Datenbanksysteme

Einführung in XML-Technologien

Teil 1: XML und XPath

30.5.2011

Martin Gieseking

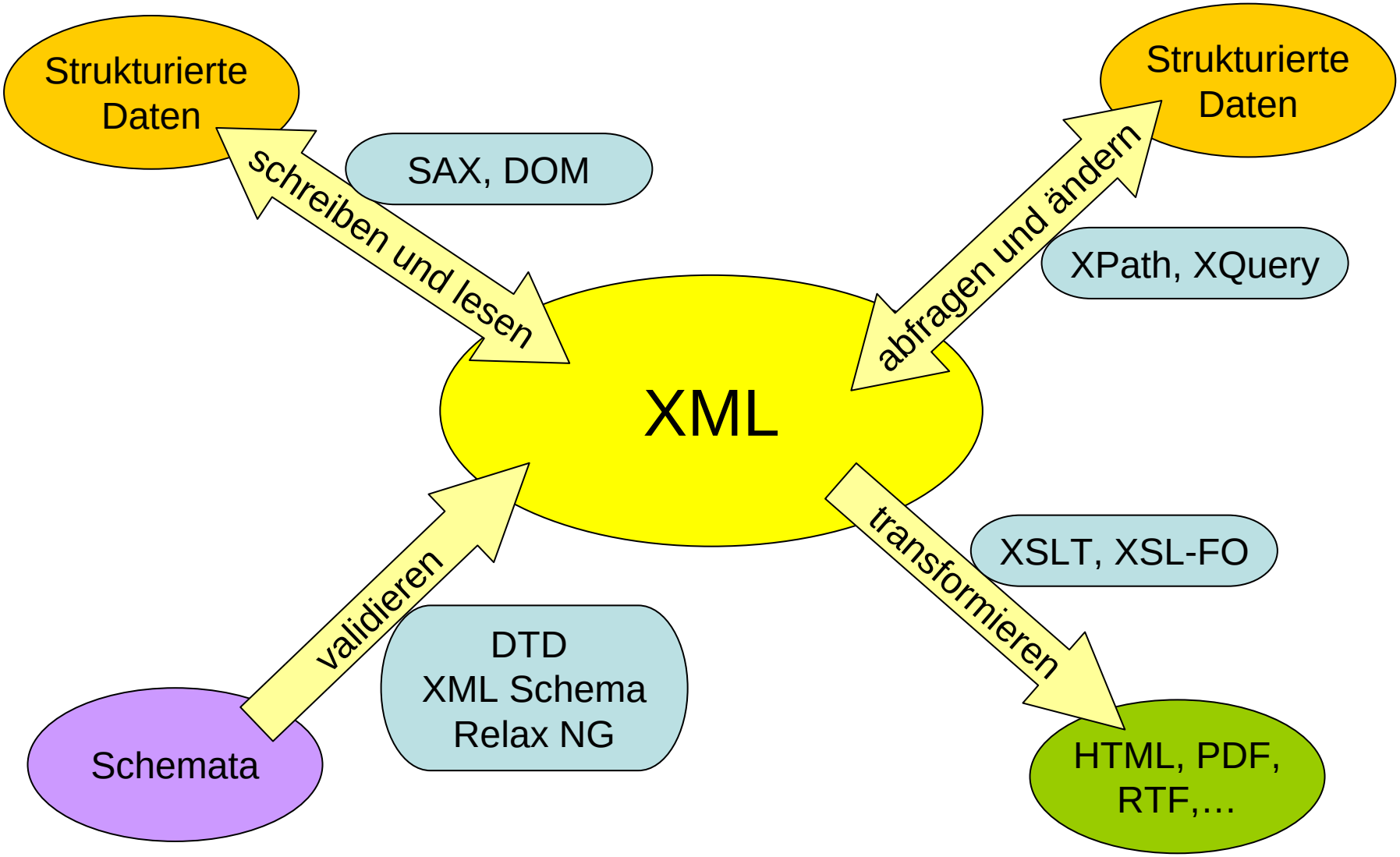
Was ist XML?

- XML (Extensible Markup Language) ist eine Meta-Auszeichnungssprache zur textbasierten Beschreibung hierarchisch strukturierter Daten
  - Spezifikation des World Wide Web Consortiums (W3C)
  - XML definiert kein konkretes Dateiformat sondern legt fest, welche syntaktischen Bestandteile verwendet und wie sie kombiniert werden dürfen
  - auf Grundlage der Regeln können konkrete Dateiformate mit definierter Struktur und Semantik abgeleitet werden
- ein konkretes Dateiformat, das auf Grundlage der XML-Spezifikation festgelegt wurde, wird allgemein **XML-Format** genannt
  - entsprechend nennt man Dateien in diesem Format **XML-Dateien**
  - Beispiele für XML-Formate sind *RSS*, *MathML*, *SVG*, *XHTML*

Beispiel einer XML-Datei

```
<?xml version="1.0"?>
<!-- Vorlesungsverzeichnis Sommersemester 2011 -->
<vorlesungsverzeichnis>
    <abschnitt titel="Mathematik &amp; Informatik">
        <abschnitt titel="Grundstudium">
            <veranstaltung nr="1.234" typ="v">
                <dozent>
                    <titel>Prof. Dr.</titel>
                    <vname>Klaus</vname>
                    <nname>Meier</nname>
                </dozent>
                <titel>Einführung in die monotone Algebra I</titel>
                <zeit>Mo 10:00-12:00</zeit>
                <raum>12/13</raum>
            </veranstaltung>
            <veranstaltung nr="1.247" typ="ü">
                <dozent>
                    <vname>Sandra</vname>
                    <nname>Schmidt</nname>
                </dozent>
                <titel>Übung zur Einführung in die monotone Algebra I</titel>
                <zeit>Mi 8:00-12:00</zeit>
                <raum>97/E9</raum>
            </veranstaltung>
        </abschnitt>
    </abschnitt>
</vorlesungsverzeichnis>
```

XML-Technologien



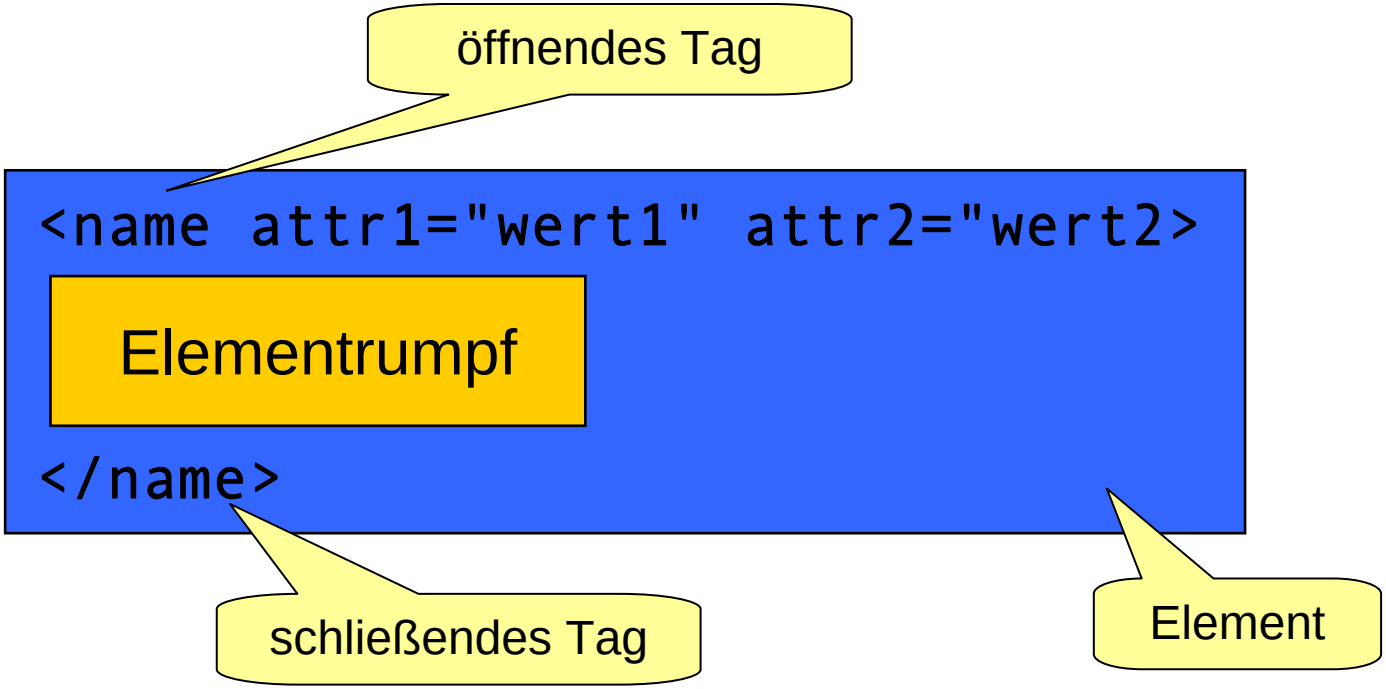
Bausteine einer XML-Datei

- XML-Dateien können aus folgenden Bausteinen zusammengesetzt werden:
  - Elemente
  - Attribute
  - Text und Entities
  - Kommentare
  - Verarbeitungsanweisungen (processing instructions)
  - CDATA-Abschnitte
  - DOCTYPE-Angabe

```
<?xml version="1.0"?>
<!-- Vorlesungsverzeichnis Sommersemester 2011 -->
<vorlesungsverzeichnis>
    <abschnitt titel="Mathematik &amp; Informatik">
        <abschnitt titel="Grundstudium">
            <veranstaltung nr="1.234" typ="v">
                <dozent>
                    <titel>Prof. Dr.</titel>
                    <vname>Klaus</vname>
                    <nname>Meier</nname>
                </dozent>
                <titel>Einführung in die ...</titel>
                <zeit>Mo 10:00-12:00</zeit>
                <raum>12/13</raum>
            </veranstaltung>
        </abschnitt>
    </abschnitt>
</vorlesungsverzeichnis>
```

Elemente

- die strukturierenden Bausteine eines XML-Dokuments werden **Elemente** genannt
- sie bestehen aus einem öffnenden und einem schließenden **Tag** sowie dem **Elementinhalt** (oder **Elementrumpf**)
  - anders als z.B. bei HTML muss jedes Element explizit ein öffnendes und schließendes Tag (Start- und End-Tag) besitzen



Elemente

- **öffnende Tags** haben immer die Form
  `<name attr1="wert1" attr2="wert2" ...>`
  - der Name kann aus Buchstaben, Ziffern und den Zeichen `_.-·` bestehen
    - erstes Zeichen muss ein Buchstabe sein
    - es wird zwischen Groß- und Kleinschreibung unterschieden
    - darf nicht mit *xml, Xml, xMl, xmL, XMl, XmL, xML* oder *XML* beginnen
  - zwischen „<“ und *name* darf sich kein Leerzeichen befinden
  - Attribute werden in der angegebenen Form durch Whitespaces getrennt hinter dem Elementnamen aufgelistet
  - für Attributnamen gelten die gleichen Vorgaben wie für Elementnamen
  - Attributwerte werden wahlweise in einfache oder doppelte Anführungszeichen eingeschlossen
  - die Reihenfolge der Attribute ist nicht signifikant
  - jeder Attributname darf in der Attributliste nur einmal auftauchen
- **schließende Tags** haben immer die Form
  `</name>`
  - der Name muss mit dem des zugehörigen öffnenden Tags übereinstimmen
  - schließende Tags enthalten keine weiteren Informationen

Elemente

- der Elementrumpf ist entweder leer oder besteht aus einer Folge von weiteren Elementen, Text, Kommentaren und/oder Verarbeitungsanweisungen
  - Elemente können beliebig tief geschachtelt werden

```
<veranstaltung nr="1.234" typ="v">
  <dozent geschlecht="m">
    <!- ein Kommentar -->
    <titel>Prof. Dr.</titel>
    <vname>Klaus</vname>
    <nname>Meier</nname>
  </dozent>
  Dies ist ein Text.
  <titel>Einführung in die ...</titel>
  <zeit>Mo 10:00-12:00</zeit>
  <raum>12/13</raum>
</veranstaltung>
```

- für Elemente mit leerem Rumpf gibt es die Kurzschreibweise `<name attr1="val1" attr2="val2" ... />`

```
<uhrzeit zeitzone="GMT"></uhrzeit>
```

ist identisch mit

```
<uhrzeit zeitzone="GMT"/>
```

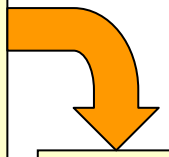
Elemente

- jedes XML-Dokument muss genau ein äußeres Element, das so genannte **Wurzelelement** besitzen
  - es stellt quasi den Rahmen oder den Container für die eigentlichen Daten dar
  - außerhalb des Wurzelelements sind nur Kommentare, Verarbeitungsanweisungen, eine DOCTYPE-Anweisung und Whitespace erlaubt

Fehler: kein (eindeutiges) Wurzelelement

```
<?xml version="1.0"?>
<veranstaltung nr="1.234">
  ...
</veranstaltung>

<veranstaltung nr="4.321">
  ...
</veranstaltung>
```



```
<?xml version="1.0"?>
<veranstaltungen>
  <veranstaltung nr="1.234">
    ...
  </veranstaltung>
  <veranstaltung nr="4.321">
    ...
  </veranstaltung>
</veranstaltungen>
```

Kommentare und Entities

- neben Text und Elementen spezifiziert XML noch weitere Konstrukte
  - Kommentare
    `<!-- dies ist ein Kommentar -->`
    - die Zeichenfolge `--` ist innerhalb von Kommentaren nicht erlaubt
  - Entity-Referenzen
    `&name; &#dez; &#xhex;`
    - bezeichnen ein einzelnes Zeichen oder eine Zeichenfolge
    - die Zeichen < und & haben in der XML-Syntax eine besondere Bedeutung (Metazeichen), und dürfen nicht als normaler Textbestandteil verwendet werden
    - zahlreiche Zeichen sind nicht im normalen Zeichenvorrat eines Zeichensatzes enthalten, so dass sie nicht direkt eingegeben werden können (z.B. mathematische oder musikalische Zeichen, Ligaturen usw.)

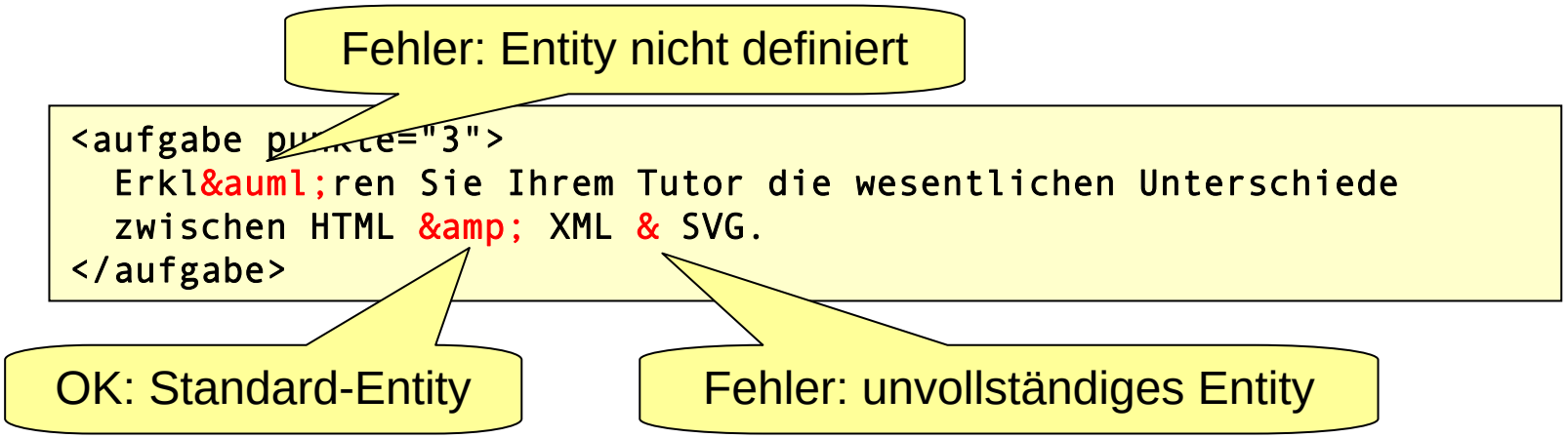
<	<
>	>
'	'
"	&quote;
&	&

```
<aufgabe punkte="1">
  <!-- eine einfache Frage zum Einstieg -->
  Wofür steht in XML die Zeichenfolge &lt;!-- ... -->?
</aufgabe>
```

Entity-Referenzen

- im Gegensatz zu HTML definiert XML nur fünf benannte Entities
- alle anderen aus HTML bekannten Entities, wie z.B. `&auml;` `&szlig;` ` ` usw. sind in XML nicht definiert und führen bei Verwendung zu einem Fehler
- es gibt die Möglichkeit, neue benannte Entities zu definieren

<	<
>	>
'	'
"	&quote;
&	&



- numerische Entity-Referenzen bezeichnen jeweils ein einzelnes Unicode-Zeichen
  - Eurozeichen: `&#8364;` (dezimal), `&#x20AC;` (hexadezimal)

XML-Namensräume

- Namensräume ermöglichen es, zusammengehörige Elemente zu bündeln und von „fremden“ Elementen abzugrenzen
- Namensräume werden in XML durch eindeutige URIs (Uniform Resource Identifiers) benannt
  - Zeichenkette, die einen Namensraum eindeutig identifiziert
  - haben meist die Form einer URL (muss nicht existieren)
- In XML-Dokumenten werden die Namensräume mit frei definierbaren Präfixen verknüpft, die den Elementen durch einen Doppelpunkt getrennt vorangestellt werden

```
<buch xmlns:pers="http://xyz.de/person">
 <titel>Das Leben der Affen</titel>
 <autor>
  <pers:person>
   <pers:titel>Prof. Dr.</pers:titel>
   <pers:vorname>Jim</pers:vorname>
   <pers:nachname>Panse</pers:nachname>
  </pers:person>
 </autor>
</buch>
```

```
<buch>
 <titel>Das Leben der Affen</titel>
 <autor>
  <person xmlns="http://xyz.de/person">
   <titel>Prof. Dr.</titel>
   <vorname>Jim</vorname>
   <nachname>Panse</nachname>
  </person>
 </autor>
</buch>
```

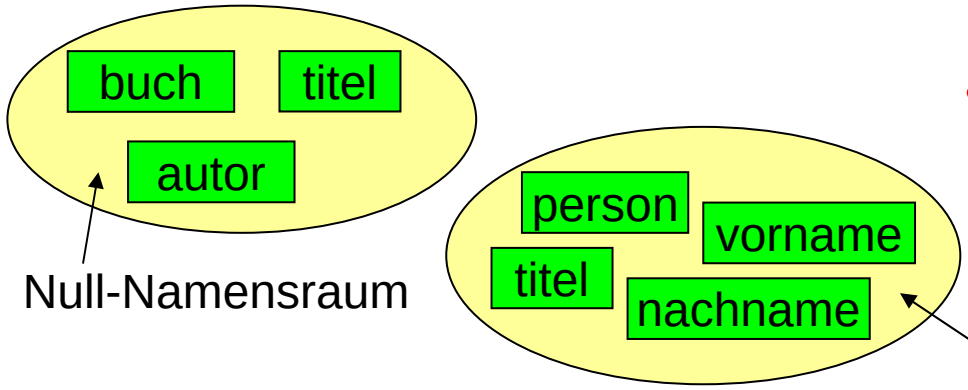
XML-Namensräume

```
<buch xmlns:pers="http://xyz.de/person">
 <titel>Das Leben der Affen</titel>
 <autor>
  <pers:person>
   <pers:titel>Prof. Dr.</pers:titel>
   <pers:vorname>Jim</pers:vorname>
   <pers:nachname>Panse</pers:nachname>
  </pers:person>
 </autor>
</buch>
```

- definiert den Präfix *pers* für den Namensraum *http://xyz.de/person*
- ist nur innerhalb des Elements bekannt, in dem er definiert wird

```
<buch>
 <titel>Das Leben der Affen</titel>
 <autor>
  <person xmlns="http://xyz.de/person">
   <titel>Prof. Dr.</titel>
   <vorname>Jim</vorname>
   <nachname>Panse</nachname>
  </person>
 </autor>
</buch>
```

- setzt den Standard-Namensraum auf *http://xyz.de/person*
- das Element und die Kindelemente werden dem Namensraum zugeordnet
- ein expliziter Präfix vor den Elementnamen ist hier nicht erforderlich



Wohlgeformte XML-Dokumente

- ein XML-Dokument heißt **wohlgeformt**, wenn es die Syntax- und Strukturvorgaben der XML-Spezifikation einhält
  - über 100 Regeln, die größtenteils intuitiv aus den bisher beschriebenen Aspekten hervorgehen
- die Wohlgeformtheit kann ohne Kenntnis der in einem Dokument zulässigen Elemente überprüft werden

nicht wohlgeformt

```
<?xml version="1.0"?>
<blatt nr="1">
  <aufgabe punkte="4">
    <list>
      <li>Punkt 1</li>
      <li>Punkt 2
  </aufgabe>
      </li>
    </list>
</Blatt>

<blatt nr="2">
  <aufgabe Punkte="6"/>
</blatt>
```

wohlgeformt

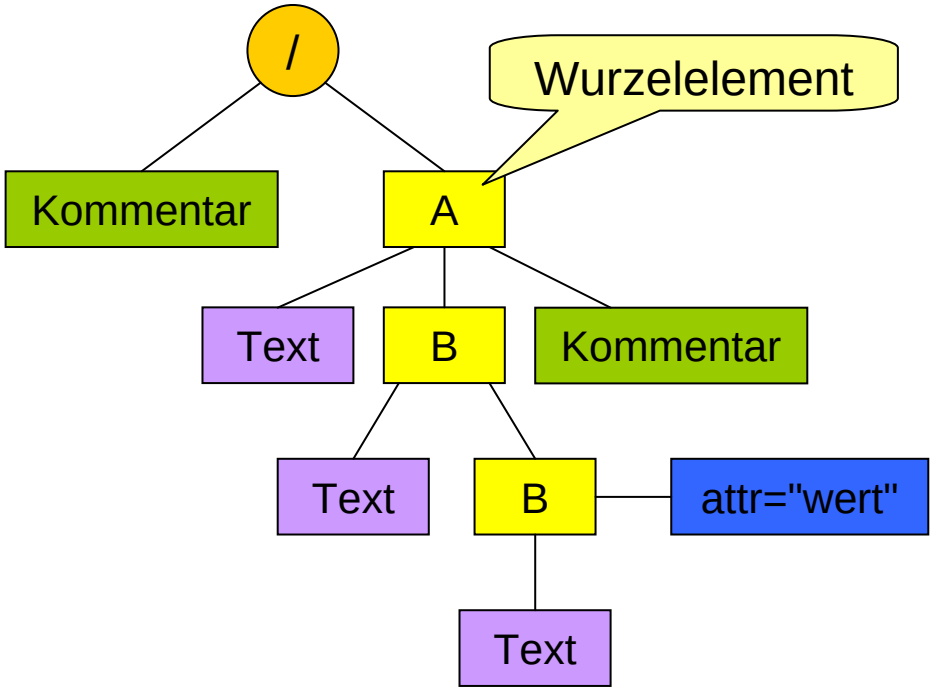
```
<?xml version="1.0"?>
<blattsammlung>
  <blatt nr="1">
    <aufgabe punkte="4">
      <list>
        <li>Punkt 1</li>
        <li>Punkt 2</li>
      </list>
    </aufgabe>
  </blatt>

  <blatt nr="2">
    <aufgabe PUNKTE="6"/>
  </blatt>
</blattsammlung>
```

Struktur von XML-Dokumenten

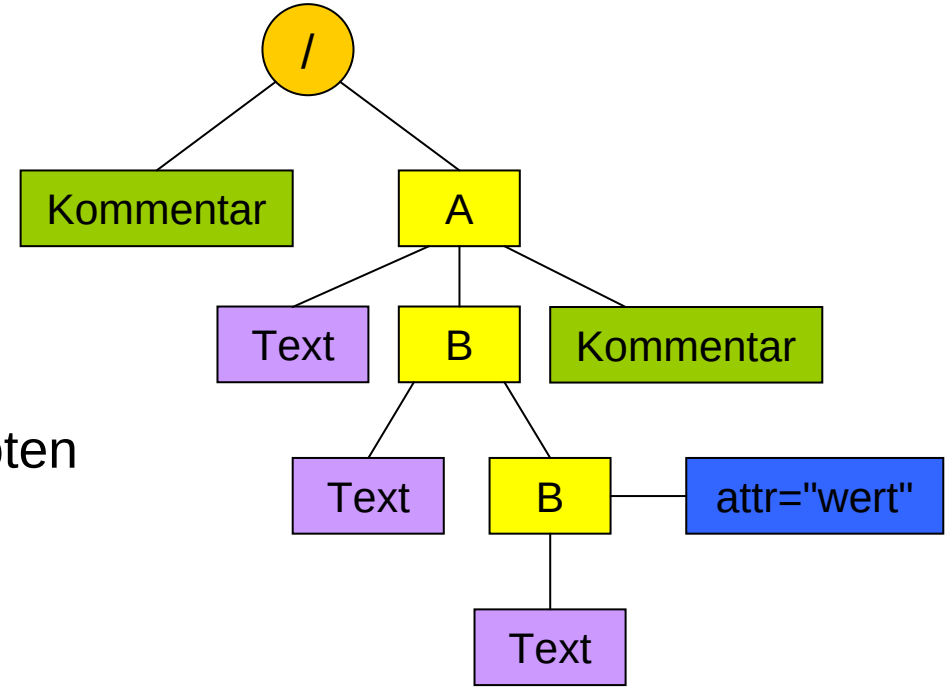
- XML-Dokumente besitzen grundsätzlich eine Baumstruktur
  - es gibt genau einen Wurzelknoten vom Typ *Document*
  - bis auf den **Dokumentknoten** haben alle Knoten einen eindeutigen Elternknoten
  - Attribute sind sog. **assoziierte Knoten**, die zwar ein Elternelement besitzen, selbst aber keine Kinder sind
- der Dokumentknoten muss genau einen Element-Kindknoten, das **Wurzelelement**, besitzen

```
<?xml version="1.0"?>
<!- erster Kommentar -->
<A>
  erster Textteil
  <B>
    zweiter Textteil
    <B attr="wert">
      dritter Textteil
    </B>
  </B>
  <!- zweiter Kommentar -->
</A>
```



XML-Knotentypen

- entsprechend der syntaktischen Elemente einer XML-Datei wird zwischen verschiedenen Knotentypen unterschieden
  - Dokumentknoten
  - Elementknoten
  - Textknoten
  - Kommentarknoten
  - Attributknoten
  - Namensraumknoten
  - Verarbeitungsanweisungsknoten



XPath: Navigieren in XML-Bäumen

- eine zentrale Operation auf XML-Bäumen ist das Auswählen von einzelnen Knoten oder Knotenmengen nach bestimmten Kriterien

- die Lokatorsprache **XPath** stellt einen Formalismus zur Adressierung von Knoten in XML-Bäumen bereit
  - zusätzlich werden mathematische und logische Operatoren sowie eine überschaubare Anzahl von Funktionen bereitgestellt (XPath 1.0)

- XPath ist integraler Bestandteil von weiteren XML-Technologien, wie XSLT und XQuery (Obermenge von XPath 2.0)

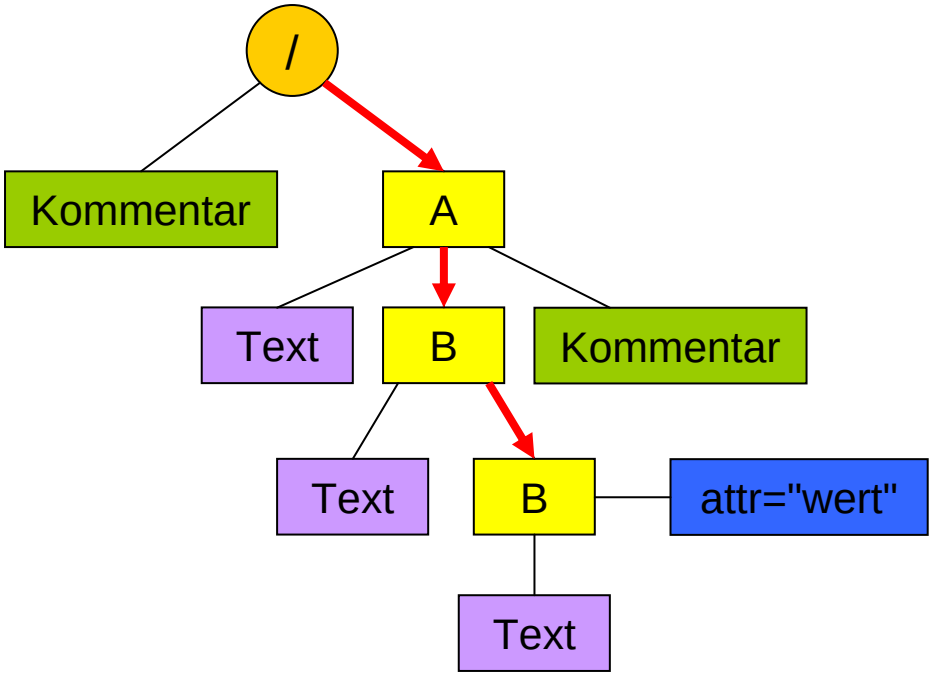
XPath-Software (Open Source)

- Firebug- und FirePath-Erweiterung für Firefox
  - Firefox unterstützt wie alle anderen Browser nur XPath 1.0
- Libxml2
  - http://xmlsoft.org
  - Kommandozeilen-Programm `xmllint` mit Option `--xpath`
- BaseX
  - http://www.inf.uni-konstanz.de/dbis/basex
  - XML-Datenbank mit grafischer Oberfläche und XQuery-Schnittstelle
  - XQuery ist eine Obermenge von XPath 2.0
- XQilla
  - http://xqilla.sourceforge.net/HomePage
  - XQuery-Prozessor

Beispiel: einfache Pfadangabe

- Knoten werden in XPath mit Hilfe von **Pfadangaben** ausgewählt
  - einfache XPath-Pfadangaben erinnern an Unix-Pfade zur Navigation im Dateisystem
  - ausgehend von der Dokumentwurzel können die Kindknoten schrittweise ausgewählt werden
- das Ergebnis einer Pfadangabe ist immer eine **Knotenmenge**
  - eine Liste mit Referenzen auf alle passenden Knoten ohne Dopplungen
- Beispiel: `/A/B/B`

```
<?xml version="1.0"?>
<!- erster Kommentar -->
<A>
  erster Textteil
  <B>
    zweiter Textteil
    <B attr="wert">
      dritter Textteil
    </B>
  </B>
  <!- zweiter Kommentar -->
</A>
```



Pfadangaben und Location-Steps

- jede Pfadangabe besteht aus einer Folge sogenannter **Location-Steps**
  - werden durch Slashes (`/`) voneinander getrennt
  - Beispiel: `step1/step2/step3`
  - jeder Location-Step wählt relativ zum vorangehenden Location-Step bzw. zum Kontextknoten eine Knotenmenge des XML-Baums aus

- jeder Location-Step besteht aus maximal drei Komponenten
  - einem optionalen Achsenbezeichner
  - einem Knotentest
  - einem oder mehreren optionalen Prädikat(en)
  - Syntax: `achenbezeichner::knotentest[prädikat]`
  - Beispiel: `ancestor::A[B]`

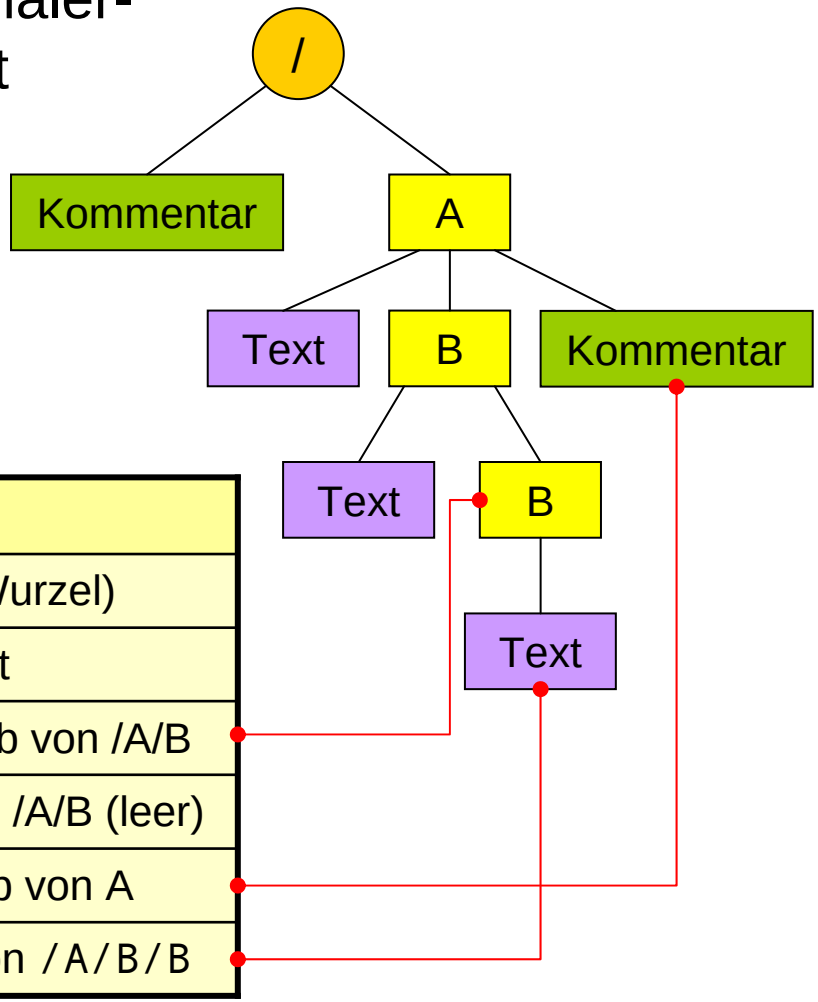
Knotentests

- im einfachsten Fall besteht ein Location-Step lediglich aus einem **Knotentest**
  - prüft, ob es an der aktuellen Position Knoten eines bestimmten Typs gibt
  - das Ergebnis eines Knotentests ist immer eine Menge aller passenden Knoten
  - falls es keine passenden Knoten gibt, ist die Ergebnismenge leer

Knotentyp	**Syntax des Knotentests**
Dokumentwurzel	`/`
Element *A*	`A`
Attribut *attr*	`@attr`
Text	`text()`
Kommentar	`comment()`
alle Knotentypen	`node()`

Knotentests

- Die Knotentests `text()` und `comment()` selektieren Text- bzw. Kommentarknoten (unabhängig von ihrem Inhalt)
- Elemente und Attribute werden normalerweise über ihren Namen ausgewählt
- der Knotentest `*` selektiert Elementknoten unabhängig vom Namen
- der Knotentest `@*` selektiert Attributknoten unabhängig vom Namen

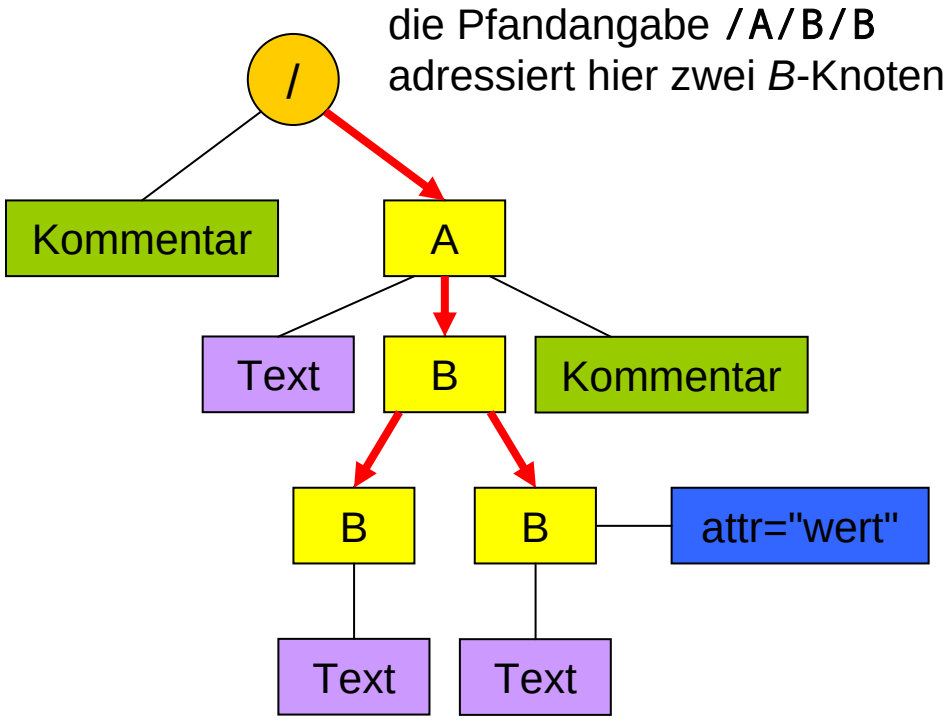


XPath-Ausdruck	Bedeutung
`/`	Dokumentknoten (Wurzel)
`/*`	Wurzelelement
`/A/B/*`	Elementknoten unterhalb von /A/B
`/A/B/A`	A-Element unterhalb von /A/B (leer)
`/A/comment()`	Kommentar unterhalb von A
`/A/B/B/text()`	Textknoten unterhalb von `/A/B/B`

Knotenmengen

- da Elementknoten mehrere gleichnamige und/oder gleichartige Knoten enthalten können, kann auch das Resultat eines Knotentests aus mehreren Knoten bestehen
  - das Resultat ist eine Knotenmenge
  - die Ergebnisknoten werden in Dokumentreihenfolge angeordnet (gilt nicht für alle Achsen)

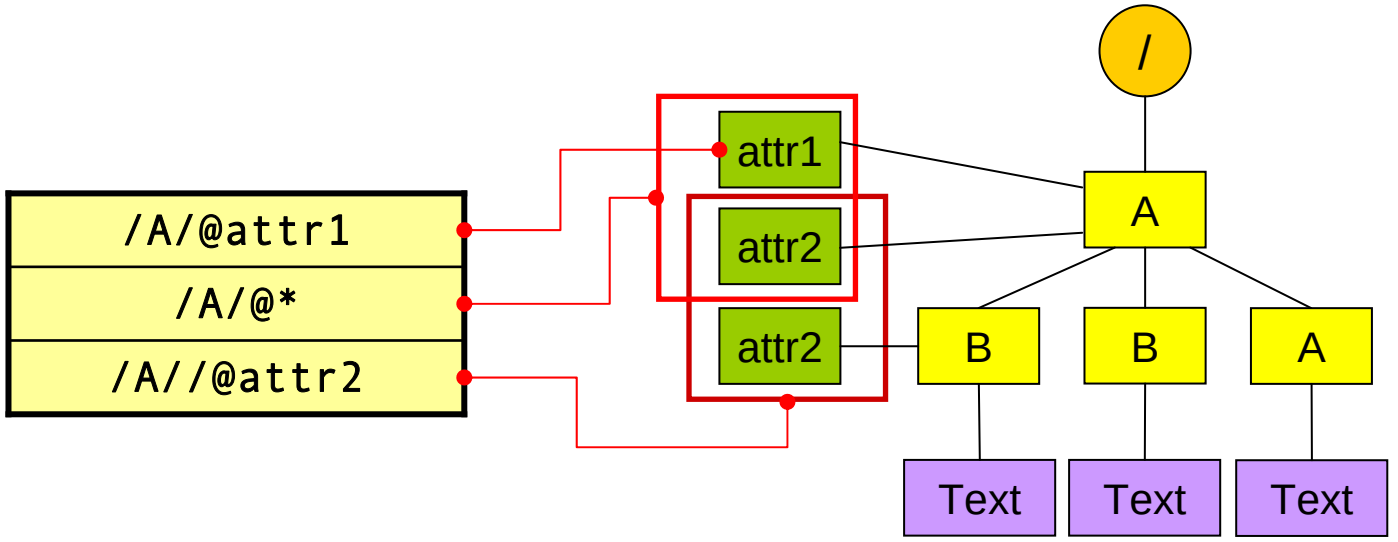
```
<?xml version="1.0"?>
<!- erster Kommentar -->
<A>
  erster Textteil
  <B>
    zweiter Textteil
    <B>
      dritter Textteil
    </B>
    <B attr="wert">
      vierter Textteil
    </B>
  </B>
  <!- zweiter Kommentar -->
</A>
```



Attributknoten

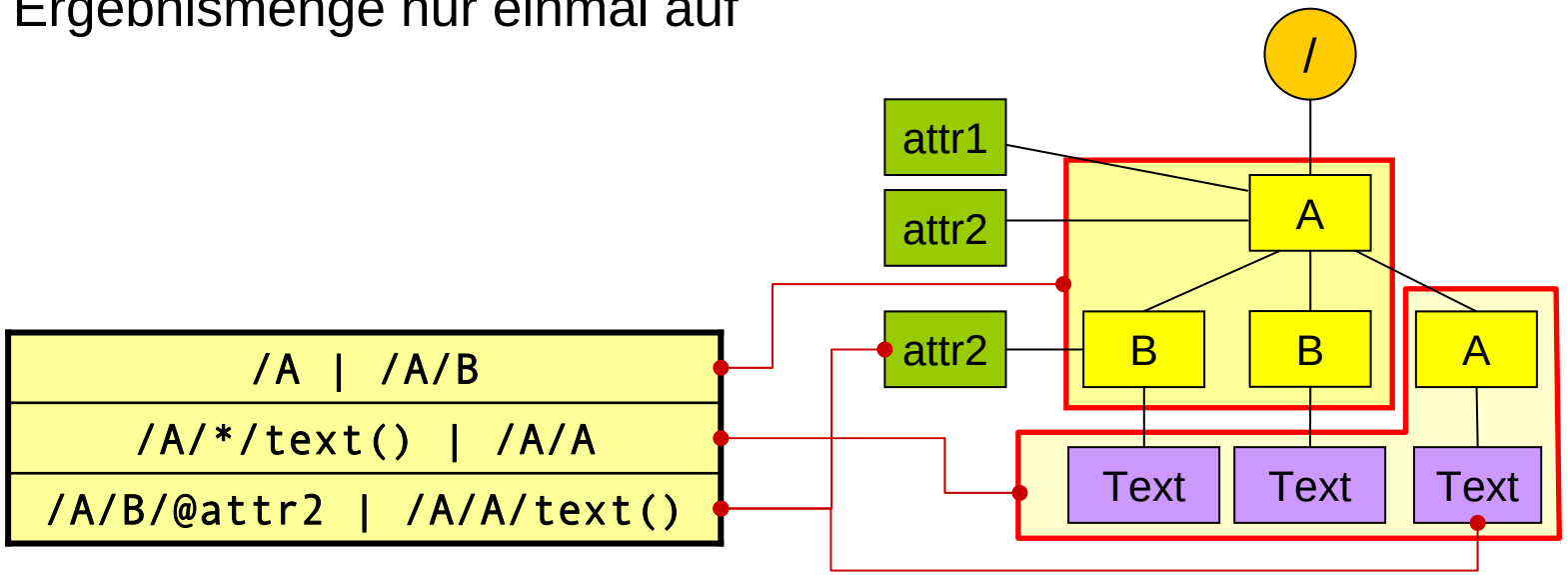
- die Attributknoten eines Elements werden durch einen Klammeraffen (@) gefolgt vom Attributnamen adressiert
  - mit @* erhält man alle Attributknoten der spezifizierten Elemente
- Attribute, die Namensraum-Präfixe definieren, können nicht mit @ adressiert werden
  - Zugriff muss mit Hilfe der *namespace*-Achse erfolgen

```
<?xml version="1.0"?>
<A attr1="val1" attr2="val2">
  <B attr2="val3">
    Text
  </B>
  <B>Text</B>
  <A>Text</A>
</A>
```



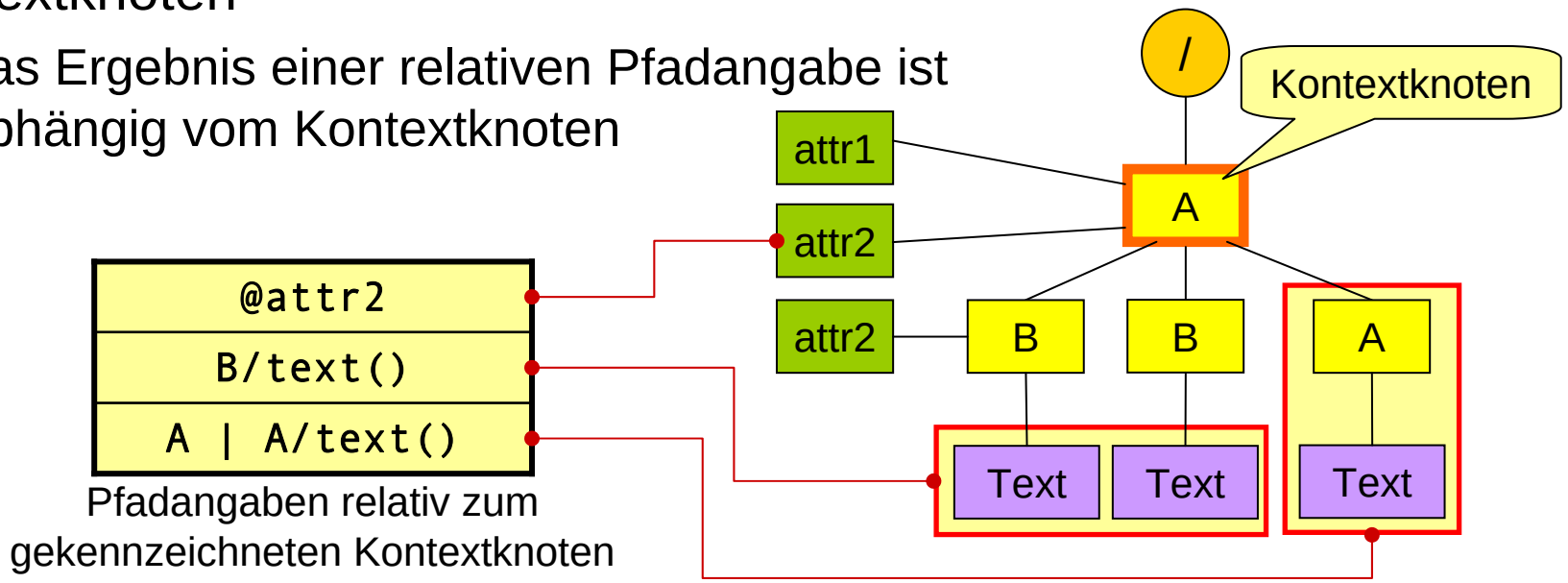
Vereinigung von Knotenmengen

- mit Hilfe des Kompositionsoperators | (senkrechter Strich) lassen sich zwei Knotenmengen vereinigen
  - auf beiden Seiten des Operators steht eine Pfadangabe
  - der Kompositionsoperator bindet schwächer als alle anderen Pfadoperatoren
  - die Knoten der Ergebnismenge sind immer in Dokumentreihenfolge angeordnet
  - Knoten, die von beiden Knotentests ausgewählt werden, tauchen in der Ergebnismenge nur einmal auf



Absolute und relative Pfadangaben

- XPath unterscheidet zwischen absoluten und relativen Pfadangaben
- **absolute Pfadangaben** beginnen immer bei der Dokumentwurzel
  - da die Dokumentwurzel mit einem Slash adressiert wird, beginnen absolute Pfadangaben immer mit einem Slash
  - das Ergebnis einer absoluten Pfadangabe ist für dasselbe XML-Dokument immer gleich
- **relative Pfadangaben** beziehen sich auf zuvor ausgewählte Kontextknoten
  - das Ergebnis einer relativen Pfadangabe ist abhängig vom Kontextknoten



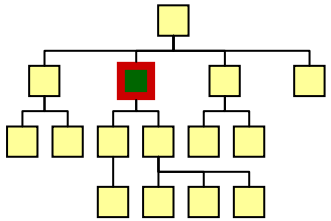
Pfadangaben relativ zum gekennzeichneten Kontextknoten

Achsen: Suchbereich von Knotentests angeben

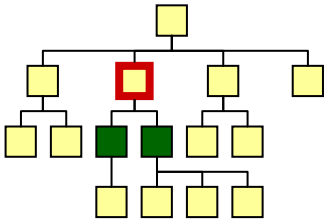
- wenn nicht anders angegeben, beziehen sich alle Kotentests immer auf die Kinder der aktuell adressierten Knoten
  - `/A/B/C` bedeutet:
    - suche alle *A*-Kindelemente des Dokumentknotens
    - suche in den gefundenen *A*-Elementen nach *B*-Kindelementen
    - suche in den gefundenen *B*-Elementen nach *C*-Kindelementen und liefere sie als Ergebnis zurück
  - jeder Location-Step bewirkt hier also einen Abstieg um eine Ebene im Baum
- mit Hilfe einer **Achsenangabe** kann der Bereich, auf den ein Knotentest angewendet werden soll, geändert werden, z.B. um
  - in Eltern- oder Geschwisterknoten zu suchen
  - rekursiv in allen Vorgänger- oder allen Folgeknoten zu suchen
  - Attribut- oder Namensraumknoten auszuwählen
- XPath 1.0 definiert 13 verschiedene Achsen

Achsen

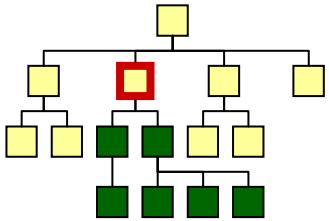
self



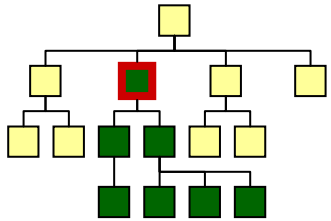
child



descendant

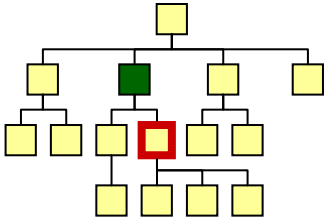


descendant-or-self

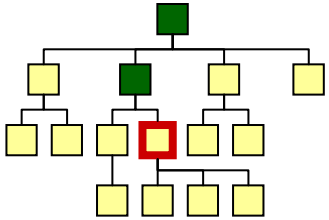


Kontextknoten
Knoten der Achse

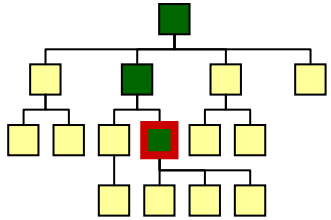
parent



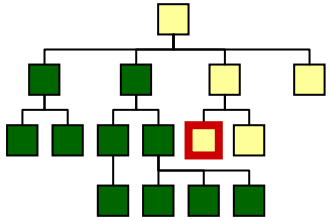
ancestor



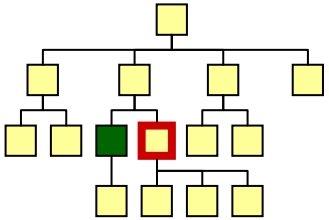
ancestor-or-self



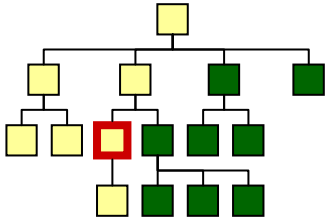
preceding



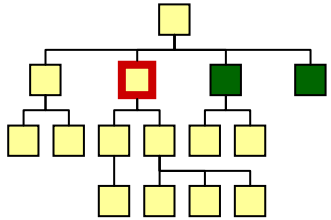
preceding-sibling



following



following-sibling



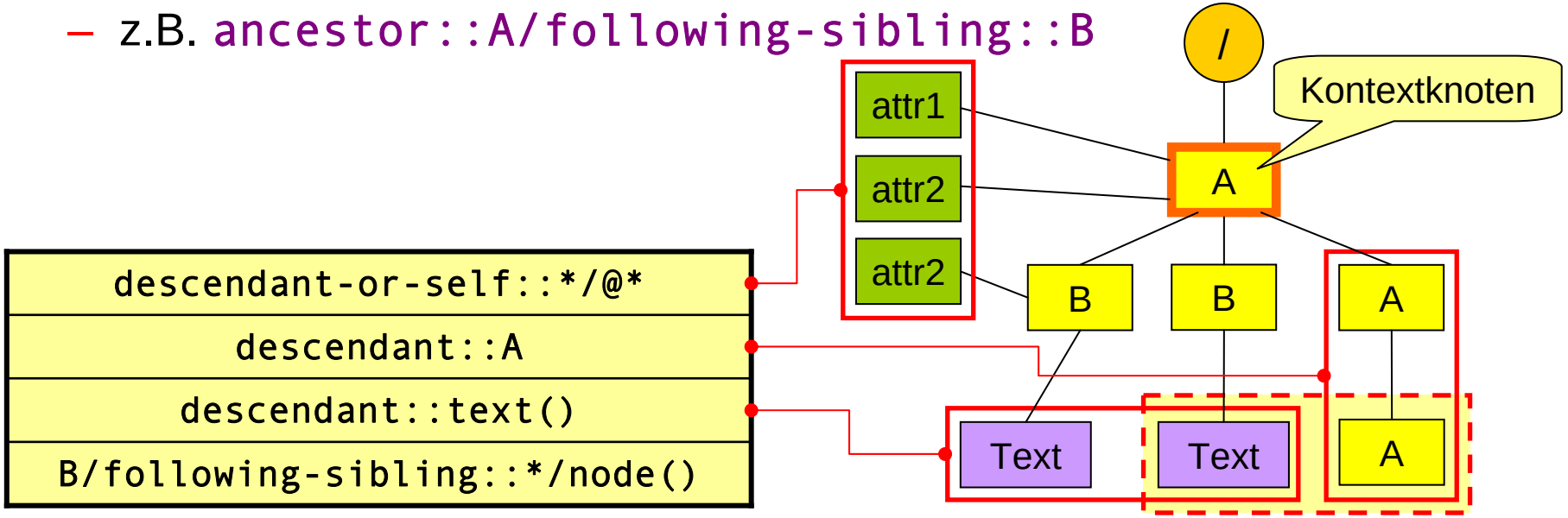
Achsen

Achse	Beschreibung
self	aktueller Knoten
child	alle unmittelbaren Kinder des aktuellen Knotens
descendant	alle Nachfahren des aktuellen Knotens
descendant-or-self	wie *descendant* aber zuzüglich des aktuellen Knotens
parent	Elternknoten des aktuellen Knotens
ancestor	alle Vorfahren des aktuellen Knotens
ancestor-or-self	wie *ancestor* aber zuzüglich des aktuellen Knotens
preceding	alle vorausgehenden Knoten ohne Vorfahren des aktuellen Knotens
preceding-sibling	alle vorausgehenden Geschwisterknoten
following	alle nachfolgenden Knoten ohne Nachfahren des aktuellen Knotens
following-sibling	alle nachfolgenden Geschwisterknoten
attribute	alle Attributknoten des aktuellen Elements

- die preceding-Achse umfasst alle Knoten, die in der XML-Datei vor dem aktuellen Knoten stehen und keine Vorfahren des aktuellen Knotens sind
- die following-Achse umfasst alle Knoten, die in der XML-Datei hinter dem aktuellen Knoten stehen und keine Nachfahren des aktuellen Knotens sind

Auswahl einer Achse

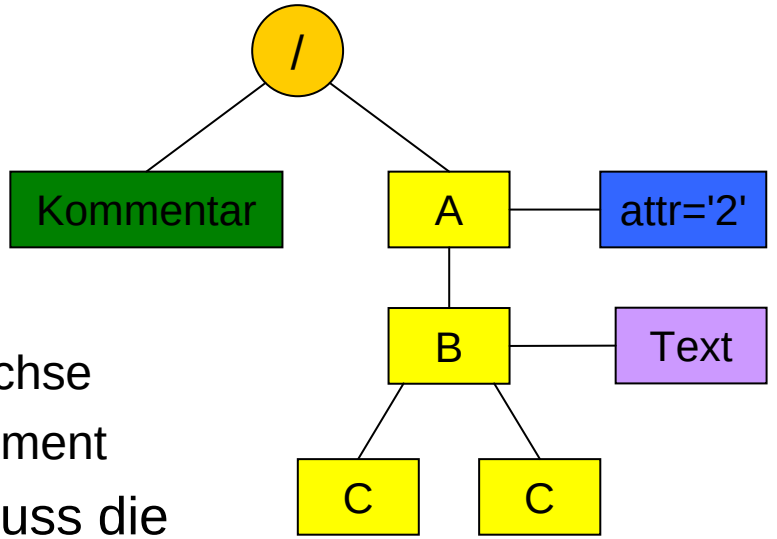
- Achsenangaben werden einem Knotentest zusammen mit zwei Doppelpunkten vorangestellt
  - z.B. `parent::A`
- Achsenangaben gelten nur für den aktuellen Location-Step
  - in der Pfadangabe `ancestor::A/B` bezieht sich die *ancestor*-Achse nur auf die Auswahl der *A*-Knoten; die *B*-Knoten werden in den Kindelementen der gefundenen *A*-Knoten gesucht
    (bei fehlender Achsenangabe wird die *child*-Achse verwendet)
- jeder Location-Step kann eine Achsenangabe enthalten
  - z.B. `ancestor::A/following-sibling::B`



attribute-Achse

- außer *self* berücksichtigen die gerade erwähnten Achsen nur Dokument-, Element-, Text- und Kommentar-Knoten (sowie Processing-Instructions)
- Attribut- und Namespace-Knoten sind nur über die *attribute*- bzw. *namespace*-Achse erreichbar
  - für nebenstehenden XML-Baum liefert `/A/node()` nur den B-Knoten
    - `node()` sucht hier auf der *child*-Achse von Element *A*
    - Attributknoten gehören nicht zur *child*-Achse

    → Knotentest `node()` findet nur das *B*-Element
  - zur Adressierung des Attributknotens muss die *attribute*-Achse gewählt werden, z.B.
    - `/A/attribute::node()`
    - `/A/attribute::attr`
    - `/A/attribute::*`



Kurzformen für häufig verwendete Location-Steps

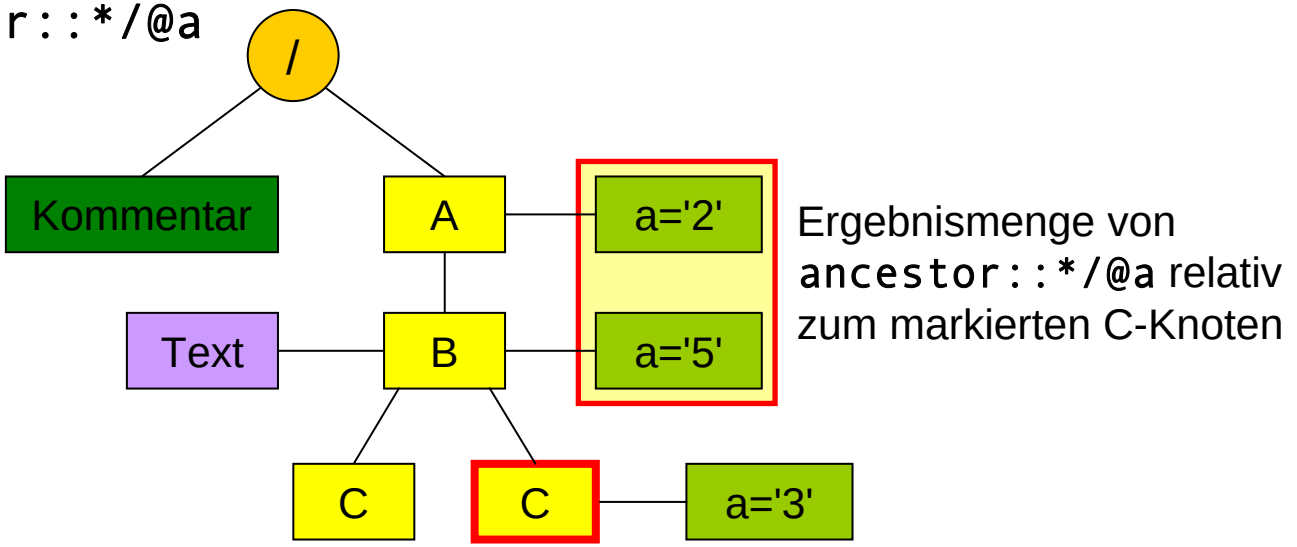
- in der Praxis werden einige Achsen zusammen mit bestimmten Knotentests besonders häufig benötigt
  - → für fünf ausgewählte Location-Steps stellt XPath Kurzschreibweisen zur Verfügung

Kurzform	ausführliche Form
`.`	`self::node()`
`..`	`parent::node()`
`//`	`/descendant-or-self::node()/`
name	`child::`*name*
`@`*name*	`attribute::`*name*

Beispiel:
`A//B/@attr` ist eine Kurzform von
`child::A/descendant-or-self::node()/child::B/attribute::attr`

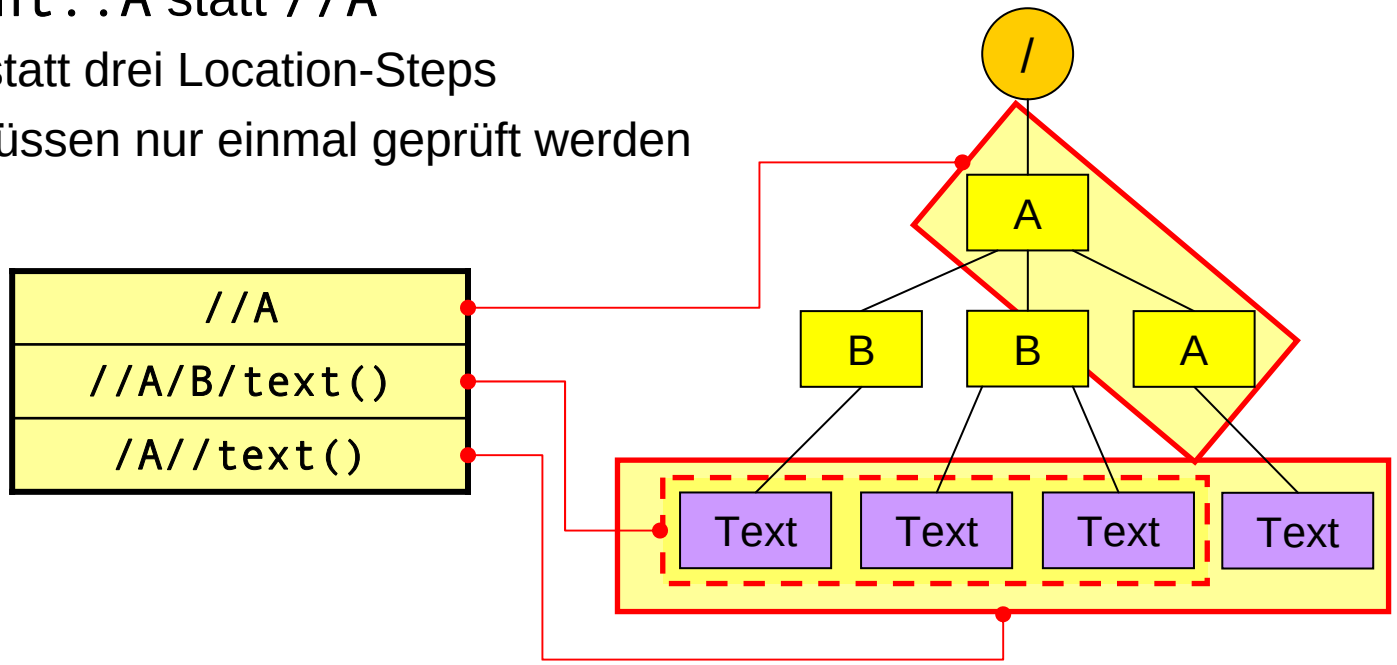
Anmerkungen zur *attribute*-Achse

- Wichtig: `@a` ist eine Kurzform für `attribute::a`
  - die Kurzform kann nicht direkt mit anderen Achsen kombiniert werden
  - etwas wie `ancestor::@a` ist syntaktisch falsch
    - die Langform wäre hier: `ancestor::attribute::a`
    - da jeder Location-Step nur eine Achsenangabe enthalten kann, ist der vorangehende Ausdruck nicht definiert
  - zur Auswahl von Attributen der Vorgängerelemente müssen zunächst die gewünschten Elemente ausgewählt werden; im zweiten Schritt sind dann die Attribute erreichbar
    - z.B. `ancestor::*/@a`



Rekursiver Abstieg mit `//`

- die Zeichenfolge `//` ist eine Kurzschreibweise für `/descendant-or-self::node()/`
  - kann anstelle einfacher Slashes verwendet werden, um anschließende Knotentests auf alle Nachfahren anzuwenden
  - sollte sorgsam verwendet werden, da die Auswertung abhängig von der Größe des Teilbaums zeitaufwendig sein kann
  - abhängig vom XPath-Prozessor kann ein direkter Knotentest auf der *descendant*-Achse etwas effizienter sein, z.B. `/descendant::A` statt `//A`
    - nur zwei statt drei Location-Steps
    - Knoten müssen nur einmal geprüft werden



Datentypen in XPath 1.0

- XPath 1.0 unterstützt vier verschiedene Datentypen
  - **boolean** (Wahrheitswerte)
    - die beiden booleschen Konstanten lauten in XPath `true()` und `false()`
  - **number** (Gleitkommazahlen)
  - **string**
    - String-Konstanten können in einfachen oder doppelten Anführungszeichen angegeben werden, z.B. `'Hallo'` oder `"Hallo"`
  - **node-set** (Knotenmengen)
    - Knotenmengen werden üblicherweise durch Pfadangaben erzeugt

XPath-Operatoren

a `=` *b*	a gleich *b*
a `!=` *b*	*a* ungleich *b*
a `<` *b*	*a* kleiner *b*
a `>` *b*	*a* größer *b*
a `<=` *b*	*a kleiner oder gleich b*
a `>=` *b*	*a* größer oder gleich *b*

a `+` *b*	Addition
a `-` *b*	Subtraktion
a `*` *b*	Multiplikation
a `div` *b*	Gleitkomma-Division
a `mod` *b*	Gleitkomma-Modulo
a `or` *b*	logisches ODER
a `and` *b*	logisches UND
a `|` *b*	Vereinigung von Knotenmengen

Ausgewählte XPath-Funktionen

Funktion	Returntyp	Beschreibung
`concat(`*s1*`,...,`*sn*`)`	string	verkettet die Strings *s1* bis *sn* zu einem neuen String
`contains(`*s1*`, `*s2*`)`	boolean	prüft, ob String *s1* den String *s2* enthält
`count(`*K*`)`	number	Anzahl der Knoten in Menge *K*
`name(`*K*`)`	string	Name des ersten Knotens in Menge *K*
`normalize-space(`*s*`)`	string	entfernt Whitespace am Anfang und Ende von *s* und ersetzt die restlichen Whitespace-Folgen durch einfache Leerzeichen
`not(`*b*`)`	boolean	boolsche Negation
`round(`*x*`)`	number	rundet *x*
`starts-with(`*s1*`, `*s2*`)`	boolean	prüft, ob String *s1* mit *s2* beginnt
`string-length(`*s*`)`	number	Länge von String *s*
`substring(`*s*`, `*n*`, `*l*`)`	string	Teilstring von *s*, beginnend beim *n*-ten Zeichen und Länge *l*
`substring-after(`*s1*`, `*s2*`)`	string	sucht *s2* in *s1* und liefert alle Zeichen hinter dem ersten Treffer zurück (liefert Leerstring zurück falls *s2* in *s1* nicht vorkommt)
`substring-before(`*s1*`, `*s2*`)`	string	sucht *s2* in *s1* und liefert alle Zeichen vor dem ersten Treffer zurück (liefert Leerstring zurück falls *s2* in *s1* nicht vorkommt)
`sum(`*K*`)`	number	Addiert die Werte der Knoten in Menge *K*
`translate(`*s1*`, `*s2*`, `*s3*`)`	string	sucht in *s1* der Reihe nach die Zeichen von *s2* und ersetzt in *s1* das *i*-te Zeichen von *s2* durch das *i*-te Zeichen von *s3*

XPath-Ausdrücke

- neben Pfadangaben sind auch Funktionsaufrufe sowie Ausdrücke mit mathematischen und logischen Operatoren gültige XPath-Ausdrücke

- Beispiele:
  - `1+2*3`
  - `concat('XML', 'und', 'XSLT')`
  - `contains('Blumentopferde', 'pferd')`
  - `count(//A)`
  - `translate('Hello', 'eo', 'ae')`
  - `(1+2*3 > (1+2)*3) or starts-with('Hallo', 'ha')`
  - `not(count(ancestor::A) > 5)`

Automatische Typumwandlung

- Falls ein Operand oder ein Funktionsparameter nicht den erwarteten Typ besitzt, führt XPath eine implizite Typkonvertierung durch, z.B.
  `"5.5" + true() * 2` → 7.5

von \ nach	**boolean**	**number**	**string**	**node-set**
boolean		false → 0 true → 1	false → `'false'` true → `'true'`	*nicht erlaubt*
number	0 → false sonst true		Dezimalzahl als String	*nicht erlaubt*
string	`''` → false sonst true	interpretiert den String als Zahl		*nicht erlaubt*
node-set	`{}` → false sonst true	interpretiert den aus erstem Knoten gebildeten String als Zahl	hängt die Texte descendant::text() des ersten Knotens aneinander	

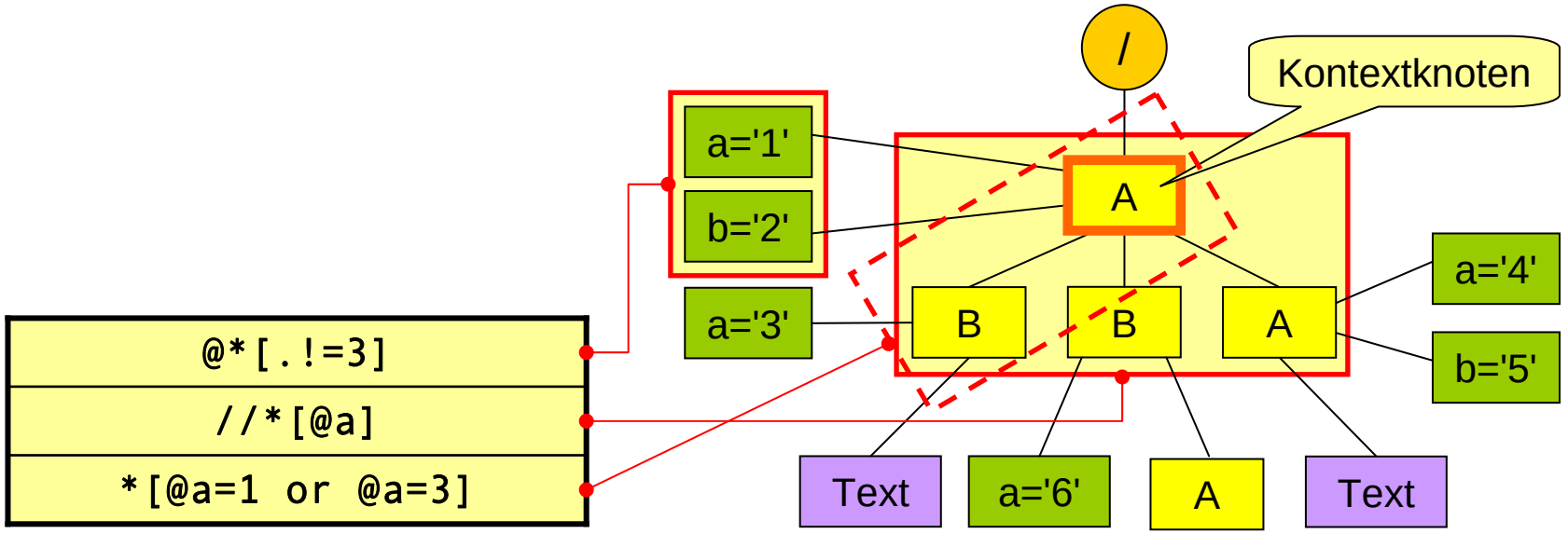
Prädikate

- **Prädikate** sind optionale Bestandteile eines Location-Steps und grenzen Knotenmengen durch Boolesche Ausdrücke weiter ein
  - nur Knoten, auf die der angegebene Ausdruck zutrifft, werden in die Ergebnismenge aufgenommen
- Prädikate werden in eckigen Klammern hinter einem Knotentest angegeben
  - `A[.='Hallo']`
    liefert alle *A*-Kindknoten, die den Text "Hallo" enthalten
  - `A[@attrib = 'Hallo']`
    liefert alle *A*-Kindknoten, die einen Attributknoten *attrib* mit Wert "Hallo" enthalten
  - `A[B]`
    liefert alle *A*-Kindknoten, die mindestens einen *B*-Kindknoten enthalten
- relative Pfadangaben in Prädikaten beziehen sich immer auf die durch den vorangehenden Pfadausdruck ausgewählte Knotenmenge

Prädikate: Beispiele

```
<?xml version="1.0"?>
<A a="5" b="8">
  <B a="20">Guten</B>
  <B>
    <A a="1">Morgen</A>
    <C>Heute</C>
  </B>
  <B a="50">Tag</B>
  <A>Abend</A>
</A>
```

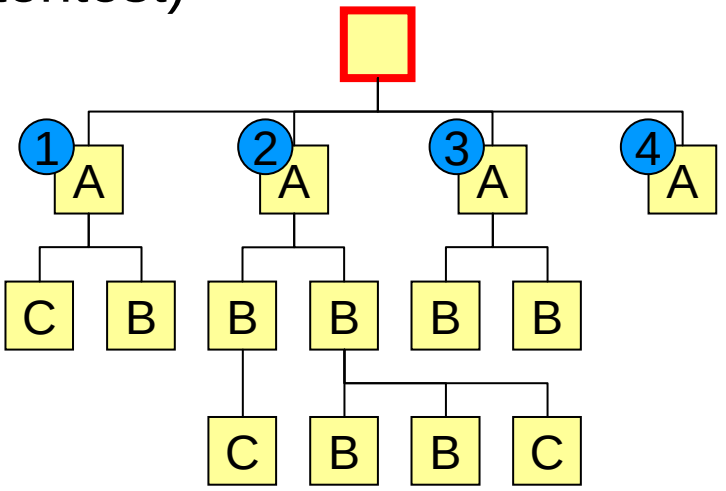
```
//A[not(@a)]
/A/B[@a < 50]
//B[.='Tag']
```




```
@*[.!=3]
//*[@a]
*[@a=1 or @a=3]
```

Knotennummerierung und Positionstests

- die von einem Location-Step ausgewählten Knoten werden implizit nummeriert (relativ zu Achse und Knotentest)
  - Nummerierung beginnt immer bei 1
- die XPath-Funktion `position()` liefert die Positionsnummer des aktuellen Kontextknotens
  - der Ausdruck `A[position()=1]` liefert alle *A*-Knoten mit Positionsnummer 1

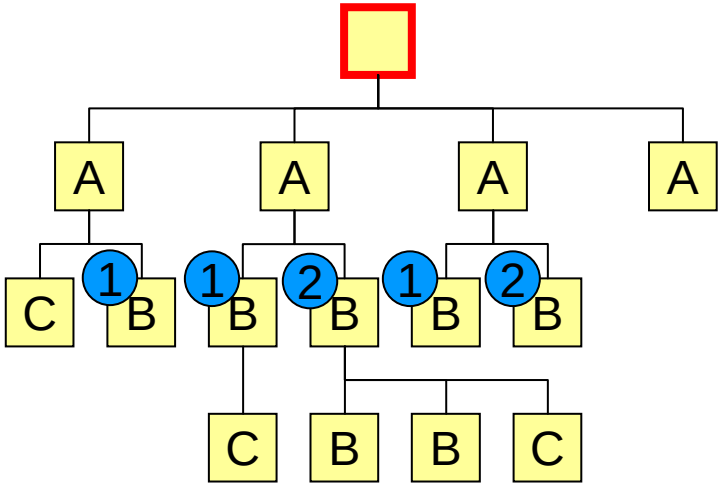


Nummerierung der Ergebnisknoten beim Location-Step `A` relativ zum rot markierten Kontextknoten

- Prädikate der Form `position()=`*n*, wobei *n* eine Zahl (Typ *number*) ist, werden **Positionstests** genannt
- Prädikate, die nur aus einer Zahl bestehen, werden immer als Positionstests interpretiert
  - `A[1]` ist eine Kurzform von `A[position()=1]`

Knotennummerierung und Positionstests

- die Knotennummerierung bezieht sich immer auf die gewählte Achse
  - im Fall der *child*-Achse werden die Kinder des Kontextknotens in Dokumentreihenfolge durchnummeriert
  - wenn ein Knotentest Kindknoten von unterschiedlichen Eltern zurückliefert, beginnt die Nummerierung der Kinder für jeden Elternknoten erneut bei 1
- für den abgebildeten Baum liefert der Knotentest `A/B[1]` eine Menge mit drei *B*-Knoten zurück
- zur gezielten Auswahl eines *B*-Knotens muss zuvor der gewünschte *A*-Knoten selektiert werden, z.B. liefert `A[3]/B[1]` den ersten *B*-Kindknoten des dritten *A*-Elements

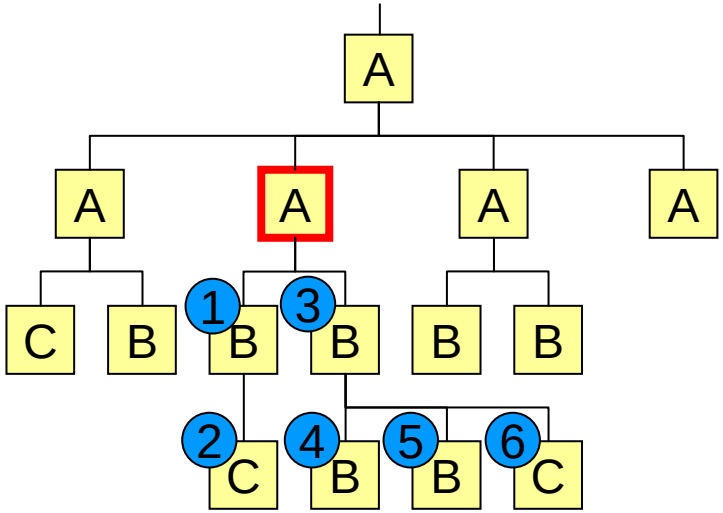


Knotennummerierung und Positionstests

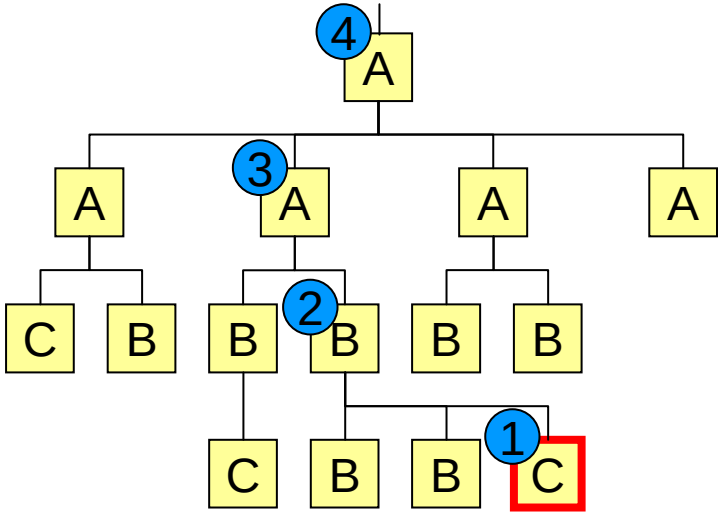
- die Knotennummerierung ist abhängig von
  - dem/den jeweiligen Kontextknoten
  - der gewählten Achse
  - dem Knotentest
- durch Änderung eines dieser Parameter ändert sich ggf. die Nummerierung

Knotennummerierung und Positionstests

- bei allen Achsen, die Dokumentbereiche hinter dem Kontextknoten beschreiben, wächst die Nummerierung in Richtung des Dokumentendes
- bei allen Achsen, die Dokumentbereiche vor dem Kontextknoten beschreiben, wächst die Nummerierung in Richtung des Dokumentanfangs



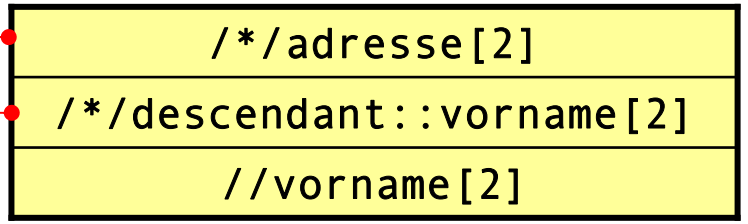
`descendant::*`



`ancestor-or-self::*`

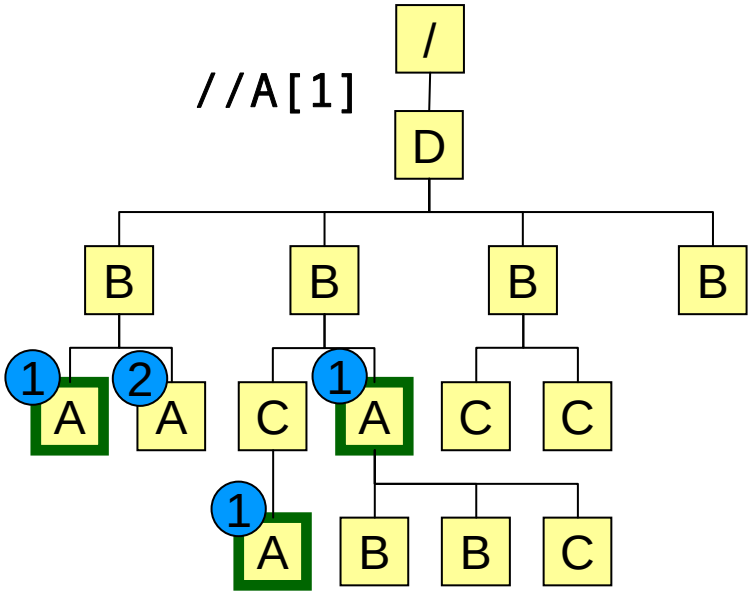
Knotennummerierung und Positionstests

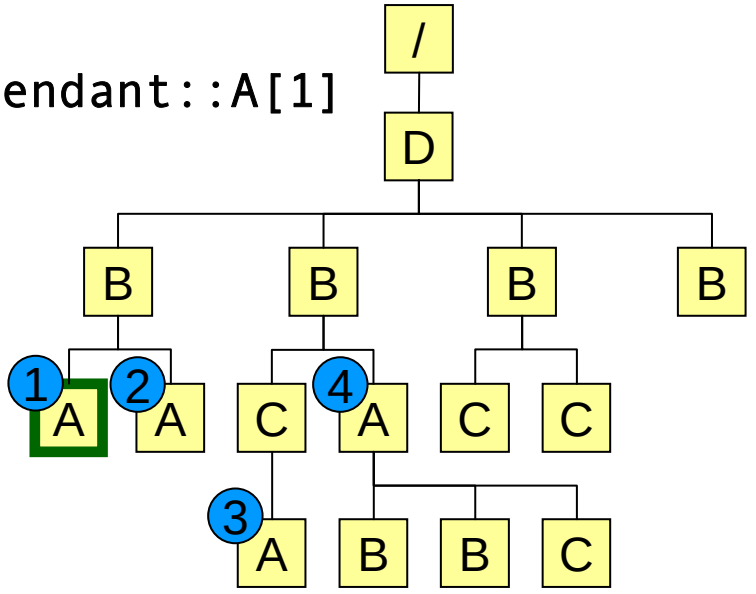
```
<adressbuch>
  <adresse>
    <name geschlecht="m">
      <vorname>Jim</vorname>
      <nachname>Panse</nachname>
    </name>
    <strasse>Primatenring 16</strasse>
    <ort plz="12345">Affenbach</ort>
  </adresse>
  <adresse>
    <name geschlecht="m">
      <vorname>Bernhard</vorname>
      <nachname>Diener</nachname>
    </name>
    <strasse>Dackelgasse 9</strasse>
    <ort plz="54321">Köteringen</ort>
  </adresse>
  <adresse>
    <name geschlecht="w">
      <vorname>Wanda</vorname>
      <nachname>Düne</nachname>
    </name>
    <strasse>Küstenstraße 1</strasse>
    <ort plz="77777">Süddeich</ort>
  </adresse>
</adressbuch>
```



Unterschied `//A[1]` und `/descendant::A[1]`

- `//A` und `/descendant::A` sammeln dieselben Knoten ein
- `//A[1]` und `/descendant::A[1]` beschreiben hingegen unterschiedliche Knotenmengen. Warum?
- die Pfadangabe `//A[1]` besteht aus drei Location-Steps
  - Langform: `/descendant-or-self::node()/A[1]`
  - → das Prädikat bezieht sich auf einen Knotentest der *child*-Achse
  - bei `/descendant::A[1]` bezieht sich das Prädikat auf einen Knotentest der *descendant*-Achse





Knotennummerierung und Positionstests

- soll sich ein Positionstest nicht auf die Position eines Knotens im XML-Baum sondern auf die Position in der Ergebnismenge beziehen, muss die Pfadangabe geklammert werden:
  `(`*pfadangabe*`)[position()=`*n*`]` oder kurz `(`*pfadangabe*`)[`*n*`]`

- `//A[1]` liefert alle Knoten des XML-Dokuments, die erstes *A*-Kindelement ihres Elternelements sind

- `(//A)[1]` liefert das erste im XML-Dokument vorkommende *A*-Element