

XML-Praxis

XPath

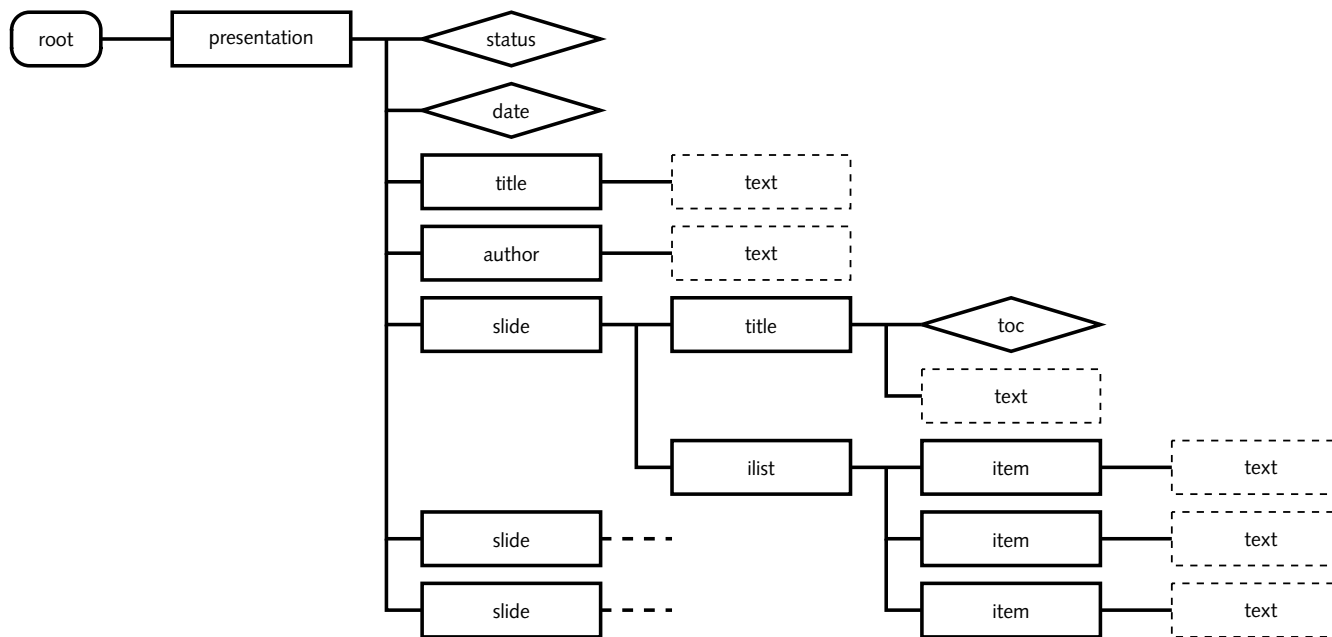
Jörn Clausen

`joern@TechFak.Uni-Bielefeld.DE`

Übersicht

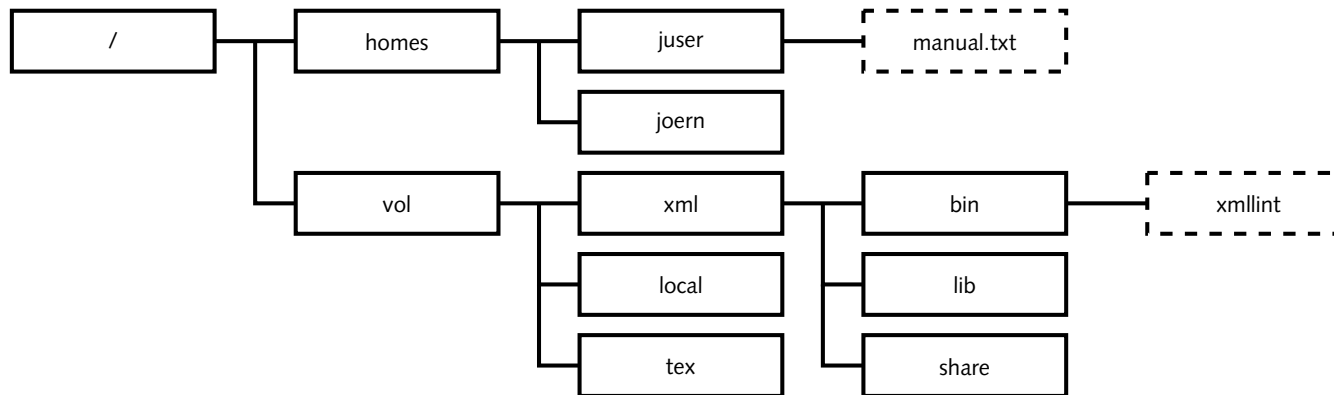
- XPath
- Namespaces

XML-Dokument als Baum



- weitere Text-Knoten durch whitespace
- Aufgabe: lokalisere einen (oder mehrere) Knoten

Analogie Dateisystem

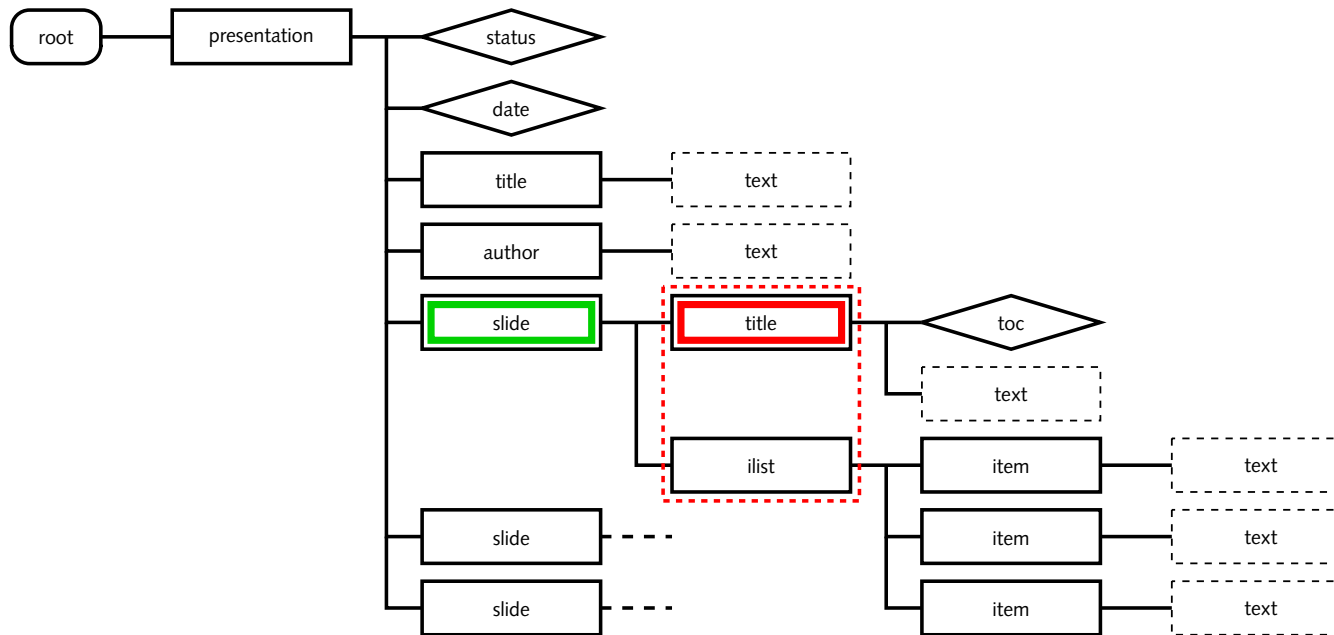


- absoluter Pfad: Wegbeschreibung vom Wurzelverzeichnis
`/vol/xml/bin/xmllint`
- relativer Pfad: Wegbeschreibung vom „aktuellen“ Verzeichnis
`../juser/manual.txt`
- Unterschied bei XML: gleichnamige Kindknoten

XPath

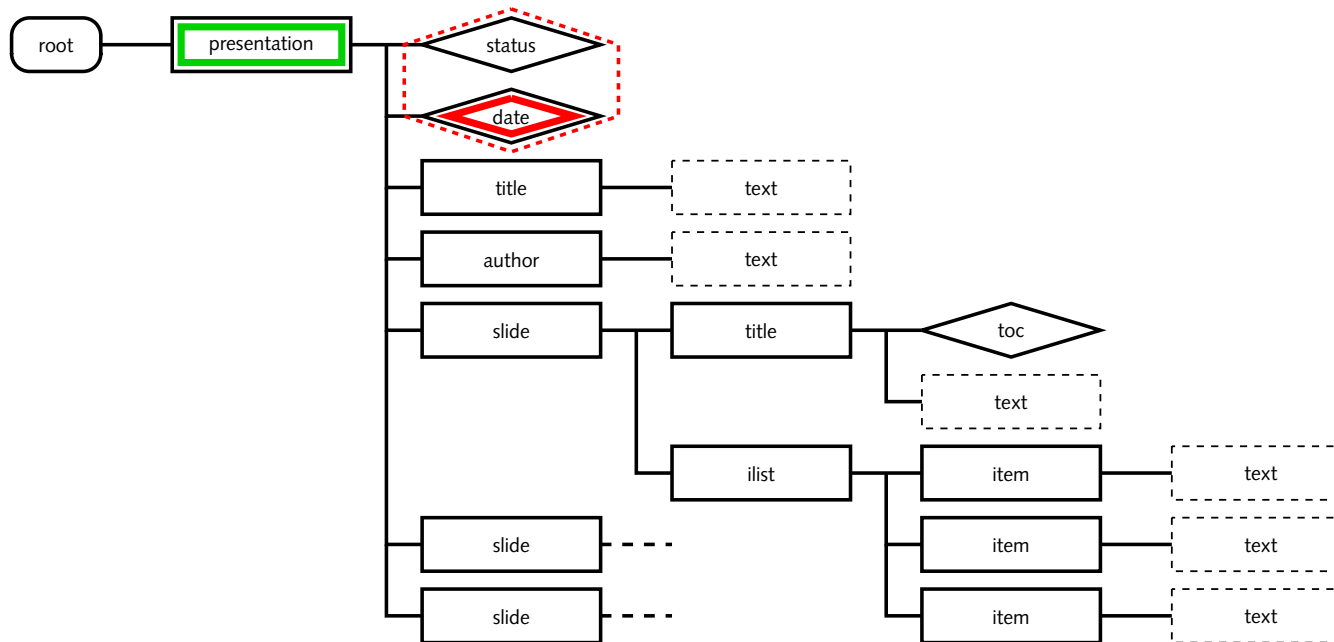
- XPath beschreibt Pfade im XML-Baum
- Bezugspunkt: Kontext-Knoten
- Knotentypen:
 - Element, Attribut, Text, Wurzelknoten
 - Kommentar, processing instruction, namespace
- Beziehungen über *Achsen*
- „Richtung“ und „Entfernung“ anderer Knoten
- XPath-Ausdruck evaluiert zu *node set*

XPath-Ausdrücke



`child::title`

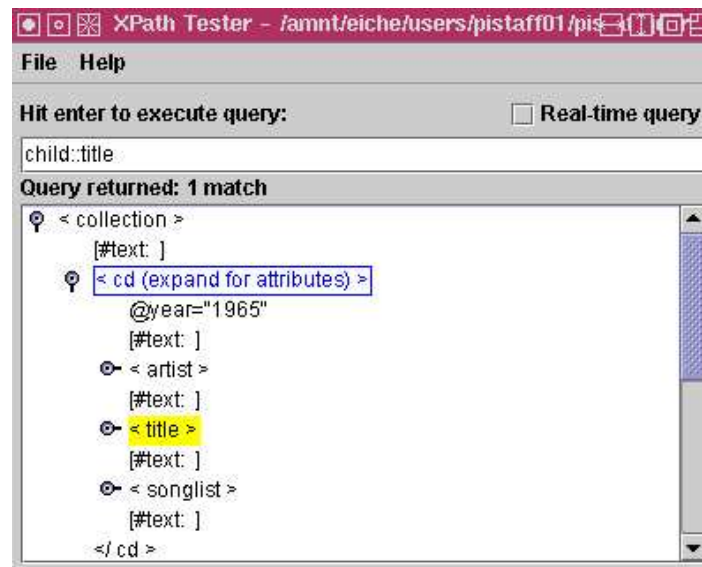
XPath-Ausdrücke, cont.



`attribute::date`

xpathtester

- Ergebnis von XPath-Ausdrücken visualisieren:



- Kontext-Knoten durch Anklicken bestimmen
- Ergebnis-Knoten werden gelb hervorgehoben
- Bug: Attribute werden nicht markiert

Aufgaben

- Probiere folgende XPath-Ausdrücke mit dem `xpath tester` und der Datei `cd-collection.xml` aus. Wähle zu jedem Ausdruck einen passenden Kontext-Knoten aus.

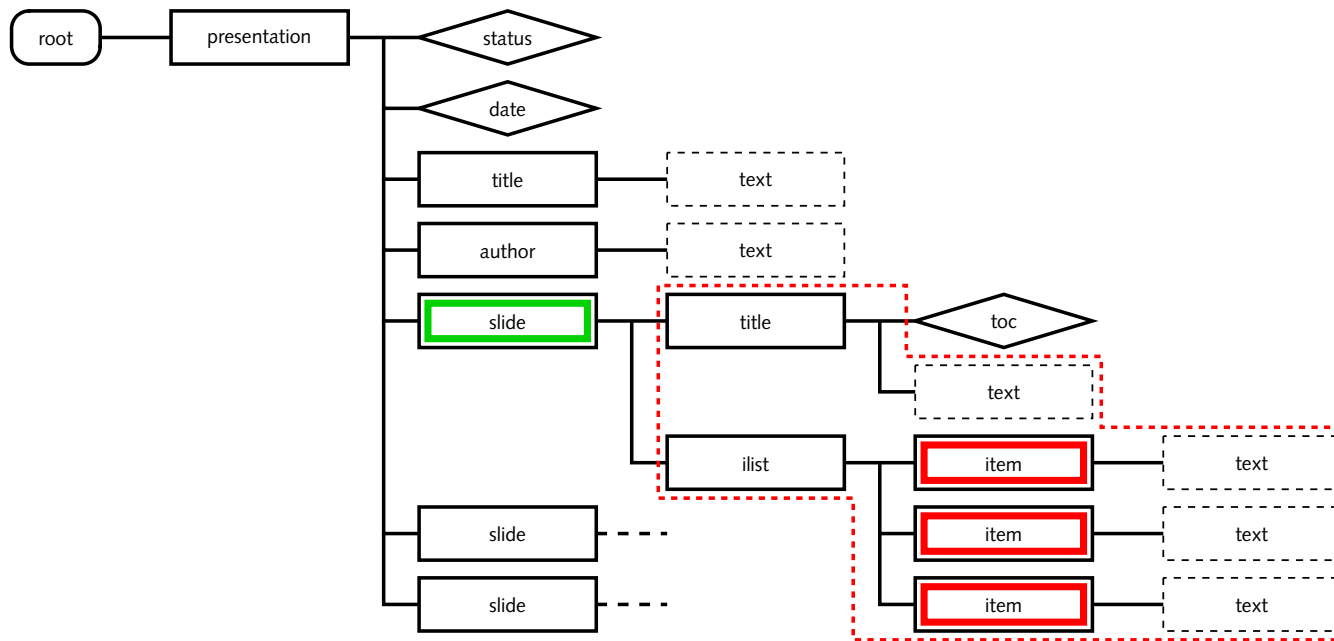
`child::cd`

`child::title`

`child::song`

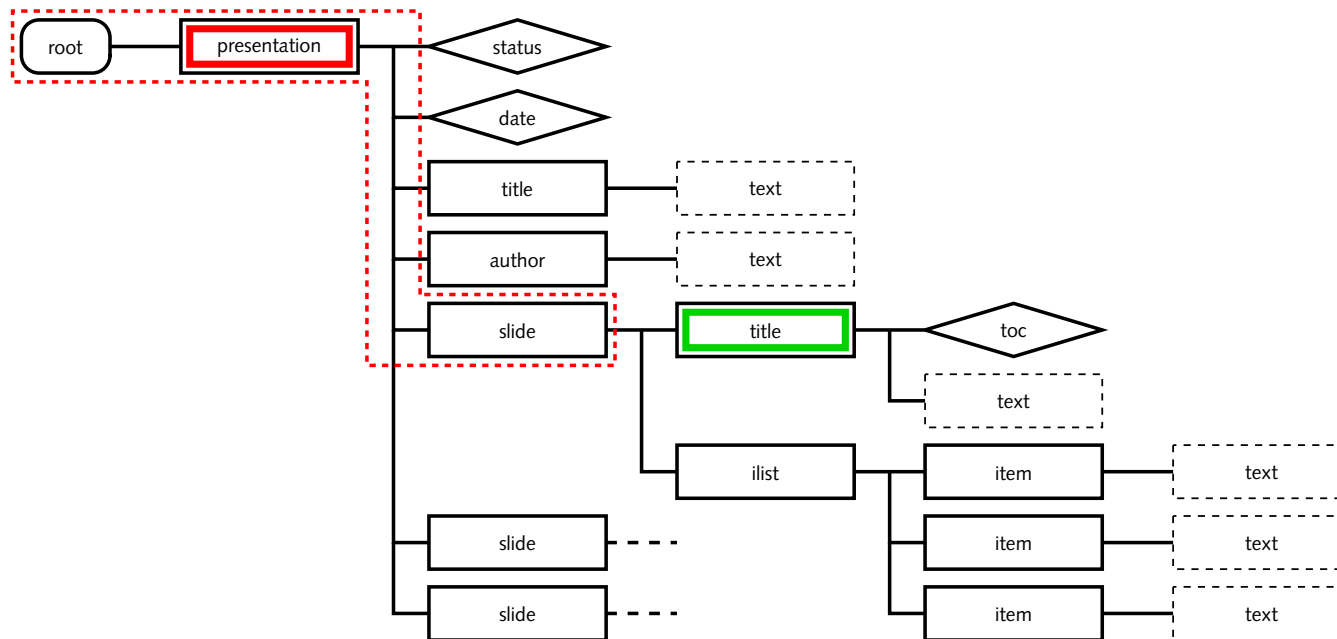
`attribute::year`

Achsen



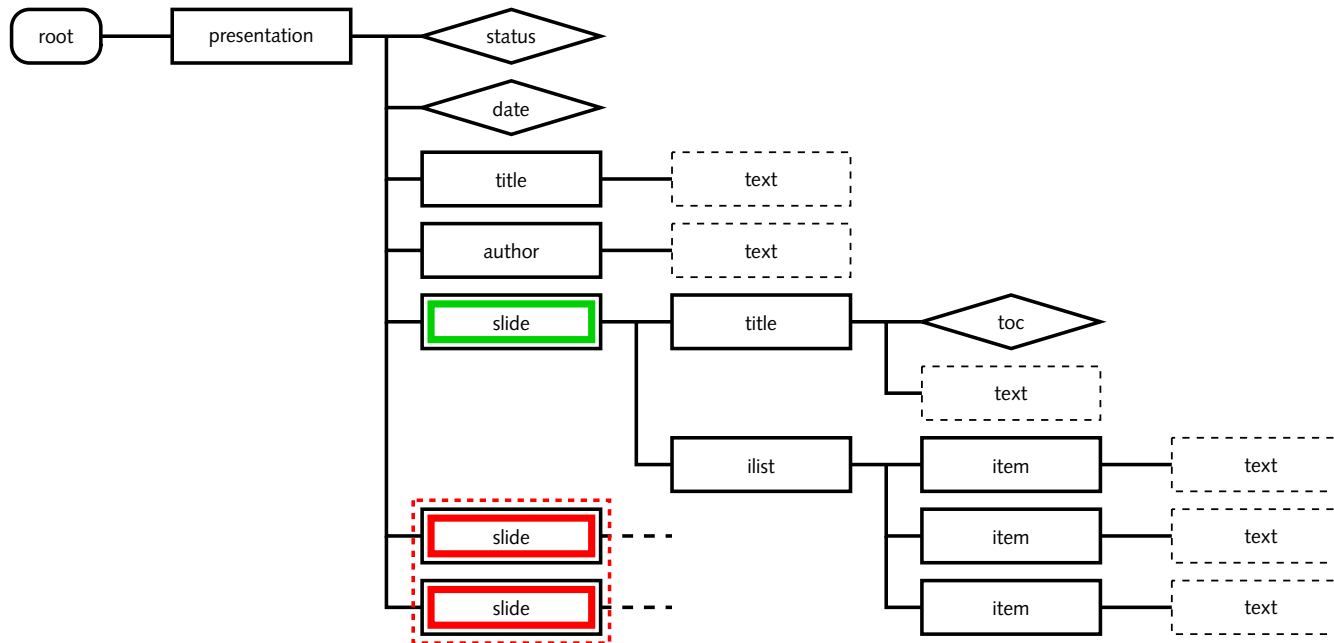
`descendant::item`

Achsen, cont.



`ancestor::presentation`

Achsen, cont.



`following-sibling::slide`

Aufgaben

- Probiere die folgenden Ausdrücke mit dem `xpathtester` aus. Finde wieder geeignete Kontext-Knoten.

`descendant::song`

`ancestor::title`

`following-sibling::cd`

`preceding-sibling::song`

`following::song`

`preceding::song`

Node Tests

- Auswahl von Knoten entlang der gewählten Achse:

`axis::node()` alle Knoten entlang der Achse

`axis::*` alle „geeigneten“ Knoten

`axis::text()` alle Text-Knoten

- Achtung: `<title toc="yes">`

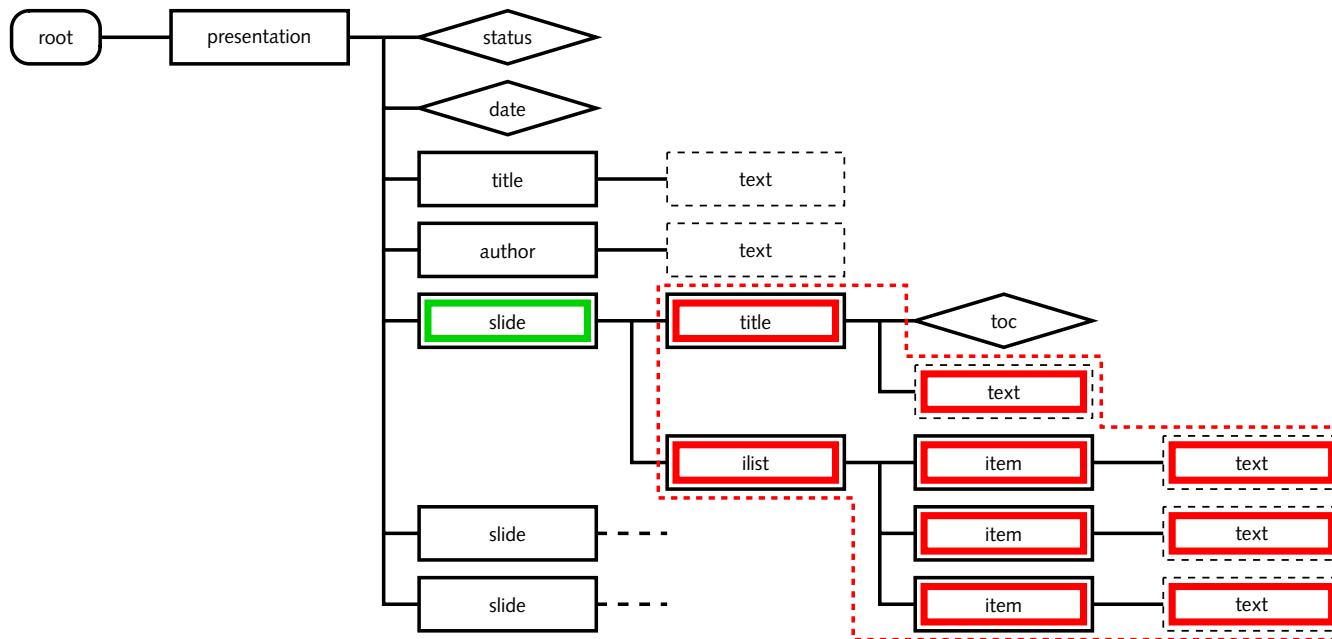
- `title` ist Vaterknoten von `toc`

`title ∈ parent::node()`

- aber: `toc` nicht Kindknoten von `title`

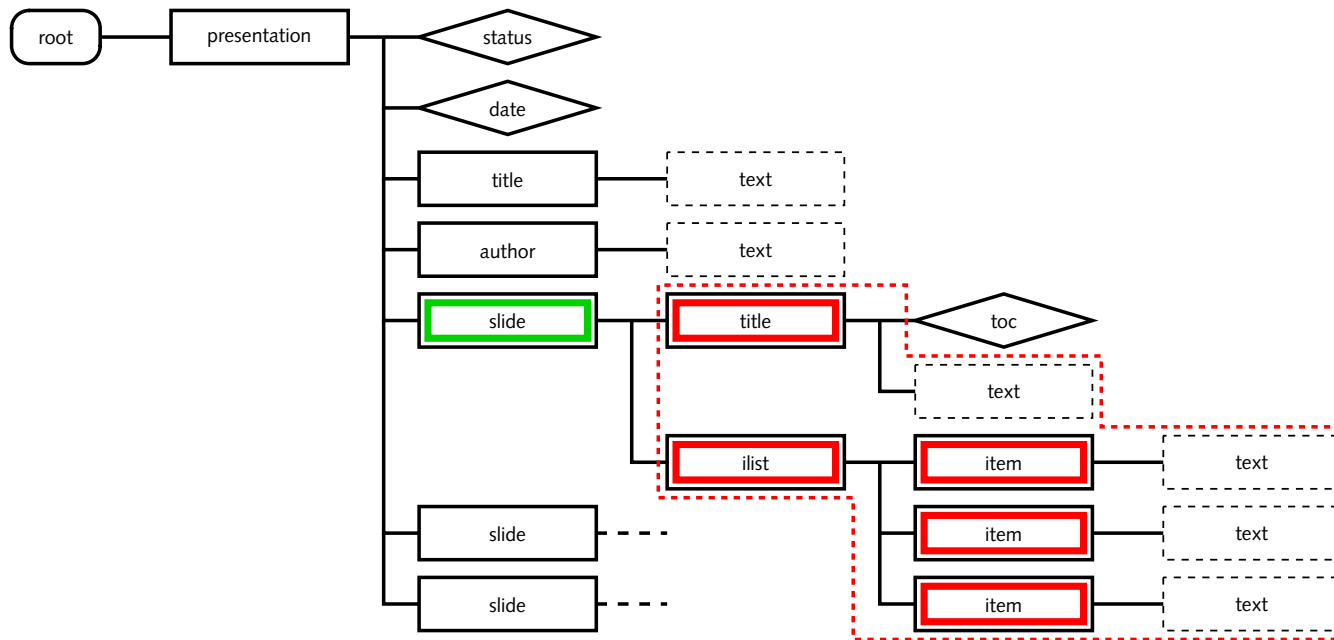
`toc ∉ child::node()`

Node Tests, cont.



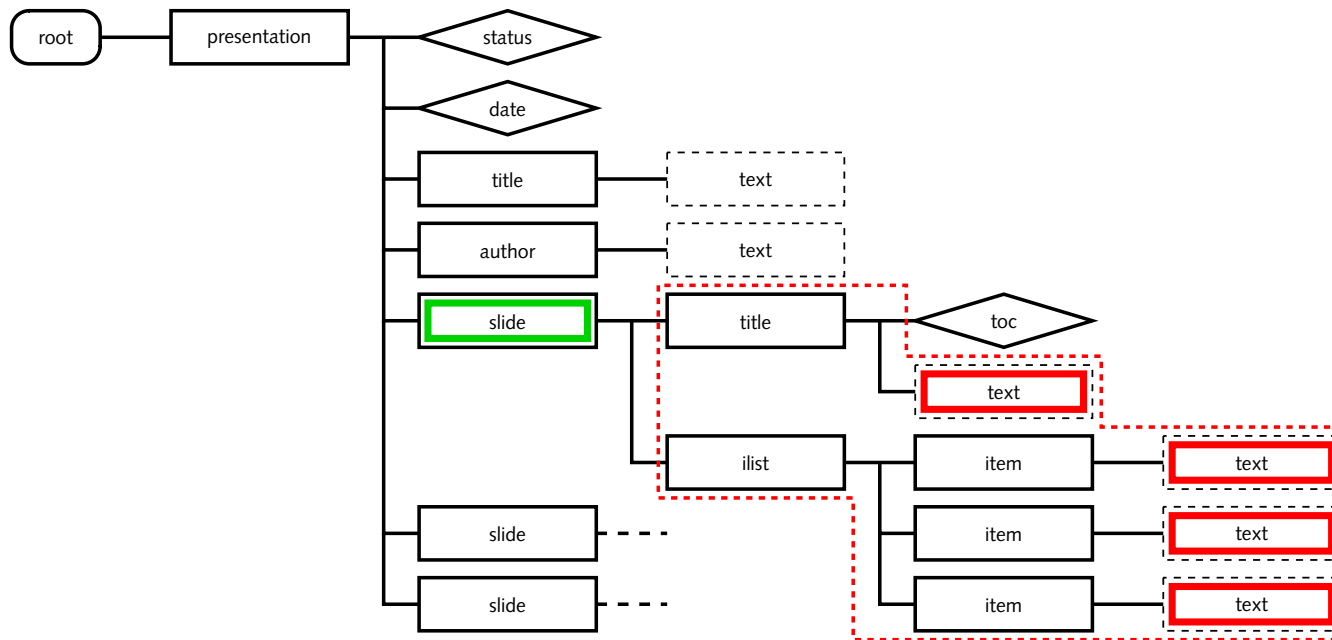
`descendant::node()`

Node Tests, cont.



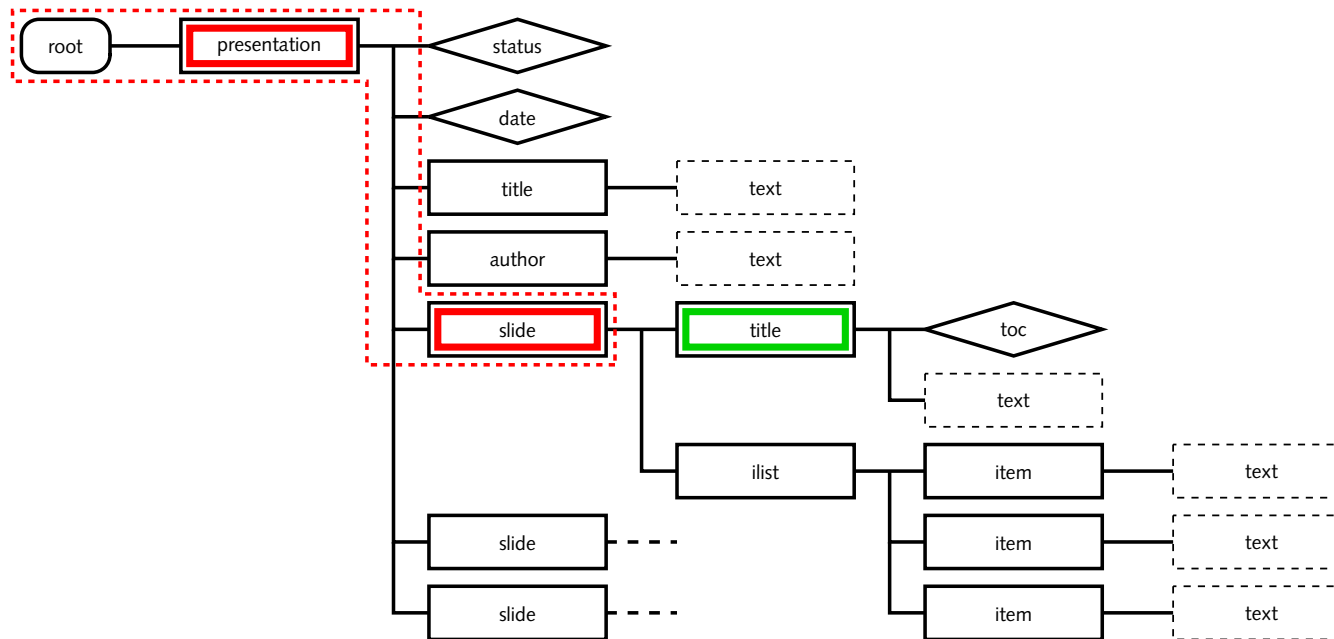
descendant::*

Node Tests, cont.



`descendant::text()`

Node Tests, cont.



`ancestor::*`

Aufgaben

- Probiere diese XPath-Ausdrücke aus. Vergleiche die Ergebnisse, die die Ausdrücke zusammen mit verschiedenen Kontext-Knoten liefern.

```
descendant::node()
```

```
descendant::*
```

```
descendant::text()
```

```
self::node()
```

- Wieviele bzw. welche Ergebnis-Knoten liefern diese Ausdrücke, wenn man als Kontext-Knoten eine `cd` wählt? Erkläre die Unterschiede.

```
following-sibling::node()
```

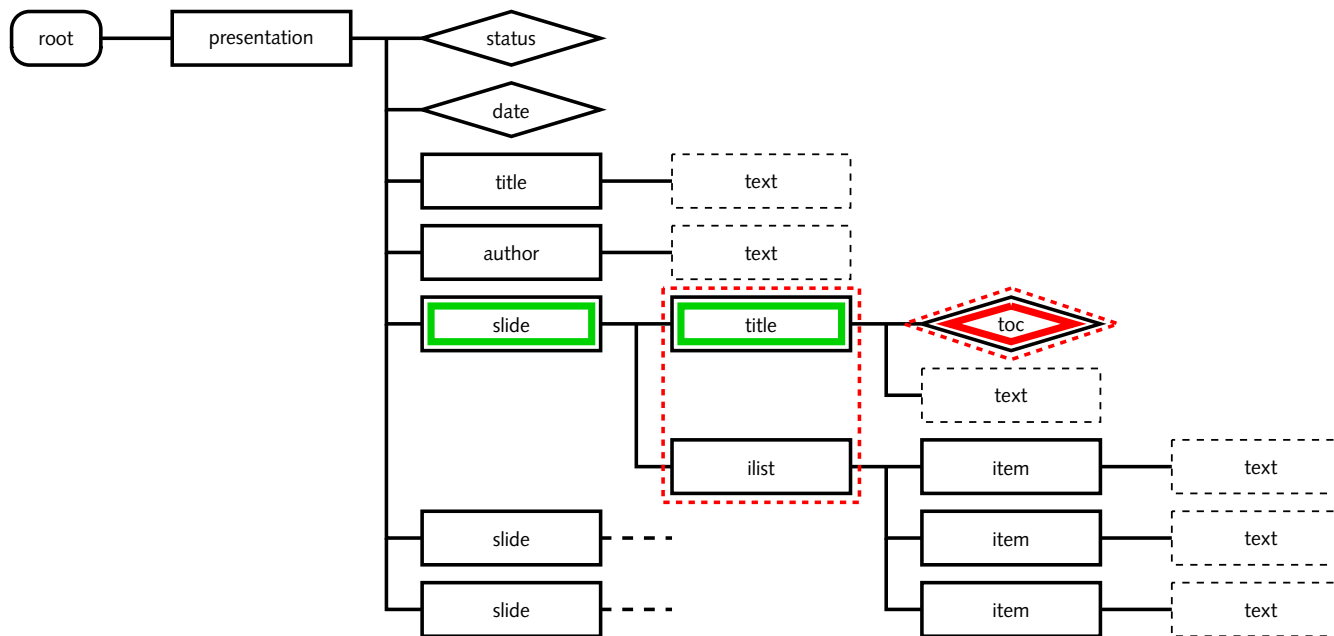
```
following-sibling::*
```

```
following-sibling::text()
```

- Was erwartest Du bei diesem Ausdruck?

```
attribute::text()
```

Pfade



`child::title/attribute::toc`

Pfade, cont.

- *location path* besteht aus *location steps*
- Knoten-Menge eines location step Kontext-Knoten des folgenden
- location steps werden durch „ / “ (slash) getrennt
`child::title/attribute::toc`
- absoluter Pfad:
`/child::presentation/child::author/child::text()`

Kurzschreibweise

- abkürzende Syntax:

child:: (*nichts*)

attribute:: @

self::node() .

parent::node() ..

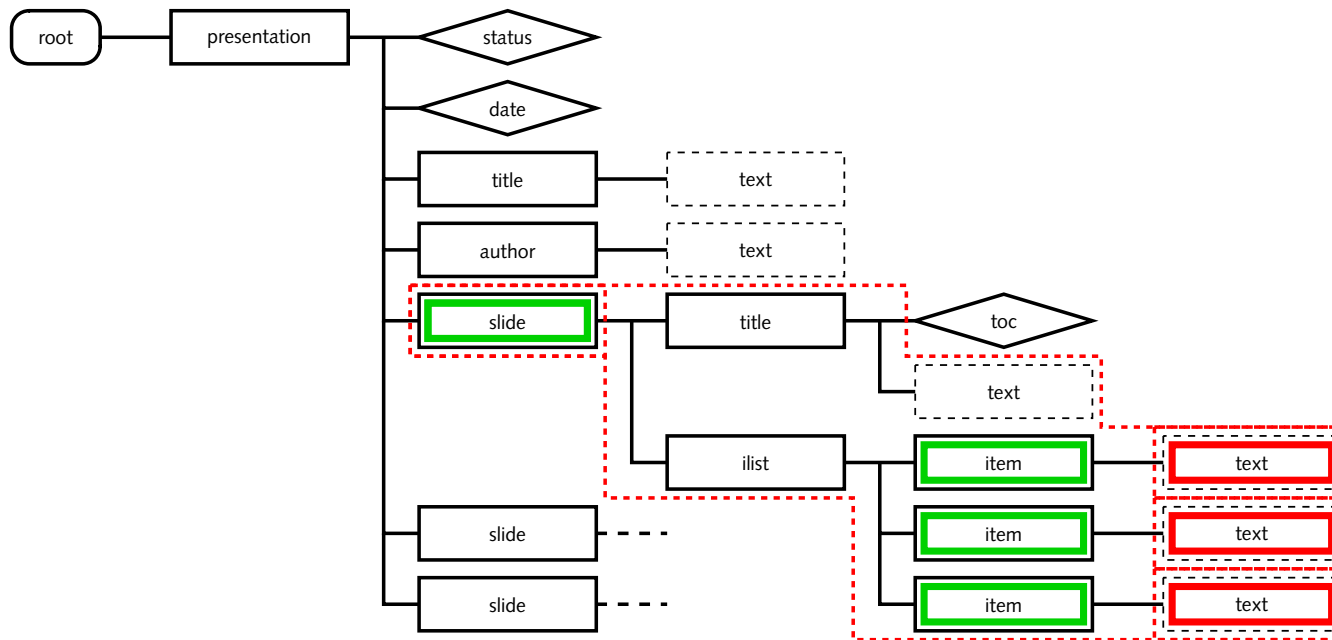
/descendant-or-self::node()// //

- Pfade wie im Dateisystem

title/@toc

/presentation/author/text()

Pfade, cont.



`./item/text()`

Aufgaben

- Wie sehen die zugehörigen XPath-Ausdrücke aus?
 - alle CDs
 - alle Künstler
 - alle Erscheinungsjahre
 - alle Lieder einer CD (wähle eine `cd` als Kontext-Knoten)

Prädikate

- (weitere) Einschränkung der Knoten-Menge:

```
title[@toc="yes"]  
ilist/item[position()=2]  
slide[position()=last()]
```

- Welche Knoten werden ausgewählt?

```
slide/title[@toc="yes"]  
slide[title/@toc="yes"]
```

- Prädikat bei jedem location step möglich:

```
slide[title/@toc="yes"]/ilist/item[1]
```

- logische Operatoren

```
slide/title[not(@toc="yes")]  
slide[title/@toc="yes" and position() != last()]
```

Aufgaben

- Finde die passenden XPath-Ausdrücke zu `cd-collection.xml`:
 - das erste Lied von jeder CD
 - die Titel aller CDs, die vor 1970 erschienen sind
 - alle Lieder, die zwischen 1970 und 1980 aufgenommen wurden
 - alle CDs, für die das Erscheinungsjahr angegeben ist
 - das letzte Lied aller CDs, die nach 1980 aufgenommen wurden

Funktionen

- bereits gesehen: `position()` und `last()`

- String-Operationen:

```
/presentation/slide[contains(title,'XML')]
```

```
/presentation/slide[contains(title,concat('X','M','L'))]
```

```
/presentation/slide[string-length(title) < 20]
```

- weitere: `starts-with`, `substring-before`,
`substring-after`, `substring`

Funktionen, cont.

- Arithmetische Operatoren:

```
item[position() = last()-1]  
item[position() > 2]
```

- weitere: Grundrechenarten, Division `div`, Modulo `mod`
- XPath-Ausdruck kann auch zu Zahl, Text oder Boolean evaluieren:

```
count(/presentation/slide)  
sum(/prod/weight)  
concat(@href, ".html")
```

Aufgaben

- Finde die passenden XPath-Ausdrücke zu `cd-collection.xml`:
 - alle Beatles-CDs
 - das letzte Lied auf jeder Beatles-CD
 - die Titel aller CDs der Beatles und der Rolling Stones
 - jedes zweite Lied einer CD
 - die Anzahl aller Lieder
 - die Anzahl aller Beatles-Lieder
- Was berechnen die folgenden Ausdrücke:

```
sum(//@year)
```

```
count(//song) div count(//cd)
```

Programmieren mit XPath

- Perl-Modul XML::XPath

```
my $xp = XML::XPath->new(filename => $ARGV[0]);

my $pres = ($xp->find('/presentation')->get_nodelist)[0];
print $pres->findvalue('title'), "\n";
print $pres->findvalue('author'), "\n";
foreach my $slide ($pres->find('slide')->get_nodelist) {
    print $slide->findvalue('title'), "\n";
    foreach my $item ($slide->find('ilist/item')->get_nodelist) {
        print $item->findvalue('.'), "\n";
    }
}
```

- einfacher als DOM
- aber: hoher Speicherverbrauch

Namespaces

- XML-Sprachen für wiederkehrende Probleme:
 - Tabellen
 - mathematischer Formelsatz
 - genetische Sequenzen
 - ...
- Kombination/Einbettung von Sprachen
- Beispiel: (X)HTML-Dokument mit Formeln in MathML
- Problem: Was gehört zu welcher XML-Sprache?

Verwendung von Namespaces

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>...</head>
  <body>
    <p>also sprach Pythagoras:</p>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mrow>
        <msup><mi>x</mi><mn>2</mn></msup>
        <mo>+</mo>
        <msup><mi>y</mi><mn>2</mn></msup>
        ...
      </mrow>
    </math>
  </body>
</html>
```


alternative Notation von Namespaces

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ml="http://www.w3.org/1998/Math/MathML">
  <head>...</head>
  <body>
    <p>also sprach Pythagoras:</p>
    <ml:math>
      <ml:mrow>
        <ml:msup><ml:mi>x</ml:mi><ml:mn>2</ml:mn></ml:msup>
        <ml:mo>+</ml:mo>
        <ml:msup><ml:mi>y</ml:mi><ml:mn>2</ml:mn></ml:msup>
        ...
      </ml:mrow>
    </ml:math>
  </body>
</html>
```

Namespaces, cont.

- namespace prefix beliebig
- namespace URI ausschlaggebend, muß exakt übereinstimmen

```
<html xmlns:ml="http://www.w3.org/1998/Math/MathML">  
<ml:msup>...</ml:msup>
```

```
<html xmlns:MathML="http://www.w3.org/1998/Math/MathML">  
<MathML:msup>...</MathML:msup>
```

- keine Verknüpfung mit Grammatik/Schema
- URL als identifier extrem schlechte Wahl
- schlecht mit DTDs zu realisieren