



**Forschungszentrum
Informatik**



Web-Technologien im Überblick

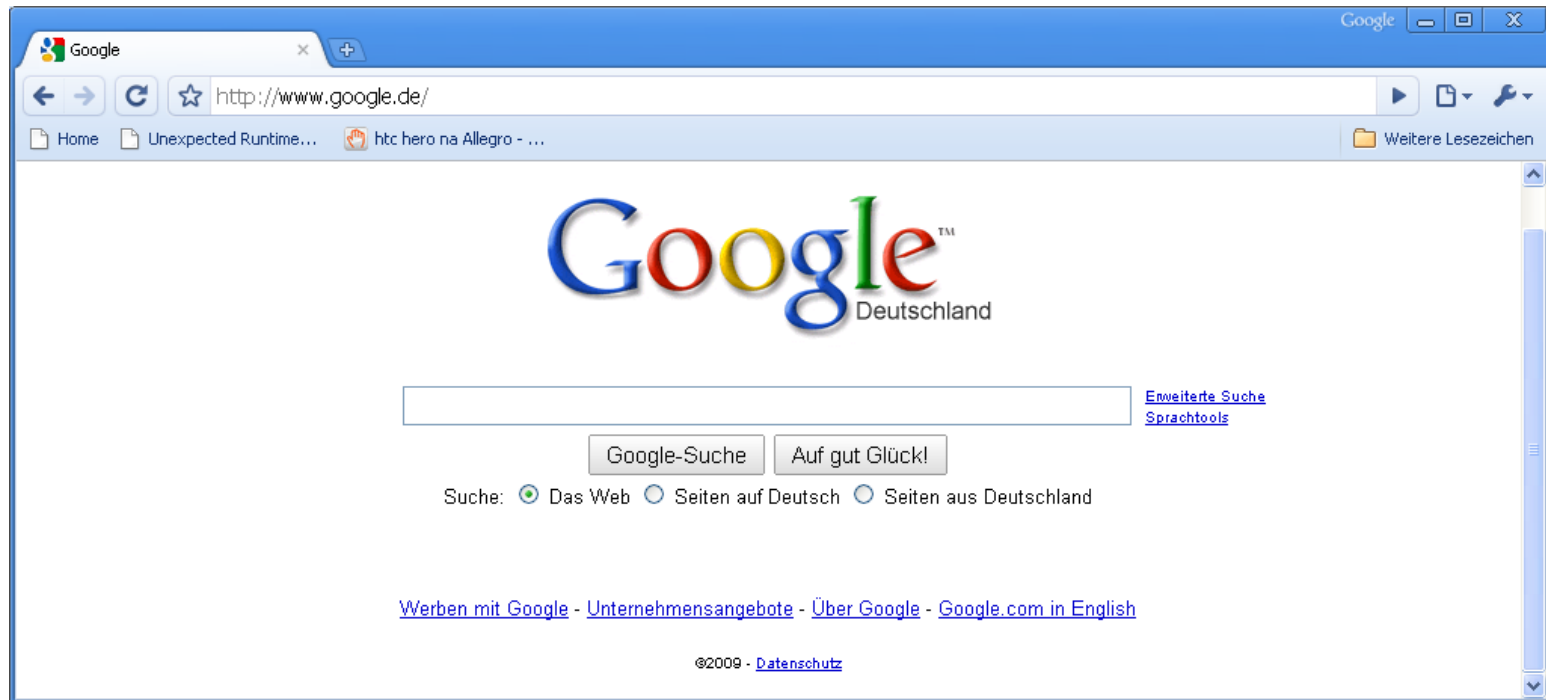
Erstellung von umfangreichen, webbasierten Anwendungen

Simone Braun, FZI

WS 2011/2012

Webanwendungen: Beispiele

- **Statische Webseiten: z.B. Google Startseite**



Webanwendungen: Beispiele

□ Dynamische Webseiten: z.B. Portale zur Flugbuchung

CheapTickets.de

09005564560 | Mo-Fr 9.00-22.00 | Sa 9.00-17.00
0900 150199 | Mo-Fr 9.00-22.00 | Sa 9.00-17.00
0900 556045 | Mo-Fr 9.00-22.00 | Sa 9.00-17.00

Stellen Sie Ihre Frage hier **FRAGEN**

Linienflüge | Hotels | Mietwagen | Angebote

Hinreise
Abflughafen: Frankfurt
Zielflughafen: Krakow
Hinflugdatum: 22 Dez 2009
 Hin/Rückflug nur Hinflug

Rückreise
Abflughafen: Krakow
Zielflughafen: Frankfurt
Rückflugdatum: 29 Dez 2009

Passagiere
Erwachsene(r): 2
Kinder: 0
Kleinkinder: 0

SUCHEN

weitere Suchoptionen

schritt 1 SUCHEN | **schritt 2 FLUGTARIFE** | schritt 3 BUCHUNG | schritt 4 BEZAHLUNG | schritt 5 BESTÄTIGUNG

Frage und Antwort

1 - 8 von 39 | 1 | 2 | 3 | 4 | 5 | Folgende 8 >

Tarife pro Person			
Online reservieren	Germanwings	(175,45 EUR + 27,54 EUR Tax) 202,99 EUR	Direktflug
Online reservieren	Czech Airlines	(114,00 EUR + 103,82 EUR Tax) 217,82 EUR	
Online reservieren	Czech Airlines	(117,00 EUR + 142,00 EUR Tax) 259,00 EUR	
Online reservieren	LOT-Polish Airlines	(120,00 EUR + 128,95 EUR Tax) 248,95 EUR	

Sicher buchen bei CheapTickets.de

IATA

Webanwendungen: Beispiele

□ Webseiten im Stil von Desktop-Anwendungen

The screenshot displays the Google Calendar BETA interface. At the top, there is a search bar with two buttons: "Search Public Calendars" and "Search My Calendars", and a link for "Show Search Options". Below the search bar, the "Create Event" link is visible. The main calendar view shows a week starting on Monday, October 29, 2007, and ending on Thursday, November 1, 2007. The time slots range from 04:00 to 11:00. A pop-up dialog box is overlaid on the calendar, showing the event title "Mon, October 29, 07:30 - 08:30" and a "What:" field with the example text "e.g., Breakfast at Tiffany's". The dialog also includes "Create Event" and "edit event details" buttons.

Google Calendar BETA

Search Public Calendars Search My Calendars Show Search Options

Create Event

Quick Add

« October 2007 »

M	T	W	T	F	S	S
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

▼ Add

My Calendars

✓ Andreas Walter

Other Calendars

Manage calendars

Today Oct 29, 2007 - Nov 4, 2007

Mon 29/10 Tue 30/10 Wed 31/10 Thu 1/11

Mon, October 29, 07:30 - 08:30

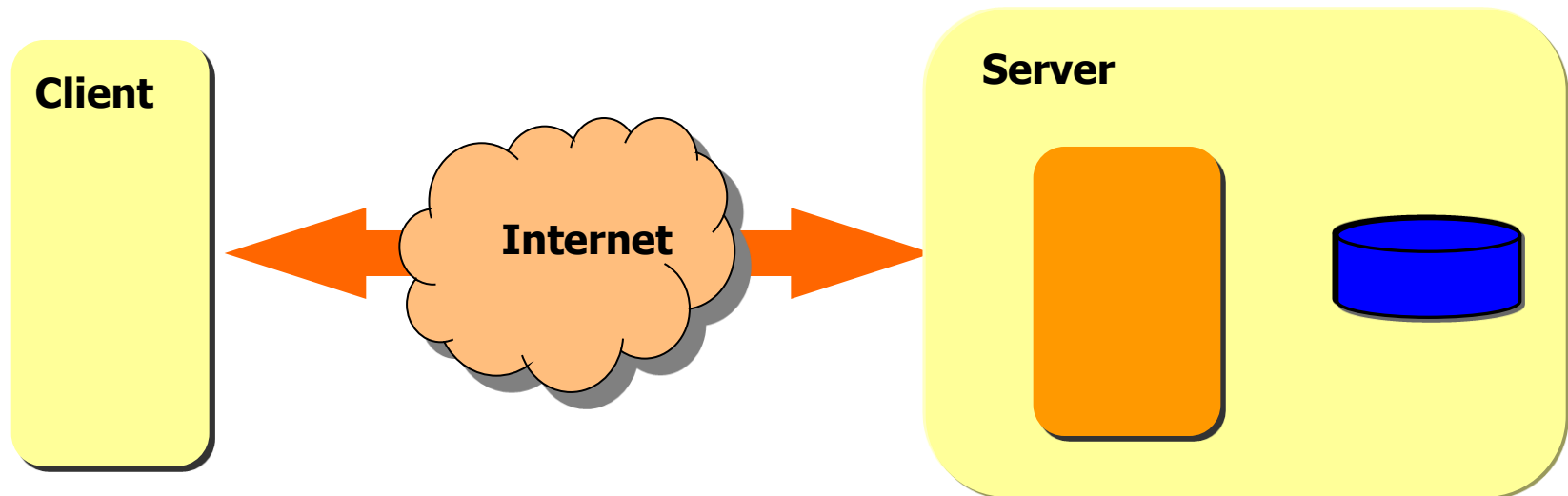
What:

e.g., Breakfast at Tiffany's

Create Event edit event details »

Ziel dieser Vorlesung:

Technologien und Anforderungen zur Erstellung von kleinen und umfangreichen Webanwendungen



- Grundlagen
- Desktop vs. Webanwendungen
- Webtechnologien für kleinere Webanwendungen
- Anforderungen bei der Erstellung umfangreicher Webanwendungen
- Konzepte von Webrahmenwerken

Grundlagen Basistechnologien im Web

Worauf basieren Webanwendungen?

http: Transportprotokoll im WWW

- **http:** HyperText Transfer Protocol (aktuell Version 1.1)
 - Von W3C standardisiertes Protokoll zum Abruf und Senden von Daten von / an einen Web-Server (RFC 2616)

Ursprünglich: Webseiten, Grafiken

Mittlerweile: auch für WebServices, Skype (Telefonie)

- Grund: Firewalls müssen Port 80 freigeben, um Abruf von Webseiten zu ermöglichen

Zustandsloses Protokoll (auf TCP/IP basierend): Jeder Abruf von Daten erfordert eine neue Verbindung (http 1.1 ermöglicht „keep alive“ – nur für z.B. Chats sinnvoll)

Bereich MESSAGE HEADER: Für Metainformationen (z.B. Anfragemethode, Statuscodes, Dokumenttyp, Länge des Dokuments), Authentifizierung

Bereich MESSAGE BODY: folgt dem HEADER, enthält eigentlichen Inhalt

Adressierung mit http-Protokoll

- **URI: Uniform Resource Identifier**
Identifizierung und Adressierung von Ressourcen

*<Schema>://[<Benutzer>[: <Passwort>]@]
<Server>[: <Port>][/ <Pfad>][? <Anfrage>][# <Fragment>]*

- Davon abgeleitet: Uniform Resource Locator (URL) für http
z.B. <http://www.fzi.de> (bedeutet <http://www.fzi.de:80/>)
- httpRequest GET-: senden von Parametern, z.B. über Adressfeld des Browsers:
<http://www.fzi.de/search.php?autor=Braun&ordering=newest>
 - Einsatz: für Seiten, die über ein Skript generiert werden und gebookmarkt werden sollen
 - Daten als Teil der URL bleiben erhalten
- httpRequest POST: senden von umfangreichen Datenmengen, Authentifizierung aus Formular heraus.
 - Datenübertragung im BODY Bereich von http
 - Keine Speicherung der übertragenen Daten möglich

Beispiel Messages

□ HTTP GET Request:

```
GET /search.php?autor=Braun&ordering=newest HTTP/1.1
Host: www.fzi.de
```

□ HTTP POST Request:

```
POST /search.php HTTP/1.1
Host: www.fzi.de
Content-type: application/x-www-form-urlencoded
Content-length: 27
```

```
autor=Braun&ordering=newest
```

HEADER

BODY

Zum Anschauen von http-Anfrage/Antworten:
<http://web-sniffer.net/>

Beispiel Messages

□ HTTP Response

```
HTTP/1.1 200 OK
Date: Sun, 23 Oct 2011 10:23:03 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Content-Type: text/html; charset=utf-8

<html xml:lang="de" ... </html>
```

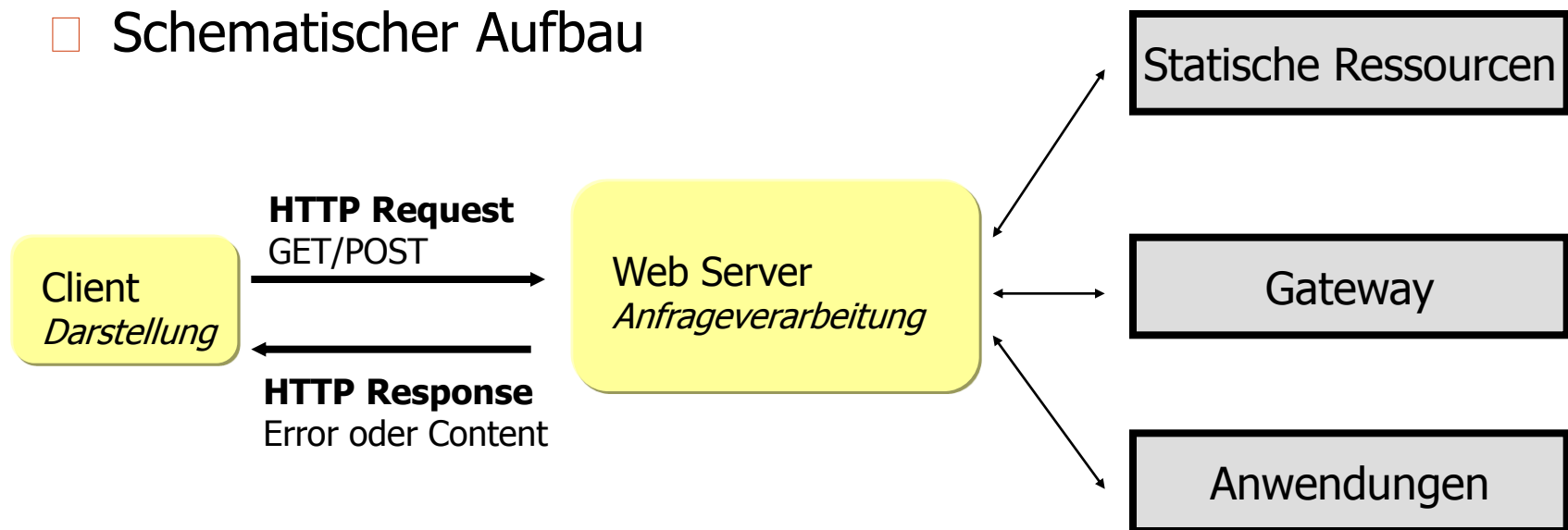
```
HTTP/1.1 404 Not Found
Date: Sun, 23 Oct 2011 10:23:03 GMT
Server: Apache/2.2.3 (CentOS)
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

□ Statuscodes: 5 Klassen

- 1xx – informatorisch
118 Connection timed out
- 2xx – erfolgreich
200 OK
- 3xx – Umleitung
301 Moved Permanently
- 4xx – Client-Fehler
403 Forbidden
- 5xx – Server-Fehler
503 Service Unavailable

- Setzt http-Protokoll um, erreichbar über Port (z.B. 80)
 - Bearbeitung von Client Anfragen, bearbeitet diese und gibt wenn möglich einen Inhalt zurück, sonst http-Fehlercode

□ Schematischer Aufbau



- Programm zum Abruf und zur Darstellung von Inhalten aus dem Internet
 - z.B. Abruf von Webseiten über das Protokoll http, zumeist auch andere Protokolle wie z.B. ftp (file transfer) integriert.
 - W3C Konsortium definiert Standards, welche alle Webbrowser können soll(t)en
 - HTML, CSS, JavaScript, XSLT
 - **Cache:** dient zur Speicherung von häufig verwendeten Inhalten, reduziert Datentransfer mit Server
 - **Cookies:** Über http können z.B. sessionrelevante Inhalte im Browser abgelegt werden, z.B. sessionID, Username, bevorzugte Sprache
 - **Plugins:** Browser sind über Plugins erweiterbar, z.B. Flash

□ HTTP Response

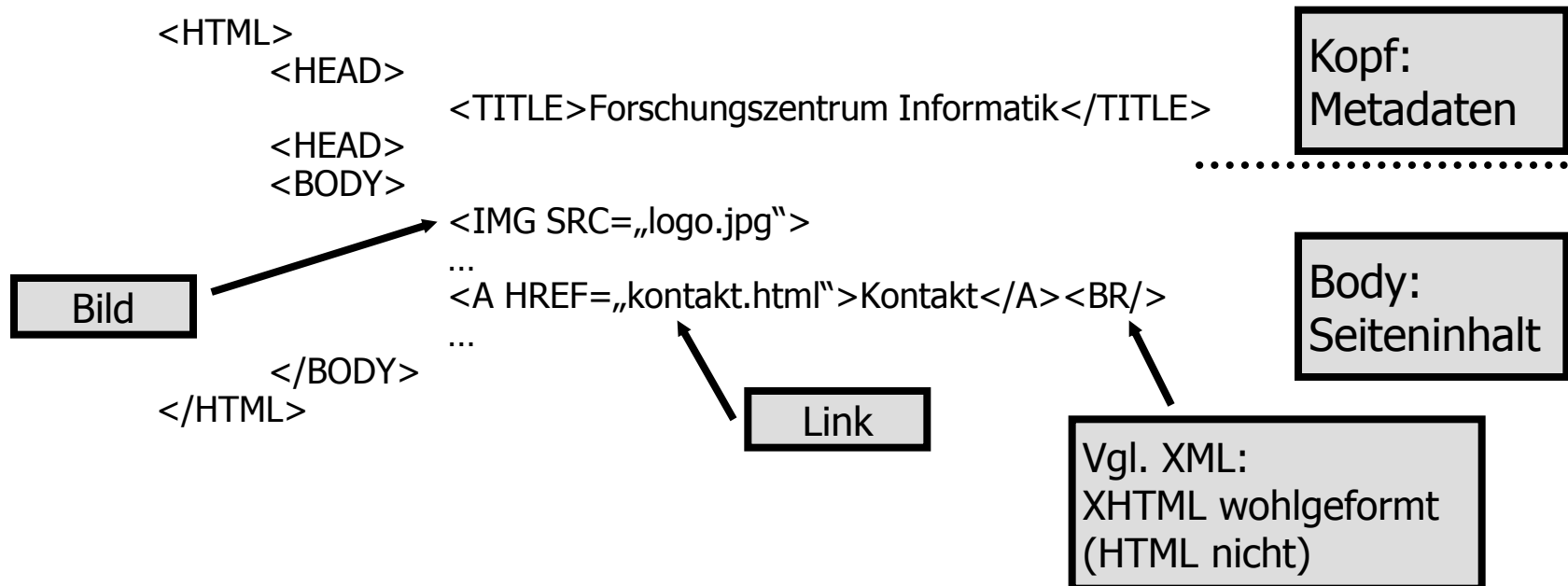
```
HTTP/1.1 200 OK
Date: Sun, 23 Oct 2011 10:23:03 GMT
Server: Apache/2.2.3 (CentOS)
Set-Cookie: d7ae737f0e17da5a64121bfb8fbe92c0=j4hcsnpjhg2fbfpucb6242a9t0; path=/
Set-Cookie: lang=deleted; expires=Sat, 23-Oct-2010 10:28:48 GMT; path=/
Expires: Mon, 1 Jan 2001 00:00:00 GMT
Last-Modified: Sun, 23 Oct 2011 10:28:50 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html; charset=utf-8

<html xml:lang="de" ... </html>
```

Formatierungsanweisung: (X)HTML (1)

□ Hyper Text Markup Language

Textbasierte Formatierungssprache zur Bestimmung der Darstellung von Inhalten wie Texten, Bildern und Hyperlinks zu anderen Dokumenten in einem Browser

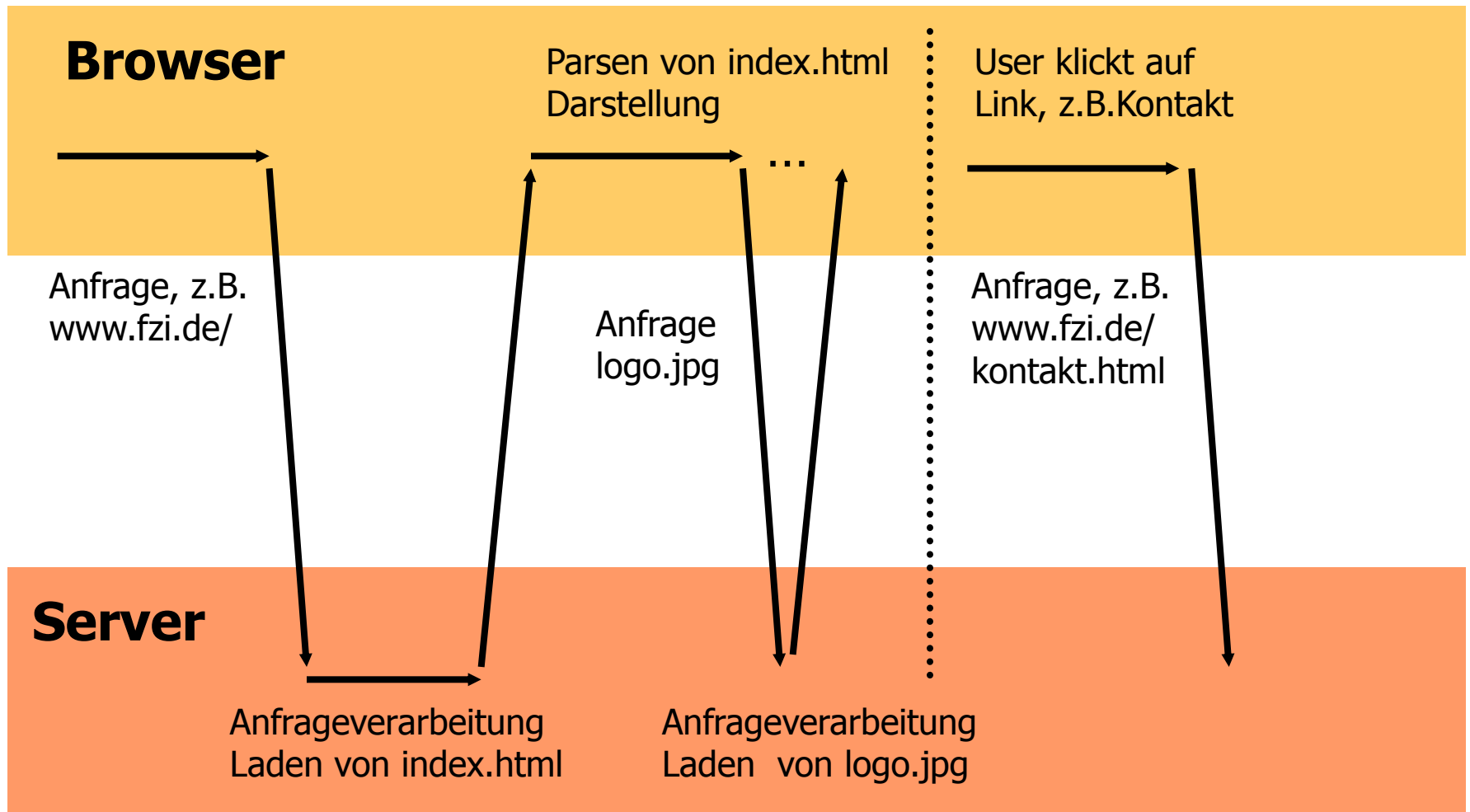


- Browser parst (X)HTML und zeigt dieses im Browser an



- Weitere wichtige Elemente: Tabellen, Ebenen, Formulare

Kommunikation zwischen Browser und Webserver



Widerverwendung von Darstellung: CSS

□ Cascading Style Sheets

Legt fest, wie ein besonders ausgezeichneter Inhalt oder Bereich einer Webseite dargestellt werden soll.

- Trennung von Inhalt (HTML, XML) und Darstellung
- CSS als eigene Ressource ladbar (z.B. stylesmobile.css)

Ermöglicht unterschiedliche Ausgabe der gleichen Seite für verschiedene Medien (Browser, PDA, Mobiltelefon, ...)

□ Beispiel:

HTML: `text mit css`

CSS: `.demostyle {color:black;background:red;font-weight: bold;}`

Ausgabe: **text mit css**

Skripte in einer Webseite: Java Script

- **JavaScript** (!= JAVA!!!) ist objektbasierte Skriptsprache, direkt in HTML integrierbar oder als Ressource integrierbar
 - Ermöglicht die Client-seitige Ausführung von Skripten, durch Browser interpretiert und umgesetzt, z.B.
 - Eventhandling, z.B. Validierung von Formulareingaben
 - Interaktive Menüs, Drag & Drop Funktionen
 - Aktualisierung von Seitenteilen
 - In Verbindung mit XML und httpRequest => AJAX (später)
- **Problem:** JavaScript ist schwer wartbar
 - Verhalten von JavaScript in Browsern nicht immer einheitlich
 - Wiederverwendung von Bibliotheken statt Selbsterstellung von Skripten wann immer möglich!

Java Script Beispiel

□ Javascript in HTML Text eingebettet

```
<script type="text/javascript">
  function checkFormValues()
  {
    if (pw1 != pw2)
    {
      alert(„reenter password and pw are not equal“)>
      return false;
    }
  }
}

<form name=„loginForm“ onSubmit=„checkFormValues()“>
  PW: <input type=„password“ name=„pw1“/>
  Reenter: <input type=„password“ name=„pw2“/>
  <input type=„submit“ value=„login“/>
</form>
```

Passwort wählen:

Muss mindestens 8 Zeichen umfassen.

Passwort nochmals eingeben:

Skript bewirkt: Überprüfung von eingegebenen Passwörtern

Zusammenfassung Grundlagen

- **http**: Protokoll im Internet, Adressierung mit URLs
- **WebServer**: Serverseite, erweiterbar (z.B. Gateways)
- **Browser**: Clientseite, dient zur Darstellung von Inhalten
- **XHTML**: dient zur Formatierung von Webseiten
- **Kommunikation**: zustandslos, Request – Response Prinzip
- **CSS**: dient zur Trennung von Inhalt und Formatierung
- **JavaScript**: ermöglicht Ausführung von Skripten im Browser

Desktop vs. Webanwendungen

... Die Unterschiede verschwinden

Desktop vs. Webanwendungen (I)

- **Desktop-Anwendungen** (ohne Netzwerkkommunikation)
 - „Fat-Client / Rich-Client“: Anwendungslogik, Darstellung (und ggf. Datenhaltung) in einer Software
 - + Zustandshaltung, z.B. geöffnetes Worddokument
 - + Umfangreiche Menüfunktionen
 - + Schnelle Reaktion auf Eingaben durch Aktualisierung von einzelnen Teilen einer Seite.
 - + Darstellung: Programmierer können Standardbibliotheken verwenden (Swing (JAVA) /...)
 - Meist sehr umfangreiche Installation erforderlich (in Firmen hierfür Kontakt mit SysAdmin nötig)
 - Updates nur manuell

Desktop vs. Webanwendungen (II)

- **Desktop-Anwendungen** (mit Netzwerkkommunikation)
 - Datenhaltung und (Teile der) Anwendungslogik ausgelagert, Software auf Client dient zur Darstellung
 - + Weiterhin clientseitige Zustandshaltung (z.B. einmaliges Laden von Terminen -> Darstellung als Kalender)
Kommunikation mit Server nur zur Synchronisation
 - + Client weniger umfangreich, Aktualisierung der Software bei jedem Start möglich (z.B. JAVA Web Start)
 - Weiterhin lokale Installationen notwendig
 - Entweder Interpreter (z.B. kompatible JAVA-Version)
 - oder Plugins (z.B. Flash Player)

Desktop vs. Webanwendungen (III)

□ **Browserbasierte Webanwendungen:**

- Webbrowser auf Clientseite wird zur Darstellung der Anwendung verwendet, Server für die Datenhaltung und Anwendungslogik

+ Keine Installation von Software auf Clientseite nötig
(Sämtliche Betriebssysteme für PCs beinhalten Browser)

„Web 1.0“ Stil:

- Eine Webseite wird komplett aktualisiert für jede Anfrage (statisch)
- Ermöglicht keine Anwendungen mit „Desktop-feeling“

„Web 2.0“ Stil

- Aktualisierung von Seitenteilen möglich („Desktop-feeling“)
- + Ermöglicht die Erstellung von umfangreichen Webanwendungen im Stile einer Desktopanwendung

- Webanwendungen werden Desktopanwendungen immer ähnlicher, profitieren aber von den Vorteilen schlanker Clients
 - Verwendung des Browsers zur Darstellung, keine Installation von Plugins oder eigenständiger Software
 - Client immer auf neuestem Stand
 - Hauptgründe und Techniken hierfür:
 - JavaScript und CSS
 - AJAX (Details später)
 - Breitbandzugänge (schnelles Laden von Seitenteilen möglich)**

Web-Technologien für kleinere Webseitenprojekte

... hier ist „skripten“ noch sinnvoll

Merkmale kleiner Webseitenprojekte

- Hoher Anteil an statischen (gleich bleibenden) Webseiten
- Merkmale der dynamischen Webseiten
 - Anwendungslogik ändert sich kaum bis selten
Wartbarkeit ist somit keine Anforderung
 - Gleichzeitige Teilnehmerzahl für Nutzung von dynamischen Seiten gering – Skalierbarkeit also keine Anforderung
- **Vorgehen generell**
 - Design der Webseite und statische Seiten erstellen (Grafiker)
 - Programmierer wählt eine geeignete Skriptsprache aus und erstellt Anwendung
 - Webspaces bei Internetprovider anmelden, die Sprache unterstützt
 - Aufladen der Dateien (Daten) und Skripte: Fertig!

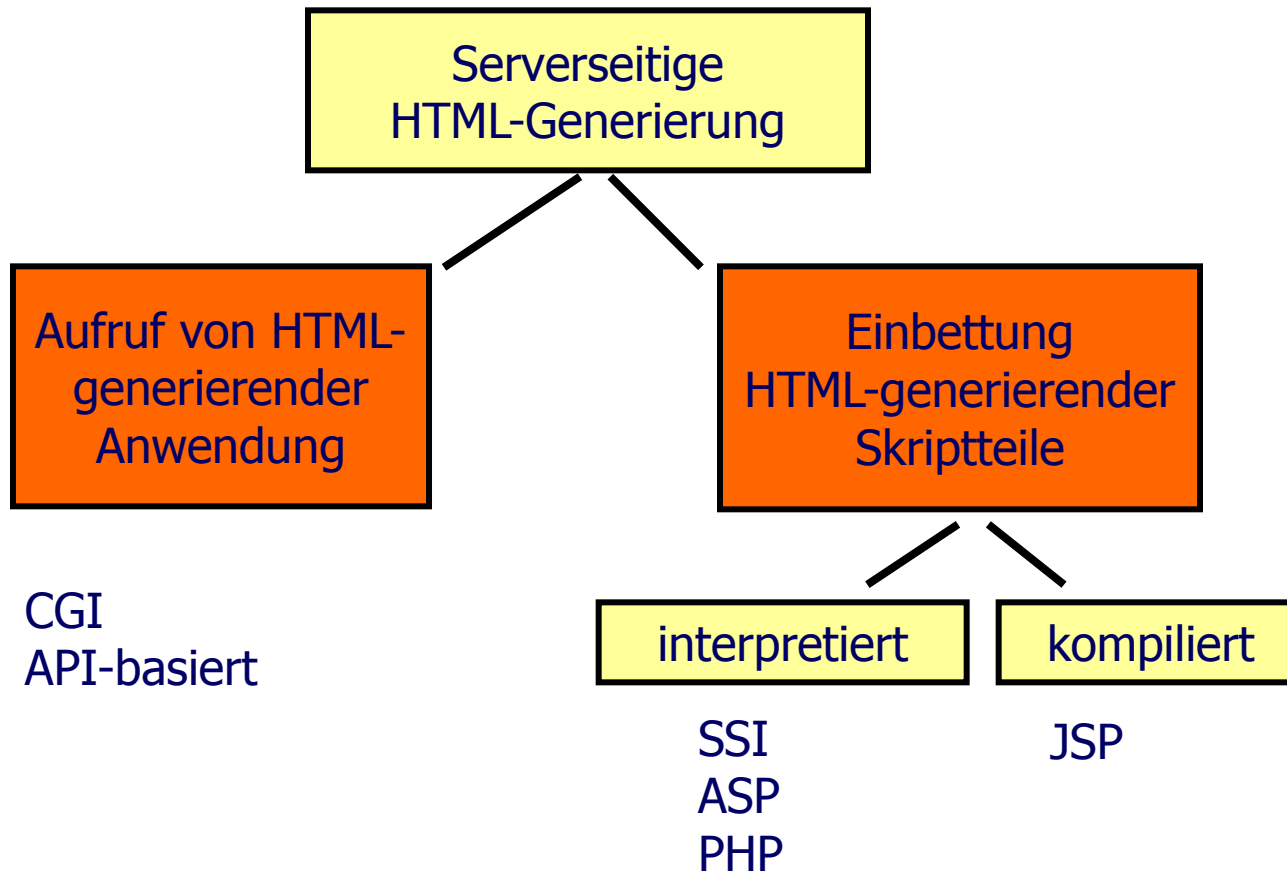
Vorgehen zur Erstellung dynamischer Seiten

Anwendung:

- Für eine geplante Feier sollen sich Freunde und Verwandte online anmelden, um Catering planen zu können

- 1. HTML Formular erstellen, das Datenfelder beinhaltet,
 - z.B. Name, Kontakt, Anzahl Personen, Auswahl Menüs, Vegetarisches Essen gewünscht, ...
- 2. Skript erstellen, das diese Werte einliest
- 3. Anwendungslogik:
 - überprüfe, ob Teilnehmer bereits registriert ist
 - wenn nein, speichere Daten ab
- 4. Dynamisches Generieren einer Antwortseite
 - z.B. „Danke für Deine Registrierung“ oder „Du bist bereits registriert, möchtest du Deine Daten ändern?“

Serverseitige Anwendungslogik und Datenhaltung

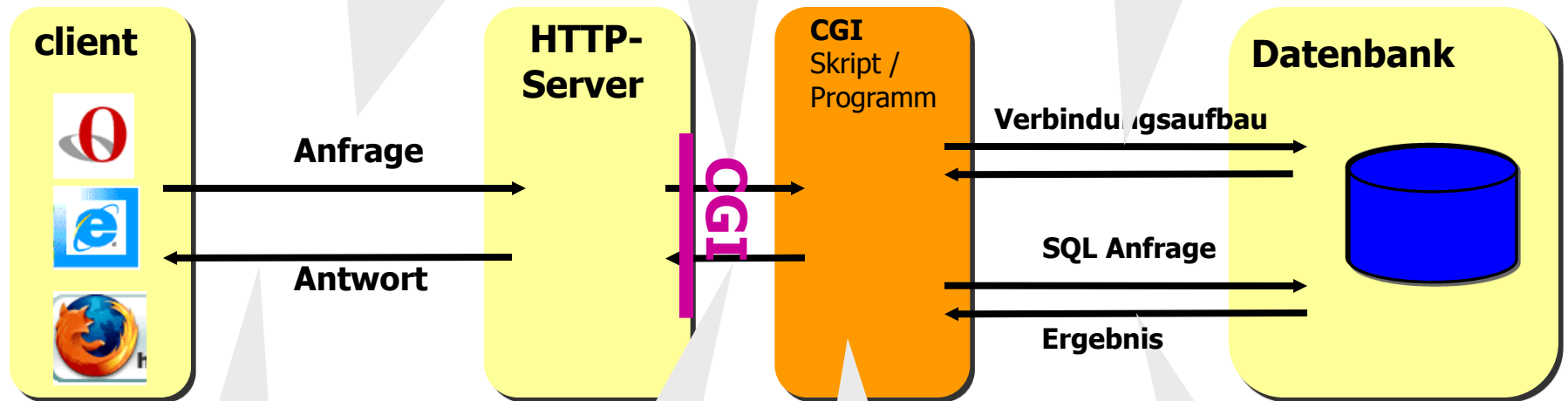


Die Old-School Variante: CGI

1) Betätigung Submit-Button
<FORM METHOD="POST"
ACTION="http://www.fzi.de/
cgi-bin/my-form">

2) Aufruf des Skriptes
mit Parametern
**Initialisierung bei
jedem Aufruf**

3) Aufbau der **DB-
Verbindung bei
jedem Aufruf**



7) HTTP Server
Antwortet

6) Skript übermittelt
HTML und **beendet
sich**

5) Skript generiert
HTML

4) Skript stellt Anfrage
an DB

Die Old-School Variante: CGI

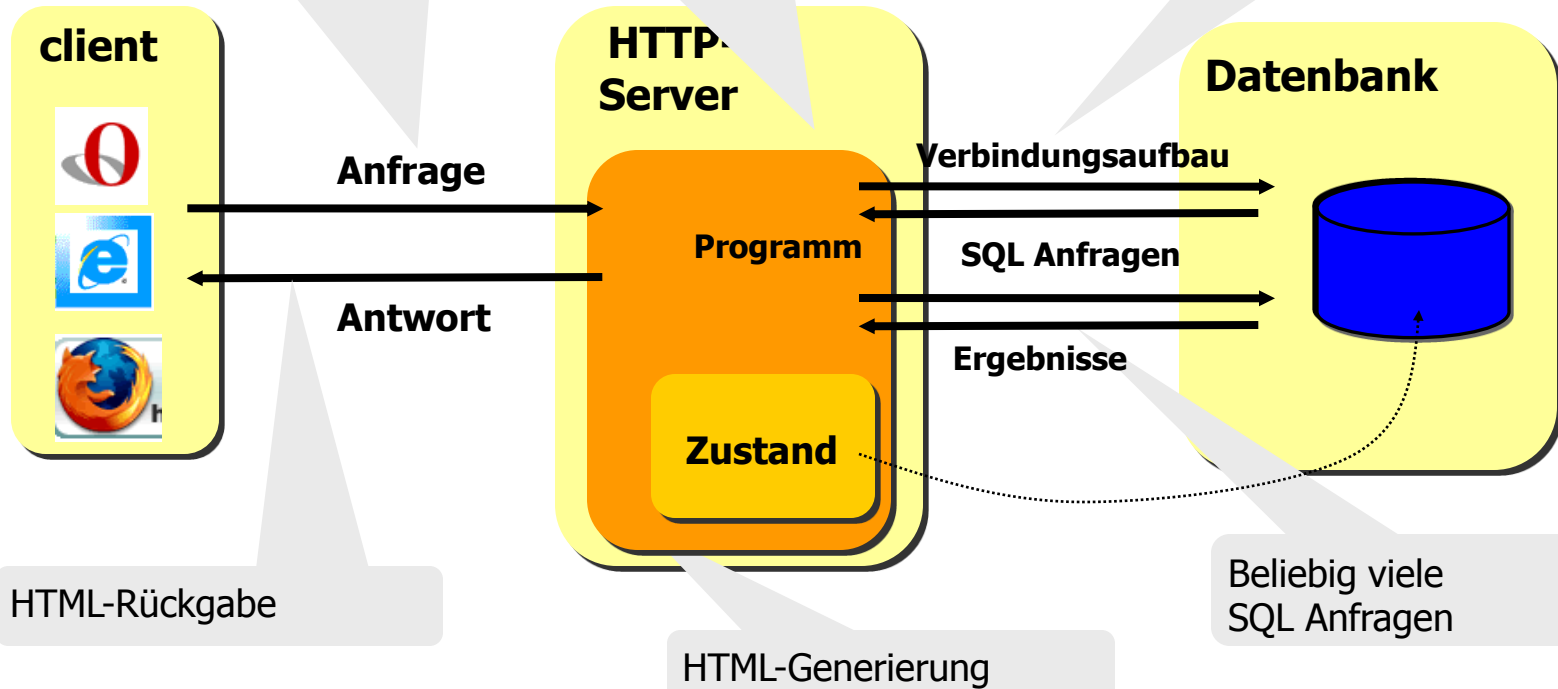
- Gateway an Web-Server als Erweiterung zur Integration von Skriptsprachen oder Programmen, z.B. Python, Perl
- + Einzelne Aufrufe konkurrieren sich nicht, da jeder Aufruf in eigenem Prozess
 - Ein Prozess pro Anfrage (Ressourcenintensiv)
 - Keine Speicherung des Zustands (CPU intensiv)
 - Für jede DB-Anfrage Verbindung aufbauen und trennen (zeitintensiv)
 - Keine Trennung von Präsentation und Anwendungslogik (Wartbarkeit für größere Projekte kaum möglich)

Optimierung durch API-basierte Ansätze

Betätigung Submit-Button
<FORM METHOD="POST"
ACTION="http://www.fzi.de/
cgi-bin/my-form">

Aufruf des Programms,
Parameterübergabe
**Initialisierung bei ersten
Aufruf (einmalig)**

Verbindungsaufbau zur
DB beim ersten Aufruf
(einmalig)

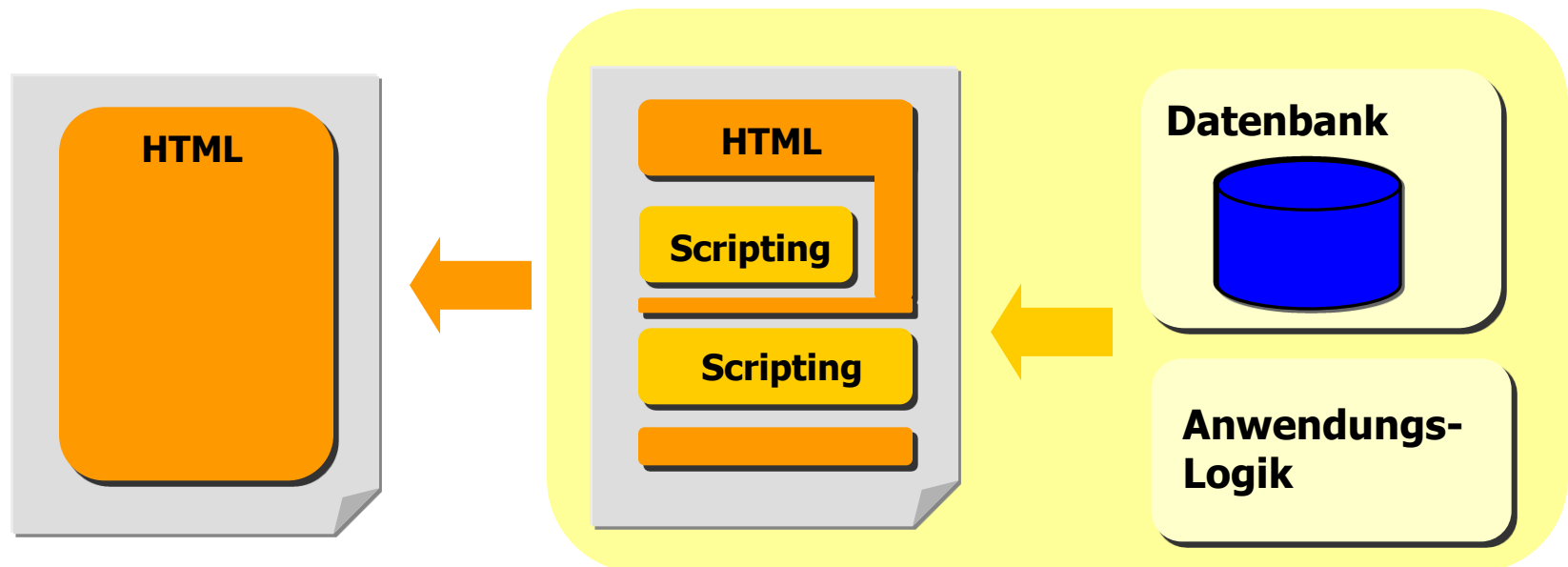


Beispiel: Java Servlet

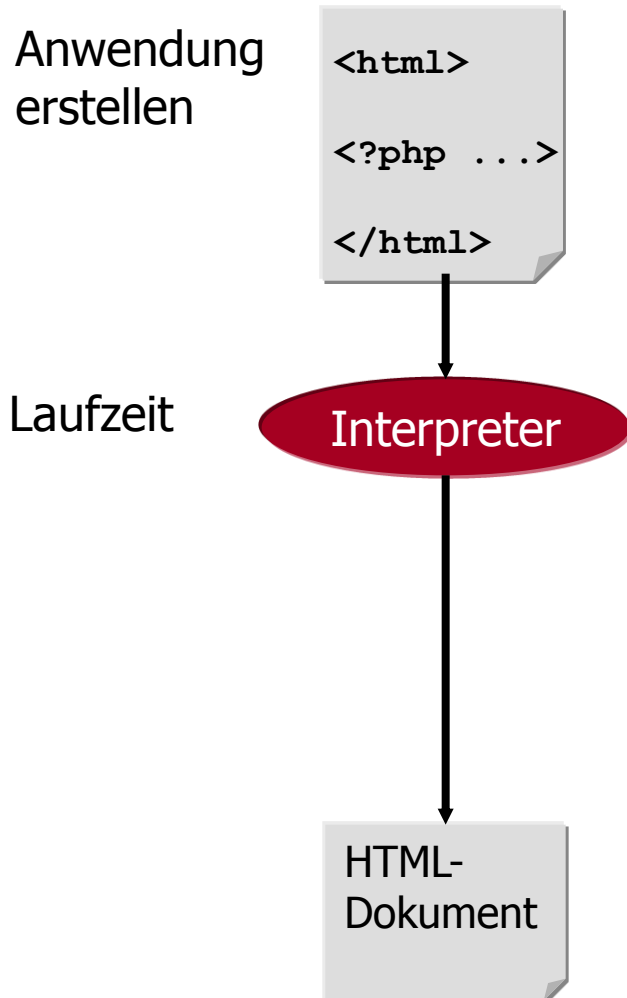
- **Programm wird in den Adressraum des Servers geladen**
 - + Programm startet einmalig, nur eine Datenbankverbindungen erforderlich
 - + Zustandshaltung möglich, z.B. durch Session-IDs
 - + Anfragen werden in Threads statt Prozessen ausgeführt (deutlich Ressourcenschonender)
 - Installation von speziellem Web-Server zur Ausführung von Java-Servlets nötig, z.B. Apache Tomcat (für kleine Projekte ungeeignet)
 - Keine Trennung von Präsentation und Anwendungslogik
- ⇒ Für kleine Anwendungen zu aufwendig, für große ungeeignet

Server-Side Scripting

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



Variante 1: Interpretation zur Laufzeit



Interpretierte Skriptsprachen: z.B. PHP

- Skript wird direkt in eine HTML-Seite geschrieben
- Bei Anfrage einer Seite wird Skript durch Interpreter interpretiert und HTML-Text geschrieben
- Generiertes Dokument wird an Server zurückgegeben

- + einfache Art, Skripte einzubinden
- + Interpreter ist nur ein einzelner Prozess
- CPU-Last bei jedem Seitenabruf durch Neuinterpretation
- Keine Trennung von Präsentation und Logik

⇒ Für kleinere Webanwendungen sehr gut geeignet

PHP für große Projekte: Wikipedia

□ **Praxisbeispiel**

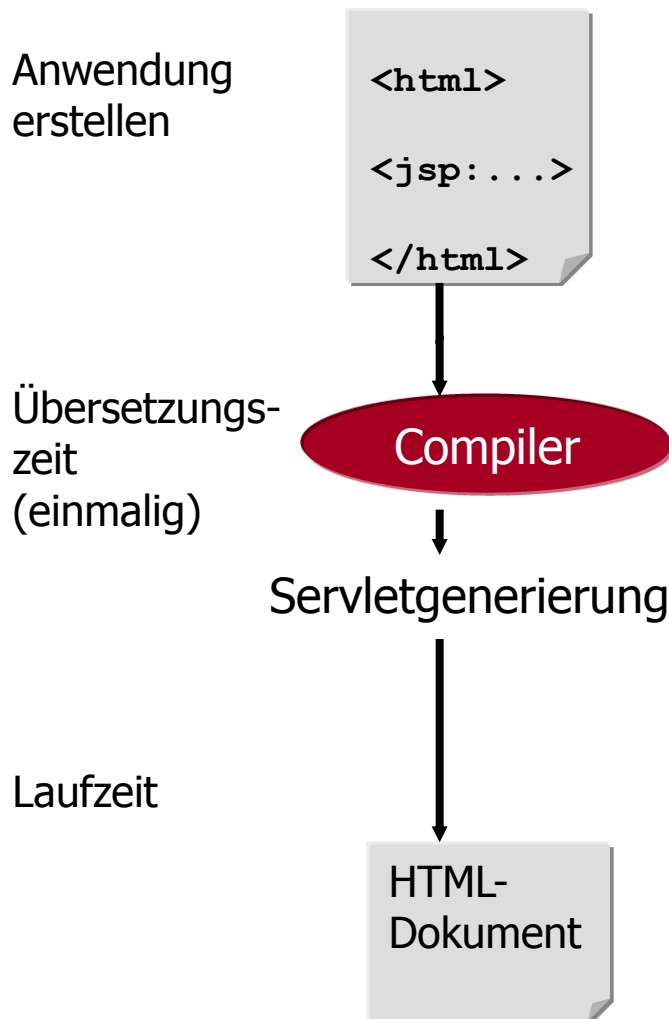
Wikipedia.org verwendet PHP für ein sehr umfangreiches Projekt.

□ **Prinzip:** Statisches Schreiben von Webseiten

- Verhältnis von Seitenabruf zu Seitenänderung sehr gering
- Anwendungslogik vergleichsweise gering: Seitenänderungen annehmen und protokollieren
- ⇒ Statisches Schreiben von Webseiten reduziert CPU-/Ressourcenlast: HTML-Seite muss nur geladen werden

! Wenn möglich, sollten aus dynamischen Seiten statische Seiten generiert werden, um die Skalierbarkeit zu erhöhen

Variante 2: Kompilierte Skriptsprachen



Kompilierte Skriptsprachen z.B. JSP

- Erfordert Installation von JAVA-fähigem Web-Server
- Kompilierung erzeugt Servlets zur Ausführung der JSP-Seiten
- + JSP 1.1 ermöglicht durch „Tags“ klarere Trennung von Darstellung und Anwendung
- JSP-Programmierung entspricht nicht den meisten Programmierern bekannte GUI-Programmierung (z.B. event-Verarbeitung)

Clientseitige Anwendungslogik

□ Motivation

- Die Funktionen im Browser reichen für geplante Anwendung nicht aus, z.B.
 - gleichzeitiges Aufladen von Bildern aus einem lokalen Ordner für ein Fotoalbum (Sicherheitsbeschränkungen von http erlauben immer nur eine Datei auszuwählen □ Umständlich für Benutzer)
 - Abspielen von Multimedia (z.B. Videos)
- Die Anwendung soll sehr dynamisch sein, z.B.
 - Produktwerbung (viele bewegte Bildsequenzen)
- Die Anwendung soll wie eine Desktop-Anwendungen wirken
 - Vor ein paar Jahren gab es noch kein AJAX ;-)

Browser Plugins

Erweiterung des Browsers um neue Softwarekomponente

- am Beispiel von Adobe Flash
 - Einsatz: sehr dynamische Anwendungen, Multimedia, Desktop
 - Integrierte Skriptsprache: Action Script
- Teile oder komplette Anwendungslogik wird integriert
 - Kommunikation mit Server über http oder soap

Starten einer Plugin-Anwendung

1. Benutzer muss jeweils kompatible Version des Plugins haben
 2. Eigener HTML-Tag (<object><embed>) zum Starten des Plugins in einer Webseite
 3. Komplettes Programm wird zum Client übertragen
 4. Plugin lädt Programm und stellt es dar
- + Verhalten der Software exakt vorhersagbar (mit HTML problematisch)
 - Plugin, speziell aktuellste Version nicht auf allen Client PCs (z.B. beschränkte Installationsrechte auf Firmen-PSs)
 - Erfordert Spezialisten im Umgang mit der jeweiligen Programmiersprache
 - Meist Wartung schwierig

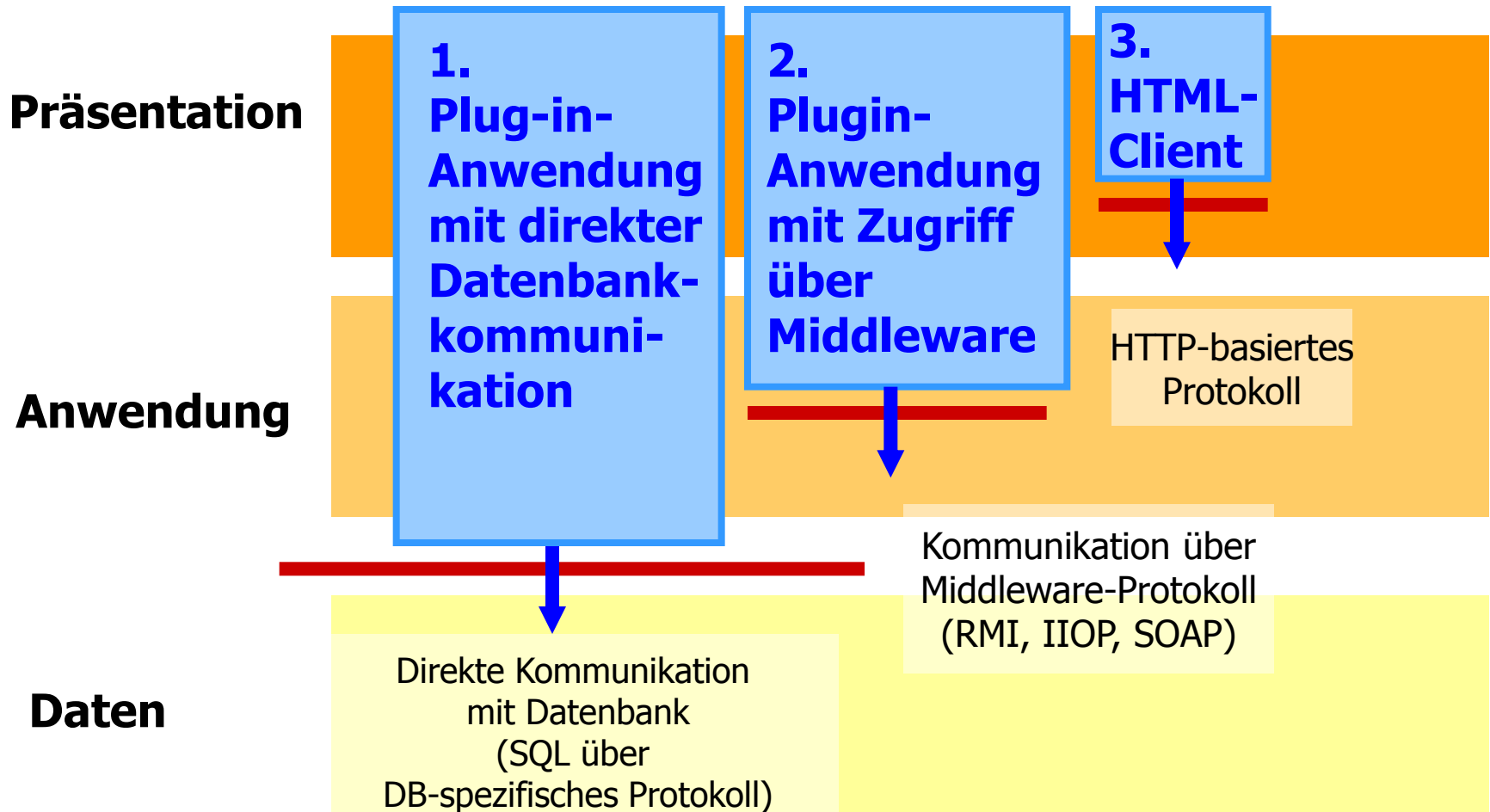
```
<object width="400" height="400"
classid="clsid:d27cdeb6-ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.adobe.com/pub/shockwave/cabs/flash
/swflash.cab#version=9,0,0,0">
<param name="src" value="example.swf">
<embed src="example.swf" type="application/x-shockwave-flash"
width="400" height="400",
pluginspage="http://www.adobe.com/go/getflashplayer">
</embed>
</object>
```

□ Motivation

- Über ein Browser soll eine Desktopanwendung geladen werden
- Anwendungslogik komplett integrierbar
- Theoretisch auch Datenhaltung direkt ansprechbar (praktisch nicht: aus Sicherheitsgründen – erfordert dies Öffnung eines Ports direkt zur Datenbank !)
- + Entwicklung der Anwendung komplett in JAVA möglich
- + Saubere Trennung von Darstellung und Anwendungslogik möglich
- + Wiederverwendbarkeit von Komponenten
- vgl. Plugins: Installationen notwendig

```
<object classid="java:example.class" codetype="application/java-vm" width=„400" height="400"></object>
```

Schichtenarchitektur mit Skripten und Plugins



Zusammenfassung: kleine Webanwendungen

□ Für kleine Anwendungen

- Skalierbarkeit, Wartbarkeit und Wiederverwendbarkeit stellen keine Anforderung dar
- Plugins ermöglichen Spezialanwendungen
- Alle gezeigten Skriptansätze ermöglichen die Generierung von dynamischen HTML-Seiten.
- Erstellung von Anwendungen mit Desktop-Charakter hier nicht nötig
- Generierung von statischen Seiten ermöglicht auch umfangreiche Anwendungen bei geringen Ressourcen

Umsetzung der Anwendung also entscheidend von Vorkenntnissen der Programmierer und den installierten Interpretern / Gateways auf einem Web-Space

Web-Rahmenwerke für umfangreiche Webseitenprojekte

.. „Mal schnell ein Skript schreiben“ reicht da nicht mehr!

Umfangreiche Webanwendungen

- **Beispiele:**
 - Ebay, Google (Mail), Flickr, ...

- **Eigenschaften**
 - Hoher Anteil an dynamischen Seiten
 - Komplexer Seitenaufbau
 - Umfangreiche Anwendungslogik, häufige Änderungen und Erweiterungen
 - Verschiedene Ausgabesprachen, Ausgabemedien
 - Zuverlässiger Betrieb auf unterschiedlichsten Browsern

- **Erstellung mit Web-Rahmenwerk**
 - Rahmenwerk, das Erstellung einer umfangreichen Anwendung ermöglicht

Rahmenwerke über

Tapestry Echo **JSF** Cocoon Millstone

OXF **Struts** Cassandra WebWork SOFIA

WebOnSwing Xoplon Groovy Flex

Genie **Wicket** Chiba Jaffa JBanana **GWT**

Smile MyFaces JWarp Melati

Eclipse RAP Japple Helma Dinamica

Liste unvollständig.

Welches Rahmenwerk verwenden?

- **Hauptfrage bei einem neuen Projekt**
 - Welches Framework verwendet man?

- **Generelle Faktoren**
 - Programmiersprache: abhängig vom Team
 - Lizenz: abhängig von Budget, Infrastruktur
(Beispiel Smartshopping: alles .NET □ Rahmenwerk ebenfalls)

Hauptparameter bei umfangreichen Projekten

Trennung von Anwendungslogik und Darstellung

Gründe

- Code gemischt mit HTML schwer lesbar und wartbar (in Skriptsprachen und JSP der Fall)
- Änderung der Darstellung können unabhängig von der Anwendungslogik gemacht werden.

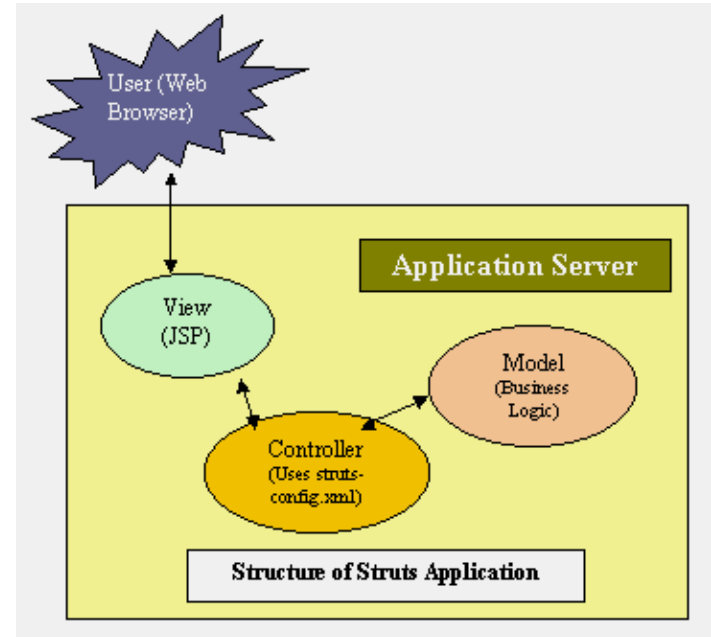
Gängige Umsetzungen

- Id-Marken in HTML-Text, die später durch HTML-Text ersetzt werden – z.B. ``
- Web-Rahmenwerk bietet Standardkomponenten an, die verwendet werden können (Tables, Pulldowns, etc.)
- Formatierung durch CSS
- Wünschenswert: eigene Komponenten erstellbar

Model View Controller

- **Dreischichtenarchitektur** trennt
 - Präsentation
 - Anwendung
 - Datenhaltung

- **Model View Controller** ähnlich, trennt
 - View: Darstellung
 - Controller: Request / Response Verwaltung
 - Model: Daten



- ⇒ Saubere Trennung von Darstellung und Anwendung
- ⇒ Wiederverwendbarkeit der Anwendungslogik
- ⇒ Im Einsatz bei SWING und den meisten Web-Frameworks

Prinzip

- meist: Model - View - Controller

Es gibt Fülle von Rahmenwerken ...

Welches nimmt man?

Ziele

- Zusammenstellung von Anforderungen an ein Rahmenwerk
- Zusätzliche Anforderungen durch „AJAX-Trend“
- Kurze Vorstellung von drei Rahmenwerken heute
- Tutorial für spezifisches Framework am 7.11.

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (1)

- **Regelmäßige Umstrukturierung**, z.B. Umbau / Erweiterung der Seite:
 - *Erfordert strikte Trennung von Anwendungslogik und Darstellung*
- Unterstützung von **einheitlichem Look & Feel**
 - *Templates, auch für verschiedene Ausgabearten (PC, PDA, Mobile, ...)*
- Betrieb der Webseite in **mehreren Sprachen**
 - *Automatisches Füllen von Textstellen in Webseiten mit Landersprache*

- Webseiten umfangreicher Projekte müssen meist in mehreren Sprachen erstellt werden
- Manuelle Erstellung von sprachspezifischen Seitenteilen ist zeit- und pflegeaufwendig, macht schwer wartbare Quelltexte

Anforderung

- Framework sollte sprachspezifische Textbausteine automatisch in Webseiten einfügen können

Gängige Umsetzung: i18n Standard

- Verwendung von Properties-Dateien für länderspezifische Sprachbausteine, z.B. (response_en.properties)
- Dynamisches Laden der jeweiligen Properties-Datei durch das Web-Framework für die Spracheinstellungen eines Nutzers

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (2)

- **Hoher Anteil an dynamischen Seiten**, z.B. für Suchergebnisse, Shopsysteme, Warenkörbe, ...
 - *Erstellung von umfangreicher Anwendungslogik*
- **Umfangreiche Datenintegration** / Umfangreiche Dateneingaben
 - *Erfordert Einbindung von Frameworks zur Datenintegration*
 - *Integrierte Prüfung der Konsistenz von Benutzereingaben mit Datenmodell und Anwendungslogik*

Datenintegration und Eingaben

□ **Umfangreiche Datenintegration**

- Erweiterung mit speziellen Frameworks mit Objektmapping
Hybernate, Spring (Details: Vorlesung Guido Sauter)

□ **Umfangreiche Dateneingaben**

- wenn nötig Transaktionen, Speicherung, ...
- Durch Web-Rahmenwerk selbst oder in Verbindung mit Erweiterungen: Prüfung der Konsistenz von Benutzereingaben,
z.B. PLZ hat fünf Ziffern
Emailadresse muss „@“ beinhalten

Anforderungen an umfangreiche Webanwendungen und Web-Rahmenwerke (3)

- **Zugangsbeschränkungen** für geschützte Bereiche
 - *User-Sessions und Weiterleitung auf Login-Seite*
- **Schnelle Reaktionszeiten** auf Benutzeranfragen
 - *Zustandshaltung von aktuell benötigten Daten / Kontext eines Nutzers*
- Anwendungen mit **Desktopstil**
 - *Einfache Integrierbarkeit von AJAX-Techniken*

Zugangsbeschränkungen für geschützte Bereiche

□ Gängige Verfahren (da http zustandslos)

- Rahmenwerk generiert nach einem erfolgreichen Login ein „Session-ID String“.

z.B. *jsessionid=0EEC1A1BE9980145F81DF6B99418B027*

- Rahmenwerk hält ein Session-Objekt pro Benutzer zur Bindung von Objekten eines Benutzers

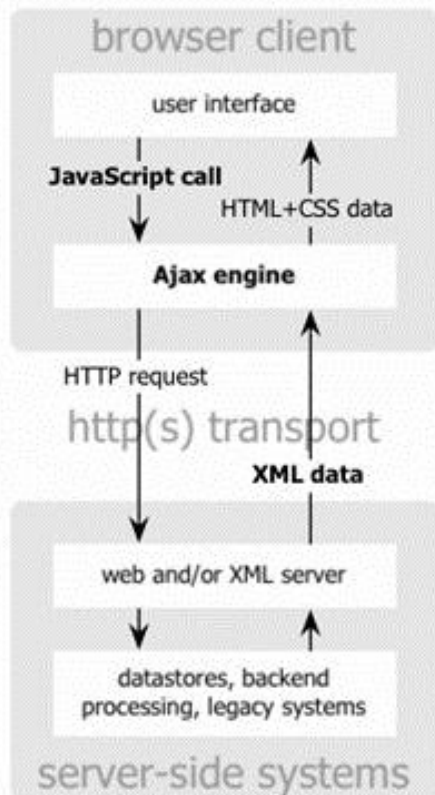
- Nutzeridentifikation:

Session-id als cookie (falls von Browser akzeptiert), sonst als GET Parameter in allen URLs der Webseite

Bei jedem Seitenaufruf:

- Rahmenwerk sucht nach vorhandenem Session-Objekt
- Falls nicht vorhanden: Weiterleitung auf Login-Seite (falls die aufgerufene Seite eine geschützte Seite ist)

Anwendungen mit Desktopstil – AJAX



- **AJAX (asynchronous JavaScript and XML)**
 - Ermöglicht die Aktualisierung von Seitenteilen

Somit muss nicht immer die ganze Seite neu geladen werden und die Anwendung bekommt desktop-feeling
 - Bestandteile:
 - Über JavaScript wird ein „XMLHttpRequest“ an Server geschickt
 - Server verarbeitet Request und gibt XML oder HTML zurück
 - JavaScript nimmt Request entgegen und verarbeitet diesen, meist durch Aktualisierung eines Seitenbereichs (z.B. Layers)
- ⇒ Keine wirklich neue Technologie, sondern eine Kombination von bestehenden Techniken

□ Beispiele

- Auto-Complete für Suchanfragen
- Pop-Up Fenster (in Seite integriert)
- Drag & Drop
- Direkte Fehlermeldungen auf Falscheingaben in Formularen

□ Best Practices

- AJAX-Funktionen manuell zu erstellen ist zeitaufwendig und fehleranfällig (Nur für JavaScript Experten)
 - **Verwendung eines geeigneten Rahmenwerks**
- Bei Aktualisierung von Seitenteilen: Aktion dem Benutzer anzeigen (z.B. durch Ladebalken)
- AJAX nicht wahllos einsetzen!

AJAX integriert in Web-Frameworks

- Ajax stellt umfangreiche Anforderungen an ein Web-Framework
 - Ohne AJAX: es wird eine ganze Seite generiert und Komponenten, Zustände der Seite komplett geändert
 - Mit AJAX: Webtechnologie sollte möglichst schnell und unkompliziert die Aktualisierung von Seitenteilen unterstützen
 - Komplexität der JavaScript Aufrufe verbergen
 - Methoden für AJAX-Calls anbieten
 - Aktualisierung von Seitenteilen / Komponenten ermöglichen
 - AJAX Unterstützung möglichst direkt in einem Framework (und nicht über viel „Hackerei“ eingebunden)

⇒ **Für zeitgemäße Webanwendungen:**
AJAX Support in Framework zwingend erforderlich

Weitere Anforderungen

- **Objektorientierte Programmierung** der Anwendung mit für Entwickler vertrauten Prinzipien (vgl. Swing: Datenbindung, Events, etc)
- **Schnelle Erlernbarkeit** der Webtechnologie, Abstraktion von zugrundeliegenden Techniken (z.B. Details von http)
- Durchführung von **automatischen Funktionstests** (vgl. JUnit Tests)
Grund: manuelle Tests zeitaufwendig!

Auswahl des geeigneten Rahmenwerks

□ Zusammenfassung Hauptkriterien

- Erfüllung der genannten Anforderungen, Fokus auf
AJAX-Support
Erweiterbarkeit

Trennung von Anwendung und Darstellung

- Lizenzierung: Proprietär vs. open-source
- Programmiersprache (im Folgenden: Focus auf JAVA)
- Zeitaufwand zur Erstellung von Web-Anwendungen
- Support: Qualität der Mailinglisten-Antwortzeit

Konzepte von Web-Rahmenwerken

- Google Web Toolkit (GWT)
- Java Server Faces (JSF)
- Weitere Web-Rahmenwerke und Technologien

Google Web Toolkit (1)

- Programmiersprache: JAVA
- Widget Library: Verhalten von GUI Komponenten wie in Swing programmierbar (Events, Container-Zuordnung, ...)
- Verhalten der Komponenten wie in Applets (mittels AJAX), d.h. möglichst alles im Client
- Proprietärer Compiler generiert daraus sämtlichen XHTML und JavaScript Code für AJAX
- Saubere Trennung von Darstellung und Anwendungslogik

Google Web Toolkit (2)

- Entwickler benötigen (fast) keine HTML-Kenntnisse (nur eine Host-HTML Seite wird benötigt)
- Design der Komponenten über CSS anpassbar. (Aufruf von CSS über Host-HTML Seite)
- Spring, Hybernate für Datenhaltung integrierbar
- Für Anwendungen in der Cloud mit Google App Engine
- GWT auch nur zur GUI-Erstellung (Webseiten-Frontend) nutzbar
 - JSF dann für die Anwendungslogik

Google Web Toolkit Bewertung



- + Trennung von Anwendung und Darstellung
- + Einfach erlernbar und schnell anwendbar
- + Wachsende Community – guter Support

- Getrennte Betrachtung von Client und Serverseite
(Bereitstellung von Daten durch SOAP basierte RPC Calls)
- Proprietärer Compiler

JSF: Java Server Faces

- **Grundidee:** Komponentenverhalten wie in SWING
 - Java Beans als Model; JSP Seiten und Komponentenbaum als View; Action Listener und aus JSP generiertes Servlet als Controller

Jedes Request durchläuft festen JSF-Lifecycle

- 1. Restore view** : stellt den Zustand der aktuellen Session her (aus Session Context)
- 2. Apply request values**; process events: Zustandsermittlung der Komponenten
- 3. Process validations**; process events : Eingabewerte validieren
- 4. Update model values**; process events : Beans aktualisieren
- 5. Invoke application**; process events : Anwendungslogik ausführen
- 6. Render response** : Antwortseite generieren

JSF: Java Server Faces

□ **Ajax Support**

- für JSF 1.0 Ajax4JSF mit SEAM, G4JSF (GWT mit JSF kombiniert)

- in JSF 2.0 integriert:

Unterteilung des Lifecycles und Komponenten in „execute“ (auf dem Server) und „render“ (Darstellung im Client bei Antwort)

z.B. `<f:ajax event="change" execute="@this" render="map"/>`

□ **JSF Facelets**

- Verzicht auf JSP Seiten; ermöglicht strikte Trennung von Anwendung
- Stattdessen HTML-Seiten mit id-Marken, in die Komponenten später geschrieben werden sollen
- in JSF 2.0 als bevorzugt integriert

JSF: Bewertung



- + Durch **Industriestandard** Zukunftssicherheit für Anwendungen gegeben,
- schwergewichtiger Lifecycle
- JSP-Seiten können Anwendungslogik enthalten
- Potpourri an Technologien und Frameworks
- recht hoher Zeitaufwand, z.B: durch Konfiguration in XML-Files nötig

Weitere Web-Rahmenwerke und Technologien

□ Apache Wicket

- Offizielles Apache Projekt, open source
- Performante Event-getriebene Architektur
- Strikte Trennung von Anwendungsentwicklung mit JAVA Objekten und Seitendesign mit HTML
 - Integration von Komponenten mit HTML Tags (`<wicket:id="".."" .. >`)
- Umfangreicher AJAX-Support
 - Allerdings AJAX-Calls seriell
- Spring, Hibernate für Datenhaltung integrierbar
- Keine Widget Library; GUI-Komponenten müssen selbst erstellt werden
- Hohe Serverlast – jeder Klick geht zum Server

Weitere Web-Rahmenwerke und Technologien



- **Grails** (ursprünglich **G**roovy on **R**ails)
 - Verwendet Sprache Groovy – auf JAVA basierende Variante von Ruby on Rails
 - Verwendet Hibernate für objektrelationales Mapping
 - Scaffolding: Basierend auf einem Bean (Domain) werden Views und Controller automatisch generiert
- **Eclipse Rich Application Client** (RAP)
 - Entwicklung von „richtigen“ Desktop Anwendungen mit Eclipse
- **Alles nur mit XML / XSLT / ...**: Vorlesung 28.11.

- Umfangreiche Webanwendungen erfordern Web-Rahmenwerke
 - Ermöglichen langfristige Wartung einer Anwendung
 - Trennung von Darstellung und Anwendung wichtig
 - Verdecken Komplexität der zugrunde liegenden Webtechnologien
 - Umfangreiche Anforderungen an ein Web-Rahmenwerk
 - Integration von AJAX in ein Rahmenwerk ermöglicht die Erstellung von Anwendungen mit Desktop-Charakter
- Auswahl des „richtigen“ Rahmenwerks
 - Bei neuen Projekten: Prüfen, ob aktuelle Rahmenwerke die Anforderungen besser/direkter/schneller erfüllen können

- **Buch: Web-Technologien**
 - von Heiko Wöhr von Dpunkt Verlag (Taschenbuch - Mai 2004)
- **JSP**
 - <http://www.jsptutorial.org/>
- **AJAX**
 - <http://www.w3schools.com/ajax/default.asp>
- **Google Web Toolkit**
 - <http://code.google.com/webtoolkit/>
- **JSF**
 - <http://www.exadel.com/tutorial/jsf/jsftutorial-kickstart.html>
- **Wicket**
 - <http://wicket.apache.org/>
- **Grails**
 - <http://www.slideshare.net/loffenauer/2007-09-10-fzi-training-groovy-grails-v-ws>,
Raphael Volz
- **Eclipse RAP**
 - <http://www.eclipse.org/rap/>