Web Data Models

Silviu Maniu

Introduction: XML, JSON, parsing, validation

Data Exchange Problem

- we are in an age of **Data**
- types of data: textual, medical, legal, GPS, …
- computers need to communicate — data need to be readable by anyone
- how to keep long-term access to data?

Web Data Models

- **Objective of the course**: study data models that are interchangeable and are extensively used on the Web

- We will concentrate on XML, JSON and algorithms for evaluating / processing data

- **Not studying**: graph models, systems and data architectures

Web Data Models

- **When/Where**: Mondays, 9h00–12h15, PUIO (PSud)
- **Who**: Silviu Maniu, U. Paris-Sud silviu.maniu@lri.fr
- **Site**: http://silviu.maniu.info/teaching/
- **Evaluation**: first session 50% Project + 50% Exam; project deadline the week after the exam; second session (if grade <10): 100% Exam

Web Data Models

- Web Data Models: XML, JSON
- XML Validation: DTD, XML Schema, RelaxNG
- Processing XML: XPath, XQuery, XSLT

Data Exchange

- **Why?** To ensure programs read and use data **from other programs**

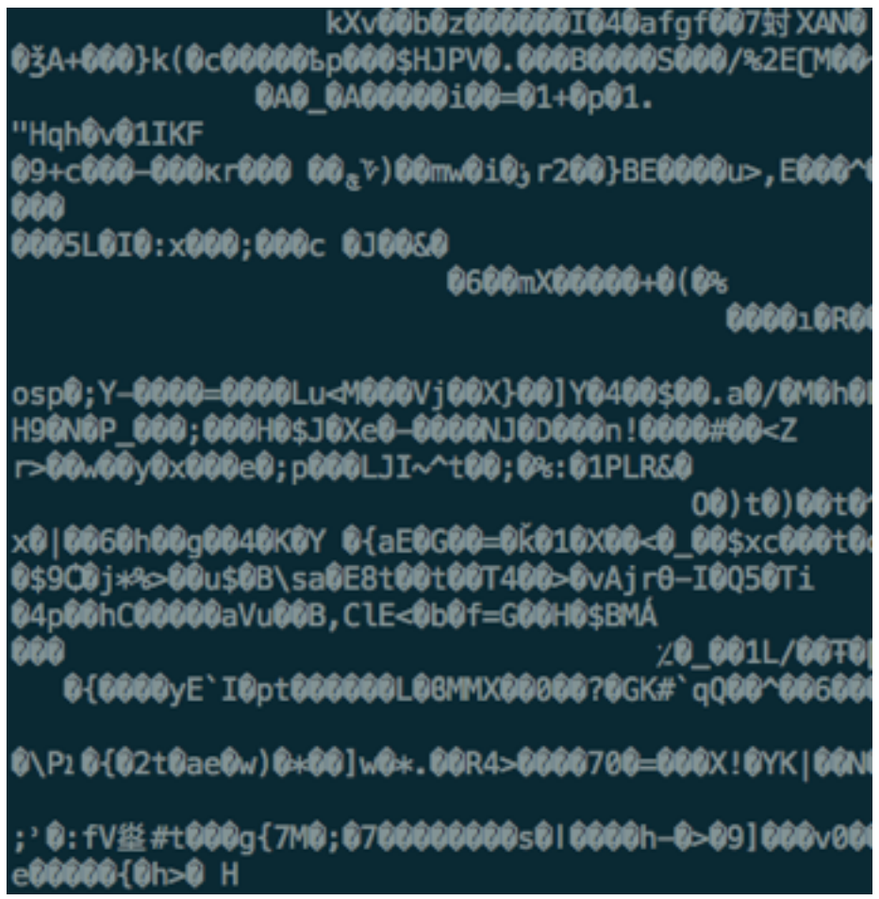
Data Exchange

- **How can we encode data?**

Data Exchange: Unstructured Data (Text/Binary)

```
Theory of Computation
Michael Sipser
Cengage Learning
2012
3

Artificial Intelligence
Stuart Russell
Peter Norvig
Pearson
2013
3
```



- What is the format? What are the fields?
- Need to send the file and the parser.

Data Exchange: Relational Database

Books

id	title	editor	year
1	Theory of Computation	Cengage	2012
2	Artificial Intelligence	Pearson	2013

Authors

id	first	last
1	Sipser	Michael
2	Russell	Stuart
3	Norvig	Peter

BookAuthor

author	book
1	1
2	2
3	2

- Fixed schema, strict typing

Data Exchange

- **How can we encode data for easy interchange?**

Two desiderata:

1. easy to read / parse (exchange)
2. easy to detect the structure (parse)

Semi-Structured Data (SSD)

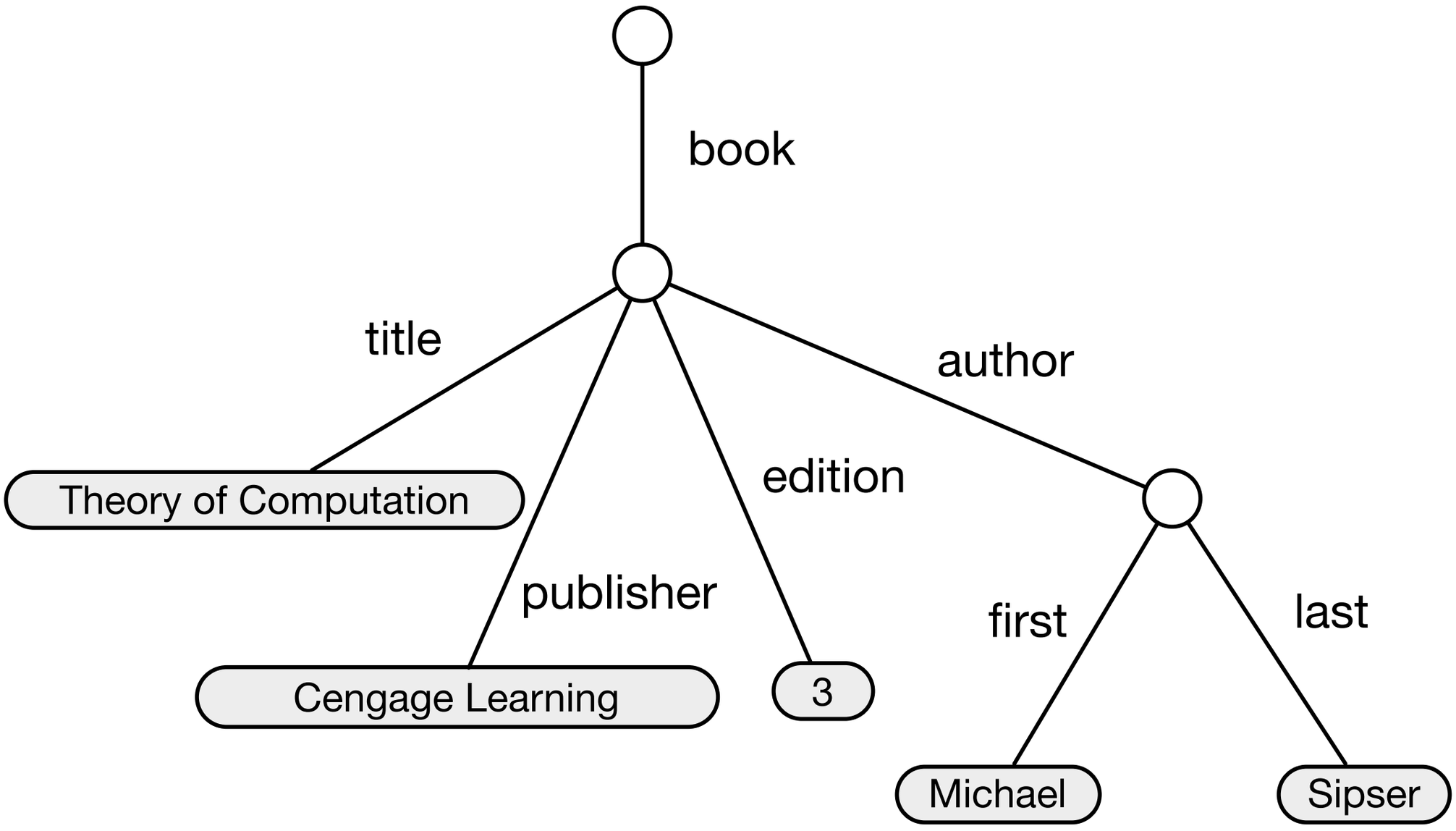
- an attempt to reconcile document view (text, HTML) and the strict structures in DB
- organized in semantic entities: similar entities grouped together, entities in the same group may not have same fields
- defined as (possibily nested) set of field-value pairs
- order of fields not important!

Semi-Structured Data (SSD)

```
{books:{
  title: “Theory of Computation”,
  edition: 3
  publisher: “Cengage Learning”,
  author: {first: “Michael”, last: “Sipser”}
},
…}
```

Semi-Structured Data (SSD)

- We need an internal representation / data model — trees



Semi-Structured Data (SSD)

- How to store / represent?

- Different formalisms to represent: Object Exchange Model, Lore, XML, JSON

XML

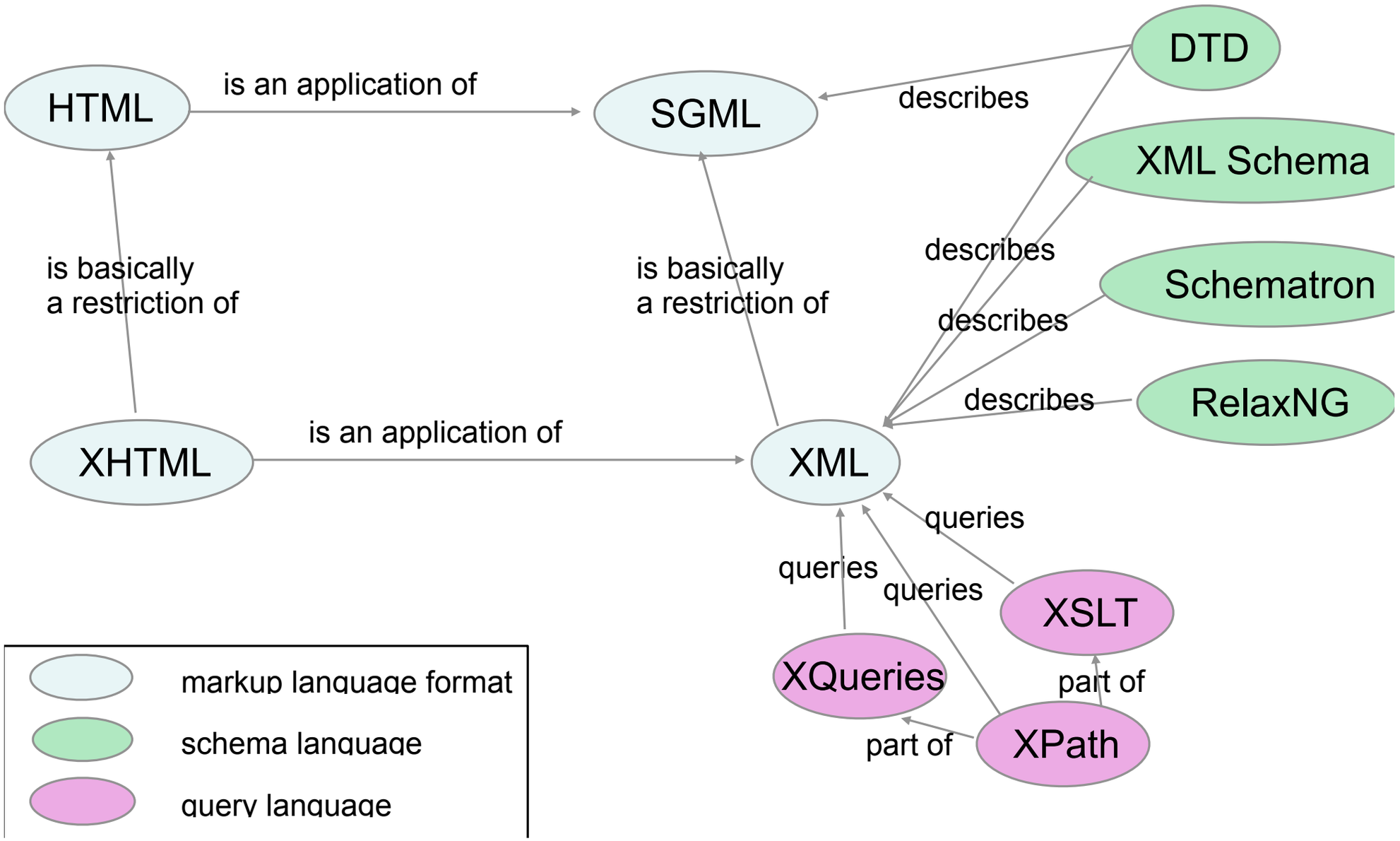
XML: e**X**tensible **M**arkup **L**anguage

- representation of semi-structured data, suitable for humans and computers
- is not for the layout of documents (HTML, CSS, …)
- querying still not perfect

XML: Brief History

- **GML** (Generalized Markup Language), ~1960, IBM
- **SGML** (Standard Generalized Markup Language), 1985 — flexible, expressive
- **HTML** (Hypertext Markup Language), ~1990 — application of SGML for presentation of Web Documents
- **XML**, 1998 (first edition) — a W3C standard, subset of SGML

XML: The Landscape



Credit: B. Parsia, C. Hedeler, U. Sattler; Univ. of Manchester

XML: Example

```
Theory of Computation
Michael Sipser
Cengage Learning
2012
3

Artificial Intelligence
Stuart Russell
Peter Norvig
Pearson
2013
3
```

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

XML Example: Wikipedia

University of Paris-Saclay

From Wikipedia, the free encyclopedia
(Redirected from Université Paris-Saclay)

Coordinates: 48.7117343°N 2.1712888°E

The **University of Paris-Saclay** (French: *Université Paris-Saclay*) is a French federal research university which is currently under development. The project is part of the research-intensive and business cluster Paris-Saclay, located near Paris in the Plateau de Saclay.[5][6] The University of Paris-Saclay is expected to be the training and research center of the Paris-Saclay technology cluster, similar to Stanford University in the Silicon Valley and the Technion – Israel Institute of Technology in Israel.

The University's first academic year started in September 2015.[7]

University of Paris-Saclay

Université Paris-Saclay

Contents [hide]

1 Mission
2 Ranking
3 Setup
4 Academic programmes
5 Research programmes
6 See also
7 References
8 External links

Mission [edit]

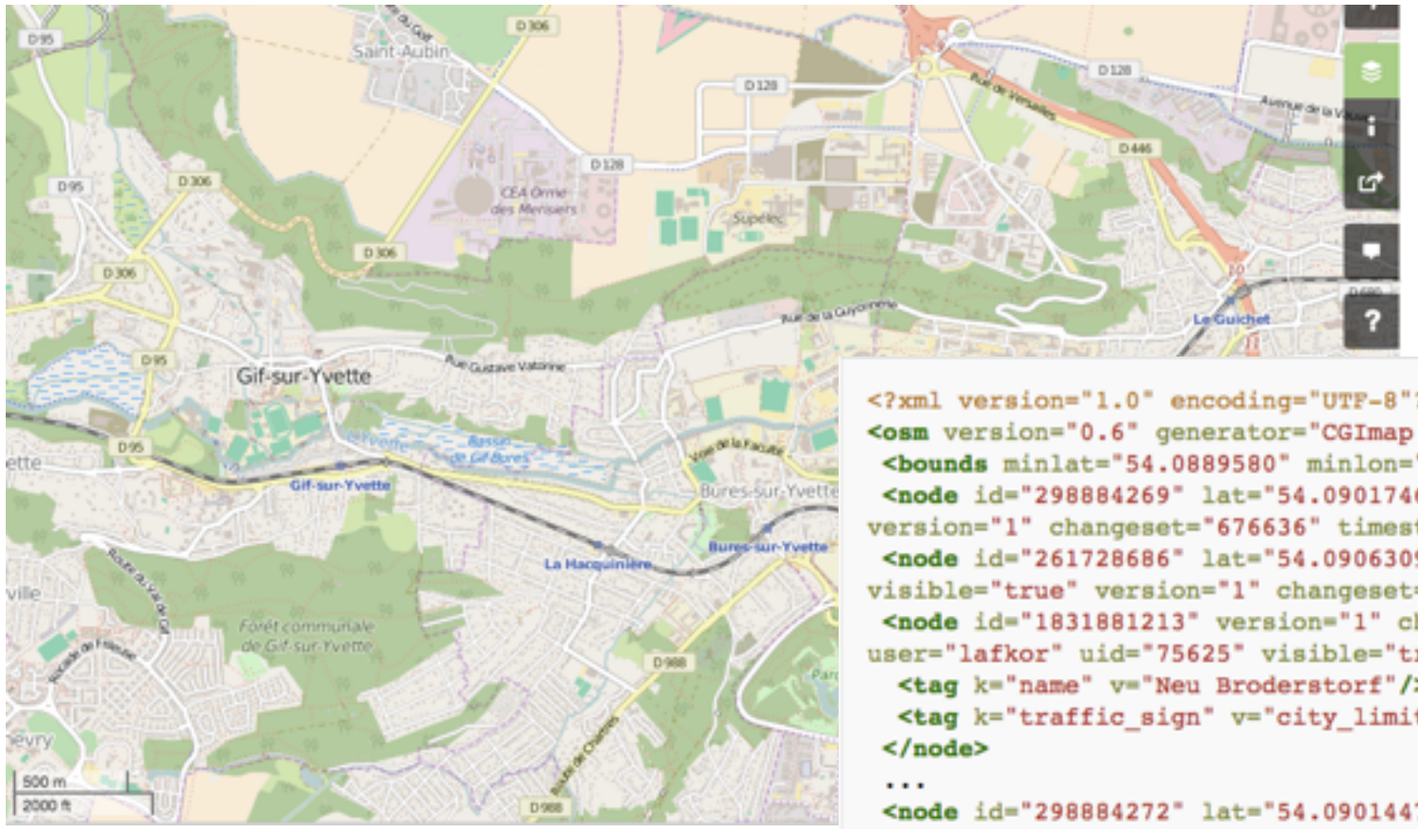
According to Dominique Vernay, chairman of the foundation developing Paris-Sacl
in the Academic Ranking of World Universities compiled by Shanghai Jiao Tong U
top university in continental Europe".[8] The university wants to maximise the econ
research, partly via university and research spin-offs, but also by working closely w

Ranking [edit]

University Paris-Saclay is still under its integrative process, therefore not yet appe
rankings. However, an independent simulation ranked the university in the 18th po

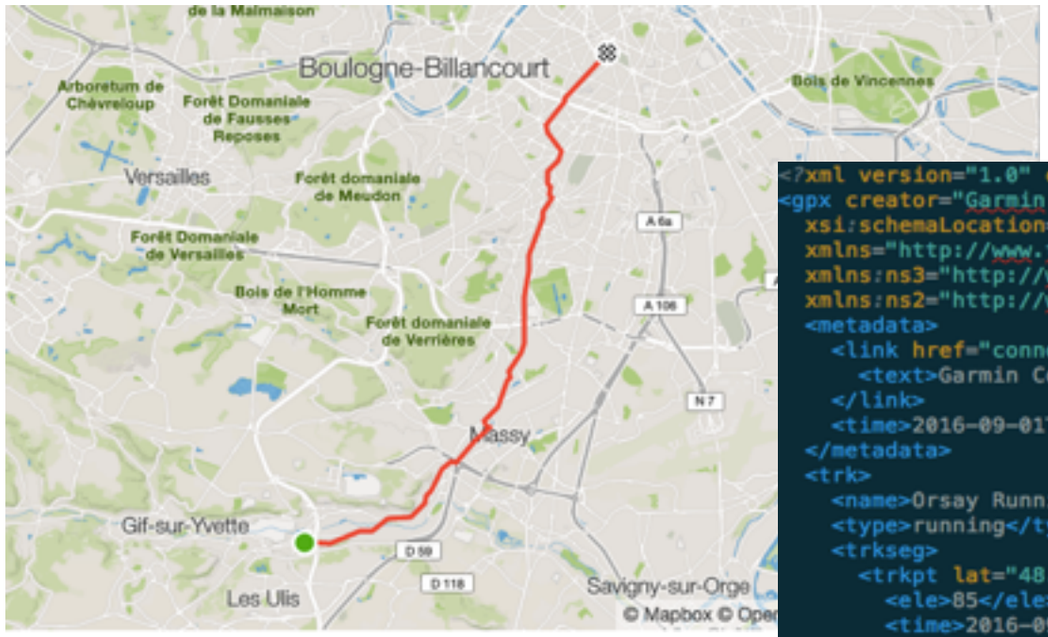
```
<?xml version="1.0"?>
<api batchcomplete="">
  <query>
    <normalized>
      <n from="University_of_Paris-Saclay" to="University of Paris-Saclay" />
    </normalized>
    <pages>
      <page _idx="42196987" pageid="42196987" ns="0" title="University of Paris-Saclay">
        <revisions>
          <rev contentformat="text/x-wiki" contentmodel="wikitext" xml:space="preserve">{{Infobox University
|name         = University of Paris-Saclay
|native_name = Université Paris-Saclay
|image        = [[File:University of Paris-Saclay.png|250px]]
{{Photomontage
  |photo1a = Polytech ParisSud.JPG
  |photo1b = Danone Saclay.JPG
  |photo2a = CEA Saclay.JPG
  |photo2b = Institut d'optique Graduate School.JPG
  |photo4a = LAL_Paris-Saclay.JPG
  |photo4b = Siege GSK Paris-Saclay.JPG
  |size    = 250
  |space   = 2
 }}
 |city  = [[Paris]], [[Versailles]], [[Saint-Quentin-en-Yvelines]], [[Saint-Aubin, Essonne|Saint-Aubin]]
 |country  = [[France]]
 |coor = {{coord|48.7117343|2.1712888|type:edu|display=inline,title}}
| campus  = [[Paris-Saclay]], [[Université de Versailles-Saint-Quentin-en-Yvelines|Versailles Saint-Quentin-en-Yvelines]],
[[Université Paris-Sud]]
 |president   = Gilles Bloch&lt;ref name = &quot;UpsPresident&quot;&gt;{{Cite web |title = Gilles Bloch elected President of
Université Paris-Saclay |url = http://www.universite-paris-saclay.fr/en/news/gilles-bloch-has-been-elected-president-of-
universite-paris-saclay |work = universite-paris-saclay.fr |accessdate = 29 May 2016}}&lt;/ref&gt;
 |area km2    = 77 &lt;ref name=&quot;area&quot;&gt;{{Cite web |title = About Paris-Saclay |url =
http://orientation.blog.lemonde.fr/2013/05/19/paris-saclay-bientot-20-de-la-recherche-francaise-sur-un-immense-campus/ |work
= Lemonde.fr}}&lt;/ref&gt;
 |established = December 29, 2014&lt;ref name = &quot;UpsEstablished&quot;&gt;{{Cite web |title = Establishment of Université
Paris-Saclay «Décret n° 2014-1674 du 29 décembre 2014» |url =
https://www.legifrance.gouv.fr/eli/decret/2014/12/29/MENS1425099D/jo |work = legifrance.gouv.fr |accessdate = 29 May
2016}}&lt;/ref&gt;
 |students    = {{formatnum:60000}}&lt;ref name=&quot;EPPS&quot;&gt;{{Cite web |title = A World Class University |url =
http://www.epps.fr/en/a-global-cluster/a-world-class-university/ |work = epps.fr}}&lt;/ref&gt;
```

XML Example: Open Maps




```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882" visible="true"
version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744"
visible="true" version="1" changeset="323878" timestamp="2008-05-03T13:39:23Z"/>
 <node id="1831881213" version="1" changeset="12370172" lat="54.0900666" lon="12.2539381"
user="lafkor" uid="75625" visible="true" timestamp="2012-07-20T09:43:19Z">
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" user="SvenHRO" uid="46882" visible="true"
version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5" changeset="4142606"
timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Straße"/>
 </way>
 <relation id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637"
timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
  ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
```

XML Example: GPS data




```
<?xml version="1.0" encoding="UTF-8"?>
<gpx creator="Garmin Connect" version="1.1"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/11.xsd"
  xmlns="http://www.topografix.com/GPX/1/1"
  xmlns:ns3="http://www.garmin.com/xmlschemas/TrackPointExtension/v1"
  xmlns:ns2="http://www.garmin.com/xmlschemas/GpxExtensions/v3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <metadata>
    <link href="connect.garmin.com">
      <text>Garmin Connect</text>
    </link>
    <time>2016-09-01T09:42:42.000Z</time>
  </metadata>
  <trk>
    <name>Orsay Running</name>
    <type>running</type>
    <trkseg>
      <trkpt lat="48.696818780153989791870117187 5" lon="2.187100304290652275085449218 75">
        <ele>85</ele>
        <time>2016-09-01T09:42:42.000Z</time>
        <extensions>
          <ns3:TrackPointExtension>
            <ns3:hr>94</ns3:hr>
            <ns3:cad>61</ns3:cad>
          </ns3:TrackPointExtension>
        </extensions>
      </trkpt>
      <trkpt lat="48.696818109601736068725585937 5" lon="2.187121678143739700317382812 5">
        <ele>85</ele>
        <time>2016-09-01T09:42:43.000Z</time>
        <extensions>
          <ns3:TrackPointExtension>
            <ns3:hr>94</ns3:hr>
            <ns3:cad>61</ns3:cad>
          </ns3:TrackPointExtension>
        </extensions>
      </trkpt>
      <trkpt lat="48.696820372715592384338378906 25" lon="2.187208095565438270568847656 25">
        <ele>84.8000030517578125</ele>
        <time>2016-09-01T09:42:46.000Z</time>
        <extensions>
          <ns3:TrackPointExtension>
            <ns3:hr>97</ns3:hr>
            <ns3:cad>61</ns3:cad>
          </ns3:TrackPointExtension>
        </extensions>
      </trkpt>
      <trkpt lat="48.696838226169347763061523437 5" lon="2.187277749180793762207031250 0">
        <ele>84.59999847412109375</ele>
        <time>2016-09-01T09:42:48.000Z</time>
```

XML: Basics

- XML is a specialization of SGML
- W3C standard since 1998 (http://www.w3.org/XML)
- designed to be simple, generic and extensible
- piece of text that describes structure and data

XML: Basics

- associated with a tree
- divided into smaller pieces of data called elements (associated with nodes in tree):
  - XML documents contains elements
  - elements can contain elements
  - non-ambiguous hierarchical structure
- XML document: a root element containing all other elements

XML: Basics

- XML is a meta-language: allows to create markup-languages

- no need for a parser, one can fix one's own "language" — describe all admissible structures (allowed element names, how they can be put together, data types, …)

- done using XML type definition languages (DTD, Relax NG, XML Schema)

XML: Components

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM “book.dtd”>
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

Declarations

Root

XML: Declarations

- XML Declaration (optional)

```
<?xml version=“1.0” encoding=“utf-8” ?>
```

- Document type declaration (optional) — contains the structure, content and tags of a document

```
<!DOCTYPE book SYSTEM “book.dtd">
```

XML: Elements

- elements are delimited by tags
- tags are enclosed in angle brackets, e.g., `<author>`, `</book>`
- tags are case-sensitive: `<BOOK>` is not the same as `<book>`
- we can have: start tags: `<...>`, e.g., `<book>`, end tags: `</...>`, e.g., `</book>`
- a pair of matching start- and end tags defines an element
- attributes specify properties of an element, `<book title="Theory of Computation">`

XML: Elements

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM “book.dtd”>
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

Start tag

End tag

Element

XML: Elements

```
<element-name attr-dec1 … attr-decn>
  content
</element-name>
```

- arbitrary number of attributes allowed
- each attribute name occurs only once in the same element
- the element content can be: empty, text and/or one or more elements
- a pair of matching start- and end tags defines an element
- an empty element cab be abbreviated as, `<book … />`

XML: Prologue

```
<?xml param1 param2 … ?>
```

- each param is of the form `parameter-name="parameter-value"`

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

- XML version
- character encoding
- an XML document should have an XML declaration, but does not need to

XML: Prologue

```
<!DOCTYPE book SYSTEM “book.dtd">
```

- XML type definition language; here, DTD (XML Document Type Definition)

- each element is associated with its content model (regular expression)

```
books -> (book)*

book  -> title, author, publisher, edition?
```

XML: Others

- comments: `<!-- comment here -->`

- processing instructions: `<?php sql("SELECT * FROM books") … ?>`

- entity references: `<p> Text: &text </p>`

XML: Full Grammar

```
[1]    document    ::=   prolog element Misc*
[2]        Char    ::=   a Unicode character
[3]           S    ::=   (‘ ’ | ‘\t’ | ‘\n’ | ‘\r’)+
[4]    NameChar    ::=   (Letter | Digit | ‘.’ | ‘-’ | ‘:’
[5]        Name    ::=   (Letter | '_' | ':') (NameChar)*

[22]     prolog    ::=   XMLDecl? Misc* (doctypedecl Misc*)?
[23]    XMLDecl    ::=   '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>
[24]VersionInfo    ::=   S'version'Eq(""'VersionNum'"'|"'VersionNum'")
[25]         Eq    ::=   S? '=' S?
[26]VersionNum     ::=   '1.0‘

[39]    element    ::=   EmptyElemTag
                       | STag content Etag
[40]       STag    ::=   '<' Name (S Attribute)* S? '>'
[41] Attribute     ::=   Name Eq AttValue
[42]       ETag    ::=   '</' Name S? '>‘
[43]    content    ::=   (element | Reference | CharData?)*
[44]EmptyElemTag   ::=   '<' Name (S Attribute)* S? '/>‘

[67] Reference     ::=   EntityRef | CharRef
[68] EntityRef     ::=   '&' Name ';‘
[84] Letter        ::=   [a-zA-Z]
[88] Digit         ::=   [0-9]
```

XML: Parsing Algorithm

XML grammar = EBNF, formed of `lhs::=rhs`

Algorithm

1. starting with **`document`**, expand symbols using the rules, thus constructing a parse tree

2. leaves are the characters which have no further expansion

3. XML parsed if it perfectly matches the parses tree left-to-right traversal

XML: Dialects

- XHTML (W3C) - XML version of HTML
- SVG (W3C) - Vector Graphics
- MathML (W3C) - mathematical formulas
- X3D (Web3D) - 3D graphics
- SOAP (RPC using HTTP), WSDL (W3C) - Web services
- RDF (W3C), OWL (W3C) - metadata/knowledge in Semantic Web
- …

XML: Correctness

Two levels of correctness:

1. well-formed XML (minimal, weak requirement): data is organized in a tree-like structure

2. valid XML (optional, strict requirement): needs to conform to a specific dialect (DTD, …)

XML: Well-Formed

- the set of tags/elements is not fixed, one can define anything (unlike HTML: `<h1>`, `<p>`, …)

- elements can be nested, and of arbitrary depth:
  `<book> <book> <book> … </book> </book> </book>`

- the same element can occur multiple times in the document

XML: Well-Formed

XML is well-formed if:

1. there is exactly one root element
2. tags inside <> are correct (escape characters)
3. tags are properly nested
4. attributes are unique and are quoted
5. no comments inside tags

Very weak property = we can parse a document into a tree

XML: Well-Formed

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM “book.dtd”>
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

Is this XML well-formed?

XML: Valid

- document type: contract between data producer and data consumer
- validation allows checking of contract
- error detection is left to the parser
- simpler applications, higher-speed XML throughput (if input validated, no need for all runtime checks)

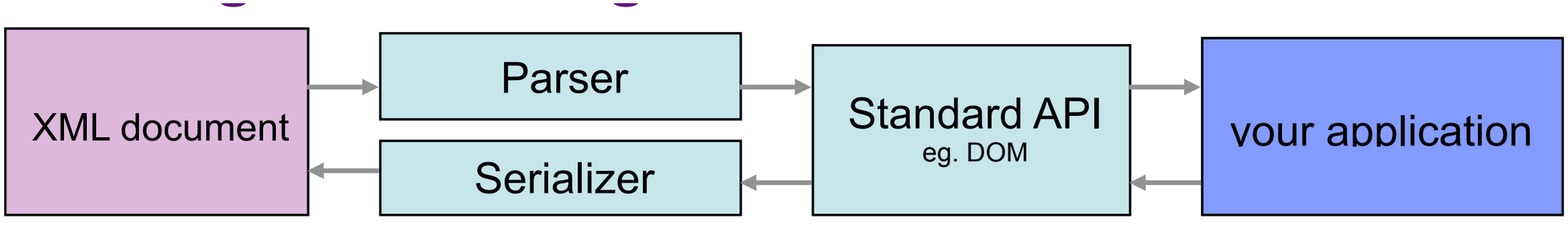
Parsing and Processing XML

Processing Model

- validation better than writing code/parser
- XML application operate on the logical tree view — provided by an XML parser
- how does an XML parser communicate the tree structure to the application?

Parsing and Serializing XML

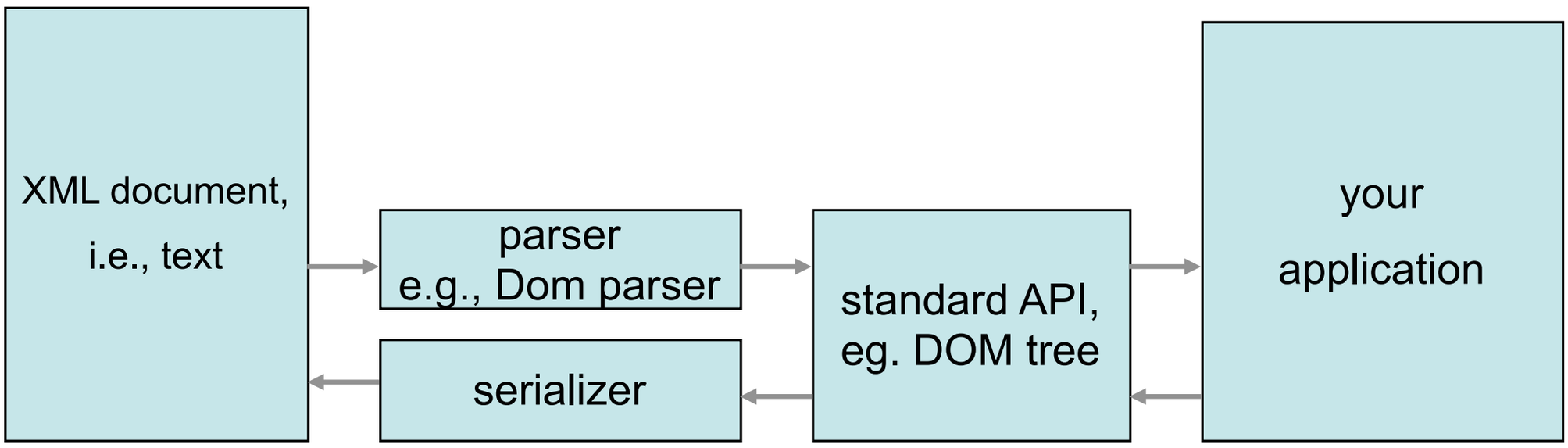
- parser: reads and analyzes XML document, may generate a parse tree that reflects document
- serializer: takes a data structure (trees, linked objects) and generates an XML document



Credit: B. Parsia, C. Hedeler, U. Sattler; Univ. of Manchester

DOM (Document Object Model)

- DOM tree: internal representation of XML, accessing XML document in the form of a tree

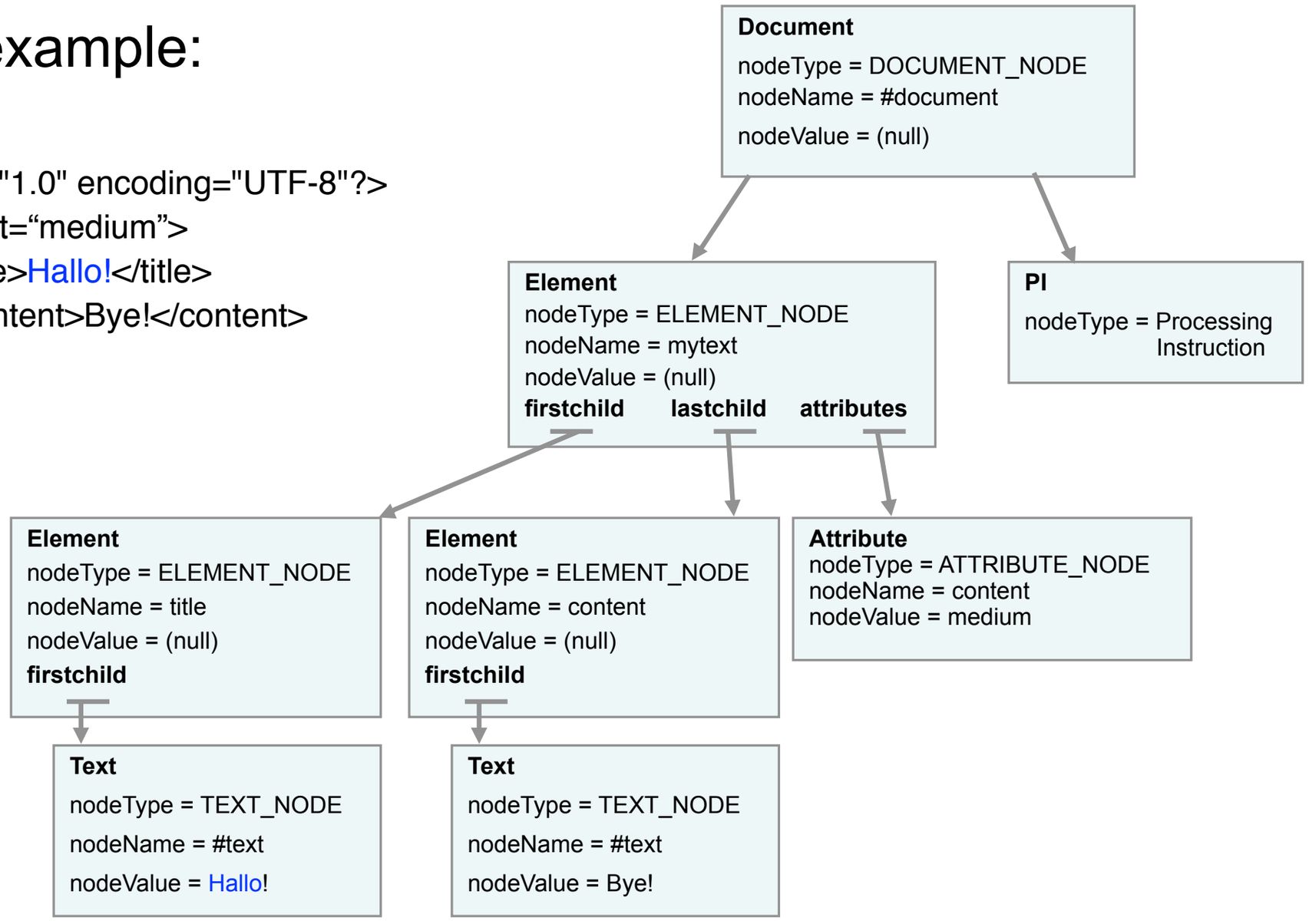


Credit: B. Parsia, C. Hedeler, U. Sattler; Univ. of Manchester

DOM tree

A simple example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mytext content="medium">
        <title>Hallo!</title>
        <content>Bye!</content>
</mytext>
```



Credit: B. Parsia, C. Hedeler, U. Sattler; Univ. of Manchester

DOM tree

Correspondences between DOM and XML document:

- XML document D = tree t(D)
- element e in D = node t(e) in t(D)
- empty element = leaf node

DOM details

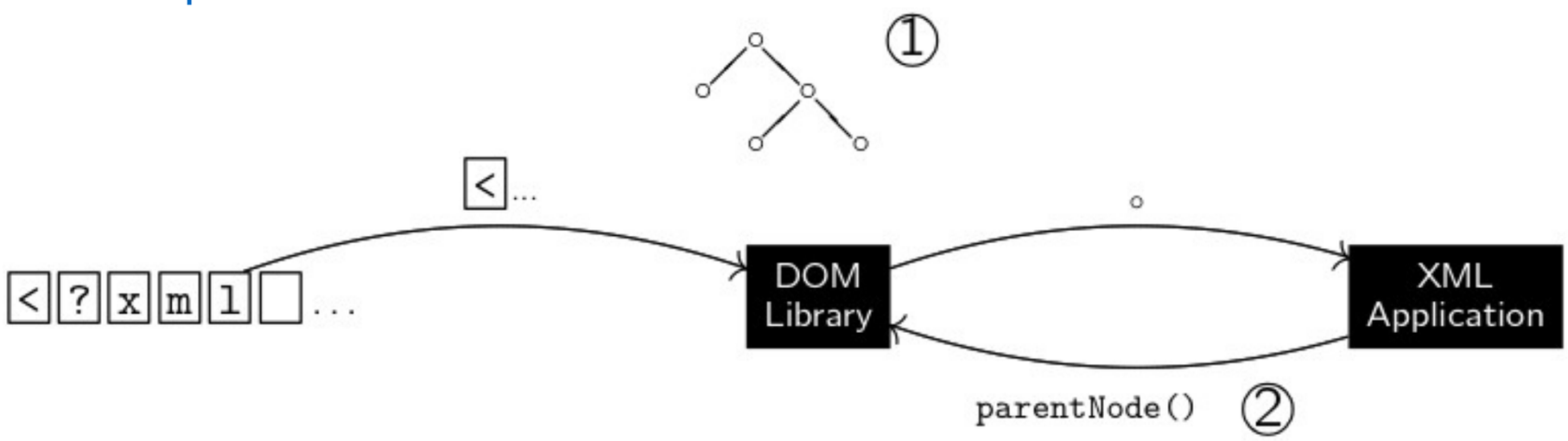
- DOM (W3C): a language- and platform-neutral view of XML documents

- APIs for DOM exist for a wide variety of programming languages (Java, C++, Python, …)

DOM details

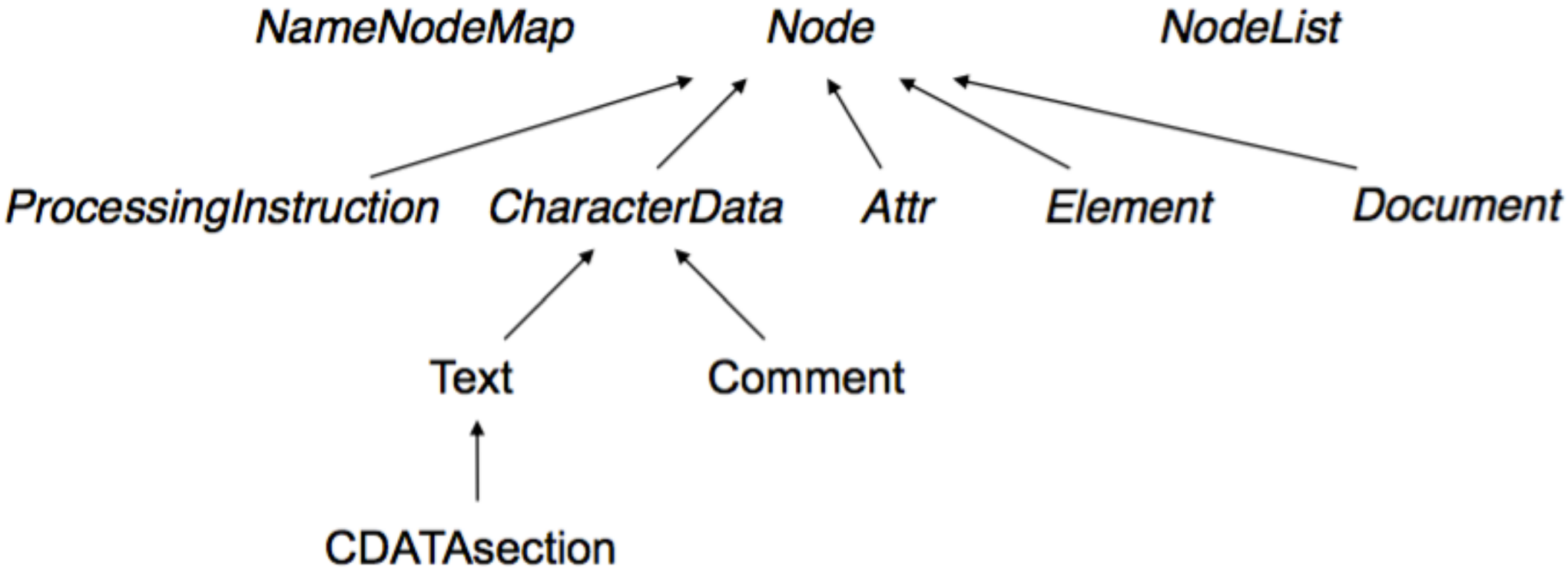
DOM design based on two concepts:

1. An XML processor parses the XML document, and keeps a DOM tree in-memory
2. XML application then issues API calls to explore and manipulate the XML document



Credit: P. Genevès, CNRS

DOM Core

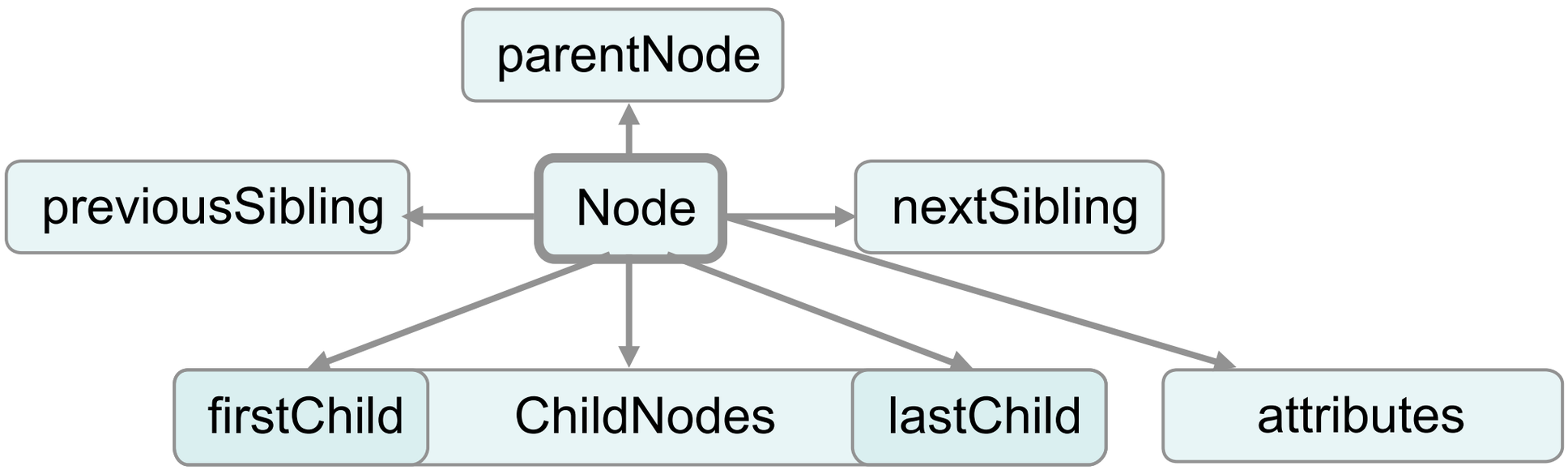

NameNodeMap
Node
NodeList
ProcessingInstruction
CharacterData
Attr
Element
Document
Text
Comment
CDATAsection

DOM Core

DOM type	Method		Comment
Node	nodeName nodeValue	: DOMString	redefined in subclasses
	parentNode	: Node	
	firstChild	: Node	leftmost child
	nextSibling	: Node	returns NULL for root elem or last child or attributes
	childNodes	: NodeList	
	attributes	: NamedNodeMap	
	ownerDocument	: Document	
	replaceChild	: Node	
Document	createElement	: Element	creates element with given tag name
	createComment	: Comment	
	getElementsByTagName	: NodeList	list of all Elem nodes in document order

DOM interface

Can use the DOM **`Node`** interface, which has the the following attributes:



- other methods: `appendChild`, `hasAttributes`, `insertBefore`, …

DOM performance

- The two step approach enables to have an object that is random-access
- Random-access: all the XML is in main memory:
    1. issues with heavy swapping activity
    2. out-of-memory if application not properly programmed

DOM performance

- DOM = memory hungry!
- Even if we only need a small part of the XML, everything is kept in memory! — size can be several times that of the original XML

- **Solutions?**

DOM performance

- Solution 1: preprocess the XML to only have the relevant parts in memory

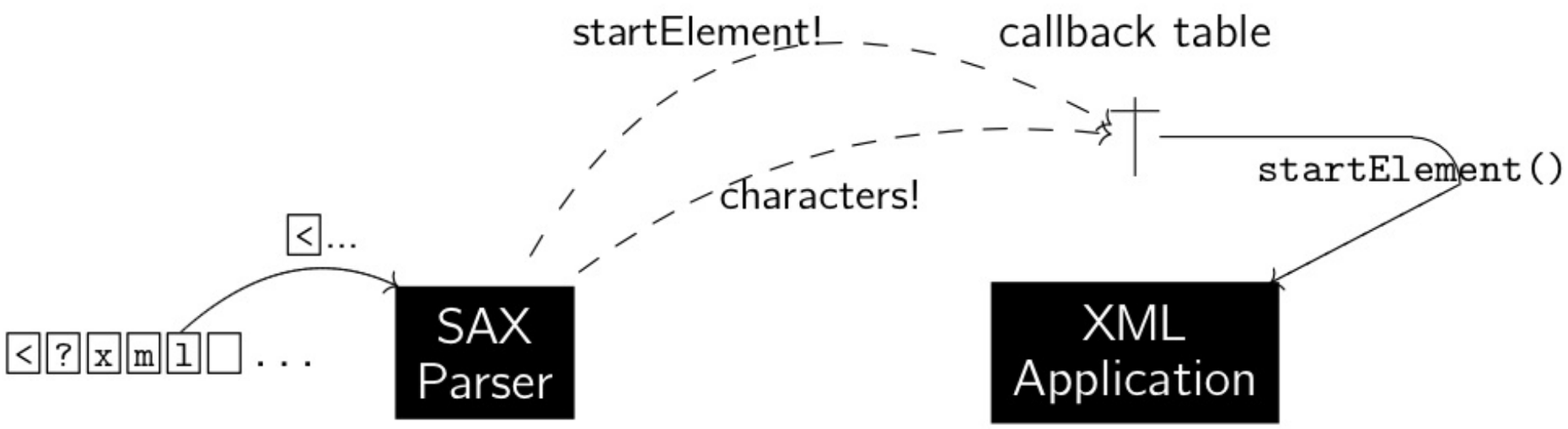
- Solution 2: use a different way to process XML (SAX)

SAX (Simple API for XML)

- SAX is not an W3C standard, but well maintained
- SAX processes in constant space:
  - ✦ no intermediate data structure
  - ✦ SAX sends events to the application for each piece of XML (streaming)

SAX Operation

- reads input sequentially and only once
- no memory of what has been sent is retained — only events are sent
- application can act in parallel with the parsing process



Credit: P. Genevès, CNRS

SAX Operation

SAX reports the following events:

Event	**...reported when seen**	**Parameters sent**
startDocument	`<?xml...?>`[8]	
endDocument	⟨EOF⟩	
startElement	<*t* a_1=v_1 . . . a_n=v_n>	t, $(a_1, v_1), \ldots, (a_n, v_n)$
endElement	</*t*>	t
characters	*text content*	Unicode buffer ptr, length
comment	<!--*c*-->	c
processingInstruction	<?*t pi*?>	t, *pi*
	⋮	

SAX example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM “book.dtd”>
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

What is the order of operations? It corresponds to what?

SAX Details

- Order of SAX events is determined by a preorder traversal of the document tree

- Well-defined, but may not allow all possible queries on the XML

- If really needed, enough to rebuild the whole XML tree. **How?**

JSON

JSON
(JavaScript Object Notation)

- another tree data structure formalism — simpler than XML
- coming from JavaScript
- details at http://www.json.org/xml.html

Twitter API (JSON)

Example Request

```
GET

https://api.twitter.com/1.1/statuses/show.json?
id=210462857140252672
```

Example Result


```
{
 "coordinates": null,
 "favorited": false,
 "truncated": false,
 "created_at": "Wed Jun 06 20:07:10 +0000 2012",
 "id_str": "210462857140252672",
 "entities": {
  "urls": [
   {
    "expanded_url": "https://dev.twitter.com/terms/display-
guidelines",
    "url": "https://t.co/Ed4omjYs",
    "indices": [
     76,
     97
    ],
    "display_url": "dev.twitter.com/terms/display-\u2026"
   }
  ],
  "hashtags": [
   {
    "text": "Twitterbird",
    "indices": [
     19,
```


https://dev.twitter.com/rest/tools/console

JSON Principles

JSON - fragment of JavaScript from set of literals called items:

- atomic: numbers, bools, strings
- arrays (composite): `[1, 2, “three”, “four”]`
- objects (composite) — sets, unordered lists, associative arrays: `{“first”: “Michael”, “second”: “Sipser”}`
- can be nested: `[{“one”:1,”two”:2},”name”: {“first”:”Michael", “second”: “Sipser”}]`

JSON - XML

```
{“book”: {
   “title”: “Artificial Intelligence”,
   “publisher: “Pearson”,
   “authors”:{
      “author”:[
         {“first”: “Peter”, “last”:
         “Norvig”},
         {“first”: “Stuart”, “last”:
         “Russell”}
      ]
}
}}
```

```
<book title=“Artificial
Intelligence” publisher=“Pearson">
  <authors>
    <author first=“Peter"
    last=“Norvig" />
    <author first=“Stuart"
    last=“Russell" />
  </authors>
</book>
```

JSON - XML

```
{“book”: {
   “title”: “Artificial Intelligence”,
   “publisher: “Pearson”,
   “authors”:{
      “author”:[
         {“first”: “Peter”, “last”:
         “Norvig”},
         {“first”: “Stuart”, “last”:
         “Russell”}
      ]
}
}}
```

```
<book title=“Artificial
Intelligence” publisher=“Pearson">
  <authors>
    <author first=“Peter"
    last=“Norvig" />
    <author first=“Stuart"
    last=“Russell" />
  </authors>
</book>
```

Order of children matters!

XML to JSON

- Elements are mapped to objects — `ElementName : contents`

- `contents` is a list

- attributes are delimited by an object **{…}**

- child elements are in an array **[…]**

- empty elements require an explicitly empty list

- no attributes requires an explicitly empty object

JSON Advantages

- much simpler to understand: less verbose, more humanly readable

- simpler processing: XML needs DOM/SAX, JSON operates on dictionaries / lists

- good for config files (name-value pairs)

Validating XML: DTD

DTD (Document Type Definition)

- XML need a schema: a way to tell a parser that the document is valid in regards to what is expected
- different (more strict) concept to well-formedness
- many different schema languages
- we will see first DTD (Document Type Definition)

DTD

- defines the allowed tags/elements in a document
- the contents of each elements by way of regular expressions
- the attributes of an element, and their respective types

DTD: Syntax

- `<!ELEMENT elem_name elem_regexp>` — an element named `elem_name` contains elements described by the regular expression `elem_regexp`

- `<!ATTLIST elem_name att_name att_type att_values>` — the element `elem_name` has an attribute named `att_name` of type `att_type` and having possible values described by `att_values`

DTD: Syntax

- regular expressions are formed of `*`,`+`,`?`, sequence `[,]`, `EMPTY`, `ANY`, `#PCDATA` (text)

- attribute types are `ID` (primary key), `IDREF` (foreign key), `CDATA` (text), `v1|v2|,…,vn` (fixed value list)

- attribute values are `v` (default value), `#REQUIRED` (mandatory attribute), `#IMPLIED` (optional attribute), `#FIXED v` (constant value `v`)

DTD Example

Books.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "book.dtd">
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

book.dtd

```
<!ELEMENT books (book*)>
<!ELEMENT book (publisher,edition, authors)>
<!ATTLIST book title CDATA #REQUIRED>
<!ELEMENT publisher #PCDATA>
<!ELEMENT edition #PCDATA>
<!ELEMENT authors (author+)>
<!ELEMENT author (first,last)>
<!ELEMENT first #PCDATA>
<!ELEMENT last #PCDATA>
```

Is this XML valid? What is the DTD for validation?

DTD Example

Books.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "book.dtd">
<books>
  <book>
    <title>Theory of Computation</title>
    <author>Michael Sipser</author>
    <publisher>Cengage Learning</publisher>
    <year>2012</year>
    <edition>3</edition>
  </book>
  <book>
    <title>Artifical Intelligence</title>
    <author>Peter Norvig</author>
    <author>Stuart Russell</author>
    <year>2013</year>
    <edition>3</edition>
    <publisher>Pearson</publisher>
  </book>
</books>
```

book.dtd

```
<!ELEMENT books (book*)>
<!ELEMENT book (publisher,edition, authors)>
<!ATTLIST book title CDATA #REQUIRED>
<!ELEMENT publisher #PCDATA>
<!ELEMENT edition #PCDATA>
<!ELEMENT authors (author+)>
<!ELEMENT author (first,last)>
<!ELEMENT first #PCDATA>
<!ELEMENT last #PCDATA>
```

How can we modify the DTD to validate?

Further Reading

- Marc H. Scholl, XML and Databases http://www.cse.unsw.edu.au/~cs4317/09s1/XMLDB.pdf
- W3C, XML Tutorial http://www.w3schools.com/xml/
- W3C, XML Standard https://www.w3.org/XML/
- W3C, DOM Standard https://www.w3.org/DOM/
- JSON Standard http://www.json.org/