Freie Universität Berlin

Rückblick – Klausurvorbereitung

Markus Luczak-Rösch
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
markus.luczak-roesch@fu-berlin.de

XML

- Extensible Markup Language
- generische Auszeichnungssprache (*generalized markup language*)
  - *keine* Tags vorgegeben, beliebige Tags erlaubt
  - Vorteil: beliebige Metainformationen darstellbar
  - Nachteil: Bedeutung der Metainformationen (Tags) offen
  - Beispiele: SGML und XML

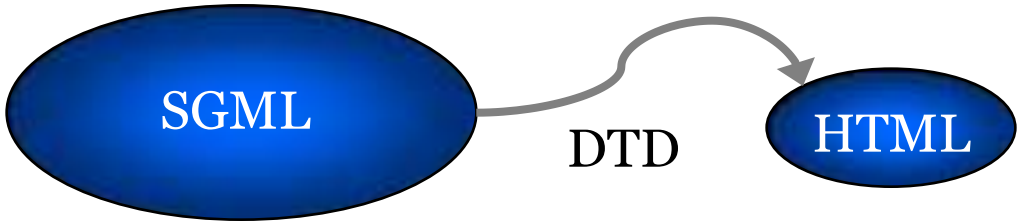
- Unterschiede zu HTML:
  - medienneutral
  - Tag-Namen `<name>…</name>` beliebig

XML ist …

- eine **Methode**, **um strukturierte Daten in einer Textdatei** darzustellen
- Text, aber nicht zum Lesen
- eine **Familie von Technologien**
- **lizenzfrei** und **plattformunabhängig**
- ein **offener Standard**, der sich weit verbreitet hat
- ein Protokoll zur Aufnahme und Verwaltung von Informationen
- eine Philosophie für den Umgang mit Informationen
- ein **Werkzeug für die Speicherung von Dokumenten**
- ein **konfigurierbares Medium**
- neu, aber nicht so neu

SGML

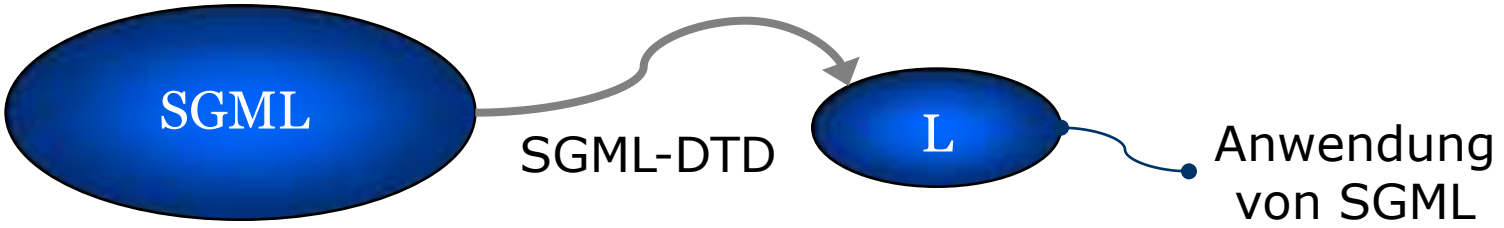
- Standard Generalized Markup Language
- *keine* vorgegebenen Tags, auch keine für das Layout von Dokumenten
- Vorgänger von XML
- Anwendungen von SGML → mit Document Type Definitions (DTDs) können spezielle Auszeichnungssprachen mit konkreten Tags definiert werden:
  - bekannteste Anwendung von SGML: HTML



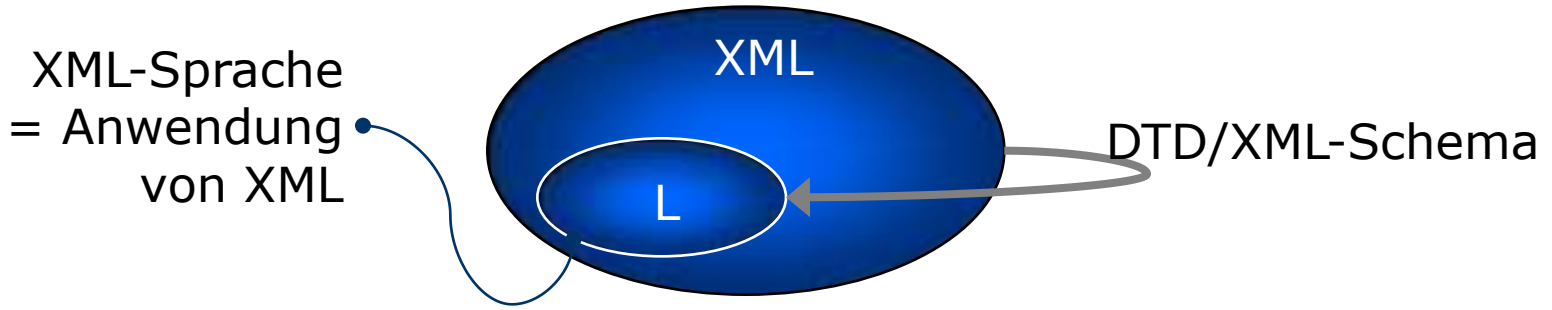
Anwendung: spezieller Wortschatz und Grammatik

- Anwendung selbst kann keine Anwendung definieren

SGML- vs. XML-Anwendungen

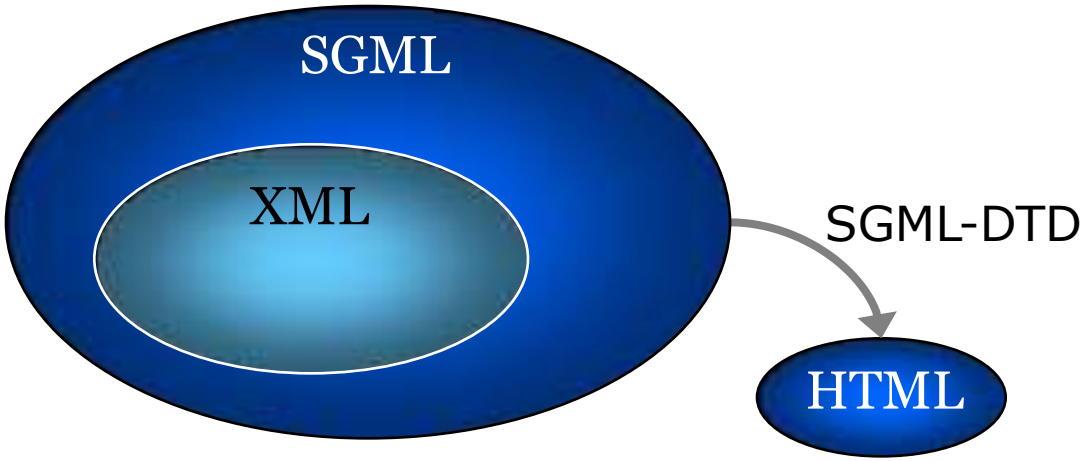


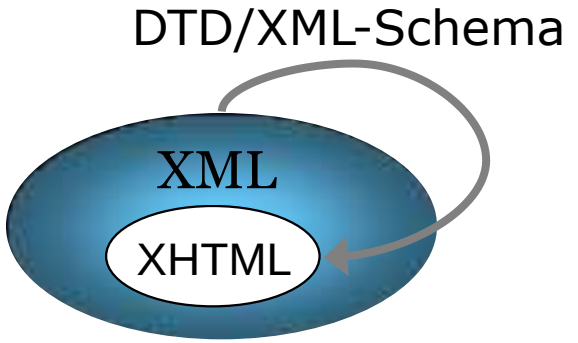
- L muss *nicht* Teilsprache von SGML sein.
- L kann *keine* neue Sprache definieren.
- Beispiel: HTML



- L immer Teilsprache von XML
- L kann *keine* neue Sprache definieren.
- Beispiel: XHTML

SGML, HTML, XML, XHTML?!





HTML

- Anwendung von SGML

XML

- Teilsprache von SGML

XHTML

- XML-Sprache = Anwendung von XML
- alle XHTML-Dokumente immer wohlgeformte XML-Dokumente

Grundbausteine von XML

- **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 Elementen eine Bedeutung

Inhalt von Elementen

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

Grundbausteine: XML-Attribute

Freie Universität Berlin

```
<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

Grundbausteine: XML-Deklaration

```
<?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

XML-Deklaration

Freie Universität Berlin

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

- Attribut: version
  - 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" – default) oder nicht ("yes").
  - Optional

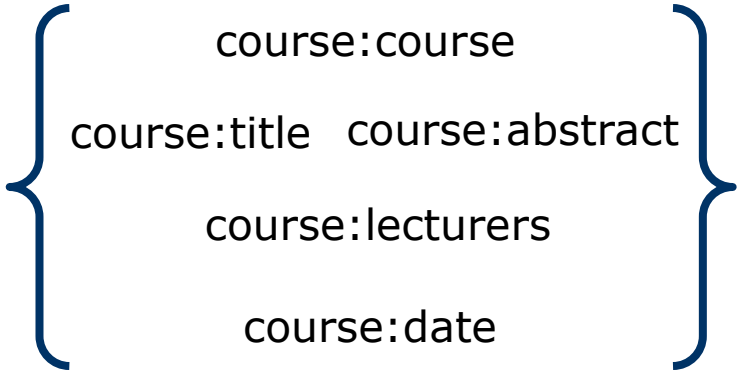
Beachte: immer in dieser Reihenfolge!

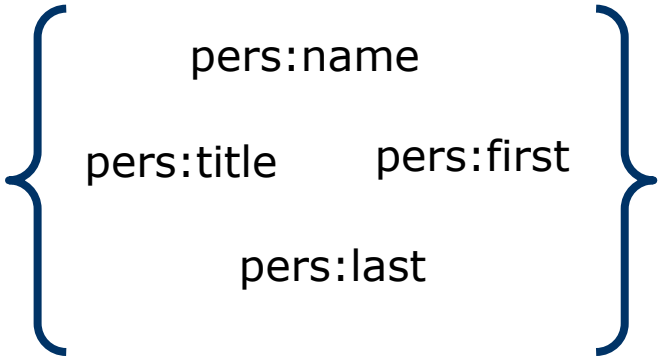
Wohlgeformte XML-Dokumente

Freie Universität Berlin

1. Jedes Anfangs-Tag muss ein zugehöriges Ende-Tag haben.
2. Elemente dürfen sich nicht überlappen.
3. XML-Dokumente haben genau ein Wurzel-Element.
4. Element-Namen müssen bestimmten Namenskonventionen entsprechen.
5. XML beachtet grundsätzlich Groß- und Kleinschreibung.
6. XML belässt White Space im Text.
7. Ein Element darf niemals zwei Attribute mit dem selben Namen haben.
8. Alle Attributwerte stehen in Anführungszeichen

Namensräume





Namensraum (namespace):

- alle Bezeichner mit identischen Anwendungskontext
- Namensräume müssen eindeutig identifizierbar sein.

Namensräume in XML

- WWW: Namensräume müssen global eindeutig sein.
- In XML wird Namensraum mit URI identifiziert.
- Zuerst wird Präfix bestimmter Namensraum zugeordnet, z.B.:

- Anschließend kann das Namensraum-Präfix einem Namen vorangestellt werden: z.B. pers:title
- Beachte: Wahl des Präfixes egal!

Standard-Namensraum

Freie Universität Berlin

- **xmlns="URI"** statt xmlns:prefix="URI"
- Namensraum-Präfix kann weggelassen werden.

- Standard-Namensraum gilt für das Element, wo er definiert ist.
- Kind-Elemente erben Standard-Namensraum von ihrem Eltern-Element.
- Ausnahme: Standard-Namensraum wird überschrieben
- Beachte: **Standardnamensräume gelten nicht für Attribute**

Qualified vs. Unqualified

- Element- oder Attribut-Name heißt **namensraumeingeschränkt** (**qualified**), wenn er einem Namensraum zugeordnet ist.

- Einschränkung vom Element-Namensraum:
  1. Standard-Namensraum festlegen
  2. Namensraum-Präfix voranstellen

- Einschränkung vom Attribut-Namensraum:
  1. Namensraum-Präfix voranstellen

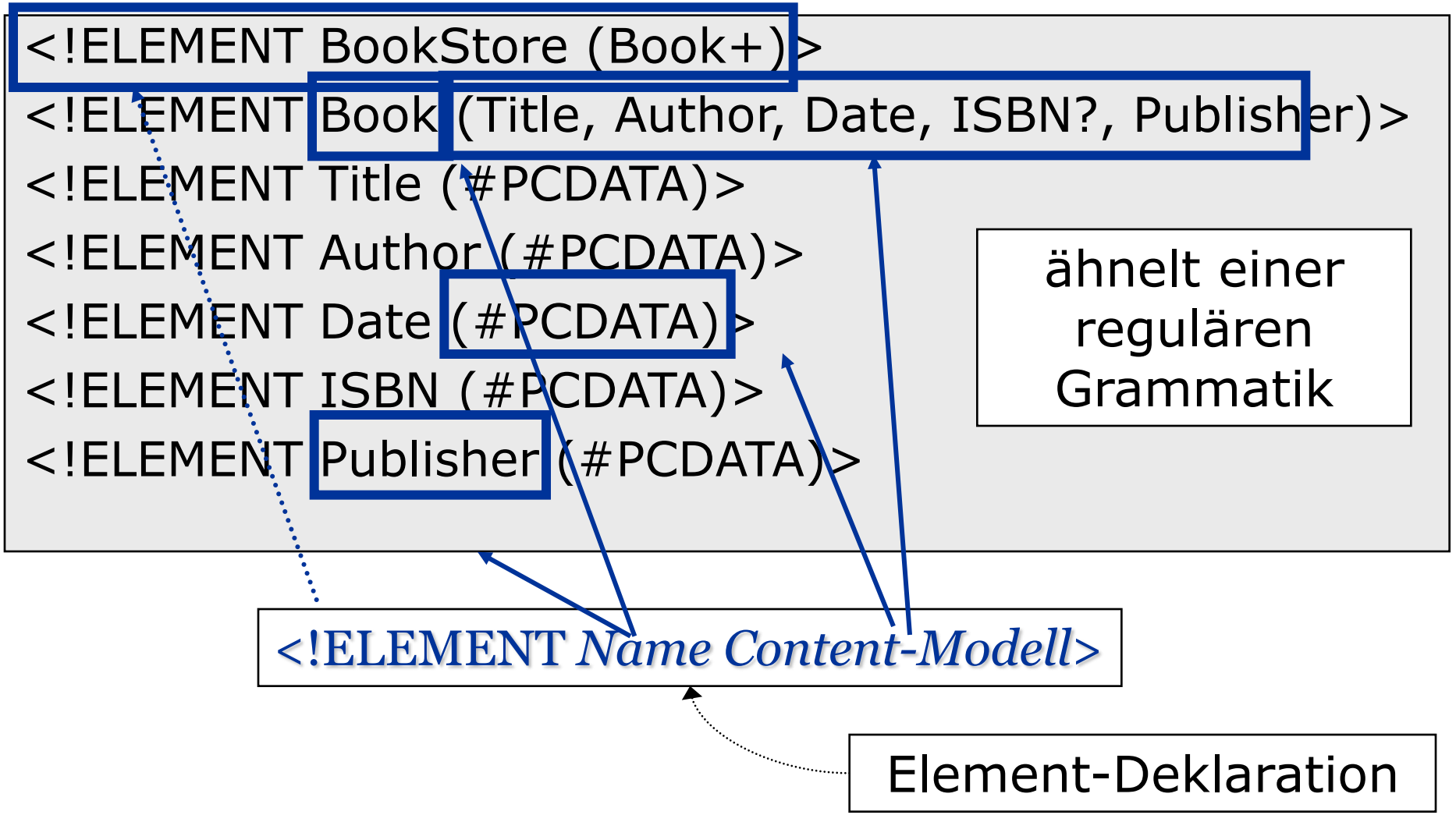
XML → Lernziele

Lernziele

- Was ist eine generische Auszeichnungssprache?
- Unterschiede zwischen SGML, HTML, XML und XHTML
- Grundbausteine von XML
- Syntaxregel & Wohlgeformtheit von XML
- Namensräume
  - Wie werden Elementen/Attributen in XML Namensräume zugeordnet?

DTD

Die DTD für das Beispiel-Dokument



Datentypen für Element-Inhalte

Freie Universität Berlin

verschiedene Datentypen:

1. **Element**: Element mit speziellen Symbolen + * | ?
2. **#PCDATA**: unstrukturierter Inhalt ohne reservierte Symbole < und &.

   <!ELEMENT Title (#PCDATA)>

2. **EMPTY**: leerer Inhalt, Element kann aber Attribute haben

   <!ELEMENT br EMPTY> ➔

3. **ANY**: beliebiger Inhalt (strukturiert, unstrukturiert, gemischt oder leer)

   <!ELEMENT title ANY>

Datentypen wie INTEGER oder FLOAT stehen nicht zur Verfügung.

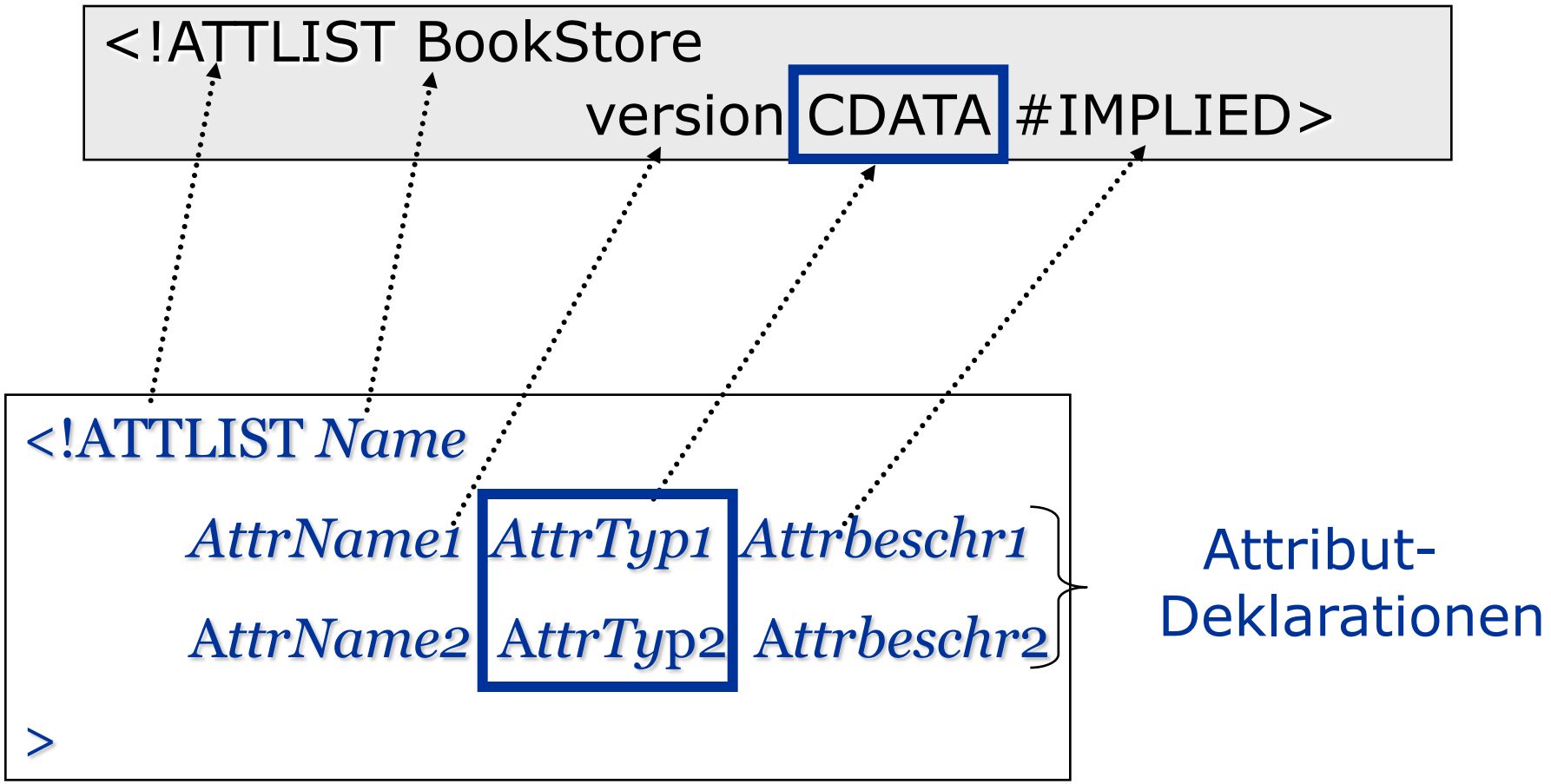
Operatoren

- + bezeichnet n Wiederholungen mit n > 0.
- * bezeichnet n Wiederholungen mit n ≥ 0.
- | bezeichnet Auswahl: genau eine der beiden Alternativen
- , bezeichnet Sequenz von Elementen.
- ? bedeutet optional
- ( ) fassen den Kontent zusammen, auf die sich ein nachfolgender Operator bezieht

Beachte:

- Rekursive Deklaration nicht äquivalent zur iterativen Definition!
- (fast) beliebige Verschachtelung von Sequenz, Auswahl |, ?, *, + und Rekursion erlaubt

Deklaration von Attributen



Deklaration von Attributen

```
<!ATTLIST BookStore
                version CDATA #IMPLIED>
```

```
<BookStore version="1.0">
  ...
</BookStore>
```

- Element **BookStore** hat Attribut **version**.
- CDATA: Attribut-Wert = String ohne <, & und ' bzw. "

```
<!ATTLIST Author
                gender (male | female) "female">
```

- statt CDATA Aufzählungstyp:
- Attribut **gender** hat entweder Wert male oder female.
- **female** ist Standard-Wert von **gender**.

Optionale/erforderliche Attribute


```
<!ATTLIST BookStore
                    version CDATA #FIXED "1.0">
```

- #FIXED: Attribut hat immer den gleichen Wert.
- #IMPLIED: Attribut optional
- #REQUIRED: Attribut obligatorisch

- "1.0": Standard-Wert des Attributes

XML Schema

XML-Schema: Überblick

- eine XML basierte Alternative zu DTD
- beschreibt die Struktur eines XML Dokuments
- eine W3C Recommendation
- statt XML Schema wird oft die Abkürzung XSD (XML Schema Definition) benutzt
- definiert
  - Elemente/Attribute , die im Dokument vorkommen dürfen
  - die Reihenfolge & Anzahl der (Kinder-)Elemente
  - wie Inhalt eines Element auszusehen hat
  - Datentypen für Elemente und Attribute

XML-Schema vs. DTD (Auszug)

Freie Universität Berlin

- Für jede DTD gibt es ein äquivalentes XML-Schema.
- Umgekehrt gibt es jedoch XML-Schemata, für die es keine äquivalente DTD gibt.

➔XML-Schemata ausdrucksmächtiger als DTDs

- XML-Schemata sind wohlgeformte XML-Dokumente.

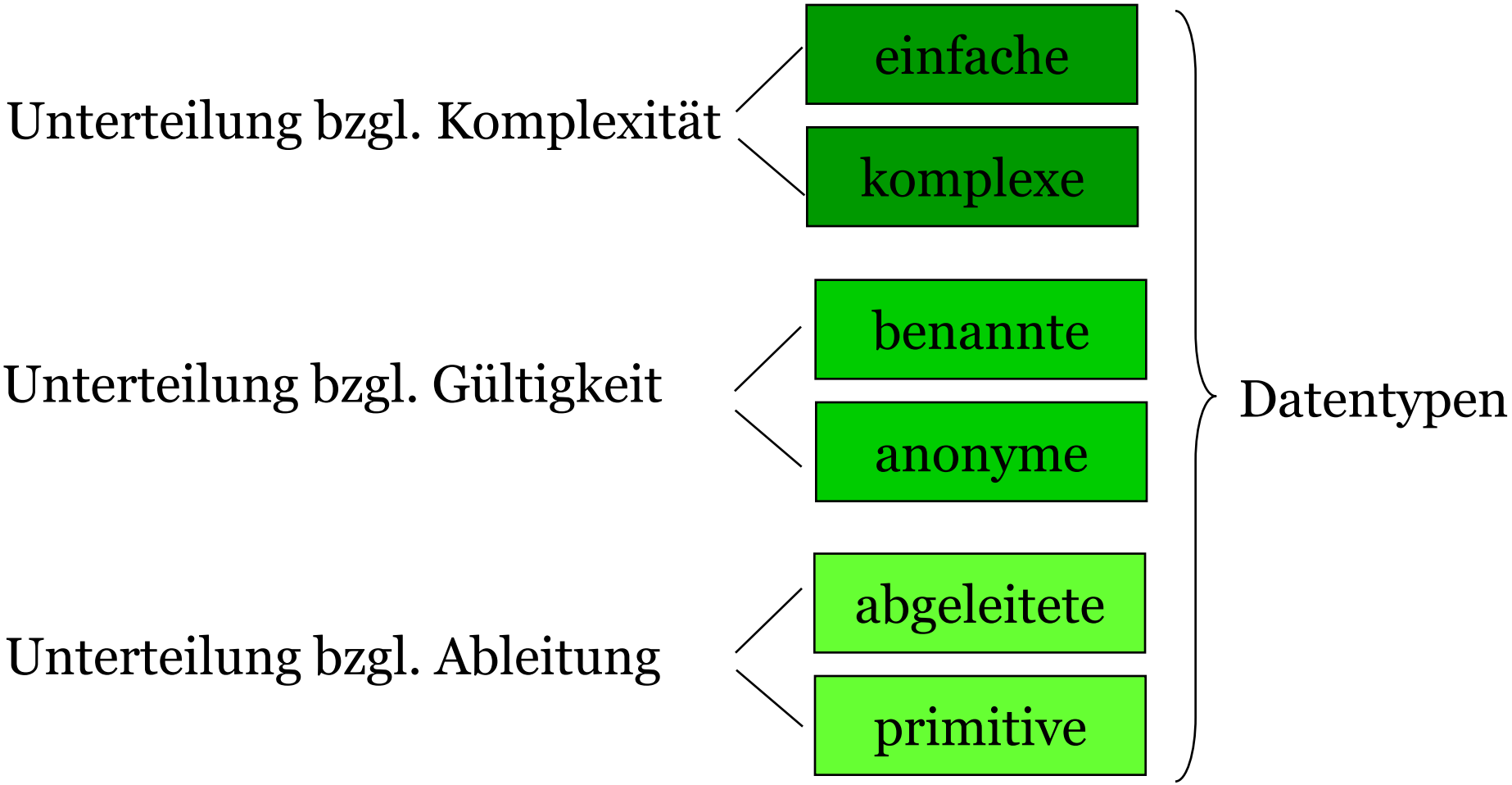
- Wurzel-Element: Schema aus W3C-Namensraum http://www.w3.org/2001/XMLSchema
  - hier XML-Schema für XML-Schema hinterlegt: Schema der Schemata

Ziel-Namensraum

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.books.org">
…
</xsd:schema>
```

- jedes XML-Schema definiert bestimmtes Vokabular (Elemente und Attribute)
- Dieses Vokabular wird einem Namensraum zugeordnet: **Ziel-Namensraum** (target namespace).
- Ziel-Namensraum wird wie jeder Namensraum mit URI identifiziert
- Definiertes Vokabular kann über URI identifiziert werden.

Verschiedene Arten von Datentypen



Einfache vs. komplexe Datentypen

Freie Universität Berlin

einfache Datentypen (simple types)

- beschreiben unstrukturierten Inhalt ohne Elemente oder Attribute (PCDATA)
- Schema der Schemata definiert 44 einfache Datentypen
- eigene einfache Datentypen können definiert werden

komplexe Datentypen (complex types)

- beschreiben strukturierten XML-Inhalt mit Elementen oder Attributen
- natürlich auch gemischten Inhalt

Elemente mit komplexen Typen können andere Elemente und/oder Attribute enthalten.

Anonyme vs. benannte Datentypen

Freie Universität Berlin

```
<xsd:element name="BookStore">
      <xsd:complexType>
        Liste von Büchern
      </xsd:complexType>
 </xsd:element>
```

- **anonymer** Datentyp
- lokale Definition

```
<xsd:complexType name="BookStoreType">
        Liste von Büchern
</xsd:complexType>
```

- **benannter** Datentyp
- globale Definition
- wiederverwendbar

Primitive vs. abgeleitete einfache Datentypen

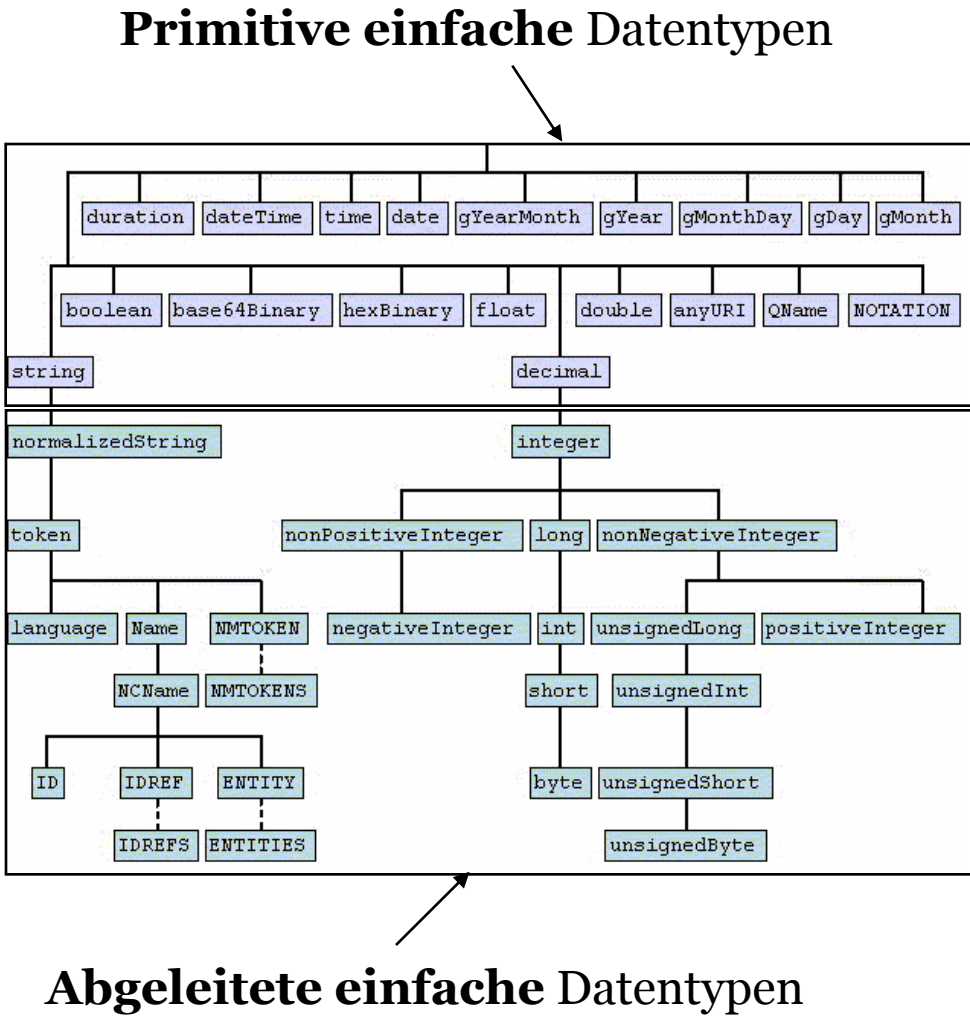
Freie Universität Berlin

primitive Datentypen (primitive types)

- nicht von anderen Datentypen abgeleitet

abgeleitete Datentypen (derived types)

- auf Basis von anderen Datentypen definiert, z.B. durch Einschränkung oder Erweiterung



Einfache Datentypen ABLEITEN

Freie Universität Berlin

```
<xsd:simpleType name="MyInteger">
 <xsd:union>
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="unknown"/>
    </xsd:restriction>
  </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```

Vereinigung
Vereinigung der Wertebereiche mehrerer einfacher Datentypen

Einschränkung (Teilmenge)
Einschränkung des Wertebereiches eines einfachen Datentyps

```
<xsd:simpleType name="longitudeType">
 <xsd:restriction base="xsd:integer">
  <xsd:minInclusive value="-180"/>
  <xsd:maxInclusive value="180"/>
 </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="IntegerList">
 <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
```

Listen
Liste als String (PCDATA): einzelne Elemente durch White Spaces getrennt

Komplexe Datentypen **BILDEN**

Freie Universität Berlin

1. Sequenz <xsd:sequence>…< /xsd:sequence>
   - **Reihenfolge vorgegeben**
   - Elemente erscheinen so oft, wie mit minOccurs/maxOccurs festgelegt.

2. Menge <xsd:all>…</xsd:all>
   - **Reihenfolge der Elemente beliebig**
   - Jedes Element erscheint hier genau einmal.

3. Auswahl <xsd:chioce>…< /xsd:choice>
   - Inhalt besteht aus **genau einem** der aufgezählten Alternativen.

Beachte: Alle Operatoren können minOccurs und maxOccurs selbst spezifizieren

Komplexe Datentypen ABLEITEN

Freie Universität Berlin

1. Erweiterung

- Datentyp wird durch zusätzliche Attribute und Elemente erweitert.
- Ergebnis: immer komplexer Datentyp

Basis-Datentyp (einfach oder komplex)	+	zusätzliche Attribute oder Elemente	=	erweiterter Datentyp (immer komplex)

2. Teilmenge

- Einschränkung des Wertebereiches eines Datentyps
- Resultierender Datentyp darf nur gültige Werte des ursprünglichen Datentyps enthalten (echte Teilmenge).
- hier wäre z.B. xsd:string statt xsd:unsignedShort nicht erlaubt: xsd:string keine Teilmenge von xsd:nonNegativeInteger

Element-Deklaration: 1. Möglichkeit

Freie Universität Berlin

```
<xsd:element name="Book" type="BookType" maxOccurs="unbounded"/>
<xsd:element name="name" type="type" minOccurs="int" maxOccurs="int"/>
```

- **name**: Name des deklarierten Elementes
- **type**: Datentyp (benannt oder vordefiniert)
- **minOccurs**: so oft erscheint das Element mindestens (nicht-negative Zahl)
- **maxOccurs**: so oft darf das Element höchstens erscheinen (nicht-negative Zahl oder unbounded).
- Default-Werte von minOccurs und maxOccurs jeweils 1
- Beachte: abhängig vom Kontext gibt es Einschränkungen von minOccurs und maxOccurs

Element-Deklaration: 2. Möglichkeit

Freie Universität Berlin

- anonymer Datentyp ist entweder komplex:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:complexType>
        ...
    </xsd:complexType>
</xsd:element>
```

- oder einfach:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
    <xsd:simpleType>
        ...
    </xsd:simpleType>
</xsd:element>
```

Deklaration von Attributen

- ähnlich wie Elemente
- aber nur **einfache Datentypen** erlaubt
- Deklaration mit **benanntem Datentyp**:

```
<xsd:attribute name= "name" type= "type" />
```

- oder Deklaration mit **anonymen Datentyp**:

```
<xsd:attribute name= "name">
    <xsd:simpleType>
     …
    </xsd:simpleType>
</xsd:attribute>
```

Deklaration von Attributen


```
<xsd:attribute name= "name" type= "type" use="use"
      default= "value" />
```

- **use="optional"** Attribut optional
- **use="required"** Attribut obligatorisch
- **use="prohibited"** Attribut unzulässig
- Beachte: Wenn nichts anderes angegeben, ist das Attribut optional!

- **default**: Standard-Wert für das Attribut

DTDs vs. XML Schema

Freie Universität Berlin

DTDs	XML Schema
vereinfachte SGML-DTD, Teil von XML 1.0/1.1	eigener W3C-Standard
eigene Sprache/Syntax	XML-Schema = XML-Sprache
kompakter und lesbarer	ausdrucksstark & mächtig
nur wenige „Datentypen“	unterstützt Datentypen
reihenfolgeunabhängige Strukturen schwierig zu definieren	reihenfolgeunabhängige Strukturen einfach zu definieren
Datentypen nicht erweiterbar, d.h. keine eigenen Datentypen	Datentypen erweiterbar, d.h. Definition von eigenen Datentypen möglich
keine Namenräume	Unterstützt Namenräume
zur Beschreibung von Text-Dokumenten ausreichend	zur Beschreibung von Daten besser geeignet

DTD & XML Schema → Lernziele

Lernziele

- DTDs und XML-Schemata lesen, verstehen und anpassen können (!)
- Vorteile von XML-Schema gegenüber DTDs
- Welche Arten von Datentypen gibt es in DTDs & welche in XML Schema?
- Wie kann man Datentypen in XML Schema bilden/ableiten?

XML Parser

Kategorien von Parser

Validierender vs. nicht-validierender Parser

- Wird die Validität des Dokumentes untersucht?

Pull- vs. Push-Parser

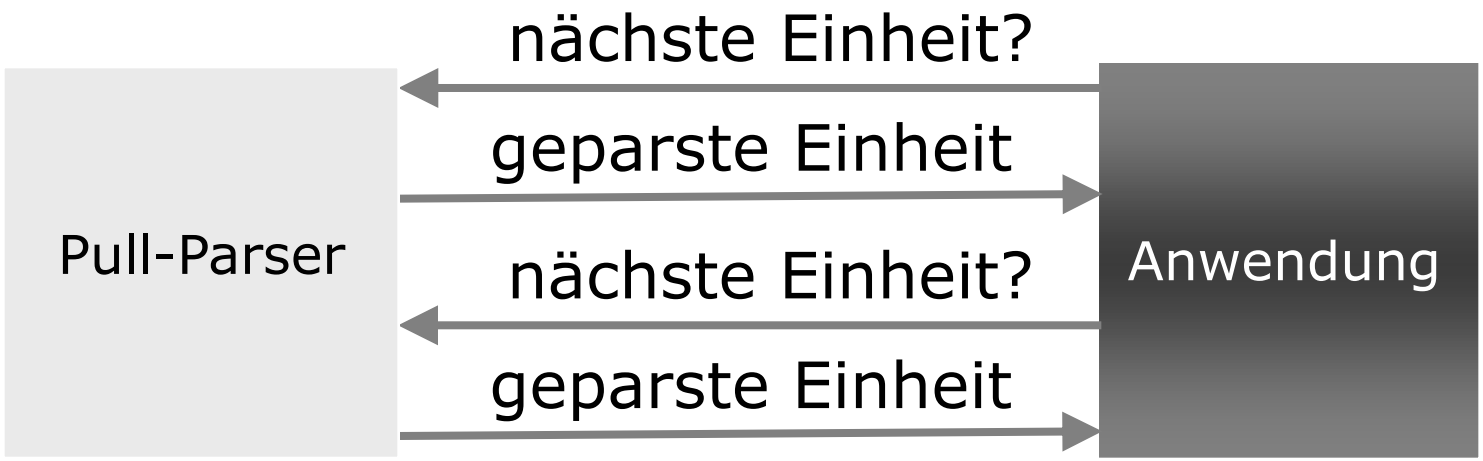
- Wer hat Kontrolle über das Parsen: die Anwendung oder der Parser?

Einschritt- vs. Mehrschritt-Parser

- Wird das XML-Dokument in einem Schritt geparst oder Schritt für Schritt?

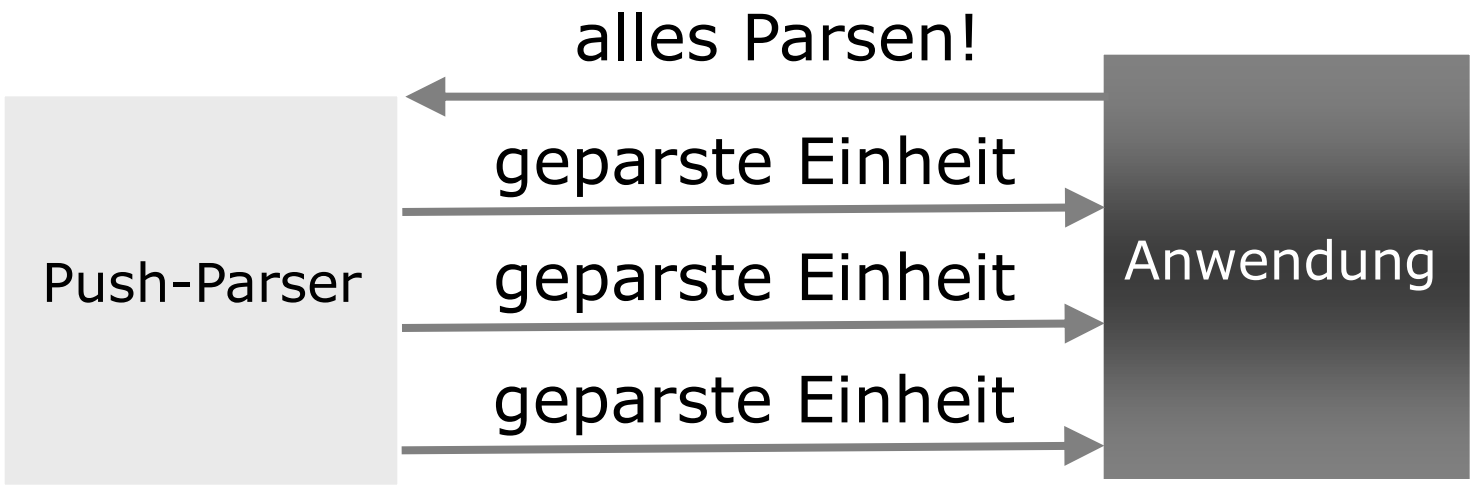
- Beachte: Kategorien unabhängig voneinander, können kombiniert werden

Pull-Parser



- Anwendung hat Kontrolle über das Parsen.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- Beachte: „Pull” aus Perspektive der Anwendung.

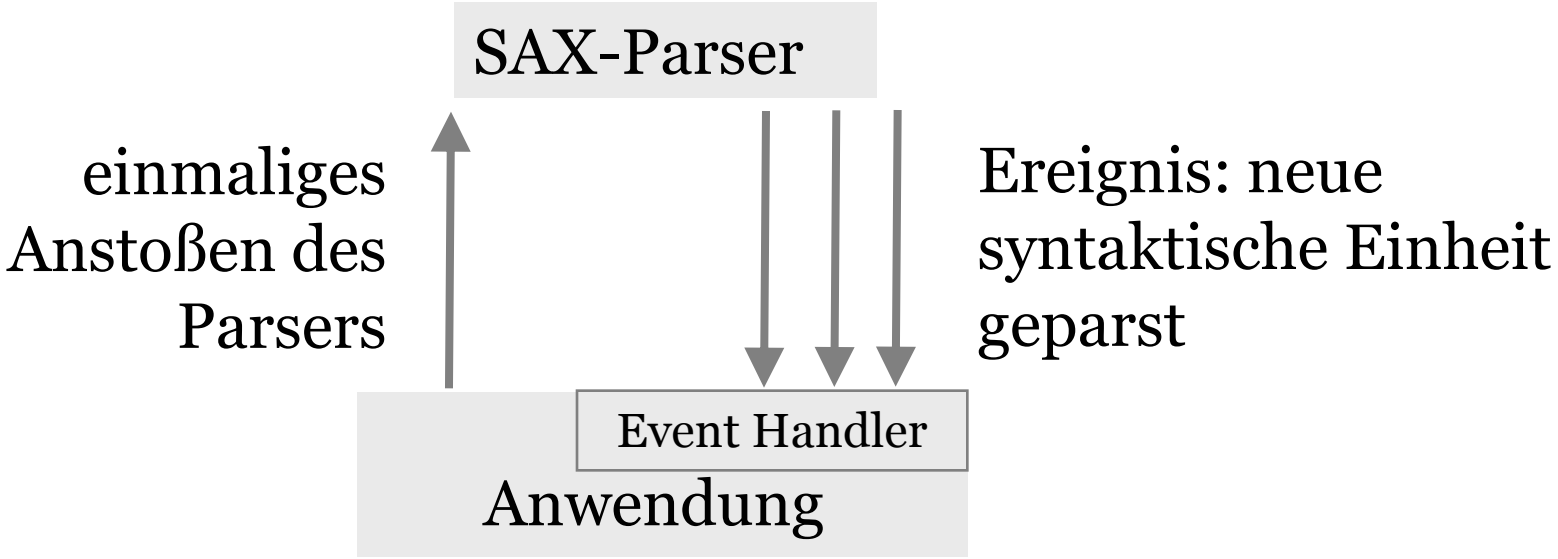
Push-Parser



- Parser hat Kontrolle über das Parsen.
- Sobald der Parser eine syntaktische Einheit analysiert hat, übergibt er die entsprechende Analyse.
- Beachte: „Push" aus Perspektive der Anwendung.

SAX: Simple API for XML

- Mehrschritt-Push-Parser für XML
- kein W3C-Standard, sondern *de facto* Standard
- standardisiertes API
- ursprünglich nur Java-API, inzwischen auch: C, C++, VB, Pascal, Perl



Beispiel

```
<priceList>
  <coffee>
      <name>
            Mocha Java
      </name>
      <price>
            11.95
      </price>
  </coffee>
</priceList>
```

Parser ruft startElement(…,priceList,…) auf.
Parser ruft startElement(…,coffee,…) auf.
Parser ruft startElement(…,name,…) auf.
Parser ruft characters("Mocha Java",…) auf.
Parser ruft endElement(…,name,..) auf.
Parser ruft startElement(…,price,…) auf.
Parser ruft characters("11.95",…) auf.
Parser ruft endElement(…,price,…) auf.
Parser ruft endElement(…,coffee,…) auf.
Parser ruft endElement(…,priceList,…) auf.

- Ereignisfluss: Sobald Einheit geparst wurde, wird Anwendung benachrichtigt.
- Beachte: Es wird kein Parse-Baum aufgebaut!

Callback-Methoden

- Methoden des **Event-Handlers** (also der Anwendung), die vom Parser aufgerufen werden
- für jede syntaktische Einheit eigene Callback-Methode, u.a.:
  - startDocument und endDocument
  - startElement und endElement
  - Characters
  - processingInstruction

DefaultHandler

- Standard-Implementierung der Callback-Methoden: tun jeweils nichts!
- können natürlich überschrieben werden

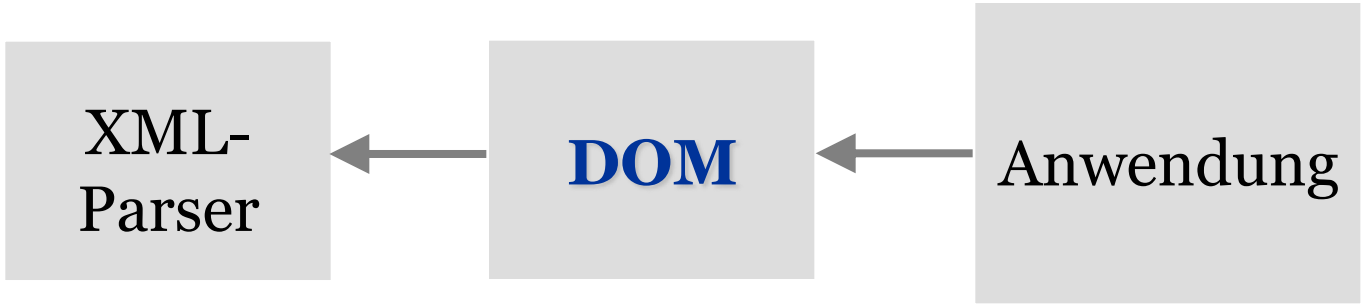
Vor- und Nachteile von SAX

Freie Universität Berlin

+ sehr effizient und schnell, auch bei großen XML-Dokumenten
+ relative einfach

- Kontext (Parse-Baum) muss von Anwendung selbst verwaltet werden.
- abstrahiert nicht von XML-Syntax
- nur Parsen möglich, keine Modifikation oder Erstellung von XML-Dokumenten

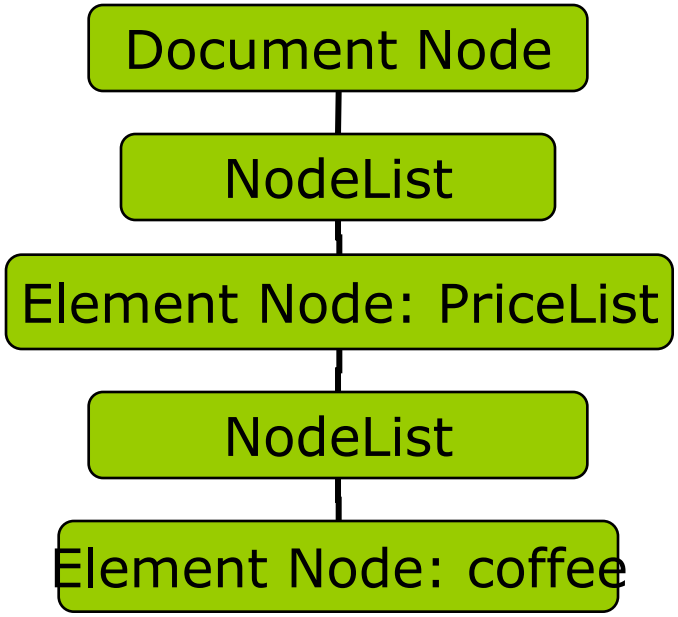
Document Object Model (DOM)



- streng genommen kein Parser, sondern abstrakte Schnittstelle zum Zugreifen, Modifizieren und Erstellen von Parse-Bäumen
- W3C-Standard
- unabhängig von Programmiersprachen
- nicht nur für XML-, sondern auch für HTML-Dokumente
- im Ergebnis aber **Einschritt-Pull-Parser**

DOM-Parse-Bäume

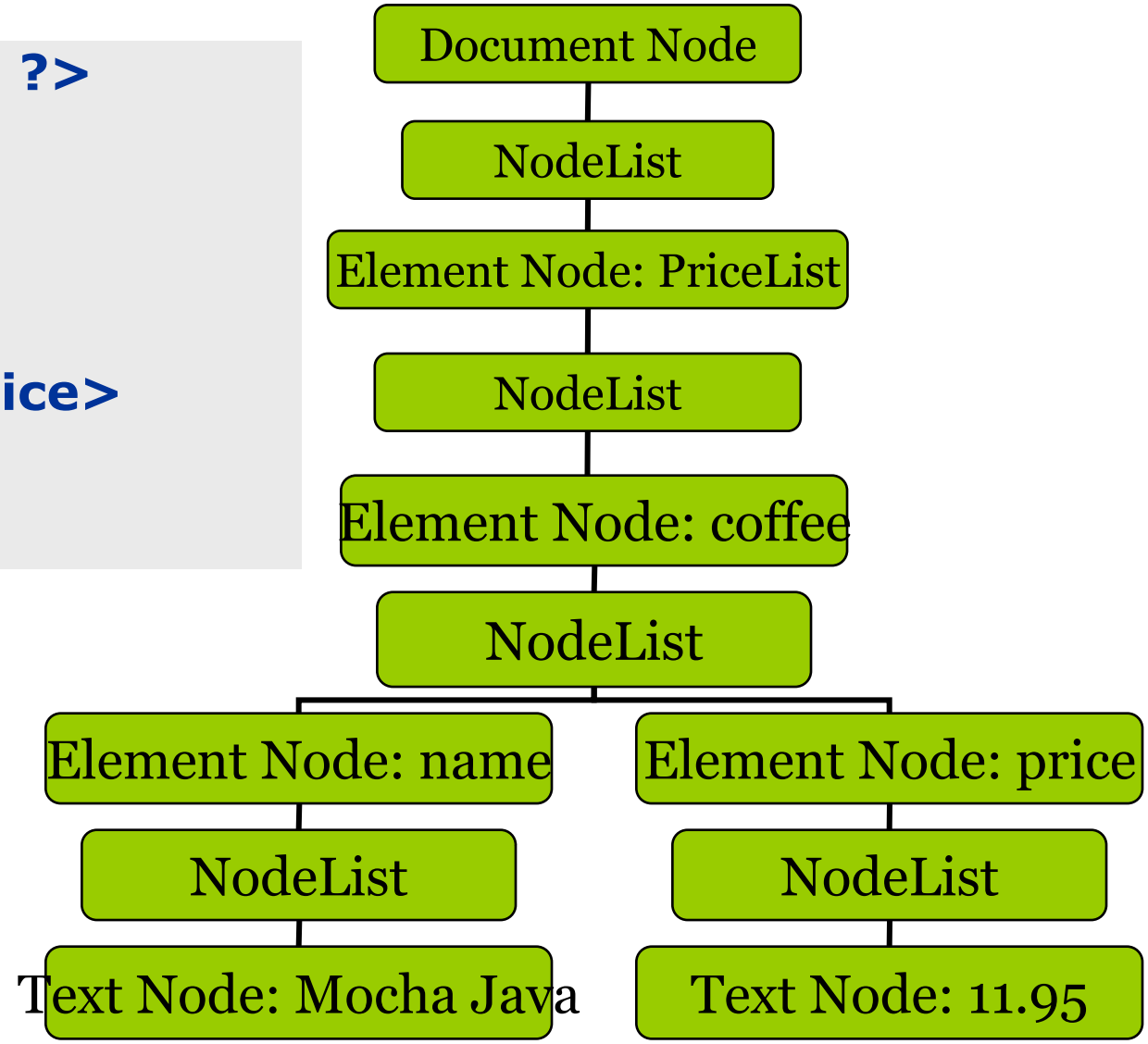
```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```



- Beachte: Dokument-Wurzel (Document Node) ≠ priceList
- Document Node: virtuelle Dokument-Wurzel, um z.B. version="1.0" zu repräsentieren
- Document Node und Element Node immer NodeList als Kind

DOM-Parse-Bäume

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha
   Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```



- Beachte: PCDATA wird als eigener Knoten dargestellt.

Vor- und Nachteile von DOM

+ Kontext (Parse-Baum) muss nicht von Anwendung verwaltet werden.
+ einfache Navigation im Parse-Baum
+ direkter Zugriff auf Elemente über ihre Namen
+ nicht nur Parsen, sondern auch Modifikation und Erstellung von XML-Dokumenten

- speicherintensiv
- abstrahiert nicht von XML-Syntax

SAX oder DOM?

Freie Universität Berlin

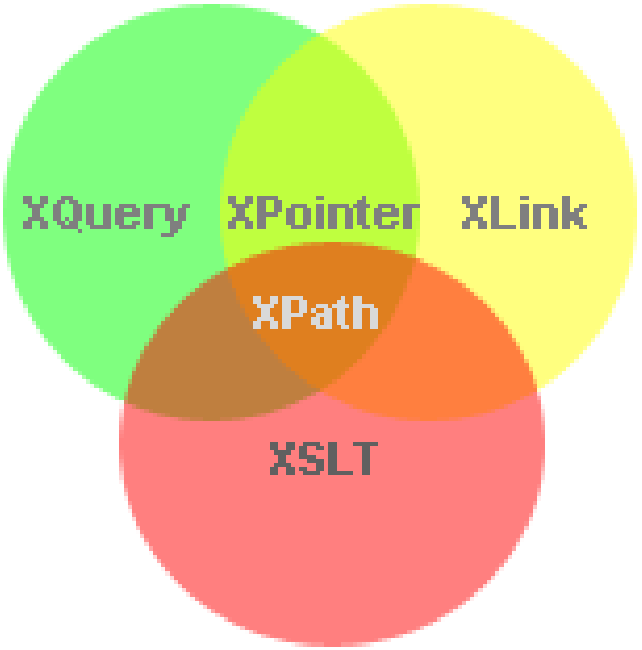
SAX	DOM
ereignis-orientierter Ansatz	**modell-orientierter** Ansatz
	vollständige Umsetzung in eine **Baumstruktur**
parsen	mehrere Verarbeitungsmöglichkeiten
XML-Dokument als **Eingabestrom** (Streaming-Verfahren)	XML-Dokument **vollständig im Speicher** (Baummodell des Dokuments)
schnelle Verarbeitung von großen XML-Dokumenten	**langsame** Verarbeitung von großen XML-Dokumenten
wenig Hauptspeicher benötigt	**mehr** Hauptspeicher benötigt
	nach dem Einlesen kann auf alle Teilstrukturen des XML-Dokuments zugegriffen werden

XML Parser → Lernziele

Lernziele

- Welche Kategorien von Parser gibt es?
- Wie arbeitet ein SAX-Parser?
  - Was sind Call-back Methoden?
  - Was ist ein Event-Handler?
- Wie funktioniert ein DOM-Parser?
  - DOM-Baum
- Vor- und Nachteile von SAX- und DOM-Parser

XPath & Co



Absolute und relative Pfade

- absolute Pfade
  - beginnen mit "/„

    z.B. /order/item

    lesen: (➔)Folge dem Pfad von dem Wurzelknoten zu einem Kind-Element order und von dort aus zu einem Kind-Elementen item!
- relative Pfade
  - beginnen mit einem Element oder Attribut

    z.B. order/item

    lesen: (➔) item-Elemente, die Kind eines Elementes order sind
  - Element order kann an beliebiger Stelle des XML-Dokumentes

Pfad-Ausdrücke (gekürzte Schreibweise)

.	aktueller Knoten
..	Eltern-Knoten
*	beliebiges Kind-Element
@*	beliebiges Attribut
//	überspringt ≥ 0 Hierarchie-Ebenen nach unten
[]	Prädikatbeschreibung (Ziel → genauere Element-Spezifikation)
|	Auswahl (Vereinigung)

- Beispiel: *|@*
  „Kind-Element oder Attribut des aktuellen Knotens“

Prädikate – Randbedingungen für Pfade

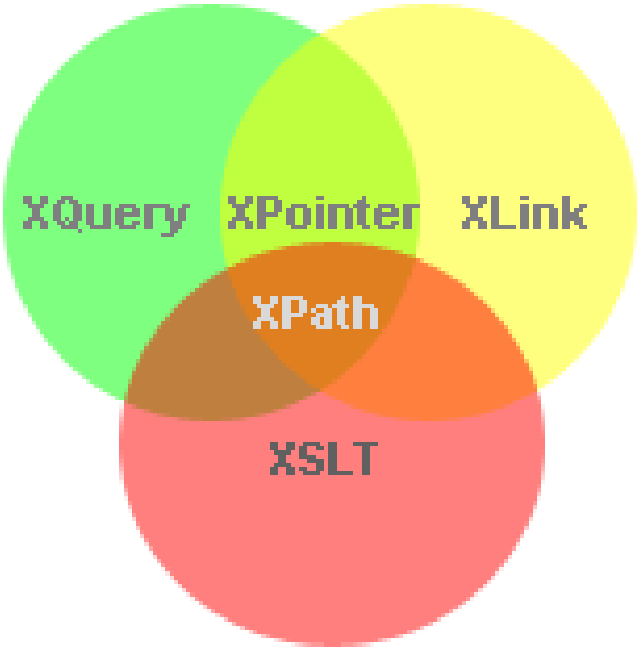
- order/item[@item-id = 'E16-2']
  - item-Elemente, die Kind von order sind und Attribut item-id mit Wert 'E16-2' haben

```
<order id="4711">
  <item item-id="E16-2">
    <name>buch</name>
  </item>
</order>
```

- Randbedingungen können an beliebiger Stelle in einem Pfad vorkommen:
  - order[@order-id = '4711']/item

```
<orders>
  <order ored-id="4711">
    <item item-id="E16-2">
      <name>buch</name>
    </item>
  </order>
  <order id="4711">
    <item item-id="E16-3"/>
  </order>
</orders>
```

XSLT



Warum XML transformieren?

Freie Universität Berlin

Trennung Inhalt und Präsentation

- XML trennt Inhalt von Präsentation (Layout)
- Für eine entsprechende Darstellung müssen XML-Inhalte transformiert werden:
  - XML-Inhalt ➔ Layout

Inhaltliche Transformationen

- Daten mit XML repräsentiert
- unterschiedliche Sichten (Views) auf XML-Inhalte erfordern Transformationen:
  - XML-Inhalt ➔ XML-Inhalt

Was ist XSLT?

- in XML beschriebene Sprache zur Transformation von XML-Dokumenten
- eine beschreibende Sprache
- XSLT-Programme (stylesheets) haben XML-Syntax
  - ➔ plattformunabhängig
- erlaubt XML-Dokumente in beliebige Textformate zu Transformieren:
  - XML ➔ XML/HTML/XHTML/WML/RTF/ASCII …
- W3C-Standard seit 1999

Ursprungs- und Ergebnisdokument


```
<?xml version="1.0"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>…</date>
  <customer>Sally
   Finkelstein</customer>
</order>
```

```
<xsl:template match="order/item">
   <p><xsl:value-of select="."/></p>
</xsl:template>
```

Template

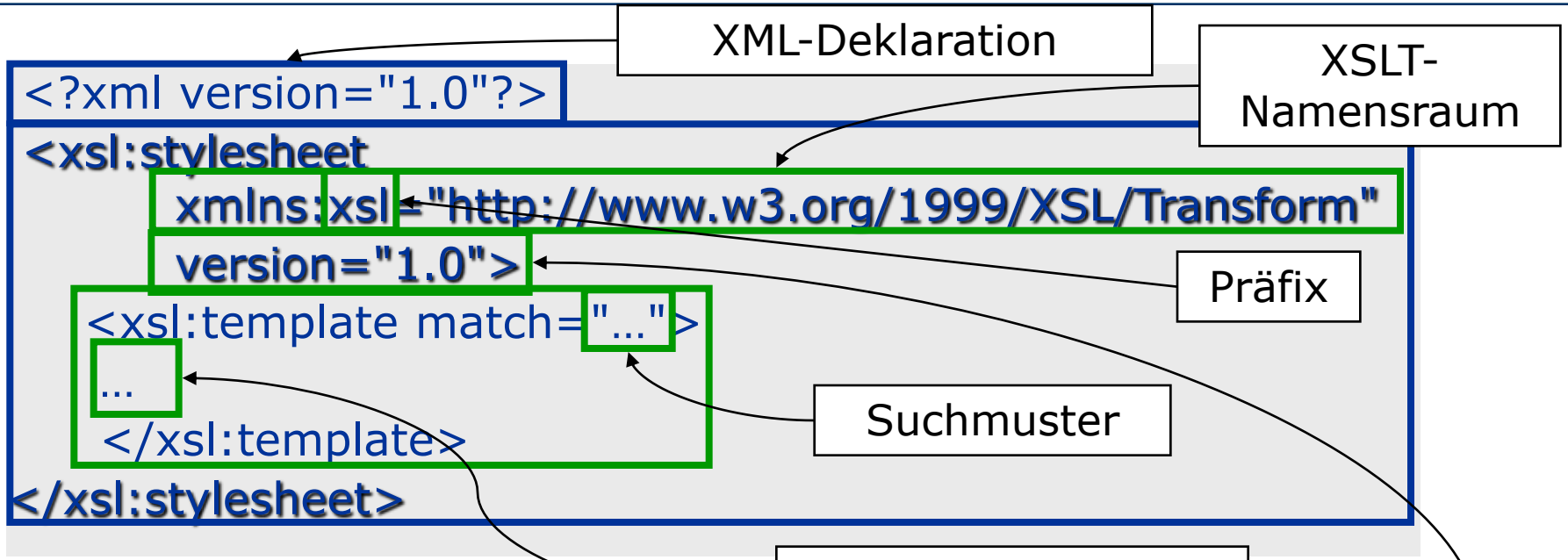
```
<p>Production-Class Widget</p>
```

Ursprungsdokument → Ursprungsbaum (source document → source tree)

Transformation

Ergebnisbaum → Ergebnisdokument (result tree → result document)

Grundstruktur von Stylesheets



- XML-Dokument
- Dokument-Wurzel:
  - stylesheet oder transform aus entsprechendem W3C-Namensraum
  - stylesheet und transform gleichbedeutend
  - obligatorisches Attribut: version

1. Neue Inhalte erzeugen (I)

- Templates können alle XML-Inhalte erzeugen: PCDATA, Elemente und Attribute
- einfach normale XML-Syntax verwenden:

```
<xsl:template match="…">
   <p style="color:red">neuer Text</p>
</xsl:template>
```

- Beachte: Stylesheets müssen wohlgeformte XML-Dokumente sein, daher z.B. nicht erlaubt:

```
<xsl:template match="…">
   <br>neuer Text
</xsl:template>
```

1. Neue Inhalte erzeugen (II)

- statt üblicher XML-Syntax

```
<xsl:template match="…">
   <p style="color:red">neuer Text</p>
</xsl:template>
```

- auch möglich:

```
<xsl:template match="…">
   <xsl:element name="p">
      <xsl:attribute name="style">color:red</xsl:attribute>
     <xsl:text>neuer Text</xsl:text>
  </xsl:element>
 </xsl:template>
```

<xsl:copy-of select="."> Element

- Kopiert aktuellen Teilbaum
- aktueller Teilbaum: Baum, der vom aktuellen Knoten aufgespannt wird, **einschließlich** aller Attribute und PCDATA

<xsl:copy> Element

- Kopiert aktuellen Knoten **ohne** Kind-Elemente, Attribute und PCDATA

⇨ Kopiert nur Wurzel-Element des aktuellen Teilbaums

<xsl:value-of select="."> Element

- Extrahiert PCDATA, das im aktuellen Teilbaum vorkommt

Transformations-Beispiel

Freie Universität Berlin

Stylesheet

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
</xsl:template>

<xsl:template match="B">
   <xsl:value-of select="@id"/>
 </xsl:template>

 <xsl:template match="C">
   <xsl:value-of select="@id"/>
 </xsl:template>

<xsl:template match="D">
  <xsl:value-of select="@id"/>
 </xsl:template>
```

Dokument

```
<source>
  <A id="a1">
     <B id="b1"/>
     <B id="b2"/>
  </A>
  <A id="a2">
     <B id="b3"/>
     <B id="b4"/>
     <C id="c1">
        <D id="d1"/>
     </C>
     <B id="b5">
        <C id="c2"/>
     </B>
  </A>
</source>
```

kein Template anwendbar

Template "A" wird angewandt

Template "B" wäre anwendbar, es werden aber keine Templates aufgerufen!

Ausgabe

a1
a2

Iteration statt Rekursion

```
<xsl:template match="A">
  <xsl:for-each select="*">
    <xsl:value-of select="@id"/>
  </xsl:for-each>
</xsl:template>
```

- **xsl:value-of** wird auf alle select-Pfade der for-each-Schleife angewandt.
- Beachte: select-Pfad von **xsl:for-each** relativ zum Kontext-Knoten des Templates, hier also "A/*".

```
<source>
  <A id="a1">
    <B id="b1"/>
    <B id="b2"/>
  </A>
  <A id="a2">
    <B id="b3"/>
    <B id="b4"/>
    <C id="c1">
      <D id="d1"/>
    </C>
    <B id="b5">
      <C id="c2"/>
    </B>
  </A>
</source>
```

→

b1
b2
b3
b4
c1
b5

Rekursiver Aufruf aller Templates

<xsl:apply-templates/>

- versucht Templates auf Kinder des aktuellen Knotens anzuwenden
- Kind bedeutet hier: Kind-Element, Text-Knoten oder Attribut-Knoten
- Mit <xsl:apply-templates select = "..."/> auch rekursiver Aufruf an beliebiger Stelle möglich.
- Vorsicht: Terminierung nicht automatisch sichergestellt!
- Beispiel:

```
<xsl:template match="A">
  <xsl:value-of select="@id"/>
  <xsl:apply-templates select="/"/>
</xsl:template>
```

Vordefinierte Templates

1.vordefinierte Template

- realisiert rekursiven Aufruf des Prozessors, *wenn kein Template anwendbar* ist

2.vordefinierte Template

- kopiert PCDATA und Attribut-Werte des aktuellen Knotens in das Ergebnisdokument

Leeres Stylesheet

- traversiert gesamtes Ursprungsdokument und extrahiert dabei PCDATA und Attribut-Werte

Überschreiben

- Vordefinierte Templates können durch speziellere Templates überschrieben werden

Vor- und Nachteile von XSLT

Freie Universität Berlin

+	-
+ plattformunabhäng + relativ weit verbreitet + Verarbeitung in Web-Browsern + Standard-Transformationen (wie XML ➔ HTML) einfach zu realisieren. + Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden. + extrem mächtig	- Entwickler müssen speziell für die Transformation von XML-Dokumenten neue Programmiersprache lernen. - Anbindung von Datenbanken umständlich - manche komplexe Transformationen nur umständlich zu realisieren

XPath & Co und XSLT → Lernziele

Lernziele

- XPath-Ausdrücke & XPath-Funktionen
- Warum XML transformieren?
- Was ist XSLT?
- Wie funktioniert XSLT?
- Iteration/Rekursion bei XSLT
- Welche vordefinierten Templates gibt es?
- Vor- und Nachteile von XSLT

Klausur → Organisatorisches

Organisatorisches

➔ 13. Juli um 14:00 **pünktlich (s.t.!!!)**

➔ im Hörsaal Informatik

Was sind die Voraussetzungen?

- Sie haben die Folien der Vorlesung verstanden und können dieses Wissen anwenden.
- Sie haben aktiv an der Projektarbeit teilgenommen
- Sie beherrschen den Vorlesungsinhalt passiv und soweit aktiv, dass Sie Änderungen vornehmen können

Klausur – Fragen

Freie Universität Berlin

Struktur: Sie haben 90 Minuten

- 25 Multiple Choice Fragen (25 Punkte insgesamt)
- 12 ausführliche Fragen (65 Punkte insgesamt)

Multiple Choice

- Aufgepasst: **nicht immer nur eine** korrekte Antwort!

ausführliche Fragen

- Typischerweise mit gegebenem XML Code
- Aufgabe: Korrektur, Änderung, Erweiterung des Codes.
- Oder: XML Instanz von Schema, XSLT Ausgabe usw.

Notenschema

Punkte	Note
≥ 85,5	1,0
≥ 81	1.3
≥ 76,5	1,7
≥ 72	2,0
≥ 67,5	2,3
≥ 63	2,7
≥ 58,5	3,0
≥ 54	3,3
≥ 49,5	3,7
≥ 41	4,0

Klausur – allg. Infos

- **Fragen** sind nur auf **Deutsch**
- **Antworten** auf **Deutsch** oder **Englisch** möglich
- Keine eigenen (Wörter-)Bücher, Papiere usw. erlaubt
- Während der Prüfung, dürfen Sie Hand hochheben wenn Sie Hilfe brauchen/Frage haben
- Sitzordnung:
  - **jede zweite Reihe** & **jeder zweite Platz**!
- Studentenausweis + Ausweis mit Foto mitbringen!
- Handy aus!

Weitere Fragen?

Viel Glück!