Freie Universität Berlin

# XSLT

Malgorzata Mochol
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mochol@inf.fu-berlin.de

# Wie geht es weiter?

Freie Universität Berlin

letzte Woche

☑ Navigation & Verknüpfungen mit XPath & Co.

heutige Vorlesung

- XSLT

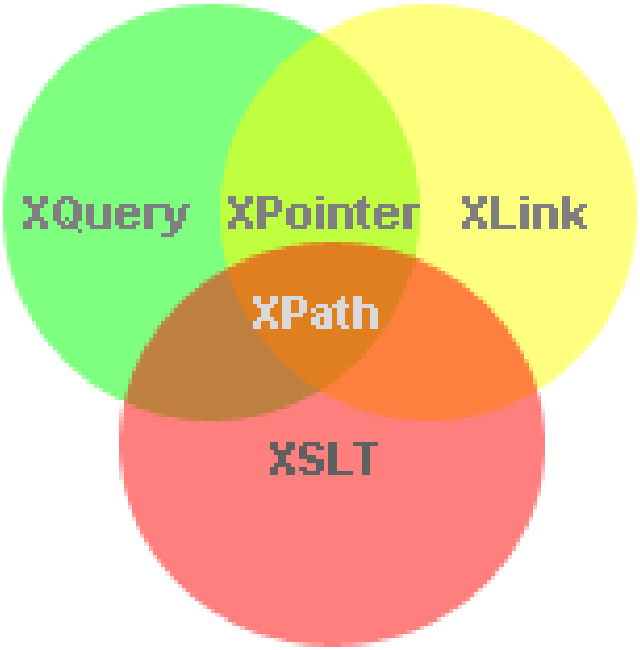



Bild-Quelle: http://www.w3schools.com/xquery/default.asp

# News

- 5.Übung
  - heute (Mi.) 14:15 – 15:45, SR 005 und
  - Do. 10:15-11:45, SR 005

  - Thema → XSLT

- 4.Übungsblatt: XSLT
  - heute Nachmittag online

# eXtensible Stylesheet Language (XSL)

# XSL

- 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-/Selektionssprache für XML-Dokumente
  - XSLT – Transformationssprache für XML-Dokumente
  - XSL-FO – Formatierungssprache für XML-Dokumente

# XML Transformation

# Warum XML transformieren?

Freie Universität Berlin

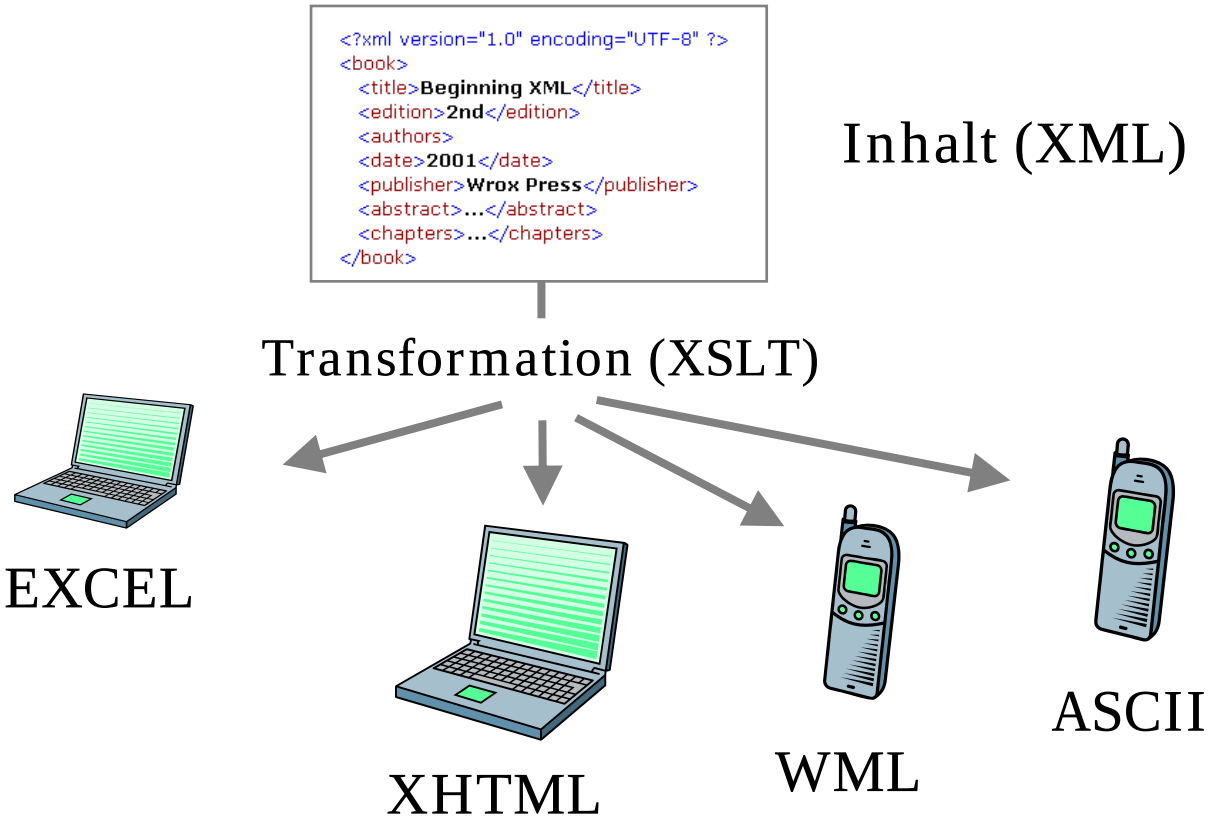
## Trennung Inhalt und Präsentation

- 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



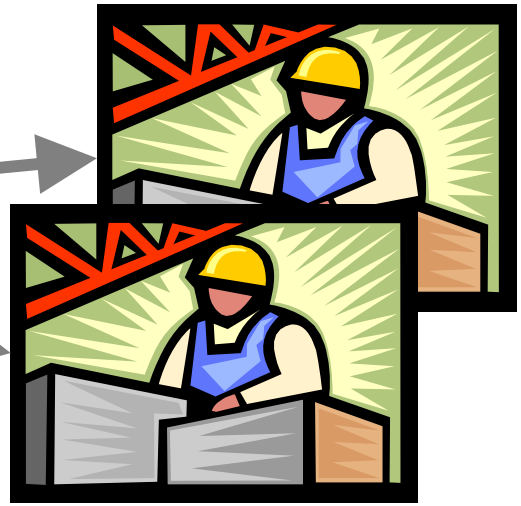

- Multi-Delivery: unterschiedliches Layout von Inhalten
- Beachte: XHTML, WML $\subset$ XML

# XML-Inhalt ➔ XML-Inhalt

Freie Universität Berlin

Großhandel

Zulieferer



*interner Kundenauftrag*

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

anpassen

anpassen

übernehmen

*externer Zulieferauftrag*

- Datum
- Produktbezeichnung aus Zuliefererkatalog
- Anzahl
- Auftraggeber

# XML-Inhalt ➔ XML-Inhalt

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</c
</order>
```

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>
```

andere Sicht (view)
auf XML-Inhalt

Freie Universität Berlin

# eXtensible Stylesheet Language Transformation (XSLT) - Einführung

# Was ist XSLT?

Freie Universität Berlin

- in XML beschriebene Sprache zur Transformation von XML-Dokumenten
- eine beschreibende Sprache
- 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

# XSLT: Ziele

Freie Universität Berlin

## Verkopplung zwei Prozesse:

- Transformation des Quelldokuments in das Ergebnisdokument
- Formatierung für die Ausgabe in dem gewünschten Format

# XSLT: Anfragesprache für XML?

Freie Universität Berlin

## SQL

- Anfrage = Sicht (View) auf Menge von Relationen
- abgeschlossen: SQL-Anfrage liefert immer eine Relation

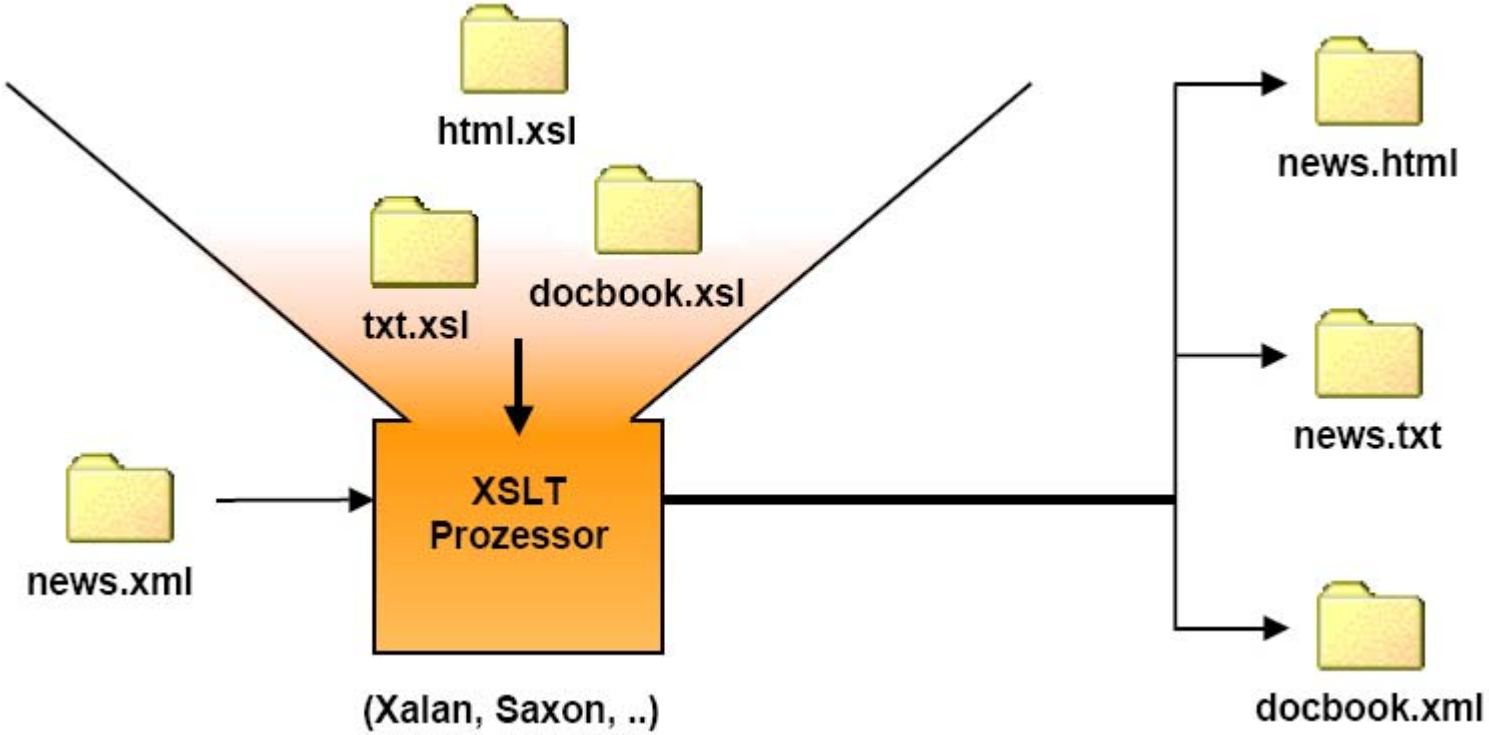
## XSLT

- Transformation = Sicht (View) auf Menge von XML-Dokumenten

⇨ Anfragesprache für XML

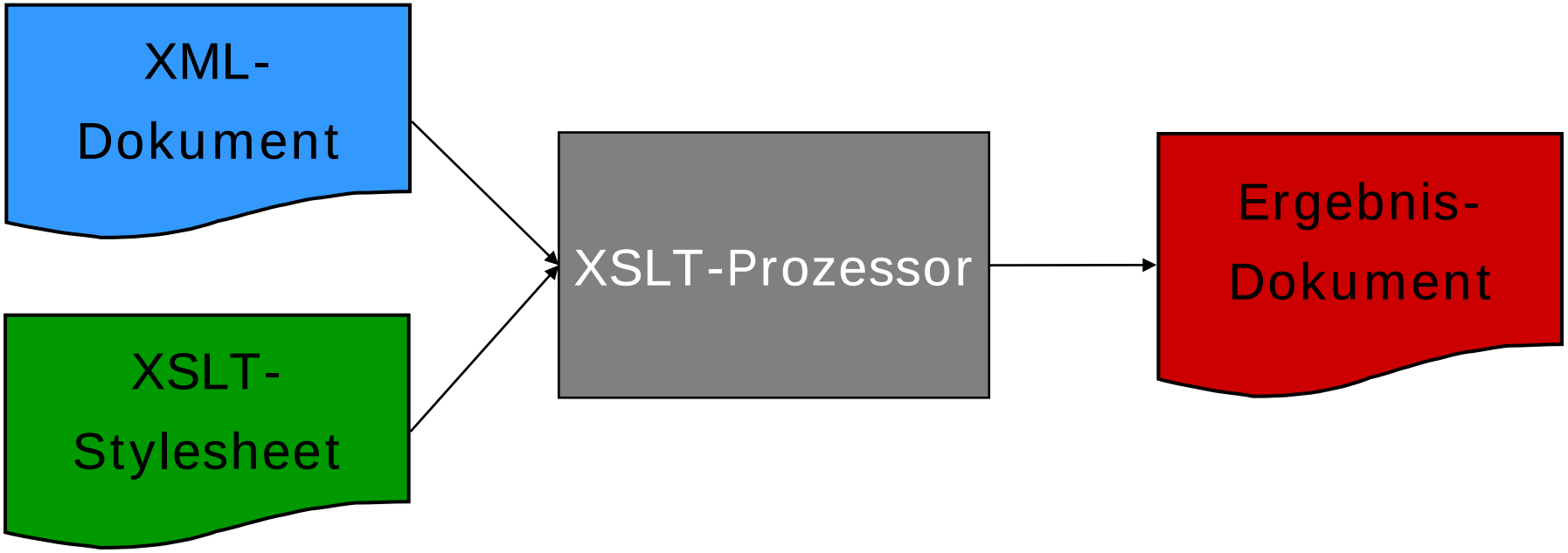
- nicht abgeschlossen: kann beliebige Textformate liefern, nicht nur wohlgeformtes XML

# Was passiert?




Quelle: http://www.oio.de/m/konf/jaxw2004/jaxw2004-XSLT-2.0.pdf, Angepasst

# Allg. Schema der Transformation




- 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

# XSLT Prozessoren

- 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/
  - Saxon in Version 9.0.0.5 unterstützt XSLT 2.0, XQuery 1.0, & XPath 2.0

# Programmierparadigma

- XSLT-Programm (stylesheet) = Menge von Transformationsregeln
- Transformationsregel (template)
  - Erzeuge aus Unterstruktur X im Ursprungsdokument 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

# Ursprungs- und Ergebnisdokument

```
<?xml version="1.0"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>...</date>
  <customer>Sally
   Finkelstein</customer>
</order>
```

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

Template

```
<p>Production-Class Widget</p>
```

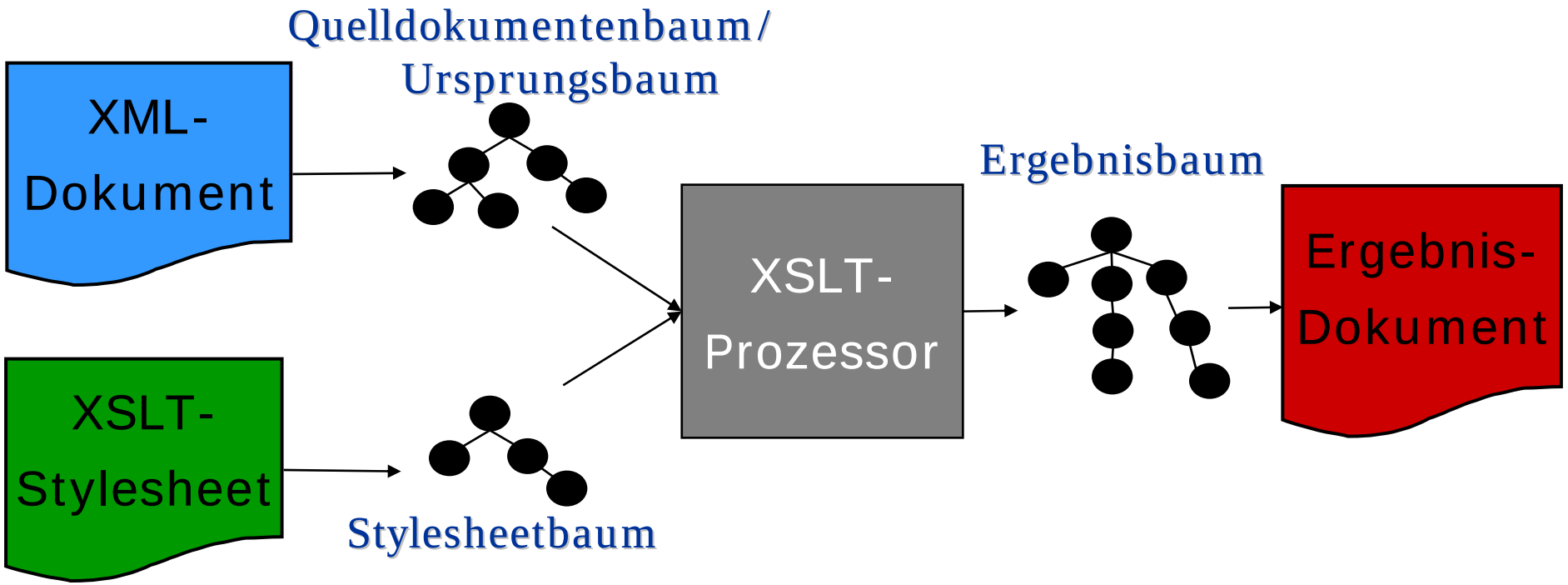
Ursprungsdokument → Ursprungsbaum (source document → source tree)

Transformation

Ergebnisbaum → Ergebnisdokument (result tree → result document)

# Schema der Transformation im Detail

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



## XSLT-Transformationsregeln

- immer auf Ursprungsdokument(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
  - ⇨ oft auch funktionales Programmierparadigma genannt

# Grundstruktur von Stylesheets

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

XML-Deklaration

XSLT-Namensraum

Präfix

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

# Top-Level-Elemente (I)

Freie Universität Berlin

- Kinder des Elements <xsl:stylesheet>
- beliebige Reihenfolge, nur <xsl:import> immer am Anfang
- <xsl:import> & <xsl:include> - importieren Stylesheets

```
<xsl:stylesheet>
  <xsl:import/>
  <xsl:include/>
  <xsl:attribute-set/>
  <xsl:output/>
  <xsl:variable/>
  <xsl:param/>
  <xsl:template/>
  ...
</xsl:stylesheet>
```

- <xsl:attribute-set> - defineirt einen Set, der aus einer Sammlung von Attributen besteht, die mit Hilfe von <xsl:attribute> festgelegt werden

# Top-Level-Elemente (II)

Freie Universität Berlin

- <xsl:output> - legt eine Methode fest, die bei der Erzeugung des Ergebnisdokuments beachtet werden soll

```
<xsl:stylesheet>
  <xsl:import/>
  <xsl:include/>
  <xsl:attribute-set/>
  <xsl:output/>
  <xsl:variable/>
  <xsl:param/>
  <xsl:template/>
    ...
</xsl:stylesheet>
```

- <xsl:output method="xml"/> - erzeugt wohlgeformtes XML
- <xsl:output method="html"/> - HTML-Elemnte & -Attribute werden erkannt
- <xsl:output method="text"/> - gibt die String-Werte aller Textknoten, die im Ausgabebaum enthalten sind

# XSLT - Templates

# Grundstruktur von Stylesheets

```
<?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>
```

Suchmuster

Template
(Template-Regeln)

# Inhalte erzeugen

Freie Universität Berlin

- Template: „Suche im Ursprungsdokument Unterstruktur X und erzeuge hieraus Ergebnisdokument Y!“

- zwei Möglichkeiten, Y zu erzeugen:
  1. neue Inhalte erzeugen
  2. Inhalte von X nach Y übertragen.

- beide Möglichkeiten beliebig miteinander kombinierbar

# 1. Neue Inhalte erzeugen (I)

Freie Universität Berlin

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

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

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

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

# 1. Neue Inhalte erzeugen (II)

Freie Universität Berlin

- statt üblicher XML-Syntax

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

- auch möglich:

```
<xsl:template match="...">
   <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

Freie Universität Berlin

**<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

# Beispiel

```
<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>
```

## Ergebnisdokument

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

```
<DIV>
   <p/>
</DIV>
```

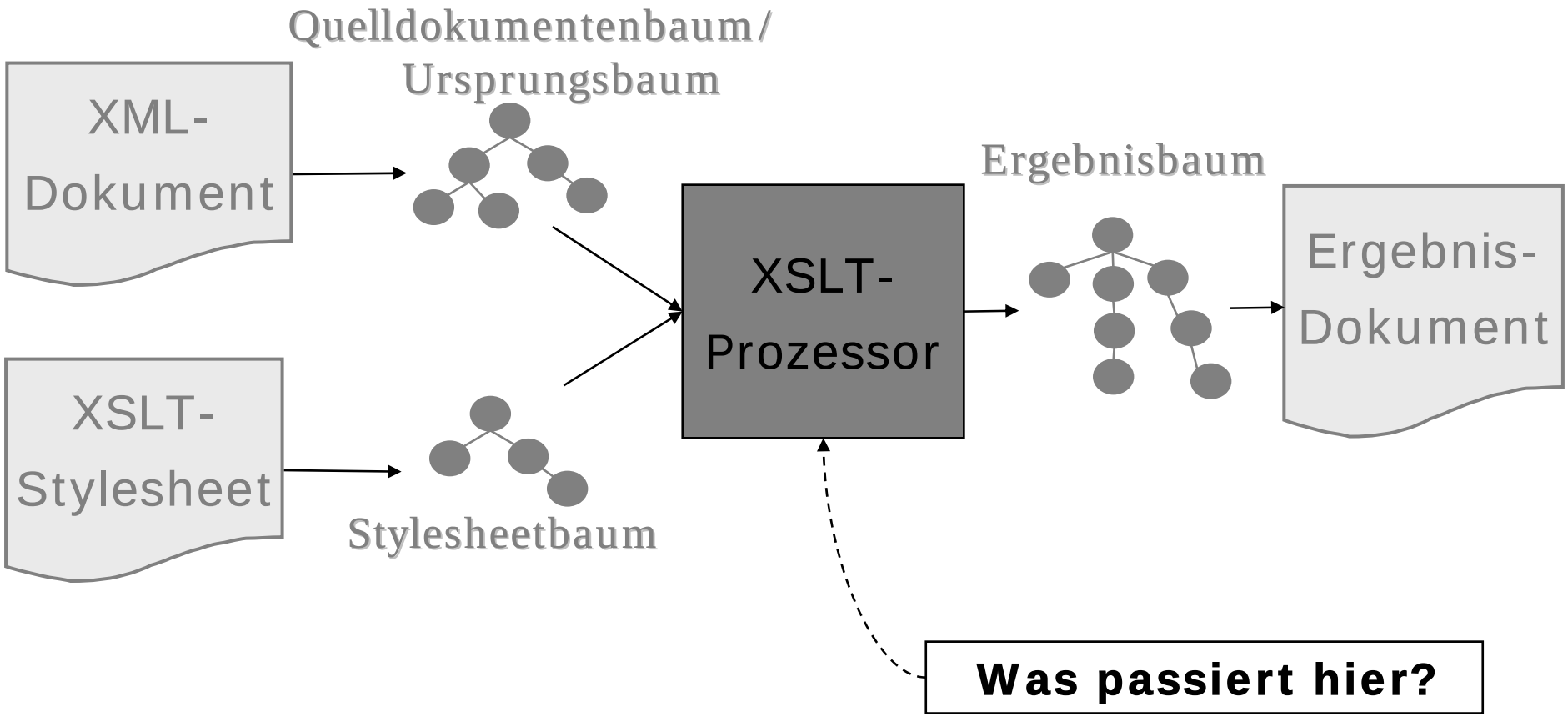
```
<DIV>
   Compare these constructs.
</DIV>
```

oder mit "text()" statt "."

```
<DIV>
   Compare
</DIV>
```

# XSLT-Prozessor im Transformationsprozess

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



# Funktionsweise des XSLT-Prozessors

Freie Universität Berlin

1. K := Dokument-Wurzel ("/") des Ursprungsdokumentes
2. Identifiziere alle Templates, die auf K anwendbar sind.
   a) Ist genau ein Template anwendbar, dann wende dieses an.

      Fertig.
   a) Sind mehre Templates anwendbar, dann wende das speziellste an:

      z.B. ist "/order" spezieller als "/*".

      Fertig.
   c) Ist kein Template anwendbar, dann wiederhole für alle Kinder K' von K Schritt 2 mit K := K'.

# Template-Konflikte

Freie Universität Berlin

- 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 Wildecards
  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

Freie Universität Berlin

**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

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

**Ausgabe**

a1

a2

# Templates mit Rekursion

Freie Universität Berlin

**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
```

# Rekursiver Aufruf aller Templates

## <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>
```

```
<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>
```

→

```
b1
b2
b3
b4
c1
b5
```

- **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

Freie Universität Berlin

## 1.vordefinierte Template

- realisiert rekursiven Aufruf des Prozessors, wenn kein Template anwendbar ist

## 2.vordefinierte Template

- kopiert PCDATA und Attribut-Werte 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. vordefinierte 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 rekursiven Aufruf des XSLT-Prozessors
- wird von jedem speziellerem Template überschrieben: z.B. sind "/" und "item" spezieller als "*|/"
- spezielleres Template anwendbar ⇨ kein automatischer rekursiver Aufruf

# 2. vordefinierte Template

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

- Template wird auf PCDATA text() und Attribute @* angewandt
- text(): XPath-Funktion, selektiert PCDATA
- Template überträgt PCDATA bzw. Attribut-Wert in das Ergebnisdokument

# Leeres Stylesheet

- 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 wird PCDATA und Attribut-Werte extrahiert

# Beispiel

```
<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

# Identitäts-Stylesheet

Freie Universität Berlin

- Stylesheet mit lediglich einem Template:

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

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

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

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

Ergebnis:

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

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

# Templates für Kommentare

```
<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

# Benannte Templates

- 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"/>**

  aufgerufen werden.

# Template-Modi

- Attribute **mode** - um Templates, die das selbe 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>

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

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

Definition der Templates mit **mode**-Attribute

# Variablen

- verwendet z.B.: um Wiederholungen gleicher Ausdrucke zu vermeiden
- Beispiel: deklariert Variable X mit X := aktuellen Teilbaum

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

- Initiale Zuweisung kann nicht überschrieben werden!
- Wert von X: $X
- Beispiel:

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

# Gültigkeit von Variablen

Freie Universität Berlin

- Variablen kommen innerhalb von <xsl:stylesheet> vor und dann entweder

- außerhalb von <xsl:tempale> (d.h. auf dem Top-Level)

  → **globale** Variable - steht allen Templates zur Verfügung

oder

- innerhalb von <xsl:tempale>

  → **lokale** Variable - gültig nur innerhalb des Templates, in dem sie notiert wurde

# Parameter

- 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‘s noch?

# Nummerierung <xsl:number>

Freie Universität Berlin

- zahlreiche Formate für Nummerierung über <xsl:number>

Template

```
<xsl:template match="kurs">
  <p>
    <xsl:number/>
    <xsl:text> </xsl:text>
    <xsl:value-of select ="@name"/>
  </p>
</xsl:template>
```

Ausschnitt aus dem Quelldokument

```
<kursprogramm>
  <kurs name="XML Technologien">
    ...
  </kurs>
  <kurs name="Datenbanken">
    ...
  </kurs>
</kursprogramm>
```

- mit **<xsl:number format="i"/>**

**1** XML-Technologien
**2** Datenbanken

**i** XML-Technologien
**ii** Datenbanken

Ausgabe

# Bedingte Ausführung <xsl:if>

Freie Universität Berlin

- das bedingte Template als Kindelement von <xsl:if>
- if-else Konstrukt NICHT vorhanden

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

# Beispiel für <xsl:if>

Freie Universität Berlin

Template

```
<xsl:template match="kurs">
  <xsl:if test=referent='Mochol'>
    <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>Mochol</referent>
  </kurs>
  <kurs name="Datenbanken">
   <referent>Bodin</referent>
  </kurs>
</kursprogramm>
```

Ausgabe

**XML-Technologien**

Mochol

# Wahlmöglichkeiten <xsl:choose>

Beispiel

```
<xsl:template match="item">
  <part-number>
    <xsl:choose>
      <xsl:when test=". = 'Production-Class Widget'">
        E16-25A
      </xsl:when>
      <xsl:when test=". = 'Economy-Class Widget'">
        E16-25B
      </xsl:when>
      <xsl:otherwise>00</xsl:otherwise>
    </xsl:choose>
  </part-number>
</xsl:template>
```

Falls Inhalt von item = 'Production-Class Widget', dann erzeuge E16-25A

- Switch-Anweisung in Java ähnlich
- Abarbeitung von oben nach unten

# 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>Mochol</referent>
  </kurs>
  <kurs name="Datenbanken">
   <referent>Bodin</referent>
  </kurs>
</kursprogramm>
```

Ausgabe

| | |
|---|---|
| 1 | XML Technologien |
| 2 | Datenbanken |

# Sonstige Möglichkeiten

Freie Universität Berlin

- **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!

# XSLT: Verarbeitung von Stylesheets

# Verarbeitung von Stylesheets

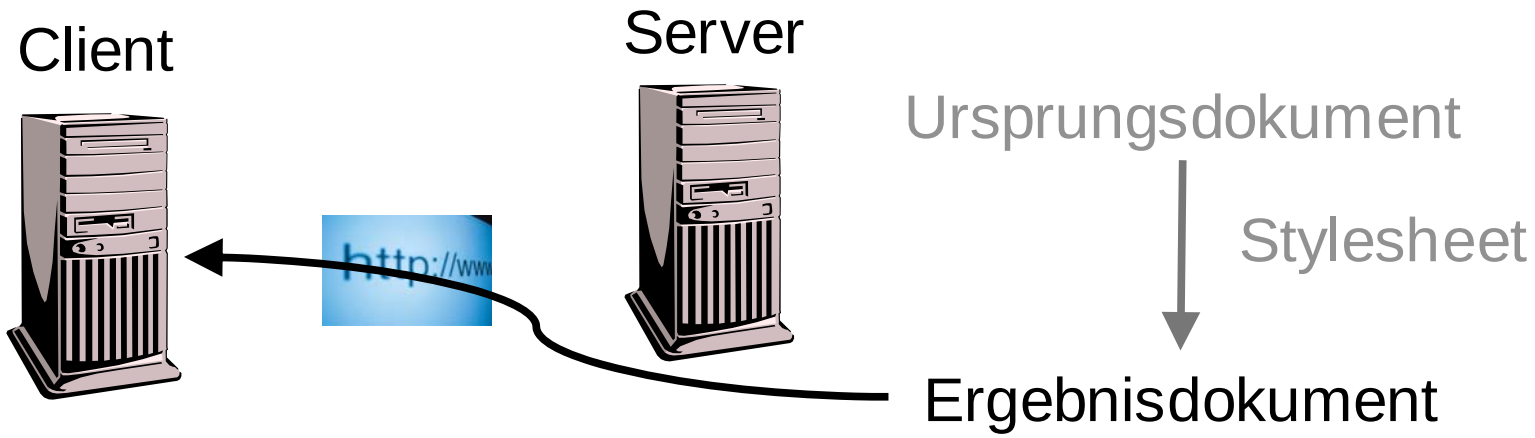
Freie Universität Berlin

Stylesheets können auf zwei Arten verarbeitet werden:

1. auf dem Server
2. im Client

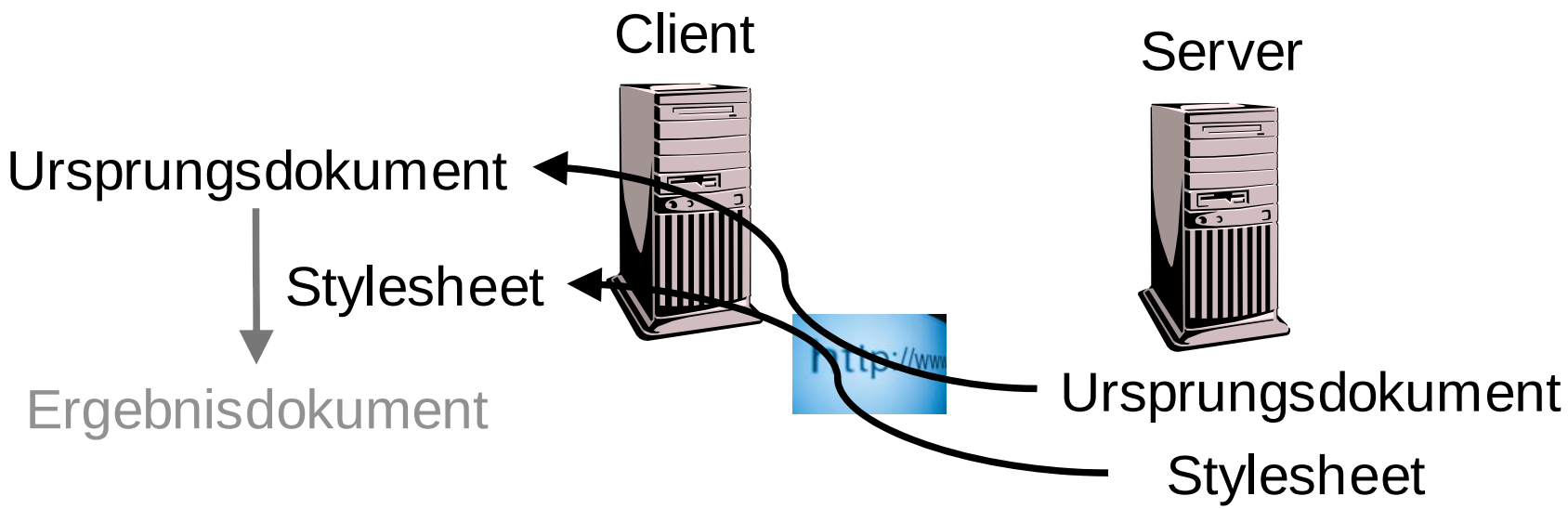
- Worin besteht der Unterschied?
- jeweiligen Vor- und Nachteile

# 1. Verarbeitung auf dem Server




- Server wendet passendes Stylesheet auf Ursprungs-dokument an.
- z.B. mit MSXML: msxsl *source stylesheet.xsl* –o *output*
- Client bekommt nur Ergebnisdokument

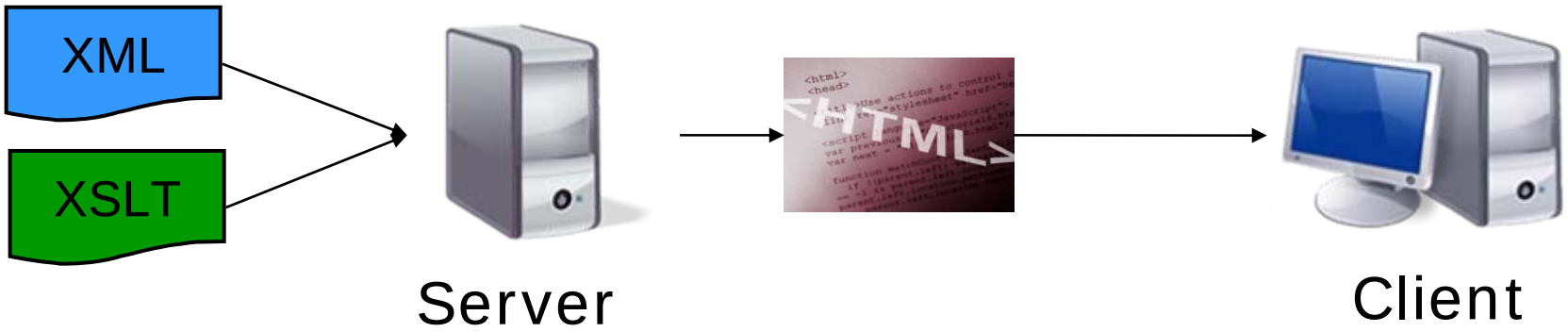
# 2. Verarbeitung im Client

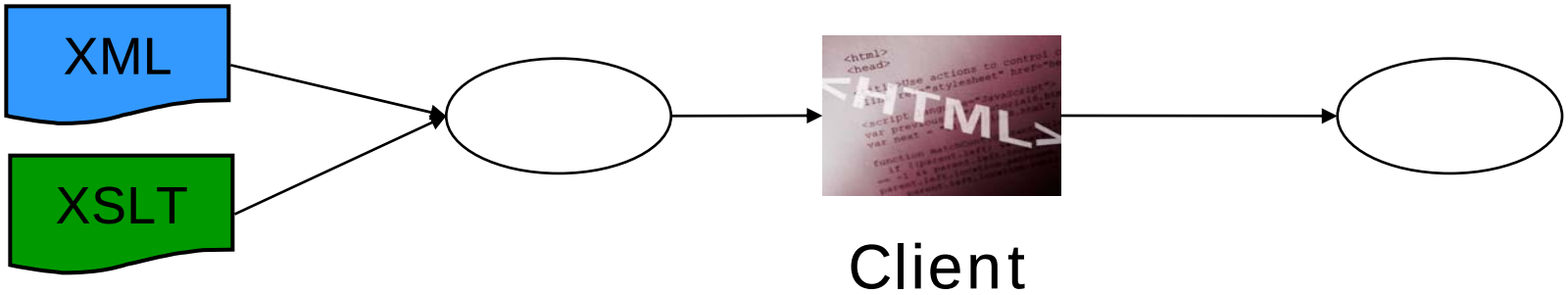



- Client bekommt Ursprungsdokument & passendes Stylesheet.
- im Ursprungsdokument:

  <?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>
- Web-Browser wendet Stylesheet automatisch an und stellt Ergebnisdokument dar.

# Alternativen bei der Transformation in HTML




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

# Wo Stylesheets verarbeiten?

Freie Universität Berlin

## 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.

# Mächtigkeit von XSLT

Freie Universität Berlin

- **Variablen** machen Stylesheets zu einem mächtigen Termersetzungssystem mit unbeschränkten Registern.
- www.unidex.com/turing definiert universelle Turingmaschine als XSLT-Stylesheet
  - Eingabe: Programm p (XML), Input i (XML)
  - Ausgabe: p(i)

⇨ Browser = vollwertigen Computer!

- Stylesheets tatsächlich **berechnungsvollständig** und damit **vollwertige Programmiersprache** (Kepser 2002)

➔ Terminierung von Stylesheets kann prinzipiell nicht garantiert werden.

# Missbrauch von XSLT

Incremental Development, Inc.

*XSL example, XSLT example, XSL sample, XSLT sample, XSL tutorial, XSLT tutorial, stupid XSL trick, stupid XSLT trick.*

## Gallery of Stupid XSL and XSLT Tricks

A Stupid XSL/XSLT Trick is a use of XSL/XSLT for something unusual or amusing for which it wasn't necessarily designed.

Let charlie@IncrementalDevelopment.com know about your XSL/XSLT tricks to and they will be put on proud display (or linked to), properly accredited, in this area.

### Bob Copeland

Bob Copeland implemented a square root routine in XSLT:

- Compute square roots using the Babylonian method.

He comments:

> It would be interesting though if someone put together a set of stylesheets implementing a standard math library, e.g. including the trig functions via log tables or Taylor series. Because you never know when you'll need to compute an arctan while generating XML documents.

### Mike Edwards

Mike Edwards thinks he saw another version of this idea somewhere. I know I've suggested that someone try it. And now here it is:

- Solve the Towers of Hanoi problem with XSLT.

### David Caveney

⇨ http://incrementaldevelopment.com/xsltrick/

# Vorteile von XSLT

Freie Universität Berlin

- plattformunabhängig
- relativ weit verbreitet
- Verarbeitung in Web-Browsern
- Standard-Transformationen (wie XML ➔ HTML) einfach zu realisieren.
- Nicht nur HTML, sondern beliebige andere Sprachen können erzeugt werden.
- extrem mächtig

# Nachteile von XSLT

- Entwickler müssen speziell für die Transformation von XML-Dokumenten neue Programmiersprache lernen.
- Anbindung von Datenbanken umständlich
- manche komplexe Transformationen nur umständlich zu realisieren.
- Terminierung kann nicht garantiert werden.

Fazit: XSLT nur für Standard-Transformationen verwenden!

# Wie geht es weiter?

Freie Universität Berlin

## heutige Vorlesung

- ☑ Warum XML-Dokumente transformieren?
- ☑ XSLT

## Vorlesung nächste Woche

- XSLT – 2. Teil
  - noch mehr von XSLT
  - XSL-FO
  - XSLT 2.0