Storing XML on Relational Tables

Federico Ulliana
GraphIK, LIRMM, INRIA

Slides collected from James Cheney and Sam Idicula

Boston, winter ’99 : XML standardization

Jan 2000 : people wondering …

Now, how can I publish online my relational data?
(XMLAGG – Xperanto)

Feb 2000 : people (again) wondering …

I created my first10GB XML document crawling web data.
Now, how can I query it ?

3 schools for processing XML data

1. Flat streams: store XML data as is in text files
   - query support: limited; fast for retrieving whole documents

2. Native XML Databases: designed specifically for XML
   - XML document stored in XML specific way
   - Goal: Efficient support for XML queries

3. Re-use existing DB storage systems
   - Leverage mature systems (DBMS)
   - How ? Map XML document into flat tables

Why transform XML data into relations?

Native XML databases need:

- storing XML data, indexing,
- query processing/optimization
- concurrency control
- updates
- access control, . . .
- Nontrivial: the study of these issues is still in its infancy – incomplete support for general data management tasks

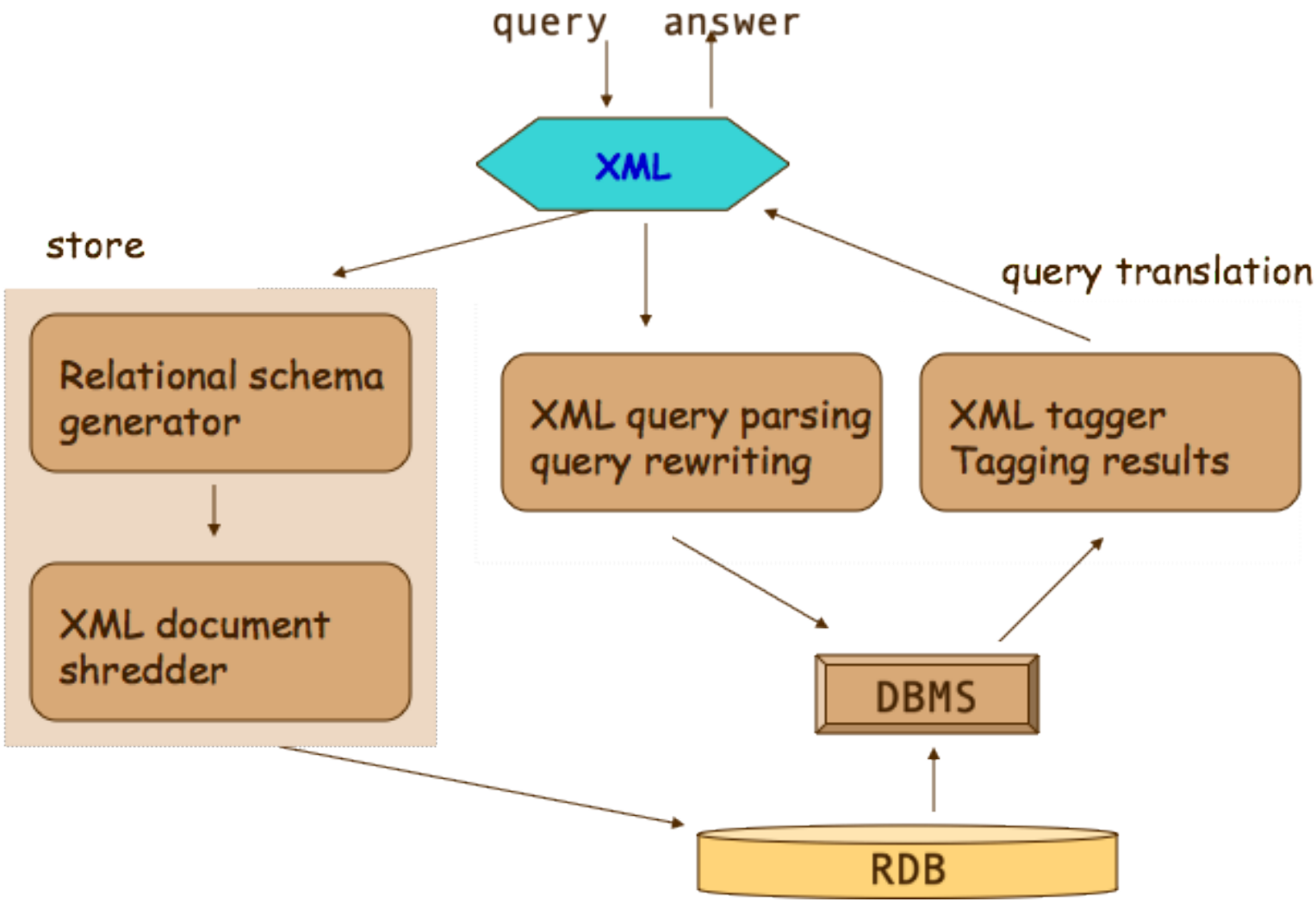
Haven't these already been developed for relational DBMS!?

- Why not take advantage of available DBMS techniques?

From XML to relations :

1. Derive a relational schema
2. Insert XML data into relational tuples
3. Translate XML queries to SQL queries
4. Convert query results back to XML

Architecture



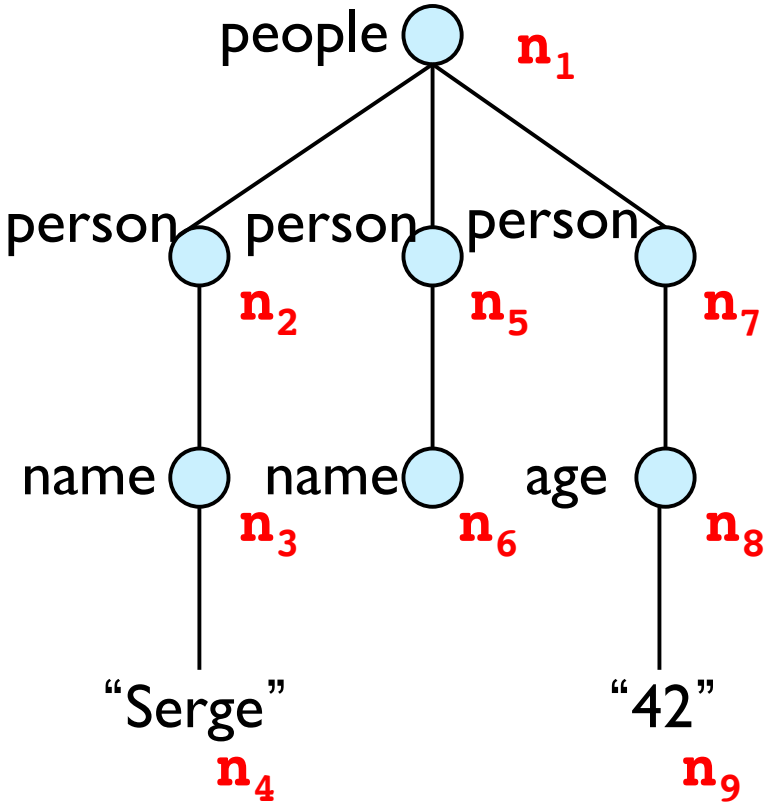
Plan

Schema-unaware

Schema-aware

Commercial solutions

Edges & Values



EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

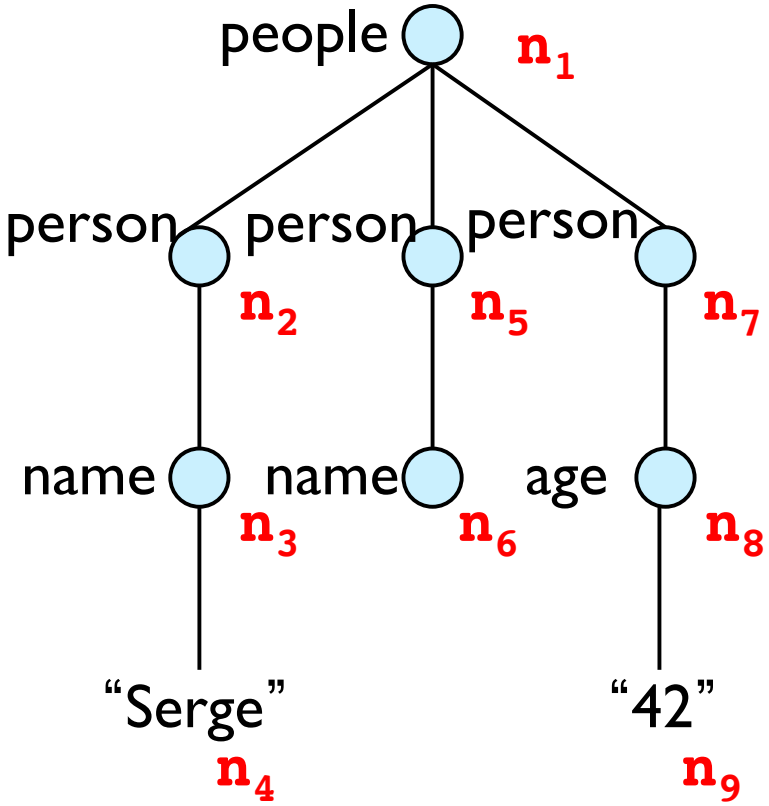
TEXTVALUES

node	value
n_4	Serge

NUMVALUES

node	value
n_9	42

VERTICAL-EDGE + Inline



people

source	target	ordinal	txtval	numval
	n_1			

person

source	target	ordinal	txtval	numval
n_1	n_2	1		
n_1	n_5	2		
n_1	n_7	3		

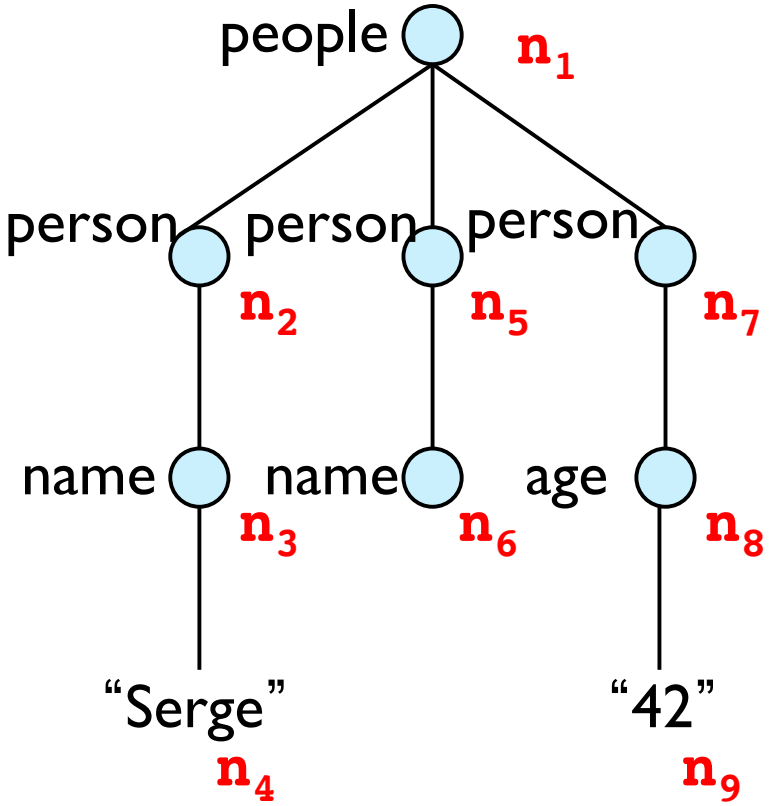
name

source	target	ordinal	txtval	numval
n_2	n_3	1	Serge	
n_5	n_6	1		

age

source	target	ordinal	txtval	numval
n_7	n_8	1		42

MONET storage



people

node	txtval	numval
n_1		

people_person

node	txtval	numval
n_2		
n_5		
n_7		

people_person_name

node	txtval	numval
n_3	Serge	
n_6		

people_person_age

node	txtval	numval
n_8		42

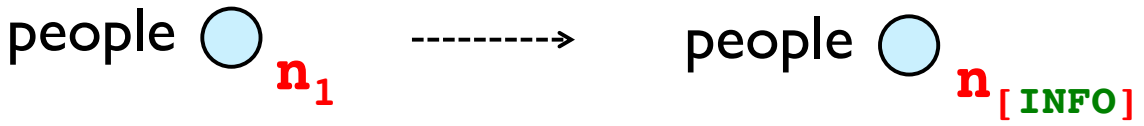
And the remaining axes ?

Maybe we need some new ideas…

INTERVALS

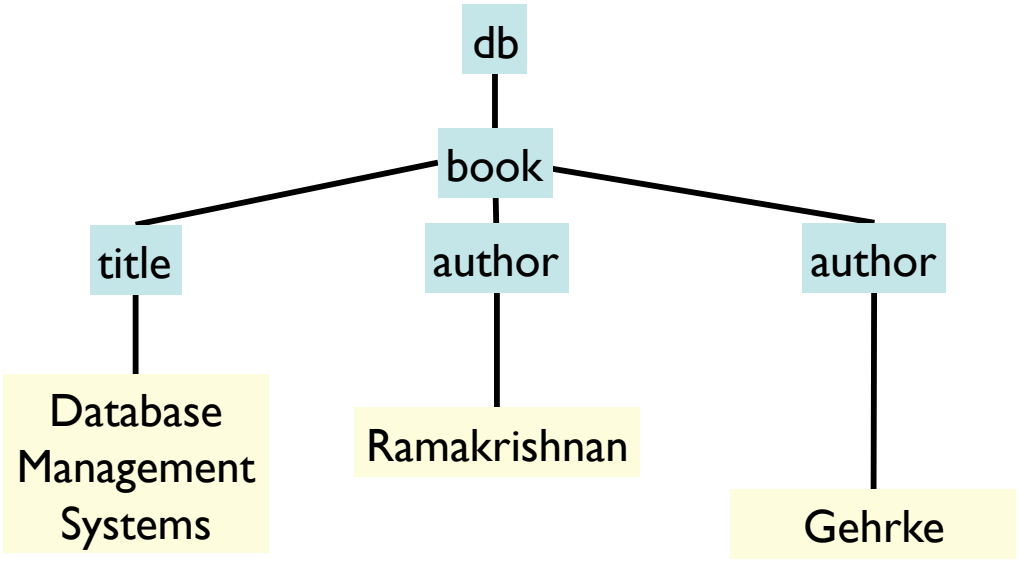
Intervals

Idea: Node-identifier embed navigational-information



Intervals

Think of XML as a linear string

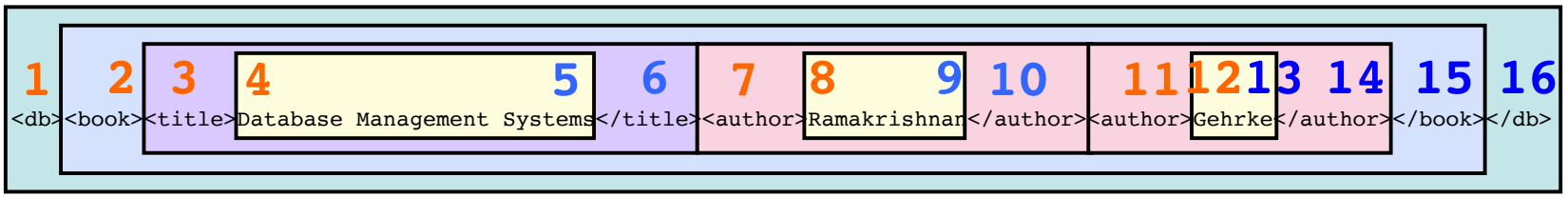



```
<db><book><title>Database Management Systems</title><author>Ramakrishnan</author><author>Gehrke</author></book></db>
```

Intervals

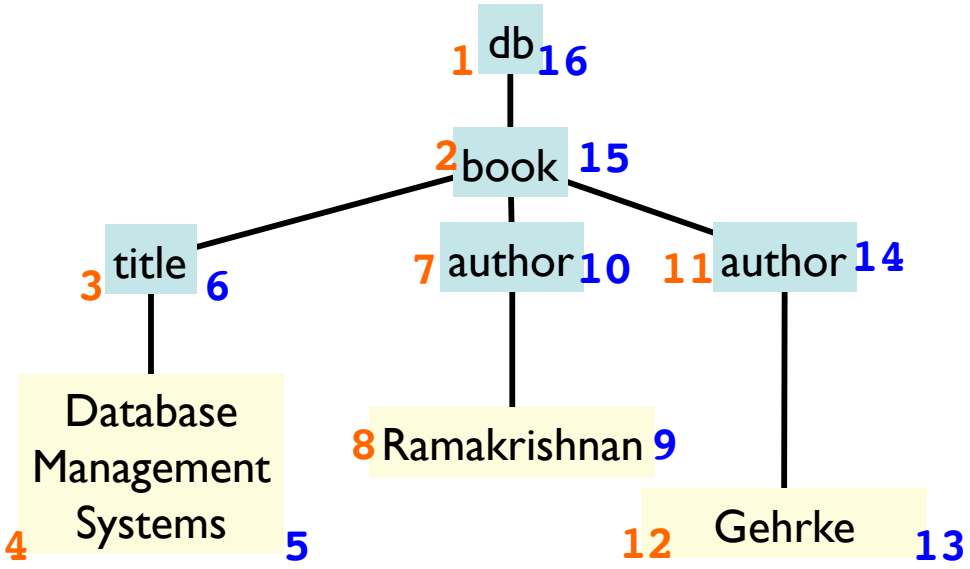
Begin : the first time we see a node (opening tag)

End : the last time we see a node (closing tag)



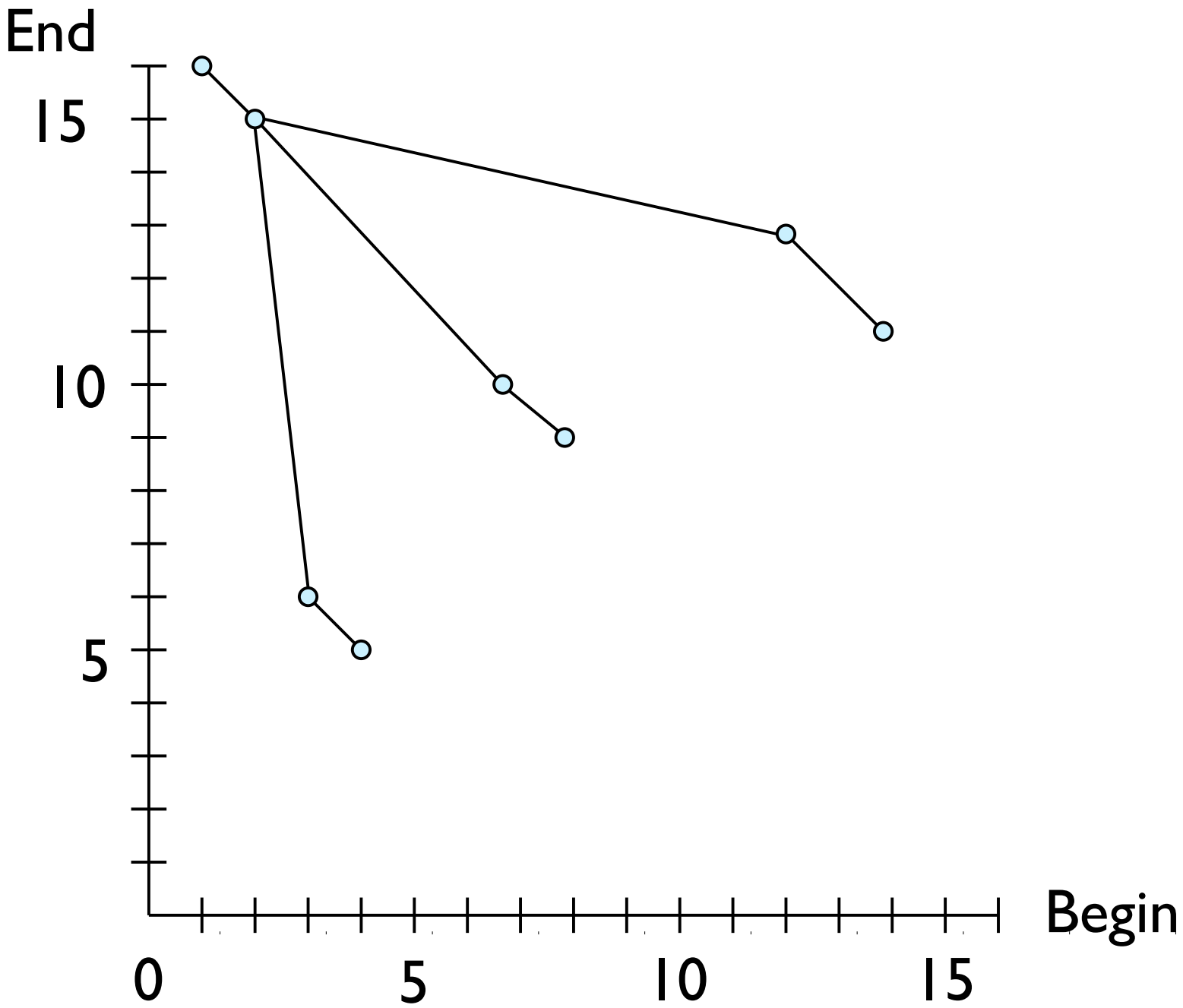
Each node corresponds to an interval on line

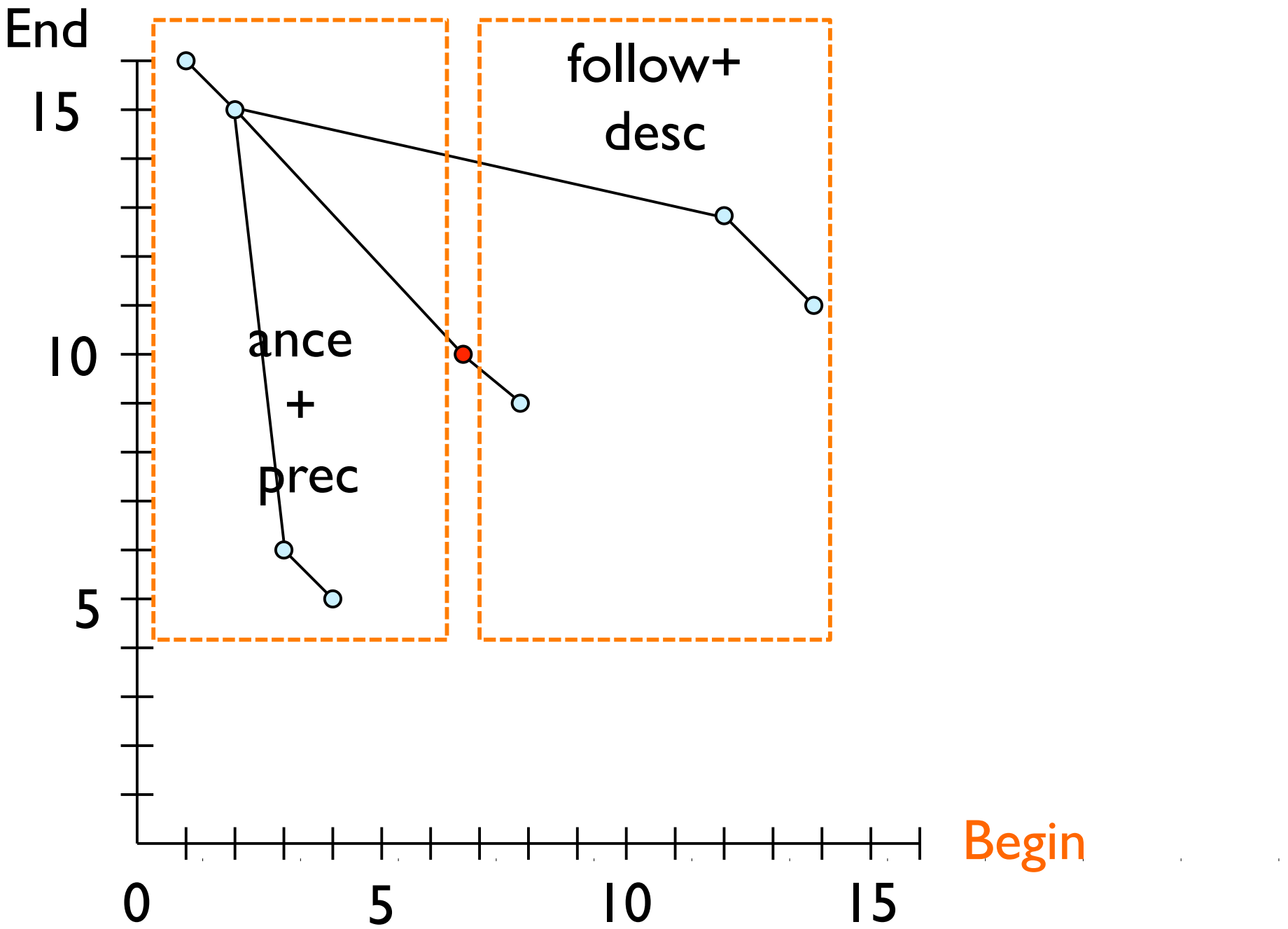
Begin/end numbering

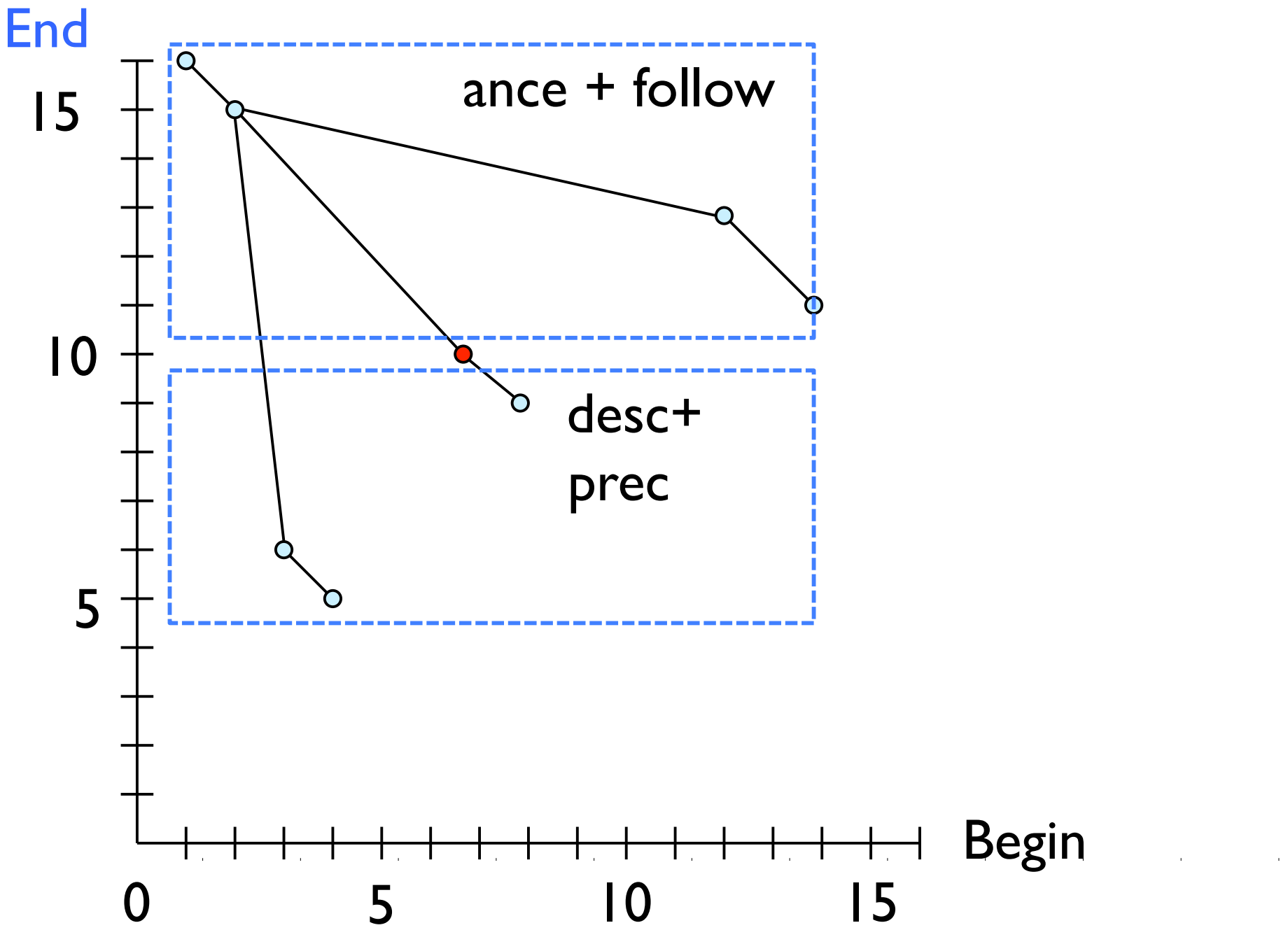


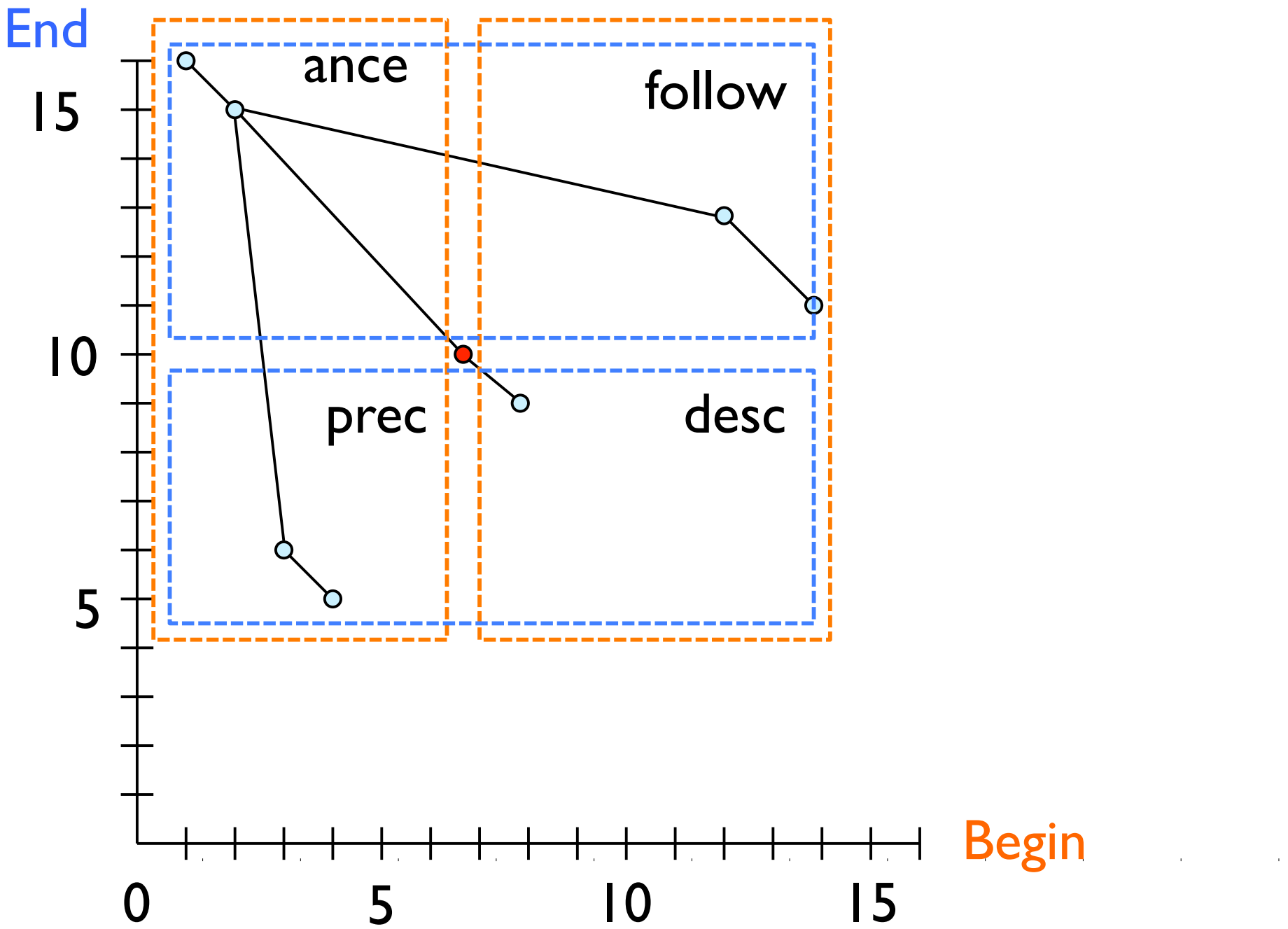
NODE Table

begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT


End
15
10
5
0
5
10
15
Begin


End
15
10
5
0
5
10
15
Begin
follow+
desc
ance
+
prec


End
15
10
5
ance + follow
desc+
prec
0
5
10
15
Begin


End
15
10
5
ance
follow
prec
desc
0
5
10
15
Begin

From Axes to Intervas

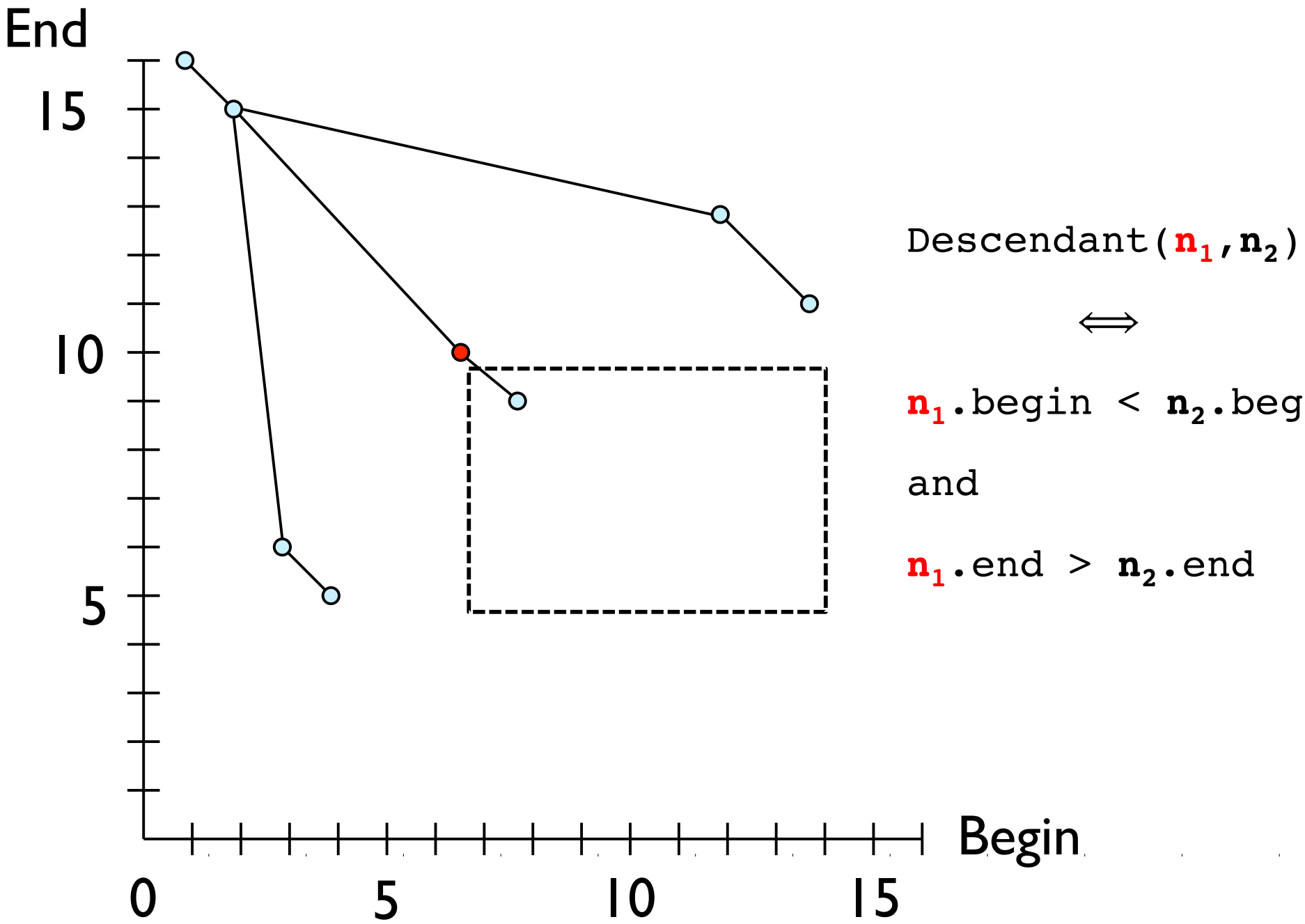
Child(n_1,n_2)

$\Leftrightarrow$

n_1.begin = n_2.par

NODE Table

begin	end	par	tag	type
1	16		db	ELT
2	15	**1**	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT



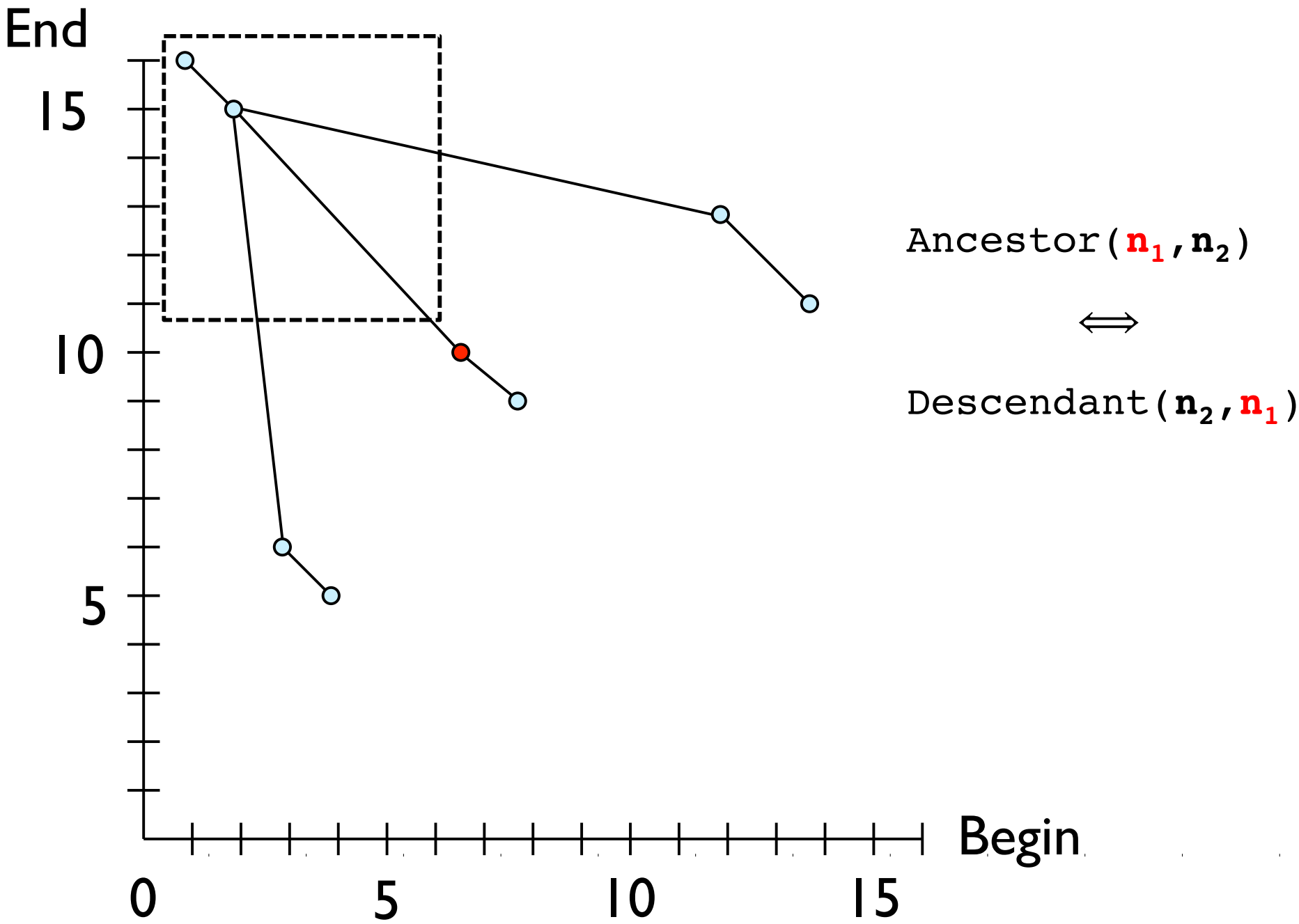
Descendant(n_1, n_2)

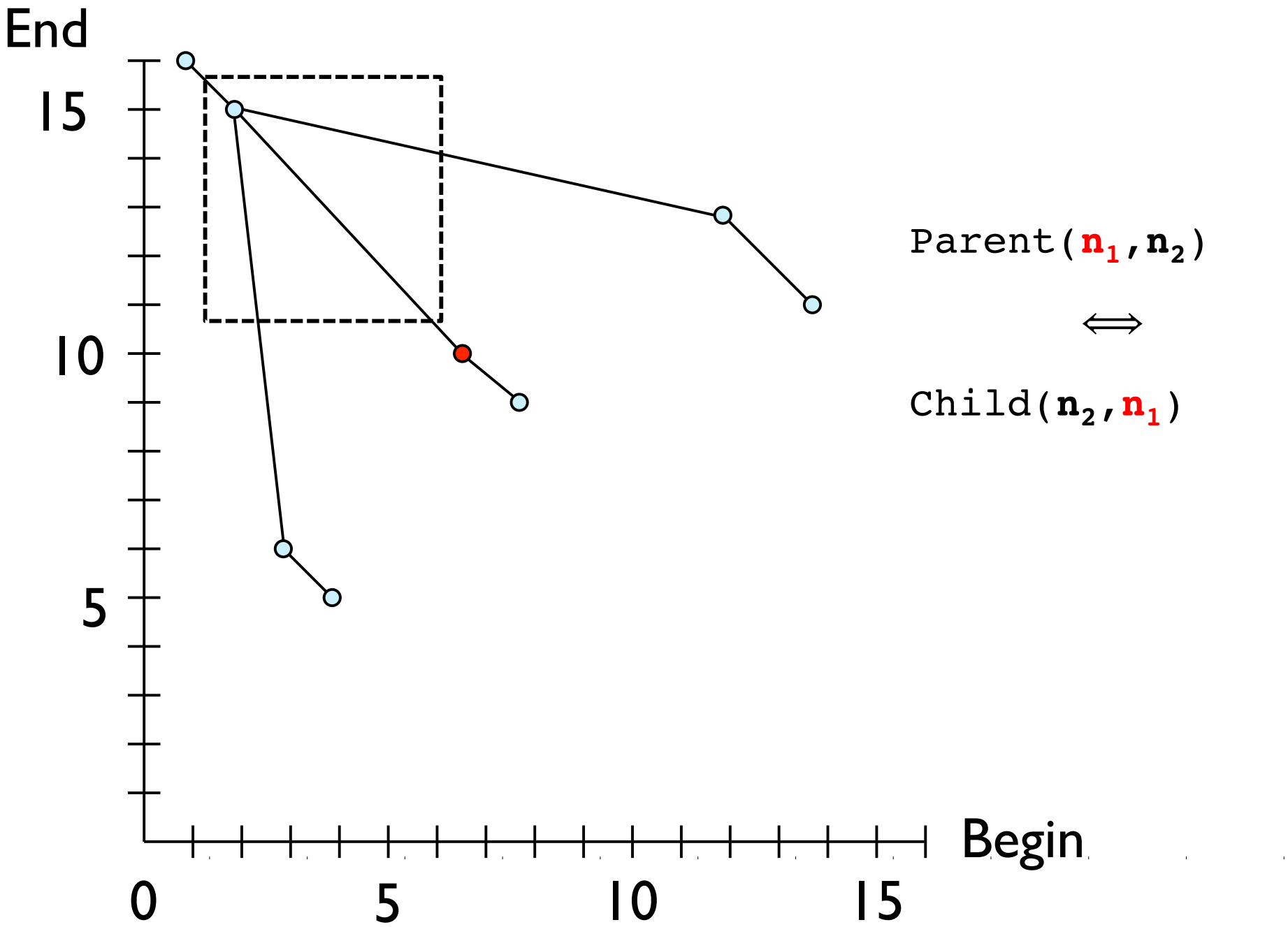
$\Leftrightarrow$

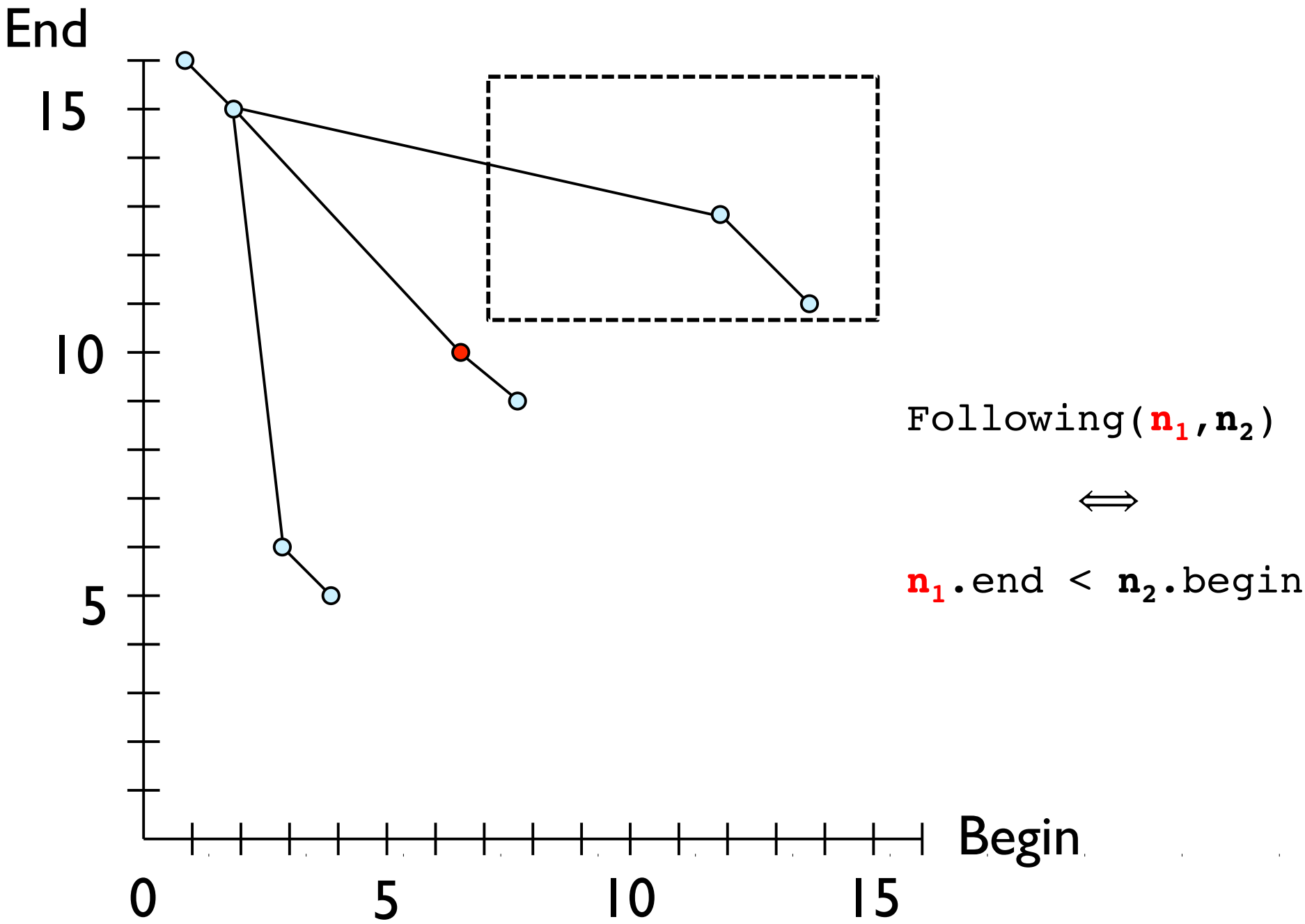
n_1.begin < n_2.begin

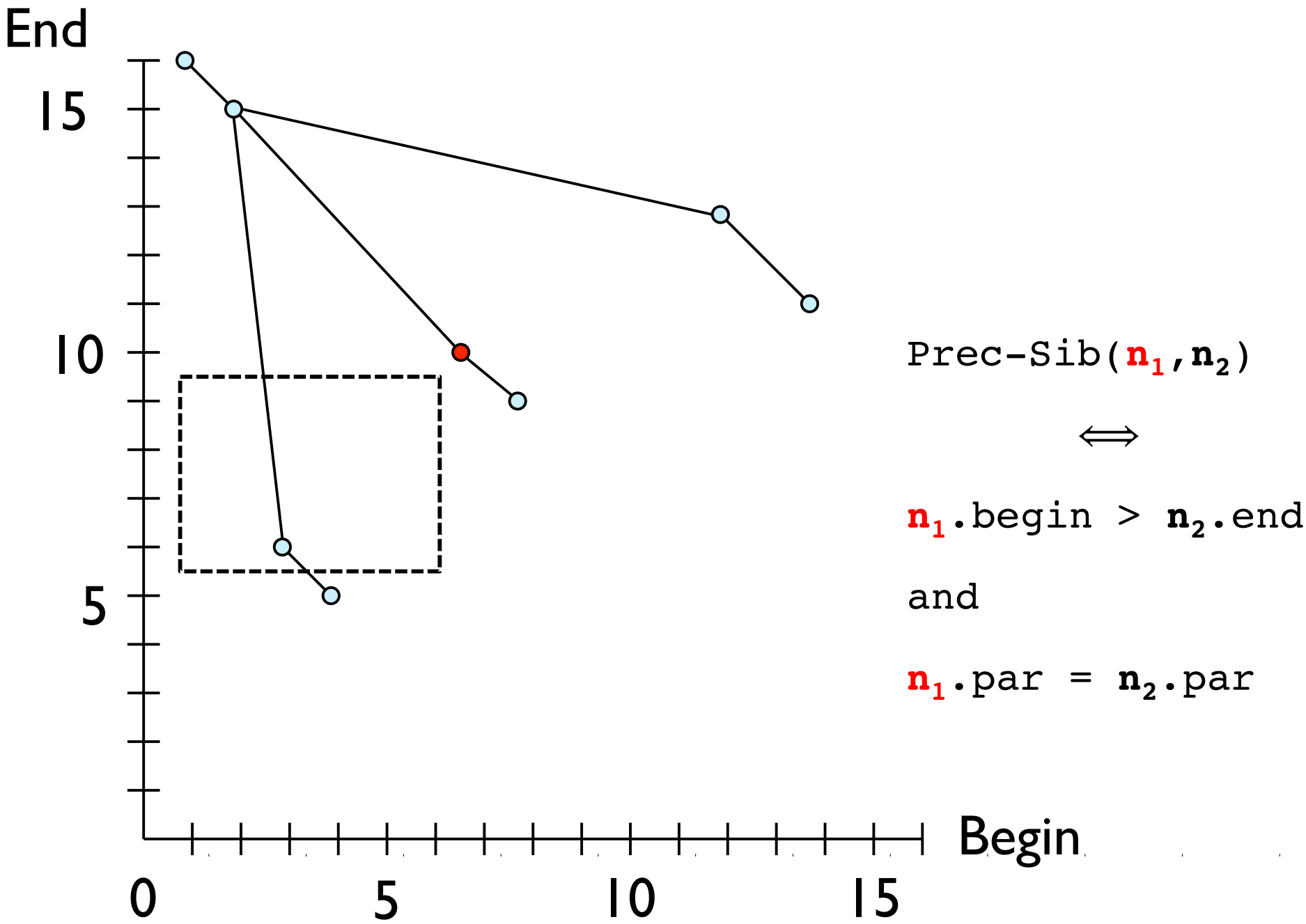
and

n_1.end > n_2.end


End
15
10
5
0
5
10
15
Begin
Ancestor(n_1, n_2)
⟺
Descendant(n_2, n_1)


End
15
10
5
0
5
10
15
Begin
Parent(n_1,n_2)
⟺
Child(n_2,n_1)


End
15
10
5
0
5
10
15
Begin
Following(n_1,n_2)
⟺
n_1.end < n_2.begin



Prec-Sib(n_1,n_2)

$\Leftrightarrow$

n_1.begin > n_2.end

and

n_1.par = n_2.par

Ready to Query (with all axes!)

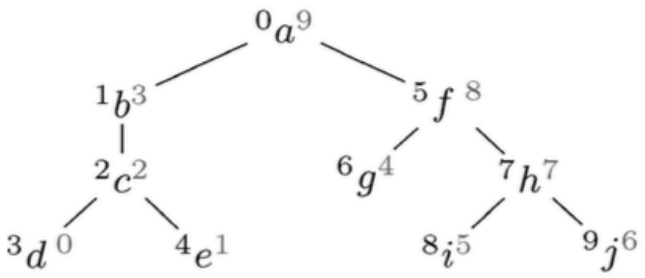
```
Q =  //a//b/ancestor::c//d/following-sibling::e
```

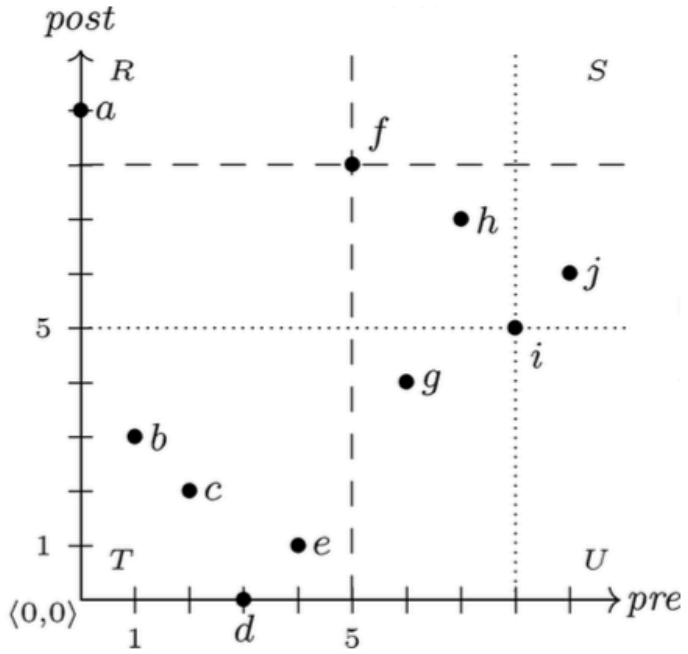
```
SELECT e.nodeID

FROM node a, node b, node c, node d, node e

WHERE
   a.tag = 'a', b.tag = 'b',
   c.tag = 'c', d.tag='d', e.tag='e'

  AND Descendant(a.nodeID,b.nodeID)
  AND Ancestor(b.nodeId,c.nodeId)
  AND Descendant(c.nodeId,d.nodeId)
  AND Following-Sibling(d.nodeId,e.nodeId)
```

to simplify the query, we assume that the nodes have also a unique **`nodeId`** but we can use **begin**

Other Approaches: Pre/Post

(Gurst et al. 2004)



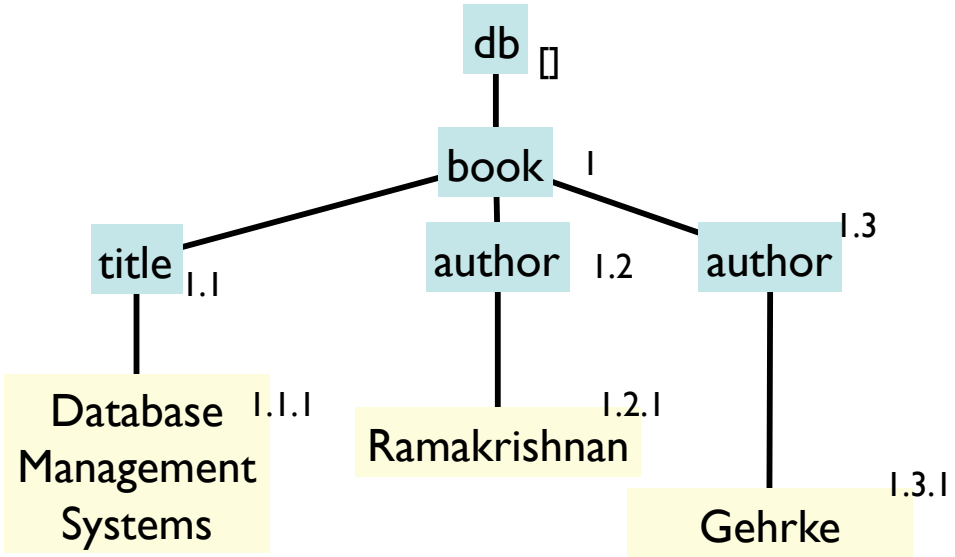


Replaces Begin/End with Pre/Post visit of the tree

Other Approaches: Dewey Decimal Encoding

Each node's ID is a list of integers

- $[i_1,i_2, \ldots ,i_n]$ (often written $i_1.i_2. \ldots .i_n$)
- giving the "path" from root to this node



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Summary

Dewey: string index, requires PREFIX, LEN UDFs

Interval: integer begin/end, pre/post indexes, only requires arithmetic

What about updates?

- Dewey: requires renumbering (exist update-friendly variants)
- Interval encoding: can require re-indexing most of the document

SCHEMA-AWARE XML STORAGE

Derivation of relational schema from DTD

Should be lossless

- the original document can be effectively reconstructed from its relational representation

Should support querying

- XML queries should be able to be rewritten to efficient relational queries

A book DTD

```
<!ELEMENT db  (book*)>

<!ELEMENT book (title,author*,chapter*, ref*)>

<!ELEMENT chapter  (text | section)*>

<!ELEMENT ref  book>

<!ELEMENT title  #PCDATA>

<!ELEMENT author  #PCDATA>

<!ELEMENT section  #PCDATA>

<!ELEMENT text  #PCDATA>
```

Complex
Regular Expressions

Recursion

Recall :regular expressions

`r ::=`	`ϵ`	empty sequence
|	`a`	(tag element name)
|	`(r,s)`	sequential composition
|	`(r|s)`	union
|	`(r*)`	repetition

`r+ = r*,r` `r? = r|ϵ`

First-step : Simplification of RegExp

```
r ::=  ϵ
    |    a
    |  (r,s)
    |  (r|s)
    |  (r1,r2)*
```

Order does not matter

Correlation does not matter

```
(a,b)*  -1->  (a*,b*)  -2->  (a*|b*)
```

A book DTD

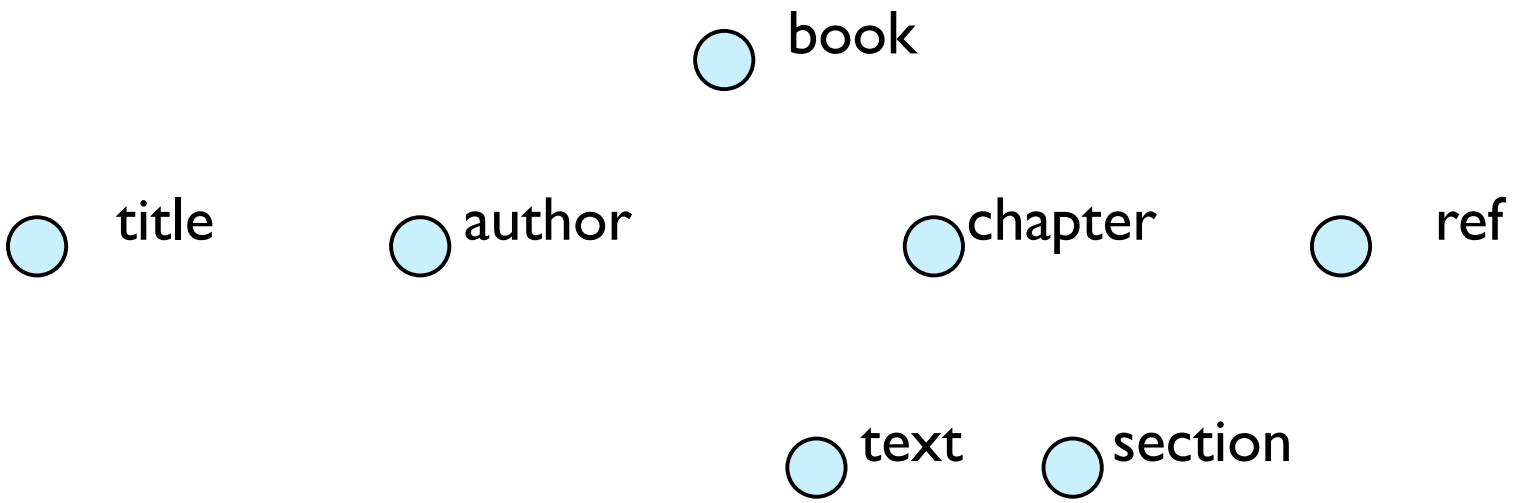
```
<!ELEMENT book (title,authors*,chapter*,ref*)>

<!ELEMENT chapter  (text | section)*>
```

is transformed in

```
<!ELEMENT book (title|authors*|chapter*| ref*)>

<!ELEMENT chapter  ( (text*) | (section*) ) >
```

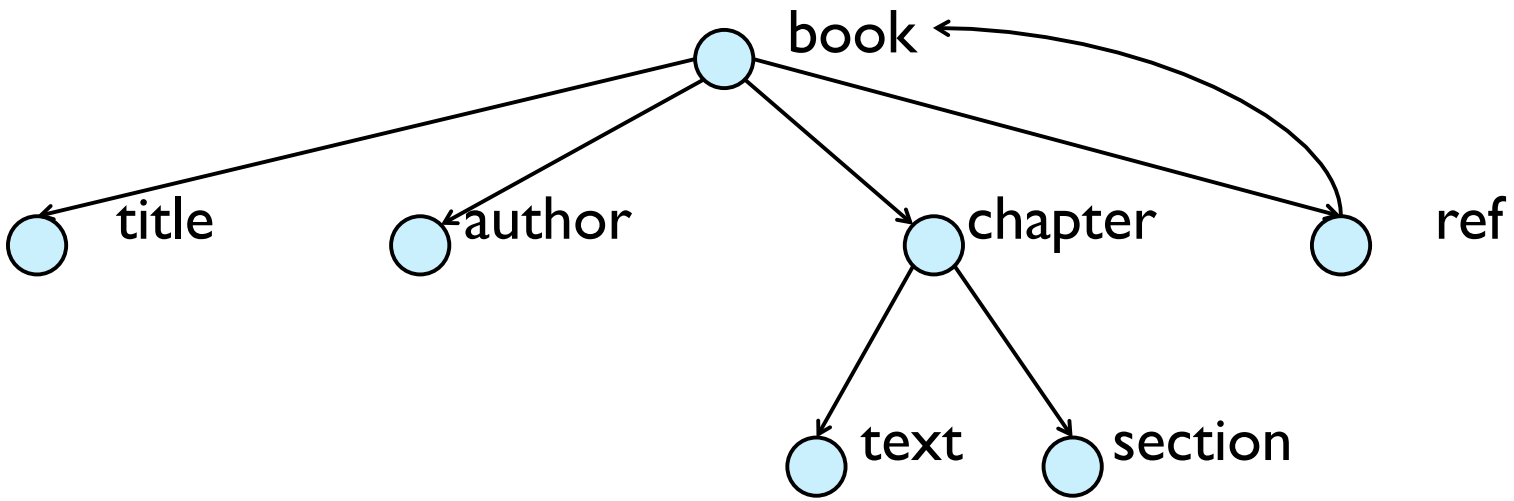
Second step: create a graph representation of the DTD

```
<!ELEMENT book (title|author*|chapter*| ref*)>
<!ELEMENT chapter  (text*) | (section*) >
<!ELEMENT ref  book>
```



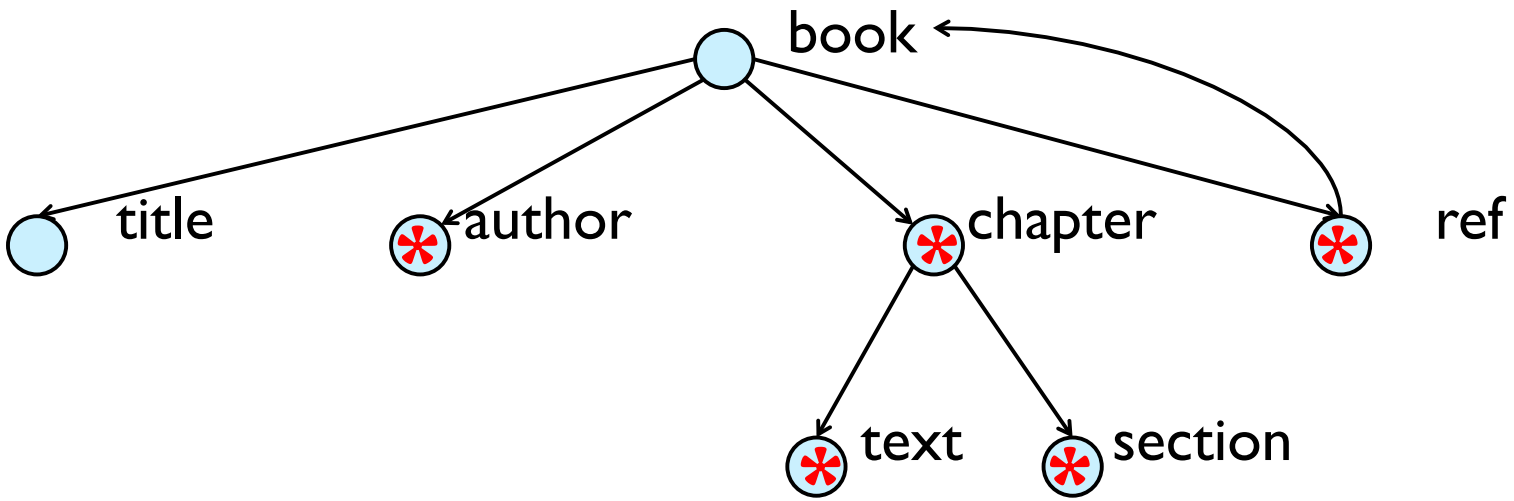
Second step: create a graph representation of the DTD

```
<!ELEMENT book (title|author*|chapter*| ref*)>

<!ELEMENT chapter  (text*) | (section*) >

<!ELEMENT ref  book>
```



Second step: create a graph representation of the DTD

```
<!ELEMENT book (title|author*|chapter*| ref*)>

<!ELEMENT chapter  (text*) | (section*) >

<!ELEMENT ref  book>
```



Graph representation of DTD

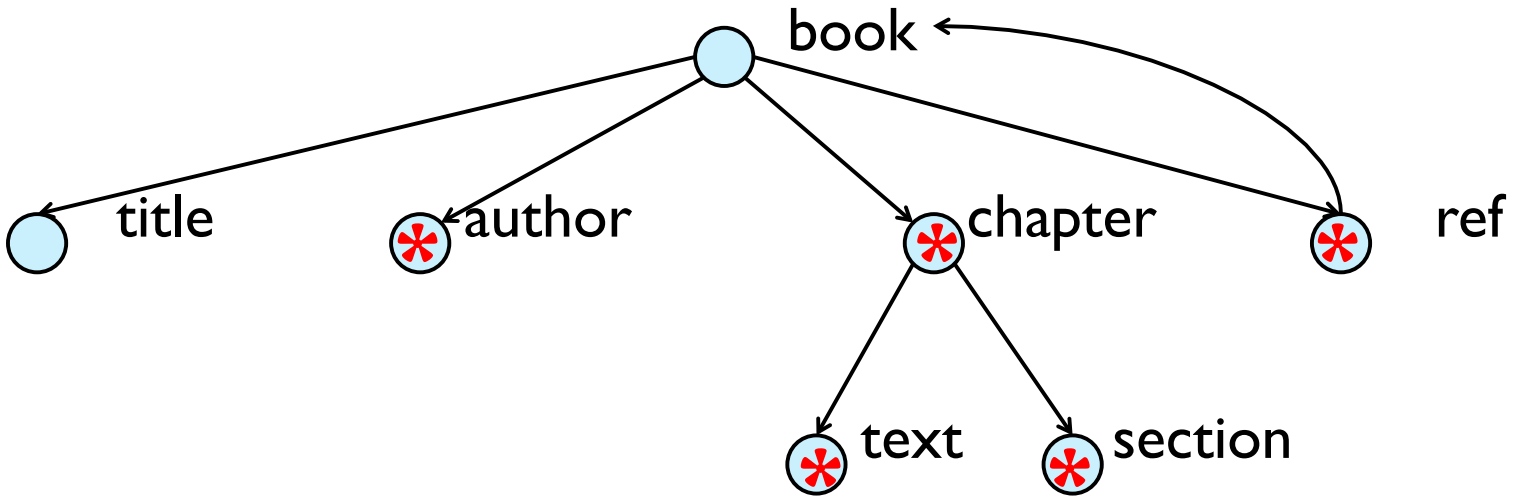
1. Each element type / attribute is represented by a unique node
2. Edges represent the subelement (and attribute) relations
3. Symbol *: denotes 0 or more occurrences of subelements
4. Cycles indicate recursion
    1. e.g., `book -> ref -> book -> ref`

Third step: Create Relations + Inline

Traverse the DTD graph depth-first and create relations for

(1) the root (2) each * node (3) each recursive node
(4) each node with at least 2 parents

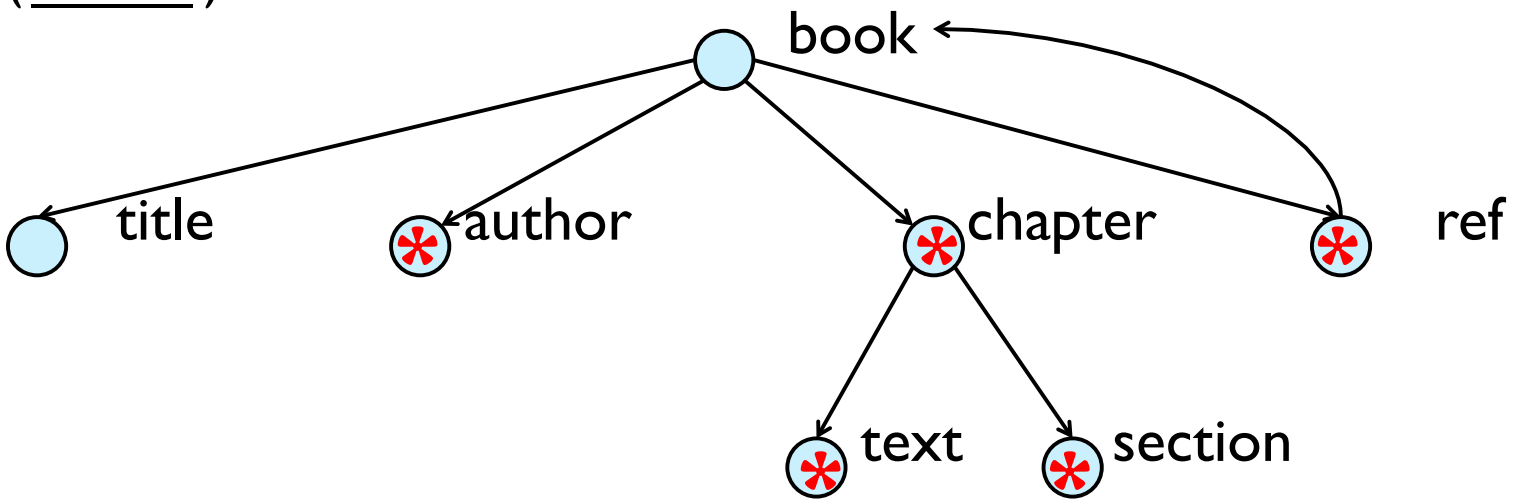
Nodes (w/out * and w/ only1parent) are inlined as fields: no relation created



Third step: Create Relations + Inline

```
book(bookID, title: string)
author(authorID, author: string)
chapter(chapterID)
text(textID, text: string)
section(sectionID, section: string)
ref(refID)
```

we forgot something..



Third step: Create Relations + Inline

book(bookID, title: string)

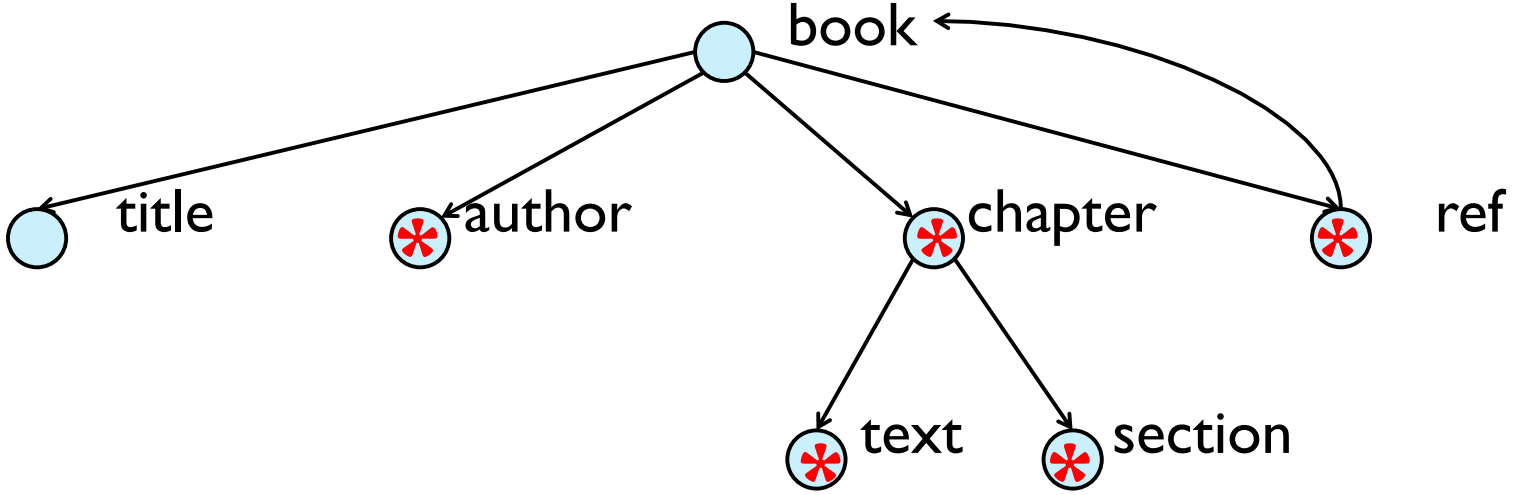
author(authorID, **bookID**, author: string)

chapter(chapterID, **bookID**)

text(textID, **chapterID**, text: string)

section(sectionID, **chapterID**, section: string)

ref(refID, **bookID**)



Still missing detail : parent-ambiguity

book(bookID, **flagRoot**, title: string)

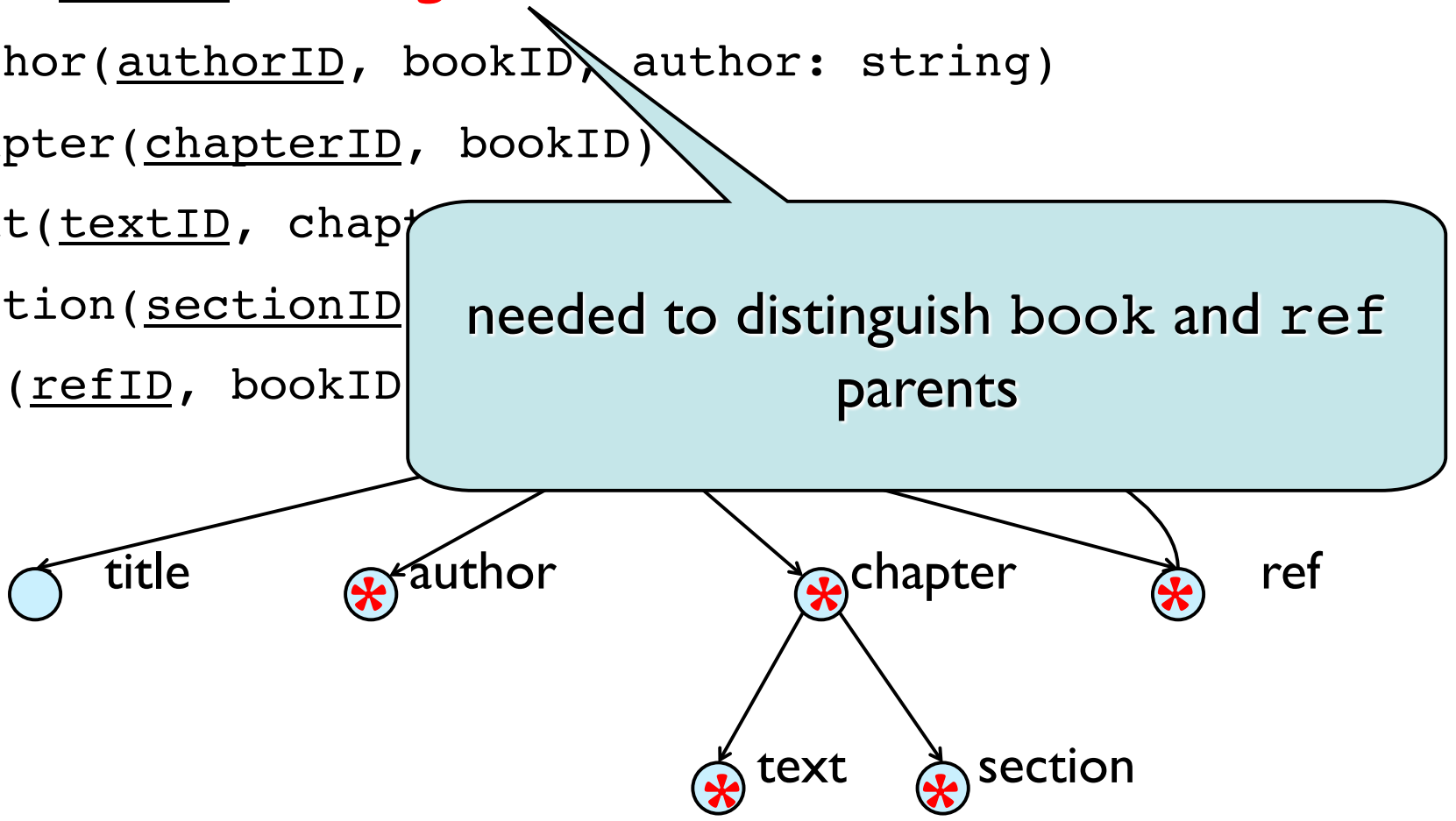
author(authorID, bookID, author: string)

chapter(chapterID, bookID)

text(textID, chap

section(sectionID

ref(refID, bookID



Still missing detail : parent-ambiguity

book(bookID, **flagRoot**, title: string, **parentID**)

author(authorID, bookID, author: string)

chapter(chapterID, bookID)

text(textID, chapterID, text: string)

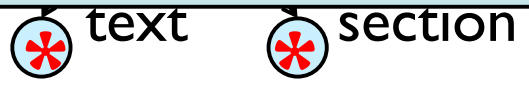
section(sectionID, chapterID, section: string)

ref(refID, bookID)

Foreign keys:

book.parentID ⊆ db.dbID if **flagRoot**= 1

book.parentID ⊆ ref.refID if **flagRoot**= 0



ref

Relational schema

```
book(bookID, flagRoot, title: string, parentID)

author(authorID, bookID, author: string)

chapter(chapterID, bookID)

text(textID, chapterID, text: string)

section(sectionID, chapterID, section: string)

ref(refID, bookID)
```

To preserve the semantics

- ID: each relation has an artificial ID (key)
- parentID: foreign key coding edge relation
- We can also add column naming path in the DTD graph

Note: `title` is inlined

Summary of schema-ware XML

Use DTD/XML Schema to decompose document

Reorganization of regular expressions

- (text , section)* → text* | section*
- document order and type-correlations are lost

Querying: Supports a large class of common XML queries

- Fast lookup & reconstruction of inlined elements
- Systematic translation unclear

COMMERCIAL SOLUTIONS

Well, XML is just text, right?

Most databases allow CLOB (Character Large Object) columns - unbounded length string.

So you just store the XML text in one of these

Surprisingly popular

- and can make sense for storing "document-like" parts of XML data (eg HTML snippets)
- But not a good idea if you want to query the XML

SQL / XML

Instead of blindly using CLOBs.. extend SQL with XML features

- "XML" column type
- XPath or XQuery queries (or updates) on XML columns

Also surprisingly popular (DB2, IBM, Oracle)

- Pro: At least DB knows it's XML
- Pro: Part of SQL 2003 (SQL/XML extensions)
- Con: Frankenstein's query language

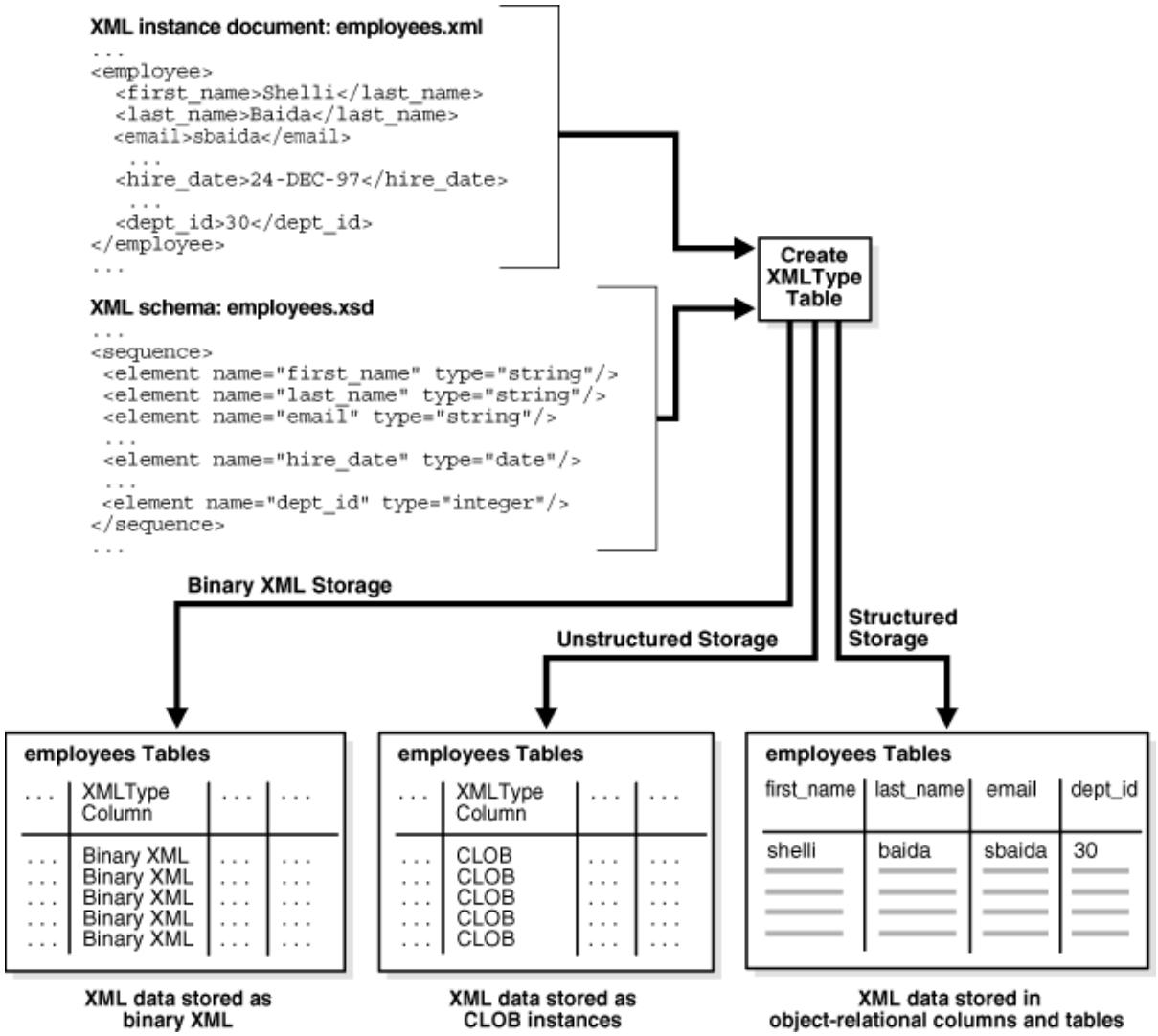
SQL/XML example

```
CREATE TABLE Customers(

  CustomerID int PRIMARY KEY,

  CustomerName nvarchar(100),

  PurchaseOrders XML, ...
)
```

SQL/XML example

```
SELECT CustomerName,

       query(PurchaseOrders,

  'for $p in /po:purchase-order

   where $p/@date < xs:date("2002-10-31")

   return <purchaseorder date="{$p/@date}">

             {$p/*}

          </purchaseorder>')

FROM  Customers

WHERE CustomerID = 42
```

XML Column Type : 3 Possible Storages



Comparison of storage models

	CLOB	OR	Binary XML
Query	poor	excellent	good/excellent
DML	poor	good/excellent	excellent
document retrieval	excellent	good/excellent	excellent
schema flexibility	good	poor	excellent
document fidelity	excellent	poor	good/excellent
mid-tier integration	poor	poor	excellent

Publishing relational data as XML

Federico Ulliana
UM, LIRMM, INRIA-GraphIK

Slides collected from James Cheney

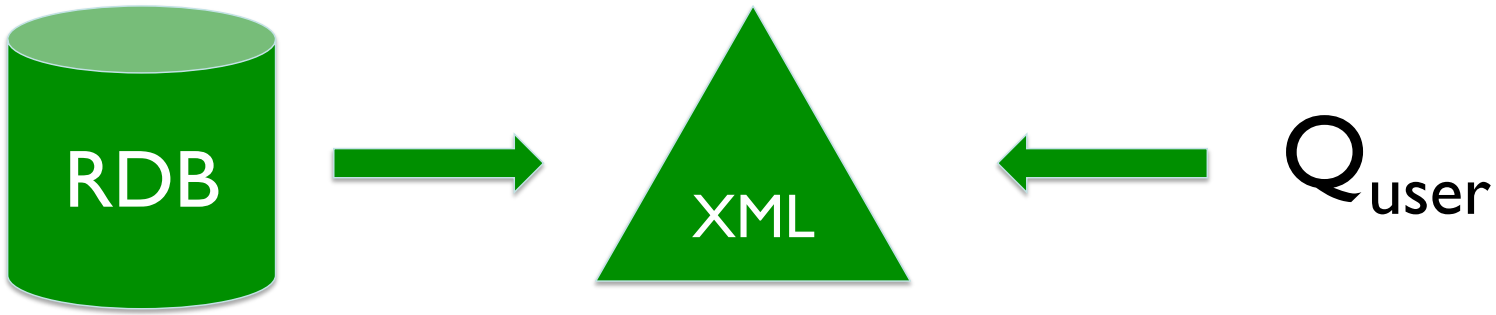
Winter ’99 : XML standardization

Jan 2000 : people (survived from millennium bug)
(and still) wondering …

Now, how can I publish online my relational data?

Data publishing

Make available in the Web a dataset which is not XML



Multiple possible exports

Actors

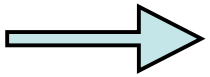
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>

<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```

Grouped by Movie

Which one to chose ?

Actors

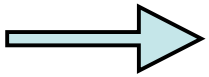
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>

<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie id="11">
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Grouped by Actor

Commercial systems

Systems:

- Oracle 10g XML SQL facilities: SQL/XML
- IBM DB2 XML Extender: SQL/XML, DAD
- Microsoft SQL Server 2005: FOR-XML, XSD

Canonical XML representation of relations

- Incapable of expressing practical XML publishing
  - default fixed XML document template

Canonical XML publishing

Actors

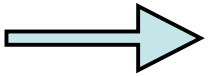
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

```
<Actors>
    <Actor aid="1">
        <LName>Maguire</LName>
        <FName>Tobey</FName>
    </Actor>
    <Actor aid="2">
        <LName>Dunst</LName>
        <FName>Kirsten</FName>
    </Actor>
</Actors>
```

Canonical XML publishing

Actors

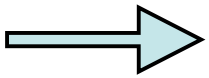
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten



```
<Actor aid="1">
 <LName>Maguire</LName>
 <FName>Tobey</FName>
</Actor>
<Actor aid="2">
 <LName>Dunst</LName>
 <FName>Kirsten</FName>
</Actor>
```

Movies

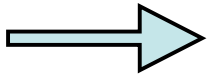
mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005



```
<Movie mid="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
</Movie>
<Movie mid="32">
 <Title>Elizabethtown</Title>
 <Year>2005</Year>
</Movie>
```

Appears

mid	aid
11	1
11	2
32	2



```
<Appears mid="11" aid="1"/>
<Appears mid="11" aid="2"/>
<Appears mid="32" aid="2"/>
```

Called **canonical** because the same rules are applied to convert any relational table to an XML view

How to go beyond canonical publishing ?

Need language to specify relational-to-XML conversion

And an efficient implementation

Two main proposals

- XPERANTO (focus today)
- SilkRoute

XPERANTO

XPERANTO

[Shanmagusundaram et al., 2001]

Commercial system: IBM DB2 XML extender, SQL/XML

SQL extension

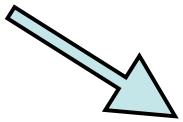
```
select   XMLAGG
from     R1 . . .  Rn
  where  conditions
```

GOAL Input : relational tables Output : XML trees (forest)

From relations to XML Views

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

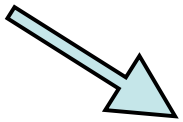


```
<Movie>
    <Title>Spider-Man</Title>
    <Year>2002</Year>
 </Movie>

 <Movie>
   <Title>Elizabethtown</Title>
   <Year>1999</Year>
 </Movie>
```

From relations to XML Views

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005



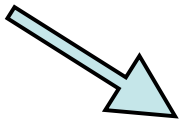
```
SELECT title , year

FROM Movies
```

```
<Movie>
    <Title>Spider-Man</Title>
    <Year>2002</Year>
 </Movie>

 <Movie>
   <Title>Elizabethtown</Title>
   <Year>1999</Year>
 </Movie>
```

From relations to XML Views

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005



```
SELECT title , year

FROM Movies
```

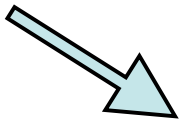
```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 title : integer ) as  {
     <movie>
          <title> $title </title>
          <year>  $title </year>
     </movie>  }
```

```
<Movie>
    <Title>Spider-Man</Title>
    <Year>2002</Year>
</Movie>

<Movie>
  <Title>Elizabethtown</Title>
  <Year>1999</Year>
</Movie>
```

From relations to XML Views

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005



```
SELECT XMLAGG(
    MOVIE(
    SELECT title , year
    FROM Movies    ))
```

```
SELECT title , year
FROM Movies
```

```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 title : integer ) as  {
     <movie>
          <title> $title </title>
          <year>  $title </year>
     </movie>  }
```

```
<Movie>
    <Title>Spider-Man</Title>
    <Year>2002</Year>
 </Movie>

 <Movie>
   <Title>Elizabethtown</Title>
   <Year>1999</Year>
 </Movie>
```

From relations to XML Views

Actors

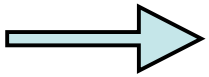
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor>
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie>
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>

<Actor>
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie>
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie>
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Step 1 : Define XML records

```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 year : integer ) as
  {  <movie>
        <title> $title </title>
        <year> $year </year>
     </movie>  }

DEFINE XML CONSTRUCT ACTOR
(fname : varchar(100),
 lname : varchar(100),
 movie : xml ) as
  {  <actor>
        <fname> $fname </fname>
        <lname> $lname </lname>
            $movie
     </actor>  }
```

Target XML Data

```
<Actor>
      <LName>Maguire</LName>
      <FName>Tobey</FName>
      <Movie>
            <Title>Spider-Man</Title>
            <Year>2002</Year>
       </Movie>
</Actor>
...
```

From relations to XML Views

Actors

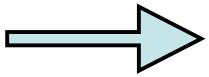
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor>
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie>
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>

<Actor>
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie>
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie>
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Idea: Reuse SQL Correlated Queries

- all movies with at least one actor

```
SELECT mid

FROM MOVIES M

WHERE M.mID IN (SELECT mid

                FROM APPEARS A)
```

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

Idea: Reuse SQL Correlated Queries

Inner-query that uses variables from outer-query

- all movies with at least one actor

```
SELECT mid

FROM MOVIES M

WHERE EXISTS (     SELECT aid

                   FROM APPEARS A

                   WHERE A.mid = M.mId    )
```

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

Idea: Reuse SQL Correlated Queries

Inner-query that uses variables from outer-query

- all movies with at least one actor

```
SELECT mid

FROM MOVIES M

WHERE EXISTS (     SELECT aid

                   FROM APPEARS A

                   WHERE A.mid = M.mId    )
```

Correlative Sub-query

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

XPERANTO (SQL/XML)

Actors

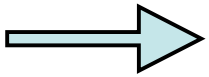
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie id="11">
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie id="32">
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Export grouping by actor

Step 1 : Define XML records

```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 year : integer ) as
  {  <movie>
        <title> $title </title>
        <year> $year </year>
     </movie>  }

DEFINE XML CONSTRUCT ACTOR
(fname : varchar(100),
 lname : varchar(100),
 movie : xml ) as
  {  <actor>
        <fname> $fname </fname>
        <lname> $lname </lname>
            $movie
     </actor>  }
```

Target XML Data

```
<Actor>
      <LName>Maguire</LName>
      <FName>Tobey</FName>
      <Movie>
            <Title>Spider-Man</Title>
            <Year>2002</Year>
       </Movie>
</Actor>
...
```

Step 1 : Define XML records

```
DEFINE XML CONSTRUCT MOVIE
(title : varchar(100),
 year : integer ) as
  {  <movie>
          <title> $title </title>
          <year> $year </year>
     </movie>  }

DEFINE XML CONSTRUCT ACTOR
(fname : varchar(100),
 lname : varchar(100),
 movie : xml ) as
  {  <actor>
          <fname> $fname </fname>
          <lname> $lname </lname>
               $movie
     </actor>  }
```

```
MOVIE ('spiderman',2002) =

<movie>
      <title>spiderman</title>
      <year>2002</year>
</movie>


ACTOR('Maguyre','Tobey',<m/>)=

  <actor>
        <fname>Maguyre</fname>
         <lname>Tobey</lname>
              <m/>
  </actor>
```

Step 2 : Define Tree Aggregation

XMLAGG(<my/>,<XML/>,<trees/> **)** =

<my/><XML/><trees/>

Ready to define a SQL/XML Query

```
SELECT XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

                    MOVIE( SELECT title , year

                           FROM Appears, Movies

                           WHERE Appears.aid = O.aid

                           AND   Appears.mid = Movies.mid )) )

           FROM Actor  O

          ORDER BY lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELECT XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

                    MOVIE( SELECT title , year

                           FROM Appears, Movies

                           WHERE Appears.aid = O.aid

                           AND   Appears.mid = Movies.mid )) )

           FROM Actor  O

          ORDER BY lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELECT XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

                    MOVIE( SELECT title , year

                           FROM Appears, Movies

                           WHERE Appears.aid = O.aid

                           AND   Appears.mid = Movies.mid )) )

           FROM Actor  O

          ORDER BY lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELECT XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

                    MOVIE( SELECT title , year

                           FROM Appears, Movies

                           WHERE Appears.aid = O.aid

                           AND   Appears.mid = Movies.mid )) )

           FROM Actor  O

          ORDER BY lname, fname ) )
```

Ready to define a SQL/XML Query

```
SELECT XMLAGG(

    ACTOR(SELECT lname, fname,

          ( SELECT XMLAGG(

                    MOVIE( SELECT title , year

                           FROM Appears, Movies

                           WHERE Appears.aid = O.aid

                           AND   Appears.mid = Movies.mid )) )

           FROM Actor  O

          ORDER BY lname, fname ) )
```

From relations to XML Views

Actors

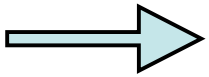
aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2



```
<Actor>
  <LName>Maguire</LName>
  <FName>Tobey</Fname>
  <Movie>
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>

<Actor>
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
  <Movie>
     <Title>Spider-Man</Title>
     <Year>2002</Year>
  </Movie>
  <Movie>
    <Title>Elizabethtown</Title>
    <Year>1999</Year>
  </Movie>
</Actor>
```

Another example

Another example

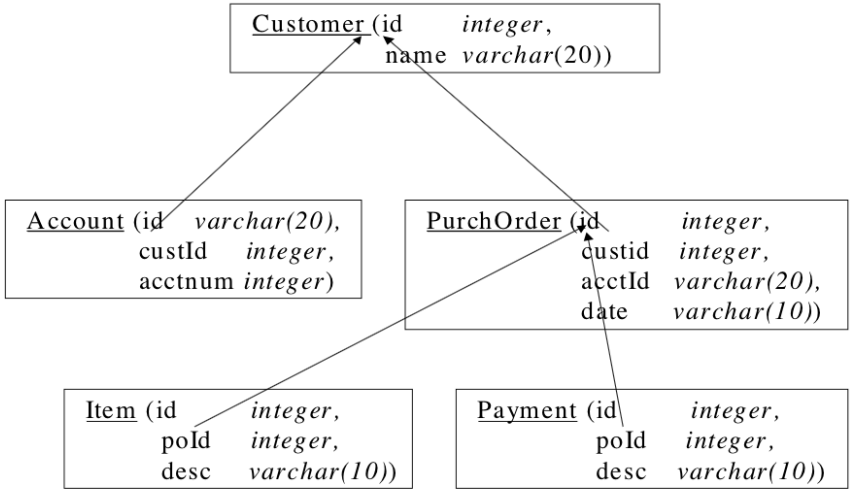


Figure 2: Customer Relational Schema

```
Define XML Constructor CUST (custId: integer,
                             custName: varchar(20),
                             acctList: xml,
                             porderList: xml) AS {
    <customer id=$custId>
        <name> $custName </name>
        <accounts> $acctList </accounts>
        <porders> $porderList </porders>
    </customer>
}
```

Figure 4: Definition of an XML Constructor

```
01. Select  cust.name, CUST(cust.id, cust.name,
02.                         (Select  XMLAGG(ACCT(acct.id, acct.acctnum))
03.                          From   Account acct
04.                          Where acct.custId = cust.id),
05.                         (Select   XMLAGG(PORDER(porder.id, porder.acct, porder.date,
06.                                                 (Select   XMLAGG(ITEM(item.id, item.desc))
07.                                                  From   Item item
08.                                                  Where item.poId = porder.id),
09.                                                (Select  XMLAGG(PAYMENT(pay.id, pay.desc))
10.                                                   From   Payment pay
11.                                                   Where pay.poId = porder.id)))
12.                          From    PurchOrder porder
13.                          Where  porder.custId = cust.id))
14. From  Customer cust
```

XML Export in Commercial RDBMS

DB2 User-defined mapping through DAD (Document Access Definition)

MS SQL Server 2005: Annotated schema (XSD): fixed tree templates; FOR-XML

Oracle 10g : SQL/XML, DBMS_XMLGEN (PL/SQL package)