Données du Web : XPath & XQuery

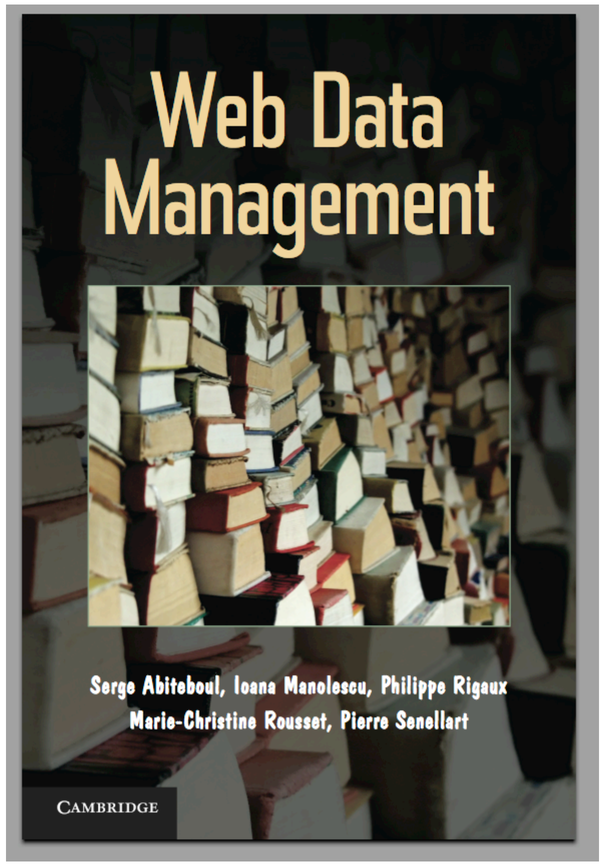
Federico Ulliana
UM, LIRMM, INRIA GraphIK Team

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

XML QUERIES

Readings

[WDM-Query]
Web Data Management – Chapter XPath / XQuery
http://webdam.inria.fr/Jorge/files/wdm-xpath.pdf


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

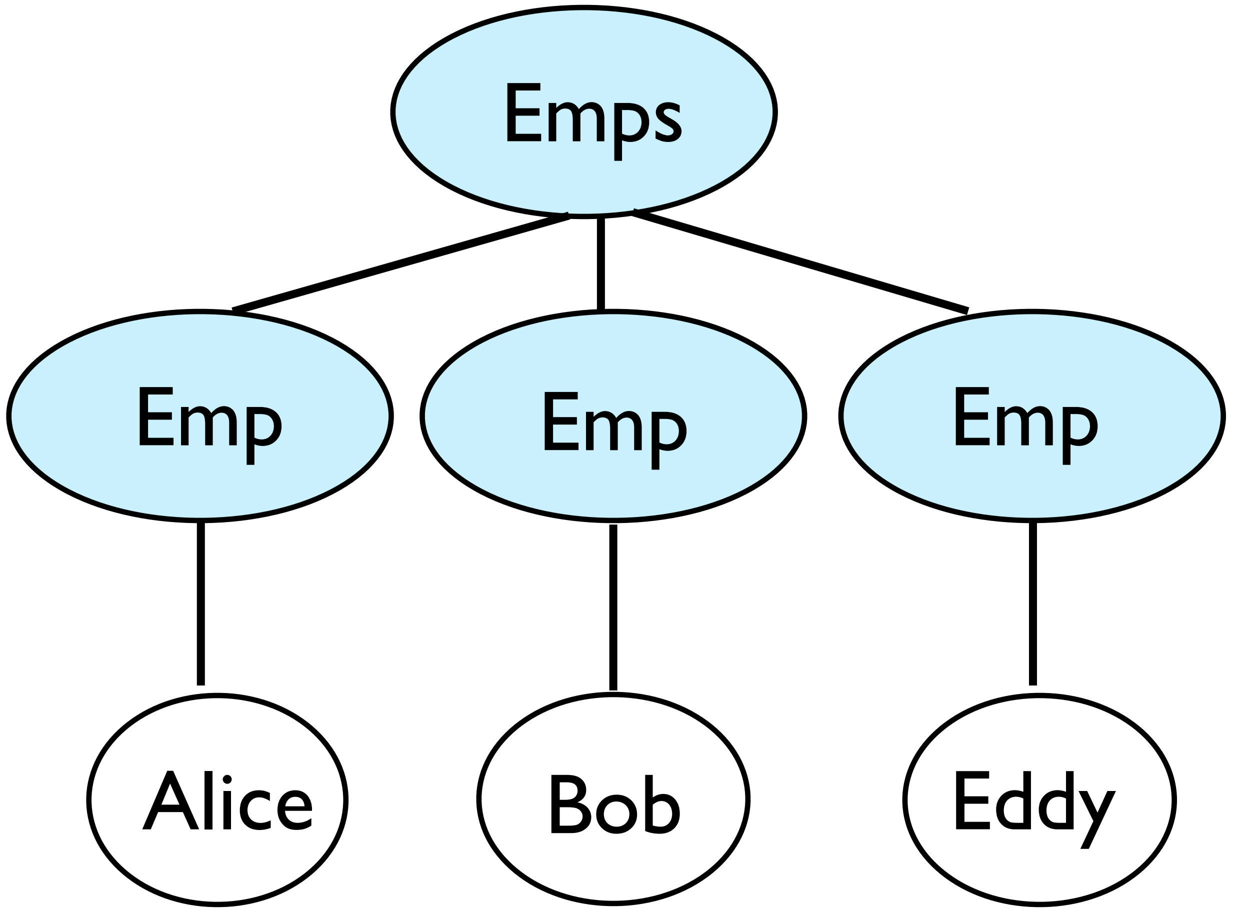
Data and Queries

Employee

name	**dept**
Alice	Sales
Bob	Production
Eddy	Sales

```
SELECT *
FROM Employee
```

Data and Queries



```
HOW TO
QUERY ???
```

XML Queries

- XML documents are hierarchical structures (trees; opposed to relational tables that are "flat")

- XML Queries should ***navigate*** hierarchical structures (XPath)

- XML Queries should ***transform*** hierarchical structures (XQuery)

W3C Standardization Roadmap

1999
XPath 1.0

2007
XPath 2.0 first edition
XQuery 1.0 first edition

2010
XPath 2.0 second edition
XQuery 1.0 second edition

2014
XQuery 3.0

2017 (support for JSON)
XPath 3.0
XQuery 3.1

XML Working Group ended on 31.08.2017

THE XPATH LANGUAGE

XPath

Path expressions to extract data from XML trees

XPath mimics “File Paths”

```
susan@bastet > cd /Volumes/Data/Documents

susan@bastet > cd ../../Users
```

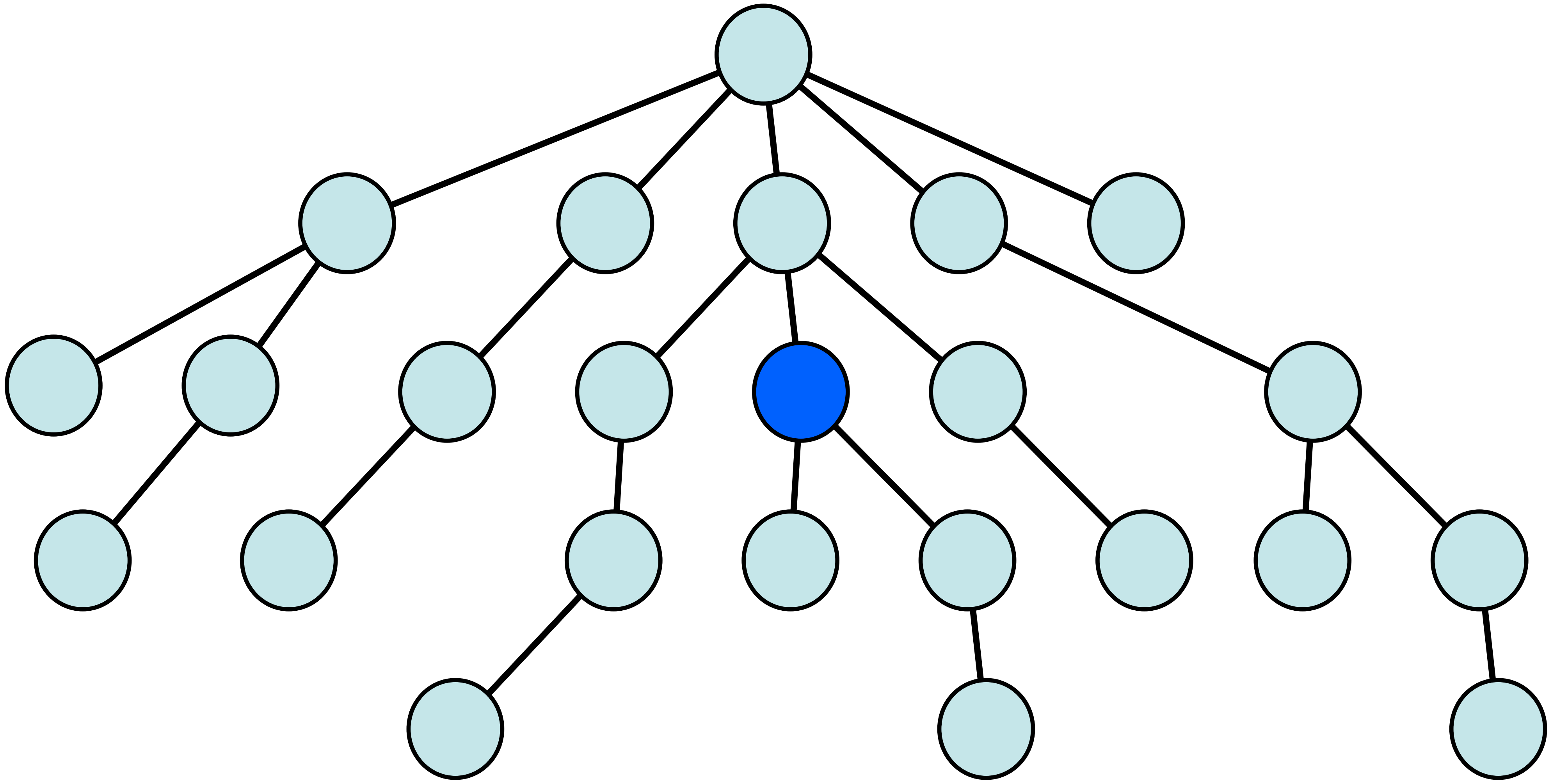
But there is more : Navigational axes, Node tests, Steps, Paths, Conditions, Built-in Functions, Equality

NAVIGATIONAL AXES

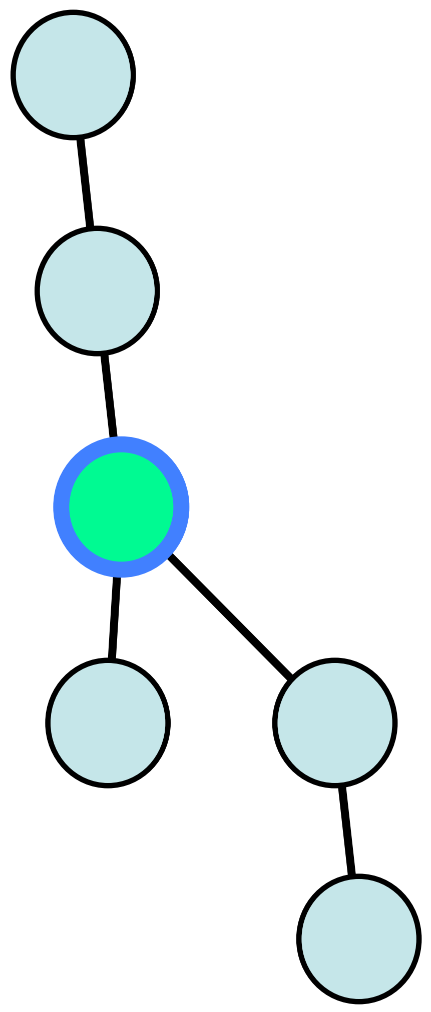
1) Navigational Axes

Principal means of accessing the nodes of an XML tree

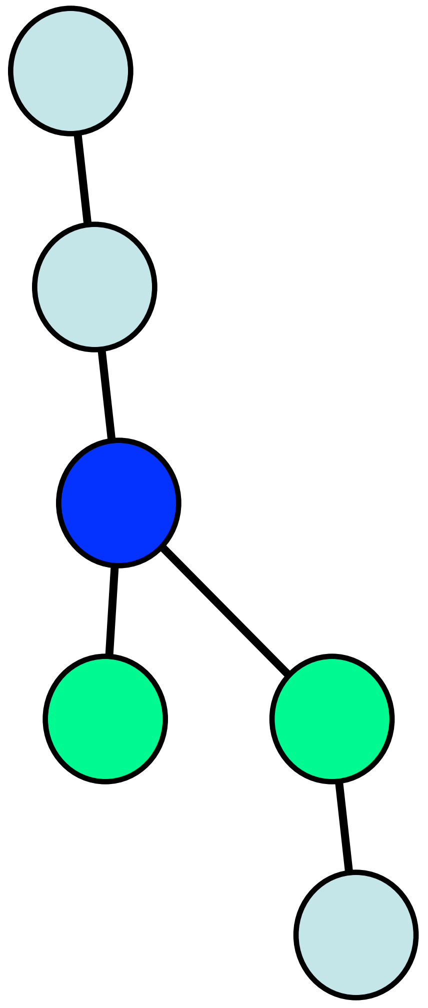
Context node = (starting point)



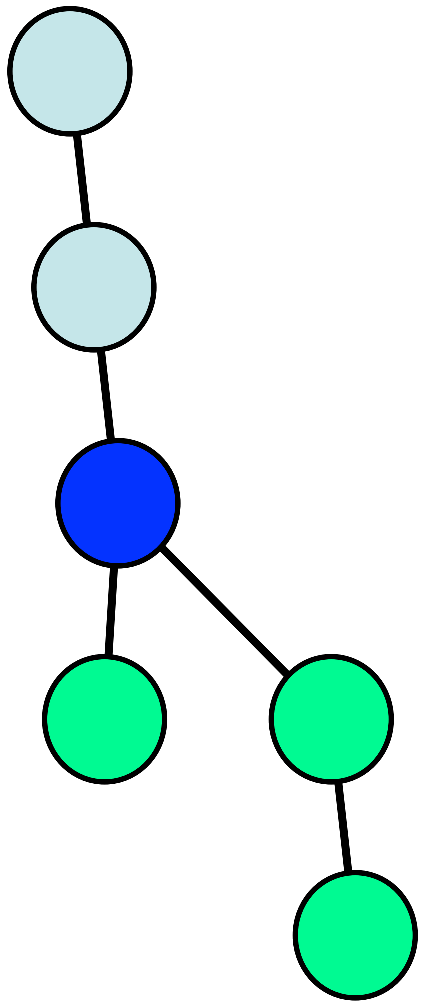
Self



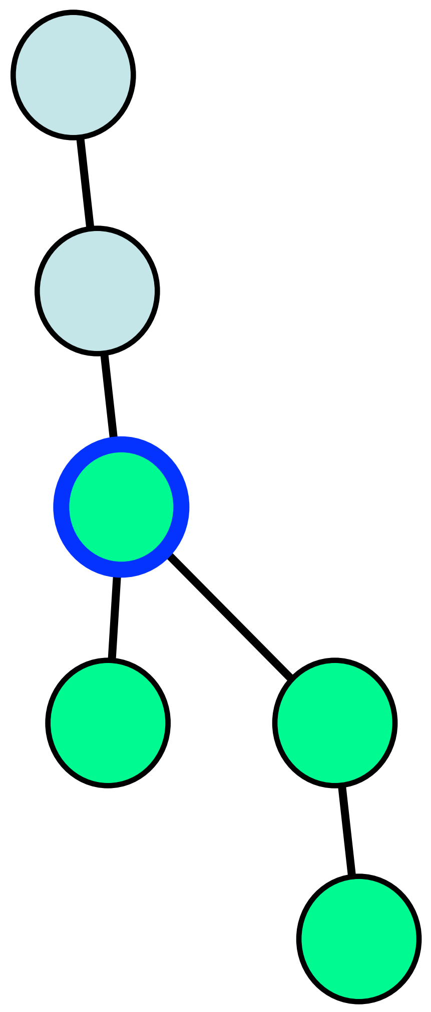
Child



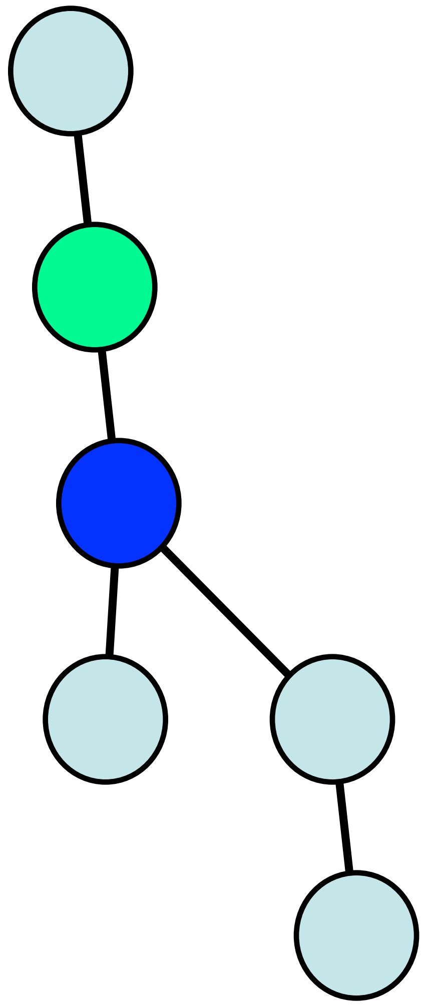
Descendant



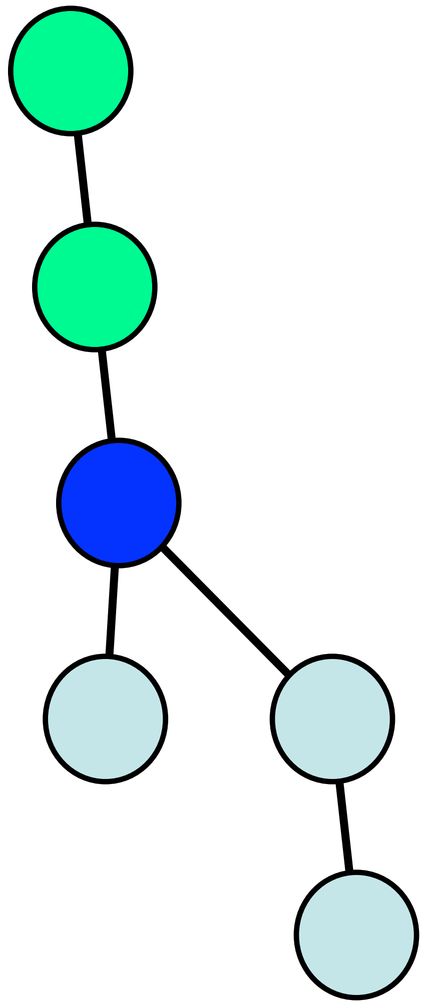
Descendant-or-self



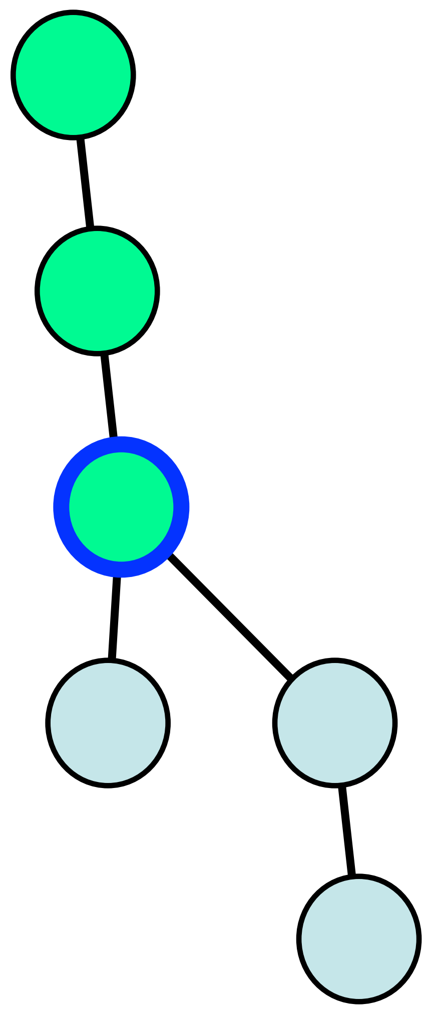
Parent



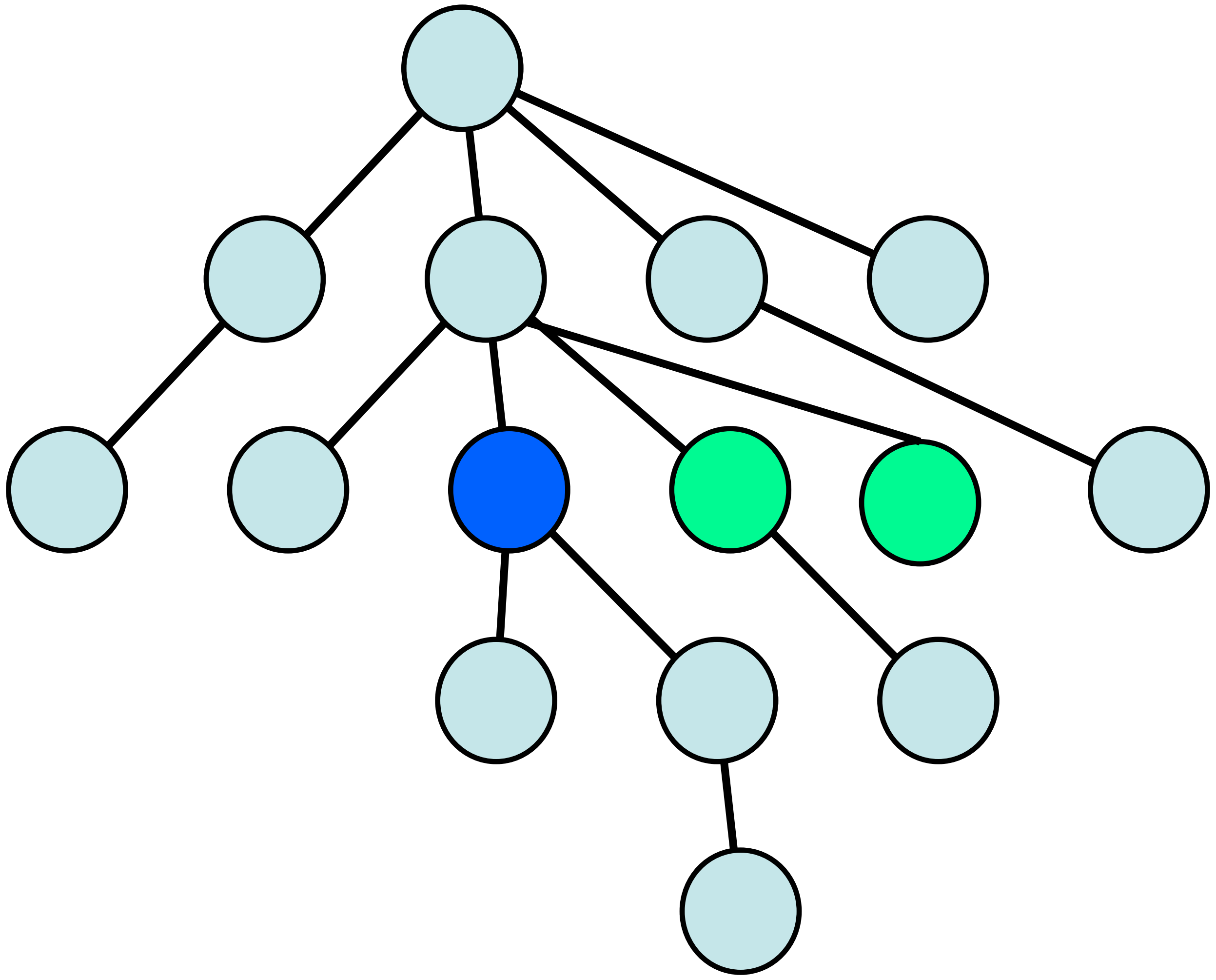
Ancestor



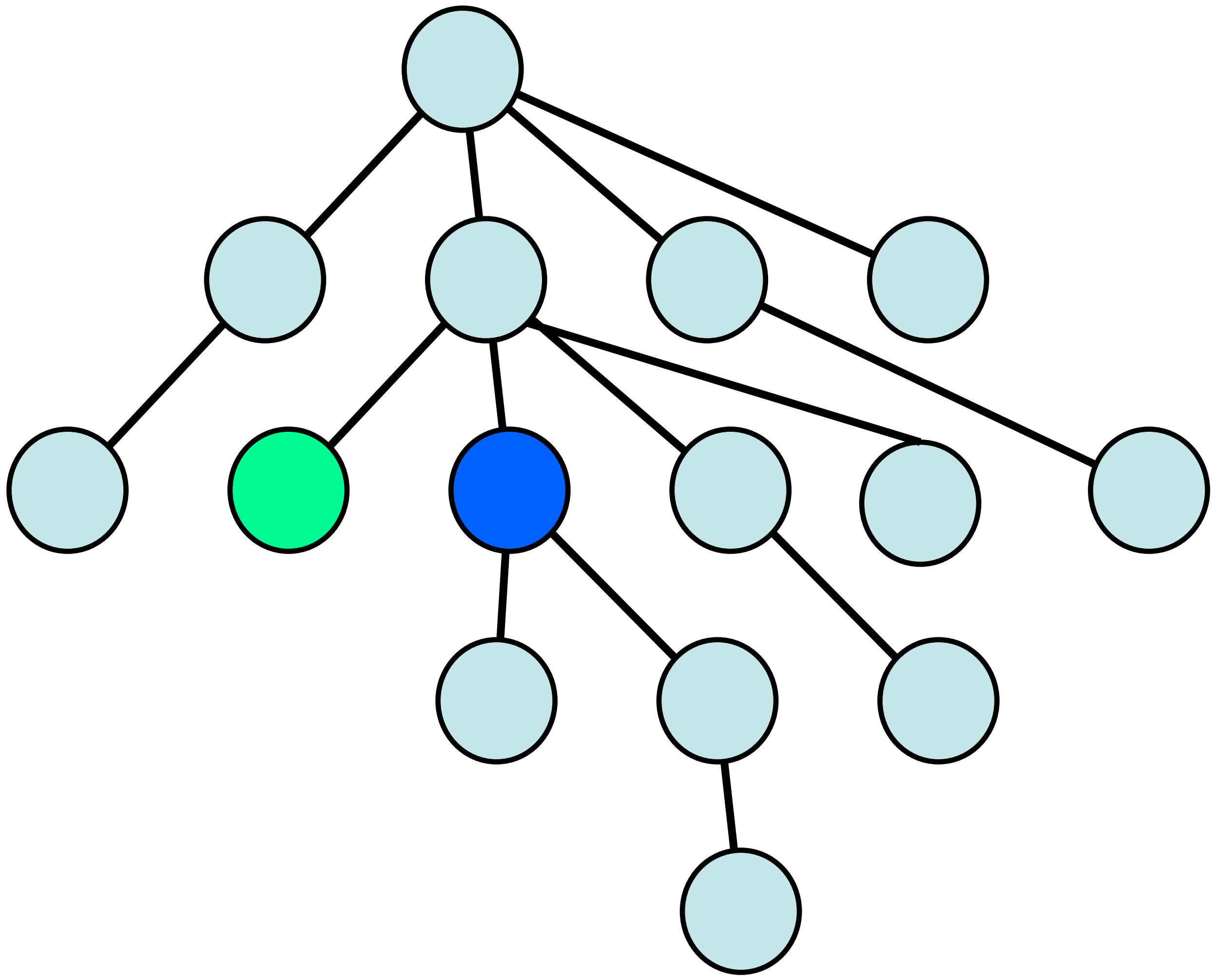
Ancestor-or-self



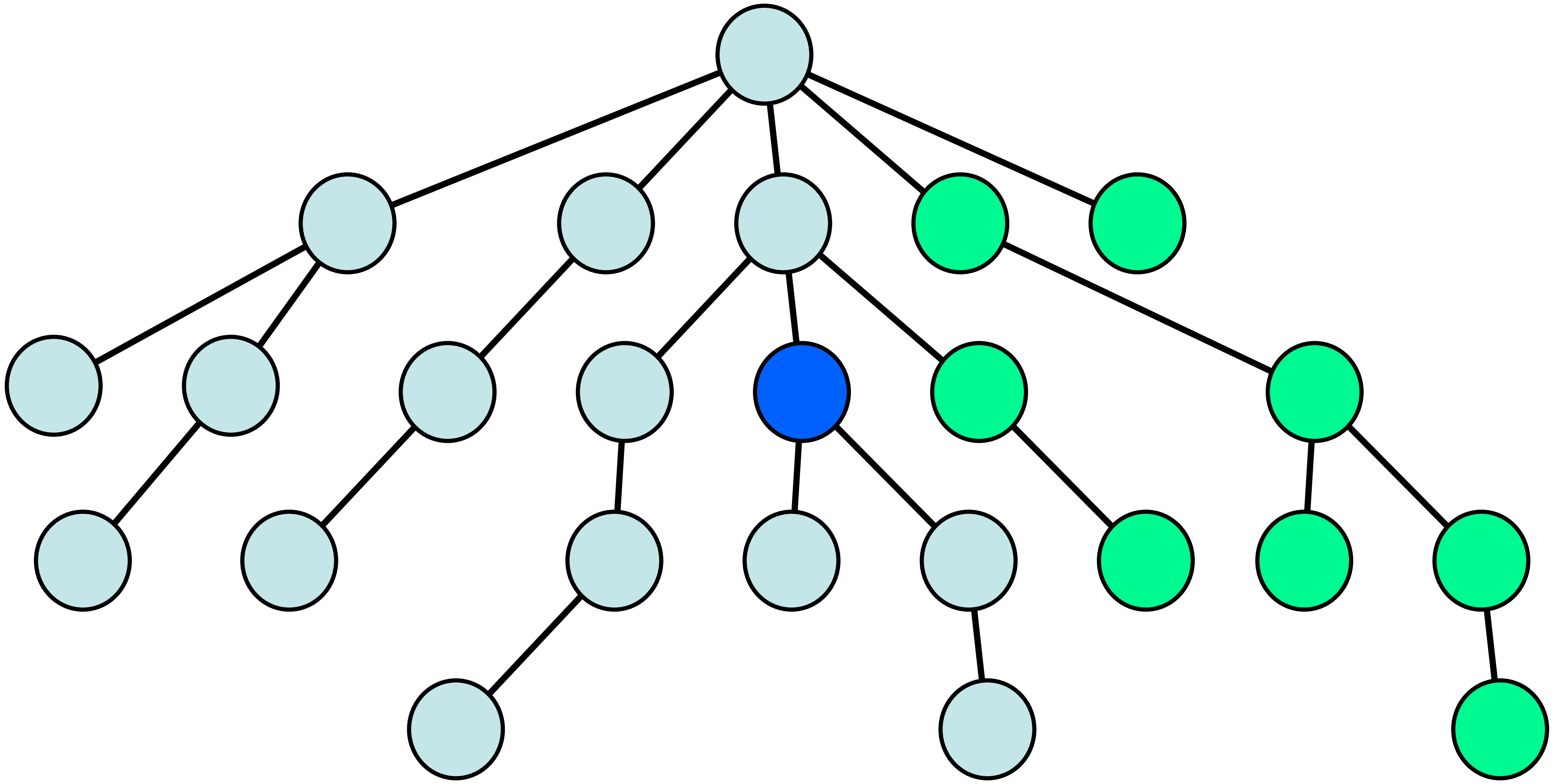
Following-sibling



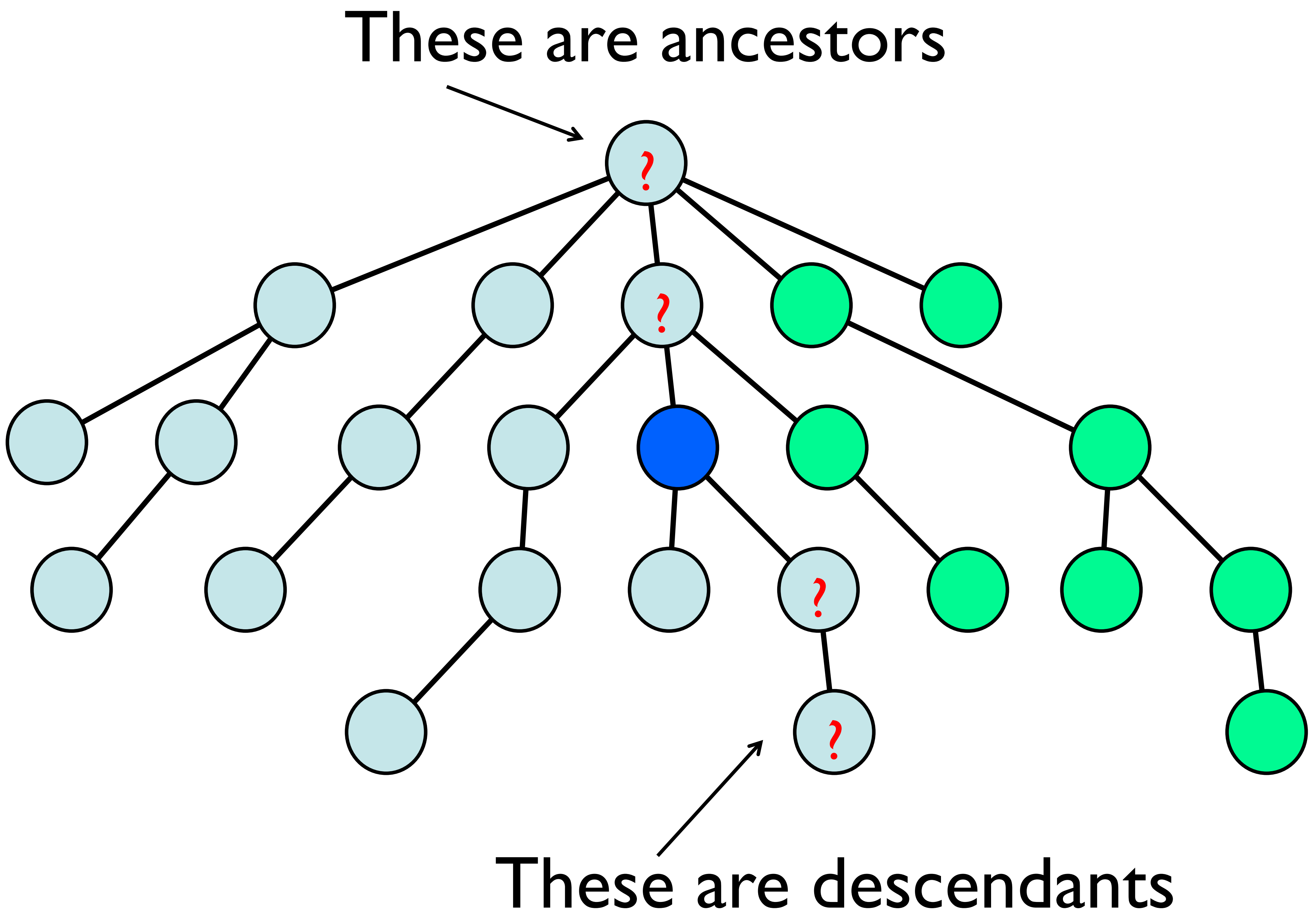
Preceding-sibling



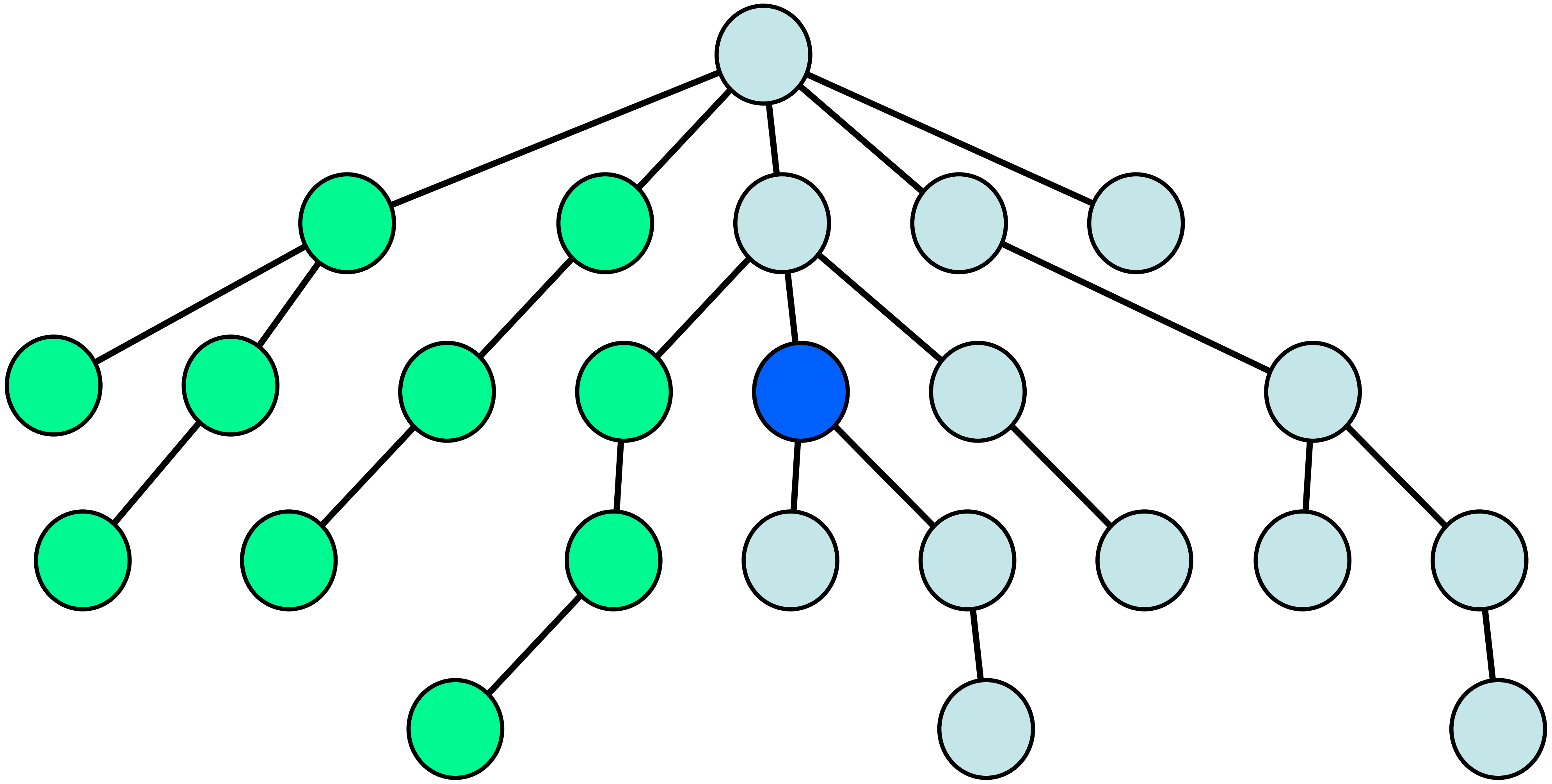
Following



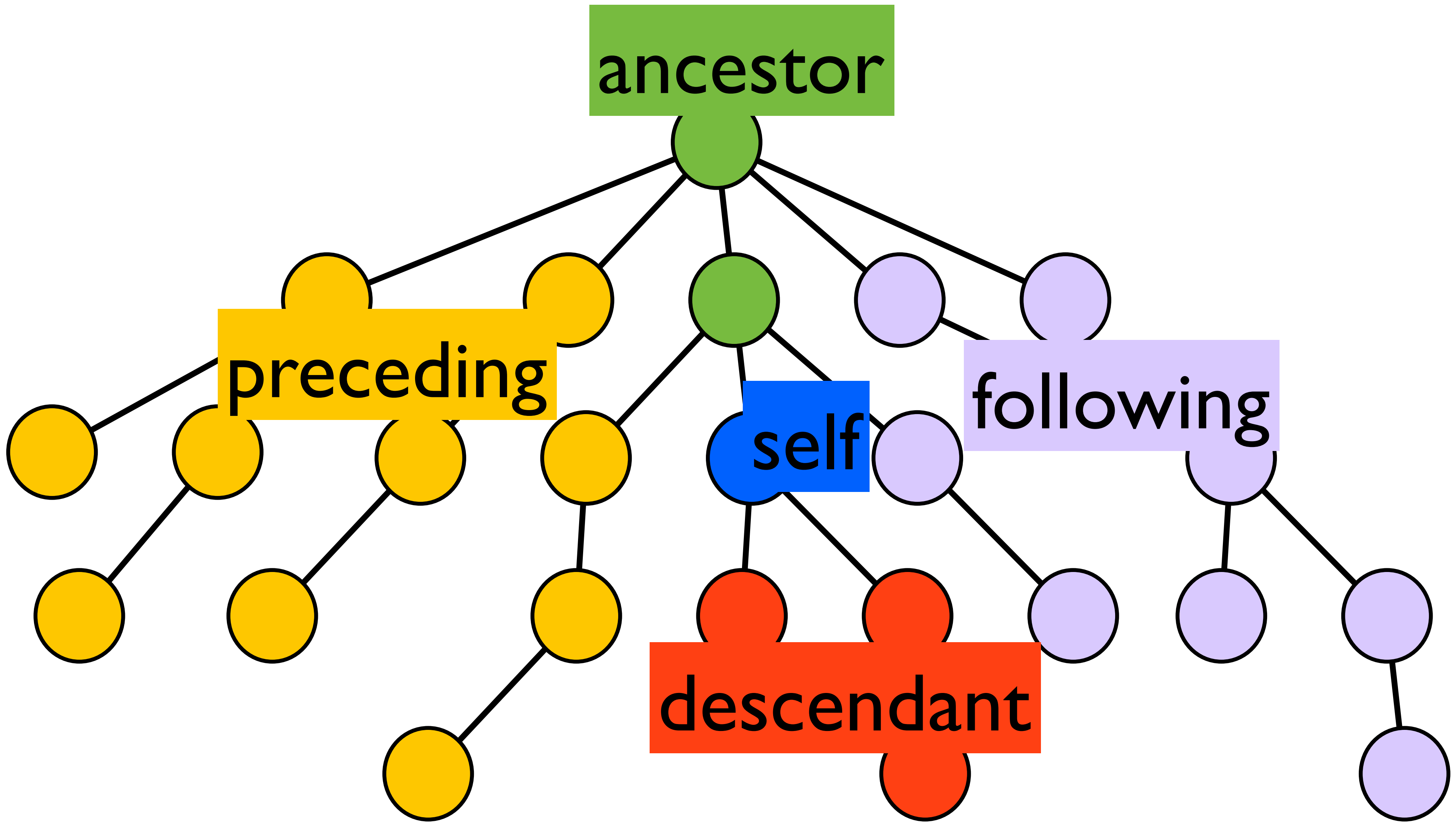
Following



Preceding



Partition

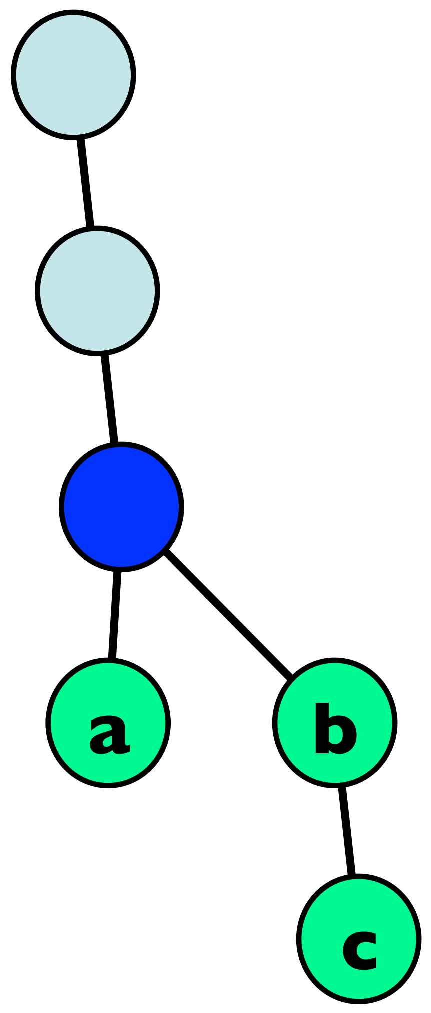


XPath Axes

```
axis  ::= self | child
        | descendant
        | descendant-or-self
        | parent
        | ancestor
        | ancestor-or-self
        | preceding-sibling
        | following-sibling
        | preceding
        | following
```

NODE FILTERS

(2) Node-Test (or Filters)



Do we want all descendants of the blue node ?

(2) Node-Test : four kinds

Input : a node (and the test)   Output : **true** or **false**

(2) Node-Test : four kinds

`node()`

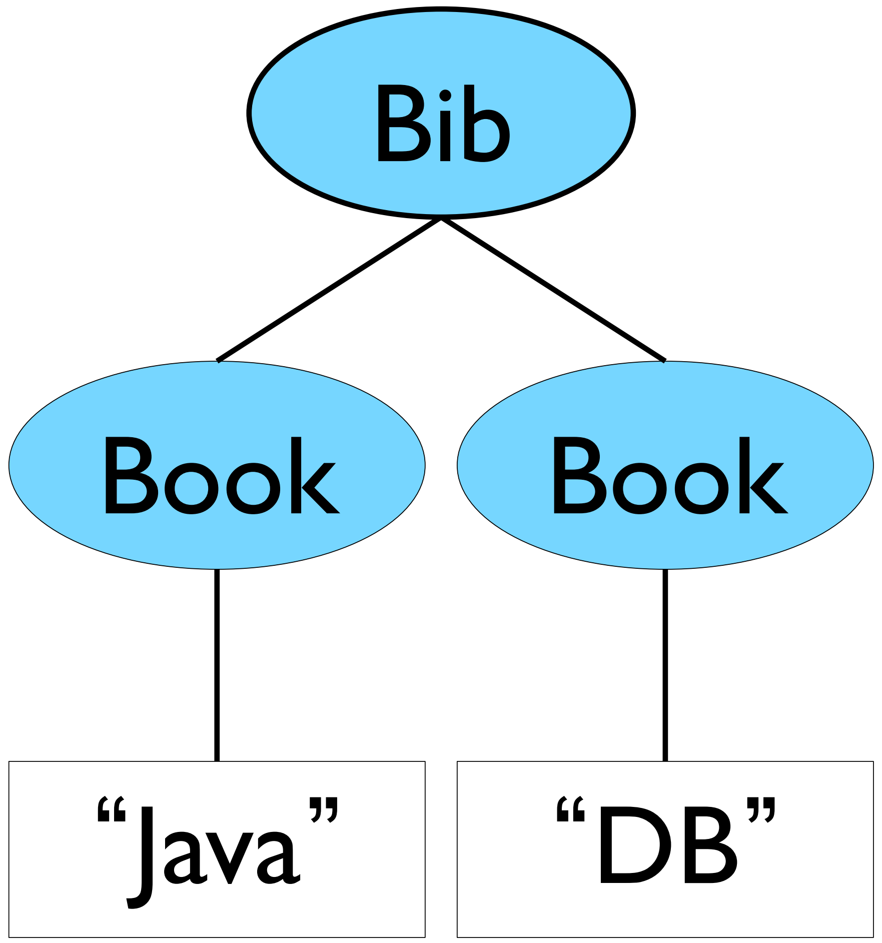
`text()`

`*` (star)

A (tag)

Input : a node (and the test) Output : **true** or **false**

(2) Node-Test : four kinds



`node()`

`text()`

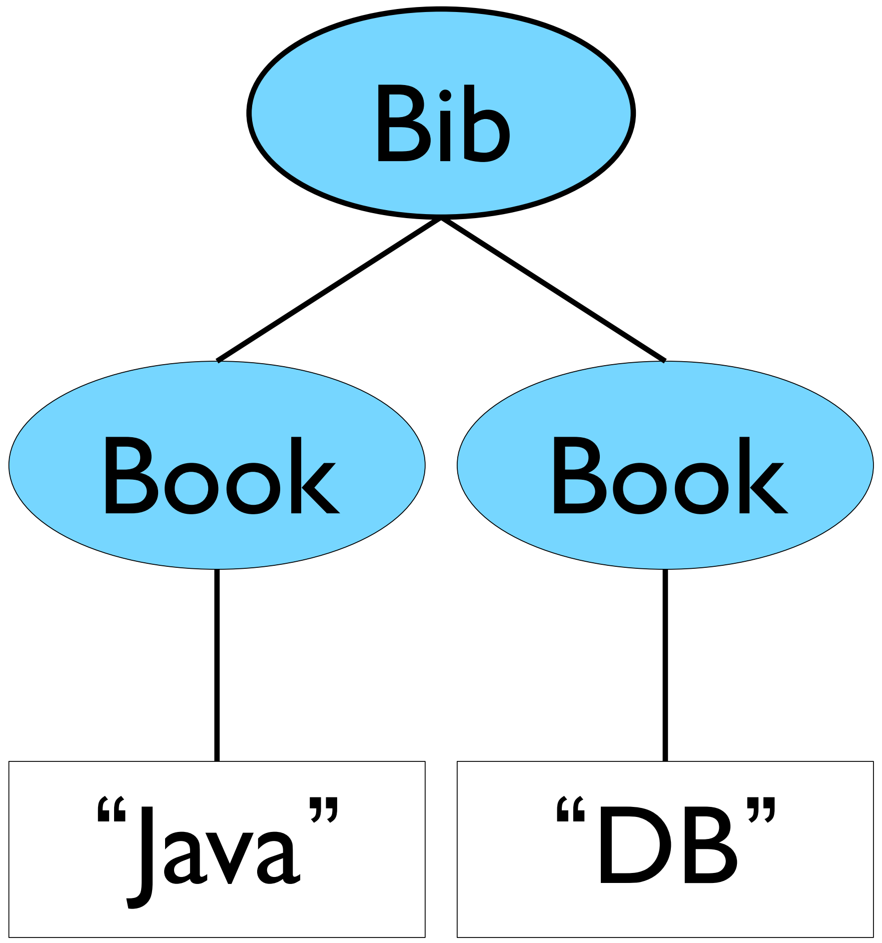
`*` (star)

A (tag)

Input : a node (and the test)

Output : **true** or **false**

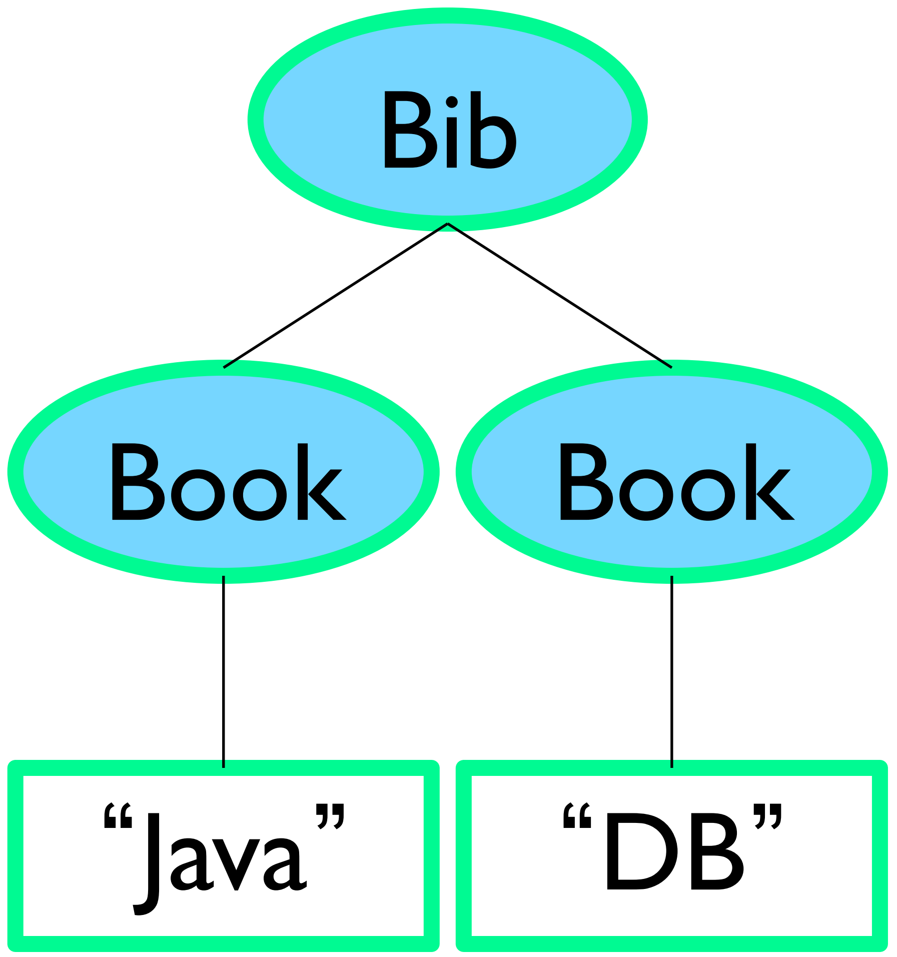
`node( )`



true for element & text nodes

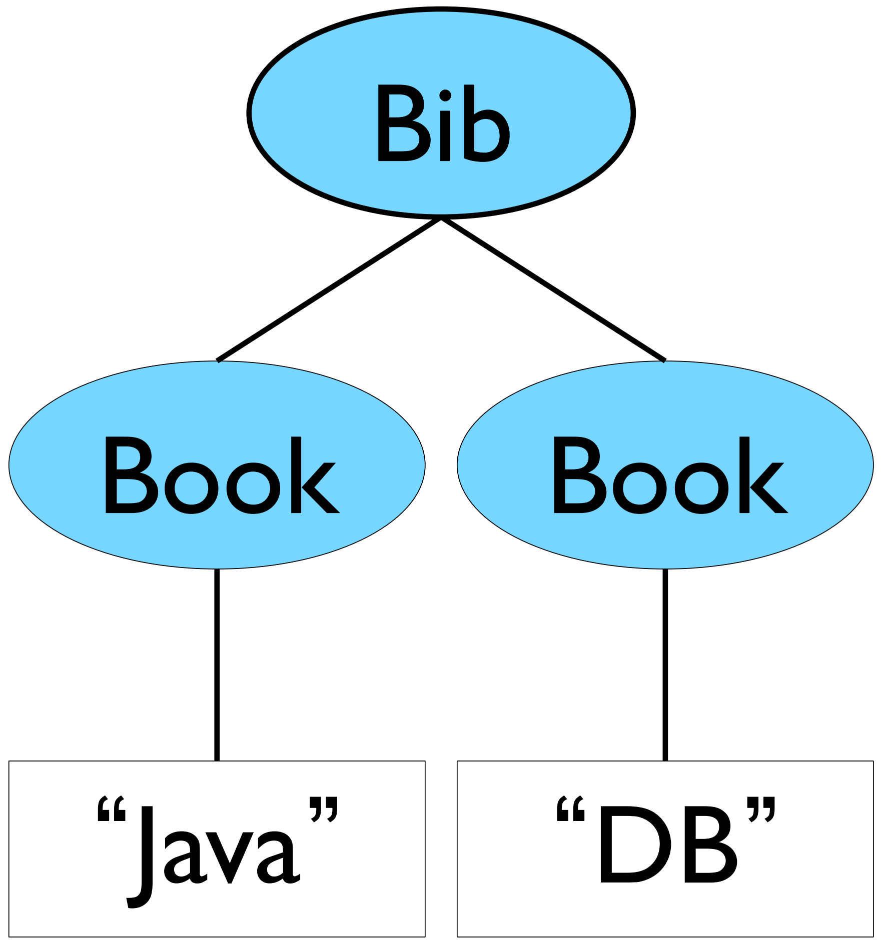
false otherwise (eg. attributes)

`node( )`



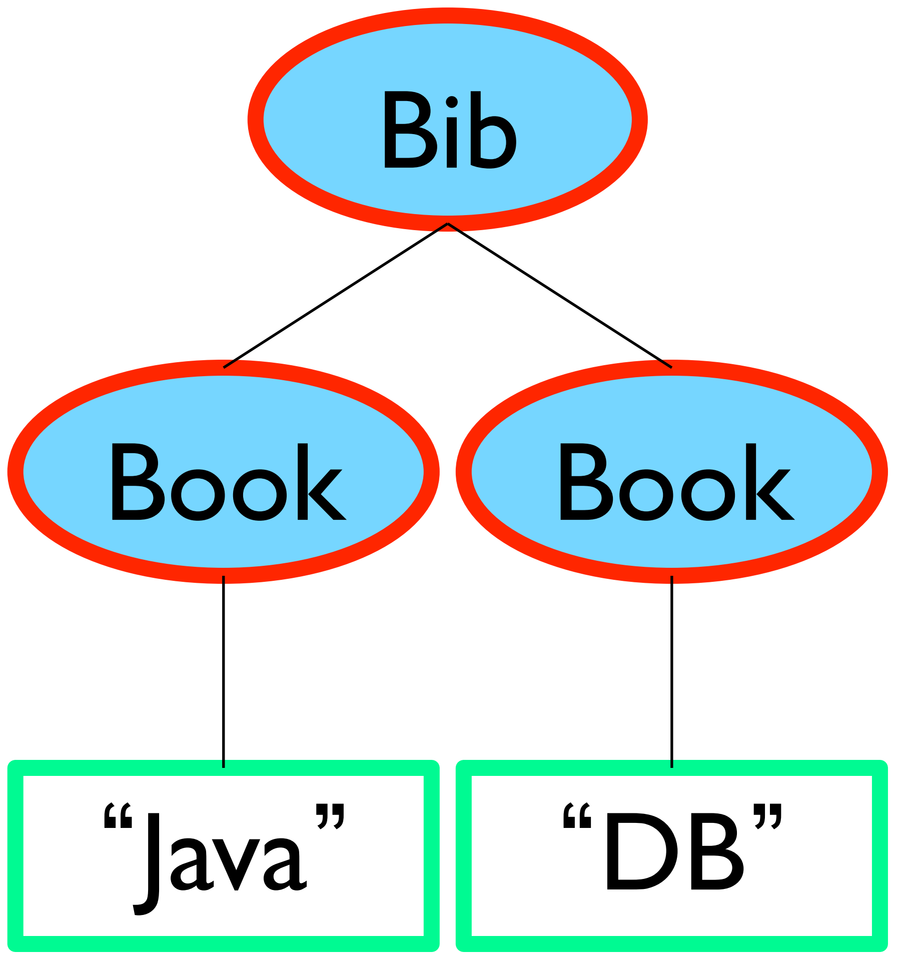
true for element & text nodes
false otherwise (eg. attributes)

`text()`



true for text nodes
false otherwise

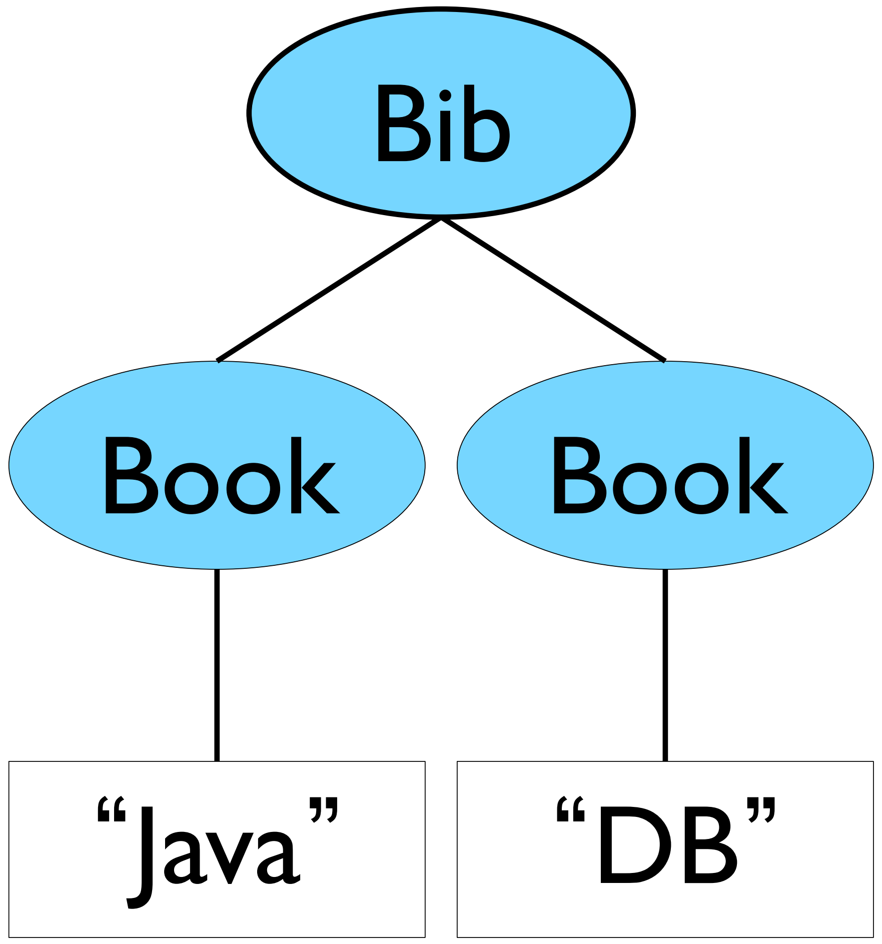
`text()`



true for text nodes

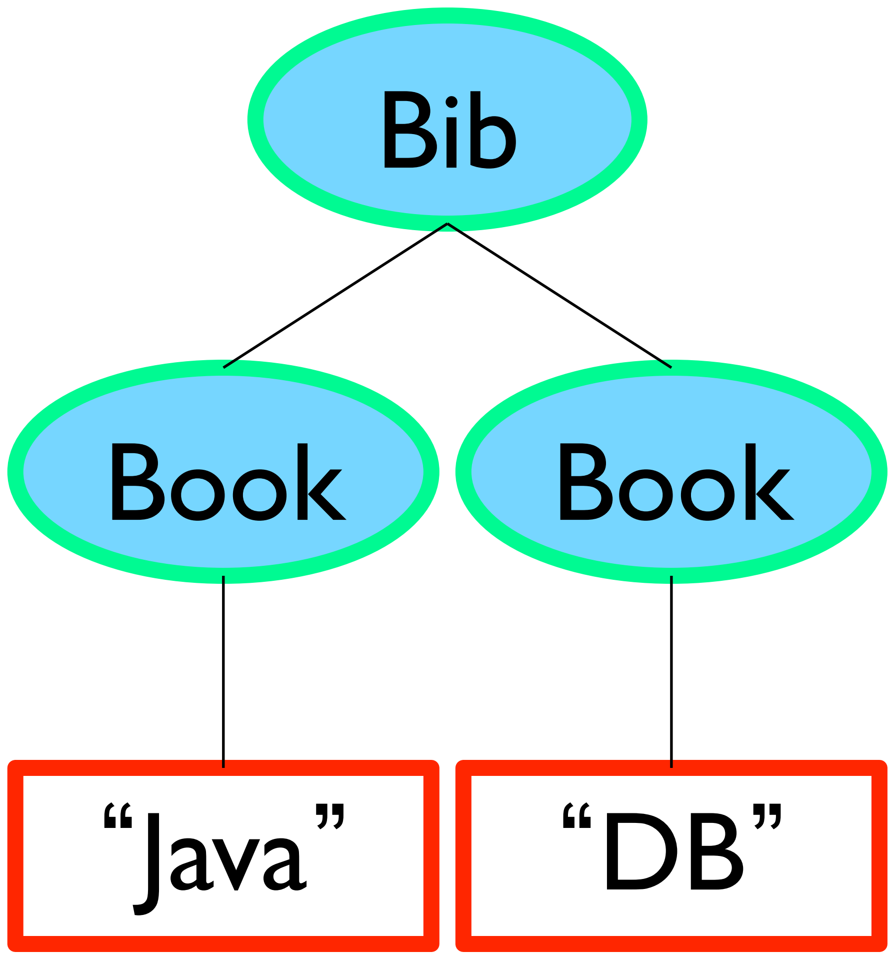
false otherwise

*   (star)



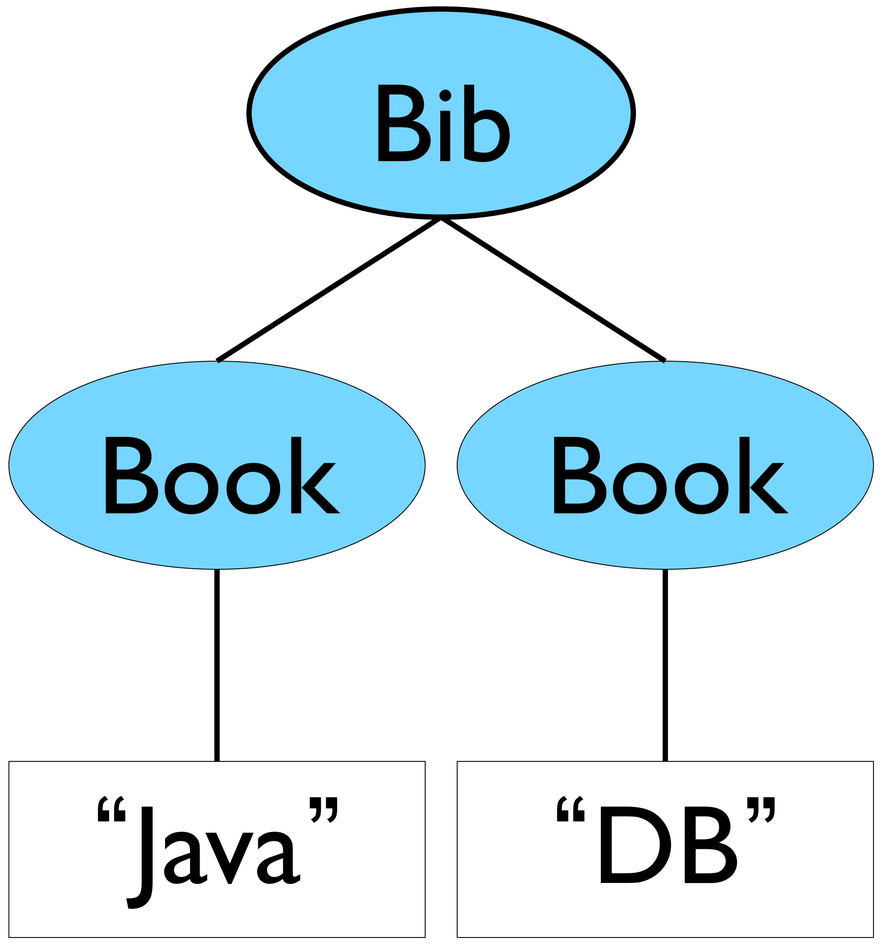
true for element-nodes only
false otherwise (eg. text nodes)

* (star)



true for element-nodes only
false otherwise (eg. text nodes)

(specific tag)

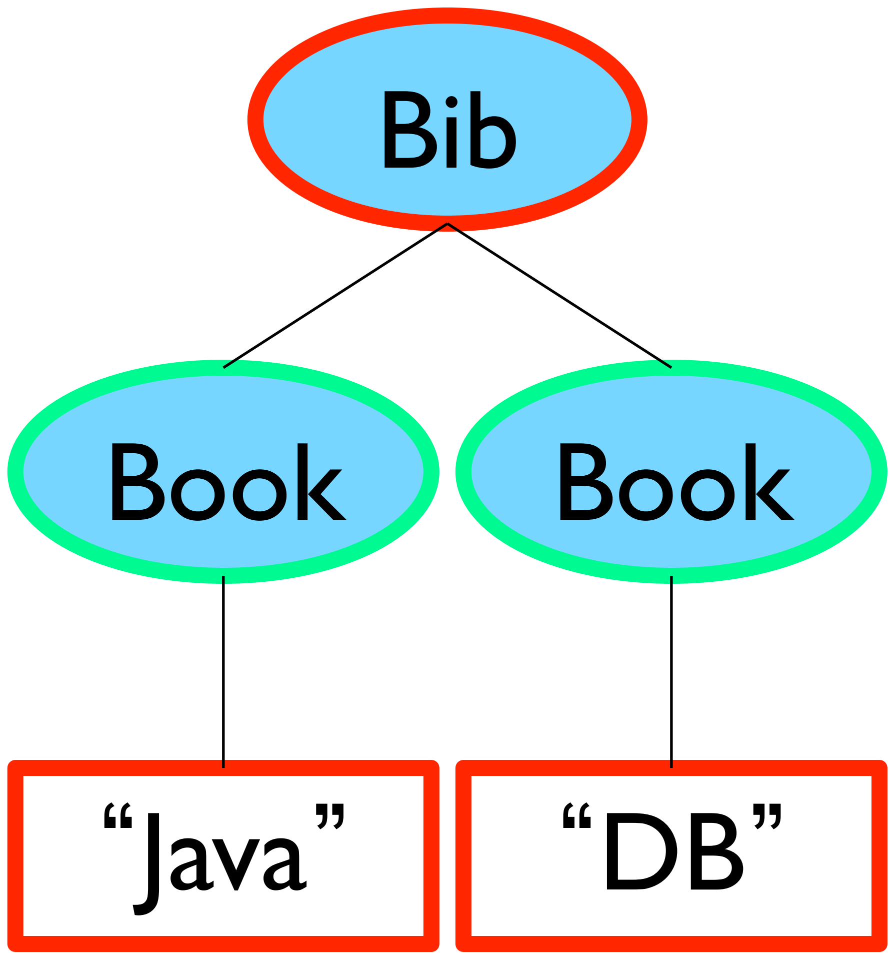


`Book`

true for all element nodes labeled with ‘`Book`’

false otherwise (e.g. `Bib` node)

(specific tag)



`Book`

true for all element nodes labeled with '`Book`'

false otherwise (e.g. `Bib` node)

STEPS AND PATHS

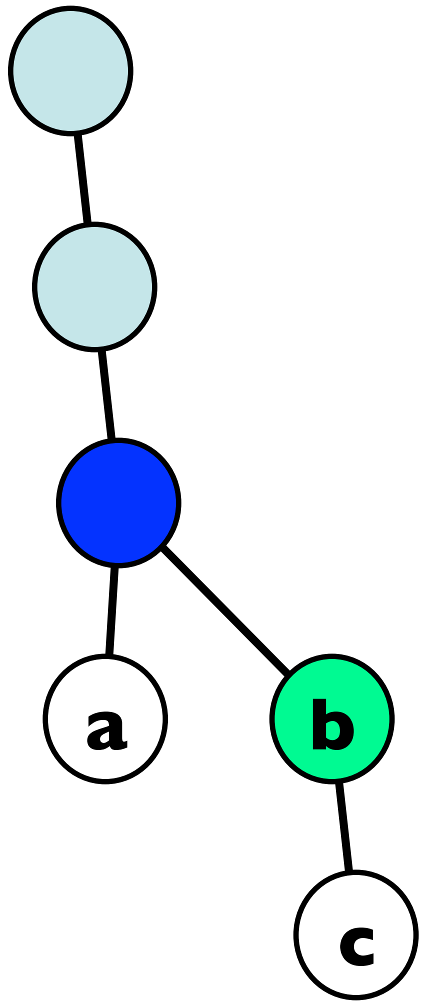
(3) Steps

`axis::test`

Examples

```
child::Book

child::text()          child::*

descendant::node()
```

Example



`child::b`

(4) Paths

/ $\texttt{step}_1$ / $\texttt{step}_2$ … / $\texttt{step}_n$

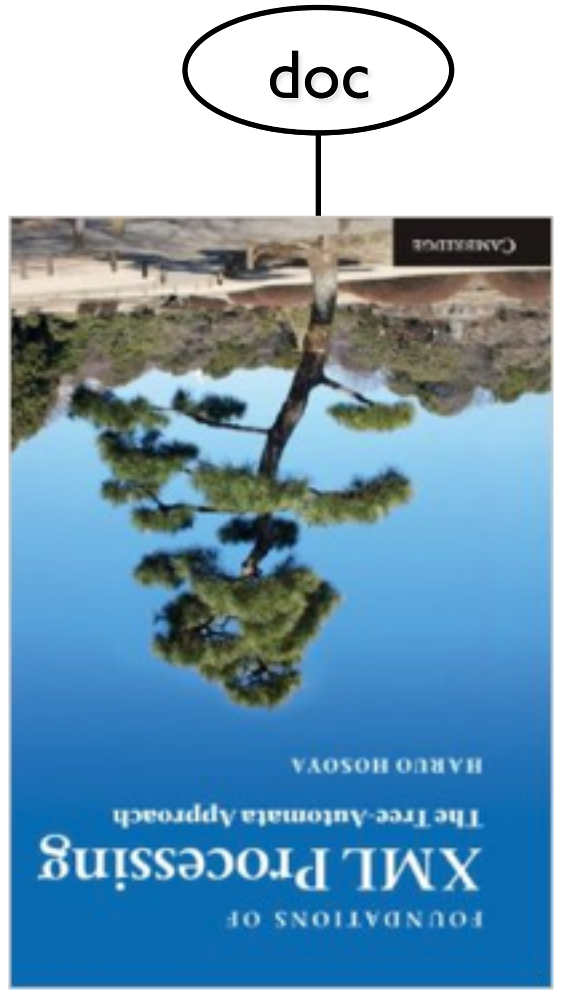
Examples

```
/child::Bib/child::Book

/descendant::text()/parent::Book

/self::node()/self::Bib
```

/

Starting a path with / will select the “document node” (using XPath 3.0 terminology)

- a “virtual” node above the root of the treee



CONDITIONAL NAVIGATIONS

(5) Conditional Navigation

```
step ⟦ condition ⟧
```

Examples

```
/descendant::Book⟦ child::text() ⟧

/descendant::Book⟦ descendant::year ⟧

/descendant::node()⟦ name() = ‘Bib’ ⟧
```

(5) Conditional Navigation

```
step [ condition ]
```

Examples

```
/descendant::Book[ child::text() ]
/descendant::Book[/descendant::year ]
/descendant::node()[ name() = 'Bib' ]
```

(5) Conditional Navigation

```
step [ con
```

The slash "/" means that we are starting again from the document node

Examples

```
/descendant::Bo                      () ]

/descendant::Book[/descendant::year ]

/descendant::node()[ name() = ‘Bib’ ]
```

(5) Conditional Navigation

```
/descendant::Book[/descendant::year]
```

A bit weird : this expression says that all nodes labelled with `Book` are selected if somewhere in the document a node labelled with `year` exists

(5) Conditions

```
condition::=  path | function
            | condition and condition
            | condition or condition
            | not(condition)

function ::= count() | name() |
            |   position() | last()
```

Example

```
child::*[ name()='Book' and position()=1 ]
```

ABBREVIATIONS

XPath : Abbreviations

`/a`	⇔	`child::a`
`.`	⇔	`self::node()`
`..`	⇔	`parent::node()`

XPath : Abbreviations

`//a`

$\Leftrightarrow$

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

- very similar (but ≠) to `descendant::a`

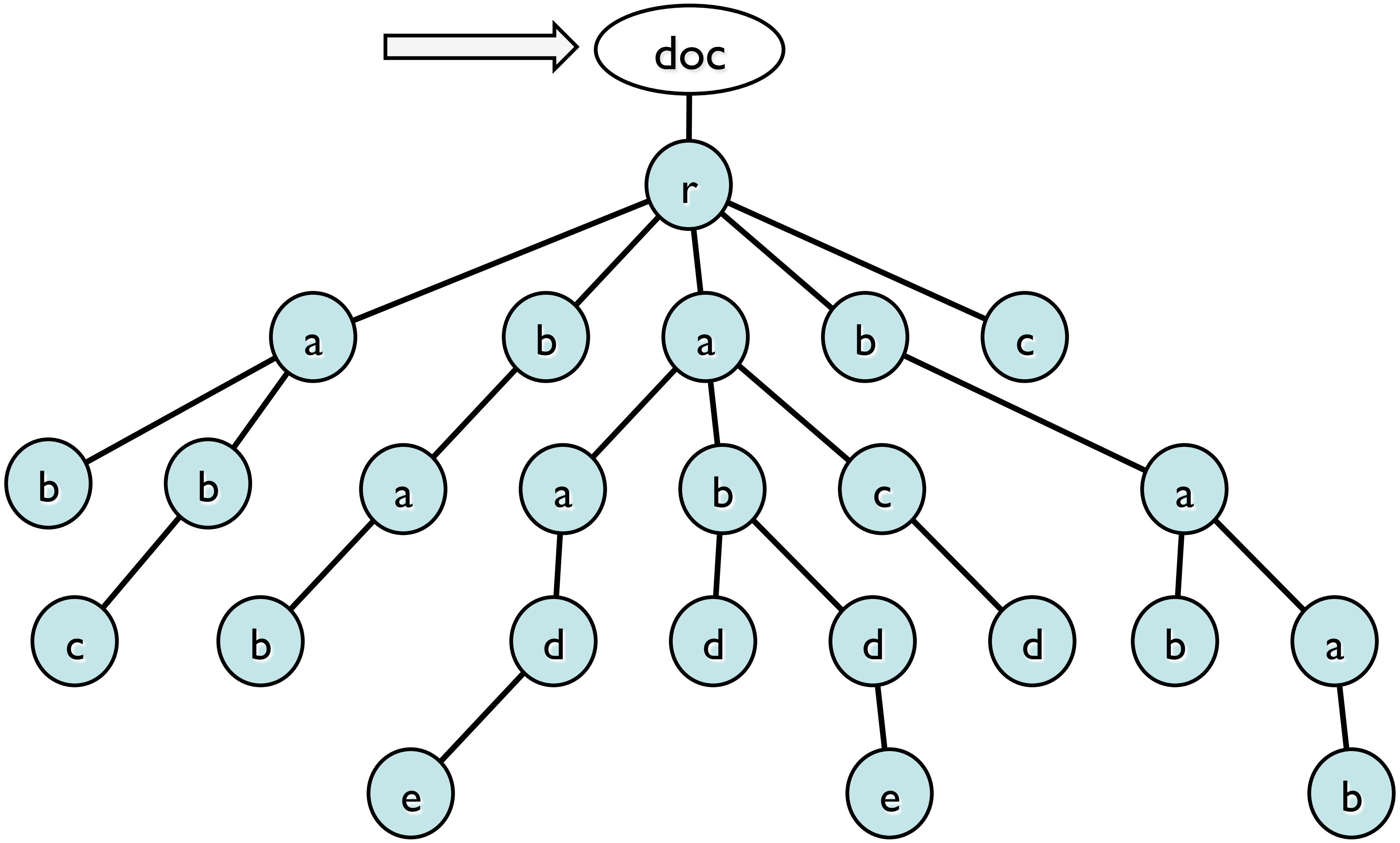
XPath Evaluation

INPUT : XML tree and an XPath expression

OUTPUT : a list of (sub-)trees

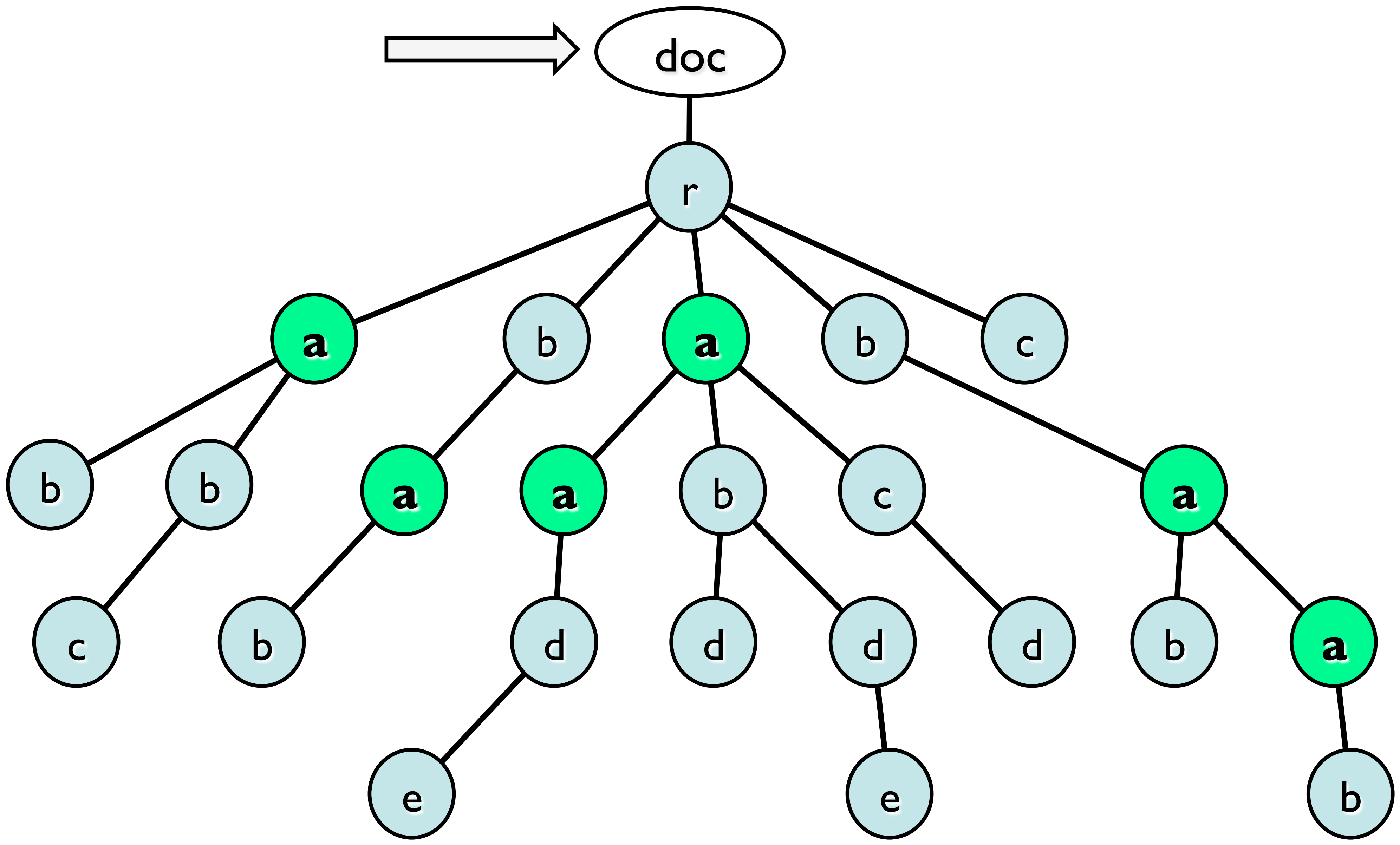
→ one for each selected node !

Single Step Navigation



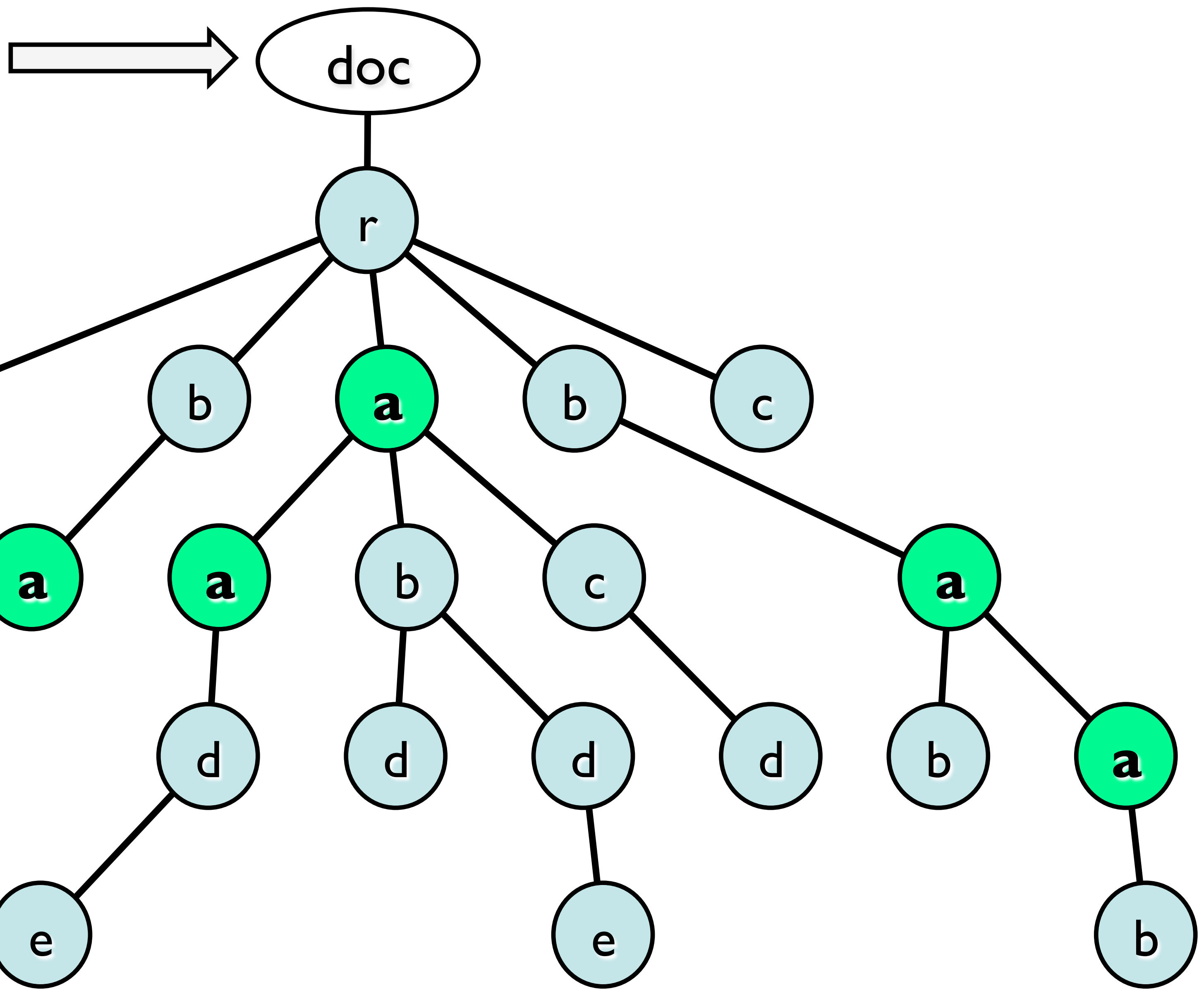
`/descendant::a`

Single Step Navigation



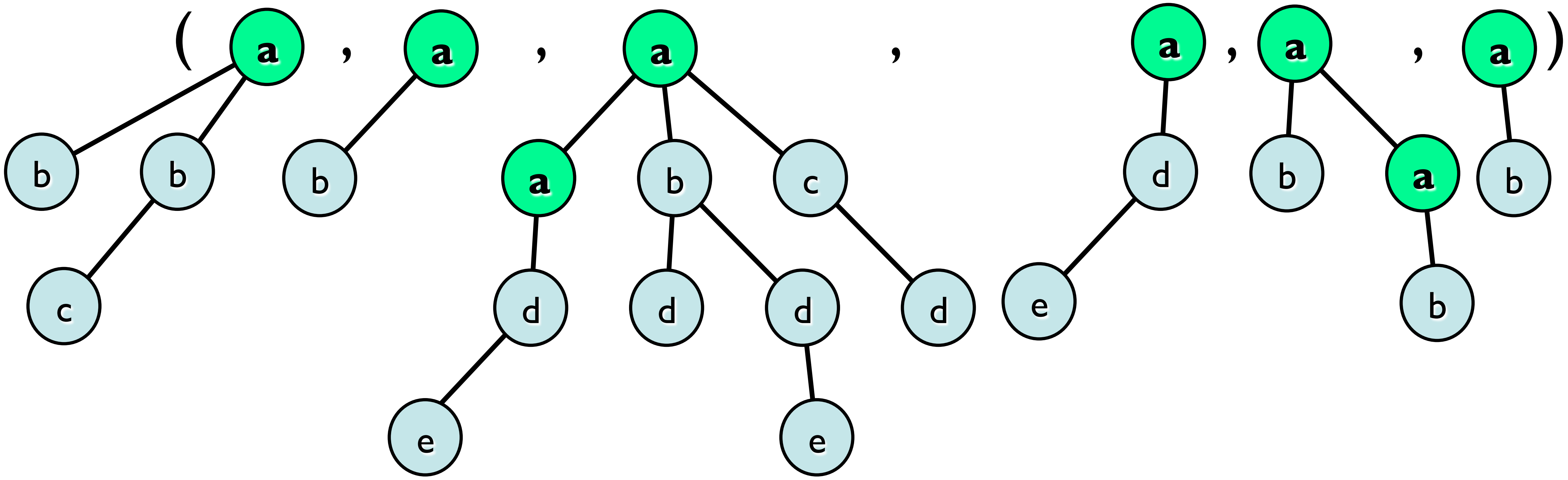
```
/descendant::a
```

Output :



`/descendant::a`

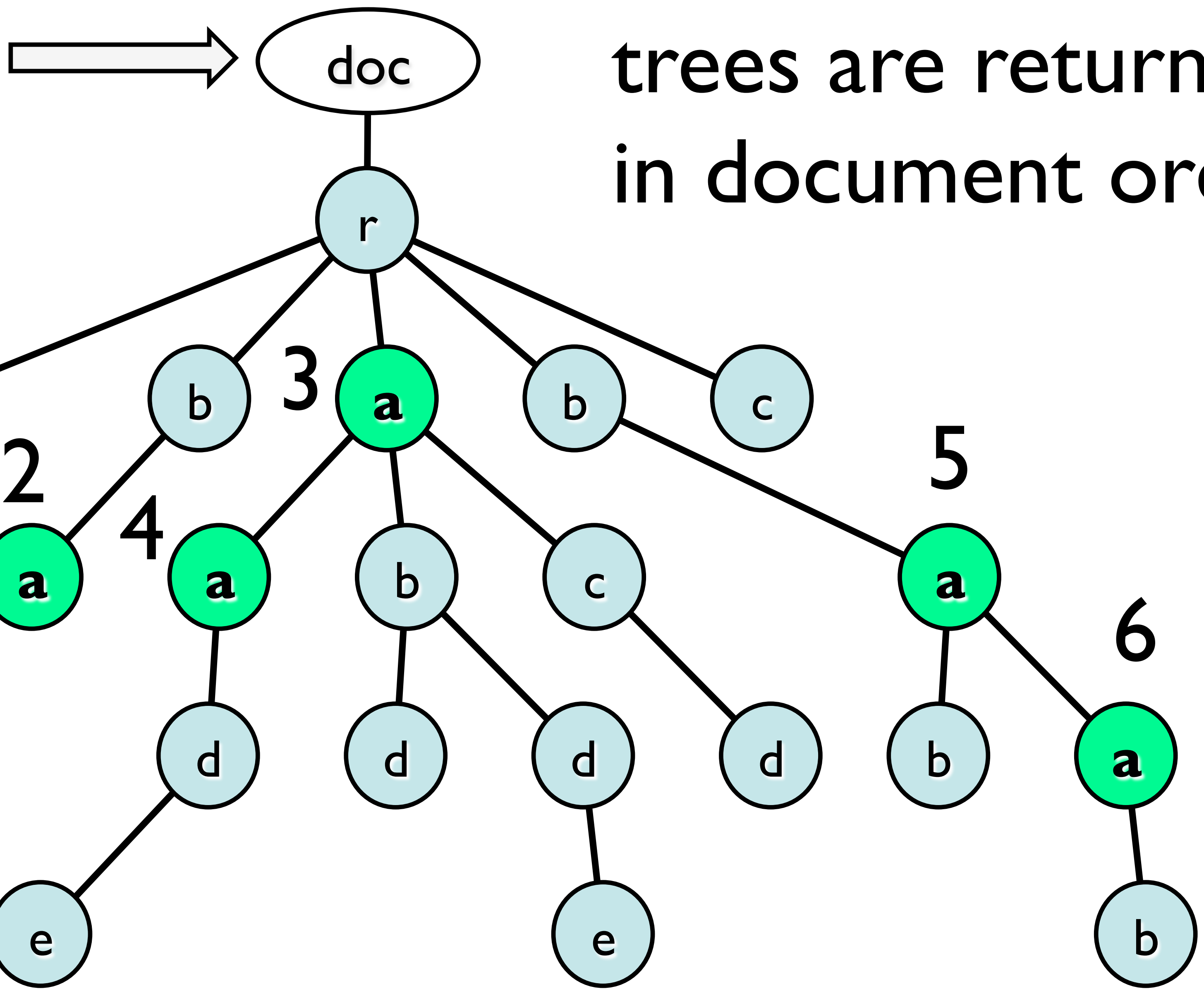
Output : Sequence of subtrees



(a , a , a , a , a , a)
b b c b a b c d d d e d d e d b a b b b

```
//descendant::a
```

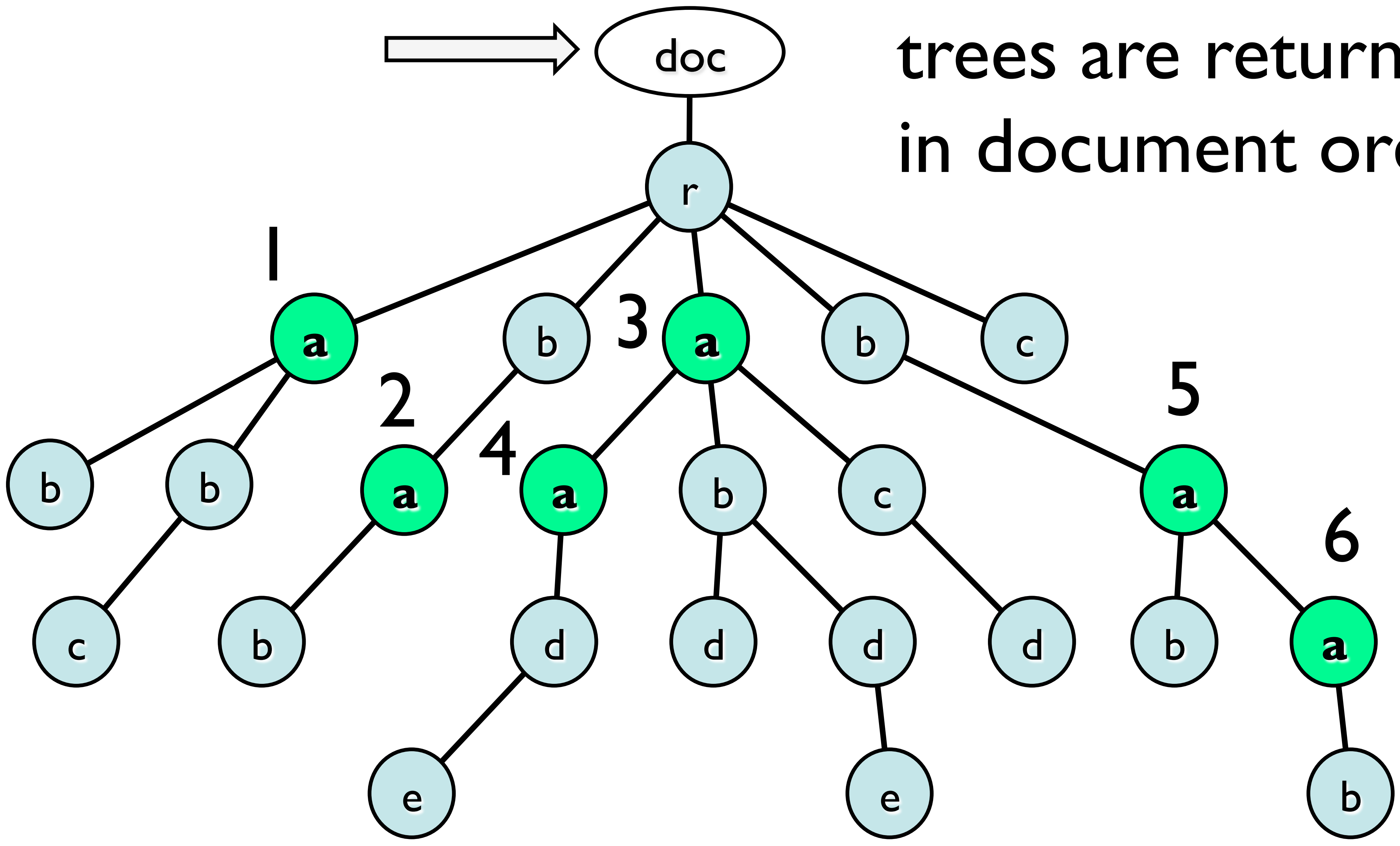
Output : Sequence of subtrees



trees are returned
in document order

`/descendant::a`

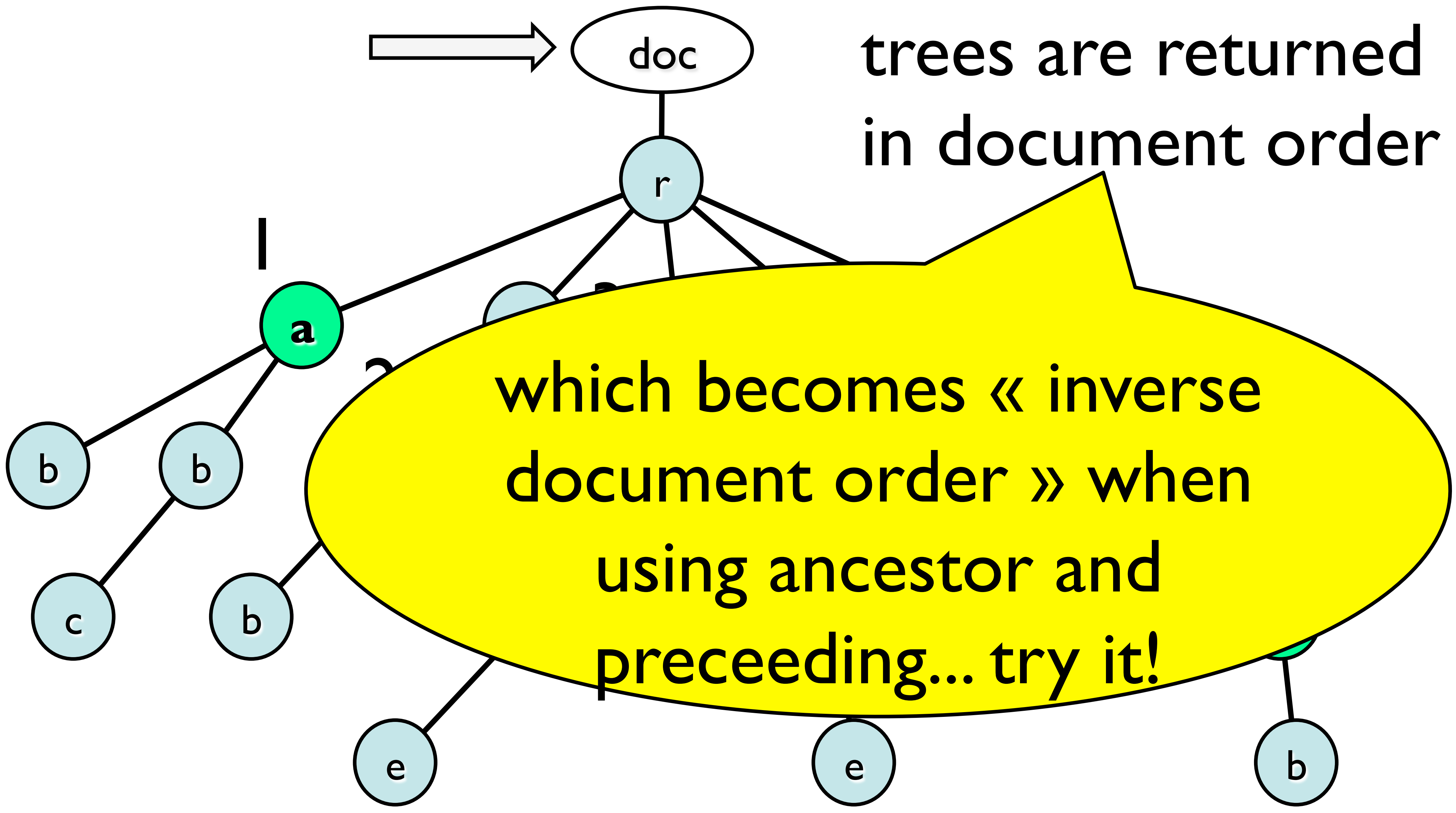
With a given order



trees are returned
in document order

`/descendant::a`

With a given order

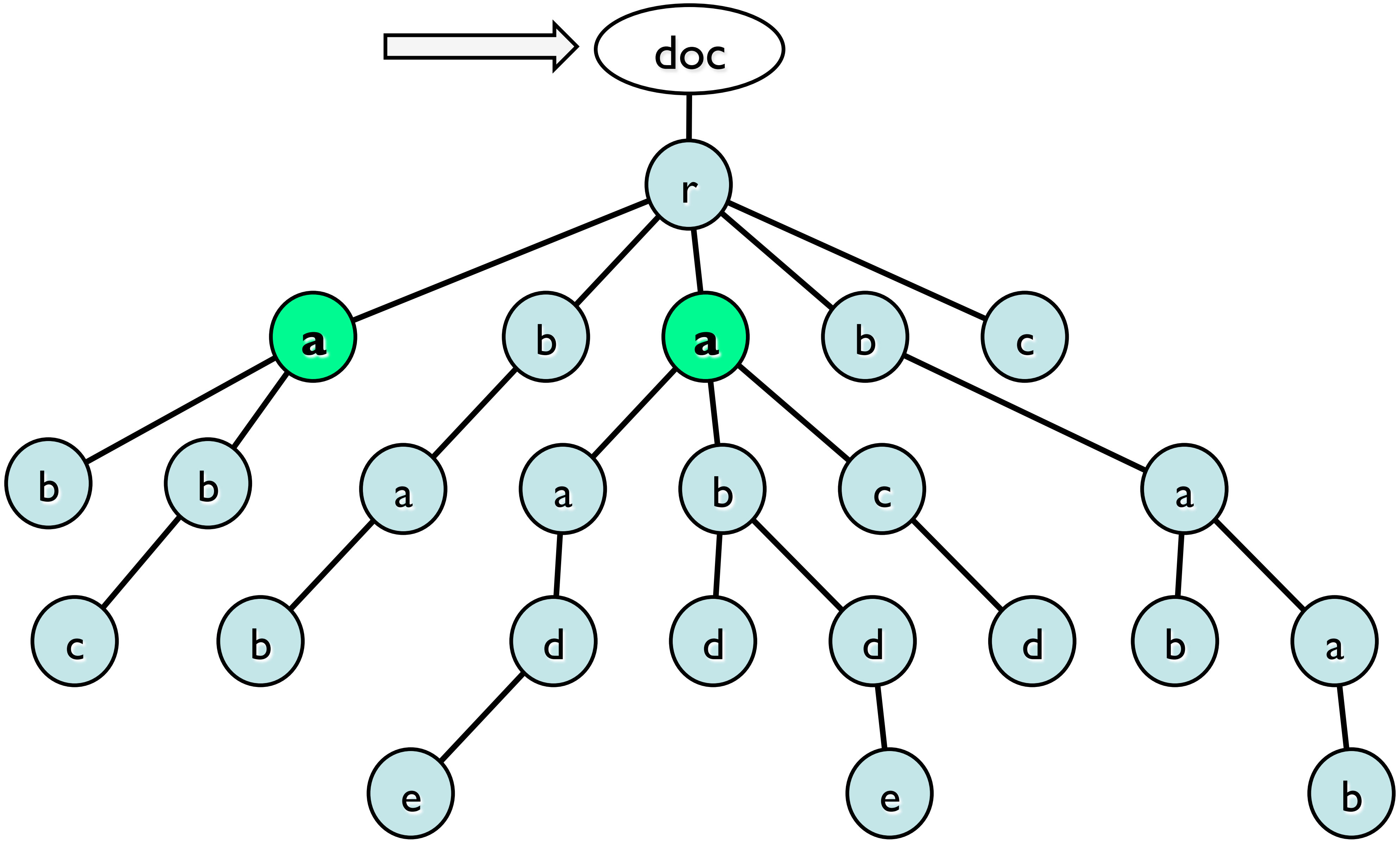


trees are returned in document order

which becomes « inverse document order » when using ancestor and preceeding... try it!

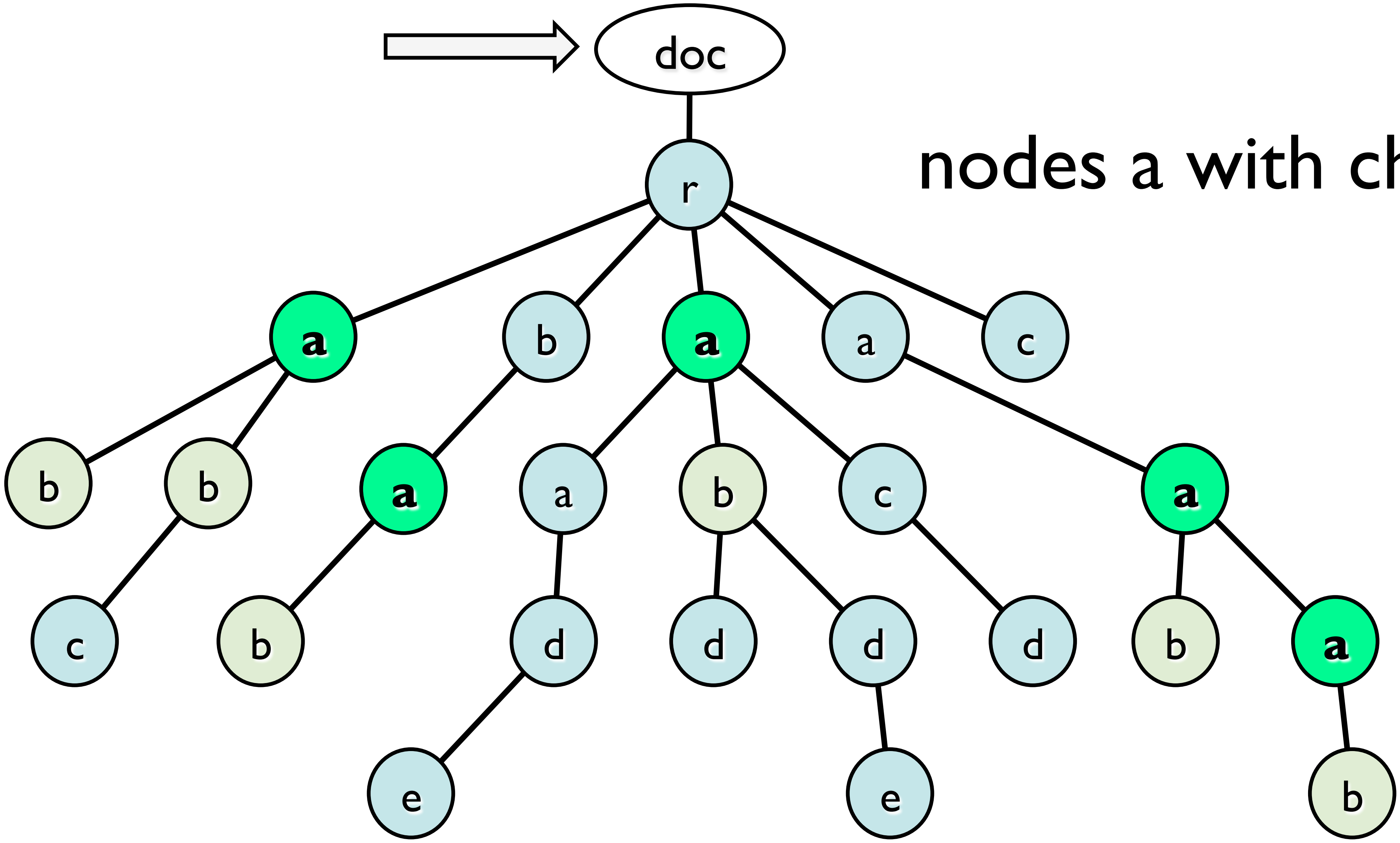
```
/descendant::a
```

Example (I)



`/child::r/child::a`

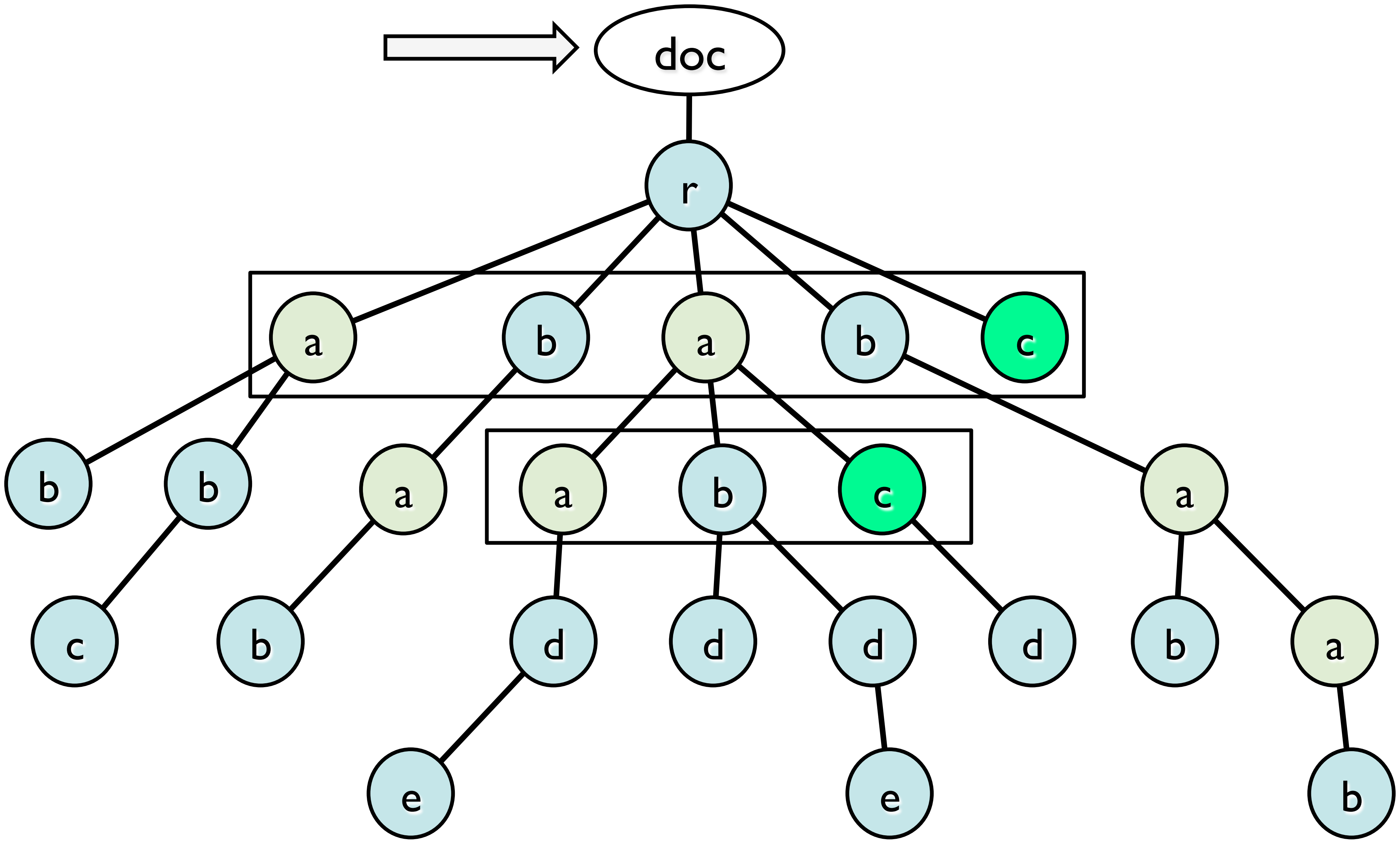
Example (2)



nodes a with child b

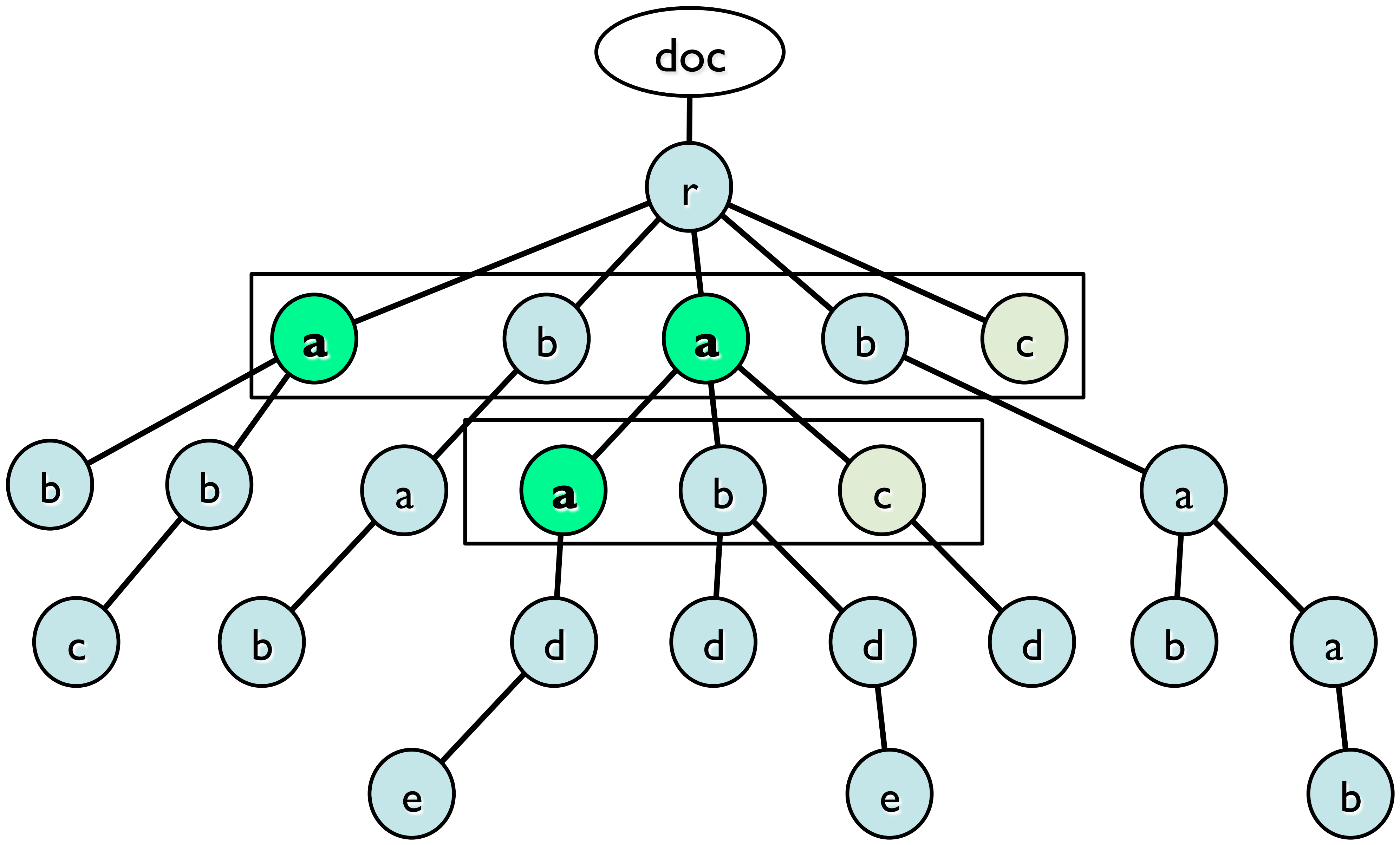
`//a[b]`

Example (3)



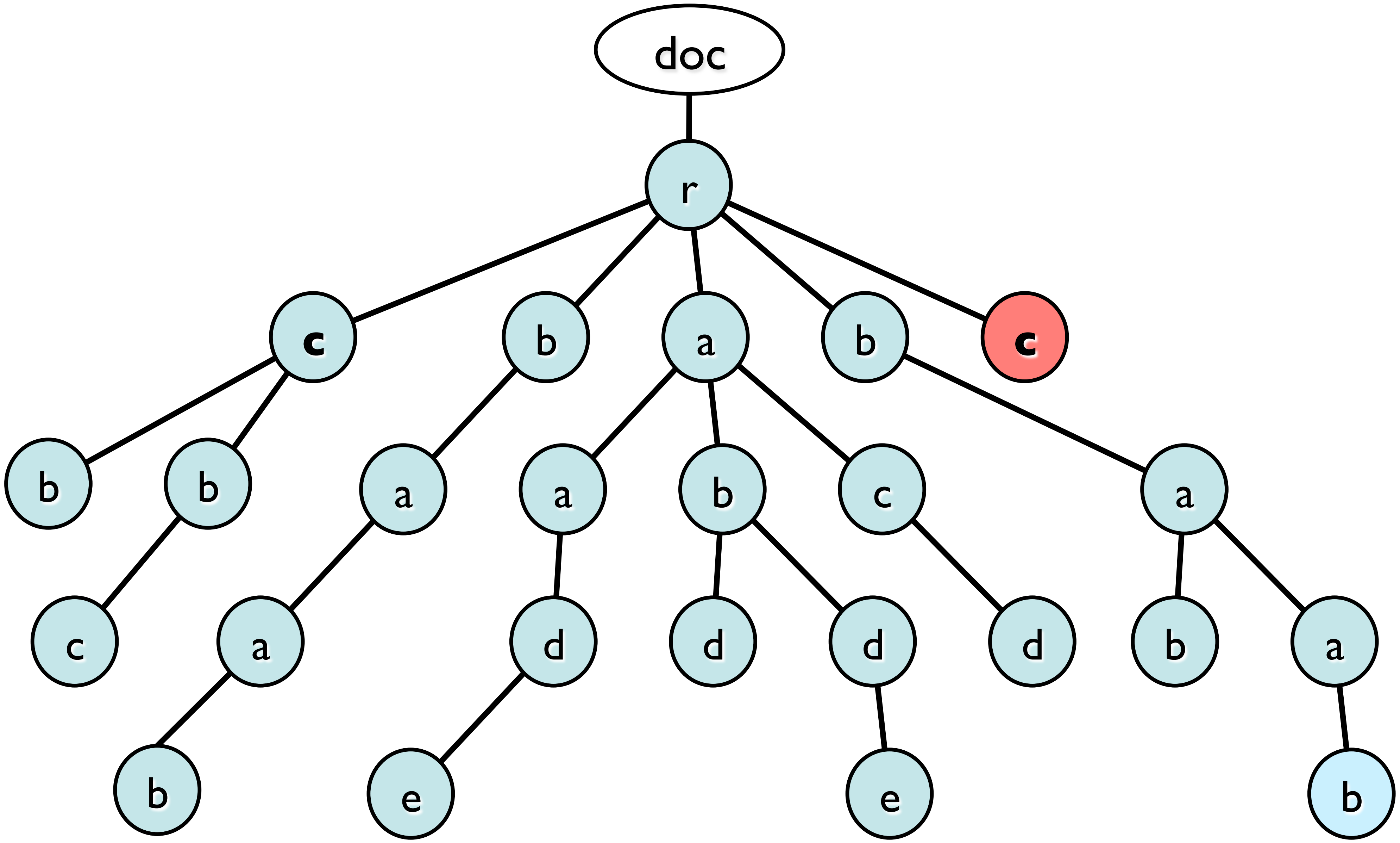
`/descendant::a/following-sibling::c`

Example (4)



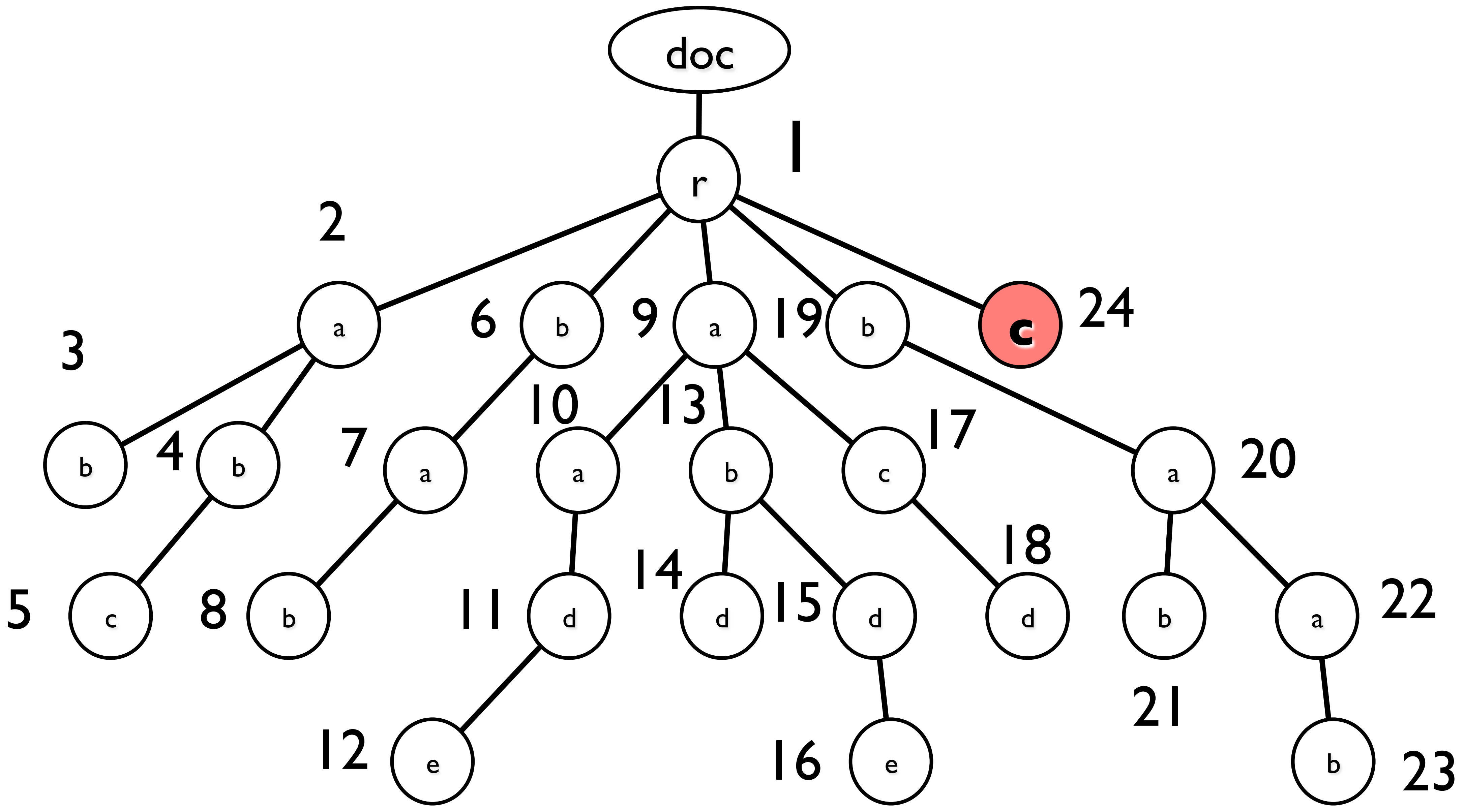
`//a[following-sibling::c]`

Quiz : select the red node



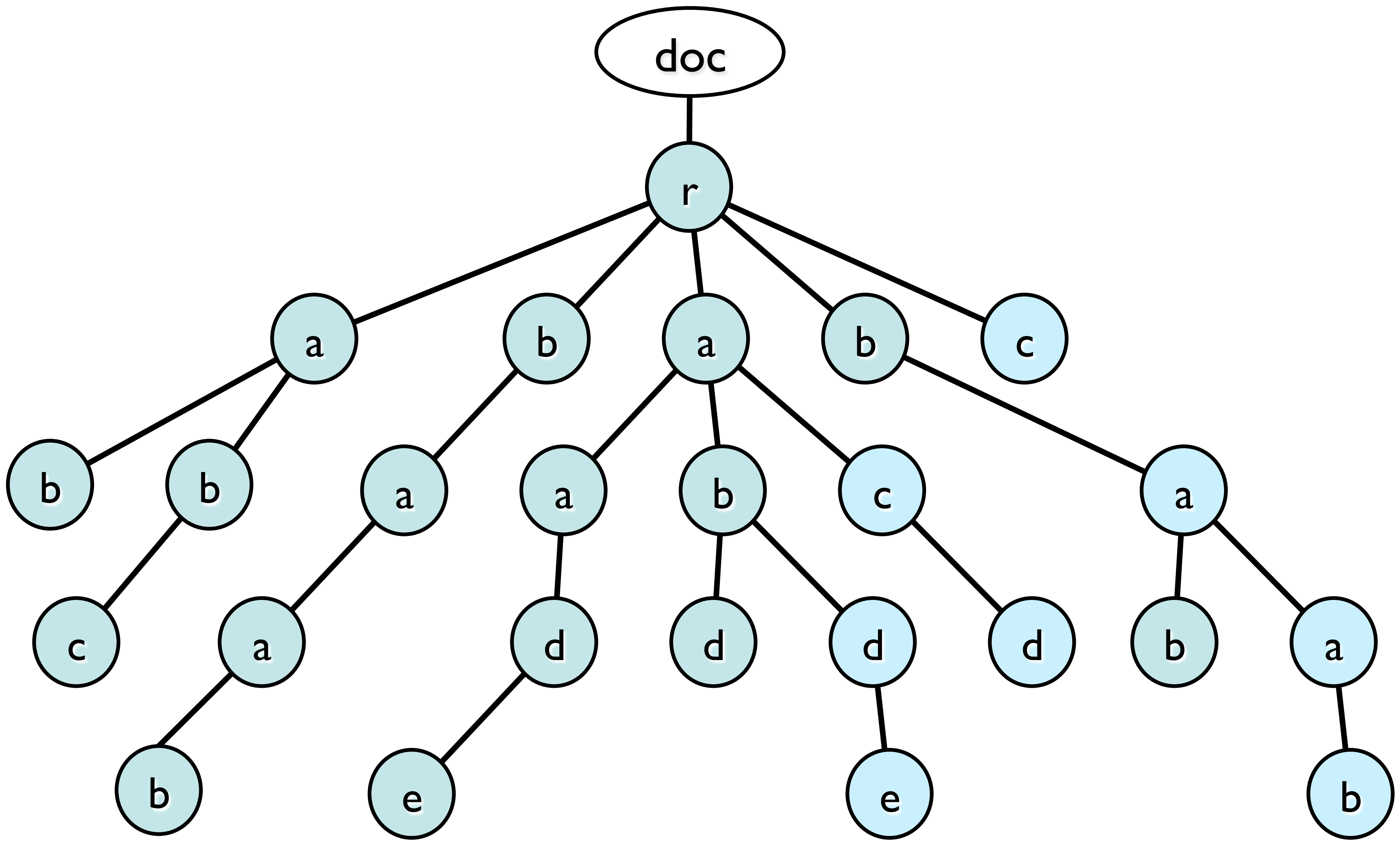
```
/descendant::*[last()]
```

Quiz : select the red node



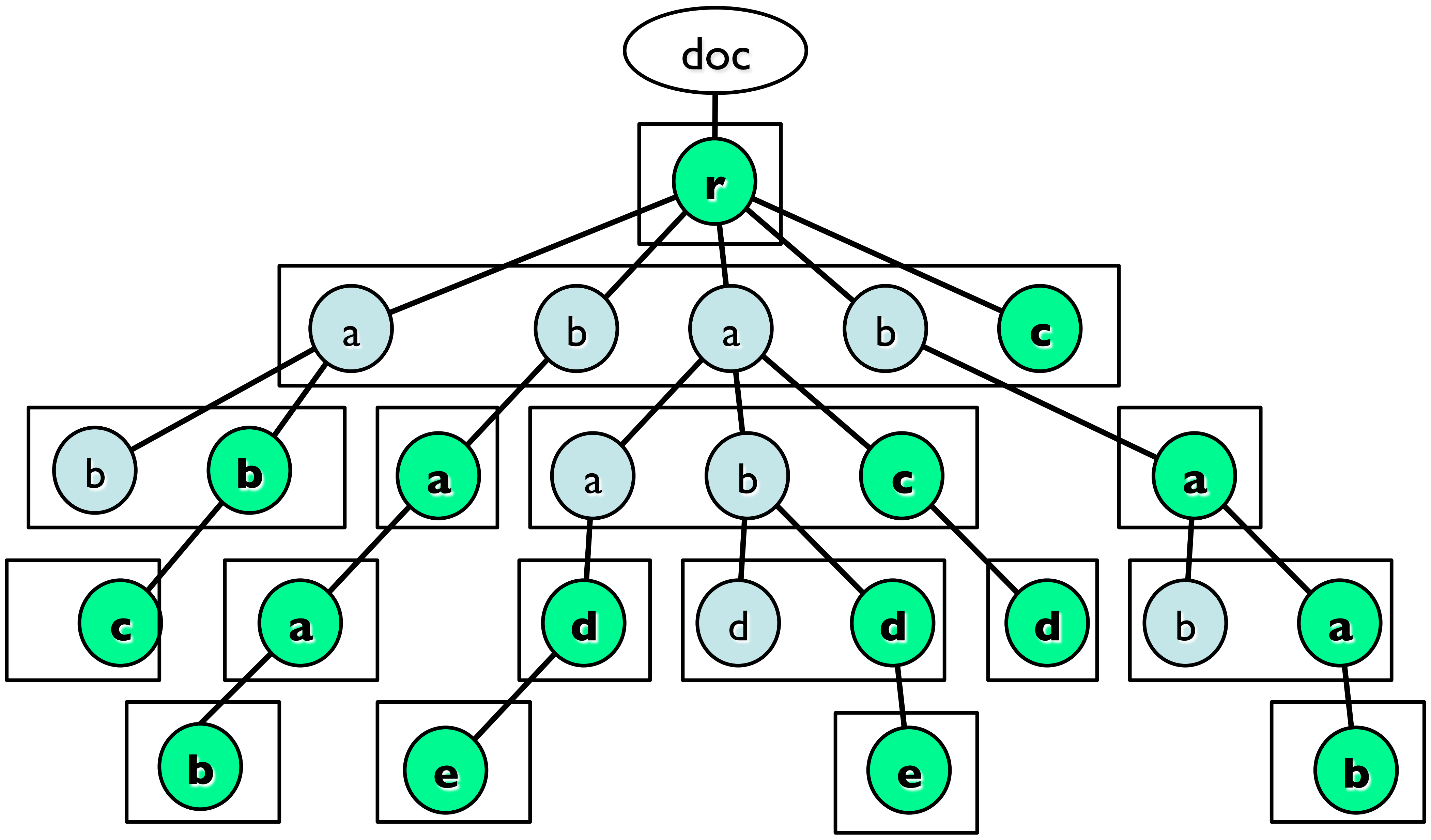
```
/descendant::*[last()]
```

Quiz 2 : `//*[last()]`



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

Quiz 2 : `//*[last()]`



`/descendant-or-self::*/child::*[last()]`

“//” vs. “/descendant”

```
/descendant::node()[b]
```

=

```
(/descendant-or-self::*/child::node())[b]
```

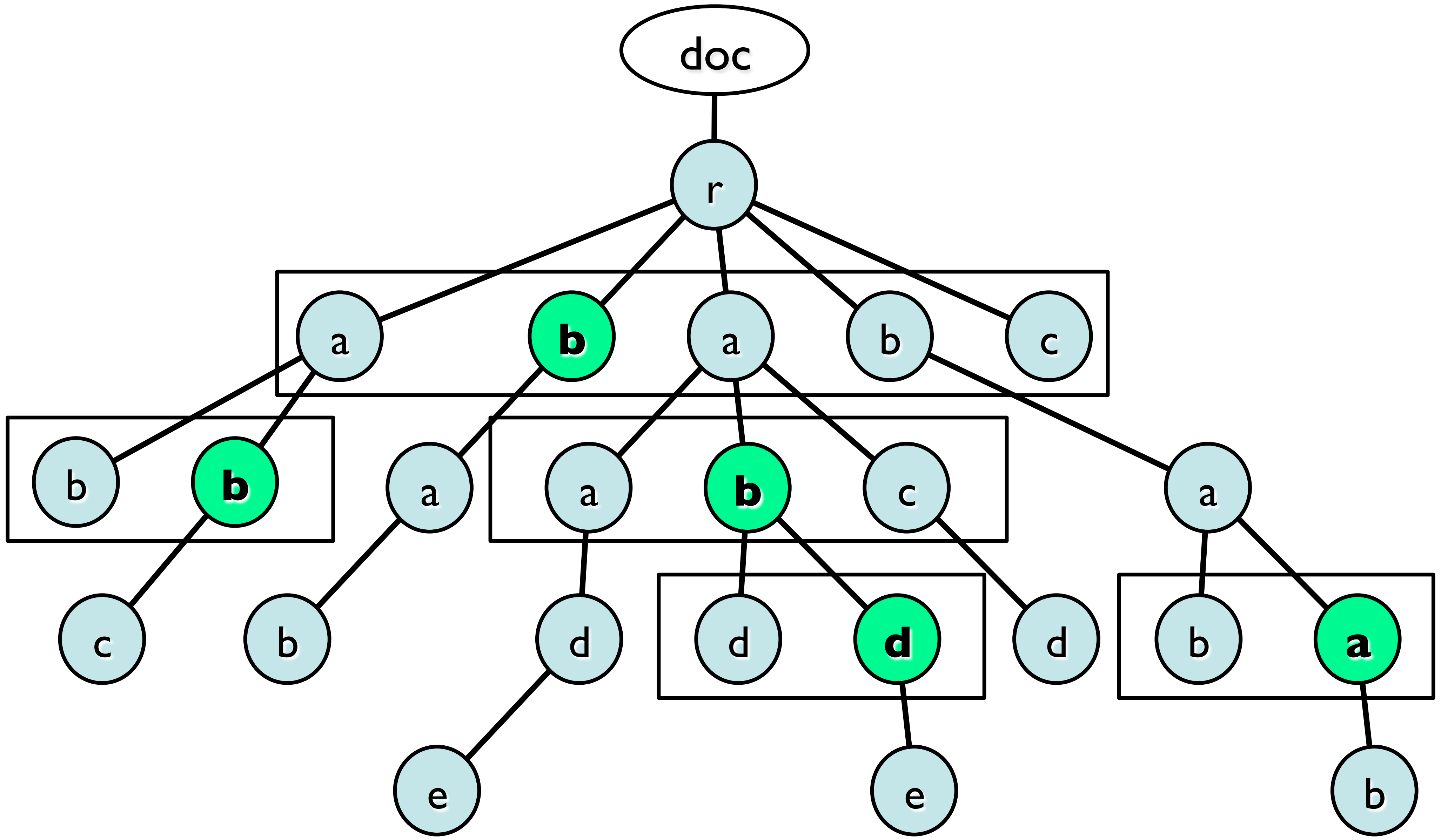
≠

```
/descendant-or-self::*/(child::node()[b])
```

=

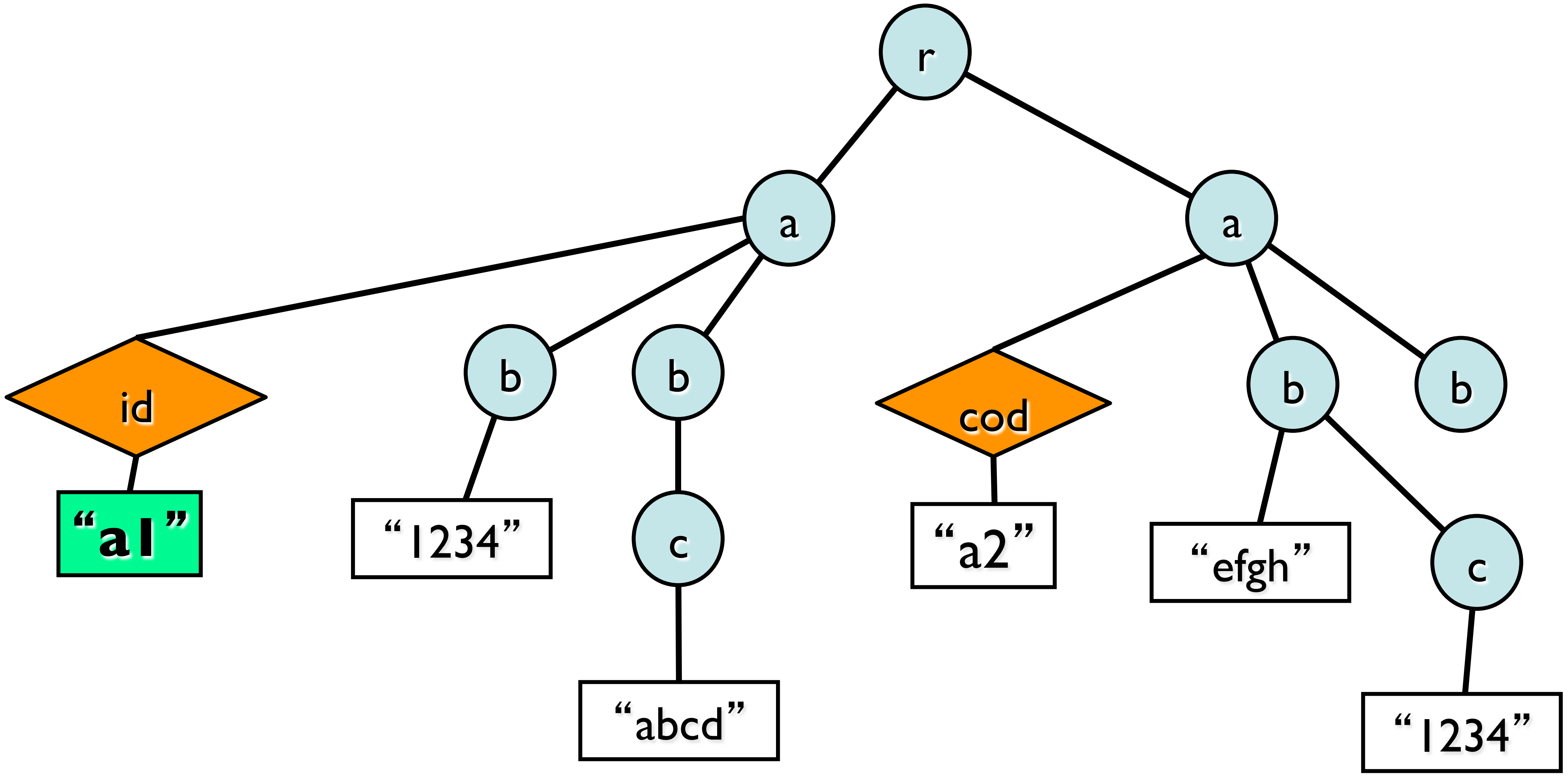
```
//node()[b]
```

Positional tests



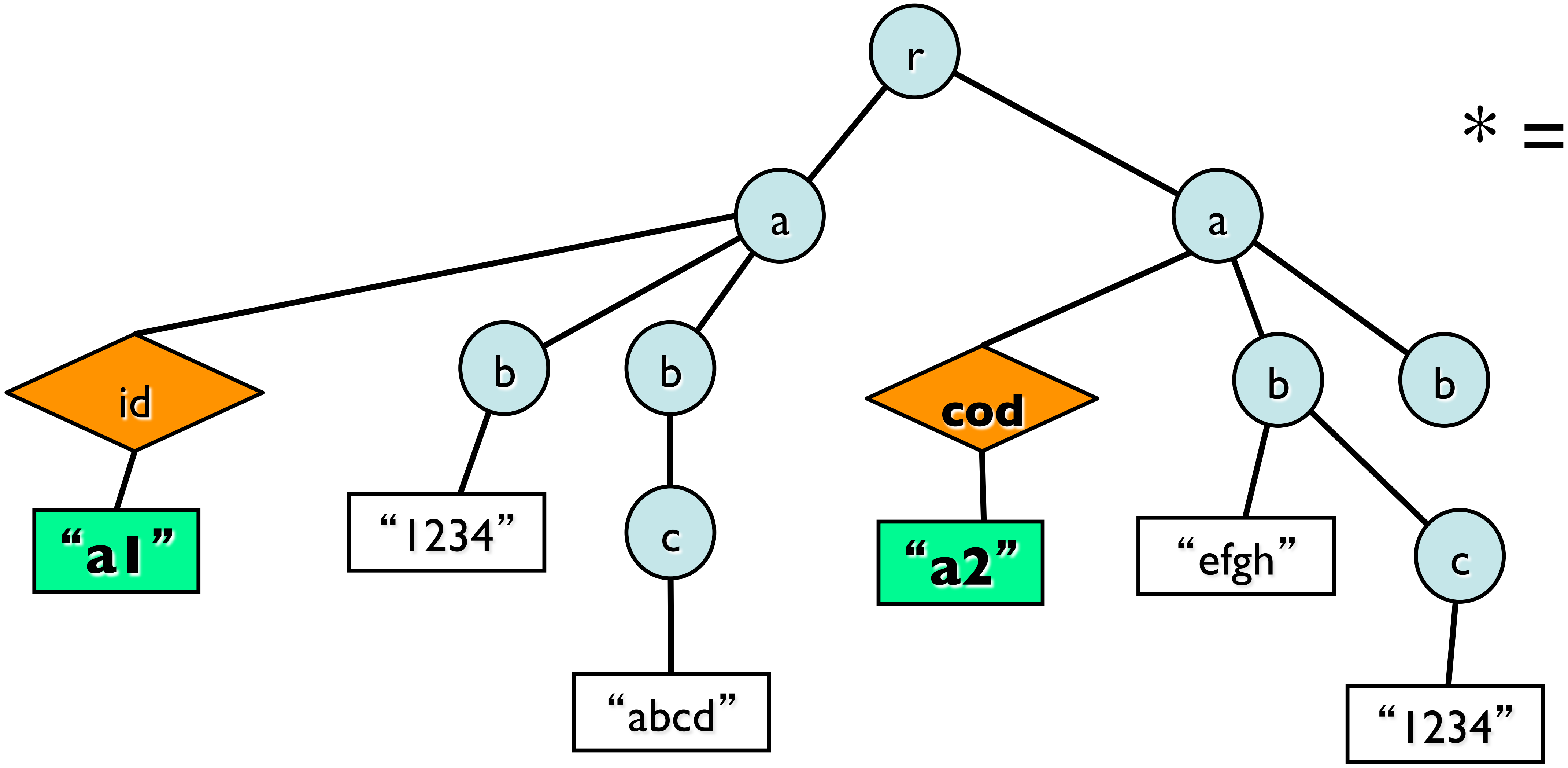
`//*[position()=2]` (or just `//*[2]`)

Querying attributes



`//a/@id` or `//a/attribute::id`

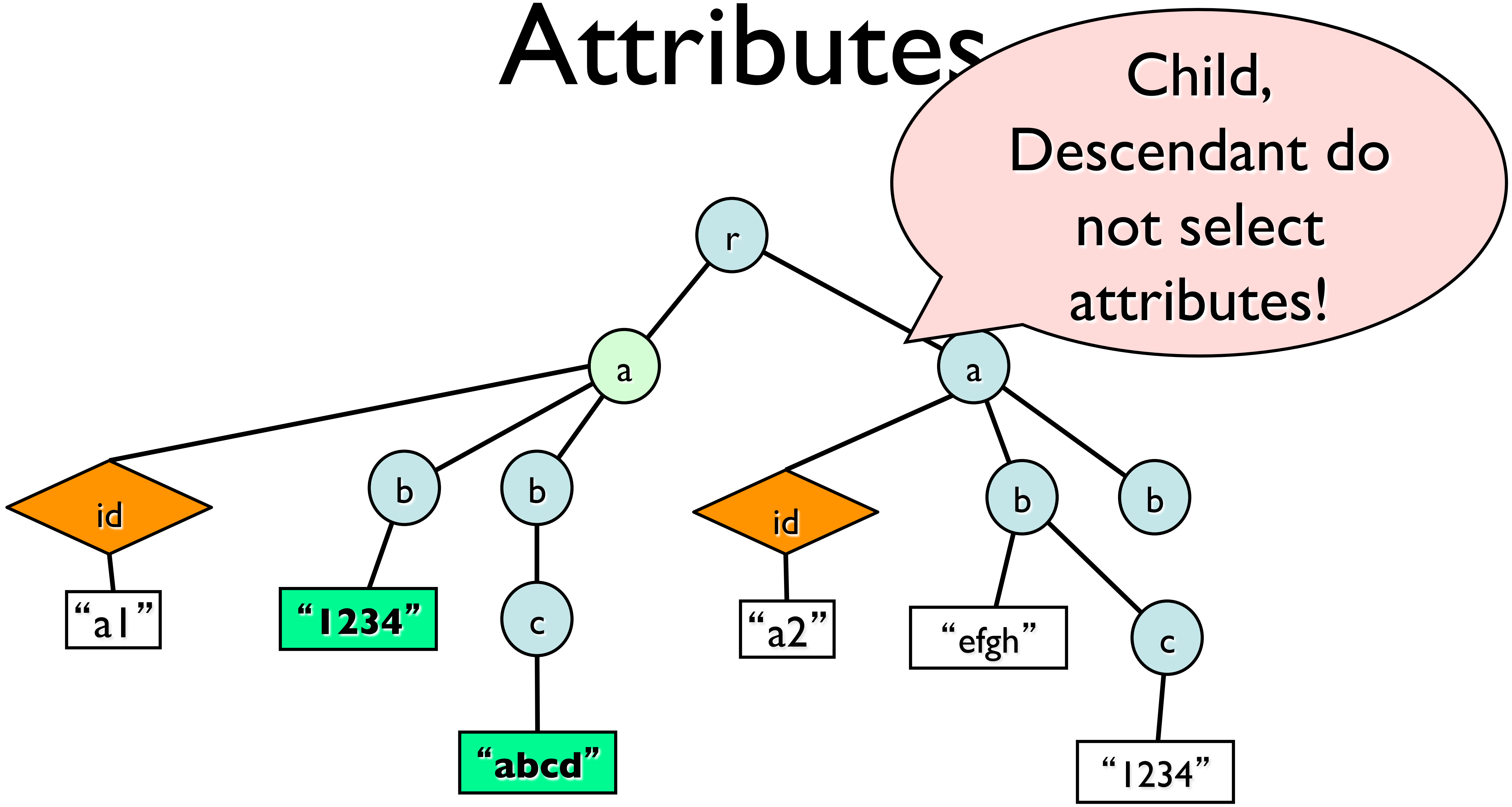
Attributes



* = any

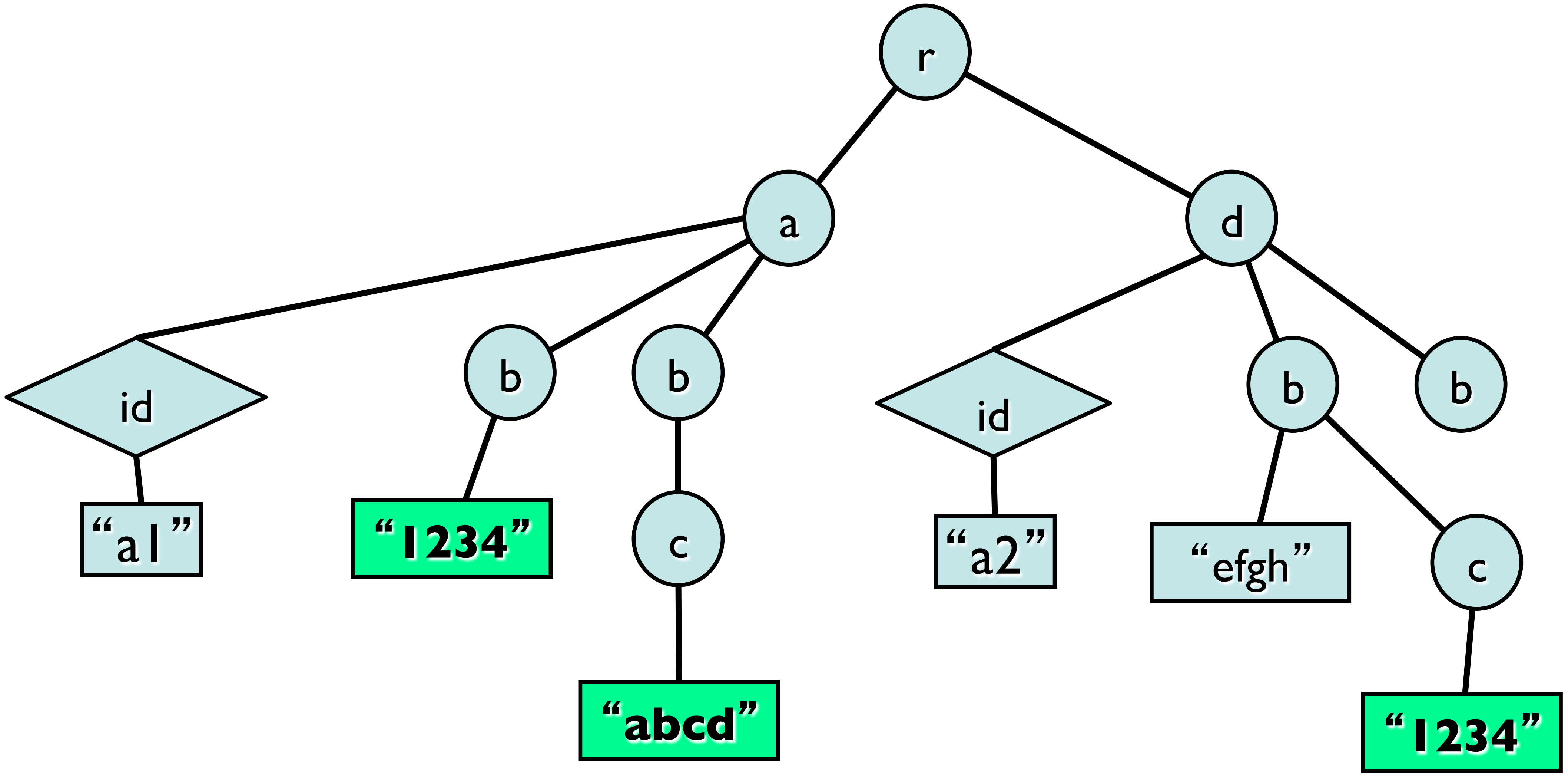
//@*

Attributes



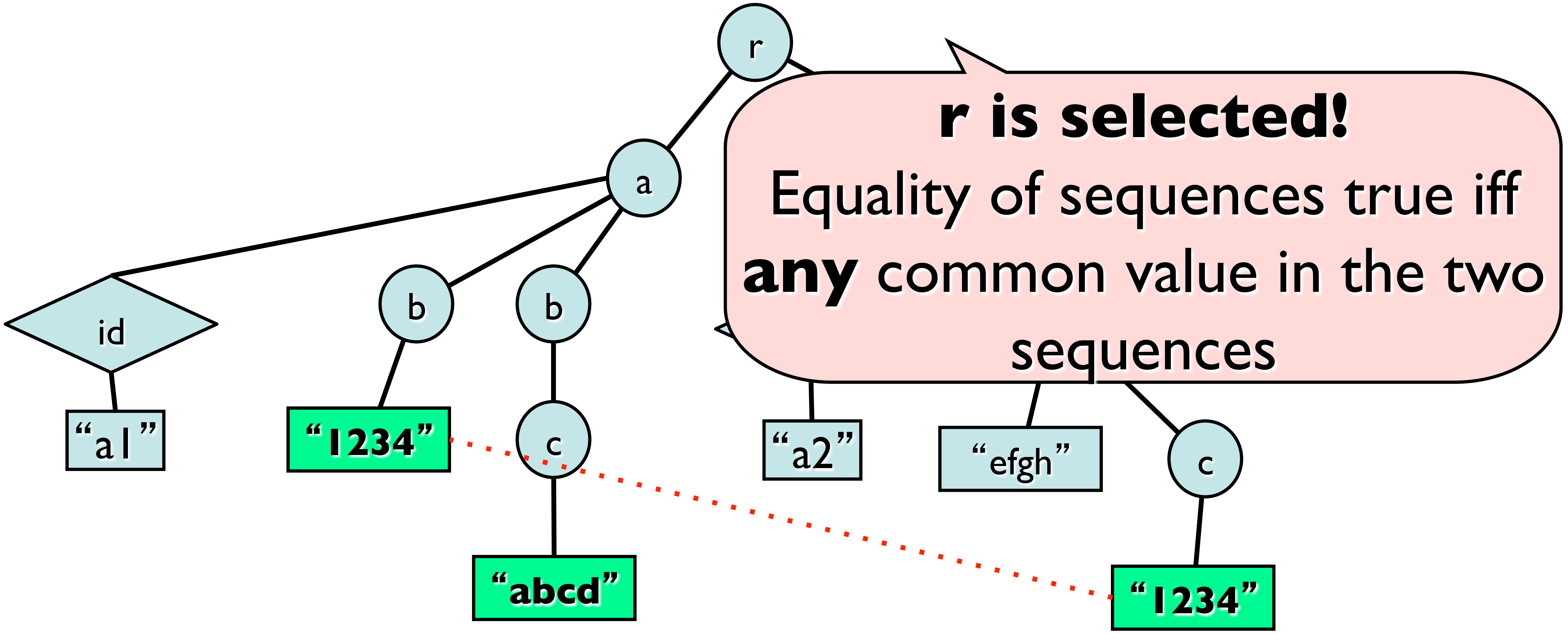
```
//a[@id="a1"]//text()
```

Equality (1) : empty query ?



```
/r[a/b//text() = d/b/c/text()]
```

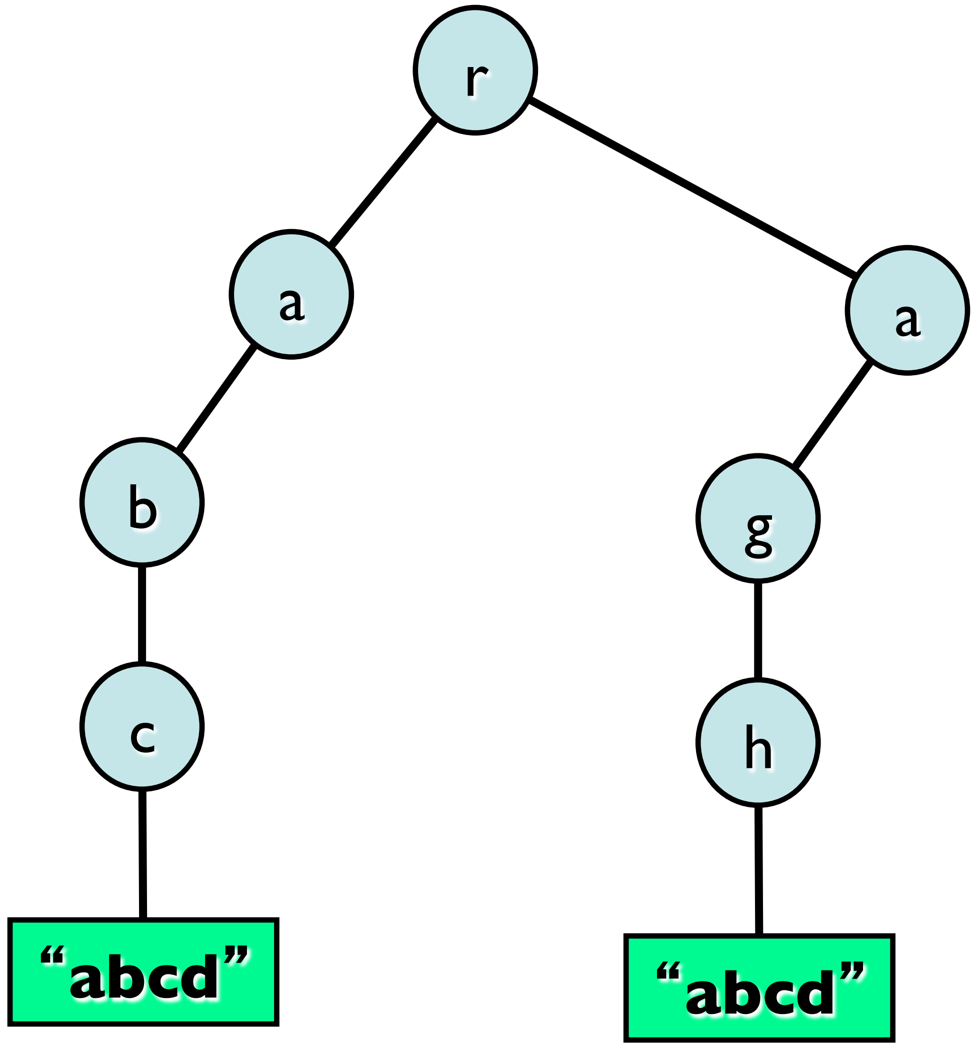
Equality (1) : empty query ?



r is selected!
Equality of sequences true iff **any** common value in the two sequences

```
/r[a/b//text() = d/b/c/text()]
```

Equality (2) : empty query ?



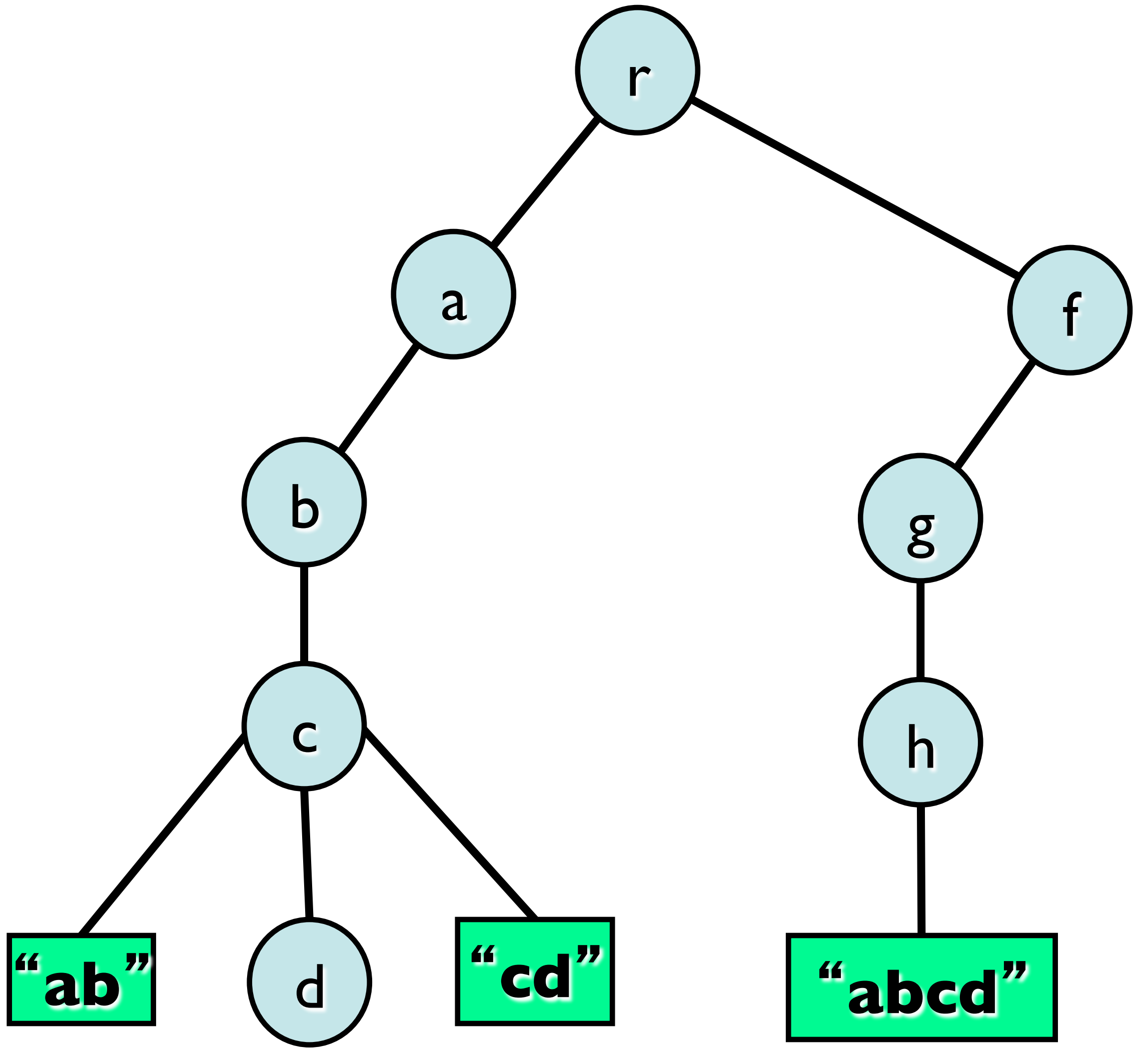
```
/r[a[1] = a[2]]
```

r is selected! Equality of two nodes is true **iff** the string-value comparison of the two nodes is true

string-value of a node

« *concatenation of the string-values of all* ***text nodes****, descendant of the context node, in document order.* »

Equality (3) : empty query ?



```
/r[ a = f ]
```

r is selected! Equality of two nodes is true **iff** the string-value comparison of the two nodes is true

Equality for the Web

- Testing tree isomorphism may be costly in the Web
  - better a text-value comparison

- List comparison may be too constraining
  - just search for a pair of similar objects

This is better adapted to the Web context !