WIFI

- SSID: Forthnet
- User: piop
- Passwd: piop1234

Γλώσσες σήμανσης και δομές αποθήκευσης

Νικόλαος Παπαδάκης
Δόκτωρ Μηχανικός

Presentation Overview

- XML and HTML
- XML basics
- More… XML
- XML Schemas and DTD
- Xpath
- Databases and SQL -similarities
- SPARQL and dbpedia
- Graph databases and Neo4J

HTML and XML

XML stands for eXtensible Markup Language

HTML is used to mark up text so it can be displayed to users

**HTML describes both structure (e.g. <p>, <h2>,) and appearance (e.g.
, <font>, <i>)**

HTML uses a fixed, unchangeable set of tags

XML is used to mark up data so it can be processed by computers

XML describes only content, or "meaning"

In XML, you make up your own tags

HTML and XML

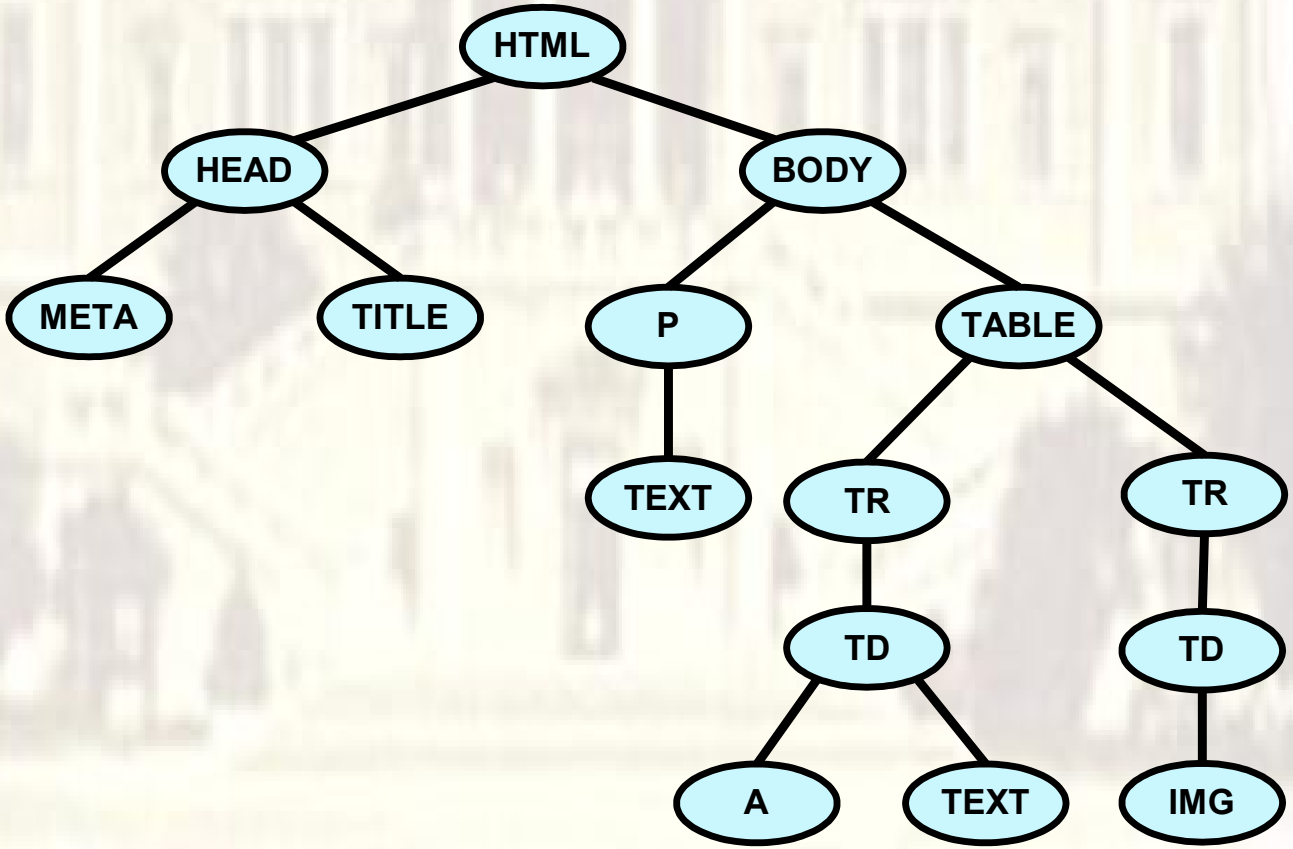
- HTML and XML look similar, because they are both SGML languages (SGML = Standard Generalized Markup Language)
  - Both HTML and XML use elements enclosed in tags (e.g. <body>This is an element</body>)
  - Both use tag attributes (e.g., <font face="Verdana" size="+1" color="red">)
  - Both use entities (<, >, &, ", ')
- More precisely,
  - HTML is defined in SGML
  - XML is a (very small) subset of SGML

A simple web page

Its html source code...


```
<html>
<head>
<meta name="generator" content="web mining extractor"/>
<title></title>
</head>
<body> <p style="font-size: 200%; font-weight: bold">Welcome</p>
<table border="1" width="100%"> <tr>
<td width="100%"><a href="http://www.ntua.gr/">NTUA</a> is National Technical University of Athens</td> </tr><tr>
<td width="100%"><img border="0" src="/aPic.JPG" width="110" height="80" /></td>
</tr></table>
</body>
</html>
```

Its tree representation



HTML and XML

- HTML is for humans
  - HTML describes web pages
  - You don't want to see error messages about the web pages you visit
  - Browsers ignore and/or correct as many HTML errors as they can, so HTML is often sloppy
- XML is for computers
  - XML describes data
  - The rules are strict and errors are not allowed
    - In this way, XML is like a programming language
  - Current versions of most browsers can display XML
    - However, browser support of XML is spotty at best

XML-related technologies

- DTD (Document Type Definition) and XML Schemas are used to define legal XML tags and their attributes for particular purposes

- CSS (Cascading Style Sheets) describe how to display HTML or XML in a browser

- XSLT (eXtensible Stylesheet Language Transformations) and XPath are used to translate from one form of XML to another

- DOM (Document Object Model), SAX (Simple API for XML, and JAXP (Java API for XML Processing) are all APIs for XML parsing

Simple XML example


```
<?xml version="1.0"?>
<weatherReport>
   <date>3/10/2014</date>
   <city>Athens</city><state>Attika</state>
   <country>Greece</country>
   High Temp: <high scale="C">20</high>
   Low Temp: <low scale="C">14</low>
   Morning: <morning>Partly cloudy, Hazy</morning>
   Afternoon: <afternoon>Sunny &amp;amp; hot</afternoon>
   Evening: <evening>Clear and Cooler</evening>
</weatherReport>
```

Overall structure

An XML document may start with one or more processing instructions (PIs) or directives:

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

Following the directives, there must be exactly *one* root element containing all the rest of the XML:

```
<weatherReport>
    ...
</weatherReport>
```

XML building blocks

Aside from the directives, an XML document is built from:

- **elements**: high in <high scale="C">14</high>
- **tags**, in pairs: <high scale="C">20</high>
- **attributes**: <high scale="C">20</high>
- **entities**: <afternoon>Sunny & hot</afternoon>
- **character data**, which may be:

parsed (processed as XML)--this is the default

unparsed (all characters stand for themselves)

Elements and attributes

- Attributes and elements are somewhat interchangeable
- Example using just elements:

```
<name>
<first>Nikolaos</first>
<last>Papadakis</last>
</name>
```

- Example using attributes:

```
<name first="Nikolaos" last="Papadakis"></name>
```

- Elements are easier to use in programs
- Attributes often contain metadata, such as unique IDs

Well-formed XML

- Every element must have *both* a start tag and an end tag, e.g. <name> ... </name>
  - But empty elements can be abbreviated: <break />.
  - XML tags are case sensitive
  - XML tags may not begin with the letters xml, in any combination of cases
- Elements must be properly nested, e.g. *not* bold and italic
- Every XML document must have one and only one root element
- The values of attributes must be enclosed in single or double quotes, e.g. <time unit="days">
- Character data cannot contain < or &

Special chars

Five special characters must be written as entities:

& for & (almost always necessary)

< for < (almost always necessary)

> for > (not usually necessary)

" for " (necessary inside double quotes)

' for ' (necessary inside single quotes)

These entities can be used even in places where they are not absolutely required

These are the *only* predefined entities in XML

XML declaration

- The XML declaration looks like this:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
```

  - The XML declaration is not required by browsers, but *is* required by most XML processors (so include it!)
  - If present, the XML declaration must be first--*not even whitespace* should precede it
  - Note that the brackets are <? and ?>
  - version="1.0" is required (this is the *only* version so far)
  - encoding can be "UTF-8" (ASCII) or "UTF-16" (Unicode), or something else, or it can be omitted
  - standalone tells whether there is a separate DTD

Processing instructions

- PIs (Processing Instructions) may occur anywhere in the XML document (but usually first)
- A PI is a command to the program processing the XML document to handle it in a certain way
- XML documents are typically processed by more than one program
- Programs that do not recognize a given PI should just ignore it
- General format of a PI: <?*target instructions*?>
- Example: <?xml-stylesheet type="text/css" href="mySheet.css"?>

Comments

<!-- This is a comment in both HTML and XML -->

Comments can be put anywhere in an XML document

Comments are useful for:

Explaining the structure of an XML document to humans

Commenting out parts of the XML during development and testing

Comments are not elements and do not have an end tag

The blanks after <!-- and before --> are optional

The character sequence -- cannot occur in the comment

The closing bracket *must* be -->

Comments are not displayed by browsers, but can be seen by anyone who looks at the source code

CDATA

- By default, all text inside an XML document is parsed
- text to be treated as unparsed *character data* by enclosing it in <![CDATA[...]]>
- Any characters, even & and <, can occur inside a CDATA
- Whitespace inside a CDATA is (usually) preserved
- The only real restriction is that the character sequence]]> cannot occur inside a CDATA
- CDATA is useful when your text has a lot of illegal characters (for example, if your XML document contains some HTML text)

Names in XML

- Names (as used for tags and attributes) must begin with a letter or underscore, and can consist of:
  - Letters, both Roman (English) and foreign
  - Digits, both Roman and foreign
  - Dot (.)
- Hyphen (-)
- Underscore (_)
- Colon (:) should be used only for namespaces

Namespaces

- Recall that DTDs are used to define the tags that can be used in an XML document
- An XML document may reference more than one DTD

Namespaces are a way to specify which DTD defines a given tag

XML, like Java, uses qualified names

This helps to avoid collisions between names

Java: myObject.myVariable

XML: myDTD:myTag

Note that XML uses a colon (:) rather than a dot (.)

Namespaces and URIs

A namespace is defined as a unique string

To guarantee uniqueness, typically a URI (Uniform Resource Indicator) is used, because the author "owns" the domain

It doesn't have to be a "real" URI; it just has to be a unique string

Example: http://www.nkpap.org/nameSpace

There are two ways to use namespaces:

Declare a default namespace

Associate a prefix with a namespace, then use the prefix in the XML to refer to the namespace

Namespace syntax

In *any* start tag you can use the reserved attribute name xmlns: <book xmlns="http://www.nkpap.org/nameSpace">

This namespace will be used as the default for all elements up to the corresponding end tag

- You can override it with a specific prefix

You can use almost this same form to declare a prefix: <book xmlns:dave="http://www.nkpap.org/nameSpace">

- Use this prefix on *every tag and attribute* you want to use from this namespace, including end tags--it is *not* a default prefix <dave:chapter dave:number="1">To Begin</dave:chapter>

You can use the prefix in the start tag in which it is defined: <dave:book xmlns:dave="http://www.nkpap.org/nameSpace">

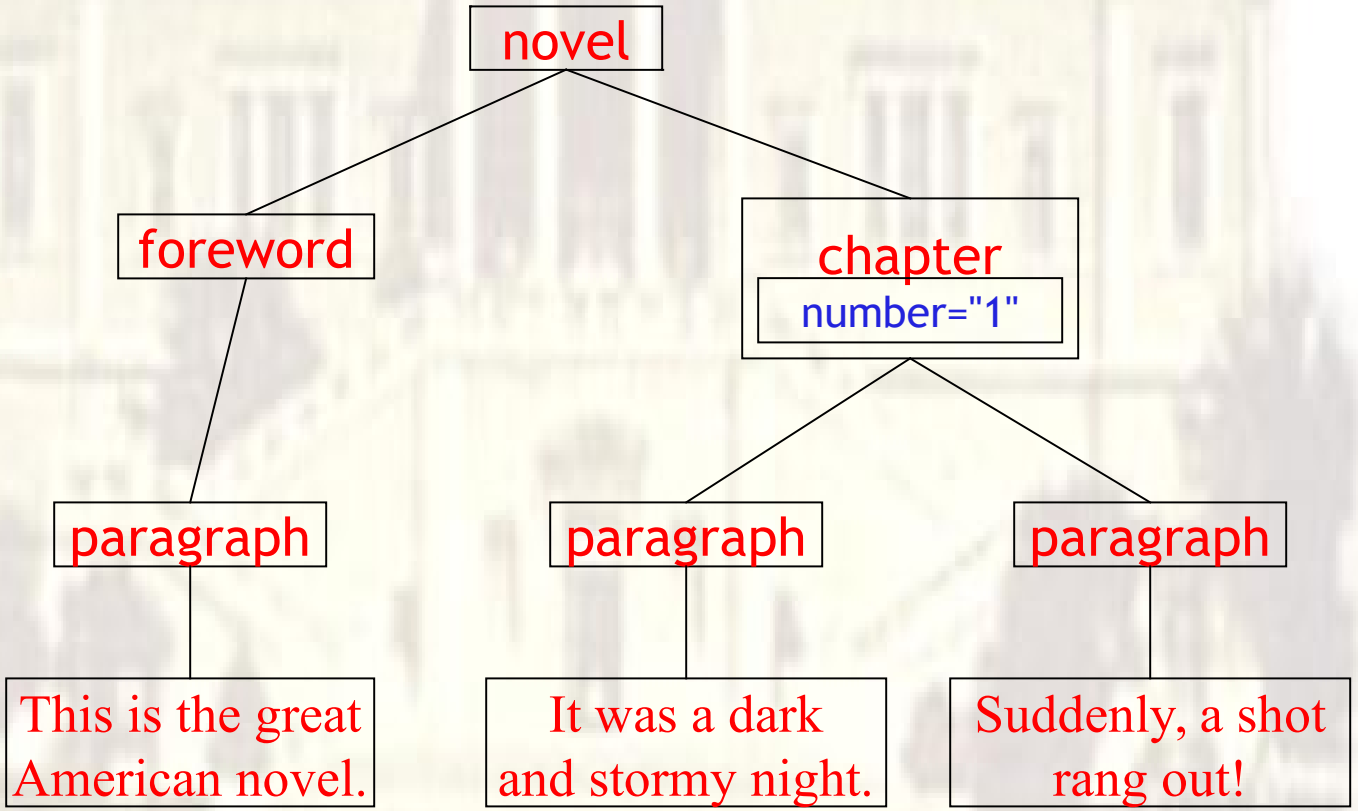
Review of XML rules

- Start with <?xml version="1"?>
- XML is case sensitive
- Must have exactly one root element
- Every element must have a closing tag
- Elements must be properly nested
- Attribute values must be enclosed in double or single quotation marks
- There are only five predeclared entities

A well-structured example


```
<novel>
  <foreword>
    <paragraph> This is the great American novel.
    </paragraph>
  </foreword>
  <chapter number="1">
    <paragraph>It was a dark and stormy night.
    </paragraph>
    <paragraph>Suddenly, a shot rang out!
    </paragraph>
  </chapter>
</novel>
```

XML as a tree


novel
foreword
chapter
number="1"
paragraph
paragraph
paragraph
This is the great American novel.
It was a dark and stormy night.
Suddenly, a shot rang out!

Valid XML

You can make up your own XML tags and attributes, *but...*

...any program that *uses* the XML must know what to expect!

- A DTD (Document Type Definition) defines what tags are legal and where they can occur in the XML
- An XML document *does not require* a DTD
- XML is well-structured if it follows the rules given earlier
- In addition, XML is valid if it declares a DTD and conforms to that DTD
- A DTD can be included in the XML, but is typically a separate document
- Errors in XML documents will *stop* XML programs
- Some alternatives to DTDs are XML Schemas and RELAX NG

Viewing XML

- XML is designed to be processed by computer programs, not to be displayed to humans
  - Nevertheless, almost all current browsers can display XML documents
  - They don't all display it the same way
  - They may not display it at all if it has errors

For best results, update your browsers to the newest available versions

Remember:

HTML is designed to be *viewed*,
XML is designed to be *used*

Extended document standards

- You can define your own XML tag sets, but here are some already available:
  - XHTML: HTML redefined in XML
  - SMIL: Synchronized Multimedia Integration Language
  - MathML: Mathematical Markup Language
  - SVG: Scalable Vector Graphics
  - DrawML: Drawing MetaLanguage
  - ICE: Information and Content Exchange
  - ebXML: Electronic Business with XML
  - cxml: Commerce XML
  - CBL: Common Business Library

Vocabulary

SGML: Standard Generalized Markup Language

XML : Extensible Markup Language

DTD: Document Type Definition

element: a start and end tag, along with their contents

attribute: a value given in the start tag of an element

entity: a representation of a particular character or string

PI: a Processing Instruction, to possibly be used by a program that processes this XML

namespace: a unique string that references a DTD

well-formed XML: XML that follows the basic syntax rules

valid XML: well-formed XML that conforms to a DTD

XPath

XML and RDBMs

Database Side: XML is a way to organize data:

- Relational databases organize data in tables
- XML documents organize data in ordered trees

Data Management: -- Relational vs. XML

Relational data are well organized – fully structured (more strict):

- E-R modeling to model the data structures in the application;
- E-R diagram is converted to relational tables and integrity constraints (relational schemas)

XML data are semi-structured (more flexible):

- Schemas may be unfixed, or unknown (flexible – anyone can author a document);
- Suitable for data integration (data on the web, data exchange between different enterprises).

Relational vs. XML

- XML is not meant to replace relational database systems
  - RDBMSs are well suited to OLTP applications (e.g., electronic banking) which has 1000+ small transactions per minute
  - XML is suitable data exchange over heterogeneous data sources (e.g., Web services) that allow them to "talk".

When should we use XML

- Web services:
  - SOAP,
  - WSDL,
  - UDDI

Any data having hierarchical structure:

- Email
  - Header – from, to, cc, bcc…
  - Body – my message, replied email …

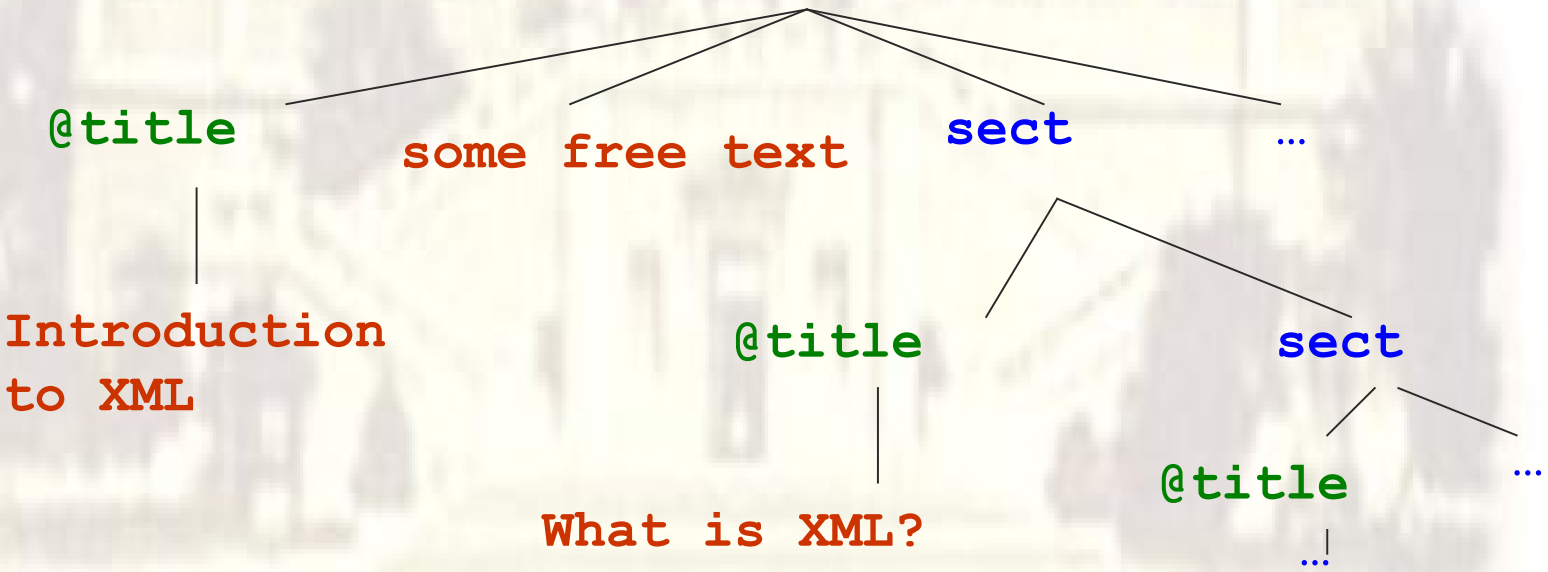
Network log files

- IP address, time, request type, error code

Remember XML IS a tree!!!

Hierarchical data model

An XML document is an ordered tree!



Paths analogies

Operating system:

/ = the root directory

/users/dave/foo = the (one) file named foo in dave in users

foo = the (one) file named foo in the current directory

. = the current directory

.. = the parent directory

/users/dave/* = all the files in /users/dave

Xpath

/library = the root element (if named library)

/library/book/chapter/section = *every* section element in a chapter in every book in the library

section = *every* section element that is a child of the current element

. = the current element

.. = parent of the current element

/library/book/chapter/* = all the elements in /library/book/chapter

Slashes

A path that begins with a / represents an absolute path, starting from the top of the document

- Example: /email/message/header/from
  - Note that even an absolute path can select more than one element
  - A slash by itself means "the whole document"

A path that does not begin with a / represents a path starting from the current element

- Example: header/from

A path that begins with // can start from anywhere in the document

- Example: //header/from selects every element from that is a child of an element header
  - This can be expensive, since it involves searching the entire document

Brackets and last()

A number in brackets selects a particular matching child (counting starts from 1, except in Internet Explorer)

- Example: /library/book[1] selects the first book of the library
- Example: //chapter/section[2] selects the second section of every chapter in the XML document
- Example: //book/chapter[1]/section[2]
- Only *matching* elements are counted; for example, if a book has both sections and exercises, the latter are ignored when counting sections

The function last() in brackets selects the last matching child

- Example: /library/book/chapter[last()]

You can even do simple arithmetic

- Example: /library/book/chapter[last()-1]

Stars

A star, or asterisk, is a "wild card"—it means "all the elements at this level"

- Example: /library/book/chapter/* selects every child of every chapter of every book in the library
- Example: //book/* selects every child of every book (chapters, tableOfContents, index, etc.)
- Example: /*/*/*/paragraph selects every paragraph that has exactly three ancestors
- Example: //* selects every element in the entire document

Attributes I

You can select attributes by themselves, or elements that have certain attributes

- Remember: an **attribute** consists of a name-value pair, for example in <chapter num="5">, the attribute is named num
- To choose the attribute itself, prefix the name with @
  - Example: @num will choose every *attribute* named num
  - Example: //@* will choose *every attribute, everywhere* in the document

To choose *elements* that have a given attribute, put the attribute name in square brackets

- Example: //chapter[@num] will select every chapter element (anywhere in the document) that has an attribute named num

Attributes II

//chapter[@num] selects every chapter element with an attribute num

//chapter[not(@num)] selects every chapter element that does *not* have a num attribute

//chapter[@*] selects every chapter element that has *any* attribute

//chapter[not(@*)] selects every chapter element with *no* attributes

Values of attributes

//chapter[@num='3'] selects every chapter element with an attribute num with value 3

//chapter[not(@num)] selects every chapter element that does *not* have a num attribute

//chapter[@*] selects every chapter element that has *any* attribute

//chapter[not(@*)] selects every chapter element with *no* attributes

The normalize-space() function can be used to remove leading and trailing spaces from a value before comparison

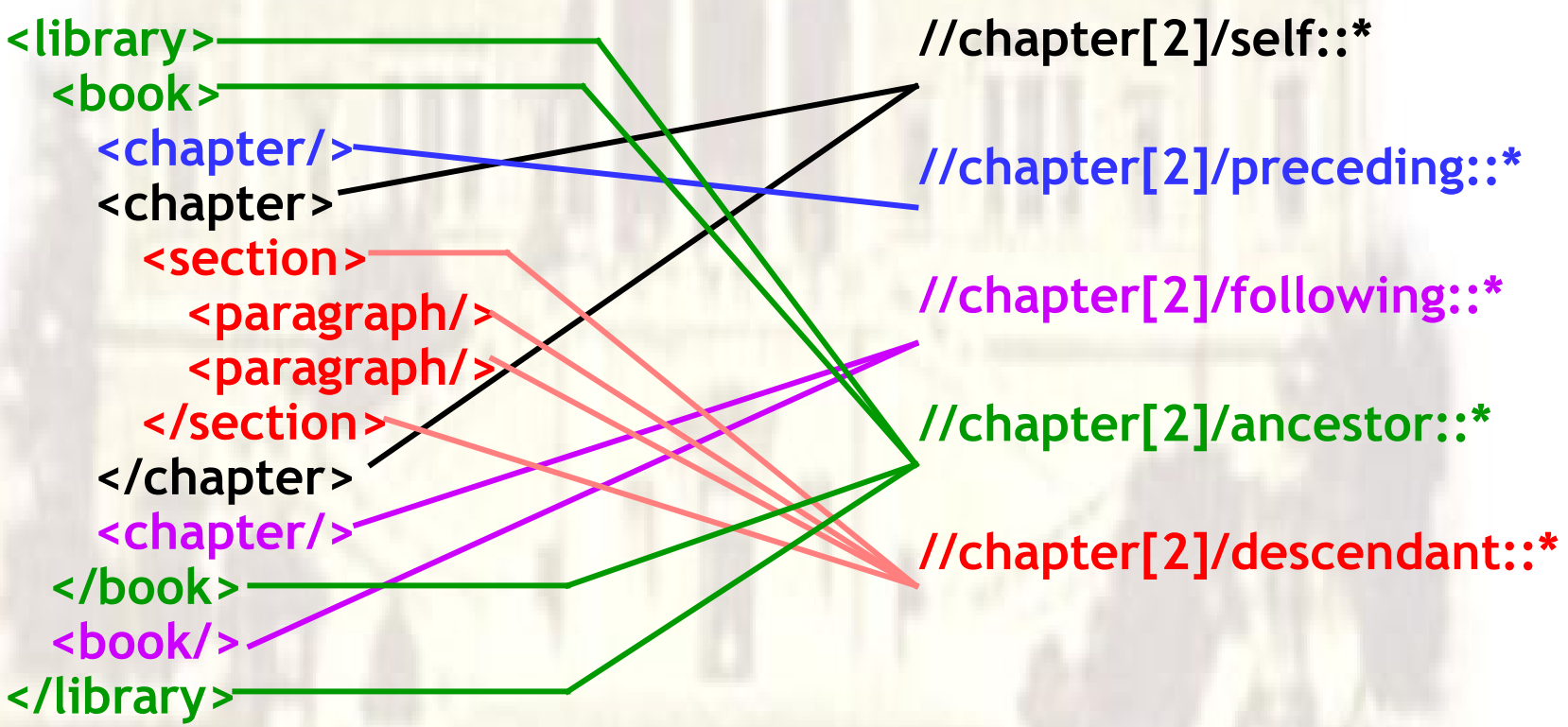
- Example: //chapter[normalize-space(@num)="3"]

Axes

An axis (plural axes) is a set of nodes relative to a given node; X::Y means "choose Y from the X axis"

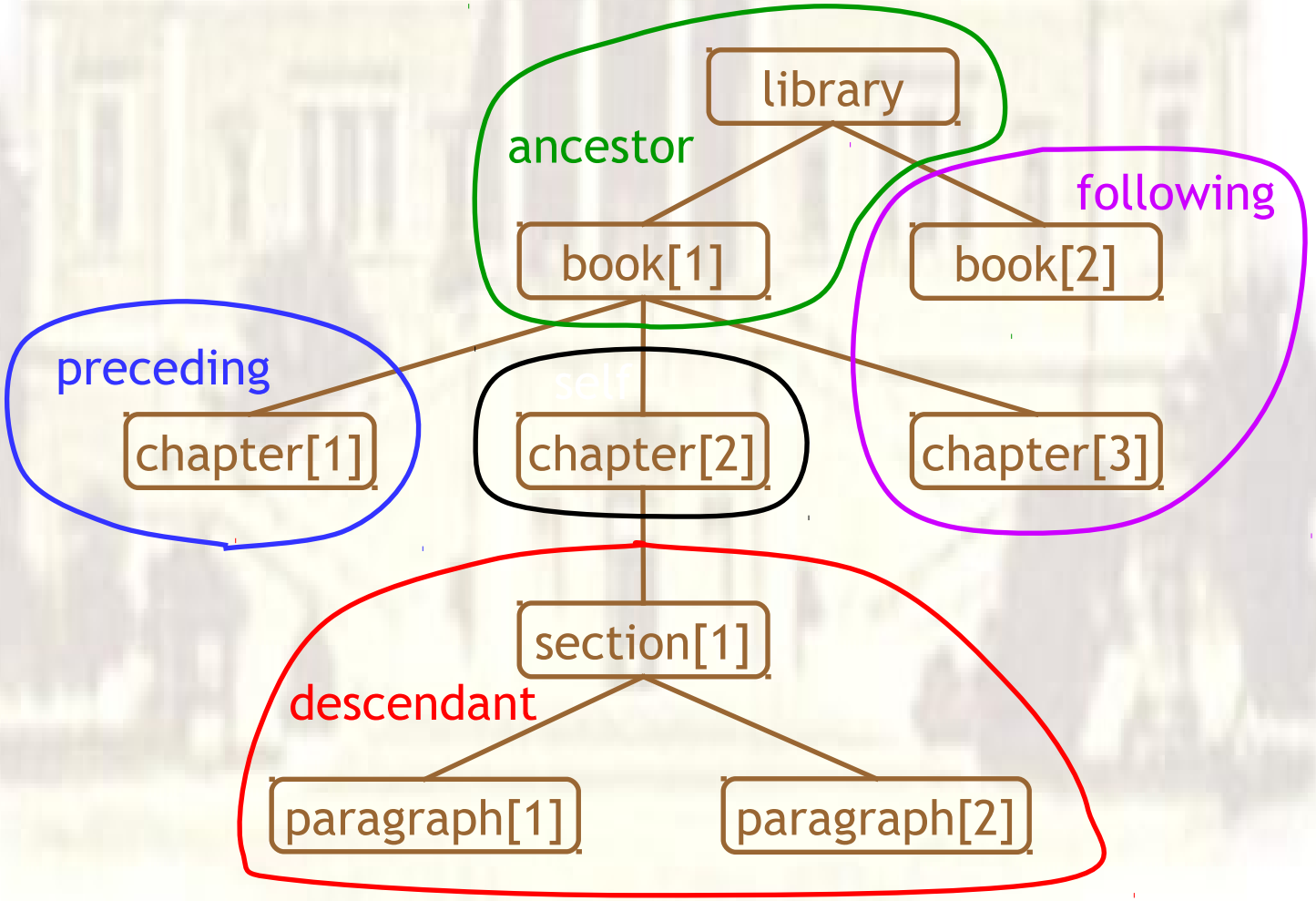
- self:: is the set of current nodes (not too useful)
- self::node() is the current node
- child:: is the default, so /child::X is the same as /X
- parent:: is the parent of the current node
- ancestor:: is all ancestors of the current node, up to and including the root
- descendant:: is all descendants of the current node
  - (Note: never contains attribute or namespace nodes)
- preceding:: is everything before the current node in the entire XML document
- following:: is everything after the current node in the entire XML document

Axes (outline view)



Starting from a given node, the self, preceding, following, ancestor, and descendant axes form a partition of all the nodes (if we ignore attribute and namespace nodes)

Axes (tree view)



Axis examples

//book/descendant::* is all descendants of every book

//book/descendant::section is all section descendants of every book

//parent::* is every element that is a parent, i.e., is not a leaf

//section/parent::* is every parent of a section element

//parent::chapter is every chapter that is a parent, i.e., has children

/library/book[3]/following::* is everything after the third book in the library

More axes

ancestor-or-self:: ancestors plus the current node

descendant-or-self:: descendants plus the current node

attribute:: is all attributes of the current node

namespace:: is all namespace nodes of the current node

preceding:: is everything before the current node in the entire XML document

following-sibling:: is all siblings after the current node

Note: preceding-sibling:: and following-sibling:: do not apply to attribute nodes or namespace nodes

Abbreviations for axes

(none) is the same as child::

@ is the same as attribute::

. is the same as self::node()

.//X is the same as self::node()/descendant-or-self::node()/child::X

.. is the same as parent::node()

../X is the same as parent::node()/child::X

// is the same as /descendant-or-self::node()/

//X is the same as /descendant-or-self::node()/child::X

Arithmetic expressions

+	add
-	subtract
*	multiply
div	(not /) divide
mod	modulo (remainder)

Equality tests

= means "equal to" (Notice it's *not* ==)

!= means "not equal to"

But it's not that simple!

value = *node-set* will be true if the *node-set contains any node* with a value that matches *value*

value != *node-set* will be true if the *node-set contains any node* with a value that does *not* match *value*

Hence,

value = *node-set* and *value* != *node-set* may both be true at the same time!

Other boolean operators

and (infix operator)

or (infix operator)

- Example: count = 0 or count = 1

not() (function)

The following are used for *numerical* comparisons only:

<	"less than"	Some places may require <
<=	"less than	Some places may require <= or equal to"
>	"greater than"	Some places may require >
>=	"greater than	Some places may require >= or equal to"

Some XPath functions

XPath contains a number of functions on node sets, numbers, and strings; here are a few of them:

count(*elem*) counts the number of selected elements

Example: //chapter[count(section)=1] selects chapters with exactly two section children

name() returns the name of the element

Example: //*[name()='section'] is the same as //section

starts-with(*arg1*, *arg2*) tests if *arg1* starts with *arg2*

Example: //*[starts-with(name(), 'sec']

contains(*arg1*, *arg2*) tests if *arg1* contains *arg2*

Example: //*[contains(name(), 'ect']