

# Intelligent Systems on the World Wide Web

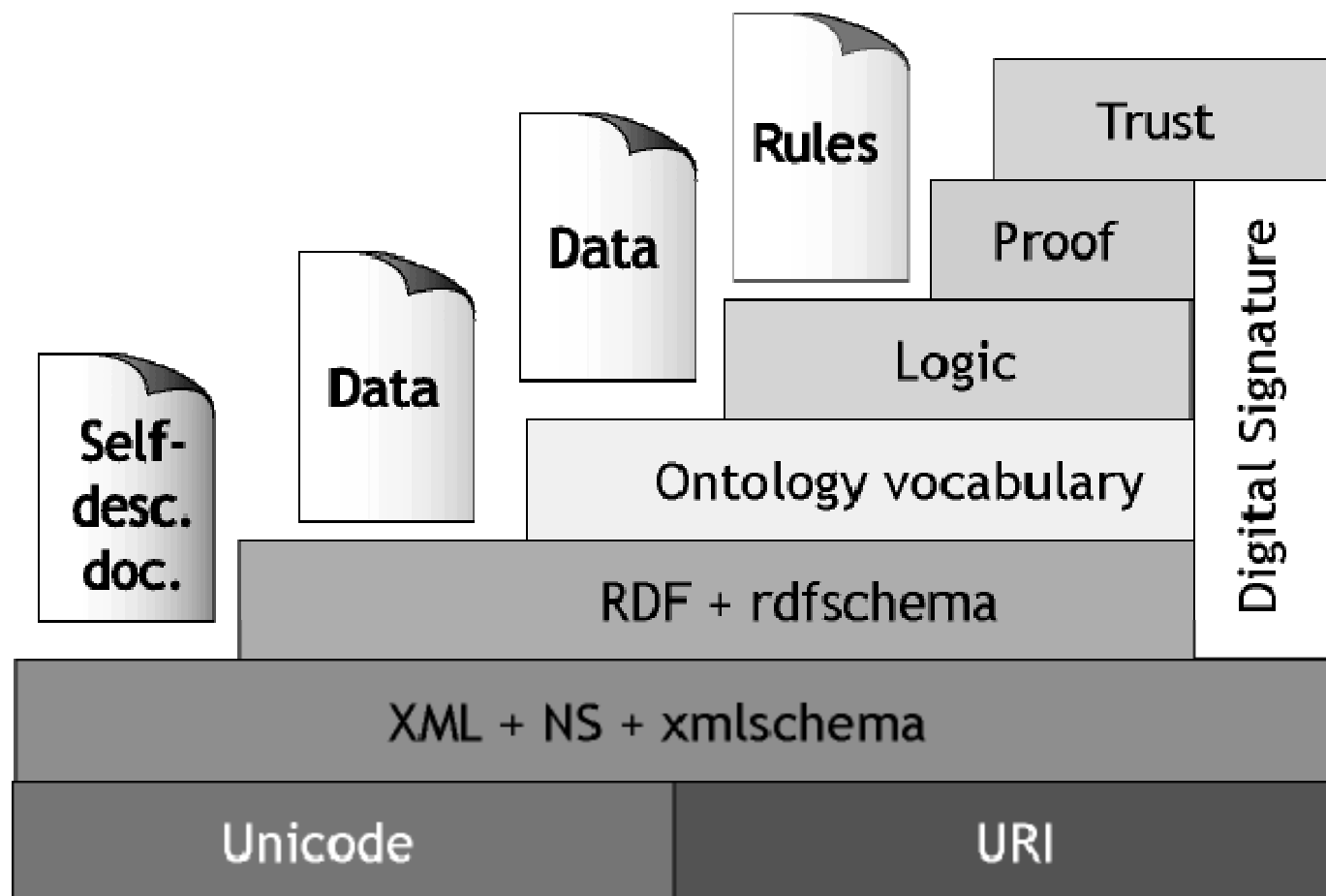
## In-Depth XML

York Sure

Institute for Applied Computer Science and Formal  
Description Methods (AIFB)

Karlsruhe University

## (One) Layer Model of the Semantic Web



# Extensible Markup Language



*Purpose here: storing and exchanging knowledge*

*Non-purpose here: structuring documents*

# XML

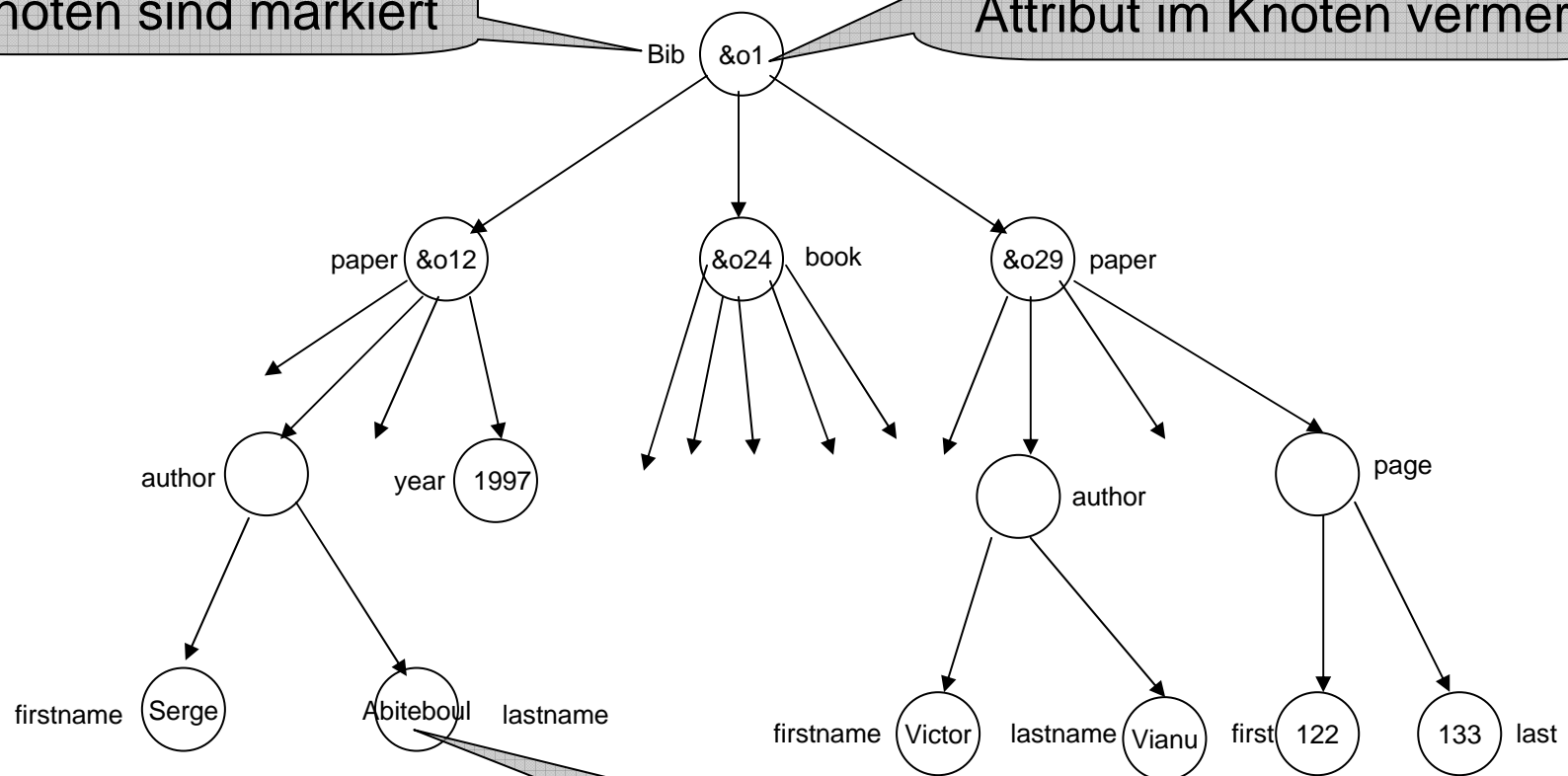
- eXtensible Markup Language
- Ursprung: strukturierter Text (HTML4.0  $\in$  XML  $\subset$  SGML)
- Web-Standard (W3C) zum Datenaustausch:
  - Ein- und Ausgabedaten von Anwendungen können mittels XML beschrieben werden
  - Industrie muß sich nur noch auf standardisierte Beschreibung einigen
- Komplementärsprache zu HTML:
  - HTML beschreibt die Präsentation
  - XML beschreibt den Inhalt
- Datenbank-Sichtweise: XML als Datenmodell für semistrukturierte Daten

# XML-Modell

## Veranschaulichung von Objekten als gerichtete Graphen

Knoten sind markiert

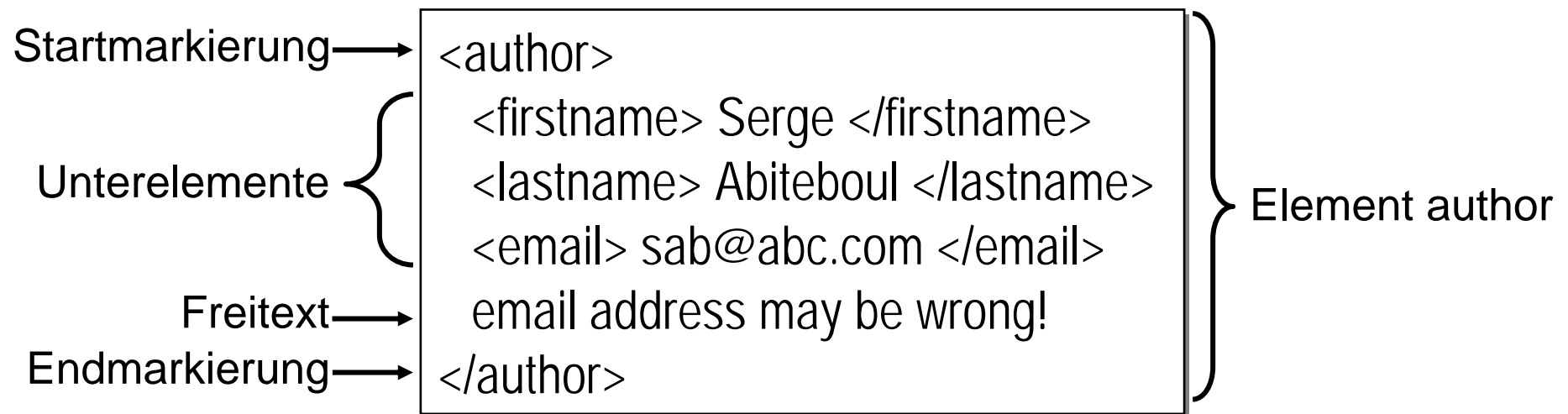
Objektidentifikator als  
Attribut im Knoten vermerkt



Text im Knoten vermerkt

## XML-Syntax (1) – XML-Element


- XML-Element (engl. element):
  - Beschreibung eines Objekts, die durch passende Markierungen (tags) wie `<author>` und `</author>` geklammert ist
  - Inhalt eines Elements: Text und/oder weitere Elemente (Unterelemente)
  - Elemente können beliebig geschachtelt sein
  - Leere Elemente: `<year></year>` kurz: `<year/>`



## XML-Syntax (2) – XML-Attribut

- XML-Attribut (engl. attribute):
  - Name-Zeichenkettenwert-Paar
  - Assoziiert mit einem Element
  - Alternative Möglichkeit, Daten zu beschreiben

Attribut email



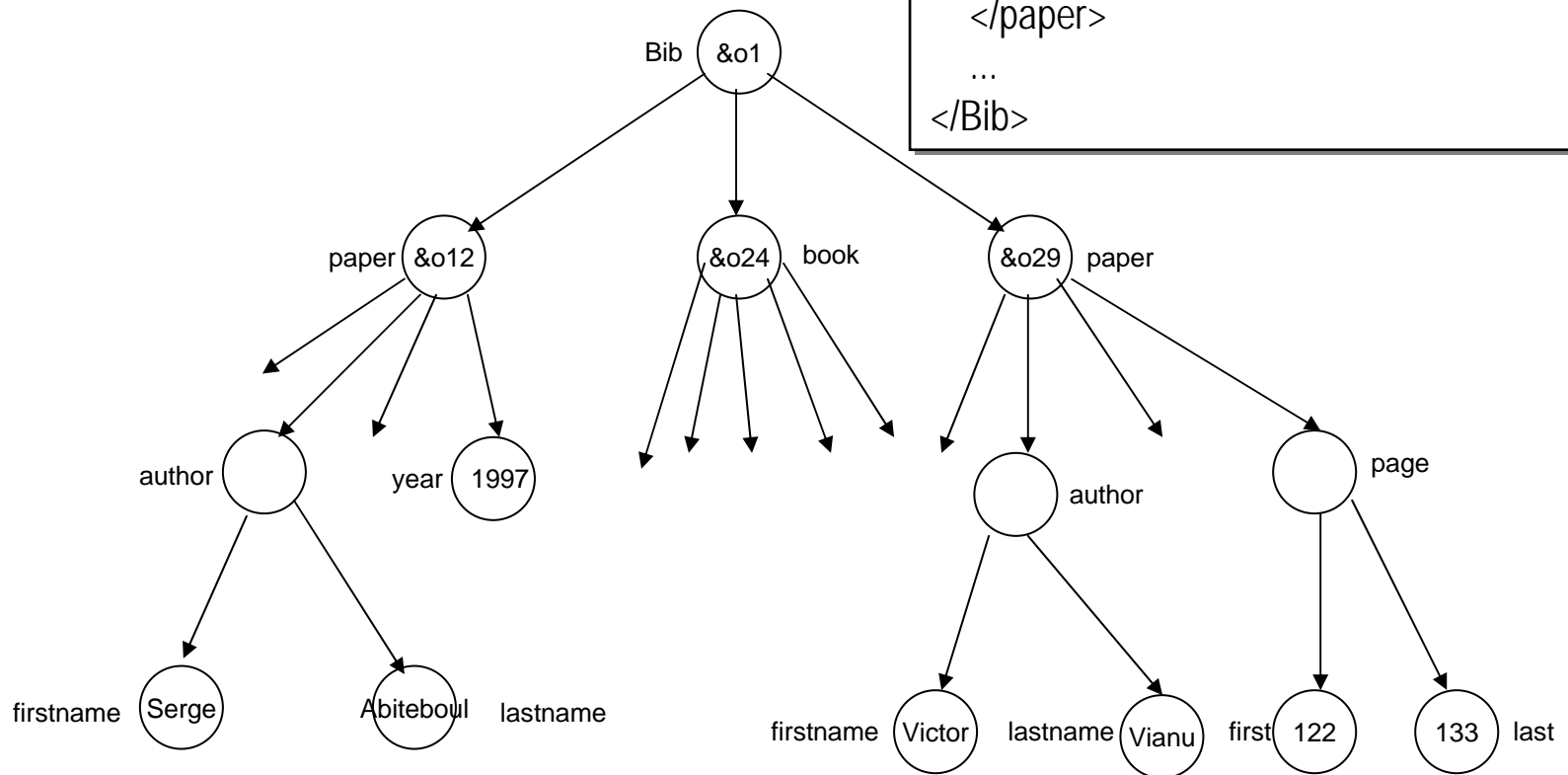
```
<author email="sab@abc.com">  
  <firstname> Serge </firstname>  
  <lastname> Abiteboul </lastname>  
</author>
```

Weitere denkbare Beschreibung derselben Daten:

```
<author firstname="Serge" lastname="Abiteboul" email="sab@abc.com"/>
```

# XML-Modell

```
<Bib id="o1">  
  <paper id="o12">  
    <title> Foundations of Databases </title>  
    <author>  
      <firstname> Serge </firstname>  
      <lastname> Abiteboul </lastname>  
    </author>  
    <year> 1997 </year>  
    <publisher> Addison Wesley </publisher>  
  </paper>  
  ...  
</Bib>
```



# XML vs HTML

- HTML: feste Bezeichner (tag) und Semantik (die Darstellung von Text)
- XML: freie Bezeichner zur Beschreibung von anwendungsspezifischer Syntax (Meta-Grammatik)
- $XML \subset SGML$

```
<h1> Bib </h1>
<p>
  <i> Foundations of Databases </i>
  Serge Abiteboul
  <br > Addison Wesley, 1997
<p>
  ...
```

**HTML**

```
<Bib id="o1">
  <paper id="o12">
    <title> Foundations of Databases </title>
    <author>
      <firstname> Serge </firstname>
      <lastname> Abiteboul </lastname>
    </author>
    <year> 1997 </year>
    <publisher> Addison Wesley </publisher>
  </paper>
  ...
</Bib>
```

**XML**

# XML-Dokument

- **XML-Dokument:**
  - Ein Text-Dokument, das XML-Beschreibungen enthält
  - Datenbank-Sichtweise: sozusagen die Datenbasis
- **Wohlgeformtes XML-Dokument:**
  - Alle Elemente sind korrekt mit Start- und End-Tags geklammert
  - Dokument enthält genau ein Wurzelement
  - Wohlgeformte Dokumente dürfen aber immer noch unstrukturierten Freitext enthalten
- **Gültiges (engl. valid) XML-Dokument:**
  - Wohlgeformtes XML-Dokument, das zu einem assoziierten Schema uneingeschränkt konform ist
  - Mittels eines Schemas kann man also die Gültigkeit eines XML-Dokumentes überprüfen
  - Sinnvoll beim Datenaustausch (standardisierte Beschreibung)

# Structuring XML

**Structuring**

## Schemata in XML (Optional !)

- **DTD – Document Type Definitions:**
  - Einfache Grammatik für ein XML-Dokument
    - **Deklaration von Elementen, Attributen, u.a.**
    - **Beschränkt die beliebige Verschachtelung von Elementen und Attributen**
  - Ist Teil des XML-Standards
  - Erbe von SGML
- **XML-Schema:**
  - Komplexere Datendefinitionssprache:
    - **Viele standardisierte Basistypen, z.B. float, double, decimal, boolean**
    - **Typen und typisierte Objektreferenzen**
    - **Klassenhierarchien / Vererbung**
    - **Konsistenzbedingungen**
  - Standard in Ergänzung zu XML (noch nicht verabschiedet!)
  - Abwärtskompatibel zu DTD

# XML-Schemata I: DTD

- Eine DTD definiert eine kontextfreie Grammatik für ein XML-Dokument
- Zuvor beliebige Elemente und Attribute werden auf eine definierte Auswahl und Struktur eingeschränkt

```
<bib>
  <paper id="o12">
    <title> Foundations of Databases </title>
    <author>
      <firstname> Serge </firstname>
      <lastname> Abiteboul </lastname>
    </author>
    <year> 1997 </year>
    <publisher> Addison Wesley </publisher>
  </paper>
  ...
</bib>
```

**XML**

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

**DTD**

## DTD – Deklaration von Elementen

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:  
<!ELEMENT Name (Definition)>
- Einziger atomarer Typ: **#PCDATA**  
(Parsed Character DATA)
- **(a,b,c)**: Liste von Unterelementen
- **(a|b|c)**: Alternativen
- Kardinalitäten:
  - \*    keinmal oder beliebig oft
  - +    einmal oder beliebig oft
  - ?    kein- oder einmal  
(optional)
  - (ohne Angabe): genau einmal
- **EMPTY** : Erzwingen von leerem Element

```

<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>

```

**DTD**

## DTD – Deklaration von Elementen (2)

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:  
<!ELEMENT Name (Definition)>
- Einziger atomarer Typ: **#PCDATA** (Parsed Character DATA)
- **(a,b,c)**: Liste von Unterelementen
- **(a|b|c)**: Alternativen
- Kardinalitäten:
  - \*  keinmal oder beliebig oft
  - +  einmal oder beliebig oft
  - ?  kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- **EMPTY** : Erzwingen von leerem Element

### Einleitung und Festlegung des Wurzelements bib

```

<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>

```

## DTD – Deklaration von Elementen (3)

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:  
<!ELEMENT Name (Definition)>
- Einziger atomarer Typ: **#PCDATA** (Parsed Character DATA)
- **(a,b,c)**: Liste von Unterelementen
- **(a|b|c)**: Alternativen
- Kardinalitäten:
  - \*    keinmal oder beliebig oft
  - +    einmal oder beliebig oft
  - ?    kein- oder einmal  
      (optional)
  - (ohne Angabe): genau einmal
- **EMPTY** : Erzwingen von leerem Element

bib kann beliebig viele Elemente vom Typ paper enthalten

```

<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
  
```

**DTD**

## DTD – Deklaration von Elementen (4)

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:  
<!ELEMENT Name (Definition)>
- Einziger atomarer Typ: **#PCDATA** (Parsed Character DATA)
- **(a,b,c)**: Liste von Unterelementen
- **(a|b|c)**: Alternativen
- Kardinalitäten:
  - \*    keinmal oder beliebig oft
  - +    einmal oder beliebig oft
  - ?    kein- oder einmal  
      (optional)
  - (ohne Angabe): genau einmal
- **EMPTY** : Erzwingen von leerem Element

paper besteht aus  
mindestens einem author  
genau einem year und  
einem optionalen publisher  
in genau dieser Reihenfolge!

```

<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
  
```

**DTD**

## DTD – Deklaration von Elementen (5)

- Beschreibt die Einschränkungen des Inhalts eines Elements
- Syntax:  
<!ELEMENT Name (Definition)>
- Einziger atomarer Typ: **#PCDATA** (Parsed Character DATA)
- **(a,b,c)**: Liste von Unterelementen
- **(a|b|c)**: Alternativen
- Kardinalitäten:
  - \*  keinmal oder beliebig oft
  - +  einmal oder beliebig oft
  - ?  kein- oder einmal (optional)
  - (ohne Angabe): genau einmal
- **EMPTY** : Erzwingen von leerem Element

firstname ist vom Typ Zeichenkette

```

<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
  
```

**DTD**

## DTD – Deklaration von Attributen

- Name-Zeichenkettenwert-Paar
- Assoziiert mit einem Element
- Syntax:  

```
<!ATTLIST Element
  Attributname1 Typ1 Zusatz1
  Attributname2 ...>
```
- Typ:
  - **CDATA** Zeichenkette
  - **ID** OID
  - **IDREF** Referenzen
  - **IDREFS** Menge von Referenzen
- Zusatz:
  - **REQUIRED** zwingend
  - **IMPLIED** optional
  - (Initialwert)

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

**DTD**

## DTD – Deklaration von Attributen (2)

- Name-Zeichenkettenwert-Paar
- Assoziiert mit einem Element
- Syntax:  

```
<!ATTLIST Element
  Attributname1 Typ1 Zusatz1
  Attributname2 ...>
```
- Typ:
  - **CDATA** Zeichenkette
  - **ID** OID
  - **IDREF** Referenzen
  - **IDREFS** Menge von Referenzen
- Zusatz:
  - **REQUIRED** zwingend
  - **IMPLIED** optional
  - (Initialwert)

paper besitzt ein Attribut id, eine OID, die zwingend mit einem eindeutigen Wert belegt werden muss

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

**DTD**

## DTD – Deklaration von Attributen (3)

- Name-Zeichenkettenwert-Paar
- Assoziiert mit einem Element
- Syntax:  

```
<!ATTLIST Element
  Attributname1 Typ1 Zusatz1
  Attributname2 ...>
```
- Typ:
  - **CDATA** Zeichenkette
  - **ID** OID
  - **IDREF** Referenzen
  - **IDREFS** Menge von Referenzen
- Zusatz:
  - **REQUIRED** zwingend
  - **IMPLIED** optional
  - (Initialwert)

Ein author hat ein Attribut age, mit dem ihm eine Zeichenkette mit dem Wert für sein Alter zugewiesen werden kann (aber nicht muss!)

```
<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>
```

**DTD**

## DTD – OIDs und Referenzen

- DTDs erlauben die Deklaration von OIDs, Referenzen und Referenzmengen als Attribute
- Beispiel:

```

<family>
  <person id="jane" mother="mary" father="john">
    <name> Jane Doe </name>
  </person>
  <person id="john" children="jane jack">
    <name> John Doe </name>
  </person>
  <person id="mary" children="jane jack">
    <name> Mary Smith </name>
  </person>
  <person id="jack" mother="mary" father="john">
    <name> Jack Smith </name>
  </person>
</family>

```

**XML**

```

<!DOCTYPE family [
  <!ELEMENT family (person*)>
  <!ELEMENT person (name)>
  <!ELEMENT name (#PCDATA)>
  <!ATTLIST person
    id      ID      #REQUIRED
    mother IDREF   #IMPLIED
    father IDREF   #IMPLIED
    children IDREFS #IMPLIED>
]

```

**DTD**

## Bewertung von DTDs

- Zur Erinnerung: DTDs definieren kontextfreie Grammatiken
  - Rekursive Definitionen sind möglich

```

<!DOCTYPE paper [
  <!ELEMENT paper (section*)>
  <!ELEMENT section ((title, section*)|text)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT text (#PCDATA)>
]>

```

**DTD**

- DTDs weisen bei der Definition eines Schemas jedoch einige Schwächen auf:
  - Ungewollte Festlegung der Reihenfolge:
 

```
<!ELEMENT person ( name, phone ) >
```

    - **Workaround:**

```
<!ELEMENT person ( (name, phone) | (phone, name) ) >
```
  - Kann teilweise zu vage werden:
 

```
<!ELEMENT person ( ( name | phone | email ) * ) >
```
  - Referenzen können nicht eingeschränkt (typisiert) werden
  - Alle Elementnamen sind global in einem Namensraum

## XML-Schemata II: XML-Schema

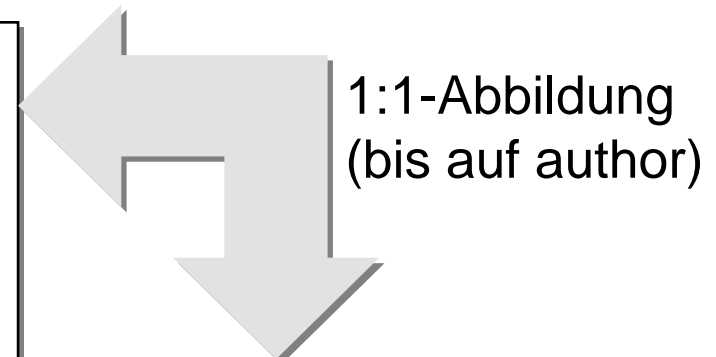
- Echter Schemamechanismus mit vielen Erweiterungen über DTDs hinaus
- Benutzt selbst wieder XML-Syntax zur Schemadefinition

```

<schema>
  <element name="bib">
    <complexType>
      <element name="paper" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <attribute name="id" type="ID" use="required"/>
          <sequence>
            <element name="author" type="authorType"
              minOccurs="0" maxOccurs="unbounded"/>
            <element name="year" type="string"/>
            <element name="publisher" type="string" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>
</schema>

```

**XML-Schema**



```

<!DOCTYPE bib [
  <!ELEMENT bib (paper*)>
  <!ELEMENT paper (author+, year, publisher?)>
  <!ATTLIST paper id ID #REQUIRED>
  <!ELEMENT author (firstname*, lastname)>
  <!ATTLIST author age CDATA #IMPLIED>
  <!ELEMENT firstname (#PCDATA)>
  <!ELEMENT lastname (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  ...
]>

```

**DTD**

## XML-Schema: Elemente

- Syntax: `<element name="Name"/>`
- Optionale Zusatzattribute:
  - Typ
    - **type = "Typ"**                      **atomarer, einfacher oder komplexer Typname**
  - Kardinalitäten (Vorgabe [1,1]):
    - **minOccurs = "x"**                       **$x \in \{ 0, 1, n \}$**
    - **maxOccurs = "y"**                       **$y \in \{ 1, n, \text{unbounded} \}$**
  - Wertvorgaben (schließen sich gegenseitig aus!):
    - **default = "v"**                      **veränderliche Vorgabe**
    - **fixed = "u"**                      **unveränderliche Vorgabe**
- Beispiele:
  - `<element name="bib"/>`
  - `<element name="paper" minOccurs="0" maxOccurs="unbounded"/>`
  - `<element name="publisher" type="string" minOccurs="0"/>`

## XML-Schema: Attribute

- Syntax: `<attribute name="Name"/>`
- Optionale Zusatzattribute:
  - Typ:
    - **type = "Typ"**
  - Existenz:
    - **use = "optional"** **Kardinalität [0,1]**
    - **use = "required"** **Kardinalität [1,1]**
  - Vorgabewerte:
    - **use = "default" value = "v"** **veränderliche Vorgabe**
    - **use = "fixed" value = "u"** **unveränderliche Vorgabe**
- Beispiele:
  - `<attribute name="id" type="ID" use="required"/>`
  - `<attribute name="age" type="string" use="optional"/>`
  - `<attribute name="language" type="string" use="default" value="de"/>`

## XML-Schema: Typen

- In XML-Schema wird zwischen atomaren, einfachen und komplexen Typen unterschieden
- Atomare Typen:
  - Eingebaute Elementartypen wie int oder string
- Einfache Typen:
  - Haben weder eingebettete Elemente noch Attribute
  - In der Regel von atomaren Typen abgeleitet
- Komplexe Typen:
  - Dürfen Elemente und Attribute besitzen
- Zusätzlich kann man noch folgende Unterscheidung treffen:
  - Reine Typdefinitionen beschreiben (wiederverwendbare) Typstruktur
  - Dokumentdefinitionen beschreiben welche Elemente wie im Dokument auftauchen dürfen

## XML-Schema: Atomare Typen

- XML-Schema unterstützt eine große Menge eingebauter Basistypen (>40):
  - Numerisch: byte, short, int, long, float, double, decimal, binary, ...
  - Zeitangaben: time, date, month, year, timeDuration, timePeriod, ...
  - Sonstige: string, boolean, uriReference, ID, ...
- Beispiele:

```
<element name="year" type="year"/>  
<element name="pages" type="positiveInteger"/>  
<attribute name="age" type="unsignedShort"/>
```

## XML-Schema: Einfache Typen

- Zusätzlich können von bestehenden Typen noch weitere, sog. einfache Typen, abgeleitet werden:
  - Typdefinition:  

```
<simpleType name="humanAge" base="unsignedShort">  
  <maxInclusive value="200"/> </simpleType>
```
  - Dokumentdefinition:  

```
<attribute name="age" type="humanAge"/>
```
- Solche einfachen Typen dürfen jedoch keine verschachtelten Elemente enthalten!
- In ähnlicher Weise können Listen definiert werden:
  - Typdefinition:  

```
<simpleType name="authorType" base="string"  
  derivedBy="list"/>
```

(Name eines Autors als mit Leerzeichen getrennte Liste von Zeichenketten)
  - Dokumentdefinition:  

```
<element name="author" type="authorType"/>
```

## XML-Schema: Komplexe Typen

- Komplexe Typen dürfen im Gegensatz zu einfachen Typen eingebettete Elemente und Attribute besitzen
- Beispiel:
  - Typdefinition:

```
<complexType name="authorType">  
  <sequence>  
    <element name="firstname" type="string" minOccurs="0"  
      maxOccurs="unbounded"/>  
    <element name="lastname" type="string"/>  
  </sequence>  
  <attribute name="age" type="string" use="optional"/>  
</complexType>
```
- Gruppierungs-Bezeichner:
  - `<sequence> ... </sequence>` Feste Reihenfolge (a,b)
  - `<all> ... </all>` Beliebige Reihenfolge (a,b oder b,a)
  - `<choice> ... </choice>` Auswahl (entweder a oder b)

# XML-Schema: Komplexe Typen

```
<complexType name="authorType">
  <sequence>
    <element name="firstname" type="string" minOccurs="0"
      maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
  <attribute name="age" type="string" use="optional"/>
</complexType>
```

... Grundlage für weitere Beispiele!

# Typhierarchien

- Gesetzmäßigkeit zwischen zwei Typen
- Typdefinition durch
  - Erweiterung (engl. extension) oder
  - Restriktion (engl. restriction) einer bestehenden Typdefinition
- Alle Typen in XML-Schema sind entweder
  - Atomare Typen (z.B. string) oder
  - Erweiterung bzw. Restriktion bestehender Typen
- Alle Typen bilden eine Typhierarchie
  - Baum mit Wurzel: Typ Zeichenkette
  - Keine Mehrfachvererbung
- Typen sind entlang der Typhierarchie abwärtskompatibel:
  - Für Typinstanzen gilt das Substituierbarkeitsprinzip
  - Elemente eines bestimmten Typs akzeptieren auch Daten einer Erweiterung oder Restriktion des geforderten Typs

## Typhierarchien: Erweiterung von Typen

- Typen können konstruktiv um weitere Elemente oder Attribute zu neuen Typen erweitert werden
- Beispiel:

```
<complexType name="extendedAuthorType">  
  <extension base="authorType">  
    <sequence>  
      <element name="email" type="string" minOccurs="0"  
        maxOccurs="1"/>  
    </sequence>  
    <attribute name="homepage" type="string" use="optional"/>  
  </extension>  
</complexType>
```
- Erweitert den zuvor definierten Typ `authorType` um
  - Ein optionales Element `email`
  - Ein optionales Attribut `homepage`

## Typhierarchien: Restriktion von Typen

- Typen werden durch Verschärfung von Zusatzangaben bei Typdefinitionen in ihrer Wertemenge eingeschränkt
- Beispiele für Restriktionen:
  - Bisher nicht angegebene type-, default- oder fixed-Attribute
  - Verschärfung der Kardinalitäten minOccurs, maxOccurs
- Substituierbarkeit
  - Menge der Instanzen des eingeschränkten Untertyps muß immer eine Teilmenge des Obertyps sein!
- Restriktion komplexer Typen
  - Struktur bleibt gleich: es dürfen keine Elemente oder Attribute weggelassen werden
- Restriktion einfacher Typen
  - Restriktion ist (im Gegensatz zur Erweiterung) auch bei einfachen Typen erlaubt

## Typhierarchien: Restriktion von Typen (2)

- Beispiel (Komplexer Typ):

```
<complexType name="restrictedAuthorType">  
  <restriction base="authorType">  
    <sequence>  
      <element name="firstname" type="string" minOccurs="0"  
        maxOccurs="2"/> Vorher: maxOccurs="unbounded"  
      <element name="lastname" type="string"/>  
    </sequence>  
    <attribute name="age" type="string" use="required"/>  
  </restriction>  
</complexType> Vorher: use="optional"
```

Gegenüber dem ursprünglichen Typ wurde die Anzahl der Vornamen (firstname) auf 2 begrenzt und das Altersattribut (age) erzwungen

# Polymorphe Konsistenzbedingungen

- Elemente und Attribute können zusätzlich mit Konsistenzbedingungen belegt werden

- Eindeutigkeit (Schlüsselkandidat):

```
<unique name="eindeutigerAutorenName">  
  <field xpath="bib/paper/author/@firstname"/>  
  <field xpath="bib/paper/author/@lastname"/>  
</unique>
```

Die Kombinationen von Vor- und Nachnamen bei den Autoren sind immer eindeutig

- Mittels field werden die entsprechenden Elemente oder Attribute identifiziert
- Mittels xpath wird der genaue Pfadausdruck angegeben, unter dem das entsprechende field innerhalb der Dokumenthierarchie gefunden werden kann
- XPath (xpath): XML-Standard für Pfadausdrücke

## Polymorphe Konsistenzbedingungen (2)

- Schlüsselbedingung:

```
<key name="papierSchlüssel">  
  <field xpath="bib/paper/@id"/>  
</key>
```

Das Attribut id in paper dient als Schlüssel

- Schlüssel sind eindeutig und können referenziert werden

- Fremdschlüsselbedingung:

```
<keyref name="papierFremdschlüssel" refer="papierSchlüssel">  
  <field xpath="bib/paper/@references"/>  
</keyref>
```

Ergänzend zum bisherigen Schema: references ist eine Liste von Papieren, die vom Papier aus referenziert werden, d.h. in dessen Literaturverzeichnis auftauchen.

- Erst jetzt macht name ein Sinn: mit refer bezieht man sich auf das name-Attribut einer Schlüsselbedingung, nicht auf das Schlüsselfeld!
- Die Werte in references müssen also immer unter den Schlüsseln zu den Papieren zu finden sein

## Beispiel-Schema (Hausaufgabe)

```
<!-- XMLSchema für eine Literaturdatenbank -->
<schema>

  <!-- Globales Wurzelement bib -->
  <element name="bib">
    <complexType>
      <element name="paper" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <attribute name="id" type="ID" use="required"/>
          <sequence>
            <element name="author" type="authorType" maxOccurs="unbounded"/>
            <element name="year" type="string"/>
            <element name="publisher" type="string" minOccurs="0"/>
            <element name="references" type="listOfPapers" minOccurs="0"/>
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>

  <!-- Liste von Verweisen auf Papiere (siehe bib/paper/@references) -->
  <simpleType name="listOfPapers" base="ID" derivedBy="list"/>
```

## Beispiel-Schema (2)

```
<!-- Reine Typdefinitionen -->
<complexType name="authorType">
  <sequence>
    <element name="firstname" type="string" minOccurs="0" maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
  </sequence>
  <attribute name="age" type="humanAge" use="optional"/>
</complexType>

<simpleType name="humanAge" base="unsignedShort">
  <maxInclusive value="200"/> </simpleType>

<complexType name="extendedAuthorType">
  <extension base="authorType">
    <sequence>
      <element name="email" type="string" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="homepage" type="simpleURLType" use="optional"/>
  </extension>
</complexType>

<simpleType name="simpleURLType" base="string">
  <pattern value="http://(?:[^\?#]*)?(?:[^\?#]*)(?:[^\?#]*)?(#(?:.)*?)?"/> </simpleType>
```

## Beispiel-Schema (3)

```
<!-- Konsistenzbedingungen -->
```

```
<unique name="eindeutigerAutorenName">  
  <field xpath="bib/paper/author/@firstname"/>  
  <field xpath="bib/paper/author/@lastname"/>  
</unique>
```

```
<key name="papierSchlüssel">  
  <field xpath="bib/paper/@id"/>  
</key>
```

```
<keyref name="papierFremdschlüssel" refer="papierSchlüssel">  
  <field xpath="bib/paper/@references"/>  
</keyref>
```

```
</schema>
```

## Bewertung von XML-Schema

- Mit XML-Schema können Datenbasis-Schemata viel einfacher als mit DTDs spezifiziert werden
  - Es kann viel mehr Semantik in einem Schema eingefangen werden als mit DTDs
- XML-Schema wird jedoch noch nicht weitläufig unterstützt (Tendenz steigend), während DTDs bereits zum Kernstandard von XML gehören
- Syntax und Ausdruckskraft von XML-Schema sind sehr umfangreich
  - Einzige Schwäche: geringe Vielfalt bei Konsistenzbedingungen
- Mehr zu XML-Schema im Web:
  - <http://www.w3.org/TR/xmlschema-0/> Einführung
  - <http://www.w3.org/TR/xmlschema-1/> Teil I: Strukturen
  - <http://www.w3.org/TR/xmlschema-2/> Teil II: Datentypen

# XML Namespaces

**Namespaces**

## XML Namespaces and Programming-Language Modules

- XML namespaces are akin to namespaces, packages, and modules in programming languages
- Disambiguation of tag names from different XML applications (“spaces”) through different prefixes
- A *prefix* is separated from the local *name* by a “:”, obtaining *prefix:name* tags
- Namespaces constitute a layer on top of XML 1.0, since *prefix:name* is again a valid tag name and namespace bindings are ignored by some tools (“flat namespaces”)

## Namespace Bindings

- Prefixes are bound to namespace URIs by attaching an `xmlns:prefix` attribute to the prefixed element or one of its ancestors, `prefix:name1`, ..., `prefix:namen`
- The value of the `xmlns:prefix` attribute is a URI, which may or (unlike for DTDs!) may not point to a description of the namespace's syntax
- An element can use bindings for multiple name-spaces via attributes `xmlns:prefix1`, ..., `xmlns:prefixm`

## Namespaceless Example: Address Variant

### Namespaceless XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
  <bill>12.50</bill>
  <phone>030/1234567</phone>
  <phone>030/1234568</phone>
  <fax>030/1234569</fax>
  <bill>76.20</bill>
</ address>
```

*bill is ambiguous  
tag (name clash  
from two XML  
applications)*

## Two-Namespace Example: Snail-Mail and Telecoms Address Parts

### Namespace XML Markup:

```
<mail:address xmlns:mail="http://www.deutschepost.de/"
              xmlns:tele="http://www.telekom.de/">
  <mail:name>Xaver M. Linde</mail:name>
  <mail:street>Wikingerufer 7</mail:street>
  <mail:town>10555 Berlin</mail:town>
  <mail:bill>12.50</mail:bill>
  <tele:phone>030/1234567</tele:phone>
  <tele:phone>030/1234568</tele:phone>
  <tele:fax>030/1234569</tele:fax>
  <tele:bill>76.20</tele:bill>
</ mail:address>
```

*bill disambiguation  
through mail and  
tele prefixes*

- The root element, mail:address, as well as the children mail:name, mail:street, mail:town, and mail:bill, use the mail prefix, bound to a deutschepost URI
- The tele:phone, tele:fax, and tele:bill children use the tele prefix, bound to a telekom URI

# Processing XML

**Processing**

## Address Example: XML to External

### XML Markup:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
</address>
```

*XML stylesheets are,  
e.g., usable to generate  
different presentations*

### External Presentations:

*Xaver M. Linde*  
Wikingerufer 7  
**10555 Berlin**

...

**Xaver M. Linde**  
Wikingerufer 7  
10555 Berlin

X  
M  
L

# Address Example: XML to XML

## XML Markup 1:

```
<address>
  <name>Xaver M. Linde</name>
  <street>Wikingerufer 7</street>
  <town>10555 Berlin</town>
</address>
```

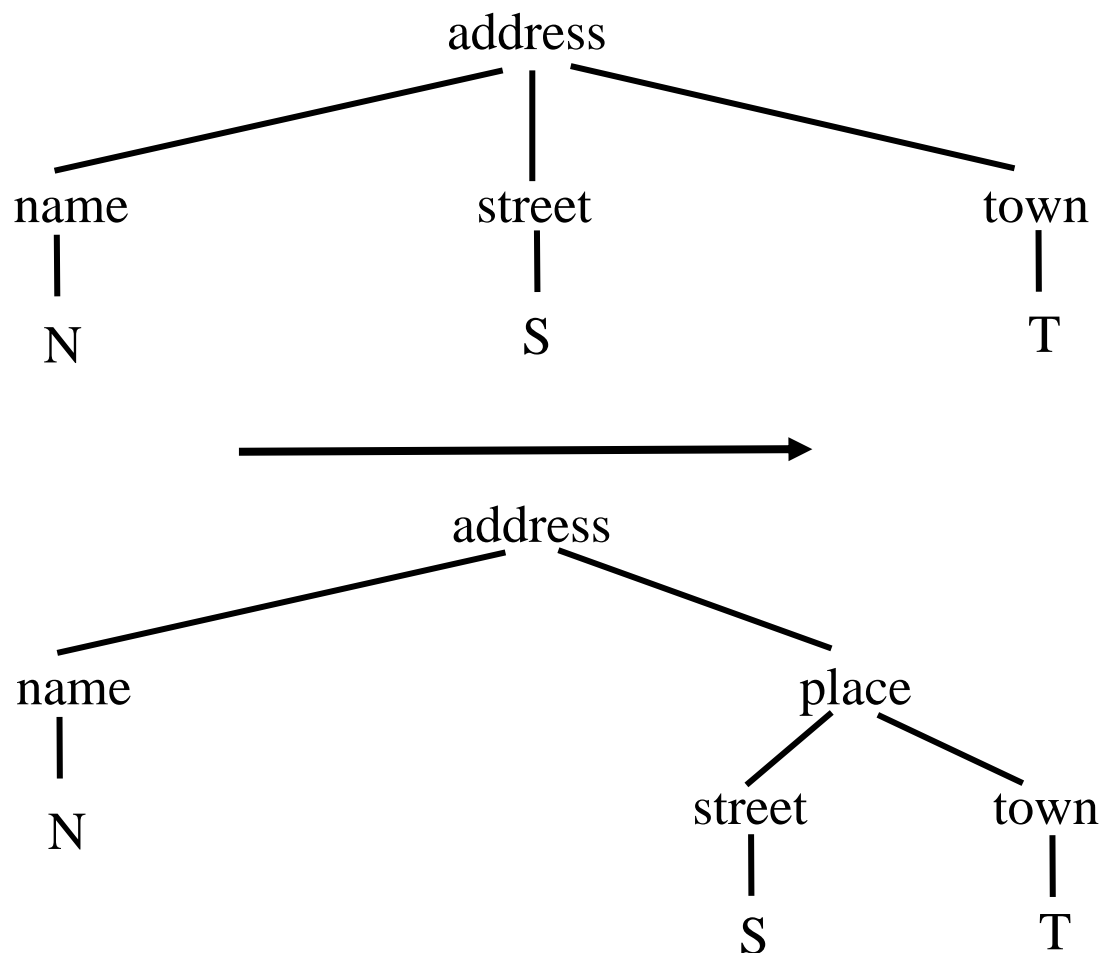
## XML Markup 2:

```
<address>
  <name>Xaver M. Linde</name>
  <place>
    <street>Wikingerufer 7</street>
    <town>10555 Berlin</town>
  </place>
</address>
```

*XML stylesheets are also usable to transform XML representations*

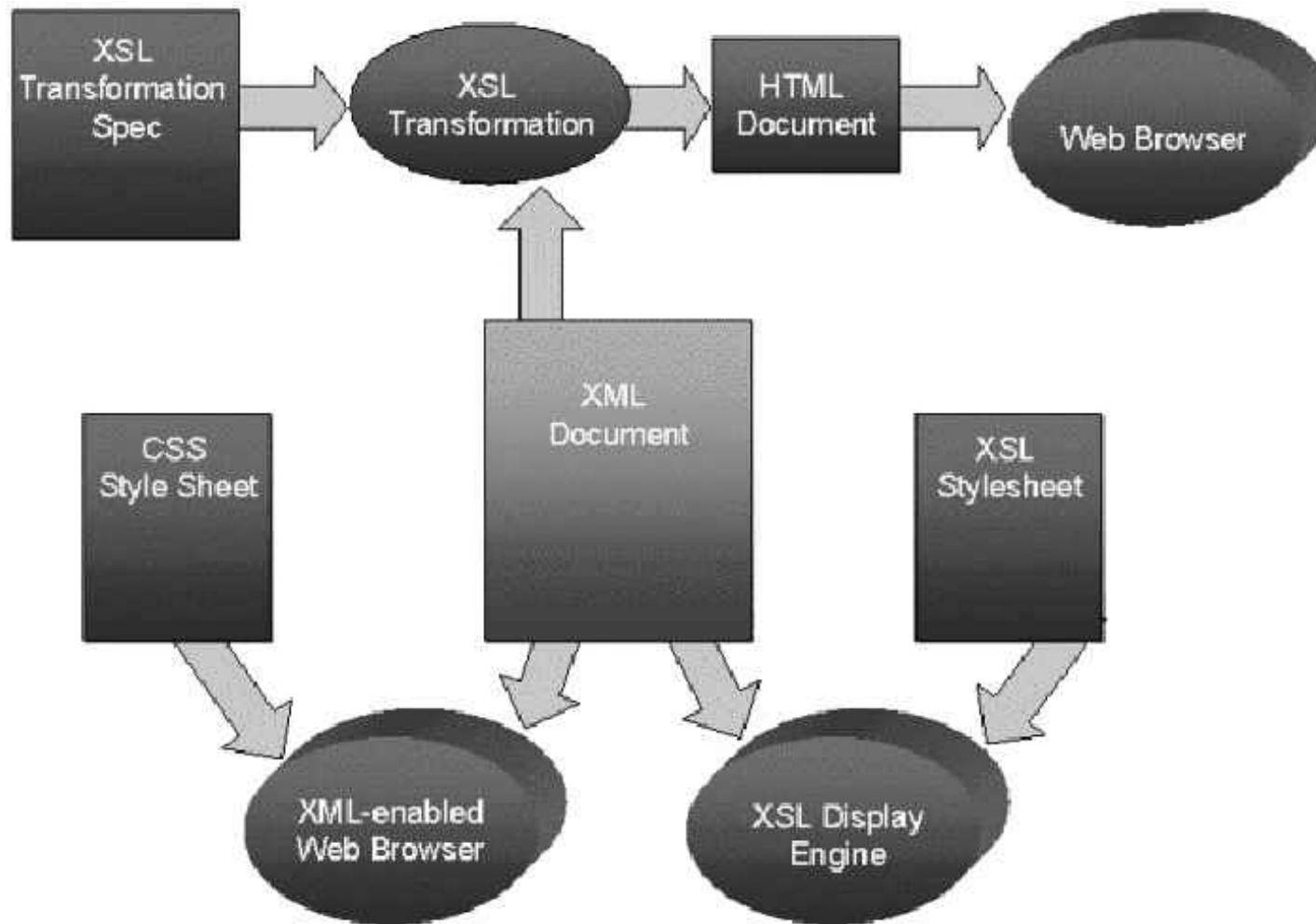
X  
M  
L

# Address Example: Some Stylesheets Will Contain Term-(Tree-)Rewriting Rules



X  
M  
L

# Displaying XML



# Cascading Style Sheets

- CSS is a language for applying styles such as *bold* and *Arial* to XML elements
- also works with HTML (supported in most modern browsers)

- example (style sheet for poems):

```
poem    { display: block }
title   { display: block; font-size: 16pt; font-weight: bold}
poet    { display: block; margin-bottom: 10px }
stanza  { display: block; margin-bottom: 10px }
verse   { display: block }
```

- attached to XML documents with processing instruction:  
<?xml-stylesheet type="text/css" href= "poem.css"?>

## XSLT (XSL Transformations)

- XSL (Extensible Stylesheet Language) =  
*XSLT* (including XPath) + Formatting Objects

**X**  
**S**  
**L**  
**T**

- XML Path Language - XPath: a language for referencing parts of an XML document
- XSLT is a rule-based transformation language for XML documents
- result of a transformation usually is again an XML document, but may also be HTML or plain text
- transformation takes place
  - on server (e.g., with Apache's Cocoon)
  - in client (browser, e.g. Netscape 6 or IE 5)
  - offline (useful for XML-to-HTML transformation)

# XSL capabilities

- A stylesheet specifies the presentation of an XML document. It allows
  - the transformation of the input document into another structure,
  - the description of how to present information.
- Transformation capabilities include:
  - generation of constant text
  - suppression of content
  - moving text
  - duplicating text
  - sorting
  - more complex transformations that compute new from existing information.

## XSLT Example – Input

X  
S  
L  
T

```
<addresses>

  <address>
    <name>Xaver M. Linde</name>
    <street>Wikingenufer 7</street>
    <town>10555 Berlin</town>
  </address>

  <address>
    <name>John Doe</name>
    <street>42 Gary Cooper Street</street>
    <town>Stanwyck City</town>
  </address>

</addresses>
```

# XSLT Example – Stylesheet

X  
S  
L  
T

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">  
  <html>  
    <head><title>Addresses</title></head>  
    <body bgcolor="white">  
      <xsl:apply-templates/>  
    </body>  
  </html>  
</xsl:template>
```

template for  
document root

```
<xsl:template match="address">  
  <p>  
    <i><xsl:value-of select="name"/></i><br/>  
    <xsl:value-of select="street"/><br/>  
    <b><xsl:value-of select="town"/></b>  
  </p>  
</xsl:template>
```

template for  
address  
elements

```
</xsl:stylesheet>
```

## XSLT Example – Output

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">
```

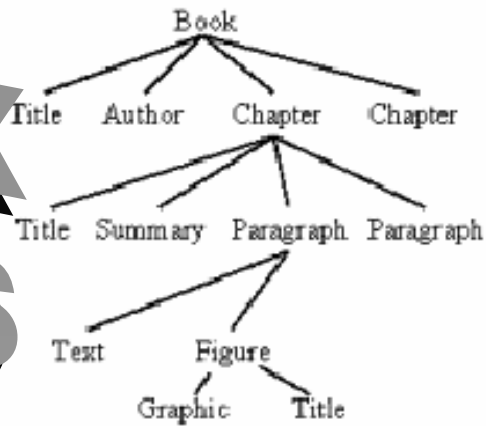
```
<html>
<head><title>Addresses</title></head>
<body bgcolor="white">
  <p><i>Xaver M. Linde</i><br>
    Wikingerufer 7<br>
    <b>10555 Berlin</b>
  </p>
  <p><i>John Doe</i><br>
    42 Gary Cooper Street<br>
    <b>Stanwyck City</b>
  </p>
</body>
</html>
```

(The HTML code was produced with Apache's Cocoon in HTML mode, hence <br/> became <br>)

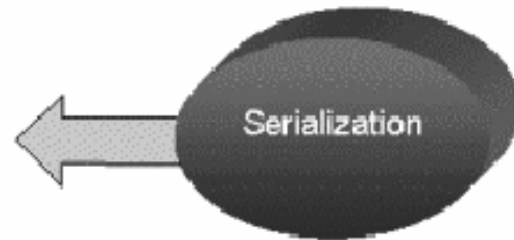
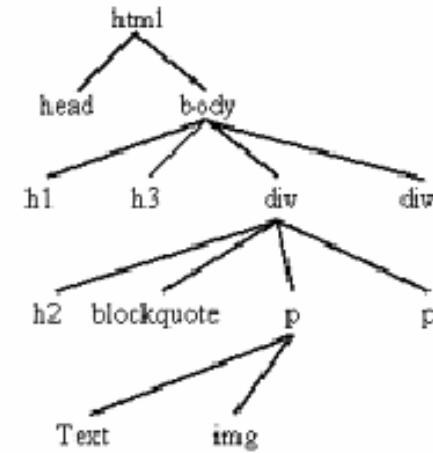


# XHTML Output

XML Source Tree



XHTML Result Tree



```

<html>
<head>...</head>
<body>
<h1></h1>
<h3></h3>
.....
</body>
</html>
    
```

X  
S  
L  
T

# XPath

# X P a t h

- XPath is a language for addressing parts of an XML document. It also has basic facilities for manipulation of strings, numbers and booleans.
- It has a compact non-XML syntax.
- It operates on the logical (tree-like) structure of an XML document and uses a path notation for navigating through the hierarchy (very much like navigating a traditional file system)
- It models an XML document as a tree of nodes: element nodes, attribute nodes, text nodes, ...

## Xpath - Location Paths

X  
P  
a  
t  
h

- **para**  
Selects all <para> children in the current context
- **para/emphasis**  
Selects all <emphasis> elements that have a parent of <para>
- **/**  
Selects the root of the document
- **para//emphasis**  
Selects all <emphasis> elements that have an ancestor of <para>
- **section/para[1]**  
Selects the first <para> child of all the <section> children in the current context

## More location paths

X  
P  
a  
t  
h

- //title  
Selects all <title> elements anywhere in the document
- section/\*/note  
Selects <note> elements that have <section> grandparents.
- stockquote[@symbol]  
Selects <stockquote> elements that have a "symbol" attribute
- stockquote[@symbol="XXXX"]  
Selects <stockquote> elements that have a "symbol" attribute with the value "XXXX"
- emphasis|strong  
Selects <emphasis> or <strong> elements

# Xpath Addressing

- An XPath pattern is a 'location path'. A location path is
  - absolute : begins with a slash ("/")
  - relative otherwise.
- A relative location path consists of a series of steps, separated by slashes.
- A **step** has three parts:
  - an axis specifier, which specifies the tree relationship between the nodes to be selected and the context node,
  - a node test, which specifies the node type of the nodes to be selected, and
  - a predicate, to further refine the set of nodes to be selected.
- Syntax for step: `axisname::nodetest[predicate]`
- Example: **`child::para/descendant::emphasis[1]`**  
Selects the first <emphasis> element which is the descendant of a <para> element being a child of the current node

X  
P  
a  
t  
h

## Axis of a Node Test

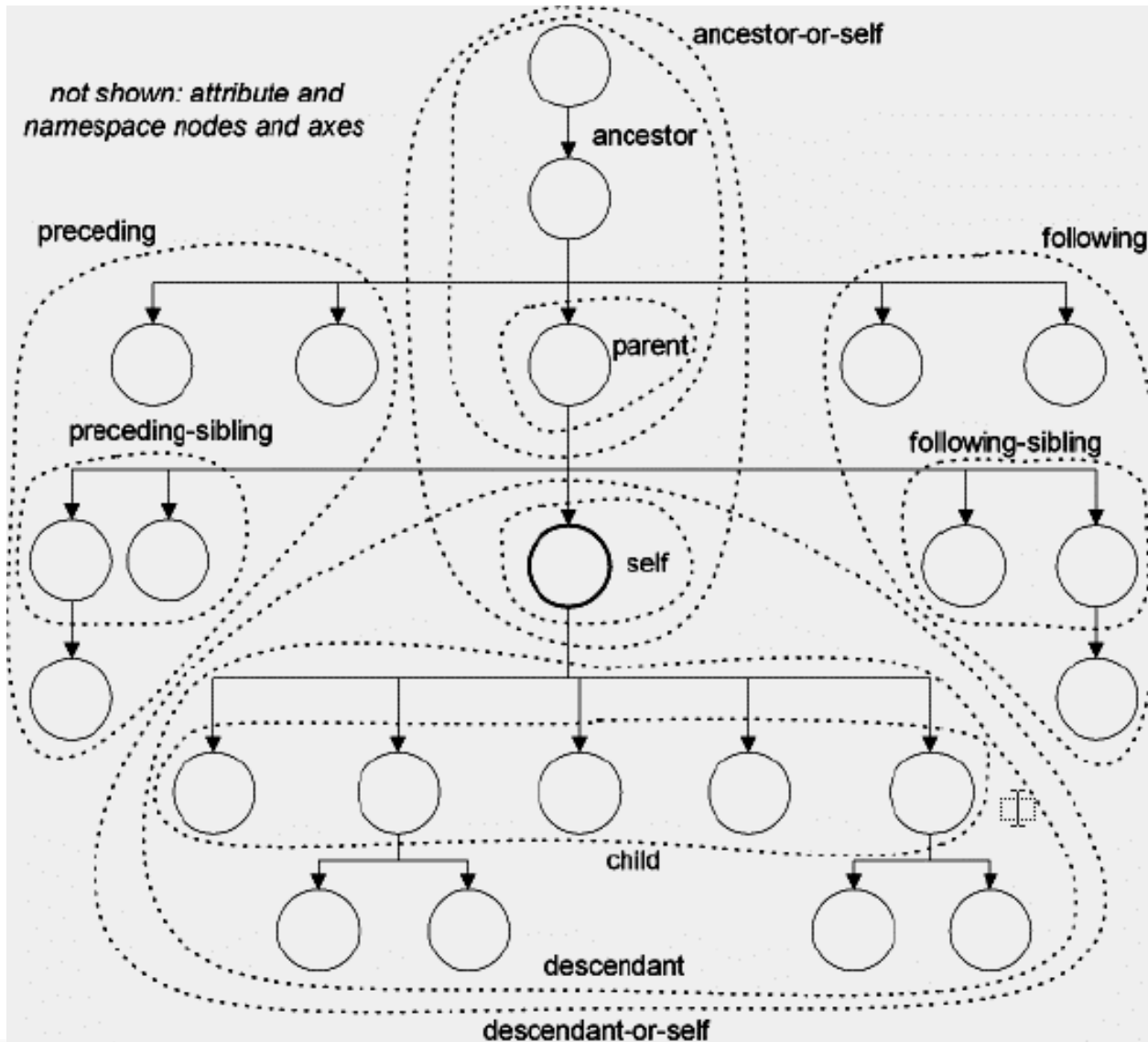
The axis of a node test determines what general category of nodes may be considered for the following node test. I.e., an axis defines a node-set relative to the current node.

X  
P  
a  
t  
h

There are thirteen axes:

- **parent, ancestor, ancestor-or-self**
  - selects ancestors of the current node
- **attribute** : selects Attributes of the current node (abbreviated "@")
- **child, descendant, descendant-or-self**
  - selects children of the current node (the default axis)
- **following / following-sibling, preceding / preceding-sibling**
  - selects siblings of the current node
- **namespace** : selects namespace nodes of the current node
- **self** : selects the current node (abbreviated ".")

X  
P  
B  
T  
S



## Node tests

Node tests are most frequently element names, but other node tests are possible:

X  
P  
a  
t  
h

- **name** : Matches <name> element nodes
- \* : Matches any element node
- **namespace:name**
  - Matches <name> element nodes in the specified namespace
- **namespace:\***
  - Matches any element node in the specified namespace
- **comment(), text(), processing-instruction(), node()**
  - Match comment nodes, text nodes, processing instructions, any node, respectively.

# Predicates

Predicates occur after the node test in square brackets.

A wide range of expressions are possible.

X  
P  
a  
t  
h

- **nodetest[1]** : Matches the first node satisfying the nodetest.
- **nodetest[position()=last()]** : Matches the last node
- **nodetest[position() mod 2 =0]**: Matches even nodes
- **element[@id='foo']**
  - Matches the element(s) with id attributes whose value is "foo"
- **element[not(@id)]**
  - Matches elements that don't have an id attribute
- **author[firstname="Norman"]**
  - Matches <author> elements that have <firstname> children with the content "Norman".

## (One) Layer Model of the Semantic Web

