XML Overview

COMP9319

Raymond Wong

About XML

- XML is just a markup language defined by W3C (officially in Feb 98)
- It’s a simplified version of SGML

$$\mathrm{HTML4.0} \in \mathrm{XML} \subset \mathrm{SGML}$$

- HTML for presentation markup,
- XML for content markup

Semistructured Data / XML

- Semistructured =>
  - loosely structured (no restrictions on tags & nesting relationships)
  - no schema required
- XML
  - under the “semistructured” umbrella
  - self-describing
  - the standard for information representation & exchange

Storage format vs presentation format - The power of markup

Traditional Database or Spreadsheet

Raymond, Wong, wong, 5932, John, Smith, jsmith, 1234, ...

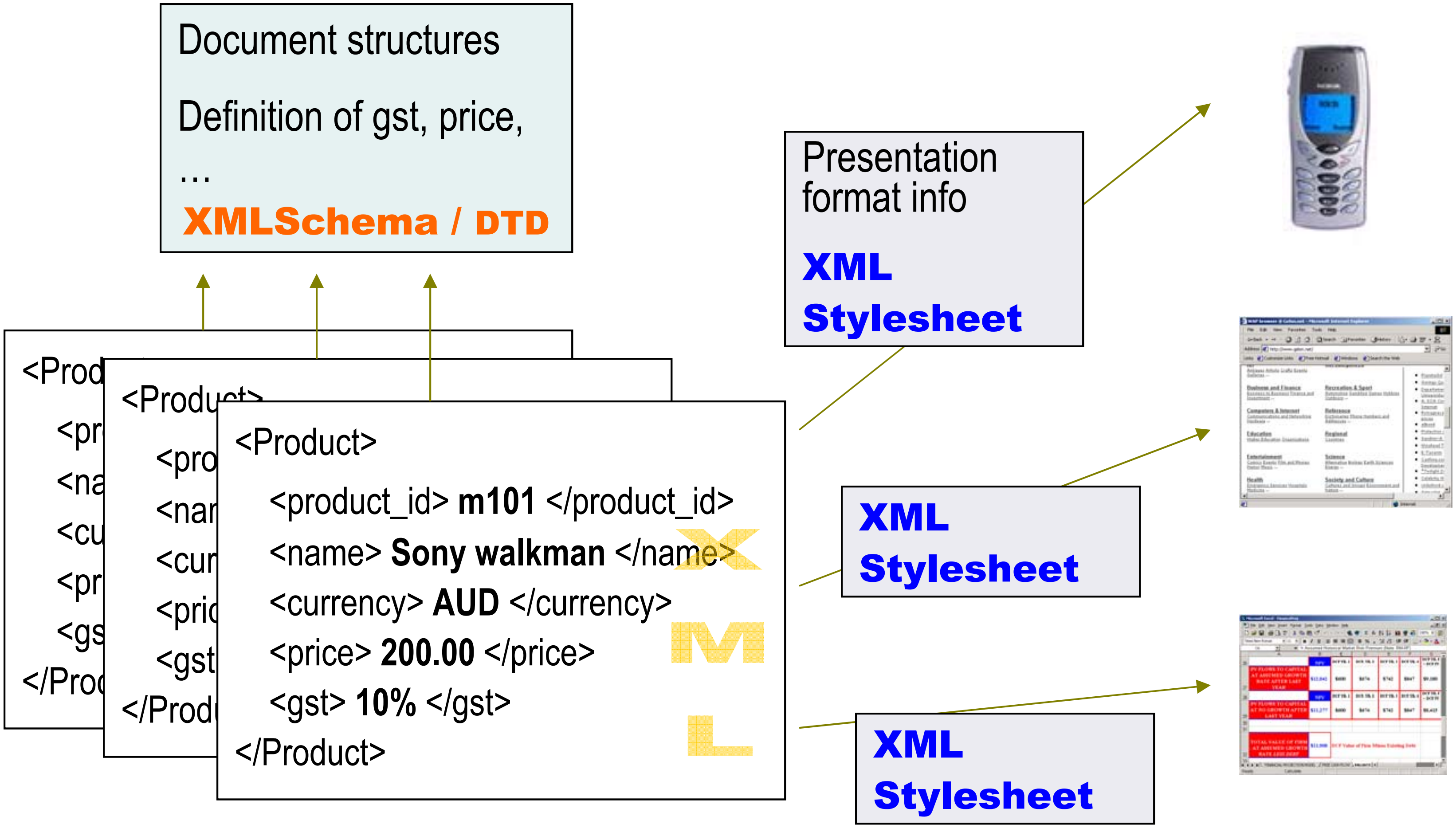
HTML

```
<br>
<font size=1 color="ff003a">
<ul>
  <li> <b> Raymond Wong </b> </li>
  <li> Login: wong </li>
  <li> Phone: <i> x5932 </i> </li>
</ul>
</font>
```

XML

```
<Staff>
    <Name>
        <FirstName> Raymond </FirstName>
        <LastName>  Wong </LastName>
    </Name>
    <Login> wong </Login>
    <Ext> 5932 </Ext>
</Staff>
```

The Family of XML Technologies



Document structures

Definition of gst, price, …

XMLSchema / DTD

```
<Prod
  <pr
  <na
  <cu
  <pr
  <gs
</Proc
```

```
<Product>
  <pro
  <nar
  <cur
  <pric
  <gst
</Produ
```

```
<Product>
  <product_id> m101 </product_id>
  <name> Sony walkman </name>
  <currency> AUD </currency>
  <price> 200.00 </price>
  <gst> 10% </gst>
</Product>
```

Presentation format info

XML Stylesheet

XML Stylesheet

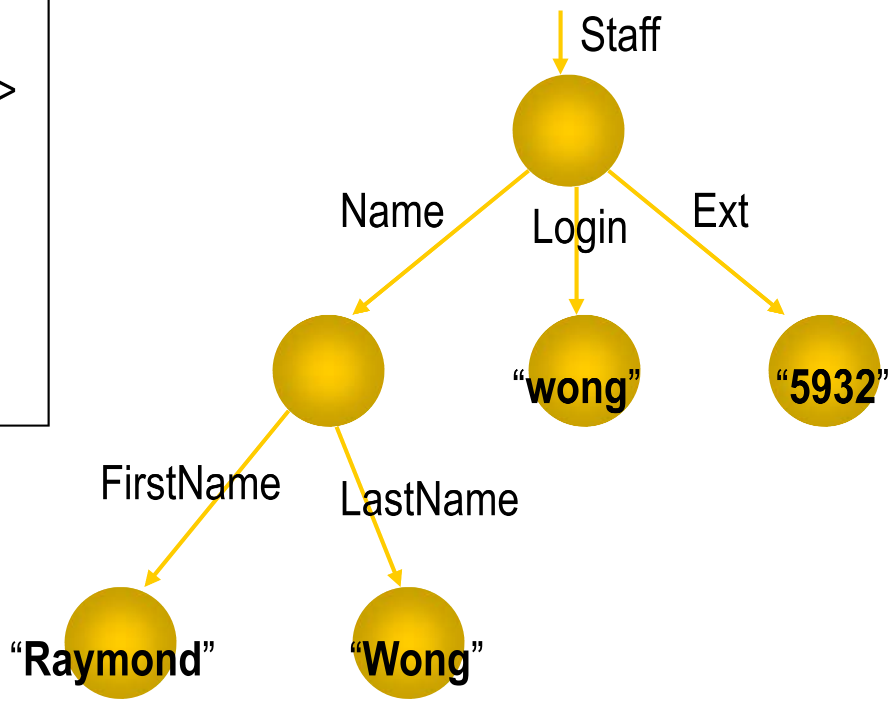
XML Stylesheet

Why need to query XML data

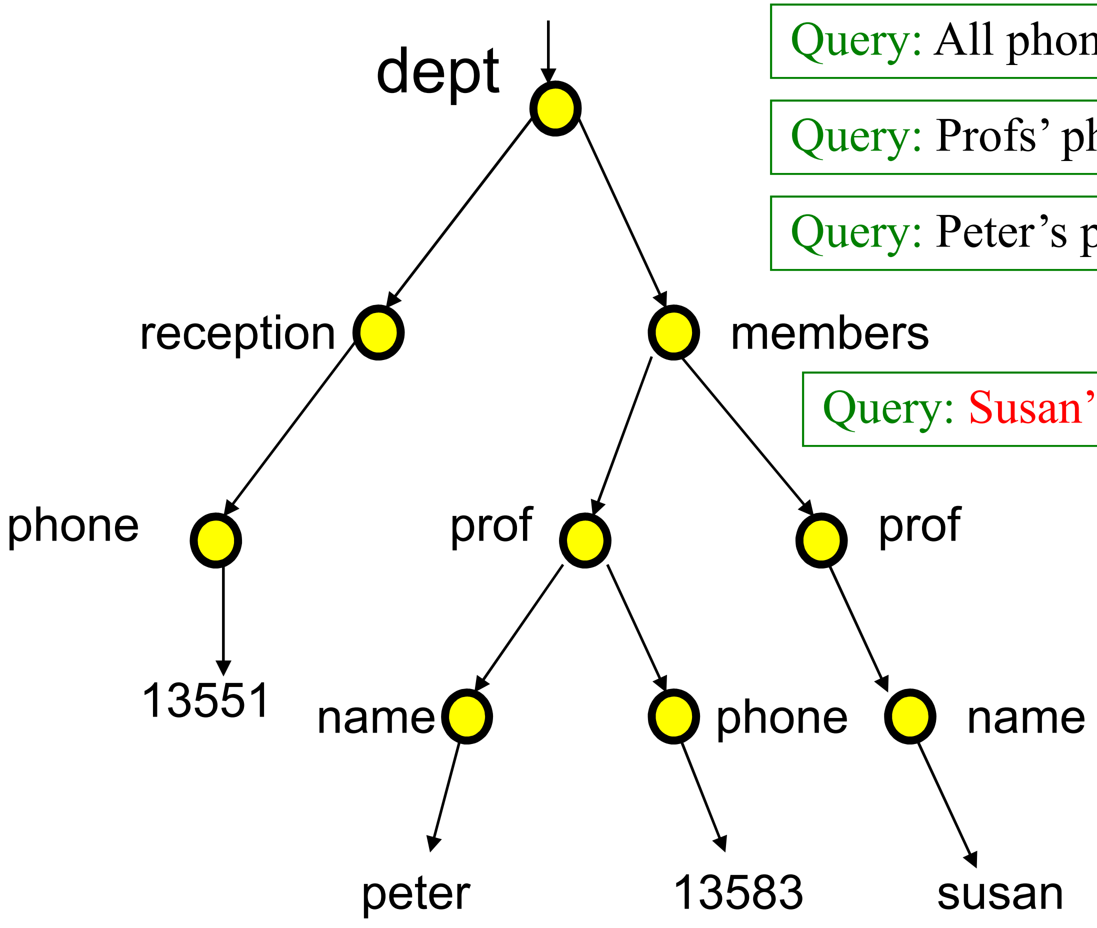
- To extract data from large XML docs
- To exchange data (data- or query-shipping)
- To exchange data beteen different user communities or ontologies or schemas
- To integrate data from multiple XML sources

XML data file can be modeled in a tree form

```
<Staff>
    <Name>
        <FirstName> Raymond </FirstName>
        <LastName>  Wong </LastName>
    </Name>
    <Login> wong </Login>
    <Ext> 5932 </Ext>
</Staff>
```



Answering queries requiring navigation of the data tree



Query: All phones ?

Query: Profs' phones ?

Query: Peter's phone ?

Query: Susan's phone ???

Example query in XPath

```
Dining/Restaurant/(Name | @Name)
```

```
<Name>The Bamboo Restaurant</Name>
<Name>Chen's Seafood Restaurant</Name>
<xql:attribute Name="Thai Palace" />
<Name>Nice Noodles</Name>
```

XML Terminology

- tags: book, title, author, …
- start tag: <book>, end tag: </book>
- elements: <book>…</book>,<author>…</author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single *root element*
- *well formed* XML document: if it has matching tags

More XML: Attributes

```
<book price = “55” currency = “USD”>
   <title> Foundations of Databases </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
</book>
```

More XML: Oids and References

```
<person id=“o555”>
   <name> Jane </name>
</person>
<person id=“o456”>
   <name> Mary </name>
   <children idref=“o123 o555”/>
</person>
<person id=“o123” mother=“o456”>
   <name>John</name>
</person>
```

More XML: CDATA Section

- Syntax: <![CDATA[.....any text here...]]>

- Example:

```
<example>
  <![CDATA[ some text here </notAtag> <> ]]>
</example>
```

More XML: Entity References

- Syntax: &entityname;
- Example:
  <element> this is less than &lt; </element>
- Some entities:

&lt;	<
&gt;	>
&amp;	&
&apos;	‘
&quot;	“
&#38;	Unicode char

More XML: Processing Instructions

- Syntax: <?target argument?>

- Example:

```
<product>
  <name> Alarm Clock </name>
  <?ringBell 20?>
  <price> 19.99 </price>
</product>
```

More XML: Comments

- ⌘Syntax <!-- .... Comment text... -->

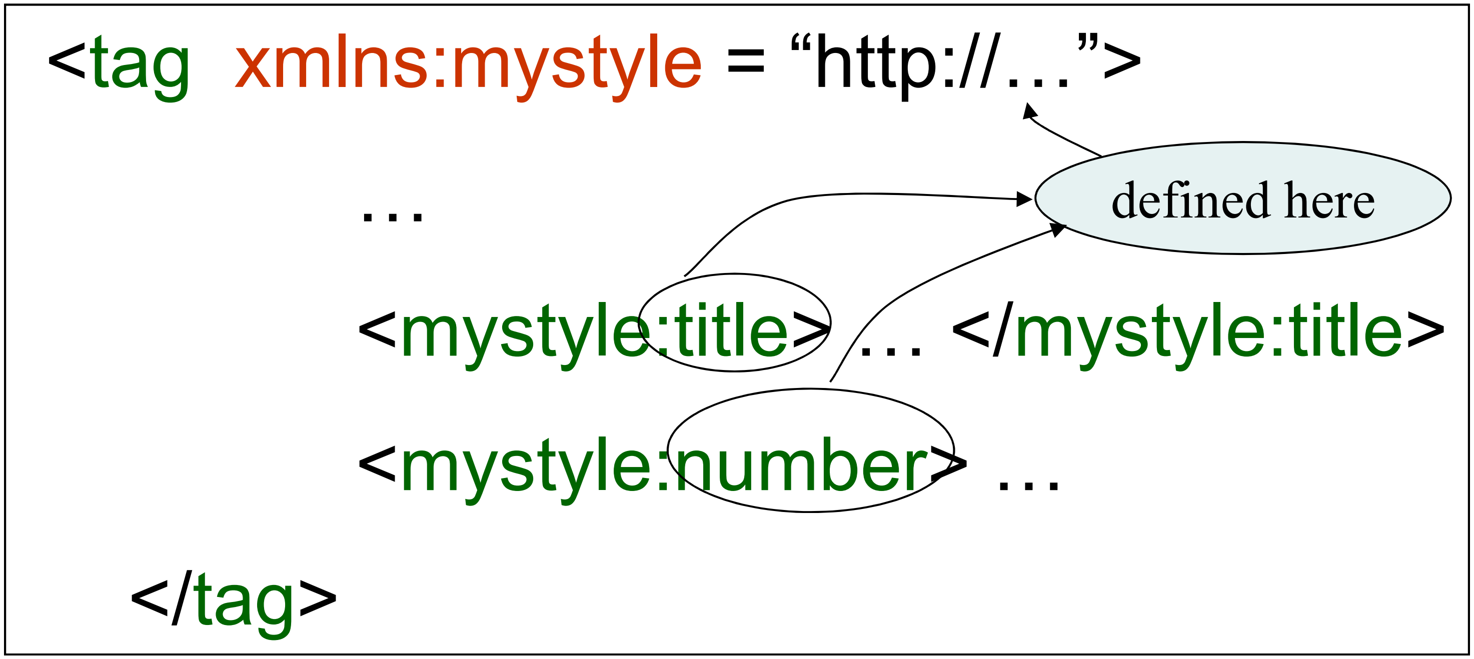
XML Namespaces

- http://www.w3.org/TR/REC-xml-names (1/99)
- name ::= [prefix:]localpart

```
<book xmlns:isbn=“www.isbn-org.org/def”>
        <title> … </title>
        <number> 15 </number>
        <isbn:number> …. </isbn:number>
</book>
```

XML Namespaces

- syntactic: <number> , <isbn:number>
- semantic: provide URL for schema



```
<tag  xmlns:mystyle = “http://…”>

              …

              <mystyle:title> … </mystyle:title>

              <mystyle:number> …

  </tag>
```

defined here

Implementing XML Repository

- Repository backend
  - plain text file
  - relational database
  - object database
  - tailor-made, specialized XML database
- Type information
  - even partial typing information can be used to improve the storage

Text files

- *it's the simplest way to store*
- *easy to handle*
- *widely available*
- have to check out an entire doc in order to retrieve a datum
- simultaneously access/update
- access/modify an item from a large catalog collection

Relational databases

- existing, proven technology to provide full database management

- it’s not easy and efficient to manage XML data in traditional RDBMS

An Example (using RDBMS)

- assume no typing information
- data can be an arbitrary graph
- let’s use two tables for the XML instances:
  - one to store all edge information
  - one to store values

The two tables

Ref(src, label, dst)

Val(oid, value)

Suppose a simple query like:

family/person/hobby

in XPath

The same query in SQL

```
select v.value
from Ref r1, Ref r2, Ref r3, Val v
where r1.src = “root” AND r1.label = “family”
AND r1.dst = r2.src AND r2.label = “person”
AND r2.dst = r3.src AND r3.label = “hobby”
AND r3.dst = v.oid
```

This is a 4-way join!!!

It’s very inefficient though index on label can help a lot.

Efficiency problem

- even simple query will have a large no of joins
- RDBMS organizes data based on the structure of tables and type info => clustering, indexing, query optimization are not working properly for XML data
- Also #ways to traverse path expressions are much more than that on tables

XML Parsers

- There are several different ways to categorise parsers:
  - Validating versus non-validating parsers
  - Parsers that support the Document Object Model (DOM)
  - Parsers that support the Simple API for XML (SAX)
  - Parsers written in a particular language (Java, C, C++, Perl, etc.)

The SAX Parser

- SAX parser is an event-driven API
  - An XML document is sent to the SAX parser
  - The XML file is read sequentially
  - The parser notifies the class when events happen, including errors
  - The events are handled by the implemented API methods to handle events that the programmer implemented

SAX Parser Events

- A SAX parser generates events
  - at the start and end of a document,
  - at the start and end of an element,
  - when it finds characters inside an element, and at several other points
- User writes the code that handles each event, and decides what to do with the information from the parser

Example Event Handlers

- startElementHandler
- endElementHandler
- charDataHandler
- CDATASectionHandler
- CommentHandler
- PIHandler
- etc...

When to (not to) use SAX

- Ideal for simple operations on XML files
  - E.g. reading and extracting elements
- Good for very large XML files (c.f. DOM)
- Not good if we want to manipulate XML structure
- Not designed for writing out XML

DOM

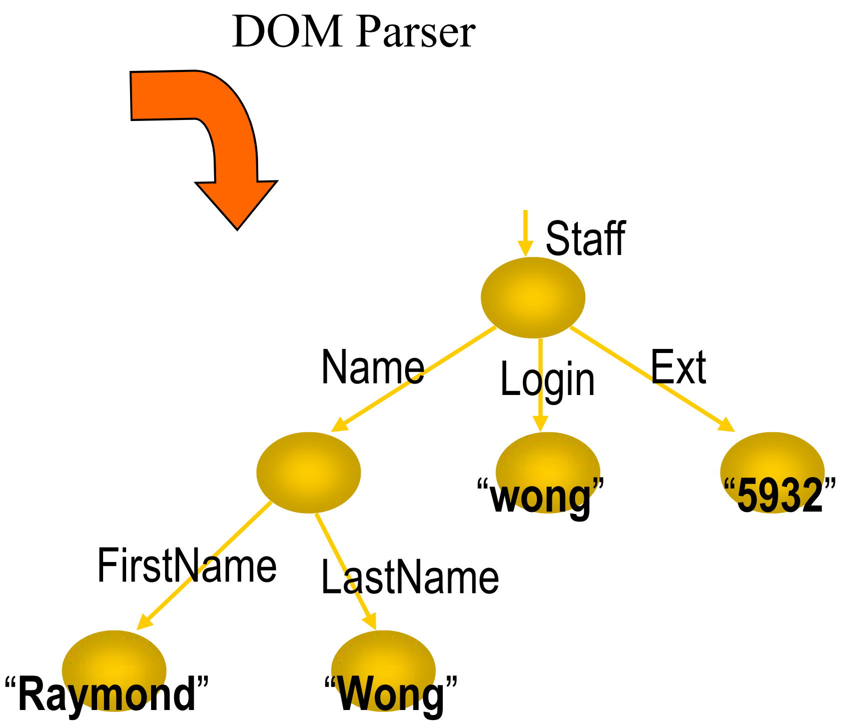
- Document Object Model
- Set of interfaces for an application that reads an XML file **into memory** and stores it as a **tree structure**
- The abstract API allows for constructing, accessing and manipulating the structure and content of XML and HTML documents

What a DOM Parser Gives

- When you parse an XML document with a DOM parser, you get back a tree structure that contains all of the elements of your document
- The DOM provides a variety of functions you can use to examine the contents and structure of the document

DOM Parser produces a memory tree (DOM Tree) after parsing

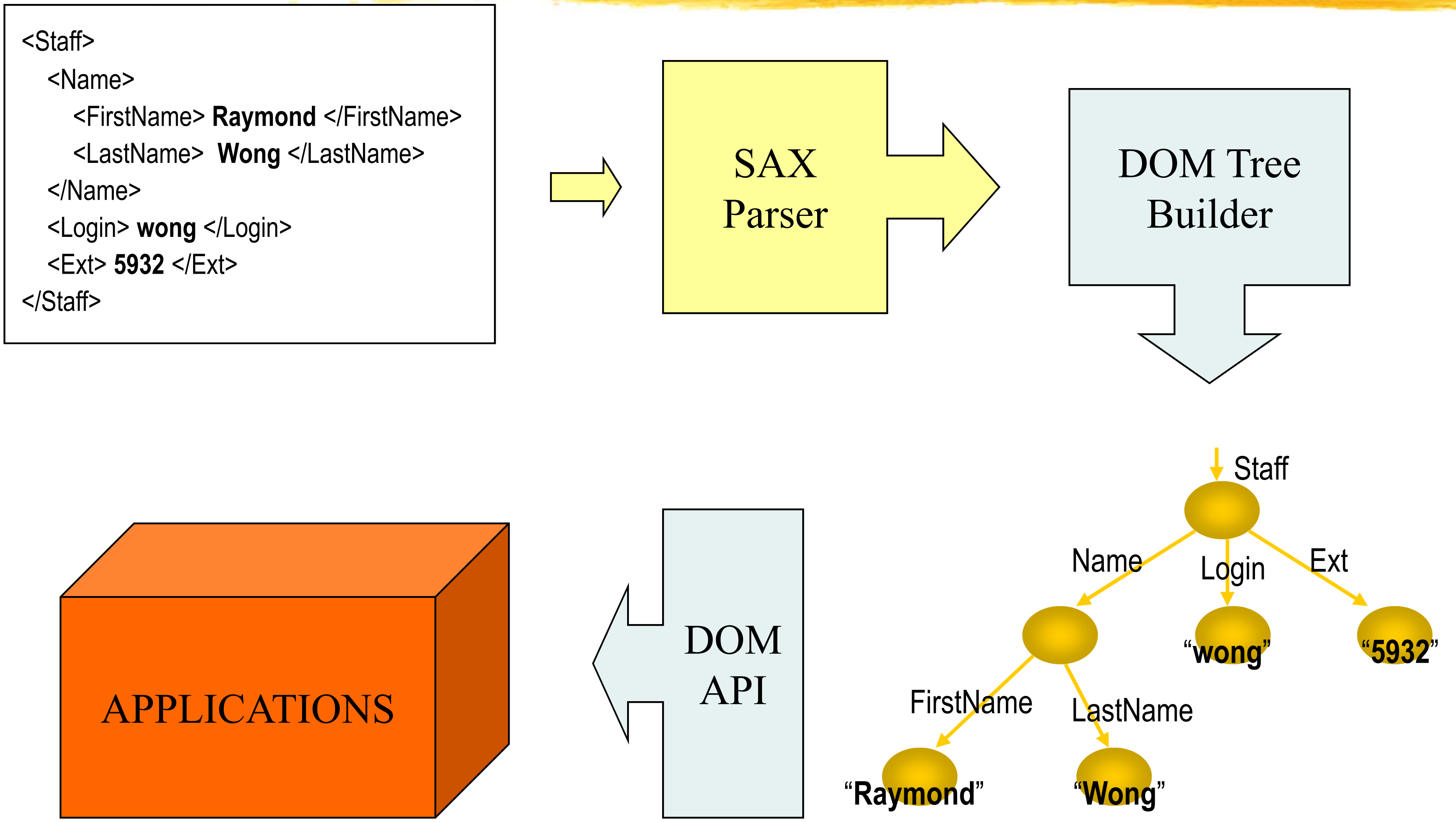
```
<Staff>
   <Name>
      <FirstName> Raymond </FirstName>
      <LastName>  Wong </LastName>
   </Name>
   <Login> wong </Login>
   <Ext> 5932 </Ext>
</Staff>
```



Why to Use DOM

- Task of writing parsers is reduced to coding against the DOM Tree API
- Domain-specific frameworks will be written on top of DOM

You can build a DOM parser using a SAX parser



```
<Staff>
   <Name>
      <FirstName> Raymond </FirstName>
      <LastName>  Wong </LastName>
   </Name>
   <Login> wong </Login>
   <Ext> 5932 </Ext>
</Staff>
```

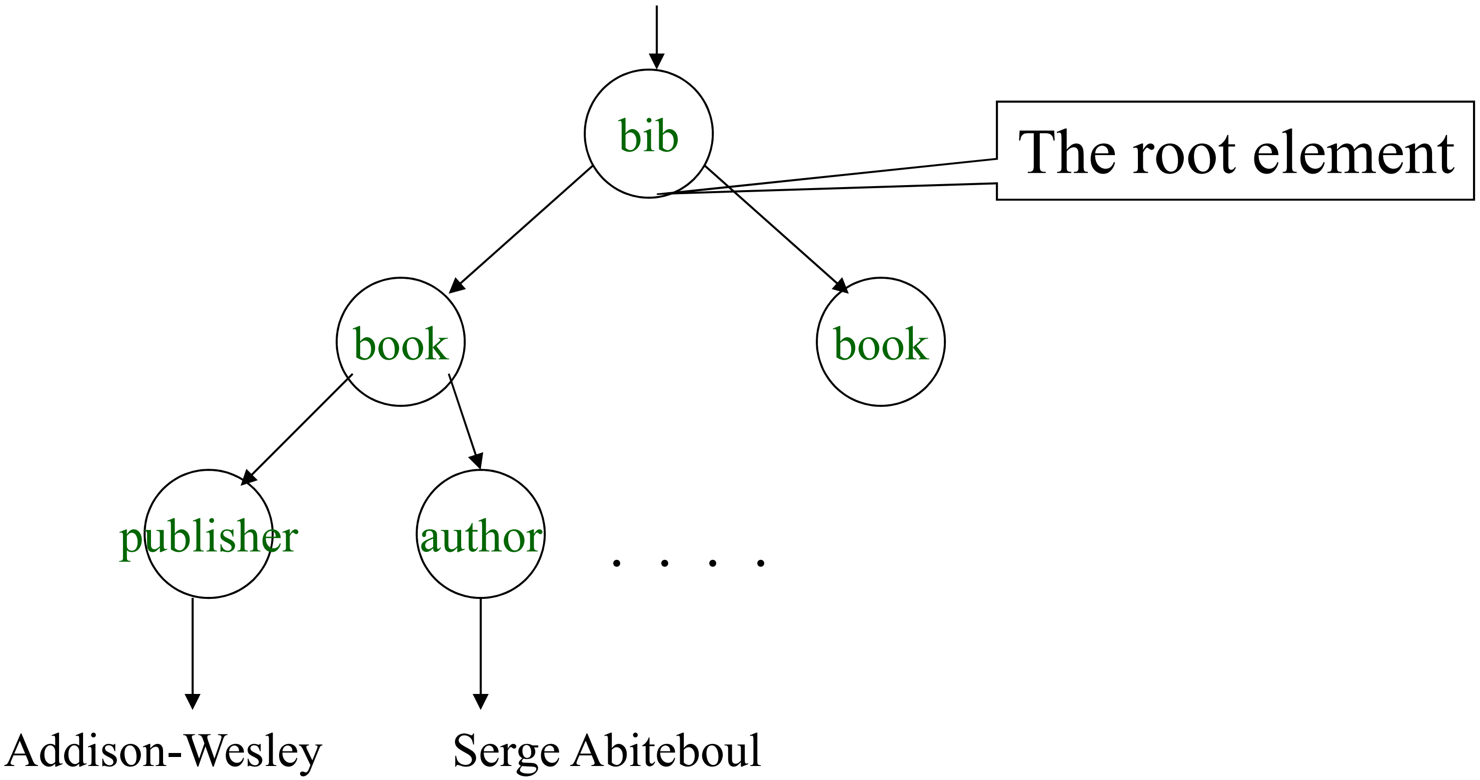
XPath 1.0

- http://www.w3.org/TR/xpath (11/99)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Link (XLink)
  - XML Pointer (XPointer)
  - XPath 2.0
  - XQuery
- Was originally part of XSL

Example for XPath Queries

```
<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



XPath: Simple Expressions

/bib/book/year

```
Result:  <year> 1995 </year>
          <year> 1998 </year>
```

/bib/paper/year

Result: empty

XPath: Restricted Kleene Closure

//author

```
Result:<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//first-name

```
Result:  <first-name> Rick </first-name>
```

XPath: Text Nodes

/bib/book/author/text()

Result: Serge Abiteboul
Victor Vianu
Jeffrey D. Ullman

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

Functions in XPath:

- text() = matches the text value
- node() = matches any node (= * or @* or text())
- name() = returns the name of the current tag

XPath: Wildcard

//author/*

Result: <first-name> Rick </first-name>
<last-name> Hull </last-name>

* Matches any element

XPath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is has to be an attribute

XPath: Qualifiers

/bib/book/author[firstname]

```
Result: <author> <first-name> Rick </first-name>
                 <last-name> Hull </last-name>
        </author>
```

XPath: More Qualifiers

/bib/book/author[firstname][address[//zip][city]]/lastname

Result: <lastname> … </lastname>
<lastname> … </lastname>

XPath: More Qualifiers

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

/bib/book[author/text()]

XPath: More Details

- We can navigate along 13 axes:

  ancestor
  ancestor-or-self
  attribute
  child
  descendant
  descendant-or-self
  following
  following-sibling
  namespace
  parent
  preceding
  preceding-sibling
  self