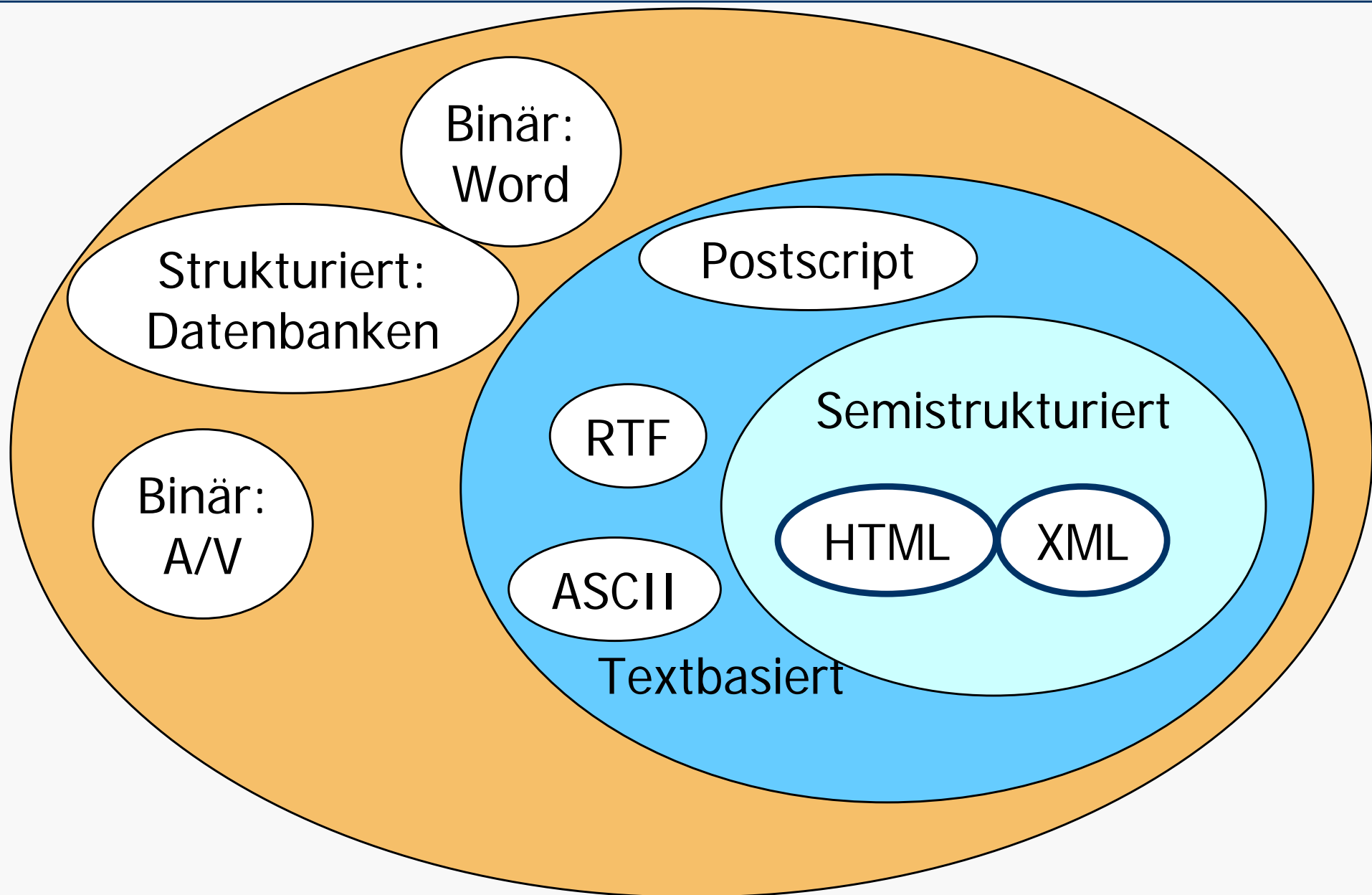




Netzprogrammierung HTML und XML

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierende Informationssysteme
mailto: tolk@inf.fu-berlin.de
<http://www.robert-tolksdorf.de>

1. HTML
2. HTML Verarbeitung
3. XML Dokumente





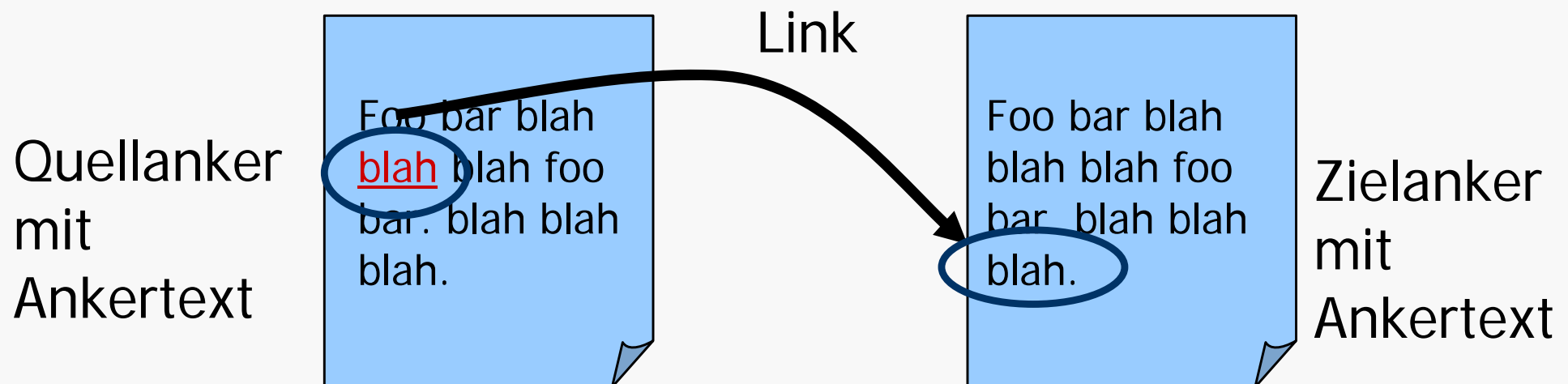
Hypertext Markup Language HTML

Hypertext Markup Language

- Dominierende Sprache zur Auszeichnung von Dokumenten im Internet
- Definiert vom World Wide Web Consortium, W3C:
 - MIT (Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory (CSAIL))
 - ERCIM (European Research Consortium in Informatics and Mathematics)
 - Keio University of Japan
- Jedes Informationssystem im Netz muss:
 - HTML Informationen integrieren können
 - HTML Ausgaben erzeugen
 - Mit HTML-Mitteln mit Nutzern interagieren

Hypertext Markup Language

- Konzepte:
 - Informationen werden als Dokumente aufgefasst
 - Dokumenteninhalte werden als Klartext dargestellt
 - Dokumententeile werden durch *Markierungen/Elemente/Tags* ausgezeichnet
 - Inhaltlich (`<h1>Einleitung</h1>`, `wichtig`)
 - Gestalterisch (`wichtig`)
 - Dokumente werden durch Links zu einem Hypertext verbunden (dadurch entsteht ein Netz, das Web)



- Sprache umfasst
 - Elemente
 - `<h1>Neue Vorlesungen</h1>`
 - `
`
 - `<hr>`
 - Attribute
 - `<hr height="3">`
 - Entitäten
 - `&`
 - `ä ;`
 - Grammatikalische Regeln über Elemente
 - `<html >` ist Startsymbol,
 - darin die Elemente `<head>` und `<body>`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN>
```

```
<html>
```

```
<head>
```

```
<title>FU-Berlin: Institut für Informatik</title>
```

```
<base href="http://www.inf.fu-berlin.de">
```

```
</head>
```

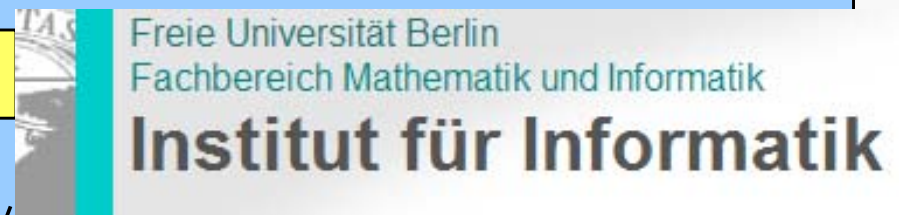
```
<body>
```

```
<p><a href="http://www.fu-berlin.de/">Freie Universität  
Berlin</a><br>
```

```
<a href="http://www.math.fu-berlin.de/">Fachbereich Mathematik  
und Informatik</a></p>
```

```
<h1>Institut für Informatik</h1>
```

```
<p><a href="http://www.inf.fu-berlin.de/index_en.html">Homepage  
in English</a>.</p>
```



HTML Beispiel/2

```
<form method="get" action="http://www.google.com/search">  
<a href="http://www.google.de">Google</a>-Sitesearch:  
  <nobr><font size=2>  
  <input type=text name=q size=15 maxlength=255 value="">  
  </font></nobr>  
  <input type=hidden name=sitesearch value="inf.fu-berlin.de">  
  <input type=hidden name=domains value="inf.fu-berlin.de">  
  
</form>
```



```
<h2>Aktuelle Meldungen</h2>  
  <p>Das Ferienprojekt <a href=  
"http://www.inf.fu-berlin.de/~block/schachprojekt.html">  
  Schachprogrammierung</a>  
  wird wegen der umfassenden Bauarbeiten im Institut auf die nächsten  
  Semesterferien verschoben!</p>  
</body>  
</html>
```

HTML – Elemente für Struktur

- Struktur
 - ! DOCTYPE gibt Art des Dokuments an:
`<!DOCTYPE HTML PUBLIC
-//W3C//DTD HTML 4.01 Transitional //EN>`
 - `<html >...</html >` umfaßt Dokument
 - `<head>...</head>` enthält Informationen zur Seite
 - `<body>...</body>` umfaßt Inhalt der Seite
- Festes Seitenschema:

```
<!DOCTYPE...>  
<html >  
  <head>...</head>  
  <body>...</body>  
</html >
```

HTML – Elemente im Kopfteil

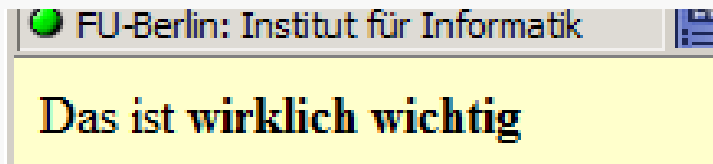
- `<base>` enthält Basis-Adresse der Seite
- `<title>` enthält Titel der Seite
`<title>FU-Berlin: Institut für Informatik</title>`



- `<meta>` enthält
 - Inhaltsklassifikation der Seite
`<meta scheme="ISBN" name="identifier" content="0-8230-2355-9">`
 - oder Protokollinformation
`<meta http-equiv="Expires" content="Tue 24 Sep 2002 00:00:00 GMT">`
- `<link>` gibt Beziehung zu anderer Seite an
`<link rel="Glossary" href="URL">`

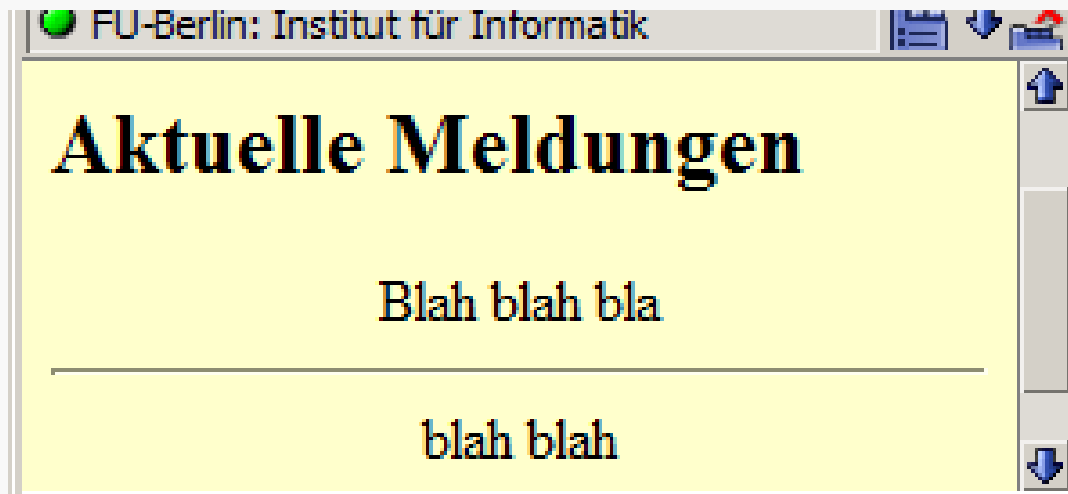
HTML – Elemente für Gestaltung

- Umbruch, Trennungen
wbr br nobr p spacer
Neue
Zeile
- Schriftarten
b i s strike tt u blink bdo marquee
Das ist wirklich wichtig



- Schriftauszeichnung
abbr acronym cite code del dfn em ins kbd samp
strong var ruby rt rb
- Formeln
sub sup
- Schriftgröße
basefont font big small

- Überschriften
h1 h2 h3 h4 h5 h6
`<h2>Aktuelle Meldungen</h2>`
- Blöcke
comment hr div span address pre xmp plaintext
listing blockquote q banner multicol center
`<center>Blah blah bla <hr> blah
blah</center>`



- Listen

ol ul dl r menu li dl dt dd

Im WS 2005/06 bieten wir an

V Netzprogrammierung

P Semantic Web

P Kundenprojekt Web Technologien

K Existenzgründungen in der IT-Industrie

Lehre - Teaching

Im WS 2005/06 bieten wir an

- V Netzprogrammierung
- P Semantic Web
- P Kundenprojekt Web Technologien
- K Existenzgründungen in der IT-Industrie

- Tabellen

table th tr td thead tbody tfoot col col group

```
<table border="1">
<tr><th align="center">Währung</th>
  <th align="center">1 EUR</th></tr>
<tr><td>Deutschland (DEM)</td>
  <td align="right">1,95583</td></tr>
<tr><td>Frankreich (FRF)</td>
  <td align="right">6,55957</td></tr>
</table>
```

- Abbildungen

img overlay
caption map area

Währung	1 EUR
Deutschland (DEM)	1,95583
Frankreich (FRF)	6,55957

- Formulare
form input select option textarea html area
button label fieldset legend
- ```
<form ACTION="/cgi-bin/telefon.cgi "
 METHOD="GET" >
 <i>Die Nummer von</i >
 <input NAME="x" VALUE="" SIZE="30" >
 <input TYPE="submit" VALUE="bitte" >
</form>
```

*Die Nummer von*

bitte

- Browserdarstellung  
style frameset frame iframe noframes layer ilayer
- Applets, Skripte und Objekte  
applet param textflow script noscript object embed bodytext
- Hyperlinks  
a
  - Zielanker:  
<a name="Ziel">Hier Ihre Infos</a>
  - Quellanker:  
<a href="URI">Quellankertext</a>  
<a href="http://x.y.com/seite.html#Ziel">Weitere Infos</a>

[Weitere Infos](#)

- Erschließen des Hypergraphen selber
  - Crawling
- Erschließen der Dokumente
  - HTTP Protokoll über Internet
- Erschließen der Inhalte von Seiten
  - Extraktion aus HTML-Text
  - Nutzung der Informationen in Tags (<address>, <title>)
- Problem: Semantik der Inhalte
  - Wie extrahieren ("Produkt" etc.)
  - Markierung nutzen (<address>, <h\*> etc.)
  - Gestaltung (Bilder, Framesets etc.)



## HTML Verarbeitung

# Verarbeitung von HTML

- Für die Verarbeitung von HTML notwendig
  - Zugriff aus Ressourcen
    - Zugriffsprotokolle
      - Sockets, TCP/UDP
      - HTTP, FTP etc.
    - Implementierungen der Protokolle
      - URLConnection in Java
  - Verarbeitung von HTML Ressourcen
    - Analyse des Inhalts
      - Parser
        - Z.B. Paket `javax.swing.text.html.parser`
    - Datenstruktur für Inhalt
      - DOM HTML

- Paket javax.swing.text.html.parser enthält einen Parser für HTML
- Basiert auf Rückrufen bei Auftreten einer bestimmten Informationseinheit in HTML:

Parserfortschritt

`<html><head><title>HTML Seite</title>...`

Ereignis:  
Start des head Tags gefunden  
Aufruf:  
`handleStartTag(HTML.tag.HEAD`

Ereignis:  
Ende des title Tags gefunden  
Aufruf:  
`handleEndTag(HTML.tag.TITLE)`

Ereignis:  
Start des html Tags gefunden  
Aufruf:  
`handleStartTag(HTML.tag.HTML)`

Ereignis:  
Start Text gefunden  
Aufruf: `handleText("HTML Seite",0)`

# Beispiel: Anzahl der Tags zählen

```
import java.net.*; import java.io.*;
import javax.swing.text.*; import javax.swing.text.html.*;
import javax.swing.text.html.parser.*;

class TagCounter extends HTMLToolkit.ParserCallback {
 int tagCount;

 public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
 tagCount++;
 }
 public void handleEndTag(HTML.Tag t, int pos) {
 if (t==HTML.Tag.HTML) {
 System.out.print(tagCount);
 }
 }
 public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
 tagCount++;
 }
}
```

# Beispiel: Anzahl der Tags zählen

```
public static void main(String[] argv) {
 try { // URL holen
 URL page=new URL(argv[0]);
 URLConnection connection=page.openConnection();
 connection.connect();
 if (connection.getContentType().startsWith("text/html")) {
 // bei HTML Inhalt Ströme aufstecken
 InputStream is = connection.getInputStream();
 InputStreamReader isr = new InputStreamReader(is);
 BufferedReader br = new BufferedReader(isr);
 TagCounter tagCounter= new TagCounter();
 // Parser erzeugen und aufrufen
 ParserDelegator parser = new ParserDelegator();
 parser.parse(br,tagCounter, false);
 }
 } catch (Exception e) {}
}
```

- `void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos)`  
Start-Tag ist an Position *pos* im Strom aufgetreten
- `void handleEndTag(HTML.Tag t, int pos)`  
Ende-Tag ist aufgetreten
- `void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos)`  
Einfaches Tag (<br>) ist aufgetreten
- `void handleComment(char[] data, int pos)`  
Kommentar (<!-- .... -->) aufgetreten
- `void handleText(char[] data, int pos)`  
Markierter Text aufgetreten

# Beispiel: Umformatierung von HTML

```
class HTMLStripper extends HTMLToolkit.ParserCallback {
 public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
 if ((t==HTML.Tag.P) ||
 (t==HTML.Tag.H1) || (t==HTML.Tag.H2) || (t==HTML.Tag.H3) ||
 (t==HTML.Tag.H4) || (t==HTML.Tag.H5) || (t==HTML.Tag.H6)) {
 System.out.println(); // Leerzeile als neuer Absatz
 System.out.println();
 }
 if ((t==HTML.Tag.B) || (t==HTML.Tag.STRONG) || (t==HTML.Tag.EM)) {
 System.out.print("**"); // Hervorhebung
 }
 }
 public void handleEndTag(HTML.Tag t, int pos) {
 if ((t==HTML.Tag.B) || (t==HTML.Tag.STRONG) || (t==HTML.Tag.EM)) {
 System.out.print("**"); // Hervorhebung
 }
 }
}
```

## Beispiel: Umformatierung von HTML

```
public void handleSimpleTag(HTML.Tag t,
 MutableAttributeSet a, int pos) {
 if (t==HTML.Tag.BR) {
 System.out.println(); // Neue Zeile
 System.out.println();
 }
}

public void handleText(char[] data, int pos) {
 System.out.print(data); // Eigentlichen Text unverändert
}
```

# Beispiel: Umformatierung von HTML

>java HTMLStripper <http://www.inf.fu-berlin.de>  
FU-Berlin: Institut für Informatik

Freie Universität Berlin

[...]

Überblick

**\*\*Arbeitsgruppen\*\*** - Auf diesen dezentral verwalteten Seiten finden Sie Themen bezogene Informationen zu Forschung & Lehre.

**\*\*Kontakt\*\*** - Geschäftsführung, Anschrift, Telefon, Standort

**\*\*Service\*\*** - Stellenausschreibungen, Sekretariate, Rechnerbetrieb, Bibliothek, Zentrum für Digitale Medien, Technical Reports, Prüfungsberatung

**\*\*Gremien\*\*** - Studierende, Frauenbeauftragte, Prüfungsausschuss

**\*\*Leute\*\*** - Private Homepages

Studium

**\*\*Lehre\*\*** - Studiengänge, Kommentiertes Vorlesungsverzeichnis, Allgemeines zu den Informatik-Lehrveranstaltungen, aktuelle und frühere Lehrveranstaltungen, Praktika

- static HTML.Tag A
- static HTML.Tag ADDRESS
- static HTML.Tag APPLET
- static HTML.Tag AREA
- static HTML.Tag B
- static HTML.Tag BASE
- static HTML.Tag BASEFONT
- static HTML.Tag BIG
- static HTML.Tag BLOCKQUOTE
- static HTML.Tag BODY
- static HTML.Tag BR
- static HTML.Tag CAPTION
- static HTML.Tag CENTER
- static HTML.Tag CITE
- static HTML.Tag CODE
- ...

# Beispiel: Testen von Attributen

```
class ImgAltTest extends HTMLEditorKit.ParserCallback {

 public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
 if (t==HTML.Tag.IMG) {
 Object attr = a.getAttribute(HTML.Attribute.ALT);
 if (attr==null) {
 System.out.println("** alt Attribut bei img fehlt bei "+
 pos+" für "+a.getAttribute(HTML.Attribute.SRC));
 System.out.println("Vorhandene Attribute: ");
 for (Enumeration e = a.getAttributeNames();
 e.hasMoreElements();) {
 Object at=e.nextElement();
 System.out.print(at+" mit Wert ");
 System.out.println(a.getAttribute(at));
 }
 }
 }
 }
}
```

## Beispiel: Testen von Attributen

```
>java ImgAltTest http://www.robert-tolksdorf.de
```

```
** alt Attribut bei img fehlt bei 14019 für
```

```
 http://m1.nedstatbasic.net/n?id=ACJn3g5wJP8TxLTpbkIYQs7VYRcA
```

Vorhandene Attribute:

nosave mit Wert #DEFAULT

border mit Wert 0

height mit Wert 18

width mit Wert 18

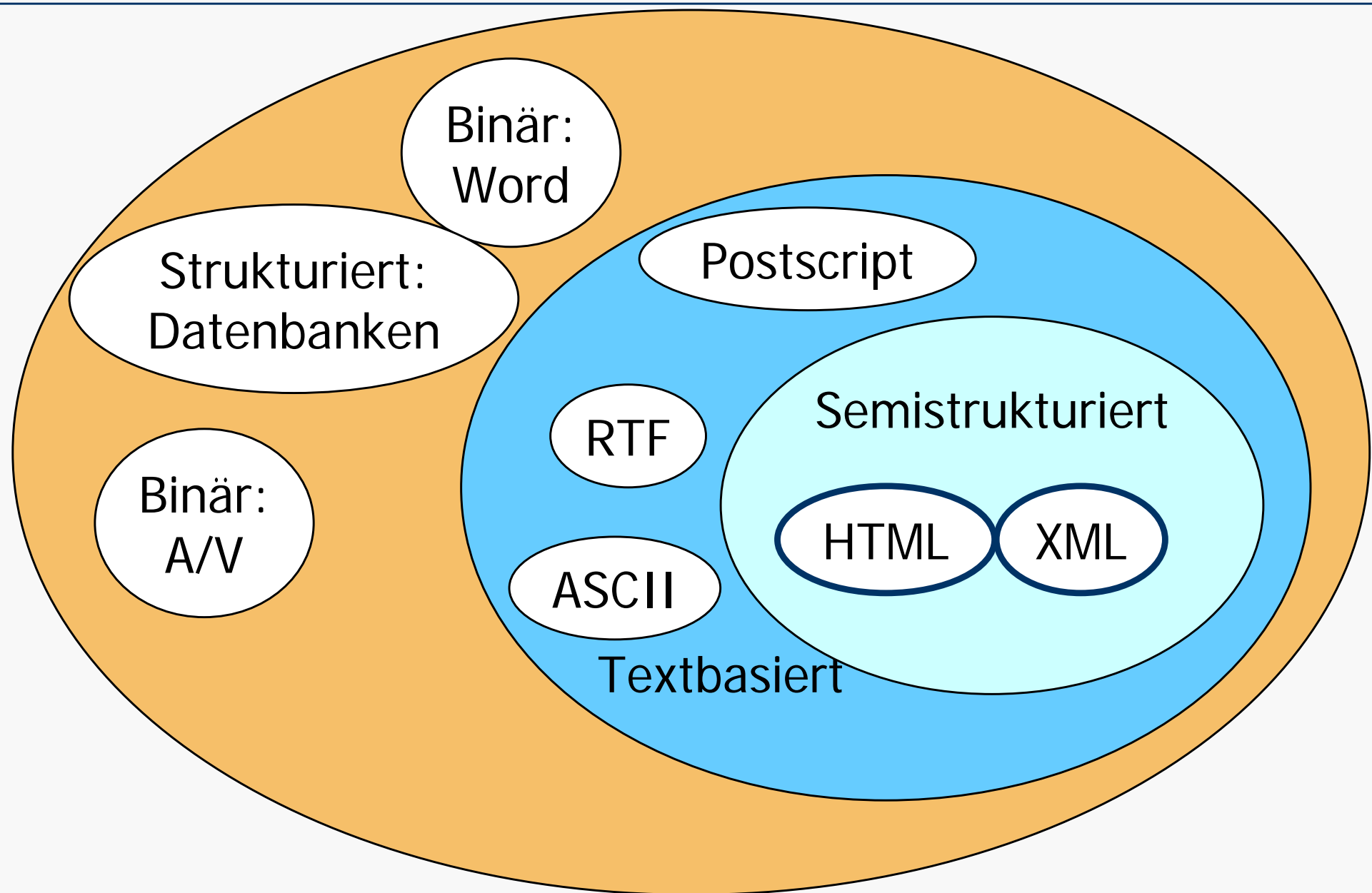
src mit Wert

```
 http://m1.nedstatbasic.net/n?id=ACJn3g5wJP8TxLTpbkIYQs7VYRcA
```

- static HTML.Attribute ACTION
- static HTML.Attribute ALIGN
- static HTML.Attribute ALINK
- static HTML.Attribute ALT
- static HTML.Attribute ARCHIVE
- static HTML.Attribute BACKGROUND
- static HTML.Attribute BGCOLOR
- static HTML.Attribute BORDER
- static HTML.Attribute CELLPADDING
- static HTML.Attribute CELLSPACING
- static HTML.Attribute CHECKED
- static HTML.Attribute CLASS
- static HTML.Attribute CLASSID
- static HTML.Attribute CLEAR
- static HTML.Attribute CODE
- static HTML.Attribute CODEBASE
- static HTML.Attribute CODETYPE
- static HTML.Attribute COLOR
- static HTML.Attribute COLS
- ...



## Daten im Netz



# Auszeichnungssprachen

- Auszeichnungssprachen fügen *Markierungen* zu einem Text hinzu
- Beispiel HTML:

```
<u>Robert Tolksdorf</u>
<address>
FU Berlin

Netzbasierte
Informationssysteme

Takustr.9

D-14195 Berlin

</address>
```

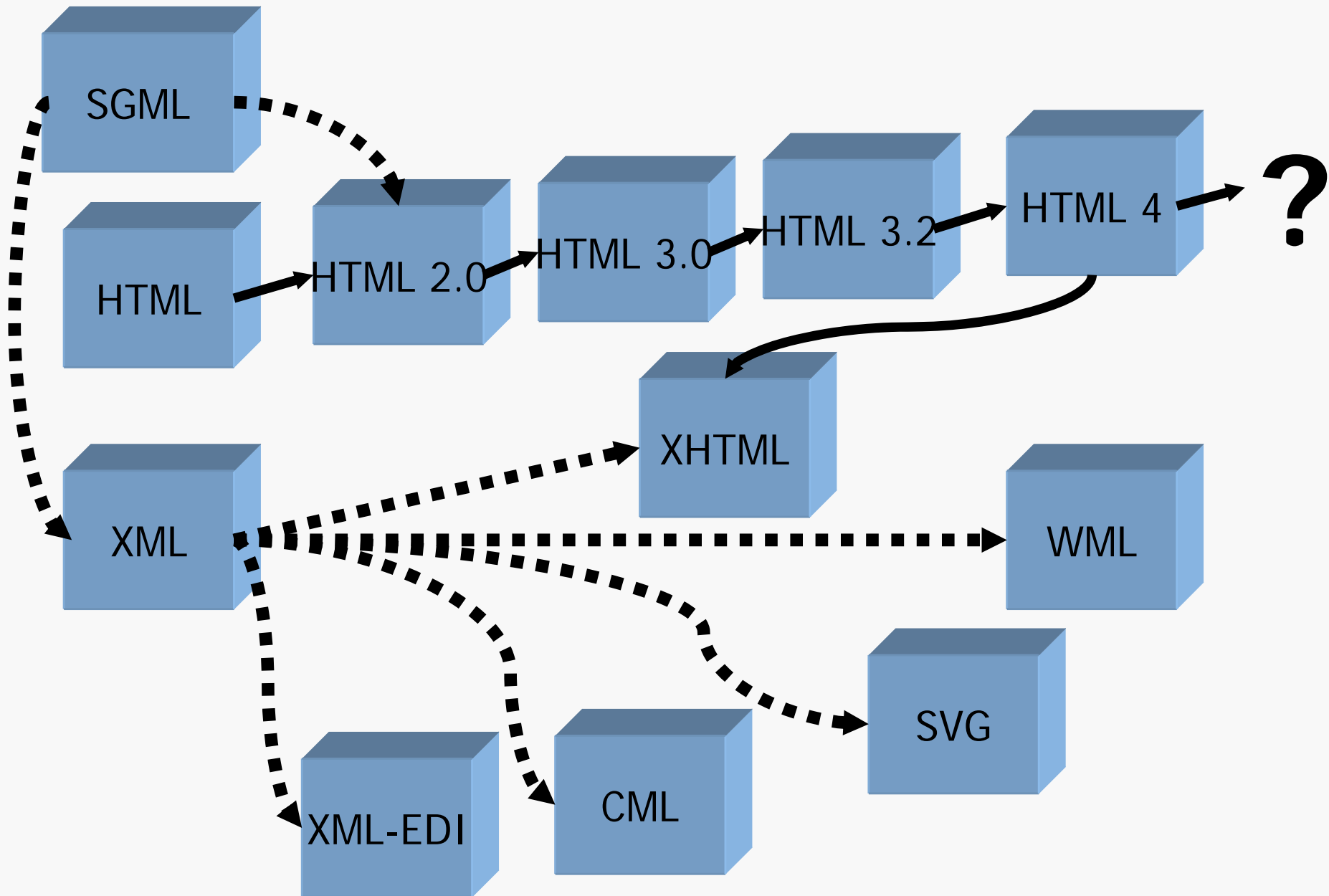
Robert Tolksdorf  
*FU Berlin*  
*Netzbasierte Informati*  
*Takustr.9*  
*D-14195 Berlin*

- *Tags* haben *logische* oder *visuelle* Bedeutung

# Auszeichnungssprachen

- Kann es eine universelle Auszeichnungssprache geben?
  - Alle visuellen und sonstigen Möglichkeiten aller Ausgabegeräte müssten durch Tags steuerbar sein
  - Alle semantischen Konzepte aller Domänen müssten durch Tags repräsentierbar sein
  - Alle notwendigen Granularitäten der Auszeichnung müssten unterstützt werden:
    - `<ADRESSE> . . . </ADRESSE>`
    - `<ADRESSE><STRASSE> . . . </STRASSE><ORT> . . . </ORT>`  
`</ADRESSE>`
    - `<ADRESSE>`  
    `<STRASSE> . . . </STRASSE>`  
    `<ORT><PLZ> . . . </PLZ><ORTSNAME> . . . </ORTSNAME></ORT>`  
`</ADRESSE>`
- Nein: Anwendungsspezifische Auszeichnung nötig

# XML als Ergebnis der HTML Entwicklung



## XML Q&A

- *Was ist XML?*

Die Extensible Markup Language ist die Definition einer Untermenge von SGML, mit der man einfach Auszeichnungssprachen definieren kann

- *Woher kommt XML?*

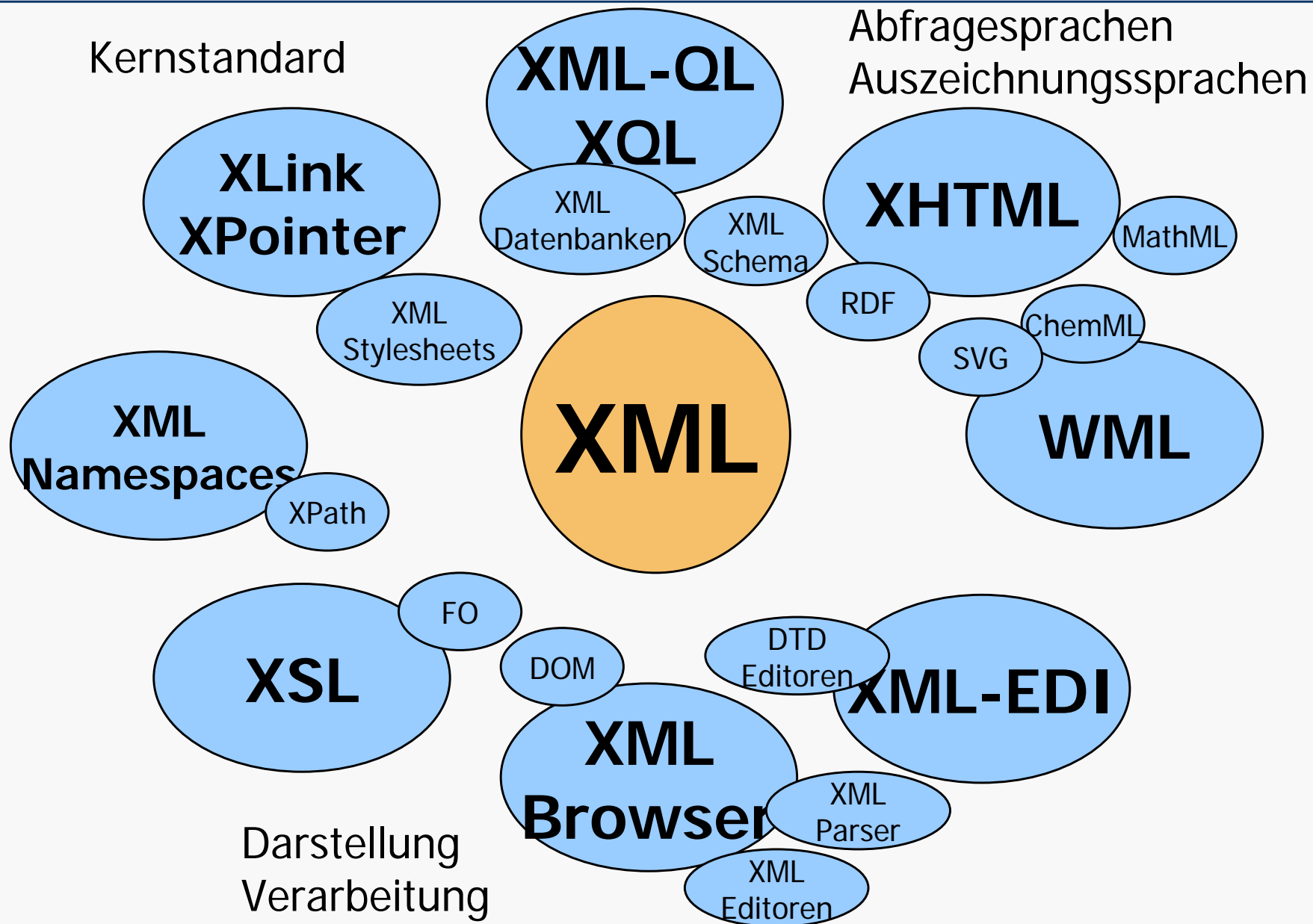
XML ist ein Standard des World Wide Web Konsortiums W3C

- *Was macht man mit XML?*

Anwendungsspezifische Auszeichnungssprachen definieren und standardisieren

- *Was ist der Vorteil von XML-basierten Auszeichnungssprachen?*

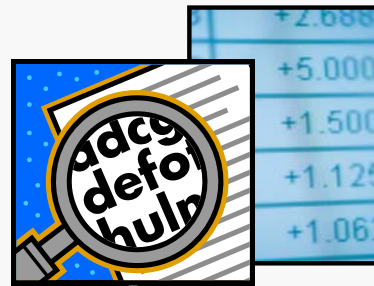
Standardisierung ermöglicht Datenaustausch



# Wohlgeformte XML-Dokumente

# Was ist ein XML-Dokument?

Inhalt: Text oder Daten



kodiert als

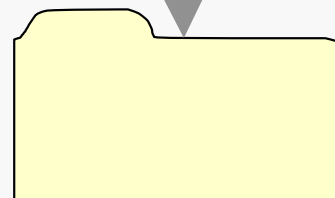
XML-Dokument

```
<name>
 <first>John</first>
 <last>Doe</last>
</name>
```

Objekt, das Syntaxregeln von XML entspricht (wohlgeformt ist)

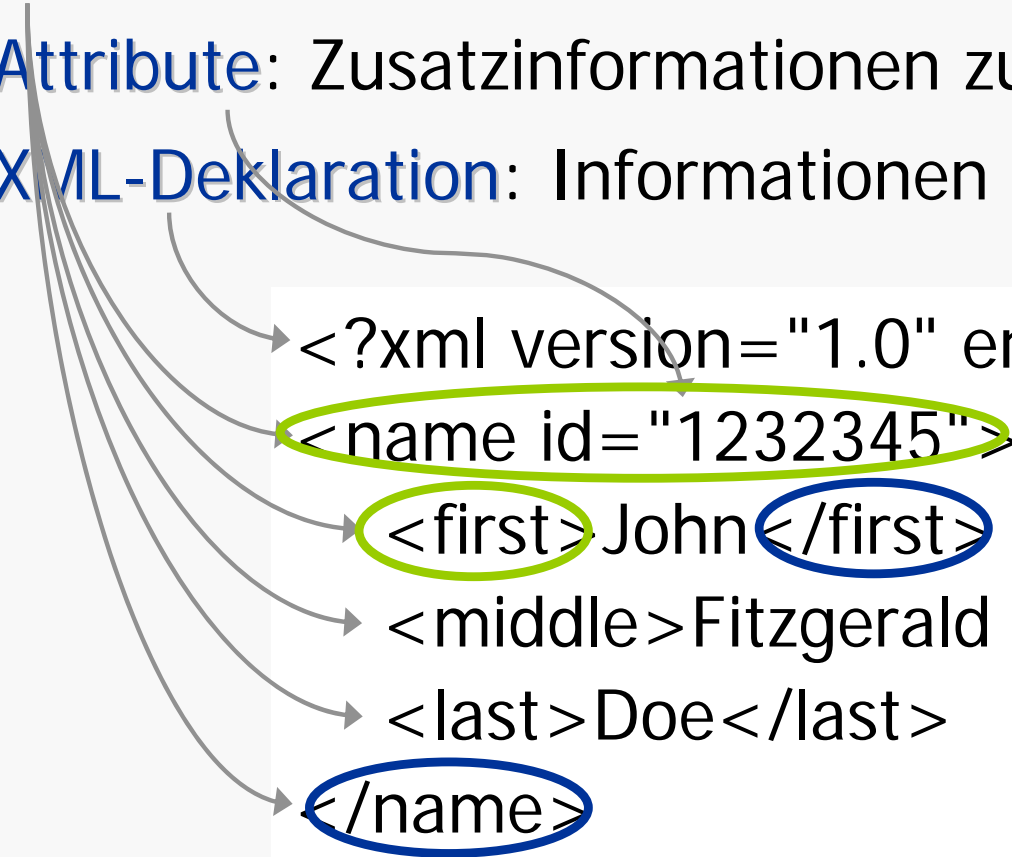
gespeichert in

XML-Datei



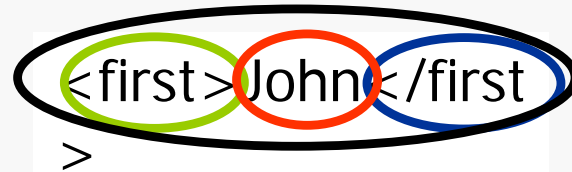
- **Elemente**: strukturieren das XML-Dokument
- **Attribute**: Zusatzinformationen zu Elementen
- **XML-Deklaration**: Informationen für Parser

```
<?xml version="1.0" encoding="UTF-8"?>
<name id="1232345">
 <first>John</first>
 <middle>Fitzgerald Johansen</middle>
 <last>Doe</last>
</name>
```



- **Namensräume**: lösen Namenskonflikte auf und geben Elemente eine Bedeutung

- Beispiel:



```
<first>John</first>
```

- besteht aus:

- einem **Anfangs-Tag** (engl. *start tag*): hier <first>
- einem dazugehörigen **Ende-Tag** (engl. *end tag*): hier </first>
- einem **Inhalt**: hier „John“

- alles zusammen bildet ein **Element**

- haben einen **Namen**: hier „first“

# Inhalt von Elementen

1. **unstrukturierter Inhalt:**  
einfacher Text ohne Kind-Elemente
2. **strukturierter Inhalt:**  
Sequenz von  $> \emptyset$  Kind-Elementen
3. **gemischter Inhalt:**  
enthält Text mit mind. einem Kind-Element
4. **leerer Inhalt**

# 1. Unstrukturierter Inhalt

- Beispiel: `<first>John</first>`
- einfacher Text ohne Kind-Elemente  
**Kind-Element**: Element, das im Inhalt eines Elementes vorkommt
- unstrukturierter Inhalt auch als **Parsed Character Data (PCDATA)** bezeichnet:
  - **character data**: einfache Zeichenkette
  - **parsed**: Zeichenkette wird vom Parser analysiert, um Ende-Tag zu identifizieren.
  - Normalisierung: u.a. Zeilenumbruch (CR+LF) → #xA

Anmerkung: Auf den Folien schreibe ich der besseren Lesbarkeit wegen *Kind-Element* statt *Kindelement* !

- Reservierte Symbole < und & in PCDATA nicht erlaubt.
- Symbole wie >, /, (, ), {, }, [, ], % allerdings erlaubt
- statt < und & **Entity References** &amp; bzw. &lt; benutzen
- Entity References in XML:

&amp;      ⇒      &

&lt;        ⇒      <

&gt;        ⇒      >

&apos;     ⇒      '      

&quot;     ⇒      "

- Unstrukturierten Inhalt mit vielen reservierten Symbolen besser als **Character Data (CDATA)** darstellen.

- Beispiel:

```
<formula>
 <![CDATA[X < Y & Y < Z]]>
</formula>
```

- Inhalt: String zwischen inneren Klammern [ ]  
hier: X < Y & Y < Z
- XML-Parser sucht in CDATA lediglich ]]>, analysiert den Inhalt aber ansonsten nicht.
- "]]>" als Inhalt von CDATA nicht erlaubt

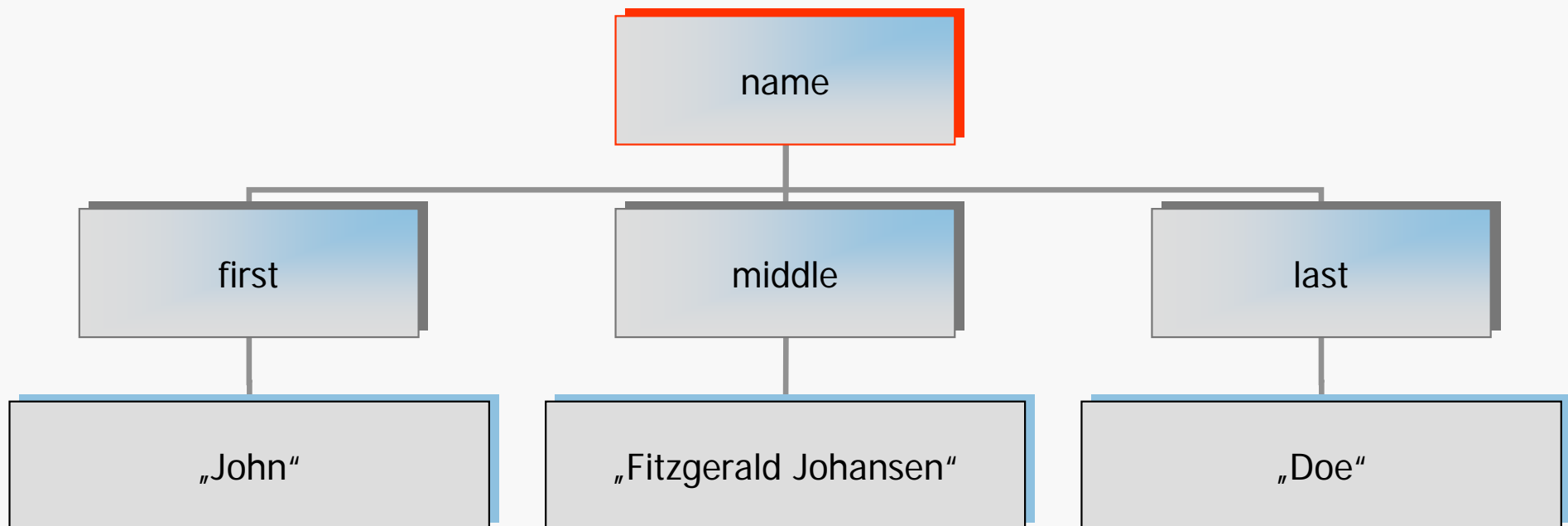
## 2. Strukturierter Inhalt

- Beispiel:

```
<name>
 <first>John</first>
 <last>Doe</last>
</name>
```

- Sequenz von  $> 0$  Kind-Elementen:  
hier: `<first>John</first>` und `<last>Doe</last>`
- kein Text vor, nach oder zwischen den Kind-Elementen
- Kind-Elemente immer geordnet:  
Reihenfolge, so wie sie im XML-Dokument erscheinen
- Elemente können beliebig tief geschachtelt werden.

```
<name>
 <first>John</first>
 <middle>Fitzgerald Johansen </middle>
 <last>Doe</last>
</name>
```



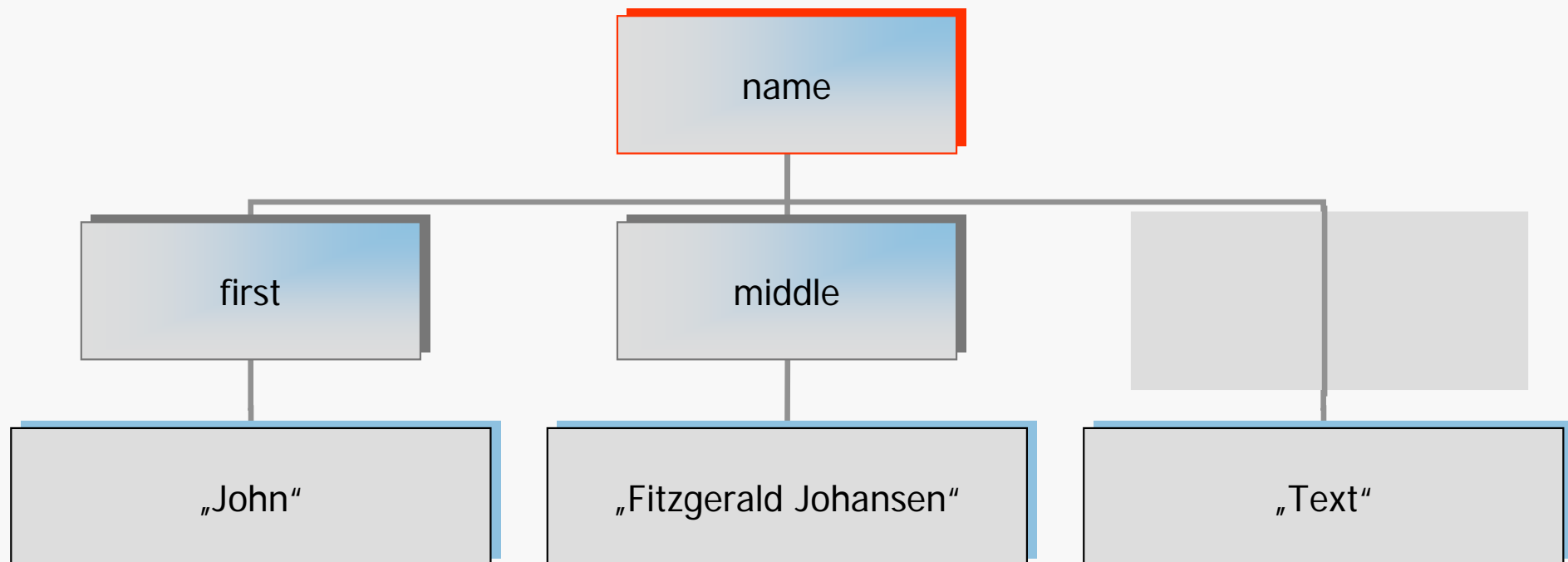
### 3. Gemischter Inhalt

- Englisch: **mixed content**
- enthält Text mit mindestens einem Kind-Element
- Beispiel:

```
<section>
Text
 <subsection> ... </subsection>
Text
</section>
```

- Wie unterscheidet ein Parser strukturierten und gemischten Inhalt, wenn Text = leerer String?
- Antwort: Nur mit zugehöriger DTD oder XML-Schema möglich!

```
<name>
 <first>John</first>
 <middle>Fitzgerald Johansen </middle>
 Text
</name>
```



## 4. Leerer Inhalt

- Beispiel:

```
<name>
 <first>John</first>
 <middle></middle>
 <last>Doe</last>
</name>
```

- weder Text noch Kind-Element
- <middle></middle> auch **leeres Element** genannt
- Abkürzung: **selbstschießendes Tag** (engl.: *self-closing tag*)  
<middle/> :

```
<name>
 <first>John</first>
 <middle/>
 <last>Doe</last>
</name>
```

# Warum leere Elemente?

```
<name>
 <first>John</first>
 <last>Doe</last>
</name>
```

VS.

```
<name>
 <first>John</first>
 <middle/>
 <last>Doe</last>
</name>
```

- leeres Element evtl. von DTD oder XML-Schema vorgeschrieben
- einfacher später mit Inhalt zu füllen
- leeres Element kann Attribute haben:  
`<middle status="unknown"></middle>` oder  
`<middle status="unknown"/>`

```
<name id="1232345" nickname="Shiny John">
 <first>John</first>
 <last>Doe</last>
</name>
```

- **Attribut:** Name-Wert-Paar
  - name="wert" oder name='wert' ~~aber name= "wert"~~
- **Attribut-Wert:**
  - immer PCDATA: keine Kind-Elemente, kein < und &
  - "we"rt" und 'we'rt' ebenfalls nicht erlaubt
  - Normalisierung: u.a. Zeilenumbruch → #xA
- Beachte: Reihenfolge der Attribute belanglos

- Jedes Attribut auch als Kind-Element darstellbar:

```
<name id="12345">
 <first>John</first>
 <middle>Fitzgerald</middle>
 <last>Doe</last>
</name>
```



```
<name>
 <id>12345</id>
 <first>John</first>
 <middle>Fitzgerald</middle>
 <last>Doe</last>
</name>
```

id als Attribut

id als Kind-Element

- Jedes Kind-Element mit **unstrukturiertem** Inhalt auch als Attribut darstellbar:

```
<name>
 <id>12345</id>
 <first>John</first>
 <middle>Fitzgerald</middle>
 <last>Doe</last>
</name>
```

id, first, middle und last  
als Kind-Elemente



```
<name id="12345"
 first="John"
 middle="Fitzgerald"
 last="Doe" />
```

id, first, middle und last  
als Attribute

Resultat: leeres Element

- Attribut kann nur einfachen String (PCDATA) als Wert haben, Element kann beliebig strukturiert sein
- `<![CDATA[ ... ]]>` in Element-Inhalten erlaubt, nicht aber in Attribut-Werten
- Reihenfolge der Attribute belanglos, diejenige von Elementen nicht
- einheitliche Darstellung mit Elementen eleganter, Darstellung mit Attributen kompakter

Fazit: Attribute für einfache, unstrukturierte Zusatzinformationen (Metadaten) geeignet.

```
<name creation-date="21.05.2003">
 <first>John</first>
 <middle>Fitzgerald Johansen</middle>
 <last>Doe</last>
</name>
```

- Erstellungsdatum creation-date ist Zusatzinformation
- falls noch andere Attribute vorhanden: Reihenfolge egal
- ⇒ Repräsentation als Attribut
- Nachteil: Datum "21.05.2003" unstrukturiert

- **xml:lang**
  - Sprache des Inhalts
  - Beispiel: `<p xml:lang="de">Übung 1</p>`
- **xml:space**
  - Leerräume im Inhalt
  - Beispiel: `<p xml:space="default">Übung 1</p>`

```
<?xml version="1.0" encoding="UTF-8"?>
<name id="1232345">
 <first>John</first>
 <middle>Fitzgerald Johansen</middle>
 <last>Doe</last>
</name>
```

- enthält Informationen für Parser: z.B. verwendete XML-Version und Kodierung
- wenn vorhanden, dann immer am Anfang der Datei
- in XML 1.0 optional, in XML 1.1 obligatorisch

## Attribut version

```
<?xml version="1.0" encoding="UTF-8"?>
```

- verwendete XML-Version: "1.0" oder "1.1"
- obligatorisch

## Attribut encoding

- Kodierung der XML-Datei
- optional

## Attribut standalone

- Gibt an, ob es eine zugehörige DTD oder ein XML-Schema gibt ("no") oder nicht ("yes").
- optional

Beachte: immer in dieser Reihenfolge!

XML-Dokument

```
<name>
<first>John</first>
<last>Doe</last>
</name>
```

gespeichert in

XML-Datei

Unicode  
(UTF-8)

windows-  
1252

...

## XML-Parser

- müssen intern mit Unicode (UTF-8 oder UTF-16) arbeiten

## Unicode

- kann alle nationalen Zeichen darstellen: insgesamt ca. 65.000 Zeichen

## encoding-Attribut

- Zeichenkodierung der XML-Datei
- Fehlt das Attribut, dann wird Kodierung in Unicode angenommen.
- Beachte: XML-Parser müssen nur Unicode verarbeiten können!

## Kommentare

- `<!-- Kommentar -->`
- `--` in Kommentaren nicht erlaubt

## Prozessorinstruktionen

- Beispiel: `<?mysql SELECT * FROM PO?>`
- selten benutzt

1. Jedes Anfangs-Tag muss ein zugehöriges Ende-Tag haben.
2. Elemente dürfen sich nicht überlappen.
3. XML-Dokumente dürfen nicht verschachtelt sein.  
Wie kann aus den Elementen ein Element gebildet werden?
4. Element-Namensräume sind erforderlich.  
Grundbausteinen ein wohlgeformtes XML-Dokument gebildet werden?
5. XML beachtet die Zeichenkodierung.  
schreibung.
6. XML belässt White Space im Text.
7. Ein Element darf niemals zwei Attribute mit dem selben Namen haben.

Jedes Anfangs-Tag muss zugehöriges Ende-Tag haben.

- In HTML gilt diese Regel nicht:

```
<HTML>
```

```
<BODY>
```

```
<P>Text
```

```

More text in the same paragraph.
```

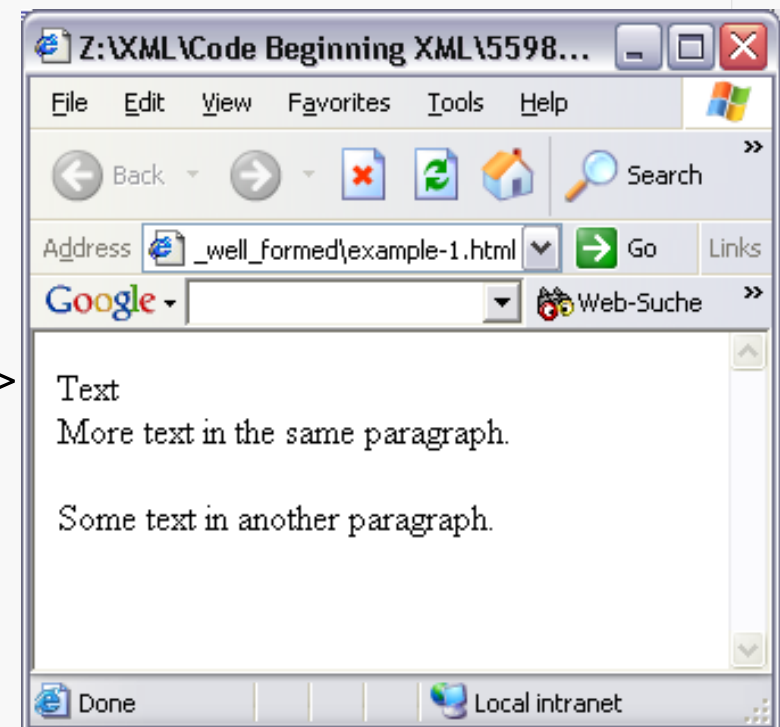
```
<P>Some text in another paragraph.</P>
```

```
</BODY>
```

```
</HTML>
```

- Wo endet das erste P-Element?

⇒ HTML mehrdeutig



# Regel 2: Überlappung von Elementen

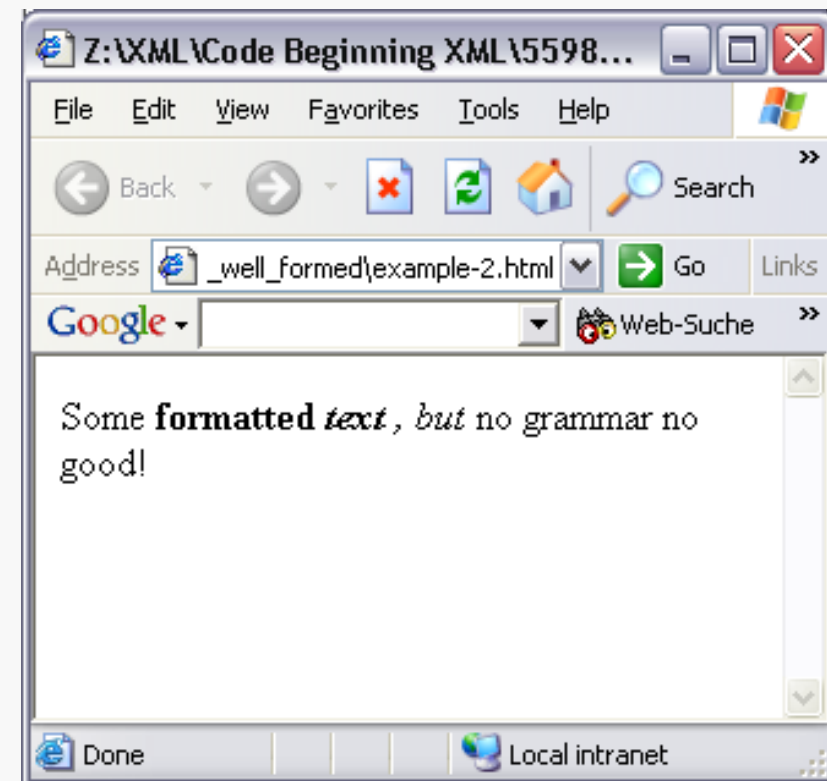
Elemente dürfen sich nicht überlappen.

- In HTML gilt diese Regel nicht:

```
<HTML>
<BODY>
 <P>Some
 formatted
 text
 , but

 no grammar no good!
</p>
</BODY>
</HTML>
```

⇒ HTML unstrukturiert



Jedes XML-Dokument hat genau ein Wurzel-Element.

- Also z.B. statt zweier Wurzel-Elemente

```
<?xml version="1.0"?>
<name>John</name>
<name>Jane</name>
```

zusätzliches Eltern-Element einführen:

```
<?xml version="1.0"?>
<names>
 <name>John</name>
 <name>Jane</name>
</names>
```

## Element- und Attribut-Namen:

- beginnen entweder mit Buchstaben oder `_` aber nie mit Zahlen:  
z.B. `first`, `First` oder `_First`
- nach erstem Zeichen zusätzlich Zahlen sowie `-` und `.` erlaubt:  
z.B. `_1st-name` oder `_1st.name`
- enthalten **keine Leerzeichen**
- enthalten **kein Doppelpunkt**
- beginnen **nicht** mit `xml`, unabhängig davon, ob die einzelnen Buchstaben groß- oder kleingeschrieben sind

- `<résumé>` ✓
- `<xml-tag>` nicht korrekt: beginnt mit „xml“
- `<123>` nicht korrekt: beginnt mit Zahl
- `<fun=xml>` nicht korrekt: enthält „=“  
erlaubt wären: `_`, `-` und `.`
- `<first name>` nicht korrekt: enthält Leerzeichen

## XML beachtet Groß- und Kleinschreibung.

- Im Gegensatz zu HTML unterscheidet XML also z.B. zwischen `<P>` und `<p>`.

Dennoch möglichst nicht gleichzeitig  
`<First>` und `<first>` verwenden!

Hinweis: eine Schreibweise im gesamten  
Dokument verwenden z.B. `<FirstName>`

## XML belässt White Space im Text.

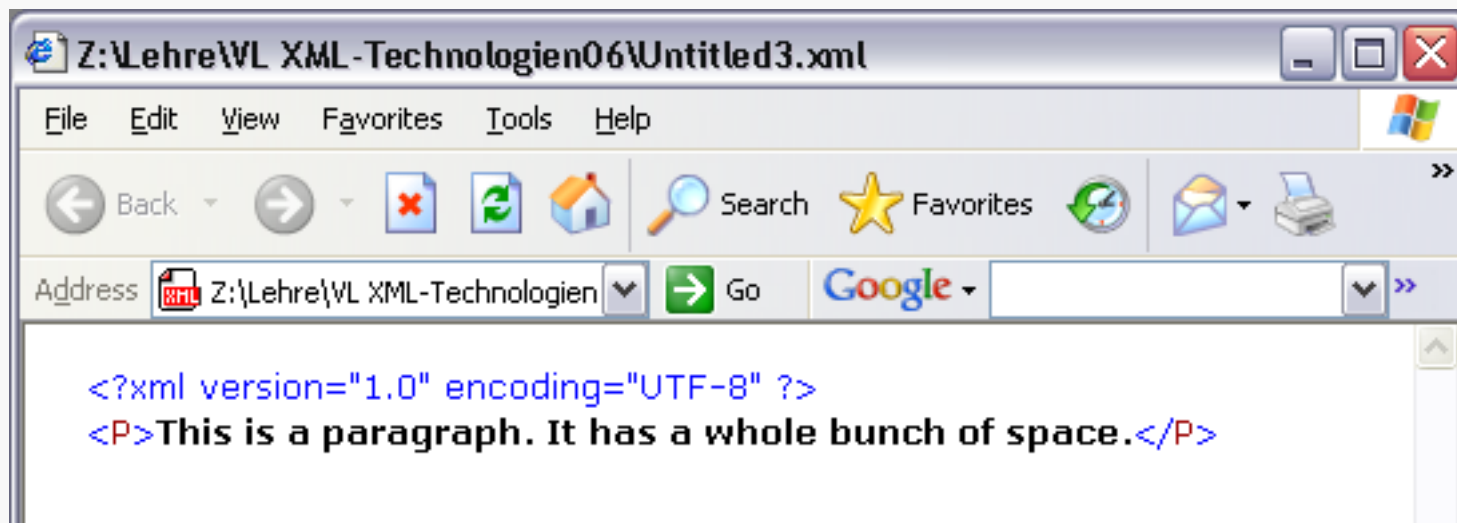
- Beispiel:

```
<?xml version="1.0" encoding="UTF-8" >
<P>This is a paragraph. It has a whole
 bunch
 of space.</P>
```

- Inhalt des P-Elementes:

```
This is a paragraph. It has a whole
 bunch
 of space.
```

- Beachte: Von Browsern wird White Space allerdings nicht angezeigt:



- Grund:
  - XML-Dokumente werden zur Darstellung im Browser in HTML umgewandelt
  - HTML reduziert White Space auf ein Leerzeichen.



## Zusammenfassung

## 1. HTML

1. Einfache Auszeichnungssprache
2. Elemente (Tags), Attribute, Entitäten
3. Elemente für
  1. Struktur
  2. Gestaltung
  3. Inhaltsstrukturen
  4. Interaktion
  5. Komplexe Inhalte

## 2. HTML Verarbeitung

1. Parser notwendig
2. Swing Paket enthält Callback-Parser
3. Element- und Attributdefinitionen

## 3. XML Dokumente

- Dave Raggett (ed). HTML 4.01 Specification. W3C Recommendation. 24 December 1999  
<http://www.w3.org/TR/html4>
- Robert Tolksdorf. XHTML und HTML - die Sprachen des Web. dpunkt.verlag, Heidelberg, 5. Auflage, 2002. ISBN 3-89864-155-4.
- Stefan Münz. SELFHTML. <http://www.selfhtml.org/>
- Hunter, David; Cagle, Kurt; Dix, Chris: Beginning XML XML Schemas, SOAP, XSLT, DOM, and SAX 2.0 2nd ed. 2001. Wrox Press 1-86100-559-8
- Empfehlenswertes Skript einer anderen XML-Vorlesung:  
<http://www.jeckle.de/vorlesung/xml/>
- mehrere Interaktive XML-Kurse <http://www.zvon.org>