Introduction to XPath

Dr James Cummings

Oxford Text Archive

University of Oxford

ahds literature, languages and linguistics

Methods of Access

- So you've created some TEI XML documents, what now?
  - XPath
  - XML Query (XQuery)
  - XSLT Tranformation to another format (HTML, PDF, RTF, CSV, etc.)
  - Custom Applications (Xaira, TEIPubisher, Philologic etc.)

ahds literature, languages and linguistics

What is XPath

- It is a syntax for accessing parts of an XML document
- It uses a path structure to define XML elements
- It has a library of standard functions
- It is a W3C recommendation
- It is one of the main components of XQuery and XSLT

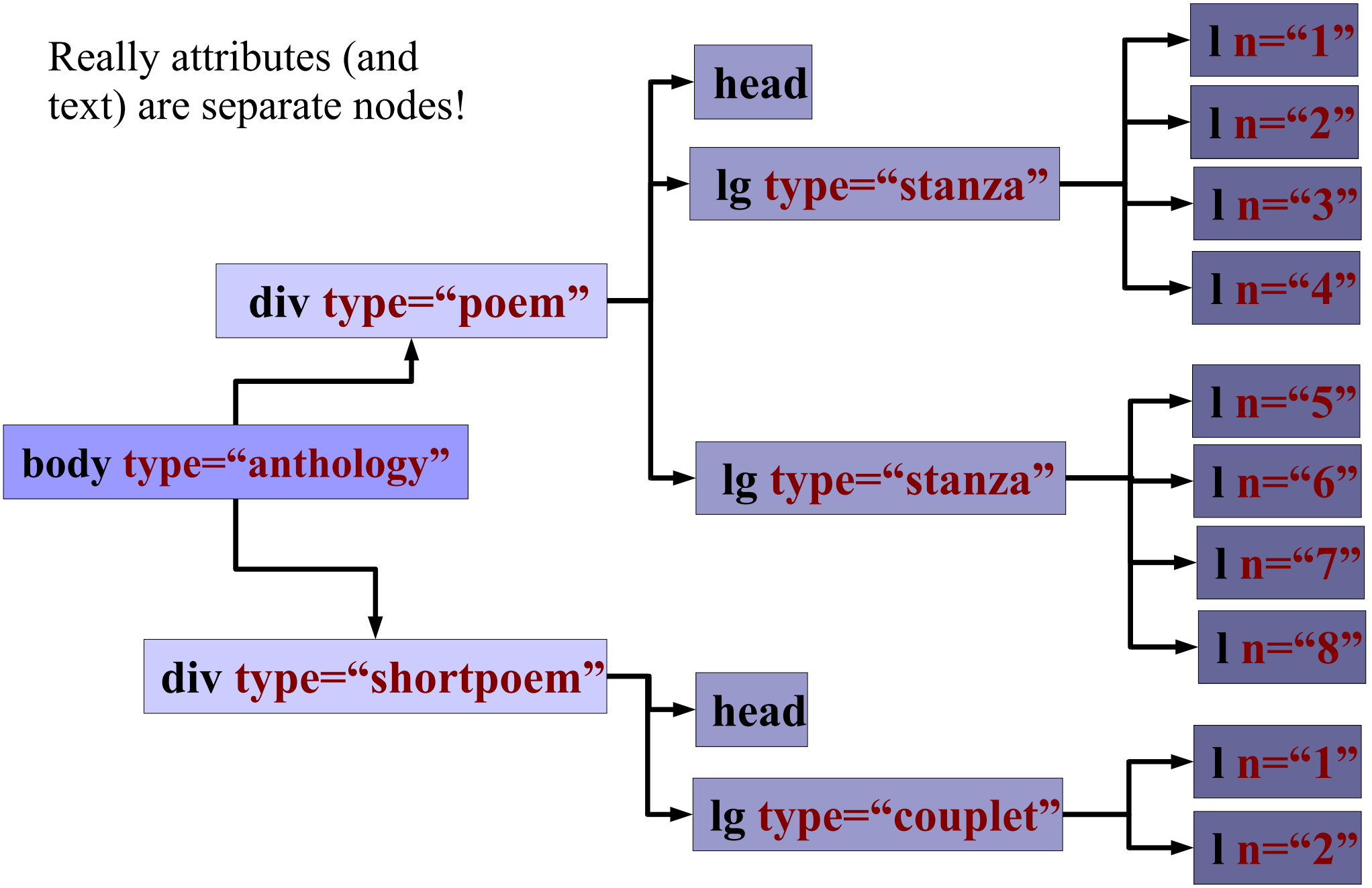
ahds literature, languages and linguistics

XML Example

```
<body type="anthology">
    <div type= "poem"><head>The SICK ROSE</head>
        <lg type="stanza">
            <l n="1">O Rose thou art sick.</l>
            <l n="2">The invisible worm,</l>
            <l n="3">That flies in the night</l>
            <l n="4">In the howling storm:</l>
        </lg>
        <lg type="stanza">
            <l n="5">Has found out thy bed</l>
            <l n="6">Of crimson joy:</l>
            <l n="7">And his dark secret love</l>
            <l n="8">Does thy life destroy.</l>
        </lg>
    </div>          <!-- more poems go here    -->
</body>
```

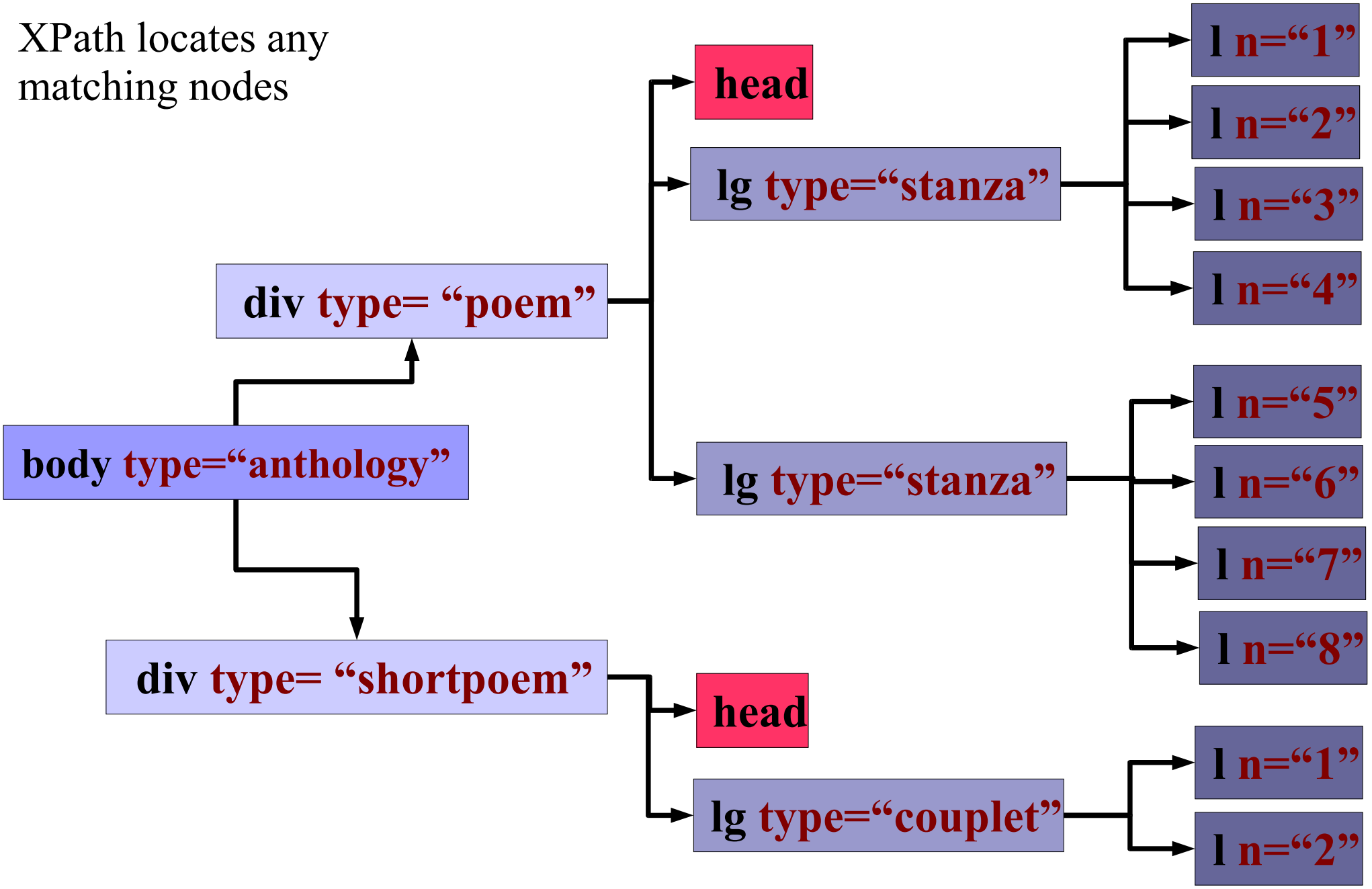
XML Structure

Really attributes (and text) are separate nodes!

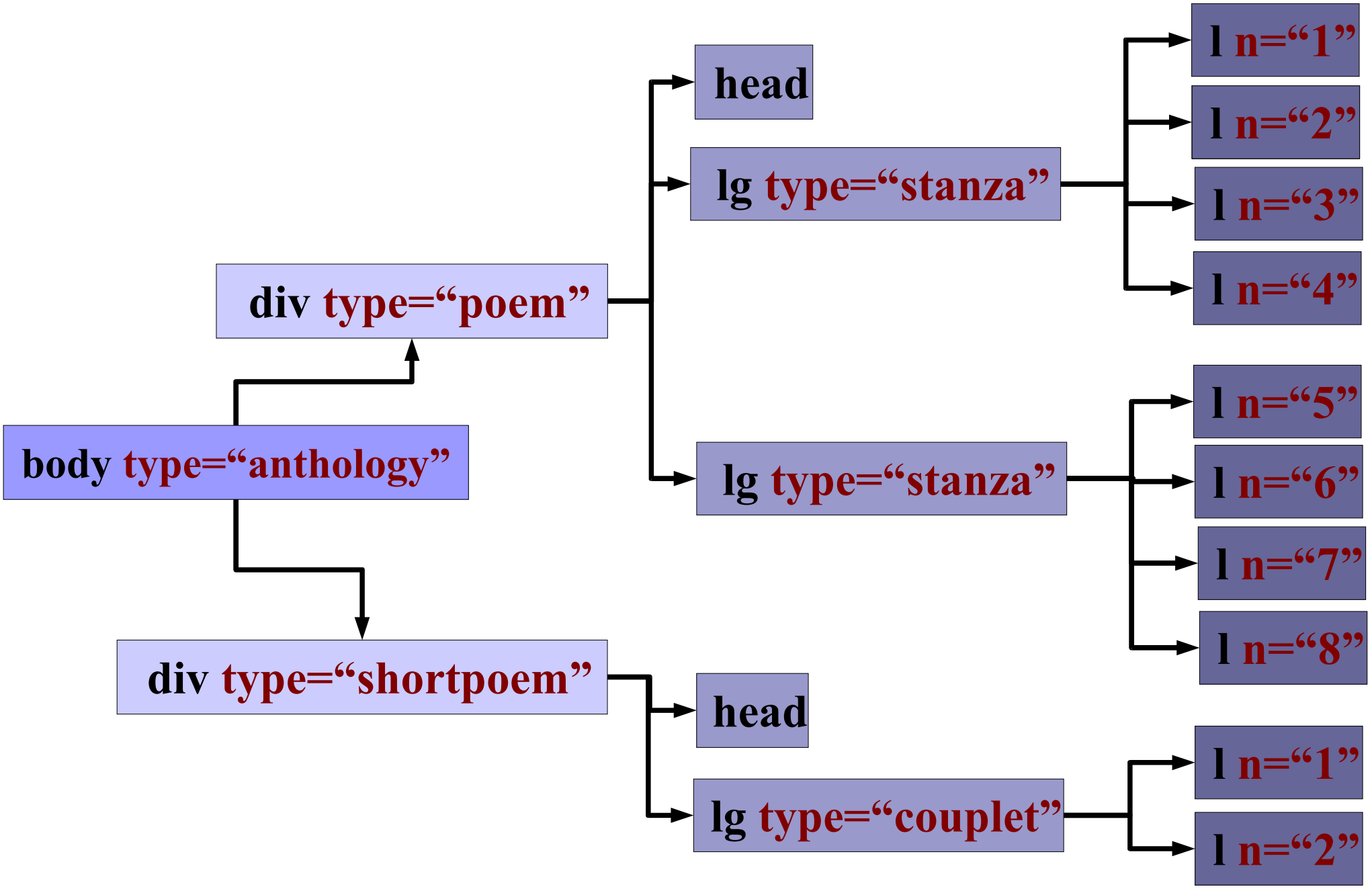


/body/div/head

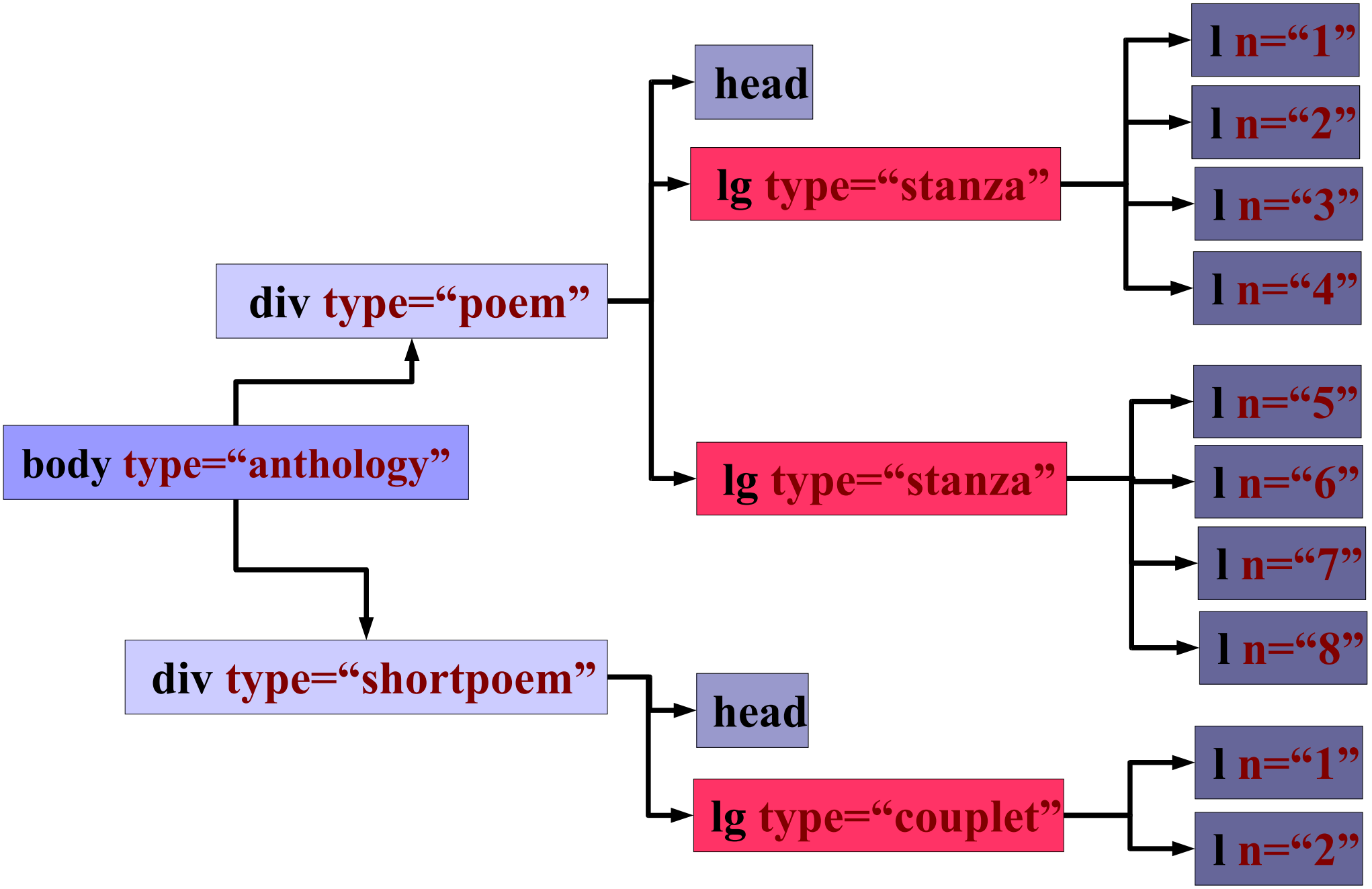
XPath locates any matching nodes



/body/div/lg ?

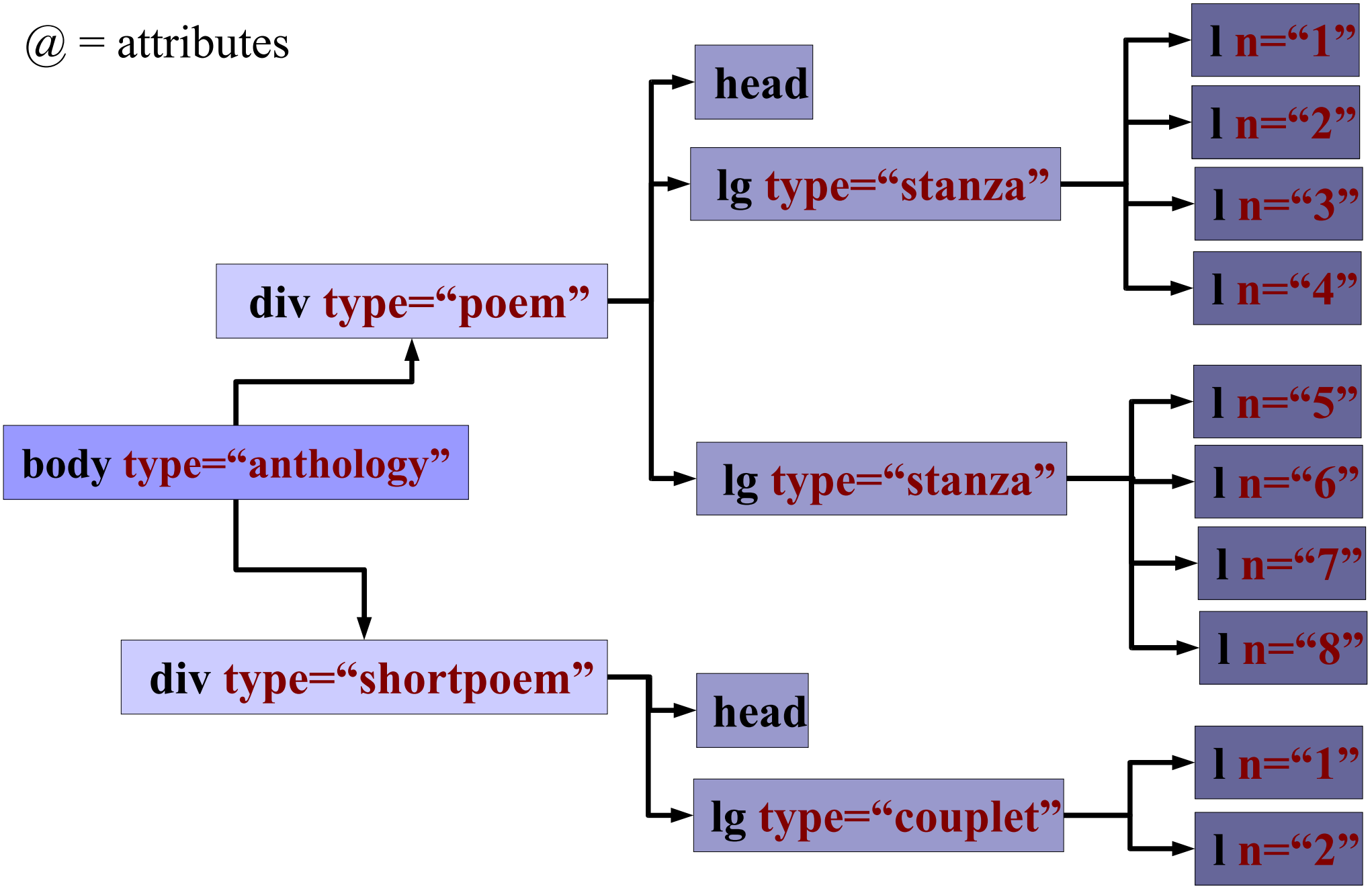

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

/body/div/lg

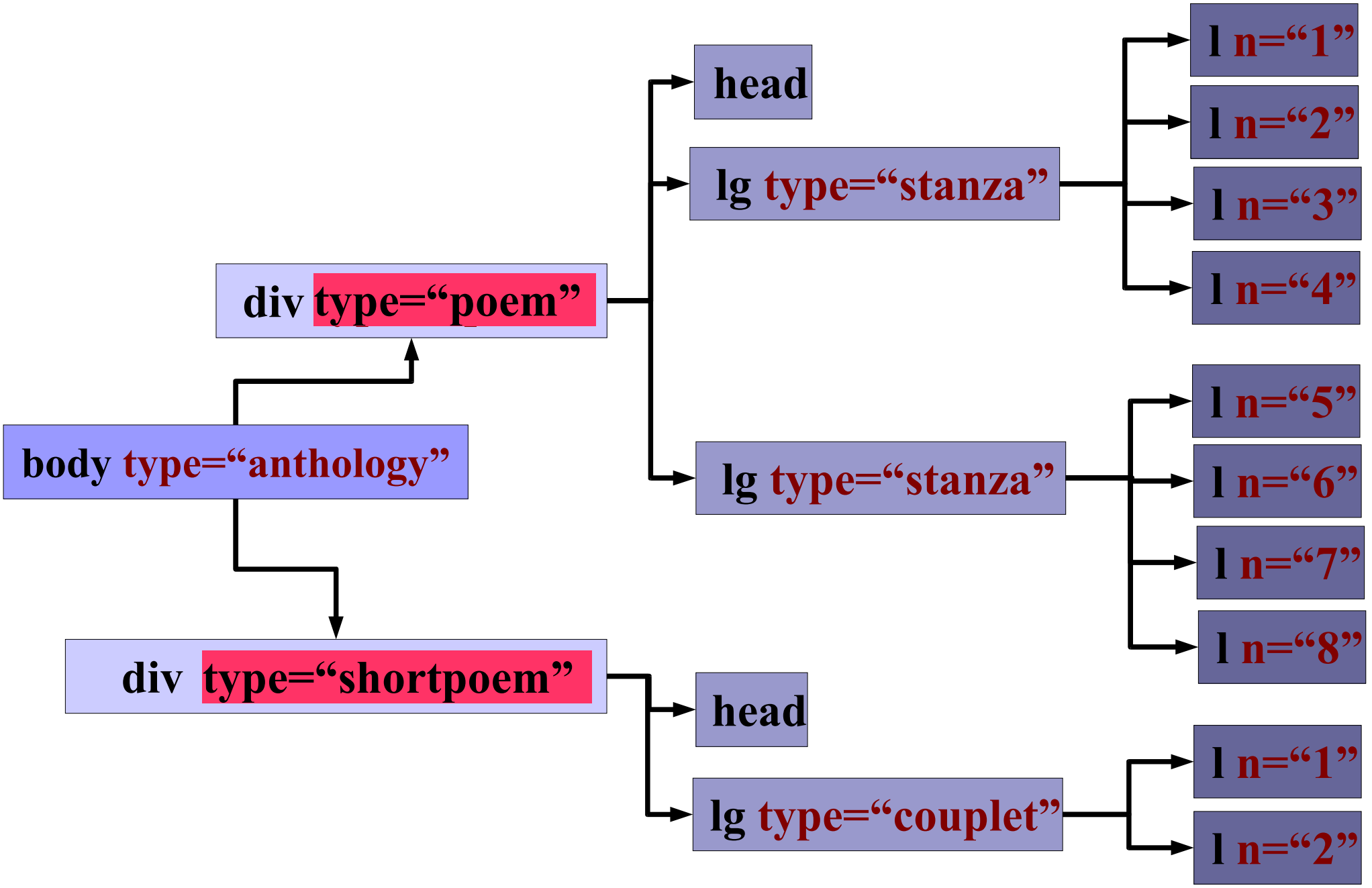

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

/body/div/@type ?

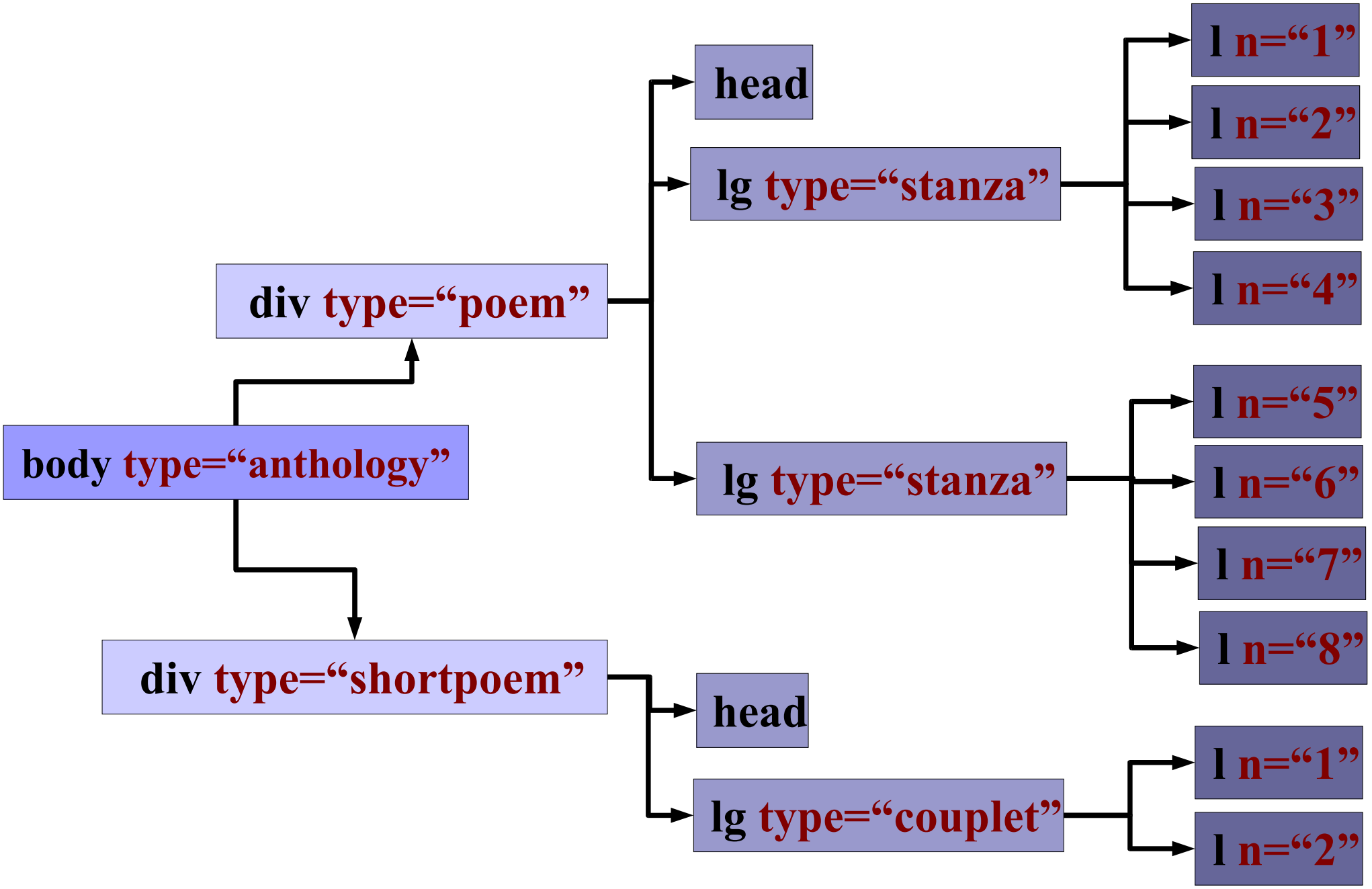
@ = attributes



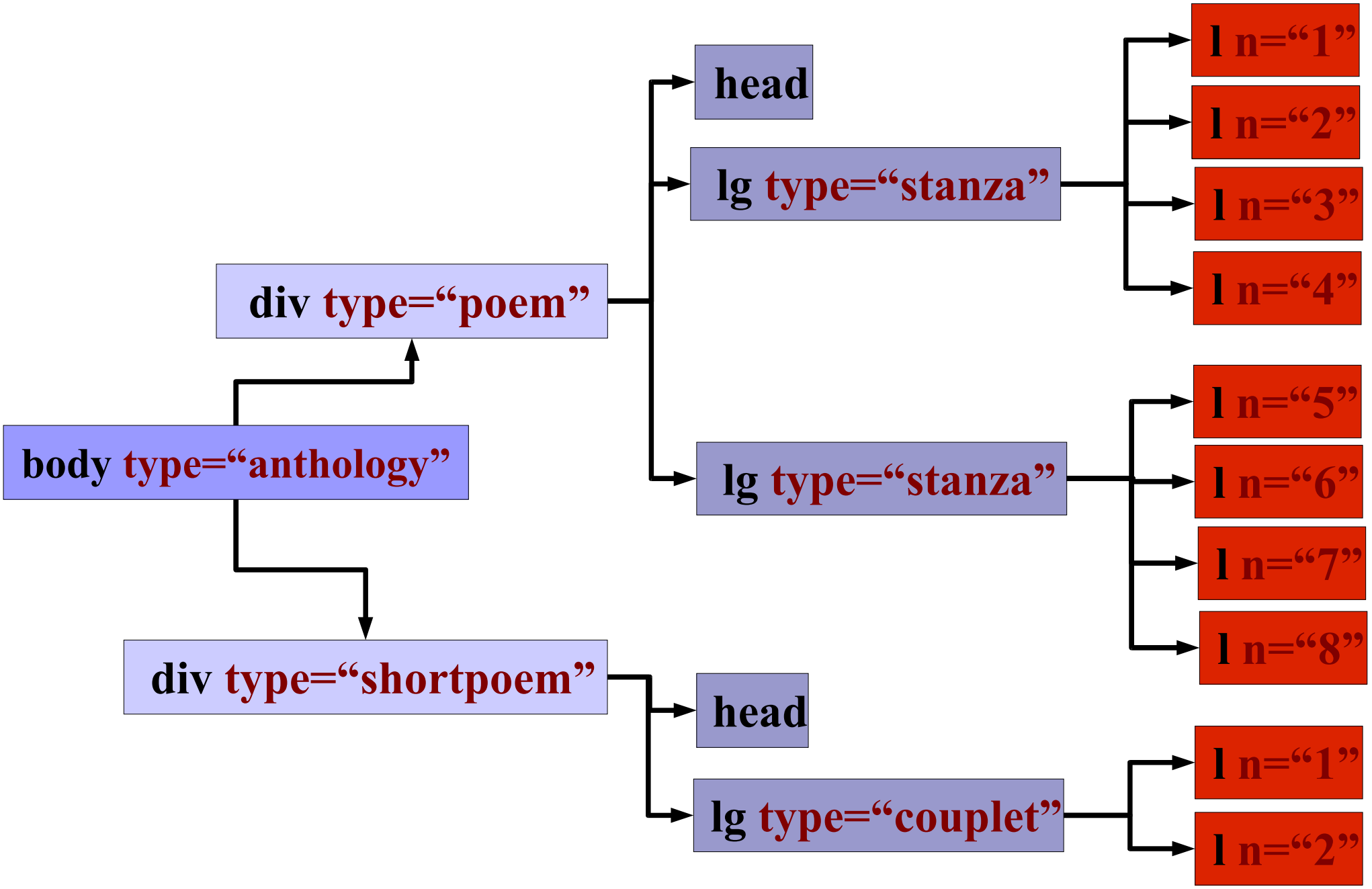
/body/div/@type



/body/div/lg/l ?

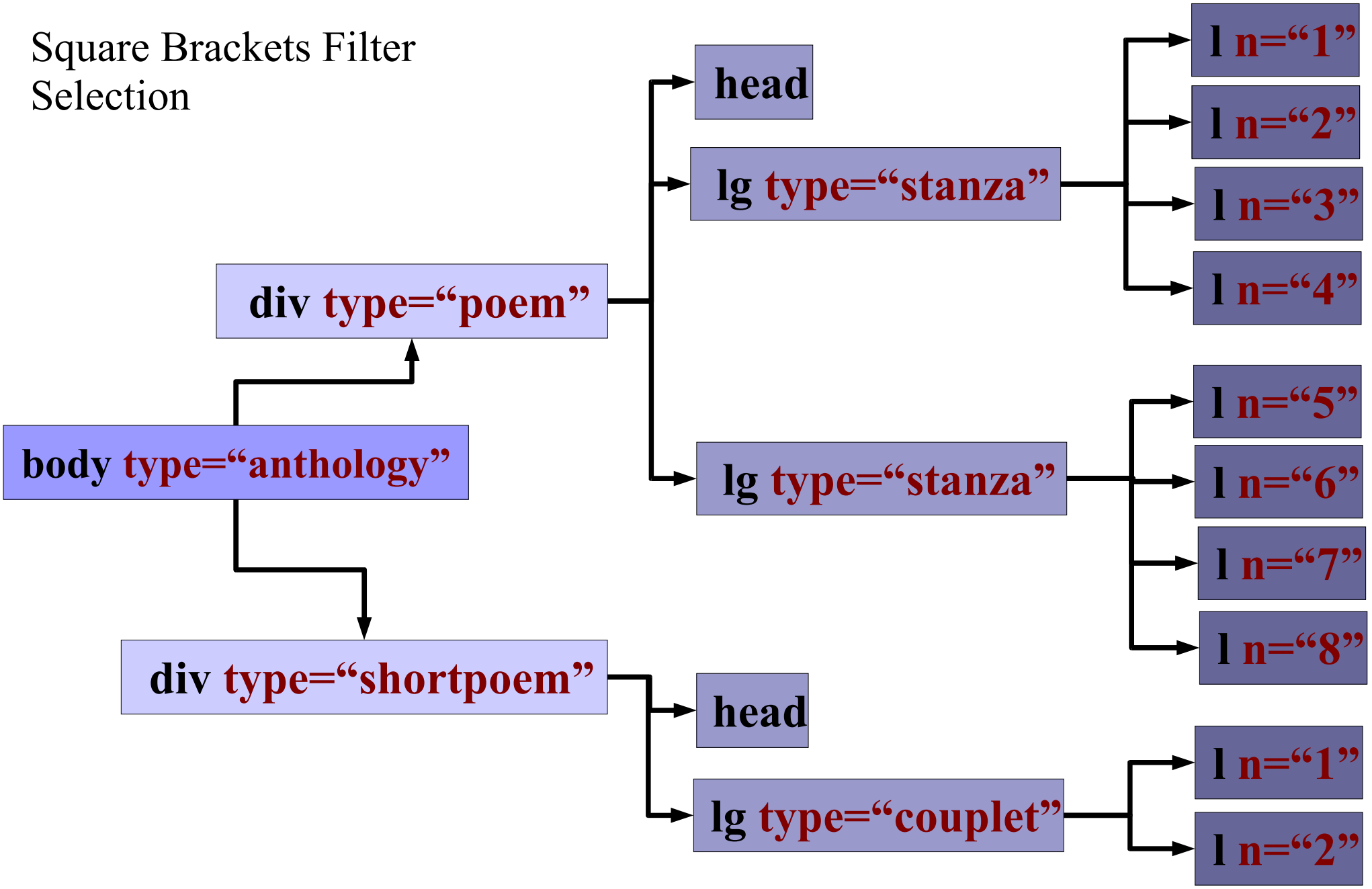

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

/body/div/lg/l

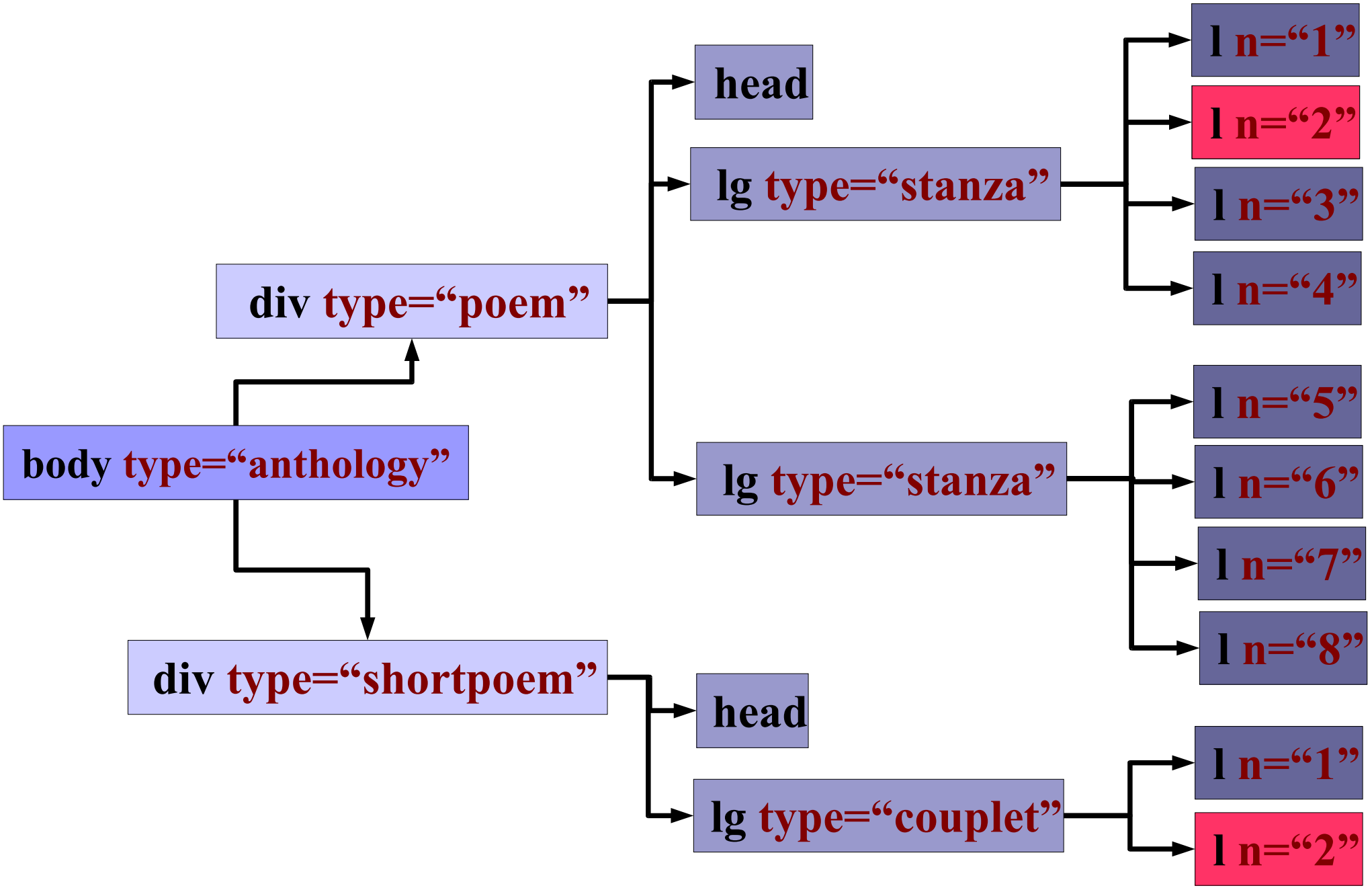

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

/body/div/lg/l[@n="2"] ?

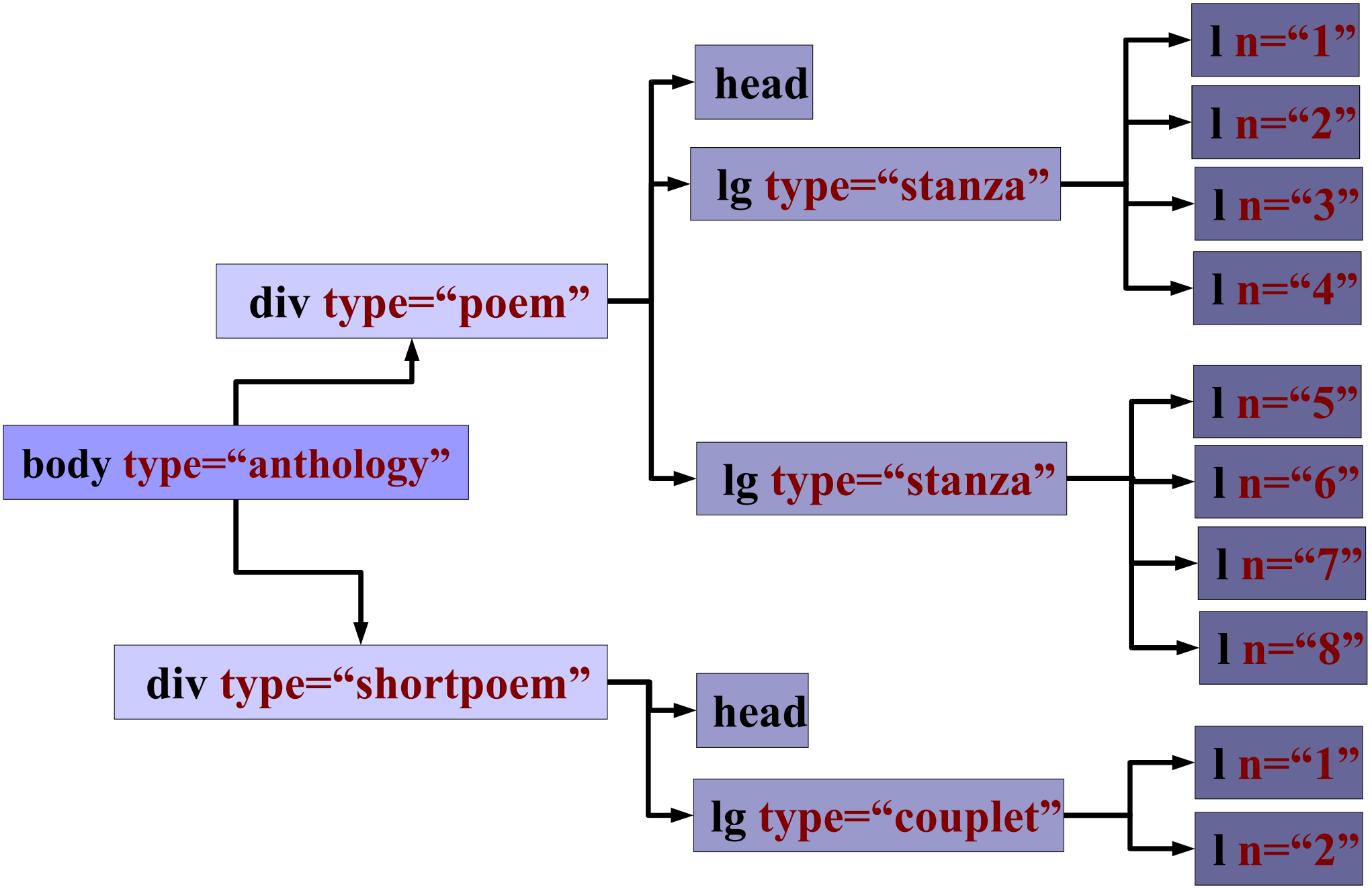
Square Brackets Filter
Selection



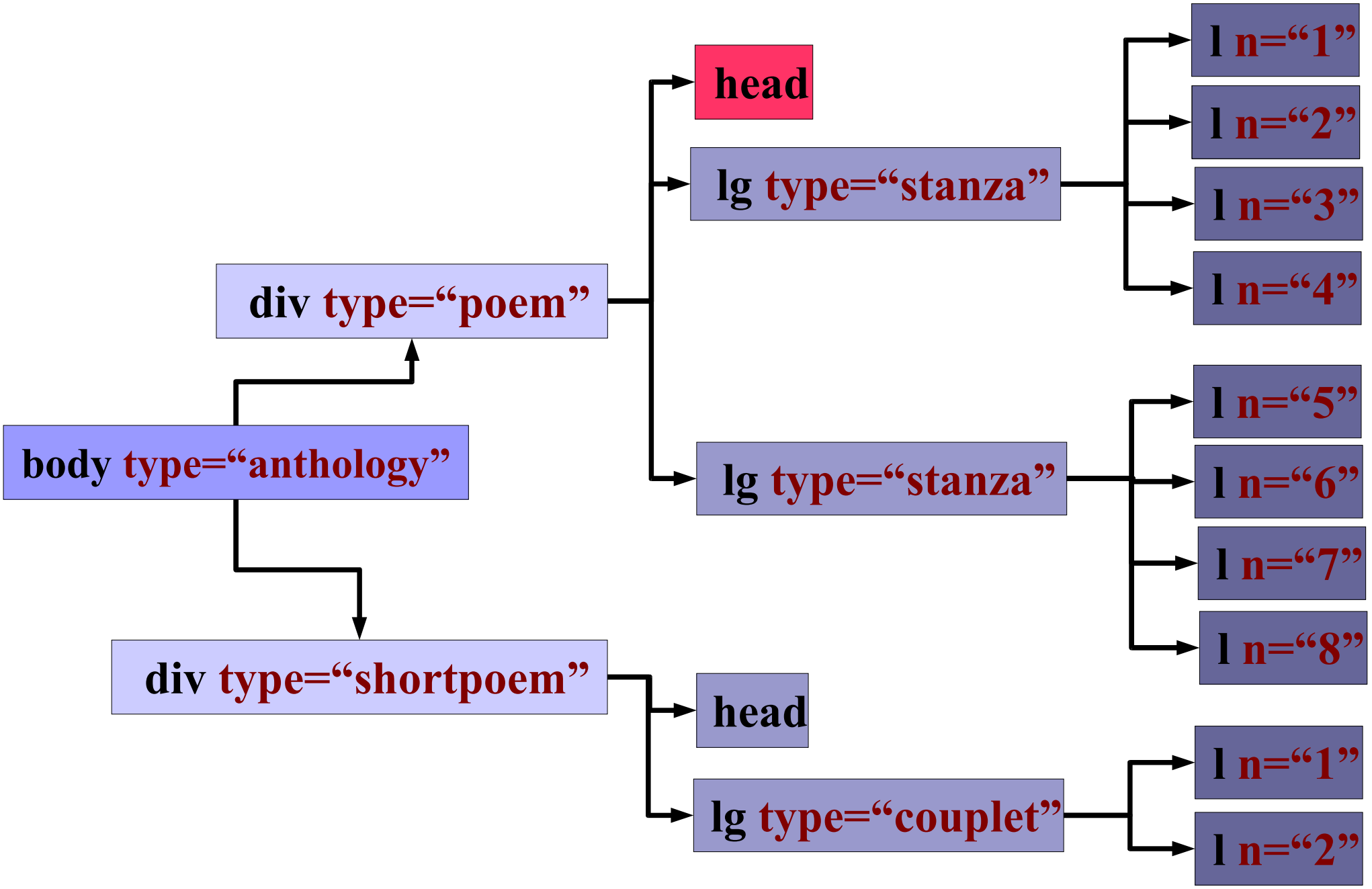
/body/div/lg/l[@n="2"]



/body/div[@type="poem"]/head ?

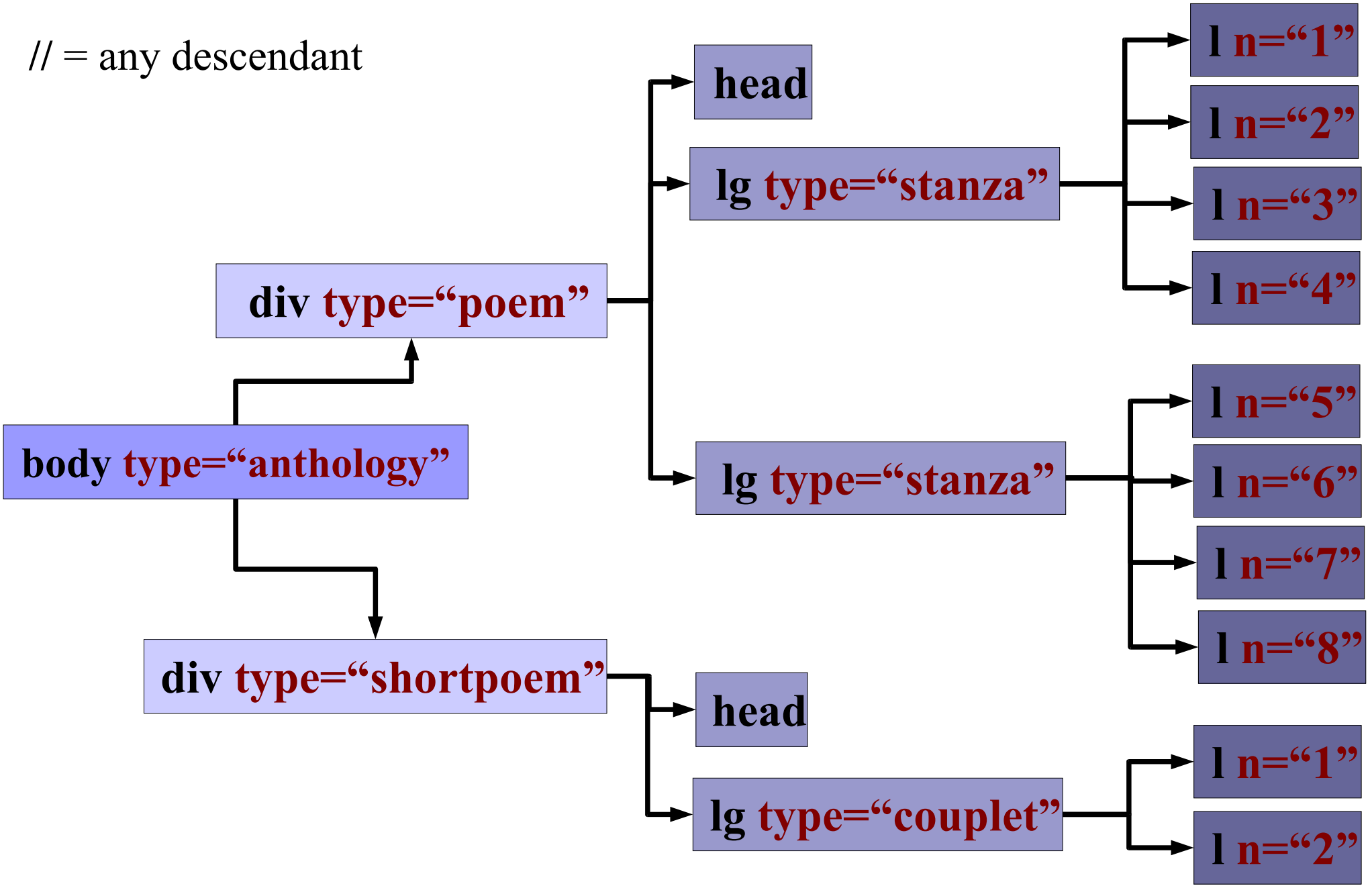


/body/div[@type="poem"]/head

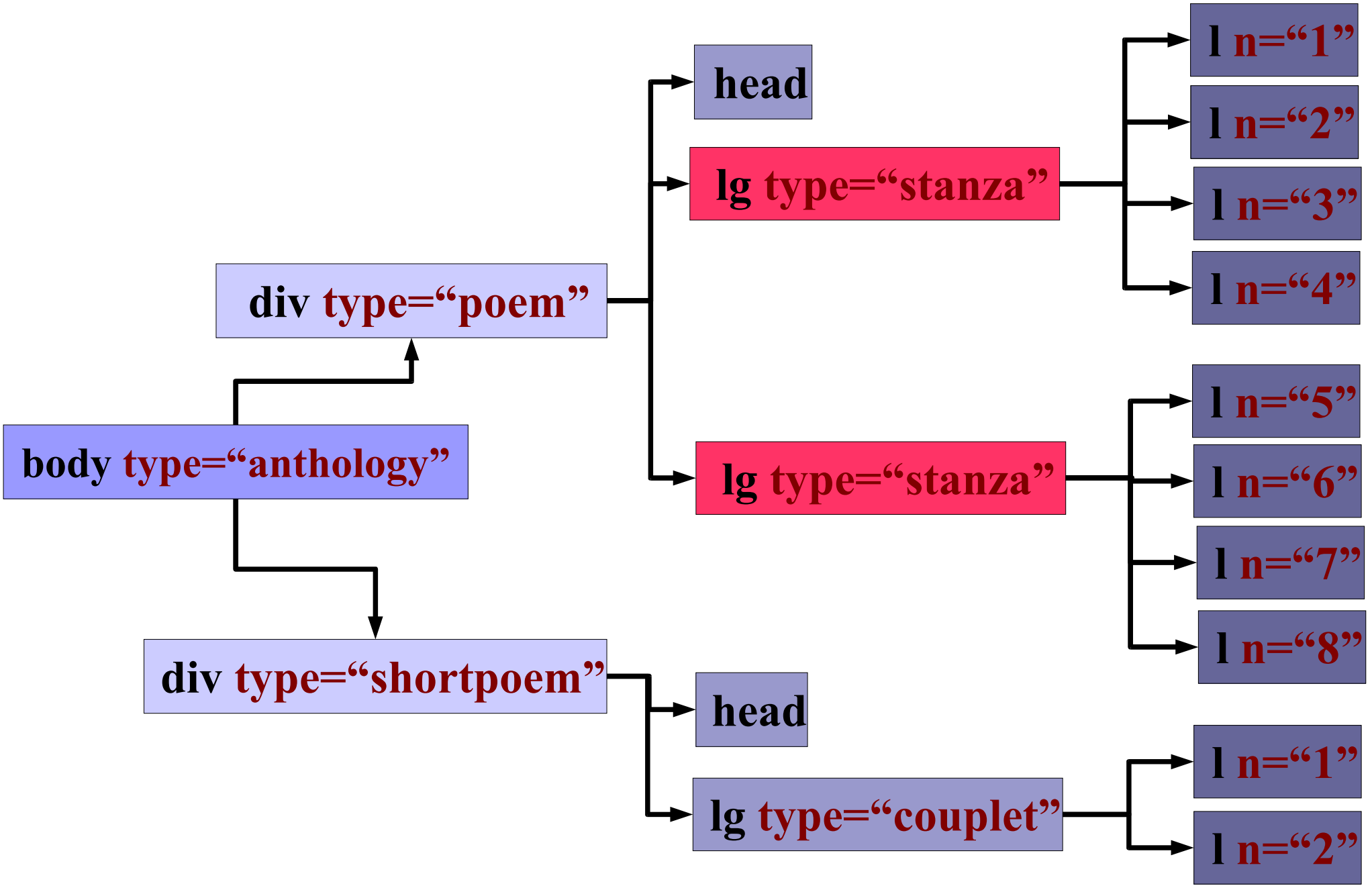


//lg[@type="stanza"] ?

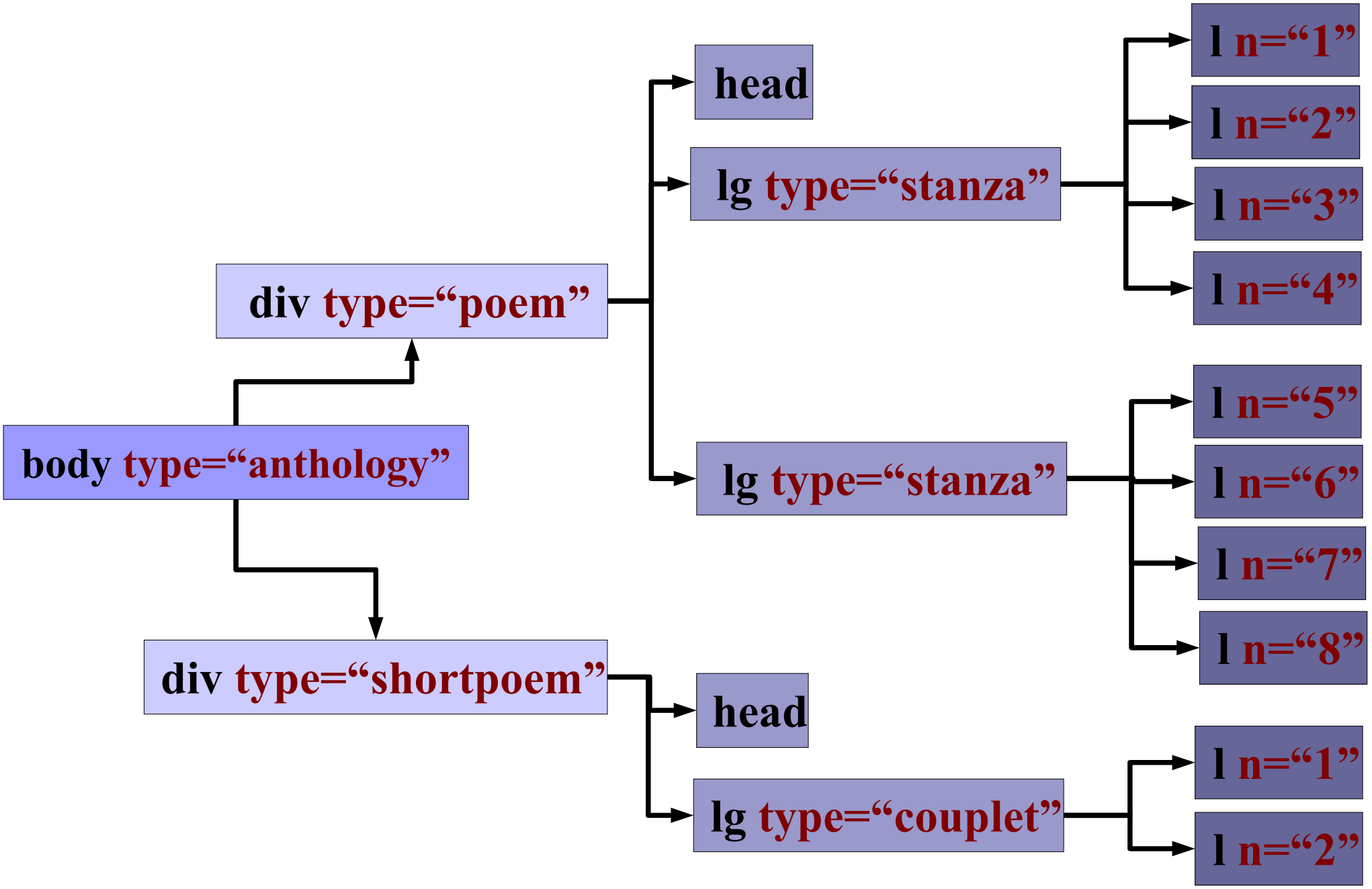
// = any descendant



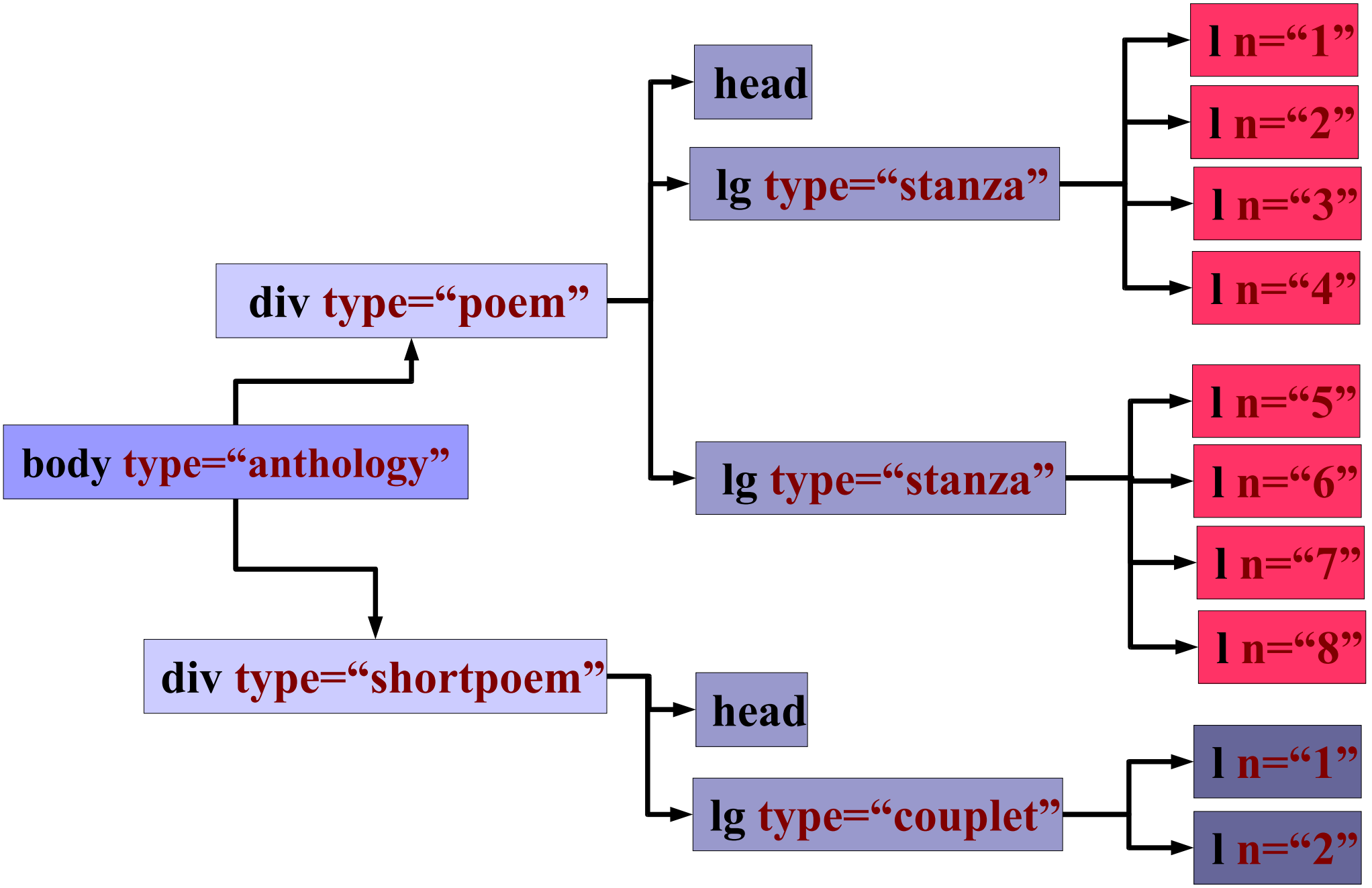
//lg[@type="stanza"]



//div[@type="poem"]//l ?

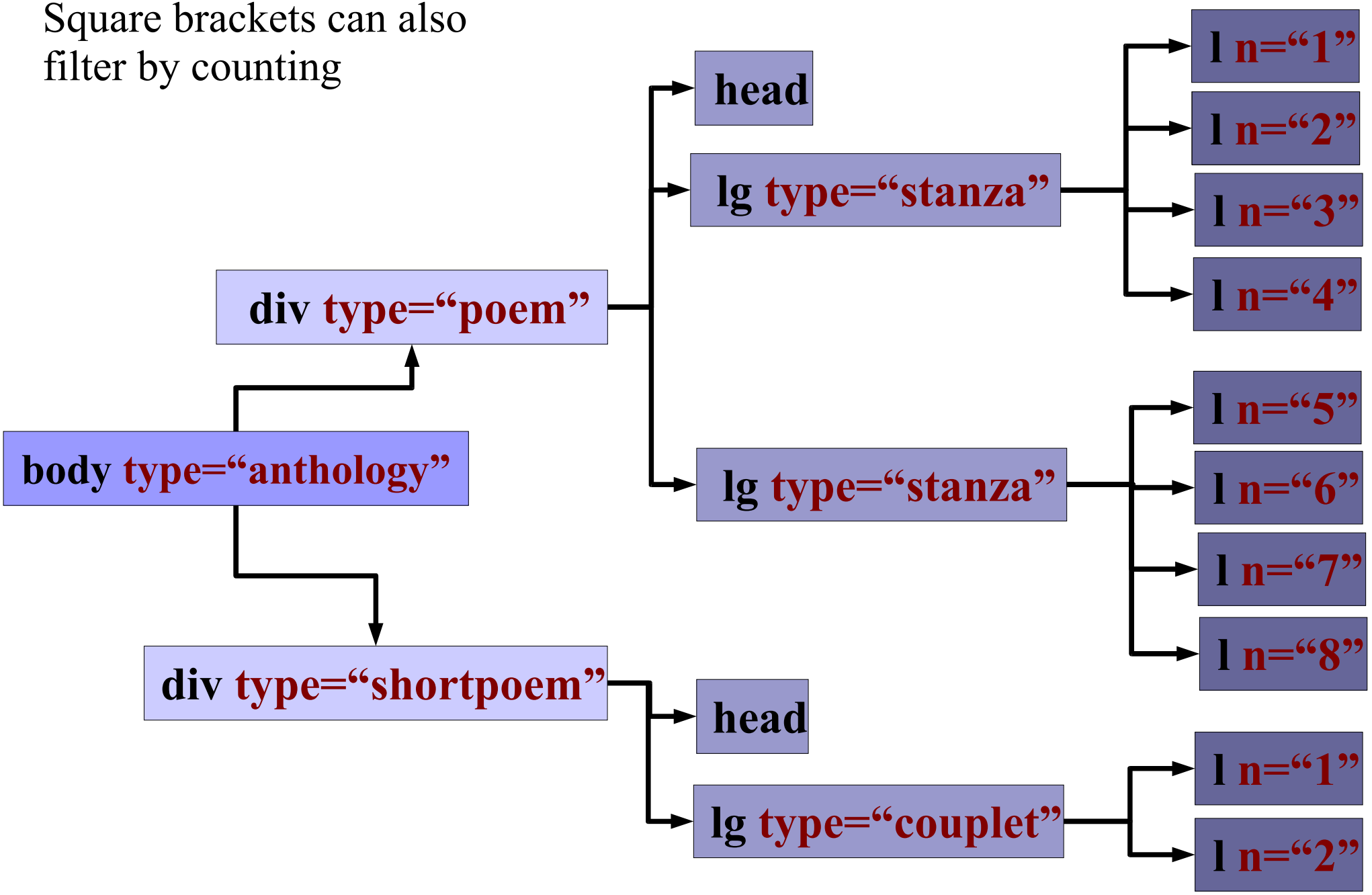


//div[@type="poem"]//l

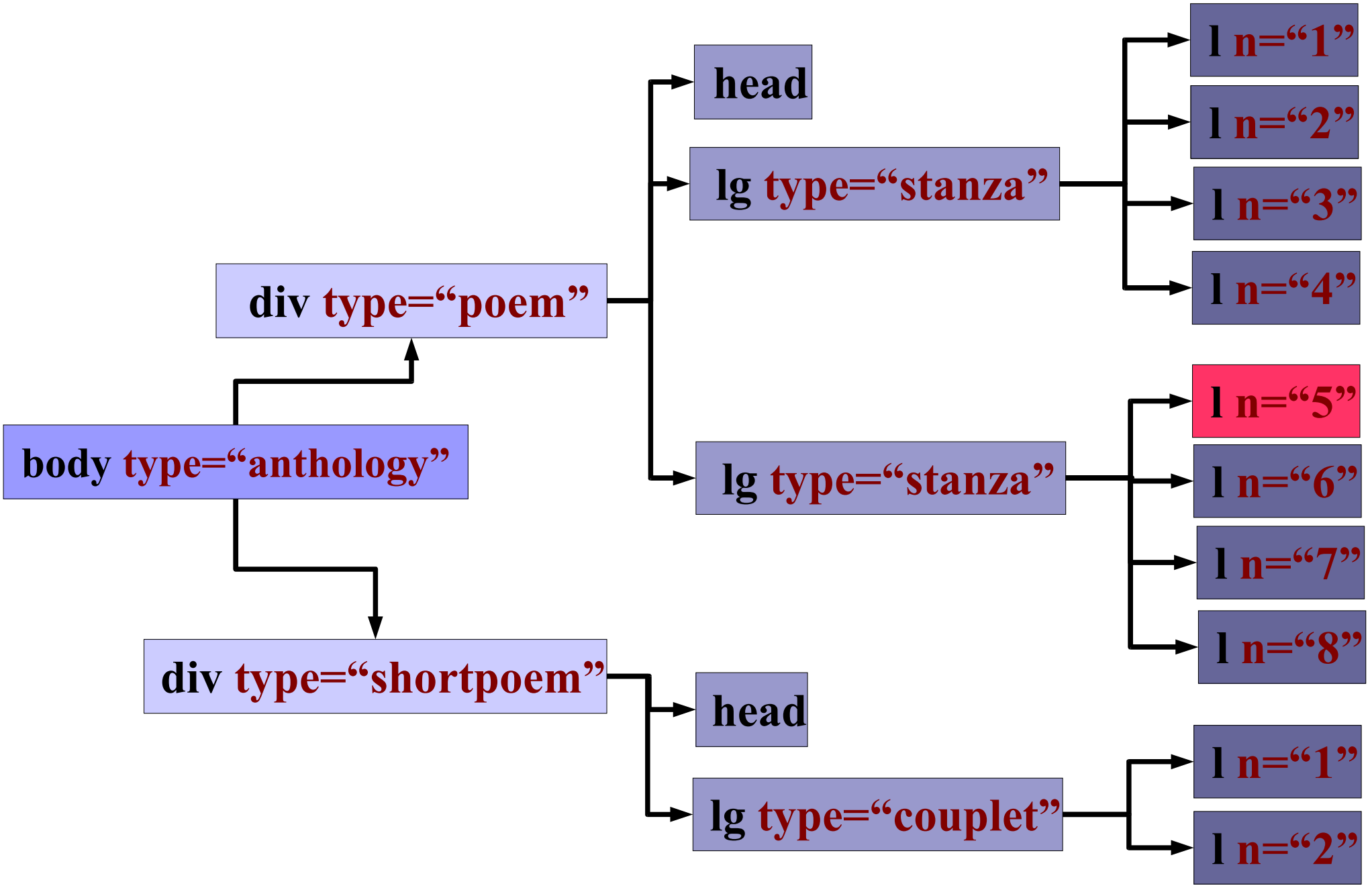


//l[5] ?

Square brackets can also filter by counting



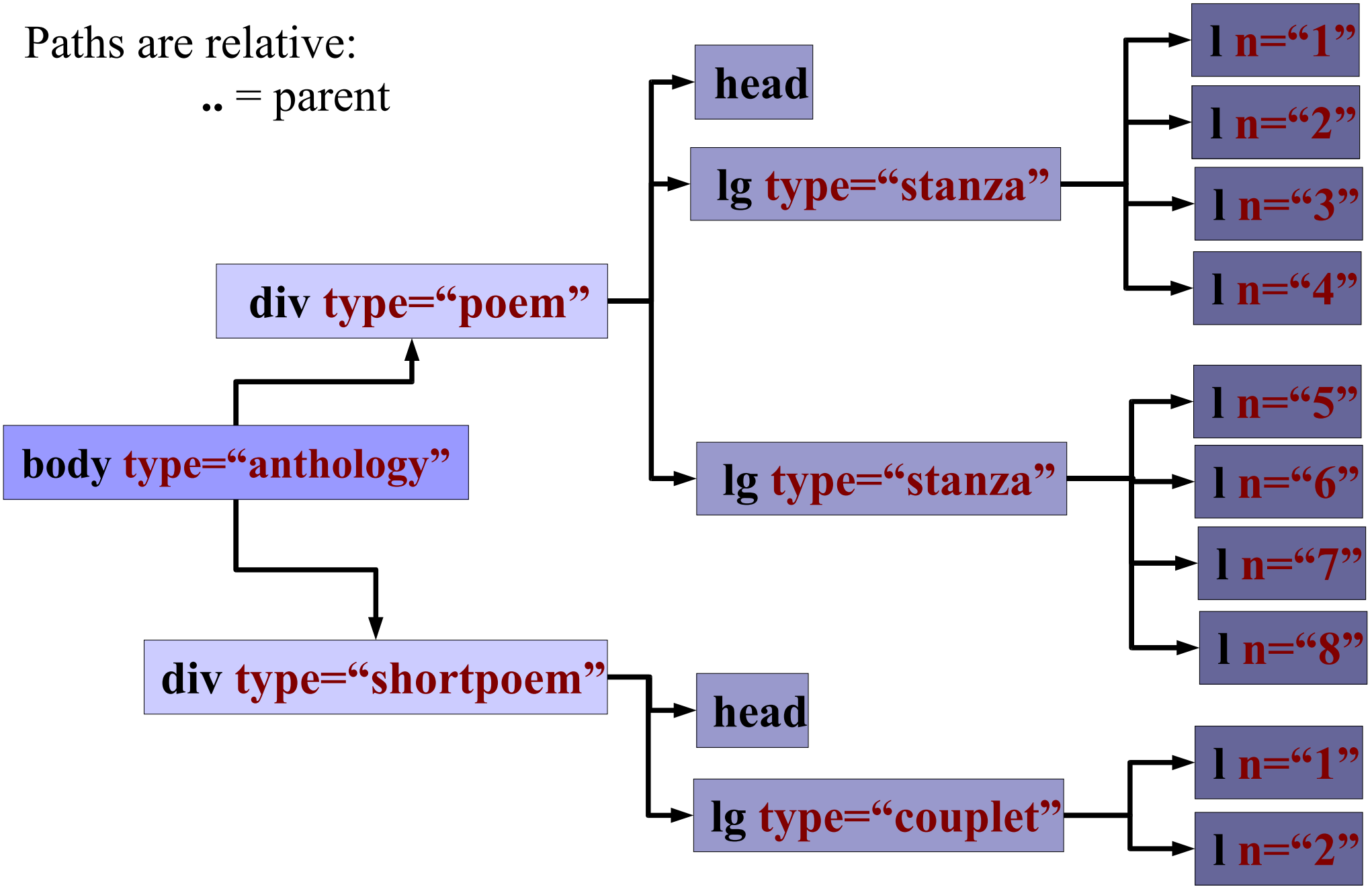
//l[5]


body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

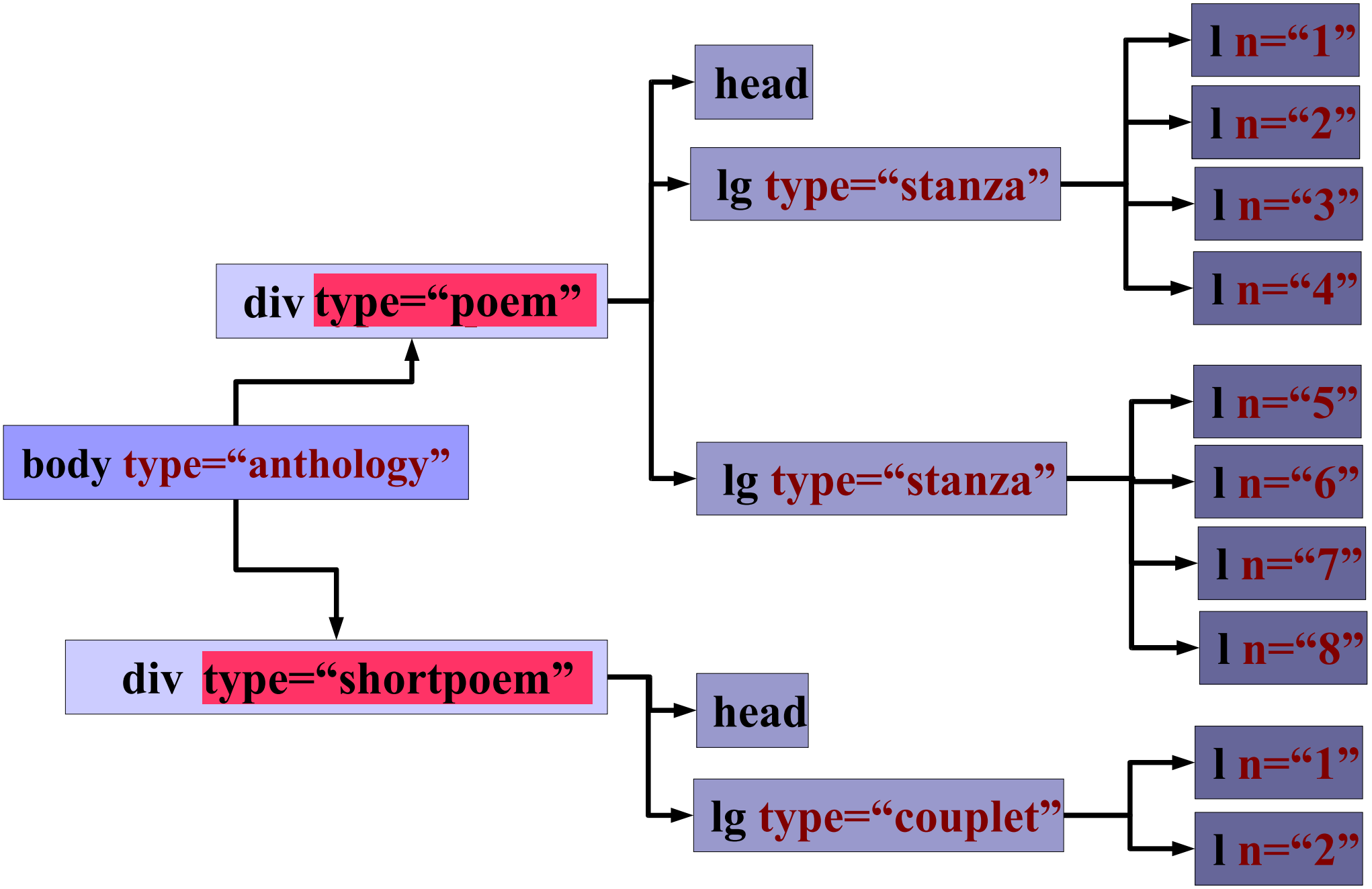
//lg/../@type ?

Paths are relative:

.. = parent

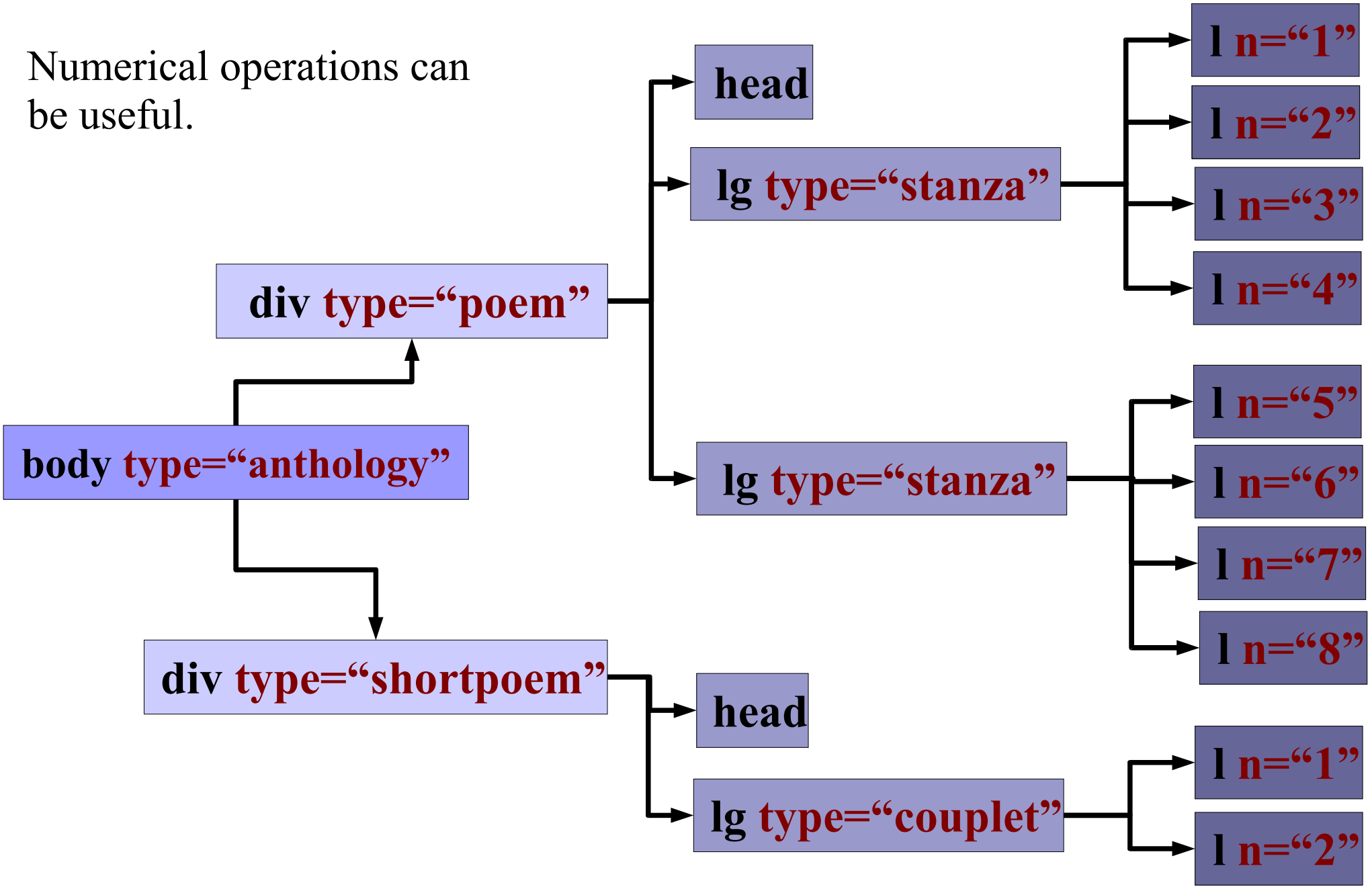


//lg/../@type

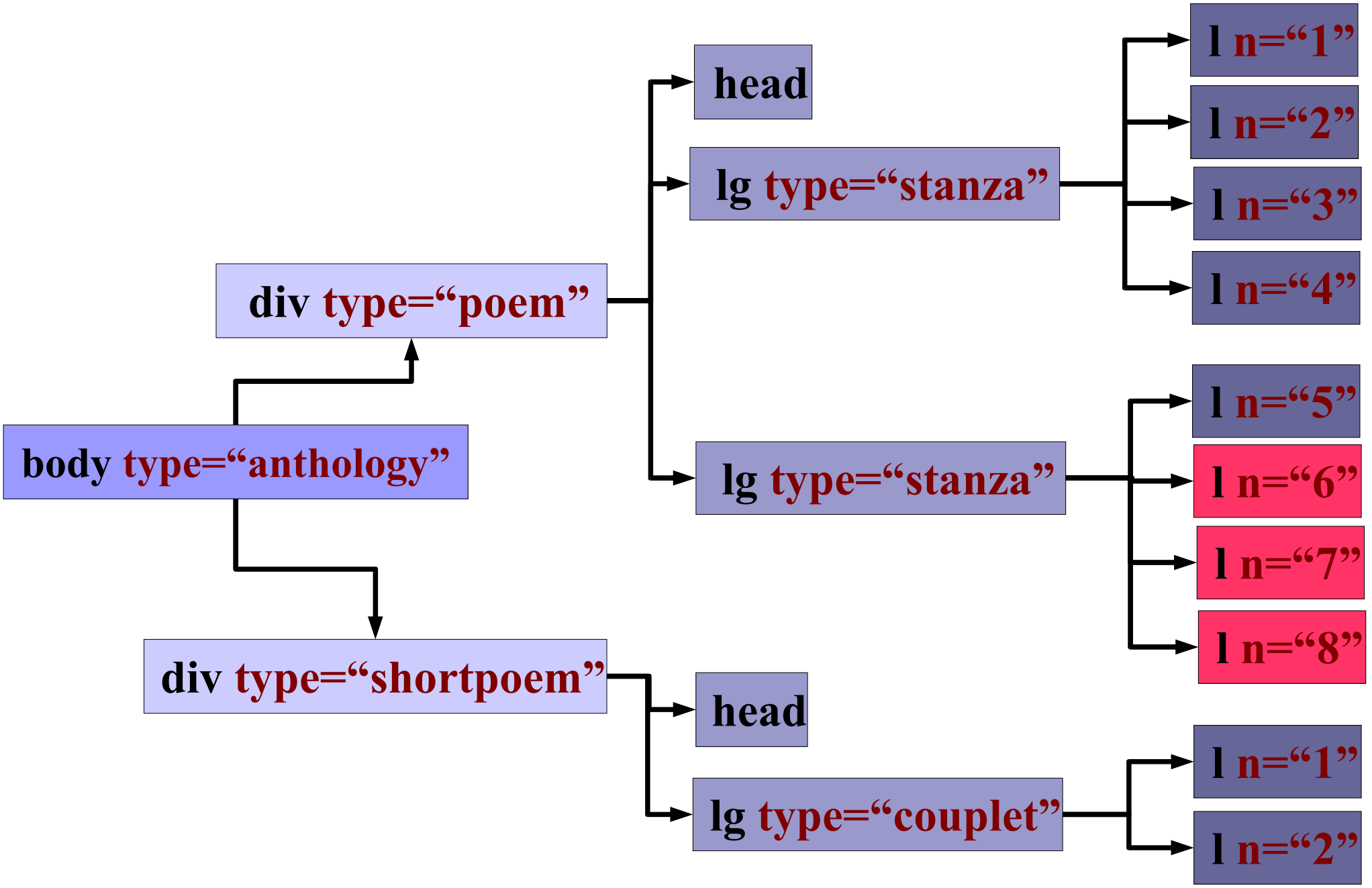

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

//l[@n > 5] ?

Numerical operations can be useful.

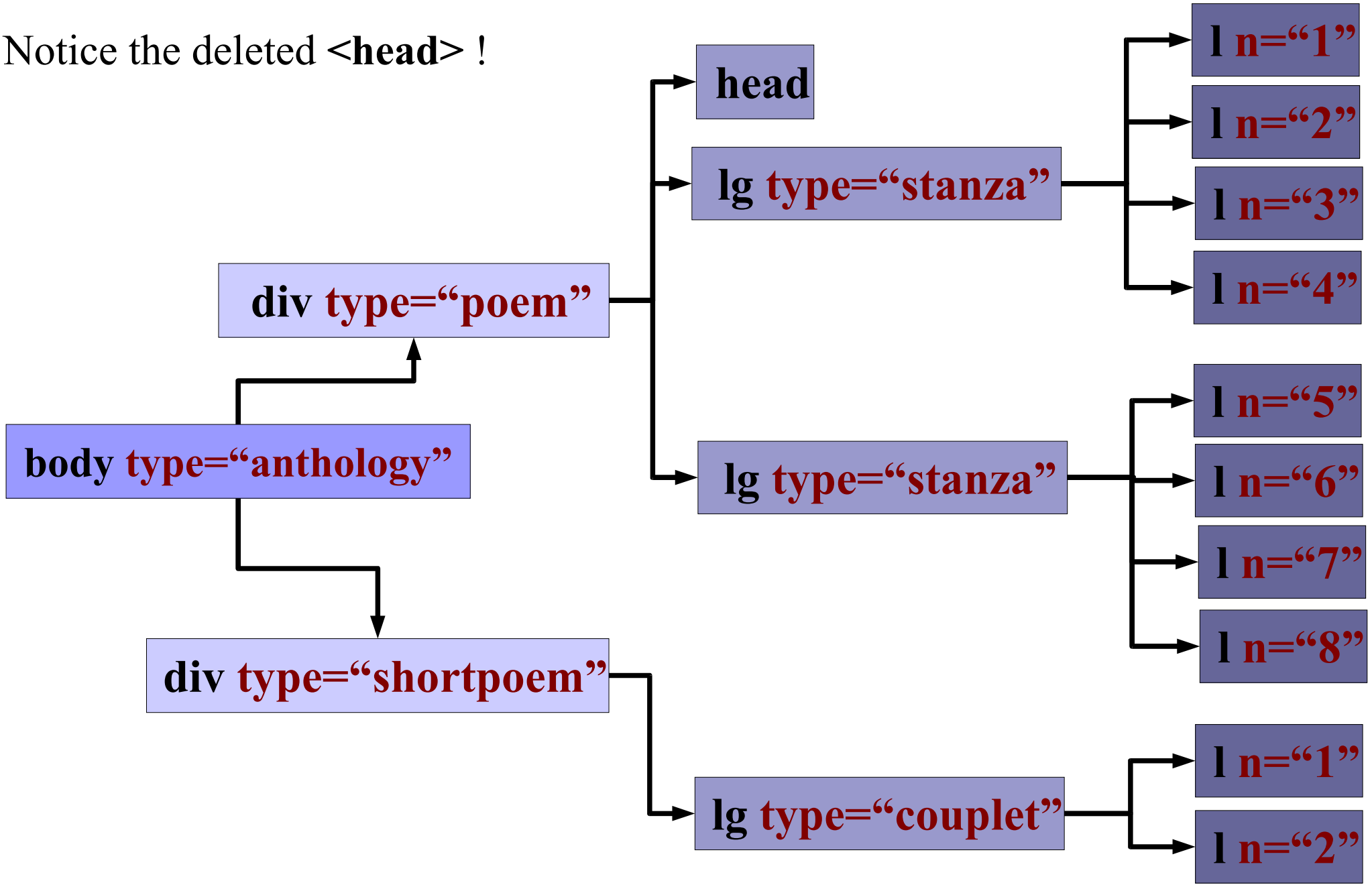


//l[@n > 5]

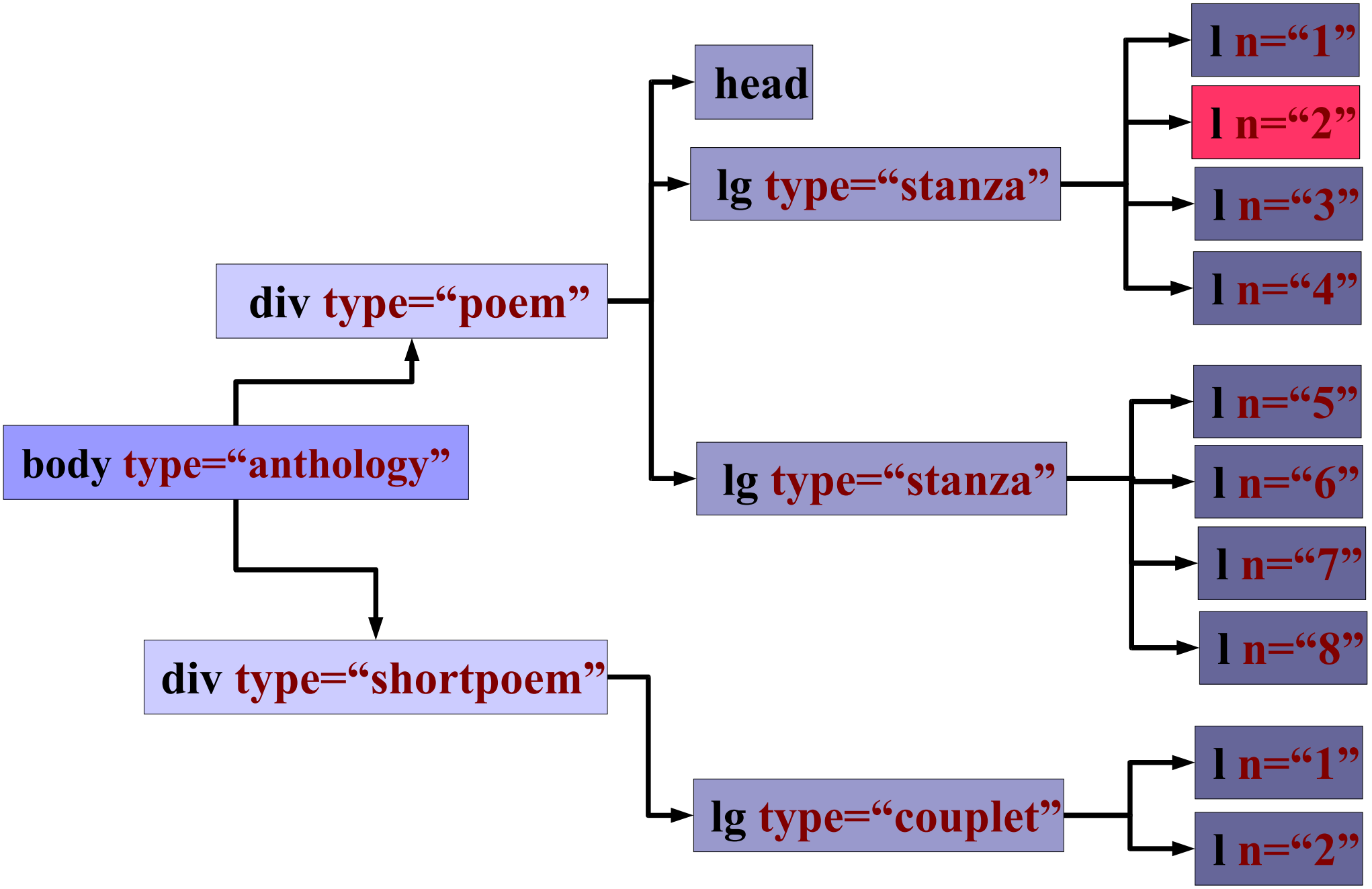

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

//div[head]/lg/l[@n="2"] ?

Notice the deleted **<head>** !

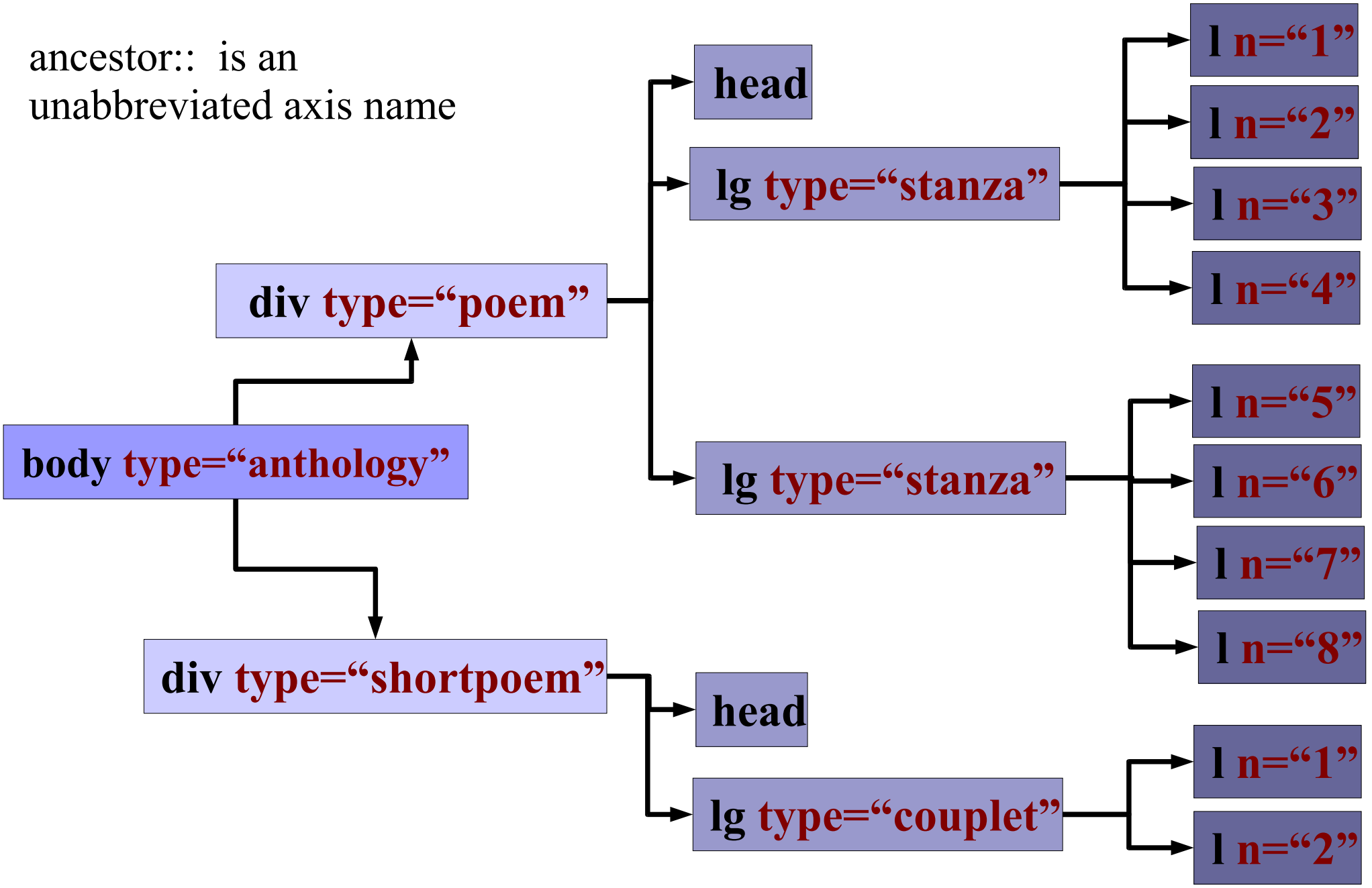


//div[head]/lg/l[@n="2"]

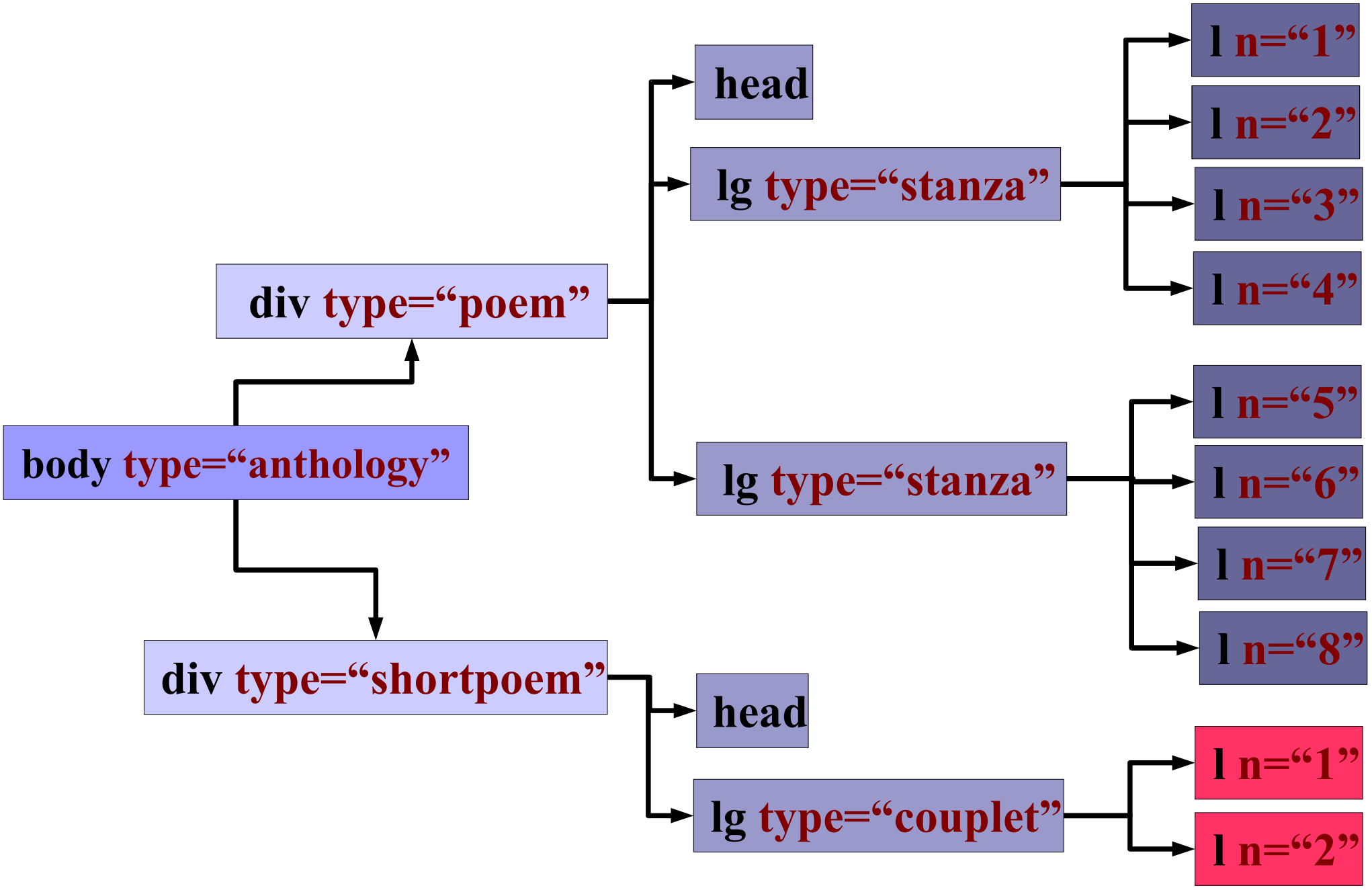


//l[ancestor::div/@type="shortpoem"] ?

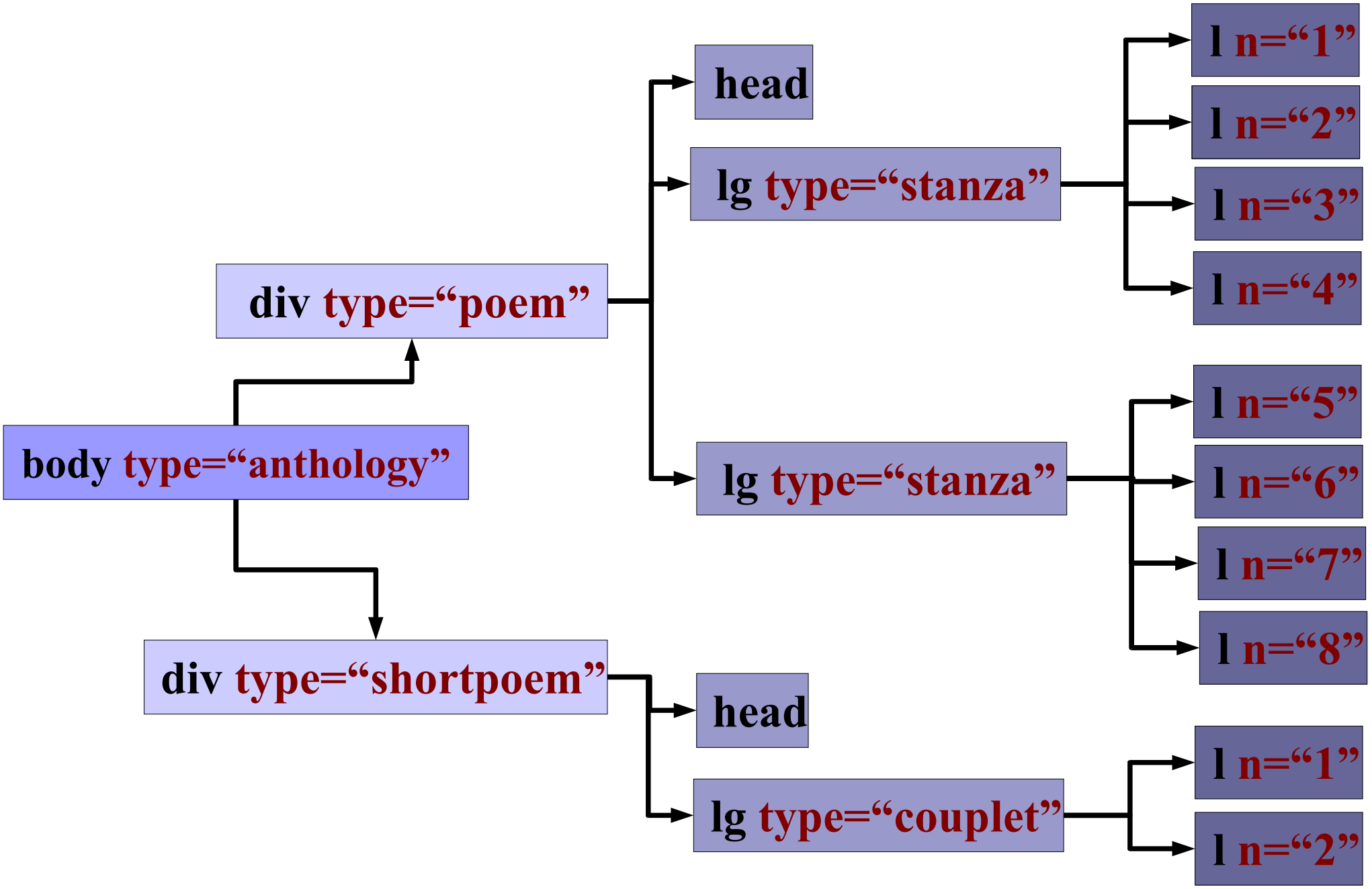
ancestor:: is an
unabbreviated axis name



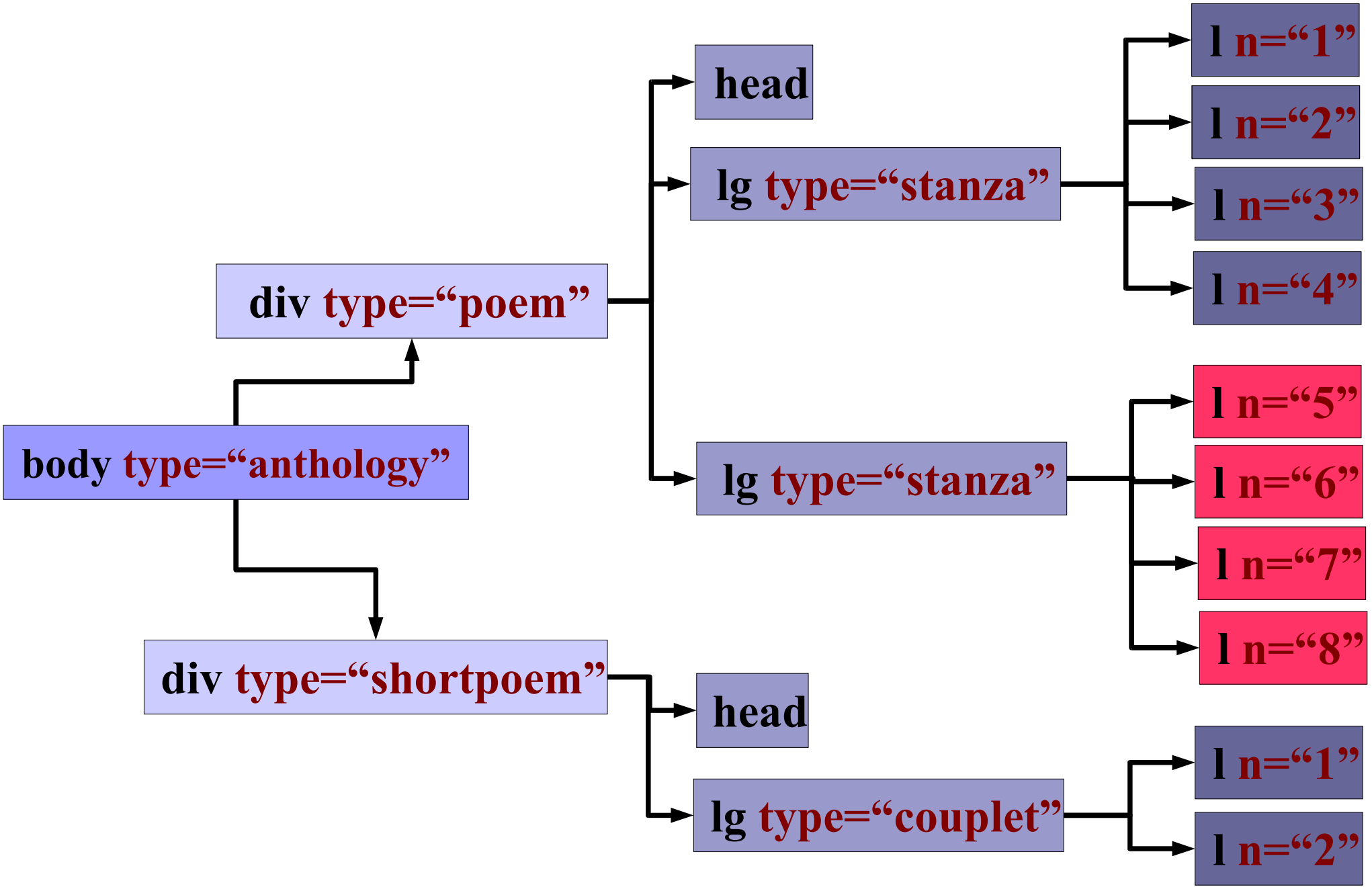
//l[ancestor::div/@type="shortpoem"]



lg[1]/following-sibling::l ?



lg[1]/following-sibling::l



More About Paths

- A location path results in a node-set
- Paths can be absolute: **/div/lg/l**
- Paths can be relative: **l/../../head**
- Formal Syntax:

  axisname::nodetest[predicate]

- for example:

  child::div[contains(head, 'ROSE')]

ahds literature, languages and linguistics

Formal Axis XPaths

- **ancestor::**

Contains all ancestors (parent, grandparent, etc.) of the current node

- **ancestor-or-self::**

Contains the current node plus all its ancestors (parent, grandparent, etc.)

- **attribute::**

Contains all attributes of the current node

Formal Axis XPaths

- **child::**

Contains all children of the current node

- **descendant::**

Contains all descendants (children, grandchildren, etc.) of the current node

- **descendant-or-self::**

Contains the current node plus all its descendants (children, grandchildren, etc.)

Formal Axis XPaths

- **following::**

Contains everything in the document after the closing tag of the current node

- **following-sibling::**

Contains all siblings after the current node

- **parent::**

Contains the parent of the current node

ahds literature, languages and linguistics

Formal Axis XPaths

- **preceding::**

Contains everything in the document that is before the starting tag of the current node

- **preceding-sibling::**

Contains all siblings before the current node

- **self::**

Contains the current node

ahds literature, languages and linguistics

Formal Axis Examples

- ancestor::lg = all <lg> ancestors
- ancestor-or-self::div = all ancestors or current
- attribute::n = n attribute of current node
- child::l = <l> elements directly under current node
- descendant::l = <l> elements anywhere under current node
- descendant-or-self::div = all children or current
- following::lg = all following <lg> elements
- following-sibling::l = next <l> element at this level
- parent::lg = immediate parent <lg> element
- preceding::lg = all preceding <lg> elements
- preceding-sibling::l = previous <l> element at this level
- self::head = current <head> element

Axis Predicates

- **child::lg[attribute::type='stanza']**
- **child::l[@n=4]**
- **child::div[position()=3]**
- **child::div[4]**
- **child::l[last()]**
- **child::lg[last()-1]**

ahds literature, languages and linguistics

XPath Abbreviated Syntax

- nothing is the same as child::
  lg is short for **child::lg**
- @ is the same as attribute::
  @type is short for **attribute::type**
- . is the same as self::node()
  ./head is short for **self::node()/head**
- .. is the same as parent::node()
  ../lg is short for **parent::node()/child::lg**
- // is the same as descendant-or-self::node()
  div//l is short for
  child::div/descendant-or-self::node()/child::l

Operators

• +	Addition	3 + 2 = 5
• –	Subtraction	10 – 2 = 8
• *	Multiplication	6 * 4 = 24
• div	Division	8 div 4 = 2
• mod	Modulus	5 mod 2 = 1
• =	Equal	@n = 74 (true for line 74)
• !=	Not equal	@n != 74 (f)
• <	Less than	@n < 84 (t)
• <=	Less than or equal	@n <= 72 (f)
• >	Greater than	@n > 25 (t)
• >=	Greater than or equal	@n >= 72 (t)
• or	Boolean OR	@n = 74 OR @age = 64 (t)
• and	Boolean AND	@n <= 84 AND @age = 74 (t)

ahds literature, languages and linguistics

Nodeset Functions

- count() -- Returns the number of nodes in a node-set

  count(//lg[1]/l)
- id() -- Selects elements by their unique ID

  id('S3')
- last() -- Returns the position number of the last node

  //l[last() - 1]
- name() -- Returns the name of a node

  //lg/*[name('head')]/text()
- namespace-uri() -- Returns the namesparce URI of a specified node

  namespace-uri(div/figure)
- position() -- Returns the position in the node list of the node that is currently being processed

  //div[@type="poem"][position()= 6]

ahds literature, languages and linguistics

String Functions

- concat() -- Concatenates its arguments

  concat('http://', $domain, '/', $file, '.html')
- contains() Returns true if 2nd is contained within the 1st

  //div[contains(lg/l, 'Rose')]
- normalize-space() -- Removes leading and trailing whitespace and replaces all internal whitespace with one space

  normalize-space(head)
- starts-with() -- Returns true if the first string starts with the second

  starts-with(head, 'The')
- string() -- Converts the argument to a string

  string(@n)
- translate() -- Character by character replacement. Characters in the 1st are replaced with characters in same location in the 2nd

  translate('1234', '24', '68')

ahds literature, languages and linguistics

Substring Functions

- substring -- Returns part of a string of specified start character and length

  substring(head, 5,4)

- substring-after() -- Returns the part of the string that is after the string given

  substring-after(head, 'The ')

- substring-before Returns the part of the string that is before the string given

  substring-before(@date, '-')

- Functions can be nested:

  substring-before(substring-after(@date, '-'), '-')

What does the final example give us if you use ISO date format of:
YYYY-MM-DD ?

ahds literature, languages and linguistics

Numeric Functions

- ceiling() -- Returns the smallest integer that is not less that the number given

  ceiling(3.1415)
- floor() -- Returns the largest integer that is not greater than the number given

  floor(3.1415)
- number() -- Converts the input to a number

  number('100')
- round() -- Rounds the number to the nearest integer

  round(3.1415)
- sum() -- Returns the total value of a set of numeric arguments

  sum(//person/@age)
- not() -- Returns true if the condition is false

  not(position() >5)

XML Namespaces

- The Namespace of an element, is the scope within which it is valid.
- Elements without Namespaces may collided when we combine bits of multiple documents together. XML Namespaces solve this problem and enable using other schemas within yours.
  (e.g. SVG or MathML inside TEI)
- An XML Namespace is identified by a URI reference.
- XML Namespaces usually appear as qualified names, which contain a single colon, separating the name into a prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace.
  (e.g. tei:teiHeader, svg:line)
- Child elements inherit the namespace declaration of their ancestor.

Namespaced XML Example

```
<TEI xmlns="http://www.tei-c.org/ns/1.0" id="S3">
  <teiHeader> [... lots omitted ...]
        <profileDesc>
            <particDesc>
                <person sex="m" age="78">
                      <persName>
                              <foreName>Hans</foreName>
                               <surname>von Hülsen</surname>
                       </persName>
                       <birth date="1890-04-05"/>
                       <death date="1968-04-14"/>
                       <nationality code="Z_"/>
                  </person>
              </particDesc>
         </profileDesc>
  </teiHeader>
  <text><body><ab> <!-- the text of the stone --> </ab></body></text>
</TEI>
```

ahds literature, languages and linguistics

Xpaths With Namespaces

- /tei:TEI/tei:teiHeader//tei:person
- //tei:person/ancestor::tei:TEI/@id
- //tei:persName[tei:foreName = 'Hans']/../tei:birth
- //tei:person[@sex='f']/tei:nationality

ahds literature, languages and linguistics

Why should I learn XPath?

- Most methods of accessing XML documents rely on XPath to located nodesets
- It is powerful and compact
- It is relatively easy to learn
- It is a W3C recommendation
- XPath 2.0 is being worked on and adds:
  - More numerical types for integer, single, double, etc.
  - Data Types for dates, times, durations etc.
  - User-defined and sequential data types

Exercises

- If we have time, there are some quick XPath exercises for you to do
- Knoppix:
  - If the machines aren't already setup, boot off Knoppix CD (be patient)
  - Wait for FireFox to load
  - go to 'eXist'
  - In its sidebar 'Basic XQuery Interface'
- You should have an XPath (and XSLT) quick reference sheet and XPath exercises