HMIN103
Presentation des données du Web : XML

Federico Ulliana
UM, LIRMM, INRIA GraphIK Team

Slides collected from J. Cheney, S. Abiteboul, I. Manolescu,
P. Senellart, P. Genevès, D. Florescu, and the W3C

Intervenants

- Pierre Pompidor
- Federico Ulliana ulliana@lirmm.fr
  - Accueuil étudiants lundi 16h au LIRMM (bat.5 salle 3-130; rdv par mail)

Programme : XML 360°

Objectif du cours : *étudier le langage XML et voir ses applications principales*

- La famille des langages XML (et relatifs)

  XML, DTD, XPath/XQuery, Updates, JSON

- Présentation de données XML

  XSLT, HTML5/Js, WebGL

MCC

- 50% examen (2 sessions)

- 25% Projet (P. Pompidor)
- 25% TPs à rendre (F. Ulliana)

  (1 session)

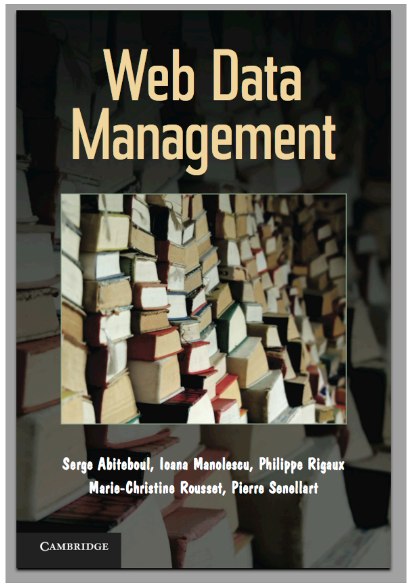
- Note TP finale : moyenne des notes
attention aux zeros! (documents non consignés)

Dates Rendus de TP

- Données et Schémas XML (27/09)
- XPath et XQuery (11/10)
- XML <-> Relationnel (18/10)
- XML Updates (25/10)

- Informations sur Moodle, en cas de problème : ulliana@lirmm.fr

Readings


Web Data Management
Serge Abiteboul, Ioana Manolescu, Philippe Rigaux
Marie-Christine Rousset, Pierre Senellart
CAMBRIDGE

Web Data Management - Abiteboul & al.

[WDM-XML] Chapter : Data-model

http://webdam.inria.fr//Jorge/files/wdm-datamodel.pdf

[WDM-DTD] Chapter : Schemas (only section 3)

http://webdam.inria.fr//Jorge/files/wdm-typing.pdf

What is XML?

eXtensible Markup Language [W3C 1998]

Ask five different people, get five different answers...

- a self-describing data format?
- a generalization of HTML?
- part of the DNA of computer science?
- “best thing since sliced bread?”
- a meta-language?

If XML is the solution, then what was the problem ?

Web data = by far the largest information system ever seen

- Billions of textual documents, images, PDF, multimedia files, provided and updated by millions of institutions and individuals.
- An anarchical process which results in highly heterogeneous data organization, steadily growing and extending to meet new requirements.
- New usage and applications of communication appear every day: yesterday P2P file sharing, today social networking, tomorrow ?

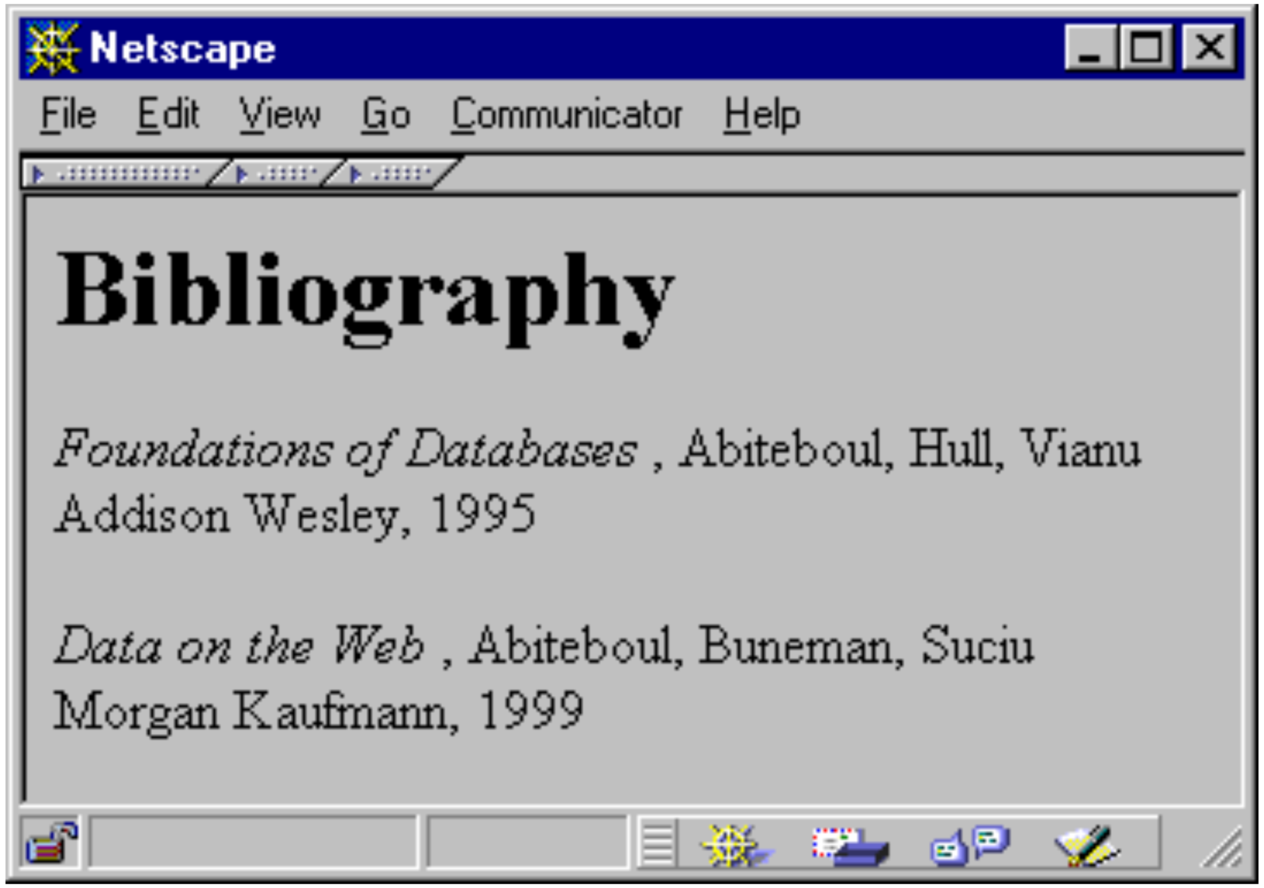
Challenges

- Master the size and extreme variability of Web information, and make it usable.
- Ensure long-term access to data. Write documents in 1998 and read in 2087.

The role of XML

Web data management has been primarily based on HTML, which describes presentation

- appropriate for humans, but falls short when it comes to software exploitation of data



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

```
<html>

<h1> Bibliography </h1>

<p> <i> Foundations of Databases </i>

        Abiteboul, Hull, Vianu

         <br> Addison Wesley, 1995

<p> <i> Data on the Web </i>

         Abiteboul, Buneman, Suciu

         <br> Morgan Kaufmann, 1999

</html>
```

The role of XML

XML describes content, and promotes machine-to-machine communication and data exchange

```
<bibliography>

    <book>
     <title> Foundations… </title>
     <author> Abiteboul </author>
     <author> Hull </author>
     <author> Vianu </author>
     <publisher> Addison Wesley
        </publisher>
      <year> 1995 </year>
    </book>…

</bibliography>
```

```
<html>

<h1> Bibliography </h1>

<p> <i> Foundations of Databases </i>

        Abiteboul, Hull, Vianu

         <br> Addison Wesley, 1995

<p> <i> Data on the Web </i>

         Abiteboul, Buneman, Suciu

         <br> Morgan Kaufmann, 1999

</html>
```

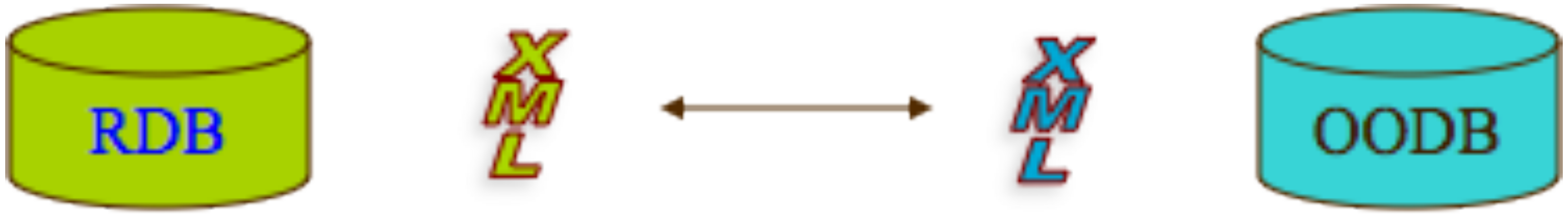
XML for Data Exchange

Web data exchange

- "*Data exchange is the oldest database problem*" (Phil Bernstein, 2003)

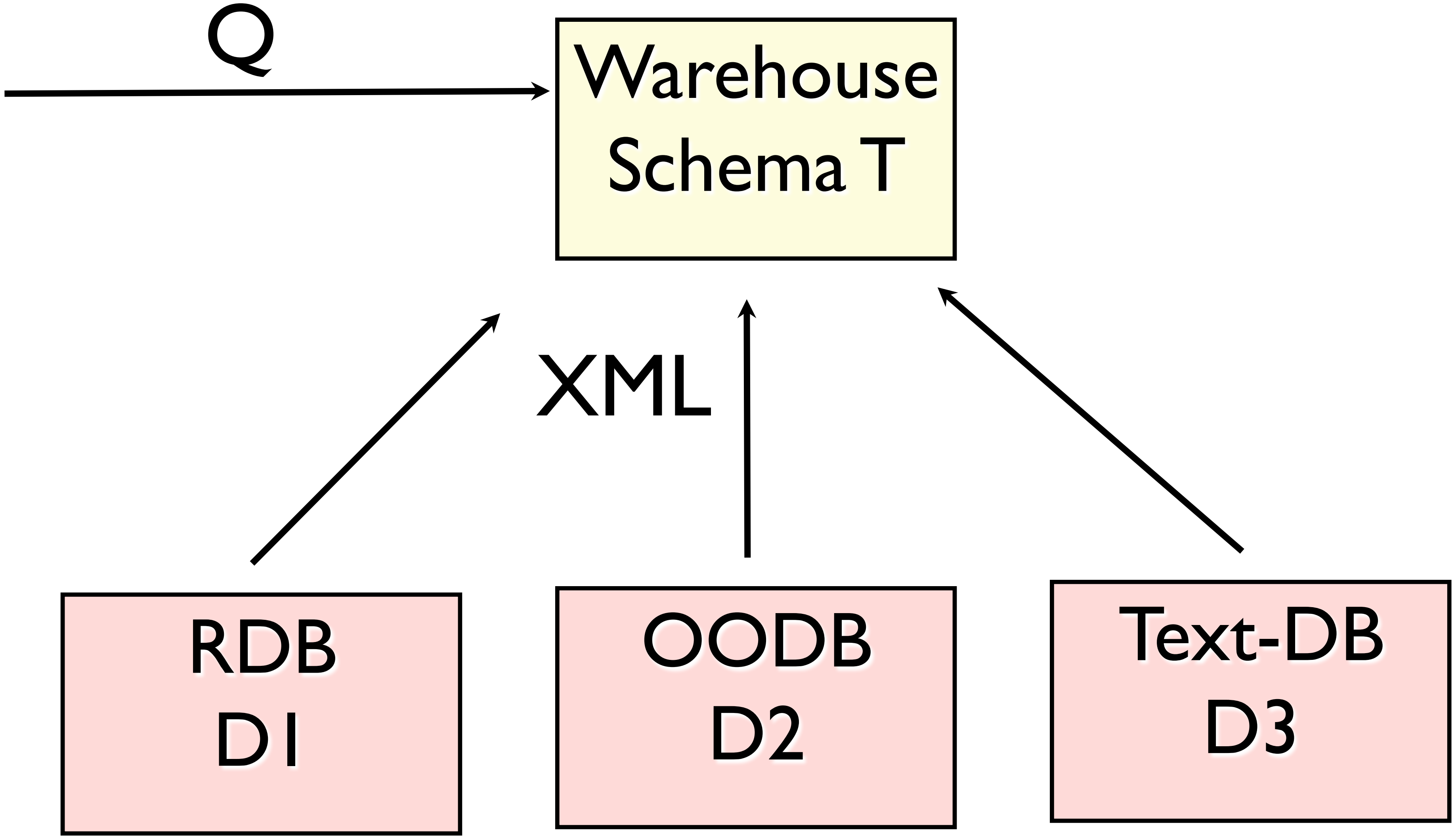
Massive demand

- across platforms/DBs
- across enterprises



- XML has become the prime standard for data exchange on the Web

Data integration



Q

Warehouse Schema T

XML

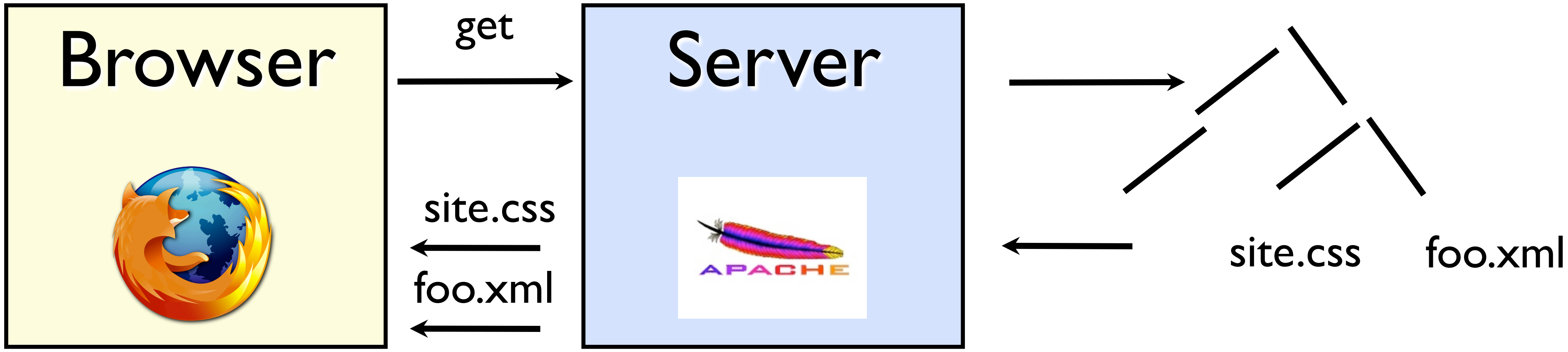
RDB D1

OODB D2

Text-DB D3

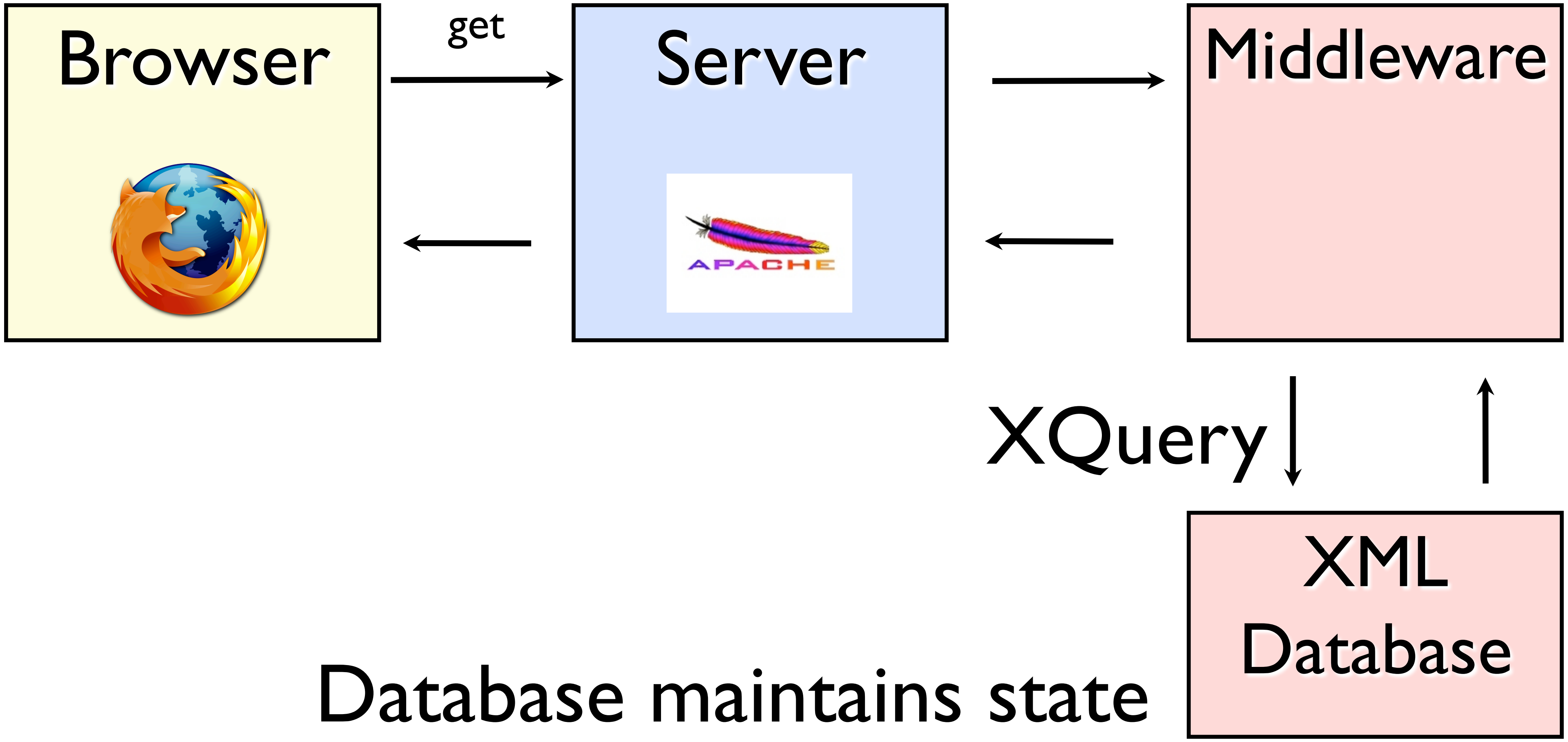
XML for Web Applications

Static Web site (XML+ CSS)



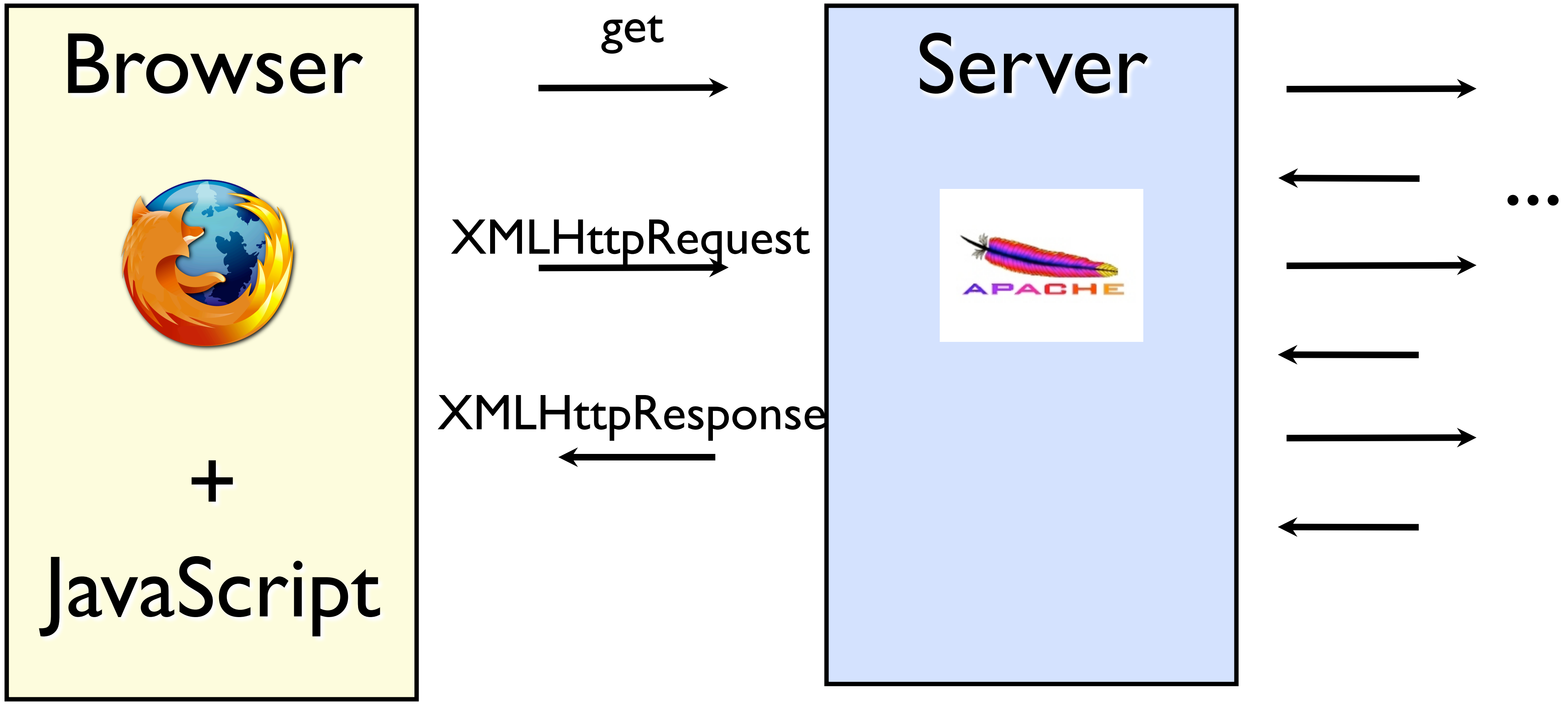
allows better factoring
into data + presentation

Dynamic Web site



Database maintains state

Asynchronous Javascript and XML



Browser

+

JavaScript

get

XMLHttpRequest

XMLHttpResponse

Server

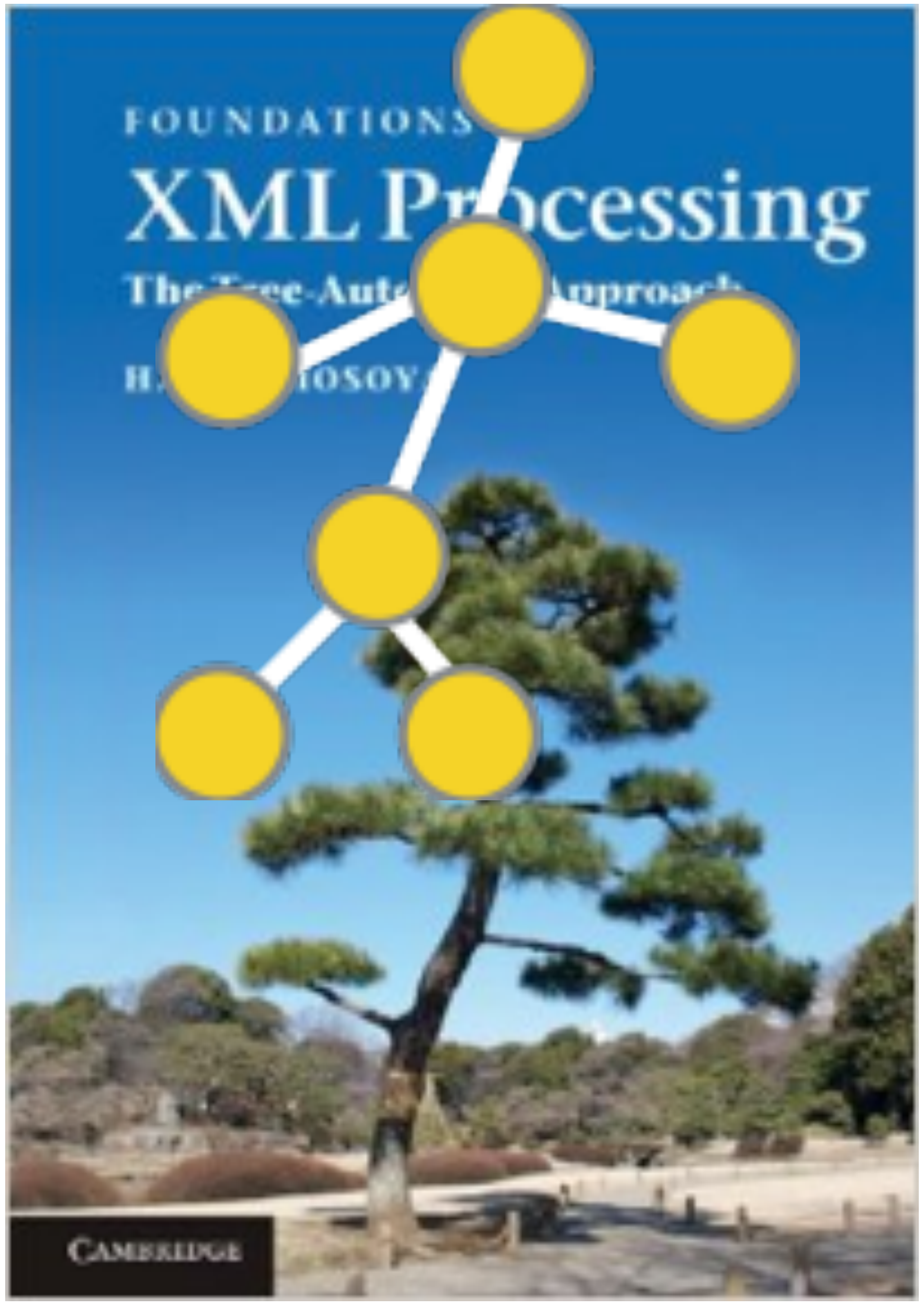
...

XML support in industry

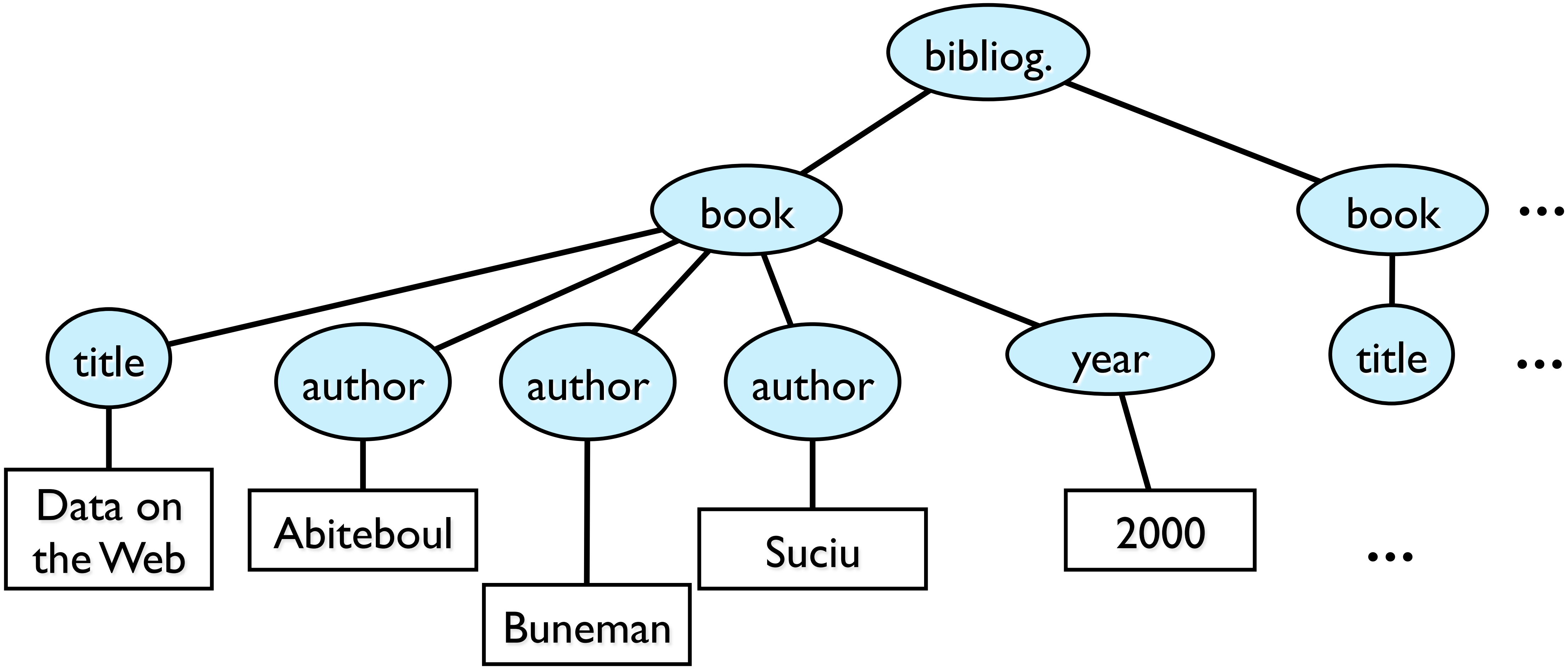
- Most commercial RDBMSs now provide some XML support
    - Oracle 11g - XML DB
    - IBM DB2 pureXML
    - Microsoft SQL Server - XML support since 2005
        - Language Integrated Query (LINQ) targets SQL & XML in .NET programs
- Data publishing, exchange, integration problems are very important
    - big 3 have products for all of these
    - SQL/XML standard for defining XML views of relational data

The essence of XML

Trees: the essence of XML

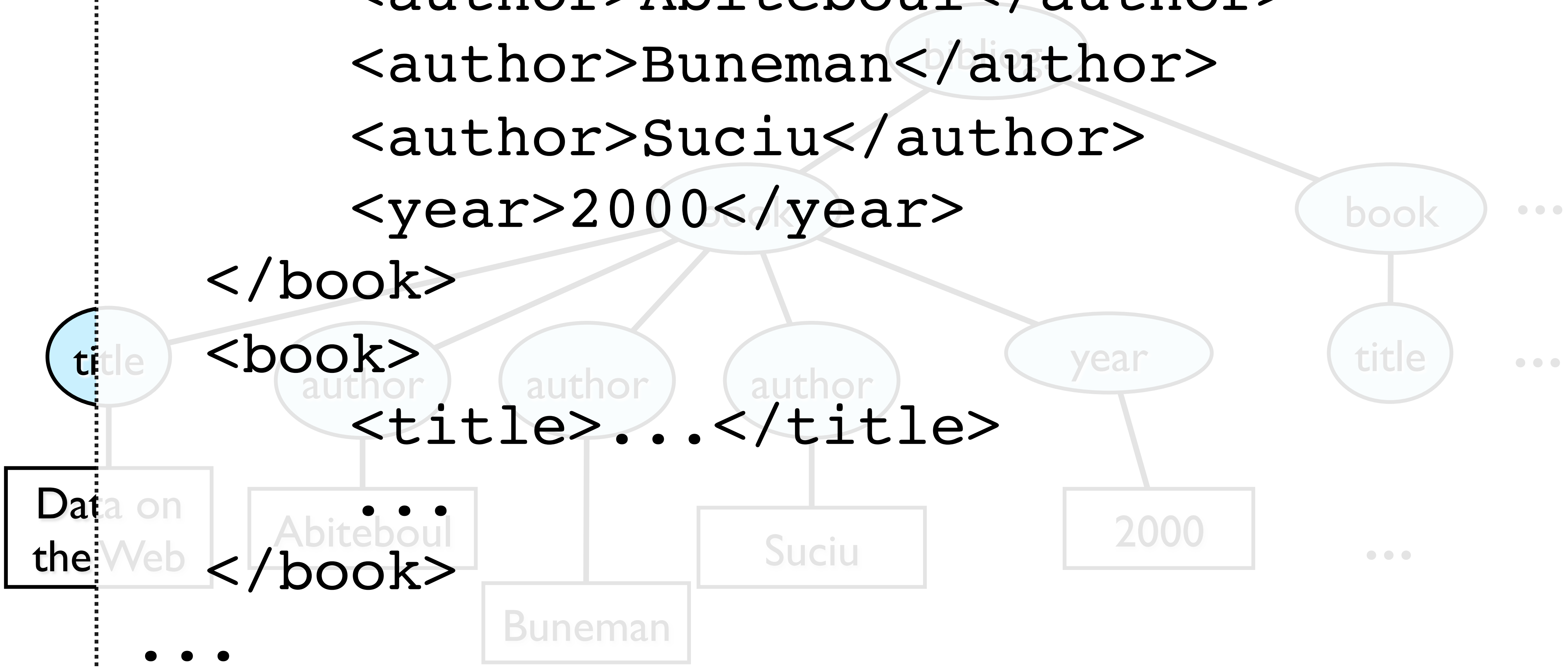


Trees: the essence of XML



Trees: the essence of XML

```
<bibliography>
   <book>
       <title>Data on the Web</title>
       <author>Abiteboul</author>
       <author>Buneman</author>
       <author>Suciu</author>
       <year>2000</year>
   </book>
   <book>
       <title>...</title>
        ...
   </book>
 ...
</bibliography>
```

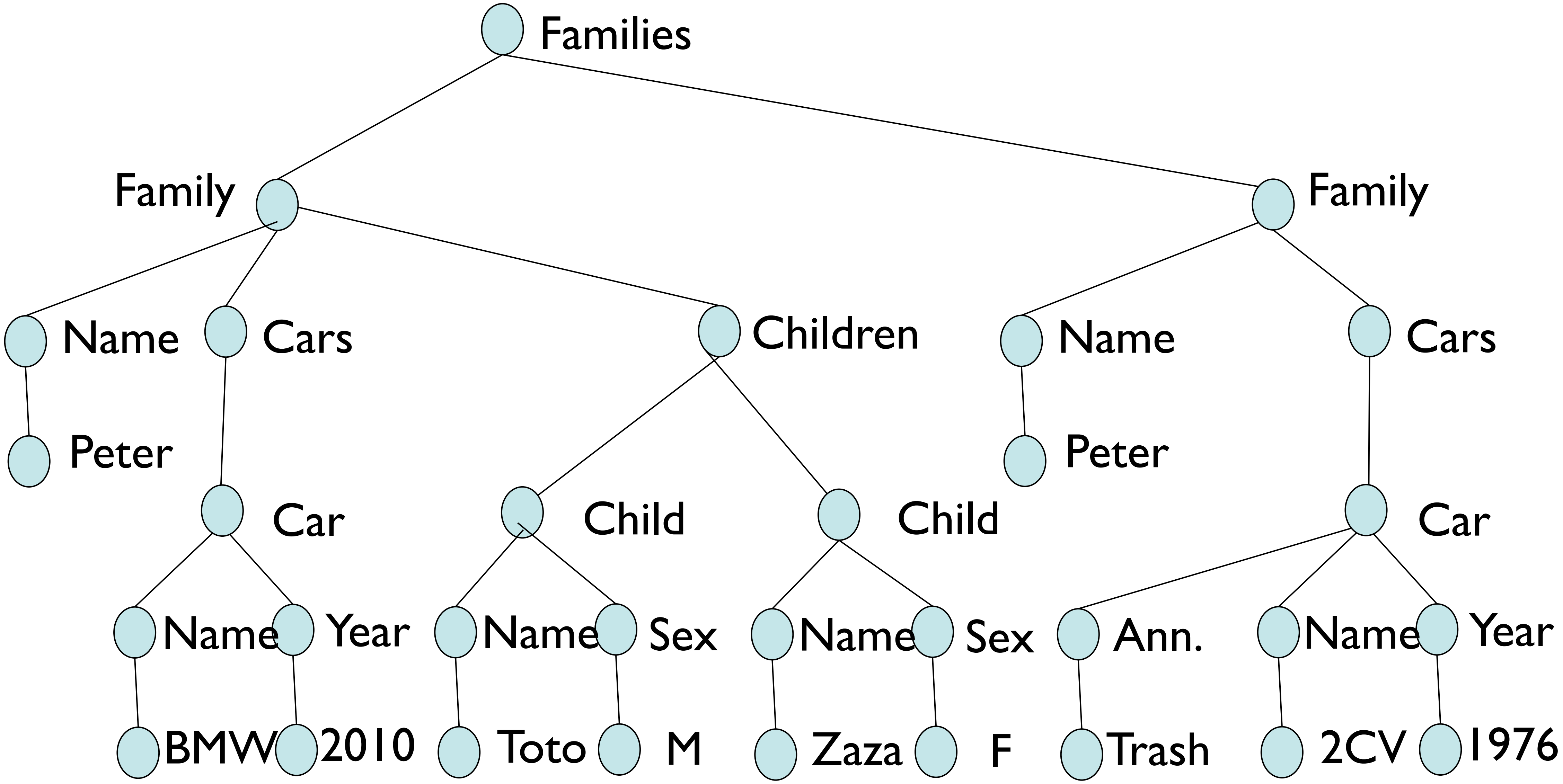


Trees: the essence of XML

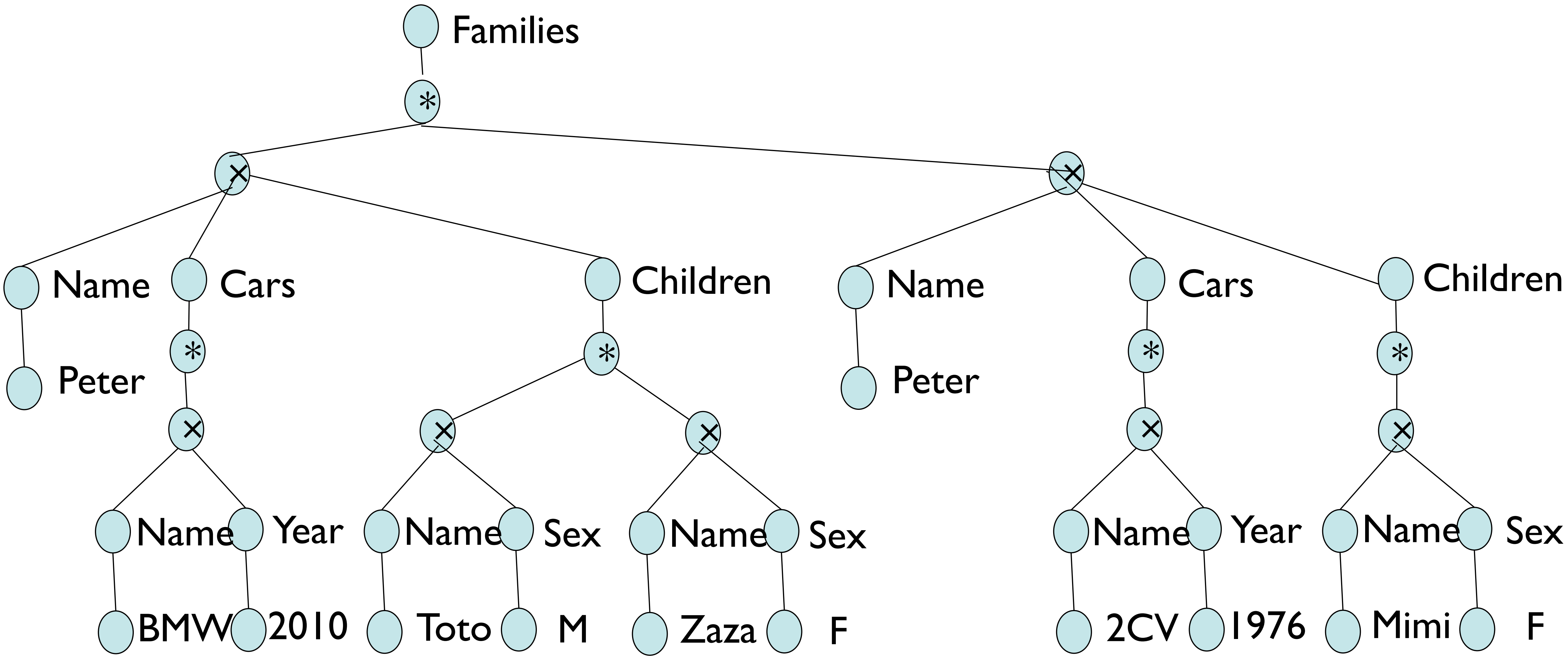
Using trees to represent data: an old idea

- From the 60s and IMS (Hierarchical model, IBM)
- From the 80s and object databases

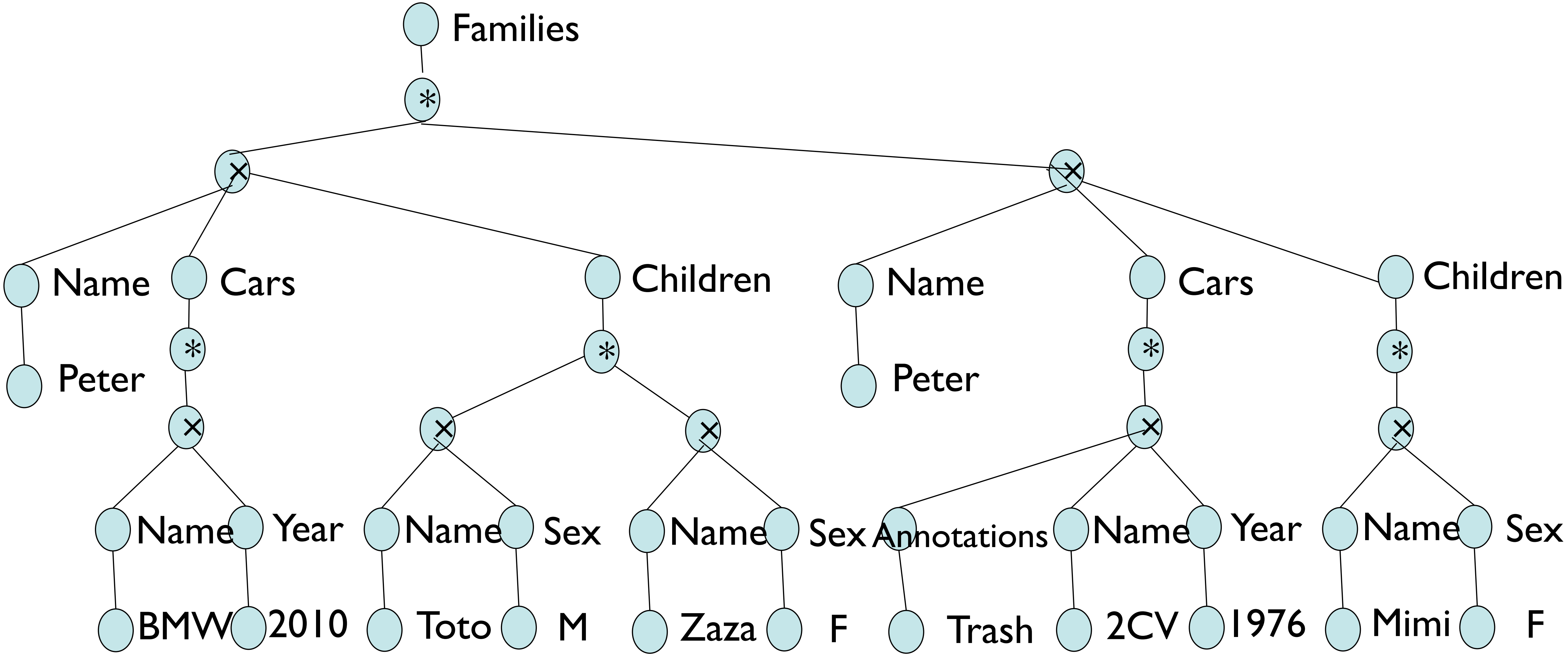
XML = ordered, labeled, unranked trees



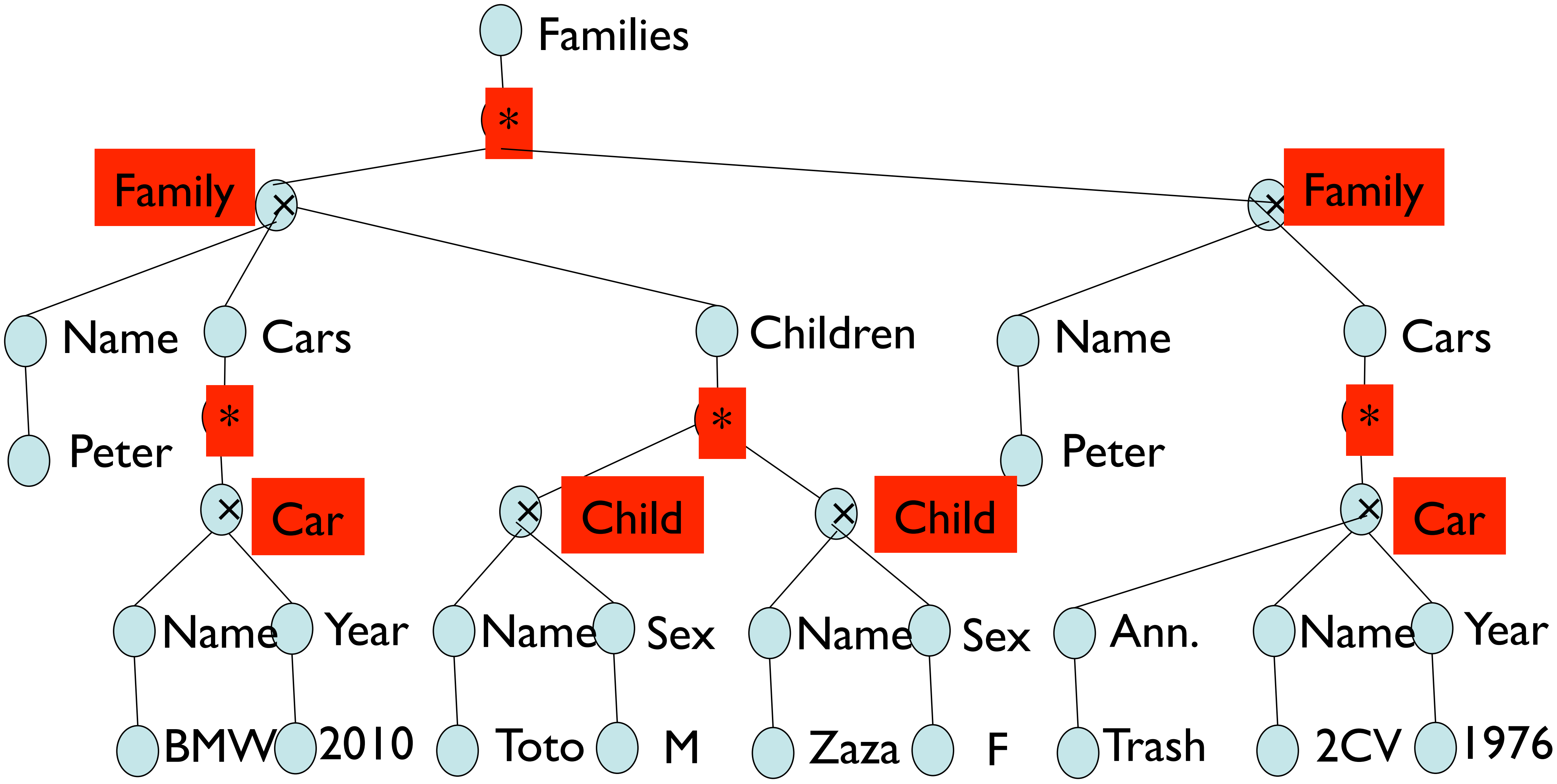
From database objects to XML trees



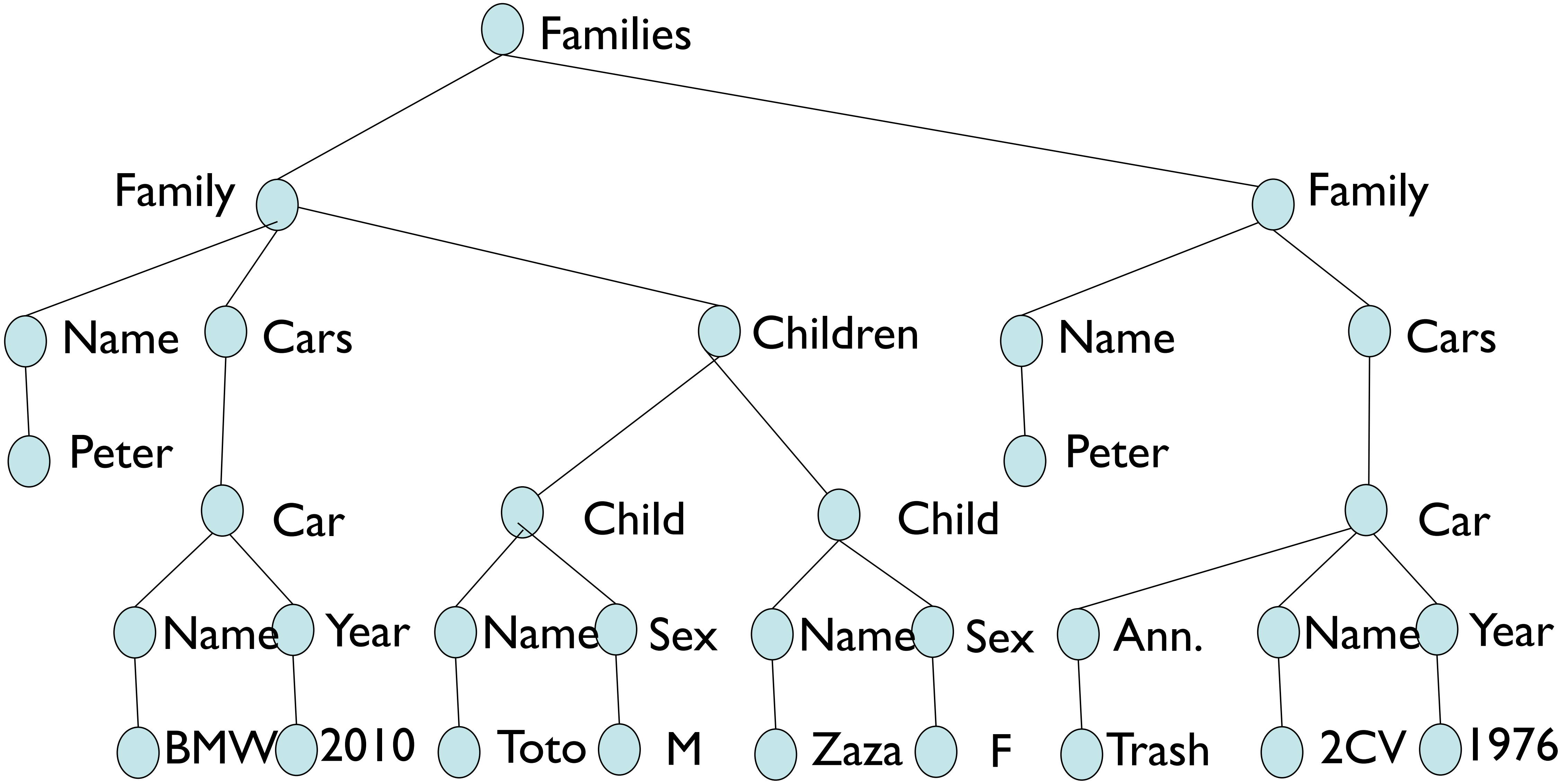
Revolution 1: more flexibility



Revolution 2: Remove some nodes; name *all*



XML = ordered, labeled, unranked trees



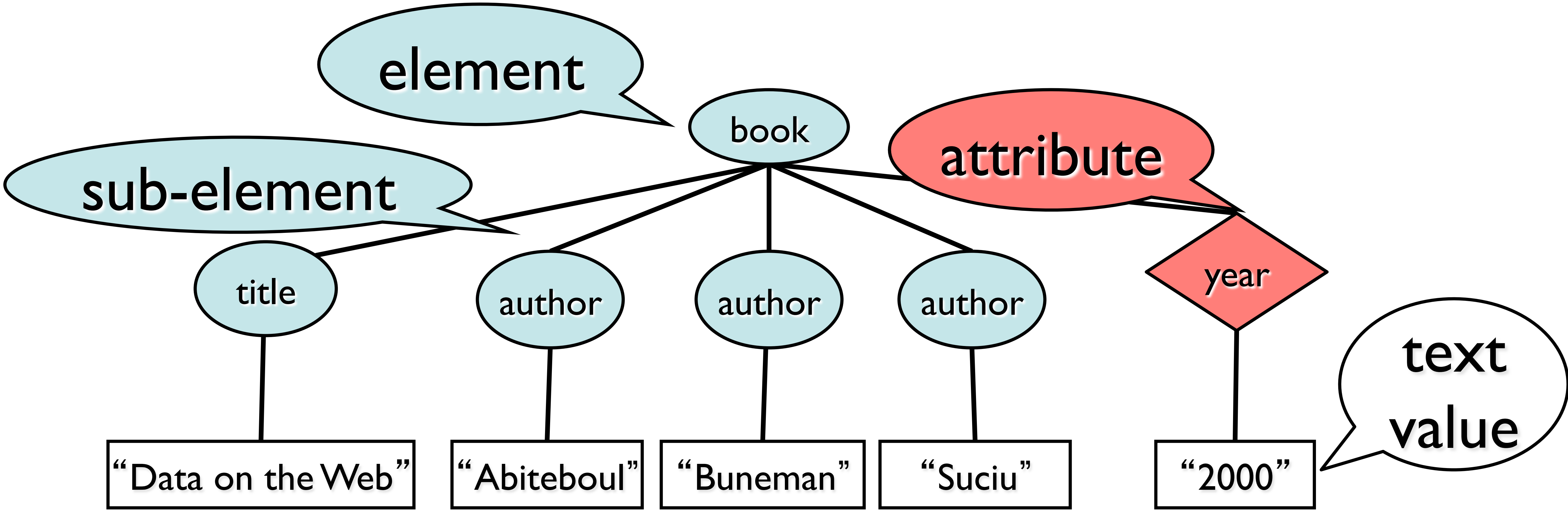
This is better adapted to a Web context

- Self describing data: no separation between schema-vocabulary (tags) and data
  - An XML document can exists without a schema !
  - NB. this was not the case for Object-DBs (and also relationals)

XML nodes

- XML nodes can be of many different kind
- We will focus on the three most important
  - Elements
  - Attributes
  - Text-nodes

A closer look : elements, attributes, and text-values



The syntax for elements, attributes, and text-values

element

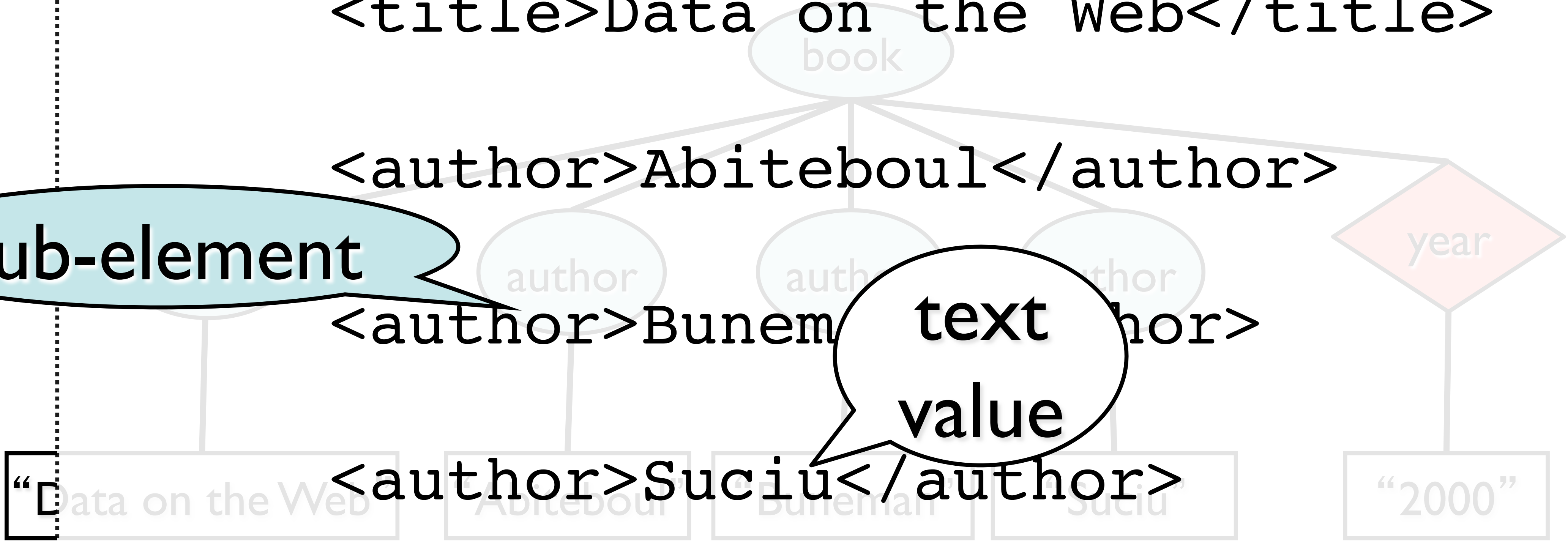
attribute

text value

```
<book year=“2000”>
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
</book>
```

sub-element

text value



Elements

Element: the segment between an start and its corresponding end tag

- Unique root element

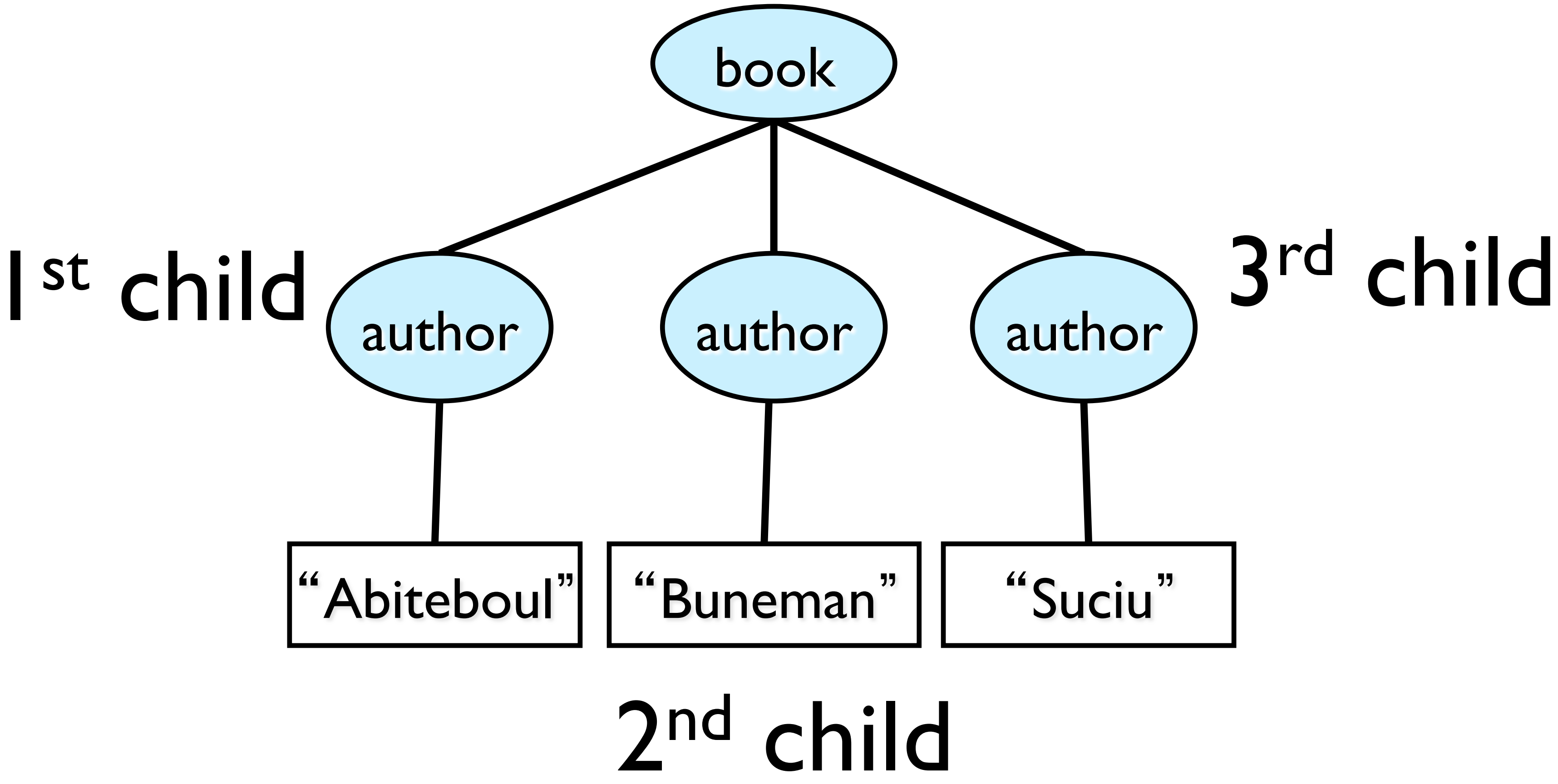
Nested tags can be used to express various “records” and “lists”

A document is

- **Well-formed**: if the opening & closing tags match (condition necessary to be deemed an XML document)

Ordering

XML elements are ordered



- How to represent sets in XML ?

Attributes : Syntax

A start tag may contain attributes describing the element

```
<picture>
    <height dim=“cm”> 2400</height>
    <width dim=“in”> 96 </width>
    <data encoding=“gif”> M05-+C$ … </data>
</picture>
```

Attributes can be used to mimic References

```
<person id = “011”   pal=“012”>
   <name> Barak Obama </name>
  </person>


<person id = “012”   pal=“011”>
   <name> Bill Clinton </name>
</person>
```

but IDs are not verified without a DTD

Attribute Structure

XML attributes cannot be nested (they are flat)

The names of XML attributes of an element must be unique.

- one can’t write

```
<person pal=“Blair” pal=“Clinton”>
```

Attributes are UNordered

```
<person   id = “011”    pal=“012”>
   <name> Bill Clinton </name>
</person>
```

is the same as

```
<person  pal=“012”    id = “011”>
   <name> Bill Clinton </name>
</person>
```

elements vs attributes

How to choose between elements and attributes?

doc 1

```
<note date="12/11/2002">

  <to>Tove</to>

  <from>Jani</from>

  <heading>
    Reminder
  </heading>

</note>
```

doc 2

```
<note>

  <date>
    12/11/2002
  </date>

 <to>Tove</to>

  <from>Jani</from>

  <heading>
    Reminder
  </heading>

</note>
```

doc 3

```
<note>

  <date>
   <day> 12 </day>
   <month>11</month>
   <year>2002</year>
  </date>

 <to>Tove</to>

  <from>Jani</from>

  <heading>
    Reminder
  </heading>

</note>
```

elements vs attributes

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable (for future changes)
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by program code
- attribute values are not easy to test against a DTD data

- Use attributes for IDs and Keys.
- Don't end up like this (this is not how XML should be used):

```
<note day="12" month="11" year="2002"
to="Tove" from="Jani" heading="Reminder">
</note>
```

Quiz : find errors

```
<books>
 <book id=”1’>
  <title>Data on the Web</title>
  <authors>
    <author id=”a1”>Abiteboul
    <author id=a2>Buneman  </author>
    <author id=’a3’>Suciu</authors>
  </author>
  <year>2000/year>
  <publisher>Addison-Wesley</publisher>
</books>
<foo>bar</foo>
```

Quiz

```
<books>
 <book id=”1”>
  <title>Data on the Web</title>
  <authors>
    <author id=”a1”>Abiteboul</author>
    <author id=”a2”>Buneman  </author>
    <author id=’a3’>Suciu</authors>
  </author>
  <year>2000</year>
  <publisher>Addison-Wesley</publisher>
</books>
<foo>bar</foo>
```

Other kinds of nodes

entity references: `&amp; &quot; &gt;`

- textual substitution; allows escaping special characters
- you can define your own if you want

processing instructions: `<? foo : bar ?>`

- can be used to pass information to processors

comments: `<!-- foo -->`

CDATA sections: - `<!CDATA[[ I <3 XML ]]>`

- allows including raw text (<, >, &, etc. uninterpreted)

Luckily, these are mostly irrelevant to use of XML for data

- but you need to know about them when writing reading/writing XML as text

Summing Up

- XML, the standard de-facto for data representation and exchange on the Web.

- Trees are the essence of XML, and there exists a precise syntax to define them.

DTDs

Schemas = Types (in XML)

In XML you can define your own markup languages

- via, external grammars, aka types, aka schemas

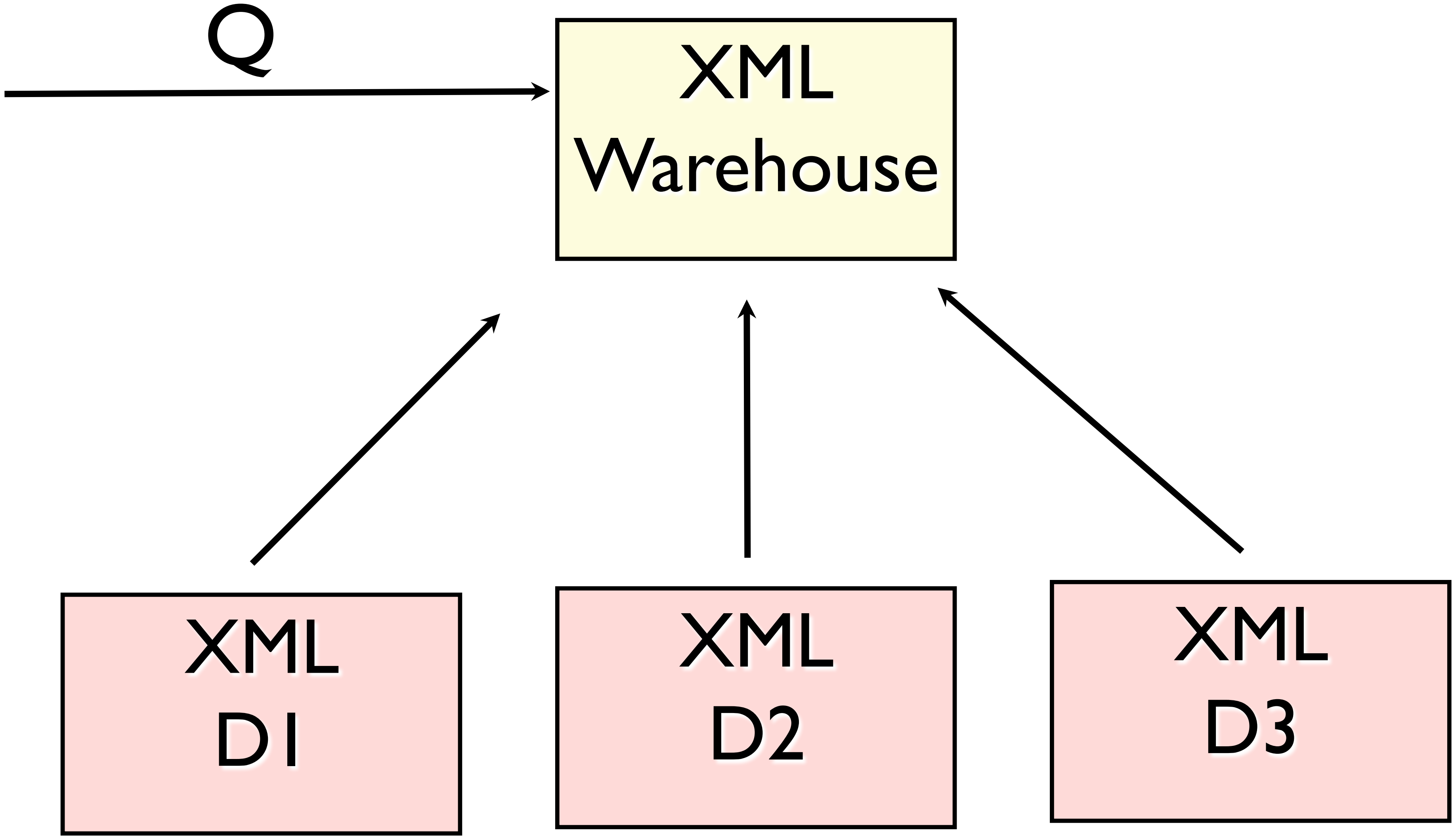
A document is

- **Valid**: means there is a schema and the document matches it

If they wanted XML to be a new universal language, why again defining schemas ?

Data Exchange/Integration

Easy only when a schema is agreed between peers



Goals of typing

Interoperability/reliability

- specify required, optional, default values

Consistency

- ensure updates or generated output is coherent

Efficiency

- use to organize storage
- use as basis for query optimization

Schemas

Many schema languages/formalisms have been considered

- DTDs (document type definition) (XML 1.0)
- XML Schema (W3C)
- Relax/NG (OASIS), DSD, Schematron, ...
- Regular expression types (XDuce, XQuery)

All of these are based on regular expressions

DTD

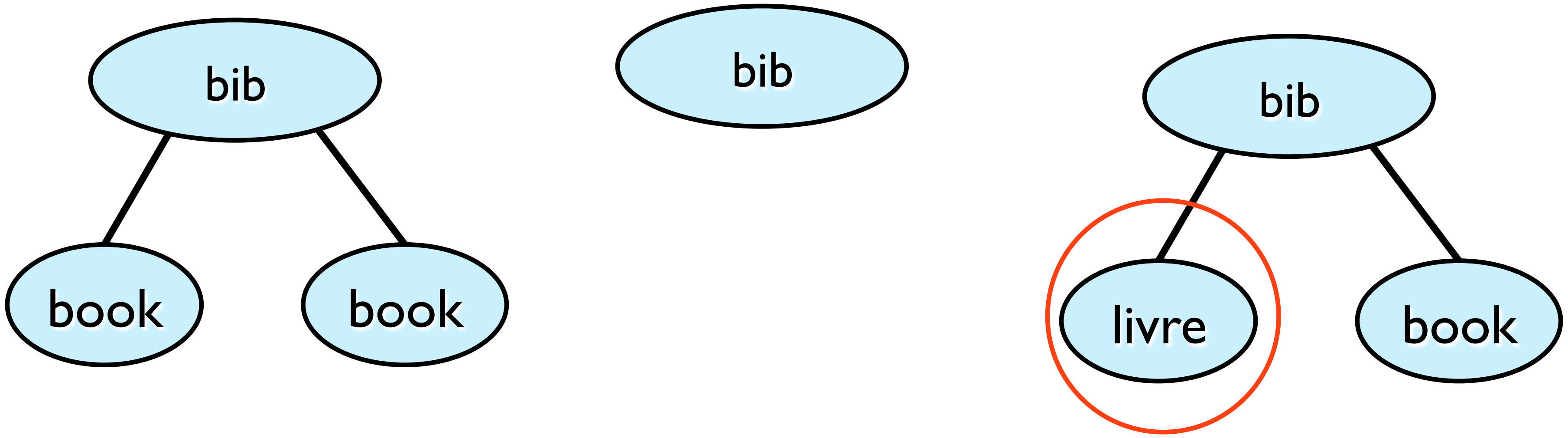
- As for XML, the main components of a DTD are the definitions of elements and attributes

Elements

Element declarations

```
<!ELEMENT bib (book*)>
```

- content usually a regular expression over element names
- also allowed: `ANY`, `EMPTY`, `PCDATA` (text)

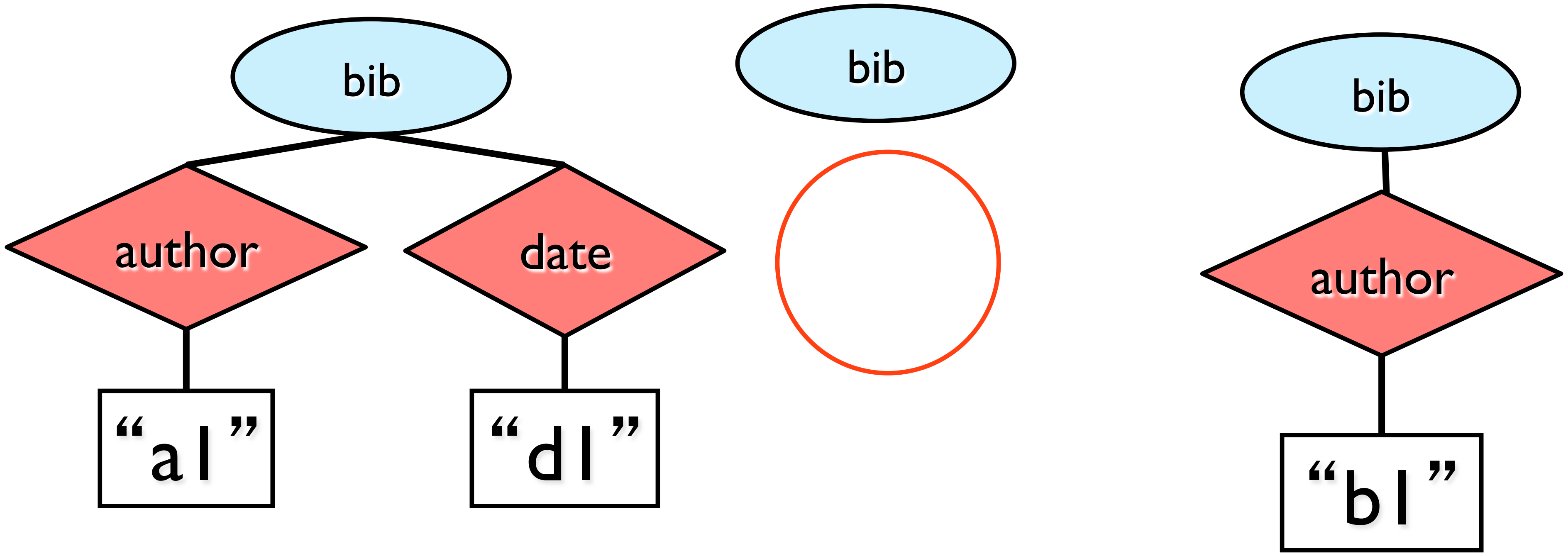


Attributes

Attribute declarations make reference to an element (e.g. bib)

```
<!ATTLIST bib author CDATA #REQUIRED>
<!ATTLIST bib date   CDATA #IMPLIED>
```

•Attributes can also be fixed (#FIXED), and have a specified default value



ID/IDRef

`ID`: Attribute value must be unique within document

- `<!ATTLIST person pid ID #REQUIRED>`

`IDREF`: Attribute must refer to an ID elsewhere

- `<!ATTLIST person pal IDREF #IMPLIED>`

```
<person pid = “011”   pal=“012”>
    <name> Barak Obama </name>
 </person>

<person pid = “012”   pal=“012”>
    <name> Bill Clinton </name>
</person>
```

Enumeration

Enumerations: one of a list

```
<!ATTLIST book type (comic|novel)>
```

DTD example: bibliography

```
<!DOCTYPE bib[
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
]>
```

Quiz : find the error(s)

```
<!ELEMENT root (row*)>

<!ATTLIST root title CDATA #REQUIRED>

<!ELEMENT row (A,(B|C))>

<!ATTLIST row (A|C) >

<!ELEMENT A (#PCDATA)>

<!ELEMENT B (#PCDATA)>

<!ELEMENT C (#PCDATA)>
```

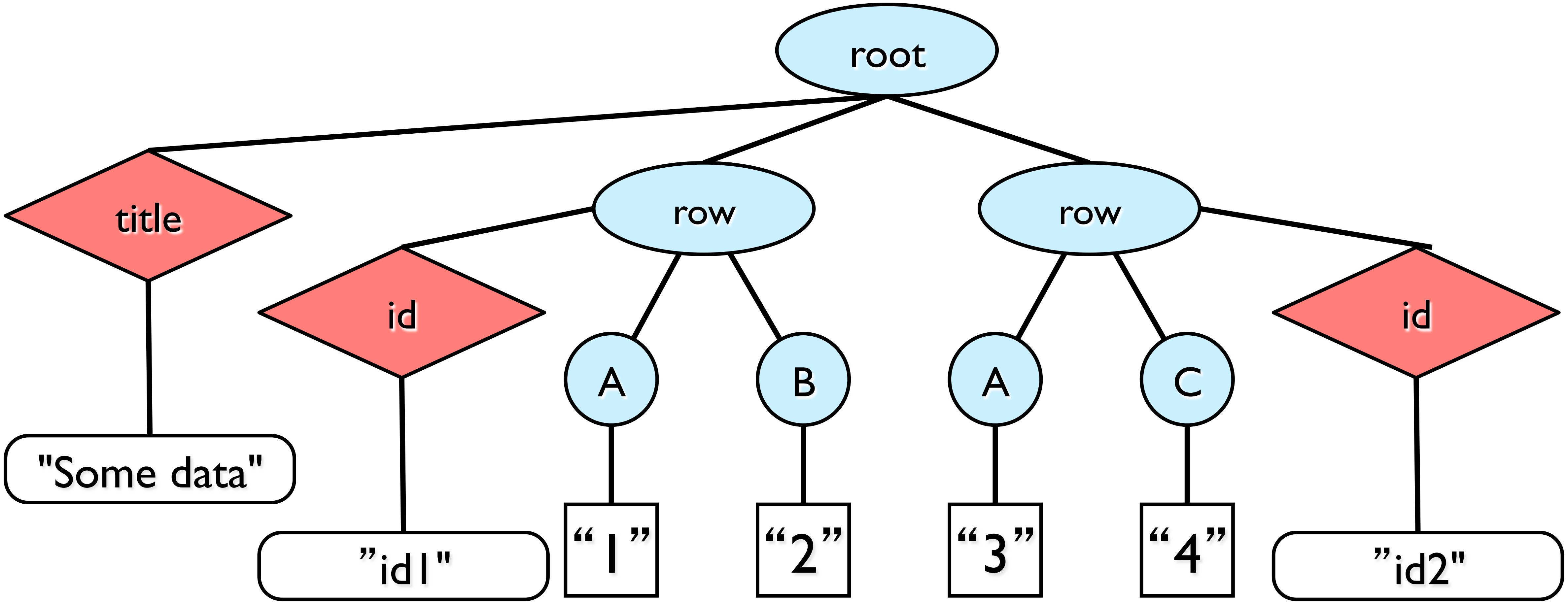
Quiz : find the error(s)

```
<!ELEMENT root (row*)>
<!ATTLIST root title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ELEMENT row (A|C) >
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

cannot define twice the same tag `row` !

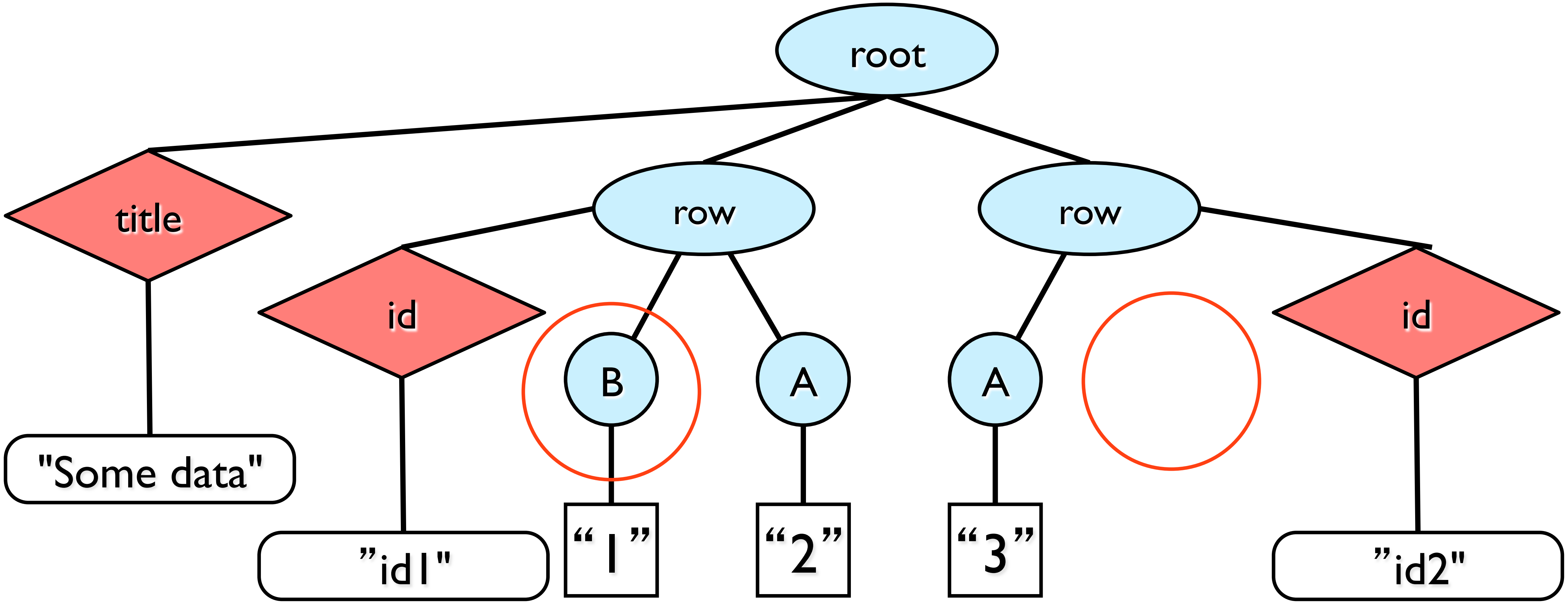
```
<!ELEMENT root (row*)>
<!ATTLIST root title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ATTLIST row id ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Valid



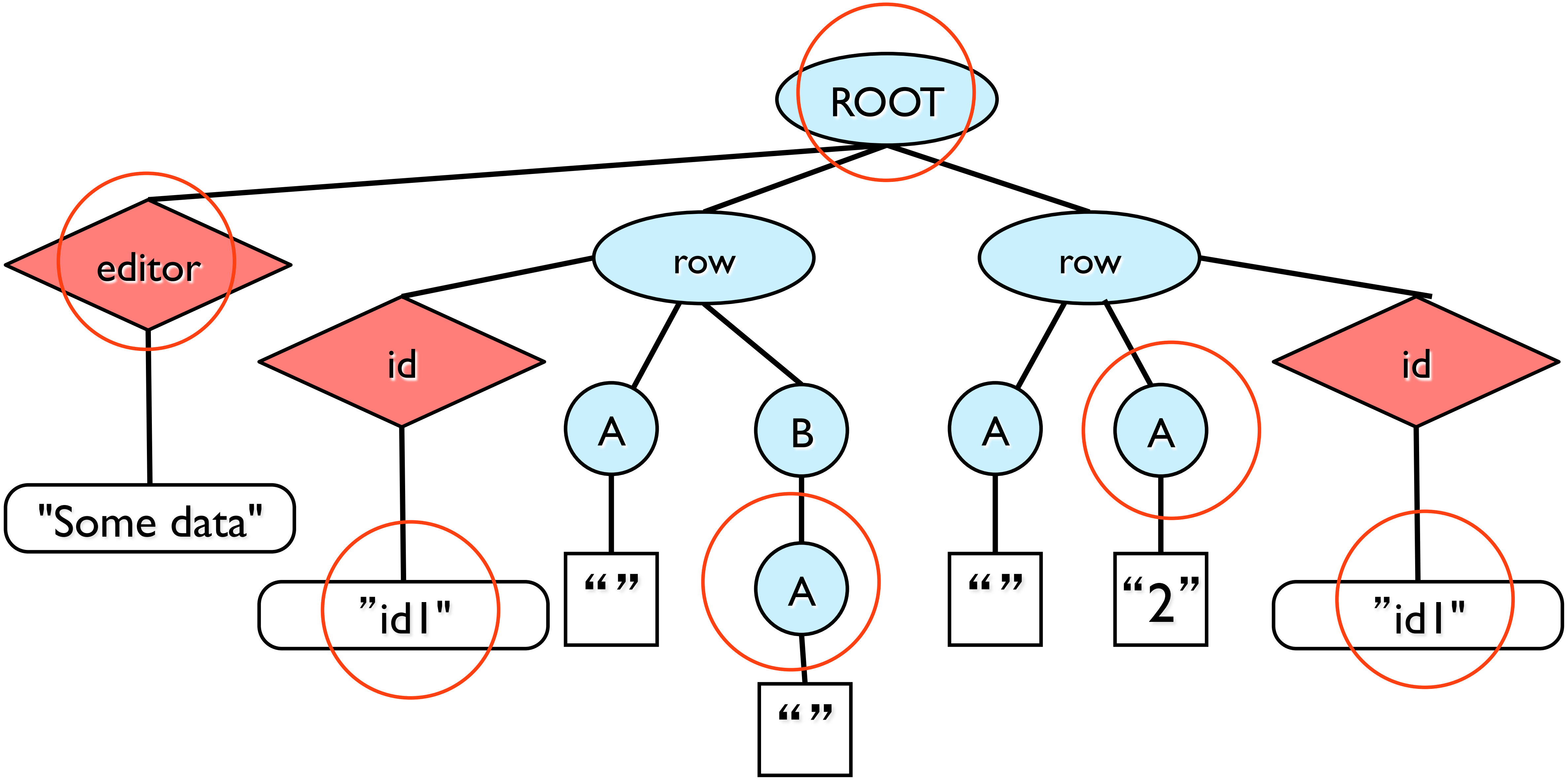
```
<!ELEMENT root (row*)>
<!ATTLIST root title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ATTLIST row id ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

Invalid



Quiz

```
<!ELEMENT root (row*)>
<!ATTLIST root title CDATA #REQUIRED>
<!ELEMENT row (A,(B|C))>
<!ATTLIST row id ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```



Recursive DTDs

DTD rules can be recursive

- `node → (node,node)?`

Recursion increases complexity of DTD

- This leads to documents of unbounded depth
- Some element types might not have any finite matching trees
- but this is easy to detect (look for unguarded cycles)
  - `silly → (silly, silly)`

Limitations of DTDs

Can't constrain text / attribute content (except in very limited ways)

Element, attribute content are context insensitive

- can't use same tag, e.g. "name", in different ways

Quiz

Give a document valid for this DTD, if it exists; otherwise explain why it does not exists.

```
<!ELEMENT X (Y)>
<!ELEMENT Y (A,B,X)>
<!ELEMENT A (#EMPTY)>
<!ELEMENT B (A,B)*>
```

Give a DTD for which only the following XML tree is valid
(=no other XML tree is valid!).

```
<A>
  <B/> <B/>  <B/>
<A/>
```

Give a document valid for this DTD, if it exists; otherwise explain why it does not exists.

```
<!ELEMENT Y (A)>
<!ELEMENT A (#EMPTY)>
<!ELEMENT A (A,B)*>
```

XML and DTD together

Coupled

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

<!DOCTYPE bib [
  <!ELEMENT bib book*>
  ...
]>

<bib> </bib>
```

Decoupled

```
<!DOCTYPE bib [
  <!ELEMENT bib book*>
  ...
]>
```

```
<?xml version="1.0">
<!DOCTYPE bib SYSTEM "bib.dtd">
<bib> </bib>
```

DTD and Regular Tree Grammars

(fun with regular expressions)

Plan

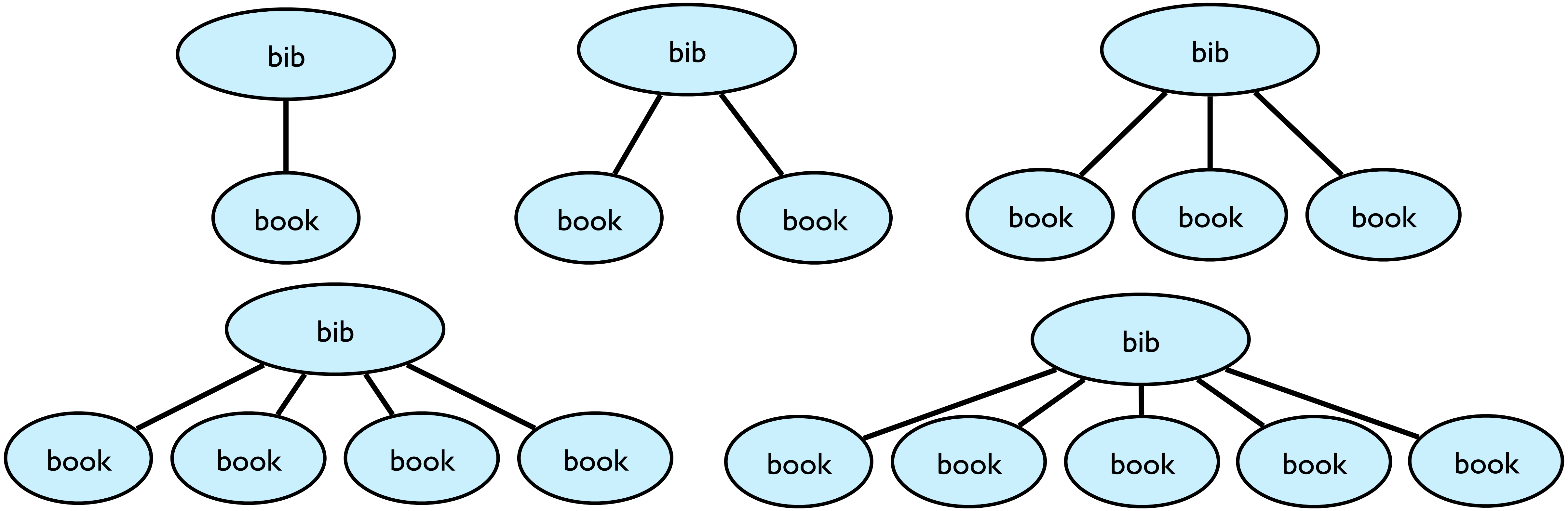
- Regular expressions
- Validation
- Determinism

Regular Trees

A DTD defines a possibly infinite set of **regular** XML trees

```
<!ELEMENT bib book+>

<!ELEMENT book EMPTY>
```



Regular Trees

DTD are a subclass of regular tree-grammars called “**local**”

- This is because any element has at most one definition

```
<!ELEMENT root child*>

<!ELEMENT child #PCDATA>

~~<!ELEMENT child EMPTY>~~
```

Regular tree grammars are equivalent to **nested** regular expressions

Regular expressions

```
r ::=  ϵ          empty sequence
    |   a         atomic symbol (in DTD, an element name)
    |  (r,s)      sequential composition
    |  (r|s)      union
    |  (r*)       repetition
```

```
r+ = r*|r          r? = r|ϵ
```

XML Validation

Problem : check if a sequence of children match regular expression

DTD Element

```
(title, (author+ | editor+ ), publisher, price )
```

XML

```
<title><author><author><publisher><price>     OK
<title><author><editor><publisher><price>     NO
```

Validation Algorithm

Traverse XML tree in document-order (=pre-order)

1. Check each element's children match regular expression

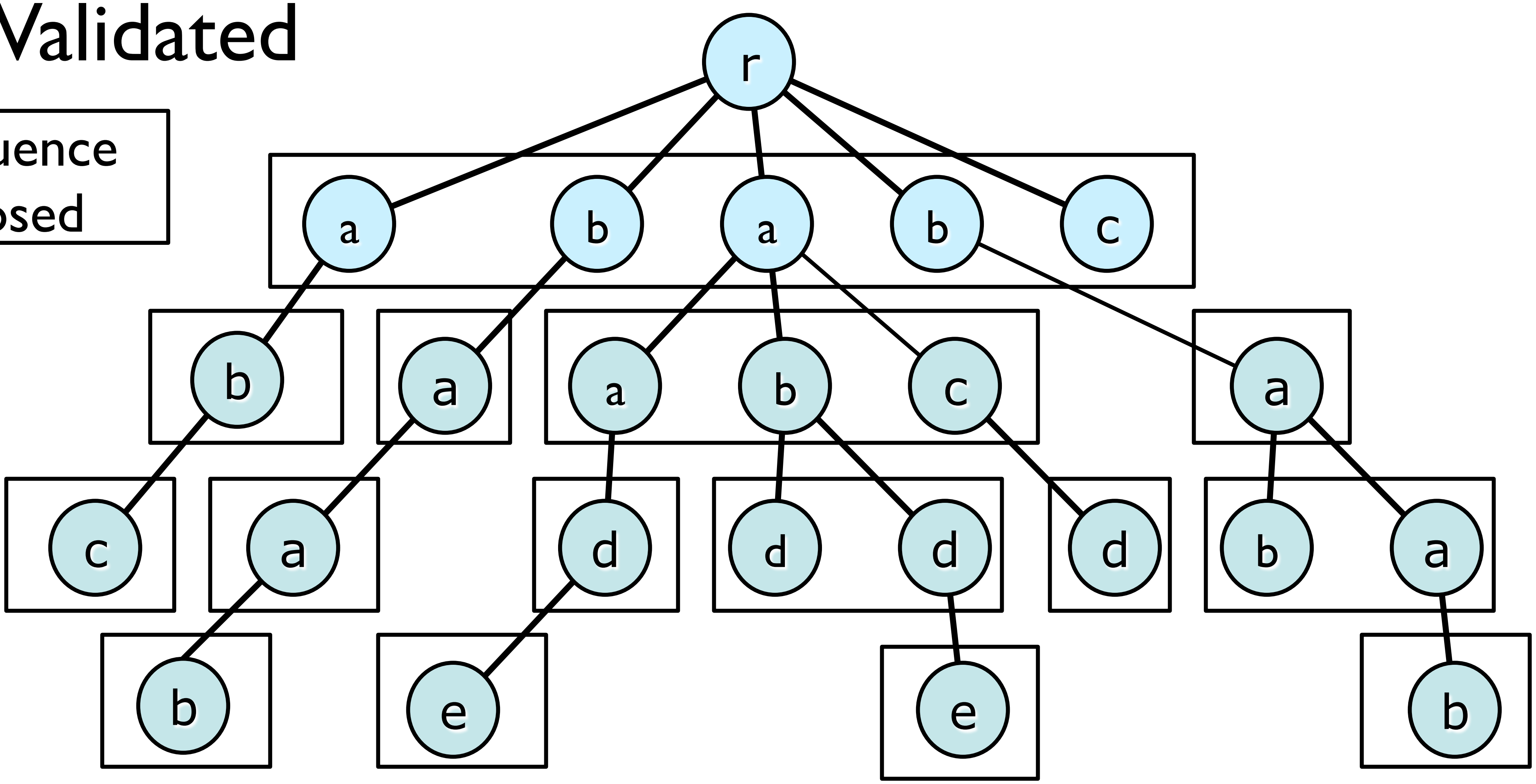
2. Check attribute types

3. Check ids are unique and idrefs refer to ids

XML Validation

● Validated

sequence closed



Needs a buffer (worst-case) proportional to the the document depth

W3C Restriction

Regular expressions in DTDs must be **deterministic**:

“there must be only one way to match any sequence of tags, no backtrack or look-ahead is required”

`( title, author* ) | ( title , editor* )` **NO**

can't decide if `<title>` matches first or second "`title`"

Better to write `title , ( author* | editor* )`

How to test Determinism?

Simplified algorithm of [Brueggemann-Klein & Wood ’98]

Ingredients : three auxiliary functions

FirstTag() LastTag() FollowsTag()

(1/3) FirstTag

*What can be the **first** tag of a sequence matching r ?*

r_1 = `(title,  (author+ | editor+ ), publisher, price )`

FirstTag(r_1) ? `title`

r_2 = `(author+ | editor+ )`

FirstTag(r_2) ? `author, editor`

(2/3) LastTag

*What can be the **last** tag of a sequence matching r ?*

r_1 = `(title,  (author+ | editor+ ), publisher, price )`

LastTag(r_1) ? `price`

r_2 = `(author+ | editor+ )`

LastTag(r_2) ? `author,  editor`

(3/3) FollowsTag

What tag can follow x in r ?

r_3 = `(title, (author+ | editor+ ), publisher, price )`

FollowsTag(r_3, `title`) ? `author, editor`

r_4 = `(author | editor )*`

FollowsTag(r_4, `author`) ? `author, editor`

Disambiguation

r_5 = `(author, title)? , author`

We resolve ambiguity by enumerating the tag occurrences

$r_5^{\#}$ = $\texttt{(author}_1\texttt{, title)? , author}_2$

FirstTag($r_5^{\#}$) = $\texttt{author}_1\texttt{, author}_2$

LastTag($r_5^{\#}$) = $\texttt{author}_2$

FollowsTag($r_5^{\#}$, `title`) = $\texttt{author}_2$

Determinism : Algorithm

```
1) Enumerate all the occurrences of a tag in r

2) Build a graph were

 - there is a node x for each tag in ($r^{\#}$), plus a root $x_0$

 - there is a directed edge ($x_0$,y) if y belongs to FirstTag($r^{\#}$)

 - there is a directed edge (x,y) if y belongs to FollowsTag($r^{\#}$,x)


3) return **false** if there exists edges (x,$y_i$) and (x,$y_j$) with $i \neq j$


4) return **true** otherwise
```

Testing Determinism

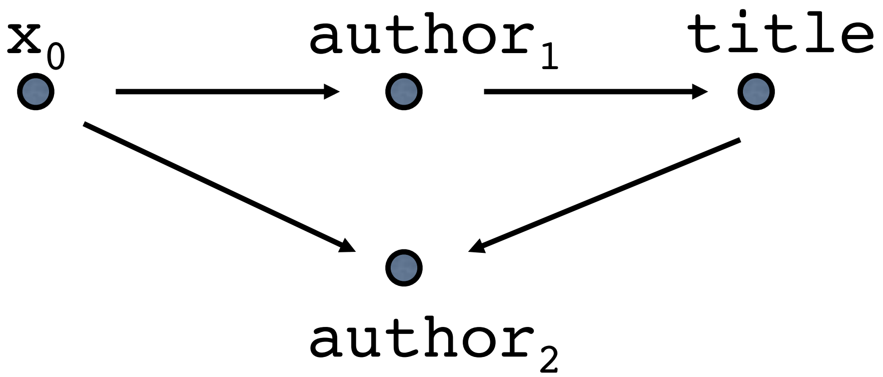
r_5 = `(author, title)? , author`

$r_5^{\#}$ = (author$_1$, title)? , author$_2$

FirstTag($r_5^{\#}$) = author$_1$, author$_2$

FollowsTag($r_5^{\#}$, author$_1$) = title

FollowsTag($r_5^{\#}$, title) = author$_2$

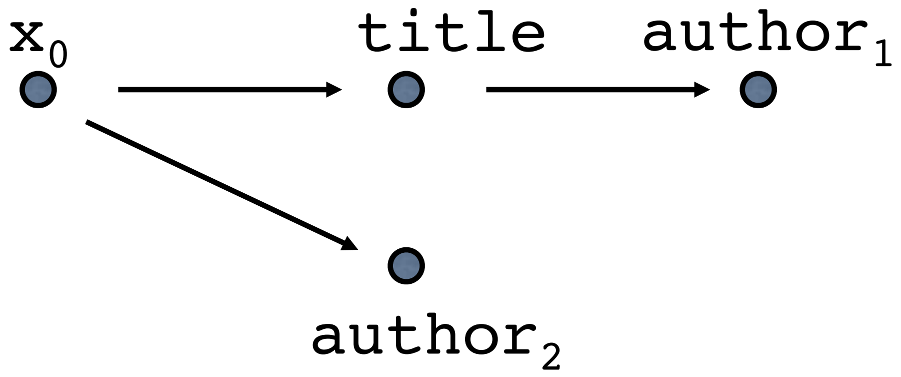


r_5 not deterministic

Testing Determinism

r_6 = `(title, author) | author`

$r_6^{\#}$ = `(title, author`$_1$`) | author`$_2$



FirstTag($r_6^{\#}$) = `title, author`$_2$

FollowsTag($r_6^{\#}$,title) = `author`$_1$

r_6 deterministic

Determinism - Quiz

Are the following regular expressions deterministic ?

- `(e|cb)*(c|bed)*`
- `(a,(a|c))|(b,(a|c))`

Why did we define **LastTag**(r) afterall ?

- It is hidden behind the definition of FollowsTag(r,x)

Exercise (pro): define (formally) the 3 auxiliary functions
(idea: by induction on the structure of a regular expression)

Back to XML Validation

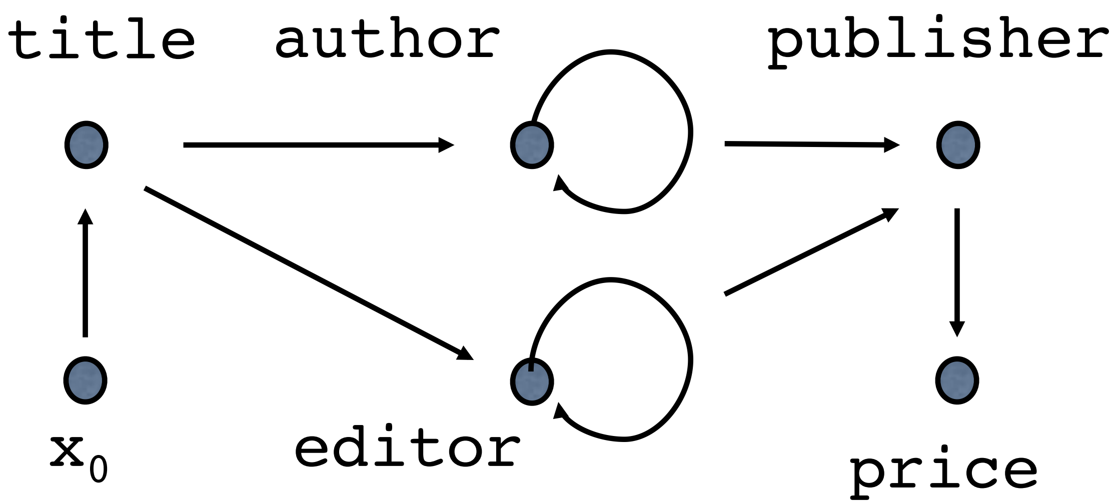
How to check if a sequence of children matches a regular expression ?

Comes for free once we computed the graph of r

Simply check if the sequence of children corresponds to a sequence of nodes of a path in the graph ending in a **LastTag**() element

Sequence Validation

r = (title, (author+ | editor+), publisher, price)



<title><author><author><publisher><price>   **OK**

<title><author><editor><publisher><price>   **NO**

Research Highlights

Checking Determinism

- Quadratic algorithm based on Glushkov automata [Brueggemann-Klein & Wood, ‘98]
- (best) Linear algorithm [Groz, Staworko, Maneth ‘11]

Checking Validity

- (best) Sublinear space algorithm [Konrad, Magniez ’11]

TD/TP
(à rendre le 27/09 ; Moodle)

- Écrire un documents XML
  - Bien formé / Valide par rapport à une DTD
- Écrire une DTD
  - Correcte / Qui permet de valider un document XML
- Vérifier la condition sur le determinisme