



Forschungszentrum
Informatik



Universität
Karlsruhe (TH)



Information Process Engineering

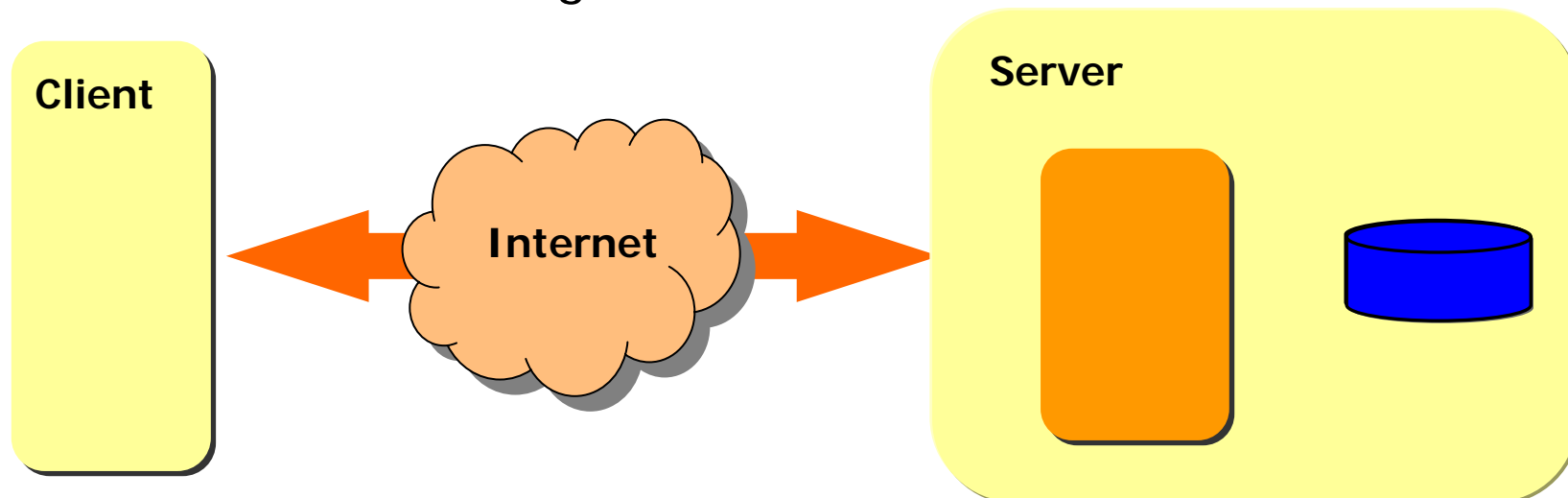
Web-Technologien im Überblick

Andreas Schmidt

WS 2006/2007

Motivation

- Endanwender sollen auf servergestützte Anwendungen über das (öffentliche) Internet zugreifen
 - Rahmenbedingungen
 - ▶ verträglich mit Netzwerkkonfigurationen (Firewalls, ...)
 - ▶ ohne separate Installation
- => Web-Technologien als Basis



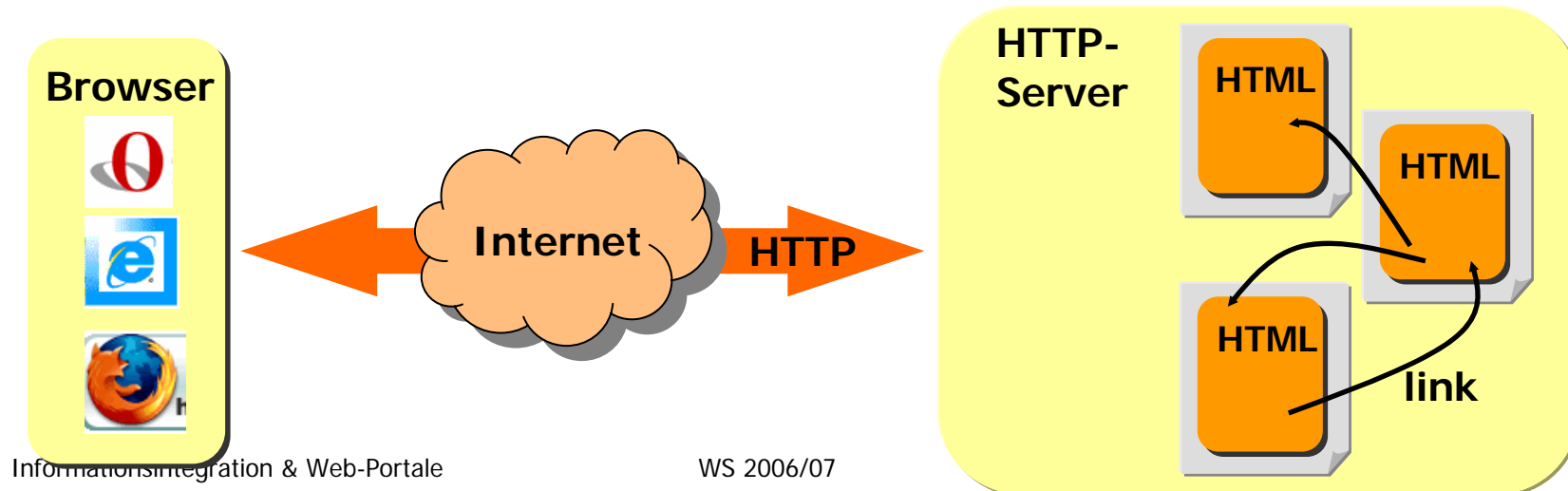
- Was sind Standard-Web-Technologien?
- Serverseitige Technologien
 - ▶ CGI
 - ▶ API-basierte Ansätze
 - ▶ Skriptbasierte Ansätze
 - ▶ JSP als Beispiel für einen skriptbasierten Ansatz
 - ▶ JavaServerFaces als komponentenorientierte Weiterentwicklung
- Clientseitige Technologien
 - ▶ Plugins
 - ▶ AJAX
- Web 2.0?

Was sind Standard-Web-Technologien?



Standard-WWW-Architektur

- Internet-basierte Client/Server-Architektur
 - **Browser** (Client) zur graphischen Darstellung
 - HTTP-Server (= **Web-Server**) zur Übertragung der HTML-Seiten
- **HTTP**: Protokoll zur Übertragung der Seiten
- **HTML**: Sprache zur Beschreibung der Seiten
- **CSS**: Layoutspezifikation
- **URL**: für unidirektionale Links



HTTP: Das Protokoll des WWW

- HTTP - HyperText Transfer Protocol
 - ▶ Dient der Übertragung von Hypermedia-Dokumenten zwischen einem Server und einem Client
 - ▶ Zustandslos
 - kleinere Aufweichungen durch HTTP 1.1
 - ▶ Übermittlung von Parametern als Name-Wert-Paare
 - GET bzw. POST

- Inzwischen universelles Transportprotokoll
 - ▶ WebDAV Zugriff auf Dateisystem
 - ▶ SOAP RPC
 - ▶ ...

HTML: Struktur, Layout und Logik



■ XHTML

- ▶ beschreibt die Inhalte und die Strukturen
- ▶ Links, Formulare etc.

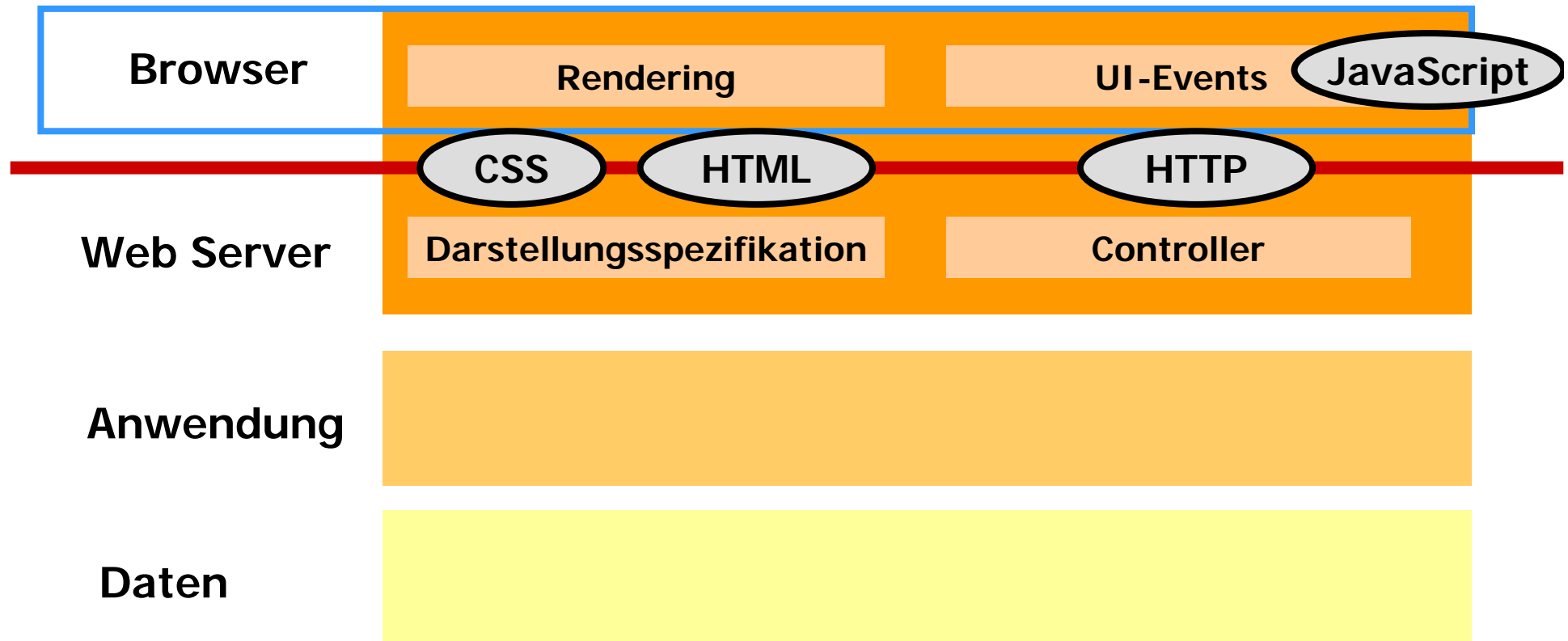
■ CSS

- ▶ beschreibt die Darstellung
- ▶ Schriften, Farben, Rahmen, Hintergründe, Positionen

■ JavaScript

- ▶ für clientseitige Verarbeitungslogik
- ▶ z.B. Feldinhaltvalidierung, fortgeschrittene UI-Methoden

Aufgabenteilung im Standardmodell



Serverseitige Technologien



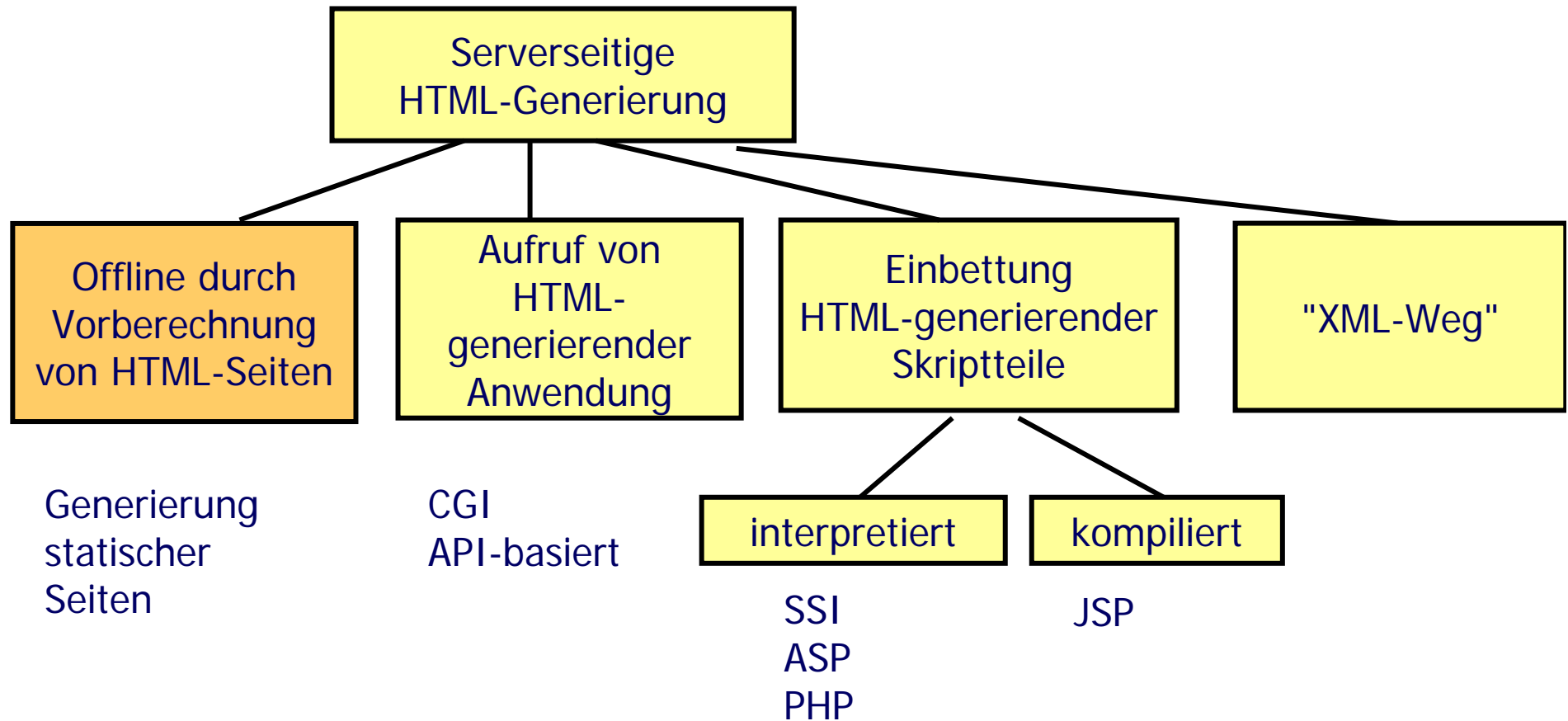
Serverseitige Technologien



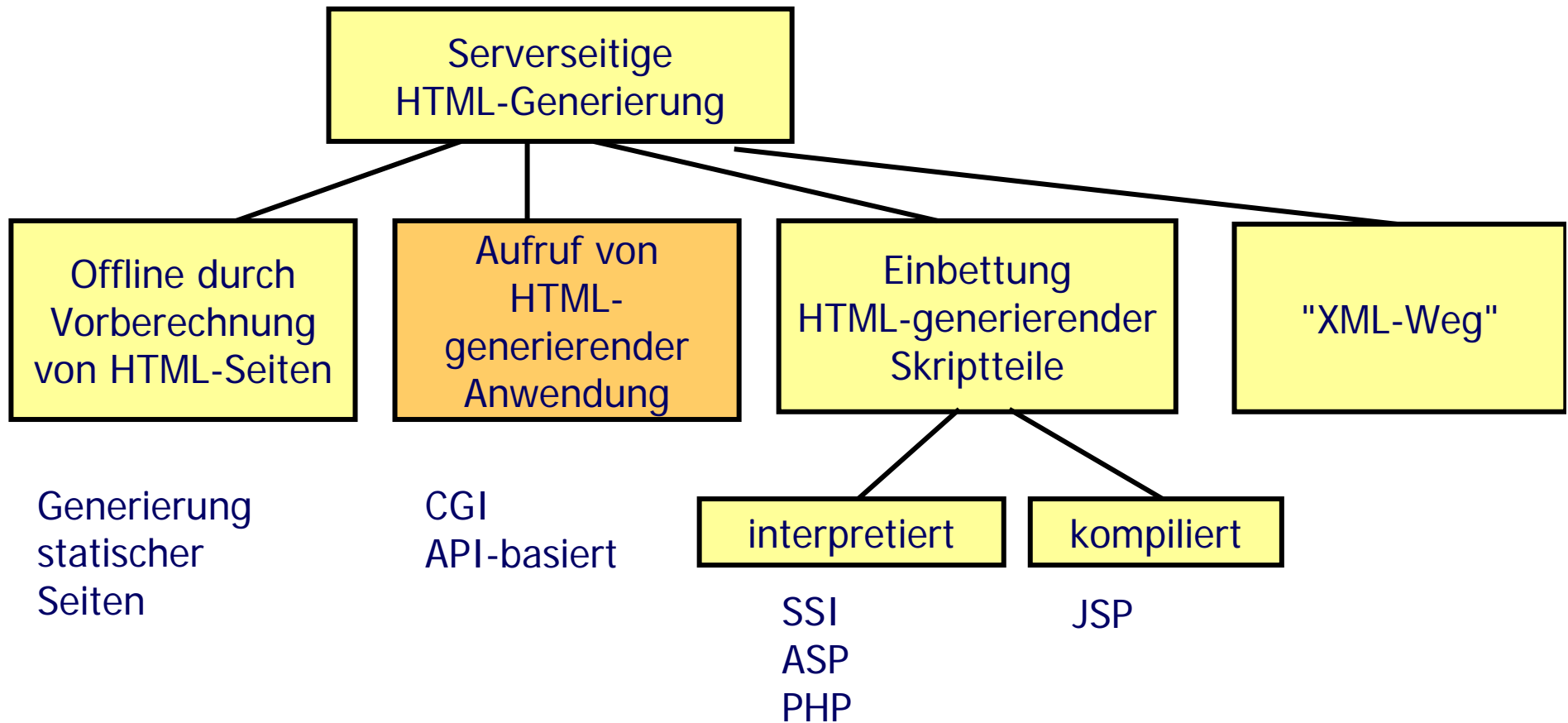
- Funktionale Anforderungen
 - ▶ Generierung der Darstellungspezifikation (HTML)
 - ▶ Verarbeitung der Benutzerschnittstellenaktionen (HTTP-Requests)
 - ▶ Aufruf der Anwendungslogik

- Nicht funktionale Anforderungen
 - ▶ Wartbarkeit
 - ▶ Flexibilität
 - ▶ Skalierbarkeit

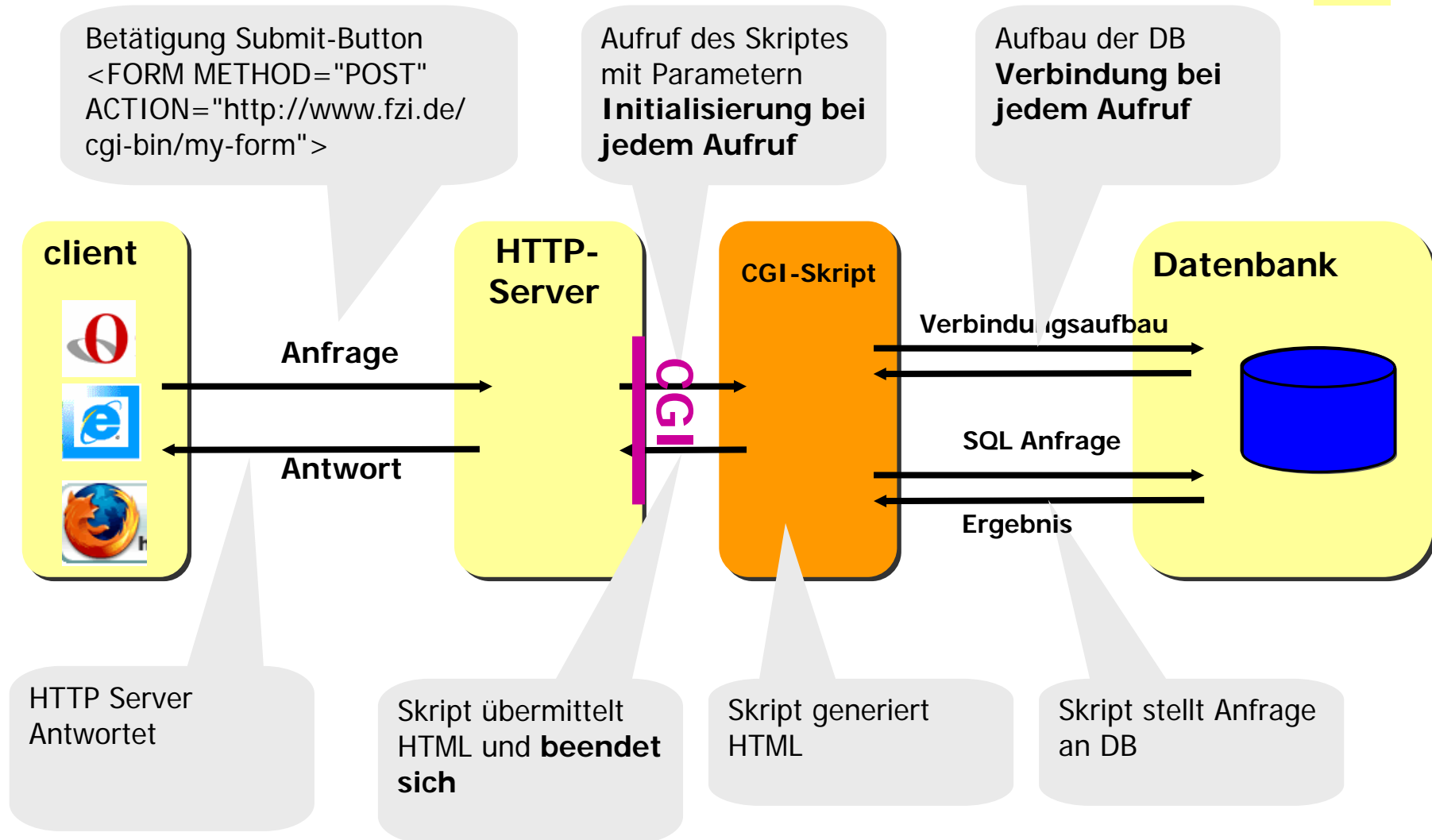
Übersicht server-seitige Ansätze



Übersicht server-seitige Ansätze



CGI-Skripte



■ Vorteile

- ▶ Beliebige Programme können integriert werden
- ▶ Sicherheit durch eigenen Prozess

■ Nachteile

- ▶ Ein Prozess pro Anfrage
- ▶ Keine Speicherung des Zustands
- ▶ Für jede DB-Anfrage Verbindung aufbauen und trennen
- ▶ Keine Trennung von Präsentation und Anwendungslogik

■ CGI als Schnittstelle immer noch geeignet und genutzt

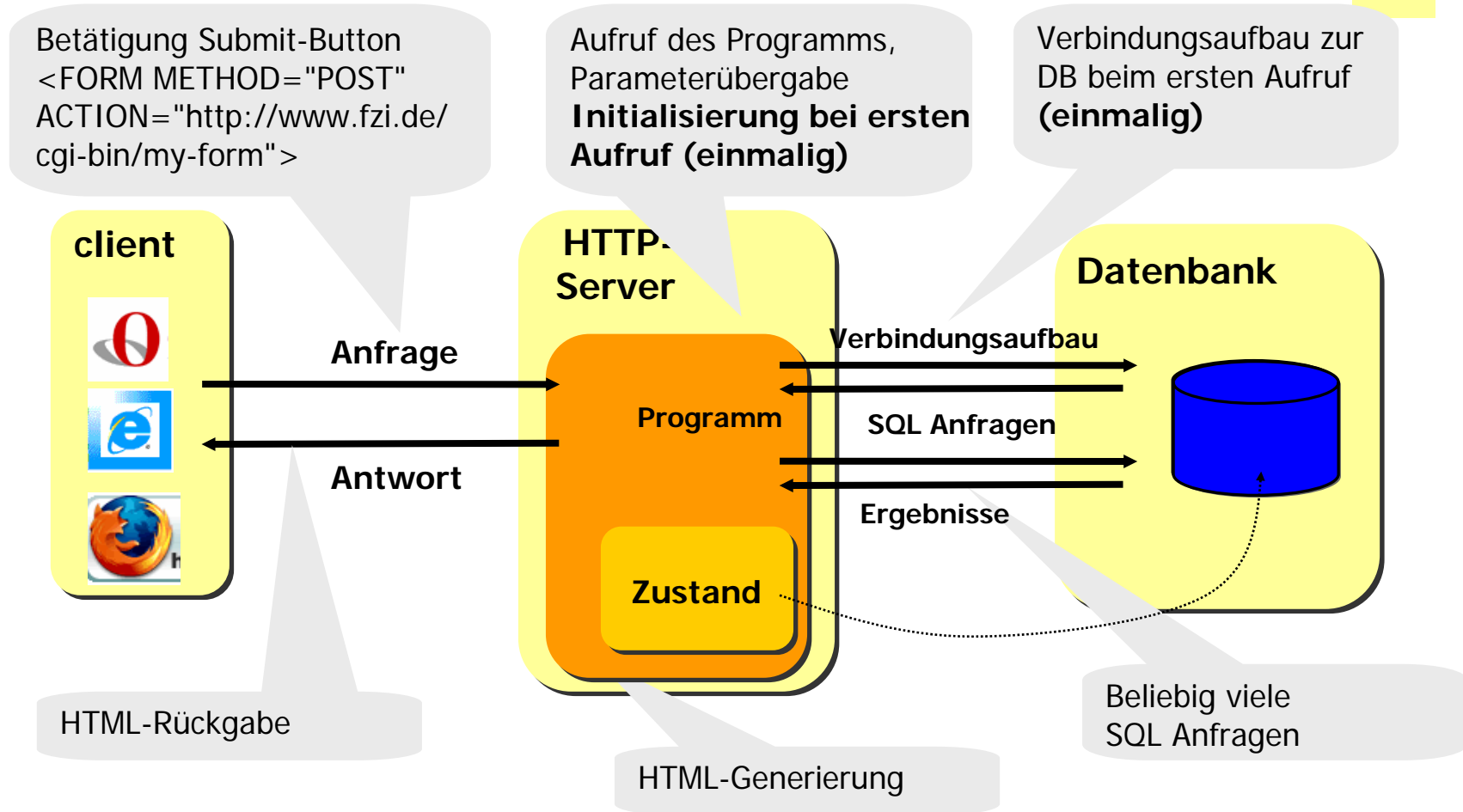
- ▶ z.B. auch als Aufruf für Skript-Interpreter

API-basierte Ansätze



- Entwickelt um Nachteile der CGI-Skripte zu überwinden
- Erweiterungen werden in den Adressraum des Servers geladen
 - ▶ Müssen nur einmal geladen werden
 - ▶ Werden in Threads statt Prozessen ausgeführt
- Bekanntesten Vertreter
 - ▶ NSAPI (Netscape)
 - ▶ ISAPI (Microsoft)
 - ▶ **Java Servlets** (Sun)

API-basierte Ansätze



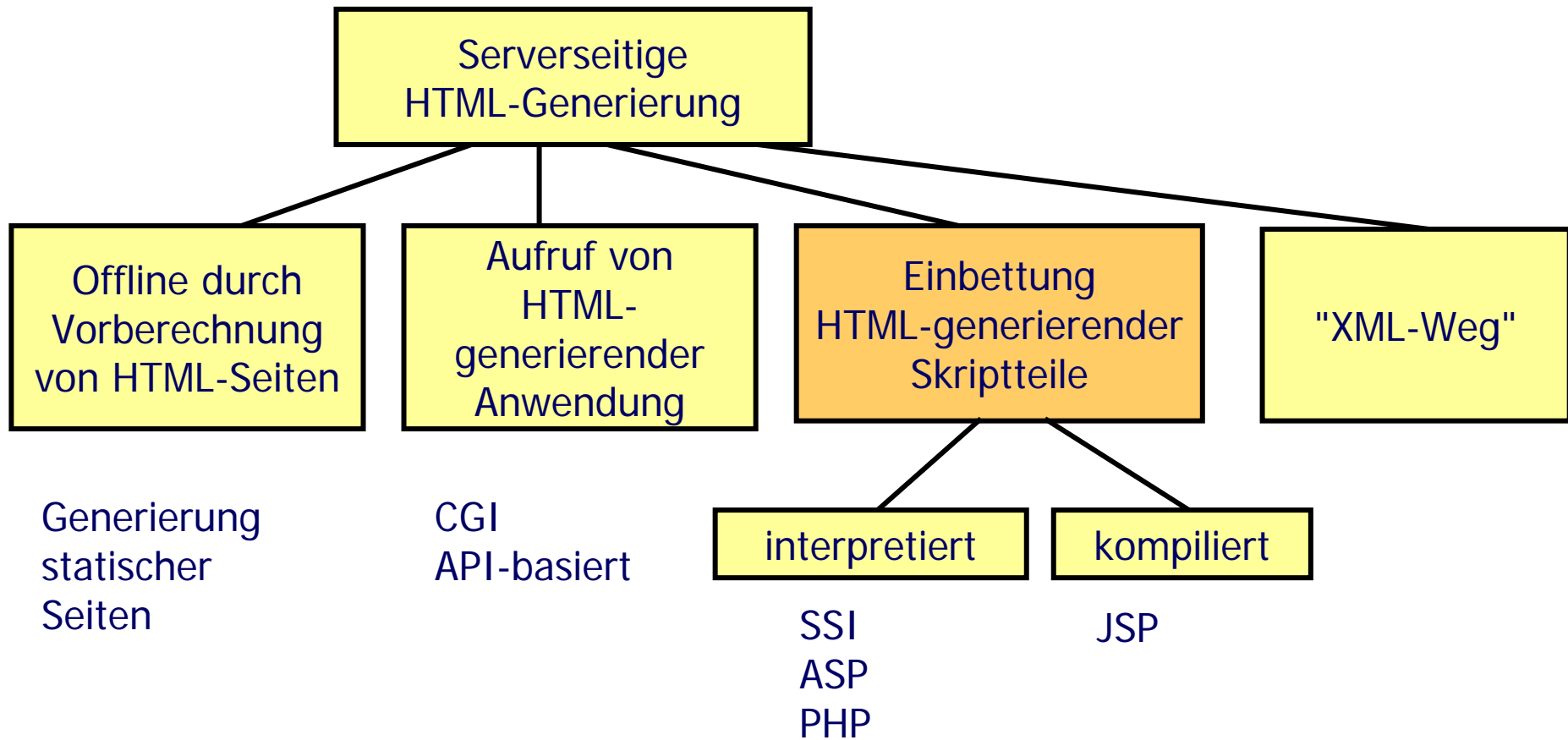
■ Vorteile

- ▶ Höhere Leistungsfähigkeit
 - Session-Verwaltung
 - Zustände, z.B. DB-Verbindung
- ▶ Weniger Ressourcenverbrauch

■ Nachteile

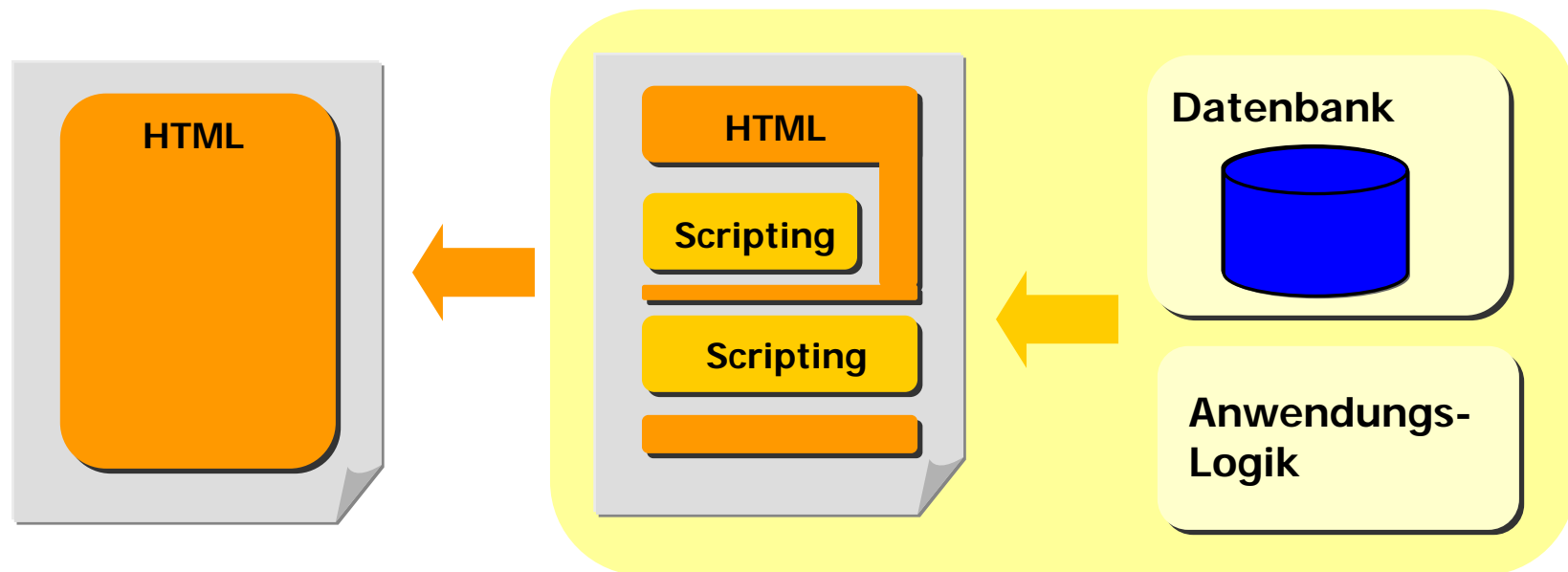
- ▶ Keine Trennung von Präsentation und Anwendungslogik

Übersicht server-seitige Ansätze

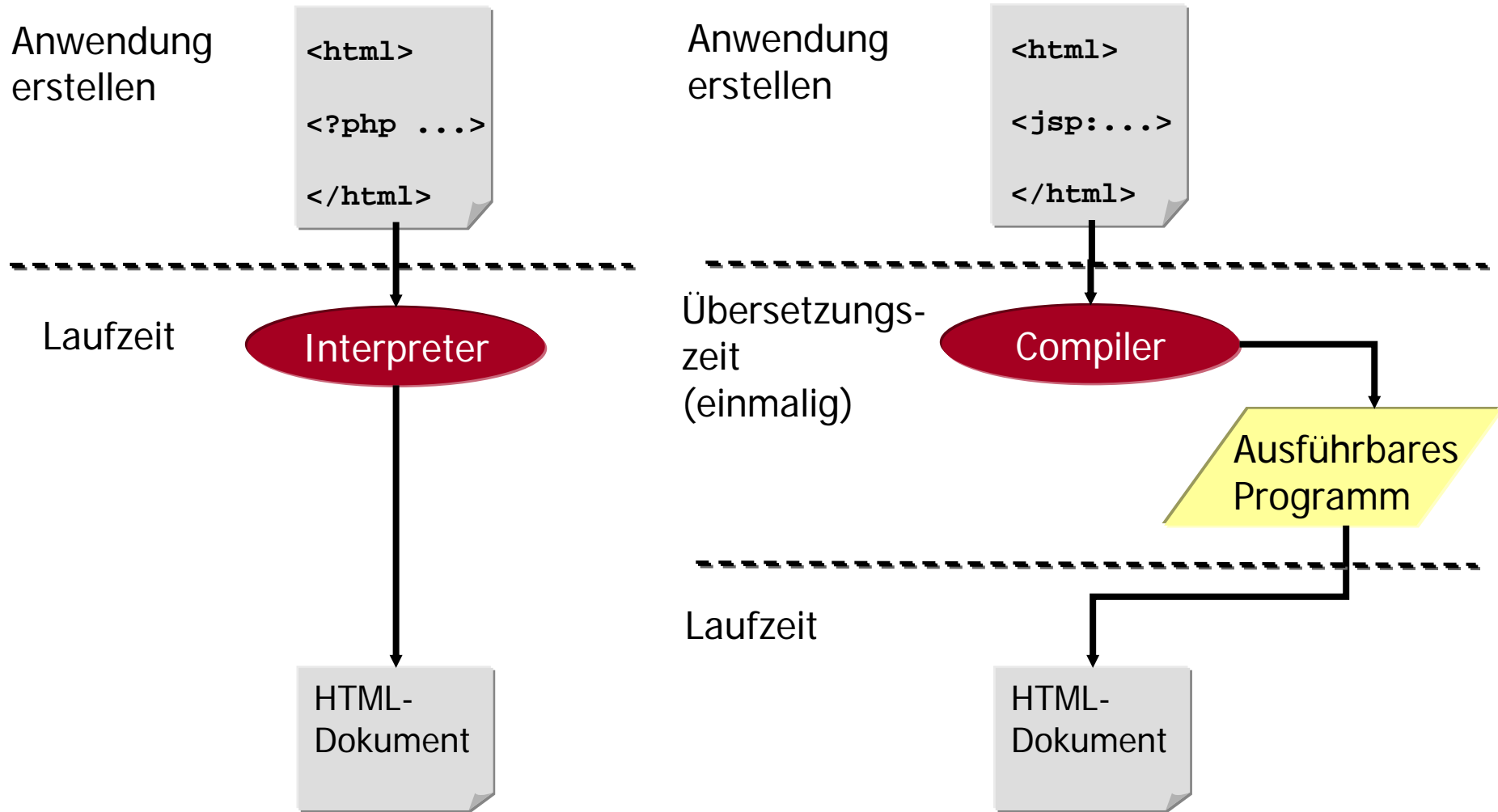


Server-Side Scripting

- In HTML-Seiten werden zusätzliche **HTML-generierende Quellen** integriert



Ausführungsvarianten

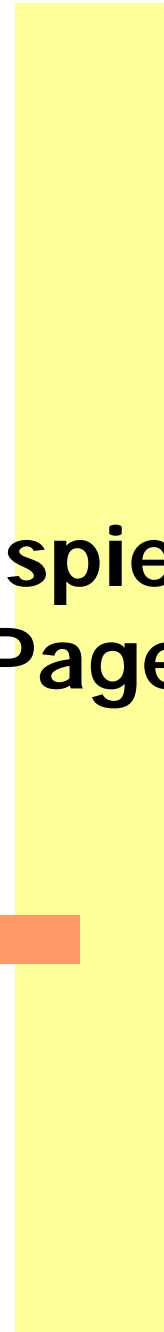


Typische Vertreter



- Server Side JavaScript (Netscape)
- PHP - PHP Hypertext Processor
- ASP - Active Server Pages (Microsoft IIS)
- JSP - Java Server Pages

Beispiel: JSP Java Server Pages



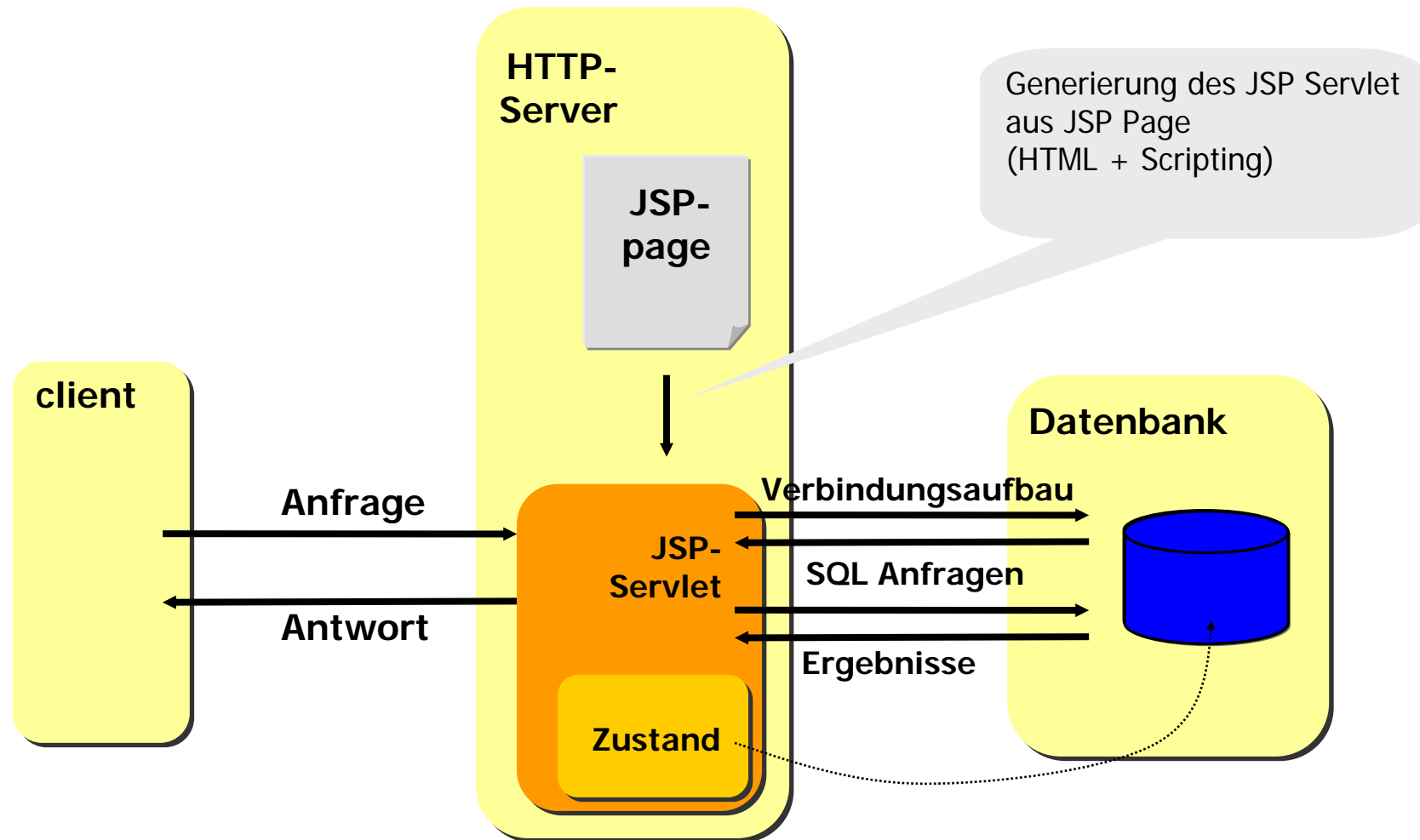
Java Server Pages (JSP)



- Bestandteil der Java 2 Plattform Enterprise Edition (J2EE)
- Motivation: Trennung von (HTML-)Präsentation und Inhalt

- Ablauf
 - Erstellung einer JSP
 - Aufruf der JSP durch den Benutzer
 - Überprüfung, ob sich die JSP geändert hat oder ob sie neu ist
 - (Gegebenenfalls Übersetzung der JSP in ein Servlet)
 - Ausführung des Servlets
 - Rückgabe des Ergebnisses

JSP Ablauf



JSP-Skriptelemente (1)

■ Vereinbarungen / Deklarationen

- ▶ Deklarationen von Variablen, Methoden und inneren Klassen
- ▶ Syntax: `<%! Vereinbarung(en) %>`
- ▶ `import` über die Direktive
- ▶ Beispiel:
 - Instanzvariablen: `<%! int i=0; float f;%>`
 - Methode: `<%! public String Zeit() {...} %>`

■ Anweisungsfragmente/Scriptlet

- ▶ Einbettung von Java-Fragmenten
- ▶ Syntax: `<% Anweisungsfragment(e) %>`
- ▶ Lokale Variablendeklaration
- ▶ Beispiel:
 - `<% int i=1; while (i<10) {out.println(i);i++;} %>`

JSP-Skriptelemente (2)

- **Ausdrücke**
 - ▶ Einfachste Art eines Skripts
 - ▶ Syntax: `<%= Ausdruck %>`
 - ▶ Umwandlung zur Laufzeit in String (`toString`-Methode)
 - ▶ Beispiel: `<%=Uhr.getTime()%>` entspricht
 - ▶ `<%out.print(Uhr.getTime());%>`
- **Kommentare**
 - ▶ Syntax: `<%-- comment --%>`

JSP: Beispiel

```
<%@ page errorPage="error.html"
import="java.sql.*" session="false" %>
<html>
  <head>
    <link ref="stylesheet" type="text/css" href="formate.css" />
    <body>
      <h1> Produktliste vom <%=new java.util.Date().toLocaleString()%>

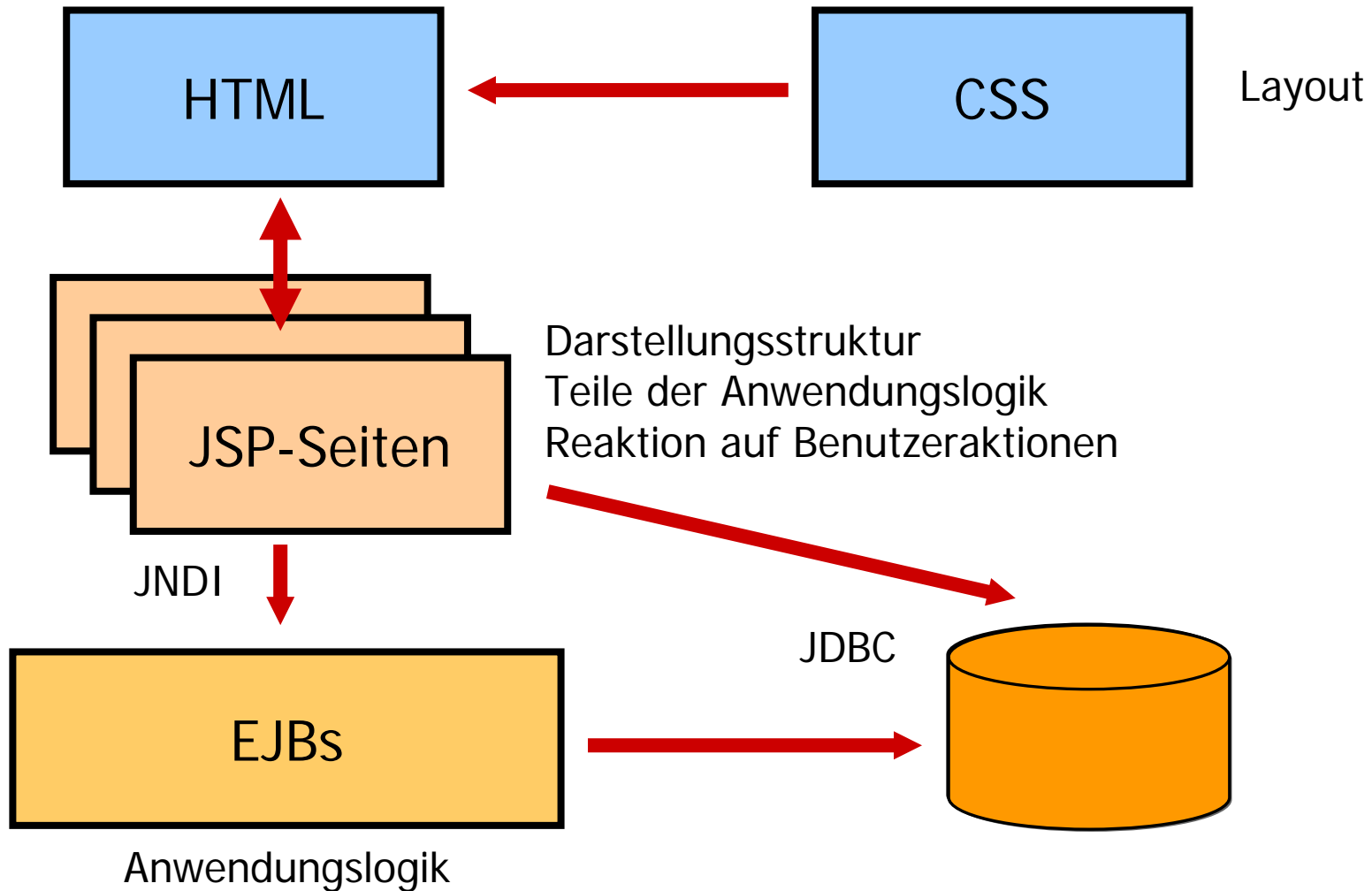
      </h1>

      <!-- Scriptlet zum Verbindungsaufbau und Abfrage-->
      <%
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection
          ("jdbc:odbc:ProduktDatenbank", "name", "pwd");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery
          ("SELECT name, photo_URL FROM artikel");
      %>
```

JSP: Beispiel (Forts.)

```
...
<%--   Ausgabe in zwei Spalten...--%>
<table>
<tr>
  <th>Name</th><th>Bild</th>
</tr>
<%--   ...Zeile für Zeile--%>
<%     while (rs.next()){ %>
  <tr><td> <%= rs.getString(1)%> </td>
  <td> </td>
  </tr>
<%}%>
</table>
<%--   Verbindung schließen--%>
<%     rs.close();%>
</body> </html>
```

Zusammenfassend: JSP 1.0



Probleme mit JSP 1.0



- Mischung von Präsentation und Anwendungslogik
 - ▶ einfache Erstellung
 - ▶ schnell unübersichtlich

- JSP soll eigentlich nur die Struktur der Darstellung festlegen
 - ▶ insbesondere Steuerelemente

- Auslagerung der aufwendigeren Logik

JSP 1.1 +: Benutzerdefinierte Tags

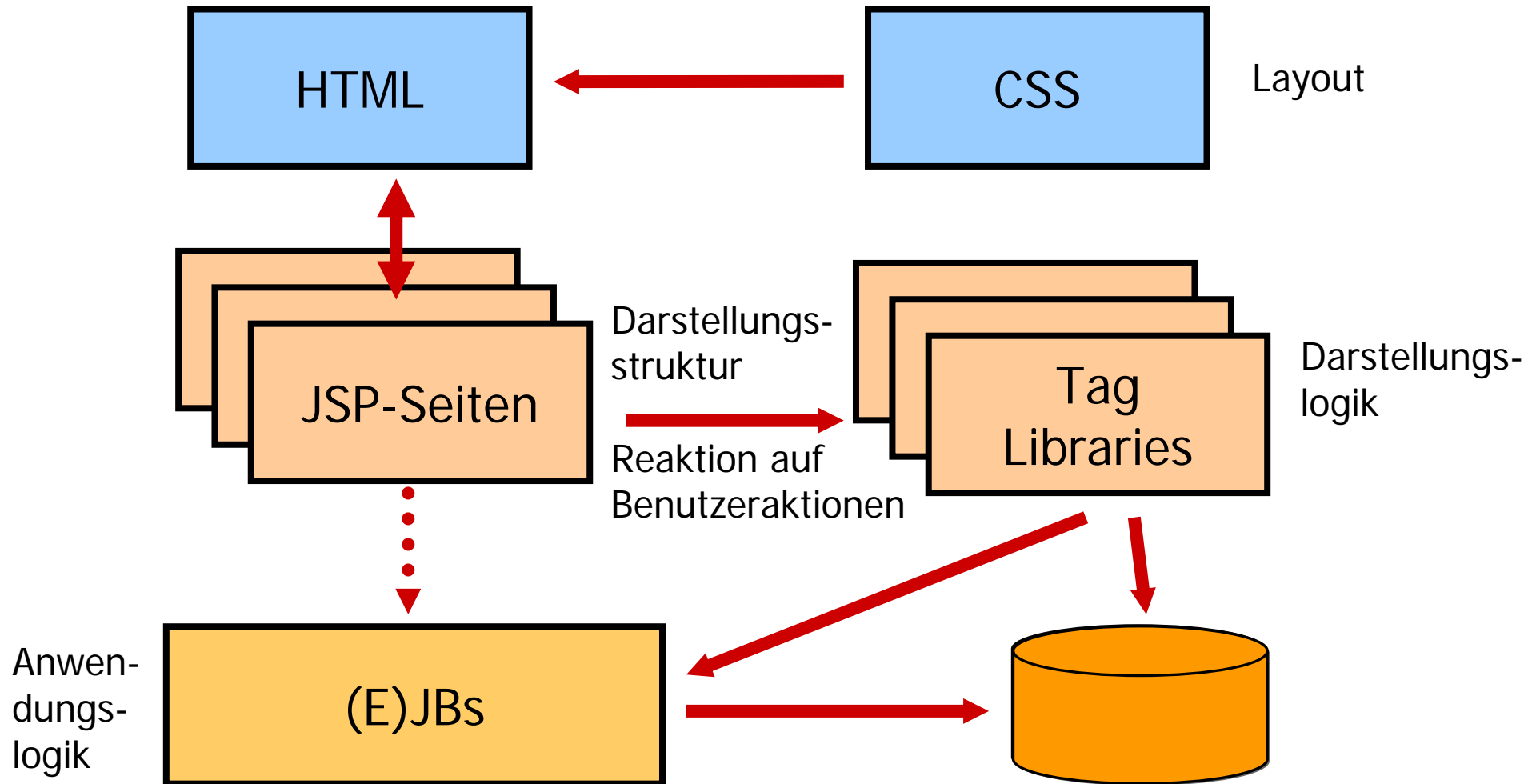
- Kapseln von immer wiederkehrender Funktionalität als Tag
- Standardbibliotheken:
JavaServer Pages Standard Tag Library (JSTL)
 - ▶ Core Tags: Standardfunktionalität
 - choose, forEach, if, ...
 - ▶ XML Tags: Zugriff auf XML-Dokumente
 - ▶ Internationalisierung
 - ▶ SQLTags: Standard DB Funktionen
- Anwendung kann weitere Tag Libraries definieren
- Zusätzlich Expression Language
 - ▶ einfacherer Zugriff auf Beans

JSP 1.1+: Beispiel

```
<table>
  <tr>
    <th>Name</th><th>Bild</th>
  </tr>

  <my:produkte query="" iterate="produkt">
    <tr>
      <td>${produkt.name} </td>
      <td></td>
    </tr>
  </my:produkte>
</table>
```

JSP 1.1+



JSP Bewertung



- Standardisierter Weg zur Trennung von
 - ▶ Gestaltung (Web-Designer) => JSP-Seiten
 - ▶ Anwendungslogik (Entwickler) => Tag Libraries, Back-End
- Grundlage für komponentenorientierte HTML-Generierung
 - ▶ Höhere Wiederverwendung der HTML-generierenden Quelle
 - ▶ Trennung von Präsentation und Anwendungslogik
- Allerdings:
 - ▶ Große Unterschiede zwischen Standard-GUI- und Web-GUI-Entwicklung trotz derselben Konzepte

Java Server Faces



JSF: Grundidee



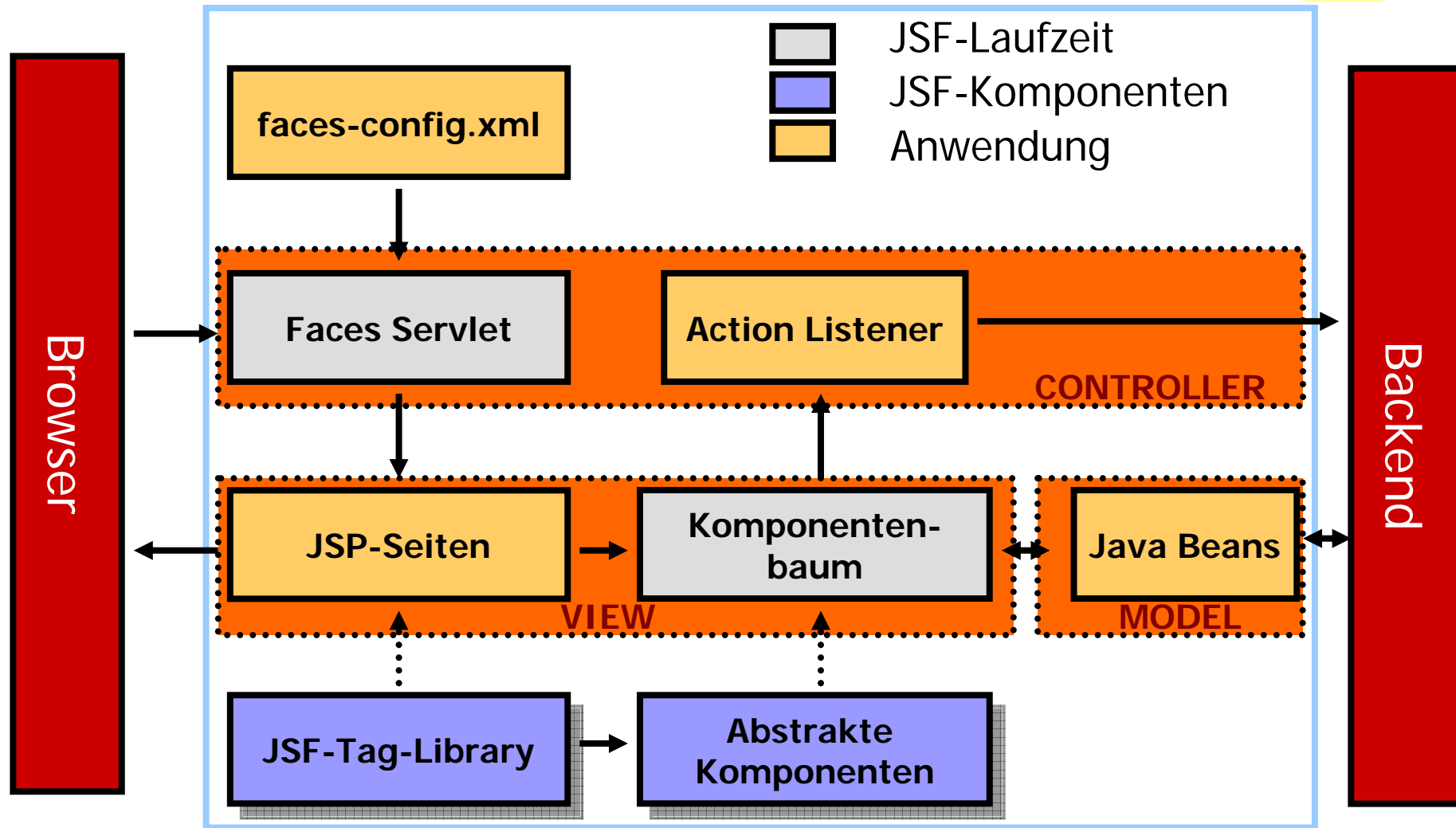
- Die Entwicklung von Web-Anwendungen soll so ähnlich wie möglich zur Entwicklung von anderen graphischen Benutzerschnittstellen sein (insbesondere Swing)
 - wiederverwendbare und erweiterbare Benutzerschnittstellen**komponenten**
 - **Model-View-Controller**-Prinzip
 - **ereignisgesteuert** (Listener-Pattern)
 - Komponentenmodell prinzipiell **unabhängig von der Zielplattform** (also auch Generierung von Swing-Oberflächen aus abstraktem Komponentenbaum)

JSF: Elemente



- UI-Komponenten
 - ▶ zustandsbehaftet (d.h. JSF übernimmt das Zustandsmanagement)
- Java-Beans
 - ▶ stellen das Modell dar
 - ▶ Schnittstelle zu den darzustellenden Daten
- Listener
 - ▶ Reaktionen auf Ereignisse der UI-Komponenten
- JSP-Seiten
 - ▶ Seitenstruktur und Anordnung der UI-Komponenten
- Navigationsregeln
 - ▶ definieren den Fluss zwischen unterschiedlichen Seiten

JSF: Ablauf



JSF Schritt 1: Bean schreiben

```
public class Kunde
{
    String kdnr;
    String name;
    public void setKundennr(String nr) { kdnr = nr; }
    public String getKundennr() { return kdnr; }
    public String getPersonName() { return name; }
```

```
    public void fetchData(ActionEvent event)
    {
        // hier Zugriff auf die Kundendatenbank
    }
```

```
}
```

JSF Schritt 2: Konfiguration

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
"http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
```

```
<faces-config>
```

```
<navigation-rule>
```

```
<from-view-id>/pages/start.jsp</from-view-id>
```

```
<navigation-case>
```

```
<from-outcome>login</from-outcome>
```

```
<to-view-id>/pages/greeting.jsp</to-view-id>
```

```
</navigation-case>
```

```
</navigation-rule>
```

```
<managed-bean>
```

```
<managed-bean-name>personBean</managed-bean-name>
```

```
<managed-bean-class>de.fzi.PersonBean</managed-bean-class>
```

```
<managed-bean-scope>session</managed-bean-scope>
```

```
</managed-bean>
```

```
</faces-config>
```

JSF Schritt 3: start.jsp schreiben

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="jsfks.bundle.messages" var="msg"/>

<html>
  <body>
    <f:view>
      <h1><h:outputText value="#{msg.inputname_header}"/></h1>
      <h:form id="helloForm">
        <h:outputText value="#{msg.prompt}"/>
        <h:inputText value="#{personBean.kundennr}" />
        <h:commandButton actionListener=#{personBean.fetchData}"
          action="login" value="#{msg.button_text}" />
      </h:form>
    </f:view>
  </body>
</html>
```

JSF Schritt 4: greeting.jsp schreiben

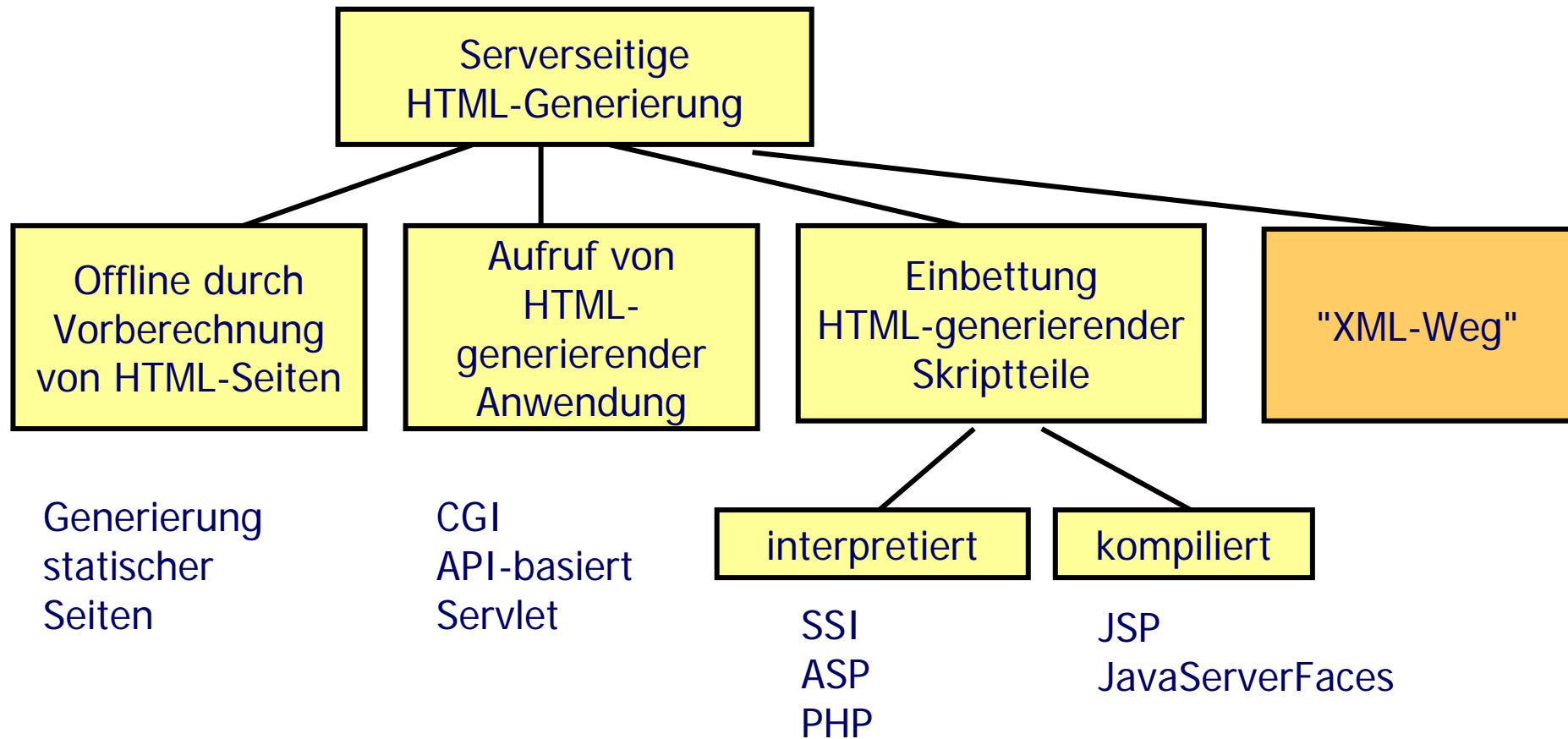
```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="jsfks.bundle.messages" var="msg"/>
```

```
<html>
<body>
  <f:view>
    <h3>
      <h:outputText value="#{msg.greeting_text}" />,
      <h:outputText value="#{personBean.personName}" />
      <h:outputText value="#{msg.sign}" />
    </h3>
  </f:view>
</body>
</html>
```

Der "XML-Weg"



Übersicht server-seitige Ansätze

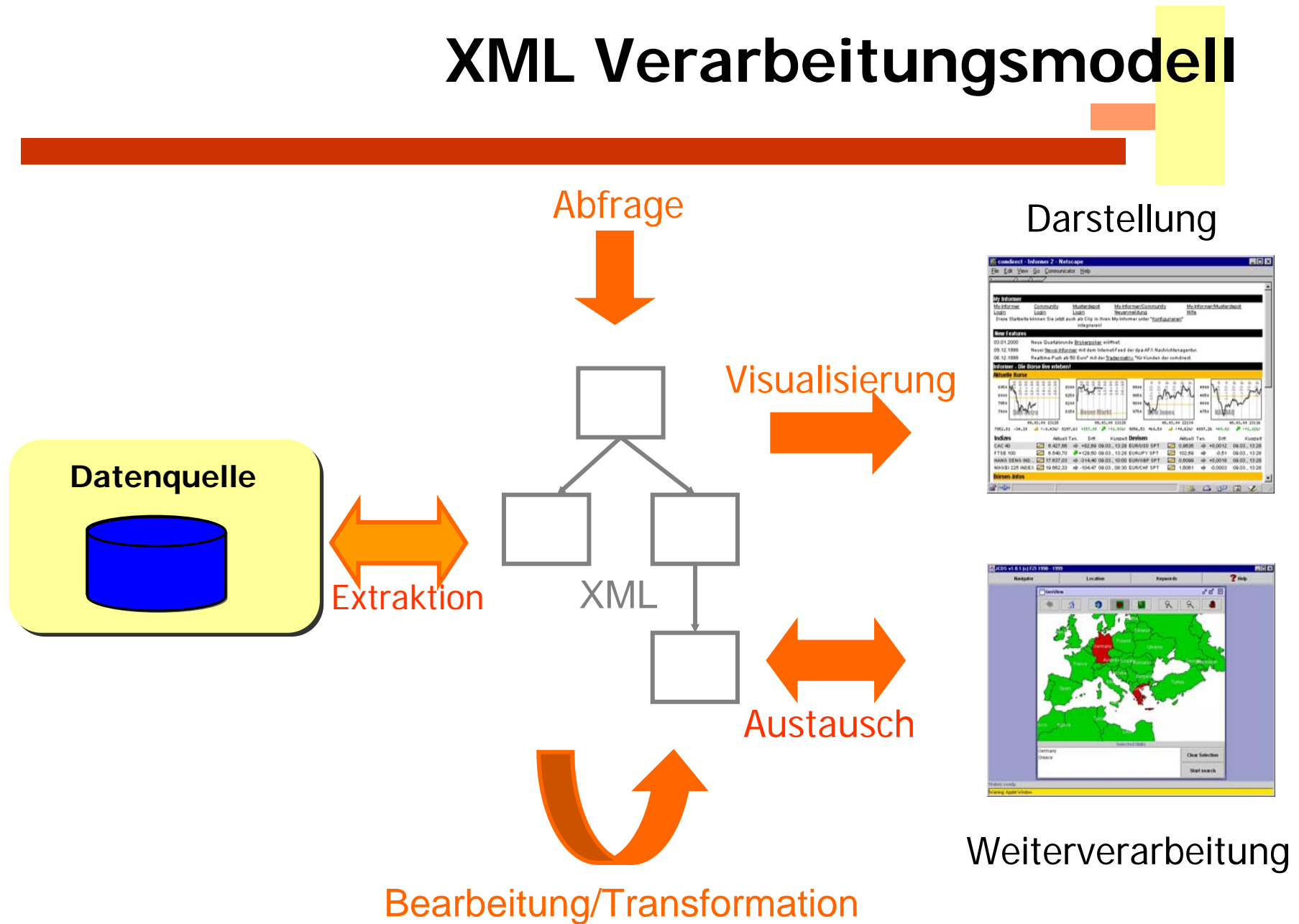


XML-Weg: Motivation

- Zwischen den einzelnen Schichten: Brüchen in den Datenmodellen
 - ▶ Datenbank: relational
 - ▶ Anwendungslogik: objektorientiert
 - ▶ Präsentation: semistrukturiert (XML)

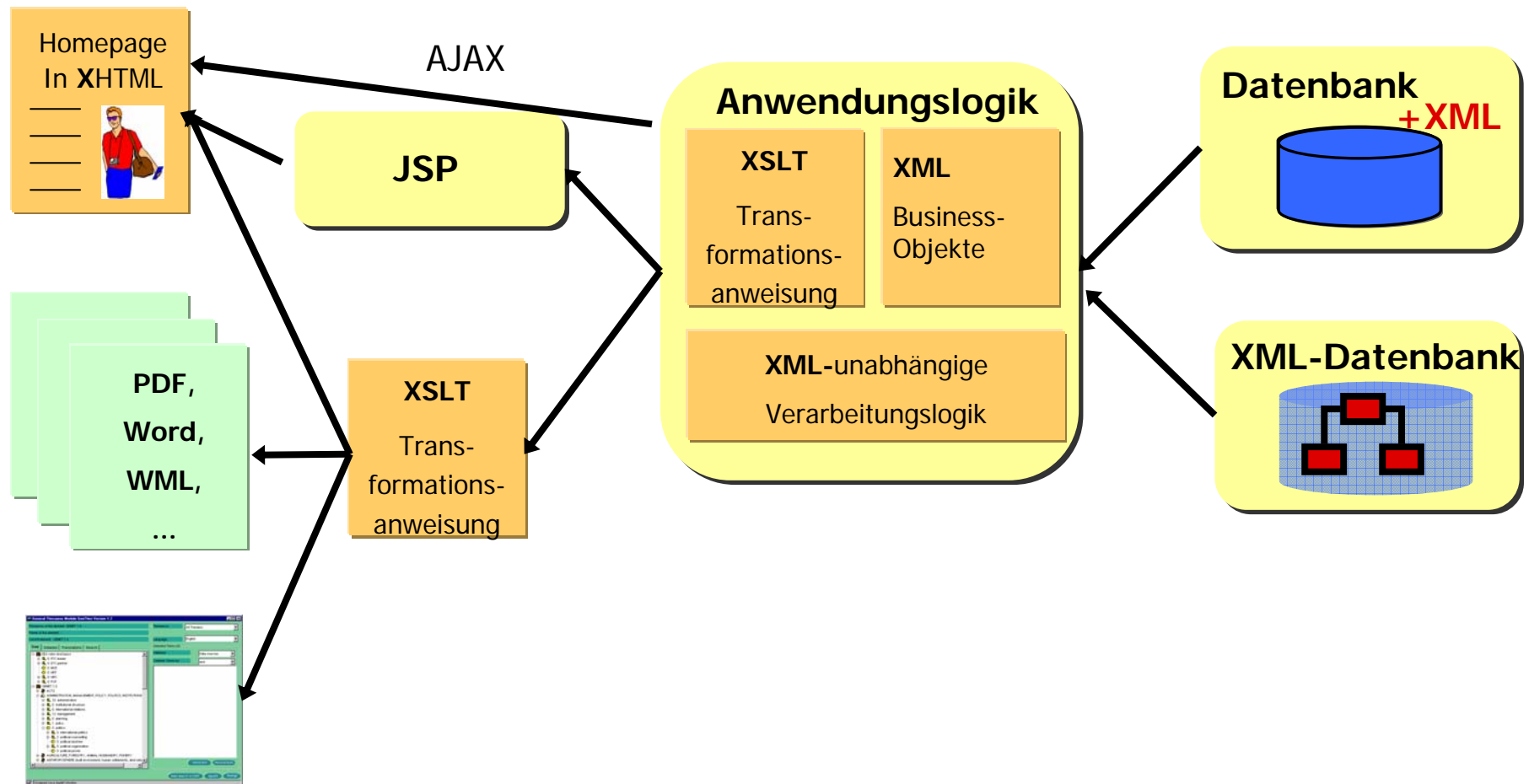
- XML tritt an, dies zu homogenisieren
 - ▶ XML als Datenmodell zur Speicherung
 - ▶ Anwendungslogik als Transformation von XML Daten
 - ▶ Präsentation als Transformation

XML Verarbeitungsmodell

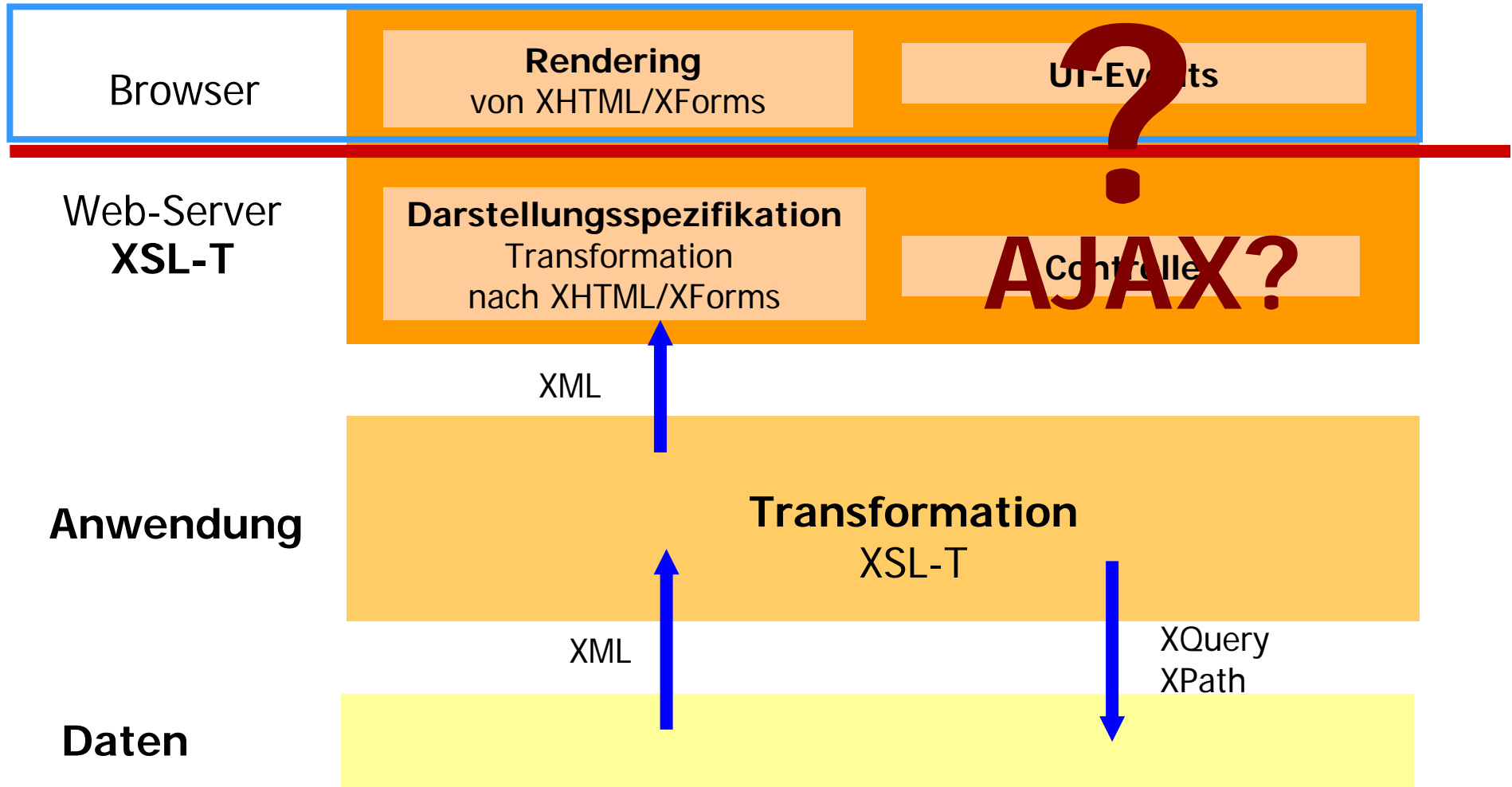


- Auf XML basierendes HTML
- Ziel: Kaum Unterschiede zu HTML 4.0
 - ▶ + Wohlgeformtheit und Gültigkeit, so dass es mit XML-Tools bearbeitet werden kann
 - ▶ + Modularisierung
- Wichtigste Änderungen
 - ▶ Alle Tags müssen abgeschlossen sein (z.B. `<hr/>`)
 - ▶ Keine Überschneidungen, nur Verschachtelung
 - z.B. `<p></p>`
 - ▶ Groß- und Kleinschreibung
 - ▶ Attributwerte in Anführungszeichen
 - ▶ 3 statt 1 DTD für XHTML

Zusammenführung: Vorschau



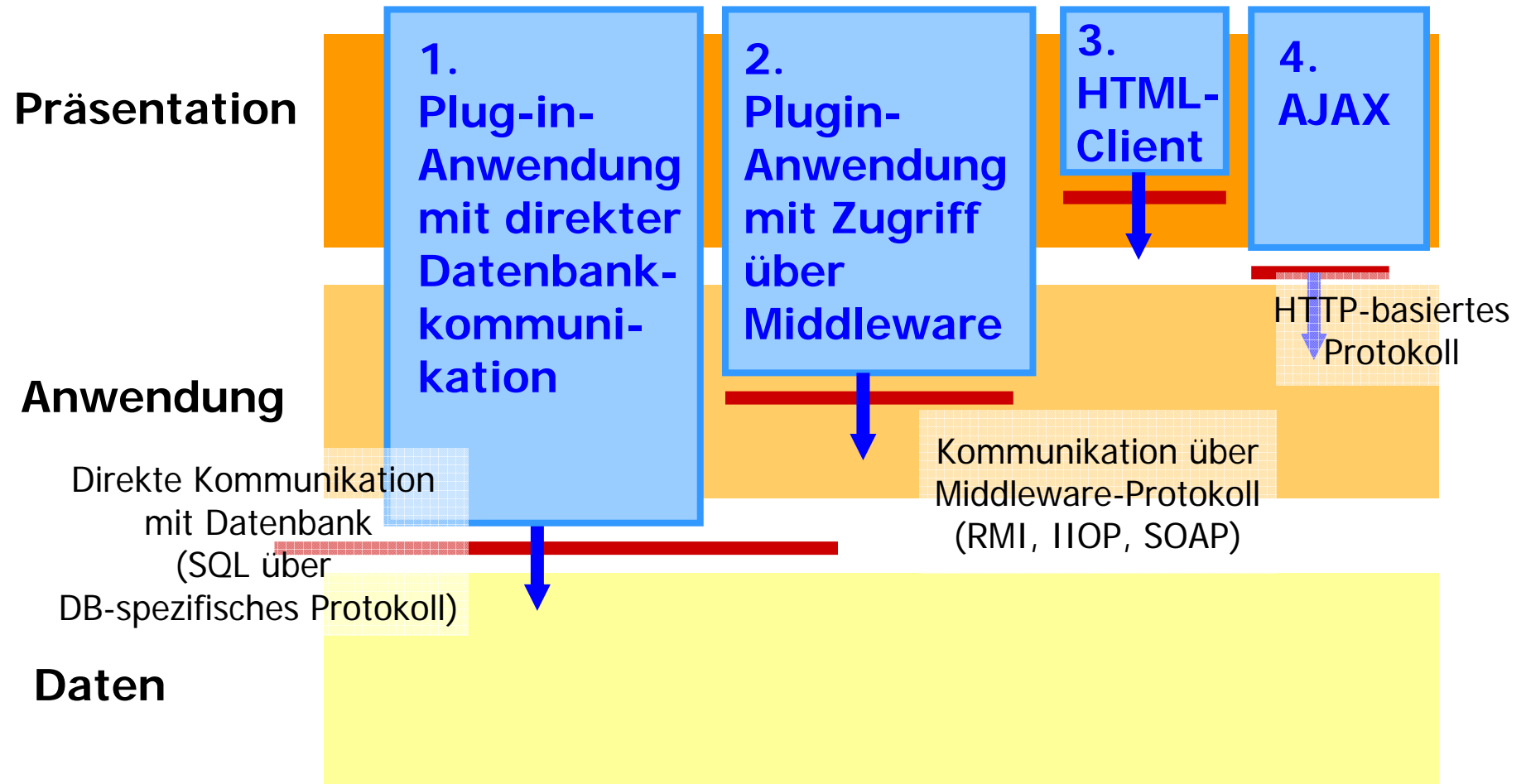
XML-Weg



Client-seitige Technologien



Alternativen: Client



Client-seitige Ansätze (2)

- 1. Plugin: Direkte Datenbankkommunikation
 - ▶ Serverseite stellt Datenbankschnittstelle bereit
 - ▶ wird so gut wie nicht praktiziert
 - ▶ Gründe: Sicherheit, Performanz, Flexibilität
- 2. Plugin: Zugriff über Middleware
 - ▶ Serverseite stellt per RPC Anwendungslogik bereit
 - ▶ in der Praxis fast ausschließlich HTTP(S)-basierte Kommunikation (inzwischen teilweise SOAP)
 - ▶ Gründe: Firewall-Konfigurationen
- 3. HTML-Client
 - ▶ Serverseite ist klassischer Web-Server
 - ▶ Client kümmert sich nur um Darstellung und die Übermittlung von UI-Ereignissen, Server bereitet HTML auf der Basis der Anwendungslogik auf
- 4. AJAX
 - ▶ Serverseite ist teilweise Web-Server, teilweise Anwendungsserver
 - ▶ Client kümmert sich um die komplette Präsentationslogik einschließlich Übersetzung von UI-Ereignissen auf Anwendungslogik

Plugin-basiert

nur "Standard"



AJAX

AJAX: Grundidee



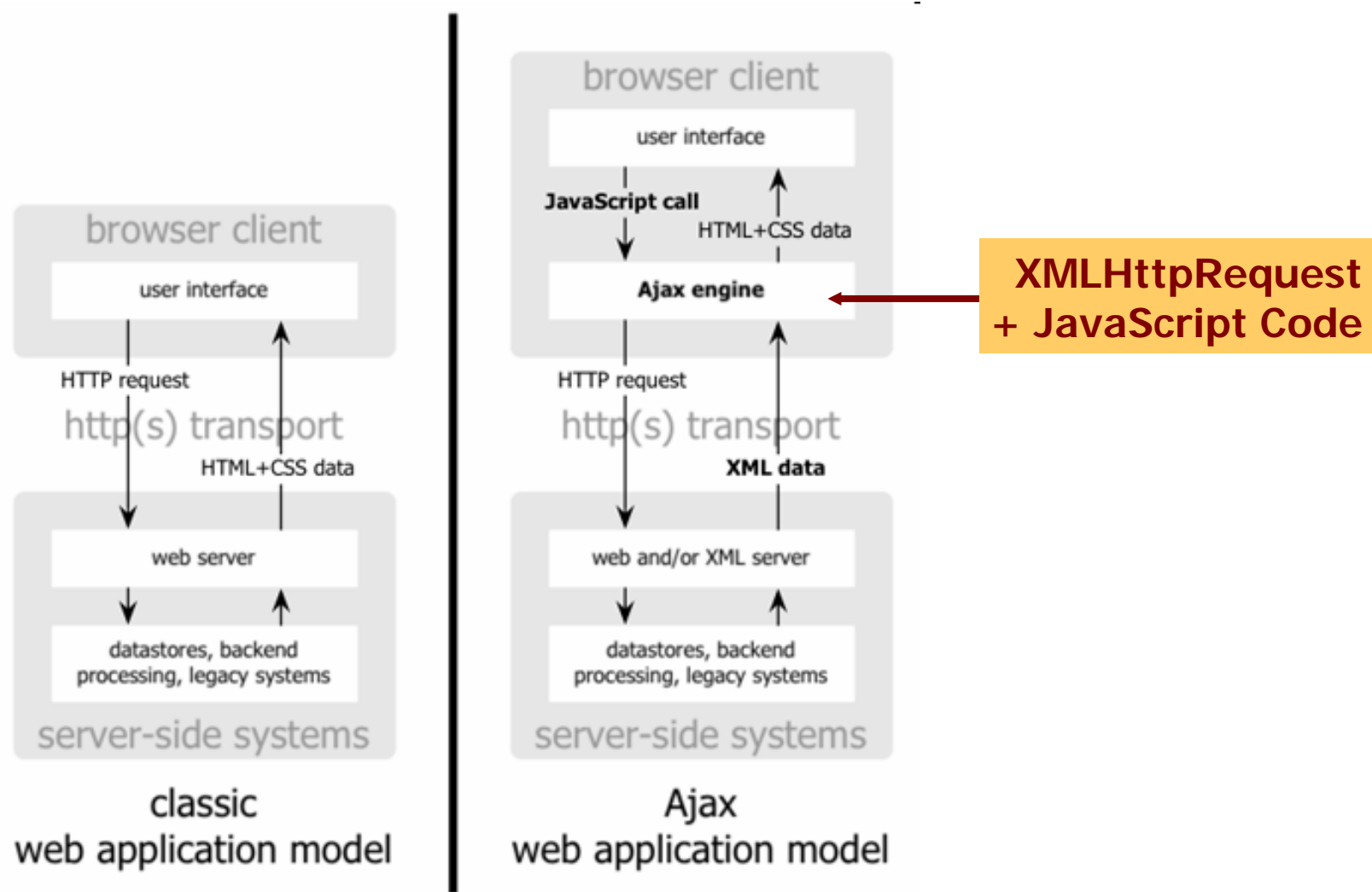
- Hauptproblem bei der Erstellung von komfortablen Web-Anwendungen ist die träge Reaktion
 - für jede Benutzeraktion muss eine Serveranfrage abgesetzt, auf deren Ergebnis gewartet und die Seite neu aufgebaut werden
- AJAX-Idee
 - Benutzerschnittstelle ("Seite") wird nicht immer wieder neu geladen oder aufgebaut, sondern dynamisch modifiziert
 - Server-Anfragen werden asynchron im Hintergrund abgewickelt, so dass der Benutzer nicht warten muss

Was ist AJAX?

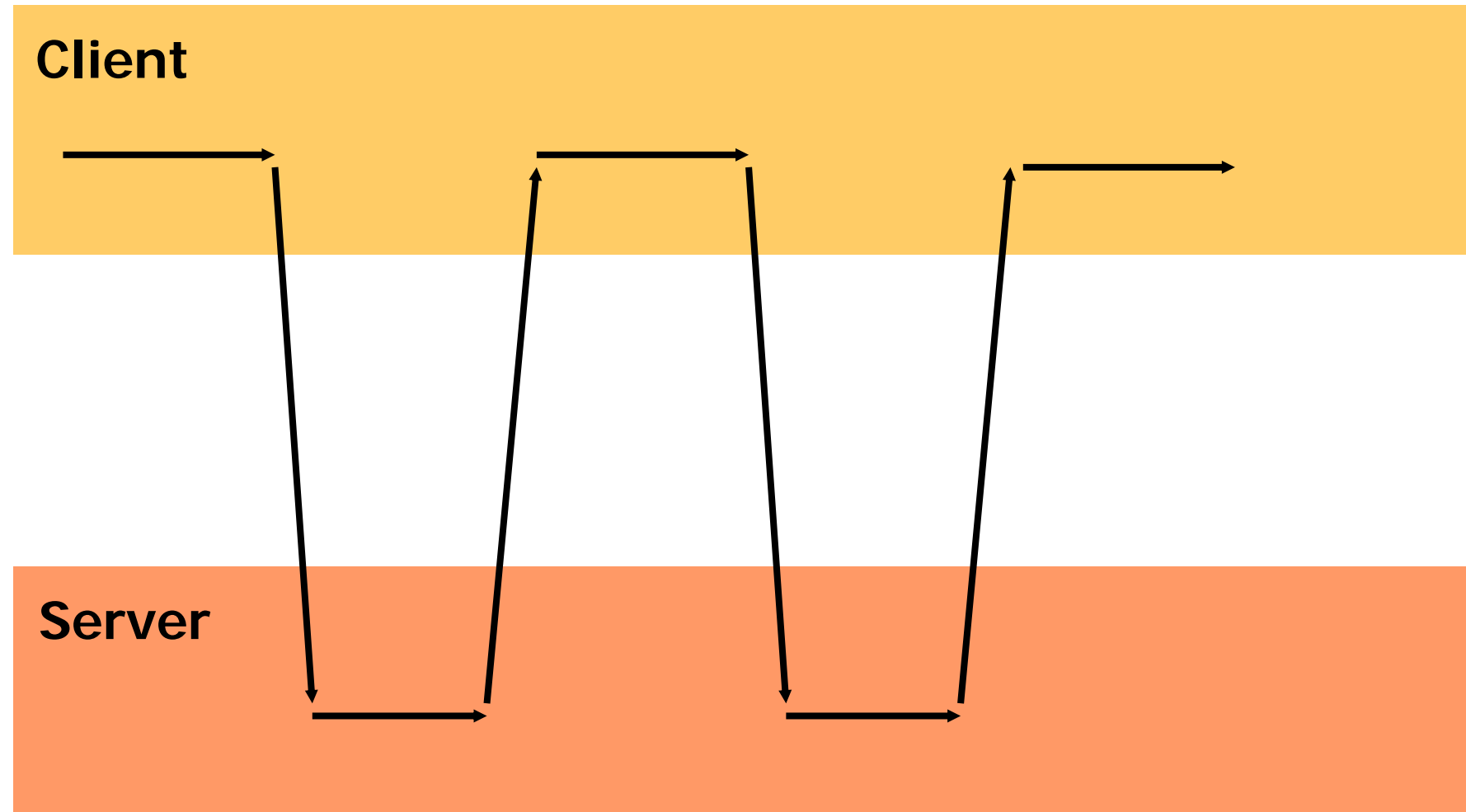


- AJAX (asynchronous Java Script and XML) ist ein Verwendungsmuster von Web-Technologien, die durch asynchrone Serverkommunikation geprägt ist.
- Bausteine
 - XHTML und CSS
 - DHTML, d.h. dynamische Modifikation des Dokumentenbaumes mittels DOM (document object model)
 - Zugriffsmethoden, Ereignismodell
 - XML (und XSLT)
 - asynchrone Datenübertragung mittels XMLHttpRequest
 - JavaScript

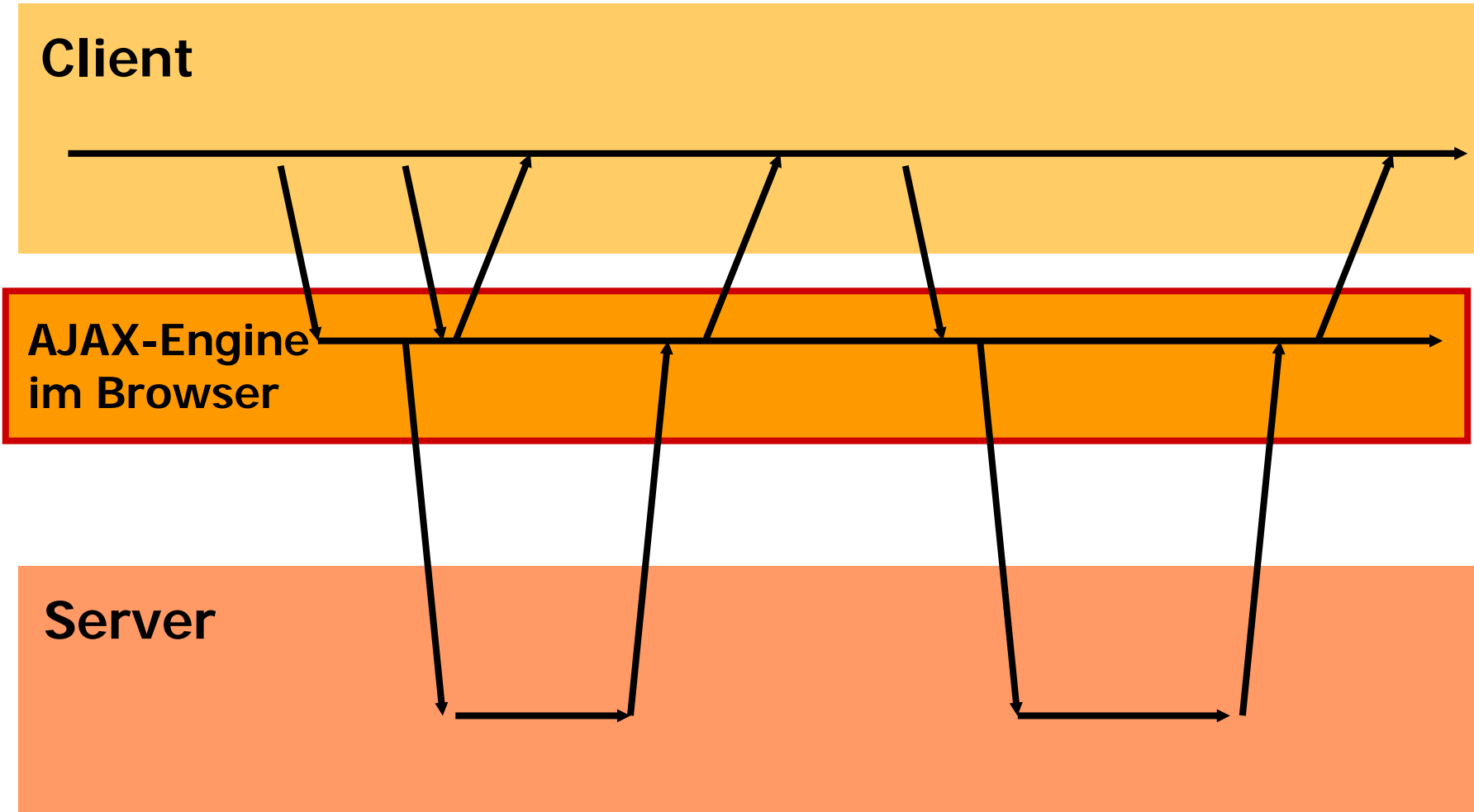
AJAX-Prinzip (1)



Ablauf ohne AJAX

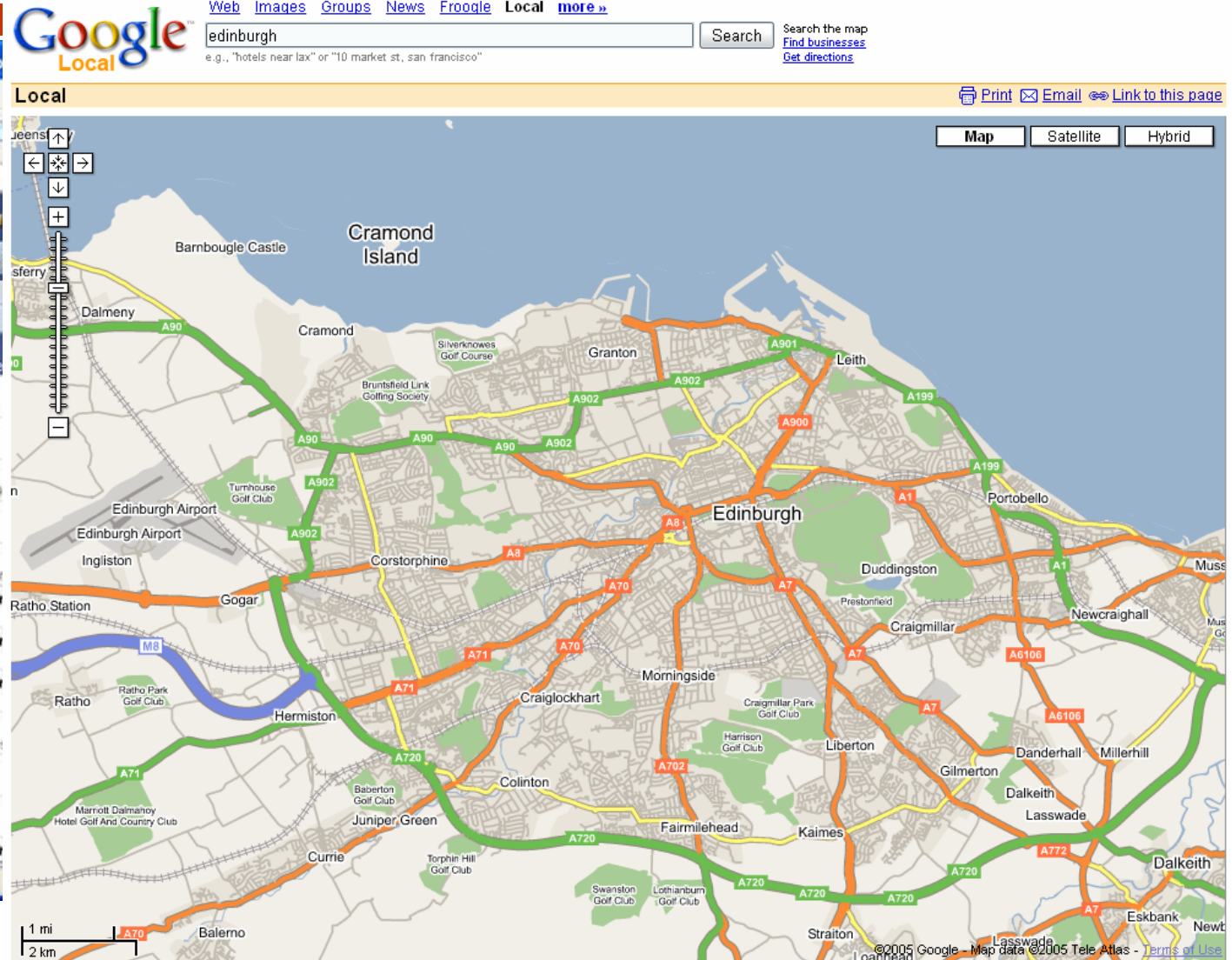


Ablauf mit AJAX



Beispiele

[Help](#)



Grundprinzip

nicht standardisiert!

```
function createXMLHttpRequest() {
  try { return new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) {}
  try { return new ActiveXObject("Microsoft.XMLHTTP"); } catch (e) {}
  try { return new XMLHttpRequest(); } catch(e) {}
  alert("XMLHttpRequest not supported");
  return null;
}
```

```
function loadFragmentIntoElement(fragment_url, element_id)
{
  xmlhttp = createXMLHttpRequest();
  var element = document.getElementById(element_id);
  element.innerHTML = '<p><em>Loading ...</em></p>';
  xmlhttp.open("GET", fragment_url);
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
      element.innerHTML = xmlhttp.responseText;
  }
  xmlhttp.send(null);
}
```

Callback

Typische AJAX-Komponenten/-Funktionen



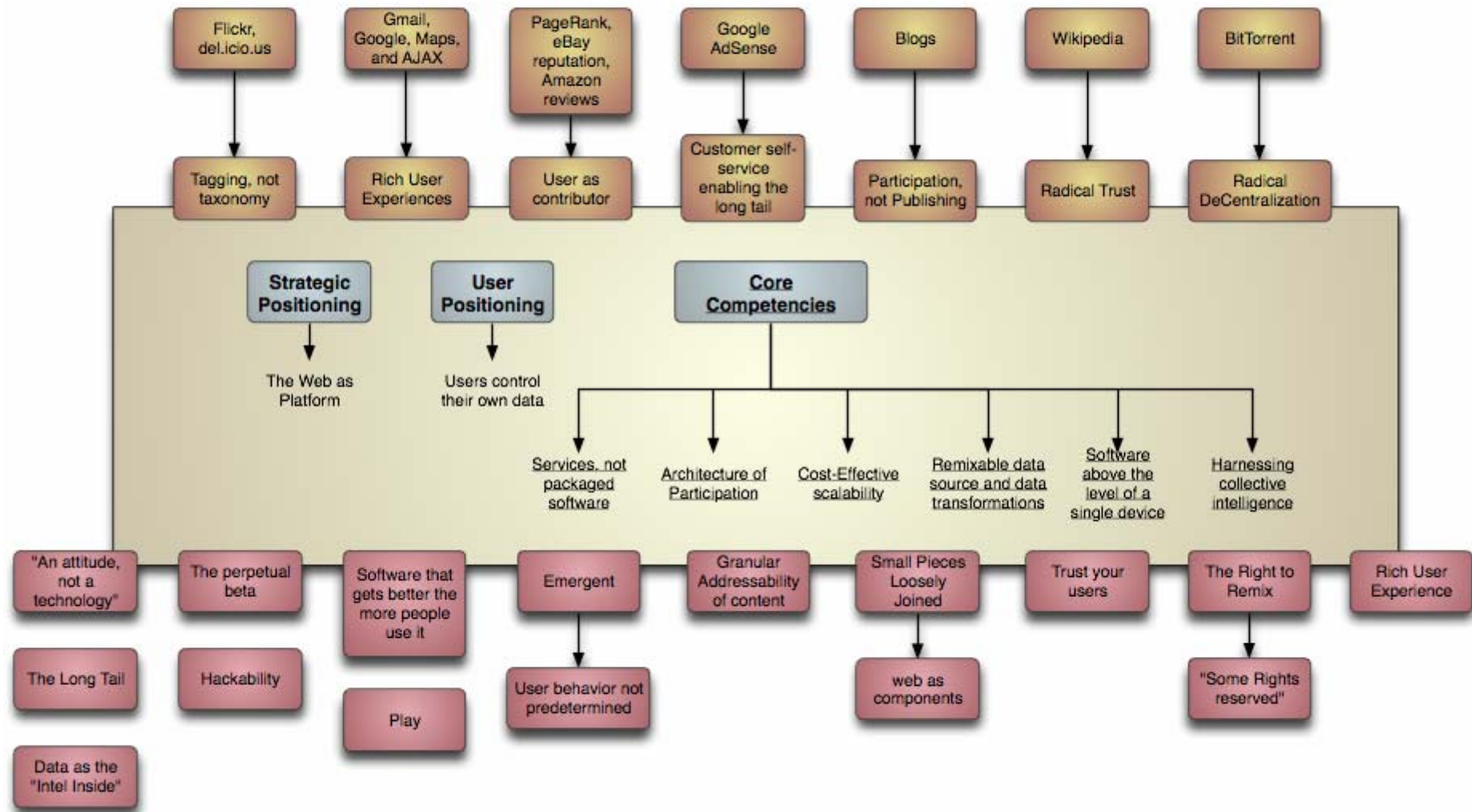
- Autocomplete in Texteingabefeldern
- LiveSearch: Suchergebnisse werden bei der Eingabe angezeigt
- Drag-and-Drop, z.B. in Baumansichten
- Anzeige serverseitiger Änderungen ohne Refresh

- AJAX erlaubt die Erstellung von Web-Anwendungen, die sich (fast) wie Desktop-Anwendungen anfühlen
 - ▶ unterstützt von praktisch allen aktuellen Browsern (IE, Firefox, Opera)
 - ▶ Saubere Trennung von Anwendung und UI möglich
- allerdings:
 - ▶ JavaScript-basierte Entwicklung
 - ▶ führt bei komplexen Projekten zu schlecht wartbarem Code
- zahlreiche Frameworks zur leichteren Erstellung von AJAX-Anwendungen -> generative Ansätze
 - ▶ z.B. auf der Basis von JavaServerFaces o.ä.

- Ist nicht primär ein technologisches Phänomen (insbesondere nicht gleich AJAX), sondern eine neue Wahrnehmung des Webs
 - ▶ Nutzer und die von ihnen generierten Inhalte stehen im Vordergrund
 - ▶ Betonung der sozialen Interaktion
 - darin liegt oft auch der primäre Mehrwert bei der Verlagerung von klassischen Desktop-Anwendungen ins Web
 - ▶ leichtgewichtige Integration: Verwendung der Dienste und/oder Inhalte in anderen Kontexten möglich

Web 2.0 Meme Map

Adapted from O'Reilly Media, FOO Camp brainstorming session, 2005



■ Web-Technologien allgemein

- ▶ C. Strobel: Web-Technologien in E-Commerce-Systemen, Oldenbourg 2003

■ JSP

- ▶ V. Turau, K. Saleck und C. Lenz: Web-basierte Anwendungen entwickeln mit JSP 2, dpunkt Verlag, 2003

■ JSF

- ▶ G. Beneken, F. Deißbröck: Inside Java Server Faces, Java-Spektrum 04/2004,
http://www.sigs.de/publications/js/2004/04/beneken_JS_04_04.pdf

■ XML

- ▶ W. Kazakos, A. Schmidt, P. Tomczyk: Datenbanken und XML, Springer 2002

■ AJAX

- ▶ Jesse James Garrett: AJAX – A New Approach to Web Applications
<http://www.adaptivepath.com/publications/essays/archives/000385.php>