7363 - Web-basierte Anwendungen

Eine Vertiefungsveranstaltung
mit Schwerpunkt auf XML-Technologien

Grundlagen:
HTML und XHTML

HTML und XHTML

- Vorbemerkungen
  - Dies ist kein Kurs, um (X)HTML zu lernen. Verwenden Sie dazu Material wie SelfHTML (http://de.selfhtml.org/).
  - Dieser Abschnitt stellt (X)HTML-Varianten kurz vor, stellt Zusammenhänge her (z.B. mit HTTP) und betont eher vernachlässigte Aspekte von HTML

- HTML 2.0, RFC 1866 (1995)
  - Quelle: http://www.ietf.org/rfc/rfc1866.txt
  - Dokumententyp-Deklaration:
    ```
    <!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
    ```
  - Bemerkungen:
    - Der RFC ist recht gut lesbar und enthält die SGML DTD
    - Er fasst den Status von HTML bis etwa Juni 1994 formal zusammen.
    - Vorher, d.h. seit den Anfängen von HTML in 1990, gab es offenbar keinen verabschiedeten HTML-Standard, sondern informelle Beschreibungen aus mehreren Quellen.

HTML und XHTML

- HTML 2.0: Einfluss auf HTTP

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
  <HEAD>
   <META HTTP-EQUIV="Expires" CONTENT="Tue, 04 Dec 1993 21:29:02 GMT">
   <meta http-equiv="Keywords" CONTENT="Fred">
   <META HTTP-EQUIV="Reply-to" content="fielding@ics.uci.edu (Roy
  Fielding)">
   <Meta Http-equiv="Keywords" CONTENT="Barney">
   <Title></Title>
  </HEAD>
  <BODY>
   <H1> Wie HTML auf HTTP-Header wirken kann</H1>
   <p> Die META-Elemente stammen direkt aus dem RFC 1866.
       Wir diskutieren ihre Wirkung auf der naechsten Seite.
  </BODY>
</HTML>
```

HTML und XHTML

- HTML 2.0: Bemerkungen
  - Die Grundlage ist SGML statt XML:
    - HTML unterscheidet Klein- und Großschrift in Tags und Attributnamen nicht.
    - Das Schließen von Elementen wird lax gehandhabt.
    - Keine SystemID in der Dokumententyp-Deklaration.
  - Die Wirkung des META-Attributs "HTTP-EQUIV" auf HTTP-Header
    - Annahme: Web-Server lesen den "HEAD"-Teil von HTML-Seiten mit.
    - Web-Server könnten dann HTTP-Header aus dem HTML-Dokument erzeugen, im o.g. Beispiel etwa:

```
Expires: Tue, 04 Dec 1993 21:29:02 GMT
Keywords: Fred, Barney
Reply-to: fielding@ics.uci.edu (Roy Fielding)
```

  - Bemerkungen
    - `Keywords, Reply-to`: Non-standard HTTP Header (RFC 822 Header)
    - `Expires`: So können HTML-Autoren Gültigkeiten begrenzen.
    - `Last-Modified, Date` u.a. Server-Header so nicht ändern!

HTML und XHTML

- HTML 3.2, W3C (1997)
  - Quelle: http://www.w3.org/TR/REC-html32/
  - Dokumententyp-Deklaration:
    ```
    <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
    ```
  - Bemerkungen:
    - Diese W3C-Empfehlung sollte HTML 2.0 ersetzen, der Zwischenschritt HTML 3.0 (www.w3.org/MarkUp/html3/CoverPage) erreichte keinen Standard-Status.
    - Sie entspricht dem Entwicklungsstand von HTML Anfang 1996
    - Das Dokument ist gut lesbar
    - Abwärtskompatibel zu HTML 2.0
    - Neu: Tabellen, Applets (Java!), "Textfluss" um Bilder
    - In Entwicklung dabei: Stylesheets (CSS)

HTML und XHTML

- HTML und Dublin Core
  - Quelle: **`http://dublincore.org/`**
  - HTML-Dokumente sollten leicht zu finden sein, etwa über Suchmaschinen.
  - Durch Beachtung der Dublin Core-Regeln können Sie Ihre Seiten mit Meta-Informationen ergänzen, die zu guten Suchtreffern führen.

- Hinweis auf die Benutzung des META-Elements
  - Normalfall: **`<META name="einName" content="einInhalt">`**
  - Namenswerte gemäß Dublin Core:
    - Allgemein: Präfix "DC.", gefolgt vom eigentlichen Namen. Beispiele:
    - `DC.Title, DC.Creator, DC.Subject, DC.Description, DC.Publisher, DC.Contributor, DC.Date, DC.Type, DC.Format, DC.Language, ...`
    - Siehe auch **`http://de.selfhtml.org/html/kopfdaten/meta.htm`**

HTML und XHTML

- HTML 4.0, W3C (Dez. 1997 - Apr 1998)
  - Quellen: http://www.w3.org/TR/REC-html40-19971224/,
    http://www.w3.org/TR/1998/REC-html40-19980424/
  - Dokumententyp-Deklarationen:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN"
        "http://www.w3.org/TR/html4/strict.dtd>
```

```
- <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd>
```

```
- <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN"
        "http://www.w3.org/TR/html4/frameset.dtd>
```

- Bemerkungen:
  - Benutzen Sie das Bugfix-Release 4.01 !

HTML und XHTML

- HTML 4.01, W3C (24. Dez. 1999)
  - Quellen: http://www.w3.org/TR/html401
  - Dokumententyp-Deklarationen:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html401/strict.dtd>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html401/loose.dtd>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
        "http://www.w3.org/TR/html401/frameset.dtd>
```

  - Neu in 4.0 (Einzelheiten: Anhang A der TR):
    - Unterstützt mehr Multimedia-Optionen, Skriptsprachen, Stylesheets
    - Verbessert Druckoptionen, barrierefreien Zugang, Internationalisierung (z.B. mehrere Textrichtungen), Tabellen
    - Das Frame-Konzept
    - Erstmals 3 DTD-Varianten: Strict, Transitional, Frameset
  - Bemerkungen:
    - Release 4.01 ersetzt frühere 4.0-Versionen
    - Ausgangspunkt für die XHTML-Entwicklung

HTML und XHTML

- Anmerkungen zu den 3 Varianten von HTML 4.0x
  - Die lange Zeit von 2 Jahren zwischen dem ersten 4.0-Release und Release 4.01 deutet auf "politische" Hindernisse hin.
  - Die Zeit der "Browser-Kriege" hinterließ vermutlich Spuren. Um einerseits dem Status quo Rechnung zu tragen, andererseits eine klare Vorgabe zu machen für einen sauberen und tragfähigen neuen Standard, kennzeichnete man z.B. eine Reihe von Elementen als "missbilligt" (*deprecated*).
  - Die Variante "strict" setzt i.d.R. die Verwendung von CSS voraus. Sie strebt eine klarere Trennung zwischen Dokumenteninhalt und -struktur einerseits (HTML-Code) und Layoutsteuerung (CSS) an.
  - In Variante "strict" fehlen alle *deprecated elements*, in "transitional" sind sie noch enthalten.
  - Die Hoffnung bestand, dass die "transitional"-Variante bald ausstirbt.
    Noch heute scheint sie aber eher der Normalfall zu sein...
  - Die "frameset"-Variante baut auf der "transitional"-Variante auf und enthält zusätzlich die für Frames notwendigen Elemente.

HTML und XHTML

- ISO HTML 4 (ISO/IEC, 15.5.2000)
  - Quellen:
    http://www.cs.tcd.ie/15445/15445.html (Spec.)
    http://www.cs.tcd.ie/15445/UG.HTML (Users guide)

  - Dokumententyp-Deklarationen / FPI:

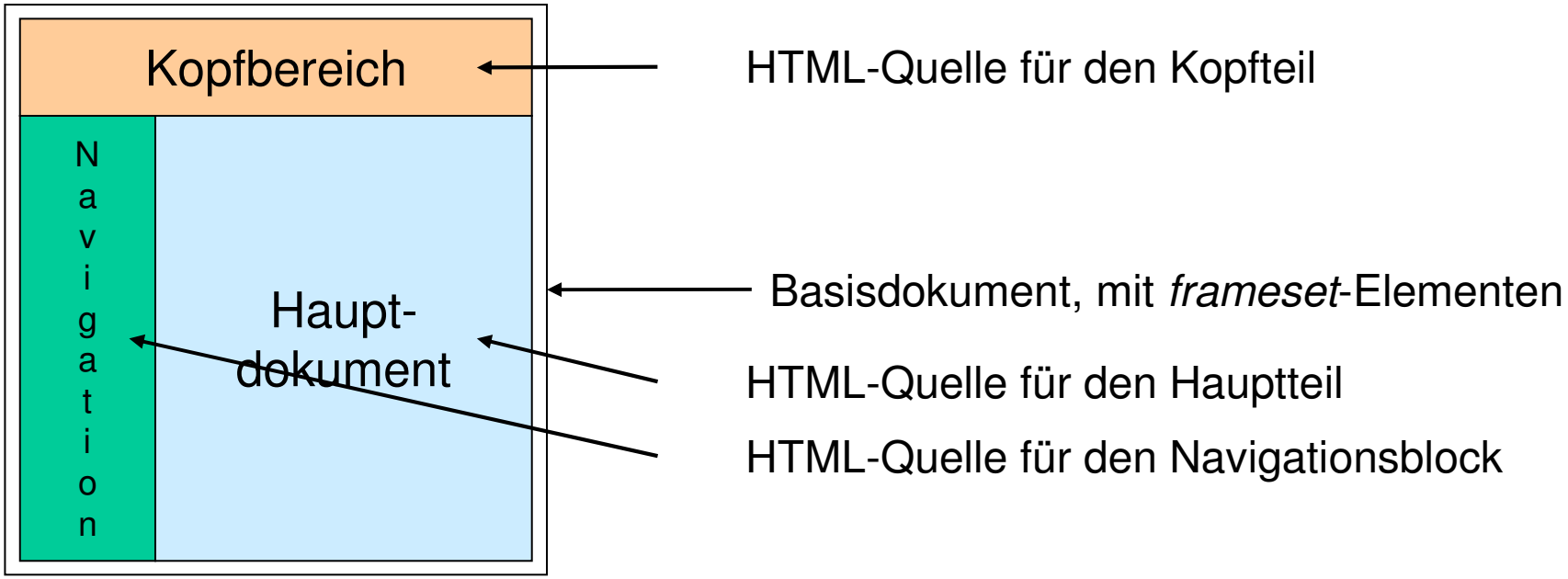
```
<!DOCTYPE HTML PUBLIC "ISO/IEC 15445:2000//DTD Hypertext Markup
Language//EN">
<!DOCTYPE HTML PUBLIC "ISO/IEC 15445:2000//DTD HTML//EN">
```

  - ISO/IEC 15445:2000 basiert auf HTML 4.01, "strict"-Variante
  - Es strebt strengere Regeln an als selbst HTML 4.01 „strict“, z.B. indem es fordert, dass zwischen einem "h1"- und einem "h3"-Element stets auch ein "h2"-Element auftritt.

  - Damit greift die Norm den Validierungsgedanken aus SGML wieder auf.
  - Trend inzwischen: XHTML und XML-basierte Validierung

HTML und XHTML

- Frames
  - Über Vor- und Nachteile von Frames ist viel diskutiert und geschrieben worden. Diese Diskussion soll hier unterbleiben.
  - Durch Frames wird die Anzeige unterteilt in Rechtecke, deren Inhalte jeweils von eigenen HTML-Dokumenten beschrieben werden.
  - Den Zusammenhang stellt ein Rahmendokument her. Beispiel:



HTML und XHTML

- Frames, Vorteile
  - Neue Möglichkeiten der Layoutkontrolle
  - Anzeigeteile lassen sich unabhängig voneinander verwalten
  - Einbinden fremder Seiten als Unterfenster
  - ...
- Frames, Nachteile
  - Einige Browser-Eigenheiten
  - Suche / Einträge in Suchmaschinen erschwert, da nur das praktisch leere Rahmendokument von außen auffindbar ist.
  - Schlecht zu verlinken
- Alternativen zur Layoutkontrolle
  - Tabellen
  - CSS (!)

HTML und XHTML

- XHTML 1.0, W3C (26. Jan. 2000, 1. Aug. 2002)
  - Quellen: http://www.w3.org/TR/xhtml1
  - Dokumententyp-Deklarationen / FPI:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-loose.dtd>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd>
```

  - **Reformulierung von HTML 4.01 in XML-Syntax**
    - Echte XML DTDs als Grundlage, Groß/Kleinschreibung nun wichtig
    - Neu: XML-Deklaration, *empty elements* (Bsp.: "
")
    - Strenge Einhaltung der Wohlgeformtheitsregeln gefordert
    - Beispiel: muss nun stets geschlossen werden ( )
    - Attributwerte stets in Anführungszeichen, ...
  - Anspruch:
    - XHTML löst HTML ab! Der Übergang zu XML als Basistechnologie des Internet soll gefördert werden.

HTML und XHTML

- XHTML Basic (W3C, 19.12.2000)
  - Quellen: http://www.w3.org/TR/xhtml-basic
  - Dokumententyp-Deklarationen / FPI:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
        "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd>
```

  - Konzept: **Reduzierung und Modularisierung von XHTML 1.0**
    - XHTML für PDAs und andere kleine, mobile Geräte
    - Enthält einen Minimalsatz an HTML-Elementen + einige Extras
    - Ein Basis-Satz von XHTML-Elementen zur Integration in größere DTDs und zu anderen Erweiterungen

HTML und XHTML

- XHTML Modularization (W3C, 10.04.2001)
  - Quellen: http://www.w3.org/TR/xhtml-modularization
  - FPI Konventionen für XHTML Module:

```
"-//MyCompany//DTD XHTML MyML 1.0//EN"
"-//MyCompany//ELEMENTS XHTML MyElements 1.0//EN"
"-//MyCompany//DTD Special Markup with XHTML//EN"
```

- Konzept:
  - Modularisierung der XHTML 1.0 DTD
  - Grundlage für weitergehende Entwicklungen

HTML und XHTML

- XHTML 1.1, W3C (31. Mai 2001)
  - Quellen: http://www.w3.org/TR/xhtml11
  - Dokumententyp-Deklarationen / FPI:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd>
```

  - Namensraum für XHTML

```
http://www.w3.org/1999/xhtml
```

  - Modularisierung von XHTML 1.0 "strict"
    - *Keine Fortsetzung von "transitional" und "frames"!*
    - Anwendung von "XHTML Modularization"
    - **Ca. 20 Module**
  - Anspruch:
    - Weiterentwicklung von XHTML 1.0
    - Grundlage für Schema-Version von XHTML (statt DTD)
    - Grundlage für zukünftige Entwicklungen, z.B. "XHTML Family doctypes" oder für die Integration ausgewählter Module in anderen Dokumenttypen.

HTML und XHTML

- Ein XHTML + MathML + SVG Profil (WD, 9.8.2002)
  - Quellen: http://www.w3.org/TR/XHTMLplusMathMLplusSVG
  - Dokumententyp-Deklarationen / FPI:

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
  "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
```

  - Ein Beispiel für die Kombination von Modulen
    - Dokumente, die aus Elementen dieser drei Standards bestehen, lassen sich dank Kombination von DTD-Modulen validieren.
    - Unterscheidung der Elemente über Namensräume!
  - Kritische Anmerkung:
    - Mit W3C XML Schema wäre eine derartige Modularisierung viel einfacher als mit den hier gewählten DTD-Mitteln.

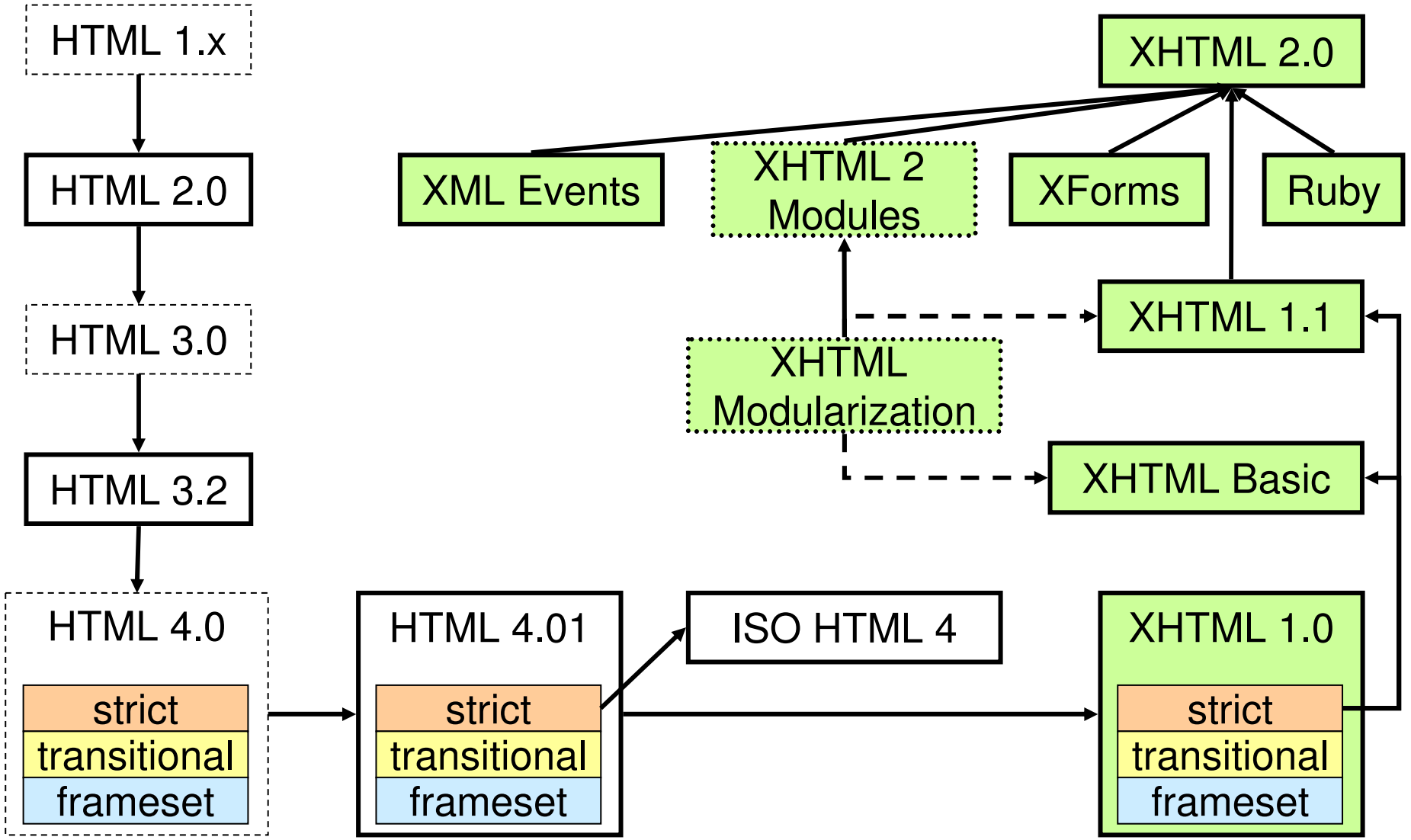
HTML und XHTML

- XHTML 2.0, W3C (WD, 27. Mai 2005)
  - Quellen: http://www.w3.org/TR/xhtml2
  - Dokumententyp-Deklarationen / FPI:

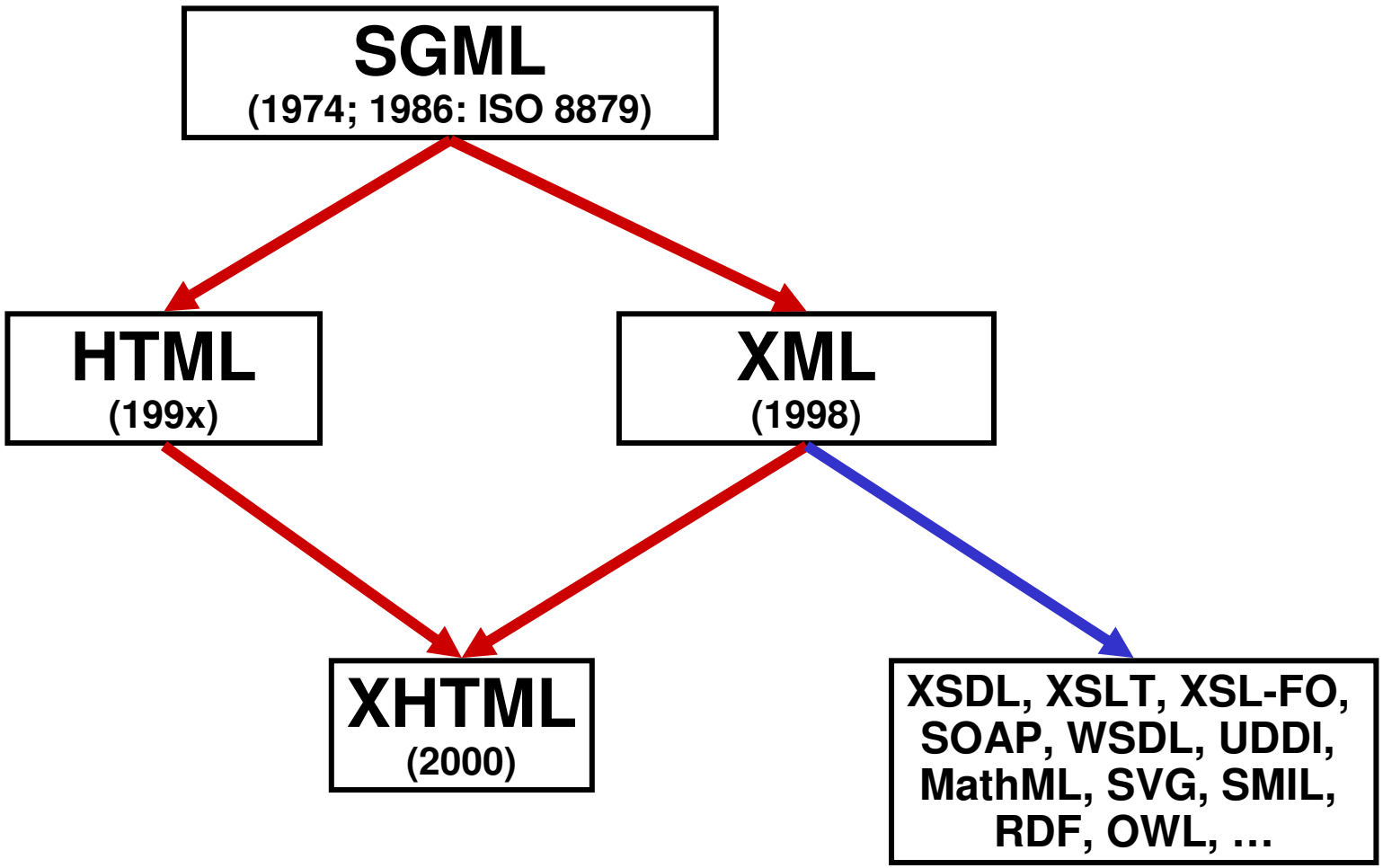
    `(tbd)`
    - Möglicherweise Umstellung von DTD auf RELAX NG (nicht: W3C Schema!)
  - Namensraum für XHTML

    `(tbd)`
  - Weiterentwicklung der Modularisierung von XHTML
  - Keine volle Abwärts-Kompatibilität!
  - Konsequent auf XML bauend.
  - Umbauten, Erweiterungen zu zahlreichen Elementen
  - Neu:
    - **XForms** statt HTML forms → erfordert neue Browser!
    - **XML Events** → erfordert neue Browser!
      Eine XML-Syntax zur Beschreibung von Ereignissen (z.B. Klicken auf *button*) und zur Assoziation mit Aktivitäten wie z.B. dem Start einer JavaScript-Funktion
  - Bem.:
    - Modul „Ruby“ hat nichts mit der Programmiersprache „Ruby“ zu tun…

HTML und XHTML: Übersicht



Von HTML zu XHTML



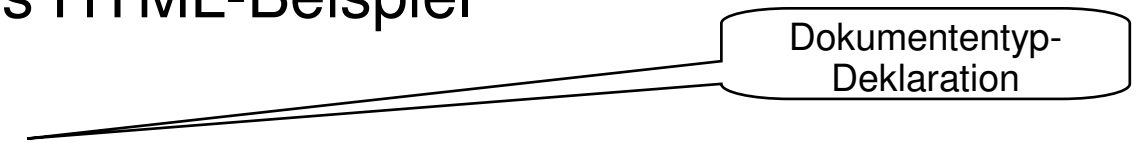
Von HTML zu XHTML

- XML-Dokument
  - Abstrakte Sicht: Bewerteter Graph in Baumform
  - Modell: XML Information Set (http://www.w3.org/TR/infoset)
  - Übliche Darstellung: XML-Syntax (http://www.w3.org/TR/xml10)
    - Bild eines Dokumentenbaumes: Siehe unten, Bild zum Beispiel
      - Ein Dokumentenknoten (root)
      - Ein Dokumentenelement
      - Kindelemente, Text/Char-Elemente
      - Werte: Mengen (von Attributen), Verweise, etc.

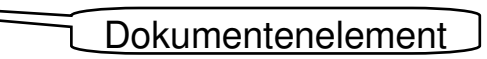
- Dokumenttyp
  - Eine Menge von Regeln, die präzise beschreibt, wie Dokumente dieses Typs aufzubauen sind **(welche Elemente sind wo wie oft zulässig, welche Attribute und Datentypen besitzen sie, etc.)**.
  - Definition per „DTD“, W3C XML Schema, RELAX NG, …

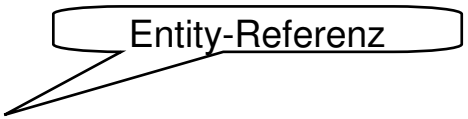
Struktur eines HTML-Dokuments

- Einfaches HTML-Beispiel




```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="de">
  <head>
    <title>Kleines XHTML-Beispiel</title>
  </head>
  <body>
    Hallo, Welt!
    <!-- Kommentar: Hier erg&auml;nzen! -->
  </body>
</html>
```

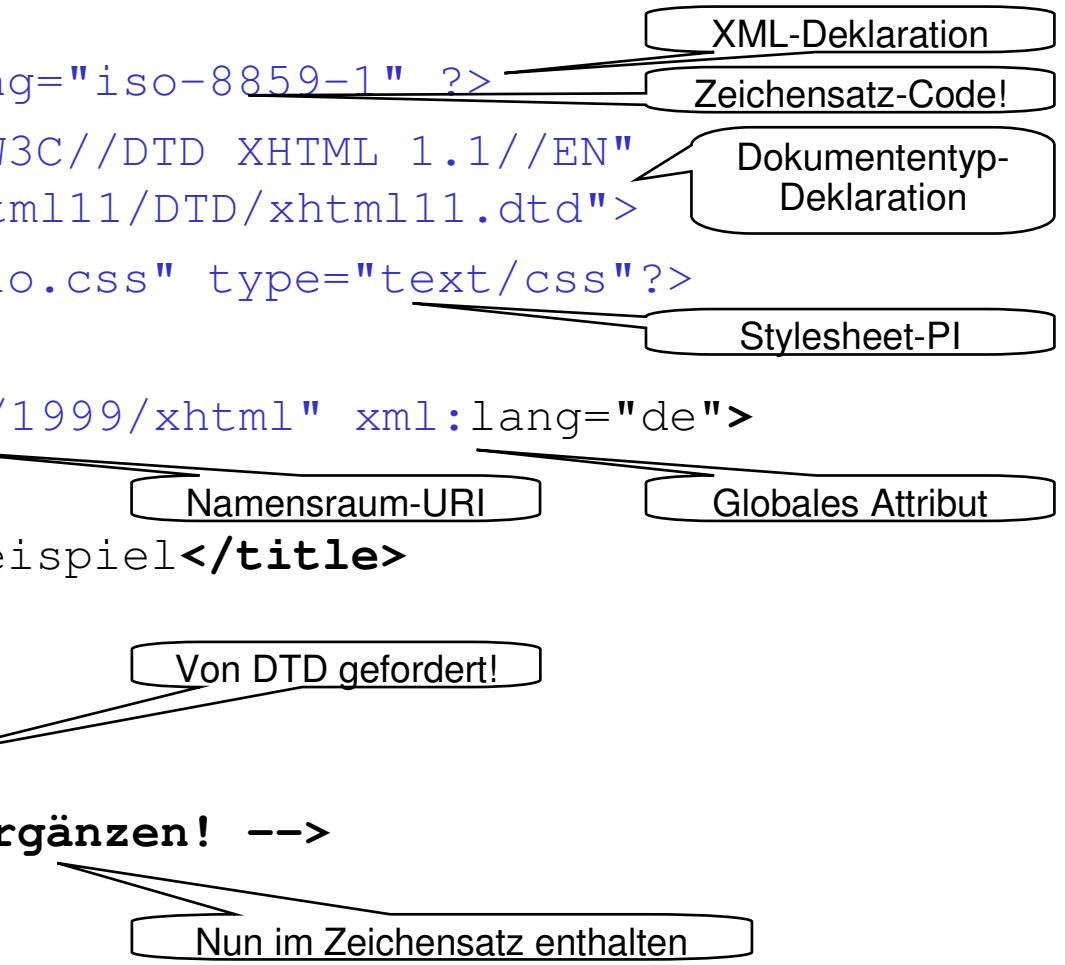




Struktur eines XHTML-Dokuments

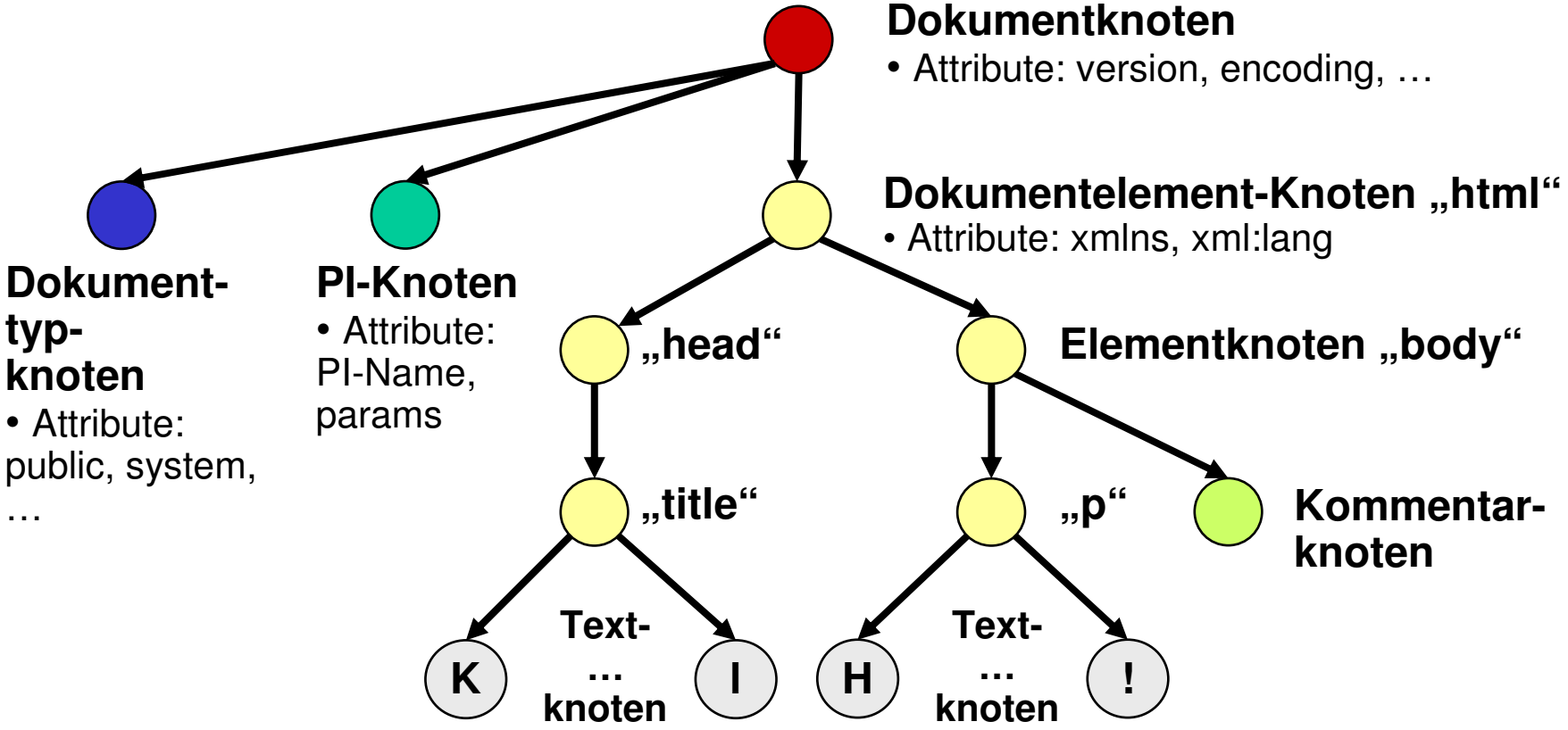
- Einfaches XHTML 1.1-Beispiel + Demo (html vs. xhtml)

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<?xml-stylesheet href="hello.css" type="text/css"?>
<html
  xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
  <head>
    <title>Kleines XHTML-Beispiel</title>
  </head>
  <body>
    <p>Hallo, Welt!</p>
    <!-- Kommentar: Hier ergänzen! -->
  </body>
</html>
```



Struktur eines XHTML-Dokuments

- XML-Dokumente:
  - Markierte (attributierte), baumartige Graphen!
    Besitzen verschiedene Knotentypen:



Validierung

- Analogie
  - Dokumententyp ⟷ Klasse
  - Dokument, D. instanz ⟷ Objekt

- Validierung – Qualitätssicherung von Dokumenten
  - Prüfung einer Dokumenteninstanz gegen ihr(e) zugrunde liegende(s) DTD/Schema
    - Werden nur zugelassene Elemente verwendet?
    - Stimmt die Elementreihenfolge und Häufigkeit?
    - Werden nur zugelassene Attribute verwendet?
    - Bei Schema: Stimmen die Inhalte mit den Datentypen überein?
  - Werkzeuge:
    - DTD- und/oder Schemavalidierer wie nsgmls, Xerces, …
      Speziell für die (X)HTML-Dokumenttypen: http://validator.w3.org

Die Module von XHTML 1.1

- Strukturmodul
  - body, head, html, title
- Textmodul
  - abbr, acronym, address, blockquote, br, cite, code, dfn, div, em, h1, h2, h3, h4, h5, h6, kbd, p, pre, q, samp, span, strong, var
- Hypertextmodul
  - a
- Listmodul
  - dl, dt, dd, ol, ul, li
- Objektmodul
  - object, param
- Präsentationsmodul
  - b, big, hr, i, small, sub, sup, tt
- Edit-Modul
  - del, ins
- *Bidirectional Text*-Modul
  - bdo
- Formularmodul
  - button, fieldset, form, input, label, legend, select, optgroup, option, textarea
- Tabellenmodul
  - caption, col, colgroup, table, tbody, td, tfoot, th, thead, tr
- *Image*-Modul
  - img
- *Client-side Image Map*-Modul
  - area, map
- *Server-side Image* Map-Modul
  - Attribute ismap on img
- *Intrinsic Events*-Modul
  - Events attributes
- Metainformationsmodul
  - meta
- Scriptingmodul
  - noscript, script
- *Stylesheet*-Modul
  - style element
- *Style Attribute*-Modul *Deprecated*
  - style attribute
- *Link*-Modul
  - link
- *Base*-Modul
  - base
- *Ruby Annotation*-Modul
  - ruby, rbc, rtc, rb, rt, rp

Durch CSS-Anweisungen zu ersetzen!

XHTML und CSS

- HTML war ursprünglich zur inhaltlichen, abstrakten Strukturierung von Dokumenten entworfen worden.
  Über die Art der Darstellung entschied der Browser.
  - Beispiel:

```
<h1>Überschrift</h1>
<p>Geben Sie <kbd>Strg-C</kbd> an, um ein
      Programm abzubrechen.</p>
```

- Spätere Sprachelemente ergänzten Darstellungsaspekte – und verletzten dadurch das Prinzip „Trennung von Inhalt & Darstellung“!
  - Beispiele:

```
<b>Fett</b> und <i>kursiv</i> gedruckte Wörter.
<p align="center">Ein zentrierter Absatz.</p>
```

- Cascading Stylesheets (CSS)
  - dienen ausschließlich der Darstellung von XHTML- und XML-Inhalten
  - sollen XHTML von Darstellungselementen wieder befreien.

Entity- und Zeichenreferenzen

- XHTML, XML und Unicode
  - Beliebige Unicode-Zeichen können in allen XML-Texten per Zeichenreferenz eingebunden werden. Beispiel:

```
Dies kostet <Preis>50 &#x20AC;</Preis>
```

Unicodewert für €

  - Die fünf für Markup reservierten Zeichen: < > & " ' lassen sich über folgende in XML vordefinierte Entity-Referenzen als normale Zeichen verwenden:

```
&lt; &gt; &amp; &quot; &apos;
<Relation> a &lt; b </Relation>
```

a < b

  - In XHTML sind ferner zahlreiche Sonderzeichen aus Unicode über Entity-Referenzen verfügbar:

```
<p>Au&szlig;erdem m&ouml;chte ich betonen, dass… </p>
```

ß ö

Von HTML zu XHTML

- Wer HTML 4.01 gewöhnt ist, beachte bei Umstellung auf XHTML 1.1:
  - Alle Elemente und Attribute klein schreiben: <html><body>…
  - Elemente stets schließen:

```
<p>Text Text</p>  <p>Mehr Text …</p>
<p>Zeilenumbruch<br/>Neue Zeile…</p>
```

  - Immer Attributwerte angeben

```
<td nowrap="nowrap"> … </td>
```

  - XML-Deklaration und Dokumententyp-Deklaration stets angeben
  - Namensraum definieren: **<html xmlns="…">**
  - Sprachkennzeichen: Nur noch mit globalem Attribut **xml:lang**
  - Anker nicht mehr mit **name**, sondern mit **id** setzen (analog: map):

```
<h2>Ein Sprungziel<a id="ziel123"/></h2>
```

  - Zeichensatz in XML-Deklaration festlegen statt in „head“
  - CSS-Datei per Stylesheet-PI einbinden statt in „head“, CSS nutzen!
  - Basis ist „strict“ - „transitional“ und „frames“-Elemente entfallen!

Grundlagen:
Dynamische Webseiten-Generierung

CGI
SSI
HTML-Templates
Client-seitig:JavaScript
Embedded Scripting

Grundlagen:
Das Common Gateway Interface

CGI-Programmierung

CGI

- Motivation
  - Der Abruf statischer Dokumente genügt nicht:
    - Upload von Daten, Formularinhalten, Bildern, ...
    - Anbindung an Inhalte von Datenbanken
    - "WUI" für Anwendungen, etc.
  - Der Weg:
    - Der Web-Server bildet bestimmte URLs auf Programme statt auf Dokumente ab.
    - Die Programme werden vom Web-Server gestartet und beendet.
    - Datenaustausch per Pipes (STDIN, STDOUT) und Umgebungsvariablen
    - Ergebnis: Das *Common Gateway Interface* (CGI), ca. 1994/1995
  - Die Folgen:
    - Einfachheit der Schnittstelle sorgte für große Verbreitung
    - Skriptsprachen (insb. Perl) wurden sehr beliebt.

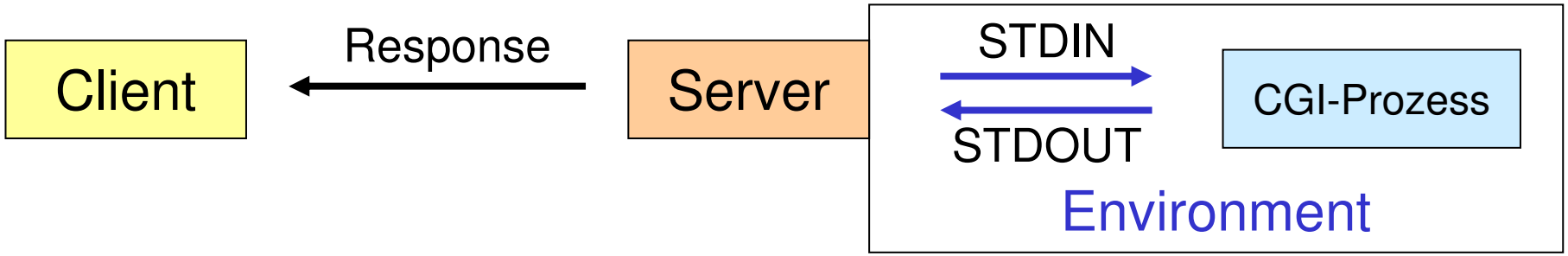
CGI

- Quellenangaben
  - http://hoohoo.ncsa.uiuc.edu/cgi/overview.html
    - Einstieg in die Beschreibung von CGI / CGI 1.1
  - http://hoohoo.ncsa.uiuc.edu/cgi/interface.html
    - Die Spezifikationen des CGI 1.1
  - http://www.w3.org/CGI/
    - Mehr zu CGI

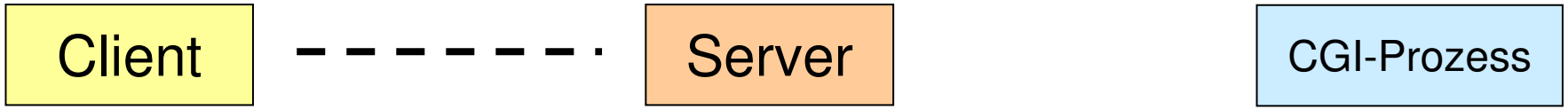
- Bemerkungen
  - CGI hat nie den Rang eines RFC oder einer W3C-Empfehlung erreicht.
    - Versuche in Richtung RFC / CGI 1.2 in den Jahren 1996...1999 wurden nicht zu Ende geführt.
  - Dennoch ist CGI wegen seiner Einfachheit auch heute (2006) noch weit verbreitet.

CGI: Der Ablauf

1) Server erkennt: URL für CGI
2) Server filtert Headerzeilen
3) Server setzt Umgebungsvariablen
4) Server startet CGI-Prozess (mit I/O über *pipes*)



5) CGI-Prozess liest Daten von STDIN (und aus Env.)
6) CGI-P. schreibt Header und Dokumentzeilen nach STDOUT
7) Server ergänzt ggf. Header
8) Server sendet HTTP *response* an Client, CGI-Prozess endet



CGI: I/O

- Bemerkungen zum I/O
  - Lesen von STDIN
    - Vorsicht - Prozess "hängt", wenn mehr Daten als vorhanden angefordert werden.
    - Daher: Header "Content-Length" beachten
    - Es gibt nicht immer einen "Body"
  - Schreiben nach STDOUT
    - Es gibt keine Garantie, dass der Client bereits vor Abschluss des Schreibens erste Daten erhält.
    - Trend: Server "reicht durch", also: Frühes Weiterleiten.
  - Was ist mit STDERR?
    - Der Standard lässt diese Frage leider offen.
    - Daher: Server-spezifische Antworten
    - Apache: STDERR-Texte enden in Datei **`error.log`**
    - Alternativen: Keine Unterscheidung zu STDOUT. *Race conditions*!

CGI: Environment

- Vorbemerkungen
  - Die Kommunikation per Umgebungsvariablen ist eine "Einbahnstraße". Sie kann prinzipiell nur vom Server zum CGI-Prozess funktionieren (Test: Warum?).
  - Die meisten Umgebungsvariablen des CGI leiten sich von HTTP-Headern ab.

- Gruppierung der Umgebungsvariablen
  1. Unspezifische Variablen - für alle *Requests*
  2. *Request*-abhängige Variablen / Standardvariablen
  3. Variablen für HTTP-Header von Clients, die der Server nicht als Standard-Header ansieht. Deren Namen beginnen mit "HTTP_".
     - Welche durchgelassen werden, entscheidet der Server
  4. Sonderfälle

CGI: Environment

- Unspezifische Variablen - für alle Requests
  - SERVER_SOFTWARE
    - Format: Name/Version
  - SERVER_NAME
    - Hostname, DNS Alias oder IP des Servers wie in selbst-referenzierenden URLs
  - GATEWAY_INTERFACE
    - Format: CGI/Revisionsnr.

CGI: Environment

- Requestabhängige Variablen
  - SERVER_PROTOCOL
  - SERVER_PORT
  - REQUEST_METHOD
    - GET, POST, HEAD
  - PATH_INFO
  - PATH_TRANSLATED
  - SCRIPT_NAME
  - QUERY_STRING
  - REMOTE_HOST
  - REMOTE_ADDR
  - AUTH_TYPE
  - REMOTE_USER
  - REMOTE_IDENT

CGI: Environment

- Requestabhängige Variablen
  - CONTENT_TYPE
  - CONTENT_LENGTH
  - DOCUMENT_ROOT *)

- Bem.: * = Nicht in Spez.

CGI: Environment

- Variablen für HTTP-Header von Clients
  - HTTP_ACCEPT
  - HTTP_ACCEPT_CHARSET
  - HTTP_ACCEPT_ENCODING
  - HTTP_ACCEPT_LANGUAGE
  - HTTP_COOKIE
  - HTTP_FROM
  - HTTP_HOST
  - HTTP_REFERER
  - HTTP_USER_AGENT
- Sonderfälle
  - HTTPS

CGI: Environment

Beispiel:

URL = http://calvin.informatik.fh-wiesbaden.de/cgi-bin/printenv?abcd=123?efgh

```
DOCUMENT_ROOT="/var/www"
GATEWAY_INTERFACE="CGI/1.1"
HTTP_ACCEPT="text/xml,application/xml,
   application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
   video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1"
HTTP_ACCEPT_CHARSET="ISO-8859-1,utf-8;q=0.7,*;q=0.7"
HTTP_ACCEPT_ENCODING="gzip,deflate"
HTTP_ACCEPT_LANGUAGE="en-us,en;q=0.5"
HTTP_CONNECTION="keep-alive"
HTTP_HOST="calvin.informatik.fh-wiesbaden.de"
HTTP_KEEP_ALIVE="300"
HTTP_USER_AGENT="Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US;
   rv:1.6) Gecko/20060308 Firefox/1.5.0.2"
```

CGI: Environment

```
PATH="/bin:/usr/bin:/sbin:/usr/sbin"
QUERY_STRING="abcd=123?efgh"
REMOTE_ADDR="80.128.xxx.yyy"
REMOTE_PORT="16660"
REQUEST_METHOD="GET"
REQUEST_URI="/cgi-bin/printenv?abcd=123?efgh"
SCRIPT_FILENAME="/usr/lib/cgi-bin/printenv"
SCRIPT_NAME="/cgi-bin/printenv"
SERVER_ADDR="195.72.96.19"
SERVER_ADMIN="webmaster@calvin.cs.fh-wiesbaden.de"
SERVER_NAME="calvin.informatik.fh-wiesbaden.de"
SERVER_PORT="80"
SERVER_PROTOCOL="HTTP/1.1"
SERVER_SIGNATURE="<ADDRESS>Apache/1.3.33 Server at
   calvin.informatik.fh-wiesbaden.de Port 80</ADDRESS>\n"
SERVER_SOFTWARE="Apache/1.3.33 (Unix) Debian GNU/Linux"
UNIQUE_ID="QIbXGsNIYBMAACYIJiY"
```

CGI: Folgen des Designs

- Konsequenzen aus diesem Schnittstellen-Design
  - CGI-Anwendungen können mit praktisch allen Programmiersprachen erstellt werden.
    - Keine Beschränkung etwa auf Skriptsprachen wie Perl!
  - CGI-Anwendungen können nur die Daten erhalten, die der Server passieren lässt.
    - Benutzer-Authentifizierung bleibt z.B. Sache des Servers.
  - Jeder Aufruf eines "CGI-URL" startet einen eigenen Prozess - Overhead beachten!
    - Bei Skriptsprachen kommt noch der Interpreter dazu!

CGI: Ausgabe

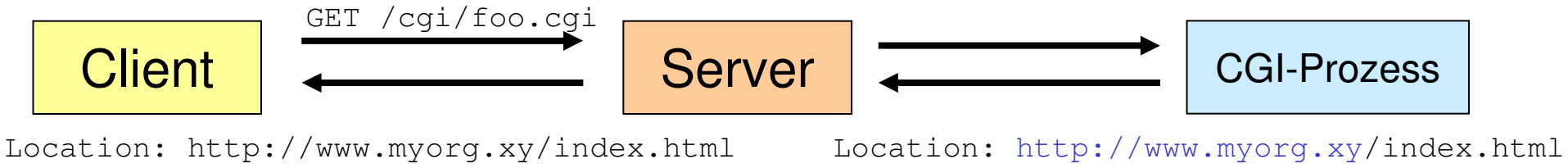
- Header
  - Jede CGI-Anwendung muss mindestens eine Headerzeile ausgeben.
    - **Content-Type** Medientyp der dann folgenden Daten
    - Location Im Fall einer Weiterleitung, s.u.
    - Status Steuerung von HTTP-Statuscodes

- Dokumentenausgabe
  - Perl-Bsp.: `print "Content-Type: text/html\n\n";`

- Weiterleitung
  - Externe W.: Der URL wird an den UA gesendet, dieser fordert neu an.
  - Interne W.: Die Umleitung wird direkt vom Web-Server ausgeführt

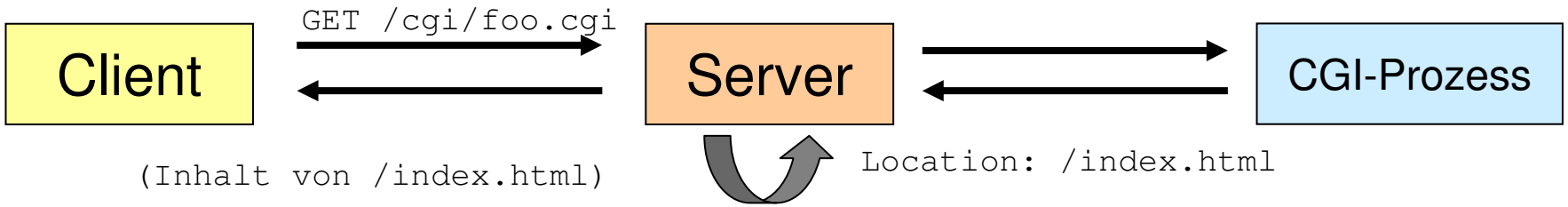
CGI: Ausgabe, Weiterleitung

Externe Weiterleitung:



Serveradresse:
http://www.myorg.xy
CGI-Prozess via:
foo.cgi

Interne Weiterleitung:



Internes "GET /index.html": Nur ein *request*, Weiterleitung unsichtbar (?)

CGI: Ausgabe

- Statuscodes
  - Sie können dem Web-Server mitteilen, einen anderen als den Standard-Statuscode in seine HTTP *response* zu schreiben, incl. Statustext.
  - Halten Sie sich dennoch möglichst an die Standardmeldungen - sonst versteht Sie der UA nicht. Präzisieren Sie vielmehr!
  - Ruby-Beispiel, ohne Dokument

```
puts "Status: 204 Keine Antwort\n"
```

  - Ruby-Beispiel, mit Dokument („*here document*“)

```
puts <<MY_EOT
Status: 503 Datenbank nicht verfuegbar
Content-Type: text/plain

Wartungsfenster der Datenbank: Heute 14-15 Uhr.
Bitte danach erneut versuchen.
MY_EOT
```

CGI: Ausgabe

- *Non-parsed-header* Anwendungen

  – CGI-Anwendungen können auch die vollständige Kontrolle über alle HTTP-Header übernehmen. Der Web-Server reicht die Daten dann einfach (und ohne Verzögerung) weiter.

  – Welche Anwendungen sich so verhalten, muss im Server konfiguriert werden. Gängige Konvention: "nph-" als Skriptnamenspräfix.

  – Beispiel: `GET /cgi-bin/nph-zaehler.cgi`
    - Server erkennt NPH-Betrieb am Skriptnamen

  – Bemerkungen
    - Früher ein häufig genutzter Weg, um Pufferung seitens der Web-Server zu umgehen und Ausgaben schnell an die Anwender zu leiten.
    - Seit Apache 1.3 weniger relevant, da dieser stets "früh" weiterleitet.

CGI: Sicherheitsfragen

- CGI-Anwendungen
  - laufen oft auf geschäftskritischen Servern
  - gestatten es anonymen Anwendern, Programme auf dem Server auszuführen
  - gestatten oft auch Eingaben von diesen Anwendern
  - Eingaben können das Programmverhalten beeinflussen (nicht nur Datenerfassung)
- Daher
  - Großes Schadenspotential
  - Strenge Kontrolle der Berechtigungen von CGI-Anwendungen erforderlich
  - Brennpunkte:
    - Compilersprachen: *Buffer overflow*-Attacken
    - Skriptsprachen: Vorsicht mit "eval". "taint"-Modus verwenden!
    - Alle: OS- und Shell-Aufrufe sehr sorgfältig prüfen!

CGI: Sicherheitsfragen

- CGI-Anwendungen auf Multihost-Servern
  - Normalerweise laufen CGI-Anwendungen mit denselben Rechten wie der Web-Server (Kind-Prozesse).
  - Auf Multihost-Servern muss dieses Konzept erweitert werden:
    - Wenn ein realer Web-Server mehreren Kunden dient, welche eigene CGIs einsetzen wollen, sollten diese sich nicht gegenseitig beeinflussen können.
  - Ausweg:
    - CGI-Prozesse laufen mit den Rechten der jeweiligen Kunden
    - Der Web-Server ändert also den Besitzer dieser Prozesse
  - Problem:
    - Dazu sind *root*-Rechte erforderlich!
    - Sicherheitslücken im Web-Server können daher zum Verlust des gesamten Servers (incl. aller Kundendaten) führen...
  - Fazit:
    - Web-Hosting incl. CGI erfordert spezielle Maßnahmen, Sorgfalt und einige Erfahrung! Vgl. etwa: c't 09/2004, S.168-171

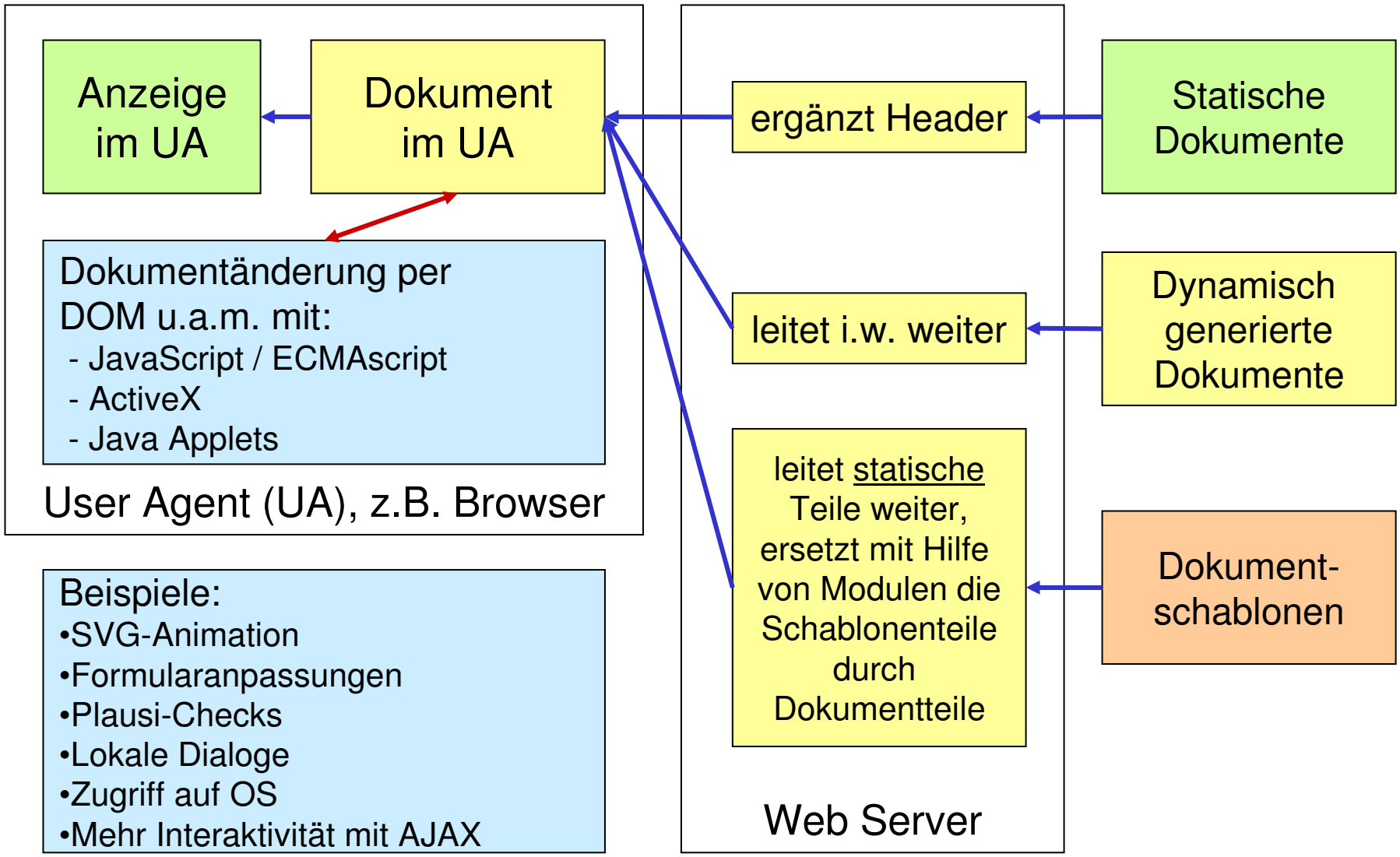
Grundlagen: Übersicht zur dynamischen Seitengenerierung

Client-seitige Techniken

HTML-Templates

Alternativen zu CGI

Dynamische Seitengenerierung: Übersicht



Dynamische Seitengenerierung

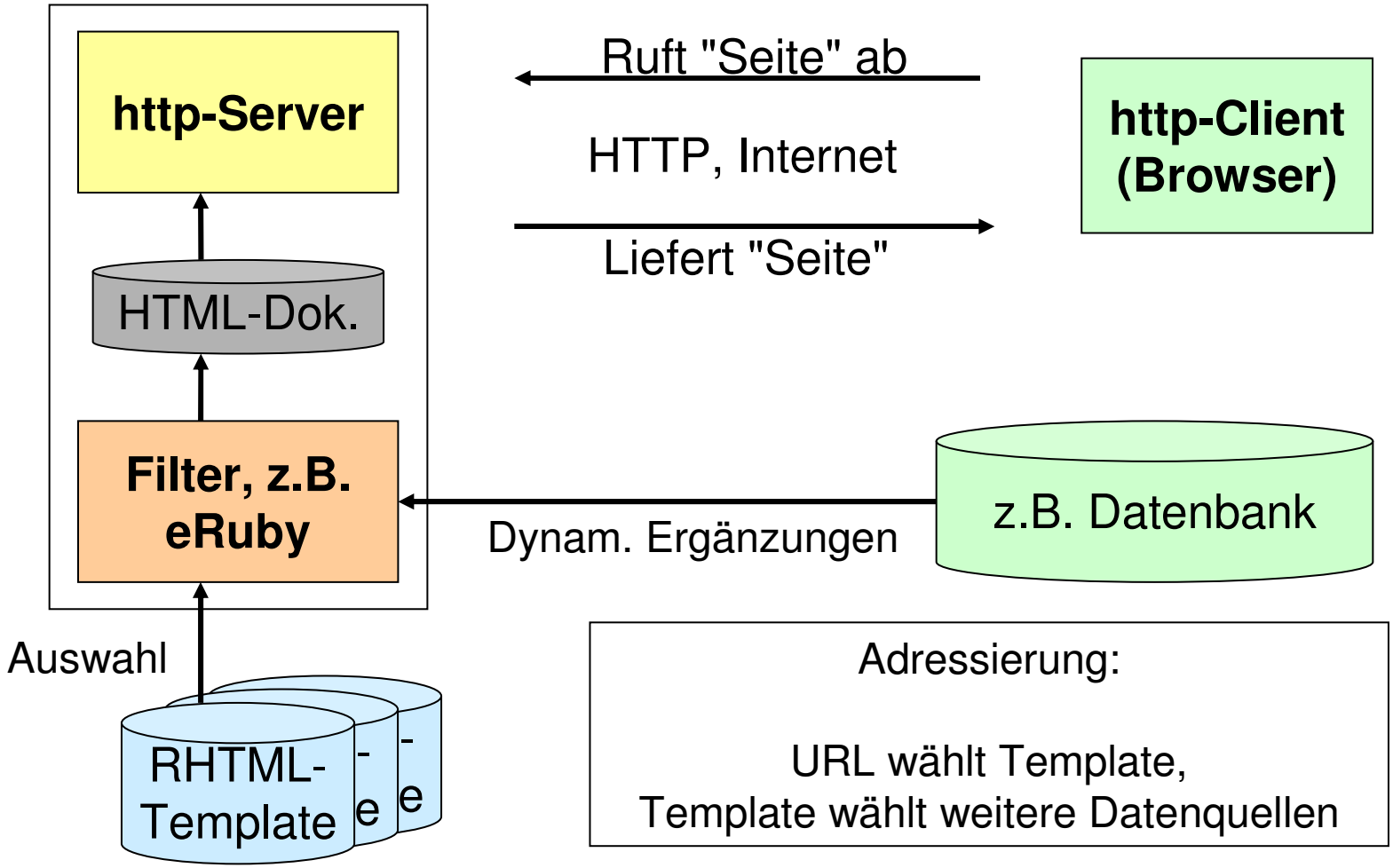
- Client-seitige Programmierung
  - Technologien:
    - JavaScript, Java Applets; ActiveX (nur MS Clients).
  - Anwendungsfälle:
    - Nur für Daten nützlich, die auf dem Client liegen.
  - Beispiele:
    - Lokale Prüfung von Formulareingaben
    - Optimierung der Formularanzeige in Abhängigkeit früherer Eingaben
    - Lokale Dialoge, etwa zur Entlastung des Servers
    - DOM: Modifikation des Dokumentenbaums direkt im Client, z.B. bestimmte SVG-Animationen, Ausblenden von Seitenteilen,...
    - Ausführung lokaler Programme zur Nutzung der Rechenleistung vieler Clients
  - Mini-Demo "alert" zu JavaScript, evtl. DOM-Demo aus LV „XML-Tech.“
    - Hinweis auf "Bookmarklets"
  - Fazit:
    - Komplementäre Technik zu serverseitigen Aktionen, z.Z. wieder aktuell (AJAX!)

Dynamische Seitengenerierung

- HTML *Templates* (Schablonen)
  - Motivation
    - Viele HTML-Seiten sind "fast" statisch - sie enthalten nur wenige Stellen, die Programmierung erfordern. Wozu dann alles per Programm erzeugen? Schablonentechnik fördert die Pflegbarkeit der Seiten.
    - Trennung von Layout und Inhalt: HTML-Code enthält (leider) viele Gestaltungsaspekte. Schablonen schaffen eine klare Schnittstelle zwischen Gestaltern und SW-Entwicklern.
  - Lösungen
    - Der Schablonengedanke ist inzwischen weit verbreitet.
    - Von einfachen Fällen (für "fast statische" Seiten) bis recht komplexen Anwendungen (und "fast dynamischen" Seiten) reichen die Lösungen.

Dynamische Seitengenerierung: *Templates*

Prinzip der Seitengestaltung mit Schablonen (hier: Ruby)



Dynamische Seitengenerierung

- HTML Templates (Schablonen)
  - Technik
    - Der Web-Server wird so konfiguriert, dass spezielle Schablonenseiten einer "Vorbehandlung" unterzogen werden.
    - Üblich: Erkennung über *extensions (*.php, *.shtml, *.jsp, *.rhtml...)*
    - Der Server lädt Module in seinen Adressraum (z.B. dynamische Bibliotheken, *.so bzw. *.dll) und überlässt deren Routinen die Vorbehandlung der Schablonen.
    - Deren Ausgabe (fertiges Dokument) wird dann an den UA gesendet.
  - Diskussion
    - Vorsicht bei Anwendermodulen im Adressraum des Web-Servers!
    - Portabilität beachten
    - Bei sehr komplexen Schablonen besser CGI & Co. einsetzen
    - Alternative zur getrennten Layoutentwicklung: CSS beachten.

Dynamische Seitengenerierung

- EmbPerl
  - Eine Perl-Lösung (von mehreren).
  - Beispiel für eine Tabelle (Fragment) der Umgebungsvariablen:

```
[$ foreach $varname ( sort keys %ENV ) $]
  <TR>
    <TD><B>[+ $varname +]</B></TD>
    <TD><B>[+ $ENV[$varname] +]</B></TD>
  </TR>
[$ endforeach $]
```

  - Ein "natürliches" Werkzeug für Perl-Entwickler

- eRb, eRuby
  - Eine ähnliche Lösung für Ruby (mehrere Implementierungen)
  - Öffnet den Ruby-Enthusiasten analoge Möglichkeiten

Dynamische Seitengenerierung mit Embedded Ruby

- Template-Technik mit **erb** oder **eruby** und DBI**:** booklist.rhtml

```
% require "dbi"
% dbh = DBI.connect("dbi:Mysql:rubydemo:localhost",
                     "werntges", "rubypw")
% sth = dbh.execute("SELECT title, author1, edition,
                     pubyear FROM morebooks")
<html><body><h1>Bücherliste</h1>
<p><table border="1">
<tr><th>Titel</th><th>Autor</th><th>Ed</th><th>Jahr</th></tr>
% sth.each do |row|
  <tr>
%   row.each do |field|
    <td><%= field.to_s%></td>
%   end
 </tr>
% end
</table></p>
</body></html>
% sth.finish
% dbh.disconnect if dbh
```

Dynamische Seitengenerierung

- PHP
  - Ursprünglich als einfaches Werkzeug zur Verarbeitung von HTML-Schablonen incl. Datenbank-Anbindung entstanden, entwickelt sich PHP schnell weiter, hin zu einer universellen Skriptsprache.
  - Inzwischen sind zahlreiche Erweiterungen für viele Anwendungsfälle erhältlich. Template-Beispiel, mit "PI"-Notation (damit XML-konform!):

```
<?php
  if ($expression) {
  ?>
    <em>Dies ist richtig.</em>
    <?php
  } else {
    ?>
    <em>Dies ist falsch!</em>
  <?php
}
?>
```

Dynamische Seitengenerierung

- Java Server Pages (JSP)
  - Zum Nachlesen: http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
  - Benötigte Umgebungen
    - Java RTE
    - Apache Tomcat
- Beispiel (aus o.g. Tutorial):

```
<H1>Welcome to Our Store</H1>
<SMALL>Welcome,
<!-- User name is "New User" for first-time visitors -->
<% out.println(Utils.getUserNameFromCookie(request)); %>
  To access your account settings, click
 <A HREF="Account-Settings.html">here.</A>
</SMALL>
```

- Vorteile:
  - Praktisch für Entwickler mit guten Java-Kenntnissen - kein Umlernen!
  - Portabel, relativ performant, mit allen Java-Möglichkeiten, ...

Dynamische Seitengenerierung

- Active Server Pages (ASP)
  - Zu JSP analoger Ansatz von Microsoft
  - Proprietär, nur auf MS-Systemen (dort aber häufig angewendet)
  - Praktisch für MS-Entwicker wegen der tiefen Integration in die MS-Produkte und wegen Weiterverwendung von MS-Sprachen.

- Weiterentwicklung: ASP.NET
  - Gute Unterstützung von Web Services, da MS diese aktiv fördert und weiterentwickelt.
  - Thema einer eigenen Lehrveranstaltung (→ LV 7337)

Dynamische Seitengenerierung

- **Server Side Includes (SSI)**
  - Für einfache Fälle, ursprünglich nur von Apache unterstützt, inzwischen verbreiteter.
  - In den Möglichkeiten begrenzt, dafür sehr einfach anzuwenden.
  - Konfiguration in Apache (Konvention: *.shtml parsen!):

```
<Location />
  ...
  Options     Includes
  AddHandler  server-parsed .shtml
  ...
</Location>
```

  - Einbettung: Per Konvention in speziellen HTML-Kommentaren. Syntax:

```
<!--#element attr="wert" attr="wert" ... -->
```

  - Bemerkung
    - Kommentare werden hier missbraucht. Besser: XMLs "PI" !

Dynamische Seitengenerierung

- Server Side Includes (SSI): Liste der SSI-Anweisungen

`echo`	***`var`***	Wert von Env.var., SSI var., oder custom var.
`include`	***`file`***	Fügt angegebene Datei hier ein
`include`	***`virtual`***	Wirkt wie ein HTTP *request*. Fügt URL-Dok ein.
`fsize`	***`file`***	Fügt Größe der angegebenen Datei ein
`fsize`	***`virtual`***	Fügt Größe der angegebenen Ressource ein
`flastmod`	***`file`***	Fügt Datum und Uhrzeit der letzten Änderung der angegebenen Datei ein
`exec`	***`cmd`***	Führt externes Kommando aus, fügt STDOUT ein
`exec`	***`cgi`***	Führt CGI-Script aus (keine Query-Strings möglich)
`printenv`		Gibt Liste der Env.-Variablen und ihrer Werte aus.
`set`	***`var`***	Umgebungsvariable setzen
`if, elif`	***`expr`***	Beginn bzw. Fortsetzung einer bedingten Anweisung
`else, endif`		Fortsetzung bzw. Ende einer bedingten Anweisung
`config`	***`errmsg, sizefmt, timefmt`***	Modifiziert SSI-Voreinstellungen

Dynamische Seitengenerierung

- Server Side Includes (SSI)
  - Für SSI vorhandene Umgebungsvariablen (ohne Erklärung)
    - DOCUMENT_NAME, DOCUMENT_URI
    - QUERY_STRING_UNESCAPED
    - DATE_LOCAL, DATE_GMT
    - LAST_MODIFIED
  - Beispiele:

```
Die aktuelle Zeit (GMT) ist: <!--#echo var="DATE_GMT"-->
<br>
<!--#config timefmt="%d.%m.%Y um %H:%M:%S h"-->
Dieses Dokument wurde
<!--#include virtual="/cgi-bin/zaehler.cgi"--> mal besucht.
  Es wurde zuletzt am
<!--#echo var="LAST_MODIFIED"--> geändert.
(Weiterer HTML-Code ...)
```

Dynamische Seitengenerierung

- **Alternativen zu CGI**
  - Nachteile des CGI
    - Hohe Prozessorlast: Jeder Aufruf startet einen eigenen (teuren) Prozess
    - Weiterer Overhead bei Skriptsprachen: Interpreter, Start & Parsen
    - Bei größeren Projekten: Parsen aller genutzter Bibliotheken, DB-Anmeldungen, andere Initialisierungen - immer wieder durchzuführen!
  - Auswege:
    - **Der FastCGI-Weg**:
      Die CGI-Anwendung sollte als autonomer Prozess einmal gestartet / initialisiert werden und dann auf Aufrufe warten.
    - **Der "mod_*xxx*"-Weg**:
      Bei Skriptsprachen wäre es schon eine große Hilfe, nicht immer den Interpreter neu laden zu müssen. Nur einmaliges Parsen wäre ebenfalls sehr vorteilhaft.
    - **Der Java-Weg**:
      Schaffung einer eigenen Laufzeitumgebung im Adressraum des Web-Servers. Ähnlich zum "mod_xxx"-Weg, aber noch weiter entwickelt.

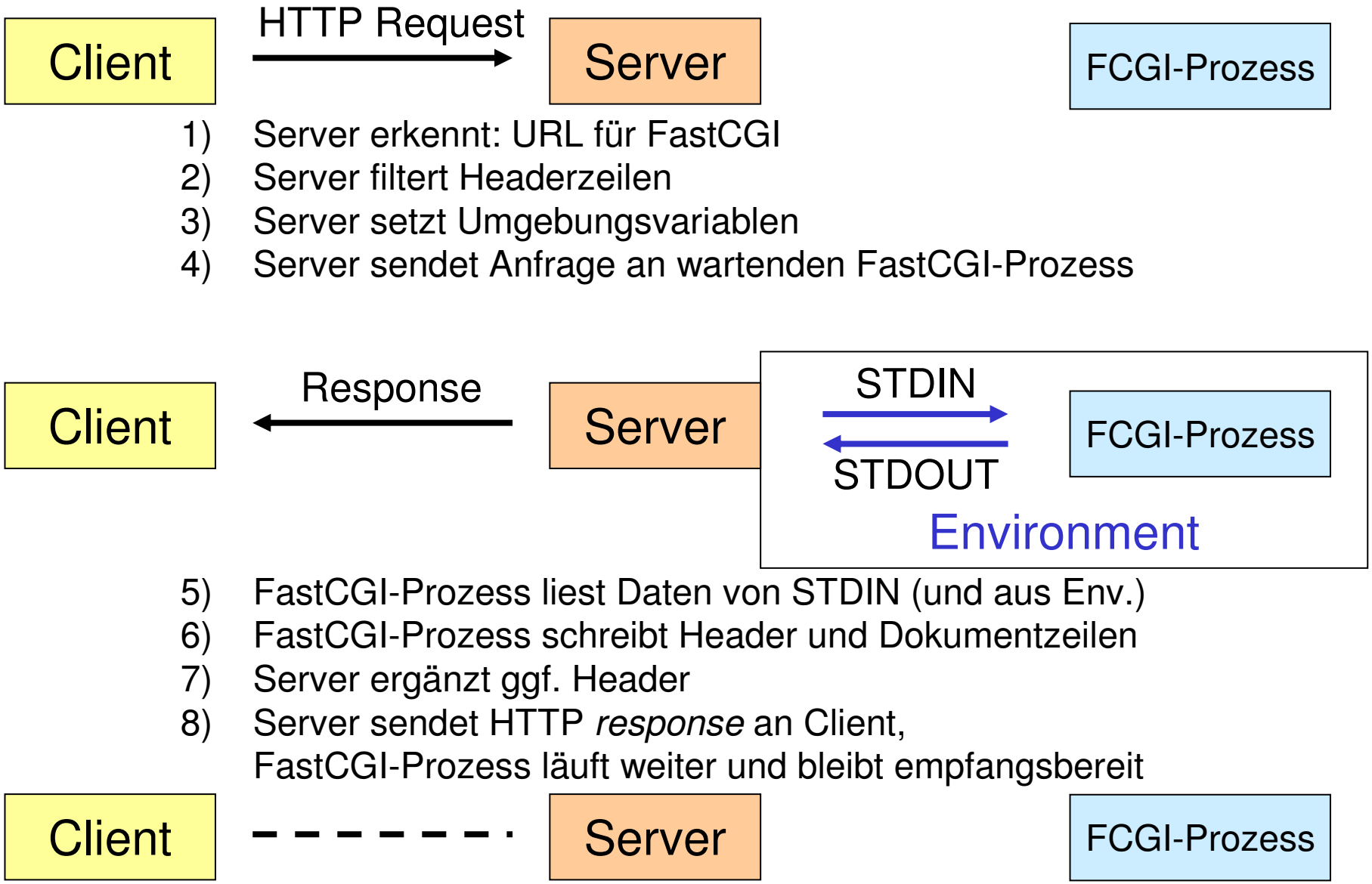
Dynamische Seitengenerierung

- mod_perl, mod_ruby, mod_*xxx*
  - Erweiterungsmodule für Web-Server wie den Apache
  - Der jeweilige Interpreter wird im Adressraum des Apache-Prozesses verfügbar - und braucht daher nicht bei jedem CGI-Aufruf erneut geladen zu werden.
  - NEU: Zugang zum Apache API. Damit ist die Entwicklung ganzer Apache-Module möglich.
  - Apache::Registry - eine Emulation der CGI-Umgebung in mod_perl. Laden von Perl-Modulen sowie der benötigten CGI-Skripte beim Serverstart ebenfalls möglich.
  - mod_ruby: Analoge Möglichkeiten für diese Skriptsprache, etc.
- Vorsicht!
  - Die CGI-Emulation ist prinzipbedingt nicht perfekt (Diskussion). Subtile Unterschiede können schwer zu findende Fehler hervorrufen.
  - Langlebige Prozesse müssen viel besser programmiert sein als "Einweg-Ware". Sie werden Ihre Skripte anpassen müssen!

Dynamische Seitengenerierung

- **FastCGI bzw. FCGI**
  - Standard-Lösung, für beliebige Sprachen und Web-Server, daher sehr portable Anwendungen.
  - Kernidee:
    - FCGI-Anwendungen sind eigene Server-Prozesse, die der Web-Server als FCGI-Client nutzt.
    - Sie werden einmal gestartet & initialisiert und bleiben dann über viele Aufrufe im Speicher.
  - IPC:
    - Lokal über Pipes
    - Remote (ja, auch dies ist möglich!) über IP-Sockets
  - Drei Varianten
    - Statische, dynamische und externe Anwendungen

FastCGI: Der Ablauf



Dynamische Seitengenerierung

- Die drei Betriebsarten von FastCGI-Anwendungen in Apache
  - FCGI-Anwendungen werden vom FCGI-Prozessmanager "fcgi-pm" verwaltet
  - [Tafelbild] Es gibt drei Arten, solche Prozesse zu verwalten:

- **Statische Anwendungen**
  - Diese werden genau einmal gestartet, zusammen mit dem Apache-Server
  - Sollten sie abstürzen, startet sie der fcgi-pm erneut
  - Zuständige Direktive: **`FastCgiServer`**

- **Dynamische Anwendungen**
  - Diese werden erst bei Bedarf gestartet
  - Bei hoher Last werden mehrere Prozesse parallel gestartet (!)
  - Bei zurückgehender Last werden Prozesse auch wieder beendet
  - Zuständige Direktive: **`FastCgiConfig`** (**default**!)

- **Externe Anwendungen**
  - Diese werden nicht von fcgi-pm verwaltet. Typisch für *remote*-Anwendungen!
  - Zuständige Direktive: **`FastCgiExternalServer`**

Dynamische Seitengenerierung

- FastCGI: Designaspekte
  - Langlebige, speicherresidente Serverprozesse erfordern mehr Sorgfalt als kurzlebige CGI-Anwendungen. Anregungen:
    - Achten Sie auf Seiteneffekte früherer Anfragen!
      Vermeiden Sie insbesondere globale Variablen, wo lokale reichen.
    - Fehlerbehandlung sollte nicht mit Prozessterminierung enden.
    - "*memory leaks*" können ein ernstes Problem werden.
    - Es sind mehrere gleichzeitig laufende Prozesse Ihrer FCGI-Serveranwendung möglich. Ihre Programmlogik muss das beachten.
- NEU:
  - FCGI-Anwendungen können auch auf anderen Rechnern als dem des Web-Servers laufen.
    - Vorteile: Gut zur Lastverteilung, effizientere (lokale) Zugriffe möglich, ...
    - Nachteile: IP-Protokoll kann Sicherheitsfragen aufwerfen oder an Firewalls scheitern, eigene Prozessüberwachung notwendig, ...

Dynamische Seitengenerierung

- Java Servlets
  - Java Servlets sind einerseits funktionell ähnlich zu CGI-Anwendungen: Sie erzeugen etwa HTML-Code als Ausgabe, also komplett dynamisch.
  - Andererseits laufen sie nicht als eigenständige Prozesse, sondern als Threads in einer Java-Laufzeitumgebung unter Kontrolle des Web-Servers.
  - Damit bieten sie ähnliche Vorteile wie die "mod_xxx"-Lösungen oder auch FastCGI.
  - Die Servlet-Laufzeitumgebung des Apache ist Tomcat, ebenfalls ein Projekt der Apache Foundation (www.apache.org).
    - Wer seine Projekte mit Java bearbeiten will, sollte also seine Apache-Installation um Tomcat erweitern bzw. durch Tomcat ersetzen!
  - Java Servlets bieten sich als Lösung für versierte Java-Entwickler an, da diese die Möglichkeiten von Java übernehmen können und für "CGI & Co." keine andere Sprache lernen müssen.

Dynamische Seitengenerierung

- WEBrick als Ruby Servlet-Container à la Apache Tomcat und Java Servlets

```
#!/usr/bin/ruby
require 'webrick'; include WEBrick

s = HTTPServer.new( :Port => 2000 )

class HelloServlet < HTTPServlet::AbstractServlet
  def do_GET(req, res)
    res['Content-Type'] = "text/html"
    res.body = %{
      <html><body>
           <h2>Hello</h2>
           <p>You're calling from a #{req['User-Agent']}</p>
           <p>I see parameters: #{req.query.keys.join(', ')}</p>
       </body></html>
    }
  end
end
s.mount("/hello", HelloServlet)

trap("INT"){ s.shutdown }
s.start
```

Demo! URLs sind:
localhost:2000/hello,
localhost:2000/hello?a=1&bc=23