

# Advanced Databases

## Querying XML - XPath

**Professor Alexandra I. Cristea**  
**Epiphany Term**

[alexandra.i.cristea@durham.ac.uk](mailto:alexandra.i.cristea@durham.ac.uk)

Room: E291 Christopherson Building

Tel: 42 761

# Course Outline

- Enhanced Entity-Relationship (EER) Model
- Semistructured Databases - XML
- **XML Data Manipulation - XPath, XQuery**
- Transactions and Concurrency
- Distributed Transactions
- Distributed Concurrency
- Privacy

# Querying XML data

How would you **query** a **directed tree**?

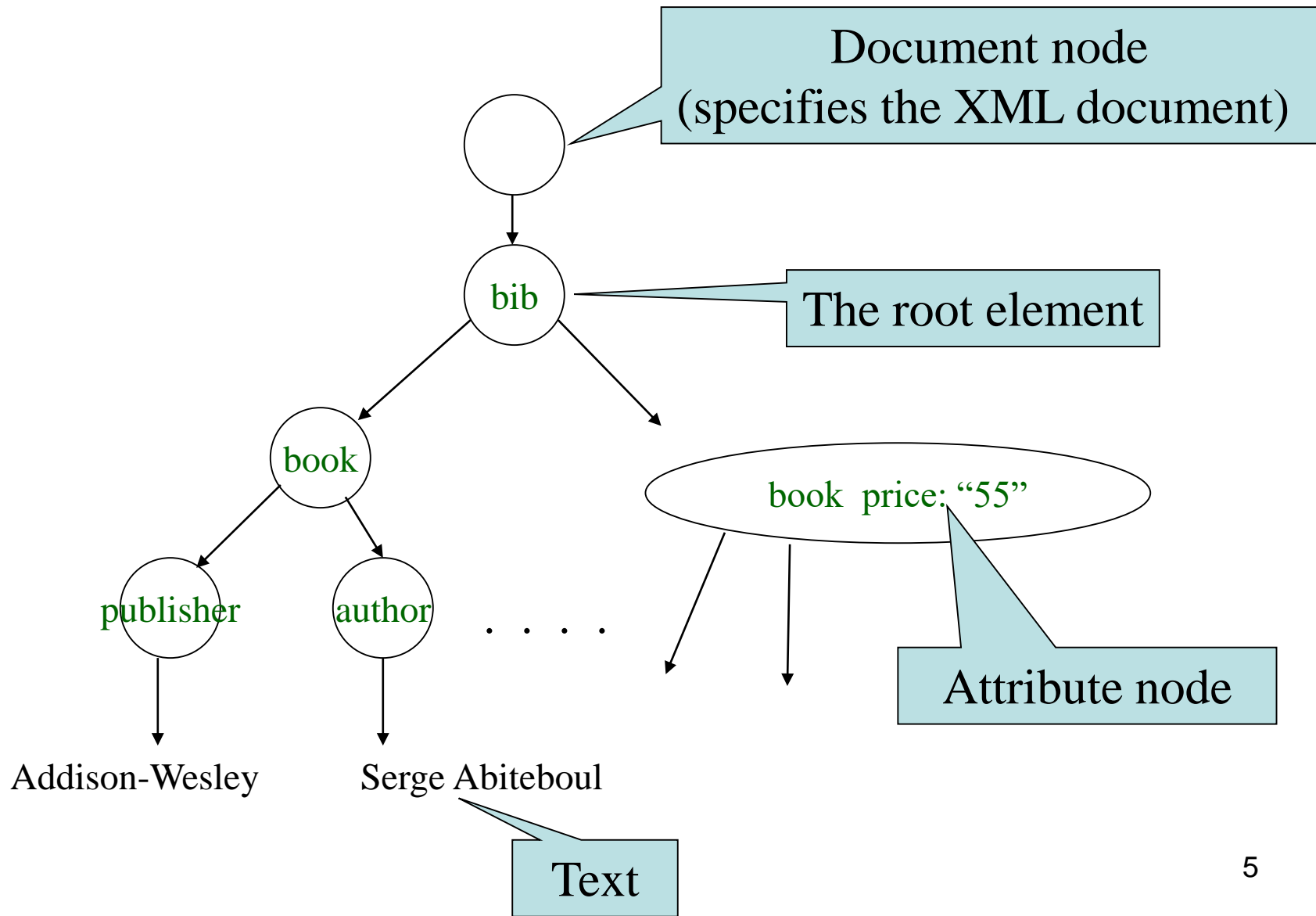
- A common approach: define a template describing **traversals** from the **root**
  - simple navigation through the tree
- **XPath**: the basis of this template
  - navigates through the tree to select data
- **XQuery**: the complete XML query language
  - “the SQL of XML”
  - selects / combines data (using XPath) and constructs output

# Path Expressions

## XPath:

- a declarative (**non-procedural**) query language for XML
- **simple syntax** for addressing parts of an XML document
- treats an **XML** document as a **logical ordered tree** with nodes:
  - root, elements, attributes, text
- a **location path** is composed by a **series** of **steps**, joined with '**/**'
  - like in a **directory path**

# Data model for XPath



# Sample Data for Queries

```
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> <firstName> Rick </firstName>
              <lastName> Hull </lastName>
    </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>

  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
```

# XPath: Simple Expressions

Navigation path: Output is the end of the path !

`/bib/book/year`

Result: `<year> 1995 </year>`  
`<year> 1998 </year>`

`/bib/paper/year`

Result: empty (there were no papers)

# Restricted Kleene Closure



`//author`

Find **any** node  
with tag “author”

Result:

```
<author> Serge Abiteboul </author>
<author>  <firstName> Rick </firstName>
           <lastName> Hull </lastName>
</author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>
```

4 nodes  
of the  
tree

`/bib/book//firstName`

Result: `<firstName> Rick </firstName>`



# Kleene Closure in Logic

In **mathematical logic** (background):

- let  $V_0 = V$  be a set of strings,  
including the *empty* string  $\varepsilon$

- define recursively:

$$V_{i+1} = \{uw \mid u \in V_i \text{ and } w \in V\}$$

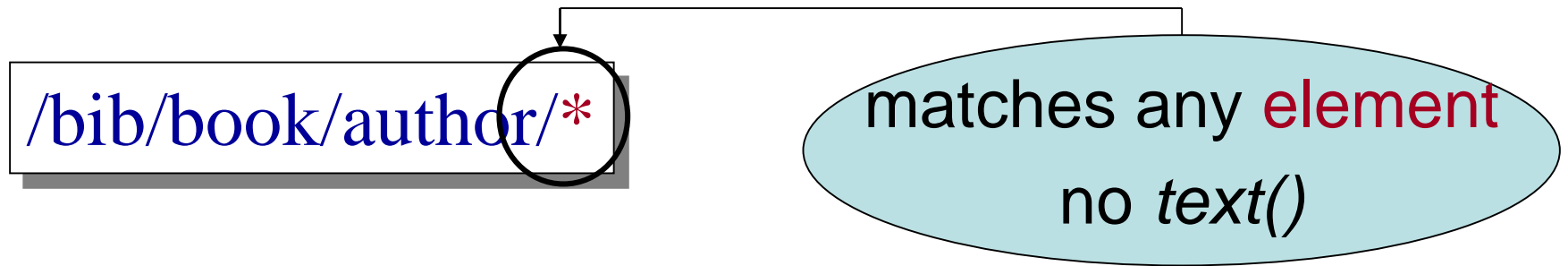
- The **Kleene closure** (or **Kleene star**) on  $V$  is:

$$V^* = \bigcup_{i \geq 0} V_i = V_0 \cup V_1 \cup V_2 \cup V_3 \dots$$

i.e. the set of all possible strings obtained  
by **concatenations** of strings in  $V$

// in **XPath**: “restricted” (i.e. finite) Kleene closure  
of elements with their subelements

# XPath: Wildcards



Result: `<firstName>` Rick `</firstName>`

`<lastName>` Hull `</lastName>`

Note: the same as `//author/*`

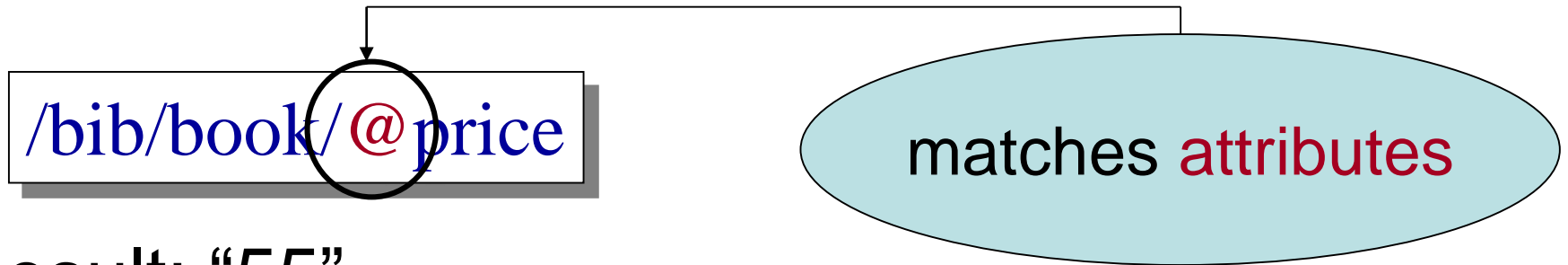
`//book/*`

Result: `<publisher>` Addison-Wesley `</publisher>`

`<author>` Serge Abiteboul `</author>`

...

# XPath: Attribute nodes



Result: "55"

`@price` means that `price` has to be an **attribute**

`@*` matches **any attribute**

# XPath: Predicates

In XPath, we can add a **predicate** after a tag:

- a **Boolean** condition in **square brackets**

⇒ follow only the subset of paths whose tags satisfy the predicate (i.e. make it “true”)

```
//book[@price = “55”]
```

Result: `<book price=“55”>`

`<publisher> Freeman </publisher>`

`<author> Jeffrey D. Ullman </author>`

`<title> Principles of Database and Knowledge Base Systems </title>`

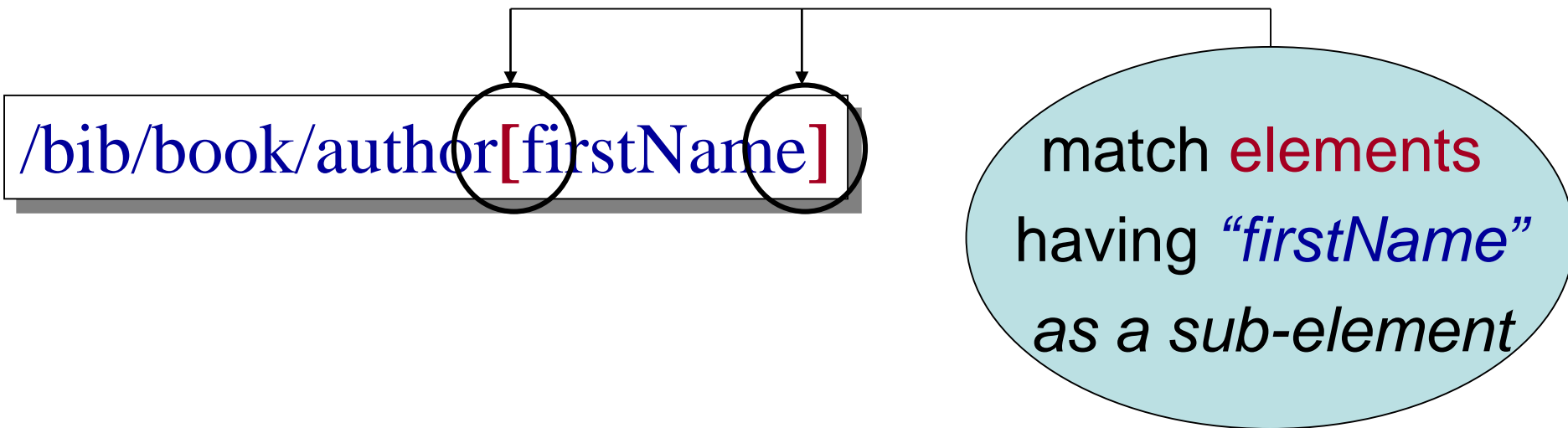
`<year> 1998 </year>`

`</book>`

# XPath: Predicates

In XPath predicates:

- comparisons have an implicit “there exists” sense



Result: `<author>`  
          `<firstName> Rick </firstName>`  
          `<lastName> Hull </lastName>`  
          `</author>`

# More than one predicate

Find all *books*, which have *price* 55 **and**  
a *firstName* (at some depth) or a *publisher* “Freeman”

```
/bib//book[@price="55"][//firstName | publisher = "Freeman"]
```

The same as:

```
/bib//book[//firstName | publisher = "Freeman"][@price="55"]
```



Result: `<book price="55">`  
    `<publisher> Freeman </publisher>`  
    `<author> Jeffrey D. Ullman </author>`  
    `<title> Principles of Database and Knowledge Base Systems`  
    `</title>`  
    `<year> 1998 </year>`  
    `</book>`

# More than one predicates

- Can we always **swap predicates** in a query?
- A **priority rule**:
  - first satisfy the first predicate
  - among those nodes that satisfy the first predicate, select the nodes that satisfy the second one, etc.
- In the previous example:
  - two **independent predicates**  $\Rightarrow$  we can **swap them**

```
/bib//book[@price="55"][//firstName | publisher = "Freeman"]
```

- **Not always** the case !

# XPath: Axes

The **general form** of an XPath query:

```
/step-1/step-2/step-3/.../step-n
```

- Each of these **steps** consists of:
  - a **basis** and (optionally) a list of **predicates**
- A basis consists of:
  - an **axis** (the **direction** in which the navigation proceeds from the current node)
  - a **node test** (the **type of node** we navigate to)
- So far we only navigated:
  - from a node to its **children** or to an **attribute**



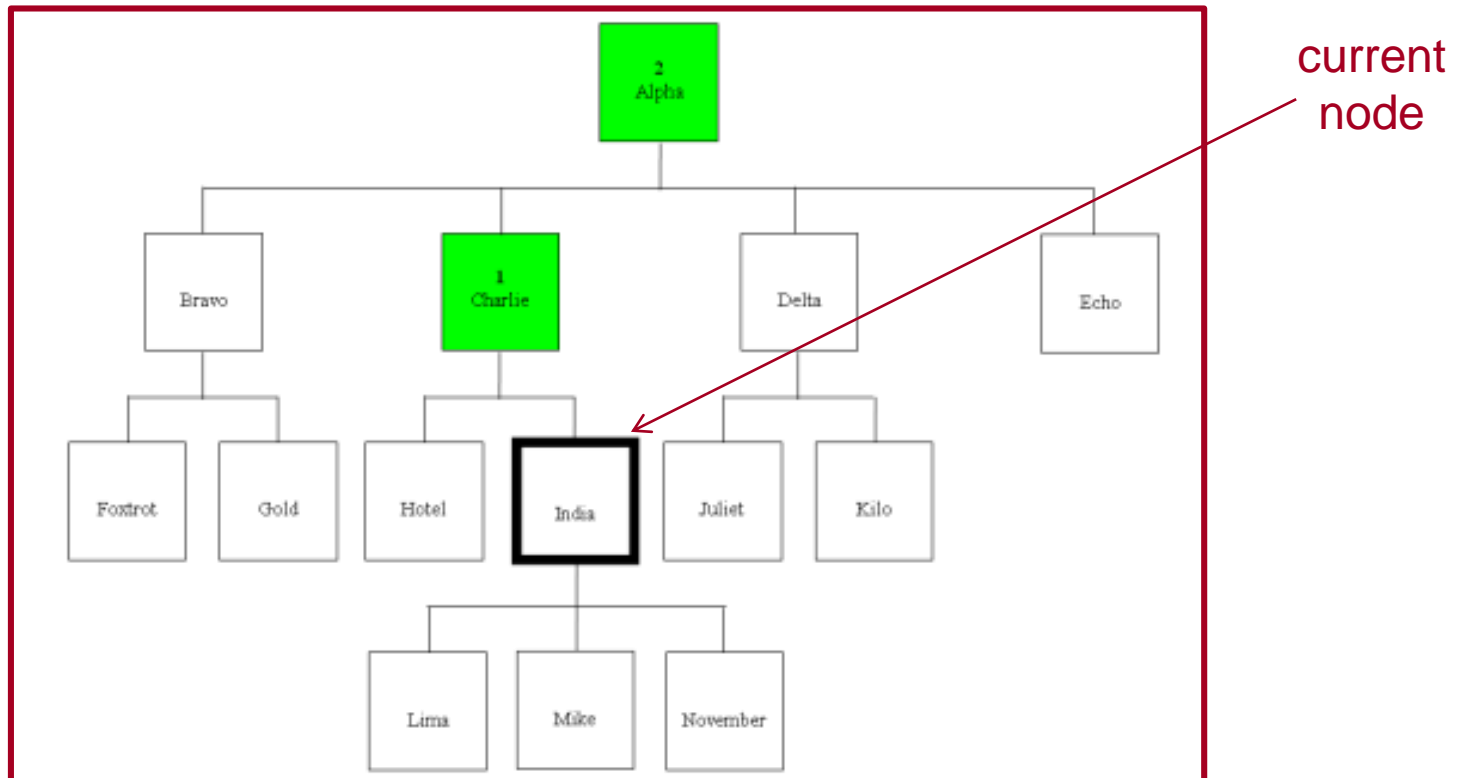
# XPath: Axes

- An **axis** in XPath is a “local navigation mode”:
- XPath provides several **axes**, among them:
  - child (the default axis)
  - parent
  - attribute
  - self
  - descendant
  - descendant-or-self
  - ancestor
  - ancestor-or-self
  - following-sibling
  - preceding-sibling
  - following
  - preceding

nodes **after** / **before** the current node in the (serial) **XML document order**, which are **not descendants** / **not ancestors** of the current node

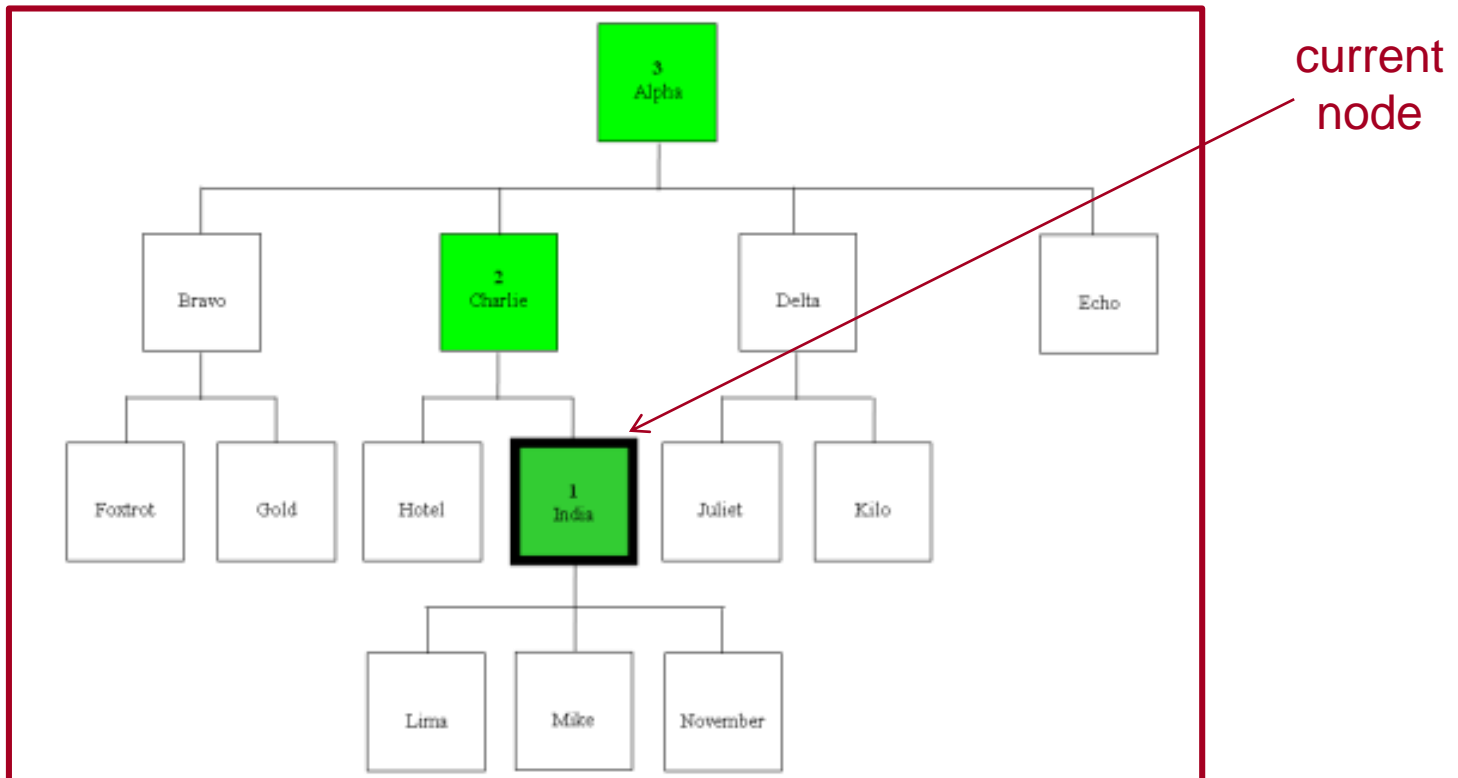
# XPath: Axes (examples)

- **Ancestor** axis:
  - parent and parent's parent, etc.



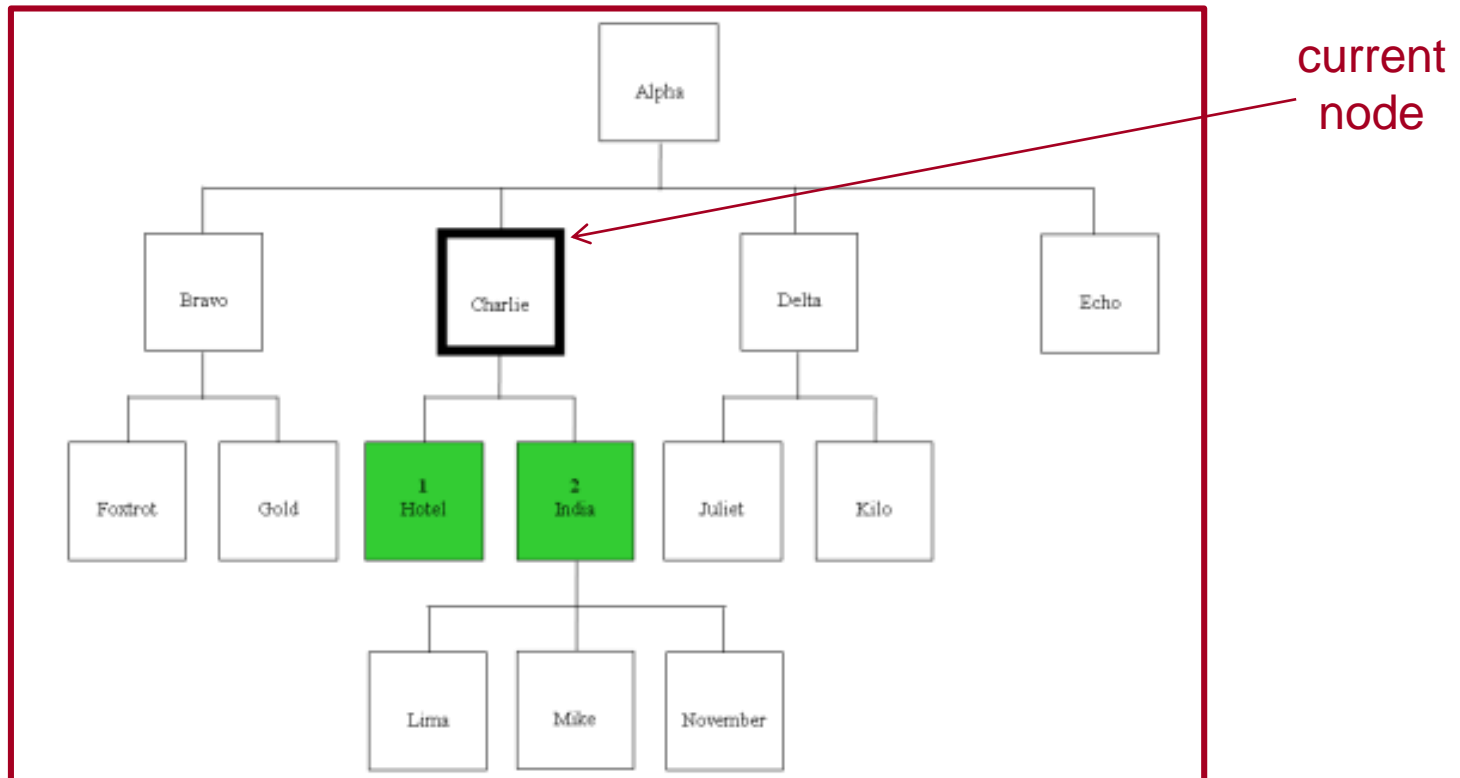
# XPath: Axes (examples)

- **Ancestor-or-self** axis:
  - current node and parent and parent's parent, etc.



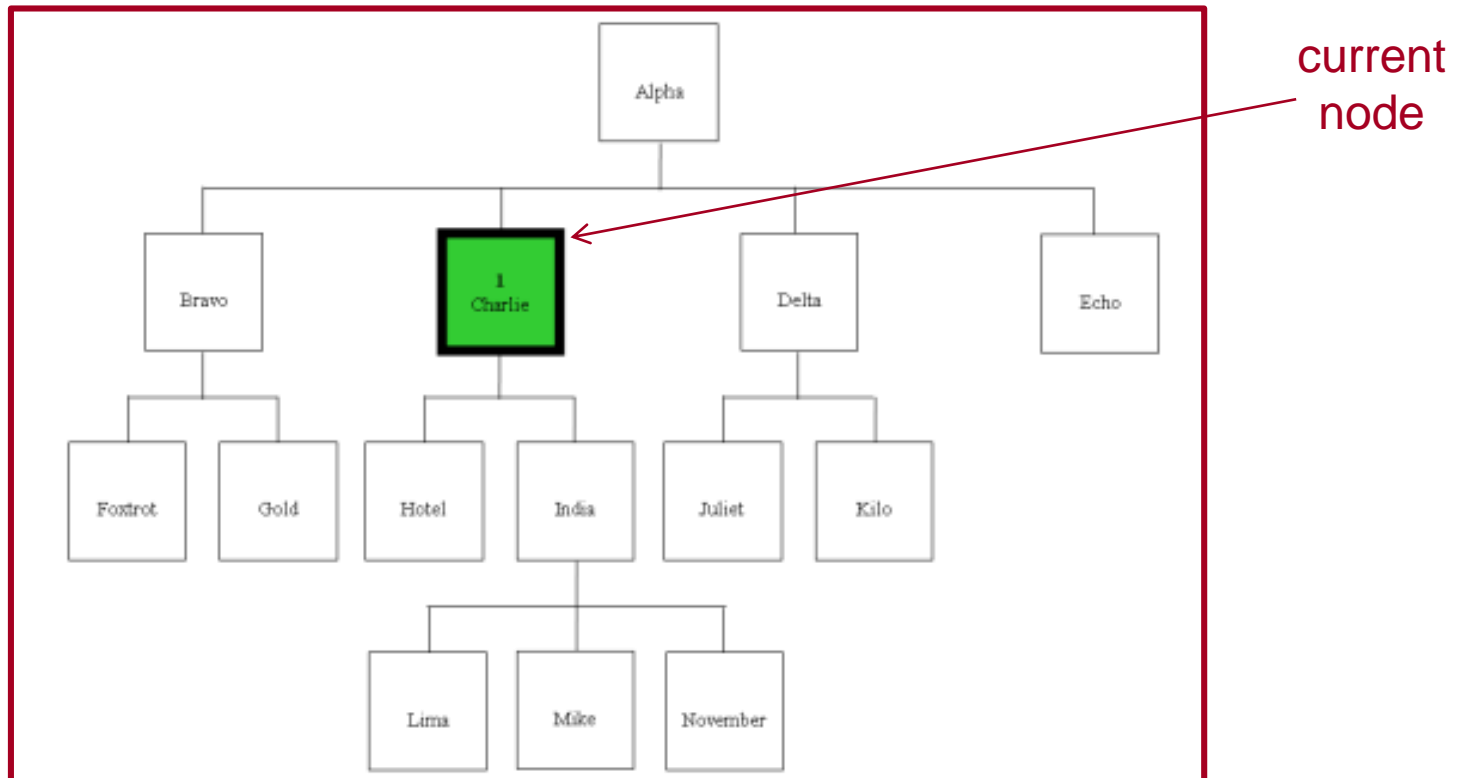
# XPath: Axes (examples)

- **Child** axis:
  - children of the current node



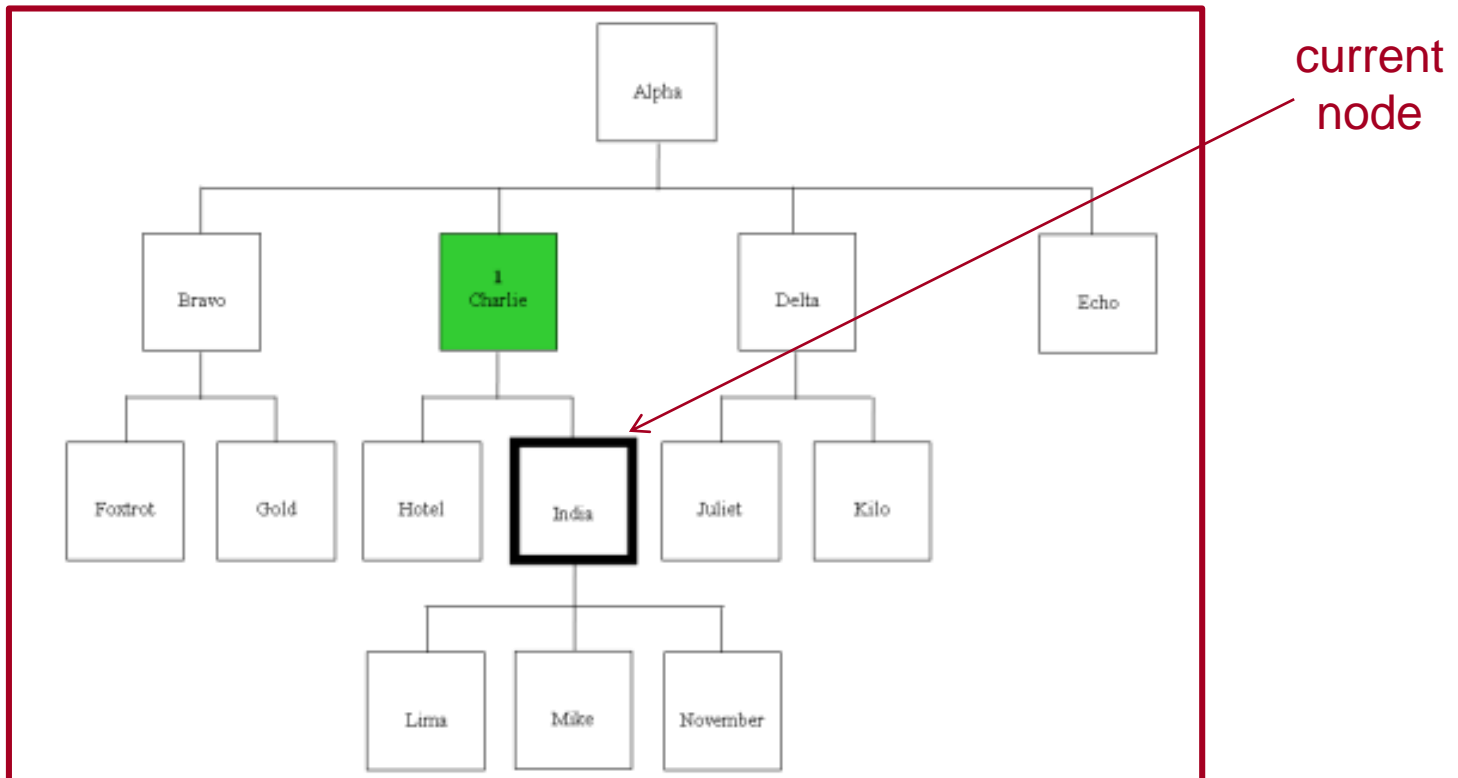
# XPath: Axes (examples)

- **Self** axis:
  - the current node



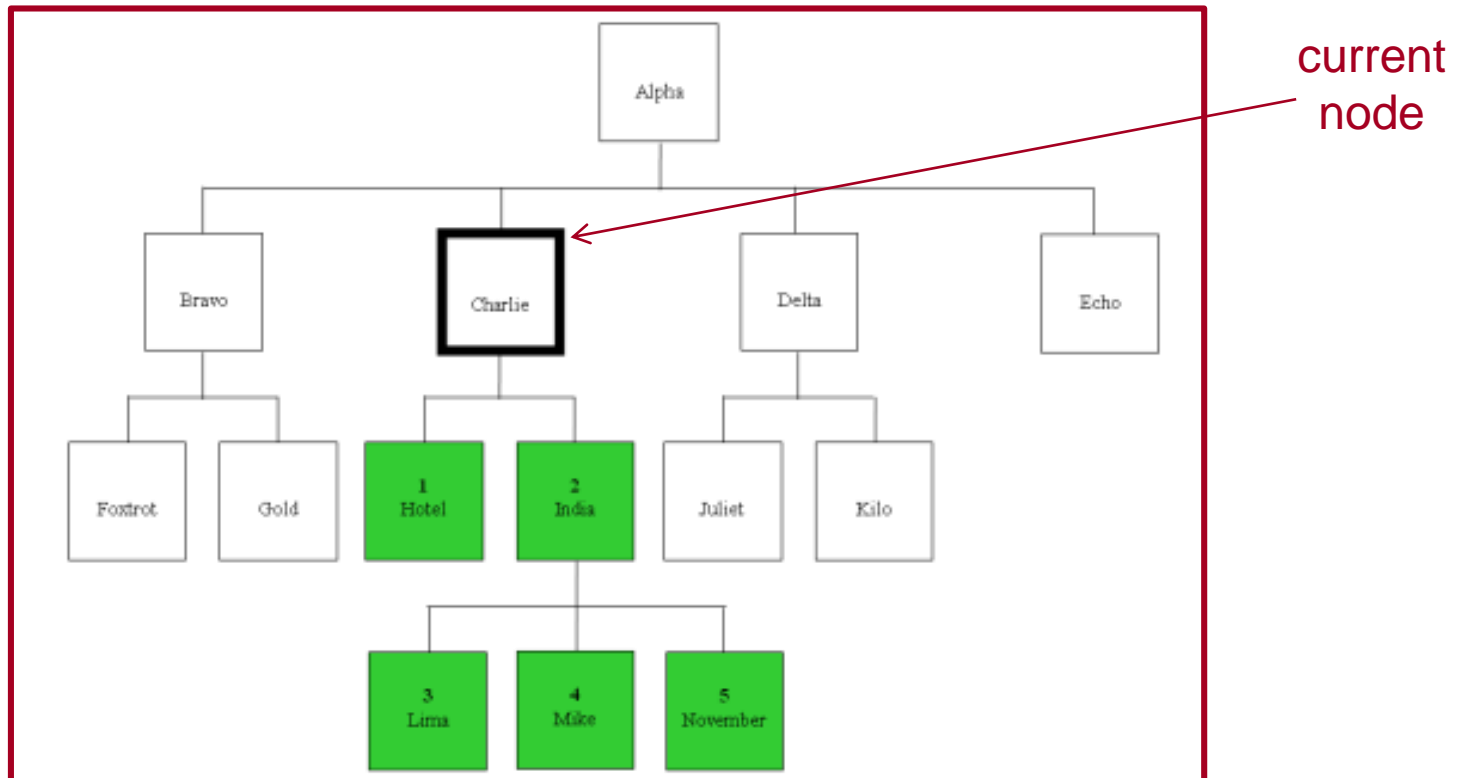
# XPath: Axes (examples)

- **Parent** axis:
  - the parent of the current node



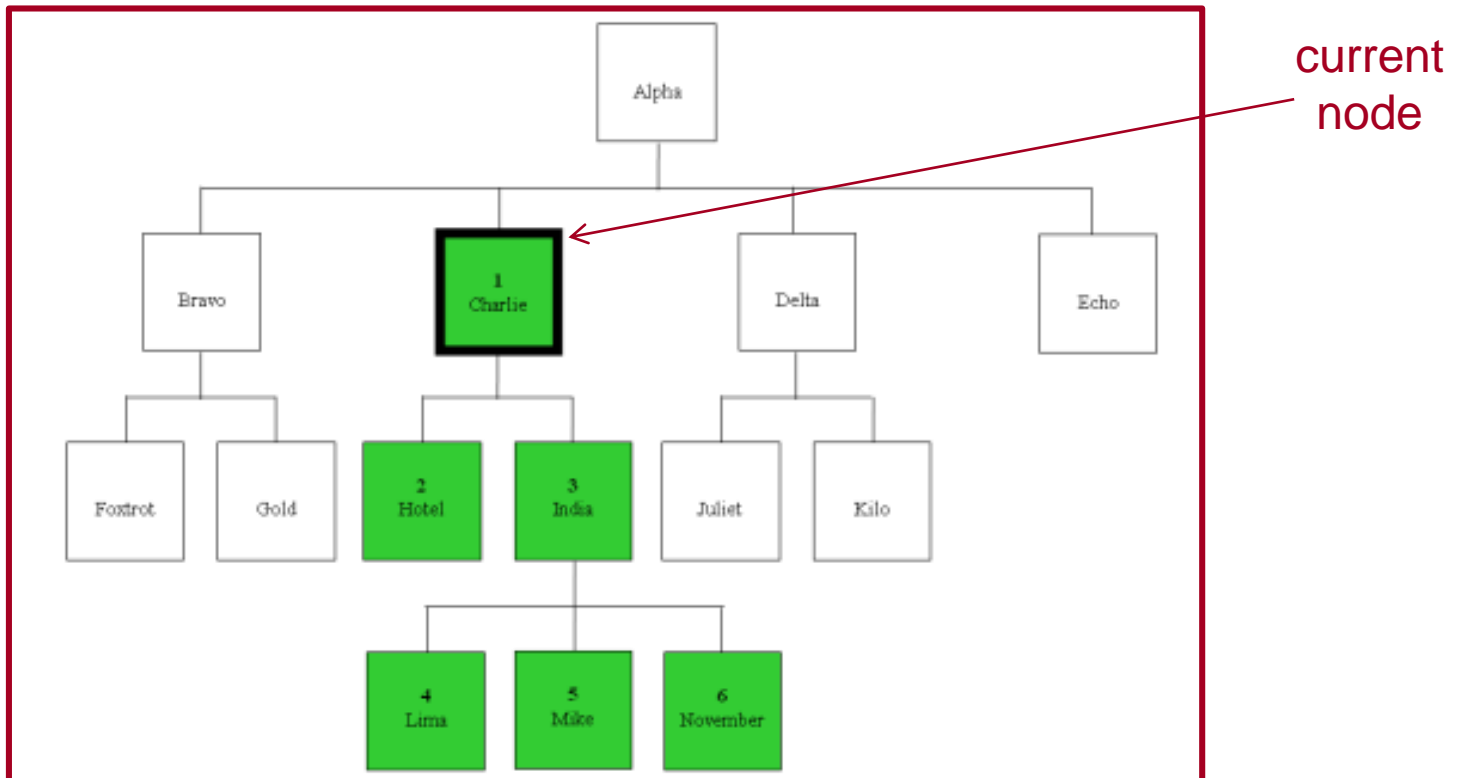
# XPath: Axes (examples)

- **Descendant** axis:
  - children and their children, etc.



# XPath: Axes (examples)

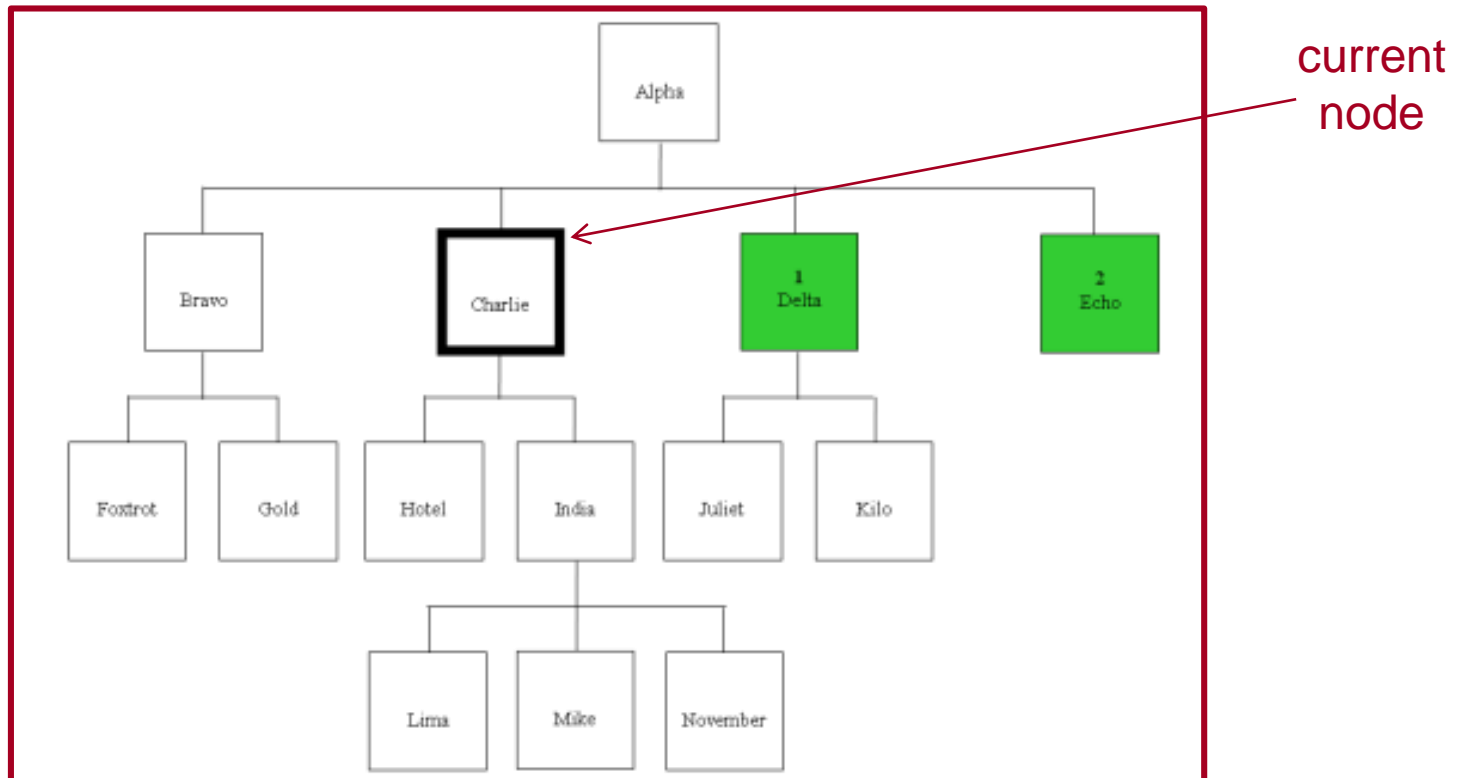
- **Descendant-or-self** axis:
  - current node and children and their children, etc.





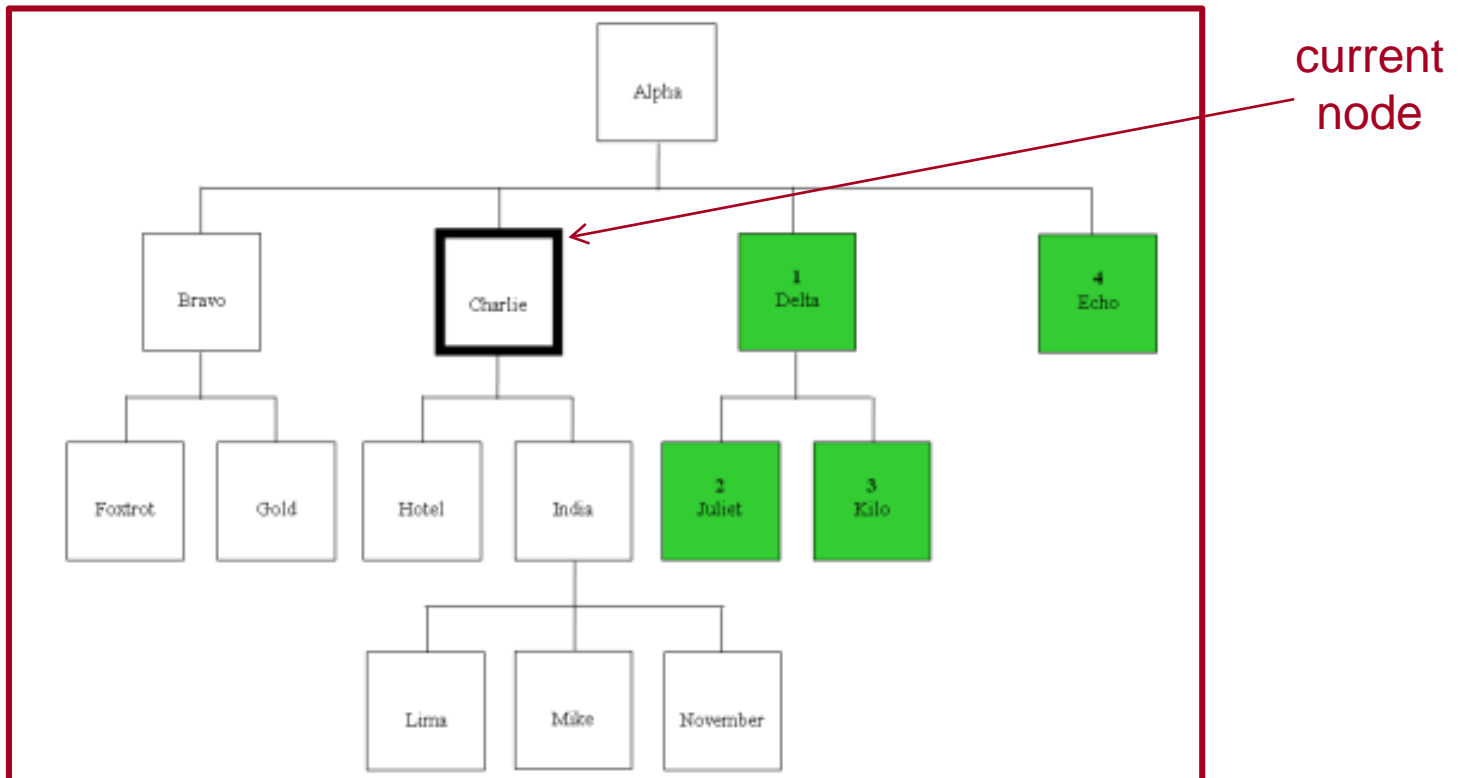
# XPath: Axes (examples)

- **Following-sibling** axis:
  - following siblings (in the XML document order)



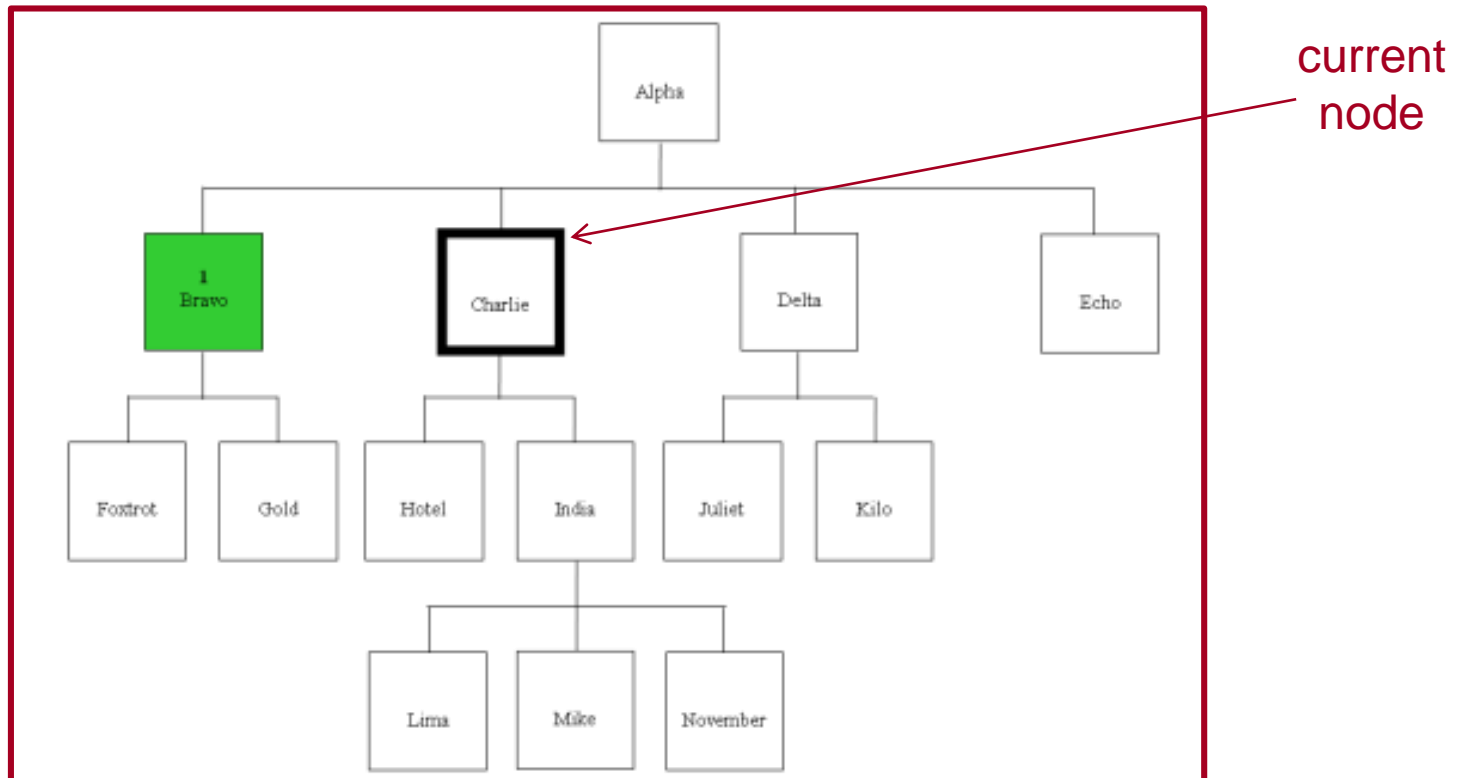
# XPath: Axes (examples)

- **Following** axis:
  - following siblings and their children and their children, etc.



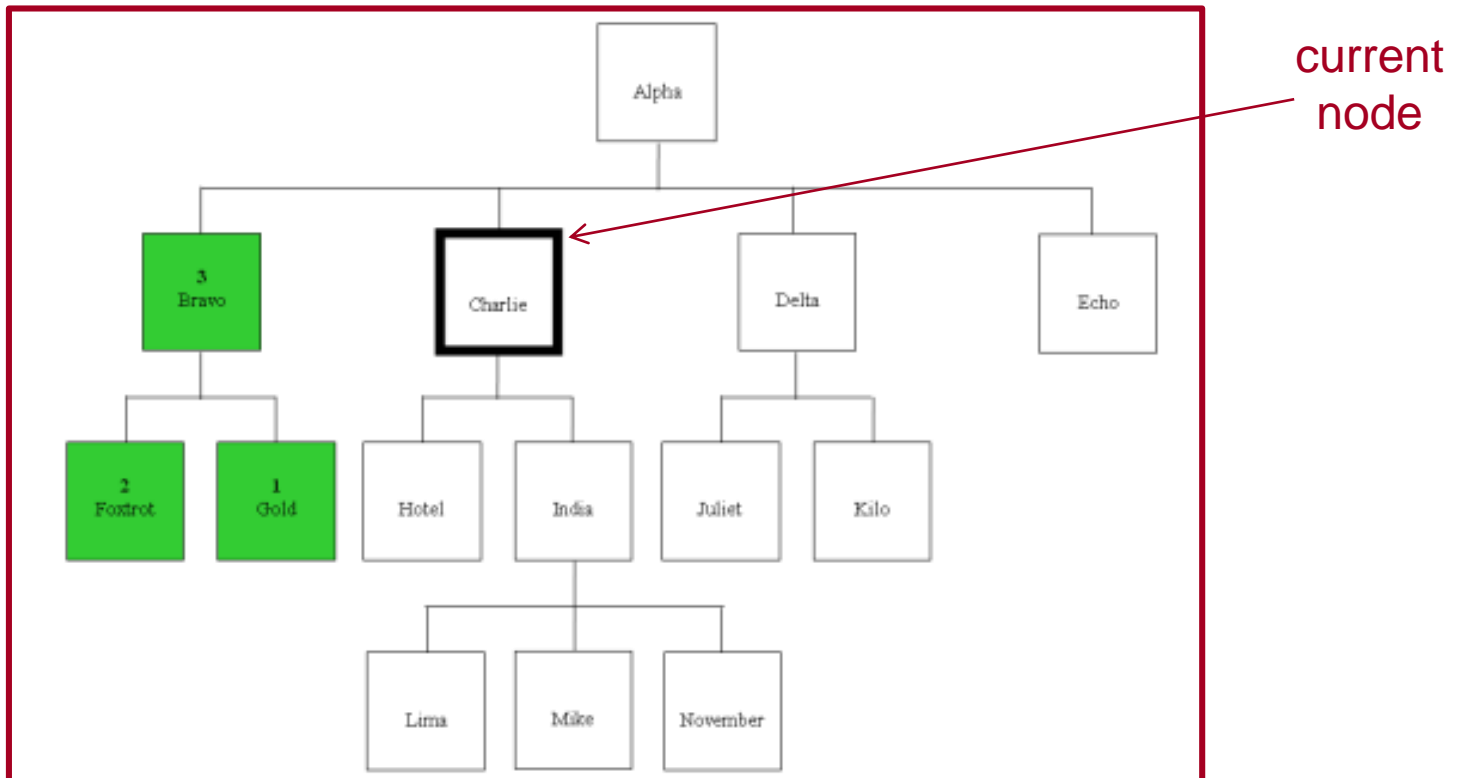
# XPath: Axes (examples)

- **Preceding-sibling** axis:
  - preceding siblings (in the XML document order)



# XPath: Axes (examples)

- **Preceding** axis:
  - preceding siblings and their children and their children, etc.



# XPath: Axes (syntax)

- **prefix** a tag (or attribute name) by an **axis name** and a **double-colon**:

```
/bib/book/@price
```

is equivalent to:

```
/child::bib/child::book/attribute::price
```

---

```
//book/*
```

is equivalent to:

```
/descendant-or-self::book/child::*
```

# Functions in XPath

## Basic functions in XPath:

- `text()` = matches the text value
- `name()` = returns the name of the current tag

`/bib/book/author/text()`

Result:     Serge Abiteboul  
              Victor Vianu  
              Jeffrey D. Ullman     } output is text,  
    not nodes !

Note: “Rick Hull” doesn’t appear,  
because he has `firstName`, `lastName`  
i.e. no `text()` within his “author” element

# Functions in XPath

## Basic functions in XPath:

- `text()` = matches the text value
- `name()` = returns the name of the current tag

```
/bib//*[name() = "book"]
```

Result:

```
<book>
    <publisher> Addison-Wesley </publisher> ...
</book>
<book price="55">
    <publisher> Freeman </publisher> ...
</book>
```

Note: the same as `/bib//book`

# Positional Predicates

- **Order** of elements matters in **XML** !
- We can select a subelement of an element using the predicate format:

```
[position() = i ]
```

where **i** is the position of the desired subelement within the element (starting from 1)

```
//author[position() = 2]
```

Result:   <author>   <firstName> Rick   </firstName>  
                  <lastName> Hull   </lastName>  
          </author>

i.e. “the second author” of an element



# Positional Predicates

- An alternative way of selecting a subelement that has a particular order:

`ElemTag[ i ]`

where `i` is the position of the desired subelement among the `ElemTags`

`/bib/book/author[2]`

i.e. “the second author of a book”

Result: `<author>` `<firstName>` Rick `</firstName>`  
          `<lastName>` Hull `</lastName>`  
          `</author>`

# XPath: More Predicates

```
/bib/ book/ author [firstName] [address [//zip] [city]] / lastName
```

## Nested predicates:

- “Find all authors who have:
  - a sub-element **firstName** and
  - a sub-element **address**, which has:
    - a **zip** code (as a descendant at some depth)
    - **and** a **city** (as a child)
- and **return the lastName** of these authors (if they have one)”

Result:     <lastName> . . . </lastName>  
             <lastName> . . . </lastName>

# XPath: More Predicates

```
/bib/book[@price < “60”]
```

```
/bib/book[author/@age < “25”]
```

```
/bib/book[author/text()=“John”]
```

```
/bib/book[year > “1996”]/year
```

# XPath: More Predicates

**Test:** “find the *first* book *that has* price 55”

```
/descendant-or-self::book[@price = “55”]  
[position()=1]
```

or equivalently:

```
//book[@price = “55”][position()=1]
```

**Test:** “find the *first* book, *if it has* price 55”

```
/descendant-or-self::book[position()=1]  
[@price = “55”]
```

or equivalently:

```
//book[position()=1][@price = “55”]
```

# Positional Predicates vs. Axes

- An important note on the positional predicates:
  - we always **count positions** on the **specified axis**
- For a **reverse axis** (e.g. ancestor, preceding, etc.):
  - we **count** in the **reverse XML document order** !

## Example:

```
.../ancestor::*[position()=1]/...
```

is the same as:

```
.../parent::*/*...
```

# XPath: Summary

bib	matches a <b>bib</b> element
*	matches <b>any</b> element
/	matches the <b>root</b> element
/bib	matches a <b>bib</b> element <b>under root</b>
bib/paper	matches a <b>paper</b> in <b>bib</b>
bib//paper	matches a <b>paper</b> in <b>bib</b> , <b>at any depth</b>
//paper	matches a <b>paper</b> <b>at any depth</b>
paper book	matches a <b>paper</b> <b>or</b> a <b>book</b>
@price	matches a <b>price attribute</b>
bib/book/@price	matches <b>price attribute</b> in <b>book</b> , in <b>bib</b>
bib/book[@price<"55"]/author/lastName	matches...

# XPath and XQuery

- XQuery (for XML, i.e. semi-structured data):
  - is based on XPath
  - is similar to SQL (for structured data)

XQuery basic form:

FLWR (“Flower”) Expressions



```
FOR ...  
LET...  
WHERE...  
RETURN...
```

SQL basic form:

```
SELECT ...  
FROM...  
WHERE...
```

# Summary of the Lecture

- XML has a (directed) tree structure  
(semi-structured data)
- Querying XML with XPath
  - navigates through the tree to select data
- XPath:
  - output is at the end of a path
  - restricted Kleene closure: //
  - functions: text(), name()
  - wildcard: \*
  - attributes: @
  - predicates: [ . . . ]
  - axes
  - nested predicates: [ . . . [ . . . ] ]