



Netzprogrammierung Aus- und Rückblick

Prof. Dr.-Ing. Robert Tolksdorf, Klaus Schild,
Malgorzata Mochol, Lyndon Nixon
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
<http://www.robert-tolksdorf.de>

Überblick

- RPC vs. Messaging
- Schwächen von RPC
- Agentensysteme



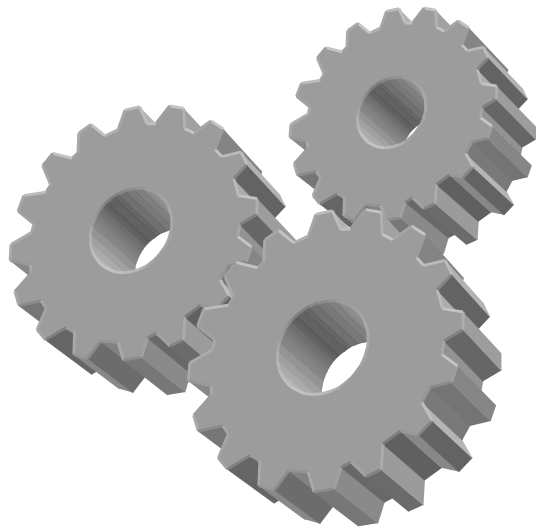
RPC vs. Messaging

über HTTP

- heute üblich
- Request-Response-Verhalten von HTTP unterstützt **RPCs**.
- verbindungsorientiert
- Übertragung: Sender und Empfänger müssen präsent sein.
- typischerweise synchron

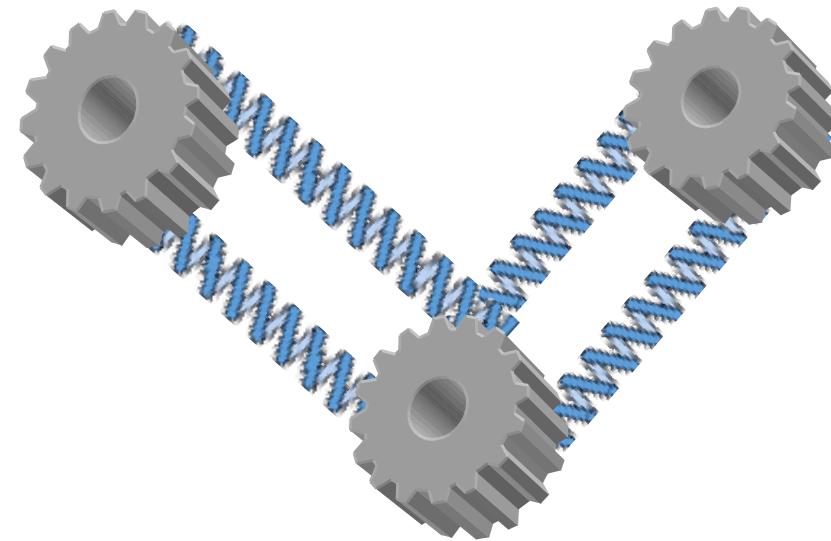
über SMTP

- heute eher selten
- realisiert **Messaging**
- persistente Kommunikation
- Übertragung: Weder Sender noch Empfänger muss präsent sein.
- typischerweise asynchron
- erlaubt Lastverteilung und Priorisierung



enge Kopplung

- verbindungsorientierte, synchrone Kommunikation
- z.B. SOAP/HTTP



lose Kopplung

- gepufferte, asynchrone Kommunikation
- z.B. SOAP/SMTP
- robuster, aber auch komplexer zu entwerfen

- Nachrichten konform:
 - zu einer vordefinierten Beschreibung des Funktionsaufrufes
 - zu der zu erwartenden Rückantwort

- Arten der Kommunikation:
 - Anfrage (Request)
 - Antwort (Response)
 - Fehlerfall (Fault)

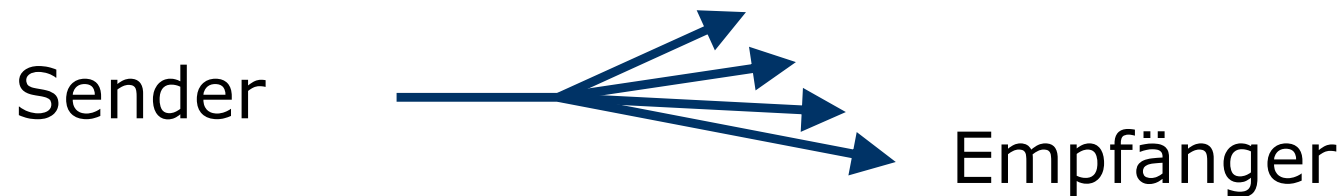
- Anwendungen interagieren durch Austausch von Nachrichten miteinander:
- Kommunikation in Anwendung sichtbar: senden und empfangen
- ⇒ Unterschied zu RPCs
- verschiedene Formen des Messaging:
 1. Kommunikationsstruktur
 2. Interaktionsmuster
 3. flüchtig vs. persistent
 4. synchron vs. asynchron
 5. Quality of Service

1. Kommunikationsstruktur

- Anzahl und Organisation der Kommunikationspartner
- wichtigste Kommunikationsstrukturen:
 - One-to-One-Kommunikation
 - One-to-Many-Kommunikation



- Sender sendet Nachricht an bestimmten Empfänger.
- Beispiel: Kunde sendet Bestellung per E-Mail an Firma



- Sender sendet identische Kopie gleichzeitig an mehrere Empfänger.
- Sender (publisher) veröffentlicht Nachricht zu bestimmten Thema (topic), zu dem sich die Empfänger (subscriber) angemeldet haben.
- ⇒ auch **publish-subscribe** oder **topic-based messaging** genannt
- Beispiel: Mailing-Liste

2. Interaktionsmuster

Client  Server

Einweg (one way,
fire and forget)

Client  Server

Anfrage-Antwort
(request-response)

Client  Server

Benachrichtigung
(notification)

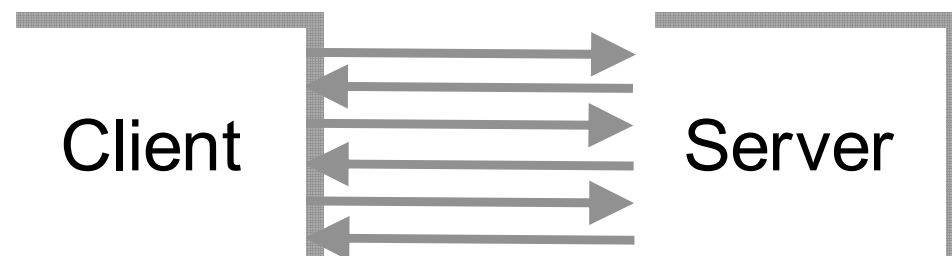
Client  Server

**Benachrichtigung-
Antwort** (notification-
response)

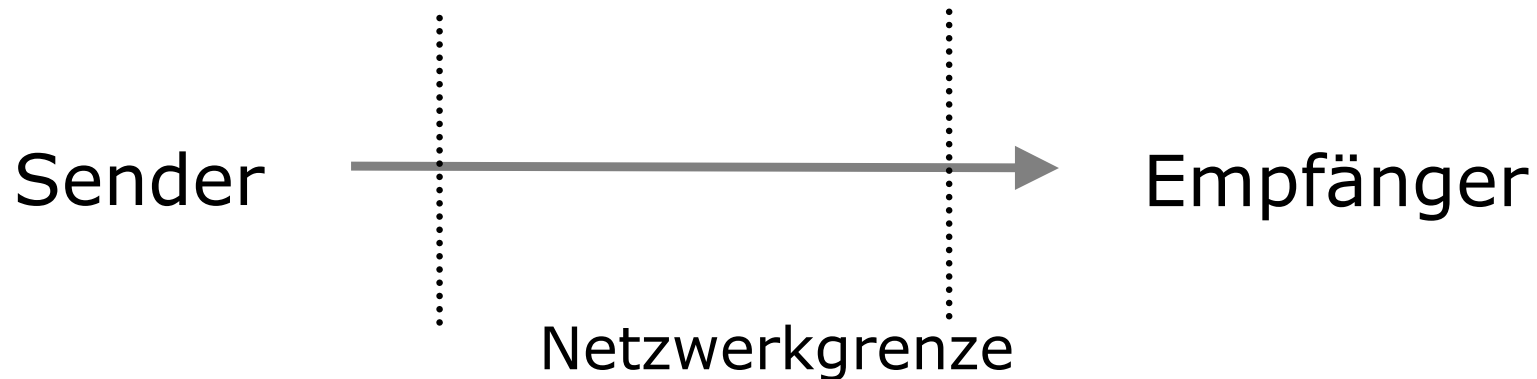
Beachte: „Antwort“ bezieht sich auf jeweilige
Anwendung, nicht auf das Netzwerk.

- ➔ Bestellanfrage mit spätestem Liefertermin
- ← Angebot mit zugesichertem Liefertermin
- ➔ Bestellung
- ← Bestätigung des Eingangs der Bestellung
- ➔ Bestätigung der Lieferung
- ← Rechnung

Bestellvorgang

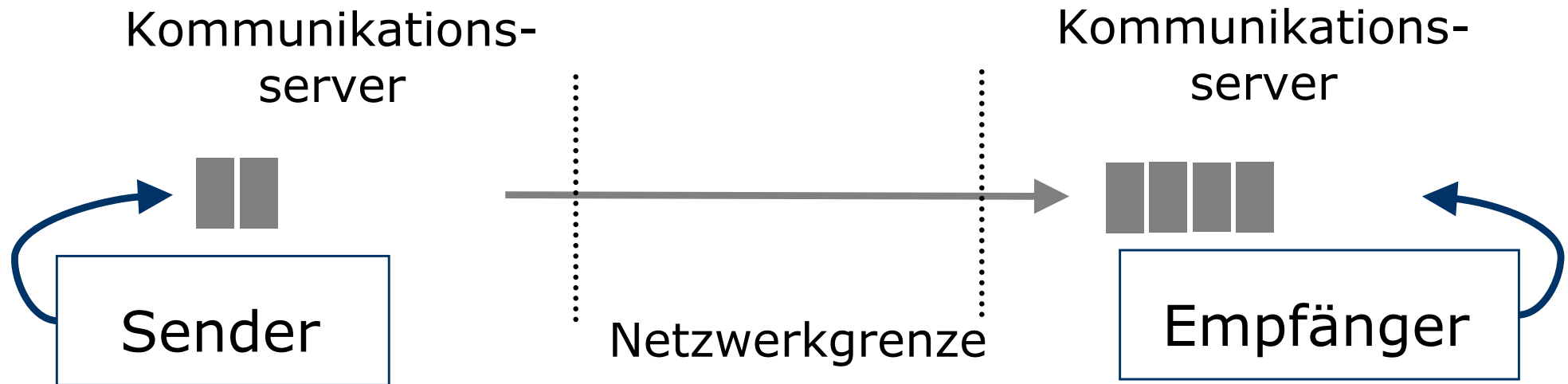


3. Flüchtig vs. persistent



flüchtige Kommunikation

- Sender und Empfänger kommunizieren direkt ohne Puffer miteinander.
- Sender und Empfänger müssen während der gesamten Übertragung präsent sein
- engl. **transient**
- Beispiele: HTTP, UDP



persistente Kommunikation

- Nachricht solange gespeichert, bis sie tatsächlich zugestellt wurde.
- Weder Sender noch Empfänger müssen während Übertragung präsent sein.
- Beispiele: E-Mail, MQSeries (IBM), JMS

4. Synchron vs. Asynchron

- **Asynchron**
 - Senden und Empfangen zeitlich versetzt
 - ohne Blockieren des Prozesses (ohne Warten auf die Antwort)
- **Synchron**
 - Senden/Empfangen synchronisieren → warten (blockieren) bis die Kommunikation abgeschlossen ist.
- bei **persistenter Kommunikation** → Messaging normalerweise asynchron
- bei **flüchtiger Kommunikation** → Messaging kann sowohl synchron als auch asynchron sein.

RPC oder Messaging?

RPC

⇒ eng gekoppelte, starre Systeme

- + einfach, abstrahiert von Kommunikation
- nur Eins-zu-Eins-Kommunikation
- Client und Server müssen präsent sein.
- skaliert weniger gut

Messaging

⇒ lose gekoppelte, robuste Systeme

- abstrahiert nicht von Kommunikation
- + erlaubt auch One-to-Many-Kommunikation
- + Weder Sender noch Empfänger müssen präsent sein.
- + erlaubt Priorisierung und Lastverteilung
- + skaliert sehr gut



Schwächen des RPC Konzepts

- RPC ist
 - sehr populär
 - weit übertragbar
 - implementierbar
- Aber:
 - RPC ist nicht abschließende Lösung
- [Tanenbaum/vanRenesse88] diskutieren einige der Probleme

Konzeptionelle Probleme

- Problem: Rollenidentifikation als Klient oder Server

- `sort < infile | uniq | wc -l > outfile`



- Wer liest, wer schreibt, wer betreibt die Berechnung?
 - fordert wc Zeilen vom uniq Prozess an?
 - fordert uniq den wc Prozess zur Weiterverarbeitung auf?
- Problem: Rollenwechsel in der Interaktion
 - Änderungsbenachrichtigungen an Klienten
 - Klienten sind dann auch Server
 - Signale des Klienten an Server
- Problem: Mehrparteieninteraktionen
 - Datenverteilung an mehrere Server

Weitere Probleme

- Transparenz bei Parametern
 - Zeiger
 - Globale Variablen
- Fehler
 - Server hat Fehler -> Klient blockiert
 - Klient hat Fehler -> Server steht alleine
 - Exactly-Once-Semantik schwer zu etablieren -> I/O
- Nebenläufigkeit
 - Blockierter Klient beim Aufruf (bei synchronem RPC)
 - Partielle Ergebnisse können nicht zur Weiterverarbeitung abgeliefert werden (z.B. bei Datenbankabfrage)

Alternativen zu RPC

- Neben dem RPC Konzept gibt es weitere Versuche, andere Interaktionsmodelle für Netzprogrammierung zu bilden
 - Koordinationssprachen: Tupelraum
 - Peer-to-Peer: Freie Rollen
 - Agenten: Autonome Komponenten



Agenten

- Die RPC-basierte Marssonde
 - Flugdatenübermittlung per RPC von der Erde
 - Ändert Kurs auf RPC Aufruf
 - Entscheidungen über Kursänderungen werden auf der Erde getroffen
- Unpraktikabel
 - Sehr lange Latenz
 - Entscheidungsfristen kleiner als Netzlatenz
 - Risiko der Fehlersemantik... Realer Absturz...
- Problem
 - Sonde ist nicht autonom in ihren Entscheidungen

Agenten

- Agent ist
 - ein Rechnersystem, das *eigenständig* für einen Benutzer agiert
- Mehragentensystem ist
 - ein Rechnersystem, das aus mehrere Agenten besteht, die miteinander *interagieren*
- Agenten müssen
 - kooperieren
 - sich koordinieren
 - verhandeln

- Vorteile
 - Autonomie
 - Verschiebung der Sichtweise auf Systeme von der Berechnung zur Interaktion
 - Besseres Abbild realer Gesellschaften

- Nachteile
 - Unklare Realisierung
 - Unterscheidung zu
 - Verteilten Systemen (aber: Autonomie)
 - KI (aber: Verteiltheit und Realisierungsfrage)
 - Spieltheorie/Ökonomie (aber: Automatisierung)
 - Gesellschaftswissenschaften (aber: Informatiker)

Agenten

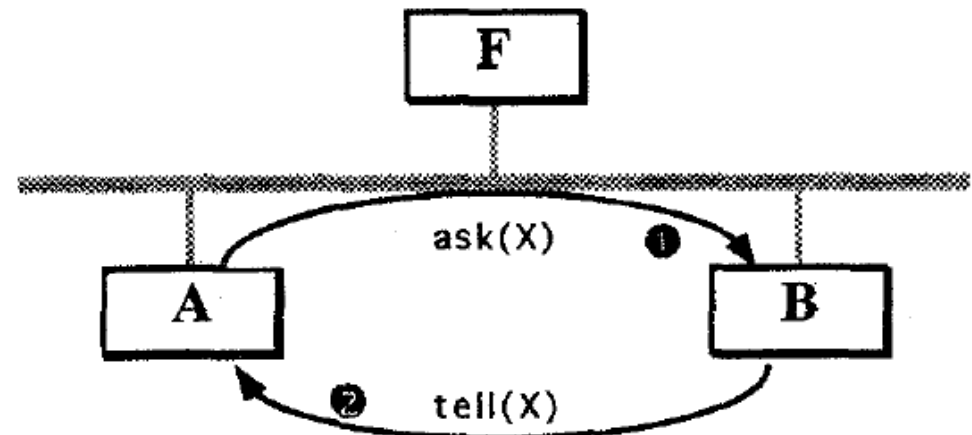
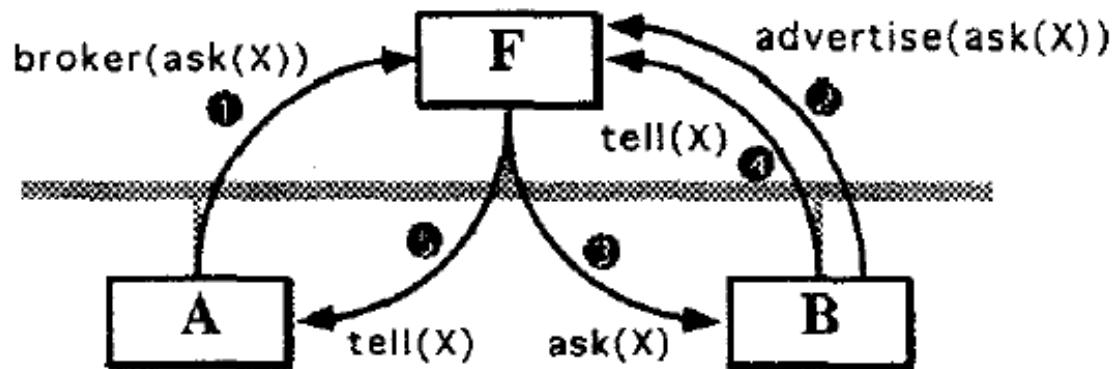
- Schwache Definition
 - Autonomie: Agent operiert ohne direkte Steuerung von außen und kontrolliert seine Aktionen selber
 - Responsiveness/Empfindlichkeit: Agent beobachtet Umgebung und reagiert auf Änderungen darin
 - Proaktiveness/Initiativ: Agent reagiert nicht nur, sondern wird selbständig zur Erreichung von Zielen aktiv
 - Sozial: Agent interagiert mit anderen zur Erreichung eigener und deren Ziele
- In der Regel mit Fokus auf Software-Agenten

- Starke Definition
 - Mobilität: Agenten bewegen sich in einem elektronischen Netzwerk (siehe auch: Roboter)
 - Veracity/Aufrichtigkeit: Agenten geben nicht wissentlich falsche Informationen weiter
 - Rationalität: Agenten beschädigen nicht durch Aktionen ihre eigenen Ziele
 - Kooperativität: Agenten arbeiten mit (menschlichen) Auftraggebern zusammen und übernehmen deren Ziele
- Zieht Aspekte menschlichen Verhaltens ein
- ... Intelligent Agents, Smart Agents ...

- Kooperation
 - Kooperative Interaktion: Agenten koordinieren ihre Aktionen. Koordination ist durch Programmierer vorgegeben
 - Vertragsbasierte Kooperation: Absichten der Agenten stehen leicht in Konflikt. Koordination durch Marktmechanismen wie Auktionen
 - Verhandlungsbasierte Kooperation: Verhandlungen zwischen Agenten, deren Ressourcenbedarf in Konflikt steht
- Rationalität
 - Alleine: Agent handelt so, dass er seine Ziele erreicht
 - In Gemeinschaft: Agent handelt so, dass der gemeinsame Nutzen größer ist als der gemeinsame Verlust bei einer anderen Aktion

- Kommunikation
 - Agenten müssen Wissen austauschen
 - Agent Communication Languages (ACL)
 - KQML: Sprache zur Äußerung von Kommunikationsakten
 - Auf Sprechakten basiert
 - Mitteilungen haben Typ
 - Definierte Semantik
 - KIF: Sprache zur Repräsentation von Wissen
 - Wie wird der semantische Inhalt der Mitteilung repräsentiert
 - FIPA ACL
 - FIPA Konsortium
 - Ontologien
 - Worüber wird geredet
 - Was sind die Konzepte der Anwendungsdomäne

- (ask-all
 :content „price(IBM, [?price, ?time])“
 :receiver stock-server
 :language standard-prolog
 :ontology NYSE-TICKS)



Beispielsystem

- Zum Ausprobieren:
JADE (Java Agent DEvelopment Framework)
- <http://sharon.csel.it/projects/jade/>
- Java-basiert, FIPA kompatibel, Grafische Oberfläche

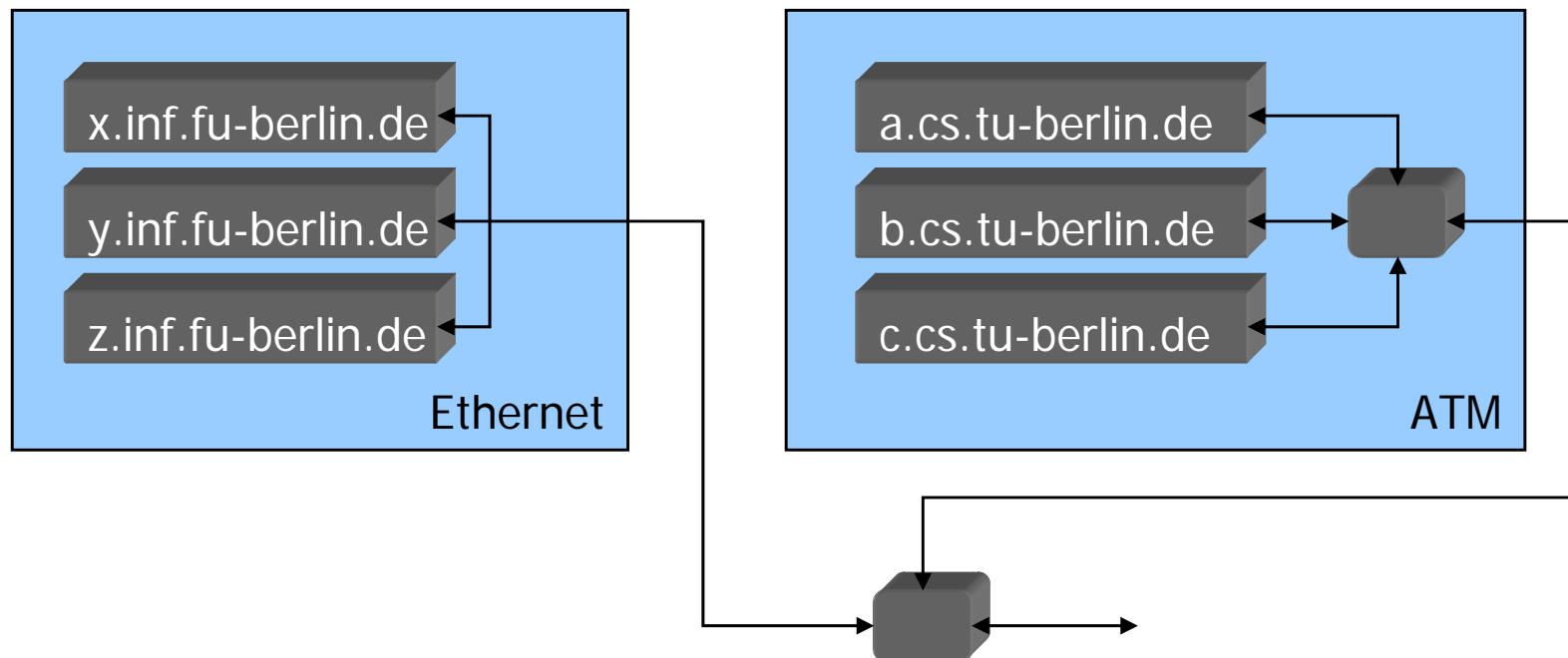


Rückblick



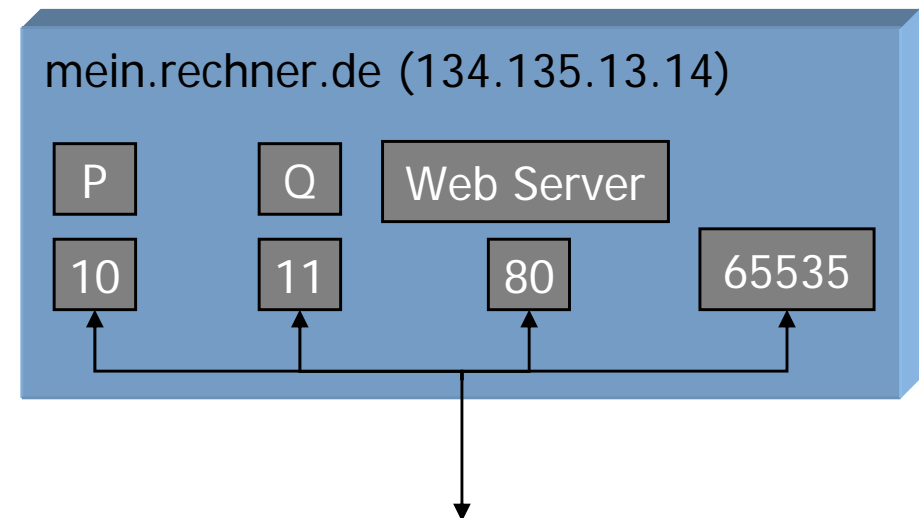
Internet-Kommunikation Sockets

- Das Internet Protokoll IP ermöglicht Internetworking durch Etablierung eines Datenformats und Transportprotokollen, die auf unterschiedlichen Datenverbindungen aufgesetzt werden können



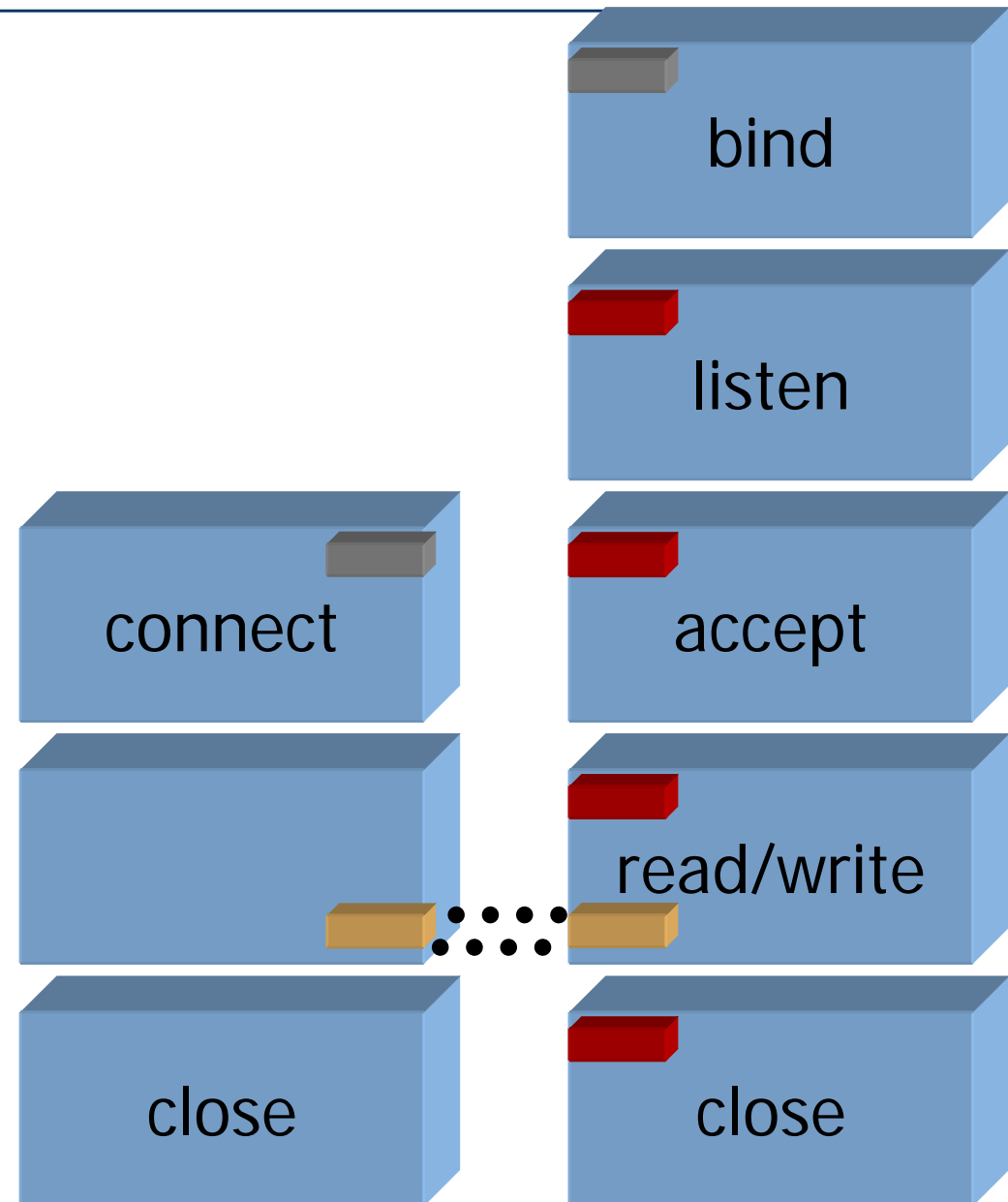
Transport Protokolle

- Zwei Protokolle zum Datentransport
 - *UDP*: Ein Paket (Datagramm) von Rechner A nach Rechner B schaffen – Verbindungslos
 - *TCP*: Pakete werden *geordnet* und *zuverlässig* über eine Verbindung transportiert – Verbindungsorientiert
- Ports als Kommunikationsadresse
 - Ein *Port* ist ein logischer Netzanschluss, benannt von 0 bis 65535
- *Socket* ist Endpunkt einer Verbindung

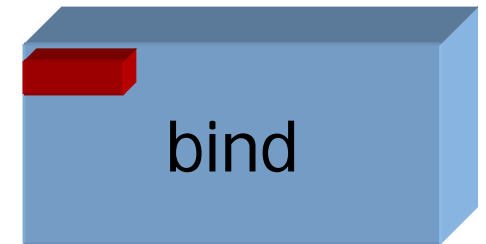


TCP Sockets

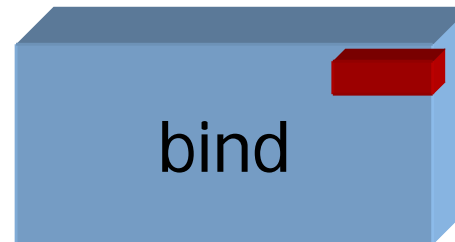
1. Server reserviert Port
2. Server nimmt Verbindungswünsche an
3. Client schickt Verbindungswunsch
4. Client und Server sind verbunden
bidirektional!
5. Verbindung wird abgebaut



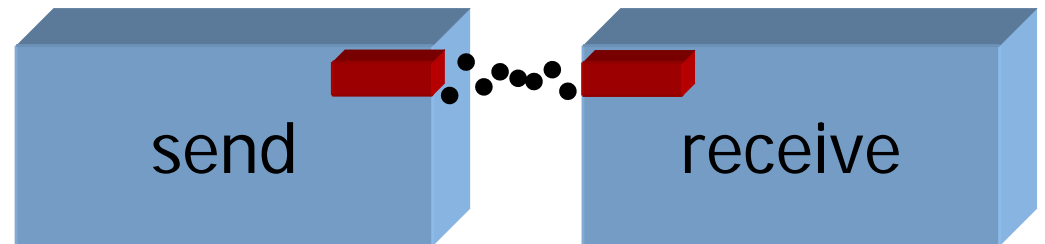
1. Server bindet Socket



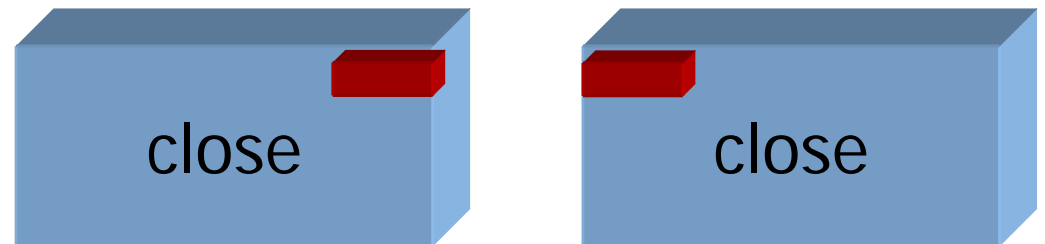
2. Client bindet Socket



3. Client und Server
senden und empfangen
bidirektional!



4. Sockets werden
aufgegeben





Internet Dienste

Internet-Protokolle und -Dienste

- Einordnung von Internet-Protokollen:

| | | | | | | | | | | | | |
|------|------|--------|------|-----|------|--------|-----|---|---|---|---|------------------|
| SMTP | NNTP | Finger | HTTP | FTP | SNMP | telnet | RTP | ⋮ | ⋮ | ⋮ | ⋮ | Dienstprotokolle |
|------|------|--------|------|-----|------|--------|-----|---|---|---|---|------------------|

| | | | |
|-----|-----|--|--------------------------|
| UDP | TCP | | Transport- protokolle |
|-----|-----|--|--------------------------|

| | | |
|----|------|--------------------------------|
| IP | ICMP | Netzverbindungs- protokolle |
|----|------|--------------------------------|

| | |
|--|----------------|
| Lokale Netze (Ethernet, ISDN, ATM, etc.) | Netzprotokolle |
|--|----------------|

- Internet Dienste sind (zumeist) definiert durch
 - Aufgabe
 - Portnummer auf dem der Dienst angeboten wird
 - Transportprotokoll (TCP oder/und UDP)
 - Protokoll
- Z.B.: Web Dienst
 - Übertragen von HTML Seiten
 - Port 80
 - TCP
 - HTTP
- Z.B.: Usenet Dienst
 - Übertragen von News
 - Port 119
 - TCP
 - NNTP

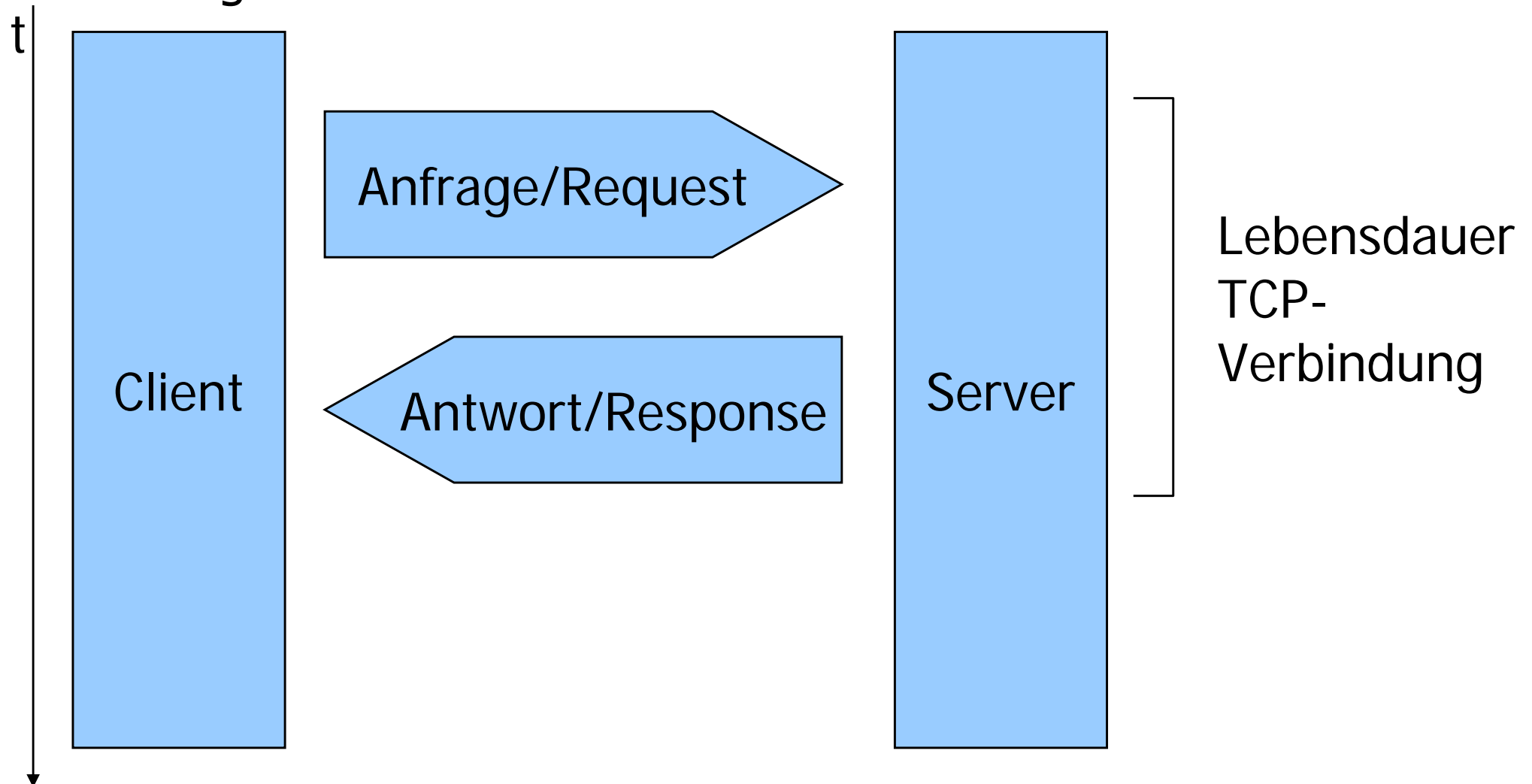


HTTP

Hypertext Transfer Protocol

- Aufgabe:
Transfer von Informationen zwischen Web-Servern und Clients
- Port:
80 ist für HTTP reserviert
- Transportprotokoll:
TCP
- Protokoll:
R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee. *Hypertext Transfer Protocol - HTTP/1.1*. RFC 2616,
<http://www.ietf.org/rfc/rfc2616.txt>

- Zustandsloses Protokoll
- Anfrage mit Antwort beantwortet



Aufbau Anfrage

- Anfrage besteht aus
 - Anfragemethode
 - Anfragebeschreibung durch Kopfzeilen
 - Allgemeine Beschreibungen
 - Anfragespezifische Beschreibungen
 - Beschreibung eventuell beiliegenden Inhalts
 - Leerzeile
 - Eventueller Inhalt
- Beispiel:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.04Gold (Win95; I)
Host: megababe.isdn:80
Accept: image/gif, image/jpeg, image/pjpeg, */*
```

Aufbau Antwort

- Antwort besteht aus
 - Antwortcode
 - Antwortbeschreibung durch Kopfzeilen
 - Allgemeine Beschreibungen
 - Antwortspezifische Beschreibungen
 - Beschreibung eventuell beiliegenden Inhalts
 - Leerzeile
 - Eventueller Inhalt

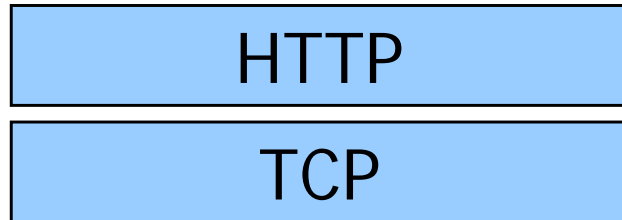
- Beispiel:

```
HTTP/1.0 200 OK
Last-Modified: Sun, 15 Mar 1998 11:26:50 GMT
MIME-Version: 1.0
Date: Fri, 20 Mar 1998 16:43:11 GMT
Server: Roxen-Challenger/1.2beta1
Content-type: text/html
Content-length: 2990
```

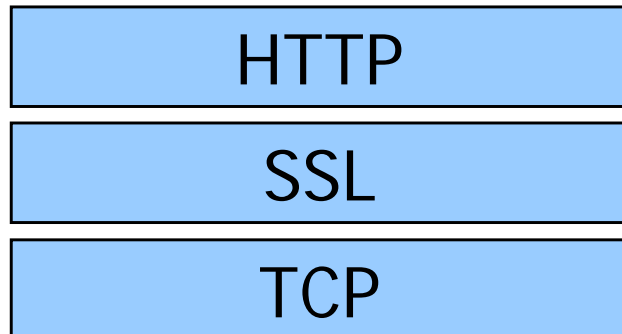
```
<HTML><HEAD><TITLE>TU Berlin ---
```

HTTP über SSL Sockets

- HTTP benutzt TCP Sockets zur Kommunikation



- Secure Sockets Layer SSL erweitert Sockets um Sicherheitsmerkmale
- HTTPS bezeichnet eine HTTP Kommunikation über solche sicheren Sockets



- Port 443 als Default-Port festgelegt

- Abfragen der Ressource und Eigenschaften
 - Object getContent()
 - String getHeaderField(String name)
 - InputStream getInputStream()
 - OutputStream getOutputStream()
- Übliche Header
 - getContentEncoding()
 - getContentLength()
 - getContentType()
 - getDate()
 - getExpiration()
 - getLastModified()
- Sind unter Umständen errechnet oder leer

Client-Pull und Server-Push

- HTTP Interaktion mit Anfrage und Antwort wird durch Client-Pull und Server-Push erweitert
- Client-Pull
 - Client lädt Inhalte in regelmäßigen Abständen nach
 - Server löst das Verhalten durch zusätzlichen Header aus
- Server-Push
 - Server schickt mehrere Antworten nacheinander
 - Client ersetzt jeweils darstellung

Interaktion zur Authentifizierung

- Seiten im Web können Zugriffsschutz tragen
- Interaktion zum Abruf
 - Normales GET
 - Antwort 401 und WWW-Authenticate: Header, der Nachweis in unterschiedlichen Schemata anfordert
 - Weiteres GET mit Authorization: Header, der je nach Schema Parameter trägt
 - Antwort 200

Parameter für Web-Skripte

- Zwei Arten der Übermittlung von Parametern an Skripte:
 - GET: Daten werden in URL kodiert
 - POST: Daten werden kodiert über Standardeingabe geliefert

- HTML:

```
<html><body>
```

```
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi"
  method="get"> <input name="Eingabe" type="text">
  <input type="submit" value="Per GET">
```

```
</form>
```

```
<form action="http://flp.cs.tu-berlin.de/~tolk/echo.cgi"
  method="post">
```

```
<input name="Eingabe" type="text">
  <input type="submit" value="Per POST">
```

```
</form>
```

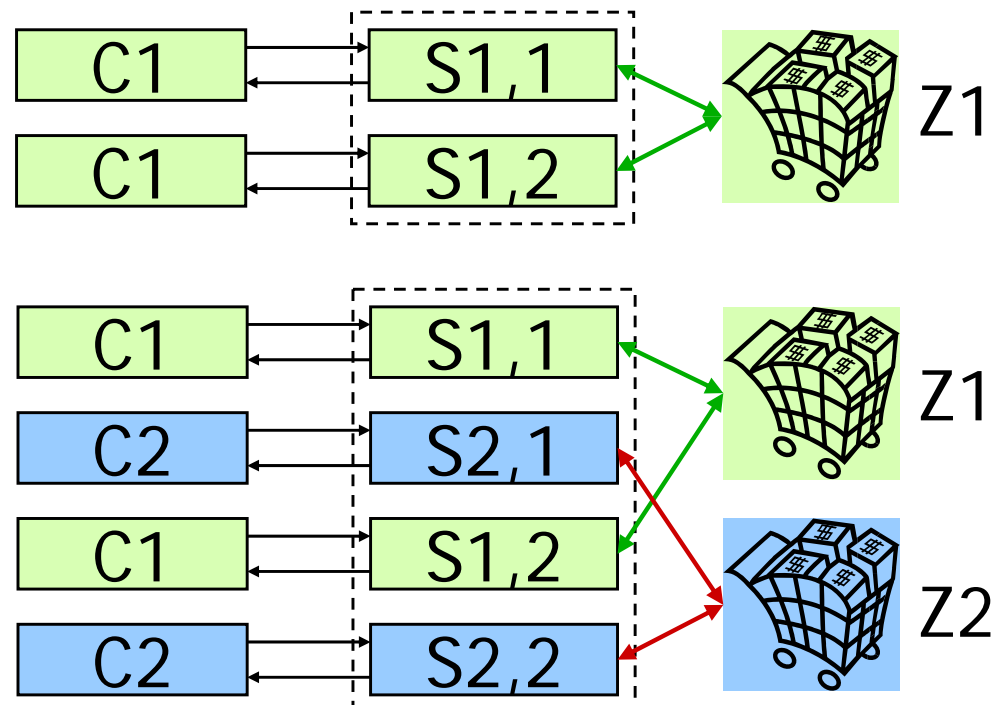
```
</body>
```

```
</html>
```

Kodierung von Eingabewerten

- Parameter haben bestimmten Übergabeformat
 - $\text{Name}_1=\text{Wert}_1\&\text{Name}_2=\text{Wert}_2$
- Dieser *Query String* wird
 - bei GET an die URL mit ? getrennt angehängt
 - `http://flp.cs.tu-berlin.de/~tolk/echo.cgi?Eingabe=Hallo!`
 - bei POST als Inhalt übermittelt und beim Web-Server über stdin einem Skript übergeben
- Query String selber muss kodiert werden
 - Um Transport zu sichern
 - Um bedeutungstragende Zeichen (=, & etc.) übertragen zu können
 - Medientyp der Nachricht ist `application/x-www-form-urlencoded`

- Einführung von Sitzungen (Sessions)
- Sitzung: Folge von Interaktionen, die einen gemeinsamen Zustand haben
- Identifikation in der Interaktion durch eindeutige Sitzungsnummer (Session-ID)
- Ermittlung des Zustand auf Basis der Session-ID

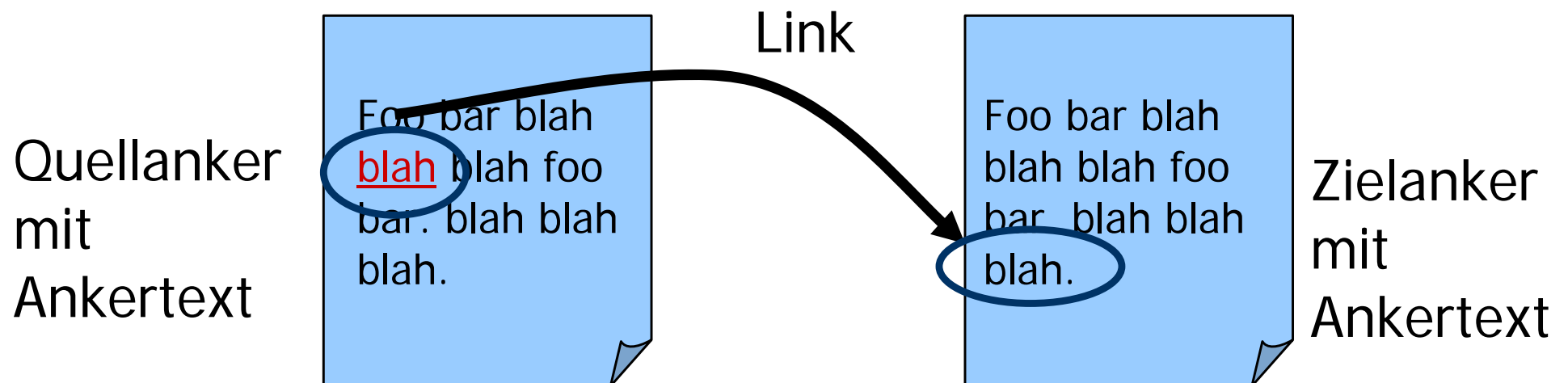




HTML

Hypertext Markup Language

- Konzepte:
 - Informationen werden als Dokumente aufgefasst
 - Dokumenteninhalte werden als Klartext dargestellt
 - Dokumententeile werden durch *Markierungen/Elemente/Tags* ausgezeichnet
 - Inhaltlich (`<h1>Einleitung</h1>`, `wichtig`)
 - Gestalterisch (`wichtig`)
 - Dokumente werden durch Links zu einem Hypertext verbunden (dadurch entsteht ein Netz, das Web)



- Paket javax.swing.text.html.parser enthält einen Parser für HTML
- Basiert auf Rückrufen bei Auftreten einer bestimmten Informationseinheit in HTML:

Parserfortschritt



<html> <head> <title>HTML Seite</title> ...

Ereignis:
Start des head Tags gefunden
Aufruf:
`handleStartTag(HTML.tag.HEAD`

Ereignis:
Ende des title Tags gefunden
Aufruf:
`handleEndTag(HTML.tag.TITLE)`

Ereignis:
Start des html Tags gefunden
Aufruf:
`handleStartTag(HTML.tag.HTML)`

Ereignis:
Start Text gefunden
Aufruf: `handleText("HTML Seite",0)`

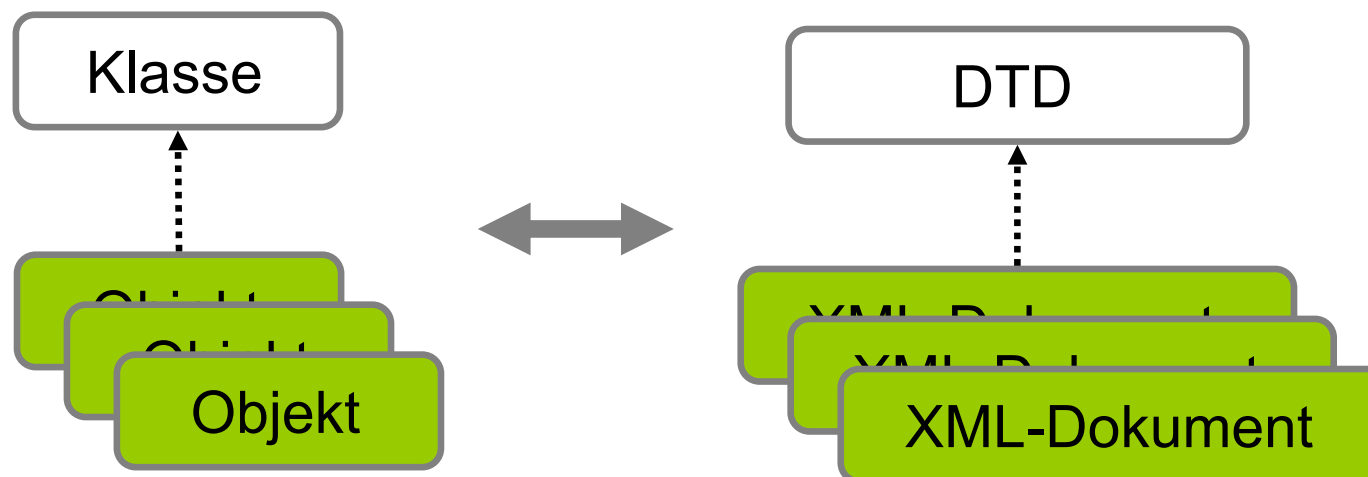


XML

- Jedes Anfangs-Tag muss ein zugehöriges End-Tag haben.
- Elemente dürfen sich nicht überlappen.
- XML-Dokumente haben genau ein Wurzel-Element.
- Elementnamen müssen den Namenskonventionen von XML entsprechen.
- XML beachtet grundsätzlich Groß- und Kleinschreibung.
- XML belässt Formatierungen (white space) im Text.

Dokument-Typen

- Beschreiben den prinzipiellen Aufbau von Dokumenten eines bestimmten Typs.
- Können mit DTDs (Document Typ Definitions) spezifiziert werden.
- DTDs wurden von SGML übernommen.
- DTDs benutzen Syntax, die regulären Ausdrücken ähnlich ist.



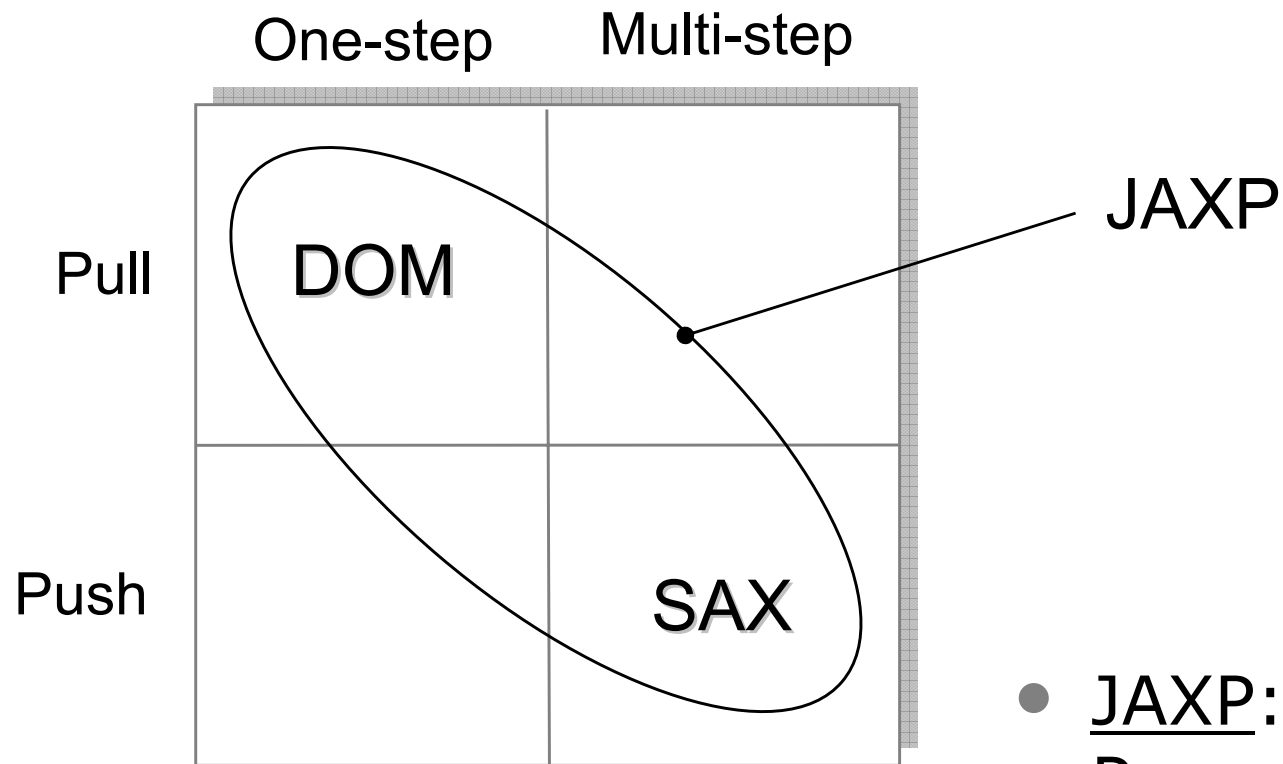
```
<!ELEMENT catalog (CD | cassette | record |  
MP3)* >
```

- Deklariert Element catalog
- * bedeutet n Wiederholung mit $n \geq 0$.
- | bedeutet Auswahl.
- CD, cassette, record und MP3 sind jeweils Kind-Elemente.

- Dokument-Typen beschreiben den prinzipiellen Aufbau von Dokumenten eines bestimmten Typs.
- Dokument-Typen können mit Klassen und XML-Dokumente mit Objekten verglichen werden.
- In einem XML-Dokument kann ein bestimmter Dokument-Typ spezifiziert werden.
- Das Wurzelement des Dokumentes muss dann genau der Struktur dieses Elementes entsprechen, wie sie im Dokument-Typ festgelegt ist.
- In diesem Fall bezeichnet man das Dokument als zulässig (engl. valid).

XML-Parser

- DOM: Document Object Model
- SAX: Simple API for XML

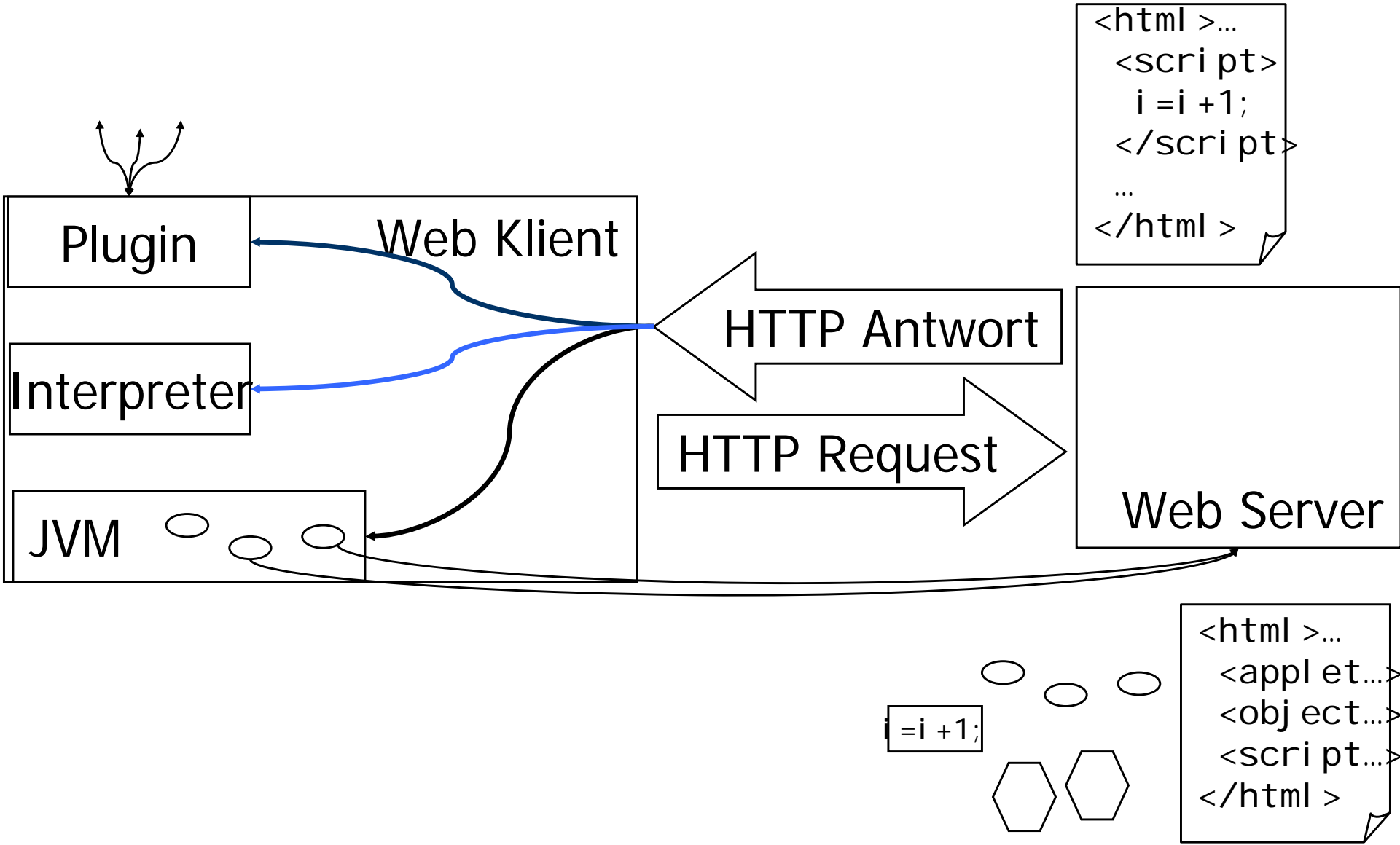


- JAXP: Java API for XML Processing



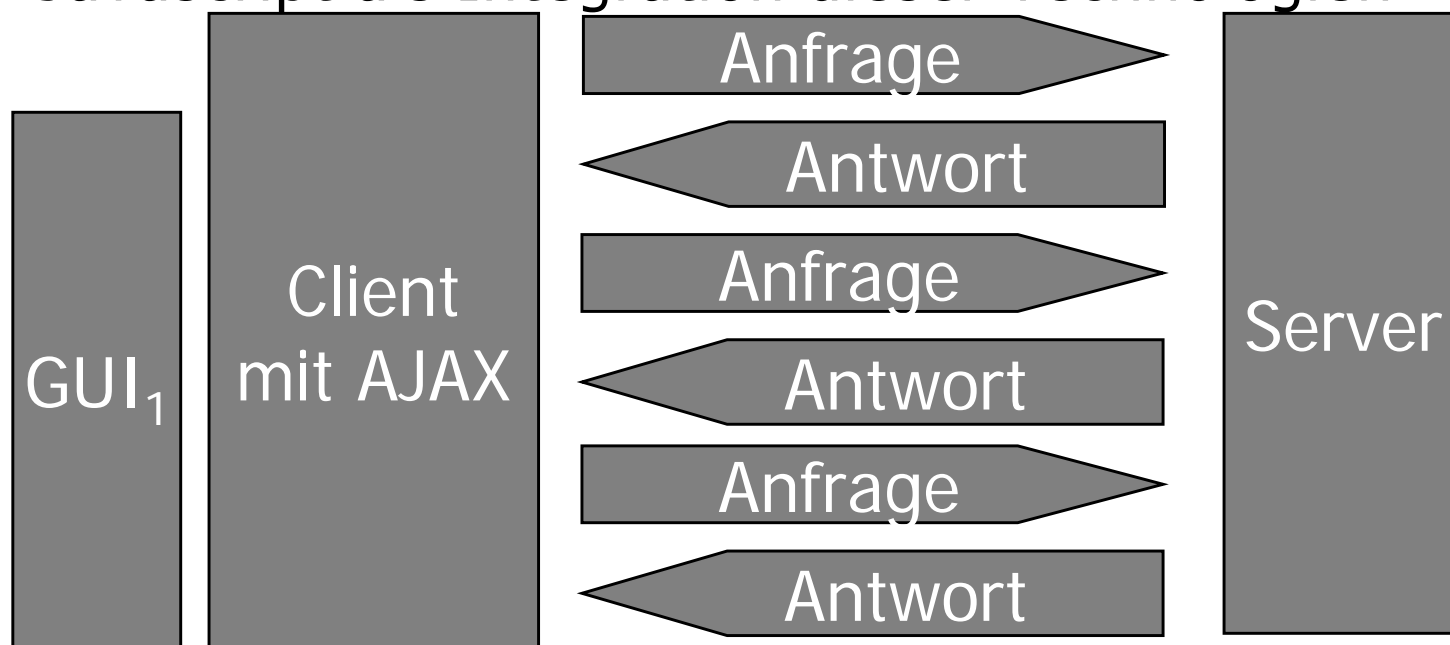
Klienten- und serverseitige Verarbeitung

Klientenseitige Verarbeitung



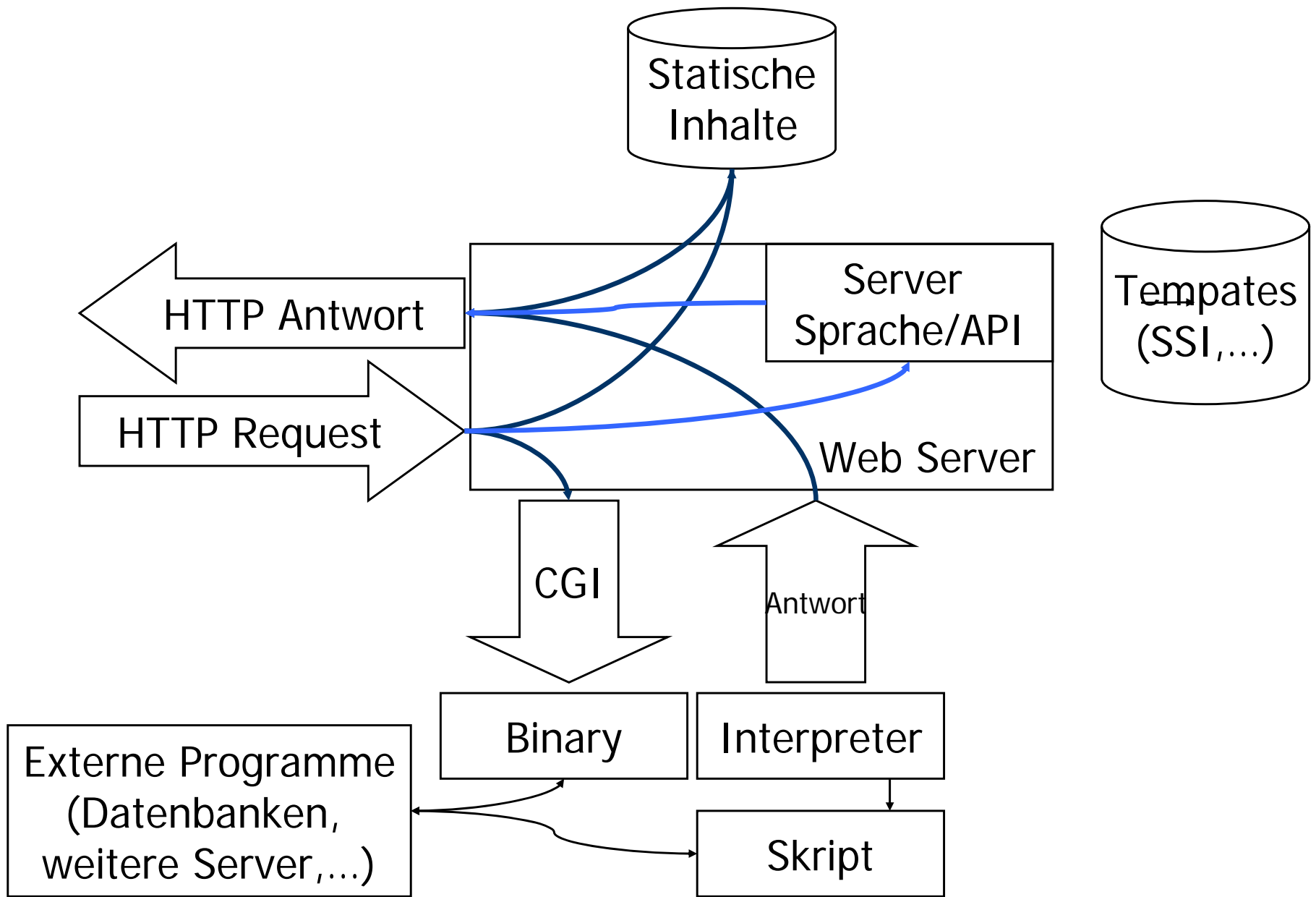
- JavaScript: Einfache imperative Programmiersprache
 - Programmcode als Quelle
 - Eingebettet in HTML-Seite
 - Ausgeführt durch Interpreter im Browser
 - Zugriff auf Ereignisse und Dokumentenstruktur
- Meilensteine und Implementierungen:
 - JavaScript (1995): Netscape/Sun
 - ECMAScript: ECMA
 - JScript: Microsoft (JavaScript+Windows)
 - Seit Version 1.2 nicht mehr kompatibel in Browsern implementiert.
 - Seit 1.5: DOM Beachtung, uniforme Repräsentation des Dokumenteninhalts

- Asynchronous JavaScript and XML (AJAX) realisiert dies durch Kombination von
 - Präsentationssprachen XHTML und CSS
 - Interaktion und Modifikation im Browser mit DOM
 - Datenaustausch mit XML
 - Datentransfer durch asynchrone HTTP-Anfragen
 - Javascript als Integration dieser Technologien



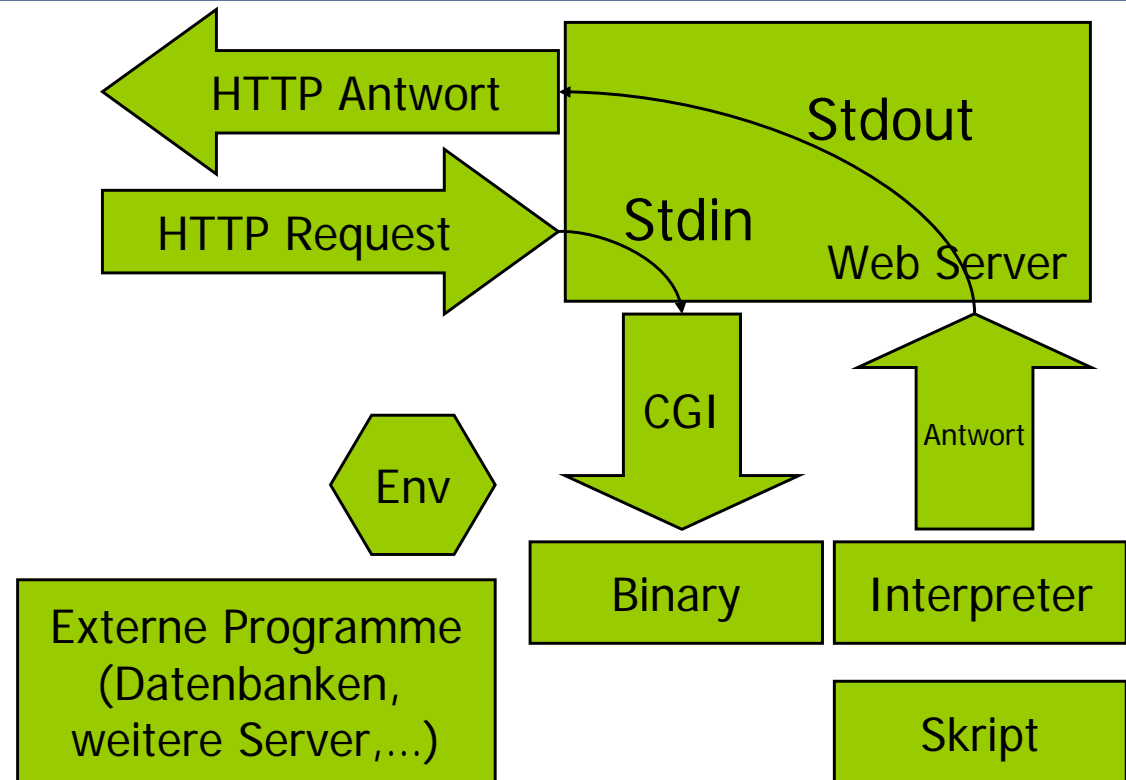
Applets

- Applets sind (kleinere) Java-Programme, die in einem Java-fähigen Web-Browser gestartet werden können.
- Das Einbinden von Applets in eine HTML-Seite erfolgt mit dem `<applet>`-Tag.
- Alle Applets sind Unterklassen von [java.applet.Applet](#)
- Die Klasse Applet hat folgende Oberklassen
 - `java.lang.Object`
 - `java.awt.Component`
 - `java.awt.Container`
 - `java.awt.Panel`



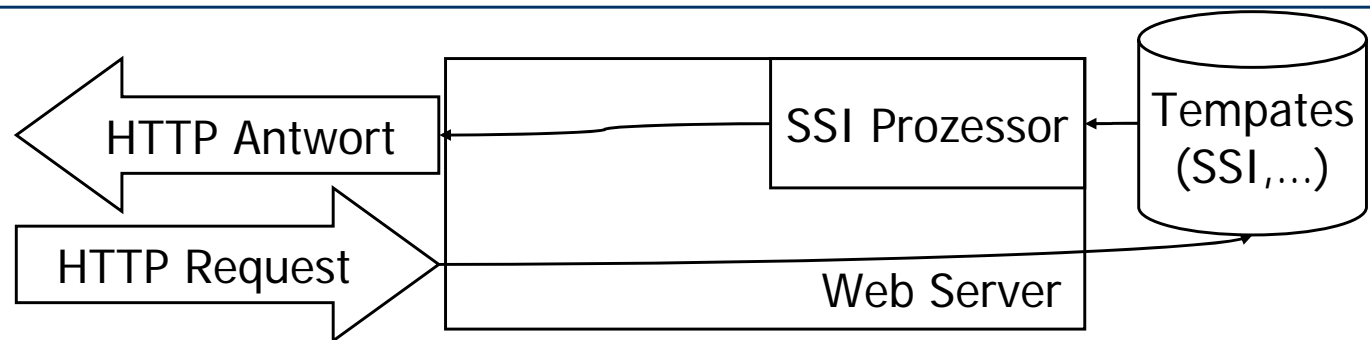
Ablauf

1. Web Server erkennt, dass URL ein Skript bezeichnet
2. Prozess wird gestartet
3. Prozess werden Eingaben mitgeteilt
 1. Initialisierte Umgebungsvariablen
 2. Standardeingabe
4. Standardausgabe des Prozesses wird zum Web-Server gelenkt
5. Skript wird ausgeführt
6. Ausgaben werden an Klienten weitergeleitet



Server Side Includes

- Server Side Includes (SSI) sind Anweisungen an den Web Server, die im HTML Code eingebunden sind



- Werden vom Server bei Auslieferung ausgeführt und zu HTML Code expandiert
- SSI Anweisung im statischen HTML-Code:
`<div align="right">`
Letzte Änderung: `<!--#flastmod file=""-->`
`</div>`
- Ergebnis nach Auslieferung
`<div align="right">`
Letzte Änderung: `Wednesday, 08-Jan-2003 09:30:15 CET.`
`</div>`

- SSI sind nicht wirklich eingebettete Programme
- PHP: HTML um „Skriptsprache“ erweitern die serverseitig ausgeführt wird und dessen Ergebnis in den HTML-Text eingebettet ist
- Einbettung im HTML-Code:

```
<html>
<head><title>Das Einführungsbeispiel schlechthin
...</title></head>
<body>
<?php echo "Brave New PHP World"; ?>
</body>
</html>
```
- Auch `<script language="php">...</script>` möglich

- Servlets:
 - CGI Programme könnten auch in Java geschrieben werden
 - Als „Interpreter“ müsste eine JVM gestartet werden
 - Wenn der Webserver selber auf einer JVM läuft, könnte er eine „CGI-Komponente“ nachladen und ausführen
 - Java Servlets sind solche Komponenten
- Unterschiede zu CGI und Applets
 - Parameterkommunikation läuft nicht über Umgebung und stdin/stdout sondern über Java-Schnittstelle
 - Applets als Komponente in JVM eines Browsers geladen
 - Servlets als Komponente in JVM eines Servers geladen
- Schnittstelle:
 - Erweitern von [HttpServlet](#)
 - Überschreiben von [doGet](#) und [doPost](#)

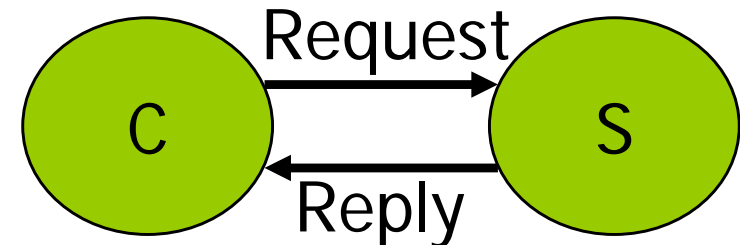
Java Server Pages

- Java Server Pages
 - In SSI gibt es rudimentäre Ausdrücke
 - Man könnte auch komplexere Programmfragmente mit HTML-Code mischen, wie bei PHP
 - Java Server Pages erlauben die Mischung von HTML-Code mit Java Fragmenten
 - Aus ihnen werden automatisch Servlets generiert und ausgeführt
 - HTML-Code nach Ausgabe schreiben
 - Java-Fragmente in Servlet Rahmen einbetten
- JSP bestehen aus
 - Scriptlets
 - JSP Ausdrücken
 - Deklarationen
 - JSP Anweisungen
 - HTML



Remote Procedure Calls

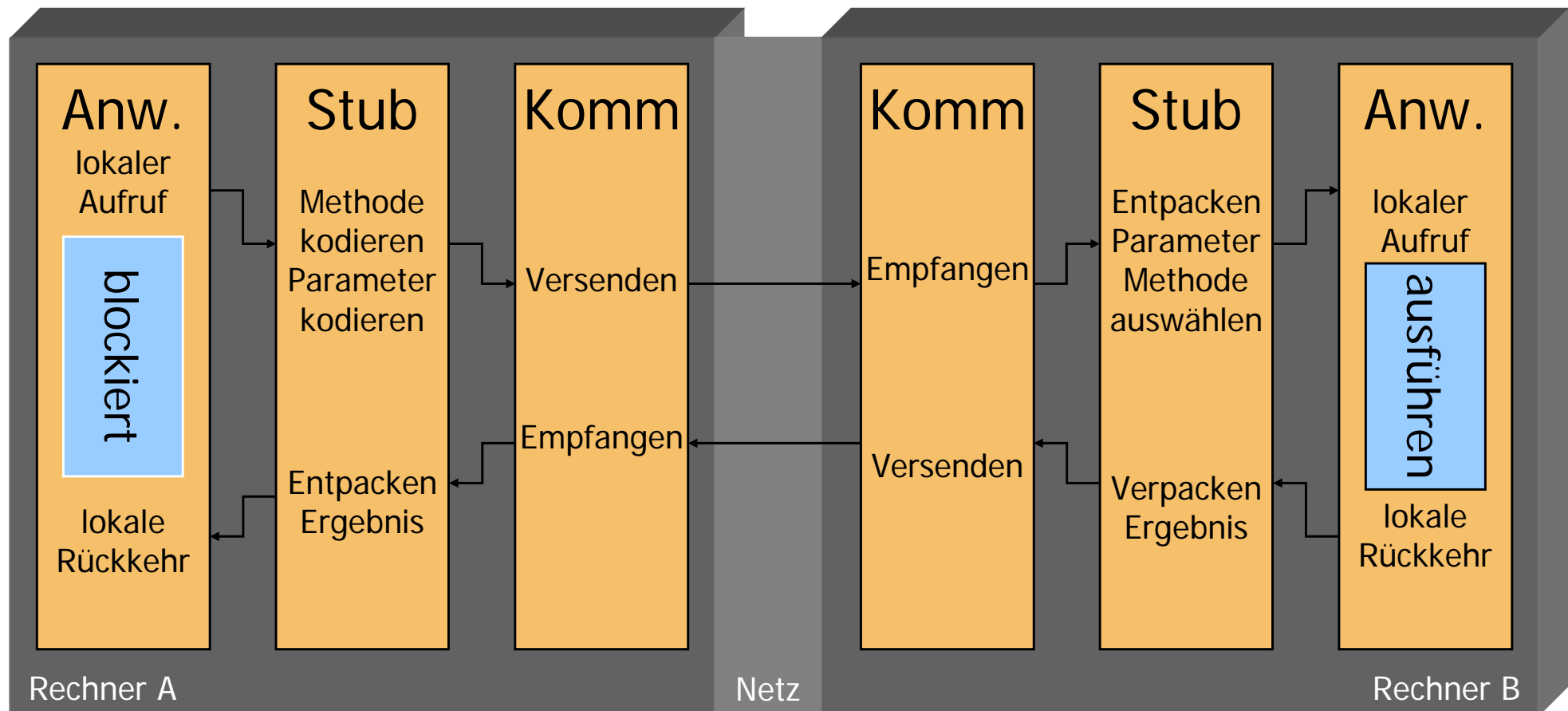
- Rechner interagieren über Rechnergrenzen durch
 - Nachrichtenaustausch (z.B. Internet Mitteilungen)
 - Fernaufruf (RPC, RMI, CORBA)
 - Simulierten gemeinsamen Speicher (Tupelraum)
 - ...
- Dominierendes Interaktionsmodell: Client/Server
- *Client*: Prozess, der Dienst von einem anderen Prozess anfordert (Anforderung, Request)
- *Server*: Prozess, der auf Anforderung eines Clients einen Dienst erbringt und Ergebnis vermeldet (Antwort, Reply)
- Feste (starre) Rollenverteilung
- Fester Interaktionsablauf, z.B. keine Zwischenergebnisse vorgesehen



| <i>Lokaler Aufruf</i> | <i>Entfernter Aufruf</i> |
|---|---|
| Aufrufer und Prozedur im selben Prozess ausgeführt | Aufrufer und Prozedur in unterschiedlichen nebenläufigen Prozessen |
| Aufrufer und Prozedur im selben Adressraum | Aufrufer und Prozedur in unterschiedlichen Adressräumen |
| Aufrufer und Prozedur in selben Hard- und Software-umgebung | Aufrufer und Prozedur in unterschiedlicher Hard- und Softwareumgebung |
| Aufrufer und Prozedur haben gleiche Lebensdauer | Aufrufer und Prozedur haben unterschiedliche Lebensdauer |
| Aufruf ist immer fehlerfrei | Aufruf ist fehlerbehaftet (Netz, Aufgerufener) |
| Nur Anwendungsfehler berücksichtigt | Zusätzlich Aufruffehler behandeln |

Komponenten beim RPC

- Anwendungsprozeduren: Eigentliche Arbeit
- Stubs: Ver- und Entpacken von Daten zum Transport
- Kommunikation: Transport von Daten

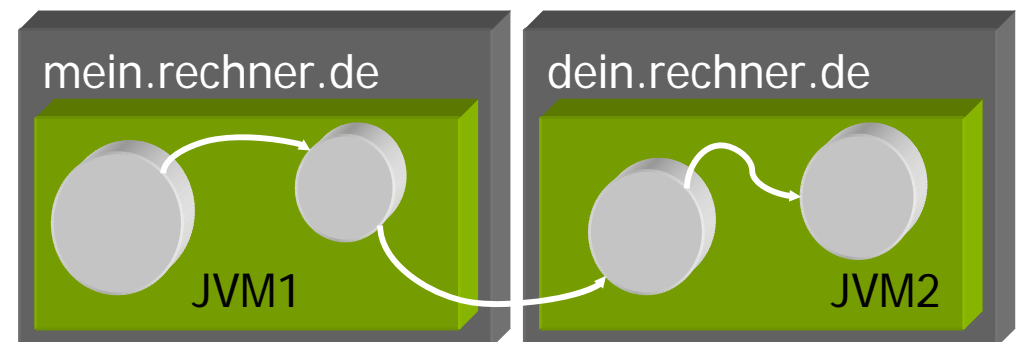
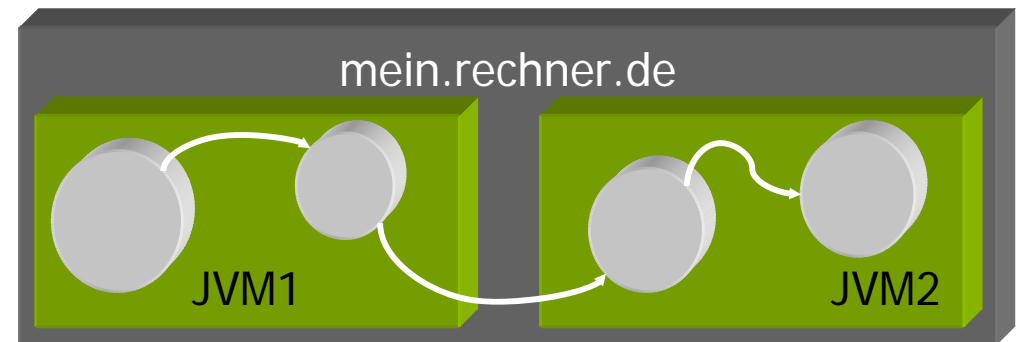
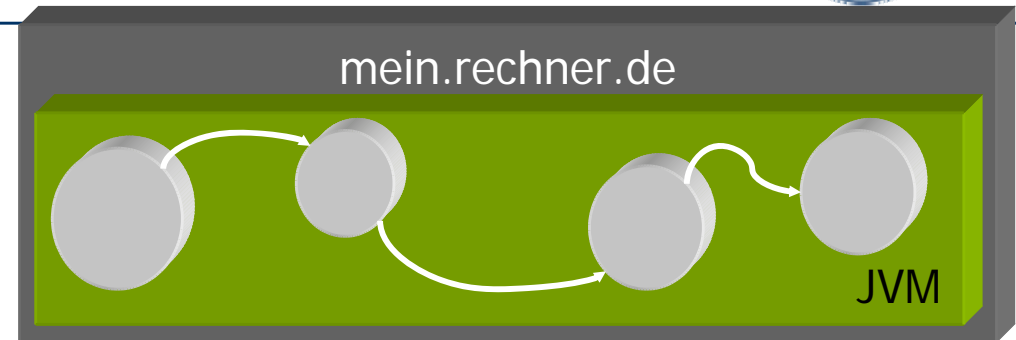




RMI

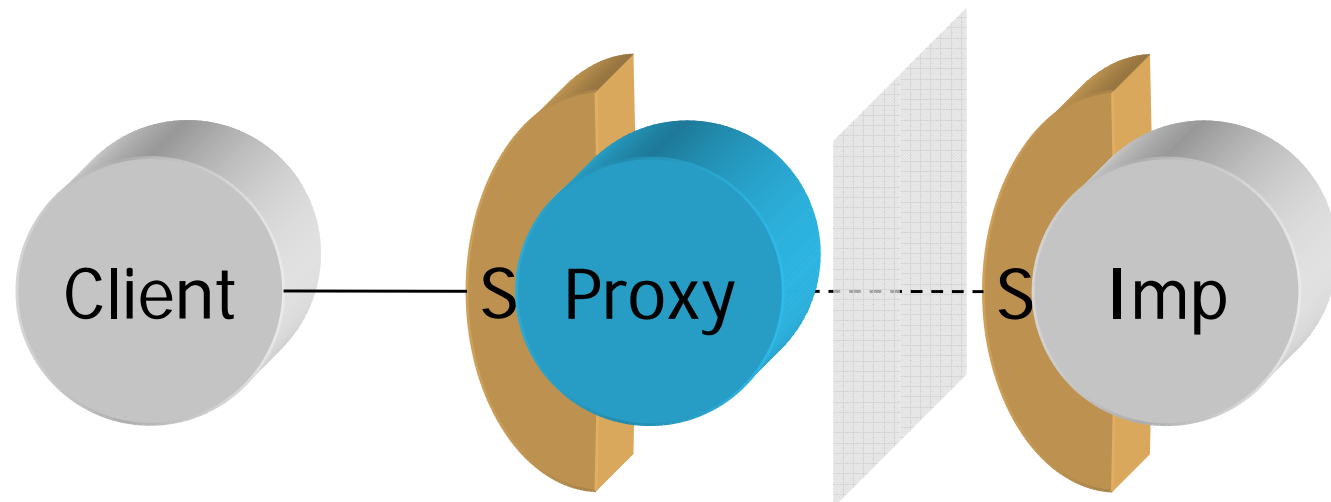
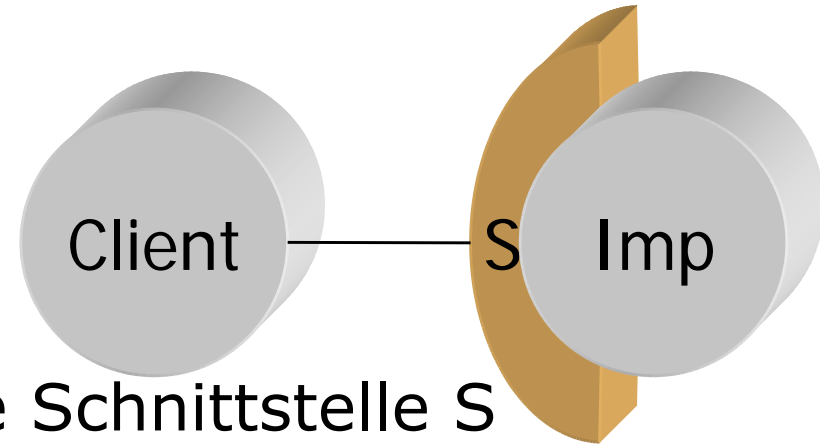
Lokale vs. verteilte Java-Programme

- Ein Java Programm arbeitet in einer virtuellen Java Maschine (JVM)
- Zwischen JVMs können mit *Remote Method Invocation* Methoden an Objekten aufgerufen werden
- JVMs können auf unterschiedlichen Internet-Rechnern laufen
- Sie müssen es aber nicht...



Schnittstellen

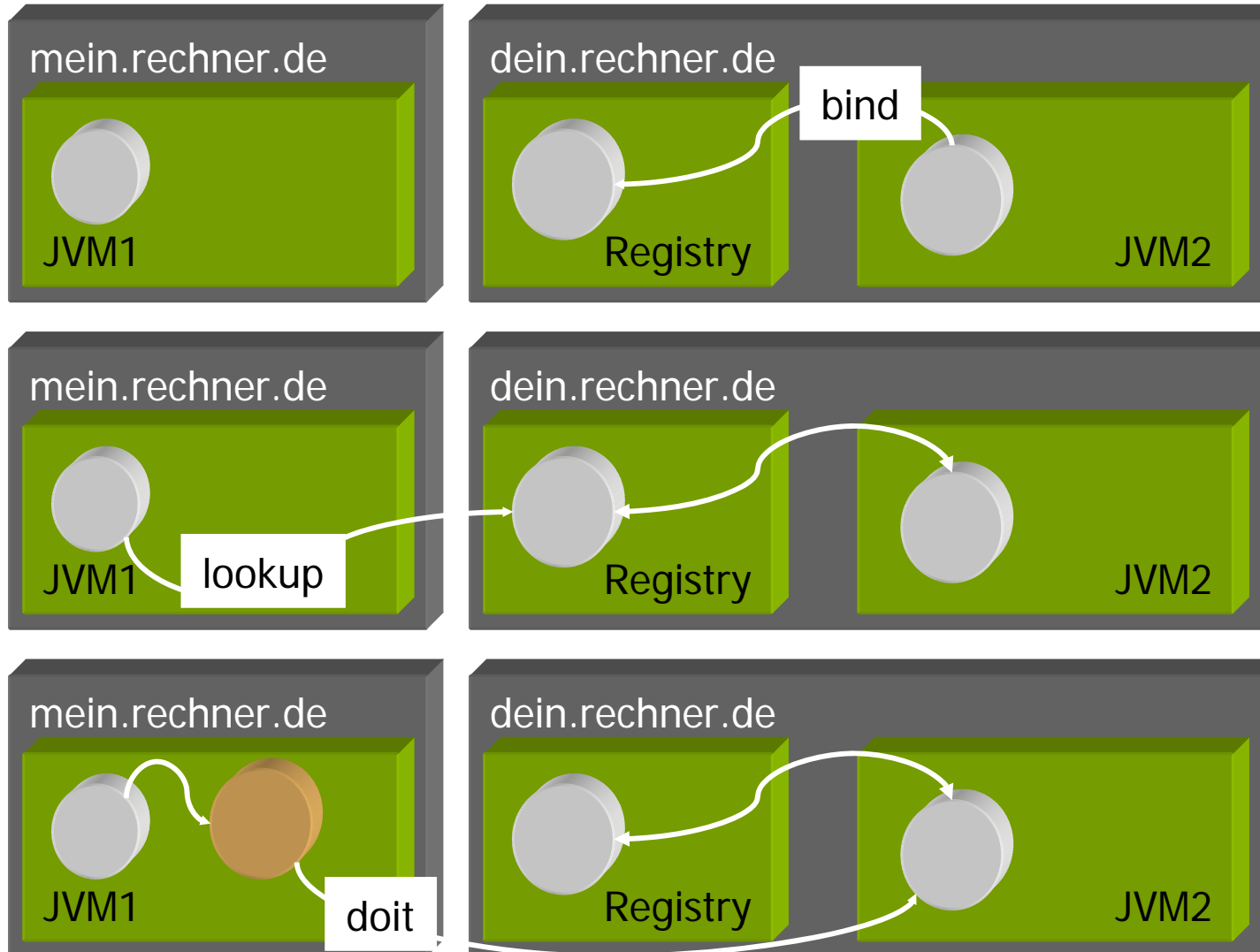
- Objektschnittstellen definieren
Methoden des Objekts
- Unterschiedliche Implementierungen für gleiche Schnittstelle S
- Modulschnittstellen des RPC werden auf Objektschnittstellen abgebildet
- Aufrufweiterleitung durch Stellvertreter (Proxy)



| <i>Lokales Objektmodell</i> | <i>Verteiltes Objektmodell</i> |
|--|--|
| Aufruf an <i>Objekten</i> | Aufruf an <i>Interfaces</i> |
| Parameter und Ergebnisse als <i>Referenzen</i> | Parameter und Ergebnisse als <i>Kopien</i> |
| Alle Objekte fallen <i>zusammen</i> aus | <i>Einzelne</i> Objekte fallen aus |
| <i>Keine</i> Fehlersemantik | <i>Komplizierte</i> Fehlersemantik (Referenzintegrität, Netzfehler, Sicherheit etc.) |
| ... | |

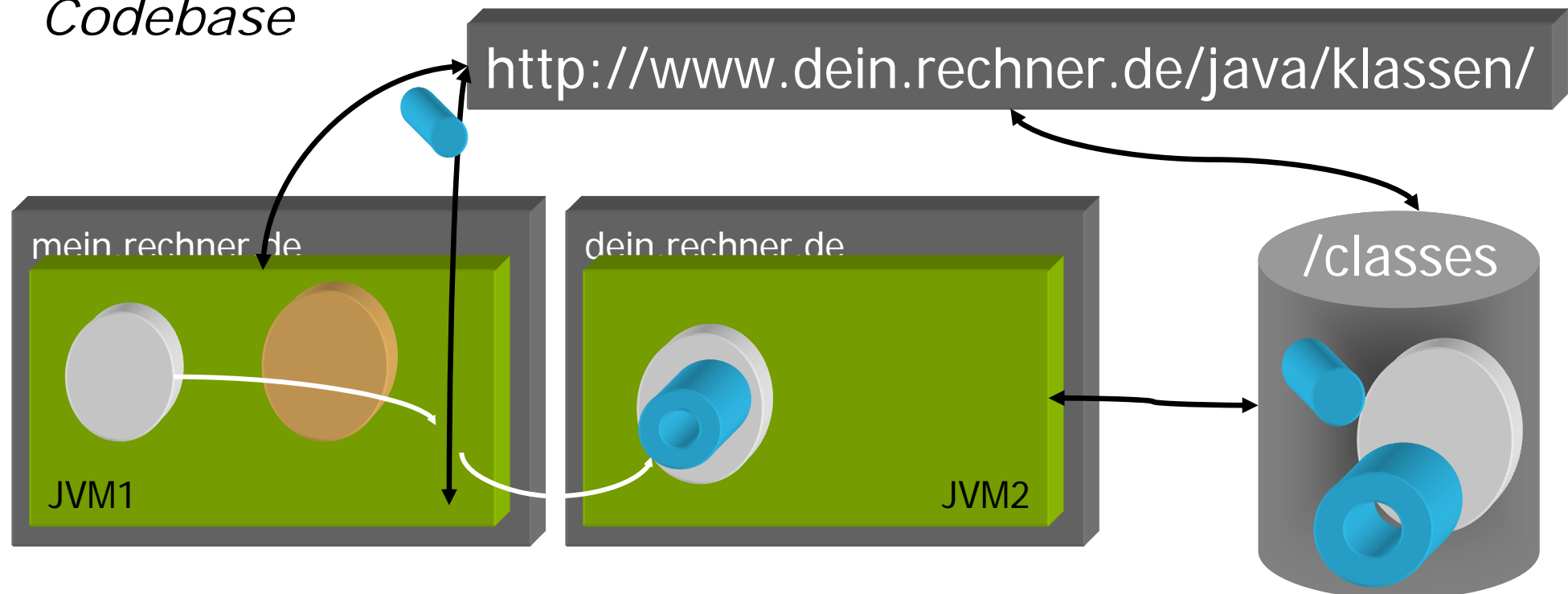
Referenzen auf entfernte Objekte

- *Registry* Objekt liefert Referenzen auf Objekte



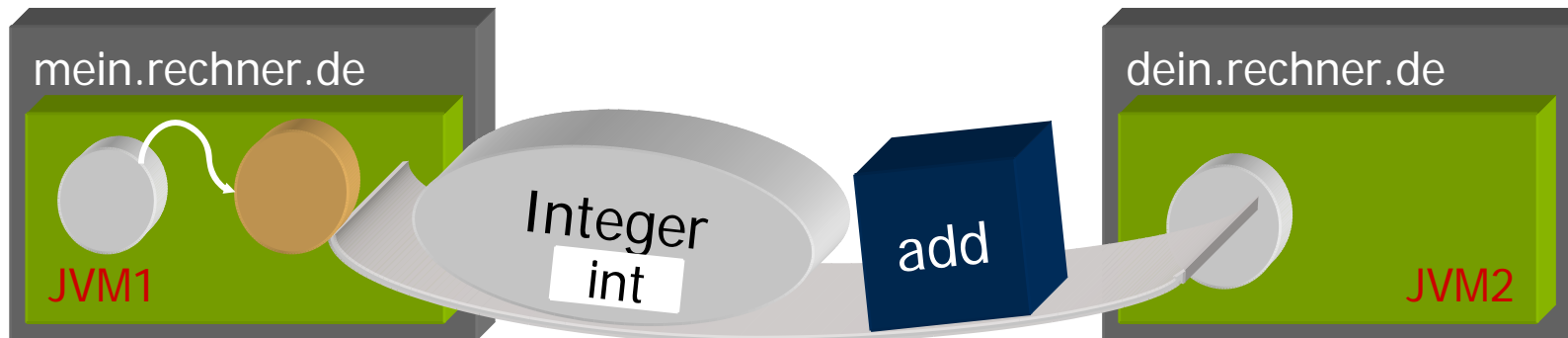
Nachladen über Web-Server

- Wenn JVM1 und JVM2 in getrennten Dateisystemen arbeiten nutzt CLASSPATH nichts
- ServerKlasse_stub.class muss über das Netz nachgeladen werden
- Dies geschieht durch Angabe einer Basis-URL, der *Codebase*



Serialisierung: Klassen nachladen

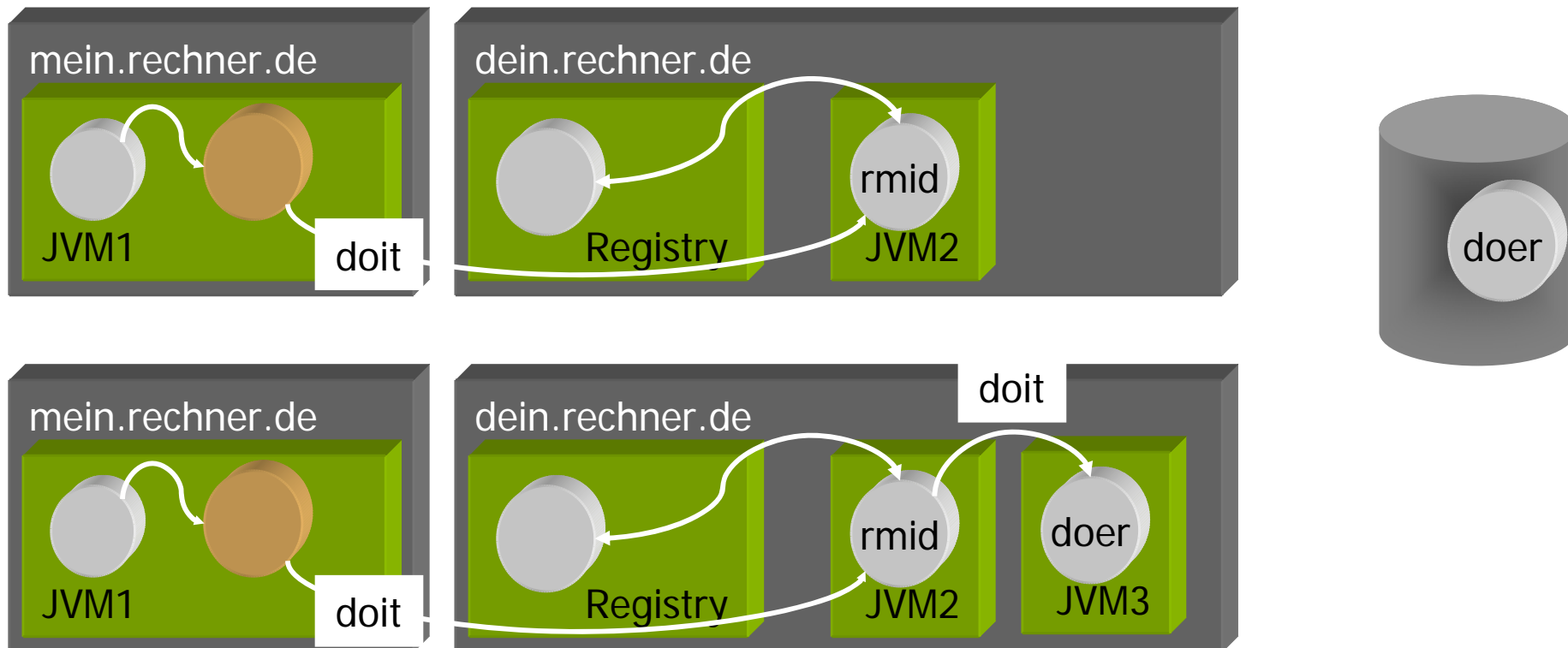
- Objekte = Daten und Verhalten
- Serialisierte Java-Objekte: Datenstrom + Klasse (konzeptionell)



- Klassen zu übermittelten Objekten nachladen, falls nicht
 - vorher nachgeladen
 - vorher schon vorhanden (java.* Klassen)
- Bedrohung durch Angreifer:
 - Bildet Unterklasse von Street, ändert dabei toString()
 - Erzeugt Objekt davon
 - Übergibt Objekt als Argument beim Methodenaufruf
 - Beim Aufruf von toString() dort wird geänderter Code ausgeführt
 - Mit allen Rechten des RMI Objekts

RMI Aktivierung

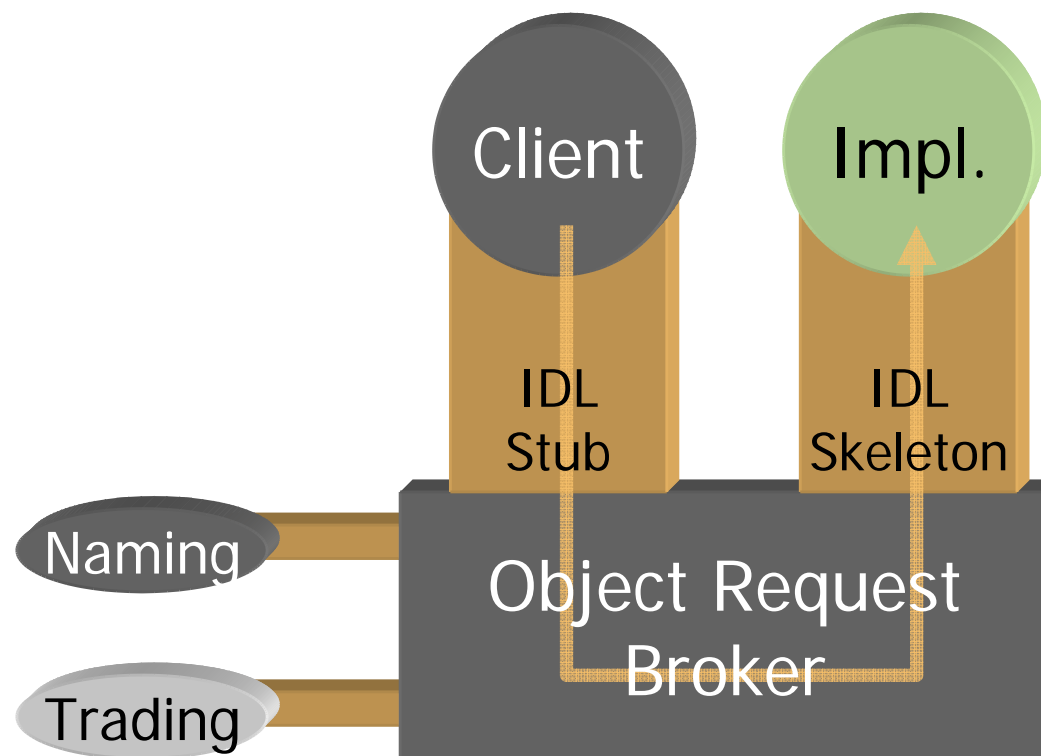
- Mit RMI Aktivierung werden Objekte in einer eigenen JVM beim Aufruf gestartet
- Zuständig für Aktivierung: rmid Programm
- rmid registriert sich für das Objekt und kann es aktivieren



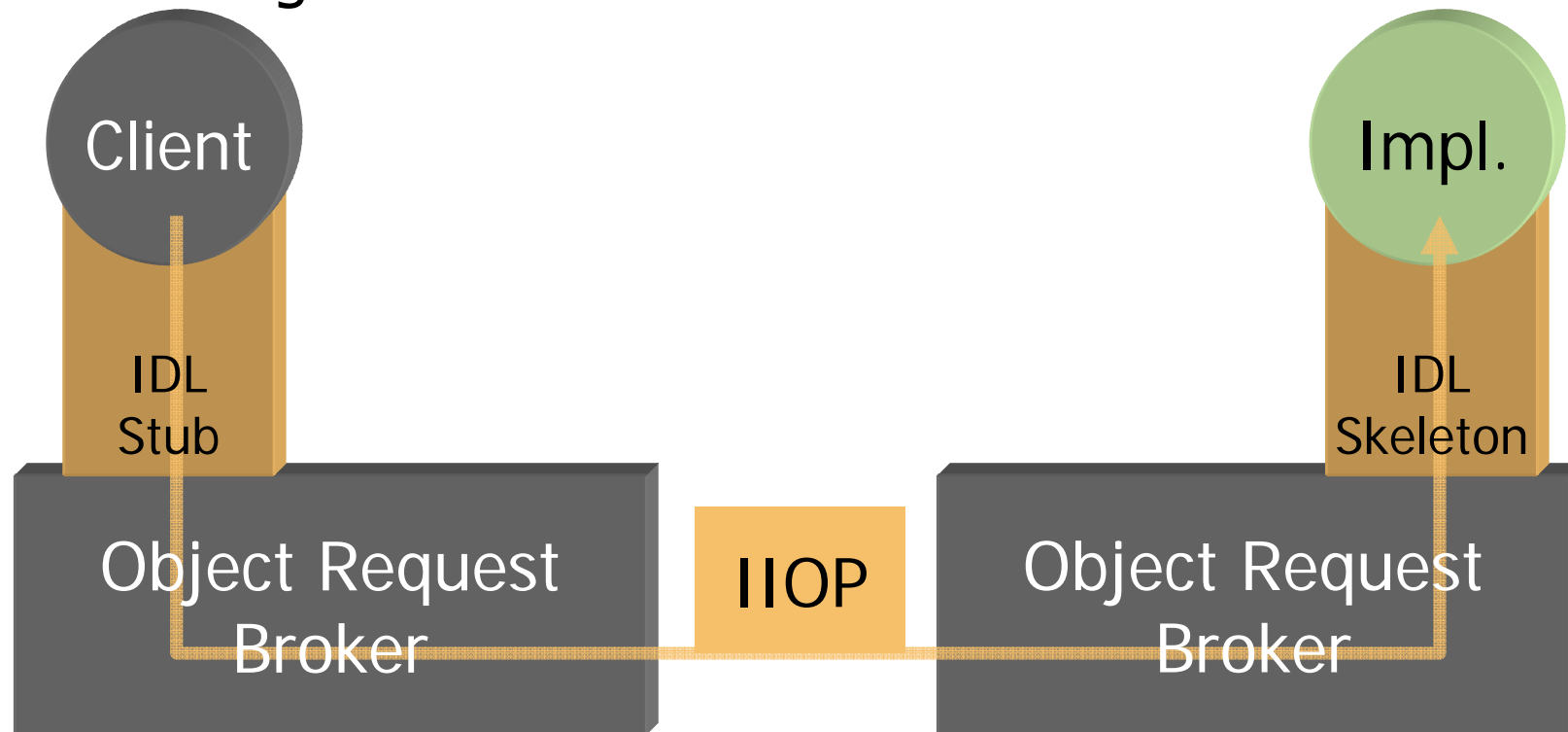


CORBA

- CORBA Objekte sind Bausteine von Anwendungen
- Sie haben eine typisierte Schnittstelle
- Von der Schnittstelle getrennte Implementierungen in unterschiedlichen Programmiersprachen
- Aufrufe werden durch Object Request Broker transportiert
- Weitere Dienste unterstützen



- Ortstransparenz wird durch Internet Inter-ORB Protocol IIOP erzeugt



- Interworking zwischen ORBs unterschiedlicher Hersteller möglich

| <i>RMI</i> | <i>CORBA</i> |
|---|---|
| Eine Sprache | Sprachunabhängig |
| keine Mappings nötig | Mappings notwendig |
| Objekte können als Argumente verwendet werden (Weil Klassen nachladbar sind) | Nur Referenzen auf Objekte möglich |
| Ein Laufzeitsystem | Unterschiedliche Laufzeitsysteme (ORBs) integriert |
| Proprietär (?) „Eigentümer“: Sun Prozess: JCP | Offener Standard (?) „Eigentümer“: OMG Prozess: OMG |

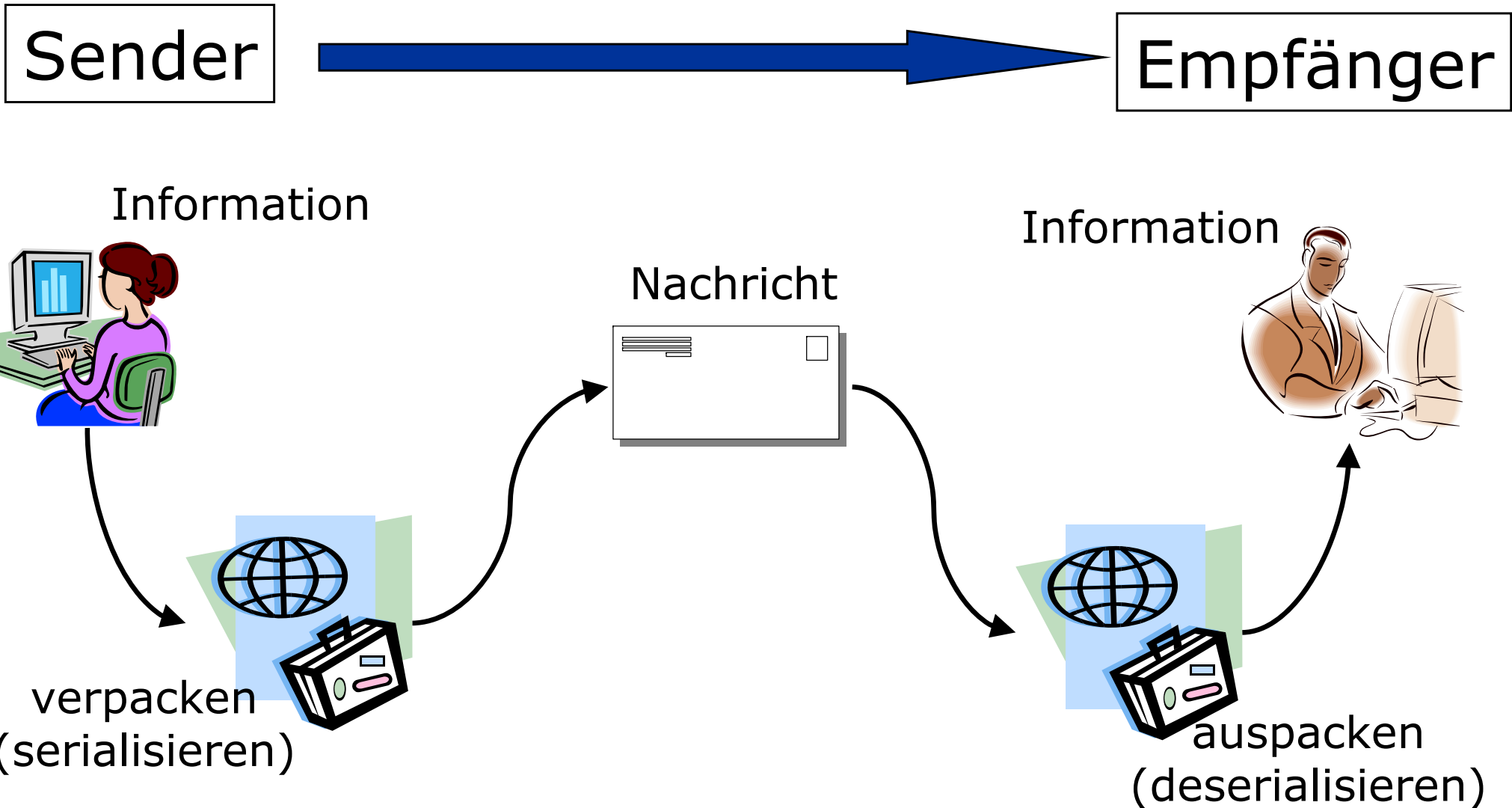
- In Java sind mehrere „Middlewares“ integriert (RMI, CORBA, EJB, ...)

- IDL ist eine Sprache zur Definition von Schnittstellen
- Sprachunabhängig
- Semantik der Typen definiert
- Zur Nutzung Mapping zu konkreter Sprache notwendig
- OMG definiert solche Mappings für unterschiedlichste Sprachen
- Abbildungsprobleme müssen zur Laufzeit durch Ausnahmen signalisiert werden



Web Services

Austausch einer SOAP-Nachricht



HTML

```
<html>  
  <head>
```

Zusatzinformationen

```
  </head>  
  <body>  
    Inhalt: Webseite  
  </body>  
</html>
```

SOAP

```
<Envelope>  
  <Header>
```

Zusatzinformationen

```
  </Header>  
  <Body>  
    Inhalt: XML-Daten  
  </Body>  
</Envelope>
```

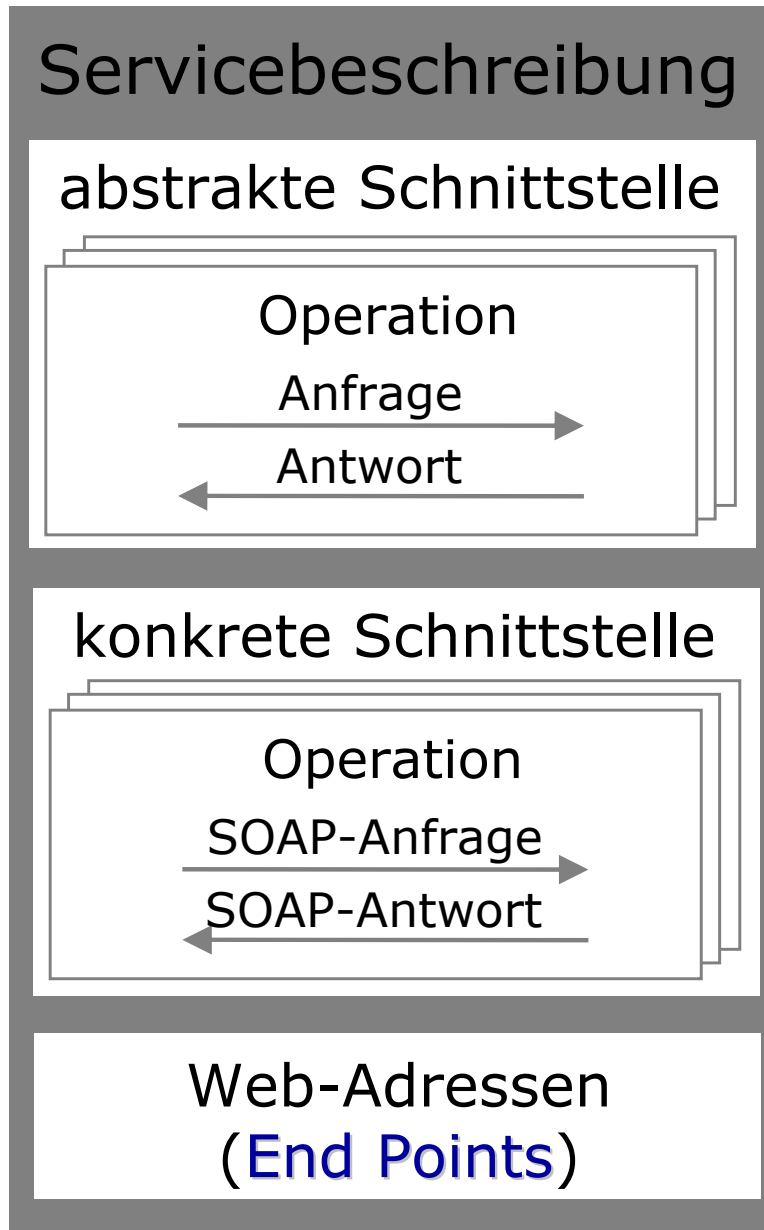
- XML-basierter W3C-Standard
 - SOAP 1.1: W3C Note von 2000
 - SOAP 1.2: W3C Recommendation von 2003
- seit SOAP 1.2: SOAP ≠ Simple Object Access

Procedure(Param-1="val-1",...,Param-n="val-n")

```
<env:Envelope ...>
  <env:Body>
    <m:Procedure xmlns:m="URI">
      <m:Parameter-1>val-1</m:Parameter-1>
      ...
      <m:Parameter-n>val-n</m:Parameter-n>
    </m:Procedure>
  </env:Body>
</env:Envelope>
```

- Name der Prozedur: Kind-Element von Body
- Eingangsparemeter: Kind-Elemente der Prozedur
- Beachte: Reihenfolge der Parameter relevant!
- Beachte: grundsätzlich **Call-by-Value!**

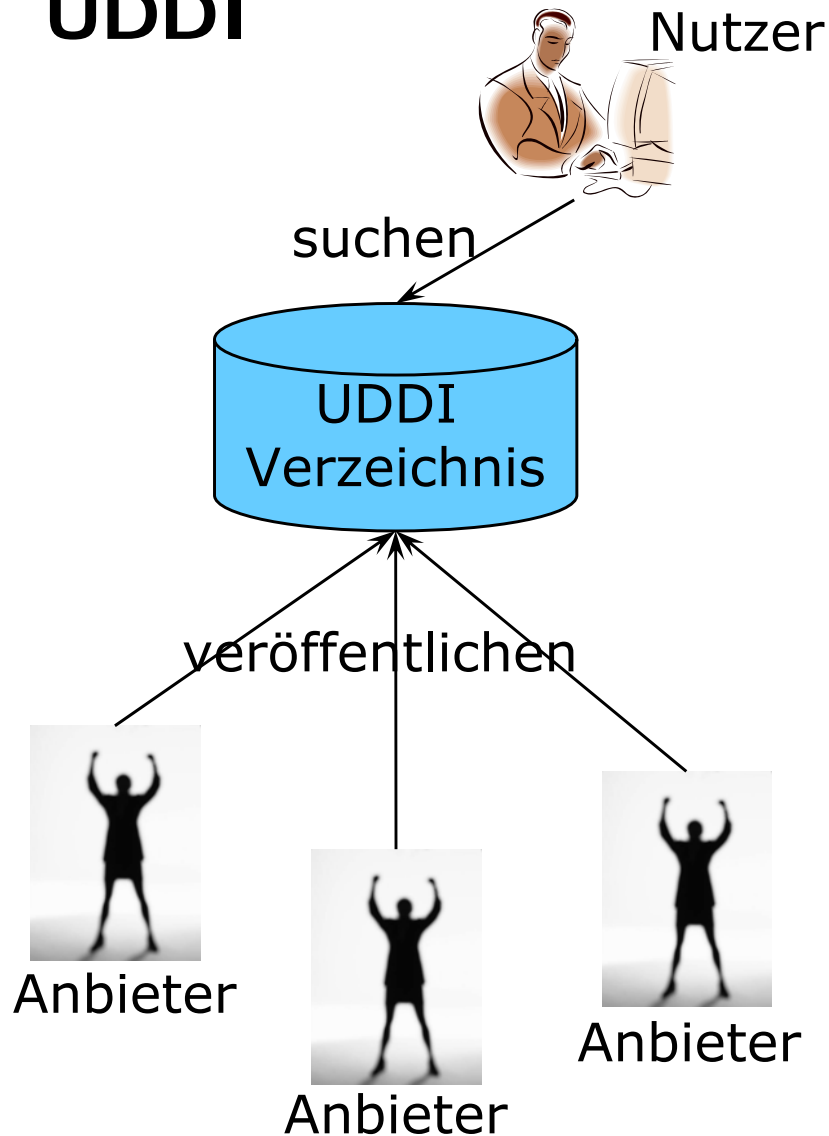
Web Services Description Language



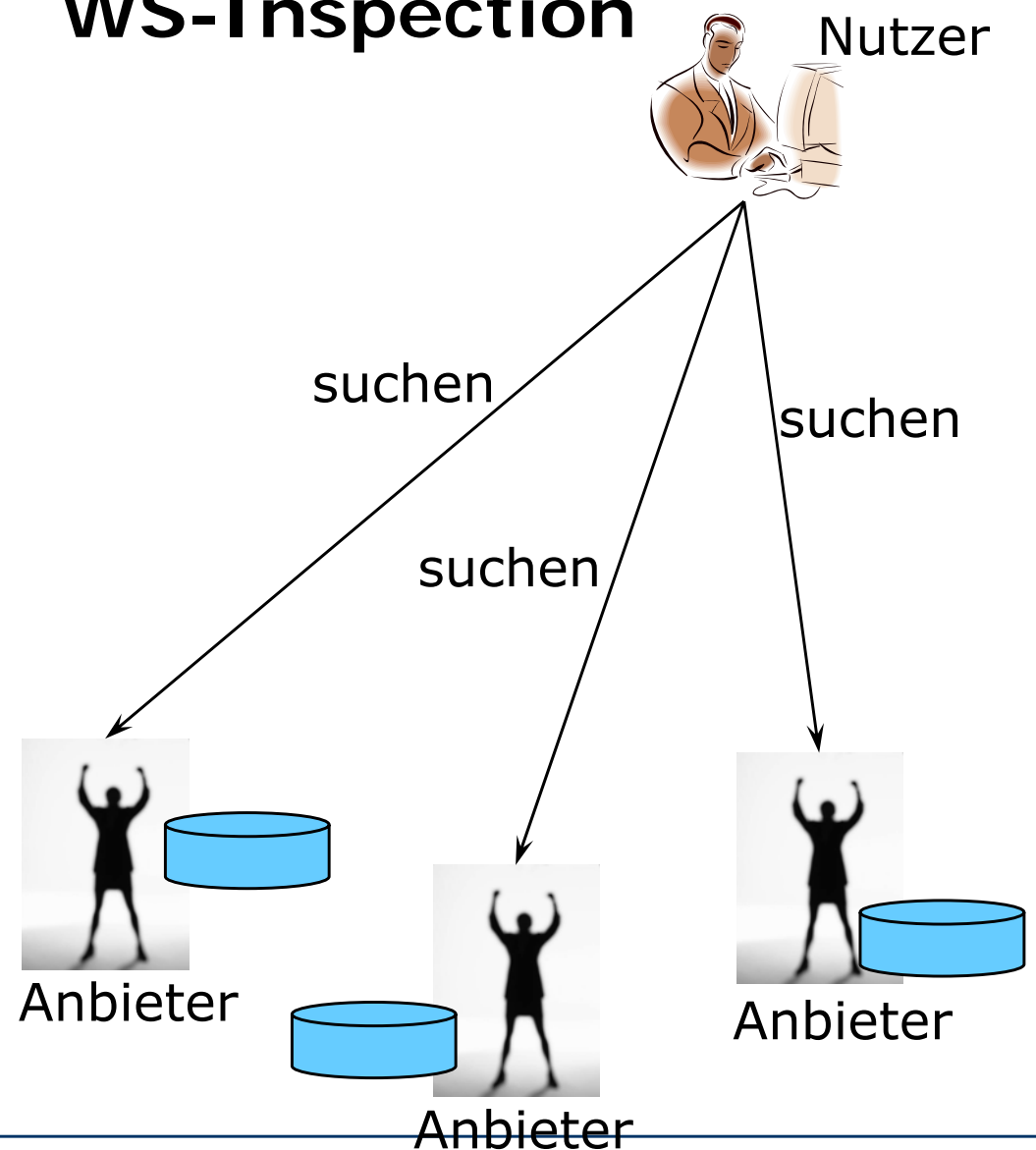
- beschreibt Interface (IDL)
- XML-basierter Standard
- W3C Note von 2001
- abstrakte Schnittstelle (port type)**
- Schnittstelle unabhängig von Nachrichtenformaten
- Beschreibung mit XML-Schema
- konkrete Schnittstelle (binding)**
- Abbildung der abstrakten Schnittstelle auf unterstützte Nachrichtenformate

UDDI vs. WS-Inspection

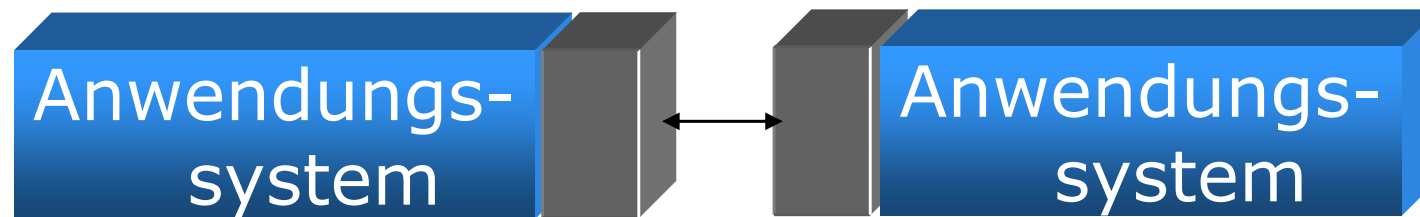
UDDI



WS-Inspection



- Anwendungssysteme durch standardisierte Schnittstelle erweitern



- Schnittstelle muss allgemein akzeptiert sein
- bei Web Services ist dies der Fall
 - ✓ SOAP
 - ✓ WSDL
 - ✓ Internet-Protokolle
- bei n Systemen:
statt n^2 Schnittstellen nur n Erweiterungen!

Vorteile aus Sicht von Amazon

- komplexe Anwendung aus relativ einfachen Web Services aufbauen
- aus komplexer Anwendung wiederum einfachen Web Service machen
- verteilte, skalierbare, robuste Architektur
- bestimmte Gruppe für Entwicklung und Betrieb eines Web Services verantwortlich