

Introduction to XML



DTD

What is a DTD?

A DTD is usually a file (or several files to be used together) which contains a formal definition of a particular type of document. This sets out what names can be used for elements, where they may occur, and how they all fit together.

It's a formal language which lets processors automatically parse a document and identify where every element comes and how they relate to each other, so that stylesheets, navigators, browsers, search engines, databases, printing routines, and other applications can be used.

A DTD contain metadata relative to a collection of XML docs.

Valid documents

a *valid* XML document is one that conforms to an existing DTD in every respect.

For example...

Unless the DTD allows an element with the name "*color*", an XML document containing an element with that name is not valid according to that DTD (but it might be valid according to some other DTD).

An *invalid* XML document can be a perfectly good and useful XML document.

Valid documents

Validity is not a requirement of XML

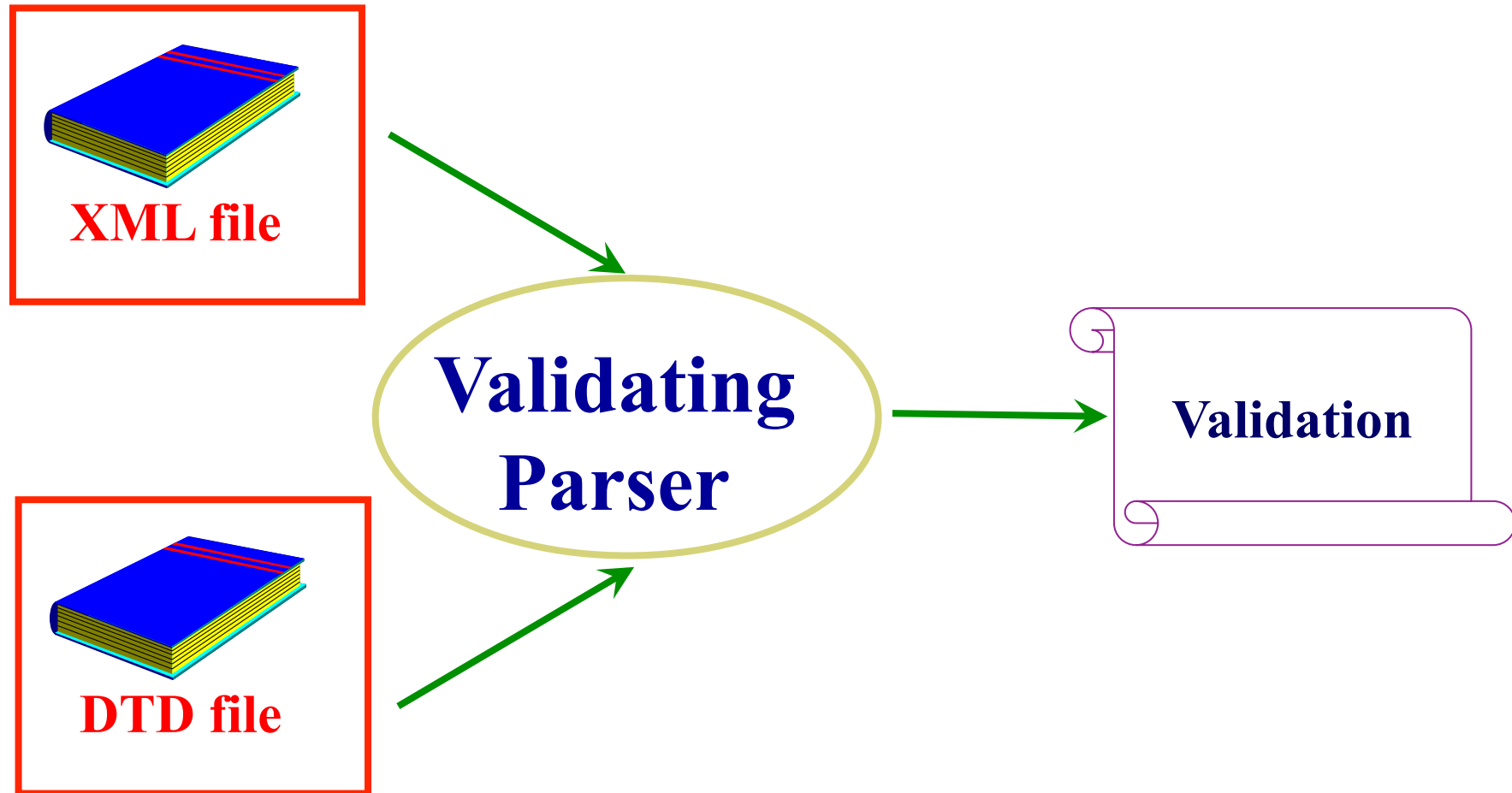
Because XML does not require a DTD, in general, an XML processor cannot require validation of the document.

Many very useful XML documents are not valid, simply because they were not constructed according to an existing DTD.

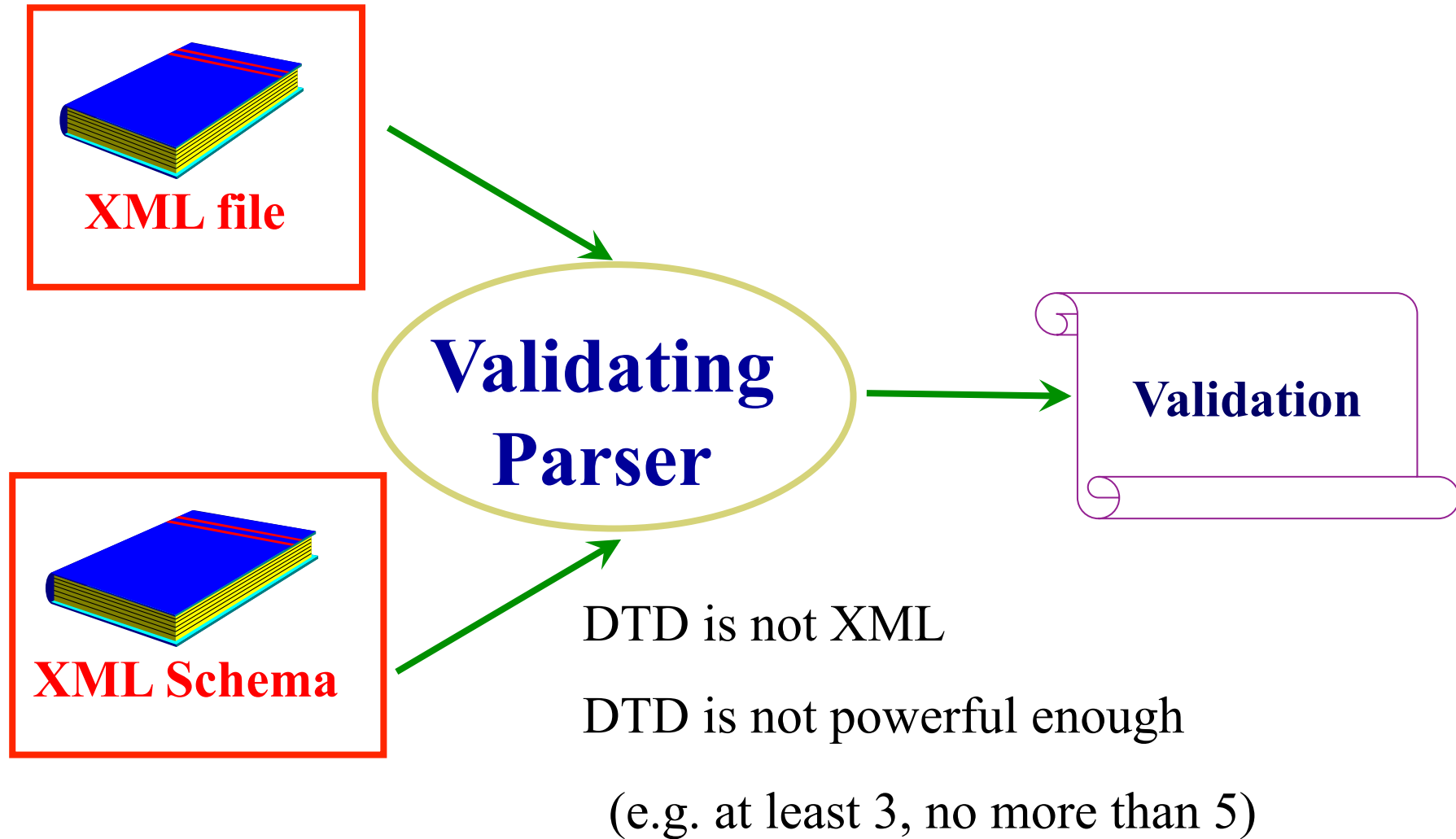
To make a long story short,

validation against a DTD can often be very useful, but is not required.

Constraining & Validating XML



Constraining & Validating XML



Where are the DTDs?

A DTD can be **external** or **internal** to a document.

`<!DOCTYPE Report>`

Internal DTD

`<!DOCTYPE Report SYSTEM "Report.dtd">`

`<!DOCTYPE Report PUBLIC "Report.dtd">`

External DTD

Broadly and publicly available

URL

DTD Markup: ELEMENT

<!ELEMENT name content-model>

<!ELEMENT book (preface?,chapter+,index)>

<!ELEMENT preface(paragraph+)>

<!ELEMENT paragraph (#PCDATA)>

<!ELEMENT chapter (title,paragraph+,reference*)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT reference (#PCDATA|URL)>

<!ELEMENT URL (#PCDATA)>

<!ELEMENT index(number,title,page_number)>

<!ELEMENT number(#PCDATA)>

<!ELEMENT page_number(#PCDATA)>

? Zero or one
+ One or more
* Zero or more
, sequence
| or (not xor!)

DTD Markup: ATTLIST

<!ATTLIST element-name attribute-name type default>

<!ELEMENT Product (#PCDATA)>

<!ATTLIST Product

Name CDATA #IMPLIED

Rev CDATA #FIXED "1.0"

Code CDATA #REQUIRED

Pid ID #REQUIRED

Series IDREF

Status (InProduction|Obsolete)

"InProduction"

>

TYPES:

CDATA character data

ID Unique key

IDREF Foreign Key

(...|...) Enumeration

DEFAULT:

#IMPLIED optional, no default

#FIXED optional, default supplied.

 If present must match default

#REQUIRED must be provided

DTD Markup: ENTITY

Entities are a sort of macro

General Entity

`<!ENTITY author “Marco Ronchetti, Universita’ di Trento”>`

External Parsed Entity

`<!ENTITY content SYSTEM “content.xml”>`

`<Tag>&content &author</Tag>`

External to the DTD

Parameter Entity

`<!ENTITY % AI “CDATA #IMPLIED”>`

`<!ATTLIST Product Name %AI>`

Internal at the DTD

The main problem of DTD's...

They are not written in XML!

Solution:

Another XML-based standard: XML Schema

For more info see:

<http://www.w3.org/XML/Schema>



XML Schema (W3C)



Thanks to Jussi Pohjolainen
TAMK University of Applied Sciences



w3schools.com

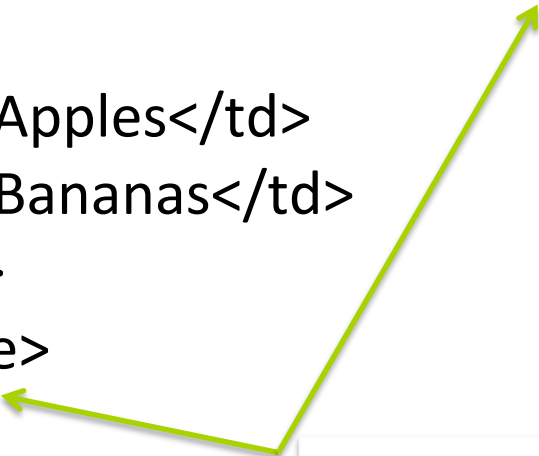
XML NAMESPACES

XML Namespaces

- ❑ The idea behind XML namespaces is to avoid **element name conflicts**.
- ❑ Example of name conflict (w3schools.com)

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```



Same tag-name, different content and meaning!

Solving Name Conflict

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

Prefix **h** has xhtml-related elements and prefix **f** has furniture-related elements

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

xmlns - attributes

- ❑ When using prefixes in XML, a so-called namespace for the prefix must be defined.
- ❑ The namespace is defined by the `xmlns` attribute in the **start tag of an element**.

xmlns - attribute

```
<root>
  <h:table xmlns:h="http://www.w3.org/TR/html4/">
    <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
    </h:tr>
  </h:table>

  <f:table xmlns:f="http://www.w3schools.com/furniture">
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
  </f:table>
</root>
```

xmlns - attribute

```
<root  
  xmlns:h="http://www.w3.org/TR/html4/"  
  xmlns:f="http://www.w3schools.com/furniture">
```

```
  <h:table>  
    <h:tr>  
      <h:td>Apples</h:td>  
      <h:td>Bananas</h:td>  
    </h:tr>  
  </h:table>
```

```
  <f:table>  
    <f:name>African Coffee Table</f:name>  
    <f:width>80</f:width>  
    <f:length>120</f:length>  
  </f:table>
```

```
</root>
```

Namespace name

- ❑ The name of namespace should be unique: `<h:table xmlns:h="http://www.w3.org/TR/html4/">`
- ❑ It is just a string, but it should be declared as URI.
- ❑ Using URI *reduces* the possibility of different namespaces using **duplicate identifiers**.

Example:

An XHTML + MathML + SVG Profile

- ❑ An XHTML+MathML+SVG profile is a profile that combines XHTML 1.1, MathML 2.0 and SVG 1.1 together.
- ❑ This profile enables mixing XHTML, MathML and SVG in the same document using **XML namespaces mechanism**.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
  "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg-flat.dtd">
<html xmlns
  = "http://www.w3.org/1999/xhtml"
  xmlns:svg
  = "http://www.w3.org/2000/svg">
```

```
<head>
  <title>Example of XHTML, SVG and MathML</title>
</head>
<body>
```

```
<h2>MathML</h2>
```

```
<p>
```

```
  <math xmlns="http://www.w3.org/1998/Math/MathML">
```

```
    <mfrac>
```

```
      <mi>a</mi>
```

```
      <mi>b</mi>
```

```
    </mfrac>
```

```
  </math>
```

```
</p>
```

```
<h2>SVG</h2>
```

```
<p>
```

```
  <svg:svg width="50px" height="50px">
```

```
    <svg:circle cx="25px" cy="25px" r="20px" fill="green"/>
```

```
  </svg:svg>
```

```
</p>
```

```
</body>
```

```
</html>
```



MathML

$$\frac{a}{b}$$

SVG





W3C SCHEMA

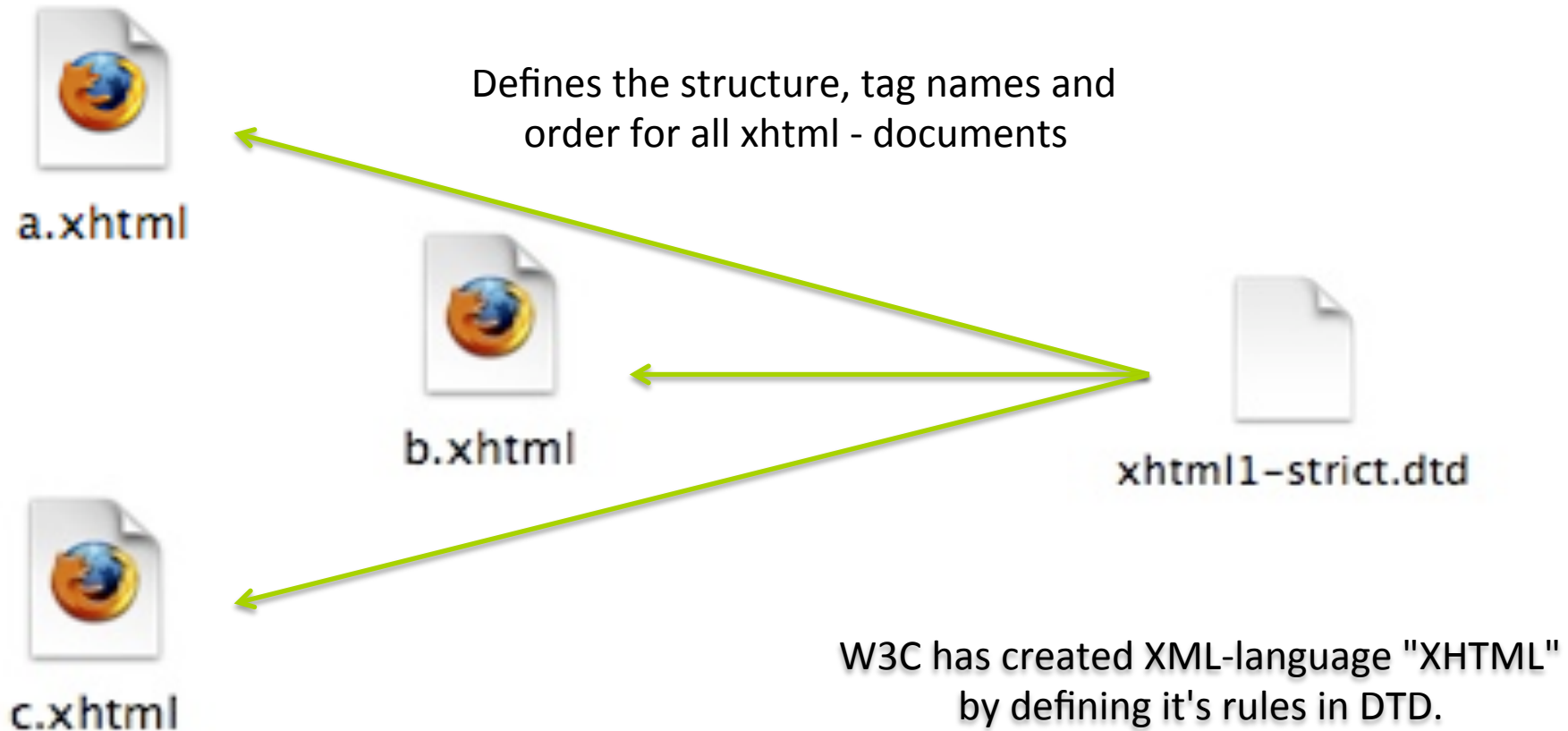
XML Schema (W3C)

- ❑ Language for defining set of rules for XML – documents.
- ❑ W3C Recommendation (2001)
- ❑ More specific than DTD
 - Datatypes!
- ❑ Is XML-language and it uses *xml namespaces*

Schema vs. DTD (W3Schools.com)

- ❑ XML Schemas are extensible to future additions
- ❑ XML Schemas are richer and more powerful than DTDs
- ❑ XML Schemas are written in XML
- ❑ XML Schemas support data types
- ❑ XML Schemas support namespaces

DTD Linking



DTD Linking



a.xml



b.xml



c.xml

Defines the structure, tag names and order for all "book"- documents



books.dtd

TAMK has created XML-language "Book" by defining it's rules in DTD.

Schema Linking



a.xml



b.xml



c.xml

Defines the structure, tag names and order for all "book"- documents



books.xsd

TAMK has created XML-language "Book" by defining it's rules in a Schema.

Linking?

- The basic idea with linking to Schema:

```
<?xml version="1.0"?>  
<root schemaLocation="note.xsd">  
    <foo>...</foo>  
</root>
```

- The problem with this is that now it is set that attribute "schemaLocation" is part of your XML-language

Linking and Namespace Usage

□ Linking with namespace

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

- Now the "schemaLocation" – attribute is in it's own namespaces (xsi) and does not belong to the "main" language.

Simple Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="complete_name" type="complete_name_type"/>

  <xsd:complexType name="type="complete_name_type">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="cognome" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Let's remove namespaces...

```
<?xml version="1.0"?>
<schema>
  <element name="complete_name" type="complete_name_type"/>

  <complexType name="complete_name_type">
    <sequence>
      <element name="nome" type="string"/>
      <element name="cognome" type="string"/>
    </sequence>
  </complexType>
</schema>
```

It doesn't look so confusing
after all?

The Basics: Element

- You define the name for the elements by using element-element. 😊
 - `<element name="foo" type="bar" />`
- Type?
 - 44 Built-in schema datatypes
 - string, double, time, date, etc.
 - [See all the datatypes](#)

Usage of Datatypes

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

```
<xsd:element name="ableToSwim"  
              type="xsd:boolean" />
```

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

minOccurs and maxOccurs

□ The amount of elements

- In DTD: *, ?, +
- In Schema: minOccurs, maxOccurs

- Example

```
<xsd:element name="date" type="xsd:date"
minOccurs="1" maxOccurs="2" />
```

□ Default and special values

- default minOccurs: 1
- default maxOccurs: same as minOccurs
- maxOccurs="unbounded" : unlimited

Defining new Datatypes

- ❑ If the the built-in datatypes are not enough, you can build your own datatypes.
- ❑ This does not necessarily work:
 - `<xsd:element name="grade" type="xsd:integer" />`
- ❑ There are **two ways** of specifying your own datatype
 - Named Data Type
 - Anonymous Data Type

1) Named Data Type

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

  <xsd:element name="grade" type="grade_type" />

  <xsd:simpleType name="grade_type">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:minInclusive value="4" />
      <xsd:maxInclusive value="10" />
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

2) Anonymous Data Type

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

  <xsd:element name="grade">
    <xsd:simpleType>
      <xsd:restriction base="xsd:positiveInteger">
        <xsd:minInclusive value="4"/>
        <xsd:maxInclusive value="10"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>

</xsd:schema>
```

Benefits of Named Data Type

□ If you want re-use your datatype:

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

  <xsd:element name="grade" type="grade_type" />
  <xsd:element name="teachers_IQ" type="grade_type" />

  <xsd:simpleType name="grade_type">
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:minInclusive value="4"/>
      <xsd:maxInclusive value="10"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

SimpleType: enumeration

□ Alternative content

```
<xsd:simpleType name="car">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="Audi"/>
    <xsd:enumeration value="Golf"/>
    <xsd:enumeration value="BMW"/>
  </xsd:restriction>
</xsd:simpleType>
```

SimpleType: pattern

▣ Using REGEX:

```
<xsd:simpleType name="lowercase_char">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[a-z]"/>
  </xsd:restriction>
</xsd:simpleType>
```

REGEX Examples

```
<xs:pattern value="[A-Z][A-Z][A-Z]" />
```

```
<xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]" />
```

```
<xs:pattern value="[xyz]" />
```

```
<xs:pattern value="[0-9][0-9][0-9][0-9][0-9]" />
```

```
<xs:pattern value="([a-z])*" />
```

```
<xs:pattern value="male|female" />
```

```
<xs:pattern value="[a-zA-Z0-9]{8}" />
```

Structure of the XML-file

- ❑ It's possible to define the structure of the XML-file using `complexType`
- ❑ If element A has child-elements, then element A's type is `complexType`

SimpleType vs. ComplexType

□ SimpleType

- `<grade>7</grade>`
- Since `grade` **does not** hold other child – elements, `grade`'s type is **simpleType**

□ ComplexType

- `<students><student>Jack</student></students>`
- Since `student` **does hold** child – element(s), `student`'s type is **complexType**

Example: XML - File

```
<?xml version="1.0"?>
<students>
  <firstname>Fernando</firstname>
  <lastname>Alonso</lastname>
</students>
```

Example: XSD – file

Named ComplexType

Use now
complexType
(vs.
simpleType)

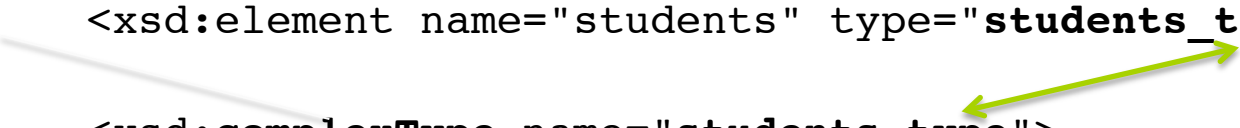
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="students" type="students_type">

    <xsd:complexType name="students_type">
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>

  </xsd:element>

</xsd:schema>
```



A diagram consisting of two arrows. A grey arrow points from the text box on the left to the `<xsd:complexType name="students_type">` line in the code. A yellow arrow points from the `students_type` attribute in the `<xsd:element name="students" type="students_type">` line to the `students_type` attribute in the `<xsd:complexType name="students_type">` line.

Example: XSD – file

Anonymous ComplexType

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

  <xsd:element name="students">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xsd:element name="lastname" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Example: ComplexType

```
<xsd:element name="employee" type="personinfo"/>
<xsd:element name="student" type="personinfo"/>
<xsd:element name="member" type="personinfo"/>

<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:element name="firstname" type="xsd:string"/>
    <xsd:element name="lastname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Deep Structure in XML - File

```
<?xml version="1.0"?>
<students>
  <student>
    <name>
      <firstname>Fernando</firstname>
    </name>
  </student>
</students>
```

Using Anonymous Data Type: The Horror!

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

  <xsd:element name="students">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="firstname" type="xsd:string"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

"There is an error in my schema, could you find it for me?"

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

  <xsd:element name="students">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="student">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="firstname" type="xsd:string"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:complexType>
</xsd:element>

</xsd:schema>
```

Use Named Datatypes! It's easier to find errors..

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

  <xsd:element name="students" type="students_type" />

  <xsd:complexType name="students_type">
    <xsd:sequence>
      <xsd:element name="student" type="student_type" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="student_type">
    <xsd:sequence>
      <xsd:element name="name" type="name_type" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="name_type">
    <xsd:sequence>
      <xsd:element name="firstname" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Order of the elements

- ❑ **Sequence:** Elements appear in same order than in Schema
- ❑ **All:** Elements can appear in any order
- ❑ **Choice:** One element can appear from the choice-list

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="employee" type="employee"/>
      <xsd:element name="member" type="member"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Attribute

□ XML

- `<student id="A1">...</student>`

□ Schema

```
<xsd:element name="student"
  type="student_type" />
```

```
<xsd:complexType name="student_type">
```

```
  <xsd:sequence>
```

```
    ...
```

```
  </xsd:sequence>
```

```
    <xsd:attribute name="id" type="xsd:ID"/>
```

```
</xsd:complexType>
```

Empty Element with Attribute

□ XML

- `<student id="A1" />`

□ Schema

```
<xsd:element name="student" type="student_type" />
```

```
<xsd:complexType name="student_type">
```

```
  <xsd:attribute name="id" type="xsd:ID" />
```

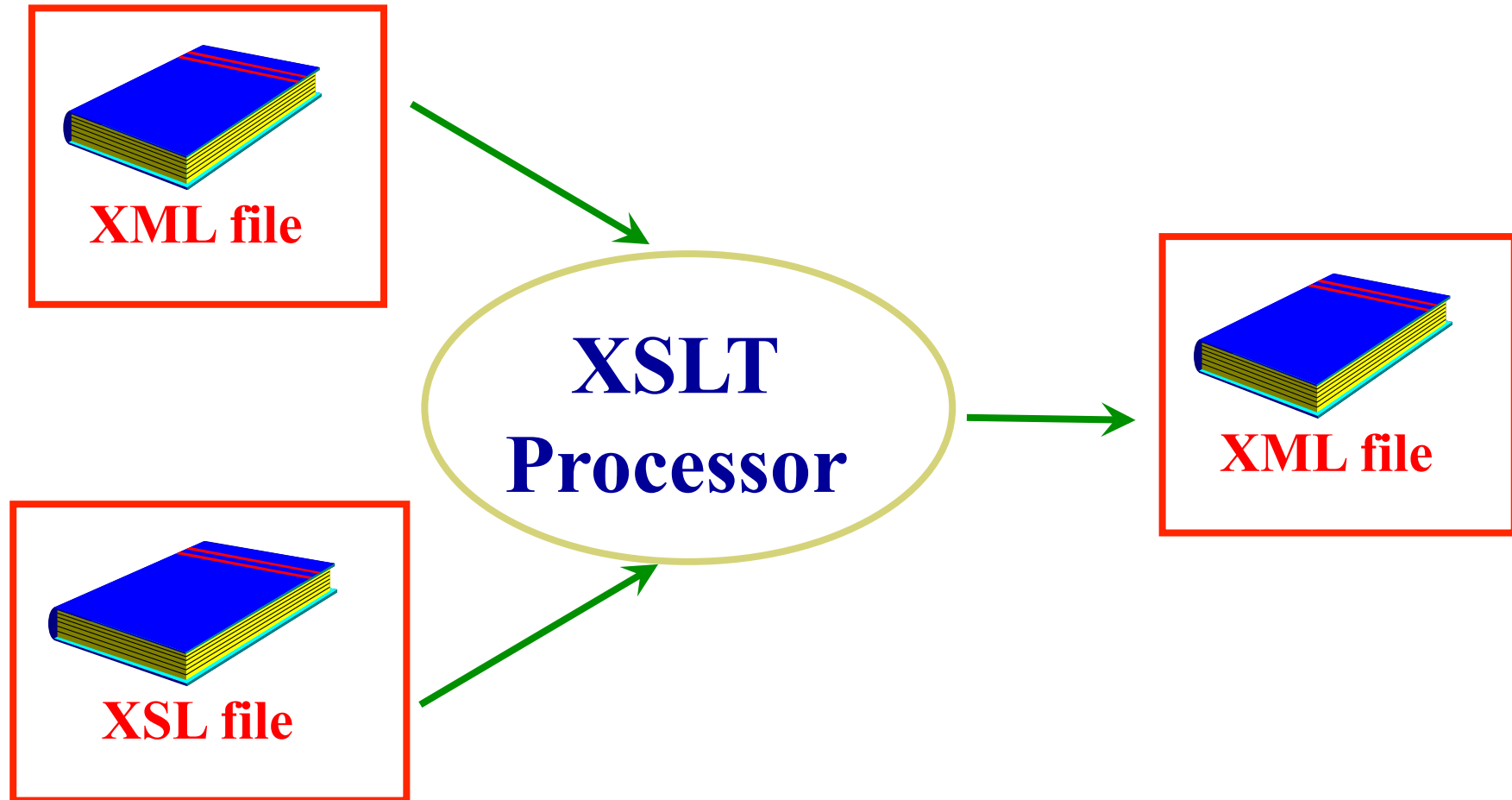
```
</xsd:complexType>
```

Introduction to XML



XSL - INTRODUCTION

Transforming XML



XSL is complex

**XSL is complex (much more complex than XML).
Designing an XSL stylesheet, to be used by a
rendering engine to properly render an XML document,
can be a daunting task.**

**Microsoft has developed an XSL debugger, and has
made it freely available for downloading.**

Some availableXSLT processors

Apache Xalan

<http://xml.apache.org>

James Clark' s XT

<http://www.jclark.com/xml/xt.html>

Lotus XSL Processor

<http://alphaworks.ibm.com/tech/LotusXSL>

Oracle XSL Processor

<http://technet.oracle.com/tech/xml>

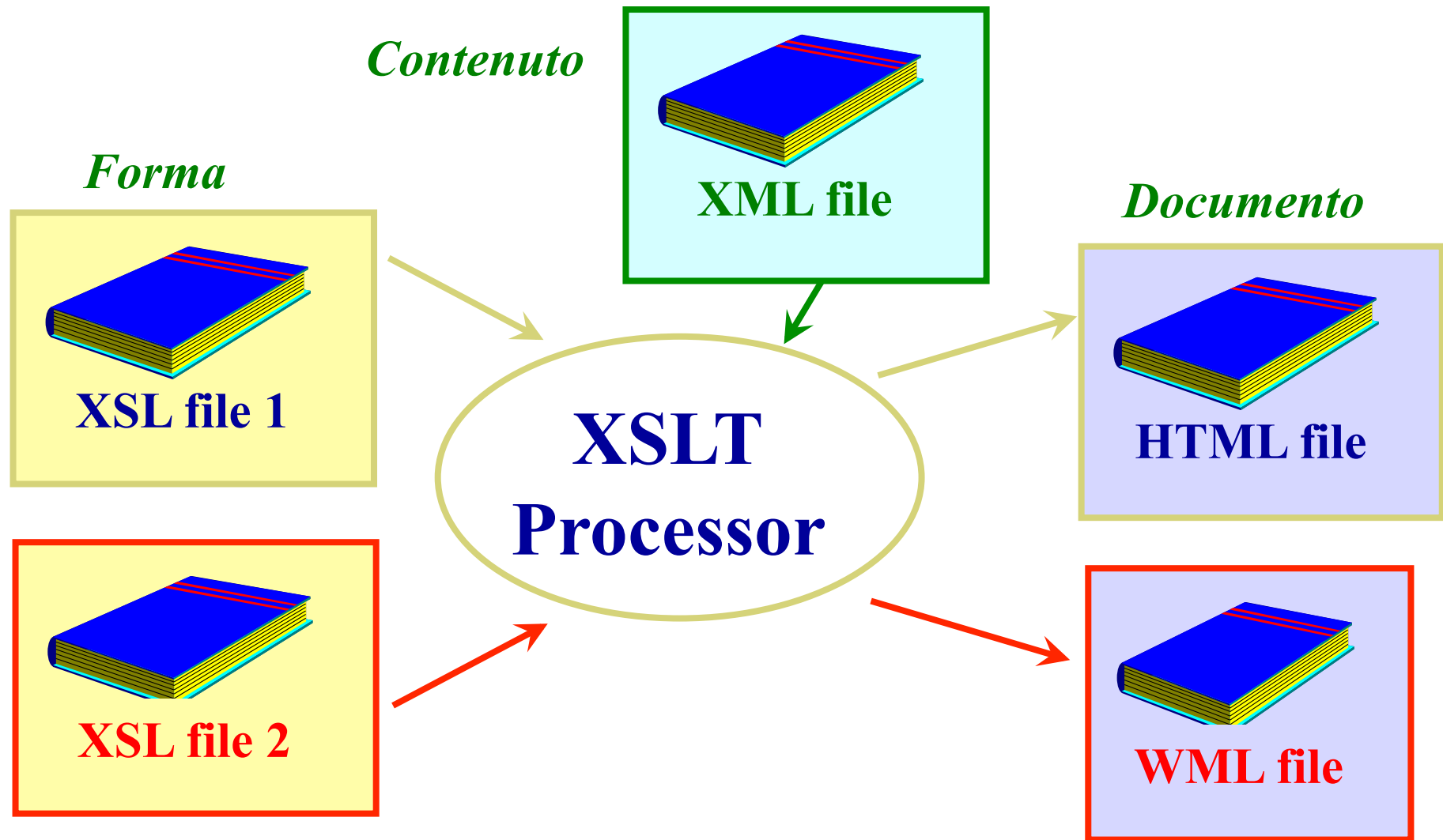
Keith Visco' s XSL:P

<http://www.clc-marketing.com/xslp>

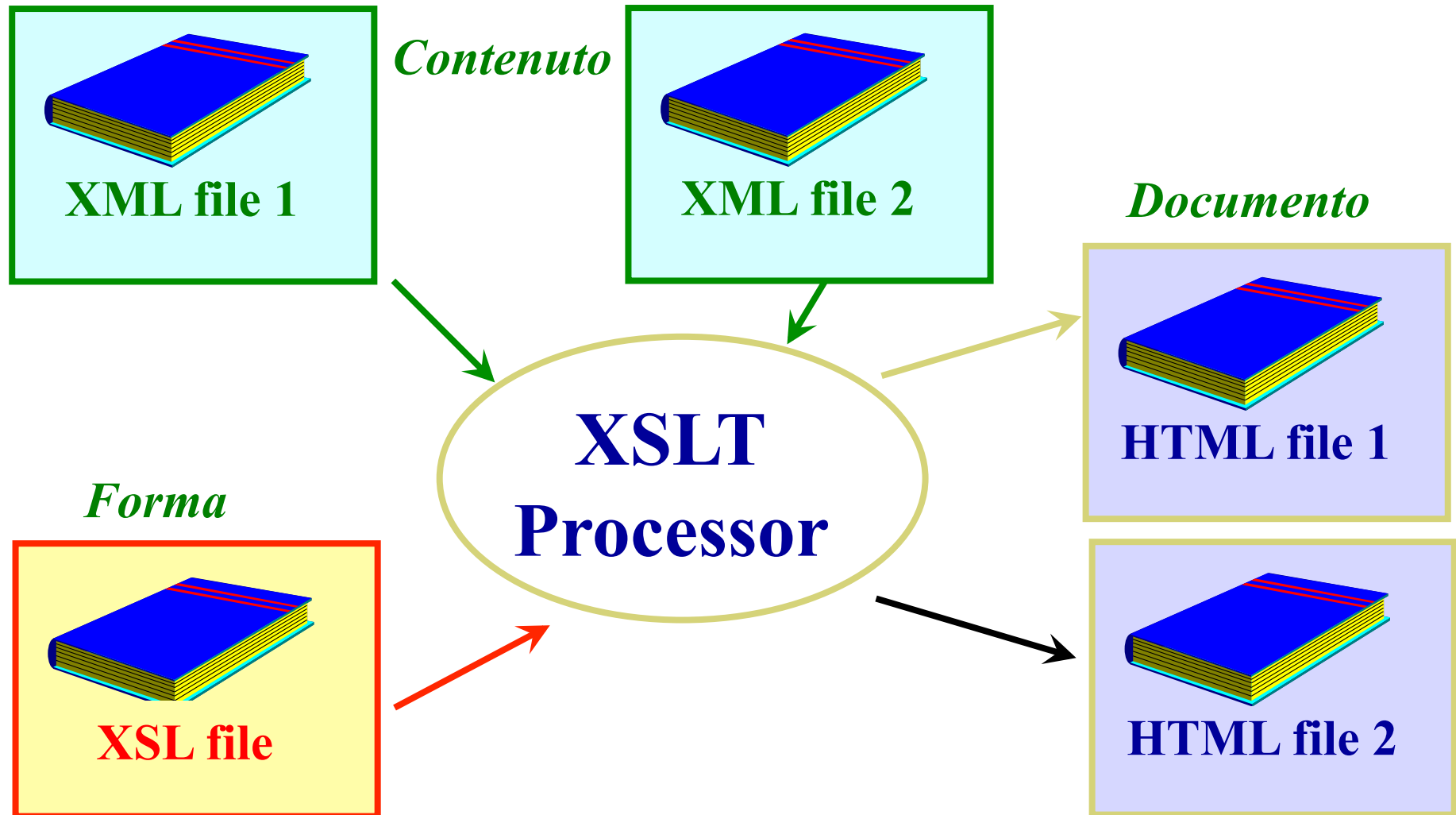
Michael Kay' s SAXON

<http://users.iclway.co.uk/mhkay/saxon>

Transforming XML



Transforming XML



Hands on XSL



XSL-basic elements

HANDS ON! - Esempio1 XML

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet href="hello.xsl" type="text/xsl"?>
```

```
<!-- Here is a sample XML file -->
```

```
<page>
```

```
  <title>Test Page</title>
```

```
  <content>
```

```
    <paragraph>What you see is what you get!</paragraph>
```

```
  </content>
```

```
</page>
```

HANDS ON! - Esempio1 XSL a

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  <xsl:template match="page">  
    <html>  
      <head>  
        <title>  
          <xsl:value-of select="title"/>  
        </title>  
      </head>  
      <body bgcolor="#ffffff">  
        <xsl:apply-templates/>  
      </body>  
    </html>  
  </xsl:template>
```

HANDS ON! - Esempio1 XSL b

```
<xsl:template match="paragraph">
  <p align="center">
    <i>
      <xsl:apply-templates/>
    </i>
  </p>
</xsl:template>
</xsl:stylesheet>
```

HANDS ON! - Esempio1 Xalan

Let us use the Apache XSLT processor: Xalan.

1) Get Xalan from xml.apache.org/xalan/index.html

2)Set CLASSPATH=%CLASSPATH%;**.../xalan.jar; .../xerces.jar**

3) java **org.apache.xalan.xslt.Process**

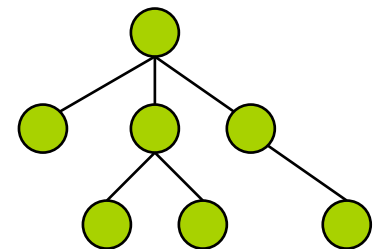
-IN testPage.xml -XSL testPage.xsl -O out.html

HANDS ON! - Esempio1 Output HTML

```
<html>
  <head>
    <title>
      Test Page
    </title>
  </head>
  <body bgcolor="#ffffff">
    <p align="center">
      <i>
        What you see is what you get!
      </i>
    </p>
  </body>
</html>
```

The process

- ❑ The process starts by traversing the document tree, attempting to find a single matching rule for each visited node.
- ❑ Once the rule is found, the body of the rule is instantiated
- ❑ Further processing is specified with the *<xsl:apply-templates>*. The nodes to process are specified in the *match* attribute. If the attribute is omitted, it continues with the next element that has a matching template.



Implicit rules

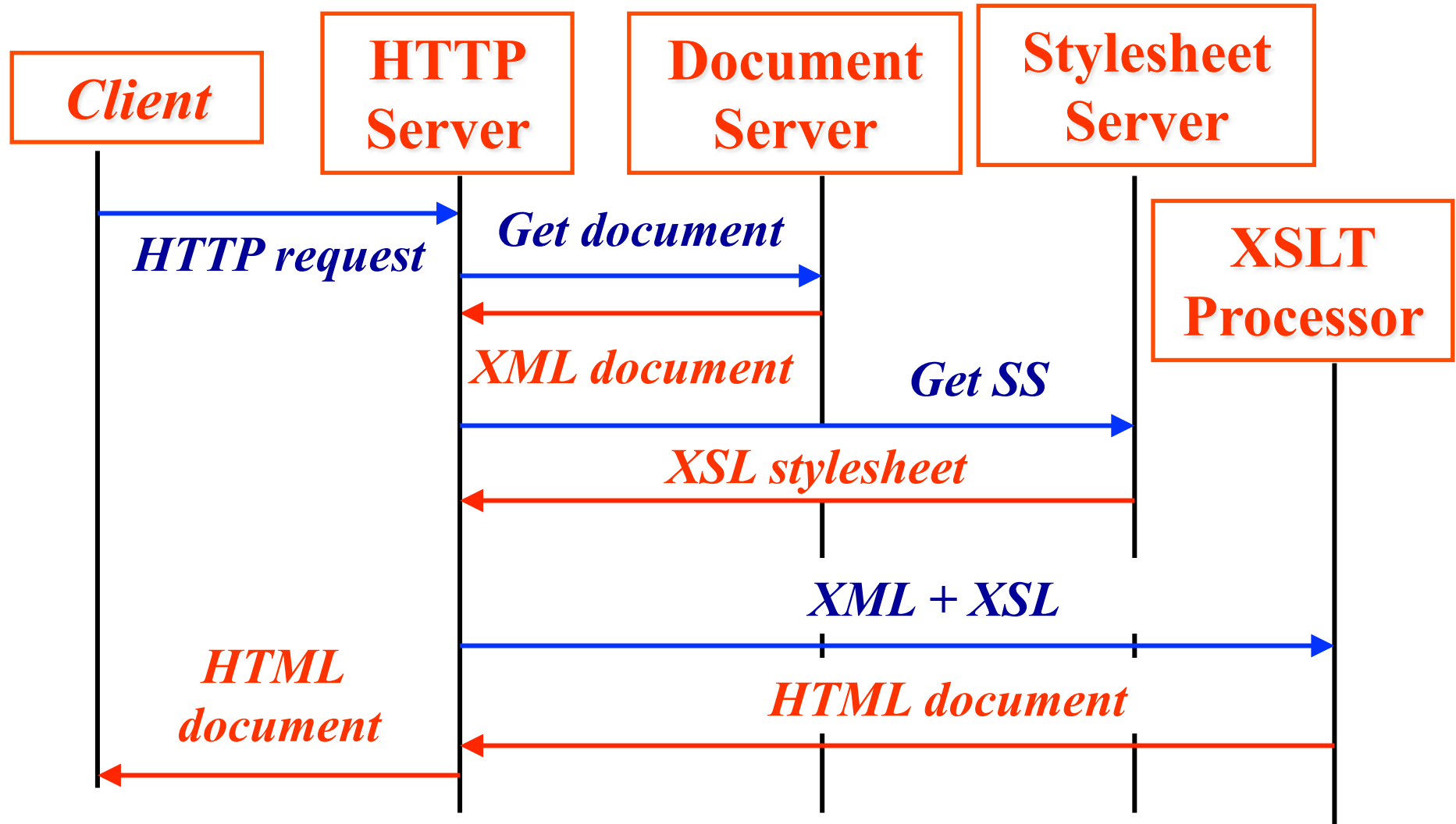
```
<template match="/|*>  
  <apply-templates/>  
</template>
```

```
<template match="text()">  
  <value-of select="."/>  
</template>
```

Publishing frameworks



XML Enabled HTTP Server



Publishing frameworks

Apache Cocoon

<http://xml.apache.org>

Enhydra Application Server

<http://www.enhydra.org/>

Bluestone XML Server

<http://www.bluestone.com/xml>

SAXON

<http://users.iclway.co.uk/mhkay/saxon>