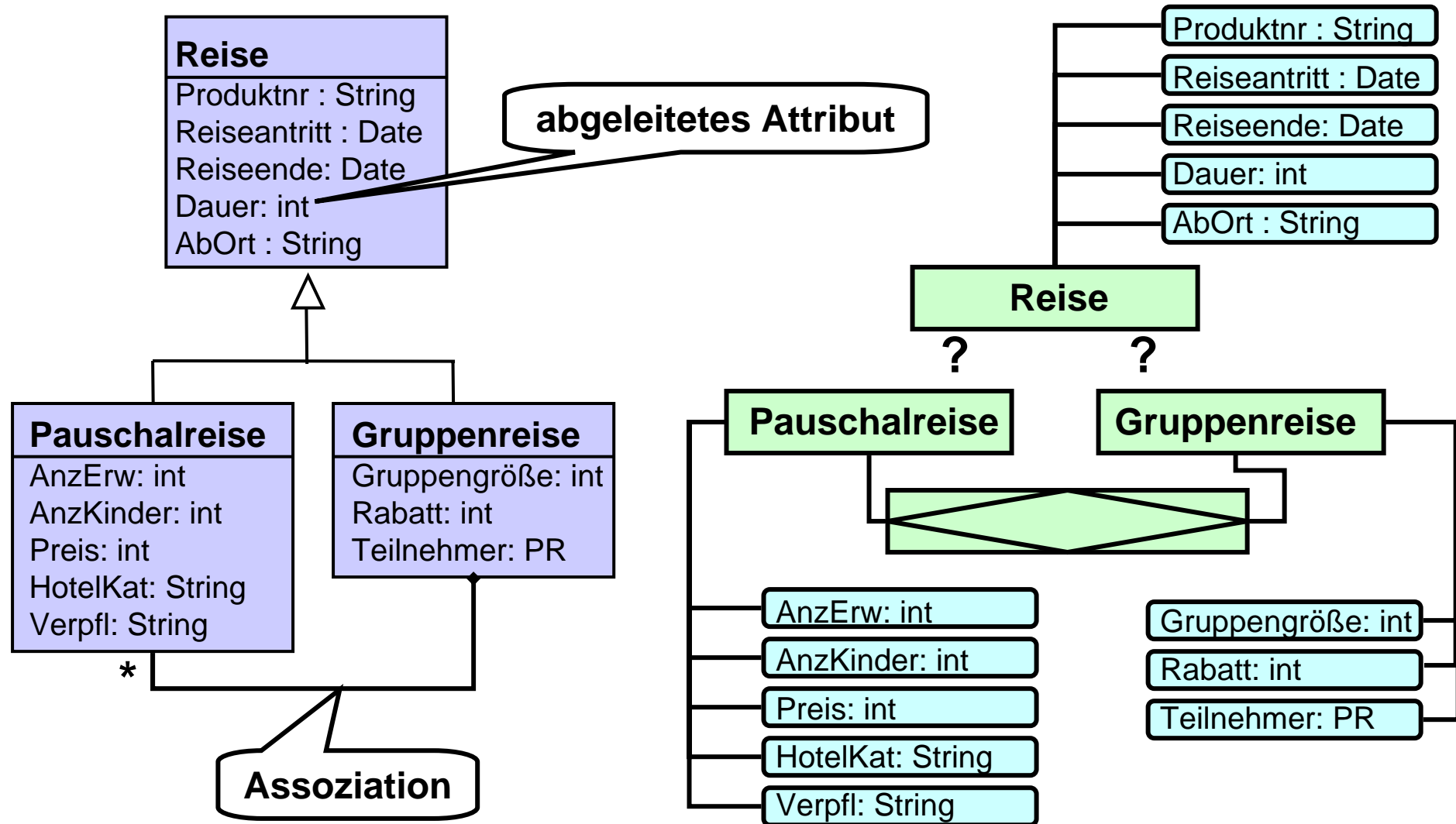
## Abbildung zwischen Objekt- und Relationenmodell

(auch: „O2R-Mapping“)

- Klasse : Tabelle
  - Typ : Domäne
  - Objekt : Zeile
  - Attribut : Spalte
  - Assoziation : Fremdschlüsselbeziehung
- 
- Vererbung wird als Delegation realisiert, d.h. über eine Assoziation.
  - Dieses Schema wird aus Effizienzgründen aufgeweicht (entspricht De-Normalisierung in der klassischen Welt).

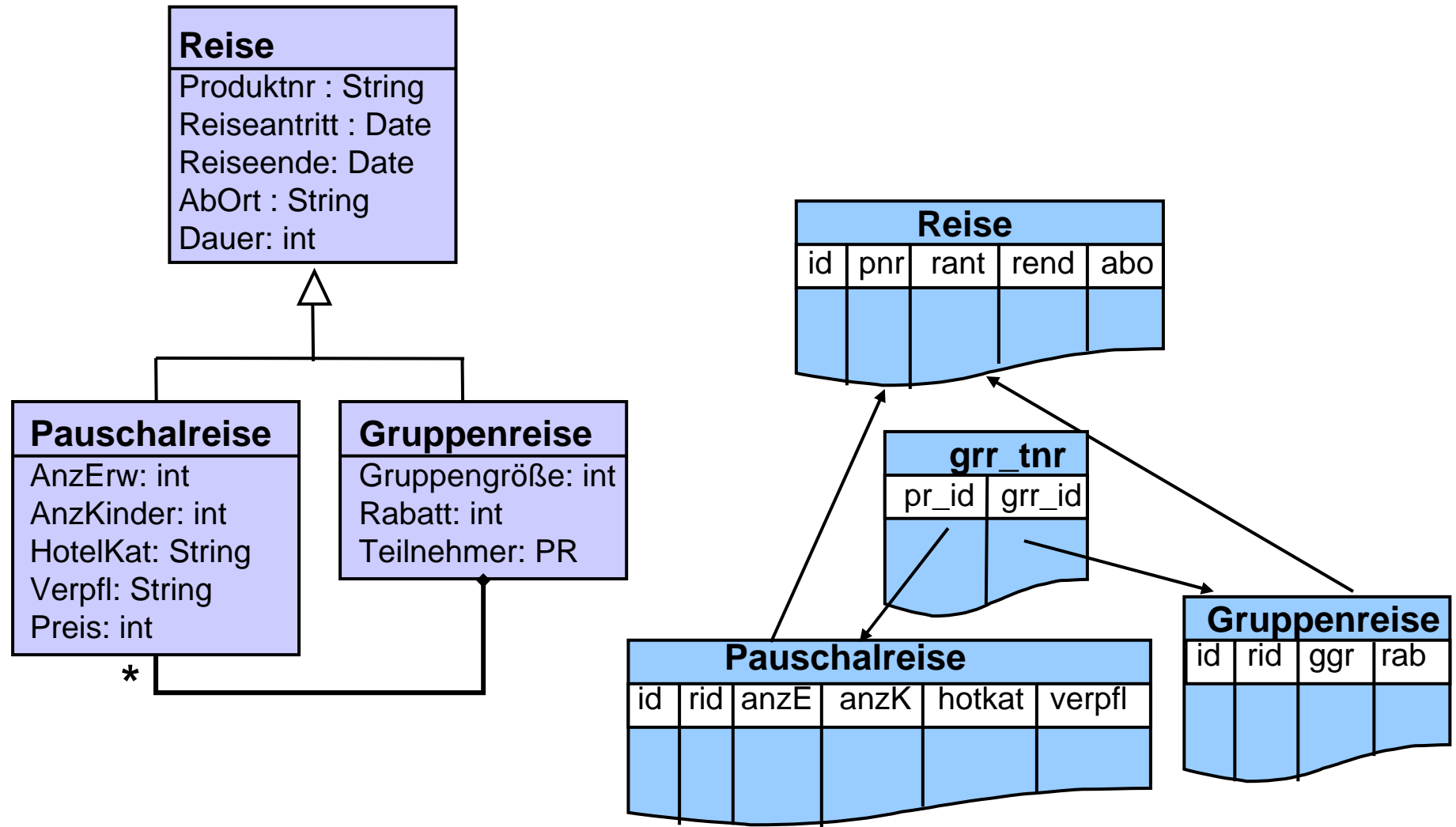
# O2R-Mapping

## Klassen- vs. ER-Diagramme



# O2R-Mapping

## Abbildung zwischen Objekt- und Relationenmodell



# Beispiele für Middleware

Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	DB	O2R-mapping eSQL	CICS
Web-Services	MQSeries	Tuple-spaces in Linda	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# Applikationsserver & Transaktionsmonitore

- **Applikationsserver (App.-Server) und Transaktionsmonitore (Txn.-Mon.)** verwalten Rechner-Ressourcen, z.B.:
  - Haupt- und Massenspeicher,
  - I/O- und Rechenleistung,
  - DB- und Kommunikationsverbindungen,
  - Transaktionskontrolle (ACID, Rollback/Recovery, usw.),um eine sehr hohe Zahl von Nutzern gleichzeitig zu bedienen.
- Sie bieten also ähnliche Funktionalität wie Betriebssysteme, aber viel leichtgewichtiger und mit zahlreichen Detailsinschränkungen, die eine effizientere Verarbeitung erlauben.
- Zu den Einschränkungen zählt insbesondere das „Programmiermodell“, also die Richtlinien und Schnittstellen, an die man sich halten muss, um diese Infrastruktur zu nutzen.

# Applikationsserver & Transaktionsmonitore

---

- Generell ist diese Art von System vor allem auf dem Host zu finden.
- Die beiden mit Abstand am weitesten verbreiteten Produkte sind
  - CICS (Customer Information and Control System) und
  - IMS (Integrated Management System).
- Auf Unix-Rechnern ist dieser Ansatz eher unüblich (→Tuxedo).
- Diese Lücke schließen seit einigen Jahren Applikationsserver wie IBM WebSphere und BEA WebLogic.
- Es gibt inzwischen auch konkurrenzfähige Open-Source-Produkte wie JBoss.

# CICS/IMS-Leistungskennzahlen

<b>Anzahl der Sachbearbeitern</b>	von einigen Hundert bis zu mehreren Tausend
<b>Tägliches Transaktionsvolumen</b>	Beispiele: <ul style="list-style-type: none"><li>• Inter Krankenversicherung: 300.000</li><li>• Volksfürsorge 2,5 Millionen</li></ul>
<b>Antwortzeitverhalten</b>	ca. 90 % der Transaktionen unter 1 Sek.
<b>Datenvolumen</b>	von einigen Hunderttausend bis zu mehreren Millionen aktive Verträge

# Beispiele für Middleware

Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	DB	O2R-mapping eSQL	CICS
Web-Services	MQSeries	Tuple-spaces in Linda	<b>CORBA</b>

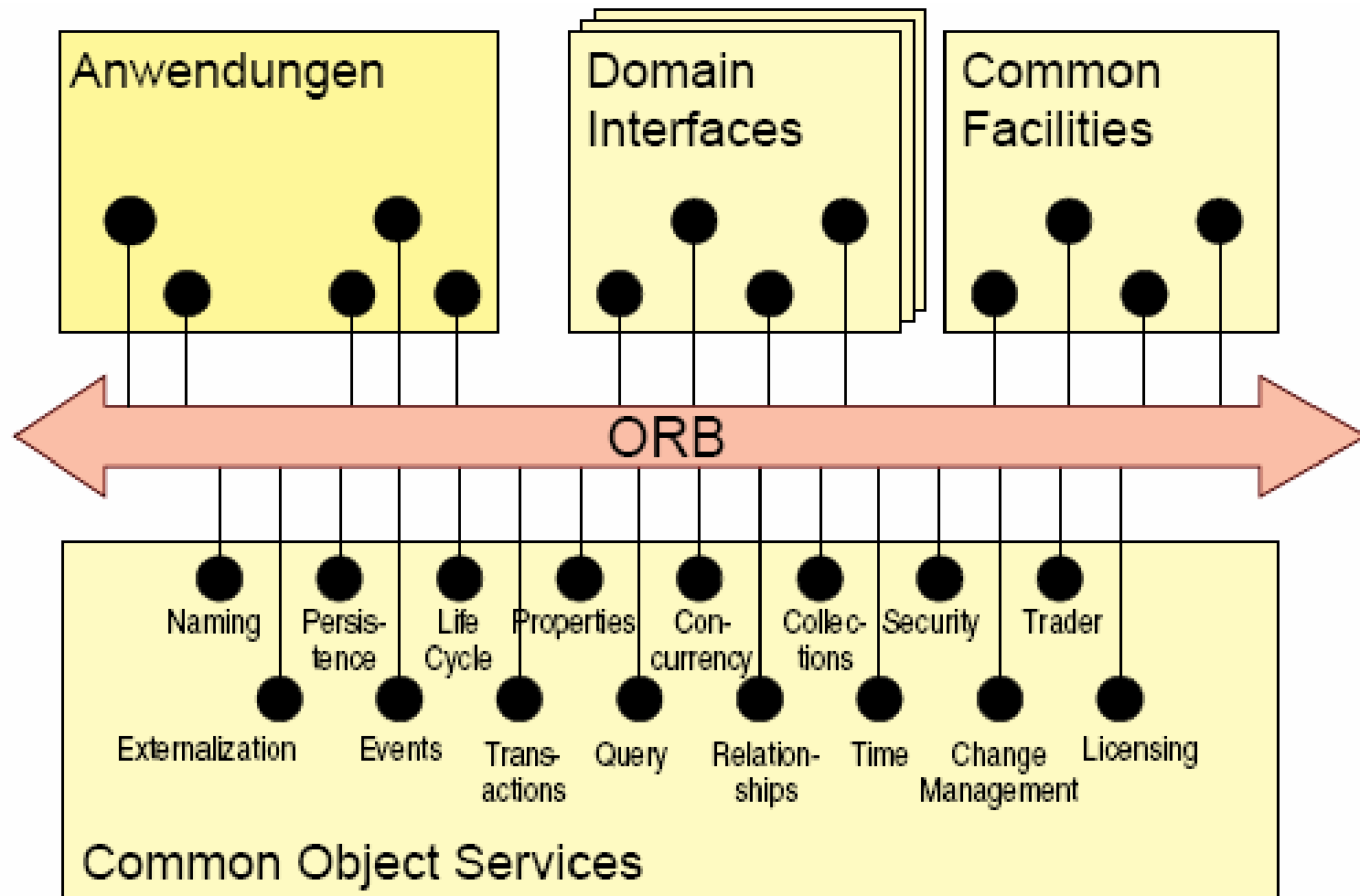
## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# CORBA

- CORBA steht für Common Object Request Broker Architecture und ist als Industriestandard von der OMG (Object Management Group) seit den frühen 90ern entwickelt worden.
- Ein Object Request Broker (ORB) ist eine Vermittlungseinheit für den Aufruf von verteilten Objekten.
- Corba arbeitet plattformübergreifend und programmiersprachen-unabhängig. Mittels einer Interface Definition Language (IDL) erstellt man eine formale Spezifikation der Klassen und Objekte sowie sämtlicher Parameter und Datentypen. Diese Schnittstellenbeschreibung wird dann in ein Objektmodell der verwendeten Programmiersprache umgesetzt, typischerweise für Java und C++, aber auch für viele weitere Sprachen.
- Es gibt inzwischen sehr viele qualitativ hochwertige kommerzielle und frei verfügbare Implementierungen von vielen verschiedenen Herstellern (Orbis, Orbacus, VisiBroker, ...).

# CORBA – Grundlegender Aufbau

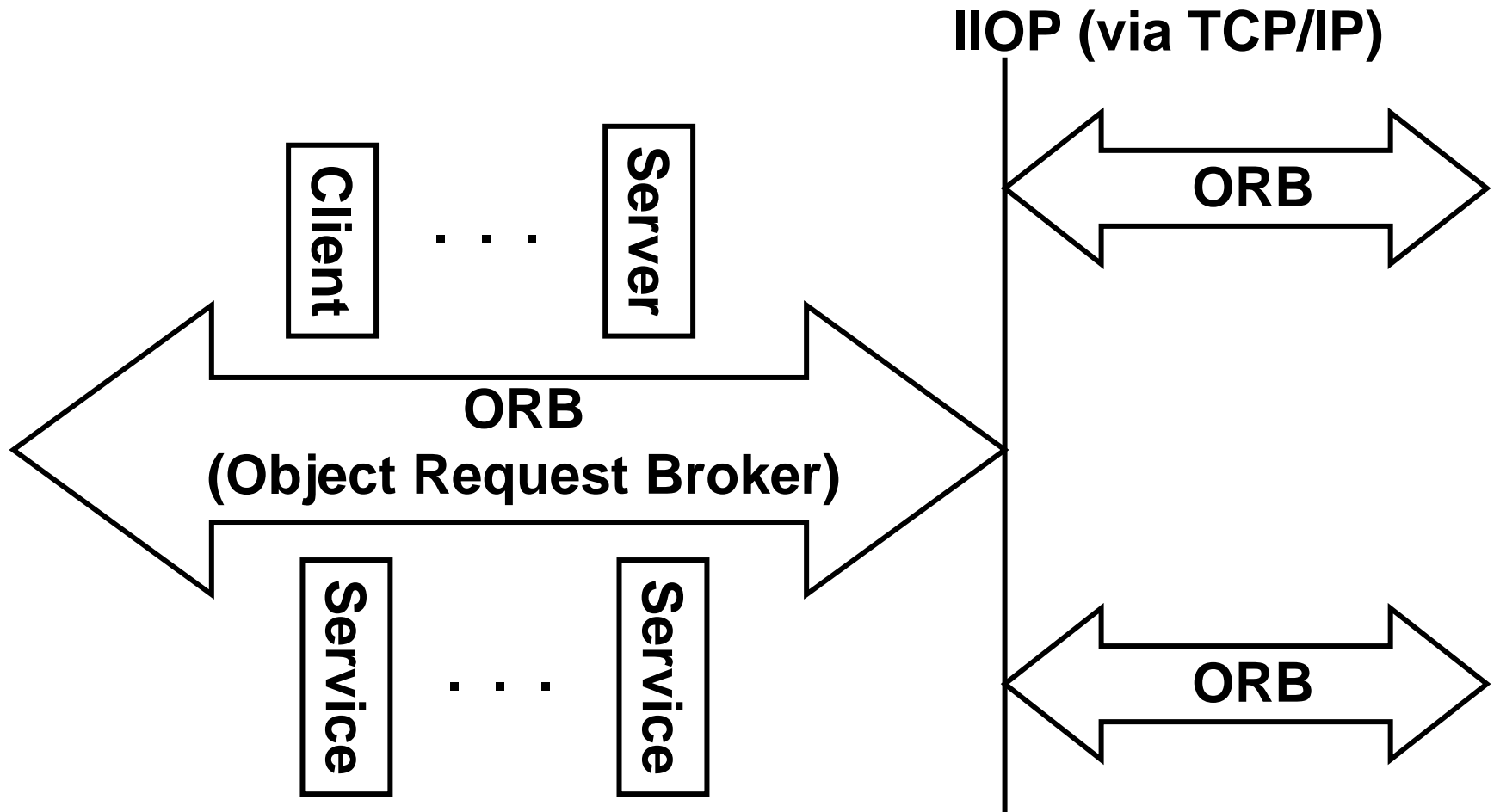


# CORBA – Grundlegender Aufbau

- Die **CORBA Architektur** besteht aus dem ORB (Object Request Broker). Dieser vermittelt Aufrufe zwischen Objekten. Der ORB weiß, wo welches Objekt liegt. Die Anwendungsobjekte können untereinander mit Hilfe des ORB kommunizieren und stehen untereinander in Client/Server-Beziehung.
- Die **Common Object Services** sind allgemeine, anwendungsunabhängige, standardisierte Systemdienste, die eine CORBA Implementierung anbieten kann bzw. muss, z.B. den Naming Service zum Auffinden von Objekten anhand eines Namens oder einen Transaktionsdienst zur Unterstützung von Transaktionen im verteilten System. CORBA-Services verhalten sich wie normale Objekte.
- **Common Facilities** sind ähnlich wie Object Services unabhängig von einem bestimmten Anwendungsbereich. Im Gegensatz zu Services legen sie aber nicht Schnittstellen für Systemdienste, sondern Schnittstellen zu Benutzeranwendungen (z.B. für Dokumentenbearbeitung oder Grafikschnittstellen) fest.
- **Domain Interfaces** legen ähnlich wie Facilities Anwendungsschnittstellen fest, die sich allerdings an bestimmten Anwendungsbereichen oder Branchen orientieren. Beispielsweise Product Data Management (PDM), Telekommunikation, Dienste des Gesundheitswesens oder Finanzanwendungen.

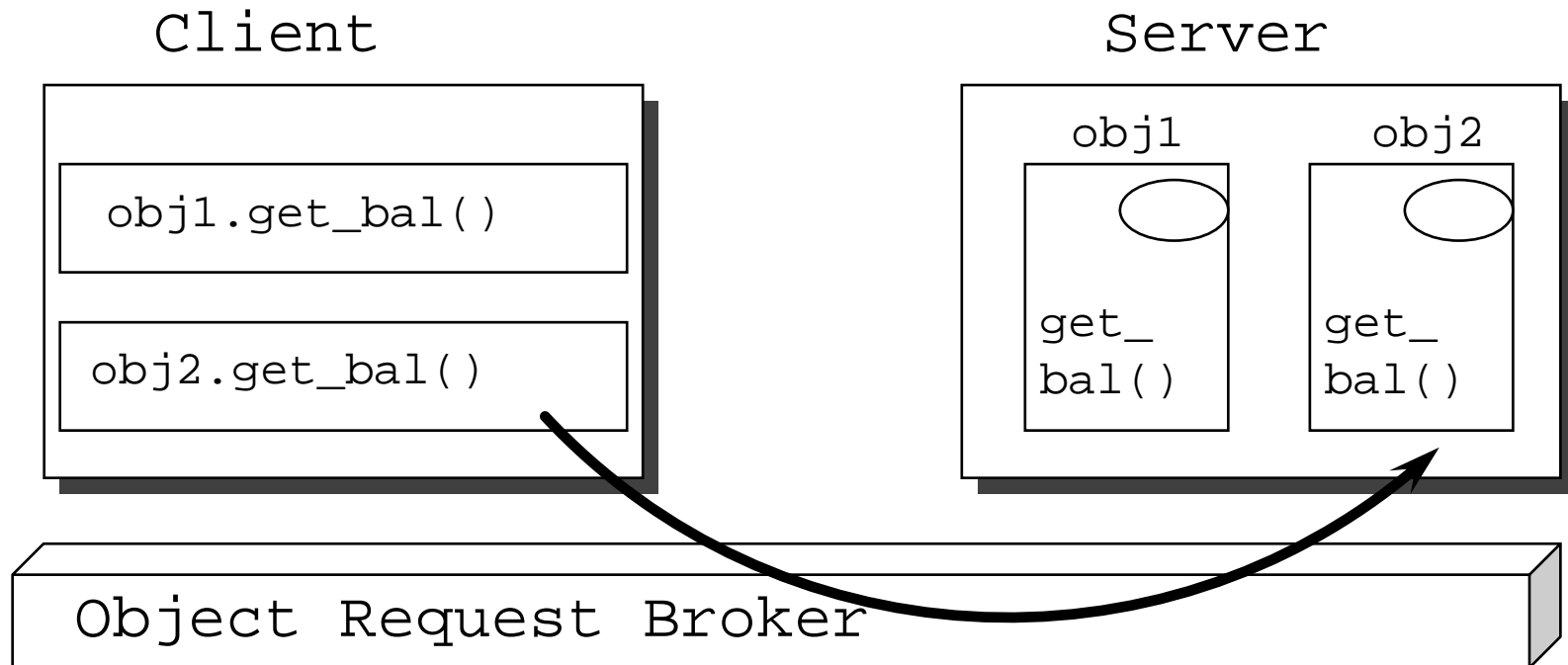
*(c) Franz J. Hauck, Univ. Ulm; Jürgen Kleinöder · Universität Erlangen-Nürnberg*

# CORBA - Kommunikation mit anderen ORBs



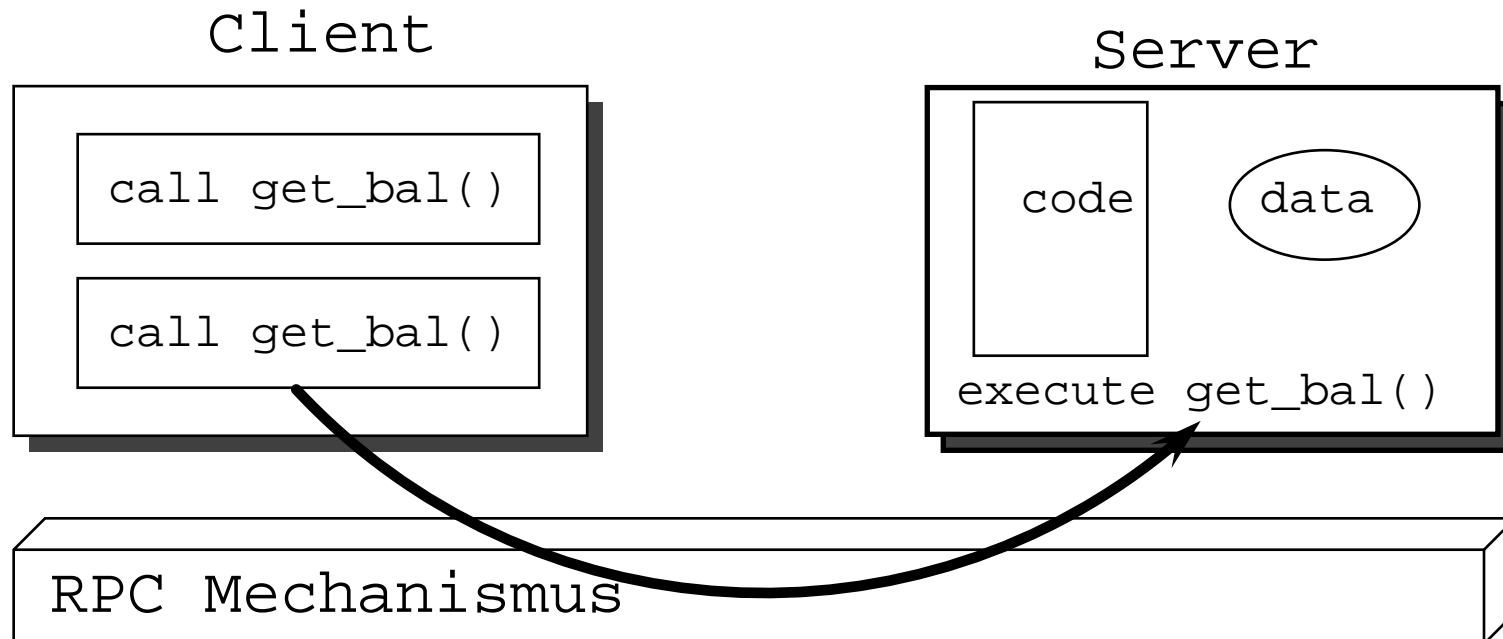
**IIOP (Internet Inter-ORB Protocol)** zur Kommunikation zwischen ORBs verschiedener Hersteller.

# CORBA-Request



- **synchron** blockiert bis zum Erhalt des Reply
- **deferred synchron** (asynchron) Client arbeitet nach Absenden des Requests weiter und erfragt später, ob Reply vorliegt (nur bei DII)
- **oneway** best-effort-Mitteilung, keine Antwort

# Im Vergleich: Remote Procedure Call



# Grundlegende Begriffe

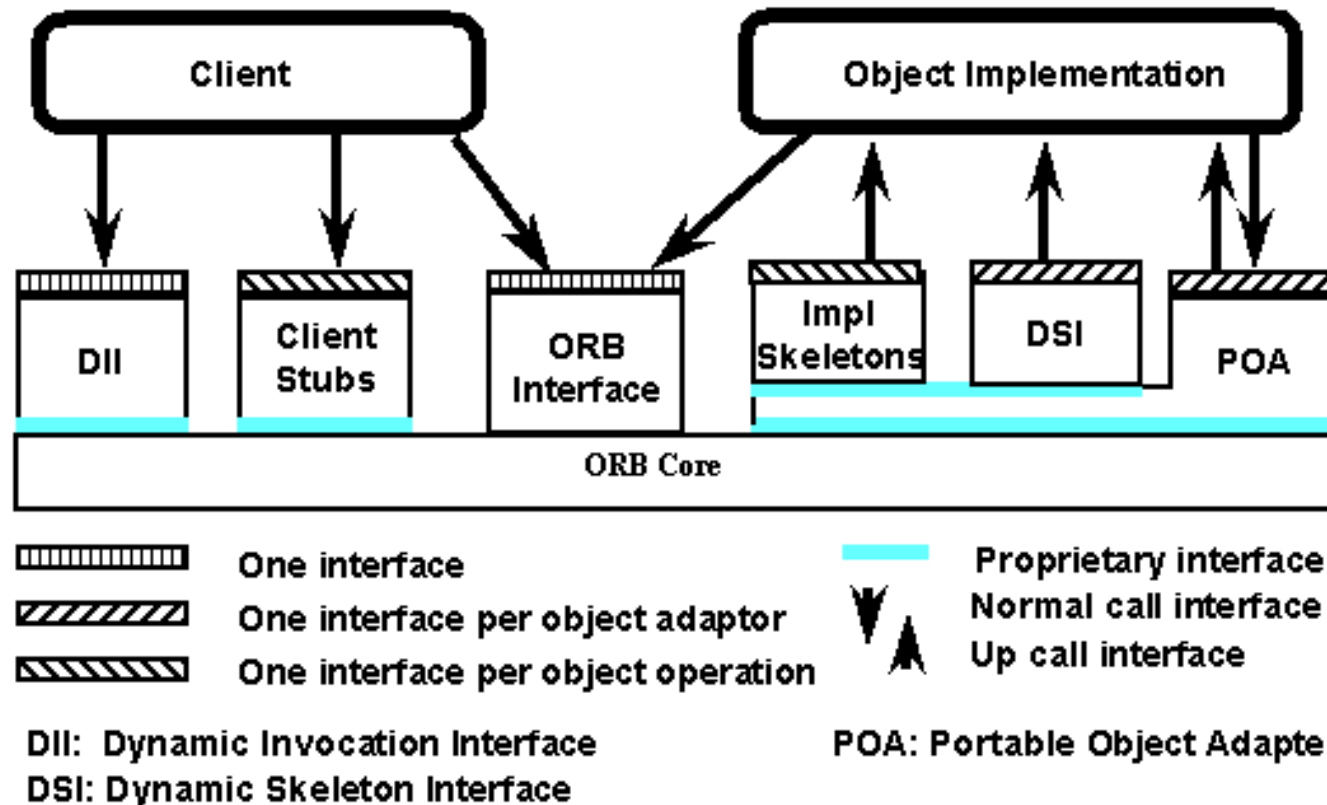
- **CORBA-Objekt**
  - Abstrakte Einheit, die vom ORB lokalisiert werden kann
  - Ziel von Operationsaufrufen
  - Besitzt ein Interface, eine Objektreferenz und kann mit einer Implementation (*Servant*) assoziiert werden.
- **Request**
  - Aufruf einer Operation an einem bestimmten Objekt und Rückgabe des Ergebnisses, falls vorhanden.
- **Client**
  - Einheit, die eine Operation eines Objekts aufruft.
- **Server**
  - Programm, in dem das gerufene Objekt lebt. Beachte: Die Begriffe Client und Server sind immer relativ zu einem bestimmten Request!
- **Servant**
  - Programmiersprachliche Einheit, die ein oder mehrere CORBA-Objekte implementiert.

# Interface-Definition mit OMG IDL

```
module Count {  
    interface Account {  
        readonly attribute double bal;  
        long get_bal ();  
        void deposit(in double value);  
        ...  
    };  
};
```

- **IDL** ist eine Sprache zur Beschreibung von Schnittstellen.
- Ein Objekt unterstützt ein Interface, wenn es zu allen seinen Operationen eine Implementation bietet.
- Aus der sprachunabhängigen IDL-Definition werden Stubs/Skeletons für versch. Implementationssprachen erzeugt.

# ORB – Grundlegender Aufbau



- Ein ORB vermittelt Methodenaufrufe von einem zum anderen Objekt
- innerhalb und zwischen Adressräumen und Prozessen eines ORBs
  - zwischen Adressräumen u. Prozessen verschiedener ORBs.

# ORB – Grundlegender Aufbau

Dynamic-Invocation-Interface (DII) bietet Methoden zur

- Bestimmung des Typs eines Objekts , der zur Entwicklungszeit einer Anwendung nicht bekannt ist, z. B. Name-Server-Implementierung, Verzeichnisdienst, ...
- Ausführung von Methodenaufrufen (als RPC, asynchroner RPC, über Datagramm-Kommunikation ohne Antwort)

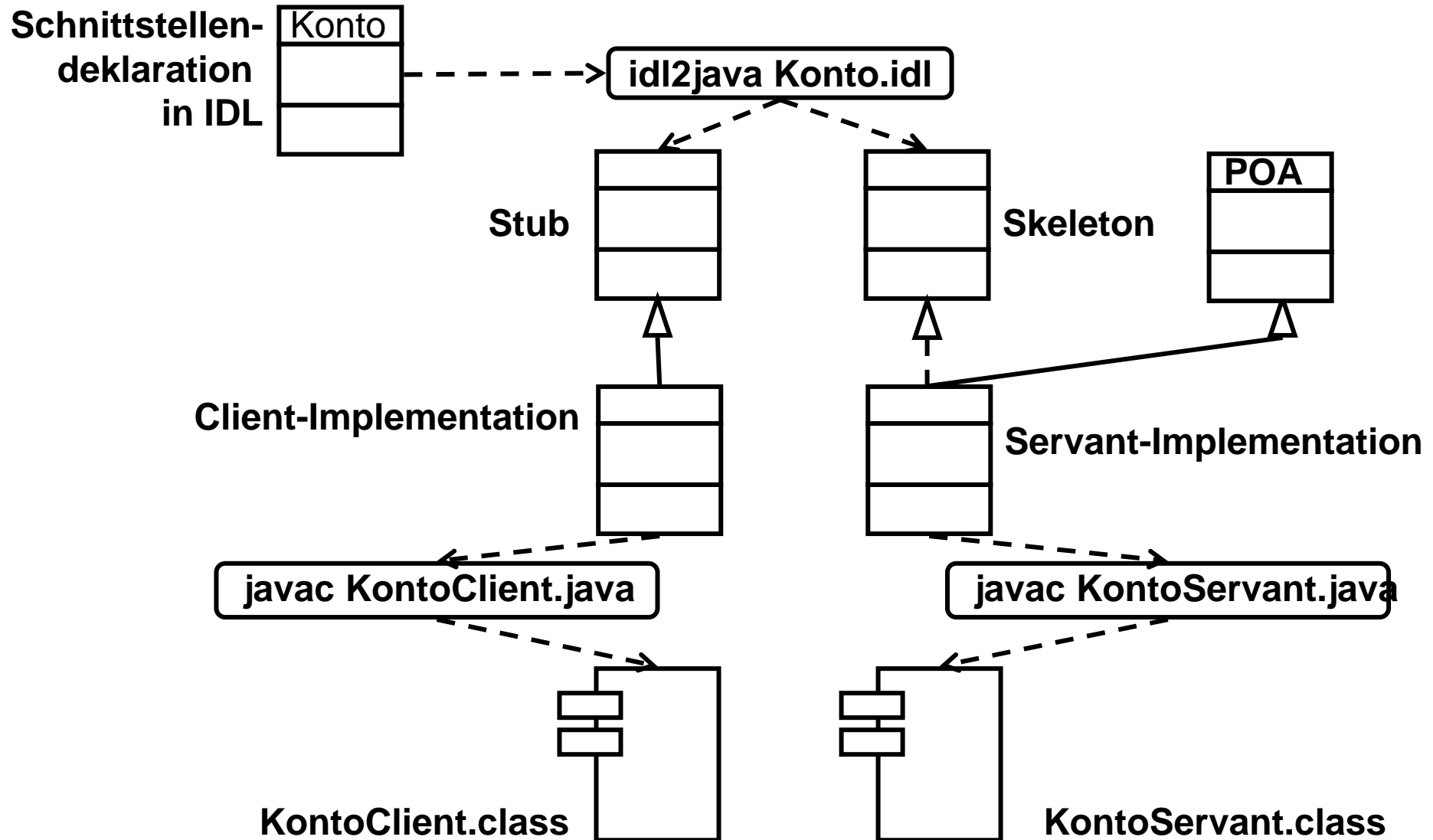
Dynamic Skeleton Interface (DSI)

- ermöglicht das Entgegennehmen von Methodenaufrufen für Objekte, deren Schnittstelle nur dynamisch ermittelbar ist.

Portable Object Adapter (POA)

- setzt Aufruf an einem *Object* in einen Aufruf an einen Servant um (d.h. in ein konkretes Sprachobjekt, das CORBA-Objekt(e) implementiert).

# CORBA – Vorgehensweise



# Beispiele für Middleware

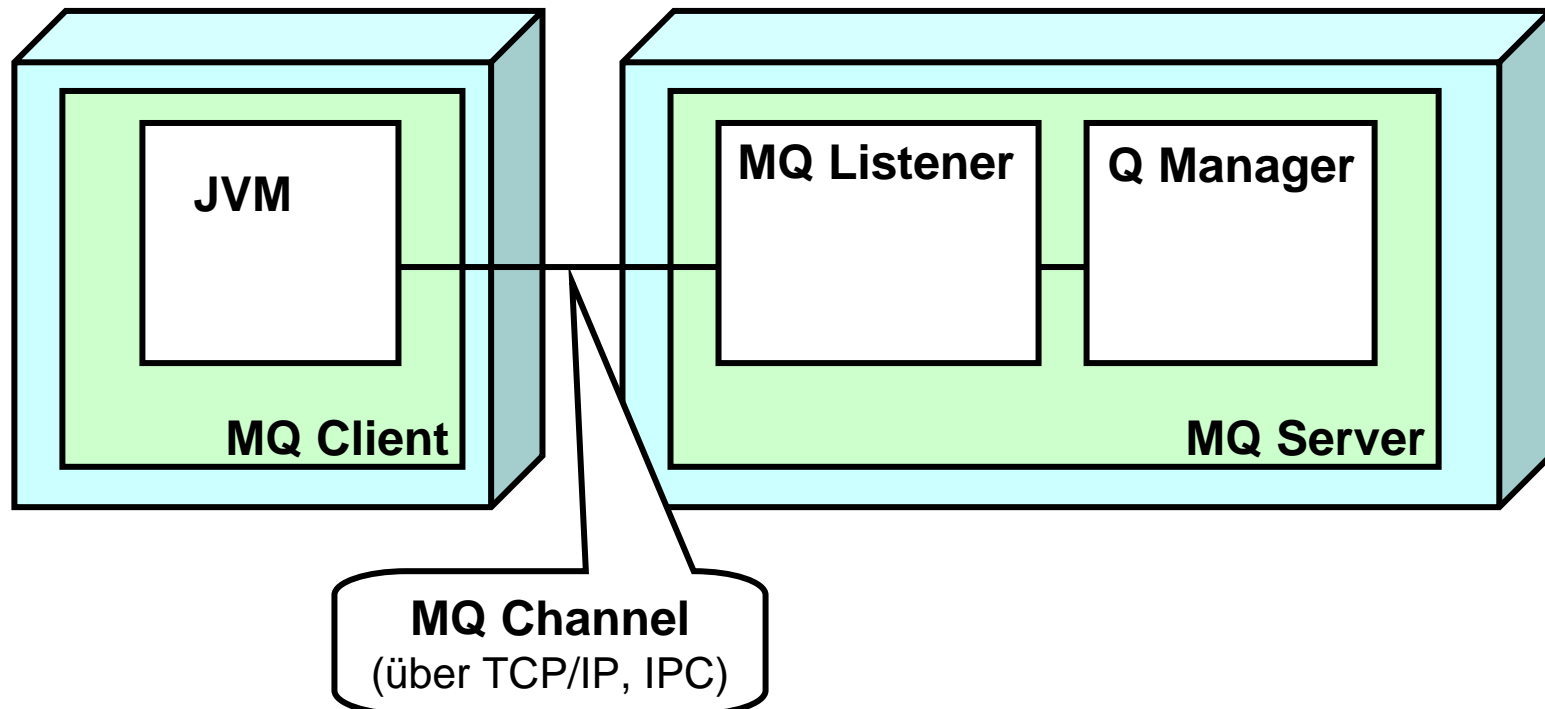
Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	DB	O2R-mapping eSQL	CICS
Web-Services	<b>MQSeries</b>	Tuple-spaces in Linda	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# MQSeries

- Paradigmatisches Beispiel für Message-Oriented Middleware
  - verbindet Anwendungen und Web-Services
  - Kommunikation über Nachrichten, die in Warteschlangen abgelegt und vom Empfänger asynchron abgeholt werden
  - 85 % Marktanteil, auf 80 Plattformen verfügbar
  - heißt neuerdings WebSphere MQ



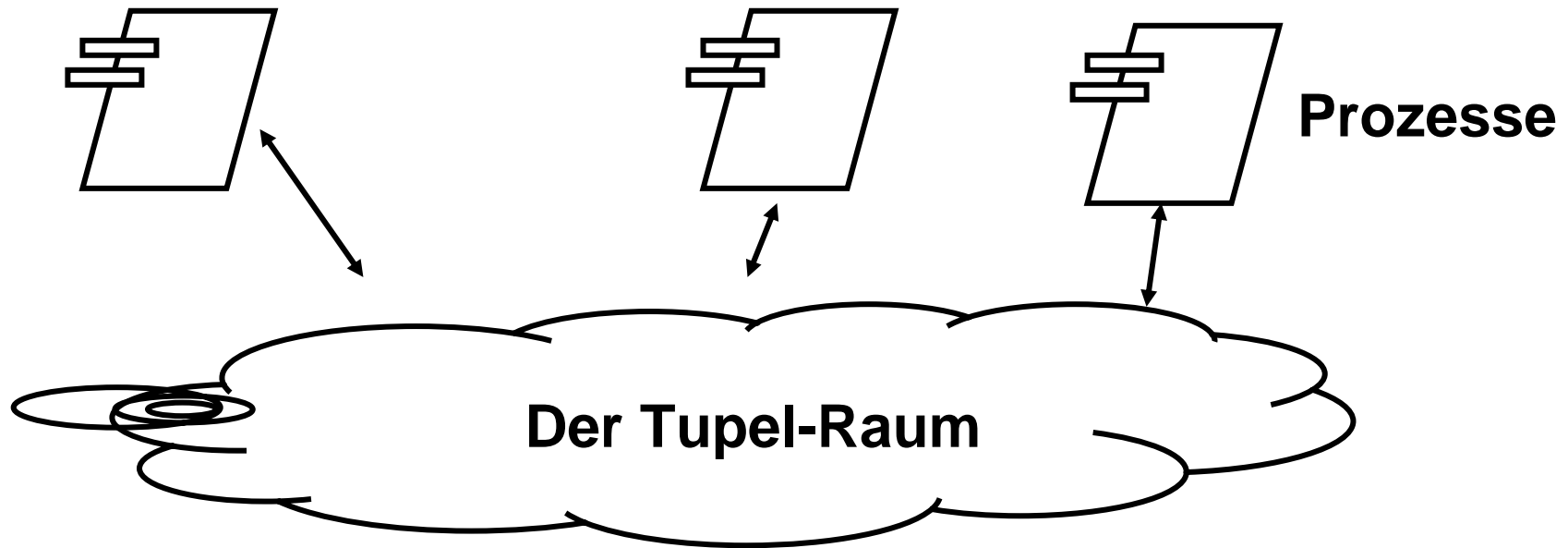
# Beispiele für Middleware

Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	DB	O2R-mapping eSQL	CICS
Web-Services	MQSeries	<b>Tuple-spaces in Linda</b>	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# Tuple-Spaces in Linda



- Linda ist ein abstraktes Konzept zur Beschreibung von Interaktionen.
- Linda selbst ist unabhängig von einer Programmiersprache.
- Es gibt Linda-Implementationen in allen möglichen Sprachen.

# Tupel und Felder

- **Tupel sind eine Folge von getypten Feldern.**
- **Beispiele:**
  - (`"arraydata", dim1, 13, 2`)
  - (`var1, var2`)
  - (`"common block", datacom`)
- **Die Elemente in den Feldern sind entweder Werte oder formale Parameter oder auswertbare Ausdrücke (`const, expr, var`).**
- **Der Tupelraum ist ein assoziativer Speicher, d.h. ein Tupel wird über seinen Inhalt adressiert.**
- **Das Tupel (`"arraydata", 4, 6, 8`) paßt auf folgende beide Templates**
  - (`"arraydata", ?dim1, ?dim2, ?dim3`);
  - (`"arraydata", 4, ?dim2, ?dim3`);

# Linda Operationen

## out

- erzeugt passives Tupel
- gibt Kontrolle sofort zurück

```
out('array data', dim1,  
dim2);
```

## eval

- erzeugt aktive Tupel („Prozess“),
- gibt Kontrolle sofort zurück.
- alle Felder werden nebenläufig ausgewertet und im TS abgelegt

```
eval("test", i, f(i));
```

## in

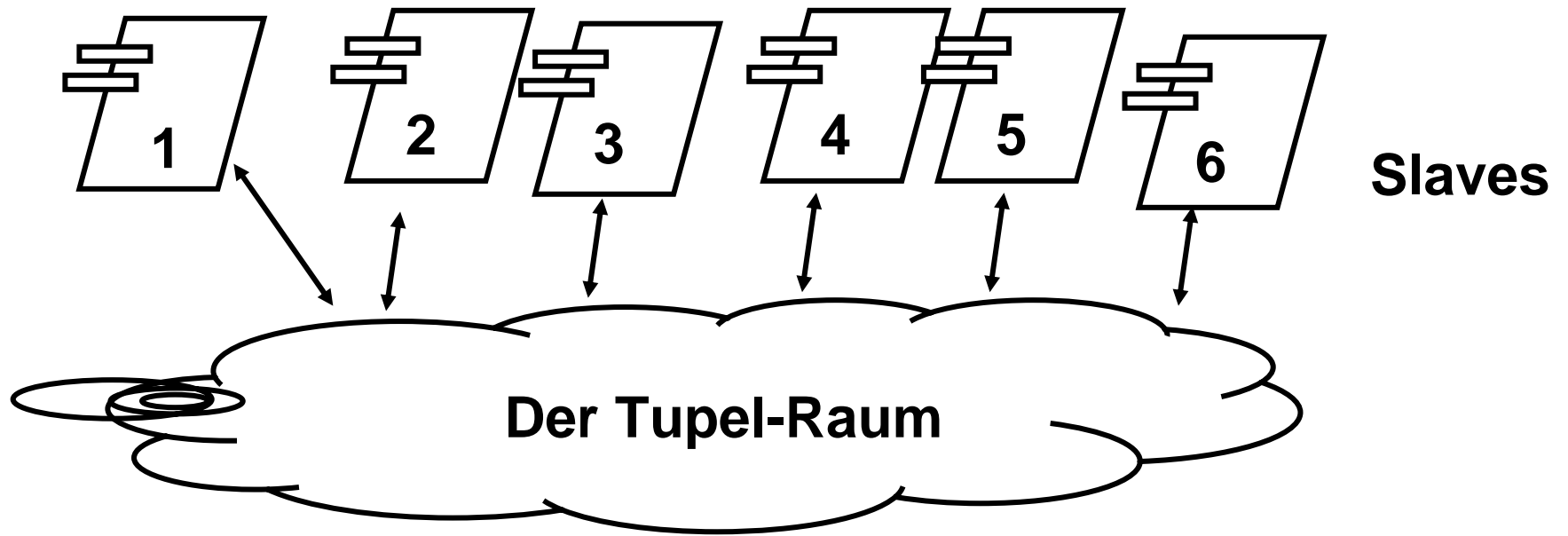
- wartet auf ein Tupel, das zum vorgegebenen Muster passt
- wenn ein Tupel gefunden wird, wird das Tupel entnommen

```
in("arraydata", ?dim1,  
?dim2);
```

## rd

- wartet auf ein Tupel, das zum vorgegebenen Muster passt
- wenn ein Tupel gefunden wird, wird das Tupel kopiert

```
rd("arraydata", ?dim1,  
?dim2);
```



# Beispiele für Middleware

Nachrichtenaustausch		Daten- verteilung	Berechnungs- infrastruktur
synchron	asynchron		
OS-Ports	DB	O2R-mapping eSQL	CICS
<b>Web-Services</b>	MQSeries	Tuple-spaces in Linda	CORBA

## Andere valide Beispiele

- Web-Server
- LDAP
- EJB, .NET, JavaBeans, COM

# Web Services als Middleware

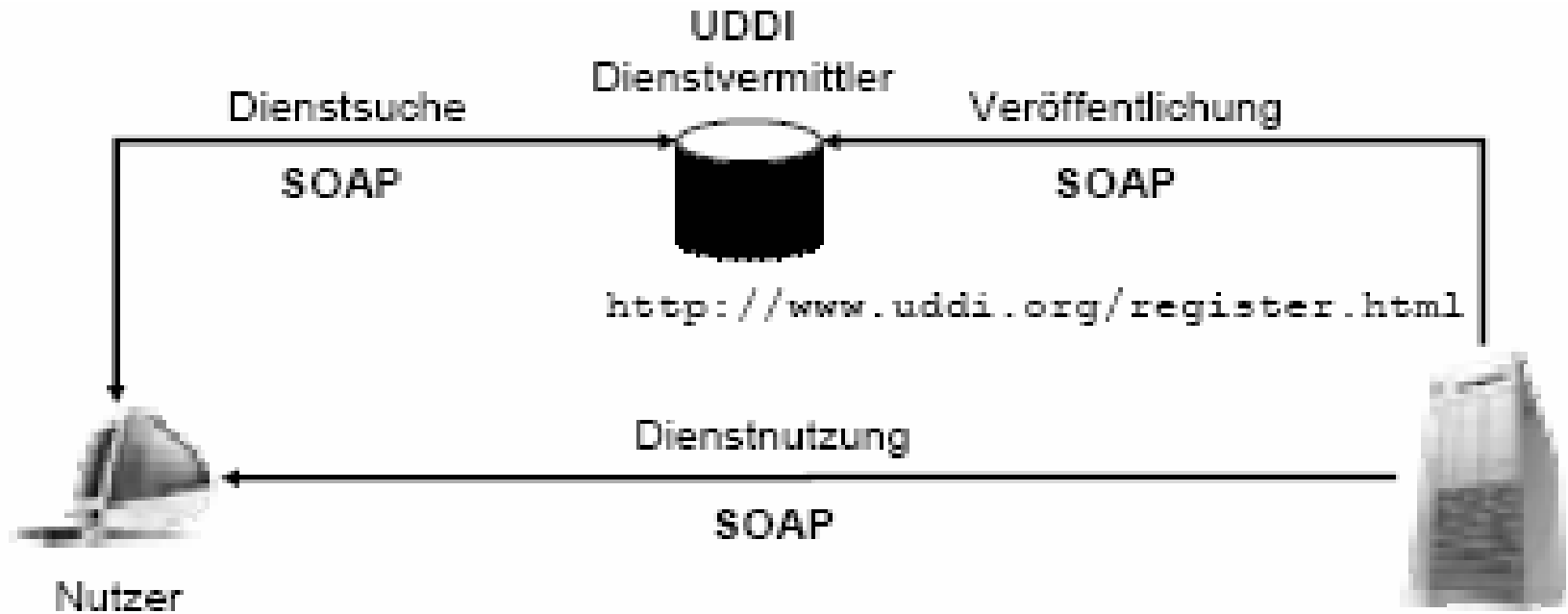
Web Services zur Entwicklung verteilter Objektsysteme über das WWW

- gewährleisten plattformübergreifende Interoperabilität bei Verwendung von Standard-Protokollen und XML-basierten Nachrichtenformat
- nutzen HTTP Request/Response-Kommunikation
- Middleware-over-HTTP

Wichtige verwendete Standards:

- UDDI
  - als Verzeichnisdienst zur Registrierung von Web Services. Es ermöglicht das dynamische Finden des Web Services (z. B. den Dienst *Kinofilme*) durch den Konsumenten.
- WSDL
  - zur Spezifikation der unterstützten Methoden (z. B. *OscarPreistraeger*) und deren Parametern (z. B. *Datum*) für den Programmierer.
- SOAP (oder XML-RPC)
  - zur Kommunikation. Hier wird der eigentliche Aufruf gestartet.

# Arbeitsweise von Web Services



- Der Anbieter veröffentlicht in einem Verzeichnis die Beschreibung seiner Dienste.
- Der Nutzer durchsucht das Verzeichnis und wählt den gewünschten Dienst aus.
- Nachdem eventuell weitere Protokolldetails ausgetauscht werden, findet die dynamische Anbindung des Nutzers an den Anbieter statt. Der Nutzer verwendet nun Methoden.

# Chancen und Probleme der dienst-orientierten Programmierung

- Dienste sind schon heute das Geschäft mit den größten Zuwachsraten in der IT-Industrie.
- Dienst-orientierte Programmierung hat große Chancen das Programmier- und Komponentenparadigma für verteilte Programmierung zu werden.
- Probleme sind heute Transaktionssicherheit, dynamische Komposition, Performanz, Sicherheit, ...

## ➤ EU-Projekt SENSORIA: Software Engineering for Service-Oriented Overlay Computers

- 18 Partner aus 7 Ländern; Koordinator: PST
- verfolgt einen neuen umfassenden Ansatz zur Entwicklung von dienst-orientierten Systemen
  - durch Integration pragmatischer SW-Entwicklungsmethoden mit grundlegenden Theorien, Modellen und Analysetechniken
- <http://www.sensoria-ist.eu/>



# Typisches Szenario für SENSORIA Service Design



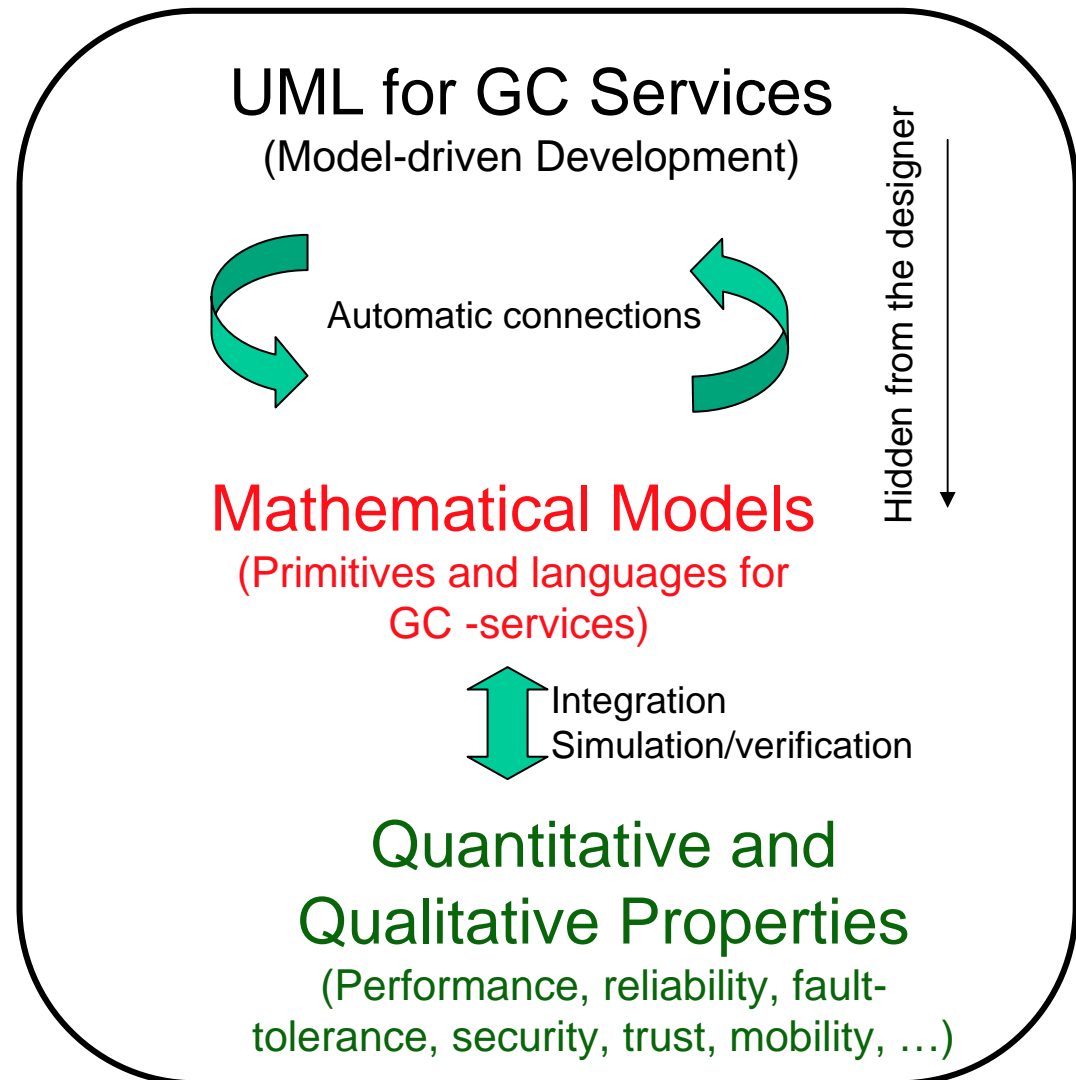
## SENSORIA Development

integrates

**practical SW Engineering**

with

**math. foundations**



# Ausblick auf Block B Teil 4: Architekturbeschreibungssprachen

