

Data for the Web

INSA Lyon

IT&CS department

3th year

Part 3: XML Galaxy

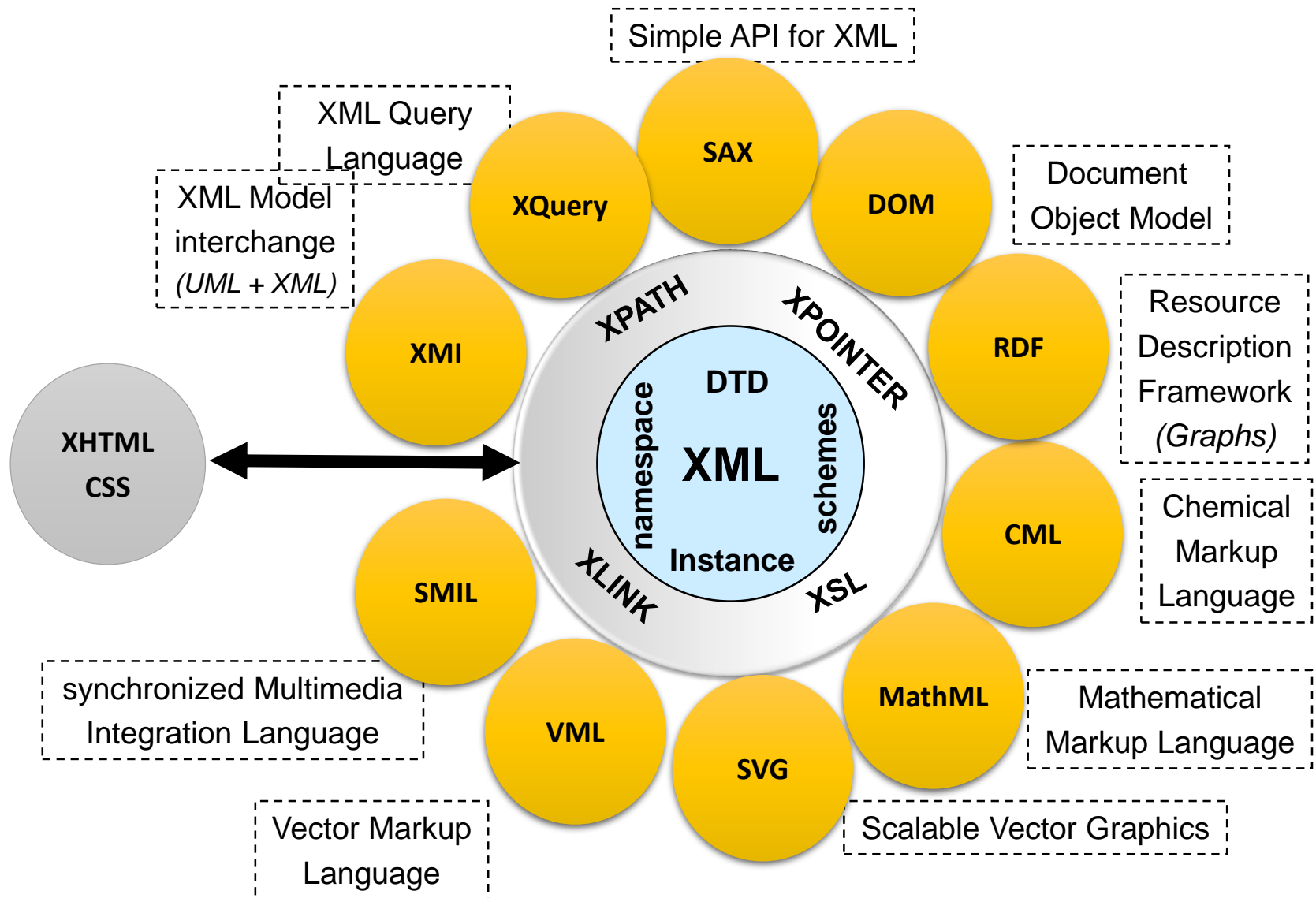
Előd EGYED-ZSIGMOND

Plan

- Introduction
- Core XML
- XML galaxy
 - XPath
 - XSL
 - Handling
 - Applications
- NOSQL
- Conclusion

The XML galaxy

(Partial)



Plan

Introduction

Core XML

XML galaxy

XPath

XSL

Handling

applications

NOSQL

Conclusion

XPath

Principles

- The axes

- The filters

- The predicates

- Basic functions

- Examples

What is XPath?

- XPath is a syntax used for selecting parts of an XML document
- The way XPath describes paths to elements is similar to the way an operating system describes paths to files
- XPath is almost a small programming language; it has functions, tests, and expressions
- XPath is a W3C standard
- XPath is not itself written as XML, but is used heavily in XSLT
- We talk about *Path expressions*

Terminology

<library>
 <book>

 <chapter>
 </chapter>

 <chapter>
 <section>
 <paragraph/>
 <paragraph/>
 </section>
 </chapter>

 </book>
 </library>

- **library** is the **parent** of **book**; **book** is the parent of the two **chapters**
- The two **chapters** are the **children** of **book**, and the **section** is the child of the second **chapter**
- The two **chapters** of the **book** are **siblings** (they have the same parent)
- **library**, **book**, and the second **chapter** are the **ancestors** of the **section**
- The two **chapters**, the **section**, and the two **paragraphs** are the **descendents** of the **book**

Objectives

Locate - reach fragments of the document

- Through absolute addressing
 - exact knowledge of the structure manipulated
 - Full path leading to the fragments to reach
- By filtering
 - partial knowledge of the manipulated structure
 - Reaching components whose position is not known a priori
 - Information on the shape of the path leading to it and / or the content of traversed nodes.

Objectives

- Uniform treatment of all components in a document
Elements - attributes - text
- Reaching and constrain:
 - A given component
 - The element **title** or attribute **Num**
 - Its content model
 - A **title** containing the word **introduction**, a **Section** containing **title**
 - The path to reach a component (taking (possibly) into account its environment)
 - A **title** contained in a **Section** or a **title** that precedes a **Paragraph**

Example

```
/library//chapter[count(paragraph)>4]
```

XPath serves in

- XPath is used in particular:
 - In *XSL* and *XSLT* to locate a specific node or set of nodes to associate their presentation / transformation
 - In *XPointer* to identify document fragments
 - In *XLink* to point to a specific target within a document.
 - In *XQuery* to locate a specific node or set of nodes
 - In *XML Schema* to express domain names
 - ...

Document Template

- For XPath, an XML document is a tree consisting of nodes of various types
 - **Document:** Root node,
 - Single and compulsory
 - Distinct from the root element of the document
 - **Element:** Element node,
 - **Attribute:** Attribute node
 - **text:** Text node,
 - **ProcessingInstruction:** Processing instruction node
 - **Comment:** Comment Node
- nodes ordered sequentially according to the reading order of the XML file

Element nodes

- An node element is associated with each element
 - It is labeled with the name of the item
 - It may contain elements nodes and / or text nodes
 - It is impossible to have two consecutive text nodes
 - The content of the item is ordered
- Remarks:
 - We can associate to a node element one or more attribute nodes
 - An node element is the parent node of the attribute but the attribute node is not a child of its parent ...

Attribute nodes and text nodes

- An attribute node
 - Is labeled with the name of the attribute
 - It contains its value
 - There is no order between the attributes nodes of the same element
- A text node is labeled with its value
- Remarks:
 - Children and descendants of a node are never of type attribute

Reference example: Structure and tags

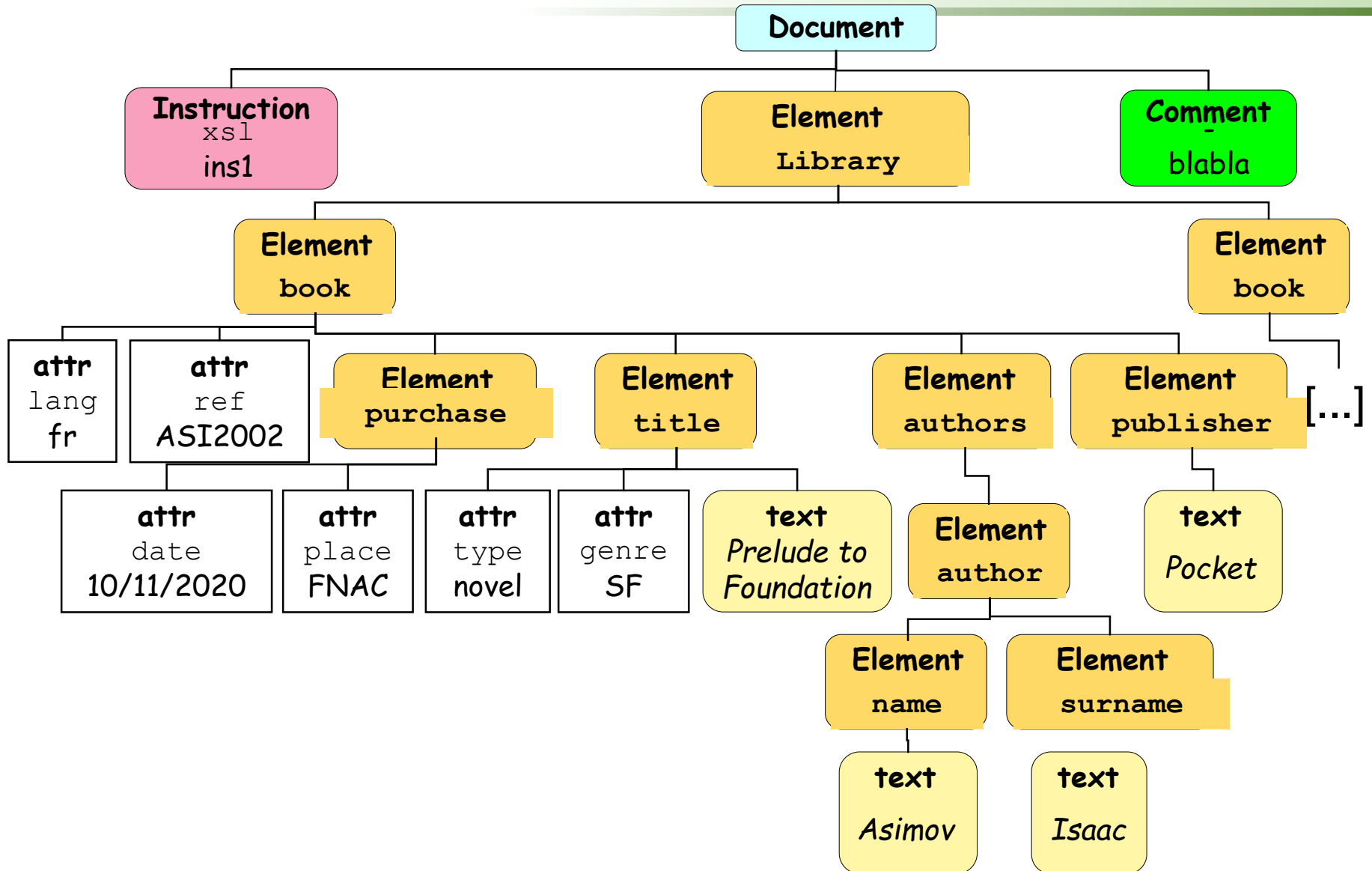
```
<?xml version = "1.0" encoding = "ISO-8859-1" standalone = "no"?>
```

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

```
<Library>
  <book lang="fr" ref="ASI2002">
    <purchase date="10/11/2003" location="FNAC"/>
    <title type="novel" genre="SF"> Prelude foundation </title>
    <authors>
      <author>
        <name> Asimov </name>
        <surname> Isaac </surname>
      </author>
    </authors>
    <publisher> Pocket </publisher>
  </book>
  <book>
    [...]
  </book>
</Library>
```

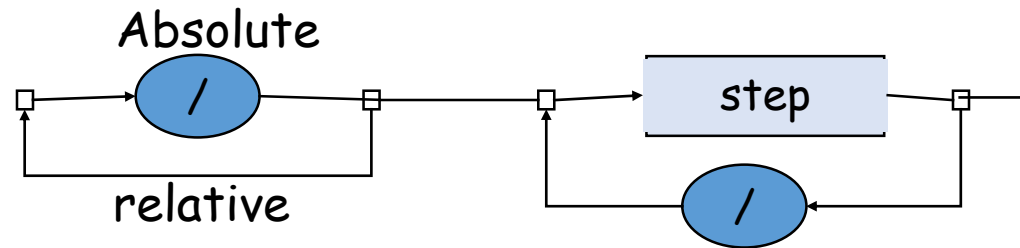
```
<!-- blabla-->
```

XPath Reference Example: Tree



XPath paths expressions

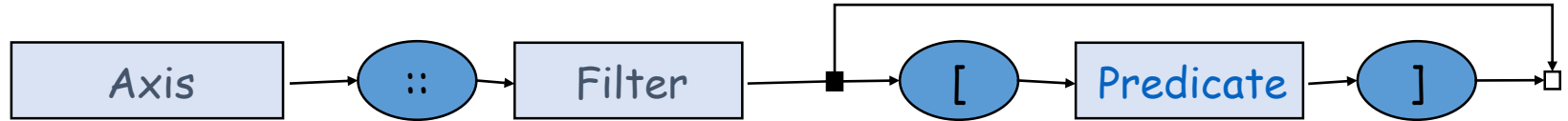
- An XPath expression:
 - Evaluates according to a context node
 - Describes one or more paths in the tree from this node
 - Results a set of nodes or a numeric, alphanumeric or boolean value
- Path through the document tree: step sequence



- A path can be:
 - Absolute: `/book/purchase/@place`
(Context = root of the doc)
 - Relative: `book/purchase/@place`

XPath steps

- A step: 3 components



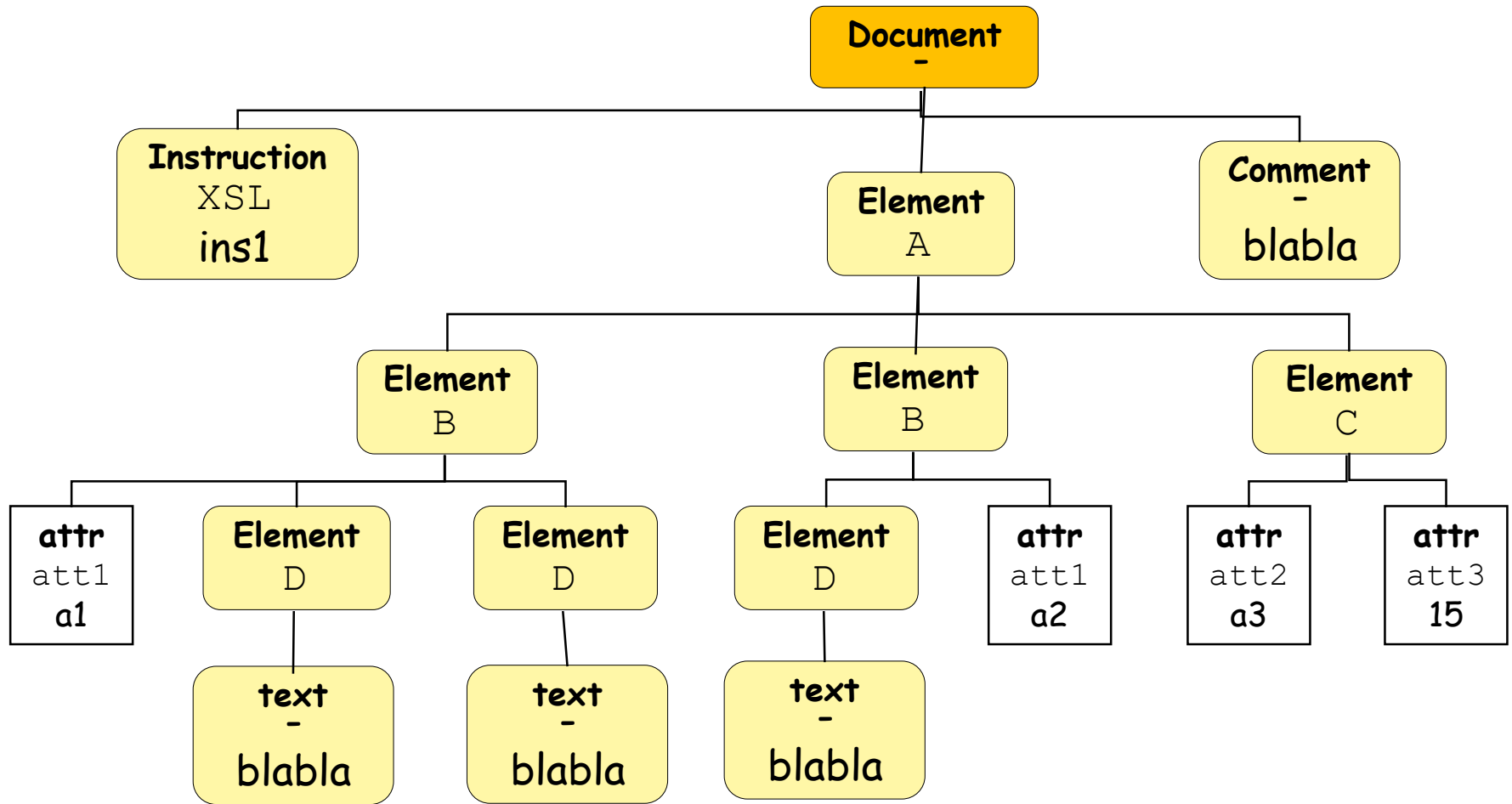
- An *axis*: a *genealogical* relation related to a node
 - A *filter*: The type of nodes that will be retained
 - A set of optional *predicates*: define the properties the selected nodes must meet
-
- One can make a path union

`//book|purchase/@place`

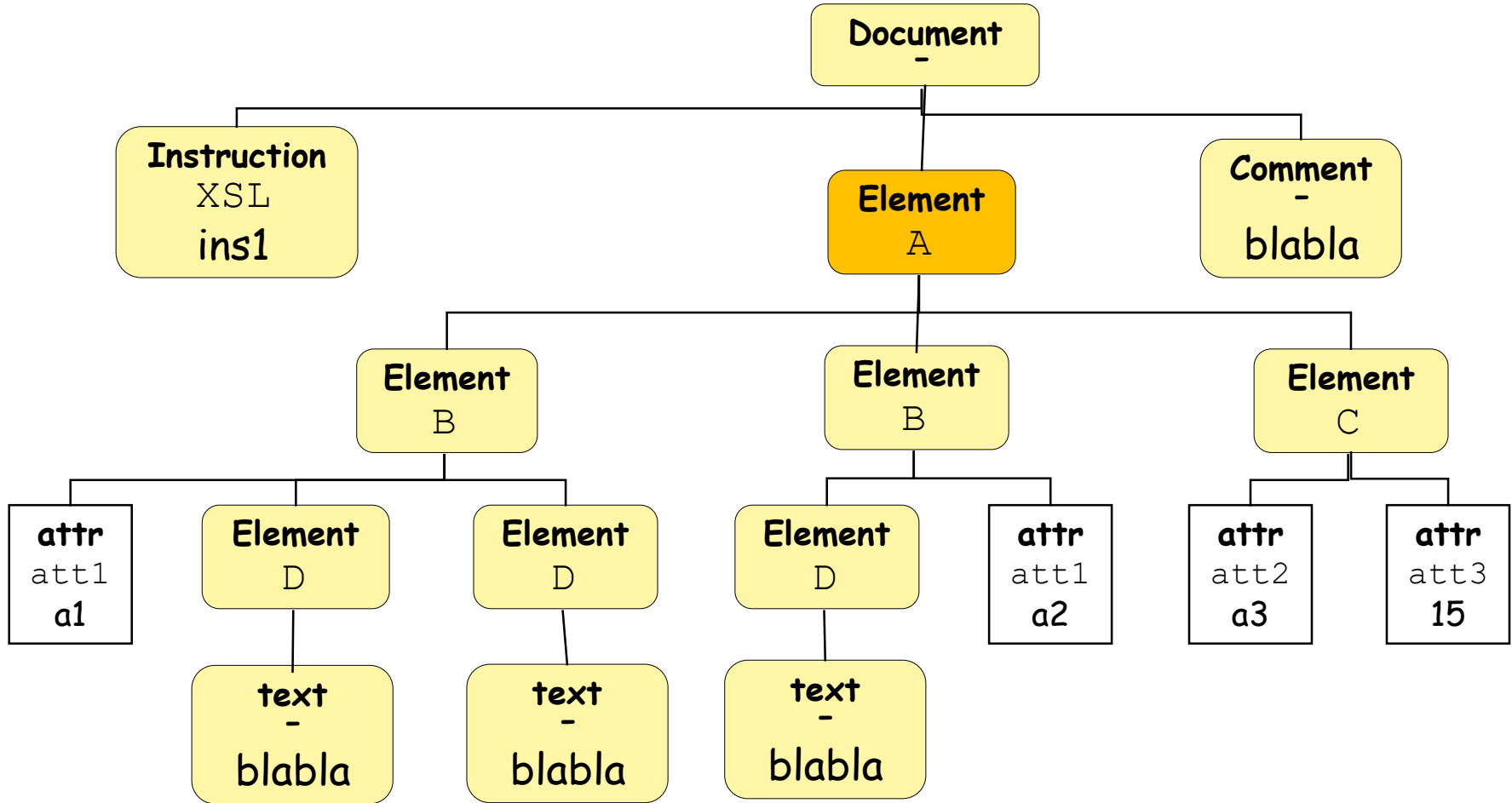
Evaluating XPath

- Each step is evaluated taking into account its **context**:
 - Node, initial position of the path
1. Evaluation of the first step from the context node:
 - This gives a set of nodes
 - **Size** of this set (function `last()`)
 - **Position** of the context node in this set (`position()` function)
 2. Evaluation of the following step:
 - The nodes of the result set are considered one by one as a context node for the next evaluation
 - At each stage, successively take as context node each result node of the previous step.

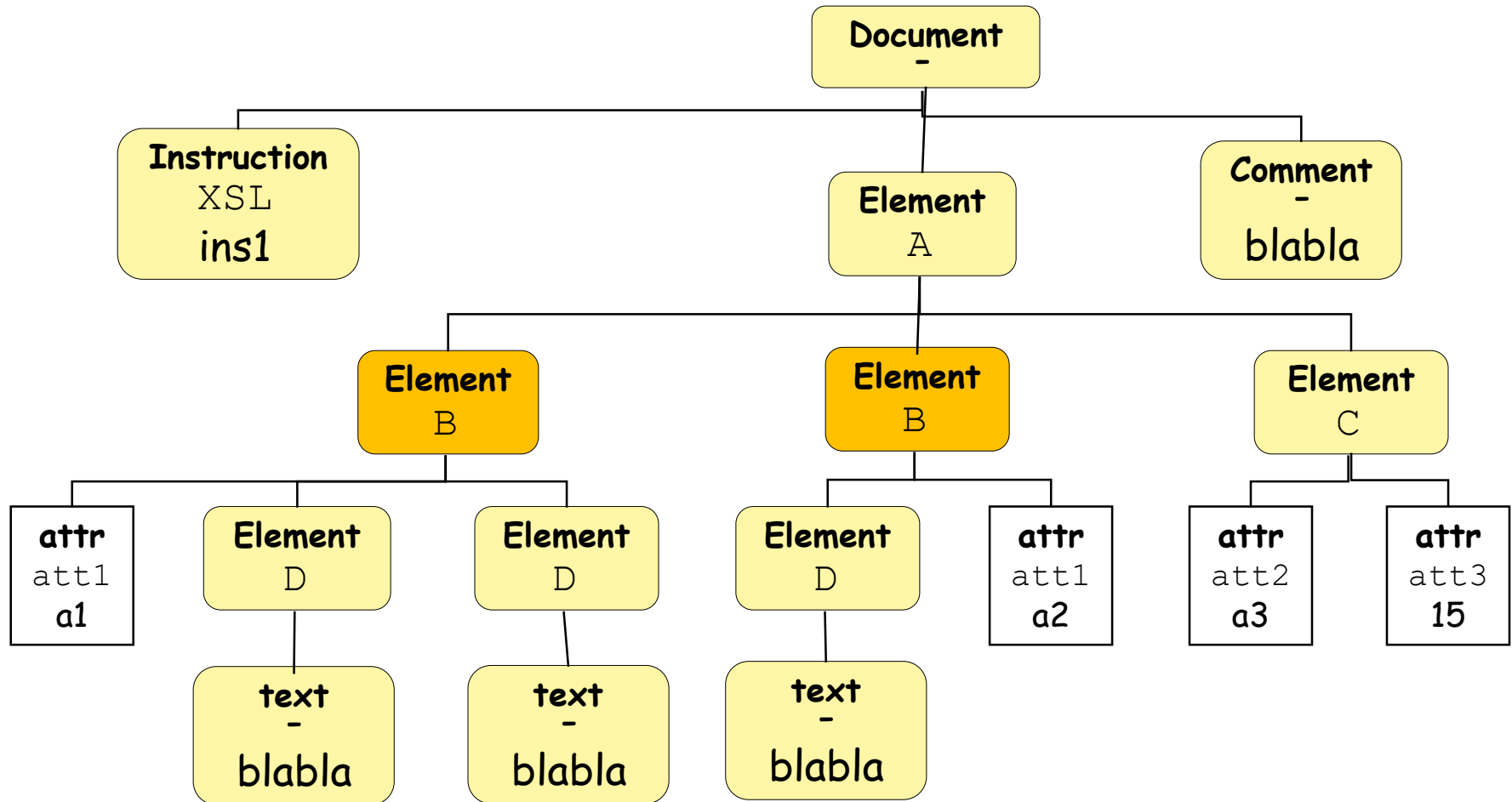
/A/B/@att1: Initial node



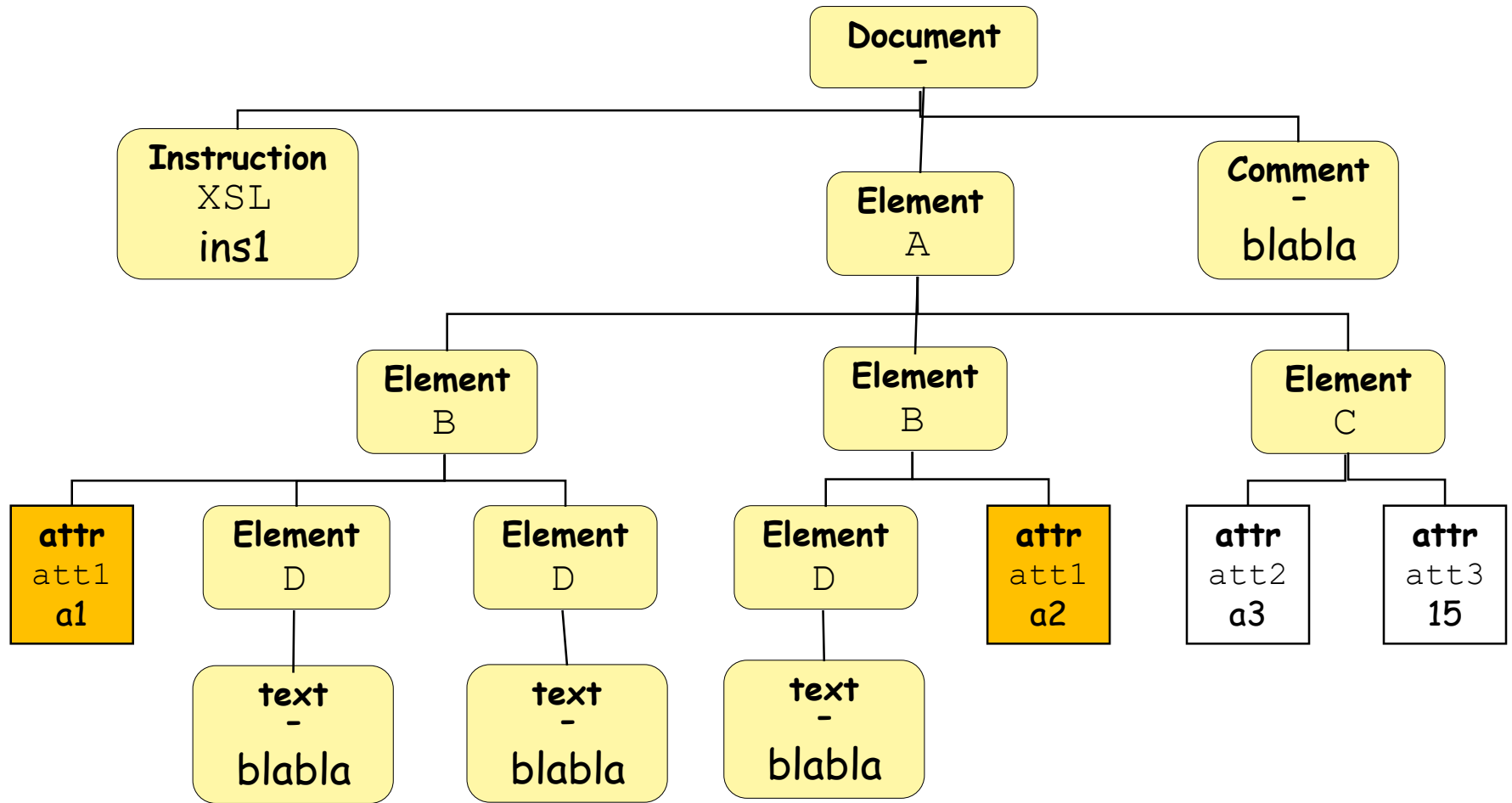
/A/B/@att1: first step



/A/B/@att1: second step



/A/B/@att1: third step



XPath

Principles

The axis

The filters

The predicates

Basic functions

Examples

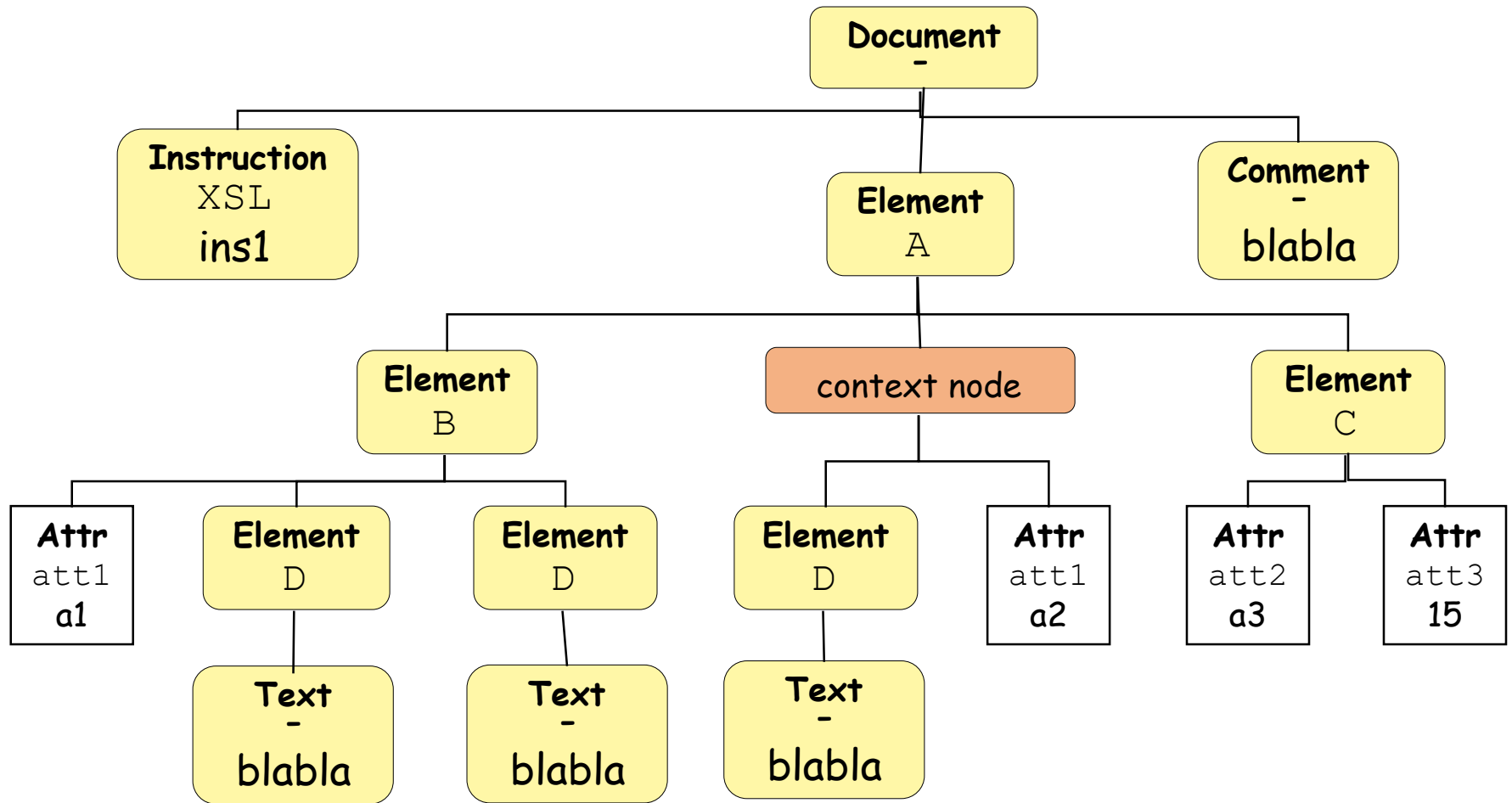
Axis

- Specifies the direction to follow - a direction- to reach the following nodes
 - direct or indirect hierarchy between nodes
 - relative position of the nodes to each other
- Involves a given scanning order of the nodes
- An axis has a type
 - **Attribute**
 - **Element**
 - **namespace**

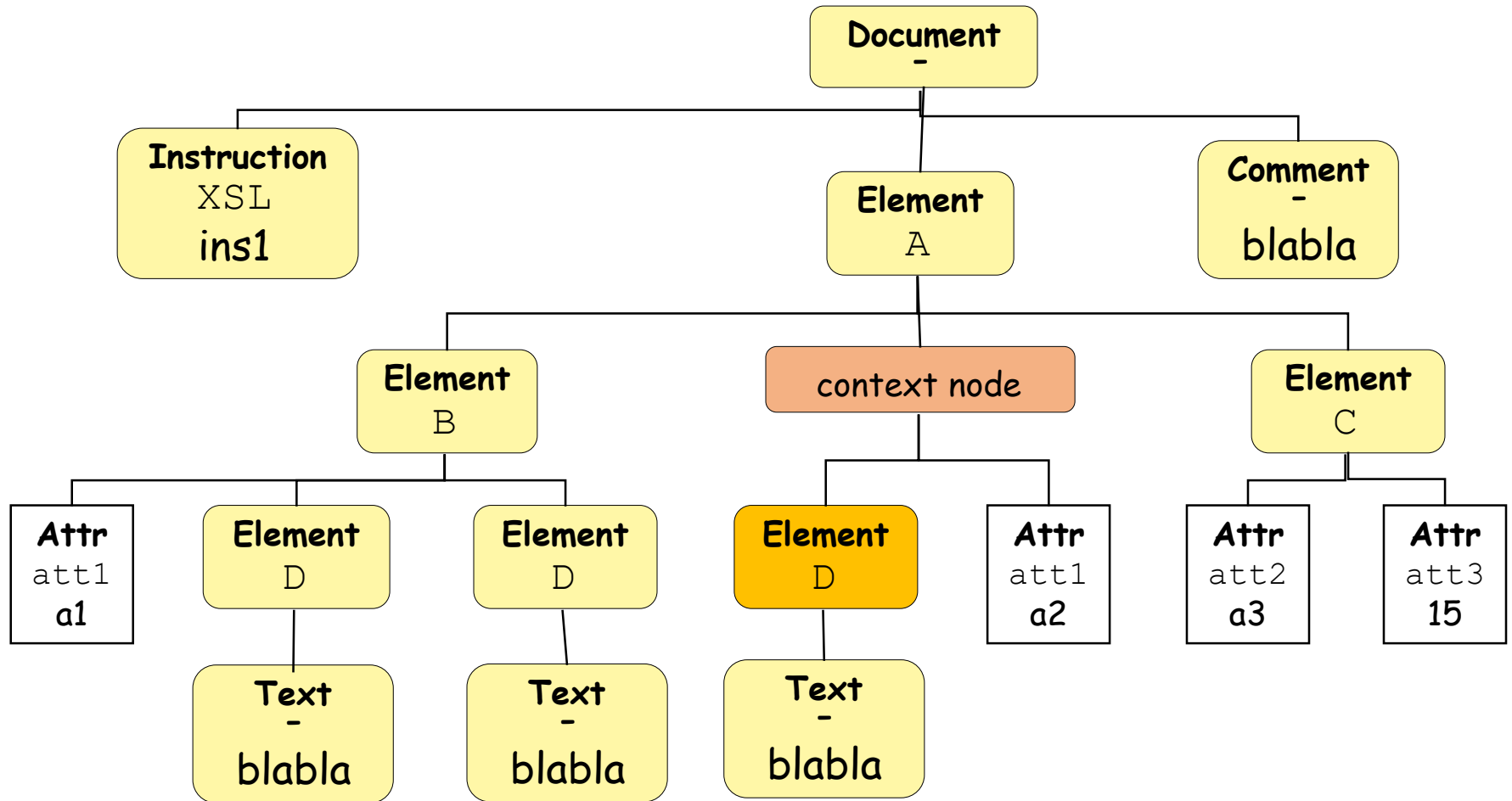
XPath axes

- **self::** (Abbreviated notation: **.**)
- **child::**name (Abbreviated notation: **name**)
- **attribute::** (Abbreviated notation: **@**)
- **parent::** (Abbreviated notation: **..**)
- **descendant::** (Abbreviated notation: **//**)
- **descendant-or-self::**
- **ancestor::**
- **ancestor-or-self::**
- **following::**
- **following-sibling::**
- **preceding::**
- **preceding-sibling::**

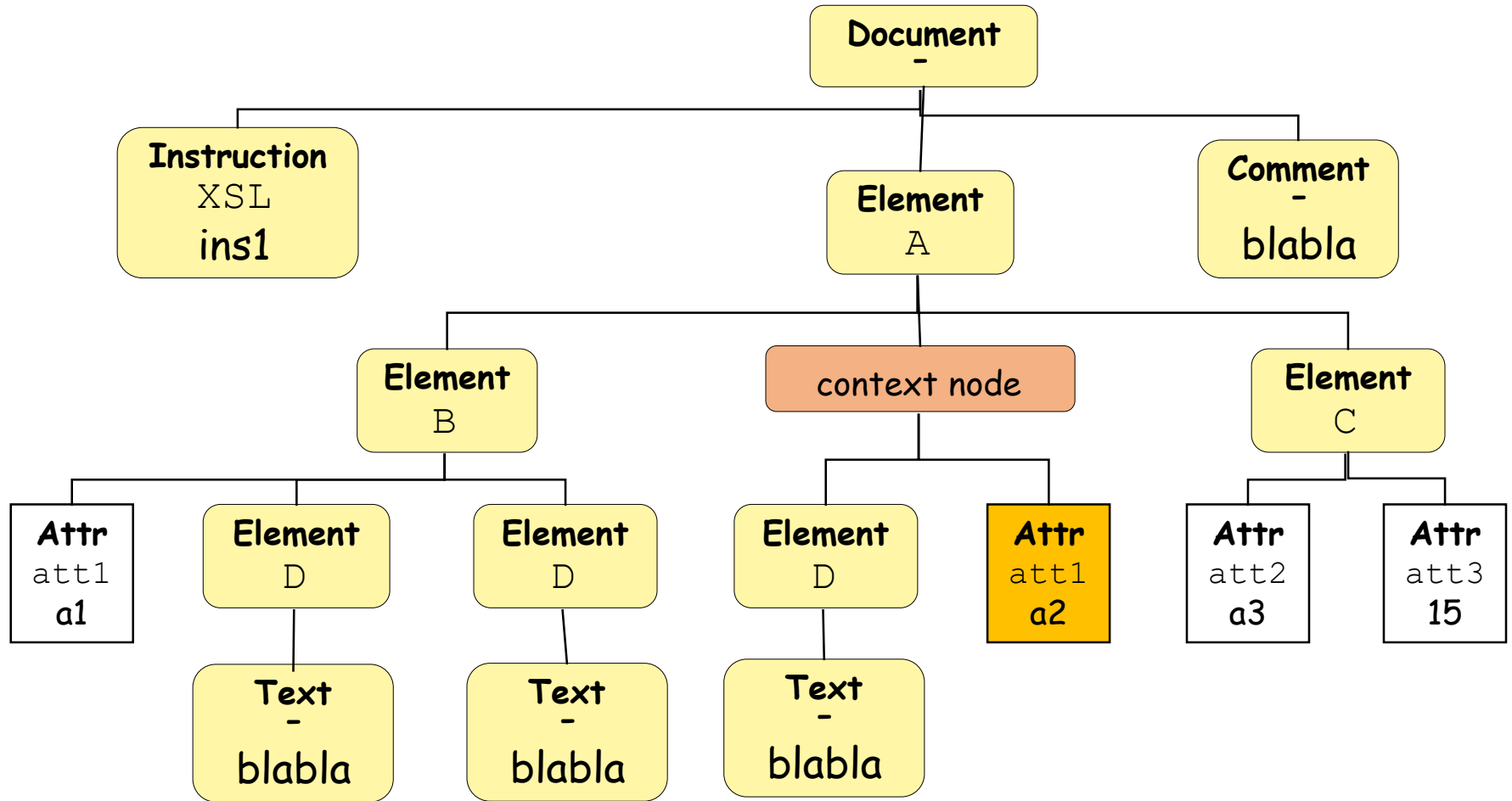
Axis self::node()



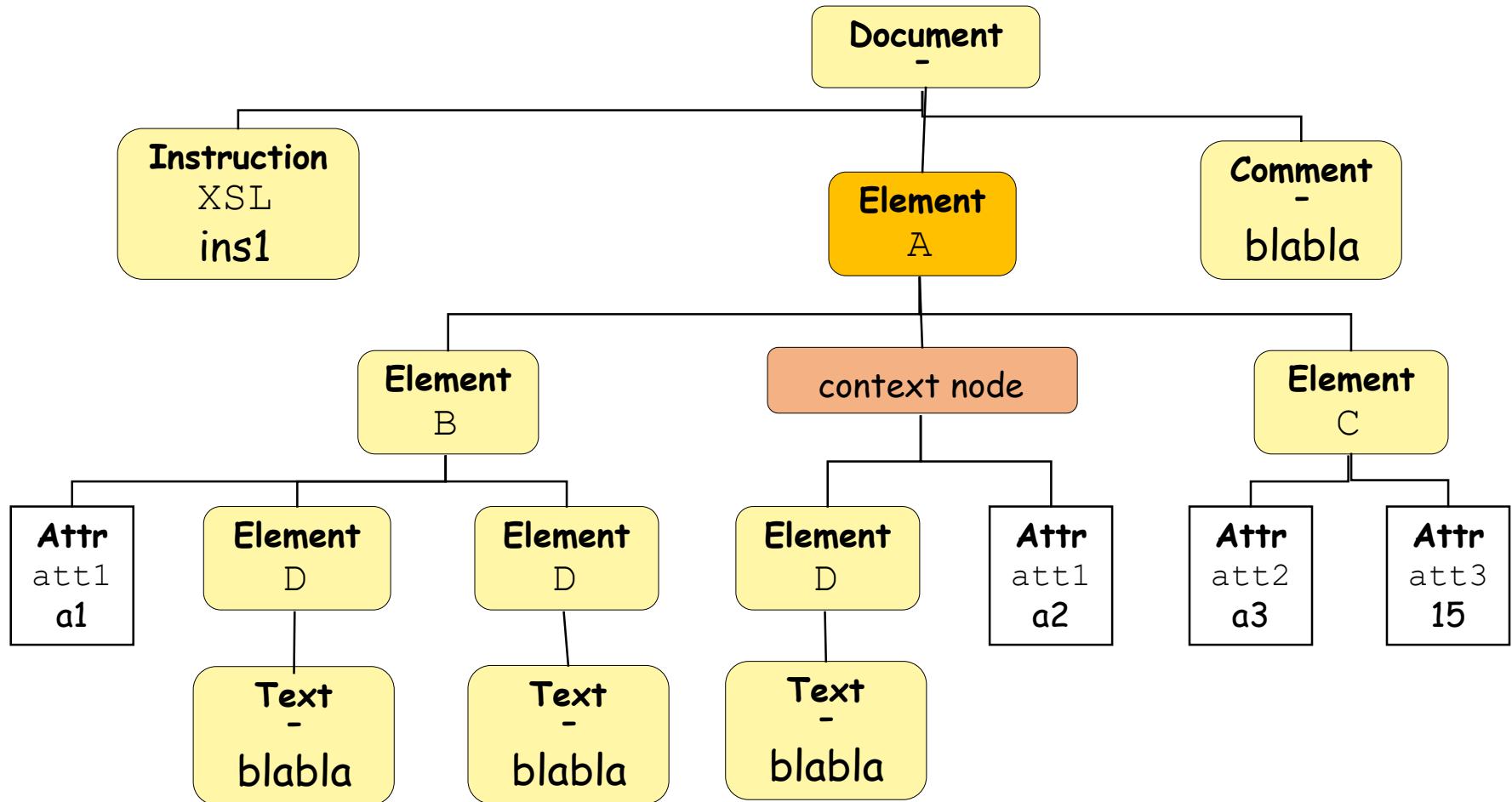
Axis child::node()



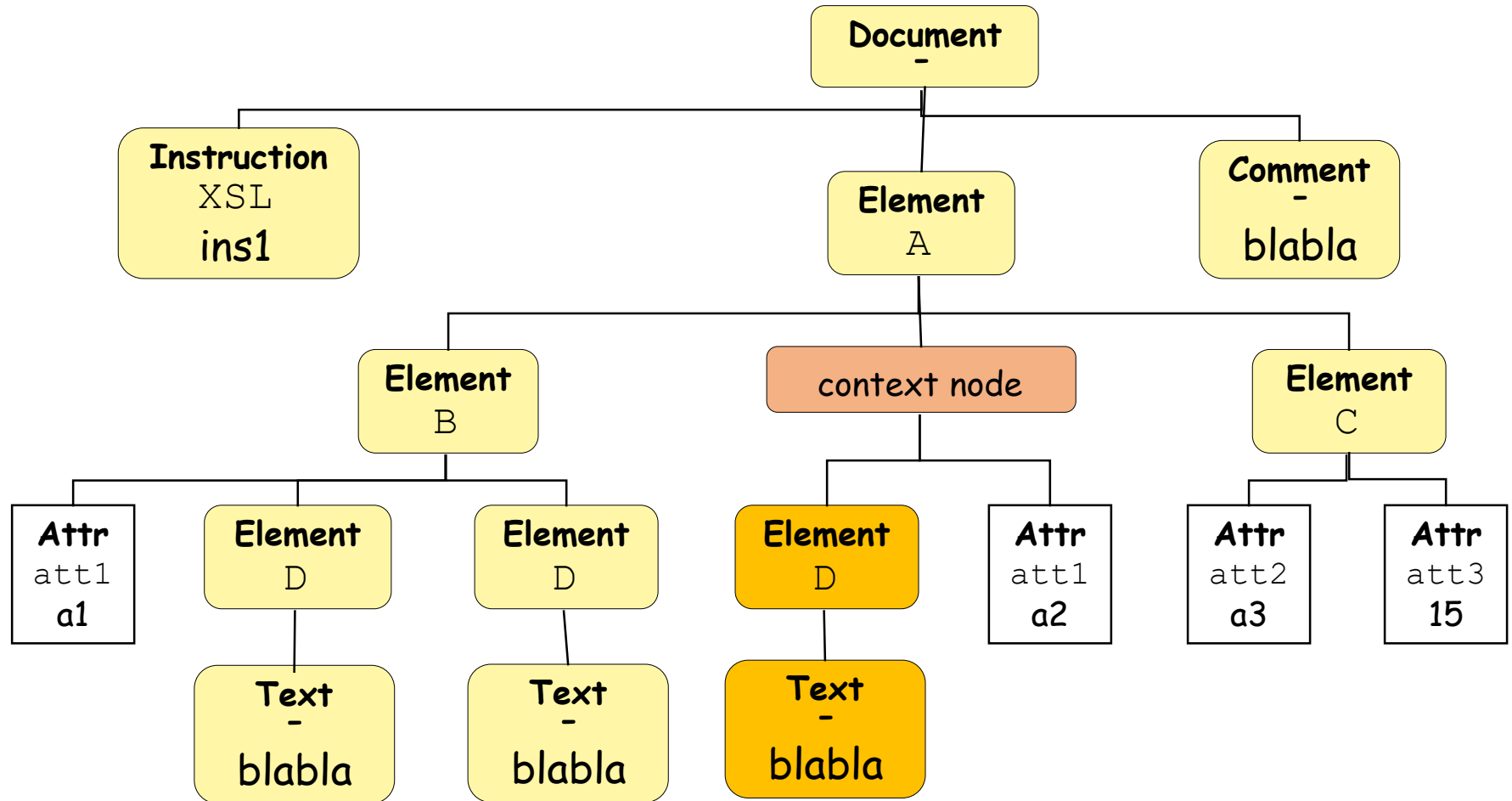
Axis attribute::att1



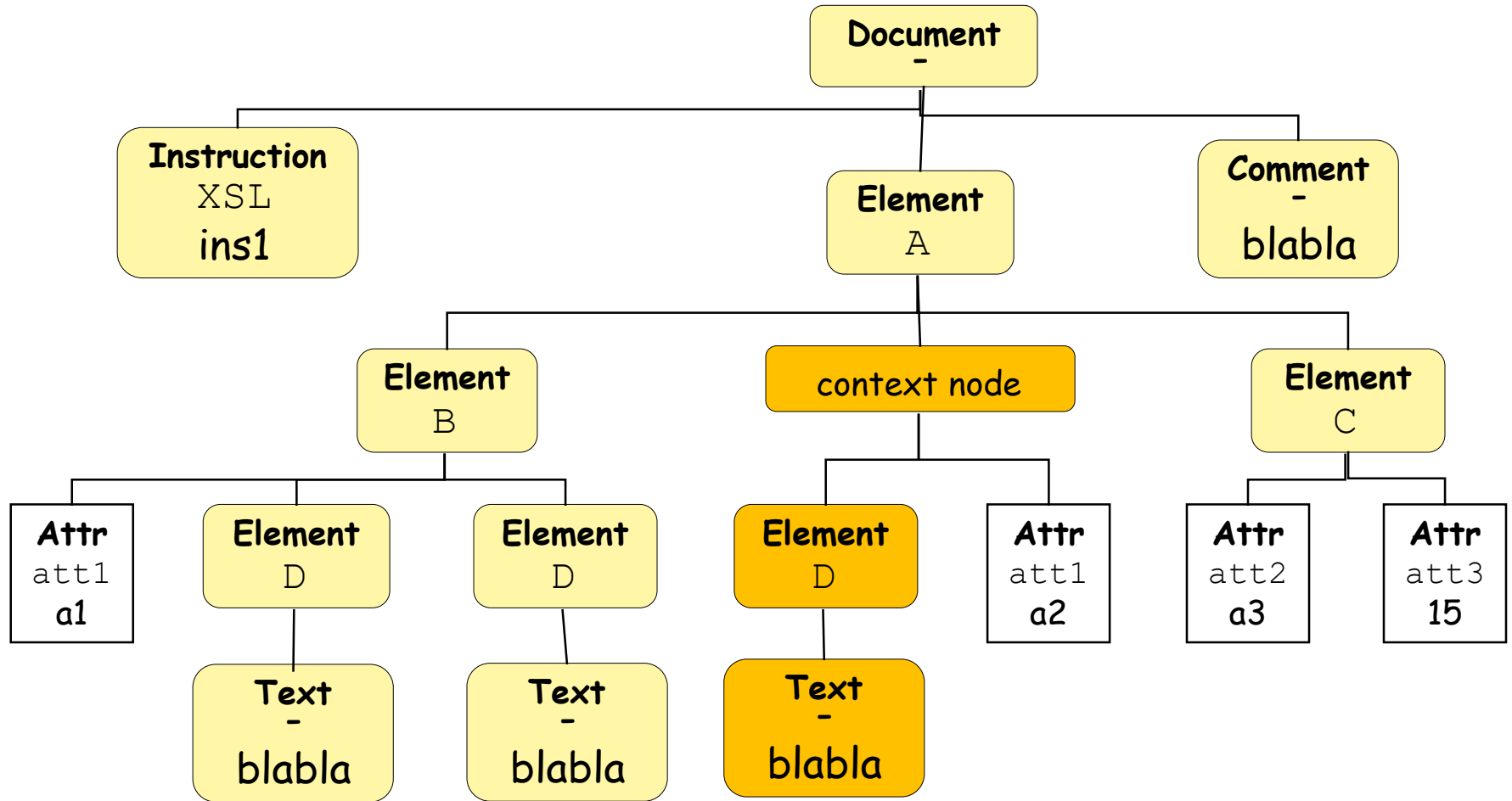
Axis parent::A



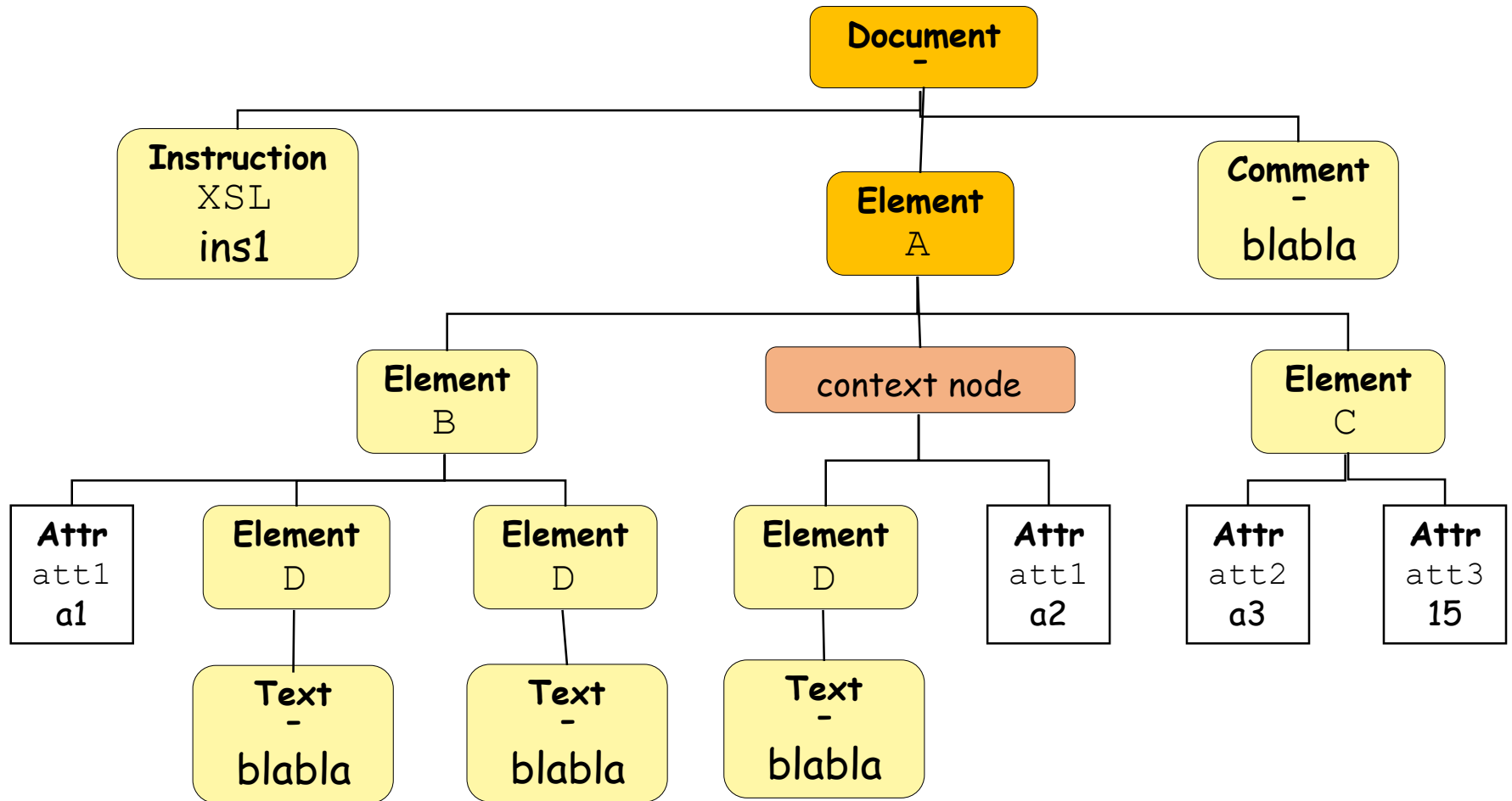
Axis descendant::node()



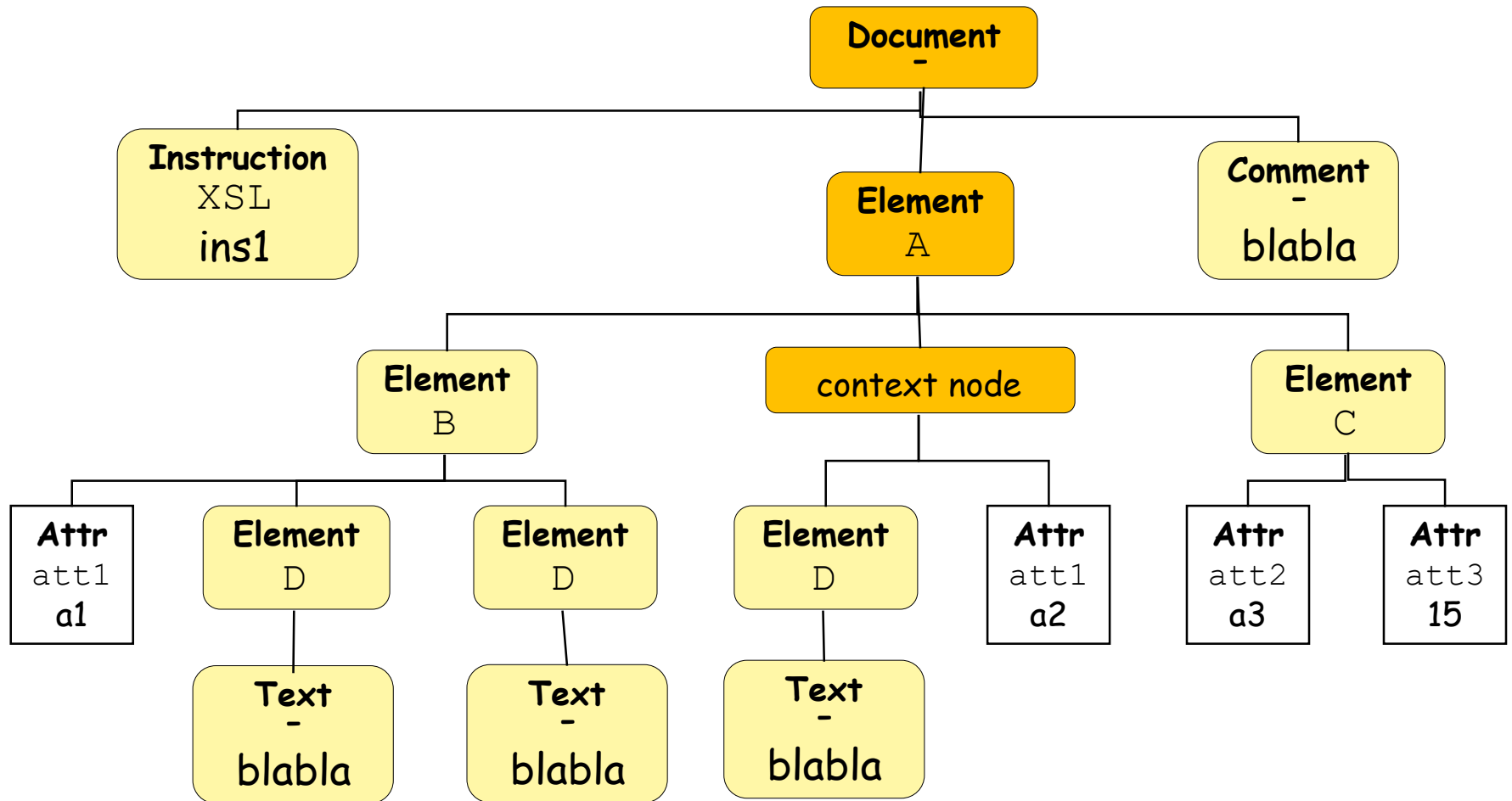
Axis descendant-or-self::node()



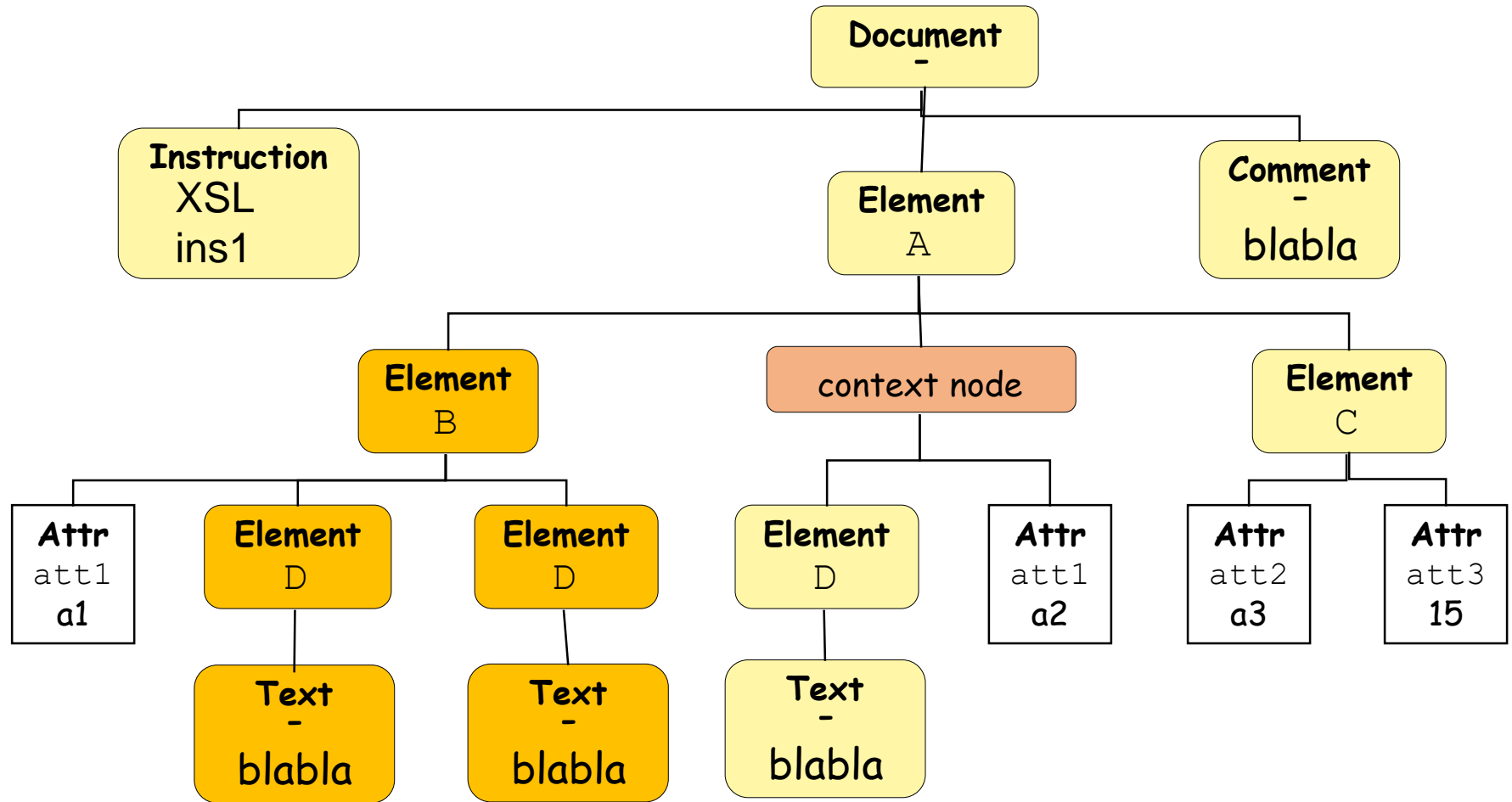
Axis ancestor::node()



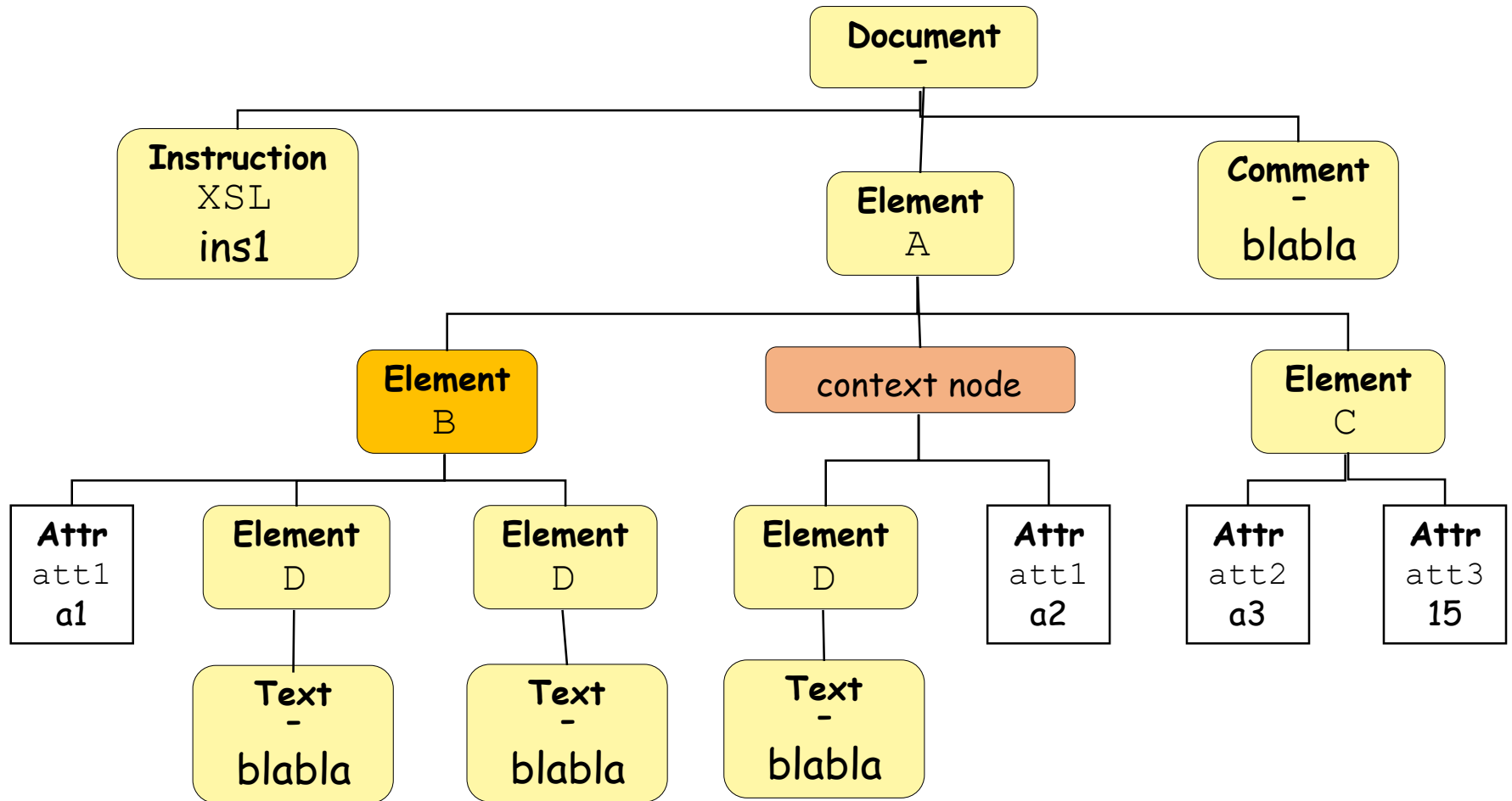
Axis ancestor-or-self::node()



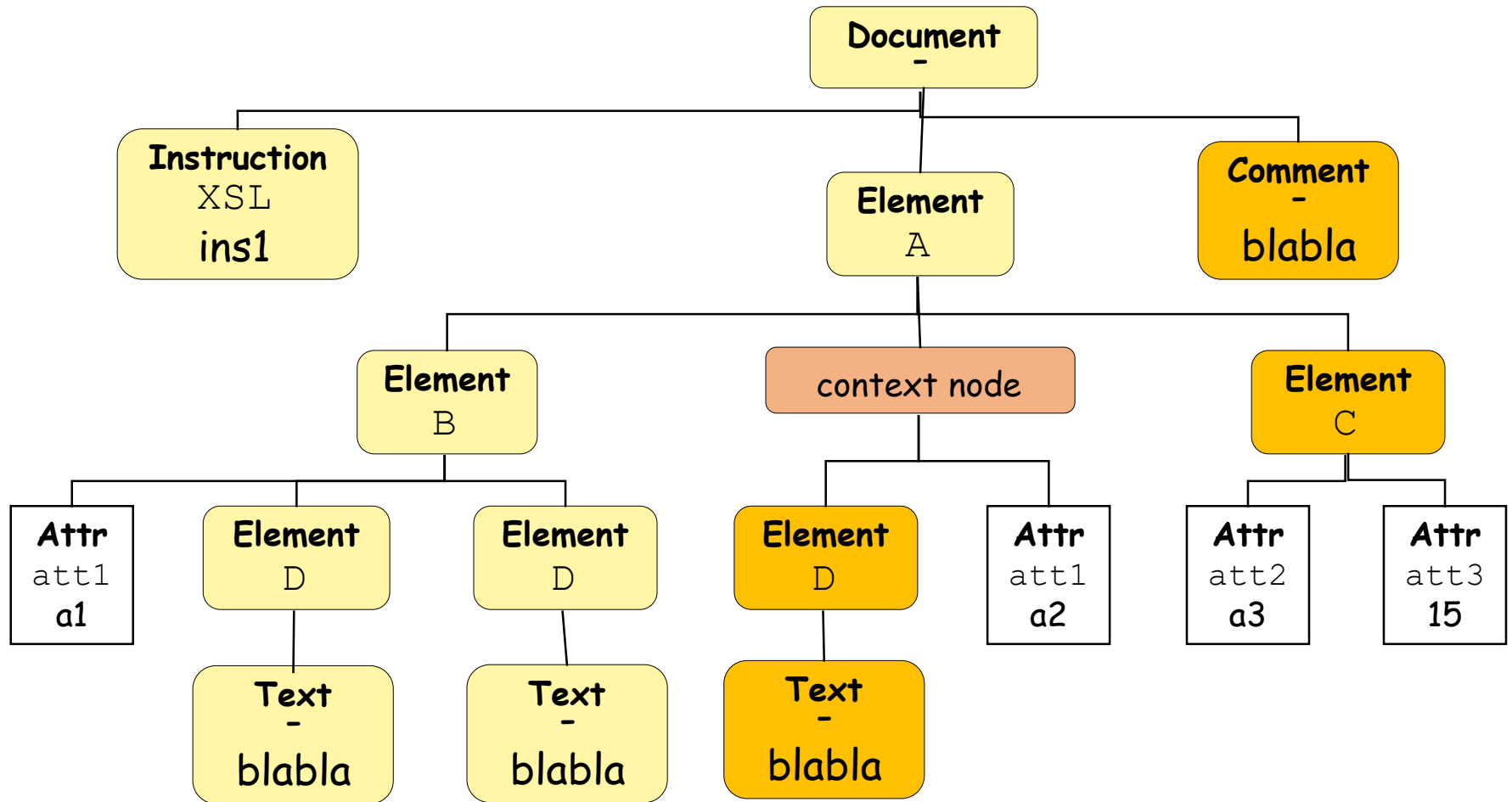
Axis preceding::node()



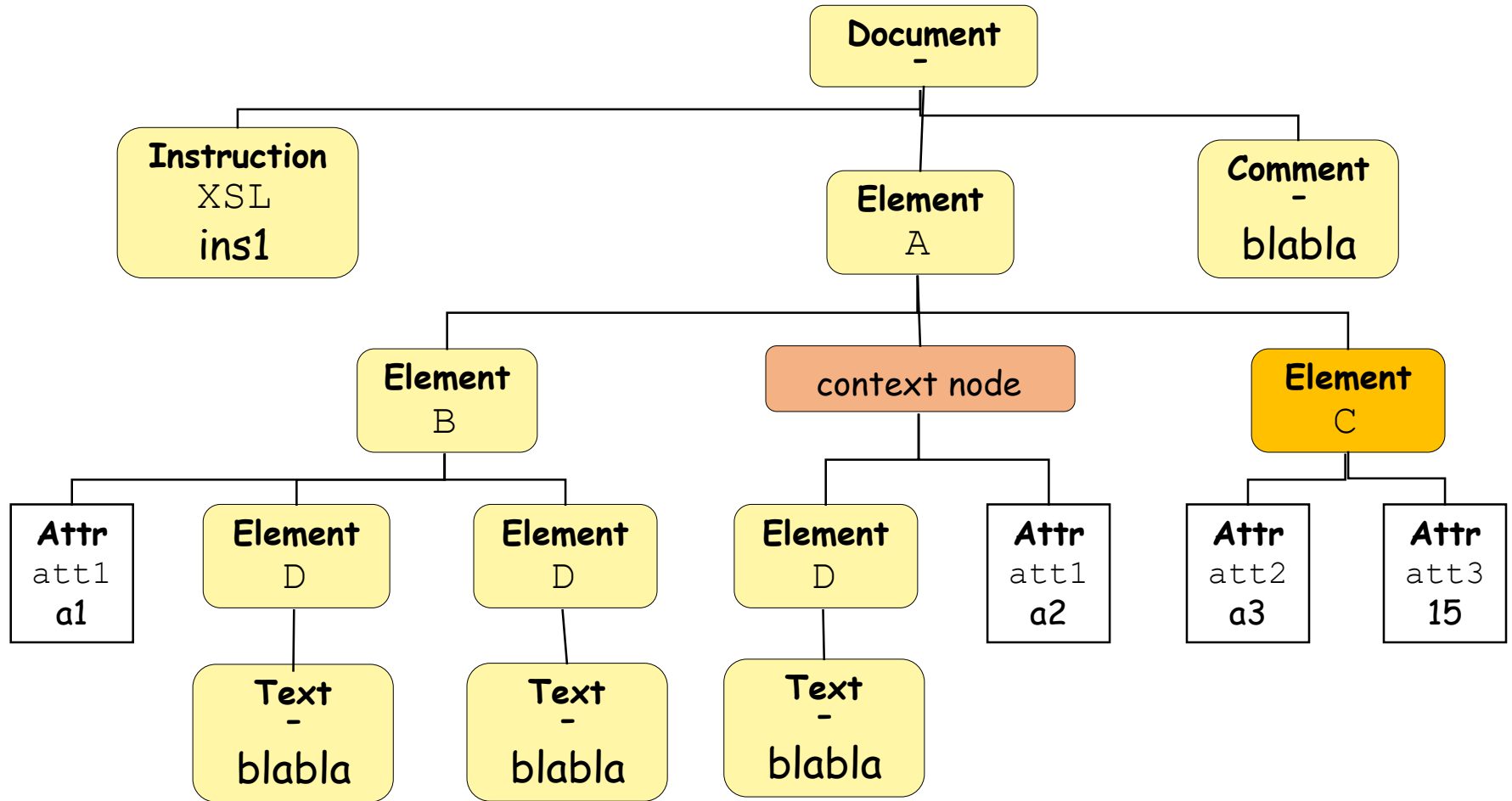
Axis preceding-sibling::node()



Axis following::node()



Axis following-sibling::node()



XPath

Principles

- The axes

- The filters

- The predicates

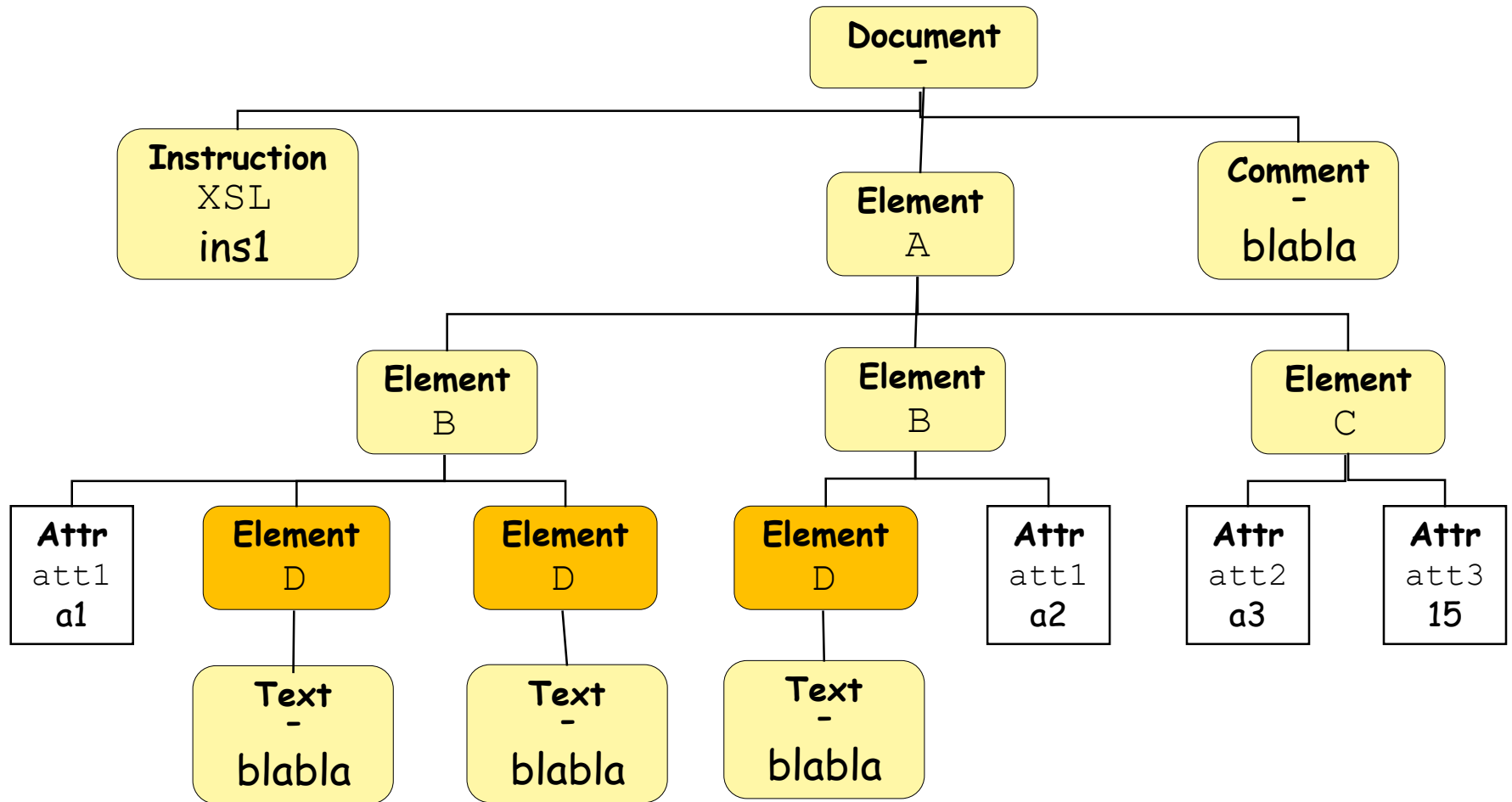
- Basic functions

- Examples

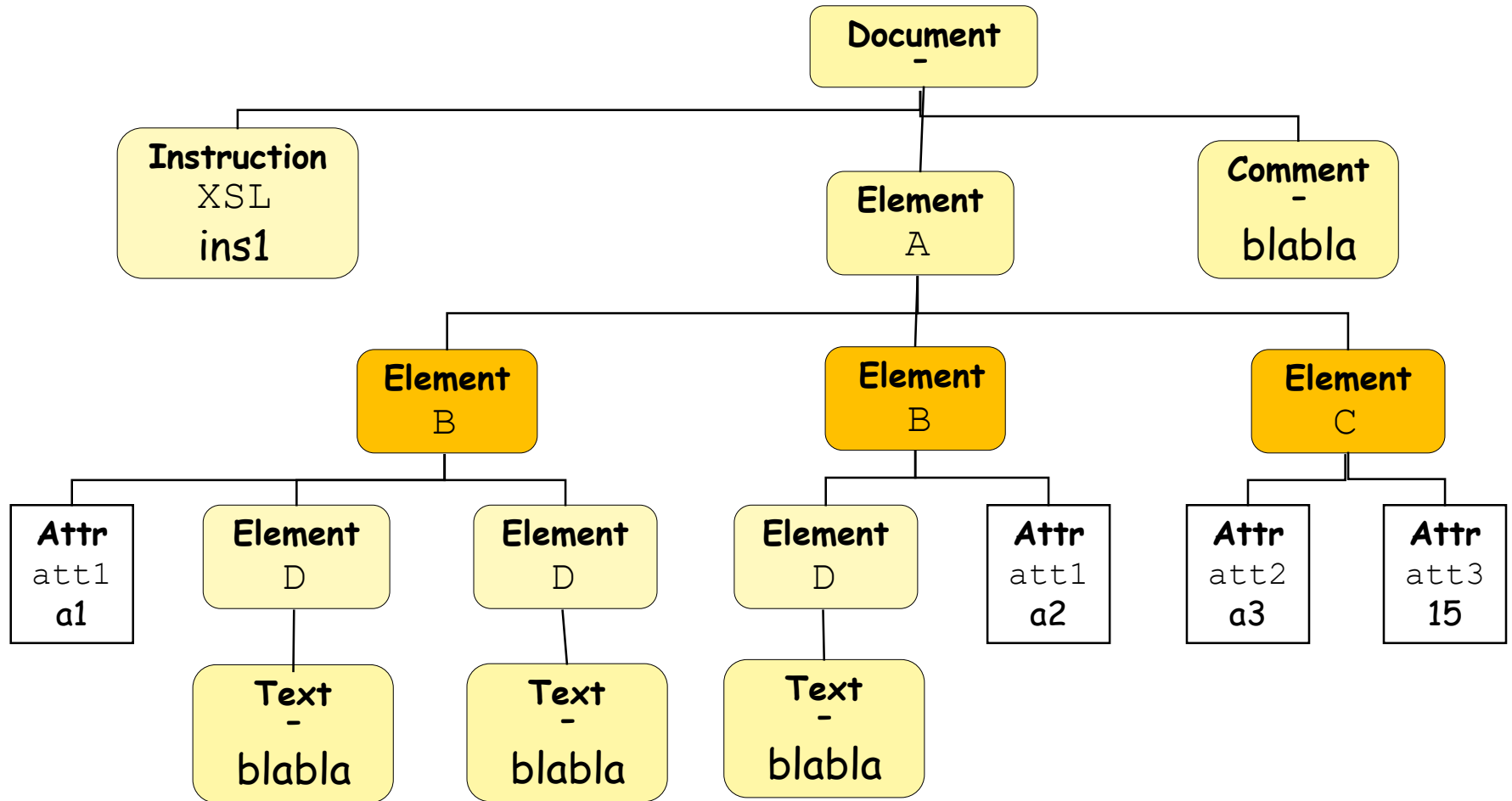
Two types of filters: the name or type

- Filtering the name (valid for the types of nodes that have a name):
 - `Element`
 - `ProcessingInstruction`
 - `attributes`
- **name**: nodes name
 - example: `/book/authors/author`
- *****: Any node of the axis
 - example: `/book/*`

/A/B/D



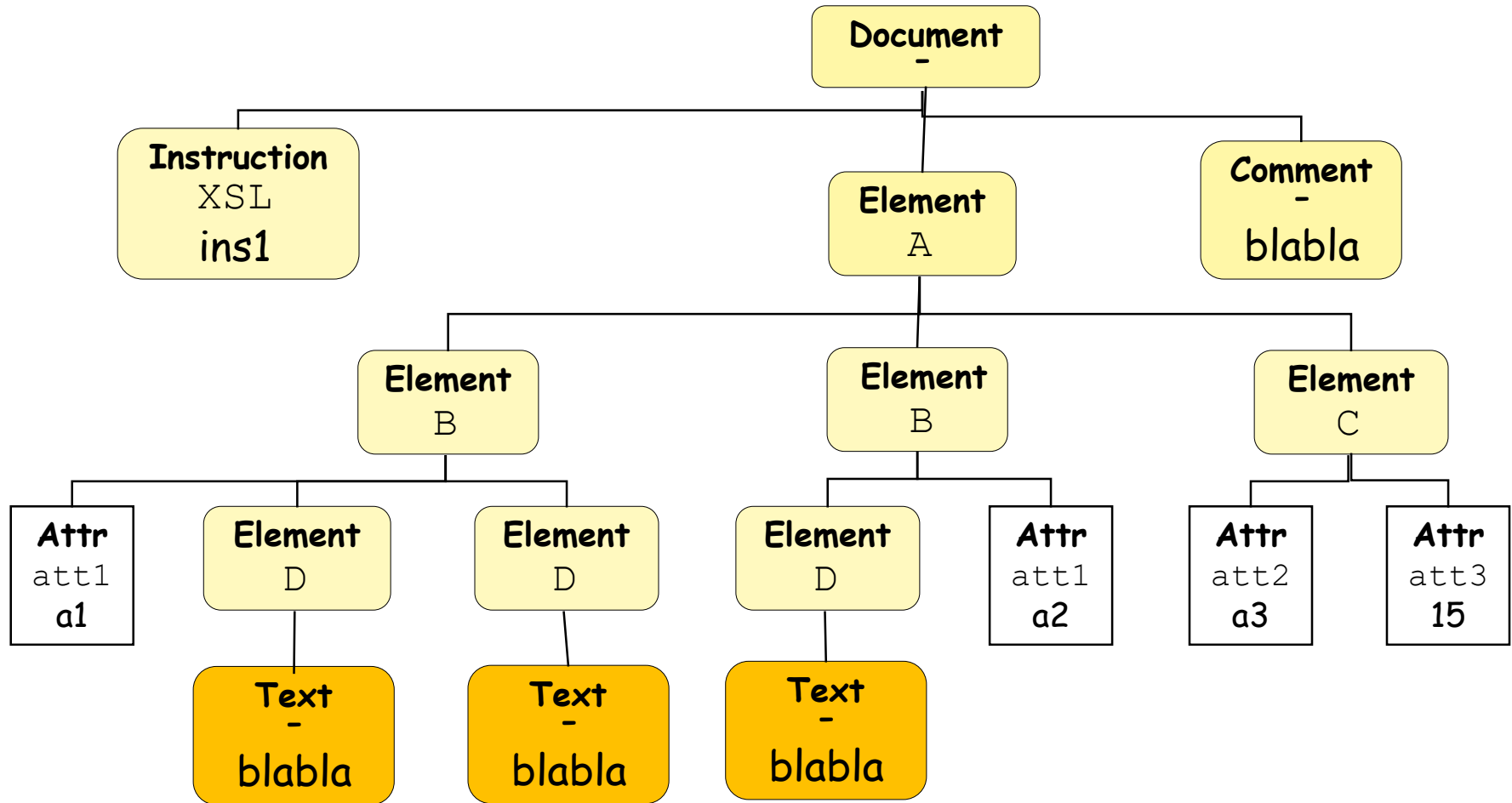
Generic name: $/A/*$



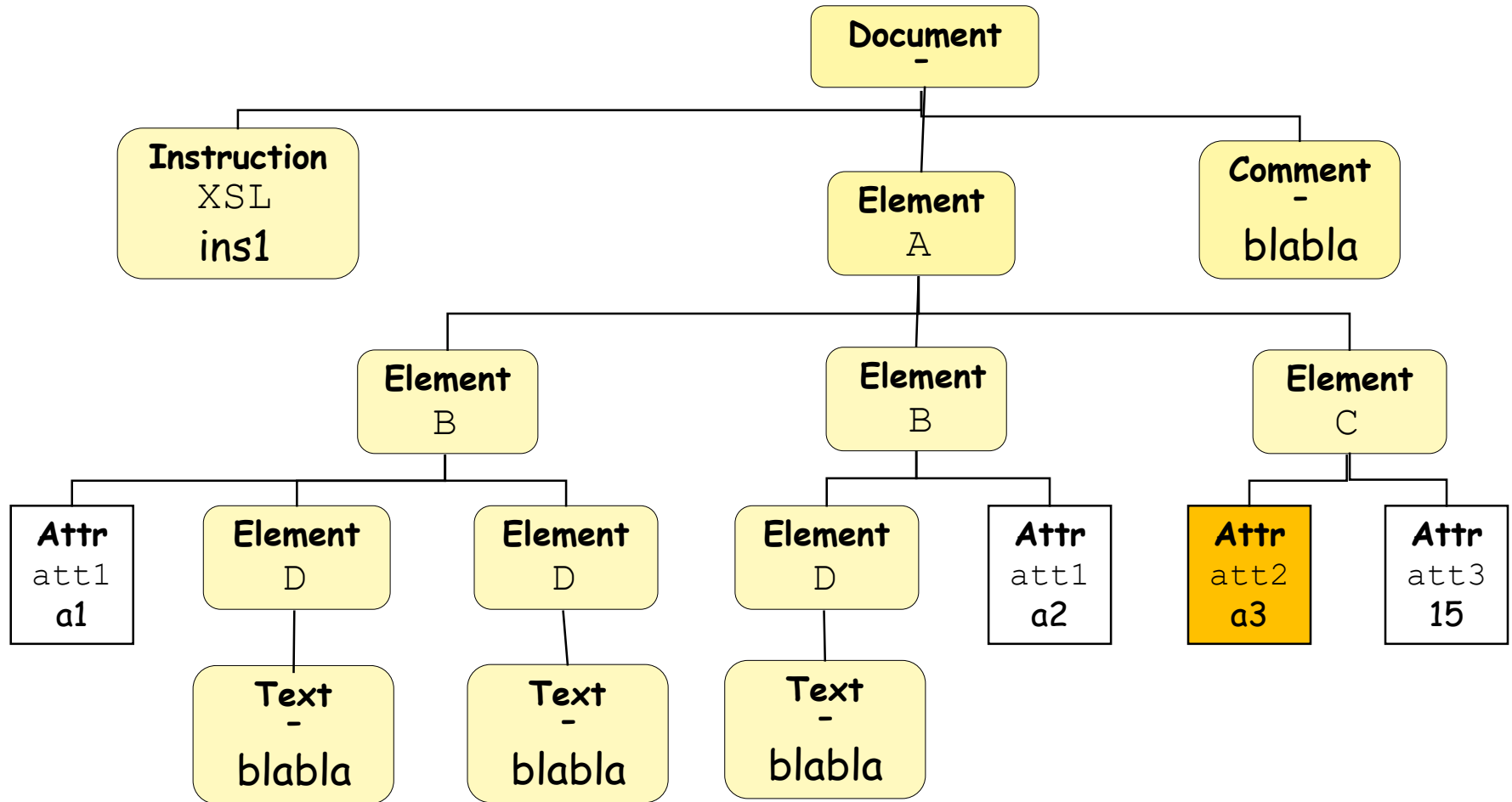
Filtering on type

- **text()**:
 - node type **text**
 - example: `/book/title//text()`
- **comment()**:
 - node type **Comment**
 - example: `/comment()`
- **processing-instruction()**:
 - node type **ProcessingInstruction**
 - example: `/processing-instruction('XSL')`
- **node()**:
 - all types of nodes

/A/B//text()



/descendant::node()/@att2



XPath

Principles

- The axes

- The filters

- The predicates

- Basic functions

- Examples

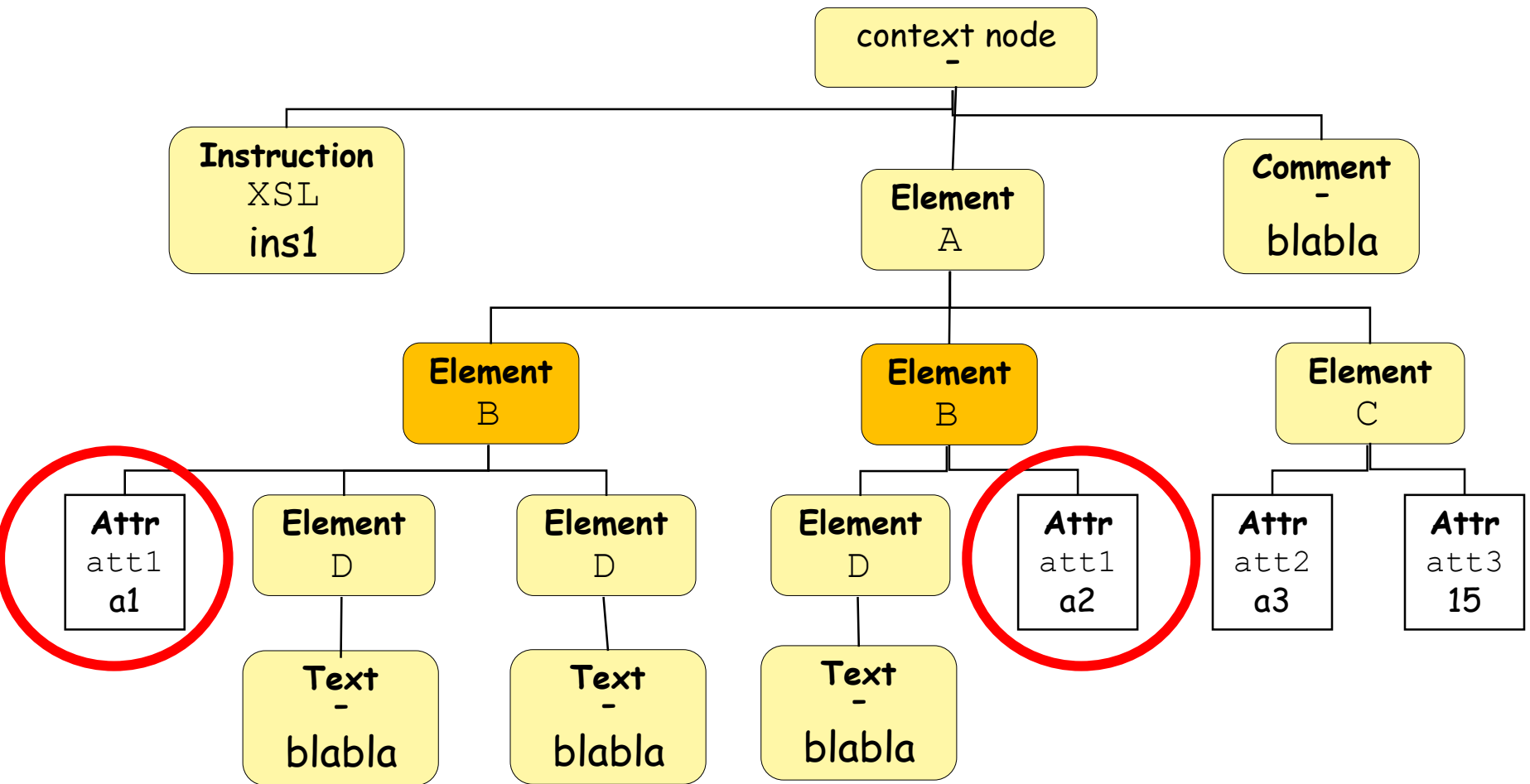
Predicates and paths

- Boolean expressions built from
 - path expressions
 - predefined functions.
- Calculated expression automatically converted into boolean result according to its type and its value.
 - Numerical value
 - 0 or NAN: predicate = *false*.
 - Other numerical value predicate = *true*.
 - Character String,
 - Empty: predicate = *false*
 - Not empty: predicate = *true*.
 - Node set,
 - Empty: predicate = *false*
 - Not empty: predicate = *true*.

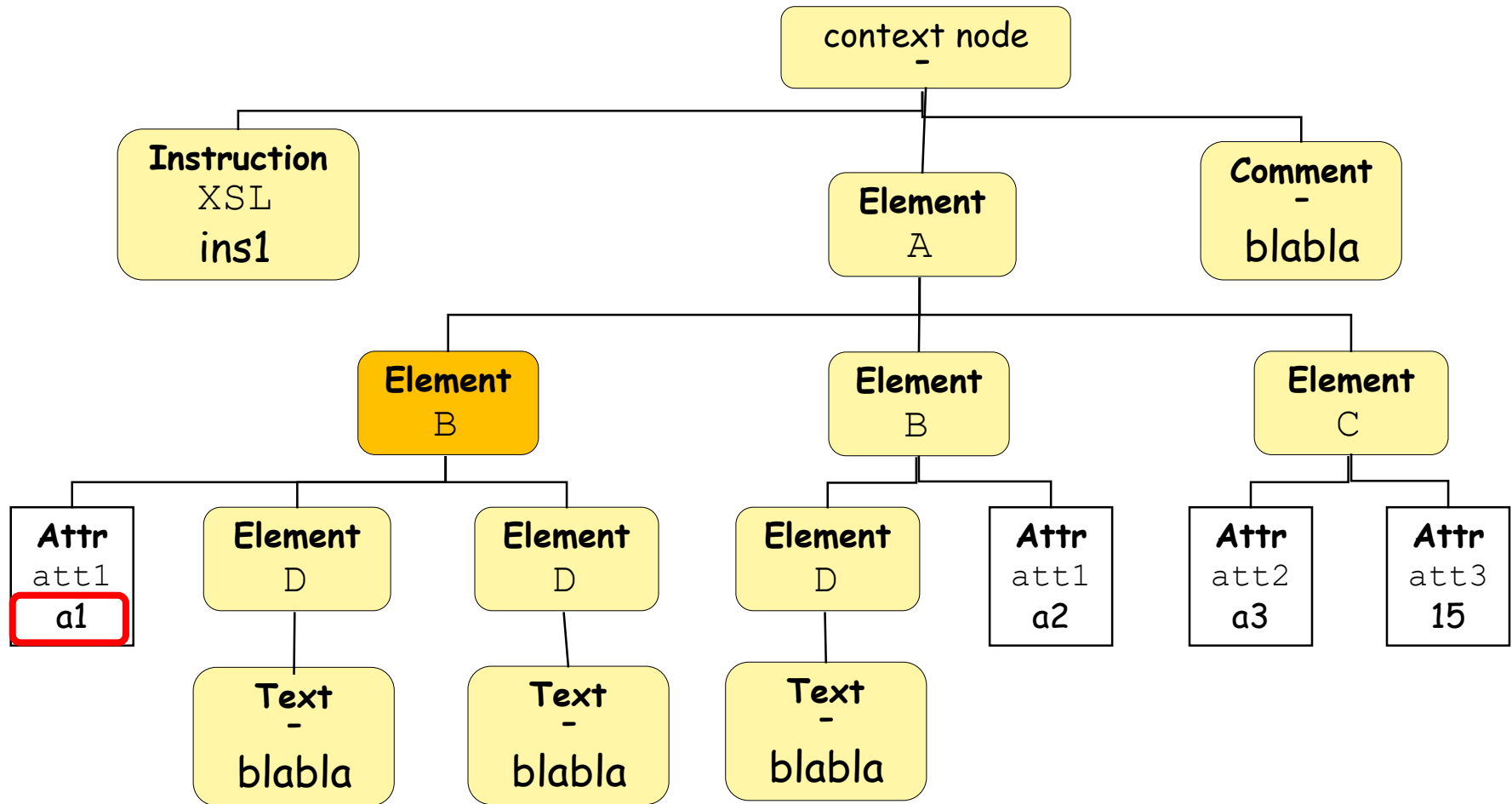
Examples

- Examples of predicates paths expressions
 - `/child::book/child::title[attribute::genre]`
 - FA: `/book/title[@genre]`
 - element *title* child of an element *book* possessing an attribute *genre*
 - `/child::book/child::title[attribute::type = 'SF']`
 - FA: `/book/title[@type = 'SF']`
 - element *title*, child of an element *book* possessing an attribute *type* whose value is 'SF'.
- In the expression `/A/B[@att1]`:
 - It focuses on the type of nodes **B** children of the root element **A**
 - Among these nodes the result contains only those for which the predicate `[@att1]` evaluates to **true**
 - `[@att1]` returns true only if the element to which it is applied (here **B**) has an attribute named **att1**.

/A/B[@att1]



/A/B[@att1='a1']



Predicates and functions

- The predicates can also use predefined **Xpath** functions to check the conditions on the nodes.
- Examples of predicates using functions
 - `/child::authors/descendant::text()[Position()= 1]`
 - FA: `/authors//text()[1]`
 - First *text* type node descending of an element *authors*
 - abbreviated syntax: `[Position()=1]` writes `[1]`.
 - `/child::title[Count(descendant::author)≥2]`
 - FA: `/title[Count(//author)≥2]`
 - *title* element which has at least two *author* type descendants

XPath

Principles

- The axes

- The filters

- The predicates

Basic functions

Examples

Functions applied to the nodes

- **count(*node_set*)**
 - Counts the number of referenced nodes.
- **position()**
 - Returns the position of the context node.
- **last()**
 - Returns the position of the last node of a set of nodes.
- **name(*node*)**
 - Returns the name of the context node.
- **string(*node*)**
 - Returns the text of the *node* without markup

String Functions (1/2)

- **concat(*string1*, *string2*, *chain*, ...)**
 - concatenates all strings passed as arguments
- Boolean functions to determine if a string is included in another.
 - **starts-with(*string1*, *string2*)**
 - **true** if *string1* starts with *string2*
 - **contains(*string1*, *string2*)**
 - **true** if *string1* contains *string2*.
- **string-length(*chain*)**
 - Returns the length of *chain*.
 - Used without argument, the function returns the length of the text value of the current node.

String Functions (2/2)

- Extracting a substring from a string
 - **substring(*string1*, *m*, *n*)**
 - Substring *string1* containing *n* characters beginning with the character at position *m*.
 - **substring-after(*string1*, *string2*)**
 - Substring of *string1* following the first occurrence of *string2*.
 - **substring-before(*string1*, *string2*)**
 - Substring of *string1* that precedes the first occurrence of *string2*.
- **translate(*string1*, *string2*, *string3*)**
 - replaces in *string1* each occurrence of any character of *string2* by the corresponding character of *string3*.
 - ex: **translate** (*chain*, "abcdefghijklmnopqrstuvwxyz" "ABCDEFGHIJKLMNOPQRSTUVWXYZ")
 - conversion of *chain* in capital letters.

Boolean functions

- `true()`, `false()` and `not ()`.
- `lang(chain)`
 - *true* if the value of the attribute `xml:lang` = the current node that received argument.
- `boolean(object)`
 - Conversion to a Boolean value by type:
 - digits
 - 0 or NaN = `false`
 - everything else is `true`
 - strings
 - empty string = `false`
 - everything else is `true`
 - node sets
 - empty set = `false`
 - everything else is `true`

Numeric operations

- basic numeric operators:
 - comparisons: `<`, `>`, `=`, `!=`
 - operations: `+`, `-`, `*`, `div`, `mod`
- Numeric functions:
 - **`number(value)`**
 - Tries to convert value into a number (in the case of failure we obtain NaN)
 - **`ceiling(number)`**
 - Returns the smallest integer greater than or equal to *number*.
 - **`floor(number)`**
 - Largest integer equal to or less *number*
 - **`round(number)`**
 - rounds *number* to the nearest integer
 - **`sum(number1, number2, ...)`**
 - Sums numbers / node values passed as arguments.

To sum up

- XPath is a powerful and complete standard filtering mechanism
- The location of a component can be expressed as
 - Absolute path (exact knowledge of the structure)
 - path model to follow (partial knowledge of the structure)
- key mechanism for a query language, handling
 - Way to filter the components of the document regardless of their type
 - Provides a set of operators to exploit these data

XPath

Principles

- The axes

- The filters

- The predicates

Basic functions

Examples

Examples of XPath expressions (1)

- **child::book[position()= last() - 1]**
 - FA: **book[last() - 1]**
 - second last *book* the context node
- **following-sibling::book[position() = 1]**
 - FA: **following-sibling::book[1]**
 - first brother right type *book* the context node
- **descendant::book[position() = 12]**
 - FA: **//book[12]**
 - twelfth *book* of the document

Examples of XPath expressions (2)

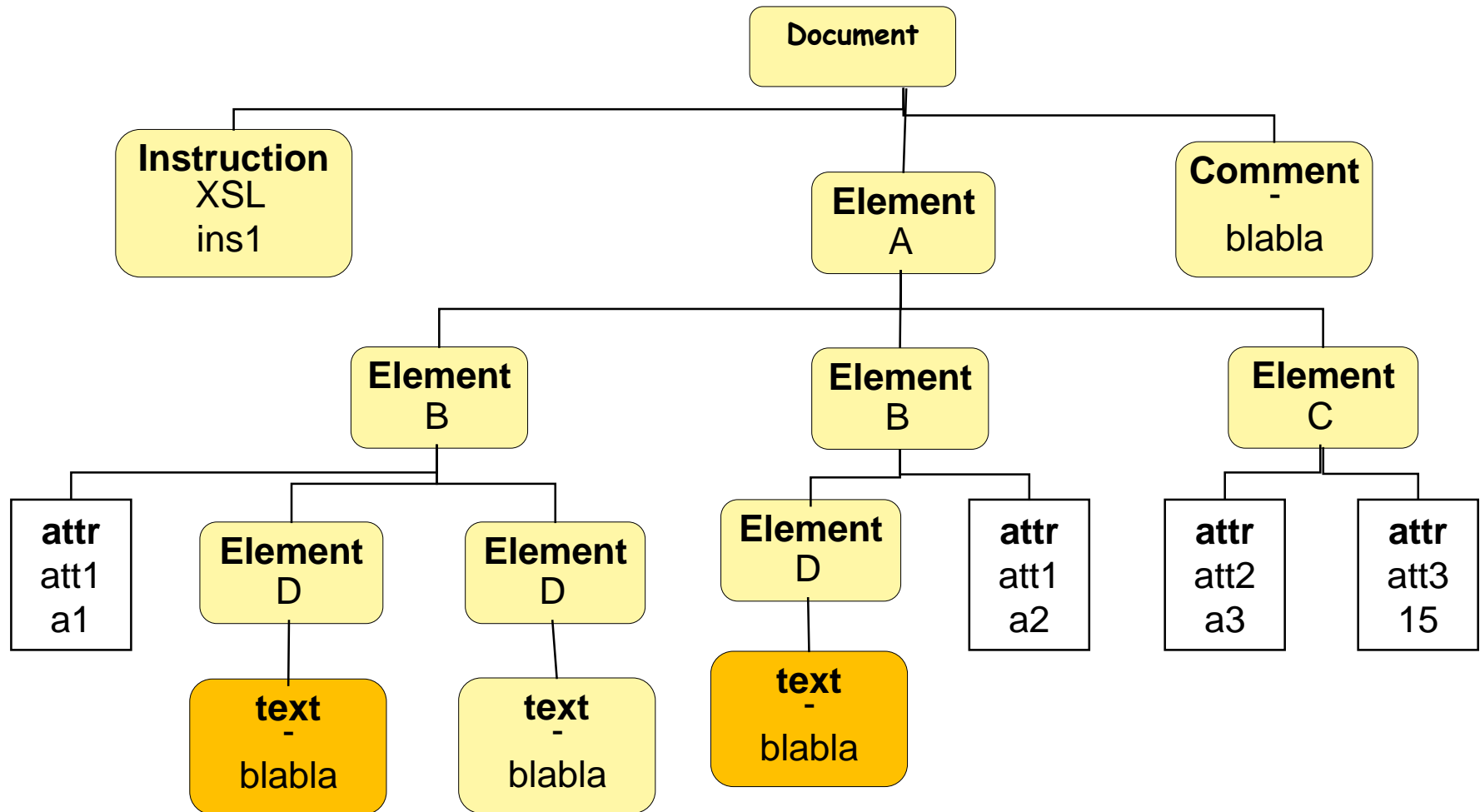
- **child::book[publisher and descendant::titre[attribute::genre='SF']]**
 - FA: **book[publisher and //titre[@genre='SF']]**
 - all *book* with an editor and whose genre is Science Fiction.
- **descendant::name[child::text() = 'Asimov']/ancestor::title**
 - FA: **//name[text()='Asimov']/../title**
 - all titles of the books if the name of an author is Asimov.

Sample (3)

`/A/B/descendant::text()[1]`

FA: `/A/B//text()[1]`

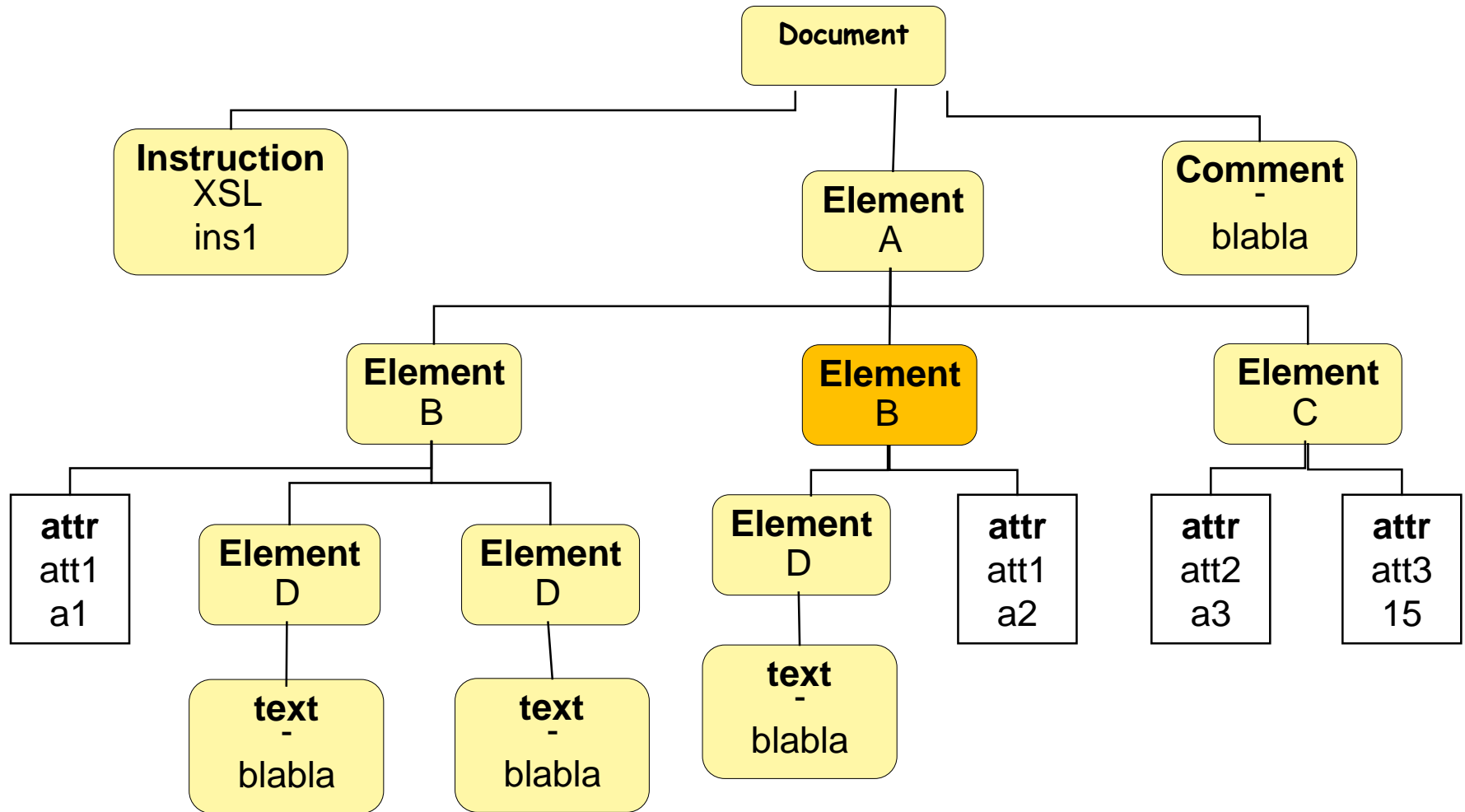
The first text node type descendant of a `/A/B`.



Sample (4)

/A/B[count(D) = 1]

Nodes / A / B having only one child D



forms shortcuts

long form	short form (FA)
child::	
attribute::	@
self::node ()	.
parent::node ()	..
descendant::node ()	//
[position() = number]	[number]

Examples of shortcuts forms

long form	short form
<code>/child::catalog/child::cd</code>	<code>/Catalog/cd</code>
<code>/child::catalog/attribute:: Country</code>	<code>/Catalog/Country/@</code>
<code>/self::node()/descendant-or-self:: node()/child::title</code>	<code>././title</code>
<code>/descendant-or-self::node/ scratch/parent::node ()</code>	<code>//scratch/..</code>

Plan

Introduction

Core XML

XML galaxy

XPath

XSL

Handling

applications

NOSQL

Conclusion

Style sheets and treatments

Document transformation

XSLT Processing Model

XSLT document elements

Conclusion

Read more ...

Document without stylesheet

```
<?xml version="1.0"?>
<library>
  <book>
    <title>N or M</title>
    <author>Agatha Christie</author>
    <ref>Police-C-15</ref>
  </book>
  <book>
    <title>The Hound of the
Baskervilles</title>
    <author>Sir Arthur Conan Doyle</author>
    <ref>Police-D-3</ref>
  </book>
  <book>
    <title>dune</title>
    <author>Franck Heckbert</author>
    <ref>Fiction-H-1</ref>
  </book>
</library>
```

visualization

- With a browser

```
<?xml version="1.0" ?>
- <bibliotheque>
+ <livre>
+ <livre>
- <livre>
    <titre>Dune</titre>
    <auteur>Franck Heckbert</auteur>
    <ref>Fiction-H-1</ref>
  </livre>
</bibliotheque>
```

```
<?xml version="1.0" ?>
- <bibliotheque>
- <livre>
    <titre>N ou M</titre>
    <auteur>Agatha Christie</auteur>
    <ref>Policier-C-15</ref>
  </livre>
- <livre>
    <titre>Le chien des Baskerville</titre>
    <auteur>Sir Arthur Conan Doyle</auteur>
    <ref>Policier-D-3</ref>
  </livre>
- <livre>
    <titre>Dune</titre>
    <auteur>Franck Heckbert</auteur>
    <ref>Fiction-H-1</ref>
  </livre>
</bibliotheque>
```

Document with CSS or XSL

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

```
<?xml-stylesheet type = "text/css" href = "biblio.css"?>
```

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

OR

```
<Library>
```

```
  <book>
```

```
    <title> N or M </title>
```

```
    <author> Agatha Christie </author>
```

```
    <Ref> Detective-C-15 </ref>
```

```
  </book>
```

```
  <book>
```

```
    <title> The Hound of the Baskervilles </title>
```

```
    <author> Sir Arthur Conan Doyle </author>
```

```
    <Ref> Detective-D-3 </ref>
```

```
  </book>
```

```
  <book>
```

```
    <title> Dune </title>
```

```
    <author> Franck Heckbert </author>
```

```
    <Ref> Fiction-H-1 </ref>
```

```
  </book>
```

```
</ Library>
```

CSS

```
book{
    display:block;
    margin-left:10pt;
    margin-bottom:5pt;
    font-size:12pt
}
title{
    margin-right:10pt;
    color:blue;
}
author{
    margin-right:10pt;
}
ref{
    color:red;
}
```

XSL stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
  <html>
    <head> <title>My library</title> </head>
    <body>
      <xsl:for-each select="library/book">
        <SPAN style="font-style: italic; padding-right: 3pt ">
          <xsl:value-of select="title"/> </SPAN>
        <SPAN style="color: red; padding-right: 3pt ">
          <xsl:value-of select="author"/> </SPAN>
        <SPAN style="color: blue">
          <xsl:value-of select="Ref"/> </SPAN>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

visualization

- CSS in Internet Explorer

Nou M	Agatha Christie	Policier-C-15
Le chien des Baskerville	Sir Arthur Conan Doyle	Policier-D-3
Dune	Franck Heckbert	Fiction-H-1

- XSL Mozilla

Bibliothèque

Nou M Agatha Christie Policier-C-15
Le chien des Baskerville Sir Arthur Conan Doyle Policier-D-3
Dune Franck Heckbert Fiction-H-1

Introduction to XSLT

- XSLT is a language for transforming XML documents into other XML documents
 - Developed by the W3C, initially for the XSL formatting language
 - W3C Recommendation since November 1999
 - XSLT 2.0 W3C recommendation since January 2007
 - Uses XML syntax, uses the expression language XPath
 - XSLT can be used in a processing chain that also uses, Javascript, ASP, PHP, etc.

The XSLT stylesheets

- XSL style sheets (eXtensible Stylesheet Language)
 - control the document layout
 - As CSS
 - Describe how an XML file should be transformed into another XML or HTML
- These are well-formed documents
 - XML declaration frontline
 - encoding declaration
 - Tag closure rules, etc.

Principle

- A source document (tree) is converted into a result document (tree) by a processing engine that applies a transformation style sheet
 - XML (+ possibly DTD) = D
 - XSL stylesheet = S
 - XSL processor = program that applies the stylesheet file = P

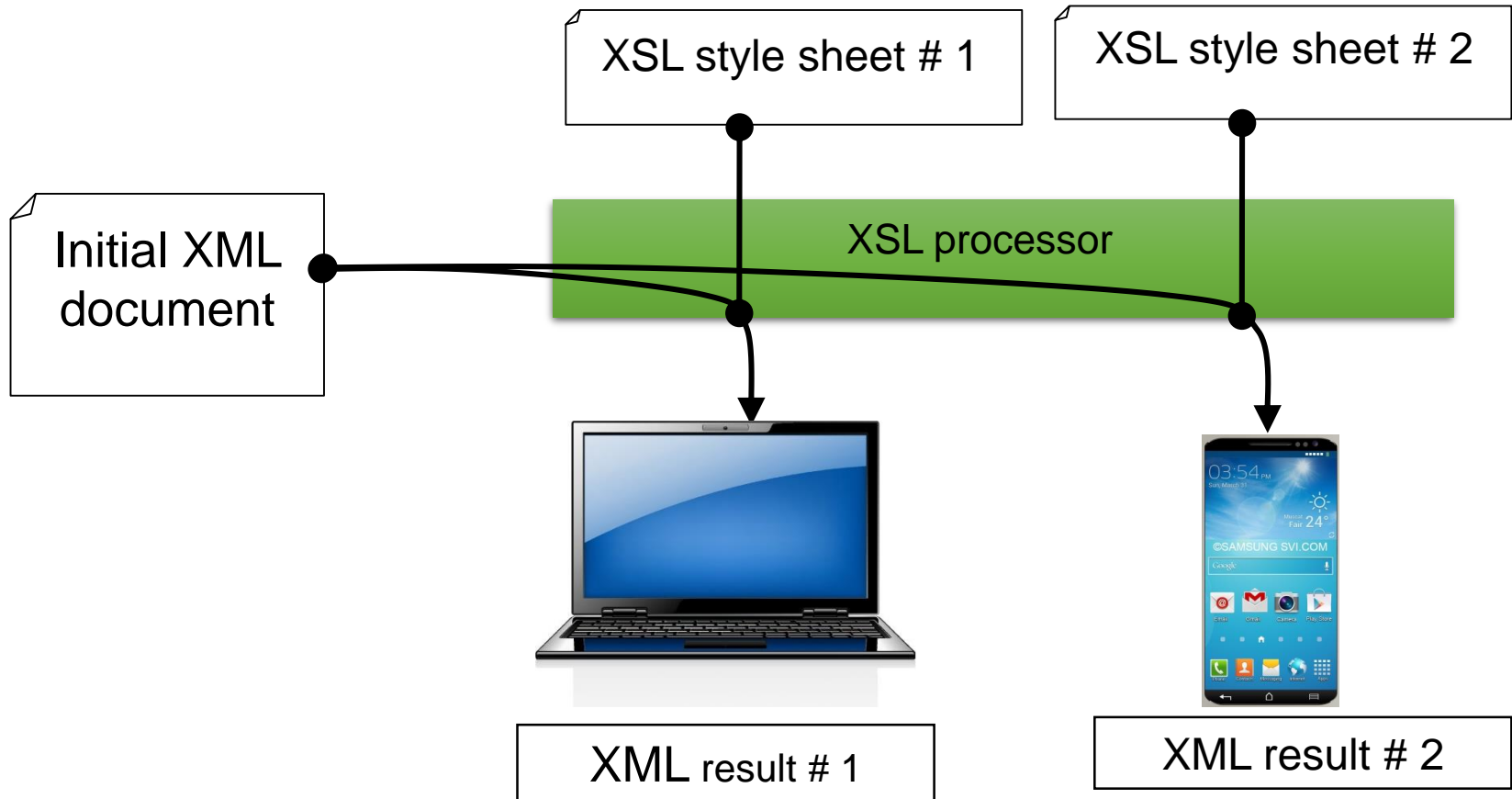


- The processor P transforms D into D' using (as defined by) F

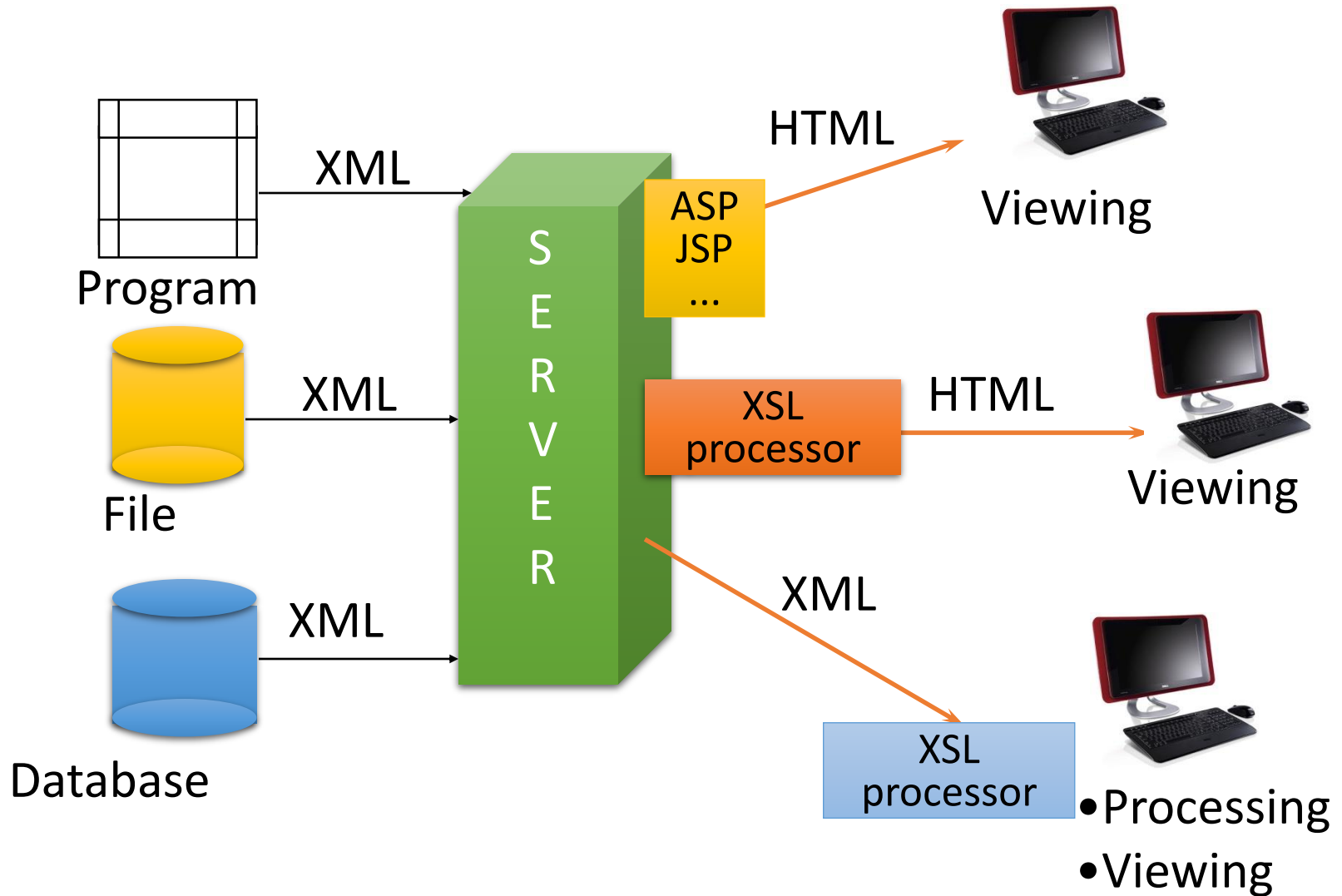
Advantages

- Several style sheets for a document
 - Display Features and content tailored to the needs
 - Content suitable for a user type or usage
 - Viewing adapted between platforms
 - Example (content adaptation): bibliographic records in HTML
 - detailed display
 - summary display
 - catalog of all cards
 - Example (style adaptation): online weather
 - 24 inches screen: weather maps, detailed information
 - Mobile: scaled down font size, Important information only, no maps

Role of XSL processor



architectures



Style sheets and treatments

Document transformation

XSLT Processing Model

XSLT document components

Conclusion

Read more ...

basic structure: Templates

- **template:** Basic structure to produce the result
 - A template applies in the context node of a tree
 - The application of the template produces a result fragment
- Global result (tree) built by a hierarchy of templates.
 - Elements of the XML source tree
 - unfiltered Elements
 - organized in a different order,
 - New elements created by the templates.

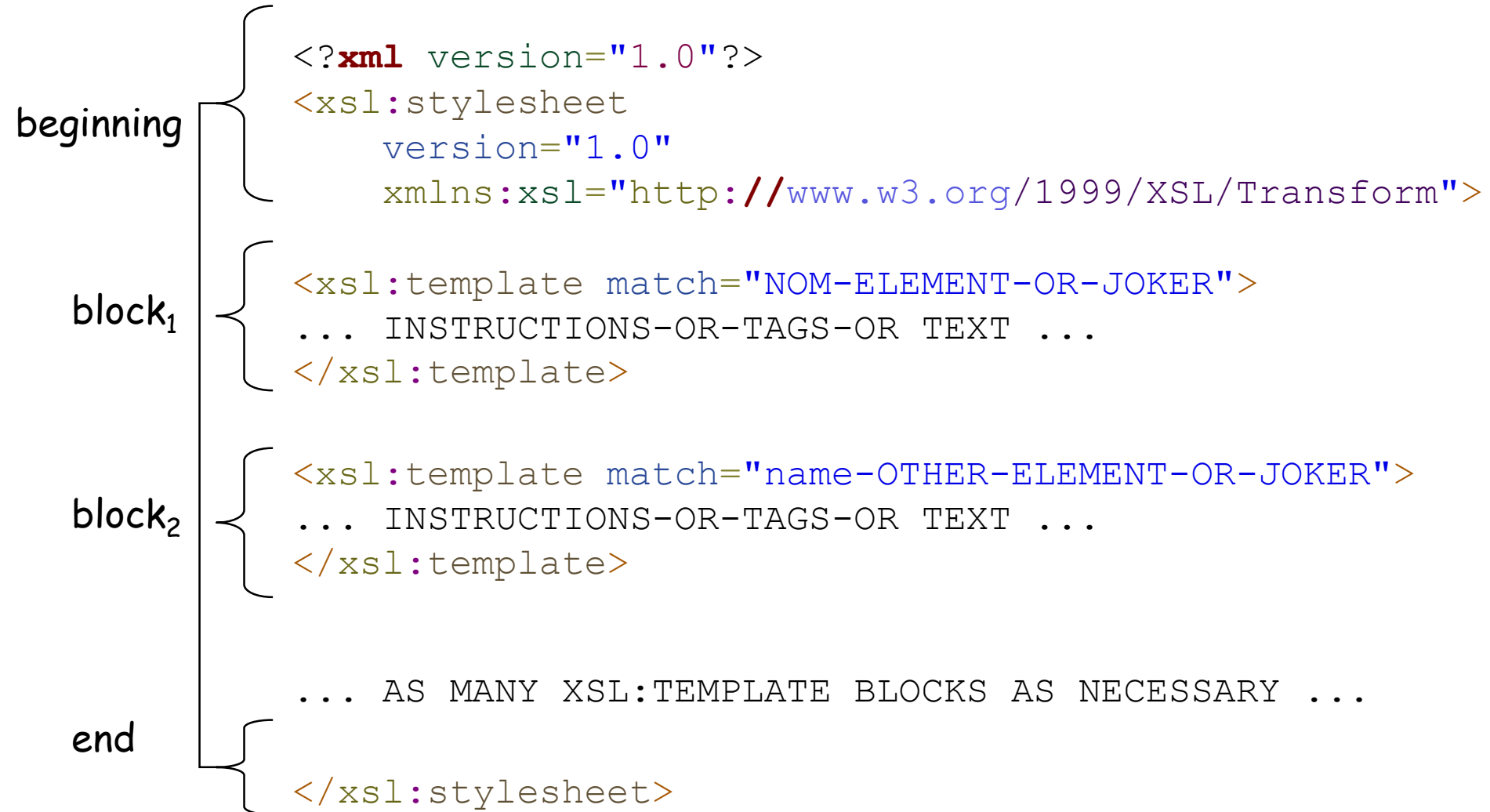
Results processing

- Templates have two parts
 - **pattern** to look for in the source tree
 - Depends on the structural context in the source tree
 - Expressed in **XPath**
 - **model** used to form the result tree
- Template Example

```
<xsl:template match = "title">  
    I am now in a "title" node.  
</xsl:template>
```

- selection pattern: term **match = "title"**
- Template Model = element content (XPath)
- The result displayed in the result document :
 "I am now in a "title" node "

Structure of a XSLT



XSLT transformation process

- Analysis of the source document nodes (sequentially)
 - Finding a matching template defined in the stylesheet
 - Application of the instructions contained in this template
- As soon as a template is found, it is applied
- If several templates found:
 - Templates from 2 style sheets (import external sheet).
 - Import priority (element `<xsl:import>`)
 - The importing sheet has a higher priority than the imported one
 - Templates of the same sheet: the highest explicit priority or the most accurate is chosen
 - **priority** Attribute of the template,
 - selectivity *pattern*: Rule most accurate = highest priority
- No template: default template is applied according to the type of the node

Default templates

- **element** type Nodes of an XML tree,
 - Relaunching the templates on every child node (recursive processing).

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

- **text()** and **attribute()** type nodes
 - Copy the text or attribute value.

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

- **processing-instruction()** and **comment()** type nodes
 - Do nothing.

```
<xsl: template match = "processing-instruction () |  
  comment ()" />
```

General model of an XSLT stylesheet

- hierarchical call of the templates.
 - Treat first the initial template that applies to the document root ('/').
 - Production of the result document frame (skeleton)
 - Calling other templates that will, in turn, produce the content of the result document.
- secondary templates only perform treatments on the nodes they target
 - Changing the layout of the result fragment.

Implementation of XSLT ⁽¹⁾

Transformation Process

The XSLT processor

- analyzes the source document (set of nodes)
- looking for a template that applies
- apply the instructions contained in this template
 - if there is one, it is applied
 - if there are several, the more specific is applied
 - if there is no applicable template, there are default templates
 - the children are processed recursively
 - the text is reproduced

Implementation of XSLT (2)

- Basic XSL Instructions

- Inside of an xsl:template:

- copy content without tags

```
<xsl:value-of select = "element name" />
```

- copy content with tags

```
<xsl:copy-of select = "element name" />
```

- Call the processing of other nodes

```
<xsl:apply-templates select = "XPath to the elements to  
process" />
```

Implementation of XSLT (3)

- Other XSL instructions
 - sorting structure
 - `xsl:sort`
 - conditional processing
 - `xsl:if`
 - `xsl:choose`
 - `xsl:otherwise`
 - ...
- Expressions in `match= "..."` and `select= "..."` are XPath expressions

Implementation of XSLT (4)

- predefined templates

- default if no transformation rules of the style sheet is applicable

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

• Research and execution of the templates that apply to every children of the context node

- relaunch the rules of the leaf to the root node or the son of the context node

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

- copy the text or attribute value if the context node is a leaf

# Implementation of XSLT (5)

## A simple example

```
<table>
 <description>4th floor staff</description>
 <person>
 <name>Bond</name>
 <office>U1</office>
 </person>
 <person>
 <name>Lupine</name>
 <office>U2</office>
 </person>
 <person>
 <name>Templar</name>
 <office>U3</office>
 </person>
</table>
```

# Implementation of XSLT (6)

## Data extraction

```
<xsl:template match ="person">

 <xsl:value-of select ="name"/> -
 <xsl:value-of select ="office"/>

</xsl:template>
```

## Value-of

- If the content of the context node
  - is a string, copy content
  - is mixed, leaves concatenation components then recopy

```
Bond - U1
Lupine - U2
Templar - U3
```

# Implementation of XSLT (7)

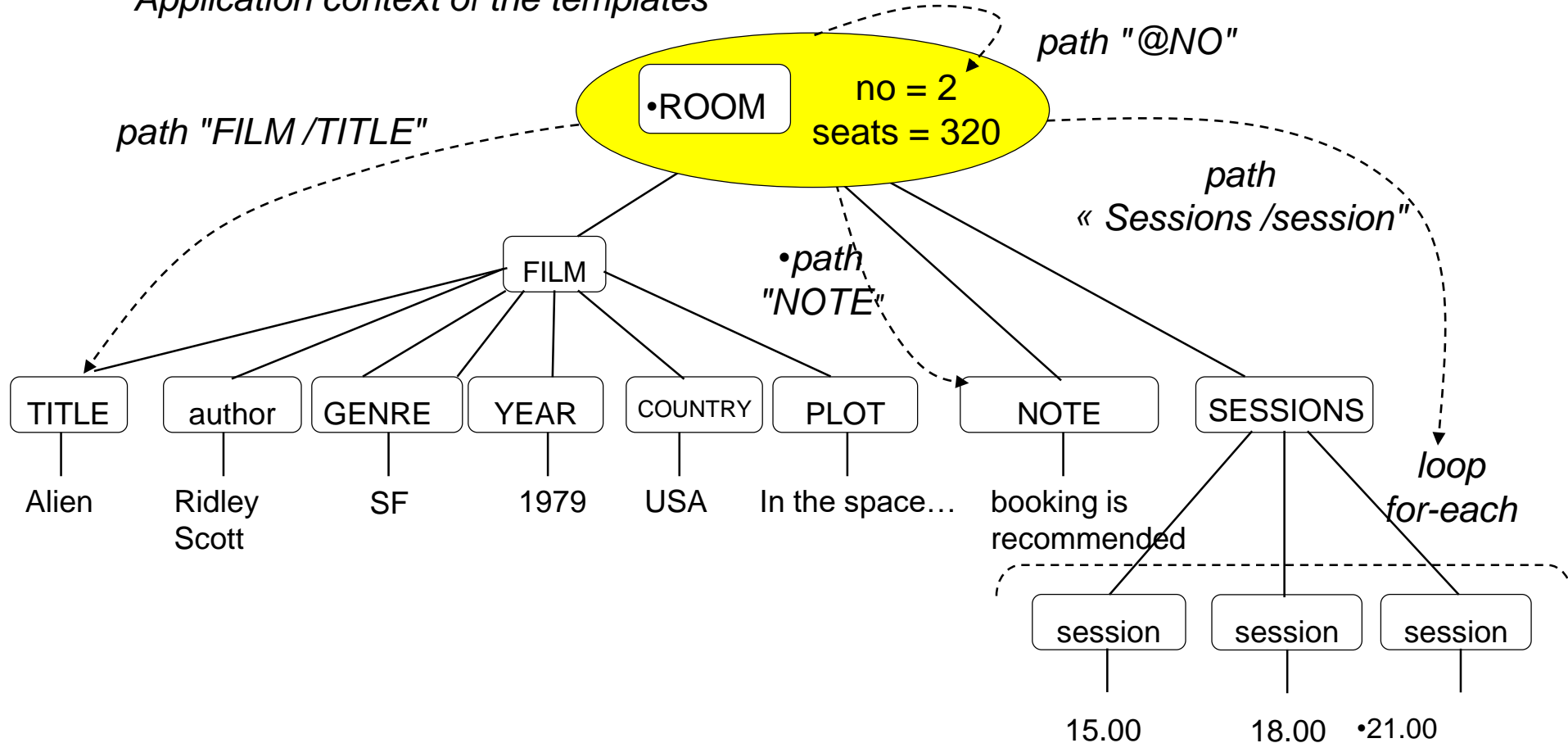
- **complex paths**

- The use of XPath for the expression of paths allows:
  - to access **descendants, relatives, siblings**
    - the title of the film is designated `select = "FILM/TITLE"`
  - to access **attributes**
    - The room number is designated `select = "@NO"`
  - perform **loops**
    - with `xsl:for-each` on all sessions designated by `select = "sessions/session"`
    - designate **the context element**
    - symbol `'.'` as `inselect = "."`

# Implementation of XSLT (8)

- Example room and sessions

*Application context of the templates*



# Implementation of XSLT (9)

## Rules on this example

```
<xsl:template match = "ROOM">
 <h2>
 Room No. <xsl:value-of select = "@no" />
 </h2>
 Movie: <xsl:value-of select = "FILM/TITLE" />
 <xsl:value-of select = "FILM/author" />

 <xsl:for-each select = "SESSIONS/SESSION">

 <xsl:value-of select = "." />

 </xsl:for-each>

</xsl:template>
```

# Implementation of XSLT (10)

- Result

- Applied Alien.xml:

```
<h2> Room No. 2 </h2>
```

```
Movie: Alien Ridley Scott
```

```

```

```
 15.00
```

```
 18.00
```

```
 21.00
```

```

```

- NB: This is an HTML fragment to be integrated into a comprehensive document.

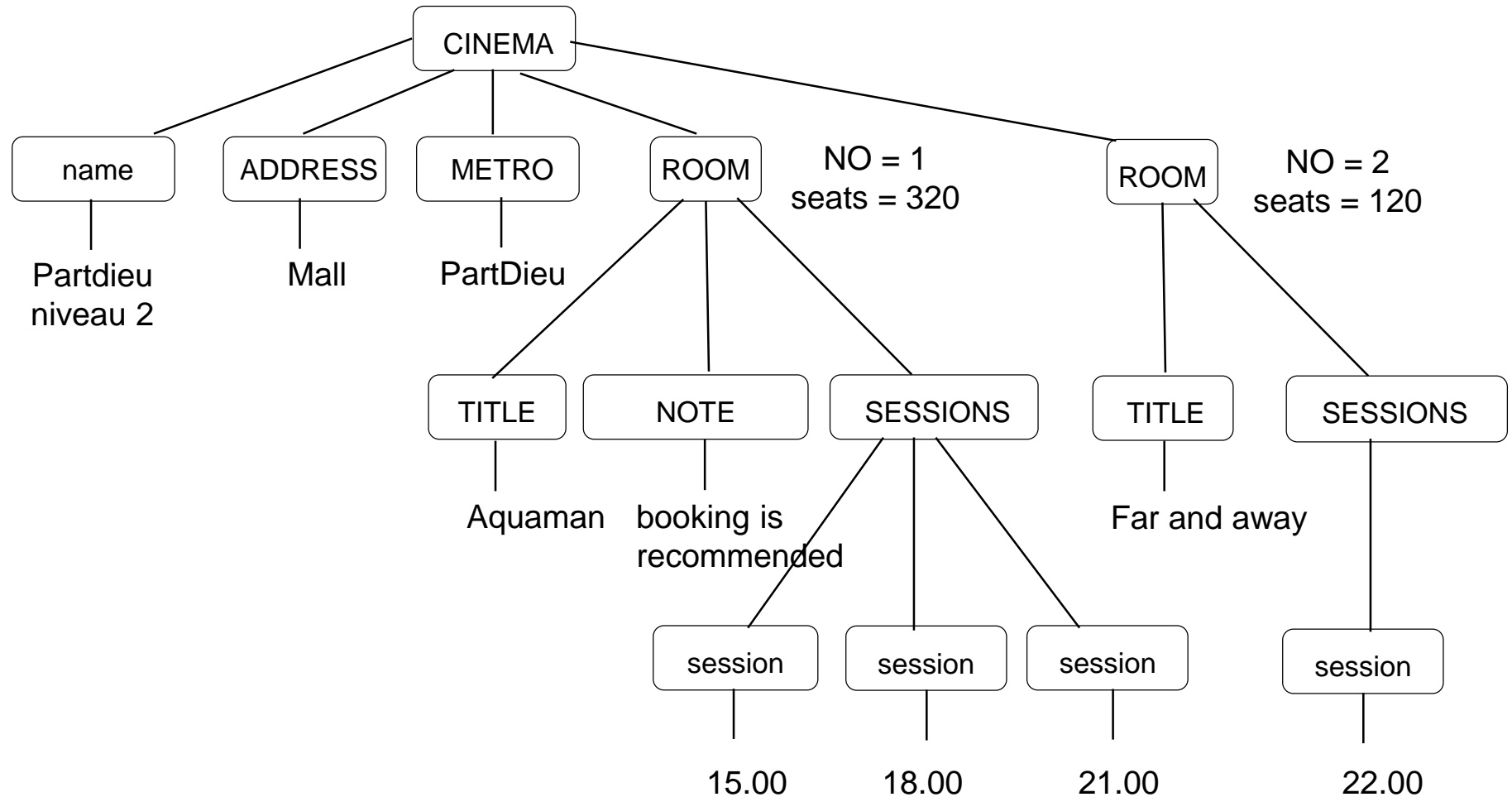
# Implementation of XSLT (11)

- template calls
  - Generally it produces a result by combining several templates:
    - The initial template applies to the document root transform ( '/')
    - It then produces the **frame of** the target document
    - other templates are then called to complete the creation of the result



# Implementation of XSLT (12)

## Cinema example



# Implementation of XSLT (13)

## first template

"Frame" HTML, and call the CINEMA template

```
<xsl:template match ="/">
 <html>
 <head>
 <title>Program of
 <xsl:value-of select ="CINEMA/name"/>
 </title>
 </head>
 <body bgcolor="White">
 <xsl:apply-templates select ="CINEMA"/>
 </body>
 </html>
</xsl:template>
```

# Implementation of XSLT (14)

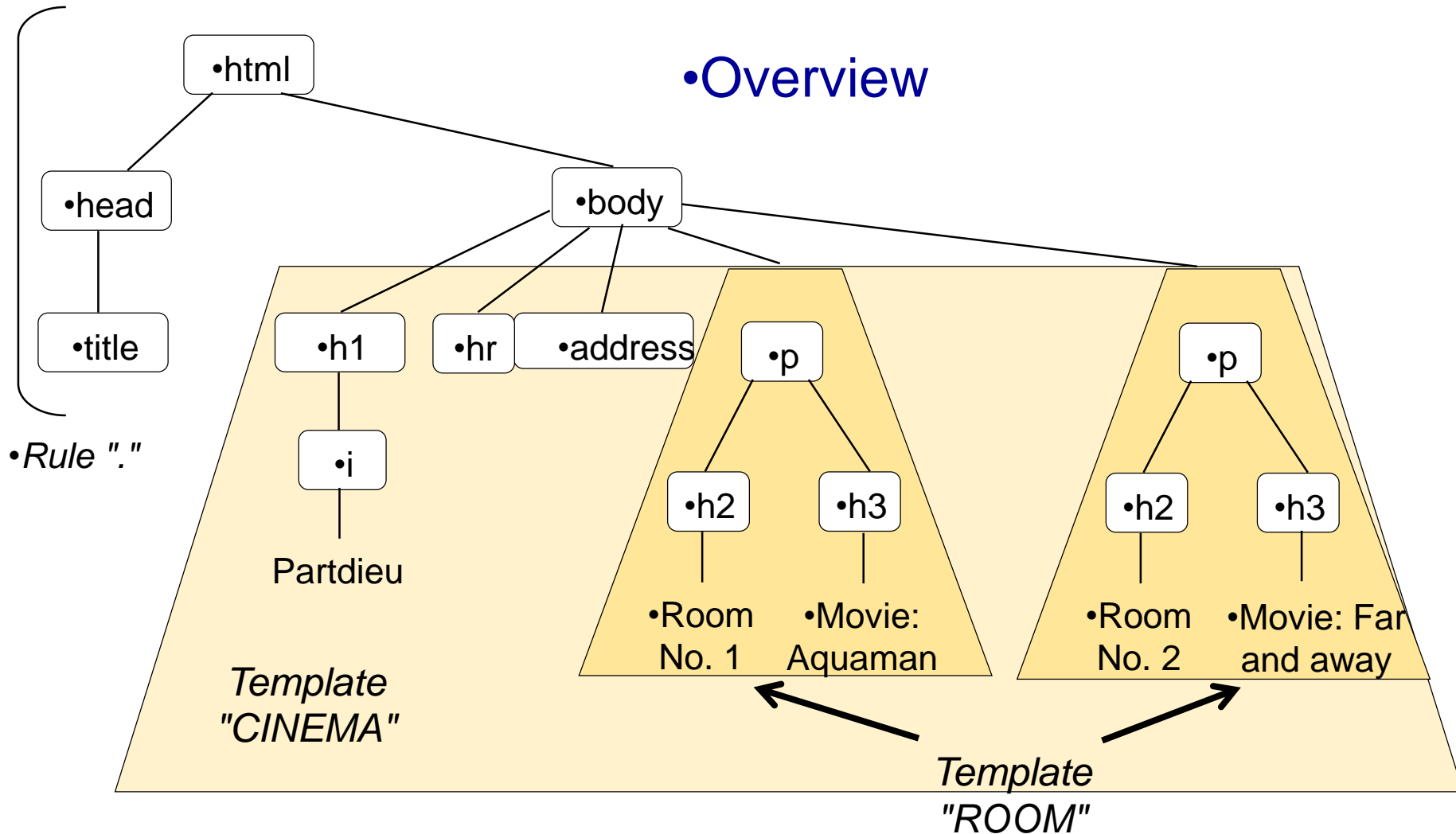
- Cinema template

- handling of the element CINEMA and call to the rutemplate ROOM

```
<xsl:template match = "CINEMA">
 <h1>
 <i> <xsl:value-of select = "name" /> </i>
 </h1>
 <hr/>
 <xsl:value-of select = "ADDRESS" />,
 <i>Metro: </i>
 <xsl:value-of select = "METRO" />
 <hr/>
 <xsl:apply-templates select = "ROOM" />
</xsl:template>
```

# Implementation of XSLT (15)

## •Overview



# Programming with XSLT <sup>(1)</sup>

- Combination of two programming modes:
  - declarative: create the rules, and the processor does the rest.
  - Imperative: use the 'Usual' programming structures (tests, loops, variables, procedure calls)
- non-modifiable variables
  - recursive programming

# Programming with XSLT (2)

## Example

```
<?xml version = '1.0' encoding= "ISO-8859-1"?>

<LECTURE>
 <SUBJECT>
 <TEACHERS>
 <!-- Responsible -->
 <name>Jean-Marie Pinon </name>
 <name>Elöd EGYED-ZSIGMOND </name>
 </TEACHER>
 <PROGRAM>
 <SESSION ID = "1"> XML </SESSION>
 <SESSION ID = "2"> XSLT - XPATH </SESSION>
 <YEAR> 2017 </YEAR>
 </PROGRAM>
 </SUBJECT>
</LECTURE>
```

# Programming with XSLT (3)

- `xsl:call-template`
  - Allows naming templates and calling them by their name
    - Is similar to a function call
    - Call does not change the context
      - Contrary to `xsl:apply-templates`
- passing parameters with `xsl:param`

# Programming with XSLT (4)

- A node display template

```
<xsl:template name="Display">node in position
 <xsl:value-of select ="position()" />: value of node = "
 <xsl:value-of select ="." />"

</xsl:template>
```

```
<xsl:template match ="name">name tag -
 <xsl:call-template name ="Display"/>
</xsl:template>
```

```
<xsl:template match ="text()">text() tag -
 <xsl:call-template name ="Display"/>
</xsl:template>
```

```
<xsl:template match ="comment()">tag comment () -
 <xsl:call-template name ="Display"/>
</xsl:template>
```



# Programming with XSLT (5)

- Result of the application of this rule on the example

tag comment () - node in position 1: value of **node** = "Responsible"

name tag - node in position 2: Value of **node** = "Jean-Marie Pinon"

name tag - node in position 3: value of **node** = " Elöd EGYED-ZSIGMOND "

text() tag- node in position 1: value of **node** = "XML"

text() tag- node in position 1: value of **node** = "XSLT - XPath"

text() tag- node in position 1: value of **node** = " 2017 "

<CURRENT>

<ON>

<TEACHERS>

<!-- responsible -->

<name> Jean-Marie Pinon </ name>

<name>Elöd EGYED-ZSIGMOND </ name>

</ TEACHER>

<PROGRAM>

<SESSION ID = "1"> XML </ SESSION>

<SESSION ID = "2"> XSLT - XPATH </ SESSION>

<YEAR> 2017 </YEAR>

</ Program>

</ TOPIC>

</ CURRENT>

# Programming with XSLT (6)

- Passing parameters

- We can pass parameters to `xsl: call-template` or `xsl: apply-templates`
  - With `xsl:param`
    - we define the expected parameters in the template
  - With `xsl:with-param`
  - It associate one or more `xsl: with-param` to `xsl: call-template` or `xsl: apply-templates`

# Programming with XSLT (7)

- Example

- The template View expects a string

```
<xsl:template name= "View">
 <xsl:param name= "text" Select =" string('unknown') "/>
 <xsl:value-of select = "concat(position(),':',$text)
"/>
</xsl:template>
```

- text is the parameter name
  - select gives the default value
  - \$text refers to the parameter

# Programming with XSLT (8)

- Passing parameters
  - Instruction xsl: with-param

```
<xsl:template match = "name">
 <xsl:call-template name= "View">
 <xsl:with-param name= "Text" select = "." />
 </xsl:call-template>
</xsl:template>
```

```
<xsl:template match = "comment ()">
 <xsl:call-template name= "View">
 <xsl:with-param name= "Text"
 select = "string('Comment')" />
 </xsl:call-template>
</xsl:template>
```

# Programming with XSLT (9)

- Definition levels

- Two levels of definition:

- In the body of a template

- The parameter is local to the template, and provided by the template call (`call` or `apply`)

- first level element

- child of the node `xsl:stylesheet`
      - the parameter is global and provided by the processor

# Programming with XSLT (10)

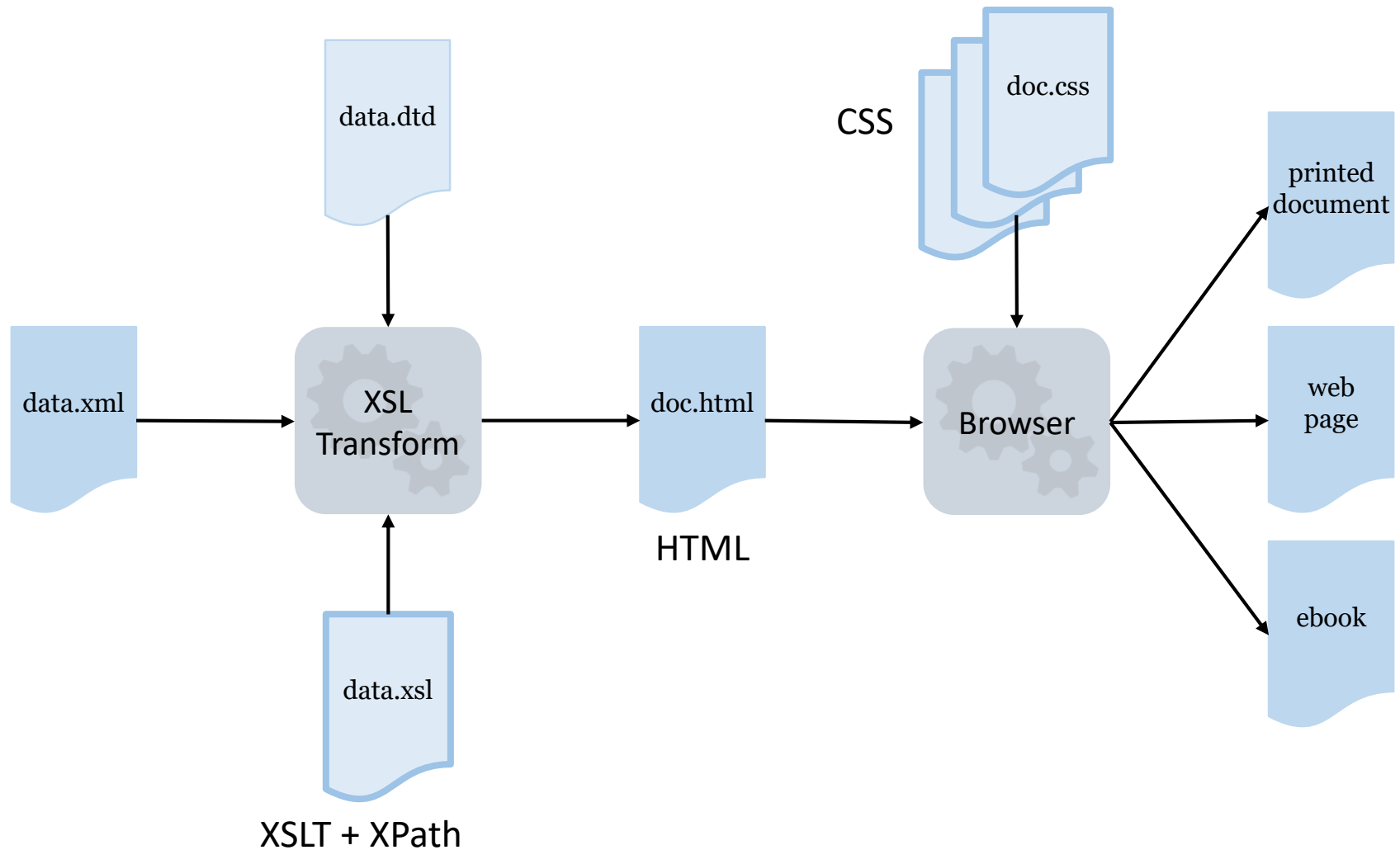
- Example: search engine

...

```
<xsl:param name= "title" />
<xsl:param name= "session"/>
<xsl:param name= "city" />

<xsl:template match = "MOTOR">
 <xsl:for-each select = "CINEMA">
 <xsl:if test = "
 ($title = ' ' or TITLE = $title)
 and ($session= ' ' or TIME > $session)
 and ($city = ' ' or CITY = $city)">
 <xsl: apply-templates select = "." />
 </xsl:if>
 </xsl:for-each>
</xsl:template>
```

# XSL processing in practice



## Style sheets and processing

Document transformation

XSLT Processing Model

XSLT document components

Conclusion

Read more ...



# Declaration and Root Element

```
<xsl:stylesheet version="Number" xmlns:xsl="URI">
 <!-- child elements -->
</xsl:stylesheet>
```

- XSLT: XML (XML declaration)
- Document element (root): **xsl:stylesheet**
  - *Must* be declared right after the document prologue
  - Contains all other elements of the document.
  - Attribute **version**: The version number of the sheet
  - Attribute **xmlns:xsl**: namespace used for the sheet
- Possible child Elements (*top-level Elements*)
  - any order (except **import** and **output** on your mind):
  - **import, include, strip-space, preserve-space, output, key, decimal-format, namespace-alias, attribute-set, variable, param, template**

# Basics: xsl:output

```
<xsl:output method= "xml" | "html" | "text" version= "NMTOKEN"
encoding= "string" omit-xml-declaration= "yes" | "no"
standalone= "yes" | "no" doctype-public= "string" doctype-
system= "String" cdata-section-elements= "elt" indent= "yes" |
"no" media-type= "string" />
```

- **<xsl:output>** specifies the options of the output shaft
- possible attributes:
  - **method**: Transformation method.
  - **version**: Version of the output method (xml 1.0 html 4.01).
  - **encoding**: Version of the character set to use for output.
  - **omit-xml-declaration**: Consideration of the XML declaration.
  - **standalone**: Output shaft with or without DTD.
  - **doctype-public**: Public identifier used by the associated DTD.
  - **doctype-system**: Identifying system used by the associated DTD.
  - **cdata-section-elements**: Elements whose content should be treated via a CDATA section.
  - **indent**: Conversion into shaft if value = yes.
  - **media-type**: MIME type of the resulting data.

# Basics: `xsl:strip-space`

- **`<xsl:strip-space>` and `<xsl:preserve-space>`:**
  - **`<xsl:strip-space Elements= "liste_Elements"/>`** allows deletion of unwanted white spaces for the items listed in the list.
    - example:  
`<xsl: strip-space elements= "name first_name"/>`
  - **`<xsl:preserve-space Elements= "liste_Elements"/>`** preserves white space such for individual items in a list.
    - example:  
`<xsl: preserve-space Elements= "Address" />`

# Basics: xsl:template

```
<xsl:template name= "templatename" priority=
"Number" match= "Expression" mode= "Mode_modele">
```

```
<xsl:call-template name= "template_name" />
```

- **<xsl:template>** defines a model / template to apply to a node and to a specific context.
- attributes:
  - **name** (Optional): name given to the template for an explicit call by the instruction **<xsl:call-template/>** in another element.
  - **match**: source nodes to which the rule applies.
    - Value = set of XPath expressions separated by '|' (union).
    - Only the axes *child* and *attribute* and *//* and */* operators are allowed.
  - **mode** (Optional): allows multiple templates depending on the output mode.

```
<xsl:template match = "book">
 {Template Content}
</xsl:template>
```

# Basics: `xsl:apply-templates`

```
<xsl:apply-templates select= "Expression" mode= "Mode_modele">
 <!-- Content -->
</xsl:apply-templates>
```

- **`<xsl:apply-templates>`** launches the templates on the nodes whose treatment it is called.
  - **`select`** (optional)
    - Absent: treatment of all children of the current node (including text nodes).
    - Present: only processing nodes selected by the expression
- **`<xsl:apply-templates>`** may be empty.
  - The processing of the children will use,
    - The templates defined in the style sheet for these items if they exist,
    - The default templates.
- **Ex:** In `biblio.xsl`, the call **`<xsl:apply-templates/>`** in the main template allows other templates to be executed.

# Basics: xsl:apply-templates

- **The modes:**

Their goal is to differentiate multiple templates applying to a given node (eg a chapter can be treated once to build the index and another time to publish content). To distinguish the two templates will have different modes.

```
<xsl:template match = "CHAPTER" mode= "index">
 <xsl:value-of select = "TITLE" />

</xsl:template>
<xsl:template match = "CHAPTER">

 <h1> <xsl:value-of select = "TITLE" /> </h1>
 <p> <xsl:value-of select = "TEXT" /> </p>

</xsl:template>
...
<xsl:apply-templates select="CHAPTER" mode="index"/>
```

# Basics: `xsl:value-of`

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

```
<xsl:copy-of select="Expression" />
```

- `<xsl:value-of>` copies the value of an expression
  - Expression can match an element, an attribute or any other node that contains a value.
  - `select`: Path to the node of which you want to copy the content.
- `<xsl:copy-of>` copies the node (including markup)
- **Ex:** In `biblio.xml`, `<xsl:value-of select = "publisher" />` used to copy the value of met `"editor"` nodes.

# Loops and choice: `xsl:for-each`

```
<xsl:for-each select= "Term">
```

- `<xsl:for-each>` to create a loop
  - `select`: Defines the list of nodes to be traversed by the loop.
  - takes the nodes one by one as context node
- **ex:** In `biblio.xsl`, the following example copy as result the value of all "*author*" nodes

```
<xsl:for-each select = ".//author">
 <xsl:value-of select = "." />
</xsl:for-each>
```



# Loops and choice: `xsl:sort`

```
<xsl:sort select= "node" data-type= "Text" | "number" |elt
order= "ascending" | "descending" lang=NMTOKEN
case-order= "Upper-first" | "lower-first" />
```

- `<xsl:sort/>` allows to sort a set of
  - used in `<xsl:for-each>` and `<xsl:apply-templates>`
  - Can be called multiple times to perform a multi-criteria sorting.
- attributes:
  - **select** Specifies a node as a sort key.
  - **data-type** (Optional): type of data to be sorted.
  - **order**: Order of the sort.
  - **lang** (Optional): character set to be used for sorting.
  - **case-order** (Optional): sorting on uppercase or lowercase
- **Ex:** In `biblio.xsl`, the following example copies the values of `"author"` nodes, alphabetically sorted.
  - ```
<xsl:for-each select = ".//author">  
  <xsl:sort select = "name" order = "ascending" />  
  <li> <xsl:value-of select = "." /> </li>  
</xsl:for-each>
```

Loops and choice: `xsl:if`

```
<xsl:if test= "Condition"> Action </xsl:if>
```

- `<xsl:if>` allows to perform tests on the attributes and XML elements.
 - `test`: Takes the value 1 or 0 depending on the result of the evaluation of the condition (*true* or *false*).
 - If the test is successful, the content of the element `<xsl:if>` is applied.
 - No `else` element...
- **EX:** In `biblio.xsl`, the following example tests if the attribute *kind* exists and displays it only if the test is successful.

```
<xsl:if test = ".//@kind">, type:  
  <SPAN style = "color: blue">  
    <xsl:value-of select = ".//@ kind" />  
  </SPAN>  
</xsl:if>
```

Loops and choice: `xsl:choose`

- `<xsl:choose>`, `<xsl:when>` and `<xsl:otherwise>` are used to define a list of choices and assign each a different processing.
 - Each choice is defined by an element `<xsl:when select="Expression">`
 - A default processing may be specified through the element `<xsl:otherwise>`.
- ex: In `biblio.xsl`, the following example tests the value of attribute *lang* and displays different results depending on its value.

```
<xsl:choose>
  <xsl:when test="@lang='fr' "> This book is in French
</xsl:when>
  <xsl:when test="@lang='en' "> This book is in English
</xsl:when>
  <xsl:otherwise> This book is in a not listed language.
</xsl:otherwise>
</xsl:choose>
```

The node creation functions

- Creating an element:

`<xsl:element name=qname>`

- Creating an attribute:

`<xsl:attribute name=qname> value`

- Creating a text node:

`<xsl:text> text`

- Creation of processing instructions:

`<xsl:processing-instruction name=qname>`

- Creating comments:

`<xsl-comment> text`

Create: `xsl:element` and `xsl:attribute`

```
<xsl:element name= "Element_name"  
  use-attribute-sets= "Ensemble_attributs">
```

```
<xsl:attribute name= "name"> value </xsl:attribute>
```

- `<xsl:element>` to insert a new element in the result tree
 - `name`: name given to the new element.
 - `Use-attribute-sets`: All attributes associated with the created element.
- `<xsl:attribute>` defines a new attribute to add to the element in the result tree.
 - `name`: name of the attribute to add to the current context.
 - Attribute Value created = the content of the element

Definition of variables: `xsl:variable`

```
<xsl:variable name= "Varname" select=
    "Value | Expression" />
```

- `<xsl:variable>` is used to create variables in XSLT.
 - `name` Specifies the name of the variable.
- Several possibilities for content initialization (variable value):
 - `select` with a value (a constant)
 - `select` with an Xpath expression
 - Variable defined by its content:
 - `<xsl:variable name= 'Var3'>`
Here are the contents of the variable
`</xsl:variable>`
- Calling the variable is done with `$variable_name`
 - `<xsl:value-of select = "$variable_name"/>`

Scope of a variable

- Scope limits variable elements to the brothers and their descendants.
 - If a variable is declared in a loop or a choice, we can not use it outside of this element.
- Two types of variables:
 - **globals** (First level element).
 - visible throughout the XSLT program
 - Declared before the **template** nodes
 - **local variables** (Declared inside the template body).
 - visible in the **following-sibling** and their descendants
 - after the **template**

Programming with XSLT

- Control instructions: global variable Example

```
<xsl:variable name= "year" Select ="1970"/>
<xsl:template match = "MOVIE">
  <xsl:choose>
    <xsl:when test = "YEAR &lt; $year">
      "<xsl:value-of select = "TITLE" />"
      is old
    </xsl:when>
    <xsl:when test = "YEAR &gt; $year">
      "<xsl:value-of select = "TITLE" />"
      is recent
    </xsl:when>
  </xsl:choose>
</xsl:template>
```


Programming with XSLT

- Control instructions: local variable Example

```
<xsl:template match = "SESSIONS">
  <SESSIONS>
    <xsl:variable name= "sentence">
      : (session of the movie:
      <xsl:value-of select = "../MOVIE/TITLE" />)
    </xsl:variable>

    <xsl:for-each select = "session">
      <xsl:value-of select = "concat(../@begining,
      $sentence) "/>
    </xsl:for-each>
  </SESSIONS>
</xsl:template>
```

Setting parameters: `xsl:param`

```
<xsl:param name= "param_name" select= "Value" />
```

```
<xsl:withparam name= "param_name" select= "Value" />
```

- `<xsl:param>` creates a parameter that can then be passed to a template through `<xsl:withparam>`
 - `name` Specifies the name of the variable
 - `select`: Gives the default parameter value.
- Two levels of definition:
 - In the body of a template (local)
 - Provided by the application (`call` or `apply`)
 - First level element (global)
 - Child of `xsl:stylesheet`
- The access to the parameter is done with `$param_name`
 - `<xsl:value-of select = "$param_name"/>`

Example setting

- The template `view` expects a string:

```
<xsl:template name= "View">  
  <xsl:param name= "Text" select = "string ( 'unknown')" />  
  <xsl:value-of select = "concat(position(),':',$ text) "/>  
</xsl:template>
```

- Passage of the parameter:

```
<xsl:template match = "comment ()">  
  <xsl:call-template name= "View">  
    <xsl:with-param name= "Text" select = "string ( 'comment')" />  
  </xsl:call-template>  
</xsl:template>
```

XSL functions

Name	Description
<u>current()</u>	Returns the current node
<u>document()</u>	Used to access the nodes in an external XML document
<u>element-available()</u>	Tests whether the element specified is supported by the XSLT processor
<u>format-number()</u>	Converts a number into a string
<u>function-available()</u>	Tests whether the function specified is supported by the XSLT processor
<u>generate-id()</u>	Returns a string value that uniquely identifies a specified node
<u>key()</u>	Returns a node-set using the index specified by an <xsl:key> element
<u>system-property()</u>	Returns the value of the system properties
<u>unparsed-entity-uri()</u>	Returns the URI of an unparsed entity

source: w3schools.com

SQL SELECT DISTINCT

- Unique city elements

```
//city[not(text()=preceding::city/text())]
```

equivalent to *not in*

Style sheets and treatments

Document transformation

XSLT Processing Model

XSLT document components

Conclusion

Read more ...

Conclusion on XSLT

- A template language totally suited to processing XML documents
 - Through a document, as a tree
 - templates trigger on some nodes
 - Combine several programs for the same document
- XPath Language Integration
- **Disadvantages of XSLT:**
 - Although version 1.0 has been stabilized, the tools that implement XSLT do not meet all the recommendation (missing features, adding features and proprietary elements).

Read more ...

Courses and tutorials online

http://www.w3schools.com/xml/xsl_intro.asp

The reference:

<http://www.w3.org/Style/XSL/>

Bernd Amann and Philippe Rigaux: Understanding XSLT, O'Reilly, 2002

Plan

- Introduction
- Core XML
- XML galaxy
 - XPath
 - XSL
 - Handling
 - applications
- NOSQL
- Conclusion

Plan

- Introduction
- Core XML
- Galaxy XML
 - XPath
 - XSL
 - Handling
 - DOM
 - SAX
 - Javascript
 - applications
- NOSQL
- Conclusion

- **The XML APIS:**

- Are parsers (*parser* in English, sometimes in Frenchified *parser*), libraries offering a set of software components
- Allow and facilitate the reading, the generation and processing of XML documents.

- **These APIs are implemented on the basis of two models:**

- *hierarchical*: XML DOM (Document Object Model) allows the manipulation of an XML document after represented in memory as an object tree
- *Events*: SAX (Simple API for XML) allows the manipulation of an XML document as and when it is playing, without having to load it entirely in memory. Data extraction is based on an event management (creation of a document, an element ...).

- DOM is a w3 consortium standard
 - <http://www.w3.org/DOM/>
- SAX is a specification available in an open source project.
 - <http://www.saxproject.org/>
- There are many implementations of the DOM standard and SAX specification

DOM

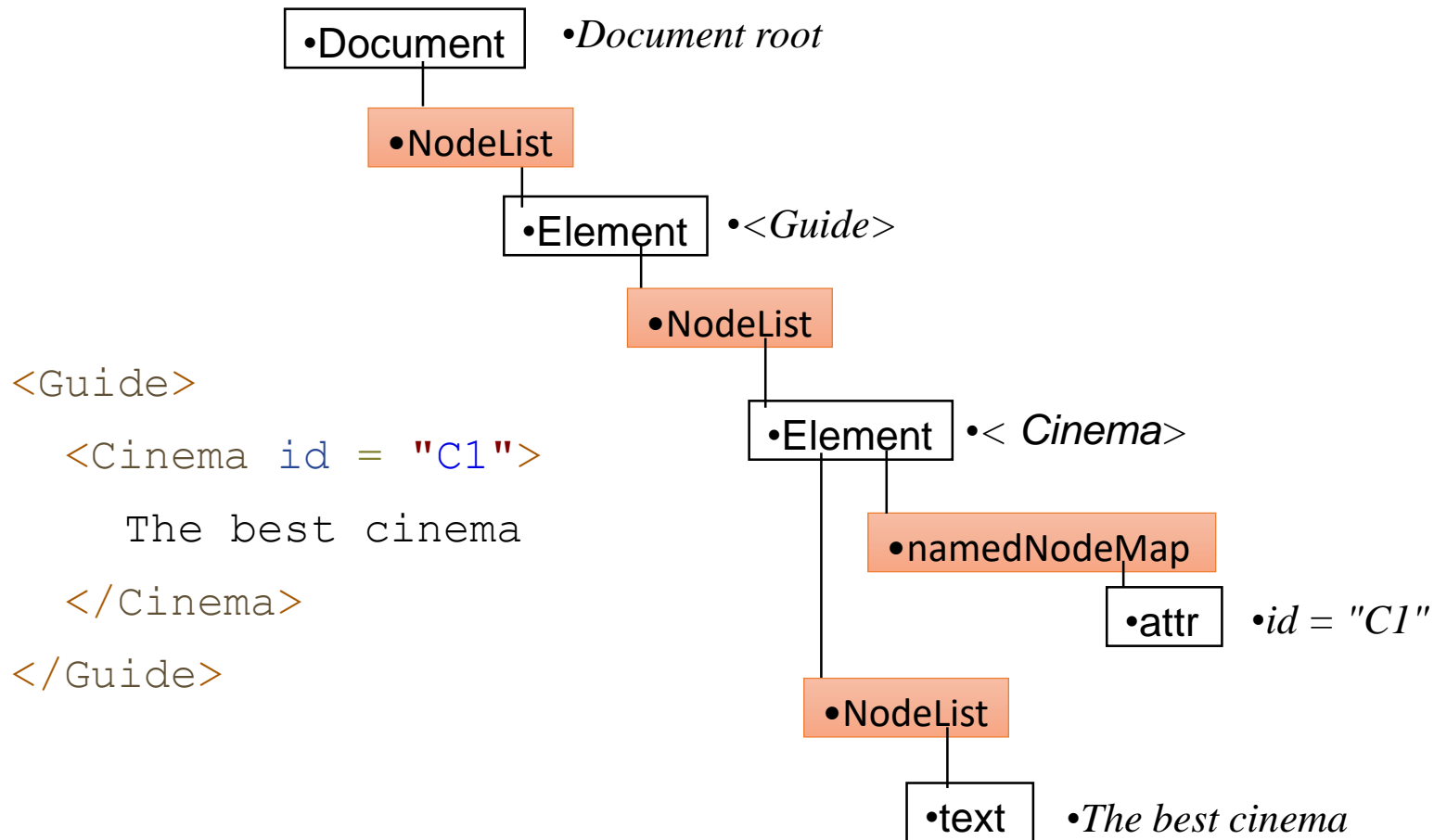
- **DOM: Document Object Model**

- APIs for HTML and XML documents
- defines a generic logical structure of the document and how to access and process the relevant nodes
- object-oriented model which allows the identification and description of:
 - interfaces and objects required to represent and manipulate a document;
 - the semantics of these interfaces and objects - structure and behavior
 - the relationships and collaborations between objects and interfaces
- navigation framework
- Document handling framework (*Logical structure and content*)

The DOM interface

- Object structure to represent a document
 - The result of a "parser"
 - Tree of objects linked together
- Object interface to navigate through a document
 - Can be used in:
 - Java, C ++, javascript
 - C #, VB
 - Python, PHP
 - ...
- XML parser
 - Generating an object tree
 - Building the whole document tree in memory

Example DOM tree



DOM (java) Core Level 1

- interfaces
 - Document,
 - Element,
 - text,
 - Comment,
 - CDATASection,
 - notation,
 - Node
- Exception
 - DOMException

org.w3c.dom.Node

- The base class of all DOM object is: Node
- Methods
 - public Node **getFirstChild()**;
 - public java.lang.String **GetNodeName()**;
 - public short **getNodeNodeType()**;
 - public Node **appendChild**(Node newChild) throws DOMException
 - ...

org.w3c.dom.namedNodeMap

- unordered list of Nodes
- Search by name
- Methods
 - public Node **getnamedItem**(Java.lang.String name);
 - public Node **setnamedItem**(Node arg) throws DOMException
 - ...

org.w3c.dom.NodeList

- Ordered list of Node
- The index is zero
- Methods
 - public int **getLength()**;
 - public Node **item**(int index) ;

org.w3c.dom.Document

- A document type object represents the XML document being processed
- Methods:
 - public NodeList **getElementsByTagName**(String tagname)
 - public Element **getDocumentElement**();
 - public attr **createAttribute**(String name) throws DOMException
 - public text **createTextNode**(String data)
 - public How **createComment**(String data)
 - ...

org.w3c.dom.Element

- An Element object represents an element of the XML document
- Methods:
 - public java.lang.String **getAttribute**(String name);
 - public NodeList **getElementsByTagName**(String name);
 - public java.lang.String **getTagName**();
 - ...

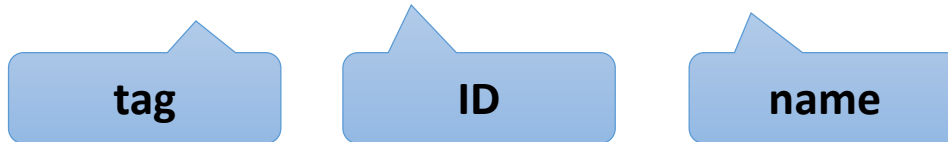
org.w3c.dom.Attr

- An Attr object represents an attribute of an element of the XML document
- Methods:
 - public java.lang.String **getName**();
 - public java.lang.String **getValue**();
 - public void **setValue**(Java.lang.String value) throws DOMException
 - ...

The HTML DOM

- important functions
 - createElement (...)
 - getElementById (...)
 - getElementByName (...)
 - getElementsByTagName (...)
- The parser looks for elements according to

```
<img id = "foo" name ="bar"/>
```



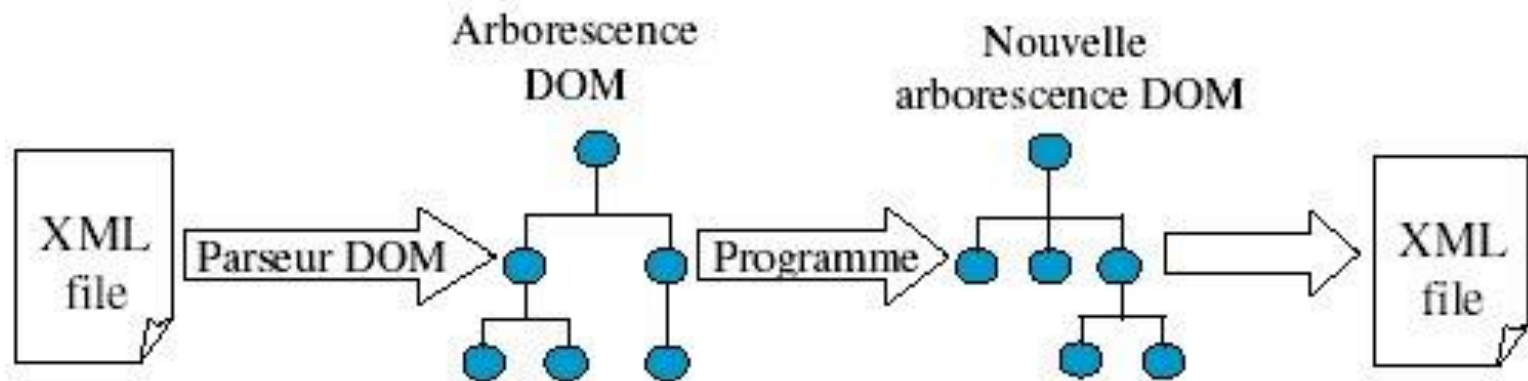
Special DOM

- Creating items
 - **Document.createElement**(Java.lang.String tagname)
 - **Document.createTextNode**(Java.lang.String data)
 - ...

Document handling

The API `org.xml.dom`:

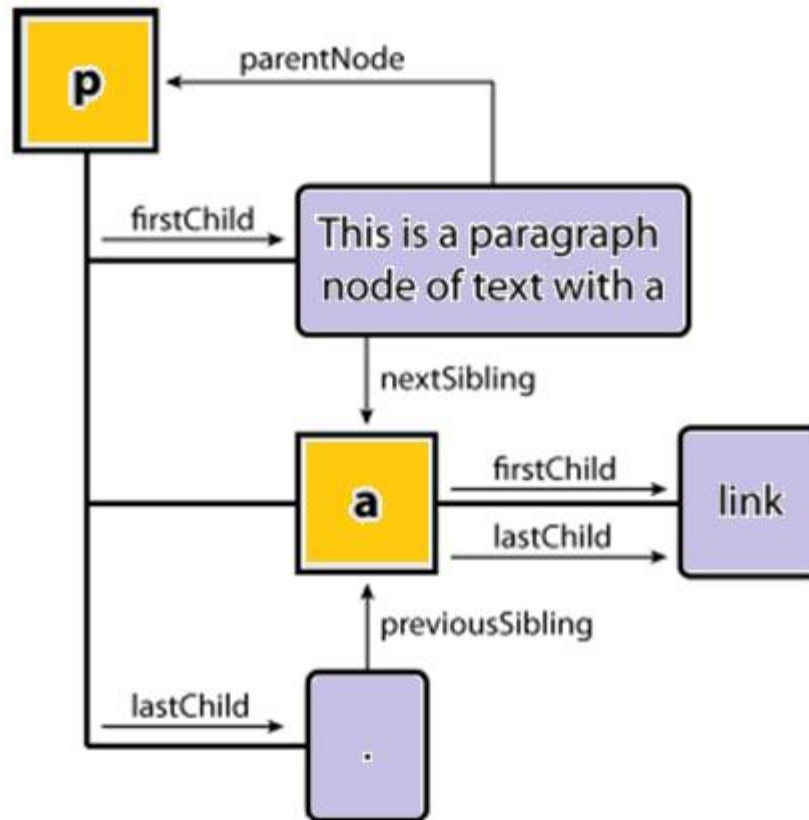
- It implements the DOM interfaces and provides classes and methods for manipulating an XML document through a tree management. We can thus add or remove node, Add or delete an attribute to a node (methods `appendChild`, `setAttribute` of style `Element` for example)...
- A DOM parser constructs a DOM object and provides a breadth-first and depth first traversal of the tree to process each node.



DOM example

```
<P id = "foo"> This is a paragraph of text with a  
<a href= "/Path/to/page.html"> link </a>. </ P>
```

HTML



Elements and text nodes

```
<div>
  <P>
    Ceci est un paragraphe avec un
    <a href= "Page.html">lien</a>.
  </P>
</div>
```

- Q: How many children has the div node above?

Elements and text nodes

```
<div>  ↵  
  <P>  
      Ceci est un paragraphe avec un  
      <a href= "Page.html">lien</a>.  
  </P>  ↵  
</div>
```

- Q: How many children has the div node above?
- R: 3
 - An element corresponding to <P>
 - 2 text nodes corresponding to line feed characters: "\n\t" (before and after the <P> element)

Elements and text nodes

```
<div>
  <P>
    Ceci est un paragraphe avec un
    <a href= "Page.html">lien</a>.
  </P>
</div>
```

- Q: How many children has the <P> node?
- R: the <a> tag?

```
$x("//div/p/child::*"):
```

```
[<a href="page.html">lien</a>]
```

```
$x("//div/p")[0].childNodes:
```

```
[ "Ceci est un paragraphe avec un", <a href="page.html">lien</a>, ". " ]
```

Java example

```
DocumentBuilderFactory manufactures =  
DocumentBuilderFactory.newInstance();  
  
// create a constructor documents  
DocumentBuilder constructeur = manufactures.newDocumentBuilder();  
  
// read the contents of an XML file with DOM  
file xml = New File (filename);  
Document document = constructeur.parse(xml);  
  
Element root = document.getDocumentElement();  
NodeList list = noeudDOMRacine.getElementsByTagName("CD");  
  
// course CD Elements  
for (int i = 0; i < liste.getLength(); i++) {  
    Element CDElement = (Element) liste.item(i);  
    ....  
}
```

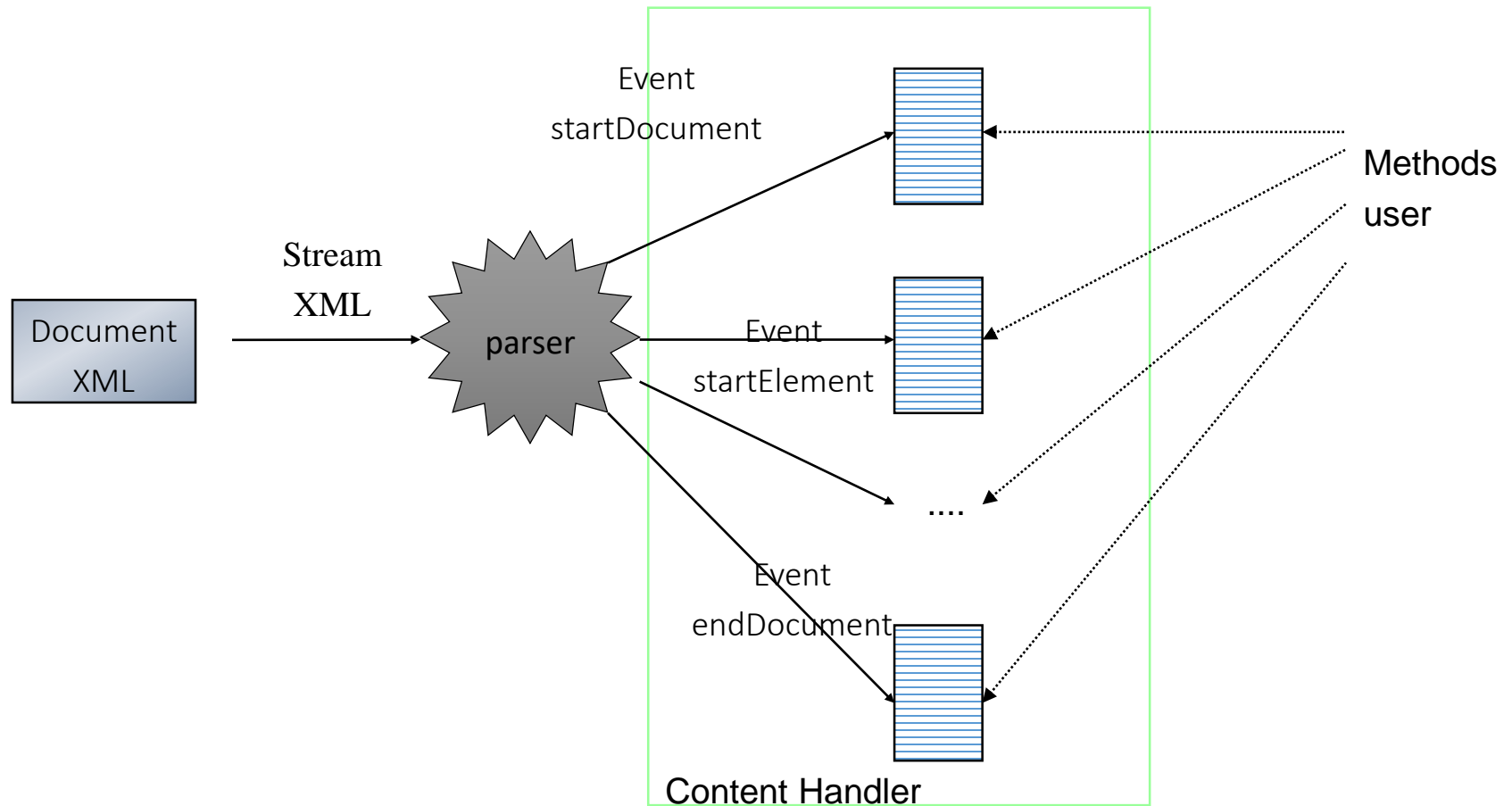
Plan

- Introduction
- Core XML
- XML Galaxy
 - XPath
 - XSL
 - Handling
 - DOM
 - SAX
 - Javascript
 - applications
- NOSQL
- Conclusion

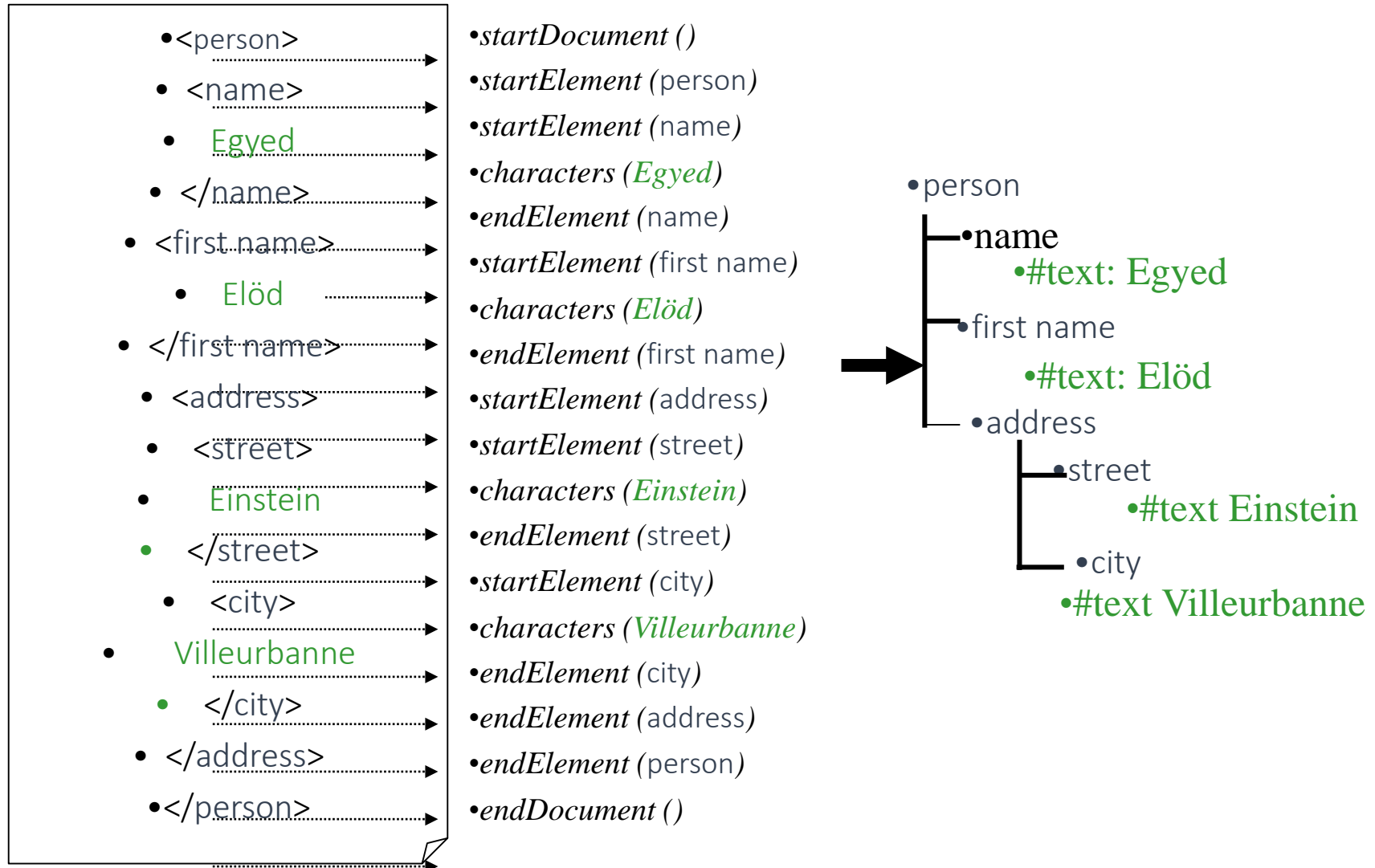
The SAX interface

- SAX (Simple API for XML)
 - simplified model of events
 - developed by an independent group.
- Event types:
 - beginning and end of the document;
 - start and end of elements;
 - attributes, text,
- Many public parsers
 - XP James Clark, Aelfred Saxon
 - MSXML3 Microsoft
 - Apache Xerces
 - JAXP SUN

Operating principle



Example SAX and DOM



SAX 2.0

- interfaces
 - XMLReader
 - ContentHandler
 - DTDHandler
 - ErrorHandler
 - ...
- classes
 - InputSource
- exceptions
 - SAXException
 - ...

CONCLUSION

- **SAX or DOM?**

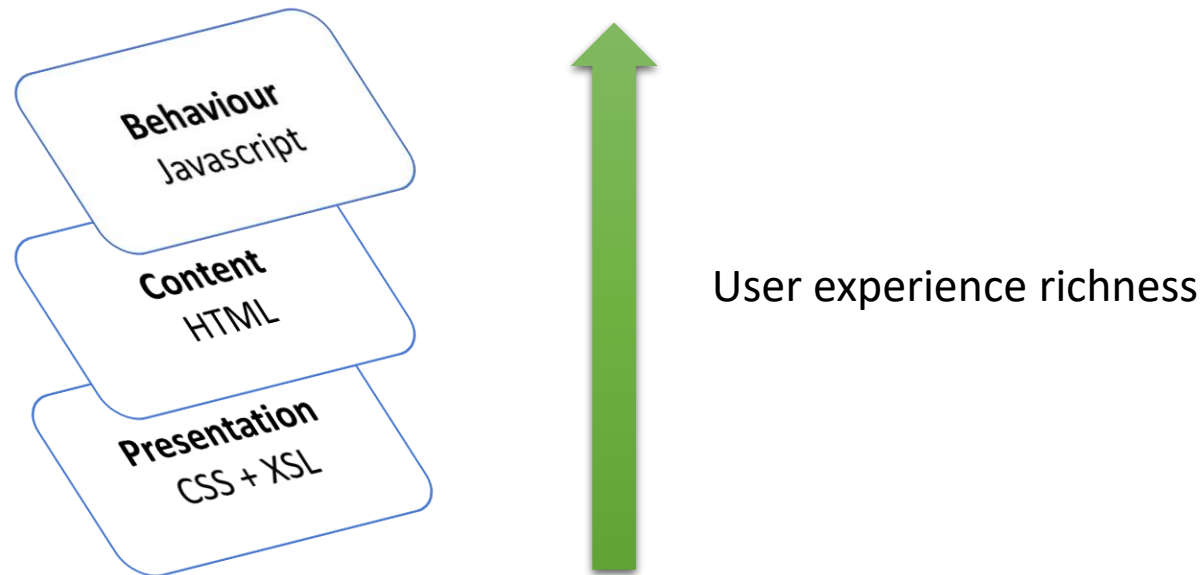
	Avantages	Inconvénients
SAX	<ul style="list-style-type: none">• le traitement du document par le programme se fait en cours d'analyse (efficacité)• seuls les éléments pertinents sont traités	<ul style="list-style-type: none">• l'écriture des fonctions de callback pour traiter des structures imbriquées est plus complexe
DOM	<ul style="list-style-type: none">• une interface navigationnelle est fournit pour parcourir un arbre d'éléments	<ul style="list-style-type: none">• un arbre est construit en mémoire• tous les éléments sont représentés• le traitement du document par le programme se fait après l'analyse

Plan

- Introduction
- Core XML
- Galaxy XML
 - XPath
 - XSL
 - Handling
 - DOM
 - SAX
 - Javascript
 - applications
- NOSQL
- Conclusion

Introduction

- JavaScript is a scripting language (programming) that can be included in HTML pages.
- With this language, it is possible to write interactive pages.



Coding Tips

- Always close an open curly bracket.
- We can create strings using single or double quotes.

JavaScript: The basic types

Here are the seven main JavaScript types that are part of the ECMA standard (European Computer Manufacturers Association):

- Object
- String
- Number
- Array
- date
- RegExp
- Function

Object oriented

- JavaScript is an object oriented language without the use of classes (prototyped language)
- The objects are handled as class
- Dynamic aspect for modifying the structure of the objects after they are created (by object-oriented prototype).

JavaScript

- High-Level vs. Low Level
- Compiled vs. Interpreted
- Structured vs. Object Oriented
- Scripting vs. Programming

Javascript

Executable contents `<script>`

Why a script: Adding fragments to static web pages, creating dynamic pages (write HTML output)

A script performs at client side

Javascript provides a fairly complete set of “set function” of integrated controls, enabling to perform calculations, manipulate strings, play sounds, open new windows and URL, access information provided by a user in an HTML form and check it.

Where are the scripts?

- In the HTML page
- Like styles, can be external, internal, or inline
 - Use these for different situations
- For scripts that respond to user events, and for functions:
 - Either external or internal
 - In `<head>...</head>`
- For scripts that write document content:
 - In `<body>...</body>`

Body example

```
<html>
```

```
<head>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
document.write("This message written by JavaScript");
```

```
</script>
```

```
</body>
```

```
</html>
```

Internal example

```
<html>
<head>
<script type="text/javascript">
function message()
{
    alert("This alert was called with the onload event");
}
</script>
</head>

<body onload="message()">
</body>
</html>
```

External javascript example

```
<html>  
<head>  
  <script type="text/javascript" src="xyz.js"></script>  
</head>  
<body>  
</body>  
</html>
```

JavaScript Basics

- Statements
- Variables
- Events
- Functions
- Dialogs

Statements

```
<script type="text/javascript">
```

```
var a = 10;
```

```
var b = 11;
```

```
var c;
```

```
c = a + b;
```

```
alert('The answer is' + c);
```

```
</script>
```


JavaScript Variables

- Variables are used to store data.
- Rules for variable names:
 - Variable names are case sensitive
 - They must begin with a letter or the underscore character
 - strname – STRname (not same)
- Variables: containers for data
 - All variables have
 - name
 - Type – JavaScript is loosely typed
 - Value or “null”
 - To declare a variable
 - `var variablename`
 - Beware of reserved words:
 - E.g., ‘if’, ‘Document’, ‘Math’, etc.

Variables

```
var name = expression;
```

```
var clientname = "Connie Client";  
var age = 32;  
var weight = 127.4;
```

- variables are declared with the var keyword (case sensitive)
- types are not specified, but JS does have types ("loosely typed")
 - Number, Boolean, String, Array, Object, Function, Null, Undefined
 - can find out a variable's type by calling `typeof`

JavaScript Operators

• Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 y=2 x+y	4
-	Subtraction	x=5 y=2 x-y	3
*	Multiplication	x=5 y=4 x*y	20
/	Division	15/5 5/2	3 2,5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

• Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
=	x=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

JavaScript Operators

• Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5 y="5" x==y returns true x===y returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

• Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
	or	x=6 y=3 (x==5 y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Javascript functions

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function Greeting ()
      {
        alert ( "Hi" );
      }
    </script>
  </head>
  <body>
    <button onclick="Greeting ()">Say hello</button>
  </body>
</html>
```

Events, Dialogs

EVENT HANDLER

onAbort
onBlur
onChange
onClick
onDbClick
onDragDrop
onError
onFocus
onKeyDown
onKeyPress
onKeyUp
onload
onMouseDown
onMouseMove
onMouseOut
onMouseOver
onMouseUp
onMove
onReset
onResize
onSelect
onSubmit
onUnload

- `alert("Hello!");`
- `confirm("Delete files?");`
- `var name=prompt("Your name?",
"Tim Berners-Lee");`

Coding Tips

- Check your statements are on one line
- Check your " and ' quotes match
- Take care with capitalisation
- Lay it out neatly - use tabs
- Use the browser developer tools

A Word About Comments

- JavaScript Comments

- These comments must be within a script
- `//` begins a single line comment
 - These can start anywhere on the line, but the remainder of the line becomes a comment
- `/*` All of your comments here... `*/`
 - Everything between the start and end comment markers are comments
 - Can be on a single line or span many...

- HTML Comments

- `<!--` All of your comments here... `-->`

Ending Statements With a Semicolon?

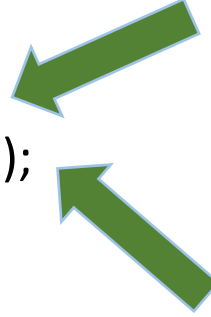
- With traditional programming languages, like C++ and Java, each code statement has to end with a semicolon (;).
- Many programmers continue this habit when writing JavaScript, but in general, semicolons are optional!
- Semicolons are required if you want to put more than one statement on a single line.

Using the DOM to get an element

```
<html>  
<body>
```

```
<p id="intro">Hello World!</p>
```

```
<script type="text/javascript">  
x=document.getElementById("intro");  
document.write(x.firstChild.nodeValue);  
</script>
```



```
</body>  
</html>
```

Javascript and DOM Example 1.1

```
<Input id ="MyButton3" type ="button" value ="Example Ajax XSL"  
onclick="ajaxBibliographie(  
'ajax.files/ajax.bib.xml''ajax.files/ajax.bib.xsl''ul'); " />
```

.....

```
function ajaxBibliographie(xmlDocumentUrl, xslDocumentUrl,  
elementID) {
```

```
    // Load the XSL file using XMLHttpRequest synchronous (the 3rd  
parameter is set to false)
```

```
    var myXMLHttpRequest = new XMLHttpRequest();  
    myXMLHttpRequest.open("GET", xslDocumentUrl, false);  
    myXMLHttpRequest.send(null);
```

```
    var xslDocument = myXMLHttpRequest.responseXML;
```

```
    var XSLTProcessor = new XSLTProcessor();
```

```
    // Importing.xsl  
    xsltProcessor.importStylesheet(xslDocument);
```

Javascript and DOM example 1.2

```
// Load the XML file using synchronous XMLHttpRequest (the 3rd
parameter is set to false)
myXMLHttpRequest = new XMLHttpRequest();
myXMLHttpRequest.open("GET", xmlDocumentUrl, false);
myXMLHttpRequest.send(null);

var xmlDocument = myXMLHttpRequest.responseXML;

// Create the XML document converted by the XSL
var newXmlDocument =
xsltProcessor.transformToDocument(xmlDocument);

// Parent Locator (with id corresponding to parameter) Of the
element to replace in the current HTML document
var elementHtmlParent = window.document.getElementById(elementID);
// Element replacement
elementHtmlParent.replaceChild(elementAInserter,
elementHtmlAReplacer);

}
```

Plan

- Introduction
- Core XML
- Galaxy XML
 - XPath
 - XSL
 - Handling
 - Applications
- NOSQL
- Conclusion

XML applications

2. Standard applications

2.1. Introduction

2.2. Hypertext: XHTML

2.3. Mathematics: MathML

2.4. Synchronized Multimedia: SMIL

2.5. Vector graphics: SVG

2.6. Document Description: RDF

Read more ...

Introduction

- XML is a meta-description language, for creating more specialized languages for various fields.
 - Mathematics,
 - 2D graphics,
 - Multimedia,
 - ...
- Development of a coherent set of languages designed to be used together
- As related standards (XPath, XSL, XLink, ...), these languages can become W3C Recommendations.

Applications Recommended W3C

- Applications developed by W3C to represent different aspects of Web documents
 - **XHTML**
 - **MathML** (Mathematical Markup Language) For description and communication of mathematical and scientific information on the Web,
 - **SMIL** (synchronized Multimedia Integration Language) to the fine management of multimedia resources on the Web
 - **SVG** (Scalable Vector Graphics) for describing XML 2D graphics,
 - **RDF** (Resource Description Framework) to associate descriptors to all XML resources to facilitate research ...

XML applications

2. Standard applications

2.1. Introduction

2.2. Hypertext: XHTML

2.3. Mathematics: MathML

2.4. Synchronized Multimedia: SMIL

2.5. Vector graphics: SVG

2.6. Document Description: RDF

Read more ...

History

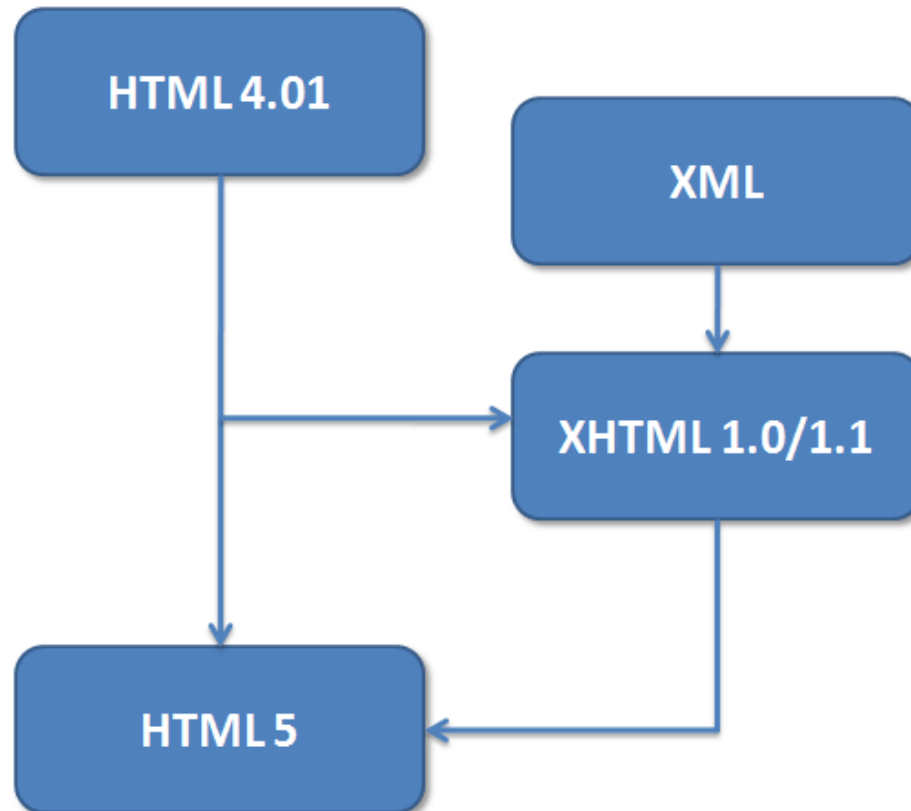
- XHTML has been validated by the w3C since January 2000 and is an XML application dedicated to web publishing.
- Facts about HTML
 - limited formatting language
 - Designers want to make more and more complex pages .
- Perversion of the original standard: code pages
 - heavy,
 - incoherent
 - Usable only for a given browser version,
 - ...

objectives

- After HTML 4.01 (1997) develop the next generation of HTML using XML technologies with several objectives:
 - Promote the development of correct pages
 - HTML integration with other XML applications
 - Allow modularity to adapt to the increasing variety of browsers
 - Reducing production costs websites
- <http://www.w3.org/MarkUp/xhtml-roadmap/>

Markup Languages

- The relationship between XHTML, HTML, and XML



Advantages

- XHTML forces a **cleaner and stricter writing** tags which is very close to HTML syntax.
- 2 main advantages:
 - **Extensibility**: Without being as open as XML, XHTML rigor makes it scalable and easily "pensionable" by DTD.
 - **Portability**: Current browsers are very tolerant of malformed HTML and flawed pages. In order to be read by various devices (PDAs, laptops ..) would require that the code is light and clean ... it's what provides the XHTML standard.

Tags and attributes

- Naming conventions XHTML:
 - **lowercase**: The tags as their attributes must be lowercase.
 - **Quotation marks**: Attributes must be quoted.
 - **minimization**: The shortcut notation of boolean attributes are no longer allowed.
 - So `<Frame noresize>`
 - becomes `<frame noresize = "noresize">`.
- **new attribute** for anchors:
 - The attribute `"name ="` of the tag `a` used to indicate the position of an anchor in HTML, is replaced by the attribute `"id ="`

stricter syntax

- **Closing:**

- More closing tags forgetting tolerance
- Empty tags must be closed.
 - `
` becomes `
`

- **imbrication:**

- Nesting strict beacon
 - ` 1 2 `

- **Obligation:**

- omissibles attributes now mandatory (Ex: `alt`)

- **Confusion:**

- Frame scripts or styles by CDATA:

```
<Script language = "JavaScript type =" text / javascript ">  
  <![CDATA [document.write ( "<b> Hello World </ b>");!]]>  
</ Script>
```

HTML vs. XHTML

Rule	Incorrect	Correct
Element names must be lowercase	<code><P>This is a paragraph.</P></code>	<code><p>This is a paragraph.</p></code>
Elements must be properly nested	<code><p>This text is bold.</p></code>	<code><p>This text is bold.</p></code>
All elements must be closed	<code><p>This is the first paragraph. <p>This is the second paragraph.</code>	<code><p>This is the first paragraph.</p> <p>This is the second paragraph.</p></code>
Empty elements must be terminated	<code>This is a line break
</code>	<code>This is a line break
</code>
Attribute names must be lowercase	<code><td ALIGN="right"></code>	<code><td align="right"></code>
Attribute values must be quoted	<code><table width=620></code>	<code><table width="620"></code>
Attributes must have values	<code><option selected></code>	<code><option selected="selected"></code>

XML applications

1. Fields XML applications

2. Standard applications

2.1. Introduction

2.2. Hypertext: XHTML

2.3. Mathematics: MathML

2.4. Synchronized Multimedia: SMIL

2.5. Vector graphics: SVG

2.6. Document Description: RDF

Read more ...

Origin

- Problem of the provision on the web of the scientific and technical documentation containing formulas and mathematical equations
 - **HTML not competent**: No suitable format for the presentation of mathematical expressions
 - use images: we mask the problem.
- MathML designed by the W3C group to include mathematical data simply.
- MathML Version 2.0 has a status of the recommendation of 21 February 2001, whose specifications are available at:
<https://www.w3.org/TR/MathML2/>
- Today browsers support MathML.

LaTeX

- Document mark-up language used for technical and scientific material

```
\documentclass{article}  
\title{A small 3IF LaTeX example}  
\author{John Doe}  
\date{February 2019}  
\begin{document}  
\maketitle  
Hello world!  
E &= \frac{mc^2}{\sqrt{1-\frac{v^2}{c^2}}}  
\end{document}
```

$$E = \frac{mc^2}{\sqrt{1 - \frac{v^2}{c^2}}}$$

MathML:

$$\frac{a + b}{2}$$

- an application of XML for describing mathematical notations and
- captures both its structure and content.
- XML is NOT page layout language.
- **LaTeX** is a mathematical type-setting markup. It does **not** contain information on the type of variable – difficult to detect semantics

```
<math display='block'>
  <mrow>
    <mfrac>
      <mrow>
        <mi>a</mi>
        <mo>+</mo>
        <mi>b</mi>
      </mrow>
      <mn>2</mn>
    </mfrac>
  </mrow>
</math>
```

LaTeX:
`\frac{a+b}{2}`

Example

$$(a + b)^2$$

Presentation	content
<pre><mrow> <msup> <mfenced> <mrow> <mi> a </mi> <mo> + </mo> <mi> b </mi> </mrow> </mfenced> <Min> 2 </Min> </msup> </mrow></pre>	<pre><mrow> <apply> <Power /> <apply> <Plus /> <i> a </i> <i> b </i> </apply> <cn> 2 </cn> </apply> </mrow></pre>

- course MathML and Examples:

<http://www.yoyodesign.org/doc/w3c/mathml2/chapter2.html>

MathML: Presentation

- Description of the presentation of mathematical expressions
 - About 30 elements and 50 attributes
 - Construction by nested boxes
 - Presentation based on visual context

- example:

$$\sum_{k=1}^n x_k = \frac{1}{\sum_{k=1}^n y_k}$$

- **MathML presentation** and **MathML content** can be combined

Elements of the presentation example

- MathML presentation elements:
 - **<Mrow>**: A row material aligned horizontally.
 - Subexpressions in turn can be contained in an item **<Mrow>**.
 - **<Mfenced>**: Fictional symbols of separation
 - braces, brackets and parentheses
 - By default, these are brackets
 - **<E>** for the display of variables a and b,
 - **<Mo>** to report the + operator.
 - **<Msup>**: Write letters above
 - Accepts two arguments in this order: basic expression (a + b) and expression by exposing (2).

MathML: Contents

- Description of the mathematical structure:
 - About 120 elements and attributes 12
 - Application features:
 - `<Apply> function arg1 ... argn </ apply>`
 - One hundred operators and functions:
 - `<Sin/>`, `<Plus/>`, `<Eq />`
 - basic elements:
 - identifiers `<I>` and numbers `<Cn>`
 - scalability:
 - `<Csymbol definitionURL = "..."> ... </ csymbol>`
 - manufacturers:
 - `<Set> elt1 eltn ... </ set>`
 - `<Interval> pt1 pt2 </ interval>` and attribute `closure`
 - `<Vector> elt1 eltn ... </ vector>`
 - `<Matrix>`
 - `<Matrixrow> elt11 elt1n ... </ matrixrow>`
 - `</ Matrix>`

Elements of such content

- Elements of MathML content:
 - **<Apply>**: Apply an operation to an expression.
 - **<Power>**: Exponentiation
 - **<More>**: Addition operator
 - **<Power>** and **<More>** are both applied.
 - The order of children is crucial to the use of the element **<Apply>**:
 - the first child (operator) takes as argument list the remaining children
 - **<I>** to mark the variables a and b,
 - **<Cn>** to mark the number 2.

Tools

- Many implementations:
 - Editors, browsers, viewers, converters, widgets
- See page MathML Software:
 - https://www.w3.org/wiki/Math_Tools
 - <https://www-archive.mozilla.org/projects/mathml/demo/texvsmml.xhtml>

XML applications

1. Fields XML applications

2. Standard applications

2.1. Introduction

2.2. Hypertext: XHTML

2.3. Mathematics: MathML

2.4. Synchronized Multimedia: SMIL

2.5. Vector graphics: SVG

2.6. Document Description: RDF

Read more ...

The graphics on the Web

- Pixel image (gif, png, jpeg, ...)
 - fixed resolution
 - large files
 - difficult search: metadata
 - No interaction: HTML image-maps are complex
- other solutions
 - Formats unsuited to the Web
 - proprietary Tools

graphic needs

- Vector graphic
 - Resolution independent
 - Preserves the original information
 - In the wake of previous work (GKS, PHIGS, PostScript, ...)
- Textual syntax
 - Easy to edit, add links
 - Enables information search
- Based on XML
 - A well-known syntax with existing tools
 - Integration with other Web technologies

SVG: Scalable Vector Graphics

- An XML language for describing 2D graphics containing:
 - Vector shapes
 - Images
 - Of text
- **Structuring:** Object hierarchy, references, style (CSS), geometric transformations.
- **Effects:** Clipping, transparency, filters
- **Dynamic:** Interaction (events, scripts, DOM), entertainment (SMIL)

History

- "SVG is the HTML graphics"
 - Work begins at W3C in 1998
 - Specification steady in 2000, with some implementations
 - W3C Recommendation in September 2001: SVG 1.0
 - Roaming Profiles in January 2003: SVG 1.1
 - Version 1.2 underway
- Recent browsers support SVG

Example



```
<svg id = "Duck" viewBox = "- 0.5 85291166">  
  <Title> Outline of a duck </ title>  
  <g id = "duck">  
    <desc> This is the outline (using b-spline curves)  
    of a single ugly duckling. </ desc>  
    <path d = "M c 0 0 40 48 120 -32 160 -6  
      c 0 0 5 4 10 -3 c 10 -103 50 -83 90 -42  
      c 0 0 20 12 30 7 c -2 12 -18 17 -40 17  
      c -55 -2 -40 25 -20 35 c 30 20 35 65 -30 71  
      c -50 4 -170 4 -200 -79 z "/>  
  </g>  
</svg>
```

Source: I. Hermann

Structure

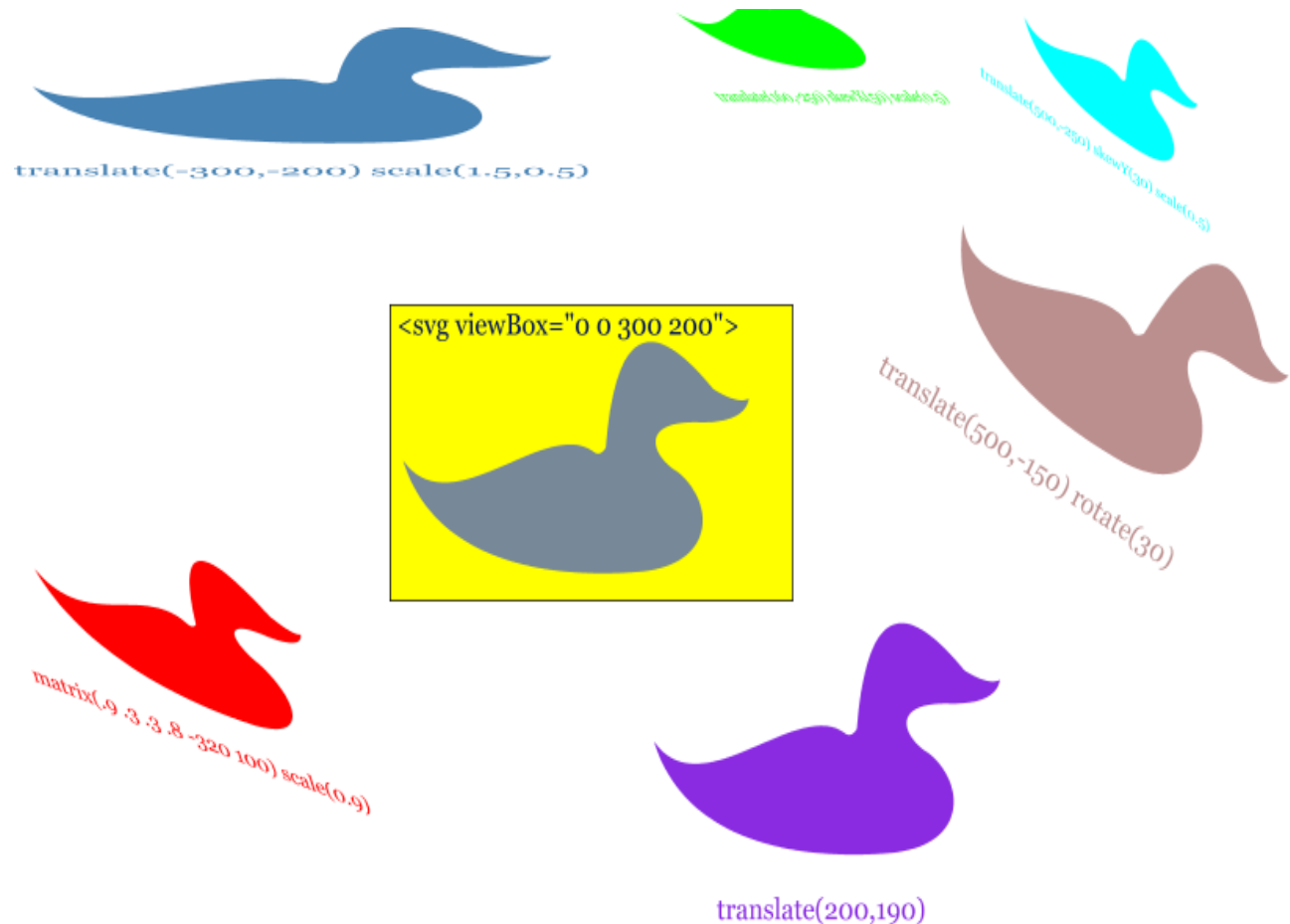
- Metadata: `<Title>` , `<Desc>`, `<Metadata>`
- Hierarchical Structure: groups `<G>`
- Sharing and reuse: `<Symbol>` and `<Use>`
- Hyperlinks: XLink
- Alternatives: `<Switch>`

Geometric shapes

- Path: a sequence of elements
 - Straight lines
 - Curves (Bezier cubic and quadratic)
 - Elliptical arcs
- Open or closed, filled or not
- Simple forms: segment, rectangle, polygon, circle, ellipse, etc.

transformations

- Alterations applied to any SVG element
 - Translation
 - Scaling
 - Rotation
 - Skewing



Text

- A particular element:

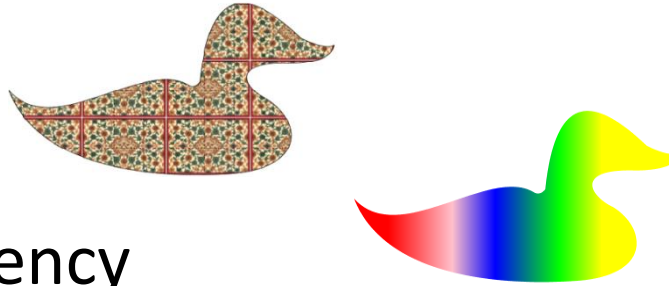
```
<Text x = ".." y = ".."> Blablabla</ text>
```

- Many attributes (mostly from CSS):
 - Font, style, size
 - Color (outline and fill)
 - Anchor
- The text can follow a curve:



other features

- Image inclusion
- Clipping, masking
- Filling gradients
- Opacity / transparency
- Filters
- Color manipulation
- Compositing



Animation

- SVG Animation = dynamically changing attributes of an element
- Animation controlled by a set of animation elements
 - SMIL 2.0 Reuse
- Declarative Syntax, very compact
- Animation made by the client
- Example

<http://www.hongkiat.com/blog/svg-animations/>

- Other svg demo links :

<https://code.google.com/archive/p/svgweb/>

https://www.w3schools.com/graphics/svg_examples.asp

XML applications

1. Fields XML applications

2. Standard applications

2.1. Introduction

2.2. Hypertext: XHTML

2.3. Mathematics: MathML

2.4. Synchronized Multimedia: SMIL

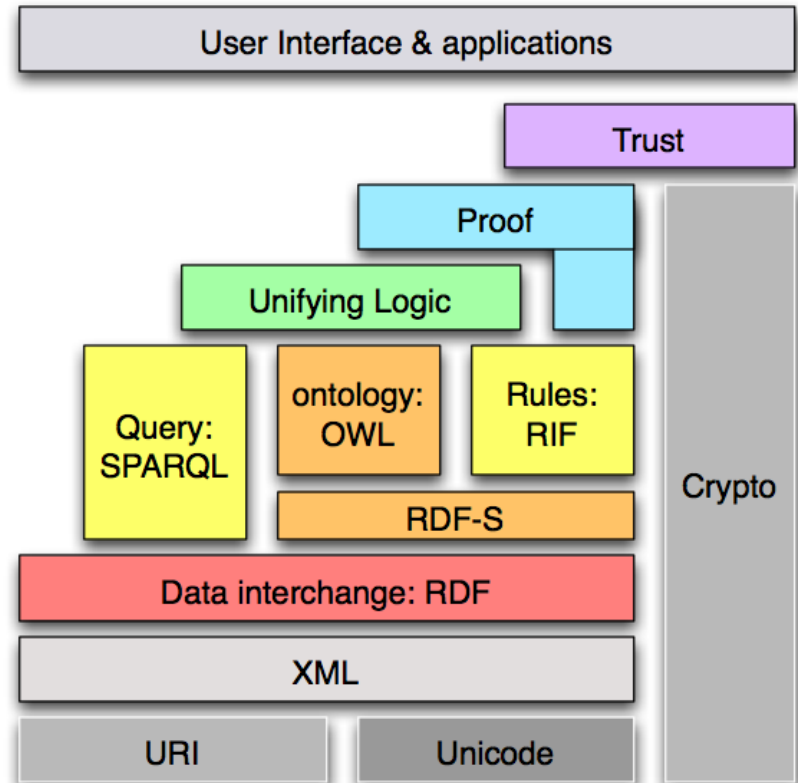
2.5. Vector graphics: SVG

2.6. Document Description: RDF

Read more ...

RDF in the stack of Semantic Web technologies

- RDF stands for:
 - **Resource**: Everything that can have a unique identifier (URI), e.g. pages, places, people, dogs, products...
 - **Description**: attributes, features, and relations of the resources
 - **Framework**: model, languages and syntaxes for these descriptions
- RDF was published as a **W3C recommendation** in 1999.
- RDF was originally introduced as a data model for **metadata**.
- RDF was generalised to cover **knowledge of all kinds**.



What is a triple?

RDF is a general syntax for representing data on the Web.

Every piece of information expressed in RDF is represented as a **triple**:

- **Subject** – a resource, which may be identified with a URI.
- **Predicate** – a URI-identified reused specification of the relationship.
- **Object** – a resource or literal to which the subject is related.

Example: name of a dataset:

<http://publications.europa.eu/resource/authority/file-type/> has a title "File types name authority List".

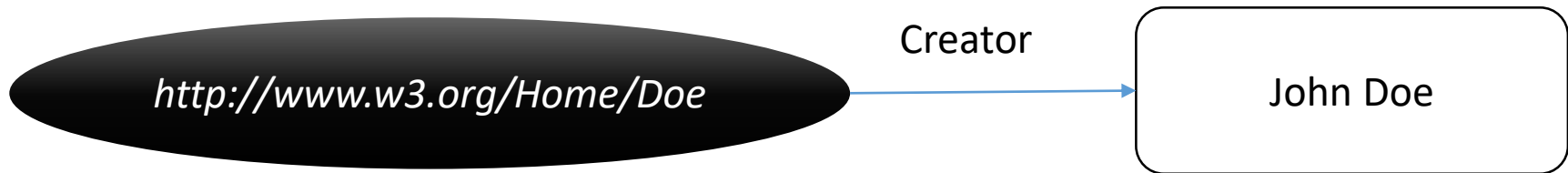
Subject

Predicate

Object

What is a triple?

- RDF statements can be viewed as nodes and arcs diagrams.
- The nodes (ovals) represent resources. The arcs represent named properties. String literals are represented by rectangles.
- *John Doe is the creator of the resource <http://www.w3.org/Home/Doe>.*



```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:s="http://description.org/schema/">
  <Description about="http://www.w3.org/Home/Doe">
    <s:Creator>John Doe</s:Creator>
  </Description>
</RDF>
```

RDF is graph based

- **Graph =**

A collection of
triples



XML applications

1. Fields XML applications

2. Standard applications

2.1. Introduction

2.2. Hypertext: XHTML

2.3. Mathematics: MathML

2.4. Synchronized Multimedia: SMIL

2.5. Vector graphics: SVG

2.6. Document Description: RDF

[Read more ...](#)

Read more ...

- References:

- XHTML: <http://www.w3.org/TR/xhtml11/>
- MathML: <http://www.w3.org/TR/MathML2/>
- SMIL: <https://www.w3.org/TR/SMIL2/>
- SVG: <http://www.w3.org/TR/SVG11/>
- RDF: <https://www.w3.org/TR/rdf11-concepts/>
- Many others...