Understanding XML

Hans C. Arents

senior IT market analyst

I.T. Works

"Guiding the IT Professional"

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56

E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/

Understanding XML

- Part 1: XML intro — 13u30 – 15u00

  coffee break

- Part 2: **XML standards** — 15u30 – 17u30

  questions & answers

XML standards

Hans C. Arents

senior IT market analyst

I.T. Works

"Guiding the IT Professional"

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56

E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/

XML standards

- How does the standards approval process work?
- Standards for handling XML
  - XML namespaces
  - XML Schemas
- Standards for building XML solutions
  - XQL: E**x**tensible **Q**uery **L**anguage
    - XPath
  - XSL: E**x**tensible **S**tylesheet **L**anguage
    - XSLT (using XPath) and XSLFO
  - XLL: E**x**tensible **L**inking **L**anguage
    - XLink and XPointer (using XPath)
  - DOM: **D**ocument **O**bject **M**odel

The W3C standards approval process

- Recommendation track:
  - **Notes**

    non-official documents, used for initiating Working Group discussions
  - **Working Drafts** (**Last Call Working Draft**)

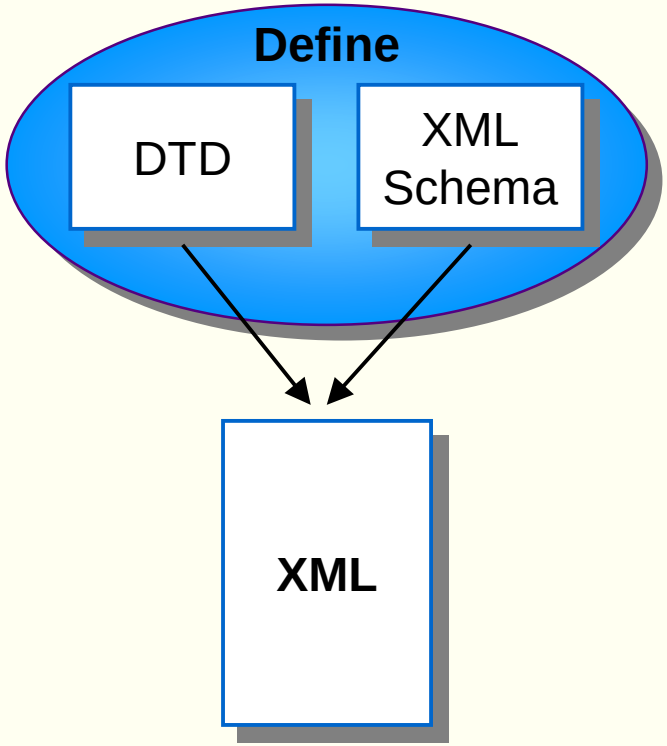
    official documents, representing work in progress without consensus
  - **Candidate Recommendation**

    an official document, a stable Working Draft being proposed for implementation experience and feedback
  - **Proposed Recommendation**

    an official document, a Candidate Recommendation which is sent to the W3C Advisory Committee for final review
  - **Recommendation**

    an official document, a Proposed Recommendation on which a consensus has been reached and which becomes a W3C standard

Standards for handling XML

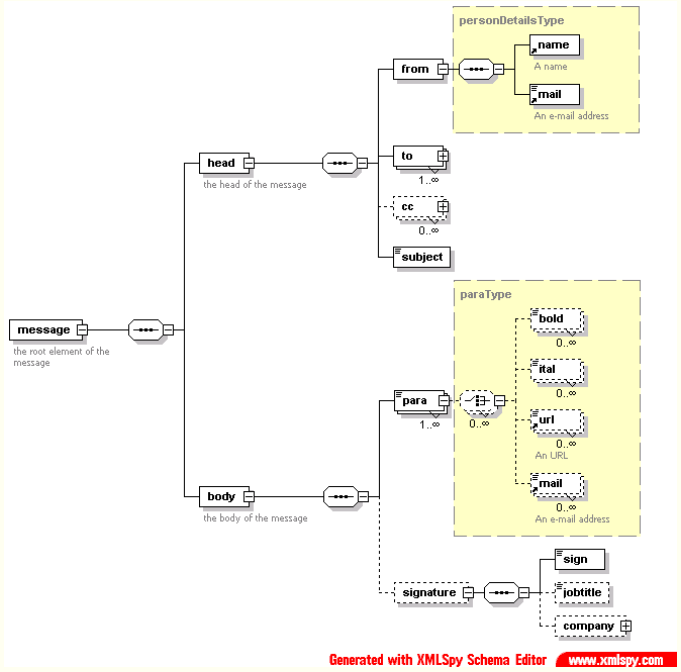


Standards for handling XML

"how to work with the XML syntax"

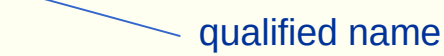
XML namespaces: *how to mix XML tags*

DTD/XML Schema: *how to define XML vocabularies*



XML namespaces

- Problem:
  - same tag may mean different things
    - e.g. **`<title>`** = document title *or* formal title of author
    - e.g. **`<quote>`** = stock quote *or* quote in a person's speech
- Solution:
  - create tags and attributes that are unique in the context of their specific use
  - by adding a prefix to the tag or attribute to make clear that it belongs to a specific namespace
    - e.g. **`<easdaq:quote>...</easdaq:quote>`**



- Note:
  - not a way of making modular DTDs
  - just a syntactic way of giving to elements and attributes programmer-friendly names that are unique across the whole Internet

XML namespaces

```
<?xml version="1.0" standalone="yes"?>
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns:a="http://www.amazon.com/books">
  <h:head>
    <h:title>Book Review</h:title>
  </h:head>
  <h:body>
    <a:bookreview>
      <a:title>XML: A Primer</a:title>
      <h:table>
        <h:tr align="center">
          <h:td>Author</h:td><h:td>Price</h:td>
          <h:td>Pages</h:td><h:td>Date</h:td>
        </h:tr>
        <h:tr align="left">
          <h:td><a:author>Simon St. Laurent</a:author></h:td>
          <h:td><a:price>31.98</a:price></h:td>
          <h:td><a:pages>352</a:pages></h:td>
          <h:td><a:date>1998/01</a:date></h:td>
        </h:tr>
      </h:table>
    </a:bookreview>
  </h:body>
</h:html>
```

XML namespaces

```
<?xml version="1.0" standalone="yes"?>
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
        xmlns:a="http://www.amazon.com/books">
  <h:head>
    <h:title>Book Review</h:title>
  </h:head>
  <h:body>
    <a:bookreview>
      <a:title h:style="font-family: Arial;">XML: A Primer</a:title>
      <h:table>
        <h:tr align="center">
          <h:td>Author</h:td><h:td>Price</h:td>
          <h:td>Pages</h:td><h:td>Date</h:td>
        </h:tr>
        <h:tr align="left">
          <h:td><a:author>Simon St. Laurent</a:author></h:td>
          <h:td><a:price>31.98</a:price></h:td>
          <h:td><a:pages>352</a:pages></h:td>
          <h:td><a:date>1998/01</a:date></h:td>
        </h:tr>
      </h:table>
    </a:bookreview>
  </h:body>
</h:html>
```

XML namespaces

```
<?xml version="1.0" standalone="yes"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:a="http://www.amazon.com/books">
  <head>
    <title>Book Review</title>
  </head>
  <body>
    <a:bookreview>
      <a:title>XML: A Primer</a:title>
      <table>
        <tr align="center">
          <td>Author</td><td>Price</td>
          <td>Pages</td><td>Date</td>
        </tr>
        <tr align="left">
          <td><a:author>Simon St. Laurent</a:author></td>
          <td><a:price>31.98</a:price></td>
          <td><a:pages>352</a:pages></td>
          <td><a:date>1998/01</a:date></td>
        </tr>
      </table>
    </a:bookreview>
  </body>
</html>
```

XML Schema

Data marked up using XML:

```
<ShoeOrder>
  <Color>Brown</Color>
  <Size>43.5</Size>
</ShoeOrder>
```

Incorrect XML:

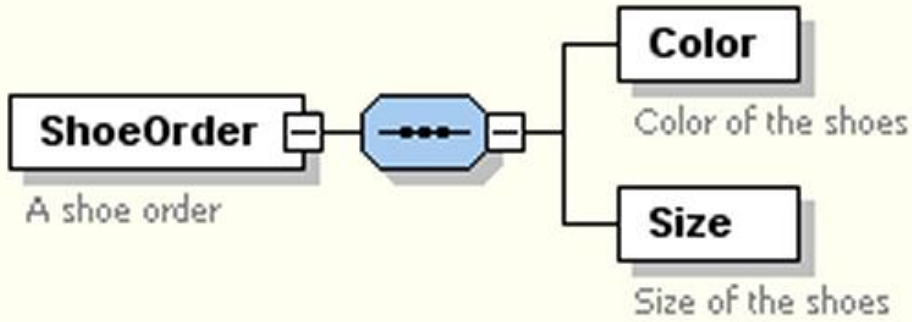
```
<ShoeOrder>
  <Color>Brown</Color>
</ShoeOrder>

<ShoeOrder>
  <Size>43.5</Size>
  <Color>Brown</Color>
</ShoeOrder>
```

Correct XML:

```
<ShoeOrder>
  <Color>Brown</Color>
  <Size>very big</Size>
</ShoeOrder>
```

Schema defined using a “classic” DTD:

```
<!ELEMENT ShoeOrder (Color,Size)>

<!ELEMENT Color     (#PCDATA)   >
<!ELEMENT Size      (#PCDATA)   >
```



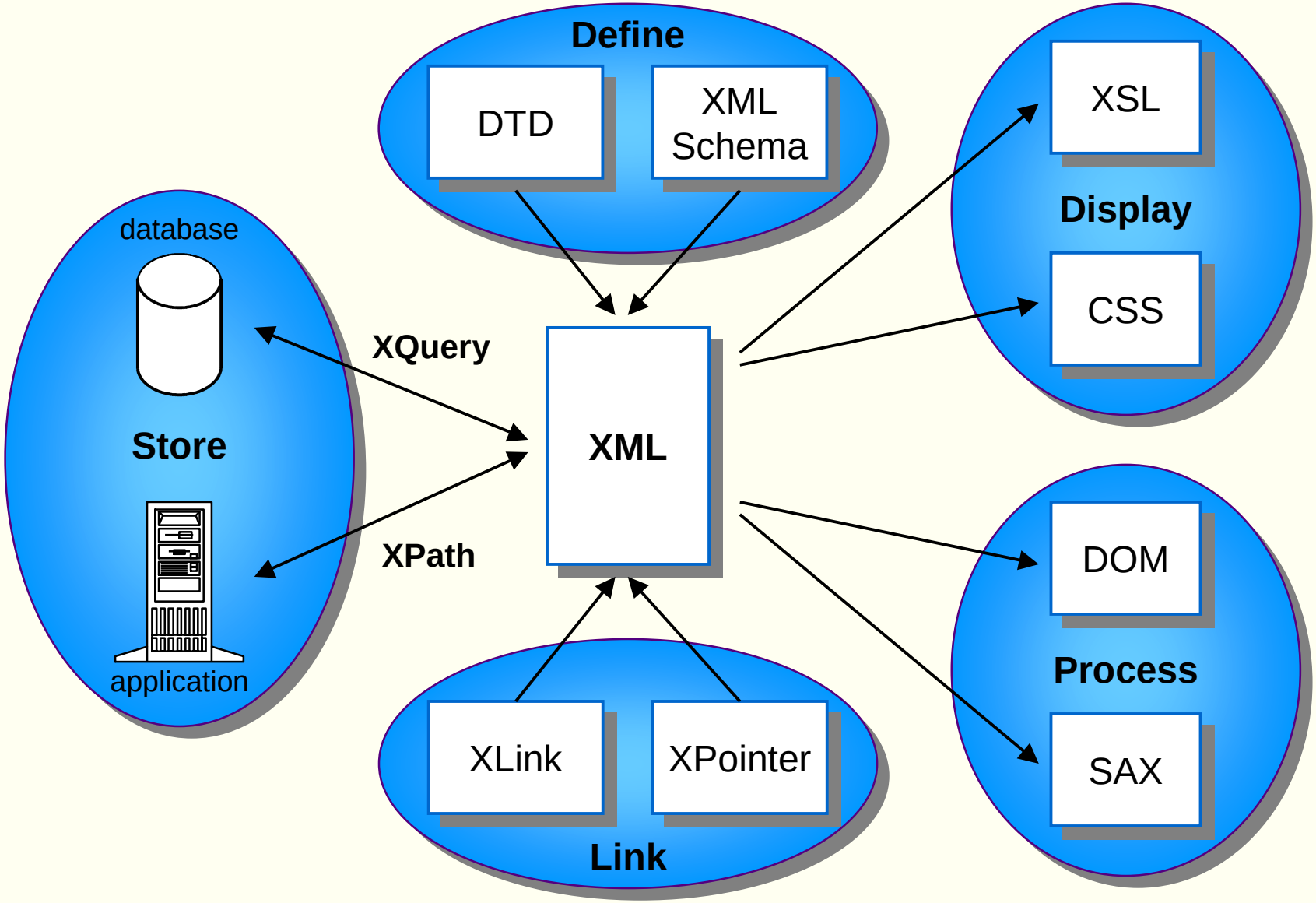
XML Schema

- Problem:
  - XML DTD is in a different syntax than XML documents
  - **è** use XML syntax
  - XML DTD is not expressive enough for rich data modeling
  - **è** add datatypes
- Solution: **XML Schema**
  - Part 1: **Structure**
    - defining XML DTDs using XML syntax
  - Part 2: **Datatypes**
    - defining datatypes
    - defining datatype constraints to be applied to the contents of XML elements or attributes
- Status: W3C Recommendation (May 2, 2001)

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            elementFormDefault="qualified">
  <xsd:element name="ShoeOrder">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Color" type="shoeColors"/>
        <xsd:element name="Size" type="shoeSizes"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="shoeColors">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Black"/>
      <xsd:enumeration value="Brown"/>
      <xsd:enumeration value="Grey"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="shoeSizes">
    <xsd:restriction base="xsd:float">
      <xsd:minInclusive value="17"/>
      <xsd:maxInclusive value="47"/>
      <xsd:pattern value="\d\d(\.5)?"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Standards for building XML solutions



Standards for building XML solutions

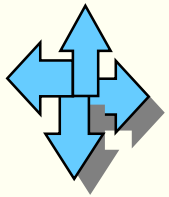
"how to do something with XML documents/data"

XQuery : XML Query Language

how to query XML

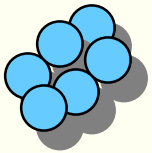
XSL : Extensible Stylesheet Language

how to transform and render XML



XLL : Extensible Linking Language

how to link in XML



DOM: Document Object Model

SAX : Simple API for XML

how to process XML

XQuery: XML Query Language

- Language for querying XML based on:
  - **XML Query Requirements**: W3C Working Draft (Feb 15, 2001)
    - declarative query language
    - must have both an XML and a non-XML syntax
    - suited for querying XML documents and data, full-text retrieval, …
      - examples in XML Query Use Cases: W3C Working Draft (Feb 15, 2001)
  - **XML Query Data Model**: W3C Working Draft (Feb 15, 2001)
    - node-labeled, tree-constructor representation of a tree
    - including the concept of node identity to simplify the representation of XML reference values (e.g. IDREF, XPointer, and URI values)
  - **XML Query Algebra**: W3C Working Draft (Feb 15, 2001)
    - well-defined semantics for the query language
      - projection, iteration, selection, quantification
      - joining, grouping, restructuring, sorting, aggregation
    - set of equivalence laws useful for query optimization

XQuery: XML Query Language

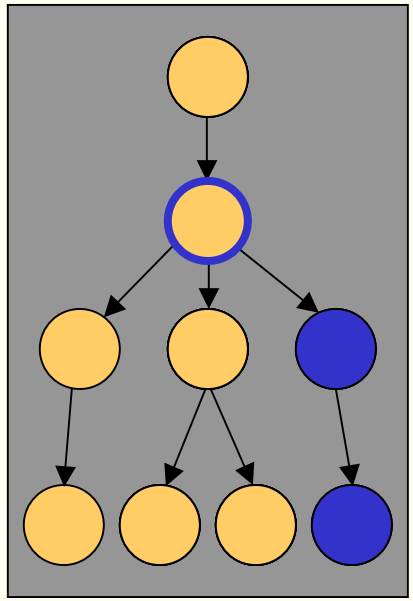
- XQuery syntax
  - based on XML Query Data Model and XML Query Algebra
  - ideas taken from SQL: use of patterns of query clauses
    - "**FLWR**" expressions: FOR, LET, WHERE and RETURN
  - ideas taken from OQL: functional language with nestable expressions
  - uses a *locating* language **XPath**
    W3C Recommendation (November 16, 1999)
    - to search for particular element structures
    - to search for elements with particular content / particular attributes
- Note:
  - before XQuery, number of competing suggestions
    - XQL (**X**ML **Q**uery **L**anguage), XML-QL, Lorel, …

    most XML data servers support (a subset/superset of) XQL
    - XQL = XPath + extensions, so not a query language!

XPath: XML Path Language

- Language for locating things in an XML document
  - by stepping through the document using a **location path**
  - location path is string-based rather than tag-based
- Location path:
  - is built from **location steps** concatenated with /
    - **`location step/location step/location step`**
  - a location step specifies a point in the document relative to another point in the document:
    - absolute
      - starting from the root "/"
    - relative from the *context node*
      - starting from the current node
    - from a well-defined position
      - using id() or document() functions

    along a certain **axis**



XPath: examples

- **`id("c725")`** the element with unique identifier c725
- **`root()`** *abbreviated to* **`/`** the root of the document
- **`child::para`** *abbreviated to* **`para`**

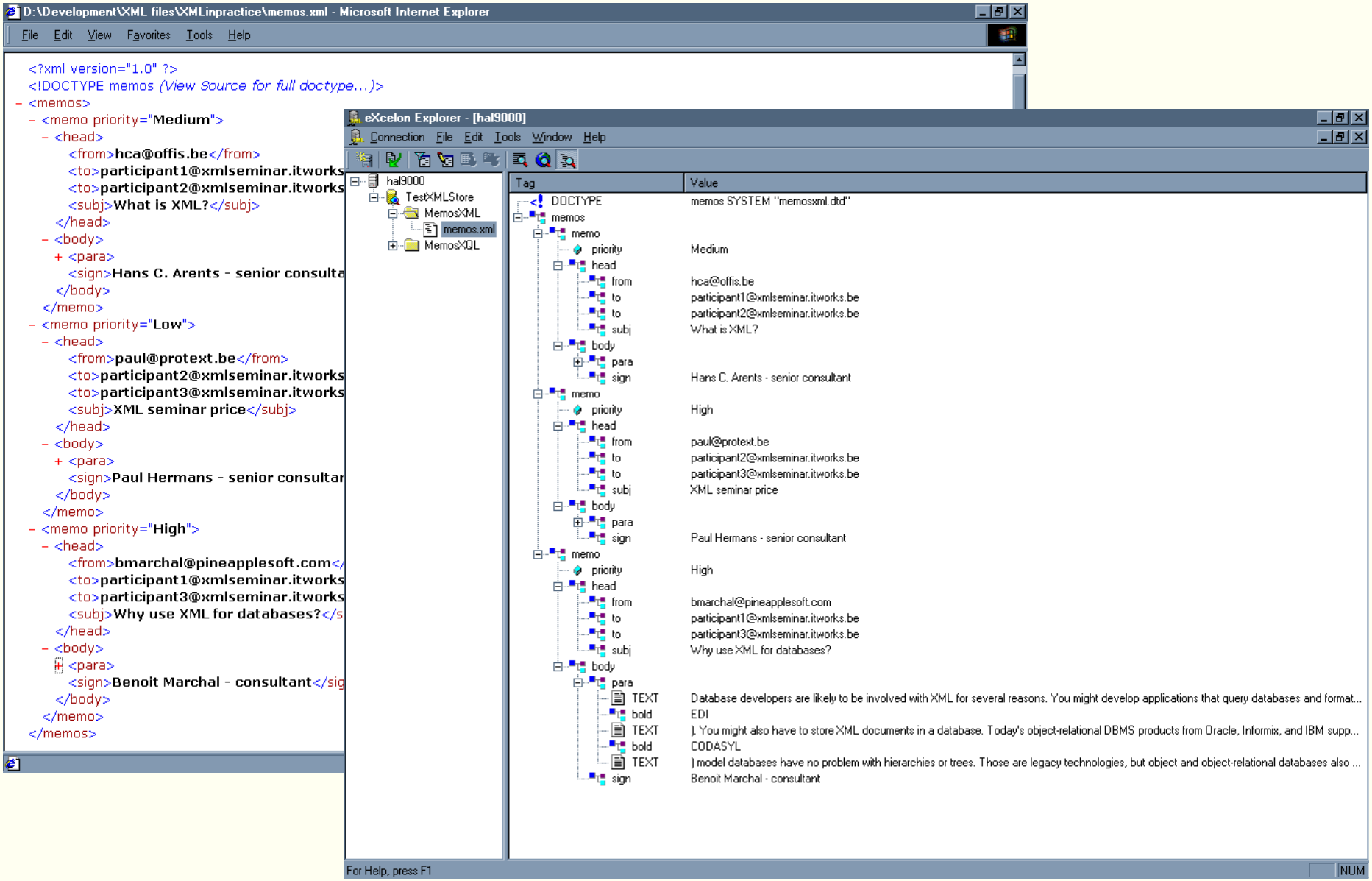
  `self::node()/para` *abbreviated to* **`./para`**
  - all of the para children of the context node
- **`parent::node()/para`** *abbreviated to* **`../para`**
  - all of the para children of the parent of the context node
- **`/descendant::node()/para[position()=5]`**

  abbreviated to **`//para[5]`**
  - the fifth para element anywhere in the document
- **`attribute::align`** *abbreviated to* **`@align`**
  - the align attribute of the context node

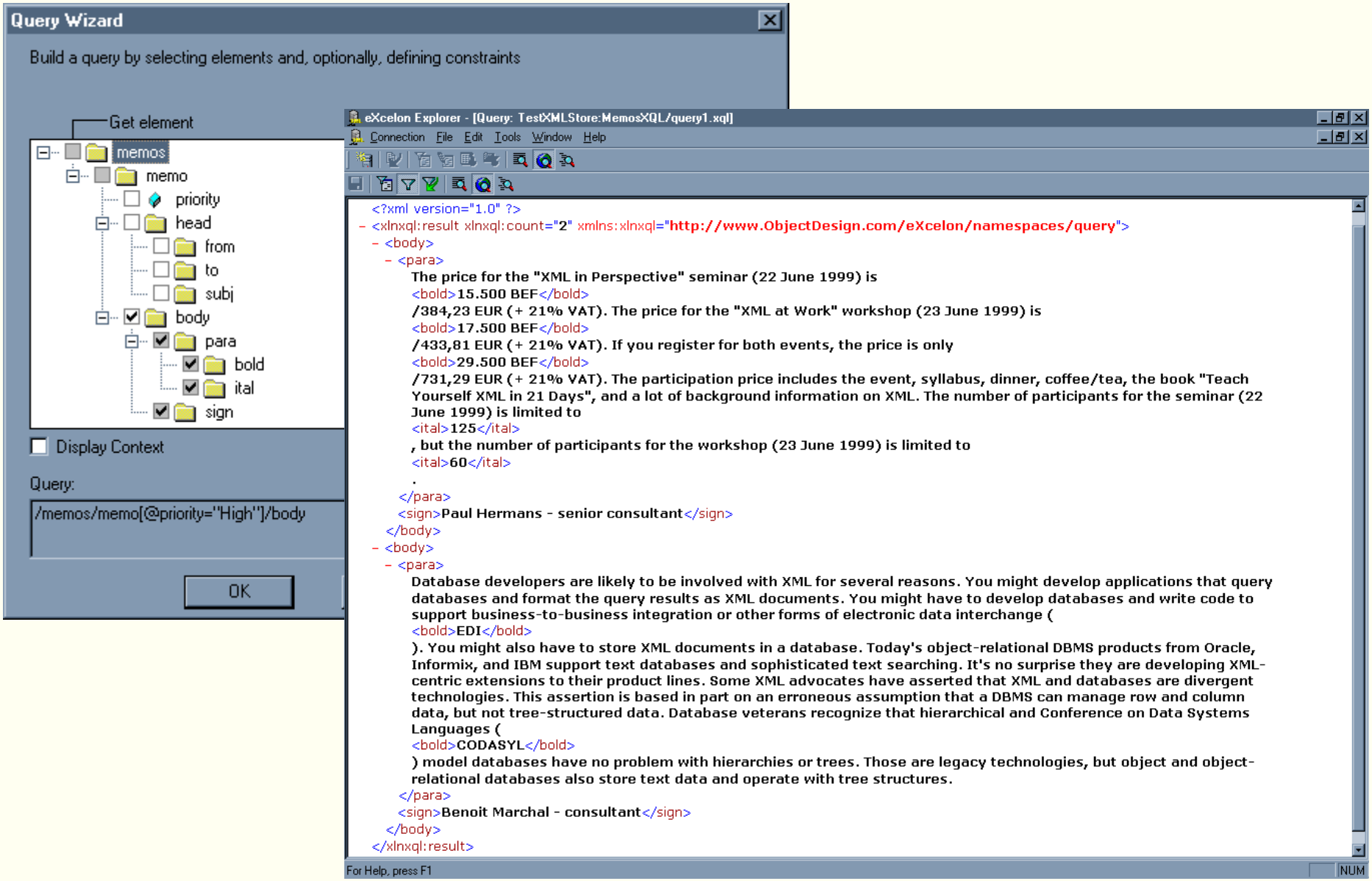
XPath: examples

- **`list/item`**
  - all of the item children of all of the list children of the context node
- **`list//para`**
  - all of the para elements (nested arbitrarily deep) under all of the list children of the context node. In other words, this location path matches list/item/para, list/item/note/para, …
- **`note[@type="warning"]`**
  - only note children that have a type attribute with the value "warning"
- **`section[title="Introduction"]`**
  - all the section children with a title of "Introduction"
- **`//section[title="Examples"]/example[last()]`**
  - the last example element in all section elements (anywhere in the document) with a title element of "Examples"

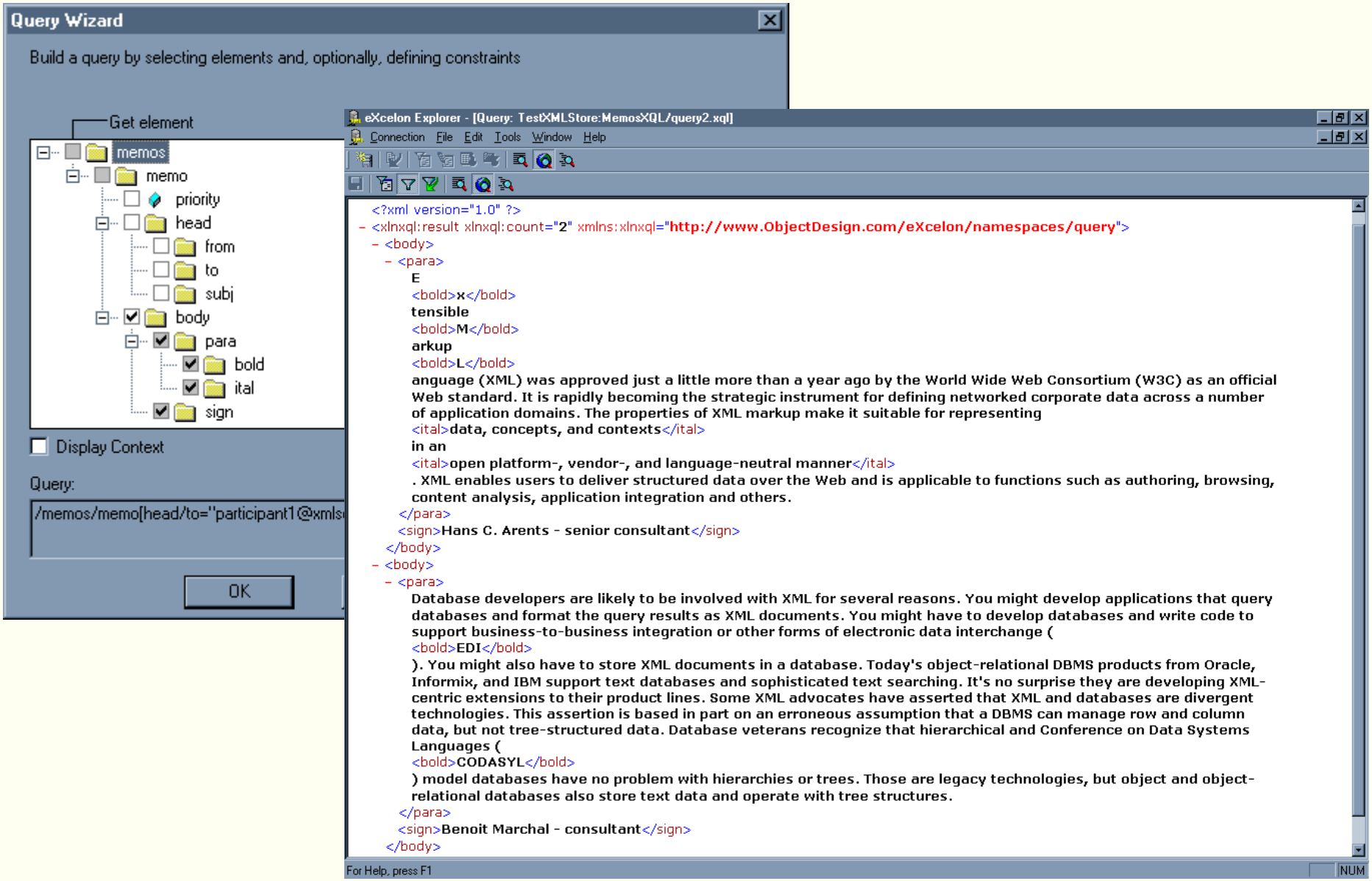
Storing XML data



Querying XML data using XPath



Querying XML data using XPath



XQuery: XML Query Language

- Examples:
  - list the titles of books published by Morgan Kaufmann in 1998

```
FOR $b IN document("bib.xml")/book
WHERE $b/publisher = "Morgan Kaufmann" AND $b/year = "1998"
RETURN $b/title
```

  - list each publisher and the average price of its books

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")/book[publisher = $p]/price)
RETURN
   <publisher>
      <name> $p/text() </name> ,
      <avgprice> $a </avgprice>
   </publisher>
```

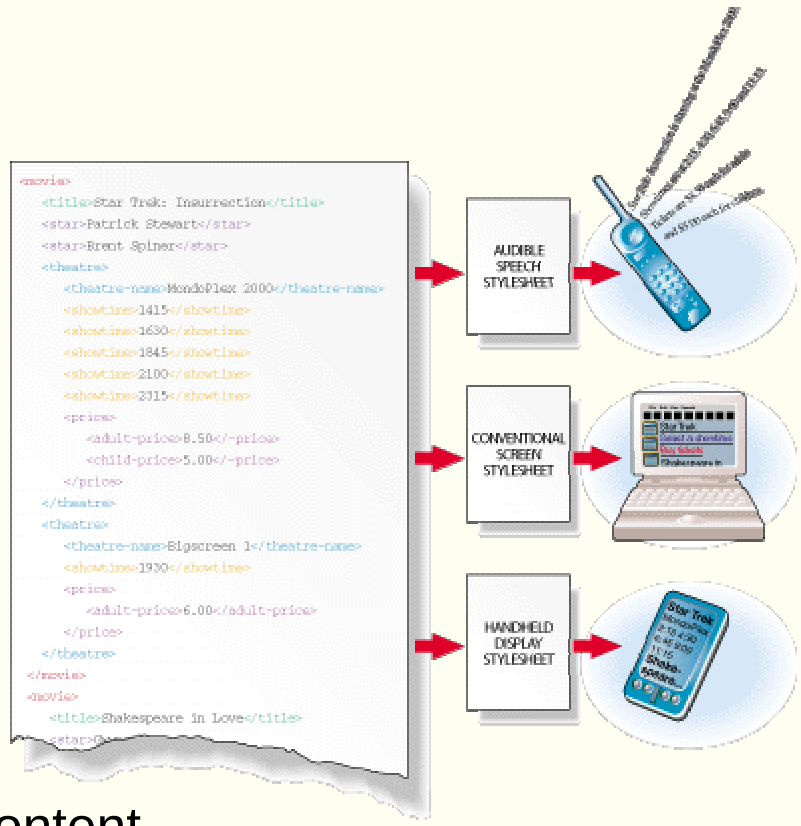
- Note: XQuery does not contain insert/update/delete operations, so it is not an XML data manipulation language (yet)!

Manipulating XML data (*eXcelon*)

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

  <remove select="/bookstore/book[-1]">
  </remove>

  <update select="/bookstore">
    <element location="lastchild">
      <book style="tour">
        <title>Sightseeing in Trenton</title>
        <price>5.99</price>
      </book>
    </element>
  </update>

  <update select="/bookstore/magazine[1]">
    <attribute name="frequency">weekly</attribute>
  </update>

  <update select="/bookstore/book/title='History of Trenton'">
    <element>
      <title>History of Trenton 1800-1900</title>
    </element>
  </update>

</xlnupdate>
```

XSL: Extensible Stylesheet Language

- XML stylesheet mechanism
  - based on DSSSL
  - compatible with CSS 1.0, 2.0 & 3.0
  - advanced capabilities:
    - contents can be reordered/sorted
    - text can be deleted/duplicated/moved
    - new information can be “computed” from existing information or structures



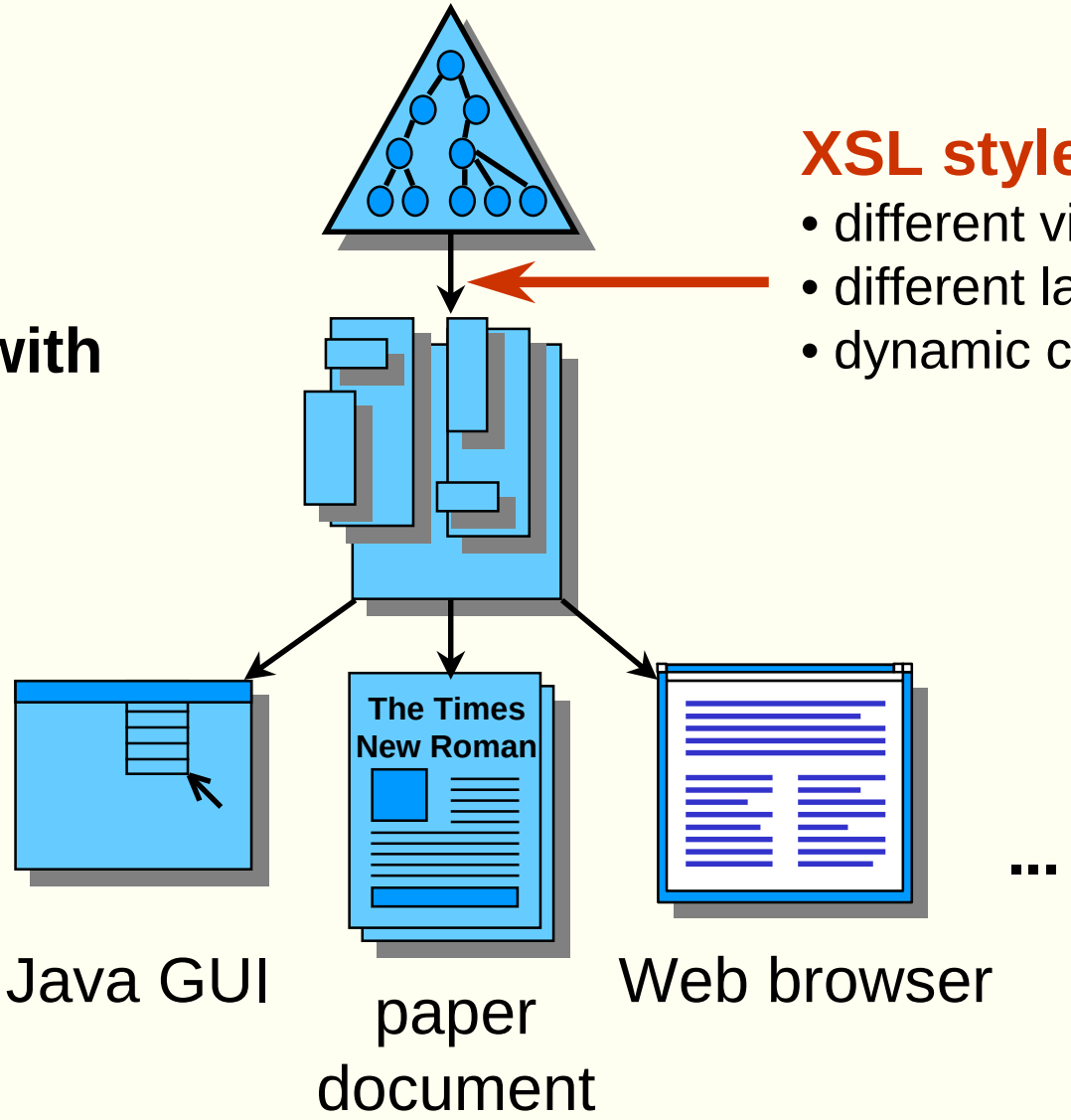
- Useful for:
  - keeping presentation separate from content
  - single source, multiple output media
  - reformatting / re-using content
- Not just document formatting, but also data restructuring!

XSL: Extensible Stylesheet Language

source tree

result tree with formatting objects

output media



XSL stylesheet

- different views
- different layouts
- dynamic contents

XSL: key concepts

- Two separate processes:
  - transformation: **XSLT** (XSL **T**ransformations) using XPath
  - formatting: **XSLFO** (XSL **F**ormatting **O**bjects)
- Transformation
  - how an XML source tree should be transformed
  - recursive and declarative programming
- Formatting
  - how the XML result tree should be presented
  - specified using **f**ormatting **o**bjects (FOs)
- Status:
  - XSLT: W3C Recommendation (November 16, 1999)
    - XSLT 1.1: W3C Working Draft (December 12, 2000)
    - XSLT 2.0 Requirements: W3C Working Draft (February 14, 2001)
  - XSLFO: W3C Candidate Recommendation (November 21, 2000)

XSLT = transformation

Input:

```
<birthday><date>December 3, 1997</date>
          <fullname>John Doe</fullname>
</birthday>
```

Stylesheet:

```
<xsl:stylesheet xmlns=...>
  <xsl:template match="birthday/date">
    <fo:block font-style="italic" color="green">
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

Output from XSLT engine:

```
<fo:block font-style="italic" color="green">
  December 3, 1997
</fo:block>
```

XSLFO = formatting

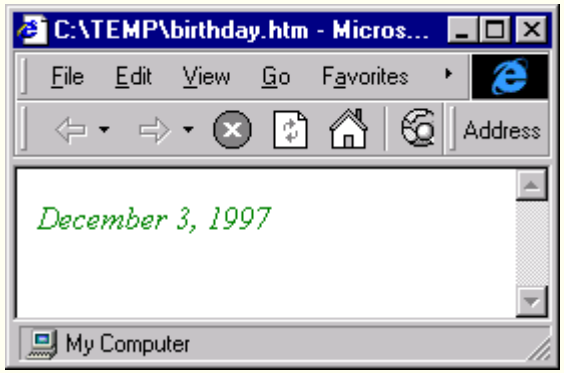
Input for XSLFO engine:

```
<fo:block font-style="italic" color="green">
  December 3, 1997
</fo:block>
```

Output:

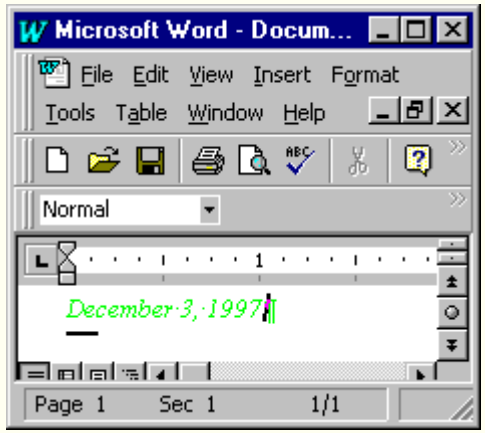
From XSLFO engine for HTML:

```
<p style="font-style:italic;color:green">
  December 3, 1997
</p>
```



From XSLFO engine for RTF:

```
{\c6\f12\i December 3, 1997\par}
```



XSLFO = formatting

- Character, block, list, page, graphics, numbering, …
  - with their appropriate formatting characteristics
- Example:

```
<xsl:template match="chapter/title">
  <fo:block
      font-family="sans-serif"
      font-weight="bold"
      keep-with-next="true"
      page-break-inside="avoid">
      <fo:character
          color="blue">
            <xsl:apply-templates/>
      </fo:character>
  </fo:block>
</xsl:template>
```

XSL » XSLT

Input:

```
<birthday><date>December 3, 1997</date>
          <fullname>John Doe</fullname>
</birthday>
```

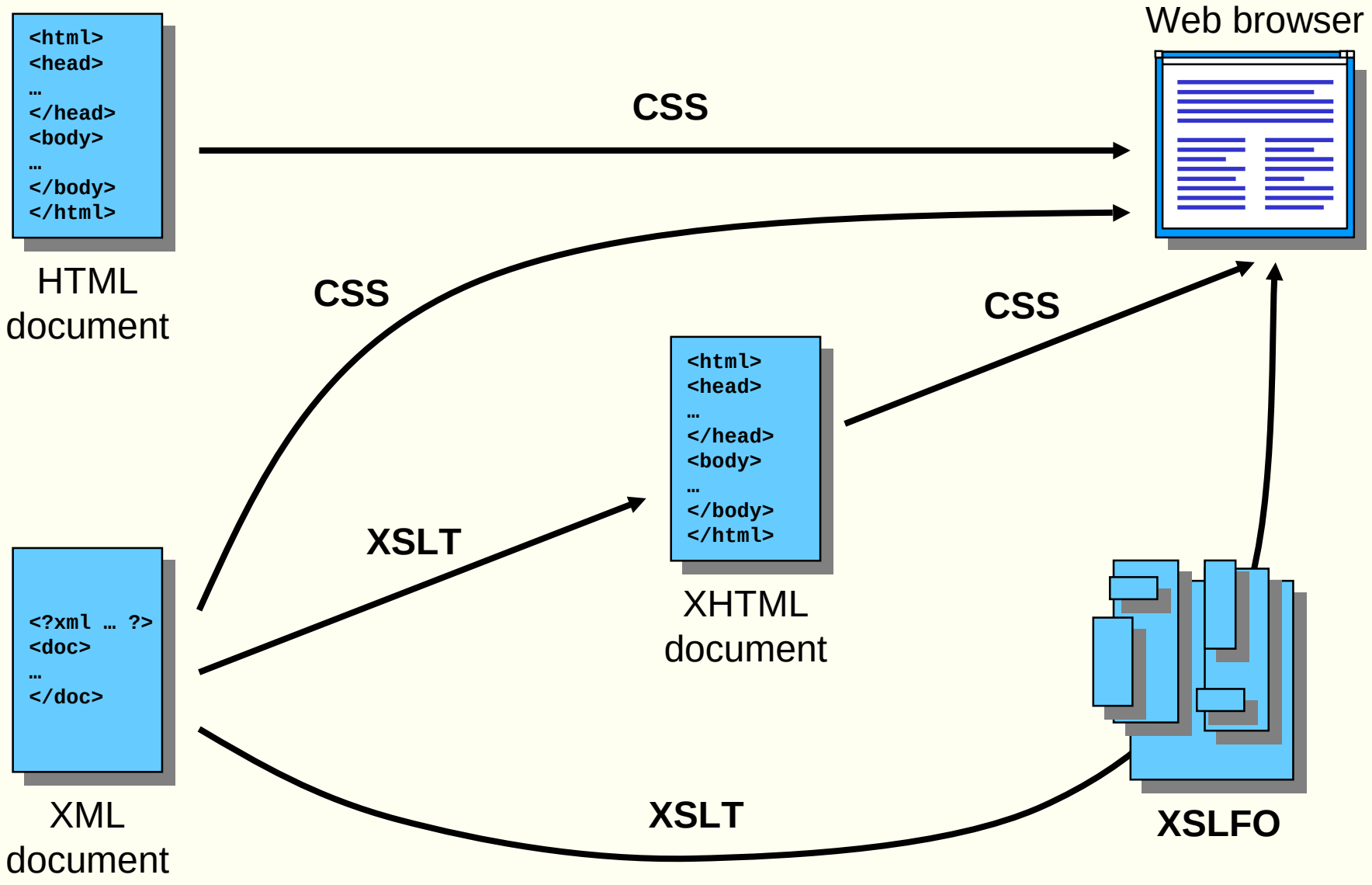
Stylesheet:

```
<xsl:stylesheet xmlns=...>
  <xsl:template match="birthday/date">
    <p style="font-style:italic;color:green">
      <xsl:apply-templates/>
    </p>
  </xsl:template>
</xsl:stylesheet>
```

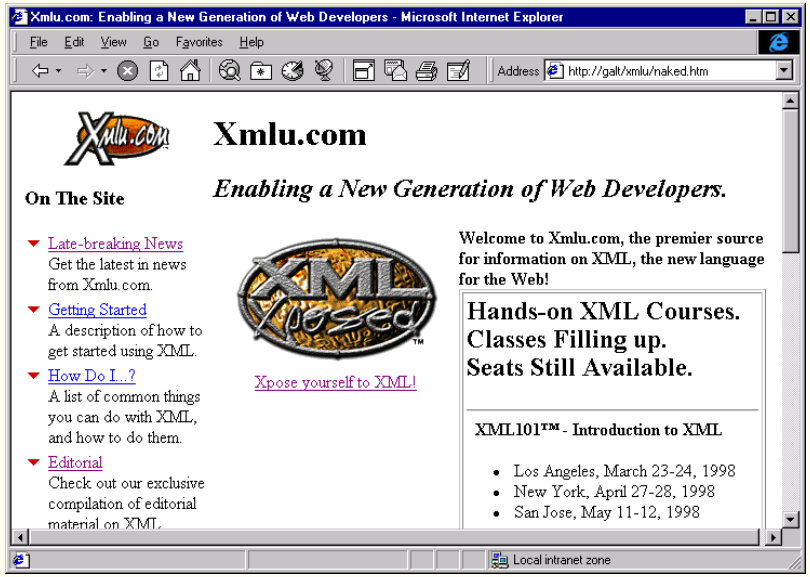
Output from XSLT engine:

```
<p style="font-style:italic;color:green">
  December 3, 1997
</p>
```

How to display XML in a Web browser



CSS (Cascading Style Sheets)

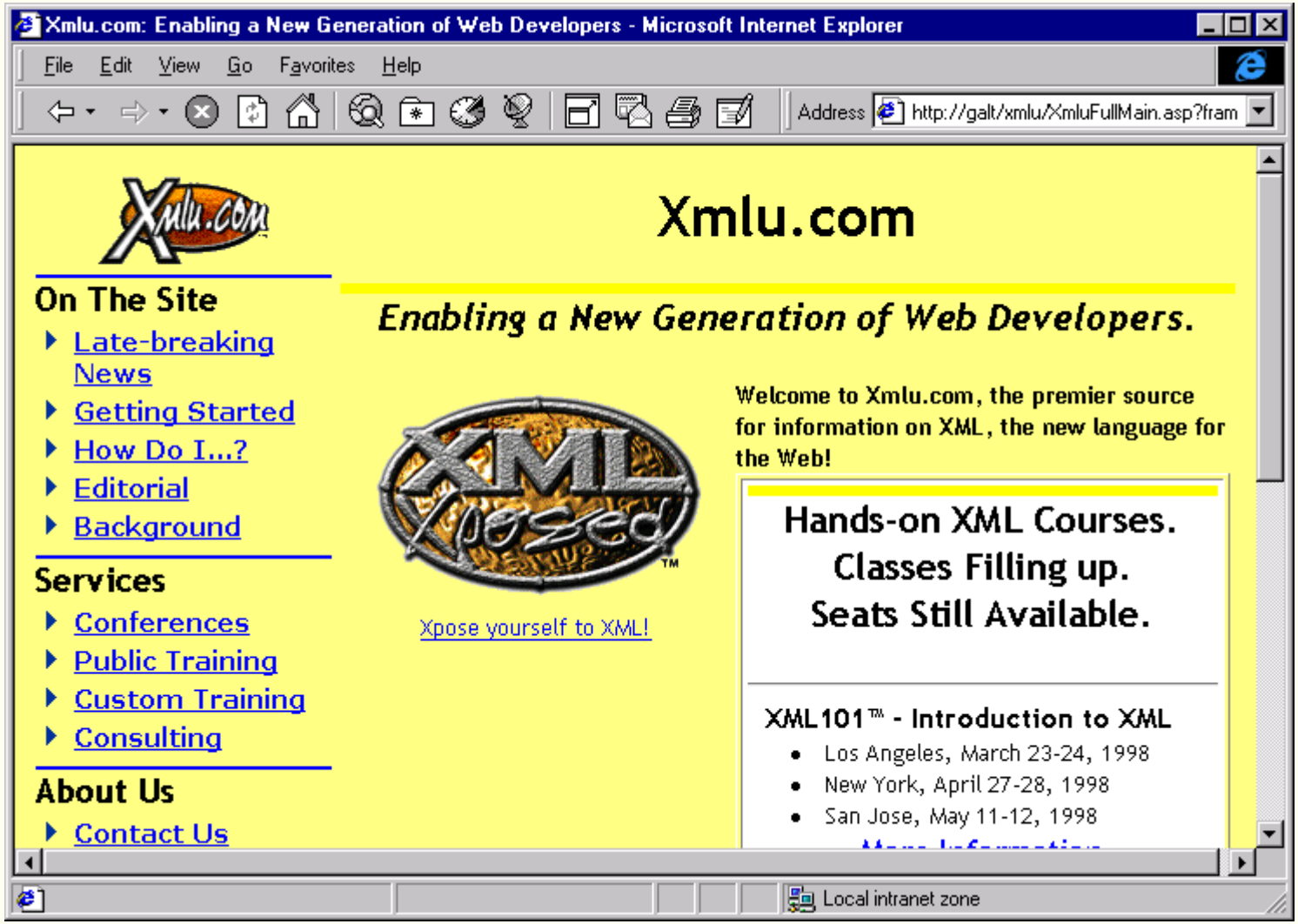


+

```
<style type='text/css'>
  BODY       {font-family:Trebuchet MS,Verdana,Arial;
              color:black;
              background-color:#FFFF80}
  A:link     {color: blue;}
  A:visited  {color: blue;}
  A:hover    {color: red;}
  center     {text-align:center;}
  TD         {font-size:10pt;bottom-margin:0;
              top-margin:0;}
  LI         {margin-left:0}
  H1         {text-align:center;
              font-size:24pt;
              font-weight:bold;}
  H2         {text-align:center;
              font-size:16pt;
              font-weight:bold;
              border-top:thick yellow solid;}
  H3         {font-size:14pt;
              font-weight:bold;
              border-top:thin blue solid;
              margin-bottom:0;
              margin-top:5}
</style>
```

=

CSS (Cascading Style Sheets)



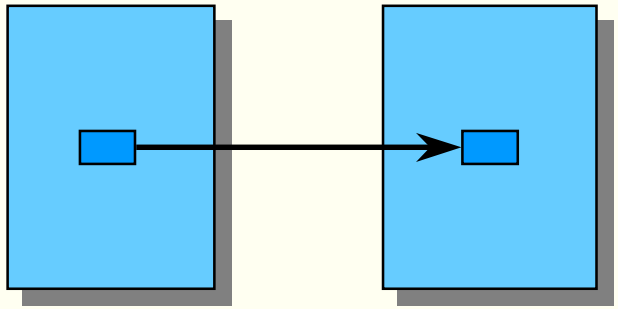
CSS vs. XSL

- CSS (**C**ascading **S**tyle **S**heets)
  - one-to-one mapping between an XML element and a built-in browser formatting object
    "sprinkling presentation hints on top of XML"
  - structure of output formatted document closely follows structure of input XML document
  - client-side only styling
- XSL (**E**xtensible **S**tylesheet **L**anguage)
  - XML element mapped to a formatting object consisting of multiple built-in formatting objects
    "defining detailed presentation using XML"
  - structure of output formatted document is not constrained by structure of input XML document
  - client-side and server-side styling

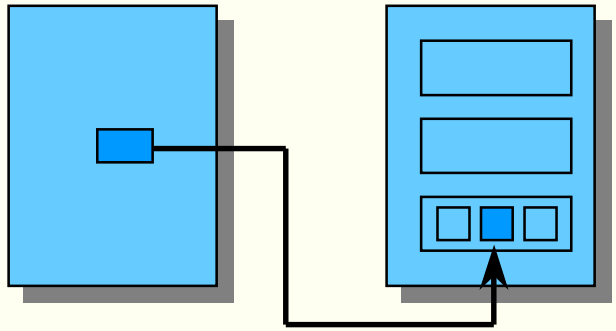
XLL: Extensible Linking Language

- XML linking mechanism: **XLink** and **XPointer** (using XPath)
  - subset of HyTime and TEI (Text Encoding Initiative)
  - compatible with existing URL linking
  - additional functionality:
    - any element can be a link
    - links can be bi-directional
    - links can be indirect
- Useful for:
  - rich hypertext functionality
  - Web site management
- Status:
  - XLink: W3C Proposed Recommendation (December 20, 2000)
  - XPointer: W3C Last Call Working Draft (January 8, 2001)

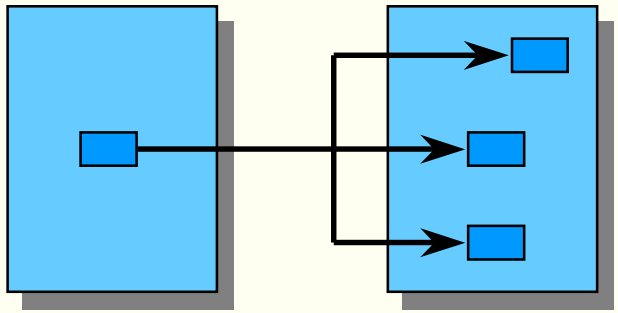
XLink: XML Linking Language



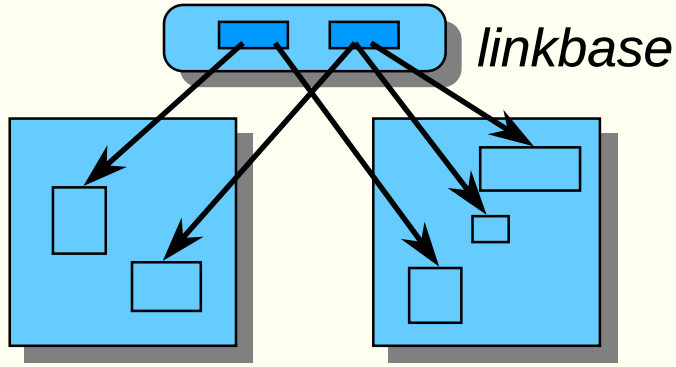
simple link



simple link (symbolic)

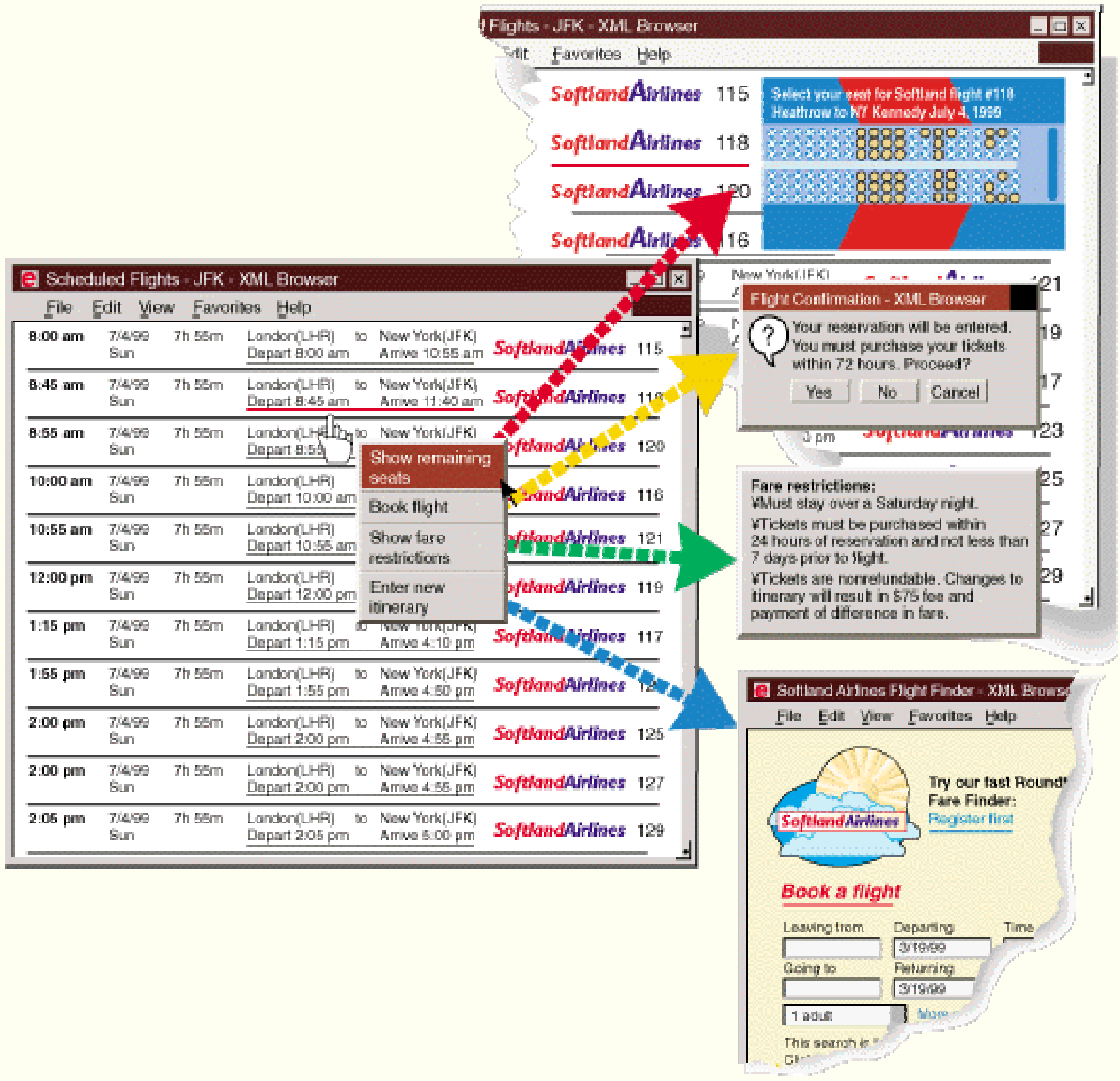


inline extended link



out-of-line extended link

Example of an extended link



insert an image

run a small program

show hidden text

link to another page

XPointer: XML Pointer Language

- Addressing scheme for pointing into an XML document
  - identifying locations in an XML document to be jumped to
  - identifying parts of an XML document to be embedded
- Syntax:

  `xpointer(`*`an XPath expression`*`)`

  to be used in an URL:

  `document.xml#xpointer(…)`
  - loads the whole document and jumps to the location being pointed to

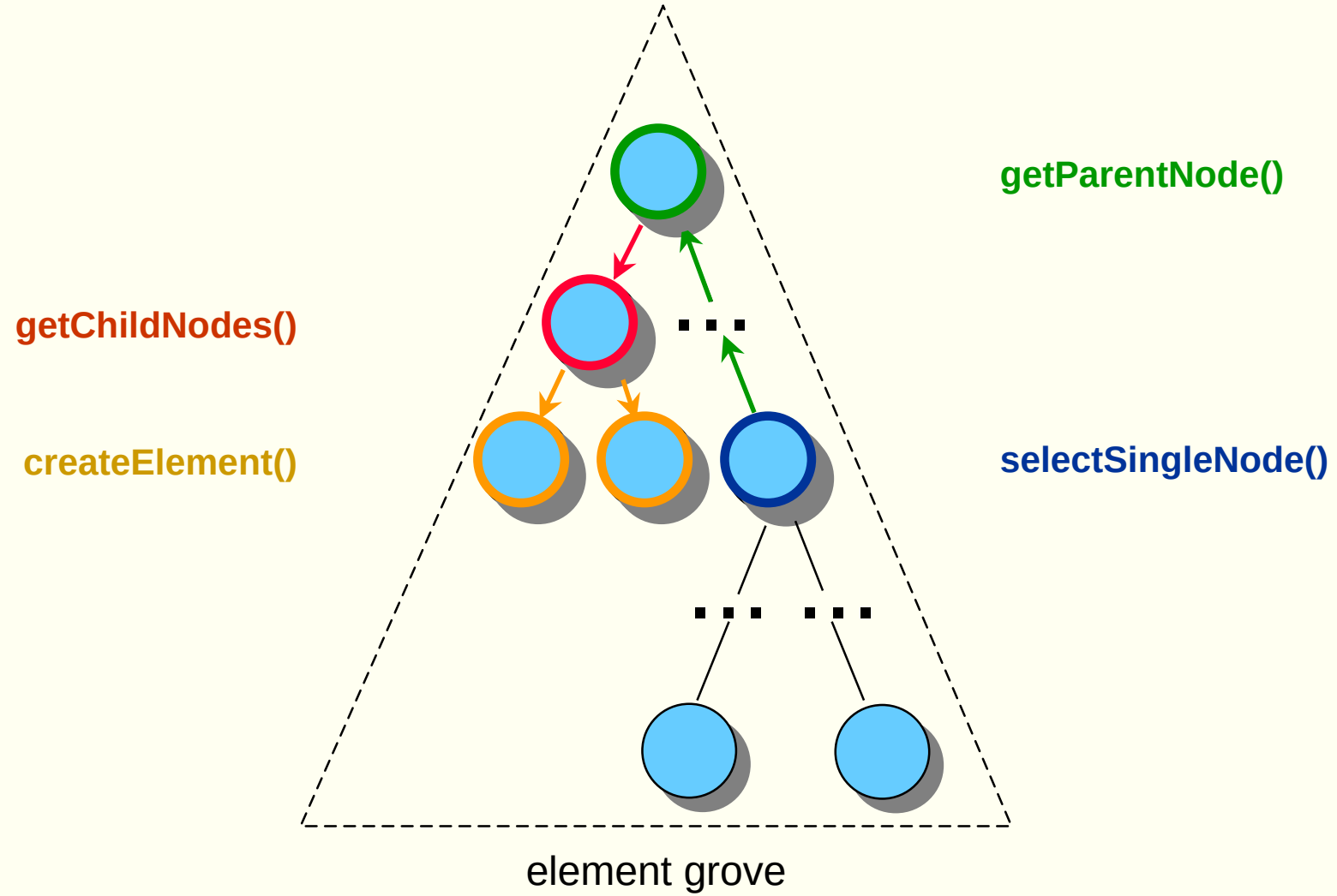
  `document.xml|xpointer(…)`
  - only loads the part of the document being pointed to
- Example:

  `xlink:href="document.xml#xpointer(/section[3]/para[2])"`

DOM: Document Object Model

- Standard API for XML documents
  - **Level 1**: access to document contents
  - **Level 2**: access to document stylesheet + mouse events + traversal
  - **Level 3**: loading/saving/serializing documents + presentation views + keyboard & device independent events
  - advanced capabilities:
    - expanding/collapsing text & dynamic tables of contents
    - client-side active documents
- Useful for:
  - editing XML documents and interacting with XML data structures
- Status:
  - Level 1: W3C Recommendation (October 1, 1998)
  - Level 2: W3C Recommendation (November 13, 2000)
  - Level 3: W3C Working Draft (February 9, 2001)

Example of a DOM tree manipulation



DOM: key concepts

- Document is a tree of objects
  - Document, Node, NodeList, NamedNodeMap, …
  - Element, Attribute, Text, Comment, Entity, …
- Language-independent standard API for navigating, querying and modifying these objects
  - navigating: getParentNode(), getChildNodes(), ...
  - querying: selectSingleNode(), selectNodes(), …
  - modifying: createElement(), createAttribute(), setAttribute(), …

  to be used in JavaScript, VBScript, …
- Alternative: **SAX** (**S**imple **A**PI for **X**ML)
  - for event-based parsing instead of building a parse tree
  - better for handling large XML documents, but it is harder to manipulate the structure of the XML documents

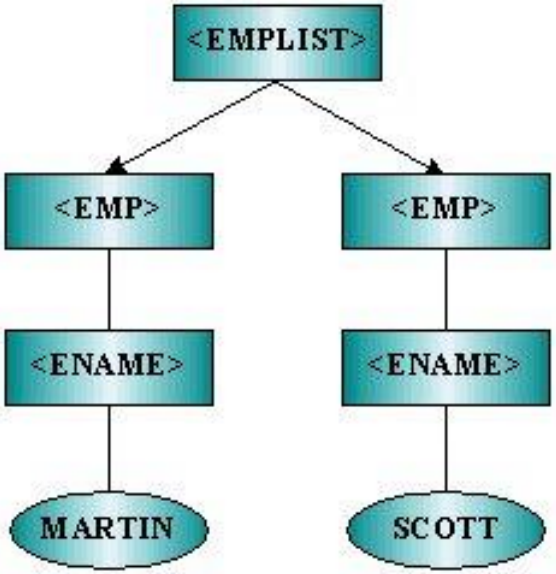
SAX vs. DOM processing

```
<?xml version="1.0"?>
  <EMPLIST>
    <EMP>
      <ENAME>MARTIN</ENAME>
    </EMP>
    <EMP>
      <ENAME>SCOTT</ENAME>
    </EMP>
  </EMPLIST>
```

SAX

```
start document
start element: EMPLIST
start element: EMP
start element: ENAME
characters: MARTIN
end element: EMP
start element: EMP
start element: ENAME
characters: SCOTT
end element: EMP
end element: EMPLIST
end document
```

DOM



SAX vs. DOM processing

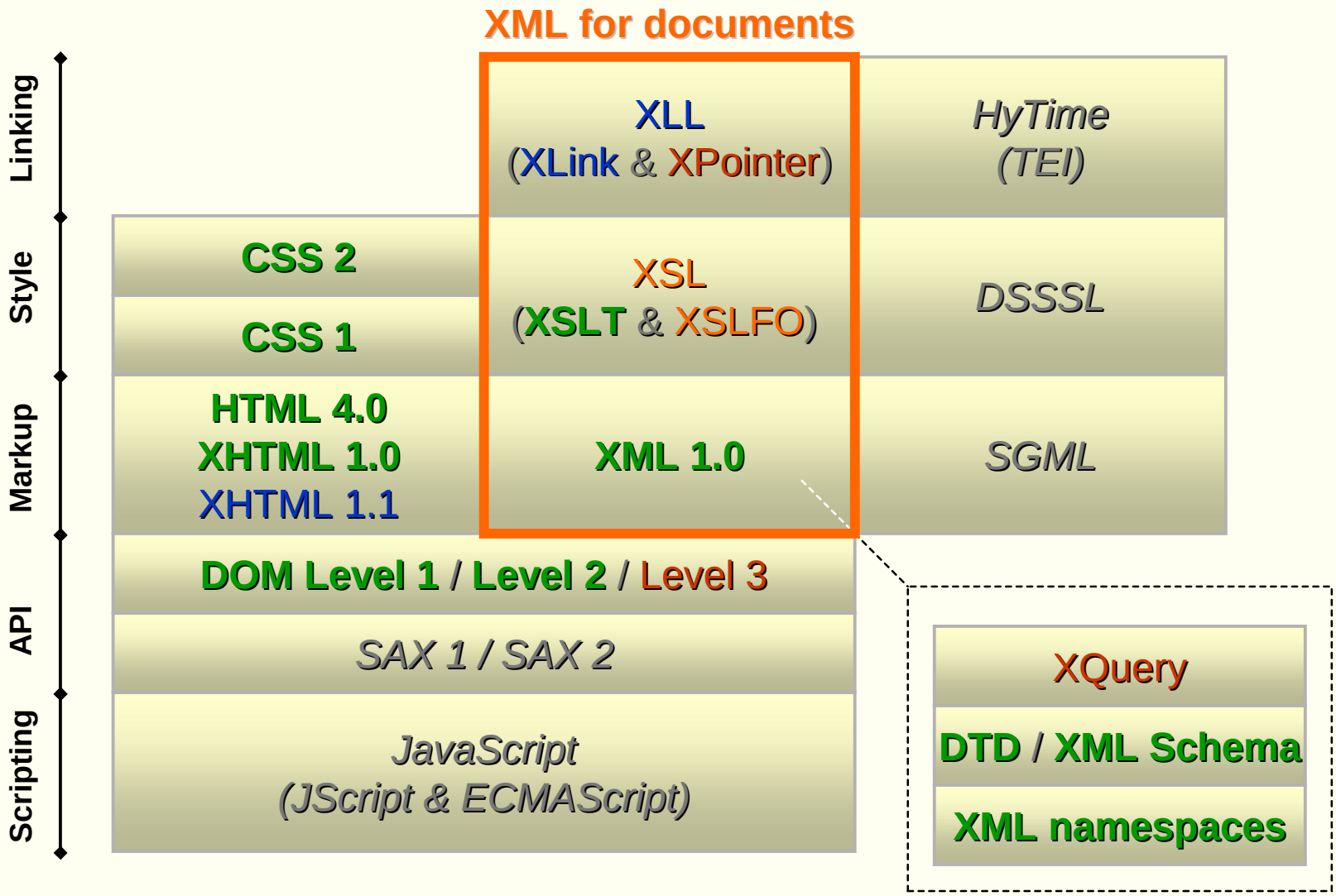
- Event-based API: SAX (**S**imple **A**PI for **X**ML)
  - uses *callbacks* to report parsing events to the application
  - application deals with these events through customized *event handlers*

  è use for: - batch processing, extracting from large volumes of data
  - when the parse tree does not have to manipulated
- Tree-based API: DOM (**D**ocument **O**bject **M**odel)
  - provides *objects* and *methods* to be used by the application
  - application uses these methods to navigate and manipulate the tree

  è use for: - interactive processing, modifying small volumes of data
  - adding / modifying / deleting elements and attributes
- SAX is faster, better suited for data (development-intensive)

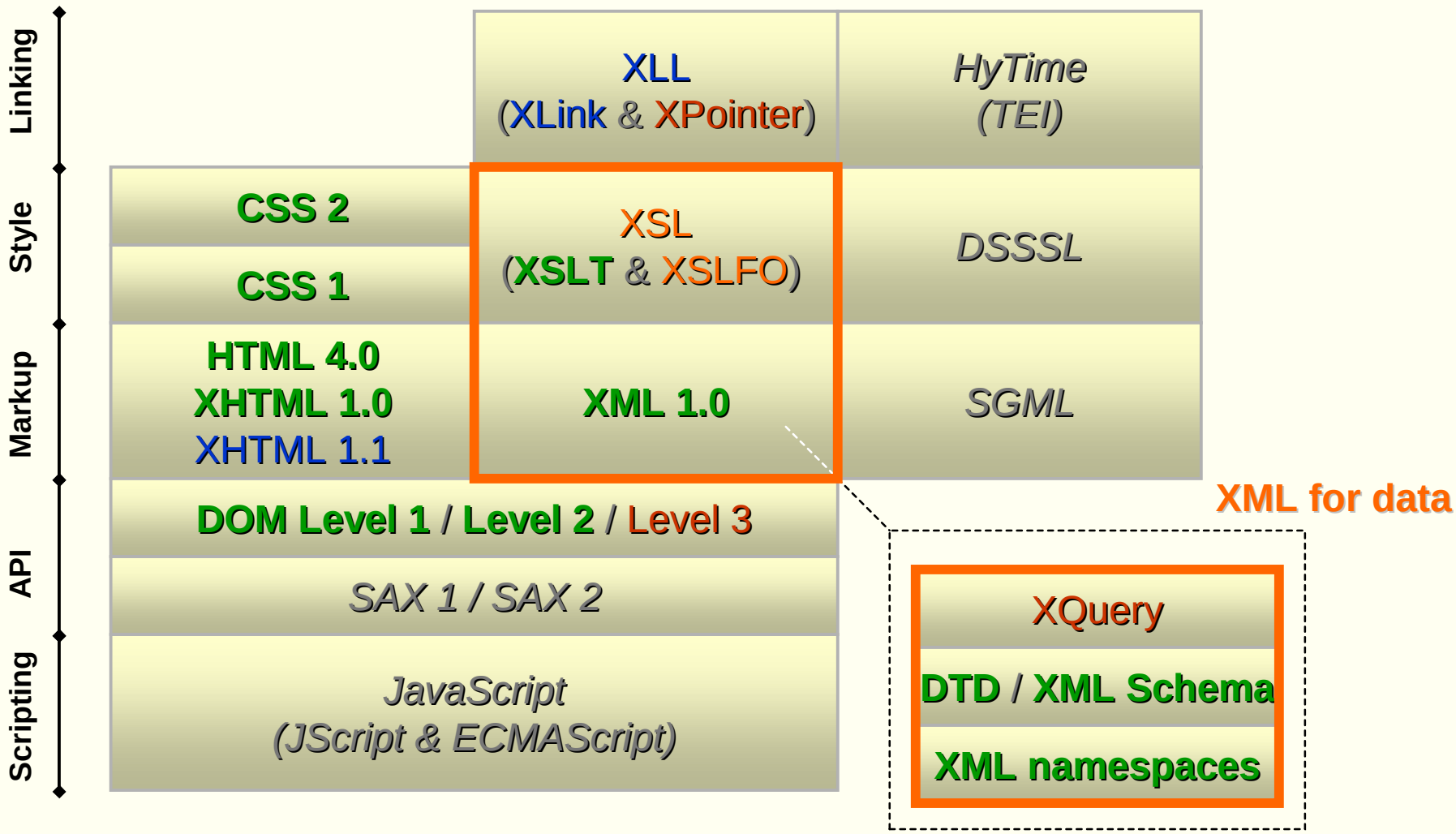
  DOM is more flexible, better suited for documents (memory-intensive)

The XML standards family



W3C: Note > Work. Draft > Cand. Recom. > Prop. Recom. > **Recommendation** / *Not W3C*

The XML standards family



W3C: Note > Work. Draft > Cand. Recom. > Prop. Recom. > **Recommendation** / *Not W3C*

Conclusions

- XML for documents standards:
  - XML: standard completed, commercial tools
  - XSL:
    - XSLT: standard completed, commercial tools
    - XSLFO: standard being refined, prototype tools
  - XLL:
    - XLink: standard almost completed, prototype tools
    - XPointer: standard just started, experimental tools
- XML for documents is as good as ready for industrial-strength, mission-critical usage
  - web site content management
  - electronic document management

… so it's already being widely used by developers

Conclusions

- XML for data standards:
  - XML: standard completed, commercial tools
  - XSL:
    - XSLT: standard completed, commercial tools
    - XSLFO: standard being refined, prototype tools
  - XML namespaces: standard completed, commercial tools
  - XML Schema: standard just completed, prototype tools
  - XQuery: standard just started, commercial tools only support XQL
- XML for data is not completely ready for industrial-strength, mission-critical usage
  - enterprise application integration
  - business-to-business e-commerce

… but it's being used in commercial tools and solutions anyway