openHPI-Kurs: **Web-Technologien**

Willkommen in Kurswoche 5

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Übersicht Kurswoche 5 (1/2)

Zentrale Frage:

Komplexe Webanwendungen lassen sich nicht allein clientseitig realisieren. Welche Techniken gibt es auf der Server-Seite?

Lernmodule:

- Serverseitige Web-Programmierung
  - CGI-Schnittstelle und Alternativen
- Web-Frameworks und -Bibliotheken
  - Warum Frameworks?
  - Vergleich *Ruby on Rails* vs. *Django*
- …

Übersicht Kurswoche 5 (2/2)

Zentrale Frage:

Komplexe Webanwendungen lassen sich nicht allein clientseitig realisieren. Welche Techniken gibt es auf der Server-Seite?

Lernmodule:

- Persistenz-Layer und Datenbanken
  - Relationale Datenbanken
  - NoSQL-Datenbanken
  - Persistenz-Layer als Abstraktionsebene
- Webservices
  - Intermaschinelle Kommunikation auf Applikationsebene
- Tutorien:
  - Einführung in Ruby und Rails

openHPI: **Web-Technologien**

Serverseitige Web-Programmierung

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

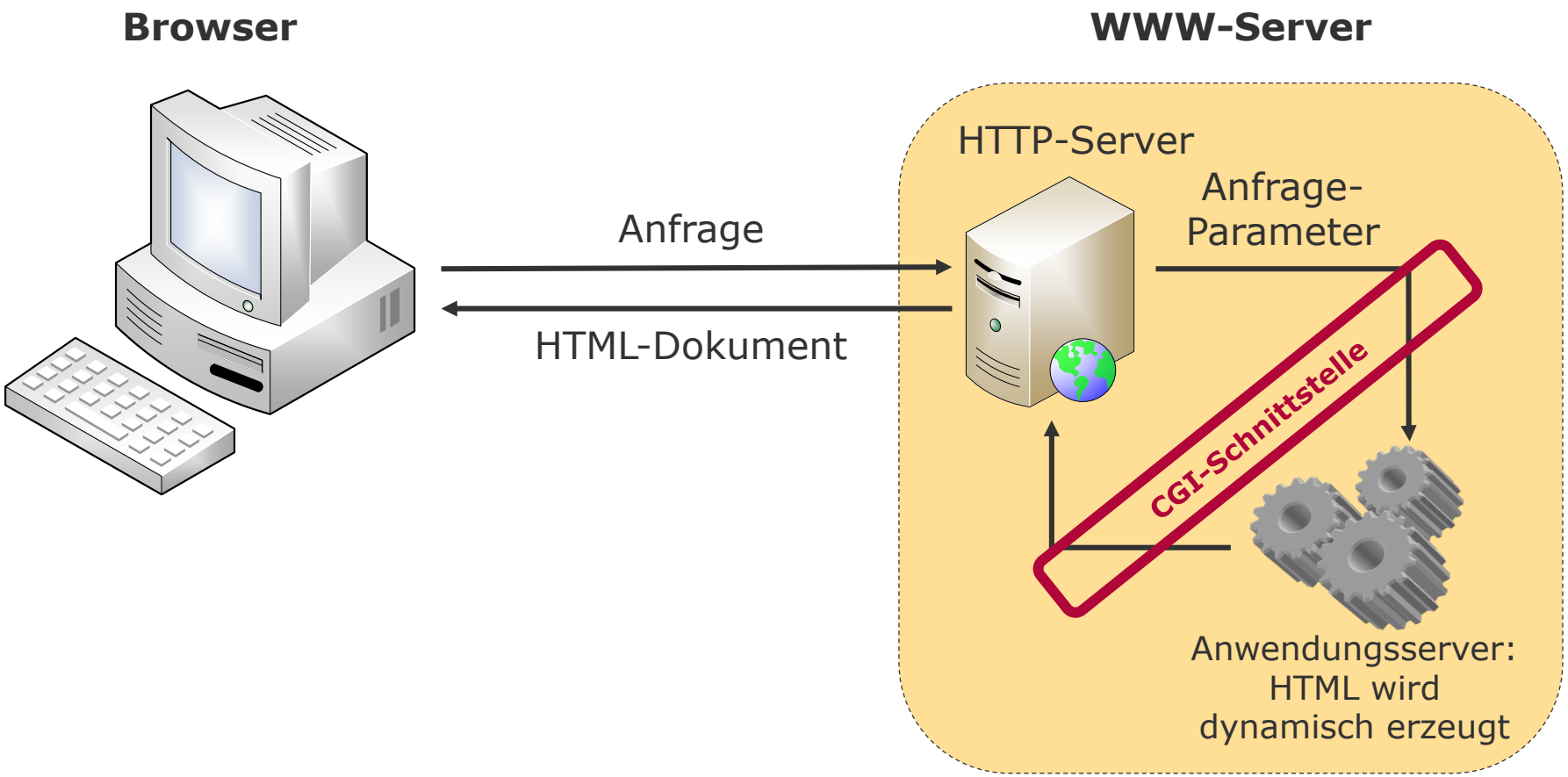
Universität Potsdam

Kommunikation zwischen HTTP- und Anwendungsserver

Standard-Schnittstelle für serverseitige Programme zur dynamischen Erzeugung von HTML-Dokumenten: **CGI** – **Common Gateway Interface**

- Zunächst nur Laufzeitumgebung für externe Programme
  - z.B. für C, C++
  - dann Einsatz von Perl-Skripten, Aufruf über Kommandozeileninterpreter
- Zur schnelleren Ausführung, Entwicklung von Skriptinterpretern als Module im Webserver, z.B.
  - für Apache: mod_perl, mod_python, mod_php
  - für Internet Information Services: ASP, ASP.NET
- Parallel dazu: Aufkommen von Java Application Servern (für Servlets)
- **Heute**: Neben Interpreter-Modulen auch neue CGI-Standards, z.B.
  - WSGI (Python), FastCGI, SCGI

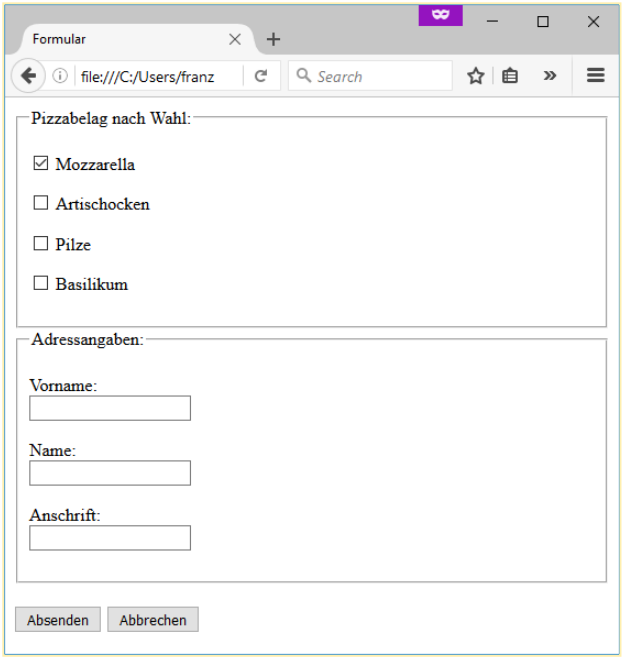
Erzeugung dynamischer Webseiten



CGI-Schnittstelle – Parameterübergabe (1/3)

Zur Steuerung der Anwendungsprogramme auf Seiten des WWW-Servers müssen Parameter übergeben werden können

- **Beispiel:** Ausfüllen und Abschicken eines HTML-Formulars




```
<form action="bestellen.php" method="POST">

<fieldset>
 <legend>Pizzabelag nach Wahl:</legend>
 <p>
 <input type="checkbox"
  name="zutat" value="mozarella" checked>
  Mozarella<br>
 <input type="ch…
  …
</fieldset>
</form>
```

CGI-Schnittstelle – Parameterübergabe (2/3)

GET

- Formulareingaben werden an die URL, die im action-Attribut angegeben ist, hinter „?“ als **Key-Value-Paare** angehängt
- HTTP-Server übergibt die Formulardaten an das über die angegebene URL aufgerufene CGI-Programm in der Umgebungsvariable **QUERY_STRING**
  - Beispiel: Google-Suche: https://www.google.de/?q=deutsche+online-kurse
- Umgebungsvariablen:
  - Parameter, die beim Start an Programm übergeben werden
  - sind im gesamten Kontext des Programms definiert

CGI-Schnittstelle – Parameterübergabe (3/3)

POST, PUT, PATCH, DELETE

- Es wird ein HTTP-Request an die im action-Attribut angegebene URL initiiert
- Formulardaten werden im Body des HTTP-Requests übergeben
- Body des HTTP-Requests wird dem in der URL aufgerufenen CGI-Programm über die Standard-Eingabe zur Verfügung gestellt
  - Beispiel: Verfassen eines Beitrags in einem Online-Forum

CGI-Schnittstelle – Umgebungsvariable (1/2)

Umgebungsvariablen des HTTP-Servers

- Server belegt vor Aufruf des durch den Client initiierten CGI-Programms bestimmte Umgebungsvariablen, z.B.
  - CONTENT_LENGTH, SERVER_NAME, REQUEST_METHOD
  - PATH_INFO, SCRIPT_NAME, QUERY_STRING, …
- Per CGI kann auf Umgebungsvariablen zugegriffen werden, z.B. für Auswertung
  - Beispiele - Portnummer:
    - Ruby: `ENV["SERVER_PORT"]`
    - PHP: `$_SERVER["SERVER_PORT"]`

CGI-Schnittstelle - Umgebungsvariable (2/2)

Umgebungsvariablen des HTTP-Servers

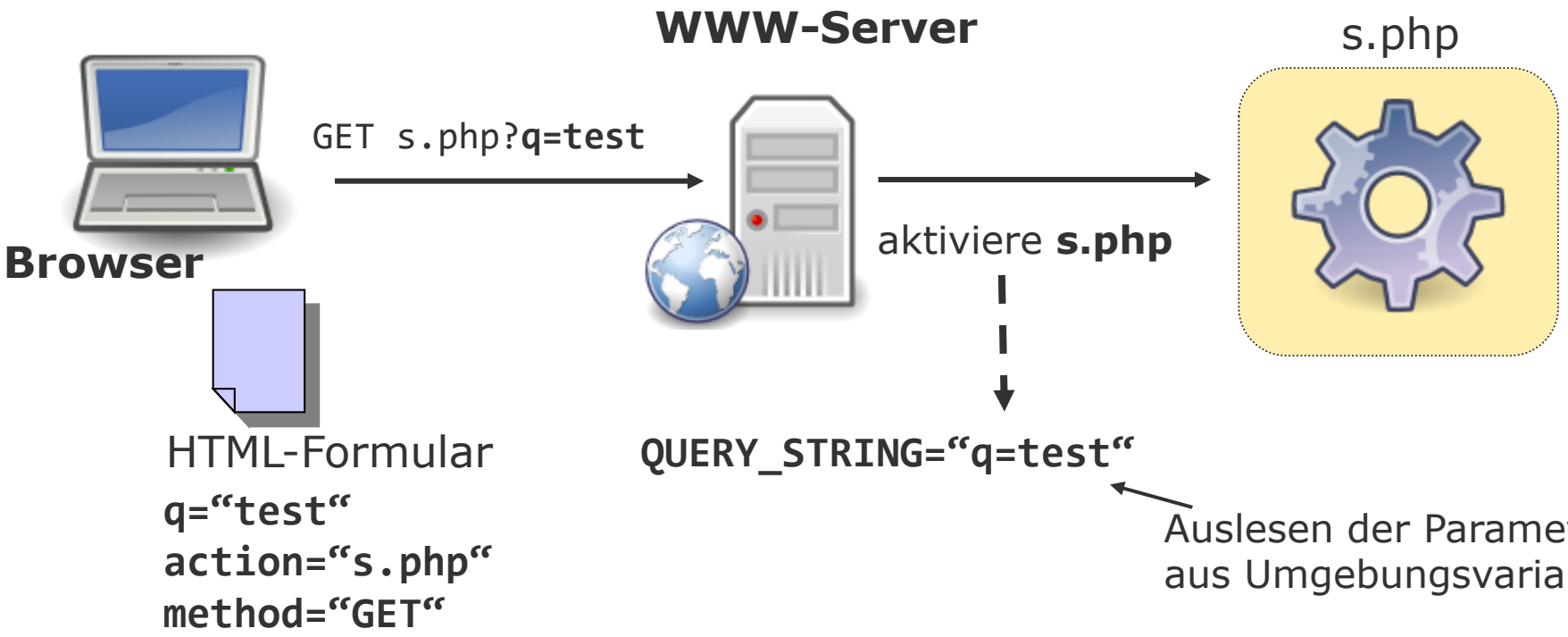
- Auch auf andere Teile des HTTP-Requests, z.B. per GET oder POST gesendete Parameter, kann zugegriffen werden
  - Beispiel PHP: **`<?php echo $_POST['zutat'] ?>`**

```
<form action="bestellen.php" method="POST">

<fieldset>
 <legend>Pizzabelag nach Wahl:</legend>
 <p>
 <input type="checkbox"
  name="zutat" value="mozarella" checked>
  Mozarella<br>
 <input type="ch…
  …
</fieldset>
</form>
```

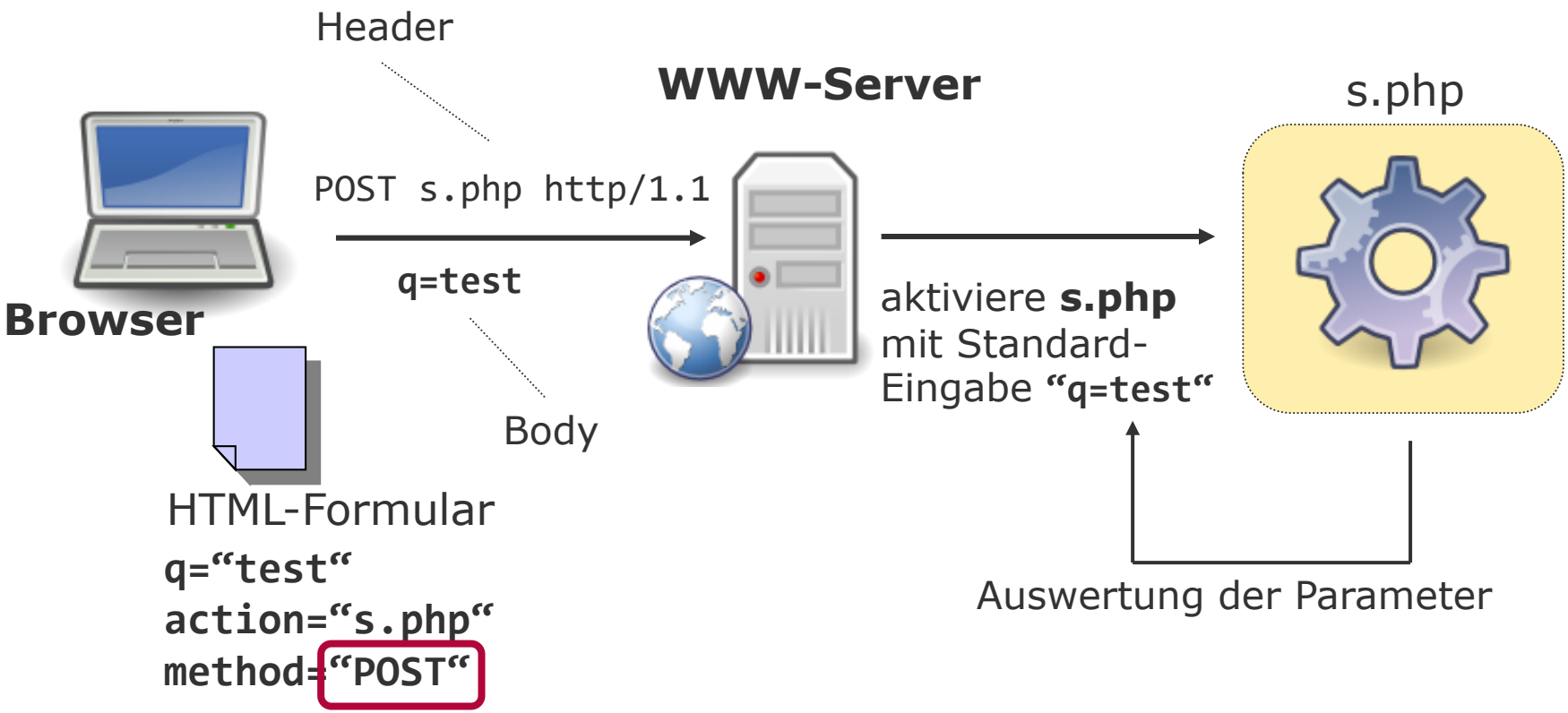
CGI-Schnittstelle – Parameterübergabe (1/2)

Parameterübergabe mit GET



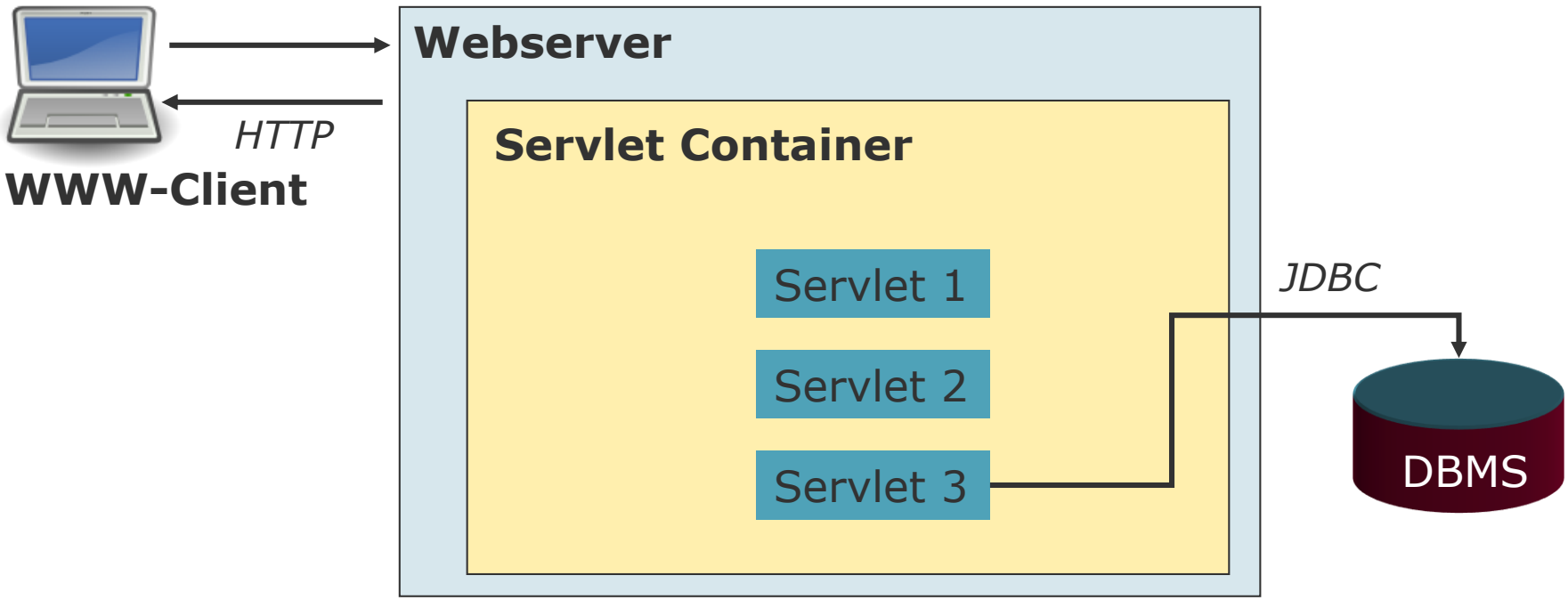
CGI-Schnittstelle – Parameterübergabe (2/2)

Parameterübergabe mit POST



Ausführungsumgebung

- Webserver + Servlet-Container, z.B. Apache + Tomcat
- Application Server, z.B. Oracle Glassfish, WildFly



Serverseitiges Java (2/2)

- **JSP Java Server Pages**
  - ermöglichen Einbinden von Template-Texten in HTML-Dokumente
  - Verwendung von Taglibs, z.B. JSTL (Java Standard Tag Library)
- **Servlets**
  - auf Application Server / im Servlet-Container ausgeführte Java-Programme
  - reagieren auf von Webseite erstellten HTTP-Request mittels einem HTTP-Response
- **JSF Java Server Faces**
  - Framework zur Erstellung von Web-Applikationen
- **Facelets**
  - Ersetzen JSP im Zusammenspiel mit JSF

Webserver-Module

- **CGI**: Webserver-Prozess kommuniziert mit separatem Interpreter-Prozess
  - schlechte Performance
  - gute Sicherheit: Prozesse sind voneinander isoliert
- **Webserver-Module**: mod_perl, mod_php, mod_python …
  - Interpreter-Prozess eingebettet in den Webserver-Prozess
  - Gute Performance für Requests, die dynamisch Code ausführen müssen
    - Aber: Zusätzlicher Speicherbedarf auch für Requests auf statische Ressourcen
  - Sicherheit der Skripte (und der Interpreter) beeinträchtigt die Sicherheit des Webserver-Prozesses

FastCGI

- Entwickelt als Nachfolger von CGI
- Übernimmt das Sicherheitskonzept von CGI; versucht, die Performance zu verbessern
  - Das Starten eines neuen Prozesses bei jedem Request fällt weg
  - (Beliebig viele) Anwendungsprozesse können mehrere Requests hintereinander verarbeiten: Overhead für Prozessstart fällt weg
- Kommunikation über **Sockets**
  - Vorteil: Anwendungsprozess kann auch auf separatem Server laufen
- Statische Inhalte werden direkt vom Web-Server ausgeliefert

openHPI: **Web-Technologien**

Web-Frameworks

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Entwicklung serverseitiger Web-Programme

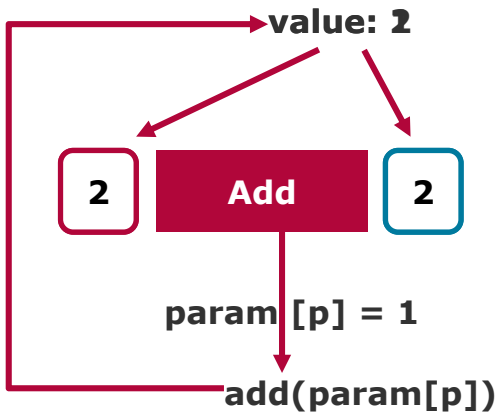
- **CGI**: HTML-Dokument wird vom CGI-Programm generiert, z.B.
  - `print("<h1>Titel</h1>");`
- **Serverseitiges Scripting**: Spezielles Skript-Markup innerhalb von (sonst statischen) HTML-Dateien, z.B.

```
<ul>
  <?php foreach ($items as $item): ?>
    <li><?php echo $item->description ?></li>
  <?php endforeach ?>
</ul>
```

- **Problem:** Vermischung verschiedener Aufgabenbereiche in selber Datei
  - Unübersichtlicher, schwer wartbarer Code
  - Designer und Programmierer arbeiten an derselben Datei
    ➔ Konfliktpotential

Lösungsansätze:

- **Template-Engines**:
  - Trennung von Programm-Logik und Darstellung
- **Moderne Web-Frameworks**:
  - weitergehende Aufsplitterung von Verantwortlichkeiten im Programm
  - oft Anwendung von MVC-(ähnlichen)-Entwurfsmustern.
    MVC: **Model-View-Controller**
    - Model: Datenstruktur und Businesslogik
    - View: Darstellung
    - Controller: verbindet Model und View



Warum Frameworks? (1/2)

Frameworks …

- Bieten Grundgerüst mit architektonischen Vorgaben für die Entwicklung von Anwendungen („framework": englisch für *Gerüst*)
  - gute und bewährte Abstraktionen
- Stellen Lösungen bereit für Probleme, die immer wieder gelöst werden müssen
- Enthalten Bibliotheken für typische Anwendungsfälle
- Erlauben beschleunigtes Programmieren
  - Code-Generatoren
  - Konventionen

Warum Frameworks? (2/2)

Web-Frameworks …

- Bieten Routing Pfade, die mit dem verarbeitenden *Controller* verbinden, z.B.
  - /courses/webtech2017 → **`CoursesController`** mit ID `webtech2017`
- Bieten Persistenz-Layer für den Datenbank-Zugriff und für die Abbildung der Anwendungslogik
- Enthalten Template-Engines für die Generierung von HTML
- Gehen mit HTTP um, Request/Response-Zyklen, Headers, Middleware, …
- Enthalten Bibliotheken für
  - Mail-Versand
  - Caching
  - Verarbeitung von langwierigen Prozessen im Hintergrund (*Queues*)
  - …

Frameworks – Vorteile

Vorteile

- Frameworks ermöglichen **einfachere**, **schnellere** Programmierung
- Bauen auf der Erfahrung und den Tools anderer auf
- Bieten Lösungen für häufige Probleme
  - meist effizient, stabil und **sicher**
- Architektur wird vorgegeben
- Open-Source: Die meisten Frameworks sind frei verfügbar
  - viele Augen sehen mehr Fehler
  - Verbesserungen sind gern gesehen

Frameworks – Nachteile

Nachteile

- Frameworks sind nicht für die Ewigkeit und ändern sich oft
  - macht oft Veränderungen am eigenen Code nötig
- Eigener Programmcode oft an den Code des Frameworks **gekoppelt**
- Frameworks lösen viele Probleme, aber nicht alle
  - wer außerhalb der vorgegebenen Grenzen gerät, muss oft „gegen das Framework kämpfen“
- Frameworks und „Magie“:
  - ohne Kenntnis der Konventionen des Frameworks ist Code oft schwer nachvollziehbar (insbesondere für Neueinsteiger)
  - (insb. Ruby on Rails:) Was kann die Programmiersprache? Was ist vom Framework?

Frameworks – Empfehlungen

Empfehlungen

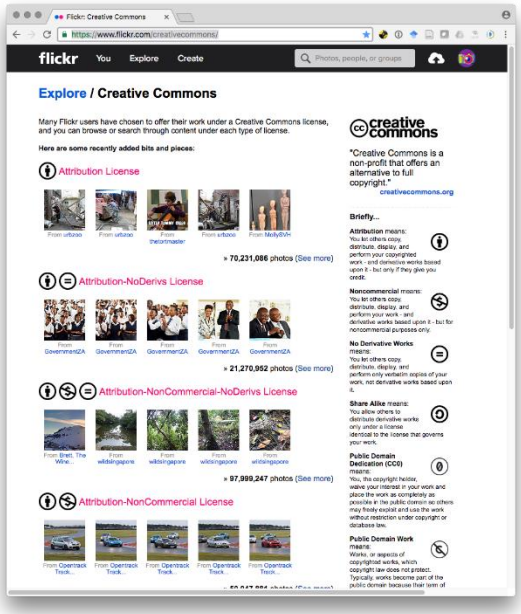
- Erst die Programmiersprache lernen, dann ein Framework!
  - Magie des Frameworks von Funktionen der Sprache unterscheiden
  - ➢ **Probleme kennen, die das Framework löst, um dessen Lösungen wertschätzen zu können**
- Frameworks garantieren keinen sauberen Code
  - Sinn und Zweck der Architekturmuster kennen lernen
- Vor- und Nachteile abwägen
  - Frameworks beschleunigen die Entwicklung (insb. zu Beginn), aber erschweren Quereinstieg
  - Entscheidung für sehr lange Zeit: benutzt man ein Framework, wird man es nicht sehr schnell wieder los

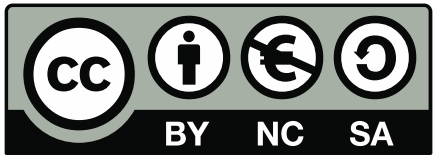
Frameworks – Lizenzen (1/2)

- Open-Source-Software: Freie Verwendung unter durch **Lizenzen** geregelten Bedingungen
- **Flexible Lizenzen**
  - Beispiele: MIT, BSD, Apache
  - normalerweise untereinander kompatibel
- **Restriktive Lizenzen**
  - Beispiel: GPL
  - Abgeleitete Software muss ebenfalls unter GPL lizensiert werden
  - Verwendung von GPL-Bibliotheken kann dazu führen, dass eigener Code veröffentlicht wird
  - **AGPL**: Schließt Schlupfloch für serverseitige Applikationen

Frameworks – Lizenzen (2/2)

- Open-Source-Software-Lizenzen sind oft schwer zu verstehen
- Einfacherer Ansatz: Creative Commons:
  - https://creativecommons.org/
- Einsatzgebiete
  - weniger im Softwarebereich
  - weit verbreitet im Grafikbereich
  - Beispiel: https://www.flickr.com/creativecommons/
- Elemente
  - Attribution, Share-Alike, No-Derivs, Non-Commercial
  - beliebig kombinierbar
  - einfache Kennzeichnung





openHPI: **Web-Technologien**

Ausgewählte Web-Frameworks

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

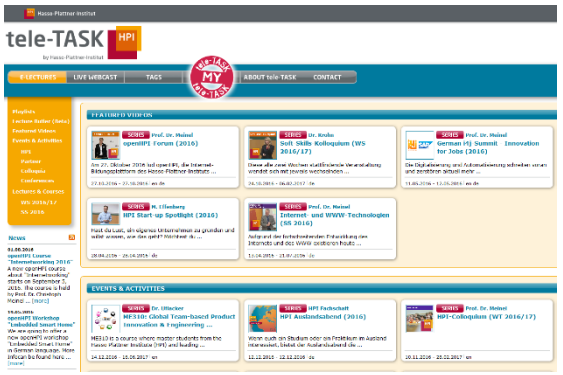
Ausgewählte Web-Frameworks

Große Web-Projekte basieren auf Web-Frameworks:

- **openHPI** benutzt **Ruby on Rails**, das wohl bekanntestes Ruby-Web-Framework)
- **tele-TASK** baut auf **Django**, ein Python-basiertes Framework

Populäre Web-Frameworks

- Python: Django, Flask
- Ruby: Ruby on Rails, Sinatra
- PHP: Zend, Symfony, Laravel
- Java: Grails, Play, JavaServer Faces (JSF)
- .NET: ASP.NET
- JavaScript: Meteor, Express



Web-Framework: Django

Django – https://www.djangoproject.com

- Beliebtes Python-Framework
- Lizenz: BSD
- Seit 2005, aktuelle Version: 1.10
- Schnell von Idee zur Umsetzung
  - Code-Generatoren
  - Automatisch generierte Administrations-Oberfläche
- Viele eingebaute Features
  - Authentifizierung
  - HTML-Formular-Helper
  - Unterstützung für HTTP-Cache
- „Don't Repeat Yourself" (DRY)
  - Redundanz vermeiden, exakt einen Ort (im Quellcode) für jedes Konzept

Web-Framework: Ruby on Rails

Ruby on Rails (kurz **Rails**)– http://rubyonrails.org

- Wohl berühmtestes Ruby-Web-Framework
- Lizenz: MIT
- Seit 2005, aktuelle Version: 5.0
- Vorreiter für moderne Web-Frameworks in vielen Belangen
  - Model-View-Controller
  - Active Record: einfacher Datenbankzugriff durch Abbilden von Datenbank-Tabellen und -Zeilen in Klassen und Objekten
- Florierendes Ökosystem mit zahlreichen Erweiterungen (Gems)
  - Manche Gems werden vom De-Facto-Standard zum Teil des Frameworks
- „Convention over Configuration“

Web-Frameworks – „Convention over Configuration“

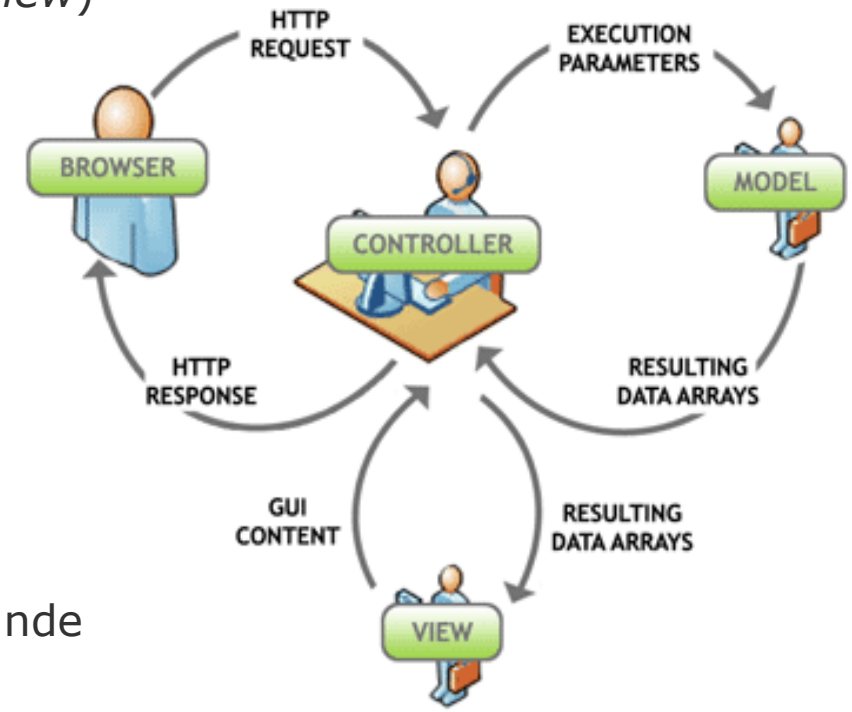
Wichtiges **Software-Design-Paradigma,** essentiell für Rails

- Entwickler haben sich an **feste Konventionen** zu halten, z.B.
  - Primärschlüssel von Datenbank-Tabellen heißen „`id`“ und sind Integer
  - Verwaltung von `Account`-Models geschieht im `AccountsController`
  - Klasse `CoursesController` liegt in app/controllers/courses_controller.rb
- **Vorteile:**
  - Zügige Entwicklung, weniger Entscheidungen, weniger *expliziter* Code
  - Für die meisten Probleme gibt es eine „richtige“ Lösung
  - Beschleunigter Projekteinstieg, wenn Konventionen bekannt sind
- **Nachteile:**
  - Konventionen müssen bekannt sein
  - Andere Herangehensweisen passen oft nicht gut ins Framework

Software-Design-Muster

- **Grundidee**: Trennung von Datenhaltung und Businesslogik (*Model*), Kontroll- (*Controller*) und Anzeigelogik (*View*)
- **Ziele:**
  - Wiederverwendbarkeit einzelner Teile
  - Änderungen in einer Ebene sind unabhängig von anderen Ebenen, z.B.
    - von Redesign im Template (View) bleibt Controller unangetastet

  Verwendung:
  - liegt fast allen Web-Frameworks zugrunde
  - unterschiedliche Varianten



openHPI: **Web-Technologien**

Persistenz-Layer und Datenbanken

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

HTTP ist zustandsloses Protokoll

- Aber: Applikationen befinden sich immer in einem bestimmten Zustand
  ➔ Daten des Zustands müssen vorgehalten werden
  - Clientseitig: Storage-Mechanismen beim Browser oder im Arbeitsspeicher
  - Serverseitig: Datenbanken und Sessions

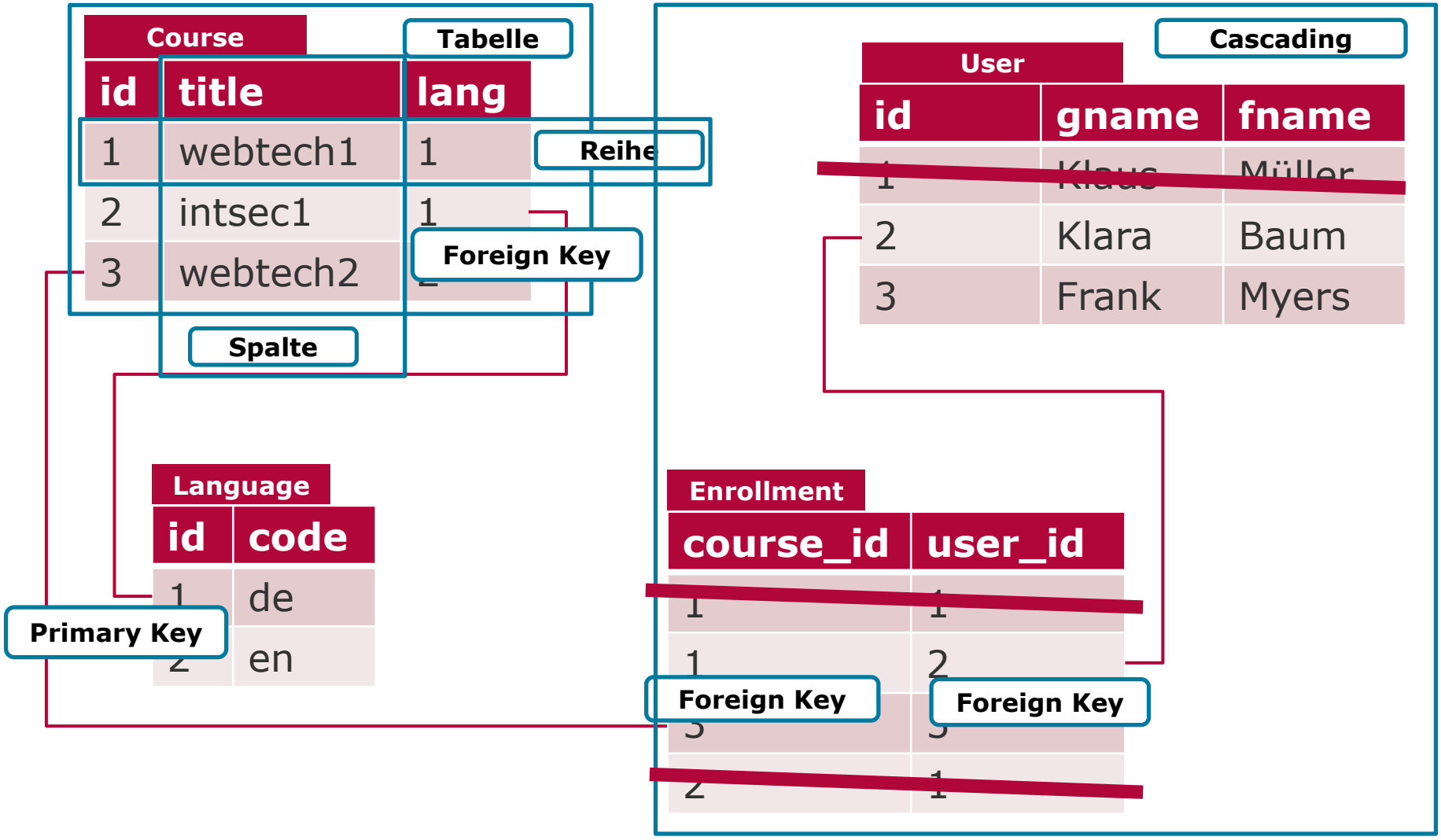
Unterschiedliche Arten von Datenbanken:

- Relationale Datenbanken, z.B. MySQL, PostgreSQL, Oracle, MSQL, ...
- NoSQL-Datenbanken
  - Graph Stores, z.B. Neo4J, Giraph, ...
  - Document Stores, z.B. MongoDB, Elastic, OrientDB, CouchDB...
  - Key-Value Stores, Redis, BerkeleyDB, ...
  - Column Stores, Hbase, Cassandra, Hadoop...
  - ...

Relationale Datenbanken (1/3)

- **Tabellenstruktur**
  - Reihe – Repräsentiert einen Datensatz (z.B. User)
  - Spalte – ein Attribut der Datensätze, z.B. Name, Vorname, E-Mail, …
  - Tabelle – Menge von Datensätzen mit denselben Attributen
- Jeder Datensatz in Tabelle hat eindeutigen Schlüssel: **Primary Key**
  - ermöglicht eindeutige Identifikation des Datensatzes
  - kann aus mehreren Attributen zusammengesetzt werden
  - kann in anderen Tabellen referenziert werden, um Bezüge zwischen Tabellen herzustellen (Foreign Key)
- **Cascading**
  - Löschen aller abhängigen Daten (in anderen Tabellen), wenn Datensatz gelöscht wird

Relationale Datenbanken (2/3)

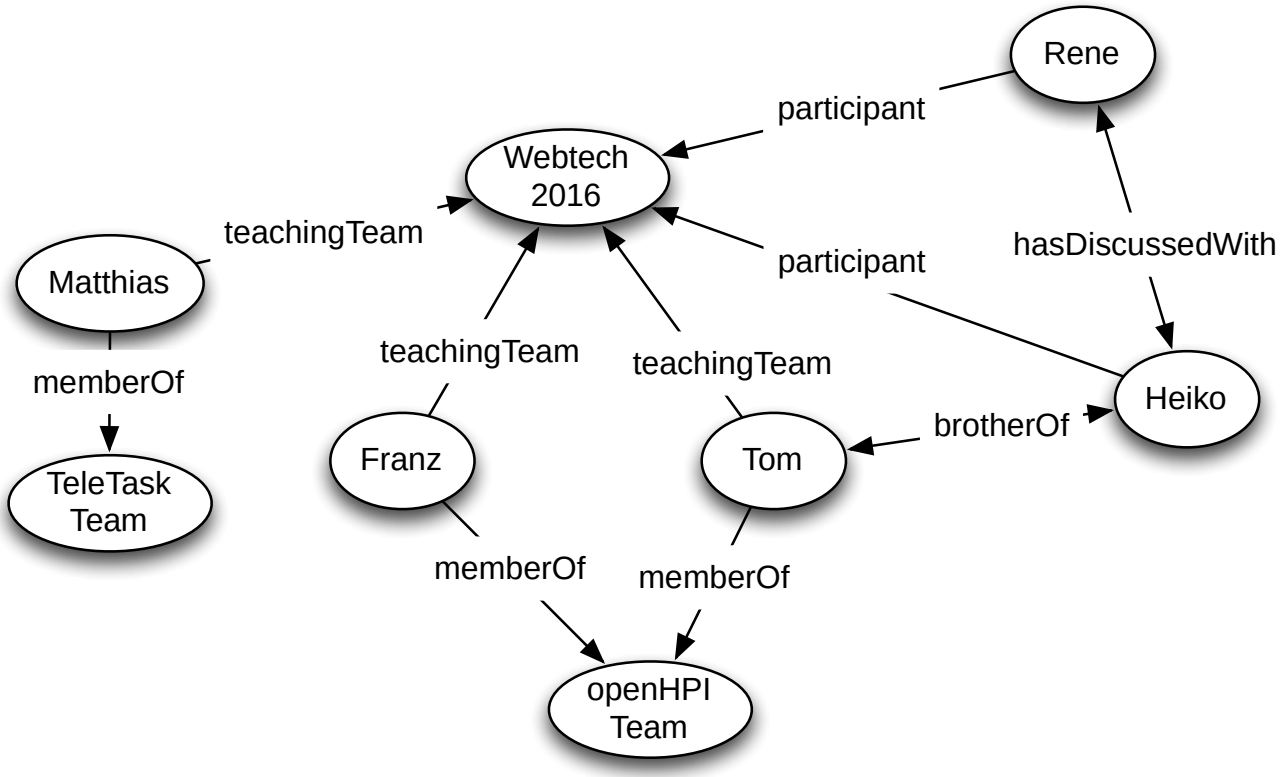


Relationale Datenbanken (3/3)

- **Indizes**
  - vergleichbar mit Index in einem Buch
  - ermöglichen direkten Zugriff auf Datensatz, anstatt Tabelle Reihe per Reihe zu durchsuchen
  - deutlicher Performance-Gewinn
- **Transaktionen**
  - zusammenfassen mehrerer Operationen
  - Operation einer Transaktion schlägt fehl ➔ Rollback
- **Stored Procedures**: In Datenbank gespeicherter, ausführbarer Code
- **Query-Sprache**: SQL (Structured Query Language)
  - SELECT, UPDATE, DELETE, INSERT
  - JOIN, UNION, INTERSECT, ...
  - WHERE, ORDER BY, LIMIT

NoSQL-Datenbanken (1/2)
Graph Stores / Graph-Datenbanken

- Basieren auf Graph-Struktur ➔ Knoten (Nodes), Kanten (Edges)



- Query-Sprachen: Gremlin, SPARQL, Cypher
- Anwendung: Soziale Netze, Semantic Web, etc.

NoSQL-Datenbanken (2/2)
Document Stores

- Speicherung komplexer Daten, Arrays, Objekte, Key-Value-Paare
- Speicherung der Daten häufig im JSON Format

Dokument

```
{ "user":
  {
    "id": "a719f263-5ddd-4f96-a337-3883f79f1dee",
    "personal_data": {
      "last_name": "Meier",
      "first_name": "Klaus",
      "email": "klaus.meier@example.com"
    },
    "courses": [
      "webtech1",
      "webtech2",
      "intsec1",
      "java1"
    ],
    "kyu": "yellow"
  }
}
```

- Keine Query Sprache – Zugriff über OOP APIs
- Alternative: XML-Datenbanken – Query-Sprache: XPath

Datenbanken für Webanwendungen
Pro und Kontra

- **Relationale Datenbanken…**
  - … unflexibler aufgrund ihrer fixen Datenstruktur
  - … mehr Möglichkeiten zur Sicherung der Datenkonsistenz
  - … schwierige horizontale Skalierbarkeit
  - Mischformen (z.B. JSONB-Datentyp in Postgres)
- **NoSQL-Datenbanken…**
  - … vereinfachen agile Entwicklung durch flexible Datenstruktur
  - … schlechtere Transaktionssicherheit
  - … einfachere horizontale Skalierbarkeit
  - Übersicht: http://nosql-database.org/

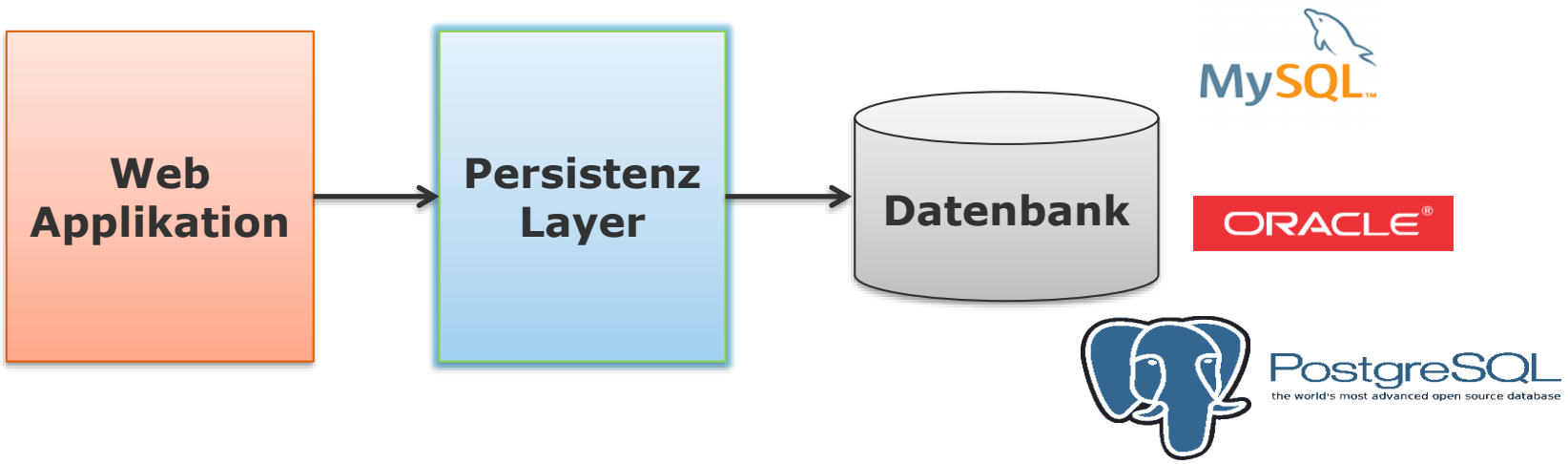
Empfehlung: Nutzen Sie das zum Zweck passende Werkzeug

In-Memory-Datenbanken → openHPI-Kurs im September

Persistenz-Layer (1/2)

Persistenz-Layer:

- Abstraktionsebene zwischen Datenbank und Applikation
- Gründe:
  - Vendor-Lock-In vermeiden: Abstraktion erleichtert Austausch der Datenbank (Maintainability)
  - Schwieriger: Umstieg von einem DB-Typ auf anderen
  - Komfortablerer Zugriff auf Daten für Entwickler (Produktivität)
  - Daten-Zugriffsoptimierung (Performance)



Persistenz-Layer (2/2)

Moderne Web-Applikationen werden in **objekt-orientierten** Sprachen entwickelt

- Anwendung von Konzepten der objekt-orientierten Programmierung verbessert
  - Kapselung & Wiederverwendbarkeit & Abstraktion
- Object-Relational-Mapping – **ORM**:
  - Abbildung von Objekten auf Datensätze
- ORM-Tools gibt es für alle verbreiteten Programmiersprachen – oft integriert in Persistenz-Layer der Frameworks
  - Ruby: ActiveRecord / Rails
  - Python: Django
  - Java: Hibernate
  - .NET: NHibernate

Caching

- Datenbank und Applikation laufen typischerweise auf verschiedener Hardware
- Performanceverlust
  - limitierte Geschwindigkeit des Netzwerks
  - komplexe Datenbank-Queries
- Performancesteigerung durch Caching der Daten auf Applikationsserver
- Verwendete Technologien
  - Key-Value-Store, z.B. Redis
  - Caching im Arbeitspeicher, z.B. Memcached

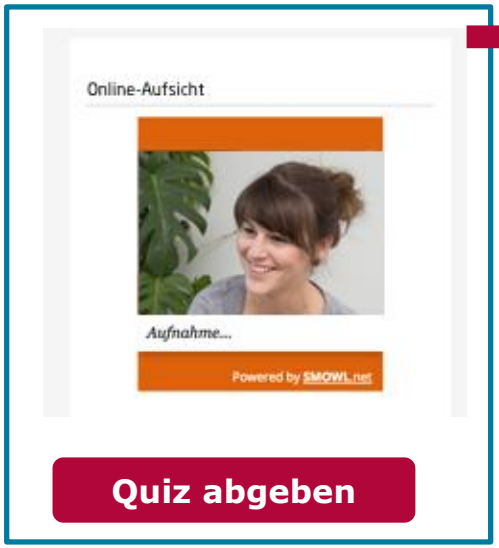
openHPI: **Web-Technologien**
Webservices

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Webservices

- Allgemeiner Ansatz für Entwicklung von über das Web verteilten Anwendungen
- Zugriff auf Dienste verschiedener Anbieter
  - unabhängig von Betriebssystemen, Programmiersprachen und binären Übertragungsprotokollen
- Ermöglichen Kommunikation zwischen verteilten Anwendungen

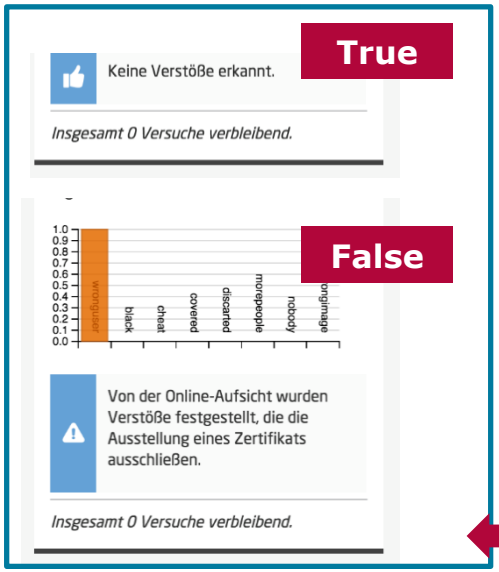
Webservices
Beispiel: Online-Proctoring bei openHPI mit SMOWL



openHPI

Request

```
<soapenv:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  ...
  xmlns:urn="urn:WS_MOOCS_Data_Deliver">
   <soapenv:Header/>
   <soapenv:Body>
      <urn:Passed_Fail_Exam soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <entity xsi:type="xsd:string">HPI</entity>
         <idQuiz xsi:type="xsd:string">56FIS0Xy1qQn711W2onRt0_4crNVMjqhRM6arOrvXXZnQ</idQuiz>
         <idUser xsi:type="xsd:string">HPI_3d88e1b37e07b747444568259c490152bf66[...]</idUser>
         <a1_Nobody xsi:type="xsd:string">5</a1_Nobody>
         <a2_IncorrectUser xsi:type="xsd:string">0</a2_IncorrectUser>
         ...
      </urn:Passed_Fail_Exam>
   </soapenv:Body>
</soapenv:Envelope>
```



SMOWL

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 ...
 xmlns:ns4="urn:WS_MOOCS_Data_Deliver"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns4:Passed_Fail_ExamResponse>
    <ack xsi:type="xsd:boolean">false</ack>
    </ns4:Passed_Fail_ExamResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

openHPI

Webservice-Protokolle 1. Generation

SOAP – XML–basiertes Protokoll

- SOAP Envelopes – Header und Body (wie HTTP)

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="... >
   <soap:Header>... </soap:Header>
   <soap:Body>  ... </soap:Body>
</soap:Envelope>
```

- SOAP Envelope wird als Payload im Body von HTTP-Request/HTTP-Response übermittelt
- Requests in der Regel per HTTP POST
- Aber: SOAP ist nicht auf HTTP als Transportmedium angewiesen

WSDL - Web Service Description Language

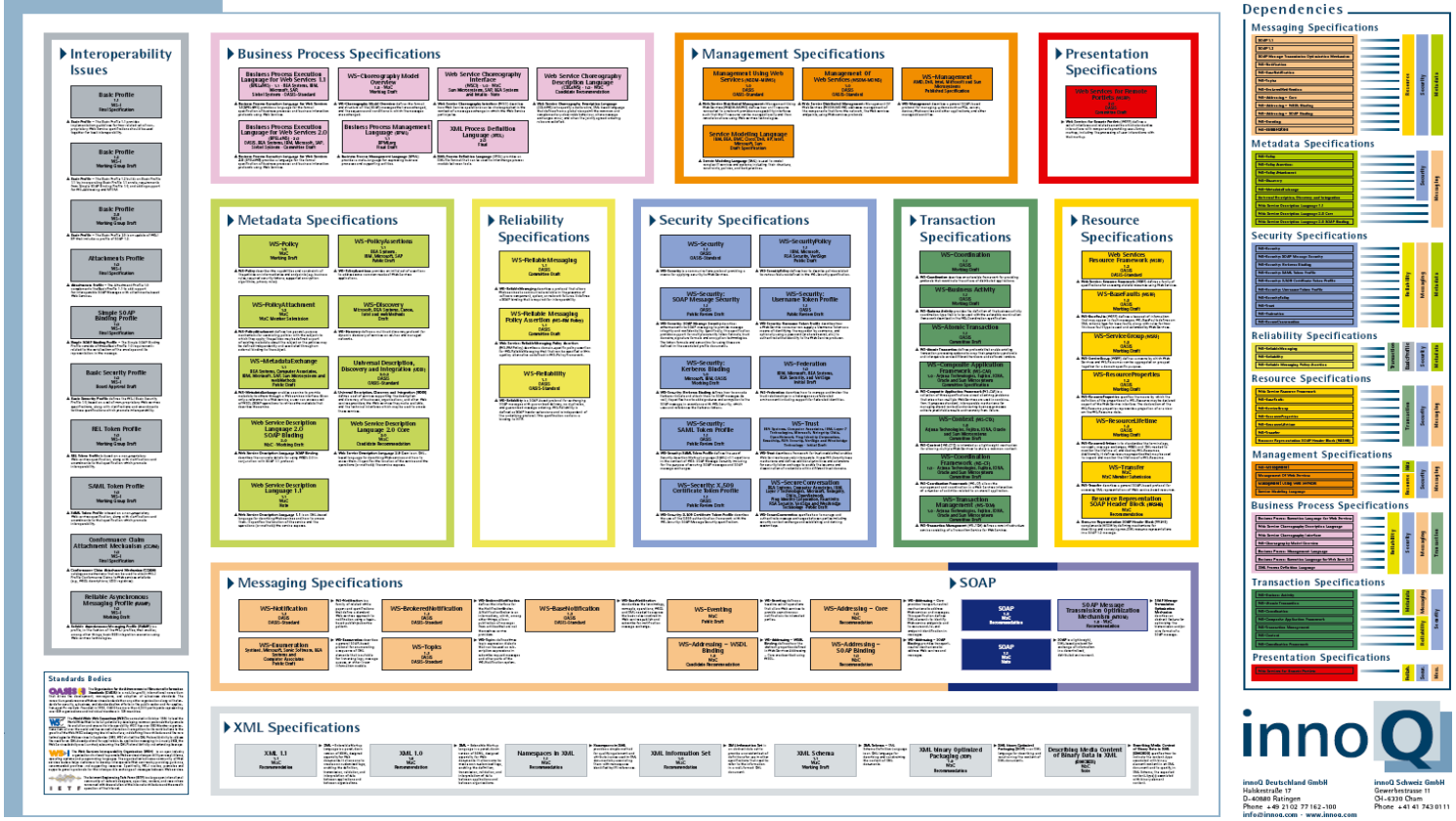
- XML-basierte Beschreibungssprache für Webservice-Schnittstellen
- Definition/Beschreibung von Methoden, Parametern, Rückgabewerten

Webservice-Protokolle 2. Generation

WS*-Protokoll-Suite Webservice-Standards der zweiten Generation

- Absicherung von Web Services
- Komposition mehrerer Web Services (Orchestrierung)
- Zuverlässige Ausführung der Web Services
- Protokolle
  - WS-Security, WS-Policy
  - WS-Coordination, WS-Federation, WS-Notification, ...
- WS* (es gibt sehr, sehr viele WS-Standards …)
- Spielen hauptsächlich im Enterprise-Bereich eine Rolle, sind im Web weniger verbreitet

Web Services Standards Overview



https://www.innoq.com/soa/ws-standards/poster/innoQ%20WS-Standards%20Poster%202007-02.pdf

Schlanke Webservices

Etablierung schlanker Webservices als Alternative zu den WS*-Standards für verteilte Systeme im Enterprise Bereich:
XML-RPC and **REST**

- **XML-RPC (Remote Procedure Call)**
  - Protokoll für Webservices – historischer Vorgänger von SOAP
  - einfaches XML-basiertes Protokoll für Aufruf von Methoden über das Web
  - Aufrufe erfolgen über HTTP POST – auszuführende Aktion werden in XML definiert

XML-RPC example:

```
POST /shoppingcart HTTP/1.1
Host: myServer.org
Content-Type: application/xml


<?xml version="1.0"?>
<methodCall>
   <methodName> deleteItem </ methodName >
   <params>
     <param><value><int>244</int></value></param>
   </params>
</methodCall>
```

Transport per POST

Auszuführende Aktion im Body des Requests

Adressat der Aktion wird ebenfalls im Body des Requests definiert

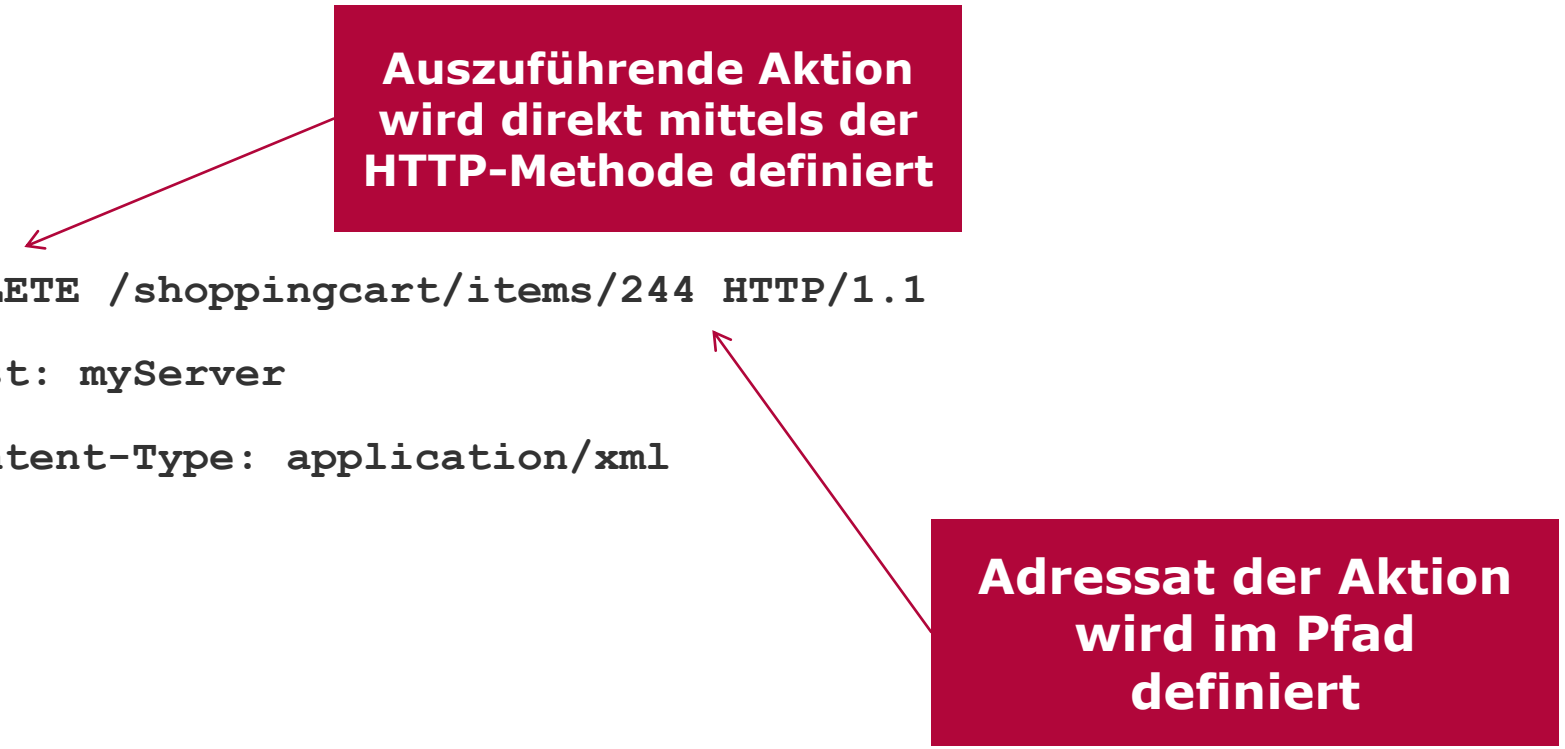
Schlanke Webservices

Etablierung schlanker Webservices als Alternative zu den WS*-Standards für verteilte Systeme im Enterprise Bereich: **XML-RPC** and **REST**

- **REST – REpresentational State Transfer**
  - keine Technologie – sondern Architektur
  - ROA – Resource Oriented Architecture
  - benutzt komplettes HTTP Vokabular: GET, POST, PUT, …
    - auszuführende Aktion werden direkt per HTTP definiert
  - erstmals beschrieben in Dissertation Roy Fielding (2000)
  - inzwischen weiter verbreitet als XML-RPC

Schlanke Webservices – REST (1/4)

REST example:



Schlanke Webservices – REST (2/4)

REST - REpresentational State Transfer

- Web-Applikationen leben von **Web-Ressourcen** (erreichbar über ihre URL)) und ihrer Ausgestaltung, z.B. User, Course, Enrollment, …
- REST bietet Weg, diese Ressourcen direkt mit Mitteln von HTTP zu manipulieren und zu verknüpfen
- HTTP Pfad bestimmt, welche Ressource manipuliert wird
- HTTP Methode bestimmt, wie Ressource manipuliert wird
  - POST – Ressource wird erzeugt (Create)
  - GET – Ressource wird gelesen (Read)
  - PUT – Ressource wird aktualisiert (Update)
  - DELETE – Ressource wird gelöscht (Delete)
- Im HTTP Body werden nur noch die eigentlichen Daten transportiert (POST/PUT)

Schlanke Webservices – REST (3/4)

Wiederholung: Elemente des HTTP-Protokolls (RFC 2616)

- HTTP Request
  - Methoden: GET, POST, PUT, DELETE
  - Pfad: Teil der URL https://openhpi.de**/courses/1/settings**
  - Request Header
  - Request Body
- HTTP Response
  - Response code: 200 OK, 403 Forbidden, 404 Not Found ...
  - Response Header
  - Response Body

Schlanke Webservices – REST (4/4)

REST – Design Prinzipien:

- Immer passende HTTP Methode GET, POST, PUT oder DELETE nutzen
- Zustandslosigkeit (Statelessness) – Alle benötigten Daten mit Request bereitstellen
- URLs orientieren sich an Dateistrukturen
- Service registriert den MIME-Type des Requests und sendet Response im passenden MIME-Format zurück

WADL - Web Application Description Language

- XML-basierte Beschreibungssprache
- Entspricht WSDL in den SOAP basierten Webservices