Internetanwendungstechnik

World Wide Web (WWW)

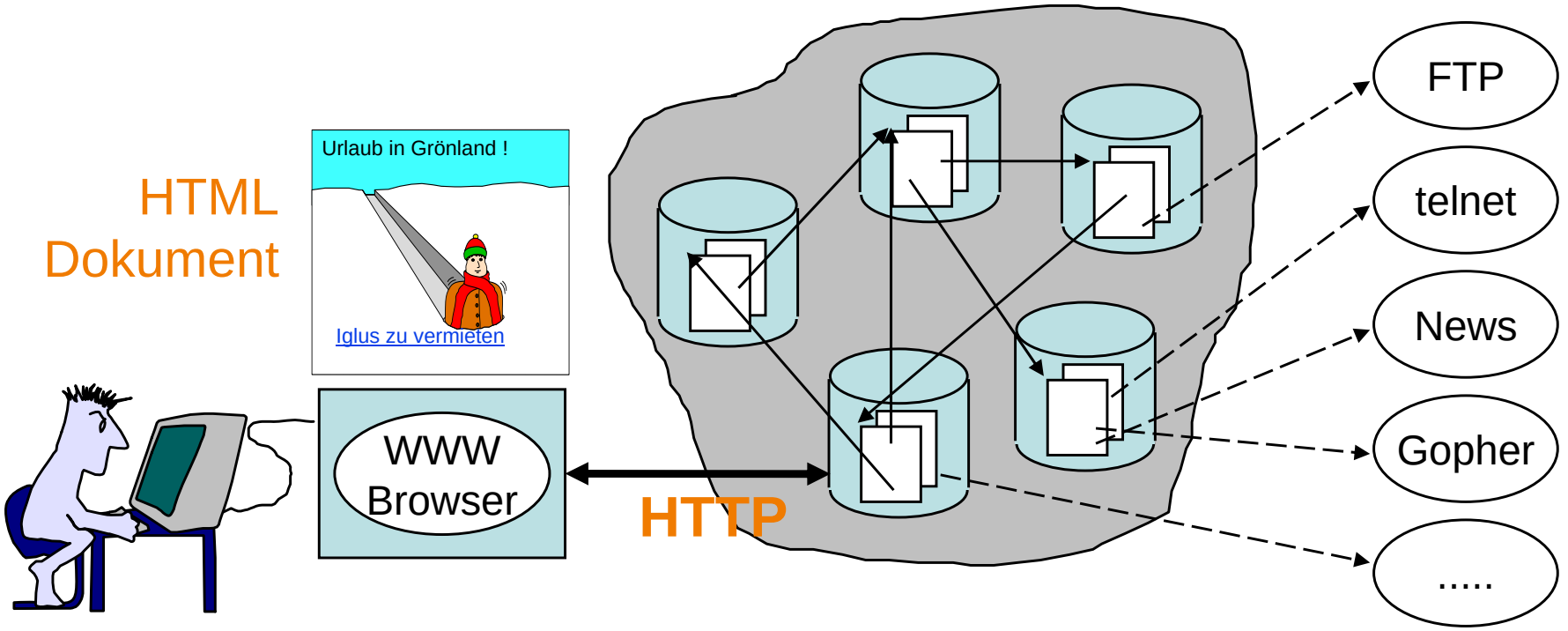
Gero Mühl

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Kommunikations- und Betriebssysteme (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

Word Wide Web (WWW)

- Das WWW ist ein verteiltes Hypermedia-System
  - Dezentrale Speicherung von Dokumenten
  - Dokumente können aus Text, Grafik, Audio, Video, etc. bestehen
  - Hyperlinks zwischen den Dokumenten
  - Einfache Navigation von Dokument zu Dokument
- Standards für die Beschreibung der Dokumentenstruktur und für das Protokoll des Dokumentenzugriffs
- Einheitliche Schnittstelle für beliebige Dokumentarten
- Integration anderer Internet-Anwendungen in *Browser*

Architektur



Architektur

- Wie werden Dokumentinhalte dem Benutzer zugänglich?
  → Web Browser und Einbindung externer Viewer
- Wie werden Dokumente adressiert?
  → Uniform Resource Locators (URL)
- Wie werden Dokumente übertragen?
  → Hypertext Transfer Protocol (HTTP)
- Wie werden Inhalte beschrieben?
  → Hypertext Markup Language (HTML)
- Wie werden Anwendungen in Verbindung mit WWW erstellt?
  → z.B. mit CGI, Java Servlets, Active Server Pages, etc.

World Wide Web (WWW)

Browser, URLs, HTTP und Web-Server

WWW Clients

Grafische Browser

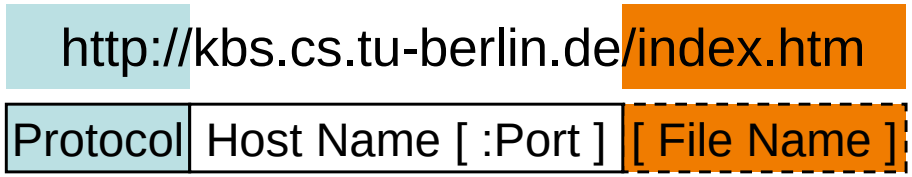
- Direkt darstellbar sind
  - HTML-, ASCII-Dokumente
  - Verbreitete Grafikformate wie GIF, XPM, JPEG, PNG, etc
- Einbindung externer Betrachter
  - Postscript, PDF
  - spezielle Grafikformate wie TIFF, PICT, etc
  - Audio (WAV, MP3, etc)
  - Video (MPEG, QuickTime, etc.)
- Beispiele
  - Firefox, Mozilla, Netscape
  - Konqueror
  - Internet Explorer

Zeilenorientierte Browser

- Reine Textdarstellung ohne Bilder, Audio, Video, etc.
- Bedienung nur über Tastatur
- Geeignet für Verbindungen mit geringer Übertragungsrate
- Beispiele
  - Lynx
  - Emacs, Xemacs

Universal Resource Locator (URL)

- Fragen beim Zugriff auf WWW Seiten
  - Wie kann auf die Seite zugegriffen werden?
  - Wo liegt die Seite?
  - Wie lautet der lokale Name der Seite beim Server?
- URL besteht aus Angaben zu Protokoll, Host und Datei
  - Protokolle können sein http, ftp, file, news, telnet, mailto, etc.
- Beispiel



URL Beispiele

- http://www.kbs.cs.tu-berlin.de/
- http://www.mit.edu:8001/people/dshapiro/macsites.html
- file://WWW/beispiel.html
- ftp://ftp.cs.mit.edu/README
- telnet://www@lynx.cc.ukans.edu/
- mailto:kbs@cs.tu-berlin.de
- news:comp.infosystems.www.users
- news:AA0134223112@cs.utah.edu
- gopher://gopher.tc.umn.edu/11/Libraries

Uniform Resource Name (URN)

- URNs
  - sind dauerhafte, ortsunabhängige Namen
  - bezeichnen WWW-Ressourcen
- Beispiele
  - urn:ietf:rfc:2141 // RFC 2141
  - urn:ietf:std:50 // ...
  - urn:ietf:id:ietf-urn-ietf-06 // ...
- Allgemein
  - URLs und URNs sind beides Uniform Resource Identifiers (URIs)

Hypertext Transfer Protocol (HTTP)

- Dient meist dem Transfer von Dateien vom Server zum Client
- Ist trotz des Namens nicht auf HTML-Dokumente beschränkt
- Übertragung stets vom Client initiiert
- Einfaches ASCII-Protokoll, benutzt Elemente von MIME
- Meistens aufgesetzt auf TCP (Standard lässt auch anderes zu)
- Kommandos (vom Client zum Server)
  - GET — Lesen einer Seite
  - HEAD — Lesen eines Seitenkopfs
  - PUT — Abspeichern einer Seite
  - POST — Anfügen an eine Seite
  - DELETE — Löschen einer Seite

HTTP Beispiel

Client

```
GET /WWW.../index.html
HTTP/1.0
```

Server

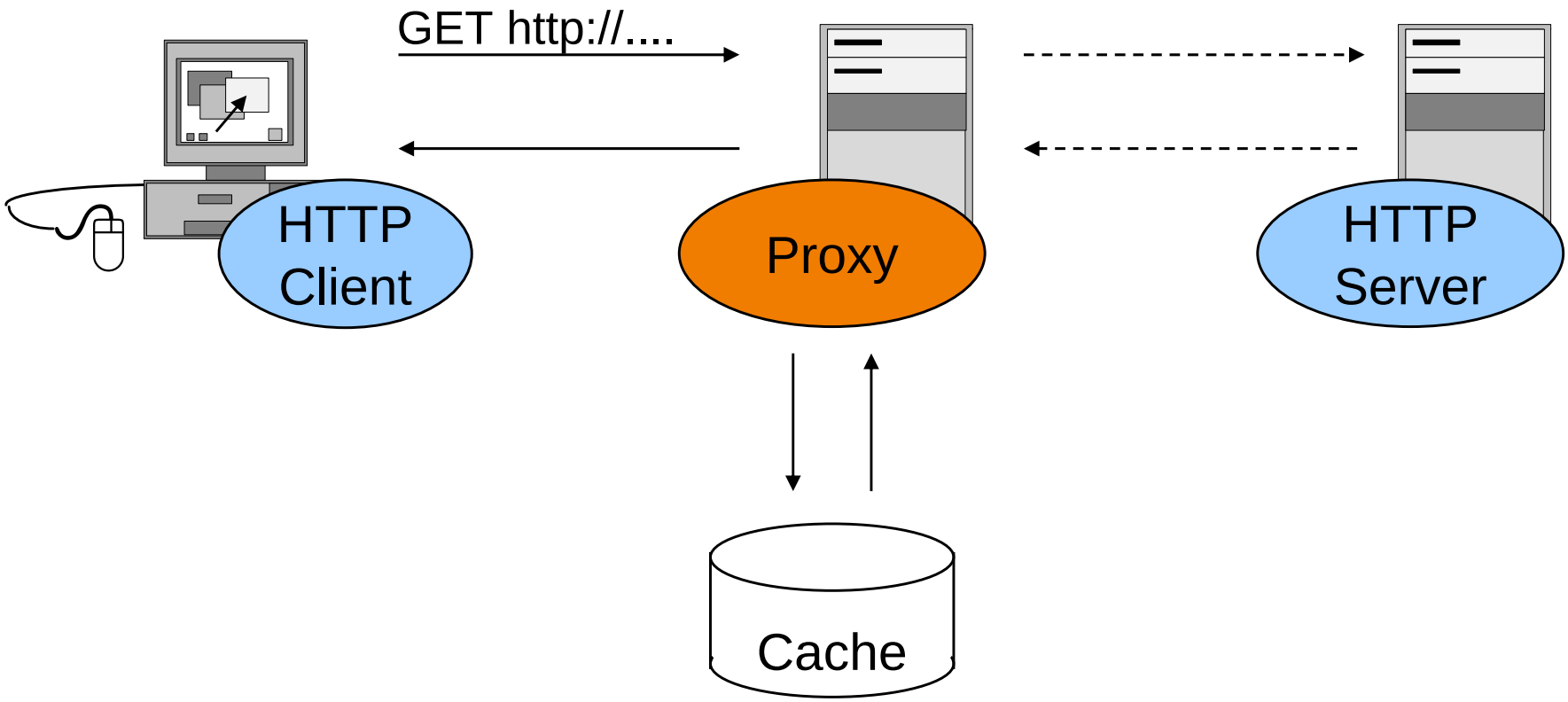
```
HTTP/1.0 200
MIME-Version: 1.0
Server: CERN/3.0
Content-Type: text/html
Content-Length: 8247

<HTML>
 <HEAD><TITLE>Foo</TITLE></HEAD>
 <BODY>
  <H1>FooBar</H1>
  text text text
  <A HREF="http://mit.edu/">MIT</A>
  text text text
 </BODY>
</HTML>
```

WWW Server

- WWW Server (meist) über Port 80 erreichbar
- Normalerweise eine TCP-Verbindung pro HTTP-Auftrag
- Für die Übertragung von Bildern, Audio, Video, etc. werden zusätzliche TCP-Verbindungen aufgebaut
- Typische Fehlermeldungen
  - 401 unauthorized // keine Zugangsberechtigung
  - 404 not found // Dokument nicht gefunden
  - 500 server error // Fehler mit Fehlermeldung

WWW Proxy Server



- Beispiel
  - www.cs.tu-berlin.de:2784

Web-Server Log-Dateien

- Server sammeln statistische Zugriffsinformationen
- Erstellung von Nutzungsprofilen möglich → Datenschutz
- Access Log
  - Pro Anfrage eine Zeile
  - Statistische Auswertung durch entsprechende Tools
  - Beispiel (Common Access Log Format)
    - ```
      wc.yale.edu -- [16/Jun/1997:04:29:55-0100]
      GET /index.html/    HTTP1.0 200 733
      ```
- Error Log
  - pro Fehler eine Zeile
  - Beispiel (Error Log)
    - ```
      [16/Jun/1997:04:55:29-0100] [NOT AUTHORIZED]
      [host: cs.yale.edu referer: http://hera.gmd.de/]
      /index.html
      ```
- Andere Error Log-Formate sind möglich

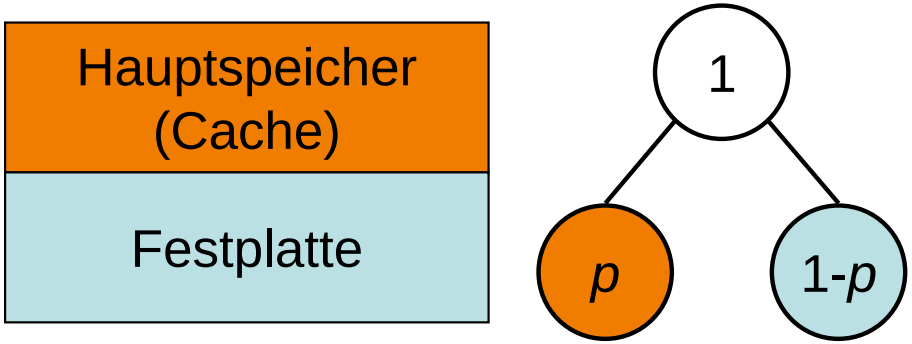
Web-Server Performanz

- Web Server werden häufig aus Performancegründen multithreaded implementiert
- Während einer IO-Operation (z.B. Plattenzugriff) kann dann ein anderer Request bearbeitet werden
- Ziel ist volle Auslastung der CPU

Wie viele Threads werden hierfür benötigt?

Einstufiger Cache

- Parameter
  - Cache Hit Rate *p*
  - Rechenzeit t_1
  - Wartezeit t_2 bei Cache Miss



- Annahmen
  - Zufällige Anfragen/Zugriffe der Clients
  - Zeit für Entscheidung Cache Hit/Miss vernachlässigbar
- Durchschnittliche Bearbeitungszeit *t* eines Requests

$$t = p * t_1 + (1 - p) * (t_1 + t_2) = t_1 + (1 - p) * t_2$$

- Auslastung *U* (Rechenzeit / durchschnittliche Bearbeitungszeit)

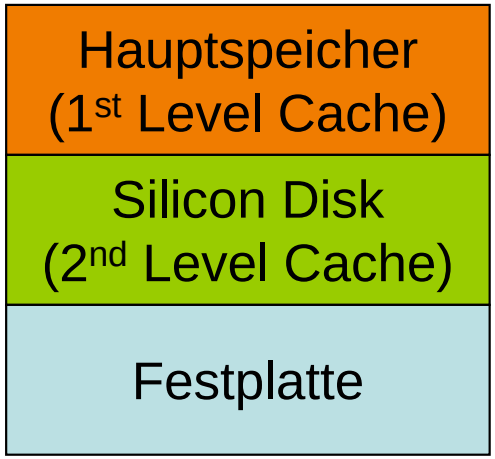
$$U = t_1 / t$$

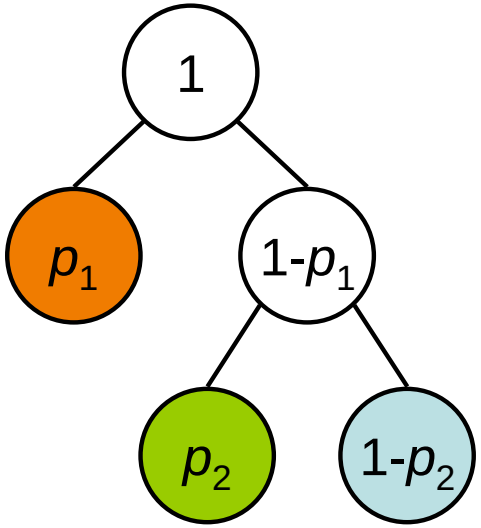
- Anzahl *n* der notwendigen Threads

$$n = 1 / U$$

Zweistufiger Cache

- Parameter
  - Cache Hit Rate p_1 und p_2 mit $p_2 > p_1$
  - Rechenzeit t_1
  - Wartezeiten t_2 bzw. t_3 bei Cache Miss





- Seite nicht im First Level Cache, aber im Second Level Cache
  → Wartezeit t_2
- Seite auch nicht im Second Level Cache
  → Wartezeit t_3
- Durchschnittliche Berabeitungszeit t

$$t = p_1 * t_1 + (1 - p_1) * p_2 * (t_1 + t_2) + (1 - p_1) * (1 - p_2) * (t_1 + t_3)$$

Beispiel

- Einstufiger Cache
  - $p = 70\%$
  - $t_1 = 400\mu s$
  - $t_2 = 7ms$
- Durchschnittliche Bearbeitungszeit

$$t = 0.4ms + 0.3 * 7ms = 2.5ms$$

- Auslastung

$$U = 0.4\text{ms} / 2.5\text{ms} = 0.16$$

- Anzahl der notwendigen Threads

$$n = 1 / 0.16 = 6.25$$

→ 7 Threads notwendig

World Wide Web (WWW)

HTML und CSS

Hypertext Markup Language (HTML)

- Beschreibung der logischen Struktur eines Dokumentes
  - Markierungen (Tags) zur Bezeichnung von Dokumentteilen
  - Beispiel: `<p>Ein Paragraph</p>`
  - Vordefinierte, nicht erweiterbare Menge an Tags → logische Struktur nur bedingt abbildbar
- Beschreibung des Layouts von Dokumenten
  - Unpräzise Formatierungstags → konkretes Layout bestimmt Client
  - Beispiel: `<b>Fettgedruckter Text<b>`
  - Genauere Layoutdefinitionen mittels Cascading Stylesheet (CSS)
- Vernetzung von Dokumenten
  - Verweise auf Seiten/Dokumente/Objekte → Link
  - Einbinden von (Multimedia-) Objekten (Grafik, Animation, Audio,…)
- Basiert auf Standard Generalized Markup Language (SGML)
  - ISO Standard 8879 mit flexibel erweiterbarer Tag-Menge

HTML Standards

- HTML 4.01 <http://www.w3.org/TR/html401/>
  - 3 Document Type Definitions (DTDs)
    - Strict: Keine veralteten (deprecated) Elemente
    - Transitional: Veraltete Elemente, jedoch keine Rahmen (Frames)
    - Frameset: Veraltete Elemente und Rahmen
  - Aktuelle und letzte Version

- XHTML 1.1 <http://www.w3.org/TR/xhtml11/>
  - Basiert auf XML
  - Umstellung auf XHTML ist zu erwarten

- Validation
  - Prüfung der Konformität zum Standard
  - W3C-Online-Validator <http://validator.w3.org/>

HTML-Tags

- Im Allgemeinen paarweise (öffnend und schließend)

```
<tag> … </tag>
```

- Keine Unterscheidung von Groß- und Kleinschreibung

```
<TitlE> … <tITle>
```

- Beliebige Verschachtelung, aber keine teilweise Überlappung

```
<h1>Ein <b>ungueltiges HTML-Konstrukt.</h1></b>
```

- Manche Tags können/müssen zusätzliche Attribute enthalten

```
<a href="http://www.tu-berlin.de">
```

- Browser-Hersteller versuch(t)en eigene Tags „durchzusetzen“
  → Kompatibilitätsprobleme

Beispiele (HTML-Tags)

`<html> …… </html>`	Start einer HTML-Seite
`<head> …… </head>`	Kopf der Seite
`<title> …… </title>`	Titel der Seite
`<body> …… </body>`	Rumpf der Seite
`<h`*n*`> …… </h`*n*`>`	Überschrift der Stufe *n*
`<b> …… </b>`	Text fett drucken
`<ul> …… </ul>`	Ungeordnete Liste
`<ol> …… </ol>`	Geordnete Liste
`<li> …… </li>`	Listenelement
` `	Neue Zeile
`<a href="url"> …… </a>`	Hyperlink auf andere Seite
`<img src="url">`	Laden eines Bildes
`<pre> …… </pre>`	Vorformatierter Text

HTML Grundstruktur

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
            "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta name="author" content="user@kbs">
    <title>Titel der Seite</title>
  </head>
  <body>
    <h1>Überschrift</h1>
    Eigentlicher Text
  </body>
</html>
```

HTML Listen

```
<ul>
  <li>Element 1</li>
  <li>Element 2</li>
  <li>Element 3
    <ol>
      <li>Element 3.1</li>
      <li>Element 3.2</li>
    </ol>
  </li>
</ul>
```

HTML Tabellen

```
<table>
  <caption>Preisvergleich</caption>
  <thead>
    <tr><th></th><th>Anbieter1</th><th>Anbieter2</th></tr>
  </thead>
  <tfoot>
    <tr><th>Gesamt</th><td>Summe1</td><td>Summe2</td></tr>
  </tfoot>
  <tbody>
    <tr><th>Artikel1</th><td>Pos1-1</td><td>Pos1-2</td></tr>
    <tr><th>Artikel2</th><td>Pos2-1</td><td>Pos2-2</td></tr>
  </tbody>
</table>
```

HTML Formulare

```
<form action="mailto:abc@email.xyz" method="post">
  <b>Name:</b>
  <input type="text" name="name" size="30"><br>
  <input type="submit" value="Senden"><br>
  <input type="reset" value="Neu">
</form>
```

```
<form action="http://www.a.com/cgi-bin/submit.pl" method="get">
  <input type="hidden" name="session_id" value="423914556">
  <input type="text" name="name" size="30"><br>
  <input type="submit" value="Senden">
</form>
```

HTML Frames

```
<html>
  <head>
    <title>Beispiel - Frames</title>
  </head>
  <frameset rows="20%,80%">
    <frame src="verweise.htm" name="Navigation">
    <frame src="startseite.htm" name="Daten">
    <noframes>
      Wird angezeigt, wenn der Browser keine
      Frames anzeigen kann.
    </noframes>
  </frameset>
</html>
```

Cascading Style Sheets (CSS)

- Ergänzender Standard zu HTML
  - Aktuelle Version CSS 2.0 <http://www.w3.org/TR/REC-CSS2/>
  - Working Drafts zu CSS 2.1 → vielfach (teilweise) unterstützt
  - Working Drafts zu CSS 3 Modulen → flexibler, modularer Aufbau
- Definieren die Formatierung einzelner HTML-Elemente
  - Abstände, Schriftart, Größe, Farbe, Fettdruck, etc.
  - Definition entweder
    - zentral für ein Dokument im `<head>` Element,
    - für einzelne Elemente (inline) oder
    - in einer extra Datei (z.B. `style.css`), die eingebunden wird.
  - Alle Möglichkeiten können auch kombiniert werden
    - Vorrang: inline vor zentral vor extern
- Validation
  - W3 Validator <http://jigsaw.w3.org/css-validator/>

Zentrale CSS Definition im Header

```
<html>
  <head>
    <style type="text/css"> <!--
      h1 { font-size:24pt; color:#FF0000;
           font-style:italic; }
      h1.hinterlegt { background-color:#FFFF00;}
      *.hinterlegt  { background-color:#FF0000;} -->
    </style>
  </head>
  <body>
    <h1 class="hinterlegt">Überschrift</h1>
    <p class="hinterlegt">Paragraph</p>
  </body>
</html>
```

Einbindung einer CSS-Datei & Inline-Definition

```
<html>
  <head>
    <title>Titel der Datei</title>
    <link rel="stylesheet" type="text/css“
          href="style.css">
    <style type="text/css"><!--
      h1 { font-size:48pt; } -->
    </style
  </head>
  <body>
    <p class="normal" style="color:#FF0000;">
      Element-bezogene Definition
    </p>
  </body>
</html>
```

Webanwendungen

Anforderungen für Web-Anwendungen

- Dynamische Inhalte
- Zustandsverwaltung
- Authentifizierung/Autorisierung
- Personalisierung
- …

Statische Hypertextdokumente sind für interaktive Anwendungen nicht ausreichend.

Techniken für Web-Anwendungen

- Vielzahl verschiedener Ansätze und Techniken
  - Unterschied in Details → klare Klassifizierung schwer
  - Großes Potential durch Kombination mehrerer Techniken
- Klassifizierung in verschiedenen Dimensionen möglich
  - Client vs. Server
  - Abstraktion
  - Funktionsumfang
  - Sicherheit
  - …

Der folgende Überblick über existierende Technologien erhebt keinen Anspruch auf Vollständigkeit.

Client vs. Server

Client

- Nutzung der „eingebauten“ Fähigkeiten des Browsers
- Einbindung nachladbarer/ installierbarer Plugins
- Programmierung der Plugins
- Beispiele
  - DHTML/JavaScript
  - ActiveX
  - Java Applets
  - Flash
  - …

Server

- Delegation des HTTP Request an externe Komponenten
- Weiterleitung aufgrund URL
  → Dispatching
- Komponenten erzeugen Response
- Beispiele
  - Server-side Includes (SSI), PHP
  - Common Gateway Interface (CGI)
  - Java Server Pages (JSP), Servlets
  - ASP.NET
  - …

Die Dynamik der Anwendung kann sowohl im Client als auch auf dem Server realisiert werden.

Laufzeitumgebung

Mit Laufzeitumgebung

- Container bietet zusätzliche Funktionalität
  - Sitzung
  - Authentifizierung und Autorisierung
  - Ressourcenüberwachung
  - ...
- Beispiele
  - Servlets/JSP
  - ASP.NET
  - ...

Ohne Laufzeitumgebung

- Systemprozesse
- Verantwortung des Programmierers für
  - Sitzungsverwaltung
  - Sicherheit
  - ...
- Beispiele
  - CGI
  - FCGI
  - ...

Abstraktion

Request/Response

- Komponente analysiert Request
- Komponente erzeugt Response direkt (z.B. HTML)
- Beispiele
  - CGI
  - Java Servlets

Seitenbeschreibung

- Spezieller Markup für dynamische Elemente
- Wird bei jeder Anfrage bearbeitet und und im HTML Dokument ersetzt
- Beispiele
  - JSP
  - PHP

Objekt-/Komponentenorientiert

- Seite besteht aus vorgefertigten Komponenten
- Ereignisbasiertes Ausführungsmodell
- Beispiele
  - ASP.NET
  - JSF

Komponierbarkeit

Niedrige Komponierbarkeit

- Konkatenation verschiedener Ausgabeströme
- Beispiele
  - Servlets
  - CGI

Hohe Komponierbarkeit

- Reiches Typsystem ermöglicht Orchestrierung einzelner Komponenten
- Beispiele
  - ASP.NET
  - JSP

Serverseitige Technologien

Überblick Serverseitige Technologien

- Server Side Includes (SSI)
- Common Gateway Interface (CGI)
- Hypertext Preprocessor (PHP)
- Servlets[1]
- Java Server Pages (JSP)[1]
- Active Server Pages (ASP.NET)[2]

[1] Besprechung als Teil von J2EE

[2] Besprechung als Teil von .Net

Server Side Includes (SSI)

- Anweisungen in HTML-Seiten → interpretiert vom Web Server
- Beispiele:
  - Datum/Zeit einblenden
    ```
    <!--#echo var="DATE_LOCAL" -->
    ```
  - CGI-Skript ausführen
    ```
    <!--#include virtual="/cgi-bin/counter.pl" -->
    ```
  - Befehl ausführen
    ```
    <!--#exec cmd="ls" -->
    ```
  - Änderungsdatum
    ```
    <!--#echo var="LAST_MODIFIED" -->
    ```
  - Einbinden einer Datei auf dem gleichen Server
    ```
    <!--#include virtual="/footer.html" -->
    ```

Common Gateway Interface (CGI)

- CGI definiert Schnittstelle für Aufruf von Programmen und Übergabe von Argumenten
  - Aufruf durch Starten eines neuen Prozesses
  - Standardausgabe des Prozesses ist Response
- Programmiersprache beliebig
  - Häufig Skriptsprachen: Shells, Perl, Python, ...
  - Aber auch: C, C++, etc.

Aufruf eines CGI-Skriptes

- Formular

```
<form action="/cgi-bin/guestbook.pl" method="get">
<form action="/cgi-bin/stats.pl" method="post">
```

- Verweis

```
<a href="/cgi-bin/statistik.pl">Tagesstatistik</a>
```

- Grafikreferenz

```
<img src="/cgi-bin/counter.pl">
```

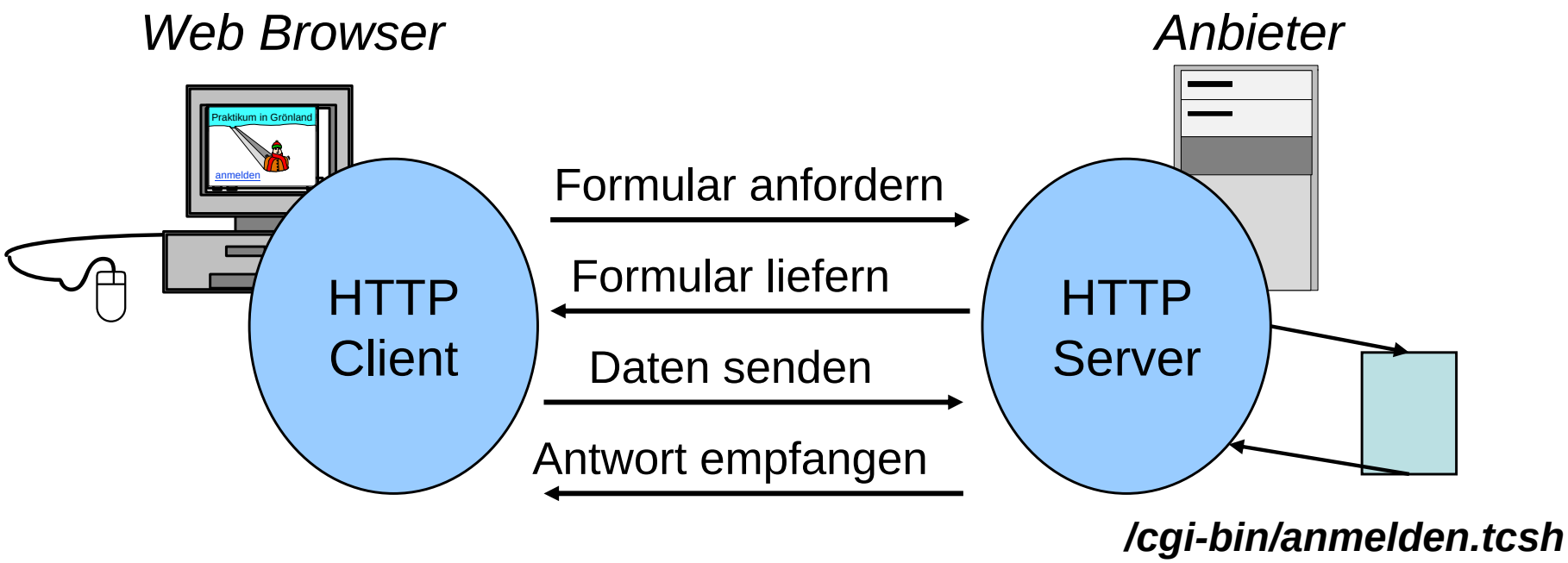
- Server Side Include (SSI)

```
<!-- #exec cgi="/cgi-bin/counter.pl" -->
```

- Automatisches Laden

```
<meta http-equiv="refresh" content="0;
      URL=/cgi-bin/welcome.pl">
```

Konfiguration und Ablauf



- Ausführbares Skript zur Behandlung der Eingaben
- Per Konvention im Verzeichnis /cgi-bin/
- Aufruf mit Parametern vom Client
- Antwort vom Server meist mit dynamisch generierter Seite

CGI Parameterübergabe

- GET
  - `$QUERY_STRING` (Umgebungsvariable) wird übergeben
  - Zum Beispiel

    ```
    ?vorname=Hans&nachname=M%FCller&...
    ?query=CGI+Tutorial
    ```

  - Beinhaltet alle Parameter in einem String (Leerzeichen durch + ersetzt)
  - Wird vom Skript geparsed und entsprechend verarbeitet
- POST
  - Einlesen der Parameter aus der Standardeingabe
- Skript antwortet mit neuer Seite (zumindest HTTP-Header)

```
echo "Content-type: text/html"
echo
```

Beispiel eines CGI-Skriptes

```
#!/bin/sh
cat - <<EOF
Content-type: text/html

<HTML><HEAD><TITLE>Datum und Zeit</TITLE></HEAD>
<BODY><H1>Zeit und Datum:
EOF
date
cat - <<EOF
</H1></BODY></HTML>
EOF
```

CGI-Sicherheit

- CGI-Programme sind „von jedem“ aufrufbar
  → keine systematische Sicherheit

- Ausführung eines Programms ohne Benutzerkonto auf der Maschine → Berechtigung des CGI-Programms ist die Berechtigung des WWW-Servers
  - Server nicht als root starten, sondern als wwwuser
  - Alternativ kann CGI-Programm mit UID ihres Besitzers laufen

- Problem der Ausführung dynamisch erzeugter System-Kommandos (Injection)

Diskussion CGI

- Vorteile
  - Geringe Anforderungen
  - Verfügbarkeit auf Server
  - Hohe Flexibilität
- Nachteile
  - Niedrige Abstraktion
  - Sicherheit dem Anwendungsentwickler überlassen (anfällig)
  - Hoher Ressourcenverbrauch (Prozess pro Request)
  - Keine Ressourcenüberwachung

Hypertext Preprocessor (PHP)

- Viel verwendete Serverseitige Skript Sprache
- PHP ist Open Source Software (OSS) [http://www.php.net]
- Oft in Verbindung mit MySQL verwendet
- Verfügbar für viele Plattformen (Windows, Linux, Unix, etc.)
- Einfaches Beispiel

```
<html>
  <head>
    <title>PHP-Test</title>
  </head>
  <body>
    <?php echo "<p>Hallo Welt</p>"; ?>
  </body>
</html>
```

PHP und HTML gemischt

```
<?php
if (strstr($_SERVER["HTTP_USER_AGENT"], "MSIE")) {
?>
<h3>strstr muss true zurückgegeben haben</h3>
<center><b>Sie benutzen Internet Explorer</b></center>
<?php
} else {
?>
<h3>strstr muss false zurückgegeben haben</h3>
<center><b>Sie benutzen nicht Internet
Explorer</b></center>
<?php
}
?>
```

HTML Forms und PHP

- Form:

```
<form action="welcome.php" method="POST">
  Enter your name: <input type="text" name="name" />
  Enter your age: <input type="text" name="age" />
  <input type="submit" />
</form>
```

- Verarbeiten der Formulardaten:

```
<html>
  <body>
    Welcome <?php echo $_POST["name"]; ?>.
    You are <?php echo $_POST["age"]; ?> years old!
  </body>
</html>
```

PHP Cookies

- Cooky setzen:

```
<?php
  setcookie("uname", $name, time()+36000);
?>
```

- Cooky abfragen:

```
<?php
if (isset($uname))
  echo "Welcome " . $uname . "!<br />";
else
  echo "You are not logged in!<br />";
?>
```

ODBC Datenbankzugriff mit PHP

```
<?php
$conn=odbc_connect('northwind','','');
if (!$conn) { exit("Connection Failed: " . $conn); }
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs) {exit("Error in SQL");}
echo "<table><tr><th>Firma</th><th>Kontakt</th></tr>";
while (odbc_fetch_row($rs)) {
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
}
odbc_close($conn);
echo "</table>";
?>
```

Java

- Java ist eine objekt-orientierte Sprache
- Einfachvererbung von Klassen, Mehrfachvererbung von Schnittstellen
- Java-Programme werden meist in Bytecode übersetzt und von einer virtuellen Maschine (VM) durch einen Bytecode-Interpreter ausgeführt
- Hierdurch wird (zu einem gewissen Grad) Plattform-Unabhängigkeit sichergestellt
- Lediglich die VM muss auf dem Zielsystem implementiert werden
- Keine neue Idee, siehe z.B. p-code bei UCSD Pascal
- Evtl. Kompilierung in Maschinensprache, z.B. mittels Just in Time Compiler (JITC)
- Gut geeignet für mobilen Code (mobile Agenten etc.)

Java-basierte Internet-Technologien

- Java Applets
  - In Web-Seiten eingebettete Java-Programme
  - Werden vom Java-fähigen Browser geladen und ausgeführt
- Java Servlets
  - CGI auf „Java-Art“
  - Erweitern Web-Server um dynamische, Java-basierte Seiten
  - Werden über URLs angesprochen und liefern Seiteninhalt
  - Werden vom Web Server instanziiert und ausgeführt
- Java Server Pages (JSP)
  - Erweiterung der Servlet Technologie zur einfachen Anfertigung von Web-Seiten, die statischen und dynamischen Inhalt mischen

Clientseitige Technologien

Überblick Clientseitige Technologien

- JavaScript
- Java Applets
- LiveConnect
- Pushlets

JavaScript

- JavaScript ermöglicht Aktivität auf Client-Seite
- Von Netscape lizenzierte objektorientierte Skriptsprache; aktuelle Version 1.5 (Microsoft Pendant ist JScript)
- Auch als ECMA (European Computer Manufacturers Association)-Standard (ECMA-262) untern dem Namen ECMASkript verabschiedet
- Keine zusätzliche Kommunikation zwischen Web Server und Web Client
- HTML+CSS+JavaScript = Dynamic HTML (DHTML)

JavaScript Anwendungsbeispiele

- Testen von Formulardaten auf Korrektheit vor Absendung
- Browsereigenschaften erfragen
  - Hersteller und Version
  - Multimediaunterstützung
  - …
- Auf Nutzer-Events reagieren
  - Mausklicks
  - Tastatureingaben
  - …
- Sich verändernde, dynamische Web-Seiten
  - aktuelle Uhrzeit einblenden
  - Aufklappen/Einklappen von Paragraphen
  - Bewegen von Objekten
  - …

JavaScript

- JavaScripts werden wahlweise direkt in der HTML-Datei oder in separaten Dateien notiert
- Browser besitzen eingebauten JavaScript Interpreter (viele Inkompatibilitäten)
- Aus Sicherheitsgründen eingeschränkte Rechte → Sandbox
  - Kein Zugriff auf das Dateisystem
  - Keine Arbeitsspeicherverwaltung
  - …
- Beispiel für Inline-Skript

```
<script type="text/javascript">
<!–
  var d = new Date()
  document.write(d.getHours()+":"+d.getMinutes())
//-->
</script>
```

Einbindung externer JavaScript-Dateien

- Ermöglicht die Wiederverwendung von Code
- Darf nur JavaScript enthalten
- Beispiel: product.js:

```
function Produkt(x,y) {
  var z = x*y;
  return z;
}
```

Einbindung externer JavaScript-Dateien

```
<html><head>
  <title>JavaScript-Test</title>
  <script src="produkt.js" type="text/javascript"></script>
</head><body>
  <form name="Form" action="">
  <input type="text" name="x" size="3">*
  <input type="text" name="y" size="3">
  <input type="button" value="=" onClick=
    "document.Form.p.value=Produkt(document.Form.x.value
                                   document.Form.y.value)">
  <input type="text" name=„p" size="3">
  </form>
</body></html>
```

Javascript und DOM

- Zugriff auf Elemente über das id-Attribut

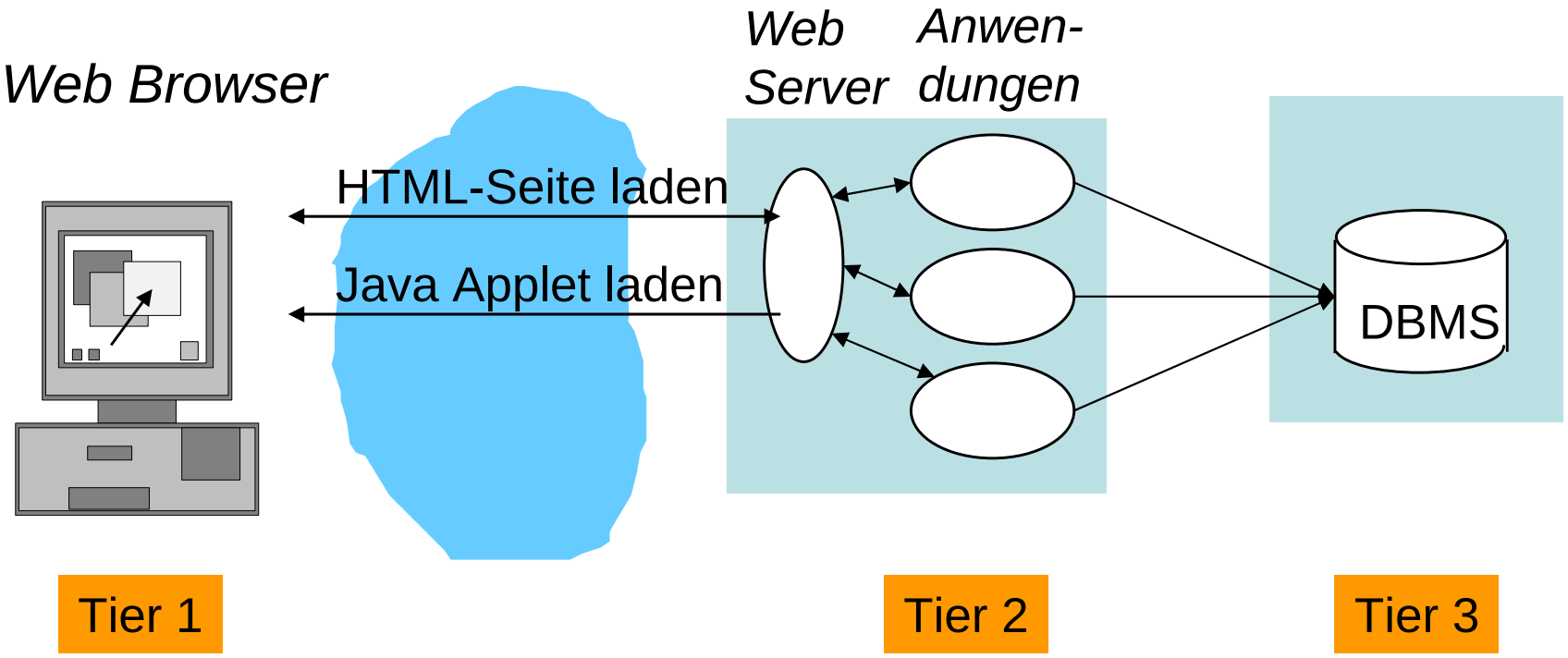
```
<html>
  <body>
    <h1 id="header">My header</h1>
    <script type="text/javascript">
      document.getElementById("header").style.color="red"
    </script>
  </body>
</html>
```

Java Applets

- Ermöglichen Aktivität auf Client-Seite
- Haben (meist) eine GUI (AWT, Swing, etc.)
- Können (fast) die gesamte Java-API nutzen
- HTML-Tag `<applet>` (deprecated!)

```
<applet code="chess.class"
        codebase="http://kbs.cs.tu-berlin.de/applets"
        archive="games.jar" width="400" height="400">
  <param name="color" value="0xFF0000">
  Your Browser ignores the APPLET tag!
</applet>
```

Java Applets



tier (engl.) = Stufe, Schicht

Java Applets

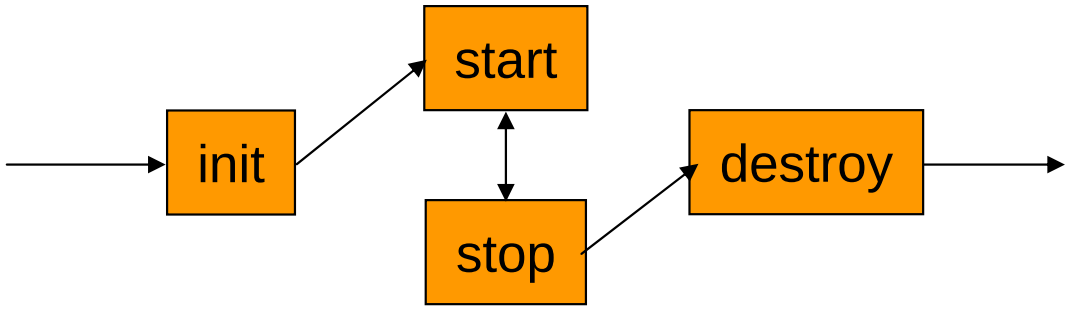
- Applets leiten von java.applet.Applet ab, welche von Panel, Container, Component und Object abgeleitet ist
- Geerbte, überschreibbare Grafik-Methoden, z.B.

```
public void paint(Graphics g)
  // does nothing by default
  // is overridden to do some drawing
public void update(Graphics g)
  // default implementation clears area and calls paint()
  // causes a lot of flickering
public void repaint()
  // called by the Applet itself
  // to trigger call of update()
```

- Methoden zur Ereignisbehandlung (Listener-Modell), z.B.

```
public void addMouseListener(MouseListener l)
public void removeMouseListener(MouseListener l)
protected void processMouseEvent(MouseEvent e)
```

Lebenszyklus eines Java Applets



- Entsprechende Methoden können überschrieben werden

```
public void init()
public void start()
public void stop()
public void destroy()
```

Java Applet Beispiel

```
public class Simple extends Applet {
  StringBuffer buffer = new StringBuffer();

  public void init()    { addItem("init... "); }
  public void start()   { addItem("start... "); }
  public void stop()    { addItem("stop... "); }
  public void destroy() { addItem("destroy..."); }

  void addItem(String newWord) {
    buffer.append(newWord);
    repaint();
  }

  public void paint(Graphics g) {
    g.drawString(buffer.toString(), 5, 15);
  }
}
```

Applet-spezifische Methoden

- Häufig genutzte Applet-spezifische Methoden

```
public String[][] getParameterInfo()
public AppletContext getAppletContext()
public String getParameter(String name)
public URL getCodeBase()
public URL getDocumentBase()
public Image getImage(URL url)
public AudioClip getAudioClip(URL url)
public void showStatus(String msg)
```

Sicherheit von Java Applets

- Ausführung geladener Applets birgt offensichtliche Risiken
- Java enthält einige Sicherheitsvorkehrungen
  - Keine Zeiger
  - Typsicherheit zur Laufzeit (nicht zum Zeitpunkt der Übersetzung)
  - Echte Feldgrenzen mit Überprüfung
  - Byte Code Verifier kann geladenen Byte Code verifizieren
- Applets unterliegen standardmäßig weiteren Sicherheitsvorkehrungen
  - Kein Zugriff auf Dateien
  - Kommunikation nur mit dem Ursprungsrechner des Applets
  - Bestimmte Systemparameter können nicht ausgelesen werden
  - Applet-Fenster sehen anders aus, als normale Fenster
  - ...

LiveConnect

- Ermöglicht die Kommunikation zwischen Java, Javascript und Plugins (ab Netscape 3.0)

1. Java → JavaScript

```
<APPLET CODE="MyApplet1.class" NAME="MyApplet"
WIDTH=150 HEIGHT=25 MAYSCRIPT></APPLET>
```

```
import netscape.javascript.*;
...
public writePage(String page) {
  JSObject win = JSObject.getWindow(applet);
  win.eval("document.open();");
  win.eval("document.write('"+page+"');");
  win.eval("document.close();");
}
```

LiveConnect

2. JavaScript → Java

```
public MyApplet1 extends Applet {
    public void setString(String s) {...}
    ...
}
```

```
<APPLET CODE="MyApplet1.class" NAME="MyApplet"
  WIDTH=150 HEIGHT=25></APPLET>

<FORM NAME="form">
  <INPUT TYPE="button" VALUE="Set"
    onClick="document.MyApplet.setString(
      document.form.str.value)">
  <INPUT TYPE="text" SIZE="20" NAME="str">
</FORM>
```

Server-to-Client Push

- Periodisches Pull (wirkt für den Client wie Push)
  - HTML REFRESH

```
<META HTTP-EQUIV="Refresh"
  CONTENT="4;URL=http://kbs.cs.tu-berlin.de">
```

- Realisierungsmöglichkeiten für Push
  - Serverseitige Callbacks
    - Server ruft Methoden eines Applets auf
    - Funktioniert z.B. mit Java RMI und Corba IIOP
    - (Unsichtbares) Applet setzt JavaScript über LiveConnect ab oder nutzt eigene GUI zur Darstellung
  - Messaging
    - Server schickt UDP-Pakete (Unicast oder Multicast) oder Daten über TCP-Streams an ein Applet
  - Pushlets
    - Server schickt per HTTP Streaming JavaScript an den Client

Pushlets

- Baut auf HTTP Streaming auf
  - HTTP Verbindung wird dauerhaft offengehalten
  - Über die Verbindung können weiterhin Daten gesendet werden
  - Wird von Multimediaanwendungen genutzt (QuickTime, Real Audio etc.)
- Dokument mit zwei Frames
  - Unsichtbarer Frame zum Empfang von JavaScript-Code
  - Sichtbarer Frame zur Darstellung
  - 2. unsichtbarer Frame kann als Ziel für HTTP GETs/POSTs genutzt werden, die lediglich der Übertragung von Daten zum Server dienen

```
window.frames['GetFrame'].location='http://kbs.cs.
  tu-berlin.de/servlet/aktien.jsp?s=ibm&a=add'
```

- Browser-unabhängige Bibliothek unter **http://www.pushlets.com**

Pushlets: Client Seite

```
<HTML><HEAD>
<script LANGUAGE="JavaScript">
  var pageStart="<HTML><HEAD></HEAD><BODY><H1>";
  var pageEnd="</H1></BODY></HTML>";
  function push(content) {
    window.frames['displayFrame'].document.open();
    window.frames['displayFrame'].document.
      writeln(pageStart+content+pageEnd);
    window.frames['displayFrame'].document.close();
  }
</script>
...
</HEAD>
<FRAMESET BORDER="0" COLS="*,0">
  <FRAME SRC="display.html" NAME="displayFrame">
  <FRAME SRC="pusher.jsp"   NAME="pushletFrame">
</FRAMESET>
</HTML>
```

Pushlets: Server Seite

```
<HTML>
  <HEAD>...</HEAD>
  <BODY>
    <%
      String jsPre =
        "<scriptlanguage=JavaScript>parent.push('";
      String jsPost = "')</script> ";
      for (int i=1; i < 10; i++) {
        out.print(jsPre+"Page "+i+jsPost);
        out.flush();
        try {Thread.sleep(3000);}
        catch (InterruptedException e) {}
      }
      out.print(jsPre+"DONE"+jsPost);
    %>
  </BODY>
</HTML>
```

Pusher.jsp

Fragen?