XML

XML

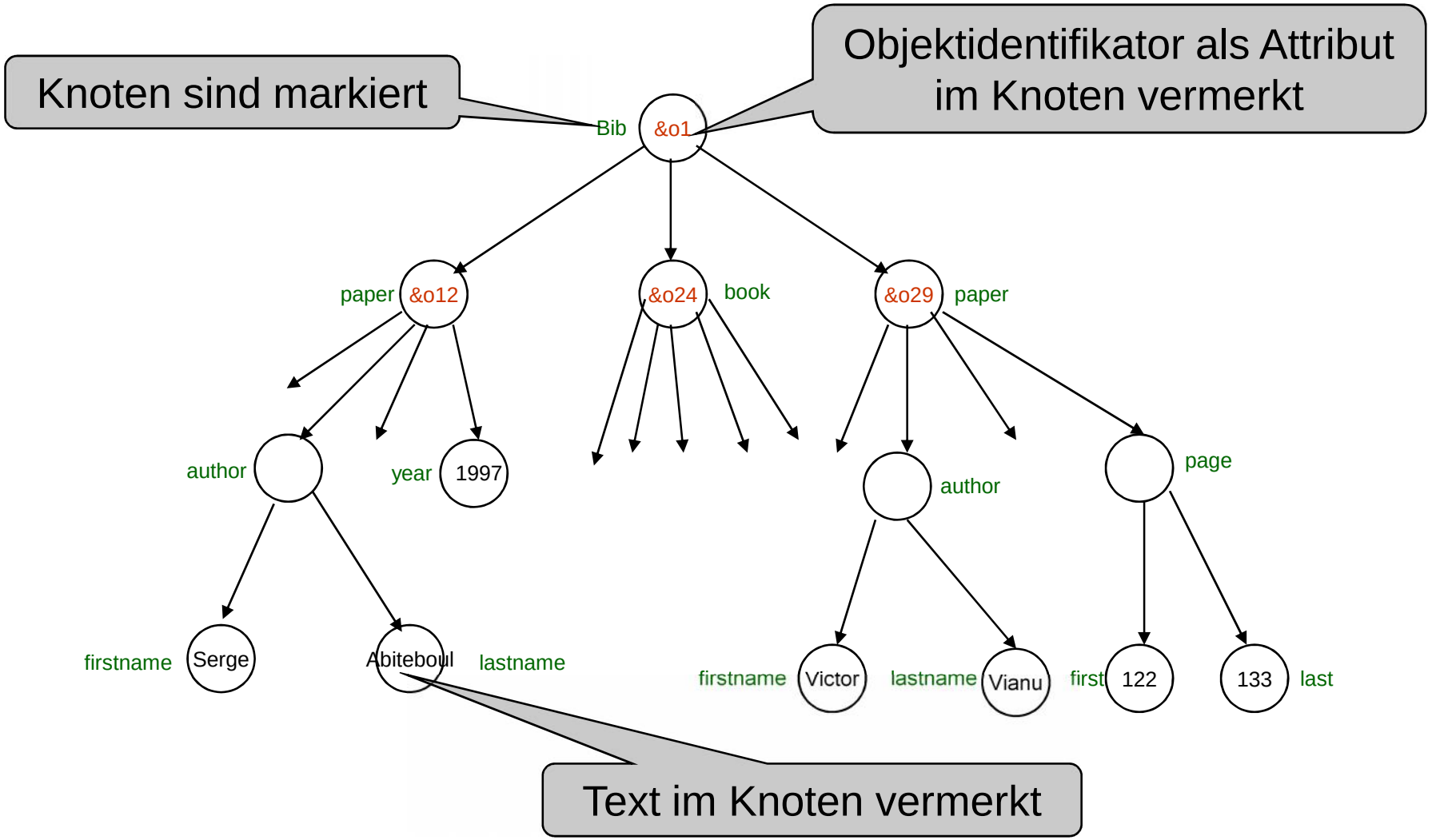
- eXtended Markup Language
- Ursprung: strukturierter Text (HTML4.0 $\in$ XML $\subset$ SGML)
- Web-Standard (W3C) zum Datenaustausch:
  - Ein- und Ausgabedaten von Anwendungen können mittels XML beschrieben werden
  - Industrie muss sich nur noch auf standardisierte Beschreibung einigen
- Komplementärsprache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

SGML: Standard Generalized Markup Language

HTML: Hypertext Markup Language

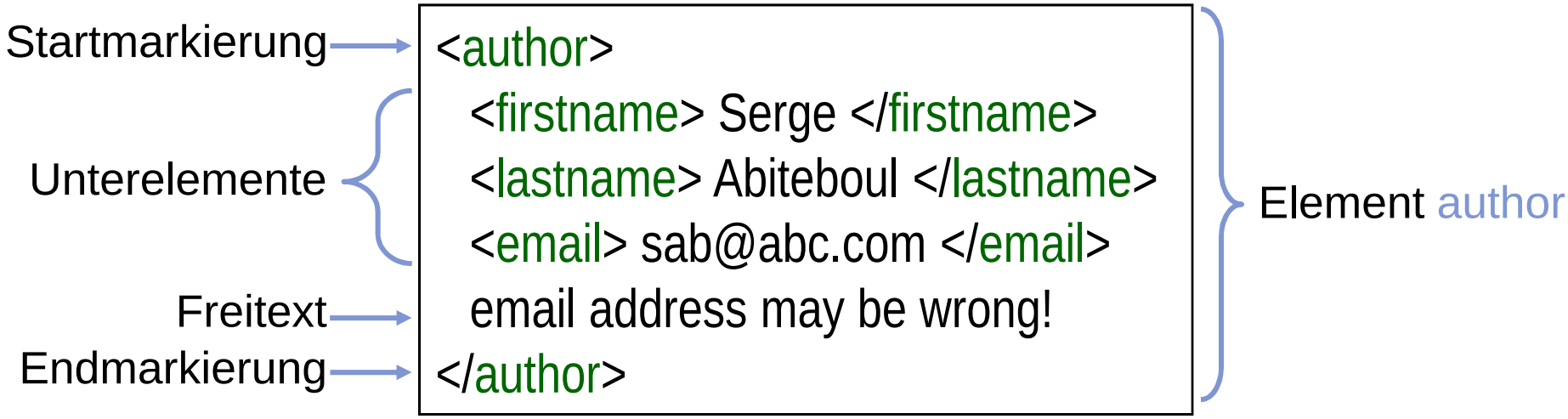
XML-Modell

Veranschaulichung von Objekten als gerichtete Graphen



XML-Syntax (1) – XML-Element

- XML-Element (engl. element):
  - Beschreibung eines Objekts, das 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

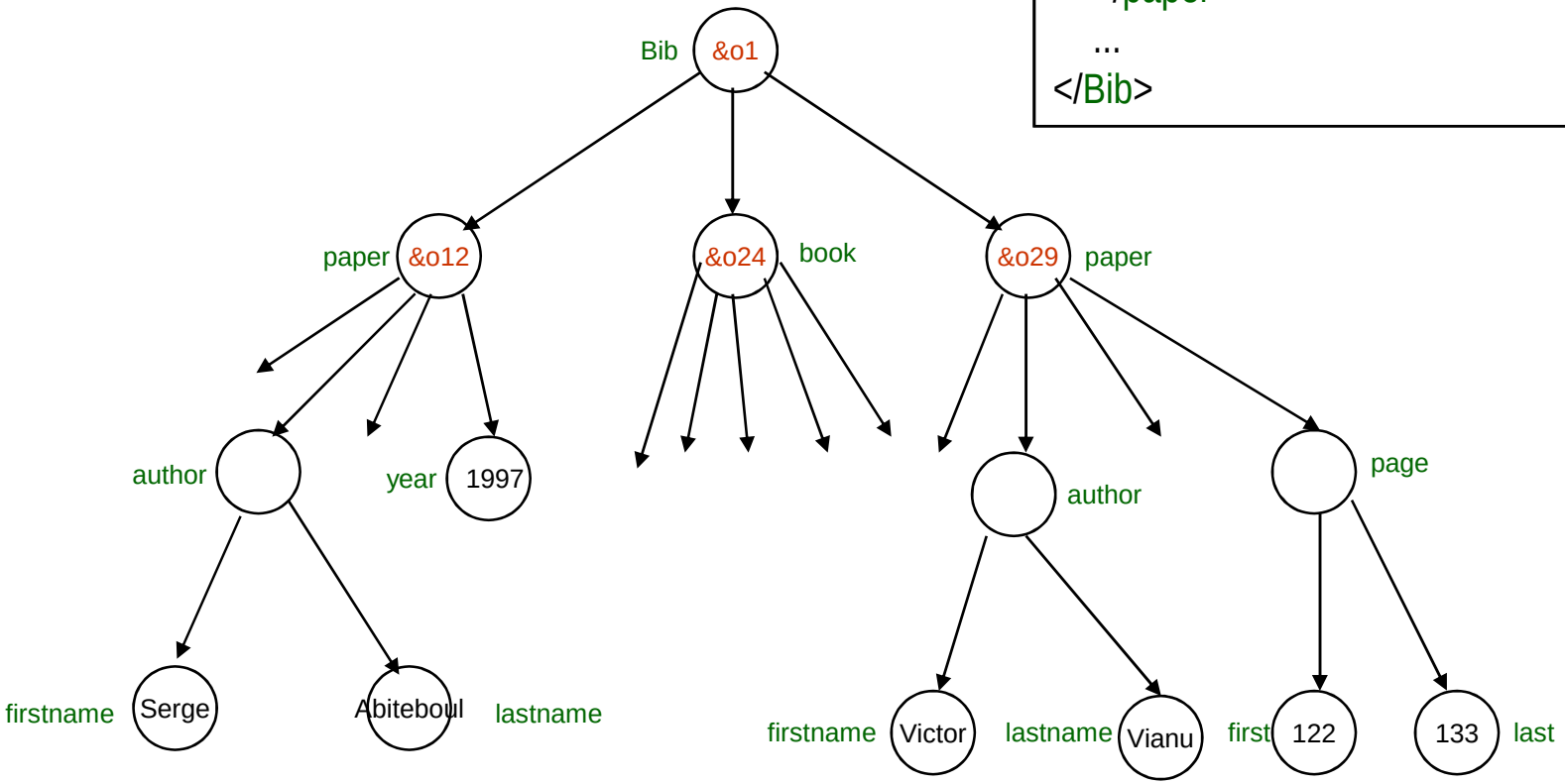
```
<author email="sab@abc.com">
  <firstname> Serge </firstname>
  <lastname> Abiteboul </lastname>
</author>
```

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⊂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 Wurzelelement
  - 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)

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
  - 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 Wurzelelements 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 vomTyp 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

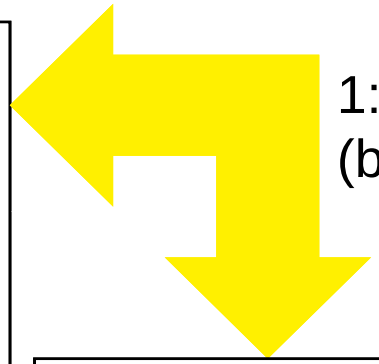
- DTDs weisen bei der Definition eines Schemas jedoch einige Schwächen auf:
  - Ungewollte Festlegung der Reihenfolge:
    <!ELEMENT person (name, phone) >
    - Workaround: <!ELEMENT person ((name, phone) | (phone, name)) >
  - Kann teilweise zu vage werden:
    <!ELEMENT person ((name | phone | email)*) >
  - 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"
       maxOccurs="unbounded"/>
      <element name="year" type="string"/>
      <element name="publisher" type="string" minOccurs="0"/>
     </sequence>
    </complexType>
   </element>
  </complexType>
 </element>
</schema>
```

XML-Schema



1:1-Abbildung
(bis auf author)

```
<!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 Č { 0, 1, n }
    - maxOccurs = "y" y Č { 1, n, 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)
  - Dokumentdefiniton:
    <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)

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: Erweiterung von Typen (2)

- Die Erweiterungen werden an die bestehenden Definitionen angehängt:

```
<complexType name="extendedAuthorType">
   <sequence>
    <element name="firstname" type="string" minOccurs="0"
          maxOccurs="unbounded"/>
    <element name="lastname" type="string"/>
    <element name="email" type="string" minOccurs="0"
          maxOccurs="1"/>
   </sequence>
   <attribute name="age" type="string" use="optional"/>
   <attribute name="homepage" type="string" use="optional"/>
</complexType>
```

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"/>
      <element name="lastname" type="string" />
    </sequence>
    <attribute name="age" type="string" use="required"/>
  </restriction>
</complexType>
```

Vorher: maxOccurs="unbounded"

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 einen 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

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

  <!-- Globales Wurzelelement 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)

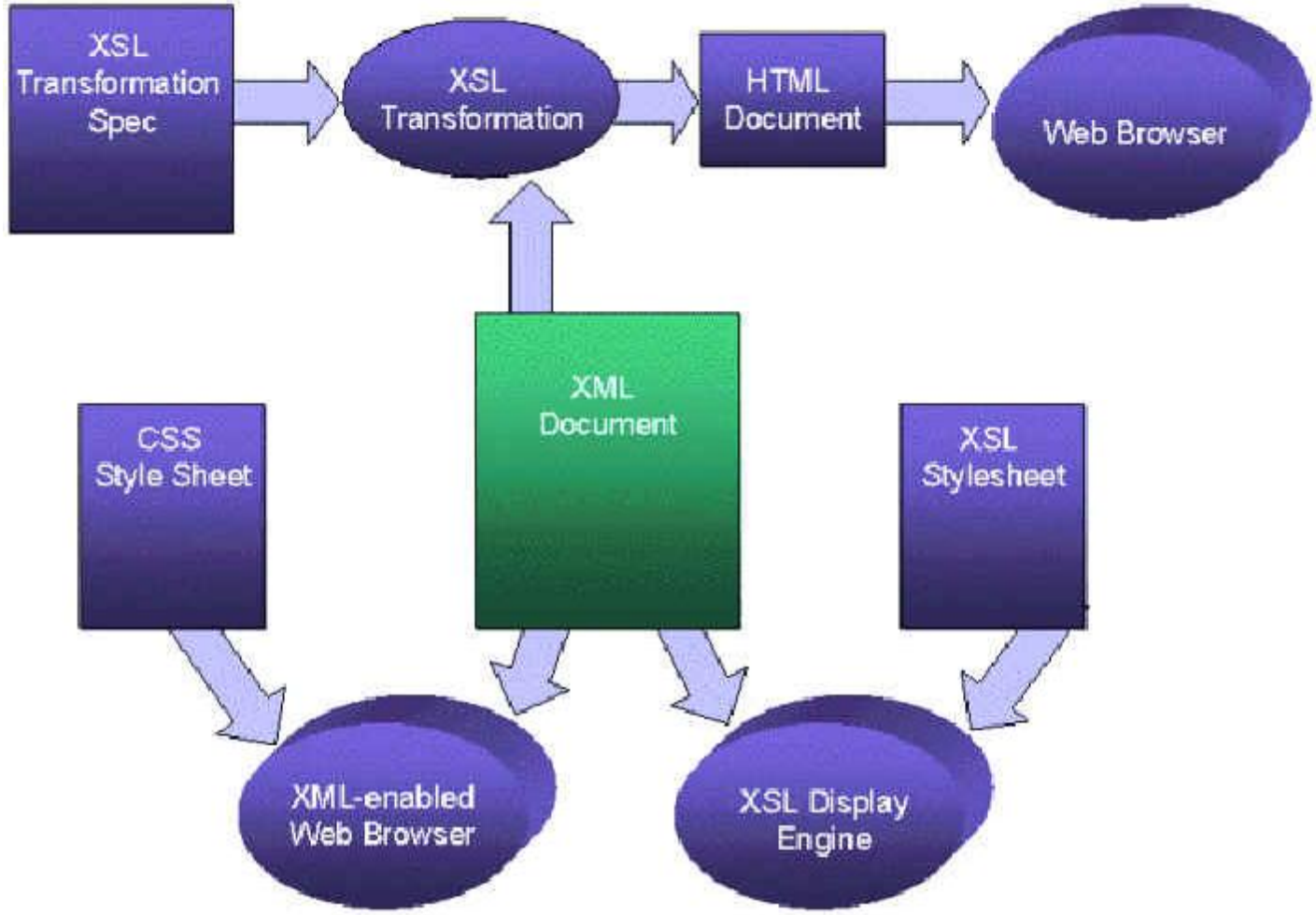
```
  <!-- Konsistenzbedingungen -->

  <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
- DTDs die zum Kernstandard von XML gehören werden durch XML-Schema mehr und mehr ersetzt
- 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

Darstellung von XML Dokumenten



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
- 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 a 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 or Internet Explorer)
  - offline (useful for XML-to-HTML transformation)

XSLT Example – Input

```
<addresses>

  <address>
    <name>Xaver M. Linde</name>
    <street>Wikingerufer 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

```
<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>

<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>

</xsl:stylesheet>
```

template for document root

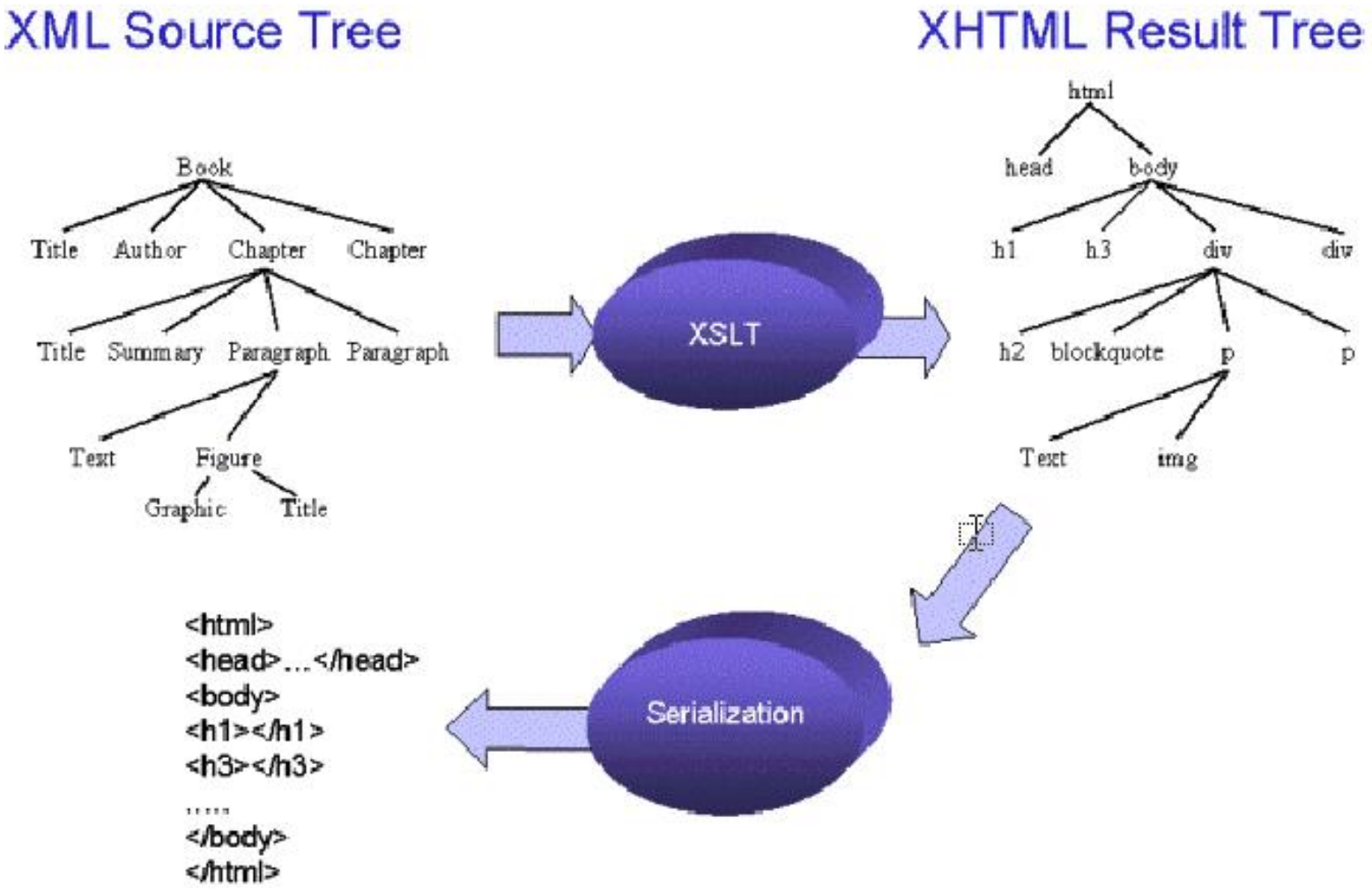
template for address elements

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
 became
)*

XHTML Output



XQL and XPath

- XSLT uses XPath to select parts of the input XML documents, e.g.:
  - <xsl:template match="/"> *document root*
  - <xsl:value-of select="name"/> *element „name“*
- Syntax is not XML-based, but intended to be used in URLs (XPointer) and XSLT as above
- XQL is a variant of XPath especially designed for querying XML documents
- XPath/XQL expressions *select parts* of XML documents (no transformations!)

XQL Expressions 1

- XQL expressions (relative to a context node):
  - address
    select element nodes with tag name „address“
  - address/name
    select „name“ element nodes *directly* below „address“ element nodes
  - address/name=´John Doe´
    as before, plus the content of the name element must be ´John Doe´
  - book//name
    select „name“ element nodes *anywhere* below „book“ element nodes

XQL Expressions 2

- Absolute XQL expressions:
  - /
    document root (the node that contains the document element)
  - /addresses/address/name
  - //name
    „name“ element nodes anywhere in the document
- Expressions with attributes (prefixed with @):
  - address/@type
    attribute nodes with name „type“ below „address“ element nodes
  - address/@type=´email´
    as above, plus the value of the „type“ attribute is ´email´

XQL Expressions 3

- filter expressions:
  - address[name]
    select „address“ element nodes with a direct „name“ sub-element node
  - address[name=´John Doe´]
    as above, plus the content of the „name“ element is ´John Doe´
  - address[@type=´email´]
    select „address“ element nodes that have a „type“ attribute whose value is ´email´

Anfragesprachen für XML

- Für XML: XML-QL
  - Vorschlag an W3C
  - von Datenbankfachleuten für den Umgang mit großen Datenbeständen – ähnlich zu SQL
  - basiert auf OEM (Object Exchange Model)
  - bietet:
    - Reguläre Pfadausdrücke
    - Muster
- Für XML: XQL
  - Eher aus Dokumentverarbeitungssicht
- Viele andere vorgeschlagen...

XML-QL (1)

Grundmuster:

where muster

in quelle, ...

construct ergebnis

Baumstruktur von Pfadausdrücken mit Tags, die mit dem Graphen des Dokuments quelle verglichen wird. Die freien Variablen werden aus dem Dokument instanziert.

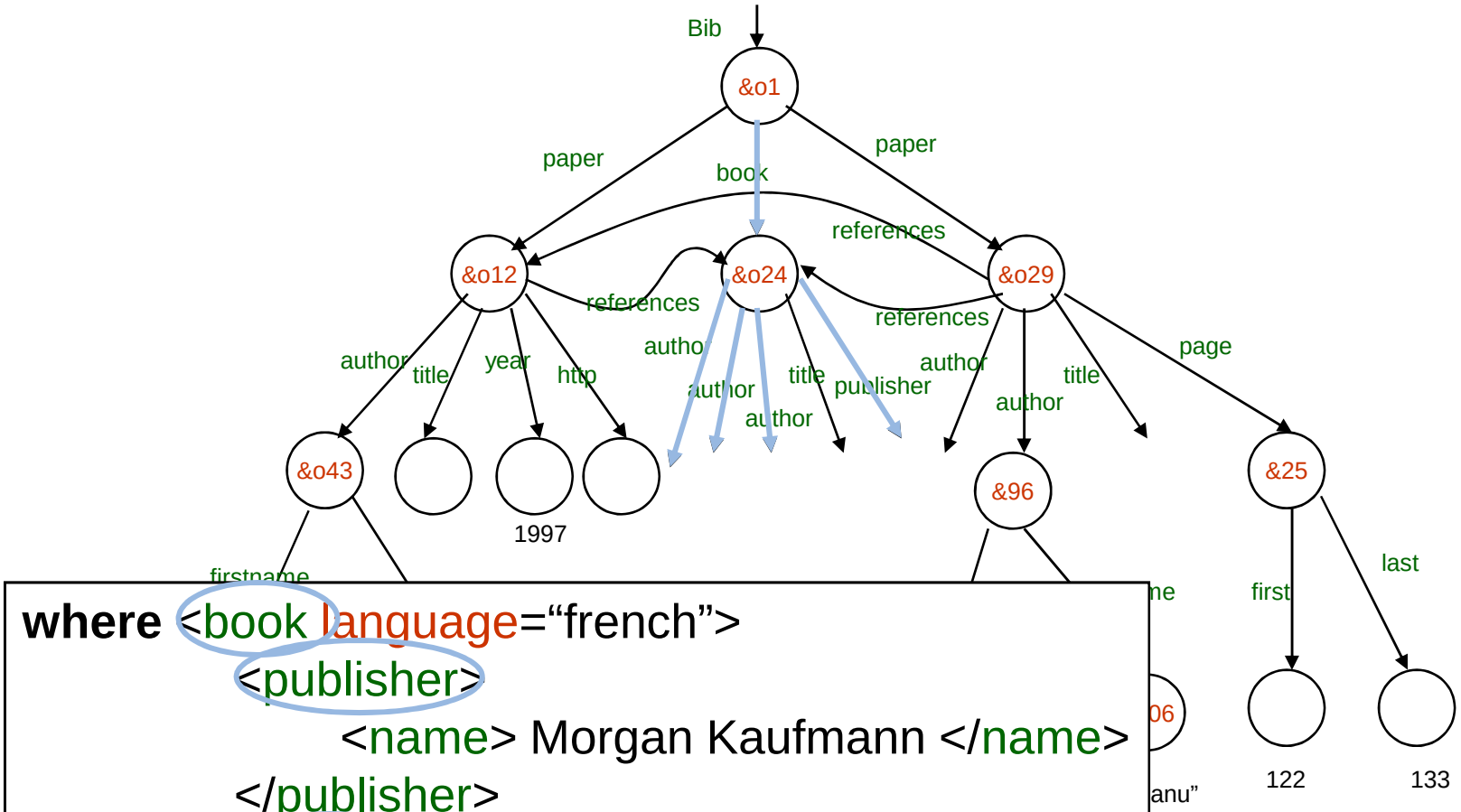
Name des zu analysierenden Quelldokuments. Es können Bedingungen an die Variablen im Muster folgen (Selektion).

Struktur des Ergebnisdokuments mit Angabe, in welcher Form die an die Variablen gebundenen Werte ausgegeben werden sollen.

XML-QL (2)

```
where <book language="french">
            <publisher>
                  <name> Morgan Kaufmann </name>
            </publisher>
            <author> $a </author>
      </book>
in "www.a.b.c/bib.xml"
construct $a
```

XML-QL (3)



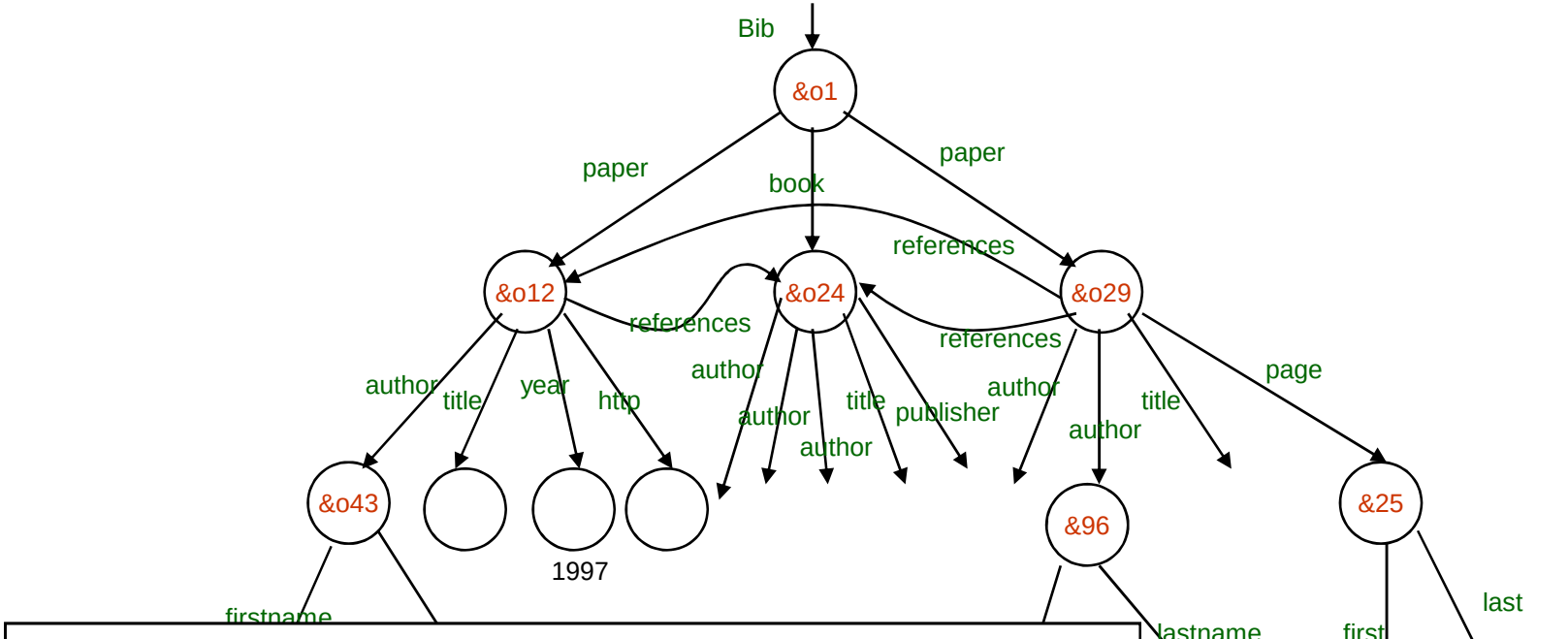

```
where <book language=“french”>
          <publisher>
                <name> Morgan Kaufmann </name>
          </publisher>
          <author> $a </author>
      </book>
in “www.a.b.c/bib.xml”
construct $a
```

XML-QL (4)

```
where <book language = $l>
                <author> $a </>
        </>
in “www.a.b.c/bib.xml”
construct <result> <author> $a </> <lang> $l </> </>
        ...
```

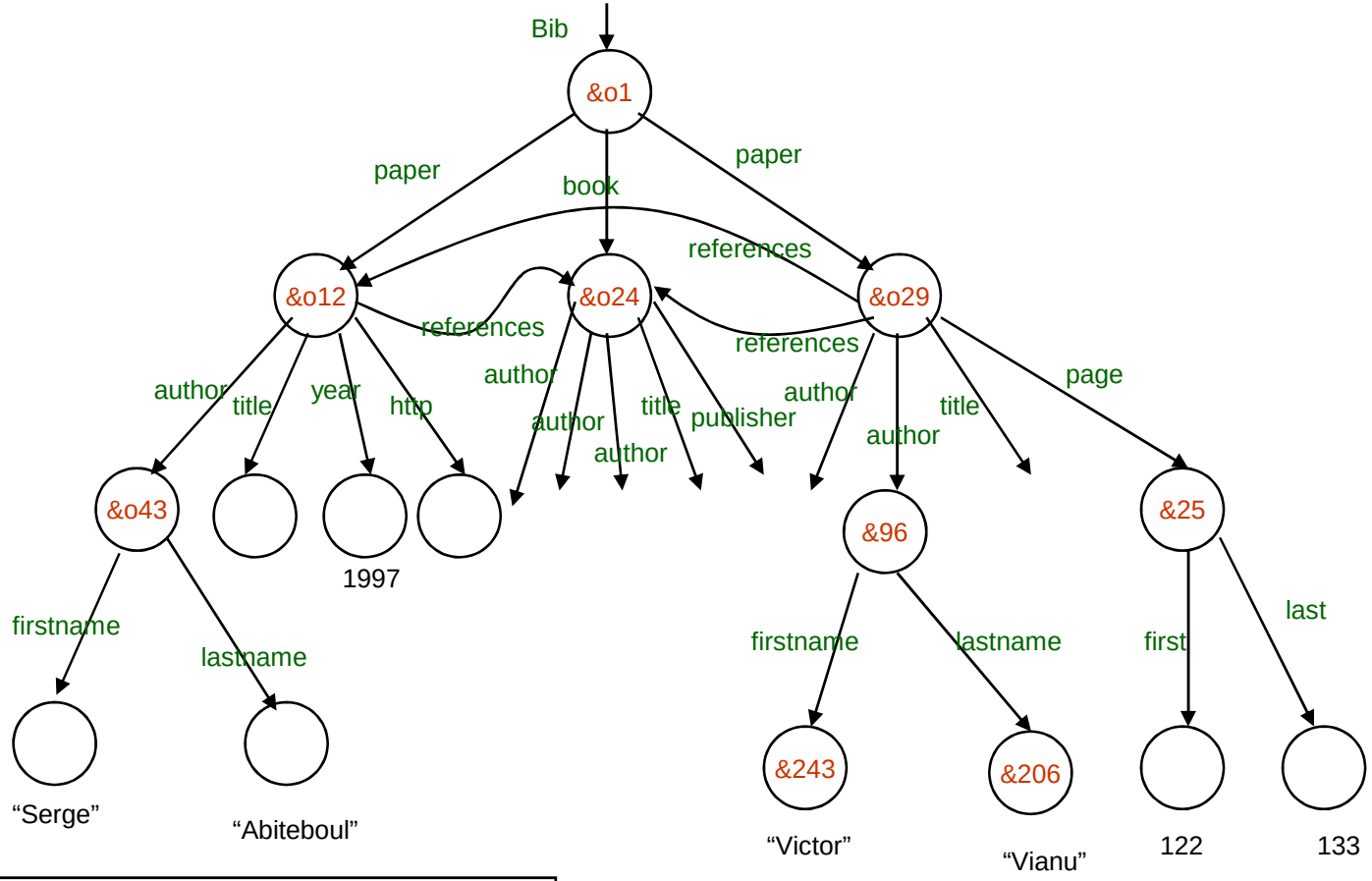
```
<result> <author>Smith</author><lang>English</lang></result>
<result> <author>Smith</author><lang>Mandarin</lang></result>
<result> <author>Doe</author><lang>English</lang></result>
```

XML-QL (5)




```
where <book language="french">
          <publisher>
              <name> $n </name>
          </publisher>
          <author> $a </author>
      </book>
in "www.a.b.c/bib.xml", $n = Morgan Kaufmann
construct $a
```

XML-QL (6)

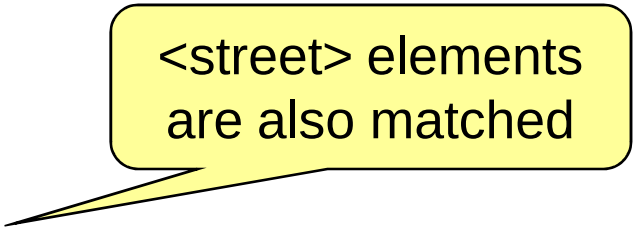


where <∗.author> $a </author>
in "www.a.b.c/bib.xml"
construct $a

XML-QL Beispiel 1

- Rückgabe einer Kurzadresse (ohne Strasse):

```
WHERE
  <address>
    <name>$n</>
    <town>$t</>
  </>
IN www.test.org/addresses.xml
CONSTRUCT
  <shortaddress>
    <name>$n</>
    <town>$t</>
  </>
```



XML-QL Beispiel 2

- XML-QL erlaub joining by value: