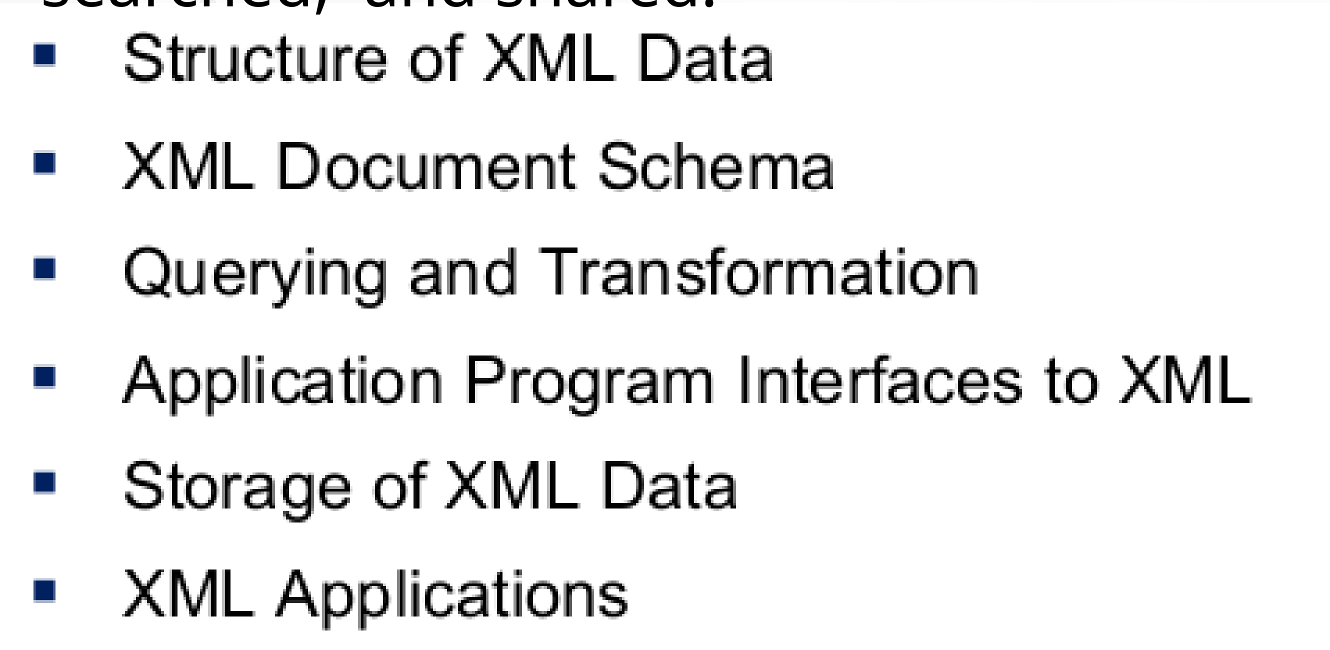
Complex Data Types

XML (Extensible Markup Language)

XML is a markup language similar to HTML, but without predefined tags to use. Instead, you define your own tags designed specifically for your needs.

This is a powerful way to store data in a format that can be stored, searched, and shared.

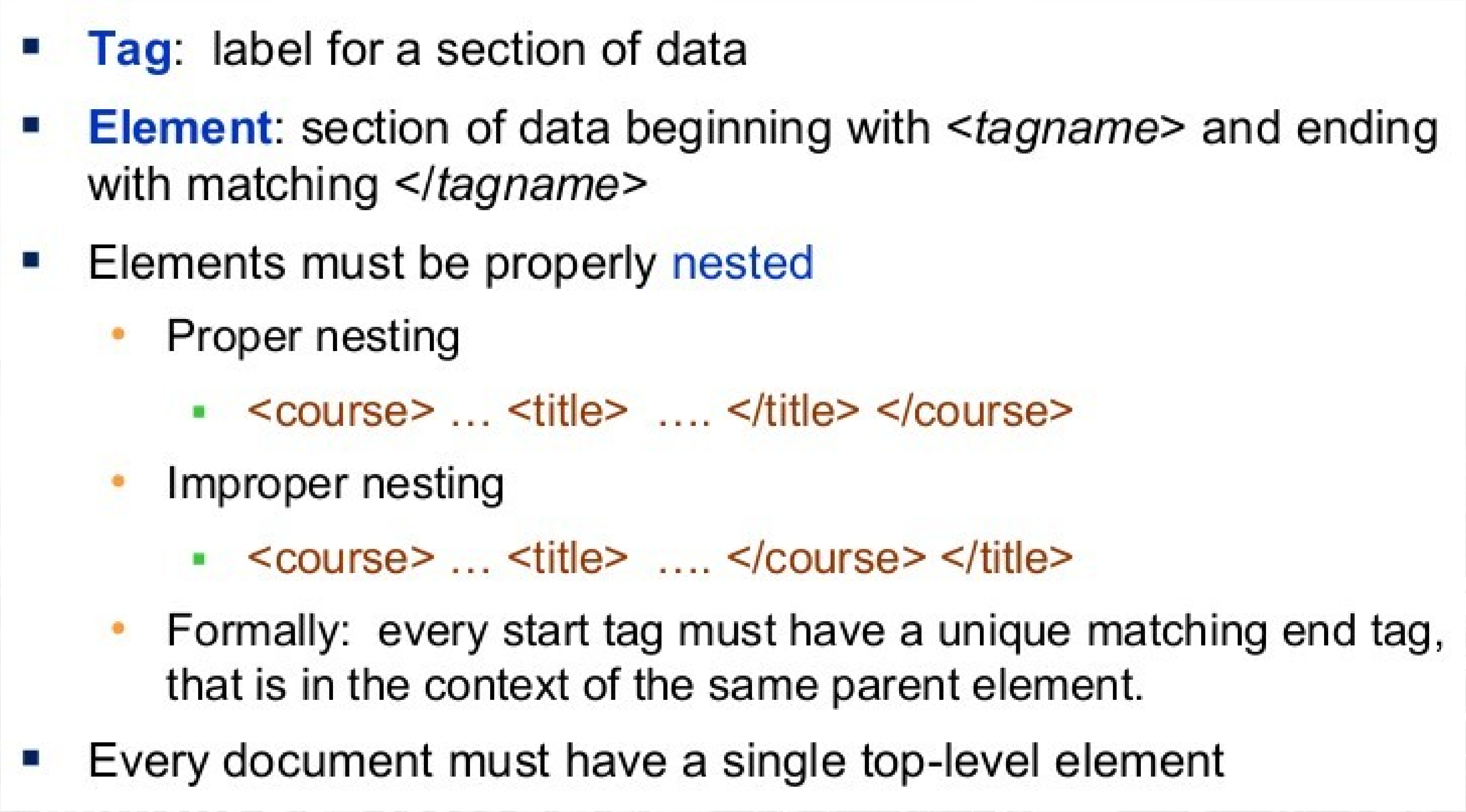


- Structure of XML Data
- XML Document Schema
- Querying and Transformation
- Application Program Interfaces to XML
- Storage of XML Data
- XML Applications

- The ability to specify new tags, and to create nested tag structures make XML a great way to exchange **data**, not just documents.
  - Much of the use of XML has been in data exchange applications, not as a replacement for HTML
- Tags make data (relatively) self-documenting
  - E.g.

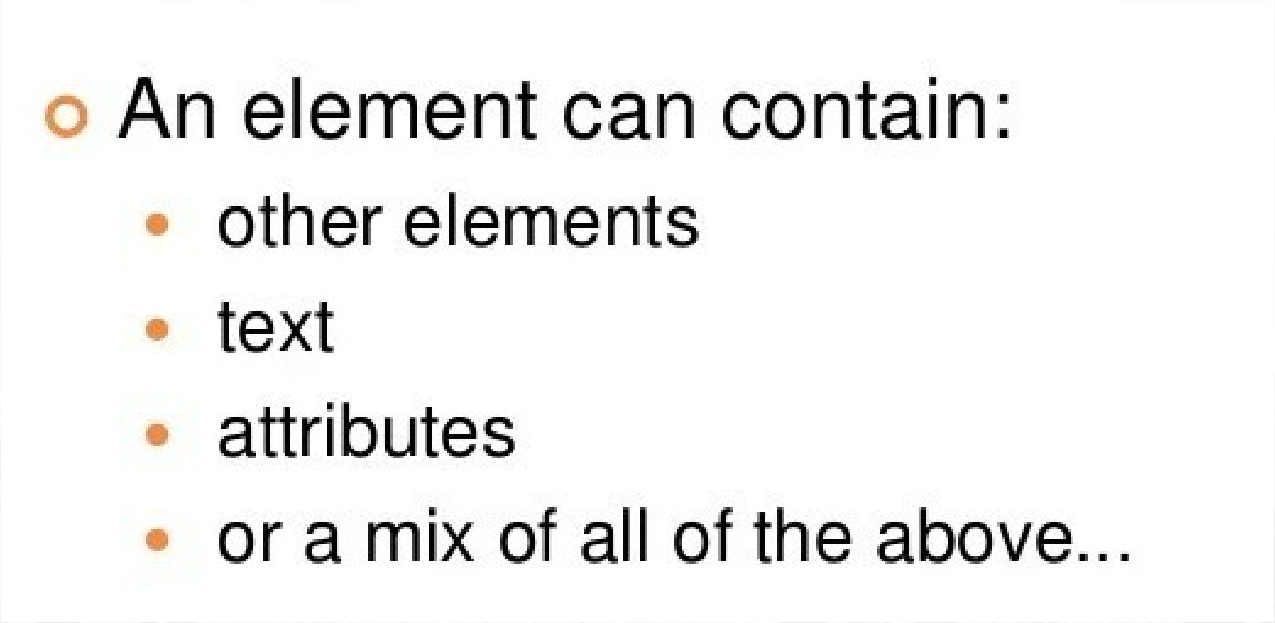
```
<university>
      <department>
        <dept_name> Comp. Sci. </dept_name>
        <building> Taylor </building>
        <budget> 100000 </budget>
     </department>
     <course>
         <course_id> CS-101 </course_id>
         <title> Intro. to Computer Science </title>
         <dept_name> Comp. Sci </dept_name>
         <credits> 4 </credits>
      </course>

</university>
```

Structure of XML data



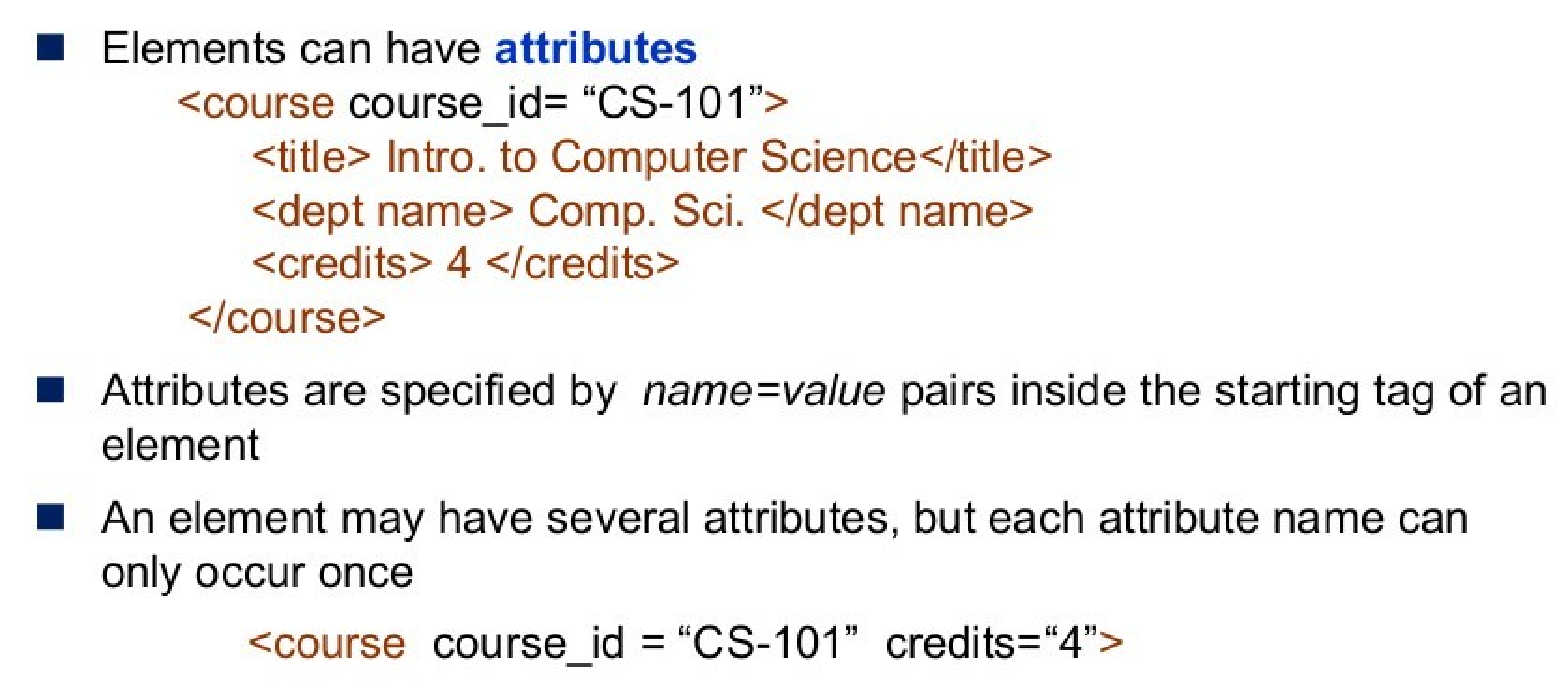
- **Tag**: label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly nested
  - Proper nesting
    - <course> … <title> …. </title> </course>
  - Improper nesting
    - <course> … <title> …. </course> </title>
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Every document must have a single top-level element

XML Element



- An element can contain:
  - other elements
  - text
  - attributes
  - or a mix of all of the above...

Attributes



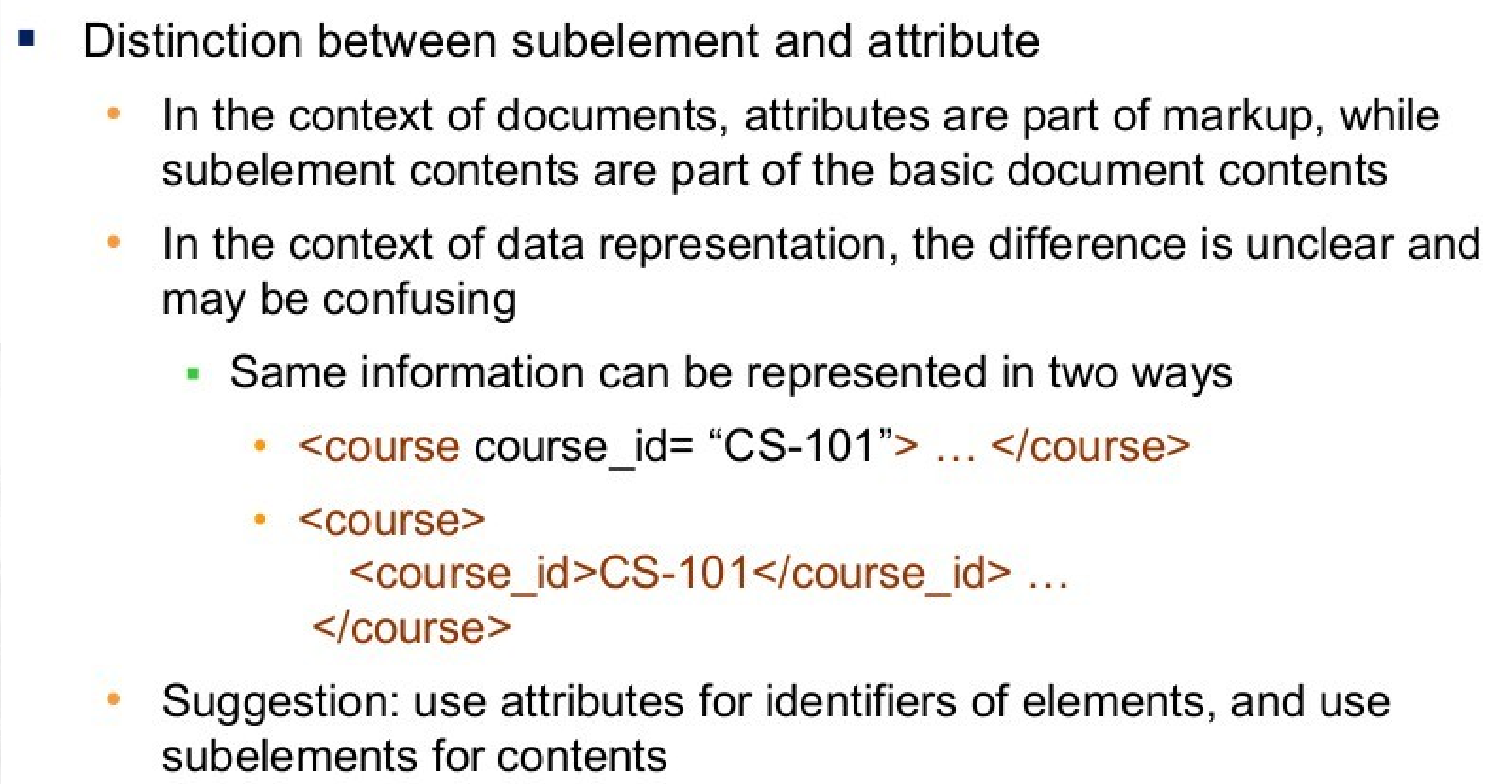
- Elements can have **attributes**

```
<course course_id= “CS-101”>
    <title> Intro. to Computer Science</title>
    <dept name> Comp. Sci. </dept name>
    <credits> 4 </credits>
</course>
```

- Attributes are specified by *name=value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once

```
<course  course_id = “CS-101”  credits=“4”>
```

Attributes vs. Subelements



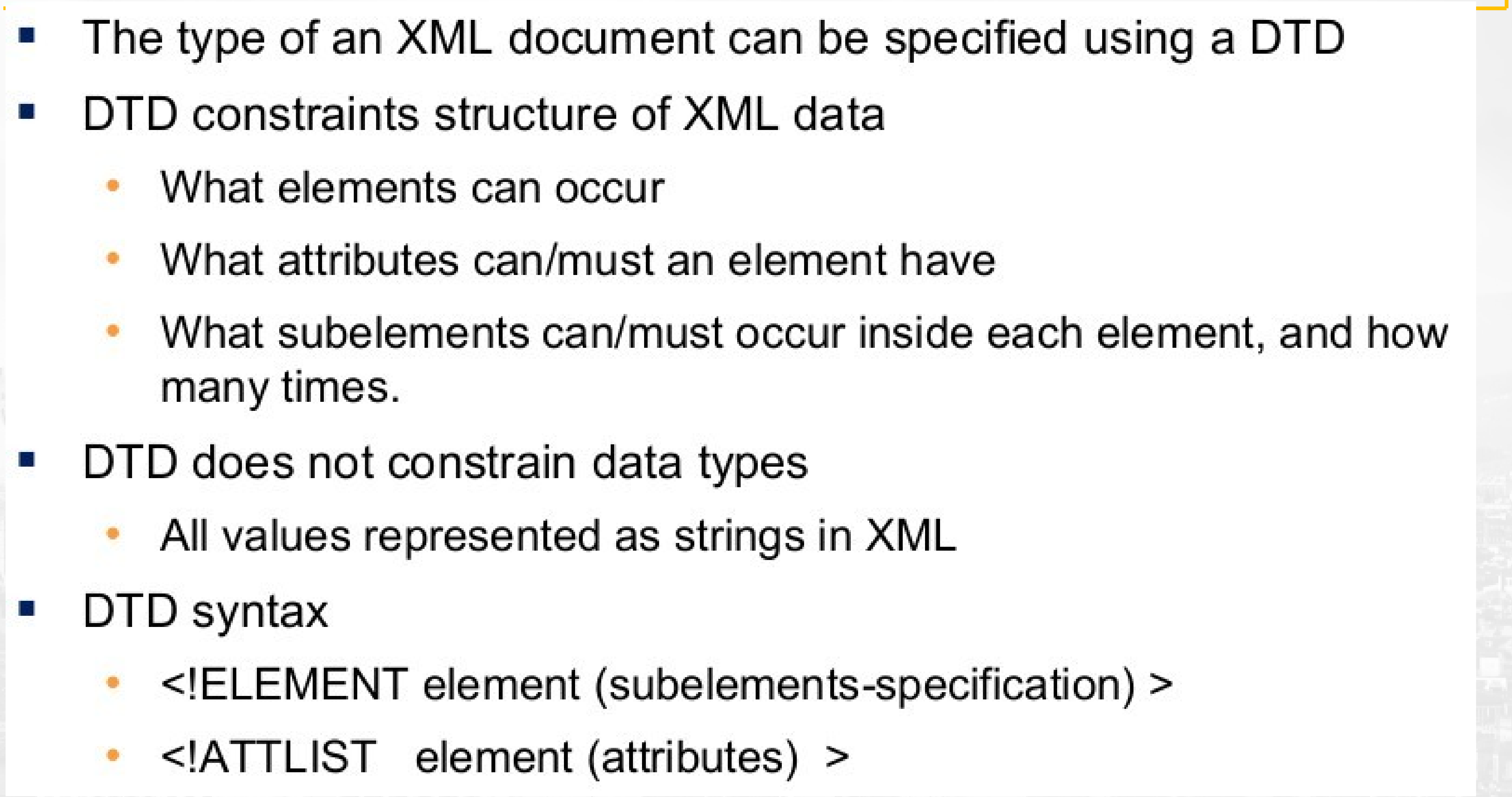
- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways
      - <course course_id= “CS-101”> … </course>
      - <course>
          <course_id>CS-101</course_id> …
        </course>
  - Suggestion: use attributes for identifiers of elements, and use subelements for contents

Namespaces

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique-name:element-name
- Avoid using long unique names all over document by using XML Namespaces

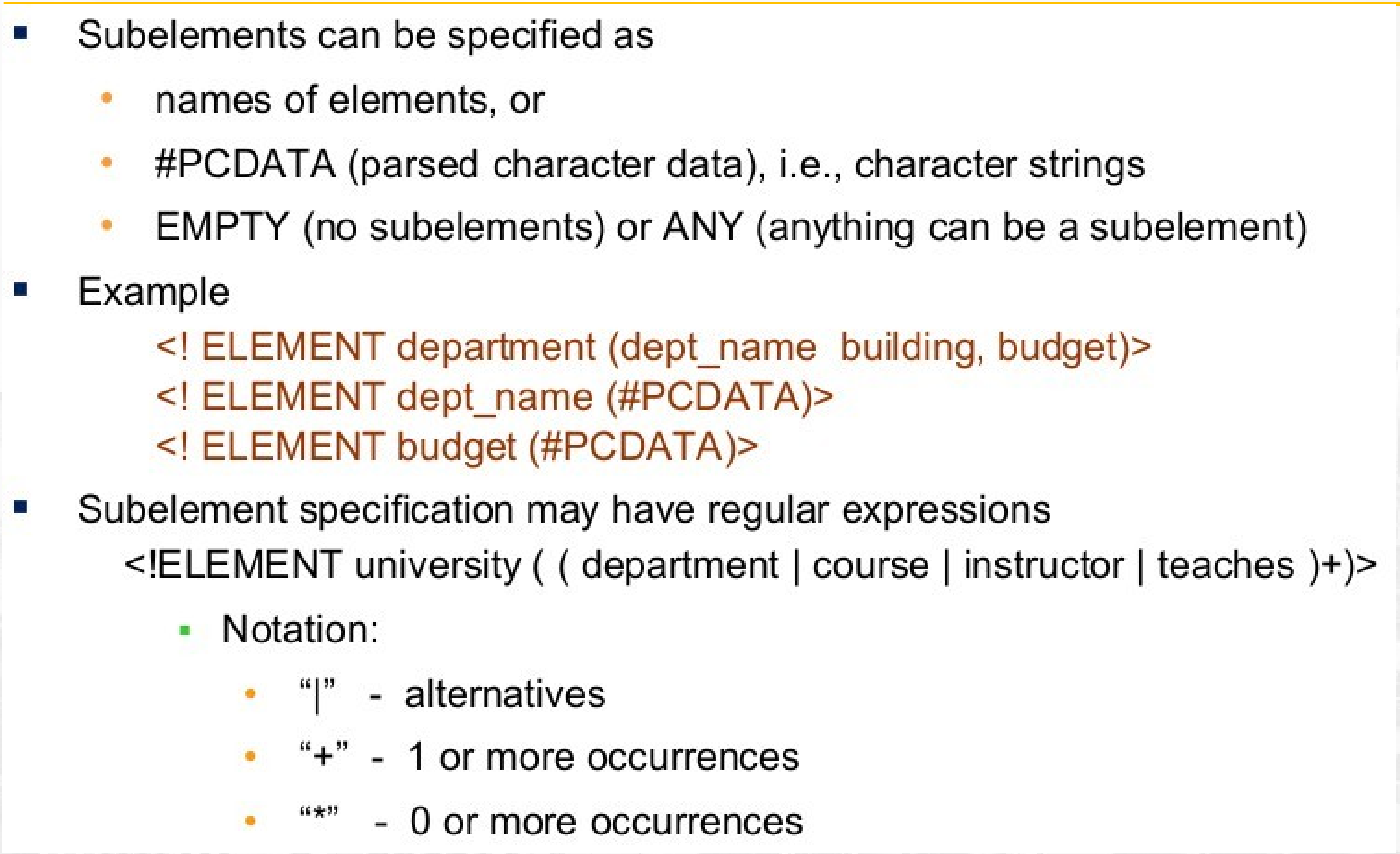
```
<university xmlns:yale="http://www.yale.edu">
      …
       <yale:course>
           <yale:course_id> CS-101 </yale:course_id>
           <yale:title> Intro. to Computer Science</yale:title>
           <yale:dept_name> Comp. Sci. </yale:dept_name>
           <yale:credits> 4 </yale:credits>
       </yale:course>
      …
</university>
```

Document Type Definition (DTD)



- The type of an XML document can be specified using a DTD
- DTD constraints structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Element Specification in DTD



- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)
- Example

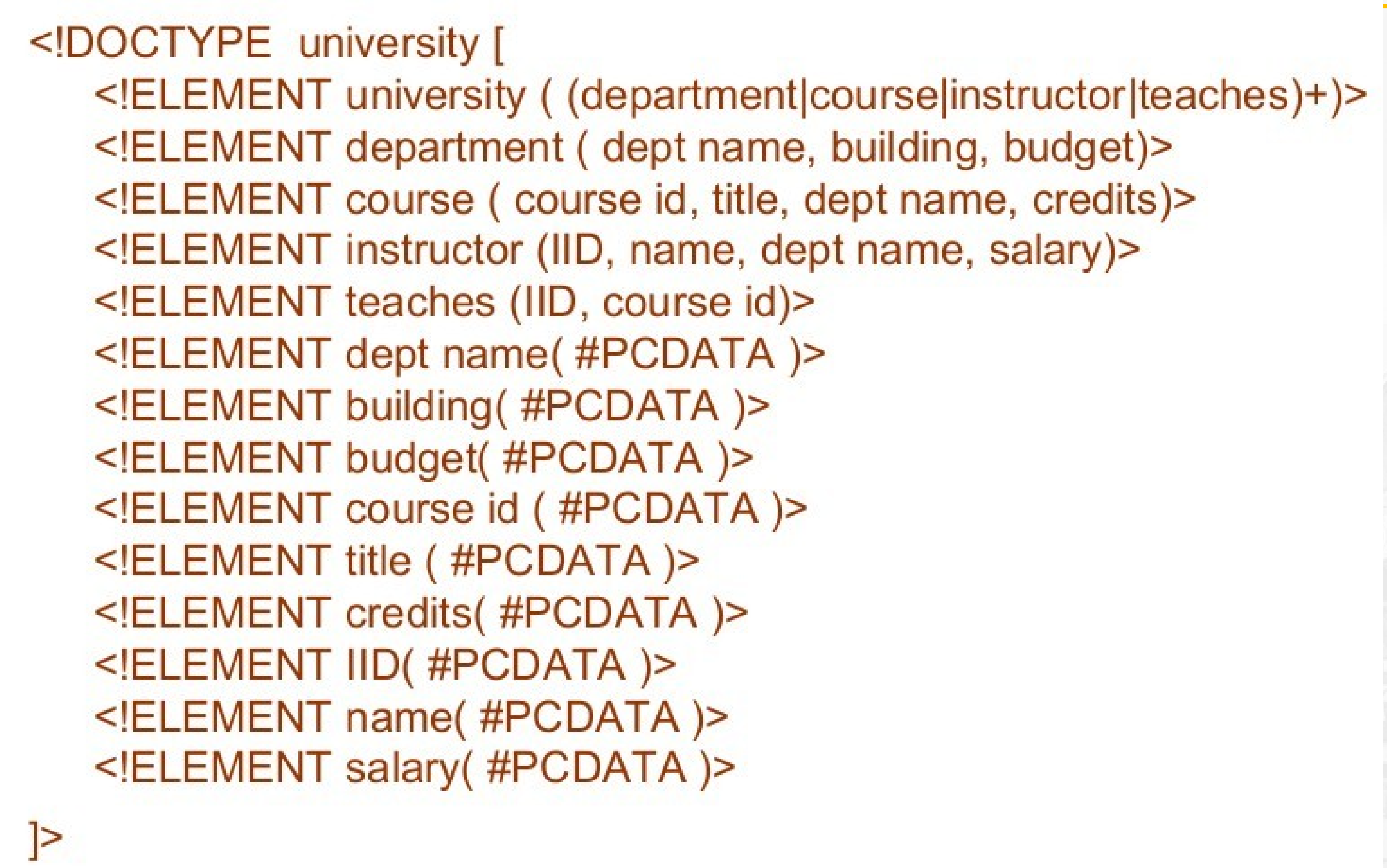
```
<! ELEMENT department (dept_name  building, budget)>
<! ELEMENT dept_name (#PCDATA)>
<! ELEMENT budget (#PCDATA)>
```

- Subelement specification may have regular expressions

```
<!ELEMENT university ( ( department | course | instructor | teaches )+)>
```

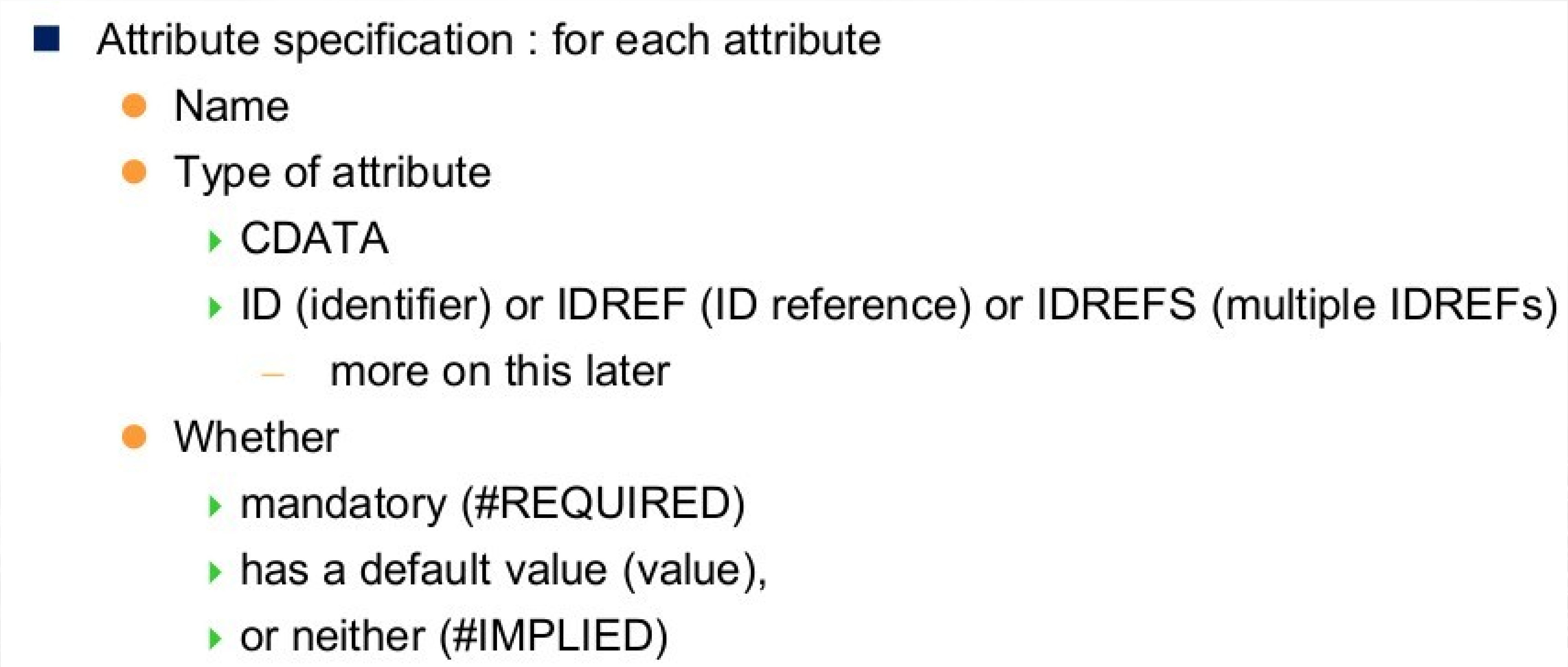
  - Notation:
    - "|" - alternatives
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences

University DTD



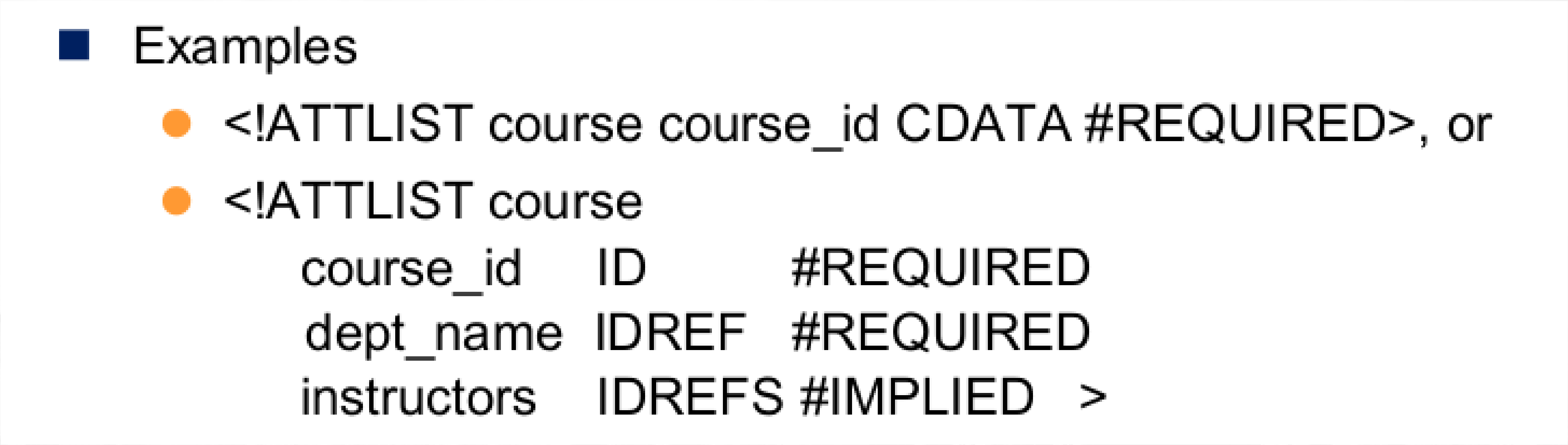
```
<!DOCTYPE  university [
     <!ELEMENT university ( (department|course|instructor|teaches)+)>
     <!ELEMENT department ( dept name, building, budget)>
     <!ELEMENT course ( course id, title, dept name, credits)>
     <!ELEMENT instructor (IID, name, dept name, salary)>
     <!ELEMENT teaches (IID, course id)>
     <!ELEMENT dept name( #PCDATA )>
     <!ELEMENT building( #PCDATA )>
     <!ELEMENT budget( #PCDATA )>
     <!ELEMENT course id ( #PCDATA )>
     <!ELEMENT title ( #PCDATA )>
     <!ELEMENT credits( #PCDATA )>
     <!ELEMENT IID( #PCDATA )>
     <!ELEMENT name( #PCDATA )>
     <!ELEMENT salary( #PCDATA )>
]>
```

Attribute Specification in DTD



- Attribute specification : for each attribute
  - Name
  - Type of attribute
    - CDATA
    - ID (identifier) or IDREF (ID reference) or IDREFS (multiple IDREFs)
      - more on this later
  - Whether
    - mandatory (#REQUIRED)
    - has a default value (value),
    - or neither (#IMPLIED)

Attribute Specification in DTD : examples



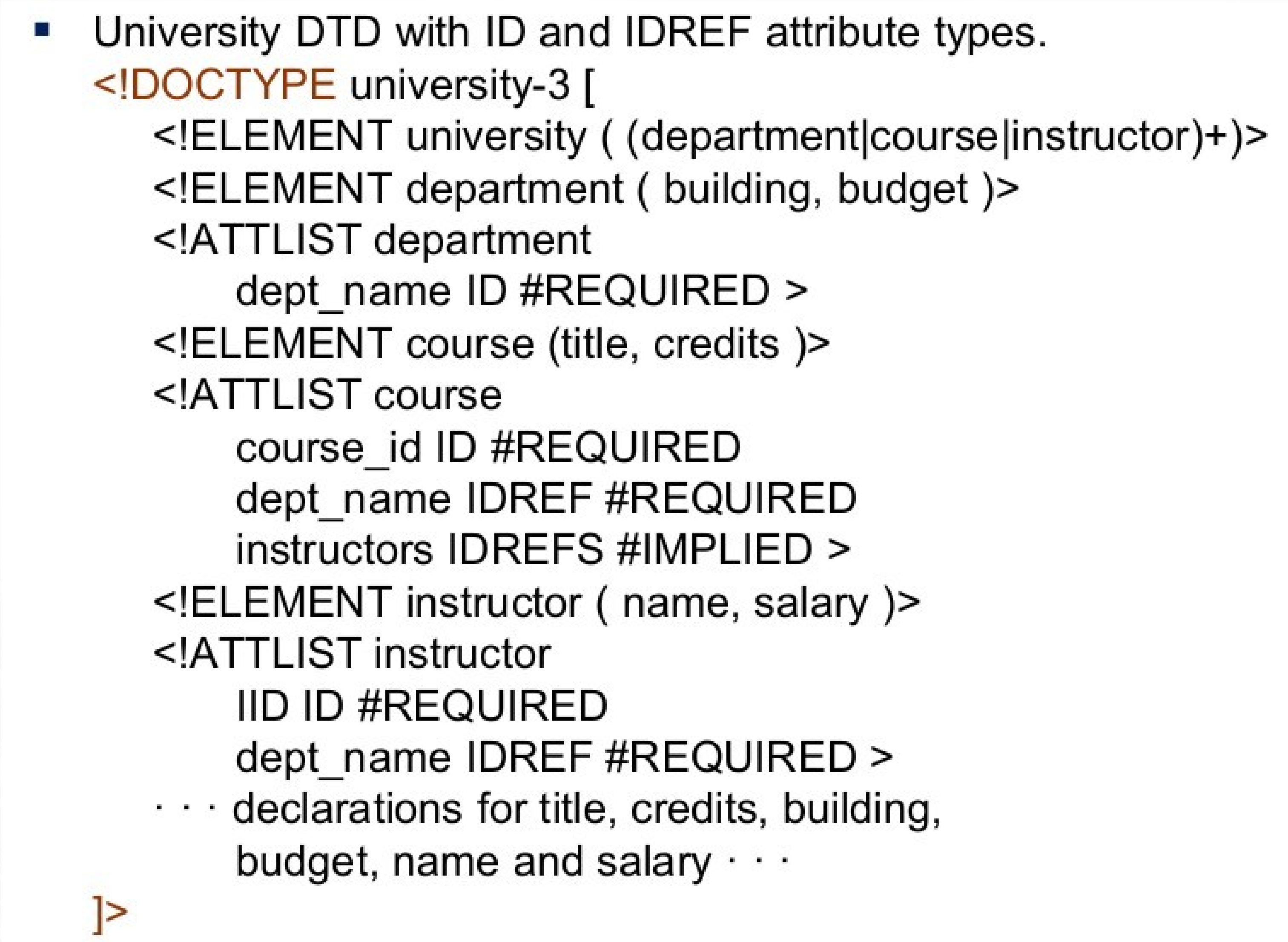
- Examples
  - <!ATTLIST course course_id CDATA #REQUIRED>, or
  - <!ATTLIST course

```
course_id     ID         #REQUIRED
dept_name     IDREF      #REQUIRED
instructors   IDREFS #IMPLIED   >
```

IDs and IDREFs

- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct
  - Thus the ID attribute value is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document
- An attribute of type IDREFS contains a set of (0 or more) ID values. Each ID value must contain the ID value of an element in the same document

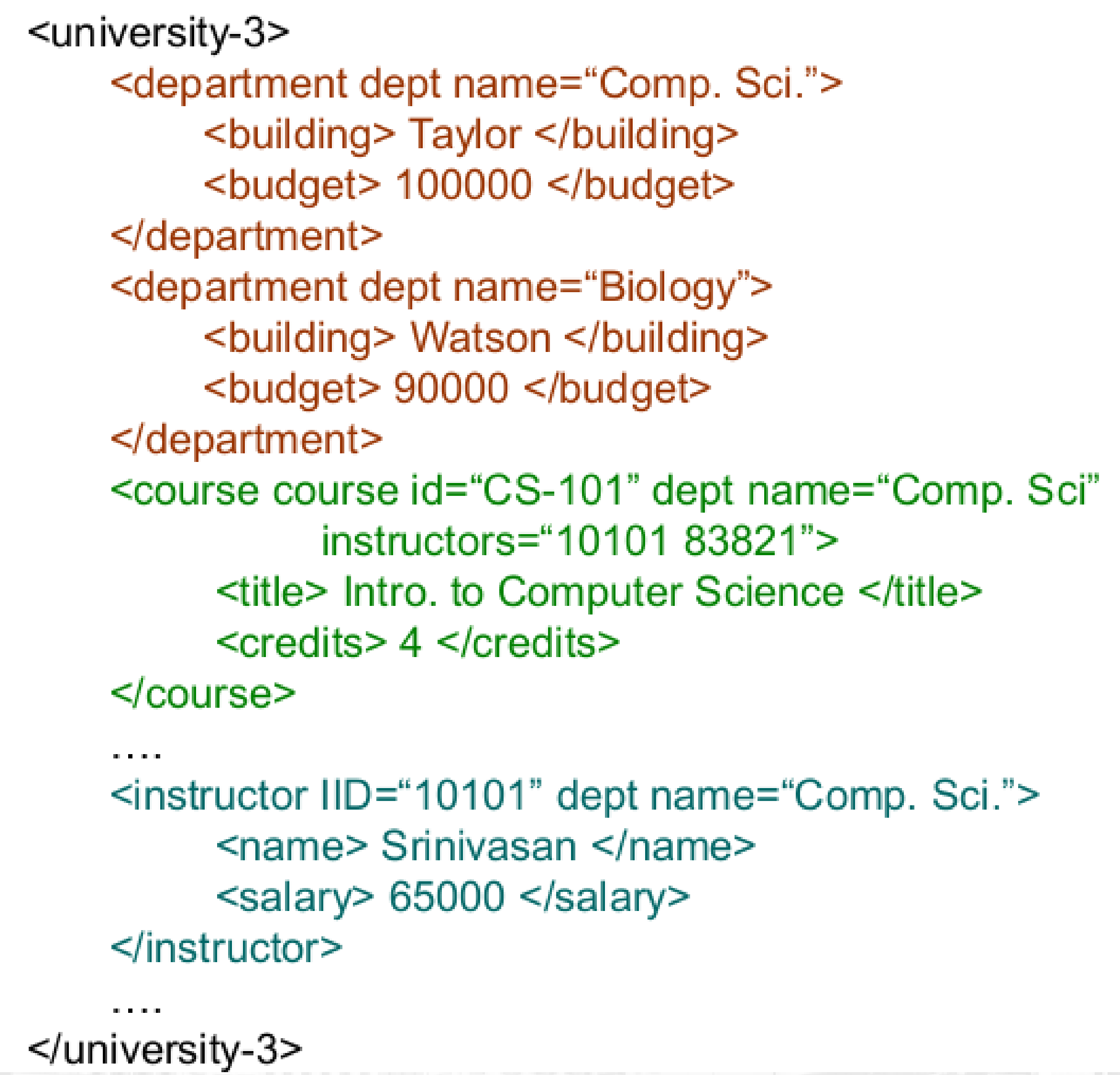
University DTD with Attributes



- University DTD with ID and IDREF attribute types.

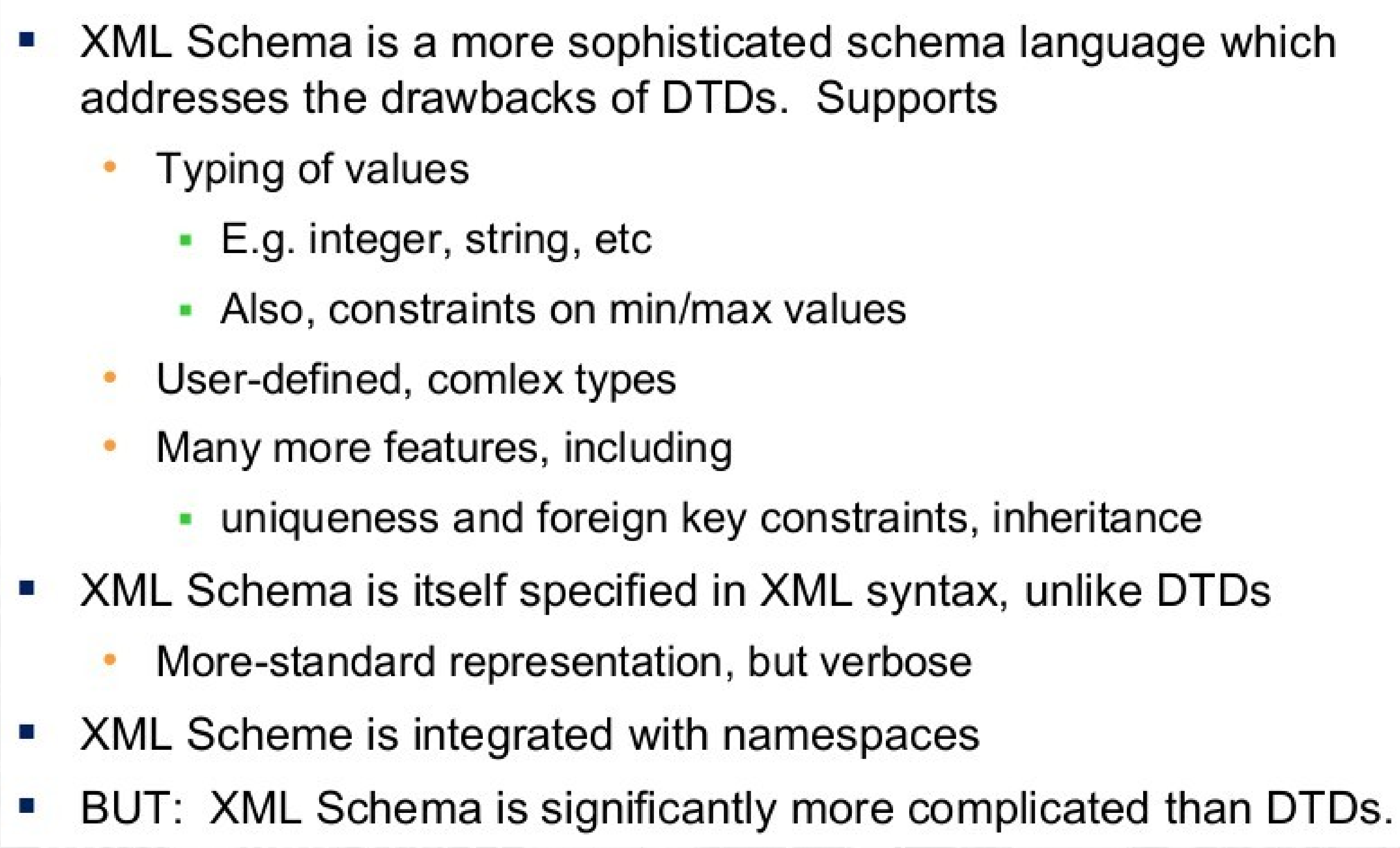
```
<!DOCTYPE university-3 [
    <!ELEMENT university ( (department|course|instructor)+)>
    <!ELEMENT department ( building, budget )>
    <!ATTLIST department
          dept_name ID #REQUIRED >
    <!ELEMENT course (title, credits )>
    <!ATTLIST course
          course_id ID #REQUIRED
          dept_name IDREF #REQUIRED
          instructors IDREFS #IMPLIED >
    <!ELEMENT instructor ( name, salary )>
    <!ATTLIST instructor
          IID ID #REQUIRED
          dept_name IDREF #REQUIRED >
    · · · declarations for title, credits, building,
          budget, name and salary · · ·
]>
```

XML data with ID and IDREF attributes



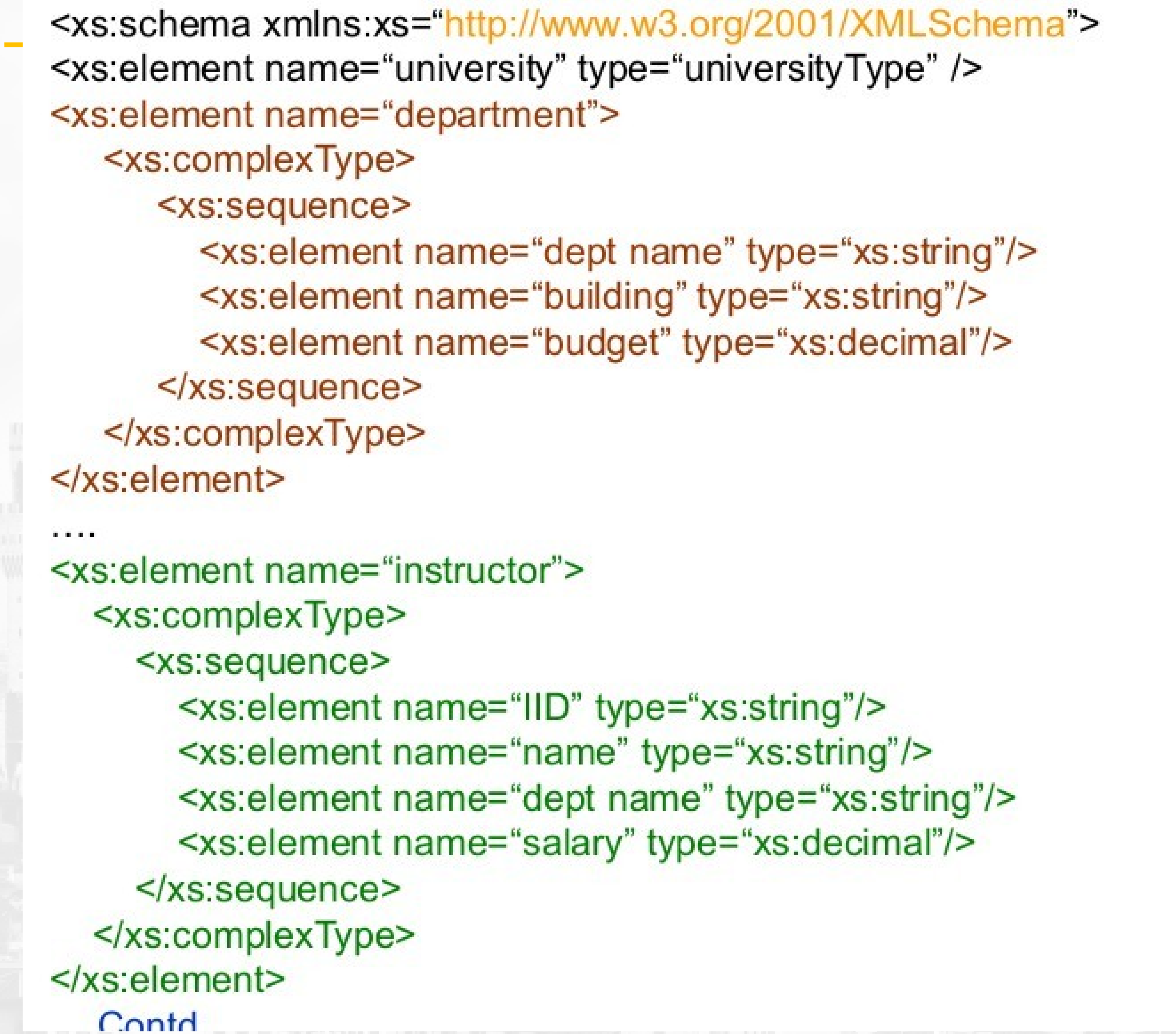
```
<university-3>
      <department dept name="Comp. Sci.">
            <building> Taylor </building>
            <budget> 100000 </budget>
      </department>
      <department dept name="Biology">
            <building> Watson </building>
            <budget> 90000 </budget>
      </department>
      <course course id="CS-101" dept name="Comp. Sci"
                    instructors="10101 83821">
             <title> Intro. to Computer Science </title>
             <credits> 4 </credits>
      </course>
      ….
      <instructor IID="10101" dept name="Comp. Sci.">
             <name> Srinivasan </name>
             <salary> 65000 </salary>
      </instructor>
      ….
</university-3>
```

XML Schema



- XML Schema is a more sophisticated schema language which addresses the drawbacks of DTDs. Supports
  - Typing of values
    - E.g. integer, string, etc
    - Also, constraints on min/max values
  - User-defined, comlex types
  - Many more features, including
    - uniqueness and foreign key constraints, inheritance
- XML Schema is itself specified in XML syntax, unlike DTDs
  - More-standard representation, but verbose
- XML Scheme is integrated with namespaces
- BUT: XML Schema is significantly more complicated than DTDs.

XML Schema Version of Univ. DTD



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="university" type="universityType" />
<xs:element name="department">
     <xs:complexType>
          <xs:sequence>
              <xs:element name="dept name" type="xs:string"/>
              <xs:element name="building" type="xs:string"/>
              <xs:element name="budget" type="xs:decimal"/>
          </xs:sequence>
     </xs:complexType>
</xs:element>
….
<xs:element name="instructor">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="IID" type="xs:string"/>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="dept name" type="xs:string"/>
            <xs:element name="salary" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
     Contd
```

XML Schema Version of Univ. DTD (Cont.)

```
….
<xs:complexType name=“UniversityType”>
    <xs:sequence>
        <xs:element ref=“department” minOccurs=“0” maxOccurs=“unbounded”/>
        <xs:element ref=“course” minOccurs=“0” maxOccurs=“unbounded”/>
        <xs:element ref=“instructor” minOccurs=“0” maxOccurs=“unbounded”/>
        <xs:element ref=“teaches” minOccurs=“0” maxOccurs=“unbounded”/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

- Choice of “xs:” was ours -- any other namespace prefix could be chosen
- Element “university” has type “universityType”, which is defined separately
    - xs:complexType is used later to create the named complex type “UniversityType”

More features of XML Schema

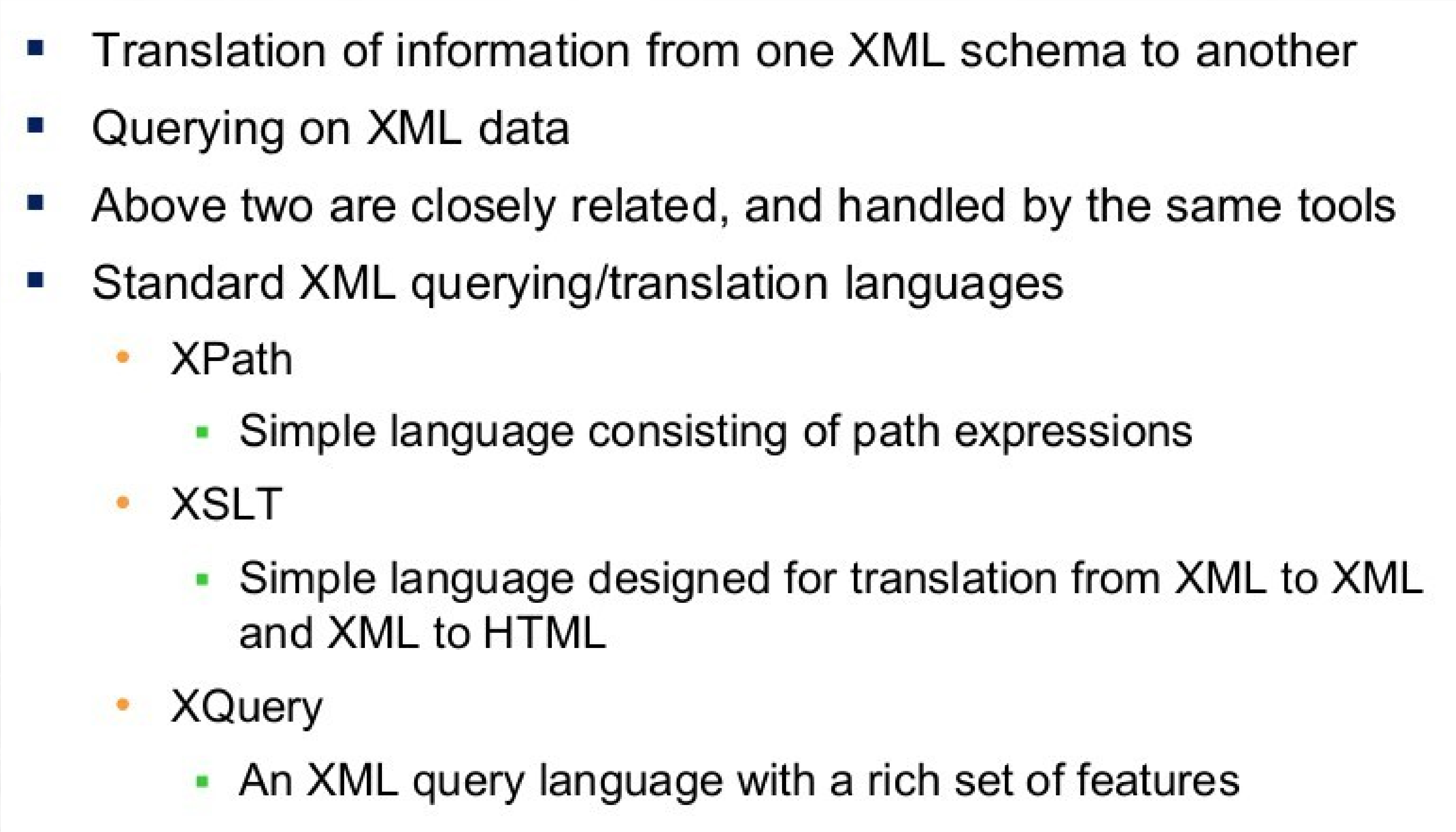
- Attributes specified by xs:attribute tag:
  - <xs:attribute name = “dept_name”/>
  - adding the attribute use = “required” means value must be specified
- Key constraint: “department names form a key for department elements under the root university element:

```
<xs:key name = “deptKey”>
        <xs:selector xpath = “/university/department”/>
        <xs:field xpath = “dept_name”/>
<\xs:key>
```

- Foreign key constraint from course to department:

```
<xs:keyref name = “courseDeptFKey” refer=“deptKey”>
        <xs:selector xpath = “/university/course”/>
        <xs:field xpath = “dept_name”/>
<\xs:keyref>
```

Querying and Transforming XML Data

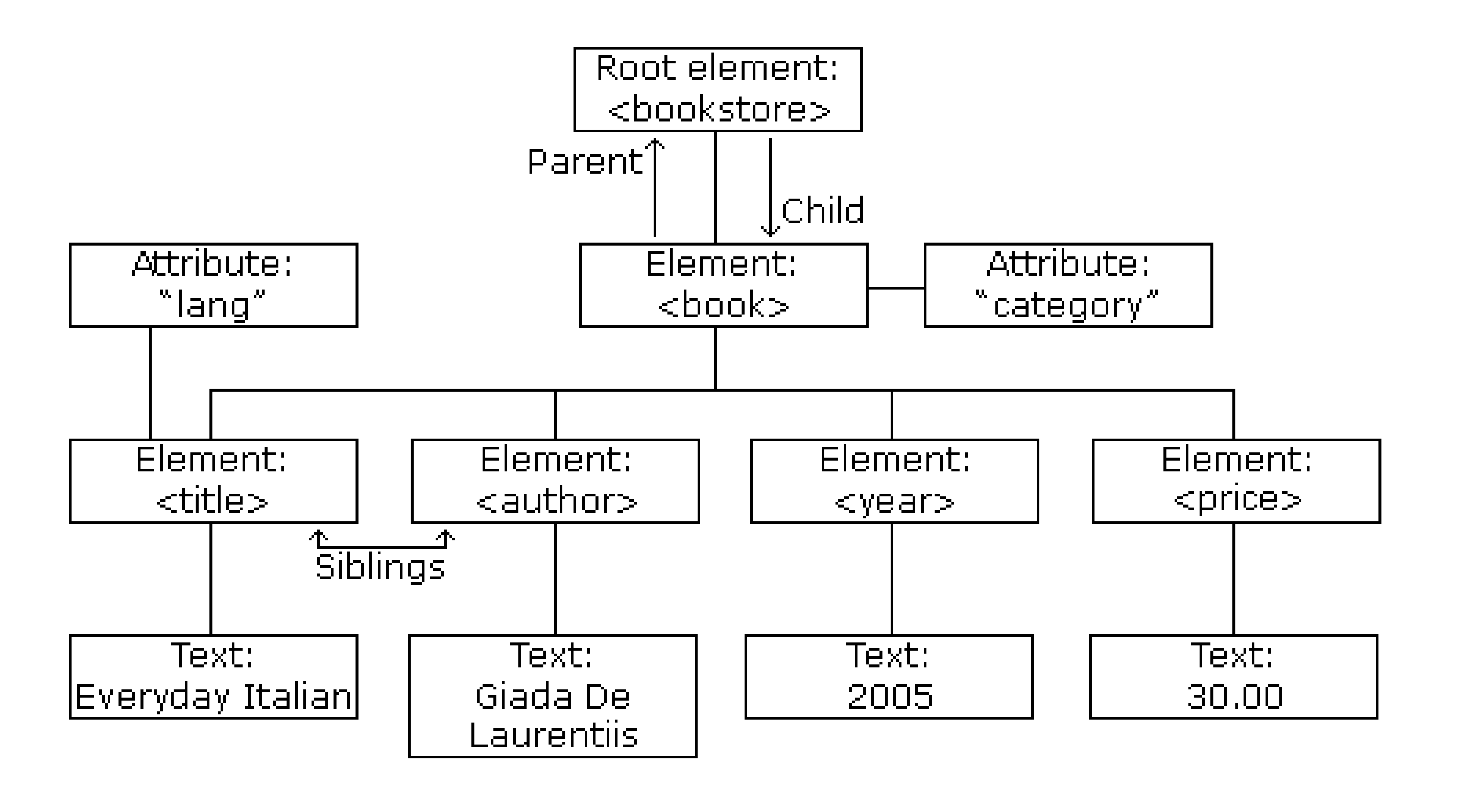


- Translation of information from one XML schema to another
- Querying on XML data
- Above two are closely related, and handled by the same tools
- Standard XML querying/translation languages
  - XPath
    - Simple language consisting of path expressions
  - XSLT
    - Simple language designed for translation from XML to XML and XML to HTML
  - XQuery
    - An XML query language with a rich set of features

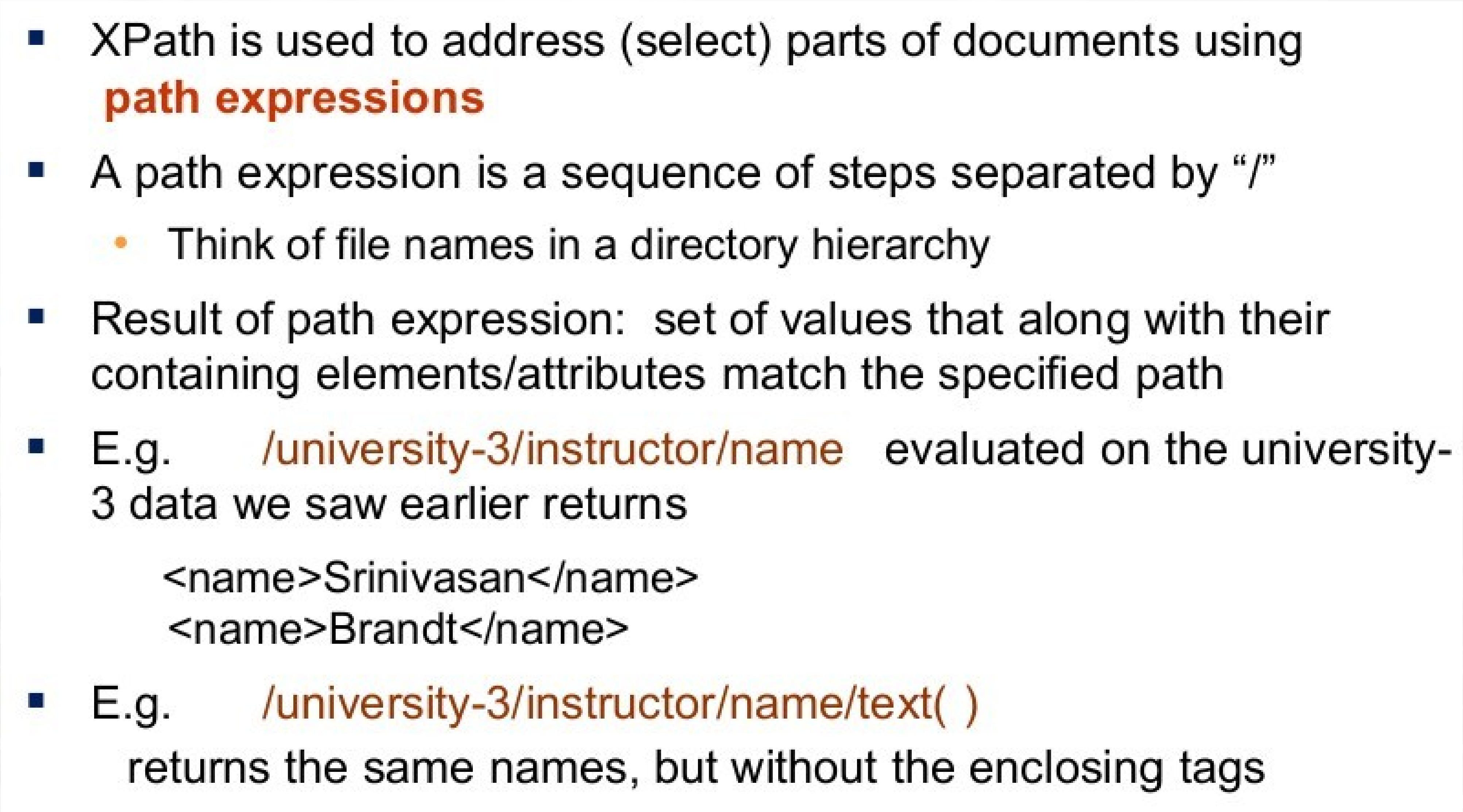
Tree Model of XML Data

- Query and transformation languages are based on a **tree model** of XML data
- An XML document is modeled as a tree, with **nodes** corresponding to elements and attributes
  - Element nodes have child nodes, which can be attributes or subelements
  - Text in an element is modeled as a text node child of the element
  - Children of a node are ordered according to their order in the XML document
  - Element and attribute nodes (except for the root node) have a single parent, which is an element node
  - The root node has a single child, which is the root element of the document

Tree Representation - Example



XPath



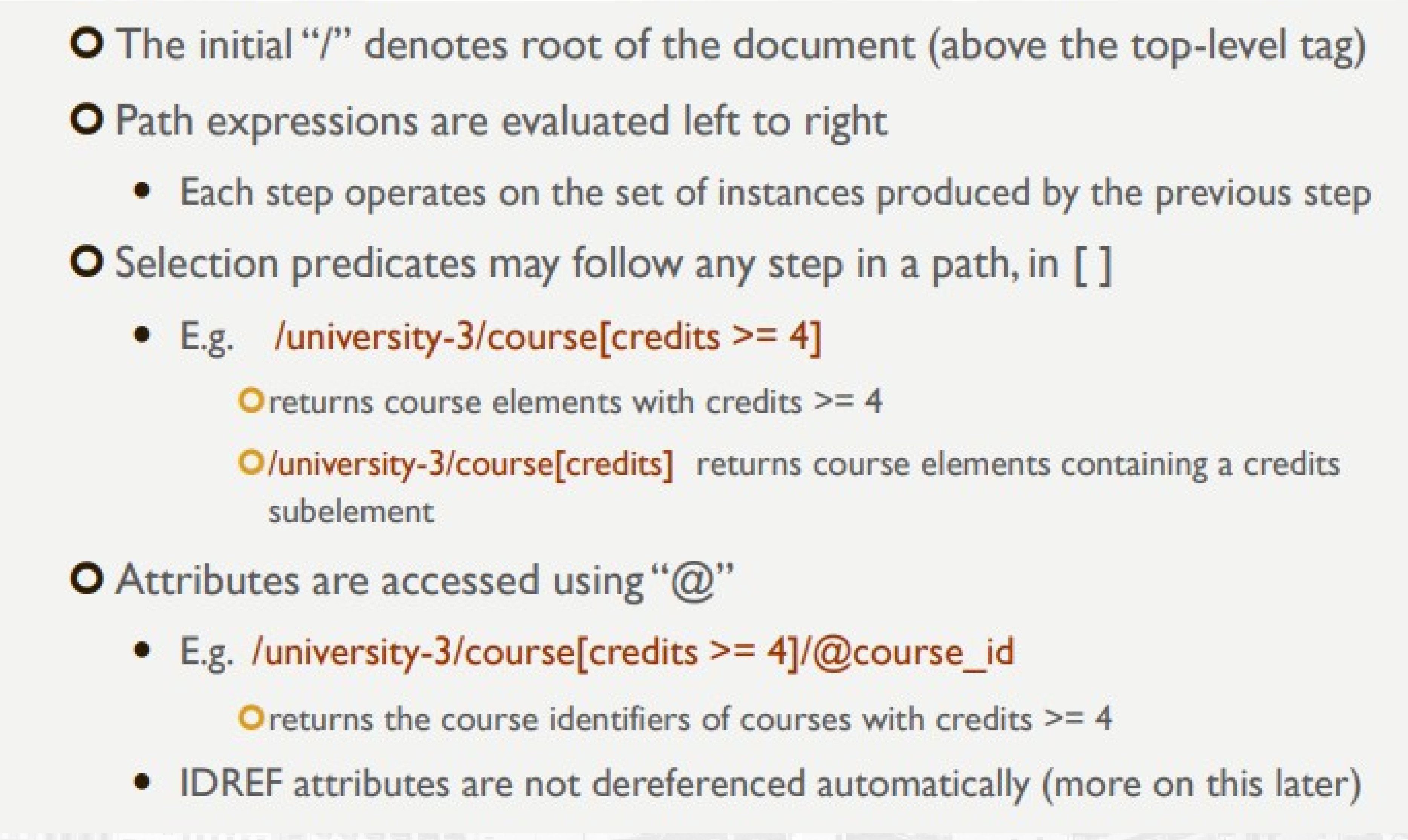
- XPath is used to address (select) parts of documents using **path expressions**
- A path expression is a sequence of steps separated by “/”
  - Think of file names in a directory hierarchy
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
- E.g. /university-3/instructor/name evaluated on the university-3 data we saw earlier returns

```
<name>Srinivasan</name>
<name>Brandt</name>
```

- E.g. /university-3/instructor/name/text( )

  returns the same names, but without the enclosing tags

XPath (Cont.)



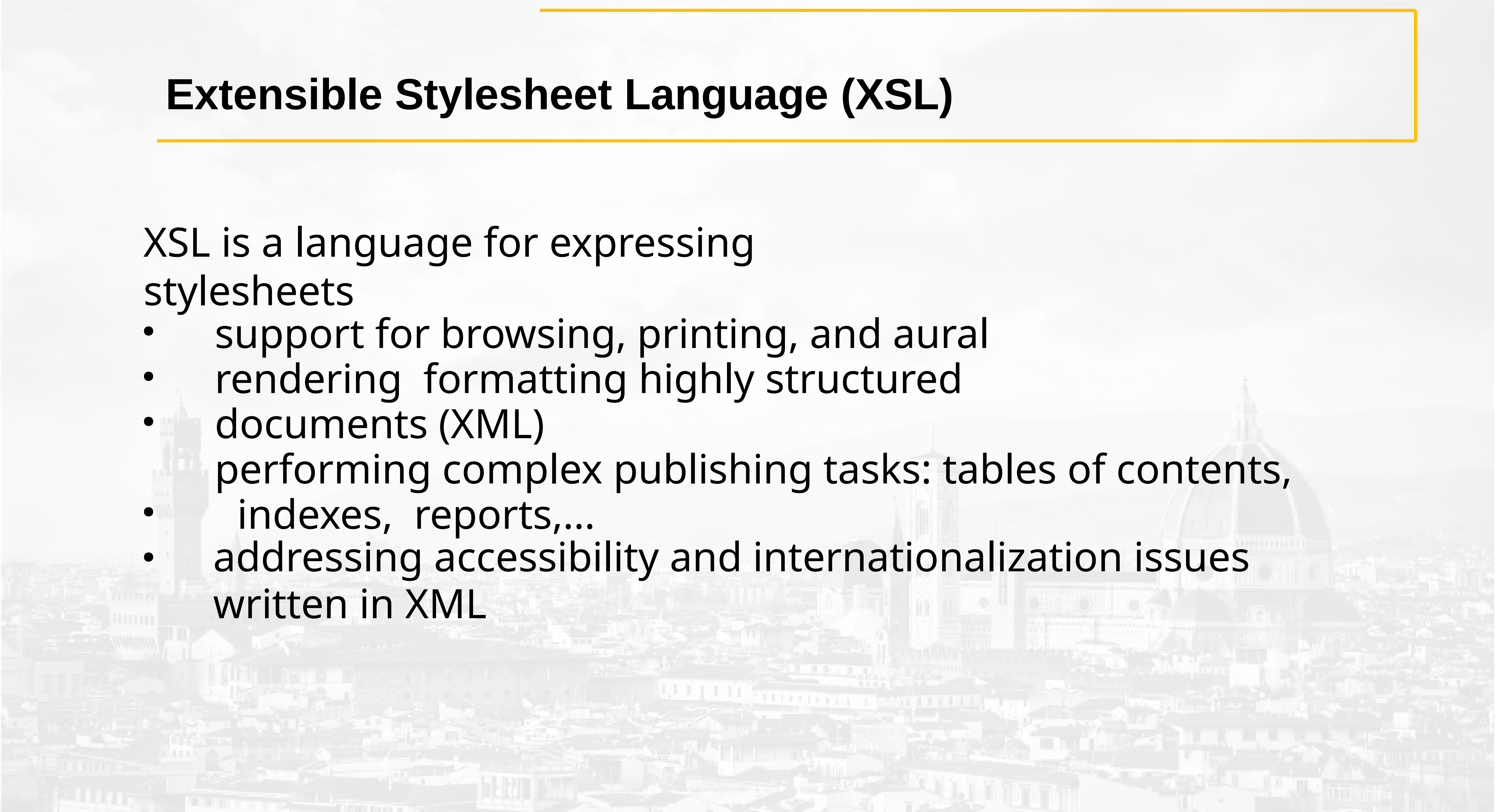
- The initial "/" denotes root of the document (above the top-level tag)
- Path expressions are evaluated left to right
  - Each step operates on the set of instances produced by the previous step
- Selection predicates may follow any step in a path, in [ ]
  - E.g. /university-3/course[credits >= 4]
    - returns course elements with credits >= 4
    - /university-3/course[credits] returns course elements containing a credits subelement
- Attributes are accessed using "@"
  - E.g. /university-3/course[credits >= 4]/@course_id
    - returns the course identifiers of courses with credits >= 4
  - IDREF attributes are not dereferenced automatically (more on this later)

Functions in XPath

- XPath provides several functions
    - The function count() at the end of a path counts the number of elements in the set generated by the path
        - E.g. /university-2/instructor[count(./teaches/course)> 2]
            - Returns instructors teaching more than 2 courses (on university-2 schema)
    - Also function for testing position (1, 2, ..) of node w.r.t. siblings
- Boolean connectives and and or and function not() can be used in predicates
- IDREFs can be referenced using function id()
    - id() can also be applied to sets of references such as IDREFS and even to strings containing multiple references separated by blanks
    - E.g. /university-3/course/id(@dept_name)
        - returns all department elements referred to from the dept_name attribute of course elements.

More XPath Features

- Operator “|” used to implement union
  - E.g. /university-3/course[@dept name=“Comp. Sci”] | /university-3/course[@dept name=“Biology”]
    - Gives union of Comp. Sci. and Biology courses
    - However, “|” cannot be nested inside other operators.
- “//” can be used to skip multiple levels of nodes
  - E.g. /university-3//name
    - finds any name element *anywhere* under the /university-3 element, regardless of the element in which it is contained.
- A step in the path can go to parents, siblings, ancestors and descendants of the nodes generated by the previous step, not just to the children
  - “//”, described above, is a short from for specifying “all descendants”
  - “..” specifies the parent.
- doc(name) returns the root of a named document

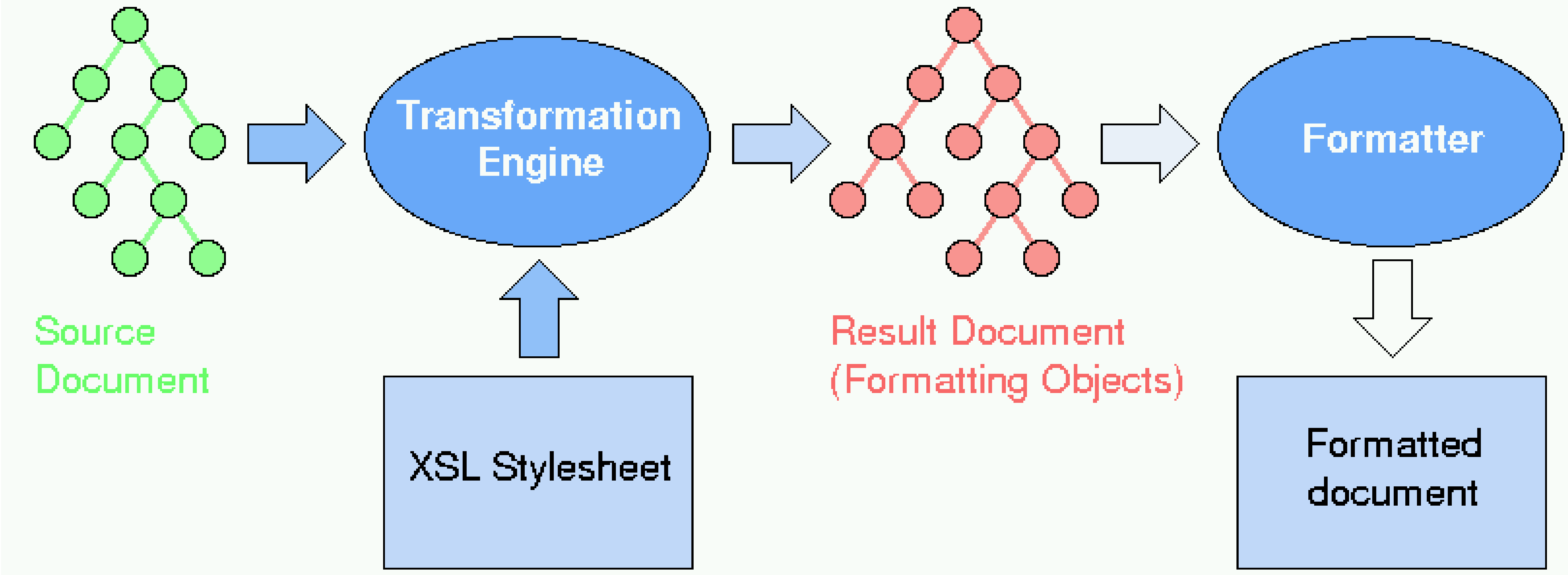


Extensible Stylesheet Language (XSL)

XSL is a language for expressing stylesheets

- support for browsing, printing, and aural
- rendering formatting highly structured
- documents (XML)
  performing complex publishing tasks: tables of contents,
- indexes, reports,...
- addressing accessibility and internationalization issues
  written in XML

XSL Architecture



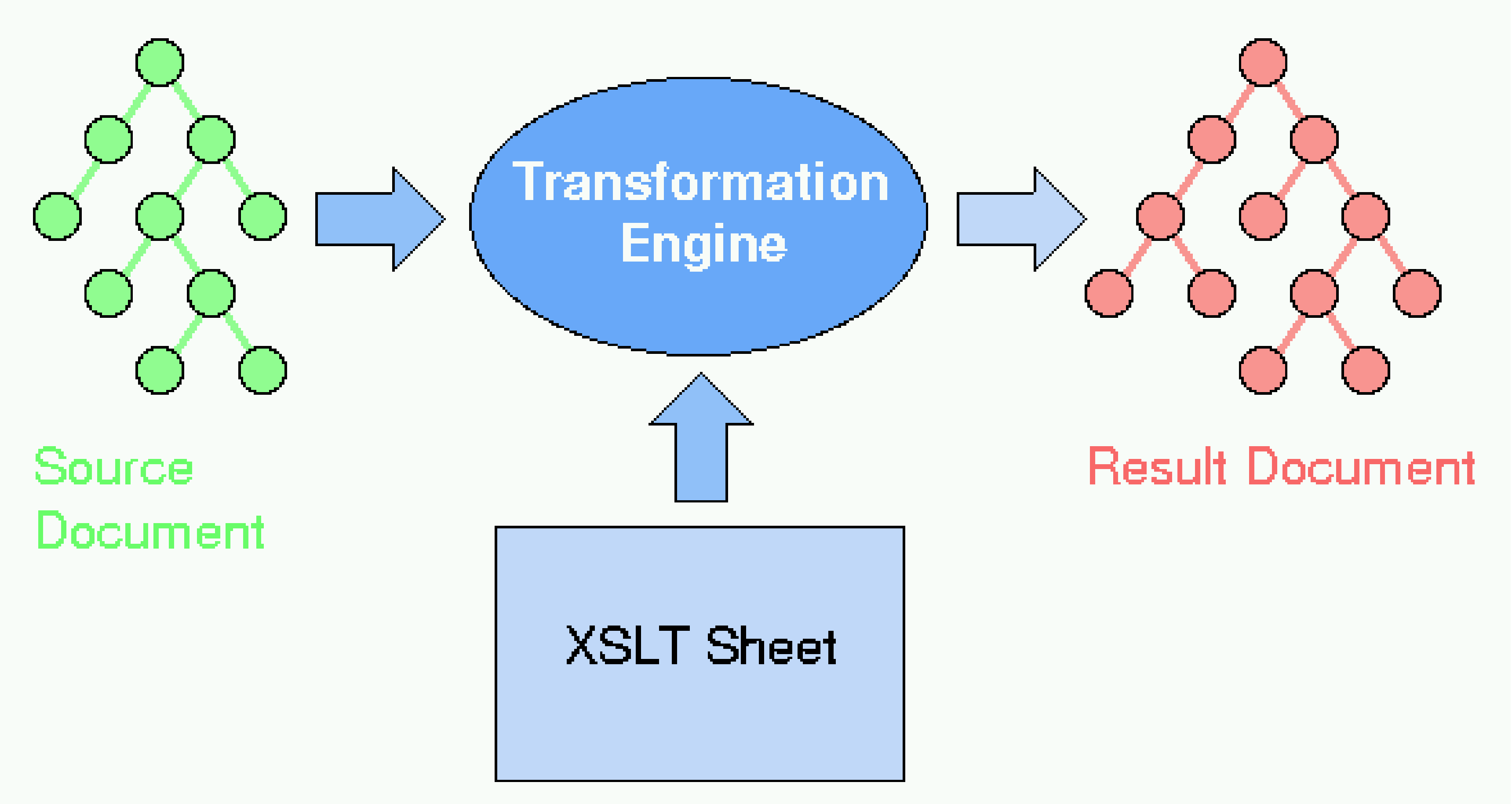
XSL Components

XSL is constituted of three main components:

- **XSLT**: a transformation language
- **XPath**: an expression language for addressing parts of XML documents
- **FO**: a vocabulary of *formatting objects* with their associated formatting properties

XSL uses XSLT which uses XPath

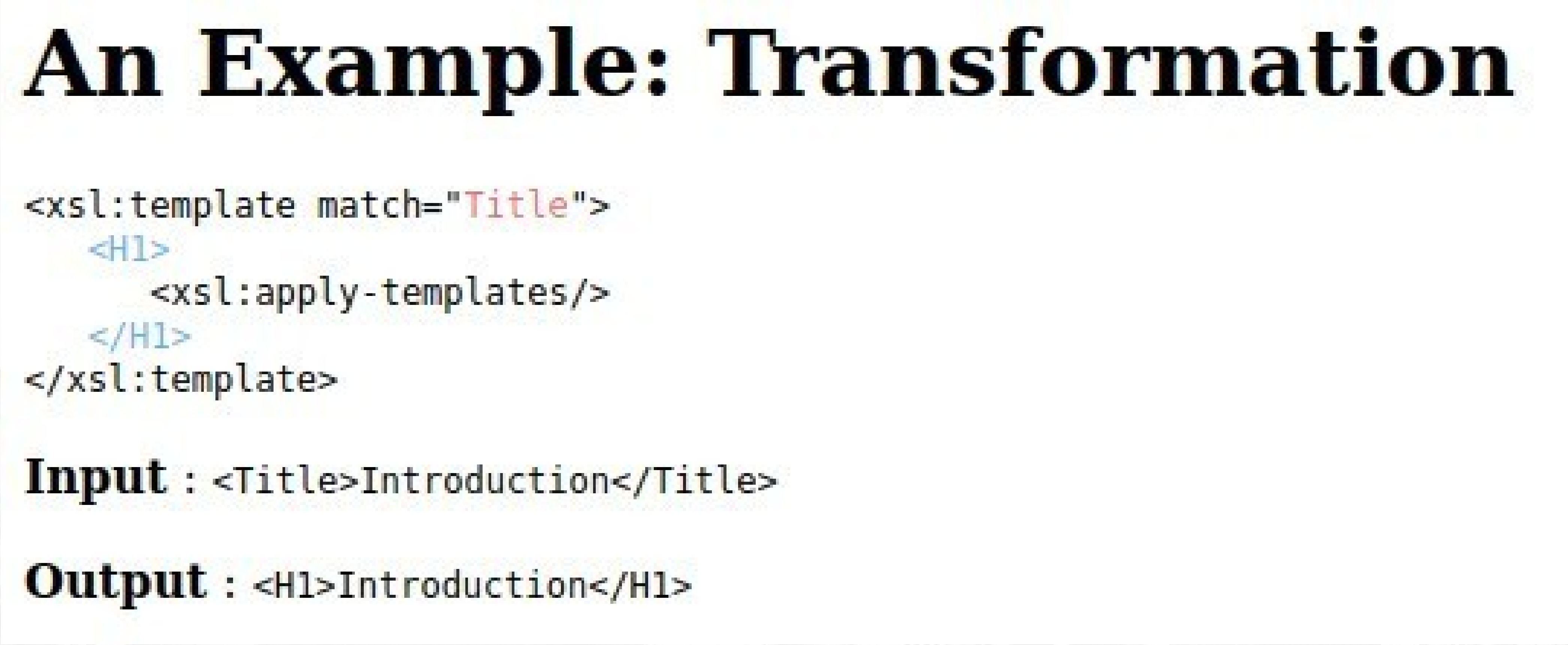
XSL Transformations



XSLT - Basic Principle

Patterns and **Templates**

- A style sheets describes transformation
- rules A transformation rule: a pattern + a
- template Pattern: a configuration in the
- source tree
- Template: a structure to be instantiated in the result tree When a pattern is matched in the source tree, the
  corresponding pattern is generated in the result tree

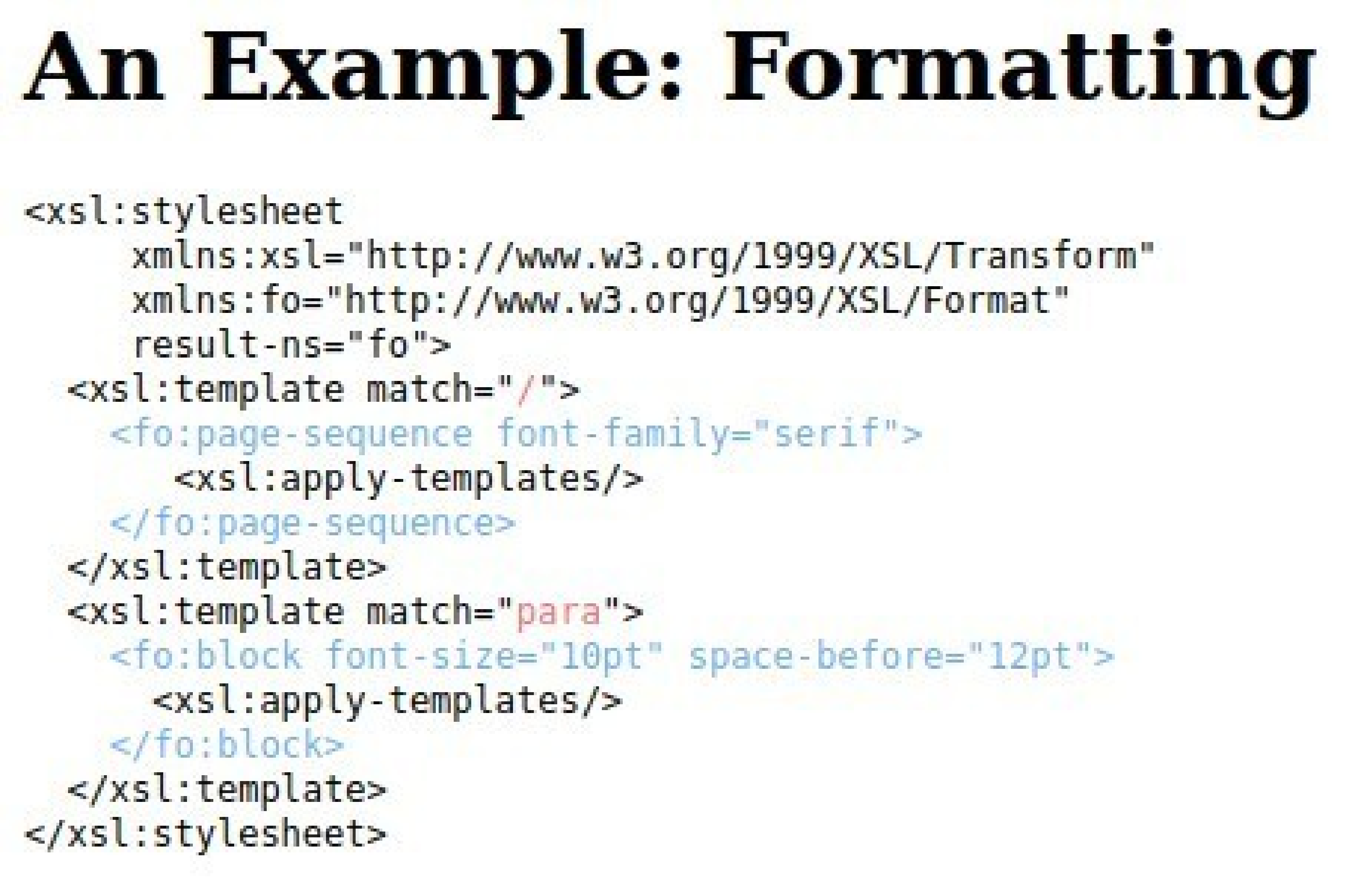


An Example: Transformation

```
<xsl:template match="Title">
   <H1>
      <xsl:apply-templates/>
   </H1>
</xsl:template>
```

Input : `<Title>Introduction</Title>`

Output : `<H1>Introduction</H1>`



An Example: Formatting

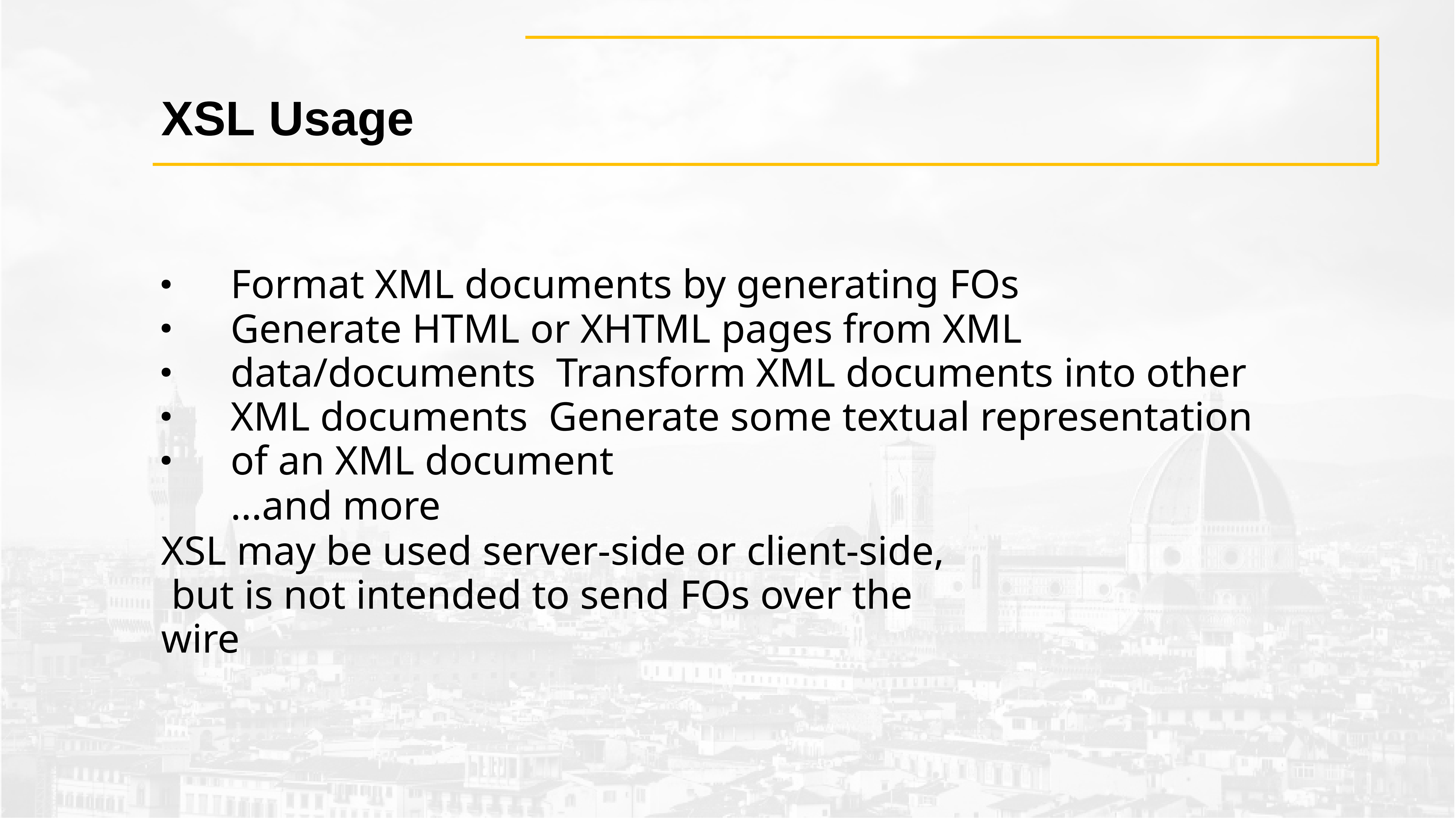
```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:fo="http://www.w3.org/1999/XSL/Format"
     result-ns="fo">
  <xsl:template match="/">
    <fo:page-sequence font-family="serif">
      <xsl:apply-templates/>
    </fo:page-sequence>
  </xsl:template>
  <xsl:template match="para">
    <fo:block font-size="10pt" space-before="12pt">
     <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

XSL Usage

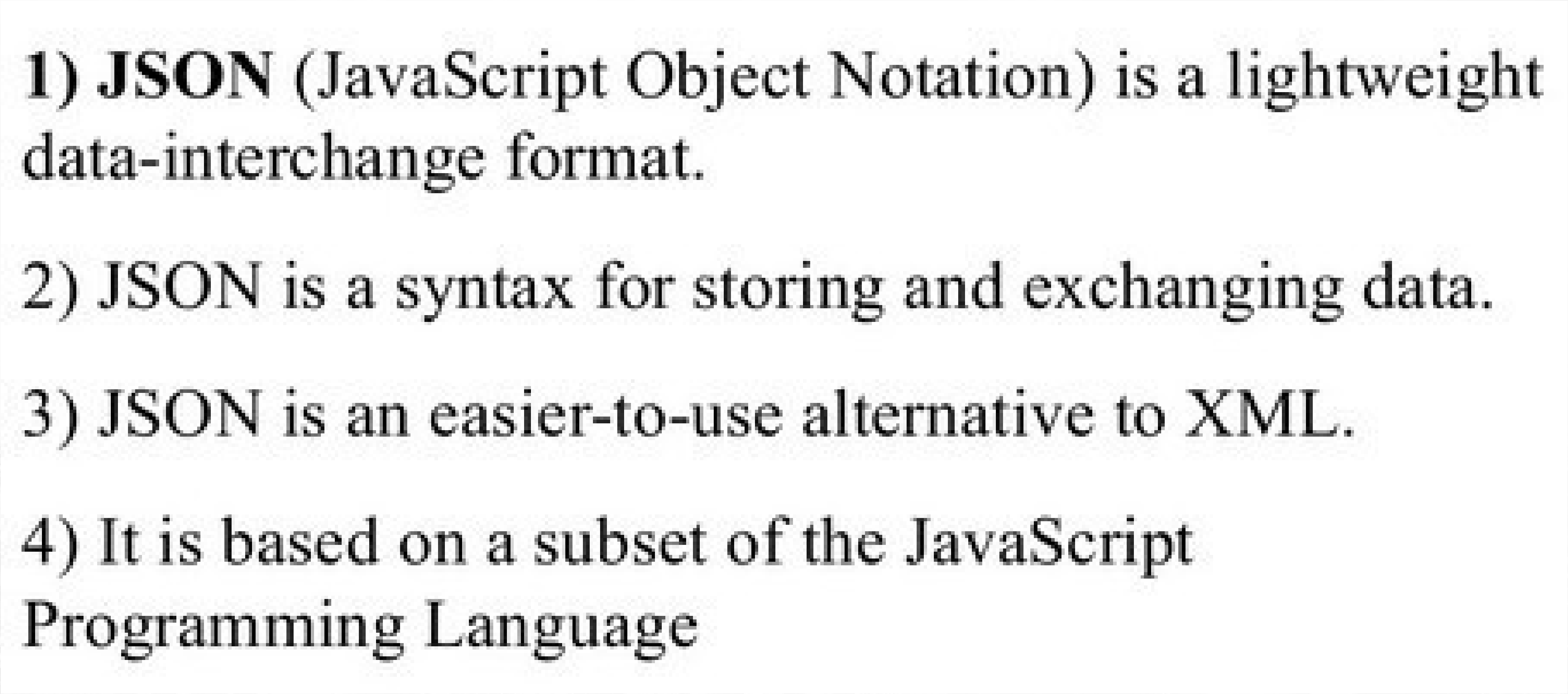
- Format XML documents by generating FOs
- Generate HTML or XHTML pages from XML
- data/documents Transform XML documents into other
- XML documents Generate some textual representation
- of an XML document

…and more

XSL may be used server-side or client-side,
but is not intended to send FOs over the
wire



JSON

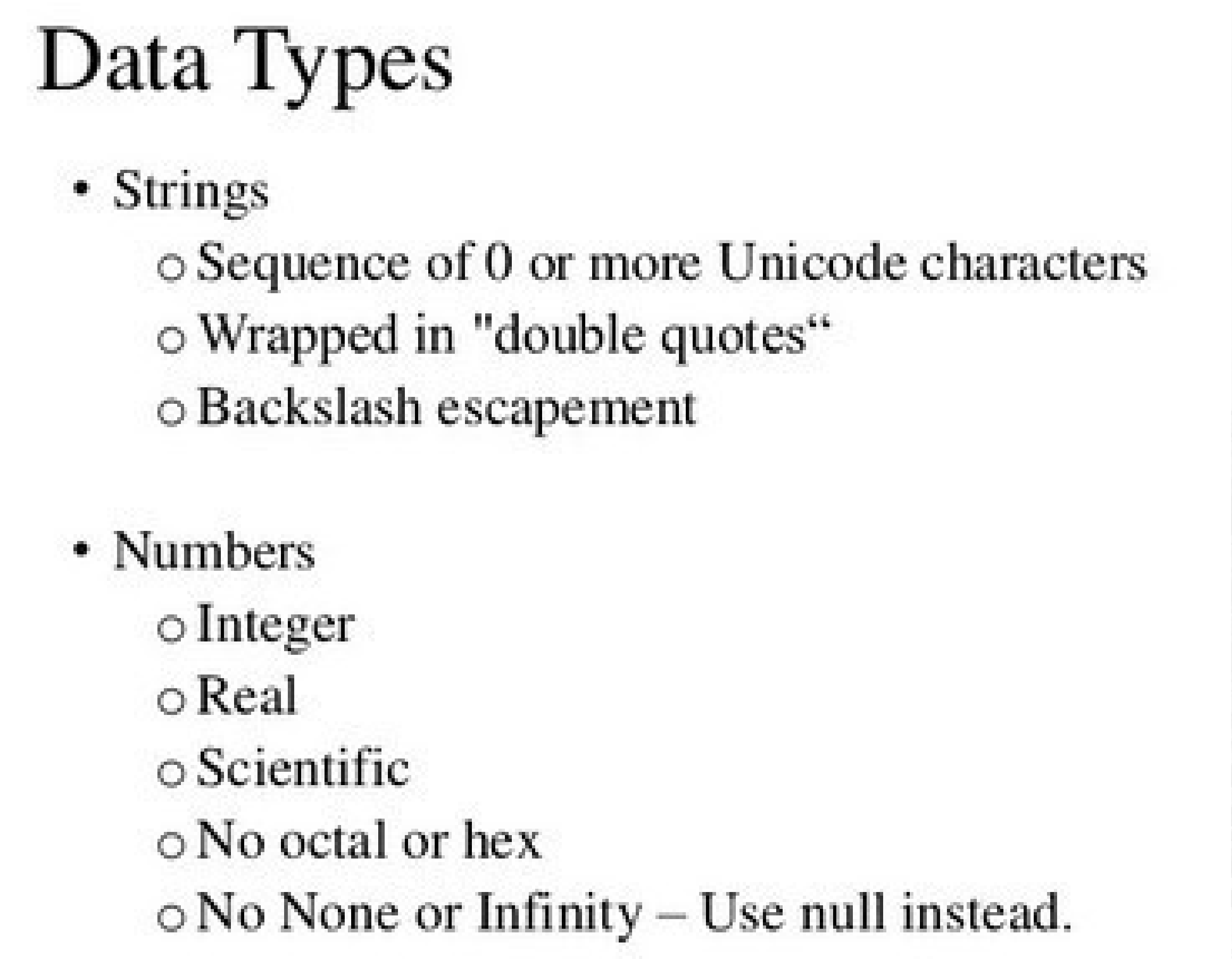


1) **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.

2) JSON is a syntax for storing and exchanging data.

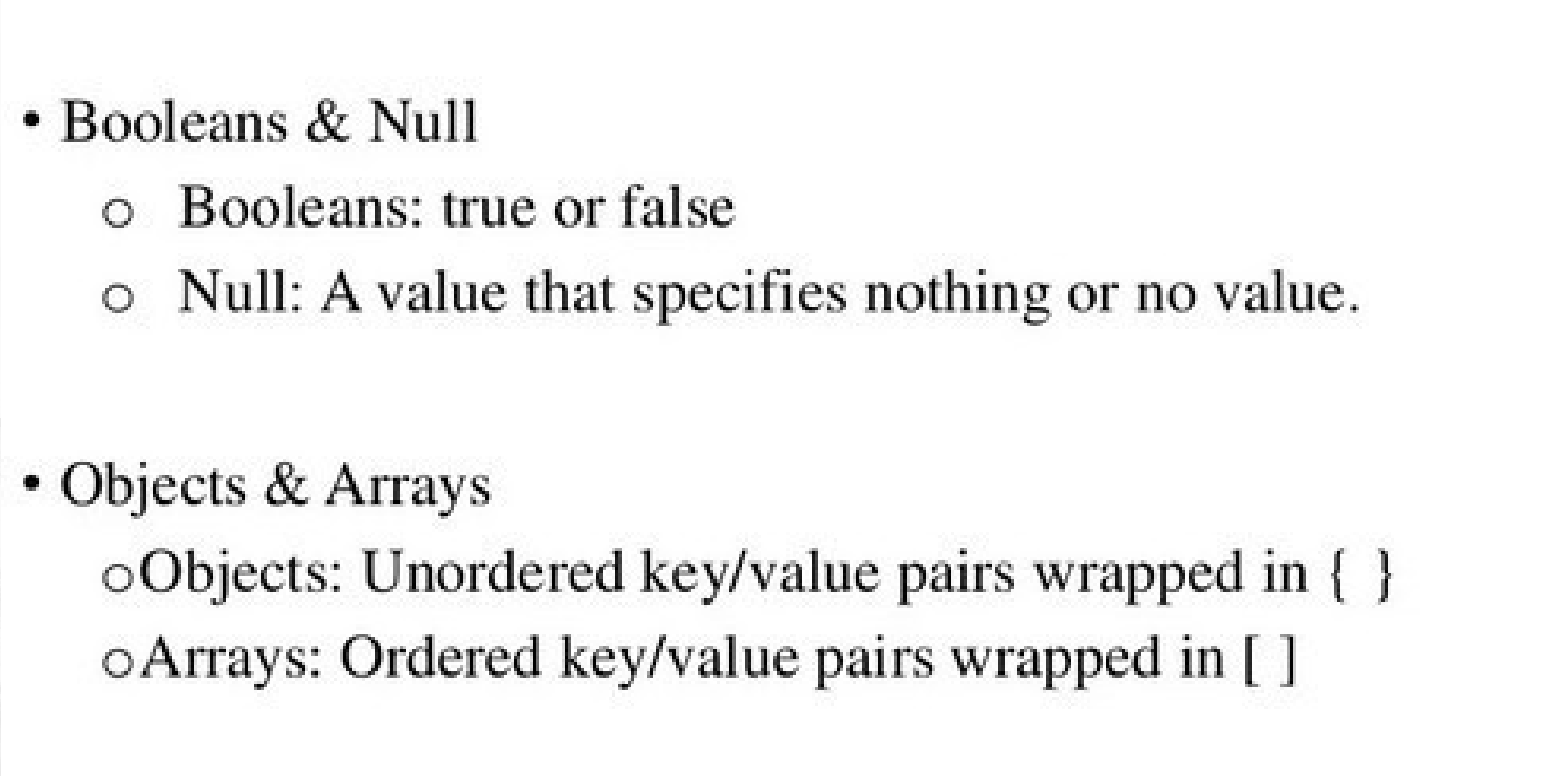
3) JSON is an easier-to-use alternative to XML.

4) It is based on a subset of the JavaScript Programming Language



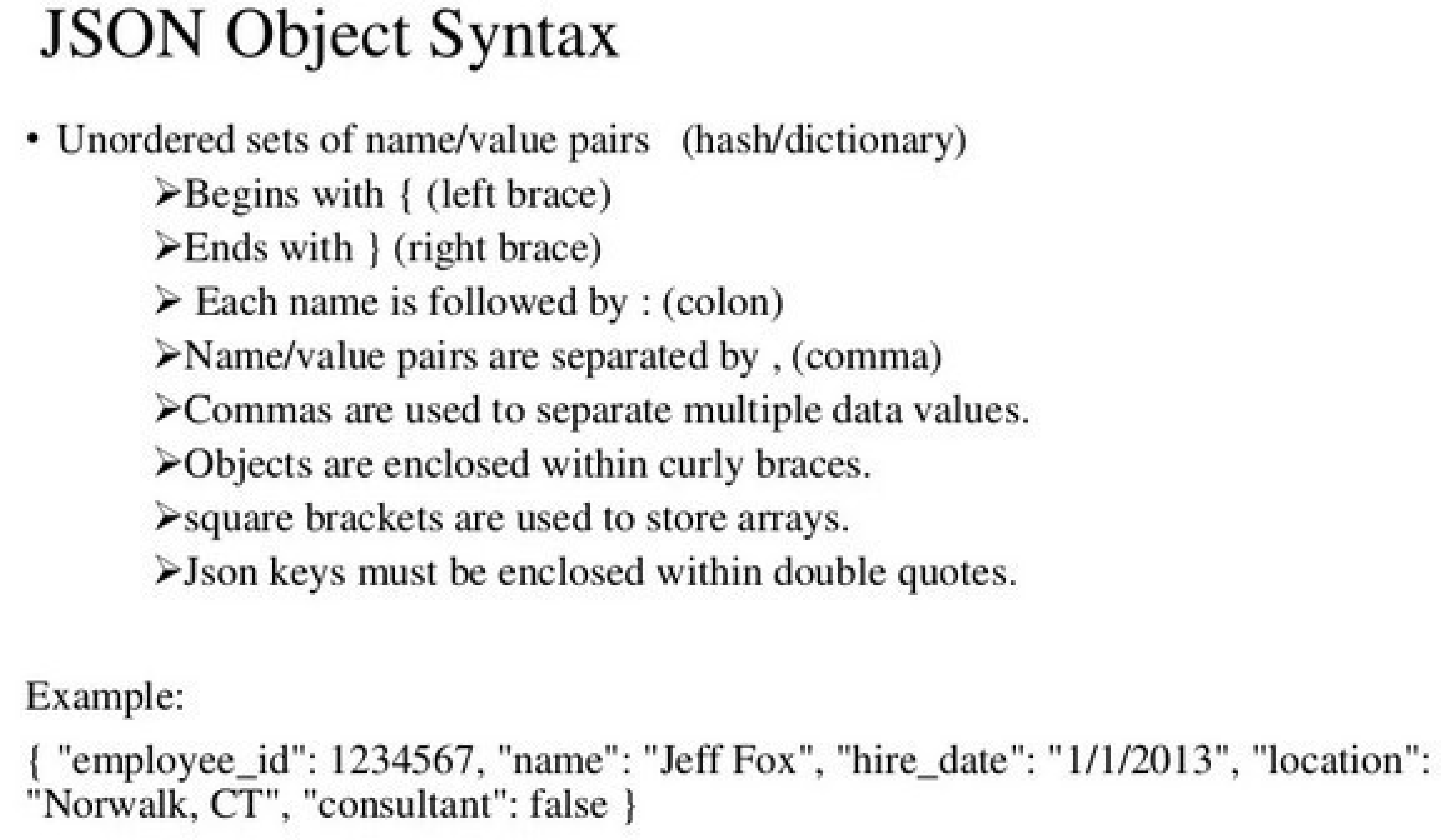
Data Types

- Strings
  - ○ Sequence of 0 or more Unicode characters
  - ○ Wrapped in "double quotes“
  - ○ Backslash escapement
- Numbers
  - ○ Integer
  - ○ Real
  - ○ Scientific
  - ○ No octal or hex
  - ○ No None or Infinity – Use null instead.



- Booleans & Null
  - Booleans: true or false
  - Null: A value that specifies nothing or no value.

- Objects & Arrays
  - Objects: Unordered key/value pairs wrapped in { }
  - Arrays: Ordered key/value pairs wrapped in [ ]

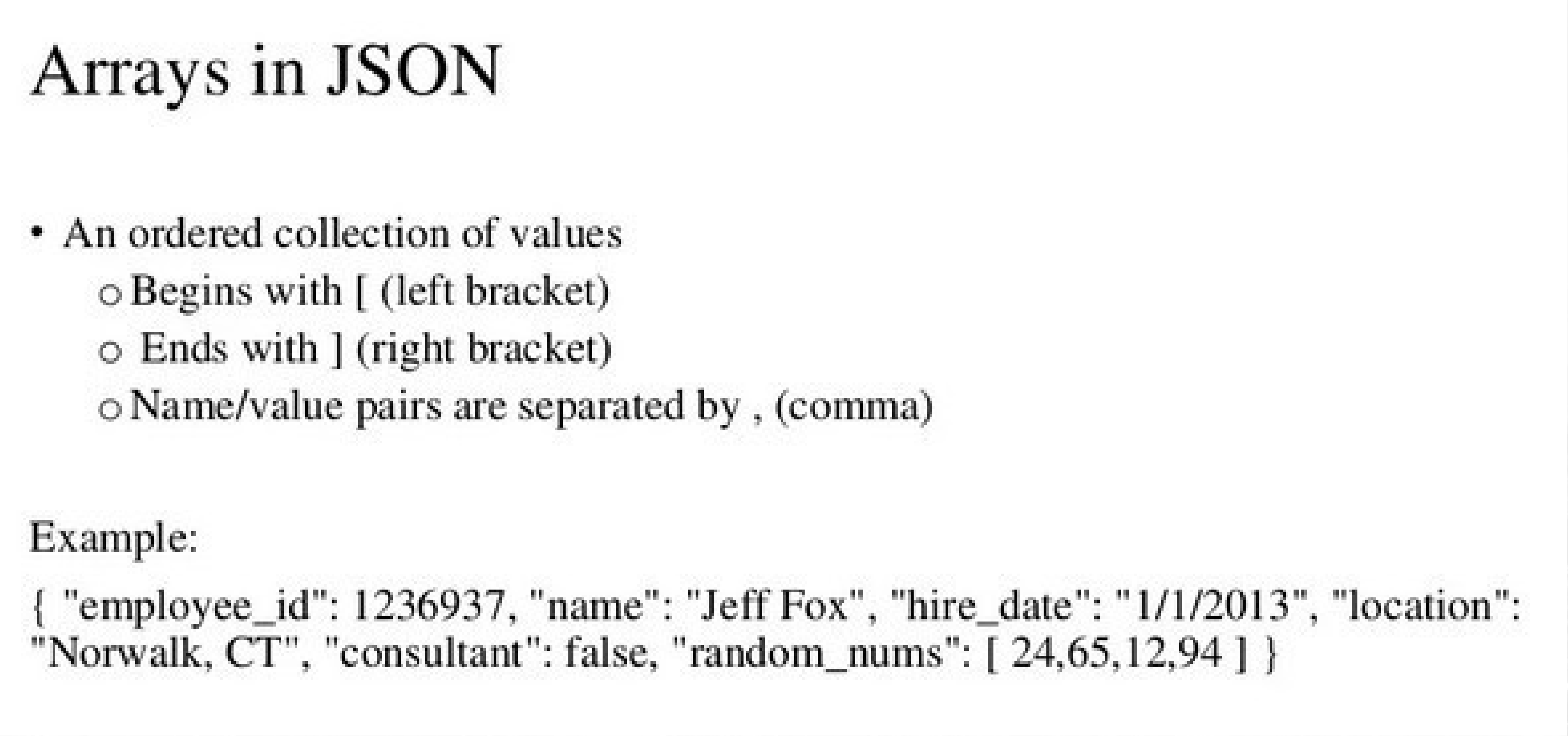


JSON Object Syntax

- Unordered sets of name/value pairs (hash/dictionary)
  - ➢Begins with { (left brace)
  - ➢Ends with } (right brace)
  - ➢ Each name is followed by : (colon)
  - ➢Name/value pairs are separated by , (comma)
  - ➢Commas are used to separate multiple data values.
  - ➢Objects are enclosed within curly braces.
  - ➢square brackets are used to store arrays.
  - ➢Json keys must be enclosed within double quotes.

Example:

```
{ "employee_id": 1234567, "name": "Jeff Fox", "hire_date": "1/1/2013", "location":
"Norwalk, CT", "consultant": false }
```



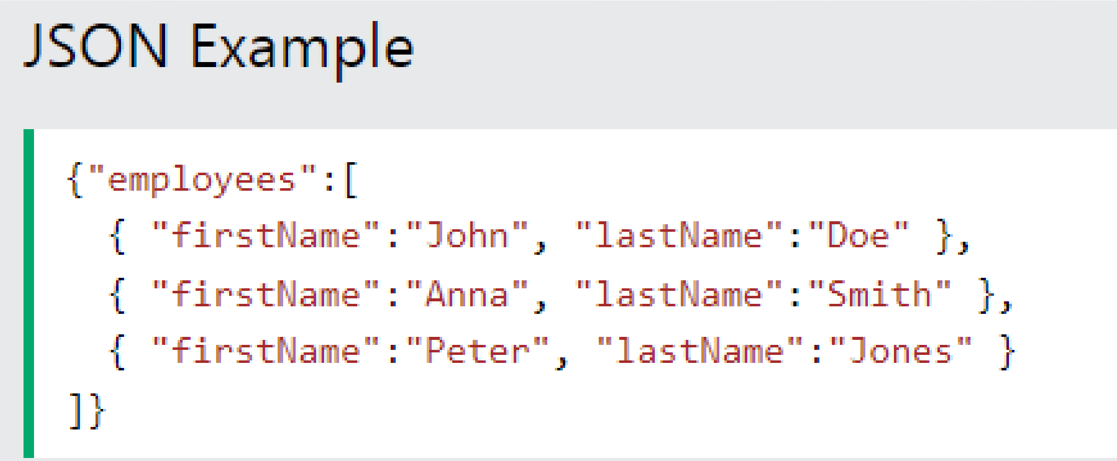
Arrays in JSON

- An ordered collection of values
  - Begins with [(left bracket)
  - Ends with] (right bracket)
  - Name/value pairs are separated by , (comma)

Example:

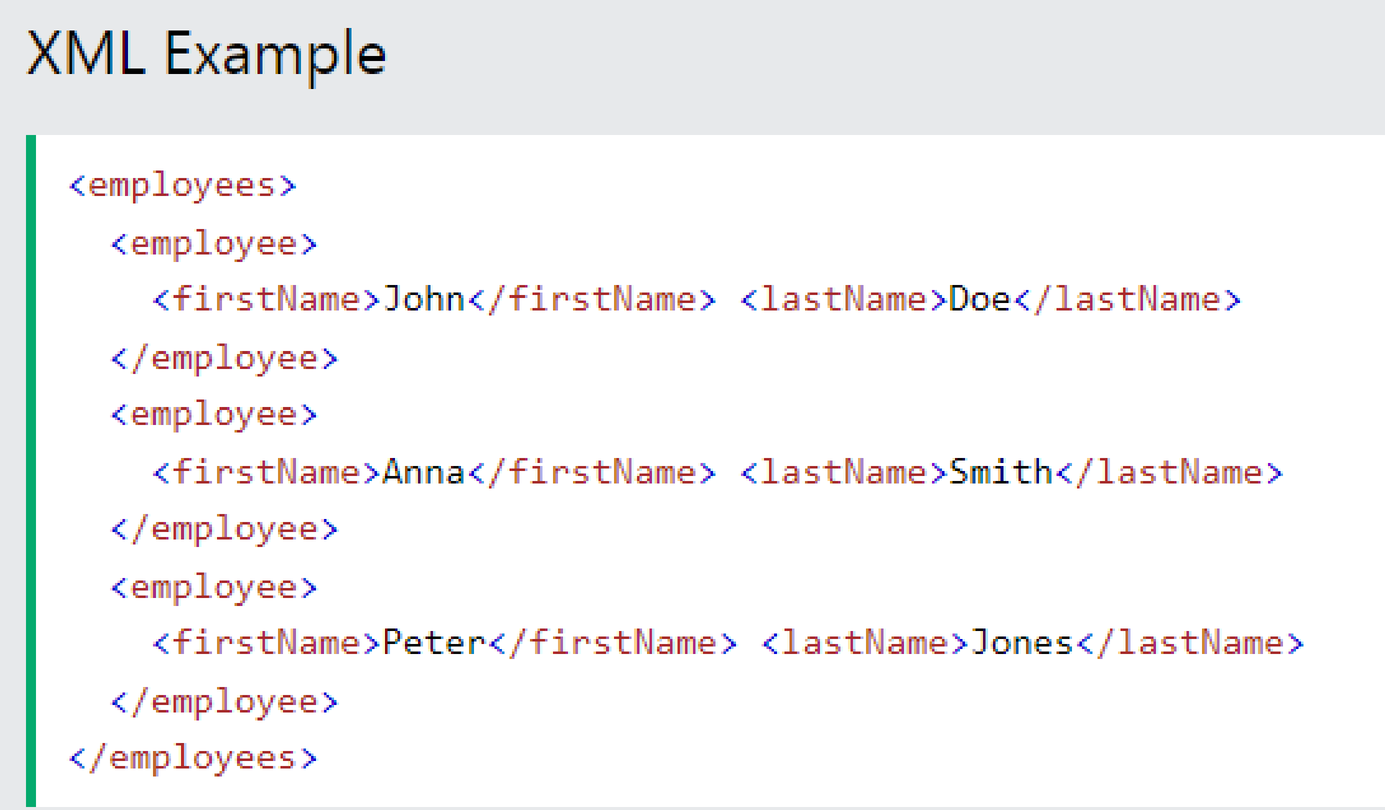
{ "employee_id": 1236937, "name": "Jeff Fox", "hire_date": "1/1/2013", "location": "Norwalk, CT", "consultant": false, "random_nums": [24,65,12,94] }

JSON Vs XML



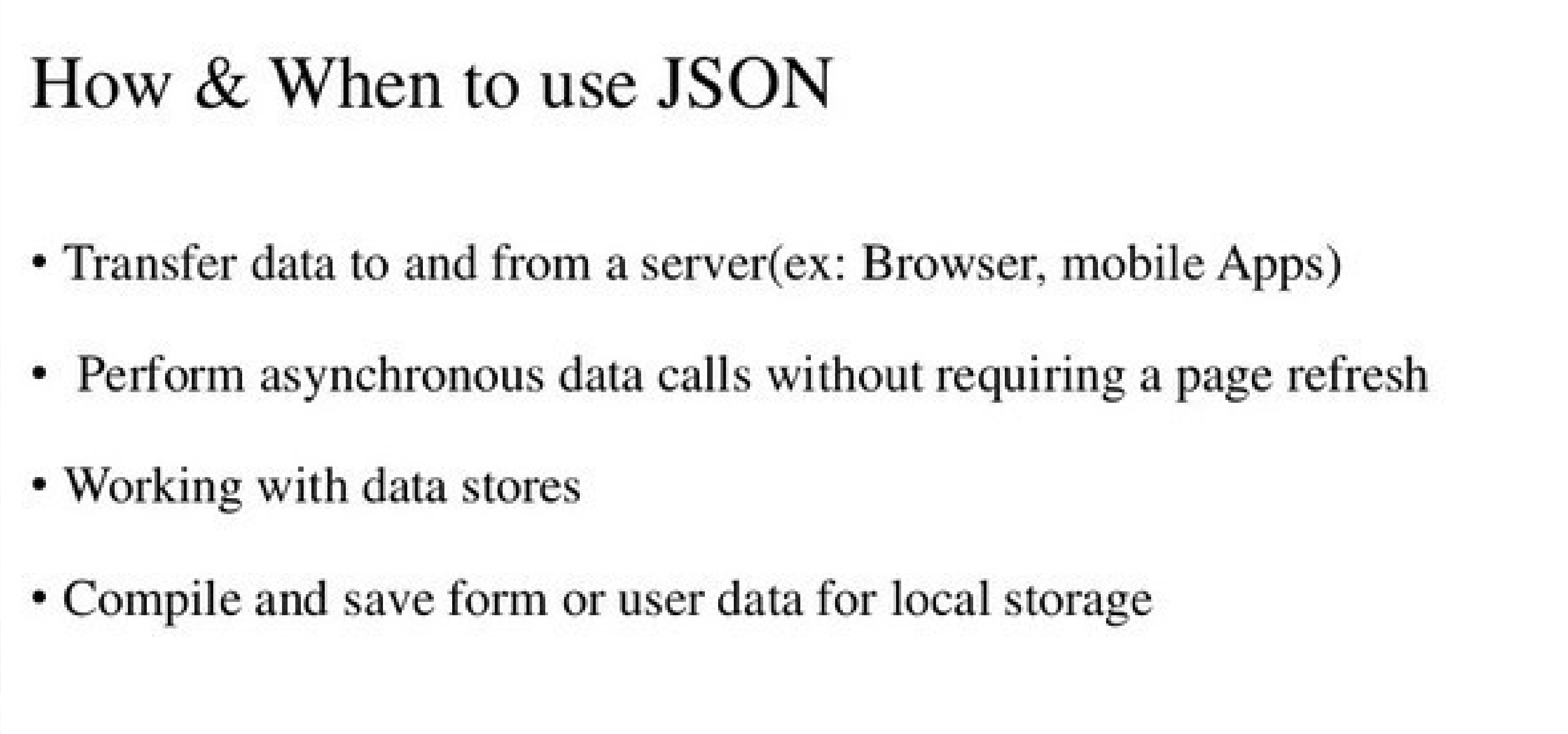
JSON Example

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```



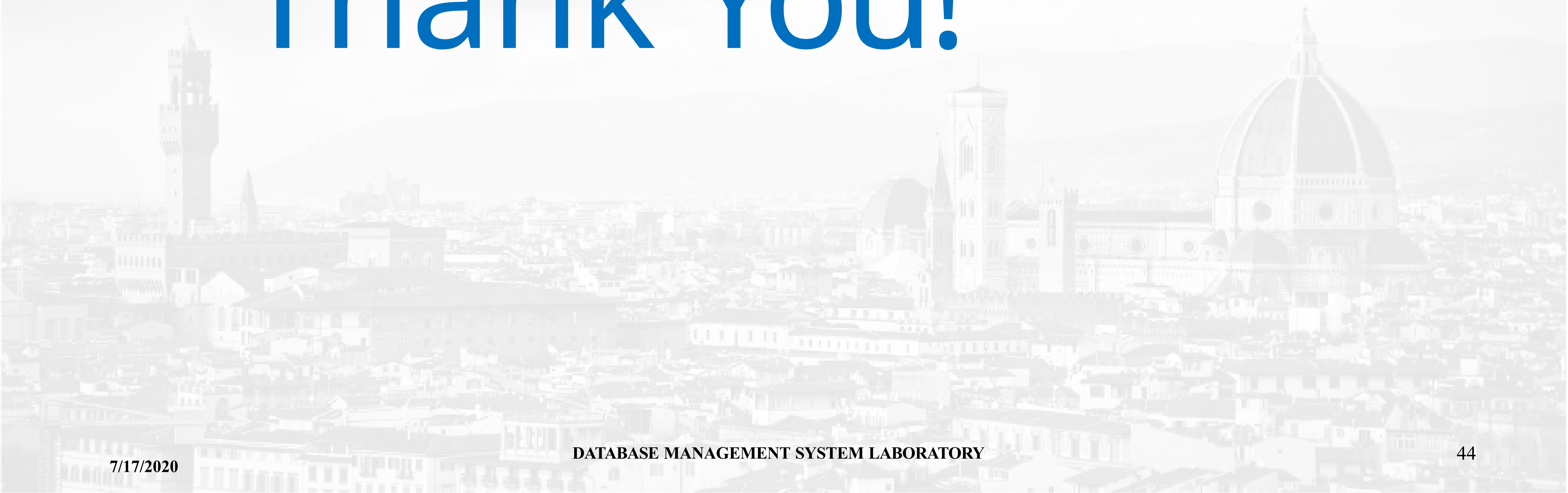
XML Example

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```



How & When to use JSON

- Transfer data to and from a server(ex: Browser, mobile Apps)
- Perform asynchronous data calls without requiring a page refresh
- Working with data stores
- Compile and save form or user data for local storage



Thank You!