XML & XPath

INF 551

Wensheng Wu

Agenda

- XML:
  - What is it and why do we care?
  - Data model
  - Query language: XPath

XML

- eXtensible Markup Language
- XML 1.0 – a recommendation from W3C, 2008
- Root: SGML (standard generalized markup language)
- After the root: a format for sharing *data*

SGML

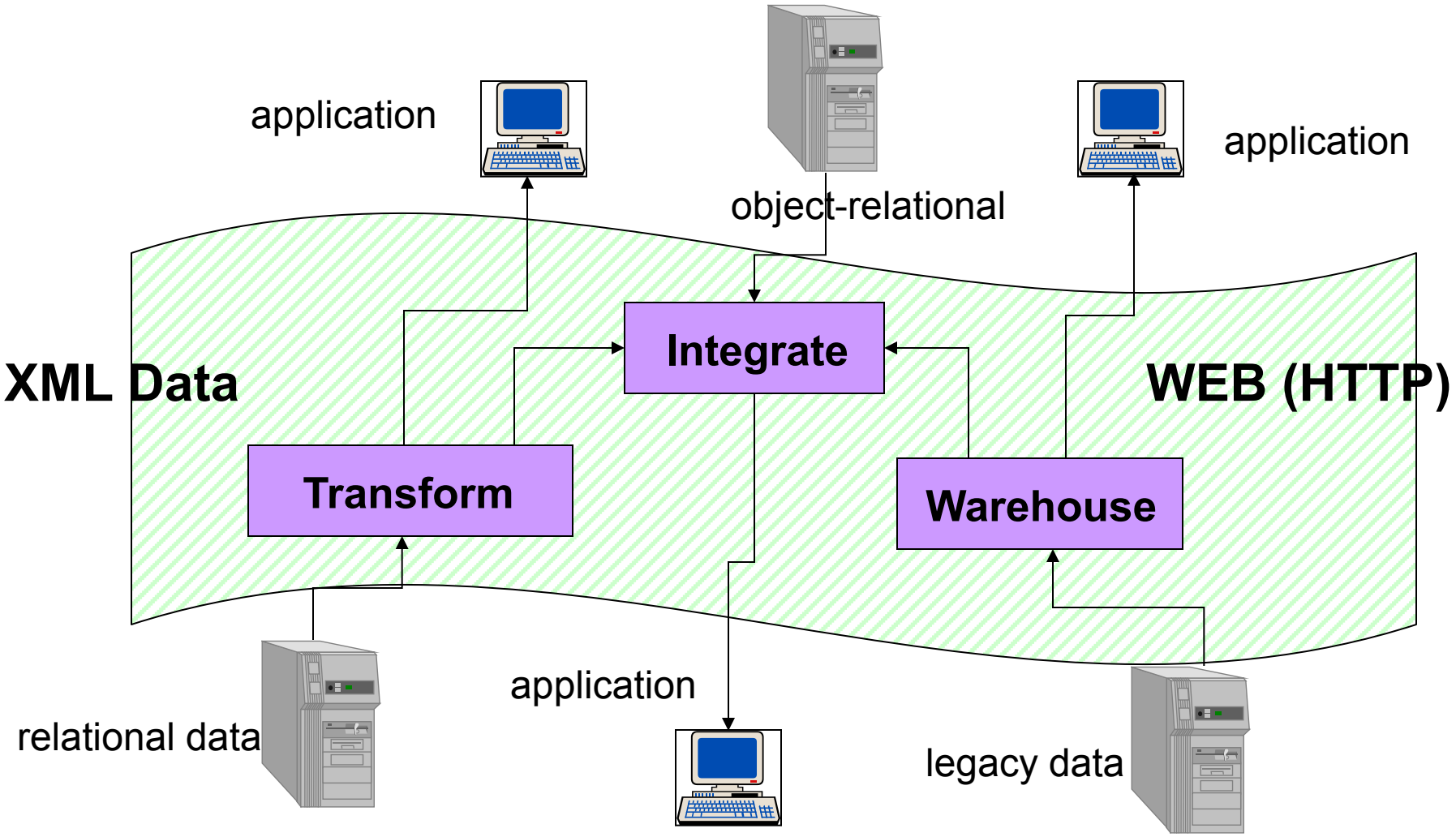
- Derived from IBM's GML developed in 1960's
  - Charles Goldfarb, Edward Mosher, and Raymond Lorie
  - For sharing of large-project documents

- Basis for HTML & XML
  - XML is roughly an augmented subset (adds more restrictions)
  - HTML is an application of SGML

Why XML is of Interest to Us

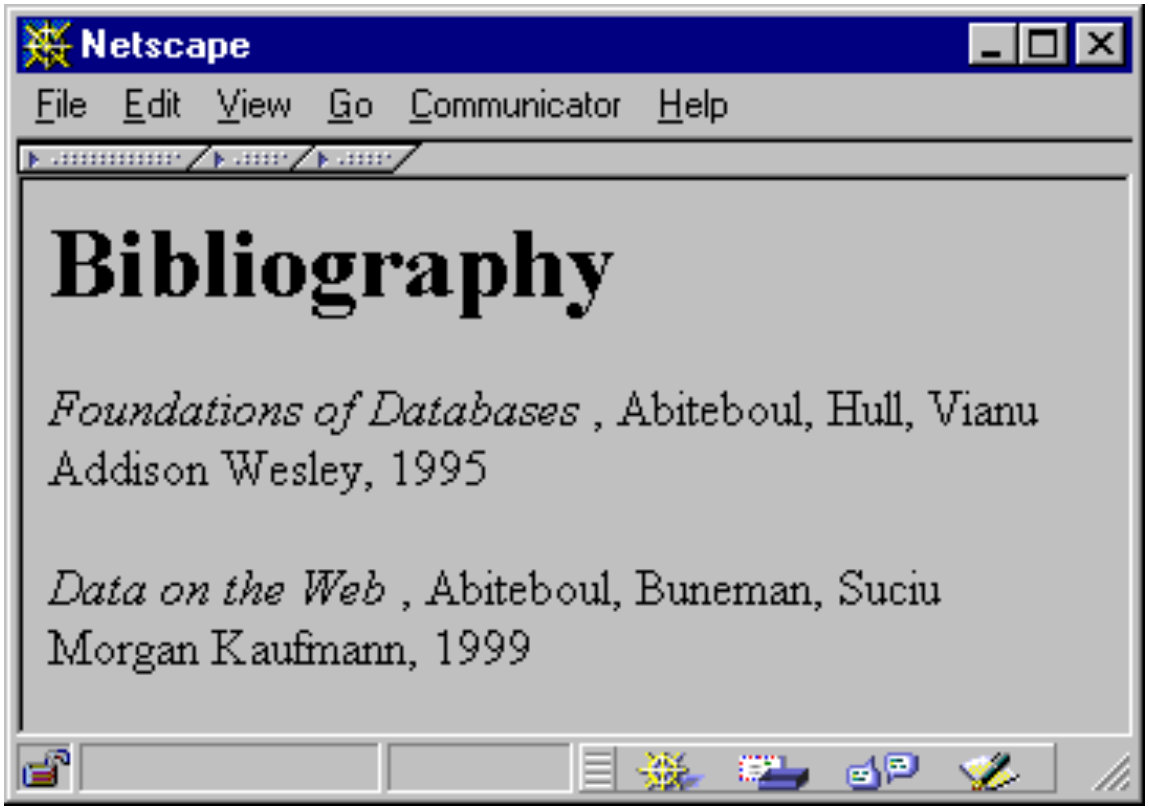
- XML is just syntax (serialization format) for data
  - Note: we have no syntax for relational data
  - But XML is not relational: *semi-structured*
- This is exciting because:
  - Can translate *any* data to XML
  - Can ship XML over the Web (HTTP)
  - Can input XML into any application
  - Thus: data sharing and exchange on the Web

XML Data Sharing and Exchange



Specific data management tasks

From HTML to XML



HTML describes the presentation

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
        Abiteboul, Hull, Vianu
        <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
        Abiteoul, Buneman, Suciu
        <br> Morgan Kaufmann, 1999
```

XML

```
<bibliography>
    <book>   <title> Foundations… </title>
             <author> Abiteboul </author>
             <author> Hull </author>
             <author> Vianu </author>
             <publisher> Addison Wesley </publisher>
             <year> 1995 </year>
    </book>
    …
</bibliography>
```

XML describes the content

Web Services

- Wikipedia: a software system designed to support interoperable machine-to-machine interaction over a network

- Use http for machine-machine communications of files
  - E.g., in XML & JSON formats
  - WSDL, SOAP (simple object access protocol)

Web service example (link)

TempConvert

Click here for a complete list of operations.

CelsiusToFahrenheit

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
Celsius:	23
	Invoke

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

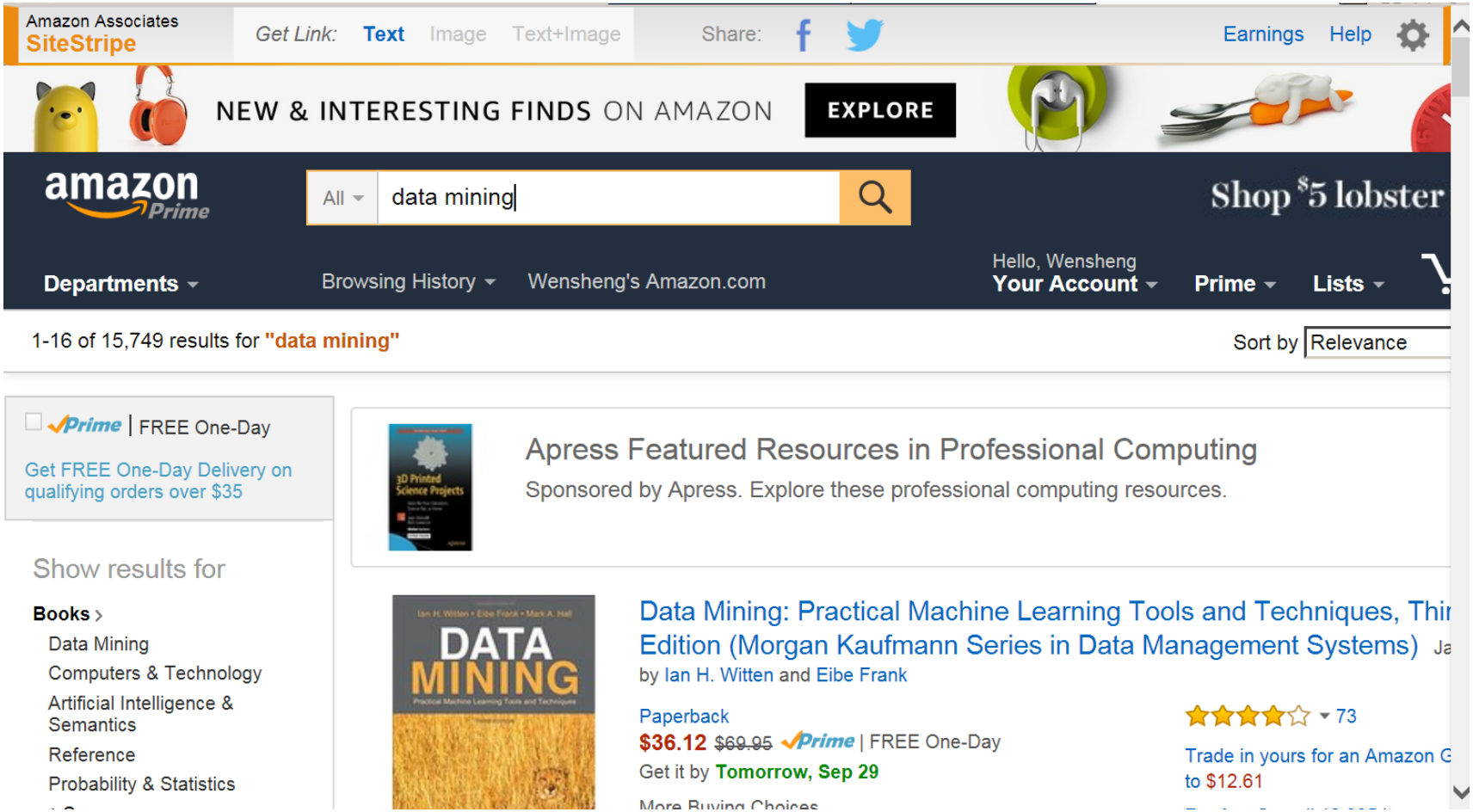
```
POST /xml/tempconvert.asmx HTTP/1.1
Host: www.w3schools.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.w3schools.com/xml/CelsiusToFahrenheit"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://sc
  <soap:Body>
    <CelsiusToFahrenheit xmlns="http://www.w3schools.com/xml/">
      <Celsius>string</Celsius>
    </CelsiusToFahrenheit>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
```

Ajax

- Asynchronous Javascript and XML
- Web clients send and receive data from server asynchronously
  - Benefit: more responsive web pages
- Common to use XML, JSON as data format

Ajax in action (link)



XML Terminology

- tags: book, title, author, …
- start tag: <book>, end tag: </book>
- elements: <book>…</book>,<author>…</author>
- elements may be nested:

  <book><author>…</author></book>
- empty element: <red></red> abbrv. <red/>
- an XML document: has a 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>
```

attributes are alternative ways to represent data

Attributes

- <book price = “55” currency = “USD”>

- Attribute values must be quoted, either double or single

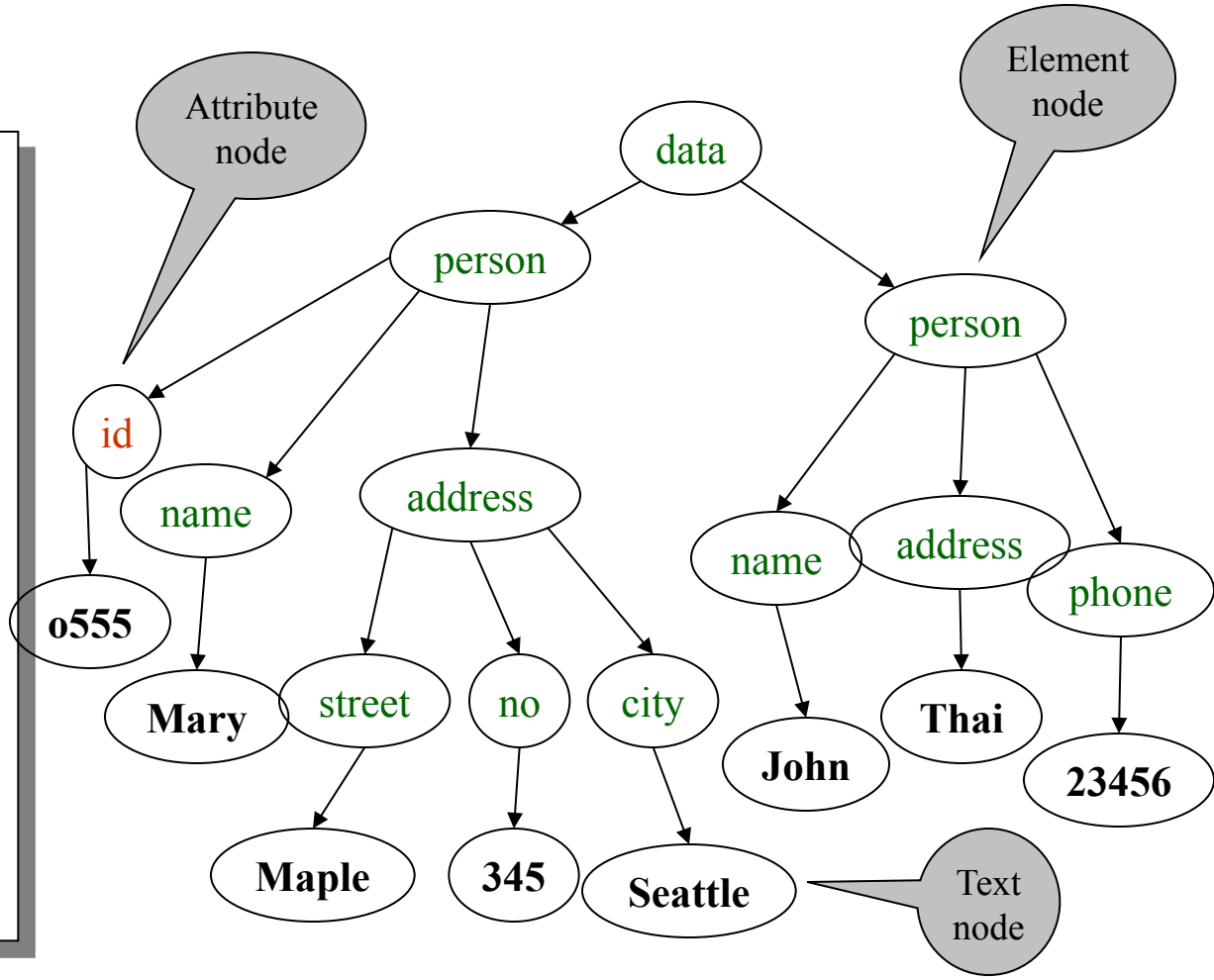
More XML: Oids and References

```
<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>
```

Must define them in schema

XML Semantics: an ordered tree

```
<data>
    <person id="o555" >
        <name> Mary </name>
        <address>
            <street> Maple </street>
            <no> 345 </no>
            <city> Seattle </city>
        </address>
    </person>
    <person>
        <name> John </name>
        <address> Thailand  </address>
        <phone> 23456  </phone>
    </person>
</data>
```



Order matters !!!

XML Data

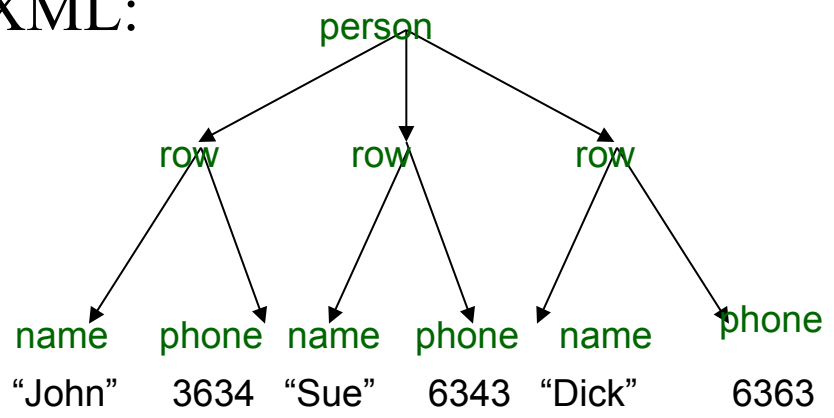
- XML is self-describing
- Schema elements become part of the data
  - Reational schema: persons(name,phone)
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

Relational Data as XML

person

name	phone
John	3634
Sue	6343
Dick	6363

XML:




```
<person>
    <row> <name>John</name>
          <phone> 3634</phone></row>
    <row> <name>Sue</name>
          <phone> 6343</phone>
    <row> <name>Dick</name>
          <phone> 6363</phone></row>
</person>
```

XML is Semi-structured Data

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

← no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

XML is Semi-structured Data

- Repeated attributes

```
<person> <name> Mary</name>
         <phone>2345</phone>
         <phone>3456</phone>
</person>
```

← two phones !

- Impossible in tables:

name	phone	
Mary	2345	3456

???

XML is Semi-structured Data

- Attributes with different types in different objects

```
<person> <name>  <first> John </first>
                 <last> Smith </last>
         </name>
         <phone>1234</phone>
</person>
```

← structured name !

- Nested structures
- Heterogeneous contents:
  - <bib> contains both <book>'s and <cd>'s

Document Type Definitions DTD

- part of the original XML specification
- an XML document may have a DTD
- XML document:

  well-formed = if tags are correctly closed

  Valid = if it has a DTD and conforms to it
- validation is useful in data exchange

Very Simple DTD

```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office, phone?)>
  <!ELEMENT ssn        (#PCDATA)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
  <!ELEMENT description    (#PCDATA)>
]>
```

Example with attributes

```
<!ELEMENT artist EMPTY>
<!ATTLIST artist name CDATA #REQUIRED>
<!ATTLIST artist artistID ID #REQUIRED>
<!ELEMENT album EMPTY>
<!ATTLIST album name CDATA #REQUIRED>
<!ATTLIST album albumArtistID IDREF #IMPLIED>
<!ATTLIST album contributingArtistIDs IDREFS
#IMPLIED>
```

"Implied" means optional

DTD as Part of XML Document

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

Example XML for Company DTD

Example of valid XML document:

```
<company>
   <person>  <ssn> 123456789 </ssn>
             <name> John </name>
             <office> B432 </office>
             <phone> 1234 </phone>
   </person>
   <person>  <ssn> 987654321 </ssn>
             <name> Jim </name>
             <office> B123 </office>
   </person>
   <product> ...  </product>
   ...
</company>
```

DTD: The Content Model

<!ELEMENT tag (*CONTENT*)>

content model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA/#CDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

- #CDATA (#PCDATA)
  - Character data not are (are) parsed by parser
  - Tags inside #PCDATA will be treated as markup

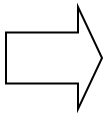
DTD: Regular Expressions

DTD

XML

sequence

```
<!ELEMENT name
                (firstName, lastName))
```



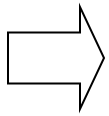
```
<name>
   <firstName>  . . . . . </firstName>
   <lastName>  . . . . . </lastName>
</name>
```

optional

```
<!ELEMENT name  (firstName?, lastName))
```

Kleene star

```
<!ELEMENT person   (name, phone*))
```



```
<person>
   <name>  . . . . . </name>
   <phone>  . . . . . </phone>
   <phone>  . . . . . </phone>
   <phone>  . . . . . </phone>
   . . . . . .
</person>
```

alternation

```
<!ELEMENT person   (name, (phone|email)))
```

Processing instructions

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

- This is the first line of an XML document
  - Declaring that the following is an XML doc…
  - that follows standard version 1.0
  - and whose encoding is UTF-8

Agenda

- XML:
  - What is it and why do we care?
  - Data model
  - Query language: XPath

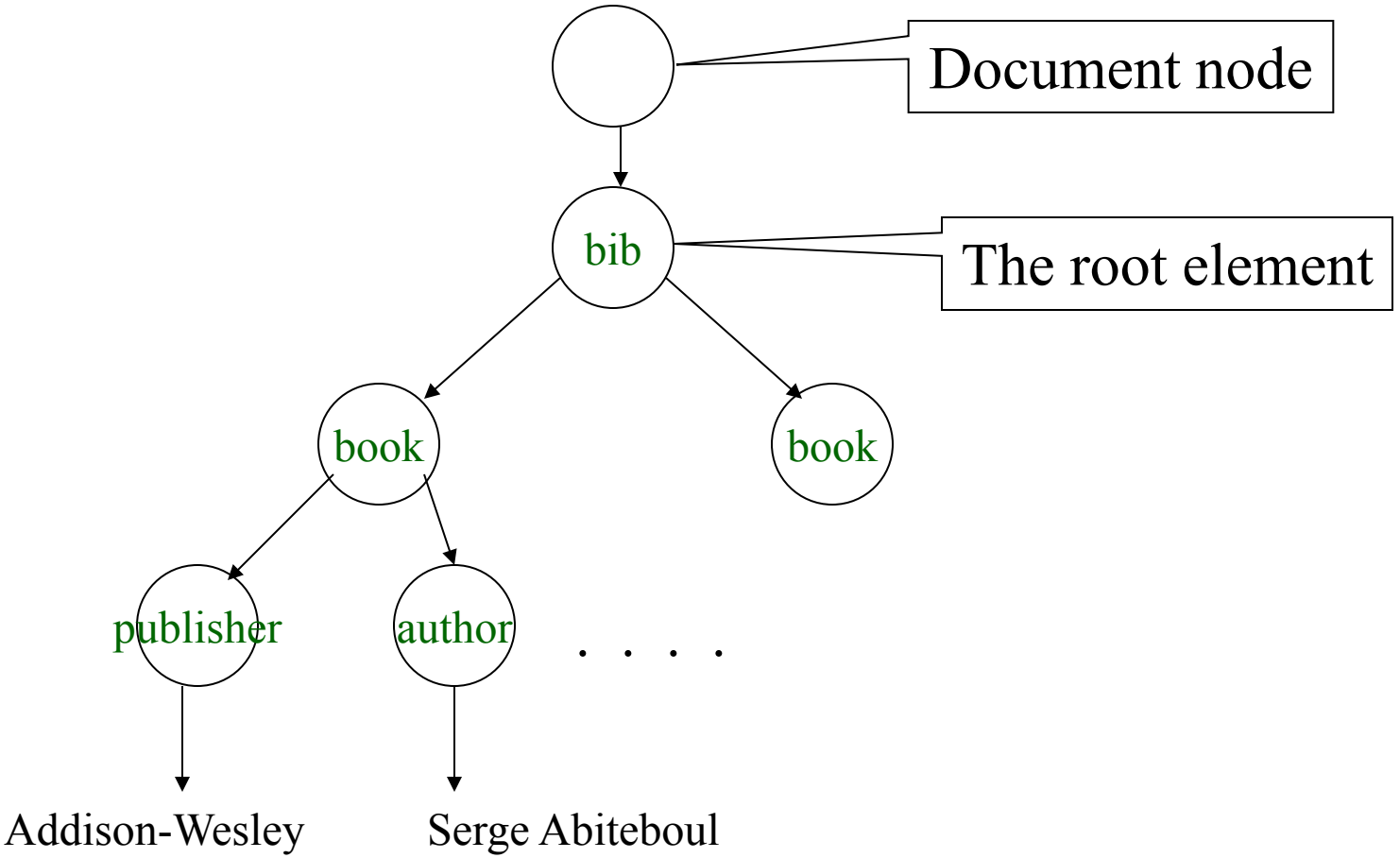
Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML

Sample Data for 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 Expression Evaluator

- XPath Expression Testbed
  - http://www.whitebeam.org/library/guide/TechNotes/xpathtestbed.rhtm

XPath: Simple Expressions

/bib/book/year

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

/bib/paper/year

Result: empty (there were no papers)

Recursive Search of Children

```
//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>

Select Child by Index

- Index of children starts from 1
- //author[1]
- /bib/book[2]/author

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 elements

Functions in XPath:

- text() = matches the text value
- node() = matches any node (element or text)
- * = matches only element nodes

Xpath: Wildcard

```
//author/*
```

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

* Matches any element

Xpath: Attribute Nodes

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

Result: “price=55”

@price means that price has to be an attribute

Same as ?

```
/bib/book[@price]
```

Xpath: Attribute nodes

- /bib/book/@*
  - Return all attribute nodes of book element

- Result: 'price=55'

Xpath: Predicates

/bib/book/author[first-name]

Return author elements (under /bib/book) which have a child element called "first-name"

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

Xpath: More Predicates

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

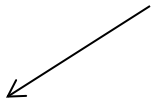
Return lastname of author elements which have child element firstname and child element "address" which itself has …

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

Xpath: More Predicates

/bib/book[@price < 60]

/bib/book[author/@age < 25]



/bib/book[author/text()]

Return books under bib that have an author element with a text node

Xpath: More Predicates

```
/bib/book[contains(author, 'Ullman')]
```

Return books under bib whose author subelment contains the word 'Ullman' in its text node
(note contains is case-sensitive)

Xpath: More Predicates

- /bib/book[price > 30 or year > 1995]
- /bib/book[price > 30 and year >= 1995]

Xpath: alternatives

```
/bib/book|/bib/cd
```

Return book and cd elements under /bib

Questions

What do these return?

```
//*
```

```
//@*
```