Web-basierte Anwendungen

Studiengänge AI (4140) & WI (4120)

Auszeichnungssprachen

Von SGML über XML bis HTML5

Textauszeichnungen

- Der Ursprung des Begriffs „Textauszeichnung“
  - Randnotizen etc. in Manuskripten, für Anweisungen des Autors an die Schriftsetzer in der Druckerei
    - zur Struktur des Dokuments (Überschrift, Fußnote, neuer Absatz)
    - mit Formatierhinweisen (fett, größer, unterstrichen, …)
  - Auch im Korrekturprozess zwischen Redakteur und Autorin genutzt
  - Englischer Begriff: „*markup*“
- DV-technische Weiterentwicklung: Auszeichnungssprachen
  - Texte wurden bald auch in Computern erfasst.
  - Aus der auch hier bestehenden Notwendigkeit für Textauszeichnungen entwickelten sich verschiedene Auszeichnungssprachen (*markup languages*)
  - Beispiele:
    - *roff (nroff, troff, ditroff, groff, …), LaTeX, SGML; PDF, PS (prozedural)

Auszeichnungssprachen

- *roff-Beispiel
  (Quelle: http://www.gnu.org/software/groff/grohtml.pdf)

```
.TL
A basic title
.NH 1
Heading at level 1
.NH 2
Heading at level 2
.LP
First paragraph body
```

- LaTeX-Beispiel

```
\title{A basic title}
\section{Heading at level 1}
\subsection{Heading at level 2}
First paragraph body
```

- SGML-Beispiel (HTML)

```
<body>
<h1>A basic title</h1>
<h2>1. Heading at level 1</h2>
<h3>1.1. Heading at level 2</h3>
<p>First paragraph body</p>
</body>
```

Beispiel für eine spätere Darstellung:

A basic title

1. Heading at level 1

1.1. Heading at level 2

First paragraph body

SGML

- … oder: Der Weg zum universellen Datenformat

- 1969: Markup, GML
  - Charles Goldfarb, Ed Mosher, Ray Lorie (IBM)
  - GML = *Generalized Markup Language*
  - Prinzipien:
    - Einheitliche *representation* als *markup*
    - Erweiterbarkeit der *Markup*-Sprache
    - Formale Definition & Beschreibung von Dokumenttypen

- 1974: SGML-Geburtsstunde
  - Erster validierender Parser

- 1986: ISO 8879 (SGML)
  - Ausgereifter, komplexer Industriestandard

SGML

- Wichtige Konzepte, von SGML realisiert
  - *Document Type Definitions* (DTDs)
    - DTD = Formale Beschreibung des Aufbaus eines „Dokuments“
      - Welche Elemente? In welcher Reihenfolge? Wie verschachtelt? Wie häufig? Wo sind welche Attribute zulässig oder erforderlich?
    - **Validierbarkeit** eines Dokuments (entspricht es den hinterlegten Aufbauregeln „seines“ Dokumententyps?)
  - *Style Language*
    - DSSSL (*Document Style Semantics and Specification Language,* sprich „dissel“)
      - zur Festlegung des späteren Aussehens der Daten
    - Damit: **Klare Trennung zwischen abstraktem Inhalt und seiner Darstellung!**
  - *Linking Language*
    - HyTime (ISO Standard zum Verlinken von SGML-Dokumenten)

DTDs

- Beispiele für DTD-Deklarationen
  (XHTML-DTD, vereinfacht)

  - Element-Deklarationen

```
<!ELEMENT p (#PCDATA|a|b|i|em)* >
<!ELEMENT ul (li+) >
```

  - Attributlisten-Deklaration

```
<!ATTLIST img
    src CDATA #REQUIRED
    alt CDATA #IMPLIED
    width  CDATA #IMPLIED
    height CDATA #IMPLIED >
```

Validierung

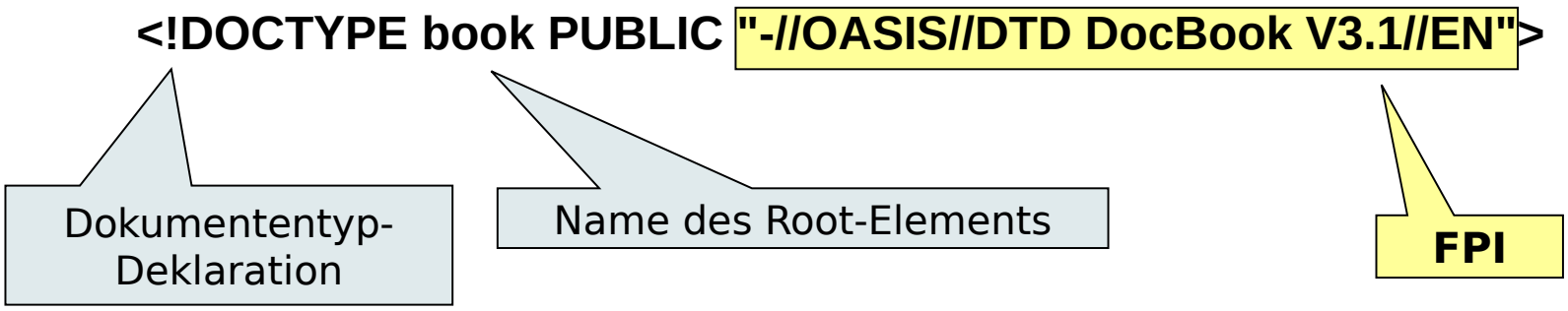
- Analogie
  - Dokumententyp ↔ Klasse
  - Dokument, D.-Instanz ↔ Objekt

- Validierung – Qualitätssicherung von Dokumenten
  - Prüfung eines Dokumentenexemplars gegen sein(e) zugrunde liegende(s) DTD/Schema(ta)
    - 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: https://validator.w3.org

Formal Public Identifier (FPI)

- Die Dokumententyp-Deklaration
  - Herstellung einer Verbindung zwischen einem Dokument und der zuständigen Dokumententyp-Definition
    - DTD eingebettet im Dokument:
      - Viel Redundanz, nur für kurze DTDs geeignet
    - DTD extern:
      - Gut: Viele Dokumente können auf „ihre“ DTD-Datei verweisen
      - Folgeproblem: Wie wird die DTD-Datei gefunden?
  - Antwort: Spezielle Zeilen am Anfang des Dokuments
    - **Der FPI dient als Index in einem Katalog zum Auffinden der DTD**
    - XML: Zusätzliche Angabe eines URI (direkter Verweis auf DTD)

- Bespiel-Dokumententyp DocBook V 3.1:



SGML

- Anwendungen von SGML
  - Schwerpunkt technische Dokumentationen
    - Beispiel Boeing
    - Bekannte DTD: **DocBook** (http://docbook.org)
  - Schwerpunkt Geisteswissenschaften, Linguistik
    - Bekannte DTD: **TEI** (http://www.tei-c.org/index.xml)

- Problemfelder von SGML
  - Sehr komplex
  - Mehrdeutige Lösungswege
  - Erfordert lokal installierte Dateien (insb. die DTDs)
  - Keine Unterstützung für Internet und Web
    - Kein Wunder – SGML entstand früher als das Web

Von SGML zu XML

- **1989: Tim Berners-Lee (CERN, bei Genf)**
  - gründet HTML – auf der Basis von SGML (eigentlich nur GML)
  - HTML = ein bestimmter SGML-Dokumententyp, ohne Erweiterbarkeit

- 199x: Das drohende HTML-Chaos
  - Proprietäre Erweiterungen, inkompatible Browser
  - Antwort: Gründung des Word Wide Web Consortiums (W³C)
  - *W3C-Reaktionen:*
    - *Style sheets* (CSS) - Übernahme eines weiteren GML-Konzepts
    - Erste Ansätze zur standardisierten Erweiterbarkeit von HTML

- 1996: XML Working Group
  - Chair: Jon Bosak, Sun

- 1998-02-10: XML 1.0 - endlich der „große Wurf“?
  - Übernahme auch des dritten Leitgedankens von SGML:
    - Strenge Dokumenttyp-Definitionen und deren Überprüfung
    - Allgemeine Erweiterbarkeit

Von SGML zu XML

- 1998-02-10: XML 1.0 (2008-11-26: 5. Ausgabe)
  - Autoren (allesamt langjährige Markup-Verfechter):
    - Tim Bray (Netscape),
    - Jean Paoli (Microsoft),
    - C.M. Sperberg-McQueen (TEI / W3C)
  - Übernahme auch des dritten Leitgedankens von SGML:
    - Strenge Dokumenttyp-Definitionen und deren Überprüfung
    - Allgemeine Erweiterbarkeit
  - Endlich der „große Wurf“?
- 2000-10-06: XML 1.0 (SE)
  - Inhaltlich unverändert, nur „*errata*“ berücksichtigt
- 2004-02-04: XML 1.1 (2006-09-29: 2. Ausgabe)
  - Kleine Erweiterungen insb. im Unicode-Umfeld

XML: Ein SGML-Subset

- Beibehaltung der wichtigsten Vorzüge von SGML
  - Jedes gültige XML-Dokument ist auch ein gültiges SGML-Dokument
  - 15 Jahre Industriepraxis von SGML werden geerbt
  - Abwärtskompatibilität führt manchmal zu nicht-intuitiven Erweiterungen
- Vereinfachungen für Web-Zwecke
  - minimalistische Tradition
- Weitere Anleihen
  - *Extensible Style Language* (XSL): abgeleitet von
    - **CSS** des Web einerseits und
    - ISO‘s DSSSL (*Document Style Semantics and Specification Language,* sprich „dissel“) andererseits
  - *Extensible Linking Language* (XLink): abgeleitet von
    - HyTime (ISO Standard zum Verlinken von SGML-Dokumenten)
    - TEI (*Text Encoding Initiative*)-Regeln (akadem. SGML-Umfeld)
  - Unicode (http://www.unicode.org), ISO 10646
  - RFC 1766 (*language ID tags*), ISO 639 (*language name codes*), ISO 3166 (*country name codes*)

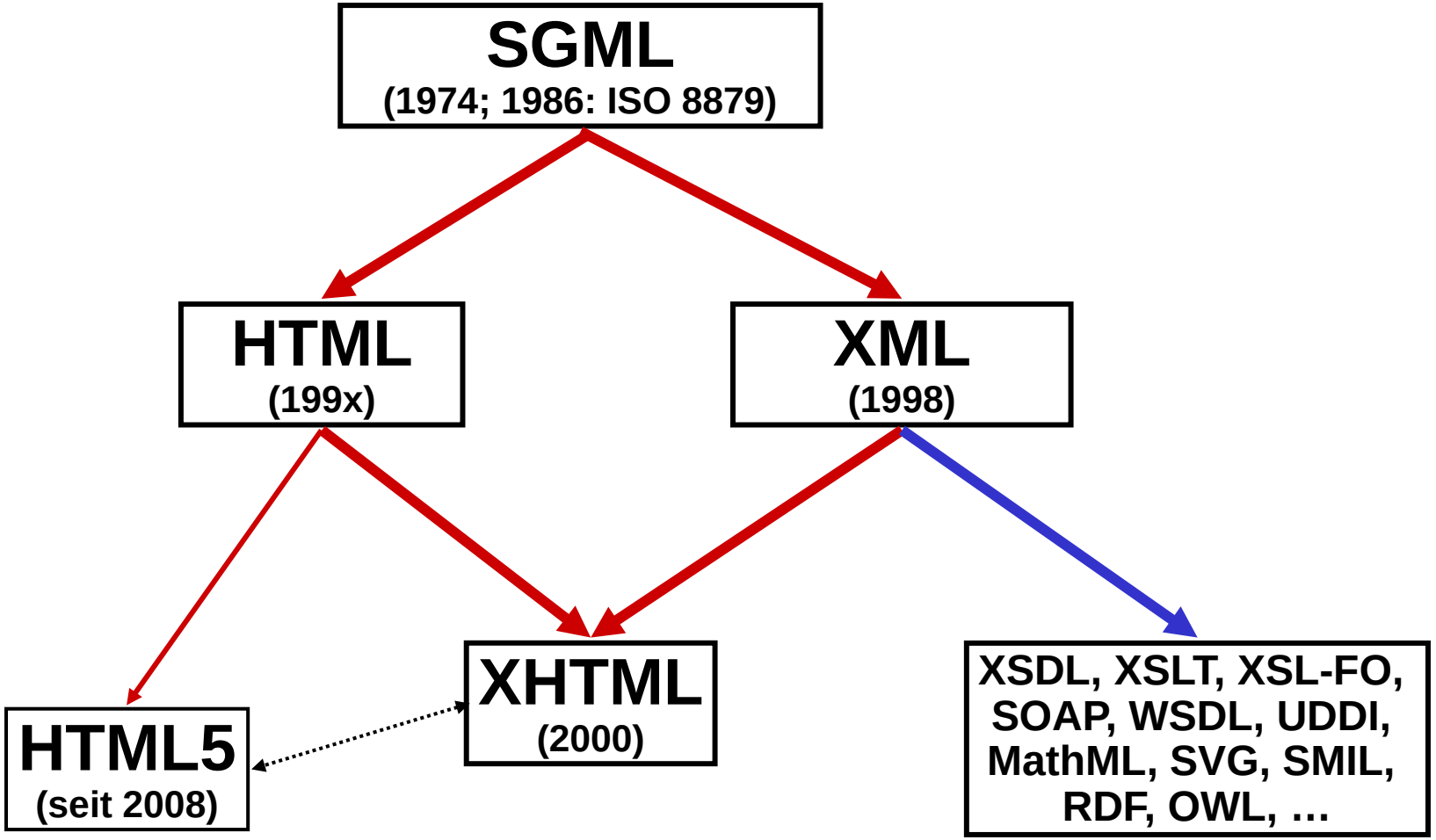
SGML und XML: Meta-Sprachen!

- Dokument-Typen
  - lassen sich als formale Sprachen auffassen
  - Ihre Grammatik besteht aus der jeweiligen DTD (und den allgemeinen Regeln von SGML bzw. XML)
  - Beispiele
    - SVG ist eine XML-basierte Vektorgrafik-Sprache
    - **HTML ist eine SGML-basierte Seitenbeschreibungssprache**
      - Genauer: Es gibt zahlreiche „Dialekte“ (Versionen) von HTML
      - HTML5 hat inzwischen die SGML-Grundlagen verlassen
    - XHTML ist eine XML-basierte Version von HTML

- SGML und XML
  - sind demnach Sprachen zur Generierung von Sprachen, also „Meta-Sprachen“ !

- HTML (auch HTML5)
  - ist dagegen eine konkrete Auszeichnungssprache

SGML, HTML und XHTML: Stammbaum



Grundbegriffe

```
<body>
   <ul>                    Start tag
      <li>Erster Punkt einer Aufzählung</li>
      <li>Zweiter Punkt</li>                          Element
   </ul>                   End tag                    Element
   <p>Etwas Freitext … <br/>Weiterer Text</p>
</body>
```

Empty element tag
(nur XML oder HTML5)

Ausschnitt aus einem XHTML- bzw. HTML5-Dokument

- Er besteht aus einem Element (namens „body“)
- Dieses Element enthält Unter-Elemente usw.: Baum-Struktur!
- Elemente können auch Freitext enthalten

Genau genommen sehen Sie hier eine Textdarstellung. Es gibt auch andere Darstellungen von HTML-Dokumenten, etwa die sehr kompakte binäre EXI-Darstellung.

Grundbegriffe

```
<!-- Dieser Text erscheint nicht im Browser -->
<body>
   <ul id='aufzaehlung1'>
      <li class='kreise'>Erster Punkt …</li>
      <li class="kreise">Zweiter Punkt</li>
   </ul>
   <p>Etwas Freitext … <br/>Weiterer Text</p>
</body>
```

Attribute

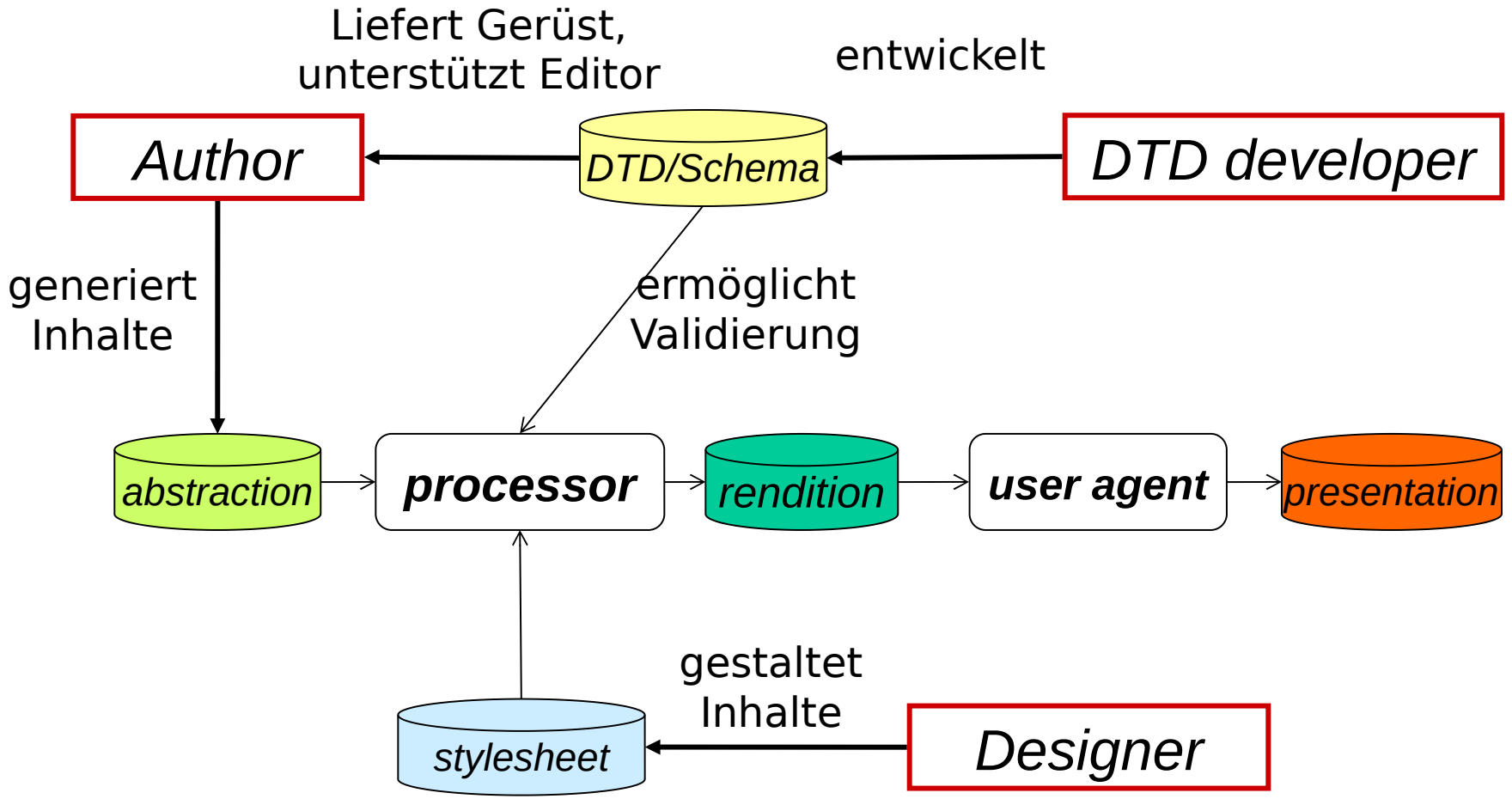
Kommentar

Attributname

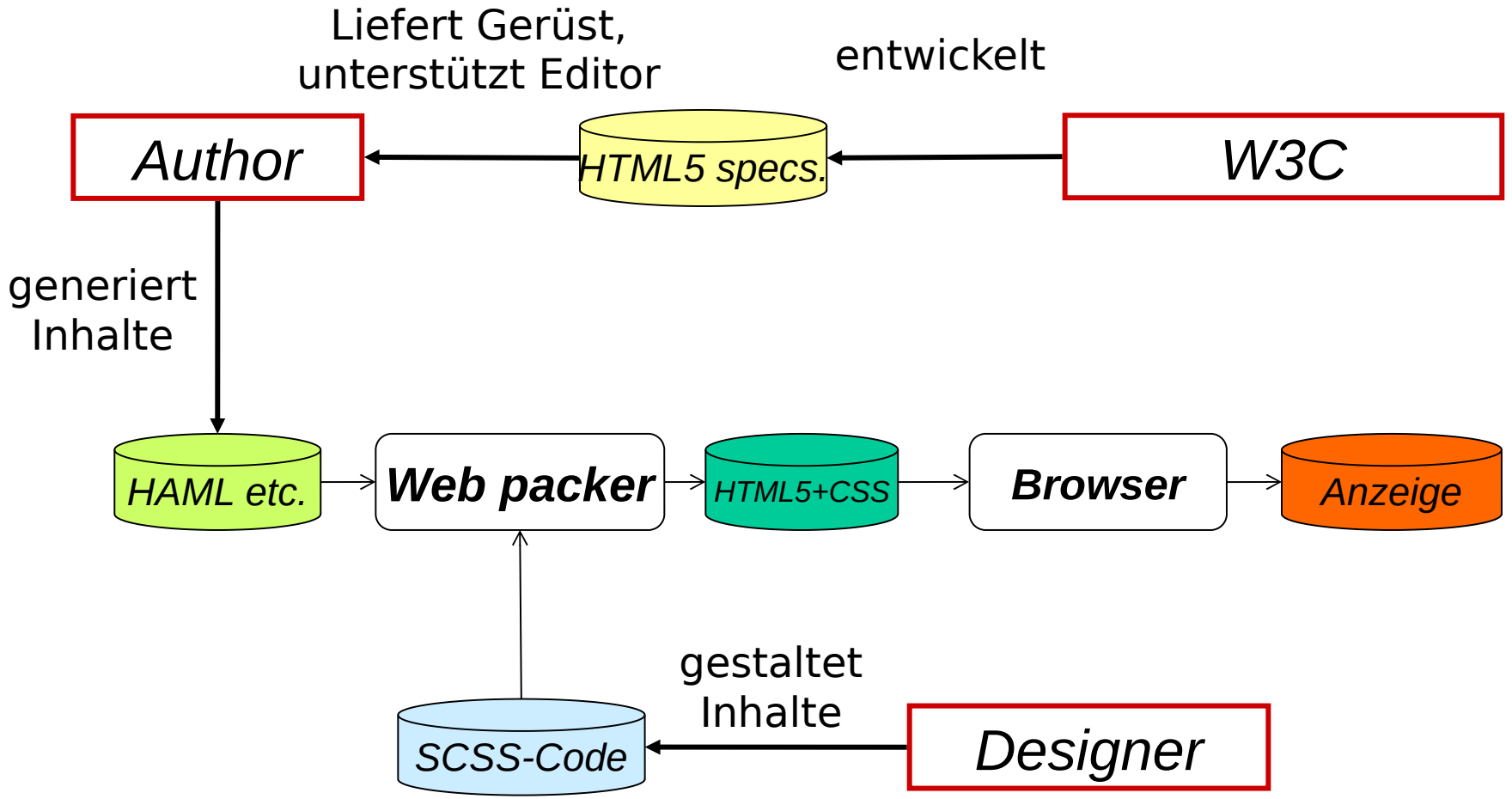
Attributwert

Elementname

Auszeichnungssprachen: Trennung der Aufgaben

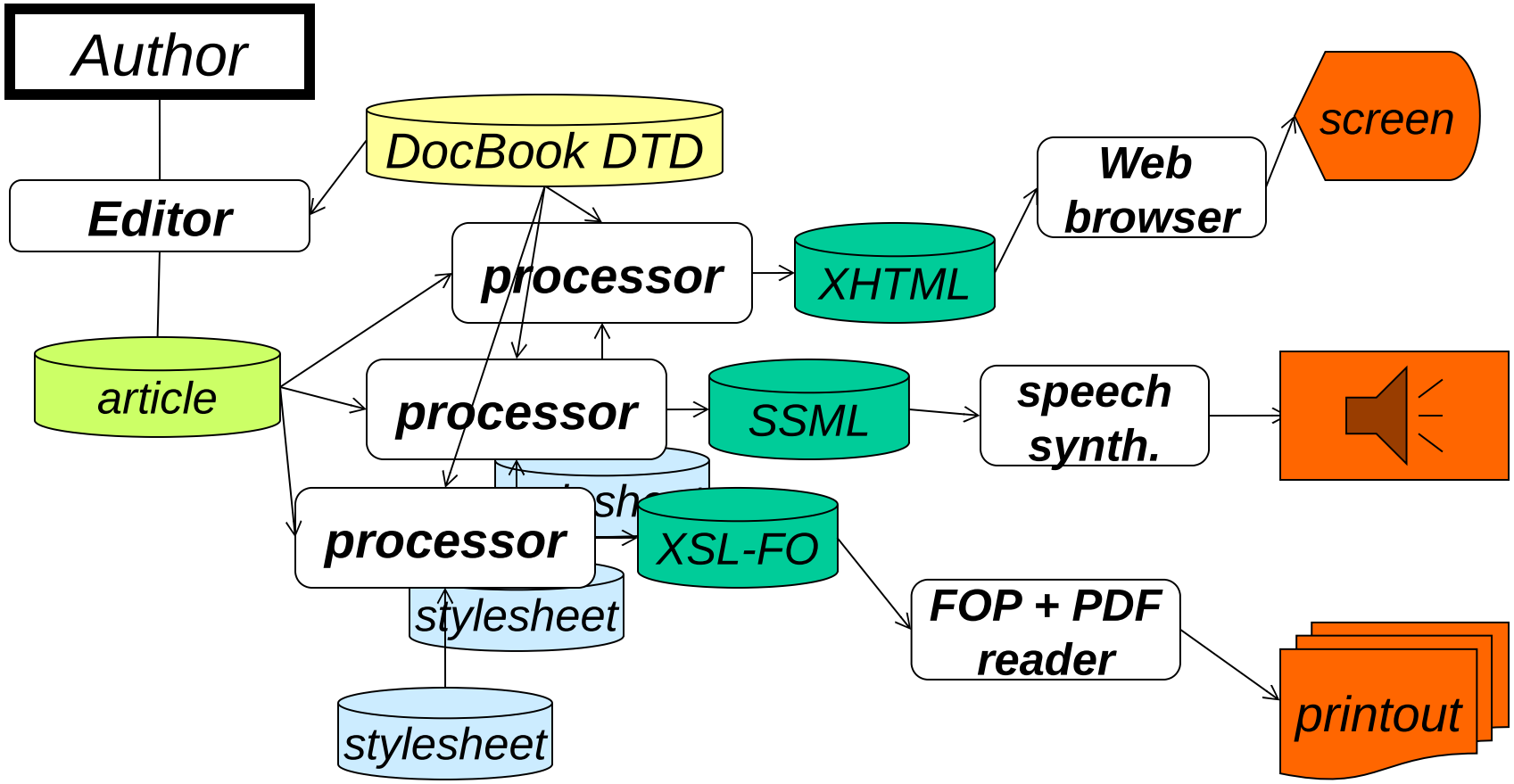


Auszeichnungssprachen: Trennung der Aufgaben



Beispiel *People-Oriented Publishing* mit XML

Vision: Mehrfach-Verwertung eines abstrakten Dokuments



Grundlagen:
XML, XHTML, HTML

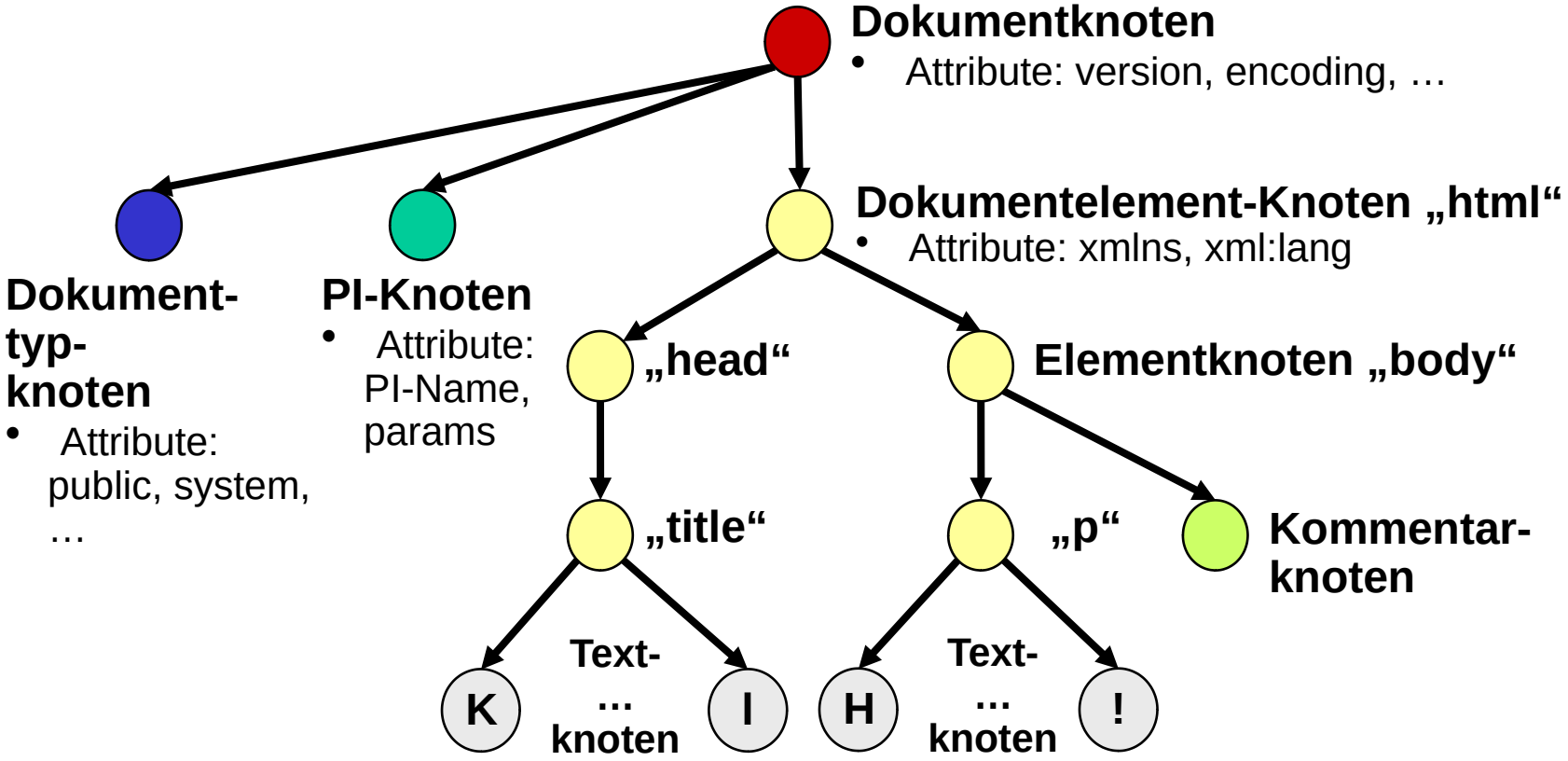
Von HTML zu XHTML

- XML-Dokument
  - Abstrakte Sicht: Bewerteter Graph in Baumform
  - Modell: XML Information Set (https://www.w3.org/TR/infoset)
  - Übliche Darstellung: XML-Syntax (https://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, …

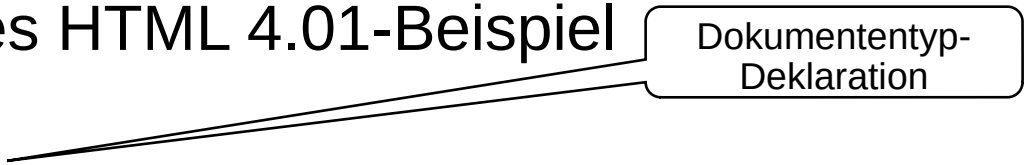
Datenmodell eines XML/HTML-Dokuments

- Abstrakte Beschreibung der XML/HTML-Dokumente :
  - Sie sind markierte (attributierte), baumartige Graphen
  - Sie besitzen verschiedene Knotentypen:

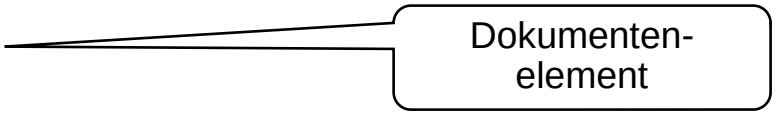


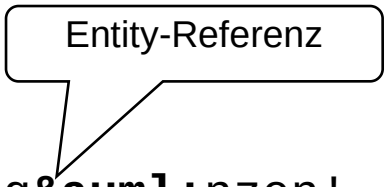
Struktur eines HTML4-Dokuments

- Einfaches HTML 4.01-Beispiel




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





Struktur eines XHTML-Dokuments

- Einfaches XHTML 1.0-Beispiel

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html
   xmlns="http://www.w3.org/1999/xhtml" lang="de" xml:lang="de">
  <head>
    <title>Kleines XHTML-Beispiel</title>
    <link rel="stylesheet" type="text/css"
          media="screen" href="hello.css" />
  </head>
  <body>
    <p>Hallo, Welt!</p>
       <!-- Kommentar: Hier ergänzen! -->
  </body>
</html>
```

XML-Deklaration

Zeichensatz-Code!

Dokumententyp-Deklaration

Namensraum-URI

Globales Attribut

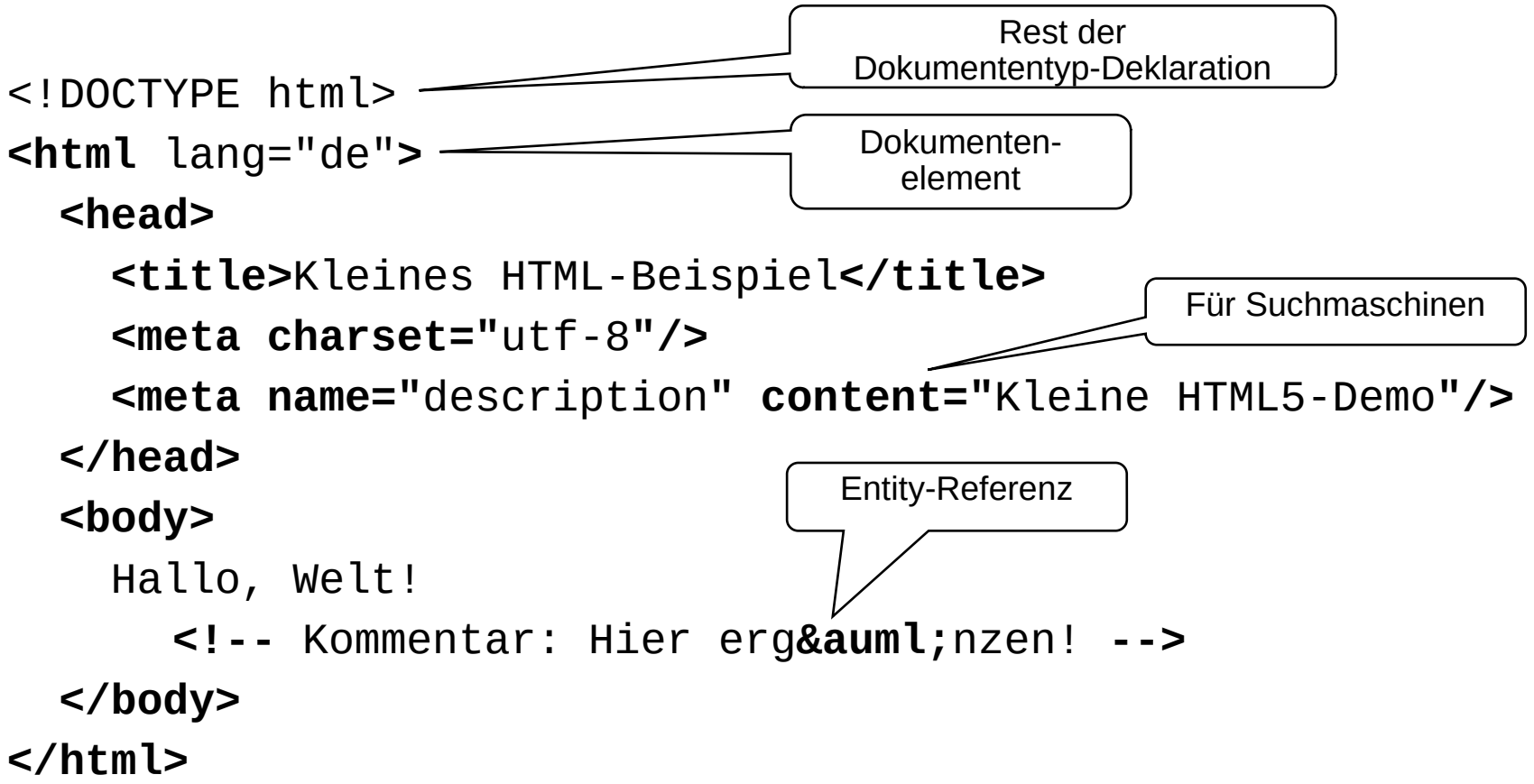
Stylesheet-Einbindung

Von DTD gefordert!

Code des Sonderzeichens muss zum Zeichensatz passen!

Struktur eines HTML5-Dokuments

- Einfaches HTML5-Beispiel mit einigen Header-Angaben




```
<!DOCTYPE html>
<html lang="de">
  <head>
    <title>Kleines HTML-Beispiel</title>
    <meta charset="utf-8"/>
    <meta name="description" content="Kleine HTML5-Demo"/>
  </head>
  <body>
    Hallo, Welt!
      <!-- Kommentar: Hier erg&auml;nzen! -->
  </body>
</html>
```

Beispiel: 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

I.d.R. durch CSS-Anweisungen ersetzen!

Blau gefärbt: Mit Relevanz im Praktikum

HTML 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 (X)HTML- und XML-Inhalten
  - sollen (X)HTML von Darstellungselementen wieder befreien.

HTML und CSS

- Gutes Beispiel: „em“

```
<p>Dies ist <em>wichtig</em></p>
```

  - „em“ bringt den (abstrakten) Wunsch nach Betonung zum Ausdruck, ohne deren gestalterische Realisierung vorwegzunehmen
  - Einige Möglichkeiten, in CSS auf „em“ einzugehen:
    - Medientyp „screen“: Unterstreichen, kursiv setzen, Textfarbe ändern, Hintergrund färben, fett oder größer drucken, blinken, …
    - Medientyp „print“: Unterstreichen, kursiv setzen, fett oder größer drucken, grau hinterlegen, …
    - Medientyp „aural“: Lauter aussprechen, Stimme erhöhen, …

- Schlechtes Beispiel: „b“

```
<p>Dies ist <b>fett gedruckt</b></p>
```

  - „b“ mischt sich in die gestalterische Realisierung ein!
  - Etwas anderes als Fettdruck ist nicht möglich, für andere Medientypen muss man „raten“
    - Umgestaltung (Änderung der Wirkung von „b“) per CSS ist zwar möglich, würde aber sehr verwirren – vermeiden!

HTML und CSS

- **Trennung von CSS- und HTML-Code**
  - **DRY-Prinzip → CSS-Anweisungen sollten nur im Ausnahmefall direkt im HTML-Code stehen.**
  - **Im Normalfall befindet sich CSS-Code in separaten Dateien!**
    - Dies ermöglicht u.a. eine klare Aufgabentrennung zwischen CSS- und HTML-Entwicklern und einheitliches Aussehen von Websites, die aus vielen einheitlich gestalteten HTML-Seiten bestehen.
  - Folgeproblem:
    Wie „adressiert“ man bestimmte HTML-Stellen mit CSS?
  - Antwort 1: Mit **CSS-Selektoren**
    - Einige davon lernen wir im CSS-Kapitel kennen
  - Antwort 2: Mit HTML-seitiger Unterstützung zur leichten Selektierbarkeit
    - Globale Attribute „**class**“ und „**id**“
    - Abstrakte Gruppierungselemente „**div**“ (blockbildend) und „**span**“ (inline)

HTML und CSS

- Kleine CSS-Vorschau
  - Grundstruktur einer CSS-Anweisung:

```
selector {
   property1: value1;
   property2: value2;
   /* comment */
}
```

  - Beispiele:

```
h1 {
 font-size: 14pt;
 text-align: center;
}

#aufzaehlung1 { margin: 2cm; }          /* id */

.kreise { list-style-type: circle; }    /* class */
```

HTML und CSS

- Das globale Attribut „**id**“
  - Es wirkt ähnlich wie der Primärschlüssel in einer DB-Tabelle
  - Es darf in allen Elementen vorkommen
  - Sein Wert muss eindeutig sein im gesamten Dokument
    - Global, nicht nur pro Element-Name!
  - Für seine Werte gelten ähnliche Regeln wie für Variablennamen
  - Beispiel:

```
<body>
  <ul id='aufzaehlung1'>
      <li class='kreise'>Erster Punkt …</li>
      <li class="kreise">Zweiter Punkt</li>
  </ul>
  <p>Etwas Freitext … <br/>Weiterer Text</p>
</body>
```

    - CSS: Dieses eine ul-Element anders gestalten als die anderen
    - DOM/AJAX: Diese Liste dynamisch um ein neues li-Element verlängern

HTML und CSS

- Das globale Attribut „**class**“
  - Es wird verwendet, um ausgewählte Elemente denselben Formatierungsregeln auszusetzen (Klassenbildung)
  - Es ist dabei möglich, auch verschiedenartige Elemente derselben „Klasse“ zuzuweisen (sofern das aus CSS-Sicht sinnvoll ist)
  - Beispiel:

```
<body>
  <ul id='aufzaehlung1'>
      <li class='kreise'>Erster Punkt …</li>
      <li class="kreise">Zweiter Punkt</li>
  </ul>
  <p>Etwas Freitext … <br/>Weiterer Text</p>
</body>
```

    - CSS: Alle li-Elemente dieser Klasse mit o statt • markieren!
  - Ein Element kann mehreren Klassen angehören. Der Attributwert von „class“ wird dann zu einer Liste von Klassennamen.

HTML und CSS

- Das abstrakte blockbildende Element „**div**“
  - Es wird verwendet, um Bereiche einer Seite logisch zu gruppieren
  - Wichtige Grundlage für das Seiten-Layout
  - Div-Elemente können auch kaskadiert werden (div in div)
  - Sie werden typischerweise per „id“ unterschieden!
  - Blockbildend: div-Grenzen sind auch immer Zeilengrenzen
  - Beispiel:

```
<body>
  <div id="nav"> … </div>
  <div id="header"><h1>Überschrift</h1></div>
  <div id="main"> … </div>
</body>
```

    - nav: Navigationsbalken, Menüleiste, …
    - header: Für Überschrift, Logo, …
    - main: Für den Hauptteil der Seite
  - Ohne CSS-Anweisungen erscheinen div-Blöcke untereinander!

HTML und CSS

- Das abstrakte Inline-Element „**span**“
  - Es wird verwendet, um Textabschnitte innerhalb eines Blocks logisch zu gruppieren
    - Typisch sind kurze Abschnitte innerhalb einer Zeile
  - Einsatz von „id“ ist möglich, aber „class“ ist meist sinnvoller
  - Beispiel:

```
<p>
  Fließtext mit einem <span style="color:blue">
  gefärbten</span> und einem <span class="special">
  speziell formatierten</span> Textabschnitt.
</p>
```

    - Attribut „style“: Eingebettete CSS-Anweisung – i.d.R. vermeiden!
    - Attribut „class“: Die Umsetzung erfolgt in CSS-Datei, Selektor: .special

  - Ohne CSS-Anweisungen bleiben span-Elemente ohne erkennbare Wirkung!

Entity- und Zeichenreferenzen

- XHTML, XML, HTML5 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:
    `< > & &quot; &apos;`

```
<Relation> a &lt; b </Relation>
```

a < b

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

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

ß ö

  - HTML5: siehe https://dev.w3.org/html5/html-author/charref

HTML vs. XHTML vs. HTML5

- *Empty elements / void elements*:
  - Einige HTML-Elemente dürfen keinen Inhalt enthalten – weder Text noch Unterelemente
    - Das sind insbesondere: **br, hr, link, meta**
  - Sie werden je nach HTML-Version unterschiedlich dargestellt:
- **HTML bis 4.01**
  - Üblich: **`<br>`** (einfach offen lassen, Browser schließt implizit)
  - Zulässig: **`<br></br>`**, unzulässig: **`<br/>`**
- **XHTML (alle Versionen)**
  - Üblich: **`<br/>`** („empty element tag“)
  - Zulässig: **`<br></br>`**, unzulässig: **`<br>`**
- **HTML5**
  - Üblich: **`<br>`** (einfach offen lassen, Browser schließt implizit)
  - Zulässig: **`<br/>`** (wie bei XHTML), unzulässig: **`<br></br>`** (!)

HTML-Generierung

Einmal codieren – mehrere HTML-Versionen erzeugen

Redundanz vermeiden

Besser lesbaren Code schreiben

HTML-Generierung

- **Nachteile der direkten HTML-Codierung**
  - *End tags* kann man leicht vergessen
  - *End tags* sind redundant
  - Erzeugung verschiedener HTML-Versionen aus einer Quelle?
    - Hier geht es um DOCTYPE und Syntax-Besonderheiten wie bei
  - Je weniger Markup, desto lesbarer die Inhalte
  - Schablonentechnik (Einbettung ausführbaren Codes) verkompliziert die Situation weiter
  - Hinweis auf Syntaxfehler bereits bei der Entwicklung erspart spätere Probleme bei Browser-Abhängigkeiten und zumindest einen Teil der externen Validierung

- **Ausweg: Schablonensprachen** (*template languages*)
  - Einführung vereinfachter Notationen für (X)HTML-Quellen
  - Nutzung von Präprozessoren, die daraus (X)HTML erzeugen und dabei bereits einige Syntaxfehler abfangen (und dabei Schablonen auswerten)
  - Hier verwendet: HAML (https://haml.info).
  - Alternativen: z.B. slim (http://slim-lang.com)

- **Alternative: IDE-Plugins**
  - HTML-Code entsteht zwar leichter, aber muss danach gepflegt werden
  - Redundanzen werden also teils automatisiert verwaltet, aber nicht beseitigt

HTML-Generierung

- **Haml** (https://haml.info)
  - Eine Markup-Sprache zur Generierung von HTML
    - Verbreitung insb. bei Rails-Projekten, aber auch allgemein verwendbar
  - Ersetzt auch Schablonen-Systeme wie PHP, ASP, eRb
  - Prüft korrekten Seitenaufbau bereits bei der Seitengenerierung
  - **Abstrahiert von den verschiedenen HTML-Darstellungen/-Versionen (!)**
  - Vermeidet Redundanzen, reduziert Code, verbessert die Lesbarkeit

Demo: https://haml.info/tutorial.html

- Wichtige Bestandteile
  - **Einrückungen sind signifikant ➞** keine *end tags* mehr erforderlich
    - Analogie zu Python
  - Einfache 1:1-Beziehung zwischen HTML-Elementnamen und **haml**
    - <body>…</body> → %body …
  - Sinnvolle Abkürzungen, **mit CSS harmonisiert**

`<!-- Kommentar -->`	➞	`/ Kommentar`
`<div id="elem1">`	➞	`#elem1`
`<div class="my_class">`	➞	`.my_class`
`<em class="loud ul">`	➞	`%em.loud.ul`
`<a href="#">…</a>`	➞	`%a{href: "#"} …`

HTML-Generierung

- **Reorganisation der HTML-Erzeugung**
  - AI: In Einf.Inf.-P-Aufgabe 06 erzeugten Sie mehrere HTML-Seiten. Die Anfänge dieser Seiten waren (fast) identisch: Doctype, html-Element und vor allem das Element „head“ waren gleich (Ausnahme: Inhalt von „title“)
  - Vermeidung dieser Redundanz: Aufteilung der Seitenerzeugung

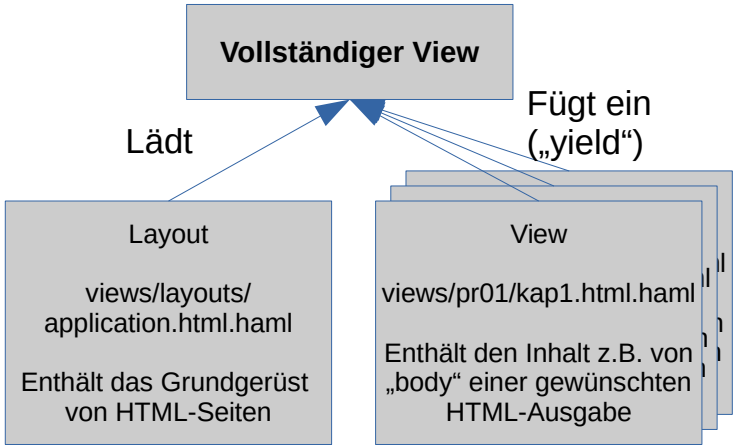
- Vorgehen:
  - Ein gemeinsamer Teil („**Layout**“) bestimmt die Struktur einer Seite.
    - In Aufgabe 01 umfasst das Layout alles außer dem Inhalt von „body“. I.A. wird hier auch die grobe Struktur einer Seite innerhalb von Body festgelegt.
  - Der variable Teil jeder Seite („**View**“) erzeugt ausschließlich die Inhalte von „body“ bzw. des für diesen Haupt-Teil vorgesehenen Unterelements von „body“
    - In Aufgabe 02 werden wir „body“ strukturieren und u.a. mit CSS-Mitteln gestalten. Aus der bisher separat erzeugten Seite „index“ wird dann eine Navigationsspalte, die bisherigen Inhalte von „body“ werden in ein Unterelement von „body“ verlagert.
  - Bei größeren Projekten gibt es mehrere Layouts. Der Controller entscheidet, welches Layout mit welchem View kombiniert wird.
    - Rails wählt hier sinnvolle Voreinstellungen. Bei Bedarf im Controller ändern!

HTML-Generierung

- **Reorganisation der Gestaltungsangaben**
  - Es gibt mehrere Wege, Gestaltungsangaben (in Form von CSS-Code) hinzuzufügen
  - Auch hier sollte nicht alles technisch Mögliche genutzt, sondern Redundanzen vermieden werden. CSS-Code, der von mehreren Views genutzt wird, gehört in eine ausgelagerte, gemeinsam genutzte Datei
  - Selbst bei nur einmaliger Verwendung von CSS-Code sollte man diesen nicht in eine HTML-Seite einbetten, sondern ebenfalls auslagern. Das erleichtert die Code-Pflege und eine arbeitsteilige Organisation: View-Entwickler und Gestalter können parallel an separaten Quelldateien arbeiten.
  - Weiterer Vorteil: Spätere Änderungen erfolgen an nur einer zentralen Stelle
  - Ausblick: JavaScript-Code wird später ähnlich behandelt (→ UJS)

HTML-Generierung

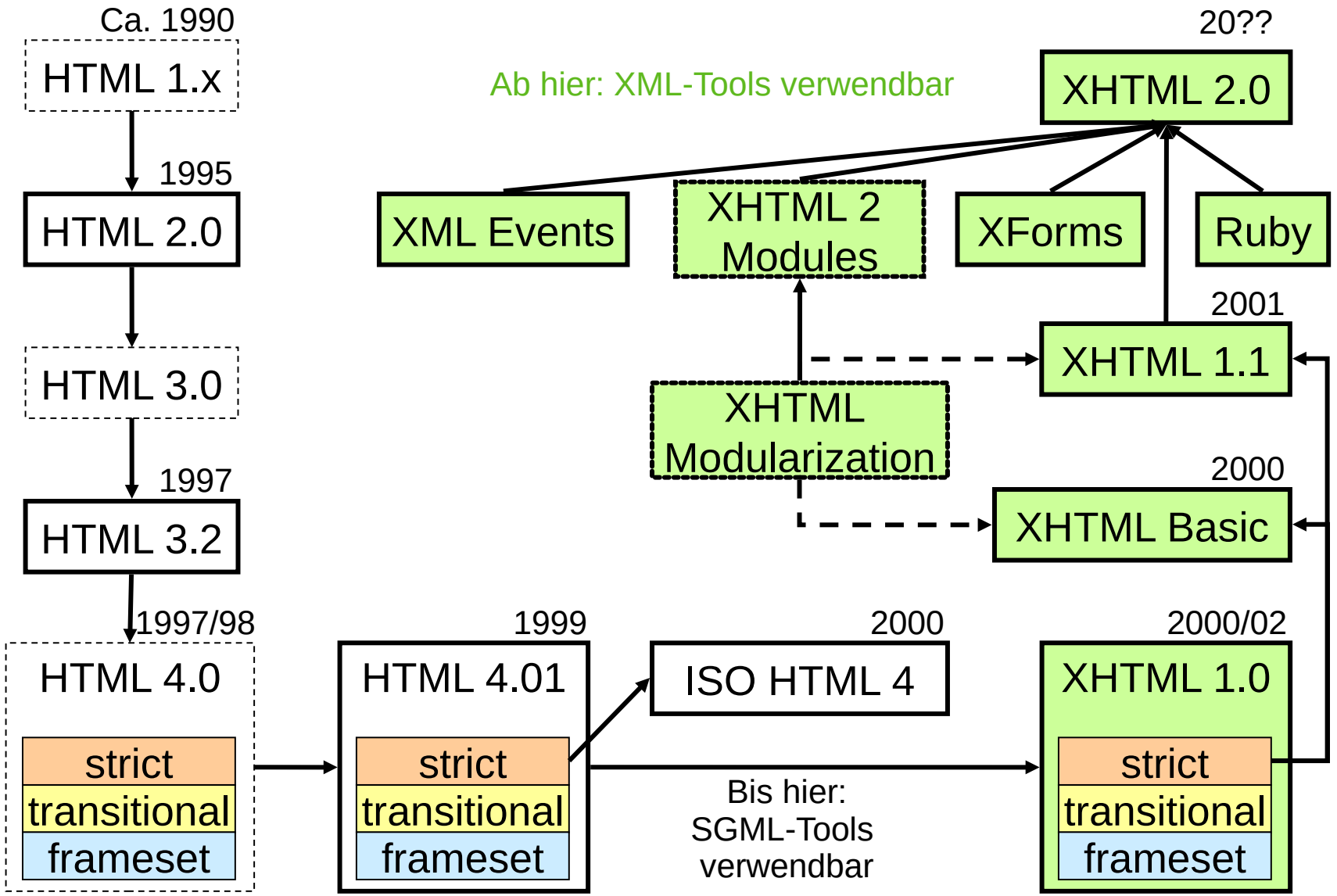
- Die Situation bei Rails:
  - Rails organisiert diese Trennungen, gemäß dem DRY-Prinzip (DRY: „Don‘t repeat yourself“)
  - Views werden ihrem Controller zugewiesen. Zu jedem Controller gibt es i.d.R. einen eigenen Ordner in „views“, der die Views/Templates für diesen Controller enthält.



  - Layouts sind spezielle Views und werden daher in einem Ordner „layouts“ unterhalb von „views“ gesammelt
  - CSS-Code wird ebenfalls nach Controller-Zugehörigkeit getrennt. CSS-Dateien befinden sich in einem eigenen Ordner „stylesheets“ unterhalb von „assets“.
    - Hinweis: Rails lädt immer alle CSS-Daten, nicht nur die zum jeweils aktiven Controller bzw. View! Einsortieren Ihres CSS-Codes entsprechend der Stellen, die ihn benötigen, ist aber hilfreich bei der späteren Code-Pflege.
    - Beispiel-Frage aus der Praxis: Welcher CSS-Code ist veraltet und sollte daher entfernt werden? Ist der Code nur einigen Views eines der Controller zugeordnet, können Sie viel rascher entscheiden.

Die Entwicklung von (X)HTML

HTML und XHTML: Übersicht



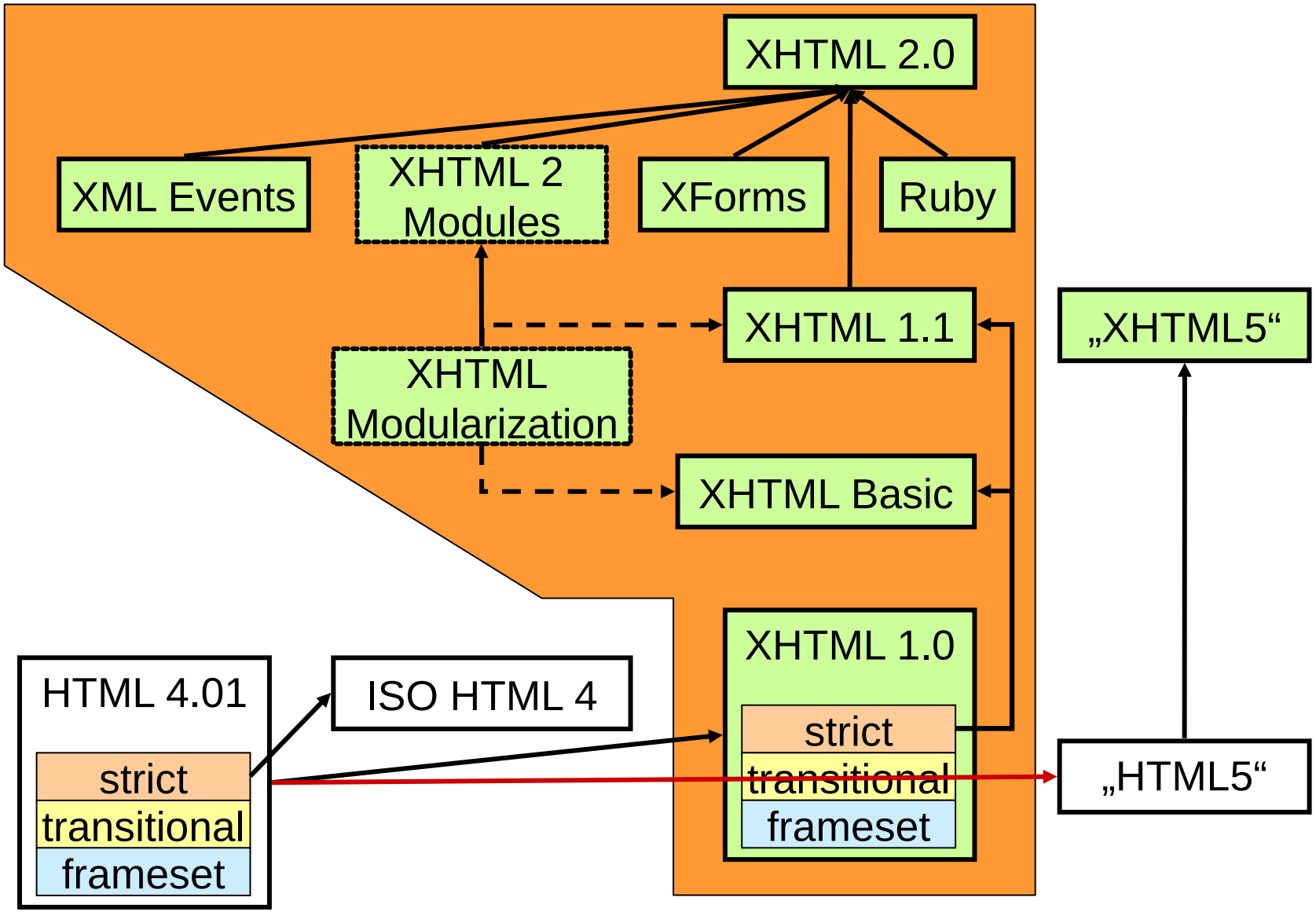
Das W3C und die WHATWG

Die Entwicklung zu HTML5

Das W3C und die WHATWG

- **Die *WHATWG***
  - *WHATWG = Web Hypertext Application Technology Working Group (www.whatwg.org)*
  - Forderungen
    - Endlich Weiterentwicklung, von HTML 4.01 zu „HTML5“
    - XML-Variante dazu namens „XHTML5“
    - Ignorieren von XHTML 2.0 u.a. W3C-Empfehlungen (!)
  - Was soll HTML5 sein?
    - Endlich eine präzise Spezifikation von HTML (interoperable Browser!)
    - Web Forms 2.0 (Übernahme vieler Ideen von XForms!)
    - Web Applications 1.0 (Neue HTML- und DOM-Elemente wie „meter“, „progress“ etc., mit Elementen / Widgets normaler Anwendungen wie Schieberegler, Drehknöpfe, Fortschrittsbalken auf Web-Seiten ermöglichen.
    - (u.v.a.m.)
  - Treibende Kräfte: **Apple, Mozilla, Opera (!)** (Google erst später)
    - Technische Gründe: HTML auch für Smartphones (und deren APIs)

(X)HTML: Neuausrichtung



Das W3C und die WHATWG

- **Diskussion**
  - Risiken
    - Fragmentierung des Web
    - Positionierung von Microsoft?
    - Drohender Verlust integrierender Möglichkeiten wie SVG und/oder MathML in XHTML
  - Chancen
    - Besser an den Status Quo angepasste, einfachere Standards
    - Auflösung der Entwicklungblockade von (X)HTML

2009: W3C ist „dabei“

2010: MS: „HTML5 gehört die Zukunft“

SVG- und MathML-Einbettung in HTML5!

- **Reaktion von Tim Berners-Lee, W3C**
  - Zunächst: Nun doch inkrementelle Weiterentwicklung von HTML (aber gemeinsam mit XHTML), dabei Erhalt des XHTML 2.0-Teams
  - Übernahme einiger Positionen der WHAT WG
  - Mitte 2009: Einstellung der Arbeiten an XHTML 2.0 zum Jahresende, Unterstützung von „HTML5“ !
  - W3C Rec-Status von HTML5 erreicht am 28.10.2014
  - Aktuell: **HTML 5.2** (W3C Rec 14.12.2021, ersetzt am 28.01.2021)
    - https://www.w3.org/TR/html52/

Mehr zum Thema in: Webstandards im Wandel, Herbert Braun, c't 1/2007, S. 162-169.

Das W3C und die WHATWG

- **Aktueller Stand (2021)**
  - Die WHATWG hat sich durchgesetzt. Sie verfolgt weiterhin ihr Konzept vom “Living Standard”. Die zentrale Standard-Pflege von HTML und DOM erfolgt nun hier:
    - https://html.spec.whatwg.org/multipage/
    - https://dom.spec.whatwg.org/
  - Die verbliebenen Browser-Hersteller unterstützen dies, sie sind ja auch der Kern der WHATWG
  - Das W3C folgt der WHATWG, hat also ihr ehemaliges „Kerngeschäft“ HTML & CSS aufgegeben! Das W3C hält formal am Release-Konzept fest. Dazu verwendet es regelmäßige „snapshots“ der WHATWG-Quellen. Aktueller Stand:
    - https://www.w3.org/blog/2019/05/w3c-and-whatwg-to-work-together-to-advance-the-open-web-platform/
    - HTML 5.2 erreichte REC-Status in 2021 (Update in 2021),
    - HTML 5.3 ist im Draft-Status abgebrochen worden und trägt nun einen Verweis auf die WHATWG-Quellen, siehe https://www.w3.org/TR/html53/
-