JavaScript
und das
Document Object Model

Dynamische Seiten

- Bestreben von Netscape und Microsoft, HTML-Seiten „dynamisch“ zu gestalten
  - Schlagwort „DHTML“
- Netscape/Sun: JavaScript
- Microsoft: JScript
- Problem
  - Gewisse Inkompatibilitäten verhindern ein effizientes Gestalten der Seite: doppeltes Programmieren
  - Einsatz anderer Mechanismen (etwa neue Skriptsprachen) ermöglichen

JavaScript

- Allgemeine Hinweise zu Javascript
  - keine Klassen, nur Objekte (als Prototypen)
  - spezielle JavaScript-Objekte (von Netscape eingeführt) werden allmählich von DOM verdrängt
- Konstrukte
  - Variablen
    - lokal (mit vorangestelltem **var**), sonst global
    - schwach typisiert: numerisch (für Rechenoperationen) / nichtnumerisch (für Zeichen-orientierte Operationen); Konvertierungen

JavaScript

- Weitere Konstrukte
  - Blöcke
    - ähnlich wie in Java: { ... }
    - if-else, switch, ?:
    - while, for, do...while, break, continue (mit Marken)
  - Funktionen
    - mit und ohne Rückgabe (keine Angabe in Signatur)

```
function multipliziere(a,b) {
   return a*b;
}
```

Objekte

- Objekte
  - meist vordefinierte (aus der gegebenen HTML-Seite oder der Browserkonfiguration)
    - Beispiele: document, window, frames, forms, history
  - haben Eigenschaften (properties) und Methoden (Funktionen als Eigenschaften)

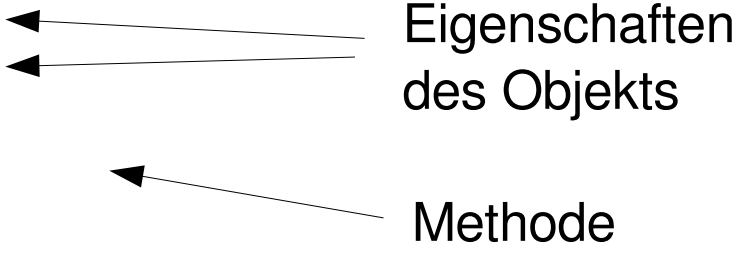
document.title	document.close()
document.fgColor	document.getElementById("dieseTabelle")
...	...

Meldung: „meinTestobjekt has no properties“ deutet darauf hin, dass es null (undefiniert) ist.
Keine *Exception*-Verarbeitung!

Objekte

- Erzeugen neuer Instanzen durch new
- Objekte selbst definierbar
  - Anlegen einer Funktion, deren Name auch das Objekt wird

```
function Komplex(real, imaginaer) {
  this.real=real;
  this.imaginaer=imaginaer;
  this.summe = function() {
    return this.real + this.imaginaer;
  }
}
...
var z = new Komplex(1,5);
document.writeln("z=(" + z.real + "," + z.imaginaer + ")");
document.writeln("Summe der Komponenten = " + z.summe());
```



Objekte

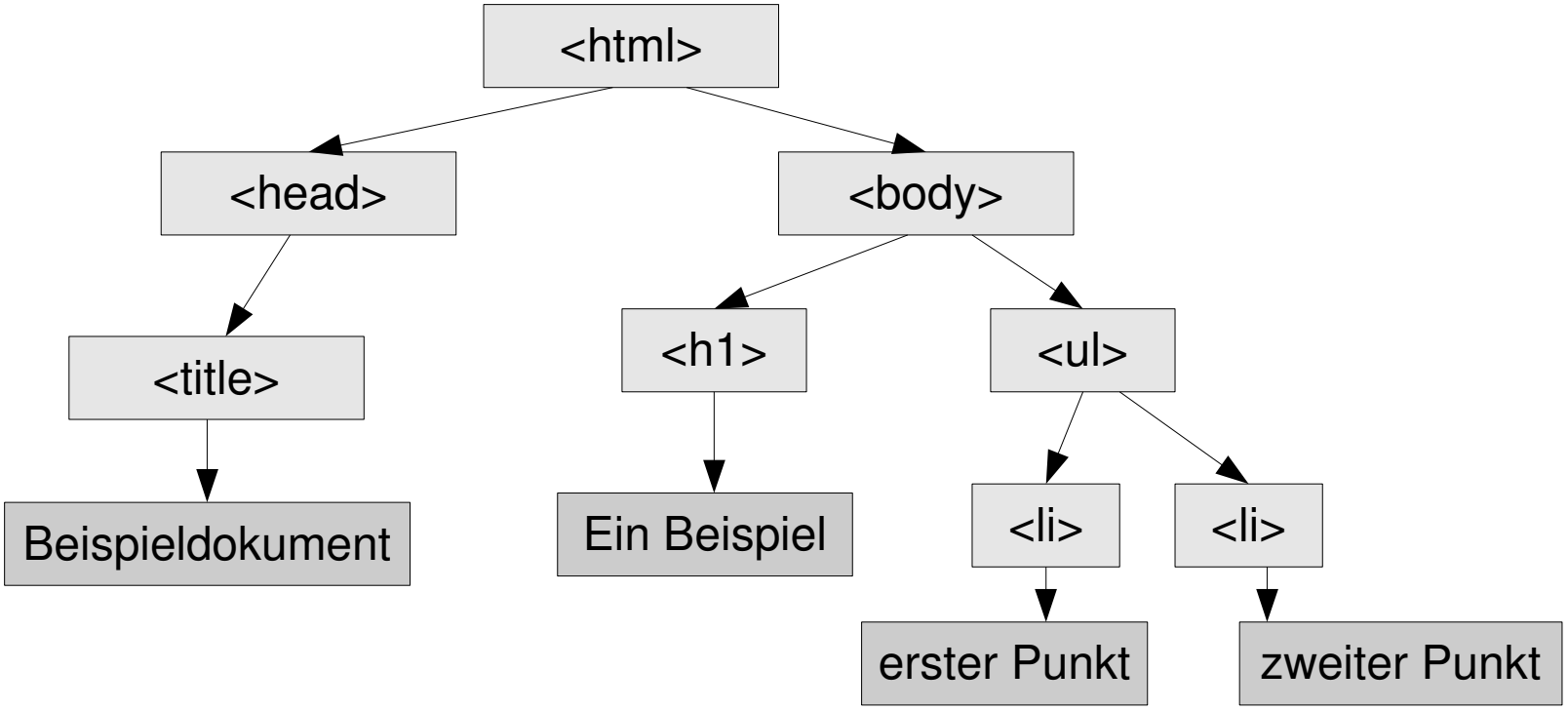
- Vererbung über *Prototyp*
  - Spezielle Eigenschaft *prototype;* ansprechbar über *constructor*

```
function Komplex() ...

function Quaternion(real, imag1, imag2, imag3) {
  this.constructor(real, imag1);
  this.imag2=imag2;
  this.imag3=imag3;
}
Quaternion.prototype = new Komplex();
...
var q = new Quaternion(1,3,6,10);
document.writeln("q=(" + q.real + "," + q.imaginaer + "," + q.imag2 + "," + q.imag3 + ")");
document.writeln("Summe von Real- und erstem Imaginärteil = " + q.summe());
```

Document Object Model

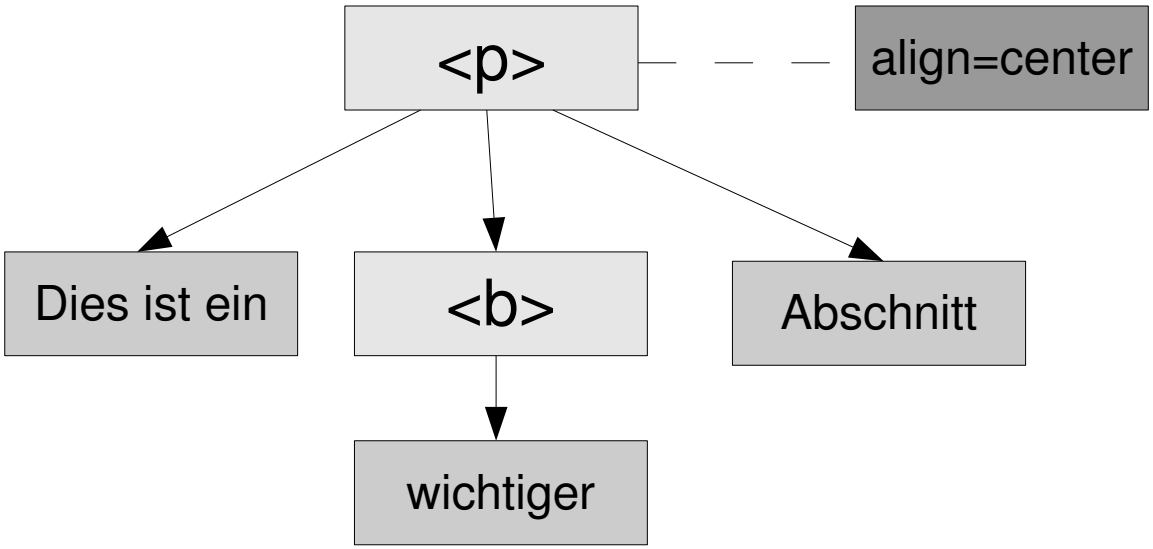
- Ansatz: programmiersprachenunabhängiges Modell eines Dokuments auf dem Web (d.h. einer Webseite)
  - gleichermaßen für XML-Dokumente geeignet
- Dokument als Baum



Elemente, Attribute, Texte

- Eingefasste Elemente werden als Kindknoten repräsentiert
- Texte stehen in Textknoten
- Attribute werden zu „assoziierten Knoten“

Dies ist ein wichtiger Abschnitt



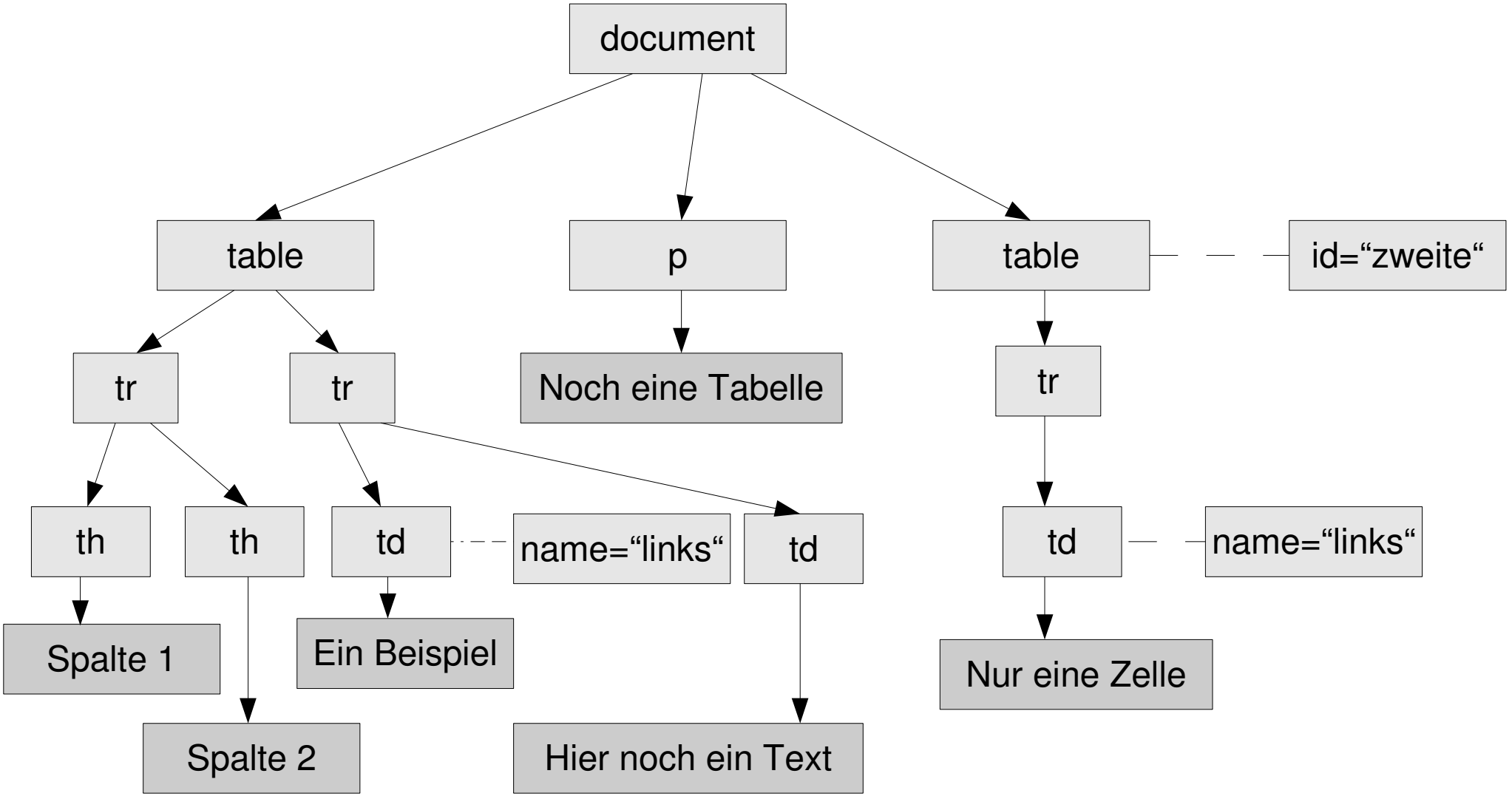
Arbeiten mit JavaScript

- Adressierung der Knoten im Baum über JavaScript möglich
  - und auch deren Modifikation!
- Oberstes Element: „document“
  - der IE führte „all“ ein (JScript) – Inkompatibilitäten!
- Darüber kommt schon der Browser
  - window
  - frames
- Einfache Handhabung einer Liste von Elementen
  - damit das ganze Dokument zu inspizieren

Beispiel

```
<html>
<head>...</head>
<body>
  <table>
    <tr>
      <th>Spalte 1</th><th>Spalte 2</th>
    </tr>
    <tr>
      <td name="links">Ein Beispiel</td><td>Hier noch ein Text</td>
    </tr>
  </table>
  <p>Noch eine Tabelle</p>
  <table id="zweite">
    <tr><td name="links">Nur eine Zelle</td></tr>
  </table>
</body>
</html>
```

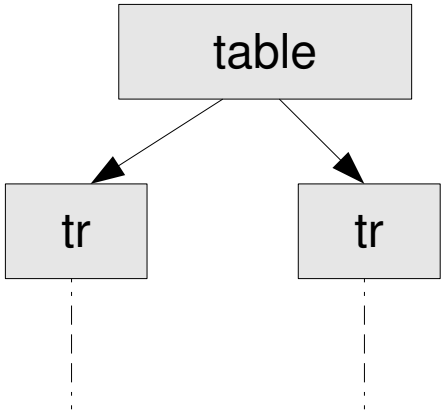
Beispiel


document
table
p
table
id=“zweite“
tr
tr
Noch eine Tabelle
tr
th
th
td
name=“links“
td
td
name=“links“
Spalte 1
Ein Beispiel
Nur eine Zelle
Spalte 2
Hier noch ein Text

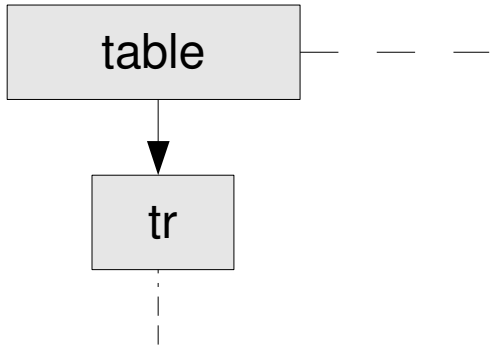
Beispiel

```
var tabelle = document.getElementsByTagName("table");
```

tabelle[0] =



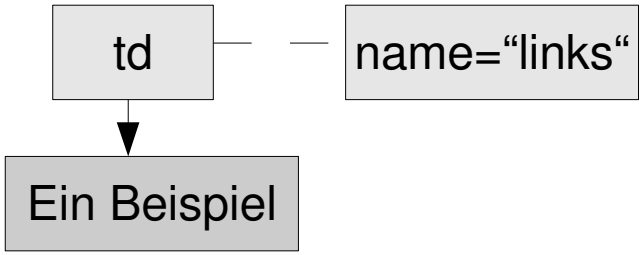
tabelle[1] =



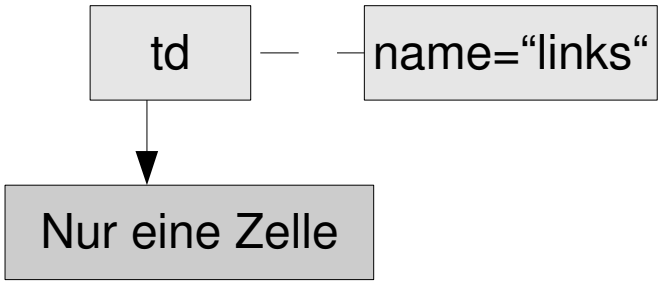
Beispiel

```
var linke = document.getElementsByName("links");
```

linke[0]=



linke[1]=



Ereignisse

- Ereignisse treten auf, wenn der Benutzer seinen Browser bedient
- JavaScript-versehene Seiten können spezifisch darauf reagieren
  - eines der Hauptanwendungsgebiete von JavaScript!
- Beispiel

```
<p onMouseOver='style.backgroundColor=“red“'
    onMouseOut='style.backgroundColor=“white“'>
Ein Beispiel<br/>
über mehrere Zeilen
</p>
```

Faustregel: CSS-Stilbefehl ohne Bindestriche
und dafür das angefügte Wort mit Großbuchstaben
am Anfang:
CSS: wort1-wort2:wert => JS: style.wort1Wort2=wert

Ereignisse

Befehl	nur bei	Zweck
onAbort	img	beim Abbruch des Ladens von Bildern
onClick	-	Klick auf das Element
onDblClick	-	Doppelklick
onMouseDown	-	Mausknopf gedrückt (und gehalten)
onMouseUp	-	Mausknopf losgelassen
onReset	form	Formular gelöscht
onMouseMove	-	Maus bewegt
onMouseOver	-	Maus über das Element bewegt
onMouseOut	-	Maus aus dem Elementbereich herausgezogen
onKeyPress	-	Tastendruck (z.B. in einem Eingabefeld)
onSelect	input, textarea	Auswahl eines Textes
onError	img	Bild konnte nicht geladen werden

Dies ist nur eine Auswahl.

Informationen

- Hinweise
  - Erst prüfen, welche Aufgabe erledigt werden soll
    - keine serverseitige Verarbeitung (kein Zähler, Gästebuch usw.)
  - Dann prüfen, was man mit HTML und CSS bereits lösen kann
    - Hervorheben beim Darüberschieben der Maus ist schon mit CSS möglich
  - Keine Standardfunktionen außer Kraft setzen
    - Chronik (history), Zurück-Knopf, Rechtsklick

- Weitere Informationen in de.selfhtml.org