PRINCIPLES OF DATA ORGANISATION

Indexing hierarchical data

MOTIVATION

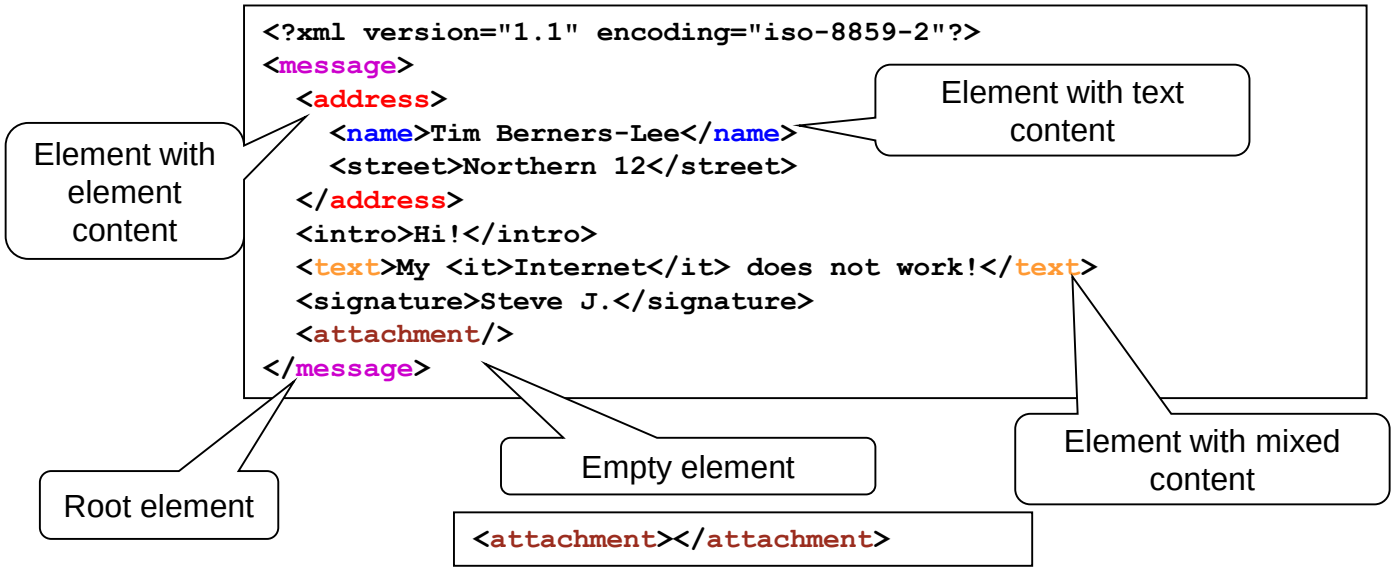
- Pure data are hard to process automatically
- We need to:
  - Ensure that a particular software understands the data
  - Add meaning (semantics) of particular data fragments
- E.g. HTML – describes the visualisation of data for an HTML browser
  - Problem 1: What if we are not interested just in visualisation?
  - Problem 2: HTML has lax rules for the structure
    - Complex processing
- Solution: semi-structured data formats
  - JSON, XML, …

BRIEFLY ON XML

XML (EXTENSIBLE MARKUP LANGUAGE)

- XML (eXtensible Markup Language) is a format for transfer and exchange of general data
  - Extensible Markup Language (XML) 1.0 (Fifth Edition)
    - http://www.w3.org/TR/xml/
  - Extensible Markup Language (XML) 1.1 (Second Edition)
    - http://www.w3.org/TR/xml11/
- XML is a subset (application) of SGML (Standard Generalized Markup Language - ISO 8879) – from 1986
- XML does not deal with data presentation
  - It enables to tag parts of the data
  - The meaning of the tags depends on the author
    - Presentation is one possible example

XML ELEMENTS



XML ATTRIBUTES

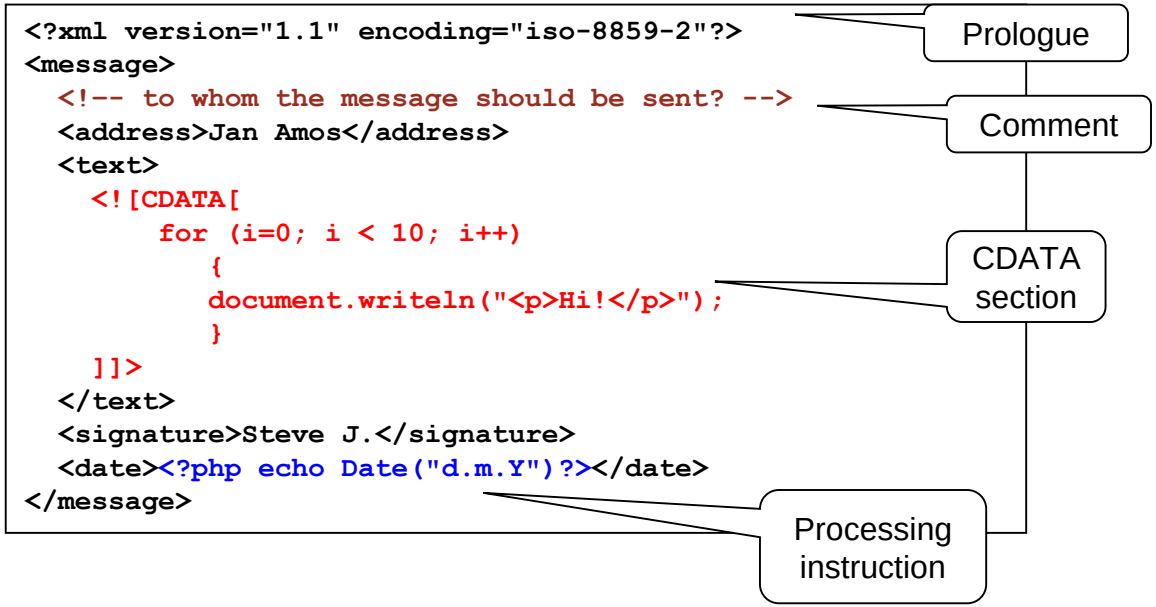
```
<?xml version="1.1" encoding="iso-8859-2"?>
<message>
  <address>
    <name>Tim Berners-Lee</name>
    <street>Northern 12</street>
  </address>
  <intro>Hi!</intro>
  <text>My <it>Internet</it> does not work!</text>
  <signature>Steve J.</signature>
  <attachment fig="image01.jpg"/>
</message>
```

Element with an attribute

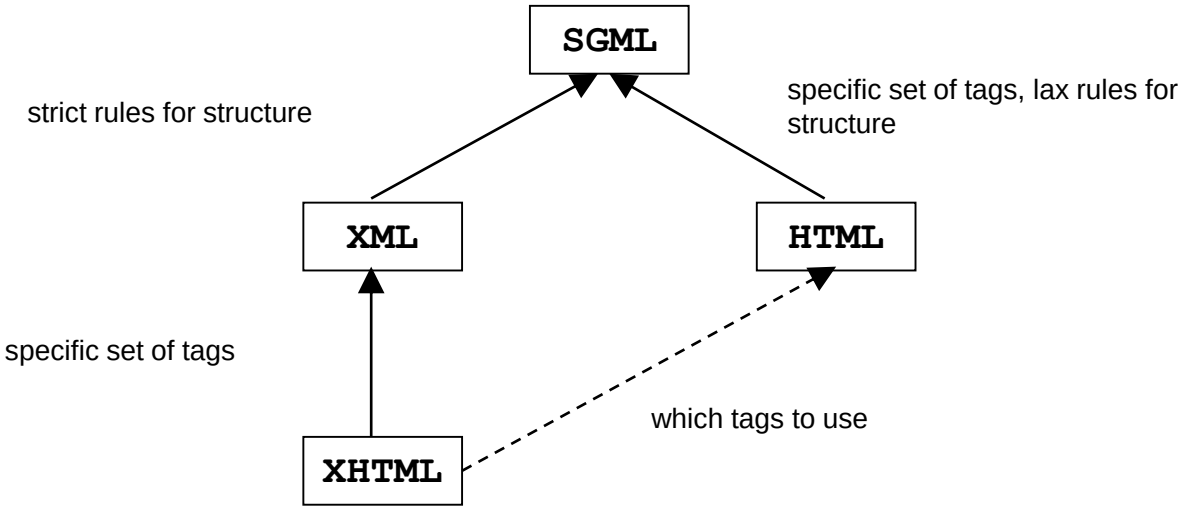
Attribute name

Attribute value

OTHER ITEMS OF XML DOCUMENT



SGML VS. XML VS. HTML VS. XHTML



XML DOCUMENT

- XML document is well-formed, if:
  - It has introductory prolog
  - Start and end tags nest properly
    - Each element has a start and an end tag
    - Corresponding tags have the same name (case sensitivity)

      `<a></A>`
    - Pairs of tags do not cross

      `<a><b></a></b>`
    - The whole document is enclosed in a single root element

DTD

- Problem: Well-formedness is insufficient
  - We need to restrict the set of tags and their content
- Document Type Definition (DTD) describes the structure (grammar) of an XML document
  - Using regular expressions
- Valid XML document = well-formed XML document corresponding to a given grammar
  - There are also other languages – XML Schema, Schematron, RELAX NG, …

DTD – EXAMPLE

,	··· sequence
|	··· selection
?	··· iteration (0 or 1)
+	··· iteration (1 or more)
*	··· iteration (0 or more)

Element =
name
+
content model

Attribute =
name
+
data type
+
presence /
default value

```
<!ELEMENT employees (person)+>
<!ELEMENT person (name, email*, relations?)>
    <!ATTLIST person id ID #REQUIRED>
    <!ATTLIST person note CDATA #IMPLIED>
    <!ATTLIST person holiday (yes|no) "no">
<!ELEMENT name ((first, surname)|(surname, first))>
<!ELEMENT first (#PCDATA)>
<!ELEMENT surname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relations EMPTY>
    <!ATTLIST relations superior IDREF #IMPLIED>
    <!ATTLIST relations subordinates IDREFS #IMPLIED>
```

BRIEFLY ON XPATH

XPATH DATA MODEL

```
<?xml version="1.0"?>
<!DOCTYPE order SYSTEM "order.dtd">
<order date="10/10/2008" status="confirmed">
 <customer number="C992">Steve J.</customer>
 <items>
  <item code="48282811">
   <amount>5</amount>
   <price>22</price>
  </item>
  <item code="929118813">
   <amount>1</amount>
   <price>91934</price>
   <color>blue</color>
  </item>
 </items>
</order>
```

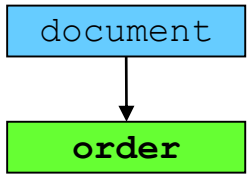
XPATH DATA MODEL

```
<?xml version="1.0"?>
<!DOCTYPE order SYSTEM "order.dtd">
<order date="10/10/2008" status="confirmed">
 <customer number="C992">Steve J.</customer>
 <items>
  <item code="48282811">
   <amount>5</amount>
   <price>22</price>
  </item>
  <item code="929118813">
   <amount>1</amount>
   <price>91934</price>
   <color>blue</color>
  </item>
 </items>
</order>
```

document

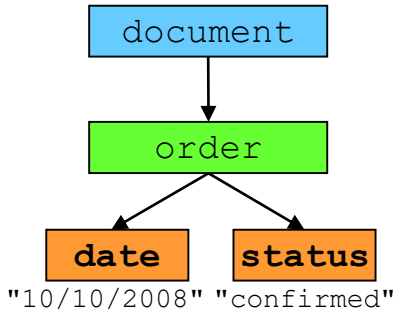
XPATH DATA MODEL

```
<?xml version="1.0"?>
<!DOCTYPE order SYSTEM "order.dtd">
<order date="10/10/2008" status="confirmed">
 <customer number="C992">Steve J.</customer>
 <items>
  <item code="48282811">
   <amount>5</amount>
   <price>22</price>
  </item>
  <item code="929118813">
   <amount>1</amount>
   <price>91934</price>
   <color>blue</color>
  </item>
 </items>
</order>
```



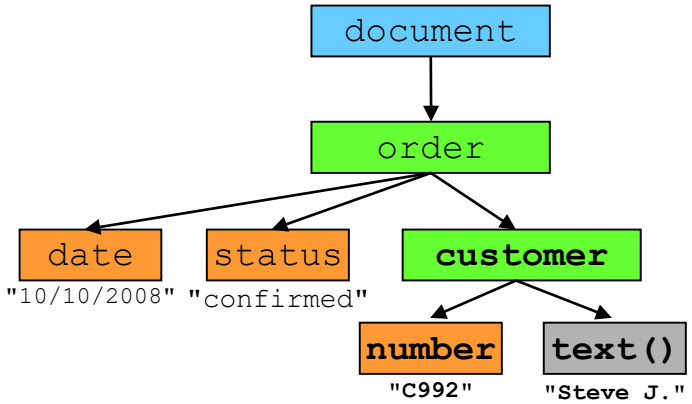
XPATH DATA MODEL

```
<?xml version="1.0"?>
<!DOCTYPE order SYSTEM "order.dtd">
<order date="10/10/2008" status="confirmed">
 <customer number="C992">Steve J.</customer>
 <items>
  <item code="48282811">
   <amount>5</amount>
   <price>22</price>
  </item>
  <item code="929118813">
   <amount>1</amount>
   <price>91934</price>
   <color>blue</color>
  </item>
 </items>
</order>
```

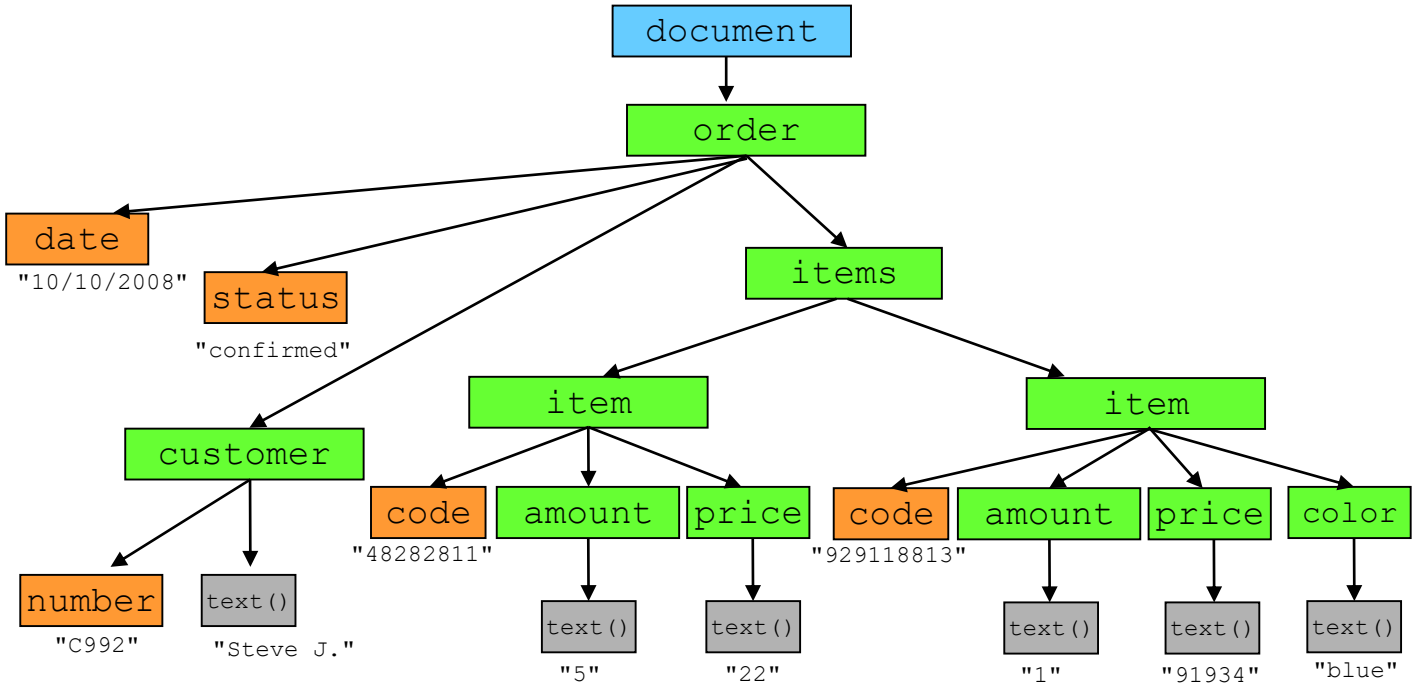


XPATH DATA MODEL

```
<?xml version="1.0"?>
<!DOCTYPE order SYSTEM "order.dtd">
<order date="10/10/2008" status="confirmed">
 <customer number="C992">Steve J.</customer>
 <items>
  <item code="48282811">
   <amount>5</amount>
   <price>22</price>
  </item>
  <item code="929118813">
   <amount>1</amount>
   <price>91934</price>
   <color>blue</color>
  </item>
 </items>
</order>
```



XPATH DATA MODEL



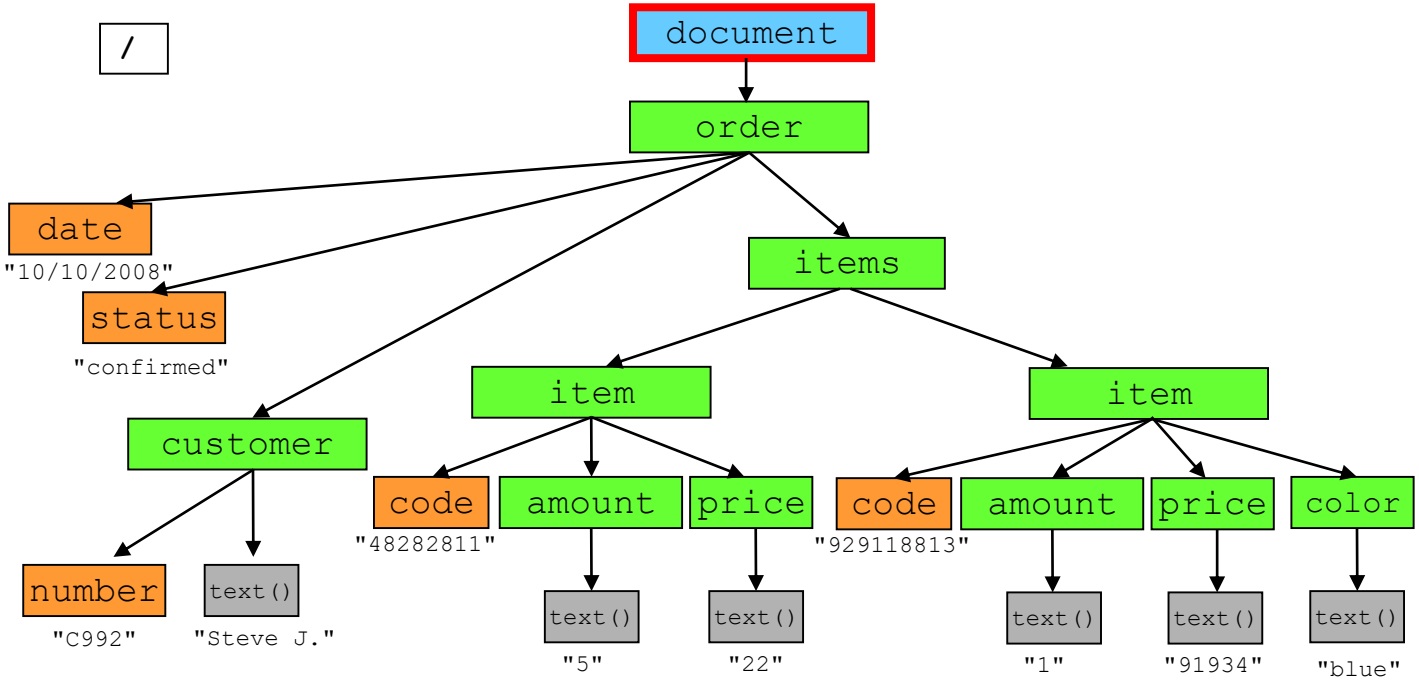
XPATH DATA MODEL

- Types of nodes in the model
  - Root node
  - Element node
  - Text node
  - Attribute node
  - Comment
  - Processing instruction
  - Namespace
- What is not included: CDATA section, entity reference, DTD

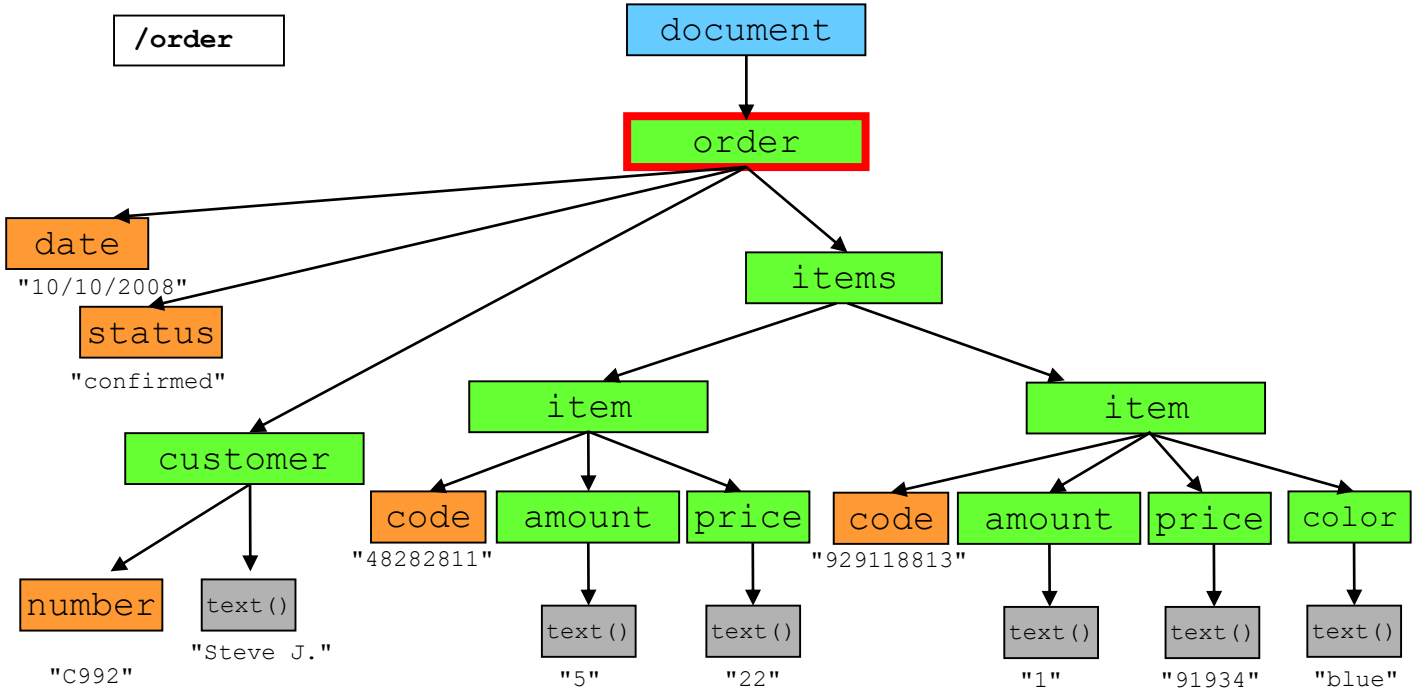
XPATH EXPRESSION

- XPath expression is a path
- Path consists of steps
  - Absolute path:
  - `/Step1/Step2/…/StepN`
  - Relative path:
  - `Step1/Step2/…/StepN`

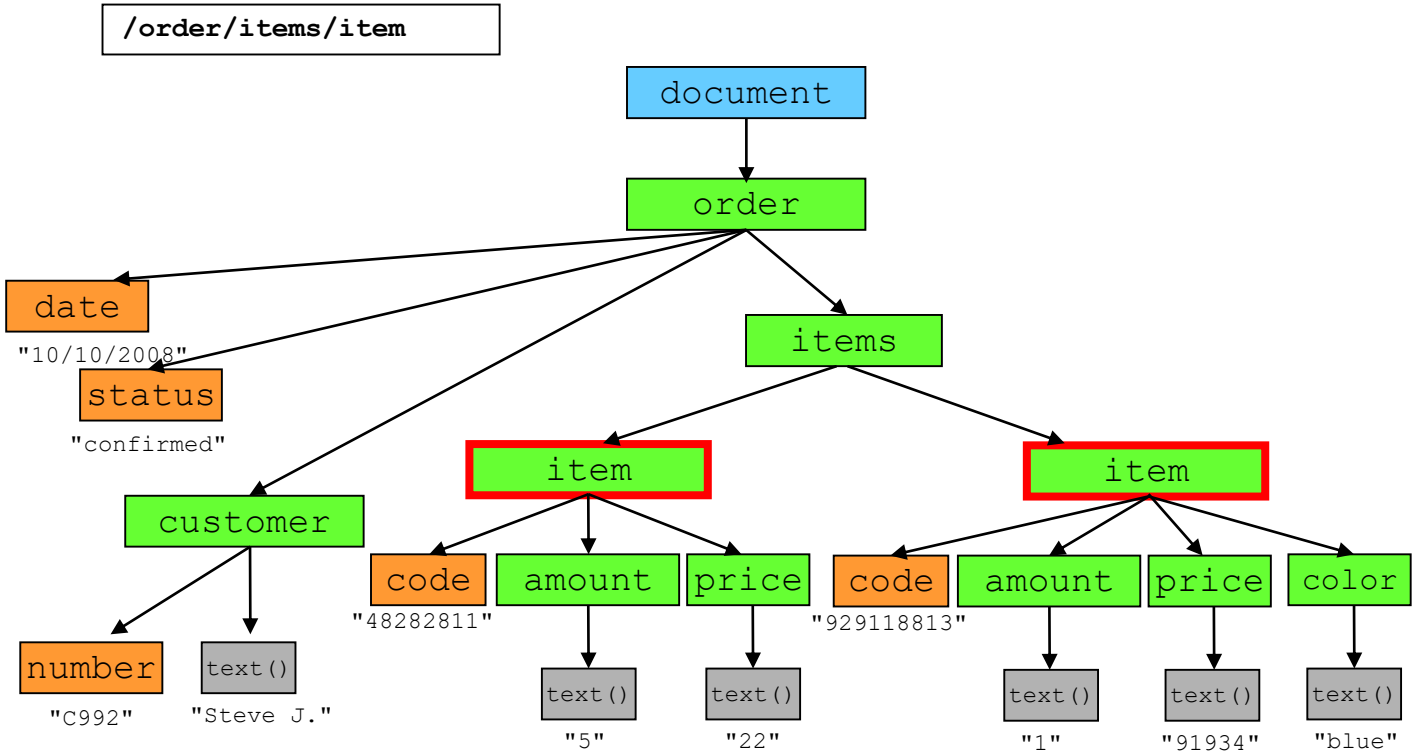
XPATH EXPRESSIONS – EXAMPLES



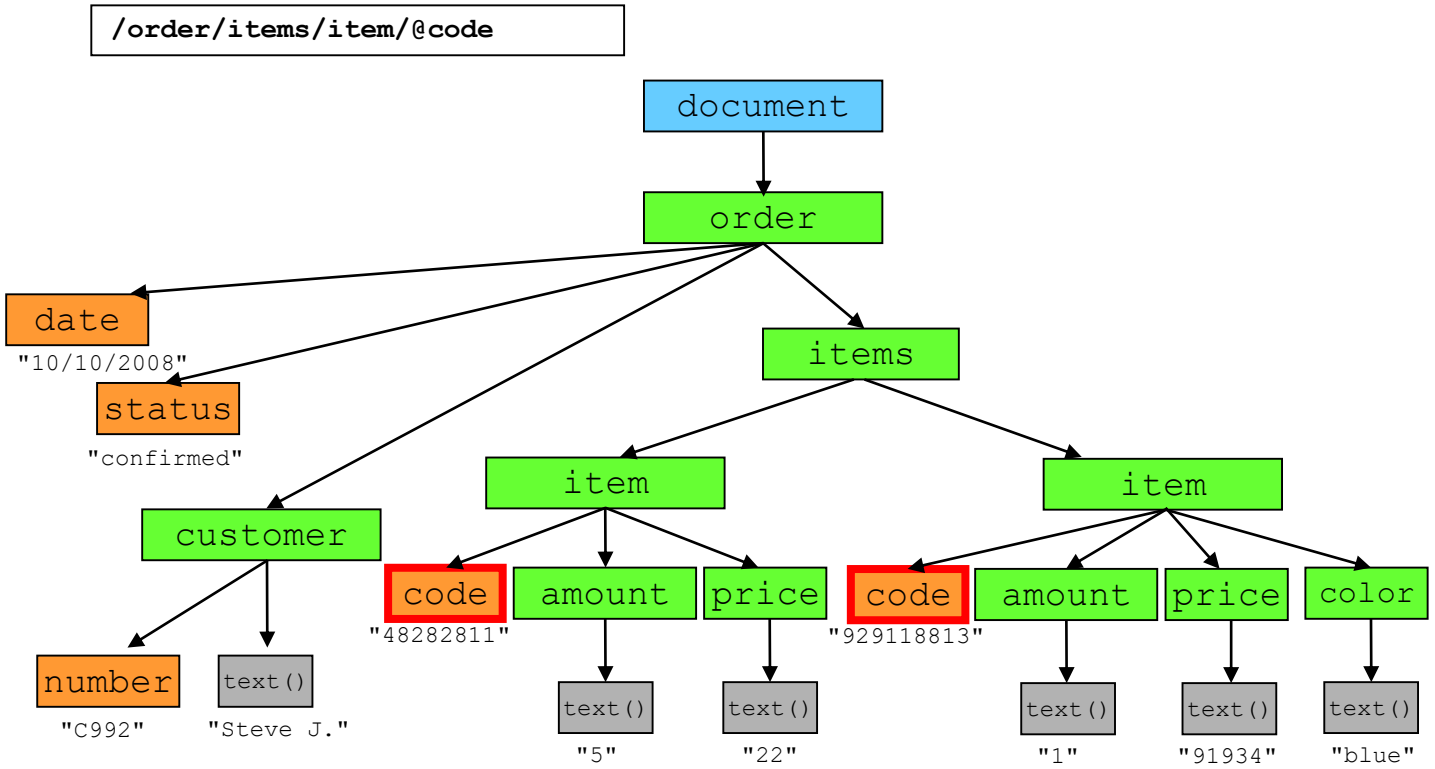
XPATH EXPRESSIONS – EXAMPLES



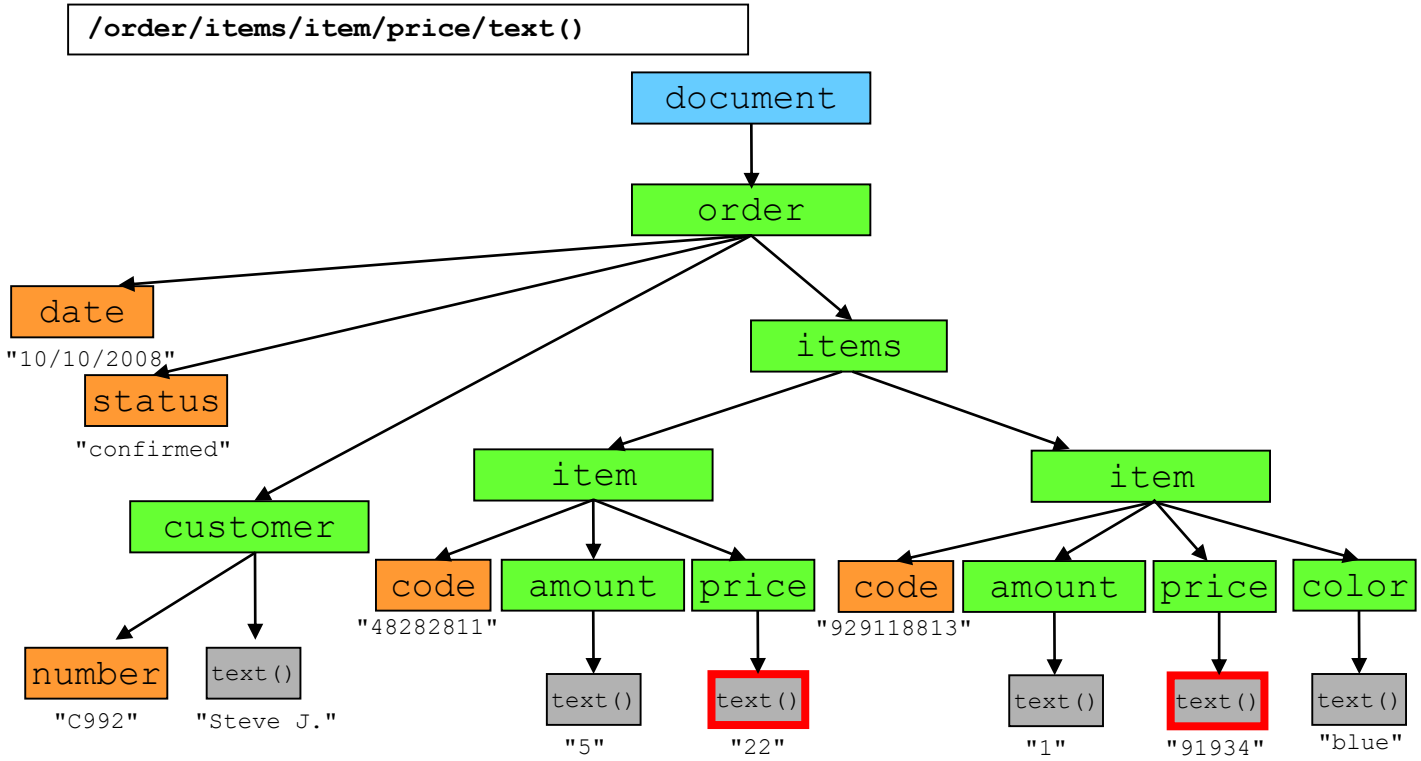
XPATH EXPRESSIONS – EXAMPLES



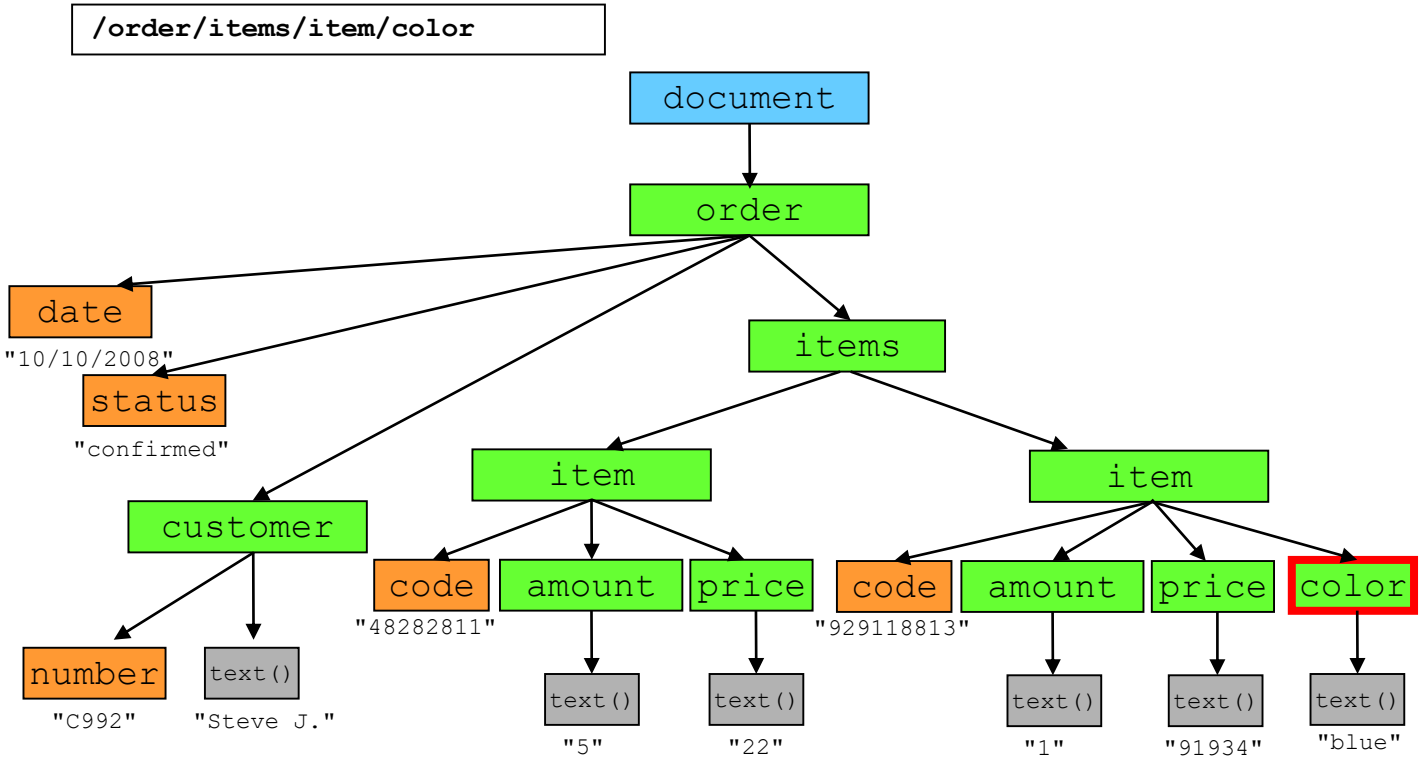
XPATH EXPRESSIONS – EXAMPLES



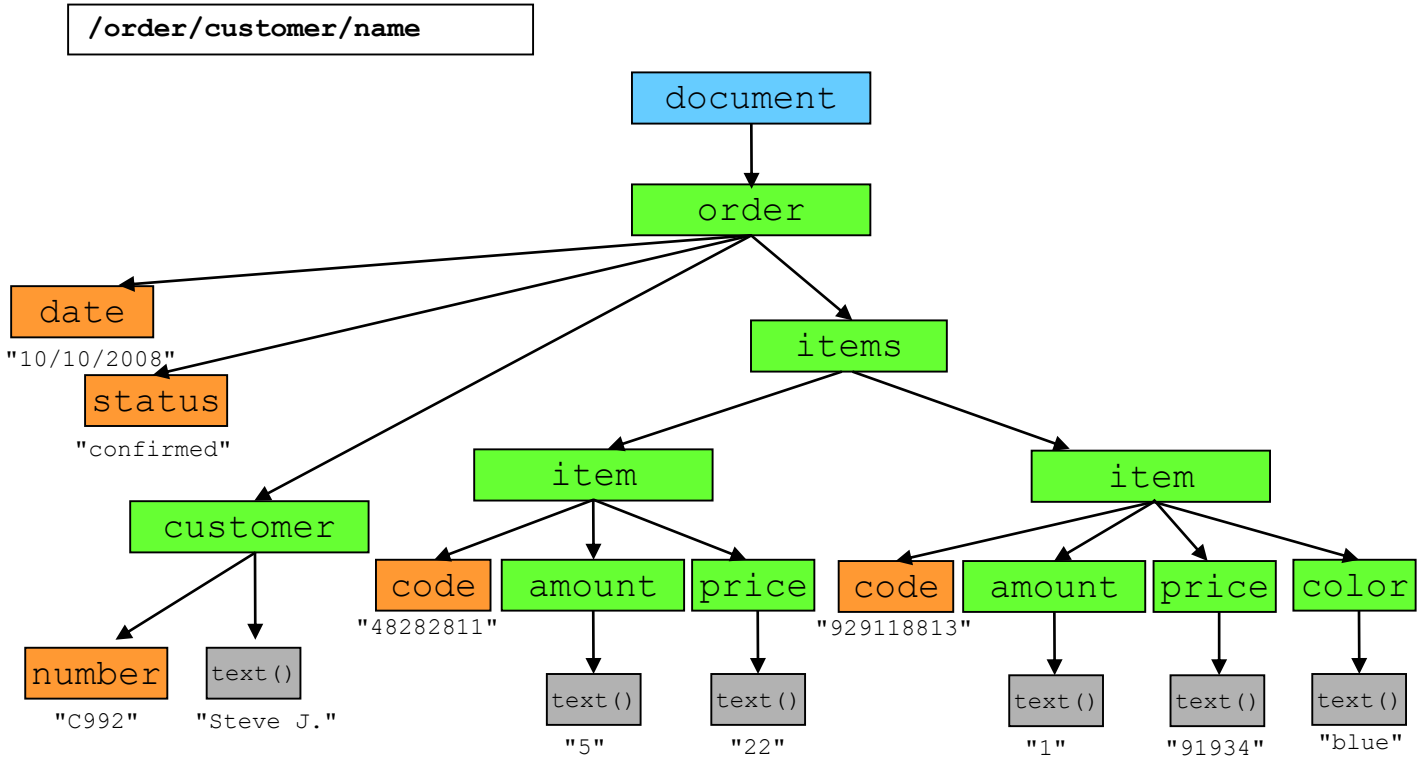
XPATH EXPRESSIONS – EXAMPLES



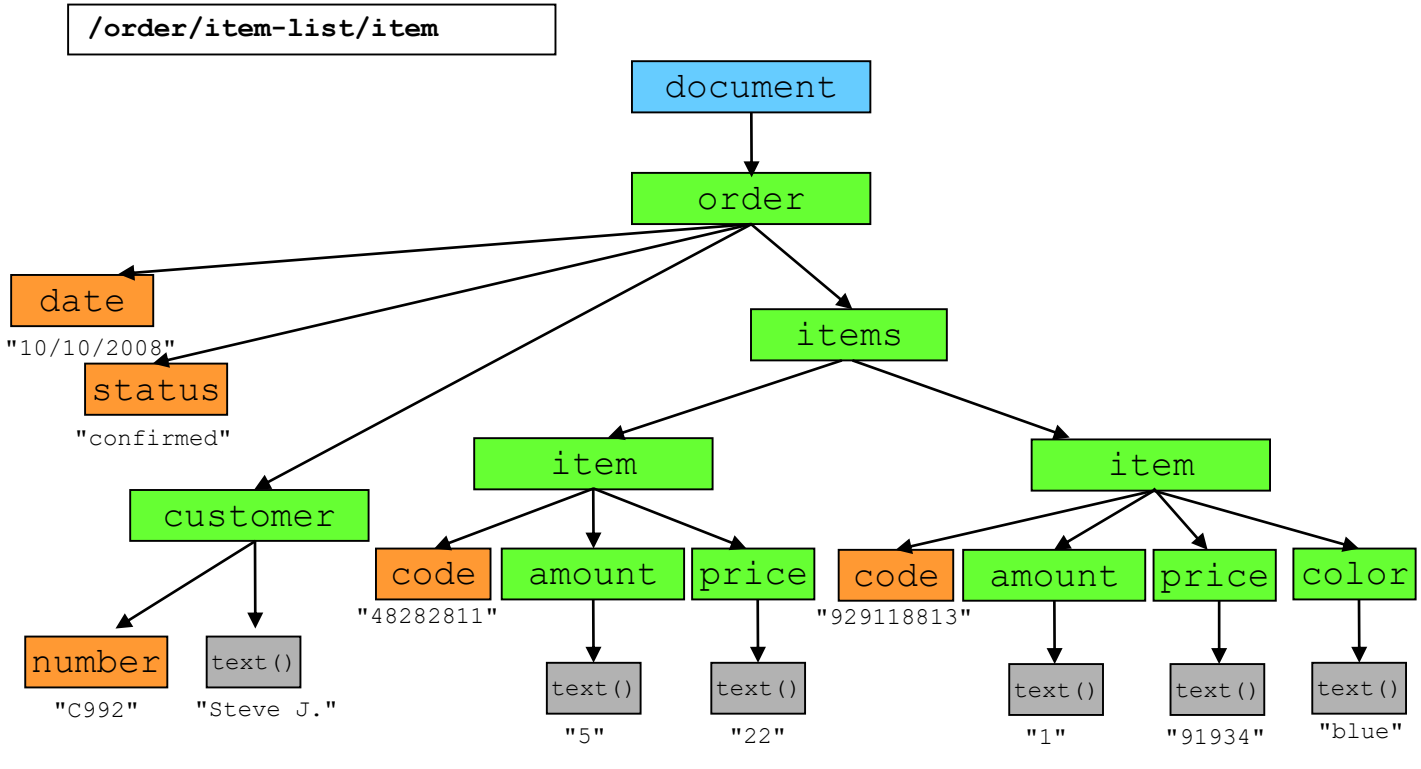
XPATH EXPRESSIONS – EXAMPLES



XPATH EXPRESSIONS – EXAMPLES



XPATH EXPRESSIONS – EXAMPLES



BRIEFLY ON JSON

JSON
(JAVASCRIPT OBJECT NOTATION)

- Text-based easy-to-read-and-write open standard for data interchange
  - Serializing and transmitting structured data
  - Considered as an alternative to XML
- Filename: `*.json`
- Internet media type (MIME type): `application/json`
- Derived from JavaScript scripting language
- Language independent
  - But uses conventions of the C-family of languages (C, C++, C#, Java, JavaScript, Perl, Python, …)
- Originally specified by Douglas Crockford in 2001
  - RFC 4627
    - Requests for comments = "standard" publication of the Internet Engineering Task Force and the Internet Society

http://www.json.org/

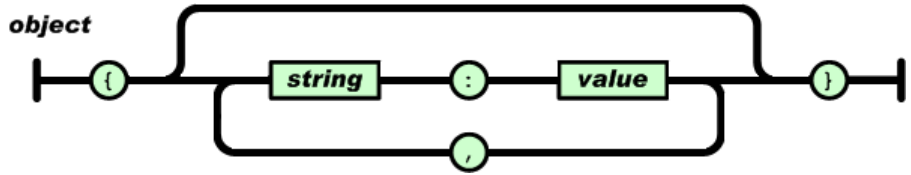
JSON
BASIC STRUCTURES

- Built on two general structures:
  - Collection of name/value pairs
    - Realized as an object, record, struct, dictionary, hash table, keyed list, associative array, …
  - Ordered list of values
    - Realized as an array, vector, list, sequence, …
- Universal data structures
  - All modern programming languages support them

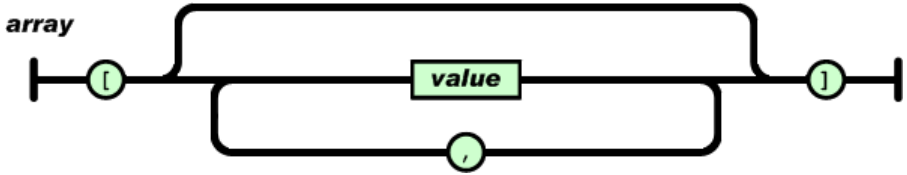
JSON

BASIC DATA TYPES

- object – an unordered set of name/value pairs
  - called properties (members) of an object
  - { comma-separated name : value pairs }



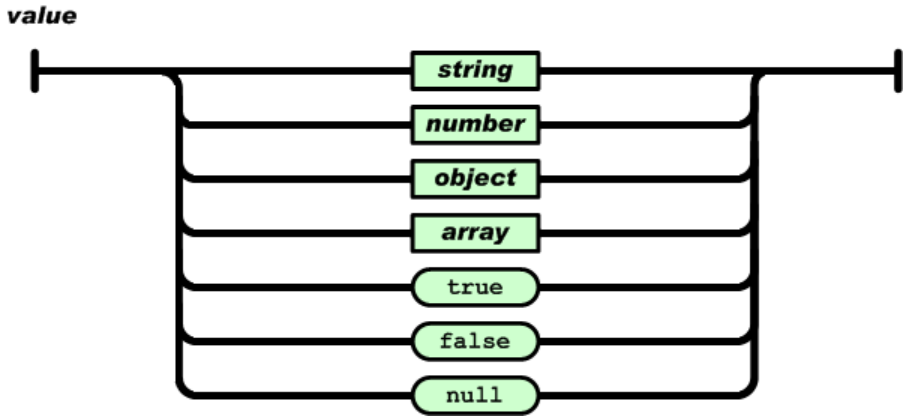
- array – an ordered collection of values
  - called items (elements) of an array
  - [comma-separated values]



JSON

BASIC DATA TYPES

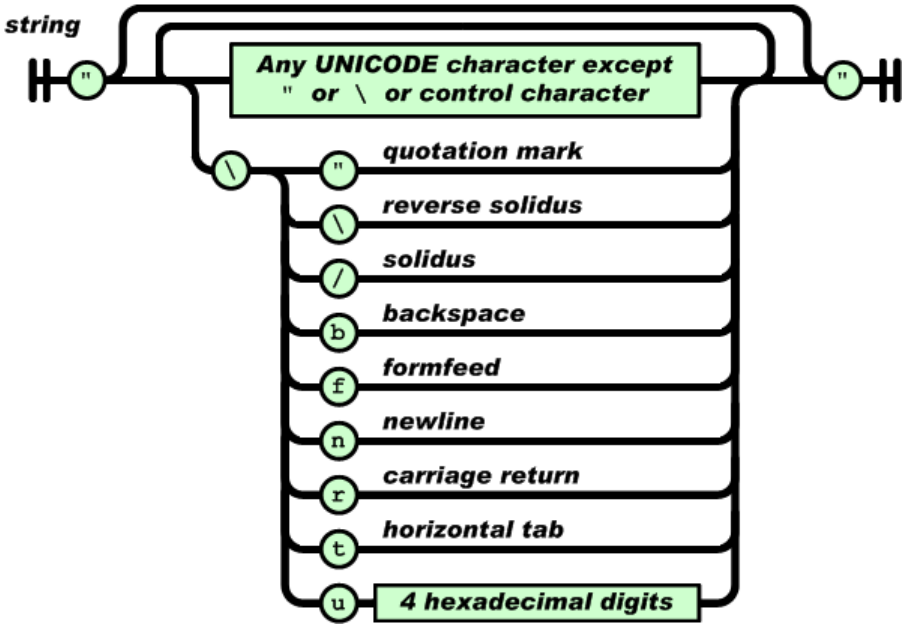
- value – string in double quotes / number / true or false (i.e., Boolean) / null / object / array
  - Can be nested



JSON

BASIC DATA TYPES

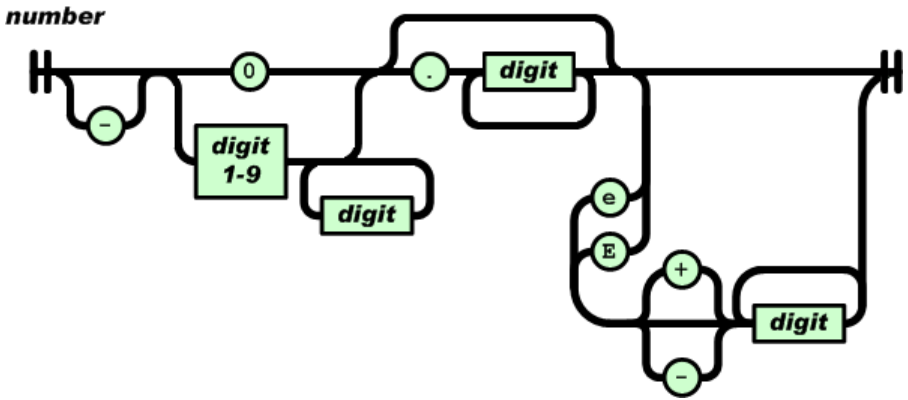
- string – sequence of zero or more Unicode characters, wrapped in double quotes
  - Backslash escaping



JSON

BASIC DATA TYPES

- number – like a C or Java number
  - Octal and hexadecimal formats are not used



JSON EXAMPLE

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

name/value pair

array

object

INDEXING OF SEMI-STRUCTURED (XML) DATA

CLASSIFICATION OF XML DOCUMENTS

- The basic classification of XML documents results from their origin and the way they were created
  - data-oriented
  - document-oriented
  - hybrid
- For the particular classes different ways of implementations are suitable

DOCUMENT-ORIENTED XML DOCUMENTS

- Usually created and processed by humans
- Irregular, less structured
  - Semi-structured data
- Often contain
  - Mixed-content elements
  - CDATA sections
  - Comments
  - Processing instructions
- The order of sibling elements is crucial
- Example: XHTML web pages

DOCUMENT-ORIENTED XML DOCUMENTS

```
<book id="12345">
  <title>All I Really Need To Know I Learned in
Kindergarten</title>
  <author>Robert Fulghum</author>
  <description>A new, edited and extended publication published
on the occasion of the fifteen anniversary of the first
edition</description>
  <Text>
    <p>Fifteen years after publishing of <q>his</q>
<i>Kindergarten</i> Robert Fulghum has decided to read it once
again, now in <i>2003</i>.</p>
    <p>He wanted to find out whether and, if so, to what extent
his opinions have changed and why. Finally, he modified and
extended his book to...</p>
  <Text>
</book>
```

DATA-ORIENTED XML DOCUMENTS

- Usually created and processed by machines
- Regular, deep structure
  - Fully structured data
- They do not contain
  - Mixed-content elements
  - CDATA sections
  - Comments
  - Processing instructions
- The order of sibling elements is often unimportant
- Example: database exports, catalogues, …

DATA-ORIENTED XML DOCUMENTS

```
<book id="12345">
  <title>All I Really Need To Know I Learned in
Kindergarten</title>
  <author>
    <name>Robert</name>
    <surname>Fulghum</surname>
  </author>
  <edition title="Argo">
    <year>2003</year>
    <ISBN>80-7203-538-X</ISBN>
  </edition>
  <edition title="Argo">
    <year>1996</year>
    <ISBN>80-7203-028-0</ISBN>
  </edition>
</book>
```

IMPLEMENTATION APPROACHES

- Differ according to the type of documents
  - Exploit typical features
  - Problem: hybrid documents
    - Ambiguous classification
- Document-oriented techniques vs. Data-oriented techniques

DOCUMENT-ORIENTED TECHNIQUES

- We need to preserve the document as whole
  - Order of sibling elements
  - Comments, CDATA sections, …
  - Even whitespaces
    - For legislative documents
- Round tripping – storing a document into a database and its retrieval
  - The level of round tripping says to what extent the documents are similar
    - The higher level, the higher similarity
  - In the optimal case they are equivalent

DATA-ORIENTED TECHNIQUES

- Idea: The data are stored in a relational database management system (RDBMS)
  - Mapping method – transforms the data into relations (and back)
  - XML queries over XML data → SQL queries over relations
  - The result of SQL query → XML document
- Exploit data-oriented aspects (low level of round tripping)
  - It is not necessary to preserve the document as a whole
    - Order of sibling elements is ignored, document-oriented constructs (comments, whitespaces, …) are ignored, …
  - No (little) support for mixed-content elements

NUMBERING SCHEMAS

A numbering schema of a tree model of a document is a function which assigns each node a unique identifier that serves as a reference to that node for indexing and query evaluation

- Enable fast evaluation of selected relationships among nodes of XML document
  - Ancestor-descendant
  - Parent-child
  - Element-attribute
  - …
  - Depth of the node
  - Order among siblings
  - …

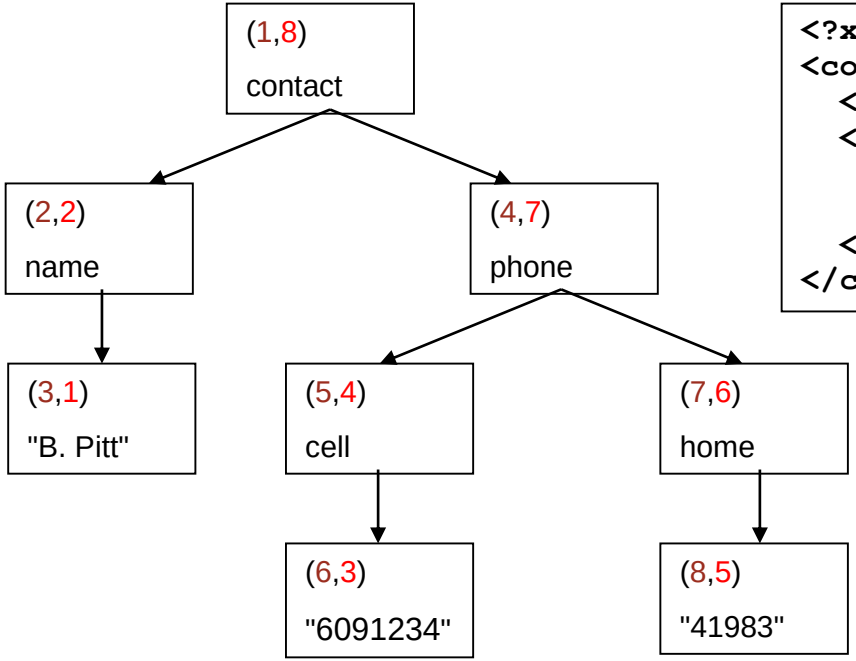
NUMBERING SCHEMAS

- Sequential numbering schema
  - The identifiers are assigned to the nodes as soon as they are added to the system sequentially, starting from 1
- Structural numbering schema
  - Enables to preserve and evaluate a selected relationship among any two nodes of the document
  - Often it is expected to enable fast searching for all occurrences of such a relationship in the document

NUMBERING SCHEMAS

- Stable numbering schema
  - A schema which does not have to be modified (except for preserving its local features) when the structure of the respective data changes
    - i.e., on insertion/deletion of nodes
- A schema of a structural numbering schema
  - Is an ordered pair (p, L), where p is a binary predicate and L is an inverse function which for the given XML tree model $T = (N, E)$ assigns each node $v \in N$ a binary sequence $L(v)$.
  - For each pair of nodes $u, v \in N$ predicate $p(L(u), L(v))$ is satisfied if v is in a particular relationship with u.
    - e.g. v is a descendant of u
  - Particular numbering schema: particular p and L

DIETZ NUMBERING

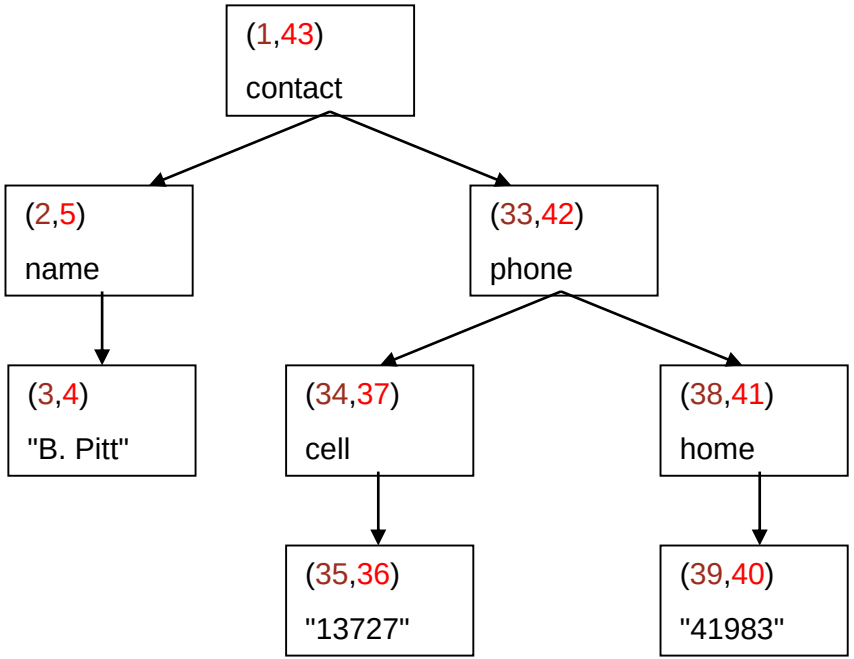



```
<?xml version="1.0"?>
<contact>
  <name>B. Pitt</name>
  <phone>
    <cell>6091234</cell>
    <home>41983</home>
  </phone>
</contact>
```

DIETZ NUMBERING

- Preorder traversal
  - Child nodes of a node follow their parent node
- Postorder traversal
  - Parent node follows its child nodes
- Construction of a numbering schema
  - Each node $v \in N$ is assigned with a pair (x, y) denoting preorder and postorder order
  - Node $v \in N$ having $L(v) = (x, y)$ is a descendant node of node u having $L(u) = (x', y')$ if $x' < x$ & $y' > y$

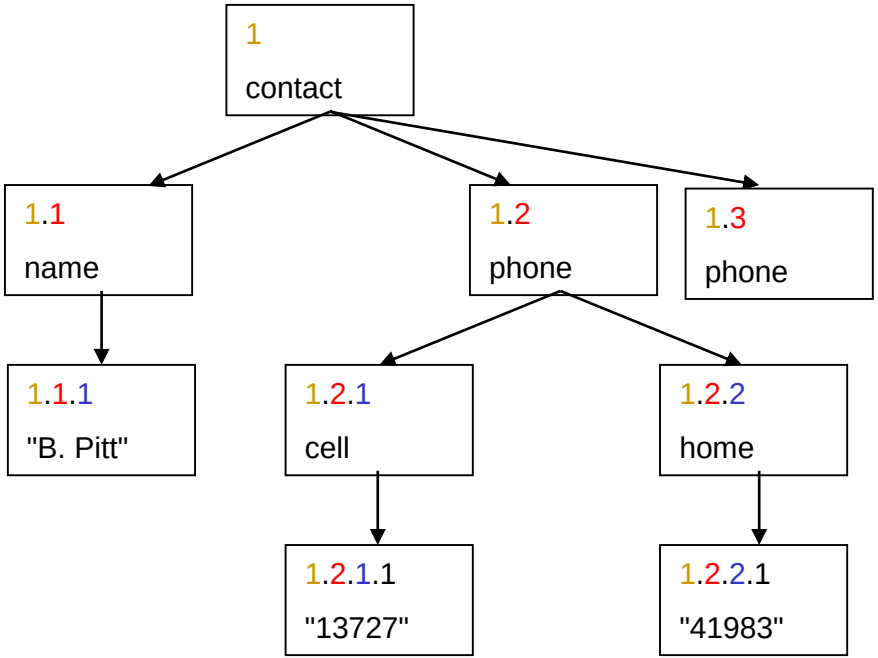
DEPTH-FIRST (DF) NUMBERING



preorder traversal +

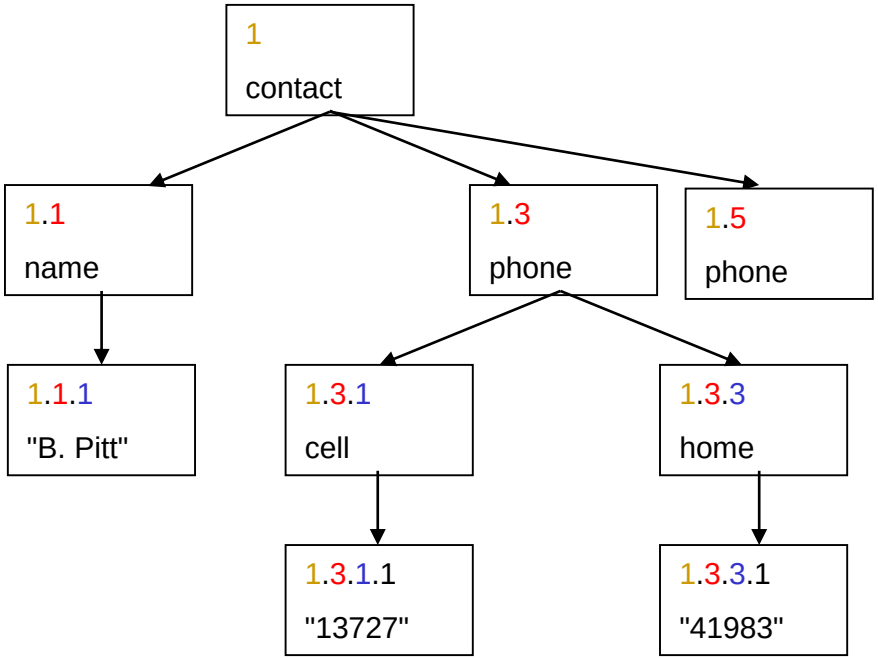
- assigning (u_{min}, u_{max}), where
- u_{min} is the time of visiting a node
- u_{max} is the time of leaving a node
- Predicate is the same as in the previous case

PATH NUMBERING



- The predicate corresponds to searching a substring
- Problem: updates

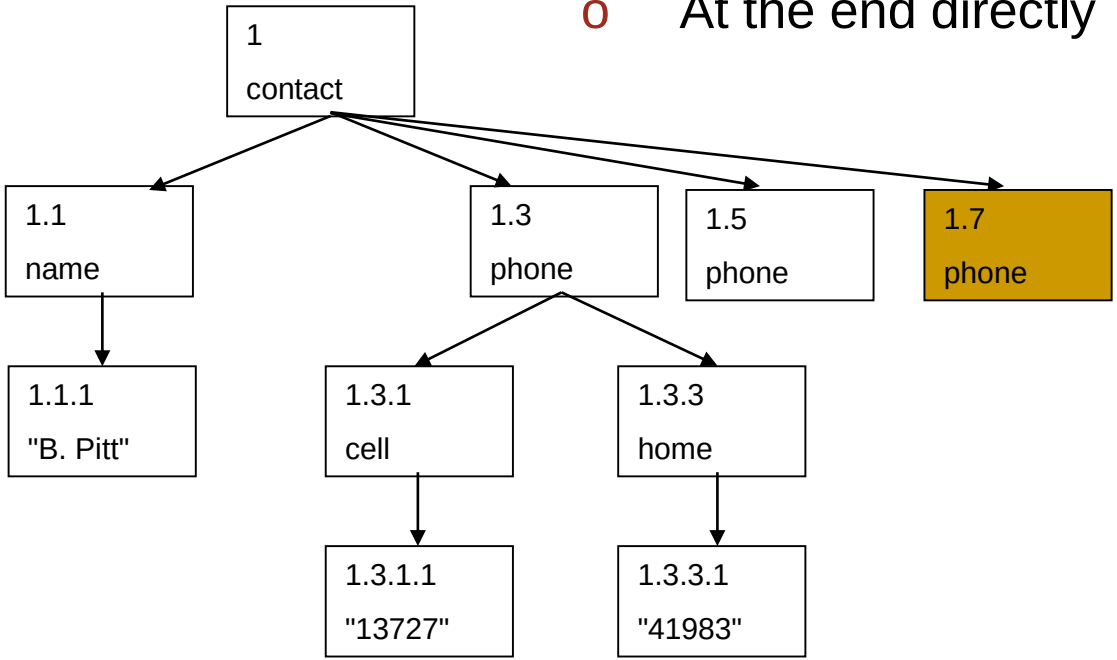
ORDPATH



- New level of tree = new level of numbering
- We use only odd numbers

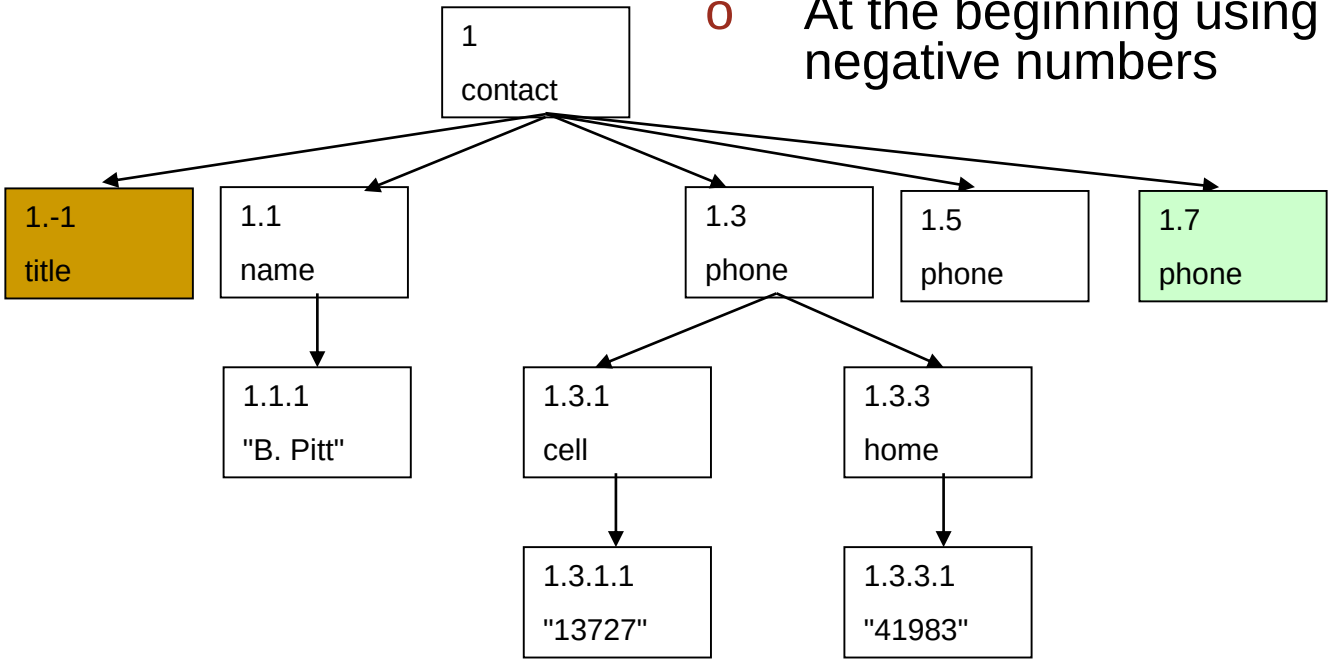
ORDPATH – INSERT

- At the end directly



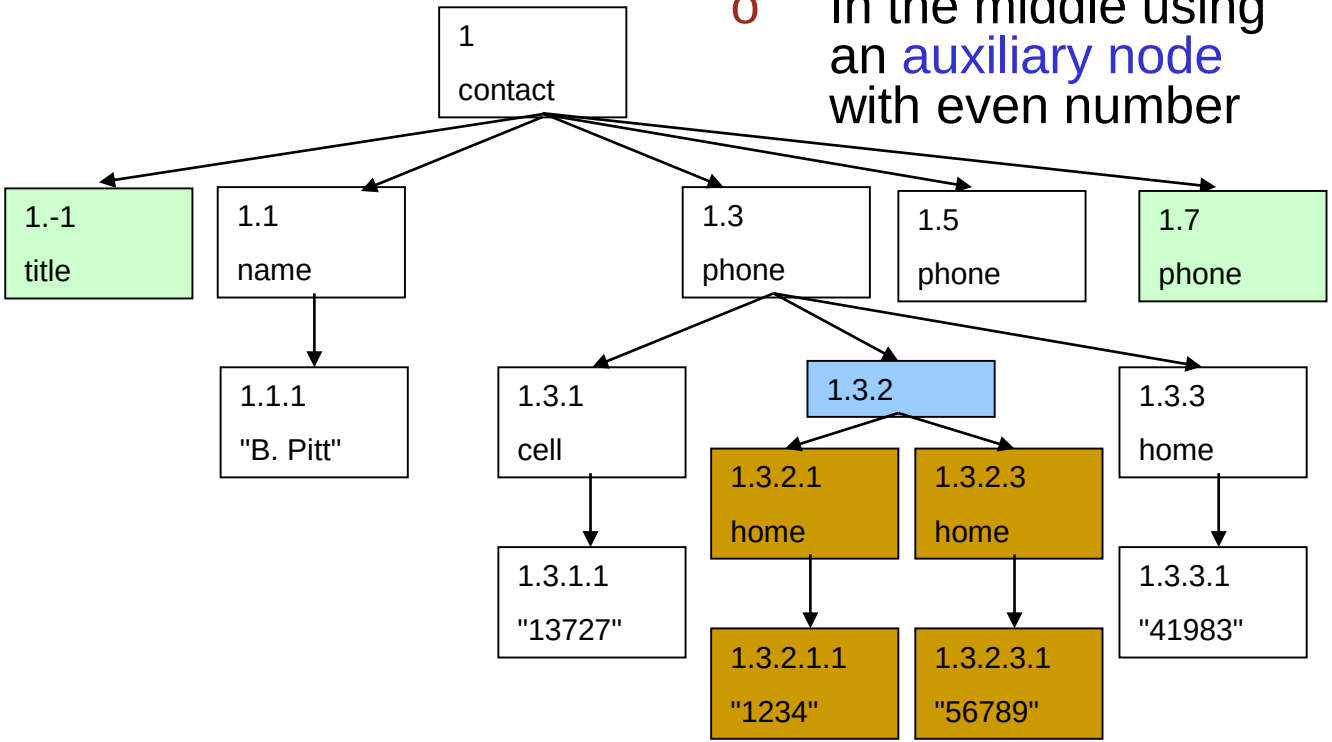
ORDPATH – INSERT

- At the beginning using negative numbers



ORDPATH – INSERT

- In the middle using an auxiliary node with even number



XML DATABASES

- What we want: persistent storage of XML data
- General classification:
  - Based on a file system
  - Based on an object model
  - Based on (object-)relational databases
    - XML-enabled databases
    - Exploit a mapping method between XML data and relations
  - Native XML databases
    - Exploit a suitable data structure for hierarchical tree data
    - Usually, a set of numbering schemas
      - Later adopted also by the XML-enable databases

LINK

- The recording of this lecture can be found here:

https://www.ksi.mff.cuni.cz/~holubova/NDBI007/download.php?file=NDBI007-2022-12-22

Login: student

Password: PDOhi2022