INFO/CS 4302
Web Information Systems

FT 2012

Week 5: Web Architecture: Structured Formats – Part 4 (DOM, JSON/YAML)

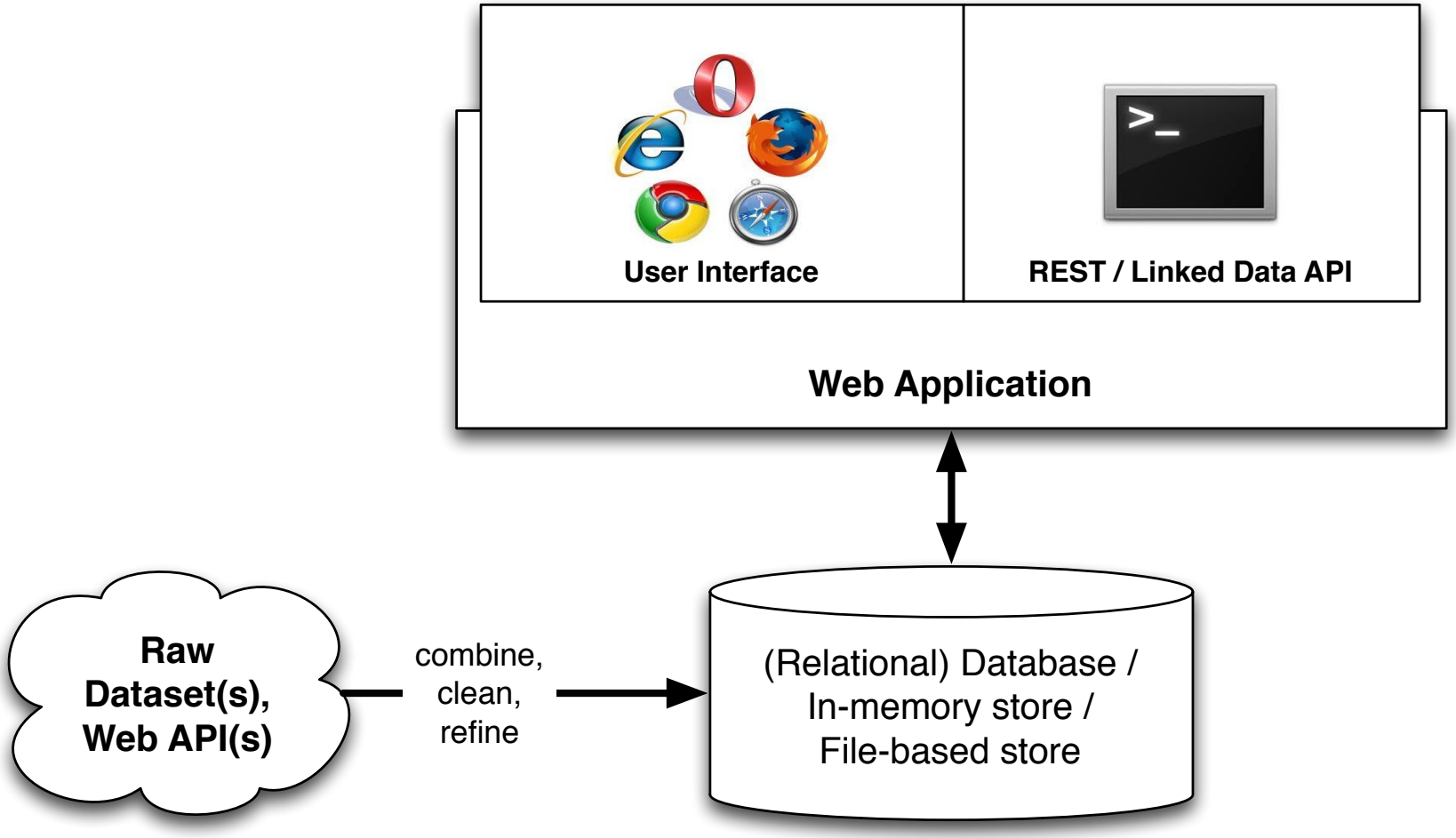
(Lecture 9)

Theresa Velden

Haslhofer & Velden

COURSE PROJECTS Q&A

Example Web Information System Architecture



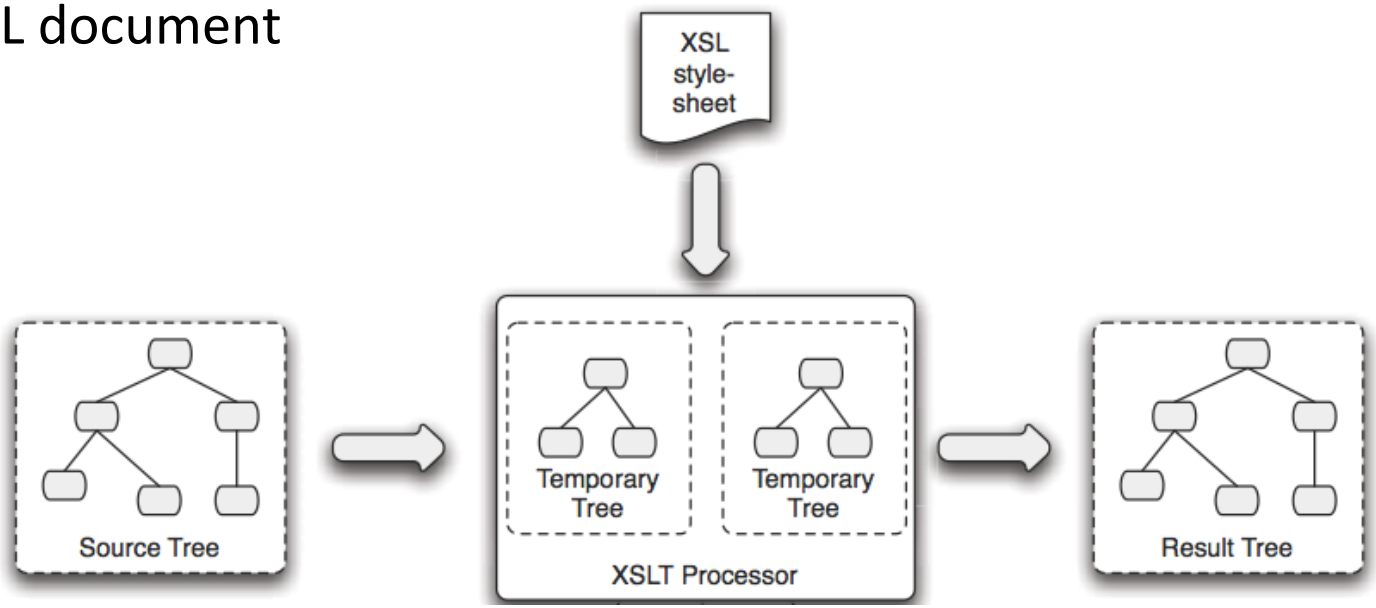
RECAP

XML & Related Technologies overview

Purpose	Structured content	Define Document Structure	Access Document Items	Transform Document
	XML	**XML Schema**	**XPath**	**XSLT**
	JSON	**RELAX NG**	**DOM**	
	YAML	DTD		

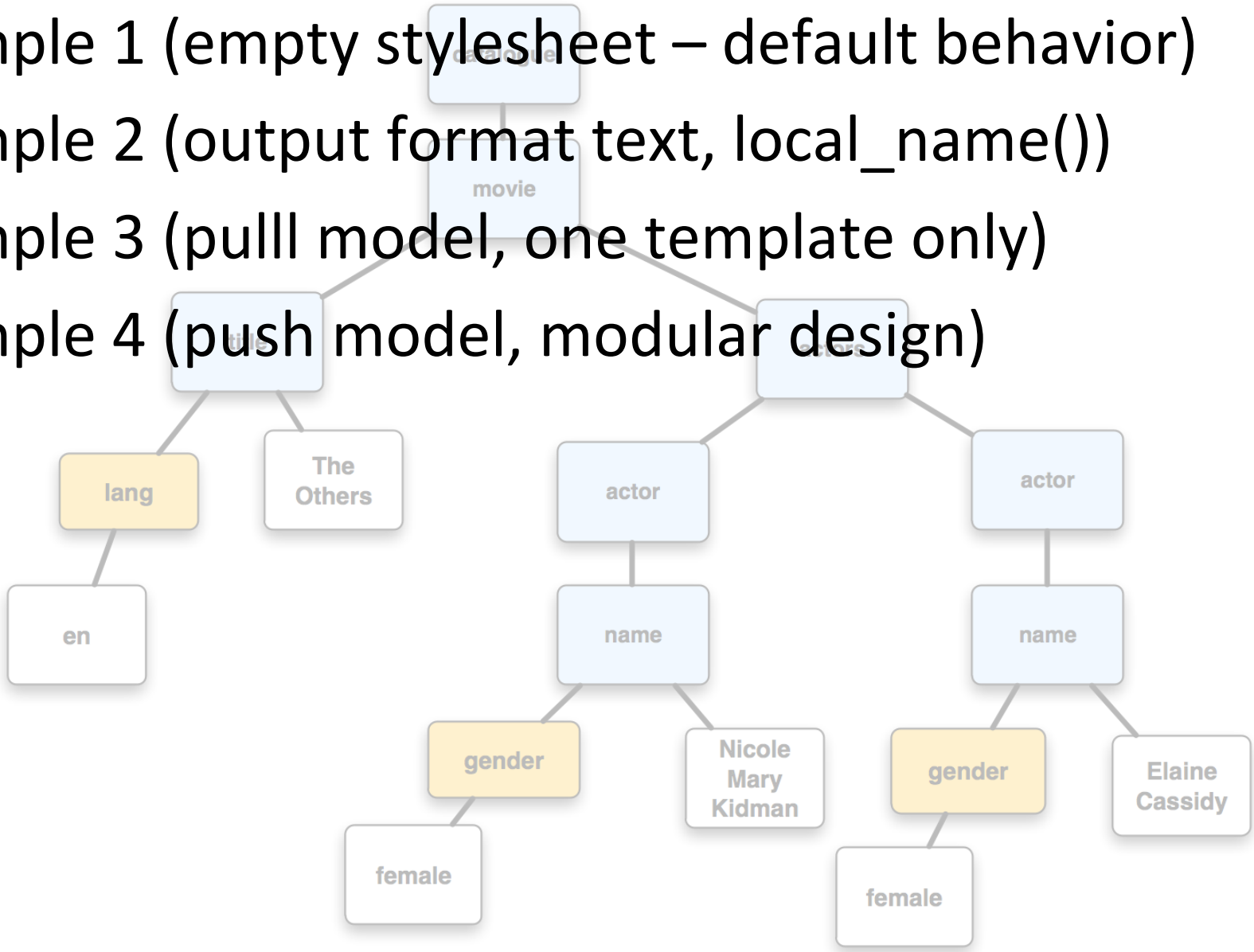
XSLT

- A transformation in the XSLT language is expressed in the form of an XSL stylesheet
  - root element: <xsl:stylesheet>
  - an xml document using the XSLT namespace, i.e. tags are in the namespace http://www.w3.org/1999/XSL/Transform
- The body is a set of templates or rules
  - The 'match' attribute specifies an XPath of elements in source tree
  - Body of template specifies contribution of source elements to result tree
- You need an XSLT processor to apply the style sheet to a source XML document



XSLT – In-class Exercise Recap

- Example 1 (empty stylesheet – default behavior)
- Example 2 (output format text, local_name())
- Example 3 (pulll model, one template only)
- Example 4 (push model, modular design)



XSLT: Conditional Instructions

- Programming languages typically provide ‘if-then, else’ constructs
- XSLT provides
  - If-then: **<xsl:if>**
  - If-then-(elif-then)*-else: **<xsl:choose>**

XML Source Document

```xml
<?xml version="1.0" encoding="UTF-8"?>
<catalogue>
    <movie>
        <title lang="en">The Others</title>
        <actors>
            <actor>
                <name gender="female">Nicole Mary Kidman</name>
            </actor>
            <actor>
                <name gender="female">Elaine Cassidy</name>
            </actor>
        </actors>
    </movie>
    <movie>
        <title lang="fr">Les Autres</title>
        <actors>
            <actor>
                <name gender="female">Nicole Mary Kidman</name>
            </actor>
            <actor>
                <name gender="female">Elaine Cassidy</name>
            </actor>
        </actors>
    </movie>
</catalogue>
```

xsl:if

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output encoding="UTF-8" indent="yes" method="html" />

    <xsl:template match="/">
        <html>
            <body>
                <h2>Movies</h2>
                <xsl:for-each select="catalogue/movie/title">
                        <xsl:if test="@lang='en'">
                            <h4><xsl:value-of select="."/> (English title)</h4>
                        </xsl:if>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<catalogue>
    <movie>
        <title lang="en">The Others</title>
        <actors>
            <actor>
                <name gender="female">Nicole Mary Kidman</name>
            </actor>
            <actor>
                <name gender="female">Elaine Cassidy</name>
            </actor>
        </actors>
    </movie>
    <movie>
        <title lang="fr">Les Autres</title>
        <actors>
            <actor>
                <name gender="female">Nicole Mary Kidman</name>
            </actor>
            <actor>
                <name gender="femal
            </actor>
        </actors>
    </movie>
</catalogue>
```

xsl:if

?

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output encoding="UTF-8" indent="yes" method="html" />

    <xsl:template match="/">
        <html>
            <body>
                <h2>Movies</h2>
                <xsl:for-each select="catalogue/movie/title">
                        <xsl:if test="@lang='en'">
                            <h4><xsl:value-of select="."/> (English title)</h4>
                        </xsl:if>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

xsl:if

```
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others (English title)</h4>
  </body>
</html>
```

HTML Result Document

xsl:choose

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output encoding="UTF-8" indent="yes" method="html" />

    <xsl:template match="/">
        <html>
            <body>
                <h2>Movies</h2>
                <xsl:for-each select="catalogue/movie/title">
                        <xsl:choose>
                            <xsl:when test="@lang='en'">
                                <h4><xsl:value-of select="."/> (English title)</h4>
                            </xsl:when>
                            <xsl:when test="@lang='fr'">
                                <h4><xsl:value-of select="."/> (French title)</h4>
                            </xsl:when>
                        </xsl:choose>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

xsl:choose

```
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others (English title)</h4>
    <h4>Les Autres (French title)</h4>
  </body>
</html>
```

HTML Result Document

xsl:choose

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output encoding="UTF-8" indent="yes" method="html" />

    <xsl:template match="/">
        <html>
            <body>
                <h2>Movies</h2>
                <xsl:for-each select="catalogue/movie/title">
                        <xsl:choose>
                            <xsl:when test="@lang='en'">
                                <h4><xsl:value-of select="."/> (<xsl:value-of select="@lang"/>.)</h4>
                            </xsl:when>
                            <xsl:when test="@lang='fr'">
                                <h4><xsl:value-of select="."/> (<xsl:value-of select="@lang"/>.)</h4>
                            </xsl:when>
                        </xsl:choose>
                </xsl:for-each>
            </body>
        </html>
    </xsl:template>

</xsl:stylesheet>
```

xsl:choose

```
<html>
  <body>
    <h2>Movies</h2>
    <h4>The Others (en.)</h4>
    <h4>Les Autres (fr.)</h4>
  </body>
</html>
```

HTML Result Document

DOM

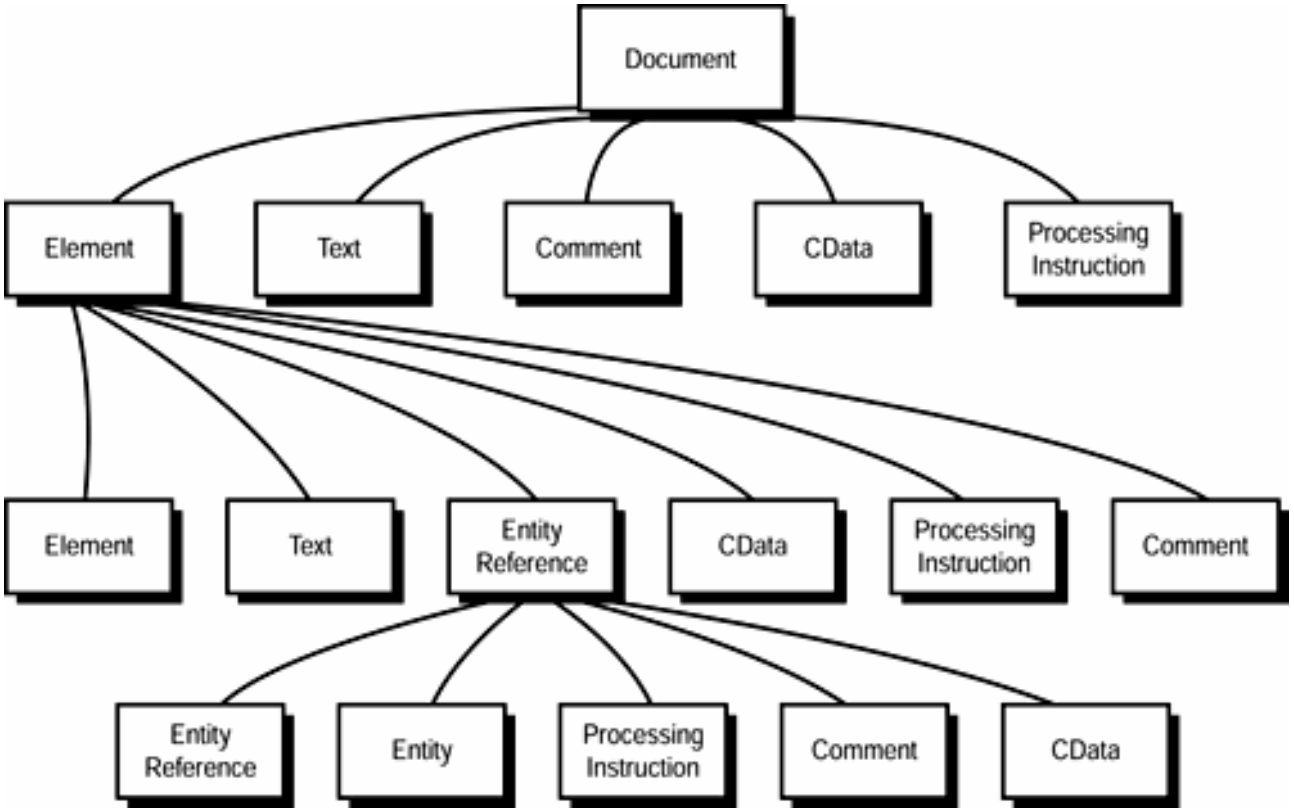
Well-formed XML Documents

- If an xml document is well-formed it can be transformed deterministically (parsed) to a single infoset (DOM)

DOM – Document Object Model

- API for manipulating document tree
  - Language independent; implemented in lots of languages
  - Suitable for XML and other document formats
  - Core notion of a **node**
  - Using DOM for a programming language requires definition of interfaces and classes ("language bindings" that provide a DOM conformant API)
- **W3C Specifications**
  - **Level 1:** functionality and navigation within a document
  - **Level 2**: modules and options for specific content models (XML, HTML, CSS)
  - **Level 3**: further content model facilities (e.g. XML validation handlers)

DOM (Core) Level 1



DOM Node Objects

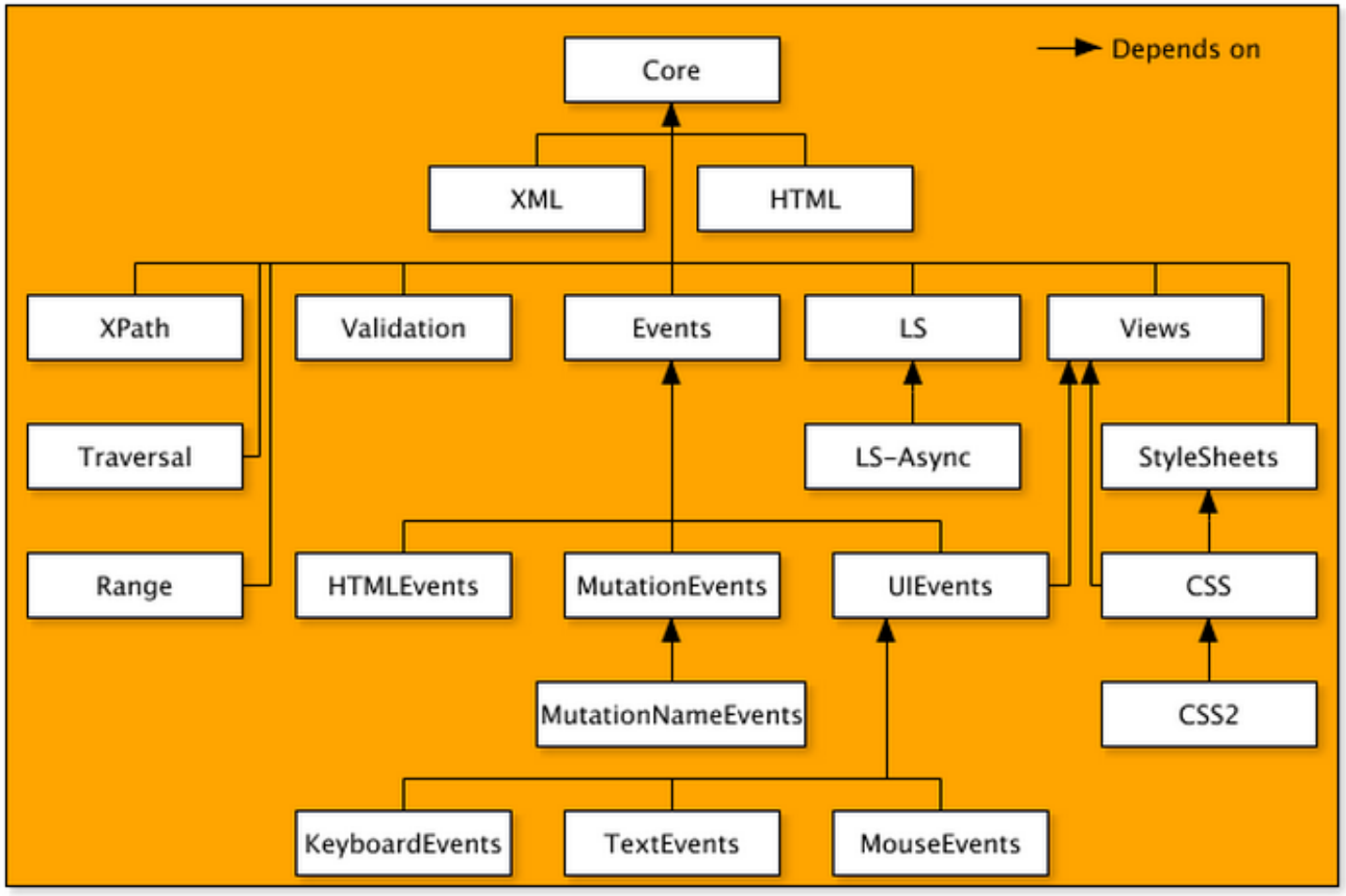
- Root must be Document
- Root children: one node and optionally Processing Instructions, DocumentType, Comment, Text
- Children of element must be either element, comment or text
- Attributes are properties of elements (not their children), and are not member of the DOM tree
- Children of attribute must be text

W3C DOM Node Types

Node type	Description	Children
Document	Represents the entire document (the root-node of the DOM tree)	Element (max. one), ProcessingInstruction, Comment, DocumentType
DocumentFragment	Represents a "lightweight" Document object, which can hold a portion of a document	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
DocumentType	Provides an interface to the entities defined for the document	None
ProcessingInstruction	Represents a processing instruction	None
EntityReference	Represents an entity reference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Element	Represents an element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	Represents an attribute	Text, EntityReference
Text	Represents textual content in an element or attribute	None
CDATASection	Represents a CDATA section in a	None

		CDATASection, EntityReference
DocumentType	Provides an interface to the entities defined for the document	None
ProcessingInstruction	Represents a processing instruction	None
EntityReference	Represents an entity reference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Element	Represents an element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
Attr	Represents an attribute	Text, EntityReference
Text	Represents textual content in an element or attribute	None
CDATASection	Represents a CDATA section in a document (text that will NOT be parsed by a parser)	None
Comment	Represents a comment	None
Entity	Represents an entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
Notation	Represents a notation declared in the DTD	None

View of DOM Architecture



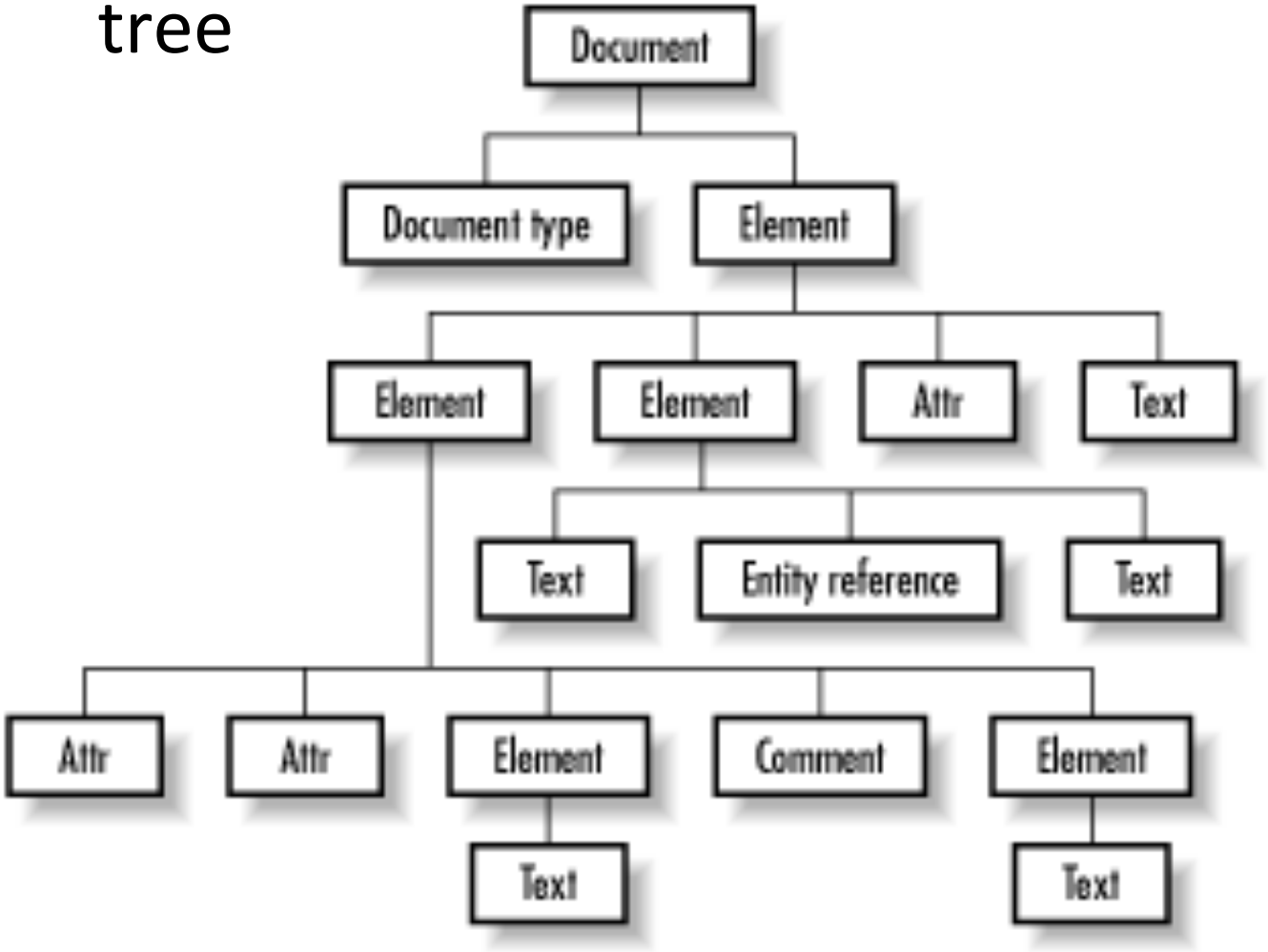
http://www.w3.org/TR/DOM-Level-3-Core/introduction.html

DOM Architecture and Conformance

- The Core module, which contains the fundamental interfaces that must be implemented by all DOM conformant implementations, is associated with the feature name "Core";
- The XML module, which contains the interfaces that must be implemented by all conformant XML 1.0 [XML 1.0] (and higher) DOM implementations, is associated with the feature name "XML".

Parsing XML Document Model (DOM)

- Complete in-memory representation of entire tree



http://docstore.mik.ua/orelly/xml/jxml/ch05_01.htm

Java, Python & DOM

- JDOM
  - Basic navigation functionality
    - Parent
    - Child (all, specific, filtered)
    - Descendants
    - Attributes (all, specific, filtered)
  - Basic tree manipulation
    - Adding, replacing, removing contents and attributes)
    - Text modification
    - Maintains well-formedness
- Python: http://docs.python.org/library/xml.dom.minidom.html

Alternative Support for Parsing of XML

- SAX (open source software)
  - Event driven, sequential
  - Mostly hierarchical, no' sibling' concept
  - More memory efficient
  - Good for quick, less intensive parsing that does not require easy-to-use, clean interface

http://docstore.mik.ua/orelly/xml/jxml/ch05_01.htm

Simple SAX Example

Document

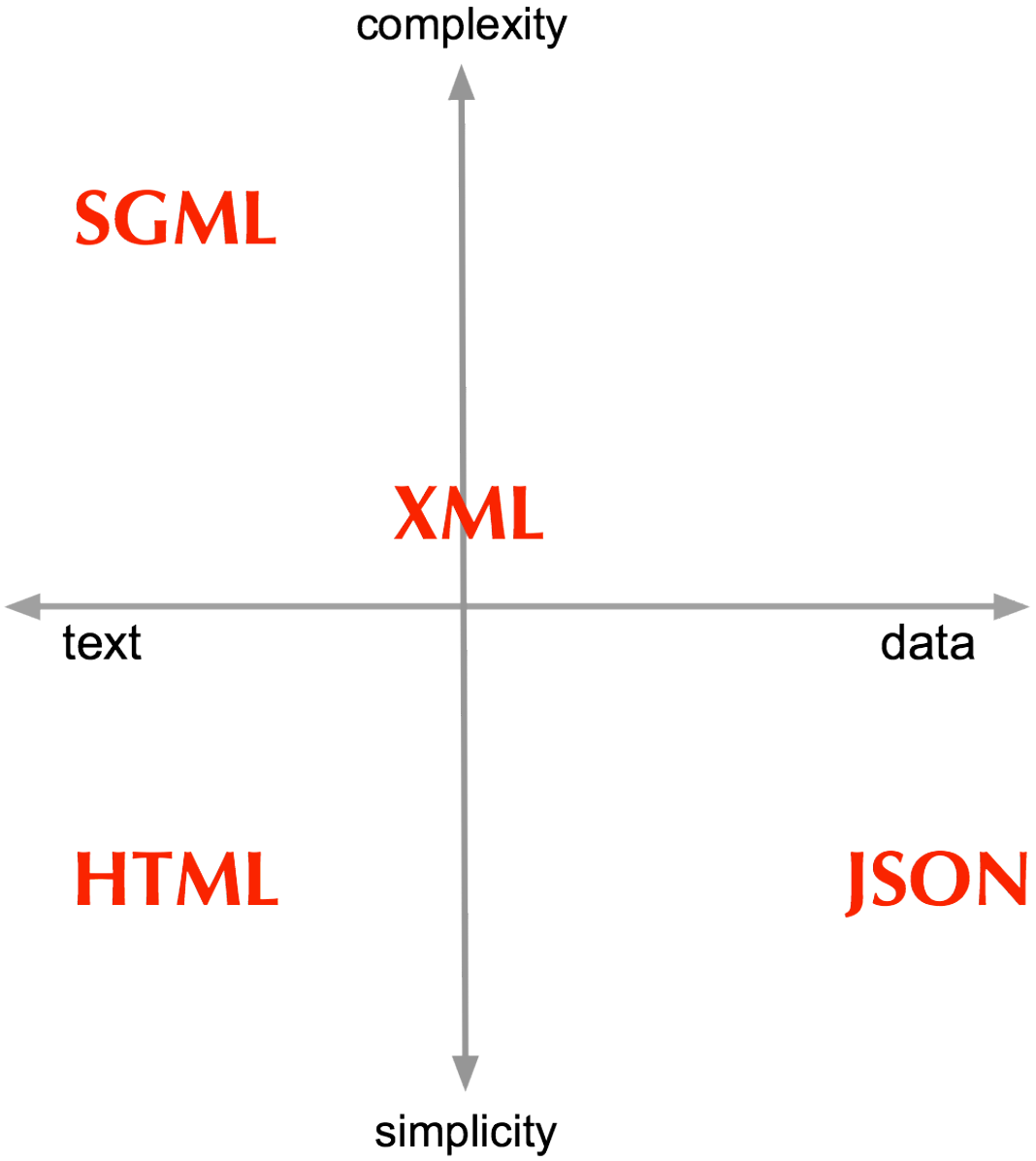
```
<?xml version="1.0" encoding="UTF-8"?>

<books>
    <book>War and Peace</book>
</book>
```

Events

startDocument()
startElement("books")
startElement("book")
characters("War and Peace")
endElement("book")
endElement("books")
endDocument()interface

JSON/YAML

Document/Data Formats



JSON

- *"JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. […] JSON defines a small set of formatting rules for the portable representation of structured data."*

[RFC 4627 http://tools.ietf.org/html/rfc4627]

- popularization and formal specification defined by Douglas Crockford (senior JavaScript architect at PayPal)
- JSON Homepage: http://www.json.org/

JSON - an open standard for ***data interchange***

- text-based & human readable
- machine readable and easy to parse
- not a mark-up language but 'self-descriptive'
- not extensible (different from XML)

```
{
      "firstName": "Nicole",
      "lastName": "Kidman",
      "phoneNumber": [
        {
          "type": "home",
          "number": "212 555-1234"
        },
        {
          "type": "fax",
          "number": "646-555-4567"
        }
      ]}
```

JSON = JavaScript Object Notation

- programming language independent
- maps to two universal structures:

1) An unordered collection of name value pairs:

```
{"firstName": "Nicole","lastName": "Kidman"}
```

in other languages: ***object***, record, struct, dictionary, hash table, keyed list, or associative array

2) An ordered list of values:

```
["Monday", "Tuesday", "Wednesday"]
```

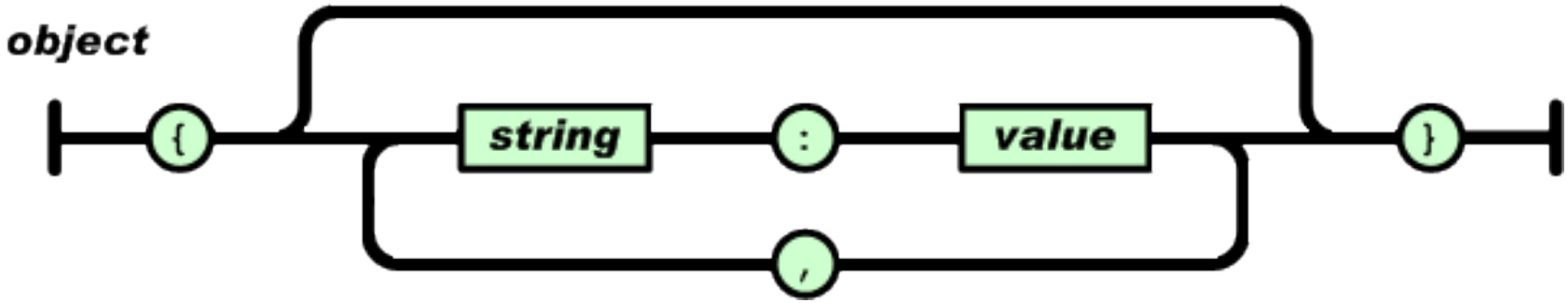
in other languages: ***array***, vector, list, or sequence

JSON Syntax

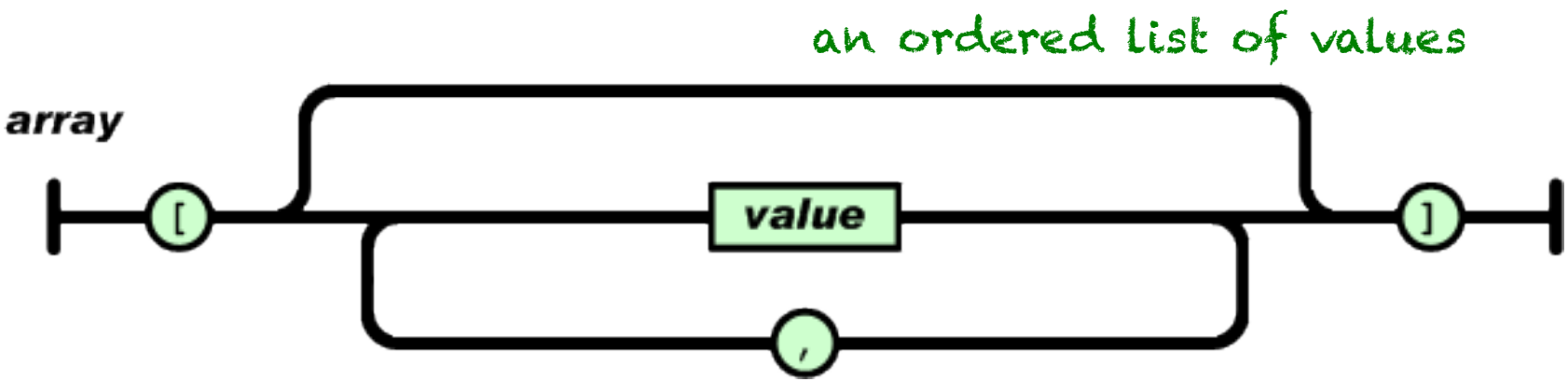
- Primitive types:
  1. Strings
  2. Numbers
  3. Booleans (true/false)
  4. Null
- Structured Types
  1. Objects
  2. Array

JSON Syntax

a collection of string value pairs

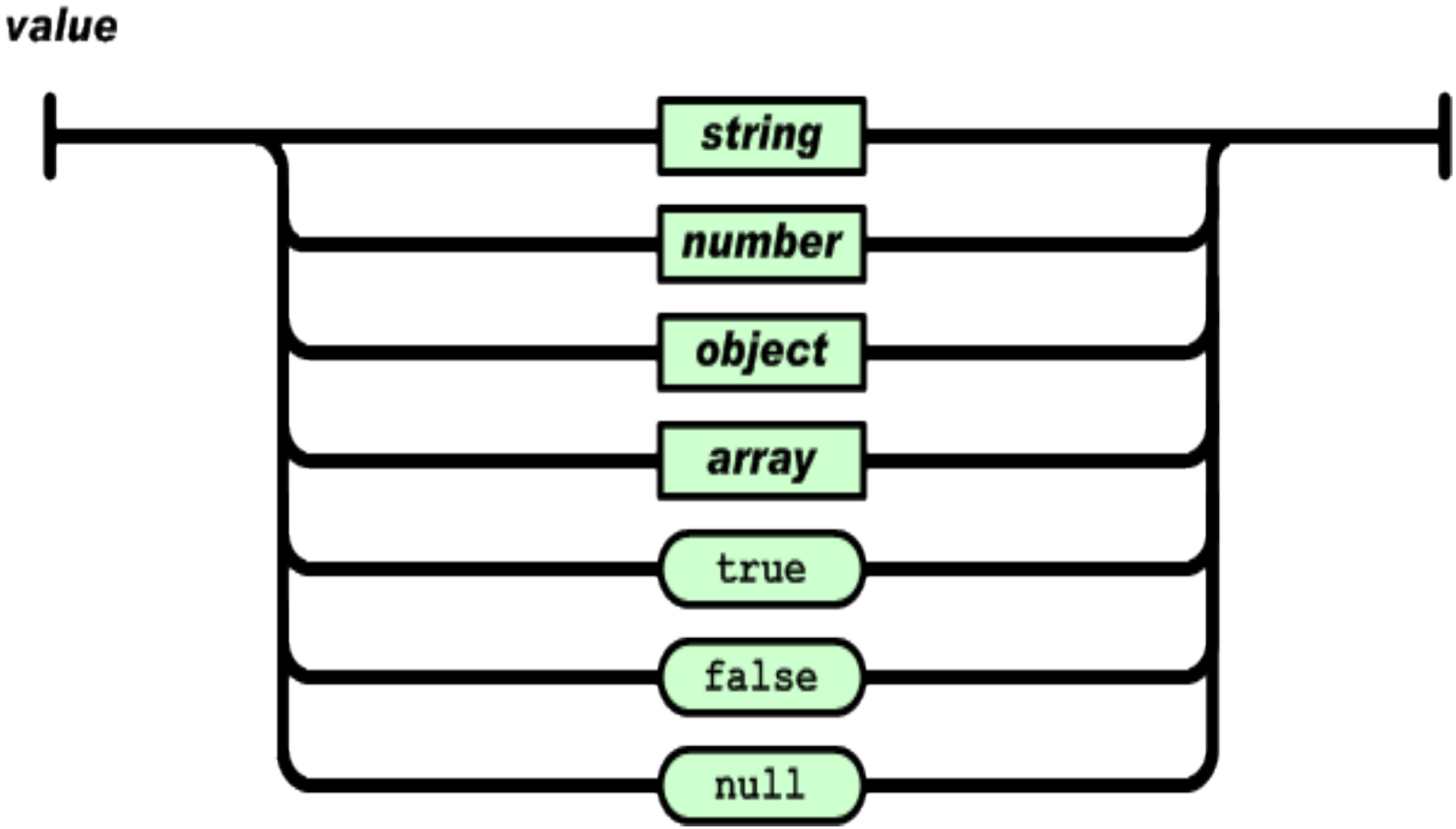


an ordered list of values



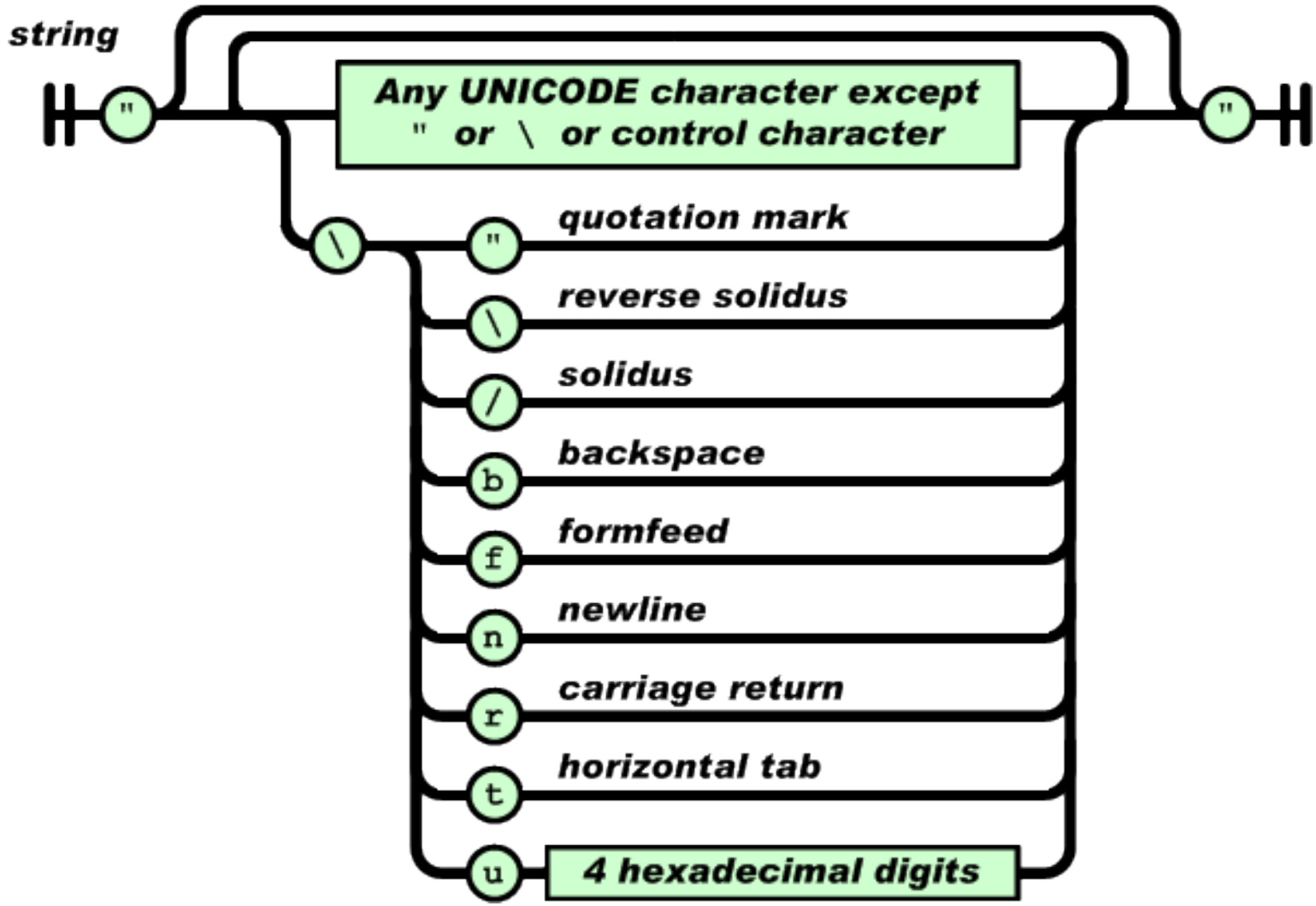
http://www.json.org/

JSON Syntax



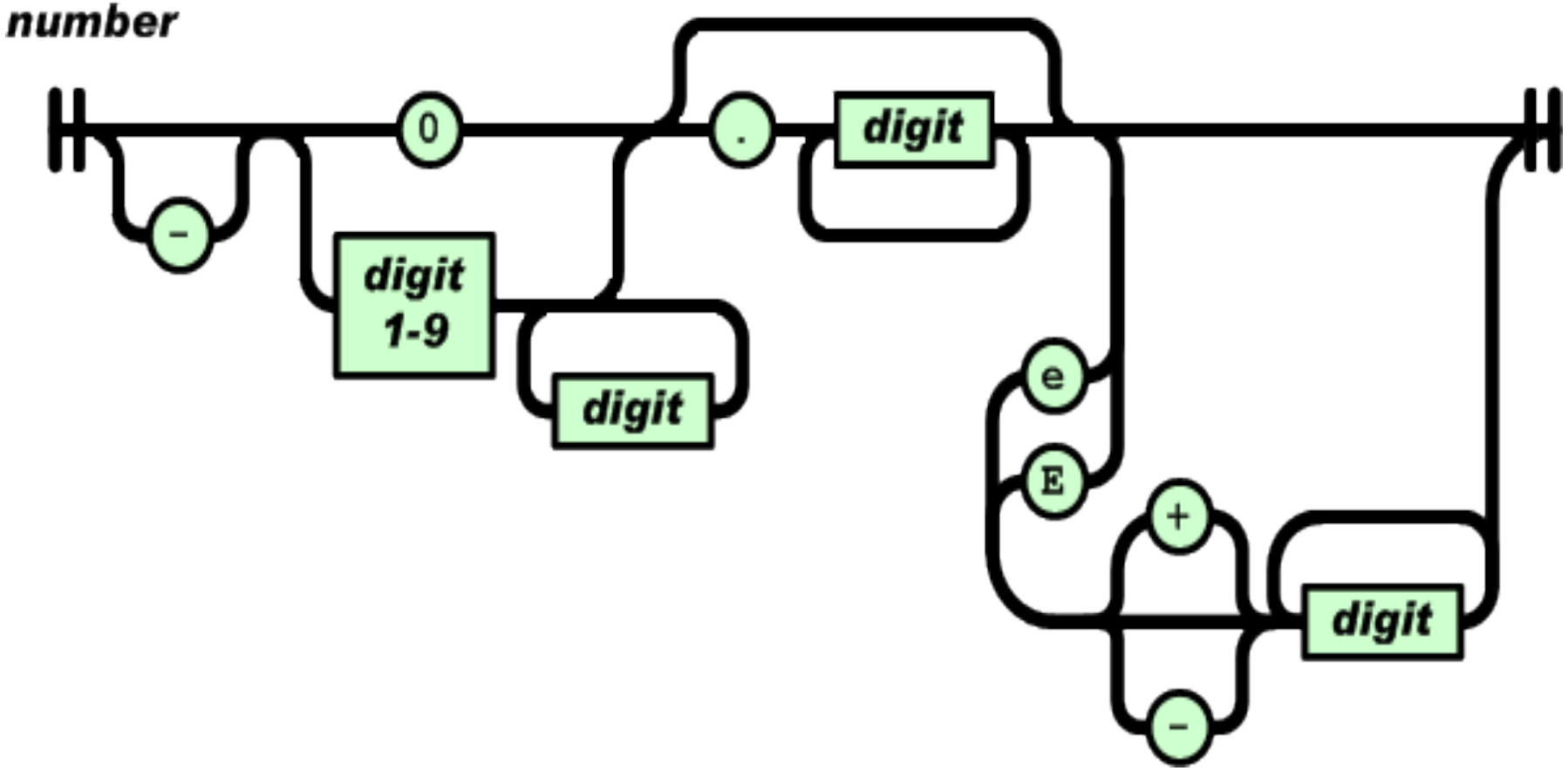
http://www.json.org/

JSON Syntax



http://www.json.org/

JSON Syntax



http://www.json.org/

JSON Example: De-constructed

```
{
      "firstName": "Julia",   "lastName":
   "Chen",   "phoneNumber": [
        {
           "type": "home",
           "number": "212 555-1234"
        },
         {
           "type": "fax",
           "number": "646-555-4567"
        }
     ]}
```

JSON Example: De-constructed

```
{
    "firstName": "Julia",   "lastName":
  "Chen",   "phoneNumber": [
      {
        "type": "home",
        "number": "212 555-1234"
      },
       {
        "type": "fax",
        "number": "646-555-4567"
      }
    ]}
```

an object with three string value pairs

JSON Example: De-constructed

```
{
    "firstName": "Julia",  "lastName":
  "Chen",  "phoneNumber": [     {
        "type": "home",
        "number": "212 555-1234"
      },
       {
        "type": "fax",
        "number": "646-555-4567"
      } ]}
```

an object with three string value pairs

the value of the third pair is an array

JSON Example: De-constructed

```
{
    "firstName": "Julia",  "lastName":
  "Chen",  "phoneNumber":
  [    {        "type": "home",
       "number": "212 555-1234"      },
      {
       "type": "fax",
       "number": "646-555-4567"      }
    ]}
```

an object with three string value pairs

the value of the third pair is an array

holding two objects, each with two string value pairs

JSON formatting and syntax rules

http://www.freeformatter.com/json-formatter.html

Syntax rules such as:

- "Objects are encapsulated within opening and closing brackets { }
- An empty object can be represented by { }
- Arrays are encapsulated within opening and closing square brackets [ ]
- An empty array can be represented by [ ]
- A member is represented by a key-value pair
- The key of a member should be contained in double quotes. (JavaScript does not require this. JavaScript and some parsers will tolerate single-quotes) [...]"

JSON to XML comparison

From: "JSON: The fat free alternative to XML" http://www.json.org/xml.html

"JSON is a better data exchange format. XML is a better document exchange format. Use the right tool for the right job."

- JSON mime type: **application/json**
- formally defined in **RFC 4627**

JSON Schema

- defines structure of JSON data
- a contract what JSON data to provide for an application and how to interact with it
- mime-type: application/schema+json
- **Expired** IETF draft:

http://tools.ietf.org/html/draft-zyp-json-schema-03

- rarely used and lack of tool support

JSON data & JSON Schema

```
{
        "catalogue": [
          {
            "movie": {
              "ID": 24,
              "title": "Star Wars Episode VI:",
              "actors": [
                {
                  "ID": 10,
                  "name": "Mark Hamill",
                  "birth_date": "09-25-1951"
                },
                {
                  "ID": 12,
                  "name": "Anthony Daniels",
                  "birth_date": "02-21-1946"
                }
              ]}}]}
```

schema excerpt:

e.g. an object has a name (the string) and properties defining the value part of the object that can include further objects…

```
{"name": "movie",
"properties":{
      "ID":{
        "type": "number",
        "description": "movie identifier",
        "required": true
        },
                …
        “actors”:{
        “type”: “object”, …
      }
}
```

JSON & AJAX

Using the XMLHttpRequest API in Javascript

returns data in JSON format from server and evaluates it on the client side as a JavaScript object

```
var my_JSON_object = {};
var http_request = new XMLHttpRequest();
http_request.open("GET", url, true);
http_request.onreadystatechange = function () {
        var done = 4, ok = 200;
        if (http_request.readyState == done &&
        http_request.status == ok) {
         my_JSON_object = JSON.pars(http_request.responseText);
          }};
      http_request.send(null);
```

[source: http://en.wikipedia.org/wiki/JSON#Schema]

Same origin policy: the URL replying to the request must reside within the same DNS domain as the server that hosts the page containing the request; need to use **JSONP** (JSON with padding) to request objects from different servers, see http://en.wikipedia.org/wiki/JSONP

Writing JSON in Python

```
def generateJson(allMovies,allActors,table):
    container={}
    catalogue={}
    movies=[]
    container['catalogue']=catalogue
    catalogue['movies']=movies
    for movieID, movieTitle in allMovies.iteritems():
        movie={}
        movie['movie']=[]
        title={}
        title['title']=movieTitle
        movie['movie'].append(title)
        movieActors={}
        movieActors['actors']=[]
        for personID in table[movieID]:
            actor={}
            actorInfo={}
            actor['actor']=actorInfo
            name=allActors[personID]
            actorInfo['name']=name
            movieActors['actors'].append(actor)
        movie['movie'].append(movieActors)
        movies.append(movie)
    JSONfile=json.dumps(container)
    return JSONfile
```

JSON result file:

{"catalogue": {"movies": [{"movie": [{"title": "The Others"}, {"actors": [{"actor": {"name": "Nicole Mary Kidman"}}, {"actor": {"name": "Elaine Cassidy"}}, {"actor": {"name": "Christopher Eccleston"}}, {"actor": {"name": "Alakina Mann"}}, {"actor": {"name": "Eric Sykes"}}, {"actor": {"name": "Fionnula Flanagan"}}]}]}, ...]}}

Writing JSON in Python

Good news: Starting with python 2.6 the JSON module is part of the standard python library

Parsing JSON versus XML

"Speeding up AJAX with JSON" by Jean Kelly (2006) at http://www.developer.com/lang/jscript/article.php/3596836

YAML = “YAML Ain’t Markup Language”

- is a human-friendly data serialization language
- design goals in decreasing priority according to specification at http://www.yaml.org/spec/1.2/spec.html:
  - YAML is easily readable by humans.
  - YAML data is portable between programming languages.
  - YAML matches the native data structures of agile languages.
  - YAML has a consistent model to support generic tools.
  - YAML supports one-pass processing.
  - YAML is expressive and extensible.
  - YAML is easy to implement and use.
- Relation to JSON
  - every JSON file is a valid YAML file
  - different design goals: YAML (human-friendly), JSON (simplicity, universality)
  - YAML more complex to generate and parse
  - YAML has a more complete information model
- Relation to XML: “YAML is the result of lessons learned from XML and other technologies”

YAML

Example 2.1. Sequence of Scalars (ball players)

```
- Mark McGwire
- Sammy Sosa
- Ken Griffey
```

Example 2.2. Mapping Scalars to Scalars (player statistics)

```
hr:  65    # Home runs
avg: 0.278 # Batting average
rbi: 147   # Runs Batted In
```

Example 2.3. Mapping Scalars to Sequences (ball clubs in each league)

```
american:
  - Boston Red Sox
  - Detroit Tigers
  - New York Yankees
national:
  - New York Mets
  - Chicago Cubs
  - Atlanta Braves
```

Example 2.4. Sequence of Mappings (players' statistics)

```
-
  name: Mark McGwire
  hr:   65
  avg:  0.278
-
  name: Sammy Sosa
  hr:   63
  avg:  0.288
```

Example 2.5. Sequence of Sequences

```
- [name        , hr, avg  ]
- [Mark McGwire, 65, 0.278]
- [Sammy Sosa  , 63, 0.288]
```

Example 2.6. Mapping of Mappings

```
Mark McGwire: {hr: 65, avg: 0.278}
Sammy Sosa: {
    hr: 63,
    avg: 0.288
  }
```

Source: http://www.yaml.org/spec/1.2/spec.html

YAML

Example 2.27. Invoice

```
--- !<tag:clarkevans.com,2002:invoice>
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    given  : Chris
    family : Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city    : Royal Oak
        state   : MI
        postal  : 48046
ship-to: *id001
product:
    - sku         : BL394D
      quantity    : 4
      description : Basketball
      price       : 450.00
    - sku         : BL4438H
      quantity    : 1
      description : Super Hoop
      price       : 2392.00
tax  : 251.42
total: 4443.52
comments:
    Late afternoon is best.
    Backup contact is Nancy
    Billsmer @ 338-4338.
```

Source: http://www.yaml.org/spec/1.2/spec.html

Next Week:

- **Tuesday:** recap of first four weeks of classes
  - Topics
    - Internet Architecture
    - Web Architecture
    - Structured Data Formats (XML & related Technologies)
  - ***Use piazza to suggest topics to recap*** **(**areas of fuzziness/lingering doubt)
- **Thursday:** 'Internet Surveillance'
  - Note: readings will be announced later today and homework 4 will be responses to these readings
  - We want you to come prepared, **so homework 4** will be due by 10 AM on Thursday 9/27 (i.e. before class!)