Chapter 6

XPath

Introduction

- XPath is a language that lets you identify particular parts of XML documents
- XPath interprets XML documents as nodes (with content) organised in a tree structure
- XPath indicates nodes by (relative) position, type, content, and several other criteria
- Basic syntax is somewhat similar to that used for navigating file hierarchies
- XPath 1.0 (1999) and 2.0 (2010) are W3C recommendations

Some Tools for XPath

- Saxon (specifically Saxon-HE which implements XPath 2.0, XQuery 1.0 and XSLT 2.0)
- eXist-db (a native XML database supporting XPath 2.0, XQuery 1.0 and XSLT 1.0)
- XPath Checker (add-on for Firefox)
- XPath Expression Testbed (available online)

Data Model

XPath's data model has some non-obvious features:

- The tree's root node is not the same as the document's root (document) element
- The tree's root node contains the entire document including the root element (and comments and processing instructions that appear before it)
- XPath's data model does not include everything in the document: XML declaration and DTD are not addressable
- `xmlns` attributes are reported as namespace nodes

Data Model (2)

- There are 6 types of *node*:
  - *root*
  - *element*
  - *attribute*
  - *text*
  - *comment*
  - *processing instruction*
- Element nodes have an associated set of attribute nodes
- Attribute nodes are *not* children of element nodes
- The order of child element nodes is *significant*
- We will only consider the first 4 types of node

Example (1)

Recall our CD library example

```
<CD-library>
  <CD number="724356690424">
    <performance>
      <composer>Frederic Chopin</composer>
      <composition>Waltzes</composition>
      <soloist>Dinu Lipatti</soloist>
      <date>1950</date>
    </performance>
  </CD>
  ...
```

Example (2)

```
...
<CD number="419160-2">
  <composer>Johannes Brahms</composer>
  <soloist>Emil Gilels</soloist>
  <performance>
    <composition>Piano Concerto No. 2</composition>
    <orchestra>Berlin Philharmonic</orchestra>
    <conductor>Eugen Jochum</conductor>
    <date>1972</date>
  </performance>
  <performance>
    <composition>Fantasias Op. 116</composition>
    <date>1976</date>
  </performance>
</CD>
...
```

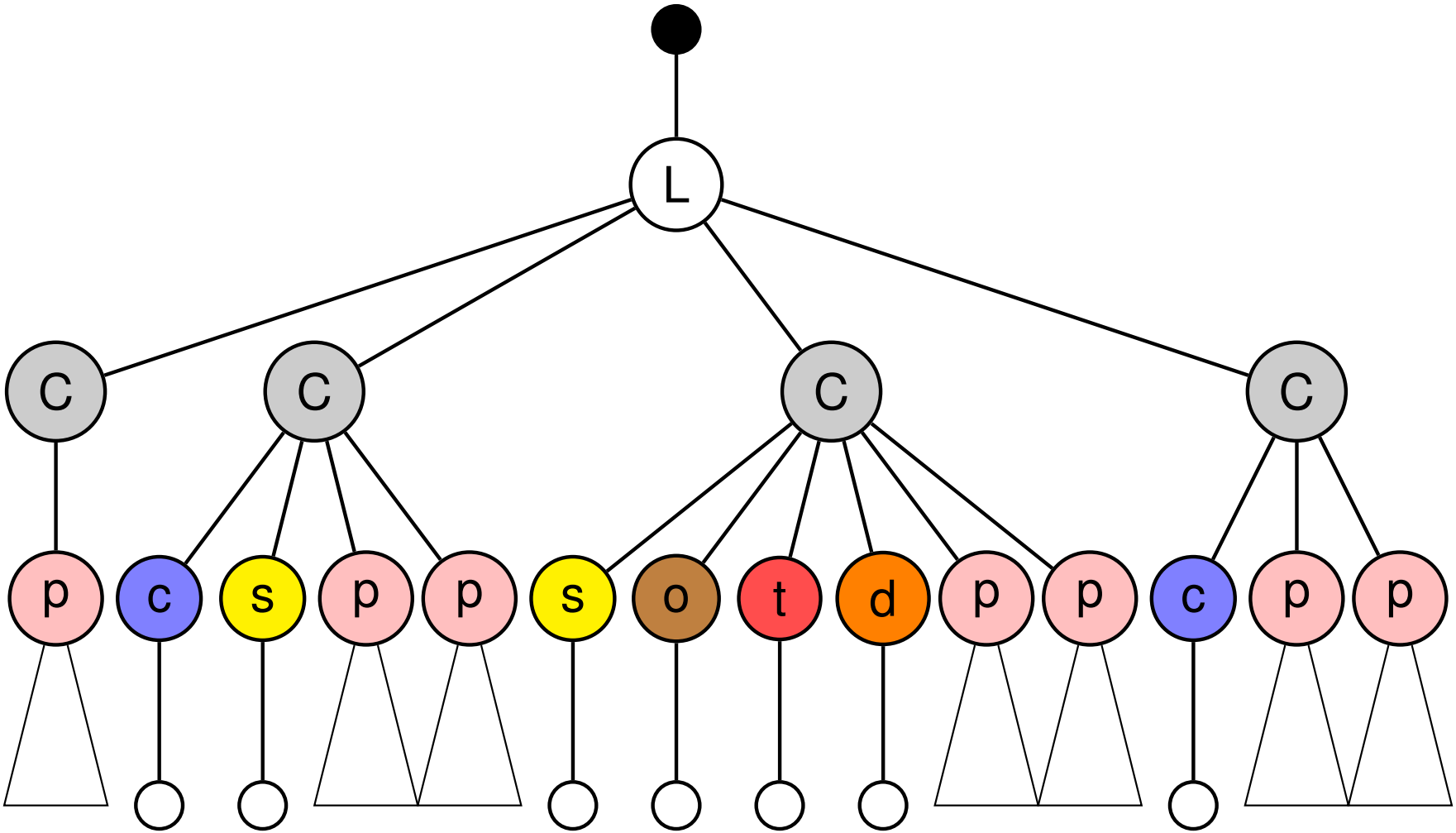
Example (3)

```
...
<CD number="449719-2">
  <soloist>Martha Argerich</soloist>
  <orchestra>London Symphony Orchestra</orchestra>
  <conductor>Claudio Abbado</conductor>
  <date>1968</date>
  <performance>
    <composer>Frederic Chopin</composer>
    <composition>Piano Concerto No. 1</composition>
  </performance>
  <performance>
    <composer>Franz Liszt</composer>
    <composition>Piano Concerto No. 1</composition>
  </performance>
</CD>
...
```

Example (4)

```
...
<CD number="430702-2">
  <composer>Antonin Dvorak</composer>
  <performance>
    <composition>Symphony No. 9</composition>
    <orchestra>Vienna Philharmonic</orchestra>
    <conductor>Kirill Kondrashin</conductor>
    <date>1980</date>
  </performance>
  <performance>
    <composition>American Suite</composition>
    <orchestra>Royal Philharmonic</orchestra>
    <conductor>Antal Dorati</conductor>
    <date>1984</date>
  </performance>
</CD>
</CD-library>
```

Example — Tree Structure



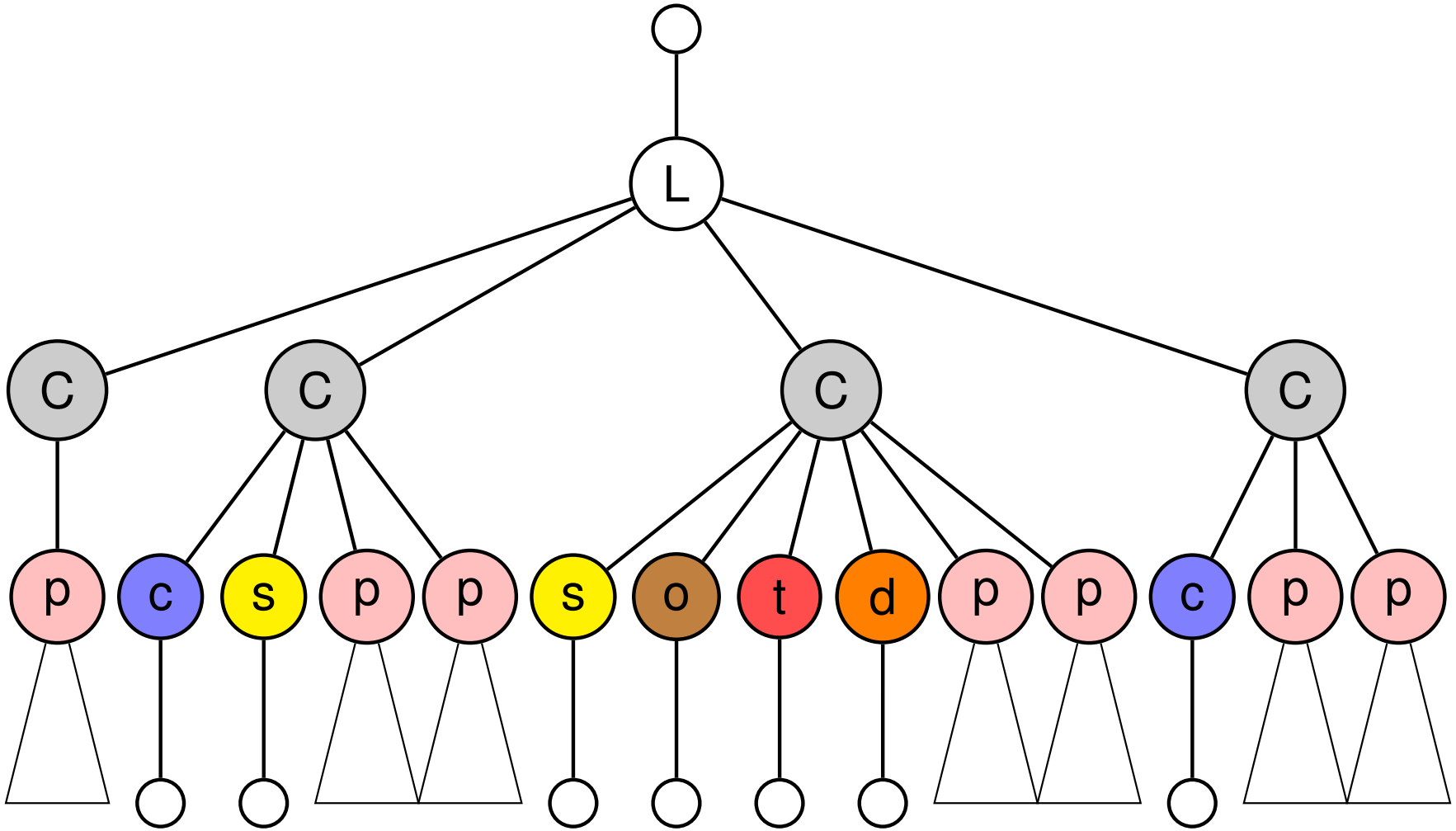
Location Path

- The most useful XPath expression is a *location path*: e.g., `/CD-library/CD/performance`
- A location path consists of at least one *location step*: e.g., `CD-library`, `CD` and `performance` are location steps
- A location step takes as input a set of nodes, also called the *context* (to be defined more precisely later)
- The location step expression is applied to this node set and results in an output node set
- This output node set is used as input for the next location step

Location Path (2)

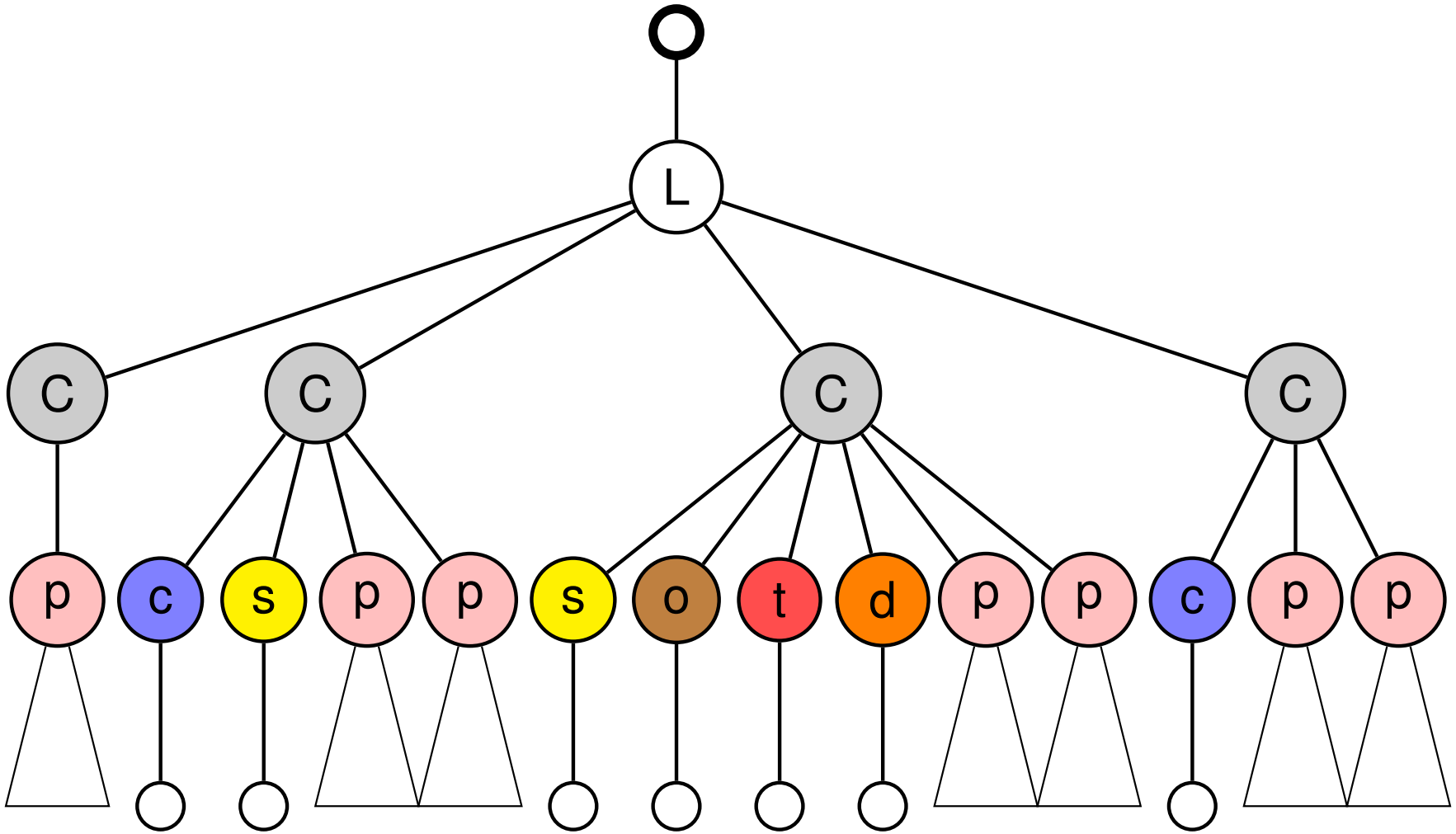
- There are two different kinds of location paths:
  - *Absolute* location paths
  - *Relative* location paths
- An absolute location path
  - starts with `/`
  - is followed by a relative location path
  - is evaluated at the root (context) node of a document
  - e.g., `/CD-library/CD/performance`
- A relative location path
  - is a sequence of location steps
  - each separated by `/`
  - evaluated with respect to some other context nodes
  - e.g., `CD/performance`

Evaluation of absolute location path



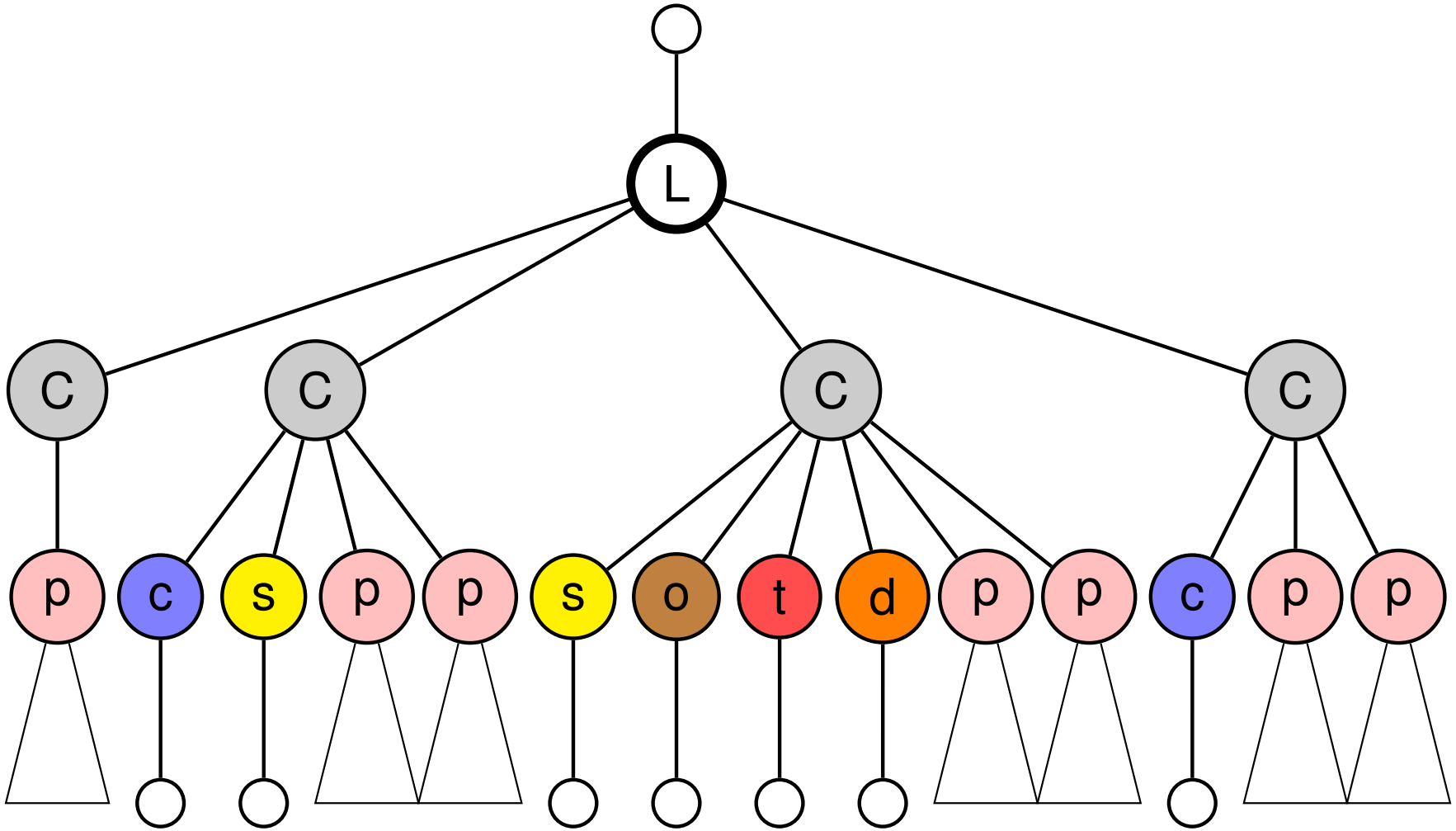
Evaluation of absolute location path

/



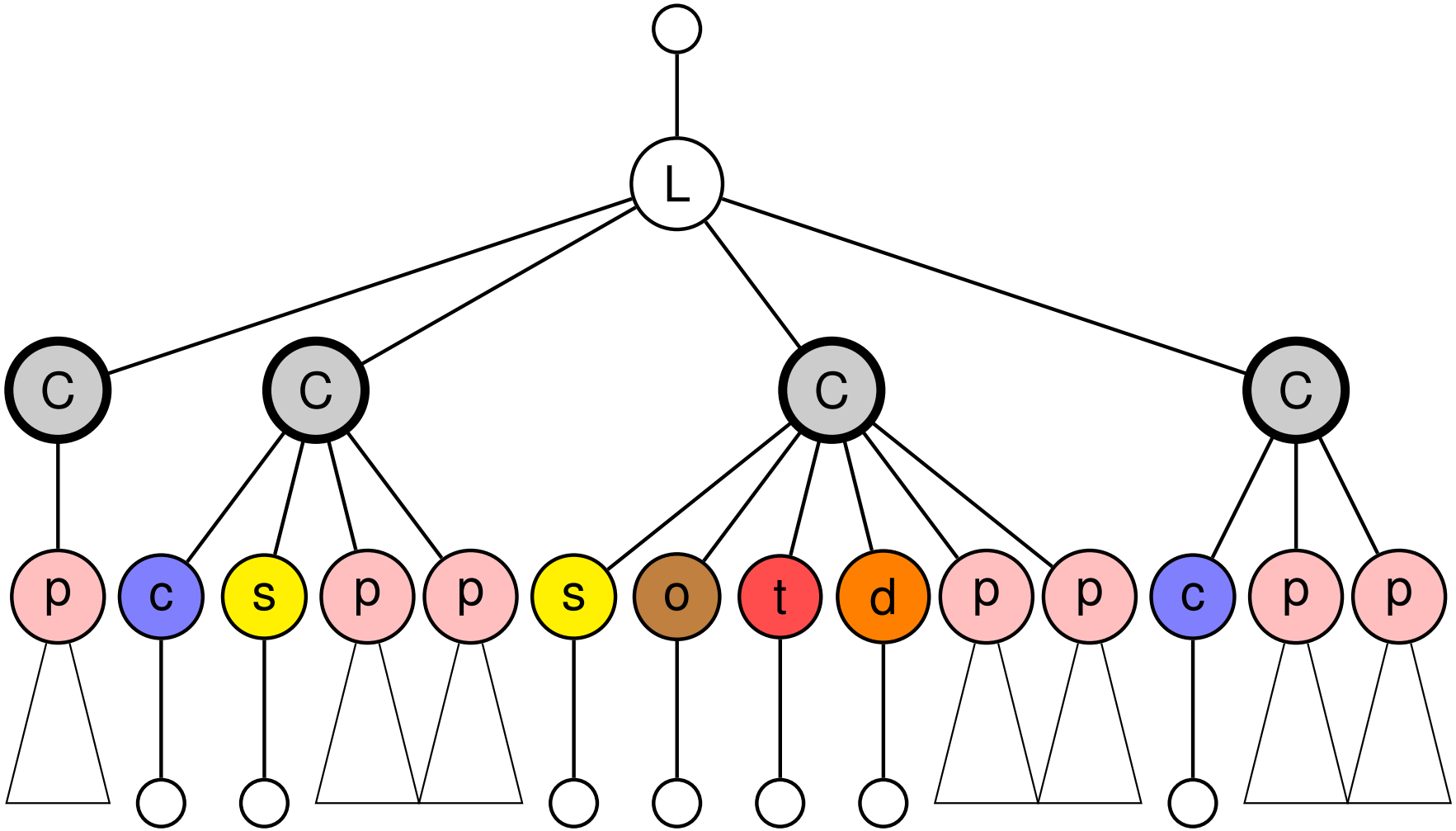
Evaluation of absolute location path

`/CD-library`



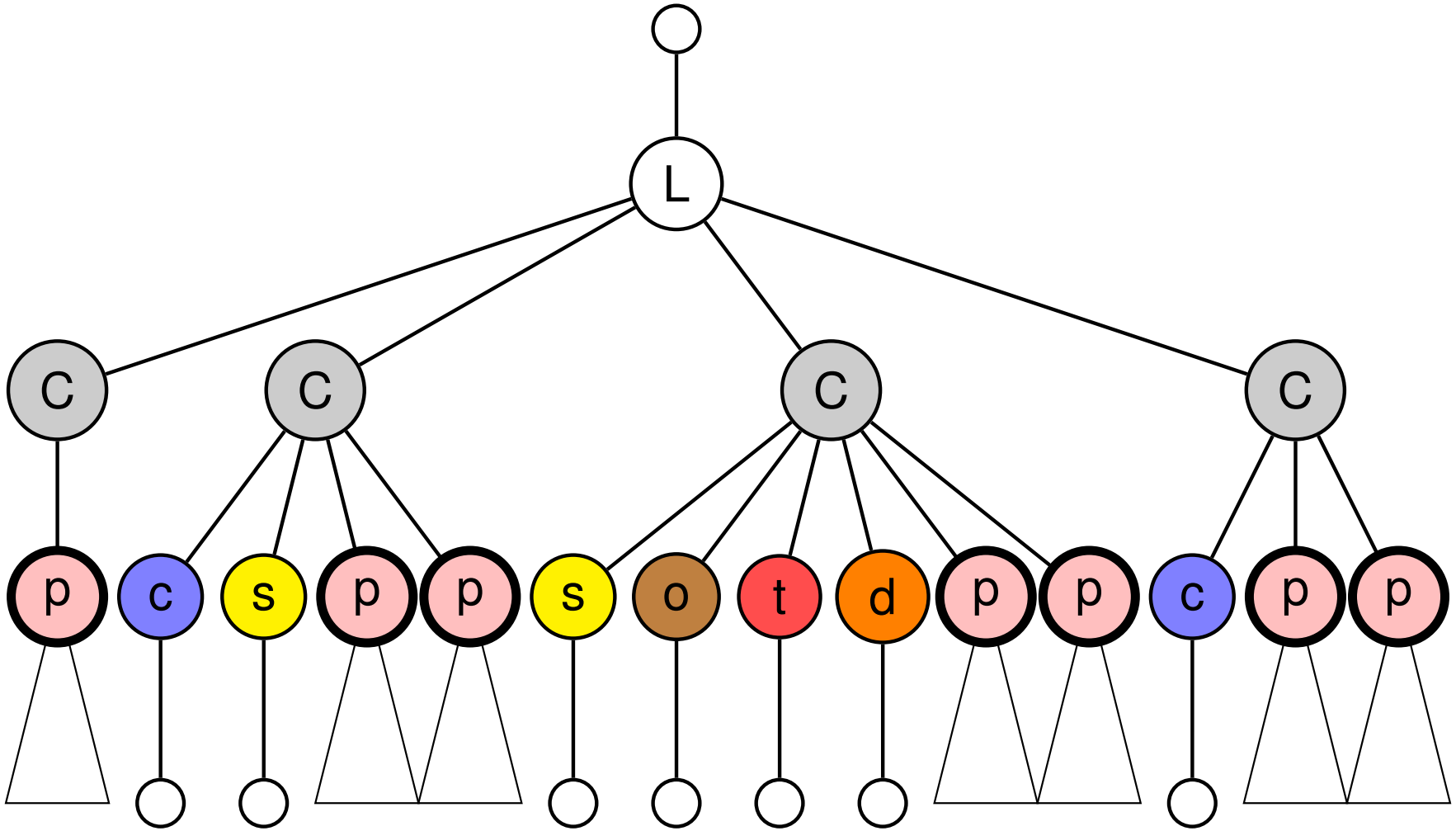
Evaluation of absolute location path

```
/CD-library/CD
```



Evaluation of absolute location path

```
/CD-library/CD/performance
```

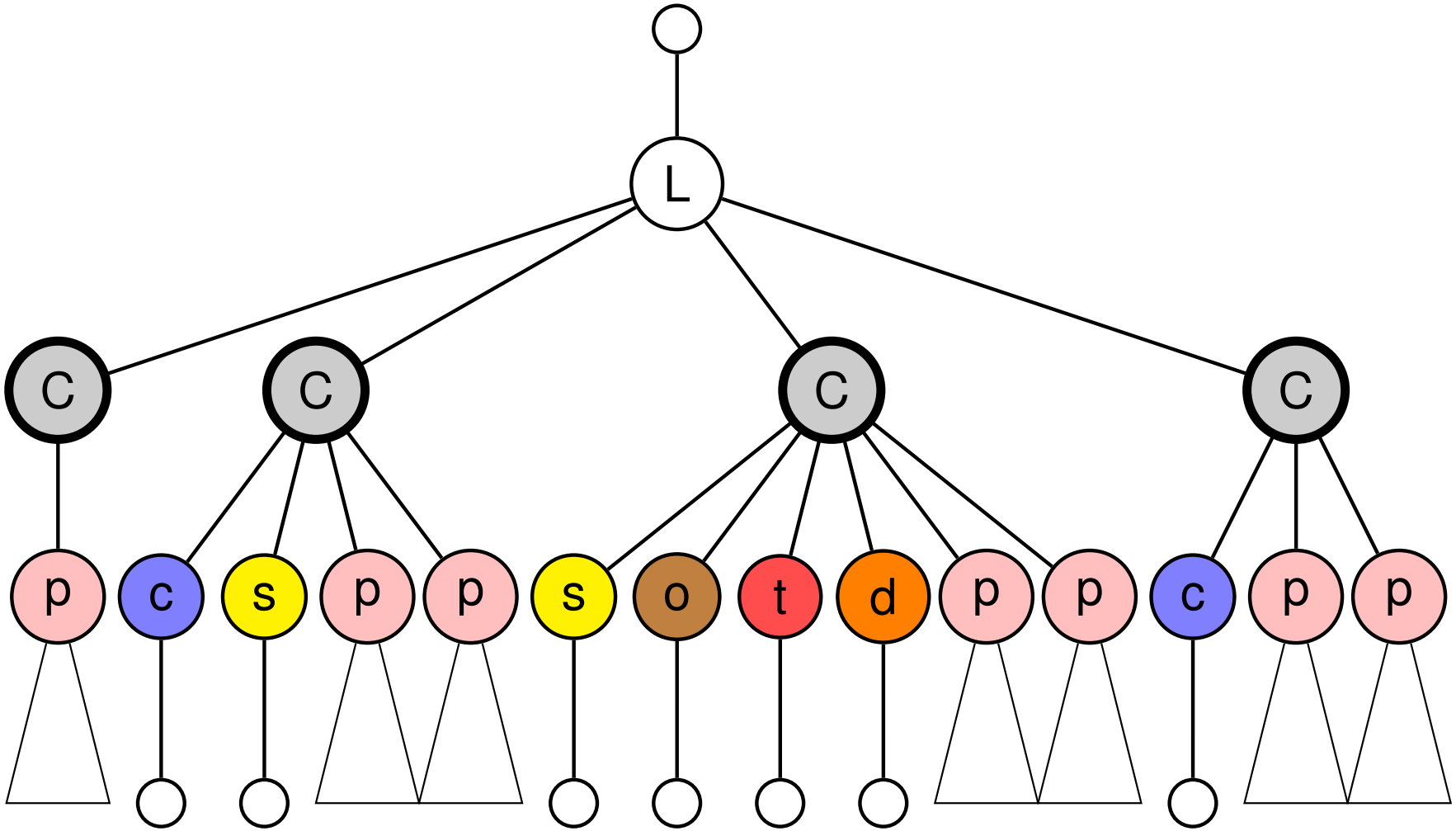


Location Step

- In general, a location step in turn consists of a
  - (navigation) axis
  - node test
  - predicate(s)
- Syntax is *axis* `::` *node test* `[` *predicate* `]` `...` `[` *predicate* `]`
- e.g., `child::CD[composer='Johannes Brahms']`
  - `child` is the axis
  - `CD` is the node test
  - `composer='Johannes Brahms'` is the predicate
- A location step is applied to each node in the context (i.e., each node becomes the context node)
- All resulting nodes are added to the output set of this location step

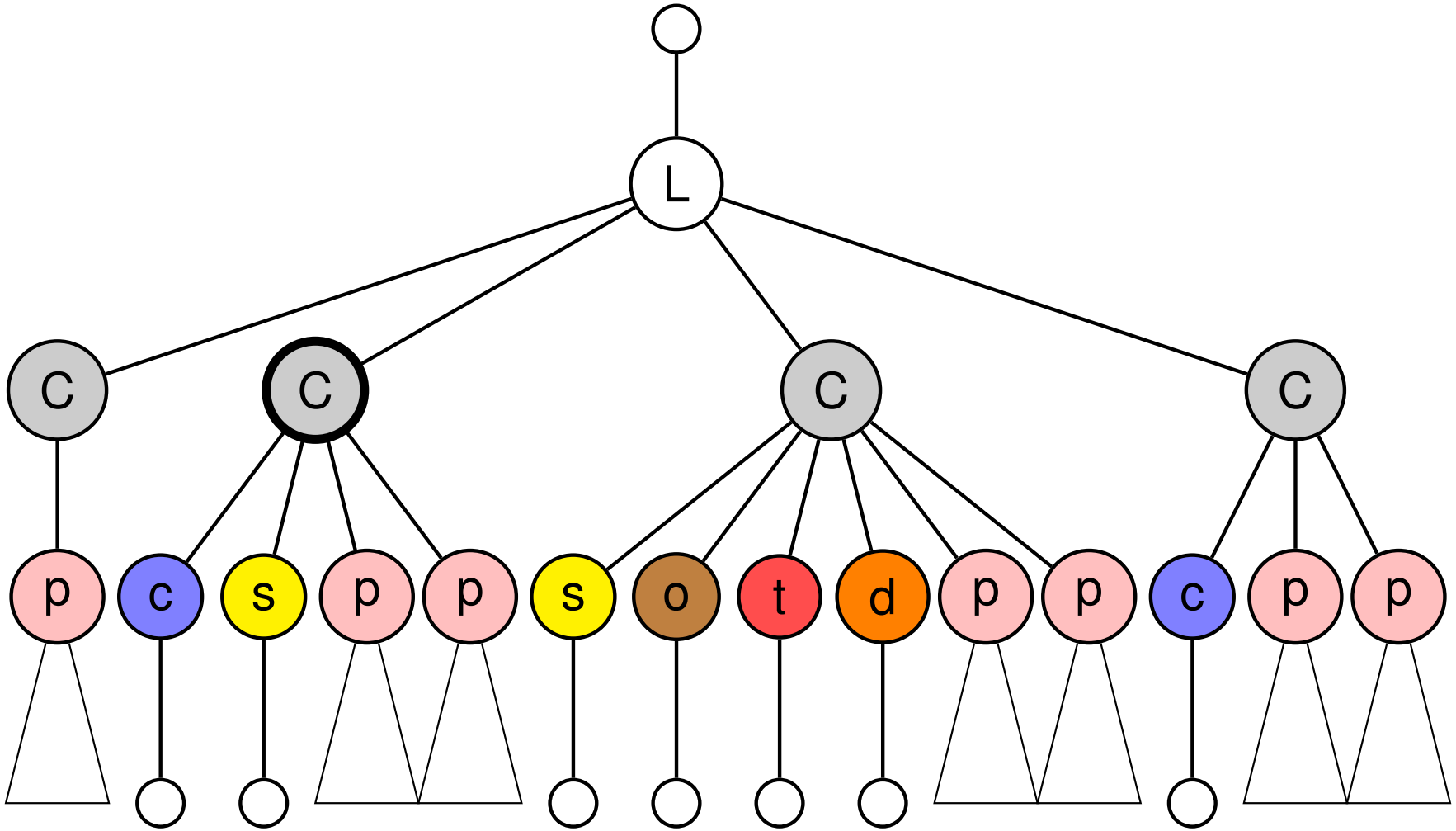
Evaluation of predicate

```
/child::CD-library/child::CD
```



Evaluation of predicate

```
/child::CD-library/child::CD[composer='Johannes Brahms']
```

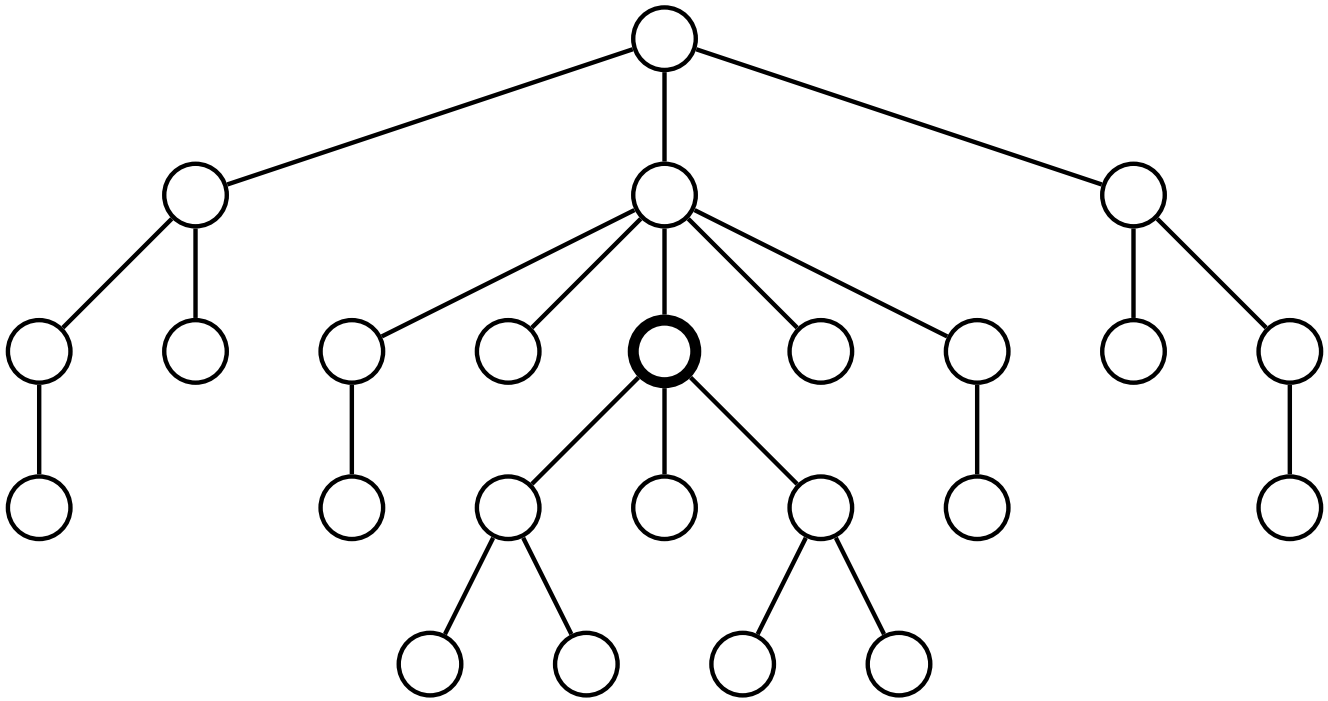


Axes

- An axis specifies what nodes, relative to the current context node, to consider
- There are 13 different axes (some can be abbreviated)
  - `self`, abbreviated by `.`
  - `child`, abbreviated by *empty axis*
  - `parent`, abbreviated by `..`
  - `descendant-or-self`, abbreviated by *empty location step*
  - `descendant`, `ancestor`, `ancestor-or-self`
  - `following`, `following-sibling`, `preceding`, `preceding-sibling`
  - `attribute`, abbreviated by `@`
  - `namespace`

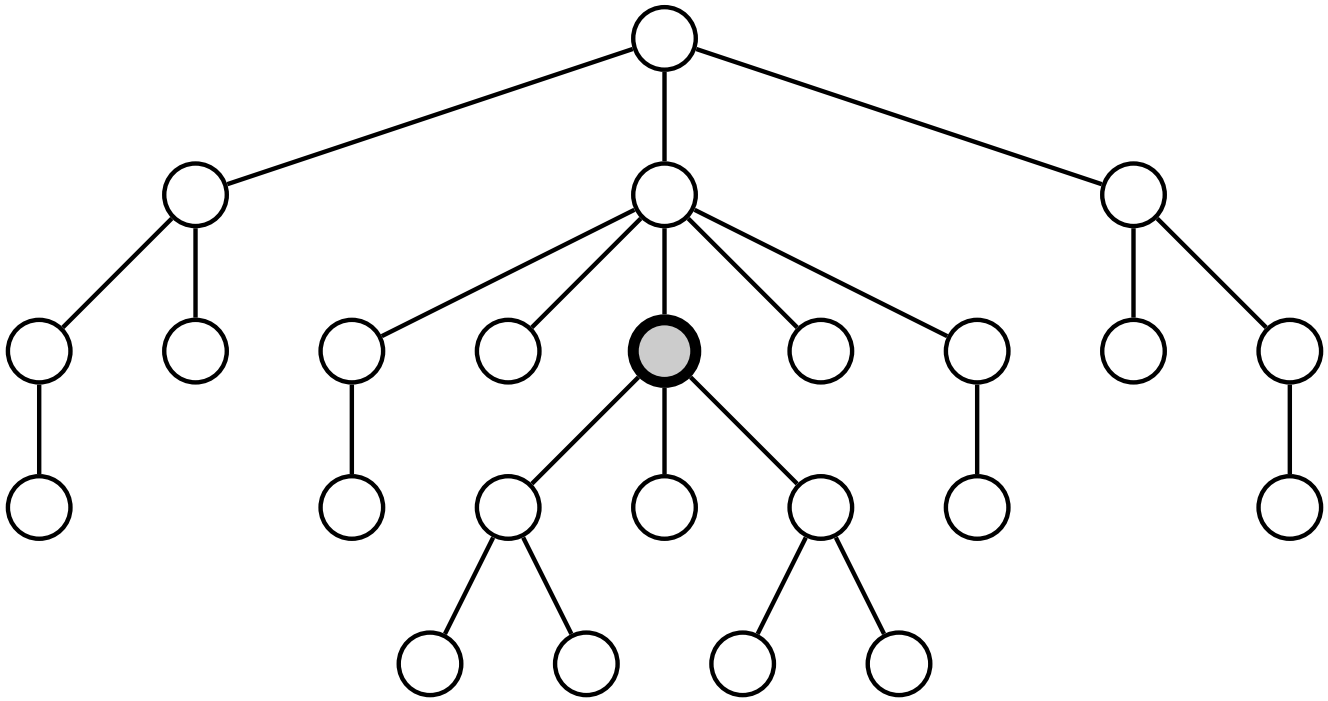
Axes

- The following slides show (graphical) examples of the axes, assuming the node in bold typeface is the context node



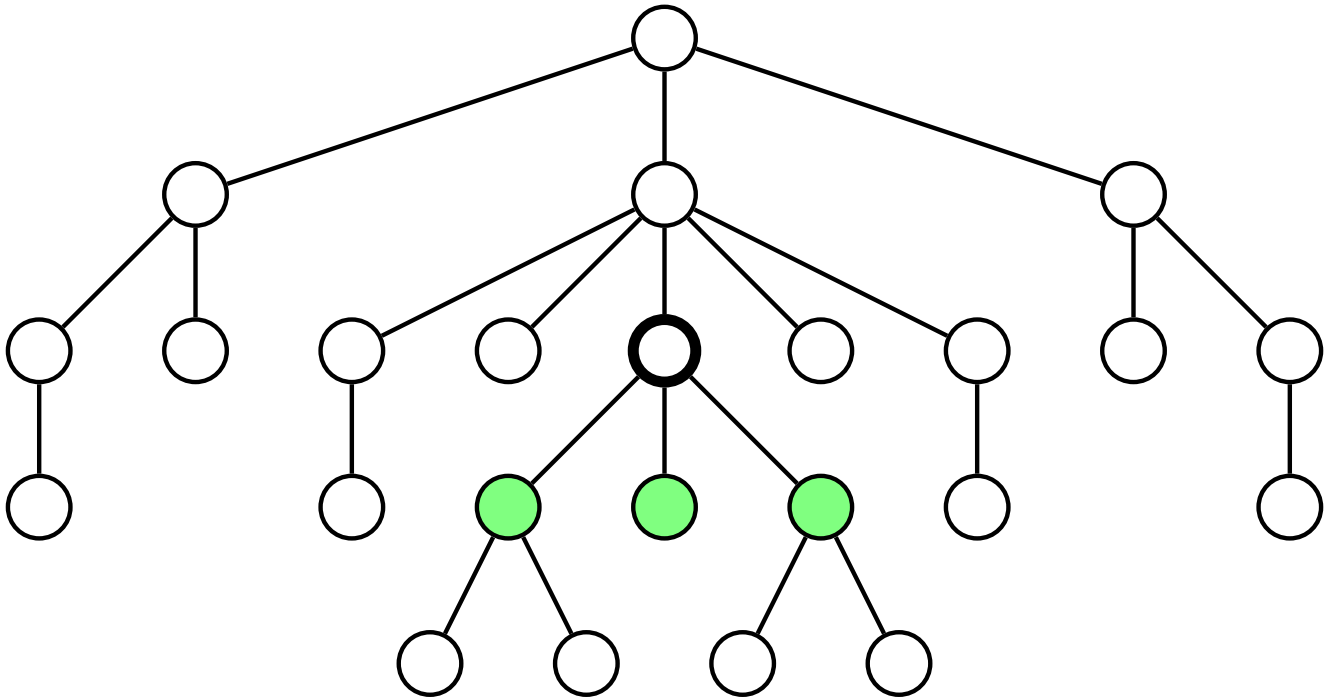
Self-Axis

- The self-axis just contains the context node itself



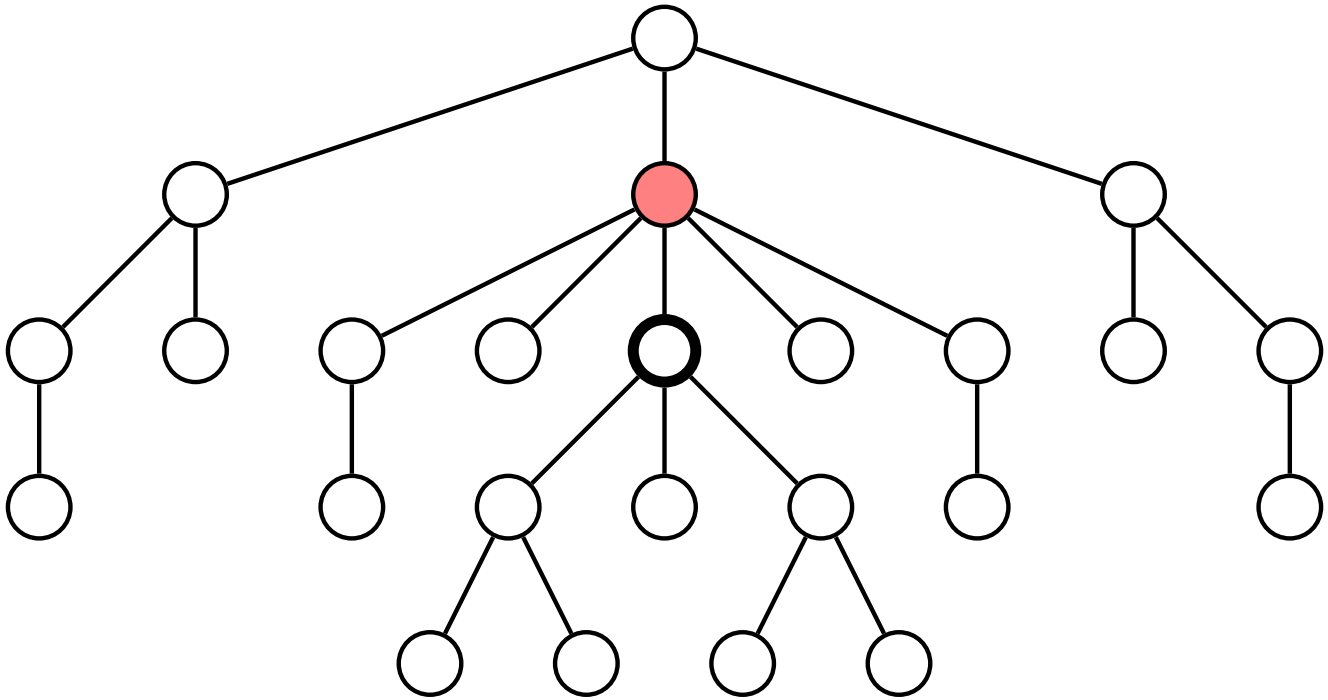
Child-Axis

- The child-axis contains the children (direct descendants) of the context node



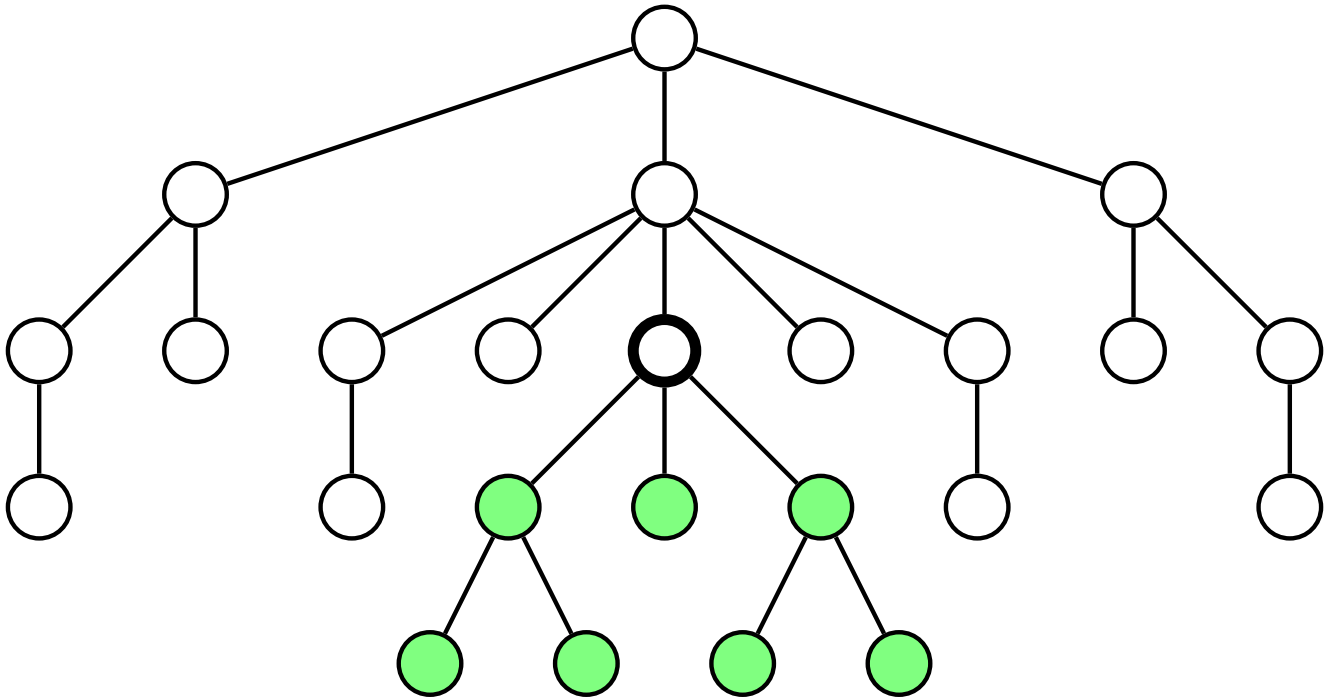
Parent-Axis

- The parent-axis contains the parent (direct ancestor) of the context node



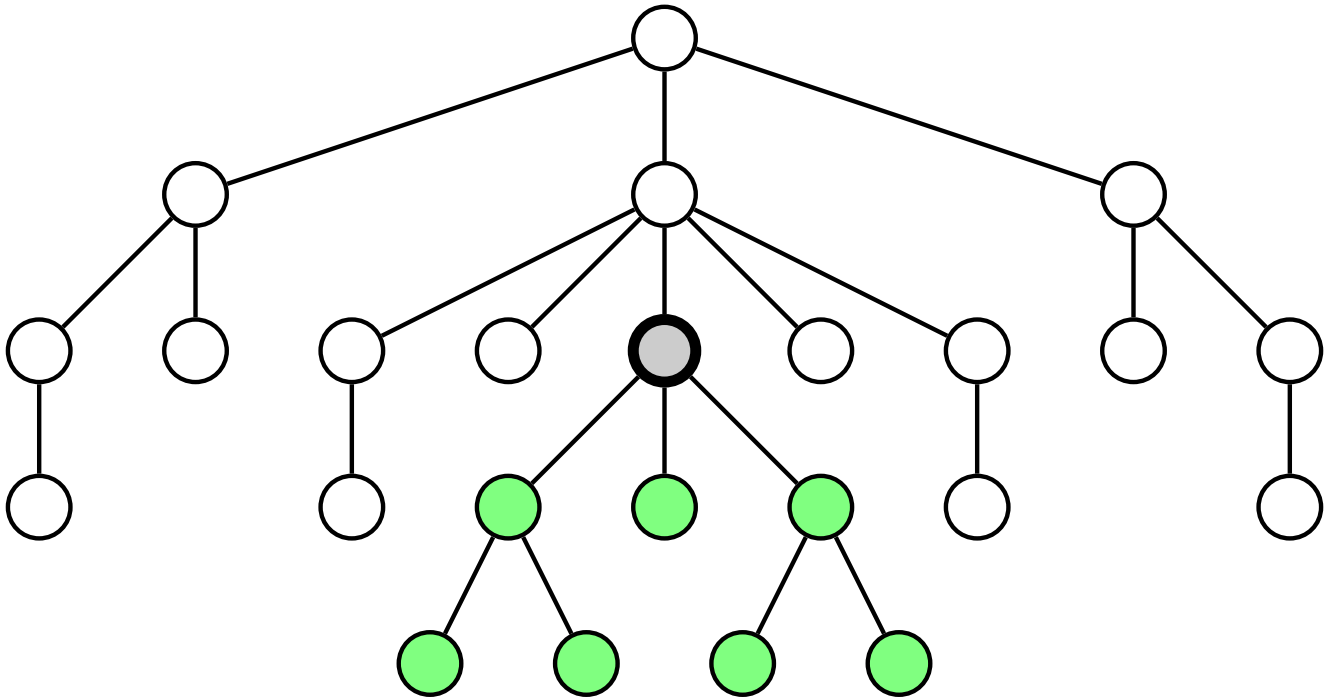
Descendant-Axis

- The descendant-axis contains all direct and indirect descendants of the context node



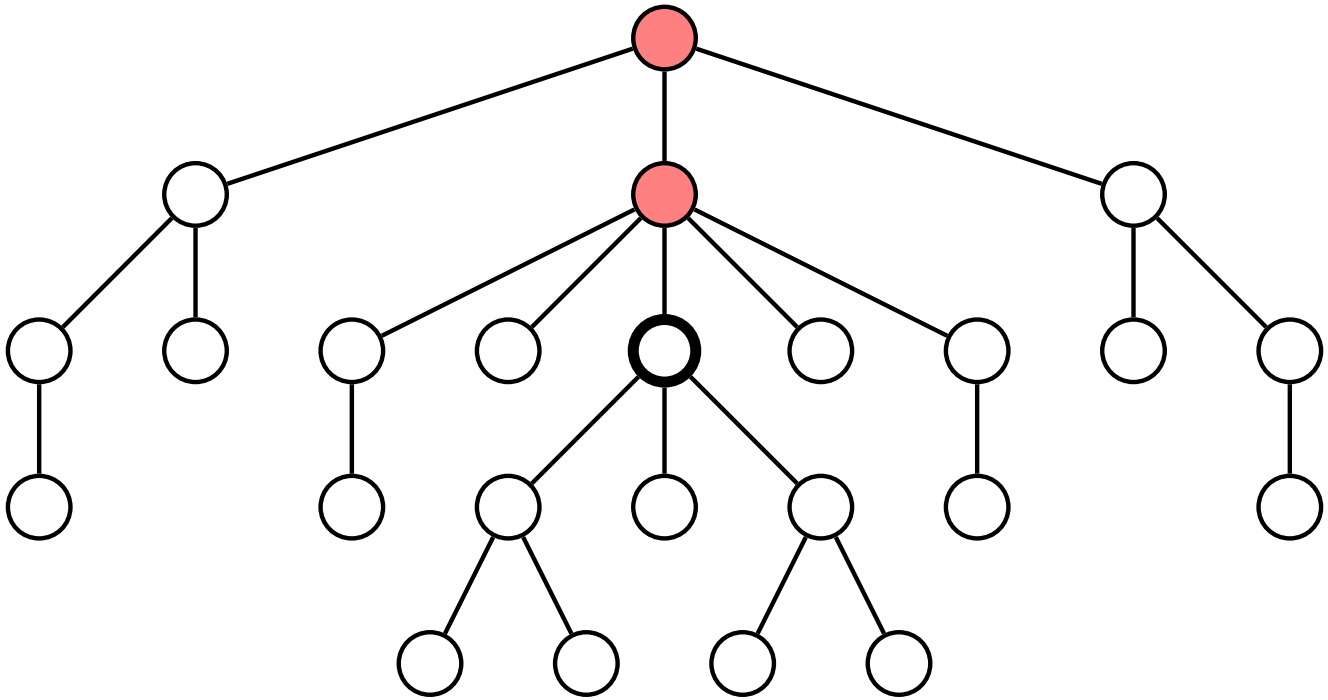
Descendant-Or-Self-Axis

- The descendant-or-self-axis contains all direct and indirect descendants of the context node + the context node itself



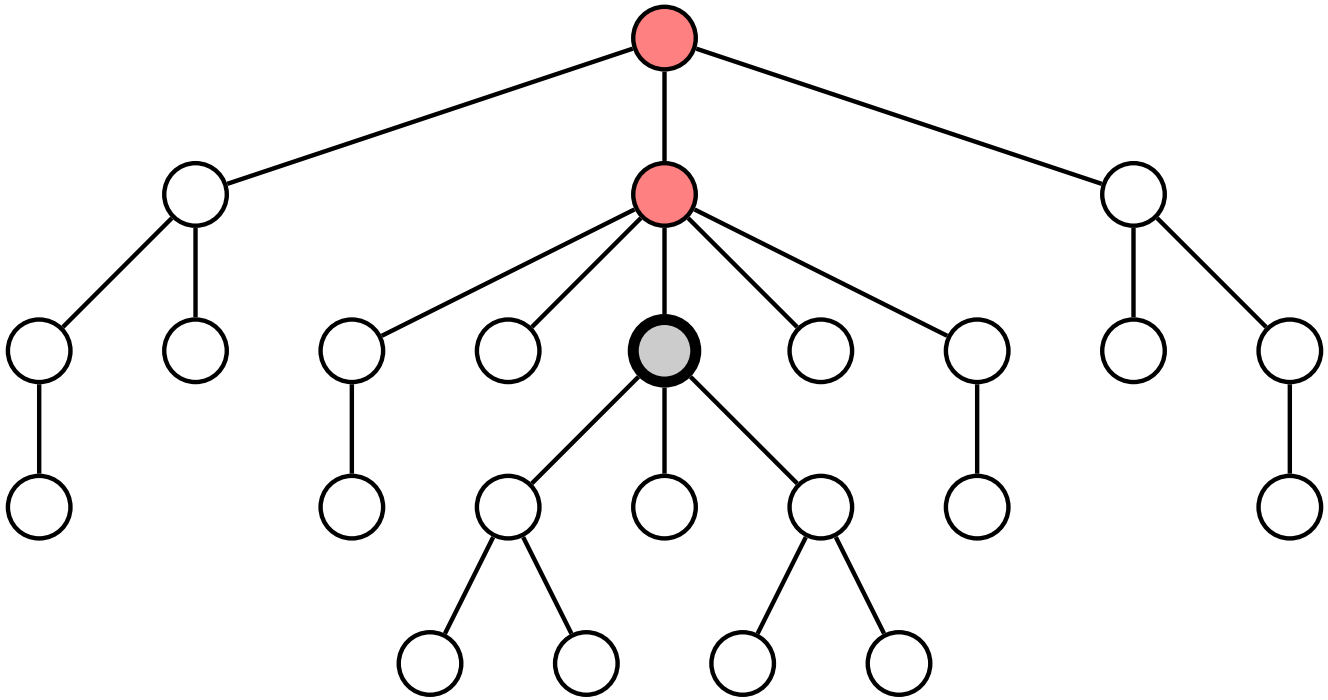
Ancestor-Axis

- The ancestor-axis contains all direct and indirect ancestors of the context node



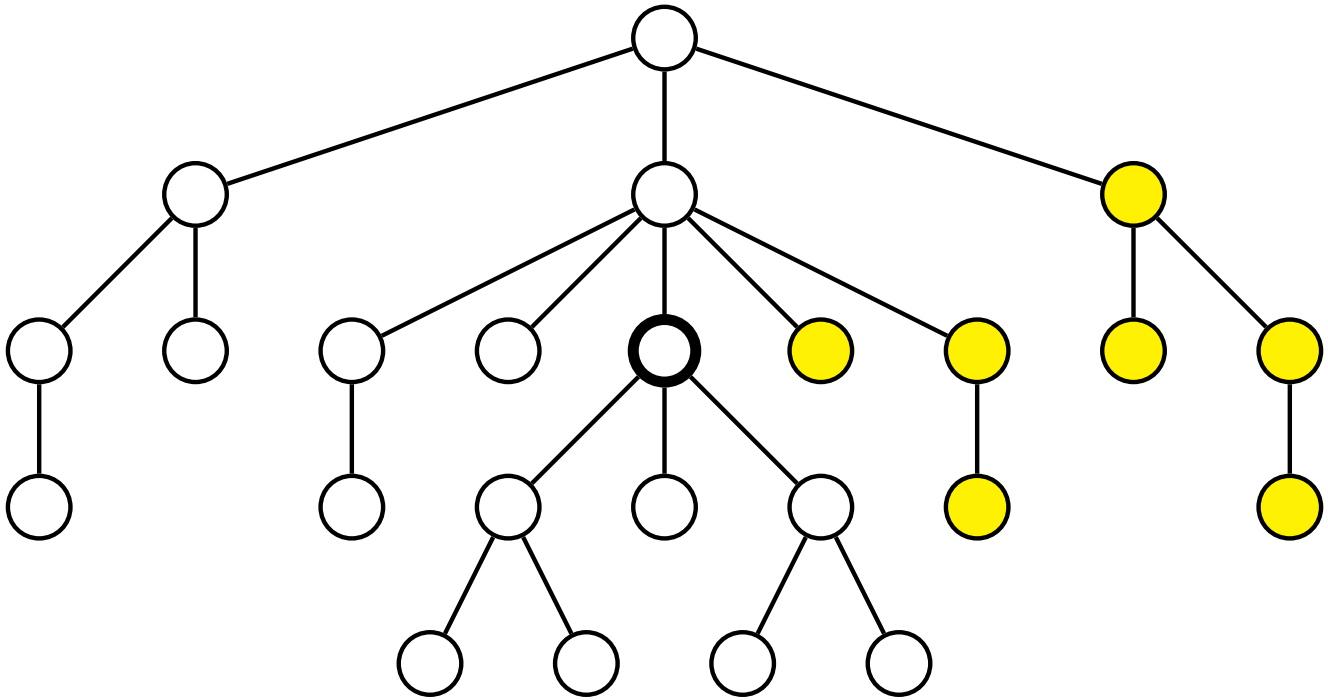
Ancestor-Or-Self-Axis

- The ancestor-or-self-axis contains all direct and indirect ancestors of the context node + the context node itself



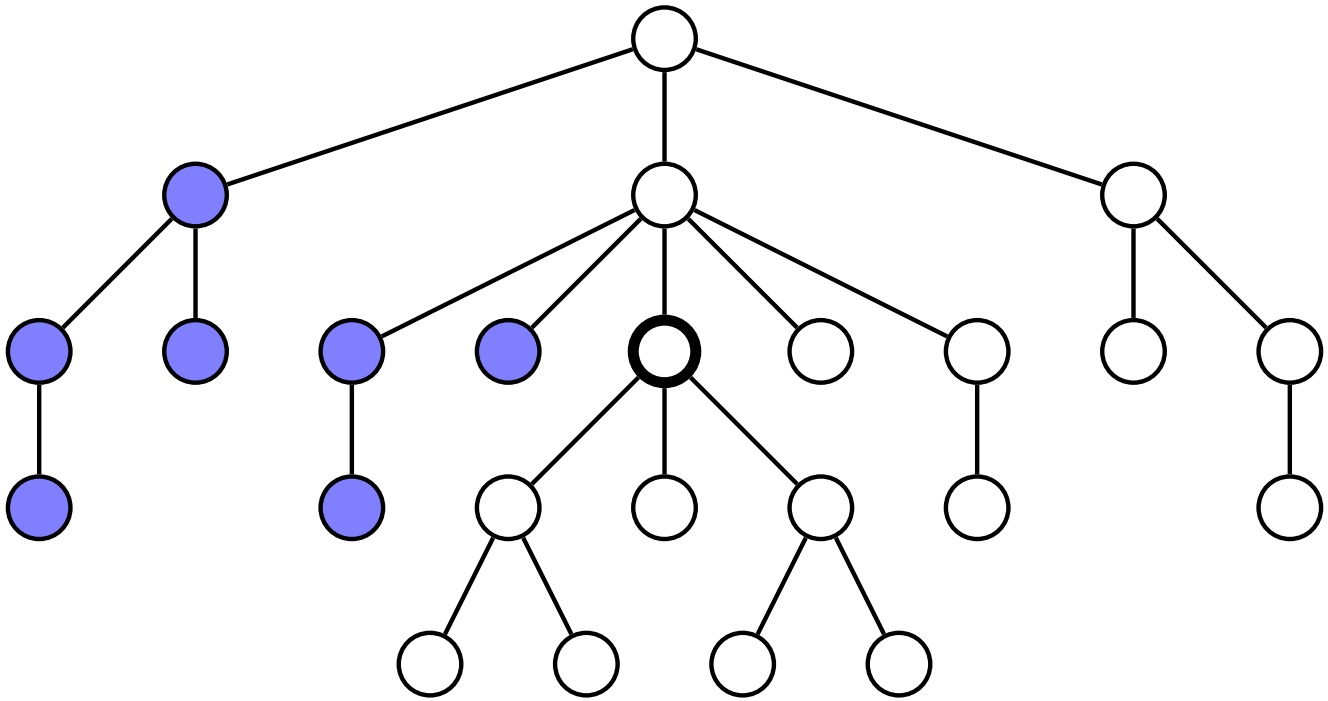
Following-Axis

- The following-axis contains all nodes that begin after the context node ends



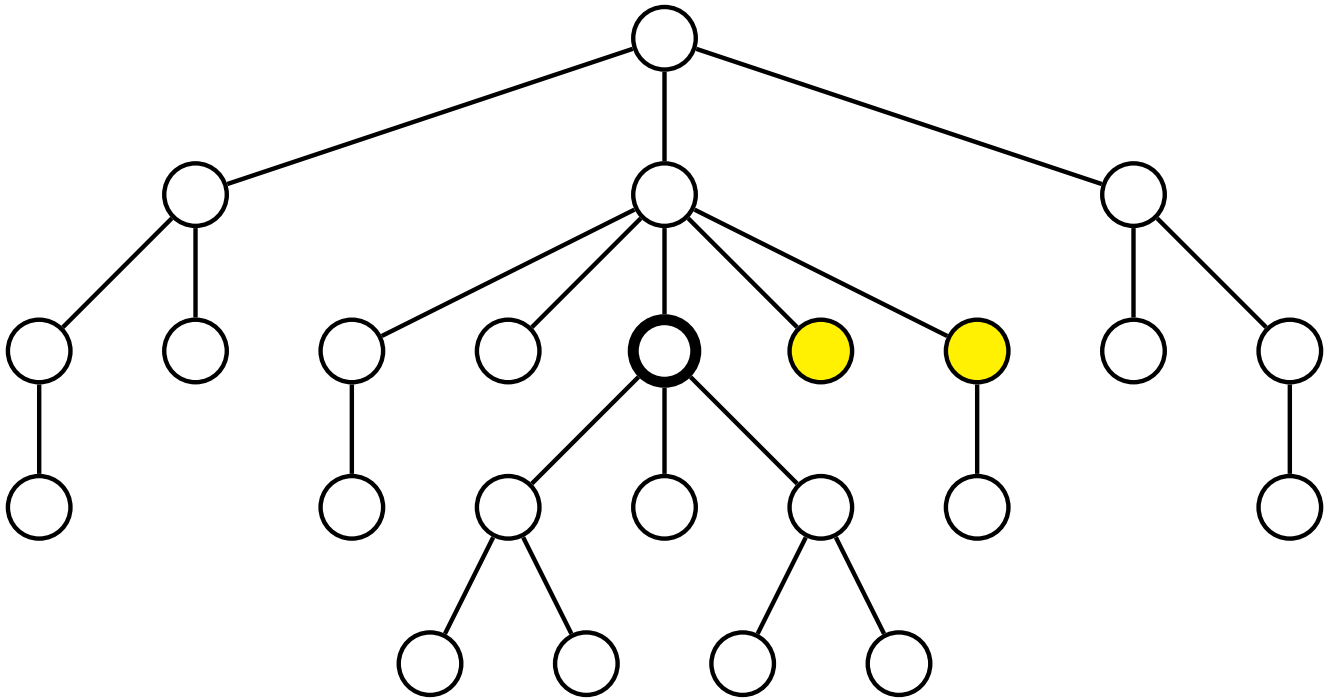
Preceding-Axis

- The preceding-axis contains all nodes that end before the context node begins



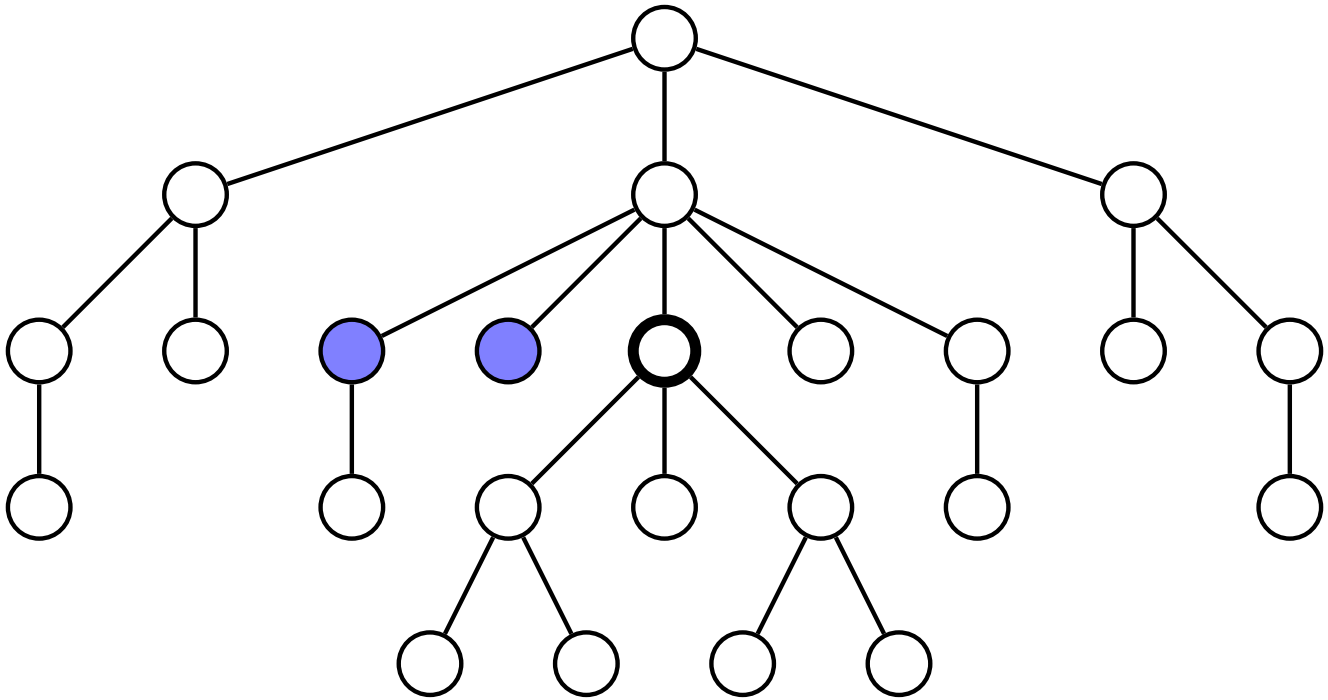
Following-Sibling-Axes

- The following-sibling-axis contains all following nodes that have the same parent as the context node



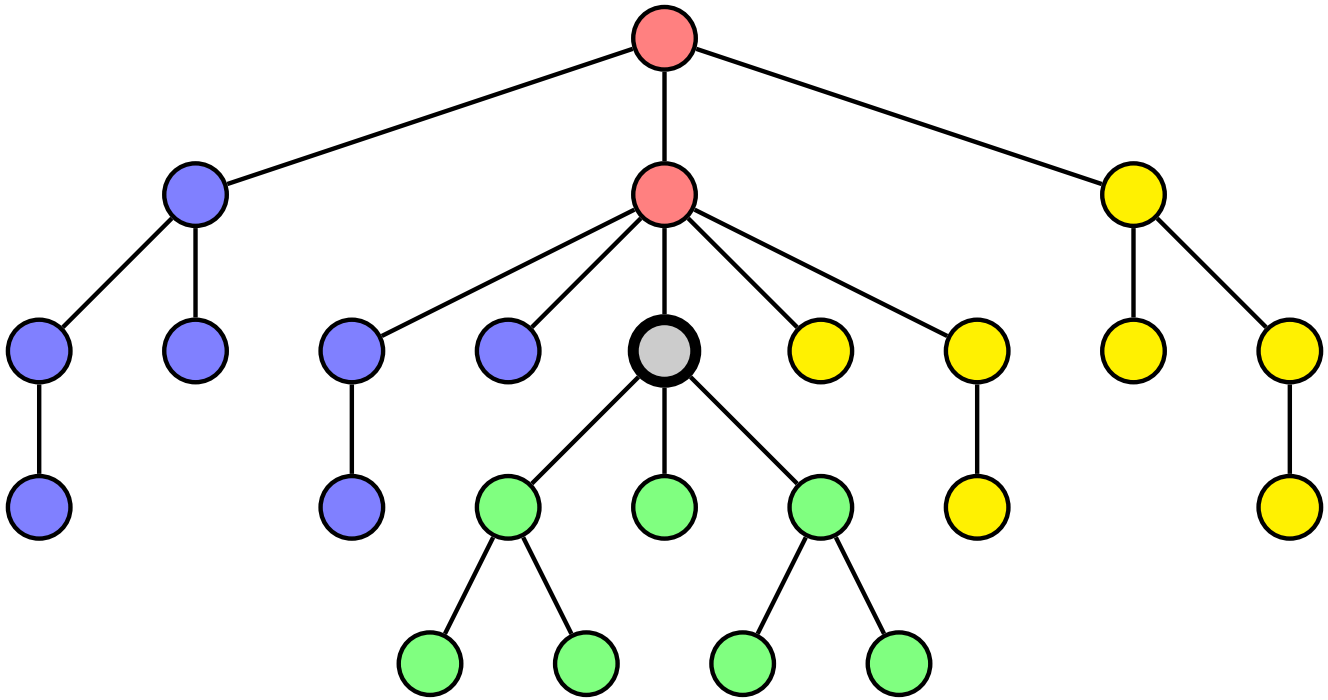
Preceding-Sibling-Axis

- The preceding-sibling-axis contains all preceding nodes that have the same parent as the context node



Partitioning

- The axes self, ancestor, descendant, following and preceding partition a document into five disjoint subtrees:



Attribute-Axis

- Attributes are handled in a special way in XPath
- The attribute-axis contains all the attribute nodes of the context node
- This axis is empty if the context node is not an element node
- Does not contain `xmlns` attributes used to declare namespaces

Namespace-Axis

- The namespace-axis contains all namespaces in scope of the context node
- This axis is empty if the context node is not an element node

Node Tests

- Once the correct relative position of a node has been identified the type of a node can be tested
- A *node test* further refines the nodes selected by the location step
- A double colon `::` separates the axis from the node test
- There are seven different kinds of node tests

name
prefix`:*`
`node()`
`text()`
`comment()`
`processing-instruction()`
`*`

Name

- The *name* node test selects all elements with a matching name
  - e.g., if our context is a set of 4 `CD` elements and the location step uses the `child` axis, then we get element nodes with different names
  - we can use the *name* node test to return, e.g., only `soloist` elements
- Along the attribute-axis it matches all attributes with the same name

Prefix:*

- Along an element axis, all nodes whose namespace URIs are the same as the prefix are matched
- This node test is also available for attribute nodes

Comment, Text, Processing-Instruction

- `comment()` matches all comment nodes
- `text()` matches all text nodes
- `processing-instruction()` matches all processing instructions

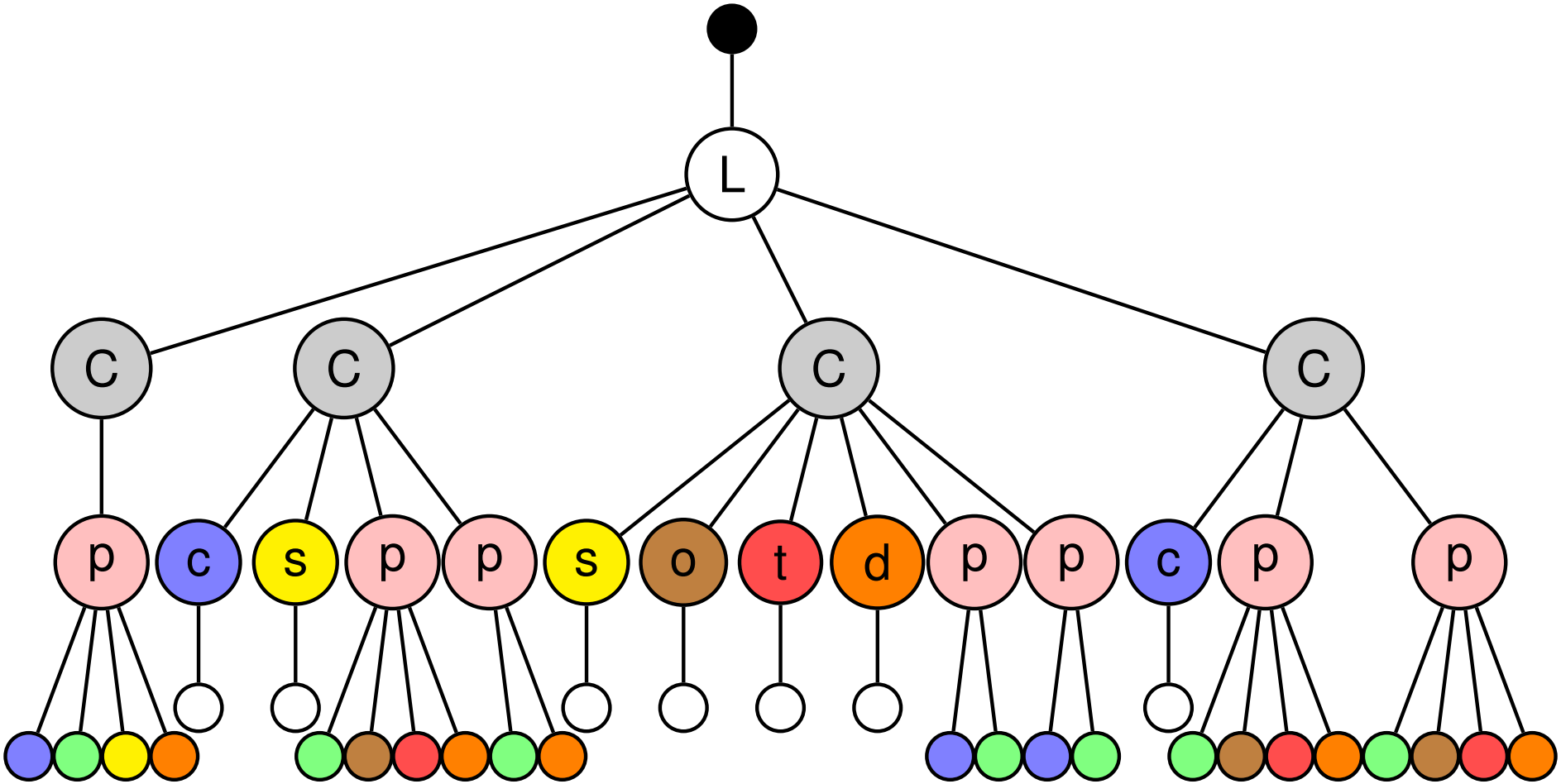
Node and *

- `node()` selects all nodes, regardless of type: attribute, namespace, element, text, comment, processing instruction, and root
- `*` selects all element nodes, regardless of name
  - If the axis is the attribute axis, then it selects all attribute nodes
  - If the axis is the namespace axis, then is selects all namespace nodes

Key for full CD library example

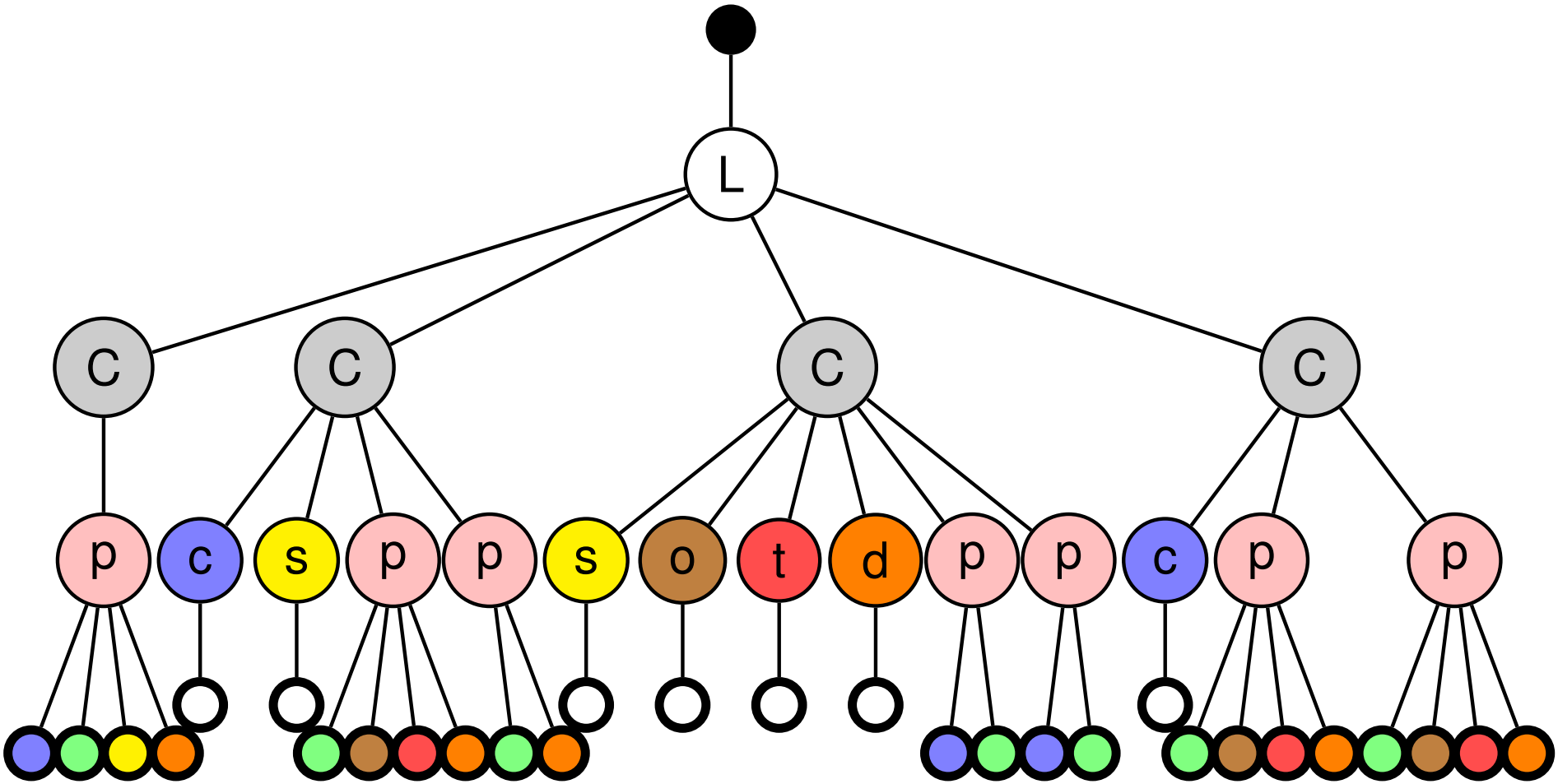
Element name	Abbreviation	Colour
root		black
library	L	white
cd	C	grey
performance	p	pink
composer	c	blue
composition		green
soloist	s	yellow
conductor	t	red
orchestra	o	brown
date	d	orange

Full CD library example



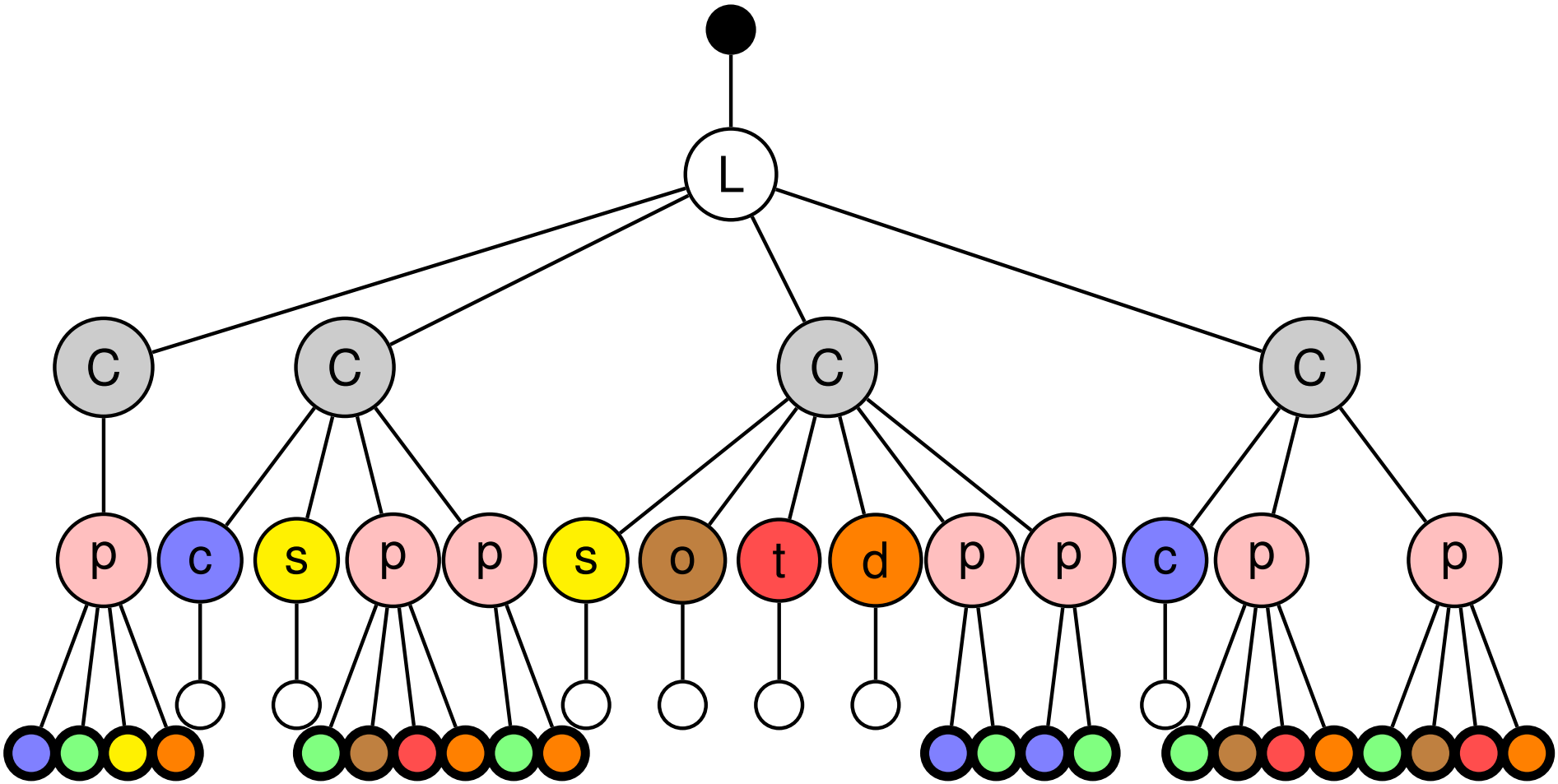
Example using * and node()

```
/CD-library/CD/*/node()
```



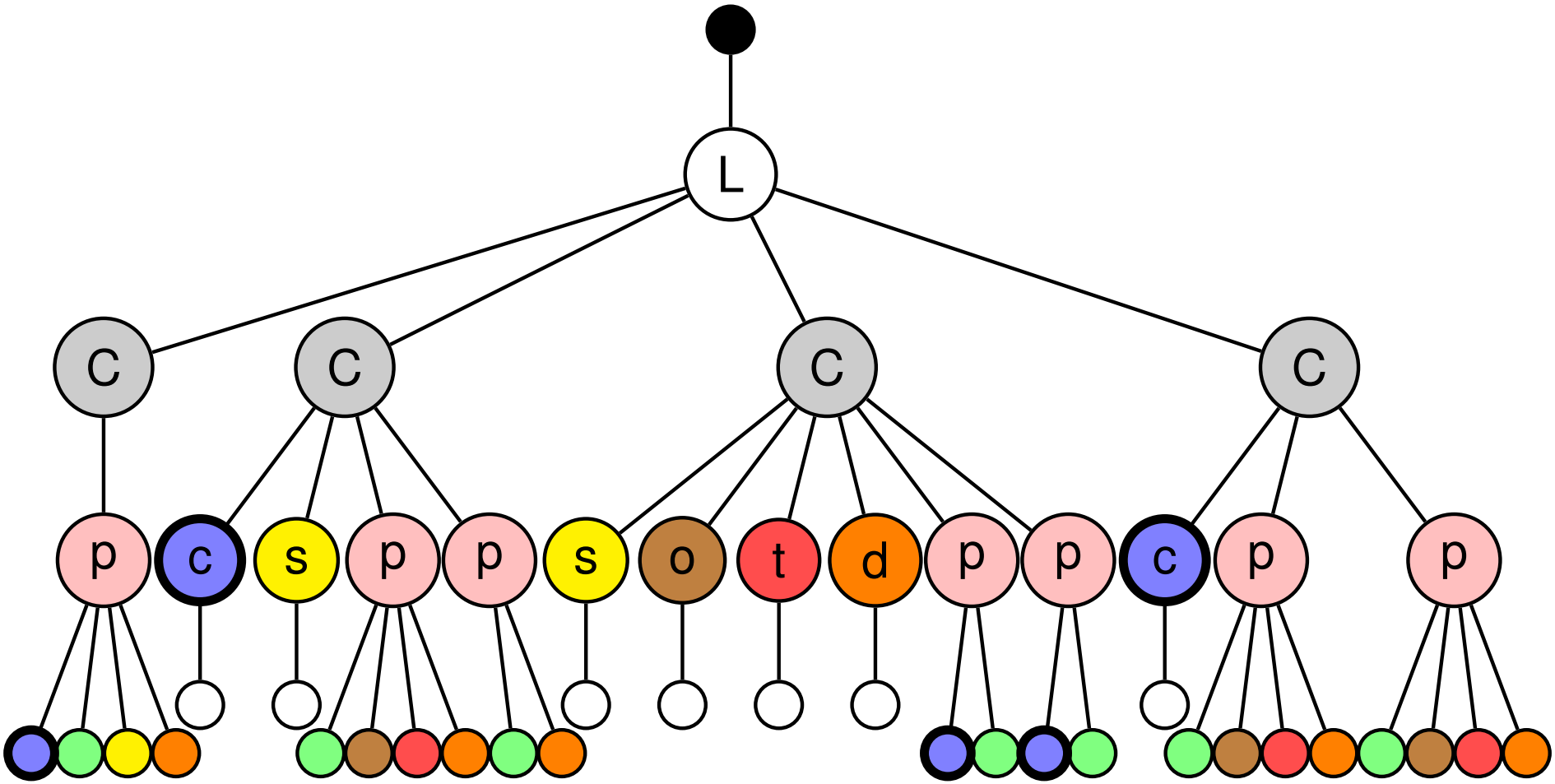
Example showing difference between * and node()

```
/CD-library/CD/*/*
```



Example using descendant

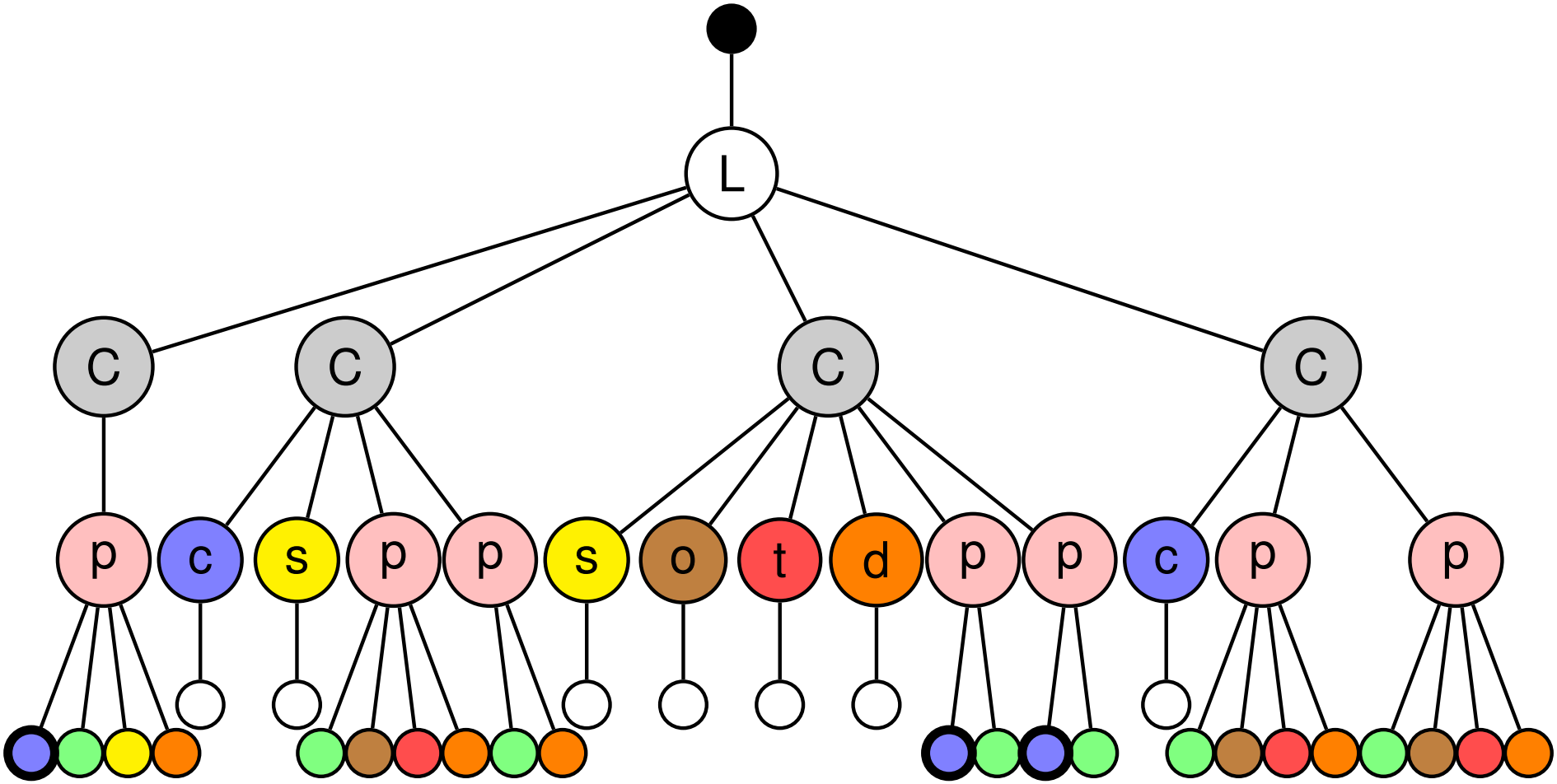
`//composer` or `/descendant-or-self::node()/composer`



Another example using descendant

`//performance/composer` or

`/descendant-or-self::node()/child::composer`



Predicates

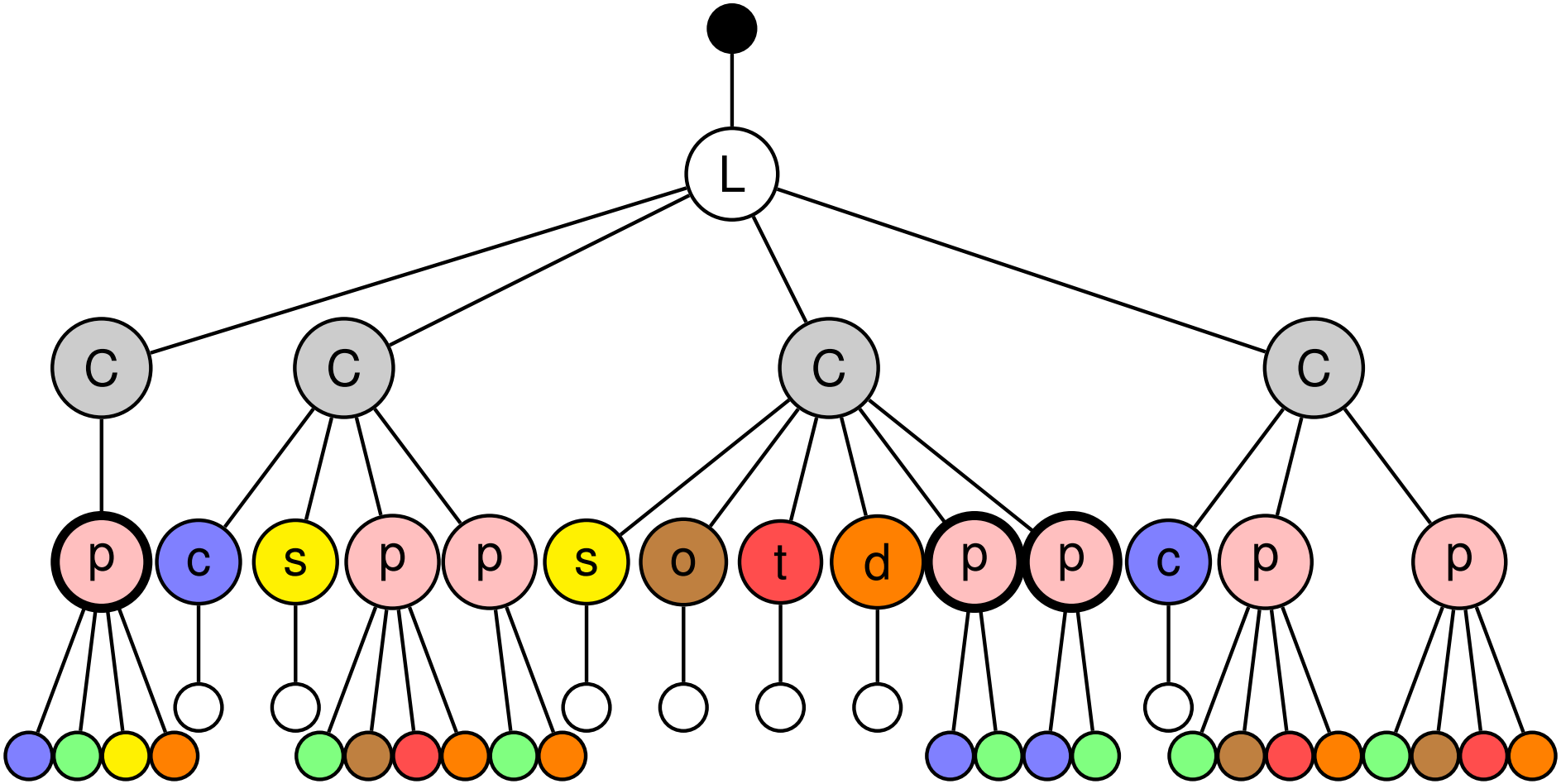
- A node set can be reduced further with *predicates*
- While each location step must have an axis and a node test (which may be empty), a predicate is optional
- A predicate contains a Boolean expression which is tested for each node in the resulting node set
- A predicate is enclosed in square brackets `[ ]`

Predicates (2)

- XPath supports a full complement of relational operators, including =, >, <, >=, <=, !=
- XPath also provides Boolean `and` and `or` operators to combine expressions logically
- In some cases a predicate may not be a Boolean; then XPath will convert it to one implicitly (if that is possible):
  - an empty node set is interpreted as false
  - a non-empty node set is interpreted as true

Example using a predicate

```
//performance[composer]
```



Another example using a predicate

```
//CD[performance/orchestra]
```