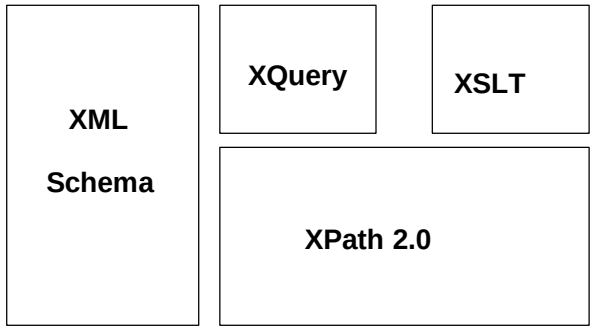
XQUERY

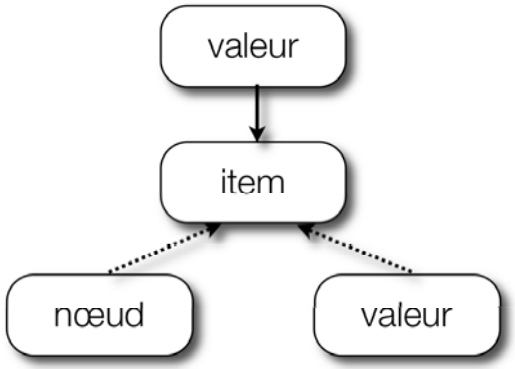
- Principe
  - ✓ XQuery est un langage de "programmation" puissant pour extraire des données XML
  - ✓ type de données: un seul "document" ou encore des collections sous forme de:
    - fichiers
    - bases de données XML
    - XML "en mémoire" (arbres DOM)
  - ✓ permet de faire des requêtes selon la structure ou encore les contenus en se basant sur des expressions Xpath (version 2.0)
  - ✓ peut générer des nouveaux documents (autrement dit: on peut manipuler un résultat obtenu et y ajouter)
  - ✓ ne définit pas les mises à jour (équivalent de update/insert de SQL), à venir…
- Résumé:
  - ✓ XQuery permet d'extraire des fragments XML, d'y effectuer des recherches et de générer des fragments XML

XQUERY (2)

- Sa place dans l'écosystème XML



- Le modèle de données XQuery
  - ✓ une valeur est une séquence ordonnée d'items (forêt)
  - ✓ un item est un nœud ou une valeur atomique
  - ✓ chaque nœud et chaque valeur a un type



SÉQUENCES

- Pas de distinction entre un item et une séquence de longueur 1: 47=(47)
- Une séquence peut contenir des valeurs hétérogènes: (1,“toto”,<toto/>)
- Pas de séquences imbriqués: (1,(2,6),"toto",<toto/>) = (1,2,6,“toto”,<toto/>)
- Une séquence peut être vide: ()
- Les séquences sont ordonnées: (1,2) Ç (2,1)

EXPRESSIONS XQUERY

- Une requête XQuery est une composition d'expressions
- Chaque expression a une valeur ou retourne une erreur
- Les expressions n'ont pas d'effets de bord (par exemple, pas de mise-à-jour)
- Expressions (requêtes) simples:
  - ✓ valeurs atomiques: 46, ‘‘Salut’’
  - ✓ valeurs construites: true(), date(‘‘2007-03-19’’)
- Expressions complexes
  - ✓ expressions de chemins (XPath 2.0): FILM//ACTEUR
  - ✓ expressions FLWOR (for-let-where-order-return)
  - ✓ tests (if-then-return-else-return)
  - ✓ fonctions: racines, XPath, utilisateurs

EXEMPLE

```
<!ELEMENT bib (book*)>
<!ELEMENT book ((author)+,publisher,price)>
<!ATTLIST book year CDATA #IMPLIED title CDATA #REQUIRED>
<!ELEMENT author (la,fi)>
<!ELEMENT la (#PCDATA)>
<!ELEMENT fi (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

```
<bib>
    <book year="2000" title="Data on the Web">
        <author><la>Abiteboul</la><fi>S.</fi></author>
        <author><la>Buneman</la><fi>P.</fi></author>
        <author><la>Suciu</la><fi>D.</fi></author>
        <publisher>Morgan Kaufmann
        Publishers</publisher>
        <price>39.95</price>
    </book>
</bib>
```

EXPRESSIONS DE CHEMIN

- Expression: document("bib.xml")//author
- Résultat:

  <author><la>Abiteboul</la><fi>S.</fi></author>,
  <author><la>Buneman</la><fi>P.</fi></author>,
  <author><la>Suciu</la><fi>D.</fi></author>

- Expression: doc(“bib.xml”)/bib/book/author
- Résultat:

- Expression: doc(“bib.xml”)/bib//book[1]/publisher
- Résultat:

CONSTRUCTIONS DE NŒUDS XML

- Le nom de l'élément est connu, le contenu est calculé par une expression
- Requête:

```
<auteurs>
    {document("bib.xml")//book[1]/author/la}
</auteurs>
```

- Résultat:

```
<?xml version="1.0"?>
<auteurs>
    <la>Abiteboul</la>
    <la>Buneman</la>
    <la>Suciu</la>
</auteurs>
```

CONSTRUCTIONS DE NŒUDS XML (2)

- Le nom et le contenu sont calculés:
  - ✓ element {expr-nom} {expr-contenu}
  - ✓ attribute {expr-nom} {expr-contenu}
- Requête:

```
element{document("bib.xml")//book[1]/name(@*[2])}{  attribute{document("bib.xml")//book[1]/name(*[4])}{
    document("bib.xml")//book[1]/*[4]
}
}
```

- Résultat:

```
<title publisher="Morgan Kaufmann Publishers"/>
```

DIFFÉRENCE DE SÉQUENCES DE NŒUDS

- Requête:

  <livre>

  Tous les sous-éléments sauf les auteurs: {document("bib.xml")//book[1]/(* except author)}

  </livre>

- Résultat:

  <livre>

  Tous les sous-éléments sauf les auteurs:

  <publisher>Morgan Kaufmann Publishers</publisher>

  <price>39.95</price>

  </livre>

CONCATÉNATION DE SÉQUENCES

- Requête:

```
<livre>
    Le prix suivi des auteurs: {document("bib.xml")//book[1]/(price,author)}
</livre>
```

- Résultat:

```
<livre>
    Le prix suivi des auteurs:
    <price>39.95</price>
    <author><la>Abiteboul</la><fi>S.</fi></author>
    <author><la>Buneman</la><fi>P.</fi></author>
    <author><la>Suciu</la><fi>D.</fi></author>
</livre>
```

- Remarque: on a changé l'ordre des nœuds (union)

FLOWR (PRONONCER *FLOWER*)

- FLWOR = "For-Let-Where-Order-Return"
  - rappelle l'idée du select-from-where de SQL
- Format d'une requête
  - **for** $<var> in <forest> [, $<var> in <forest>]+
    - itération sur une liste de collections xml
  - **let** $<var> := <subtree>
    - assignation du résultat d'une expression à une variable
  - **where** <condition>
    - élagage avec une sélection
  - **return** <result>
    - construction de l'expression à retourner
- Les forêts sont soit des collections, soit sélectionnées par des XPath
- Le résultat est une forêt: un ou plusieurs arbres

FOR, LET

- La clause **for $variable in expression** affecte la variable **$variable** successivement avec chaque item dans la séquence retournée par expression
- La clause **let $variable := expression** affecte la variable **$variable** avec la séquence "entière" retournée par expression
- Exemple

```
for $b in document("bib.xml")//book[1]
let $al := $b/author
    return <livre nb_auteurs="{count($al)}"> {$al} </livre>
```

- Résultat

```
<livre nb_auteurs="3">
    <author><la>Abiteboul</la><fi>S.</fi></author>
    <author><la>Buneman</la><fi>P.</fi></author>
    <author><la>Suciu</la><fi>D.</fi></author>
</livre>
```

WHERE

- La clause **where expression** permet de filtrer le résultat par rapport au résultat booléen de l'expression **expression** (= prédicat dans l'expression de chemin)
- Requête:

```
<livre>
        {for $a in document("bib.xml")//book
        where $a/author[1]/la eq "Abiteboul"
    return $a/@title }
</livre>
```

- Résultat:

```
<?xml version="1.0"?>
    <livre title="Data on the Web"/>
```

ORDER

- La clause **order by** permet de trier les résultats
- Requête

  for $t in //book/author/la order by $t return $t

- Un exemple plus complet

  for $t in document("bib.xml")//book let $n := count($t/author)

  where ($n > 1) order by $n

  return <result> {$t/@title} possède {$n} auteurs </result>

RETURN

- La clause **return** construit l'expression à retourner à chaque iteration
  - attention: chaque iteration doit retourner un seul fragment XML (pas une collection)
- Correct

```
for $t in document("bib.xml")//book   let $n := count($t/author)
return <result>
        {$t/@title} possède {$n} auteurs
</result>
```

- Incorrect

```
for $t in document("bib.xml")//book
let $n := count($t/auteurs)
return $t/@title possède $n auteurs
```

EXEMPLE

- Sélection: lister les noms et téléphones des restaurants de Cabourg

- Résultat

```
<result>
<titre>Restaurants de Cabourg</titre>
{for $R in document("Guide")/Restaurant
where $R//Ville = "Cabourg"
return <Restaurant>
    <Nom>{$R/Nom}</Nom>
    <Tel>{$R/Téléphone}</Tel>
</Restaurant>}
</result>
```

```
<Guide Version= "2.0">
<Restaurant type="francais" categorie="***">
   <Nom>Le Moulin</Nom>
   <Adresse>
      <Rue>des Vignes</Rue>
      <Ville>Mougins</Ville>
   </Adresse>
   <Manager>Dupuis</Manager>
</Restaurant>
<Restaurant type="francais" categorie ="**">
   <Nom>La Licorne</Nom>
   <Adresse>
      <Rue>Des Moines</Rue>
      <Ville>Paris</Ville>
   </Adresse>
   <Téléphone>0148253278</Téléphone>
   <Manager>Dupuis</Manager>
</Restaurant>
<Bar type = "anglais">
   <Nom>Rose and Crown</Nom>
</Bar>
</Guide>
```

JOINTURE

- Jointure: Lister le nom des Restaurants avec téléphone dans la rue de l'Hôtel Lutecia
- Résultat

```
for   $R in collection("Guide")/Restaurant,
      $H in document(Répertoire)/Hotel
where     $H//Rue = $R//Rue
          AND $H//Nom= "Le Lutecia"
return
   <Result>
      <Nom>{$R/Nom}</Nom>
      <Tel>{$R/Téléphone}</Tel>
   </Result>
```

```
<Guide Version= "2.0">
<Restaurant type="francais" categorie="***">
   <Nom>Le Moulin</Nom>
   <Adresse>
      <Rue>des Vignes</Rue>
      <Ville>Mougins</Ville>
   </Adresse>
   <Manager>Dupuis</Manager>
</Restaurant>
<Restaurant type="francais" categorie ="**">
   <Nom>La Licorne</Nom>
   <Adresse>
      <Rue>Des Moines</Rue>
      <Ville>Paris</Ville>
   </Adresse>
   <Téléphone>0148253278</Téléphone>
   <Manager>Dupuis</Manager>
</Restaurant>
<Bar type = "anglais">
   <Nom>Rose and Crown</Nom>
</Bar>
</Guide>
```

AGRÉGAT

- Combien de restaurants y-a-t-il dans les guides?

- Résultat

```
<result>
    <entete>Nombretotal de restaurants</entete>
    {let$R := collection(“Guide”)/Restaurant
    return <NombreRestaurant>
        {count ($R)}
    </NombreRestaurant>}
</result>
```

XQUERY ET XSLT

- XSLT est un peu plus verbeux que XQuery
  - c'est du XML
  - c'est un peu moins vrai en XSLT 2.0
- XSLT est un système basé sur deux langages: XPath pour les expressions et XSLT pour les instructions
- Le fait que XSLT soit en XML présente plusieurs avantages
  - réutilisation de tous les outils XML
  - la syntaxe est extensible
  - liens avec les autres langages XML (XSD,…)
- Le résultat d'une requête XSLT est un (ou plusieurs) document(s) XML, alors que cela peut être un nombre pour une requête XQuery
- En XQuery, il n'y a pas de apply-templates, donc la récursivité doit être programmée à la main
- XQuery est plus modulaire que XSLT (l'import dans XSLT est assez peu puissant)

XQUERY ET XSLT (2)

- On peut facilement transformer presque toutes les expresssions FLWR en équivalent XSLT

Construction XQuery	Équivalent XSLT
for $var in SEQ	<xsl:for-each select=”SEQ”> <xsl:variable name=”var” select=”.”/>
let $var := SEQ	<xsl:variable name="var" select="SEQ"/>
where CONDITION	<xsl:if test=”CONDITION”>
order by $x/VALUE	<xsl:sort select=”VALUE”>

XQUERY ET XSLT (3)

- Soit la requête suivante (extraite du benchmark XMark)

```
for $b in doc("auction.xml")/site/regions//item   let $k := $b/name
order by $k
return <item name="{$k}">{ $b/location } </item>
```

- Son équivalent XSLT

```
<xsl:for-each select="doc('auction.xml')/site/regions//item">
 <xsl:sort select="name"/>
 <item name="{name}"
        <xsl:value-of select="location"/>
 </item>
</xsl:for-each>
```

XQUERY ET XSLT (4)

- Soit la requête suivante: liste des éditeurs qui ont publié plus de 100 livres

```
<big_publishers>
  FOR $p IN distinct(document("bib.xml")//publisher)
  LET $b := document("bib.xml")/book[publisher = $p]
  WHERE count($b) > 100
  RETURN $p
</big_publishers>
```

- Son équivalent XSLT

```
<big_publishers xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:for-each select="document('bib.xml')//publisher[not(.=preceding::publisher)]">
    <xsl:variable name="b" select="document('bib.xml')/book[publisher=current()]"/>
    <xsl:if test="count($b) > 100">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:for-each>
</big_publishers>
```

XQUERY ET XSLT (4)

Soit la requête suivante: liste des éditeurs qui ont publié plus de 100 livres

```
<big_publishers>
  FOR $p IN distinct(document("bib.xml")//publisher)
   LET $b := document("bib.xml")/book[publisher = $p]
   WHERE count($b) > 100
   RETURN $p
</big_publishers>
```

Son équivalent XSLT

```
<big_publishers xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:for-each select="document('bib.xml')//publisher[not(.=preceding::publisher)]">
   <xsl:variable name="b" select="document('bib.xml')/book[publisher=current()]"/>
   <xsl:if test="count($b) > 100">
     <xsl:copy-of select="."/>
   </xsl:if>
 </xsl:for-each>
</big_publishers>
```

XQUERY ET XSLT (4)

Soit la requête suivante: liste des éditeurs qui ont publié plus de 100 livres

```
<big_publishers>
  FOR $p IN distinct(document("bib.xml")//publisher)
  LET $b := document("bib.xml")/book[publisher = $p]
  WHERE count($b) > 100
  RETURN $p
</big_publishers>
```

Son équivalent XSLT

```
<big_publishers xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:for-each select="document('bib.xml')//publisher[not(.=preceding::publisher)]">
   <xsl:variable name="b" select="document('bib.xml')/book[publisher=current()]"/>
   <xsl:if test="count($b) > 100">
     <xsl:copy-of select="."/>
   </xsl:if>
 </xsl:for-each>
</big_publishers>
```

XQUERY ET XSLT (4)

Soit la requête suivante: liste des éditeurs qui ont publié plus de 100 livres

```
<big_publishers>
  FOR $p IN distinct-document("bib.xml")//publisher)
  LET $b := document("bib.xml")/book[publisher = $p]
   WHERE count($b) > 100
   RETURN $p
</big_publishers>
```

Son équivalent XSLT

```
<big_publishers xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:for-each select="document('bib.xml')//publisher[not(.=preceding::publisher)]">
  <xsl:variable name="b" select="document('bib.xml')/book[publisher=current()]"/>
   <xsl:if test="count($b) > 100">
    <xsl:copy-of select="."/>
   </xsl:if>
  </xsl:for-each>
</big_publishers>
```

XQUERY ET XSLT (4)

Soit la requête suivante: liste des éditeurs qui ont publié plus de 100 livres

```
<big_publishers>
  FOR $p IN distinct(document("bib.xml")//publisher)
  LET $b := document("bib.xml")/book[publisher = $p]
  WHERE count($b) > 100
   RETURN $p
</big_publishers>
```

Son équivalent XSLT

```
<big_publishers xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:for-each select="document('bib.xml')//publisher[not(.=preceding::publisher)]">
  <xsl:variable name="b" select="document('bib.xml')/book[publisher=current()]"/>
  <xsl:if test="count($b) > 100">
    <xsl:copy-of select="."/>
   </xsl:if>
 </xsl:for-each>
</big_publishers>
```

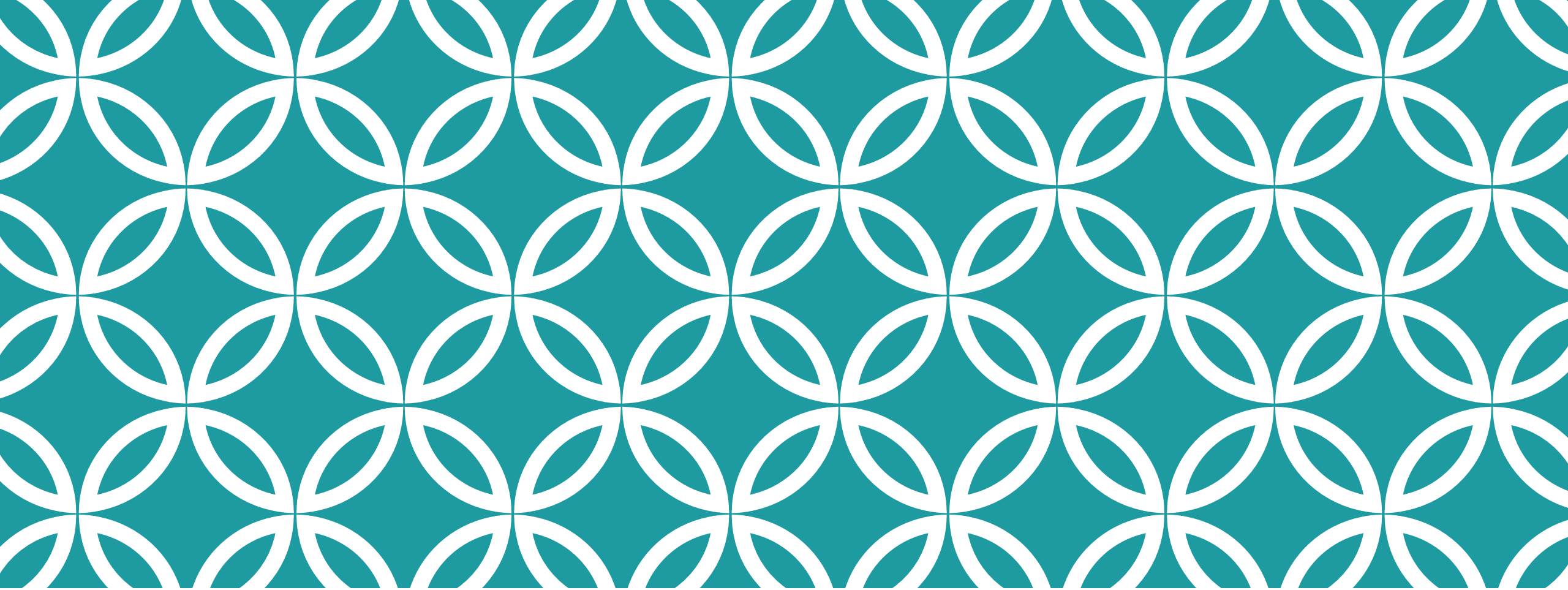
XQUERY ET XSLT (4)

Soit la requête suivante: liste des éditeurs qui ont publié plus de 100 livres

```
<big_publishers>
 FOR $p IN distinct(document("bib.xml")//publisher)
 LET $b := document("bib.xml")/book[publisher = $p]
 WHERE count($b) > 100
 RETURN $p
</big_publishers>
```

Son équivalent XSLT

```
<big_publishers xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:for-each select="document('bib.xml')//publisher[not(.=preceding::publisher)]">
  <xsl:variable name="b" select="document('bib.xml')/book[publisher=current()]"/>
  <xsl:if test="count($b) > 100">
   <xsl:copy-of select="."/>
  </xsl:if>
 </xsl:for-each>
</big_publishers>
```



PARSEUR XML

- Dom
- SAX

ANALYSE XML

DOM :

- DOM levels, DOM level 1
- Principes de l'API
- Objets DOM
- Traitement des blancs
- Navigation, parcours, et mise à jour de l'arbre
- Attributs et entités dans le DOM
- Héritage des objects, hierarchie des nœuds Clonage des nœuds, échange des nœuds
- Les espaces de nommage dans le DOM
- Spécialisation des APIs DOM

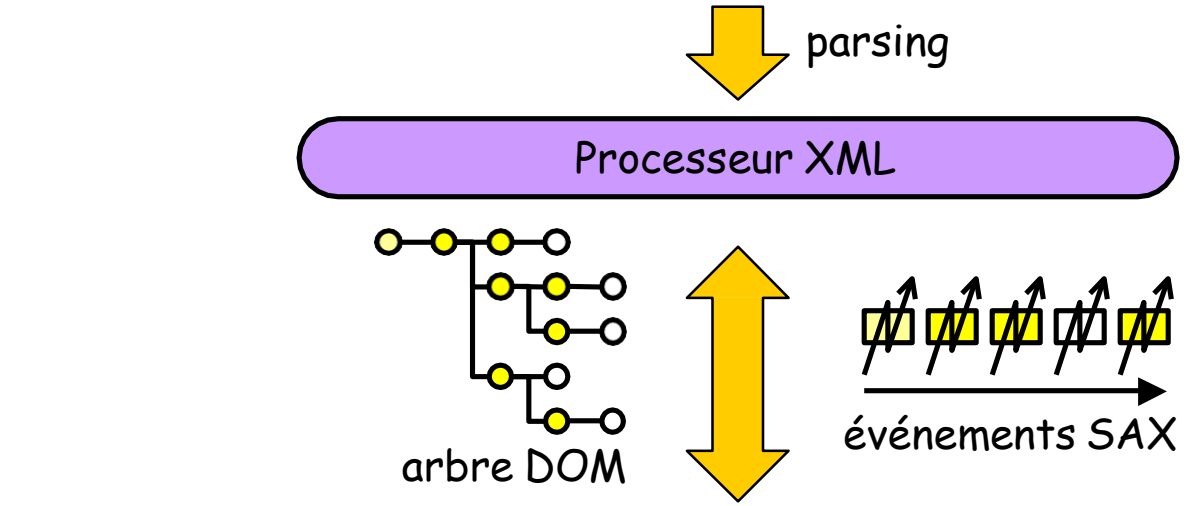
SAX:

- SAX levels
- Comment fonctionne SAX
- Principaux handlers L'interface
- ContentHandler Enregistrement d'un handler
- Exemple
- Evénements caractères
- Filtres et pipelines SAX
- Analyseurs SAX validants
- Les espaces de nommage dans SAX

ANALYSEUR XML

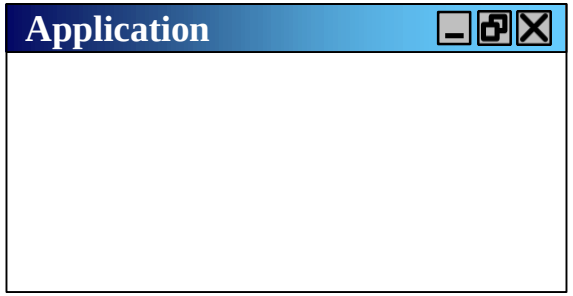
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Cours SYSTEM "Cours.dtd">
<Cours>
    <Titre>Cours XML</Titre>
    <Auteur>        <Nom>Poulard</Nom>
                    <Prénom>Philippe</Prénom>
    </Auteur>
    <Description>Ce cours aborde les concepts
         de base mis en œuvre dans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

Structure physique



Modèle logique (Infoset)

Accède au document grâce aux APIs DOM ou SAX

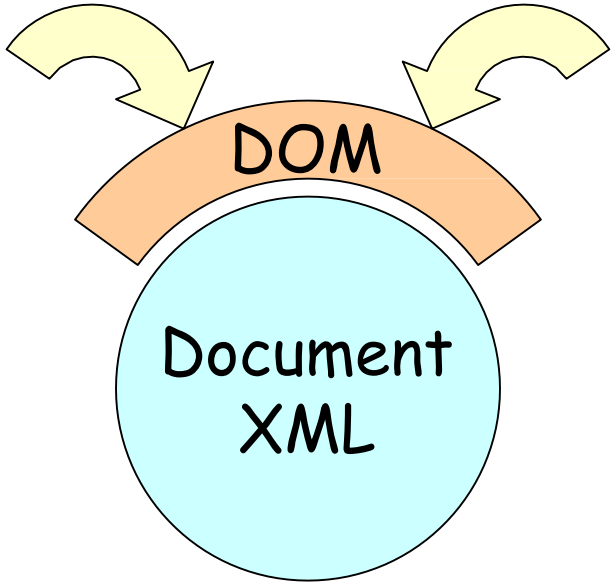


Application

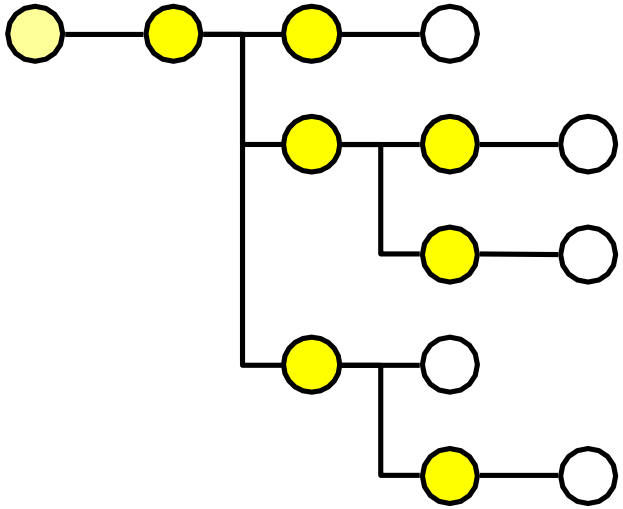
Document Object Model

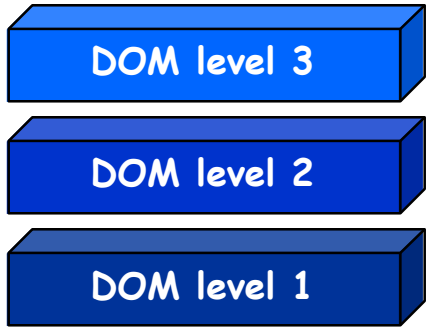
DOM, norme du W3C

Modèle Object du Document :
une API pour **accéder** et **agir** sur
le contenu et la structure d'un document



Le document est vu comme un arbre
Sa représentation ne l'est pas nécessairement
(DOM spécifie une API, pas une implémentation)



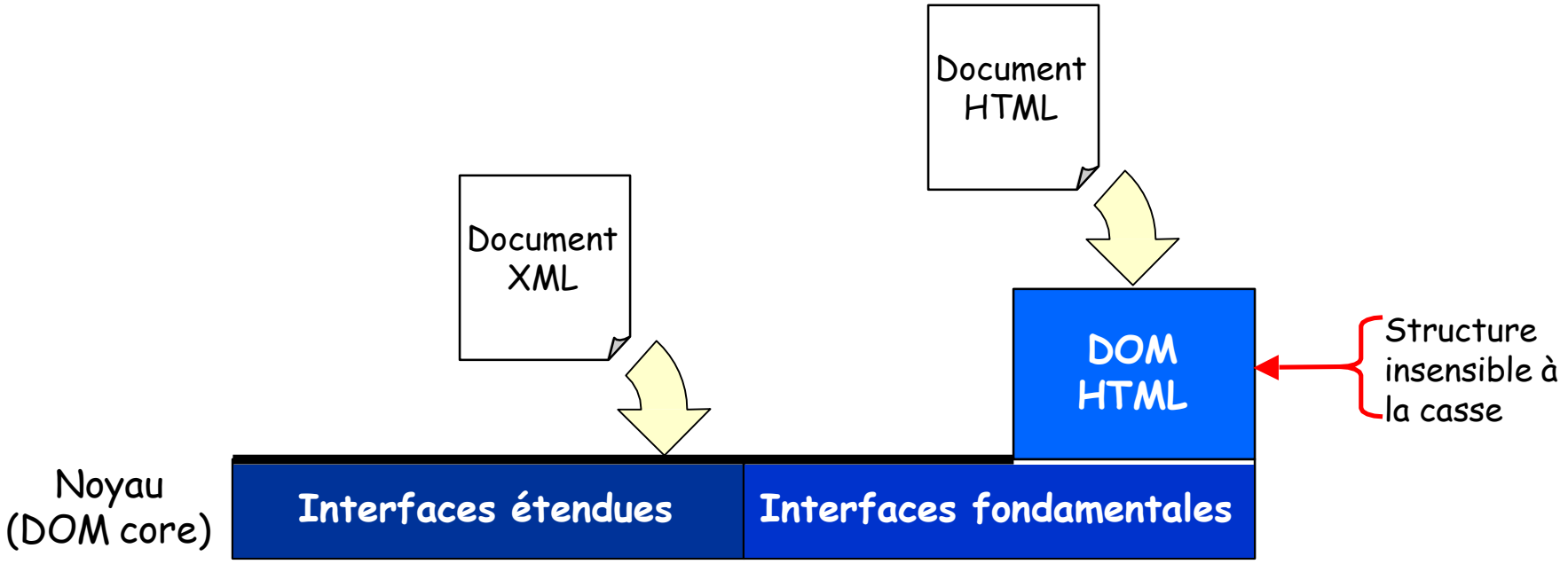


Schémas, XPath, entrées/sorties

Namespaces, vues, événements, styles, parcours

Noyau, HTML

Pas de compatibilité binaire entre différentes implémentations



- CDATASection
- DocumentType
- Notation
- Entity
- EntityReference
- ProcessingInstruction

- Document
- DocumentFragment
- Element
- Attr
- Comment
- DOMImplementation
- Node
- NodeList
- NamedNodeMap
- CharacterData
- Text
- ***DOMException*** ← Classe

API DOM

- Le modèle d'objet spécifié par le W3C défini 12 types de nœuds différents.
- Le modèle d'objet de document fourni tout une panoplie d'outils destinés à construire et manipuler un document XML. Pour cela, le DOM met à disposition des interfaces, des méthodes et des propriétés permettant de gérer l'ensemble des composants présents dans un document XML.
- Le DOM spécifie diverses méthodes et propriétés permettant notamment, de créer (createNode…), modifier (replaceChild…), supprimer (remove…) ou d'extraire des données (get…) de n'importe quel élément ou contenu d'un document XML.
- De même, le DOM définit les types de relation entre chaque noeud, et des directions de déplacement dans une arborescence XML. Les propriétés parentNode, childNodes, firstChild, lastChild, previousSibling et nextSibling permettent de retourner respectivement le père, les enfants, le premier enfant, le dernier enfant, le frère précédent et le frère suivant du noeud courant.
- Le modèle d'objet de document offre donc au programmeur les moyens de traiter un document XML dans sa totalité

PRINCIPES DE L'API DOM

- Vues héritées → Hiérarchie d'objet
- Vues aplaties → Tout est `Node`

Exemple

```
Element.tagName
=
Node.nodeName
```

Si le nœud est un élément

Les implémentations sont libres d'utiliser ou non des accesseurs (`get`) et des mutateurs (`set`) pour protéger les propriétés

MSXML →
```
theName = node.nodeName;
```
Java →
```
theName = node.getNodeName();
```

DOM spécifie une API à minima. Les implémentations sont libres d'étendre les fonctionnalités

MSXML →
```
resultHTML = source.transformNode(stylesheet);
```

DOM level 2

Introduction des espaces de nommage : méthodes postfixées par `NS`

```
node.setAttributeNS(null, attr, value);
```
→ Default namespace

OBJETS DOM

Node Représente un nœud de l'arborescence d'un document XML.

Propriétés :

attributes, childNodes, firstChild, lastChild, localName, namespaceURI, nextSibling, nodeName, nodeType, nodeValue, ownerDocument, parentNode, prefix, previousSibling

Méthodes :

appendChild, cloneNode, hasAttributes, hasChildNodes, insertBefore, isSupported, normalize, removeChild, replaceChild

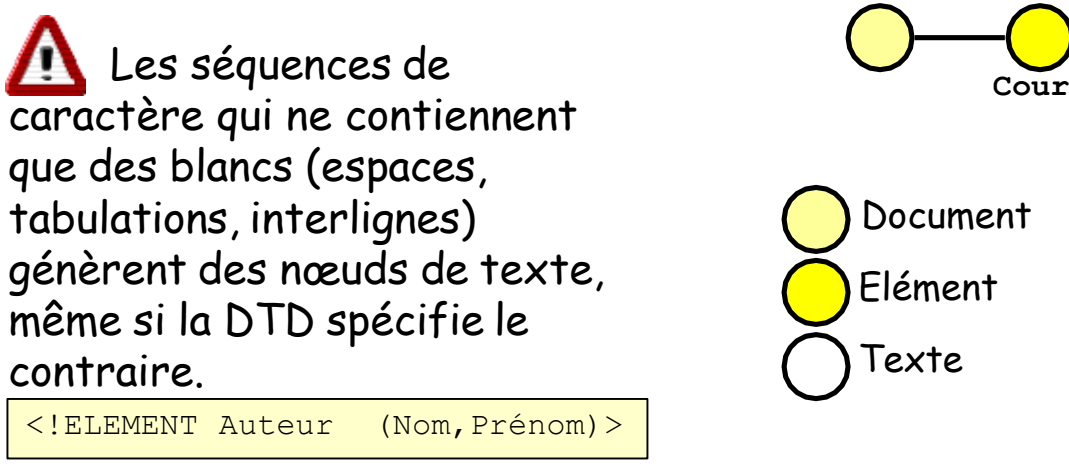
L'interface `Node` défini des constantes qui permettent de connaître le type du noeud

nodeType	nodeName	nodeValue
ELEMENT_NODE = 1	Nom d'élément	null
ATTRIBUTE_NODE = 2	Nom de l'attribut	Valeur de l'attribut
TEXT_NODE = 3	"#text"	Contenu du nœud du texte
CDATA_SECTION_NODE = 4	"#cdata-section"	Contenu de la section CDATA
ENTITY_REFERENCE_NODE = 5	Nom de l'entité référencée	null
ENTITY_NODE = 6	Nom de l'entité	null
PROCESSING_INSTRUCTION_NODE = 7	Cible	Contenu complet, sans la cible
COMMENT_NODE = 8	"#comment"	Contenu du commentaire
DOCUMENT_NODE = 9	"#document"	null
DOCUMENT_TYPE_NODE = 10	Nom du type de document	null
DOCUMENT_FRAGMENT_NODE = 11	"#document-fragment"	null
NOTATION_NODE = 12	Nom de notation	null

`nodeType` permet d'obtenir le type de nœud

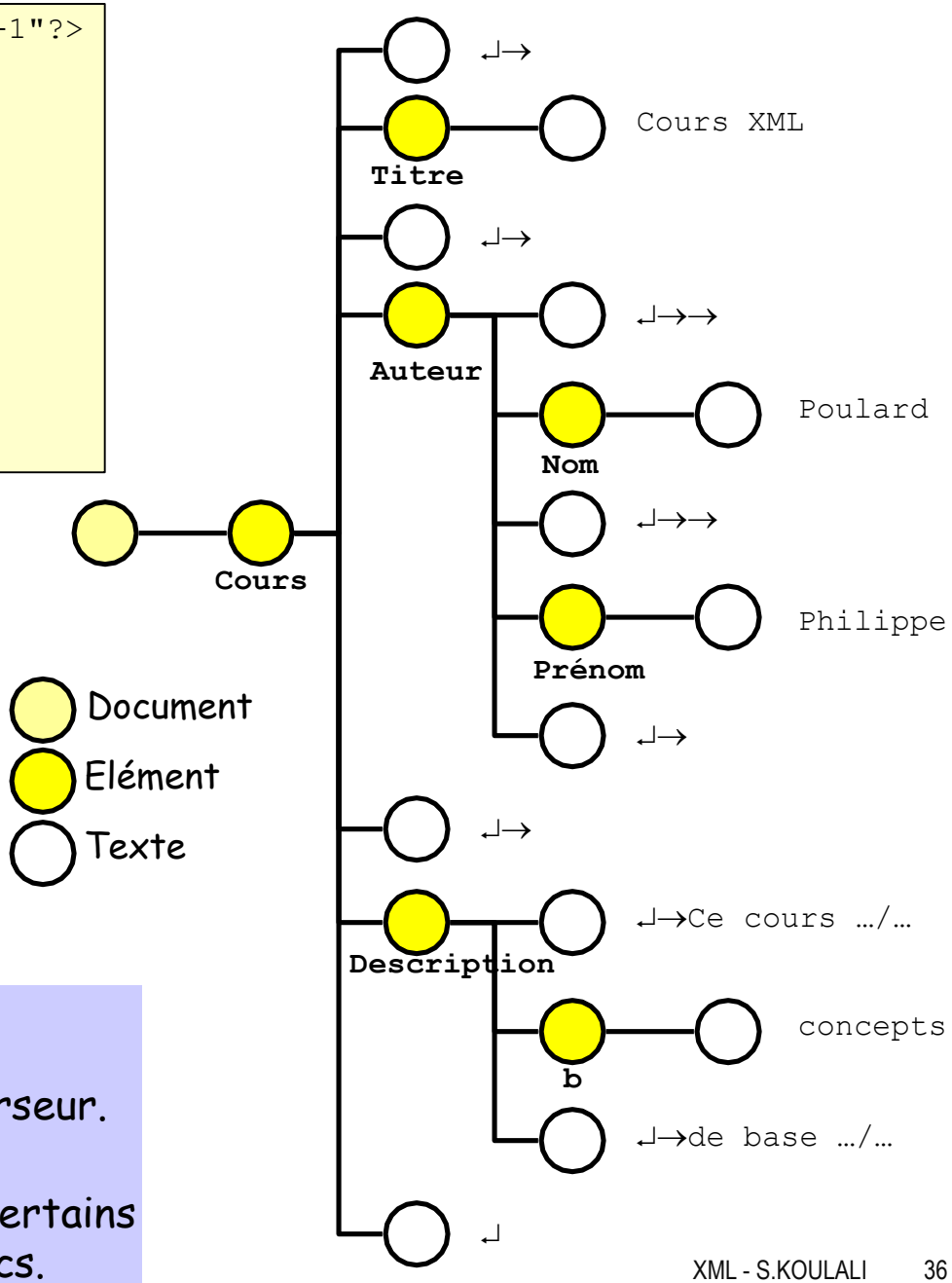
LES BLANS

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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.↵
</Cours></Description>↵
```



Les séquences de caractère qui ne contiennent que des blancs (espaces, tabulations, interlignes) génèrent des nœuds de texte, même si la DTD spécifie le contraire.

```
<!ELEMENT Auteur   (Nom,Prénom)>
```



Il est possible d'infléchir ce comportement grâce aux options du parseur.

Le comportement par défaut de certains parseurs est d'éliminer les nœuds blancs.