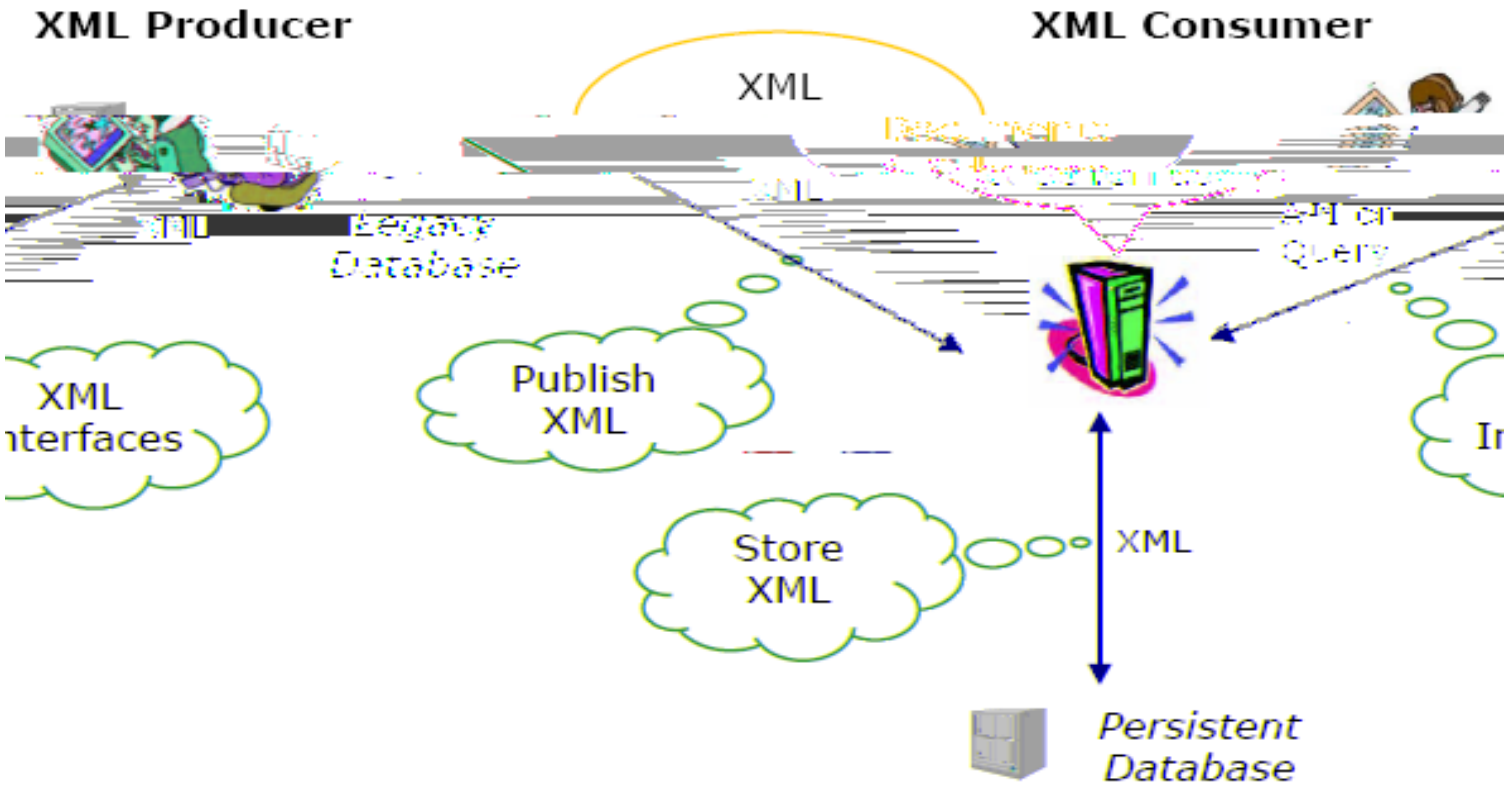
D M

Q ML

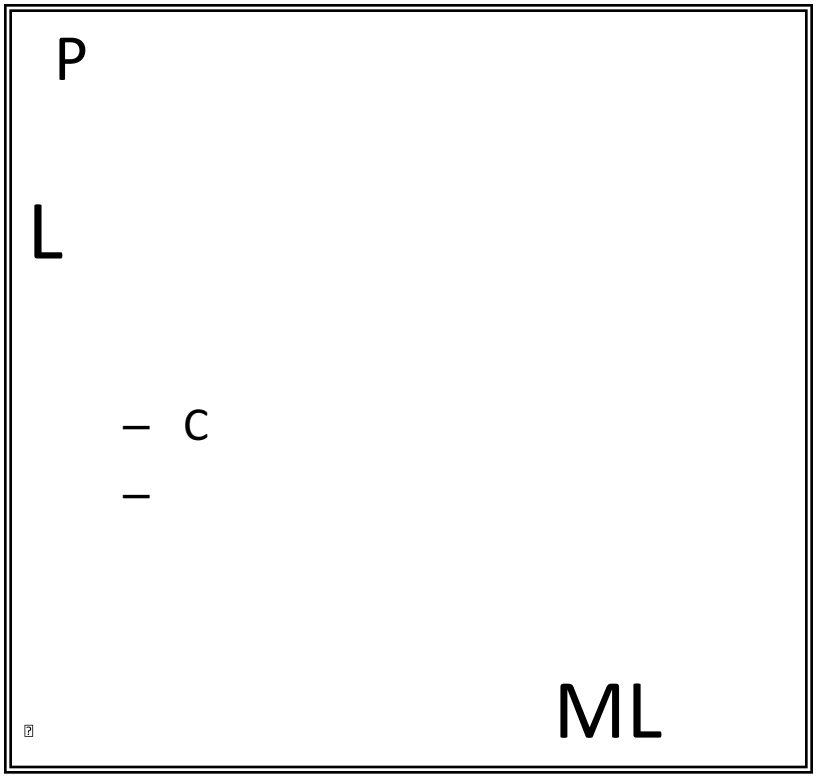
ML D M


XML Producer
XML Consumer
XML
Legacy
Database
XML
nterfaces
Publish
XML
Store
XML
XML
Persistent
Database

C ML

– *Data comes from persistent (disk-based) storage*
  - *First load, then query*
– Q
  - D
  - N LMP F C BCF FHK

ML



"At first sight"

Fast processing of incoming documents

- Web service messages
- Workflow coordination

Many small documents to process

- In-memory, programming language approach feasible

ML

P

H

– N

A DBM

–

– Q

– P

–

– C

–

"At first sight"

Lighter

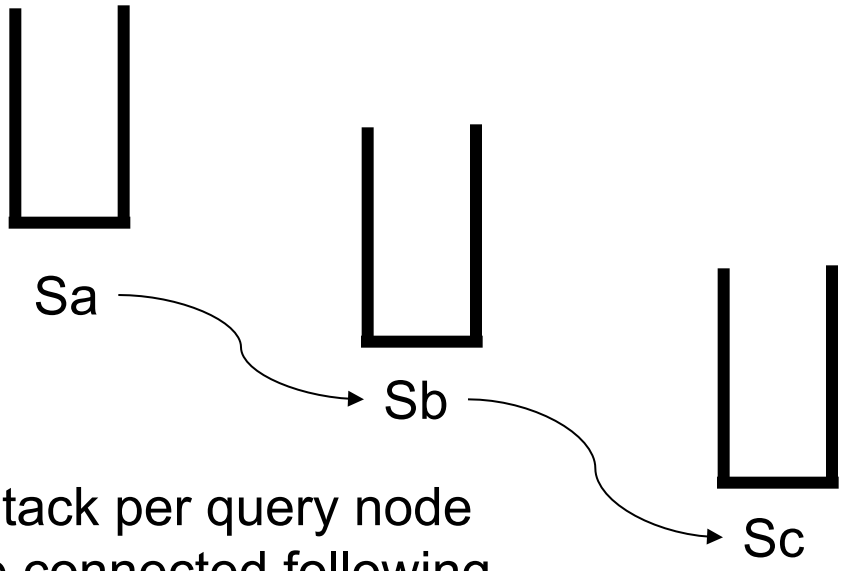
May blend easily into a programming framework

- In real life, there are not just databases

XML document:

```
<r>
  <a1><b1>
            <c1/>
            <c2/>
      </b1>
      <b2>
            <c3/>
      </b2>
  </a1>
  <a2>
      <c4/>
      <b3/>
  </a2>
</r>
```

Query: //a/b/c



Create 1 stack per query node
Stacks are connected following the query structure

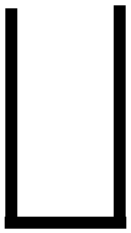
<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

XML document:

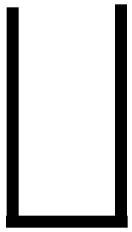
```
<r>
  <a1><b1>
          <c1/>
          <c2/>
      </b1>
      <b2>
          <c3/>
      </b2>
  </a1>
  <a2>
      <c4/>
      <b3/>
  </a2>
</r>
```

SAX traversal

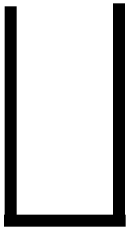
Query: //a/b/c



Sa



Sb

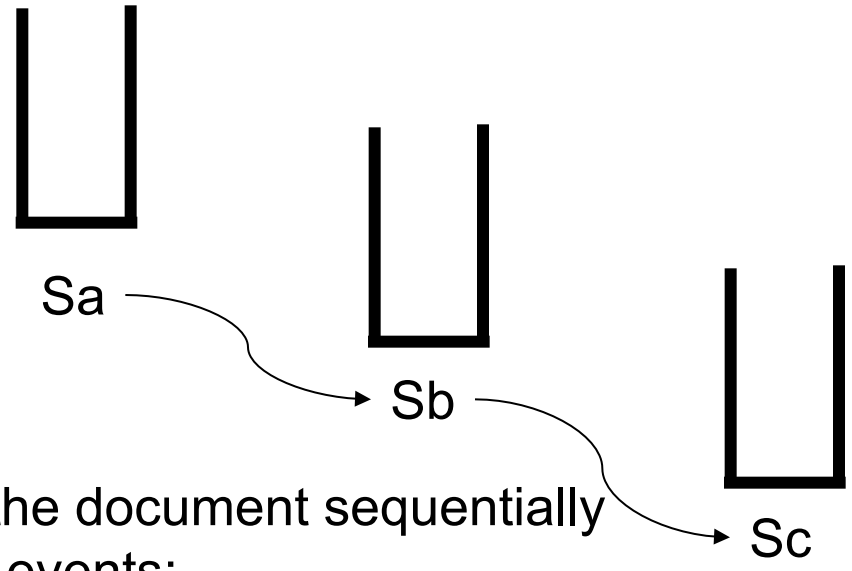


Sc

Traverse the document sequentially and issue events:
Start element x / end element x / text s

<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

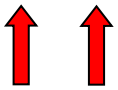
Query: //a/b/c



Traverse the document sequentially and issue events:
Start element x / end element x / text s

<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

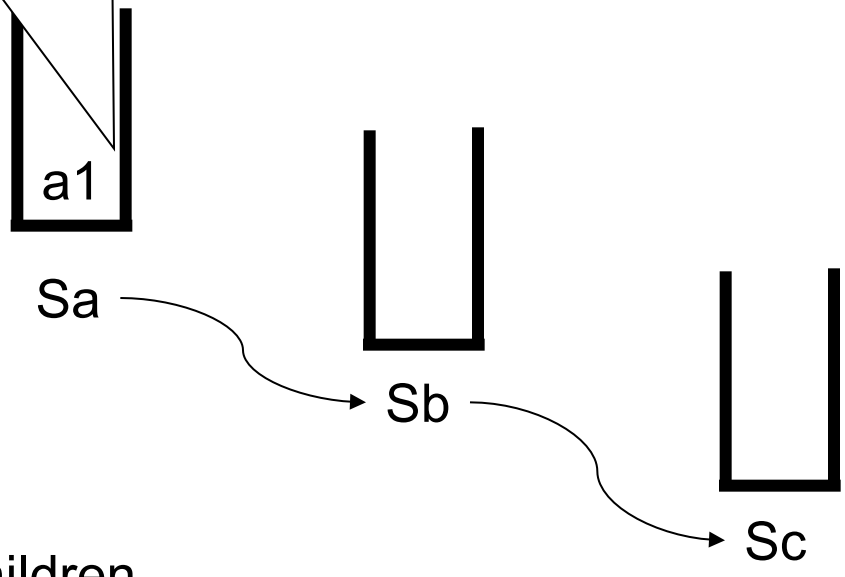
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20



When pushed, matches are open

Query: //a/b/c

On begin element x:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On end element x:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc



<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Query: //a/b/c

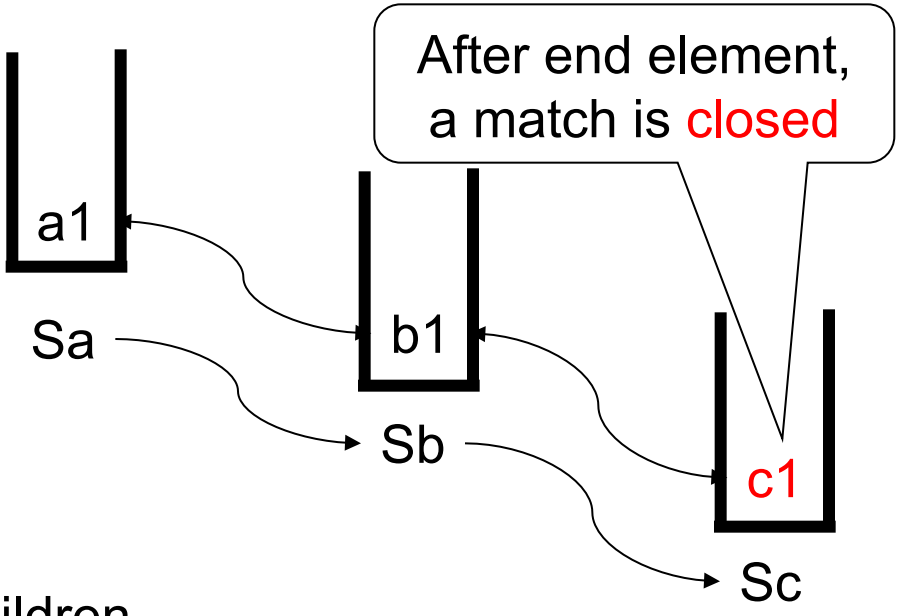
On begin element x:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On end element x:
If there is a stack for x
Then if x is on top of the stack

<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

On <u>begin element x</u>:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On <u>end element x</u>:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc

Query: //a/b/c

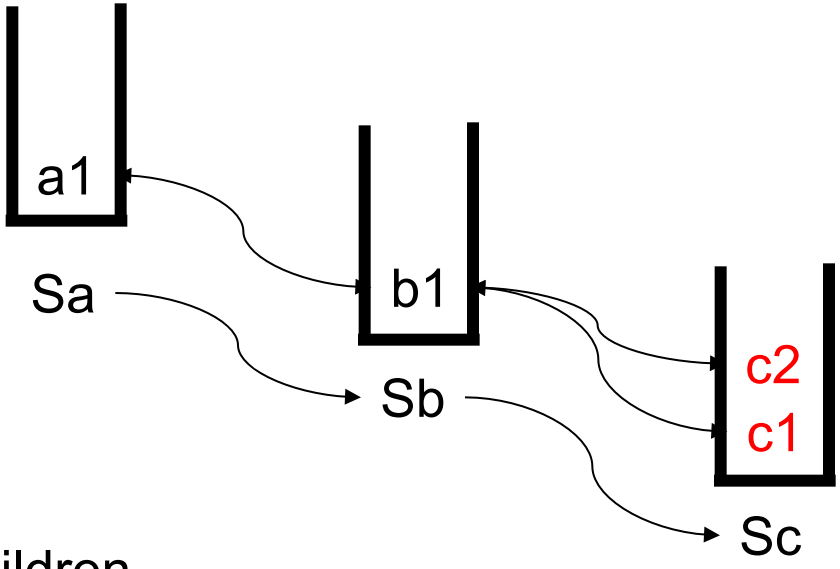


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

On begin element x:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On end element x:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc

Query: //a/b/c

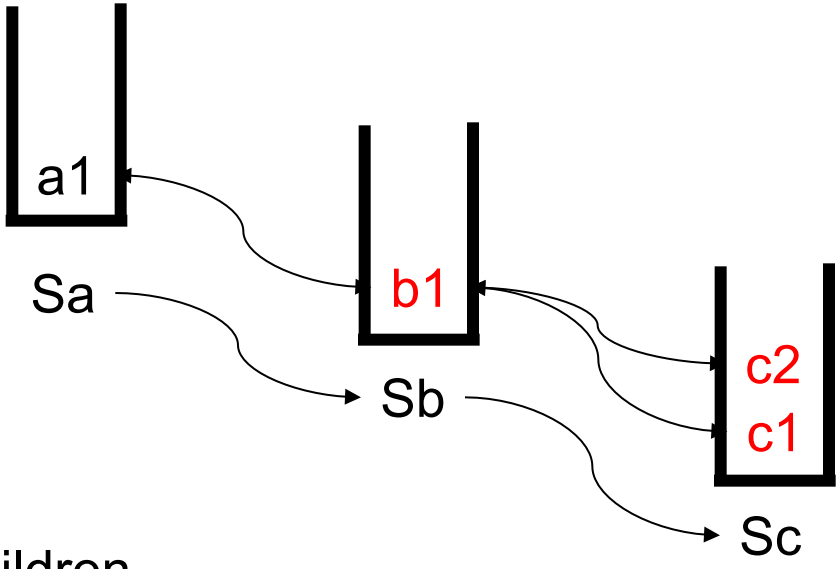


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

On <u>begin element x</u>:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On <u>end element x</u>:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc

Query: //a/b/c

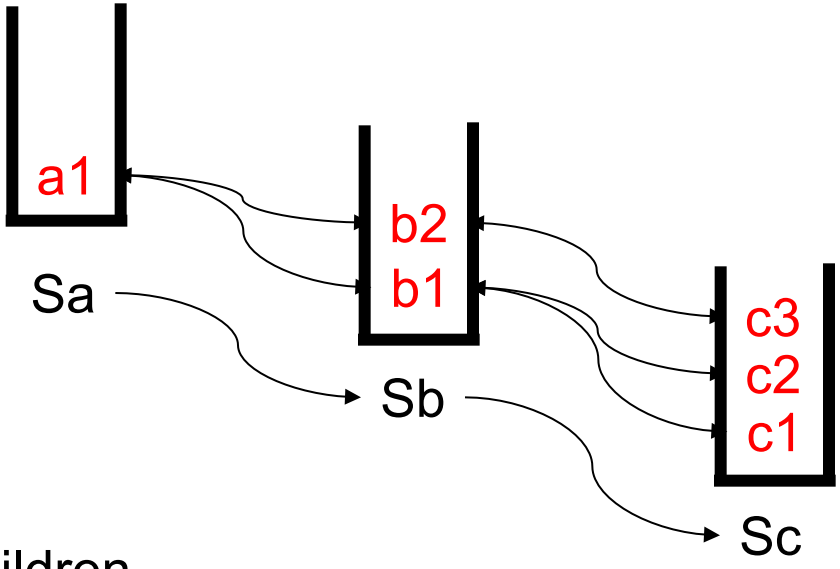


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

On <u>begin element x</u>:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On <u>end element x</u>:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc

Query: //a/b/c

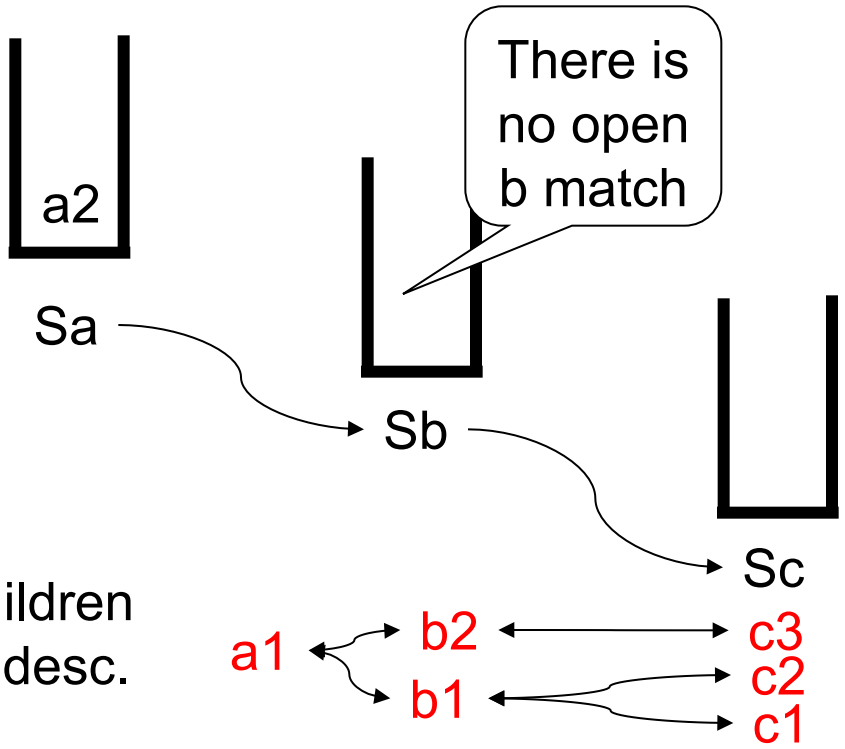


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Query: //a/b/c

On <u>begin element x</u>:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On <u>end element x</u>:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc.

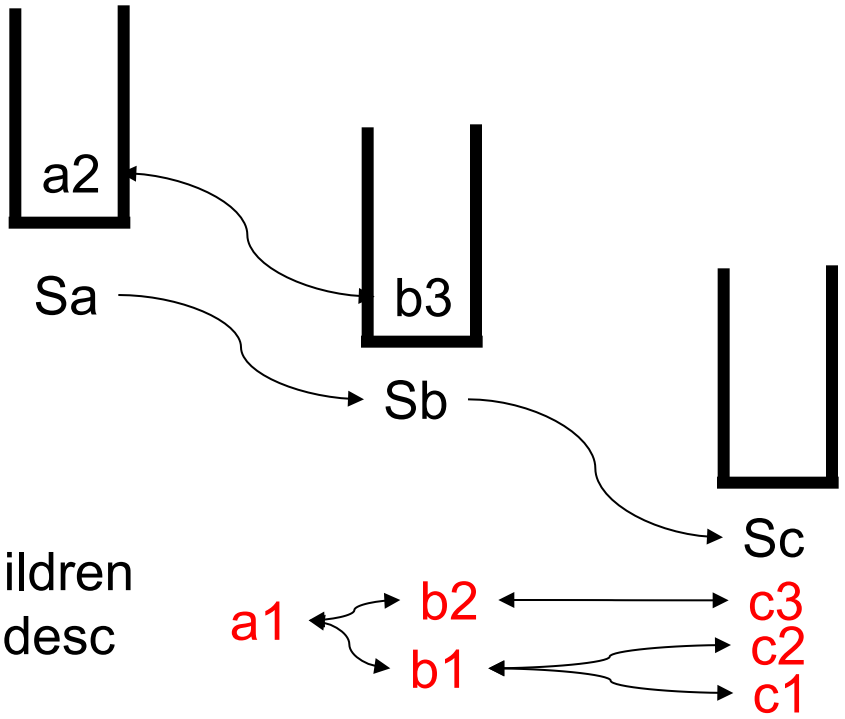


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Query: //a/b/c

On begin element x:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On end element x:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc

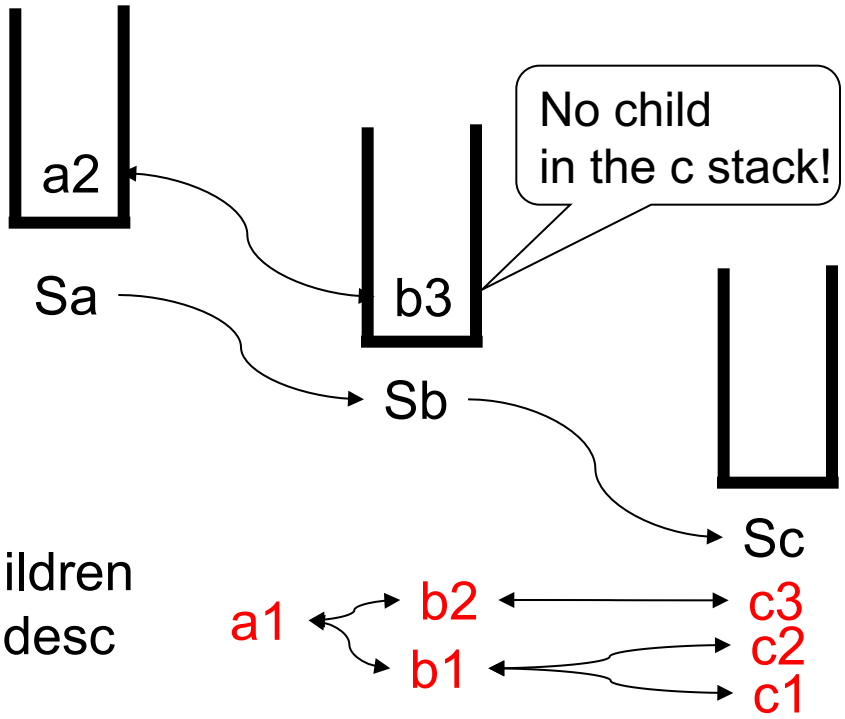


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Query: //a/b/c

On <u>begin element x</u>:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On <u>end element x</u>:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc

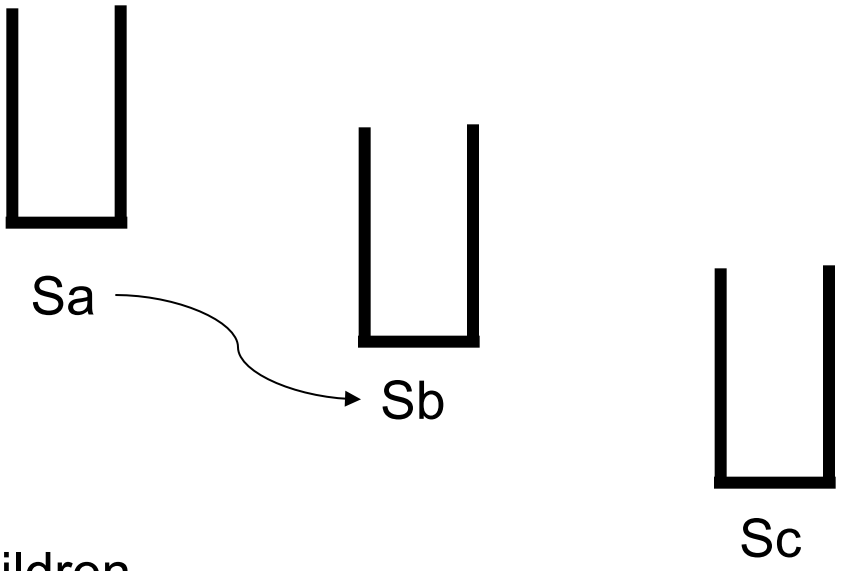


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

On begin element x:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On end element x:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc.

Query: //a/b/c

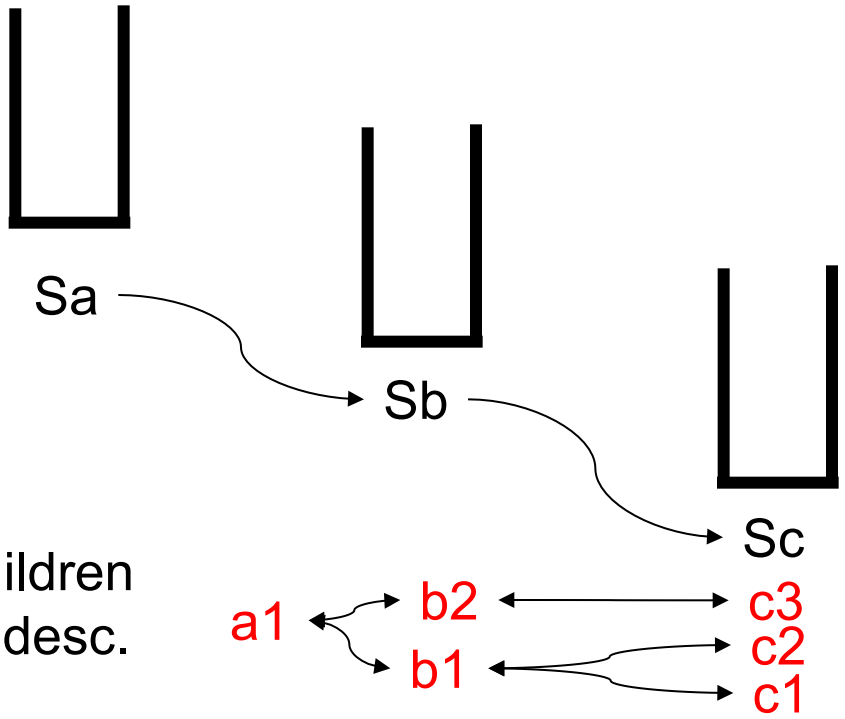


<r><a1><b1><c1/><c2/></b1><b2><c3/></b2></a1><a2><c4/><b3/></a2></r>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Query: //a/b/c

On begin element x:
If there is a stack for x
Then if the element appears
in the right context
then push it on the stack;
connect it to the
parent match
On end element x:
If there is a stack for x
Then if x is on top of the stack
then if x lacks some required children
then pop x, possibly some desc.



C

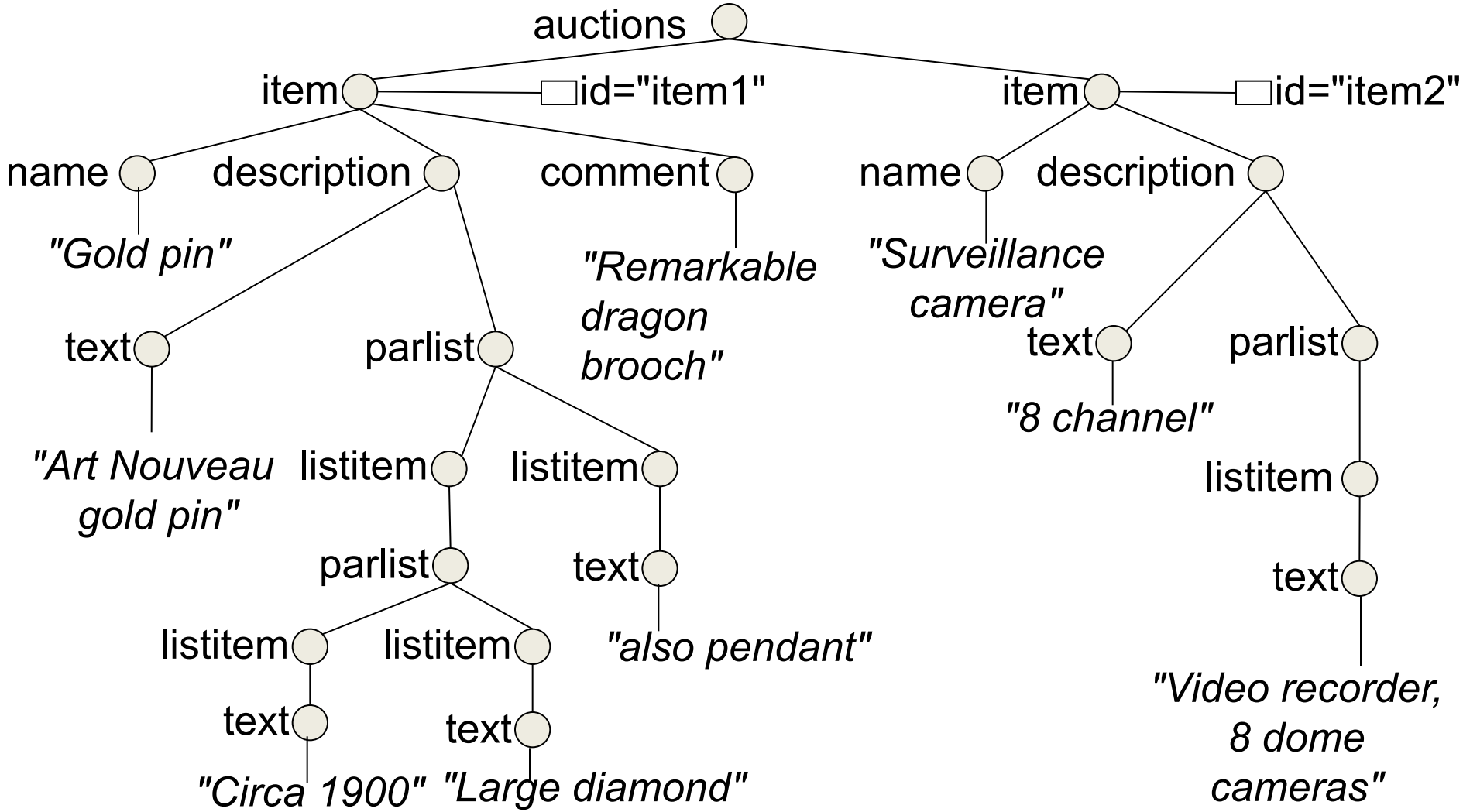
-
-
  - N
  - M
  - I

R ML

M

- C
  - M
- A
  - N
  - R
- M

ML


auctions
item
id="item1"
item
id="item2"
name
description
comment
name
description
"Gold pin"
"Remarkable dragon brooch"
"Surveillance camera"
text
parlist
text
parlist
"Art Nouveau gold pin"
listitem
listitem
"8 channel"
listitem
parlist
text
text
listitem
listitem
"also pendant"
"Video recorder, 8 dome cameras"
text
text
"Circa 1900"
"Large diamond"

N

element attribute text

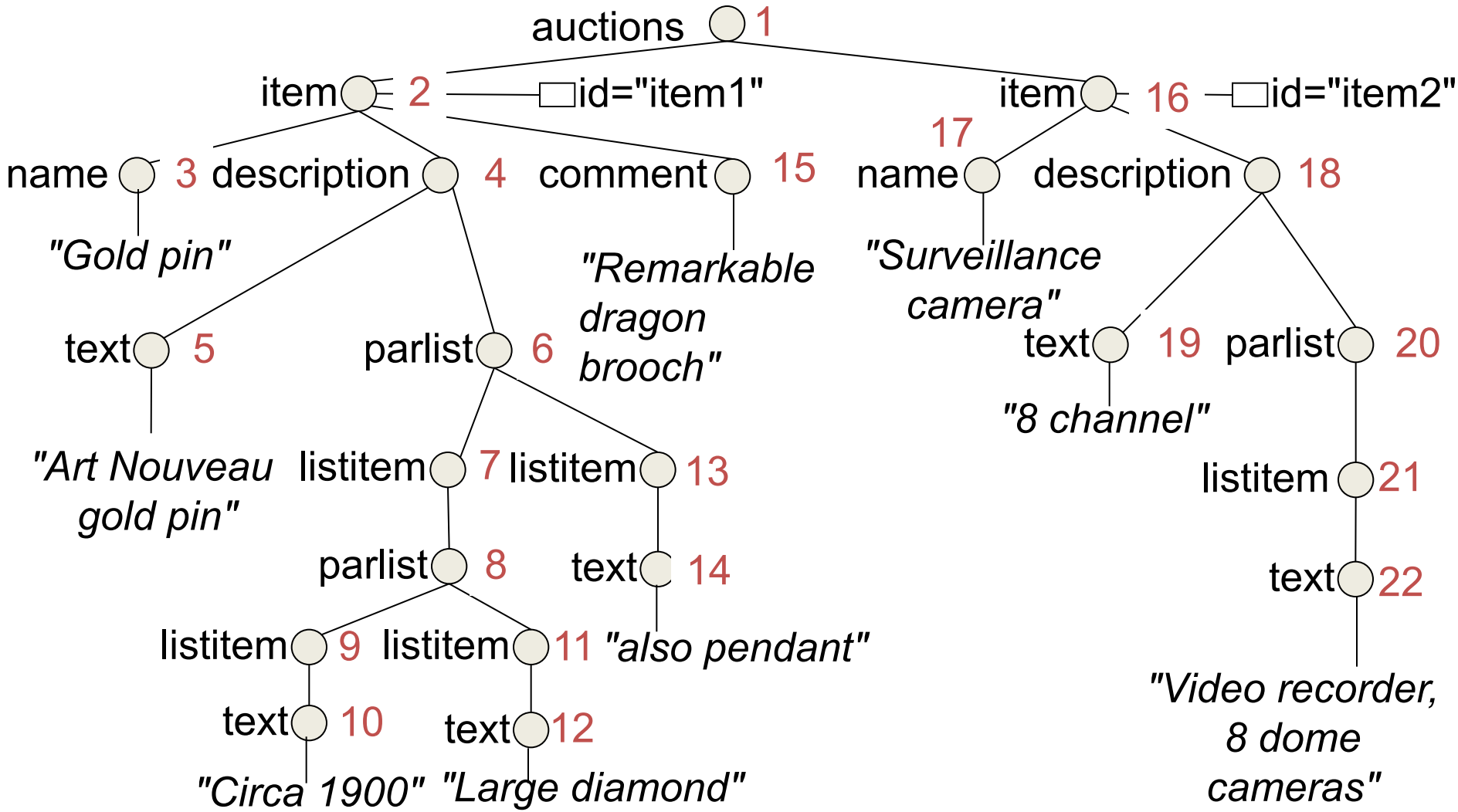
namespace, processing instruction, comment [XQDM]

Element, attribute PI

E ID ⇒

key issue is element identity

- I ID
- *Persistent stores: must materialize some persistent IDs (not necessarily for all elements)*

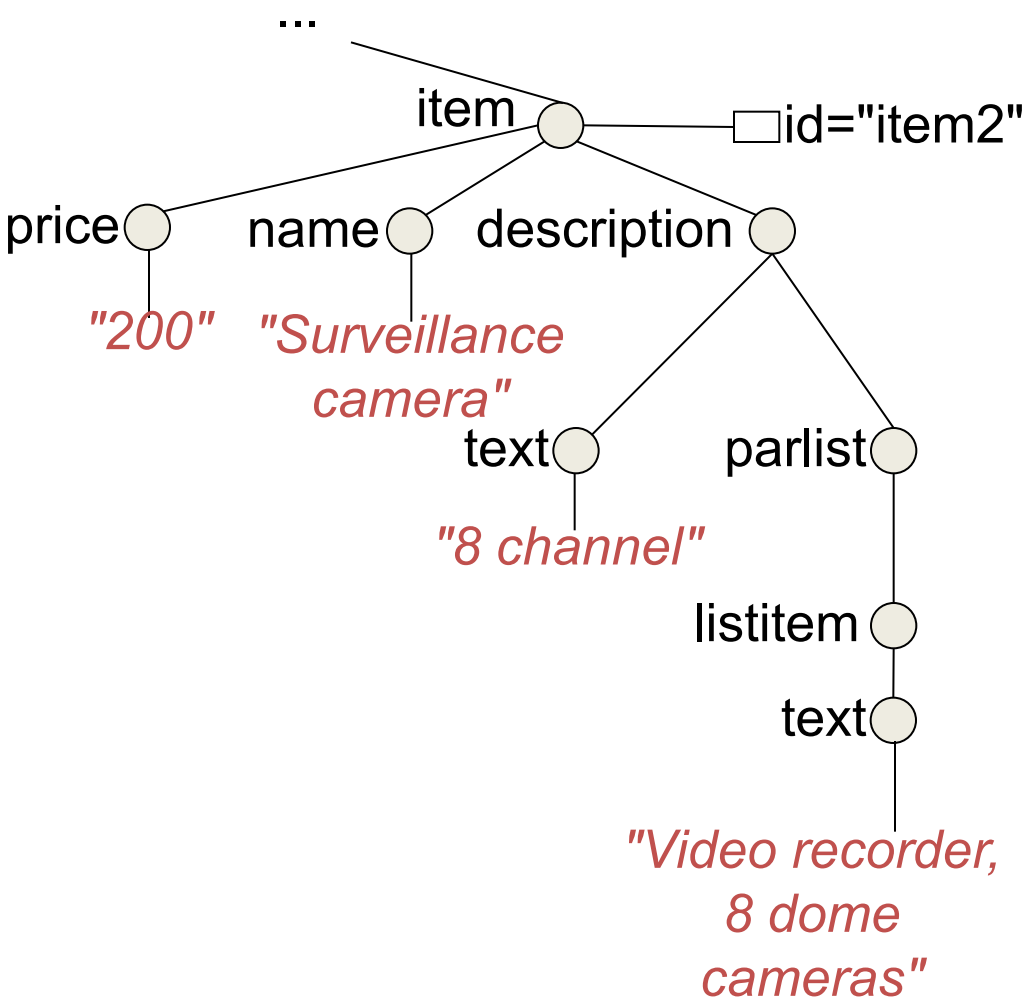

A
ID
auctions 1
item 2
id="item1"
item 16
id="item2"
name 3
description 4
comment 15
17
name
description 18
"Gold pin"
"Remarkable dragon brooch"
"Surveillance camera"
text 5
parlist 6
text 19
parlist 20
"8 channel"
"Art Nouveau gold pin"
listitem 7
listitem 13
listitem 21
parlist 8
text 14
text 22
listitem 9
listitem 11
"also pendant"
"Video recorder, 8 dome cameras"
text 10
text 12
"Circa 1900"
"Large diamond"

D

L

L

L

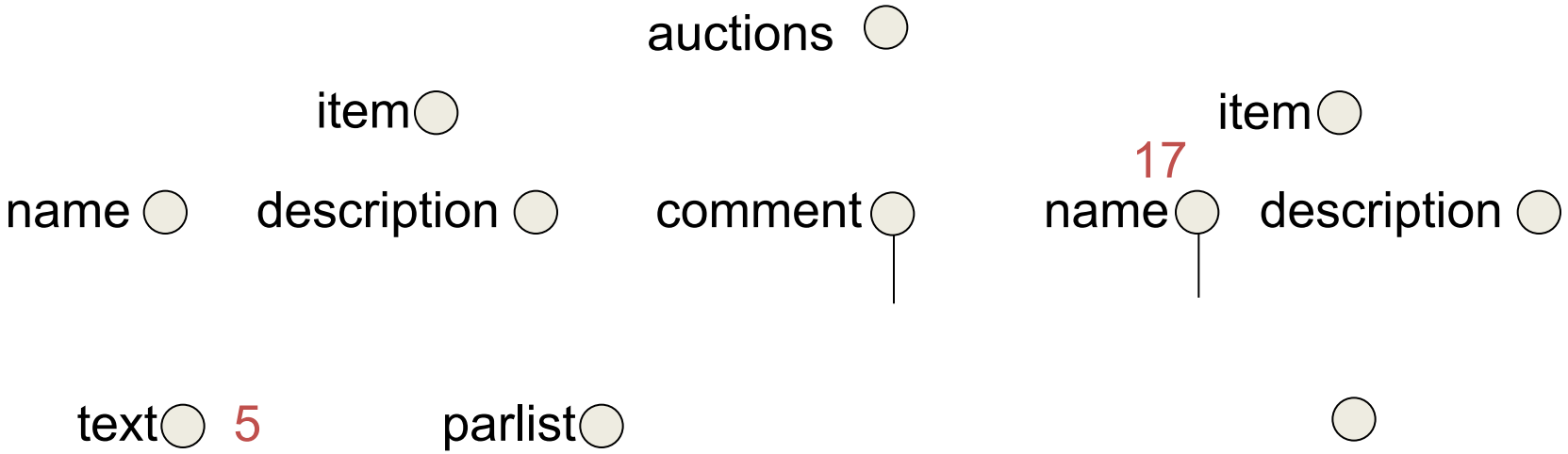

...
item
id="item2"
price
name
description
"200"
"Surveillance camera"
text
parlist
"8 channel"
listitem
text
"Video recorder, 8 dome cameras"

D

L

- G
  - I
  - I
- P
- O
-
- E

D



Element 1 is parent of elements 2 and 16

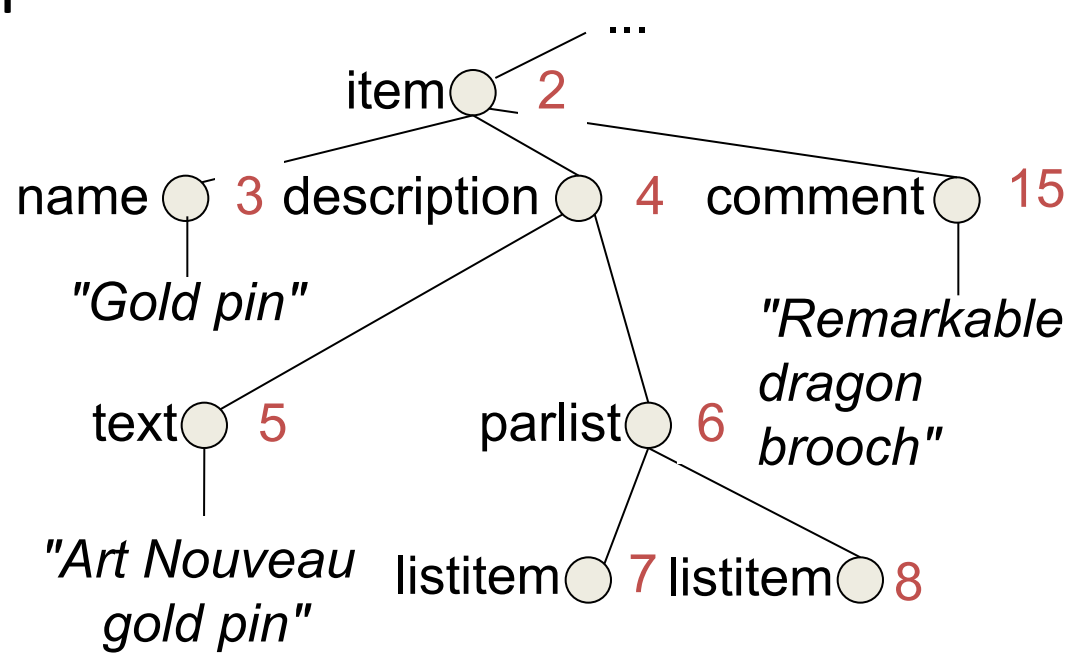
Element 2 has the attribute id="item1"

Element 3 has the text child "Gold pin"

D

N ML

I



Item name before item description

N

N

D

M

D D ML

C

ML

-
-
  - Choose some **property**: node name, node path,…
  - **Group** together all tuples/trees that have the same value for the same property
    E.g. table A contains all *A* elements
    E.g. collection C1 has all trees on path */A/B*
  - Store each group in a separate structure

- F
  - ML
- N ML D
  - ML
- C
  -

ML F

- ML
  CLOB
- F

- Q
  - N
  - F
  - N

ML N

- N ML
- ML
- E ML
- M

  – *Re-design features for XML (isolation, recovery, etc)*

  –

N I D L

- R
  - C
  - E ML API
  - A
- M
  - L

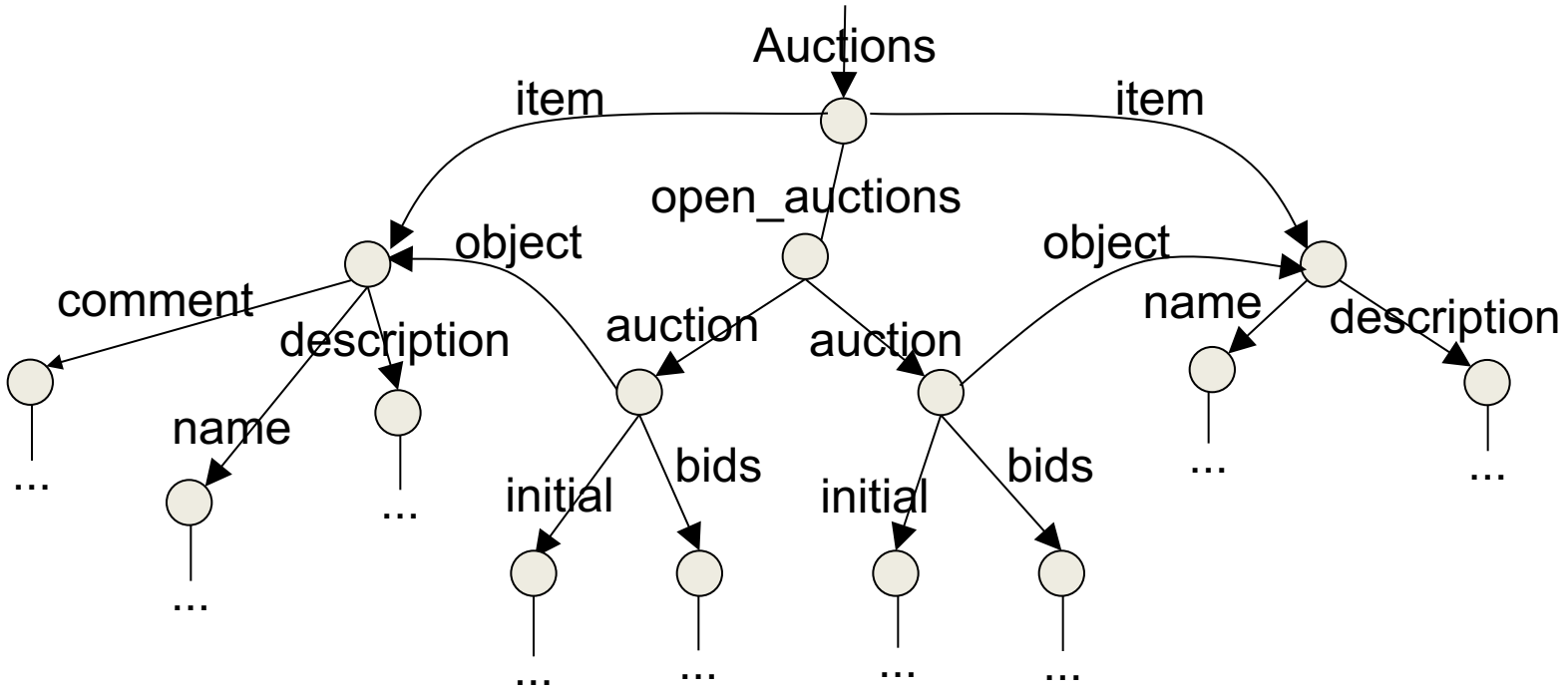
OEM PG

OEM: Object exchange model

Labeled, directed, unordered graph of objects

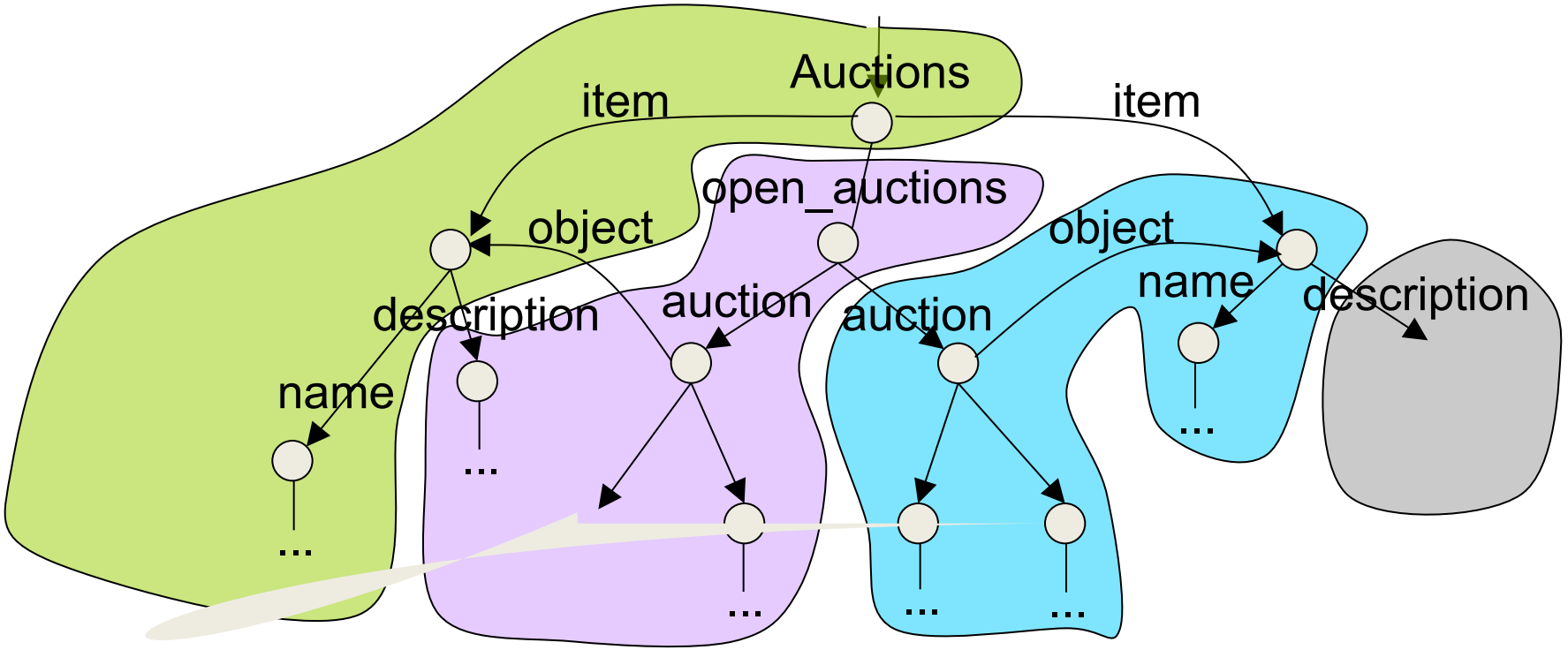
Objects have unique identity

Atomic objects = values (simple atomic types)



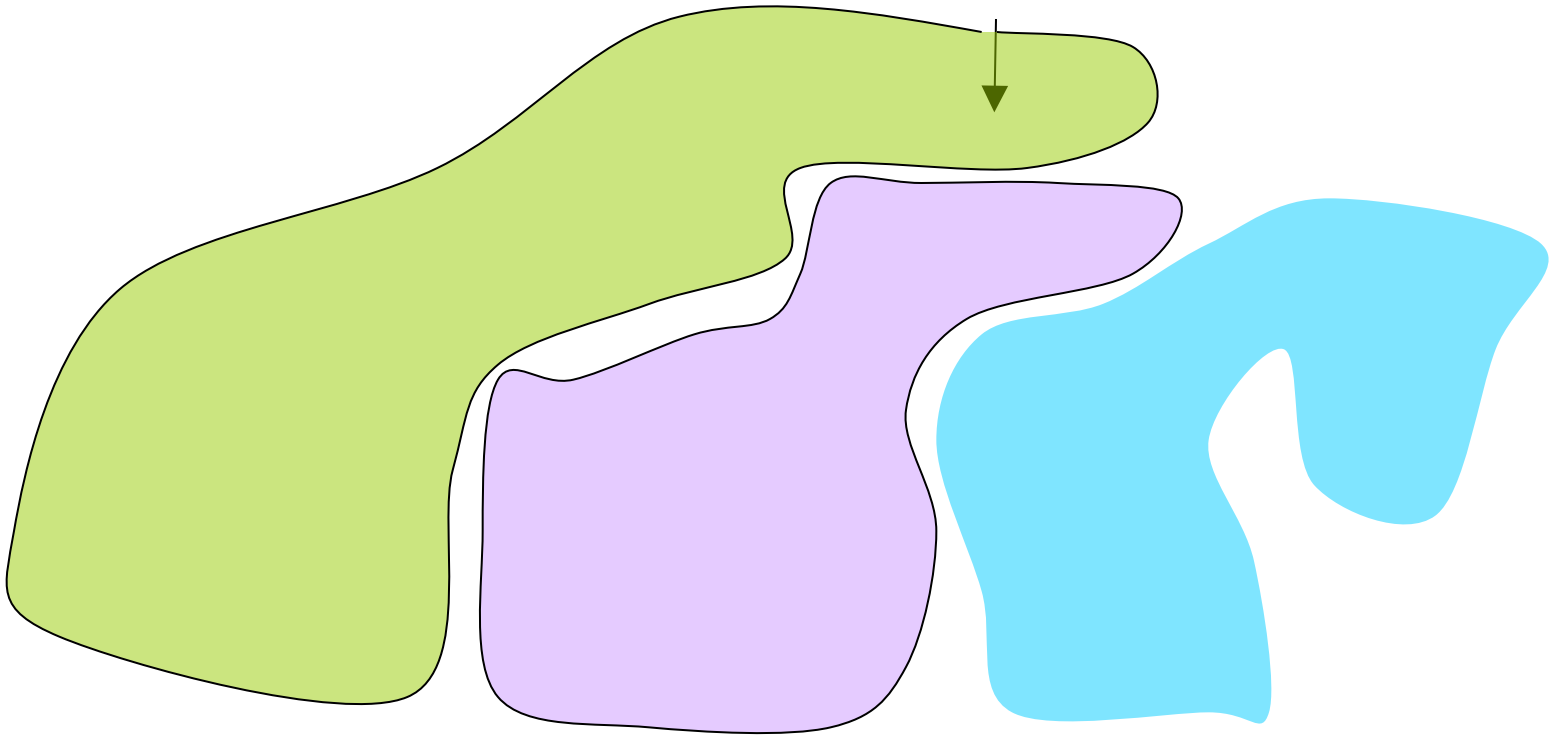
N

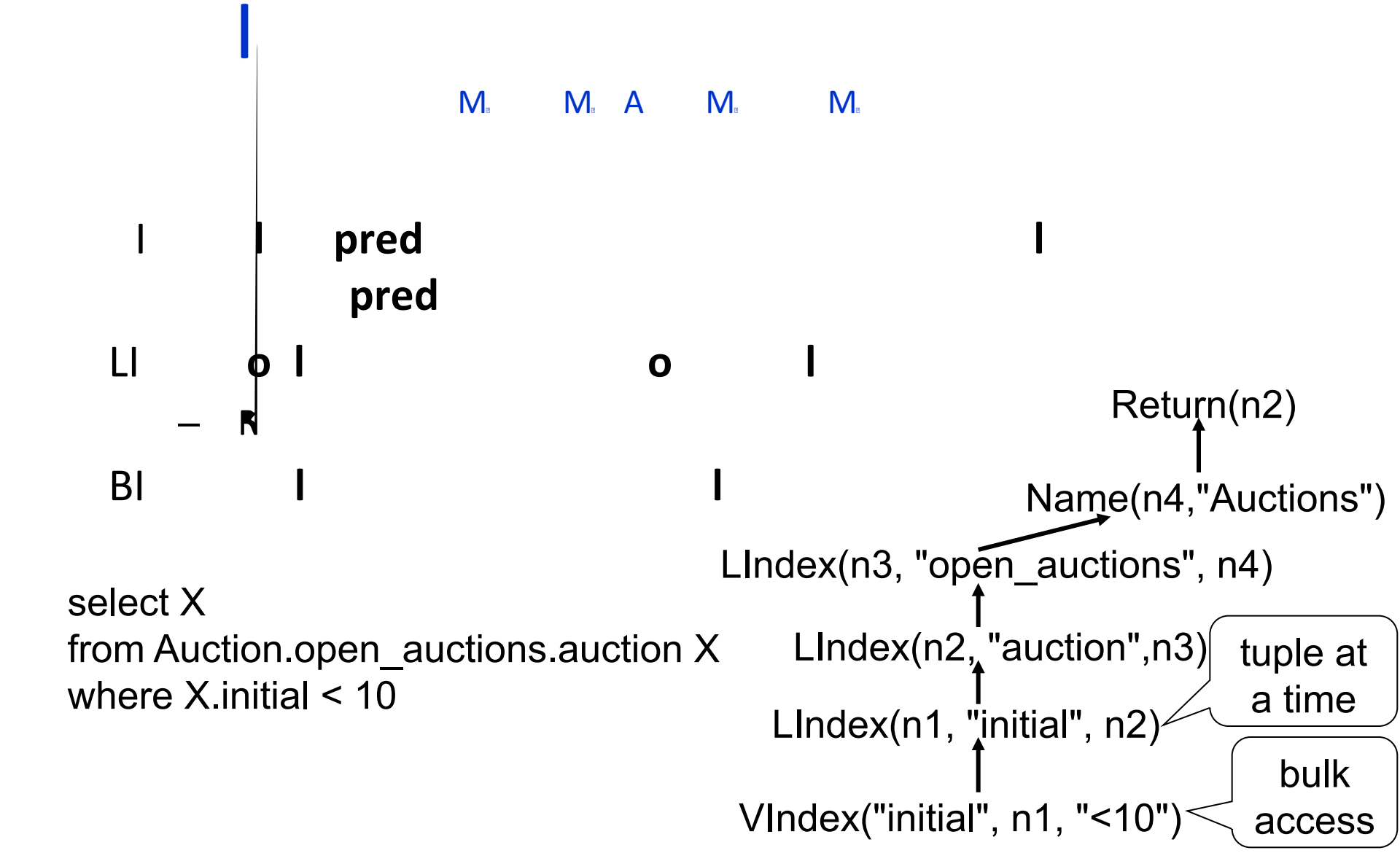
N


Auctions
item
item
open_auctions
object
object
description
auction
auction
name
name
description
...
...
...
...
...
...

N

Scan(Auctions, "item.description"):

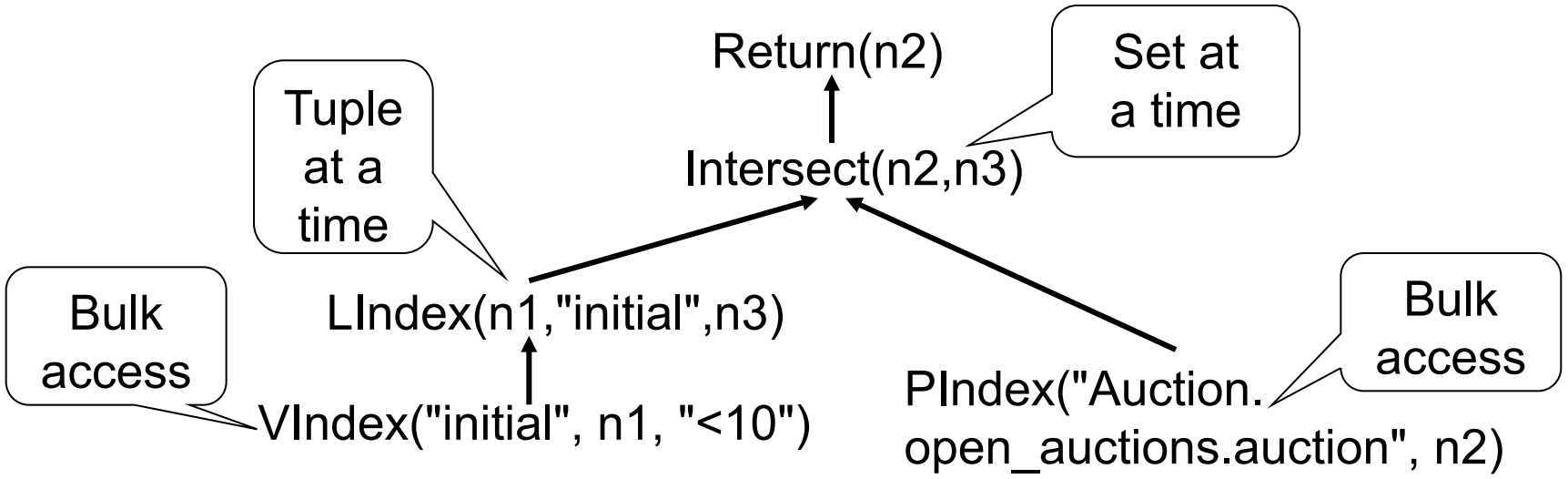





```
select X
from Auction.open_auctions.auction X
where X.initial < 10
```

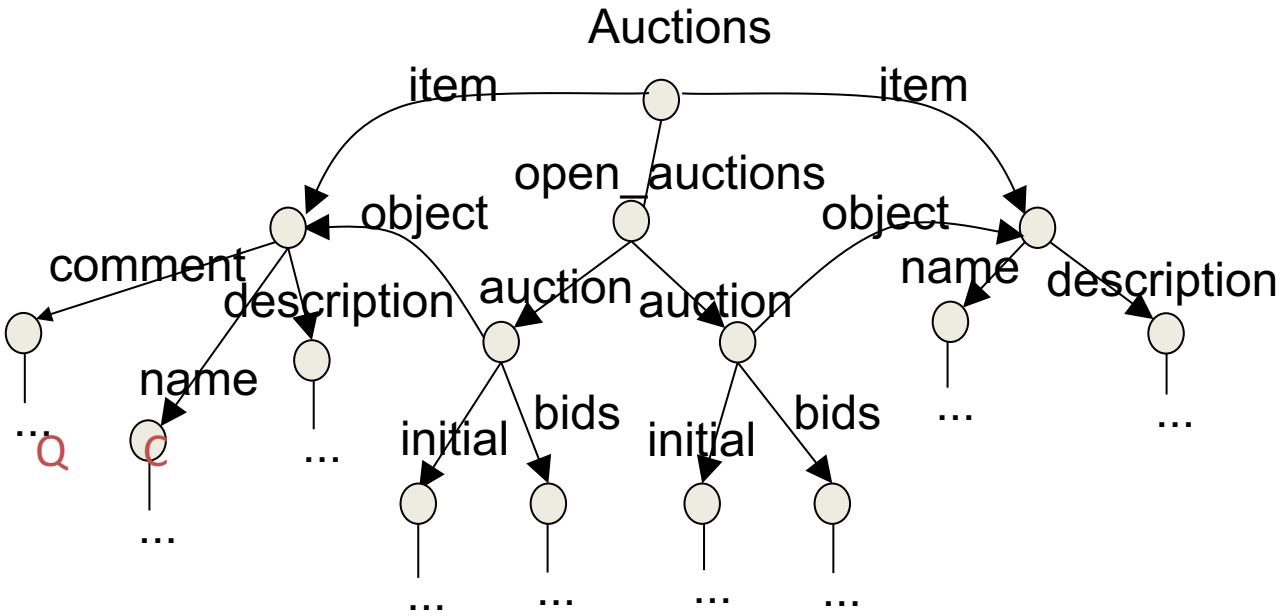
PI p

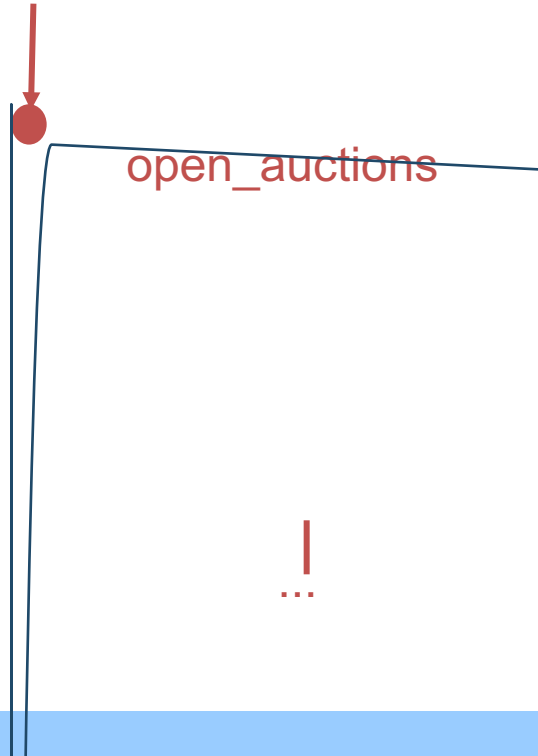
```
select X
from Auction.open_auctions.auction.initial  X
where X.initial < 10
```



D G

G

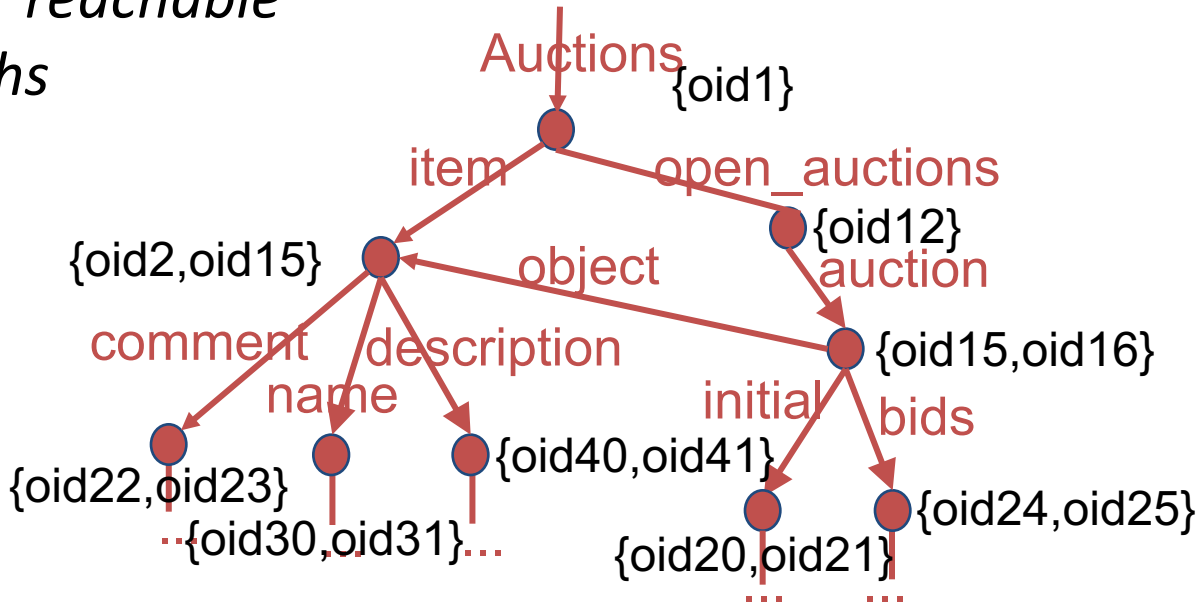

Auctions
item
item
open_auctions
object
object
comment
description
auction
auction
name
description
name
bids
initial
bids
initial
...
...
...
...
...
...
...
...
...
...
...
...


open_auctions
...

D G G

G

- I
- G *reachable by the same paths*



M

G

1. **Partition nodes into equivalence classes**
2. **Store the extent of each equivalence class, use it as "pre-cooked" answer to some queries**

Very

Q M A

M

- **Identify invariants / regularity / interesting node groups**
- **Use interesting node groups:**
  - **Simplify path queries**
  - **Basis for indexing:**
    - ***Store IDs of all nodes in an interesting group. Access them directly (avoid navigation).***

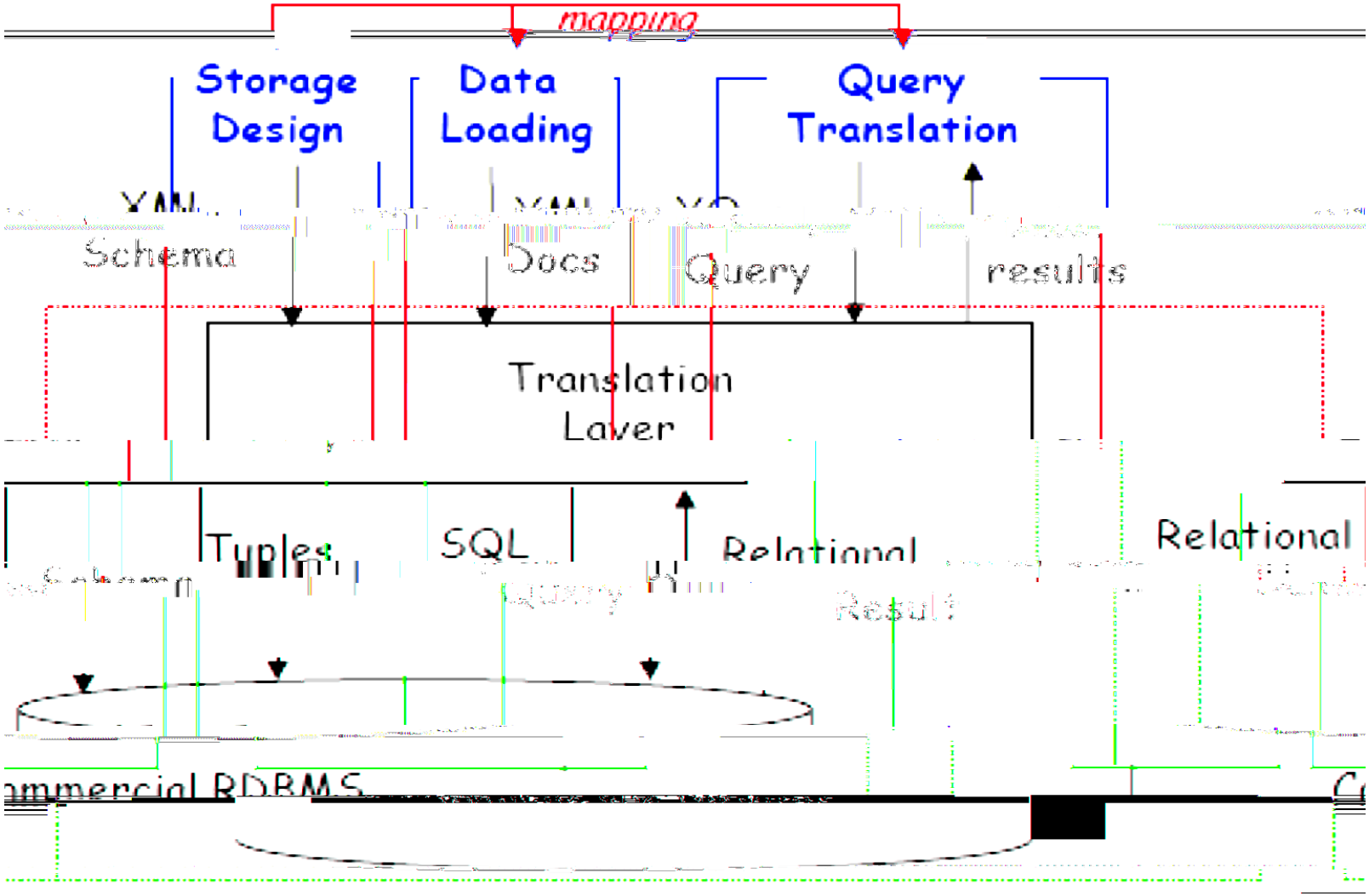
ML C

- R ML
  - E
- L
-
-
- Q
- M

I C I

- ML
  - ML →
- D ML
  - ML →
- Q ML
  - Q P → QL OQL
- R ML

ML RDBM

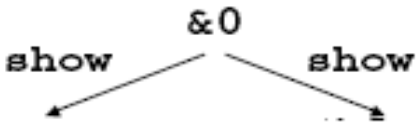

mapping
Storage Design
Data Loading
Query Translation
Schema
Docs
Query
results
Translation Layer
Tuples
SQL
Relational
Result
Relational

D

-
  - G
  -
  - D
  - C
  -

D D

G M E

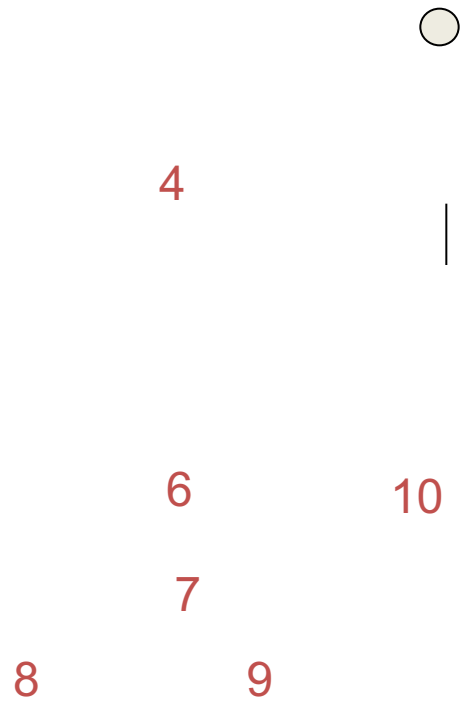


Edge Table

source	Child no.	tag	target
&0	1	show	&1

Find titles for all sho

SELECT Value , value


4
6
10
7
8
9

P

E

Edge(pID, ord, name, target)

-	1	auctions	1
1	1	item	2
2	1	id	"item1"
2	2	name	"Gold pin"
2	3	comment	"Remark.."
3	1	text	"Art..."
3	2	parlist	5
.....	.....	.....	.....

//item

//comment

//item [@id="i description

select target from Edge where name="item"

Actual query answering also requires reconstructing the element...

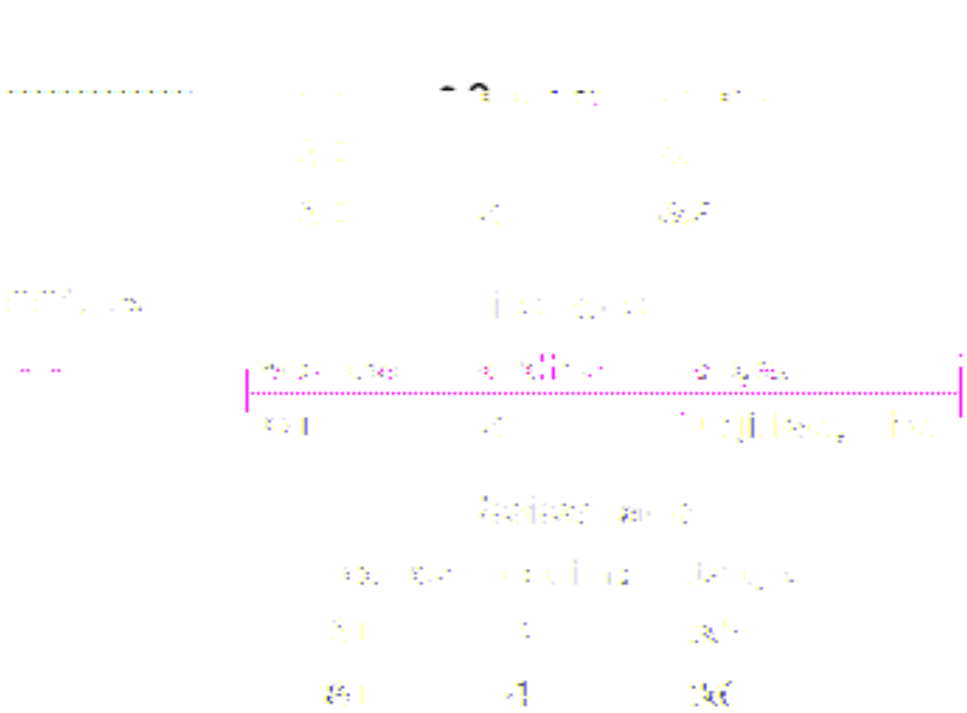
e2,

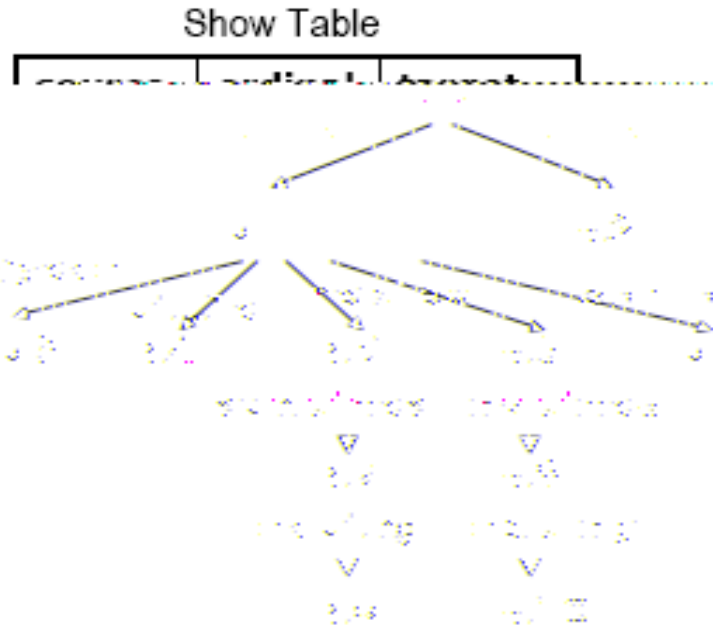
where e1.name="item" and e1.target=e2.pID and e2.name="id" and e2.target="item1" and e1.target=e3.pID and e3.name="descr"

3-way join on the Edge table

(Index-) join algorithms better than navigation
Still, too much data to read

G M A





e.source

Find titles for all shows

```
SELECT Title.target
FROM Title, Show
WHERE Show.target=Title
```

P E

EdgeAuction ID

EdgeItem ID

EdgeID ID

EdgeDescription ID

```
//item[@id="item1"]/description

    select e3.target
    from EdgeItem e1, EdgeID
         e2, EdgeDescription e3
    where e1.target=e2.pID and
          e2.target="item1" and
          e1.target=e3.pID and
```

3-way join on (much) smaller tables

Si nila g a h i de ,
b ***this is the storage***

l e e i g g f de : ***thos e label***

Store tags in schema, not in data

Some code on the side keeps the mapping between tags and table names

G M

- I
- C
  - **Edge: store all edges in one table**
  - **Attribute: horizontal partition of Edge relation on element tag**
- Q
  - **Requires multi-table joins or self joins for element reconstruction**
  - **Transitive closure for answering descendant queries**

D M

- J K G H R
  D Q ML D L
  O LDB
- I D D R
  - E
  - A C
  - N
  - I
- D D
- A ML

D D N

- D D
  - **<!ELEMENT a ((b|c|e)?,(e?|(f?,(b,b)*))*)>**
- D D D D

- P I D
  D D
  almost D
  –

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^{*?} \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

(b c e)?,(e? f+)

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^*? \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^*? \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

(b c e)?,(e? f+)

(b?,c?,e?)?,e??,f+?

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^*? \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^*? \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

(b c e)?,(e? f+)

(b?,c?,e?)?,e??,f+?

b??,c??,e??,e??,f+?

b??,c??,e??,e??,f*?

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^*? \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

(b c e)?,(e? f+)

(b?,c?,e?)?,e??,f+?

b??,c??,e??,e??,f+?

b??,c??,e??,e??,f*?

b?,c?,e?,e?,f*

$(e_1, e_2)^* \rightarrow e_1^*, e_2^*$

$(e_1, e_2)? \rightarrow e_1?, e_2?$

$(e_1\ e_2) \rightarrow e_1?, e_2?$

$e_1^{**} \rightarrow e_1^*$

$e_1^*? \rightarrow e_1^*$

$e_1?^* \rightarrow e_1^*$

$e_1?? \rightarrow e_1?$

$e_1+ \rightarrow e_1^*$

..., a*, ..., a*, ... → a*, ...

..., a*, ..., a?, ... → a*, ...

..., a?, ..., a*, ... → a*, ...

..., a?, ..., a?, ... → a*,

, ...a, , a, → a*,

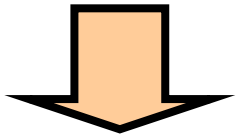
(b c e)?,(e? f+)

(b?,c?,e?)?,e??,f+?

b??,c??,e??,e??,f+?

b??,c??,e??,e??,f*?

b?,c?,e?,e?,f*

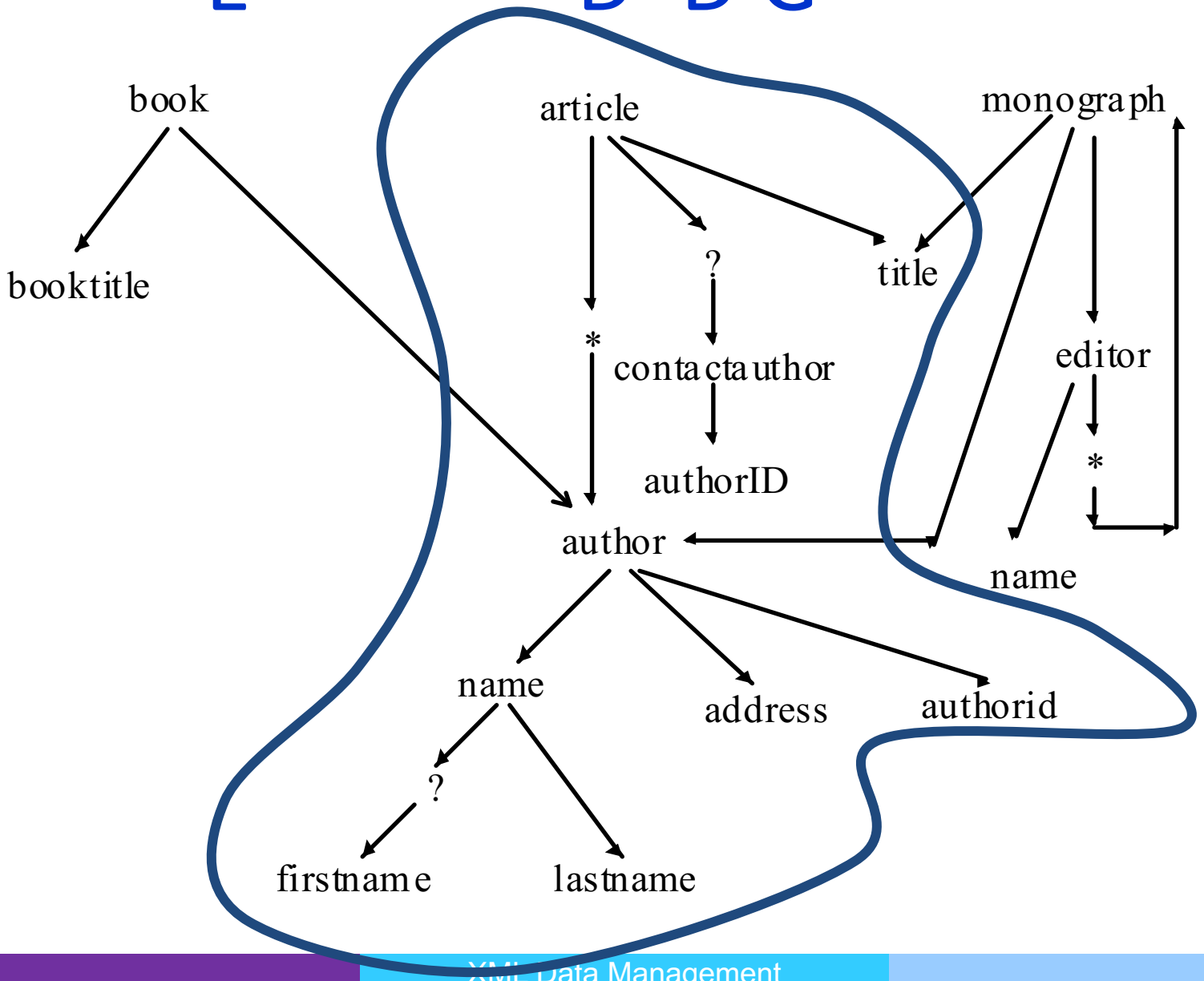


b?,c?,e*,f*

D D G

- I D D
  D D
  D D
- I
  D D
- E
- A
  D D
- C

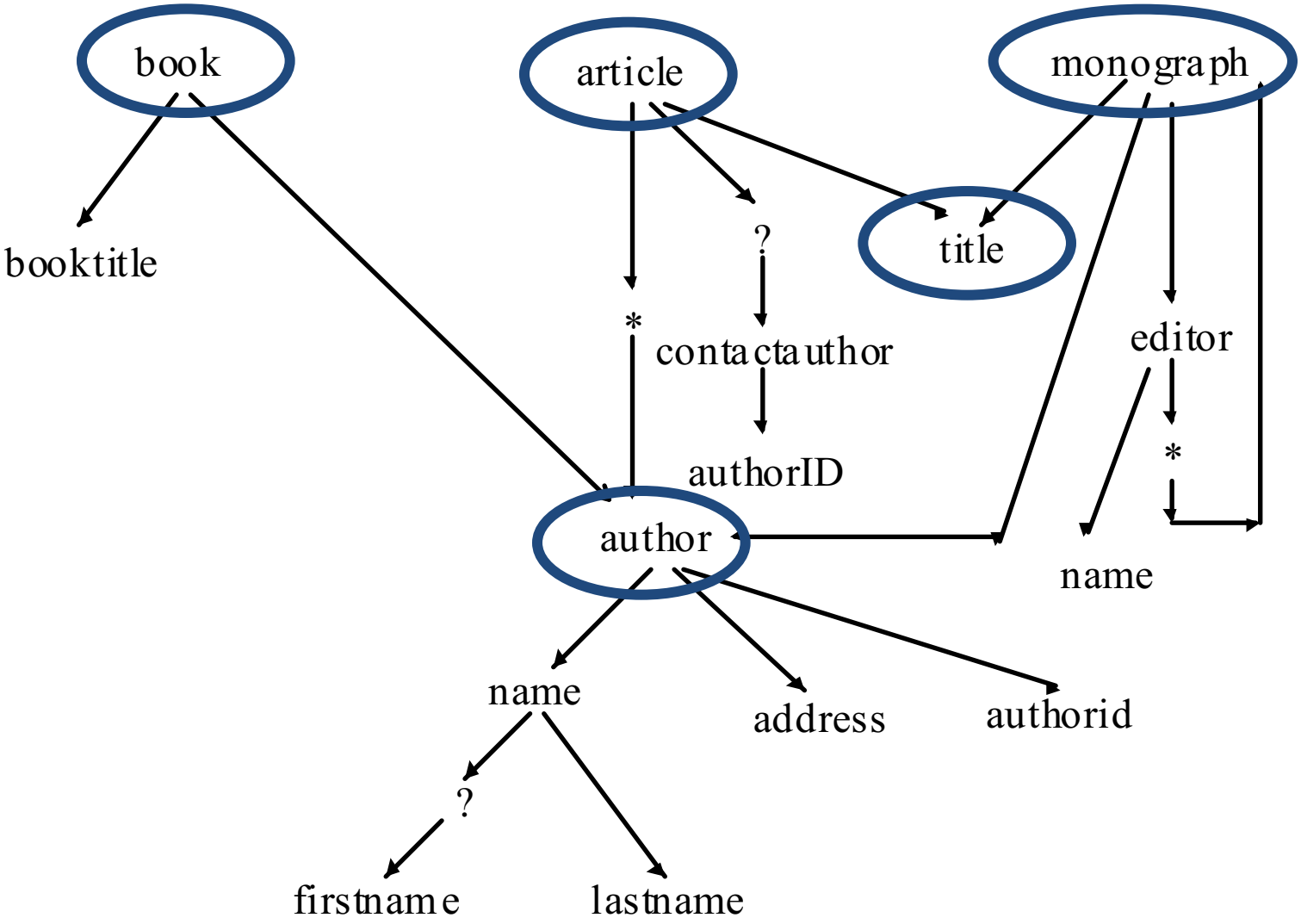
E D D G

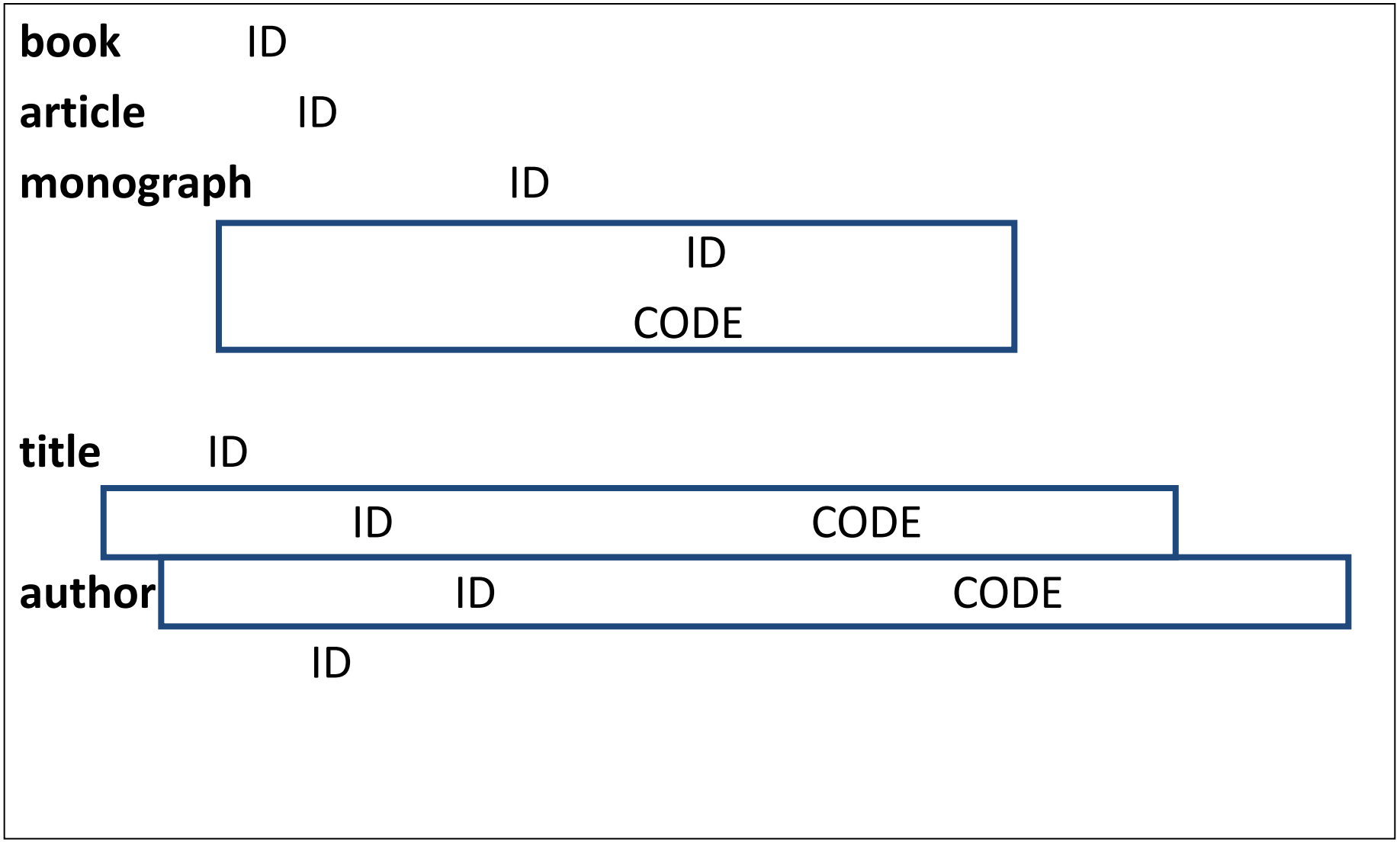

book
booktitle
article
?
title
*
contactauthor
authorID
monograph
editor
*
author
name
name
address
authorid
?
firstname
lastname

C

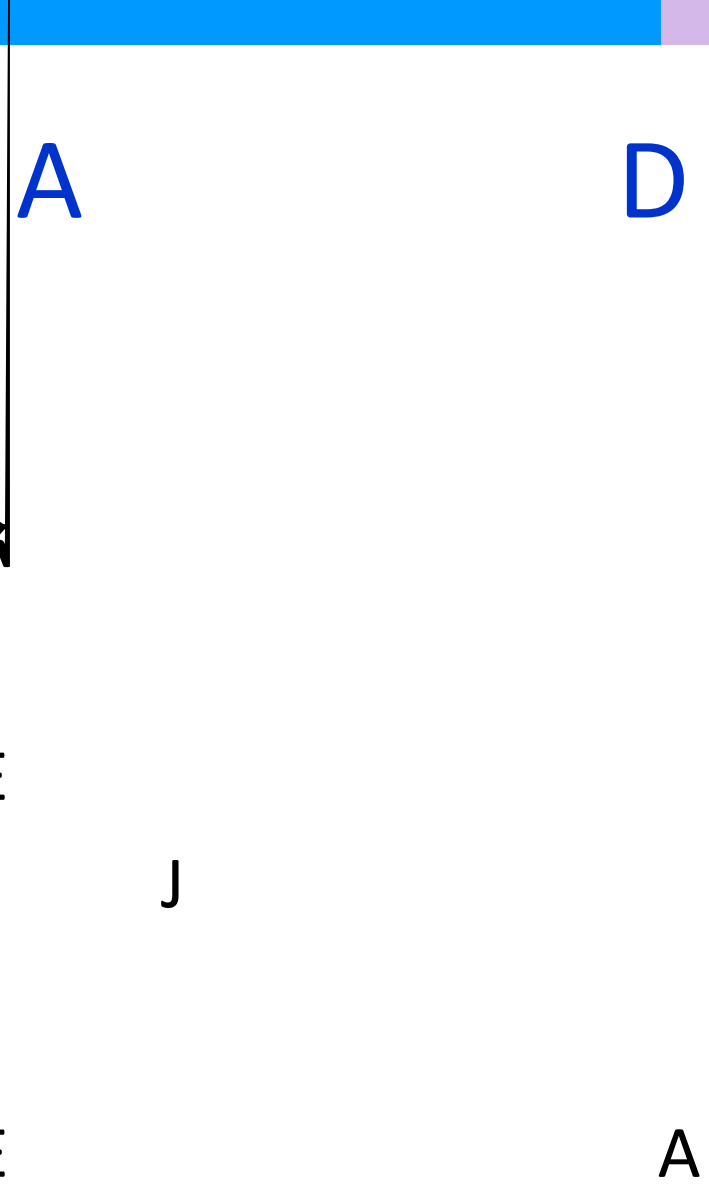
I

- D D
  - –
  - –
  - –
  - –
- A


book
booktitle
article
*
?
contactauthor
authorID
monograph
title
editor
*
name
author
name
address
authorid
?
firstname
lastname


book ID
article ID
monograph ID
ID
CODE
title ID
ID CODE
author ID CODE
ID

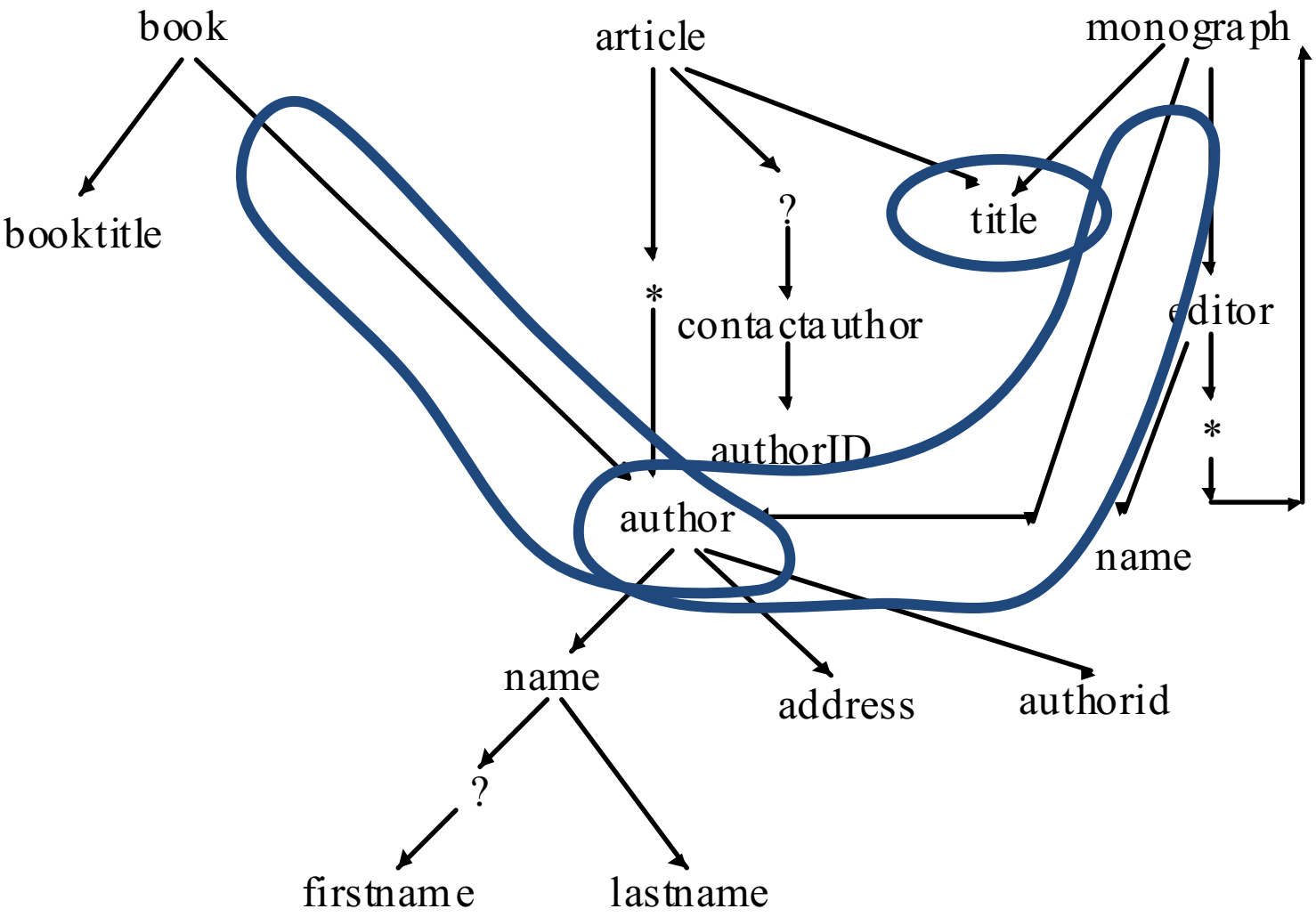
Wha a e he e f ?

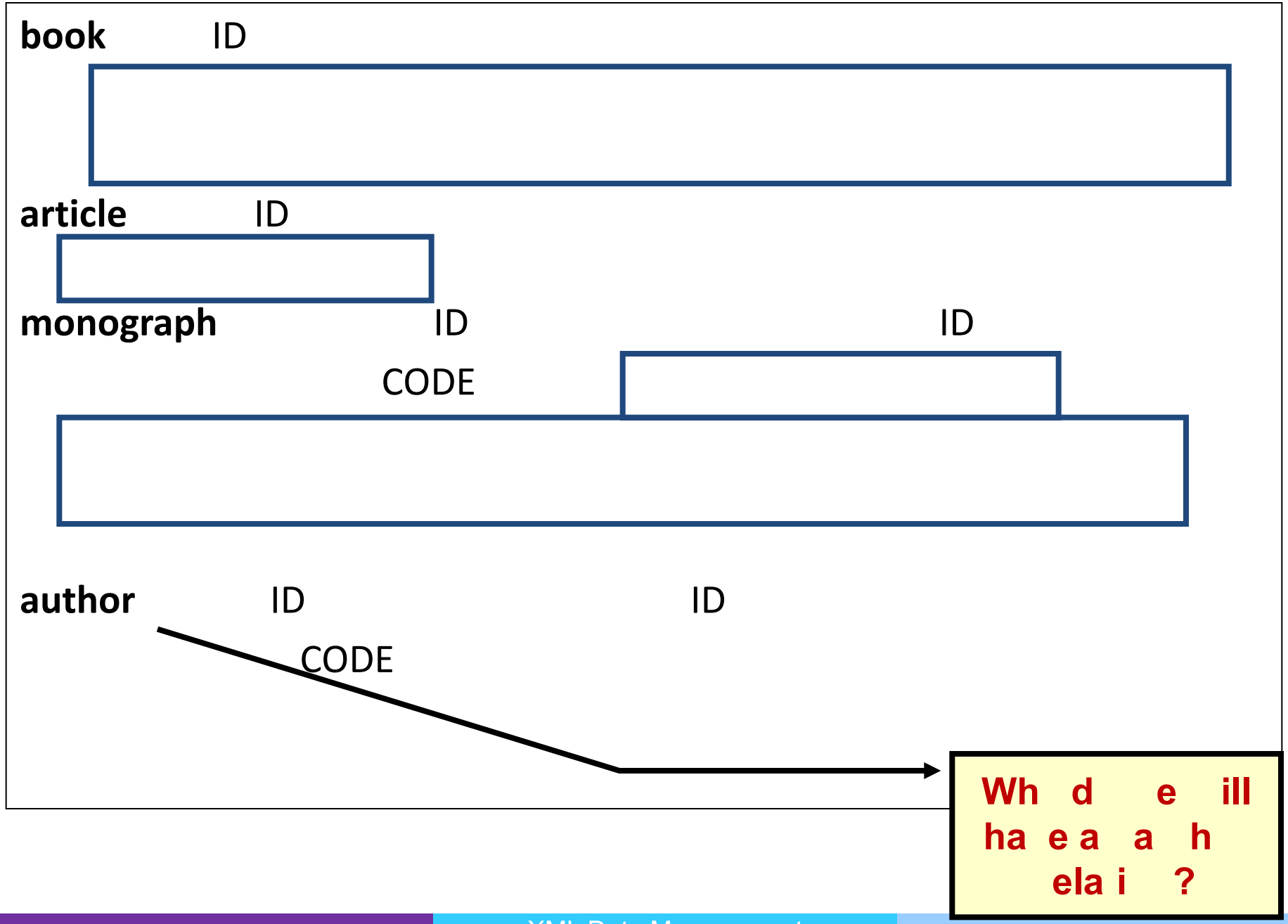


- A
  - R
  - E
    J
- D
  - E
    A

H I

-


book
booktitle
article
?
*
contactauthor
authorID
author
name
?
firstname
lastname
address
title
monograph
editor
*
name
authorid


book
ID
article
ID
monograph
ID
ID
CODE
author
ID
ID
CODE
Wh d e ill
ha e a a h
ela i ?

A D

- A
  - R
  - R
- D
  - R QL

J

-

  - H

D

- D D ML
- H
  - R
  - H
- Q
  - F
  - R

P C

- D L C C P
  ML D D R ER
- D D
  - D A LI
  - C ELEMEN
  - I A LI IDREF REQ IRED
- H

QL

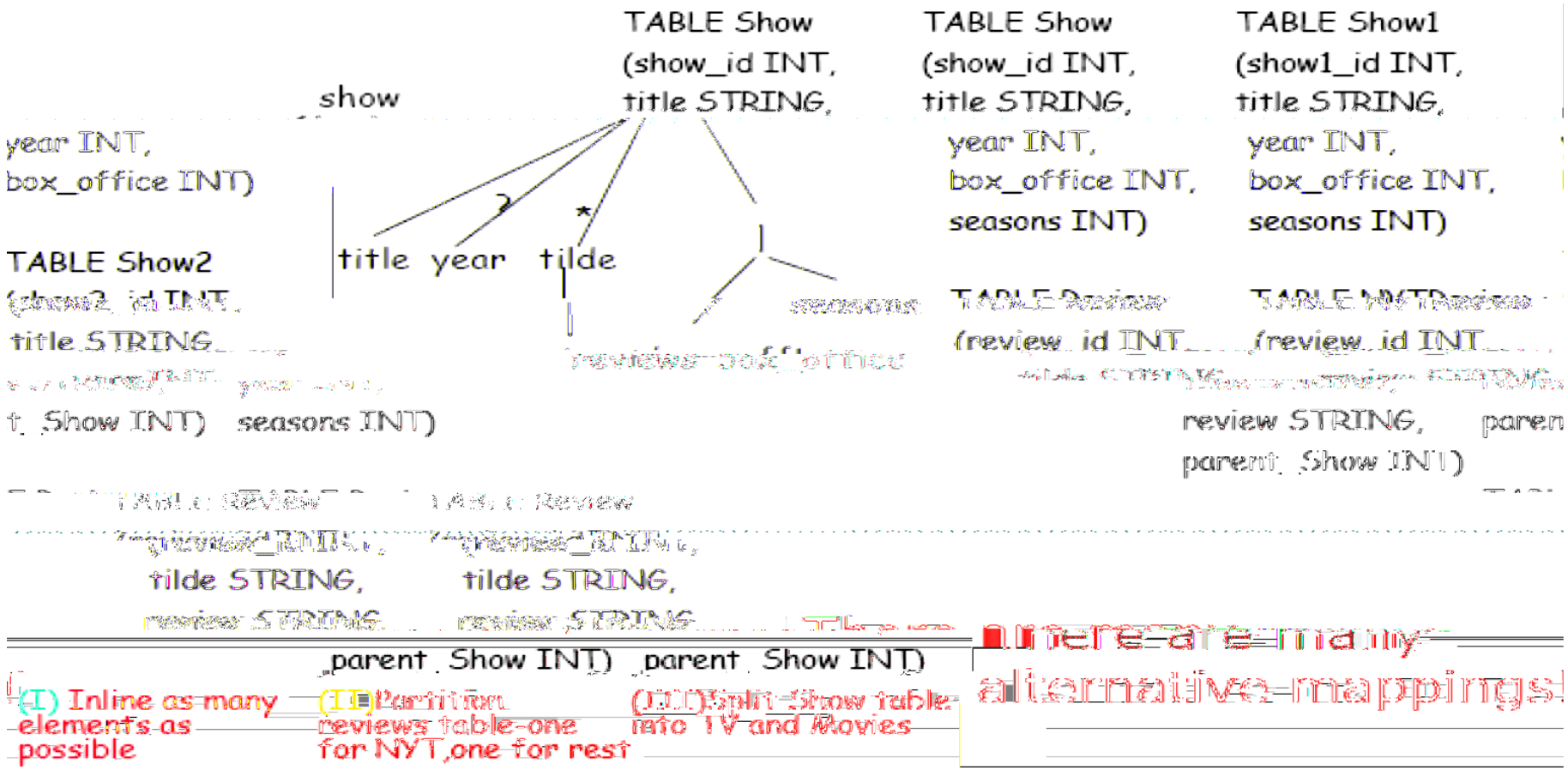
NO N LL NIQ E

  - ID

D D ORED

-
- A
- E
  - D IMDB
  - C
  -
- Q

M M

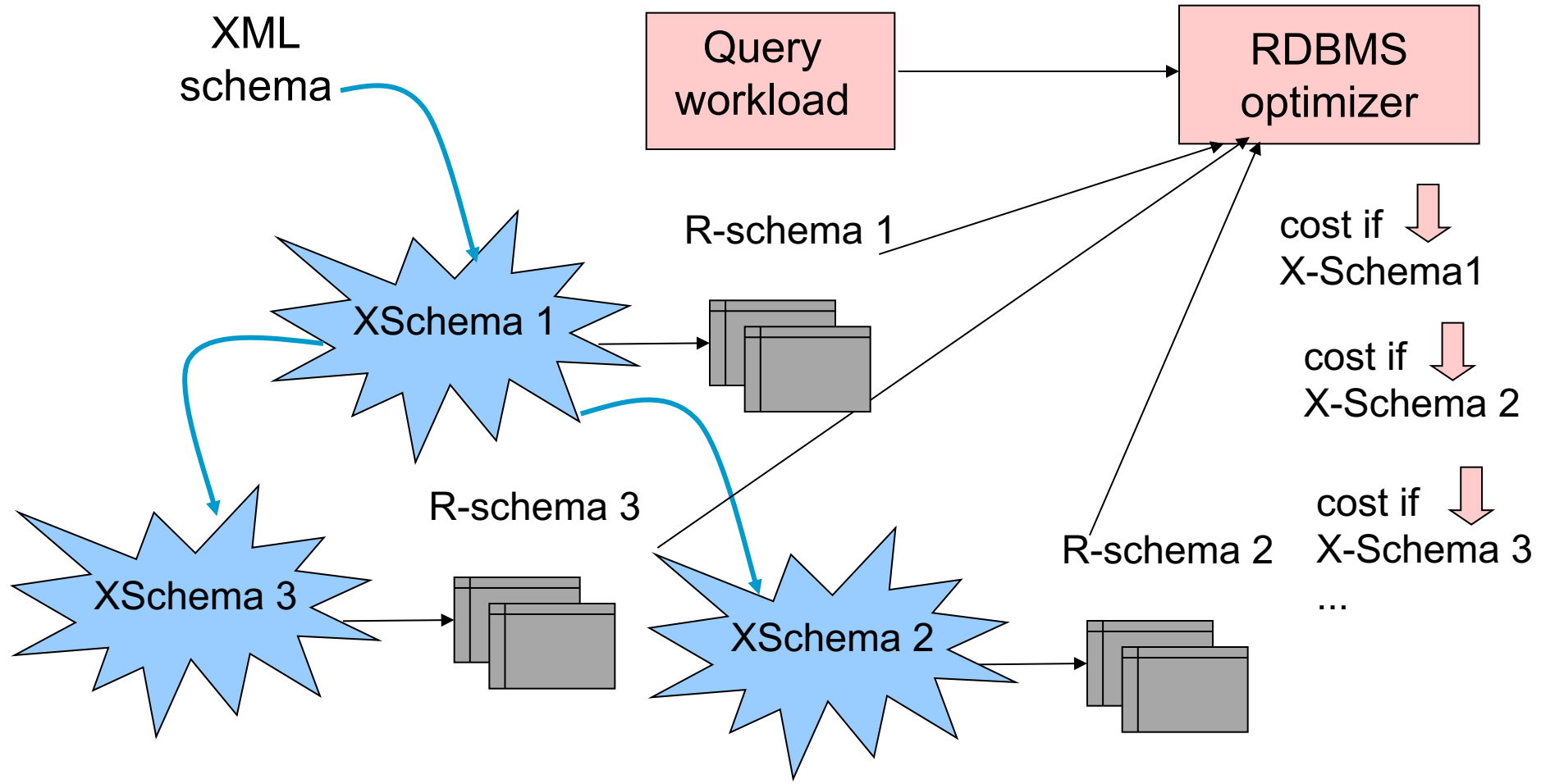


- Performance depends on data, schema and query workload
- A fixed mapping is unlikely to be the best for all applications

C B L DB

- A
- A
  - *schema data statistics query workload*
-
  -

C B


XML
schema
Query
workload
RDBMS
optimizer
R-schema 1
cost if
X-Schema1
XSchema 1
cost if
X-Schema 2
R-schema 3
cost if
X-Schema 3
...
R-schema 2
XSchema 3
XSchema 2

- I O
  - A I C C
    A I
  - *Inlining useful if C is always queried through ancestor A*
- F D
  -
  -
  - *Useful to separate if a[t1] often queried together, a[t2] rarely or never queried together*

-
  -
  - *If the first <a> is isolated, it can be inlined with parent*
-
  - A
  - *If a/b/tag1 often queried, a/b/other never queried, separate them.*

C B

- M
- O

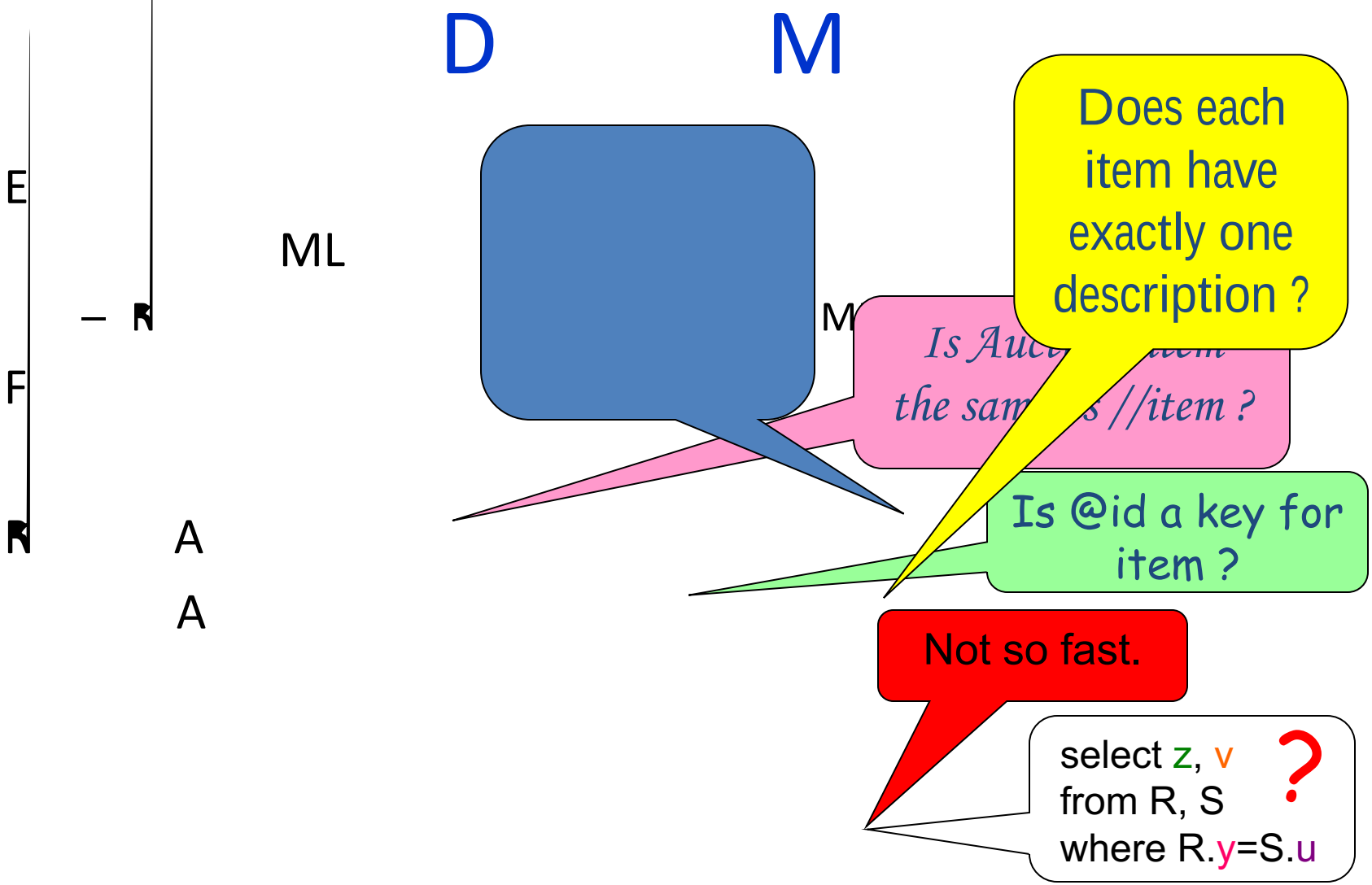
  the same mistake

-
  - N
  -
  -

unions *joins*

D M

- DBM
  -
- F
-
  - ML
  - M
    - H
  - D →


D
M
E
ML
F
A
A
Does each item have exactly one description ?
Is @id a key for item ?
Not so fast.
select z, v
from R, S
where R.y=S.u
?

D M

Express (relational) storage by custom expressions over the XML document

- Relation = materialized view over the XML document

Finding useful tables requires view-based query rewriting

XPath containment

Is Auctions.item the same as //item ?

Functional dependency

Cardinality c ai

D M

- E
  ML
- M
- M

- C
- *Rewriting is complex.*

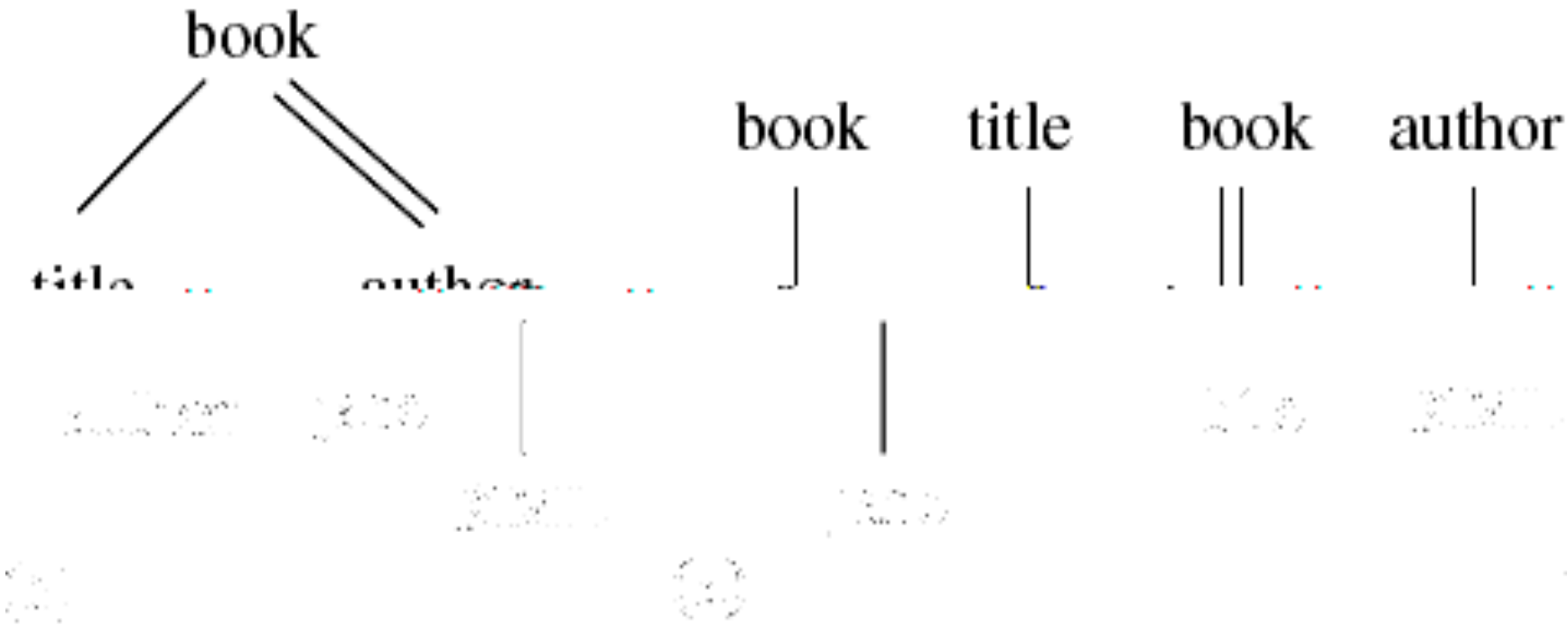
- P
- L
  - A ID
  - M

ML

- R *alone*
- M *S-P-J materialized view selection over partitioned Edge table*
- F *view-based query rewriting*
- I *encodings: path, ID, ...*
- F *facilitates navigation complicates reconstruction*

J A

P Q

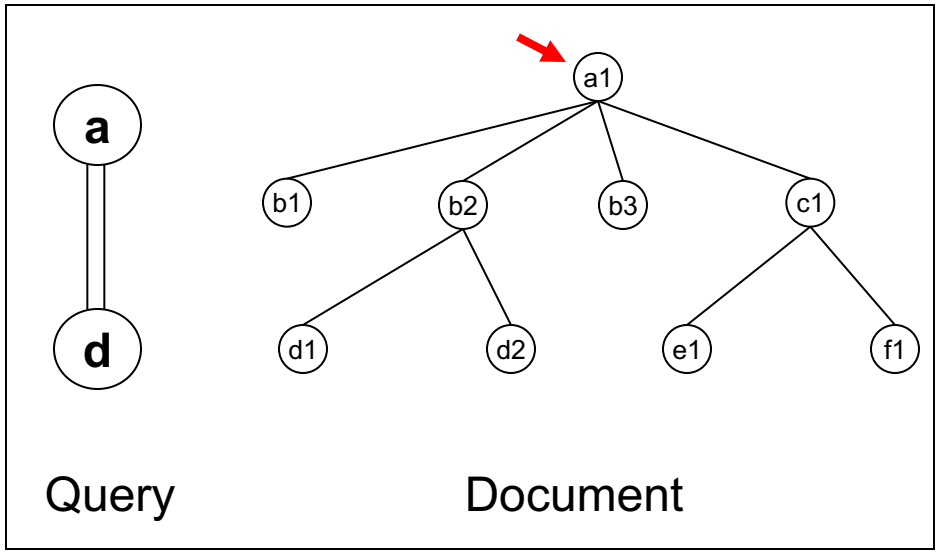


Q E M

- I
  - N
  -
-
  -

N

- N



ML N L

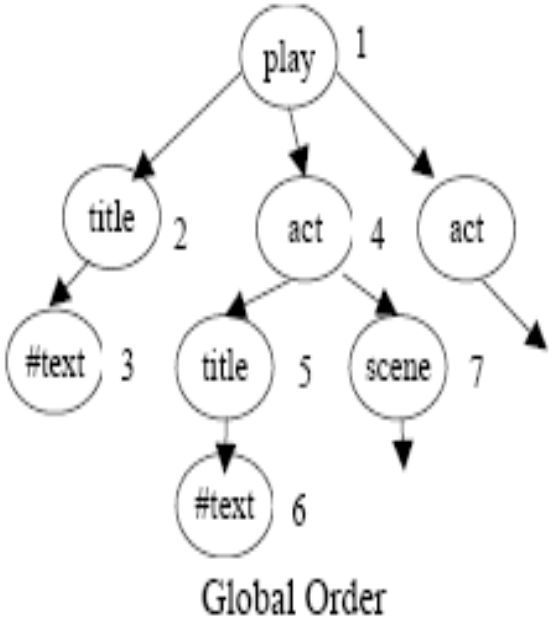
- ML RDB
  RDB
- ML *ordered*
  unordered
- H *ordered* ML
  unordered
- E *order*
- ML

L

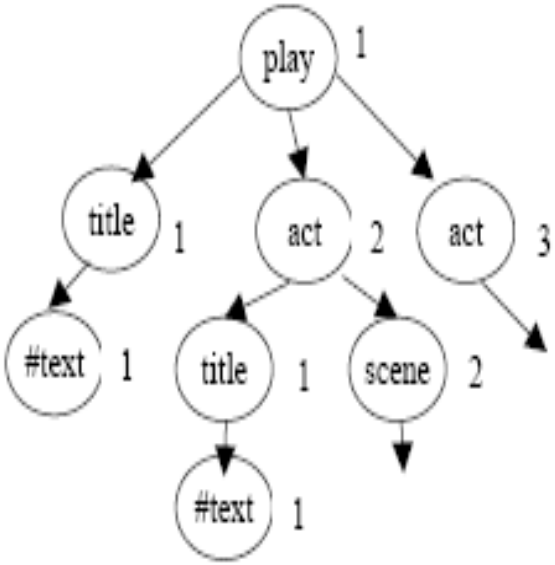
- I
  - –
  - –
-
- M
  - –
  - –
    - •
  - –

ML

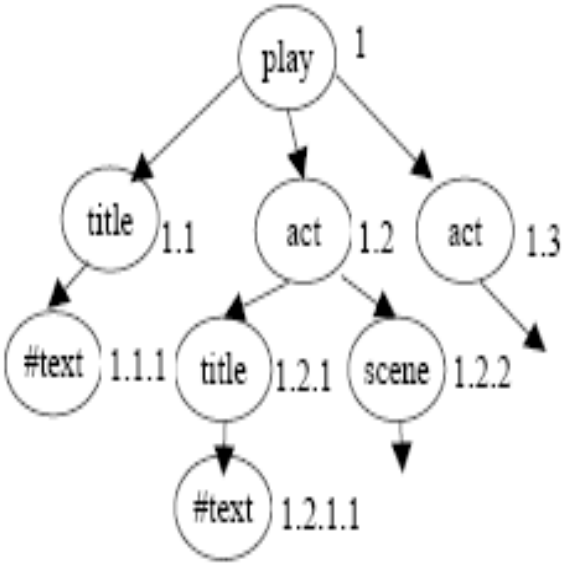
ML N L



Global Order



Local Order



Dewey Order

P L

- G O

  P

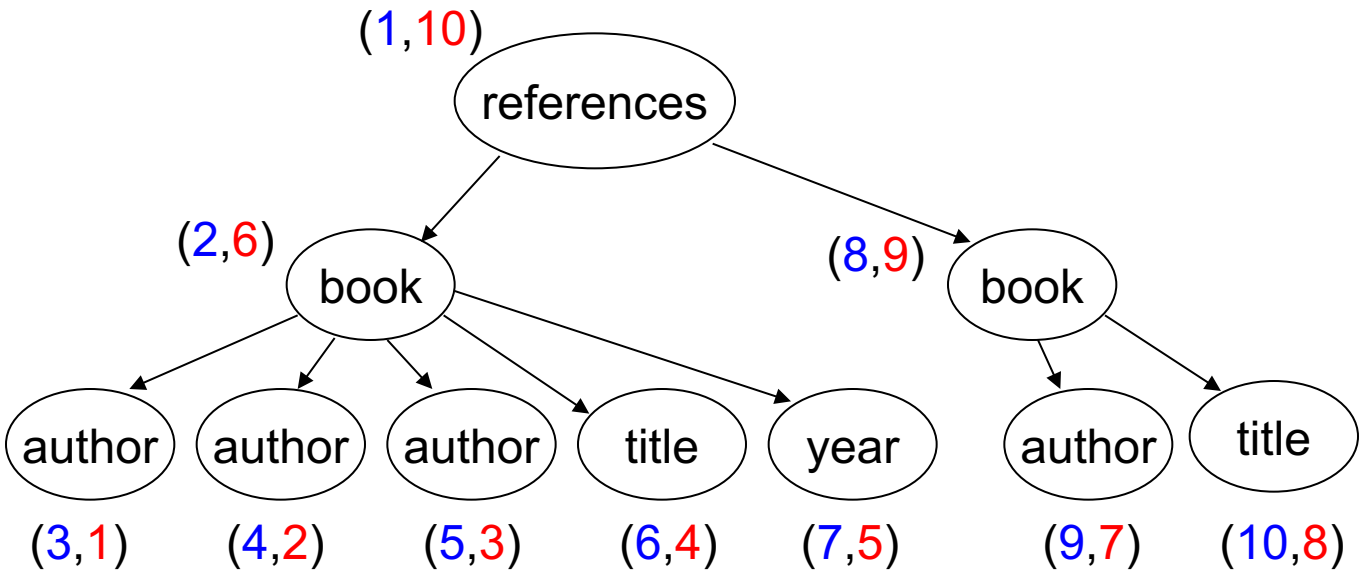
- L O

- D O

N L

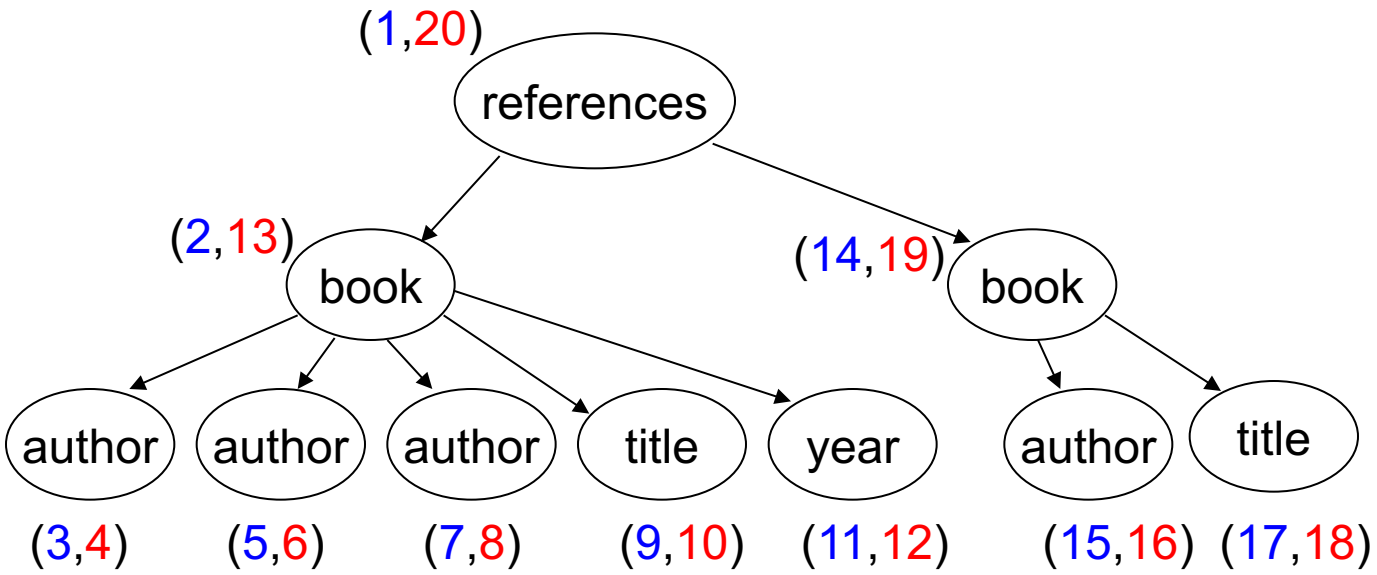
- D

–


(1,10)
references
(2,6)
book
(8,9)
book
author
author
author
title
year
author
title
(3,1)
(4,2)
(5,3)
(6,4)
(7,5)
(9,7)
(10,8)

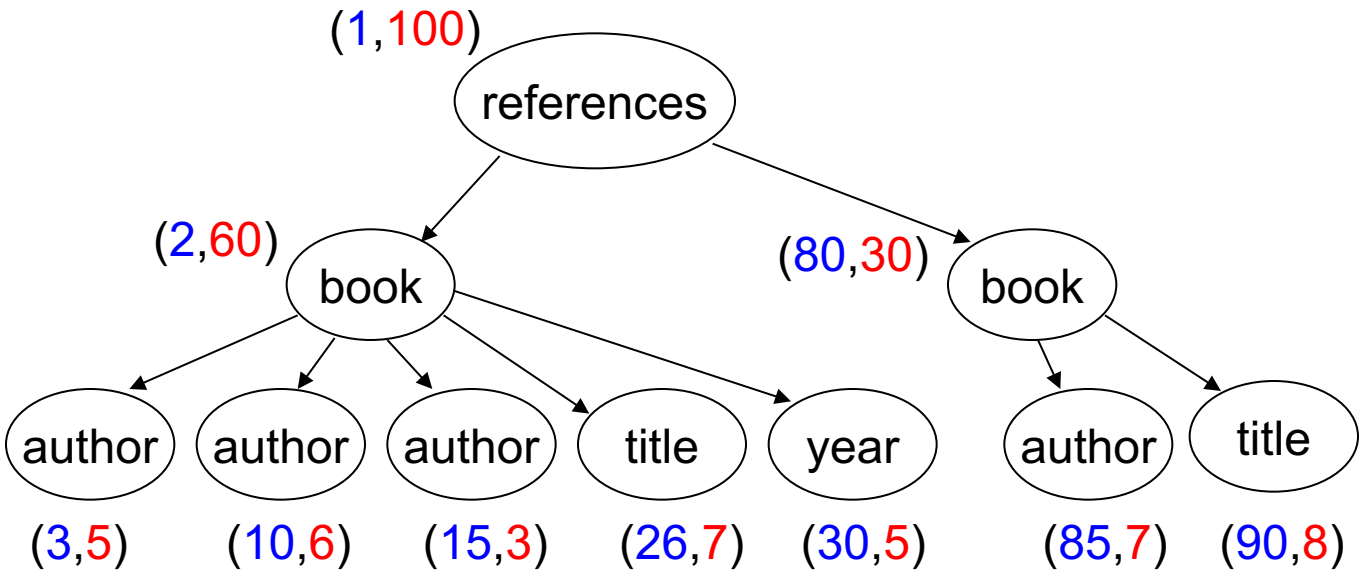
N L

-
  –


(1,20)
references
(2,13)
book
(14,19)
book
author
author
author
title
year
author
title
(3,4)
(5,6)
(7,8)
(9,10)
(11,12)
(15,16)
(17,18)

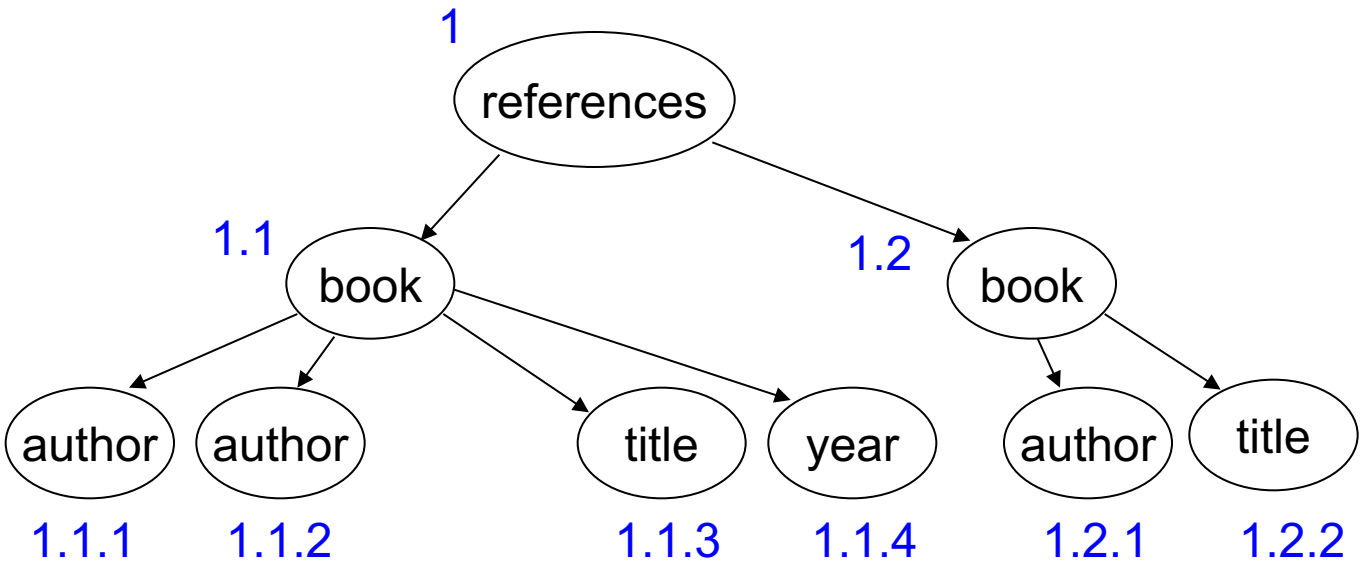
N L

- L
  - –
  - –


(1,100)
references
(2,60)
book
(80,30)
book
author
author
author
title
year
author
title
(3,5)
(10,6)
(15,3)
(26,7)
(30,5)
(85,7)
(90,8)

N L

- D D C


1
references
1.1
book
1.2
book
author
author
title
year
author
title
1.1.1
1.1.2
1.1.3
1.1.4
1.2.1
1.2.2

N L

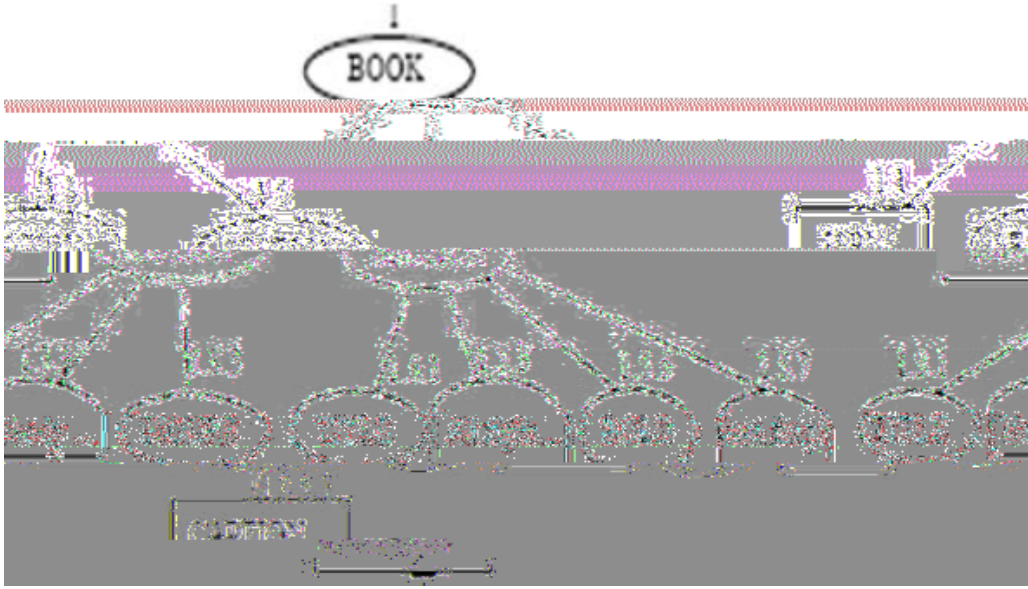
- l

I ORDPA H

- ORDPA H ML
- ORDPA H D O
- B ML
- ORDPA H ORDPA H
- ORDPA H ML
- ORDPA H

E ORDPA H

```
<BOOK ISBN="1-55860-438-3">
    <SECTION>
        <TITLE> Bad Bugs</TITLE>
            Nobody loves bad bugs.
        <FIGURE CAPTION="Sample bug"/>
    </SECTION>
    <SECTION>
        <TITLE> Tree Frogs </TITLE>
            All right-thinking people
            <BOLD> love </BOLD> tree frogs.
    </SECTION>
</BOOK>
```



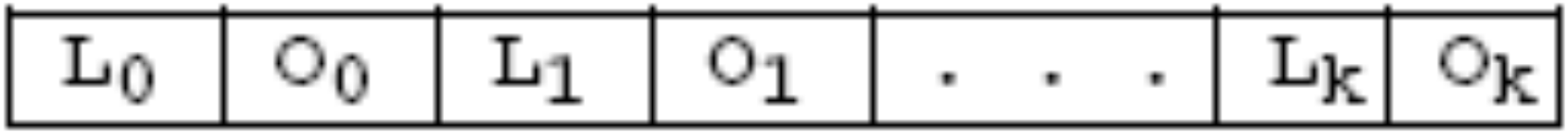
- O
- E
-

ML ORDPATH

ORDPATH	TAG	NODE TYPE	VALUE
1.	1 (BOOK)	1 (Element)	null
1.1	2 (ISBN)	2 (Attribute)	'1-55860-438-3'
1.3	3 (SECTION)	1 (Element)	null
1.3.1	4 (TITLE)	1 (Element)	'Bad Bugs'
1.3.3	--	4 (Value)	'Nobody loves bad bugs.'
1.3.5	5 (FIGURE)	1 (Element)	null
1.3.5.1	6 (CAPTION)	2 (Attribute)	'Sample bug'
1.5	3 (SECTION)	1 (Element)	null
1.5.1	4 (TITLE)	1 (Element)	'Tree frogs'
1.5.3	--	4 (Value)	'All right-thinking people'
1.5.5	7 (BOLD)	1 (Element)	'love '
1.5.7	--	4 (Value)	'tree frogs'

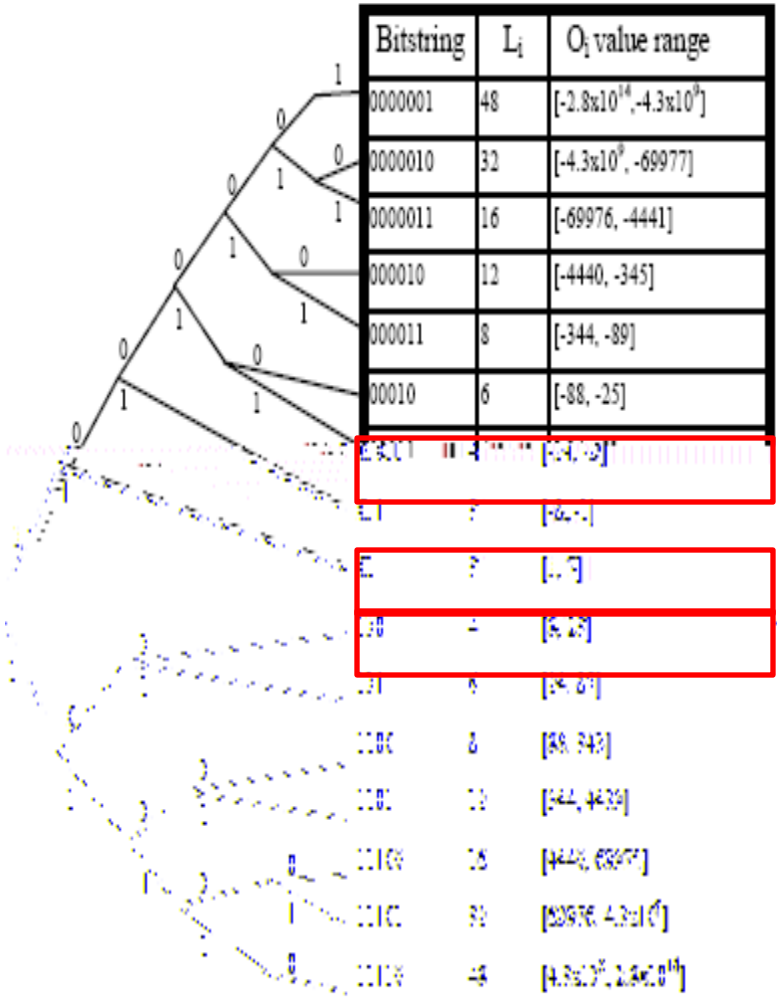
- ML
- A ID

C ORDPA H F



- ORDPA H
  L O
- L O
- O ORDPA H
- L
- ORDPA H

E



Bitstring	L_i	O_i value range
0000001	48	[-2.8x10^{14},-4.3x10^{9}]
0000010	32	[-4.3x10^{9}, -69977]
0000011	16	[-69976, -4441]
000010	12	[-4440, -345]
000011	8	[-344, -89]
00010	6	[-88, -25]

A table of Li with Oi value range

- L

OR DPA H

01 001	01 101	01 011	00011 1111	100 0011
L_0=3 O_0=1	L_1=3 O_1=5	L_2=3 O_2=3	L_3=4 O_3=-9	L_4=4 O_4=11

- I

- ML

ORDPATH

Bitstring	L_i	O_i value range
[illegible]	[illegible]	[illegible]

(a)

Bitstring	L_i	O_i value range
000000001	20	[-1118485, -69910]
00000001	16	[-69909, -4374]
0000001	12	[-4373, -278]
000001	8	[-277, -22]
00001	4	[-21, -6]
0001	2	[-5, -2]
001	1	[-1, 0]
01	0	[1, 1]
10	1	[2, 3]
110	2	[4, 7]
1110	4	[8, 23]
11110	8	[24, 279]
111110	12	[280, 4375]
1111110	16	[4376, 69911]
11111110	20	[69912, 1118487]

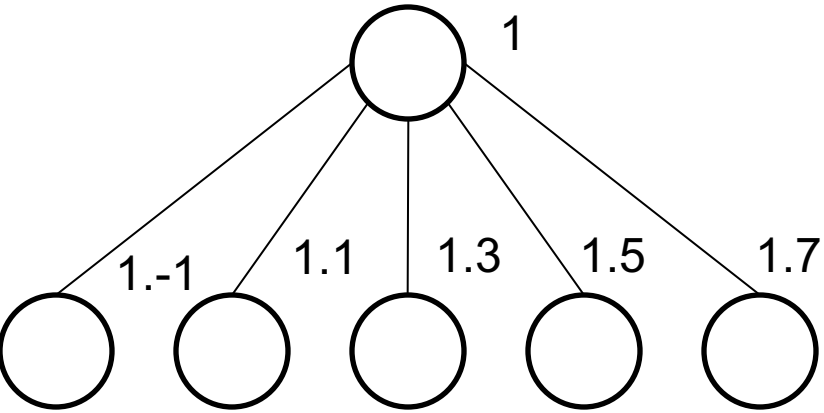
(b)

- D

ML

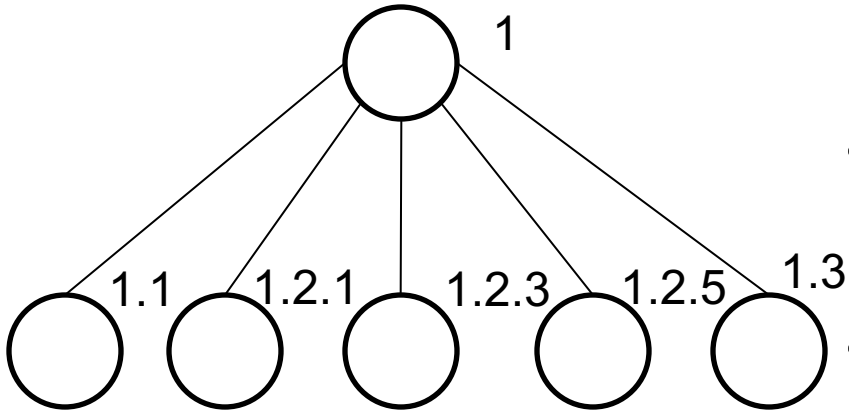
L

N I ORDPA H



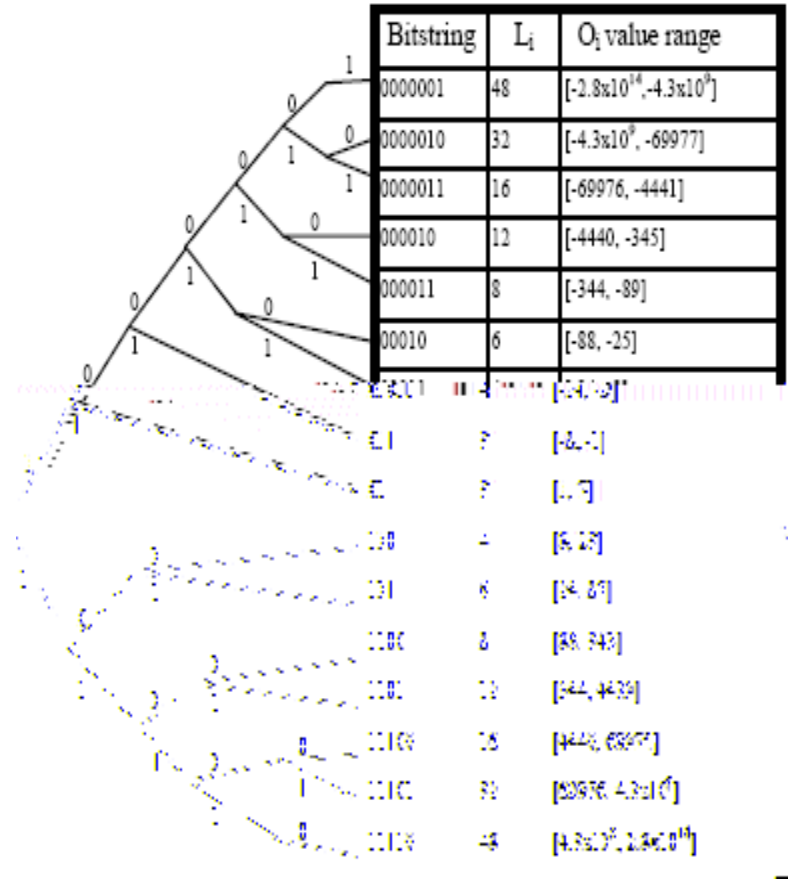
-
-
-
-

caret in

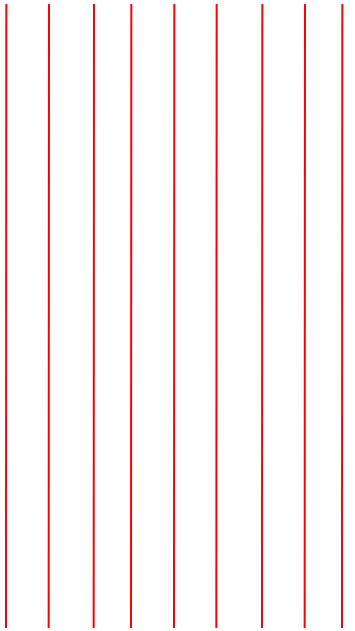


- E
- M

ORDPATH C



Bitstring	L_i	O_i value range
0000001	48	[-2.8×10^{14}, -4.3×10^{9}]
0000010	32	[-4.3×10^{9}, -69977]
0000011	16	[-69976, -4441]
000010	12	[-4440, -345]
000011	8	[-344, -89]
00010	6	[-88, -25]



-
-
-
-
-

1) Simply compare the bitstring

2) 0 padding & bit-bit comparison

I ORDPA H

- ORDPA H
  - ML ORDPA H
  - P
  - E A
  - A
-
  - AG
  - AL E
  - LE EL

- ORDPATH ML
- ORDPATH
- L

- D P F D M ACM
  C M
- C O
  IGMOD
- L Q L B M I ML
  LDB
- I ML
  IGMOD
- ON P O N ORDPA H ML
  IGMOD
- JKC H J A K A C IMBER N
  ML LDB J
- C C C E
  ML EDB
- MB L M D B P ML F

J

Relationship established through simple comparisons:

D		

J

A P E ML

Q P M

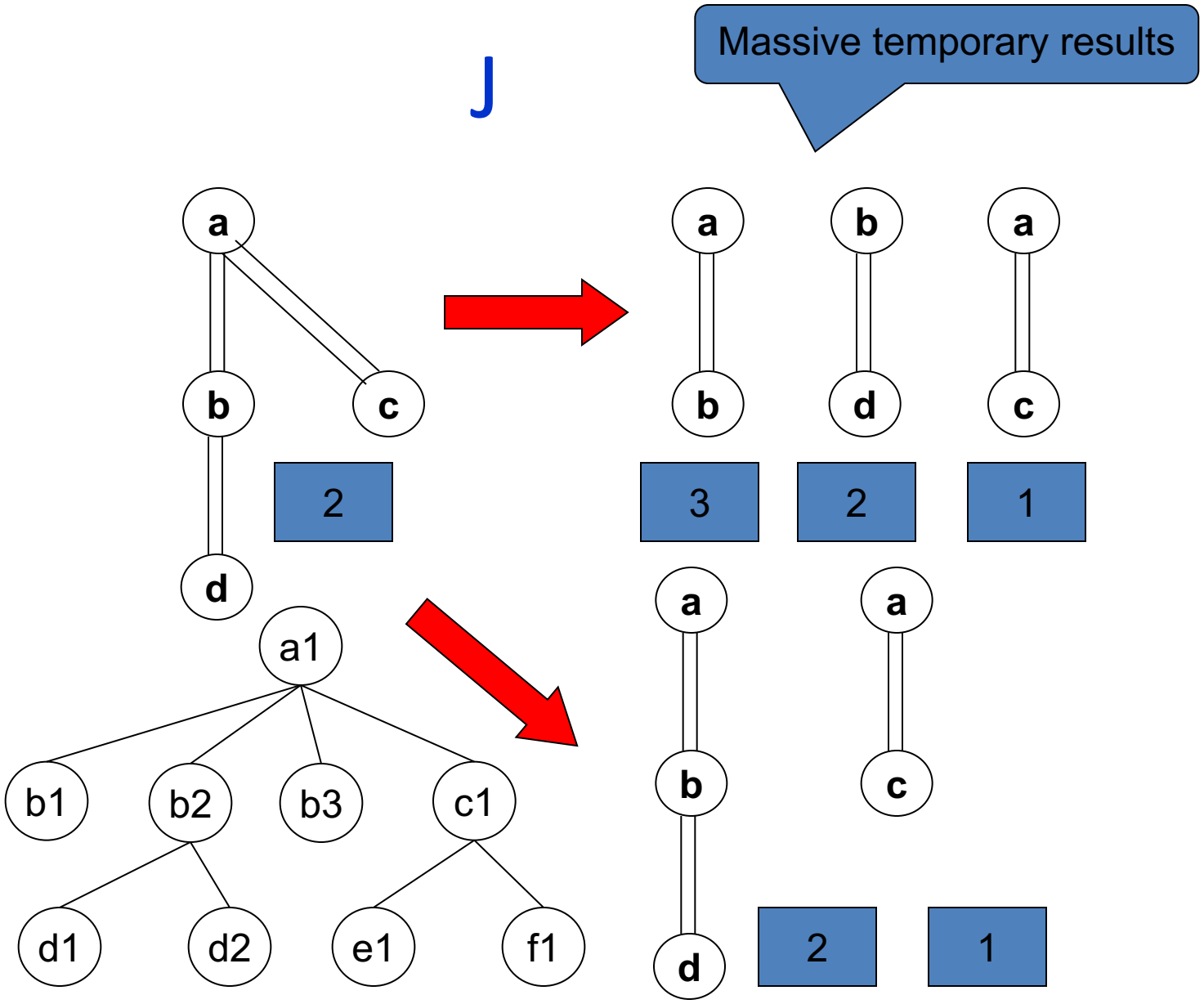
A K ICDE

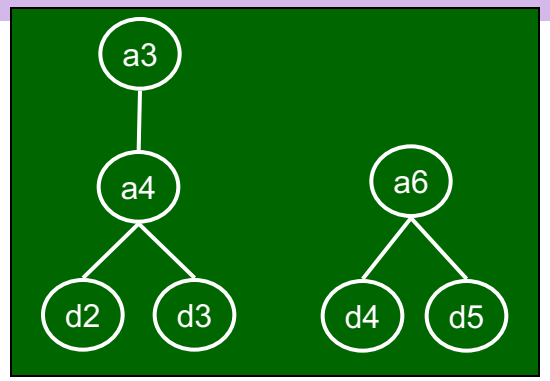
J A

-
  - A
  - B
- O
  - P
  -
-
  - *With* *without*
  - O *ancestor* *descendant*

M J A

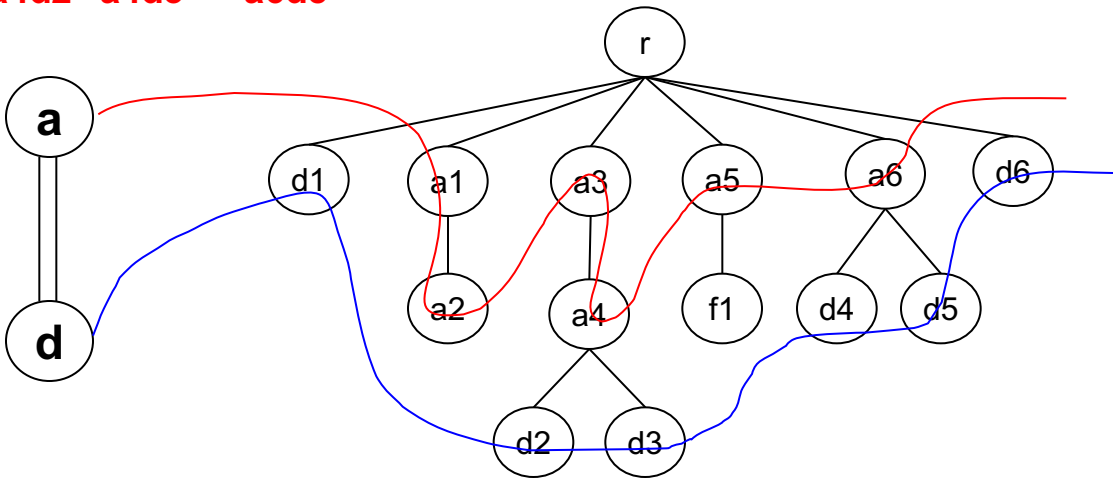
- N
  - E MPMGJN
-

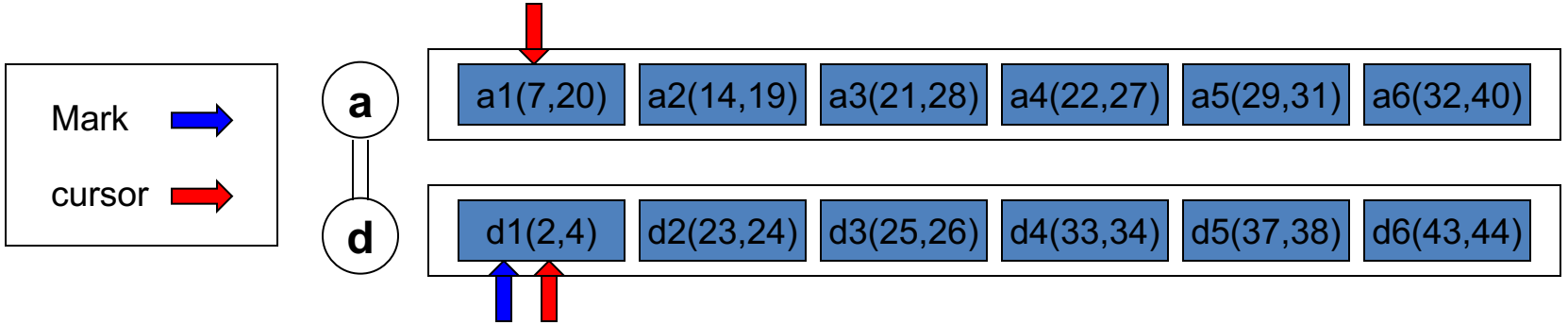

Massive temporary results
J
a
b
c
d
2
a
b
3
b
d
2
a
c
1
a1
b1
b2
b3
c1
d1
d2
e1
f1
a
b
d
2
a
c
1



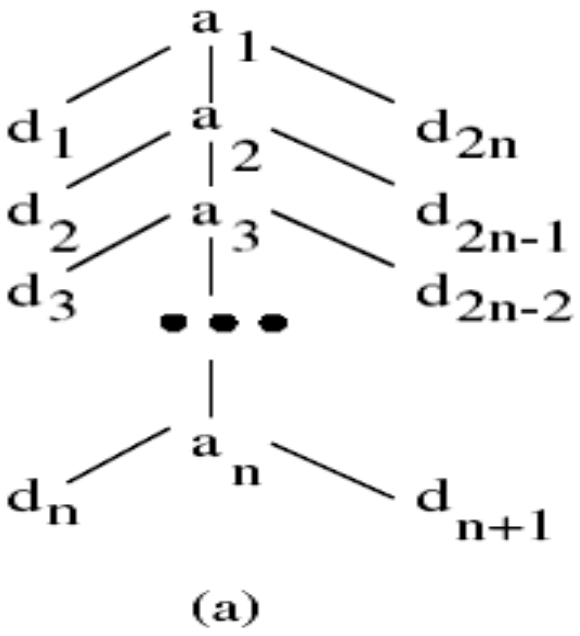
a3d2 a3d3 a6d4

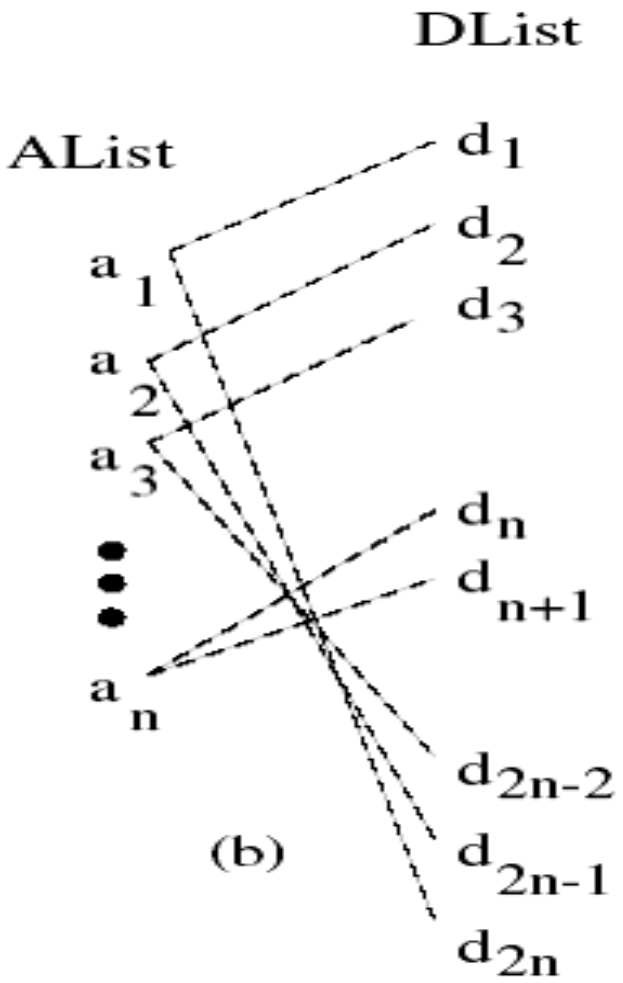
a4d2 a4d3 a6d5



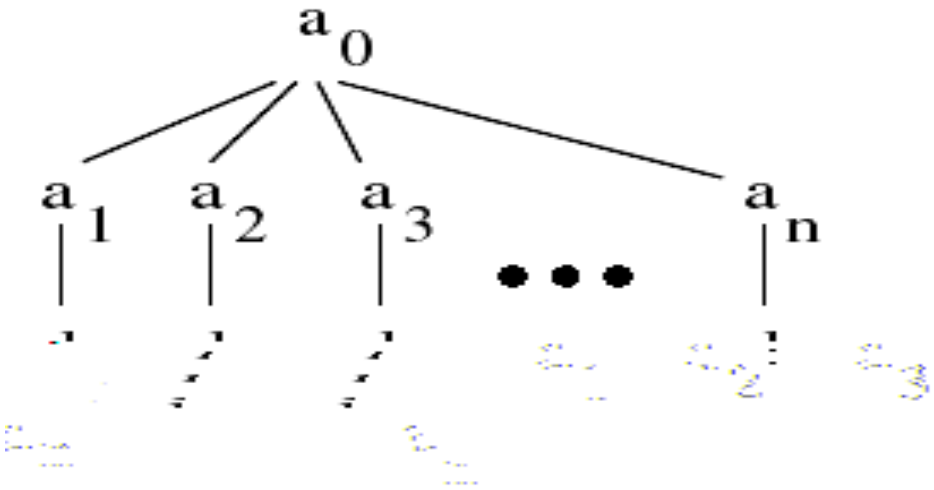


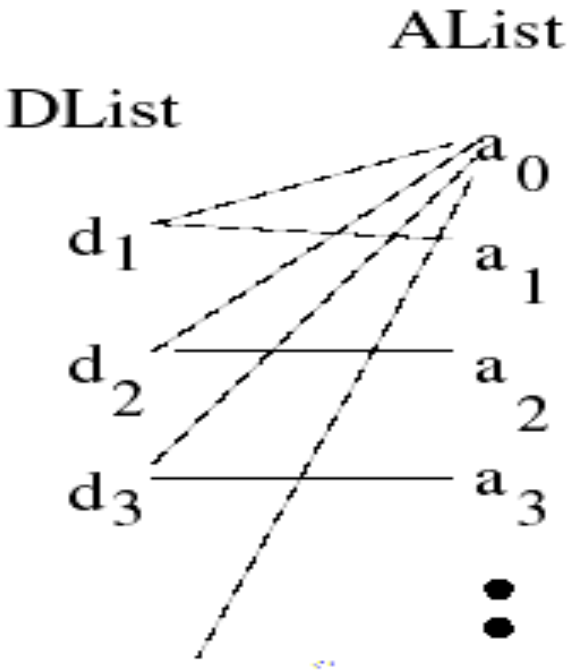
C M A


a 1
a 2
a 3
a n
d 1
d 2
d 3
d n
d 2n
d 2n-1
d 2n-2
d n+1
(a)


DList
AList
a 1
a 2
a 3
a n
d 1
d 2
d 3
d n
d n+1
d 2n-2
d 2n-1
d 2n
(b)

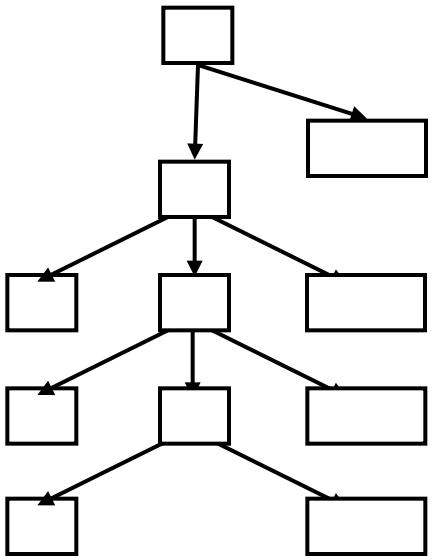
C M D



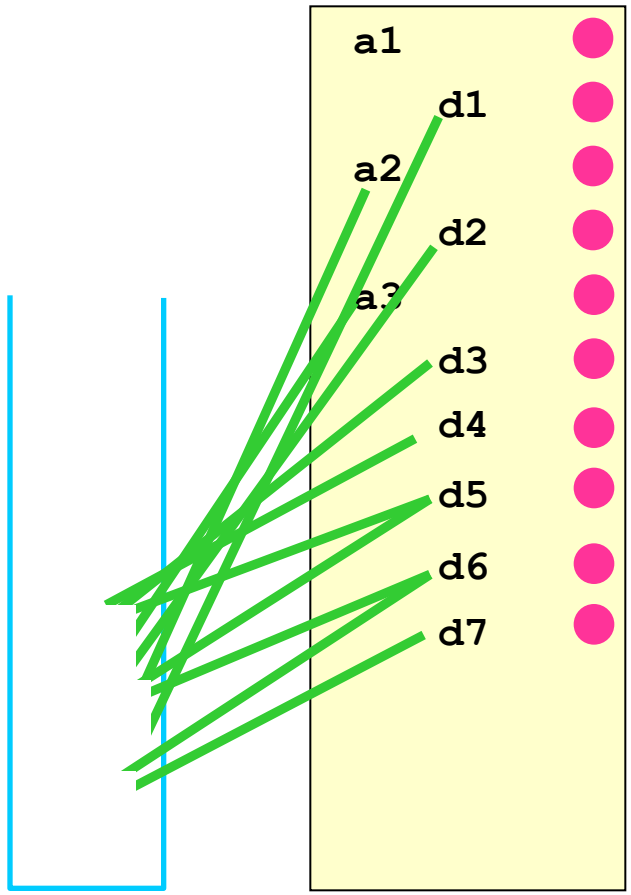


J A

- B ML
  - *Linear*
  - A
  -
- M
  AL DL



D



S ack:

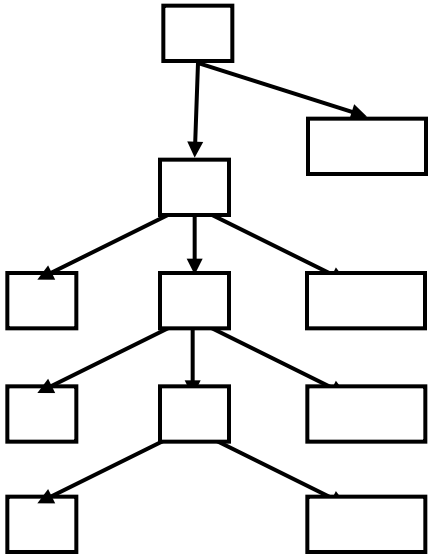
O :

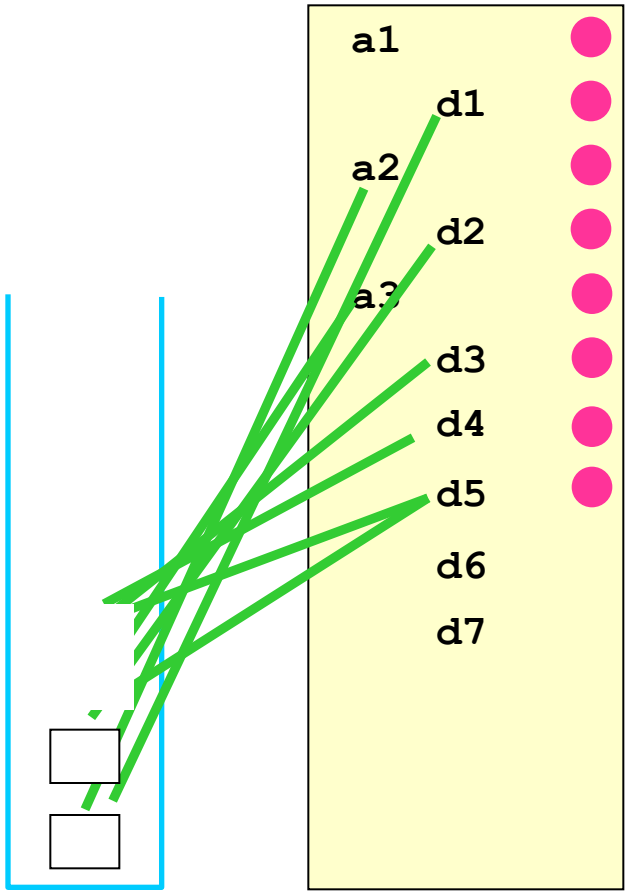
(a1,d1)	(a1,d2),(a2,d2)	(a1,d3),(a2,d3),(a3,d3)
(a1,d4),(a2,d4),(a3,d4)	(a1,d5),(a2,d5)	(a1,d6)

A

- G R
- B I I
-
- E
  - P
  - P

A





(a3,d3),(a3,d4)

(a2,d2),(a2,d3),(a2,d4)

(a1,d1),(a1,d2),(a1,d3),(a1,d4)

I J

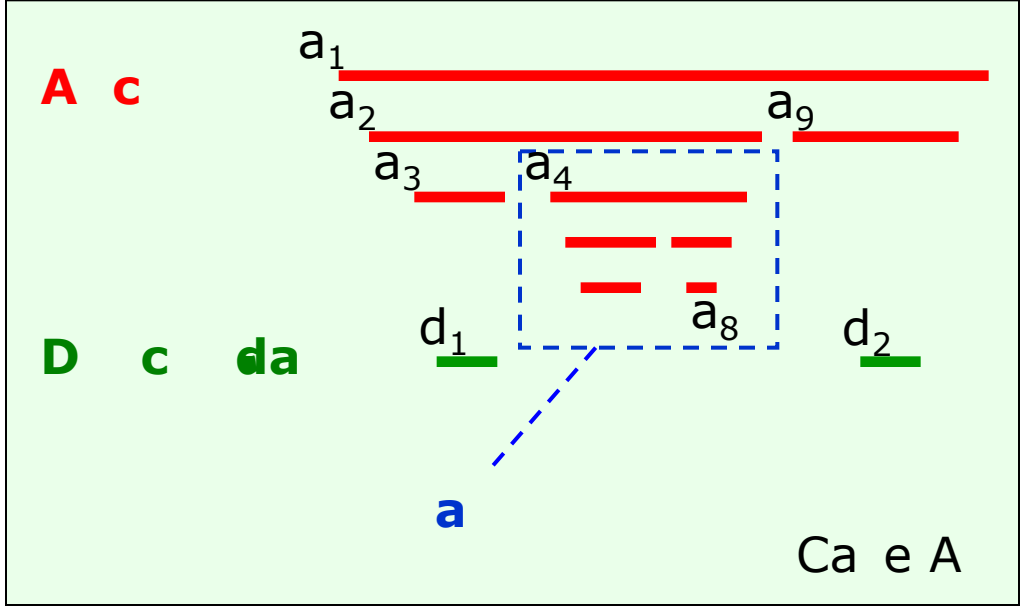
- B

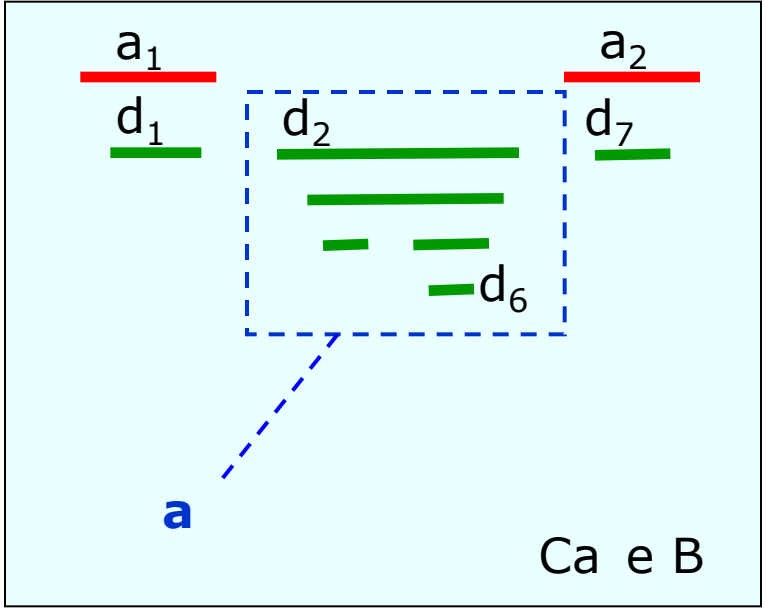
  – C *Efficient Structural Joins on Indexed XML Documents* LDB

- ML

  – H J *XR-Tree: Indexing XML Data for Efficient Structural Joins* ICDE

  –

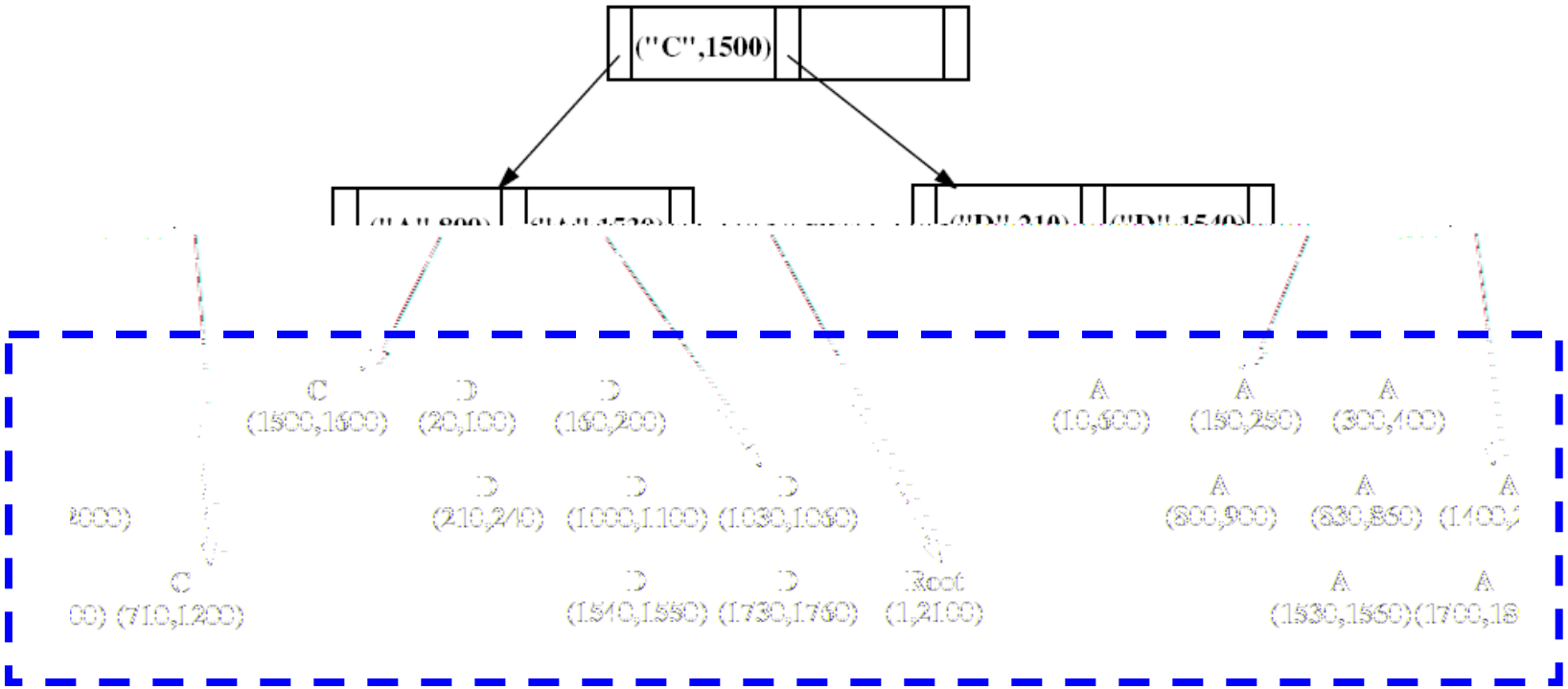


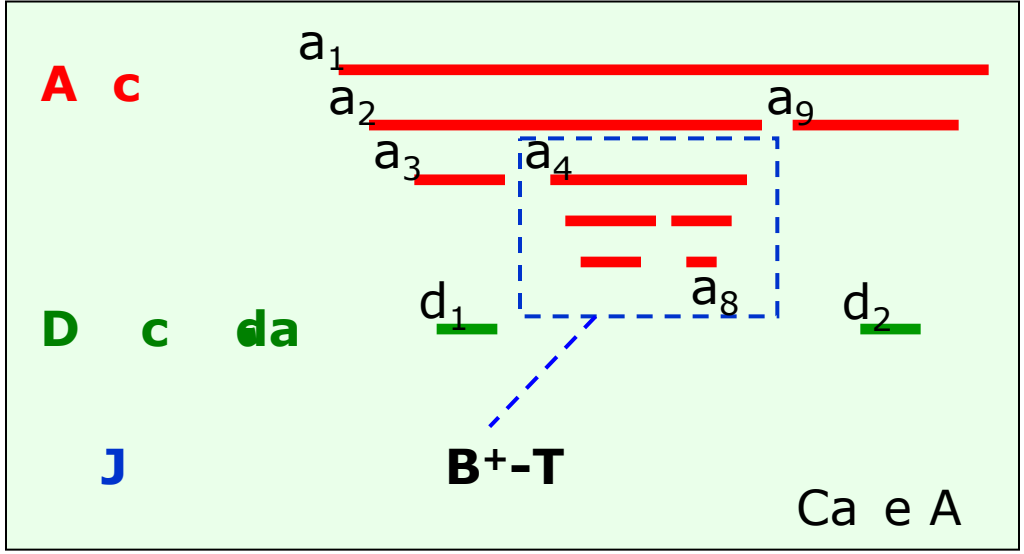


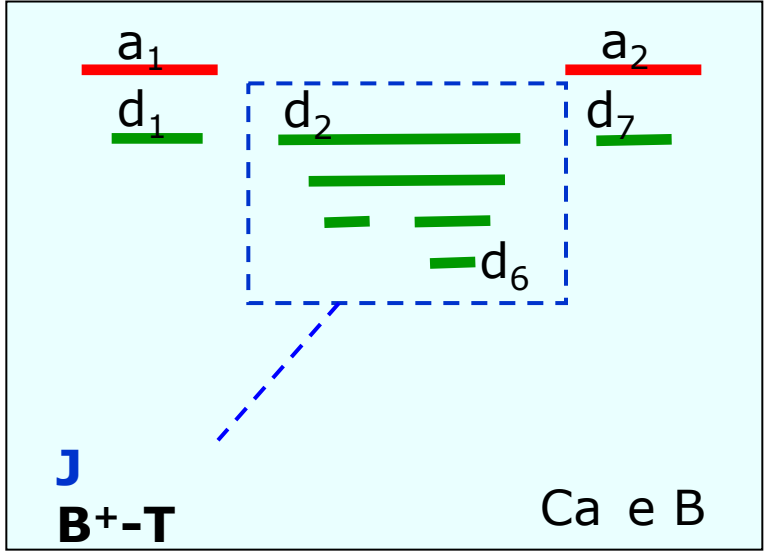
- C A
  - P
  - P
  - A
    L
  - P
- C B
  - C A
  - A

J B

- C
- M .


("C",1500)
C
(1500,1600)
C
(20,100)
C
(160,200)
C
(210,270)
C
(1000,1100)
C
(1030,1060)
C
(710,1200)
C
(1540,1550)
C
(1730,1760)
Root
(1,2100)
A
(10,500)
A
(150,250)
A
(300,400)
A
(800,900)
A
(830,850)
A
(1530,1550)
A
(1700,18





I

P A

O

E

P L

L A

E

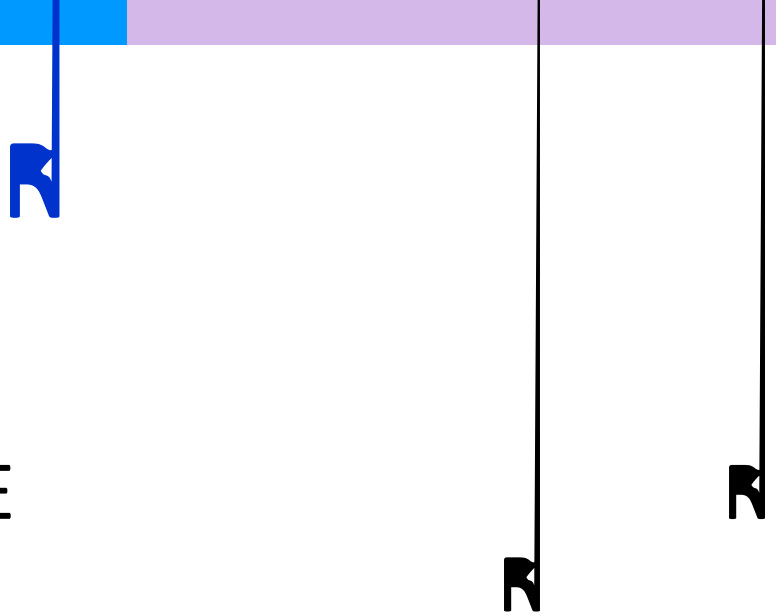
O

I

L D

E

L D

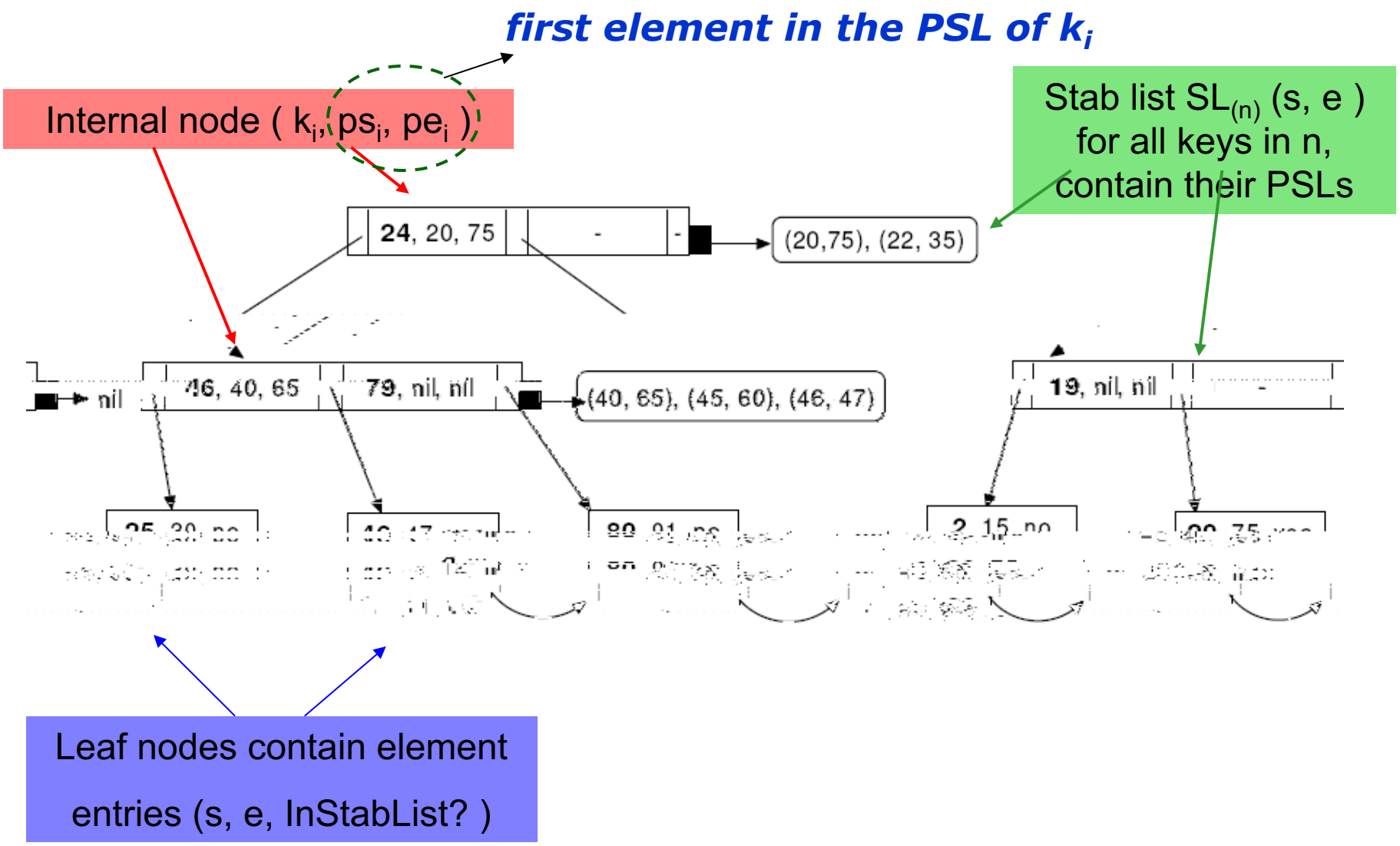


- B B
- G E
  E

  –

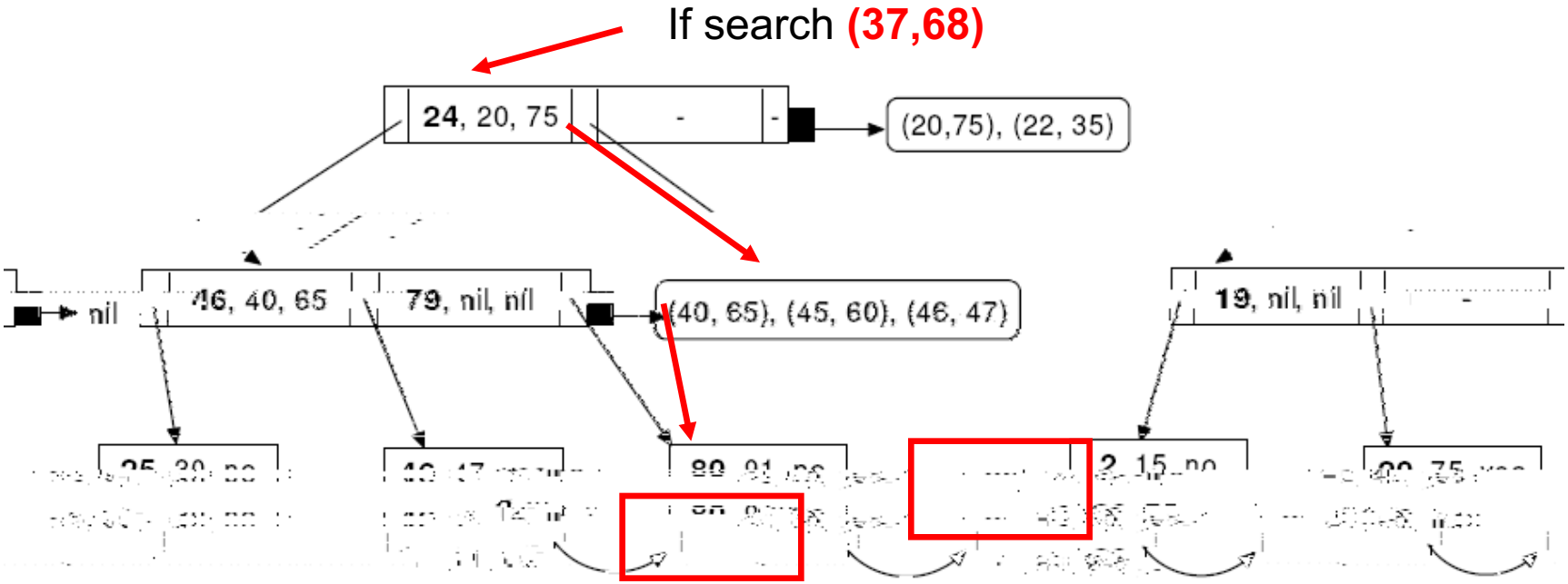
- L
  – G *k* *E(s,e)* *k* *E* *s*
    k *e* *k primarily stabs E,* *k*
    E

A E R


first element in the PSL of k_i
Internal node (k_i, ps_i, pe_i)
Stab list $SL_{(n)}$ (s, e) for all keys in n, contain their PSLs
24, 20, 75
(20,75), (22, 35)
nil
46, 40, 65
79, nil, nil
(40, 65), (45, 60), (46, 47)
19, nil, nil
Leaf nodes contain element entries (s, e, InStabList?)

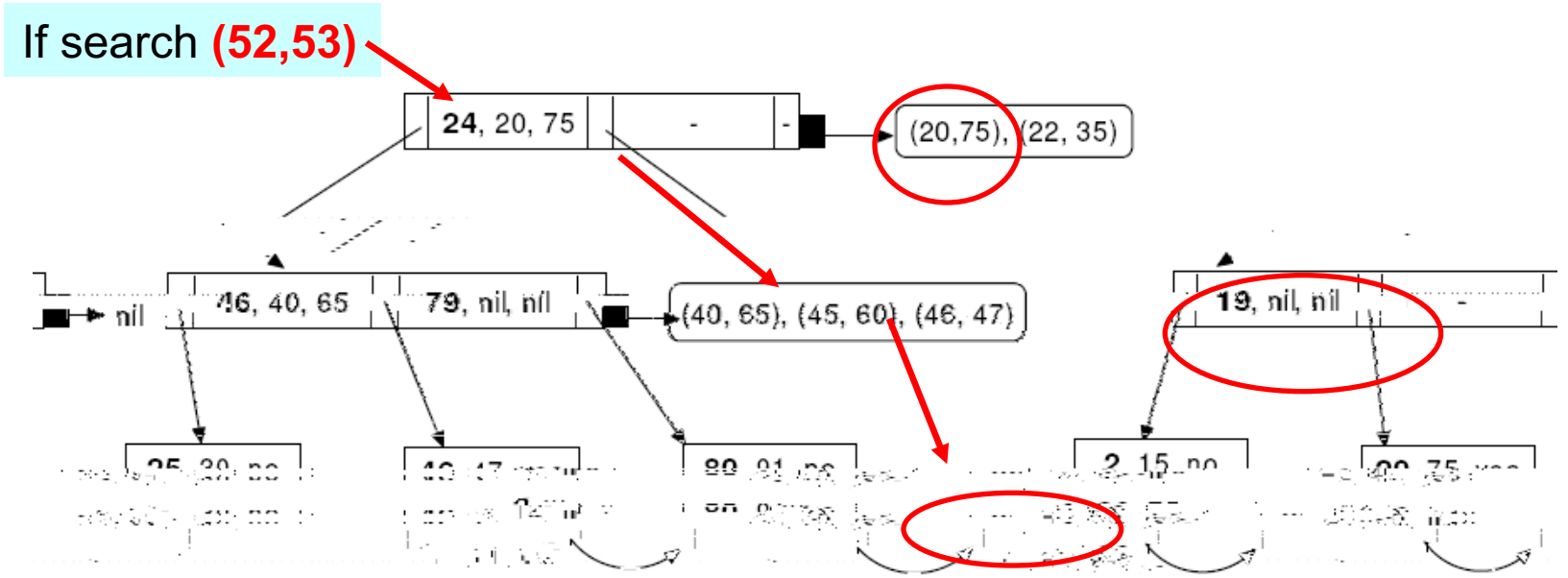
D

- G E
  - B


If search (37,68)
24, 20, 75
(20,75), (22, 35)
nil
46, 40, 65
79, nil, nil
(40, 65), (45, 60), (46, 47)
19, nil, nil

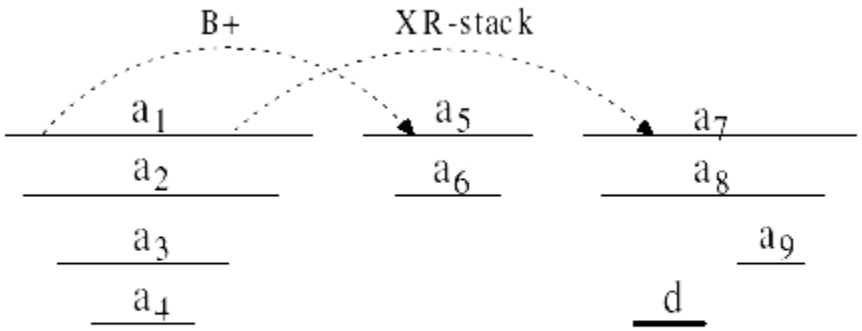
A

- D
  - I P L
  - F C
  - I P L

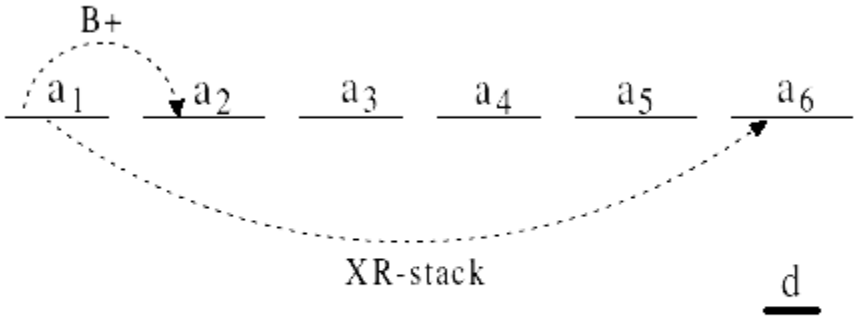

If search (52,53)
24, 20, 75
(20,75), (22, 35)
nil
46, 40, 65
79, nil, nil
(40, 65), (45, 60), (46, 47)
19, nil, nil

J R

- A A D
- B R
- M J



(a) Highly nested



(b) Less nested

P Q

- B I
- J
  - J P H J J O
    ML Q O ICDE
- H
  - N B N K D H J
    O ML P M IGMOD

H J

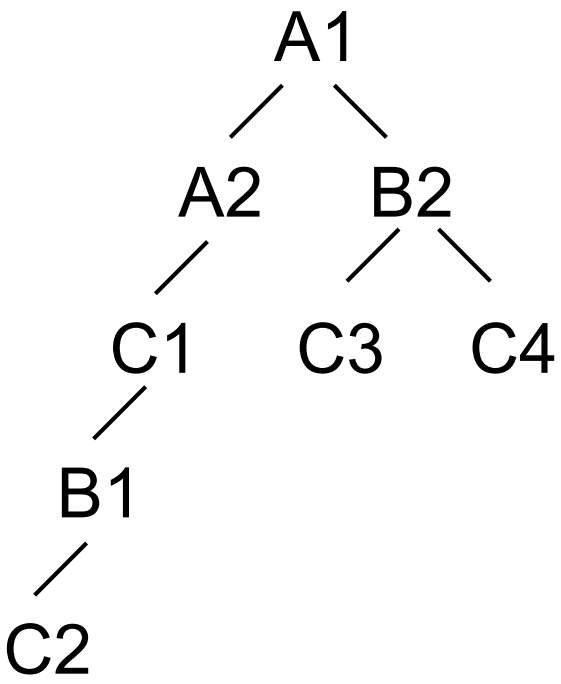
- P
- M
- C
  - P
  - E B

P

- H q q qn
- I T_q T_q T_{qn} S_q S_q S_{qn}
- T_{qn}
  - L T_{qmin} *begin*
  - C *end* *head* T_{qmin} *begin*
  - P *head* T_{qmin} S_{qmin} *top* $S_{parent\ q}$
  - I q qn S_{qmin}

- C
  - E
  - E

P



E

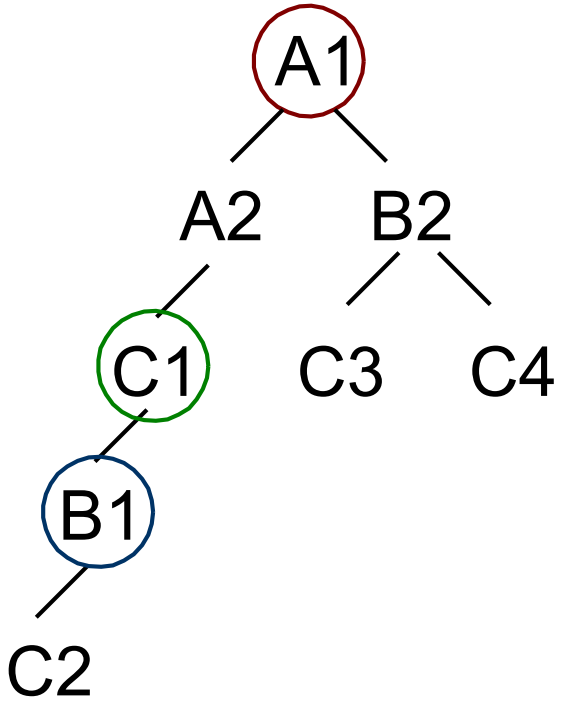
S

A

B

C

P



E

S

A

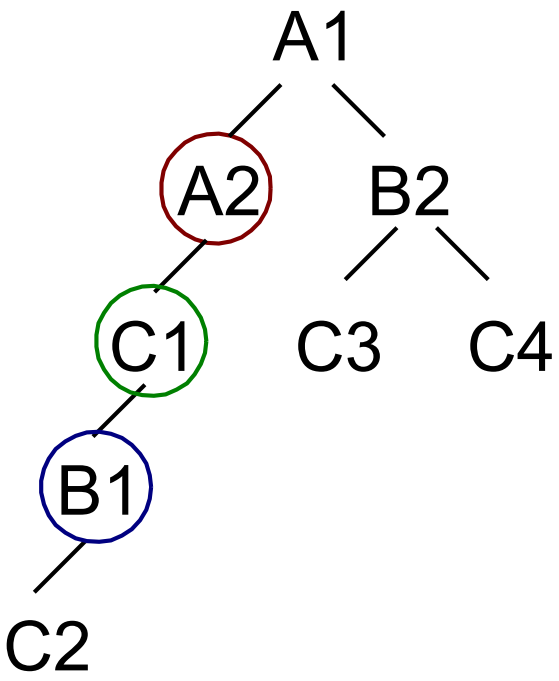
||

B

||

C

P



E

S

A1

A

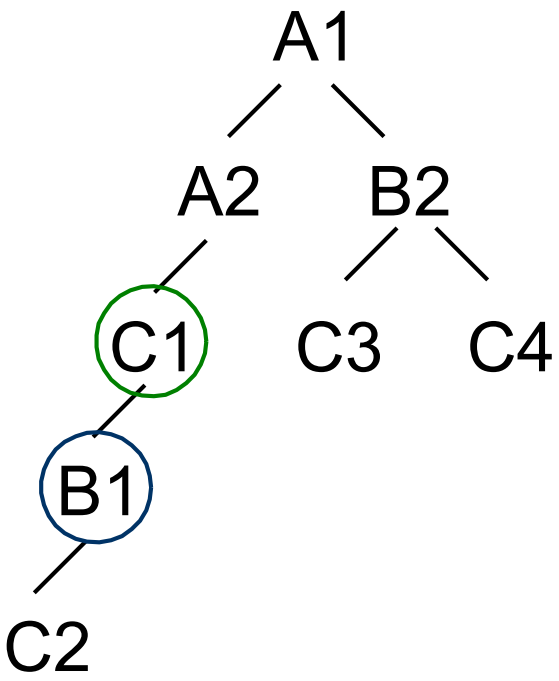
||

B

||

C

P



E

S

A1 - A2

A

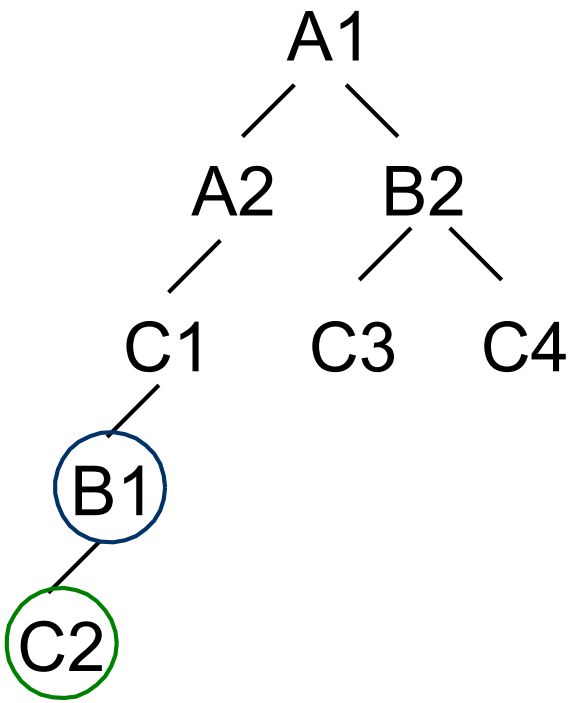
||

B

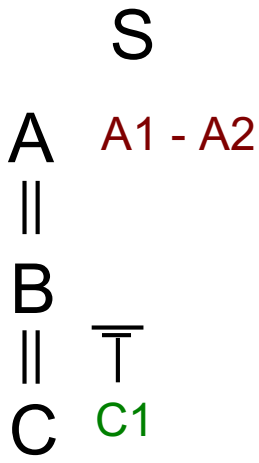
||

C

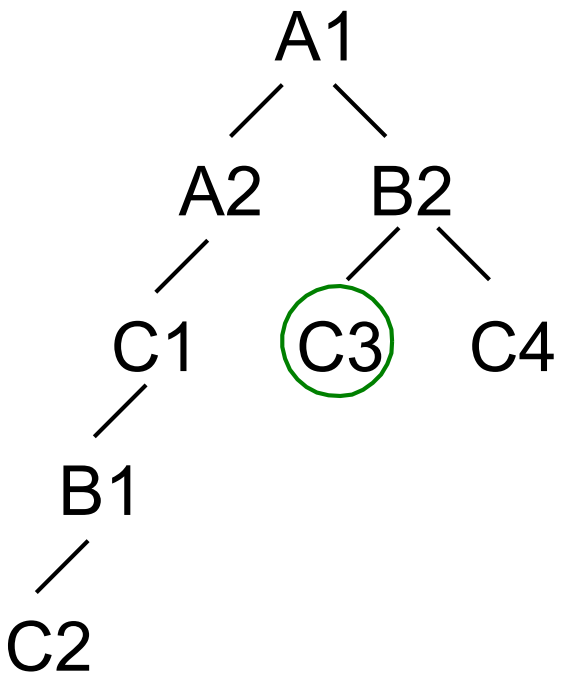
P



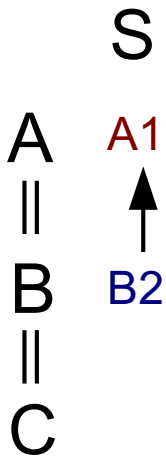
E



P

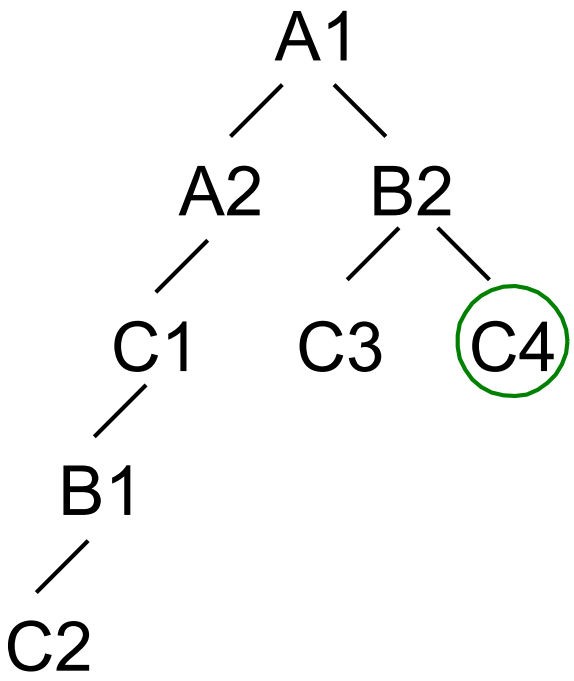


E

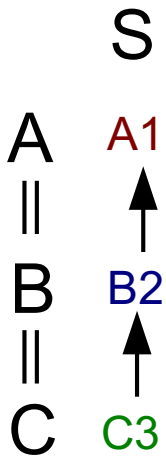


A1,B1,C2
A2,B1,C2

P

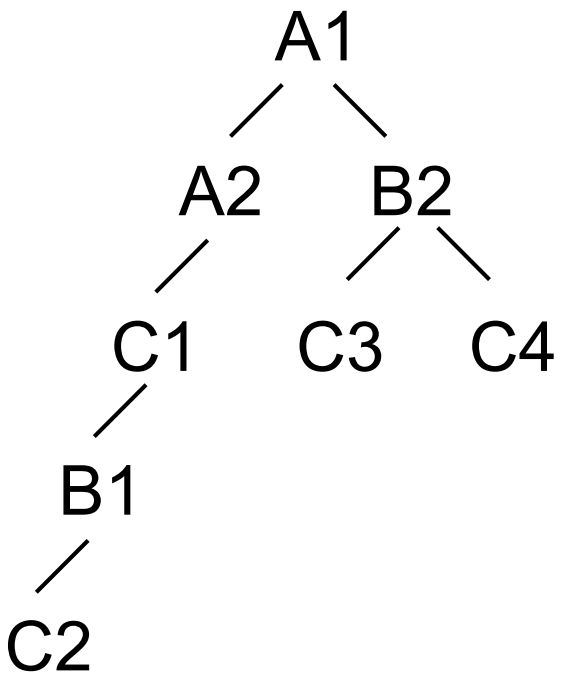


E

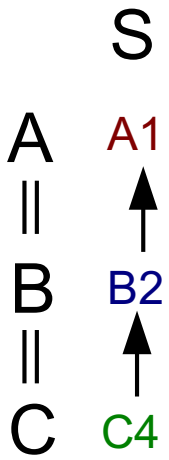


A1,B1,C2
A2,B1,C2
A1,B2,C3

P



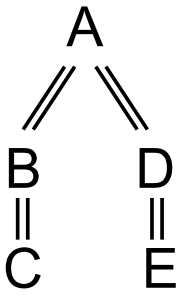
E

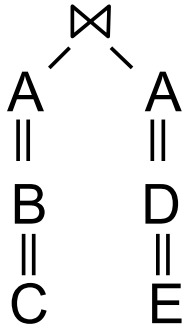


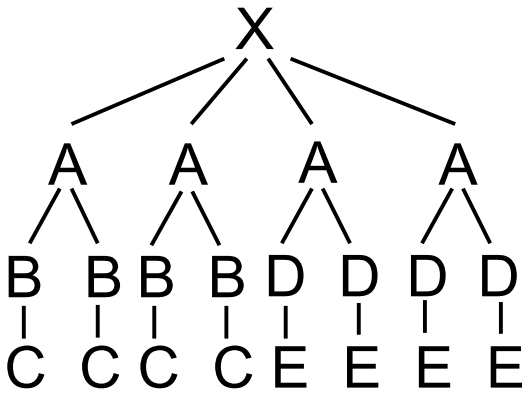
A1,B1,C2
A2,B1,C2
A1,B2,C3
A1,B2,C4

Q

- N P
  - –
  - – M
- P M







- C

  e_q

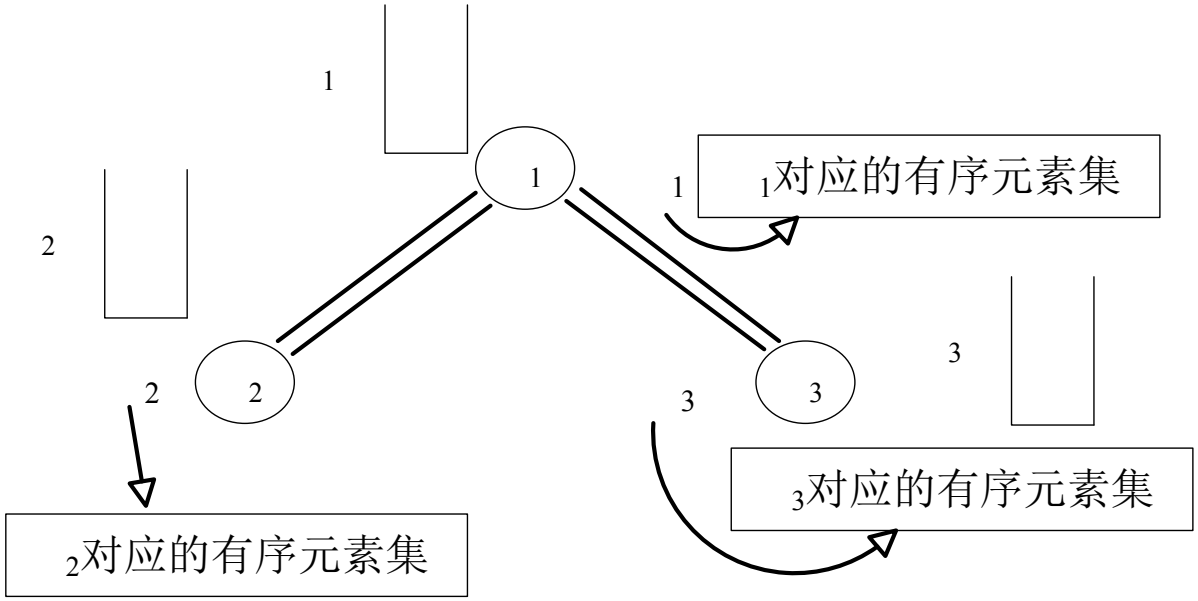
  S_q

  – e_q $e_{q'}$ $T_{q'}$ q'

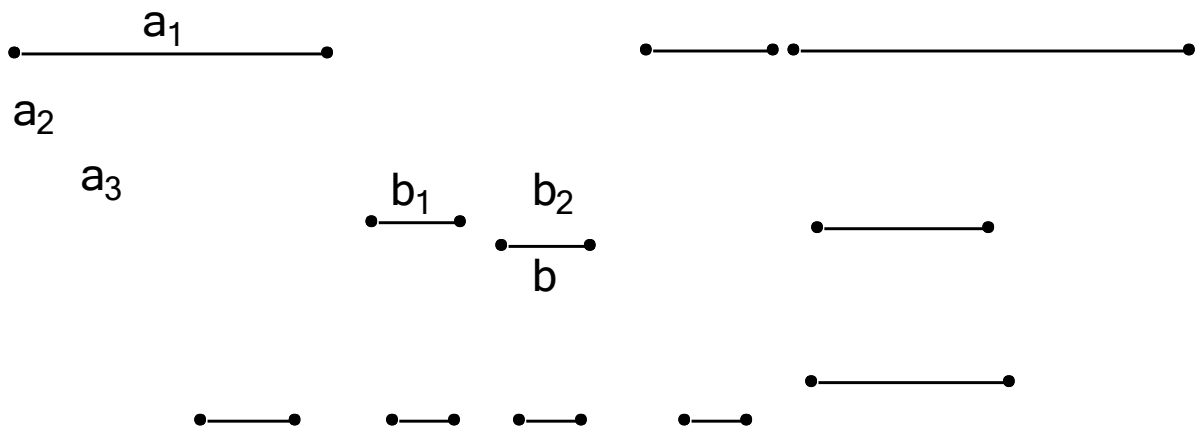
  q

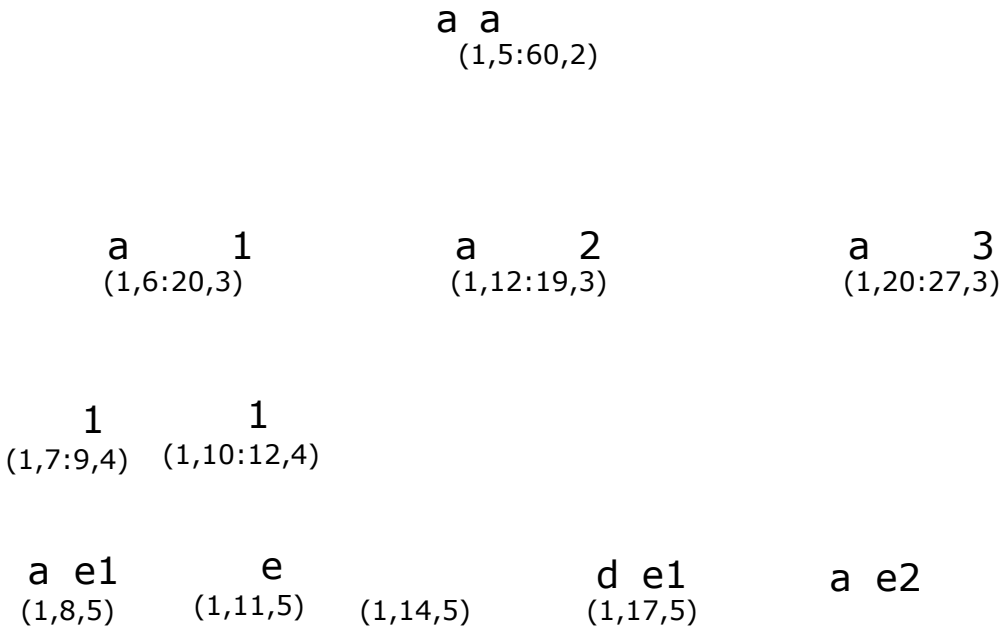
  – E $e_{q'}$

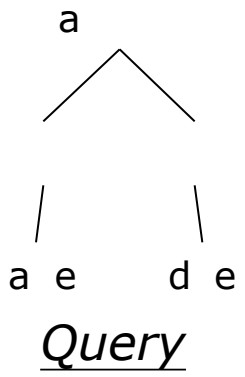
- M



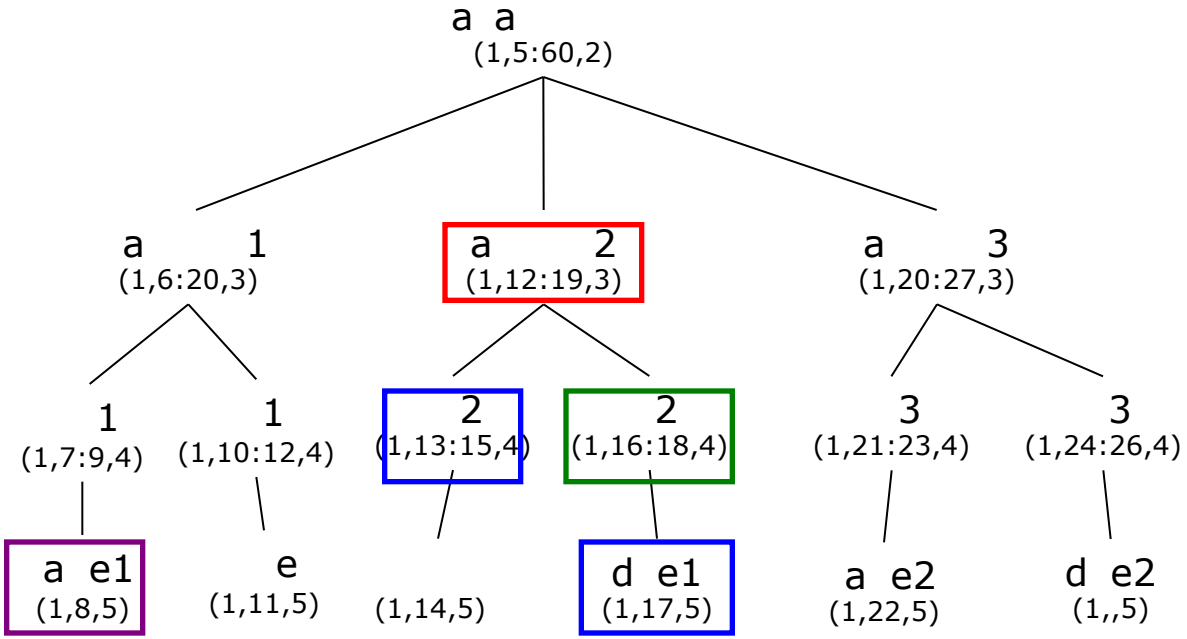
TwigStack


a_1
a_2
a_3
b_1
b_2
b

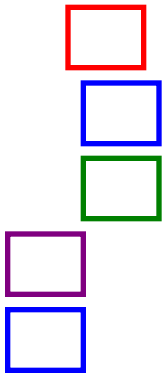

a a
(1,5:60,2)
a 1
(1,6:20,3)
a 2
(1,12:19,3)
a 3
(1,20:27,3)
1
(1,7:9,4)
1
(1,10:12,4)
a e1
(1,8,5)
e
(1,11,5)
(1,14,5)
d e1
(1,17,5)
a e2

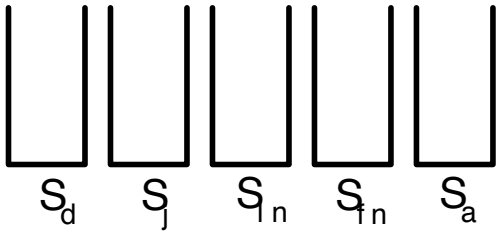


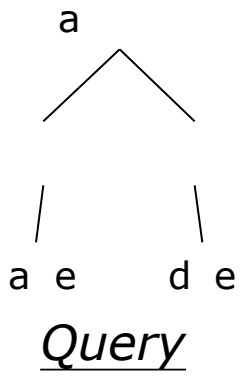
Query



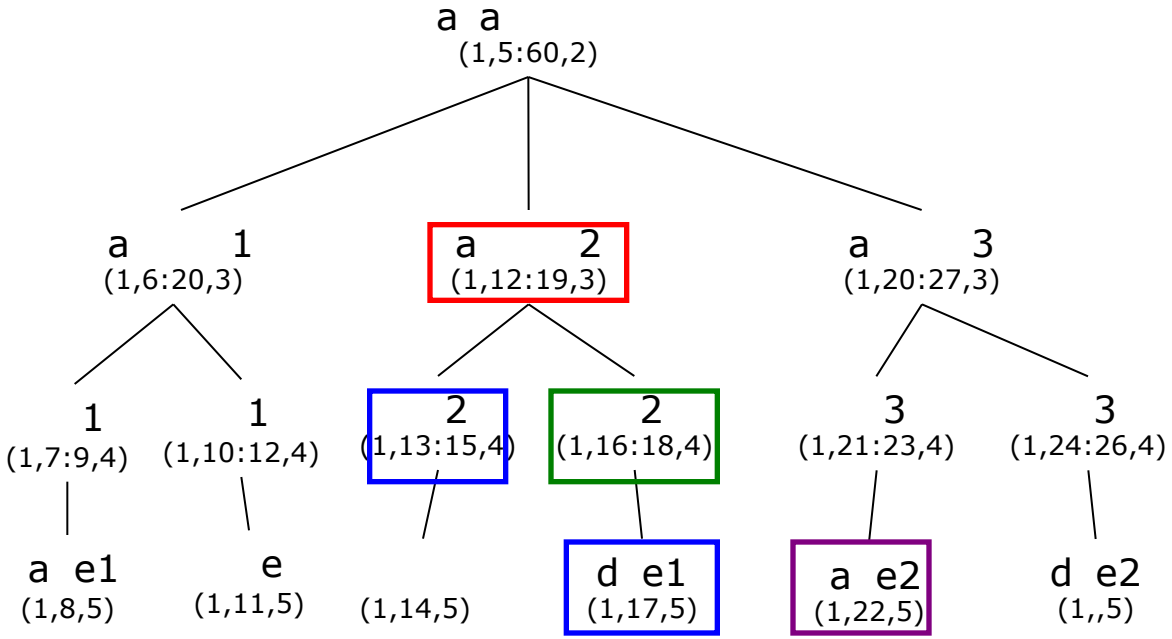
Document



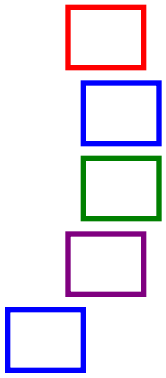


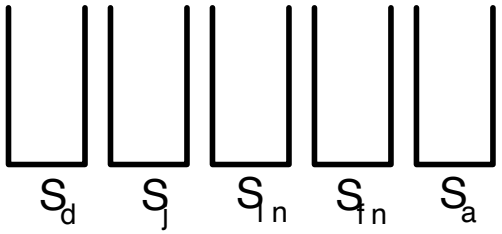


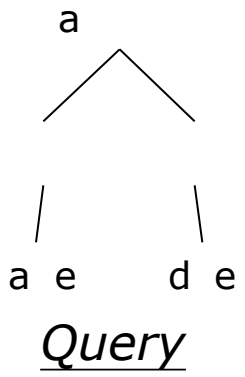
Query



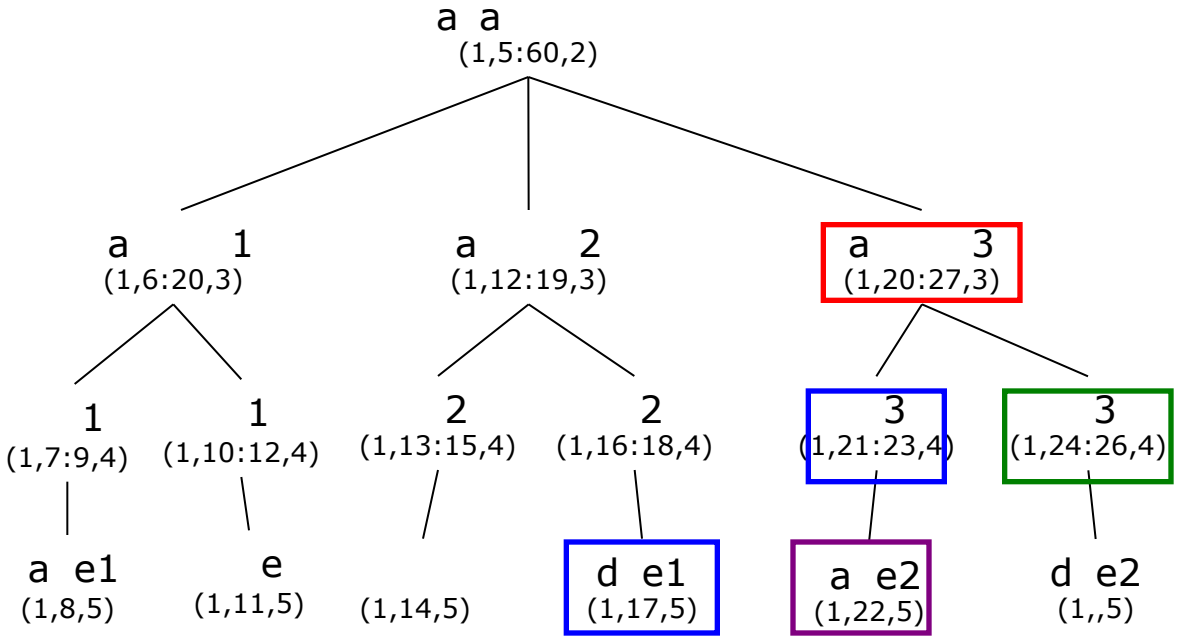
Document



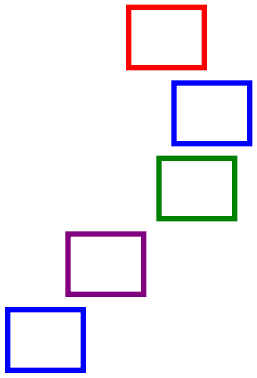


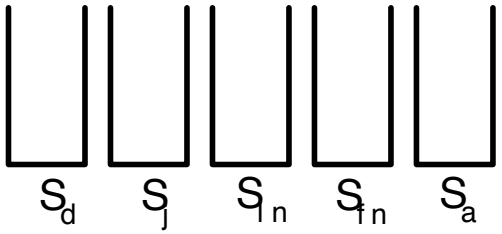


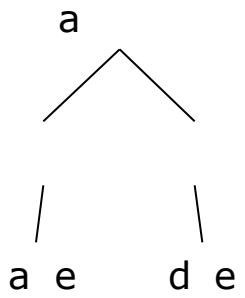
Query



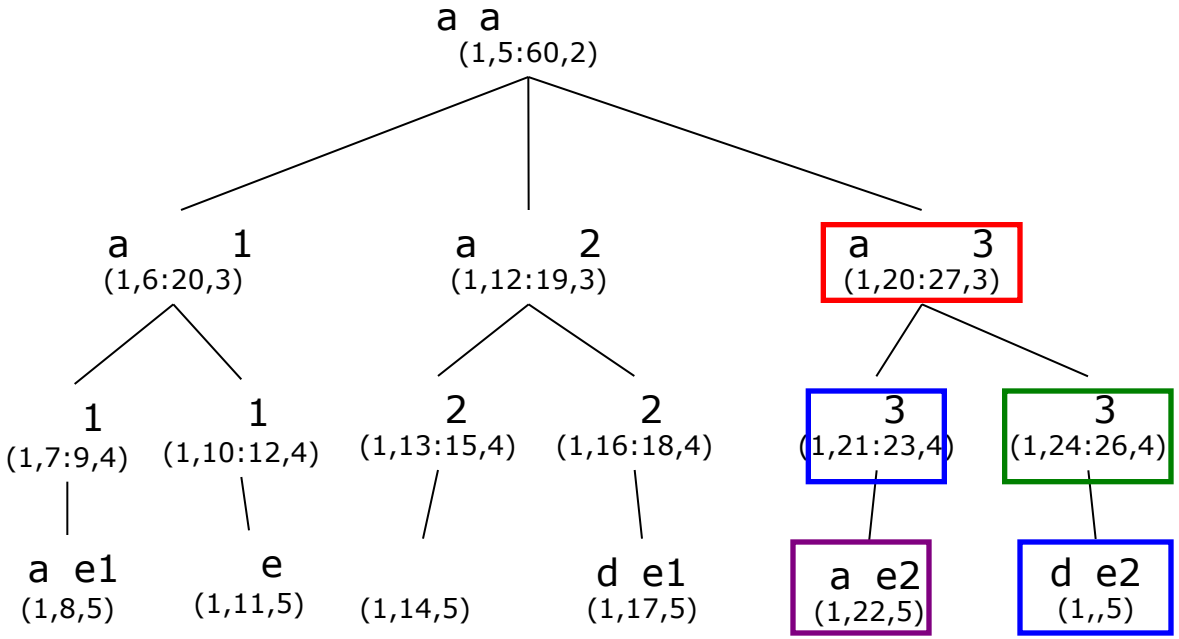
Document



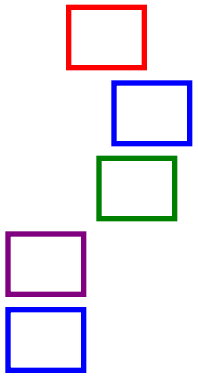


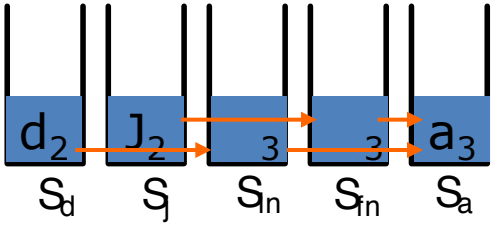


Query



Document





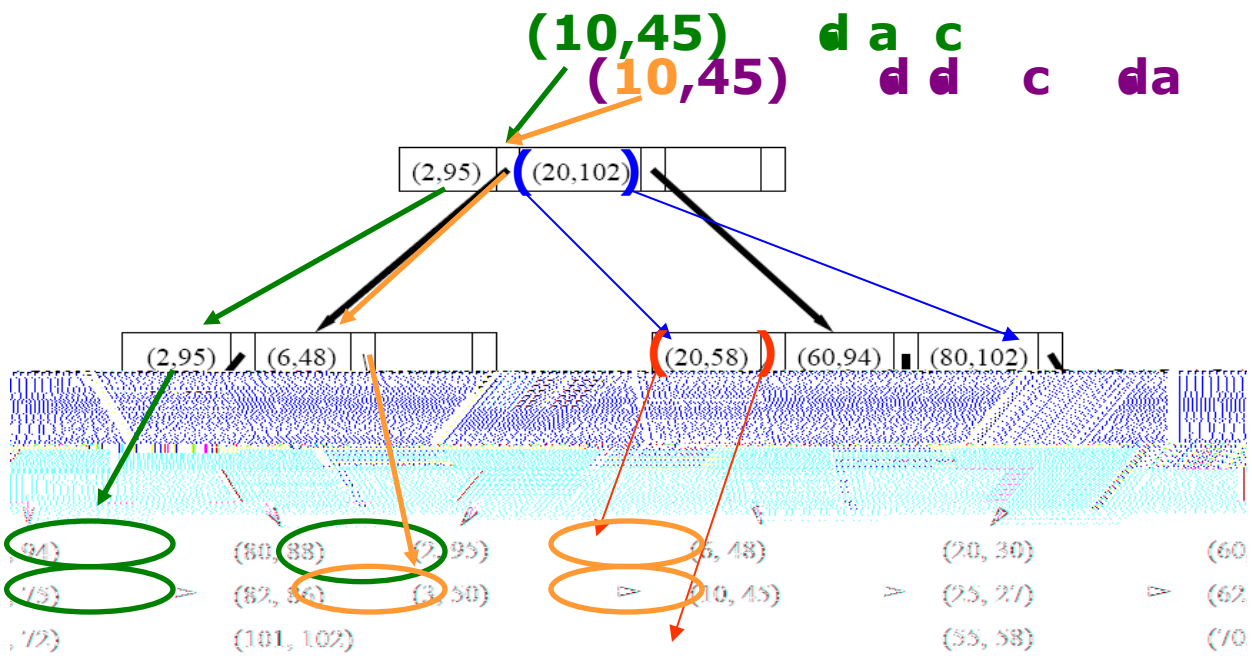
Pa 1: a3- 3- 2

Pa 2: a3- 3-d2

Me e (2, 3, d2, 3, a3)

B

- B B
  - P
- B



M J A

- G J LDB
  - I
- P P C DE A
  - E
- L L CIKM
  - P
- J C IGMOD
  - E L P P
- JF L LDB
  - E D
- C LDB
  - H