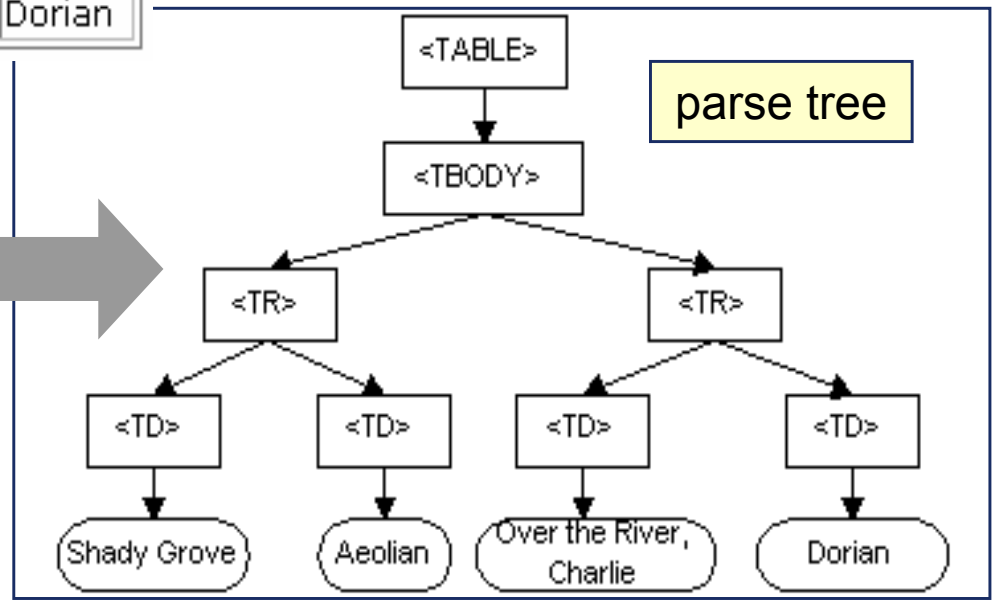
HTML als Baumstruktur

- ergibt sich zwanglos aus der Schachtelung von Tags

```
<TABLE border=1>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
    </TR>
    <TR>
      <TD>Over the River, Charlie</TD>
      <TD>Dorian</TD>
    </TR>
  </TBODY>
</TABLE>
```

Shady Grove	Aeolian
Over the River, Charlie	Dorian



Attribute und Texte sind eigenen Unterknoten !

(aus W3C: Document Object Model Level 2 Core Specification)

Grundproblem / Grundidee DOM

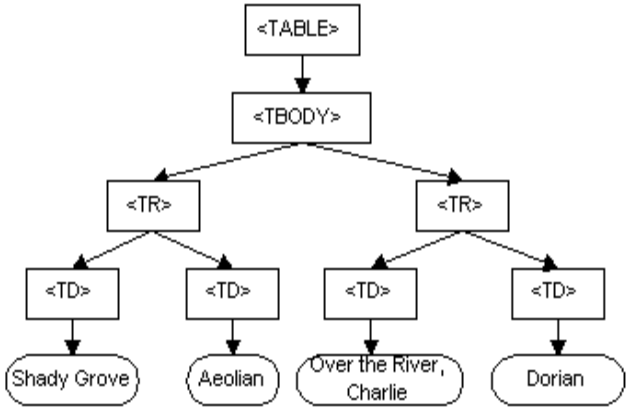
Browser

Shady Grove	Aeolian
Over the River, Charlie	Dorian

sichtbar

```
<TABLE border=1>
  <TBODY>
    <TR>
      <TD>Shady Grove</TD>
      <TD>Aeolian</TD>
```

public



Browser-intern; spezif. Zugriffe

- mache den "internen Baum" durch Standard-schnittstellen allgemein zugänglich
- Der Browser muss die Abbildung der Schnittstellen auf die interne Datenstruktur implementieren

  ⇨ " DOM"

Sinn und Zweck des DOM

- API für XML Dokumente, spezialisiert für HTML
  - ⇨ Programmierschnittstelle, die Zugriffsmöglichkeiten auf HTML Seiten definiert
  - ⇨ Anwendungen: Animationen mit Dynamic HTML, Autorenwerkzeuge
- Ausgangspunkt: Skriptsprachen browserunabhängig
  - ⇨ soweit diese auf das HTML Dokument zugreifen
- Status (2006): DOM Level 3 als W3C Empfehlung (Apr. 2004)
  - ⇨ Vorläufer: browserspezifisches Dynamic HTML (bei Netscape ursprünglich nur Zugriff auf bestimmte Elemente)
- Perspektive: vgl. VBA-Anwendungen in MS Office

Sprachunabhängige Definition

- 1. Schritt: Definition von Interfaces
  - ⇨ Interface ist öffentlicher Teil einer Klassendeklaration
    - d.h. ohne private Elemente und ohne Methodenkörper
  - ⇨ enthält keine Information zur Implementierung
  - ⇨ sprachunabhängig durch Interface Definition Language
    - IDL von OMG (ursprünglich für CORBA)

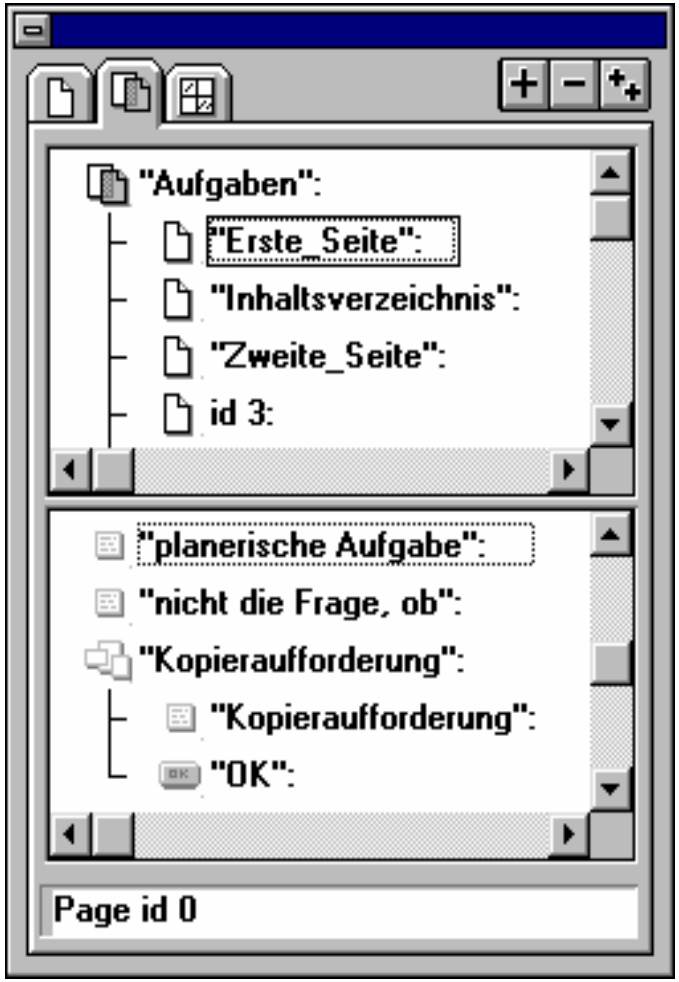
vgl. Java

- 2. Schritt: Abbildung auf Programmiersprachen
  - ⇨ Implementierung der Interfaces in echter Programmiersprache
  - ⇨ bisher Java und JavaScript (ECMAScript)

"language binding"

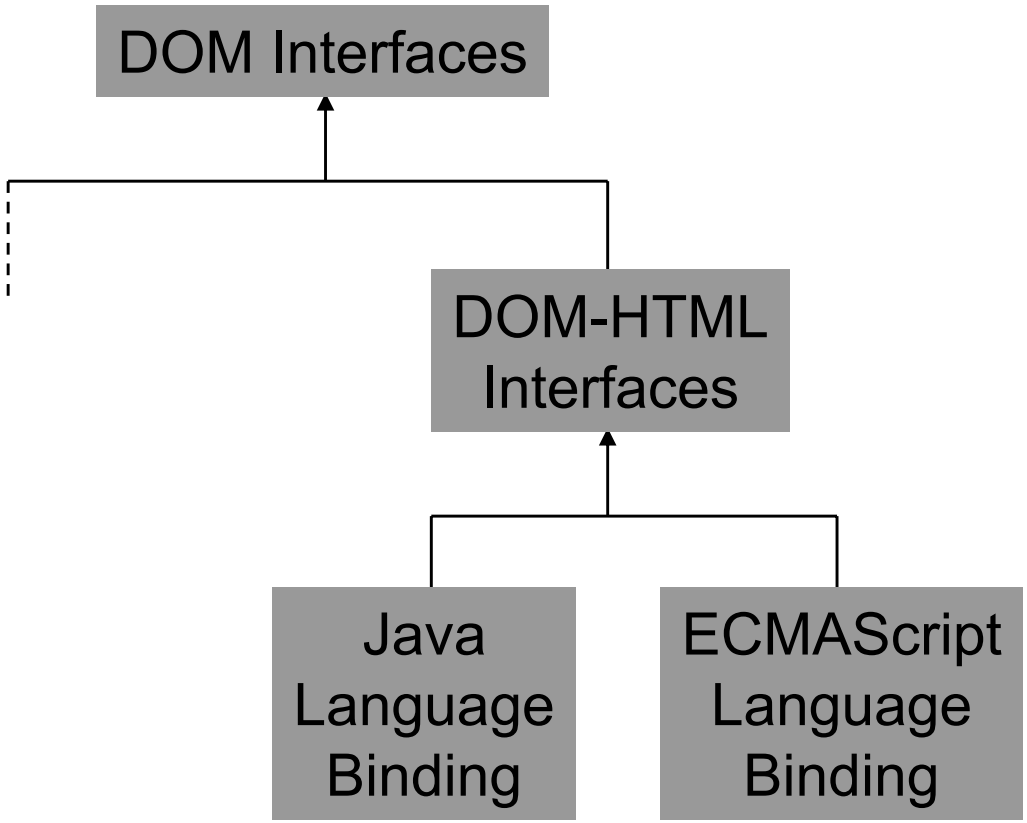
DOMs in Autorensystemen

- proprietäre DOMs
  - ⇨ Attribute und Methoden sind feste Bestandteile
  - ⇨ ein Großteil des Lernaufwands steckt im jeweiligen DOM
- siehe z.B. Director (Lingo), ToolBook
  - ⇨ Director: 2 Objektarten in 2 Tabellen
    - 2-dim. Tabelle von Sprites; zeigen in 1-dim. Tabelle von Cast Members
  - ⇨ ToolBook: baumartige Obj.hierarchie
    - Bild ⇦ Gruppe ⇦ Seite ⇦ Hintergrund ⇦ Buch



Object Browser in ToolBook II

Hierarchie der Standardisierungsdokumente



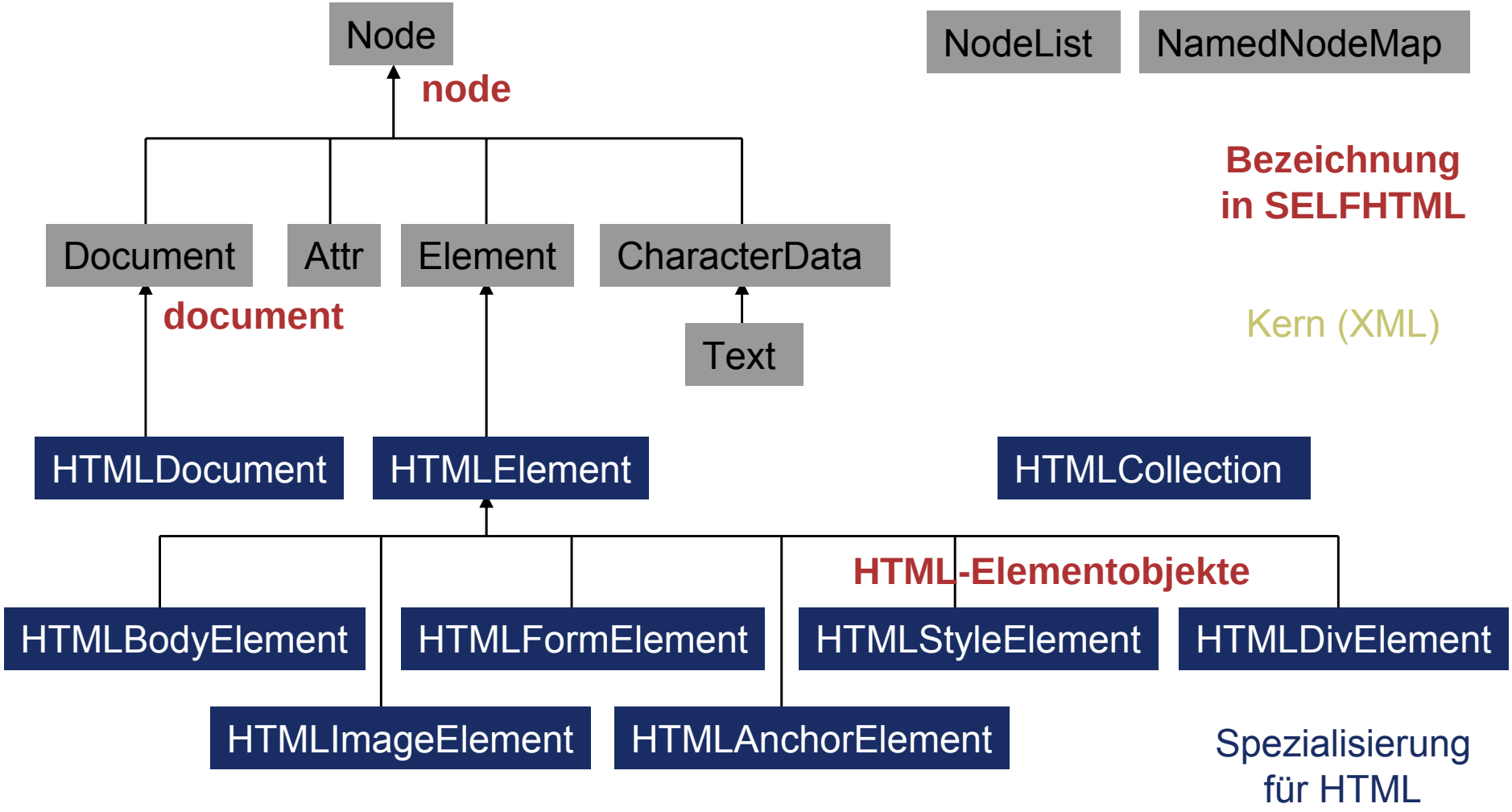
Kern (XML)

Spezialisierung für HTML

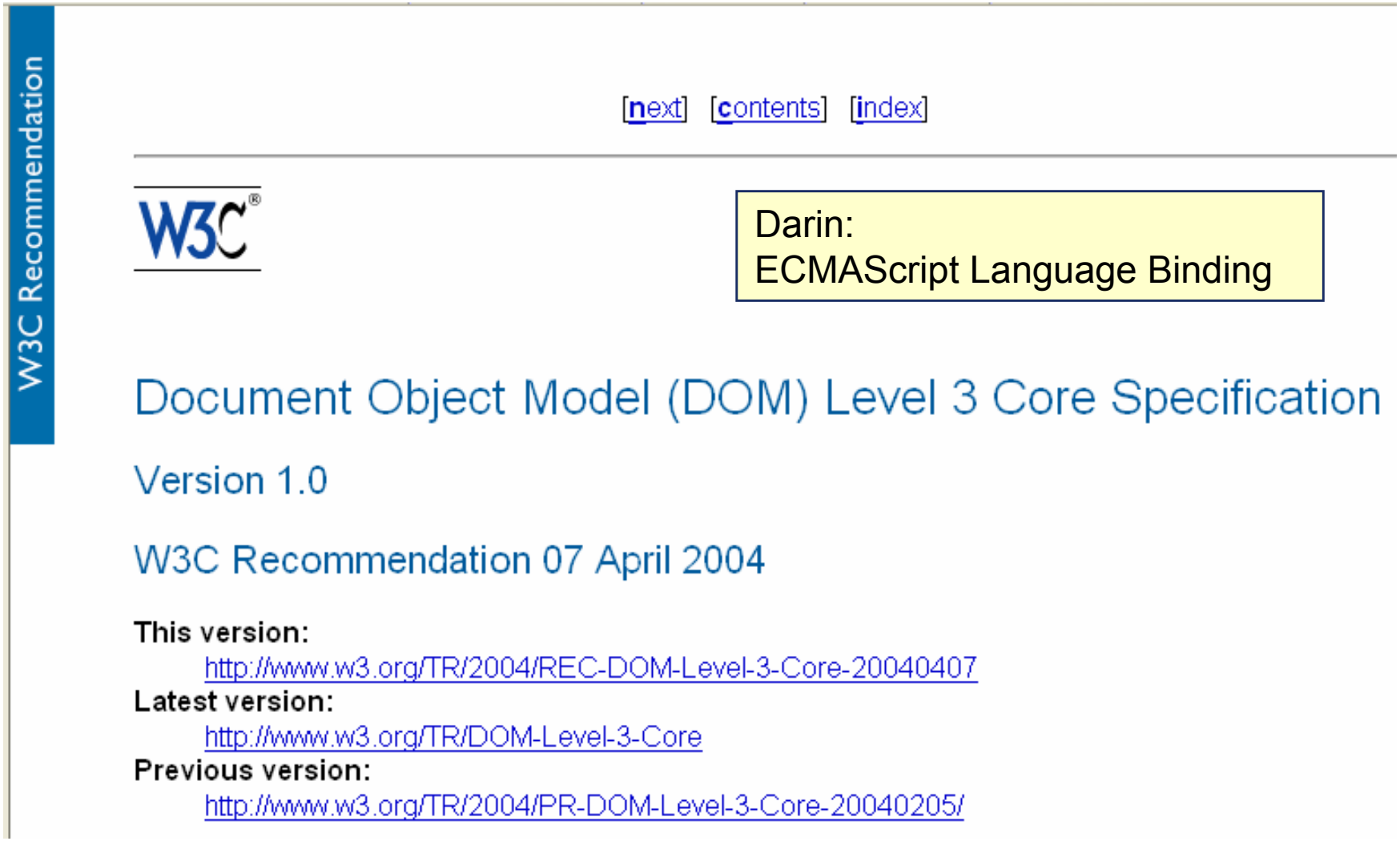
Konkretisierung für Programmiersprachen

vgl. Analysemodell vs. Designmodell in SWT

Hierarchie der Interfaces und Klassen



Auszüge DOM-Standard



W3C Recommendation

[next] [contents] [index]

W3C

Darin:
ECMAScript Language Binding

Document Object Model (DOM) Level 3 Core Specification

Version 1.0

W3C Recommendation 07 April 2004

This version:
http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407
Latest version:
http://www.w3.org/TR/DOM-Level-3-Core
Previous version:
http://www.w3.org/TR/2004/PR-DOM-Level-3-Core-20040205/

Zugriff auf Knoten über Kern-Klassen

DOM3 Core

- Ausgangspunkt ist `document` oder ein Element-Knoten
- Direkter Zugriff auf eindeutiges Element per `id`

```
Knoten = document.getElementById("xyz")
```

  - ⇨ jedes benötigte HTML-Element mit eindeutiger `id` bezeichnen
- Zugriff auf Elemente aus Collection mit demselben Tag

```
Knoten = document.getElementsByTagName("h3")[2]
```

- Zugriff auf Elemente aus Collection mit demselben `name`

```
Knoten = document.getElementsByName("abc")[0]
```

  - ⇨ nicht jedes Tag darf `name` haben
  - ⇨ evtl. mehrere Elemente mit demselben `name` (vgl. Radiobuttons)

nur für document

- *alle Varianten liefern einen HTML-Element-Knoten ab*

Zugriff auf Knoten über HTML-Collections

- die Klassen HTMLxxx haben die althergebrachten Collections
  - ⇨ HTMLDocument: images, applets, links, forms, anchors
  - ⇨ HTMLFormElement: elements
- Anwendungsbeispiel

```
MeinFormular = document.forms["Anmeldung"];
MeinFormular.elements["Eingabe"].value = 0;
```

- aus diesen Collections kann per id oder per name ausgewählt werden
  - ⇨ `"Anmeldung"` und `"Eingabe"` können in HTML als id oder als name eingetragen sein
    - id hat Vorrang
    - name ist nicht bei allen Tags zulässig

name-Attribut versus id-Attribut

- `name` ist nur bei *manchen* Tags zulässig
  - ⇨ muss nicht eindeutig sein
    - bei Radiobuttons: Gruppenbildung über denselben Namen
  - ⇨ wird für verschiedene Zwecke eingesetzt
    - `<a>` Sprungmarke
    - `<input>` Parametername für die Datenübertragung
- `id` ist bei *allen* Tags zulässig
  - ⇨ muss dateiweit eindeutig sein
  - ⇨ ist gedacht für eindeutige Knotenadressierung im DOM
- für Knotenadressierung `id` bevorzugen
  - ⇨ `name` ist dafür nur aus Kompatibilitätsgründen noch zulässig; in DOM2 Core nicht enthalten

Aber: Daten in Forms werden nur übertragen, wenn die Felder ein name-Attribut haben!

Weitere Möglichkeiten zum Zugriff auf Knoten

speziell beim *Aufruf* von Handlerfunktionen:

- `this` verweist auf das Element, in dem der Code steht

```
<div onclick="Verbergen(this);"> ... </div>
```

  - ⇨ nur gültig innerhalb eines HTML-Tags
  - ⇨ innerhalb einer Funktion, die zu einer Klasse gehört, verweist this auf das Objekt, auf das die Funktion gerade angewandt wird

veraltete Methode für *manche* Objekte:

nicht standard-konform

- wahlfreier Zugriff ohne Nennung der Collection unter Verwendung des *name*-Attributs

```
document.MeinFormular.Eingabe.value = "alt";
document.MeinBild.src = "bild.gif";
```

Baumoperationen über Kern-Klassen

DOM3 Core

- Die Klassen Node, Document und Element bieten Methoden zum Durchwandern und Manipulieren des Baums
- Erzeugung mit
  - `Document.createAttribute()` — Attributknoten
  - `Document.createElement()` — HTML-Elementknoten
  - `Document.createTextNode()` — Knoten für Textinhalt
- Eigenschaften — zum Durchwandern
  - `Node.attributes, Node.childNodes`
  - `Node.firstChild, Node.lastChild`
  - `Node.nextSibling, Node.previousSibling`
- Methoden — zur Strukturänderung
  - `Node.appendChild(...), Node.removeChild(...)`
  - `Element.setAttributeNode(...)`
  - `Element.removeAttribute(...)`

Zugriff auf Attribute

- alle Attributwerte in DOM2 HTML sind Strings
- Zugriff über Kern-Klassen (empfohlen) — DOM3 Core
  - ⇨ jedes Attribut ist in einem eigenen Unterknoten gespeichert
  - ⇨ Element.getAttribute und .setAttribute zum Zugriff auf bereits existierende Attributknoten (das sind alle HTML-Attribute)

```
var meinBild = document.images["BildId"];
meinBild.setAttribute("src") = "bild.gif";
```

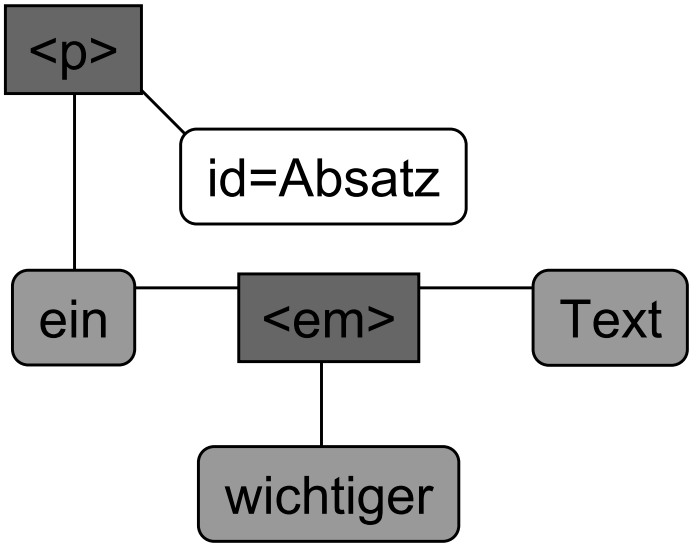
  - ⇨ (Attributknoten allokieren und in Baum einbauen nur für XML)
- Zugriff über HTML-Klassen (kompakt) — DOM2 HTML
  - ⇨ alle Klassen HTMLxxxElement haben ihre jeweiligen Attribute

```
var meinBild = document.images["BildId"];
meinBild.src = "bild.gif";
```

Sonderfall: `class` → `className`

Zugriff auf Text

- von einem Tag eingeklammerter Text ist in einem eigenen Unterknoten gespeichert



⇨ der Text steckt dort im Attribut `nodeValue`

⇨ Änderung durch Zuweisung, aber unformatiert (ohne HTML-Tags)

```
<p id="Absatz">ein <em>wichtiger</em> Text</p>

var Absatz = document.getElementById("Absatz");
 var ein = Absatz.firstChild.nodeValue;
 var em = Absatz.firstChild.nextSibling;
 var wichtiger = em.firstChild.nodeValue;
 var Text= Absatz.lastChild.nodeValue;
```

Achtung: Netscape erzeugt auch für reinen whitespace einen Textknoten

Zugriff auf Styles

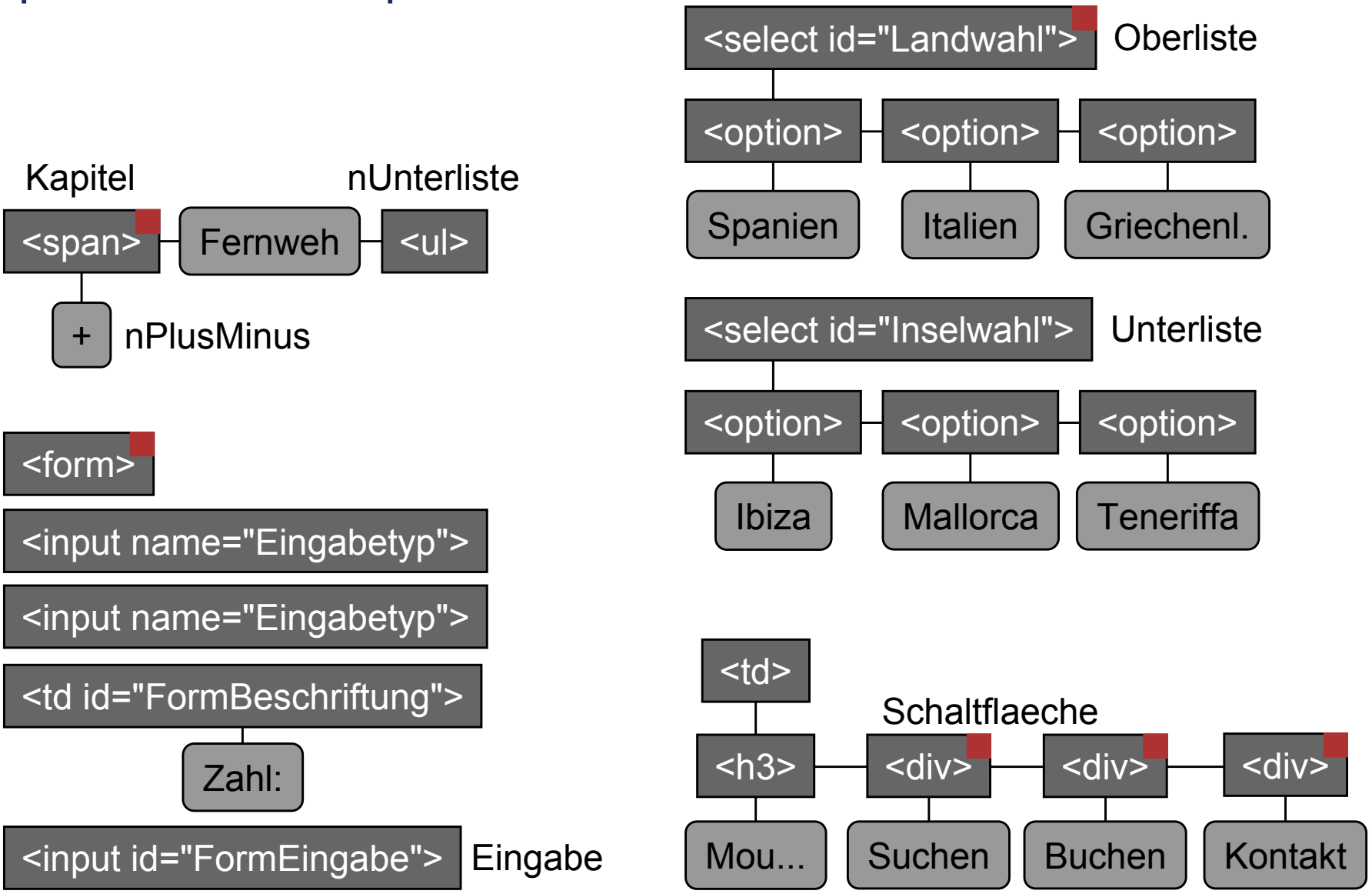
- auf inline-Styles des HTML-Elements, nicht auf CSS-Datei

```
<p id="Hugo"
     style="font-size:12pt; font-weight:bold">
```

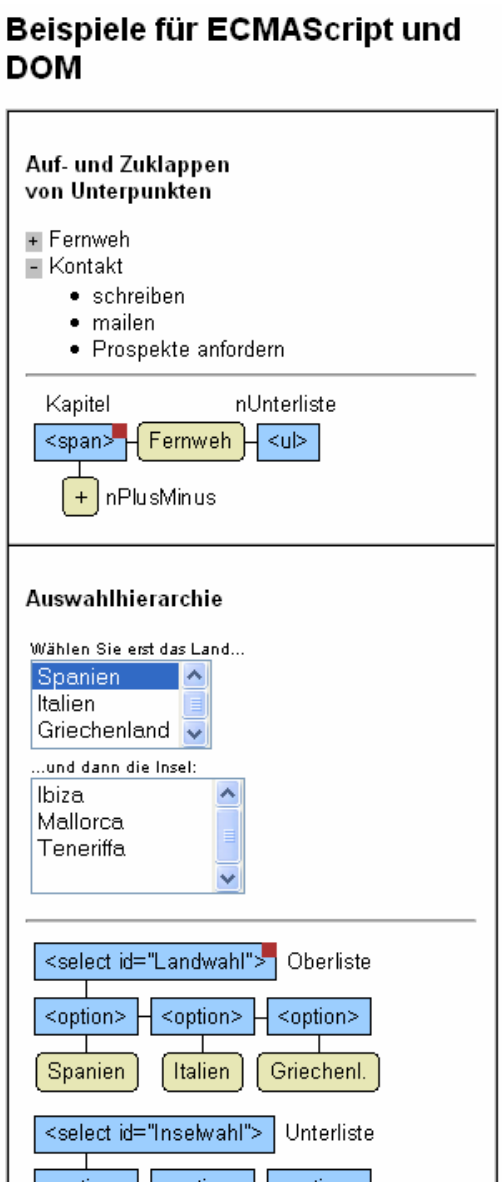
  - ⇨ haben Vorrang vor CSS-Datei, vgl. Kaskadierungsregeln
- Werte sind Strings
  - ⇨ Vorsicht bei Arithmetik; Strings enthalten px, %, pt, em
- Sonderregel für CSS Attributnamen im Skript
  - ⇨ Bindestriche sind nicht zulässig in Bezeichnern;
    deshalb Bindestrich weglassen und nächsten Buchstaben groß:
    `fontSize, fontWeight`
- Style ist als Unterobjekt realisiert

```
document.getElementById("Hugo").
                style.fontSize = "14pt";
```

Beispiele für ECMAScript und DOM



Beispiel ECMA-Script und DOM




```
// Auswahlhierarchie

                var Inseln_Spanien = new Array ("Ibiza", "Mallorca", "Teneriffa");
                var Inseln_Italien = new Array ("Elba", "Sardinien");
                var Inseln_Griechenland = new Array ("Korfu", "Kreta", "Rhodos", "Samos");

                function Vorauswahl(OberlisteID, UnterlisteID)
                {
                        var Oberliste = document.getElementById(OberlisteID);
                        var Unterliste = document.getElementById(UnterlisteID);
                        var Auswahl = Oberliste.options[Oberliste.selectedIndex].text;
                        var Inseln = eval ("Inseln_" + Auswahl);

                        while (Unterliste.firstChild != null)
                                Unterliste.removeChild (Unterliste.firstChild);
                        for (i=0; i<Inseln.length; i++) {
                                var neuesElement = document.createElement ("option");
                                var neuerText = document.createTextNode (Inseln[i]);
                                neuesElement.appendChild (neuerText);
                                Unterliste.appendChild (neuesElement);
                        }
                }


// Mouseover-Effekte

                function Mouseover (Schaltflaeche)
                {
                        Schaltflaeche.className = "ButtonOver";
                        // Sonderfall, weil class ein reserviertes Wort ist
                }
                function Mouseout (Schaltflaeche)
                {
                        Schaltflaeche.className = "ButtonNormal";
                }
                function Mousedown (Schaltflaeche)
                {
                        Schaltflaeche.className = "ButtonDown";
                }
                var bgStyle = null;
                var bgColor = null;
```

Beispiel: DOM-Zugriffe aus ECMA-Script

```
// UnterlisteID ist die ID meiner Select-Listbox
var Unterliste = document.getElementById(UnterlisteID);

// offline!! den neuen Teilbaum anlegen und initialisieren
var neuesElement = document.createElement ("option");
var neuerText = document.createTextNode ("Meine Option");

// den Textknoten an die Option anhängen
neuesElement.appendChild (neuerText);

// jetzt den neuen Teilbaum einhaengen
Unterliste.appendChild (neuesElement);
```

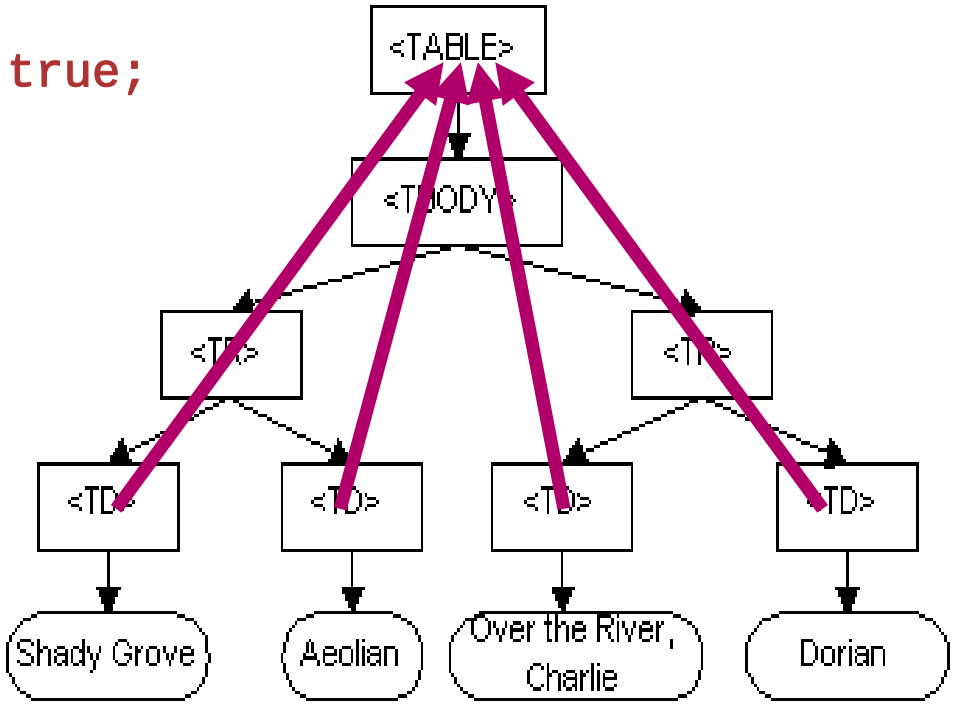
Event Bubbling

- Ereignisse werden in der DOM-Hierarchie weitergeleitet
  - ⇨ d.h. in der Schachtelungsstruktur von HTML nach außen
  - ⇨ an allen übergeordneten Elementen werden Handler aufgerufen
  - ⇨ Weiterleitung unterdrücken für das aktuelle Ereignis

    (nur MS Internet Explorer):

    ```
    window.event.cancelBubble = true;
    ```
- typisch für Ereignisorientierung
  - ⇨ vgl. ToolBook,

    Director, Windows



Werkzeuge

- Skripte: Große „Freiheit“ in der Entwicklung
  - ⇨ Irgendwie funktioniert es – oder auch nicht!
  - ⇨ Das DOM zu einer HTML-Seite ist unterschiedlich - je nach Browser
  - ⇨ Unterschiedliches Browser-Verhalten "knapp außerhalb" des Standards (z.B. xxx&euroxxx ohne ;) wird vom Firefox verstanden

⇨ Konzepte vorher überlegen (siehe Abschnitt Entwurf)

⇨ an den Standard halten

⇨ Tools verwenden
  - ⇨ Validatoren für HTML, CSS
  - ⇨ Skript-Debugger

⇨ Gründlich Testen

z.B. Browser-Plugins für Firefox verwenden: WebDeveloper, HTML-Validator, DOM Inspector...

Zusammenfassung

- ECMA-Script-Grundlagen
  - ⇨ Grundidee, Grundgerüst, HTML-Einbindung
  - ⇨ Standardisierte Schreibregeln und Syntax
  - ⇨ Objektbasierend
- Document Object Model (DOM)
  - ⇨ Grundidee, Sinn und Zweck, Standard
  - ⇨ Zugriffsmöglichkeiten auf Knoten, Attribute und Styles
  - ⇨ Baumoperationen
  - ⇨ Einbettung in ECMA-Script
  - ⇨ Werkzeugunterstützung

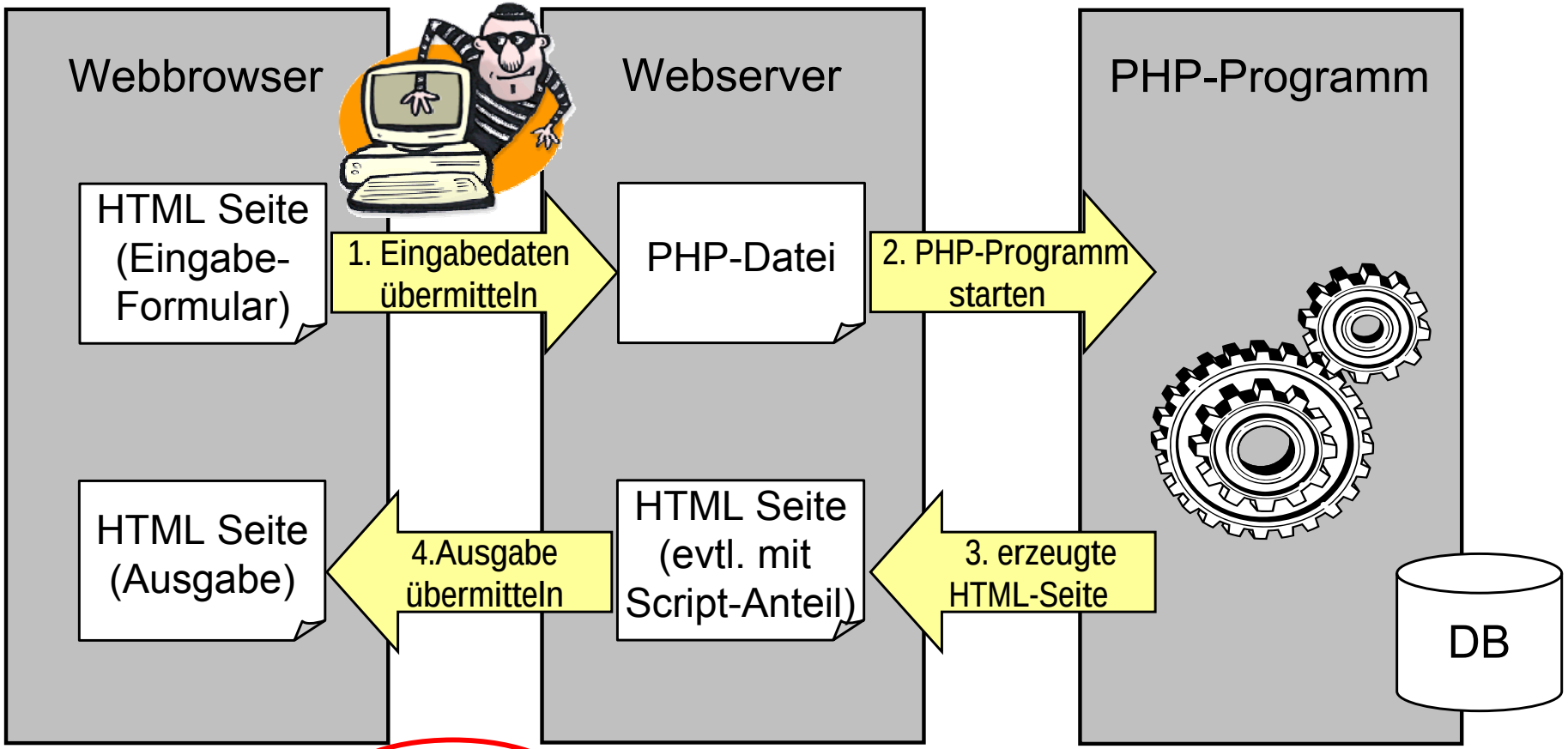
Jetzt wissen Sie alles um eine dynamisch änderbare HTML-Seite zu entwickeln, die Formulardaten überprüft!

Entwicklung webbasierter Anwendungen

6. Kapitel: HTTP

Quellenhinweis:
Viele Folien dieser Vorlesung entstammen der gleichnamigen Vorlesung von Prof. B. Kreling

Einsatz der Technologien im Zusammenhang



- HTML
- CSS
- ECMA-Script
- DOM
- Animation

- HTTP

- Server-Konfiguration

- CGI
- PHP
- MySQL

Begriffe: Client / Server, Pull / Push

- Server bietet Dienstleistung an
  - ⇨ Dienste sind z.B. WWW, FTP, Mail
  - ⇨ nimmt Anfragen entgegen, führt Anweisungen aus, liefert Daten
- Client nimmt Dienstleistung in Anspruch
  - ⇨ initiiert dazu eine Verbindung mit dem Server
  - ⇨ meistens "dichter am Benutzer"
- Pull — dafür ist HTTP gedacht
  - ⇨ Aktivität geht vom Client aus, Server reagiert nur
  - ⇨ "klassischer" Stil der Kommunikation im Internet
- Push — hat sich im Internet bisher nicht durchgesetzt
  - ⇨ Server übermittelt von sich aus (ungefragt) Daten
  - ⇨ Broadcast Systeme, Channels, Abonnements

HyperText Transfer Protocol (HTTP)

- einfaches Protokoll speziell für die Übertragung von Hypertext-Dokumenten über das Internet
  - ⇨ regelt die Kommunikation zwischen WWW-Client und -Server
- Request / Response Verfahren über eine TCP-Verbindung
  - ⇨ mehrere Nachrichten über dieselbe Verbindung
  - ⇨ einfacher Zugriffsschutz über IP-Adresse oder Passwort
- reines ASCII, Klartext, unverschlüsselt, zeilenorientiert
  - ⇨ Browser ⇨ Server:

    ```
    GETSPhttp://www.xyz.de/datei.htmlSPHTTP/1.1CRLF
    ```

  - ⇨ Server ⇨ Browser:

    ```
    HTTP/1.1SP200SPOKCRLF
    Content-type: text/htmlCRLFCRLF
    <!DOCTYPE...><html>...</html>
    ```

HTTP-Nachrichten

Details unter
http://www.w3.org/Protocols/rfc2616/rfc2616.html

Browser ⇨ Server	Server ⇨ Browser
Request-Line: Methode, URI, Version	**Response-Line:** Statusinformation
General-Header: (Cache-Control, Proxy-Info, Datum)	
Request-Header: Anforderungs- und Client-Information	**Response-Header:** Antwort- und Server-Information
Entity-Header: Information über Entity-Body (Komprimierung, Modifikationsdatum, Sprache, Länge)	
Entity-Body: Nutzinhalt (HTML-Datei)	

Request-Line (Methode, URI, Version)

Browser
⇨
Server

HTTP Request-Methoden

- Methoden sind die "Grundfunktionen" für Client-Requests
  - ⇨ in Nachrichten übermittelt, mit Zusatzinformationen ergänzt
- GET: Web-Seite lesen
  - ⇨ auch: wenige Daten an CGI-Prozess übermitteln, vorzugsweise für Abfrage von Daten
  - ⇨ HEAD: liefert dieselben HTTP-Header, aber ohne Web-Seite
- POST: Daten an CGI-Prozess übermitteln
  - ⇨ vorzugsweise für Speichern von Daten
  - ⇨ auch Datei-Upload an CGI-Prozess
- PUT: Web-Seite schreiben / ersetzen
- DELETE: Web-Seite löschen
- OPTIONS (Server-Fähigkeiten), TRACE (loop-back)

GETSPhttp://www.xyz.de/datei.htmlSPHTTP/1.1CRLF

Response-Line (Version, Status, Reason)

Server ⇨ Browser

Status für Menschen

HTTP Statusmeldungen

- Antwort vom Server in Response-Line:

  3-stellige Zahl (für Browser) und Klartext (für Benutzer)

- Information 100-101
- Erfolg 200-206
  - ⇨ Anfrage erfolgreich bearbeitet
- Umleitung 300-307
  - ⇨ der Client muss anderswo anfragen
- Fehler des Clienten 400-417
  - ⇨ nicht gefunden, Zugriffsverletzung, Authentifikation erforderlich
- Fehler des Servers 500-505
  - ⇨ interner Fehler, Service nicht verfügbar

HTTP/1.1SP**200**SP**OK**CRLF

HTTP Header (1)

- dienen dem Austausch von Hilfsinformationen zwischen Client und Server
- bestehen aus Namen und Wert, getrennt durch Doppelpunkt, abgeschlossen mit Zeilenende

```
Content-type: text/htmlCRLF
```

General Header

- Date
  - ⇨ liefert den Zeitpunkt der Anforderung oder Antwort
    z.B. Sun, 01 Apr 2000 08:05:37 GMT
- Pragma
  - ⇨ no-cache: Antwort nicht aus Cache, sondern vom Server holen

```
Pragma:no-cacheCRLF
```

HTTP Header (2)

Request Header

- If-Modified-Since
  - ⇨ bei GET: fordert nur ein aktualisiertes Dokument an

```
If-Modified-Since: Tue, 07 Apr 2004 23:24:25 GMT
```

- Referer
  - ⇨ von welchem Dokument wurde auf das angeforderte verwiesen ?

- User-Agent
  - ⇨ Informationen über den Browser (z.B. Mozilla/... für Netscape)

```
User-Agent: Mozilla/4.1
```

HTTP Header (3)

Entity Header

- Content-Type
  - ⇨ Typ des Nutzinhalts, z.B. Content-Type: text/html
- Content-Length
  - ⇨ Länge des Inhalts in Bytes
- Content-Encoding
  - ⇨ bei komprimierten Dokumenten, z.B. gzip
- Last-Modified
  - ⇨ letzte Änderung des übertragenen Inhalts für Caching

Response Header

Server
⇨
Browser

- Server
  - ⇨ Information über den Server z.B.: Apache/1.3.17 (Win32)
- WWW-Authenticate
  - ⇨ verlangt vom Clienten eine Authentifizierung

Beispiel

Browser ⇨ Server

(Request)

```
GET /index.html HTTP/1.1
Host: www.xyz.de
User-Agent: Mozilla/4.0
Accept: image/gif, image/jpeg, */*
Connection: Keep-Alive
```

Server ⇨ Browser

(Response)

```
HTTP/1.1 200 OK
Date: Thu, 15 Jul 2004 19:20:21 GMT
Server: Apache/1.3.5 (Unix)
Accept-Ranges: bytes
Content-length: 42
Connection: close
Content-type: text/html

<h1>Antwort</h1>
<p>Jetzt kommts</p>
```