XSLT programming language

Jaana Holvikivi
Metropolia

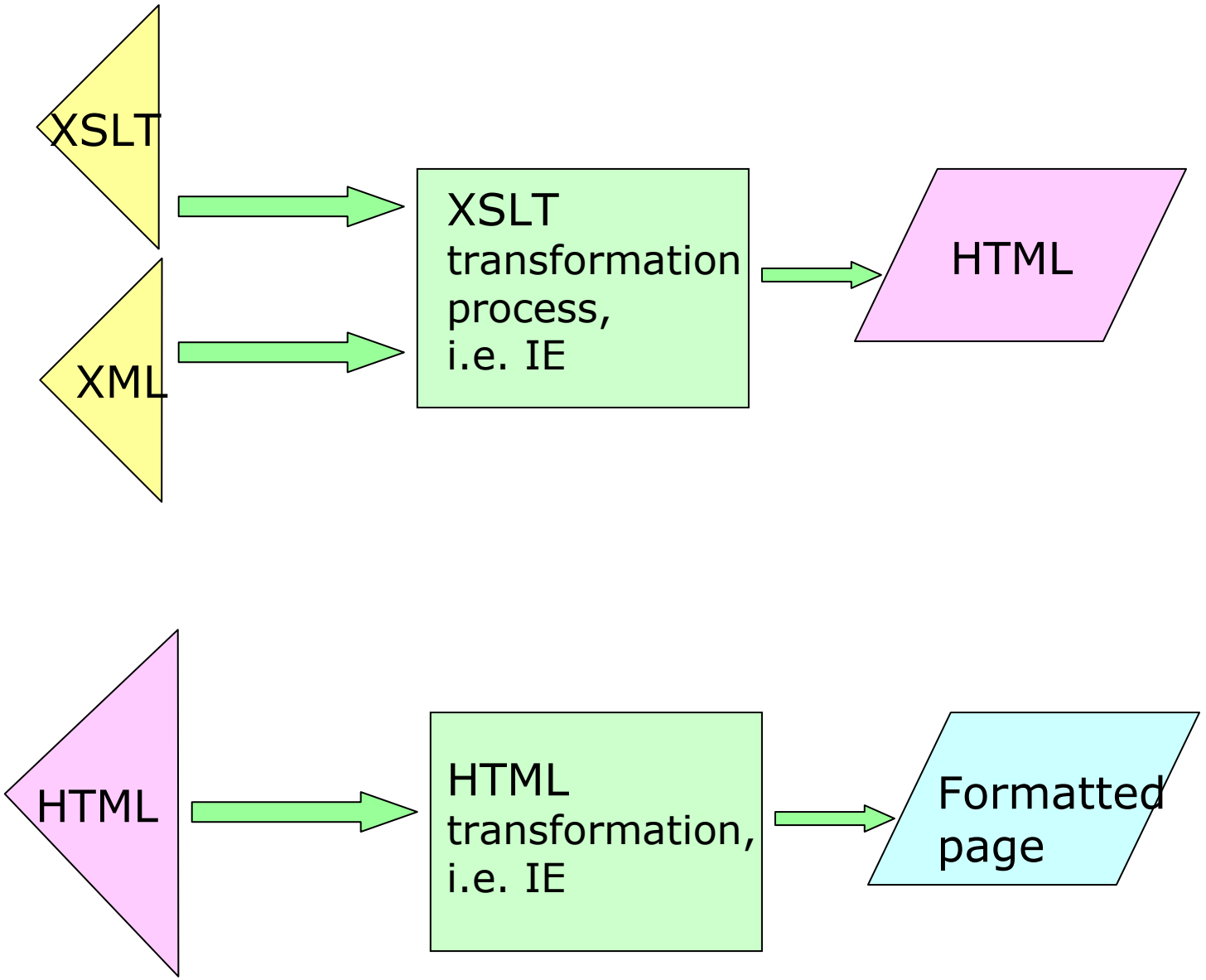
1 Transforming an XML document into HTML

```
<?xml version="1.0" >
<?xml-stylesheet type="text/xsl" href="cd_catalog.xsl"?>
<CATALOG>
   <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
    </CD>
</CATALOG>
```

This example is developed further in http://www.w3schools.com

Transforming an XML document into HTML: xsl

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform ">
<xsl:template match="/">
     <html><body>
     <table border="2" bgcolor="yellow">
     <tr>
     <th>Title</th>
     <th>Artist</th>
     </tr>
     <xsl:for-each select="CATALOG/CD">
          <tr> <td><xsl:value-of select="TITLE"/></td>
          <td><xsl:value-of select="ARTIST"/></td> </tr>
     </xsl:for-each>
     </table>
     </body> </html>
</xsl:template>
</xsl:stylesheet>
```


XSLT
XML
XSLT transformation process, i.e. IE
HTML
HTML
HTML transformation, i.e. IE
Formatted page

Explanation: XSL transformation

- XSL style sheet is an XML file, starting with the xml declaration
- **xsl:stylesheet** element tells that it is a style sheet or xsl:transform
- Template
  - primary transformation process
  - <xsl:template> has an optional attribute test (match)
- **xsl:template match="/"**
  indicates the start of processing (template) from the root node (/)
  which is default, and progresses from one node to another

Header of the file

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

in case of IE 5 use the draft recommendation (outdated)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

version 2.0 is located at:
http://www.w3.org/tr/xslt20/
refer to namespace:

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="2.0">
```

Explanation continues

- **xsl:for-each** element locates the element in the XML document and repeats the template for each
- **xsl:value-of** element selects a child node from the tree and gives the value to the template.
- **select** attribute selects a node from the source file. It's syntax is called **XSL Pattern**,
  and it works like directory tree navigation with a slash (/) indicating a subdirectory

XSL transformation example 2

XML -document

```
<?xml version="1.0"?>
   <?xml-stylesheet type="text/xsl"
   href="message.xsl"?>
<message type="final">
   <greeting>So long, and thanks for all the fish!
   </greeting>
   </message>
```

The transformation

```
<?xml version="1.0"?>
    <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform ">
<xsl:output method="html" indent="yes"/>
<xsl:template match="/">                    template with attribute match and rule "/"
<html>                                      text is copied to result
    <xsl:apply-templates/>                  set of nodes is called for processing
    </html>
</xsl:template>
<xsl:template match="message">              rule: message
<head><title>                               text
    <xsl:value-of select="@type"/>          extracts the value of the attribute
    message                                 text is copied to the result
    </title></head>
    <body><p>
    <xsl:value-of select="."/>              extracts the values of all child nodes
</p></body>
</xsl:template>
</xsl:stylesheet>
```

XSLT structure

Stylesheet	top level
Output	top level
Import, Include	top level
Variable, Param	top level (or lower)
Template	top level
apply-templates	
Template	top level
call-template	
Template	top level
other elements	

Explanation

- Current node: is the default, from where the tree structure is processed in any given situation
- If there is no ”match” attribute, processing starts in the root node
- Every document has one root node. It is not the same as document element, which is the outermost element that contains all others.
- The root node is defined as the concatenation of string values of all its element and text children.
- Every node, except the root, has a parent.
- If the element has no xsl prefix (it is not part of the XSL namespace) it is not processed. It is copied as such to the output.

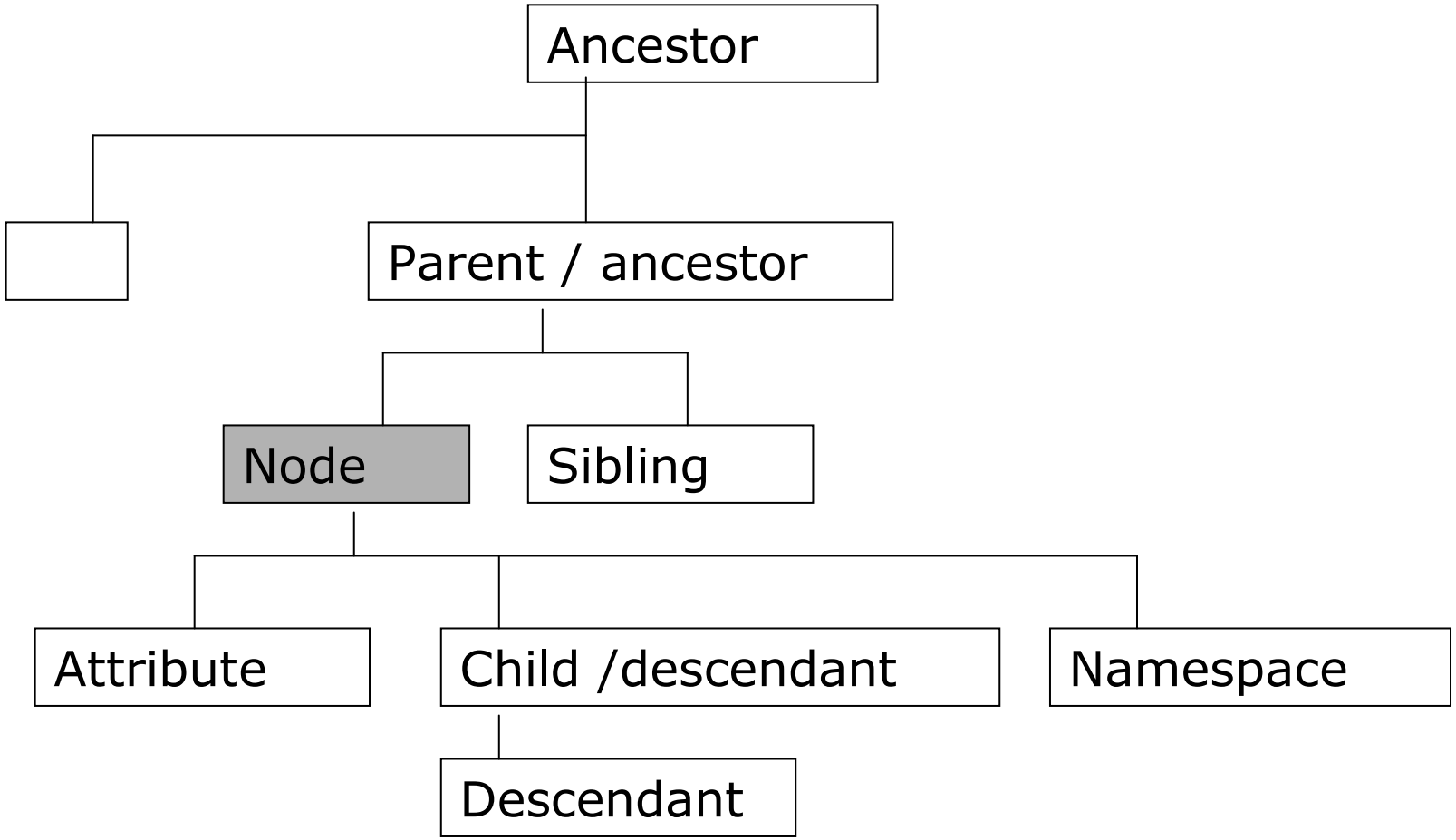
Explanation

- **<xsl:apply-templates/>** declaration causes that all applicable templates are processed, and the results are written in the result tree (document);

  if there is no other template defined for a situation, the data contents are written to the result
- If there is no matching template rule, the processor invokes the built-in template rule for element nodes, which executes <xsl:apply-templates/>
- The built-in template rule for text nodes copies the text node to the output.

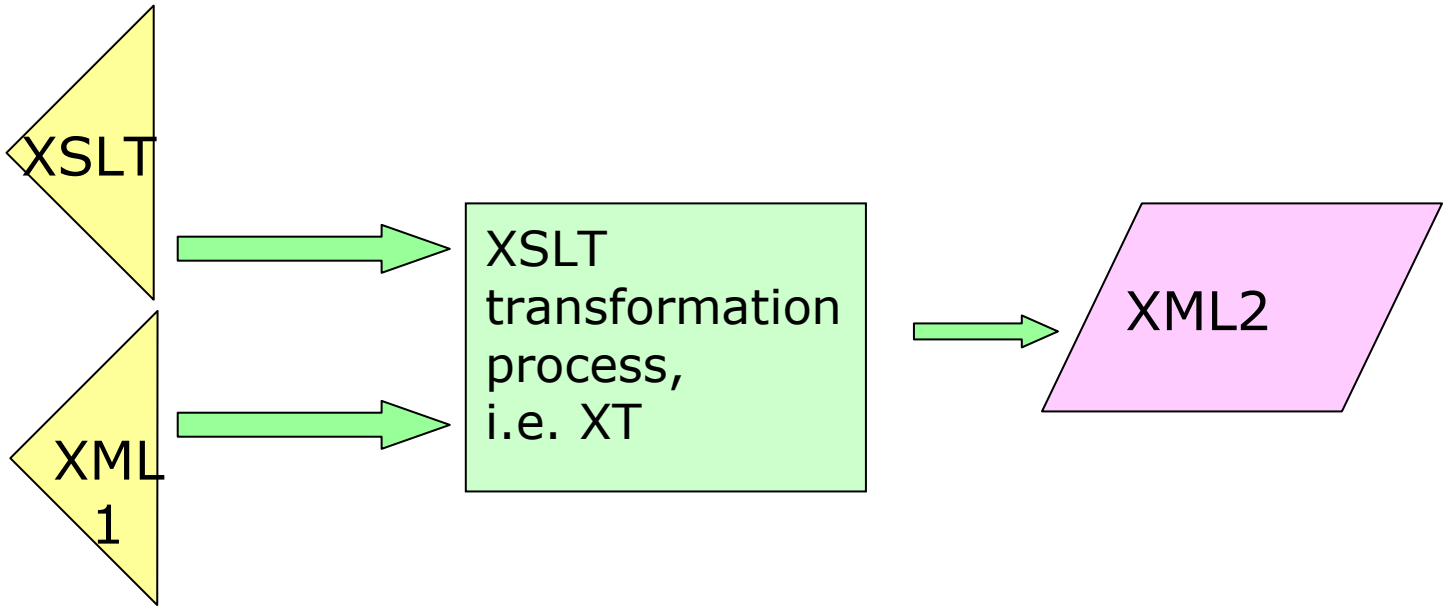
```
<xsl:template match="text()">
    <xsl:value-of select="." />
</xsl:template>
```

Document tree



What an XSL transformation does ?

- When there are two different data models (XML Schema, DTD), a transformation is needed to make one similar to the other for processing in the other system by its applications
- Extensible Style sheet language is for creating style sheets (and more)
- source tree
- result tree
- the program for the transformation is XSLT
- a language for layout formatting is XSL Formatting display
- XSLT can output HTML code, XML code, CSS or even a text file


XSLT
XML
1
XSLT
transformation
process,
i.e. XT
XML2

What XSL transformation does, example?

Company A order data:

```
<?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>
```

What XSL transformation does (2) ?

Company B order data:

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

XSL transformation

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

<xsl:template match="/">
  <order>
    <date>
      <xsl:value-of select="/order/date/year"/>/<xsl:value-of
      select="/order/date/month"/>/<xsl:value-of
    select="/order/date/day"/>
    </date>
    <customer>Company A</customer>
    <item>
      <xsl:apply-templates select="/order/item"/>
      <quantity><xsl:value-of
    select="/order/quantity"/></quantity>
    </item>
  </order>
</xsl:template>
```

XSL transformation cont.

```
<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>
      <!--other part-numbers would go here-->
      <xsl:otherwise>00</xsl:otherwise>
    </xsl:choose>
  </part-number>
  <description><xsl:value-of select="."/></description>
</xsl:template>
</xsl:stylesheet>
```

The main template

- The primary XML processing feature in XLM is to apply “template” procedures to matching XML elements in the source document
- <xsl:template> uses an optional attribute "match" that specifies the element type that the template should be applied to.
- All matching nodes are specified in relation to the current node.
- If match is not specified, the default mach is the root of the document.
- The <xsl:apply-templates/> declaration causes all matching templates to be processed and their output to be inserted at this point in the output document.
- If an element is not part of the XSL namespace (tags prefixed with xsl) they will not be processed by XSLT. The elements will be directly copied to the output.

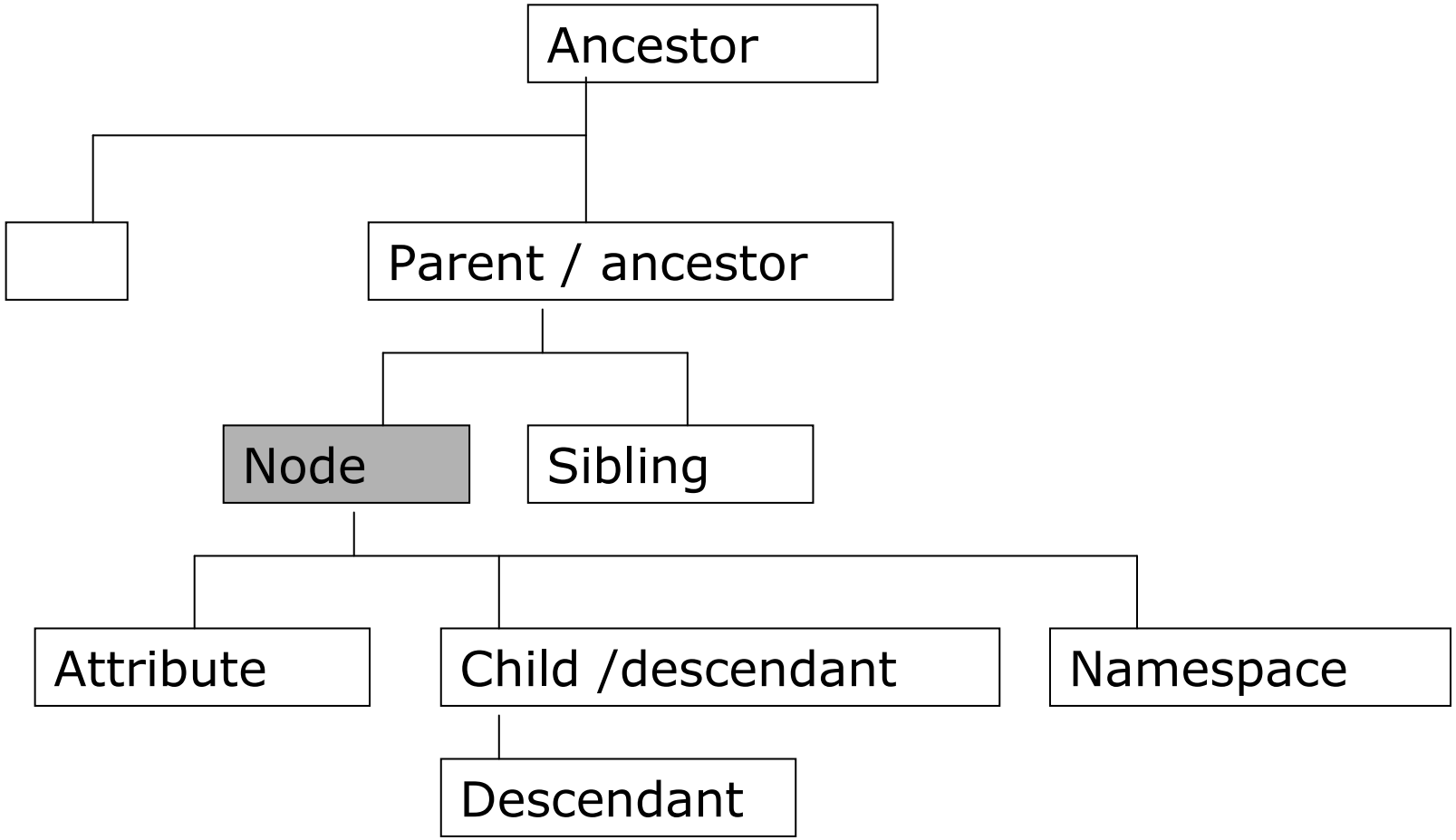
Location paths

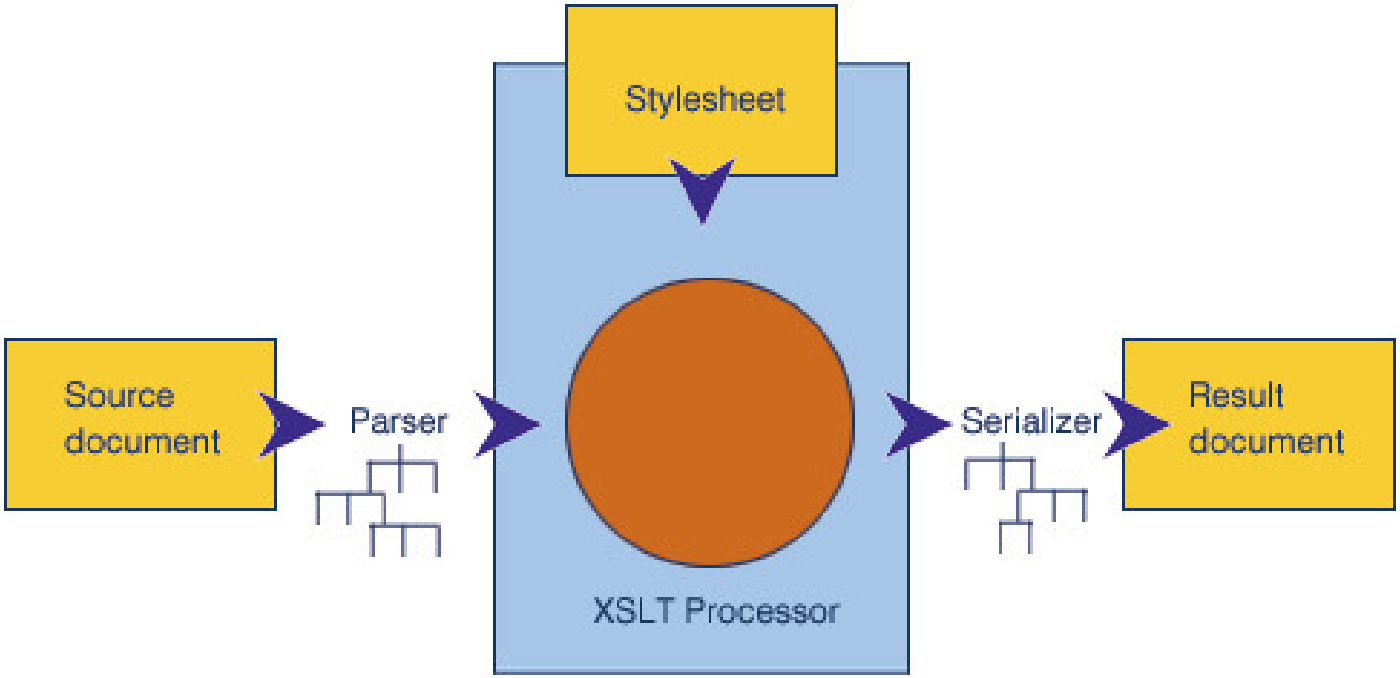
- Relative location paths
  - a path that starts from an existing location
  - sequence of one or more location steps separated by /
  - steps are composed from left to right
  - the initial step selects a set of nodes relative to the context node
  - each node in this set is used as a context node for the following step
- An absolute location path
  - consists of / optionally followed by a relative location path
  - A / by itself selects the root node of the document
- <xsl:template match=”/”>
- <xsl:value-of select=”/name/first”>

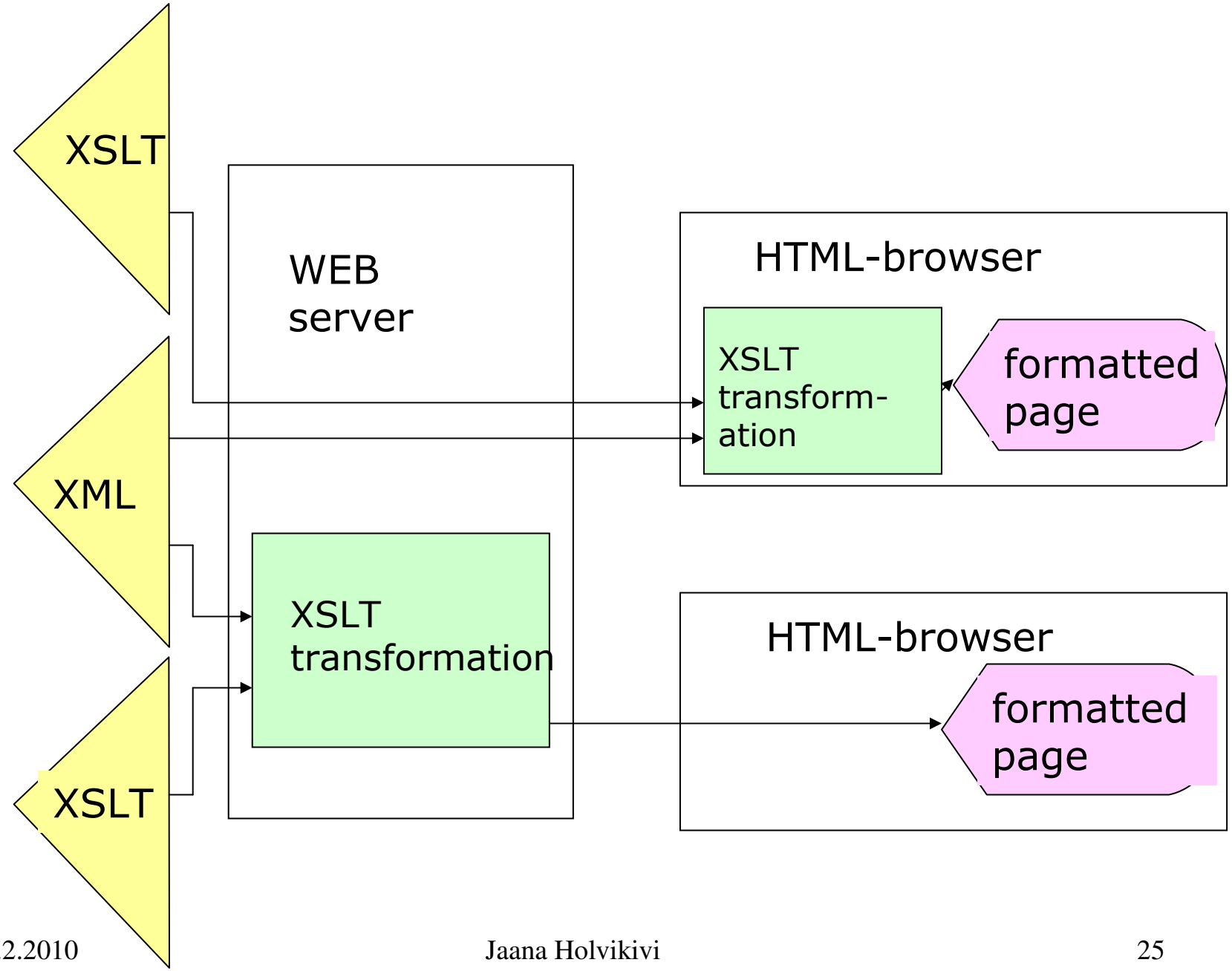
Xpath

- document root:<xsl:template match=”/”>
  <xsl:value-of select=”order”/>
  <xsl:value-of select=”/order/*/price”/>
  <xsl:value-of select=”.”/> *context node (relative)*
- *<xsl:template match="//atom">*
  finds any children
- attributes:
  <xsl:value-of select=”customer/@id”>
- <xsl:output method=”*xml* or *html* or *text*”
  version=”*version*”
  encoding=”*encoding*” *ie. ”utf-8”*
  omit-xml-declaration=”*yes* or *no*” *when result is a subdocument*
  standalone=”*yes* or *no*”
  cdata-section-elements=”*CDATA sections*”
  indent=”*yes* or *no*”/> *adds formatting*

Document tree




Stylesheet
Source document
Parser
XSLT Processor
Serializer
Result document


XSLT
XML
XSLT
WEB
server
XSLT
transformation
HTML-browser
XSLT
transform-
ation
formatted
page
HTML-browser
formatted
page

XSLT elements

- Elements used to define template rules and control the way they are invoked
  - <xsl:template> (top-level)
  - <xsl:apply-templates>
  - <xsl:call-template>
- Elements defining the structure of the stylesheet (top-level)
  - <xsl:stylesheet>
  - <xsl:include>
  - <xsl:import>

- Elements used to generate output
  <xsl:value-of>
  <xsl:element>
  <xsl:attribute>
  <xsl:comment>
  <xsl:processing-instruction>
  <xsl:text>
- Elements to control sorting and numbering
  <xsl:sort>
  <xsl:number>
- Elements used to control the final output format
  <xsl:output> (top-level)

- Elements used to define variables and parameters
  <xsl:variable> (top-level)
  <xsl:param> (top-level)
  <xsl:with-param>
- Elements used to copy information from the source document to the result
  <xsl:copy>
  <xsl:copy-of>
- Elements used for conditional processing and iteration
  <xsl:if test=" "> </xsl:if>
  <xsl:choose>
    <xsl:when> </xsl:when>
    <xsl:otherwise> </xsl:otherwise> </xsl:choose>
  <xsl:for-each>

Functions: Character strings

```
<?xml version="1.0"?>
    <crew>
    <member>Mamma</member>
    <member>Pappa</member>
    <member>Moomintroll</member>
    </crew>
```

functions:

translate (string, from, to) – to change characters in a string

translate (string, $upper, $lower) - changes uppercase characters to lower case

sum() - calculates a sum

boolean () – gets values true or false,

count() – number of nodes

Transformation PROGRAM

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/crew">
  <FAMILY>
   <xsl:apply-templates/>
    </FAMILY>
</xsl:template>
<xsl:template match="member">
  <CREATURE>
   <xsl:value-of select="translate(current(),
   'abcdefghijklmnopqrstuvwxyz',
   'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
    </CREATURE>
</xsl:template>
</xsl:stylesheet>
```

Sorting

```
<?xml version="1.0"?>
<crew>
  <member name="Mamma">
      <gear>handbag</gear>
      <gear>apron</gear>
  </member>
  <member name ="Pappa">
   <gear>pipe</gear>
   <gear>hat</gear>
   </member>
  <member name ="Mymlan">
   <gear>mirror</gear>
   <gear>bow</gear>
   <gear>dress</gear>
  </member>
</crew>
```

xsl:sort

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:template match="/">
<html>
<head><title>Moomin belongings</title></head>
<body>
<xsl:apply-templates select="/crew/member">
<xsl:sort select="@name" />
</xsl:apply-templates>
</body></html>
</xsl:template>
```

- ...cont.

xsl:sort (continues)

```
<xsl:template match="member">
<h2><xsl:value-of select="@name"/>'s belongings</h2>
<ul>
<xsl:for-each select="gear">
<xsl:sort select="." />
<li><xsl:value-of select="."/></li>
</xsl:for-each>
</ul>
</xsl:template>

</xsl:stylesheet>
```

Descriptive markup (XML)

- logical structure
- self-descriptive (element names)
- content and form separated
- syntactic structure, no semantics

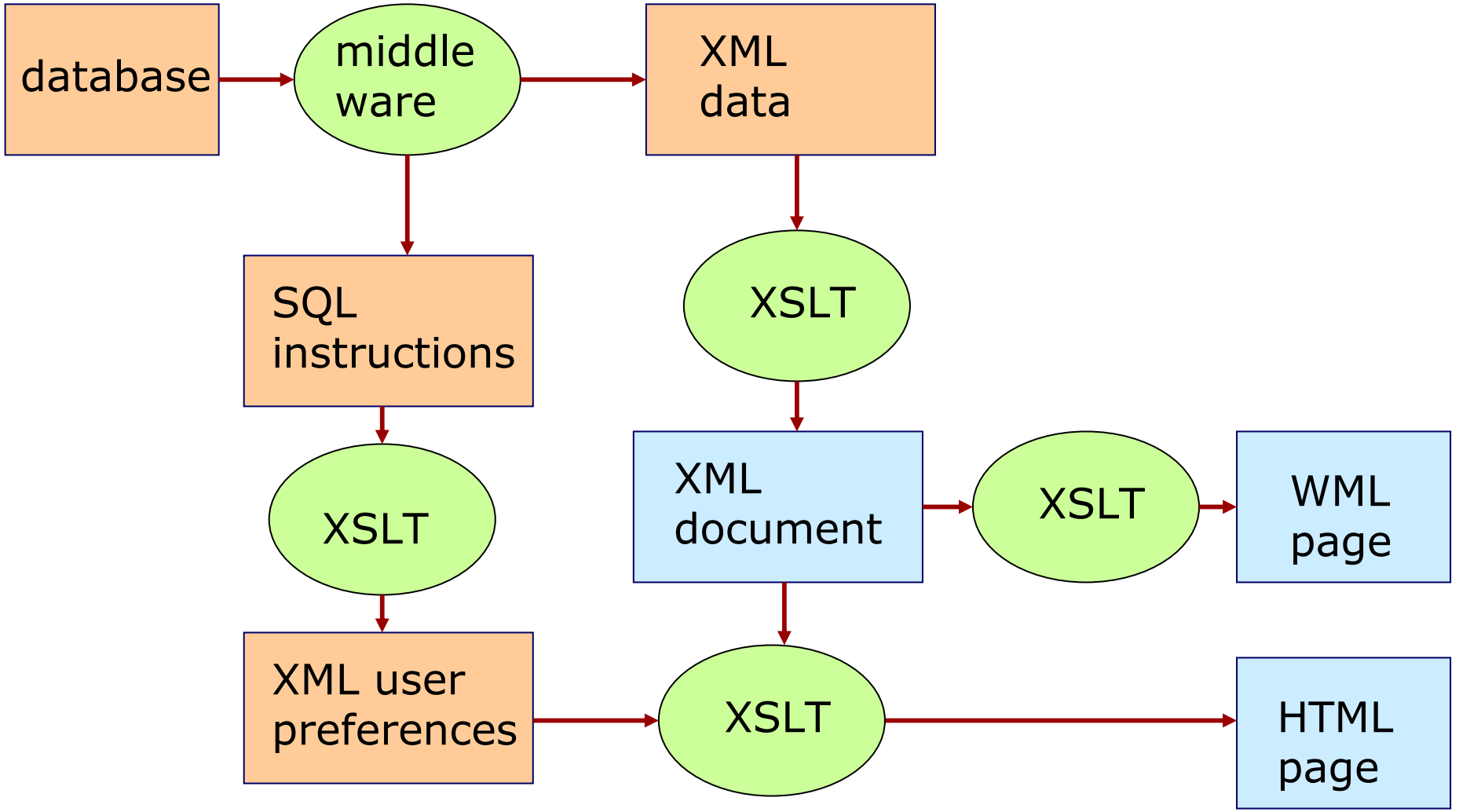
Imperative versus Declarative programming

- Imperative or procedural programming
  - Java, C++
  - what to do and how to do it (order of processing)
  - content and format mixed
- declarative programming (xslt)
  - templates, conditions, output
  - order of processing not defined, no algorithms
  - Prolog, xslt, Haskell
  - WHAT to do, not HOW to do it

XSLT advantages

- XSLT is not only a formatting and style language but a declarative programming language
- functional language
- "no side-effects": changes somewhere in the XML do not affect other processing (in theory)
- "XSLT gives you all the traditional benefits of a high-level declarative programming language, specialized to the task of transforming XML documents." Kay 2001
- "data independence" compared to procedural languages
- inefficient use of memory: the tree structure is created in the memory
- XSLT uses Xpath language to address the tree structure: it is kind of query language that understands the tree structure

A pipeline for tranformations



A functional programming language

offers you:

- Substantially increased programmer productivity (Ericsson measured an improvement factor of between 9 and 25 in one set of experiments on telephony software).
- Shorter, clearer, and more maintainable code.
- Fewer errors, higher reliability.
- A smaller "semantic gap" between the programmer and the language.
- Shorter lead times.

Much of a software product's life is spent in *specification*, *design* and *maintenance*, and not in *programming*. Functional languages are superb for writing specifications which can actually be executed (and hence tested and debugged).

Higher security