



# XSLT

**Robert Tolksdorf**  
**Freie Universität Berlin**  
**Institut für Informatik**  
**Netzbasierte Informationssysteme**  
**[tolk@inf.fu-berlin.de](mailto:tolk@inf.fu-berlin.de)**

- eine Familie von Sprachen zur Erzeugung von Layouts für XML-Dokumente
- keine vordefinierten Tags

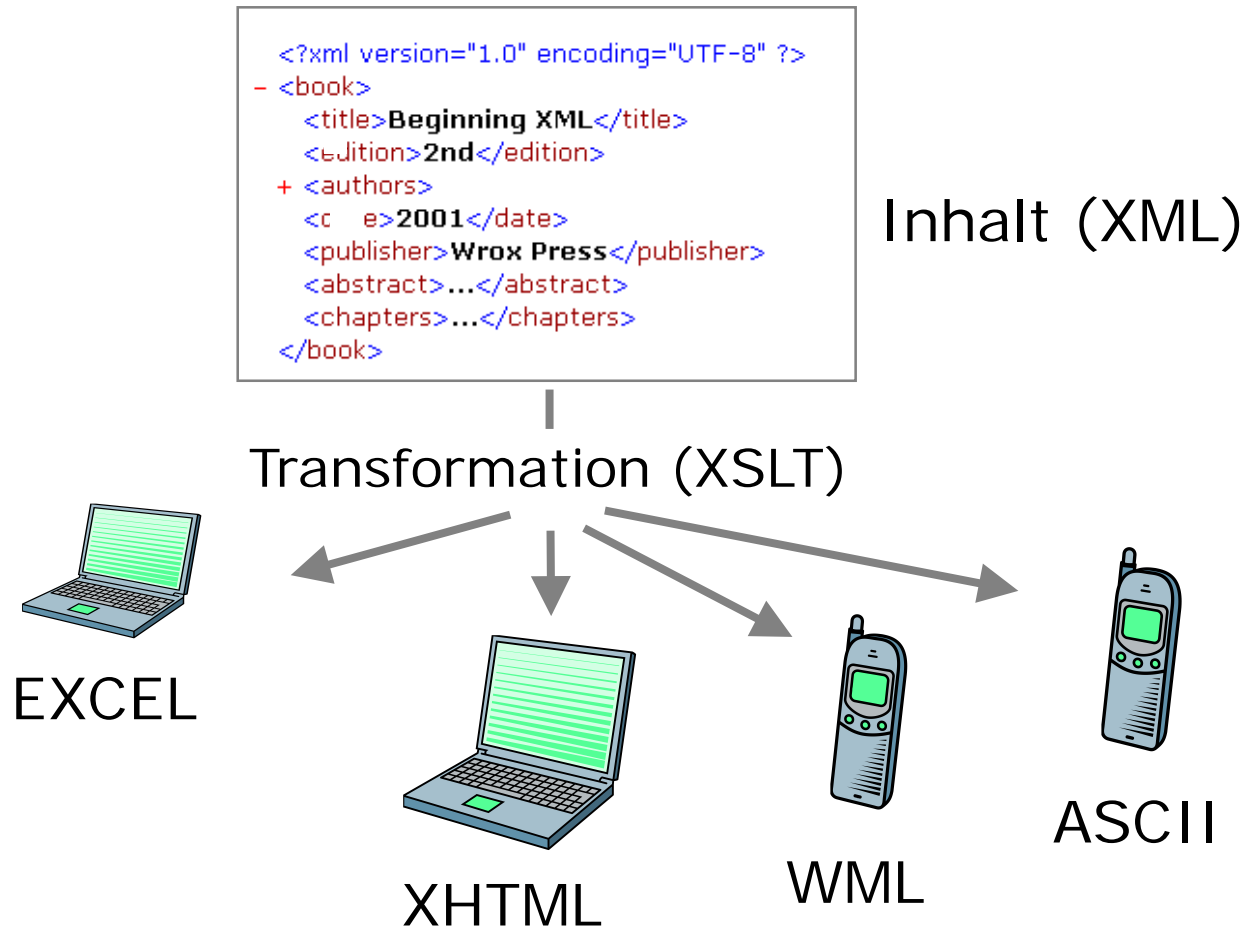
## **XSL beschreibt, wie XML-Dokumente dargestellt werden sollen**

- besteht aus:
  - XPath – Navigations-/Selektion für XML-Dokumente
  - XSLT – Transformationssprache für XML-Dokumente
  - XSL-FO – Formatierungssprache für XML-Dokumente

# Warum XML transformieren?

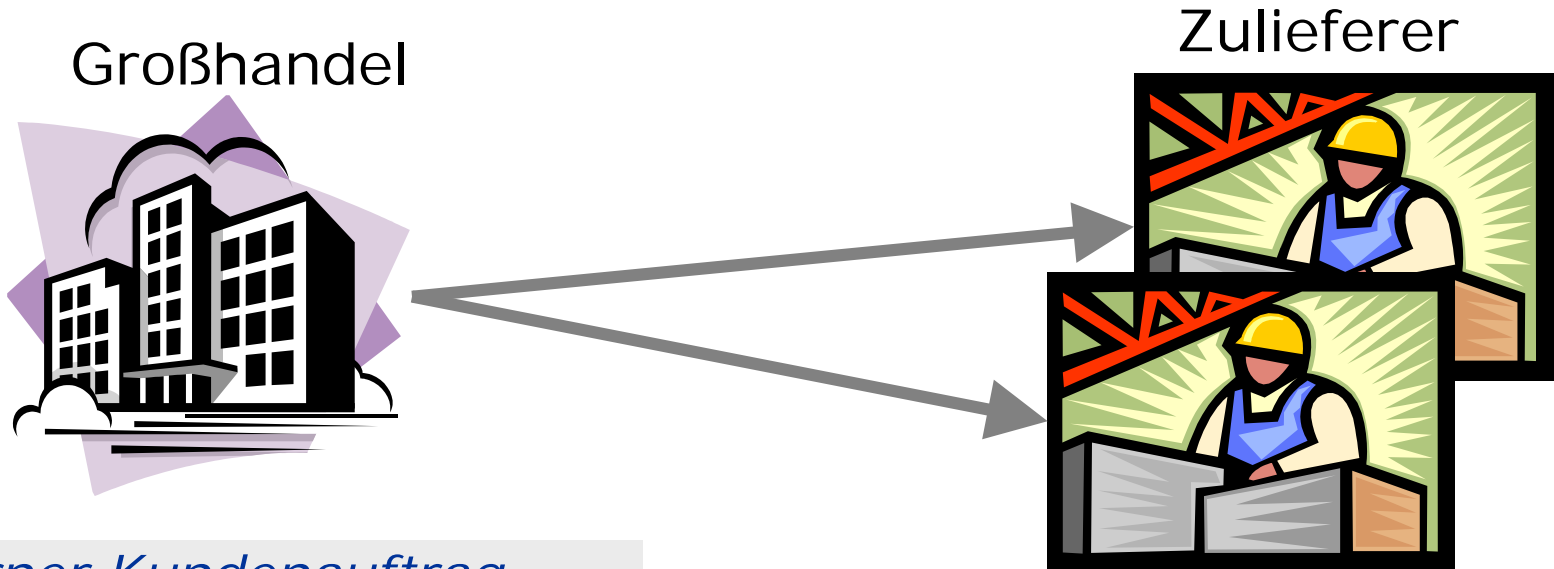
- **Trennung Inhalt und Präsentation**
  - XML trennt Inhalt von Präsentation (Layout)
  - Für eine entsprechende Darstellung müssen XML-Inhalte transformiert werden:
    - XML-Inhalt → Layout
- **Inhaltliche Transformationen**
  - Daten mit XML repräsentiert
  - unterschiedliche Sichten (Views) auf XML-Inhalte erfordern Transformationen:
    - XML-Inhalt → XML-Inhalt

# XML-Inhalt → Layout



- Multi-Delivery: unterschiedliches Layout von Inhalten
- Beachte: XHTML, WML  $\subset$  XML

# XML-Inhalt → XML-Inhalt



## *interner Kundenauftrag*

- ~~Name des Verkäufers~~
- Datum
- Produktbezeichnung aus Großhandelskatalog
- Anzahl
- ~~Kunde~~

anpassen

anpassen

übernehmen

## *externer Zuliefererauftrag*

- Datum
- Produktbezeichnung aus Zuliefererkatalog
- Anzahl
- Auftraggeber

## Kundenauftrag

```
<?xml version="1.0"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>
    <month>1</month>
    <day>13</day>
    <year>2000</year>
  </date>
  <customer>Sally Finkelstein</customer>
</order>
```

andere Sicht (view)  
auf XML-Inhalt



## Zulieferauftrag

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <customer>Company A</customer>
  <date>2000/1/13</date>
  <item>
    <part-number>E16-25A</part-number>
    <description>Production-Class Widget</description>
    <quantity>16</quantity>
  </item>
</order>
```

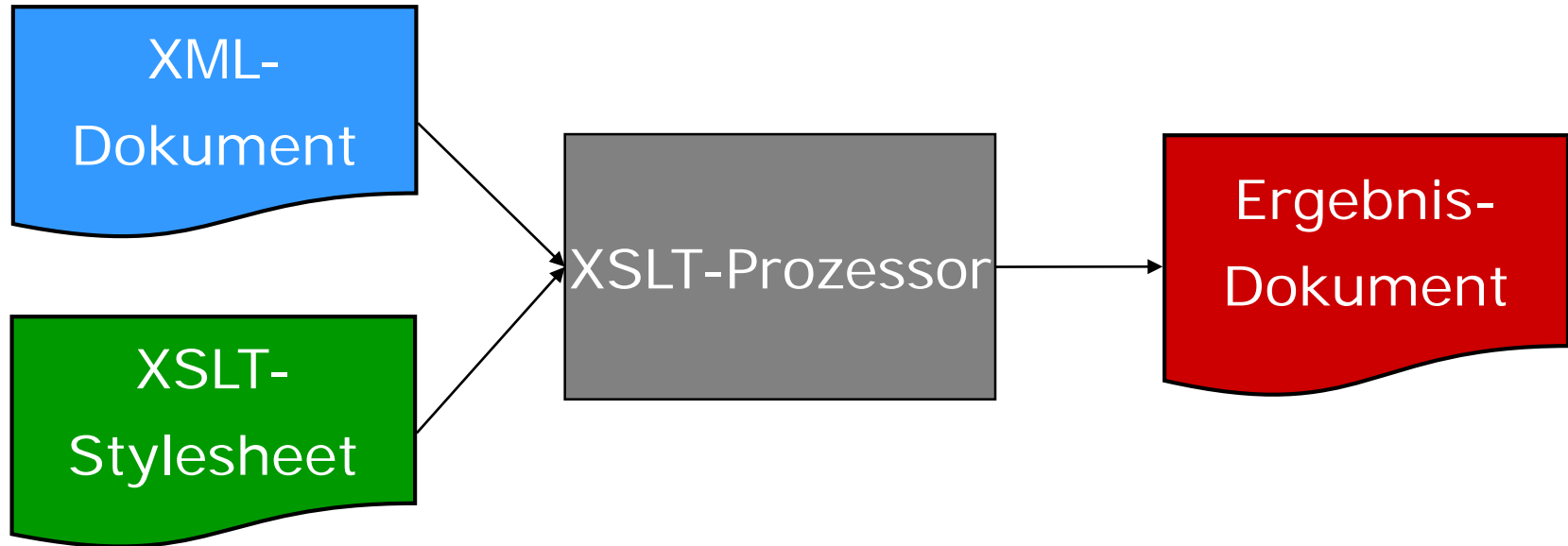


## **eXtensible Stylesheet Language Transformation (XSLT) - Einführung**

# Was ist XSLT?

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





- Verknüpfung zwischen Stylesheet & Dokument im Dokument

```
<?xml version=".. "?>  
<?xml-stylesheet type="text/xsl" href="file.xsl"?>  
  <element>  
    ...  
  </element>
```

Quelle: H. Vonhoegen, „Einstig in XML: Grundlagen, Praxis, Referenzen“, ISBN 978-3-8362-1074-4, 2007

- **Xalan**
  - Open Source XSLT-Prozessor
  - <http://xalan.apache.org/>
  - default Xerces XML-Parser
  - unterstützt W3C Recommendations: XSLT 1.0 & XPath 1.0
  - Xalan C (C++) & Xalan J (Java)
- **SAXON**
  - Open Source XSLT-Prozessor
  - <http://saxon.sourceforge.net/>
- **Mittlerweile viele weitere**
  - <http://www.w3.org/Style/XSL/>

# Programmierparadigma

- XSLT-Programm (stylesheet) = Menge von Transformationsregeln
- Transformationsregel (template)
  - Für einen passenden Knoten X erzeuge aus X im Ursprungsdocument Y im Ergebnisdokument
- Beispiel:

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

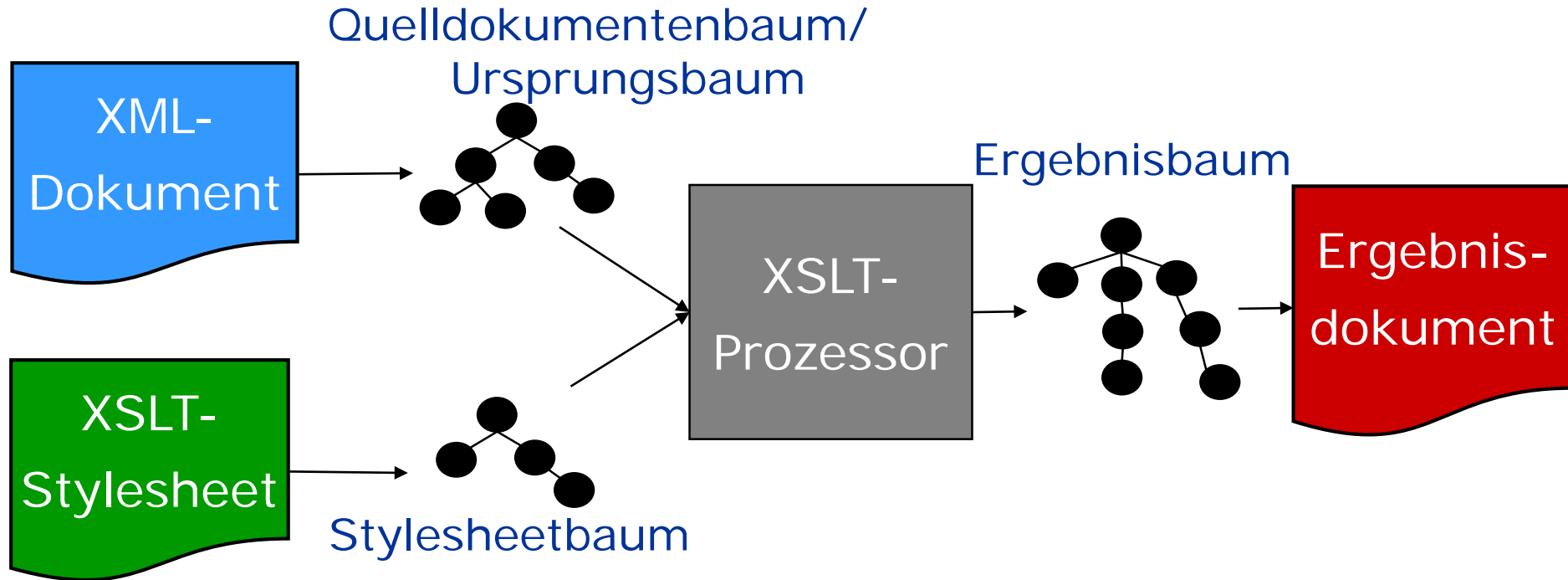
```
<order>
  ...
  <item>Item</item>
</order>
```



```
<p>Item</p>
```

- Identifizierung von Unterstrukturen mit XPath

Quelle: H. Vonhoegen, „Einstig in XML: Grundlagen, Praxis, Referenzen“, ISBN 978-3-8362-1074-4, 200



## XSLT-Transformationsregeln

- immer auf Ursprungsdocument(en) angewandt, niemals auf Zwischenergebnissen
- keine Seiteneffekte:
  - Template angewandt auf X liefert immer das gleiche Ergebnis
  - = Templates haben keine Zustände
  - ⇒ keine Variablen, die überschrieben werden können
  - ⇒ **funktionales** Programmierparadigma

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  ...
</xsl:stylesheet>
```

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

```
<?xml version="1.0"?>  
<xsl:stylesheet  
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  
  version="1.0">  
  <xsl:template match="...">  
    ...  
  </xsl:template>  
</xsl:stylesheet>
```

Muster

Template  
(Template-Regeln)

- „Nehme aktuellen Knoten und wende passendes Template an. Wenn keins vorhanden passen die *Default Templates*.“
- Nicht ganz zutreffend:  
„Suche im Ursprungsdokument Unterstruktur X und erzeuge hieraus Ergebnisdokument Y!“ (ungefähr...)
- zwei Möglichkeiten, Y zu erzeugen:
  - neue Inhalte erzeugen
  - Inhalte von X nach Y übertragen.
- beide Möglichkeiten beliebig miteinander kombinierbar



# 1. Neue Inhalte erzeugen (I)

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

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

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

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

# 1. Neue Inhalte erzeugen (II)

- statt üblicher XML-Syntax

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

auch möglich:

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

## 2. Inhalte übertragen

- **<xsl:copy-of select="."> Element**
  - Kopiert aktuellen Teilbaum
  - aktueller Teilbaum: Baum, der vom aktuellen Knoten aufgespannt wird, einschließlich aller Attribute und PCDATA
- **<xsl:copy> Element**
  - Kopiert aktuellen Knoten ohne Kind-Elemente, Attribute und PCDATA
  - Kopiert nur Wurzel-Element des aktuellen Teilbaums
- **<xsl:value-of select="."> Element**
  - Extrahiert PCDATA, das im aktuellen Teilbaum vorkommt

## Ergebnisdokument

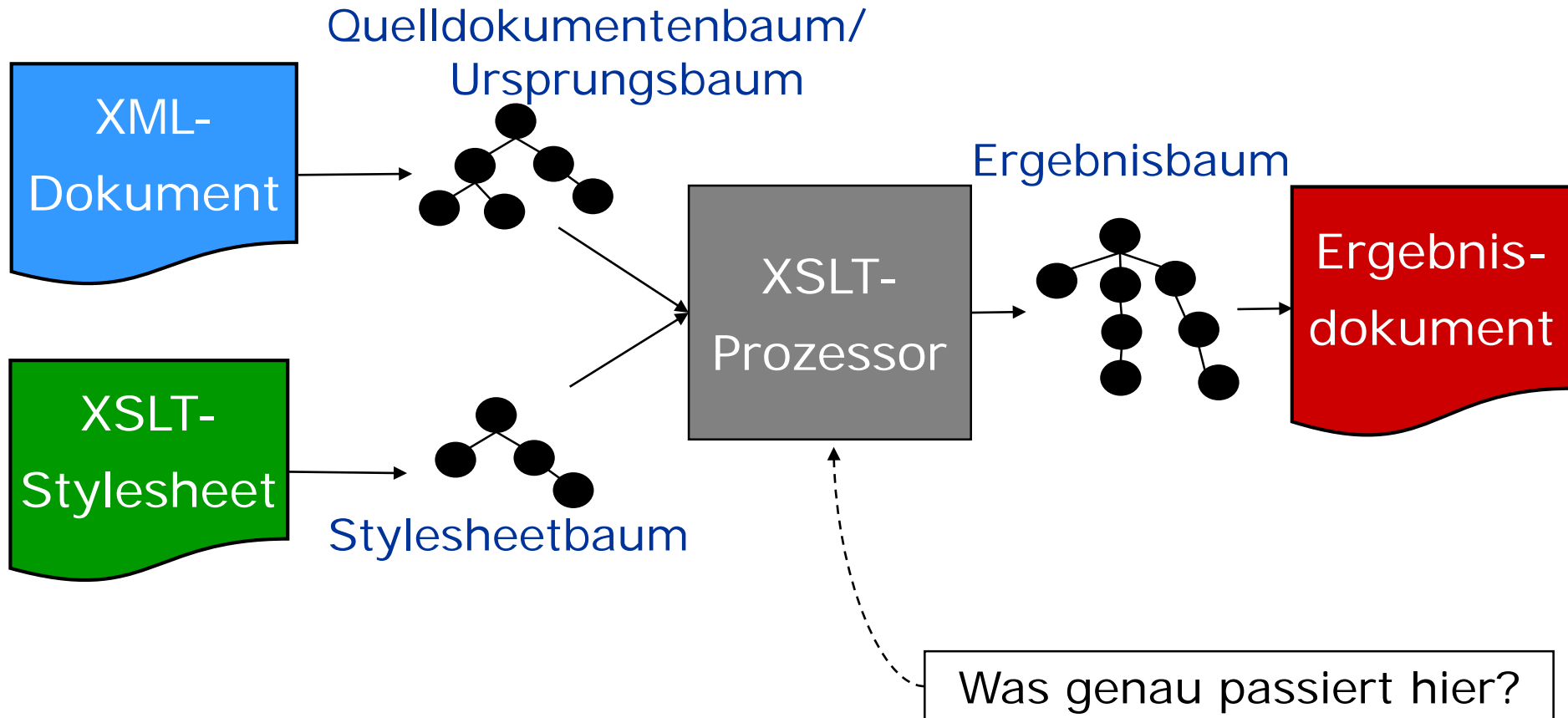
```
<xsl:template match="p">  
  <DIV>  
    <xsl:copy-of select="."/>  
  </DIV>  
  <DIV>  
    <xsl:copy/>  
  </DIV>  
  <DIV>  
    <xsl:value-of select="."/>  
  </DIV>  
</xsl:template>
```

```
<source>  
  <p id="a12">Compare  
    <B>these constructs</B>.  
  </p>  
</source>
```

```
<DIV>  
  <p id="a12">Compare  
    <B>these constructs</B>.  
  </p>  
</DIV>  
<DIV>  
  <p/>  
</DIV>  
<DIV>  
  Compare these constructs.  
</DIV>
```

# XSLT-Prozessor im Transformationsprozess

Quelle: H. Vonhoegen, „Einstig in XML: Grundlagen, Praxis, Referenzen“, ISBN 978-3-8362-1074-4, 200



a)  $K :=$  Dokument-Wurzel ("/") des Ursprungsdocumentes

b) Identifiziere alle Templates, die auf  $K$  anwendbar sind.

1. Ist genau ein Template anwendbar

- wende es an
- Fertig.

2. Sind mehre Templates anwendbar, dann

- wende das speziellste an
  - z.B. ist "/order" spezieller als "/\*".
- Fertig.

3. Ist kein Template anwendbar

- wiederhole b) für alle Kinder  $K'$  von  $K$  mit  $K := K'$ .

- mehrere Templates auf den gleichen Knoten anwendbar
- Lösung → Prioritätsregeln:
  1. Eine spezifische Information hat Vorrang vor einer Regel für allgemeinere Information  
Beispiel: `match="/buch/authors/autor"`  
`match="//autor"`
  2. Suchmuster mit Wildcards (\* oder @\*) sind allgemeiner als entsprechende Muster ohne Wildcards
  3. Nur wenn 1. & 2. nicht zutreffen → Reihenfolge der Templates entscheidend
  4. Priorität der Templates durch Attribut `priority` bestimmbar
    - Standard = 0
    - niedrigere Priorität < 0 < höhere Priorität

# Transformations-Beispiel

## Stylesheet

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

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

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

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

## Dokument

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

kein  
Template  
anwendbar

Template  
"A" wird  
angewandt

## Ausgabe

a1  
a2

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



# Templates mit Rekursion

## Stylesheet

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

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

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

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

## Dokument

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

## Ausgabe

```
a1
  b1
  b2
a2
  b3
  b4
  c1
    d1
  b5
  c2
```



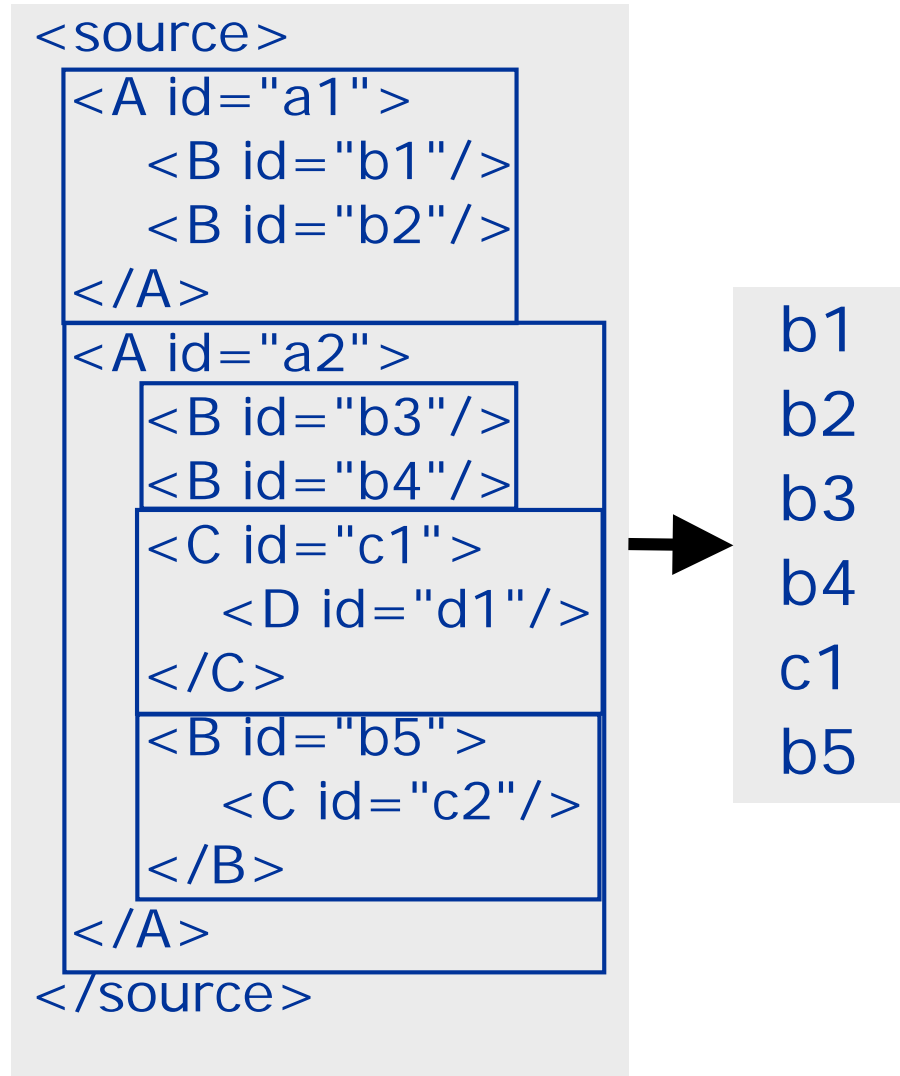
- `<xsl:apply-templates/>`
  - versucht Templates auf Kinder des aktuellen Knotens anzuwenden
  - Kind bedeutet hier:  
Kind-Element, Text-Knoten oder Attribut-Knoten
  - Mit `<xsl:apply-templates select = "..."/>` auch rekursiver Aufruf an beliebiger Stelle möglich
  - Vorsicht: Terminierung nicht automatisch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

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

# Iteration statt Rekursion

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

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





## XSLT – Templates: vordefinierte Templates

# Vordefinierte Templates

- 1. vordefiniertes Template
  - realisiert rekursive Anwendung, wenn kein Template auf den aktuellen Knoten passt
- 2. vordefiniertes Template
  - kopiert PCDATA des aktuellen Knotens in das Ergebnisdokument
- Leeres Stylesheet
  - traversiert gesamtes Ursprungsdokument und extrahiert dabei PCDATA und Attribut-Werte
- Überschreiben
  - Vordefinierte Templates können durch speziellere Templates überschrieben werden

# 1. vordefiniertes Template

```
<xsl:template match="*|/">  
  <xsl:apply-templates/>  
</xsl:template>
```

1. wird zuerst auf Dokument-Wurzel ("/") angewandt
  2. versucht alle Templates anzuwenden
  3. wird auf alle Kind-Elemente ("\*") angewandt
- realisiert rekursive Anwendung des XSLT-Prozessors
  - wird von jedem speziellerem Template überschrieben:  
z.B. sind "/" und "item" spezieller als "\*|/"
  - spezielleres Template anwendbar ⇨  
keine automatische rekursive Anwendung

## 2. vordefiniertes Template

```
<xsl:template match="text()|@*">  
  <xsl:value-of select="."/>  
</xsl:template>
```

- Template wird auf PCDATA text() angewandt
- text(): XPath-Funktion, selektiert PCDATA
- Template überträgt PCDATA in das Ergebnisdokument

# Ausgabe von Attributen

- Wählt "text()|@\*" nicht PCDATA und Attributwerte aus?
- Ja, aber!
- Attribut A des Element E
  - Hat E als Elternknoten in der Datenstruktur
  - Ist *kein* Kindknoten von E in XML
- Template muss für die Ausgabe von Attribut werden erweitert werden, z.B.:

```
<xsl:template match="/|*">  
  <xsl:apply-templates select="@*" />  
  <xsl:apply-templates />  
</xsl:template>
```



- Bei Stylesheet ohne Templates sind nur die beiden vordefinierten Templates aktiv:

```
<xsl:template match="*|/">  
  <xsl:apply-templates/>  
</xsl:template>
```

```
<xsl:template match="text()|@"*>  
  <xsl:value-of select="."/>  
</xsl:template>
```

- Gesamtes Ursprungsdokument wird traversiert, dabei werden PCDATA extrahiert

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template
  match="text()|@"*>
  <xsl:value-of select="."/>
</xsl:template>
```

```
<?xml version="1.0"?>
<name>
  <first>
    John
  </first>
  <middle>
    Fitzgerald Johansen
  </middle>
  <last>
    Doe
  </last>
</name>
```

match="/" ⇒ apply-templates

match="\*" ⇒ apply-templates

match="\*" ⇒ apply-templates

match="text()" ⇒ John

match="\*" ⇒ apply-templates

match="text()" ⇒ Fitzgerald Johansen

match="\*" ⇒ apply-templates

match="text()" ⇒ Doe

- **Stylesheet mit lediglich einem Template:**

- wird auf jedes Element ("**\***") angewandt
- kopiert Wurzel des aktuellen Teilbaumes
- wendet rekursiv alle Templates an

```
<xsl:template match="*" >  
  <xsl:copy >  
    <xsl:apply-templates/>  
  </xsl:copy >  
</xsl:template >
```

- überschreibt 1. vordefinierte Template `<xsl:template match="*|/">`, da spezieller
- Zusammen mit 2. vordefinierten Template `<xsl:template match="text()|@*">` wird Ursprungsdokument (fast) kopiert

# Position des rekursiven Aufrufes?

```
<xsl:template match="*">
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

```
<xsl:template match="*">
  <xsl:copy>
  </xsl:copy>
  <xsl:apply-templates/>
</xsl:template>
```

Ergebnis:

```
<root>
  <a>aaa</a>
  <b>bbb</b>
  <c>ccc</c>
</root>
```

```
<root>
  <a>aaa</a>
  <b>bbb</b>
  <c>ccc</c>
</root>
```

```
<root/>
<a/>
aaa
<b/>
bbb
<c/>
ccc
```

```
<xsl:template match="comment()|/processing-instruction()"/>
```

- vordefiniertes Template
- Kommentare und Prozessanweisungen werden nicht übernommen
- Beispiel für Template, wenn Kommentare im Ergebnisdokument erscheinen sollen

```
<xsl:template match="comment()">  
  <xsl:comment>  
    <xsl:value-of select="."/>  
  <xsl:/comment>  
</xsl:template>
```



## **XSLT – Templates: benannte Templates, Variablen & Parameter**

- Templates können auch einen Namen haben:

```
<xsl:template match="/order/item" name="order-template" >  
  ...  
</xsl:template >
```

- Benannte Templates können gezielt mit  
    **<xsl:call-template name="order-template" />**  
angewandt werden

# Template-Modi

- Attribute mode - um Templates, die dasselbe match-Kriterium verwenden, unterscheiden zu können

```

<xsl:stylesheet ...>
  <xsl:template match="/" >
    ...
    <xsl:apply-templates select="//buch" mode="kurzfassung" >
    <xsl:apply-templates select="//buch" mode="langfassung" >
  </xsl:template>

```

Verwendung der Templates entsprechend des **mode**-Attributes

```

<xsl:template match="buch" mode="kurzfassung" >
  ...
</xsl:template>
<xsl:template match="buch" mode="langfassung" >
  ...
</xsl:template>
</xsl:stylesheet ...>

```

Definition der Templates mit **mode**-Attribute



- werden z.B. verwendet um Wiederholungen gleicher Ausdrücke zu vermeiden
- Initiale Zuweisung kann nicht überschrieben werden!
- Beispiel:  
deklariert Variable X mit  $X :=$  aktueller Teilbaum

```
<xsl:variable name="X">  
  <xsl:copy-of select=".">  
</xsl:variable>
```

- Wert von X: \$X
- Beispiel:

```
<xsl:variable name="N">2</xsl:variable>  
...  
<xsl:value-of select="item[position()=$N]"/>
```

- Variablen kommen innerhalb von `<xsl:stylesheet>` vor und dann entweder
  - außerhalb von `<xsl:template>` (d.h. auf dem Top-Level)
  - globale Variable - steht allen Templates zur Verfügung
- oder
  - innerhalb von `<xsl:template>`
  - lokale Variable - gültig nur innerhalb des Templates, in dem sie notiert wurde

- Templates können Parameter haben:

```
<xsl:template name="printRows" >
```

```
<xsl:param name="N"/>
```

...

```
<xsl:call-template name="printRows" >
```

```
<xsl:with-param name="N" select="$N + 1"/>
```

```
</xsl:call-template >
```

```
</xsl:template >
```

Aufruf des  
Parameters

Festlegung/Überschreibung  
des Parameters



**XSLT: und was gibt es noch?**

- das bedingte Template als Kindelement von <xsl:if>

```
<xsl:template match="kurs">  
  <xsl:if test="referent='Harasic'">  
    <h3><xsl:value-of select="@name"/></h3>  
    <p>Referent: <xsl:value-of select="referent"/></p>  
  </xsl:if>  
</xsl:template>
```

Template

- Wenn es sich bei der Bedingung um einen XPath-Ausdruck handelt
  - bei einem Knotenset "true", wenn das Knotenset mindestens einen Knoten enthält
  - bei einem String "true", wenn der String nicht leer ist
  - bei einer Nummer "true", wenn diese ungleich Null ist

# Beispiel für <xsl:if>

## Template

```
<xsl:template match="kurs">
  <xsl:if test="referent='Harasic'">
    <h3><xsl:value-of select="@name"/></h3>
    <p>Referent: <xsl:value-of select="referent"/></p>
  </xsl:if>
</xsl:template>
```

## Ausschnitt aus dem Quelldokument

```
<kursprogramm>
  <kurs name="XML Technologien">
    <referent>Harasic</referent>
  </kurs>
  <kurs name="Datenbanken">
    <referent>Bodin</referent>
  </kurs>
</kursprogramm>
```

## Ausgabe

**XML-Technologien**

Harasic



# Schleifen <xsl:for-each>

- Anweisungen als Kinderknoten von <xsl:for-each>

Template

```

...
<table width="..." border="1" cellspacing="1" ...
...
<xsl:template name="...">
  <xsl:for-each select="//kurs">
    <tr>
      <td width="..." height="...">
        <xsl:value-of select="position()">
      </td>
      <td width="..." height="...">
        <xsl:value-of select="@name">
      </td>
    </tr>
  </xsl:for-each>
</xsl:template>
...

```

```

<kursprogramm>
  <kurs name="XML Technologien">
    <referent>Harasic</referent>
  </kurs>
  <kurs name="Datenbanken">
    <referent>Bodin</referent>
  </kurs>
</kursprogramm>

```

Ausgabe

1	XML Technologien
2	Datenbanken

# Sonstige Möglichkeiten

- Sortieren

```
<xsl:sort select="name/nachname"/>  
<xsl:sort select="name/vorname"/>
```

- XPath-Funktionen

```
<xsl:if test="not(position()=last())">...</xsl:if>
```

- Mehrere Ursprungsdokumente

```
<xsl:apply-templates select="document('bib.xml)'"
```

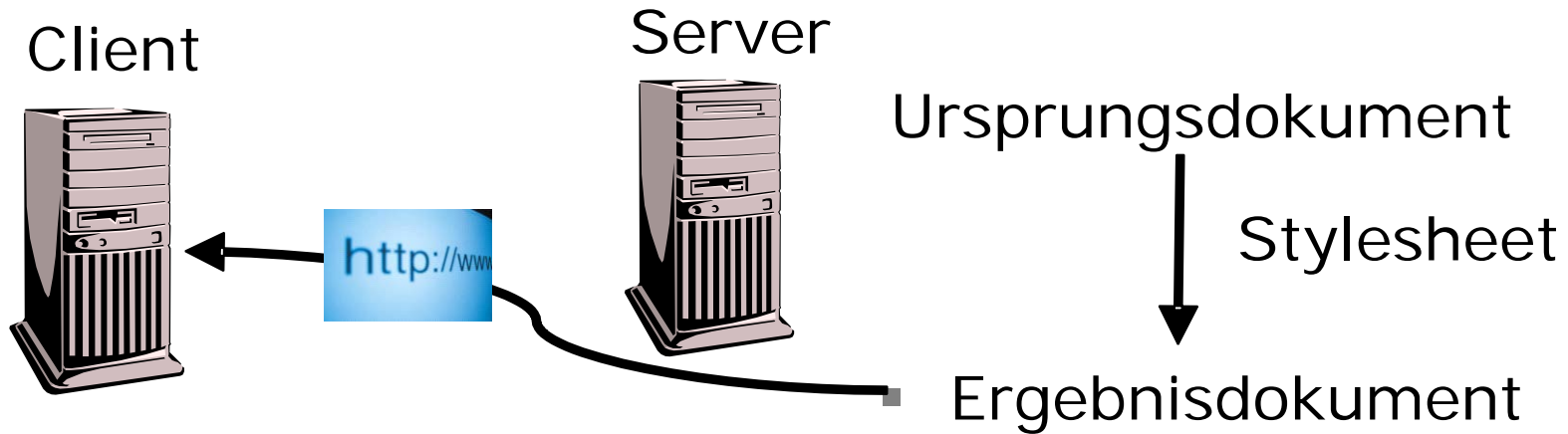
- ... und vieles mehr!



- Variablen machen Stylesheets zu einem mächtigen Termersetzungssystem mit unbeschränkten Registern
- [www.unidex.com/turing](http://www.unidex.com/turing) definiert universelle Turingmaschine als XSLT-Stylesheet
  - Eingabe: Programm  $p$  (XML), Input  $i$  (XML)
  - Ausgabe:  $p(i)$
- Browser = vollwertiger Computer!
- Stylesheets tatsächlich berechnungsvollständig und damit vollwertige Programmiersprache (Kepser 2002)
  - <http://www.unidex.com/turing/utm.htm>
- Terminierung von Stylesheets kann prinzipiell nicht garantiert werden

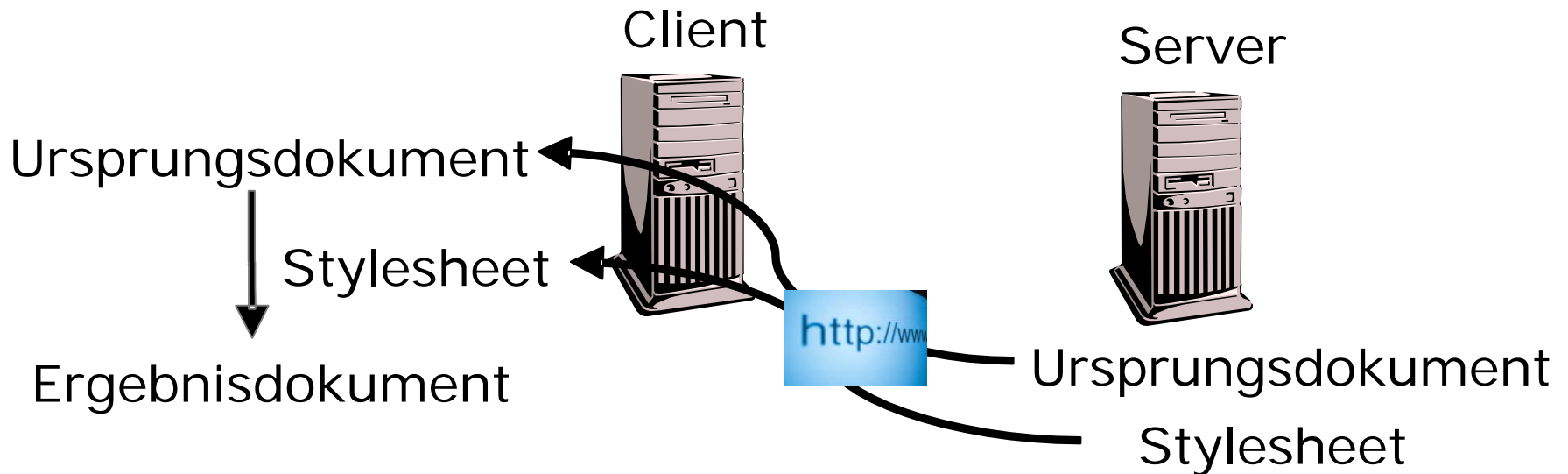
- Stylesheets können auf zwei Arten verarbeitet werden:
  - auf dem Server
    - Ursprungsdokument verdeckt
    - alle Transformationen auf zentralen Server
  - im Client
    - Transformationen auf Clients verteilt: spart Server-Ressourcen
    - Ursprungsdokument sichtbar

# 1. Verarbeitung auf dem Server



- Server wendet passendes Stylesheet auf Ursprungsdokument an
- z.B. mit MSXML:  
`msxsl source stylesheet.xsl -o output`
- Client bekommt nur Ergebnisdokument

## 2. Verarbeitung im Client



- Client bekommt Ursprungsdokument & passendes Stylesheet
- im Ursprungsdokument:  
`<?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>`
- Web-Browser wendet Stylesheet automatisch an und stellt Ergebnisdokument dar

# Wo Stylesheets verarbeiten?

## Verarbeitung im Client

- + Transformationen auf Clients verteilt: spart Server-Ressourcen
- Ursprungsdokument sichtbar

XSLT: stellt sicher, dass Transformation im Web-Client ausgeführt werden kann.

---

## Verarbeitung auf dem Server

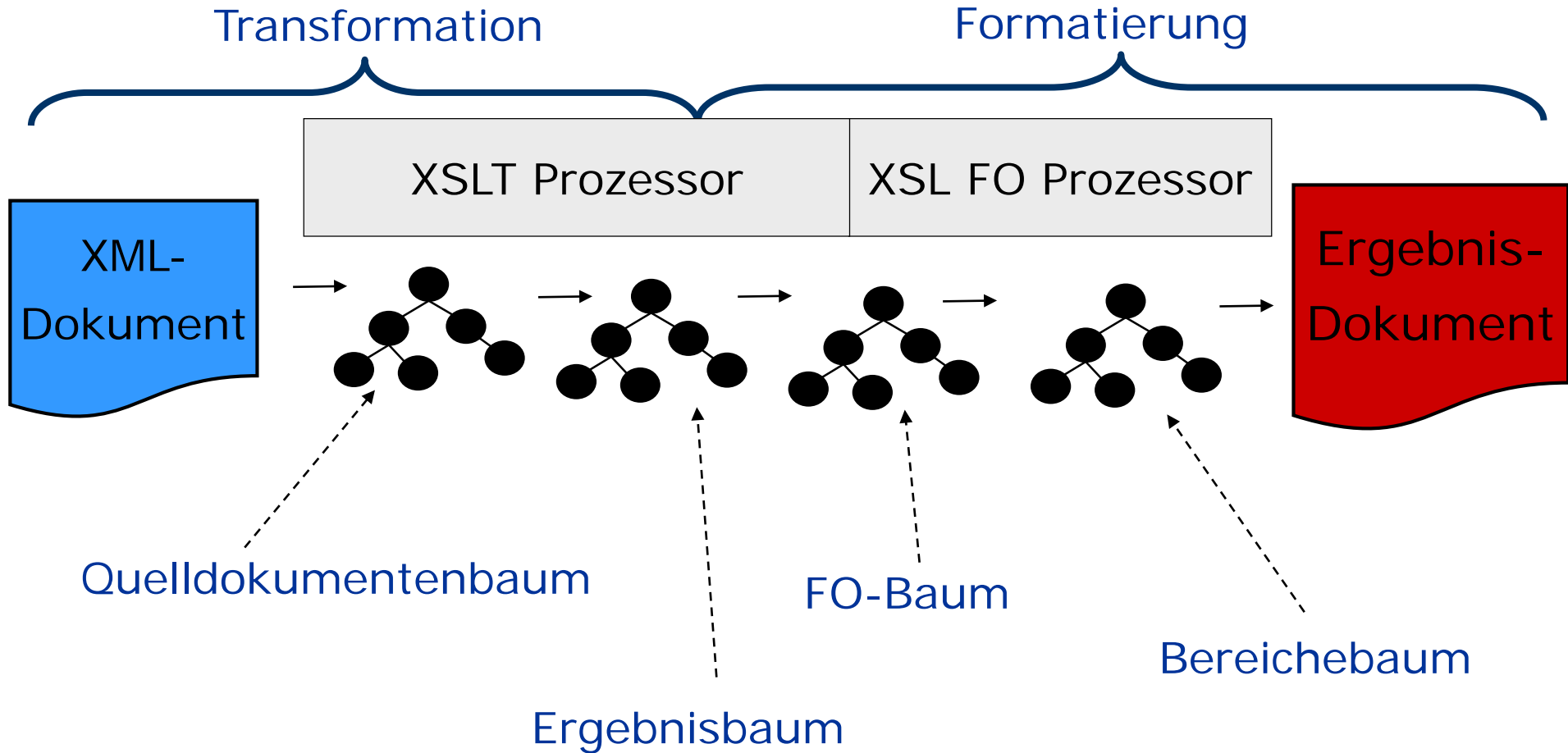
- + Ursprungsdokument verdeckt
- alle Transformationen auf zentralen Server

XSLT: nicht unbedingt nötig, da Transformation auf eigenem Server durchgeführt wird.



## **eXtensible Stylesheet Language Formatting Objects (XSL-FO)**

- XSLT
  - erlaubt Transformation von XML → HTML
  - ungeeignet für druckfähige Formatierungen (PDF, RTF)
- XSL-FO
  - erlaubt XML-Dokumente mit druckfähigen Layout zu versehen
  - Transformation XML → PDF oder RTF möglich
  - basiert auf auf Cascading Style Sheets (CSS2)
  - W3C-Standard von 2001





<b>CSS</b>	<b>XSL-FO</b>
Darstellung auf <b>Bildschirm</b>	Darstellung auf <b>seitenorientiertem</b> Ausgabemedium
Ausgabe durch <b>Webbrowser</b>	Ausgabe durch <b>Drucker</b> und andere <b>Seitenausgabegeräte</b>
Formatierungsinformation für vorhandenes <b>Markup</b>	Komplette Ersetzung von Markup durch - <b>Formatierungsmarkup</b>

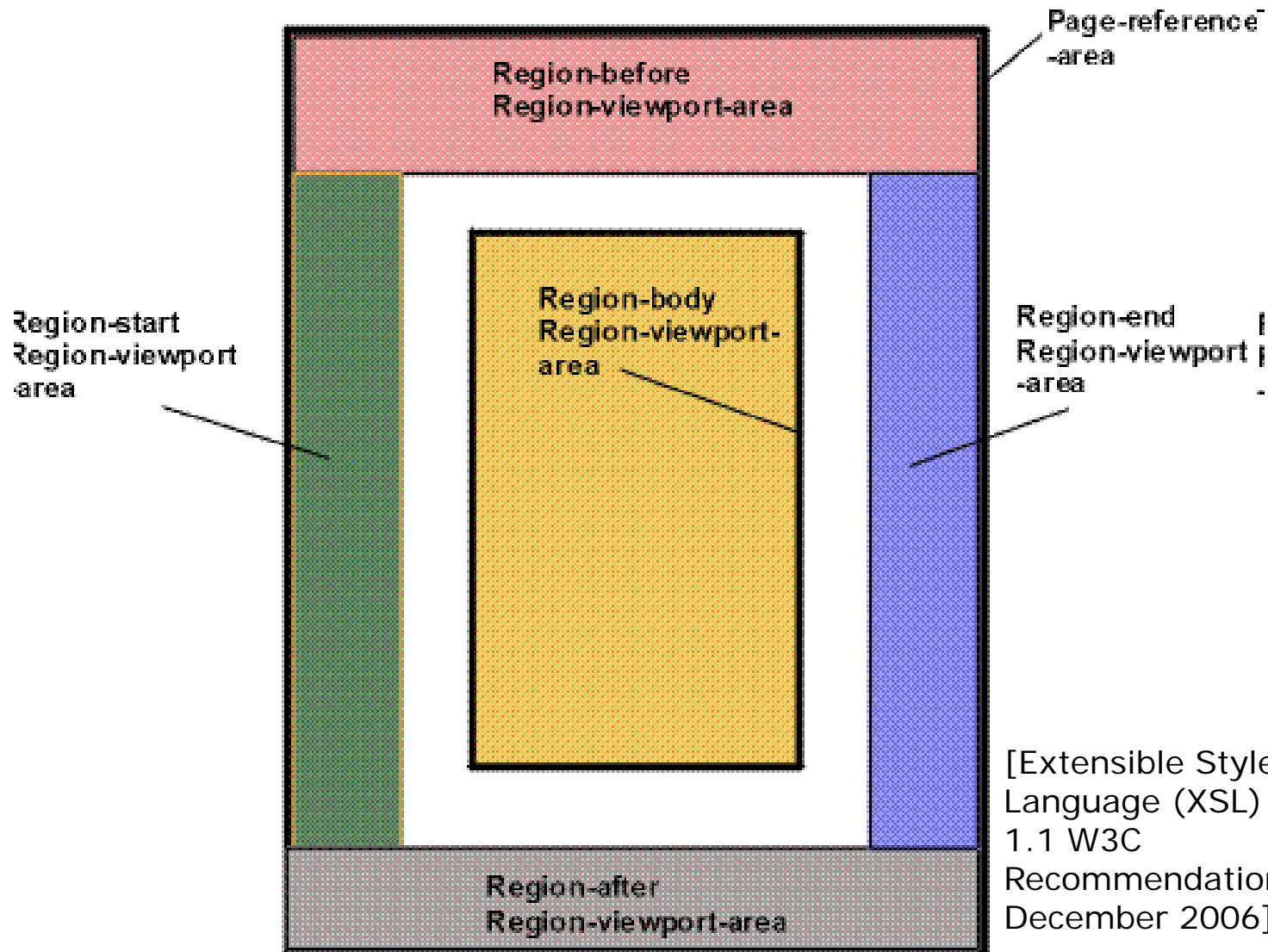
- Massensatz, z.B. bei der technischen Dokumentation
- gleichzeitige Ausgabe derselben Inhalte in unterschiedlichen Formaten:
  - verschiedene Medien
  - gleiches Medium aber verschiedene Bedürfnisse der Nutzer
- Individualisierung bzw. Personalisierung von Dokumenten

- Wie in Office-Programmen Vorlagen für Seitenstruktur:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="my-page">
      <fo:region-body margin="1in"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="my-page">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Hello, world!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Quellenhinweis: XSL-FO Beispiele auf den folgenden Folien aus Nikolai Grigoriev. XSL Formatting Objects Tutorial. <http://www.renderx.com/tutorial.html>



- CSS artige Darstellungseigenschaften:

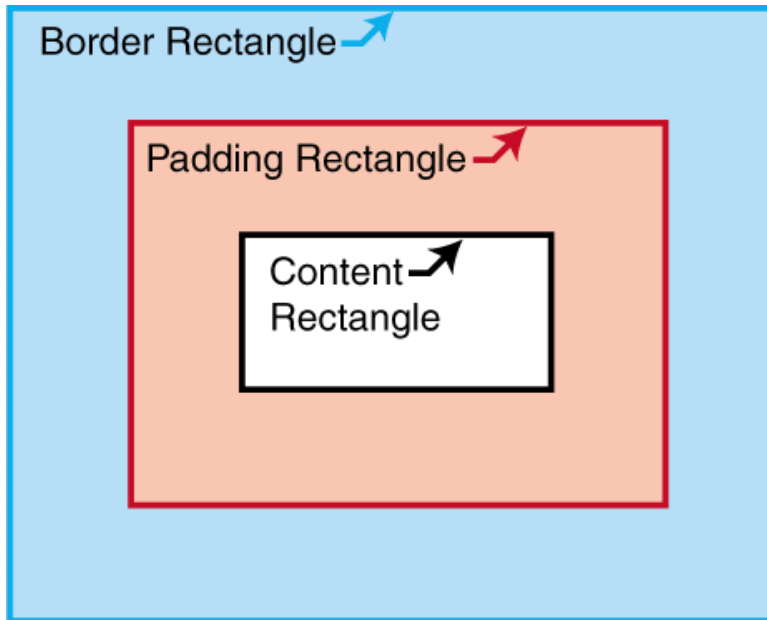
```
<fo:block font="italic 14pt Times" >  
  <fo:inline color="red" >H</fo:inline>ello,  
  <fo:inline font-weight="bold" >world!</fo:inline>  
</fo:block>
```

- Für Blöcke:

```
<fo:block text-align="justify" text-indent="1in"  
  text-align-last="end" last-line-end-indent="1in" >
```

This is an example of double-justified text with an indented first line. The last line of the text is aligned to the right, and indented by 1 inch from the right.

```
</fo:block>
```



[Extensible Stylesheet Language (XSL) Version 1.1  
W3C Recommendation 05 December 2006]

```
<fo:block border="thin solid navy"  
  text-align="center"  
  padding-before="18pt" padding-bottom="18pt" >  
<fo:block border="thin solid maroon" >  
  The outer block has a 18 pt padding from top and bottom  
</fo:block >  
</fo:block >
```