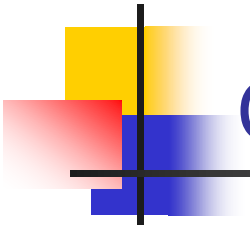
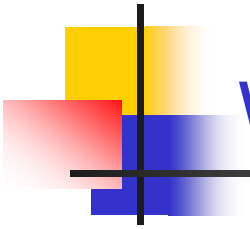
XML Schema



Outline

- XML Schema Overview
- XML Schema Components
- XML Schema Reusability & Conformance
- XML Schema Applications and IDE

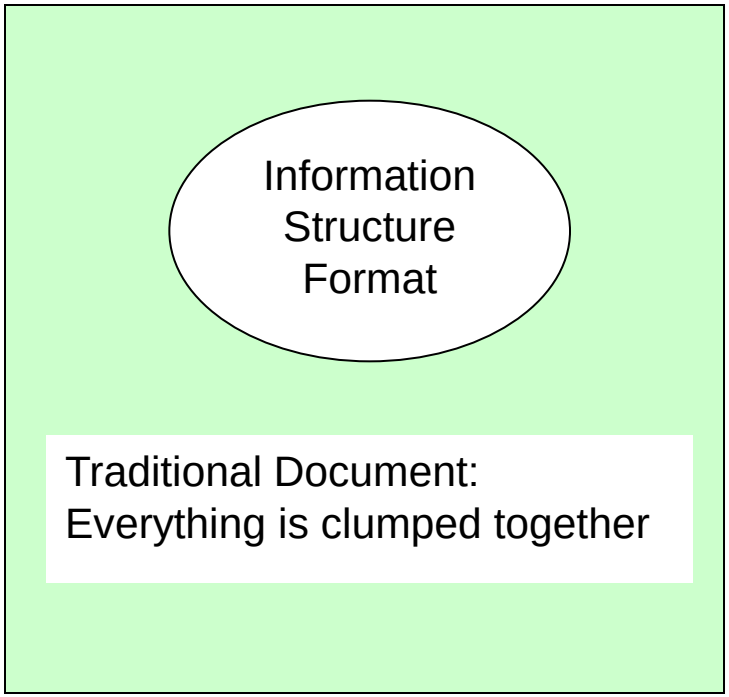


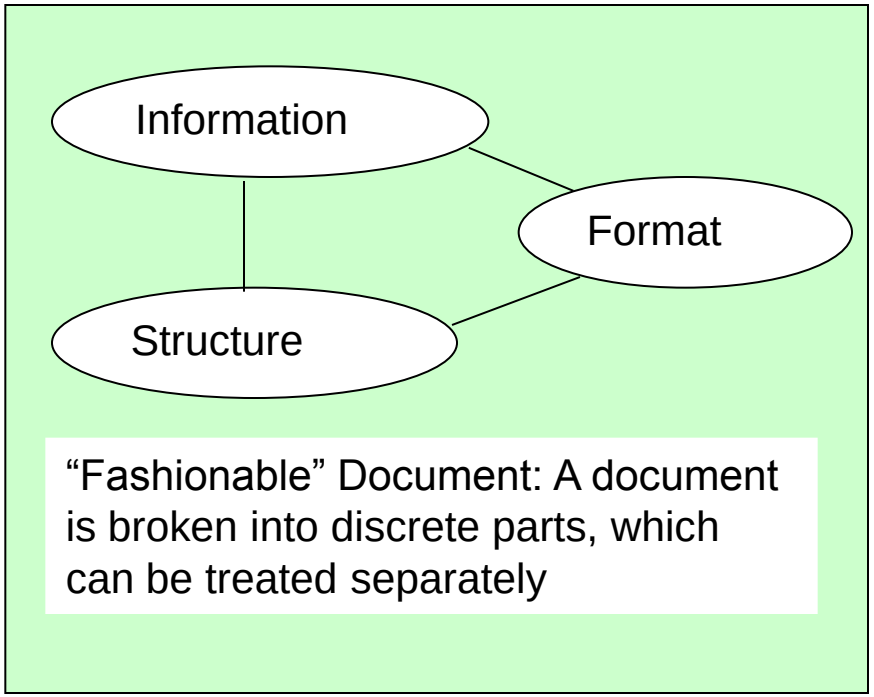
What is XML Schema?

- The origin of schema
  - XML Schema documents are used to define and validate the content and structure of XML data.
  - XML Schema was originally proposed by Microsoft, but became an official W3C recommendation in May 2001

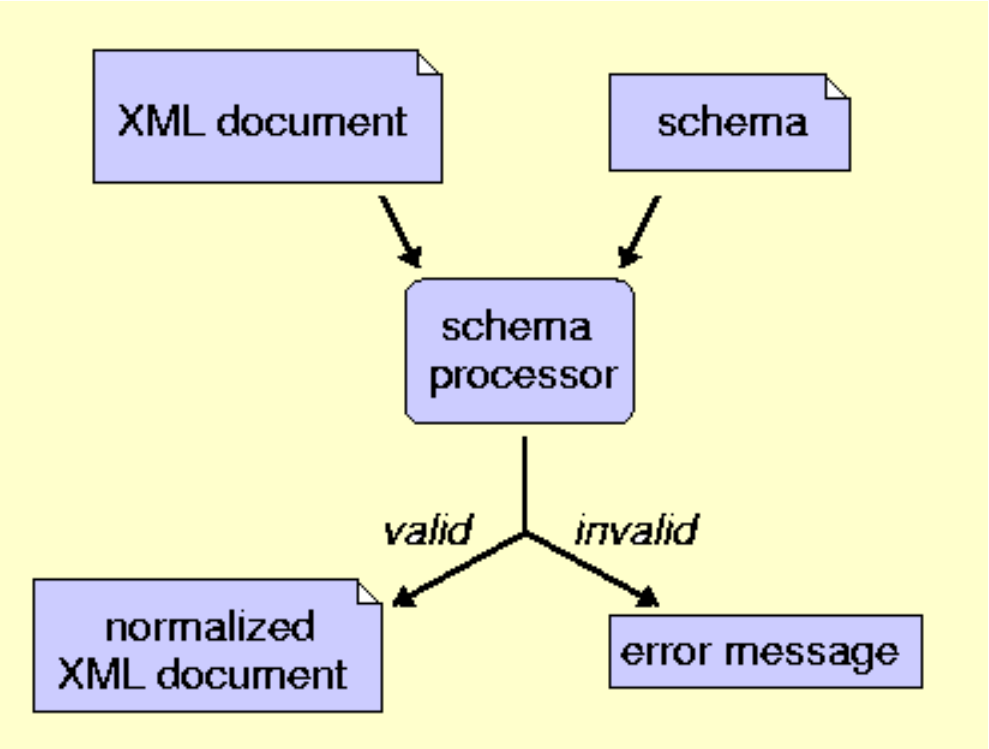
Why Schema? (1)

Separating Information from Structure and Format

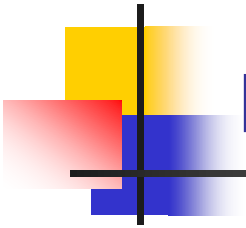




Why Schema? (2)



Schema Workflow



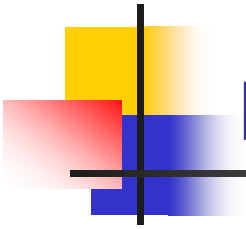
DTD versus Schema

Limitations of DTD

- No constraints on character data
- Not using XML syntax
- No support for namespace
- Very limited for reusability and extensibility

Advantages of Schema

- Syntax in XML Style
- Supporting Namespace and import/include
- More data types
- Able to create complex data type by inheritance
- Inheritance by extension or restriction
- More …



Problems of XML Schema

- General Problem
  - Several-hundred-page spec in a very technical language
- Practical Limitations of expressibility
  - content and attribute declarations cannot depend on attributes or element context.
- Technical Problem
  - The notion of “type” adds an extra layer of confusing complexity
- …

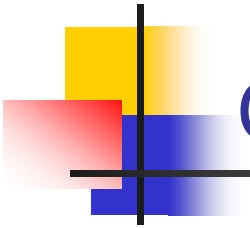
An XML Instance Document Example

```
<book isbn="0836217462">
    <title> Being a Dog Is a Full-Time Job</title>
    <author>Charles M. Schulz</author>
    <qualification>    extroverted beagle  </qualification>
</book>
```

The Example's Schema

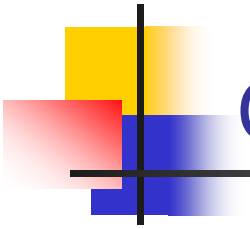
```
<?xml version="1.0" encoding="utf-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="book">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="qualification" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
</xs:schema>
```

book.xsd



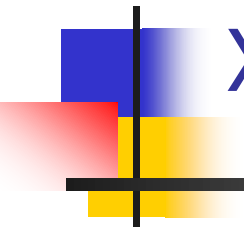
Outline

- XML Schema Overview
- XML Schema Components
- XML Schema Reusability & Conformance
- XML Schema Applications and IDE



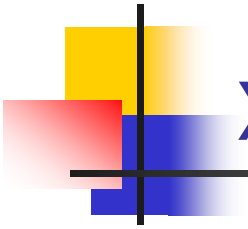
Outline

- XML Schema Overview
- XML Schema Components
- XML Schema Reusability & Conformance
- XML Schema Applications and IDE



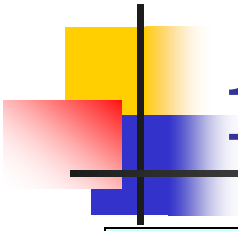
XML Schema Components

- Abstract Data Model
- Simple and Complex Type Definitions
- Declarations
- Relationship among Schema Components



XML Abstract Data Model

- The XML Abstract Data Model
  - composes of Schema Components.
  - is used to describe XML Schemas.
- Schema Component
  - is the generic term for the building blocks that compose the abstract data model of the schema.



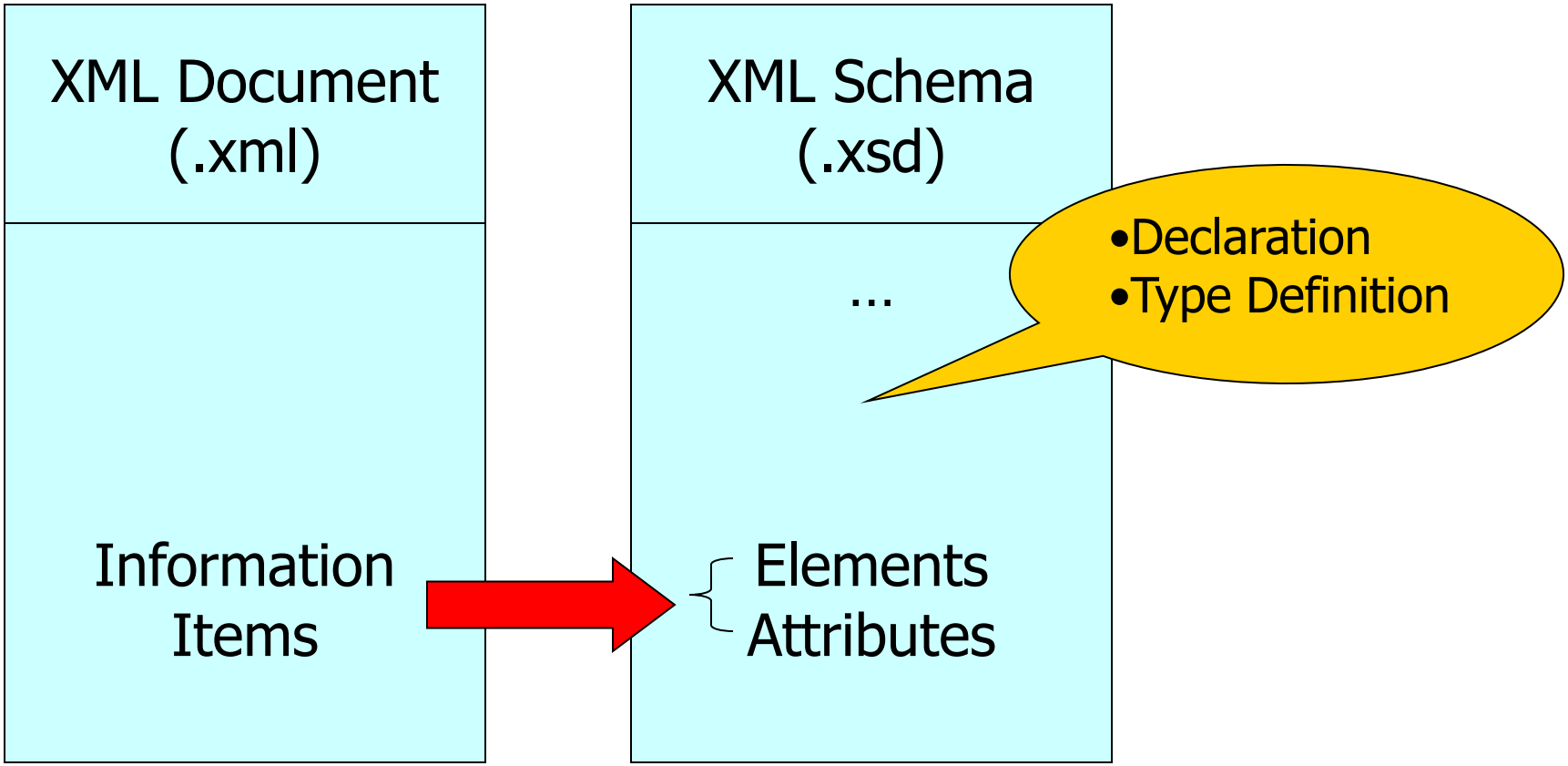
13 Kinds of Schema Components

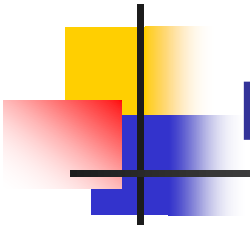
- Simple type definitions
- Complex type definitions
- Attribute declarations
- Element declarations

- Attribute group definitions
- Identity-constraint definitions
- Model group definitions
- Notation declarations

- Annotations
- Model groups
- Particles
- Wildcards
- Attribute Uses

XML document & XML Schema

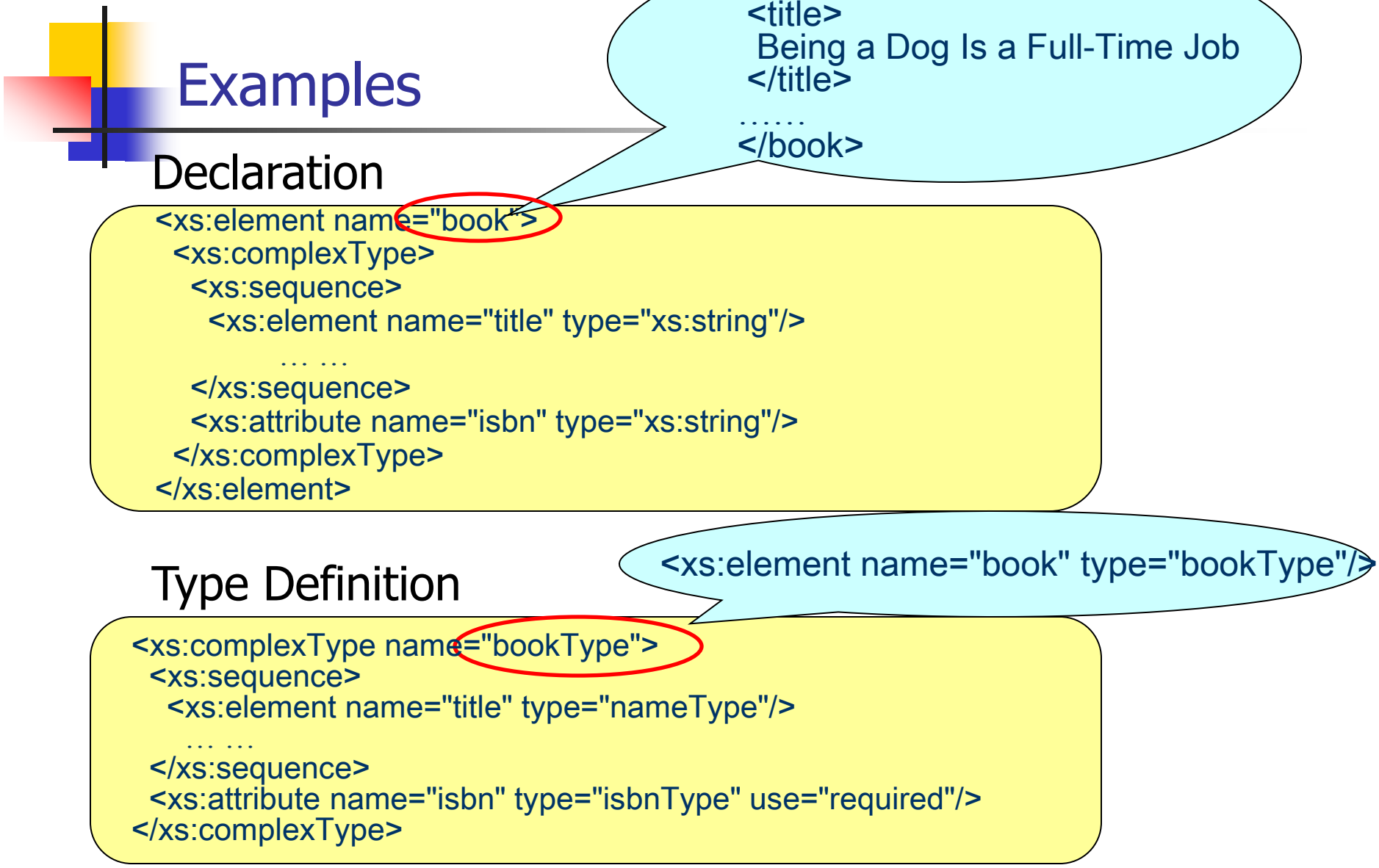




Declaration & Definition

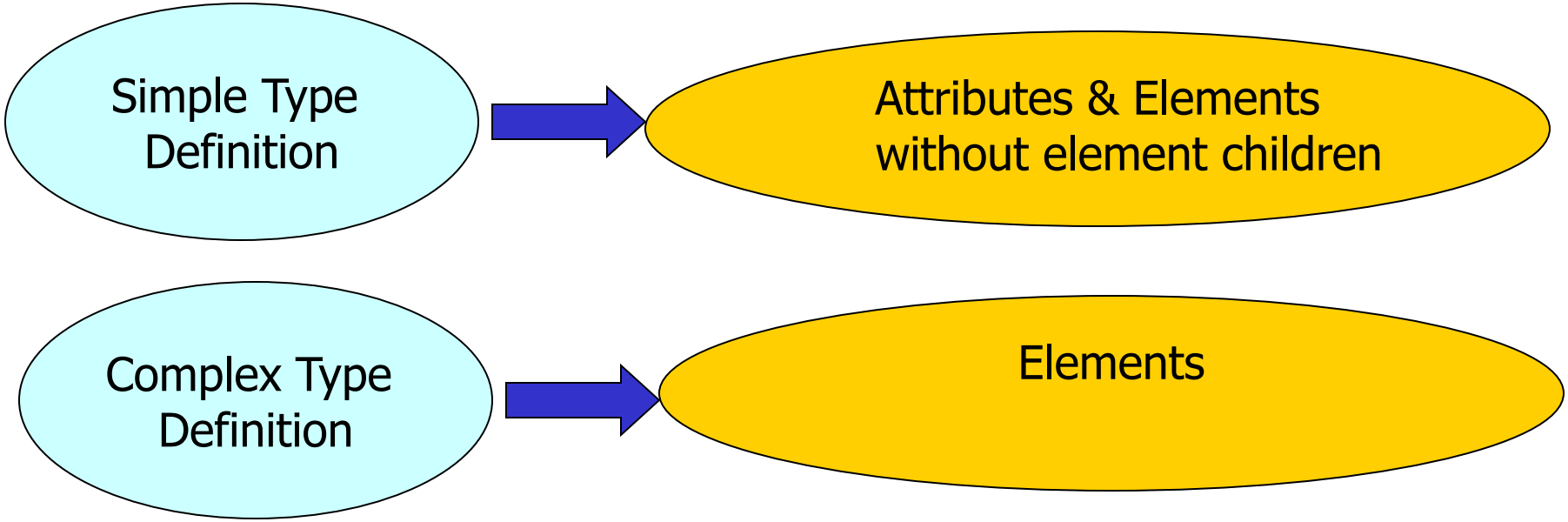
- Declaration Components
  - are associated by (qualified) names to information items being validated.
  - It is like declaring objects in OOP.
- Definition Components
  - define internal schema components that can be used in other schema components.
  - Type definition is like defining classes in OOP.

Examples


<book isbn="0836217462">
<title>
Being a Dog Is a Full-Time Job
</title>
……
</book>
Declaration
<xs:element name="book">
<xs:complexType>
<xs:sequence>
<xs:element name="title" type="xs:string"/>
… …
</xs:sequence>
<xs:attribute name="isbn" type="xs:string"/>
</xs:complexType>
</xs:element>
<xs:element name="book" type="bookType"/>
Type Definition
<xs:complexType name="bookType">
<xs:sequence>
<xs:element name="title" type="nameType"/>
… …
</xs:sequence>
<xs:attribute name="isbn" type="isbnType" use="required"/>
</xs:complexType>

Type Definitions

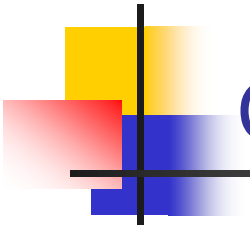
- Why Type Definitions?
- Simple Type Definition VS. Complex Type Definition



Simple Type Definition

- Simple Type Definition can be:
  - a restriction of some other simple type;
  - a list or union of simple type definition;or
  - a built-in primitive datatypes.
- Example

```
<xs:simpleType name="farenheitWaterTemp">
 <xs:restriction base="xs:number">
  <xs:fractionDigits value="2"/>
  <xs:minExclusive value="0.00"/>
  <xs:maxExclusive value="100.00"/>
 </xs:restriction>
</xs:simpleType>
```



Complex Type Definition(1)

- Inheritance
  - Restriction: We restrict base types definitions.
  - Extension: We add something new.
- Composition
  - Group model

Complex Type Definition(2)

- Inheritance

  Each complex type definition is either

  - a restriction of a complex type definition
  - an extension of a simple or complex type definition
  - a restriction of the ur-type definition.

- Example

```
<xs:complexType name="personName">
 <xs:sequence>
  <xs:element name="title" minOccurs="0"/>
        … …
 </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="extendedName">
 <xs:complexContent>
  <xs:extension base="personName">
   <xs:sequence>
    <xs:element name="generation"
                minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Complex Type Definition(3)

- Composition

  Model Group is composed of

  - Compositor (*sequence* |*choice*| *all*)
  - Particles, which can be
    - Element Declaration
    - Wildcard
    - Model Group

Complex Type Definition(4)

- Model Group Examples

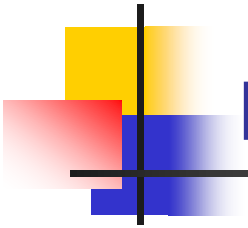
```
<xs:all>
 <xs:element ref="cats"/>
 <xs:element ref="dogs"/>
</xs:all>
```

```
<xs:sequence>
 <xs:choice>
  <xs:element ref="left"/>
  <xs:element ref="right"/>
 </xs:choice>
 <xs:element ref="landmark"/>
</xs:sequence>
```

Complex Type Definition(5)

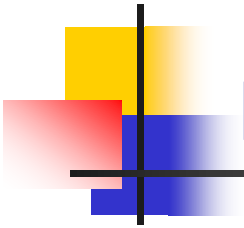
- Reusable Fragments of Type Definition
  - Model group definition
  - Attribute group definition

```
<xs:group name="mainBookElements">
  <xs:sequence>
    <xs:element name="title" type="nameType"/>
    <xs:element name="author" type="nameType"/>
  </xs:sequence>
</xs:group>
<xs:complexType name="bookType">
  <xs:sequence>
    <xs:group ref="mainBookElements"/>
      … …
  </xs:sequence>
  <xs:attributeGroup ref="bookAttributes"/>
</xs:complexType>
```



Declarations(1)

- Element Declaration
- Attribute Declaration
  - Attribute uses
- Notation Declaration
  - Notation declarations reconstruct XML 1.0 NOTATION declarations.



Declarations(2)

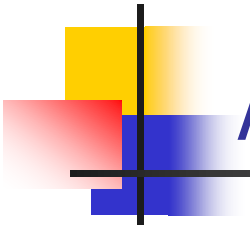
Examples

```
<xs:element name="PurchaseOrder" type="PurchaseOrderType"/>
```

```
<xs:element name="gift">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="birthday" type="xs:date"/>
   <xs:element ref="PurchaseOrder"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<xs:attribute name="age" type="xs:positiveInteger" use="required"/>
```

```
 <xs:notation name="jpeg" public="image/jpeg" system="viewer.exe">
```

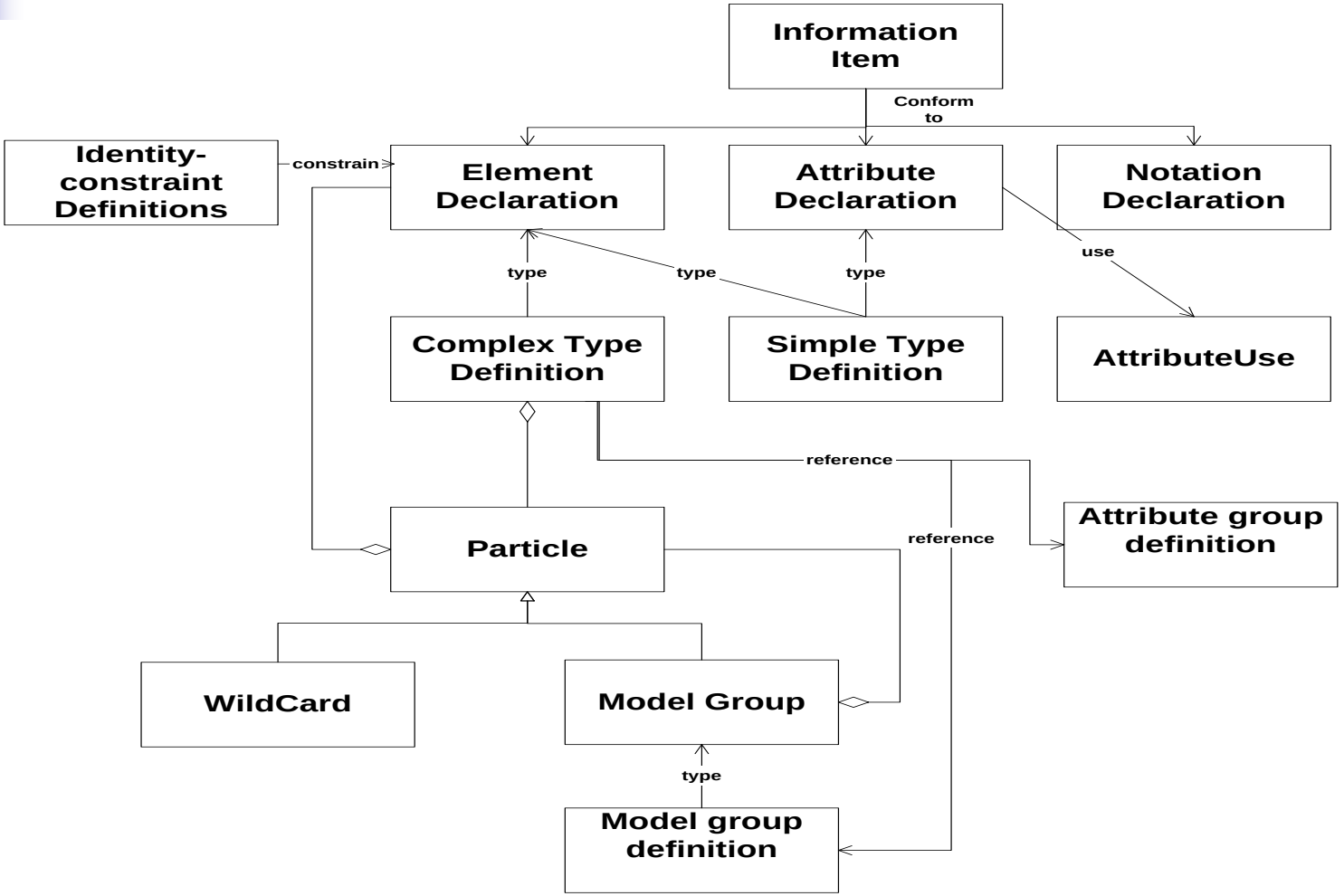


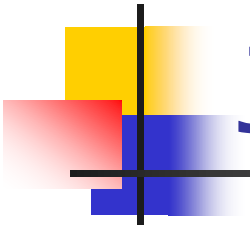
Annotations

- Annotations provide for human- and machine-targeted documentations of schema components.
- They are meta-data.

```
<xs:simpleType fn:note="special">
 <xs:annotation>
  <xs:documentation>A type for experts only</xs:documentation>
  <xs:appinfo>
   <fn:specialHandling>checkForPrimes</fn:specialHandling>
  </xs:appinfo>
 </xs:annotation>
```

Relationships among Schema Components





Javascript and the DOM

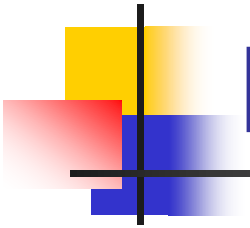
- Originally, the Document Object Model (DOM) and Javascript were tightly bound
- Each major browser line (IE and Netscape) had their own overlapping DOM implementation
- There's also some jargon issues, eg. DHTML…
- Now, the DOM is a separate standard, and can be manipulated by other languages (eg Java, server side javascript, python, etc)
- Browsers still differ in what parts of the standard they support, but things are much better now

Review: DOM Structure

- Objects are in a hierarchy
- The window is the parent for a given web page
- Document is the child with the objects that are most commonly manipulated

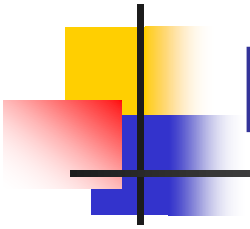
window
- * location
- * frames
- * history
- * navigator
- * event
- * screen
- * document
  - o links
  - o anchors
  - o images
  - o filters
  - o forms
  - o applets
  - o embeds
  - o plug-ins
  - o frames
  - o scripts
  - o all
  - o selection
  - o stylesheets
  - o body

table from: http://www.webmonkey.com/webmonkey/97/32/index1a.html?tw=author



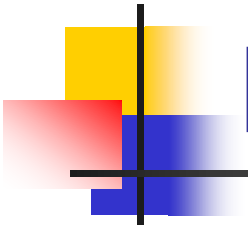
Referencing Objects

- Objects can be referenced
  - by their id or name (this is the easiest way, but you need to make sure a name is unique in the hierarchy)
  - by their numerical position in the hierarchy, by walking the array that contains them
  - by their relation to parent, child, or sibling (parentNode, previousSibling, nextSibling, firstChild, lastChild or the childNodes array



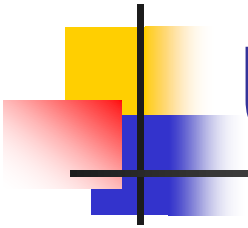
Learning The DOM

- The only way is to read and try things out
- Build a test document, with things you've learned
- Gecko_test.html is an example adapted from the mozilla site….



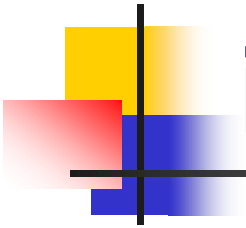
Manipulating Objects

- As said, it's easiest to reference objects by id
- To do this easily, use getElementById(), which returns the element with the given id
- For example, if you want to find a div with the id of "my_cool_div", use getElementById("my_cool_div")
- Keep in mind that it's the element itself that's returned, not any particular property



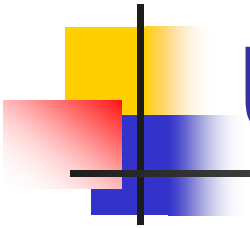
Using divs

- Div properties can be dynamically manipulated
- You can use this to make menus more dynamic



innerHTML

- innerHTML is a property of any document element that contains all of the html source and text within that element
- This is not a standard property, but widely supported--it's the old school way to manipulate web pages
- Much easier than building actual dom subtrees, so it's a good place to start
- Very important--innerHTML treats everything as a string, not as DOM objects (that's one reason it's not part of the DOM standard)



Using These….

- You can reference any named element with getElementById()
- You can read from or write to that element with innerHTML
- For example:
  getElementById("mydiv").innerHTML
  ="new text string";

A Simple DOM example

```
<div id="mydiv">
<p>
This some <i>simple</i> html to display
</p>
</div>
<form id="myform">

<input type="button" value="Alert innerHTML of mydiv"
  onclick="
      alert(getElementById('mydiv').innerHTML)
      " />
</form>
```

simple_dom_example.html

Manipulating Visibility

- You can manipulate the visibility of objects, this from http://en.wikipedia.org/wiki/DHTML
- The first part displays an element if it's hidden...

```
function changeDisplayState (id)
   {
   trigger=document.getElementById("showhide");
   target_element=document.getElementById(id);
   if (target_element.style.display == 'none'
      || target_element.style.display == "")
      {
      target_element.style.display = 'block';
      trigger.innerHTML = 'Hide example';
      }
...
```

31_dhtml_example.html