Introduction to XML

XML: basic elements

Markup languages

a markup language is NOT a programming language. It is

- a system for annotating a document (metadata)

in a way that is syntactically distinguishable from the text,

meaning:

when the document is processed for display, the markup language is not shown, and is only used to format the text.

In most cases is human-readable.

Example of Markup Language

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<data>
  <NETWORK>
    <IP>172.150.1.101</IP>
    <IP_LODESERVER>172.150.1.3</IP_LODESERVER>
  </NETWORK>
  <LECTURE id="27">
      <COURSE_NAME>Web Programming</COURSE_NAME>
      <LECTURE_NAME>Introduction to XML</LECTURE_NAME>
      <TEACHER_NAME>Marco Ronchetti</TEACHER_NAME>
       <TIME>5225.00</TIME>
  </LECTURE>
</data>
```

Types of Markup languages - 1

1) Presentational markup

used by traditional word-processing systems: binary codes embedded within document text that produce the WYSIWYG ("what you see is what you get") effect.

Such markup is usually hidden from the human users, even authors and editors.

Types of Markup languages - 2

2) Procedural markup

Markup is embedded in text which provides instructions for programs to process the text, such as e.g. TeX, and PostScript.

The processor runs through the text from beginning to end, following the instructions as encountered.

Text with such markup is often edited with the markup visible and directly manipulated by the author.

```
/Times-Roman findfont
12 scalefont
setfont
newpath
100 200 moveto
(Example 3) show
```

Example 3

PostScript example

Types of Markup languages - 3

3) Descriptive markup

Markup is specifically used to *label parts of the document for what they are*, rather than how they should be processed: e.g. LaTeX, HTML, and XML.

The objective is to decouple the structure of the document from any particular treatment or rendition of it. Such markup is often described as "semantic".

Descriptive markup (*logical markup, conceptual markup, semantic markup*) encourages authors to write in a way that describes the material conceptually, rather than visually.

Example: HTML

File | /Users/ronchet/Downloads/prova.html

Did you know?

Time flies like an arrow,
fruit flies like a banana **G.Marx**

Elements Console Sources Network Performance Memory Application Se

```
...<!DOCTYPE html> == $0
<html lang="en">
▼<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Hi there</title>
  </head>
▼<body>
    <h1>Did you know?</h1>
  ▼<cite>
      "Time flies like an arrow,"
      <br>
      "fruit flies like a banana"
    </cite>
    <b> G.Marx</b>
  </body>
</html>
```

The roots: SGML

What is SGML

SGML is an ISO standard (ISO 8879:1986) which provides a formal notation for the definition of generalized markup languages. SGML is not a language in itself. Rather, it is a metalanguage that is used to define other languages.

SGML: the three parts

An SGML document is really the combination of three parts. Let's refer to the parts as files (but they don't have to be separate physical files).

One file contains the *content* of the document (words, pictures, etc.). This is the part that the author wants to expose to the client.

A second file is the *grammar* (*DTD – data type definition)* that defines the accepted syntax.

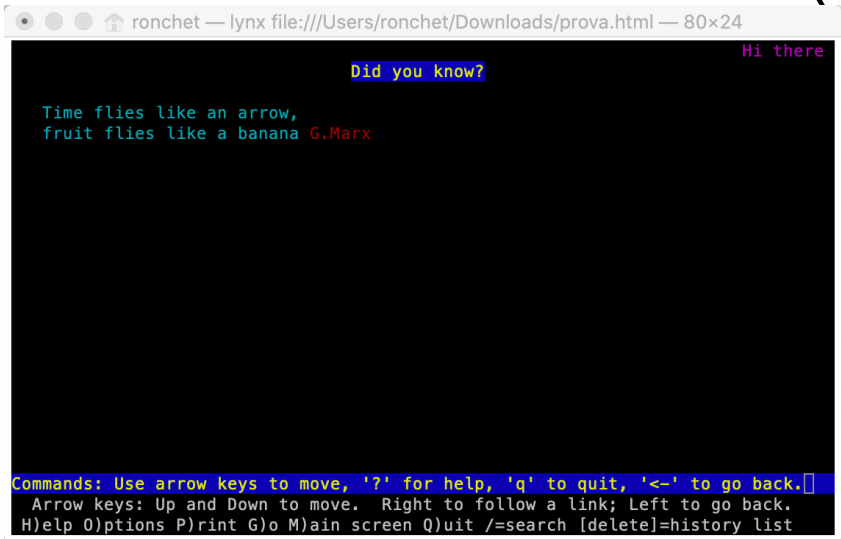
A third file is a *stylesheet* that establishes how the content that conforms to the grammar is to be rendered on the output device.

HTML versus SGML

HTML implements some of the concepts derived from SGML but in effect the DTD is hard-coded into the browser software.

Also a (base) Style Sheet is hard-coded into the browser (but can be redefined via CSS – cascading style sheet)

Because each browser manufacturer has some flexibility in implementing the intended style, the same document could look different when rendered with two different browsers. This is a (wanted) shortcoming of HTML.



XML

“Trying to wrap your brain around XML is sort of like trying to put an octopus in a bottle. Every time you think you have it under control, a new tentacle shows up. XML has many tentacles, reaching out in all directions. “

(Dick Baldwin)

```
<book>
    <chap>
        Text for Chapter 1
    </chap>
    <chap>
        Text for Chapter 2
    </chap>
</book>
```

What is XML?

eXtensible Markup Language, or XML for short, is a new technology for web applications.

XML is a World Wide Web Consortium standard that lets you create your own tags.

XML is not a single technology, but a group of related technologies that continually adds new members

XML is a *lingua-franca* that simplifies business-to-business transactions on the web.

Vendor independence in the data-formatting context

"Other successful Internet technologies let people run their systems without having to take into account another company's own computer systems, notably:
TCP/IP for networking,
Java for programming,
Web browsers for content delivery.
XML fills the data formatting piece of the puzzle."

"These technologies do not create dependencies. It means you can build solutions that are completely agnostic about the platforms and software that you use."

Phipps, IBM's chief XML and Java evangelist

XML Jargon

- Computer people are the world's worst at inventing new jargon.
- XML people seem to be the worst of the worst in this regard.

(Dick Baldwin)

XML
DTD
XSL
XSLT

DOM
SAX
JAXP
JDOM

XML Schema
XPath
XLink
XPointer

XQL
XML-RPC
XSP

Related stuff

SGML XHTML CSS

XML applications

Semantic Web

RDF (Resource Description Framework), OWL, Topic Maps

Web Services

SOAP, UDDI, WSDL, XML-RPC

Configuration files

XML: element, content, and attribute

What is a tag?

A tag is a CASE SENSITIVE sequence of characters that begins with < and ends with >

Every tag must be closed with an end tag, which begins with </

What is an element?

An element is a sequence of characters that begins with a *start tag* and ends with an *end tag* and includes everything in between.

```
<chap number="1">Text for Chapter 1</chap>
```

What is the content?

The characters in between the tags (rendered in green in this presentation) constitute the CONTENT.

See https://www.w3schools.com/xml/xml_elements.asp

XML: element, content, and attribute

An element may include optional attributes

The *start tag* may contain optional *attributes*. In this example, a single *attribute* provides the number value for the chapter.

```
<chap number="1">Text for Chapter 1</chap>
```

The characters rendered in blue in the above element constitute an attribute.

See https://www.w3schools.com/xml/xml_attributes.asp

Well formed documents

All XML documents must be well-formed

XML documents need not be valid, but all XML documents must be well-formed.

(HTML documents are not required to be well-formed)

There are several requirements for an XML document to be well-formed.

Well formed documents

Caution: XML is case sensitive

Start and end tags are required

To be well-formed, all elements that can contain character data must have both start and end tags.

(Empty elements have a different requirement: see later.)

For purposes of this explanation, let's just say that the *content* that we discussed earlier comprises character data.

Elements must nest properly

If one element contains another element, the entire second element must be defined inside the start and end tags of the first element.

Well formed documents

Dealing with empty elements

We can deal with empty elements by writing them in either of the following two ways:

```
<book></book>
<book/>
```

You will recognize the first format as simply writing a start tag followed immediately by an end tag with nothing in between.

The second format is preferable

Empty element can contain attributes

Note that an empty element can contain one or more attributes inside the start tag:

```
<book author="eckel" price="$39.95" />
```

Well formed documents

No markup characters are allowed

For a document to be well-formed, it must not have some characters (entities) in the text data: < > “ ‘ &.

If you need for your text to include the < character you can represent it using < or < or &#x3C instead.

All attribute values must be in quotes (apostrophes or double quotes).

You can surround the value with apostrophes (single quotes) if the attribute value contains a double quote. An attribute value that is surrounded by double quotes can contain apostrophes.

XML: tree structure

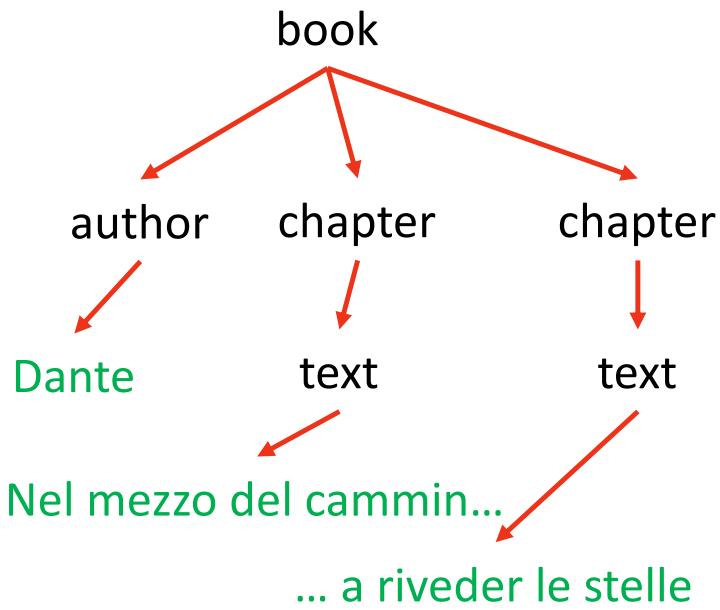
An XML document must have a root tag.

An XML document is an information unit that can be seen in two ways:

As a linear sequence of characters that contain characters data and markup.

As an abstract data structure that is a tree of nodes.

```
<book>
  <author>Dante</author>
  <chapter id=1>
     <text>Nel mezzo del cammin…<text>
  </chapter>
  <chapter id=2>
     <text>… a riveder le stelle</text>
  </chapter>
</book>
```



See https://www.w3schools.com/xml/xml_tree.asp

XML: Which tags can I use?

You define them!

Provide a grammar to:

- **define tags**
- **define rules for the tags**
  - **allowed attributes**
  - **containment rules**

The grammar is defined in a

- **DTD file**
- **XML-Schema file**

examples later

or is NOT DEFINED AT ALL!

XML: additional elements

An XML document can contain:

Processing Instructions (PI): **<? … ?>**

Comments **<!-- … -->**

When a XML document is analyzed, character data within comments or PIs are ignored.

The content of comments is ignored, the content of PIs is passed on to applications.

XML: CDATA sections

Note: the element content that are going to be parsed are called **PCDATA**

An XML document can contain sections used to escape character strings that may contain elements that you do not want to be examined by your XML engine, e.g. special chars (<) or tags:

CDATA sections **<![CDATA[…]]>**

When a XML document is analyzed, character data within a CDATA section are not parsed, by they remain as part of the element content.

```
<java>
<![CDATA[
   if (arr[indexArr[4] ]>3) System.out.println("<HTML>");
]]>
</java>
```

Avoid having]]> in your CDATA section!

Logical structure of an XML document

- XML declaration (or "Prolog": optional, but if present MUST be the first element)

  <?xml version=‘1.0’ encoding=‘utf-8’>

- Optional DTD declaration
- Optional comments and Processing Instructions
- The root element’s start tag
- All other elements, comments and PIs
- The root element closing tag

See https://www.w3schools.com/xml/xml_syntax.asp

XML: namespaces

How do you avoid tag conflicts?

Since you can define your own tags, if you reuse XML files from other authors you might find tag conflicts.

These can be avoided by declaring a namespace as an attribute of the root element:

```
<xsl:stylesheet version =“1.0”
  xmlns:xsl=“http://www.w3.org/1999/XSL/Transform”>
```

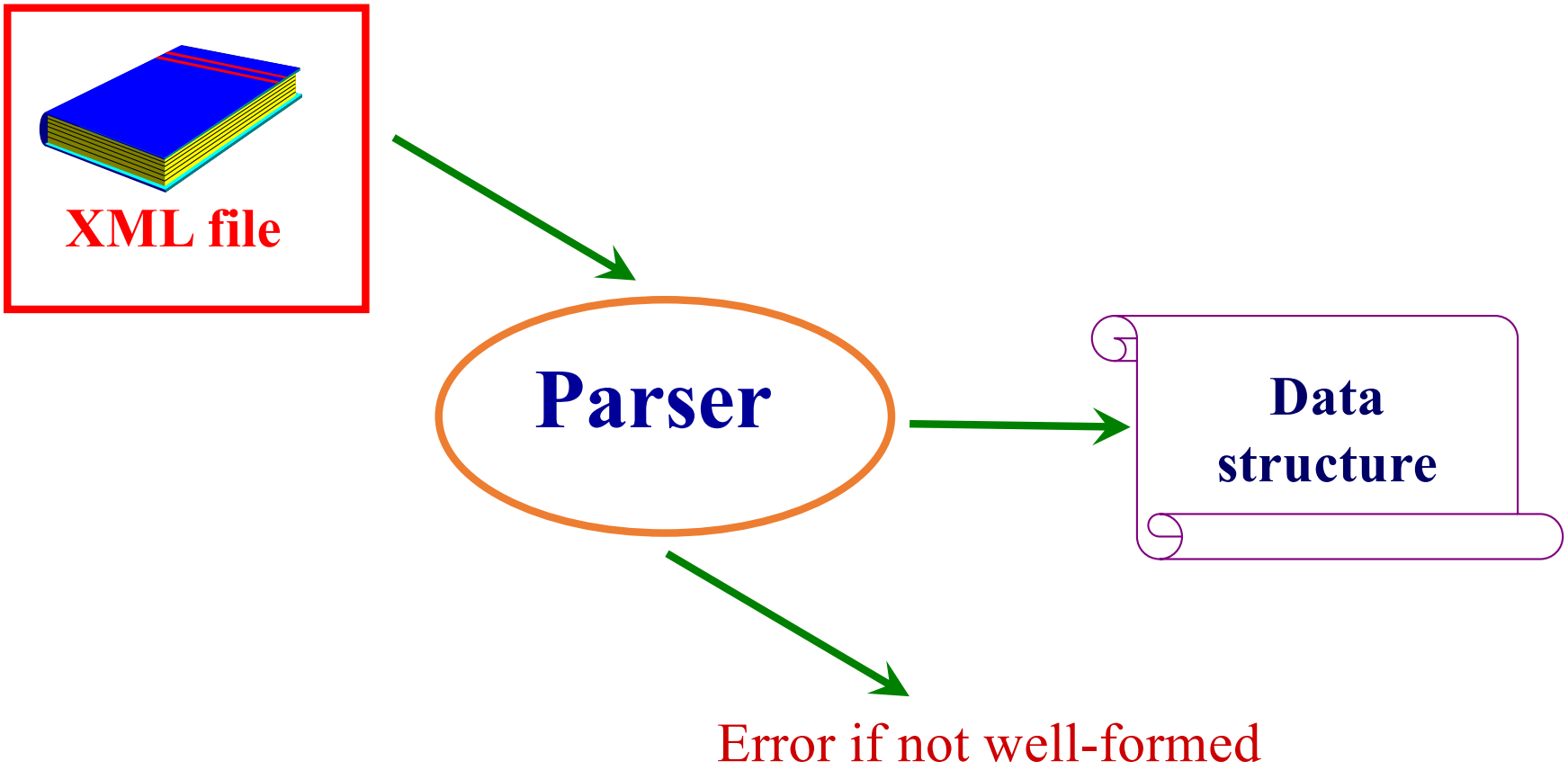
(more about namespaces in the next lectures)

What is a parser?

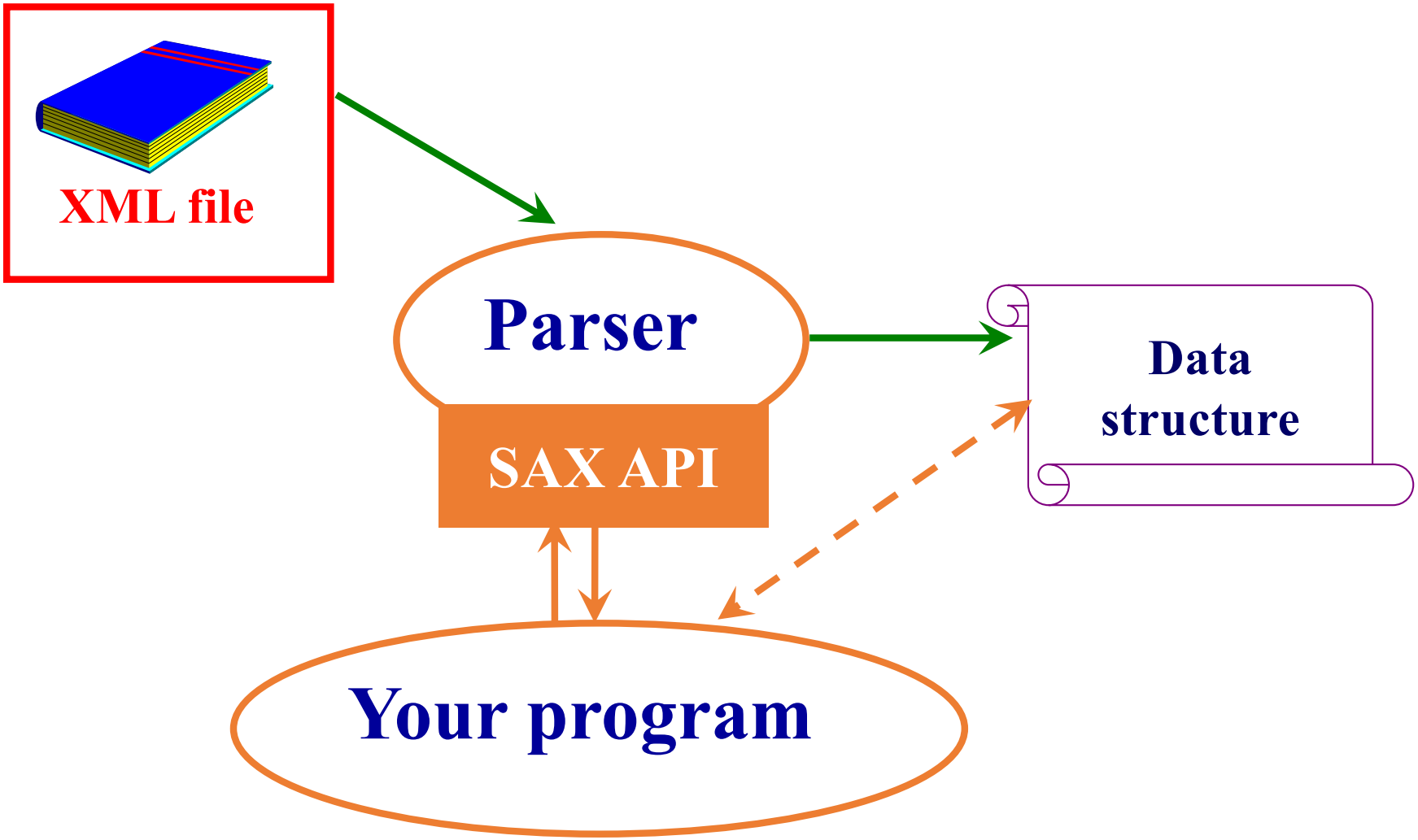
A parser, in this context, is a software tool that preprocesses an XML document in some fashion, handing the results over to an application program.

The primary purpose of the parser is to do most of the hard work up front and to provide the application program with the XML information in a form that is easier to work with.

Making sense of XML: the Parser



Making sense of XML:the Parser



Tree-based vs Event-based API

- **Tree-based API**
  A tree-based API compiles an XML document into an internal tree structure. This makes it possible for an application program to navigate the tree to achieve its objective. The Document Object Model (DOM) working group at the W3C developed a standard tree-based API for XML.

- **Event-based API**
  An event-based API reports parsing events (such as the start and end of elements) to the application using *callbacks*. The application implements and registers event handlers for the different events. Code in the event handlers is designed to achieve the objective of the application. The process is similar to creating and registering event listeners in the Event Model by Java and other languages.

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.

For a tutorial, see https://www.w3schools.com/xml/xml_dtd_intro.asp

Valid documents

An XML document is *valid* if it conforms to an existing grammar 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.

A *non well-formed* document cannot be valid, and is not an XML document

Valid documents

Validity is not a requirement of XML

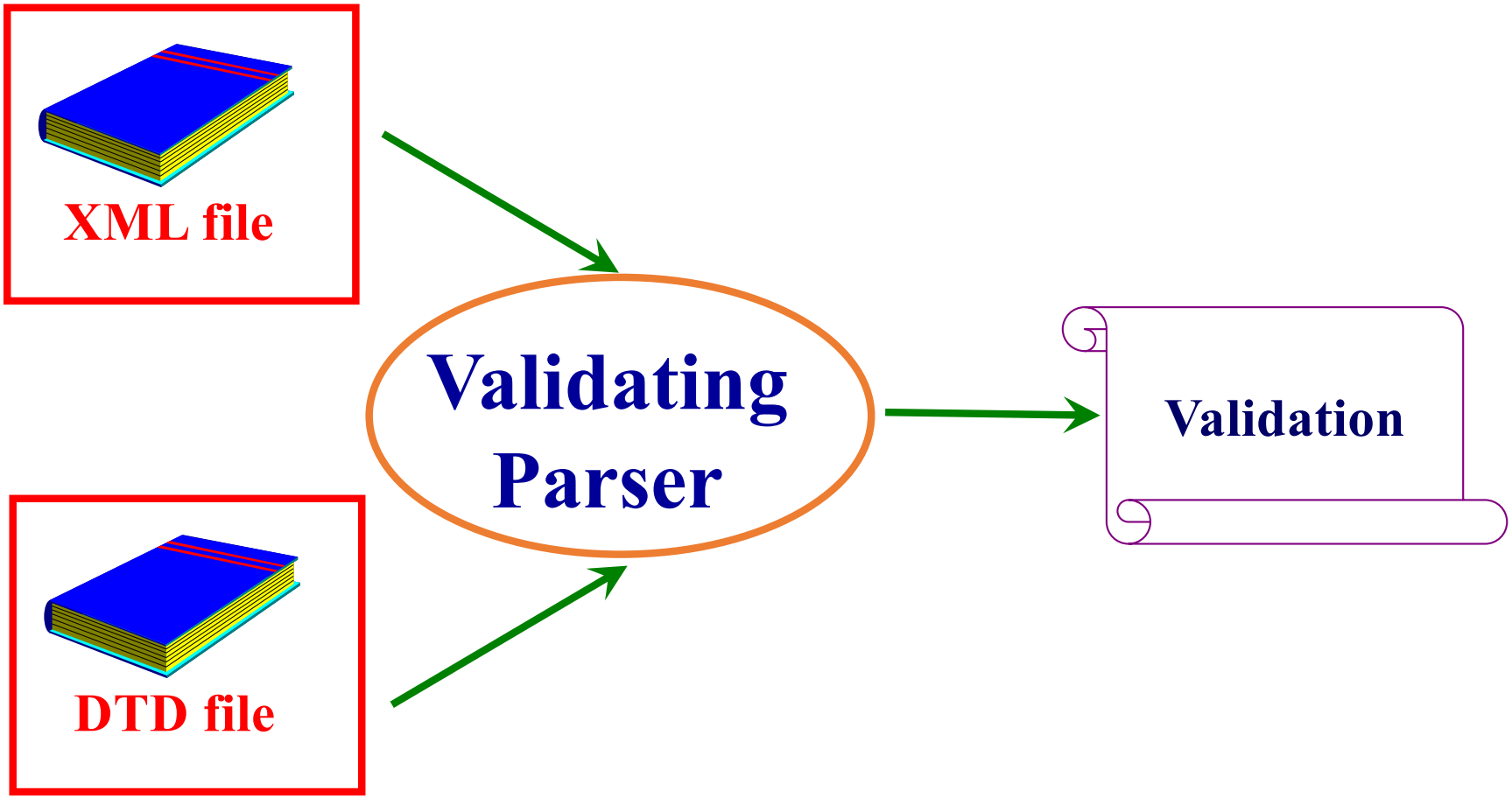
Because XML does not require a DTD, in general, an XML processor cannot require validation of the document.

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.

Constraing &ValidatingXML



Where are the DTDs?

A DTD can be external or internal to a document.

Internal DTD

```
<!DOCTYPE Report>
<!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

The main problem of DTD's...

They are not written in XML!

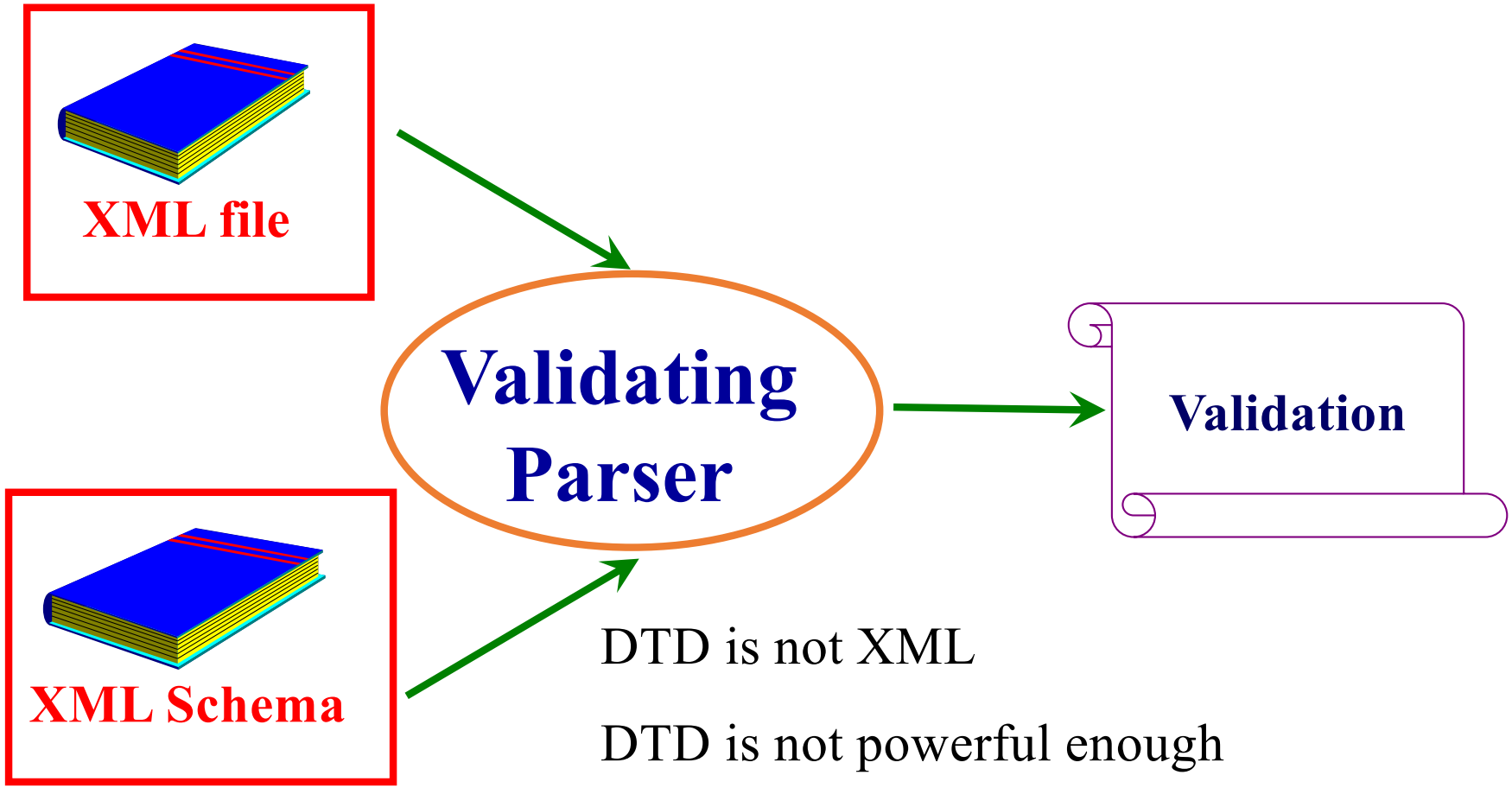
Solution:

Another XML-based standard: XML Schema

For more info see:

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

Constraing & Validating XML



DTD is not XML

DTD is not powerful enough

(e.g. at least 3, no more than 5)

(A simplified) XML-Schema

```
<?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>
```

Defines tags such that the following is valid:

```
<complete_name>
   <nome>Marta</nome>
   <cognome>Bassino</cognome>
</complete_name>
```

For a tutorial, see https://www.w3schools.com/xml/schema_intro.asp

Introduction to XML

Now that I know about xml, what can I do with it?

What can I do with XML?

Navigate its data structure:

- **DOM, JDOM**
- **JAXP**
- **XPath**
- **SAX**

Query XML data:

- **XQuery**

Transform XML data:

- **XSLT**

Use XML for Single Page Web Applications

- **AJAX**

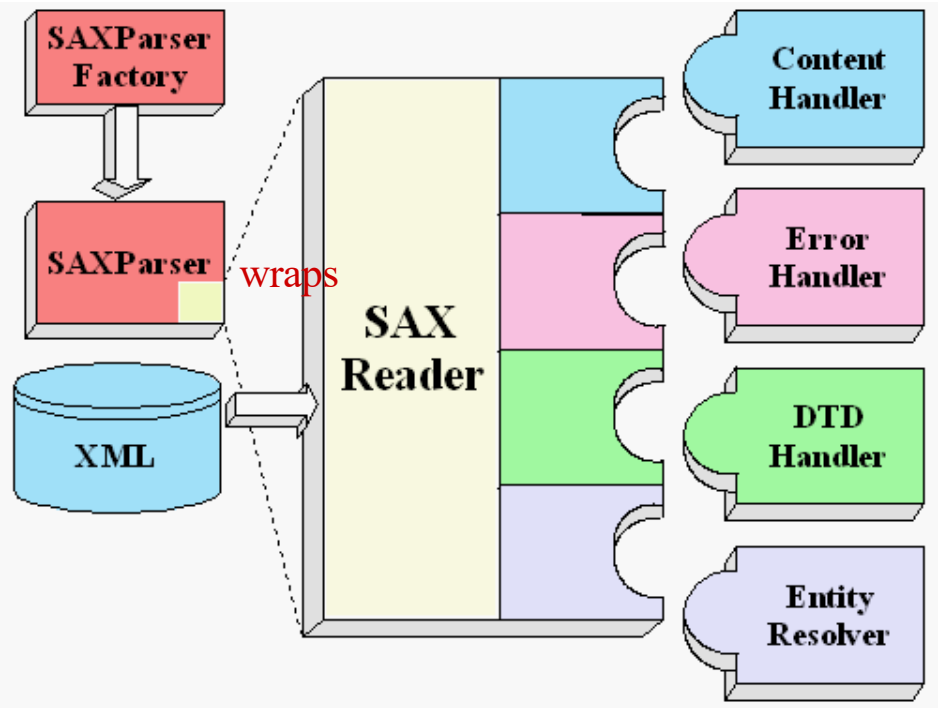
Use XML in configuration files

SAX architecture and example

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true); //optional - default is non-validating
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(File f, DefaultHandler-subclass h)
```

File containing input XML

Default-handler (classe che implementa le callback)



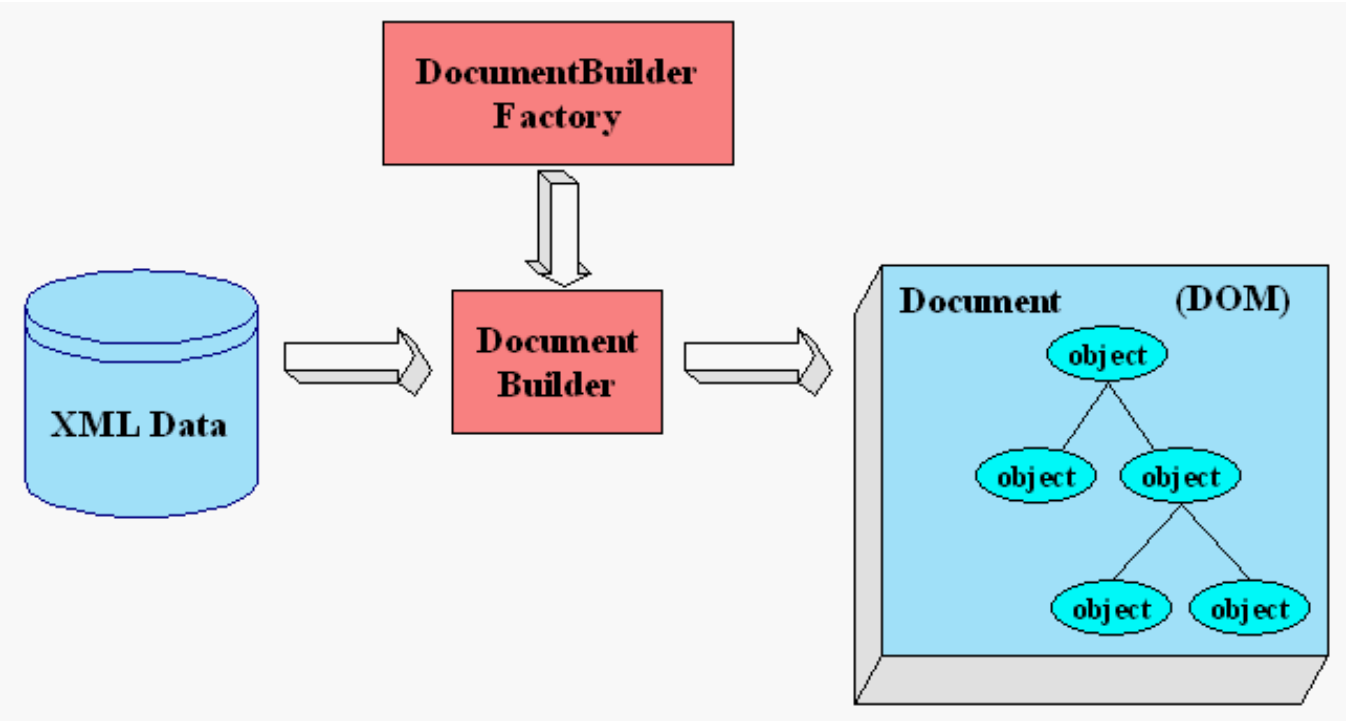
Interfaces implemented by DefaultHandler class

SAX callbacks

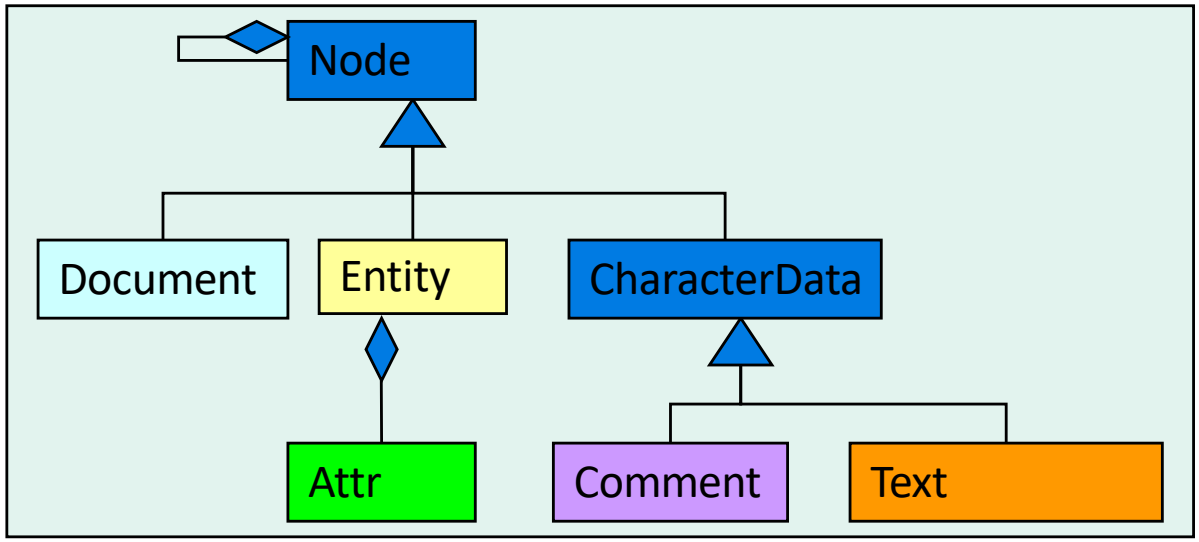
- // ---------- ContentHandler methods
- void characters(char[] ch, int start, int length)
- void startDocument()
- void startElement(String name, AttributeList attrs)
- void endElement(String name)
- void endDocument()
- void processingInstruction(String target,String data)

JAXP example: the DOM

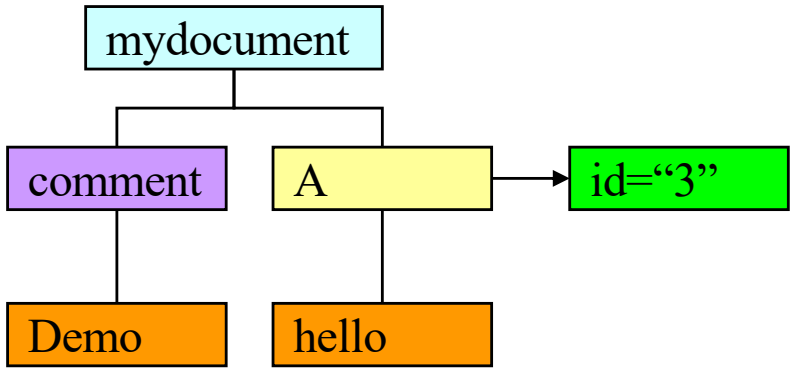
```
DocumentBuilderFactory dbf =  DocumentBuilderFactory.newInstance();
dbf.setValidating(true); // optional – default is non-validating
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(file);
```



The Node hierarchy




```
<!-- Demo -->
<A id="3">hello</A>
```



JAXP example

```
public int getElementCount(Node node) {
  if (null == node)  return 0;
  int sum = 0;
  boolean isElement = (node.getNodeType() == Node.ELEMENT_NODE);
  if (isElement)   sum = 1;
  NodeList children = node.getChildNodes();
  if (null == children)       return sum;

  for (int i = 0; i < children.getLength(); i++) {
    sum += getElementCount(children.item(i));   // recursive call
  }
  return sum;
 }
}
```

use DOM methods to count elements:
for each subtree if the root is an Element,
 set sum to 1, else to 0;
 add element count of all children of the root to sum

XPath

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT
- XPath is a W3C Standard

See https://www.w3schools.com/xml/xpath_intro.asp

XPath example

```
// prepare the XPath expression
   XPathFactory factory = XPathFactory.newInstance();
   XPath xpath = factory.newXPath();
   XPathExpression expr = xpath.compile("//book[author='Dante Alighieri']/title/text()");
// evaluate the expression on a Node
   Object result = expr.evaluate(doc, XPathConstants.NODESET);
// examine the results
   NodeList nodes = (NodeList) result;
   for (int i = 0; i < nodes.getLength(); i++) {
    System.out.println(nodes.item(i).getNodeValue());
   }
 }
```