# Le langage XPath

# À quoi sert XPath ?

Le langage permet de désigner un ou plusieurs nœuds dans un document XML, à l'aide **d'expressions de chemin**. Exemples :

- *Extraction de valeurs*

```
<xsl:value-of select="SALLE/@NO">
```

- *Prédicats de test*

```
 <xsl:if test=" ($titre = ” or
TITRE = $titre)
and ($seance = ” or HEURE &gt;=
$seance)
and ($ville = ” or VILLE =
$ville)">
```

# Les arbres XPath

Un typage simplifié par rapport à celui de DOM

- **Document** ;
- **Element** ;
- **Text** ;
- **Attribut** ;
- **ProcessingInstruction** ;
- **Comment**.

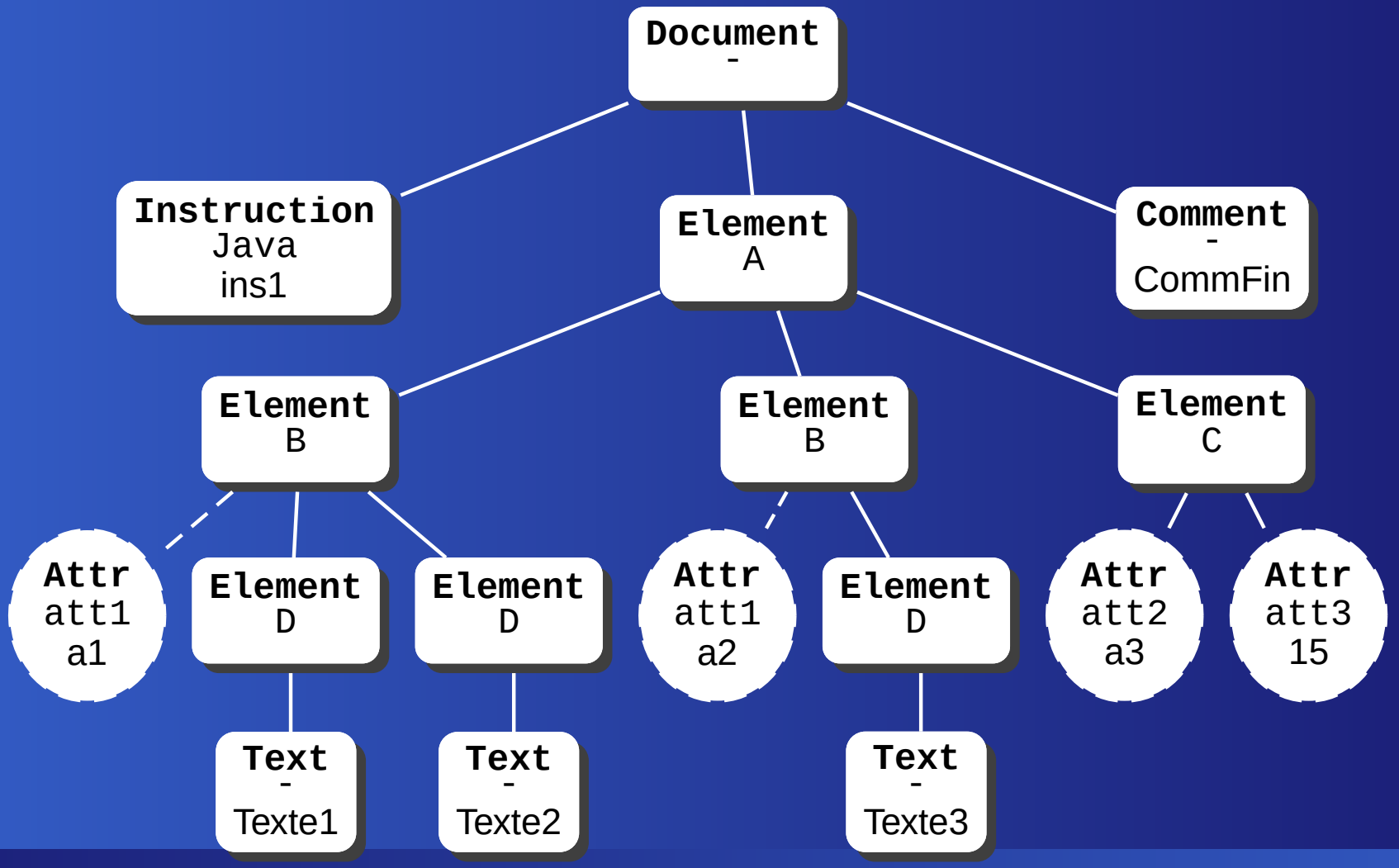
$\Rightarrow$ pas d'entité, pas de section littérale

# Expressions XPath

Une expression XPath :

- s'évalue en fonction d'un **nœud contexte**
- désigne un ou plusieurs chemins dans l'arbre à partir du nœud contexte
- a pour résultat
  - $\Rightarrow$ un ensemble de nœuds
  - $\Rightarrow$ ou une valeur, numérique, booléeenne ou alphanumérique

# Exemple de référence



# Chemins XPath

Un chemin XPath est une suite **d'étapes** :

$$[/]étape_1/étape_2/.../étape_n$$

Deux variantes :

- Un chemin peut être **absolu** :
  `/A/B/@att1`
  Le nœud contexte est alors la racine du document
- ou **relatif**
  `A/B/@att1`

# Étapes XPath

Une étape : trois composants

*axe*`::`*filtre*`[`*prédicat1*`][`*prédicat2*`]`...

- *l'axe* : sens de parcours des nœuds
- *le filtre* : type des nœuds qui seront retenus
- *le(s) prédicat(s)* : propriétés que doivent satisfaire les nœuds retenus

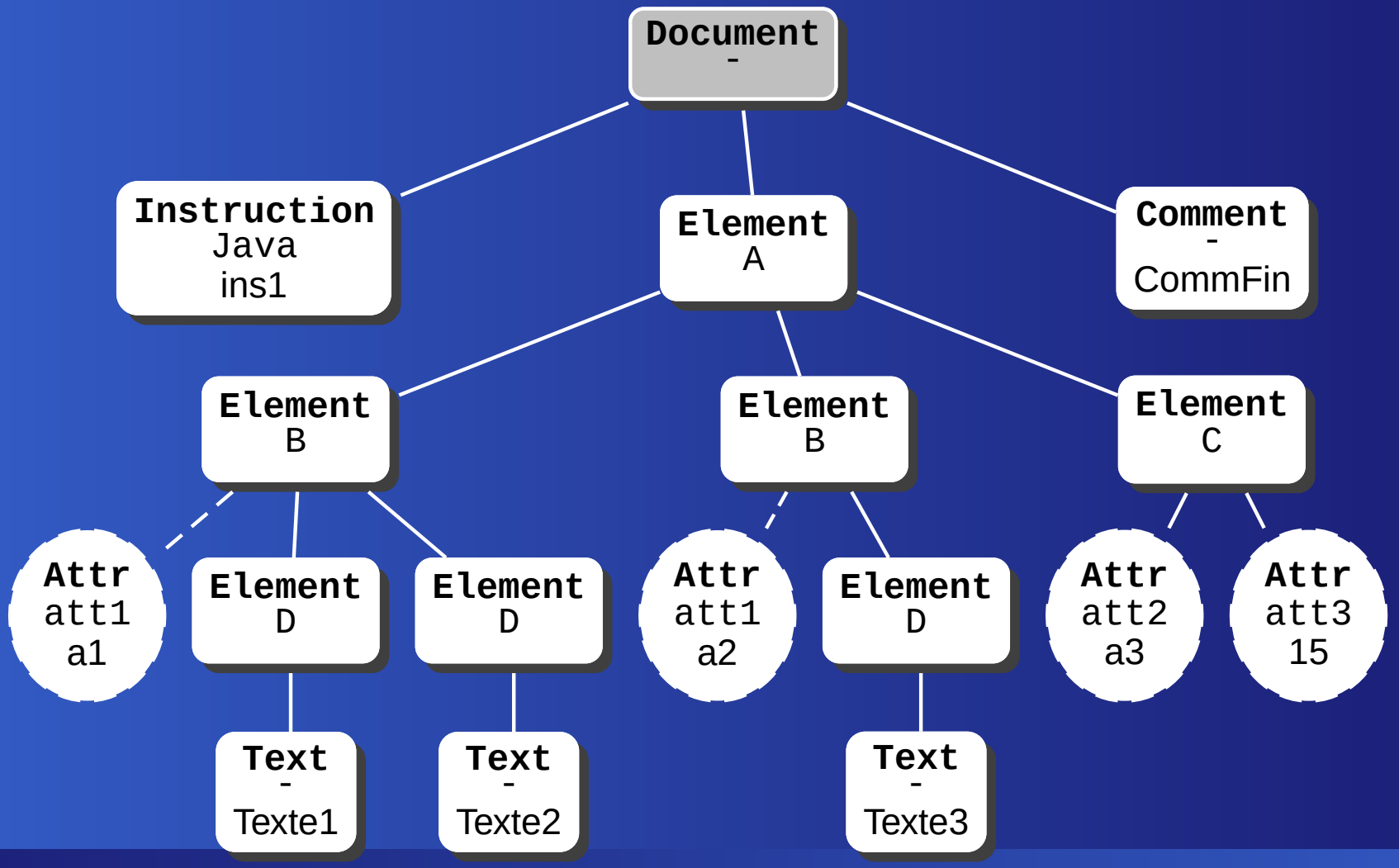
On peut faire une union de chemins :

```
//A | B/@att1
```

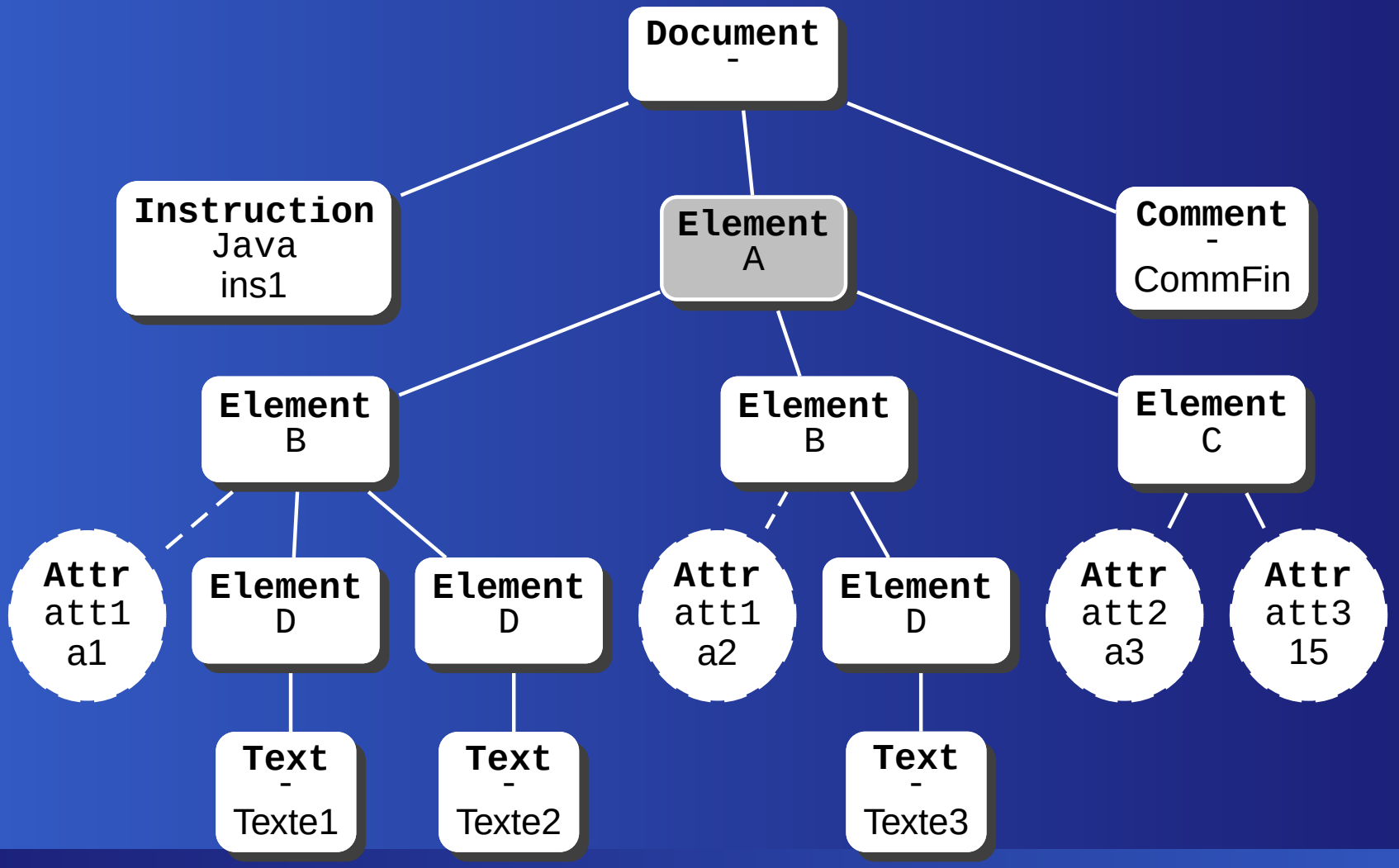
# Évaluation d'une expression XPath

- à partir du nœud contexte, on évalue l'étape 1 ; on obtient un ensemble de nœuds ;
- on prend alors, un par un, les nœuds de cet ensemble, *et on les considère chacun à leur tour comme nœud contexte pour l'évaluation de l'étape 2* ;
- à chaque étape, on prend successivement comme nœud contexte chacun des nœuds faisant partie du résultat de l'étape précédente.

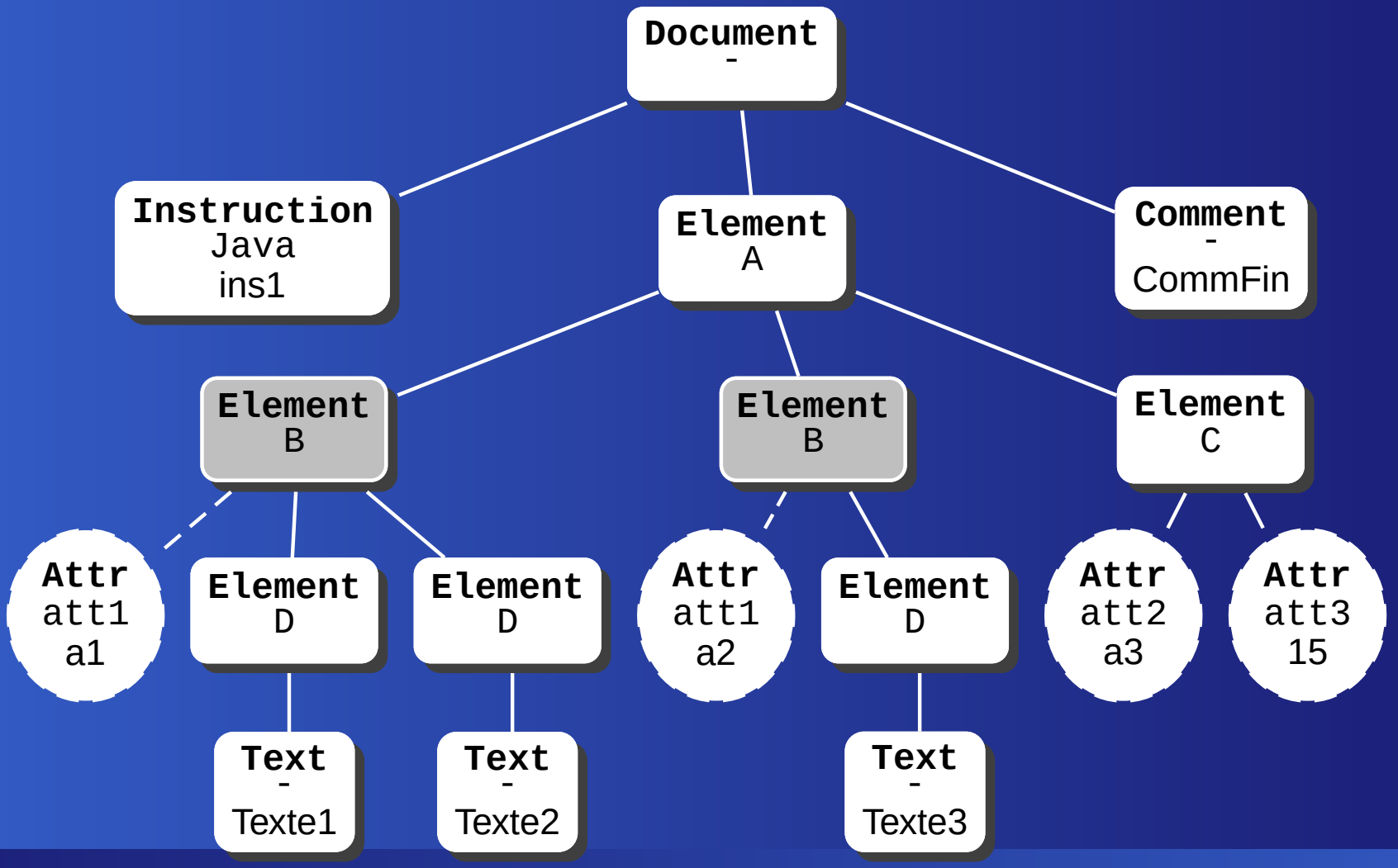
# /A/B/@att1 : nœud initial



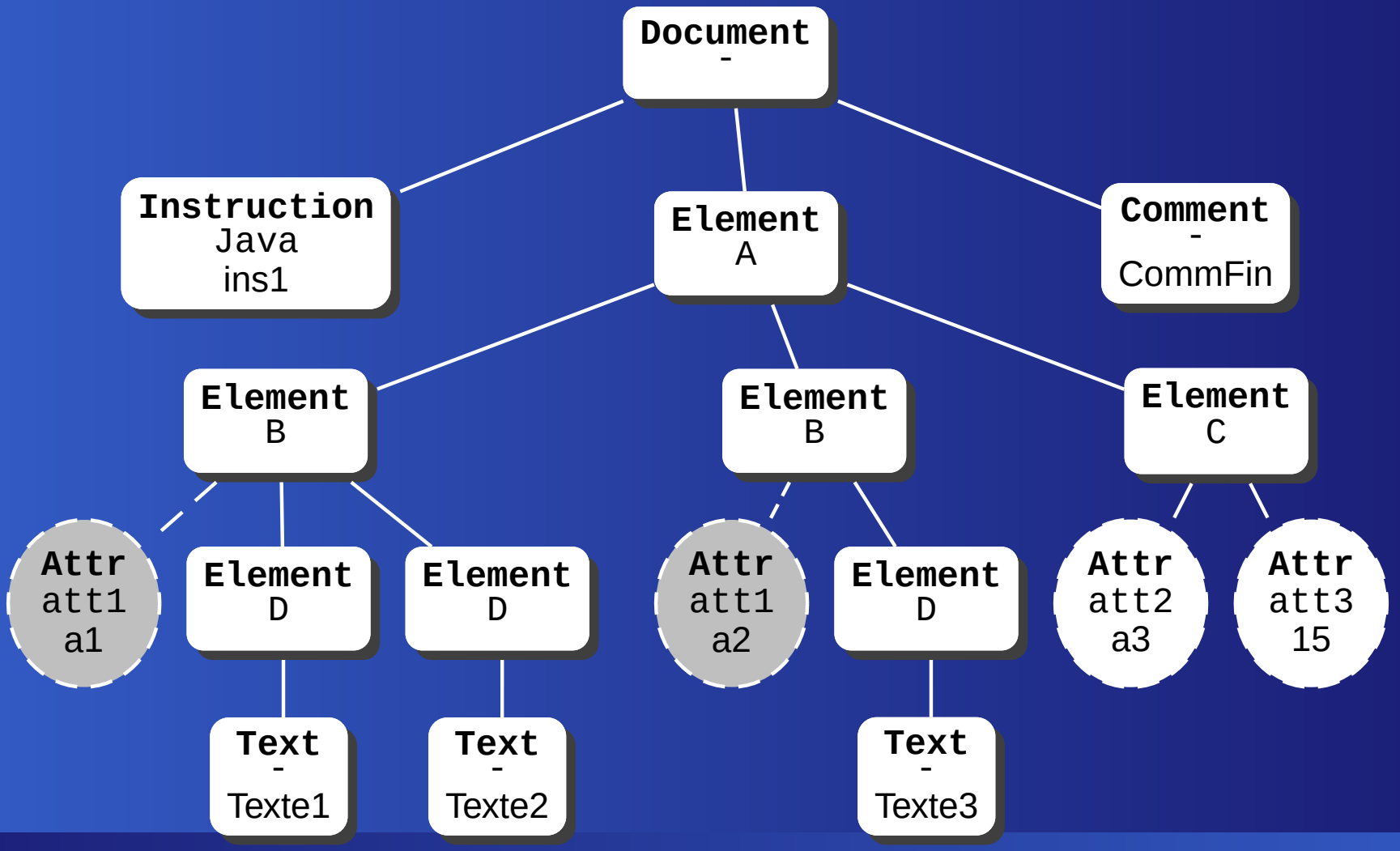
# /A/B/@att1 : première étape



# /A/B/@att1 : seconde étape



# /A/B/@att1 : troisième étape



# Contexte d'évaluation

Une étape s'évalue en tenant compte d'un **contexte** constitué de

- un nœud contexte, position initiale de l'étape ;
- ce nœud fait lui-même partie d'un ensemble obtenu par évaluation de l'étape précédente
  - $\Rightarrow$ on connaît la **taille** de cet ensemble (fonction *last*())
  - $\Rightarrow$ on connaît la **position** du nœud contexte dans cet ensemble (fonction *position*())

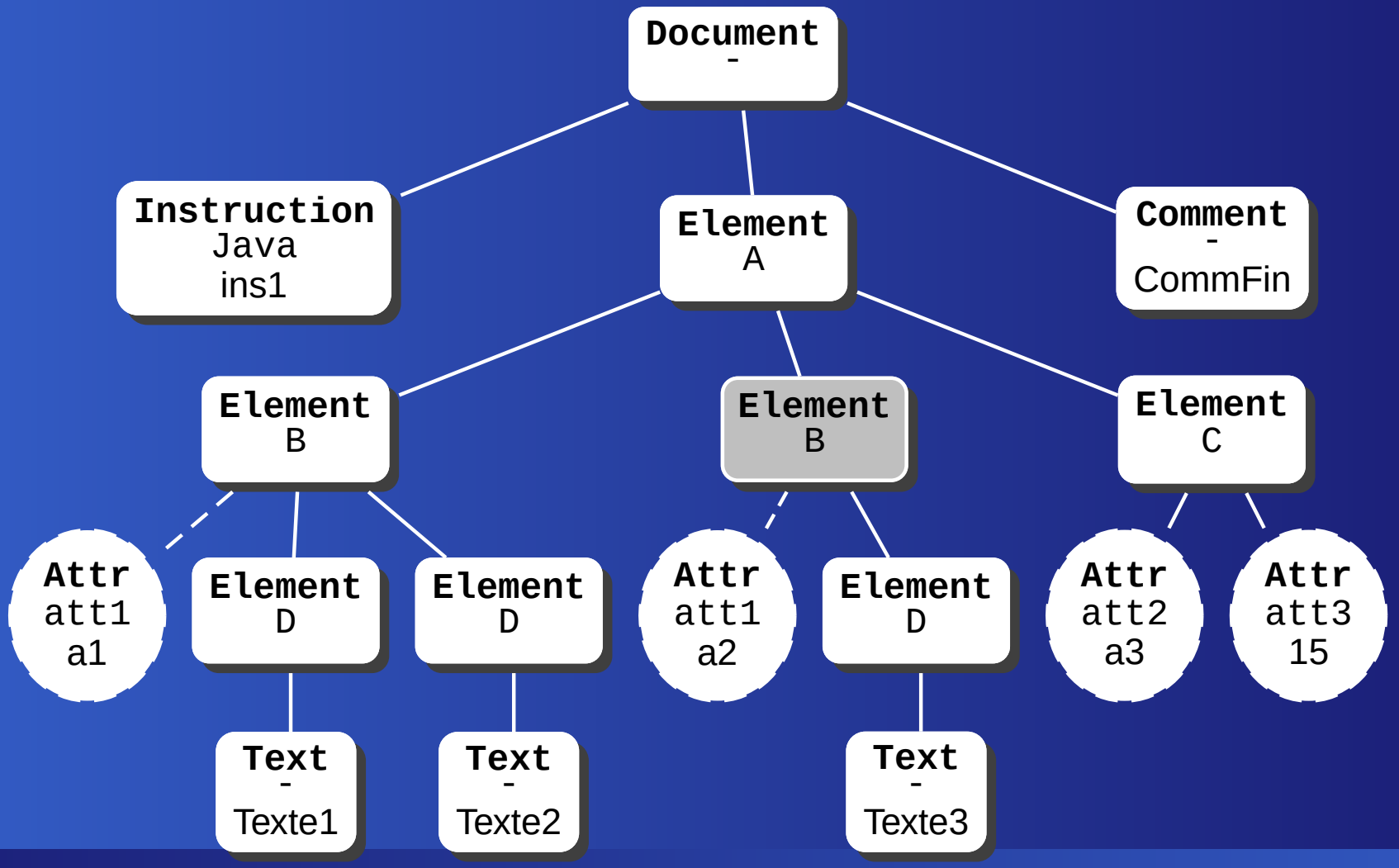
# Les axes XPath

Un axe XPath recouvre les deux notions suivantes :

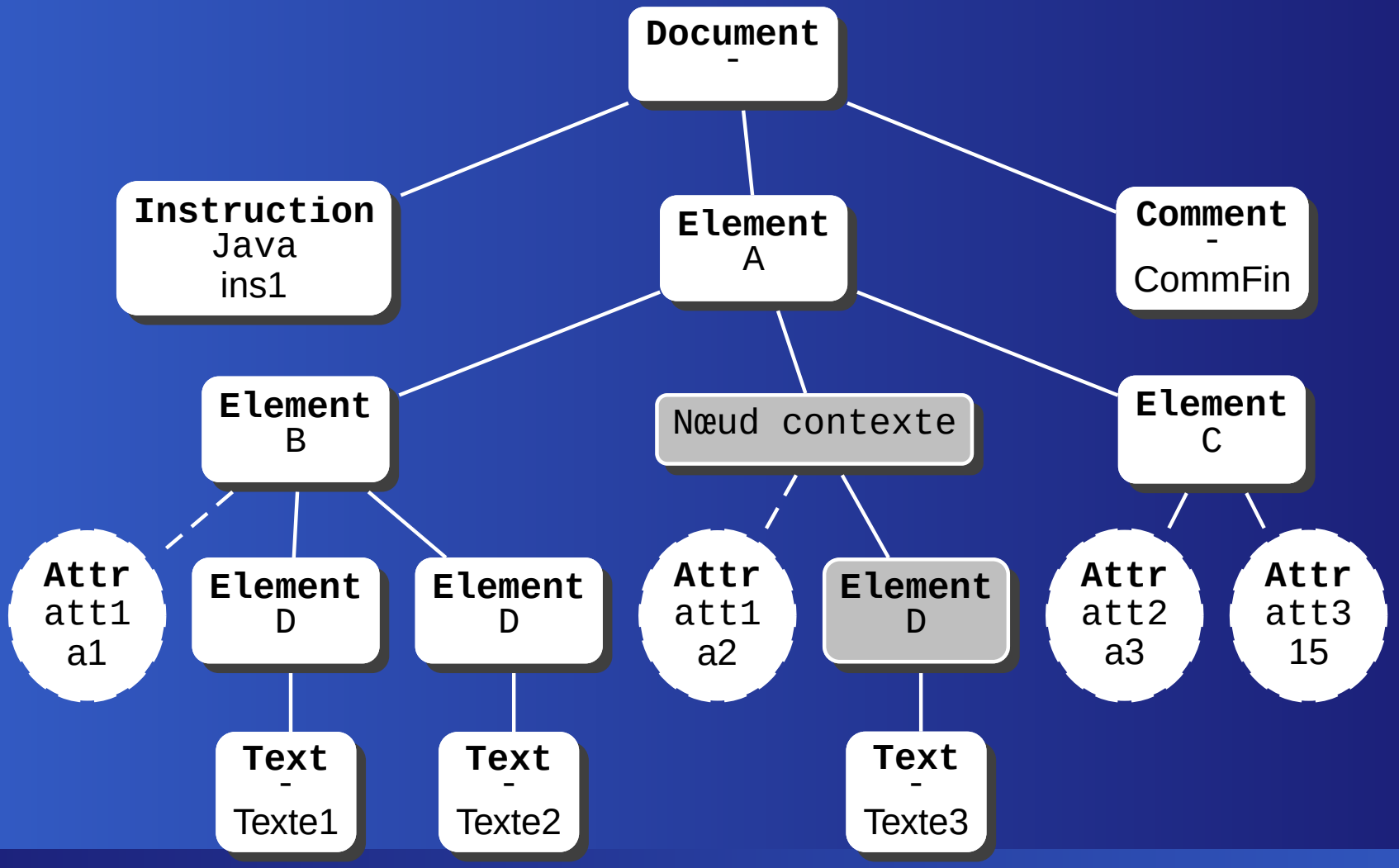
- un sous-ensemble des nœuds de l'arbre relatif au nœud contexte ;
- l'ordre de parcours de ces nœuds à partir du nœud contexte.

NB : il existe des notations abrégées : « @ » remplace `attribute::`.

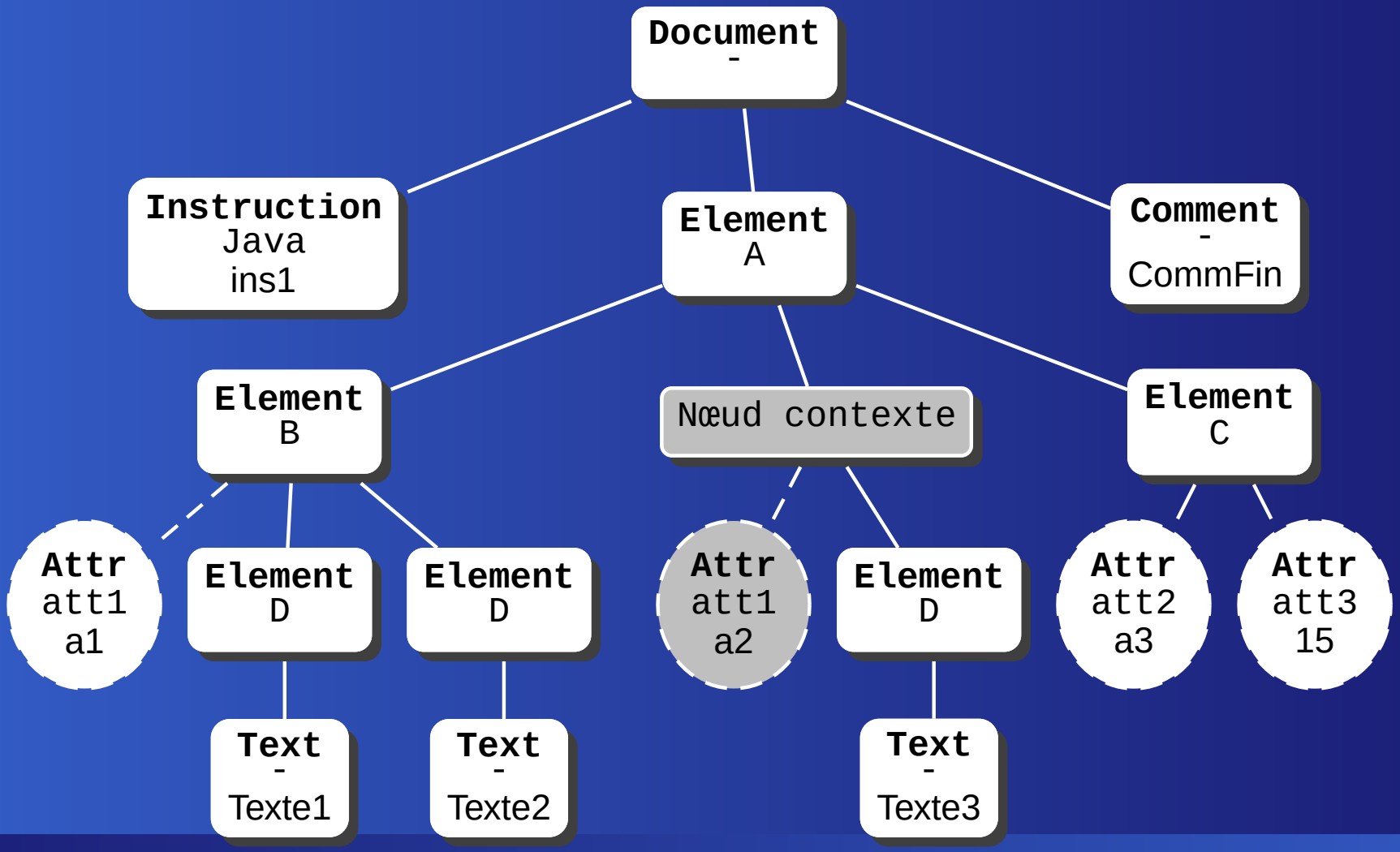
# Exemple : le nœud contexte



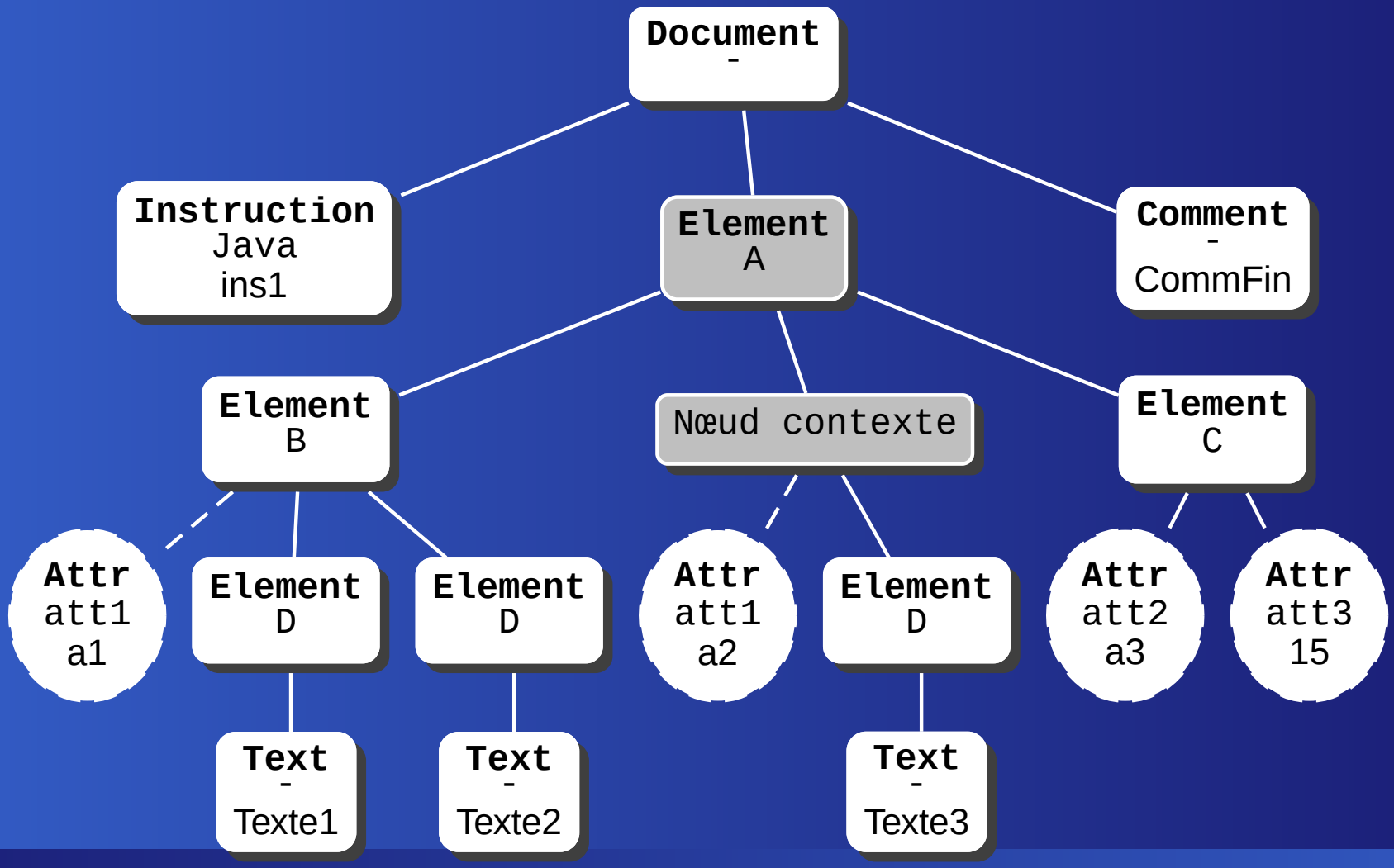
# L'axe child


Document
-
Instruction
Java
ins1
Element
A
Comment
-
CommFin
Element
B
Nœud contexte
Element
C
Attr
att1
a1
Element
D
Element
D
Attr
att1
a2
Element
D
Attr
att2
a3
Attr
att3
15
Text
-
Texte1
Text
-
Texte2
Text
-
Texte3

# L'axe `attribute::att1` (@att1)



# Le père : `parent::A`



# Notation abrégée de `parent::A`

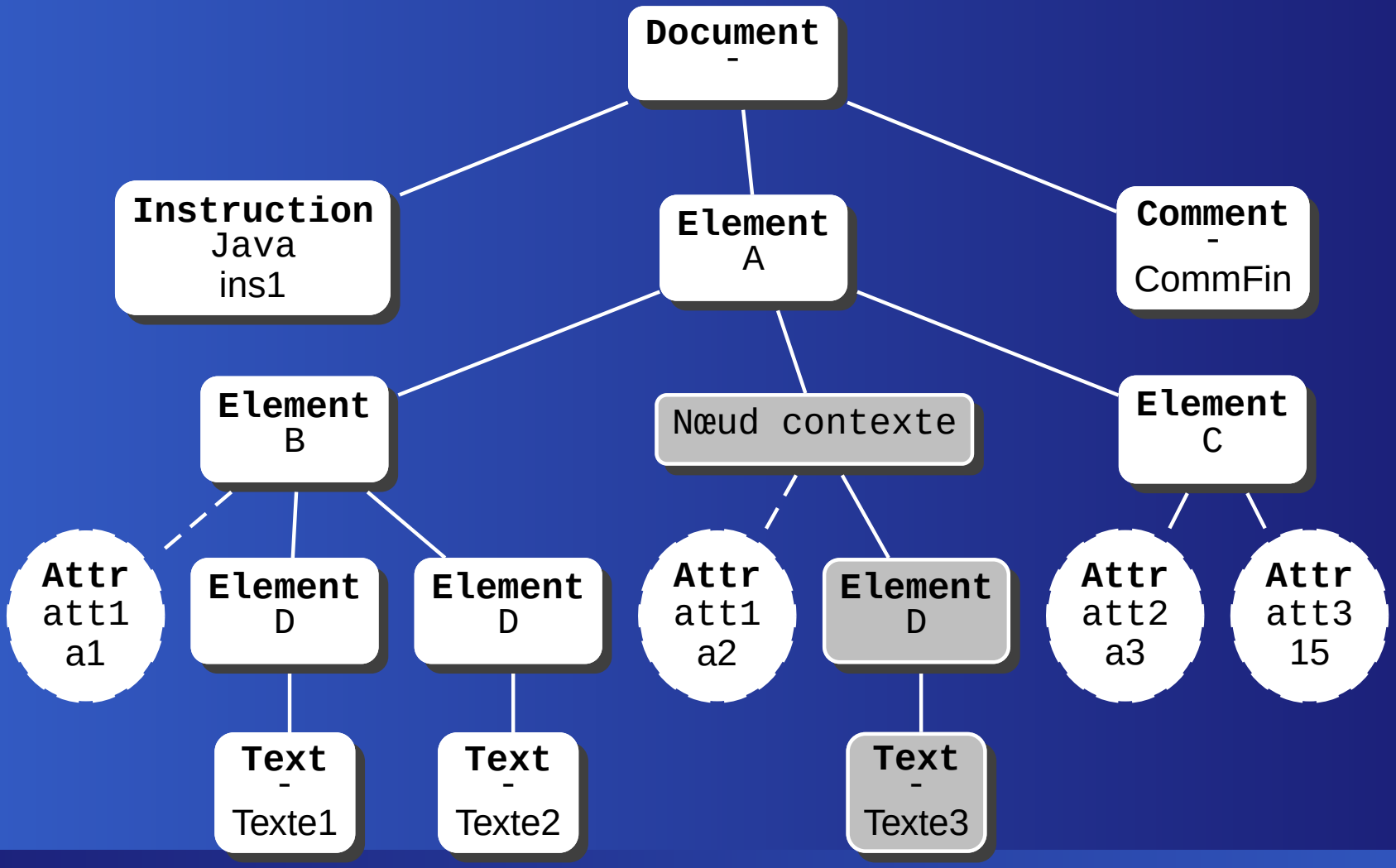
La notation abrégée `..` désigne le père du nœud contexte, **quel que soit son type**.

Équivalent à :

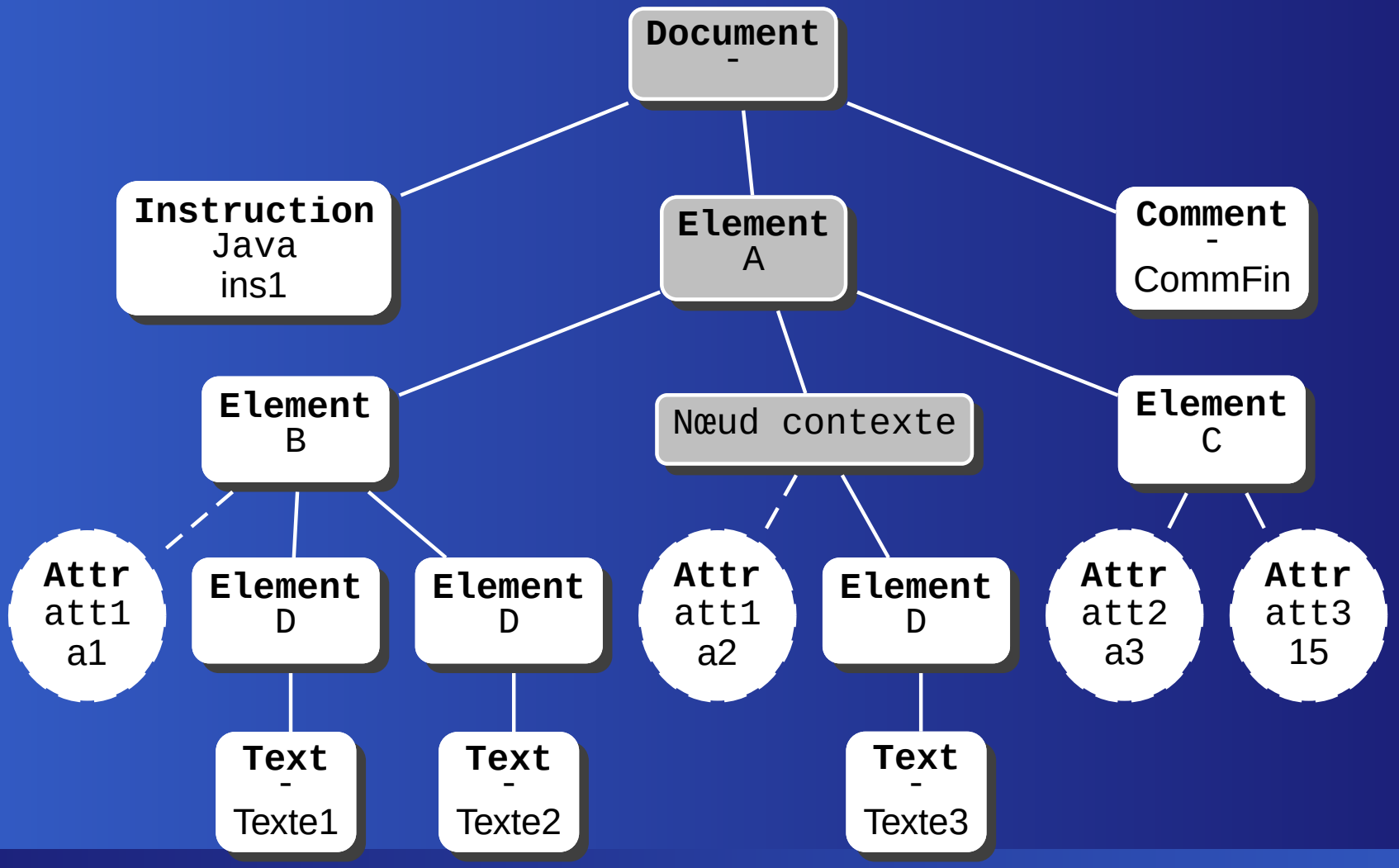
```
parent::node()
```

`node()` est un **filtre** qui désigne tous les types de nœuds (sauf les attributs)

# Descendants : descendant::node(



# Ancêtres : `ancestor::node()`



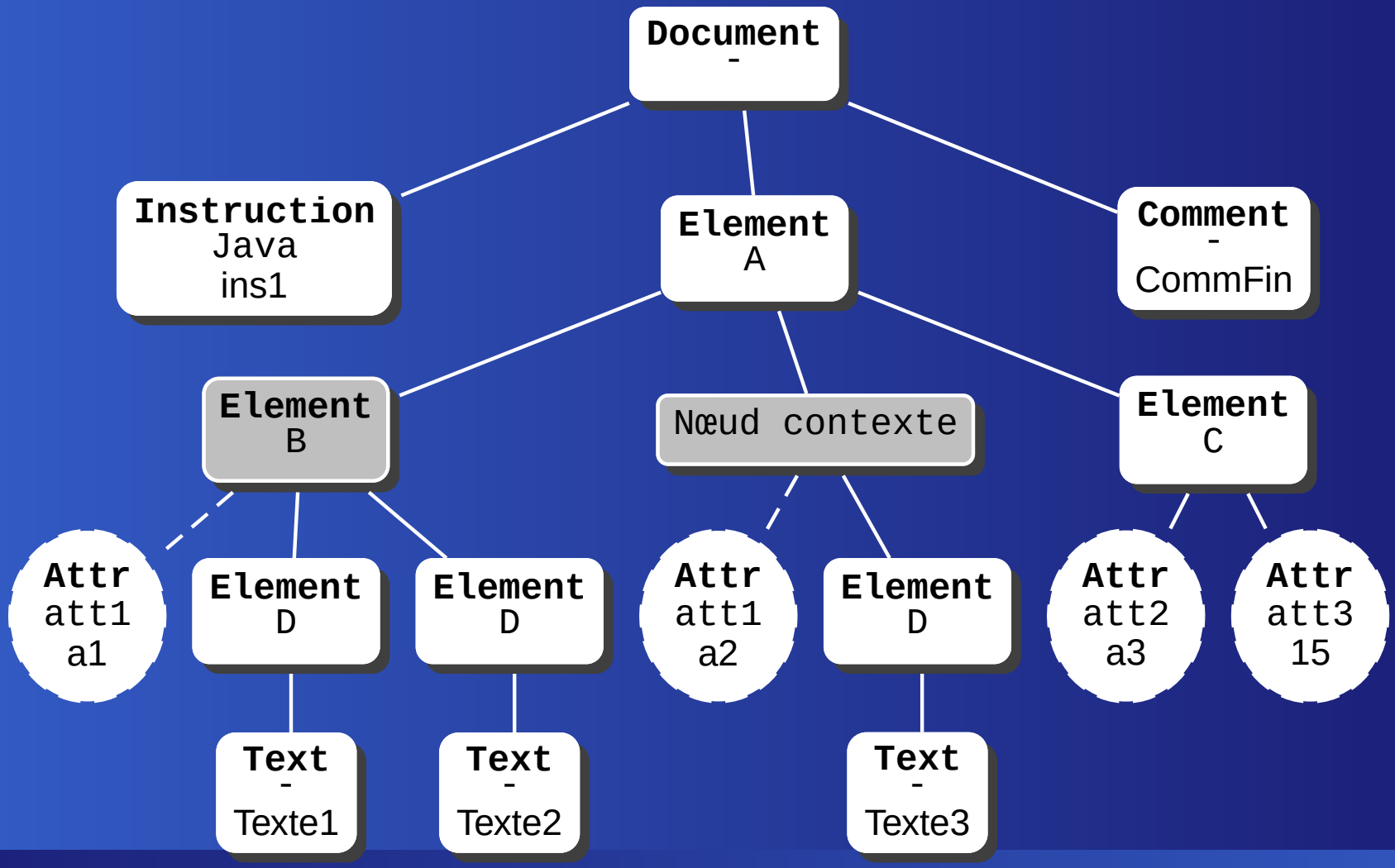
# L'axe `self`

- il désigne le nœud contexte lui-même. Doit être complété par un filtre. Exemples :
  - ⇒ notre nœud contexte : `self::B`
  - ⇒ mais `self::A` renvoie un ensemble vide
- Pour prendre le nœud **quel que soit son type** :

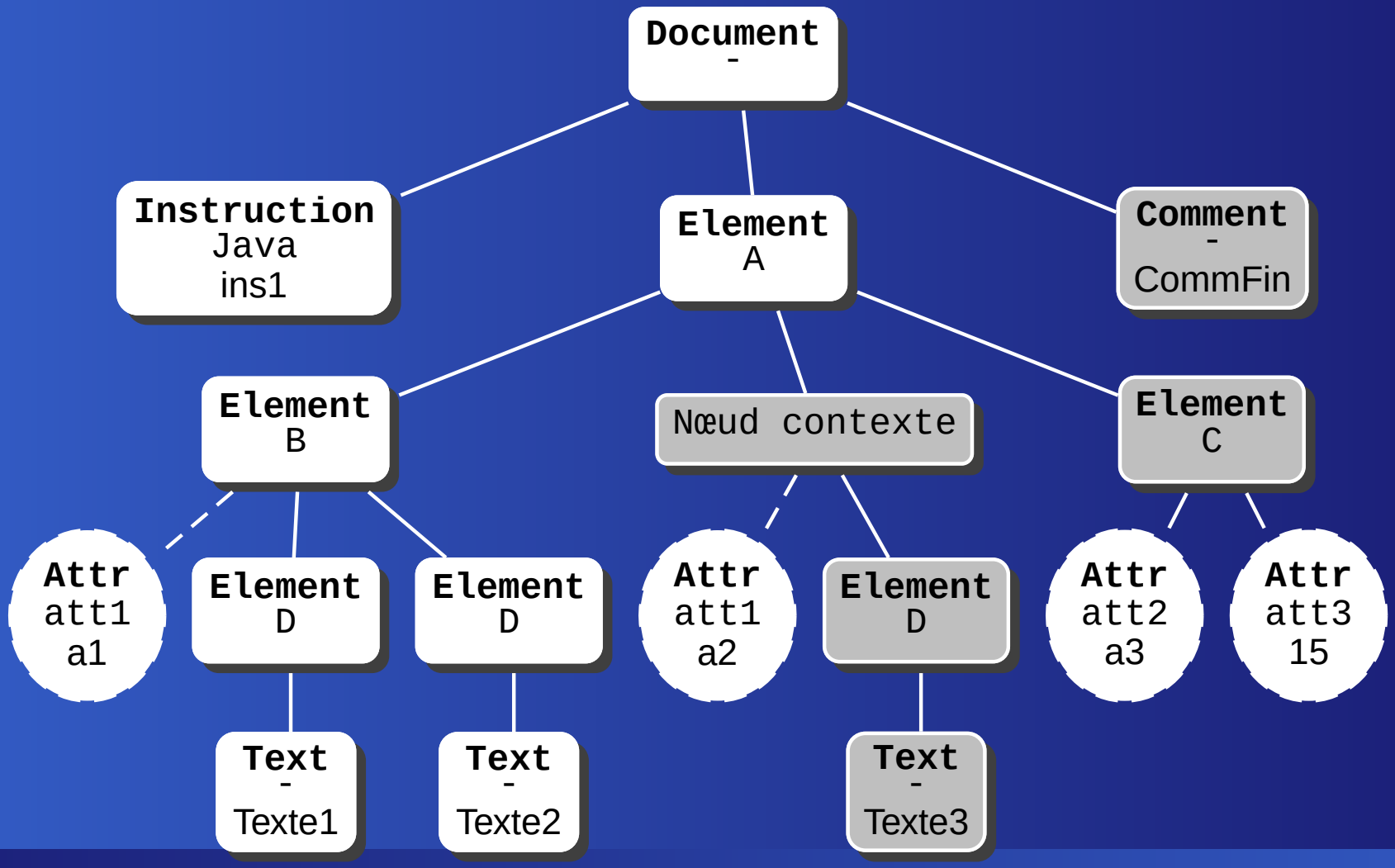
```
self::node()
```

- Notation abrégée : « . »

# preceding-sibling::node()



# following::node()



# Autres axes

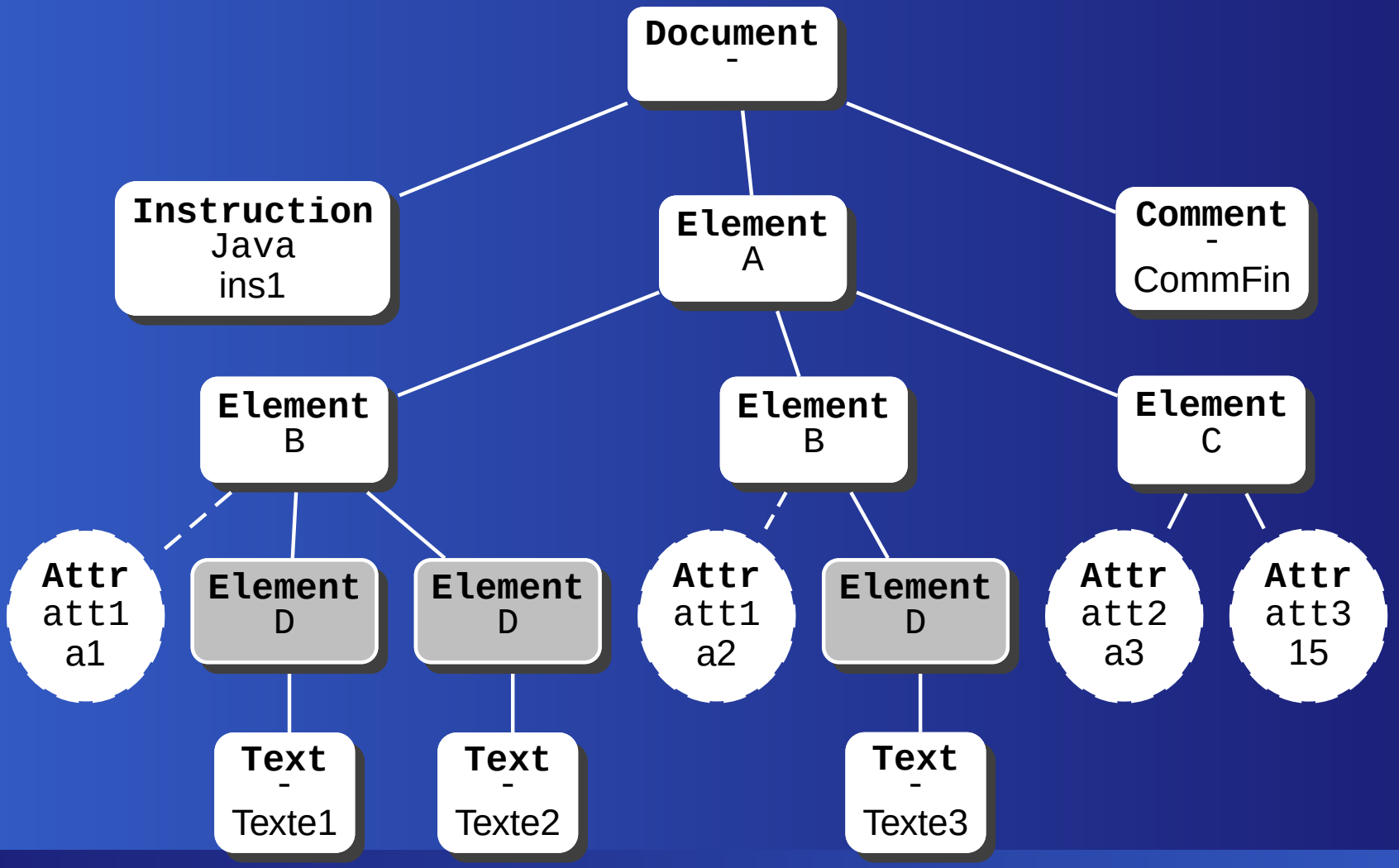
- `preceding` : les précédents (dans l'ordre du document)
- `descendant-or-self` : les descendants, plus le nœud contexte
  La notation abrégée : « // » exprime `descendant-or-self::node()/child::`
- `ancestor-or-self` : les ancêtres, plus le nœud contexte

# Les filtres

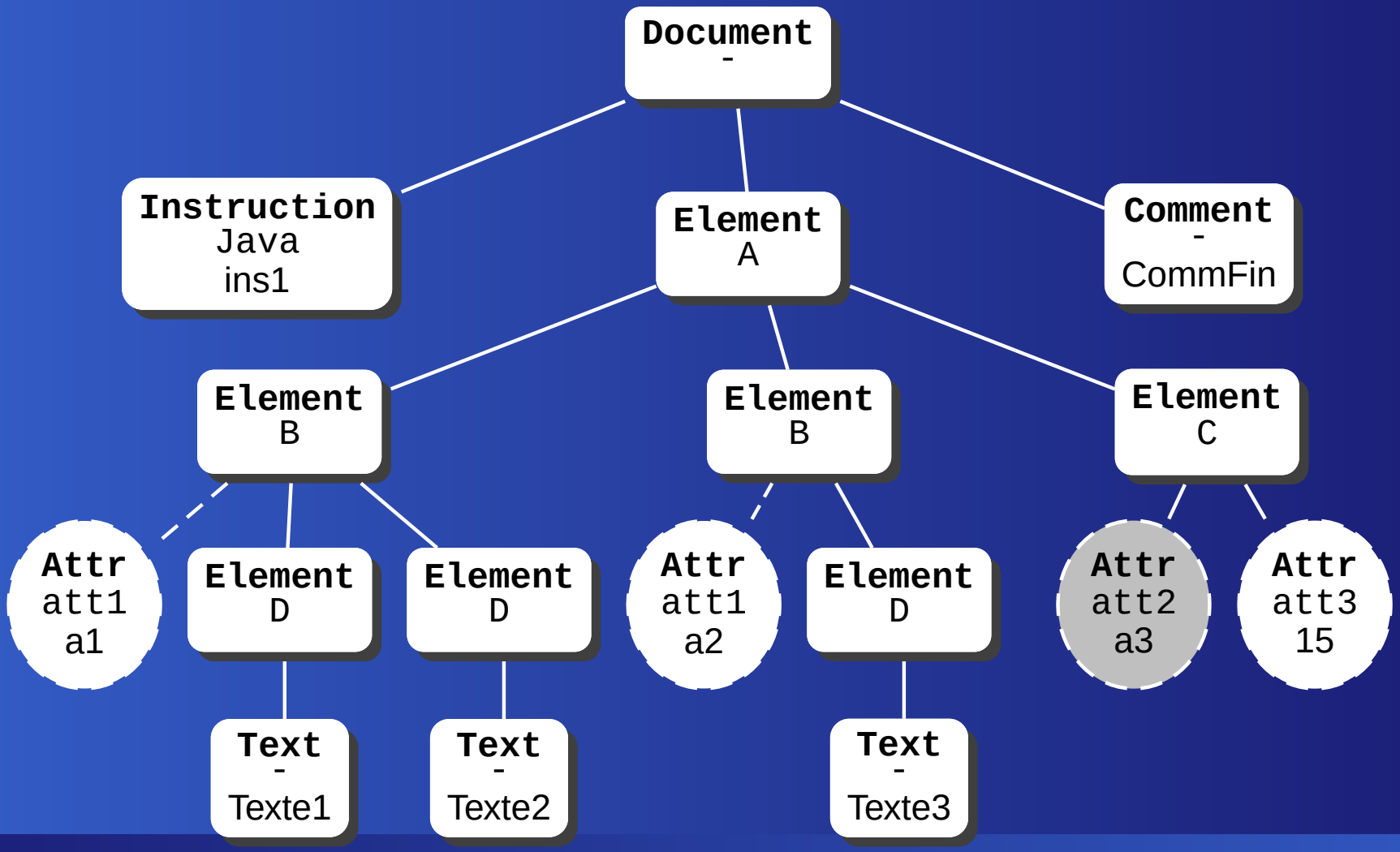
Deux manières de filtrer les nœuds :

- par leur nom ;
  - $\Rightarrow$ valable pour les types de nœuds qui ont un nom : **Element**, **ProcessingInstruction** et **Attr**
- par leur type.

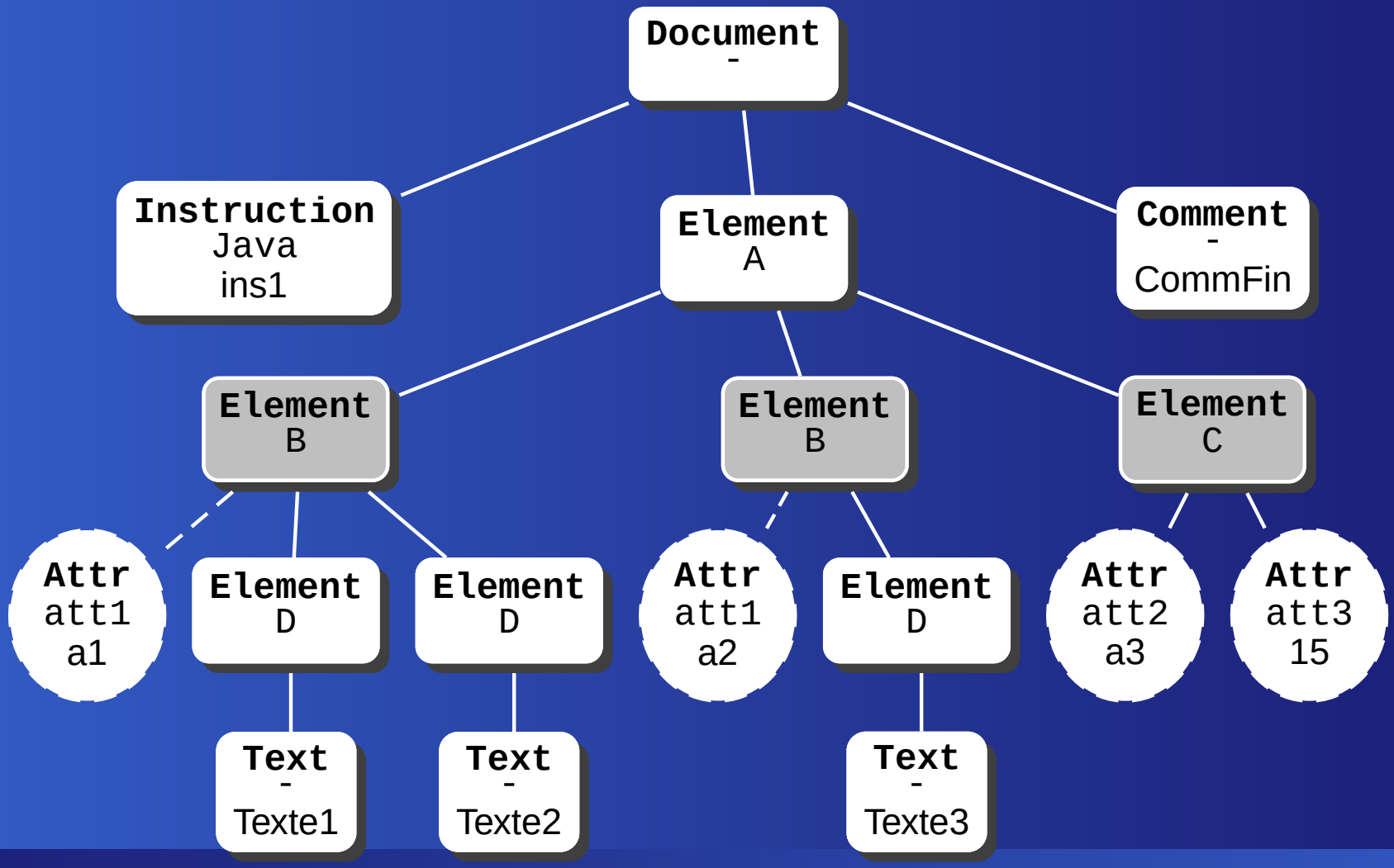
# /A/B/D


Document
-
Instruction
Java
ins1
Element
A
Comment
-
CommFin
Element
B
Element
B
Element
C
Attr
att1
a1
Element
D
Element
D
Attr
att1
a2
Element
D
Attr
att2
a3
Attr
att3
15
Text
-
Texte1
Text
-
Texte2
Text
-
Texte3

# /descendant::node()/@att2



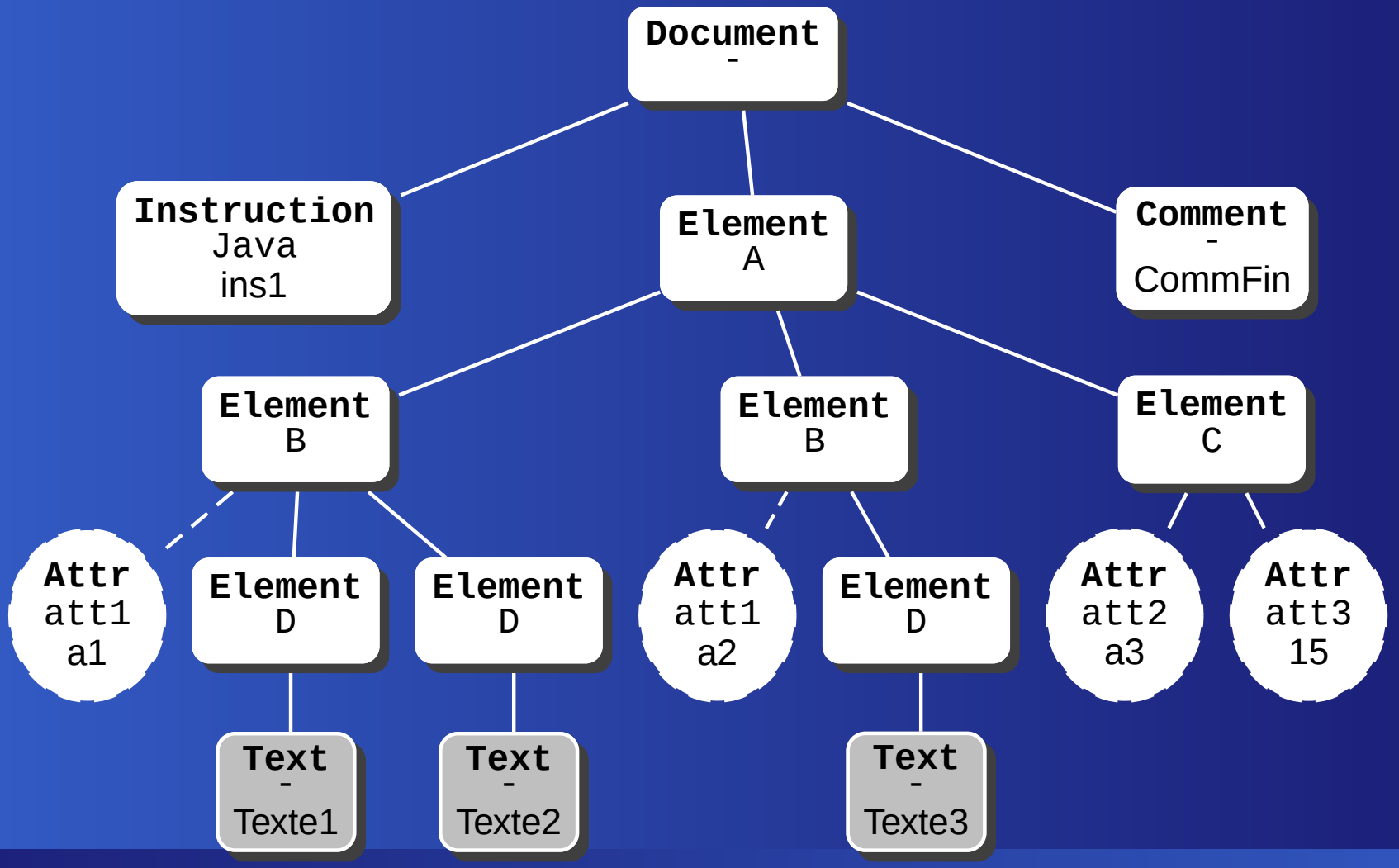
# Nom générique : /A/*



# Filtrage sur le type de nœud

- `text()`. Nœuds de type **Text**
  Exemple : `/A/B//text()`
- `comment()`. Nœuds de type **Comment**
  Exemple : `/comment()`
- `processing-instruction()`. Nœuds de type **ProcessingInstruction**
  Exemple : `/processing-instruction()`, ou `/processing-instruction('java')`
- `node()`. Tous les types de nœud

# Résultat de `/A/B//text()`



# Prédicats

- **Prédicat** : expression booléenne constituée d'un ou plusieurs tests, composés avec les connecteurs logiques habituels `and` et `or`
- **Test (1)** : toute expression XPath, dont le résultat est convertie en booléen
- **Test (2)** : une comparaison ou un appel de fonction.

$\Rightarrow$ il faut connaître les règles de conversion

# Quelques exemples

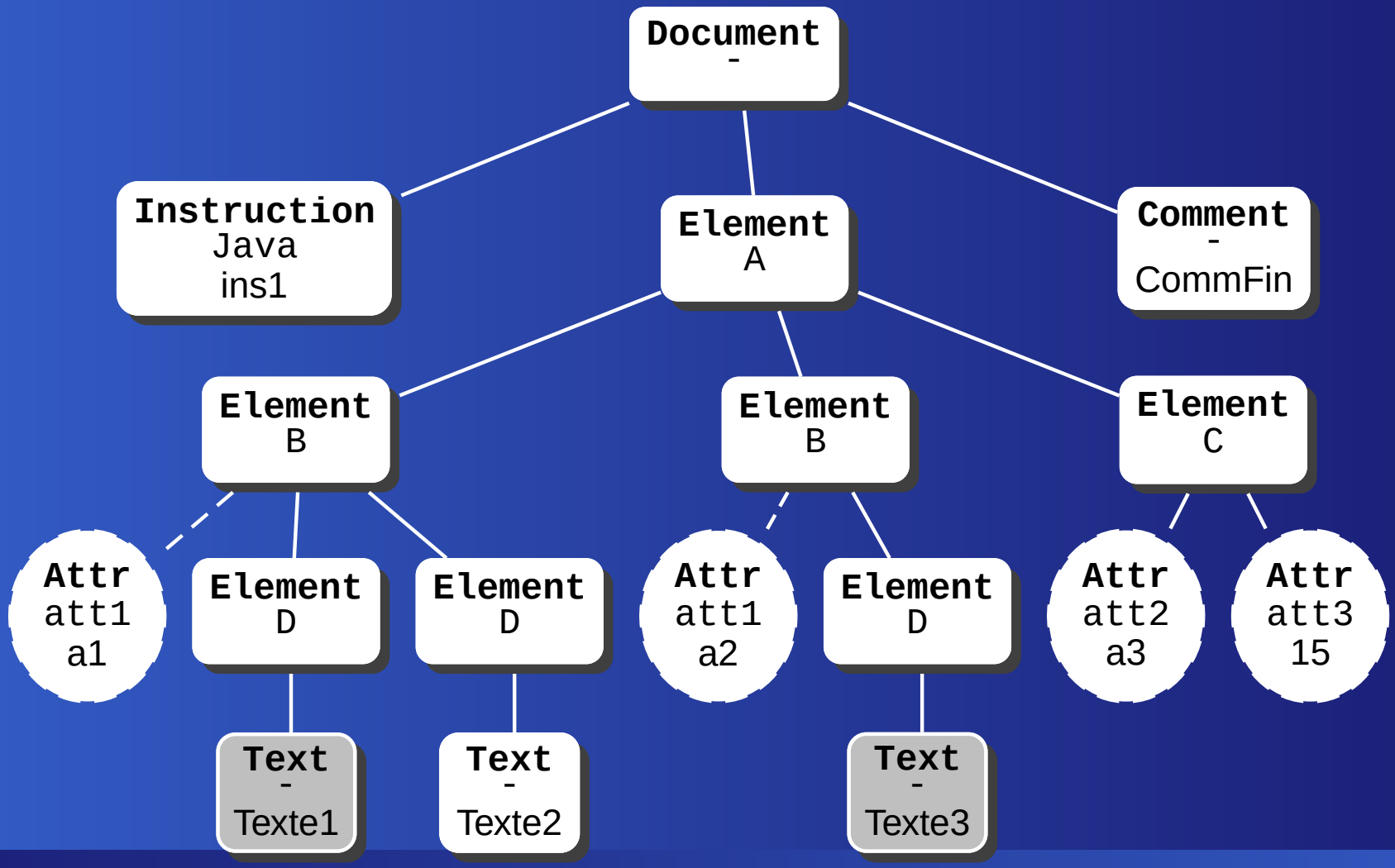
- `/A/B[@att1]`
  Les nœuds `/A/B` qui ont un attribut `@att1`
- `/A/B[@att1='a1']`
  Les nœuds `/A/B` qui ont un attribut `@att1` valant 'a1'
- `/A/B/descendant::text()[position()=1]`
  Le premier nœud de type **Text** descendant d'un `/A/B`. S'abrège :
  `/A/B/descendant::text()[1]`

# Pour bien comprendre

Dans l'expression `/A/B[@att1]` :

- On s'intéresse aux nœuds de type `B` fils de l'élément racine `A`.
- Parmi ces nœuds on ne prend que ceux pour lesquels le prédicat `[@att1]` s'évalue à `true`
- Cette expression s'évalue avec pour nœud contexte un élément `B`
- `[@att1]` vaut `true` ssi `@att1` renvoie un ensemble de nœuds **non vide**

# /A/B/descendant::text()[1]



# Typage avec XPath

On peut effectuer des comparaisons, des opérations. Cela implique un **typage** et des conversions de type.
Types XPath :

- *les numériques*
- *les chaînes de caractères*
- *les booléens* (`true` et `false`)
- enfin *les ensembles de nœuds*

# Numériques

Notation décimale habituelle

- Comparaisons habituelles (<, >, !=)
- Opérations : +, -, *, div, mod
- La fonction *number*() permet de tenter une conversion
- Si la conversion échoue on obtient `NaN` (*Not a Number*). **À éviter...**

Ex : `//node()[number(@att1) mod 2 = 1]`

# Conversions

Deux conversions sont toujours possibles.

- **Vers une chaîne de caractères**.
  - ⇒ utile pour la production de texte en XSLT (balise `xsl:value-of`)
- **Vers un booléen**
  - ⇒ utile pour les tests effectués dans XSLT (`xsl:if`, `xsl:when`)

# Conversions booléennes

- **Pour les numériques** : `0` ou `NaN` sont `false`, tout le reste est `true`
- **Pour les chaînes** : une chaîne vide est `false`, tout le reste est `true`
- **Pour les ensembles de nœuds** : un ensemble vide est `false`, tout le reste est `true`

Comparaisons : des règles de conversion étranges...

# Fonctions XPath

Quelques fonctions utiles dans les prédicats :

- *concat(chaîne1, chaîne2, ...)* pour concaténer des chaînes
- *contains(chaîne1, chaîne2)* teste si *chaîne1* contient *chaîne2*
- *count (expression)* renvoie le nombre de nœuds désignés par *expression*
- *name()* renvoie le nom du nœud contexte
- *not(expression)* permet d'exprimer la négation

# Exemples d'expressions XPath

- `child::A/descendant::B`. Eléments de type `B` descendants d'un élément de type `A`, lui-même fils du nœud contexte
  Forme abrégée : `A//B`
- `child::*/child::B` donne les éléments de type `B` petit-fils du nœud contexte ;
  Forme abrégée : `*/B`
- `descendant-or-self::B` : donne les éléments de type `B` fils du nœud contexte, et le nœud contexte lui-même s'il est de type `B` ;

# Exemples XPath (suite)

- `child::B[position()=last() - 1]` : donne l'avant-dernier élément de type `B` fils du nœud contexte ;
  Forme abrégée : `B[last()-1]`
- `following-sibling::B[position()=1]` : donne le premier frère de droite du nœud contexte dont le type est `B` ;
- `/descendant::B[position()=12]` : donne le douzième élément de type `B` du document ;

# Exemples XPath (suite et fin)

- `child::B[child::C]` : les fils de type `B` avec au moins un fils de type `C`
  Forme abrégée : `B[C]`
- `/descendant::B[attribute::att1 and attribute::att2]` : donne les éléments de type `B` qui ont au moins un attribut `att1` et un attribut `att2` ;
  Forme abrégée : `//B[@att1 and @att2]`
- `child::*[self::B or self::C]` : donne les fils de type `B` ou de type `C` ;