CSE 510
Web Data Engineering

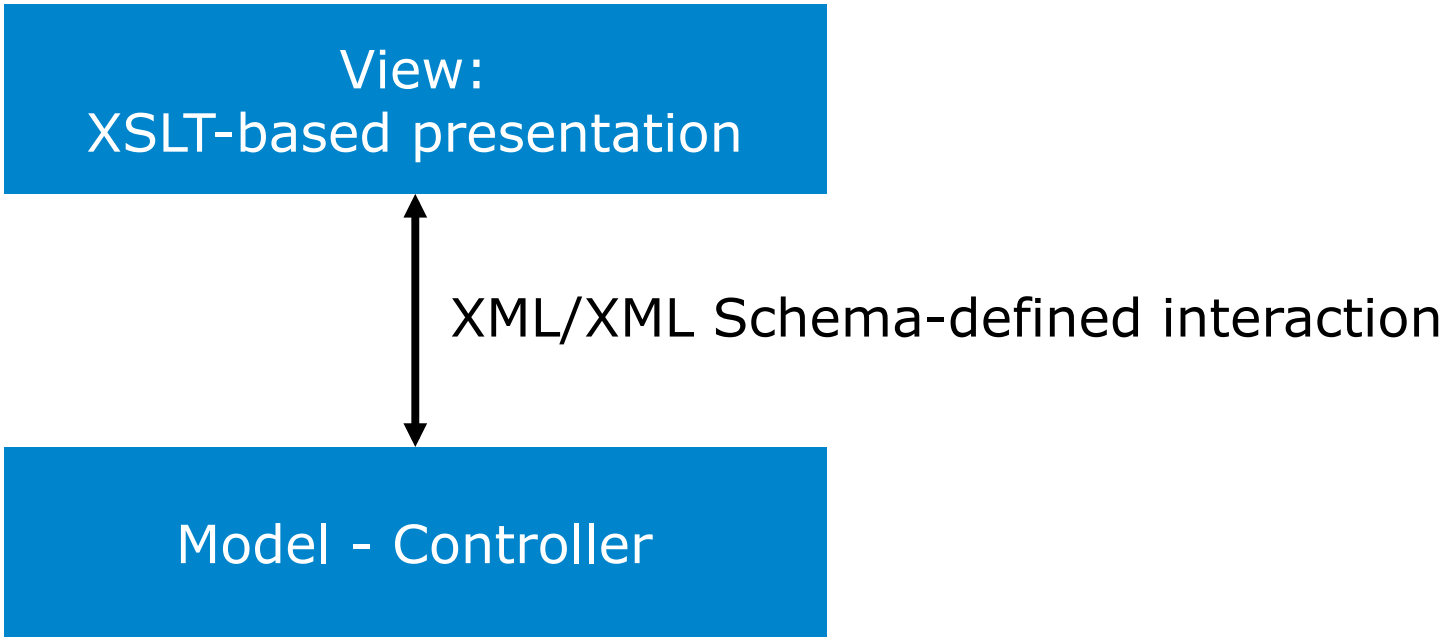
XML, XHTML, XSLT

and Web Application Programming

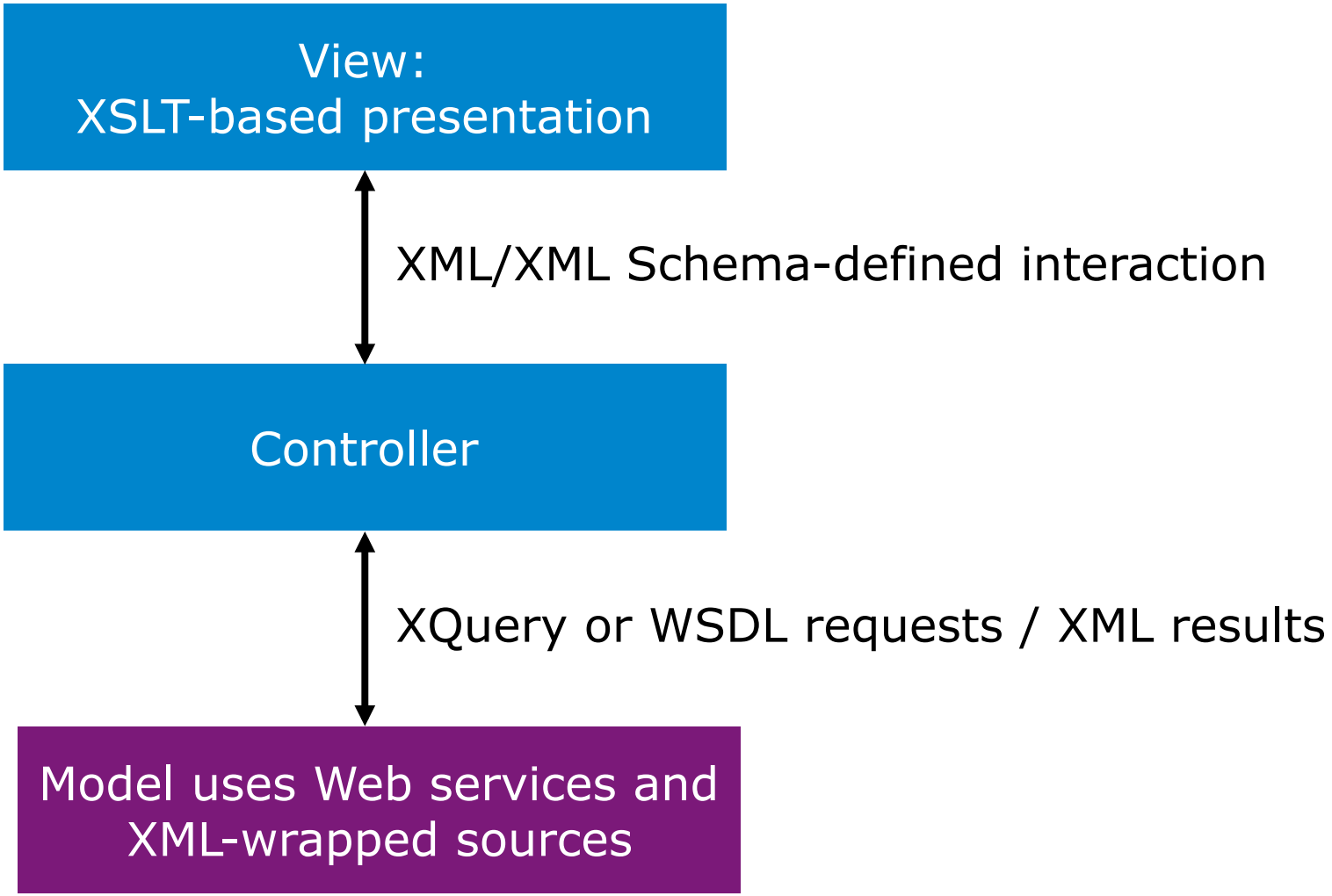
Evolving as We Speak…

cse@buffalo

XML-based Model 2 (MVC) Architecture of Today



XML-based Model 2 (MVC) Architecture of Tomorrow



XHTML: HTML Without Syntactic Sugar

- Tim Berners-Lee, Robert Cailliau, Jean-François Groff:
  The World-Wide Web.
  Computer Networks 25(4-5): 454-459 (1992)

```
<dt name="www">
   <img src="greenball.gif"/>
   Tim Berners-Lee, Robert Cailliau, Jean-François Groff:
   <a href="http://scholar.google.com/scholar?q=%22The
   +World%2DWide+Web%22">
        The World-Wide Web.
   </a>
   <i>Computer Networks 25(4-5): 454-459 (1992)</i>
</dt>
```

XML

Executive Summary:

- XML = XHTML + user-definable ("semantic") tags
- Separation of data and its presentation

=> Simple, very flexible data exchange format:

- semistructured data model

=> New applications:

- Information exchange (B2B), sharing (diglib), integration ("mediation"), archival, …
- Web application development:
  - Cleaner separation of View from Controller
  - Unifying data model for multiple information sources

What's Wrong with HTML ...

No Explicit Structure, Semantics, or Object-Orientation

```
<dt name="www">
   <img src="greenball.gif"/>
   Tim Berners-Lee, Robert Cailliau, Jean-François Groff:
   <a href="http://scholar.google.com/scholar?q=%22The
   +World%2DWide+Web%22">
        The World-Wide Web.
   </a>
   <i>Computer Networks 25(4-5): 454-459 (1992)</i>
</dt>
```

Author

Title

Conference

… and Some Repercussions

=> HTML is inappropriate for

- Data exchange
- Separation of logical from presentation aspects of a Web application's view

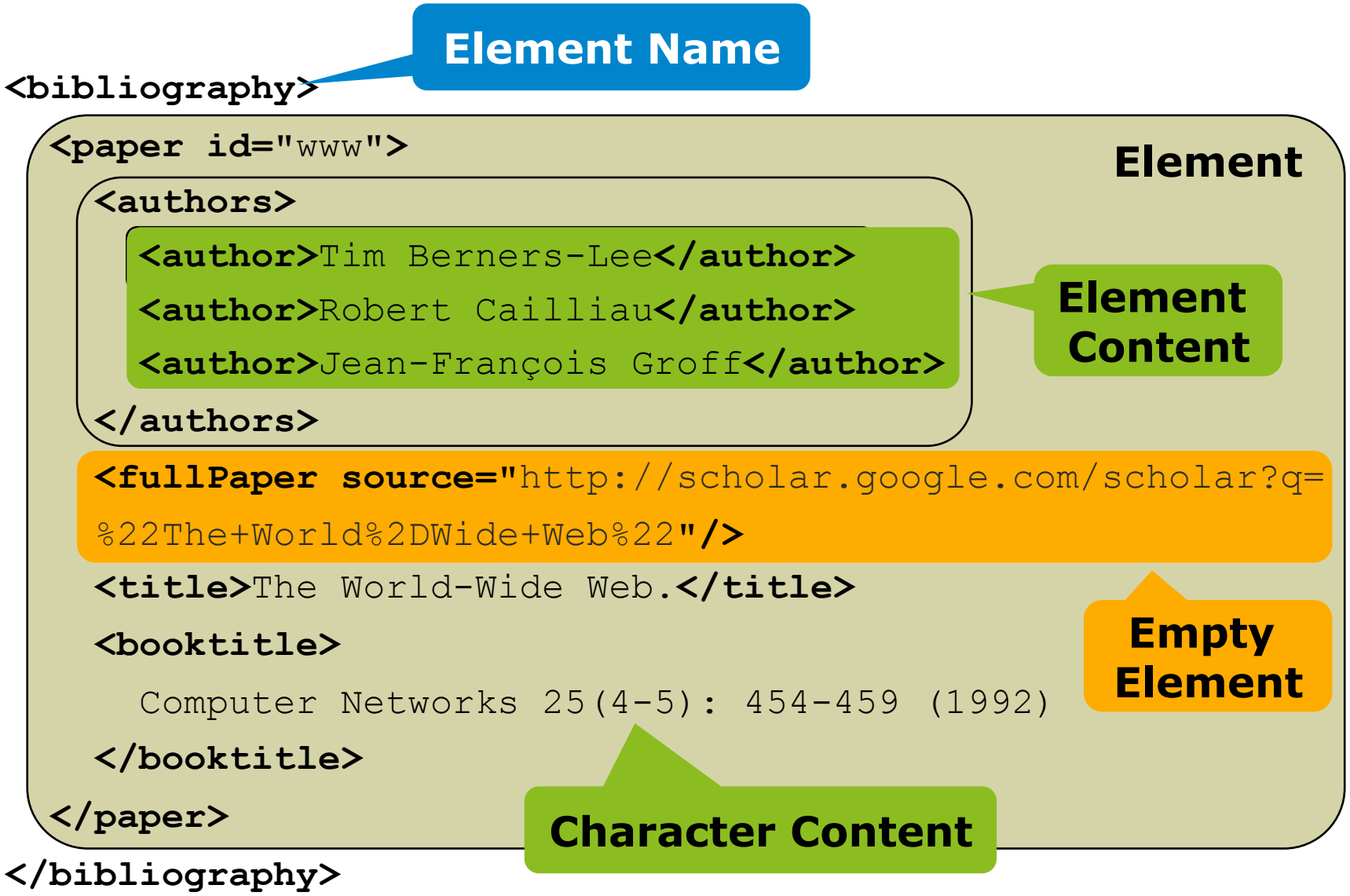
XML is Based on Markup

```
<bibliography>
  <paper id="www">
    <authors>
      <author>Tim Berners-Lee</author>
      <author>Robert Cailliau</author>
      <author>Jean-François Groff</author>
    </authors>
    <fullPaper source="http://scholar.google.com/scholar?q=
    %22The+World%2DWide+Web%22"/>
    <title>The World-Wide Web.</title>
    <booktitle>
      Computer Networks 25(4-5): 454-459 (1992)
    </booktitle>
  </paper>
</bibliography>
```

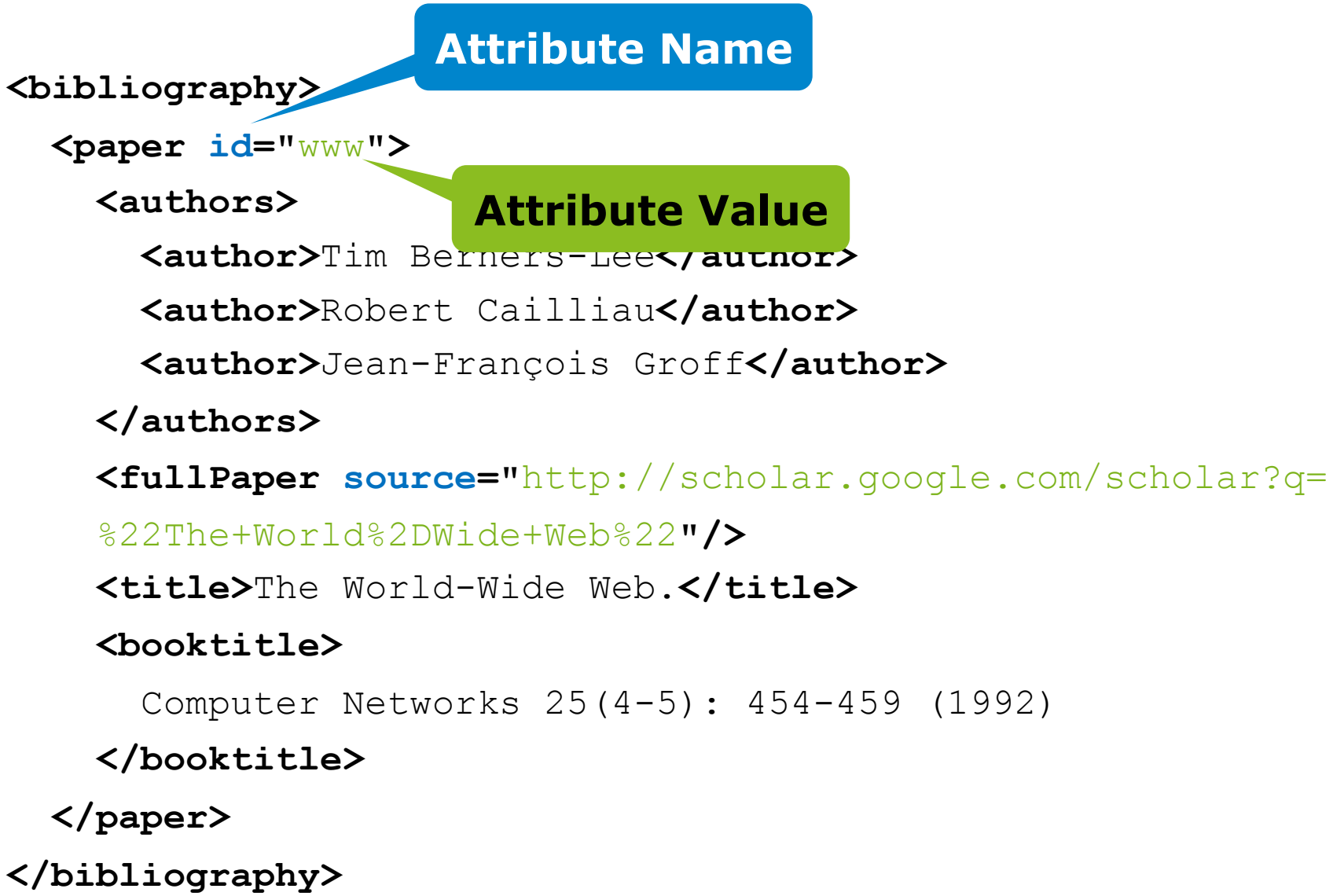
Markup indicates structure and semantics

Decoupled from presentation

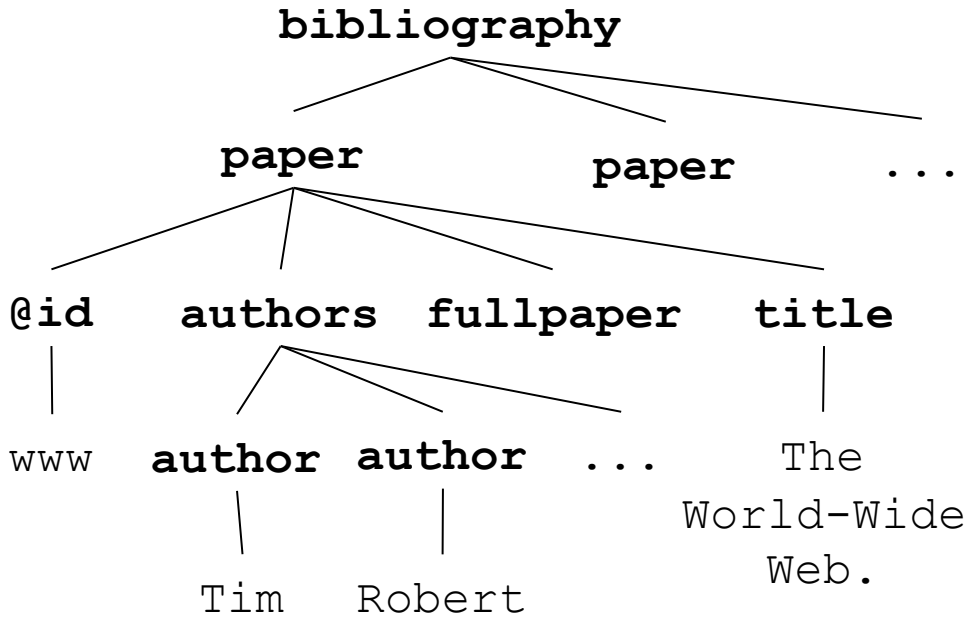
Elements and their Content



Element Attributes



XML, XHTML = Labeled Ordered Trees




```
<bibliography>
  <paper id="www">
    <authors>
      <author>Tim</author>
      <author>Robert</author>
      ...
    </authors>
    <title>
      The World-Wide Web.
    </title>
    ...
  </paper>
</bibliography>
```

≈ **semistructured data**

≈ **labeled trees/graphs**

can also represent

- relational and
- object-oriented data

Attributes

```
<person id="tim"> Tim's info </person>

<bibliography>
  <paper id="www" role="publication">

    <authors>
      <author authorRef="tim">
       Tim Berners-Lee</author>
    </authors>
    <fullPaper source="wwwPaper"/>
    <title>The World-Wide Web.</title>
    <related papers="brow rs html"/>
  </paper>
</bibliography>
```

Object Identity Attribute

CDATA (character data)

IDREF intra-document reference

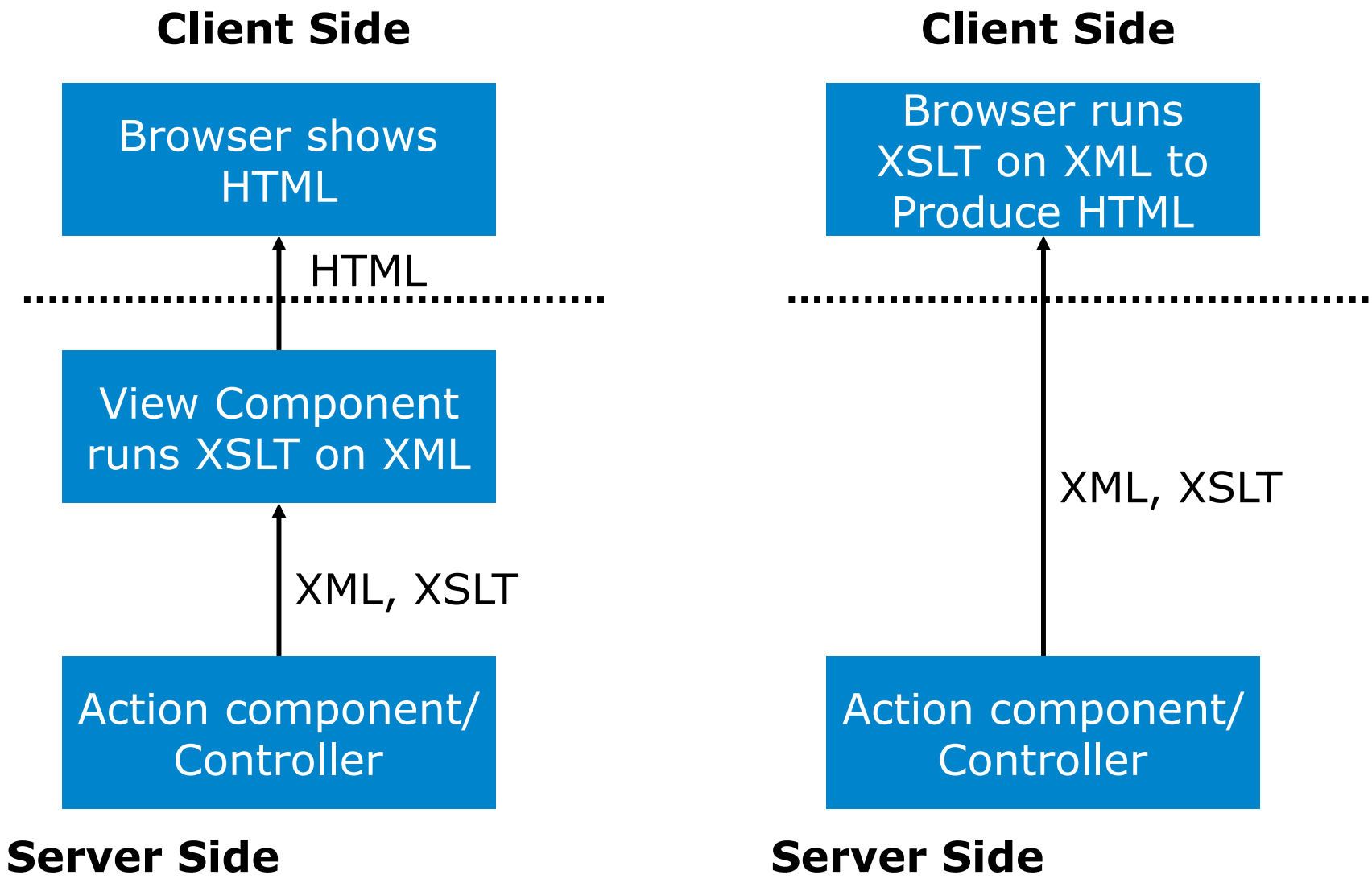
Reference to external ENTITY

Turning XML into HTML: XSLT

- Why Stylesheets?
  - Separation of content (XML) from presentation (XSL)

- Why not just CSS for XML?
  - XSL is far more powerful:
    - **selecting** elements
    - **transforming** the XML tree
    - **content** based display (result may depend on data)

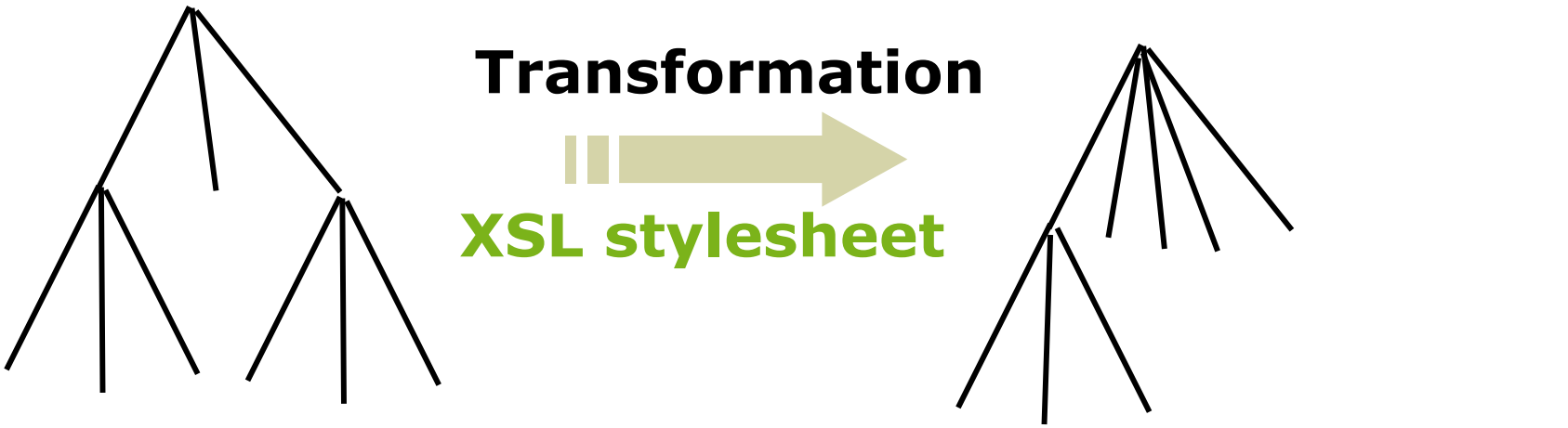
Using XSLT for View Components



XSLT Overview

- XSLT stylesheets are denoted in **XML syntax**
- XSL components:
  1. A language for **transforming** XML documents (**XSLT**: integral part of the XSL specification)
  2. An XML **formatting vocabulary** (Formatting Objects: >90% of the formatting properties inherited from CSS)

XSLT Processing Model



XML source tree

XML,HTML,… result tree

XSLT Processing Model

- **XSL stylesheet**: collection of **template rules**
- **template rule**: (**pattern** ⇒ **template**)
- main steps:
  - **match pattern** against source tree
  - **instantiate template** (replace current node "**.**" by the template in the result tree)
  - **select** further nodes for processing
- control can be
  - **program-driven** ("pull": <xsl:foreach> ...)
  - **data/event-driven** ("push": <xsl:apply-templates> ...)

Template Rule: Example

Pattern

Template

```
<xsl:template match="product">
  <table>
    <xsl:apply-templates select="sales/domestic"/>
  </table>
  <table>
    <xsl:apply-templates select="sales/foreign"/>
  </table>
</xsl:template>
```

1. **match pattern**: process <product> elements
2. **instantiate template**: replace each product with two HTML tables
3. **select** the <product> grandchildren ("sales/domestic", "sales/foreign") for **further processing**

Example of Turning XML into HTML

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="FitnessCenter.xsl"?>
<FitnessCenter>
    <Member level="platinum">
        <Name>Jeff</Name>
        <Phone type="home">555-1234</Phone>
        <Phone type="work">555-4321</Phone>
        <FavoriteColor>lightgrey</FavoriteColor>
    </Member>
</FitnessCenter>
```

HTML Document in an XSL Template

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head><title>Welcome</title></head>
      <body>
        Welcome!
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Extracting the Member Name

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head><title>Welcome</title></head>
      <body>
        Welcome
        <xsl:value-of select="/FitnessCenter/Member/Name"/>!
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

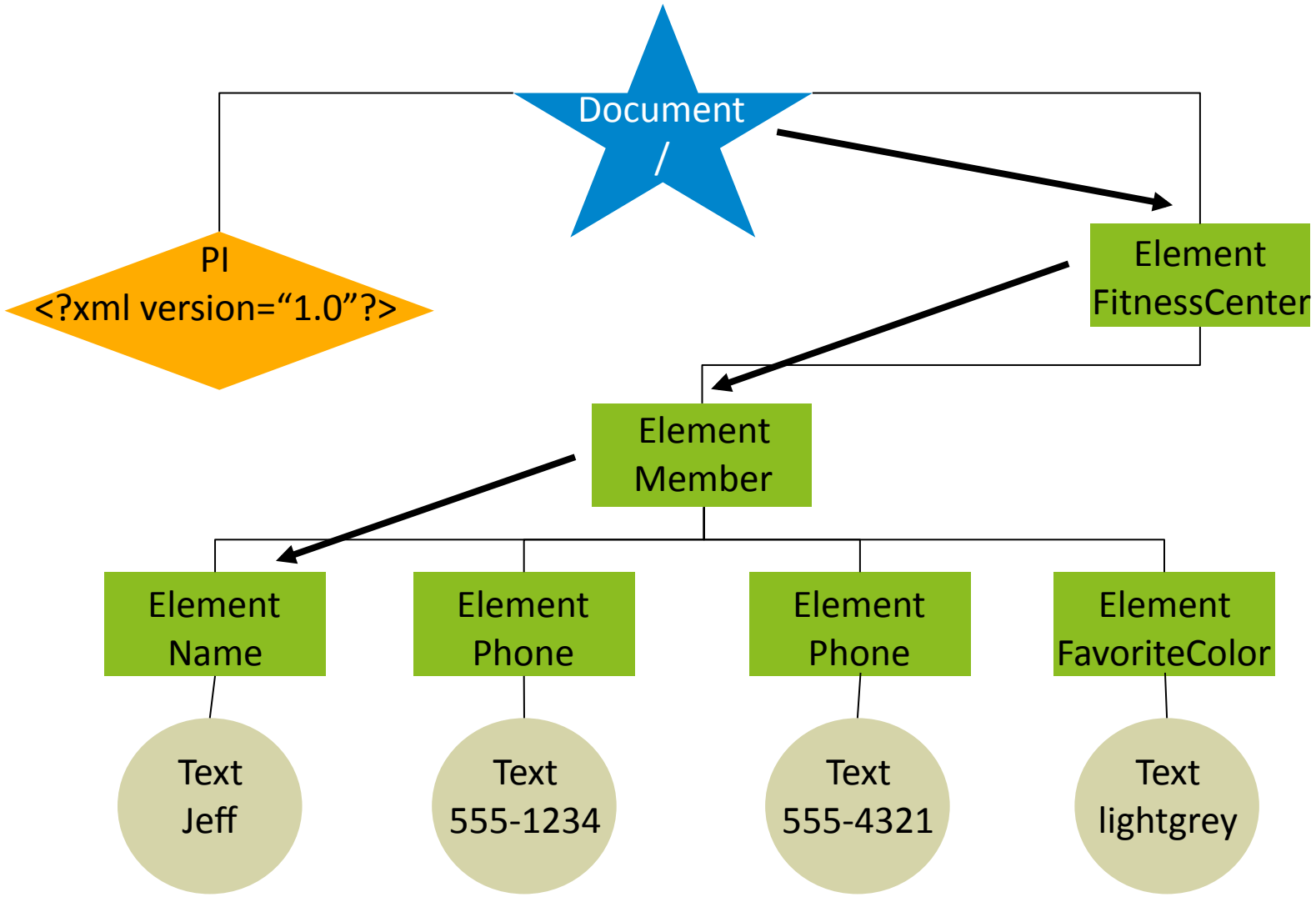
Navigating the XML Document

- Extracting values from an XML Document:
  - use the <xsl:value-of select="…"/> XSL element
- Navigating:
  - The slash ("/") indicates parent/child relationship
  - A slash at the beginning of the path indicates that it is an absolute path, starting from the top of the XML document

```
/FitnessCenter/Member/Name
```

Start from the top of the XML document, go to the FitnessCenter element, from there go to the Member element, and from there go to the Name element

XML DOM Tree


Document
/
PI
<?xml version="1.0"?>
Element
FitnessCenter
Element
Member
Element
Name
Element
Phone
Element
Phone
Element
FavoriteColor
Text
Jeff
Text
555-1234
Text
555-4321
Text
lightgrey

Extract FavoriteColor and Use It As bgcolor

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head><title>Welcome</title></head>
      <body bgcolor="{/FitnessCenter/Member/FavoriteColor}">
        Welcome
        <xsl:value-of select="/FitnessCenter/Member/Name"/>!
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Note

- Attribute values cannot contain "<" nor ">"
  - Consequently, the following is NOT valid:

```
<body bgcolor="
  <xsl:value-of
       select='/FitnessCenter/Member/FavoriteColor'
  />
">
```

- To extract the value of an XML element and use it as an attribute
- value you must use curly braces:

```
<body bgcolor="{/FitnessCenter/Member/FavoriteColor}">
```

Evaluate the expression within the curly braces, and assign the value to the attribute

Extract the Home Phone Number

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head><title>Welcome</title></head>
      <body bgcolor="{/FitnessCenter/Member/FavoriteColor}">
        Welcome
        <xsl:value-of select="/FitnessCenter/Member/Name"/>!
        <br />
        Your home phone number is:
        <xsl:value-of
         select="/FitnessCenter/Member/Phone[@type='home']"/>
      ...
```

Creating the Result Tree

- Further XSL elements for
  - Numbering
    - `<xsl:number value="position()" format="1">`
  - Conditions
    - `<xsl:if test="position() mod 2 = 0">`
  - Repetition…

More on XSL

- XSL(T):
  - Conflict resolution for multiple applicable rules
  - Modularization **<xsl:include> <xsl:import>**
  - …
- XSL Formatting Objects
  - a la CSS
- XPath (navigation syntax + functions)

  = XSLT ∩ XPointer
- …