ADVANCED DATABASE

Semi-structured databases - XML

Dr. NGUYEN Hoang Ha

Email: nguyen-hoang.ha@usth.edu.vn

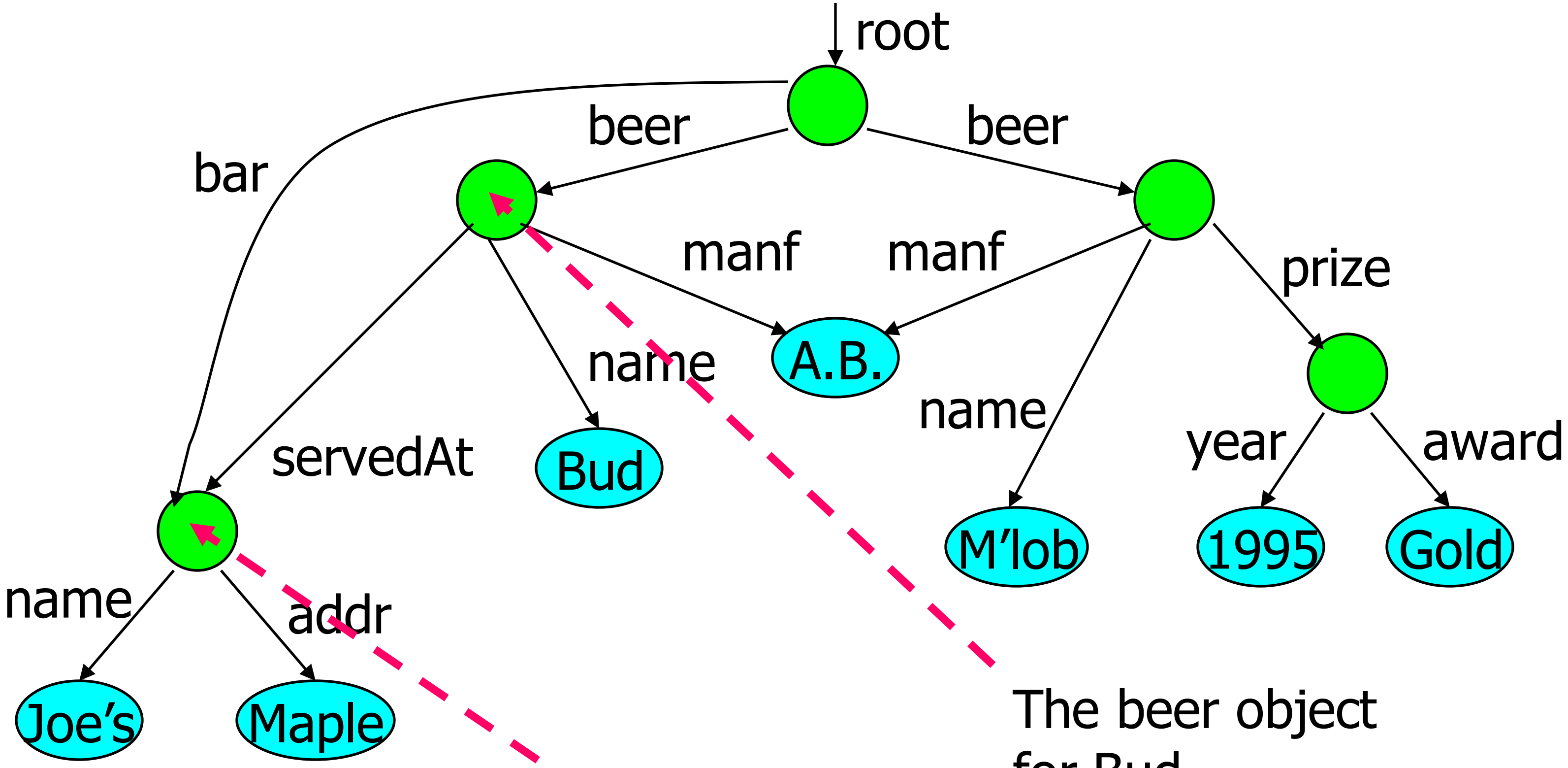
Agenda

- Semis-structured Data
- Background of XML
- XML syntax
- XML validation
  - DTD
  - XSD
- XQuery

Semis-tructured Data

- Motivation:
  - Flexible representation of data: often, data comes from multiple sources with differences in notation, meaning, etc.
  - Sharing of documents among systems and databases.
- Semi-structured data
  - Not conform with formal structure (tables)
  - Another data model, based on trees.

Example: Data Graph



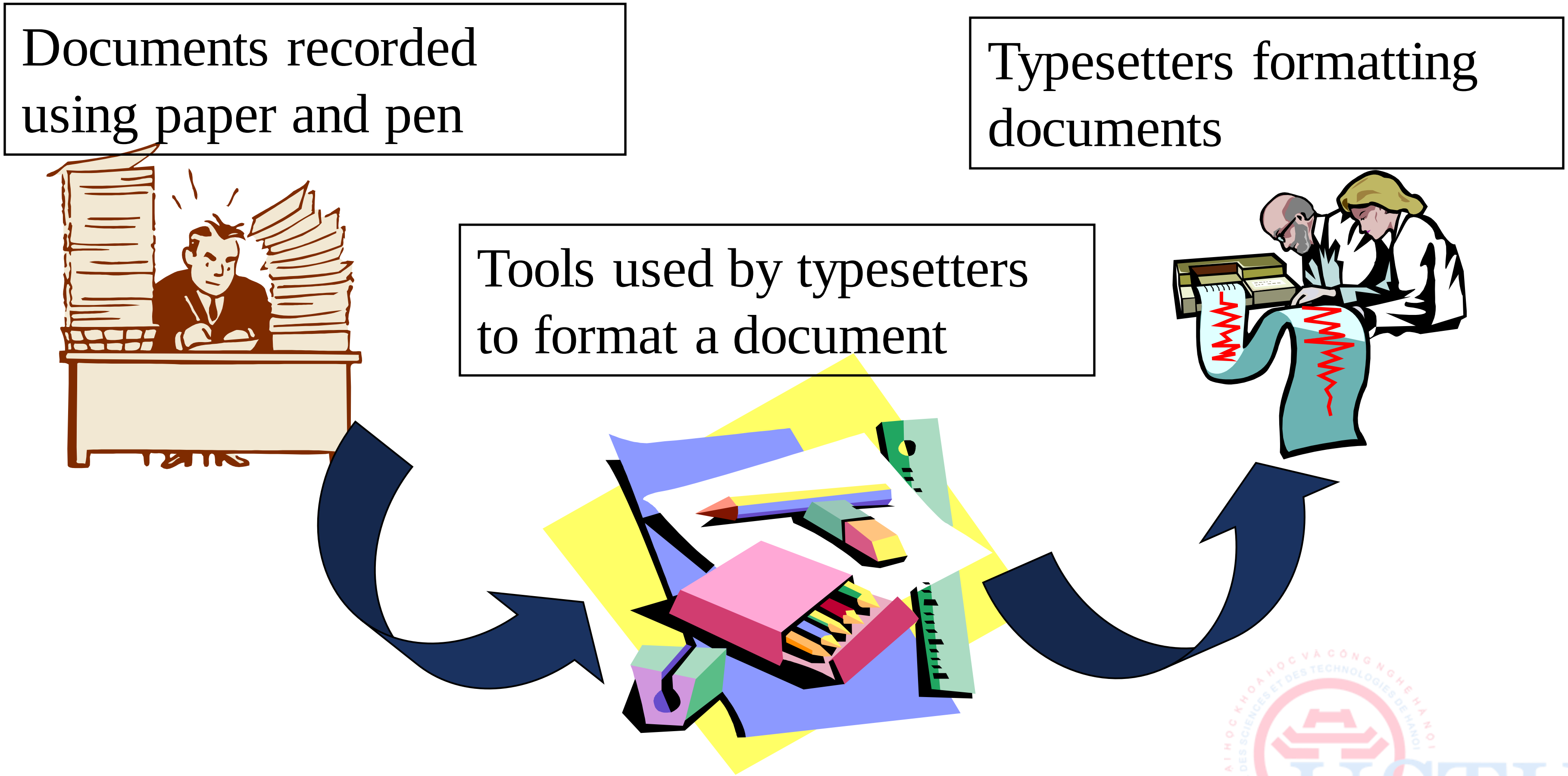
The beer object
for Bud

The bar object
for Joe's Bar

Semi-structured Data Representation

- Nodes = objects.
  - Atomic values at leaf nodes (nodes with no arc out).
  - Interior nodes have one or more arcs out
    - Labels on arcs (attributes, relationships).
- Flexibility: no restriction on:
  - Labels out of a node.
  - Number of successors with a given label.

History of Markup



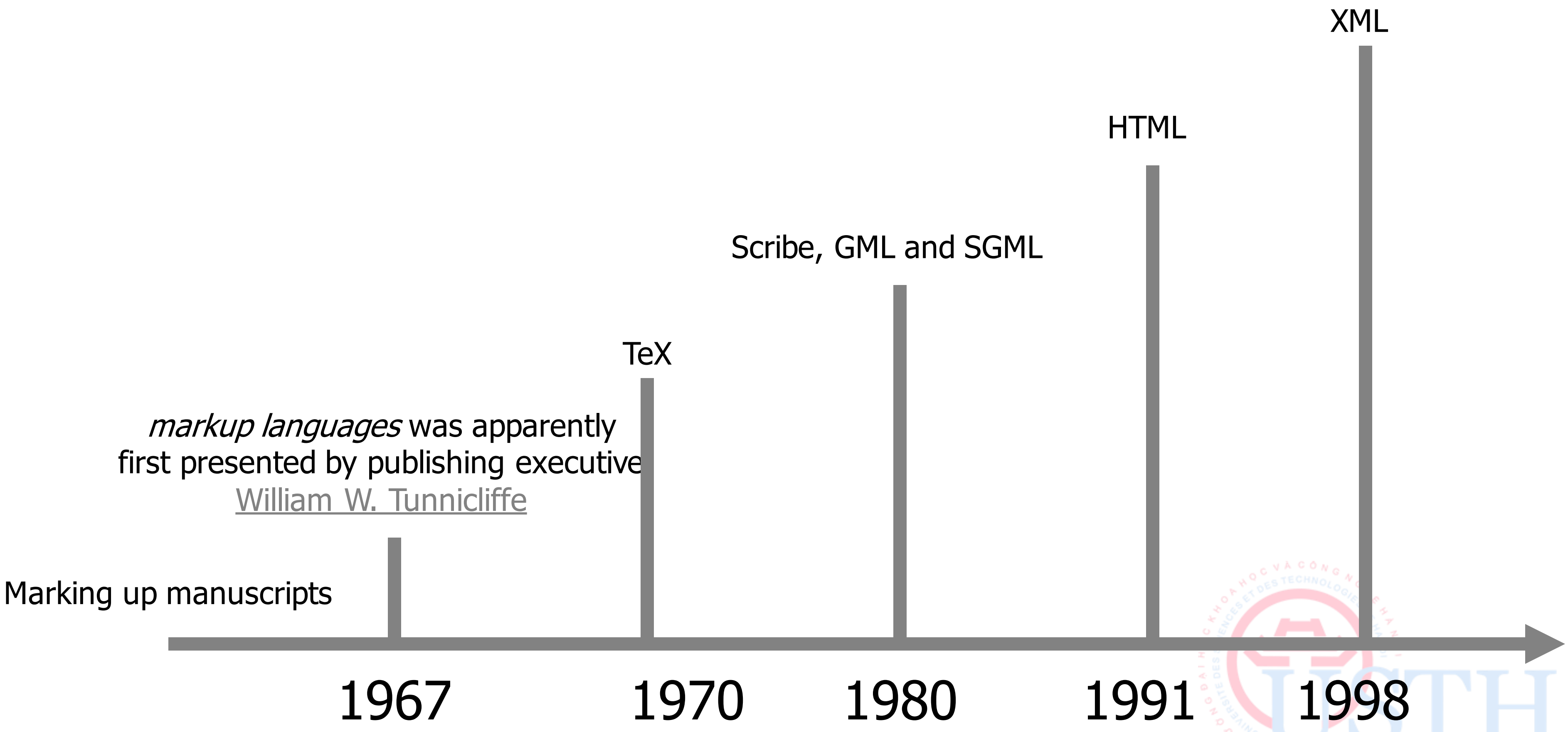
Documents recorded using paper and pen

Tools used by typesetters to format a document

Typesetters formatting documents

History of Markup

A Markup language defines the rules that help to add meaning to the content and structure of documents



Marking up manuscripts

markup languages was apparently first presented by publishing executive William W. Tunnicliffe

TeX

Scribe, GML and SGML

HTML

XML

1967

1970

1980

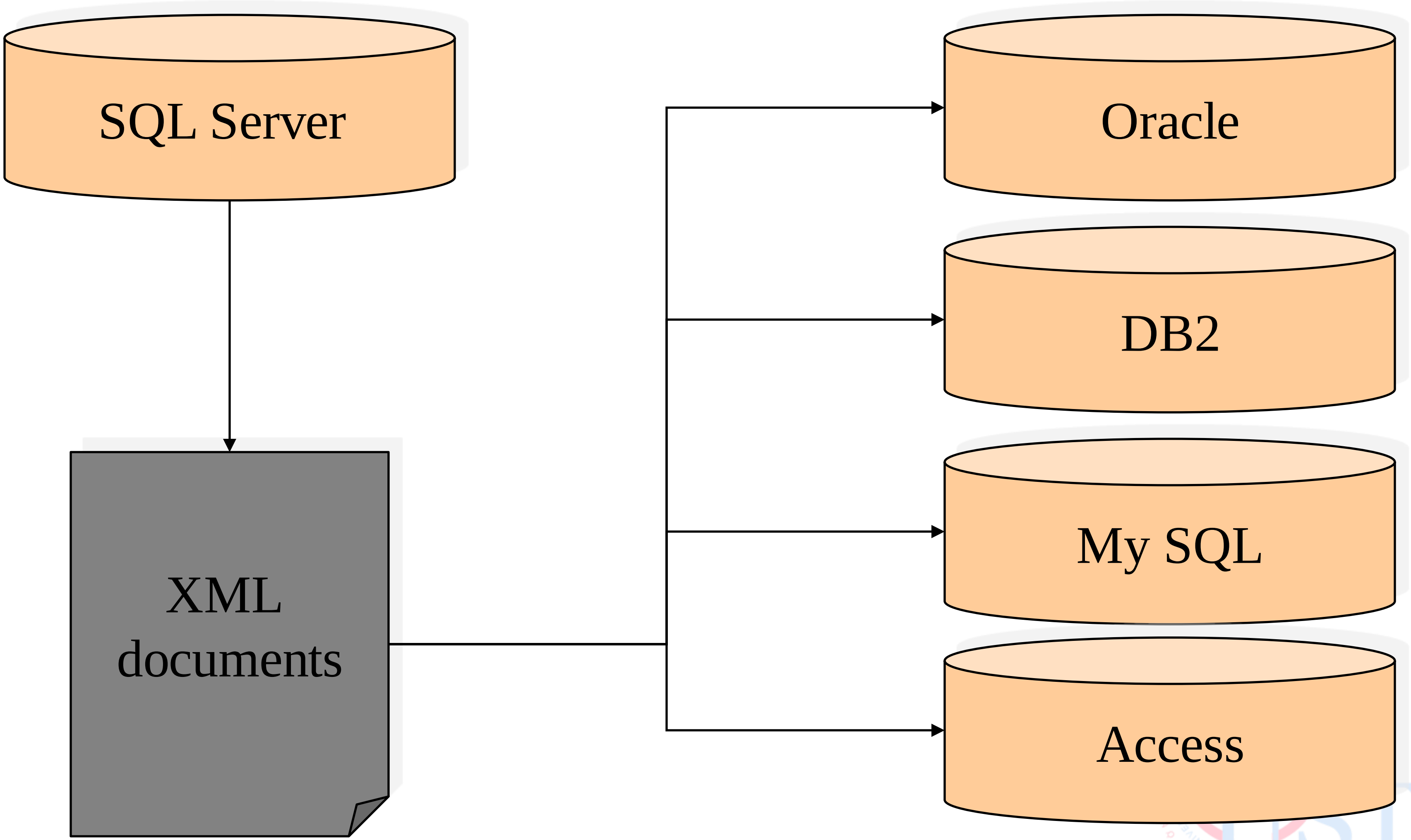
1991

1998

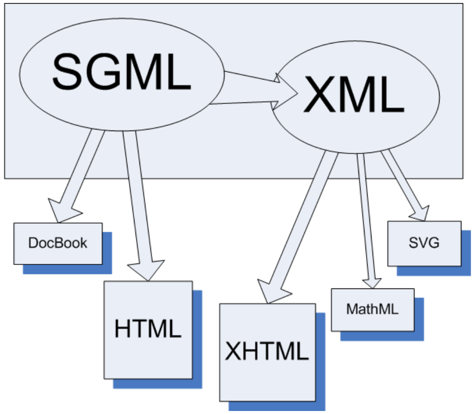
Extensible Markup Language - XML

- A smaller version of SGML.
- For data description, not for presentation like HTML
  - Data in XML can be displayed in different ways
- More flexible than HTML.
  - Users can define their tags understandable for others (people, programs)

XML as a data integration mean



Background for XML

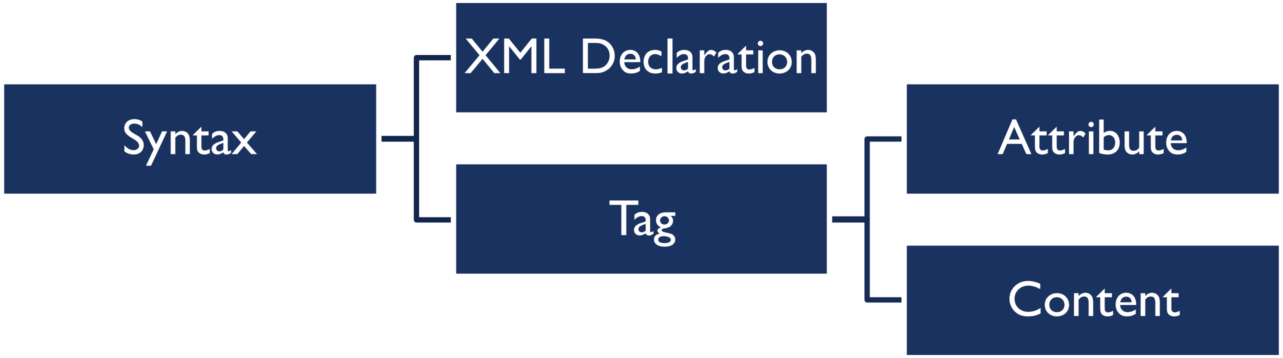


- An Extensible Markup Language (XML) document describes the *structure of data*
- XML and HTML have a similar syntax … both derived from SGML
- XML has no mechanism to specify the format for presenting data to the user
- An XML document resides in its own file with an '.xml' extension

Example of XML

```
<?xml version="1.0" standalone = "yes" ?>
<BusinessCard>
    <Name>Joe Marini</Name>
    <phone type="mobile">(415) 555-4567</phone>
    <phone type="work">(800) 555-9876</phone>
    <phone type="fax">(510) 555-1234</phone>
    <email>joe@joe.com</email>
</BusinessCard>
```

XML syntax



XML declaration

```
<?xml version = "1.0" standalone= "no" encoding = "UTF-8"?>
```

- At very beginning, not even white space before
- XML declaration is optional
- <?xml?> must be in lower case
- Standalone: yes or no → whether this documents has rules (DTD, XSD)

Tags

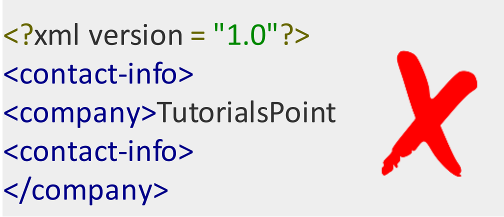
- Tags (a.k.a elements) are nodes
- Format
  - Starting tag: <_____>
  - Ending tag: </_____>
- Starting and ending tags must match
- Tag name:
  - Case sensitive
  - Starts with underscore or letter
  - Followed by letters, digits, underscore, hyphens, dots
  - Cannot use "xml" in any combination
- Tag content: string between tags: <email>joe@joe.com</email>
- Short format for empty tag: <____/>. E.g: <email></email> ≈ </email>

Tags

- Every XML document has one root tag

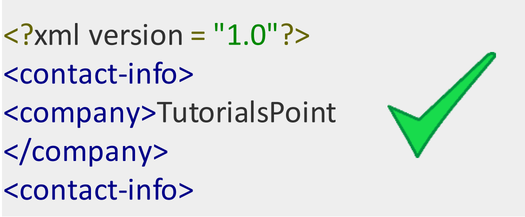
```
<?xml version = "1.0" standalone= "no" encoding = "UTF-8"?>
<BusinessCard>
</BusinessCard>
```

- Nesting tag
  - Tag can contains child tags
  - Children must not overlap



```
<?xml version = "1.0"?>
<contact-info>
<company>TutorialsPoint
<contact-info>
</company>
```

✗



```
<?xml version = "1.0"?>
<contact-info>
<company>TutorialsPoint
</company>
<contact-info>
```

✓

Attribute

```
<phone type="mobile">(415) 555-4567</phone>
```

- Similar to Attribute of HTML
- Specify on opening only
- Name
  - Starts with underscore or letter
  - Followed by letters, digits, underscore, hyphens, dots
- Value
  - Between double quotes
  - Can be converted into appropriate data type in programming languages
- Not duplicate name

Namespace

- Motivation: same tag (attribute) name but with different meanings in different contexts

→ use namespace to uniquely identify them

- Syntax
  - The Namespace starts with the keyword **xmlns**.
  - The **URL** is the Namespace identifier.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<cont:contact xmlns:cont = "www.usth.edu.vn">
 <cont:name>Tanmay Patil</cont:name> <cont:company>Adventure
Works</cont:company>
<cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Comments

- Comments embed human-readable infor
- Start with <!-- and end with -->
- Appear anywhere after declaration, but not nest to other comments

```
<?xml version = "1.0" standalone= "no" encoding = "UTF-8"?>
<!--- Business card information---->
<BusinessCard>
      <Name>Joe Marini</Name>
      <phone type="mobile">(415) 555-4567</phone>
      <phone type="work">(800) 555-9876</phone>
      <phone type="fax">(510) 555-1234</phone>
      <email>joe@joe.com</email>
</BusinessCard>
```

Character Data Section (CDATA)

- Used to contain data with special characters
  - <,>,&,
- Define a block ignored by XML parser
- <![CDATA[________]]>

```
<script>
<![CDATA[ <message> Welcome to TutorialsPoint </message> ]] >
 </script >
```

- Nesting is not allowed

Special Characters

- Some characters are reserved for XML syntax
- There are replacements

Not Allowed Character	Replacement	Character Description
<	&lt;	less than
>	&gt;	greater than
&	&amp;	ampersand
'	&apos;	apostrophe
"	&quot;	quotation mark

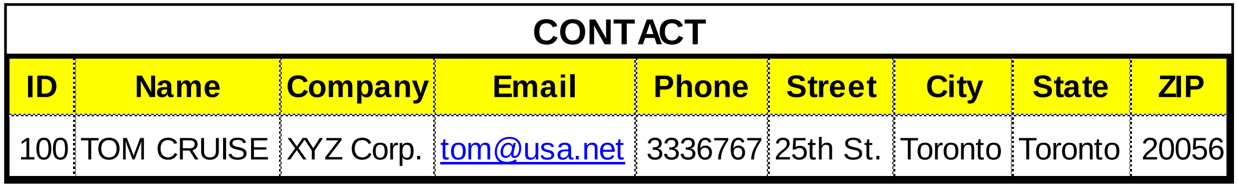
The Basic Rules

- XML is case sensitive
- All start tags must have end tags
- Elements must be properly nested
- XML declaration is the first statement
- Every document must contain a root element
- Attribute values must have quotation marks
- Certain characters are reserved for parsing

XML formatting guidelines

- Add an indent for every new level of tags
- Tags do not contain other tags can have start and end tags on the same line
- Tags containing other tags should be on their own lines

Exercise: Convert a Table into XML



CONTACT

ID	Name	Company	Email	Phone	Street	City	State	ZIP
100	TOM CRUISE	XYZ Corp.	tom@usa.net	3336767	25th St.	Toronto	Toronto	20056

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<CONTACT>
    <Id>        100       </Id>
    <Name> TOM CRUISE     </Name>
    <Company> XYZ Corp.   </Company>
    <Email>   tom@usa.net </Email>
    <Phone>   3336767     </Phone>
    <Street>  25th St.    </Street>
    <City>    Toronto     </City>
    <State>   Toronto     </State>
    <ZIP>     20056       </ZIP>
</CONTACT>
```

XML VERIFICATION

Valid vs. Well Formed

- An XML document following XML syntax is called "Valid"
- A valid XML document conforming to structure rule is a "Well Formed" XML document
- 02 ways of structure rule definition:
  - DTD (Doctype definition)
    - Simple but not powerful
    - Written in a syntax different from XML
  - XML Schema
    - More powerful and flexible than DTD
    - Written in XML syntax

DTD example

```
<?xml version="1.0" standalone = "yes" ?>
<!DOCTYPE BusinessCard [
    <!ELEMENT BusinessCard (Name, phone+, email?)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT phone (#PCDATA)>
    <!ATTLIST phone type (mobile | fax | work | home) #REQUIRED>
    <!ELEMENT email (#PCDATA)>
]>
<BusinessCard>
    <Name>Joe Marini</Name>
    <phone type="mobile">(415) 555-4567</phone>
    <phone type="work">(800) 555-9876</phone>
    <phone type="fax">(510) 555-1234</phone>
    <email>joe@joe.com</email>
</BusinessCard>
```

Online Validator at http://xmlvalidator.new-studio.org/

DTD Structure

```
<!DOCTYPE <root tag> [
    <!ELEMENT <name> (<components>) >
    . . . more elements . . .
]>
```

- The description of an element consists of its name (tag), and a parenthesized description of any nested tags.
  - Includes order of subtags and their cardinality.
- Leaves (text elements) have #PCDATA (*Parsed Character DATA*) in place of nested tags.

Element Descriptions

- Subtags must appear in order shown.
- A tag may be followed by a symbol to indicate its multiplicity.
  - * = zero or more.
  - + = one or more.
  - ? = zero or one.
- Symbol | can connect alternative sequences of tags.

Attributes

- Opening tags in XML can have *attributes*.
- In a DTD,

`<!ATTLIST` *E* `...>`

declares attributes for element *E*, along with its datatype.

Example

```
<!ATTLIST phone type (mobile | fax | work | home) #
  REQUIRED>
```

```
 <phone type="mobile">(415) 555-4567</phone>
    <phone type="work">(800) 555-9876</phone>
    <phone type="fax">(510) 555-1234</phone>
```

The same data in different format

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

<!DOCTYPE BusinessCard [
        <!ATTLIST BusinessCard name CDATA #REQUIRED>
        <!ATTLIST BusinessCard email CDATA #REQUIRED>
        <!ELEMENT BusinessCard (phone+)>
        <!ELEMENT phone (#PCDATA)>
        <!ATTLIST phone type (mobile | fax | work | home) #REQUIRED>
        <!ELEMENT email (#PCDATA) >
]>

<BusinessCard name = "Joe Marini" email = "joe@joe.com">
        <phone type="mobile">(415) 555-4567</phone>
        <phone type="work">(800) 555-9876</phone>
        <phone type="fax">(510) 555-1234</phone>
        <phone type="home">(425) 555-8989</phone>
</BusinessCard>
```

Use of DTD’s

1. Set standalone = “no”.
2. Either:
   a) Include the DTD as a preamble of the XML document, or
   b) Follow DOCTYPE and the <root tag> by SYSTEM and a path to the file where the DTD can be found.

Outer DTD file

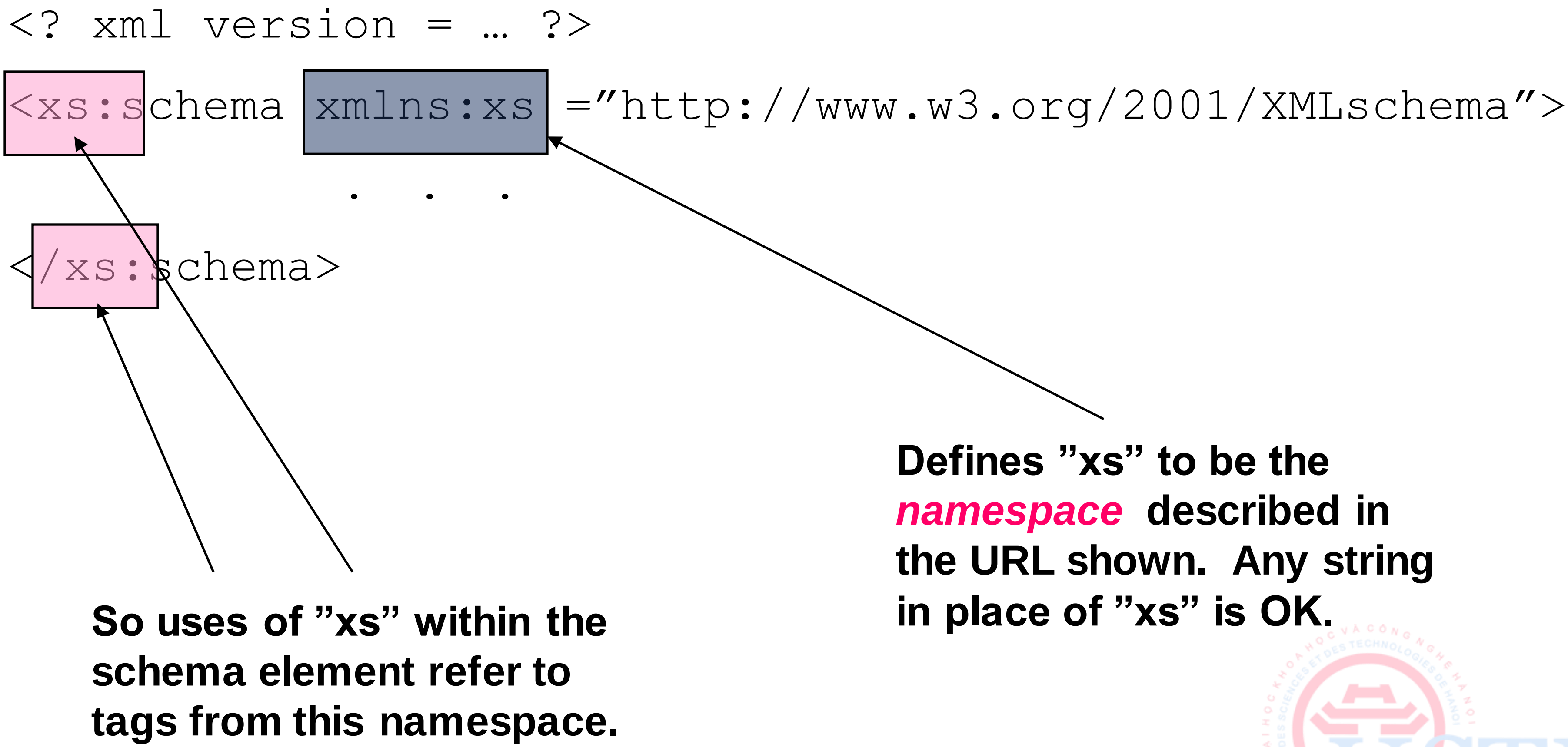
Assumse
“BusinessCard.dt
is the outer dtd file

```
<?xml version="1.0" standalone = ”no" ?>
<!DOCTYPE BusinessCard SYSTEM “BusinessCard.dtd”>
<BusinessCard>
    <Name>Joe Marini</Name>
    <phone type="mobile">(415) 555-4567</phone>
    <phone type="work">(800) 555-9876</phone>
    <phone type="fax">(510) 555-1234</phone>
    <email>joe@joe.com</email>
</BusinessCard>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="BusinessCard">
  <xsd:complexType  mixed="true">
   <xsd:sequence>
    <xsd:element name="Name" type="xsd:string"/>
    <xsd:element name="phone" maxOccurs="unbounded">
     <xsd:complexType mixed="true">
      <xsd:attribute  name="type" use="required">
       <xsd:simpleType>
        <xsd:restriction base="xsd:string">
         <xsd:enumeration value="mobile"/>
         <xsd:enumeration value="fax"/>
         <xsd:enumeration value="work"/>
         <xsd:enumeration value="home"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="email" type="xsd:string" minOccurs="0" />
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

https://www.liquid-technologies.com/online-xsd-validator

Structure of an XML-Schema Document



```
<? xml version = … ?>
<xs:schema xmlns:xs =”http://www.w3.org/2001/XMLschema”>
              . . .
</xs:schema>
```

Defines ”xs” to be the *namespace* described in the URL shown. Any string in place of ”xs” is OK.

So uses of ”xs” within the schema element refer to tags from this namespace.

The xs:element Element

Has attributes:

- name = the tag-name of the element being defined.
- type = the data type of the element.
  - Could be an XML-Schema type, e.g., xs:string.
  - Or the name of a type defined in the document itself
- maxOccurs, minOccurs: restrict number of instances

Example

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

Complex Types

- To describe elements that consist of sub elements, we use xs:complexType.
  - Attribute name gives a name to the type.
- Typical subelement of a complex type is xs:sequence, which itself has a sequence of xs:element subelements.
  - Use minOccurs and maxOccurs attributes to control the number of occurrences of an xs:element.

```
<xsd:element name="BusinessCard">
    <xsd:complexType>
```

xs:attribute

- xs:attribute elements can be used within a complex type to indicate attributes of elements of that type.
- attributes of xs:attribute:
  - name and type as for xs.element.
  - use = ”required” or ”optional”.

XQUERY

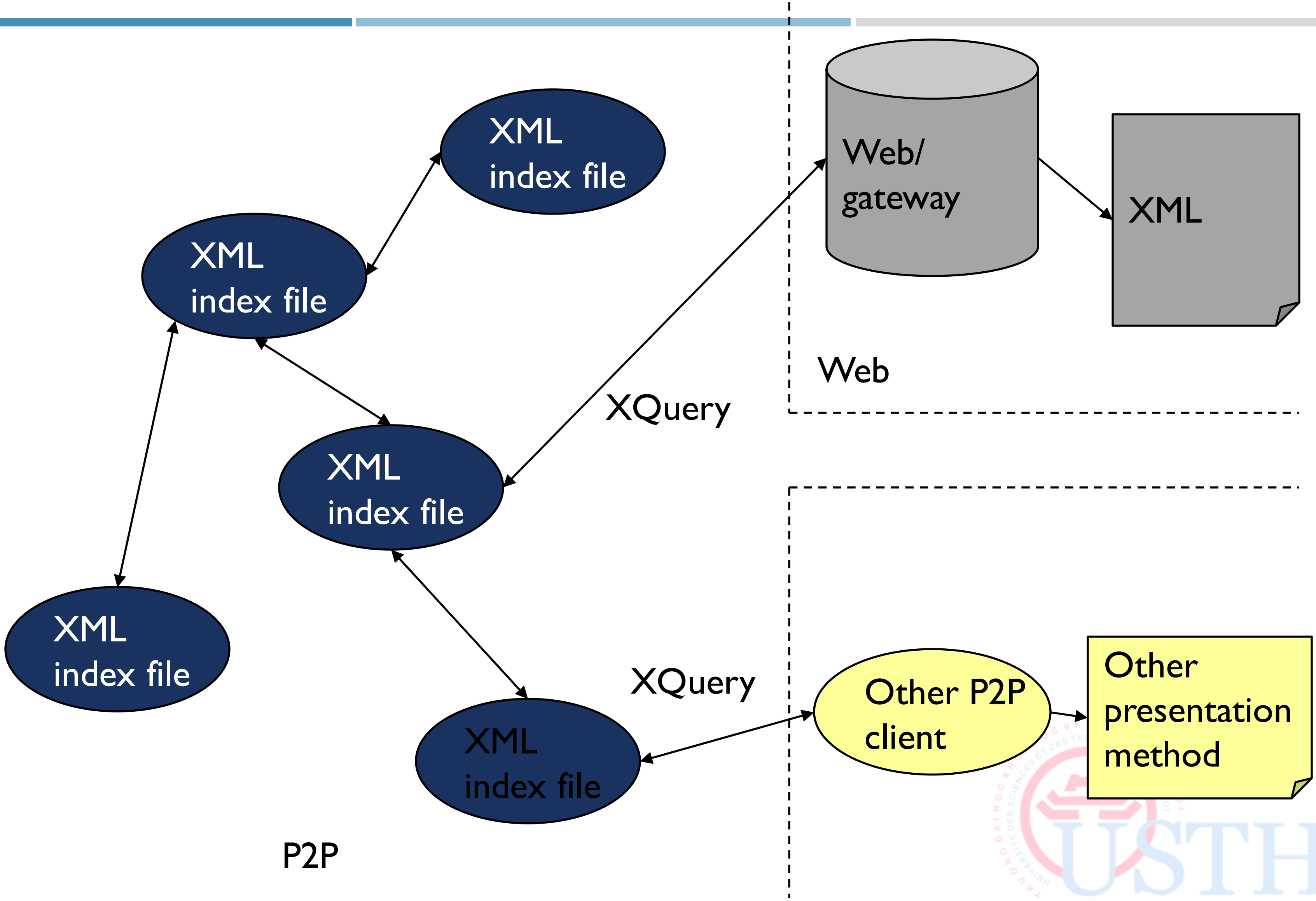
Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML
  - FLWOR (“Flower”) Expressions

```
FOR ...
LET...
WHERE...
ORDER BY …
RETURN...
```

Test XPath and Xquery at http://www.xpathtester.com/xquery

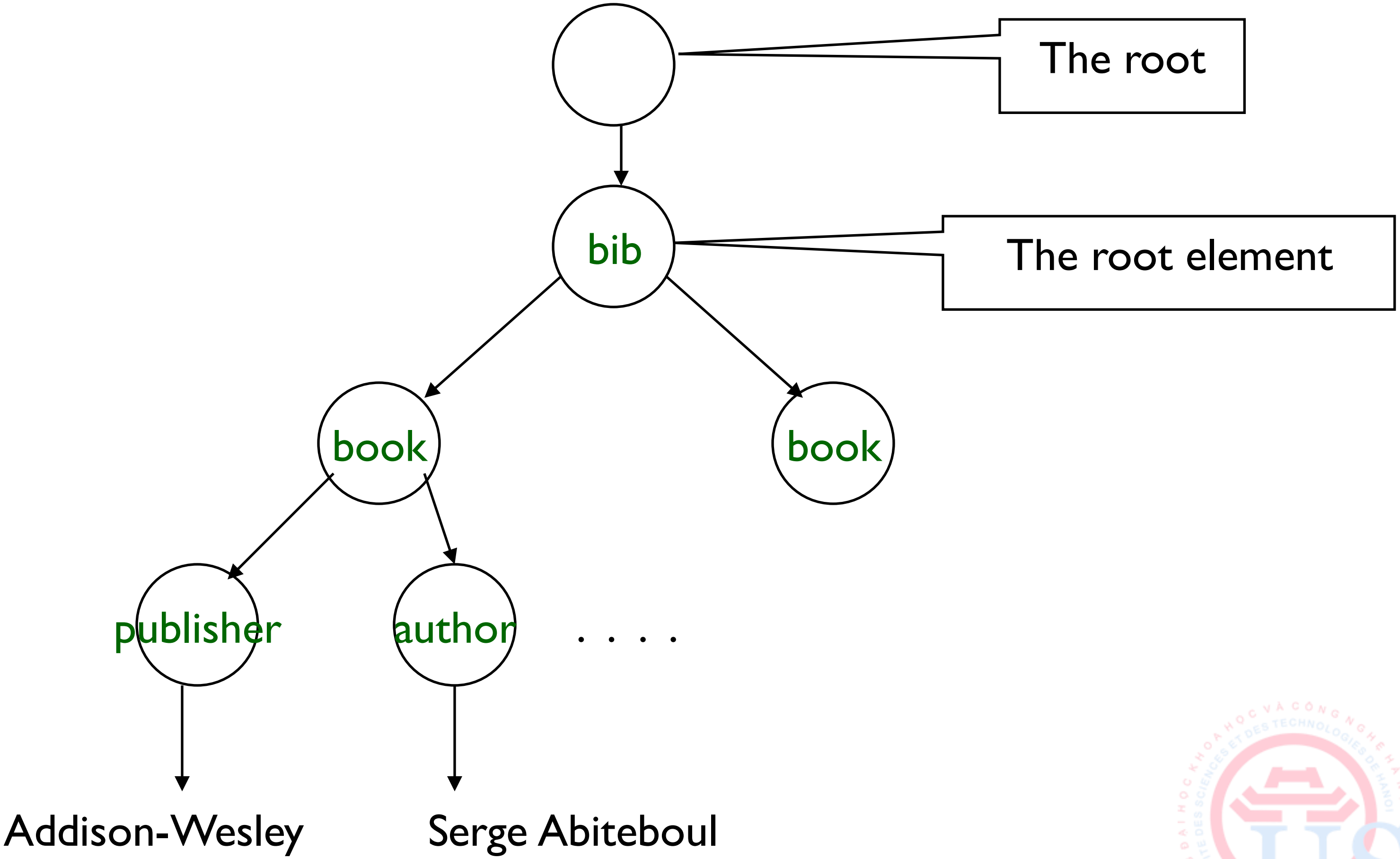
Possible Application



Sample for Xquery

```xml
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author>
      <first-name> Rick </first-name>
      <last-name> Hull </last-name>
    </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
```

Data Model for XPath



Selecting Nodes

Expression	Description
nodename	Get all nodes with nodename
/	Root node
//nodename	Get all nodes in document with nodename
.	Current node
..	Parent node
@attName	Get attribute names

XPath: Simple Expressions

/bib/book/author

```
<?xml version="1.0" encoding="UTF-8"?>
<author> Serge Abiteboul </author>
<author>
      <first-name> Rick </first-name>
      <last-name> Hull </last-name>
    </author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>
```

/bib/book[year<1996]

/bib/paper/year

Result: empty   (there were no papers)

Xpath: Attribute Nodes

```
/bib/book/@price
```

Result: “55”

@price means that price is has to be an attribute

Xpath: Text Nodes

```
/bib/book/author/text()
```

Result: Serge Abiteboul
Jeffrey D. Ullman

Rick Hull doesn't appear because he has firstname, lastname

Functions in XPath:

- text() → text value
- node() → matches any node
- name() → returns the name of the current tag

XQuery

- **Syntax**
  - **For -** selects a sequence of nodes
  - **Let -** binds a sequence to a variable
  - **Where -** filters the nodes
  - **Order by -** sorts the nodes
  - **Return** - what to return (gets evaluated once for every node)
- Example: Find all book titles published after 1995:

```
FOR $x IN /bib/book

WHERE $x/year > 1995

RETURN { $x/title }
```

Result:

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```

XQuery

Same as before, but eliminate duplicates:

```
FOR $x IN bib/book[title/text() = “Database Theory”]/author
       $y IN distinct(bib/book[author/text() = $x/text()]/title)

RETURN <answer> { $y/text() } </answer>
```

distinct = a function
that eliminates duplicates

E.g. FLOWR query filtering by attribute

```
for $b in /bib/book

let $title := $b/title

where $b/@price < 100

order by $title

return $title
```

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

<title> Foundations of Databases </title>

<title> Principles of Database and Knowledge Base Systems </title>
```