# Transforming TEI with XSLT

Andrew Morrison

Bodleian Digital Library Systems and Services

# What is XSLT?

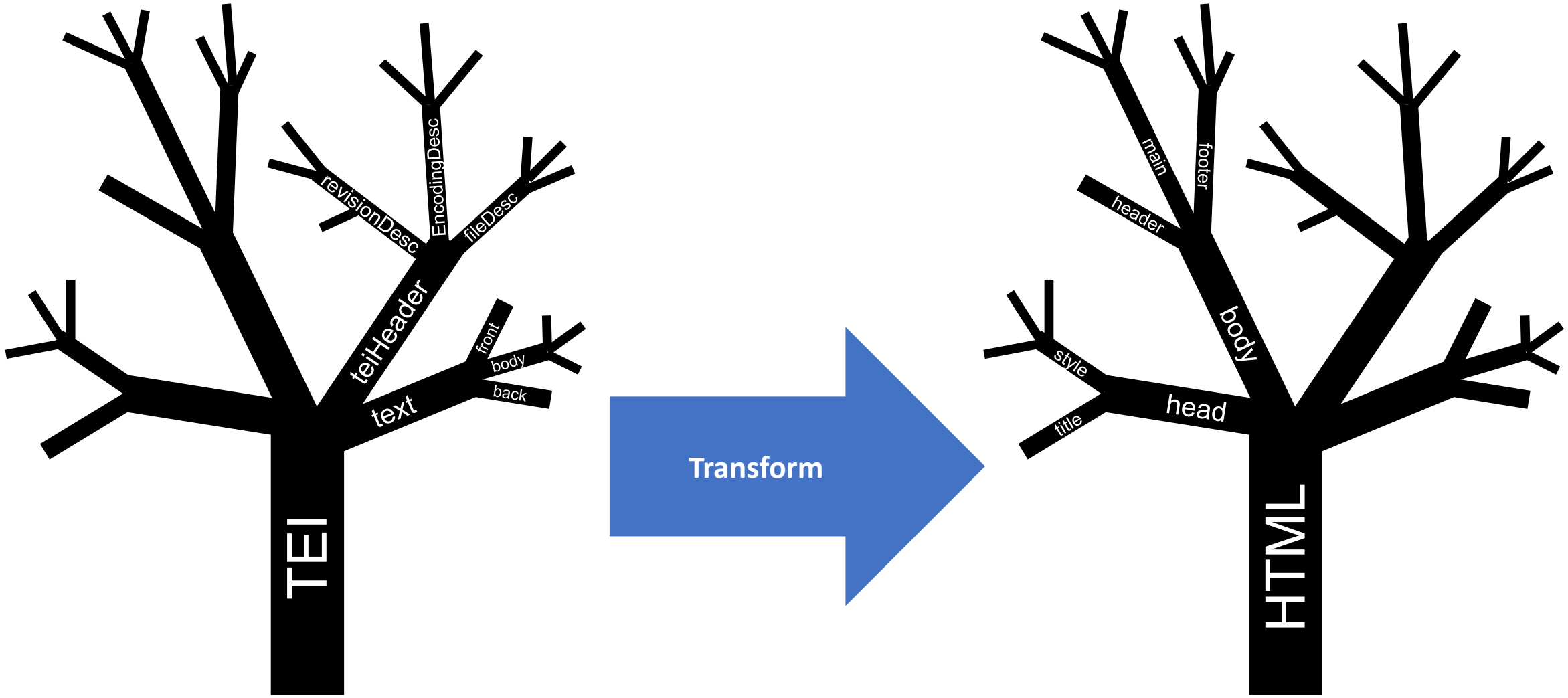
- eXtensible Stylesheet Language Transformations
- W3C recommendation
- Part of a family with XPath, XSL-FO, XQuery and XML Schemas
- A programming language for transforming XML documents, itself expressed in XML

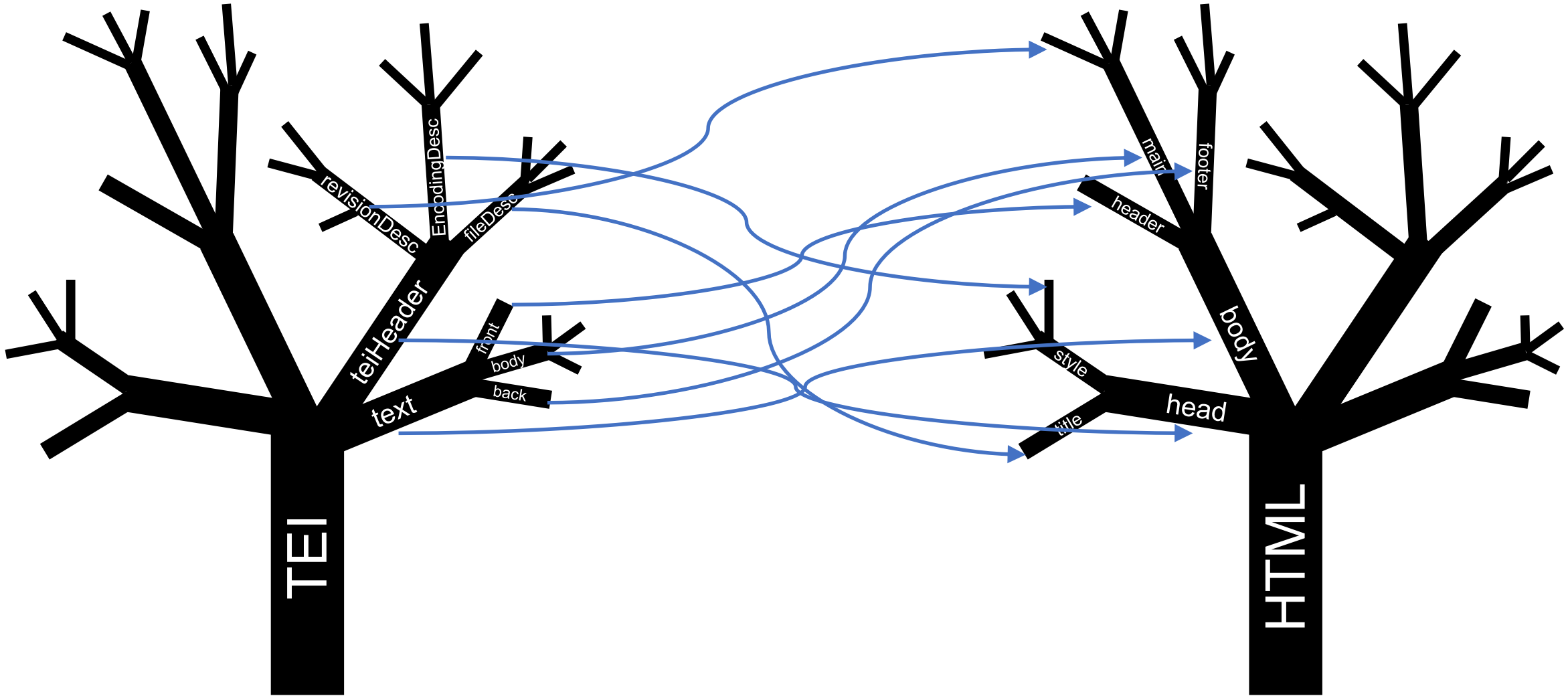
# Versions of XSLT

- XSLT 1.0
  - Circa 1999
  - Still the only version supported by some command-line tools and web server
  - Built into a lot of web browsers, but seldom used
- XSLT 2.0
  - Circa 2007
  - Added a lot of advanced features (only a couple of which we're going to mention here)
- XSLT 3.0
  - Circa 2017
  - A marginal gain on 2.0. Really only of interest when working with very large data sources or using very advanced techniques

# What can XSLT be used for?

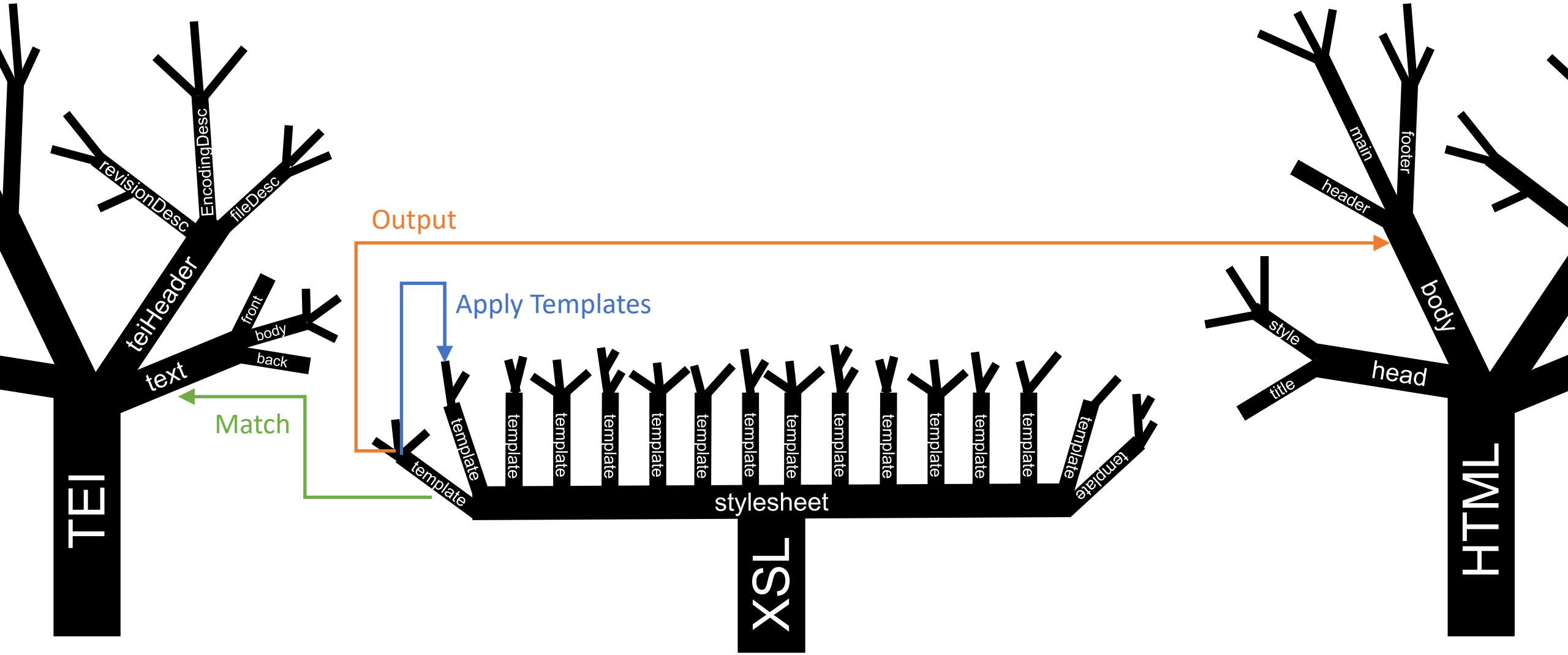
- Publishing (e.g. TEI to HTML, data to SVG, EAD to PDF)
- Upgrading (e.g. an old implementation of TEI to a new one)
- Bulk changes (e.g. fixing dates)
- Merging data sources (e.g. creating METS from EAD, augmented with some more detailed descriptions from TEI)
- Analysis (e.g. extracting information from TEI into a comma-separated text file, for importing into spreadsheet)

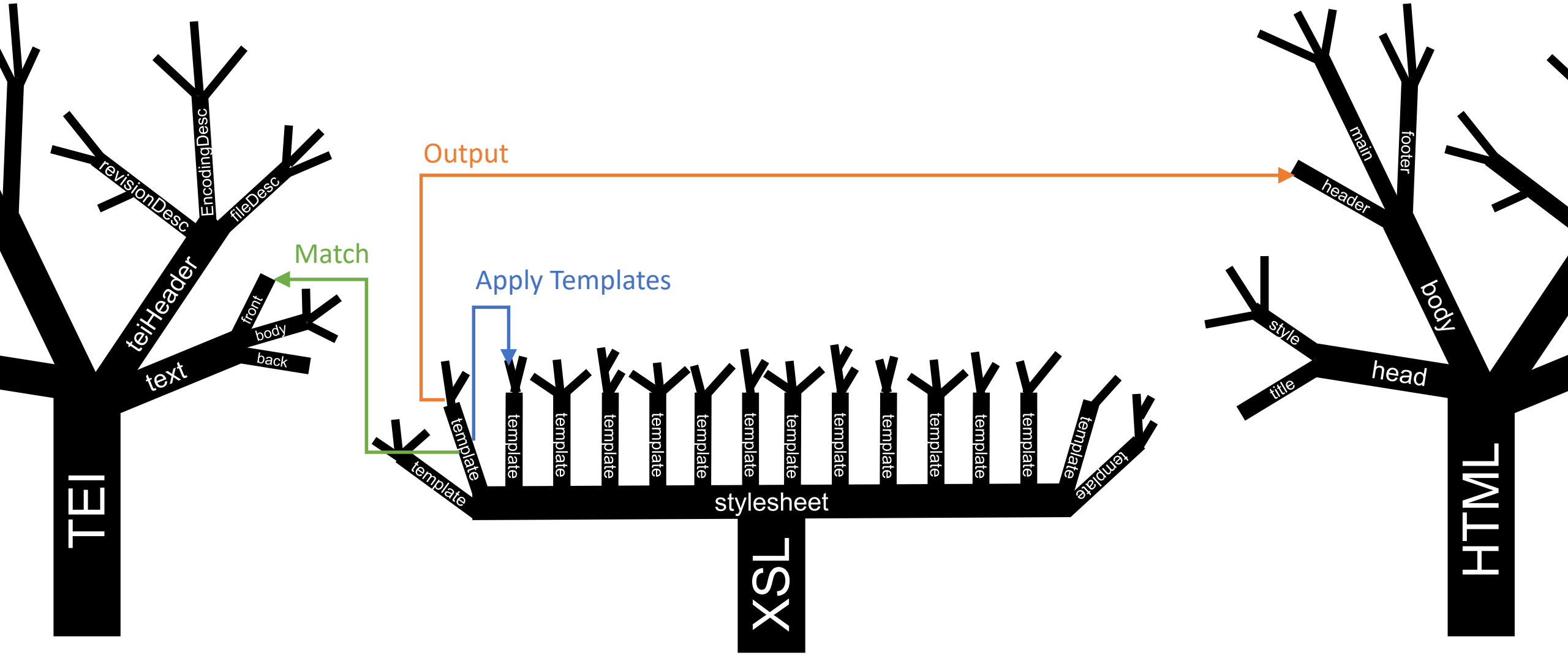

TEI
teiHeader
revisionDesc
EncodingDesc
fileDesc
text
front
body
back
Transform
HTML
head
title
style
body
header
main
footer


TEI
teiHeader
revisionDesc
EncodingDesc
fileDesc
text
front
body
back
HTML
head
style
title
body
header
main
footer

# The basics of XSLT

- Stylesheet
  - The root element of the .xsl file
- Templates
  - Each matches something in the source document
  - Each outputs a snippet of the output document
- Apply-templates
  - Instruction to pass processing on to another template


TEI
teiHeader
revisionDesc
EncodingDesc
fileDesc
text
front
body
back
Output
Apply Templates
Match
template
stylesheet
XSL
HTML
head
body
header
main
footer
style
title


TEI
teiHeader
revisionDesc
EncodingDesc
fileDesc
text
front
body
back
Match
Output
Apply Templates
XSL
stylesheet
template
HTML
head
title
style
body
header
main
footer

# XSLT processors

- The software that run the transformation specified in the XSLT
- Descends the tree structure, applying templates
- Picks templates on the basis of the most specific match
- If two templates are equally specific, the order in which they appear in the XSLT stylesheet is used
- A priority attribute can also be specified for each template
- One XSLT stylesheet can import templates from another (and override them)

# Matching using XPath

- XPath is a syntax for selecting nodes in a hierarchy, used in XSLT, XQuery, Schematron, and many others.
- In its simplest form, it is similar to how you navigate a hard drive (e.g. */TEI/teiHeader*)
- Context is key: e.g. once the teiHeader element is selected, use *fileDesc/titleStmt/title* to select the title, and from there *../author* to select the author, and from there *@role* to select its role attribute
- By default it selects XML elements but you can also select *text()*, *comment()*, *processing-instruction()*

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

  <xsl:template match="TEI">
    <html><body>
        <xsl:apply-templates/>
    </body></html>
  </xsl:template>

  <xsl:template match="teiHeader">
    <h2>About</h2>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="text">
    <h2>The Text</h2>
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```

# XPath axis specifiers

- ancestor::
- ancestor-of-self::
- attribute:: (same as @)
- child:: (same as ./)
- descendant::
- descendant-or-self::
- following::
- following-sibling::
- parent:: (same as ../)
- preceding::
- preceding-sibling::
- self:: (same as .)

# One template for multiple elements

- Quite often, you want several elements in the source document to be displayed in exactly the same way in the output.
- Instead of creating a template for each, use the union operator, which is a pipe symbol, to specify multiple elements to be matched by a single template:

```
<xsl:template match="incipit | explicit | rubric">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>
```

# Multiple templates for the same element

- Sometimes you want different output for the instances of the same element, based on attribute values. That can be done with a predicate: an extra condition in square brackets.
- For example, to render TEI's *hi* element differently, based on its *rend* attribute:

```
<xsl:template match="hi[@rend='bold']">
  <b><xsl:apply-templates/></b>
</xsl:template>

<xsl:template match="hi[@rend='italic']">
  <i><xsl:apply-templates/></i>
</xsl:template>
```

# Filtering out things you don't want

- To prevent a TEI element, and everything within, from being published, create a blank template, e.g.:

```
<xsl:template match="publicationStmt"></xsl:template>
```

- To skip over a TEI element, but still allow its children to be published, create a template which only contains an apply-templates, e.g.:

```
<xsl:template match=“editionStmt">
  <xsl:apply-templates/>
</xsl:template>
```

# Re-ordering the output

- So far, we've only been mapping one-to-one from input to source, but *xsl:apply-templates* has a *select* clause to pick which child elements to process, and in what order.
- For example, to display the title of a work first, then the incipit, explicit and/or rubric, but nothing else:

```
<xsl:template match="msItem">
  <xsl:apply-templates select="title"/>
  <xsl:apply-templates select="incipit | explicit | rubric"/>
</xsl:template>
```

# Outputting something more than once

- Usually, you only want each instance of an element in the source document to appear once in the output. So, even if multiple XSL templates match, only one will be applied, and the rest ignored.
- But, for example, to display a table of contents, one way to do this would be to have two sets of templates: one to convert your TEI document into a TOC, and another to actually render the contents. This is achieved using *mode* attributes:

```
<xsl:template match="msItem[@xml:id]" mode="toc">
  <li>
    <a href="#{ @xml:id }"><xsl:value-of select="title"/></a>
    <ul>
      <xsl:apply-templates mode="toc"/>
    </ul>
  </li>
</xsl:template>
```

# Leaf nodes

- Usually, it is best to include the *xsl:apply-templates* instruction in every template. XSLT will work its way down the tree structure, and when it gets to something without child elements, only text, output the text.
- However, if you decide you only want the text, and ignore any other markup within (e.g. formatting in titles), then you can use *xsl:value-of* to just output the text:

```
<xsl:template match="title">
  <h3>
    <xsl:value-of select="string(.)"/>
  </h3>
</xsl:template>
```

- It can also be used to display attributes:

```
<xsl:value-of select="@n"/>
```

# Inserting text

- You can usually just add text within templates if you want to add a label, heading, or other text not in the source document
- Occasionally, when spacing is important, you might need to use *xsl:text*:

```
<xsl:template match="history">
  <h3>
    <xsl:text>History of the </xsl:text>
    <xsl:value-of select="string(../msIdentifier/idno)"/>
    <xsl:text> manuscript</xsl:text>
  </h3>
  <xsl:apply-templates/>
</xsl:template>
```

# Adding attributes

- To add an attribute to output, you can use *xsl:attribute:*

```
<xsl:template match="ref">
  <a>
    <xsl:attribute name="href">
      <xsl:value-of select="@target"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </a>
</xsl:template>
```

- However, in XSLT 2.0, there is a shortcut using curly brackets:

```
<a href="{ @target }">
  <xsl:apply-templates/>
</a>
```

# Making decisions within templates

- Use *xsl:if* statements to add something to the output based on a whether it passes a test.
- For example, to only output a HTML link if a persName has a key attribute:

```
<xsl:template match="persName">
  <xsl:apply-templates/>
  <xsl:if test="@key">
    <sup>
      <a href="https://www.viaf.org/viaf/{ @key }">VIAF</a>
    </sup>
  </xsl:if>
</xsl:template>
```

# Making more complicated decisions

- Use *xsl:choose* to handle multiple alternative cases
- For example, to handle different combinations of date attributes:

```
<xsl:template match="origDate">
  <xsl:choose>
    <xsl:when test="@notBefore and @notAfter">
      <xsl:value-of select="@notBefore"/> – <xsl:value-of select="@notAfter"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="@when|@notBefore|@notAfter"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

# Loops

- Use *xsl:for-each* to iterate over multiply-occurring elements and programmatically re-order them
- For example, to sort a revisionDesc chronologically:

```
<xsl:template match="revisionDesc">
  <xsl:for-each select="change">
    <xsl:sort select="@when" order="descending"/>
    <xsl:apply-templates/>
  </xsl:for-each>
</xsl:template>
```

# Aggregating / grouping

- Use *xsl:for-each-group* to aggregate lists
- For example, to group together each contributor's changes:

```
<xsl:template match="revisionDesc">
  <xsl:for-each-group select="change" group-by="@who">
    <b><xsl:value-of select="current-grouping-key()"/>:</b>
    <xsl:for-each select="current-group()">
      <xsl:sort select="@when" order="descending"/>
      <xsl:apply-templates/>
    </xsl:for-each>
  </xsl:for-each-group>
</xsl:template>
```

# More advanced topics

- Namespaces allow XML documents containing mixtures of schemas (e.g. EAD containing XLink, HTML containing SVG). XPath and XSLT handle these using prefixes
- Built-in XSLT and XPath functions provide mathematical operations and string manipulation (e.g. if you want to count the number of a certain element, or replace semi-colons with full stops)
- User-defined functions (added in XSLT 2.0) allows stylesheet developers to create their own functions
- Variables and sequences can hold numbers, string, or snippets of XML in memory, to use as lookup tables, or pass between functions. In XSLT 2.0 they can be typed for more resilient code