Module BD et Sites Web

Cours 6 – XSLT

Remerciements à Bernd Amann

Plan

- Axes XPath
- Introduction à XSLT : comprendre les mécanismes du langage
  - Règles XSLT
  - Instructions XSLT
  - Application : de XML à HTML

XPath (axes)

• Position (noeud courant)	self	.
• Descendants		
– direct	child	/
– indirect	descendant	//
• Prédécesseurs		
– direct	parent	..
– indirect	ancestor	
• Frères		
– même niveau	following /preceding	
– navigation	following-/preceding-sibling	

- Autres
  - ancestor-or-self, descendant-or-self, namespace, attribute (@)

Axes

- L'axe enfant (**`child`**) contient les enfants du nœud contextuel. L'axe enfant est l'axe par défaut, et il peut être omis.
- L'axe **`self`** renvoie le nœud courant.
- L'axe **`parent`** contient le parent du nœud contextuel, s'il en a un.
- L'axe **`descendant`** contient tous les descendants du nœud contextuel (enfant, petit-enfant, etc.), à l'exception des nœuds attributs et espaces de nom.
- L'axe ancêtre (**`ancestor`**) contient tous les éléments ancêtres du nœud contextuel (parent, parent du parent, etc.). Il contient forcément le nœud racine (sauf si le nœud contextuel est la racine).

Axes

- L'axe **`following-sibling`** contient tous les nœuds frères qui suivent le nœud contextuel.
- L'axe **`preceding-sibling`** contient tous les frères prédécesseurs du nœud contextuel.
- L'axe suivant (**`following`**) contient tous les nœuds du même document que le nœud contextuel qui sont après le nœud contextuel dans l'ordre du document, à l'exclusion de tout descendant, des attributs et des espaces de noms.
- L'axe cible précédente (**`preceding`**) contient tous les prédécesseurs du nœud contextuel à l’exclusion des ancêtres; si le nœud contextuel est un attribut ou un espace de noms, la cible précédente est vide.

Axes

- L'axe **`descendant-or-self`** contient le nœud contextuel et ses descendants

- L'axe **`ancestor-or-self`** contient le nœud contextuel et ses ancêtres; ainsi l'axe **`ancestor-or-self`** contient toujours le nœud racine

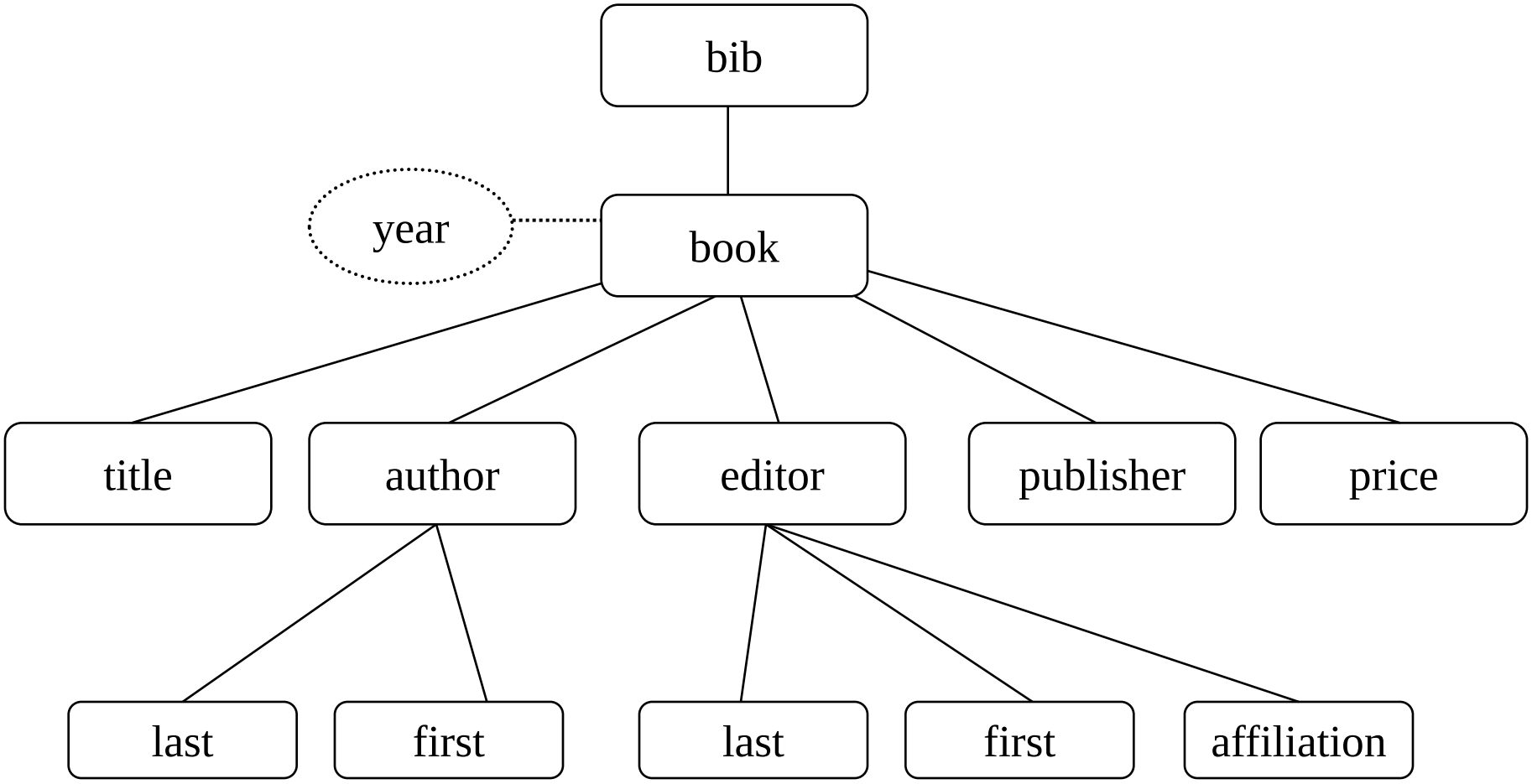
- Les axes **`ancestors, descendants, following, preceding`** et **`self`** partitionnent un document (ignorant les attributs et les nœuds d'espace de nom) : il ne se chevauchent pas et ensemble ils contiennent tous les nœuds d'un document

```
//author/ancestor::* |//author/descendant::* |
  //author/following::* | //author/preceding::* |
  //author/self::*
```

renvoie tout le document.

Le symbole | permet de combiner des chemins.

Exemple


bib
year
book
title
author
editor
publisher
price
last
first
last
first
affiliation

Exemples

- `//editor/parent::*` **renvoie les parents du nœud** `editor`**, càd** `book` **et** `bib`
- `/book/editor/descendant::*` **renvoie les descendants de** `editor`**, càd** `last, first, affiliation`**.**
- `//last/ancestor::*` **renvoie tous les ancêtres de** `last`**, càd** `author, editor, book, bib`
- `//editor/following-sibling::*` **renvoie les nœuds** `publisher, price`
- `//editor/preceding-sibling::*` **renvoie les nœuds** `author, title`
- `//author/following::*` **renvoie les nœuds** `editor, last, first, affiliation, publisher, price`
- `//author/preceding::*` **renvoie les nœuds** `title`

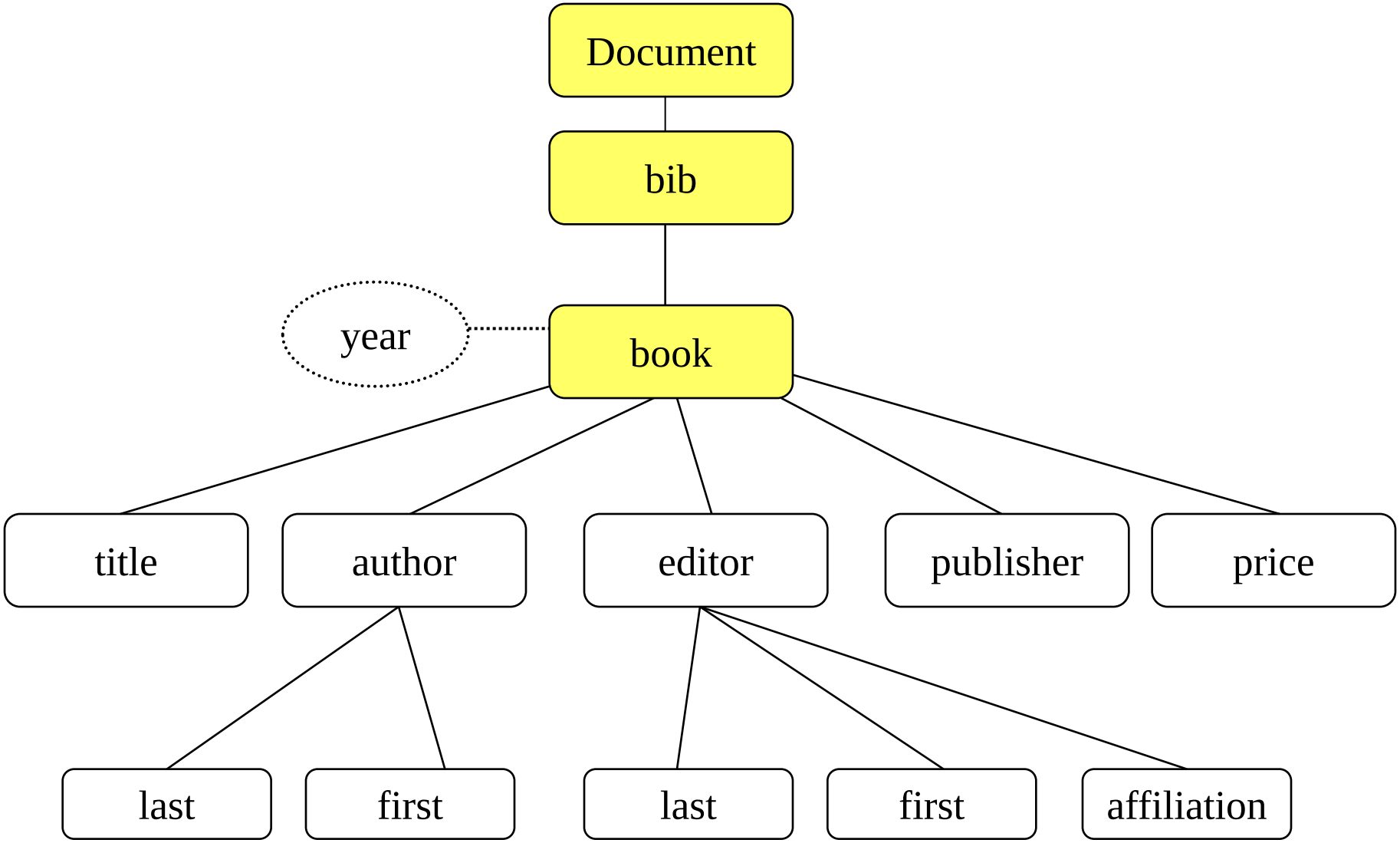
Exemples (suite)

- `//publisher/ancestor::*` renvoie les ancêtres du nœud `publisher`, càd `book`, `bib` et `document`

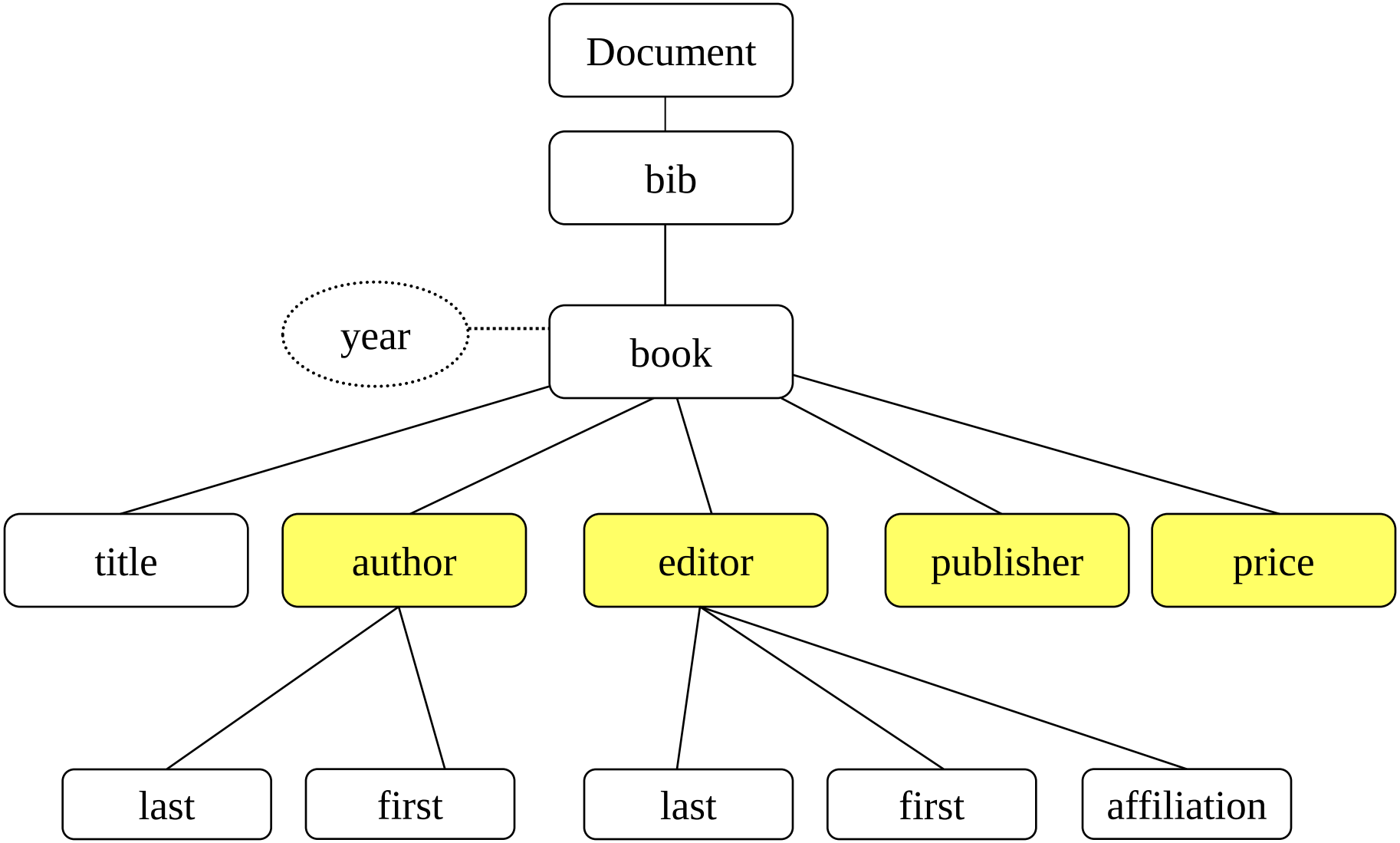
- `//title/following-sibling::*` renvoie les nœuds `author`, `editor`, `publisher` et `price`

- `//publisher/preceding-sibling::*` renvoie les nœuds `editor`, `author`, `title`

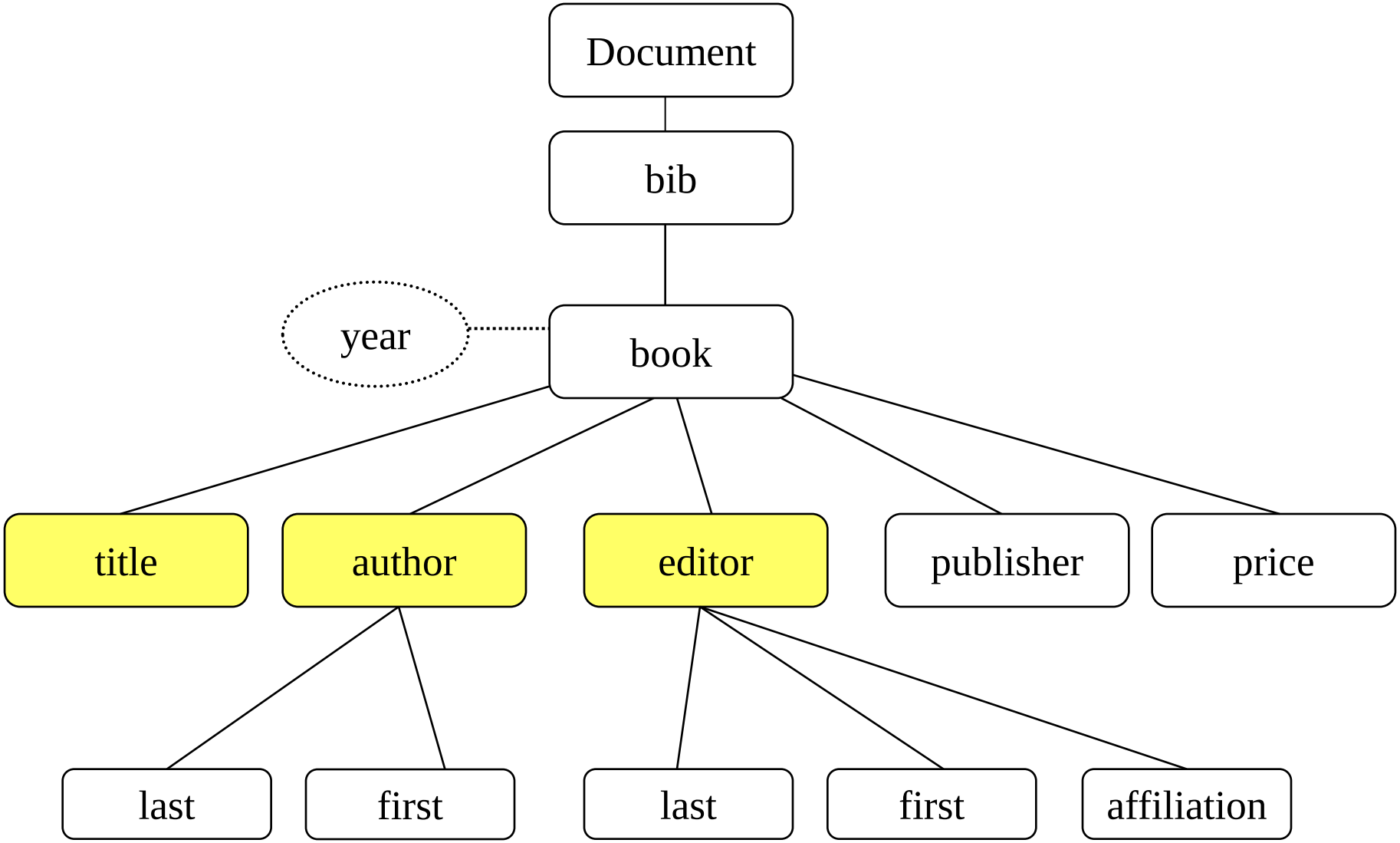
//publisher/ancestor::*



//title/following-sibling::*



//publisher/preceding-sibling::*



XPath (abréviations)

`child::` **est l'axe par défaut, et peut être omis**

`/child::book` **est équivalent à** **`/book`**

`child::book/child::title` **peut s'écrire** **`book/title`**

`attribute::` **peut être remplacé par** **`@`**

`child::book[attribute::year= "2002"]` **peut s'écrire**
`book[@year= "2002"]`

`//` **est l'abréviation de** **`/descendant-or-self::node()/`**

`.` **est l'abréviation de** **`self::node()`**

`..` **est l'abréviation de** **`parent::node()`**

XSLT

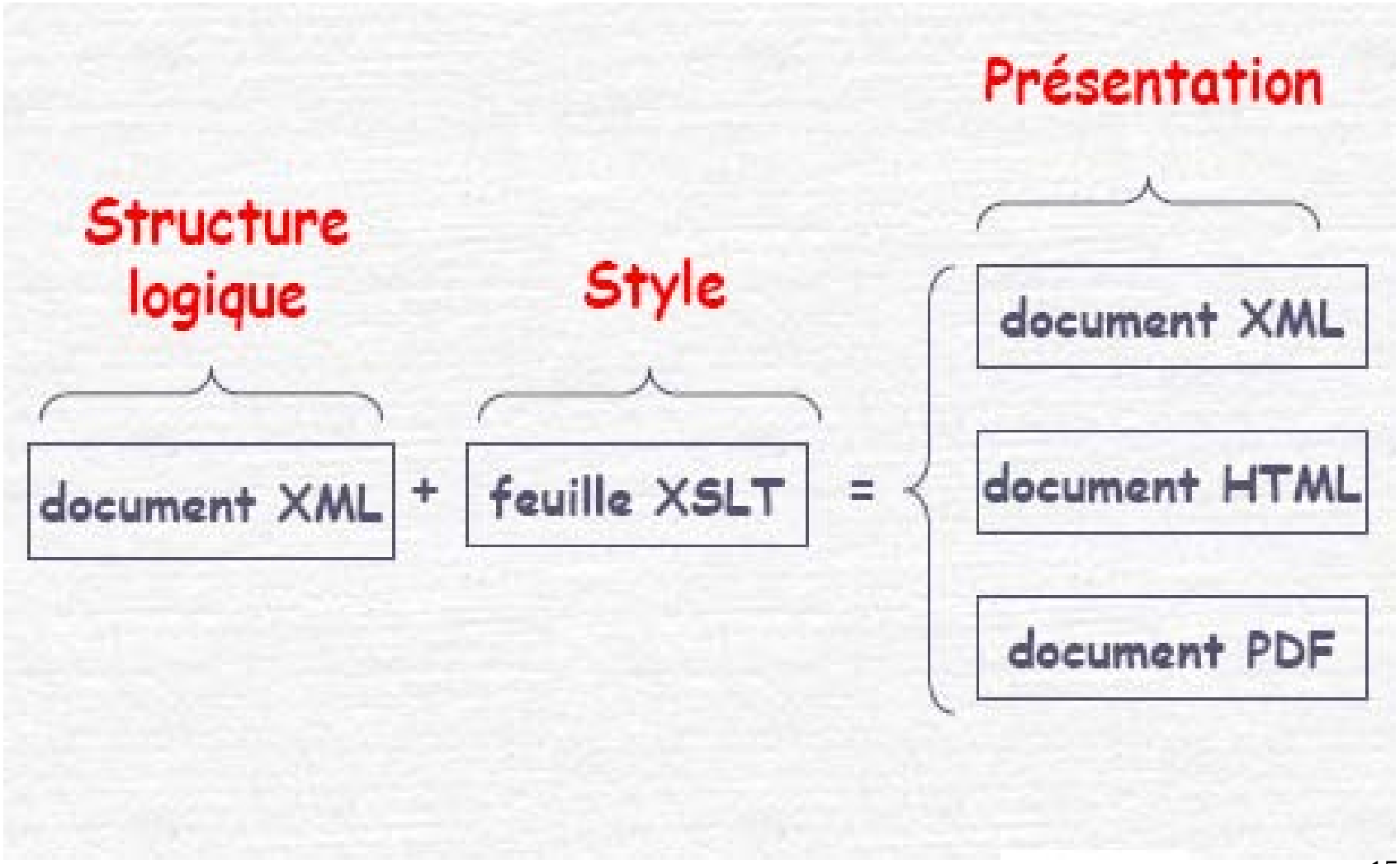
- ✓ Principe : séparer la gestion et la structuration du contenu d'un document, de sa présentation

- ✓ XSLT : traduction d'un document XML en un document sous forme

  - HTML : présentation Web standard
  - WML : présentation Wap
  - SMIL : présentation multimedia
  - XSL-FO (PDF) : production de documents papier
  - LateX : documents papier
  - Etc.

XSLT

(tiré de F-Y Villemin)



Un document sous forme XML

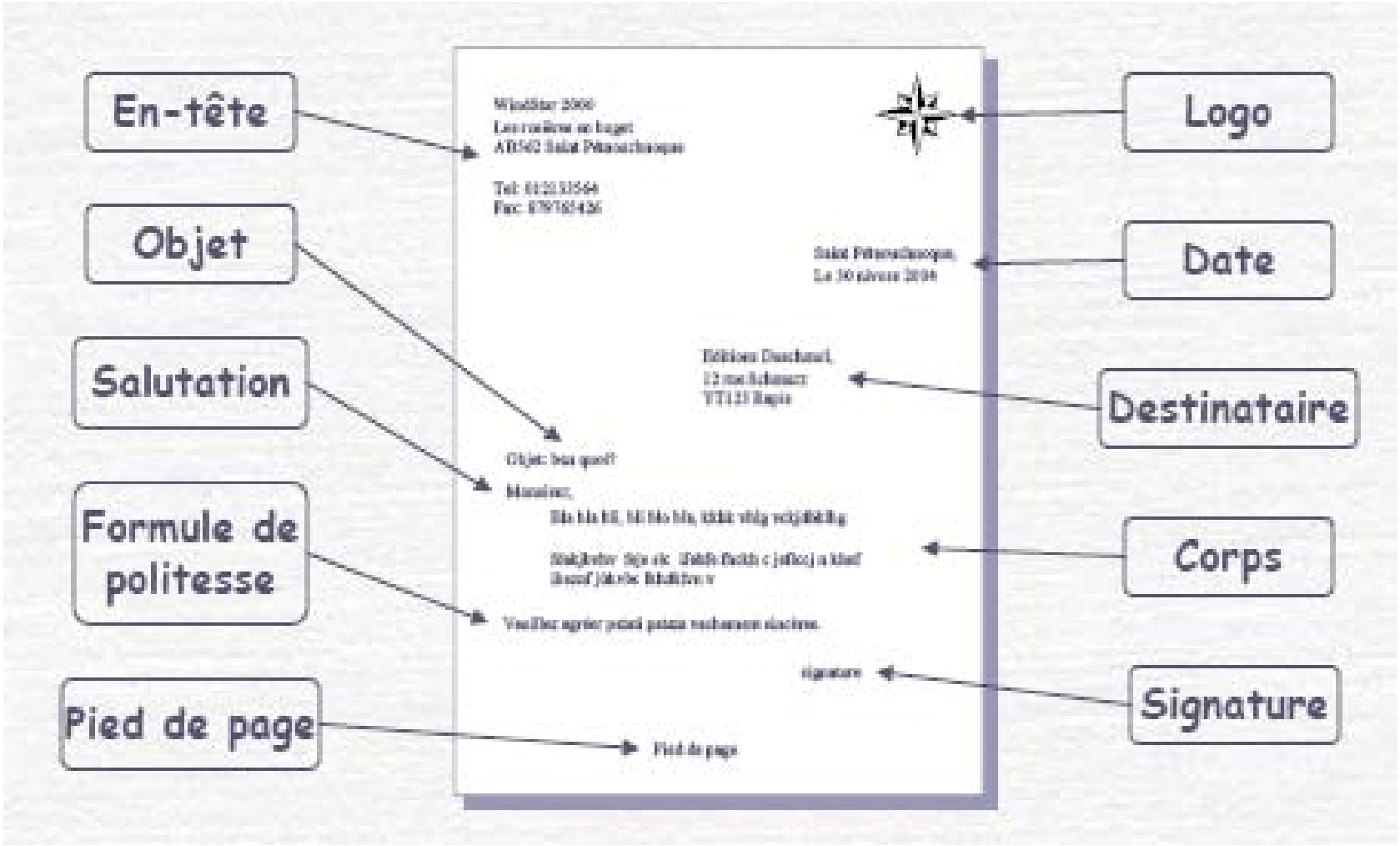
(Abiteboul et al, F.-Y. Villemin)

```
<lettre>
<entete>
  <logo loc="logo-
    graph.vml"/>
  <adresse>
    &abrev-adresse;
  </adresse>
</entete>
<destinataire>
  <nom > Editions Duschmoll
    </nom>
  <adresse>
    <rue>
      12 rueSchmurz
    </rue>
    <ville>
      YT123 Rapis
    </ville>
  </adresse>
</destinataire>
<objet> bon quoi?</objet>
...
<date>
    Le 30 Nivose 2004
</date>

<salutation>
    Monsieur,
</salutation>

<corps>
    <para>
   Bla bla bli, bli blo bla, kkkk ...
</para>
    <para>
  fdskjbvhv  feje slc  ifehfe fhckh ...
    </para>
</corps>
...
</lettre>
```

Et sa présentation



Structure de base : les règles

- Règle = *template* : élément de base pour produire le résultat.
- Une règle s'applique dans le contexte d'un nœud de l'arbre
- L'application de la règle produit un fragment du résultat
- Un programme XSLT = ensemble de règles pour construire un résultat

Programme XSLT

- Un programme XSLT est un document XML bien formé contenant des éléments **`xsl`** (préfixe **`xsl:`**), et éventuellement d'autres éléments XML.
- Il a comme racine l'élément :

```
<xsl:stylesheet
   version='1.0'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
```

- Un programme XSLT s'applique à un document XML et produit un document résultat.

Programme XSLT

- Sous l'élément racine **`<xsl:stylesheet>`** on a des *éléments de premier niveau* utilisés pour transformer le document source (définir les règles) et produire le document résultat, et des *instructions*.

- L'ordre des éléments n'a pas d'importance

- Les instructions sont dans le corps des règles et sont interprétées par le processeur.

Principaux éléments de premier niveau

`xsl:import`	Import d'un pgm XSLT
`xsl:include`	inclusion d'un pgm XSLT
`xsl:output`	indique le format de sortie
`xsl:param`	définit un paramètre
`xsl:strip-space`	supprime les blancs pour certains éléments
`xsl:template`	définit une règle XSLT
`xsl:variable`	définit une variable XSLT

Exécution d’un programme XSLT

L’exécution d’un programme XSLT consiste à instancier des règles :

- ✓ Le corps de la règle est inséré dans le document résultat
- ✓ Les instructions XSLT contenues dans le corps de la règle sont exécutées à leur tour
- ✓ L’instruction est remplacée par son résultat dans le document résultat.

Fonction d’une feuille de style XSLT

- Fonction de base : langage de règles de transformation de documents XML (sous forme d’arbre) :
  - Extraction de données
  - Génération de texte
  - Suppression de contenu (nœuds)
  - Déplacer du contenu (nœuds)
  - Dupliquer du contenu (nœuds)
  - Trier

Définition

- Une règle est définie par l'élément **`<xsl:template>`**
- Cet élément a 4 attributs :
  - **`match`** est le pattern XPATH définissant les cibles de la règle (l'expression XPATH doit toujours désigner un ensemble de nœuds)
  - **`name`** donne un nom à la règle, qui pourra être appelée par ce nom.
  - **`mode`** définit des catégories de règle
  - **`priority`** donne une priorité explicite à la règle
- Ces attributs sont optionnels, mais soit **`name`** soit **`match`** doivent être définis.

Ex : **`<xsl:template match='Film' name ='R1'>`**

Déclenchement des règles

- Les règles sont déclenchées par

`<xsl:apply-templates/>`

Comporte deux attributs optionnels :

`select` : contient l'expression XPath désignant les nœuds à traiter

`mode` : catégorie des règles à considérer

L'expression XPath de **`select`** doit renvoyer un ensemble de nœuds. S'il n'y a pas d'attribut **`select`**, tous les fils du nœud courant sont considérés.

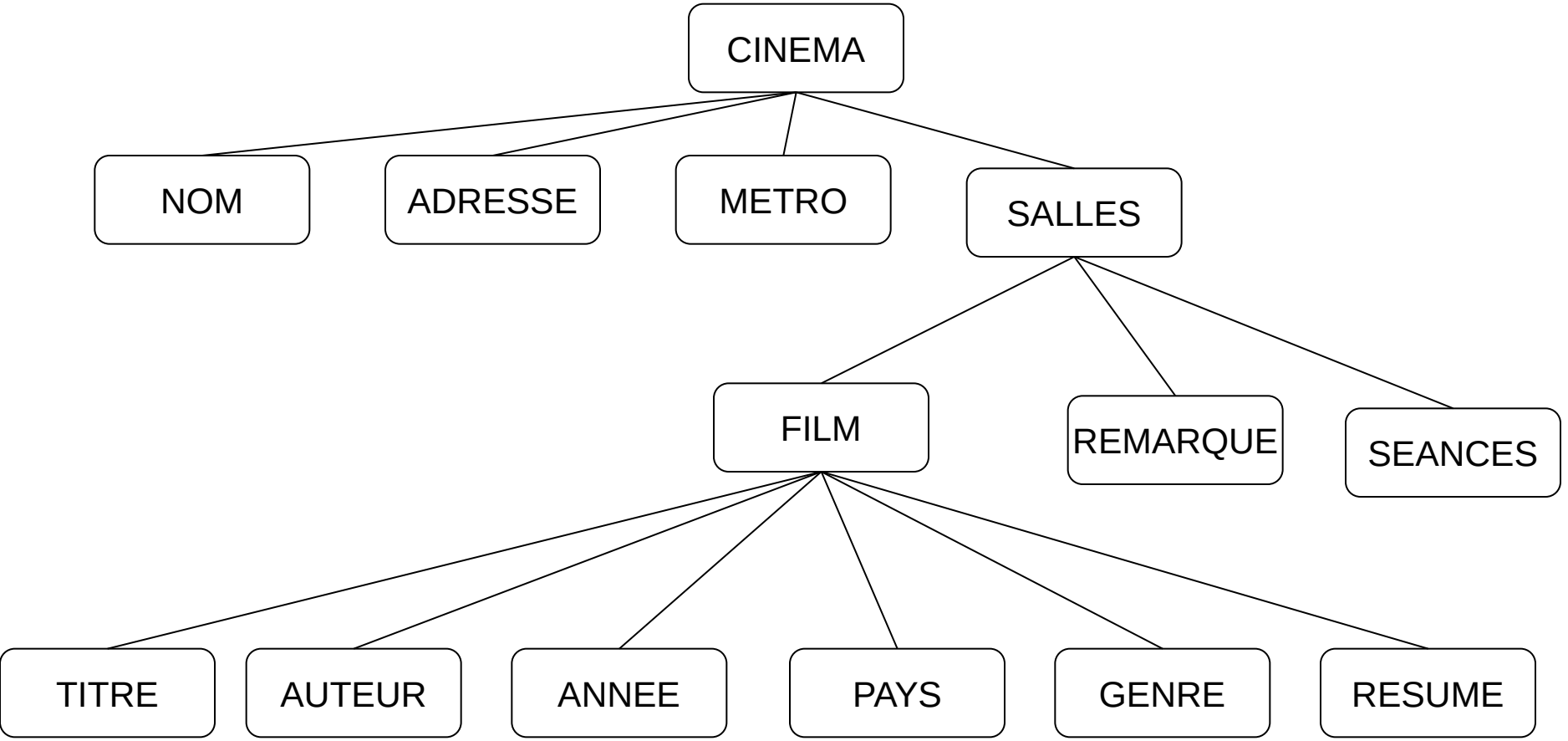
`<xsl:call-template name ='R1'/>`

S'utilise lorsqu'une règle a un nom (attribut **`name`**).

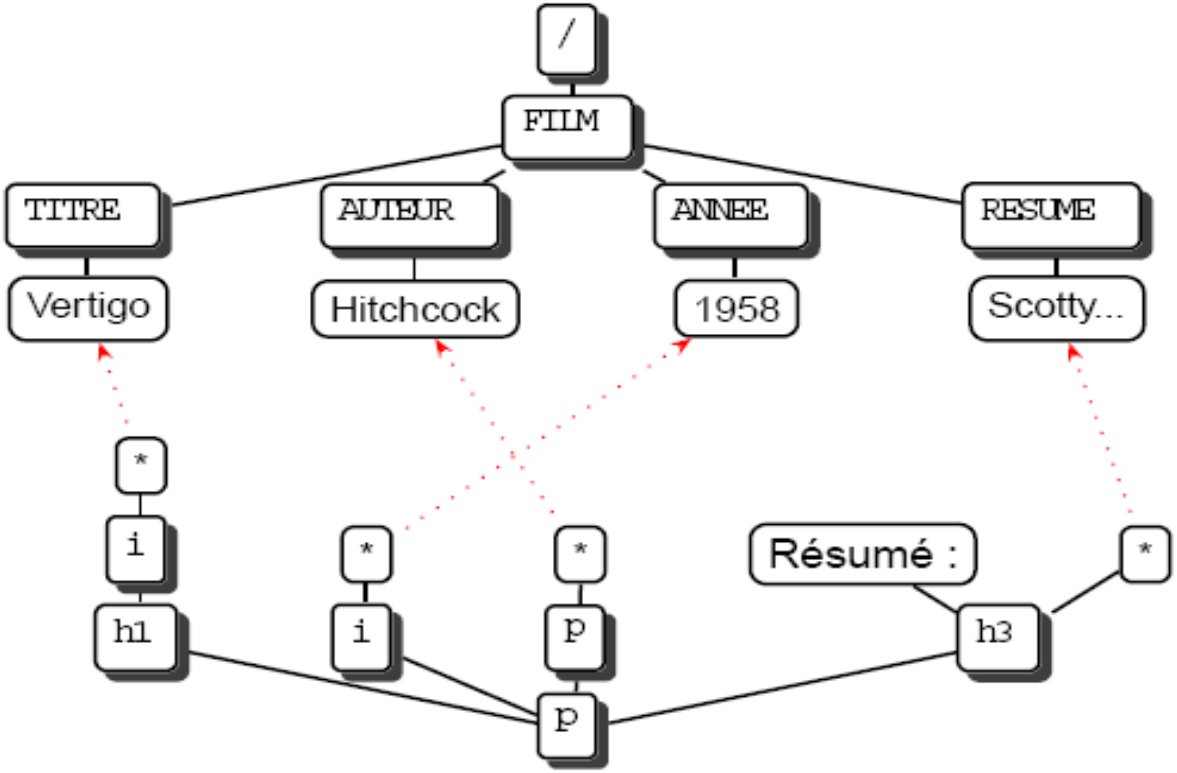
Principales instructions

xsl:apply-templates	déclenche une règle
xsl:call-template	déclenche une règle nommée
xsl:comment	insère un nœud Comment
xsl:copy	copie un nœud du document source dans le document résultat
xsl:copy-of	copie un nœud et tous ses descendants
xsl:for-each	effectue des itérations
xsl:if	branchement conditionnel
xsl:param	pour définir un paramètre
xsl:text	insère un noeud Text
xsl:value_of	évalue une expr.Xpath et insère le résultat
xsl:variable	pour définir une variable

Schéma exemple



Exemple


/
FILM
TITRE
AUTEUR
ANNEE
RESUME
Vertigo
Hitchcock
1958
Scotty...
*
i
h1
*
i
*
p
p
Résumé :
h3
*

Exemple : règle de transformation

```
1 <xsl:template match='FILM'>
2    <p>
3      <h1>
4        <i>
5         <xsl:value-of select='TITRE'/>
6        </i>
7      </h1>
8      <i>
9       <xsl:value-of select='ANNEE'/>
10     </i>
11    <p>
12      <xsl:value-of select='AUTEUR'/>
13    </p>
14    <h3>Résumé:
15      <xsl:value-of select='RESUME'/>
16    </h3>
17   </p>
18  </xsl:template>
```

Extraction de données

- Exemple : recherche du titre pour le nœud FILM :

```
<xsl:value-of  select='TITRE'/>
```

- Plus généralement, on donne un chemin d'accès XPath à un nœud à partir du nœud courant

- Nœud courant : nœud auquel s'applique la règle
- Nœud contexte : nœud d'une étape Xpath (change à chaque étape). Au départ, nœud courant=nœud contexte

Génération de texte

- Produire une phrase quand on rencontre un nœud FILM :

```
<xsl:template  match='FILM'>
    Ceci est le texte produit par
  application de cette règle
</xsl:template>
```

Génération d'un arbre XML

- Produire un document/fragment/arbre XML quand on rencontre un nœud FILM :

```
<xsl:template  match='FILM'>
     <body>
          <p>Un paragraphe</p>
     </body>
</xsl:template>
```

Génération avec extraction

- Produire un document/fragment/arbre XML quand on rencontre un noeud FILM :

```
<xsl:template  match='FILM'>
  <body>
     <p>
          <xsl:text>titre: </xsl:text>
          <xsl:value-of select='TITRE'/>
     </p>
  </body>
</xsl:template>
```

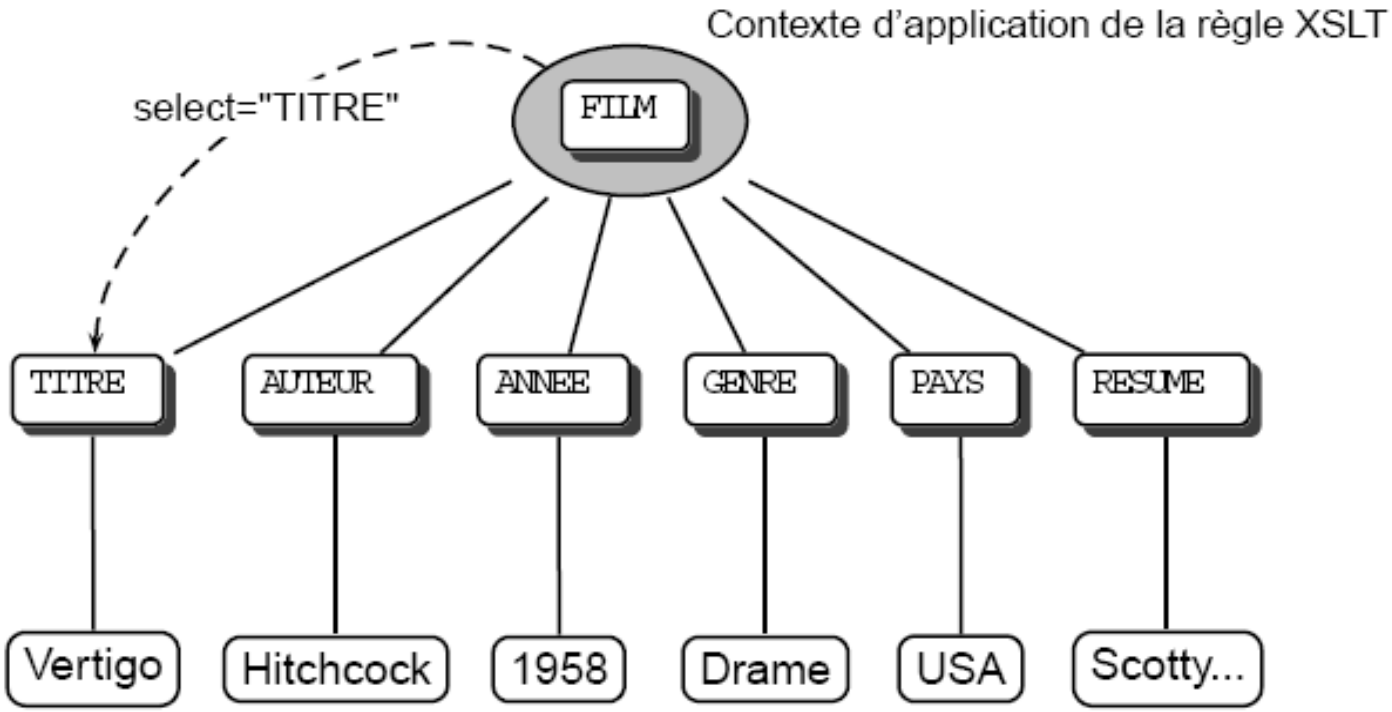
Exemple

```
1 <xsl:template   match='FILM'>
2    <p>
3      <h1>
4        <i> <xsl:value-of select='TITRE'/> </i>
5      </h1>
6     <i> <xsl:value-of select='ANNEE'/> </i>
7     <p> <xsl:value-of select='AUTEUR'/> </p>
8     <h3>Résumé:
9       <xsl:value-of select='RESUME'/>
10    </h3>
11   </p>
12 </xsl:template>
```

Motif de sélection : `match='FILM'`

Corps de la règle : fragment d'arbre à produire

Illustration



Une règle complète

```
1 <xsl:template match='FILM'>
2  <html>
3    <head>
4     <title>Film:
5        <xsl:value-of select='TITRE'/>
6     </title>
7    </head>
8    <body>
9    Le genre du film est
10  <b><xsl:value-of select='GENRE'/></b>
11   </body>
12  </html>
13 </xsl:template>
```

Le résultat

On obtient :

```
<html>
  <head>
    <title>Film:
        Vertigo
    </title>
  </head>
 <body>
      Le genre du film est
      <b>Suspense</b>
 </body>
</html>
```