Murdoch
UNIVERSITY

XML Schema + XPath + XSLT

XML Basic

Lecture 6 (A)

Assignment One

- Assignment One is due next week (Week 7).
  All students should submit their assignment on LMS according to the instructions in the assignment question sheet AND host their working applications residing under their web home directory on `ceto.murdoch.edu.au`
  - Late submission penalties will apply - refer to the assignment question sheet

Murdoch UNIVERSITY

Learning Objectives

- Understand what an XML Schema is, and how it compares to a DTD
- Know the format and syntax of XML Schema as specified by W3C's XML Schema Recommendation

Learning Objectives

- In the scope of this unit:
  - XML is an important set of Internet technologies for use in many different areas
  - The ability to create new mark-up languages is a key factor in XML technologies
  - **XML Schema** is an alternative method (to DTD) used to define the syntax of XML documents
  - XML Schema is preferred to the DTD in many workplaces today, and is therefore a critical technology to allow us to create new mark-up languages

XML Schema

- In last week's lectures, we discussed how W3C's XML 1.0 Recommendation specifies:
  - The syntax of well-formed XML documents, and
  - The DTD to validate XML documents
- XML Schema specifies the structure of XML documents (just like the DTD), and also has many different features compared to DTDs
  - http://www.w3.org/XML/Schema

Why Have XML Schema?

- Some people became dissatisfied with DTDs:
  - DTDs use a different syntax to define elements and attributes compared to XML documents, so they cannot be checked by standard XML parsers, and they are not easy to transform using XSLT (more on XSLT later)
  - DTDs have very limited datatype capabilities:
    - For example, they can not specify that a piece of data must be a number between 0 and 100

A Successor to DTD

- XML Schemas are:
  - **Extensible** to future additions
  - **Richer** and **more powerful** than DTDs
  - **Written in XML**
  - **Support datatype definitions** better than DTDs
  - **Support namespaces**

Some Features of XML Schema

- XML Schemas provide some advancement over DTDs:
  - They are written in the XML syntax, so all the tools for XML (like style sheets, parsers and processors which we will be discussing in future lectures) can be used on the Schema
  - They provide enhanced datatype capabilities:
    - Over 44 built-in datatypes in Schemas compared to 10 built-in datatypes in DTDs
    - You can create your own datatypes with XML Schemas

Some Features of XML Schema

- XML Schemas provide some advancement over DTDs (cont):
  - They provide more powerful `content_models`, so we can specify more interesting contents for XML documents (and thus derive more interesting mark-up languages)
  - They are eXtensible because they are written in XML!

Purpose of an XML Schema

- To define **elements** and **attributes** that can appear in an XML document
- To define which element is the **root element** and which elements are **child elements**
- To define the **order** of child elements
- To define the **number** of child elements

Purpose of an XML Schema

- To define whether an element is an **empty element or can include text**
- To define **datatypes** for elements and attributes
- To define **default** and **fixed values** for elements and attributes

Schemas Support DataTypes

- XML Schemas make it easier to:
  - Describe allowable document content
  - Validate the correctness of data
  - Work with data from a database
  - Define data facets (restrictions on data)
  - Define data patterns (data formats)
  - Convert data between different datatypes

The XML Schema Specifications

- W3C's XML Schema 1.0 specifications comes in 3 parts:
  - **Part 0: Primer**
    - Just a tutorial - not really part of the specifications
  - **Part 1: Structures**
    - How to define the basic structure of XML documents using XML Schema. Eg:"this element contains these elements, which contains these other elements, etc.."
  - **Part 2: Datatypes**
    - How to define datatypes of elements and attributes used in an XML document. Eg: "this element shall hold an integer within a range of 0 to 10, etc..."

A Simple XML Document (note.xml)


```
<?xml version="1.0"?>
<note>
  <to>Eric</to>
  <from>Ted</from>
  <heading>Advice</heading>
  <body>Watch out for spiders!</body>
</note>
```

External DTD (note.dtd)

```
<!ELEMENT note (to ,from ,heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

- The first `content_model` defines the `note` element as having 4 child elements
- The other four `content_models` define the child elements to be of the type `#PCDATA`

Schema (note.xsd)


```
<?xml version="1.0"?> <!-- NOTE: xml declaration -->
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3schools.com"
   targetNamespace="http://www.w3schools.com"
   elementFormDefault="qualified">
    <xsd:element name="note">
      <xsd:complexType>
          <xsd:sequence>
          <xsd:element name="to" type="xsd:string"/>
          <xsd:element name="from" type="xsd:string"/>
          <xsd:element name="heading" type="xsd:string"/>
          <xsd:element name="body" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Murdoch UNIVERSITY

Schema (note.xsd)


```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

- The above line indicates that the elements and data types used in the schema comply with the correct syntax rules as specified at the official Schema namespace: "http://www.w3.org/2001/XMLSchema"
- It also specifies that the elements and data types that are verified at that namespace should be prefixed with **xsd**
  - Recall xsd means XML Schema Definition

Schema (note.xsd)


```
xmlns="http://www.w3schools.com"
```

- The above line indicates that the default namespace is http://www.w3schools.com.

```
targetNamespace="http://www.w3schools.com"
```

- The above line indicates that "http://www.w3schools.com" is also the target namespace.
- All elements defined in the schema belong to this namespace.

Schema (`note.xsd`)


```
elementFormDefault="qualified">
```

- The above line indicates that any elements used by an XML instance document , must be namespace qualified (i.e., qualified against what was declared in this schema)

Referencing Our Schema in an XML Instance Document


```
<?xml version="1.0"?>
<note xmlns="http://www.w3schools.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3schools.com note.xsd">

<note>
  <to>Eric</to>
  <from>Ted</from>
  <heading>Advice</heading>
  <body>Watch out for spiders!</body>
</note>
```

Referencing Our Schema in an XML Instance Document


```
<note xmlns="http://www.w3schools.com"
```

- The above line specifies the default namespace declaration; its declaration tells the schema-validator that all the elements used in this XML document are valid according to the namespace http://www.w3schools.com

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                                    instance"
```

- The above line specifies the XML Schema Instance namespace; prefixed by **xsi.** This is needed when using `schemaLocation` from that namespace.

Murdoch UNIVERSITY

Referencing Our Schema in an XML Instance Document

- Once you have the XML Schema Instance namespace available you can use the `schemaLocation` attribute
- This attribute has two values, separated by a space
  - The 1st value is the target namespace
  - The 2nd value is the location of the XML schema to use with that namespace – in this case, the **note.xsd** schema file

```
xsi:schemaLocation="http://www.w3schools.com note.xsd">
```

Schema Elements

- Just as in DTDs, the primary purpose of XML Schema is to declare elements
- Elements are the basic components or building blocks of XML documents (refer to lecture on The XML Document again)
- To declare elements in a schema, XML Schema has an element called “element”
- Note in our example, that the root element is also the name of the schema file **note**.xsd:

```
<xsd:element name="note"> … </xsd:element>
```

Schema Elements

- An element has a **name** and is associated with a **data type**
- Elements can be declared as:
  - Built-in types
    - eg: `type="xsd:string"`
  - Simple types
    - `<simpleType></simpleType>`
  - Complex types
    - `<complexType></complexType>`

Some Useful Built-in Datatypes

- Primitive types:
  - string, boolean, number, float, double, date, time, gYear, gMonth, gDay, hexBinary, AnyURI, ...
- Derived types:
  - normalizedString, integer, short, long, negativeInteger, positiveInteger, nonPositiveInteger, ...

Simple vs Complex Types

- Element with **simple types** can have character data content but no child elements or attributes
- Some examples:

```
<size>10</size>
<comment>Help me if you can!</comment>
<availableSizes>10 large 2</availableSizes>
```

Simple vs Complex Types

- Element with **complex types** can have child elements (simple or complex) or attributes, as well as character data content

```
<size system="AUS-DRESS">10</size>
<comment>Please <b>HELP</b> me?</comment>
<availableSizes>
  <size>10</size>
  <size>12</size>
</availableSizes>
```

Ways of Declaring Elements

- Formal declaration using attributes:

```
<xsd:element name="name" type="type"
             minOccurs="int" maxOccurs="int" />
```

  - `minOccurs` and `maxOccurs` specify how many times this element can exist in the XML document; they both have default values of 1

- Examples:

```
<xsd:element name="Title" type="xsd:string" />
```

```
<xsd:element name="Result" type="xsd:float"
                           maxOccurs="unbounded"/>
```

```
<xsd:element name="DateOfBirth"
                    type="xsd:date" minOccurs="0" />
```

Ways of Declaring Elements: Containing Sub-Elements

- Formal declaration using `complexType`:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
   <xsd:complexType>
      …
   </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="Staff" minOccurs="1" maxOccurs="50">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Surname" type="xsd:string" />
      <xsd:element name="Age" type="xsd:positiveInterger"
                              minOccurs="0" />
      <xsd:element name="Address" type="xsd:string"
                                  minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Example:

Murdoch UNIVERSITY

Ways of Declaring Elements: Containing Sub-Elements

- An XML document based on previous schema:

```
<Staff>
  <Surname>Smith</Surname>
</Staff>
<Staff>
  <Surname>Jones</Surname>
  <Age>23</Age>
</Staff>
<Staff>
  <Surname>Santa Clause</Surname>
  <Age>200</Age>
  <Address>123, Ice Palace, North Pole.</Address>
</Staff>
```

Facets/Restrictions

- Restrictions are used to define acceptable values for XML elements or attributes
- Restrictions on XML **elements** are called **facets**
- “minInclusive”, “maxInclusive” and “pattern” in the following examples are example **facets**
- We extend a base type by changing its facet values

Facets: Extending a Simple Type to Define New Types

- Formal declaration:

```
<xsd:element name="name" minOccurs="int" maxOccurs="int">
  <xsd:simpleType>
    <xsd:restriction base="type">
      …
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Example:

```
<xsd:element name="Age" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:nonNegativeInteger">
       <xsd:minInclusive value="0"/>
       <xsd:maxInclusive value="150"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Declare New Types: Extending Existing Simple Types

- This defines a new type called "ISBNType", extended from the "string" type

```
<xsd:simpleType name="ISBNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{1}-\d{5}-\d{3}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{3}-\d{5}-\d{1}"/>
    <xsd:pattern value="\d{1}-\d{2}-\d{6}-\d{1}"/>
  </xsd:restriction>
</xsd:simpleType>
```

  - Borrowed from http://www.xfront.com/xml-schema.html

Declare New Types: Extending Existing Simple Types

- 1st Pattern: 1 digit followed by a dash followed by 5 digits followed by another dash followed by 3 digits followed by another dash followed by 1 more digit
- 2nd Pattern: 1 digit followed by a dash followed by 3 digits followed by another dash followed by 5 digits followed by another dash followed by 1 more digit
- 3rd Pattern: 1 digit followed by a dash followed by 2 digits followed by another dash followed by 6 digits followed by another dash followed by 1 more digit

Restrictions

- Restrictions can be placed on:
  - A value
  - A set of values
  - A series of values
  - Whitespace characters
  - Length

Ways of Declaring Elements: Reference to Another Element

- Formal declaration:

```
<xsd:element ref="name" minOccurs="int"
                        maxOccurs="int"/>
```

- Examples:

```
<xsd:element name="Surname" type="xsd:string" />
<xsd:element name="Staff" minOccurs="1" maxOccurs="50">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="Surname"/>
        ...
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Declaring Attributes

- We can use the "attribute" element in Schemas to add attributes to elements in our new language

```
<xsd:element name="Student">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Surname" type="xsd:string"/>
        ...
    </xsd:sequence>
    <xsd:attribute name="Workrate" type="xsd:string"
                                   use="optional"/>
  </xsd:complexType>
</xsd:element>
```

Declaring Attributes: Referencing Attributes

- We can also make a reference to an "attribute" element in Schemas

```
<xsd:attribute name="Workrate" type="xsd:string"
                                use="optional"/>
...
<xsd:element name="Student">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Surname" type="xsd:string"/>
        ...
    </xsd:sequence>
    <xsd:attribute ref="Workrate"/>
  </xsd:complexType>
</xsd:element>
```

Example XML Document Content from the Previous Schemas

- An XML document based on our previous schema:

```
<Student workrate="space cadet">
  <Surname>Smith</Surname>
</Student>
<Student workrate="high achiever">
  <Surname>Jones</Surname>
</Student>
<Student>
  <Surname>MacPherson</Surname>
</Student>
```

Annotations

- Since an XML Schema is supposed to be self-describing, there is an “annotation” element which can be used to annotate the Schema

```
...
<xsd:annotation>
  <xsd:documentation>
    This schema is to specify information about the
    students within an academic institution.
    This Schema was developed by ...
  </xsd:documentation>
</xsd:annotation>
...
```

Annotations: Another Example

```
...
<xsd:annotation>
  <xsd:documentation>
    The following "StudentID" element is extracted from
    ...
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="StudentID">
    ...
</xsd:element>
```

Namespaces in Schemas

- XML Schema elements are associated with, or identified by, designated **namespace/s**
- The **targetNamespace** can be used as a root element attribute to indicate the namespace being described by the Schema

Namespaces in Schemas

- XML Schemas support Namespaces naturally
- Eg: attaching a namespace to a Schema

uni.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:"http://university.edu.au"
            targetNamespace="http://university.edu.au">
...
```

course.xml

```
<?xml version="1.0"?>
<uni:course xmlns:uni="http://university.edu.au"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://university.edu.au/ uni.xsd">
    <uni:name>...</uni:name>
    <uni:unit>...</uni:unit>
...
```

Example Schema

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://course.murdoch.edu.au"
            xmlns:"http://course.murdoch.edu.au">

    <xsd:element name="course">
        <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="name" />
              <xsd:element ref="unit" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="unit">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="title" />
                <xsd:element ref="lecturer" minOccurs="0" maxOccurs="unbounded" />
                <xsd:element ref="tutor" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
```

Declaring the “course” root element

"name" and "unit" are from the default namespace

Declaring the “unit” element

Example Schema

```
    <xsd:element name="lecturer">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="surname" />
                <xsd:element ref="othernames" minOccurs="0" maxOccurs="1" />
                <xsd:element ref="email" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="tutor">
        <xsd:complexType>
            <xsd:sequence>
                ...
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="title" type="xsd:string" />
    <xsd:element name="surname" type="xsd:string" />
    <xsd:element name="othernames" type="xsd:string" />
    <xsd:element name="email" type="xsd:string" />
</xsd:schema>
```

Declaring the "lecturer" element

Declaring the "tutor" element

Declaring the reference elements as simple string types. Note the references may be positioned at the beginning of the Schema (just after the namespace declarations) or at the end, as shown in this example.

Example XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<course xmlns ="http://course.murdoch.edu.au"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://course.murdoch.edu.au course.xsd">
    <name>Bachelor of Science – Mobile and Web Application
            Development</name>
    <unit>
        <title>ICT375 Advanced Web Programming</title>
        <lecturer>
            <surname>Xie</surname>
            <othernames>Hong</othernames>
        </lecturer>
    </unit>
    <unit>
        <title>ICT283 Data Structures And Abstraction</title>
        <lecturer>
            <surname>Rai</surname>
            <othernames>Shri</othernames>
            <email>s.rai@murdoch.edu.au</email>
        </lecturer>
    </unit>
</course>
```

Root element refers to the XML Schema

Similar to the XML in the last lecture

Murdoch UNIVERSITY

XML Schema Validating Parsers

- Like DTDs, there are some parsers that validate XML documents based on their XML Schemas
- See examples:
  - http://www.w3.org/XML/Schema#Tools

XML Schema

- The practical lab work involving XML Schema is at a fairly basic level
  - However, you may wish to explore more advanced usage of XML Schemas
- XML Schema and other methods for defining document types are very important technologies, and you will need to be familiar with them if you do future work in XML

References

- Online tutorials:
  - http://www.w3.org/TR/xmlschema-0/
- Available for download in ZIP file
  - http://www.xfront.com/xml-schema.html
- Some example XML Schemas:
  - http://www.w3.org/XML/Schema#Usage

XPath: Navigating The XML Tree

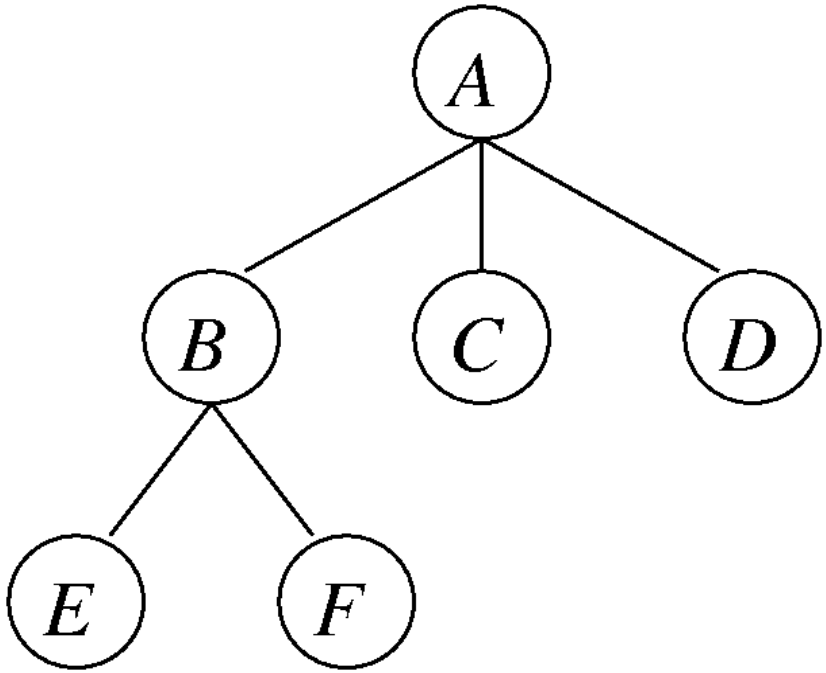
Lecture 6 (B)

Learning Objectives

- XML is an important set of Internet technologies for use in different solutions in different areas
- When processing XML documents, data stored in the documents will need to be retrieved
- XPath is a standard method for data retrieval from XML documents
- So there is a need to understand and be able to use simple **XPath** expressions

XML as Trees

- An XML document can conceptually be represented by a **tree structure**
  - nodes (XML elements), edges (link nodes)
  - root node,
    intermediate nodes,
    leaf (end) nodes
  - child, parent
  - sibling (ordered),
    ancestor, descendant



Introduction to XPath

- XPath is an official W3C Recommendation
  - http://www.w3.org/TR/xpath
- It is used to express the path to one or a group of nodes in an XML tree
- As such, it is a language for addressing parts of an XML document
- It is designed to be used by other XML technologies such as XSLT

Introduction to XPath

- Though the primary purpose of XPath is to address parts of an XML document, it also provides basic facilities for manipulation of strings, numbers and booleans
- XPath uses a compact, non-XML syntax
  - This syntax facilitates the use of XPath within URIs and XML attribute values

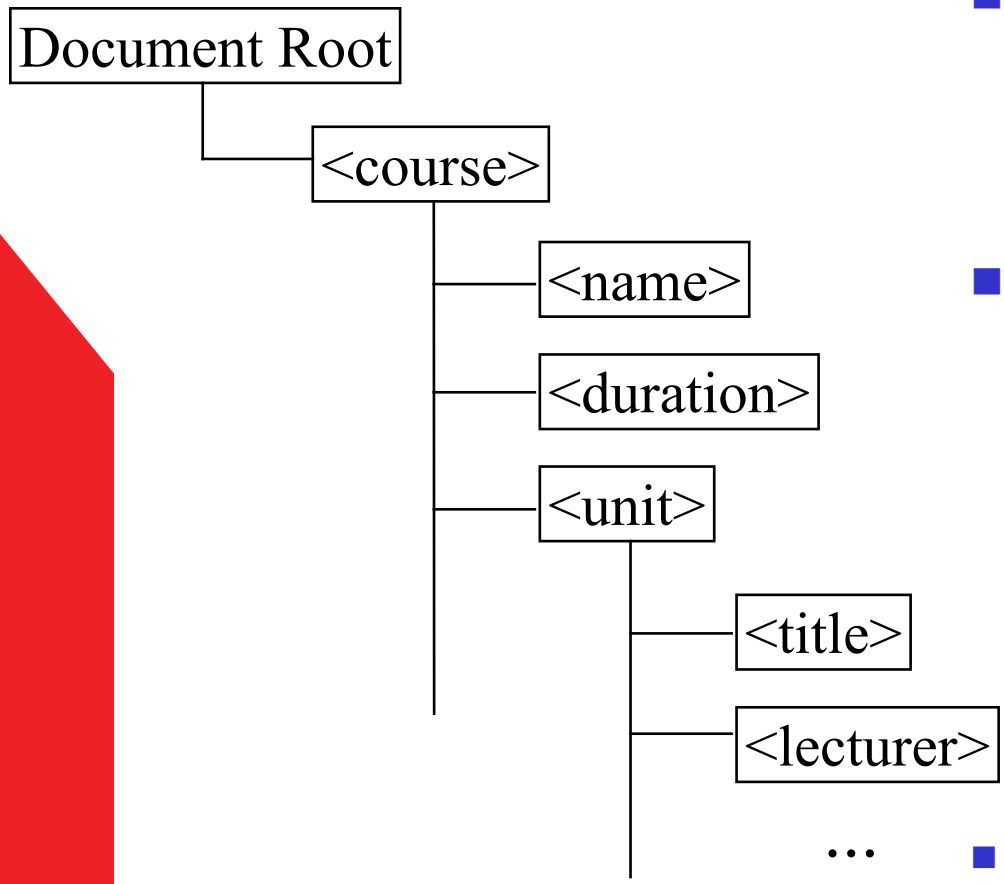
Introduction to XPath

- XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax
- XPath gets its name from its use of a path notation (similar to URLs and file systems) for navigating through the hierarchical structure of an XML document

Example XML Document


```
<?xml version="1.0"?>
<course>
     <name>Bachelor of Science – Mobile and Web Application Development </name>
     <duration>3 years</duration>
     <unit>
          <title>ICT375 Advanced Web Programming</title>
          <lecturer>
               <surname language="English">Xie</surname>
               <othernames language="English">Hong</othernames>
               <email>H.Xie@murdoch.edu.au</email>
          </lecturer>
     </unit>
     <unit>
          <title>ICT283 Data Structures And Abstraction</title>
          <lecturer>
               <surname>Rai</surname>
               <othernames>Shri</othernames>
               <email>s.rai@murdoch.edu.au</email>
          </lecturer>
     </unit>
</course>
```

Nodes in our Example Document



- We refer to the document root as “/” (**Note**: the document root is **not** the root element)
- We refer to the rest of the tree using the same notation as a URL or a directory path in a UNIX file system. Eg:

  “/course”

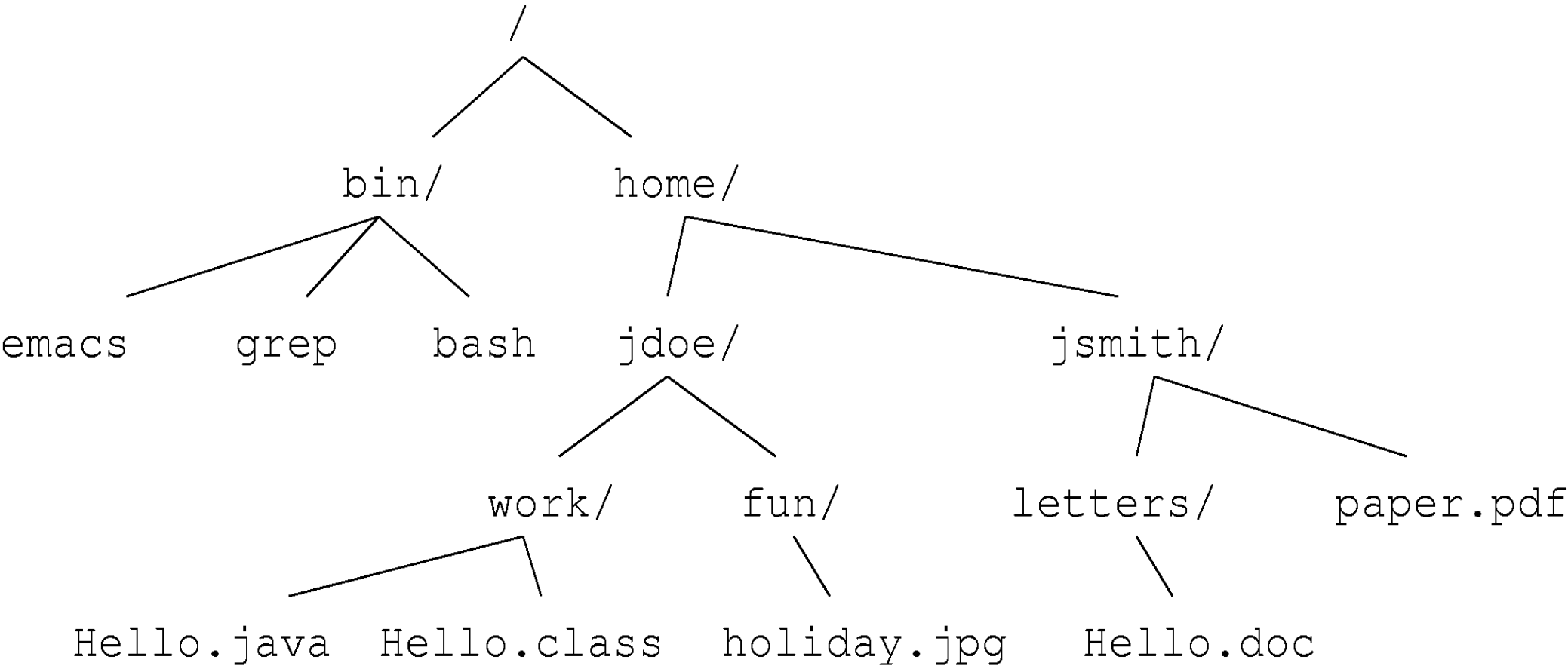
  “/course/unit/lecturer”
- These are called **location paths** in XSLT

Murdoch UNIVERSITY

An Analogy: URL And File Systems

http://ceto.murdoch.edu.au/~jsmith/letters/

-> cd /home/jsmith/letters/



XPath Evalution: Context

- The *context* of an XPath evaluation consists of:
  - The **context node** (current node in an XML tree)
  - Two integers > 0 from evaluating the step:
    - context **size** (number of nodes in node-set)
    - context **position** (index of context node in node-set)
  - A set of **variable bindings,** and a set of **namespace declarations**
  - A **function library**

Context Node

- At any point in time, XPath will consider one of the nodes in the tree to be the **context node**
  - i.e., “the node at which it is currently processing”
- XPath expressions are evaluated based on the current context node

XPath Evalution: Context

- Navigation ‘propagates’ the context
  - i.e., evaluation of a **step** yields a new **context state**
- The application determines the **initial context**
- If the XPath expression starts with ‘/’ then
  - The initial context node is the document root (not the root element)
  - The initial position and size are 1

XPath Expressions

- XPath language fulfils the need for a flexible notation for **pointing into** and **navigating around** XML trees
- It is a basic technology that is **widely used** because it allows for:
  - *Uniqueness* and *scope* in XML Schema
  - *Pattern matching* on a *selection* in XSLT
  - *Computations* on values in XSLT
  - *Relations* in XLink and XPointer

Nodes in XPath

- XPath refers to anything in the source tree as a **node**
  - Source tree is the tree representation of an XML document
- A node may consist of:
  - Elements
  - Attributes
  - Processing Instructions
  - Etc.

Nodes in XML Trees

- **Text nodes**:
  - Leaf nodes that carry the actual contents
- **Element nodes**:
  - Define hierarchical logical groupings of content
  - Each has a **name**
- **Attribute nodes**:
  - Unordered
  - Each is associated with an element node, and has a **name** and a **value**

Nodes in XML Trees

- **Comment nodes**:
  - Ignorable meta-information
- **Processing instructions**:
  - Instructions to specific processes
  - Each has a **target** and a **value**
- **Root node**:
  - Every XML tree has one root node that represents the entire source tree

Textual Representation

- **Text nodes**: written as the text they carry
- **Element nodes**: start-end tags
  - `<blah ...> ... </blah>`
  - short-hand notation for empty elements: `<blah />`
- **Attribute nodes**: associated with an element node
  - name="value" in start tags
- **Comment nodes**: `<!-- blah -->`
- **Processing instructions**: `<?target value?>`
- **Root nodes**: implicit

Location Paths

- A **location path** evaluates to a **sequence** of nodes in a given XML source tree
- The sequence is **sorted** in document order
- The sequence will **never** contain **duplicates**

Location Steps

- The location path is built as a sequence of **location steps**, separated by a '/'
- A **location step** consists of:
  - An **axis**
  - A **node test**
  - Zero or more **predicates** (path expressions)

```
axis :: nodetest [Exp1] [Exp2] …
```

Location Steps

- The axis, node test, and predicates may be viewed as increasingly detailed descriptions of the sequence of nodes to which the step should lead
- In the following example, `child` is the axis, `section` is the node test, and the expression `position()<6` is a predicate

Location Steps

Example:

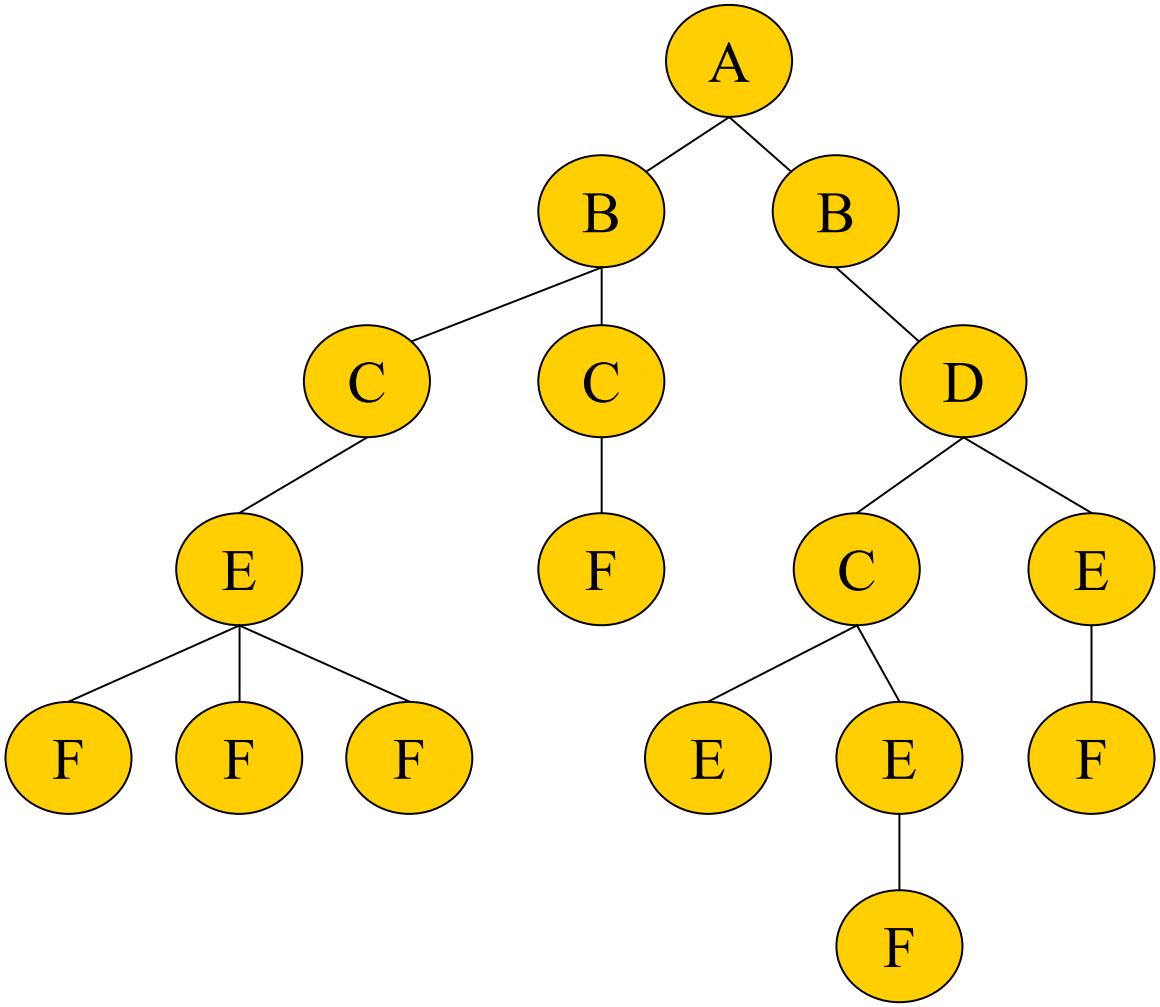
```
child::section[position()<6]
    /descendant::cite
    /attribute::href
```

Result: Selects all `href` attributes in the `cite` elements in the first 5 `section`s of an XML document

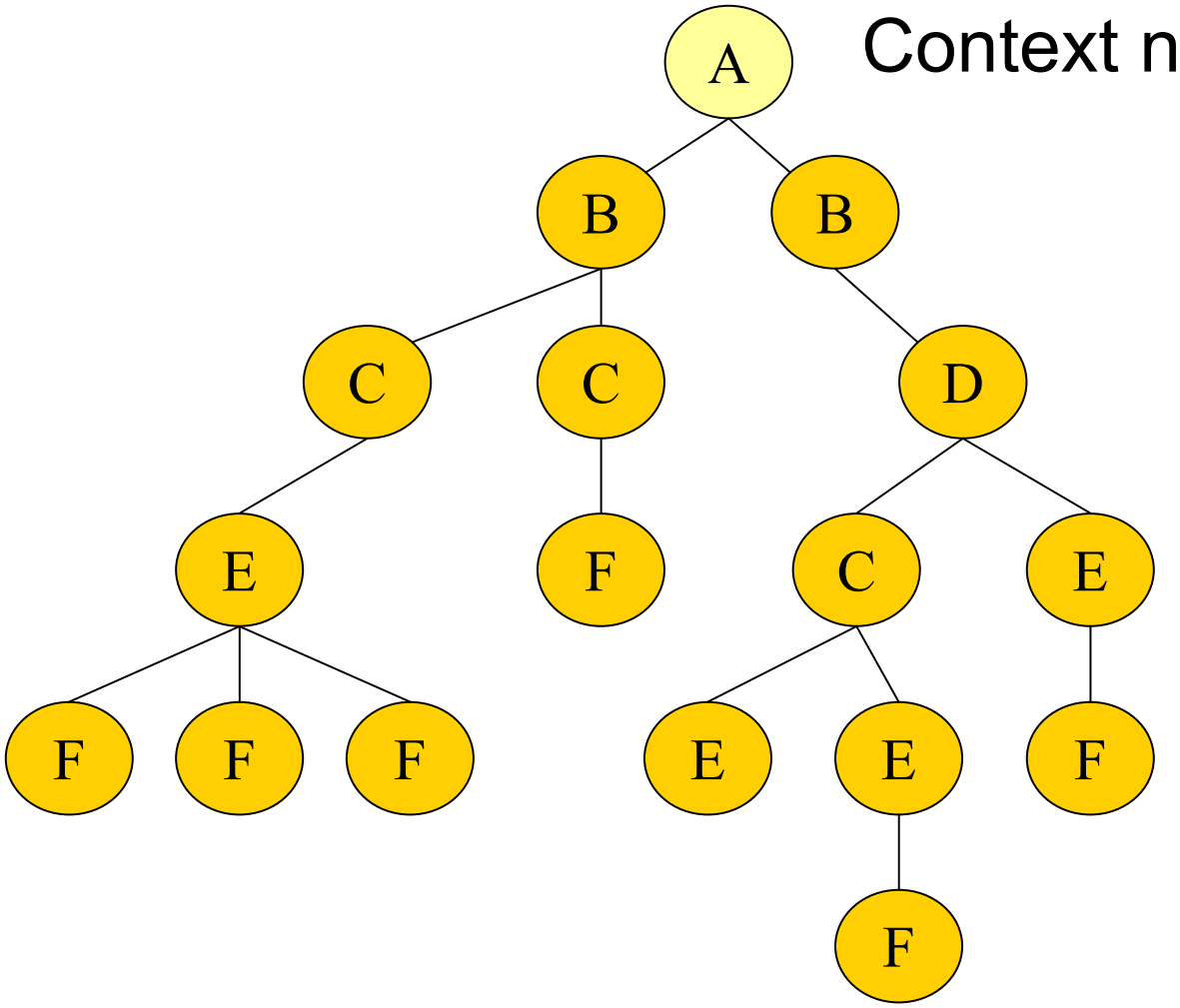
Evaluating a Location Path

- The location step starts at a **context node** and evaluates to a sequence of nodes
  - That is, starting from a context node, a location path returns a **node-set**
- Each node in the **node-set** becomes in turn the context node for evaluating the next **step**
- The path applies each step in turn

An Example



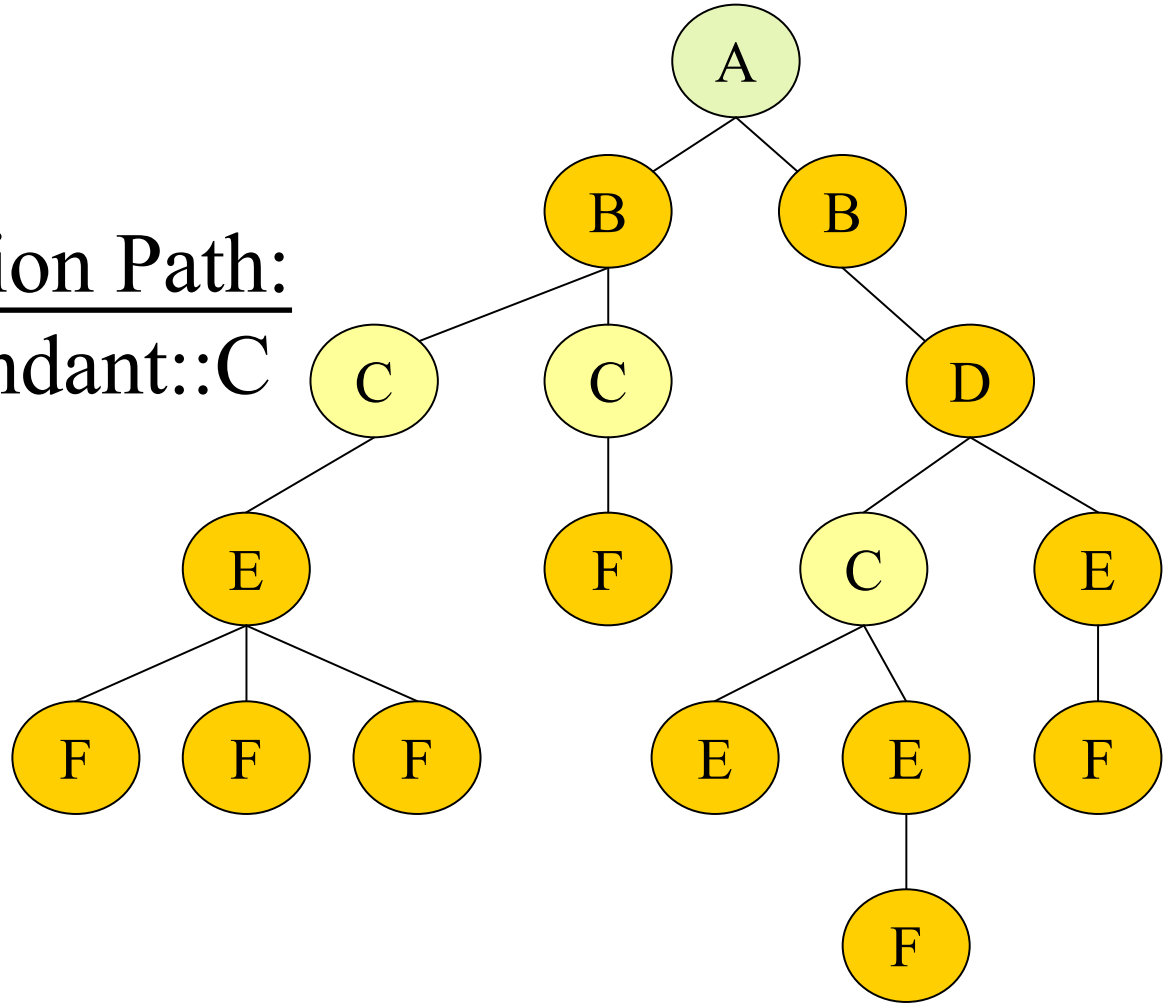
Example: Starting At Node A



Example: Descendant

Location Path:

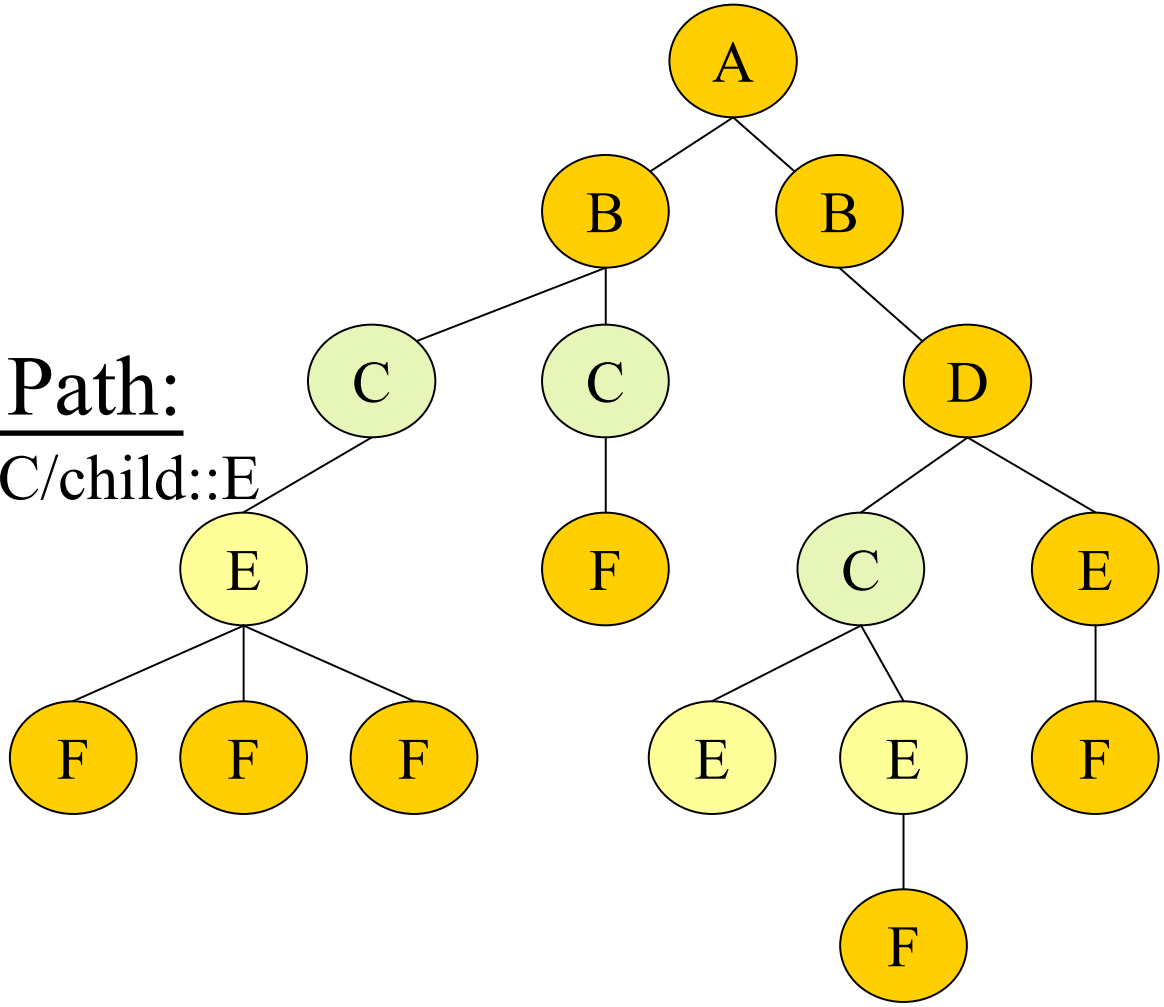
descendant::C



Example: Child

Location Path:

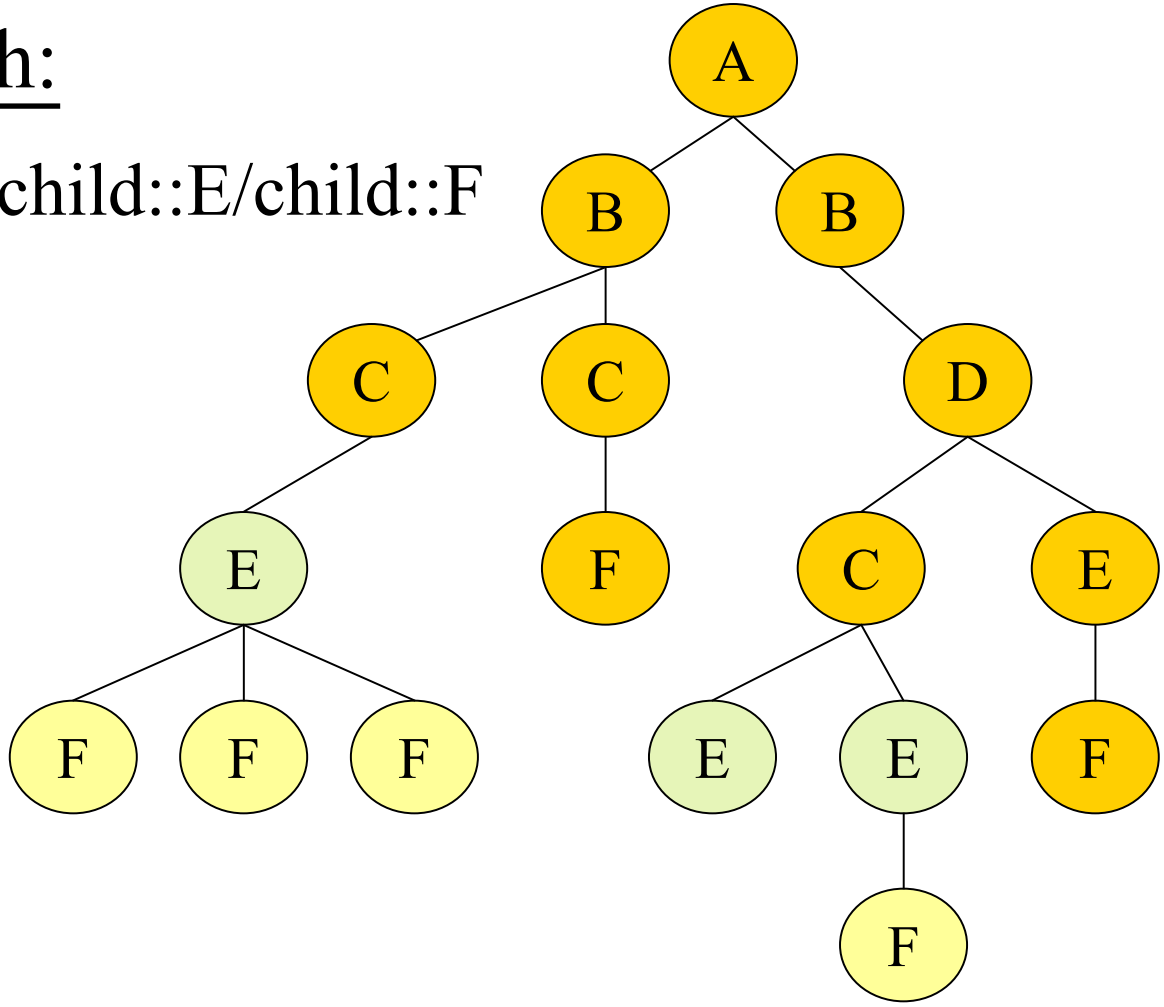
descendant::C/child::E



Example: Child

Location Path:

descendant::C/child::E/child::F



Murdoch UNIVERSITY

Axes

- An axis indicates where in the tree (with respect to the context node) to search for selected nodes

Axes

- XPath supports 12 different axes:
  - Child: the children of the context node (not including attribute nodes)
  - Descendant: the descendants of the context node (not including attribute nodes)
  - Parent: the unique parent of the context node (empty sequence if context node is the root node)
  - Ancestor: all ancestors of the context node, from parent to root node
  - Following-sibling: the right-hand siblings of the context node (empty sequence for attribute nodes)
  - Preceding-sibling: the left-hand siblings of the context node (empty sequence for attribute nodes)

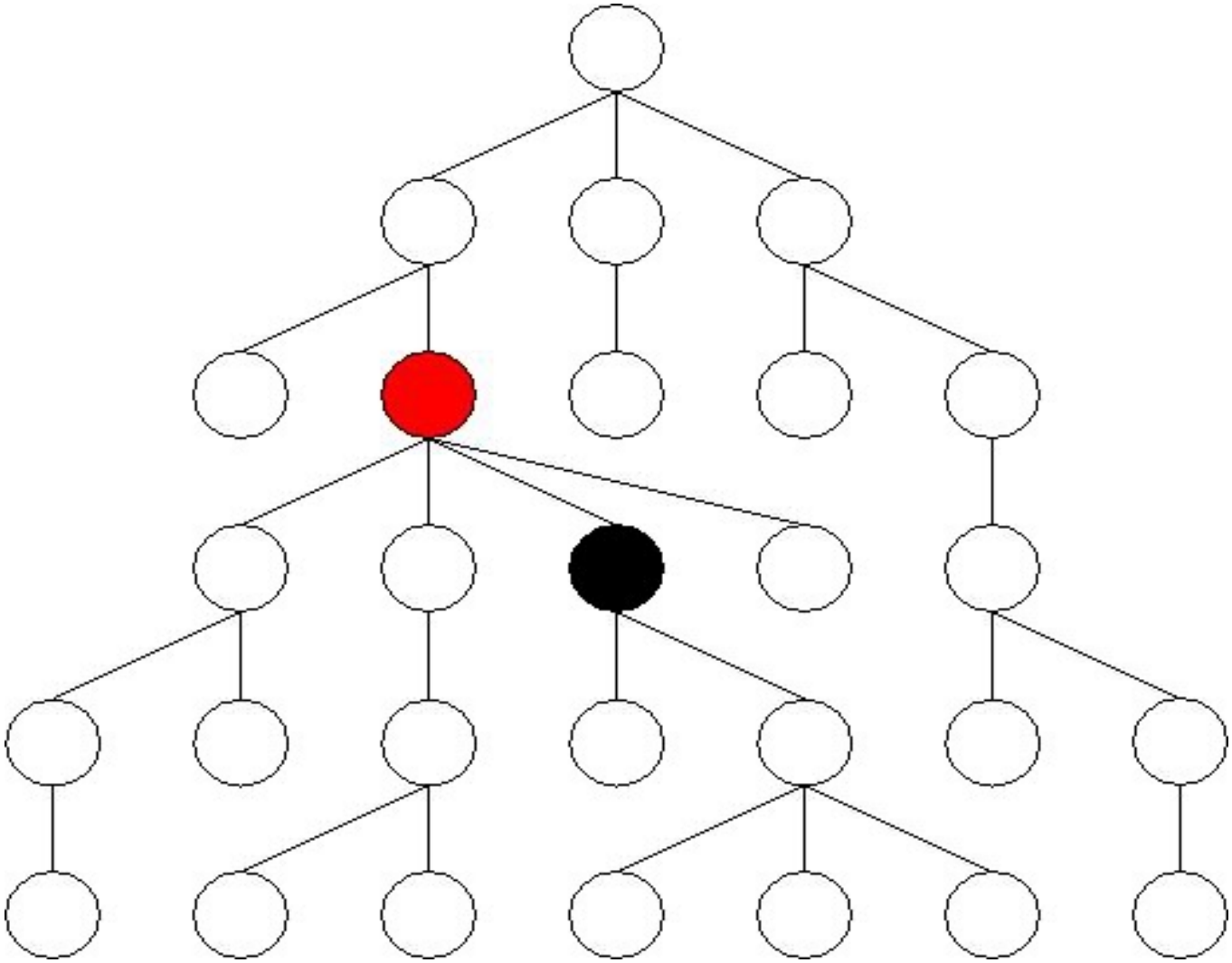
Axes

- XPath supports 12 different axes:
  - Following: all nodes appearing later in the document than the context node, excluding descendants
  - Preceding: all nodes appearing earlier in the document than the context node, excluding ancestors
  - Self: the context node itself
  - Attribute: all attribute nodes of the context node
  - Descendant-or-self: concatenation of self and descendant sequences
  - Ancestor-or-self: concatenation of self and ancestor sequences

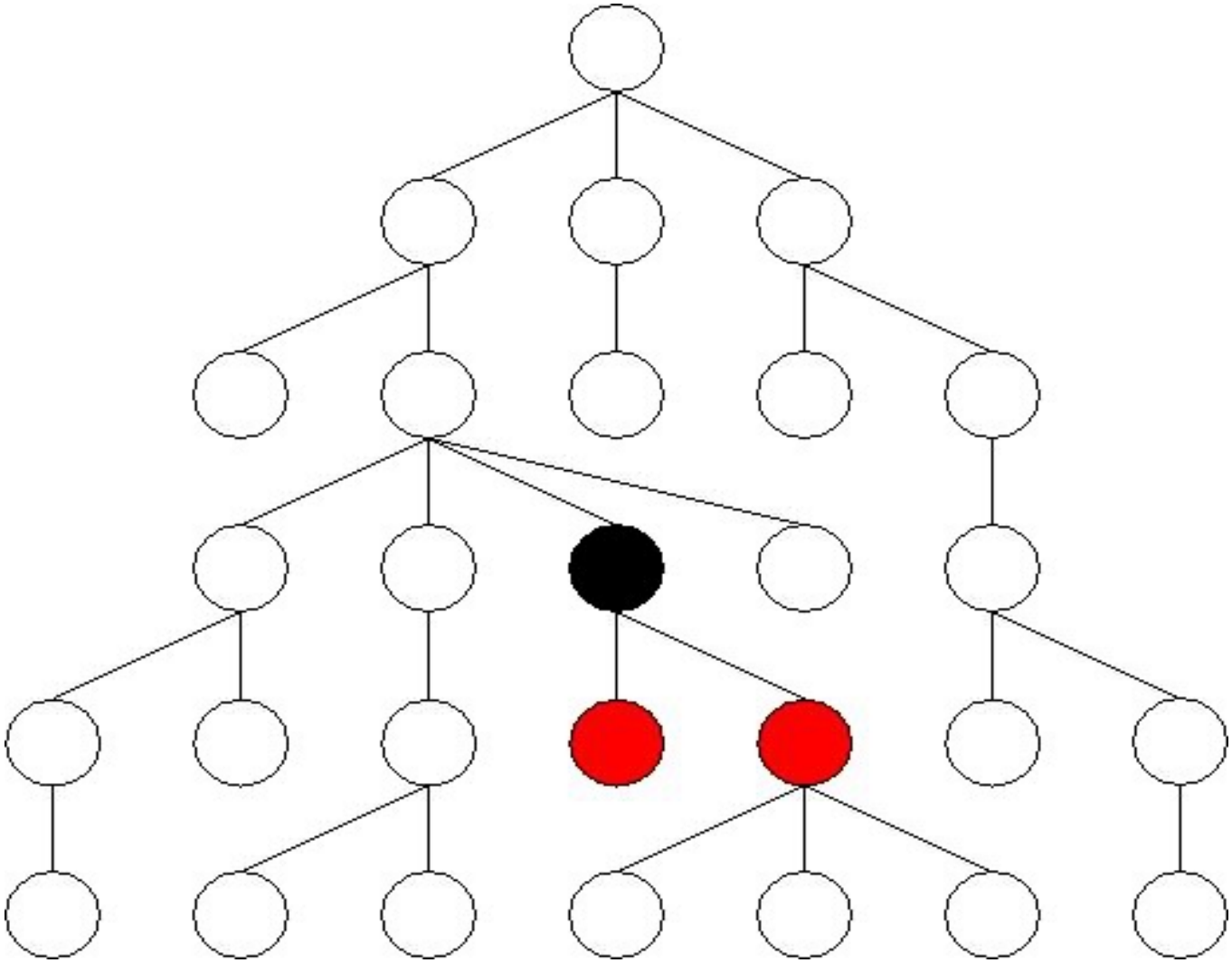
Axis Directions

- Each axis has a **direction**, with respect to document ordering
- **Forwards** means document order:
  - `Child, descendant, following-sibling, following, self, descendant-or-self`
- **Backwards** means reverse document order:
  - `Parent, ancestor, preceding-sibling, preceding, ancestor-or-self`
- Direction determined by associated element:
  - `Attribute`

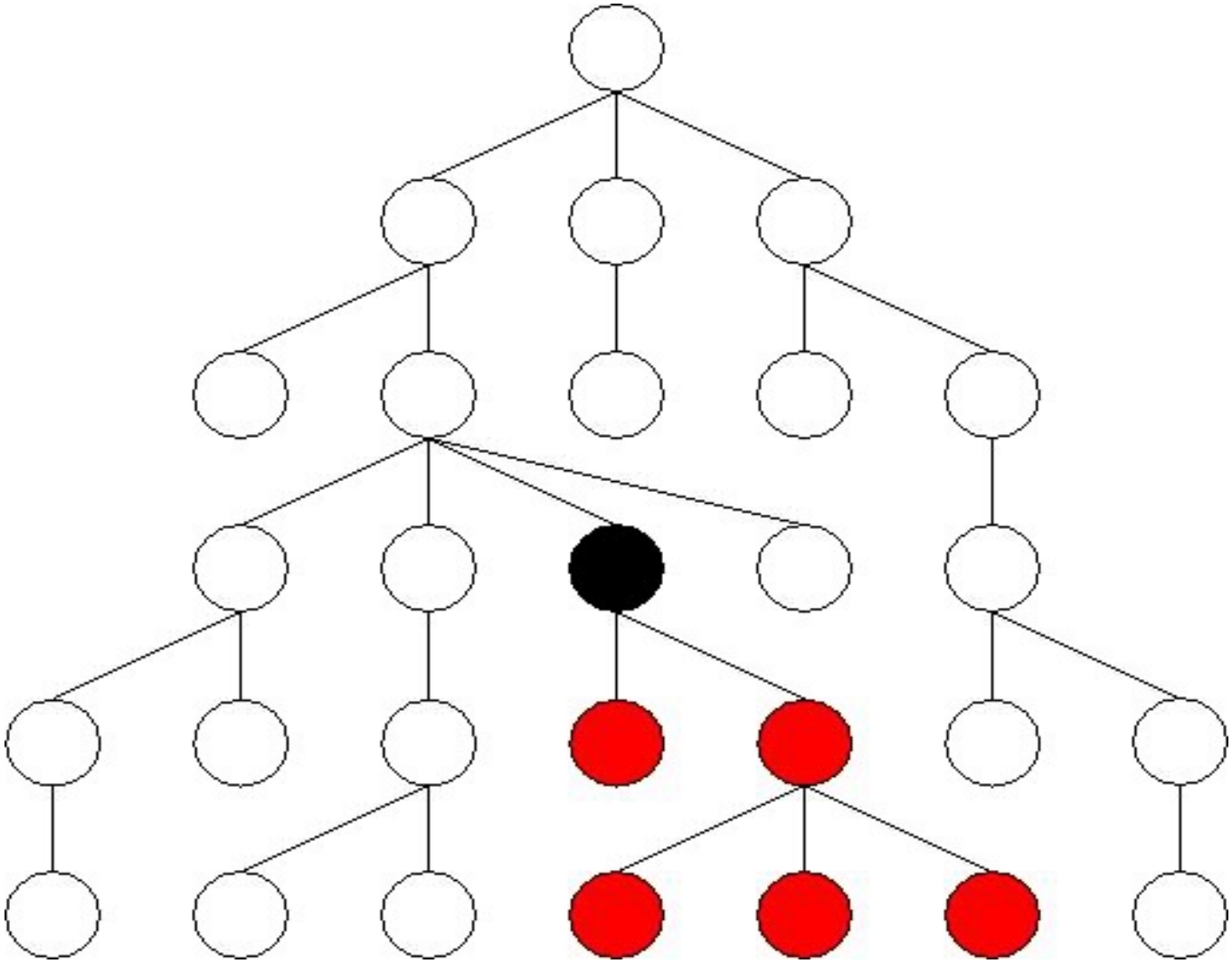
The Parent Axis



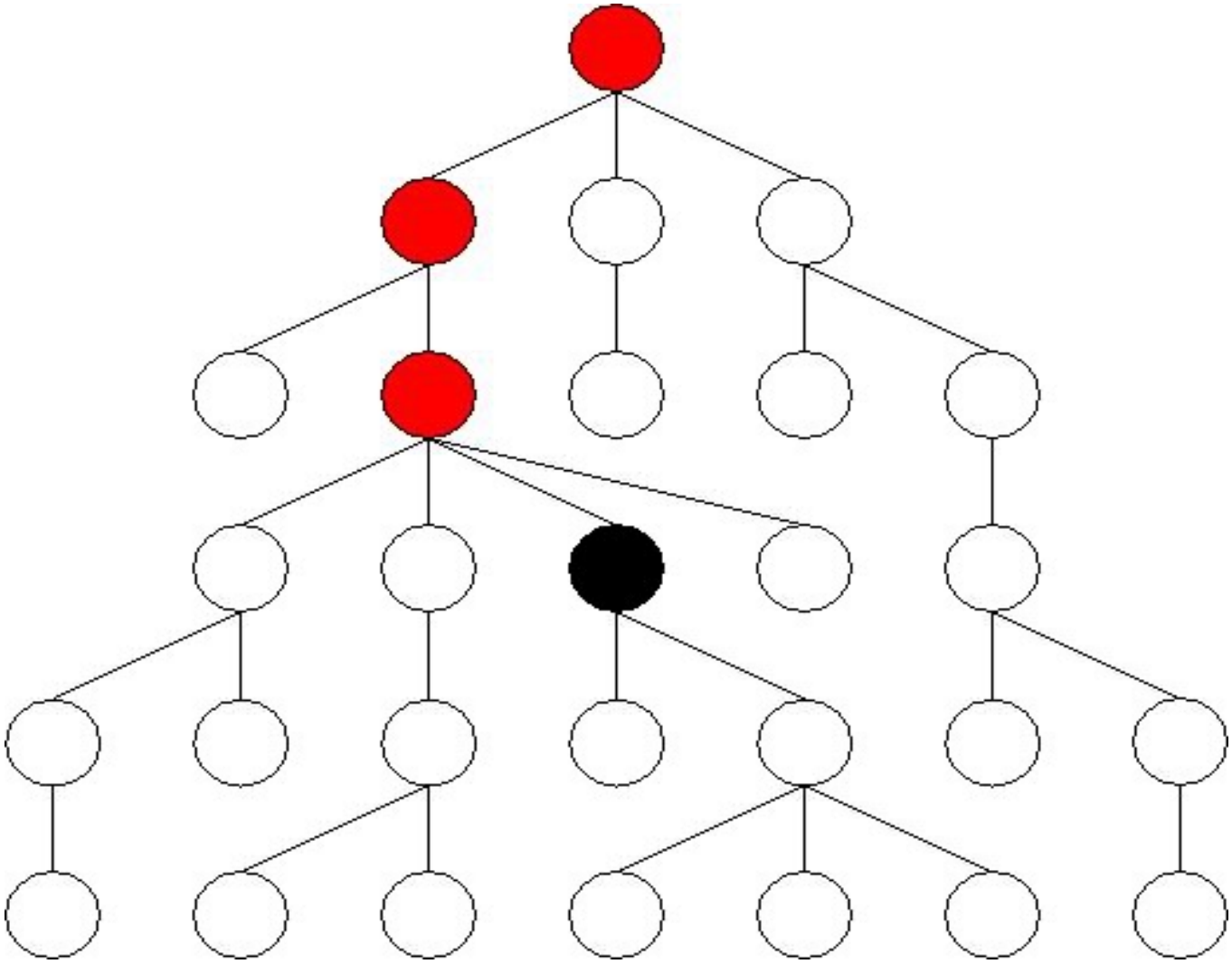
The Child Axis



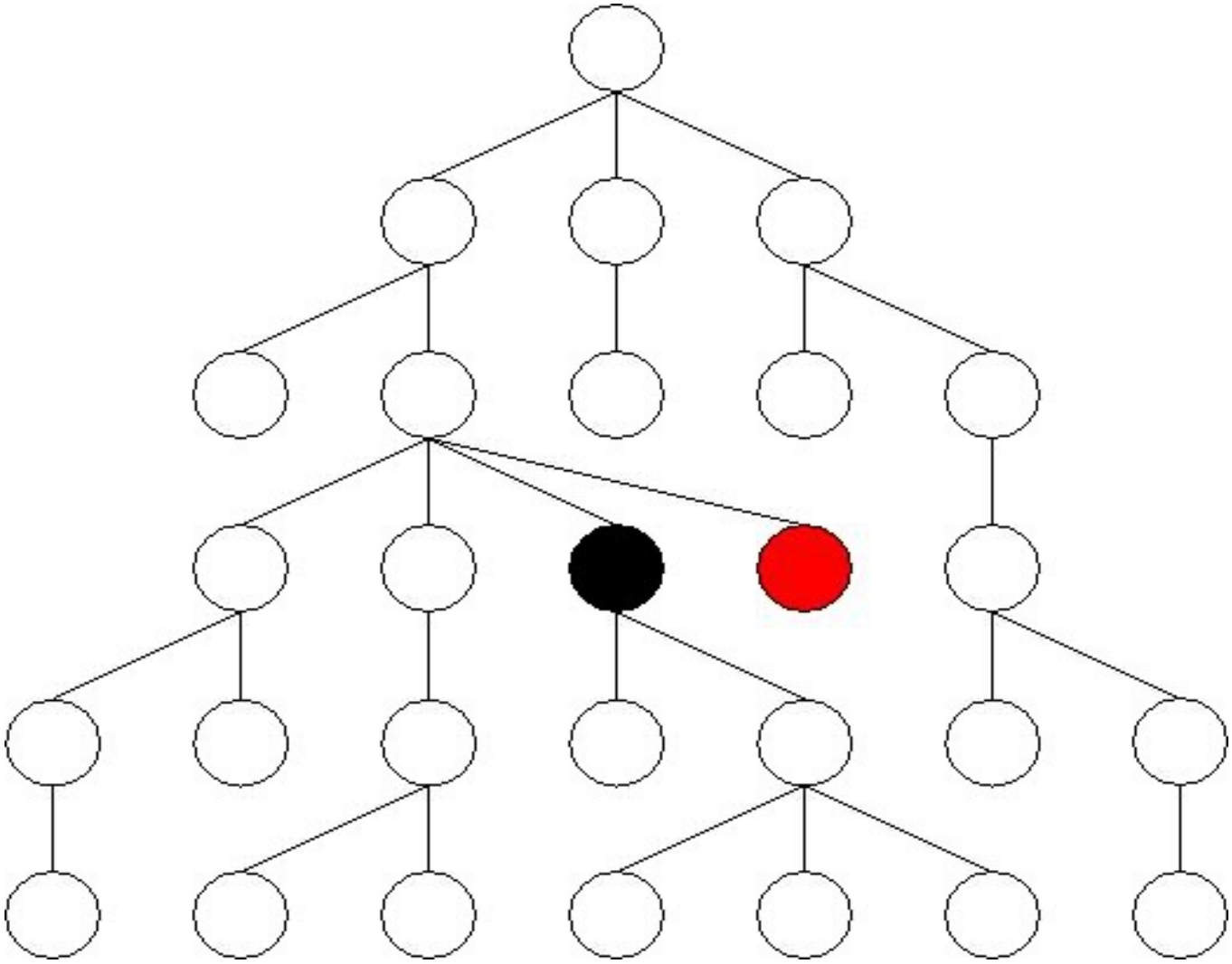
The Descendant Axis



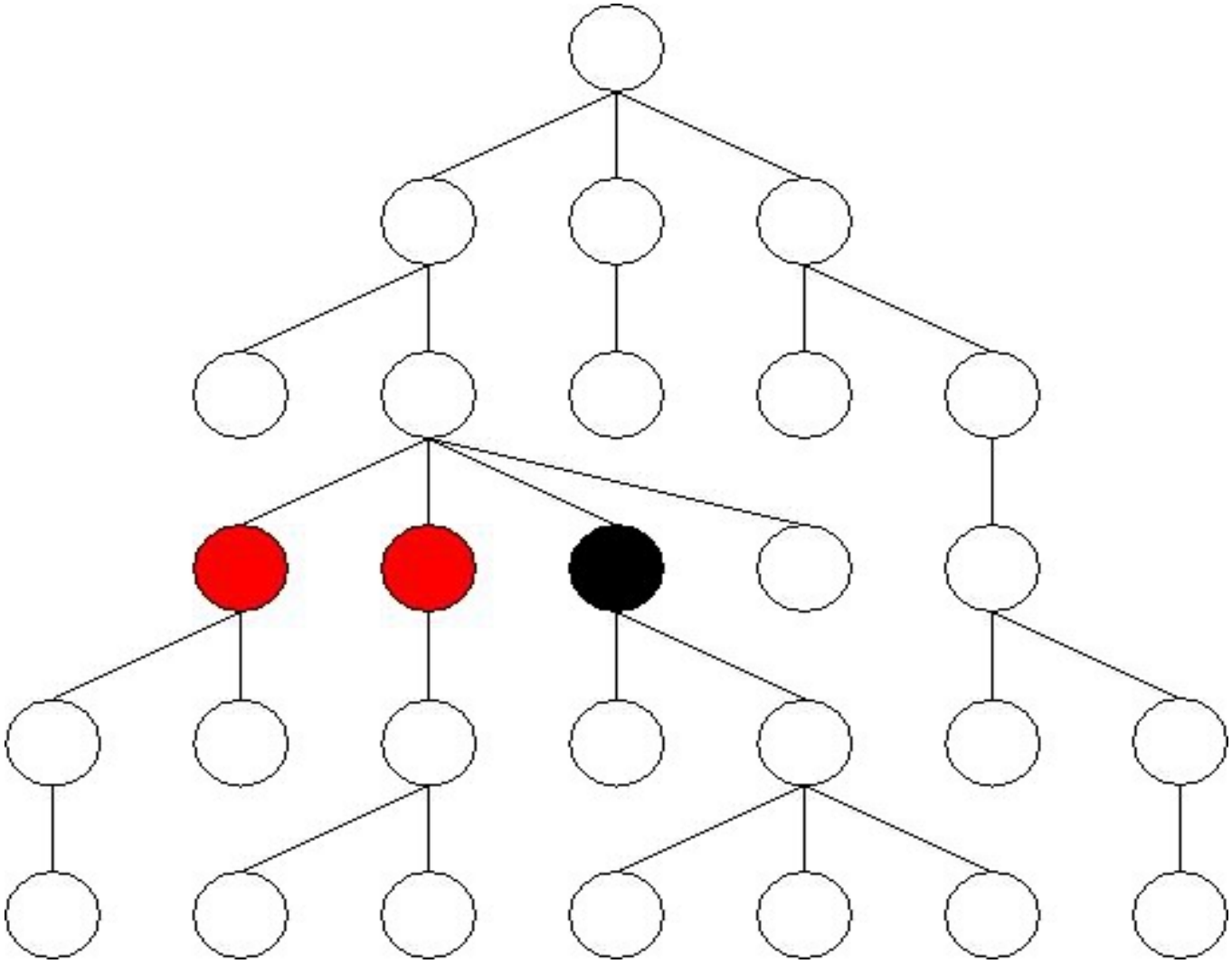
The Ancestor Axis



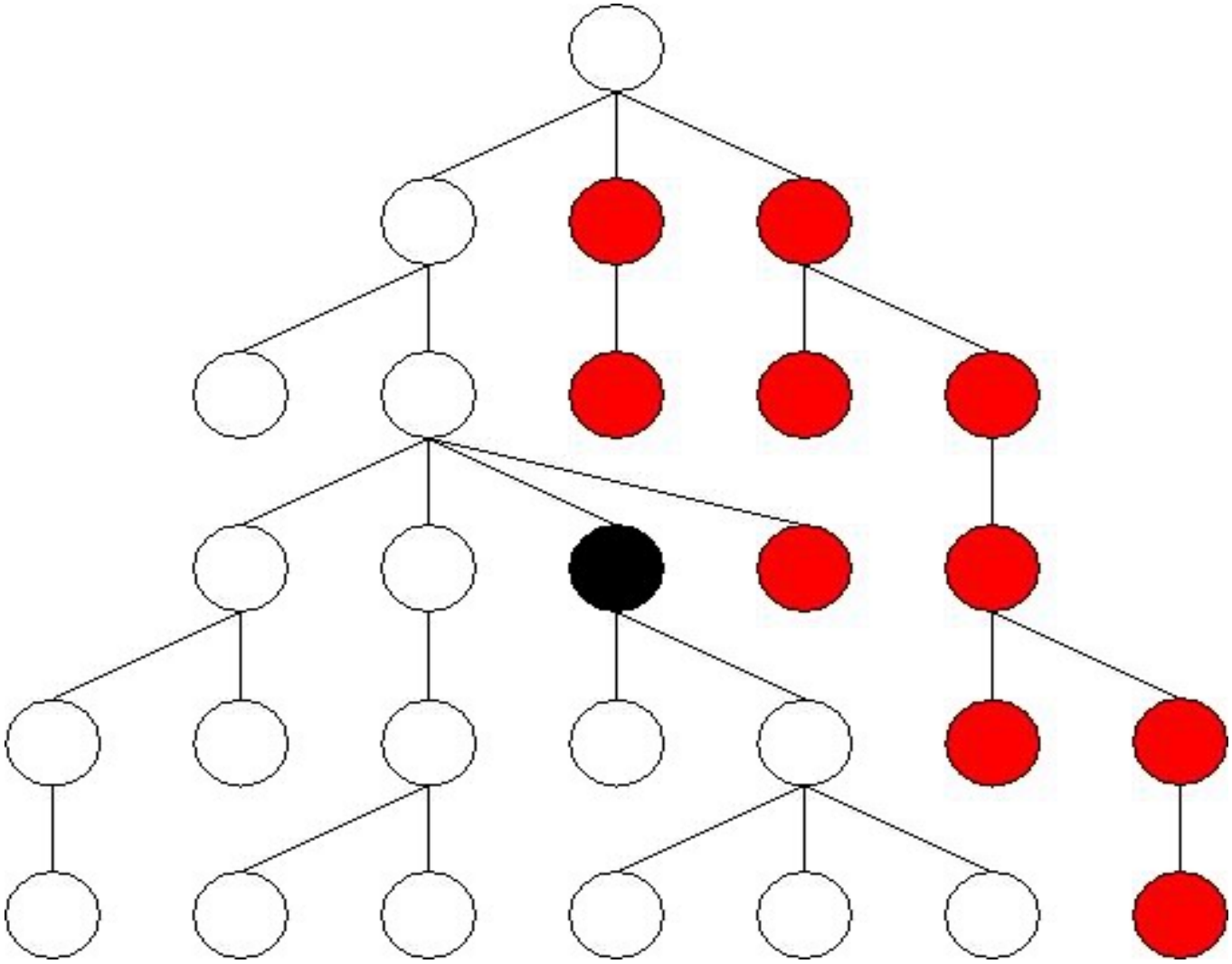
The Following-Sibling Axis



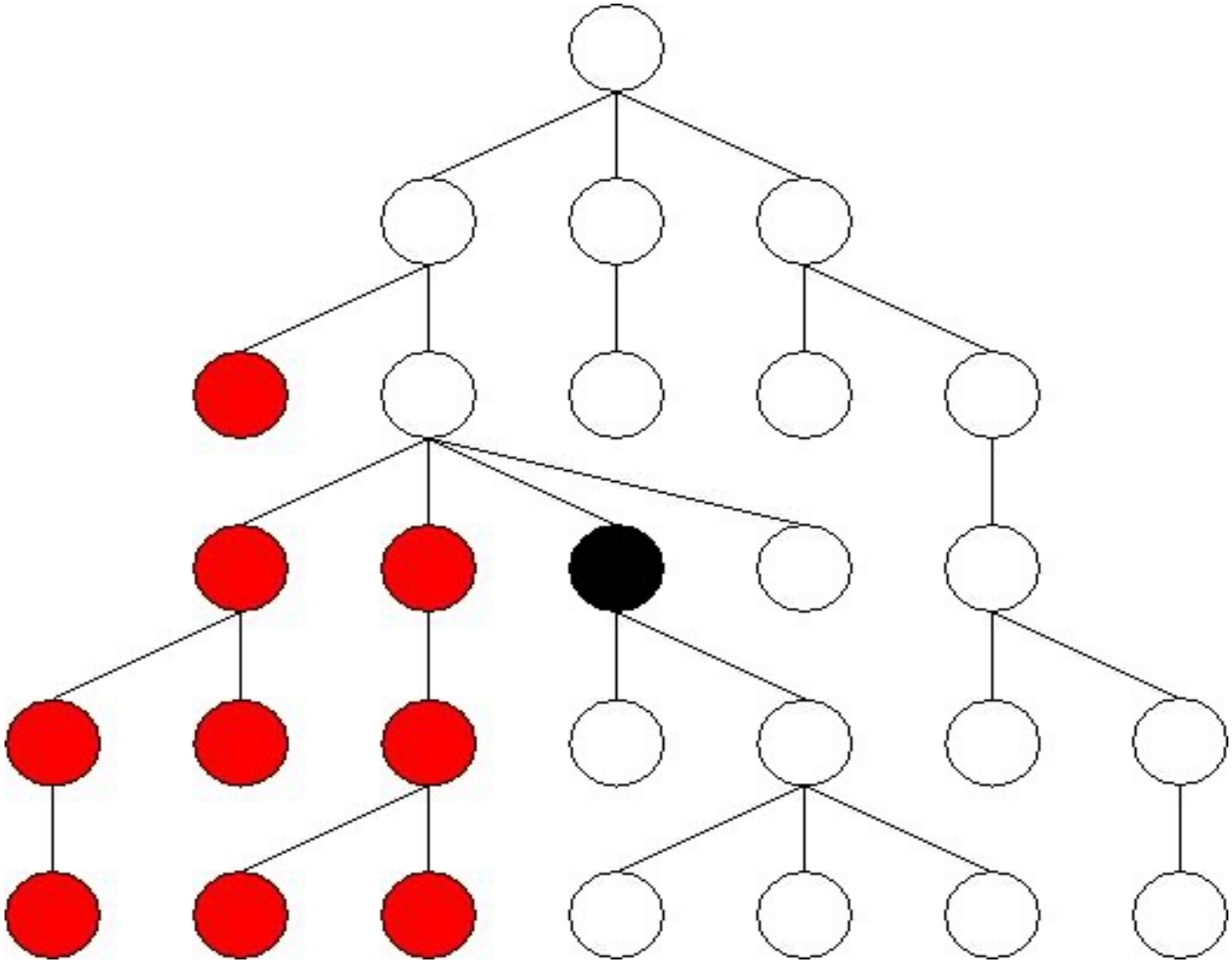
The Preceding-Sibling Axis



The Following Axis: No Descendants

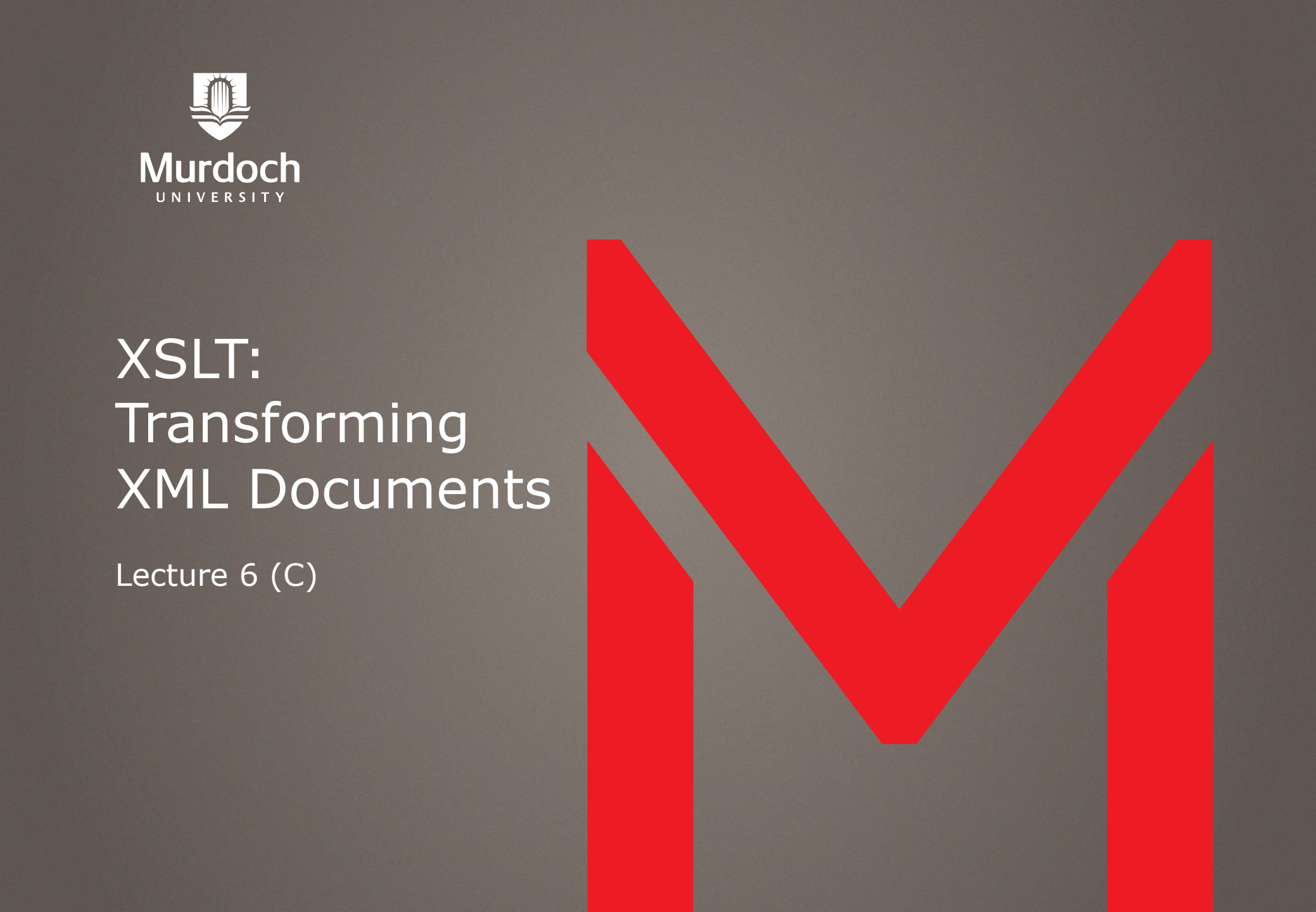


The Preceding Axis: No Ancestors



Extra Reading

- The unit readings on My Unit Readings: Navigating XML Trees with Xpath
- Otherwise, read the “Official Reference” at:
  - http://www.w3.org/TR/xpath
  - This online reference has more detail than the condensed view of the unit reader, and so comprises more valuable information



XSLT: Transforming XML Documents

Lecture 6 (C)

Learning Objectives

- XML is an important set of Internet technologies for use in different solutions in different areas
- When processing XML documents, data stored in the documents will need to be retrieved, and the information processed into other formats
  - XPath is a standard approach for data retrieval
- XSLT is a standard approach for processing information into other formats

Learning Objectives

- Understand the role of **XSL** and **XSLT** in XML technologies
- Understand and be able to write **XSLT stylesheets** to transform XML documents into desired output

Questions

- What are XSL and XSLT?
- How are XML documents rendered in browsers?
- How does the XSLT language transform XML documents?
- How is XPath used in XSLT?

The eXtensible Stylesheet Language

- The eXtensible Stylesheet Language (XSL) is an XML-based language used to create "style sheets"
  - This is analogous to CSS to create style sheets for HTML and XHTML
- XML software can use XSL to transform an XML document into another document
  - That is, into another XML format, HTML, or any other text-based format (even RTF and PDF, although these are quite complex)

What is XSLT?

- eXtensible Stylesheet Language Transformation (XSLT) transforms XML documents into XML, HTML, XHTML or plain text documents
- The style sheet defines how to get from the input document (**source tree**) to the output document (**result tree**)
  - It relies on XPath to identify and find nodes in XML documents

Why is XSLT Important?

- Extracting appropriate information
  - Software and users would probably only require a small part of the information in one XML document
  - And software and users would probably require information from more than one XML document
- Use across different applications
  - We must be able to get the information stored in an XML document into and out of legacy systems which do not deal with XML
  - Even systems which deal with XML may not use the same XML mark-up language

Presenting a Business Card


```
<card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```

An XML document for representing business cards

As displayed in a browser – not the desired outcome

Using CSS


```
card  { background-color: #cccccc; border: none; width: 300;}
name  { display: block; font-size: 20pt; margin-left: 0; }
title { display: block; margin-left: 20pt;}
email { display: block; font-family: monospace; margin-left: 20pt;}
phone { display: block; margin-left: 20pt;}
```

Result looks more like a Business card

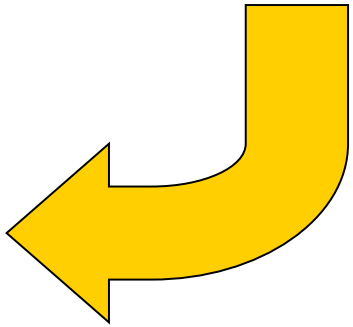
However:

- Additional structure cannot be introduced
- The information cannot be re-arranged
- Information encoded in attributes cannot be exploited

Murdoch UNIVERSITY

Using XSL

```
<?xml-stylesheet type="text/xsl" href="businesscard.xsl"?>
<card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```



XSLT for Business Cards (1/2)

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

  <xsl:template match="bc:card">
    <html>
      <head>
        <title><xsl:value-of select="bc:name/text()"/></title>
      </head>
      <body bgcolor="#ffffff">
        <table border="3">
          <tr>
            <td>
              <xsl:apply-templates select="bc:name"/><br />
              <xsl:apply-templates select="bc:title"/><p />
              <tt><xsl:apply-templates select="bc:email"/></tt><br />
```

XSLT for Business Cards (2/2)

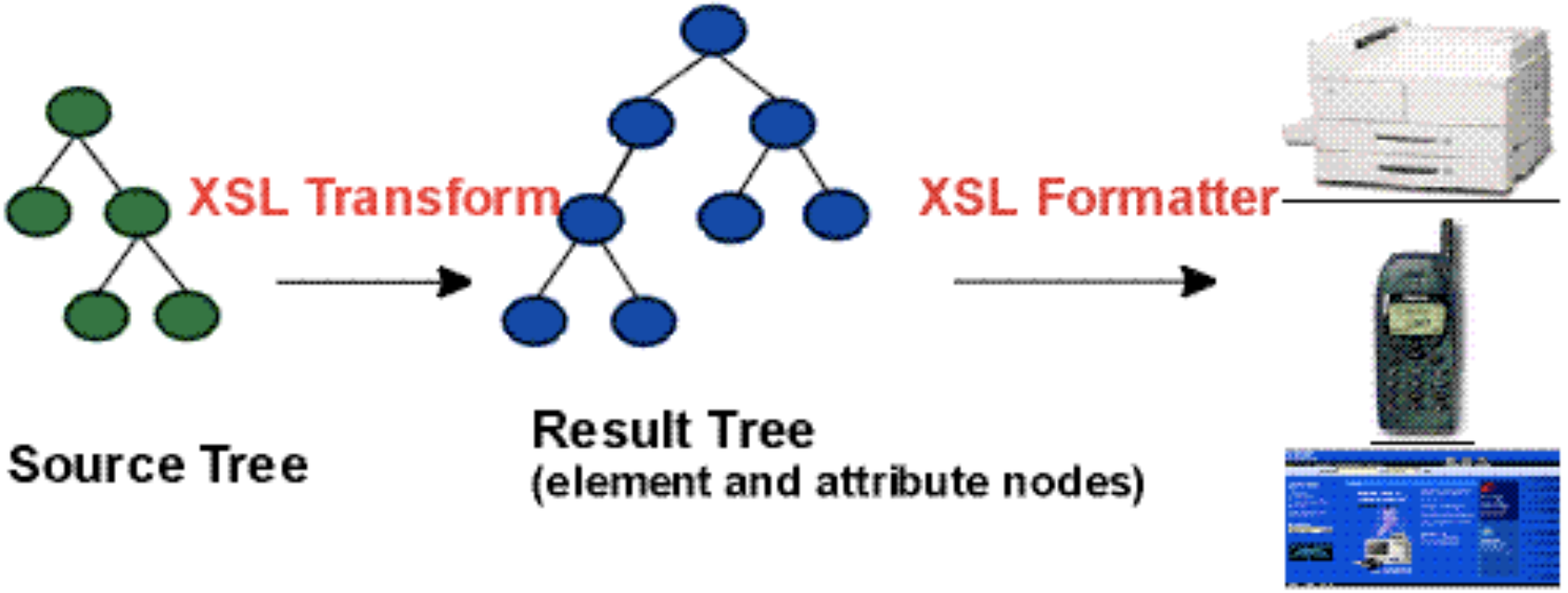
```
            <xsl:if test="bc:phone">
              Phone: <xsl:apply-templates select="bc:phone"/><br />
            </xsl:if>
          </td>
          <td>
            <xsl:if test="bc:logo">
              <img src="{bc:logo/@uri}" />
            </xsl:if>
          </td>
        </tr>
      </table>
    </body>
  </html>
 </xsl:template>
 <xsl:template match="bc:name|bc:title|bc:email|bc:phone">
   <xsl:value-of select="text()"/>
 </xsl:template>
</xsl:stylesheet>
```

Murdoch UNIVERSITY

XSLT and XSL:FO

- There are two complete languages under XSL:
  - XSL for Transformation (XSLT) - to transform the XML document to another document format
  - XSL Formatting Objects (XSL:FO) - to format the result tree for display (on different devices)
- The official W3C Recommendation for both can be found at:
  - http://www.w3c.org/TR/xslt
  - http://www.w3.org/TR/xsl

XSL Two Processes: Transformation And Formatting



Result XML tree is the result of XSLT processing.

Tree Transformations With XSLT

- Tree transformation constructs the **result tree** which is also called the **element and attribute tree**
- The objects are primarily in the “formatting object” namespace
- Tree Transformation is defined in the XSLT Recommendation
- We will return to look at XSLT in more detail, after first looking at XSL:FO

Tree Transformations with XSLT

With tree transformation, the structure of the result tree can be quite different from the structure of the source tree

Source Tree

Result Tree
(element and attribute nodes)

In constructing the result tree, the source tree can be filtered and reordered, and arbitrary structure and generated content can be added.

Murdoch UNIVERSITY

Formatting with XSL:FO

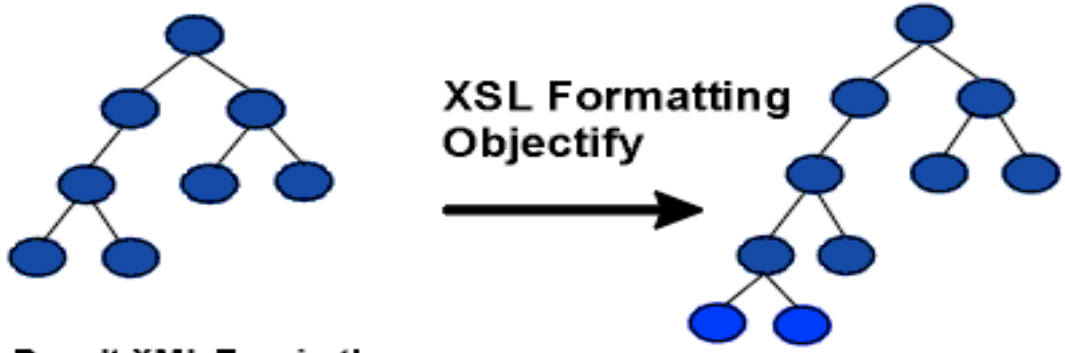
- Formatting interprets the result tree in its **formatting object tree form** to produce the presentation intended by the designer of the style sheet
- The **properties** associated with an instance of a formatting object control the formatting of that object

Formatting: 1st Phase

- The **first phase** in formatting is to "objectify" the element and attribute tree obtained via an XSLT transformation
- Objectifying the tree basically consists of turning the elements in the tree into formatting object nodes, and the attributes into property specifications
- The result of this first step is the **formatting object tree**

Formatting: 1st Phase

The XSL FO tree is processed: characters are converted to character FOs and compound properties are built.



Result XML Tree in the "fo" namespace (element and attribute nodes)

XSL formatting objects/properties

Some of the properties, for example "color", directly specify the formatted result.

Other properties, for example 'space-before', only constrain the set of possible formatted results without specifying any particular formatted result.

Murdoch UNIVERSITY

Formatting: 2nd Phase

- The **second phase** in formatting is to refine the formatting object tree to produce the **refined formatting object tree**
- The refinement process handles the mapping from properties to traits

Formatting: 2nd Phase

- Refinement consists of:
  1. Shorthand expansion into individual properties
  2. Mapping of corresponding properties
  3. Determining computed values (may include expression evaluation)
  4. Handling white-space-treatment and linefeed-treatment property effects
  5. Inheritance

Formatting: 2nd Phase

The XSL formatting object tree is refined in an iterative fashion.

Property inheritance is resolved, computed values are processed, expressions are evaluated, and duplicate corresponding properties are removed.

[D]

Murdoch UNIVERSITY

Formatting: 3rd Phase

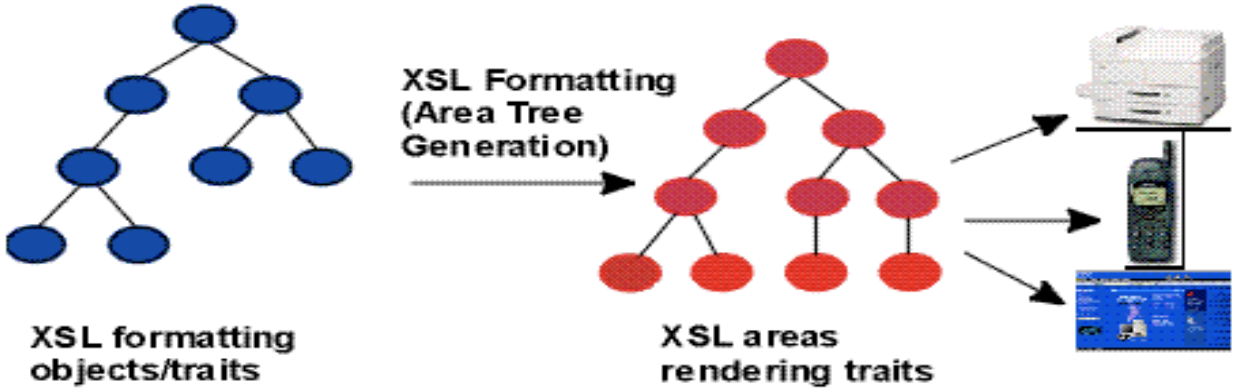
- The **third phase** in formatting is the construction of the **area tree**
- The area tree is generated as described in the semantics of each formatting object
- The traits applicable to each formatting object class control how the areas are generated

Formatting: 3rd Phase

- Although every formatting property may be specified on every formatting object, for each formatting object class only a subset of the formatting properties are used to determine the traits for objects of that class

Formatting: 3rd Phase

The last part of formatting describes the generation of a tree of geometric areas. These areas are positioned on a sequence of one or more pages.



Each geometric area has a position on the page, a specification of what to display in that area and may have a background, padding, and borders.

Murdoch UNIVERSITY

XSLT: Pattern Matching

- The XSLT language basically defines a set of **templates**
- The templates specify
  1. What to look for in the source tree, and
  2. What to put in the result tree
- The processing of a style-sheet involves matching templates against the contents of XML documents
  - If matches are determined, then the templates are applied

XSLT: Pattern Matching

- XSLT is a *declarative* language **not** a *procedural* language
- Many people encounter problems writing XSLT because they fail to think in terms of pattern matching using templates
  - The natural tendency is to think of XSLT statements as "instructions"

XSLT: Pattern Matching

- If you think in terms of "instructions" rather than "pattern matching using templates", your style-sheets will **not** work as you expect them to
- Please take time to understand this topic
- Read the examples in the Unit readings to grasp this concept of defining "patterns" rather than defining "instructions"

Associating XML Document with XSLT Style Sheets

- An XML document can be associated with a given XSLT style sheet by putting the tag `<?xml-stylesheet … ?>` in the **Processing Instructions** parts of the XML document

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
                 href="myxsl.xsl" ?>
```

- You would have done something similar to this with CSS in HTML or XHTML

Associating XML Document with XSLT Style Sheets

- This gives the software processing the XML document (eg: browsers) the option of using that style sheet to transform the document if it is deemed appropriate
  - However, the software will first have to know how to deal with "`text/xsl`" documents

An XSLT Style Sheet

- Since an XSLT style sheet is also an XML document, it conforms to all the well-formed rules we have discussed previously
  - It should have the `<?xml …?>` declaration
  - The root element of the style sheet is `<xsl:stylesheet>`
  - `<xsl:stylesheet>` contains the zero or more elements `<xsl:template>` which define the templates for the transformations

The XSLT Style Sheet

- So the basic structure of the XSLT style sheet is formally:

```
<?xml version="1.0" ?>
<xsl:stylesheet ... >
  <xsl:template match=XPath expression >
    ... Something to do ...
  </xsl:template>
  <xsl:template match=XPath expression >
    ... Something different to do ...
  </xsl:template>
  ...
</xsl:stylesheet>
```

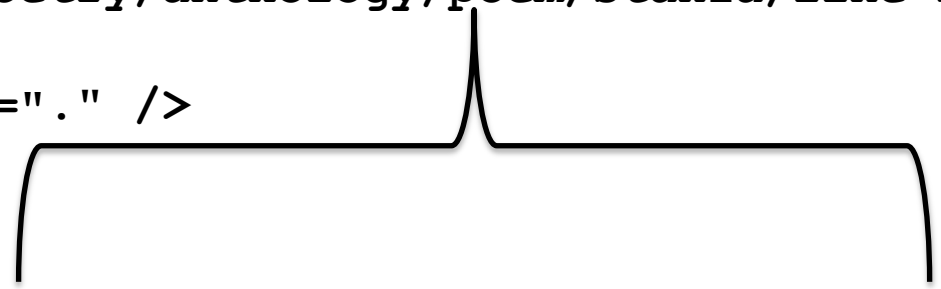
Example XML Document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="poetry.xsl"?>
<poetry>
   <anthology>
     <poem>
          <title>The SICK ROSE</title>
          <author>William Blake</author>
          <stanza>
               <line>O Rose thou art sick.</line>
               <line>The invisible worm,</line>
               <line>That flies in the night</line>
               <line>In the howling storm</line>
          </stanza>
          <stanza>
               <line></line>
               <line></line>
               <line />
               <line />
          </stanza>
     </poem>
   </anthology>
</poetry>
```

An XSLT Style Sheet

- Transforms from XML to HTML or XHTML

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <xsl:for-each select="/poetry/anthology/poem/stanza/line">
          <p>
            <xsl:value-of select="." />
          </p>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```



Path to required XML element using XPath

Murdoch UNIVERSITY

Matching the Source Tree Using XPath Expressions

- As mentioned, a major part of XSLT's job is to retrieve data from a source tree (the original XML document)
- For this to happen, there must be some convenient way of locating components of the source tree
- As seen in the previous example, we do this using XPath expressions

Evaluating the Templates

- What XSLT will do is set the current context node to the document root, and start finding the template that matches the current node
  - It will then produce the result tree according to what is specified in the body of the template
- If more than one template matches the current node, the last one (counting from top to bottom) will be evaluated

Evaluating the Templates

- If you want the suitable templates to be applied to the nodes below the current node (context node), you **must** specify the element

```
<xsl:apply-templates … >
```

within the body of the template which matches the current node

Evaluating the Templates

- Eg. The following will not work:

```
<xsl:template match="/">
  <!-- Do nothing -->
</xsl:template>

<xsl:template match="line">
  The line is: <xsl:value-of select="."/>
</xsl:template>
```

This template is **never** matched!

- But this will:

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

<xsl:template match="line">
  The line is: <xsl:value-of select="."/>
</xsl:template>
```

Match the document root "/" or any other node "*" - so this recursively applies all templates to EVERY node.

Evaluating the Templates

- In summary: at any time while going through the XML source tree …

… if the current context node matches this XPath expression …

```
<xsl:template match=XPath expression>

    ... blah blah ...

</xsl:template>
```

… then put the content specified here into the result tree. If this part of the content is a valid xsl tag (eg: `<xsl:apply-templates />`), then the tag is evaluated, and those results are put in the result tree.

Creating the Result Tree

- We have seen the basics of traversing through the source tree
- Now let's look at how we can get the required information out of the source tree to put in the result tree
- The most commonly used element for doing this is `<xsl:value-of>`

<xsl:value-of>

- The common format for this element is:

```
<xsl:value-of select="XPath Expression">
```

- This puts the value, as specified by the XPath expression, into the result tree
- The XPath expression can contain a lot more than just a location path
  - For example: XPath functions

Functions in XPath

- To get information out of the source tree to use in elements like `<xsl:value-of>`, we can use the set of functions available in XPath
  - `name()` The name of the node
  - `text()` The #PCDATA of the current node
  - `sum()` The sum of the numbers given in the #PCDATA of specified nodes
  - `concat()` Concatenate two strings
- Obviously, there are many more functions available; you should investigate yourself to learn more

Functions in XPath

- Each function in the function library is specified using a function prototype, which specifies the return type, the name of the function, and the type of the arguments
- If an argument type is followed by a question mark, then the argument is optional; otherwise, the argument is required

Functions in XPath

- XPath has an extensive function library
- There are 106 functions specified
- The default *namespace* for XPath functions is:

  `http://www.w3.org/2006/xpath-functions`

- There are more functions in the *namespace*:

  `http://www.w3.org/2001/XMLSchema`

Functions in XPath

- Examples:

```
<xsl:template match="*|/">
  The node is: <xsl:value-of select="name()" />
  <xsl:apply-templates />
</xsl:template>
```

```
<xsl:template match="/">
  The sum of all numbers in this document is:
  <xsl:value-of select="sum(/spreadsheet/numbers)" />
</xsl:template>
```

Functions in XPath

- XPath functions are also useful for specifying location paths
- Eg: Finding different lines

```
<xsl:template match="/course/unit(position()=1)">
```

```
<xsl:template match="/course/unit[1]">
```

```
<xsl:template match="/course/unit(position()=last)">
```

  - (See Core Function Library at http://www.w3.org/TR/xpath)

Functions in XPath

- Node Set Functions – Eg:
  - number `last()` – returns a number equal to the context size from the expression
  - number `position()`
  - number `count(node-set)`
  - node-set `id(object)`
  - ...
- String Functions
- Boolean Functions
- Number Functions

Conditional Processing

- When we are traversing the source tree, we can also perform processing based on conditions
- We do so using the elements such as:

`<xsl:if>` and `<xsl:choose>`

Conditional Processing

- The basic syntax:

```
<xsl:if test="Boolean expression"> ... </xsl:if>

<xsl:choose>
  <xsl:when test="Boolean expression"> ... </xsl:when>
  <xsl:when test="Boolean expression"> ... </xsl:when>
     ...
</xsl:choose>
```

Iterative Processing: Loops

- You can also do loops (iterations) using the element

```
<xsl:for-each>
```

- The basic syntax:

```
<xsl:for-each select="XPath expression">
```

Iterative Processing: Loops

- Eg:

```
<xsl:template match="unit">
    <xsl:for-each select="lecturer">
       Another lecturer
    </xsl:for-each>
</xsl:template>
```

Built-in Templates

- By default, the following templates will already exist:

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

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

- Some syntax abbreviations are listed in XPathAbbreviationSyntax.txt (on LMS)

Built-in Templates

- So even if you have no templates in your style sheet, the `#PCDATA` in every node of your document will be put in the result tree
  - If a node doesn't have `#PCDATA`, a new-line character will be put in the result tree
- If you don't define any templates to override the default behaviours, they will be invoked. This can sometimes cause confusion among students because their templates are producing results they never defined

Extra Reading

- An introduction to eXtensible Style Sheets at: “http://www.xml.com/pub/a/1999/01/walsh1.html?page=1”
- The unit readings on My Unit Readings: Navigating XML Trees with Xpath and Transforming XML Documents with XSLT
- Otherwise, read the “Official References” at
  - http://www.w3.org/TR/xslt
  - http://www.w3.org/TR/xpath
- These have far more detail than the condensed view of the unit readers, and so comprise more valuable information