

XSLT - Transforming XML documents

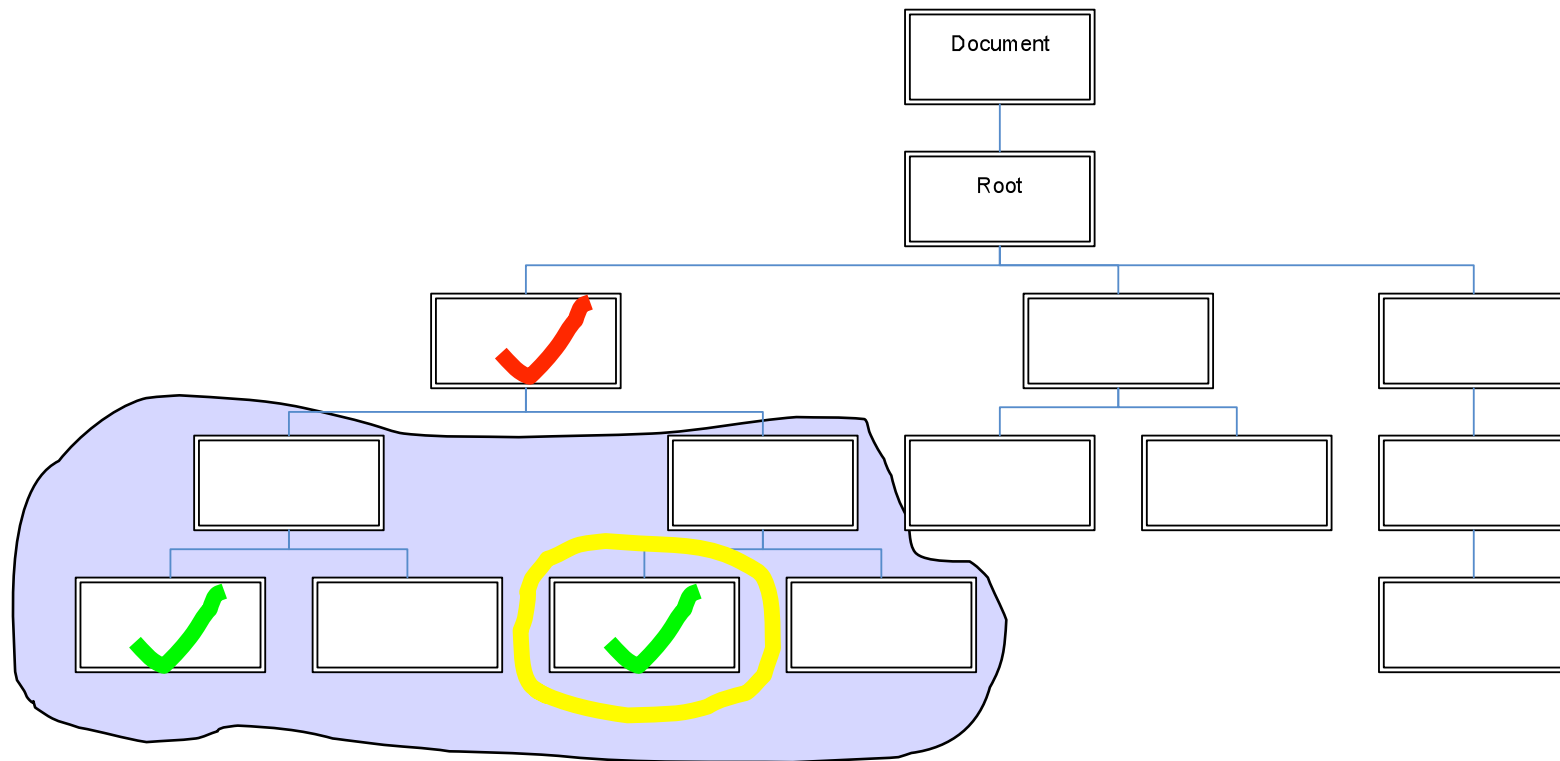
CS 431 - February 25, 2008

Carl Lagoze - Cornell University

Xpath Concepts

- Context Node (starting point)
 - current node in XML document that is basis of path evaluation
 - Default to root (remember that root is "Document")
- Location Steps (directions)
 - Sequence of node specifications
 - Evaluation of each node specification creates a new context
 - always within previous context
 - Think of file paths
 - /nodeSpec/nodeSpec/nodeSpec
- Node Specification
 - Increasingly detailed specification of the sequence of nodes to which the location step should lead

Context, Axis, Node Test, Predicate



xpath examples

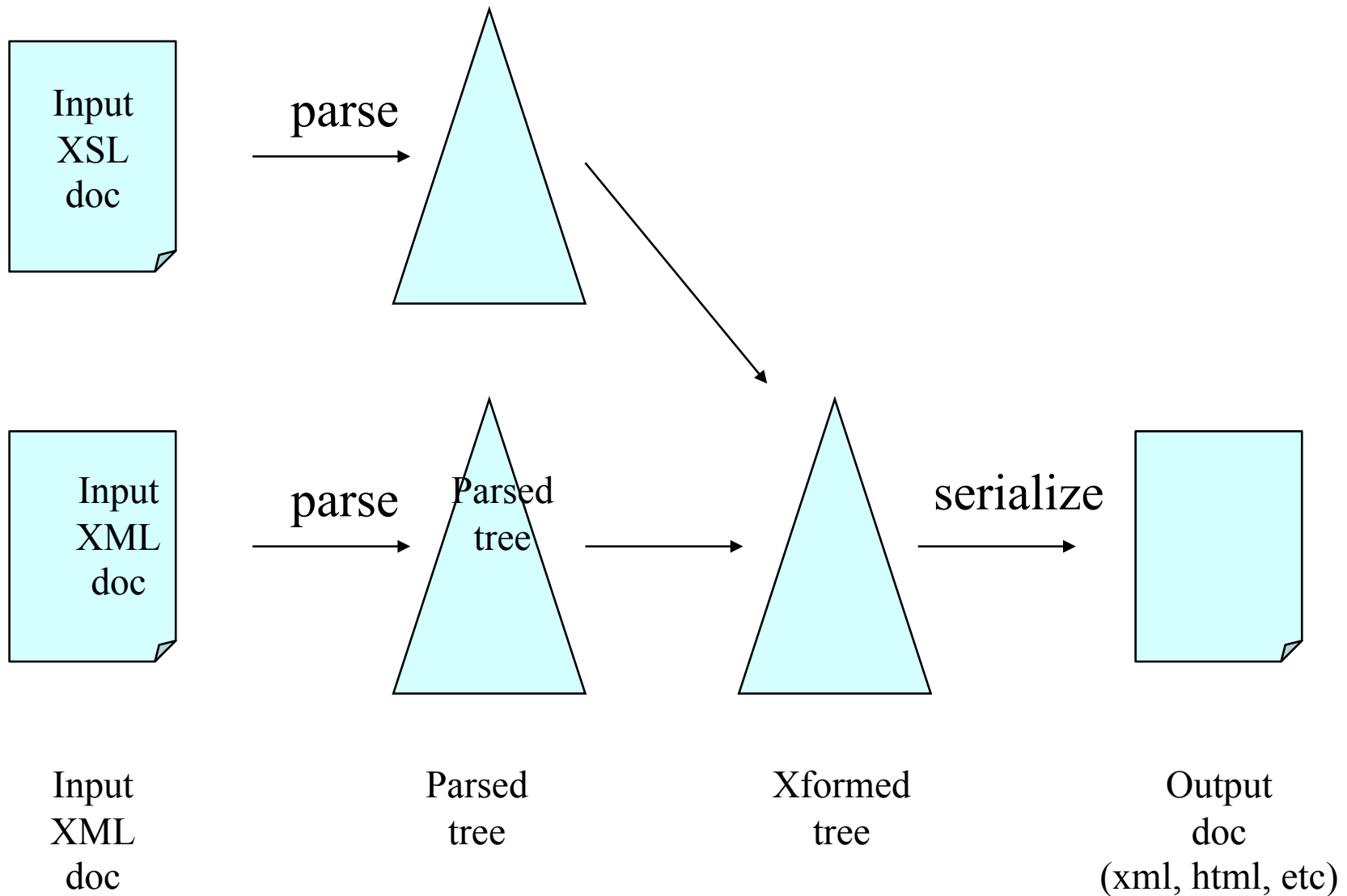
- `/child::source/child::AAA`
 - or `/source/AAA` since child is default axis
- `/child::source/child::*[position()=2]`
 - or `/source/*[2]`
- `/child::source/child::AAA[position()=2]/attribute::id`
 - or `/source/AAA[2]/@id`
- `/child::source/child::AAA/@*`
 - or `/source/AAA/@*`
- `/child::source/child::AAA[contains(., 'a1')]`
 - `/source/AAA[contains(., 'a1')]`
- `/descendant::BBB/child::CCC`

<http://www.cs.cornell.edu/courses/CS431/2007sp/examples/xpath/base.xml>

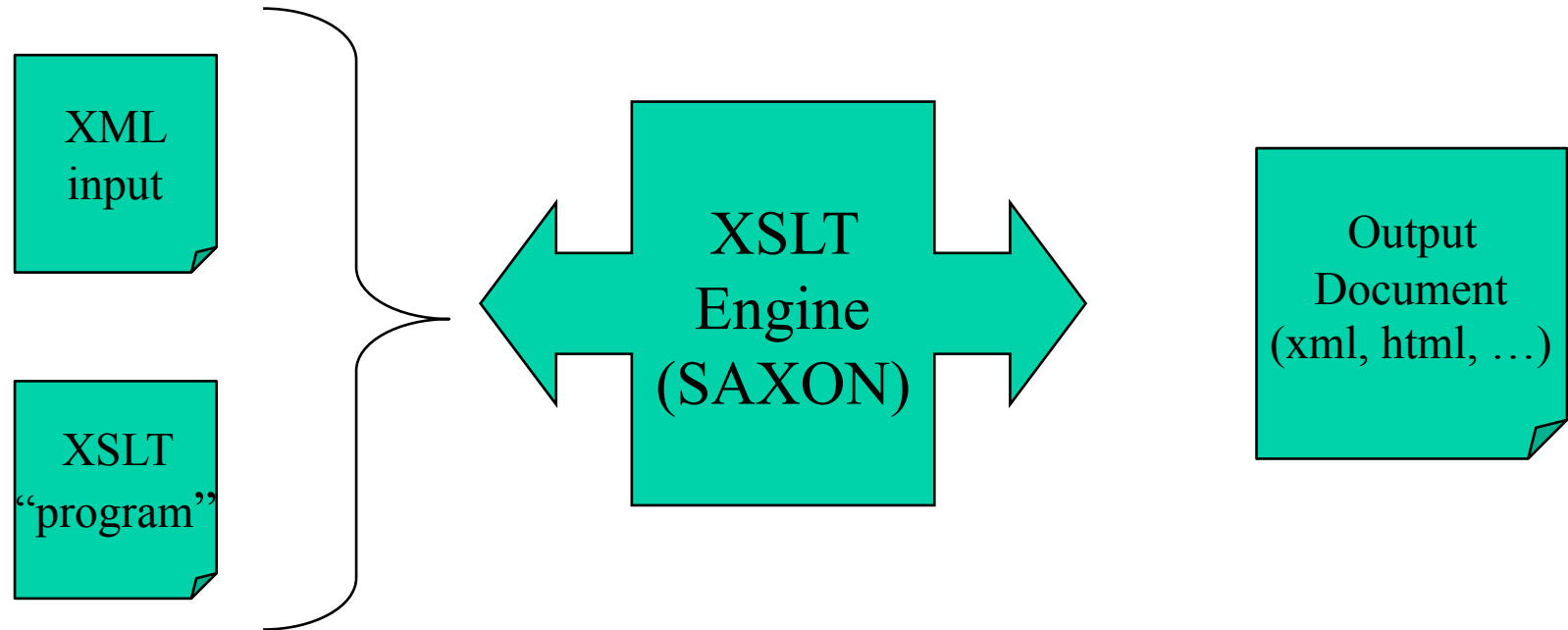
XML Transformations (XSLT)

- Generalized notion of transformation for:
 - Multiple renderings
 - Structural transformation between different languages
 - Dynamic documents
- Rule-based (declarative) language for transformations

XSLT Processing Model



XSLT “engine”



Stylesheet Document or Program

- XML document rooted in <stylesheet> element
- XSL tags are in namespace <http://www.w3.org/1999/XSL/Transform>
- Body is set of templates or rules
 - match attribute specifies xpath of elements in source tree
 - Body of template specifies contribution of source elements to result tree

Stylesheet Document or Program

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="para">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>
<xsl:template match="emphasis">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>
</xsl:stylesheet>
```

Template Form

```
<xsl:template match="XPathExpression">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>
```

- Elements from **xsl** namespace are transform instructions
- **match** attribute value is xpath expression setting rule for execution of body
- Sequential execution within template
- Non-xsl namespace elements are literals.
- **<xsl:apply-templates>**
 - set context to next tree step (default depth-first)
 - re-evaluate rules

XSL Execution Model

- Templates represent a set of rules
- Rule matching is done within current tree context
- Rules are not executed in order
- Precedence given to more specific rules
- Default behavior
 - Write element value
 - Reevaluate rules in new context resulting from depth-first tree step
 - Default behavior will **ALWAYS** happen unless overwritten by specific rule

Default Rules (Must replace to change them)

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

- Default <apply-templates> action is to walk all but attributes
- Applies to root node and element nodes
- Tree walk is depth first

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

- Applies to text and attribute nodes
- Copies value to output tree

Examples of default behavior

- base.xml
- null.xml
- nullExplicit.xml

Result Tree Creation

- Generates the output file
- Always occurs within a template

Result Tree Creation (1)

- Literals - any element not in xsl namespace is inserted into result tree

```
<xsl:template match="para">  
  <p>  
    This is the sentence  
  <xsl:apply-templates/>  
  </p>  
</xsl:template>
```

Result Tree Creation (2)

- `<xsl:text>` - send content directly to output (retain whitespaces)

```
<xsl:text>, and </xsl:text>
```

Result Tree Creation (3)

- `<xsl:value-of>` - extract element values (anywhere in the tree)

```
<tr>
  <td><xsl:value-of select="catalog/cd/title"/></td>
  <td><xsl:value-of select="catalog/cd/artist"/></td>
</tr>
```

Result Tree Creation (4)

- `<xsl:copyof>` - Copy selected nodes into result tree

```
<xsl:copy-of select="table"/>
```

Result Tree Creation (5)

- `<xsl:element>` - instantiate an element
- `<xsl:attribute>` - instantiate an attribute

```
<xsl:element name="newEl">  
  <xsl:attribute name="newAttr">  
    <xsl:value-of select="/top/AAA[1]"/>  
  </xsl:attribute>  
</xsl:element>
```

A simple example

- XML base file
 - simple.xml
- XSLT file
 - simple.xsl

Modifying rule set and context

- Context setting
 - `<xsl:apply-templates select="//bar">`
 - Modifies default depth-first behavior
- There are conflict resolution rules
- base.xml
- elements2.xsl

Modifying rule set and context

- Mode setting
 - `<xsl:apply-templates mode="this">`
 - `<xsl:template match="foo" mode="this">`
 - `<xsl:template match="foo" mode="that">`
- `students.xml`
- `students.xsl`

Namespaces in XSLT

- The XSL document **MUST** know about the namespaces of elements that it references (via XPATH expressions) in the instance document
 - baseNS.xml
 - elementsNS.xsl
- Watch out for the default namespace!!
 - baseNoNS.xml
 - elementsNoNS.xsl

XSLT Procedural Programming

- Sequential programming style
- Basics
 - for-each - loop through a set of elements
 - call-template - like a standard procedure call

For-each programming example

- XML base file
 - foreach.xml
- XSLT file
 - foreach.xsl

Call-template programming example

- XML base file
 - call.xml
- XSLT file
 - call.xsl

Variables

- Scoped in normal fashion
 - Global
 - Within tree nesting level
- No static typing - take type of setting
 - string, number, boolean, node-set (set of nodes, sub-tree)

Variables

- Initialization

- `<xsl:variable name="age" select="25"/>`
- Distinguish between string literal and xpath
 - `<xsl:variable name="city" select="'ithaca'"/>`
 - set variable to string "ithaca"
 - `<xsl:variable name="city" select="ithaca"/>`
 - set variable to result of xpath expression "ithaca"

Variables

- Initialization
 - Construct temporary tree
 - `<xsl:variable name="temptree">
 <foo><bar></bar></foo>
 </xsl:variable>`
 - Usage - precede by '\$'
 - `<xsl:value-of select="$city"/>`

Variables (assignment)

- No assignment after initialization
- Think functional programming model (LISP, ML, Scheme)
- Use conditional initialization (<xsl:choose>)
- Use recursion rather than iteration for repetitive tasks

Variables example

- `variables.xsl`

Inputs and outputs

- Inputs - Default is single input document
 - `Document('URL')` function returns root node of document at URL
- Outputs - Default is single XML document
 - `<xsl:output method="{“xml” | “html” | “text”}”/ >` changes output format
 - `<xsl:document href=“URL” >`
 - Anywhere in xsl file changes the output destination.

Associating an XML document with a transform

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="http://www.cs.cornell.edu/Courses/cs502/2002SP/Demos/xslt/simple.xsl"?>
<para>
  this is
  <emphasis>
    big
  </emphasis>
  text
</para>
```