# XSLT pour les nuls — Esquisse d'une initiation aux normes de transformation XML

Lou Burnard

mai 2011

# Objectifs

Ceci n'est pas une formation complète! son objectif est de ...

- vous donner un avant-gout des possibilités affordés par les normes XSLT et XPath
- surtout dans le domaine de traitement des documents 'XML-TEI', principalement donc sortis des sciences humaines
- aborder les concepts fondamentaux et les usages les plus répandus du norme XSLT
- vous préparer pour une formation plus approfondie de Sebastian Rahtz (demain)

# Un mot sur MEET

MEET c'est une action transversale du TGE ADONIS pour la ...

- Mutualisation d'
- Experiences sur l'
- Encodage
- TEI

Entre autres activités, il est charge de promouvoir l'échange et la mutualisation des compétences et des experiences techniques sur le traitement des documents TEI-XML en France.

D'où cette formation, realisée en collaboration avec l'action MUTEC.

# XSL: un ensemble de normes complémentaires

- XPath: un syntaxe normalisé pour définir et accéder aux sous-parties d'une arborescence XML
- XSLT: un norme informatique pour la transformation des arborescences XML
- XSL FO: un vocabulaire XML pour la description d'affichage des pages

Tous les trois dévelopés et maintenus par le W3C, comme le norme XML.

# A quoi sert le XSL ?

- Un document XML n'est qu'une chaîne de caractères Unicode avec des balises: pourquoi pas le traîter en tant que tel?
- Parce que les balises representent quelque chose de plus signifiant : l'arborescence.
- Parce que le syntax XML permet des variations ...

```
<foo bar= "x" baz = "z"></foo>
<foo
baz= "z" bar =
"x"/>
```

- Parce que la *structuration* d'un document XML fait partie de sa signification

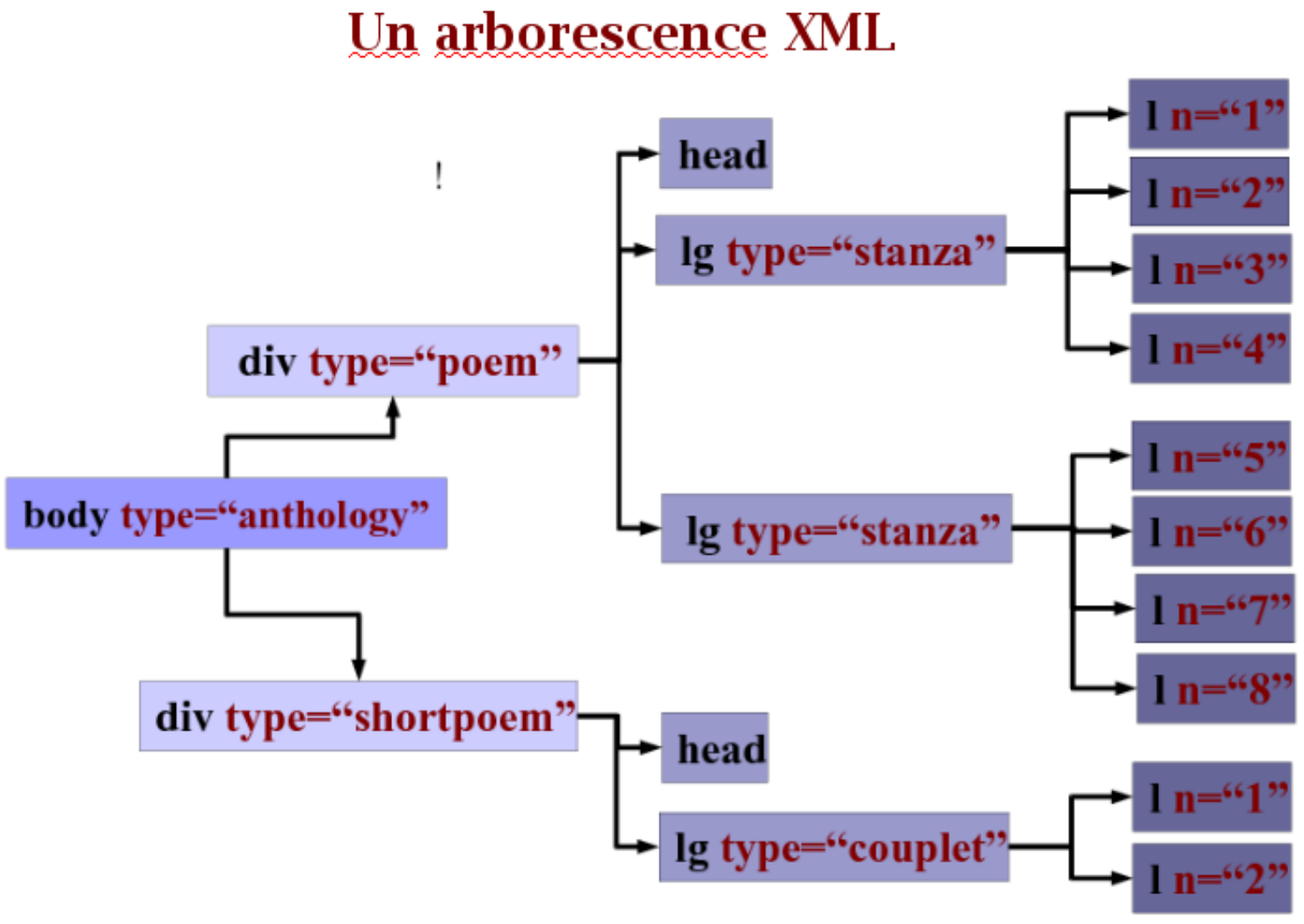
# C'est quoi une arborescence ?

- un ensemble de *noeuds*, organisé de manière hiérarchique
- chaque noeud porte un *identifiant générique* (son "type")
- il y a un seul *noeud racine* qui contient (ou domine) tous les autres
- chaque noeud peut contenir (ou dominer)
  - un sous-arborescence
  - ou un morceau de texte
  - facultativement un ensemble d'*attributs*
- chaque attribut comporte un *nom* et un *valeur*

# Exemple texte XML ...

```
<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>
  <div type="shortpoem">
    <head>Queen Anne's tipple</head>
    <lg type="couplet">
      <l n="1">Here thou Great Anna whom three realms obey</l>
      <l n="2">Doth sometimes council take, and sometimes tea.</l>
    </lg>
  </div>
</body>
```

# .. ou, en forme d'arborescence:



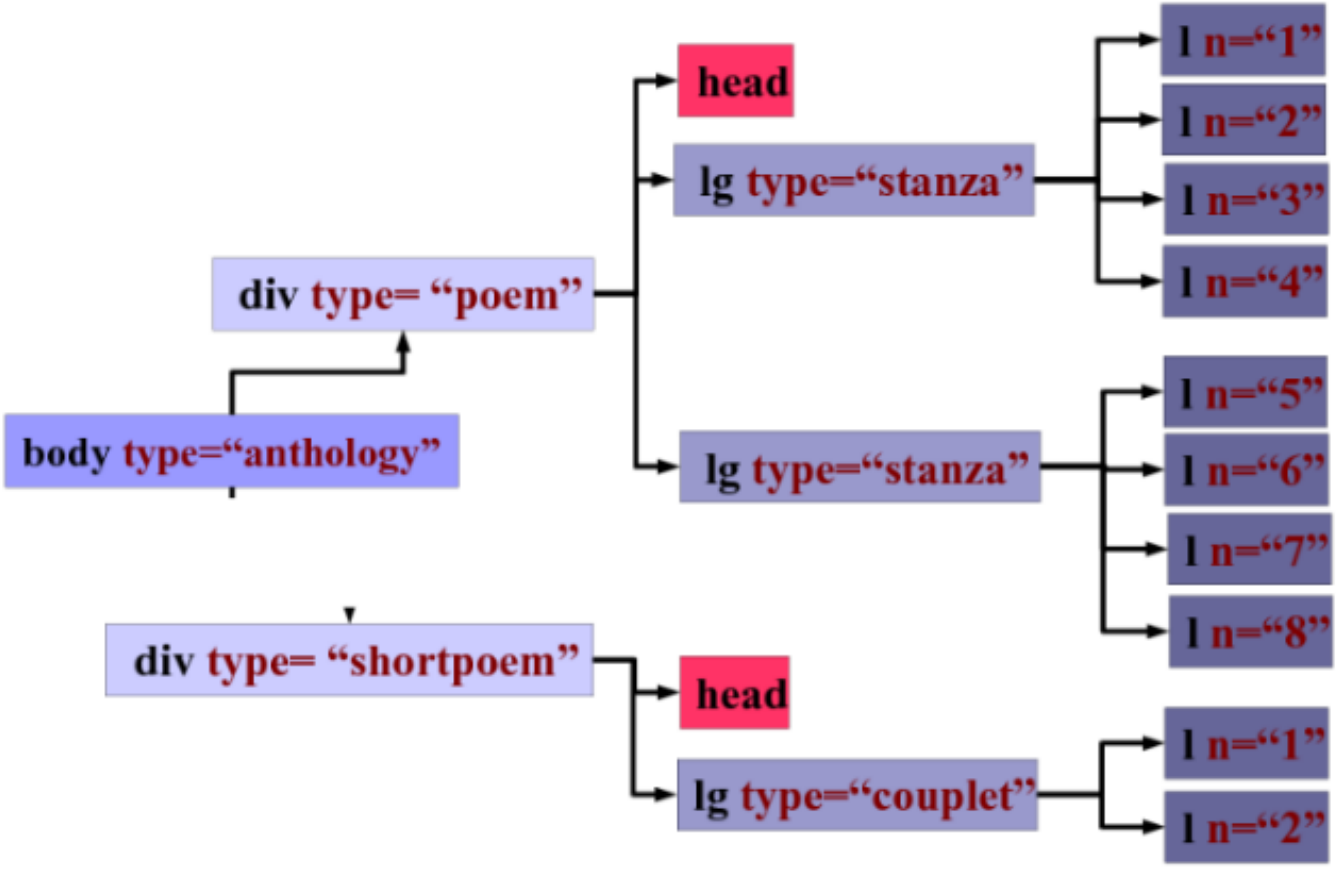
# XPath : une feuille de route

Pour accéder aux composants d'un document XML, on spécifie un *chemin*, spécifiant les noeuds qu'il faut traverser pour arriver à la partie souhaité

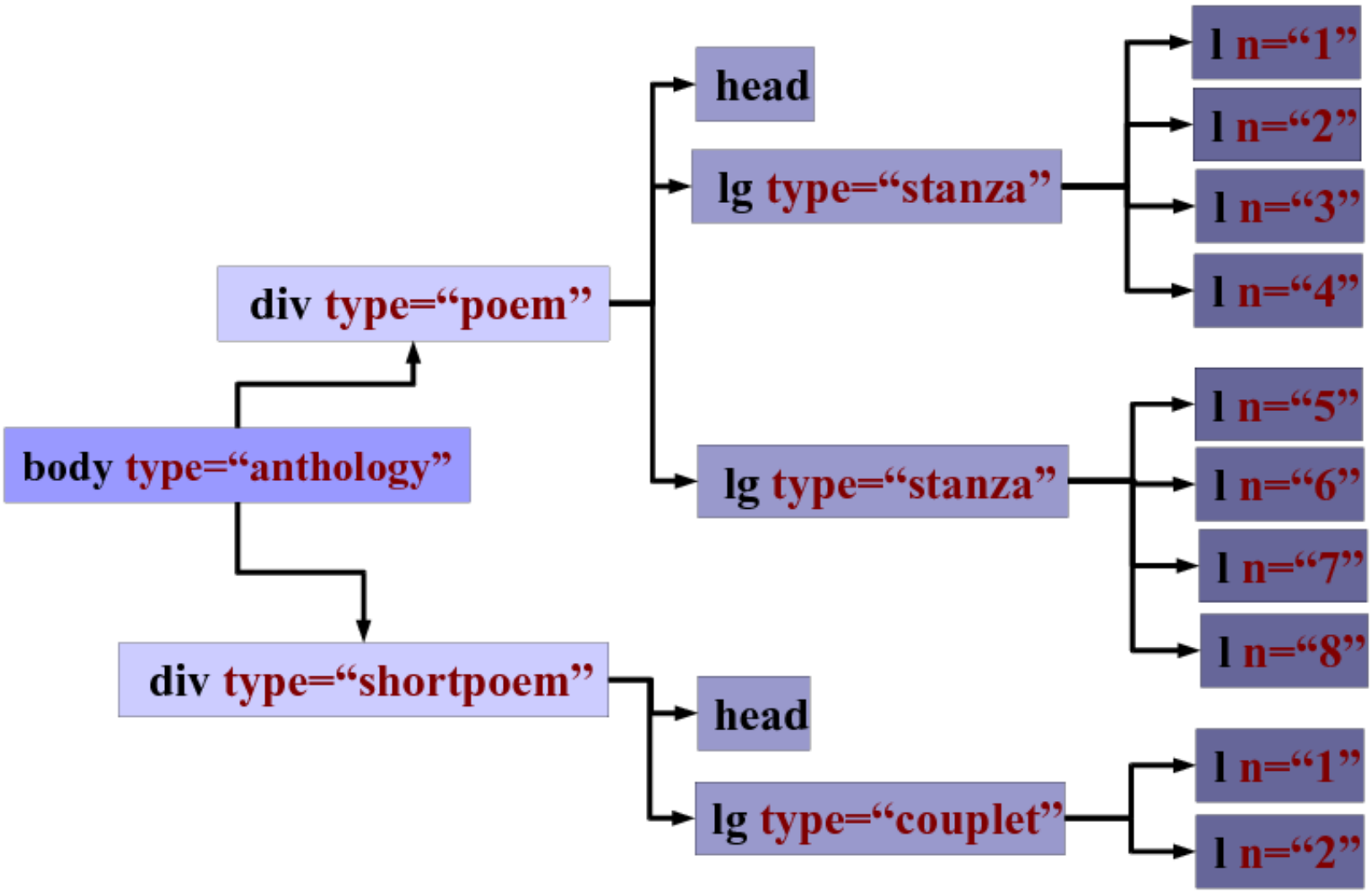
Par exemple, pour arriver aux <head>s dans cet exemple, on commence au <body>, puis passe à un  fils, et ensuite on arrive à un <head>

En XPath, on dit : `/body/div/head`

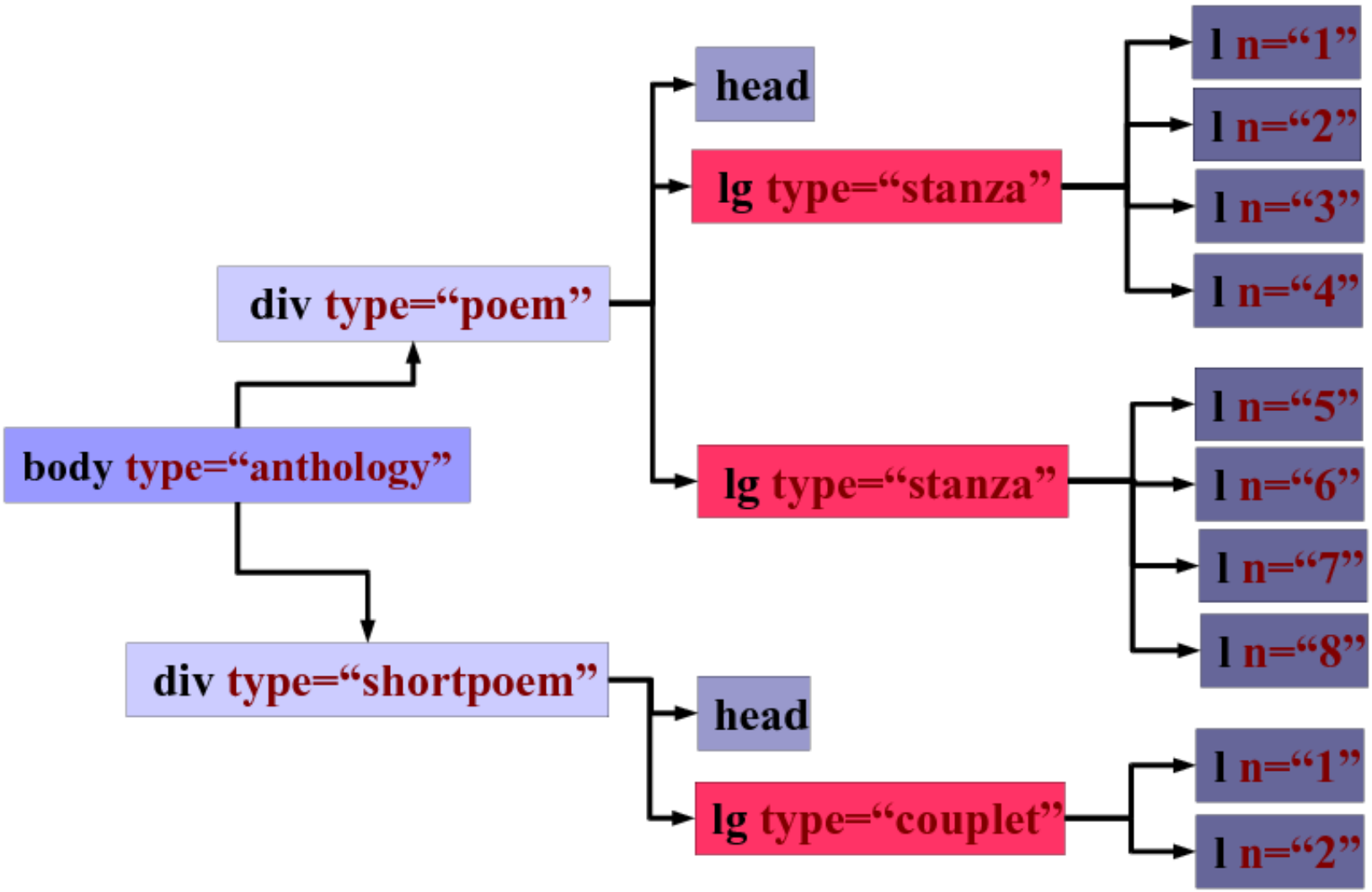
# /body/div/head



# /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"

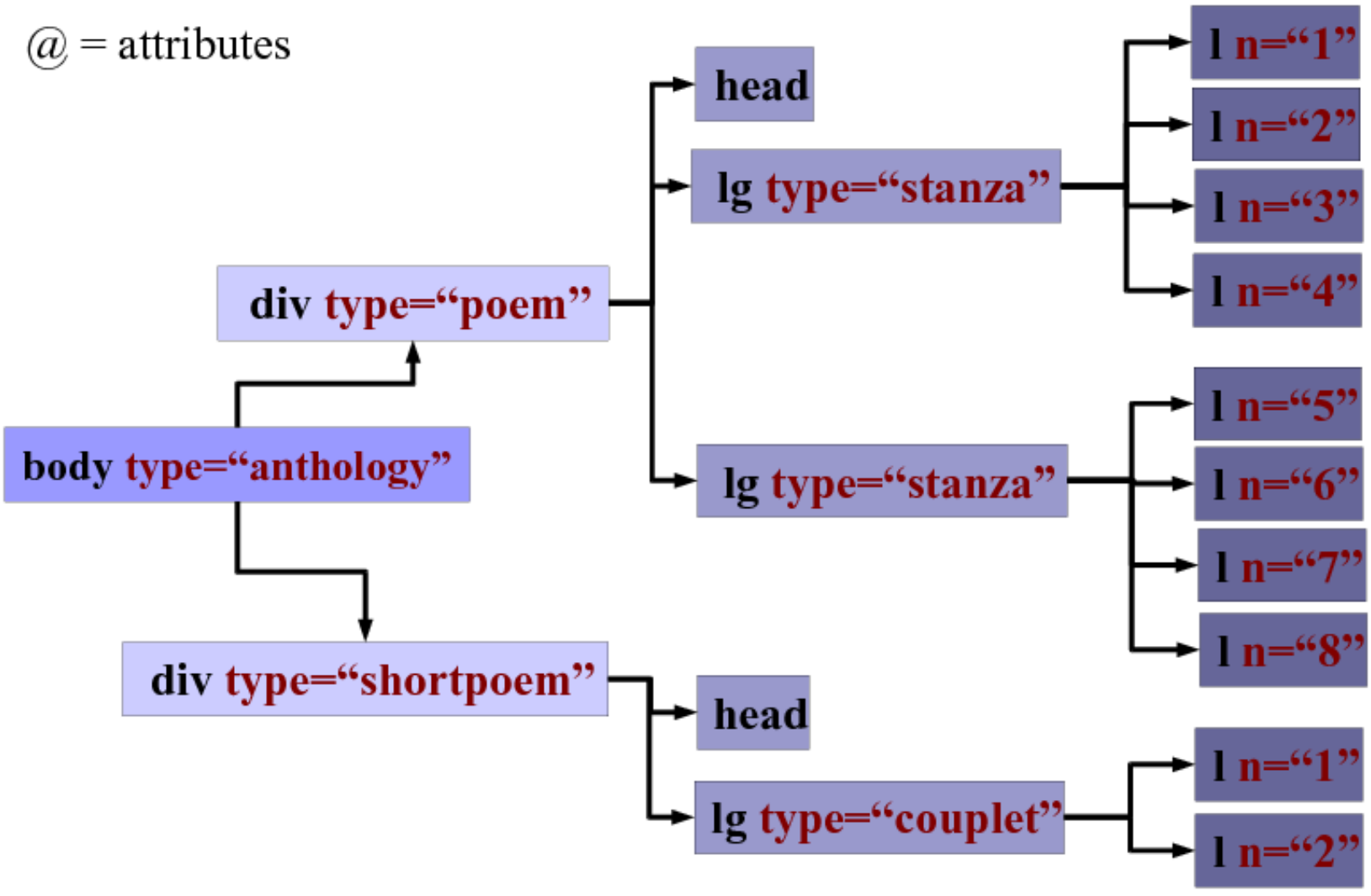
# /body/div/lg


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"

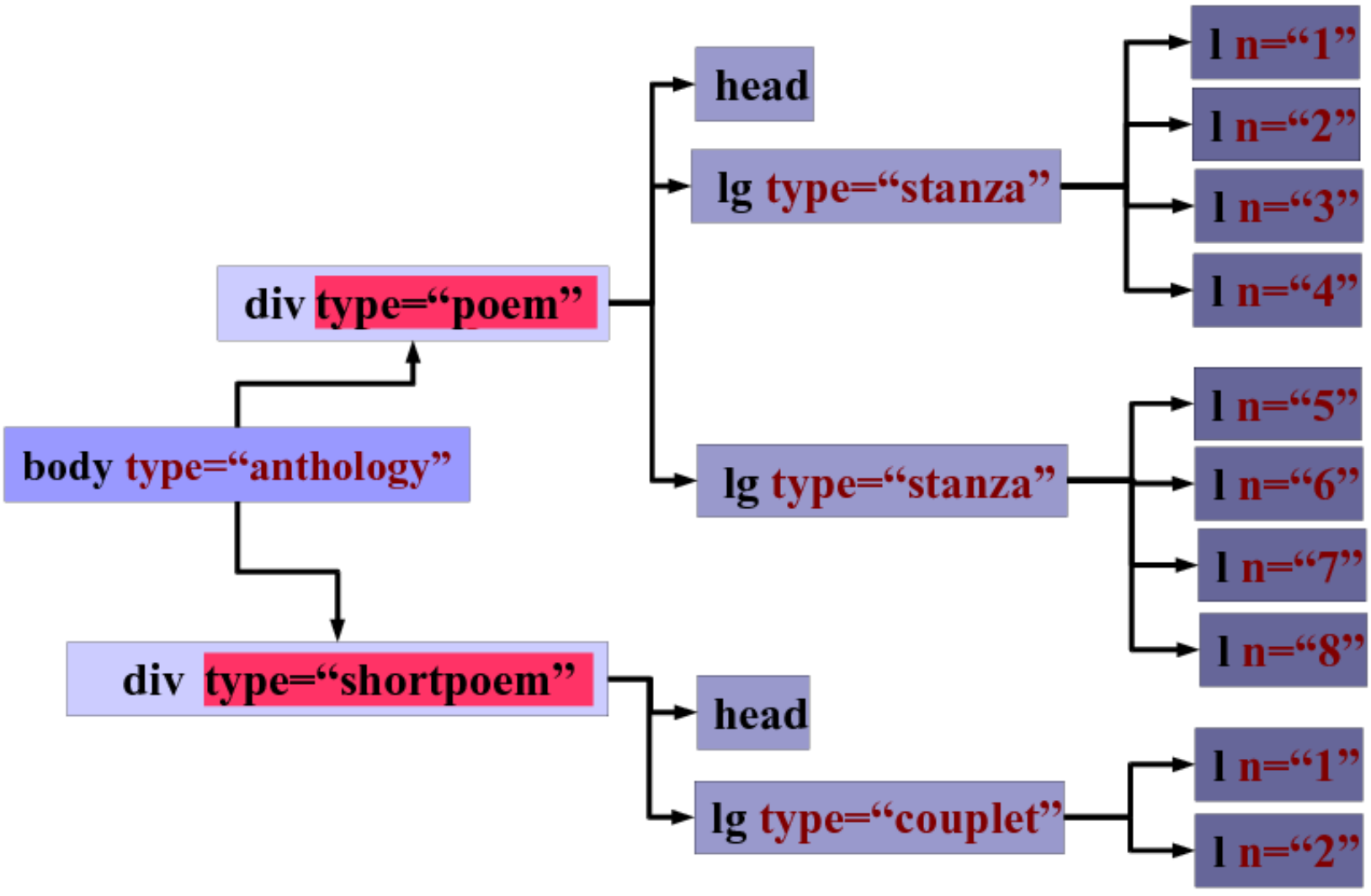
# Les étapes

- Chaque étape sur ce chemin n'est pas forcément un élément XML...
- on peut aussi regarder les attributs
- ou des morceaux de texte

# /body/div/@type ?



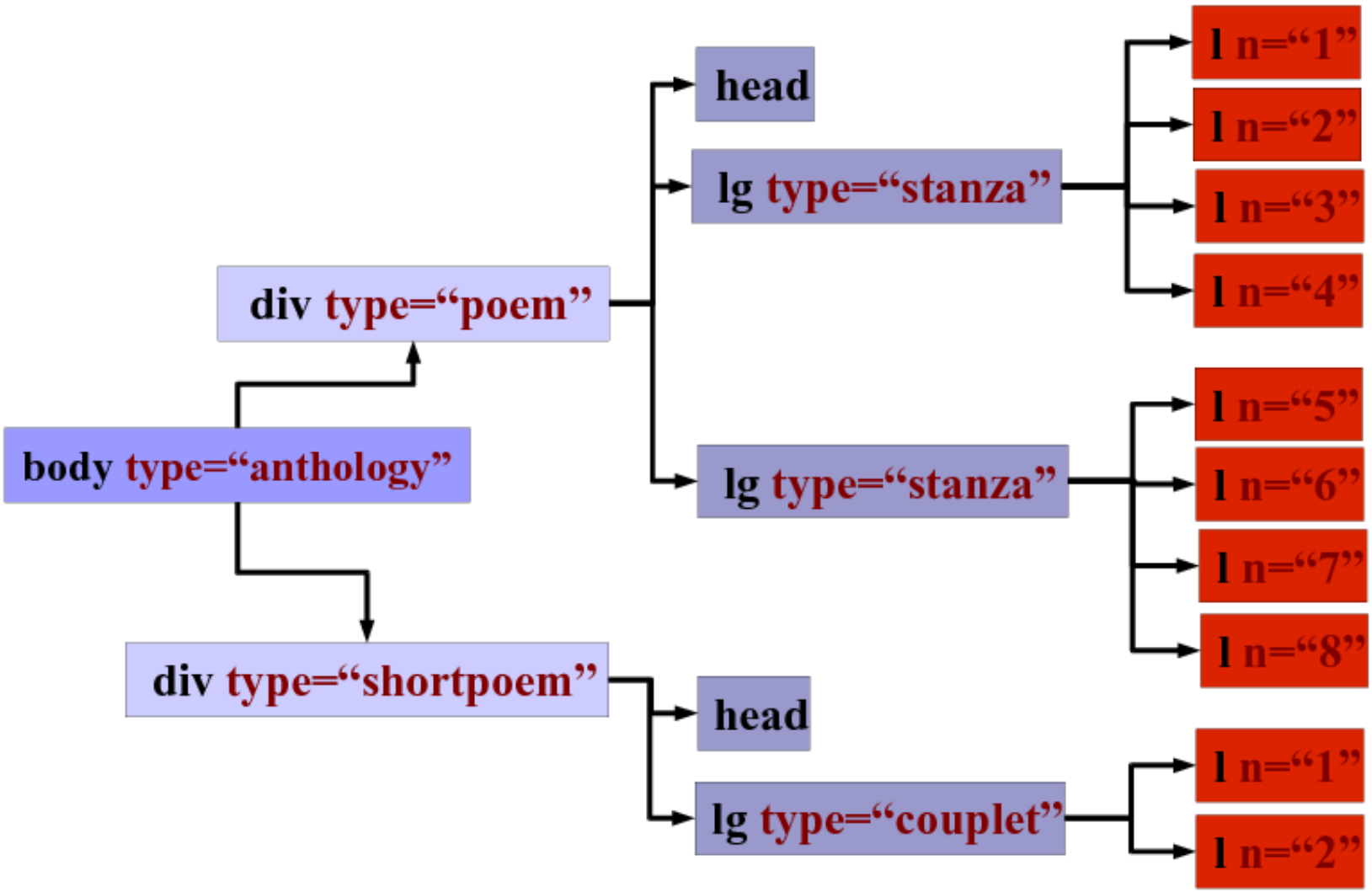
# /body/div/@type



# Les sélections

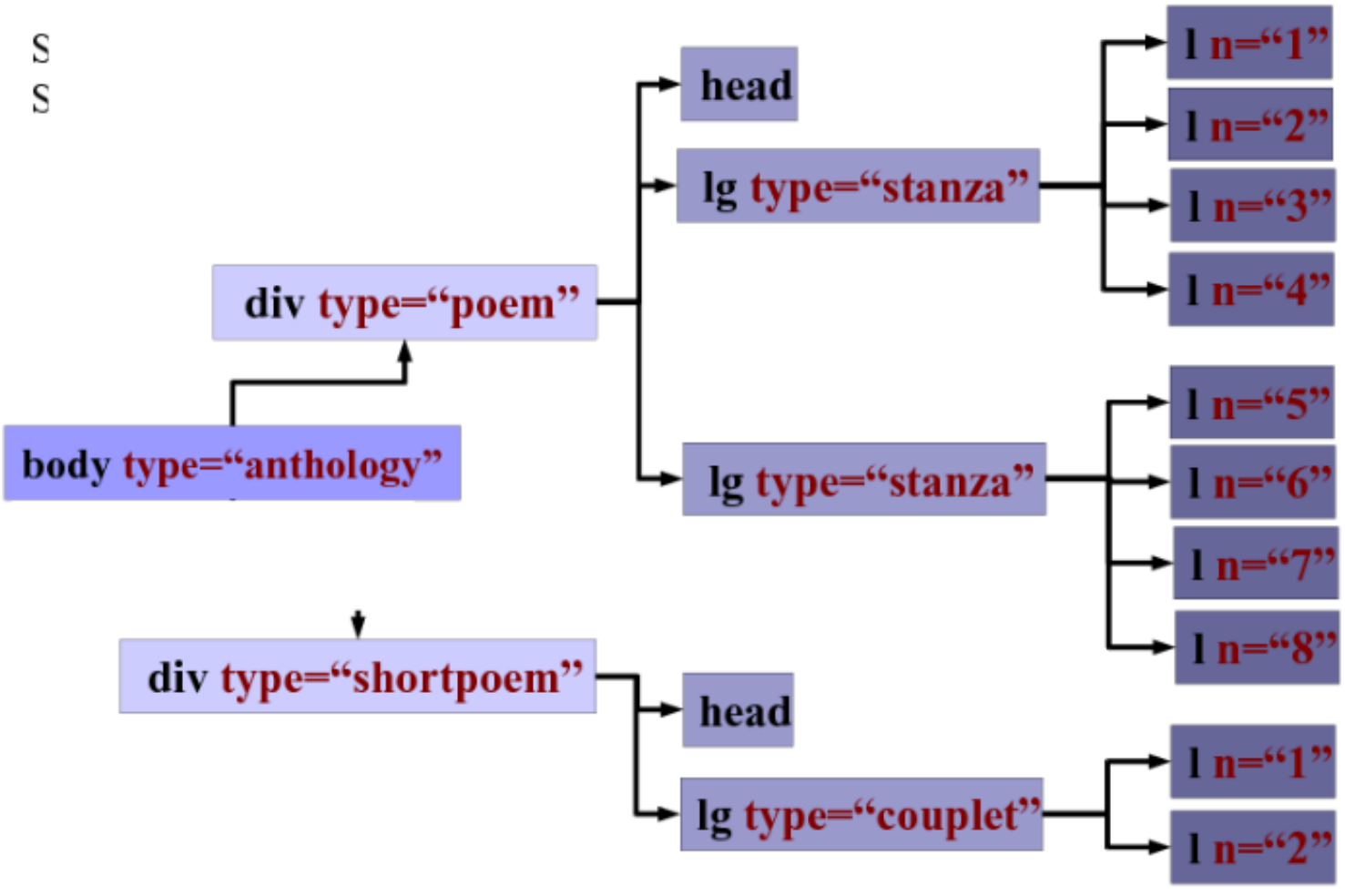
- On peut séléctionner parmi les noeuds resultants, en forme de restriction entre parenthèses [ et ]
- Une restriction peut tenir en compte la valeur d'un attribut
- ou la position ordinale du noeud dans l'arbre
- ou l'existence d' un élément du type indiqué

# /body/div/lg/l

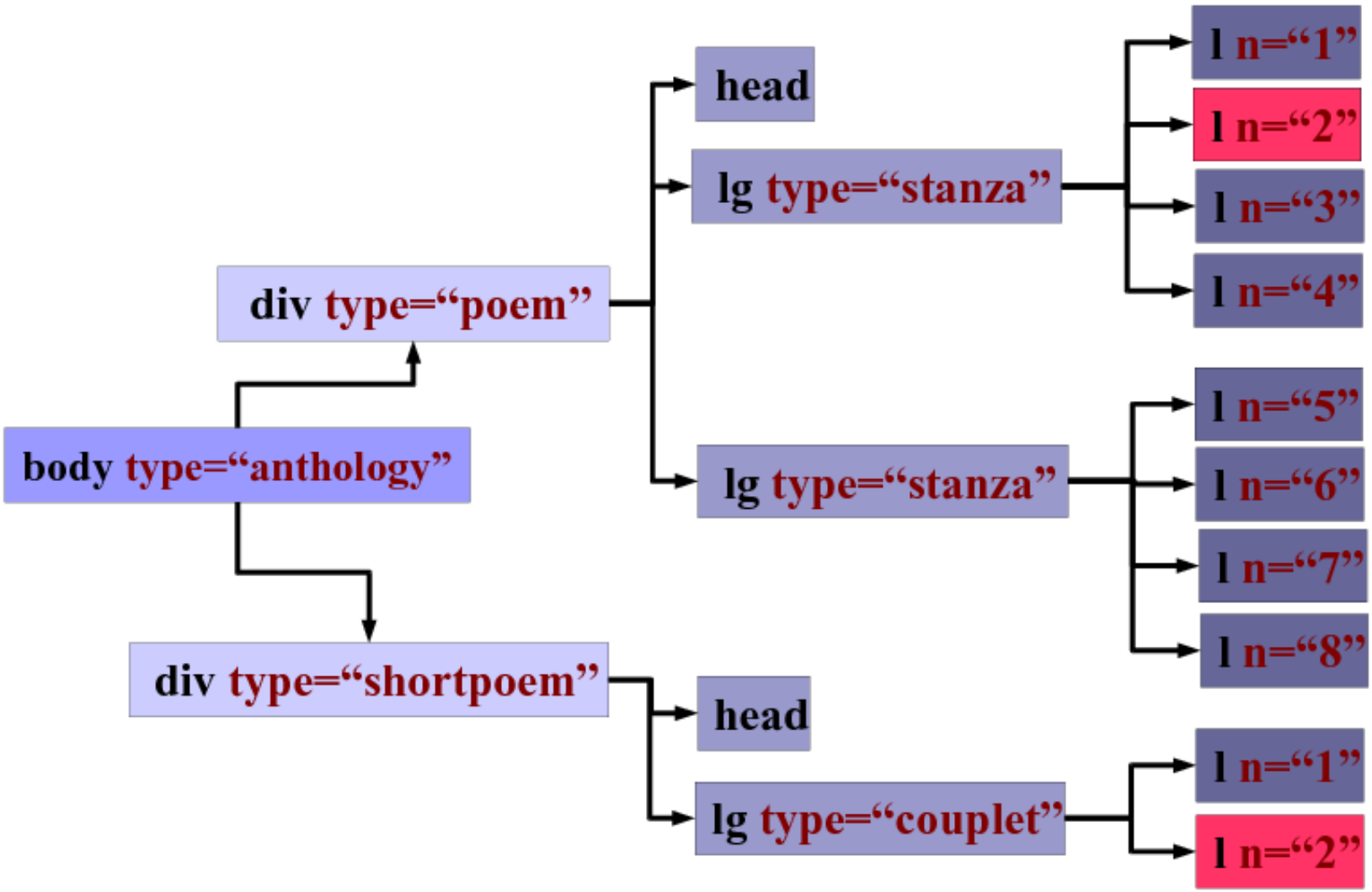

body type="anthology"
div type="poem"
head
lg type="stanza"
l n="1"
l n="2"
l n="3"
l n="4"
lg type="stanza"
l n="5"
l n="6"
l n="7"
l n="8"
div type="shortpoem"
head
lg type="couplet"
l n="1"
l n="2"

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

S
S



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



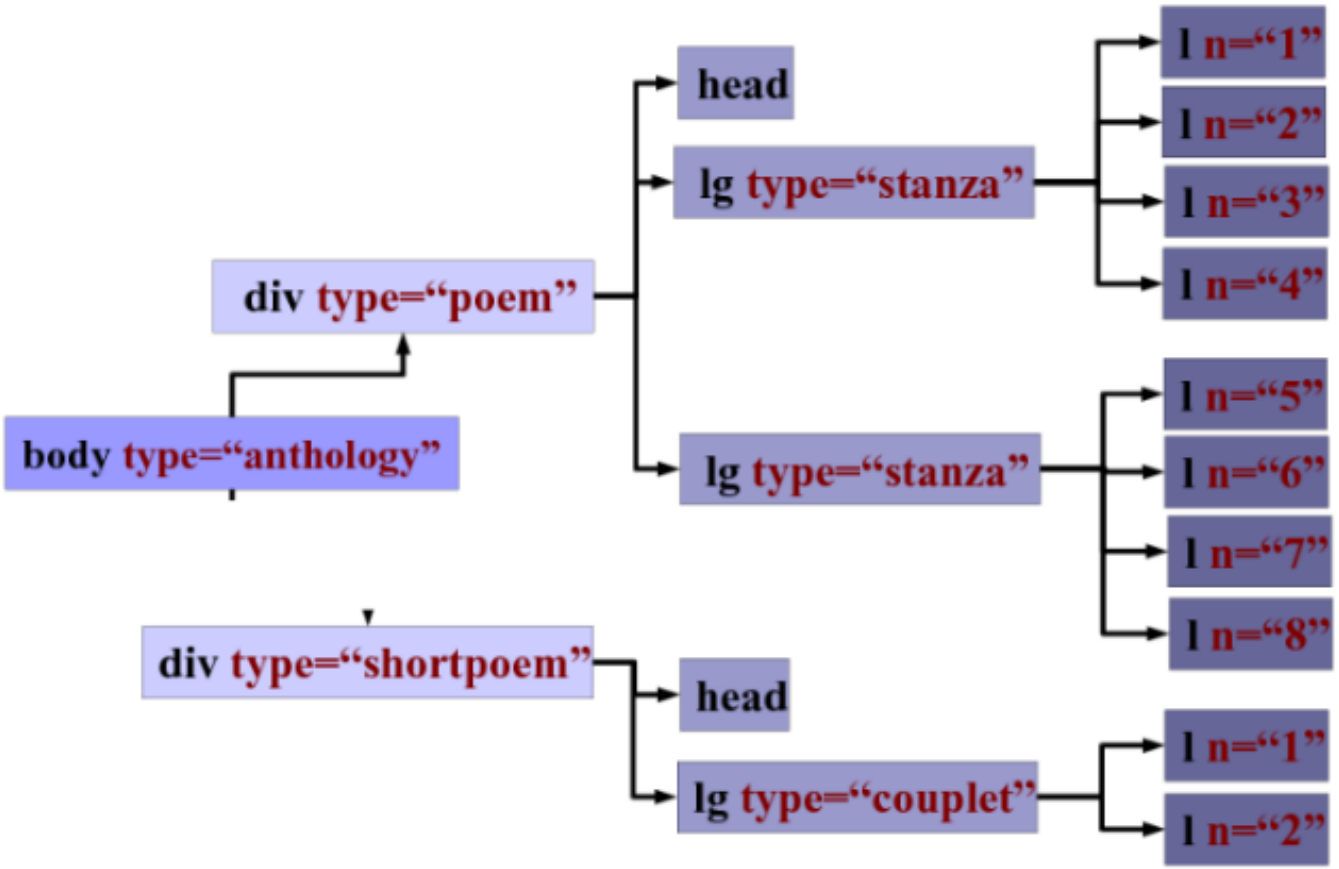
# Le point de depart

On peut partir de n'importe quel point dans l'arborescence:

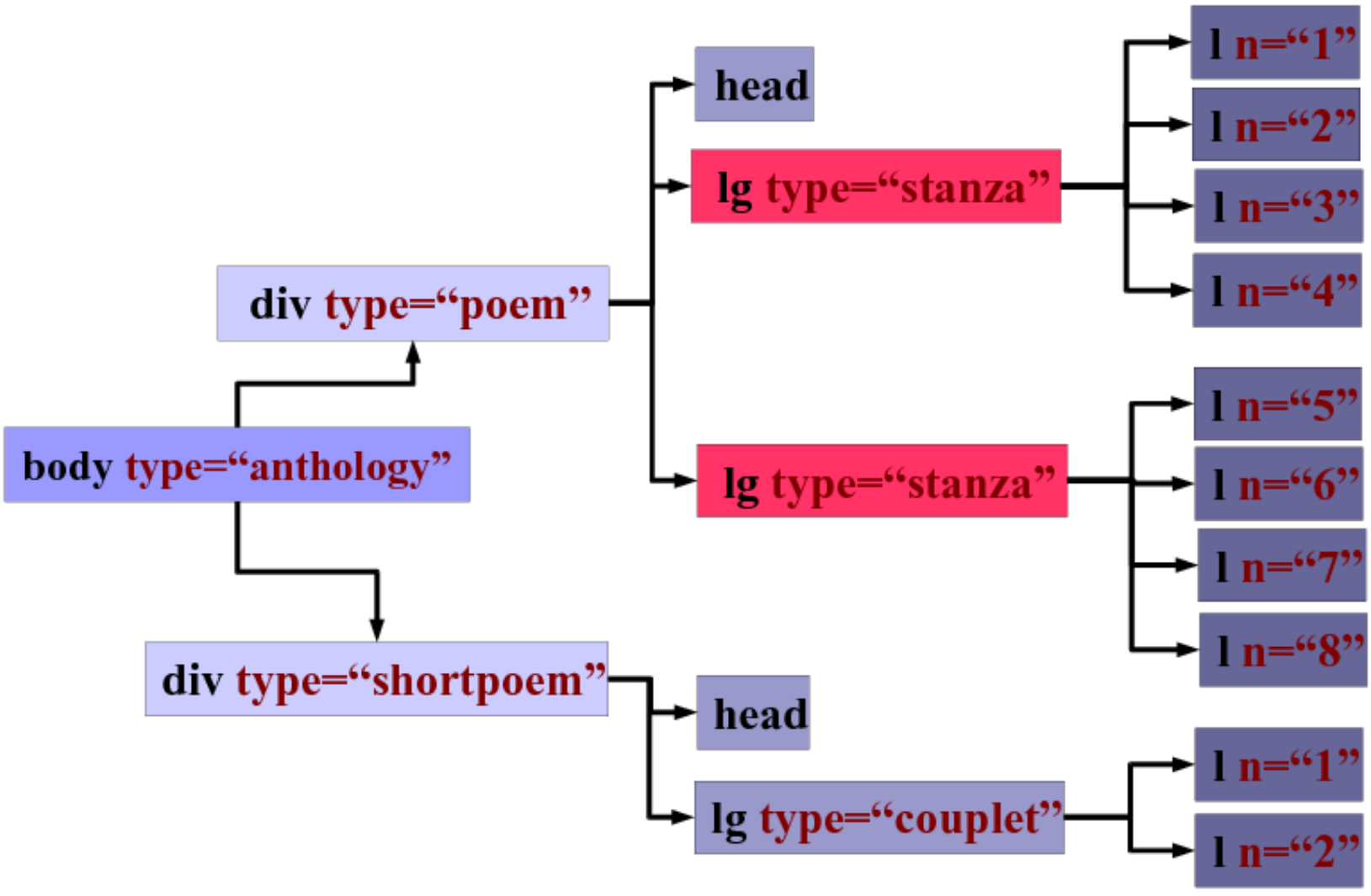
- `//` signifie ‘n'importe ou’
- `..` signifie ‘mon parent’

On peut aussi naviguer l'hierarchie, en se servant des *axes* tels que `ancestor::`, `following-sibling::`, `descendant::` ...

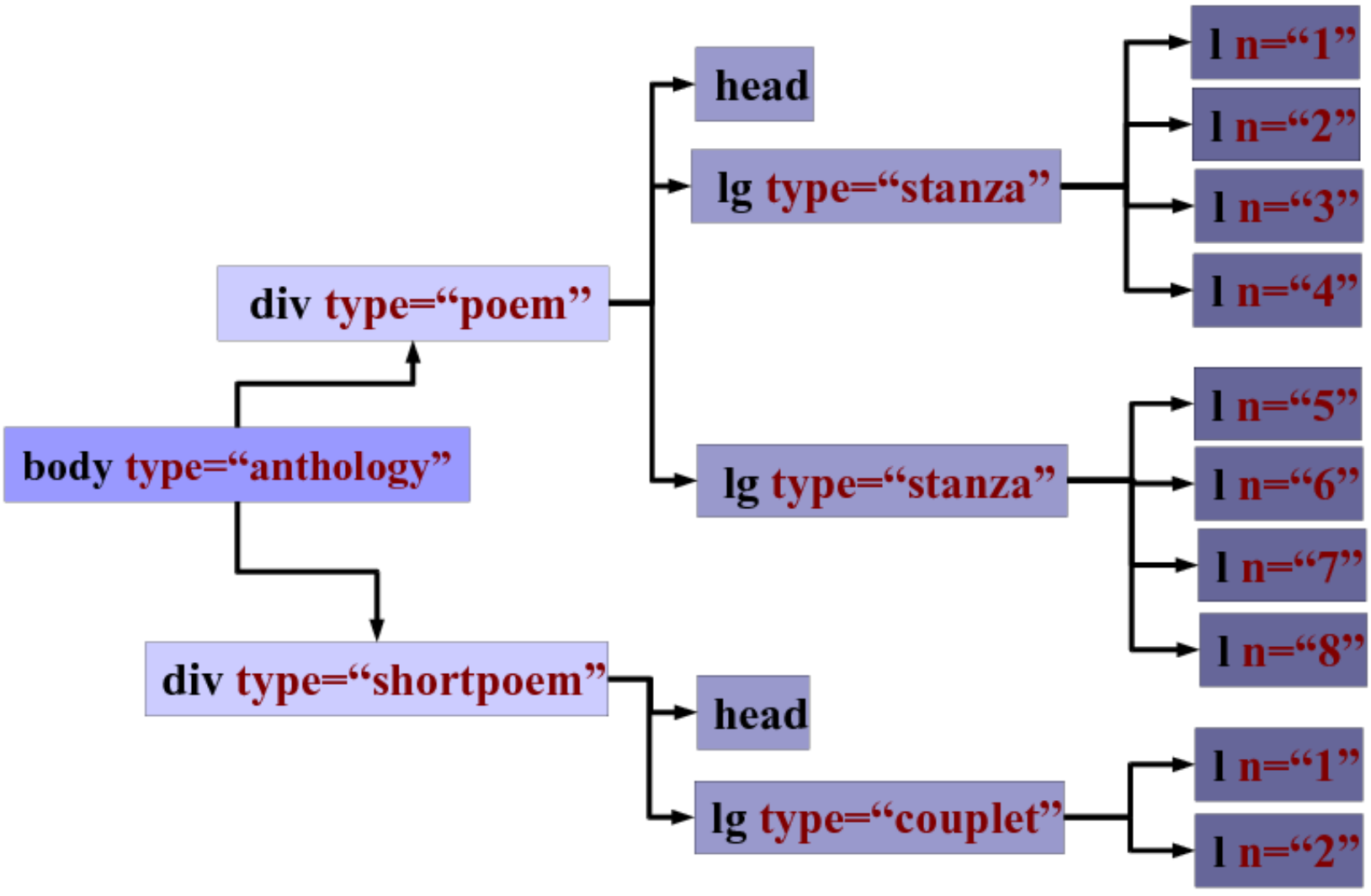
# //lg[@type="stanza"] ?



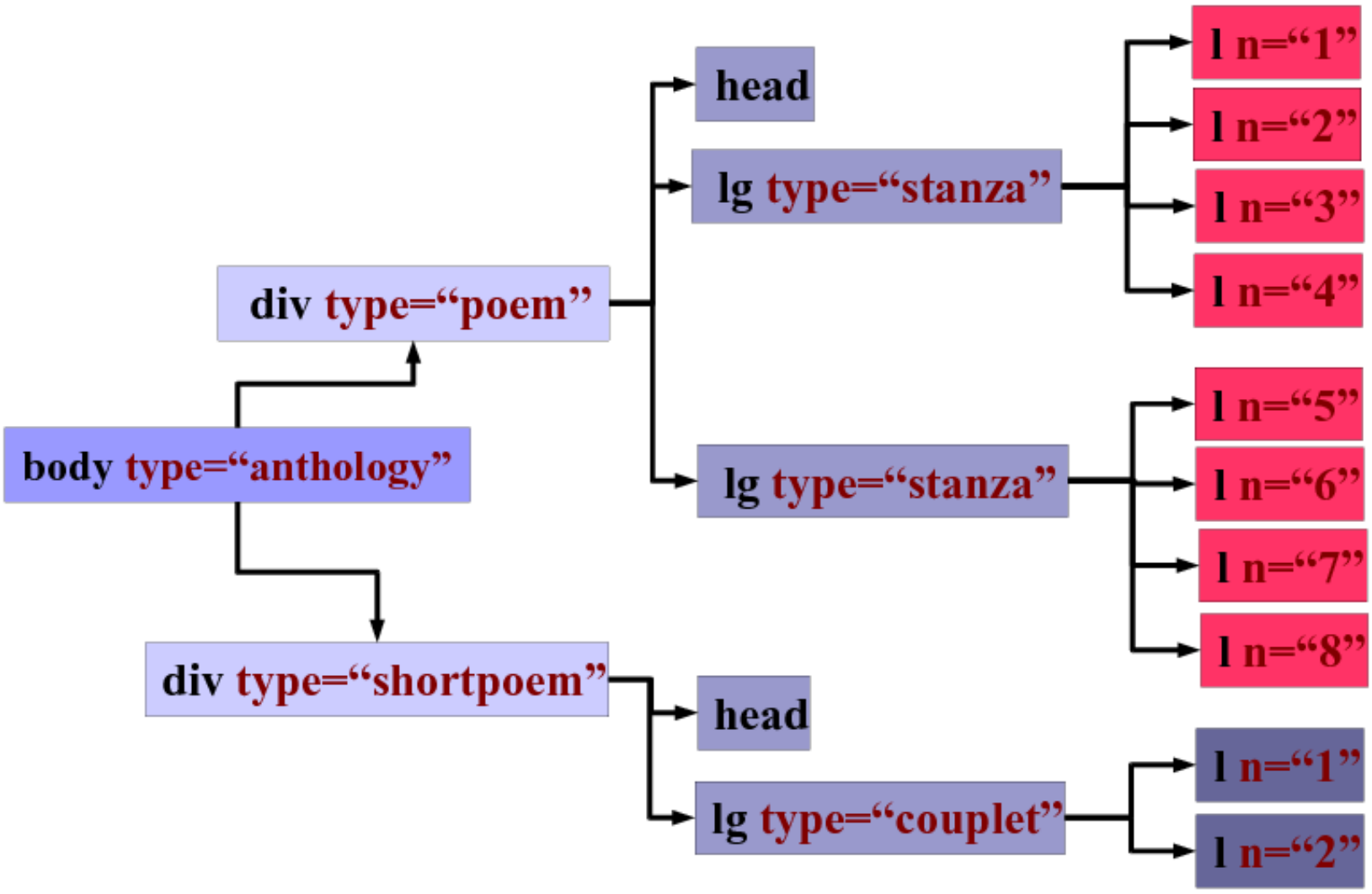
# //lg[@type="stanza"]



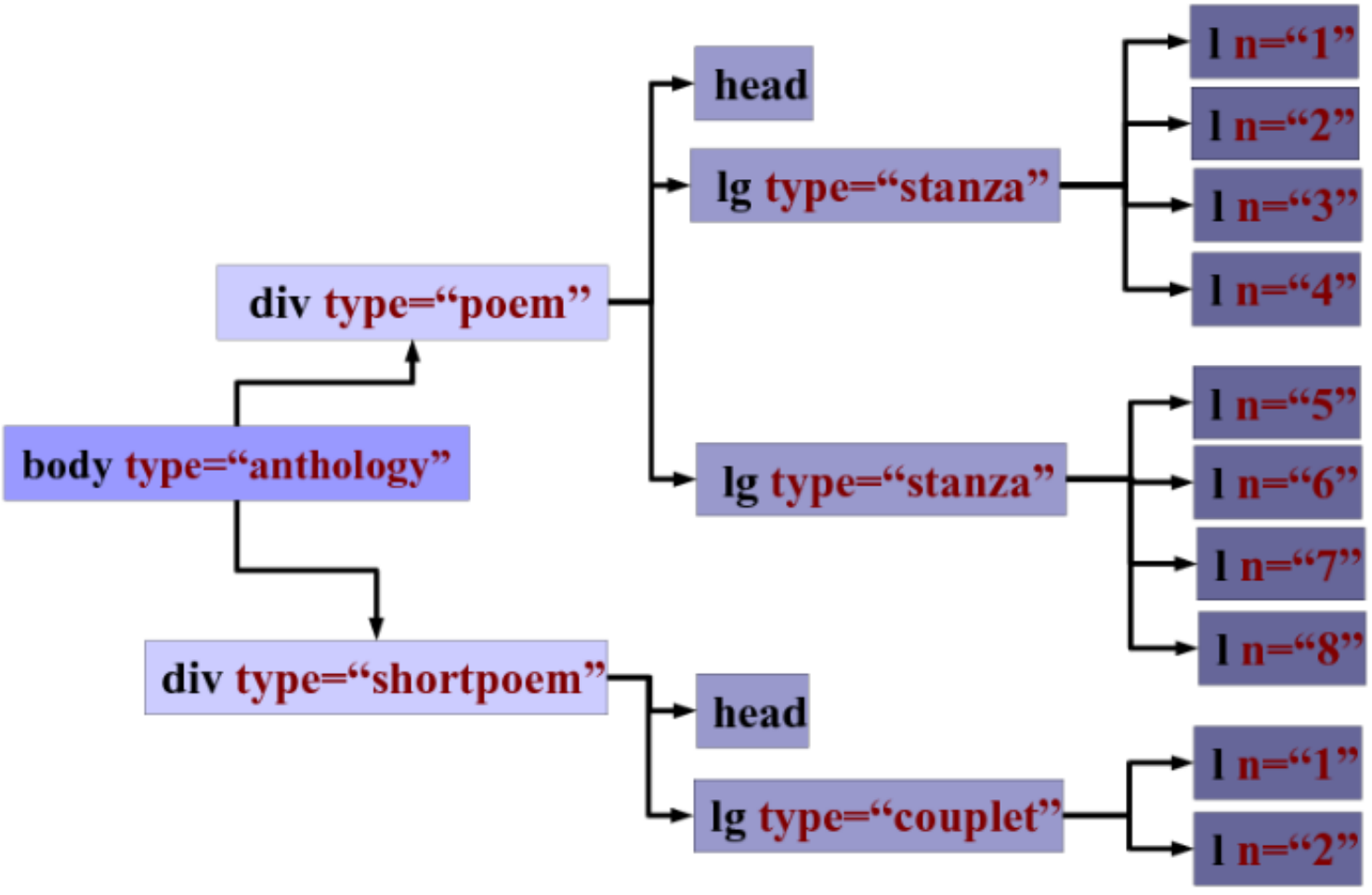
# //div[@type="poem"]//l ?



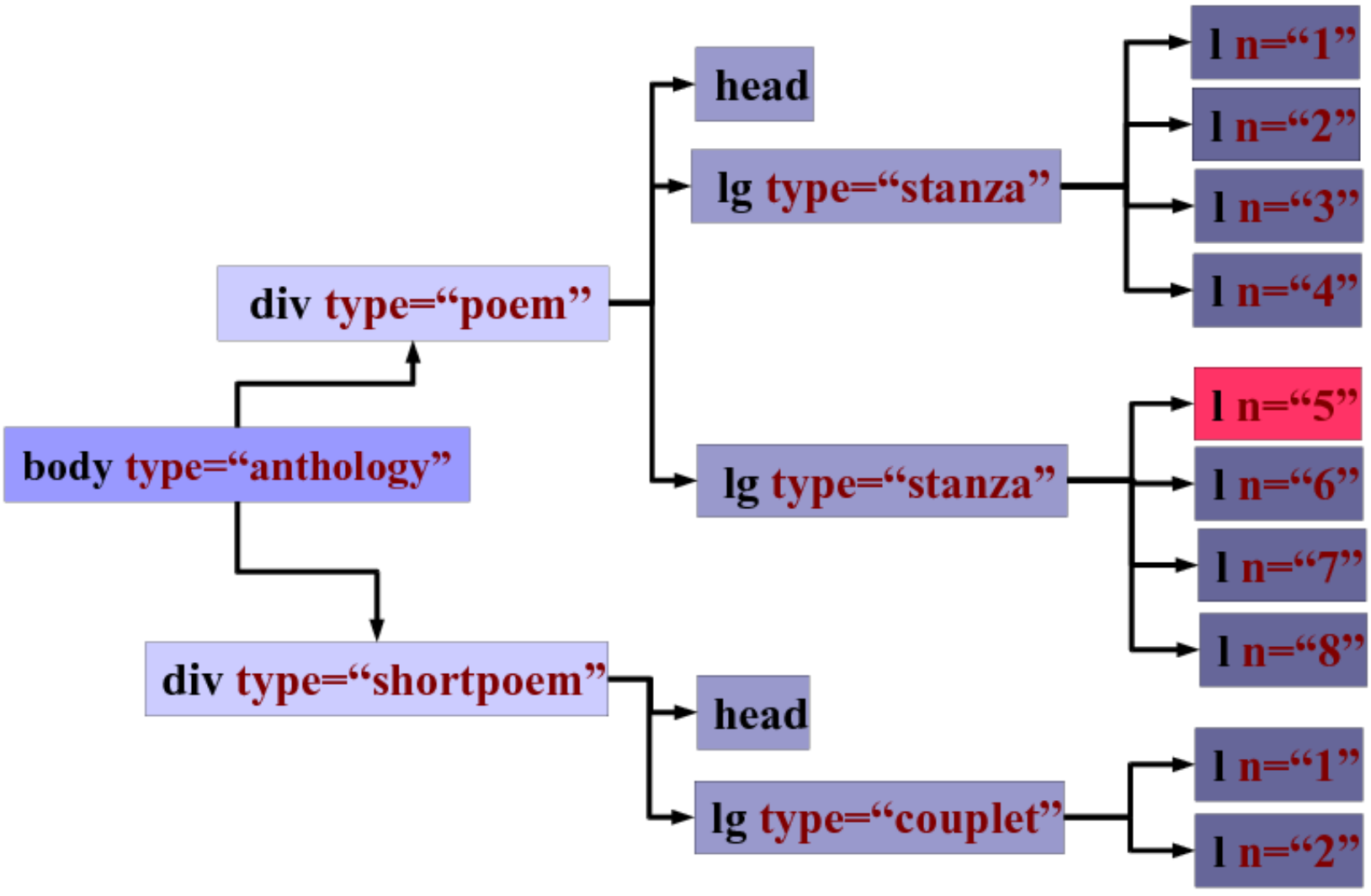
# //div[@type="poem"]//l



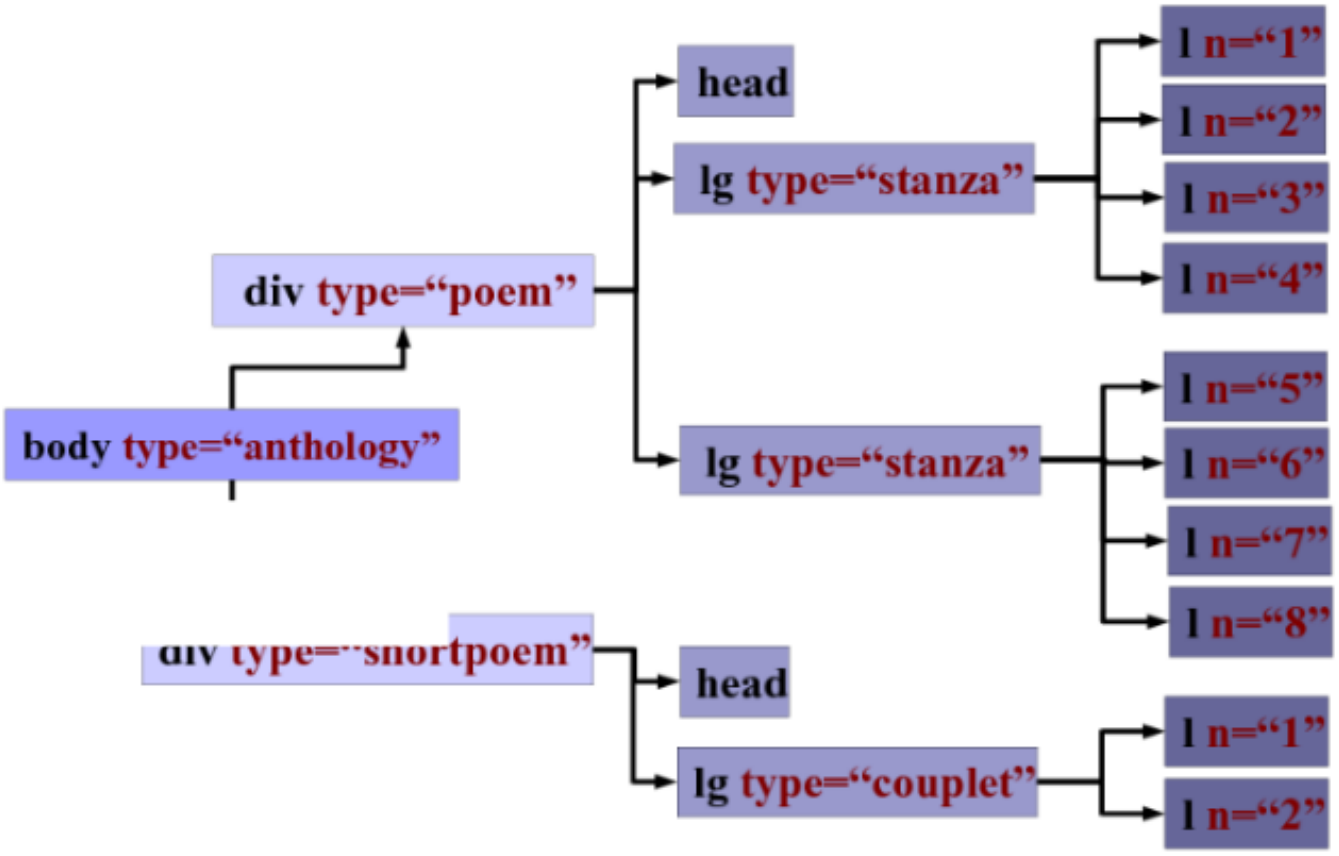
# //l[5] ?


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"

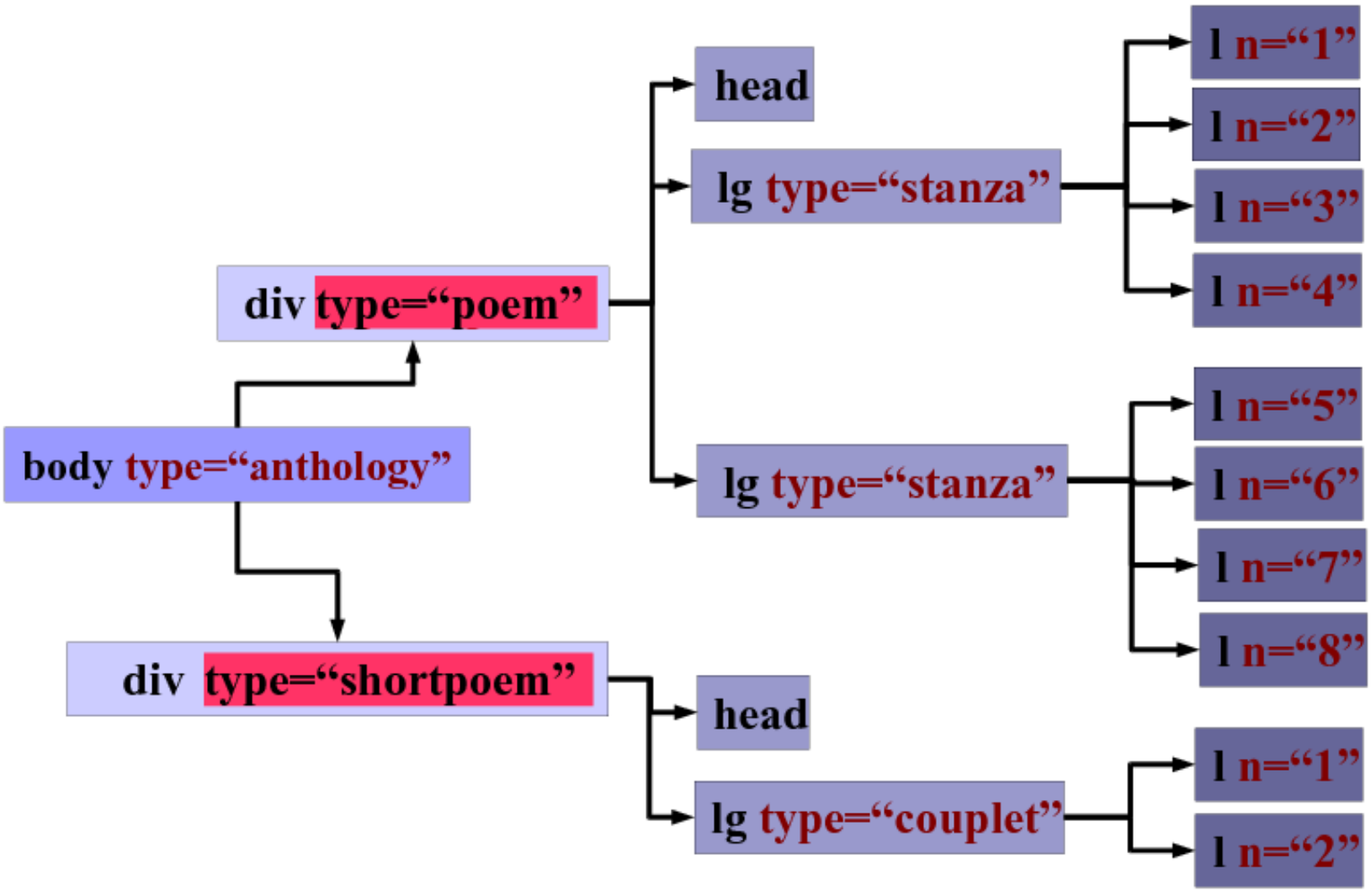
# //l[5]


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"

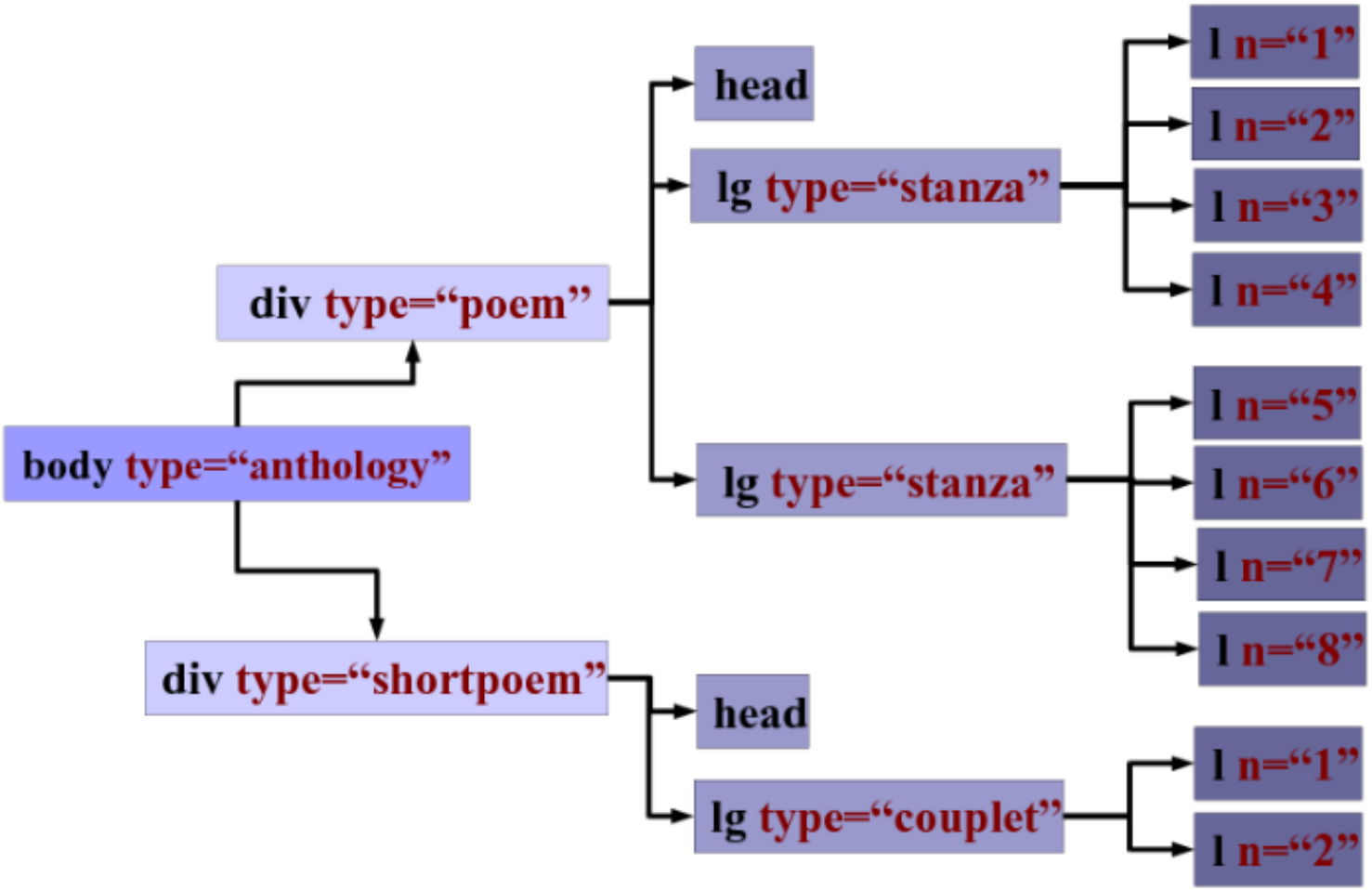
# //lg/../@type ?


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"

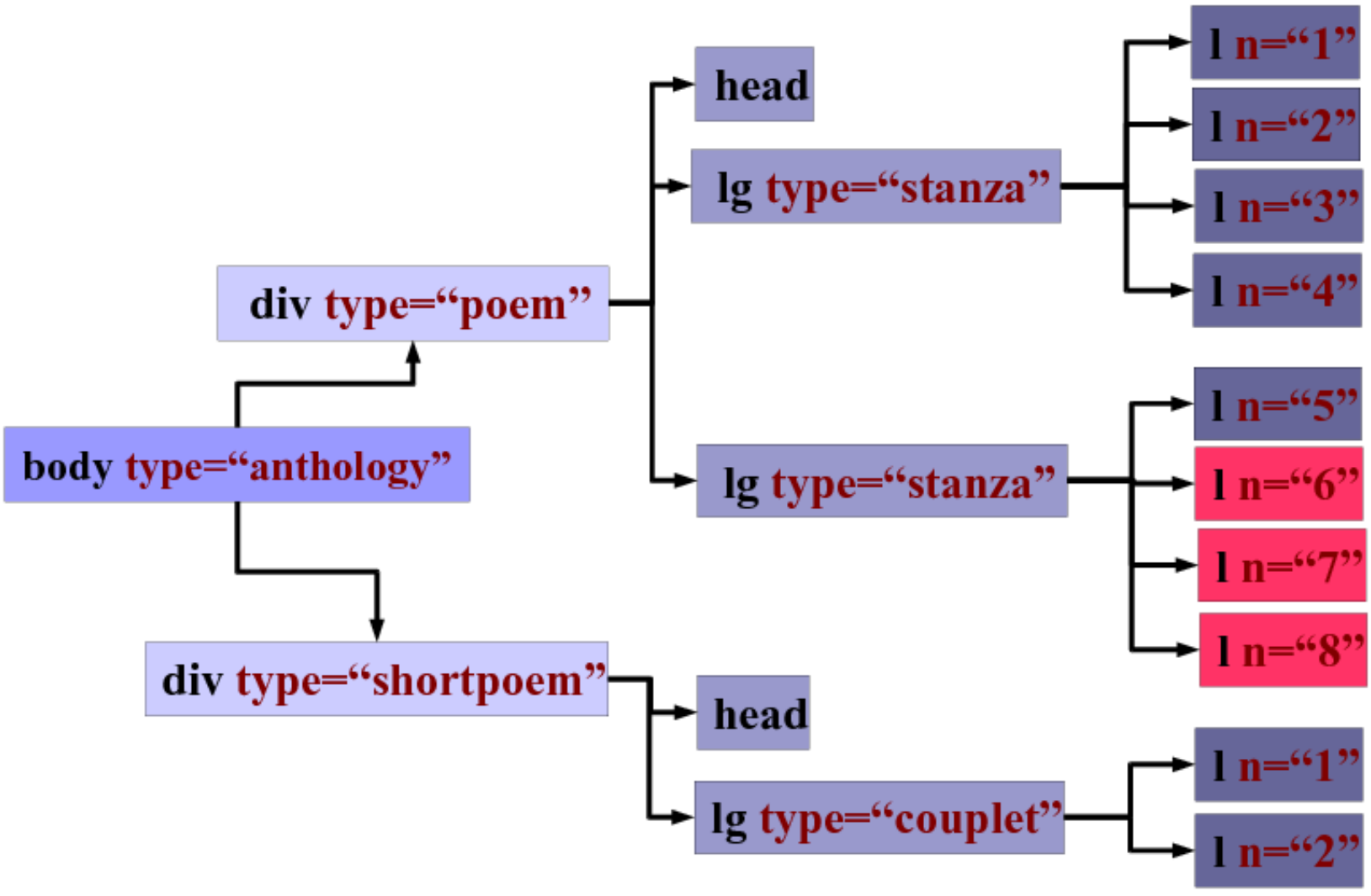
# //lg/../@type


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] ?


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]


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"

# Fonctions XPath

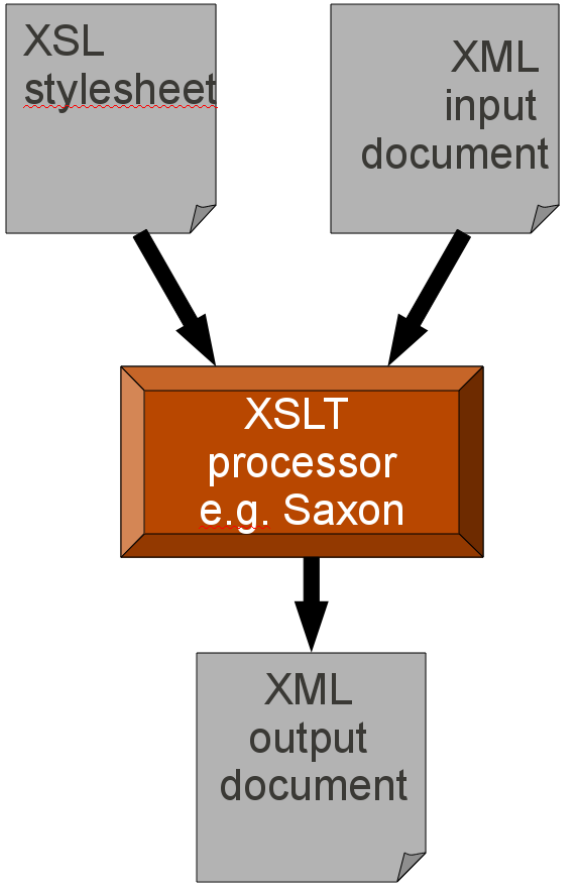
XPath fournit également une librarie extensive de fonctions utiles. On signale ici seulement quelques-unes :

- `count(x)` fournit le nombre des noeuds dans l'arbre x
- `position()` fournit le nombre ordinal du noeud courant par rapport à son contexte
- `last()` fournit fournit le nombre ordinal du dernier noeud courant par rapport à son contexte
- `contains(x,y)` test l'existence de la chaine `y` dans le morceau de texte `x`

# Premier exercice

Tester votre compréhension de XPath avec la première partie de l'exercice...

# Comment se servir d' XSLT ?




XSLT est un langage de transformation

# Une transformation typique

A partir de ceci :

```
<div type="recette" n="34">
  <head>Pasta pour les debutants</head>
  <list>
    <item>pates</item>
    <item>fromage râpé</item>
  </list>
  <p>Faire bouiller les pates, et melanger avec le fromage.</p>
</div>
```

on veut produire :

```
<html>
  <h1>34: Pasta pour les novices</h1>
  <p>Ingrédients: pates fromage râpé</p>
  <p>Faire bouiller les pates, et melanger avec le fromage.</p>
</html>
```

# Comment exprimer cela en XSL?

```
<xsl:stylesheet
   xpath-default-namespace="http://www.tei-c.org/ns/1.0"  version="2.0">
  <xsl:template  match="div">
    <html>
      <h1>
        <xsl:value-of  select="@n"/>:
      <xsl:value-of  select="head"/>
      </h1>
      <p>Ingrédients:
      <xsl:apply-templates  select="list/item"/>
      </p>
      <p>
        <xsl:value-of  select="p"/>
      </p>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

# Une feuille de style XSLT

- est un document XML, contenant des éléments de l'espace de noms `http://www.w3.org/1999/XSL/Transform`
- <xsl:stylesheet> (élément racine de tout stylesheet) permet de spécifier tous les noms d'espace utilisés, un nom d'espace par defaut, et la version du norme XSLT employé (1 ou 2)
- <xsl:output> : spécifie quelques options pour l'arbre de sortie, par exemple son format (HTML, XML, TEXT...), encodage (ISO-8859-1, UTF-8 ...) etc.

```
<xsl:stylesheet
   xpath-default-namespace="http://www.tei-c.org/ns/1.0"  version="2.0">
  <xsl:output  method="html"  encoding="ISO-8859-1"/>
  <xsl:template  match="/">
    <xsl:apply-templates/>
  </xsl:template>
</xsl:stylesheet>
```

# Dix éléments XSLT essentiels

- <xsl:template> spécifie un modèle de transformation
- <xsl:apply-templates/> applique des templates
- <xsl:value-of> sort une valeur
- <xsl:text> sort un morceau de texte
- <xsl:élément>, <xsl:attribute> et <xsl:comment> sortent un élément, attribut, ou commentaire
- <xsl:if> et <xsl:choose> actions conditionnels
- <xsl:for-each> bouclage des actions
- <xsl:variable> définition de variable
- <xsl:number> effectue une numerotation
- <xsl:sort> effectue un tri

# <xml:template>

Cet élément spécifie un modele (des actions) à appliquer à l' arborescence spécifiée par son attribut *@match*

Il peut contenir d'autres éléments XSL, des éléments d'autres noms d'espace (qui seront copiés), ou rien de tout.

```
<xsl:stylesheet
   xpath-default-namespace="http://www.tei-c.org/ns/1.0"  version="2.0">
  <xsl:template  match="div"><!-- .... actions pour les éléments div.....-->
  </xsl:template>
  <xsl:template  match="head"><!-- .... actions pour tous les éléments
head.....-->
  </xsl:template>
  <xsl:template  match="div/head"><!-- .... actions pour les éléments head
contenus par un div.....-->
  </xsl:template>
  <xsl:template  match="teiHeader"/>
</xsl:stylesheet>
```

# Wild cards

A part des éléments et des attributs, l'attribut match sur <xsl:template> peut indiquer...

| | |
|---|---|
| `/` | la racine du document |
| `*` | tout élément |
| `@*` | tout attribut |
| `text()` | tout morceau de texte |

```
<xsl:template match="*"><!-- actions par defaut pour tout élément -->
</xsl:template>
<xsl:template match="@*"><!-- actions par defaut pour tout attribut -->
</xsl:template>
```

```
<xsl:template match="text()"/>
```

# Les règles d'or de XSLT

Par defaut, le document est a traiter élément par élément...

1. Si aucun template ne correspond à un élément, traiter les éléments qu'il contient
2. Si aucun élément reste à traiter par regle 1, sortir les morceaux de texte contenus par l' élément
3. Un élément n'est traite que si un template lui correspond
4. L'ordre des templates dans le stylesheet est sans significance
5. Tout partie du document est traitable part tout template, eventuellement plusieurs fois
6. Un stylesheet ne peut contenir que de XML bien-forme

# Contenu d'un template

Les éléments XML d'un nom d'espaces autre que le XSL se trouvant dans un template sont sortis sans changement.

Les fragments textuels (plus ou moins) pareils.

Un template vide requiert la sortie de ... rien, donc (s'il est invoqué) il supprime les noeuds concernes.

# Plusieures templates peuvent etre spécifies pour un meme élément en des contextes divers

Comparer

```
<xsl:template match="head"> ....
</xsl:template>
```

avec

```
<xsl:template match="div/head"> ...
</xsl:template>
<xsl:template match="figure/head"> ....
</xsl:template>
```

En cas de conflit, c'est le template le plus spécifique qui gagne.

# <xsl:apply-templates>

Cet élément rend disponible dans le contexte courant les regles contenus par les templates indiques dans son attribut *@select*. Si aucun template n'est indiqué, tous les templates sont disponibles.

```
<xsl:template match="/">
  <html>
    <xsl:apply-templates/>
  </html>
</xsl:template>
```

```
<xsl:template match="TEI">
  <xsl:apply-templates select="text"/>
</xsl:template>
```

Il est tres utile pour varier l'ordre des sorties:

```
<xsl:template match="text">
  <h1>Corps du texte</h1>
  <xsl:apply-templates select="body"/>
  <h1>Pièces liminaires</h1>
  <xsl:apply-templates select="front"/>
  <xsl:apply-templates select="back"/>
</xsl:template>
```

# <xsl:value-of>

Cet élément fait sortir la valeur d'un élément ou d'un attribut :

```
<xsl:value-of
   select="/TEI/teiHeader/fileDesc/titleStmt/title"/>
```

Attention aux doublons potentiels!

```
<xsl:template match="div">
  <h2>
    <xsl:value-of select="@n"/>
    <xsl:value-of select="head"/>
  </h2>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="div/head"/>
```

# Attention: à ne pas confondre...

- <xsl:value-of select="XX"> fait sortir le contenu des noeuds indiques par le XPath "XX".
- <xsl:apply-templates select="XX"> fait traiter les templates disponibles pour les noeuds indiques par le XPath "XX"
- <xsl:template match="XX"> n'a aucun effet : il definit un correspondence entre un template et un élément

# Evaluation des valeurs d'attribut

On a :

```
<ref target="http://www.gallica.bnf.fr">site Gallica</ref>
```

On veut :

```
<a href="http://www.gallica.bnf.fr"/>
```

Ceci ne sera *pas* efficace :

```
<xsl:template match="ref">
  <a href="@target">
    <xsl:apply-templates/>
  </a>
</xsl:template>
```

parce qu'il donnera à l'attribut @*href* la valeur '@target'!

# Une astuce syntaxique...

On utilise {} pour indiquer qu'une expression doit être evaluée:

```
<xsl:template match="ref">
  <a href="{@target}">
    <xsl:apply-templates/>
  </a>
</xsl:template>
```

Ceci donnera à l'attribut *@href* la valeur de l' attribut *@target* quelle que soit la valeur de ce dernier

# <xsl:élément>, <xsl:attribute>, <xsl:comment>

Ces éléments font apparaitre des éléments XML dans l'arbre de sortie. Ils sont un peu plus verbose que les abbréviations presentées jusqu'à là, mais peut-être plus élégant...

```
<xsl:template match="ref">
  <xsl:élément name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="@target"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </xsl:élément>
</xsl:template>
```

est l'equivalent de

```
<xsl:template match="ref">
  <a href="{@target}">
    <xsl:apply-templates/>
  </a>
</xsl:template>
```

# <xsl:text> : faire sortir un morceau de texte

Cet élément est utile pour englober un morceau de texte explicitement (au lieu de le donner directement dans le corps du template).

```
<xsl:template match="item">
  <xsl:élément name="p">
    <xsl:text> ITEM </xsl:text>
    <xsl:apply-templates/>
  </xsl:élément>
</xsl:template>
```

# <xsl:if> : faire une épreuve

Cet élément permet des actions conditionelles :

```
<xsl:template match="person">
  <xsl:if test="@sex='1'">
    <li>
      <xsl:value-of select="persName"/>
    </li>
  </xsl:if>
</xsl:template>
```

cf.

```
<xsl:template match="person[@sex='1']">
  <li>
    <xsl:value-of select="persName"/>
  </li>
</xsl:template>
<xsl:template match="person"/>
```

# <xsl:choose>: faire un choix

Cet élément permet des actions conditionelles un peu plus complexes :

```
<xsl:template match="person">
  <xsl:apply-templates/>
  <xsl:choose>
    <xsl:when test="@sex='1'">(mâle)
    </xsl:when>
    <xsl:when test="@sex='2'">(femelle)
    </xsl:when>
    <xsl:when test="not(@sex)">(sexe non spécifié)
    </xsl:when>
    <xsl:otherwise>(code de sexe inconnu: <xsl:value-of select="@sex"/>)
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

# <xsl:for-each> : faire une iteration

Cet élément permet un bouclage des actions :

```
<xsl:template match="listPerson">
  <ul>
    <xsl:for-each select="person">
      <li>
        <xsl:value-of select="persName"/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

cf.

```
<xsl:template match="listPerson">
  <ul>
    <xsl:apply-templates select="person"/>
  </ul>
</xsl:template>
<xsl:template match="person">
  <li>
    <xsl:value-of select="persName"/>
  </li>
</xsl:template>
```

# <xsl:variable> : creation de variable

Cet élément permet d'associer un nom avec un ensemble de noeuds, ou avec une chaine de caracteres, par exemple pour faciliter sa ré-utilisation plusieurs fois dans un même stylesheet.

```
<xsl:variable  name="modernise">oui
</xsl:variable>
<xsl:if  test="$modernise = "oui""> ....
</xsl:if>
```

# <xsl:number> : numerotation

On peut generer une numérotation derivée de la séquence des éléments dans l'arborescence XML...

1. par rapport a l'élément parent:

```
<xsl:template match="p">
  <xsl:number/>
</xsl:template>
```

2. par rapport au document entier :

```
<xsl:template match="p">
  <xsl:number level="any"/>
</xsl:template>
```

3. par rapport a un élément ancêtre spécifique:

```
<xsl:template match="l">
  <xsl:number level="any" from="lg"/>
</xsl:template>
```

# <xsl:sort> : faire un tri

Cet élément permet de trier un ensemble de noeuds resultant d'un élément <xsl:apply-templates> ou <xsl:for-each>.

```
<xsl:template match="text">
  <ul>
    <xsl:for-each select="//persName">
      <xsl:sort select="normalize-space(.)" data-
type="text" order="ascending"/>
      <li>
        <xsl:value-of select="."/>
      </li>
    </xsl:for-each>
  </ul>
</xsl:template>
```

# Sommaire

Maintenant vous savez comment

1. créer des templates
2. séléctionner des morceaux de texte
3. ajouter des éléments
4. définir des actions conditionnelles
5. numéroter et trier les objets de sortie

On va expérimenter cela dans la deuxieme partie de l'exercice....

# Pour en savoir plus

- La formation de SPQR demain!
- A http://www.gchagnon.fr/cours/xml/ vous trouverez deux cours complets et très clairs
- Un texte classique: Philippe Rigaux et Bernd Amann *Comprendre XSLT*. O'Reilly, 2002.
- Beaucoup, beaucoup, d'autres ressources anglophones...