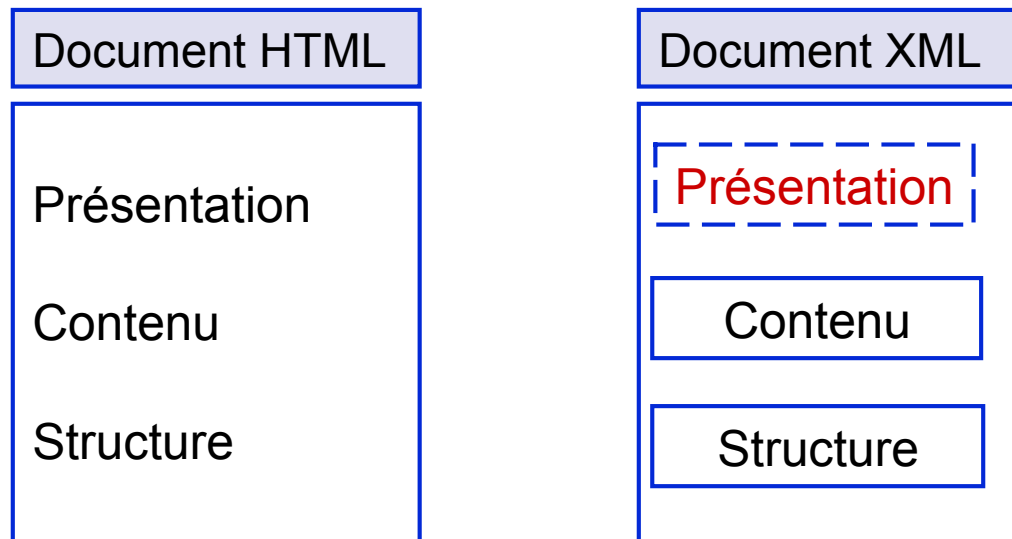




# XML et ses applications : plan

1. Introduction à XML
2. Schémas de documents XML
  - a. DTD
  - b. XML Schema
3. Le modèle DOM
4. Interrogation : XPath
- 5. Transformation et présentation : XSLT**
6. Programmation : XML et Java (API DOM, SAX, ...)
7. Application : intégration de données

# HTML versus XML



# Objectifs du langage XSLT

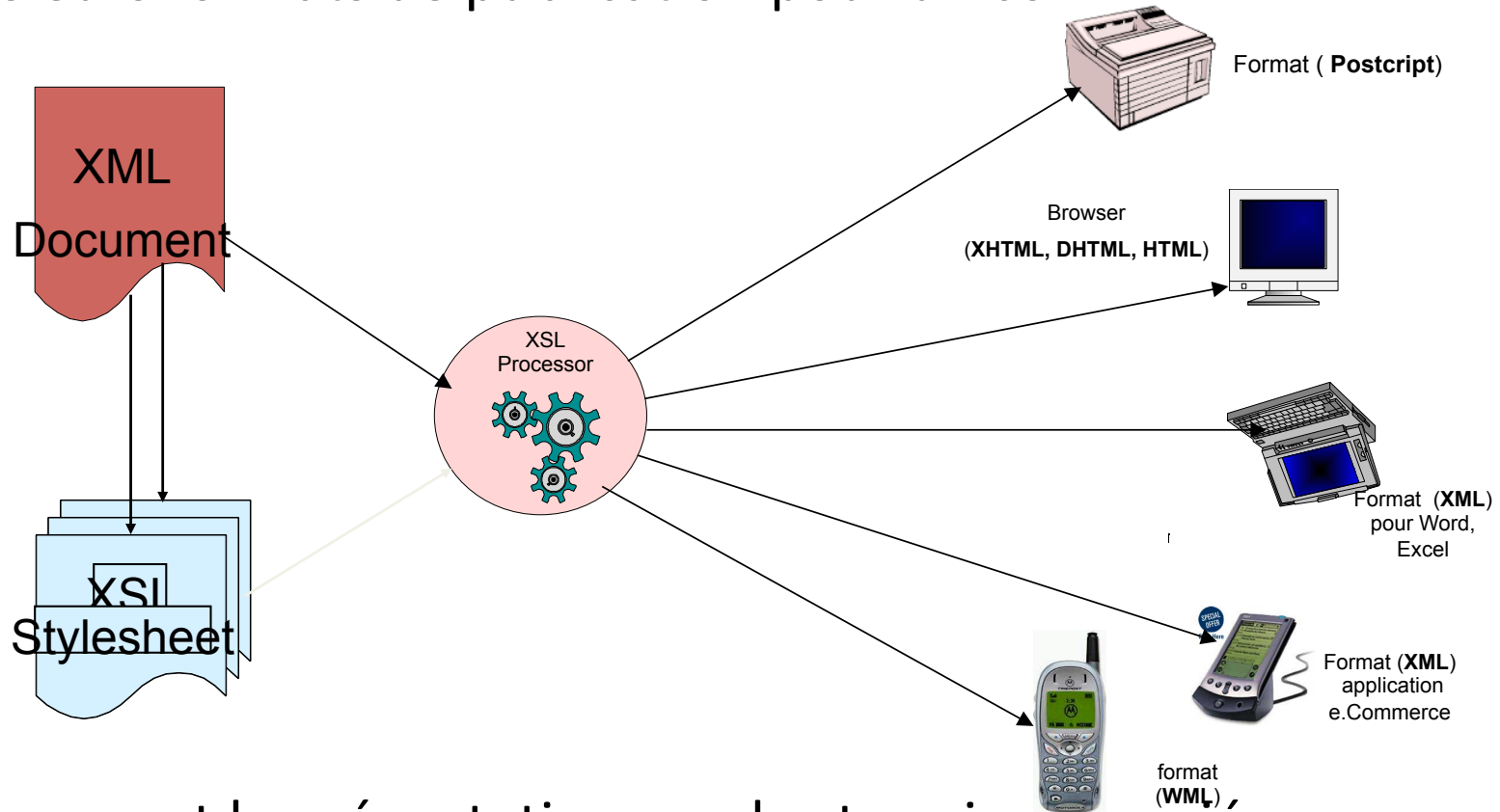
- XSLT permet de produire un nouveau document (XML, HTML, ...) par **transformation** d'un document XML existant.
- Une transformation s'exprime sous la forme d'une **feuille de style** qui est un document XML composé d'un ensemble de **règles** de transformation.
  - **Une règle comporte deux parties :**
    - un modèle de chemin exprimé en termes du langage **XPath**,
    - une transformation sous la forme d'une suite d'**instructions**.

# Objectifs du langage XSLT

- L'espace de noms associé au langage XSLT est :  
<http://www.w3.org/1999/XSL/Transform>
- dont le préfixe usuel est **xsl**.

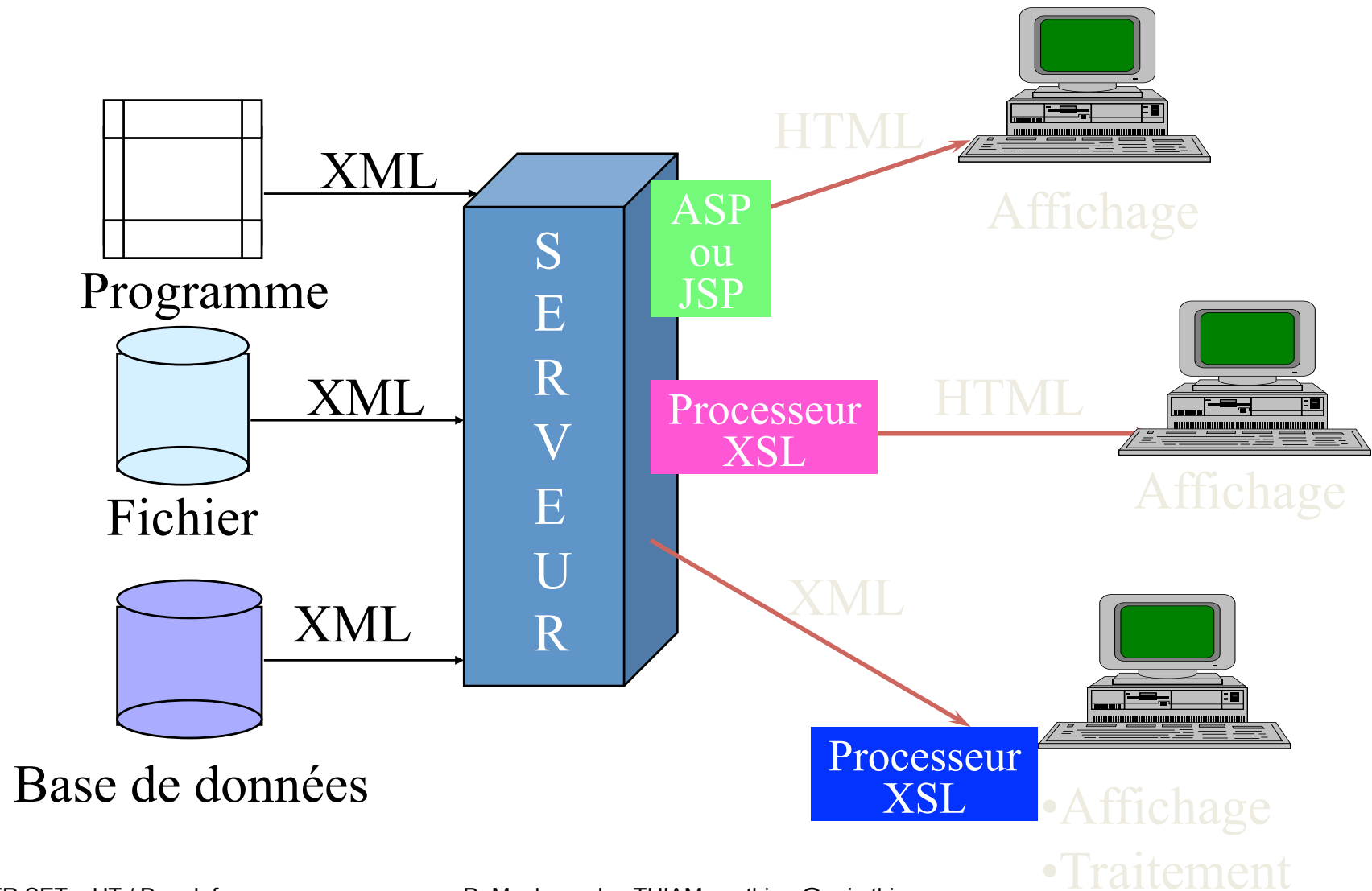
# Publications avec XSL

- Plusieurs formats de publication pour un contenu

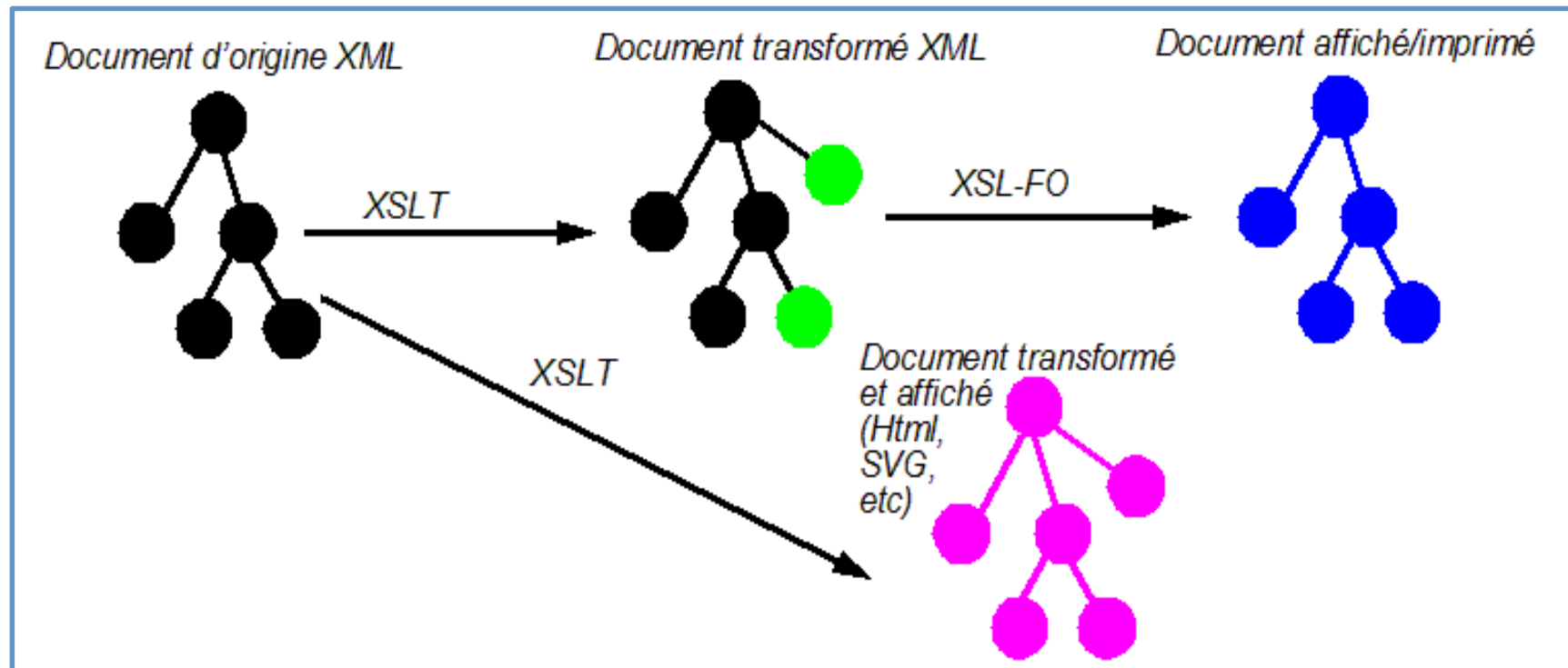


- XSL permet la présentation sur des terminaux variés

# Architectures



# Utilisation de XSLT



# Travaux pratiques

- **Usine à gaz**
  - Utilisez vos IDE
    - NetBeans
    - Eclipse
    - XML Editor
- **SAXON**
  - Installer le processeur
  - En ligne tapez
  - **Transform -s:source -xsl:stylesheet -o:output**



# Le document XSLT

- Un document XSLT est un document XML, on y trouve donc
  - Un prologue
  - Un corps
  - (Un épilogue)
- Le prologue
  - ✓ `<?xml version="1.0" encoding="UTF-8"?>`

# Structure d'un document XSLT

- Le corps

- `<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`  
    *... liste de templates (règles)*  
`</xsl:stylesheet>`

Ou la formulation équivalente

- `<xsl:transform version="1.0"`  
    `xmlns:xsl="http://www.w3.org/1999/XSL/Transform">`  
    *... liste de templates (règles)*  
`</xsl:transform>`

# Règles

- Une règle (***template***) est décrite sous forme d'un élément XML :

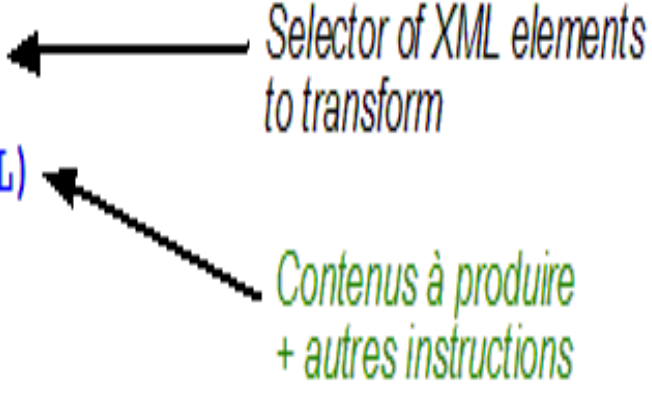
```
<xsl:template match="p">  
    Transformation  
</xsl:template>
```

- où :
  - ***p*** est un chemin de localisation : une requête **XPath**,
  - ***Transformation*** une séquence de caractères ou d'éléments dont certains sont des instructions **XSLT**.

# Règles

- D'autres attributs permettent :
  - de nommer une règle pour l'invoquer par ce nom,
  - de donner une priorité à une règle
  - Plus ce nombre est grand plus la règle a une priorité forte, ...

```
<xsl:template match="XML_tag_name">
  .... contents to produce (i.e. HTML)
  .... further instructions
</xsl:template>
```



*Selector of XML elements to transform*

*Contenus à produire + autres instructions*

# Instruction

- Une instruction est décrite par un élément XML prédéfini dans le langage XSLT. Par exemple :

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

- XSLT possède un jeu d'instructions très complet :
  - application d'une règle,
  - extraction de la valeur textuelle d'un élément,
  - instructions de branchement et de contrôle (IF, Then, ELSE)
  - instructions de tri et de groupement,
  - définitions de variables,
  - ...

# XSLT : du XML vers le XHTML

- On crée un template pour la racine du document

```
<xsl:template match="/">
  <html>
    <head><title>TITRE</title></head>
    <body>
      HTML
      + constructions XSLT
      + des <xsl:template match= "requête XPath">
        ...
      </xsl:template>
    </body>
  </html>
</xsl:template>
```

# Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
  <xsl:template match="/bdtheque/personnes">
    <html>
      <head>
        <title>Liste des personnes de la bdthèque</title>
      </head>
      <body>
        <h2>Les personnes </h2>
        <xsl:apply-templates select="personne" />
      </body>
    </html>
  </xsl:template>
  <xsl:template match="personne">
    <h2> <xsl:value-of select="."></h2>
  </xsl:template>
</xsl:stylesheet>
```

# Moteur de transformation XSLT

- Le moteur de transformation :
  - opère sur :
    - l'arbre du document XML à transformer,
    - une liste de noeuds à traiter,
    - un noeud à traiter : le **noeud contexte** ;
  - produit un **flot de sortie** : en général un document XML ou HTML.
- Lancement du moteur :
  - La liste des noeuds à traiter est constituée d'un noeud unique : le noeud racine du document à traiter.
  - Appliquer les règles.



# Application des règles

- Pour chaque noeud de la liste des noeuds à traiter, appelé **noeud contexte** :
  - Chercher les règles `<xsl:template match="p">t</...>` telles que le noeud contexte est l'extrémité du chemin  $p$ .
  - S'il y en a plusieurs choisir la plus spécifique c.-à-d. celle dont le chemin  $p$  est le plus précis.
    - Par exemple, la règle `<... match="vallon/nom">` est plus spécifique que la règle `<... match="vallon">`.
  - puis la plus prioritaire.
  - Appliquer la transformation  $t$  en parcourant son texte et :
    - en recopiant dans le flot de sortie tout caractère qui n'appartient pas à une instruction XSLT,
    - en exécutant les instructions XSLT.

# Règles prédéfinies (par défaut)

- Les règles prédéfinies s'appliquent en l'absence de règles applicables définies dans la feuille de style. Elles ont une priorité + faible que celles-ci.
- Les deux principales sont :
  - règle prédéfinie **pour les noeuds racine et éléments**, provoquant la relance du traitement sur les noeuds fils du noeud contexte :

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

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

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

- règle prédéfinie **pour les noeuds textes et attributs**, produisant la recopie de leurs valeurs dans le flot de sortie :

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

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

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

# Quelques instructions classiques

- **<xsl:apply-templates/>**
  - La liste des noeuds à traiter est constituée des noeuds fils du noeud contexte.
  - Appliquer les règles.
- **<xsl:apply-templates select="p"/>**
  - La liste des noeuds à traiter est constituée des noeuds atteints par le chemin *p* depuis le noeud contexte.
  - Appliquer les règles.
- **<xsl:value-of select="p"/>**
  - Recopier dans le flot de sortie la valeur-chaîne de chaque noeud atteint par le chemin *p* depuis le noeud contexte.
- **<xsl:copy-of select="p"/>**
  - Recopier dans le flot de sortie le fragment du document à transformer dont la racine est le noeud atteint par le chemin *p* depuis le noeud contexte.
- **<xsl:text>t</xsl:text>**
  - Recopier dans le flot de sortie le texte *t*.

# Encore des instructions XSL

## 1. Les fondamentaux

- a) xsl:stylesheet
- b) xsl:output
- c) xsl:template
- d) xsl:value-of

## 2. Ajout d'éléments et d'attributs

- a) xsl:element
- b) xsl:attribute
- c) Syntaxe courte

# Encore des instructions XSL

## 1. Gestion des boucles

- a) `xsl:for-each`
- b) `xsl:sort`
- c) `xsl:number`

## 2. Conditions de test

- a) `xsl:if`
- b) `xsl:choose`

# Ex 1: Itinéraires les plus difficiles

- On veut produire un document XML listant les itinéraires \*\*\*\* avec pour chacun, son nom et le nom du vallon dans lequel il se déroule :

```
<best-of>
```

```
<nom>Roche Gauthier (Vallon de Granon)</nom>
```

```
<nom>Le Guyon (Vallon des Acles)</nom>
```

```
<nom>Roche des prés (Vallon des Acles)</nom>
```

```
<nom>Crête de Marapa (Vallon des Acles)</nom>
```

```
<nom>Pointe des Rochers-Charniers (Vallon des Acles)</nom>
```

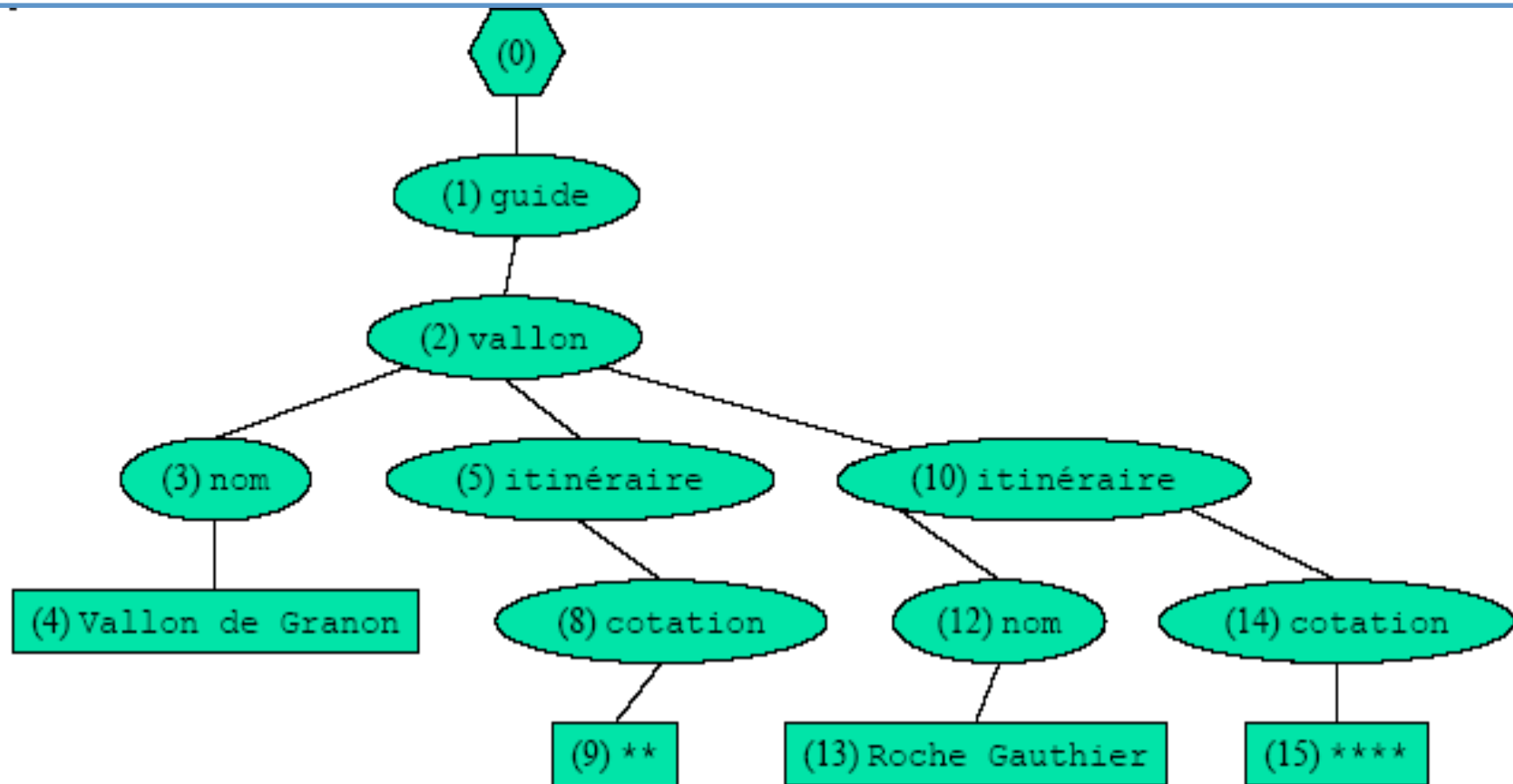
```
<nom>Pic du Lauzin (Vallon des Acles)</nom>
```

```
<nom>Pointe de Pécé (Vallon des Acles)</nom>
```

```
...
```

```
</best-of>
```

# Ex 1 : fragment de l'arbre du document à transformer



# Ex 1 : la feuille de style

```
1. <xsl:stylesheet version="1.0"
2.           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3. <xsl:output method="xml" indent="yes"/>
4. <xsl:template match="/">
5.   <best-of>
6.     <xsl:apply-templates select="//itinéraire[cotation='*****']"/>
7.   </best-of>
8. </xsl:template>
9. <xsl:template match="itinéraire">
10.   <nom>
11.     <xsl:value-of select="nom"/>
12.     <xsl:text> (</xsl:text>
13.     <xsl:value-of select="ancestor::vallon/nom"/>
14.     <xsl:text>)</xsl:text>
15.   </nom>
16. </xsl:template>
17. </xsl:stylesheet>
```



# Ex 1: la transformation (sur l'extrait)

- La règle 4 s'applique au noeud 0 :
  - on écrit **<best-of>** dans le flot de sortie (ligne 5).
  - on applique les règles aux noeuds vérifiant :  
itinéraire[cotation='\*\*\*\*'] soit 5.
- La règle 9 s'applique au noeud 5 :
  - on écrit <nom> dans le flot de sortie (ligne 10),
  - on écrit Roche Gauthier dans le flot de sortie (instruction 11),
  - on écrit ( dans le flot de sortie (ligne 12),
  - on écrit Vallon de Granon dans le flot de sortie (instruction 13),
  - on écrit ) dans le flot de sortie (ligne 14),
  - on écrit </nom> dans le flot de sortie (ligne 15).
- On écrit </best-of> dans le flot de sortie (ligne 7).

## Ex 2 : liste des vallons et des itinéraires

- On veut produire un document XML listant les noms des vallons et de leurs itinéraires

```
<vallons>
```

```
<vallon>
```

```
  <nom>Vallon de Granon</nom>
```

```
  <itinéraires>
```

```
    <nom>Col de Granon</nom>
```

```
    ...
```

```
  </itinéraires>
```

```
</vallon>
```

```
<vallon>
```

```
  <nom>Vallon des Acles</nom> ...
```

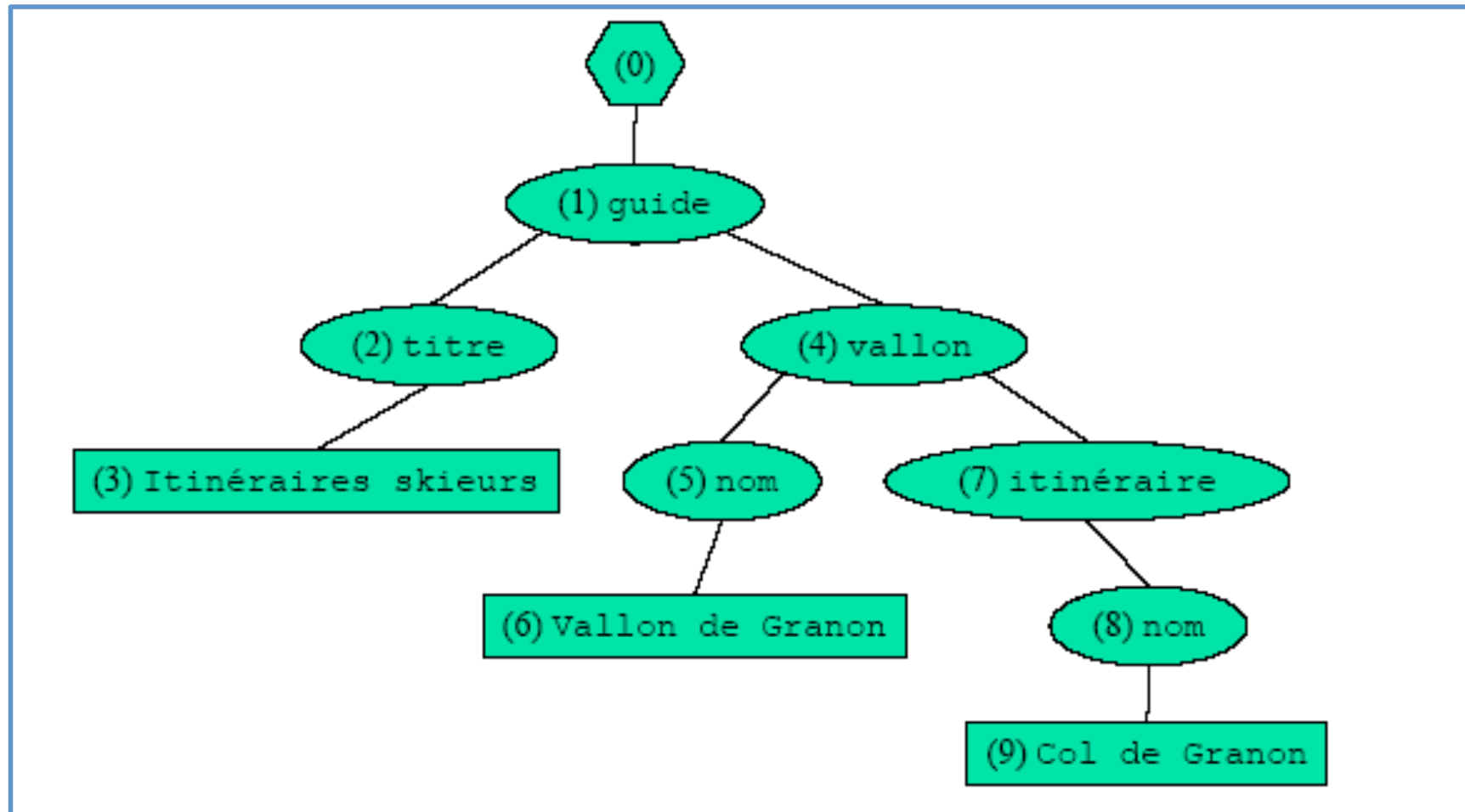
```
</vallon>
```

```
...
```

```
</vallons>
```

- Cet exemple illustre l'emploi des règles prédéfinies.

## Ex 2 : fragment de l'arbre du document à transformer



## Ex 2 : la feuille de style

```
<xsl:stylesheet version="1.0"
2.           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3. <xsl:output method="xml" indent="yes"/>
4. <xsl:template match="/">
5.     <vallons>
6.         <xsl:apply-templates/>
7.     </vallons>
8. </xsl:template>
9. <xsl:template match="vallon">
10.    <xsl:copy-of select="nom"/>
11.    <itinéraires>
12.        <xsl:apply-templates/>
13.    </itinéraires>
14. </xsl:template>
15. <xsl:template match="itineraire/nom">
16.    <xsl:copy-of select="."/>
17. </xsl:template>
18. <xsl:template match="text()|@"*>
19. </xsl:stylesheet>
```

Cette règle cache la règle prédéfinie pour les noeuds texte ou attribut. Elle empêche leur valeur d'être écrite dans le flot de sortie.

## Ex 2 : la transformation (sur l'extrait)

- applique les règles aux noeuds 1 fils du noeud 0.
- La règle prédéfinie sur les noeuds racine et élément s'applique au noeud 1 : il est ignoré puis on applique les règles aux noeuds 2 et 4 fils du noeud 1.
- La règle prédéfinie sur les noeuds racine et élément s'applique au noeud 2 : il est ignoré et l'application des règles est relancée sur le noeud 3 fils du noeud 2.
- La règle 18 s'applique au noeud 3 : aucune action n'est réalisée.
- La règle 9 s'applique au noeud 4 : on écrit <nom>Vallon de Granon</nom> (instruction 10) puis <itinéraires> (ligne 11) dans le flot de sortie, puis on applique les règles aux noeuds 5, 7 et 11 fils du noeud 4.
- La règle prédéfinie sur les noeuds racine et élément s'applique au noeud 5 : il est ignoré et l'application des règles est relancée sur le noeud 6 fils du noeud 5. La règle 18 s'applique au noeud 6 : aucune action n'est réalisée.
- La règle 15 s'applique au noeud 8 : on écrit <nom>Col de Granon</nom> dans le flot de sortie (instruction 16).
- On écrit </itinéraires> dans le flot de sortie (ligne 13).
- On écrit </vallons> dans le flot de sortie (ligne 7).

# Autres instructions : boucles

- La boucle

```
<xsl:for-each select="requête XPath">
```

...

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

- Le corps du **for-each** est relatif à la racine du résultat de la requête XPath ou de la liste de valeurs du résultat

- Exemple

```
<xsl:for-each select='//personnes/personne'>
```

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

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

# Autres instructions : tri, condition

- Les constructions XSLT

- Le tri

- `<xsl:sort select="requête XPath"/>`

- Les éléments sont atteints en fonction du critère de tri (valeur d'élément ou d'attribut) désigné par la requête

- La condition

- `<xsl:if test="condition XPath">`

- ...

- `</xsl:if>`

# Autres instructions : choix

- Les constructions XSLT

- Le choix

- `<xsl:choose>`  
    `<xsl:when test="condition XPath">`  
        ...  
    `</xsl:when>`  
    ...  
    `<xsl:otherwise>`  
        ...  
    `</xsl:otherwise>`  
  `</xsl:choose>`



# XSL:CALL-TEMPLATE

- `<xsl:call-template name="nom_template">`  
    `<xsl:with-param name="nom_param" select="expression" />`  
`</xsl:call-template>`

# Transformation XSL via un navigateur

- Pour transformer un document XML en utilisant une feuille de style XSL, ajouter l'instruction suivante après le prolog du document XML

`<?xml version=....>` :

**`<?xml-stylesheet type="text/xsl" href="fichier.xsl"?>`**

# Résumé XSLT

- XSLT est un langage très puissant de transformation d'un arbre XML en un autre
- XSLT permet en particulier de publier des données XML sur le Web par transformation en document HTML standard
- XSLT est très utilisé :
  - Pour publier des contenus XML
  - Pour transformer des formats (EAI, B2B)

END

