Il linguaggio XQuery

XQuery in breve

- XQuery è un linguaggio di interrogazione per dati espressi in XML, e può essere dunque utilizzato per accedere a documenti strutturati e semi-strutturati.
- E' diventato una Recommendation del W3C nel gennaio 2007.
- XQuery è un linguaggio case sensitive (come XML), estende l'ultima versione di XPath e ha funzionalità analoghe a XSLT (eXtensible Stylesheet Language Transformations).
- Vedremo gli aspetti essenziali (cioè largamente implementati e utilizzati) dell'attuale versione 1.0. Non tratteremo dunque:
  - La definizione di funzioni.
  - Le estensioni per l'information retrieval (XQuery-FullText).
  - Le estensioni per l'aggiornamento di documenti (XQuery-Update).

Argomenti trattati

- Il modello dei dati di XQuery.
- Espressioni di navigazione (Path Expressions).
- Espressioni FLWOR.
- Altre espressioni di uso comune: espressioni condizionali (if-then-else), confronti, operatori logici e aritmetici, quantificatori (esiste/per ogni).
- Alcune funzioni standard fondamentali.

Il modello dei dati di XQuery

Sequenze (1)

- A differenza di SQL, che opera su relazioni (cioè insiemi), XQuery opera su **sequenze**, che possono contenere:
  - **Valori atomici**, come la stringa “ciao” o l’intero 3.
  - **Nodi**.
- Un’espressione XQuery riceve in input zero (nel caso di costruttori) o più sequenze e produce una sequenza.
- Le caratteristiche principali delle sequenze sono le seguenti:
  - Le sequenze sono in generale **ordinate**, per cui:

    (1, 2) è diversa da (2, 1).
  - Le sequenze **non sono annidate**, per cui:

    (( ), 1, (2, 3)) è uguale a (1, 2, 3).
  - Non c’è differenza tra un *item* e una sequenza con lo stesso *item:*

    (1) è uguale a 1.

Sequenze (2)

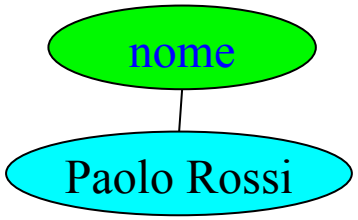
- Per manipolare sequenze XQuery mette a disposizione i seguenti operatori:
- **,** (virgola) e **to** (i seguenti sono esempi di sintassi alternative per definire una sequenza di interi 1, 2 e 3):
  - (1, 2, 3)
  - (1, (), (2, 3))
  - (1 to 3)
  - (1, 2 to 3)
- union (anche nella forma equivalente |), intersect, except
  - (A) union (A, B) -> (A, B)
  - (A, B) intersect (B, C) -> (B)
  - (A, B) except (B) -> (A)

Nodi (1)

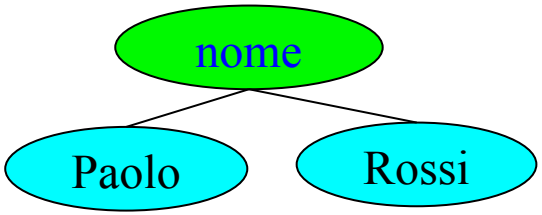
- Oltre ai valori atomici, una sequenza può contenere nodi.
- Documenti e frammenti XML sono rappresentati come alberi, i cui nodi possono essere di tipo: **documento, elemento, attributo, namespace, testuale, commento,** e **processing instruction**.
- I nodi di tipo **namespace**, come <xml:lang>, non sono rappresentati nel modello dei dati di XQuery, per cui non li tratteremo.
- In modo analogo, non tratteremo **commenti** e **processing instructions**, che non sono (o non dovrebbero) essere utilizzati per memorizzare dati.
- Il **valore testuale** (string value) dei nodi di tipo documento ed elemento corrisponde alla concatenazione dei valori testuali di tutti i propri discendenti di tipo testuale, nell'ordine in cui essi si trovano nel documento.

Nodi (2)

- I nodi di tipo **attributo** non sono ordinati – se si estraggono tramite una interrogazione XQuery, l'ordine non necessariamente corrisponde a quello in cui vengono incontrati nel documento XML di origine.
- Una volta estratti, essi sono però inseriti in una sequenza, di cui mantengono l'ordine.
- Due **nodi testuali** non possono essere adiacenti: un elemento `<nome>Paolo Rossi</nome>` è dunque rappresentato come segue.



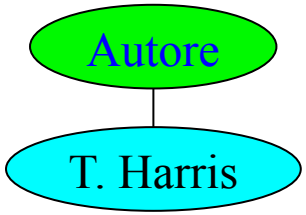
CORRETTO



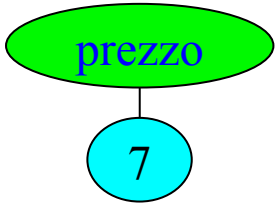
SCORRETTO!

Nodi (3)

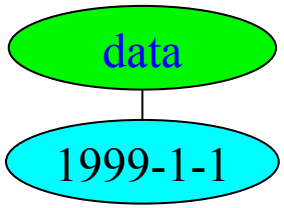
- Se al documento è associato uno schema (XML Schema), i nodi possono avere un tipo, come nei seguenti esempi:



E' di tipo *string* con valore testuale "T.Harris".



E' di tipo *integer con valore intero 7.*

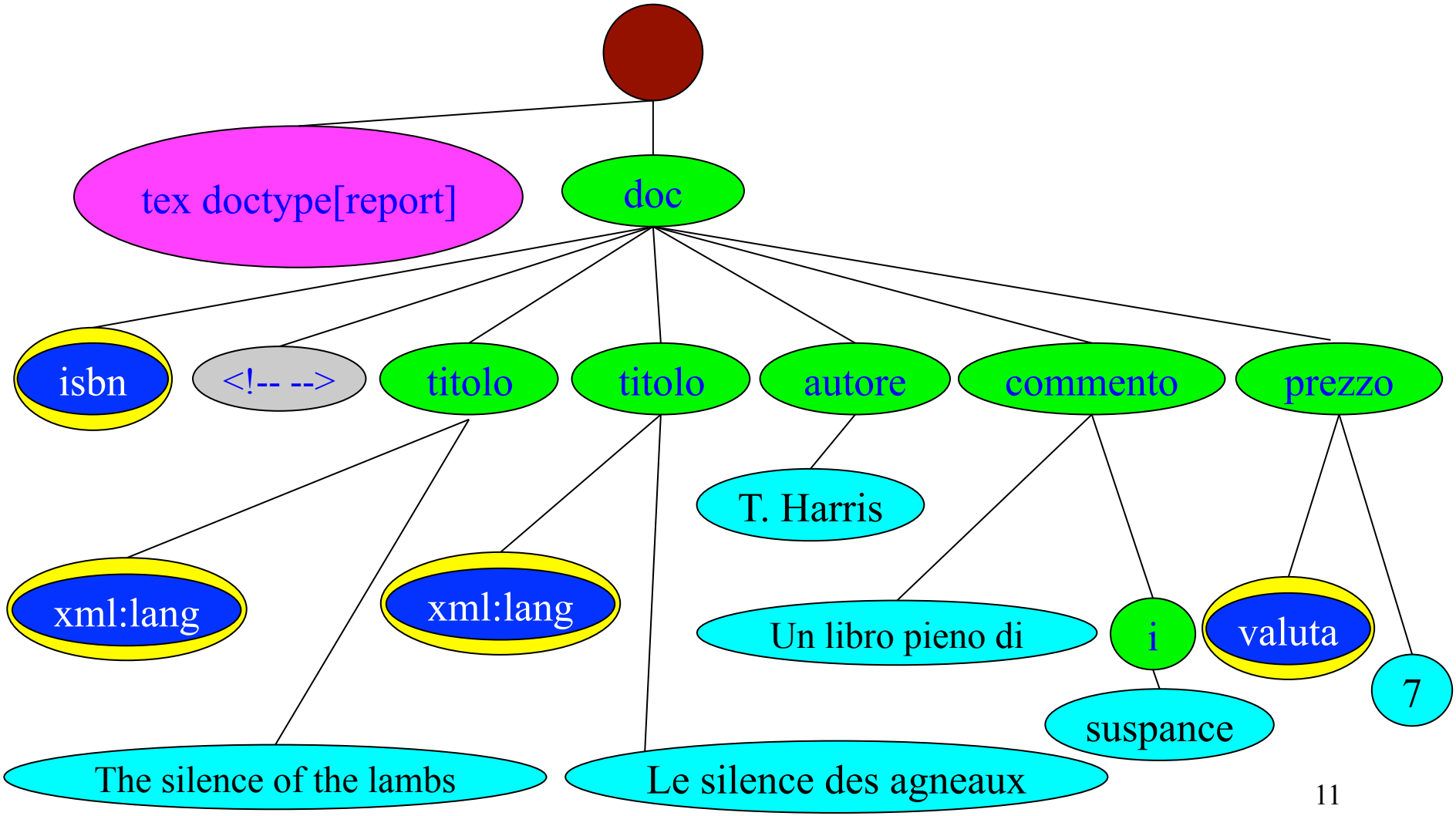


E' di tipo *date* con valore "Primo Gennaio 1999".

Esempi di nodi (1)

```
<?xml version="1.0"?>
<?tex doctype[report] ?>
<doc isbn="2-266-04744-2">
        <!-- manca l'editore! -->
        <autore>T. Harris</autore>
        <titolo xml:lang="en">The silence of the lambs</titolo>
 <titolo xml:lang="fr">Le silence des agneaux</titolo>
<commento>
        Un libro pieno di <i>suspance</i>
        </commento>
        <prezzo valuta="euro">7</prezzo>
</doc>
```

Esempi di nodi (2)



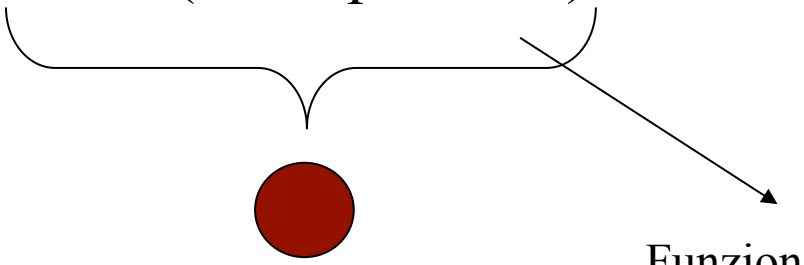
Path Expressions

Osservazioni

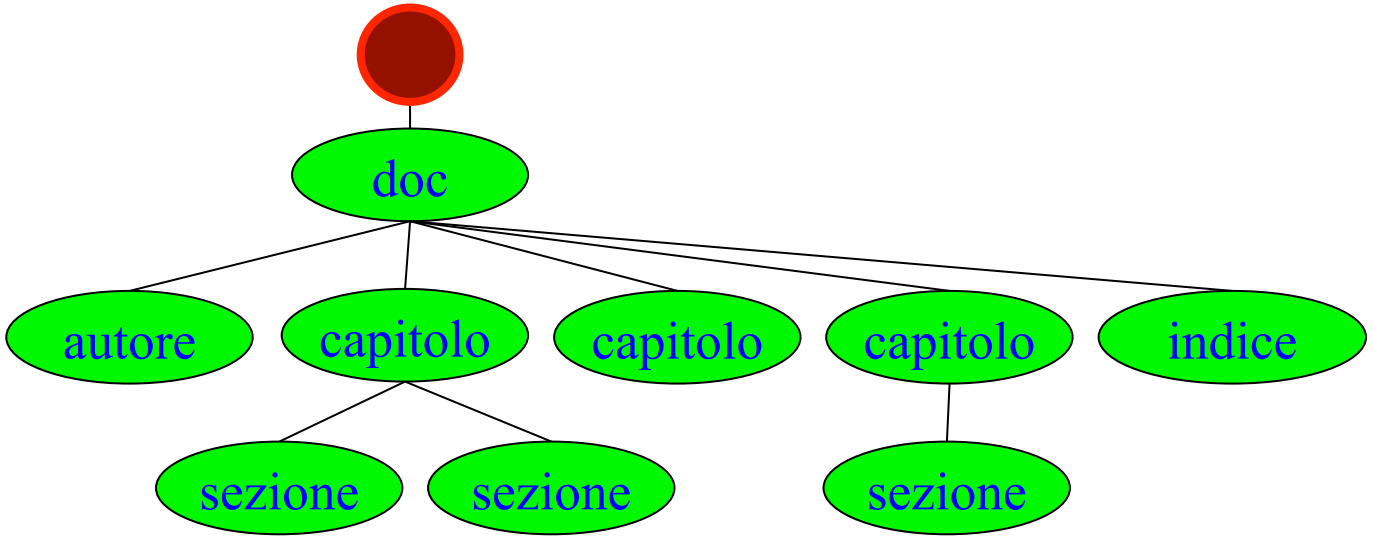
- Le Path Expressions possono essere utilizzate per estrarre nodi e valori da alberi XML, e per verificarne proprietà.
- Ricordiamo che, come per ogni altra espressione in XQuery, anche le Path Expressions elaborano **sequenze**.
- Una Path Expression consiste in una serie di step, separati dal carattere /.
- Ogni step viene valutato in un **contesto**, cioè una sequenza di nodi (con informazioni accessorie, ad esempio la posizione del nodo), e produce una sequenza.
- Lo step successivo viene valutato utilizzando come contesto la sequenza di nodi prodotta dallo step precedente.

Esempio di valutazione (step 1)

fn:doc(‘esempio.xml’) / child::doc / child::capitolo / child::sezione

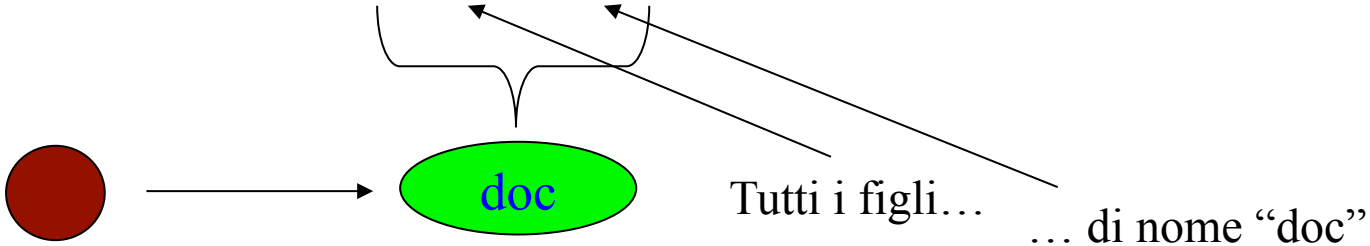


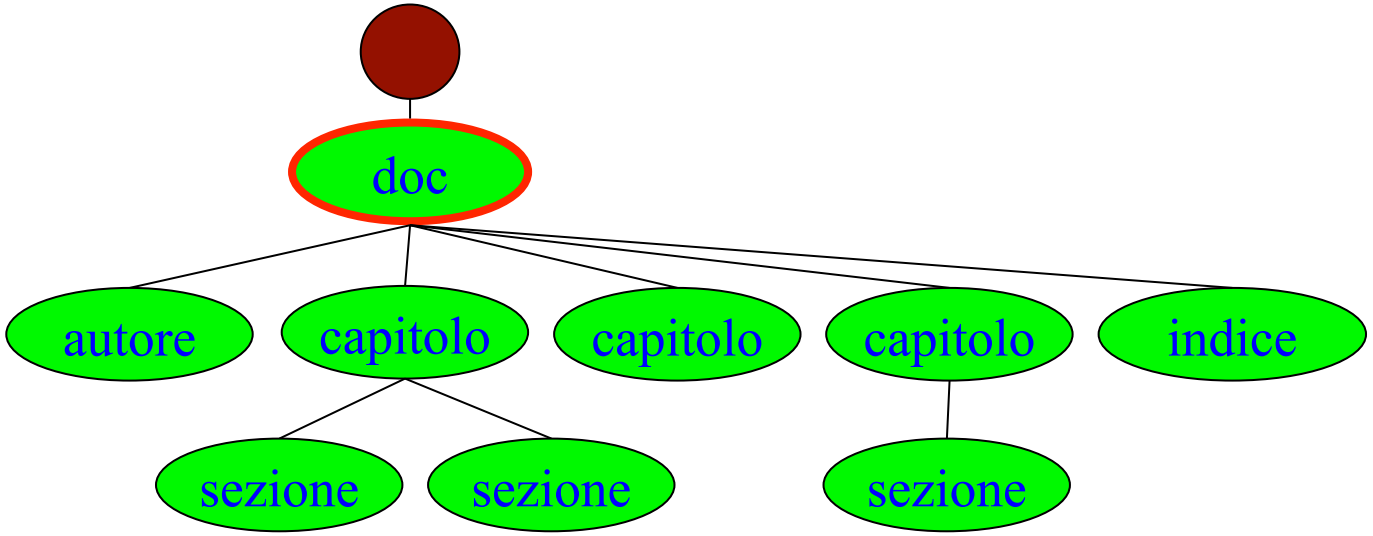
Funzione standard di input



Esempio di valutazione (step 2)

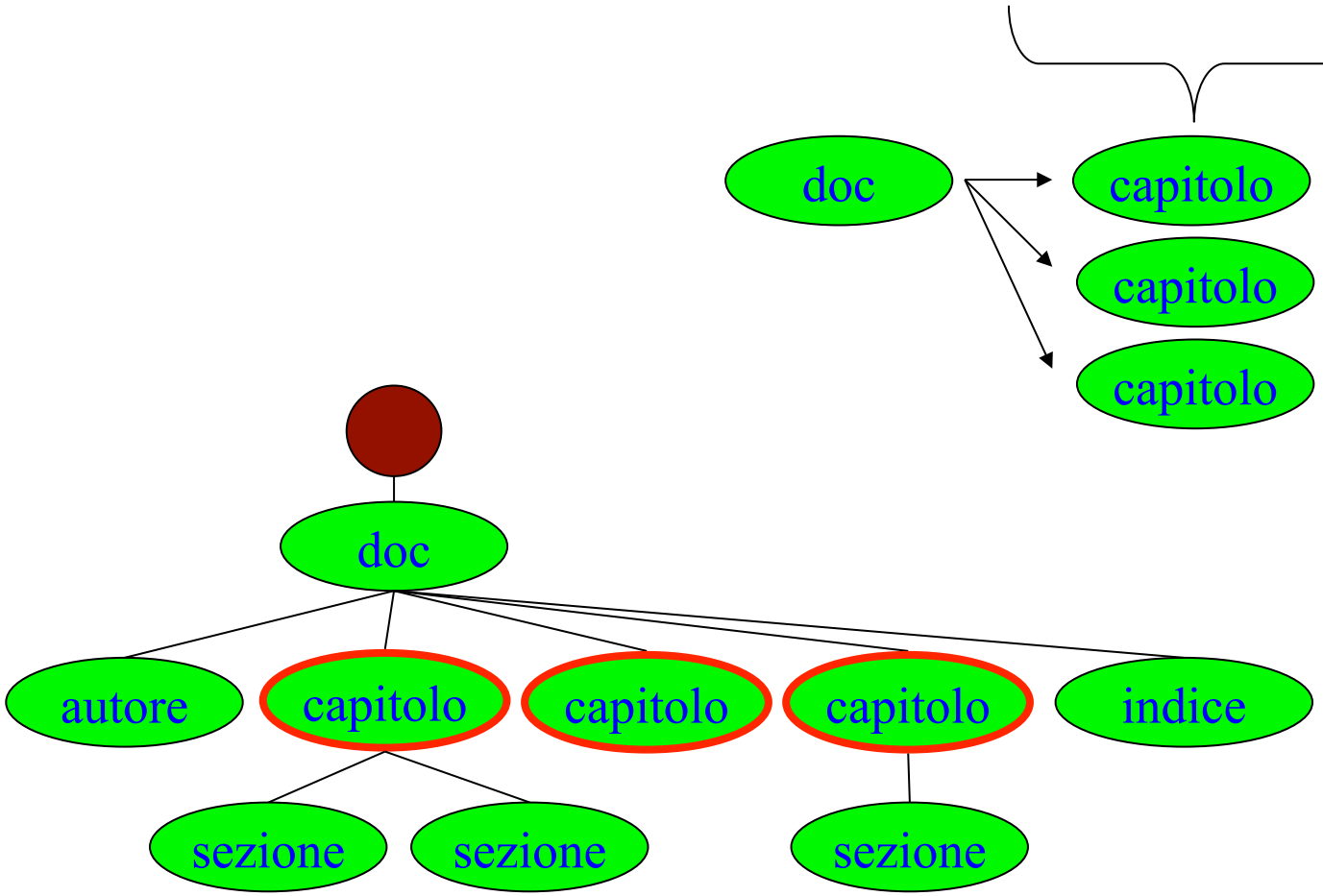
fn:doc(‘esempio.xml’) / child::doc / child::capitolo / child::sezione





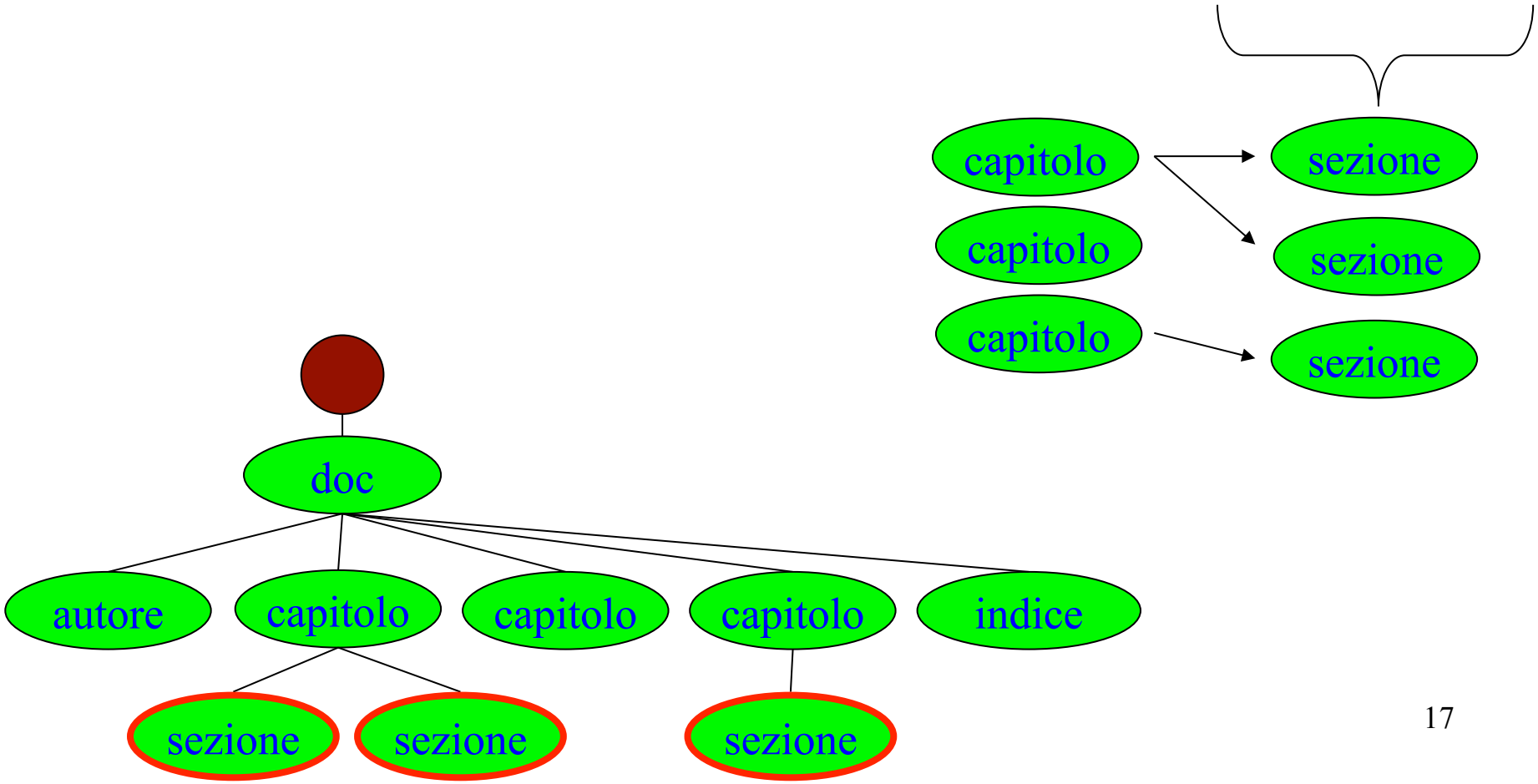
Esempio di valutazione (step 3)

fn:doc(‘esempio.xml’) / child::doc / child::capitolo / child::sezione



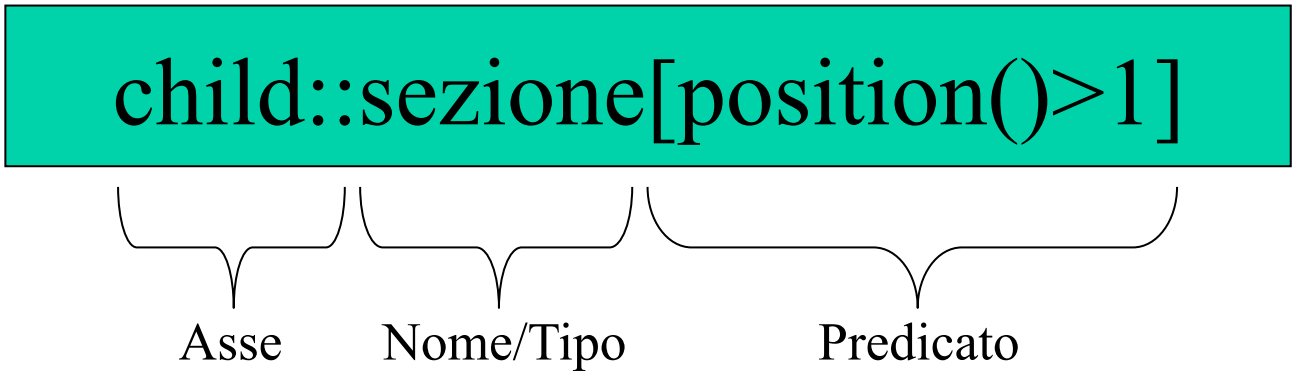
Esempio di valutazione (step 4)

fn:doc(‘esempio.xml’) / child::doc / child::capitolo / child::sezione



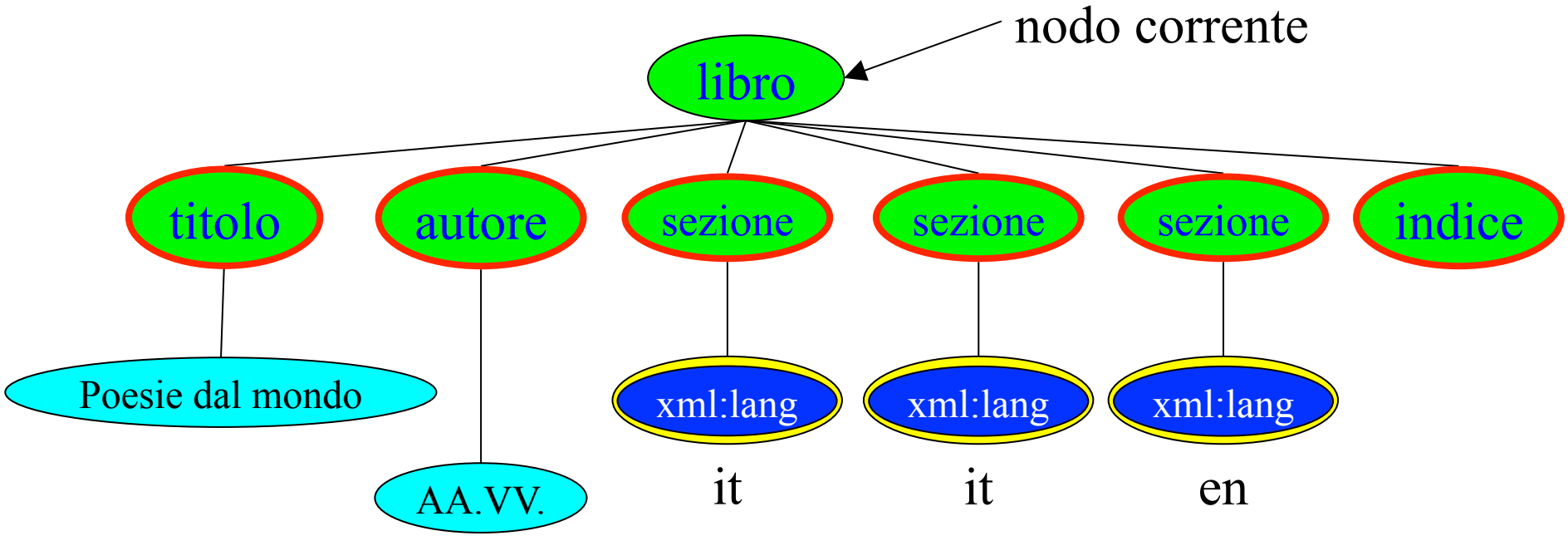
Struttura di uno step

- Uno step di una Path Expression può essere composto da tre parti principali:
  - Un Asse, che seleziona nodi sulla base della propria posizione rispetto al context node (nell'esempio, i figli – **child::**).
  - Un test che filtra questi nodi sulla base del nome o del tipo (nell'esempio, **sezione**).
  - Uno o più predicati, che filtra ulteriormente i nodi sulla base di criteri più generali (nell'esempio, il fatto di **non essere il primo figlio**).



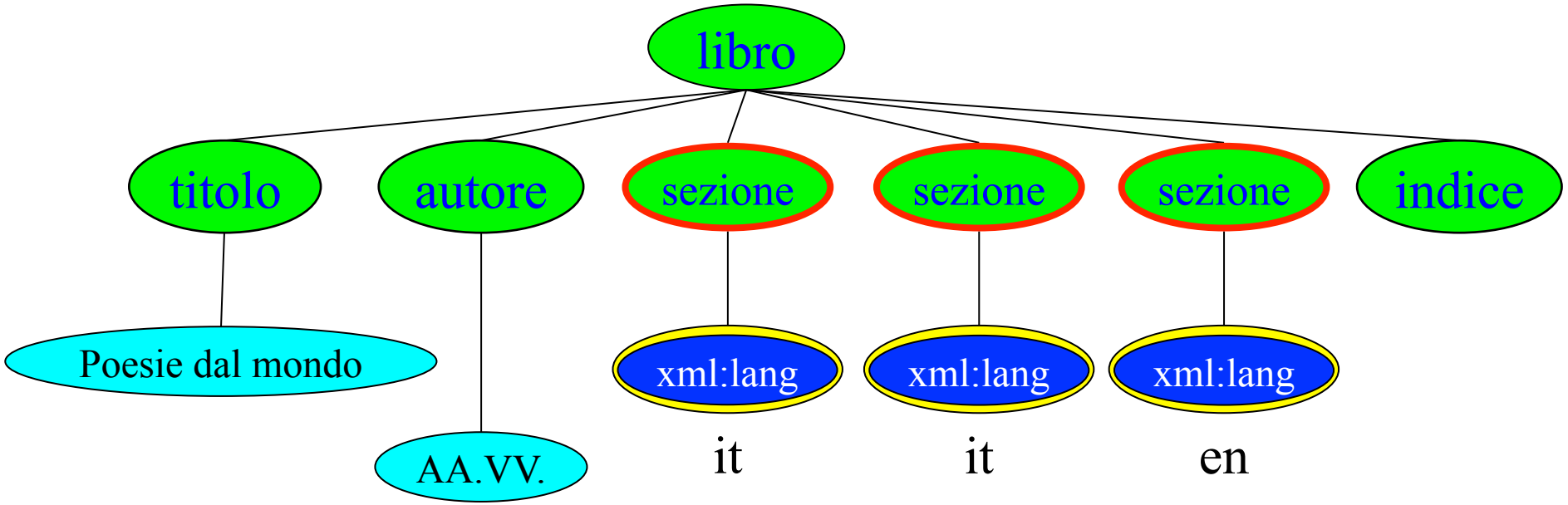
Esempio di valutazione di uno step (1)

child::sezione[attribute::xml:lang = 'it']



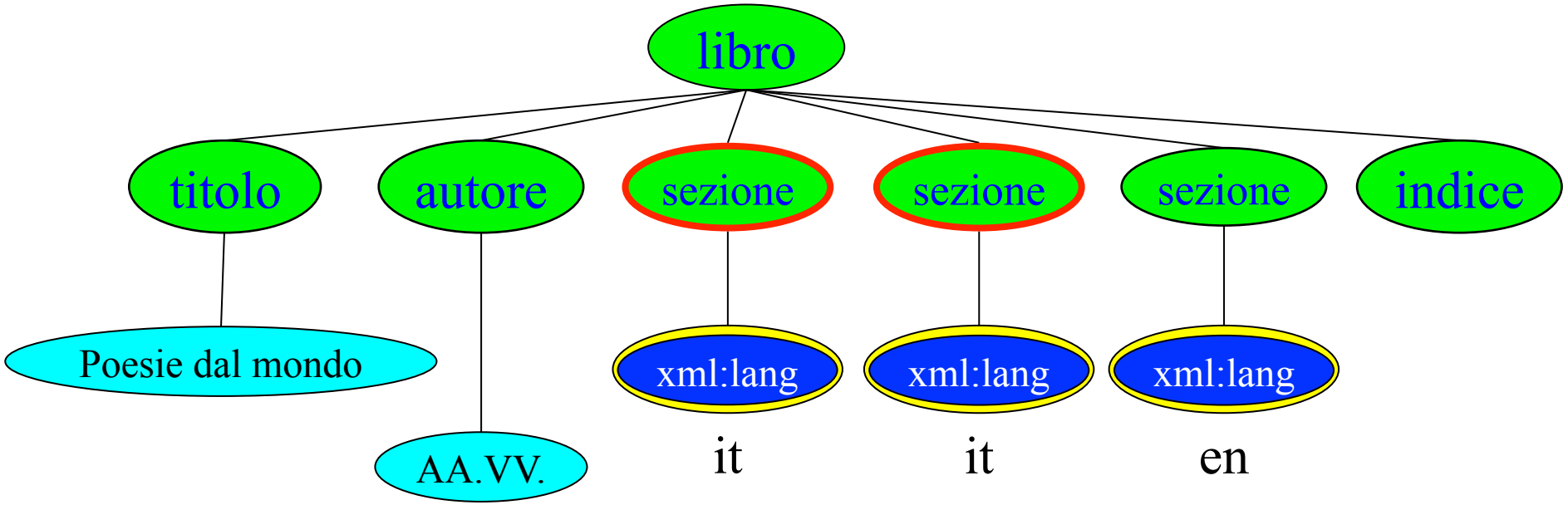
Esempio di valutazione di uno step (2)

child::sezione[attribute::xml:lang = 'it']



Esempio di valutazione di uno step (3)

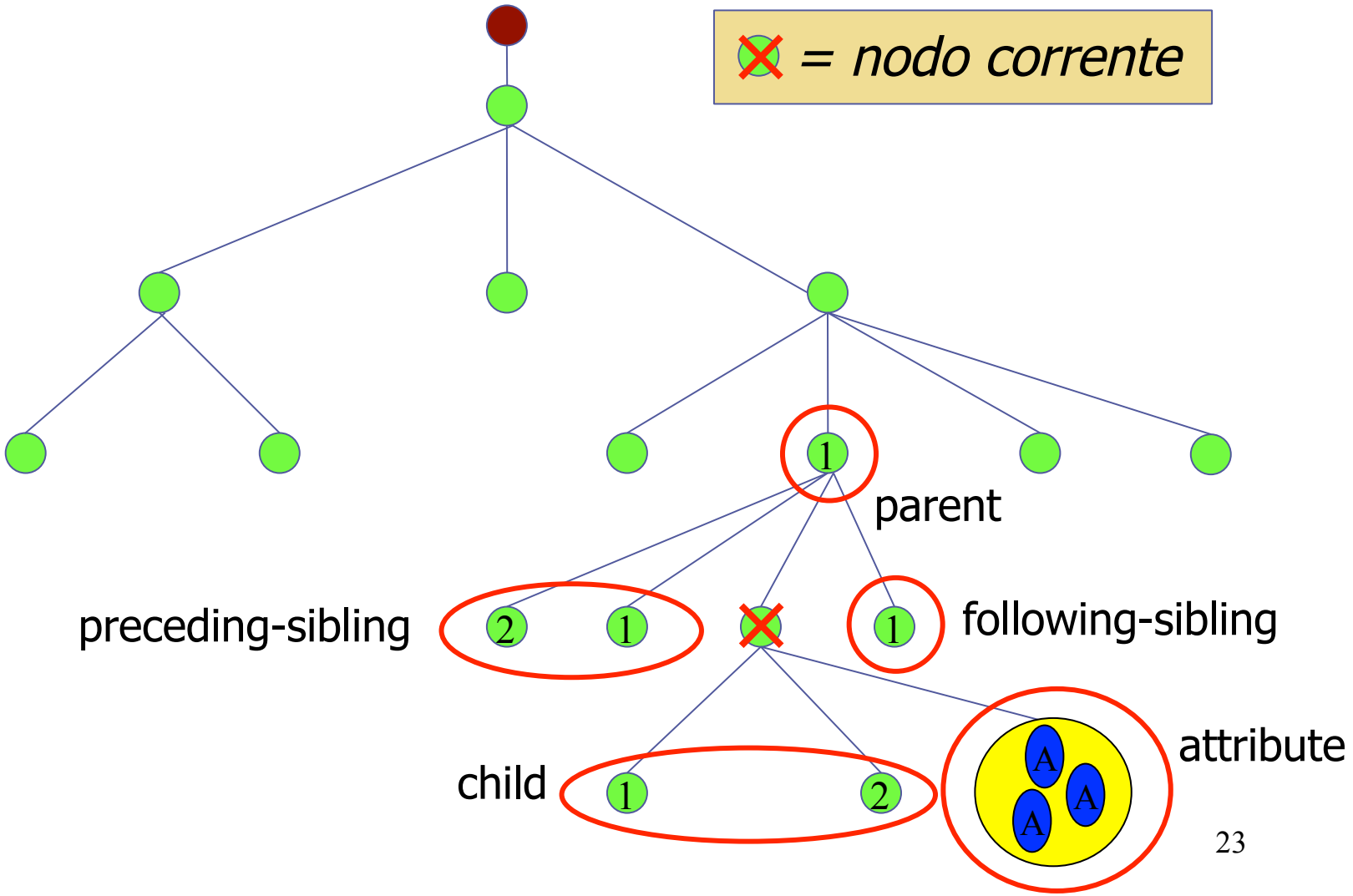
child::sezione[attribute::xml:lang = 'it']



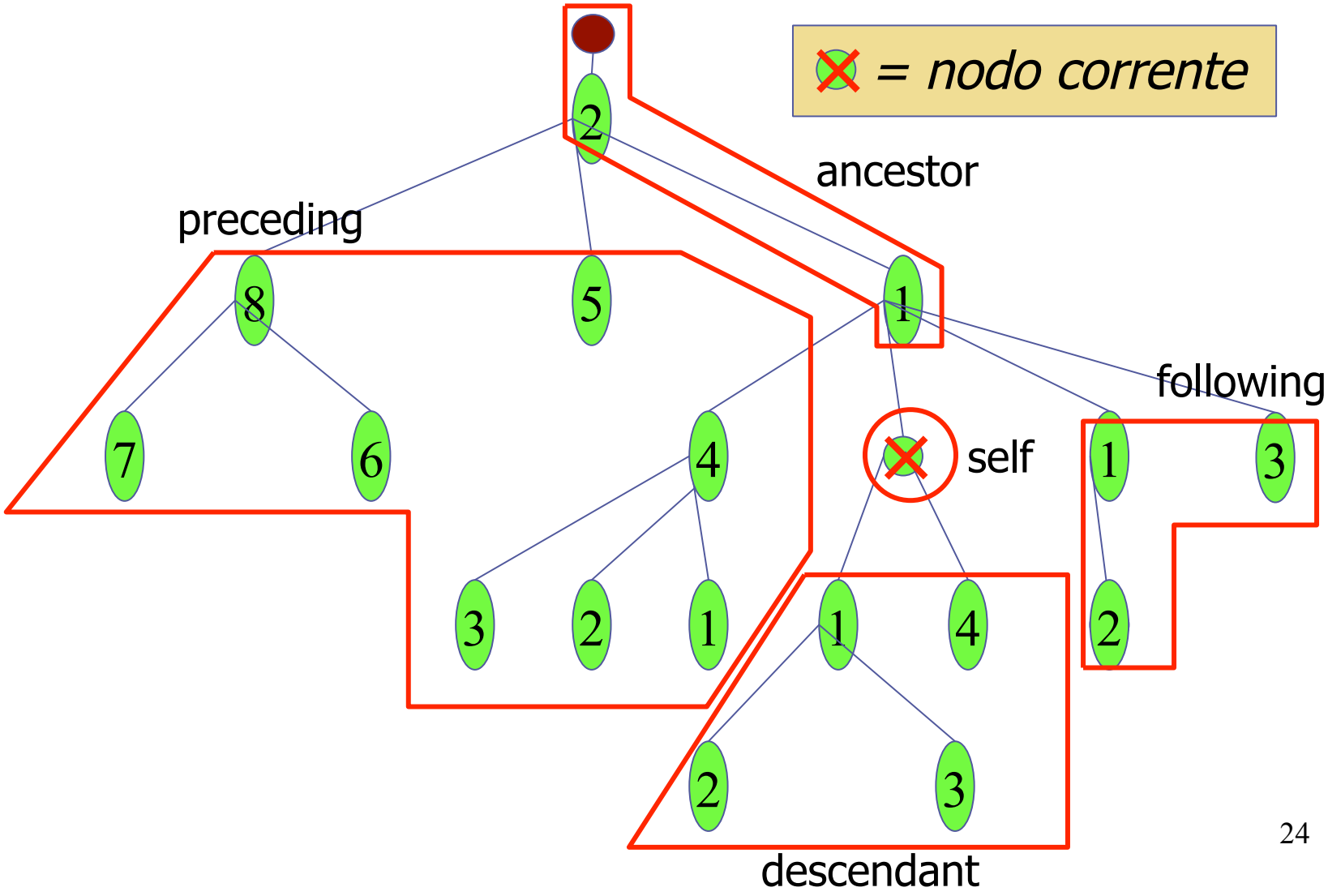
Assi

- Gli assi principali definiti in XQuery (e XPath) sono i seguenti, esemplificati nelle prossime diapositive:
  - self::
  - child::
  - parent::
  - ancestor::
  - descendant::
  - following-sibling::
  - preceding-sibling::
  - attribute::
- Vi sono in aggiunta gli assi composti: descendant-or-self:: e ancestor-or-self::

Esempi di Assi (1)



Esempi di Assi (2)

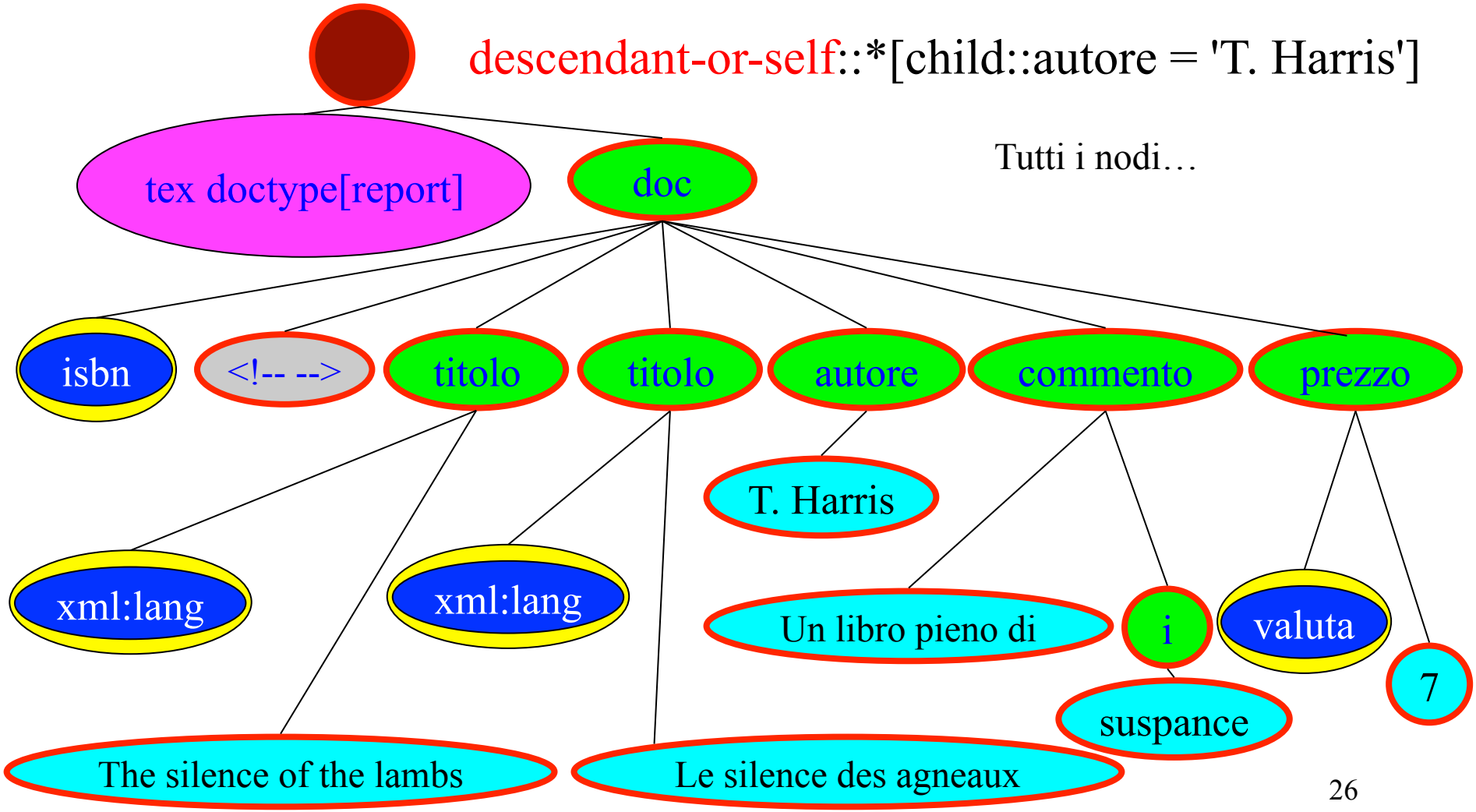


Test sui nomi/tipi di nodo: esempi

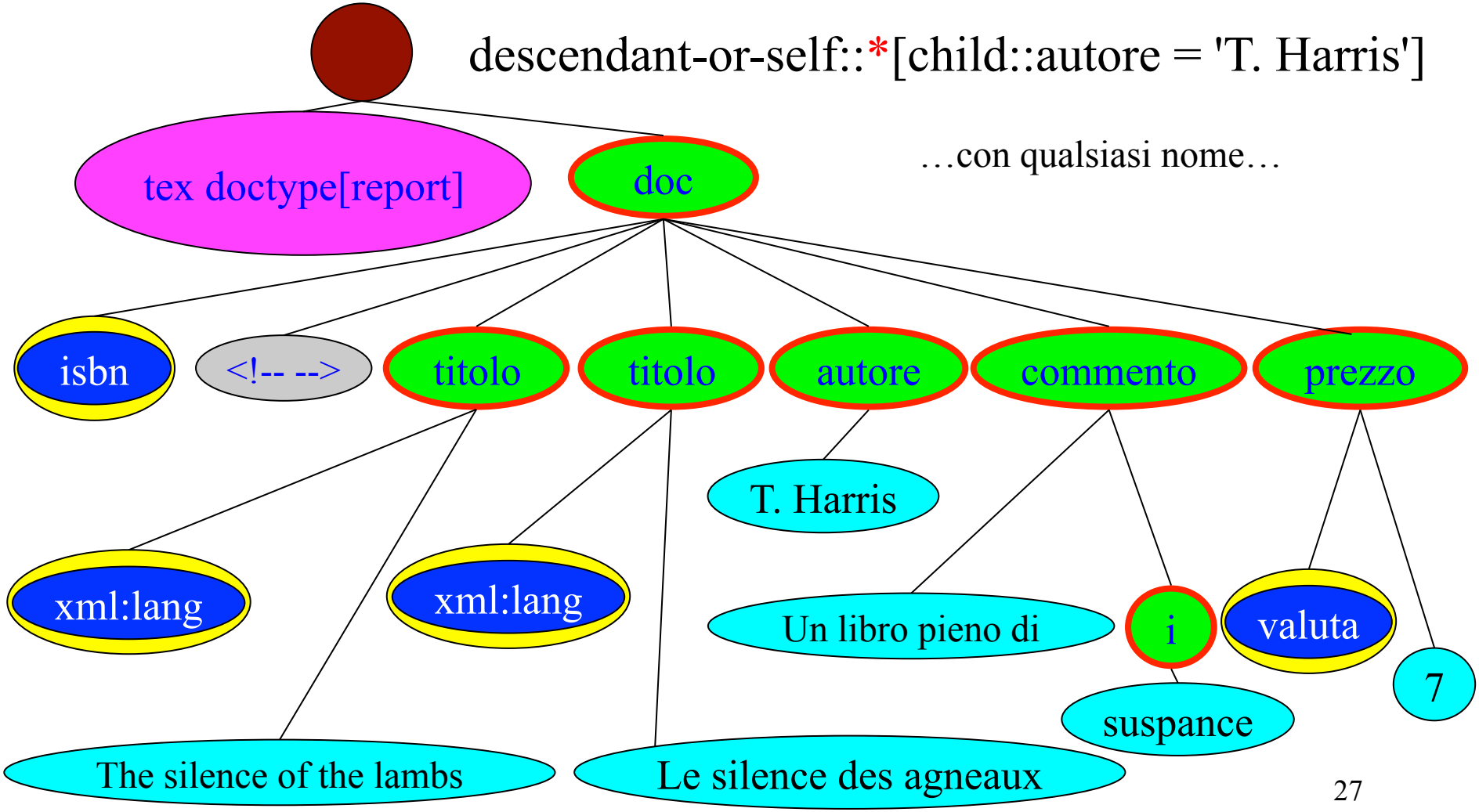
- Il secondo componente di uno step di navigazione filtra i nodi selezionati dall'asse verificandone il nome:
  - **child::section** ritorna solo gli elementi figli con tag `<section>`.
  - **child::*** ritorna tutti gli elementi figli.
  - **attribute::xml:lang** ritorna l'attributo xml:lang.
- Oppure il tipo:
  - **descendant::node()** ritorna tutti i nodi discendenti.
  - **descendant::text()** ritorna tutti i nodi di tipo testo discendenti.
  - **descendant::element()** ritorna tutti i nodi elemento discendenti.
- Oppure entrambi:
  - **descendant::element(persona, xs:decimal)** – elementi persona di tipo decimale.

Un altro esempio di step (1)

descendant-or-self::*[child::autore = 'T. Harris']



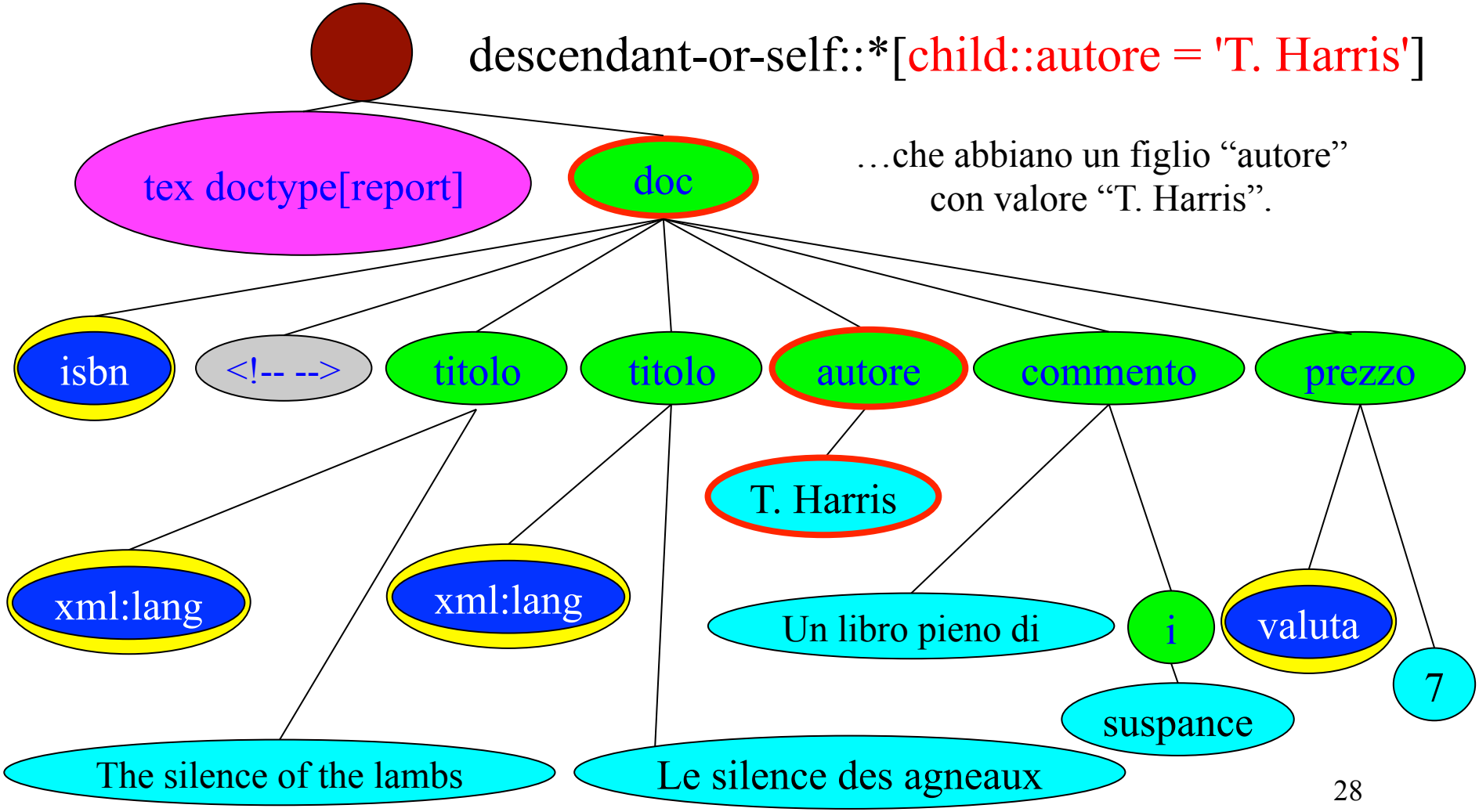
Un altro esempio di step (2)



Un altro esempio di step (3)

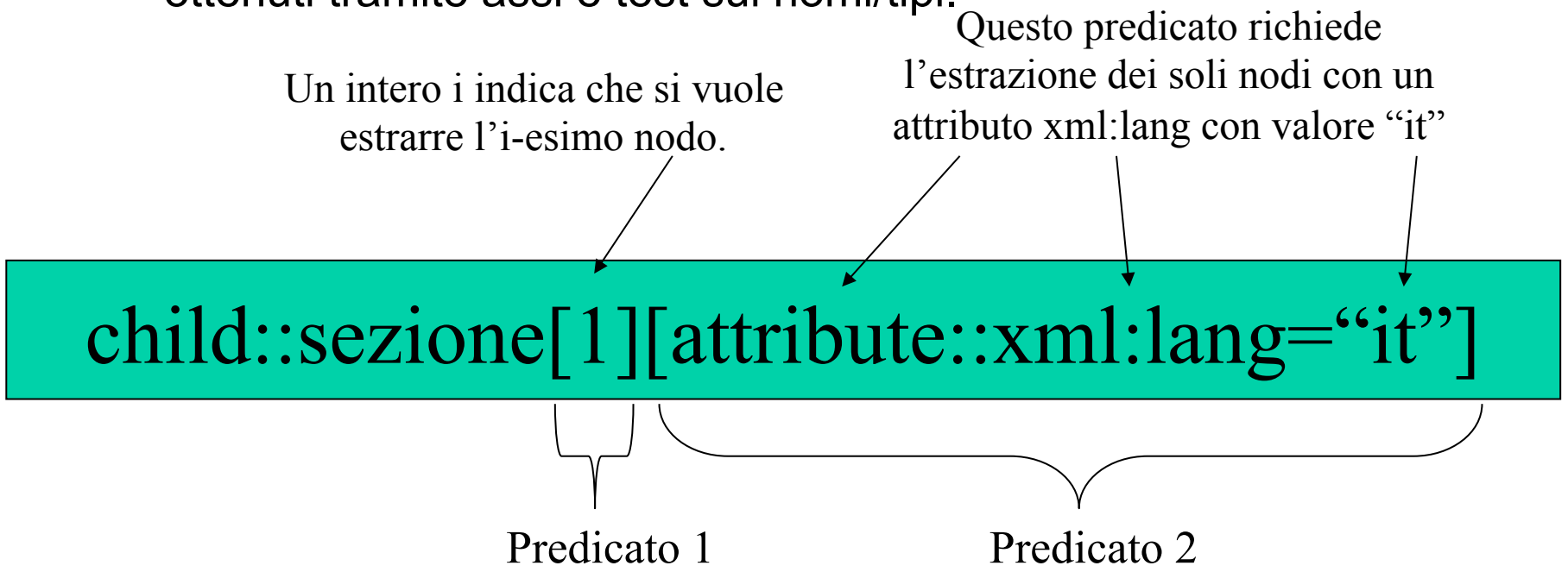
descendant-or-self::*[child::autore = 'T. Harris']

…che abbiano un figlio “autore” con valore “T. Harris”.



Predicati

- Ogni step può essere concluso da uno o più predicati (in and), inclusi tra parentesi quadre, che filtrano ulteriormente i nodi ottenuti tramite assi e test sui nomi/tipi.



Valutazione dei Predicati

- Se l'espressione ritorna un singolo valore intero, essa è vera se la posizione del nodo in oggetto all'interno della sequenza valutata corrisponde al valore.
  - **child::capitolo[2]** restituisce solamente il secondo figlio con tag `<capitolo>`.
- Se l'espressione ritorna una sequenza vuota, il predicato è falso, mentre se il primo item è un nodo ritorna vero.
  - **child::capitolo[child::titolo]** restituisce tutti i figli con tag `<capitolo>` che abbiano almeno un figlio con tag `<titolo>`.
- Altrimenti, vengono seguite le tipiche convenzioni per i predicati:
  - **child::capitolo[attribute::xml:lang = "it"]** restituisce tutti i figli con tag `<capitolo>` che abbiano un attributo xml:lang con valore "it".

Path Expressions complete

- Una path expression può iniziare anche con i seguenti prefissi:
  - Con il carattere /: corrisponde a una sequenza di input dell'espressione che contiene la radice dell'albero.
  - Con i caratteri //: corrisponde a una sequenza di input dell'espressione che contiene tutti i nodi del documento.
- Seguono alcuni esempi di path expression complete:
  - **/descendant::figura[fn:position() = 42]**

    Seleziona la quarantaduesima figura del documento.
  - **/child::book / child::chapter[5] / child::section[2]**

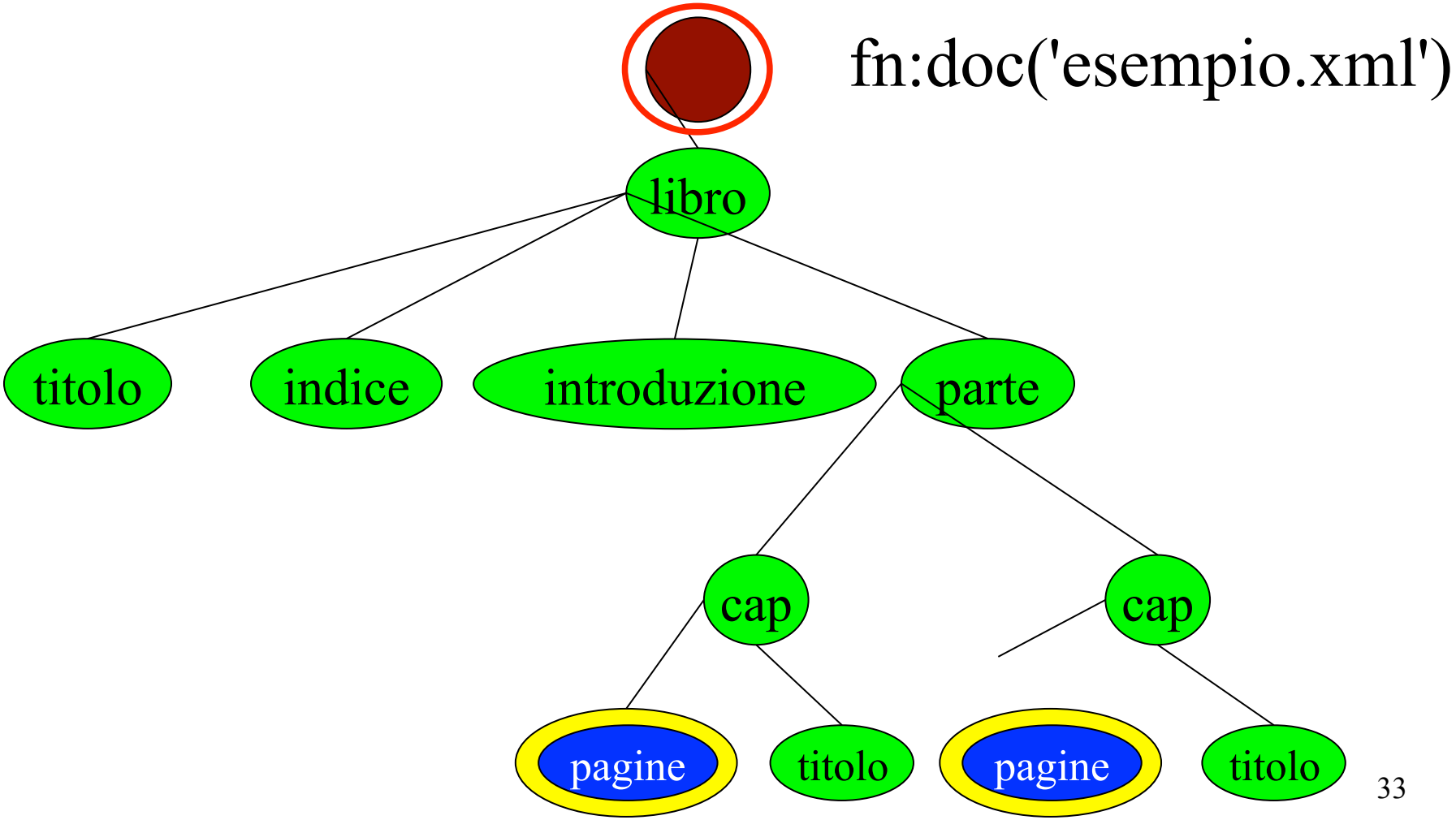
    Seleziona la seconda sezione del quinto capitolo.
  - **//self::chapter[child::titolo]**

    Seleziona tutti i capitoli che hanno almeno un figlio `<titolo>`.

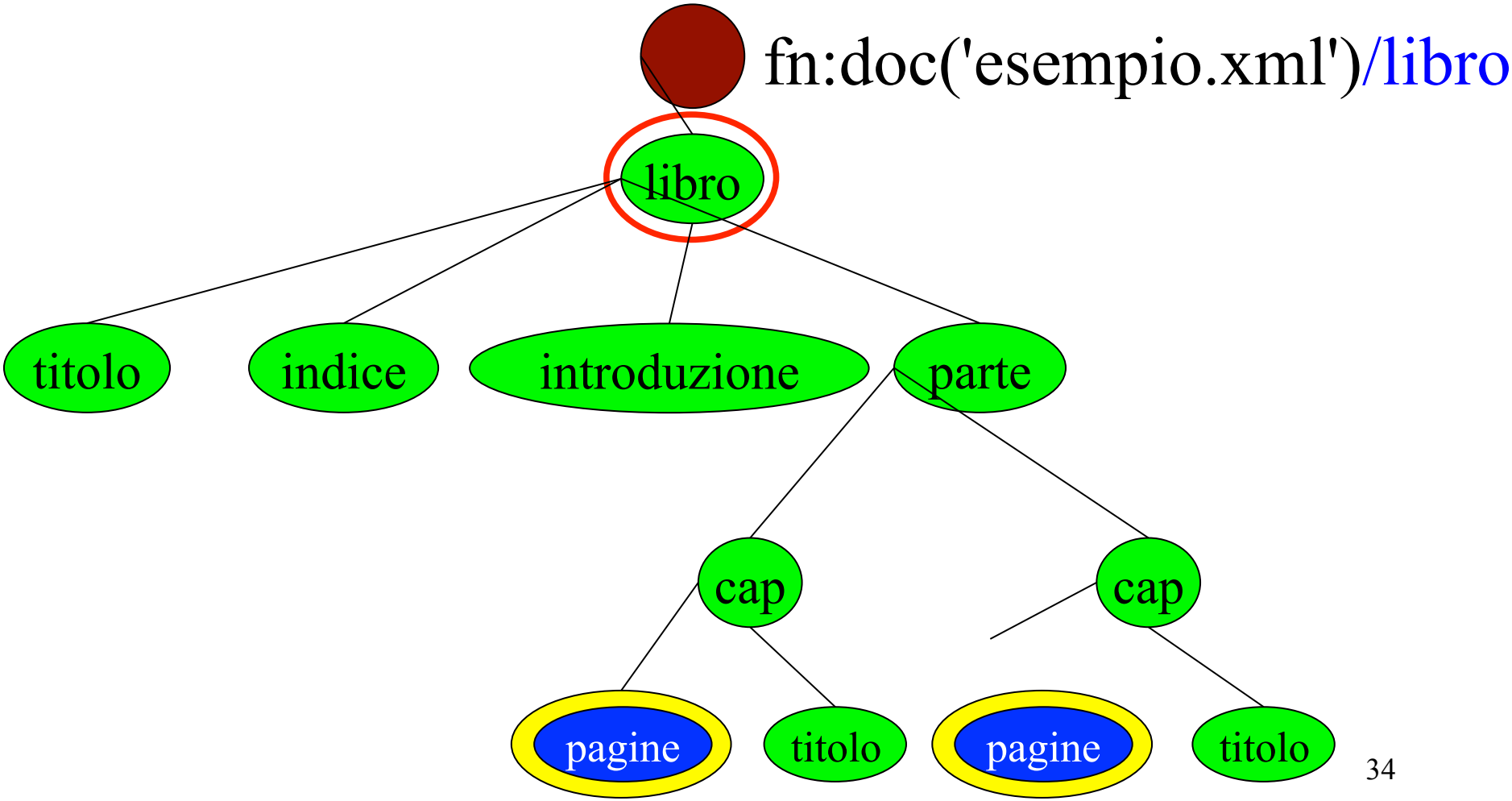
Sintassi abbreviata

- Per scrivere espressioni più compatte, sono possibili alcune abbreviazioni:
  - Omissione dell'asse child::, ad esempio:
    child::sezione/child::paragrafo -> sezione/paragrafo
  - Sostituzione dell'asse attribute:: con il carattere @, ad esempio:
    para[attribute::tipo="warning"] -> para[@tipo="warning"]
  - Sostituzione di descendant-or-self::node() con doppio slash (//):
    div/descendant-or-self::node()/child::paragrafo -> div//paragrafo
  - Sostituzione di self::node() con un punto (.):
    self::node()/descendant-or-self::node()/child::para -> .//para
  - Sostituzione di parent::node() con due punti (..):
    parent::node()/child::sezione -> ../sezione

Esempi (1)

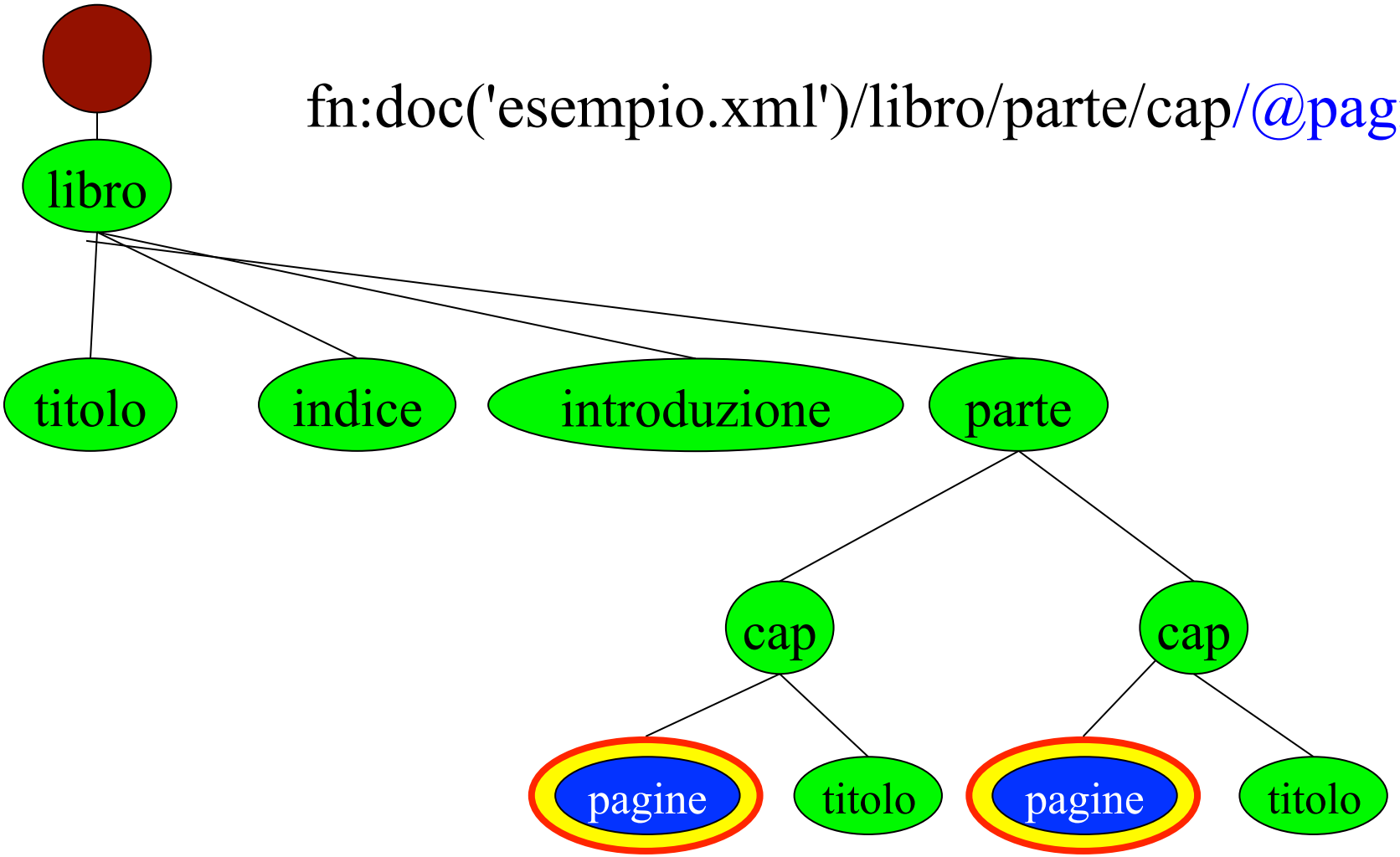

fn:doc('esempio.xml')
libro
titolo
indice
introduzione
parte
cap
cap
pagine
titolo
pagine
titolo

Esempi (2)


fn:doc('esempio.xml')/libro
libro
titolo
indice
introduzione
parte
cap
cap
pagine
titolo
pagine
titolo

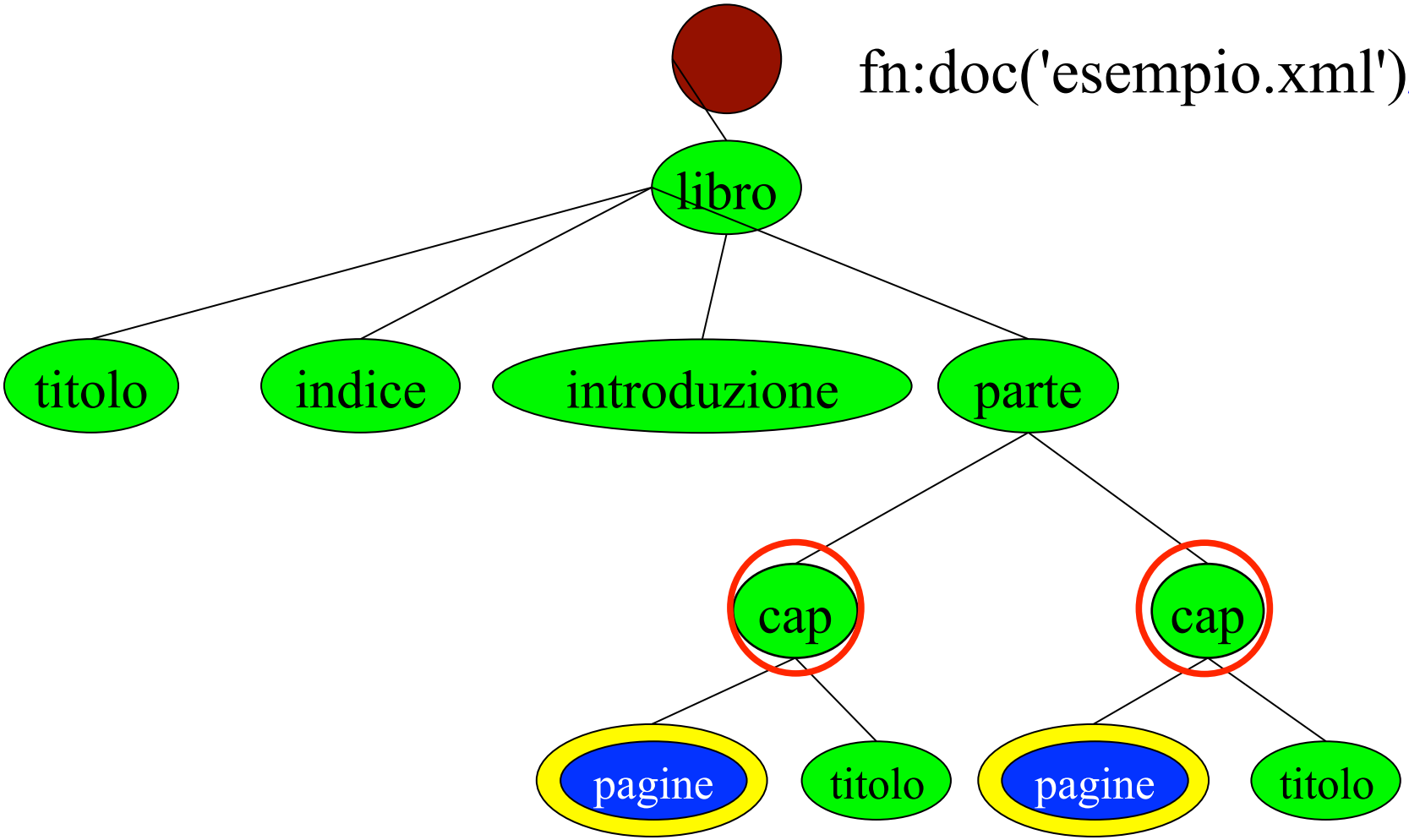
Esempi (3)

fn:doc('esempio.xml')/libro/parte/cap/@pagine



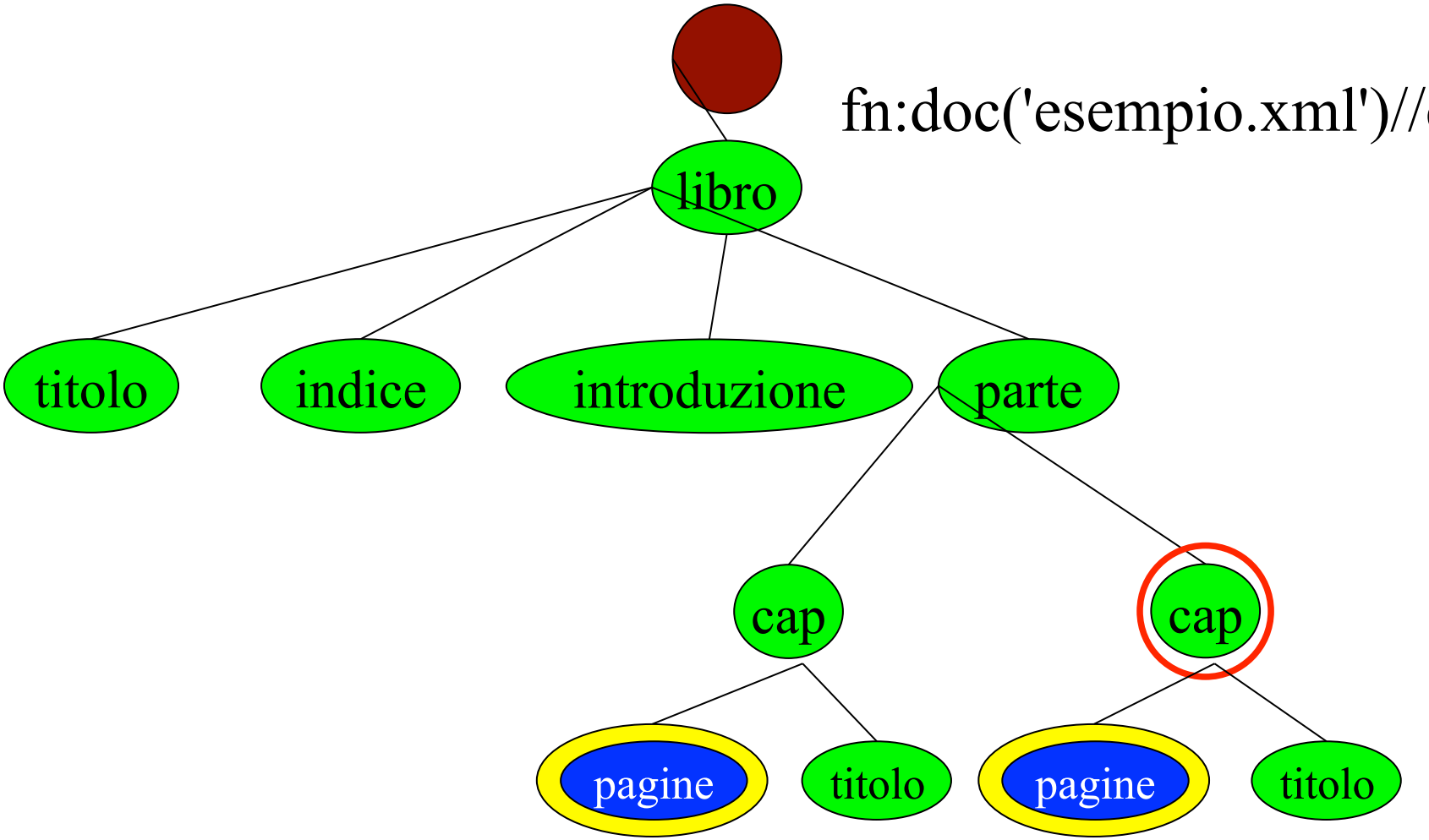
Esempi (4)

fn:doc('esempio.xml')//cap



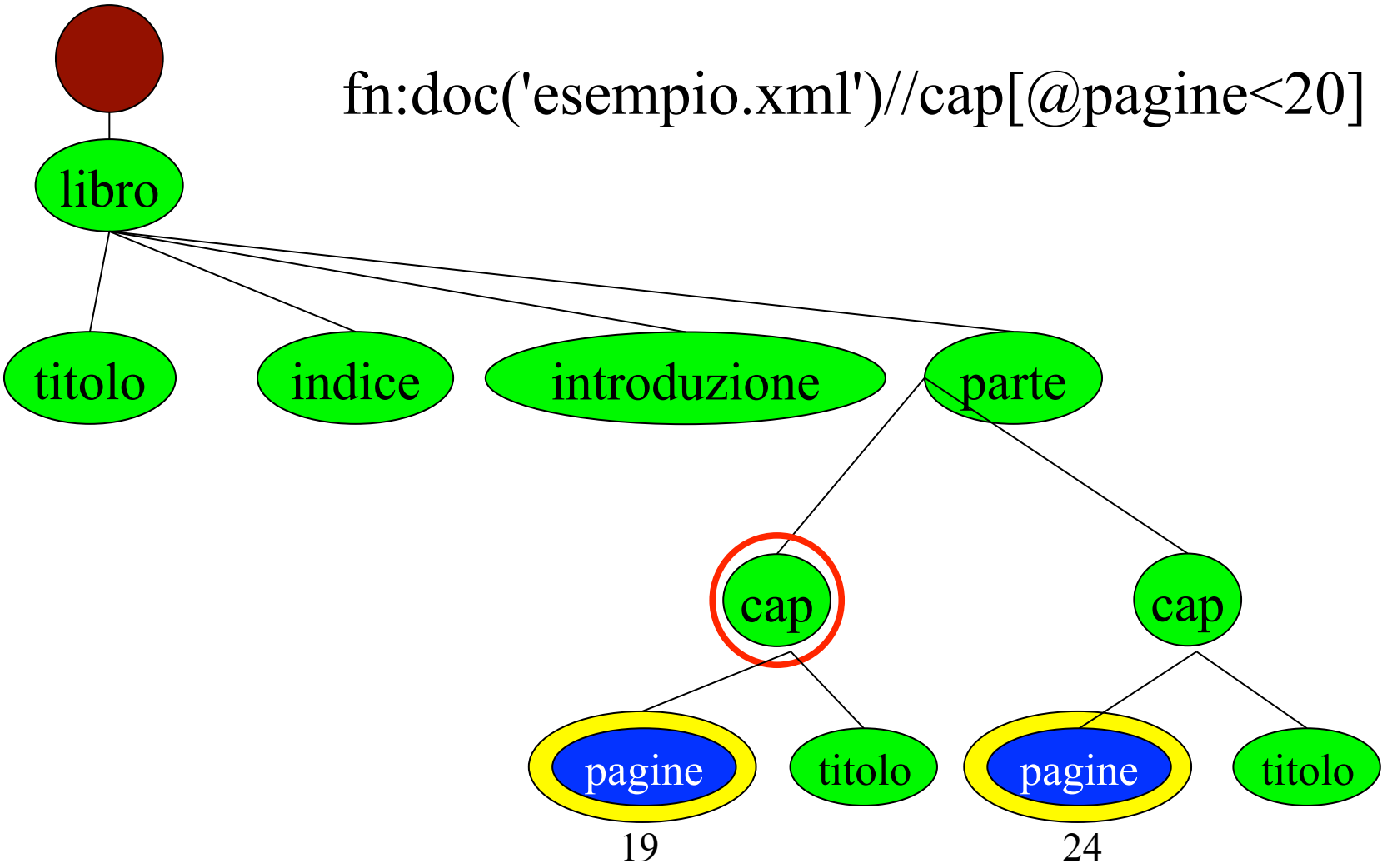
Esempi (5)

fn:doc('esempio.xml')//cap[2]



Esempi (6)

fn:doc('esempio.xml')//cap[@pagine<20]



Espressioni FLWOR

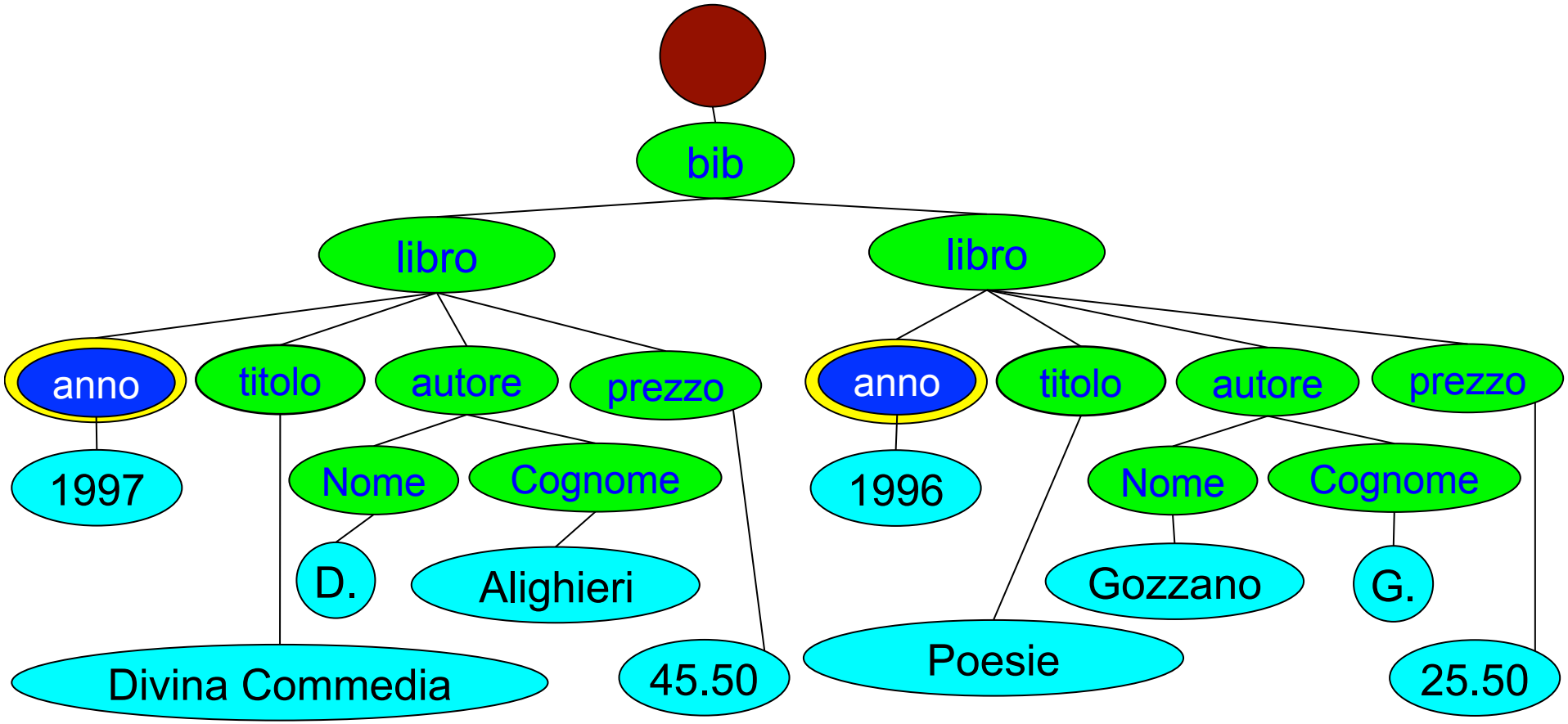
Espressioni FLWOR

- Una espressione FLWOR (pronuncia “flower”) è simile ad uno statement SQL Select-From-Where; è però definita in termini di binding di variabili. E’ composta da 5 parti, alcune delle quali opzionali:
  - **For**: associa una o più variabili ad espressioni.
  - **Let**: costruisce un alias dell’intero risultato di un’espressione;
    - For e Let creano un elenco con tutte le associazioni possibili, chiamate anche “tuple” nella specifica di XQuery.
  - **Where**: filtra l’elenco di associazioni in base ad una condizione;
  - **Order by**: ordina l’elenco di associazioni;
  - **Return**: costruisce il risultato dell’espressione FLWOR.
- Queste espressioni, così come ogni altra espressione XQuery, possono essere commentate utilizzando i simboli (: e :)

  (: Questo è un commento (: con un commento annidato… :) :)

Iterazione di elementi – clausola for (1)

- Per ogni libro, elenca l'anno e il titolo.



Iterazione di elementi – clausola for (2)

- Prima selezioniamo tutti i libri:

  doc("esempio.xml")/bib/libro

- Poi **per ogni libro**, che associamo alla variabile $b,

  for $b in doc("esempio.xml")/bib/libro

- Scriviamo il risultato:

  return

  <libro anno="{*Estrazione dell'anno di $b, in XQuery*}">

  {*Estrazione del titolo di $b, in XQuery*}

  </libro>

Iterazione di elementi – clausola for (3)

```
for $b in doc("esempio.xml")/bib/libro
return
<libro anno="{ $b/@anno }">
    { $b/titolo }
</libro>
```

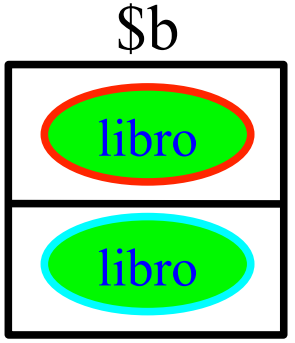
All'interno della clausola RETURN puo' essere specificata una qualsiasi espressione XQuery.
In questo caso, stiamo utilizzando **costruttori** che generano XML, e che si scrivono direttamente utilizzando la sintassi XML.

Le parentesi graffe delimitano un'espressione XQuery, che deve essere valutata per creare il risultato

Iterazione di elementi – clausola for (4)

```
for $b in doc("esempio.xml")/bib/libro
return
<libro anno="{ $b/@anno }">
    { $b/titolo }
</libro>
```

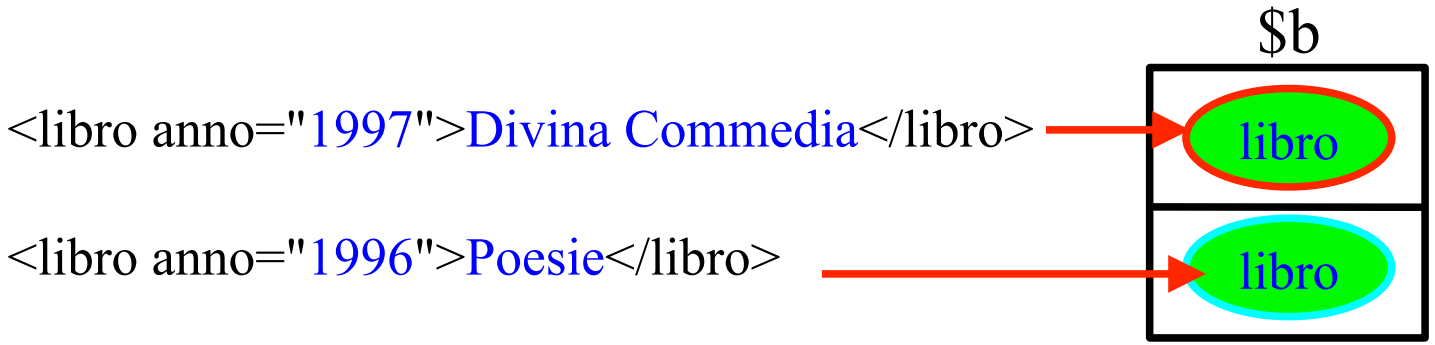
Valuto questa espressione, e ottengo una sequenza di nodi (che vengono associati alla variabile $b):



Iterazione di elementi – clausola for (5)

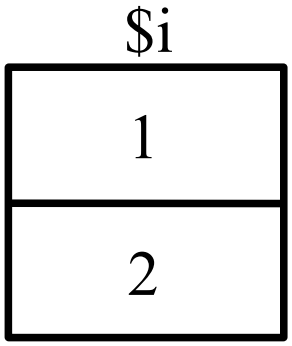
```
for $b in doc("esempio.xml")/bib/libro
return
<libro anno="{ $b/@anno }">
    { $b/titolo }
</libro>
```

Per ogni associazione ($b) valuto questa parte dell' espressione:



Altri esempi (1)

- for $i in (1, 2) return $i + 1
- La sequenza in input è composta da 1 e 2.
- Per ogni $i, viene valutato ($i + 1).
- Il risultato è (2, 3).



Altri esempi (2)

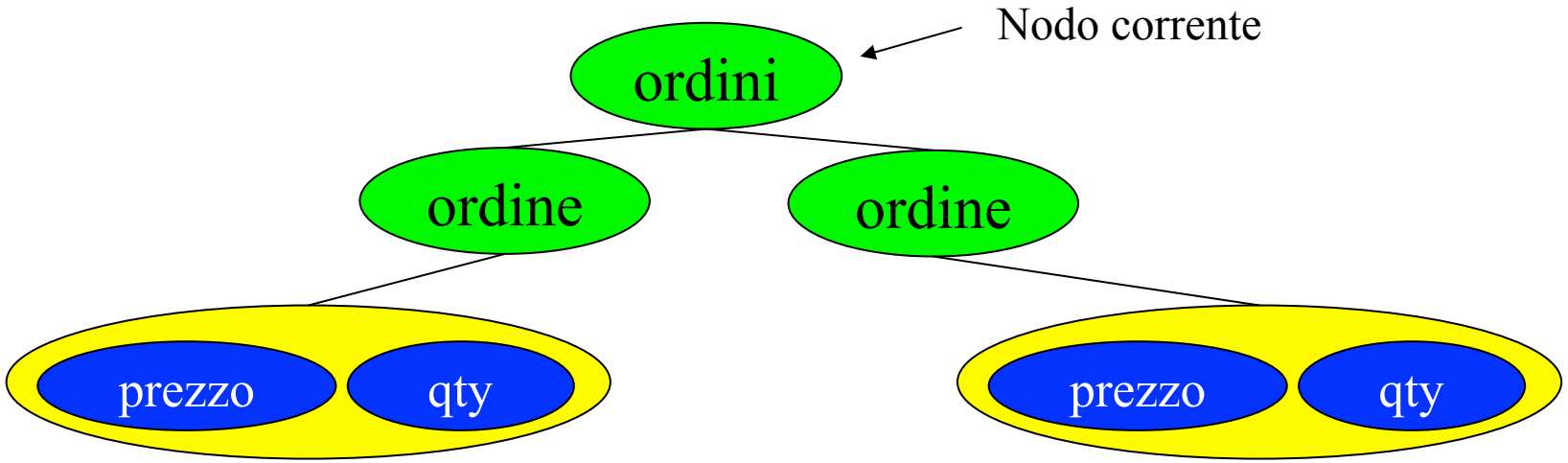
- Più variabili possono essere utilizzate in un'unica espressione.
- for $i in (10, 20), $j in (1, 2) return ($i + $j)
  restituisce (11, 12, 21, 22)

$i	$j
10	1
10	2
20	1
20	2

L'elenco di associazioni è dato dal prodotto cartesiano dei possibili valori delle variabili.

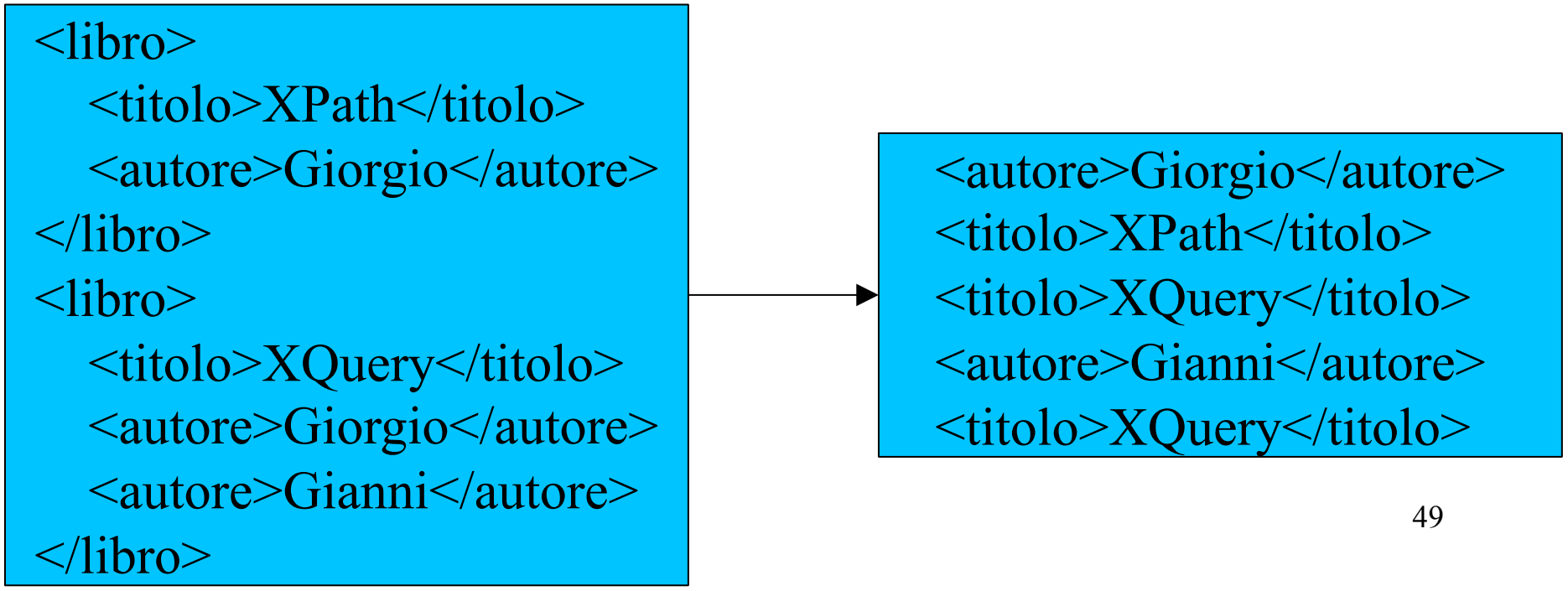
Altri esempi (3)

- Calcola il prezzo totale degli ordini:
- sum(for $i in ordine return $i/@prezzo * $i/@qty)



Altri esempi (4)

- Per ogni autore, elenca i suoi libri (distinct-values è equivalente alla clausola distinct di SQL):
- for $a in distinct-values(//autore) return
  ($a, for $b in //libro[autore = $a] return $b/titolo)



FOR e variabili posizionali (1)

- Oltre ad associare una variabile a piu' valori, puo' talvolta essere utile ricordare a che posizione corrisponde ogni valore.
- Mostriamo le tuple (associazioni) prodotte da:

  for $auto at $i in ("Ford", "Ferrari"),
  $animali at $j in ("Cat", "Dog")

$i	$auto	$j	$animali
1	Ford	1	Gatto
1	Ford	2	Cane
2	Ferrari	1	Gatto
2	Ferrari	2	Cane

FOR e variabili posizionali (2)

- Il predicato at puo' essere utile ad esempio nel caso seguente:
- Seleziona da una sequenza di elementi $seq solo gli elementi pari.

```
for $x at $i in $seq
where $i mod 2 = 0
return $x
```

Iterazioni con filtro – clausola where

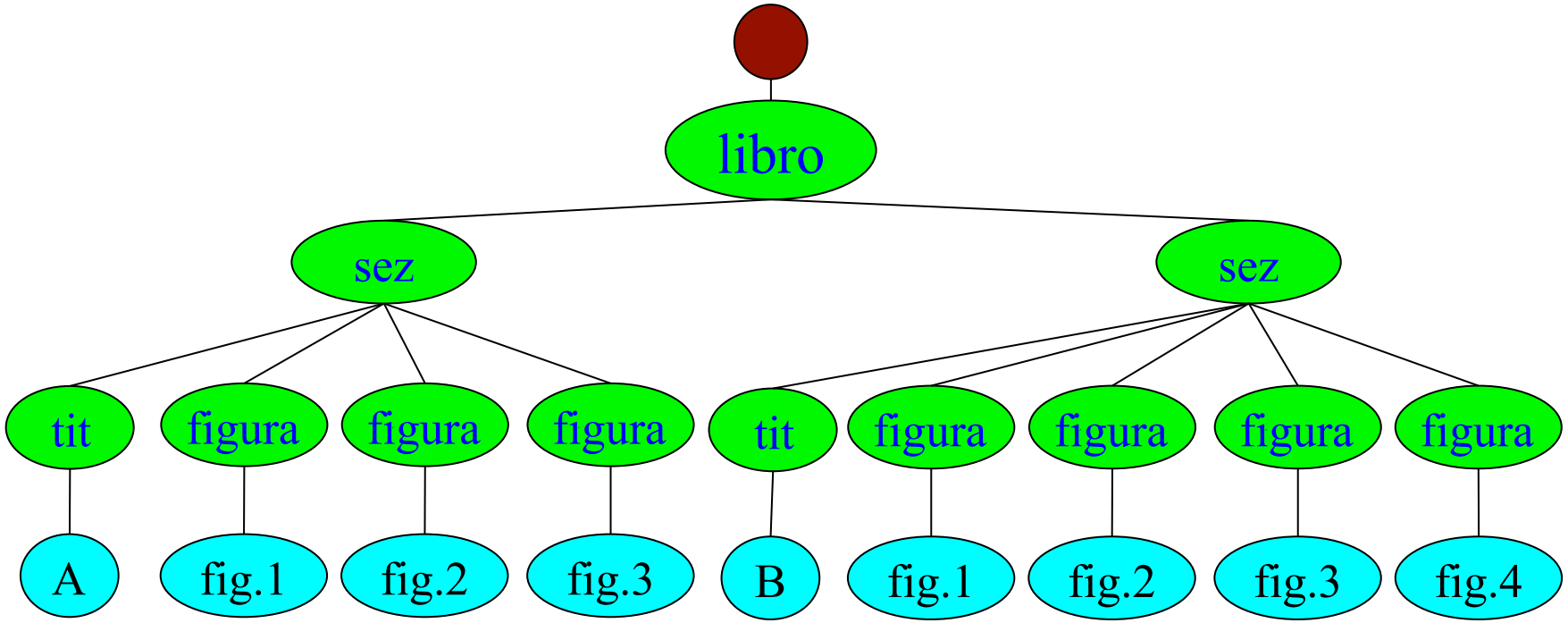
- Trova i libri pubblicati da Rizzoli dopo il '91.

```
for $b in doc("esempio.xml")/bib/libro
where $b/edizione = "Rizzoli"
          and $b/@anno > 1991
return
    <libro>
        { $b/titolo }
    </libro>
```

Rispetto all'esempio precedente, viene aggiunto un filtro.

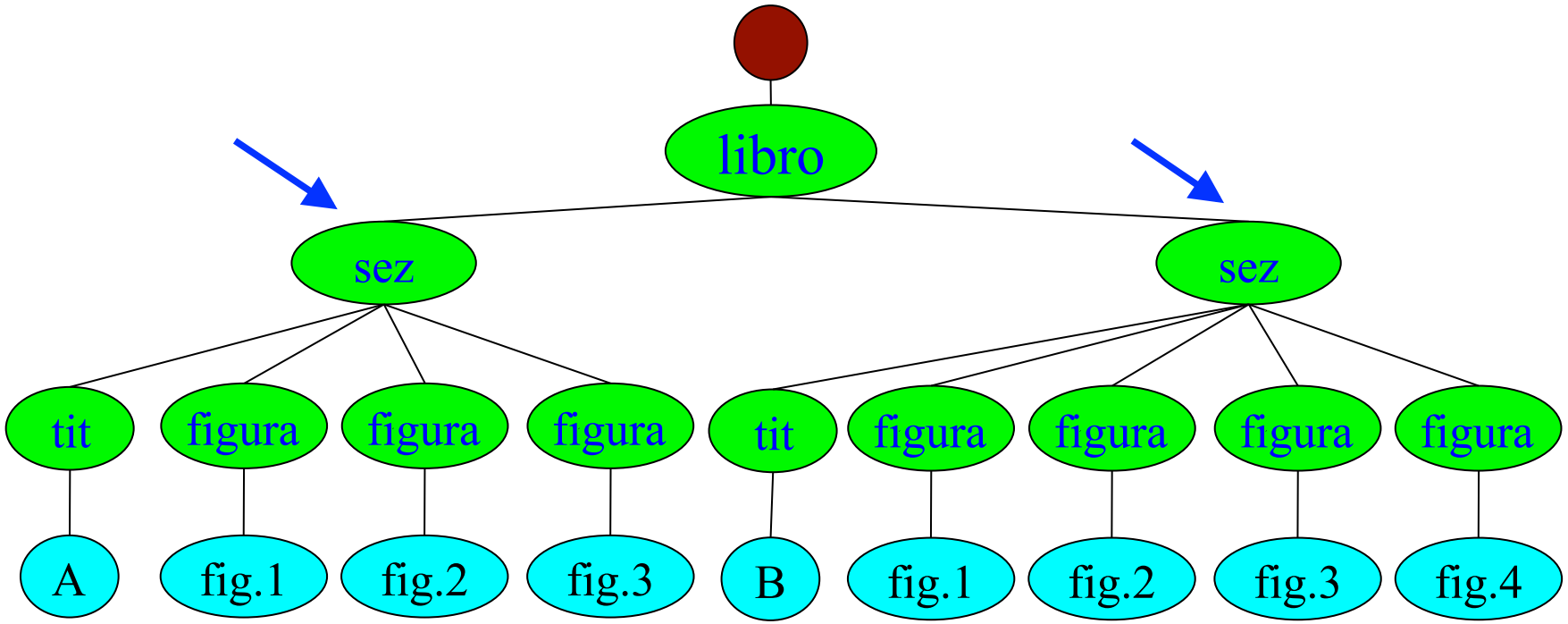
Espressioni con funzioni aggregate – let (1)

- Talvolta e' necessario raggruppare elementi insieme.
- Questo e' necessario per contarli, oppure per calcolarne la media, il minimo o il massimo (se si tratta di numeri).
- Ad esempio, elenca per ogni sezione, il titolo e il numero di figure.



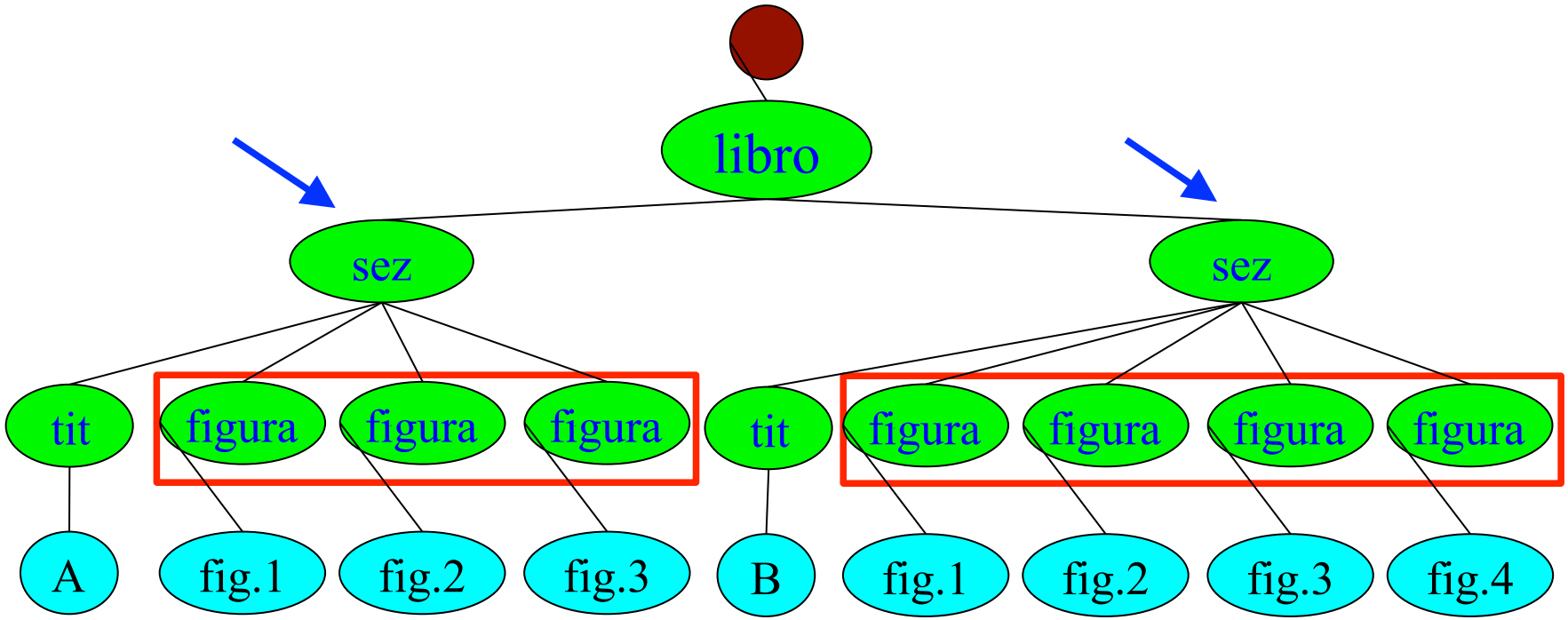
Espressioni con funzioni aggregate – let (2)

- Elenca, per ogni sezione, il titolo e il numero di figure.



Espressioni con funzioni aggregate – let (3)

- Elenca, per ogni sezione, il titolo e il numero di figure.



Espressioni con funzioni aggregate – let (4)

```
<risultato>
 {
   for $s in .//sezione
   let $f := $s/figura
   return
      <sezione titolo="{ $s/titolo/text() }"
      numfig="{ fn:count($f) }"/>
 }
</risultato>
```

.// seleziona tutti i discendenti del nodo corrente

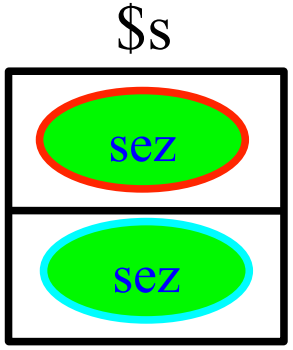
La funzione text() ritorna il valore testuale del nodo

fn: è un namespace che indica che count() è una funzione standard di XQuery

Espressioni con funzioni aggregate – let (5)

```
<risultato>
 {
  for $s in .//sezione
  let $f := $s/figura
  return
     <sezione titolo="{ $s/titolo/text() }"
     numfig="{ count($f) }"/>
 }
</risultato>
```

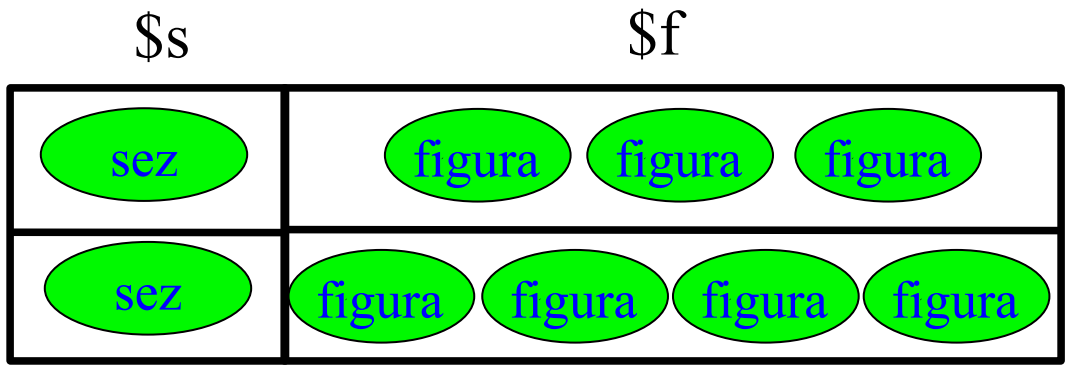
Ogni nodo sezione viene associato a $s.



Espressioni con funzioni aggregate – let (6)

```
<risultato>
 {
  for $s in .//sezione
  let $f := $s/figura
  return
     <sezione titolo="{ $s/titolo/text() }"
     numfig="{ count($f) }"/>
 }
</risultato>
```

Per ogni sezione $s, $f e' uguale **all'insieme** di elementi *figura*.

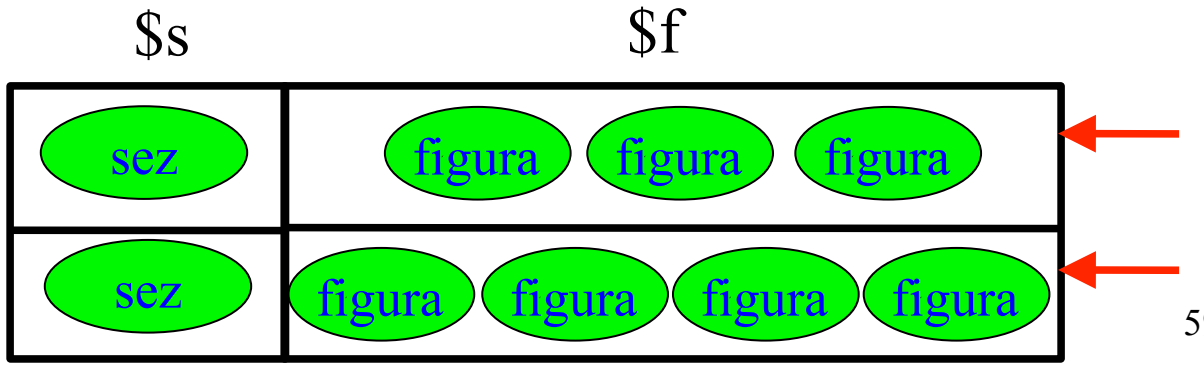


Espressioni con funzioni aggregate – let (7)

```
<risultato>
 {
   for $s in .//sezione
   let $f := $s/figura
   return
      <sezione titolo="{ $s/titolo/text() }"
      numfig="{ count($f) }"/>
 }
</risultato>
```

```
<risultato>
<sezione titolo="A" numfig="3"/>
<sezione titolo="B" numfig="4"/>
</risultato>
```

Questa parte dell'espressione viene valutata una volta per ogni possibile associazione.

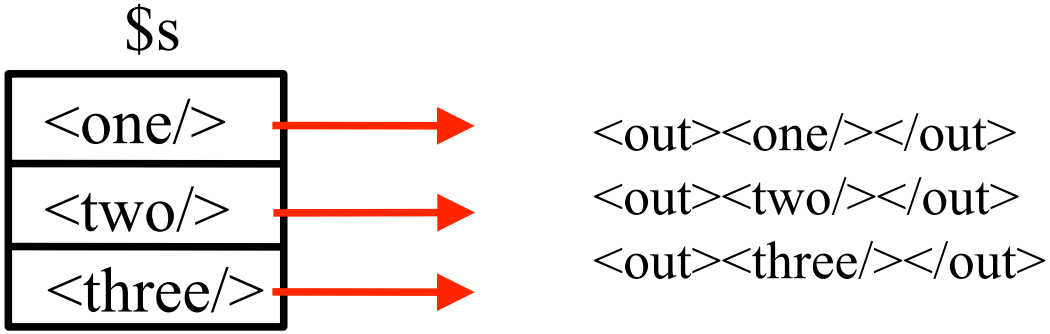


Attenzione alla differenza tra FOR e LET

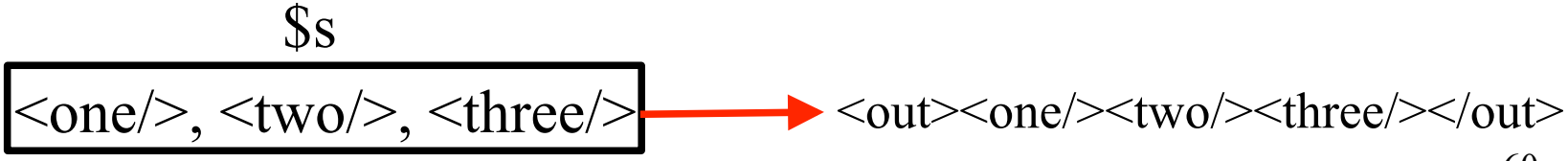
A)
```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

B)
```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

A)



B)



Join in XQuery (1)

- Assumiamo di avere due file XML, uno che descrive il programma di un concerto e un altro che descrive compositori.
- In entrambi i file il compositore e' caratterizzato da un identificatore comune.
- Vogliamo produrre un programma di sala, in cui vengono elencati i brani, con l'autore e le date di nascita e morte.

Join in XQuery (2)

conc.xml

```
<concerto>
  <brano>
     <titolo>Ciaccona</titolo>
     <autore>B001</autore>
  </brano>
  <brano>
     <titolo>Polacca</titolo>
     <autore>C001</autore>
  </brano>
  <brano>
     <titolo>Notturno</titolo>
     <autore>C001</autore>
  </brano>
</concerto>
```

comp.xml

```
<comp>
  <compositore>
     <nome>J.S.Bach</nome>
     <vita>1685-1750</vita>
     <id>B001</id>
  </compositore>
  <compositore>
     <nome>F.Chopin</nome>
     <vita>1810-1849</vita>
     <id>C001</id>
  </compositore>
</comp>
```

Join in XQuery (3)

```
<programma>{
for $comp in fn:doc('comp.xml')/comp/compositore,
    $brano in fn:doc('conc.xml')/concerto/brano
where $comp/id eq $brano/autore
return
  <brano>
     <titolo>{$brano/titolo}</titolo>
     <di>{$comp/nome}</di>
     <date>{$comp/vita}</date>
  </brano>
}</programma>
```

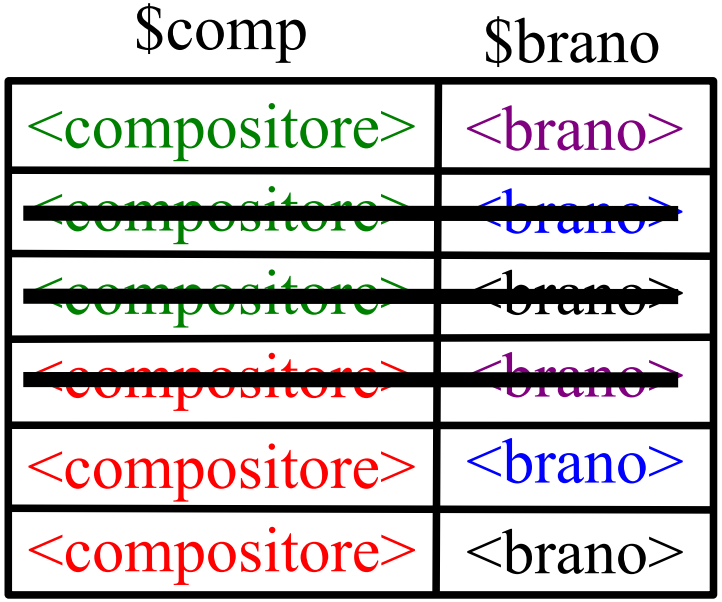
Join in XQuery (4)

```
<programma> {
for $comp in fn:doc('comp.xml')/comp/compositore,
    $brano in fn:doc('conc.xml')/concerto/brano
where $comp/id eq $brano/autore
return
  <brano>
    <titolo>{$brano/titolo}</titolo>
    <di>{$comp/nome}</di>
    <date>{$comp/vita}</date>
  </brano>
}</programma>
```

$comp	$brano
<compositore>	<brano>
<compositore>	<brano>
<compositore>	<brano>
<compositore>	<brano>
<compositore>	<brano>
<compositore>	<brano>

Join in XQuery (5)

```
<programma> {
for $comp in doc('comp.xml')/comp/compositore,
    $brano in doc('conc.xml')/concerto/brano
where $comp/id eq $brano/autore
return
  <brano>
     <titolo>{$brano/titolo}</titolo>
     <di>{$comp/nome}</di>
     <date>{$comp/vita}</date>
  </brano>
}</programma>
```



Risultato dell'interrogazione

```
<programma>
  <brano>
     <titolo>Ciaccona</titolo>
     <di>J.S.Bach</di>
     <date>1685-1750</date>
  </brano>
  <brano>
     <titolo>Polacca</titolo>
     <di>F.Chopin</di>
     <date>1810-1849</date>
  </brano>
   ...
```

Ordinamento – clausola order by (1)

- Come in SQL, XQuery mette a disposizione una clausola per ordinare il risultato di una espressione FLWOR.
- ORDER BY puo' essere specificato insieme a molti parametri che ne alterano la semantica.
- Ne vedremo solamente l'uso piu' comune, tramite alcuni esempi.

Ordinamento – clausola order by (2)

- Gli impiegati associati alla variabile $impiegati vengono restituiti dall'espressione in ordine di salario.

```
for $i in $impiegati
order by $i/salario
return $i/cognome
```

Ordinamento – clausola order by (3)

- Gli impiegati associati alla variabile $impiegati vengono restituiti dall'espressione in ordine di salario dal piu' alto al piu' basso.

```
for $i in $impiegati
order by $i/salario descending
return $i/cognome
```

Ordinamento – clausola order by (4)

- Se si vuole ordinare il risultato di una interrogazione che si potrebbe altrimenti scrivere con una semplice espressione di navigazione, e' necessario utilizzare comunque una espressione FLWOR.

```
for $l in $libri//libro[prezzo < 50]
order by $l/titolo
return $l
```

Altre espressioni

Espressioni condizionali

- I linguaggi di programmazione imperativi (come il C) mettono a disposizione espressioni condizionali, nella forma if-then-else.
- Anche in XQuery si possono utilizzare costrutti di questo tipo.
- In questo modo si possono facilmente esprimere query del tipo: aggiungi un elemento <cognomeDaNubile> se il genere della persona è “F”.

Espressioni condizionali: esempio (1)

- La valutazione di questa espressione ritorna il valore della variabile $prodotto*x* che contiene il prezzo piu' alto.

```
if ($prodotto1/prezzo < $prodotto2/prezzo)
then $prodotto2
else $prodotto1
```

Espressioni condizionali: esempio (2)

- Questa query controlla l'esistenza di un attributo @scontato, e di conseguenza sceglie gli elementi da selezionare.

```
if ($prodotto/@scontato)
then $prodotto/ingrosso
else $prodotto/dettaglio
```

Operatori di confronto (1)

- I seguenti operatori agiscono su **sequenze** composte da piu' item:
- =, !=, <, <=, >, >=
- Ad esempio,

  $libro1/autore = "Pirandello“

  restituisce true se **almeno uno dei nodi** autore selezionati ha valore testuale "Pirandello".
- Questi operatori devono essere utilizzati con estrema cautela, come evidenziato dagli esempi seguenti.

Operatori di confronto (2)

- Il fatto che = e != siano veri se **almeno uno** degli elementi delle sequenze confrontate soddisfa il predicato determina alcuni comportamenti controintuitivi:

Es.1) Questi operatori non sono transitivi:

- (1, 2) = (2, 3)
- (2, 3) = (3, 4)
- (1, 2) != (3, 4)

Es.2) Entrambe le seguenti espressioni ritornano true:

- (1, 2) = (2, 3)
- (1, 2) != (2, 3)

Operatori di confronto (3)

- XPath 2.0 introduce nuovi operatori per confrontare sequenze composte da **un solo** item:
- eq, ne, lt, le, gt, ge
- $libro1/autore eq "Pirandello" e' vero solo se viene selezionato **un unico nodo** autore.
- In caso contrario, viene segnalato un errore.

Operatori logici e aritmetici

- XQuery mette a disposizione i seguenti operatori logici:
- or, and
- Una funzione standard di negazione logica:
- fn:not()
- Ed i seguenti operatori aritmetici:
- +, -, * div, idiv (divisione intera), mod (resto della divisione).
- Ad esempio:
- 1 eq 1 and 1.5 eq 3 div 2
- -1 eq -3 idiv 2 or 2 eq 3

Espressioni con quantificatori (1)

- Una variabile in XQuery puo' essere associata ad un singolo valore. Ad esempio, $prezzo puo' avere un valore intero 12000.
- Spesso le variabili vengono pero' associate a insiemi di oggetti.
- Ad esempio l'espressione

  for $lib in doc(libri.xml)/libri/libro

  puo' associare la variabile $lib a diversi elementi <libro>.
- Sono quindi necessari operatori che verifichino le proprieta' di insiemi di oggetti.

Espressioni con quantificatori (2)

- Introduciamo i due operatori di XQuery che servono a questo scopo con qualche esempio:

  some $imp in //impiegato

  satisfies ($imp/stipendio > 13000)

- Questa espressione e' vera se **almeno un** impiegato percepisce uno stipendio maggiore di 13000.

Espressioni con quantificatori (3)

- Una cosa simile avviene utilizzando l'altro quantificatore: every.

  every $imp in //impiegato

  satisfies ($imp/stipendio > 13000)

- Questa espressione e' vera se **tutti gli** impiegati percepiscono uno stipendio maggiore di 13000.

Espressioni con quantificatori: esempi

```
some $x in (1, 2, 3), $y in (2, 3, 4)
satisfies $x + $y = 4
```

Restituisce true.

```
every $x in (1, 2, 3), $y in (2, 3, 4)
satisfies $x + $y = 4
```

Restituisce false.

Alcune funzioni standard

Funzioni di input

- Servono per ottenere frammenti XML da interrogare.
- Vengono utilizzate per accedere ad un singolo documento (doc) o ad una sequenza di documenti (collection), messa a disposizione tramite il sistema di gestione.
- fn:doc('bib.xml')
- fn:collection('compositori')

Funzioni su sequenze di nodi

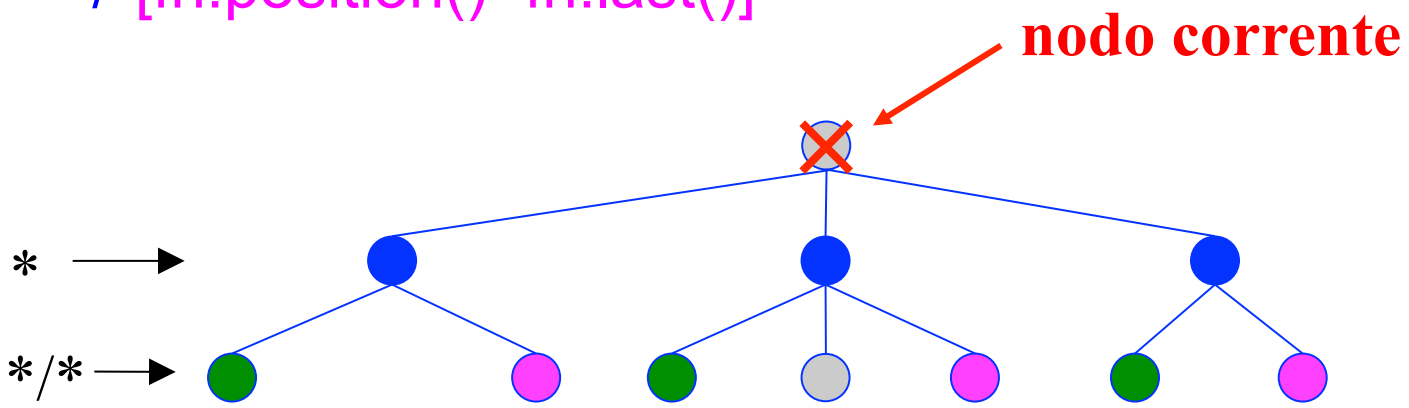
- fn:position()
- Posizione del nodo corrente.

- fn:last()
- Numero di nodi.

- fn:count($elementi)
- Cardinalita' della sequenza di nodi passata come parametro.

Esempio

- Ogni colore mostra i nodi selezionati dalla corrispondente espressione:
- fn:count(*) *restituisce* 3
- */*[fn:position()=1]
- */*[fn:position()=fn:last()]



Funzioni aggregate

- Le seguenti funzioni sono di solito utilizzate insieme al costrutto let, come gia' abbiamo visto.
- count
- avg
- max
- min
- sum

Esempi di funzioni aggregate

- $seq3 = (3, 4, 5)
- count($seq3) ritorna 3
- avg($seq3) ritorna 4
- max($seq3) ritorna 5
- min($seq3) ritorna 3
- sum($seq3) ritorna 12

Riferimenti

- Specifiche di XQuery e link:
  - http://www.w3.org/XML/Query/
- Alcuni sistemi che supportano XQuery (nelle versioni attuali):
  - Oracle database server.
  - DB2.
  - SQL Server.
- Lista con tutte le principali implementazioni di XQuery:
  - http://www.w3.org/XML/Query/#implementations