

# Séance 3

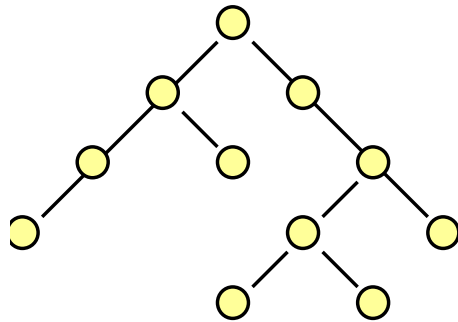
## Transformation des XML

**Prof. Yassin Aziz REKIK**  
**[Yassin.rekik@he-arc.ch](mailto:Yassin.rekik@he-arc.ch)**

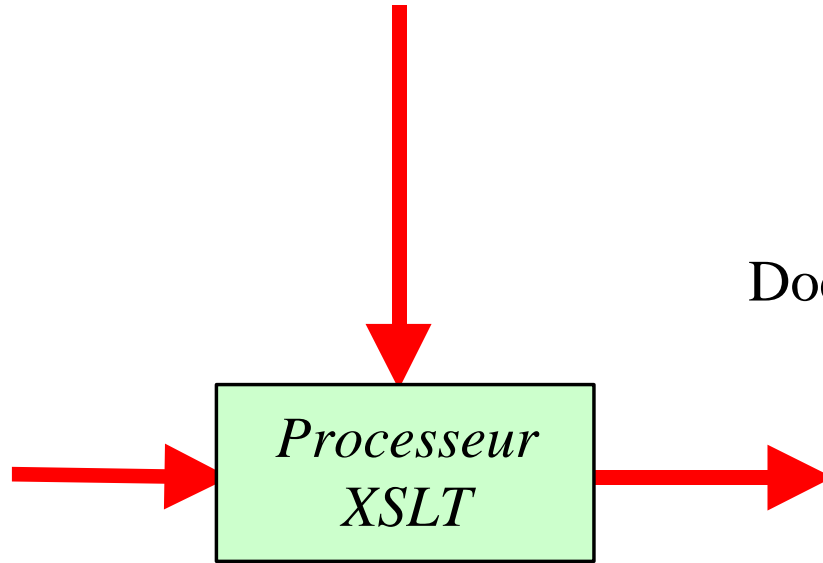
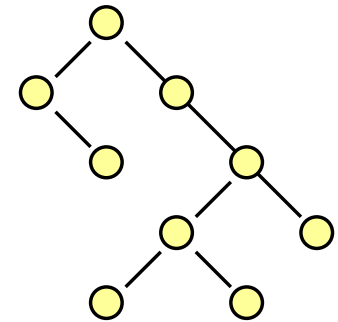
# XSLT = Transformation d'arbre

Feuille de style XSLT

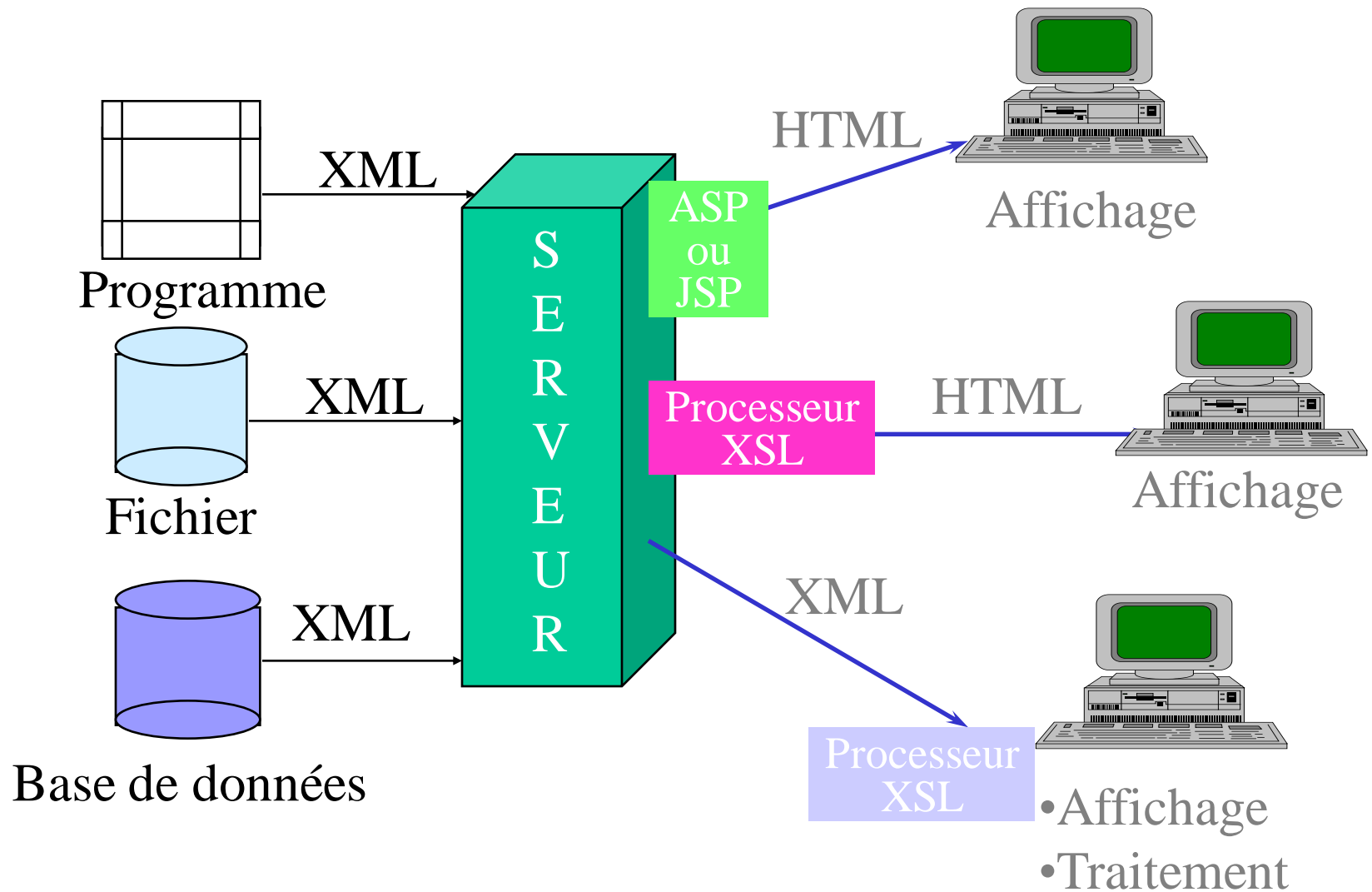
Document source



Document de sortie



# Architecture



# XSLT est un langage XML

- Les instructions sont des éléments XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
  <!-- Format de sortie -->
  <xsl:output method="xml" version="1.0"
    encoding="UTF-8" indent="yes"/>

  <!-- ... règles XSLT ... -->

</xsl:stylesheet>
```

# Prélude d'une feuille de style

# Élément <xsl:stylesheet>

- Élément racine d'un document XSLT

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

- Attribut `version` : version de langage XSL (obligatoire)
- Attribut `xmlns:xsl` : espace de nom XSL

# Élément <xsl:output>

- Format de sortie du document résultat

```
<xsl:output method="xml" version="1.0"  
encoding="UTF-8" indent="yes"/>
```

- Attribut `method` : type du document en sortie
- Attribut `encoding` : codage du document
- Attribut `indent` : indentation en sortie

# Type de document en sortie

- Trois types de document en sortie
  - **xml** : vérifie que la sortie est bien formée
    - (*sortie par défaut*)
  - **html** : accepte les balises manquantes, génère les entités HTML (&acute; ...)
    - (*sortie par défaut si XSL reconnaît l'arbre de sortie HTML4*)
  - **text** : tout autre format textuel :
    - du code Java, format Microsoft RTF, LaTeX



# Parcours - transformation d'arbre

- Règle de réécriture : template
  - `<xsl:template>`
- Spécifier un parcours de l'arbre d'entrée
  - `<xsl:apply-template>`
  - `<xsl:for-each>`
- Obtenir une valeur dans l'arbre source
  - `<xsl:value-of>`
  - les crochets dans un attribut
  - `<a href="{@src}">`

# Élément <xsl:template>

- Règle de réécriture *condition* → *action*

```
<xsl:template match="condition">  
    ... action ...  
</xsl:template>
```

- Attribut **match** : expression XPATH
- Contenu : sous-arbre en sortie
- Un programme XSLT est un ensemble de règles

# Premier exemple complet

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

```
<xsl:output method="html"/>
```

Condition :

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

« A la racine du document d'entrée ! »

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Welcome</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
  Welcome!
```

```
</BODY>
```

```
</HTML>
```

```
</xsl:template>
```

Arbre en sortie

Action :

« Document html inclus à générer ! »

```
</xsl:stylesheet>
```

# Arbre en sortie

- Le texte HTML est inclus comme contenu d'élément `<xsl:template>`
  - *le texte HTML doit être bien formé*
- On peut mélanger du texte XSLT au texte HTML
  - *pour extraire des informations du document source*
  - *pour générer un texte HTML en relation avec le contenu du document source*

```
<xsl:template match="titre">  
    <h1><xsl:value-of select="." /></h1>  
</xsl:template>
```

*Expressions xpath*



*Modèle de sous-arbre paramétré*



# Second exemple complet (1)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="no"
    encoding="iso-8859-1"/>
```

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

Expression xpath

Modèle de sous-arbre

...

# Second exemple complet (suite)

```
<xsl:template match="carnetDAdresse">
  <body>
    <h1>Liste des Noms</h1>
    <xsl:apply-templates/>
  </body>
</xsl:template>
```

```
<xsl:template match="carteDeVisite">
  <p>Nom : <xsl:value-of select="nom"/>
</p>
</xsl:template>
```

```
</xsl:stylesheet>
```

# Résultat

- Pour un document source contenant 4 cartes de visite

```
<html>
<body>
<h1>Liste des Noms</h1>
    <p>Nom : Bekkers</p>
    <p>Nom : Bartold</p>
    <p>Nom : Letertre</p>
    <p>Nom : Apolon</p>
</body>
</html>
```



# Élément

## <xsl:apply-templates>

- Descente dans les fils d'un nœud

```
<xsl:template match="carnetDAdresse">  
  <body>  
    <h1>Liste des Noms</h1>  
    <xsl:apply-templates/>  
  </body>  
</xsl:template>
```

- équivalent à

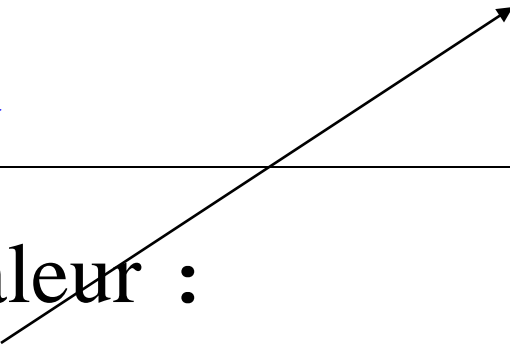
```
<xsl:apply-templates select="child::*">
```



# Élément <xsl:value-of>

- Générer le contenu d'un élément

```
<xsl:template match="carteDeVisite">  
  <br>Nom : <xsl:value-of select="nom"/>  
  </br>  
</xsl:template>
```



- Sélection de la valeur :
  - attribut `select` : expression xpath
  - ici : le texte contenu dans l'élément `nom` de l'élément `carteDeVisite`

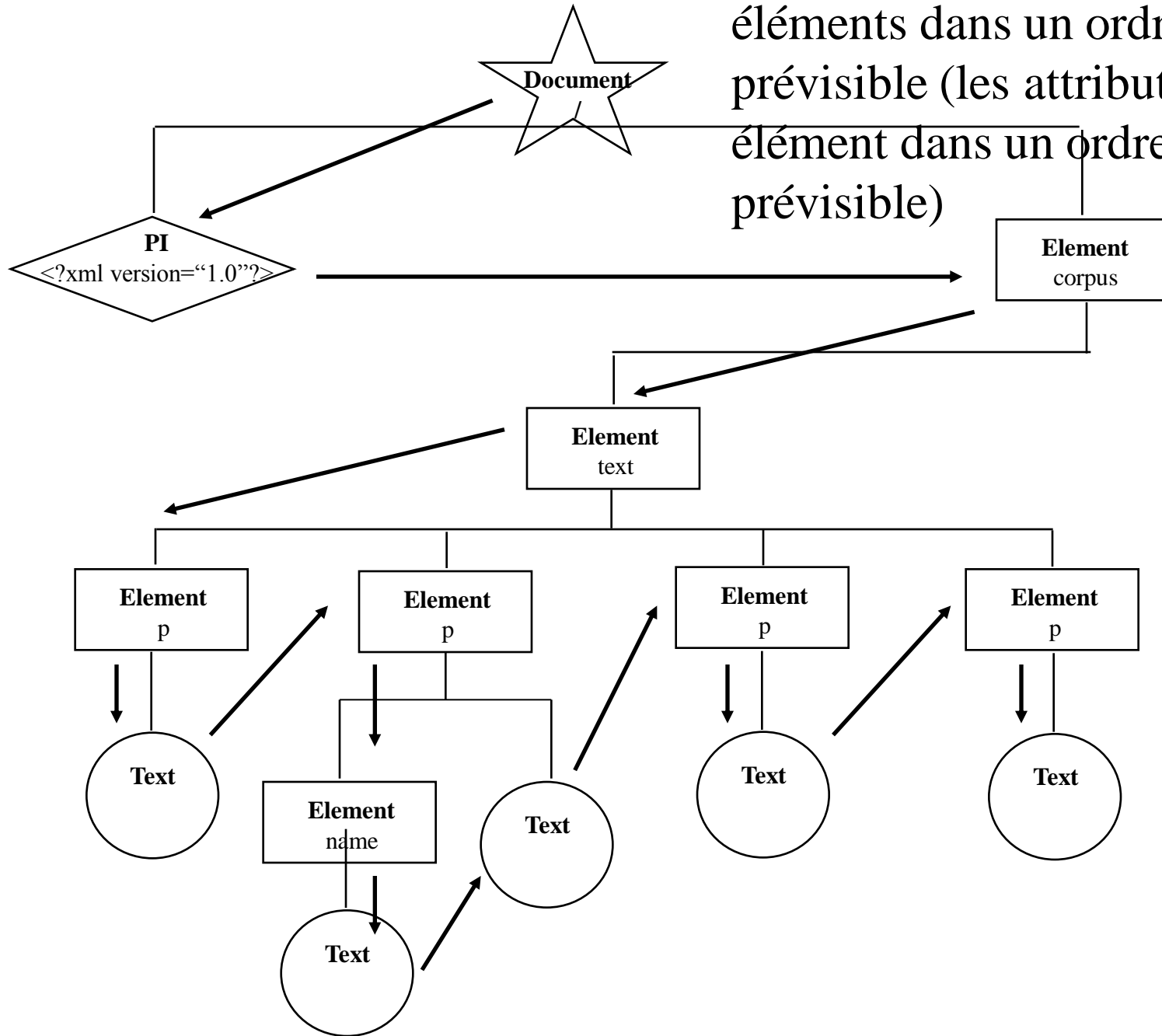
# xsl:apply-templates

- Dans l'exemple précédent, `apply-templates` sert à relancer l'analyse (la recherche d'un `template` avec un attribut `match` qui correspond) dans la sous-arborescence du nœud traité par le template
- Si on ne met pas `apply-templates` le processeur est « éteint », il abandonne tout ce qui est en dessous

# Que fait le processeur ?

- Après avoir construit une représentation du document, le processeur rentre dans l'arborescence.
- Il cherche dans la feuille de style un template qui correspond à sa première position, qui est "/"
- S'il n'en trouve pas, il passe aux descendants et recommence.
- S'il trouve un nœud texte, il imprime son contenu.

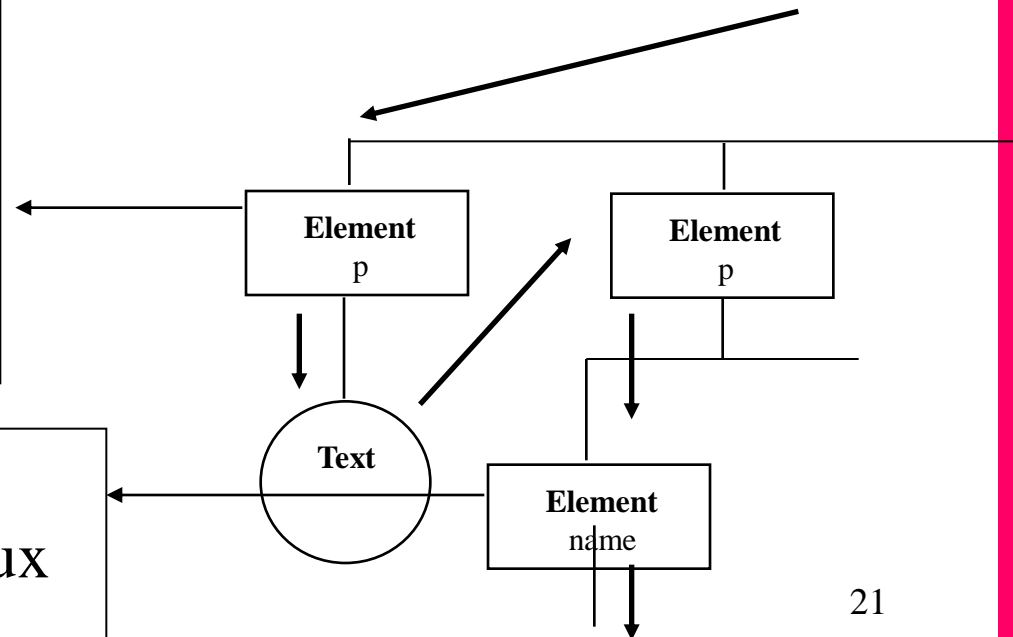
Le processeur parcourt les éléments dans un ordre prévisible (les attributs d'un élément dans un ordre non prévisible)



- Tant que parmi les descendants le processeur ne trouve pas de templates à appliquer, il continue à descendre dans l'ordre enfant / frère, en imprimant les nœuds texte.

Si un template avec match="p" est défini, le processeur l'applique, sinon il descend au nœud texte, l'imprime, et remonte au p suivant

Idem : s'il n'existe pas de template pour name, passe aux nœuds enfants

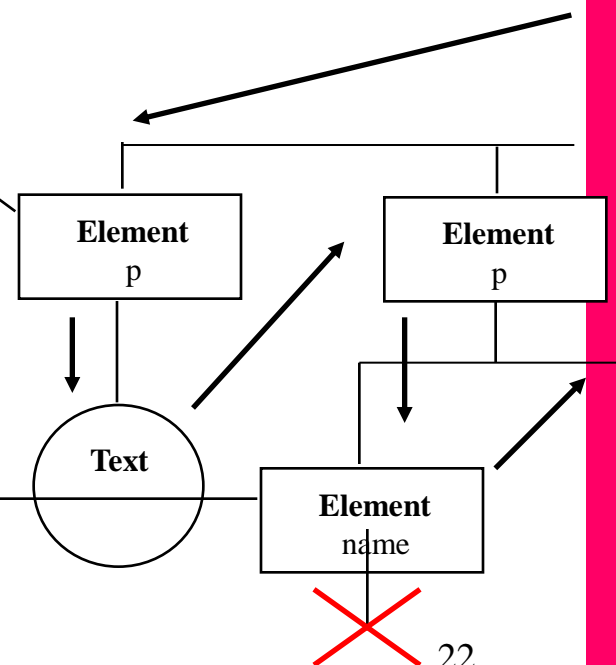


- Si un template correspond, le processeur l'exécute, et ne passe aux descendants que lorsqu'il est explicitement relancé avec l'instruction `xsl:apply-templates`

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

Ce template est exécuté à chaque p rencontré. Une balise p est ouverte, l'analyse reprend dans la sous-arborescence (d'autres templates peuvent être exécutés), puis le p est fermé

```
<xsl:template match="name">  
  M. ***  
</xsl:template>
```



Toutes les balises name et leur contenu sont remplacés par « M. \*\*\* ».

# L'enchâssement

- Dans un template, l'instruction `xsl:apply-templates` a un effet spécial : elle relance la recherche de templates parmi les enfants.
  - Quand le processeur a fini de traiter les descendants, il reprend et termine l'exécution du template d'où il est parti du fait de l'instruction `apply-templates`
- Sinon, il ne rentre pas dans la sous-arborescence.
- Avec les `xsl:apply-template` les templates sont donc "enchâssés" autant de fois que nécessaire pendant l'exécution.
- `xsl:template` permet de traiter les éléments inclus, contrairement à `value-of`

# L'enchâssement (2)

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

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

  <xsl:template match="p">
    <item><xsl:value-of select="." /> </item>
  </xsl:template>

</xsl:stylesheet>
```

1. Imprimé en premier

2. Passe à la descendance, alors que le template n'est pas fini. Imprime le texte contenu dans les éléments sauf p

3. Tous les p sont traités

4. Quand la descendance est traitée "remonte" de templates appelant en templates appelant et fini chacun. Cette ligne est donc imprimée en tout dernier



# Utilisation de xsl:apply-templates

- On peut mettre plusieurs xsl:apply-templates dans le même template.

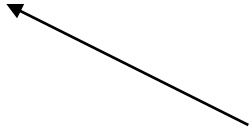
```
<xsl:template match="/">
  <html><body>
    <h1>1. les ref</h1>
    <xsl:apply-templates
select="//xptr">
    <h1>2. les id</h1>
    <xsl:apply-templates
select="//xptr/@id">
  </body></html>
</xsl:template>
```

# apply-templates : utiliser @select

@select permet de préciser au processeur où reprendre la recherche de templates

```
<xsl:template match="/">
  <sous-corpus>
    <xsl:apply-templates select='teiCorpus.2//text/body//p' />
  </sous-corpus>
</xsl:template>

<xsl:template match="p">
  <item><xsl:value-of select="." /> </item>
</xsl:template>
```

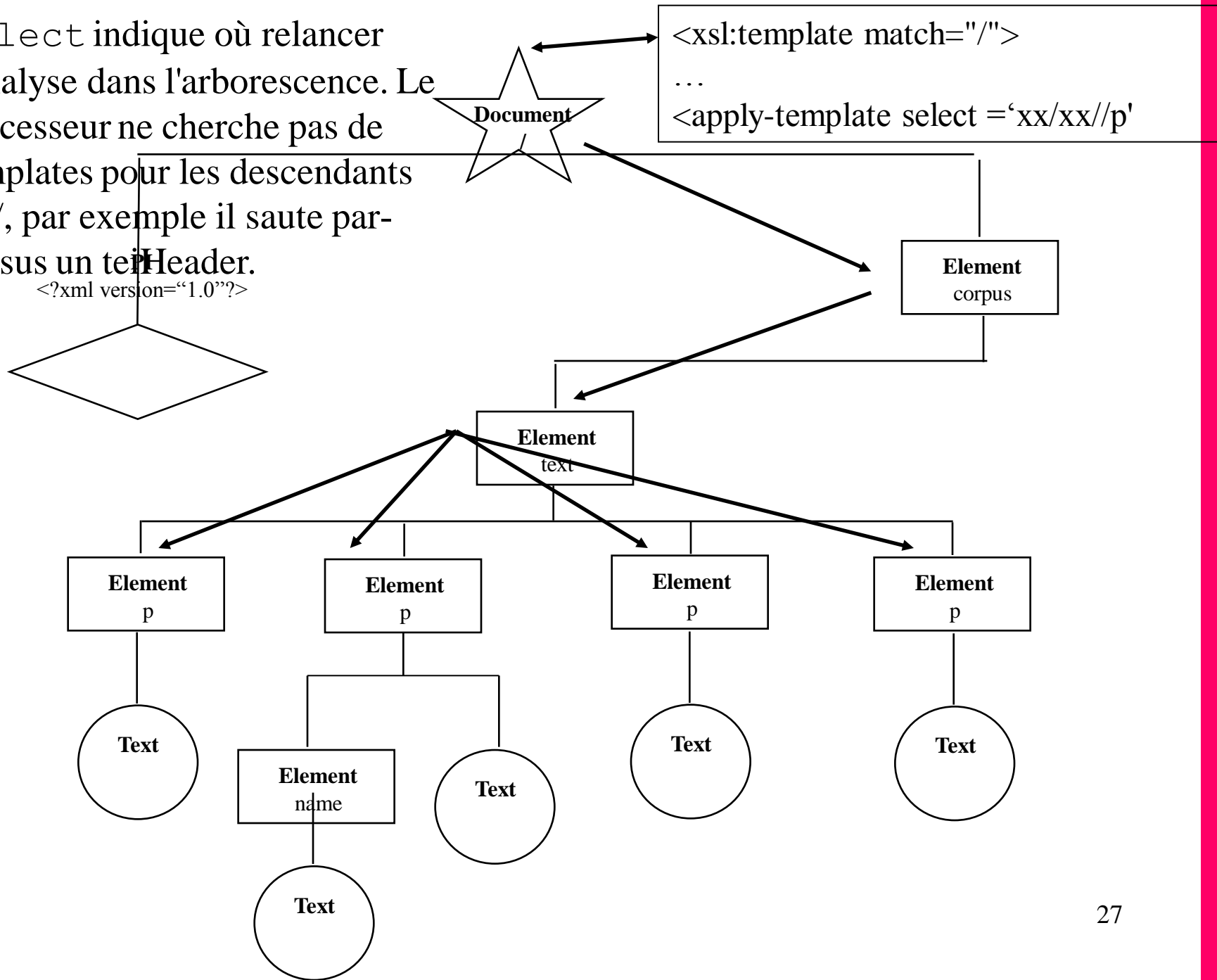


Ici on évite de recopier les header, front et back, en passant directement aux p

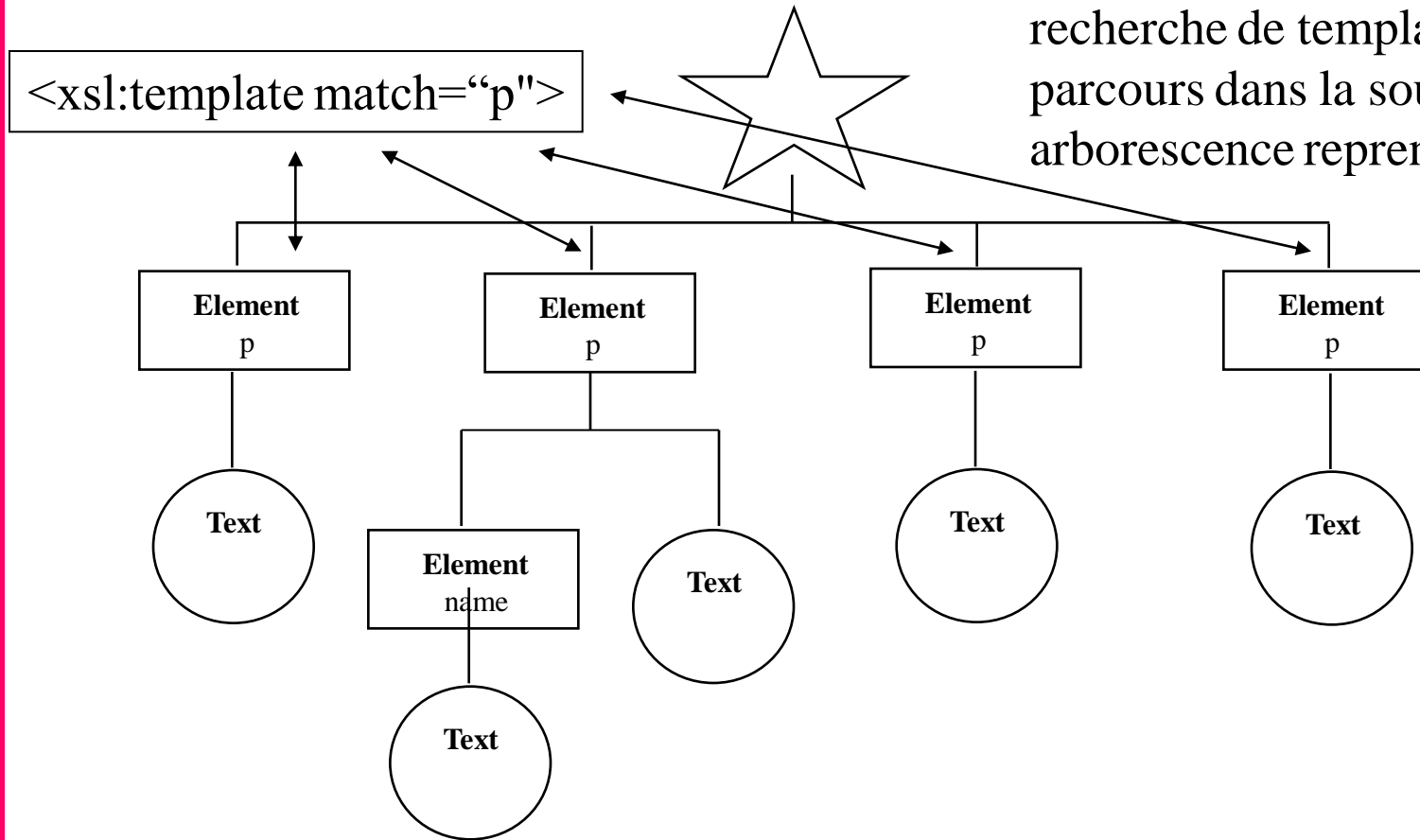
**Voir les deux schémas suivants...**

select indique où relancer l'analyse dans l'arborescence. Le processeur ne cherche pas de templates pour les descendants de /, par exemple il saute par-dessus un `Header`.

`<?xml version="1.0"?>`



Pour chacun des fragments désignés par `select`, la recherche de templates et le parcours dans la sous-arborescence reprend



# Suite

- `@select` permet ici de descendre directement plus bas dans la sous-arborescence en "sautant" toute une partie.
- On peut aussi désigner un endroit qui n'est pas dans la sous arborescence, voire relancer l'analyse à la racine avec un `select="/"` qui fait boucler indéfiniment.
- Note : par défaut, un `xsl:apply-templates` contient donc un `@select="node()"` qui le fait sélectionner ses enfants

# Supprimer les balises

- Une feuille de style contenant ce seul template recopie le document source en enlevant toutes les balises :

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

- Par défaut, dans tous les nœuds où il ne trouve pas de `xsl:template` qui s'applique, le processeur imprime le contenu pour les noeuds de type texte et passe aux enfants pour les autres types de nœuds.

# select : exemple d'utilisation

- On veut recopier le texte du corpus sans balises, et sauter les `header` du corpus et des textes.
- Dans ce cas de figure, la majorité du corpus est recopié, et c'est seulement un tag dont le contenu est à exclure. La méthode la plus économique consiste donc à laisser le processeur recopier par défaut, et indiquer le tag à exclure.

# Méthode 1

- `select` permet de paramétrer le chemin du processeur pour exclure les headers

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

  <xsl:template match="teiCorpus.2">
    <!-- on saute le header du corpus en passant directement aux TEI.2 -->
    <xsl:apply-templates select="//TEI.2"/>
  </xsl:template>

  <xsl:template match="TEI.2">
    <!-- dès qu'on arrive à un TEI.2, on fait sauter le header des TEI.2
    en passant au texte -->
    <xsl:apply-templates select='text'/>
  </xsl:template>

</xsl:stylesheet>
```



# Méthode 2

- Plutôt que d'indiquer au processeur des chemins à parcourir qui excluent les headers, on peut modifier son action par défaut (recopier le texte) sur les nœuds à exclure :

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

<xsl:template match="teiHeader">
    <!-- le template ne contient rien : ni action, ni relance du
    processeur avec apply-templates. Le contenu est donc abandonné -->
</xsl:template>

</xsl:stylesheet>
```

# Valeur d'un nœud selon son type

- Il s'agit d'un *élément*
  - Concaténation de tous les textes qui se trouvent comme contenu de cet élément et de ses descendants
- Il s'agit d'un nœud *text*
  - Texte du nœud lui même
- *Attribut*
  - Valeur de l'attribut normalisée (pas d'espace de début et fin)
- *Instruction de traitement*
  - Valeur de l'instruction de traitement  
(sans les marques <? et ?> et sans le nom)
- *Commentaire*
  - Le texte du commentaire (sans les marques <!-- et -->)

# Exemple 1

- Arbre en entrée

```
<carteDeVisite>  
  <nom>Bekkers</nom>  
</carteDeVisite>
```

- Template

```
<xsl:template match="carteDeVisite">  
  <p>Nom : <xsl:value-of select="nom" /></p>  
</xsl:template>
```

- Arbre en sortie

```
<p>nom : Bekkers</p>
```

# Exemple 2

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Template

```
<xsl:template match="note">  
  <xsl:value-of select="."/>  
</xsl:template>
```

- En sortie

```
enseigne XML au SEP
```

# Exemple 3

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Template

```
<xsl:template match="note">  
  <xsl:value-of select="text()"/>  
</xsl:template>
```

- En sortie

```
enseigne
```

Seul le premier élément sélectionné est produit

# Exemple 4

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Template

```
<xsl:template match="*">  
  <xsl:value-of select="name()"/>  
</xsl:template>
```

- En sortie

```
note
```

# Exemple 5

- Arbre en entrée

4 cartes de visite : Bekkers, Bartold, Letertre, Apolon

- Template

```
<xsl:template match="/carnetDAdresse">  
  <xsl:value-of select="carteDeVisite/nom"/>  
</xsl:template>
```

- En sortie

Bekkers

Seul le premier élément sélectionné est produit

# Exemple 6

- Arbre en entrée

4 cartes de visite : Bekkers, Bartold, Letertre, Apolon

- Template

```
<xsl:template  
  match="/carnetDAdresse/carteDeVisite">  
    <xsl:value-of select="nom"/>  
</xsl:template>
```

- En sortie

BekkersBartoldLetertreApolon

Pour chaque carte de visite le template est appliqué



# Règles par défaut (1)

Traverser la racine et tous les noeuds « élément »

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

Sortir les feuilles « texte » et les « attributs »

```
<xsl:template match="text()|@">  
  <xsl:value-of select="."/>  
</xsl:template>
```

# Règles par défaut (2)

- Commentaires et instructions de traitement

```
<xsl:template match="processing-  
instruction() | comment()" />
```

- Ne rien faire

# Feuille de style minimum

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

  <xsl:output method="html"/>

</xsl:stylesheet>
```

- Traverse tout l'arbre et sort les feuilles (contenu d'élément texte et valeur d'attribut)

# Les attributs

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">  
  <b id="{@href}"/>  
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```

# Les attributs - autre moyen

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">  
  <b><xsl:attribut name="id">  
    <xsl:value-of select="@href"/>  
  </xsl:attribut></b>  
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```

# Élément <xsl:for-each>

- Itération sur un ensemble de nœuds

```
<xsl:template match="/carnetDAdresse">  
  <xsl:for-each select="carteDeVisite">  
    <br><xsl:value-of select="nom"/></br>  
  </xsl:for-each>  
</xsl:template>
```

# Deux styles de programmation

- Récursive

```
<xsl:apply-templates>
```

- Itérative

```
<xsl:for-each>
```

- Attribut `select` donne l'ensemble de nœuds vers lequel on se déplace

# Élément <xsl:comment>

- Sortir les commentaires à l'identique

```
<xsl:template match="comment()">
  <xsl:comment>
    <xsl:value-of select="."/>
  </xsl:comment>
</xsl:template>
```



# Élément <xsl:processing-instruction>

- Sortir les instructions de traitement à l'identique

```
<xsl:template match="processing-  
instruction()">  
  <xsl:processing-instruction name="."/name()>  
    <xsl:value-of select="."/>  
  </xsl:processing-instruction>  
</xsl:template>
```

# Conflits de Règles

- Règle implicite de priorité
  - La règle la plus sélective gagne
  - Parmi 2 templates de même sélectivité, le dernier dans la feuille de style gagne
- Exemple
  - `nom` est plus sélectif que `/` `|` `*`
  - `note[clé]` est plus sélectif que `note`
  - `ville[@codepostal='35000']` est plus sélectif que `ville[@codepostal]`

# Les modes

- Permet de déclarer plusieurs règles pour un même élément
- Chaque règle traite l'élément différemment

```
<xsl:template match="h1" mode="normal">
```

```
<xsl:template match="h1" mode="table-index">
```

# Attributs mode

- Dans un élément `apply-templates`

```
<xsl:apply-templates mode="passer1"/>
```

- Dans un élément `template`

```
<xsl:template match="carteDeVisite"  
  mode="passer1">  
  ...  
</xsl:template>
```

- **Attention** un `apply-templates` n'hérite pas du mode du `template` englobant

# Modularité

# Modularité des documents sources

- Document composé de plusieurs documents

```
<livre>
  <chapitre>chap1.xml</chapitre>
  <chapitre>chap2.xml</chapitre>
  <chapitre>chap3.xml</chapitre>
</livre>
```

- Utiliser la fonction document ()

```
<xsl:template match="chapitre">
  <xsl:apply-templates
    select="document (. )/*" />
</xsl:template>
```

# Élément <xsl:import>

- Modularité des feuilles de style

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0"
    encoding="ISO-8859-1" indent="yes"/>
  <xsl:import href="feuille1.xsl"/>
  ...
</xsl:stylesheet>
```

# Elément <xsl:document>

- Modularité des documents en sortie

```
<xsl:template match="preface">  
  <xsl:document href="{ $dir }\preface.html">  
    <html><body>  
      <xsl:apply-templates/>  
    </body></html>  
  </xsl:document>  
  <a href="{ $dir }\preface.html">Preface</a>  
</xsl:template>
```

- Version 1.1 de XSLT
  - Seul Saxon l'implémente actuellement



# Autres services

- Variables
- Caractères espaces, TAB, CR, LF en sortie
- Tri
- Conditionnelle
- Appel récursif et passage de paramètres
- Template nommés (procédures)

# <xsl:variable>

- Déclaration de variable 1

```
<xsl:variable name="bgcolor"  
  select="'#FFFFCC'" />
```

- Déclaration de variable 2

```
<xsl:variable  
  name="bgcolor">#FFFFCC</xsl:variable>
```

- Référence à une variable

```
<BODY BGCOLOR='{ $bgcolor } '>
```

## <xsl:variable>

- XSL est un langage à assignation unique
- Les « variables » sont des constantes à la manière des constantes `#define` de C
- Une variable ne peut être réaffectée
- La visibilité d'une variable est son élément père
- Une variable peut en cacher une autre

# Les espaces

- Les espaces non significatifs dans l'arbre xsl ne sont pas produits

```
<xsl:template match="nom">
  <p><xsl:value-of select="." />
</p>
</xsl:template>
```

et

```
<xsl:template match="nom">
  <p><xsl:value-of select="." /></p>
</xsl:template>
```

ont le même effet

# Élément <xsl:text>

- Permet de sortir des espaces, des tabulations ou des fins de ligne dans le document de sortie

```
<xsl:text> </xsl:text>
```

# Attribut

## disable-out-escaping

- Pour sortir des caractères spéciaux tels quel (sans être sous forme d'entité)
- Valeurs possible : `yes` ou `no` (par défaut)
- où
  - Dans un élément `xsl:text`
  - Dans un élément `xsl:value-of`
- Attention : cela peut produire des documents qui ne sont pas bien formés
- Utiles pour produire des pages ASP ou JSP

# Élément <xsl:sort>

- Permet de trier l'ensemble des nœuds sélectionnés par les instructions avant de les traiter

```
<xsl:apply-templates>  
<xsl:for-each>
```

- Exemple : trier les cartes de visite par noms

```
<xsl:for-each  
  select="carnetDAdresse/carteDeVisite">  
  <xsl:sort select="nom"/>  
  ...  
</xsl:for-each>
```

# Tri sur plusieurs critères

- Trier d'abord par noms puis par prénoms

```
<sx1:templates match="carnetDAdresse">
  <xsl:apply-templates
    select="carteDeVisite">
    <xsl:sort select="nom"/>
    <xsl:sort select="prénom"/>
  </xsl:apply-templates>
</xsl:template>
```



# Élément <xsl:if>

- Conditionnelle

```
<xsl:for-each select="carteDeVisite">  
  <xsl:value-of select="nom"/>  
  <xsl:if test="position() != last()">,  
</xsl:if>  
</xsl:for-each>
```

- Génère une virgule après chaque nom sauf pour le dernier
- En sortie

Bekkers, Bartold, Letertre, Apolon

# Élément <xsl:choose>

- Conditionnelle à choix multiple

```
<xsl:choose>
  <xsl:when test="start-with('35',@codep)">
    <!-- cas 1 -->
  </xsl:when>
  <xsl:when test="start-with('44',@codep)">
    <!-- cas 2 -->
  </xsl:when>
  <xsl:otherwise>
    <!-- autres cas -->
  </xsl:otherwise>
</xsl:choose>
```

# Initialisation conditionnelle

- Exemple

## Java

```
if (niveau > 20)
    code = 3;
else
    code = 5;
```

## XSLT

```
<xsl:variable name="code">
  <xsl:choose>
    <xsl:when test="$niveau gt; 20">
      <xsl:text>3</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>5</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

# Passage de paramètres

- Déclaration (*paramètre formel*) *Valeur par défaut*

```
<xsl:template match="...">  
  <xsl:param name="p" select="0"/>  
  ... utilisation de p ...  
</xsl:template>
```

- Obtenir la valeur d'un paramètre

```
select="$p"
```

- Appel (*paramètre effectif*) *Affectation de valeur*

```
<xsl:apply-templates>  
  <xsl:with-param name="p" select="$p+1"/>  
</xsl:apply-templates>
```

# Templates només : attributs name

- Les procédures
- Déclaration

```
<xsl:template name="...">  
    ...  
</xsl:template>
```

- Appel

```
<xsl:call-template name="..." />
```

# Exemple (1)

- Exemple : Mettre sous forme d'arbre

<a>

<h1>titre 1</h1>

<par>bla bla 1</par>

<par>bla bla 2</par>

<par>bla bla 3</par>

<h1>titre 2</h1>

<par>bla bla 4</par>

<par>bla bla 5</par>

<par>bla bla 6</par>

</a>

# Exemple (2)

- Resultat attendu

```
<a>
  <h1>
    <titre>titre 1</titre>
    <p>bla bla 1</p>
    <p>bla bla 2</p>
    <p>bla bla 3</p>
  </h1>
  <h1>
    <titre>titre 2</titre>
    <p>bla bla 4</p>
    <p>bla bla 5</p>
    <p>bla bla 6</p>
  </h1>
</a>
```

# Exemple (3)

- Itération sur tous les éléments `<h1>`

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

```
<a>
```

```
<xsl:for-each select="//h1">
```

```
<h1>
```

```
<titre><xsl:value-of select="."/></titre>
```

```
<xsl:call-template name="frere">
```

```
<xsl:with-param name="nds" select="following-sibling::*"/>
```

```
</xsl:call-template>
```

```
</h1>
```

```
</xsl:for-each>
```

```
</a>
```

```
</xsl:template>
```

*Sélection des frères  
qui suivent*





# Exemple (4)

```
<xsl:template name="frere">  
  <xsl:param name="nds"/>  
  <xsl:choose>
```

*Traitement du  
premier frère si  
c'est un élément par*

```
    <xsl:when test="$nds[position()=1 and name()='par']">
```

```
      <p><xsl:value-of select="$nds[1]"/></p>
```

```
      <xsl:call-template name="frere">
```

```
        <xsl:with-param name="nds"
```

```
          select="$nds[position() != 1]"/>
```

*Appel récursif  
sur les autres frères  
après traitement  
du premier*

```
      </xsl:call-template>
```

```
    </xsl:when>
```

```
    <xsl:otherwise/>
```

```
  </xsl:choose>
```

```
</xsl:template>
```

*Arrêt de la récursivité  
si le premier frère est un h1*

# Paramètre au sein d'un élément

## <xsl:stylesheet>

- Les paramètres descendants directs d'un élément `<xsl:stylesheet>` sont autorisés

```
<xsl:stylesheet>  
  <xsl:param name="dir" select="'mondir'"/>  
  ...  
</xsl:stylesheet>
```

- On peut passer une valeur dans la ligne de commande

```
java ... dir=monDir
```

# Mise au point

- Élément `<xsl:message>`

```
<xsl:message>
```

```
    code = <xsl:value-of select="$code"/>
```

```
</xsl:message>
```

- Trace en sortie dans la fenêtre de commande

```
code = 25
```

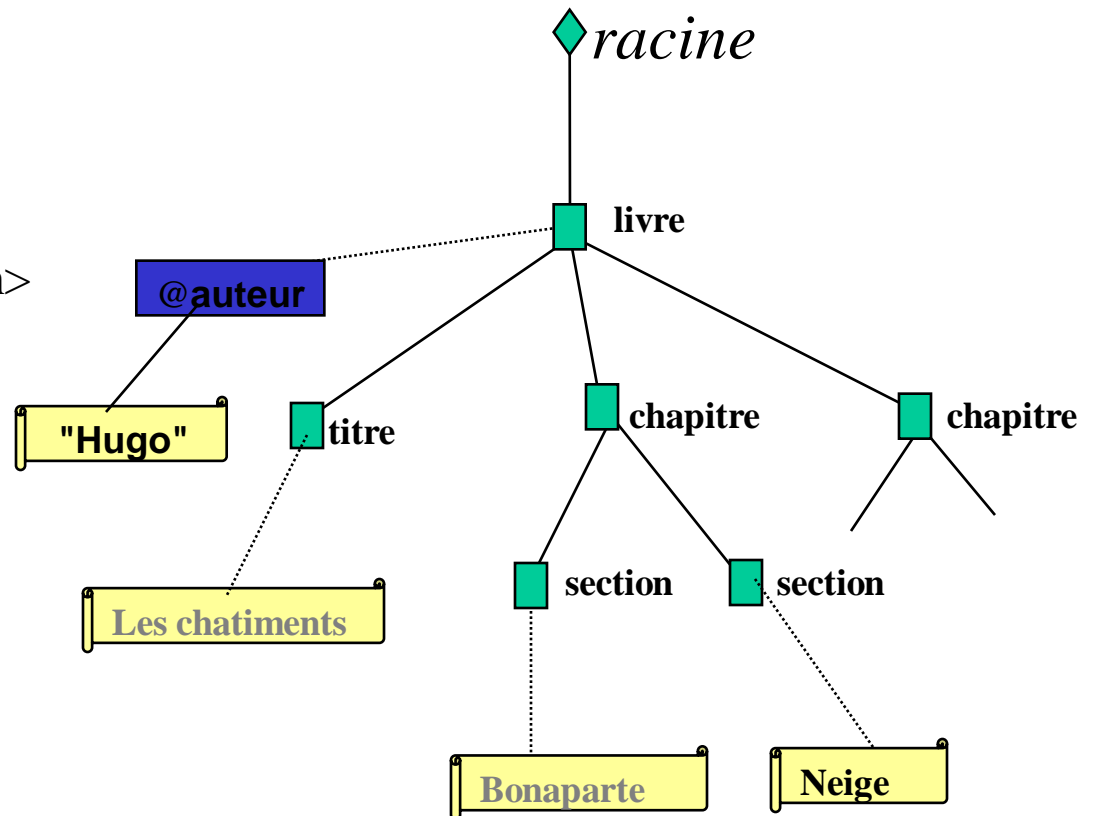
# XPath

- XML Path Language
  - recommandation W3C pour expressions de chemins
  - acceptée le 16 novembre 1999
  - version 2 en cours d'élaboration
- Expressions de chemins communes à :
  - XSL
  - Xpointer (liens)
  - XQuery (queries)
- Xpath permet
  - de rechercher un élément dans un document
  - d'adresser toute sous partie d'un document

# XPath - Parcours d'arbre

- XPath opère sur l'arbre d'un document

```
<livre auteur = "Hugo">  
  <titre>Les chatiments</titre>  
  <chapitre>  
    <section>Buonaparte </section>  
    <section>Neige</section>  
  </chapitre>  
  ...  
</livre>
```



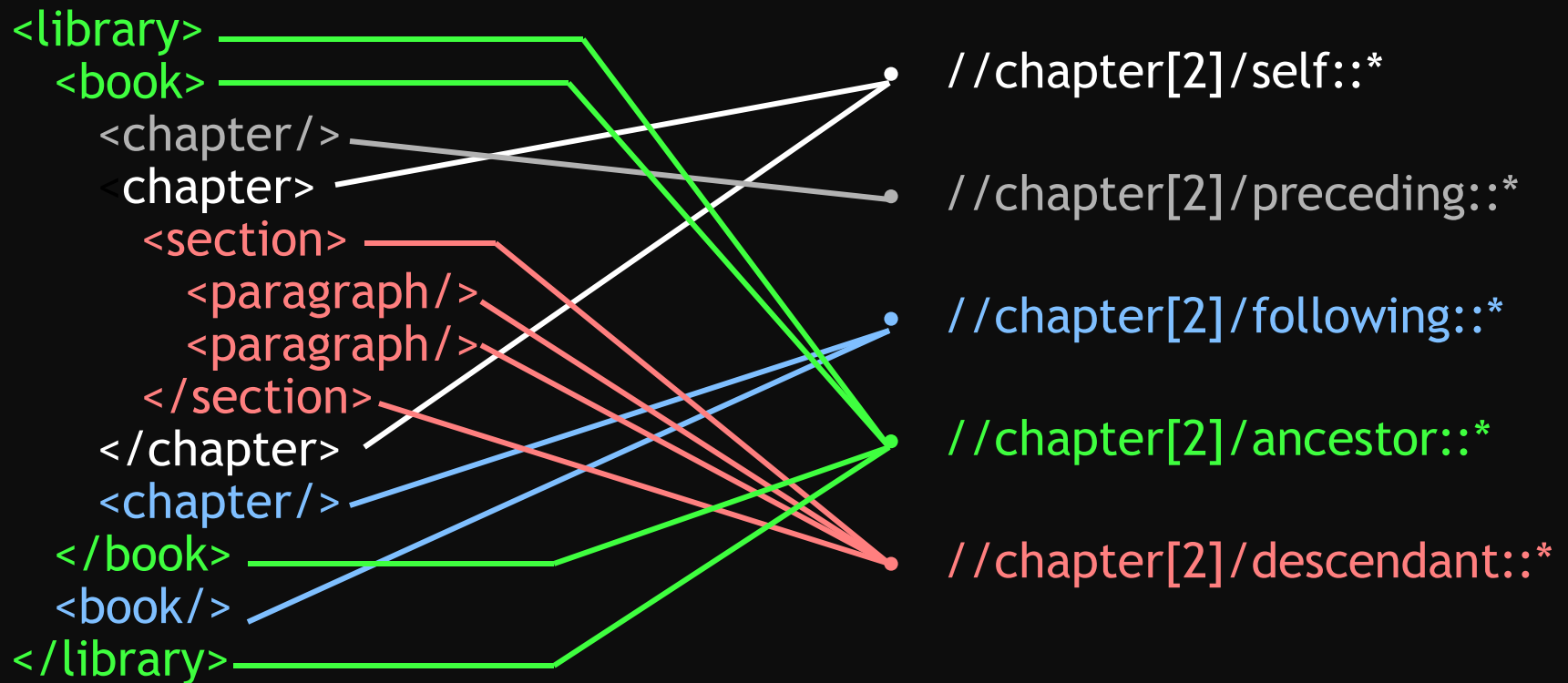
# XPath - Expression de chemins

- Une expression de chemins spécifie une traversée de l'arbre du document :
  - depuis un nœud de départ
  - vers un ensemble de nœuds cibles
  - les cibles constituent la valeur du cheminement
- Un chemin peut être :
  - absolu
    - commence à la racine
    - /étape1/.../étapeN
  - relatif
    - commence à un nœud courant
    - étape1/.../étapeN

# Syntaxe et sémantique

- Cheminement élémentaire
  - direction::sélecteur [predicat]
- Directions
  - parent, ancestor, ancestor-or-self
  - child, descendant, descendant-or-self
  - preceding, preceding-sibling, following, following-sibling
  - self, attribute, namespace
- Sélecteur
  - nom de nœud sélectionné (élément ou @attribut)
- Prédicat
  - [Fonction(nœud) = valeur]

# Les axes





# XPath - Exemples

- Sections d'un chapitre
  - /child::livre/child::chapitre/child::section
  - /livre/chapitre/section
- Texte du chapitre 1 section 2
  - /descendant::chapitre[position() = 1]  
  /child::section[position() = 2]/child::text()
  - //chapitre[1]/section[2]/text()

# XPath - Synthèse

Pattern	Exemple	Signification
Nom	<b>section</b>	Sélectionne les éléments de nom donné
Nom[0]	<b>section[0]</b>	Sélectionne le premier élément ayant le nom donné
Nom[end())]	<b>section[end())]</b>	Sélectionne le dernier élément ayant un nom donné
	<b>Droite Gauche</b>	Indique une alternative (un nœud OU bien l'autre (ou les deux))
/	<b>/</b>	Sélectionne le nœud racine d'une arborescence
/arbre/Nom	<b>/livre/chapitre</b>	Sélectionne les nœuds descendants par la balise de nom donné de l'arbre
*	<b>*</b>	Motif "joker" désignant n'importe quel élément
//	<b>//personne</b>	Indique tous les descendants d'un nœud
.	<b>.</b>	Caractérise le nœud courant
..	<b>..</b>	Désigne le nœud parent. Permet de remonter d'un niveau dans l'arborescence
@	<b>@nom</b>	Indique un attribut caractéristique ( <i>@nom</i> désigne la valeur de l'attribut). La notation <i>@ *</i> désigne tous les attributs d'un élément
text()	<b>text()</b>	Désigne le contenu d'un élément (le texte contenu entre ses balises)
ID()	<b>ID('a2546')</b>	Sélectionne l'élément dont l'identifiant (la valeur de l'attribut ID) est celui spécifié en paramètre
Comment()	<b>Comment()</b>	Désigne tous les nœuds commentaires
Node()	<b>Node()</b>	Désigne tous les noeuds

# Conclusion

- Oui
  - XSLT est un vrai langage de programmation
  - XSLT n'a pas son équivalent pour la transformation d'arbre
- Mais
  - La mise au point de programmes XSLT peut s'avérer « délicate »
  - La maintenabilité est discutable

# XSL-FO : le formatage

- Permet les mises en pages sophistiquées
- Objets de mise en forme applicables aux résultats avec XSLT
- Distinction
  - Formatage des pages
  - Formatage des objets à l'intérieur des pages
    - Statiques
    - Dynamiques

**CEV<sup>®</sup> (Customer Economic Value) Comparison**

International Model: 4300 SBA LP 4X2 Competitor Model: Freightliner FL700

Application: Dry Van - 5 years/25,000miles/yr  
(Applies to other applications with similar mileage)

**Expected Differential Savings for International Trucks and Tractors versus Competition**

VALUE CATEGORY	ISB 5.9L	ISC 6.3L	MBE 900 4.3L	MBE 900 6.4L
Resale value	\$ 860 / 1,300	\$ 860 / 1,300	\$ 1,650 / 1,300	\$ 960 / 1,300
Engine overhaul cost	Not Available	Not Available	Not Available	Not Available
Preventive maintenance costs	\$ 160 / 250	\$ 280 / 432	\$ (57) / 1,130	\$ 21 / 41
Repairability	Not Available	Not Available	Not Available	Not Available
<b>TOTAL (see note below)</b>	<b>\$ 960 / 1,402</b>	<b>\$ 1,080 / 1,632</b>	<b>\$ 1,590 / 2,430</b>	<b>\$ 981 / 1,341</b>

(Figures in table are not differences in expected dollar value or value of International vs. competitor)

**VALUE NOTES**

**RESALE VALUE** Consistent used vehicle pricing.

**ENGINE OVERHAUL COST** Repair cost savings for engine rebuilds.

**PREVENTIVE MAINTENANCE COSTS** Lower preventive maintenance costs due to longer service intervals.

**REPAIRABILITY** Lower repair costs resulting from replacing individual parts instead of the entire section.

**ADDITIONAL VALUE POINTS**

**VISIBILITY (PRODUCTIVITY)** It's easier to see the gauges on our raised instrument panel. Improved visibility features like this one help drivers keep their eyes on the road.

**VISIBILITY (PRODUCTIVITY)** 140" sq. ft. loading package contains a 620 sq. ft. radiator side-by-side with a 475 sq. ft. charge air cooler. A highly efficient, light-weight, low profile design that enhances forward visibility.

**VISIBILITY (PRODUCTIVITY)** With up to 2074 sq. ft. of front windshield glass area, and 540 sq. ft. of available glass area in each door the driver has a commanding view of the road. The repositioned A-pillar increases overall road view. A swept back angle helps deflect debris, thus minimizing glass damage. Standard blind glass helps reduce glare.

**RESALE VALUE (LIFECYCLE COST)** International's galvanneal steel cabs are constructed with welded-in reinforcements, a deep ribbed back panel, and a single piece steel door frame. It's built to withstand the toughest conditions—keeping you out of the shop and on the job.

**RESALE VALUE (LIFECYCLE COST)** Standard rear cab air suspension minimizes cab vibration, extending cab, and cab-mounted component life and delivering an exceptional ride, resulting in lower driver turnover and associated costs.

**RESALE VALUE (LIFECYCLE COST)** Our redesigned all-aluminum cab results corrosion, and it weighs less. A lighter truck means you can increase your payload and your profits.

**RESALE VALUE (LIFECYCLE COST)** Polished aluminum wheels can increase resale value, improve image, and reduce chassis weight. An A-10 aluminum hub to reduce chassis weight up to 100 pounds at the front axle.

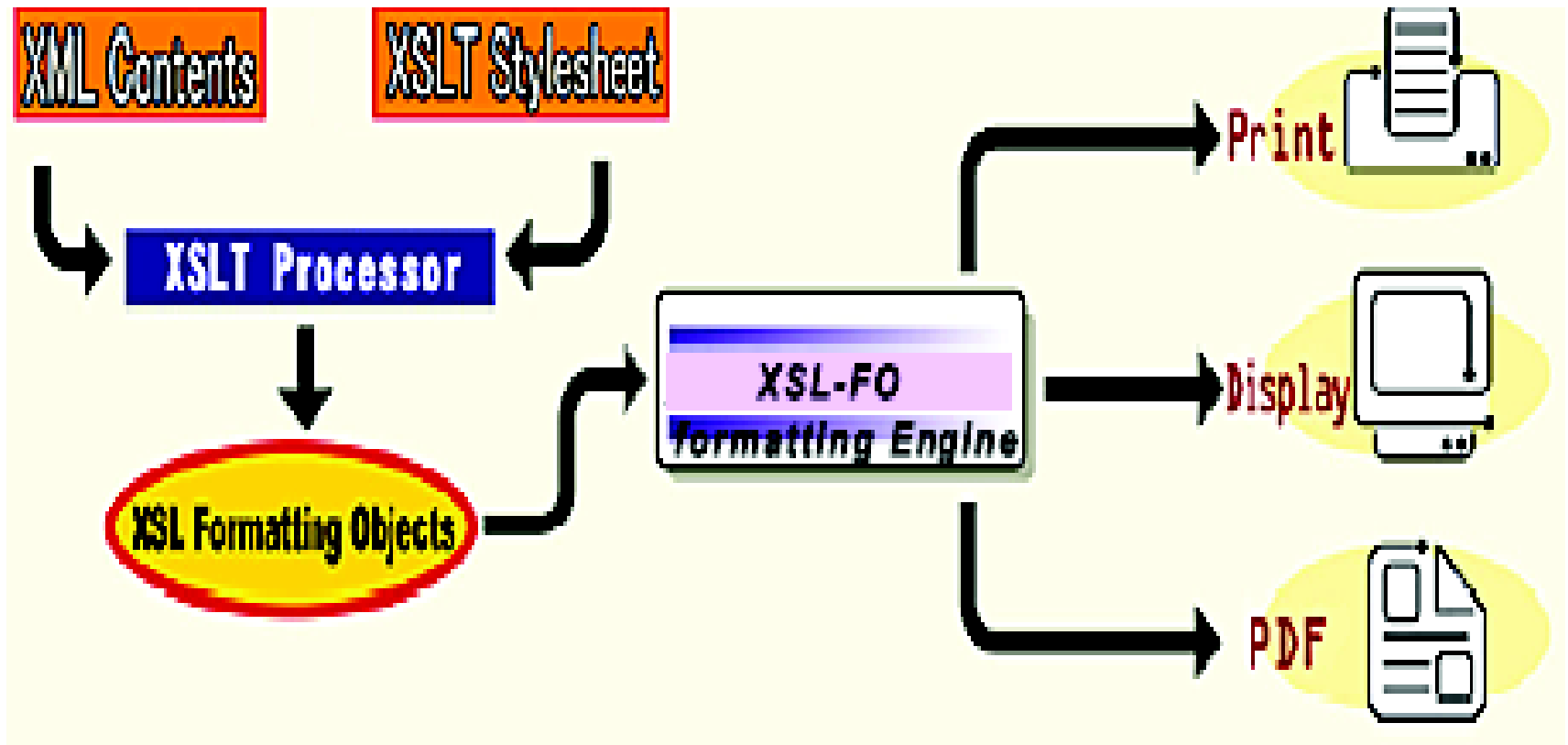
**BODY AND CHASSIS INTEGRATION (OPERATING EFFICIENCY)** Whether you're in the construction business, hauling garbage or plowing snow, the International 1000 Series has been designed to handle the loads CA necessary to mount bodies and equipment with ease.

**BODY AND CHASSIS INTEGRATION (OPERATING EFFICIENCY)** Front frame extensions with integral reinforcements provide the strength required to mount glow valves in combination with front PTO driven pumps and other heavy equipment.

CEV (Customer Economic Value) Comparisons are provided for illustration purposes only. Individual results will vary based on most common specific application, trade cycle and miles per year.

# Principes

- Peut s'appliquer aux résultats des feuilles XSLT



# Organisation du document

- Un document FO est formé d'un élément fo:root qui comprend deux parties distinctes
  - une description des modèles de pages
    - fo:layout-master-set
  - une description du contenu
    - fo:page-sequence
- Le contenu comporte :
  - Des flux contenant les données mêmes du document
  - Des éléments statiques dont le contenu se répète sur les pages (en-têtes courants, no de page, etc.)

# Objets de formatage

- Les objets de formatage sont multiples :
  - `<fo:block>`
    - utilisé pour les blocs de textes, paragraphes, titres...
  - `<fo:display-rule>`
    - ligne de séparation
  - `<fo:external-graphic>`
    - zone rectangulaire contenant un graphisme (SVG)
- Ils possèdent de nombreuses propriétés
  - Pour un block on peut définir
    - la marge gauche et droite
    - l'espace avant et après le paragraphe
    - la couleur du texte .....

# Fonctionnalités

- Pages portrait ou paysage
- Pages recto-verso
- Page de tailles variées
- Marges multiples
- Colonnes multiples
- Entête et pieds de page
- Caractères unicode
- Formatage multi-langages
- Tables des matières générées
- Multiple directions d'écritures
- Numérotation des pages
- Graphiques et SVG
- Tables, avec entêtes, lignes et colonnes fusionnables
- Listes
- Zones flottantes
- Tris à l'édition
- Notes de bas de pages



# XSL-FO: hello World

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root
  xmlns:fo="http://www.w3.org/1999/XSL/
  Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-
      name="my-page">
      <fo:region-body margin="2 cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="my-
    page">
    <fo:flow flow-name="xsl-region-body">
      <fo:block>Hello, world!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

- Element Root
  - Permet de définir le namespace XSL-FO
- Layout master set
  - Permet de déclarer une ou plusieurs page masters (masque) et page sequence masters pour définir la structure des pages (ici une de 2 cm de marges)
- Page sequence
  - Les pages sont groupées en séquences et structurées selon la référence au masque.
- Flow
  - C'est le container du texte utilisateur dans le document. Le nom du flot lit le texte à une zone de la page définie dans le masque.
- Block
  - C'est le bloc de formatage qui inclut un paragraphe de texte pouvant être produit pas XSLT.

# XSL-FO et XSLT : Exemple

- Définition de propriétés pour la racine
  - `<xsl:template match="/">`
    - `<fo-display-sequence`
      - `font-style='italic'`
      - `start-indent='4pt'`
      - `end-indent='4pt'`
      - `font-size='18pt'`
    - `<xsl:apply-templates/>`
    - `</fo-display-sequence>`
    - `</xsl:template`
- La définition d'une propriété locale est prioritaire devant l'héritage

# Les processeurs XSL-FO

- Apache Group : FOP
  - Formating Object Processor
  - Génère du PDF <http://www.apache.org>
- JFOR (Open Source)
  - Génère du RTF <http://www.jfor.org>
- Antenna House
  - XSL Formatter <http://www.antennahouse.com>
- RenderX
  - Génère du PDF <http://www.renderx.com>
- Altova
  - StyleVision [http://www.altova.com/products\\_xsl.html](http://www.altova.com/products_xsl.html)
- XML Mind FO Converter
  - Génère du RTF <http://www.xmlmind.com/foconverter>
- Autres
  - Arbortext, Adobe, Scriptura, XMLPDF, APOC, XSL-FO to TeX