

eXtensible Styleheet Language (XSL)

XML Stylesheet Language

- Stylesheets contain *templates* that describe how to present a document
- Using stylesheets the same document can be presented in *multiple* output formats
- Stylesheets are expressed in XML syntax, using *tags* to define formatting rules

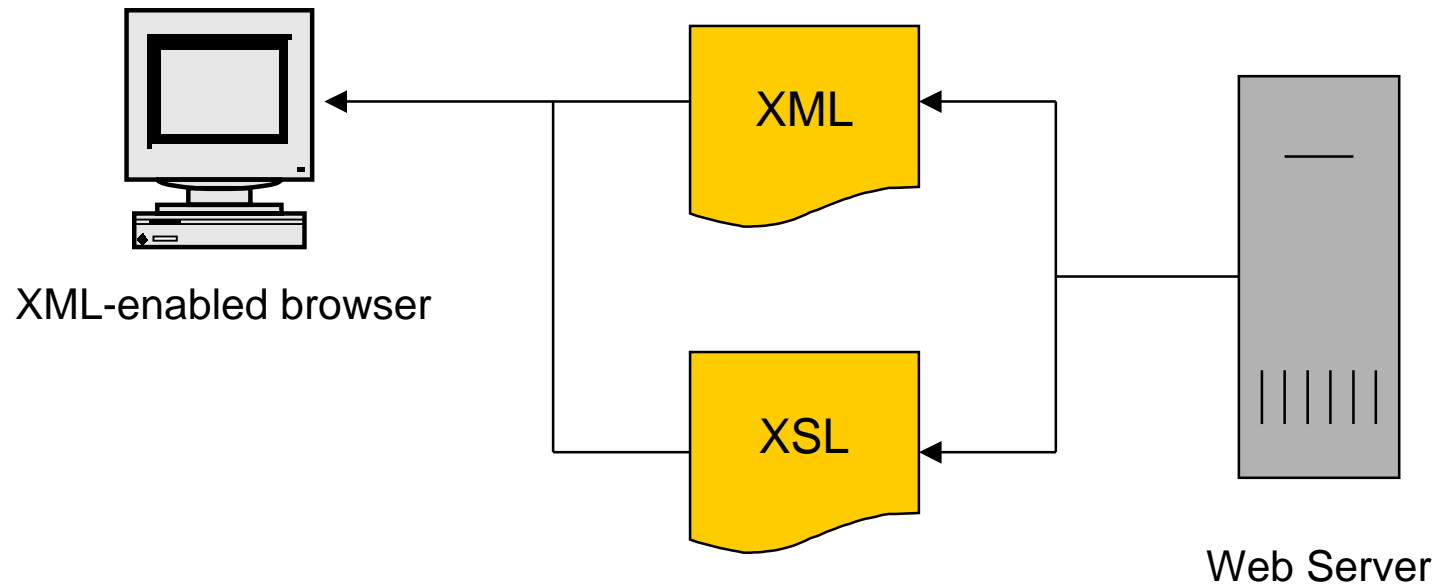
Purpose

- XSL stylesheets are used for two purposes
- Transform XML documents into other formats (e.g. posting a document as HTML, RTF, or speech)
- Manipulate the information in XML documents (e.g. perform searches, apply filters, add information)

Client-Side XSL (1)

- Client-side XSL takes advantage of the computing power on the client
- The client performs processing separately from the server (even customization)
- Different views of the data can be created on the client without taxing the server
- As attractive as this model is, the majority of browsers are not XML-enabled

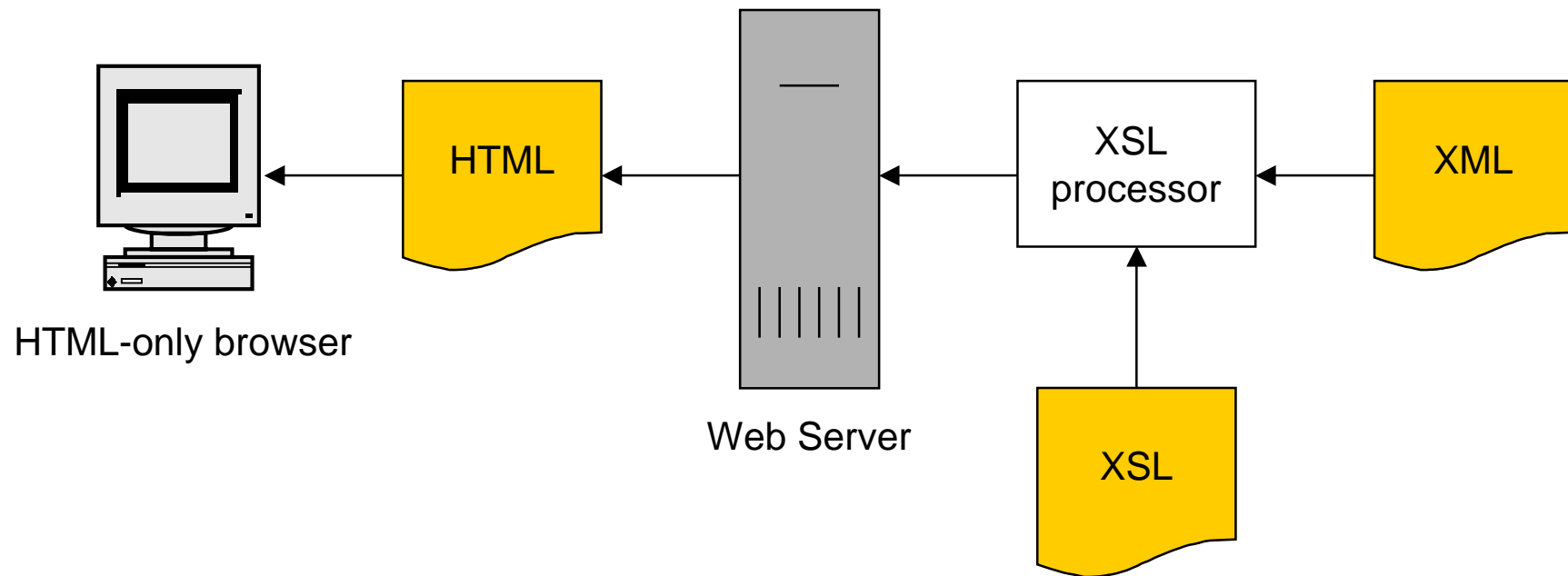
Client-Side XSL (2)



Server-Side XSL (1)

- In this model the conversion of XML to HTML takes place on the server
- Although an average exchange between client and server will be smaller, the exchanges will likely be more frequent
- Every time the client wants a different view of the data, the server needs to generate another Web page and send it to the client

Server-Side XSL (2)



Templates

- Templates describe how to present the data contained in an XML document
- A template has two components
 - *Patterns* which are matched against nodes in the source document tree
 - *Output elements* which can be instantiated to form part of the result document tree
- Consider the following examples

Examples

- The following template rule presents a portfolio element as a table

```
<xsl:template match="portfolio">  
  <table><xsl:apply-templates/></table>  
</xsl:template>
```

- The next rule selects stocks on the NYSE and shows their content in a table row

```
<xsl:template match="portfolio/stock[@exchange='nyse']">  
  <tr><xsl:apply-templates/></tr>  
</xsl:template>
```

Pattern Syntax

- The following table shows the syntax that you will most commonly use in patterns

Syntax	Meaning	Example
@	attribute	[@exchange= 'nyse']
//	descendant	. //symbol
[number]	position	stock[0]
.	current node	. /name
..	parent node	.. /stock[0]

More Pattern Examples

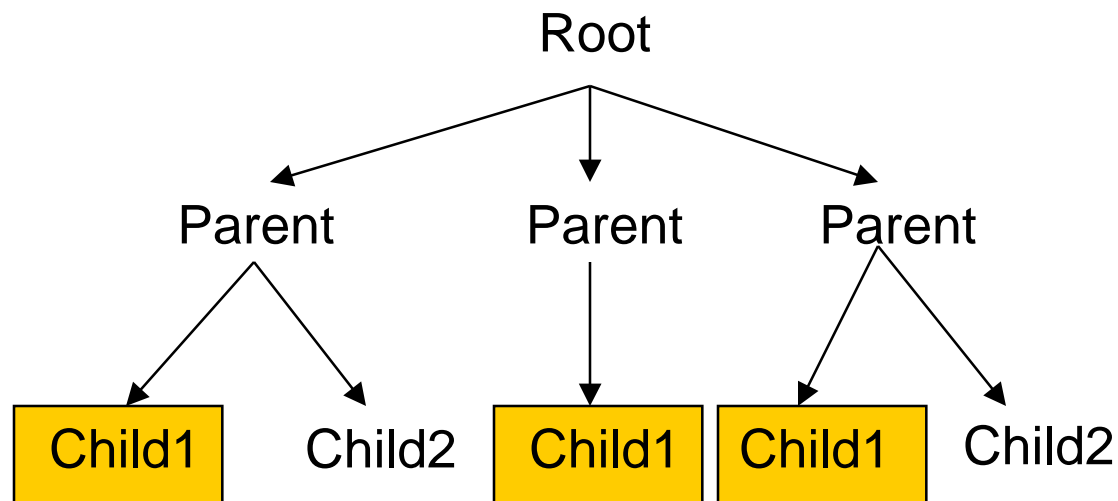
- Here are examples of patterns:
 - `chapter/section[last()-1]`
 - `/doc/chapter[0]/section[2]`
 - `section[@author="mw"]`
- This what these example represent:
 - Collection of all last sections in chapters
 - Section 3 in Chapter 1 in a document
 - All sections authored by "mw"

Processing Model

- The result tree is constructed
 - By applying the template rule for the root node, and instantiating the template
 - This usually triggers further template rules, which when instantiated trigger ...
- Using the following transformation algorithm, the tree is created *iteratively*

Algorithm (1)

1. At any given time, there is a list of *source nodes*, found by matching a pattern



Pattern: Root//Child1

Algorithm (2)

This list of source nodes defines the *context* in which the template associated with the pattern is applied

2. For each node in the context, a fragment of the *result tree* is constructed by instantiating the template associated with the pattern

Algorithm (3)

3. In the course, new source nodes will be added to the list, usually by applying other templates (e.g. apply-templates)
4. The transformation comes to an end when *no further* nodes in the source document can be matched against any of the patterns

xsl:stylesheet

- Used once for every XSL stylesheet and is the root node of every stylesheet.

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

- The namespace URL refers to the actual XSL standard and cannot be processed

xsl:template

- Used to represent a template in which transformation actions are specified. Each stylesheet contains at least one of them
- Attributes:
 - Match: defines the context of the template, ie, where the template should be applied. The value of this attribute is a path expression.

```
<xsl:template match= "portfolio/stock">
```

xsl:value-of

- Used to insert the text value of the node indicated by the path expression
- Attributes:
 - Select: defines the path expression whose value is to be entered in the current context.

<xsl:value-of select="symbol"/>

xsl:for-each

- Used to set up a loop in which the same template is applied against multiple nodes
- Attributes:
 - Select: defines the path expression whose value is to be run through the template.

```
<xsl:for-each select= "stock[@exchange='nasdaq']">
```

xsl:if

- Used to make part of the formatting optional, and instantiated only when a certain condition is true
 - Attributes:
 - Test: condition to check
- `<xsl:if test="position() = 0"> ... </xsl:if>`

xsl:apply-templates

- Used to invoke the application of other templates as specified by the path
- Attributes:
 - Select: defines the template to use

`<xsl:apply-templates select="portfolio/stock"/>`

xsl:element & xsl:attribute

- Creates a new element (attribute) in the result tree
- Attributes:
 - Name: name of the new element (attribute)

```
<xsl:element name="a">  
  <xsl:attribute name="href">  
    <xsl:value-of="@href"/>  
  </xsl:attribute>  
</xsl:element >
```

Push or Pull? (1)

- Data-driven
- This model is referred to as *pull* model
 - Apply this model if your data is irregular
 - An example is a news clip where each element (e.g. author, title, date) is generally used only once and may need to be formatted differently
- Files
 - portfolio-pull.xsl, portfolio-pull.xml

Push or Pull? (2)

- Template-driven style
- Referred to as a *push* model
 - Apply this model if your data is regular
 - An example is a phone directory where a repetitive structure (name, number etc.) is used to represent directory the entries
- Files
 - portfolio-push.xsl, portfolio-push.xml