# XPath

Philippe.Poulard@inria.fr

```
<?xml version="1.0"?>
<Contacts>
  <Contact>
    <Name>John Doe</Name>
    <Phone>626-555-3456</Phone>
  </Contact>
  <Contact>
    <Name>Joe <!--I need to check whether
              he prefers Joe or Joseph --> Smith </Name>
    <Phone>212-555-3456</Phone>
  </Contact>
  <Contact>
    <Name>Jane <![CDATA[& Ben]]> Reilly</Name>
    <Phone>212-555-2341</Phone>
  </Contact>
  <Contact>
    <Name>Mohammed Jones</Name>
    <Phone>718-555-2349</Phone>
    <Phone>914-555-7698</Phone>
  </Contact>
  <Contact>
    <Phone>914-555-3145</Phone>
    <Name>David Smith</Name>
  </Contact>
  <Contact>
    <Name>Jason Smith</Name>
    <Phone></Phone>
  </Contact>
  <Contact>
    <Name>Xao Li</Name>
    <Name>Jonathan Li</Name>
    <Phone>212-555-0998</Phone>
  </Contact>
</Contacts>
```

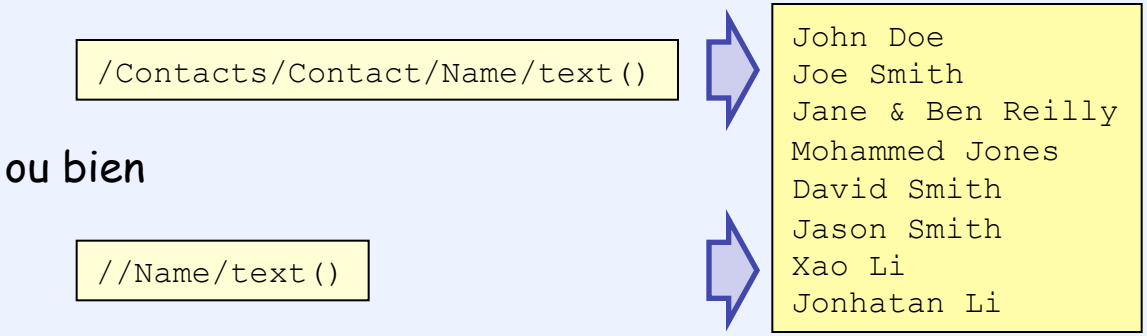
Ecrire un programme qui affiche les noms

```
Element contactsElement = doc.getDocumentElement();
NodeList contacts = contactsElement.getChildNodes();
for (int i = 0; i < contacts.getLength(); i++) {
    Node current = contacts.item(i);
    if (current.getNodeType() == Node.ELEMENT_NODE) {
        Element contact = (Element) current;
        NodeList children = contact.getChildNodes();
        for (int j = 0; j < contacts.getLength(); j++) {
            Node candidate = children.item(j);
            if (candidate.getNodeType() == Node.ELEMENT_NODE) {
                // Assuming name is the first child element
                String name = candidate.getFirstChild().getNodeValue();
                System.out.println(name);
                // Assuming there's only one name per contact
               break;
            }
        }
    }
}
```

```
John Doe
Joe
Jane
Mohammed Jones
914-555-3145
Jason Smith
Xao Li
```

Déjà long et pas très robuste :

- Support des CDATA ?
- Support des commentaires et PI ?
- Et si le nom n'est pas en première position ?
- Support des entités ?




ou bien

```
//Name/text()
```

(en vérité seuls les nœuds sont sélectionnés; comme en SQL il faudrait itérer sur les résultats et afficher la ''stringValue'' de chaque noeud)

# Sommaire


- Qu'est-ce que XPath ?
- Analogies entre XPath et *file systems*
- Fonctionnalités de base :
  - Sélection d'un ensemble de nœuds
  - Sélection de nœud d'attribut
- Modèle de données XPath
- Chemins et étapes de localisation, tests de nœud, jokers
- Axes XPath
- Formes abrégées et verbeuses
- Predicats, collections, prédicats composés
- Fonctions XPath :
  - fonctions d'ensembles de nœuds,
  - fonctions de chaînes,
  - fonctions booléennes,
  - fonctions numériques
- Divers
- Contexte d'évaluation / Utilisation
- XPath2

Standard du w3c

`http://www.w3c.org/TR/xpath`

XPath est un langage non XML (syntaxe non XML) utilisé pour **adresser** des items dans un document XML (modèle logique)

Il est intensivement utilisé dans XSLT

- Sa syntaxe élémentaire ressemble à l'expression d'un chemin dans un *file system*
- Sa syntaxe formelle plus du tout

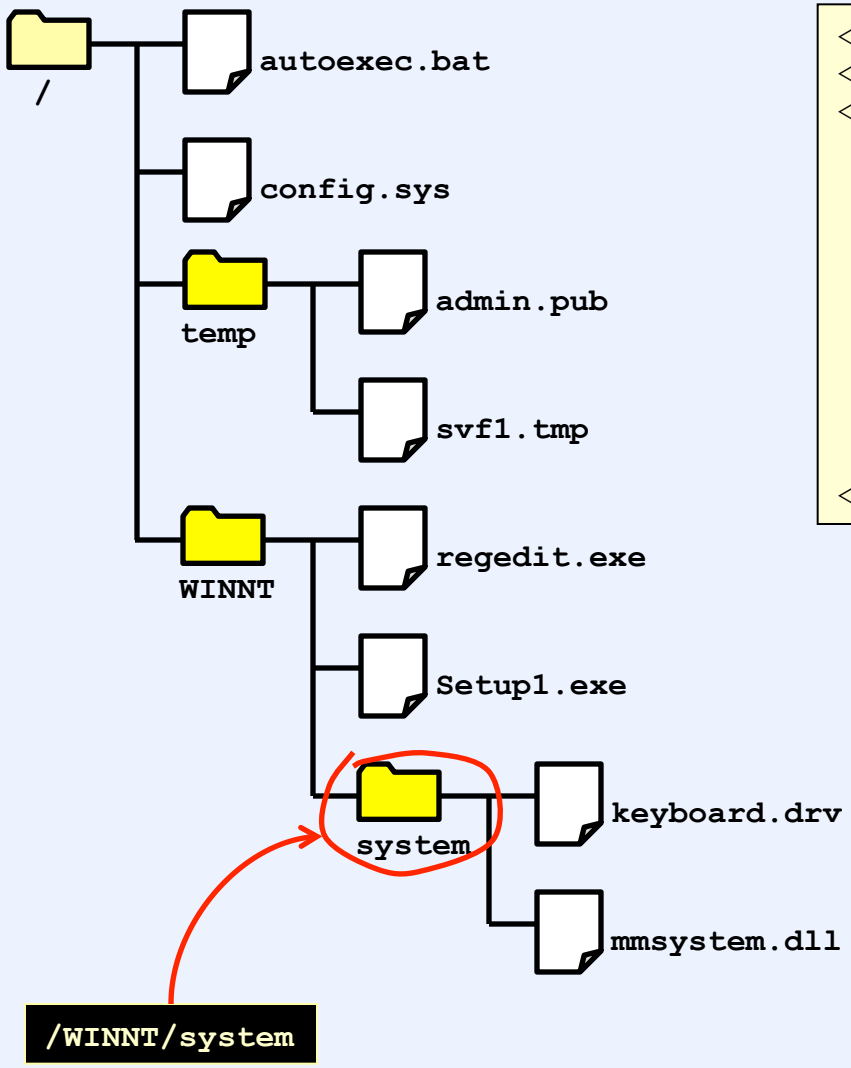
Une **expression XPath** peut s'exprimer :
- relativement, vis à vis d'un **nœud contextuel**
- d'une manière absolue

Le résultat d'une expression XPath peut être :
- **un ensemble de nœuds** (ou un nœud seul)
- une chaîne de caractères
- un nombre
- un booléen

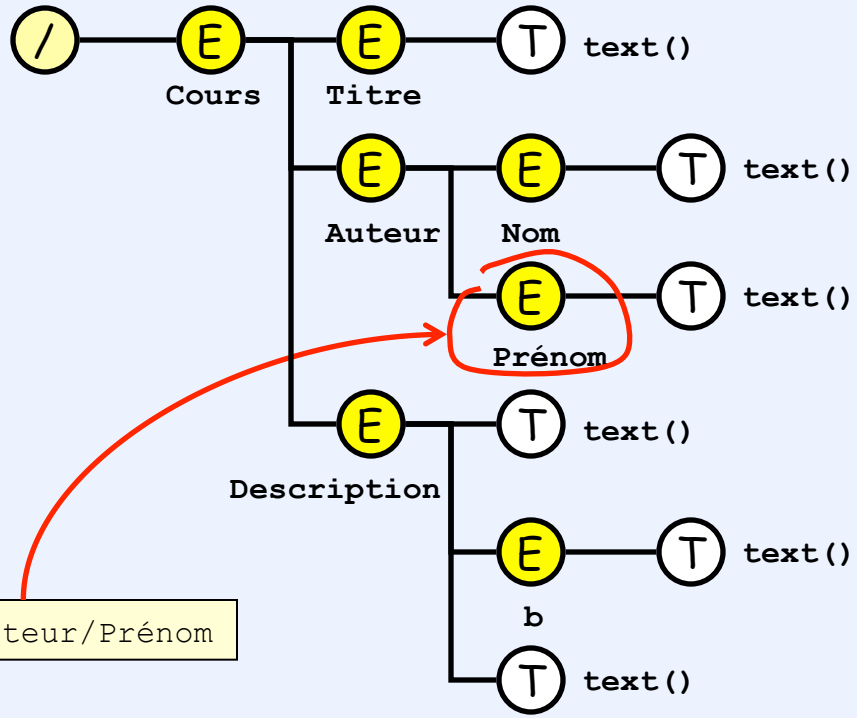
XPath est un langage de requêtes pour extraire des données d'un document XML

# Analogies XPath et *file system*





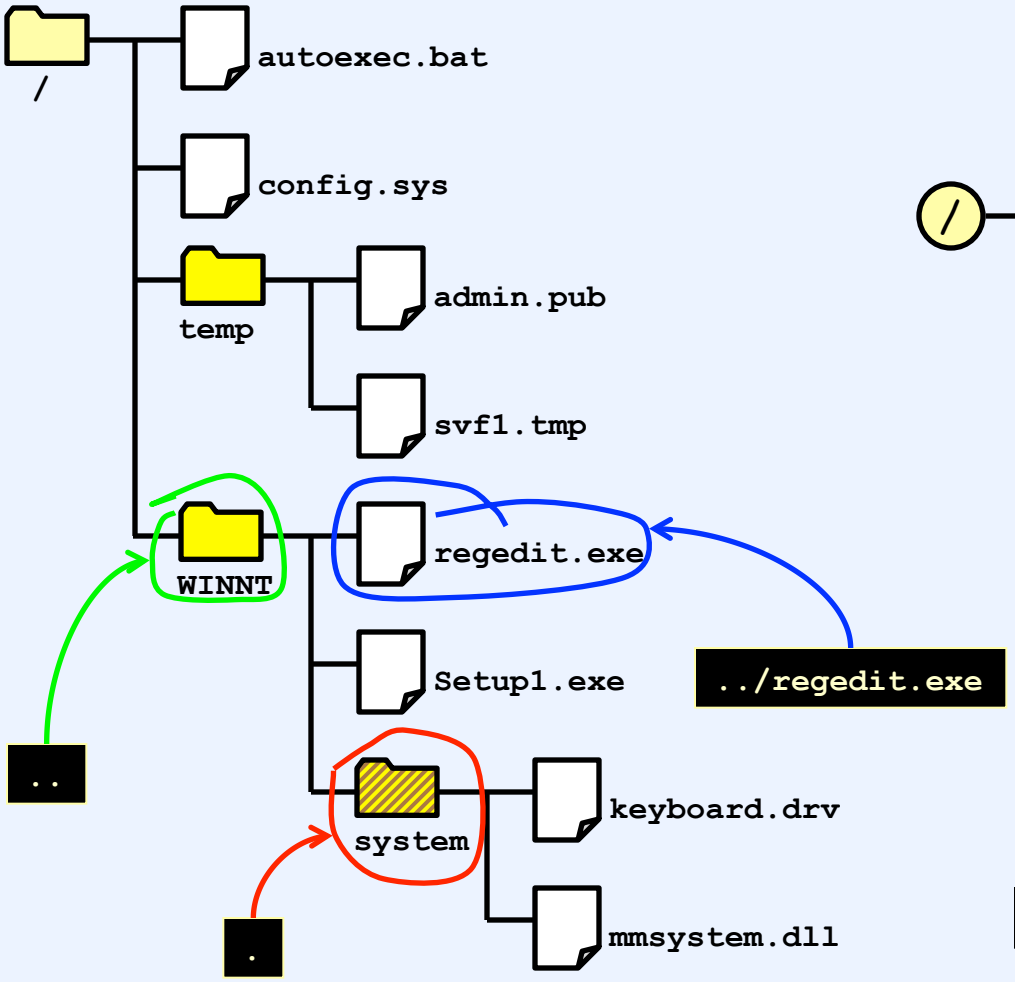

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Cours SYSTEM "Cours.dtd">
<Cours>
          <Titre>Cours XML</Titre>
          <Auteur>
                    <Nom>Poulard</Nom>
                    <Prénom>Philippe</Prénom>
          </Auteur>
          <Description>
          Ce cours aborde les <b>concepts</b> de
          base mis en &#339;uvre dans XML.
          </Description>
</Cours>
```

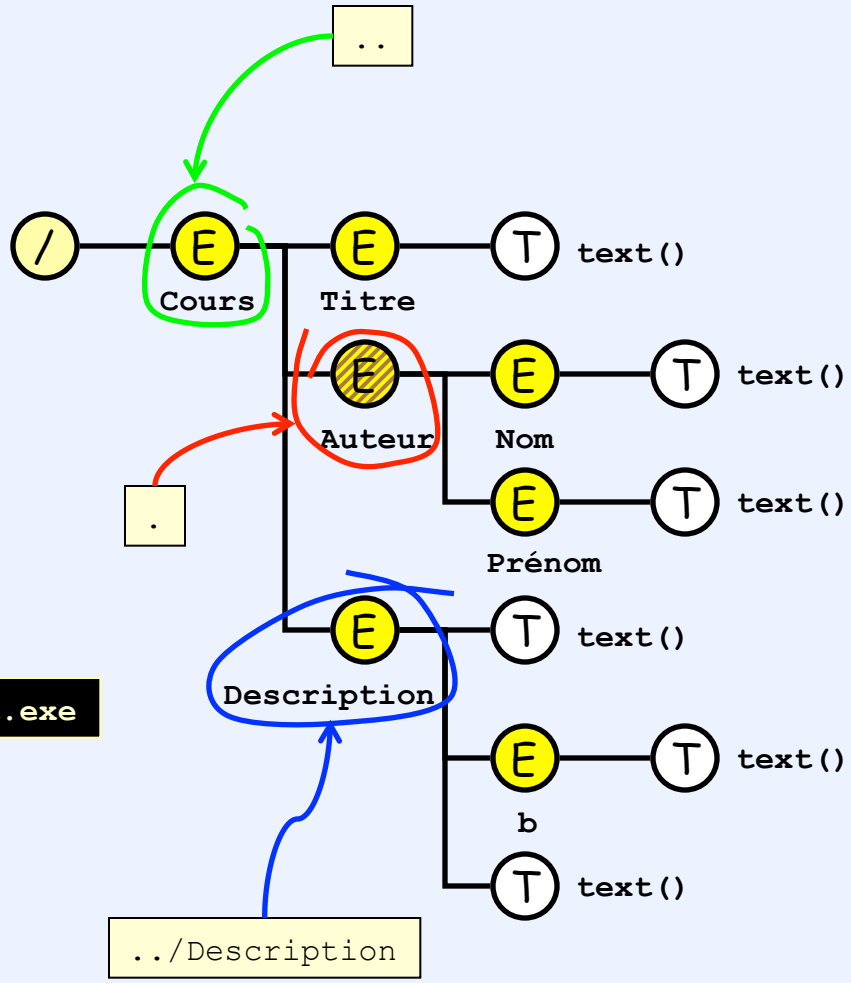



Expression XPath qui retourne 1 nœud → `/Cours/Auteur/Prénom`

# Adressage relatif dans XPath






Répertoire courant

Le répertoire courant est déterminé par l'interpréteur de commandes

Nœud contextuel

Le nœud contextuel est déterminé par l'application utilisatrice de XPath

# Comparaison XPath / *file system*

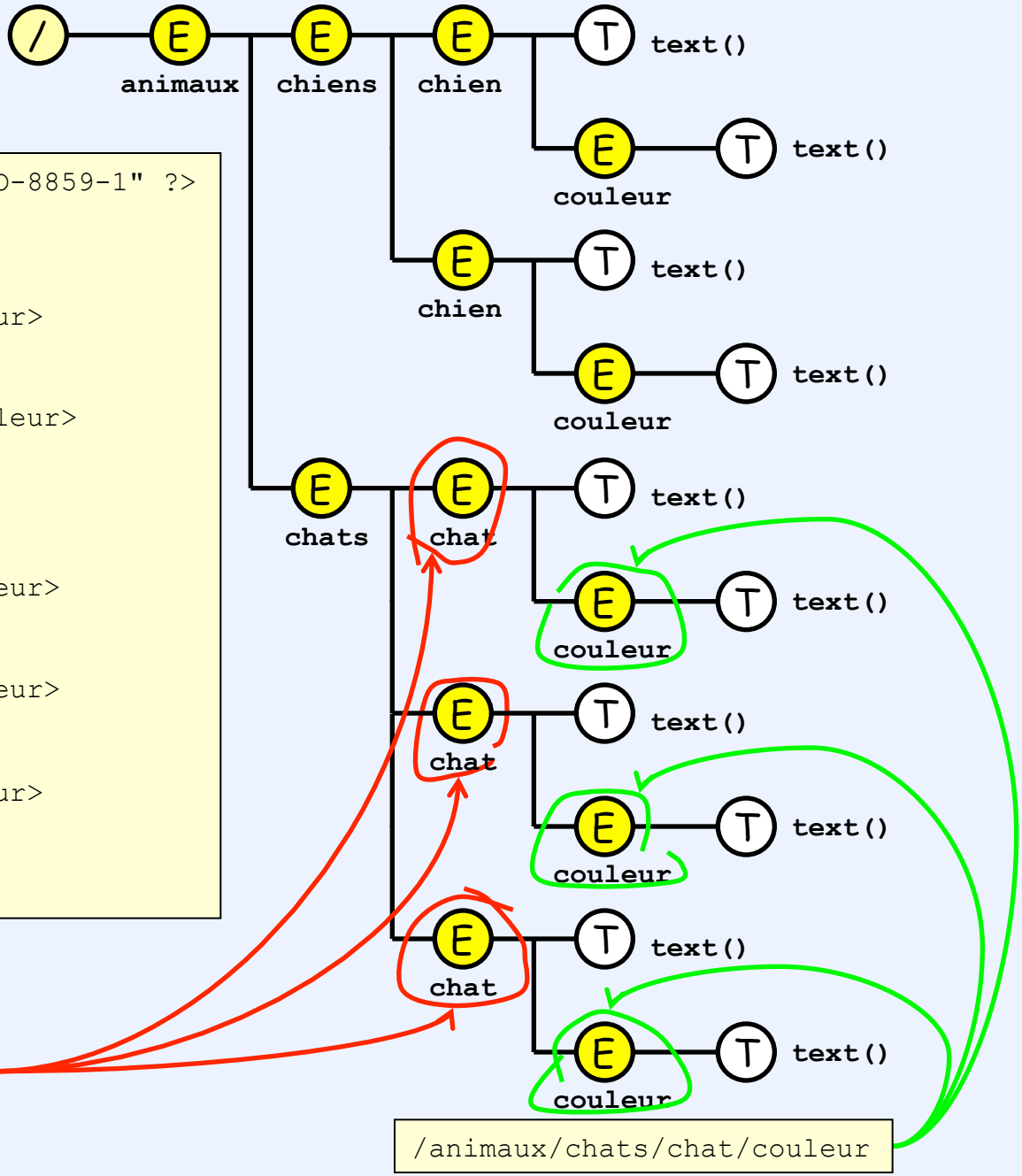


| | Expressions XPath | File system URLs |
|---|---|---|
| Analogies | Hiérarchie constituée d'éléments et d'autres types de nœuds | Hiérarchie constituée de fichiers et de répertoire |
| | Une expression peut être évaluée à partir d'un nœud spécifique, appelé "le **nœud contextuel**" de la requête | Une URL peut être évaluée à partir d'un d'un répertoire spécifique, appelé "le **répertoire courant**" |
| Différences | Une expression XPath identifie un ensemble d'objets qui peut être un ou plusieurs nœuds | Une URL identifie un fichier unique |
| | A chaque niveau les noms des éléments peuvent ne **pas** être **uniques** | A chaque niveau les noms des fichiers sont **uniques** |

# Sélection d'un ensemble de nœuds



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<animaux>
    <chiens>
        <chien>Lassie
            <couleur>noir</couleur>
        </chien>
        <chien>Médor
            <couleur>marron</couleur>
        </chien>
    </chiens>
    <chats>
        <chat>Félix
            <couleur>crème</couleur>
        </chat>
        <chat>Tom
            <couleur>crème</couleur>
        </chat>
        <chat>Rominet
            <couleur>gris</couleur>
        </chat>
    </chats>
</animaux>
```

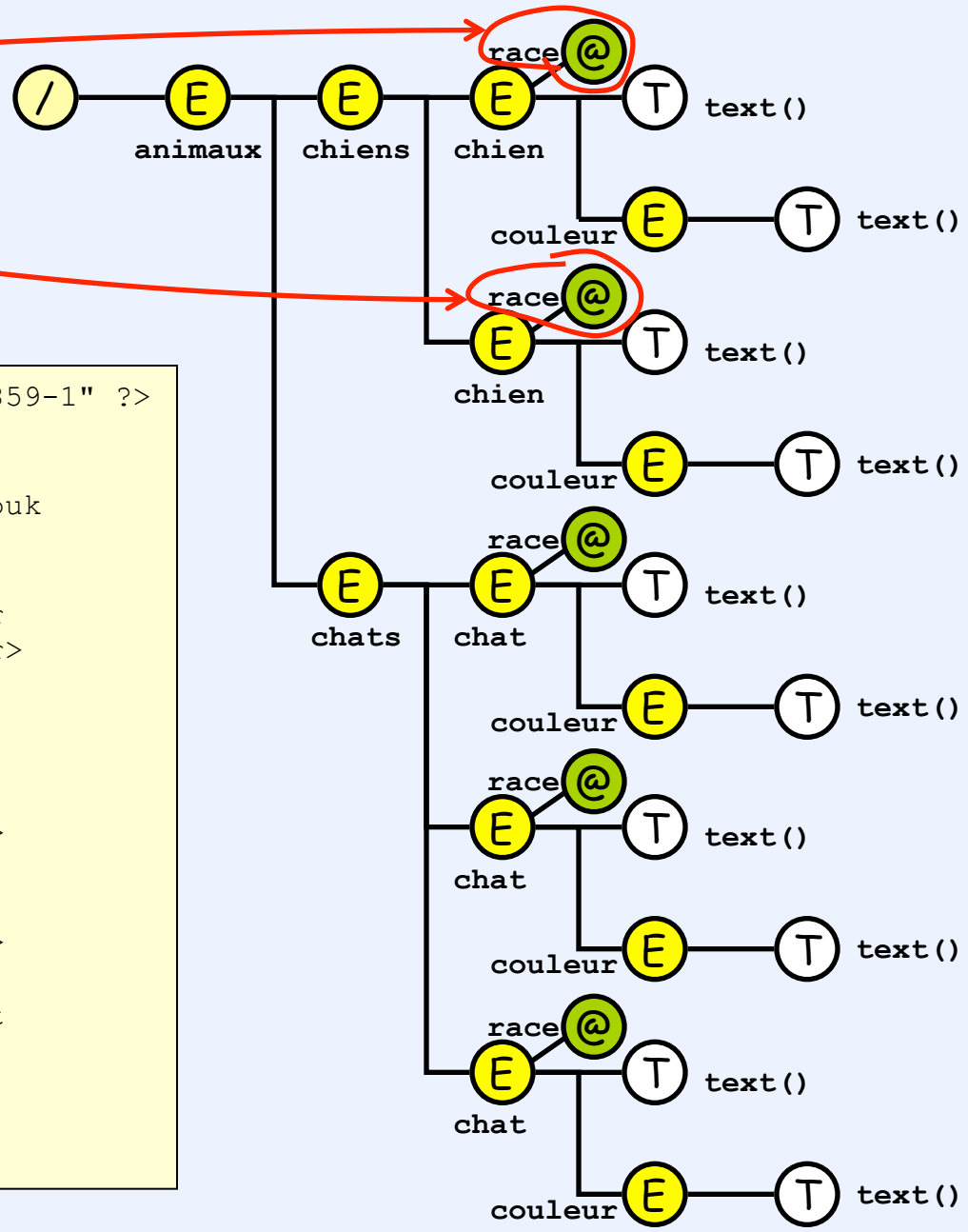


# Sélection de nœud d'attribut

@ ➔ attribut "at"

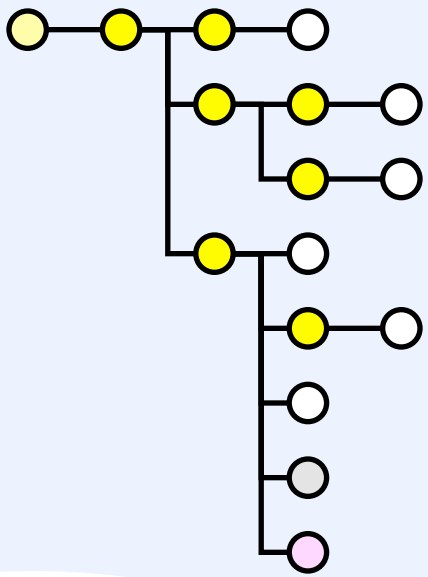
```
/animaux/chiens/chien/@race
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<animaux>
    <chiens>
        <chien race="Labrador">Mabrouk
            <couleur>noir</couleur>
        </chien>
        <chien race="Labrador">Médor
            <couleur>marron</couleur>
        </chien>
    </chiens>
    <chats>
        <chat race="Siamois">Félix
            <couleur>crème</couleur>
        </chat>
        <chat race="Birman">Tom
            <couleur>crème</couleur>
        </chat>
        <chat race="Abyssin">Rominet
            <couleur>gris</couleur>
        </chat>
    </chats>
</animaux>
```



## Types de nœuds XPath

Document XML ➔ arborescence de noeuds



- nœud racine (nœud document)
  - / `/`
- nœuds d'élément
  - E `foo-élément`
- nœuds de texte
  - T `text()`
- nœuds d'attribut
  - @ `@bar-attribut`
- nœuds de commentaire
  - -- `comment()`
- nœuds d'instruction de traitement
  - ? `processing-instruction()`
- nœuds d'espace de nom
  - ns `namespace::foo-ns`

Sont exclus du modèle :
- les sections CDATA
- les appels d'entités
- la clause DTD
- la déclaration XML

## Nœuds XPath

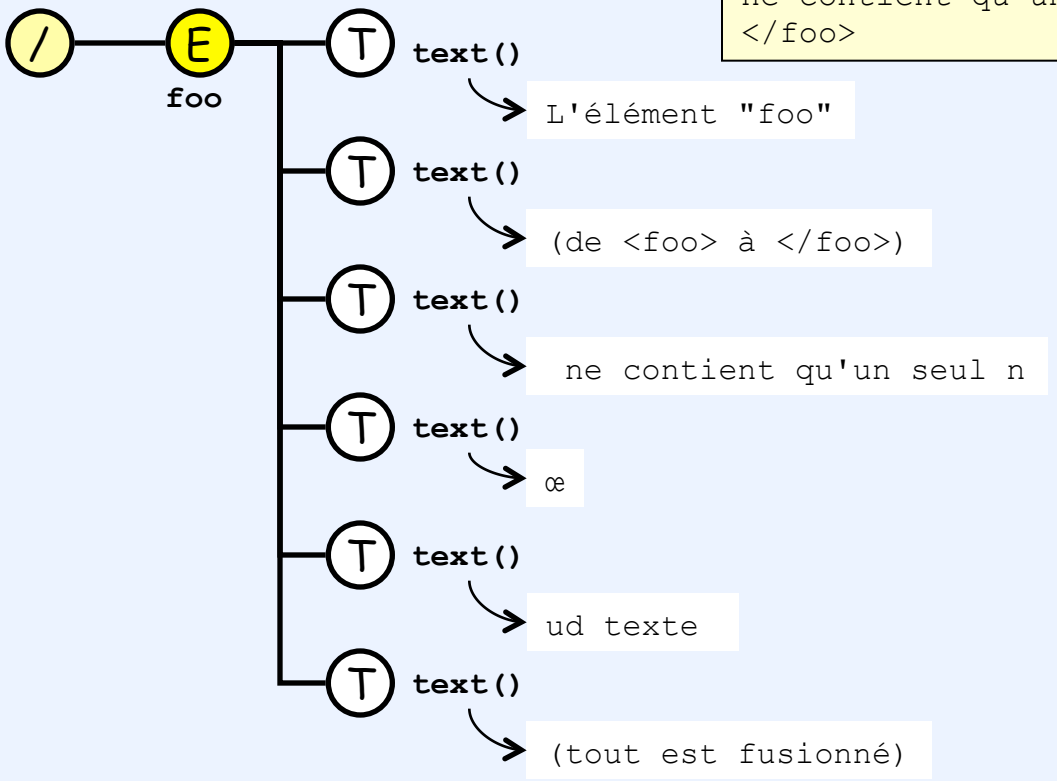
Le modèle n'est pas seulement basé sur les nœuds eux-mêmes mais aussi sur leurs relations.

XPath ne permet pas d'accéder directement aux éléments, aux attributs et autres constructions de balisage.
Il permet seulement d'accéder aux nœuds construits à partir de ce balisage.

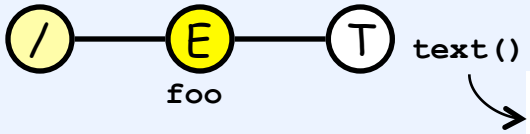
Les appels d'entités parsées sont résolus

```
<?xml encoding="ISO-8859-1"?>
<!DOCTYPE foo [
          <!ENTITY fusion "(tout est fusionné)">
]>
<foo>L'élément "foo" <![CDATA[(de <foo> à </foo>)]]>
ne contient qu'un seul n&#339;ud texte &fusion;
</foo>
```

## Normalisation




*normalization*

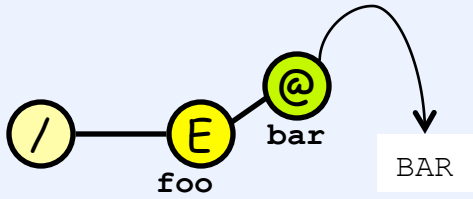



L'élément "foo" (de <foo> à </foo>) ne contient qu'un seul nœud texte (tout est fusionné)

Les appels d'entité, les appels de caractère et les sections CDATA sont indiscernables des contenus textuels

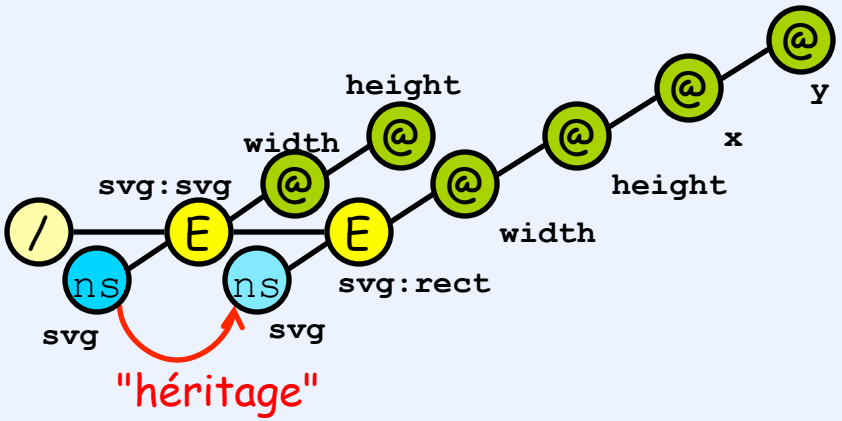
Les valeurs par défaut des attributs fournis par la DTD sont fixées

```
<?xml encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ATTLIST foo bar CDATA #FIXED "BAR">
]>
<foo/>
```




Les attributs `xmlns` sont rapportés comme des nœuds d'espace de noms, et sont attachés à tous les nœuds d'élément et d'attributs dans la portée de la déclaration

```
<?xml version="1.0"?>
<svg:svg width="18cm" height="6cm"
         xmlns:svg="http://www.w3.org/2000/svg">
    <svg:rect x="1" y="1" width="198"
          height="118"/>
</svg:svg>
```



# Modèle de données XPath



| Construction XML | Caractéristiques du noeud | | |
|---|---|---|---|
| | Type | Nom | Valeur |
| - | Document (/) | - | - |
| `<élément>` | Element (E) | Nom de l'élément | - |
| `attribut="valeur"` | Attr (@) | Nom de l'attribut | Valeur de l'attribut |
| `texte` | Text (T) | - | Texte |
| `<!-- commentaire -->` | Comment (--) | - | Commentaire |
| `<?cible argument?>` | ProcessingInstruction (?) | Nom de la cible | Argument |
| `xmlns:préfixe="URI"` | Namespace (ns) | Préfixe | URI |

# Chemin de localisation

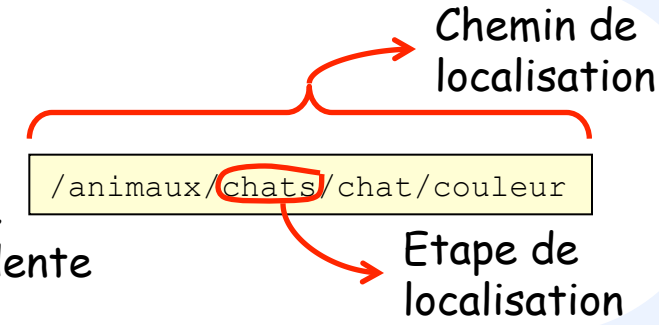


- `..`
- `/`
- `../Description`
- `/animaux/chats/chien/@race`
- `/animaux/chats/chat`
- `.`
- `/animaux/chats/chat/couleur`

Adressage d'un ensemble hétérogène de nœuds
→ Peut être vide

Chaque étape est évaluée dans le **contexte** de l'étape précédente

Le contexte d'évaluation d'une étape est l'ensemble des nœuds obtenu par l'évaluation de l'étape précédente




Chemin de localisation de la racine : `/`

il s'agit de la racine du document, pas de l'élément racine

## Unions d'étapes de localisation

Avec XSLT, on peut réaliser des unions d'expression avec `|`

```
/animaux/chats/chat|/animaux/chiens/chien
```

```
/animaux/(chiens|chats)/(chat|chien)
```

# Tests de nœud dans les étapes de localisation



## Tests de nœuds nommés

| | | |
|---|---|---|
| Nœuds d'élément | S'exprime par le nom de l'élément | `chats` |
| Nœuds d'attribut | S'exprime par le nom de l'attribut précédé de `@` | `@race` |
| Nœuds d'instruction de traitement | | `processing-instruction("xml-stylesheet")` |

## Tests de type de nœuds

| | |
|---|---|
| Nœuds de commentaire | `comment()` |
| Nœuds de texte | `text()` |
| Nœuds d'instruction de traitement | `processing-instruction()` |

Les nœuds d'espace de nommage ne sont pas adressables par des tests de nœud

# Jokers de tests de nœud

Les jokers permettent de sélectionner différents items et types de nœuds différents

## Joker pour les éléments : *

Exemple : `/animaux/(chiens|chats)/(chien|chat)`

↓

`/animaux/*/*`

} S'il n'y a pas d'autres animaux, ces expressions sont équivalentes

Préfixage par un espace de noms : `svg:*`

( c'est l'URI associée qui est prise en compte, et non pas le préfixe lui-même
l'espace de nom doit être spécifié au niveau du processeur XPath )

dépend du processeur et de l'application utilisatrice

## Joker pour les attributs : @*

Exemple : `/animaux/chats/chat/@*`

Sélectionne tous les attributs de l'élément `chat`, s'il y en a plusieurs

## Joker universel : node()

Sélectionne tous les nœuds, quel que soit leur type

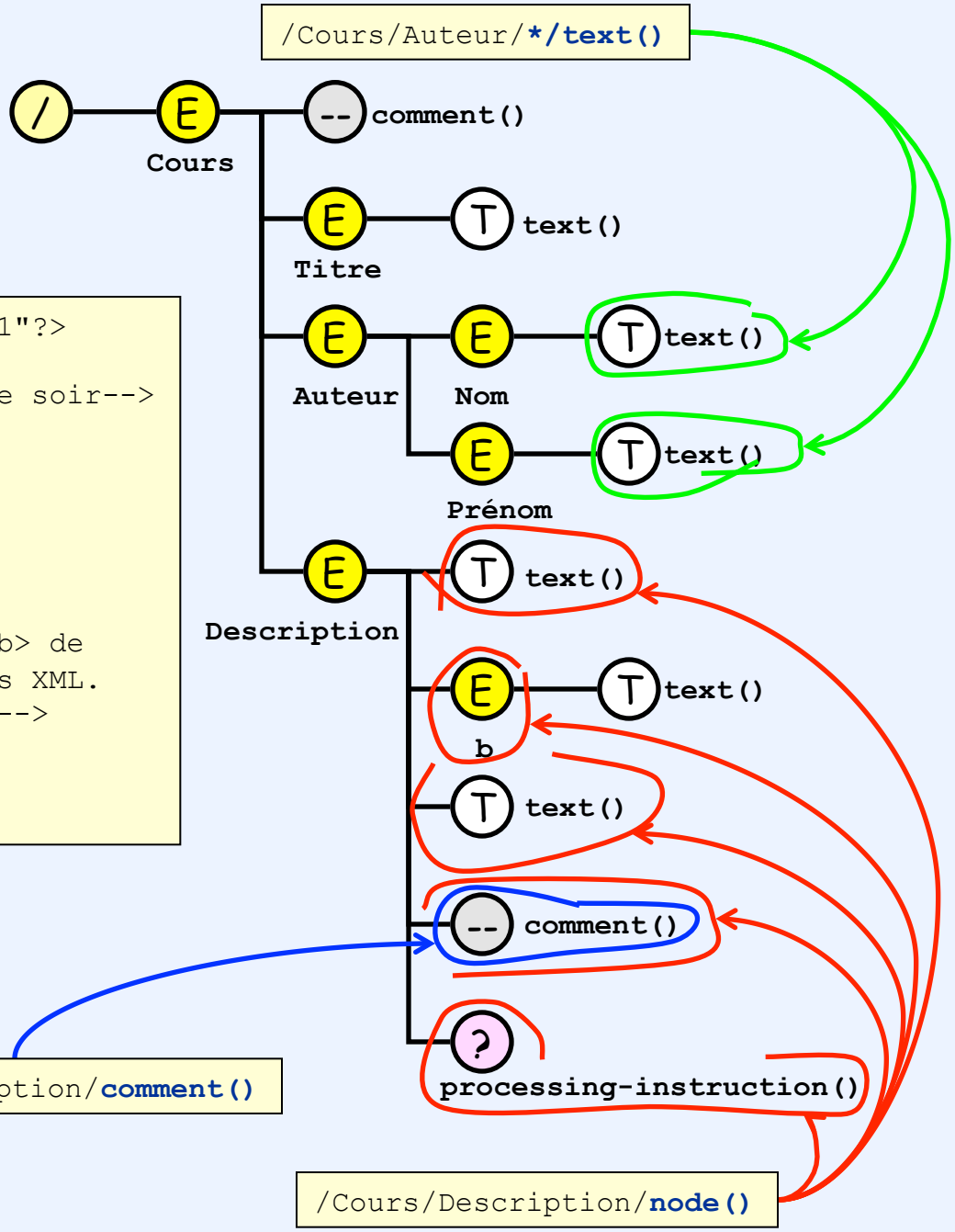
Exemple : `/animaux/chats/chat/node()`

Sélectionne tous les nœuds de l'élément `chat`, s'il y en a plusieurs

# Exemples d'utilisation des jokers



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
    <!--penser à acheter du pain pour ce soir-->
    <Titre>Cours XML</Titre>
    <Auteur>
        <Nom>Poulard</Nom>
        <Prénom>Philippe</Prénom>
    </Auteur>
    <Description type="html">
      Ce cours aborde les <b>concepts</b> de
            base mis en &#339;uvre dans XML.
      <!-- c'est le meilleur cours !!! -->
      <?diaporama file="xml.ppt"?>
    </Description>
</Cours>
```



XPath permet de spécifier la "direction" dans laquelle un test de nœud doit être réalisé.
Il y a 13 axes différents.

Syntaxe : `axe::test-de-noeud`

XPath propose une syntaxe abrégée pour les axes les plus utilisés.

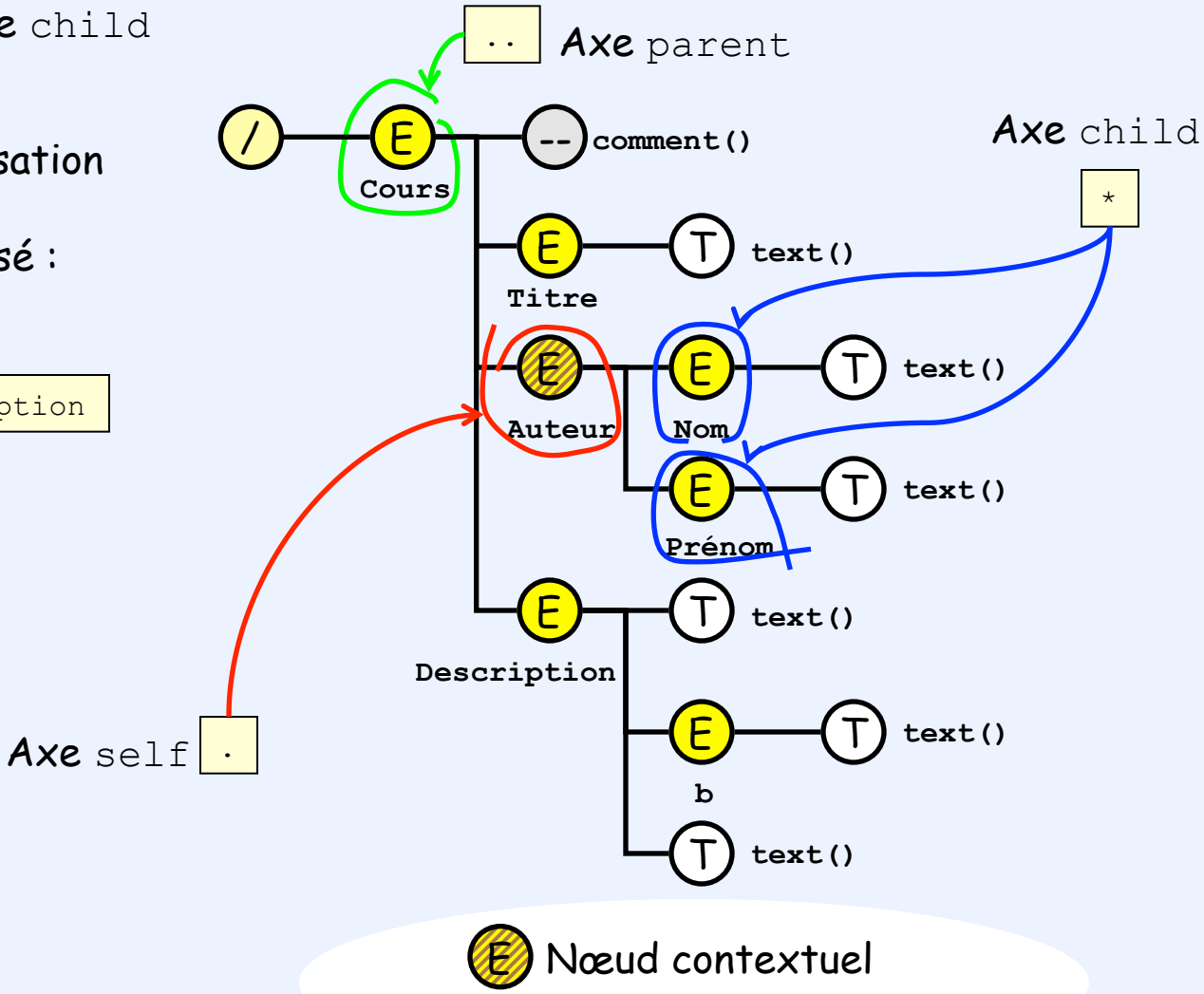
| | | |
|---|---|---|
| `.` | = | `self::*` |
| `..` | | `parent::*` |
| `*` | | `child::*` |

L'axe par défaut est l'axe `child`

A chaque étape de localisation (séparées par `/`), un axe différent peut être utilisé :

`../Description`

= `parent::*/child::Description`




**self**
**child**
**parent**
attribute
namespace
descendent
descendent-or-self
ancestor
ancestor-or-self
following
following-sibling
preceding
preceding-sibling

# Axes typés XPath

2 axes particuliers permettent de sélectionner des types de nœud :
`attribute` et `namespace` permettent d'accéder aux nœuds correspondants

`namespace::foo-ns`

`attribute::bar-attribut` = `@bar-attribut`

L'axe des attributs bénéficie d'une forme abrégée

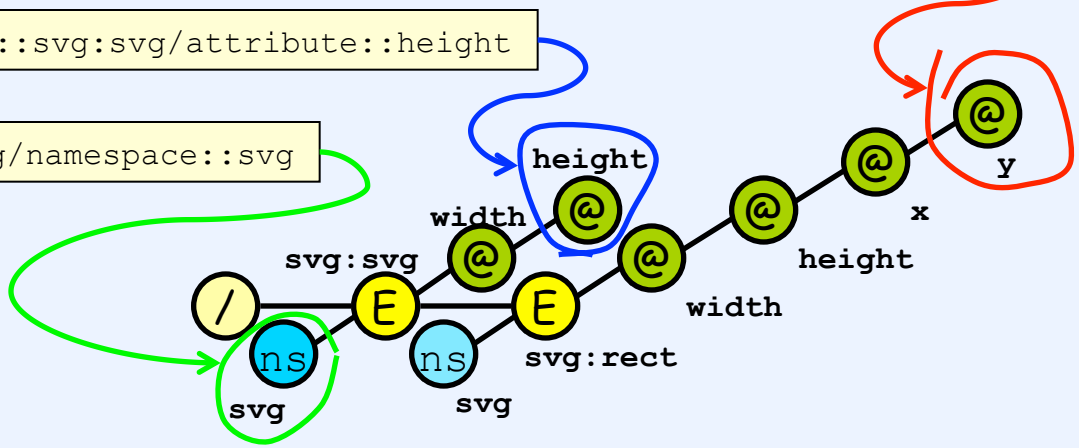
Les 11 autres axes permettent d'accéder à tous les types de nœud,
**SAUF** les nœuds de type **attribut** et les nœuds d'**espace de nommage**

```
<?xml version="1.0"?>
<svg:svg width="18cm" height="6cm"
         xmlns:svg="http://www.w3.org/2000/svg">
    <svg:rect x="1" y="1" width="198"
          height="118"/>
</svg:svg>
```

`/svg:svg/svg:rect/@y`

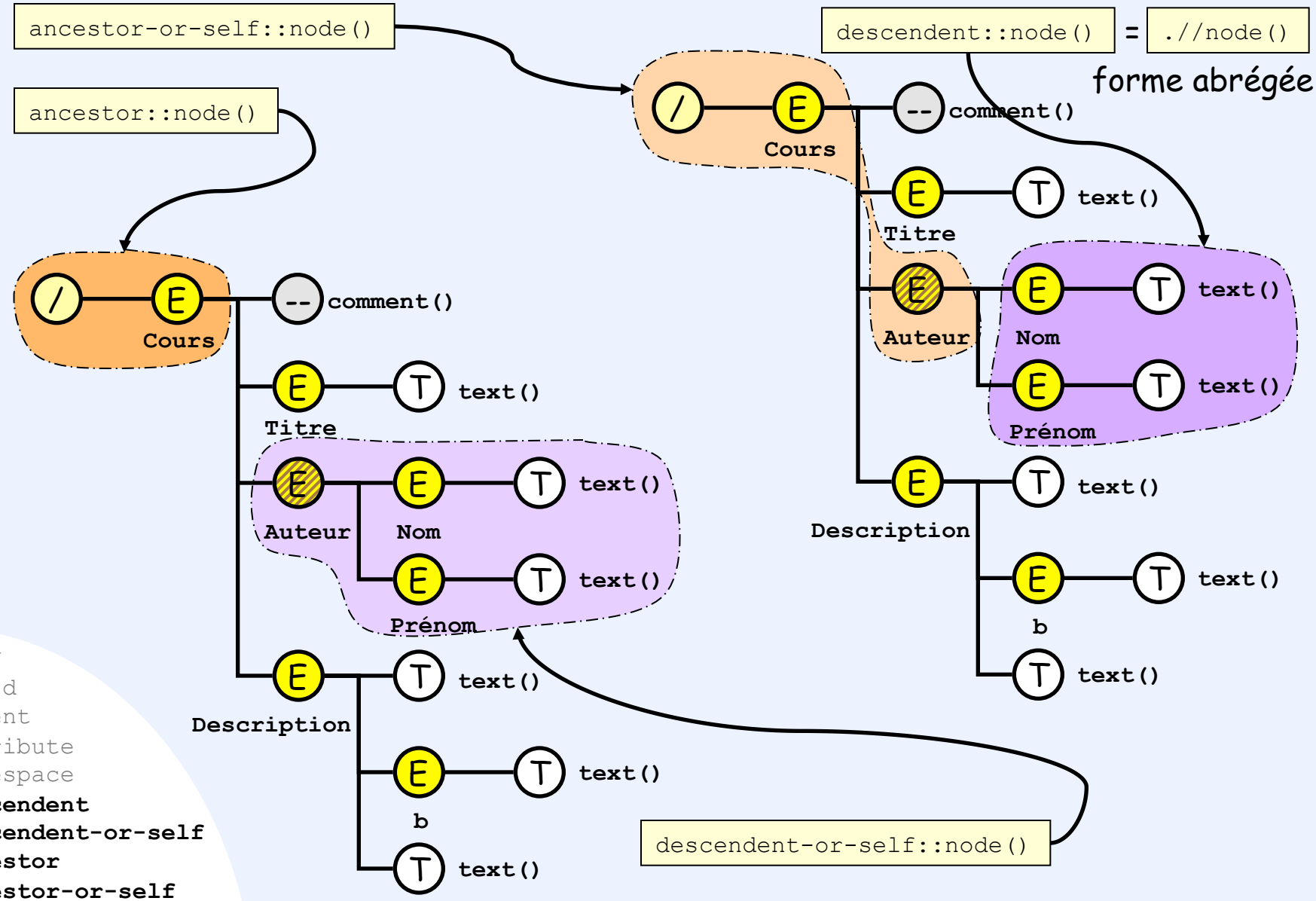
`/child::svg:svg/attribute::height`

`/child::svg:svg/namespace::svg`




self
child
parent
**attribute**
**namespace**
descendent
descendent-or-self
ancestor
ancestor-or-self
following
following-sibling
preceding
preceding-sibling

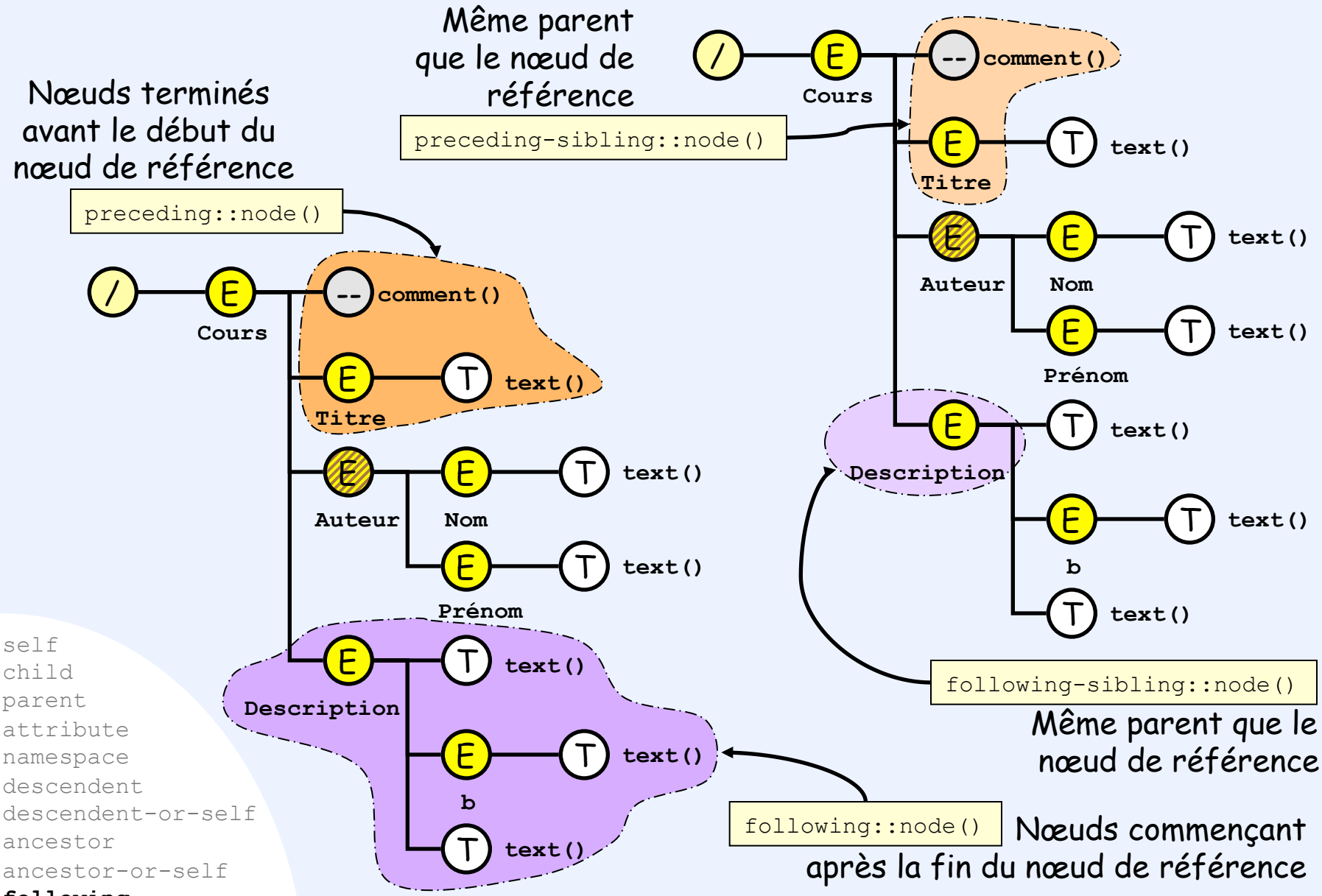
# Axes non typés XPath




self
child
parent
attribute
namespace
**descendent**
**descendent-or-self**
**ancestor**
**ancestor-or-self**
following
following-sibling
preceding
preceding-sibling

# Axes non typés XPath






self
child
parent
attribute
namespace
descendent
descendent-or-self
ancestor
ancestor-or-self
**following**
**following-sibling**
**preceding**
**preceding-sibling**

Joker universel : `node()`

Sélectionne tous les nœuds, quel que soit leur type

Exemple : `/animaux/chats/chat/node()`

Sélectionne tous les nœuds de l'élément `chat`, s'il y en a plusieurs

Sont exclus : les **attributs** et les **nœuds d'espace de noms**, puisqu'ils ne sont pas dans l'axe par défaut **`child`**

`/animaux/chats/chat/child::node()`

L'axe `child` contient seulement :

- les nœuds d'élément
- les nœuds de texte
- les nœuds de commentaire
- les nœuds d'instruction de traitement

`/animaux/chats/chat/attribute::node()`

`/animaux/chats/chat/attribute::*`

Ces 2 expressions sont équivalentes puisqu'il n'y a qu'un seul type de nœud dans l'axe `attribut`

# Formes verbeuses et abrégées des expressions



| Forme verbeuse | Exemples | | Forme abrégée |
|---|---|---|---|
| child::foo-élément | /child::animaux | /animaux | foo-élément |
| child::* | /child::* | /* | * |
| attribute::foo-élément | attribute::race | @race | @foo-élément |
| attribute::* | attribute::* | @* | @* |
| /descendant-or-self::node() | /descendant-or-self()::node()/chat | //chat | // |
| self::node() | self::node()/text() | text() | . |
| parent::node() | parent::node()/chiens | ../chiens | .. |
| [position()=n] | //chat[position()=3] | //chat[3] | [n] |

Limitation des axes XPath : il n'existe pas d'axe de parcours "naturel" d'un arbre

# Prédicats XPath

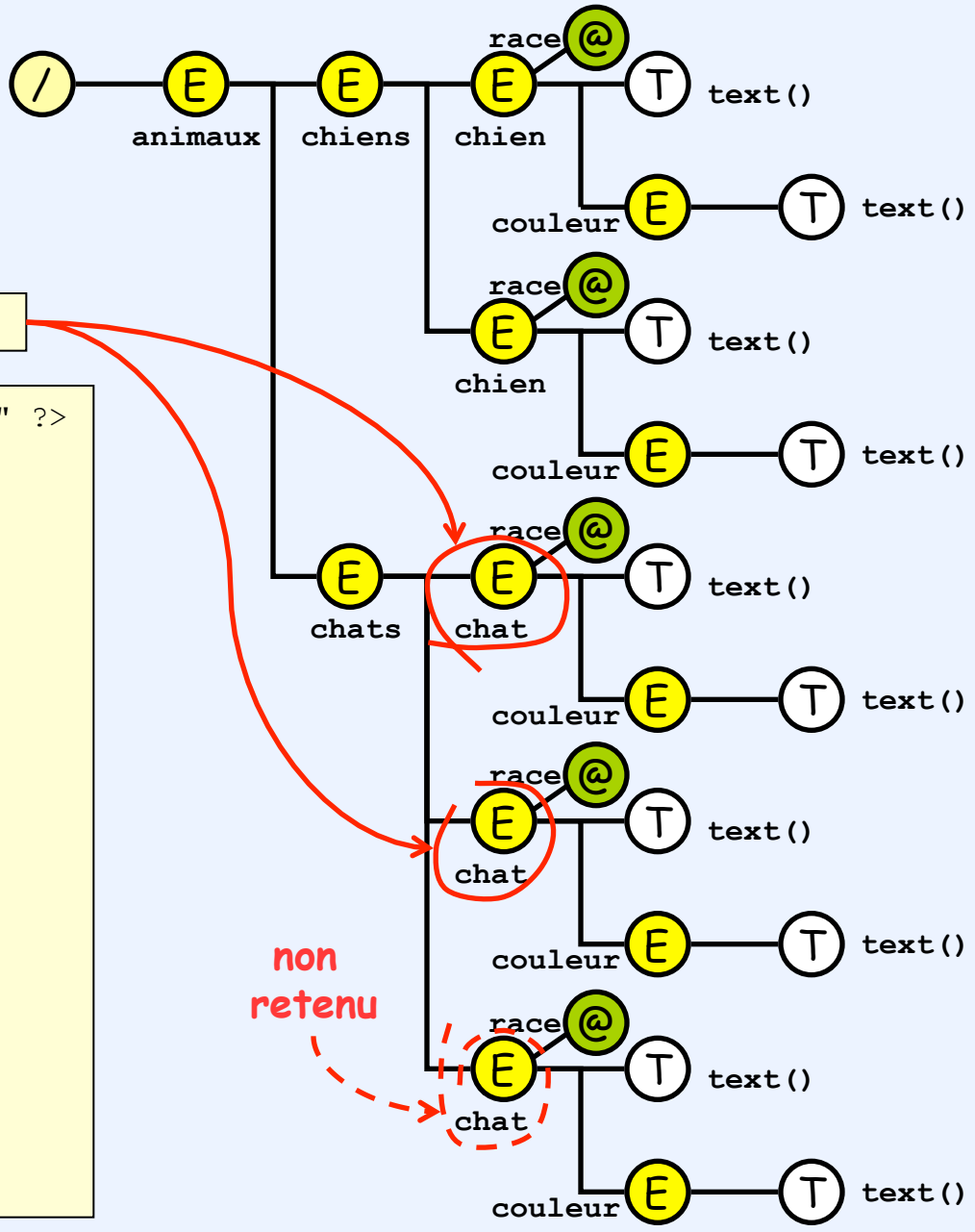
Un prédicat permet de spécifier un filtre dans une étape de localisation

Un prédicat est une expression booléenne qui sera testée pour chaque nœud de l'ensemble des nœuds de l'étape de localisation

Syntaxe : `axe::test-de-noeud[prédicat]`

```
//chat[couleur="crème"]
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<animaux>
    <chiens>
        <chien race="Labrador">Mabrouk
            <couleur>noir</couleur>
        </chien>
        <chien race="Labrador">Médor
            <couleur>marron</couleur>
        </chien>
    </chiens>
    <chats>
        <chat race="Siamois">Félix
            <couleur>crème</couleur>
        </chat>
        <chat race="Birman">Tom
            <couleur>crème</couleur>
        </chat>
        <chat race="Abyssin">Rominet
            <couleur>gris</couleur>
        </chat>
    </chats>
</animaux>
```



# Expressions booléennes dans les prédicats

```
and
or
not()
=
!=
<
<=
>
>=
( )
```

Test sur un attribut : `//*[*/@race="Birman"]`

`//*[@race!="Birman") and (@race!="Siamois")]` (suppose que l'attribut existe)

`/Cours/*[not(self::Titre)]`

## Expressions non booléennes

Les prédicats qui retournent des expressions non booléennes sont converties :

| Type | Résultat | Valeur |
|---|---|---|
| Chaîne | Vide<br>Non vide | Faux<br>Vrai |
| Ensemble de nœuds | Vide<br>Non vide | Faux<br>Vrai |
| Numérique | Numéro du nœud identique<br>Numéro du nœud différent | Vrai<br>Faux |

`//chat[string(couleur)]`

`//chat[couleur]`

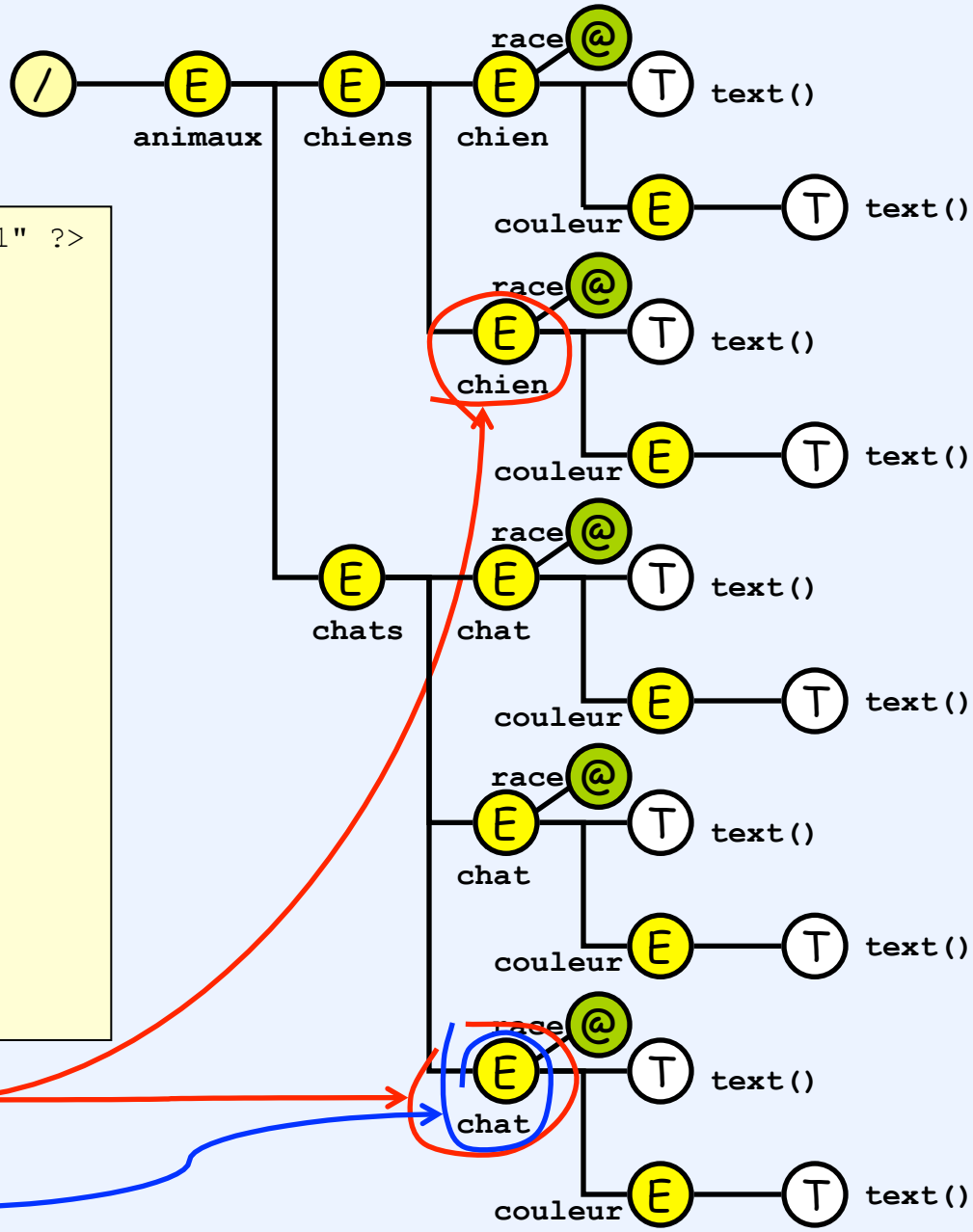
`//chat[2]`

Tous les nœuds peuvent être convertis en chaîne de caractère

Les expressions booléennes peuvent être exprimées avec un jeu de fonctions XPath, comme `string()`, `position()`, `last()`…

Les crochets [ et ] ont une priorité plus grande que le slash /, mais on peut aussi utiliser des parenthèses ( et ).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<animaux>
    <chiens>
        <chien race="Labrador">Mabrouk
            <couleur>noir</couleur>
        </chien>
        <chien race="Labrador">Médor
            <couleur>marron</couleur>
        </chien>
    </chiens>
    <chats>
        <chat race="Siamois">Félix
            <couleur>crème</couleur>
        </chat>
        <chat race="Birman">Tom
            <couleur>crème</couleur>
        </chat>
        <chat race="Abyssin">Rominet
            <couleur>gris</couleur>
        </chat>
    </chats>
</animaux>
```




```
/animaux/*/*[position()=last()]
```

```
(/animaux/*/*)[position()=last()]
```

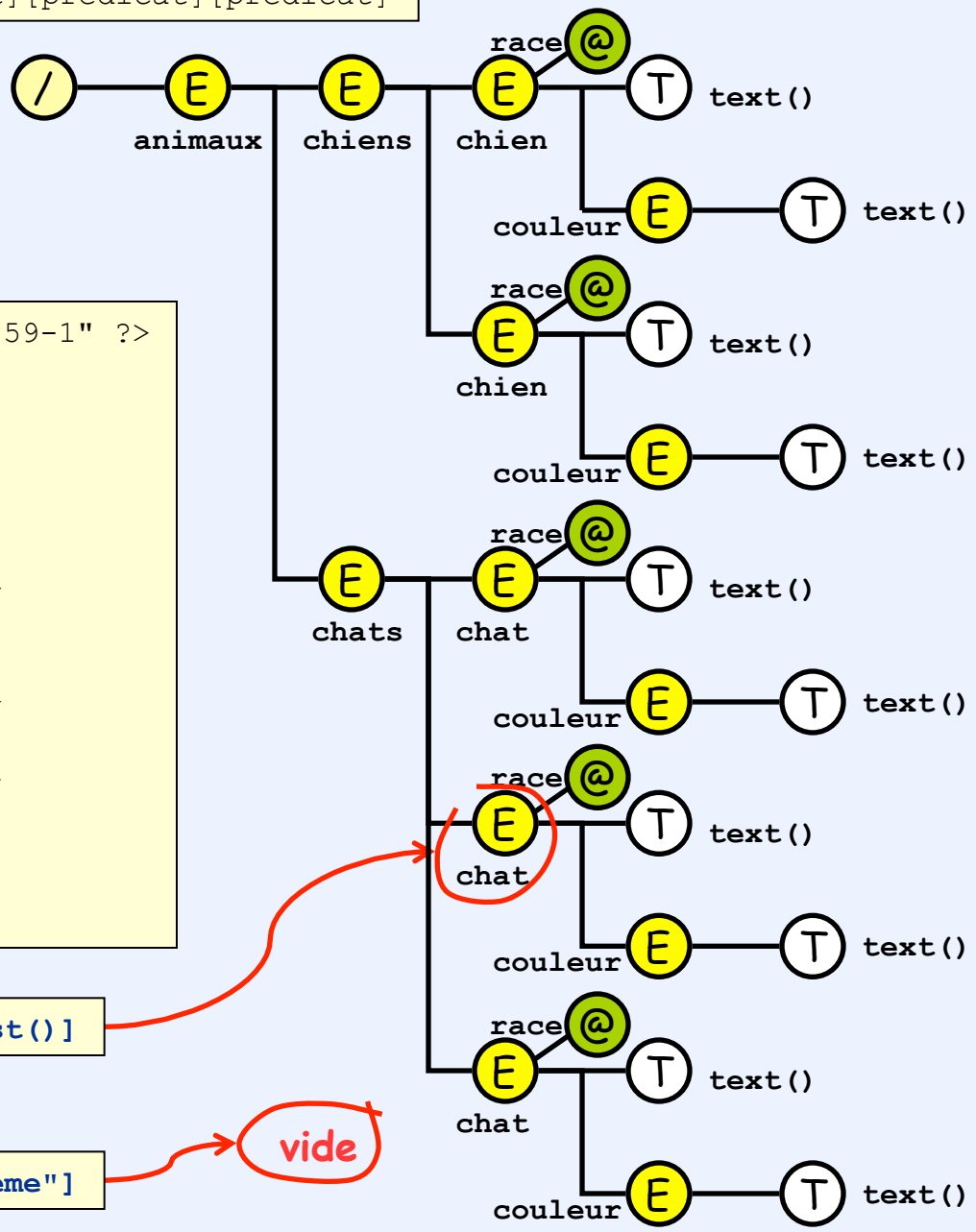
# Prédicats composés



Syntaxe : `axe::test-de-noeud[prédicat][prédicat][prédicat]`

Les prédicats sont évalués successivement sur le résultat du précédent.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<animaux>
    <chiens>
          …/…
    <chiens>
    <chats>
        <chat race="Siamois">Félix
            <couleur>crème</couleur>
        </chat>
        <chat race="Birman">Tom
            <couleur>crème</couleur>
        </chat>
        <chat race="Abyssin">Rominet
            <couleur>gris</couleur>
        </chat>
    </chats>
</animaux>
```




`//chat[couleur="crème"][position()=last()]`

`//chat[position()=last()][couleur="crème"]`

vide

# Fonctions d'ensemble de nœuds XPath

position() Retourne le numéro d'index du nœud de référence dans le contexte

```
/animaux/*/*[position()=1]
```

Forme abrégée :

```
/animaux/*/*[1]
```

last() Retourne le nombre de nœuds de l'ensemble de nœuds de référence

```
/animaux/*/*[position()=last()]
```

ou

```
/animaux/*/*[last()]
```

count(*node-set*) Retourne le nombre de nœuds de l'ensemble de nœuds passé en argument

```
/animaux/*[count(*)>2]
```

name(*node-set?*) Retourne le nom qualifié du premier nœud de l'ensemble de nœuds passé en argument

```
//*[name(.)!="svg:rect"]
```

local-name(*node-set?*) Retourne la partie locale du nom du premier nœud de l'ensemble de nœuds passé en argument

```
//*[local-name(.)!="rect"]
```

namespace-uri(*node-set?*) Retourne l'URI de l'espace de nom du premier nœud de l'ensemble de nœuds passé en argument

```
//*[namespace-uri(.)="http://www.w3.org/2000/svg"]
```

id(*string*) Retourne les nœuds ayant un ID égal à l'un des mots de la chaîne passée en argument

```
id("ref123 ref456 ref789")
```

# Fonctions de chaînes XPath

| | |
|---|---|
| concat(*string**) | Retourne la concaténation des chaînes passées en argument |
| contains(*string, string*) | Retourne vrai si la deuxième chaîne passée en argument est contenue dans la première chaîne passée en argument |
| starts-with(*string, string*) | Retourne vrai si la première chaîne passée en argument commence par la deuxième chaîne passée en argument |
| normalize-space(*string*) | Retourne la chaîne après normalisation des espaces |
| string(*object*) | Retourne la chaîne correspondante |
| string-length(*string*) | Retourne la longueur de la chaîne passée en argument |
| substring(*string, number, number?*) | Retourne la partie de la chaîne passée en argument qui commence à la position indiquée par le 2e argument sur la longueur indiquée par le 3e argument |
| substring-after(*string, string*) | Retourne la partie de la chaîne passée en argument qui suit la première occurrence de la 2e chaîne passée en argument |
| substring-before(*string, string*) | Retourne la partie de la chaîne passée en argument qui précède la première occurrence de la 2e chaîne passée en argument |
| translate(*string, string, string*) | Retourne la chaîne passée en 1er argument dont les caractères de la 2e chaîne passée en argument ont été remplacés par ceux de même index de la 3e chaîne passée en argument |

# Conversion des nœuds en chaînes



Tous les nœuds peuvent être convertis en chaîne de caractère

string(*object*) Retourne la chaîne correspondante, en fonction du type :

| Type de nœud | Résultat |
|---|---|
| Text | La valeur du nœud |
| Comment | La valeur du nœud |
| ProcessingInstruction | La valeur du nœud |
| Attribute | La valeur du nœud |
| Element | La concaténation des valeurs de nœuds texte du contenu |
| Document | La concaténation des valeurs de nœuds texte du contenu |
| Namespace | La valeur du nœud |

Ensemble de nœuds Le premier nœud seulement est converti

…donc :

`string(/animaux/*)` retourne Mabrouk noir Médor marron

`string(/animaux)` retourne Mabrouk noir Médor marron Félix crème Tom crème Rominet gris

Il n'y a pas de fonctions *uppercase* ni *lowercase*… mais il y a une fonction *translate*

```
translate("Texte à Casse Mixte","abcdefghijklmnopqrstuvwxyz","ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

TEXTE à CASSE MIXTE

# Fonctions booléennes XPath

not(*boolean*)

boolean(*object*)

| Type | Résultat | Valeur |
|---|---|---|
| Chaîne | Vide<br>Non vide | Faux<br>Vrai |
| Ensemble de nœuds | Vide<br>Non vide | Faux<br>Vrai |
| Numérique | 0 ou NaN<br>≠0 | Faux<br>Vrai |

true()

false()

lang(*string*) Retourne vrai si la langue passée en argument correspond à la langue du nœud courant.
La langue du nœud courant est déterminée par l'attribut `xml:lang` de ce nœud ou de ses ancêtres.
La casse est ignorée.
Si l'argument ne contient pas de suffixe (exemple : `fr`), la comparaison se limite à la langue non suffixée (exemple : `fr` pour `fr-ca`).

# Fonctions numériques XPath

ceiling(*number*) Retourne l'arrondi supérieur du nombre passé en argument

round(*number*) Retourne le plus proche arrondi du nombre passé en argument

floor(*number*) Retourne l'arrondi inférieur du nombre passé en argument

number(*object?*)

| Type | Contenu | Conversion |
|---|---|---|
| Chaîne | Expression numérique<br>Autre | selon IEEE 754<br>NaN |
| Booléen | Vrai<br>Faux | 1<br>0 |
| Ensemble de nœuds | Converti en chaîne, puis comme ci-dessus | |

sum(*node-set*) Retourne la somme des nœuds de l'ensemble de nœud passé en argument, après leur conversion numérique

## Opérations numériques

```
+
-
*
div
mod
```

Pour `-`, `div` et `mod`, il faut laisser un espace de part et d'autres de l'opérateur

# Ordre de parcourt d'un axe XPath



Le sens de parcourt d'un axe est significatif pour certaines opérations.

Tous les axes "remontants" ou "arrières" commencent au plus près du nœud contextuel.

Ainsi, les opérations numériques sur les nœuds sont sensibles au sens de parcours.

Sur un même ensemble de nœud, le premier nœud dans un sens de lecture "arrière" ou "remontant" est le dernier nœud dans un sens de lecture "avant" ou "descendant"

## Corrolaire

```
ancestor-or-self::node()[last()] = /*
```

```
ancestor-or-self::*[1] = self::*
```

(si le contexte est un élément)

```
ancestor::*[1] = parent::*
```

(sauf si on est sur le premier élément ou avant)

```
preceding-sibling::*[last()] = ../*[1]
```

Est-ce vrai pour tout les nœuds ?

# Comparaison d'ensembles de nœuds



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data>
      <a>
            <item>1</item>
            <item>2</item>
            <item>3</item>
            <item>4</item>
            <item>5</item>
      </a>
      <b>
            <item>2</item>
            <item>3</item>
            <item>4</item>
            <item>6</item>
            <item>7</item>
      </b>
</data>
```

Tests sur les *nodes*

- 2 ensembles de nœuds (*nodeset*) sont équivalents si la **valeur textuelle** de l'un de leur nœud est égal
- 2 ensembles de nœuds sont différents si la **valeur textuelle** de l'un de leur nœud est différente

→ donnée par la fonction `string()`

```
//a/item[2] = //b/item
//a/item[2] != //b/item
```

Sont 2 expressions vraies !!!

Les tests d'égalité et d'inégalité sur les nœuds doivent être compris comme des tests sur les ensembles

Un test d'**égalité** avec `NaN` retourne **faux**

Un test d'**inégalité** avec `NaN` retourne **vrai**

`NaN=NaN` est **faux** !

`NaN!=NaN` est **vrai** !

`x` est-il numérique ?

Si `number(x)!=number(x)` {
- est **vrai**, alors `x` n'est pas numérique
- est **faux**, alors `x` est numérique

## Variables

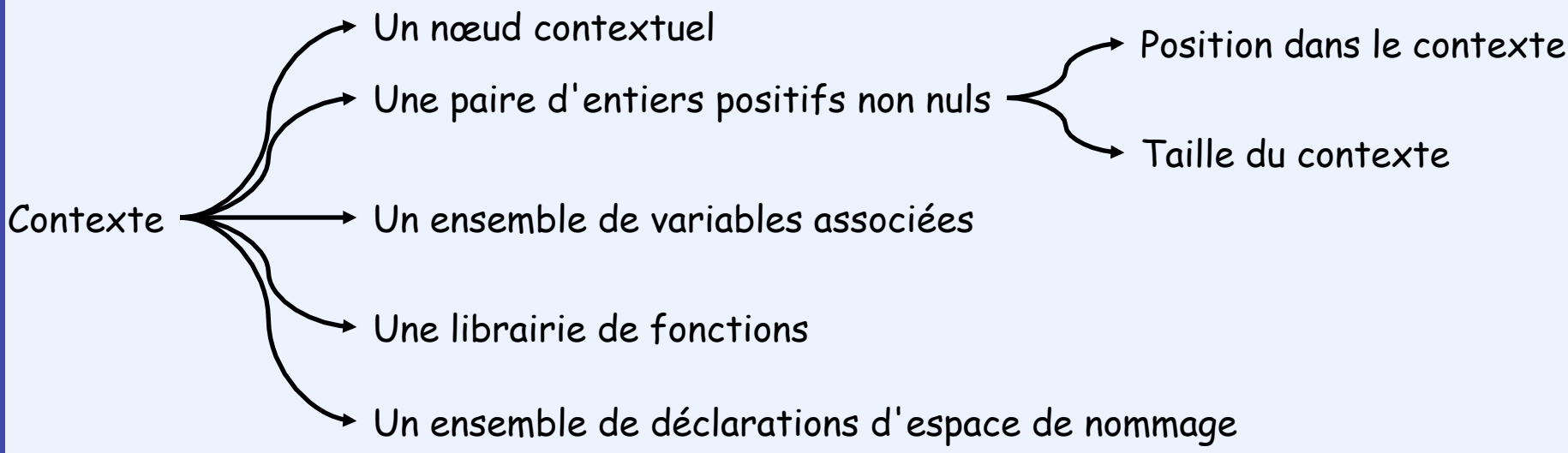
Les expressions XPath peuvent contenir des variables nommées dont les valeurs peuvent être n'importe quel objet. Les variables sont référencées par `$var`

```
//personnes[ @âge > $âge-du-capitaine ]
```

L'association de la valeur de la variable à son nom est donnée par le **contexte** d'évaluation de l'expression

Les applications qui utilisent XPath offrent les moyens nécessaires à la définition de ces variables.

## Contexte



Chaque application qui utilise XPath dispose d'un moyen propre nécessaire à la définition :

- des variables
- des librairies de fonctions supplémentaires
- de la résolution des préfixes

```
//geom:arc[ math:cos( @theta ) > $custom:limit ]/@xlink:href
```

contexte

Espaces de nommage :

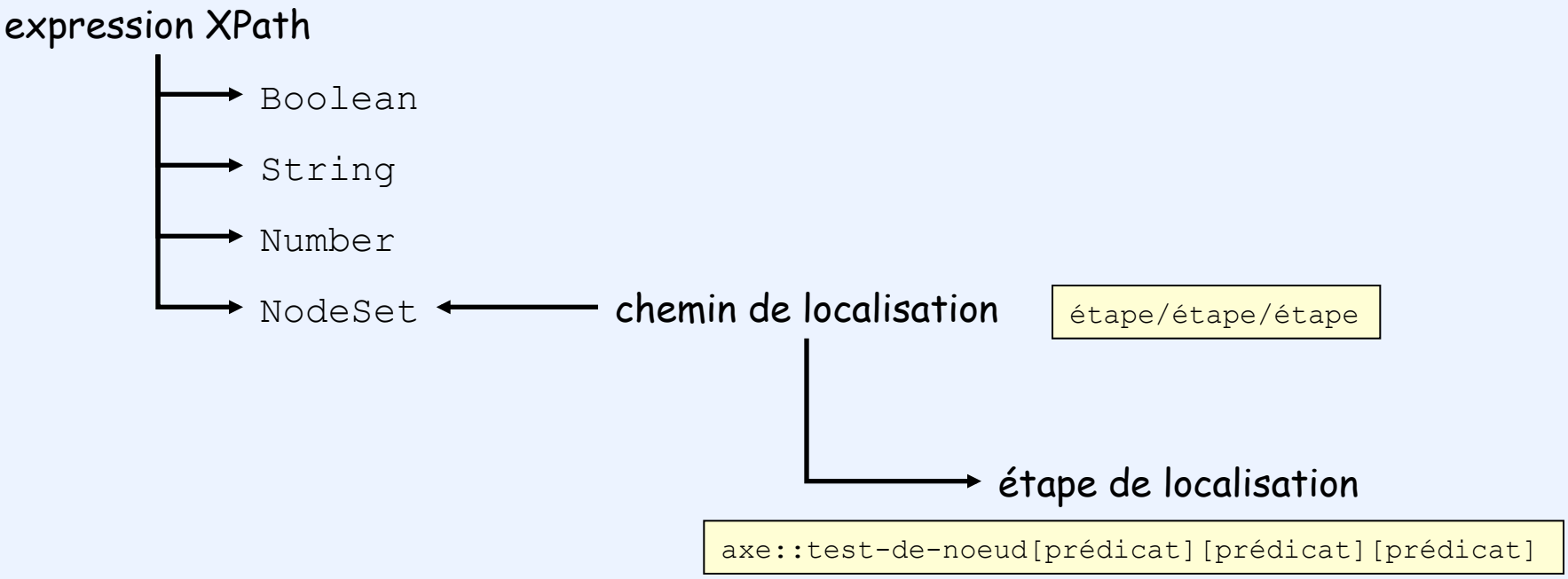
```
geom  ➔ http://www.foo.com/geometry
math  ➔ http://www.foo.com/maths
xlink ➔ http://www.w3.org/1999/xlink
custom ➔ http://www.bar.org/my-ns
```

Variables :

```
( http://www.bar.org/my-ns , limit ) ➔ 0,5
( http://www.foo.com/maths , pi )    ➔ 3,1415926535
( null , date )                      ➔ 10 juin 1969
```

Fonctions :

```
( http://www.foo.com/maths , cos ) ➔ cos( )
( http://www.foo.com/maths , sin ) ➔ sin( )
( http://www.foo.com/maths , tan ) ➔ tan( )
( null , ends-with )               ➔ ends-with( )
```


expression XPath
Boolean
String
Number
NodeSet
chemin de localisation
étape/étape/étape
étape de localisation
axe::test-de-noeud[prédicat][prédicat][prédicat]

- Interpréteur XPath : exemple avec Jaxen

```
XPath xpath = new DOMXPath("/foo:bar/@xlink:href", nav);
SimpleNamespaceContext nsContext = new SimpleNamespaceContext();
nsContext.addNamespace("foo", "http://www.foo.org/");
nsContext.addNamespace("xlink", "http://www.w3.org/1999/xlink");
xpath.setNamespaceContext(nsContext);
List result = contextpath.selectNodes(document);
```

jaxen

http://jaxen.org

- XSLT

```
<xsl:template match="/">
  <html>
    <body>
      <h1><xsl:value-of select="Cours/Titre"/></h1>
      <xsl:apply-templates select="Cours/Auteur"/>
      <xsl:apply-templates select="Cours/Description"/>
    </body>
  </html>
</xsl:template>
```

- Active Tags http://ns.inria.org/active-tags

```
<xcl:active-sheet xmlns:io="http://www.inria.fr/xml/active-tags/io"
                  xmlns:xcl="http://www.inria.fr/xml/active-tags/xcl">
  <xcl:parse-stylesheet name="xslt" source="file:///path/to/stylesheet.xsl"/>
  <xcl:for-each name="file"
          select="{ io:file('file:///path/to/base/dir/')/*[@io:extension='xml'] }">
    <xcl:parse name="document" source="{ $file }" style="stream"/>
    <xcl:transform source="{ $document }" stylesheet="{ $xslt }"
          output="file:///path/to/published/{ $file/@io:short-name }.html" />
  </xcl:for-each>
</xcl:active-sheet>
```

# XPath 2

XML Path Language (XPath) 2.0

http://www.w3.org/TR/xpath20/

XQuery 1.0 and XPath 2.0 Data Model

http://www.w3.org/TR/xpath-datamodel/

XQuery 1.0 and XPath 2.0 Functions and Operators

http://www.w3.org/TR/xpath-functions/

XQuery 1.0 and XPath 2.0 Formal Semantics

http://www.w3.org/TR/xquery-semantics/

XSL Transformations (XSLT) Version 2.0

http://www.w3.org/TR/xslt20/