Purely Relational XQuery

Database-Supported XML Processors

Torsten Grust
Technische Universität München

http://www-db.in.tum.de/~grust/

LUG Erding, Oct 2007

A Database Guy's View of XML, XPath, and XQuery

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable.**

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable**.
- Wait! Sizable data volume...? Use Relational Databases!

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable.**
- Wait! Sizable data volume...? Use Relational Databases!

```
<?xml version="1.0"?>
<root>
  <elem att="val">
    <!-- comment -->
    text
  </elem>
</root>
```

```
let $doc := doc("in.xml")
for $t in $doc//text()
return
  count($t/../comment())
```

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable.**
- Wait! Sizable data volume...? Use Relational Databases!

```
<?xml version="1.0"?>
<root>
  <elem att="val">
    <!-- comment -->
    text
  </elem>
</root>
```

```
let $doc := doc("in.xml")
for $t in $doc//text()
return
  count($t/../comment())
```

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable.**
- Wait! Sizable data volume...? Use Relational Databases!

pre	size	level
1	1	10
2	1	20
3	1	30
⋮	⋮	⋮

```
let $doc := doc("in.xml")
for $t in $doc//text()
return
  count($t/../comment())
```

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable.**
- Wait! Sizable data volume...? Use Relational Databases!

pre	size	level
1	1	10
2	1	20
3	1	30
⋮	⋮	⋮

```
SELECT    pre,size,
          ROW_NUMBER (…)
FROM      t000
WHERE     value
GROUP BY  iter,…
```

A Database Guy's View of XML, XPath, and XQuery

- Build XML processors that do not stumble once XML **input volumes become sizable.**
- Wait! Sizable data volume...? Use Relational Databases!

OFF-THE-SHELF

pre	size	level
1	1	10
2	1	20
3	1	30
⋮	⋮	⋮

```
SELECT   pre,size,
         ROW_NUMBER (…)
FROM     t000
WHERE    value
GROUP BY iter,…
```

RDBMSs as Processors for Non-Relational Languages

- Relational databases contain the best understood and most scalable query processors available today.
- Can we use relational query engines *as-is* to efficiently process non-relational languages?

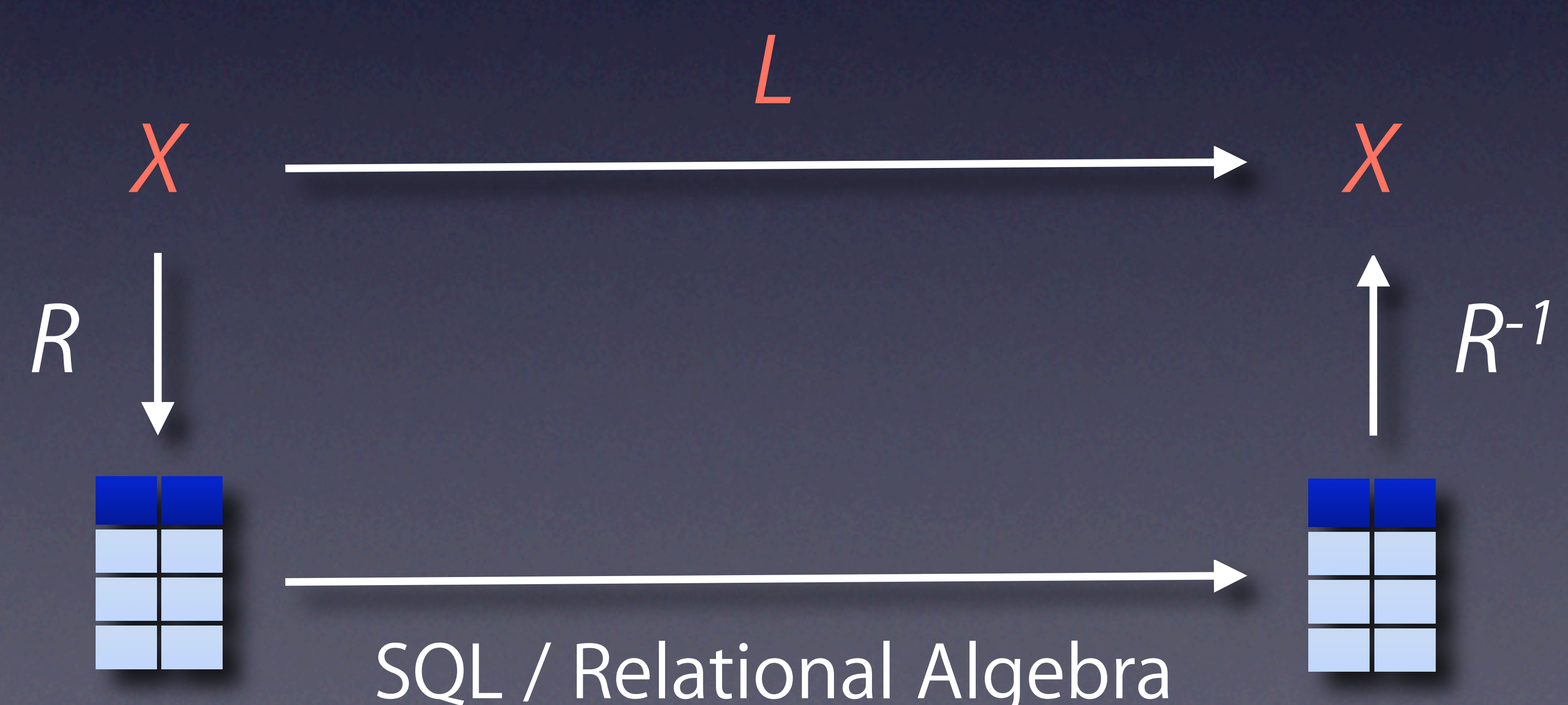
RDBMSs as Processors for Non-Relational Languages

- Relational databases contain the best understood and most scalable query processors available today.
- Can we use relational query engines *as-is* to efficiently process non-relational languages?

$$X \xrightarrow{L} X$$

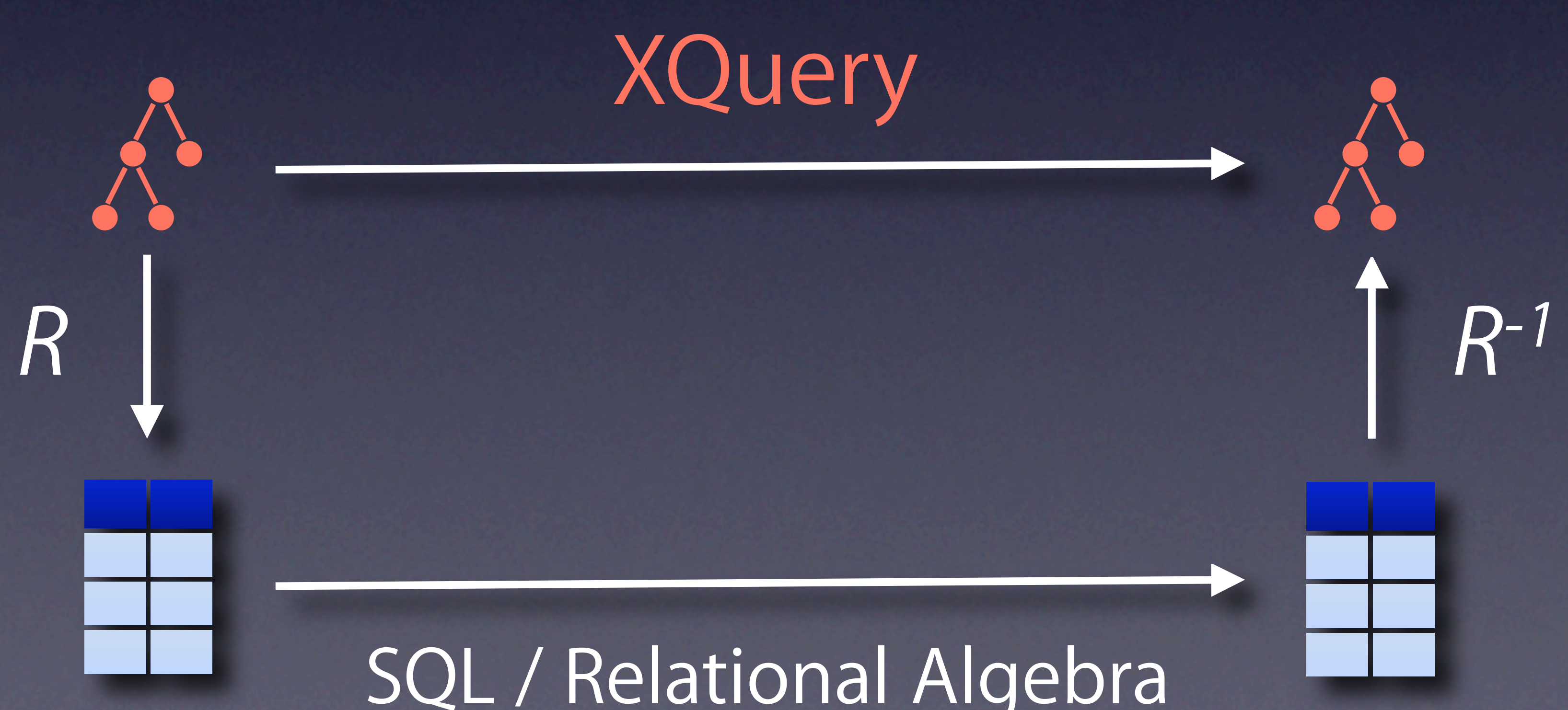
RDBMSs as Processors for Non-Relational Languages

- Relational databases contain the best understood and most scalable query processors available today.
- Can we use relational query engines *as-is* to efficiently process non-relational languages?



RDBMSs as Processors for Non-Relational Languages

- Relational databases contain the best understood and most scalable query processors available today.
- Can we use relational query engines *as-is* to efficiently process non-relational languages?



Purely Relational XQuery

- Find a relational representation of XQuery (XML, XPath) that leverages the table-based processing model:
  - Exploit functionality already built into the database system (do *not* invade the database kernel):

    SQL idioms, OLAP extensions, partitioned B-Trees.
  - Run on top off-the-shelf relational database systems:

Purely Relational XQuery

- Find a relational representation of XQuery (XML, XPath) that leverages the table-based processing model:
  - Exploit functionality already built into the database system (do *not* invade the database kernel):

    SQL idioms, OLAP extensions, partitioned B-Trees.
  - Run on top off-the-shelf relational database systems:

IBM DB2 V9 | SQL Server 2005 | PostgreSQL | kdb+ | MonetDB

Pathfinder

pathfinder-xquery.org

Relational Database System
(Kernel)

Pathfinder

pathfinder-xquery.org

Pathfinder XQuery Compiler

XPath/XQuery Semantics

XML Encoding

Relational Database System
(Kernel)

Pathfinder

pathfinder-xquery.org

XQuery

Pathfinder XQuery Compiler

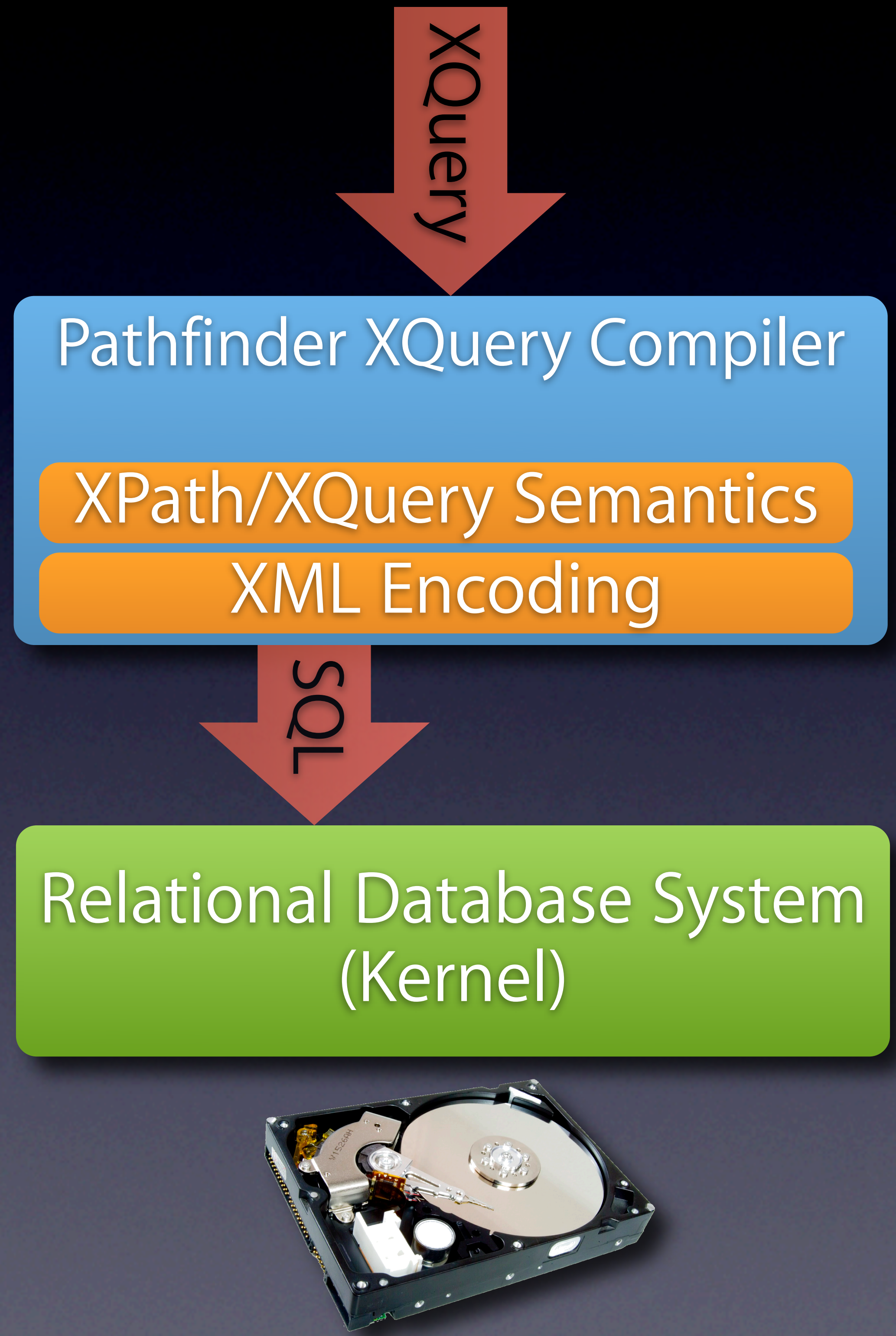
XPath/XQuery Semantics

XML Encoding

Relational Database System
(Kernel)

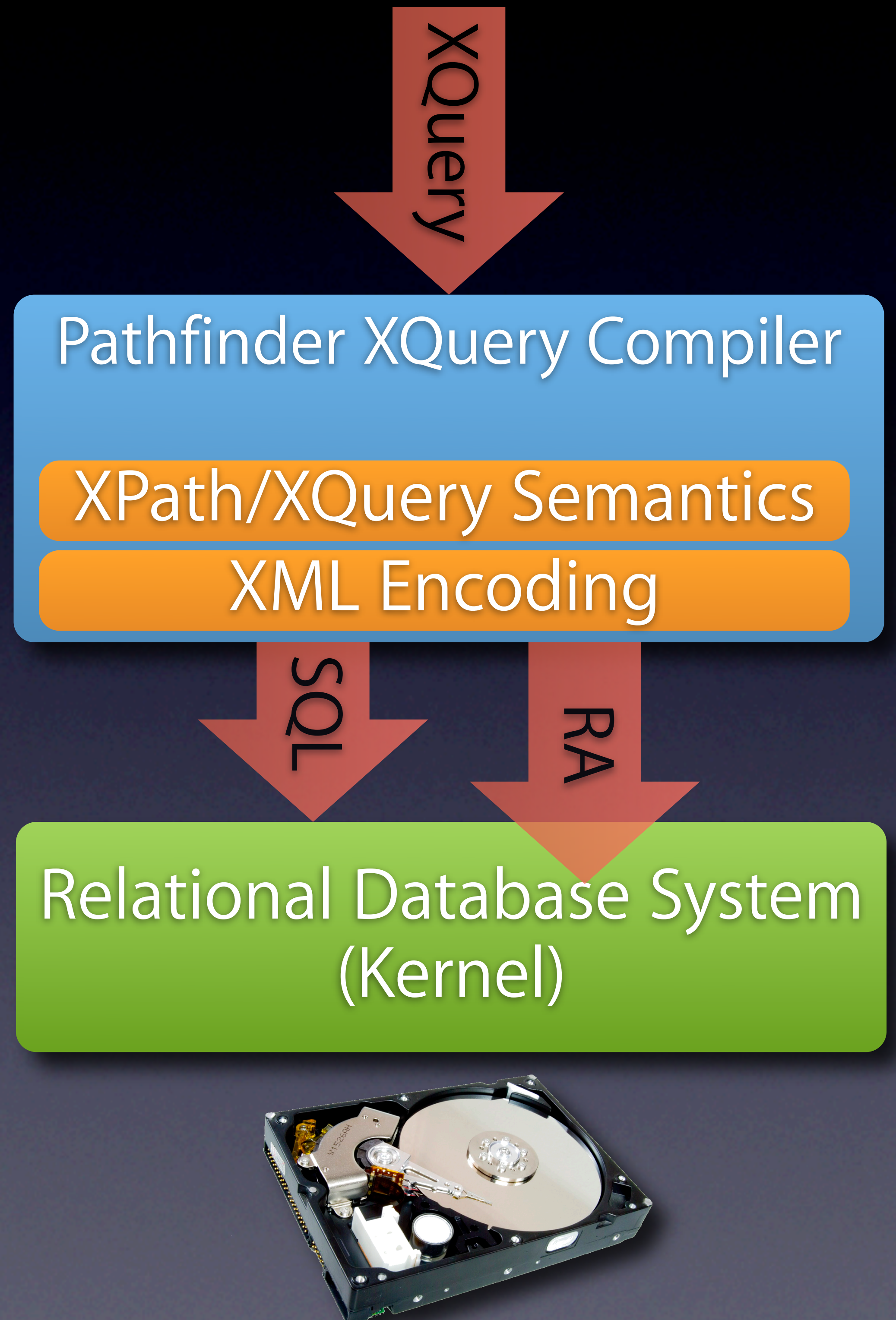
Pathfinder

pathfinder-xquery.org



Pathfinder

pathfinder-xquery.org



Pathfinder & DB2® Against 110+ MB of XML

Pathfinder & DB2®
Against 110+ MB of XML

```
$
```

Slow Motion Replay

Slow Motion Replay

XQuery (XMark Q8)

```
let $a := doc("XMark-110mb.xml")
return
     ⋮
  <item person="{ $p/name/text() }">
   { count($ca) }
  </item>
```

Slow Motion Replay

SQL:1999 (*no* SQL/XML)

```
WITH
  t0000 (iter,pre) AS
    SELECT … FROM … WHERE …
  t0001 (iter) AS
    SELECT … FROM … WHERE …

SELECT   pre,size,kind,…
FROM     …
WHERE    …
ORDER BY …
```

Slow Motion Replay

Relational Encoding of Result

pre	size	kind	···
5072509	2	1	
5072510	0	2	
5072511	0	3	
⋮			

Slow Motion Replay

Serialized XML Text

```
<?xml version="1.0" encoding="UTF-8"?>
<XQuery>
  <item person="Jixiang …">0</item>
  <item person="Harpreet …">0</item>
   ⋮
</XQuery>
```

Table Access Modes

Table Access Modes

- Relational query engines derive much of their efficiency from the simplicity of the *table of tuples* model.

A	B

Table Access Modes

- Relational query engines derive much of their efficiency from the simplicity of the *table of tuples* model.

    1. Sequential scans:
       read-ahead on disks,
       prefetching CPU caches.

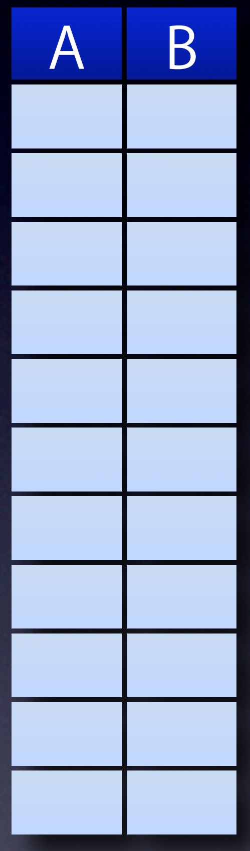


Table Access Modes

- Relational query engines derive much of their efficiency from the simplicity of the *table of tuples* model.

  1. Sequential scans:
     read-ahead on disks,
     prefetching CPU caches.

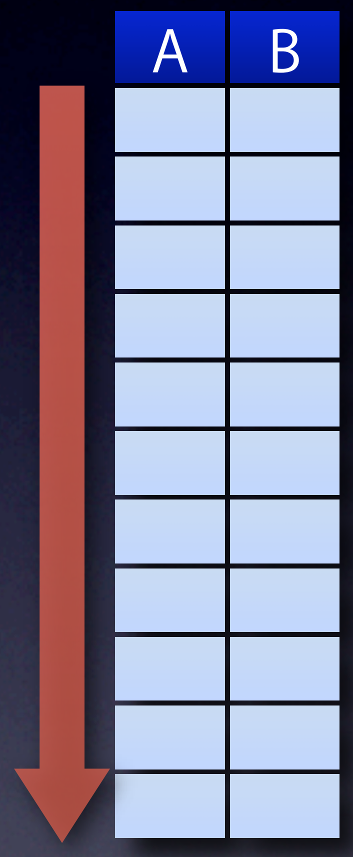


Table Access Modes

- Relational query engines derive much of their efficiency from the simplicity of the *table of tuples* model.

1. Sequential scans:
   read-ahead on disks,
   prefetching CPU caches.

2. Index-based access:
   B-Trees, range scans.

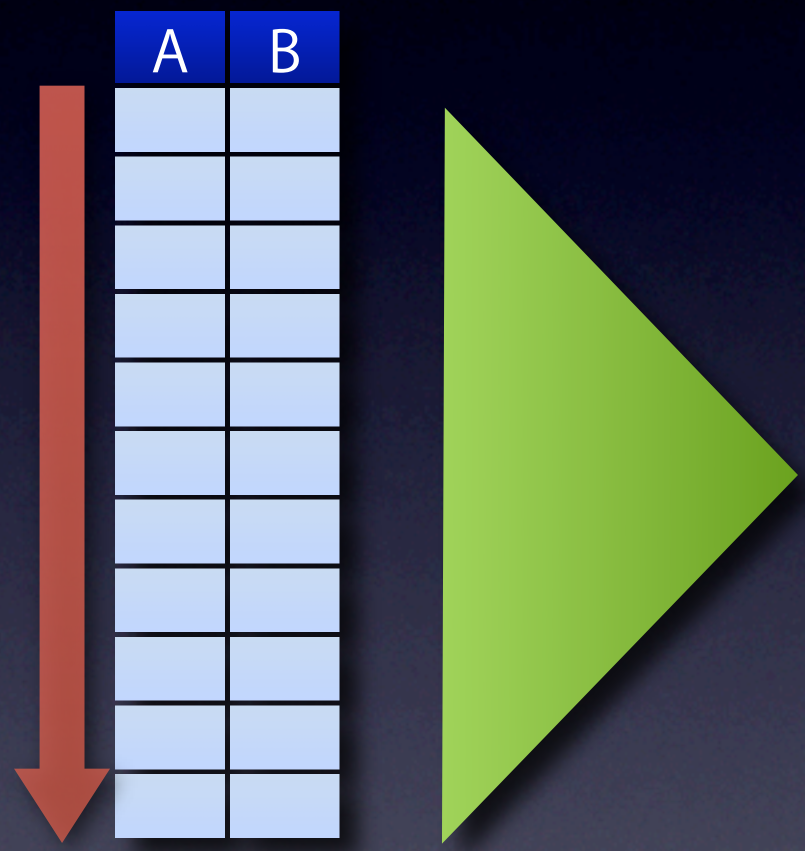


Table Access Modes

- Relational query engines derive much of their efficiency from the simplicity of the *table of tuples* model.

1. Sequential scans:
   read-ahead on disks,
   prefetching CPU caches.

2. Index-based access:
   B-Trees, range scans.

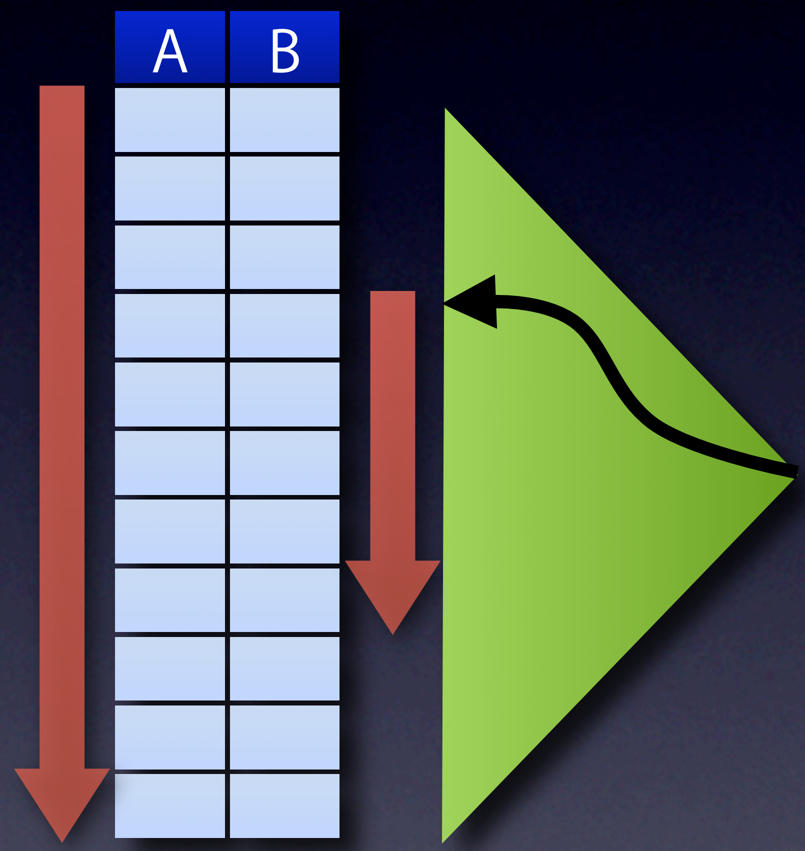
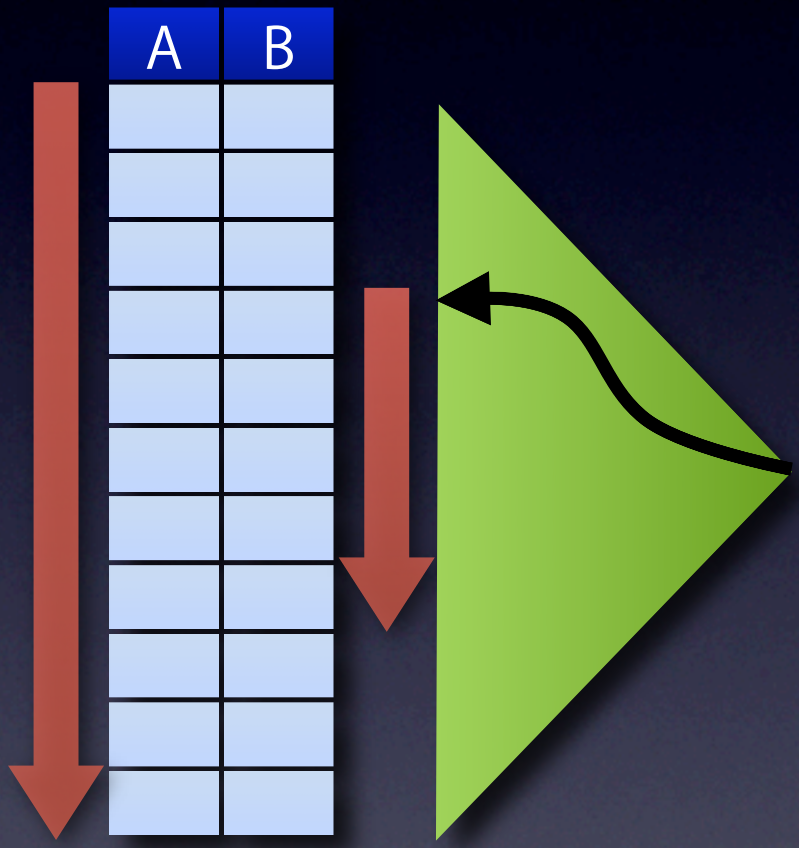


Table Access Modes

- Relational query engines derive much of their efficiency from the simplicity of the *table of tuples* model.

  1. Sequential scans: read-ahead on disks, prefetching CPU caches.

  2. Index-based access: B-Trees, range scans.



- But: We will benefit **only if** we actually operate the relational database in this bulk-oriented mode.

The Core of XQuery

The Core of XQuery

$(e_1, \ldots, e_n)$

ordered item sequences

The Core of XQuery

```
(e1,..,en)
for $x in e1 return e2
```

ordered item sequences
iteration

The Core of XQuery

```
(e1,..,en)
for $x in e1 return e2
let $x := e1 return e2
```

ordered item sequences
iteration
variable binding

The Core of XQuery

```
(e1,..,en)
for $x in e1 return e2
let $x := e1 return e2
if (e1) then e2 else e3
```

ordered item sequences
iteration
variable binding
conditionals

The Core of XQuery

```
(e1,..,en)
for $x in e1 return e2
let $x := e1 return e2
if (e1) then e2 else e3
<t> { e1 } </t>
```

ordered item sequences
iteration
variable binding
conditionals
XML node construction

The Core of XQuery

`(`e_1`,..,`e_n`)`	ordered item sequences
`for $x in` e_1 `return` e_2	iteration
`let $x :=` e_1 `return` e_2	variable binding
`if (`e_1`) then` e_2 `else` e_3	conditionals
`<t> {` e_1 `} </t>`	XML node construction
e_1`/child::t`, e_1`/following::t`	XPath location steps

The Core of XQuery

`(`e_1`,...,`e_n`)`	ordered item sequences
`for $x in` e_1 `return` e_2	iteration
`let $x :=` e_1 `return` e_2	variable binding
`if (`e_1`) then` e_2 `else` e_3	conditionals
`<t> {` e_1 `} </t>`	XML node construction
e_1`/child::t`, e_1`/following::t`	XPath location steps
`unordered {` e_1 `}`	local order indifference

The Core of XQuery

`(`e_1`,...,`e_n`)`	ordered item sequences
`for $x in` e_1 `return` e_2	iteration
`let $x :=` e_1 `return` e_2	variable binding
`if (`e_1`) then` e_2 `else` e_3	conditionals
`<t> {` e_1 `} </t>`	XML node construction
e_1`/child::t`, e_1`/following::t`	XPath location steps
`unordered {` e_1 `}`	local order indifference
`fn:doc(`e_1`)`, `fn:data(`e_1`)`	built-in functions

\+ Document order, node identity, dynamic typing, schema validation, user-defined functions, ...

The Core of XQuery

`(`e_1`,...,`e_n`)`	ordered item sequences
`for $x in` e_1 `return` e_2	iteration
`let $x :=` e_1 `return` e_2	variable binding
`if (`e_1`) then` e_2 `else` e_3	conditionals
`<t> {` e_1 `} </t>`	XML node construction
e_1`/child::t`, e_1`/following::t`	XPath location steps
`unordered {` e_1 `}`	local order indifference
`fn:doc(`e_1`)`, `fn:data(`e_1`)`	built-in functions

+ Document order, node identity, dynamic typing, schema validation, user-defined functions, ...

- The XQuery–SQL gap appears substantial.

FP-Style Iteration

FP-Style Iteration

- The XQuery `for..in..return` construct performs side-effect free iteration:

$$\texttt{for \$x in (}e_1\texttt{,..,}e_\text{n}\texttt{)}$$
$$\texttt{return } b$$

FP-Style Iteration

- The XQuery `for..in..return` construct performs side-effect free iteration:

$$\texttt{for \$x in (}e_1\texttt{,..,}e_n\texttt{)}$$
$$\texttt{return } b$$
$$\equiv$$
$$(\ b[e_1/\texttt{\$x}], \ldots, b[e_n/\texttt{\$x}]\ )$$

- Individual iterations cannot interfere and may be evaluated in any order (or in parallel).

Loop Lifting

- "Relational loop unrolling":
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

Loop Lifting

- "Relational loop unrolling":
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

```
for $x in (10,20,30)
return $x + 42
```

Loop Lifting

- “Relational loop unrolling”:
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	42
2	1	42
3	1	42

Loop Lifting

- "Relational loop unrolling":
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	42
2	1	42
3	1	42

```
for $x in (10,20,30)
return $x + 42
```

Loop Lifting

- "Relational loop unrolling":
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	42
2	1	42
3	1	42

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

Loop Lifting

- "Relational loop unrolling":
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	42
2	1	42
3	1	42

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

```
let $x := (10,20,30)
return $x
```

Loop Lifting

- "Relational loop unrolling":
  For any XQuery subexpression, generate a relational plan that evaluates the expression in *all iterations*.

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	42
2	1	42
3	1	42

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

```
let $x := (10,20,30)
return $x
```

iter	pos	item
1	1	10
1	2	20
1	3	30

Loop Lifting

Loop Lifting

```
for $x in (10,20,30)
return $x + 42
```

Loop Lifting

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

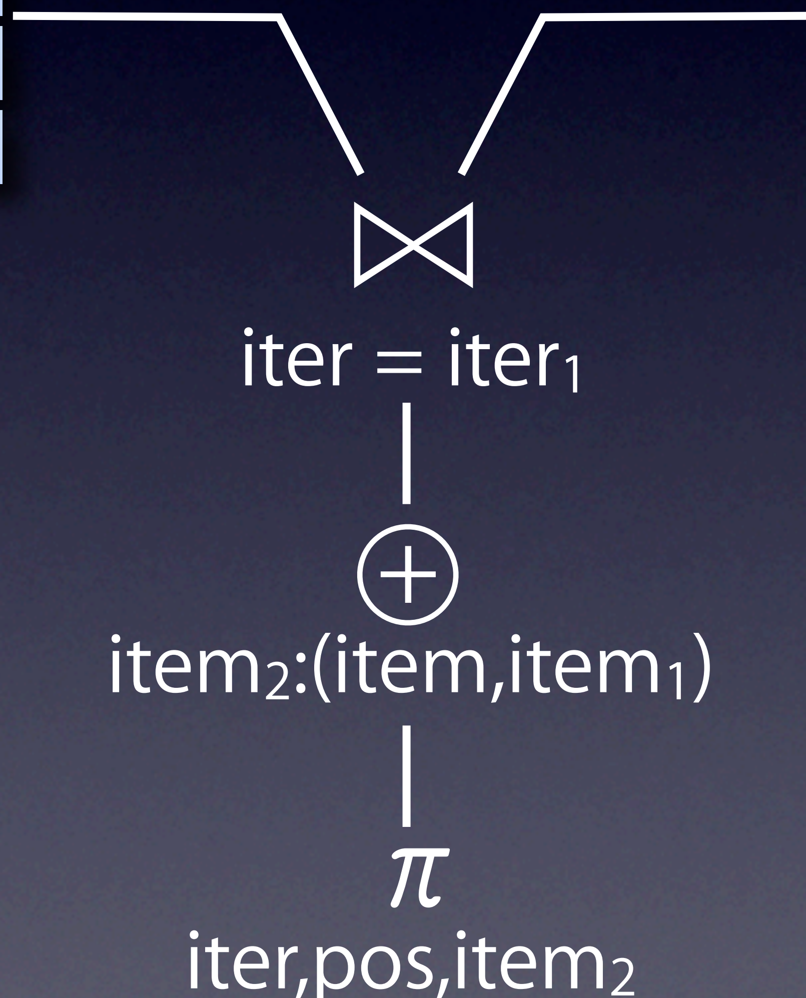
$iter_1$	pos_1	$item_1$
1	1	42
2	1	42
3	1	42

Loop Lifting

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

$iter_1$	pos_1	$item_1$
1	1	42
2	1	42
3	1	42



Loop Lifting

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

$iter_1$	pos_1	$item_1$
1	1	42
2	1	42
3	1	42

$\bowtie$
$iter = iter_1$

$\oplus$
$item_2{:}(item, item_1)$

π
$iter, pos, item_2$

iter	pos	item	$iter_1$	pos_1	$item_1$
1	1	10	1	1	42
2	1	20	2	1	42
3	1	30	3	1	42

Loop Lifting

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

$iter_1$	pos_1	$item_1$
1	1	42
2	1	42
3	1	42

$\bowtie$
$iter = iter_1$

$\oplus$
$item_2$:$(item, item_1)$

iter	pos	item	$iter_1$	pos_1	$item_1$	$item_2$
1	1	10	1	1	42	52
2	1	20	2	1	42	62
3	1	30	3	1	42	72

π
$iter, pos, item_2$

Loop Lifting

```
for $x in (10,20,30)
return $x + 42
```

iter	pos	item
1	1	10
2	1	20
3	1	30

$iter_1$	pos_1	$item_1$
1	1	42
2	1	42
3	1	42

$\bowtie_{iter = iter_1}$

$\oplus_{item_2:(item, item_1)}$

$\pi_{iter, pos, item_2}$

iter	pos	$item_2$
1	1	52
2	1	62
3	1	72

Loop Lifting

Loop Lifting

```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```

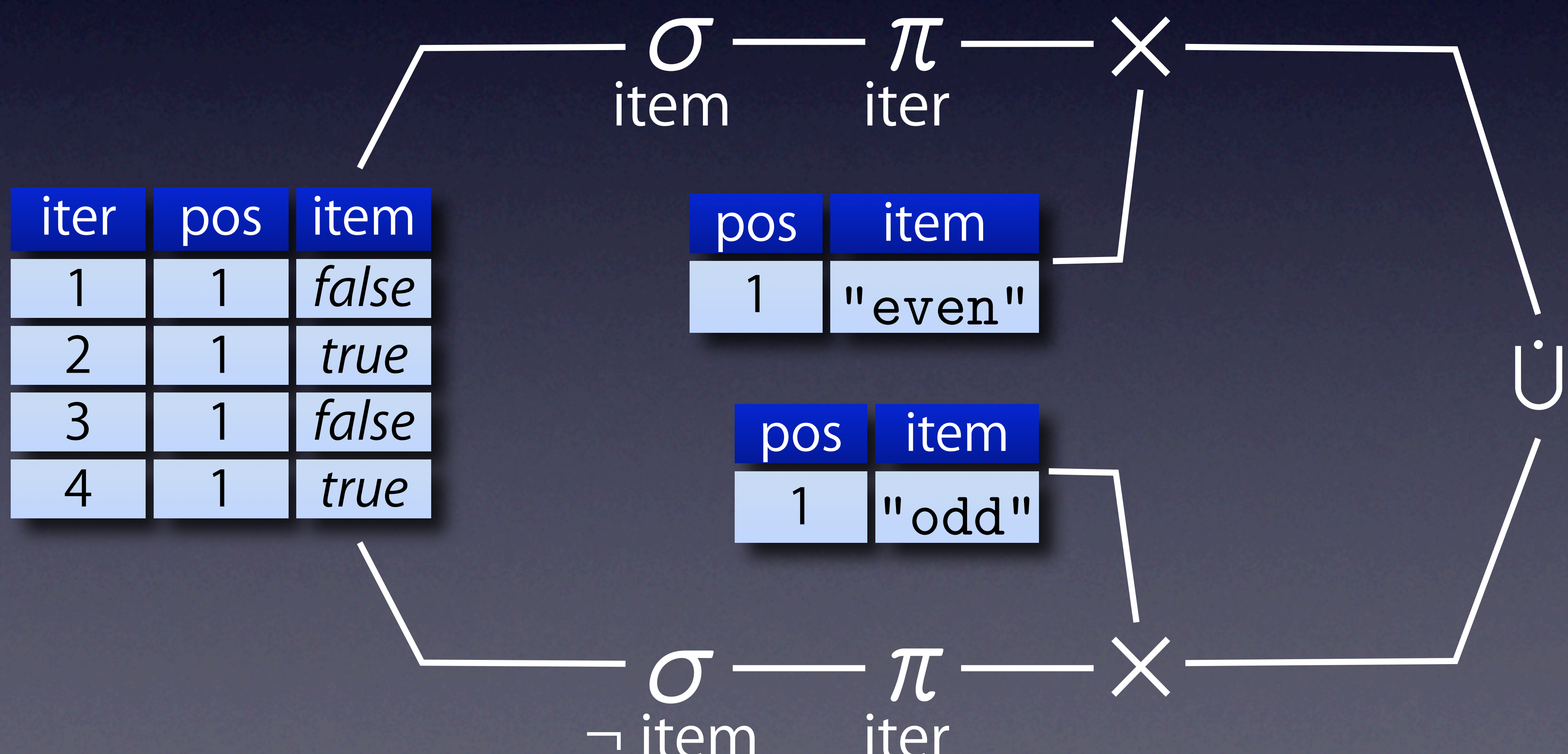
Loop Lifting

```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```

iter	pos	item
1	1	*false*
2	1	*true*
3	1	*false*
4	1	*true*

Loop Lifting

```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```



iter	pos	item
1	1	*false*
2	1	*true*
3	1	*false*
4	1	*true*

pos	item
1	"even"

pos	item
1	"odd"

Loop Lifting

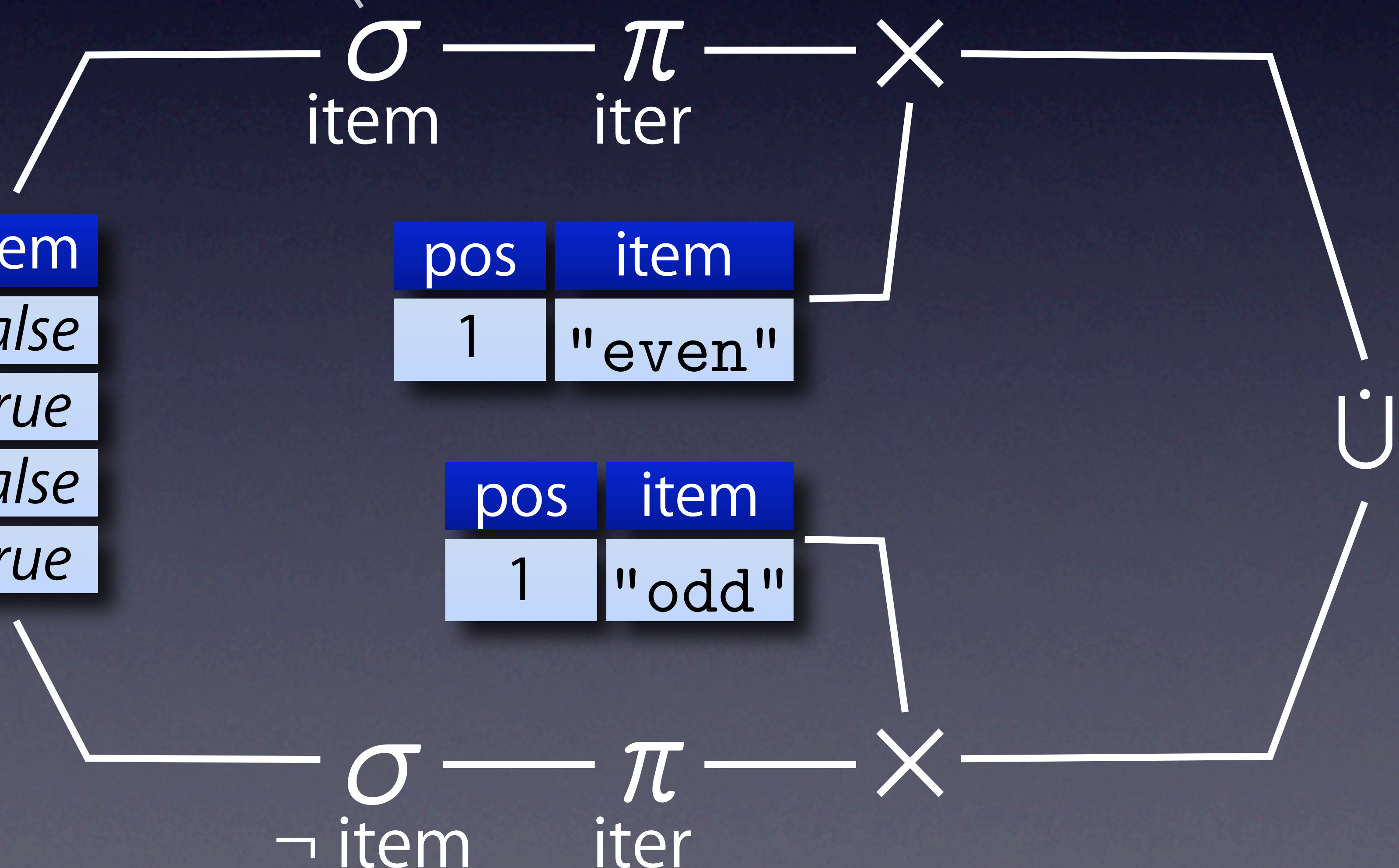
```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```

iter	pos	item
2	1	*true*
4	1	*true*

iter	pos	item
1	1	*false*
2	1	*true*
3	1	*false*
4	1	*true*

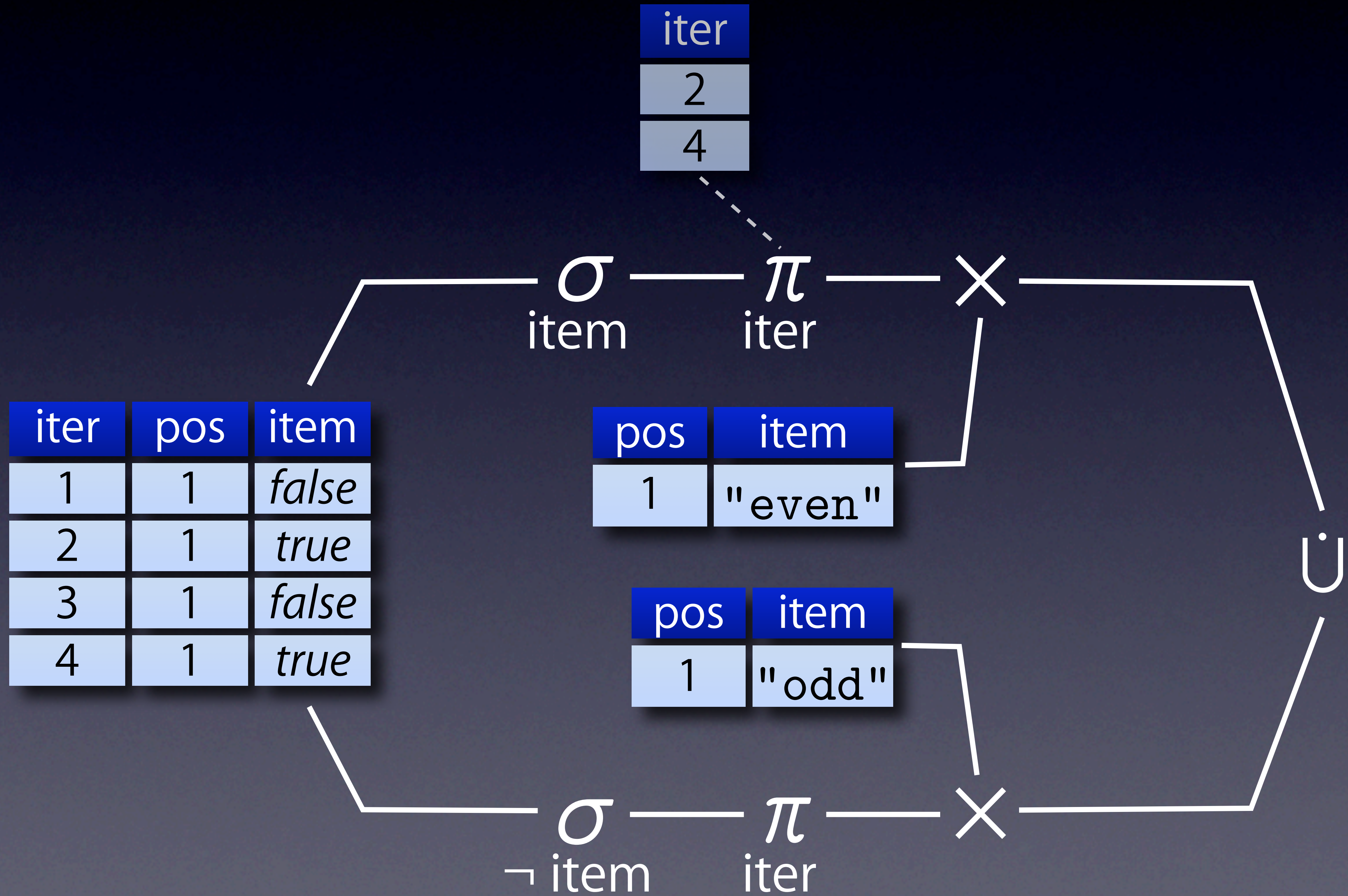
pos	item
1	"even"

pos	item
1	"odd"



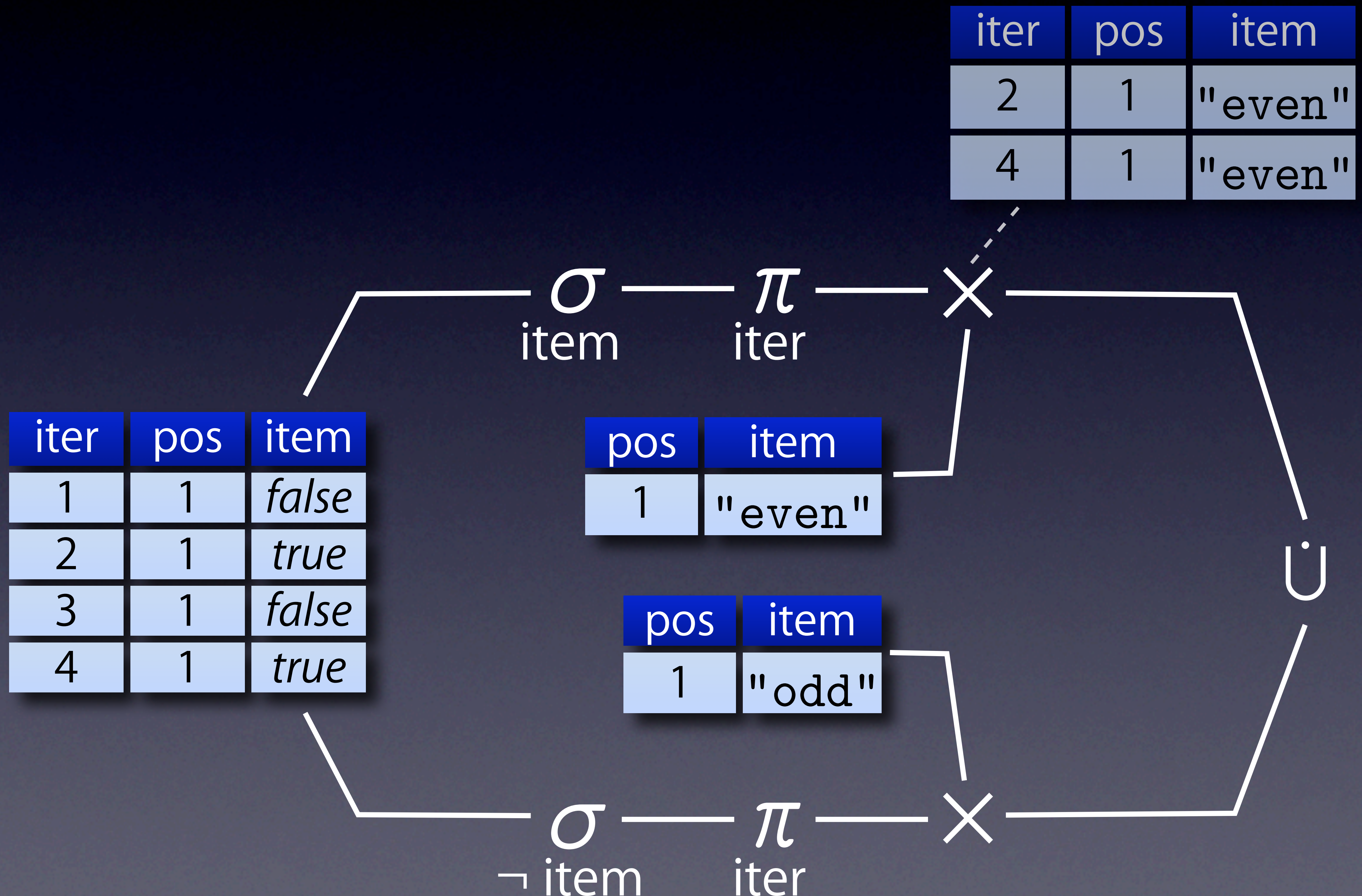
Loop Lifting

```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```



Loop Lifting

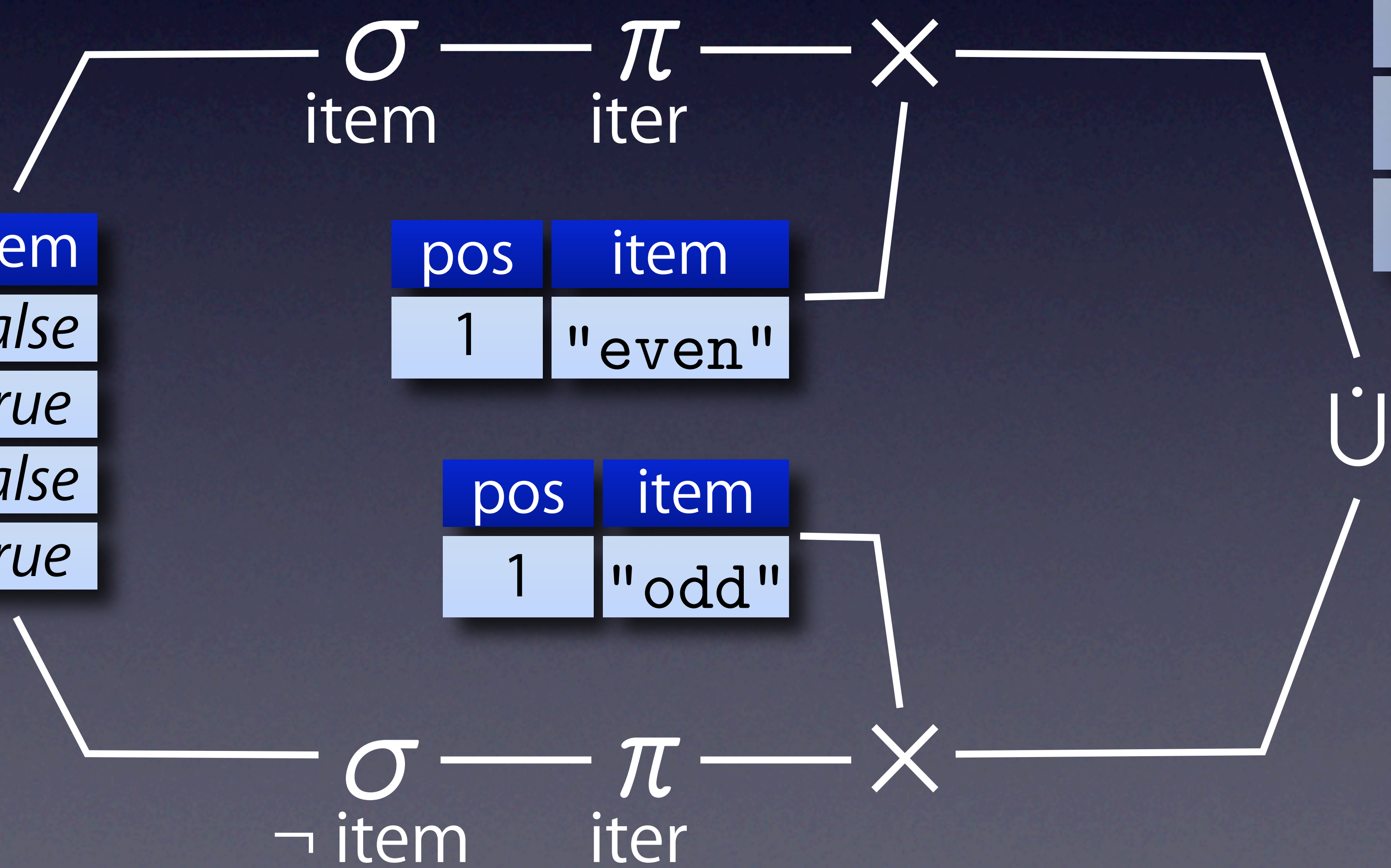
```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```



Loop Lifting

```
for $x in (1,2,3,4)
return if ($x mod 2 = 0) then "even" else "odd"
```

iter	pos	item
1	1	*false*
2	1	*true*
3	1	*false*
4	1	*true*

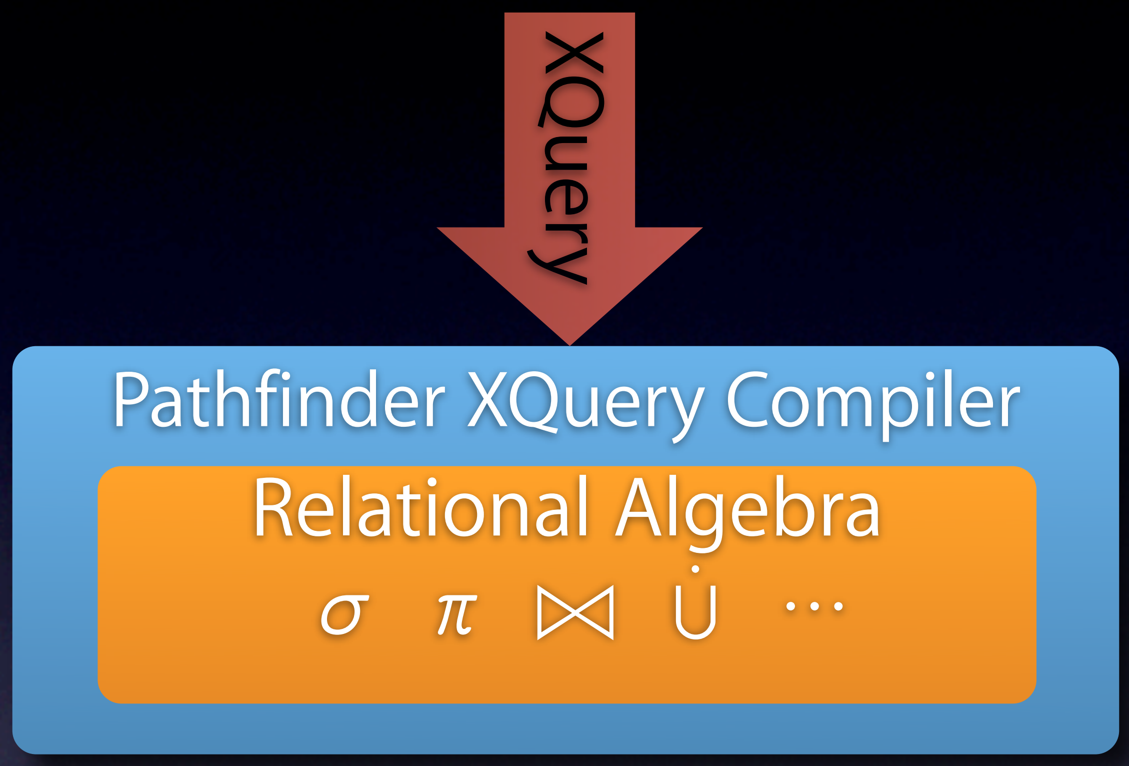


pos	item
1	"even"

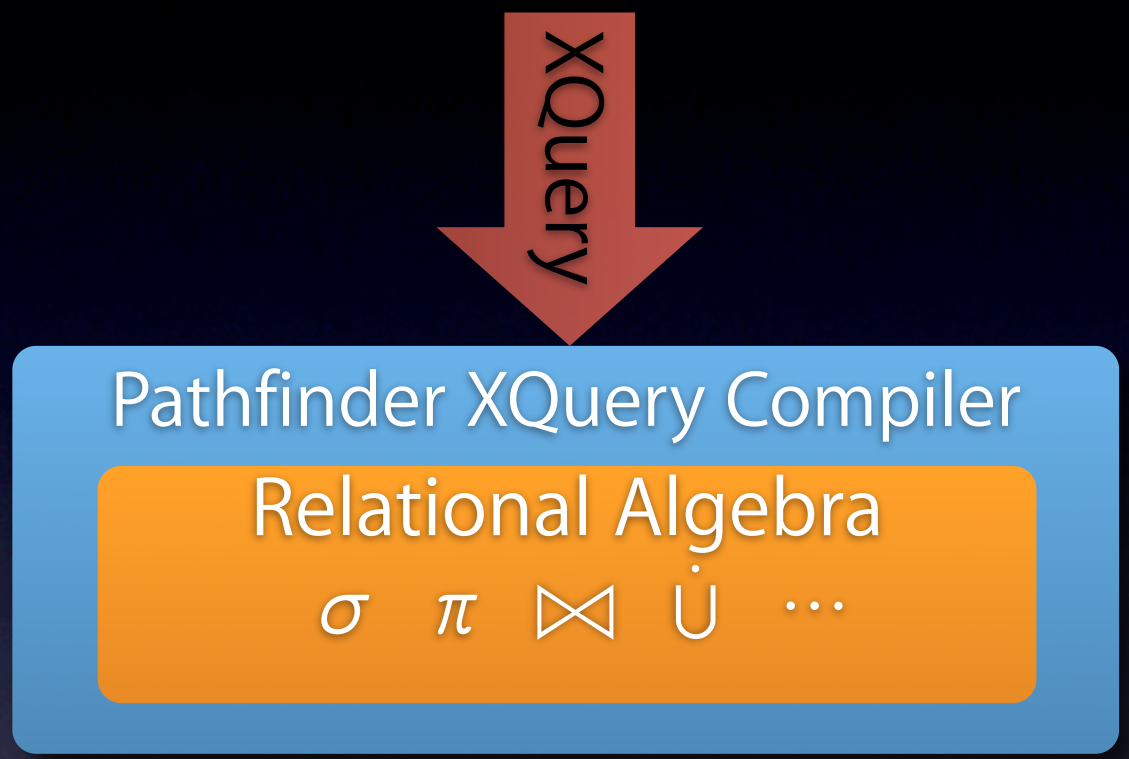
pos	item
1	"odd"

iter	pos	item
1	1	"odd"
2	1	"even"
3	1	"odd"
4	1	"even"

Intermediate Code: Relational Algebra

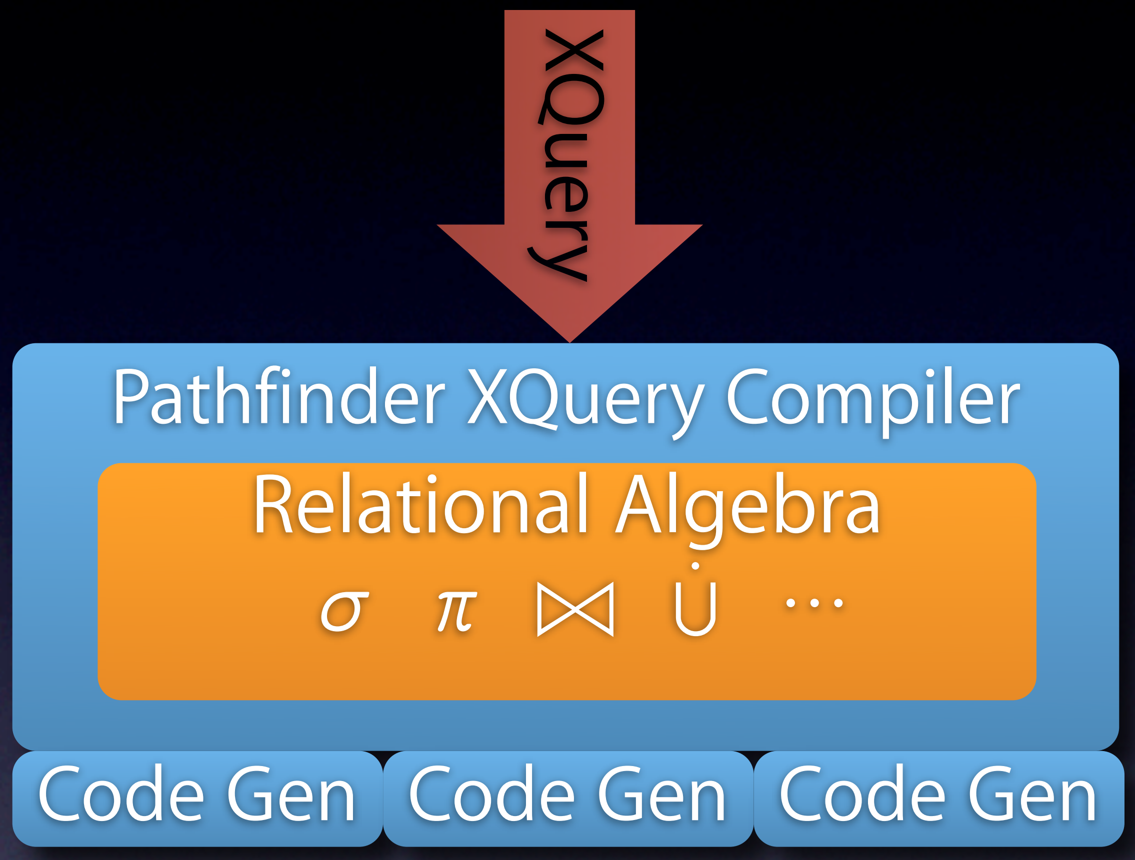


Intermediate Code: Relational Algebra



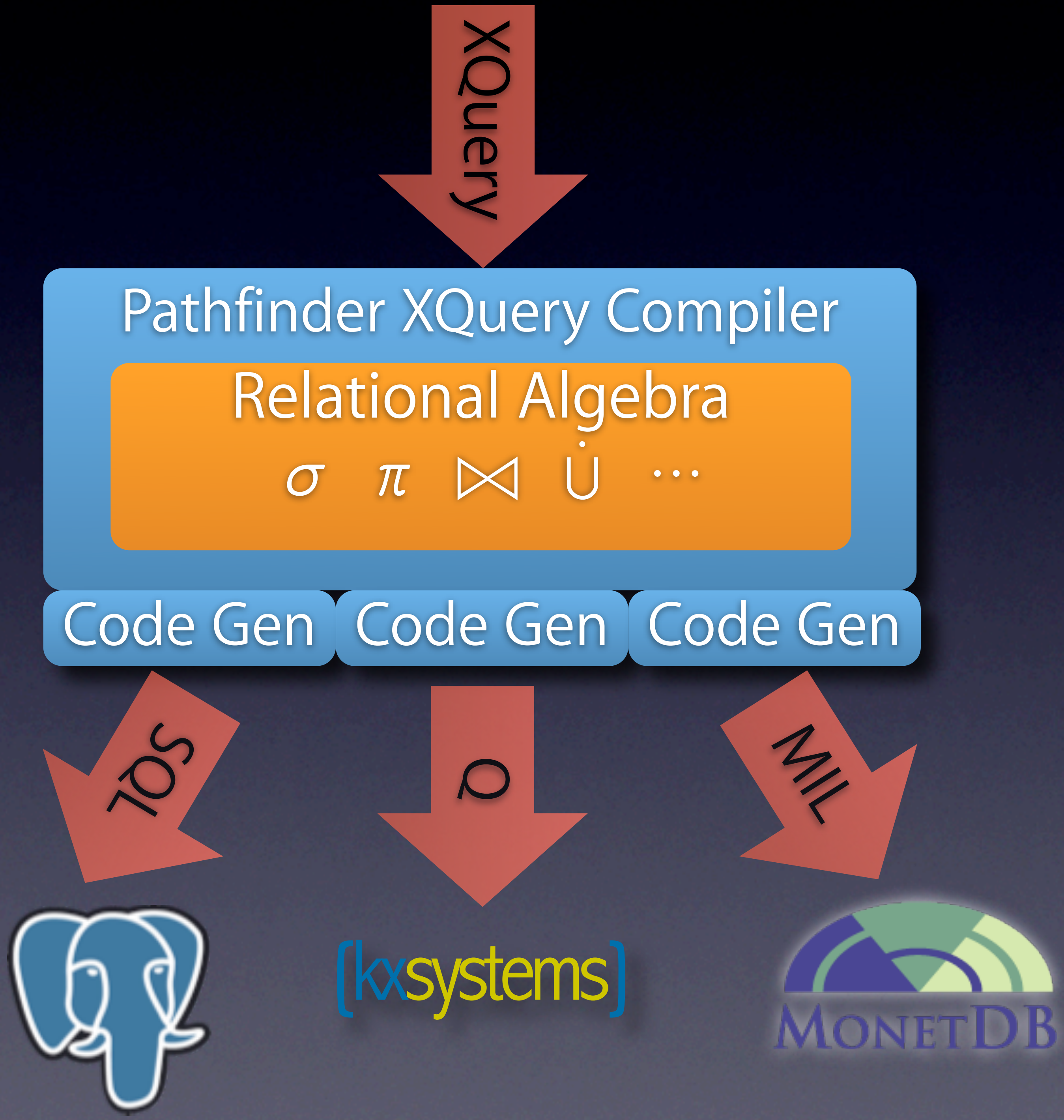
- Compiles into RA that mimics capabilites of SQL:1999 query engines.
- RA dialect is ...
  1. primitive
  2. explicit
  3. designed to be easily analyzable

Intermediate Code: Relational Algebra



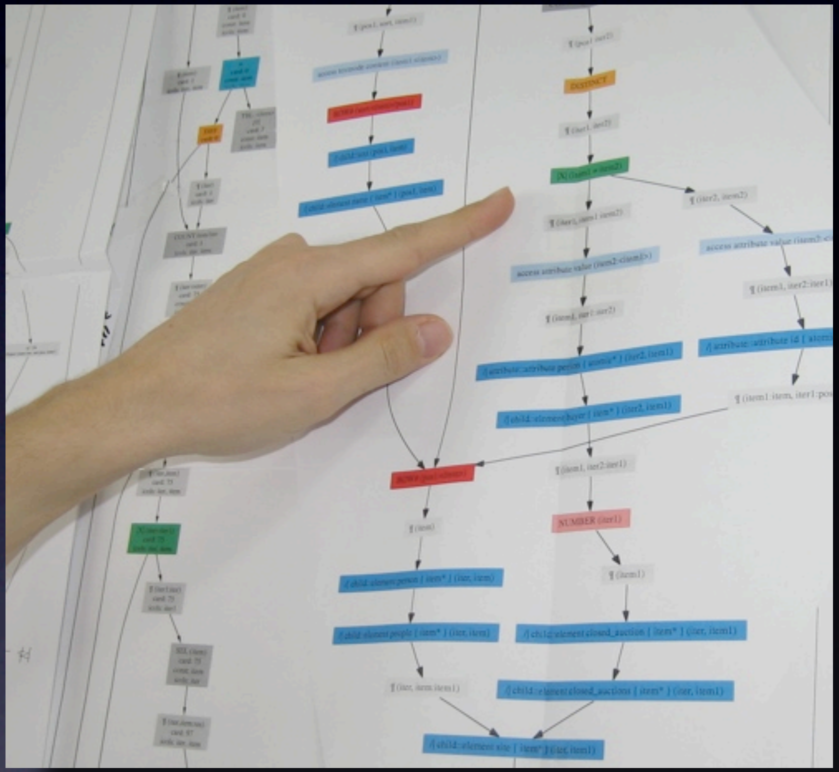
- Compiles into RA that mimics capabilites of SQL:1999 query engines.
- RA dialect is ...
  1. primitive
  2. explicit
  3. designed to be easily analyzable

Intermediate Code: Relational Algebra

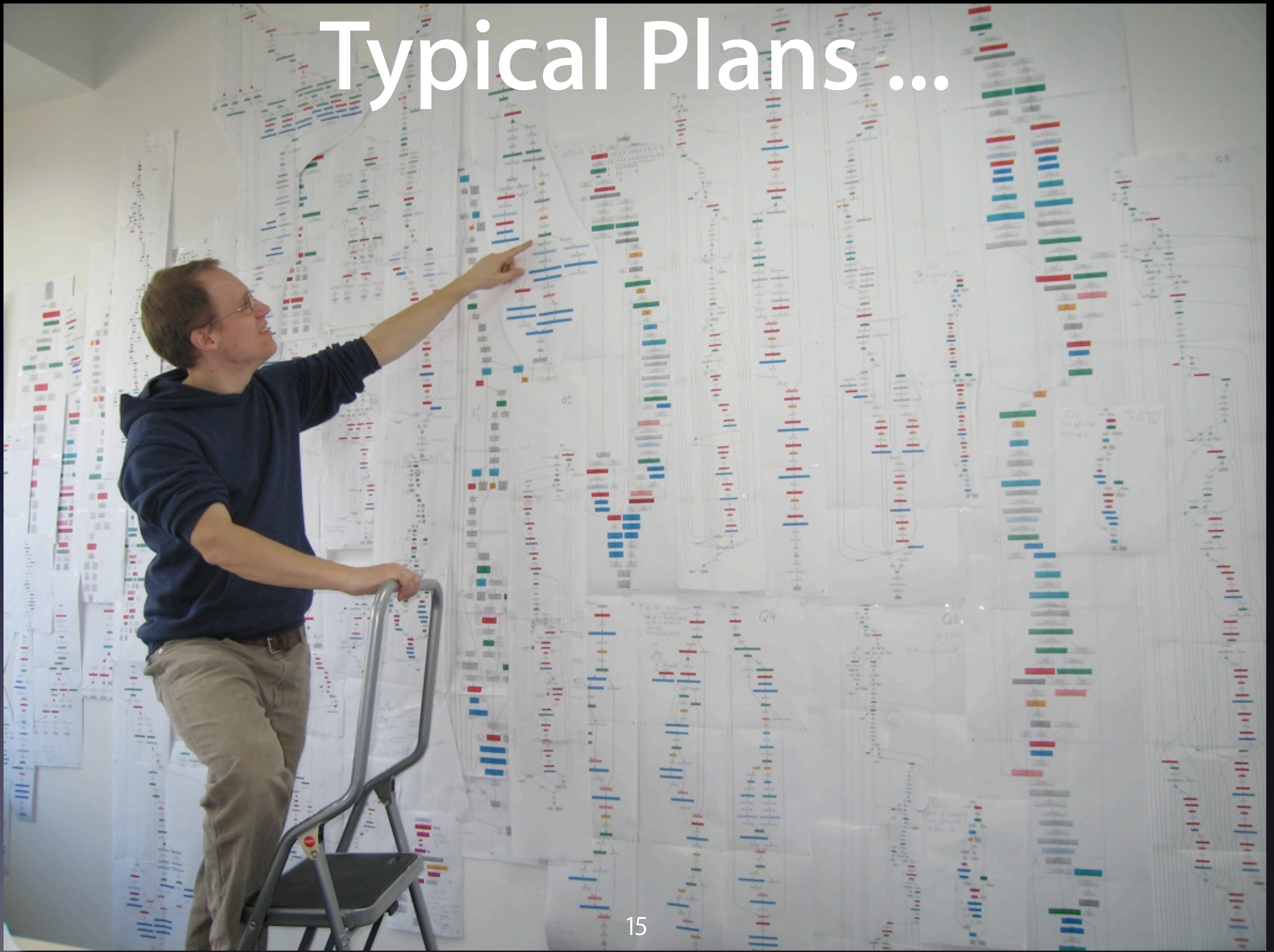


- Compiles into RA that mimics capabilites of SQL:1999 query engines.
- RA dialect is ...
  1. primitive
  2. explicit
  3. designed to be easily analyzable

Typical Plans …



Typical Plans ...



Relational XQuery Optimization

```
let $d := fn:doc(···)
return for $a in $d//a
            for $b in $d//b
            where $b/@c = $a/@d
            return $b
```

Relational XQuery Optimization

- We use concepts in the *relational domain* to analyze plans and to steer simplification and optimization.
- Detect join-like XQuery expressions. Let XQuery's syntactic diversity not affect the detection:

```
let $d := fn:doc(···)
return for $a in $d//a
            for $b in $d//b
            where $b/@c = $a/@d
            return $b
```

Relational XQuery Optimization

- We use concepts in the *relational domain* to analyze plans and to steer simplification and optimization.
- Detect join-like XQuery expressions. Let XQuery's syntactic diversity not affect the detection:

```
let $d := fn:doc(···)
return $d//b[@c = $d//a/@c]
```

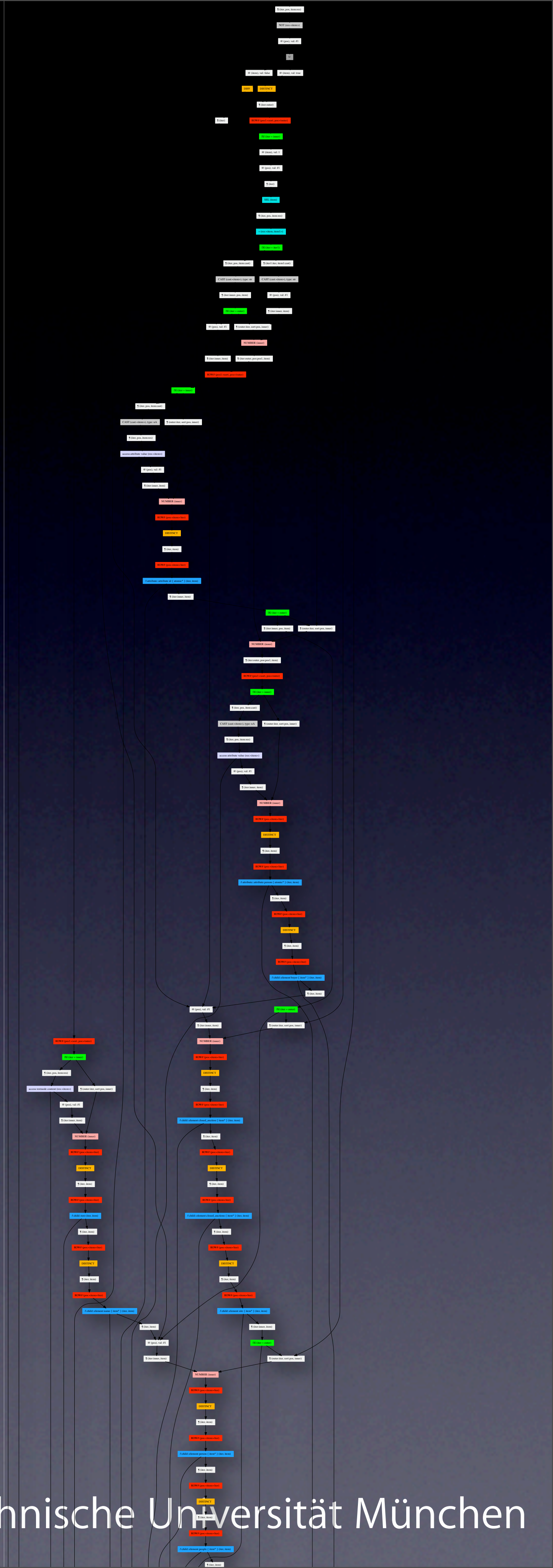
Relational XQuery Optimization

- We use concepts in the *relational domain* to analyze plans and to steer simplification and optimization.
- Detect join-like XQuery expressions. Let XQuery's syntactic diversity not affect the detection:

```
let $d := fn:doc(⋯)
return $d//b[@c = $d//a/@c]
```

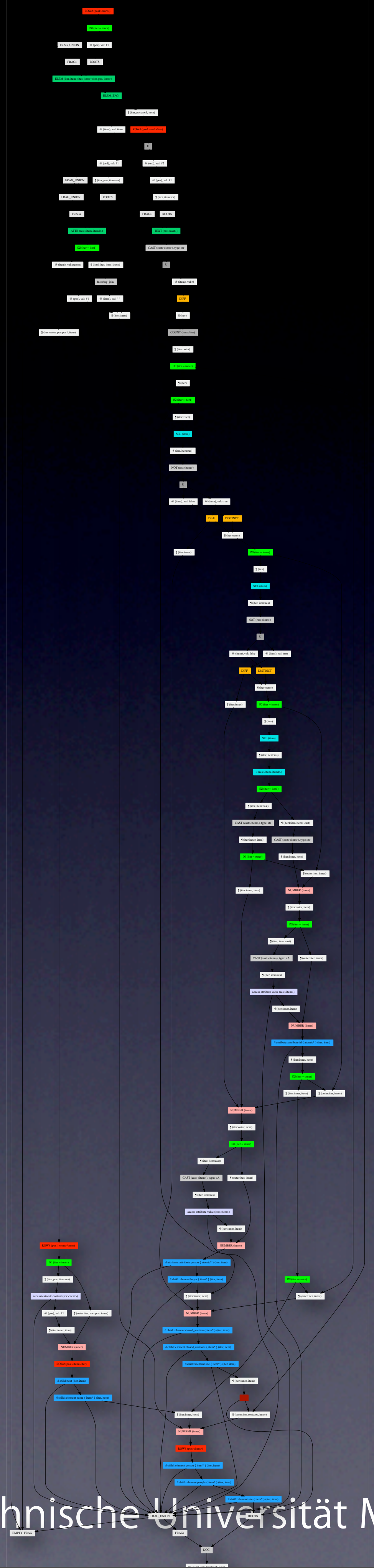
- For both queries, the value-based XQuery join surfaces as a multi-valued dependency in the algebraic plans.

Rewriting XMark Q8



Rewriting XMark Q8

- Rewriting phases:
  1. Constant propagation and projection pushdown,



Rewriting XMark Q8

- Rewriting phases:

  1. Constant propagation and projection pushdown,
  2. functional dependency and data flow analysis,

Rewriting XMark Q8

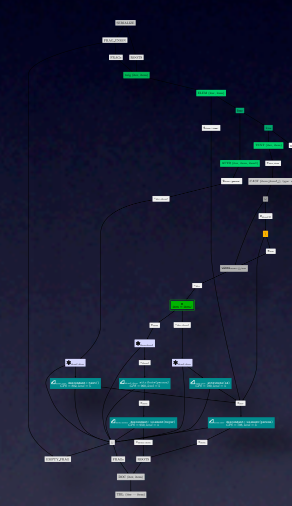
- Rewriting phases:
  1. Constant propagation and projection pushdown,
  2. functional dependency and data flow analysis,
  3. algebraic XQuery join detection,

Rewriting XMark Q8

- Rewriting phases:
  1. Constant propagation and projection pushdown,
  2. functional dependency and data flow analysis,
  3. algebraic XQuery join detection,
  4. XPath join graph isolation,

Rewriting XMark Q8

- Rewriting phases:
  1. Constant propagation and projection pushdown,
  2. functional dependency and data flow analysis,
  3. algebraic XQuery join detection,
  4. XPath join graph isolation,
  5. data guide exploitation (“*Node GPS*”).



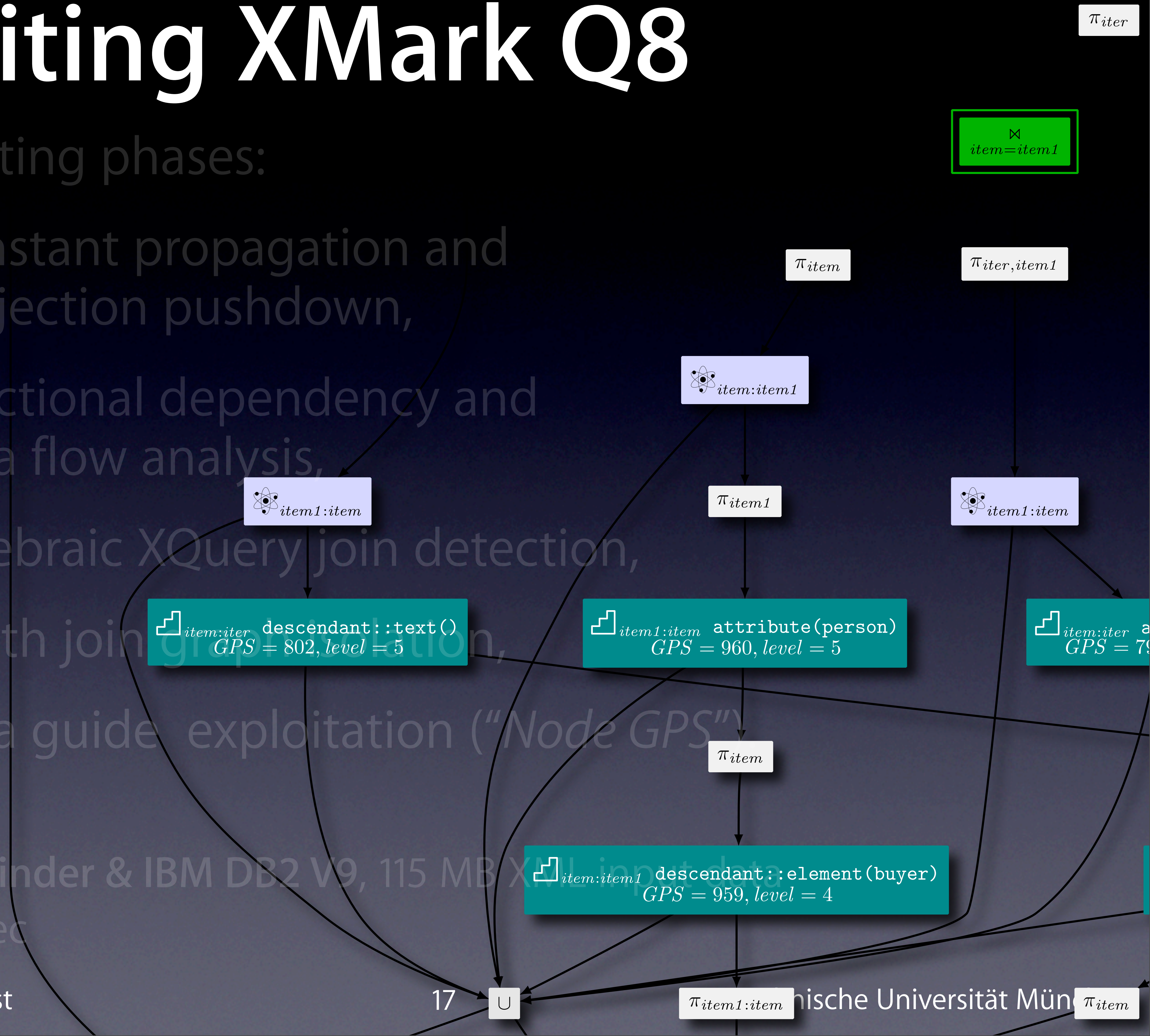
Pathfinder & IBM DB2 V9, 115 MB XML input data
< 1 sec

Rewriting XMark Q8

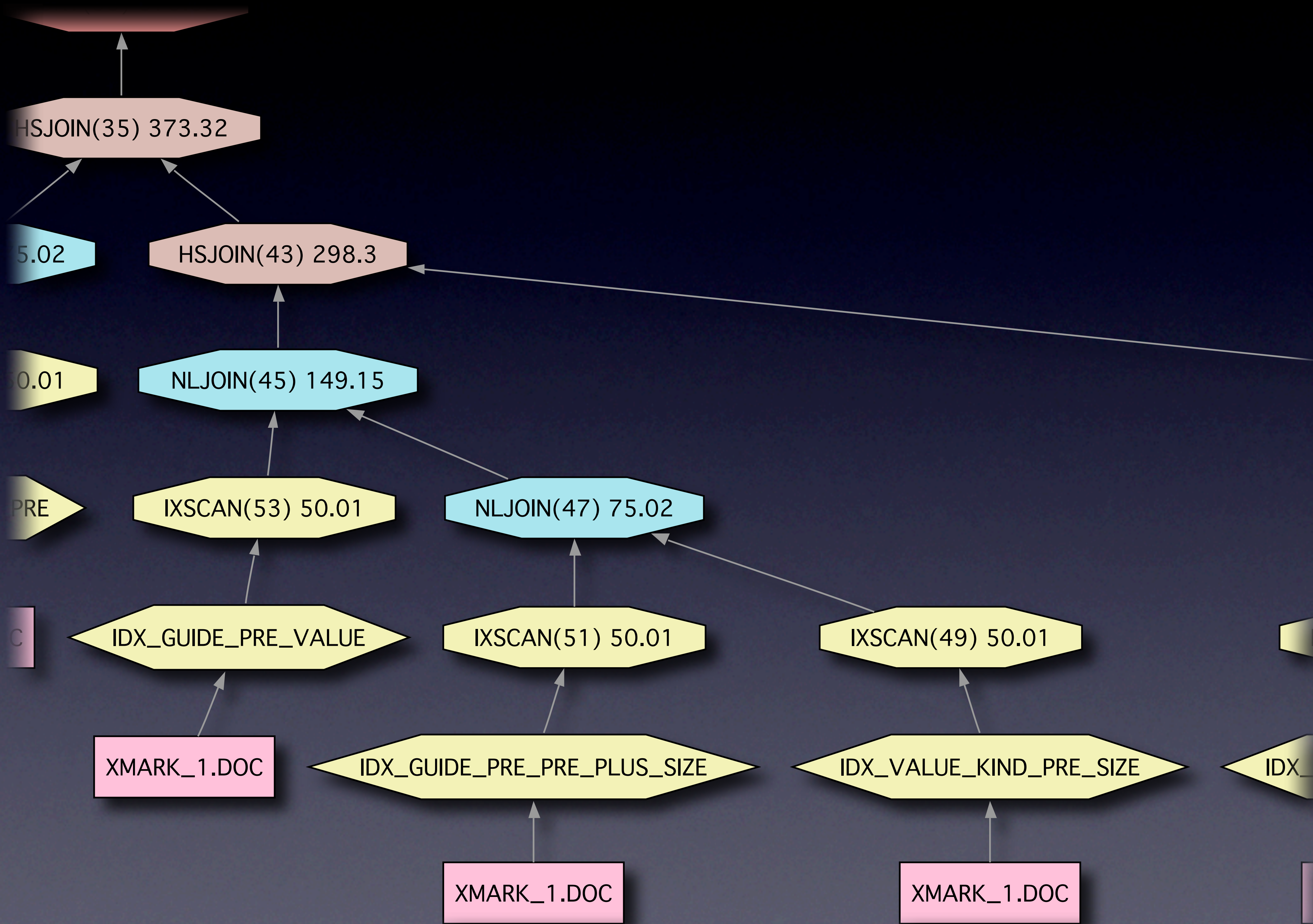
- Rewriting phases:

1. Constant propagation and projection pushdown,
2. functional dependency and data flow analysis,
3. algebraic XQuery join detection,
4. XPath join graph isolation,
5. data guide exploitation (“*Node GPS*”)

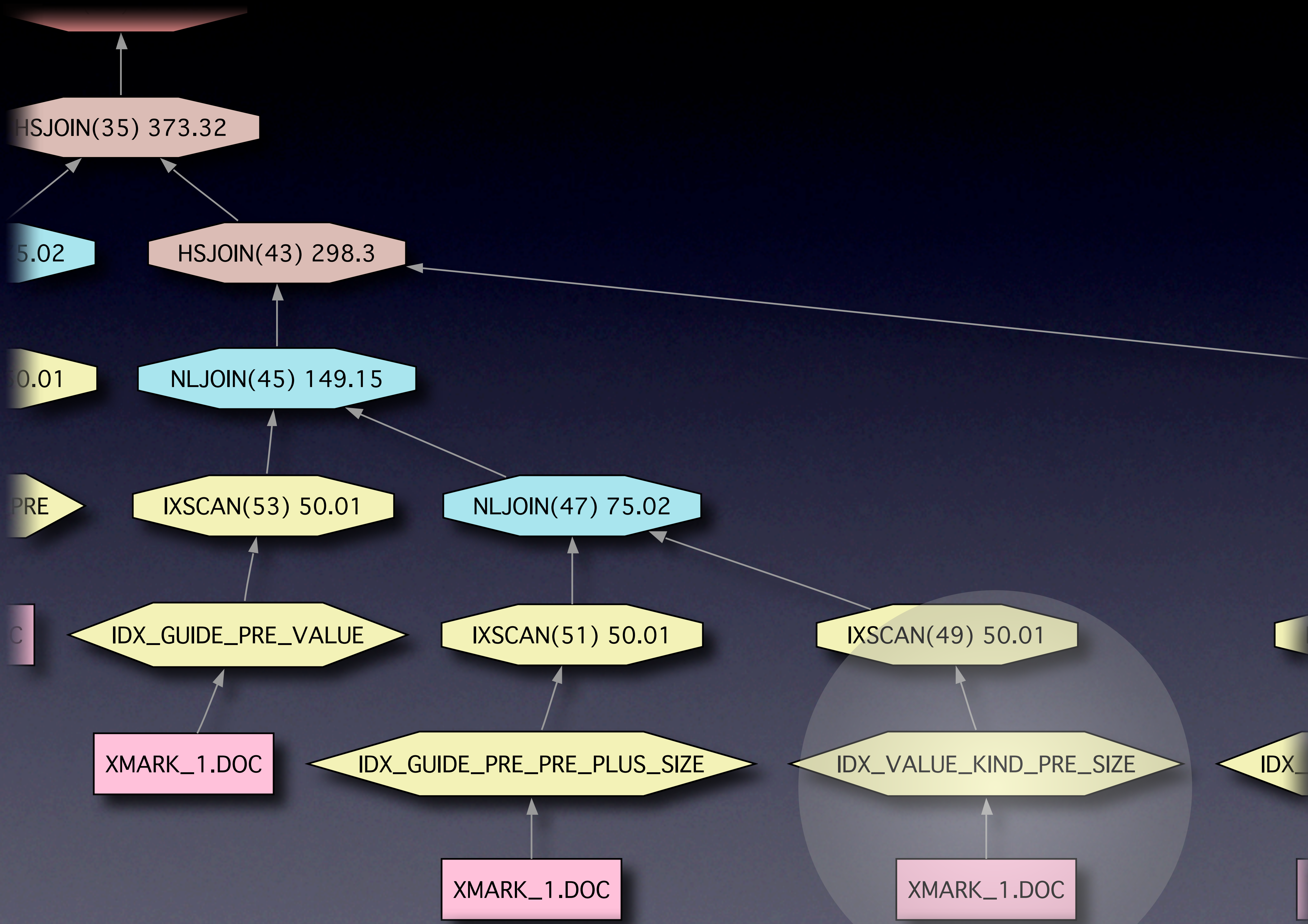
Pathfinder & IBM DB2 V9, 115 MB XML input data
< 1 sec



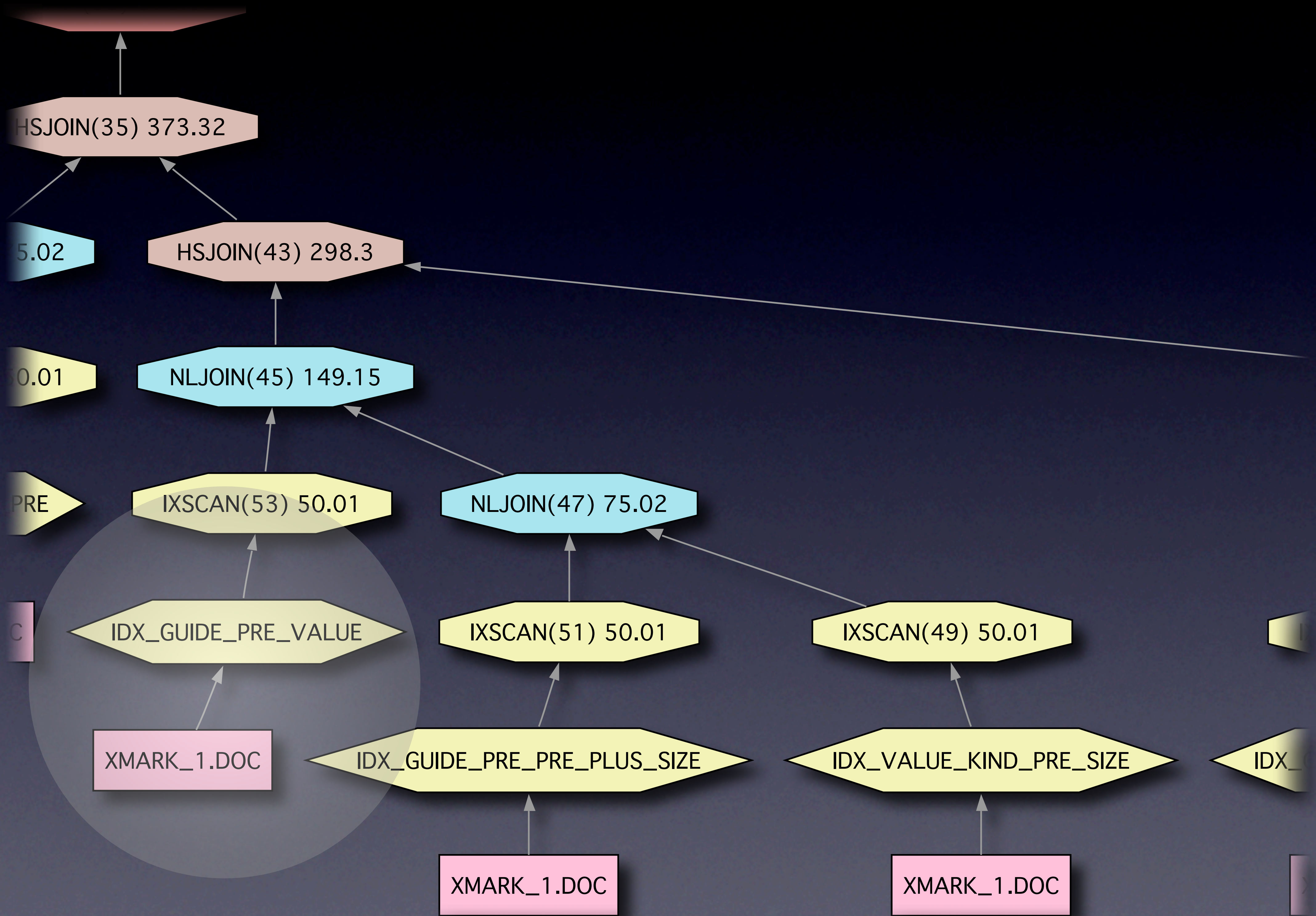
Can DB2® Cope?


HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
IDX_
XMARK_1.DOC
XMARK_1.DOC

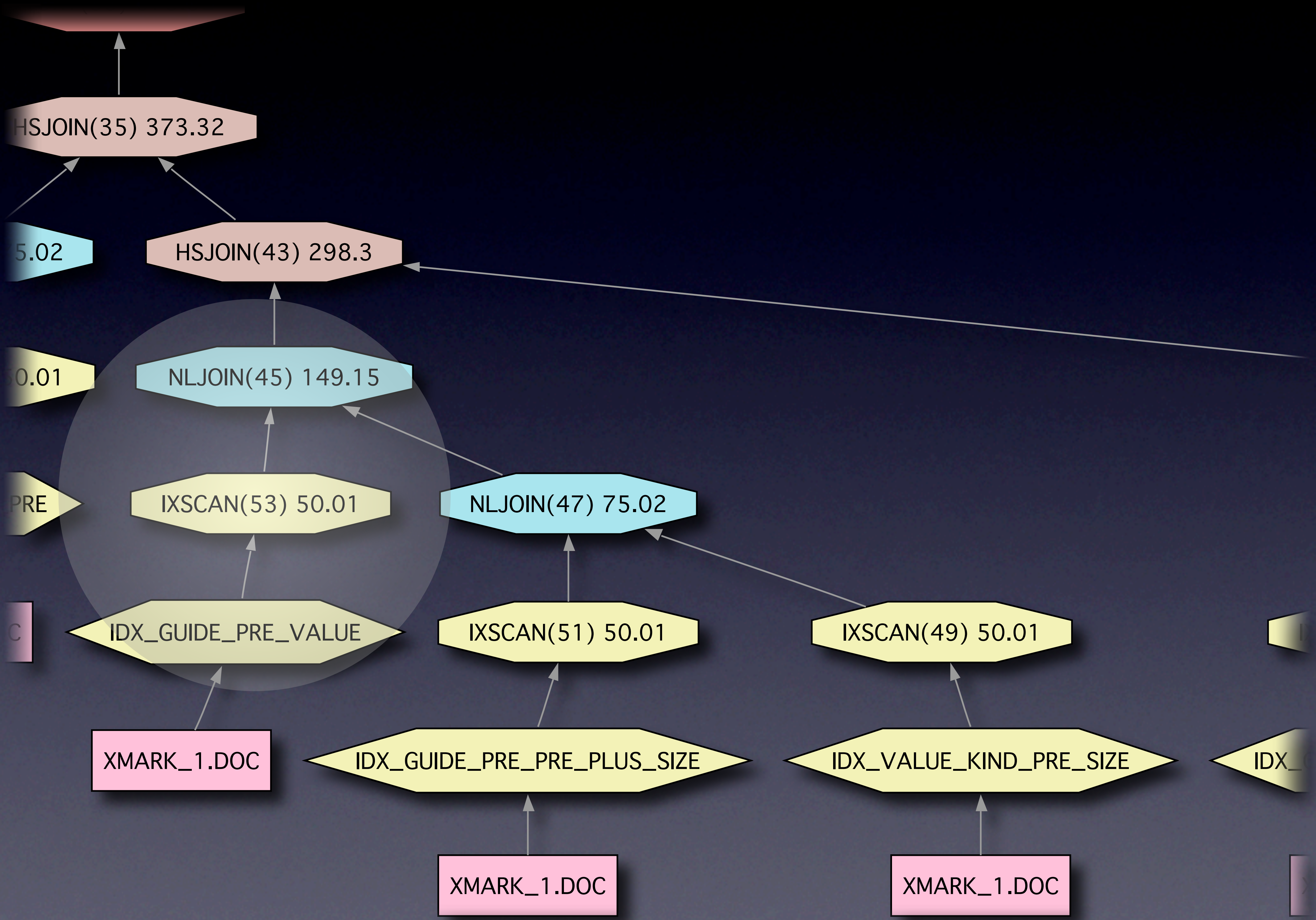
Can DB2® Cope?


HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
XMARK_1.DOC
XMARK_1.DOC

Can DB2® Cope?

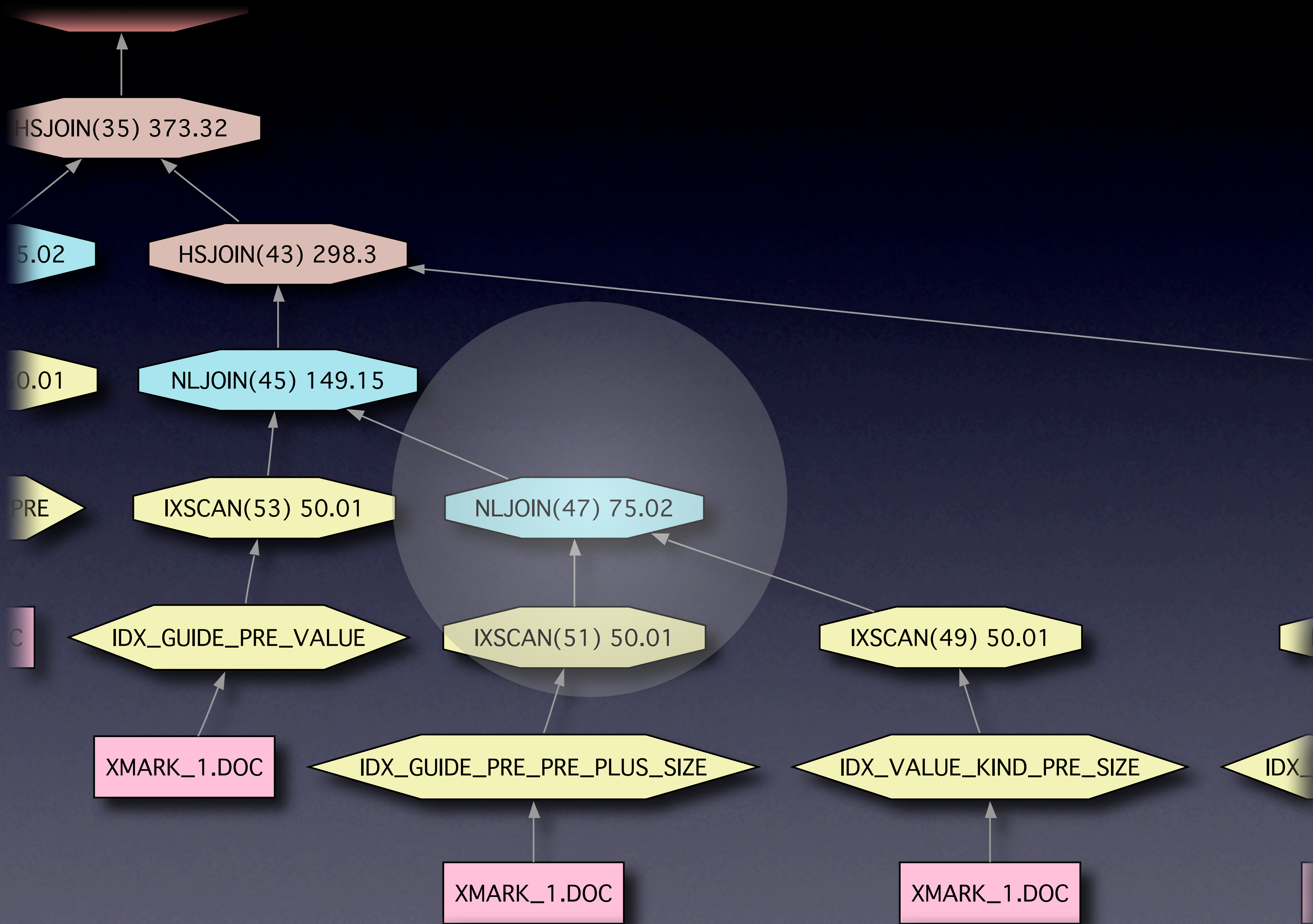

HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
XMARK_1.DOC
XMARK_1.DOC

Can DB2® Cope?

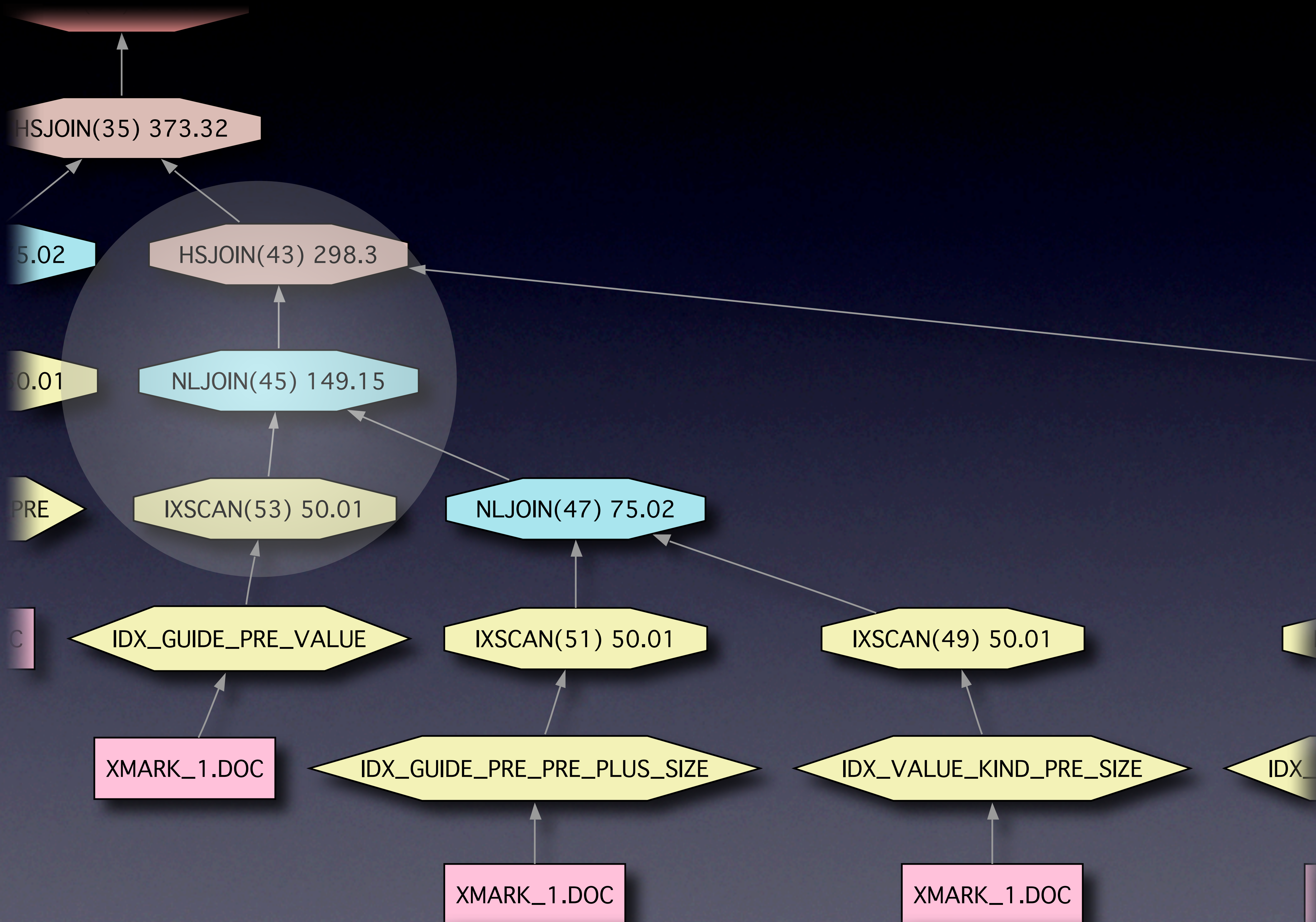


HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
XMARK_1.DOC
XMARK_1.DOC

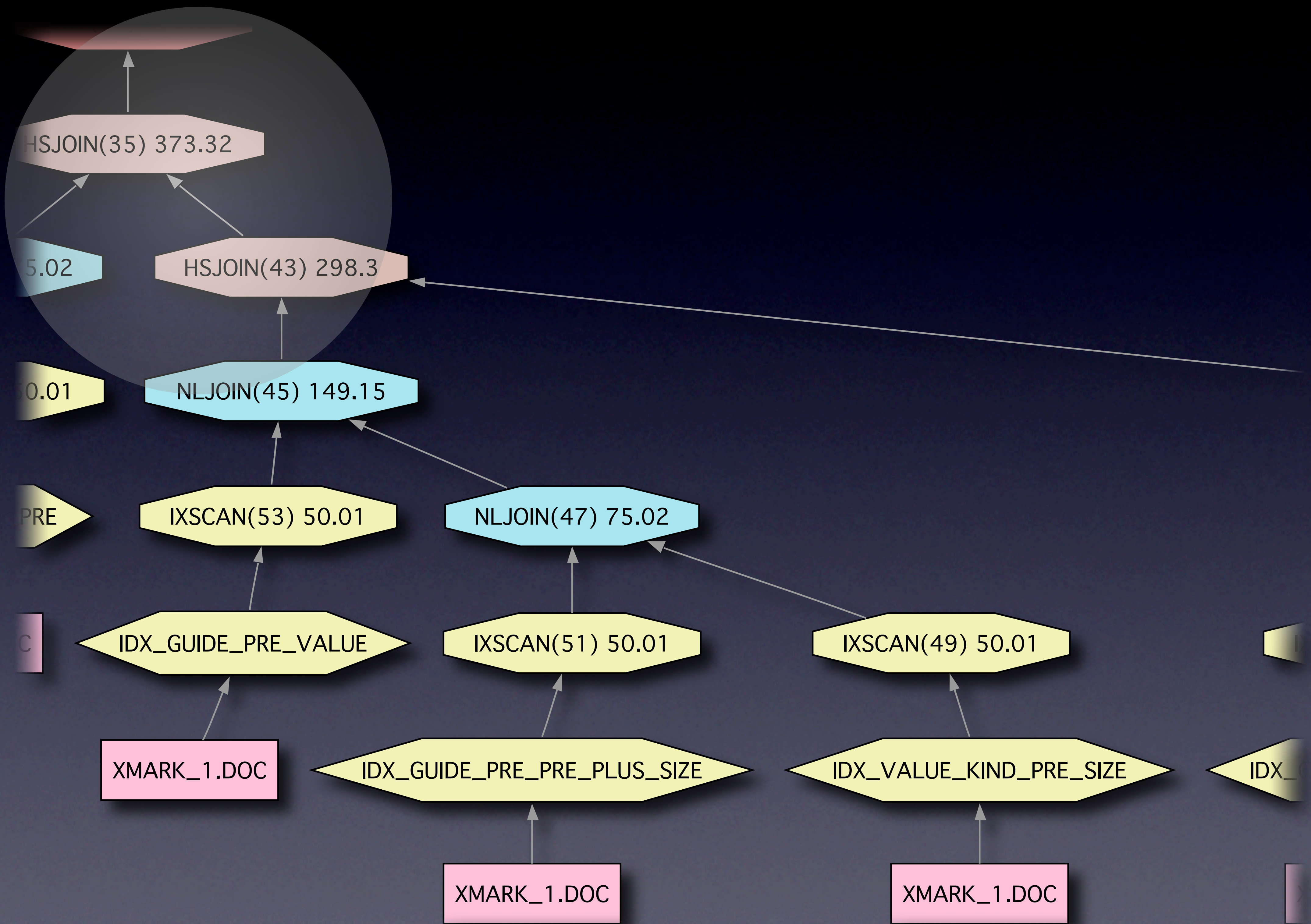
Can DB2® Cope?


HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
XMARK_1.DOC
XMARK_1.DOC

Can DB2® Cope?

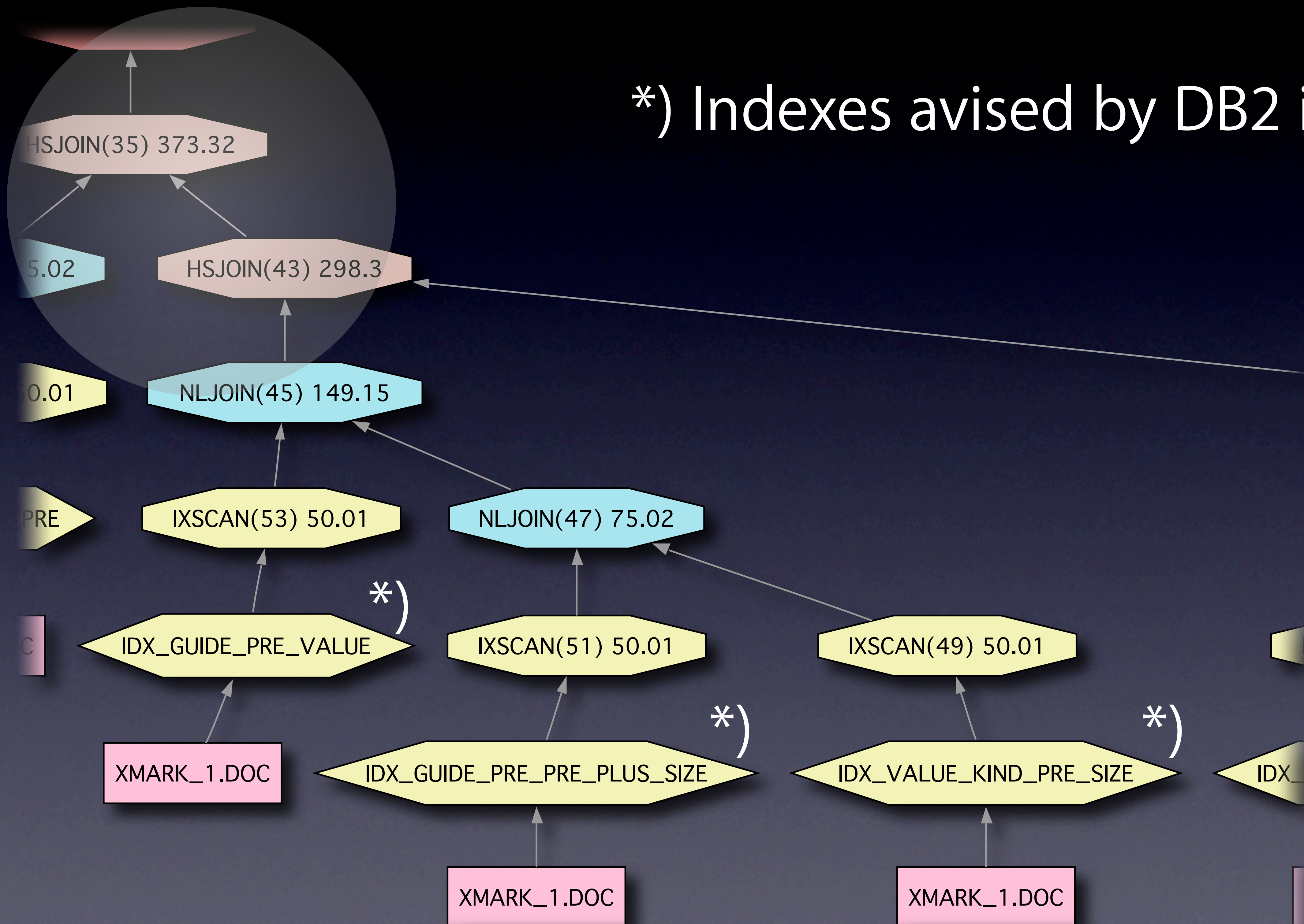

HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
XMARK_1.DOC
XMARK_1.DOC

Can DB2® Cope?



HSJOIN(35) 373.32
5.02
HSJOIN(43) 298.3
0.01
NLJOIN(45) 149.15
PRE
IXSCAN(53) 50.01
NLJOIN(47) 75.02
IDX_GUIDE_PRE_VALUE
IXSCAN(51) 50.01
IXSCAN(49) 50.01
XMARK_1.DOC
IDX_GUIDE_PRE_PRE_PLUS_SIZE
IDX_VALUE_KIND_PRE_SIZE
IDX_
XMARK_1.DOC
XMARK_1.DOC

Can DB2® Cope?



*) Indexes avised by DB2 itself

Order Indifference

```
let $warning := <p>Do <em>not</em> press button,
                   computer will <em>explode!</em></p>
return fn:distinct-doc-order( for $x in $warning//*
                              return $x/text()      )
```

Order Indifference

- Order in XML and XQuery processing is essential:

```
let $warning := <p>Do <em>not</em> press button,
                  computer will <em>explode!</em></p>
return fn:distinct-doc-order( for $x in $warning//*
                              return $x/text()     )
```

Order Indifference

- Order in XML and XQuery processing is essential:

```
let $warning := <p>Do <em>not</em> press button,
                   computer will <em>explode!</em></p>
return fn:distinct-doc-order( for $x in $warning//*
                              return $x/text()      )
```

`Do not press button, computer will explode!`

Order Indifference

- Order in XML and XQuery processing is essential:

```
let $warning := <p>Do <em>not</em> press button,
                   computer will <em>explode!</em></p>
return for $x in $warning//*
       return $x/text()
```

```
Do press button, computer will not explode!
```

Order Indifference

- Order in XML and XQuery processing is essential:

```
let $warning := <p>Do <em>not</em> press button,
                  computer will <em>explode!</em></p>
return for $x in $warning//*
       return $x/text()
```

```
Do press button, computer will not explode!
```

- Order-awareness thus is often **deeply wired** into XQuery processors.
- Supporting constructs like `unordered` { e_1 } is challenging.

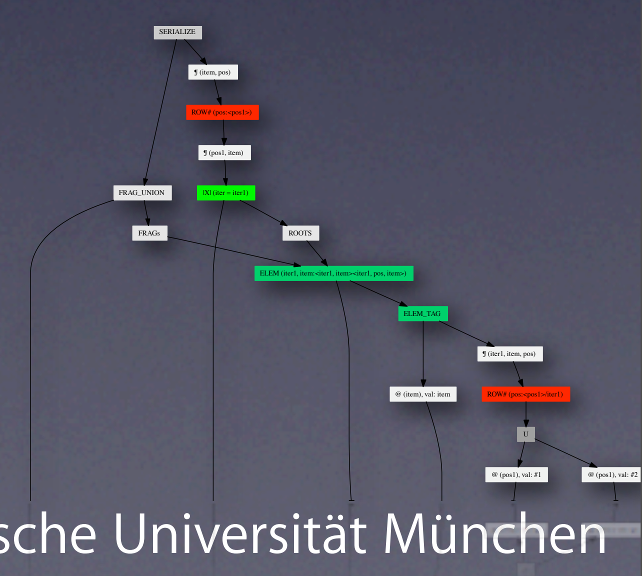
Order Indifference

- Order in XML and XQuery processing is essential:

```
let $warning := <p>Do <em>not</em> press button,
                  computer will <em>explode!</em></p>
return for $x in $warning//*
       return $x/text()
```

```
Do press button, computer will not explode!
```

- Order-awareness thus is often **deeply wired** into XQuery processors.
- Supporting constructs like `unordered` { e_1 } is challenging.



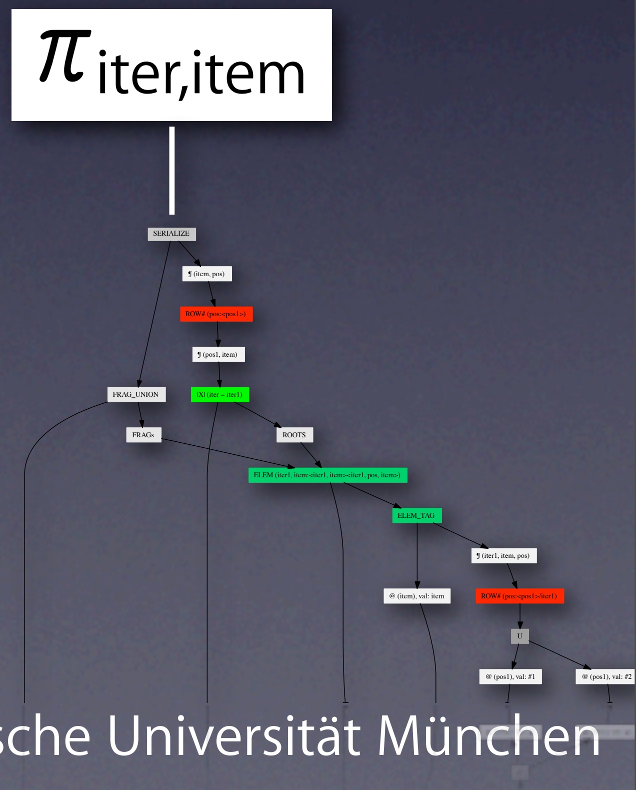
Order Indifference

- Order in XML and XQuery processing is essential:

```
let $warning := <p>Do <em>not</em> press button,
                  computer will <em>explode!</em></p>
return for $x in $warning//*
       return $x/text()
```

```
Do press button, computer will not explode!
```

- Order-awareness thus is often **deeply wired** into XQuery processors.
- Supporting constructs like `unordered` { e_1 } is challenging.



More Purely Relational XQuery

More Purely Relational XQuery

- Declarative XQuery debugging with “time travel”.
  - Users observe expressions and may traverse iterations forward/backward.

More Purely Relational XQuery

- Declarative XQuery debugging with “time travel”.
  - Users observe expressions and may traverse iterations forward/backward.

iter	pos	item
1		
2		
3		
4		
⋮		

More Purely Relational XQuery

- Declarative XQuery debugging with “time travel”.
  - Users observe expressions and may traverse iterations forward/backward.

iter	pos	item
1		
2		
3		
4		
⋮		

- Dependable cardinality estimates at XQuery subexpression level (# items per iteration and overall contribution).

```
for $x in $doc//x
return if (e1) then e2 else e3
```

More Purely Relational XQuery

- Declarative XQuery debugging with “time travel”.
  - Users observe expressions and may traverse iterations forward/backward.

iter	pos	item
1		
2		
3		
4		
⋮		

- Dependable cardinality estimates at XQuery subexpression level (# items per iteration and overall contribution).

for \$x in \$doc//x

return $\overbrace{\texttt{if } (e_1) \texttt{ then } \underbrace{e_2}_{8} \texttt{ else } \underbrace{e_3}_{2}}^{4.3}$

Atomization Indexes

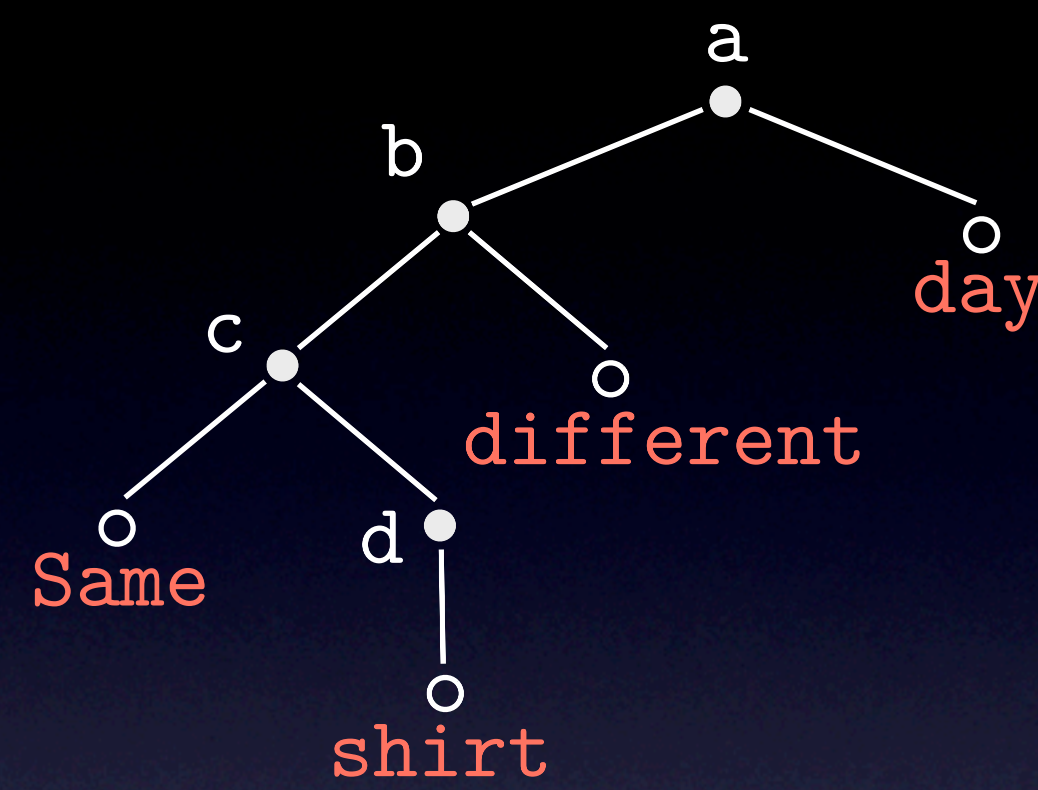
```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

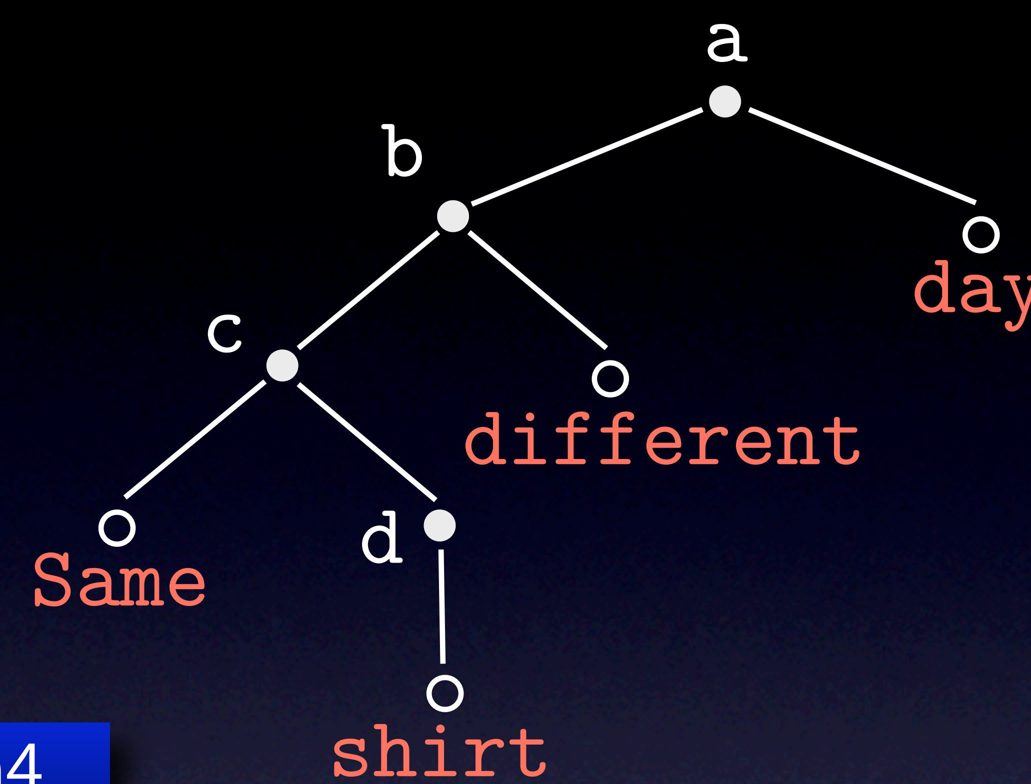
Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



Atomization Indexes

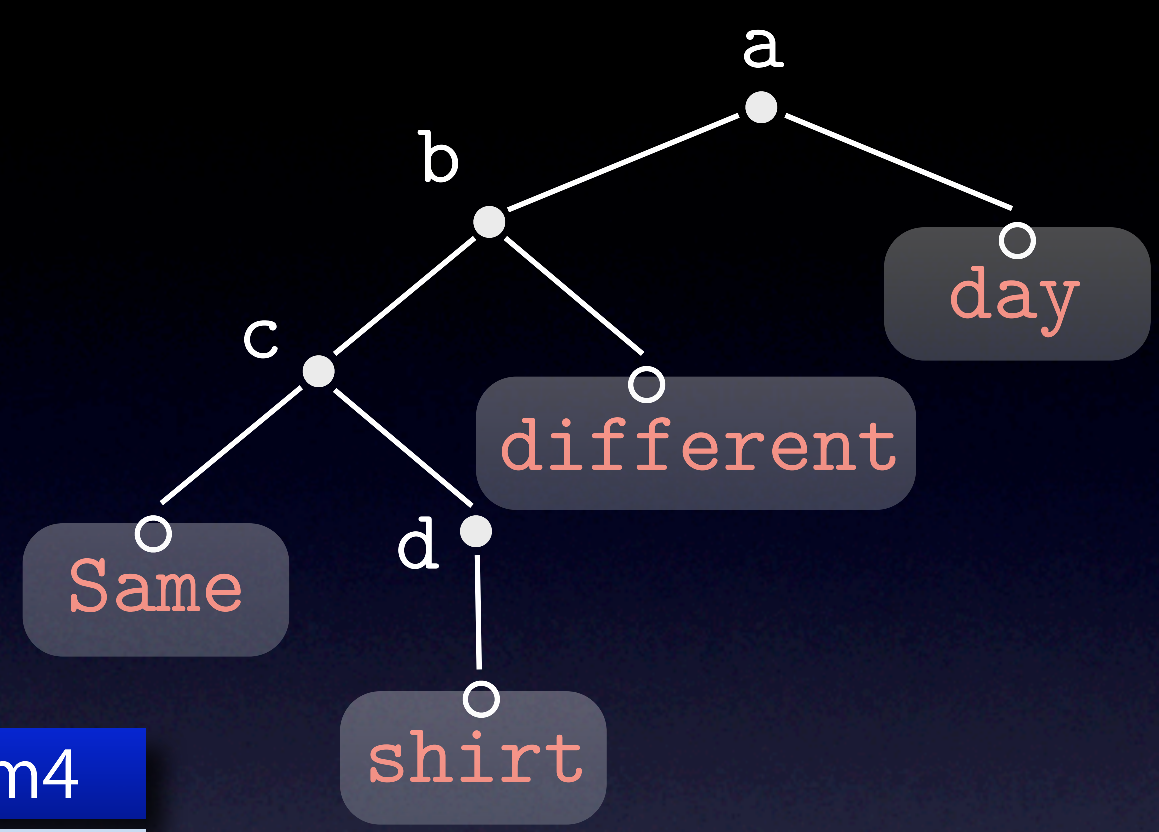
```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



node	atom1	atom2	atom3	atom4
a				
b				
c				
○				
d				
○				
○				
○				

Atomization Indexes

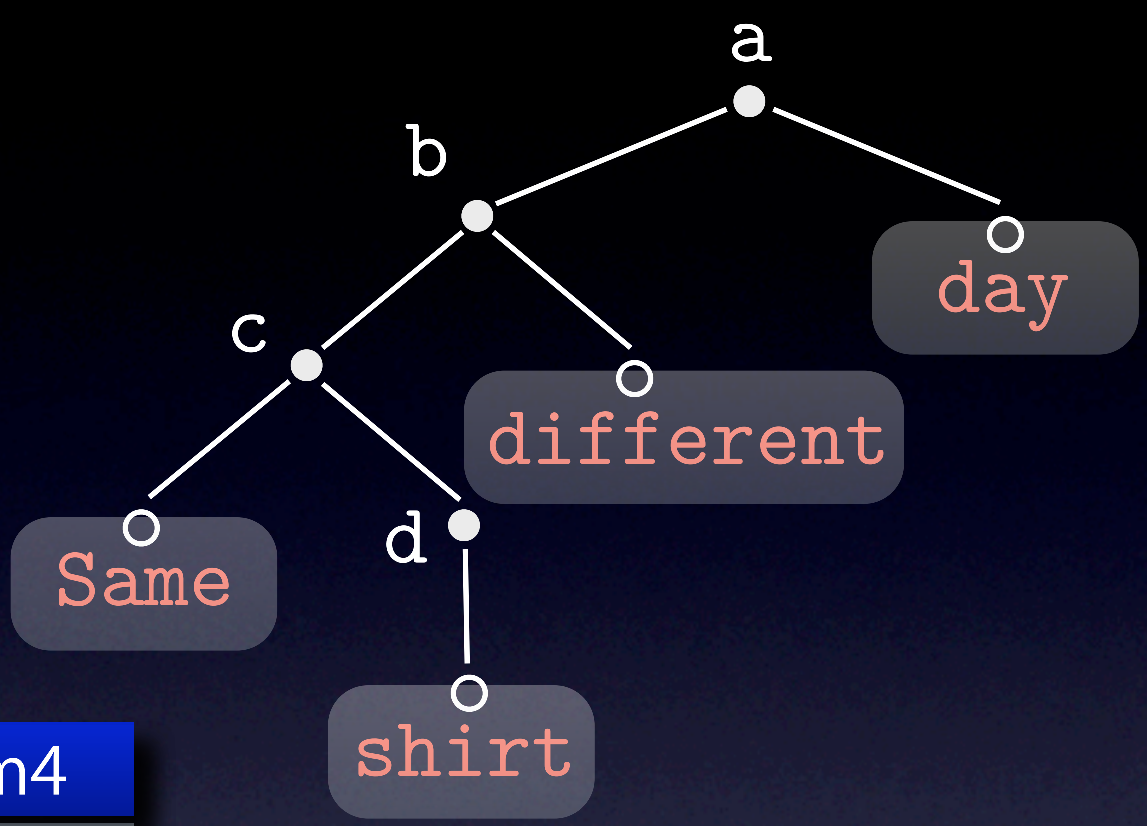
```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



node	atom1	atom2	atom3	atom4
a				
b				
c				
o				
d				
o				
o				
o				

Atomization Indexes

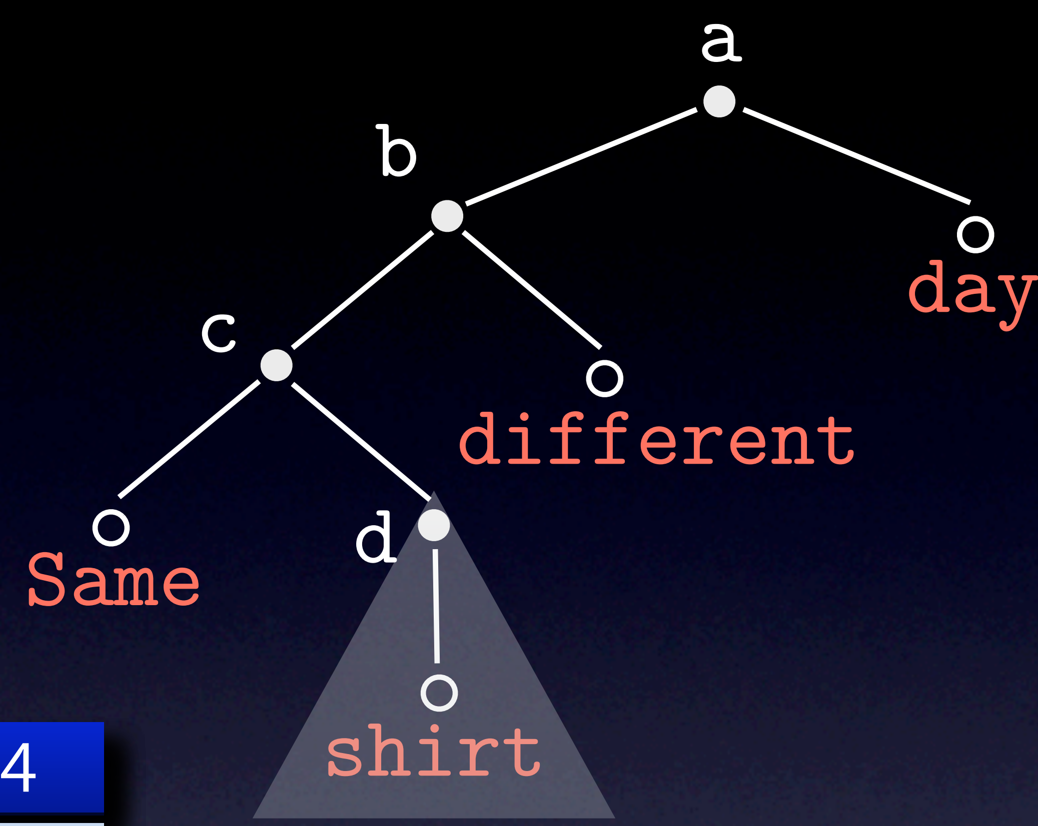
```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



node	atom1	atom2	atom3	atom4
a				
b				
c				
o	Same			
d				
o	shirt			
o	different			
o	day			

Atomization Indexes

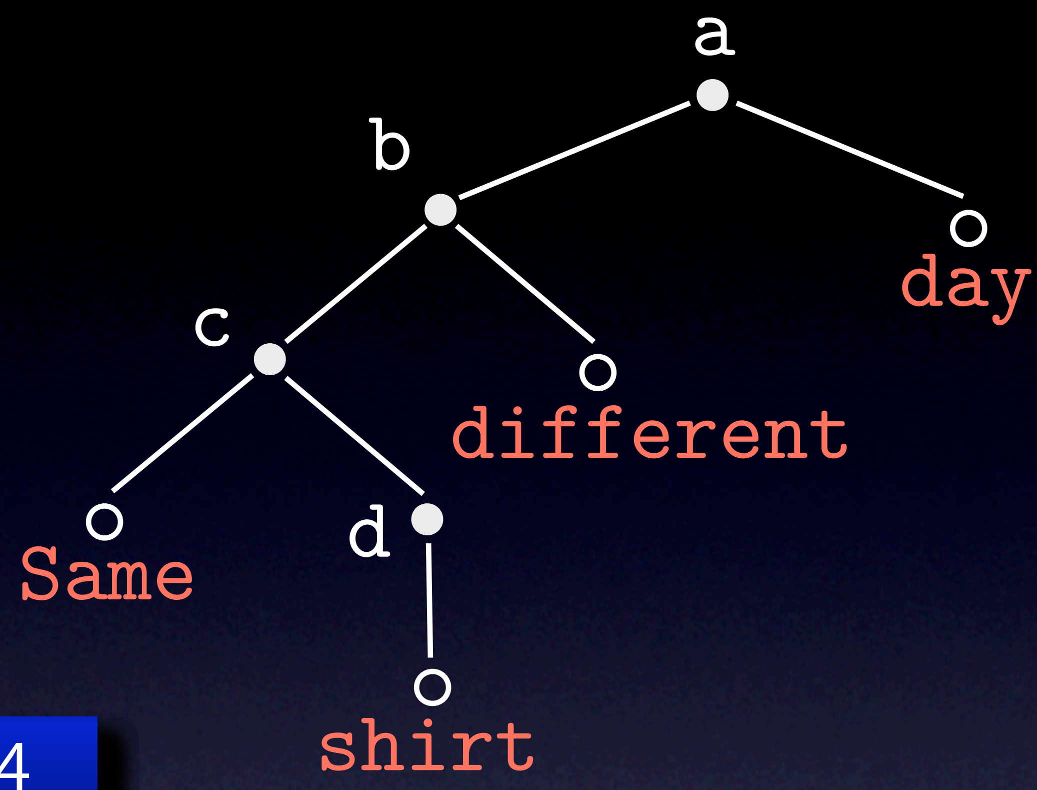
```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



node	atom1	atom2	atom3	atom4
a				
b				
c				
○	Same			
d	shirt			
○	shirt			
○	different			
○	day			

Atomization Indexes

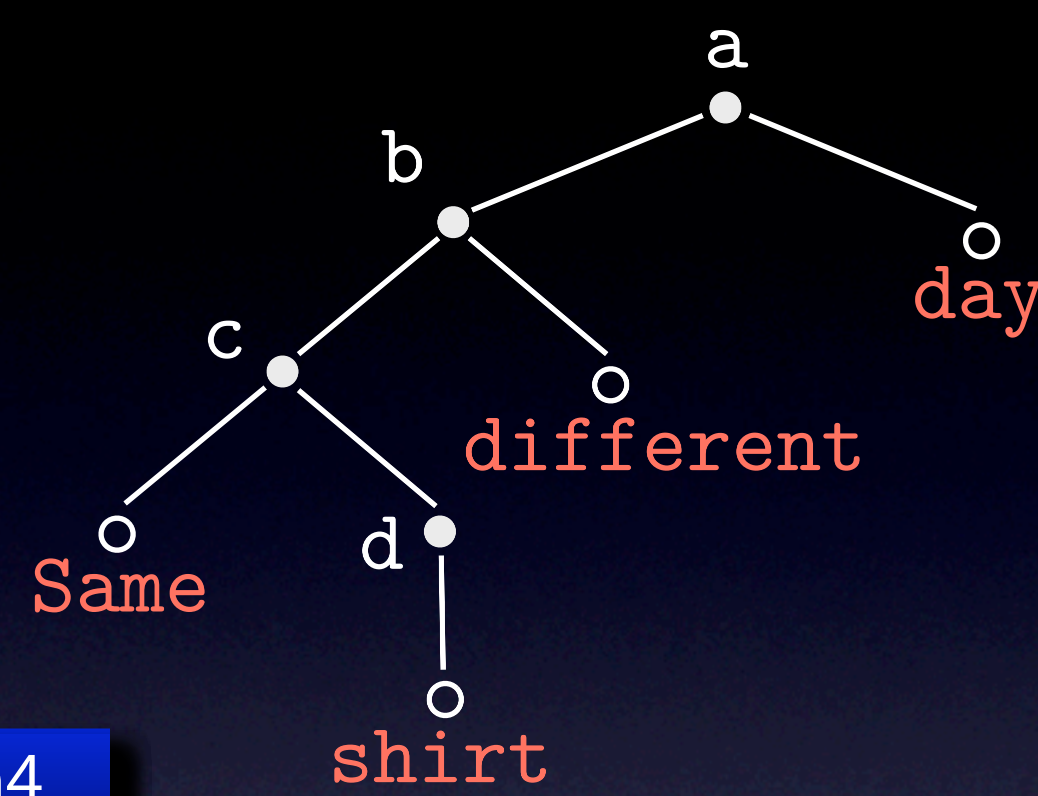
```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



node	atom1	atom2	atom3	atom4
a	Same	shirt	different	day
b	Same	shirt	different	
c	Same	shirt		
o	Same			
d	shirt			
o	shirt			
o	different			
o	day			

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

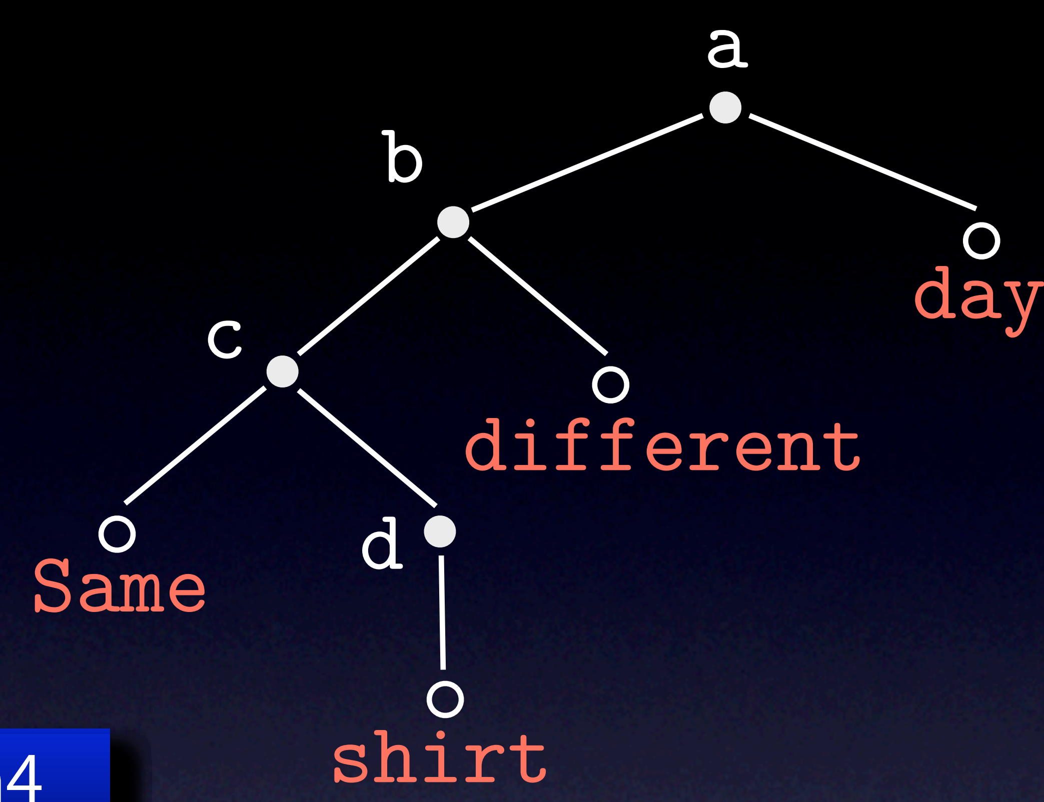


node	atom1	atom2	atom3	atom4
a	Same	shirt	different	day
b	Same	shirt	different	
c	Same	shirt		
○	Same			
d	shirt			
○	shirt			
○	different			
○	day			

dict	atom

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

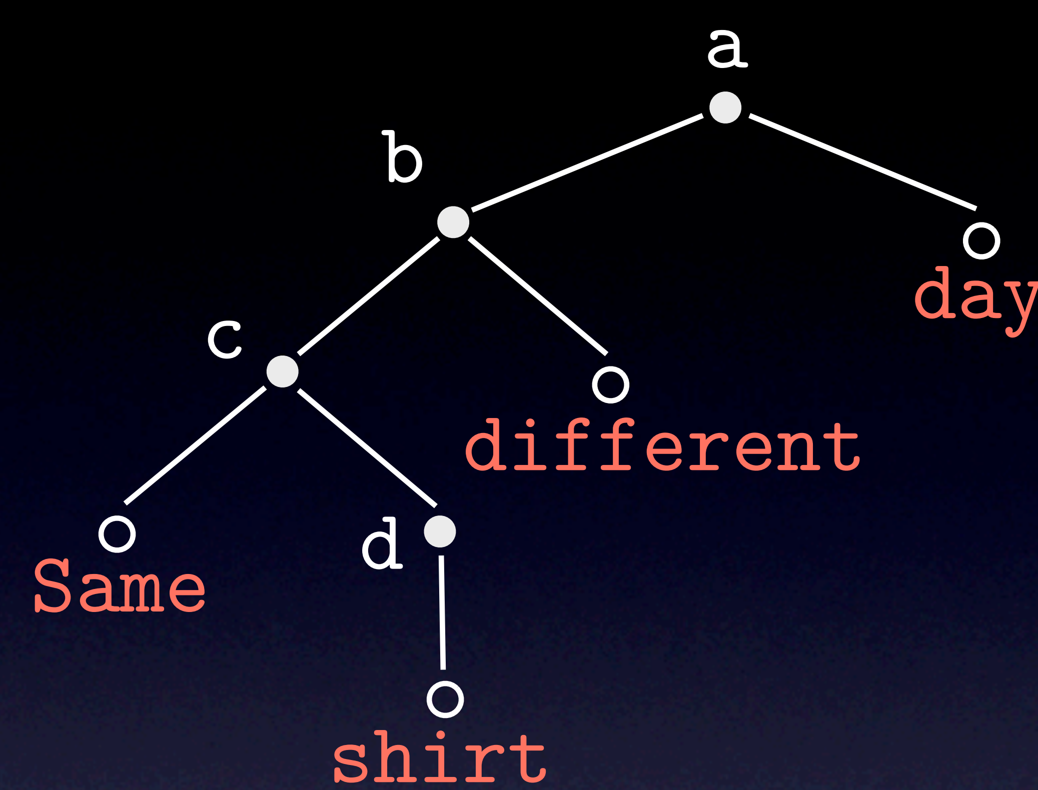


node	atom1	atom2	atom3	atom4
a	δ_1	shirt	different	day
b	δ_1	shirt	different	
c	δ_1	shirt		
○	δ_1			
d	shirt			
○	shirt			
○	different			
○	day			

dict	atom
δ_1	Same

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

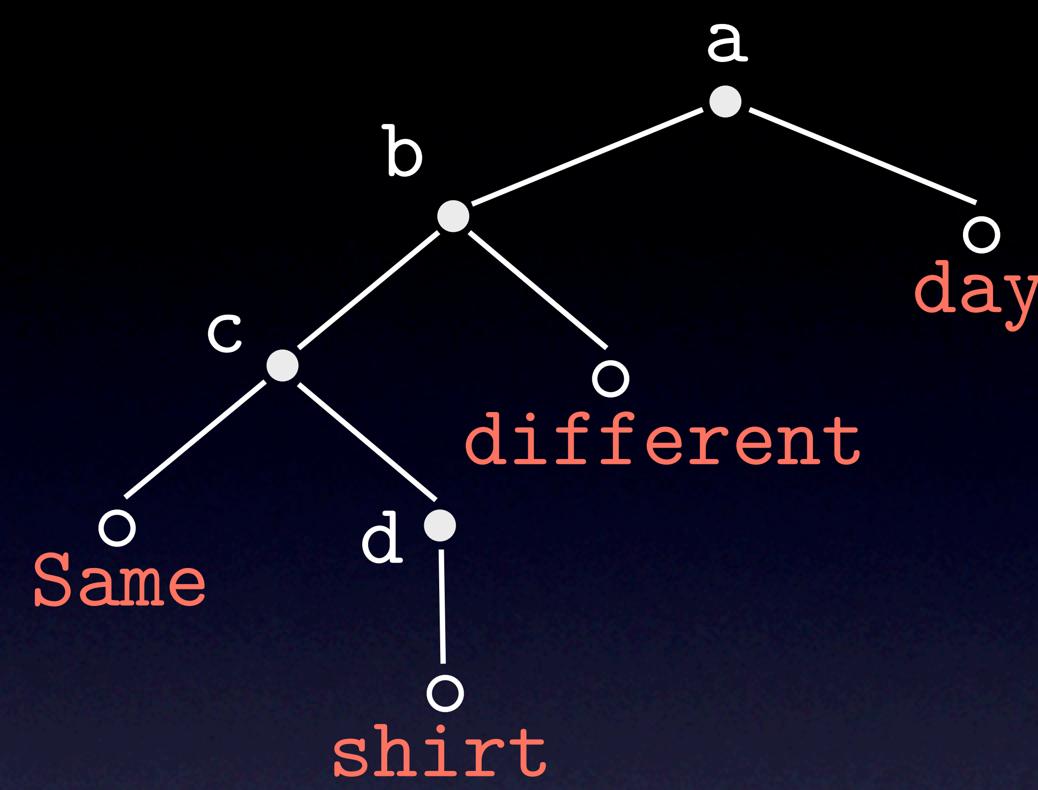


node	atom1	atom2	atom3	atom4
a	δ_1	δ_2	δ_3	δ_4
b	δ_1	δ_2	δ_3	
c	δ_1	δ_2		
o	δ_1			
d	δ_2			
o	δ_2			
o	δ_3			
o	δ_4			

dict	atom
δ_1	Same
δ_2	shirt
δ_3	different
δ_4	day

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

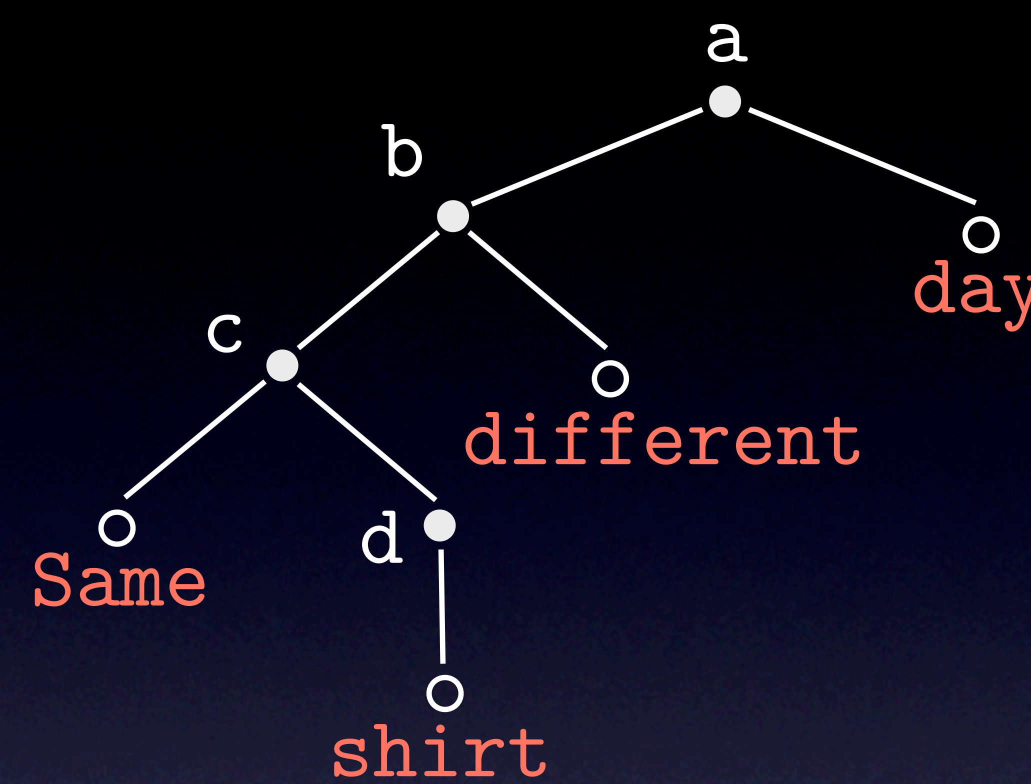


node	atom1	atom2	atom3	atom4
a	δ_5		δ_3	δ_4
b	δ_5		δ_3	
c	δ_5			
o	δ_1			
d	δ_2			
o	δ_2			
o	δ_3			
o	δ_4			

dict	atom	
δ_1	Same	
δ_2	shirt	
δ_3	different	
δ_4	day	
δ_5	Same	shirt

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```

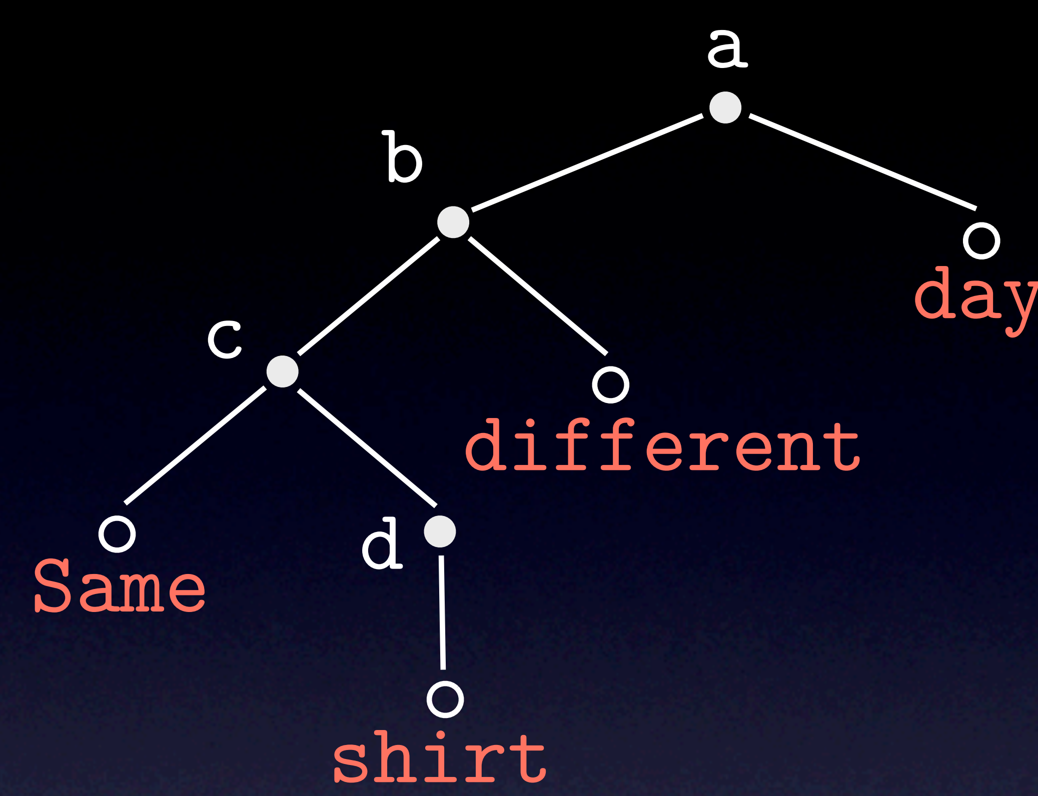


node	atom1	atom2	atom3	atom4
a	δ_6			δ_4
b	δ_6			
c	δ_5			
o	δ_1			
d	δ_2			
o	δ_2			
o	δ_3			
o	δ_4			

dict	atom	
δ_1	Same	
δ_2	shirt	
δ_3	different	
δ_4	day	
δ_5	δ_1	δ_2
δ_6	δ_5	δ_3

Atomization Indexes

```
<a>
 <b>
  <c>Same<d>shirt</d></c>
  different
 </b>
 day
</a>
```



node	atom1	atom2
a	δ_6	δ_4
b	δ_6	
c	δ_5	
o	δ_1	
d	δ_2	
o	δ_2	
o	δ_3	
o	δ_4	

dict	atom	
δ_1	Same	
δ_2	shirt	
δ_3	different	
δ_4	day	
δ_5	δ_1	δ_2
δ_6	δ_5	δ_3

Relational Encodings for XML

- XML documents represent ordered, unranked trees (nodes of 7 kinds).
- Relational XQuery processing calls for an adequate relational encoding of such trees:

1. Schema-oblivious
   (XQuery constructs arbitrary XML fragments),
2. accessible for the query processor and index support
   ($\gg 10^7$ nodes in typical GB-range XML instances).

Relational Encodings for XML

- XML documents represent ordered, unranked trees (nodes of 7 kinds).
- Relational XQuery processing calls for an adequate relational encoding of such trees:

1. Schema-oblivious (XQuery constructs arbitrary XML fragments),
2. accessible for the query processor and index support ($\gg 10^7$ nodes in typical GB-range XML instances).

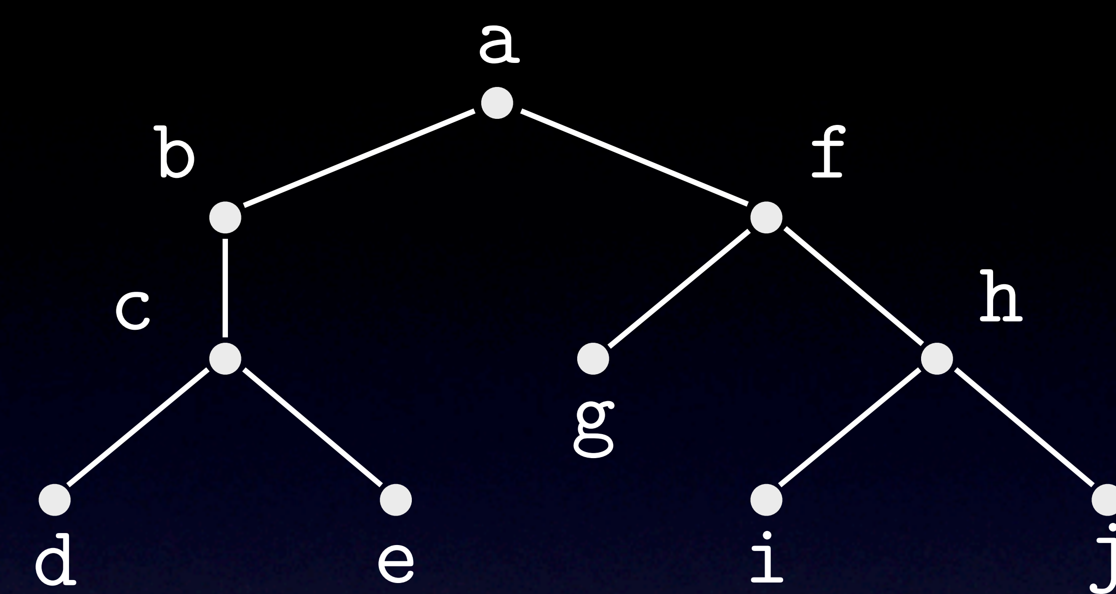
doc	X
1	"<a>b<c/>d<e/><f>g</f>"
2	...

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```

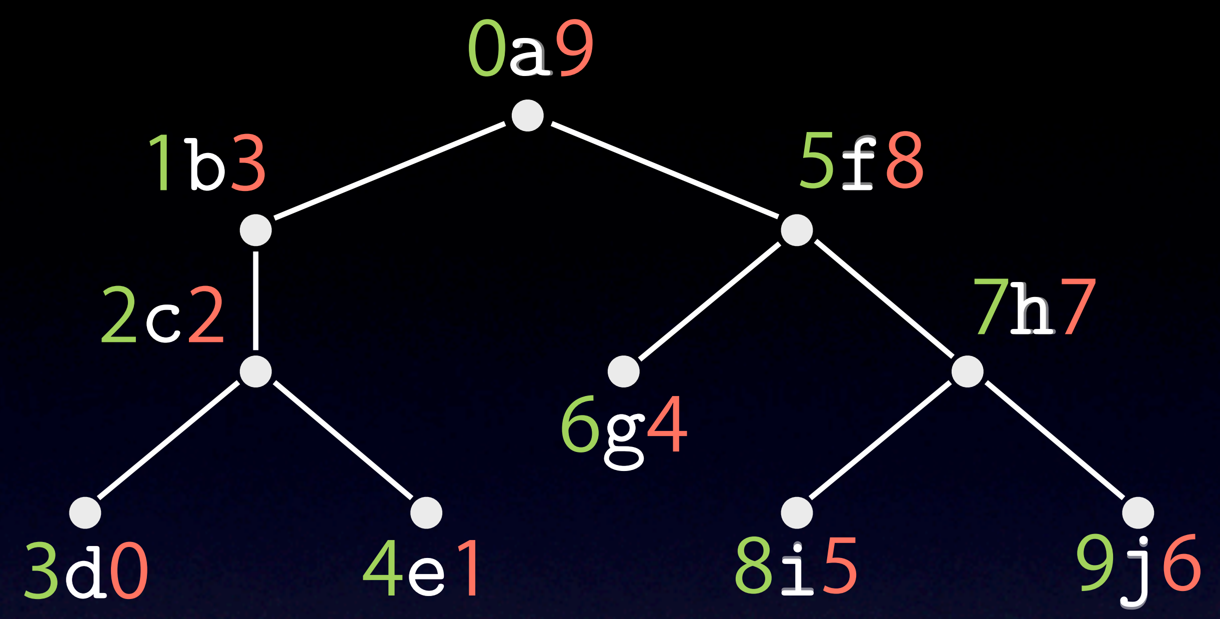
Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```



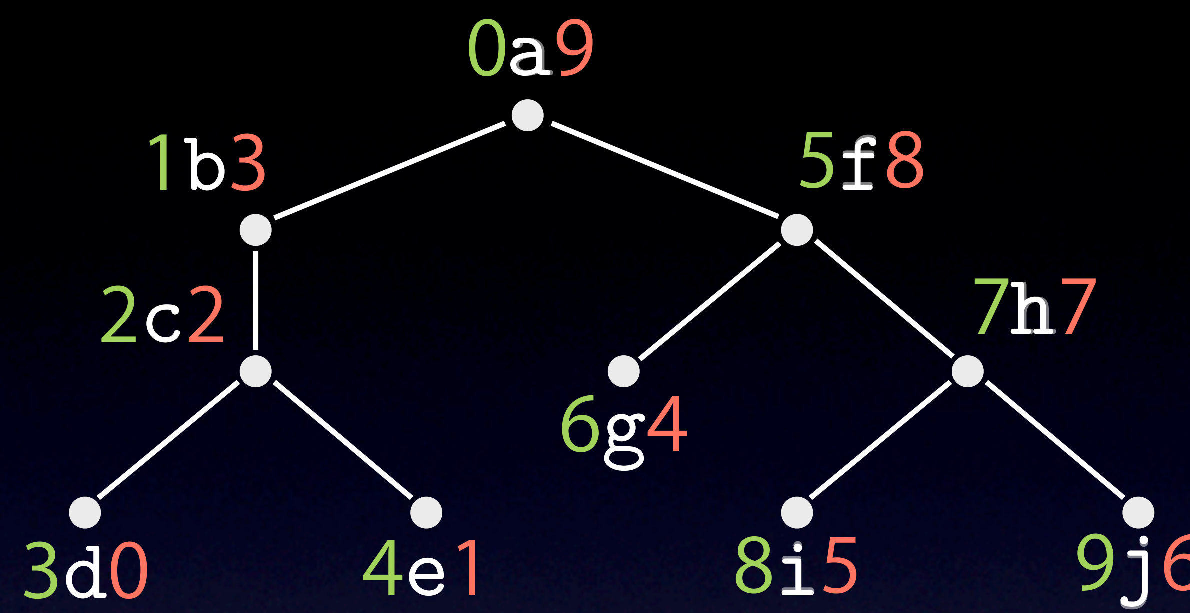
Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```



Pre/Postorder Encoding

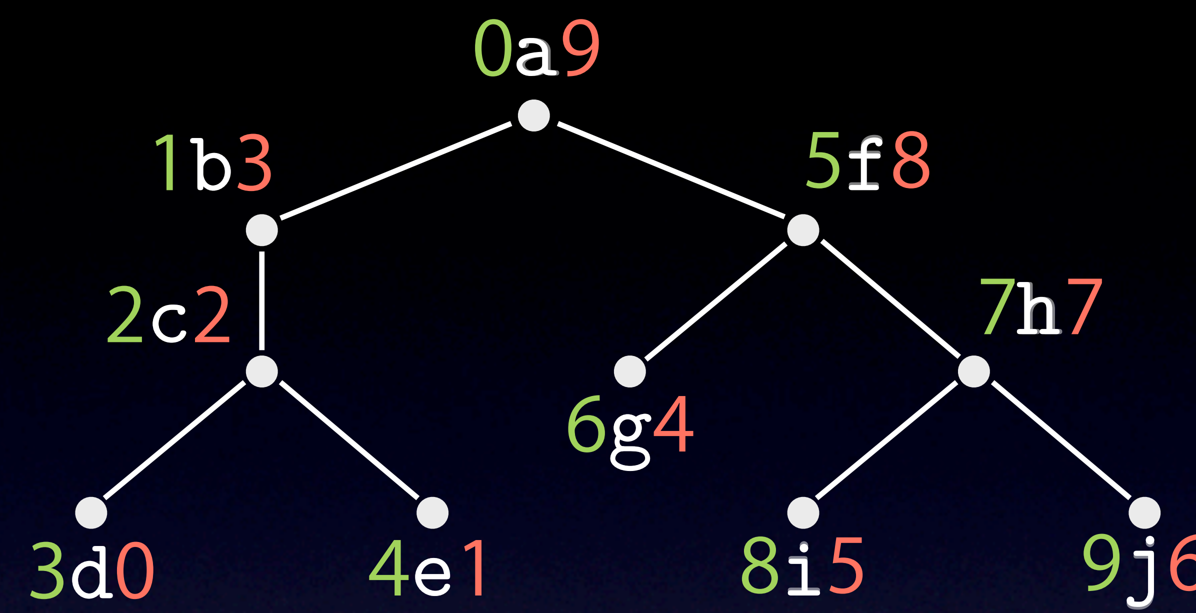
```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```



pre	post	node
0	9	a
1	3	b
2	2	c
3	0	d
4	1	e
5	8	f
6	4	g
7	7	h
8	5	i
9	6	j

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```

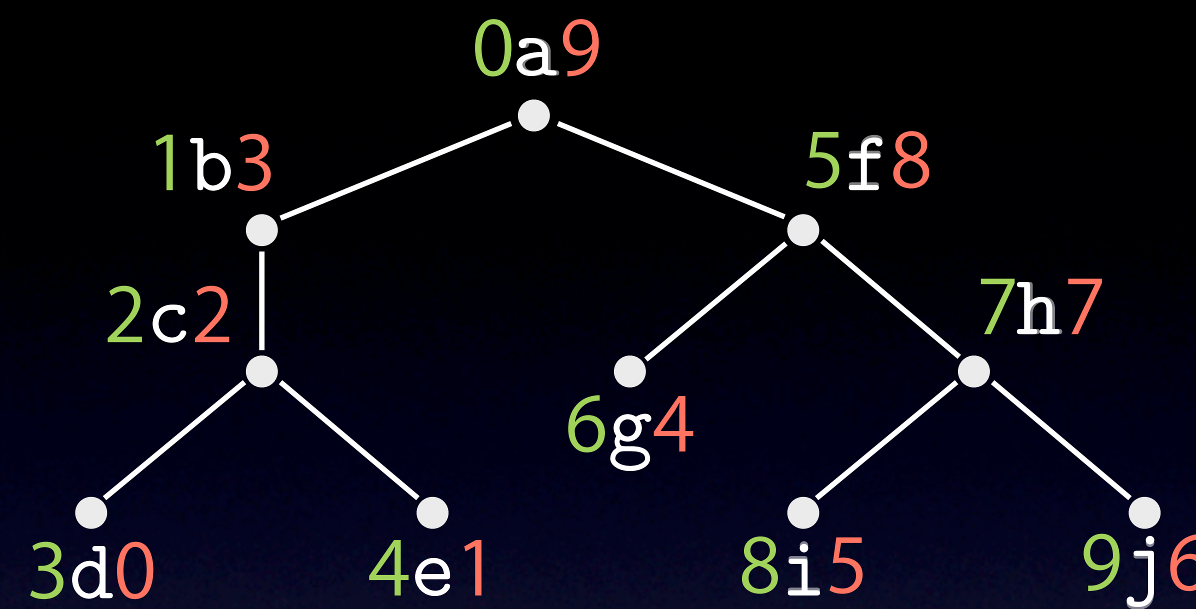


document order

pre	post	node
0	9	a
1	3	b
2	2	c
3	0	d
4	1	e
5	8	f
6	4	g
7	7	h
8	5	i
9	6	j

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```

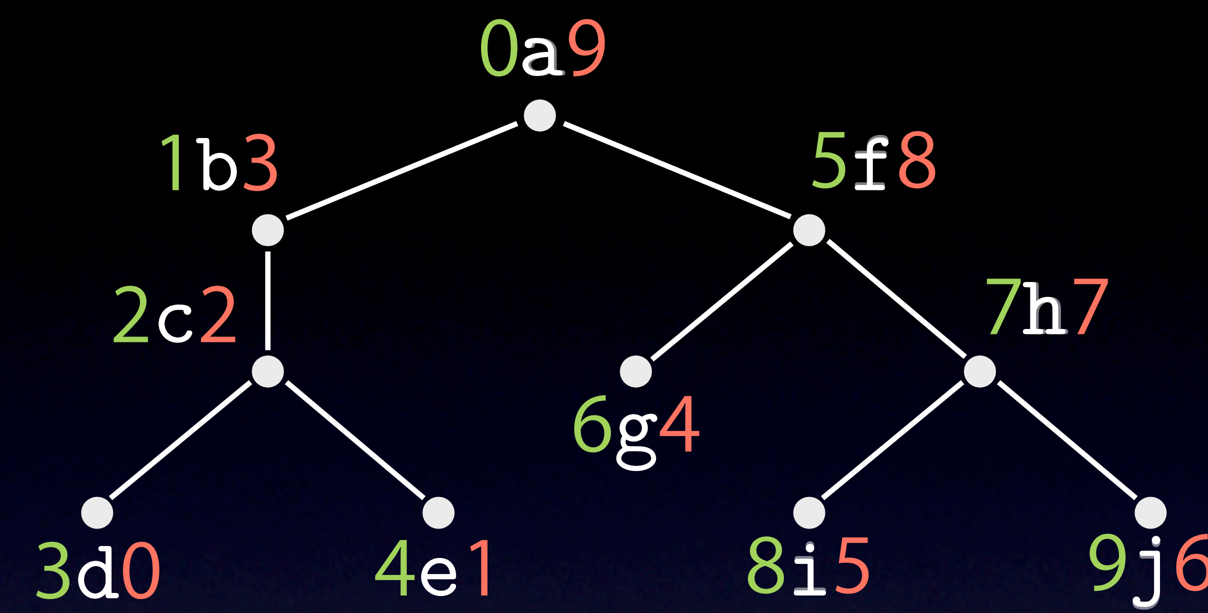


document order

pre	post	node	kind
0	9	a	elem
1	3	b	elem
2	2	c	elem
3	0	d	elem
4	1	e	text
5	8	f	elem
6	4	g	comm
7	7	h	elem
8	5	i	elem
9	6	j	elem

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```

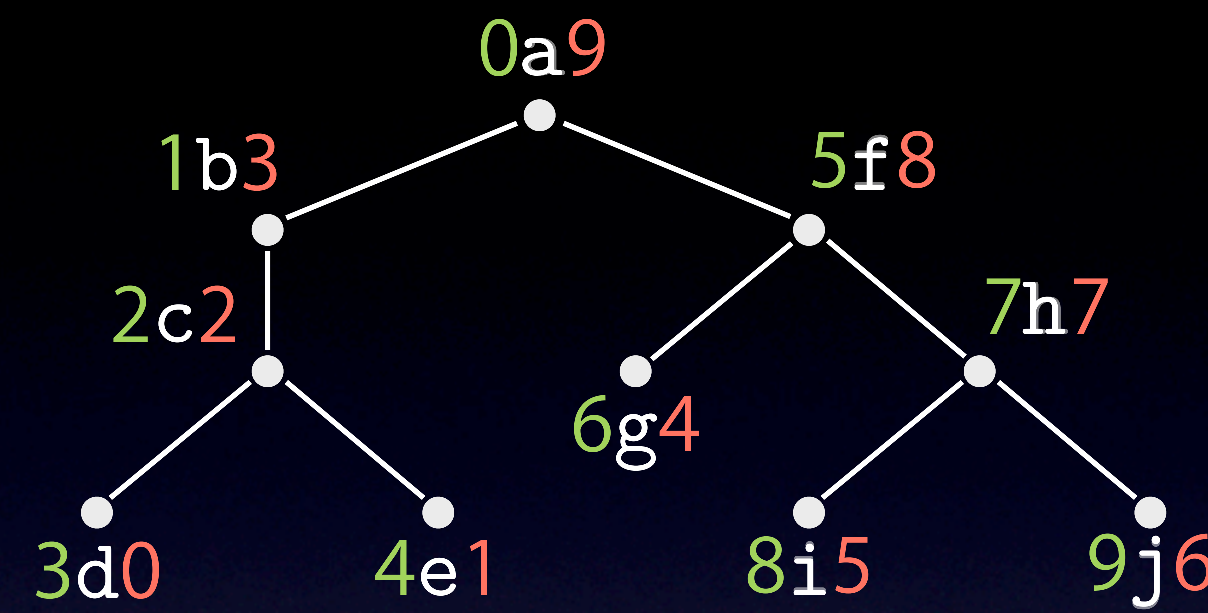


document order

pre	post	node	kind	size	level
0	9	a	elem	9	0
1	3	b	elem	3	1
2	2	c	elem	2	2
3	0	d	elem	0	3
4	1	e	text	0	3
5	8	f	elem	4	1
6	4	g	comm	0	2
7	7	h	elem	2	2
8	5	i	elem	0	3
9	6	j	elem	0	3

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```

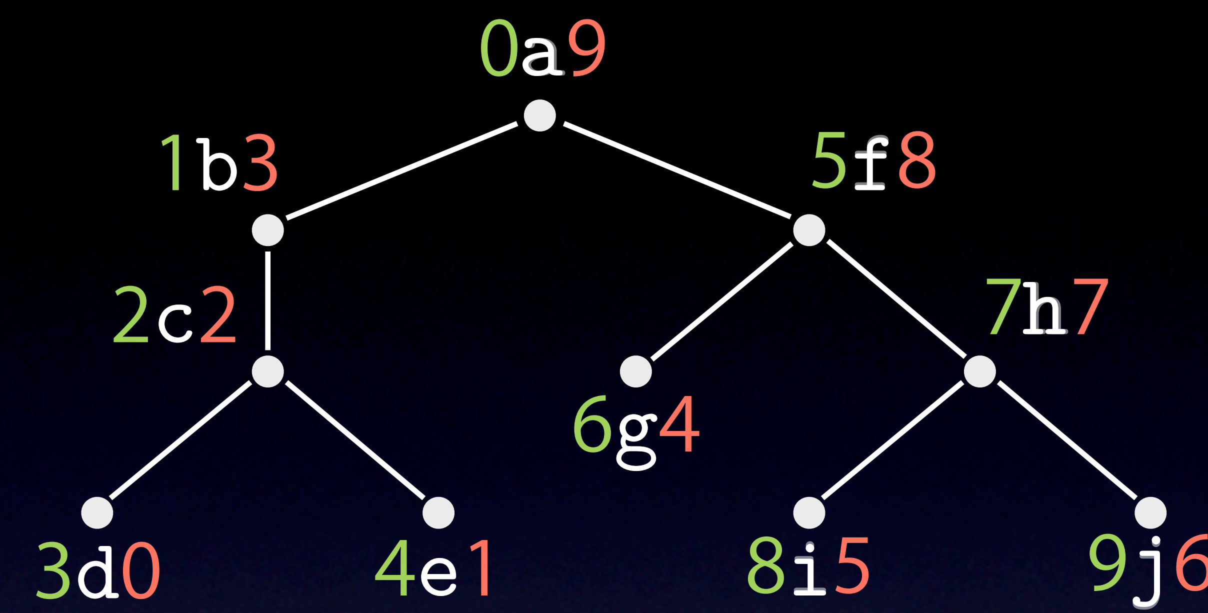


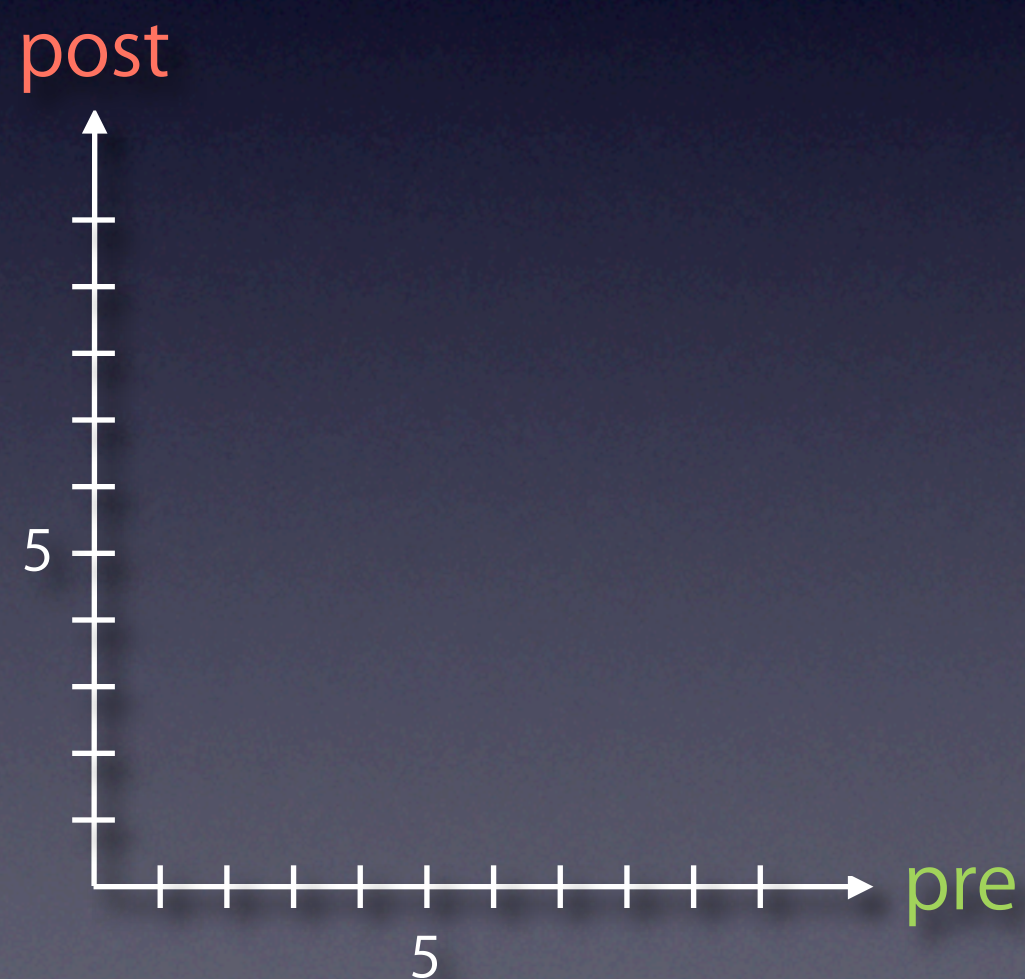
document order

pre	node	kind	size	level
0	a	elem	9	0
1	b	elem	3	1
2	c	elem	2	2
3	d	elem	0	3
4	e	text	0	3
5	f	elem	4	1
6	g	comm	0	2
7	h	elem	2	2
8	i	elem	0	3
9	j	elem	0	3

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```



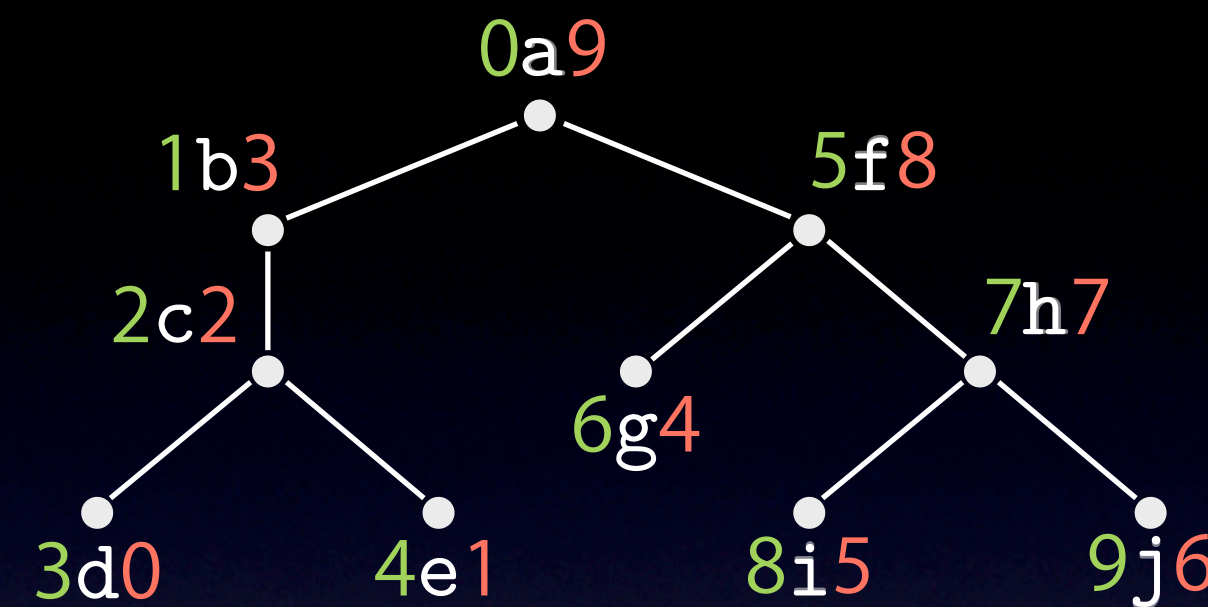


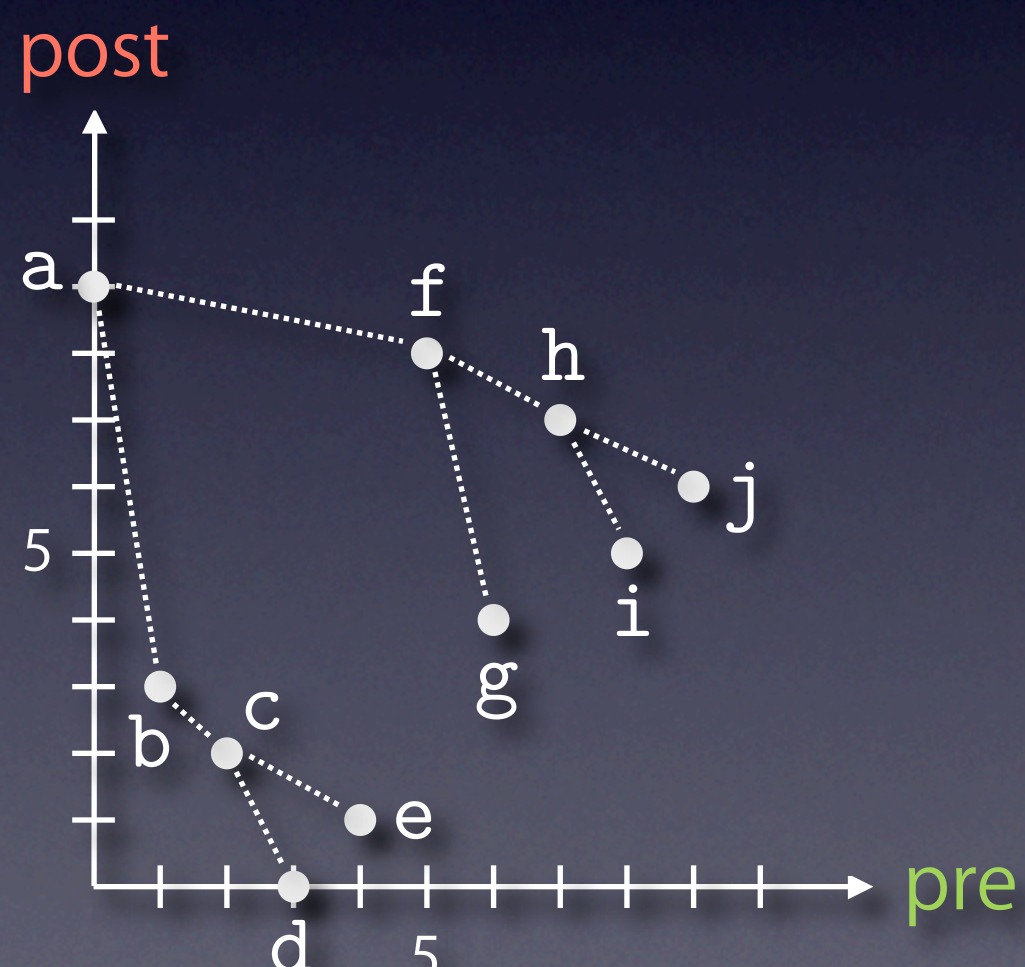
document order

pre	node	kind	size	level
0	a	elem	9	0
1	b	elem	3	1
2	c	elem	2	2
3	d	elem	0	3
4	e	text	0	3
5	f	elem	4	1
6	g	comm	0	2
7	h	elem	2	2
8	i	elem	0	3
9	j	elem	0	3

Pre/Postorder Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```





document order

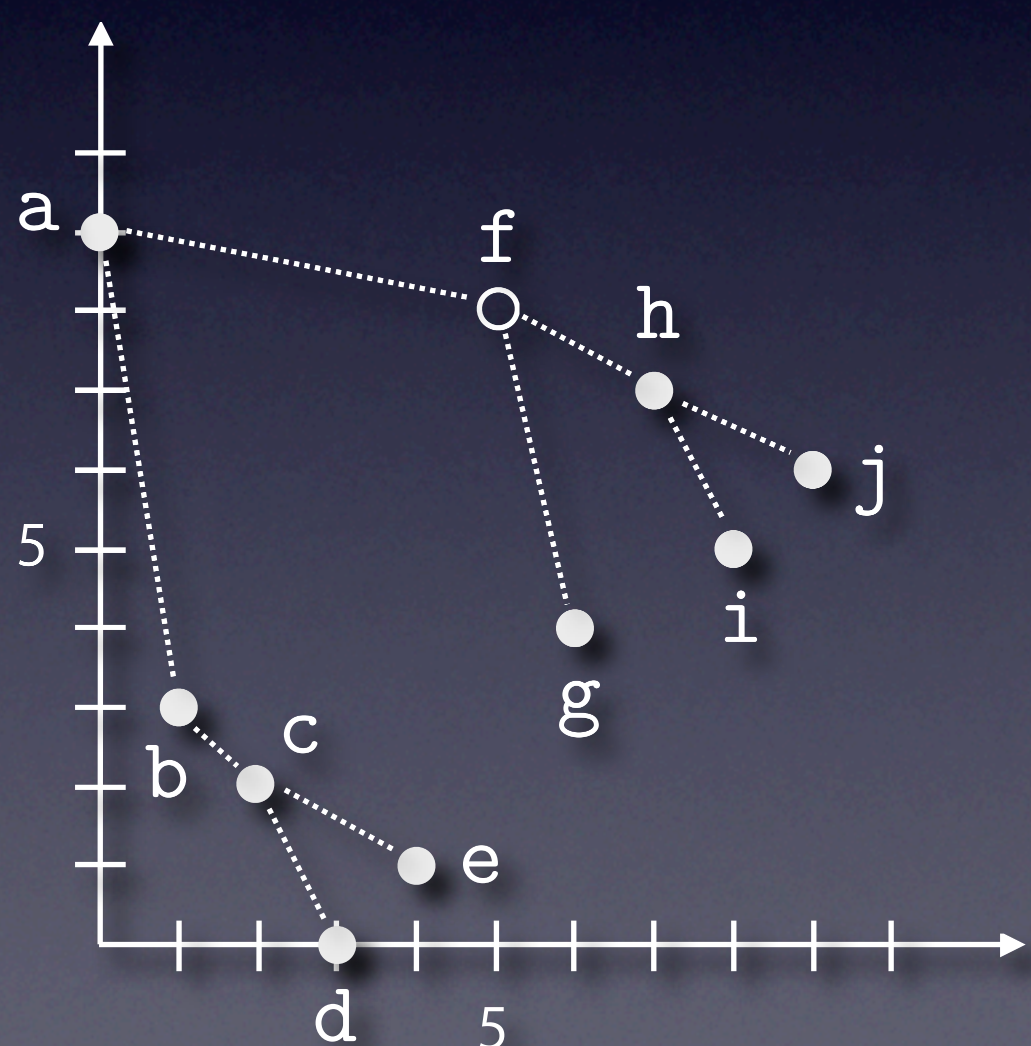
pre	node	kind	size	level
0	a	elem	9	0
1	b	elem	3	1
2	c	elem	2	2
3	d	elem	0	3
4	e	text	0	3
5	f	elem	4	1
6	g	comm	0	2
7	h	elem	2	2
8	i	elem	0	3
9	j	elem	0	3

B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:

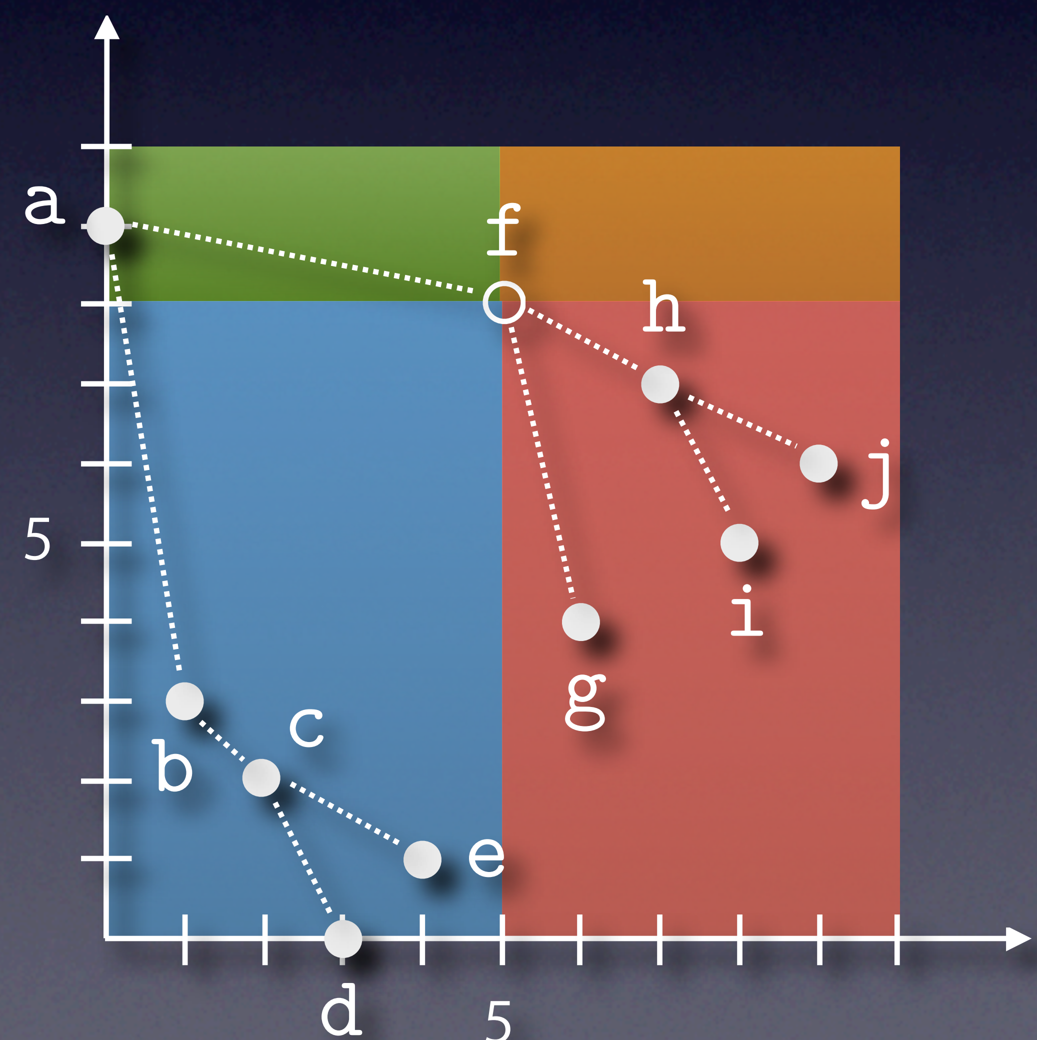
B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



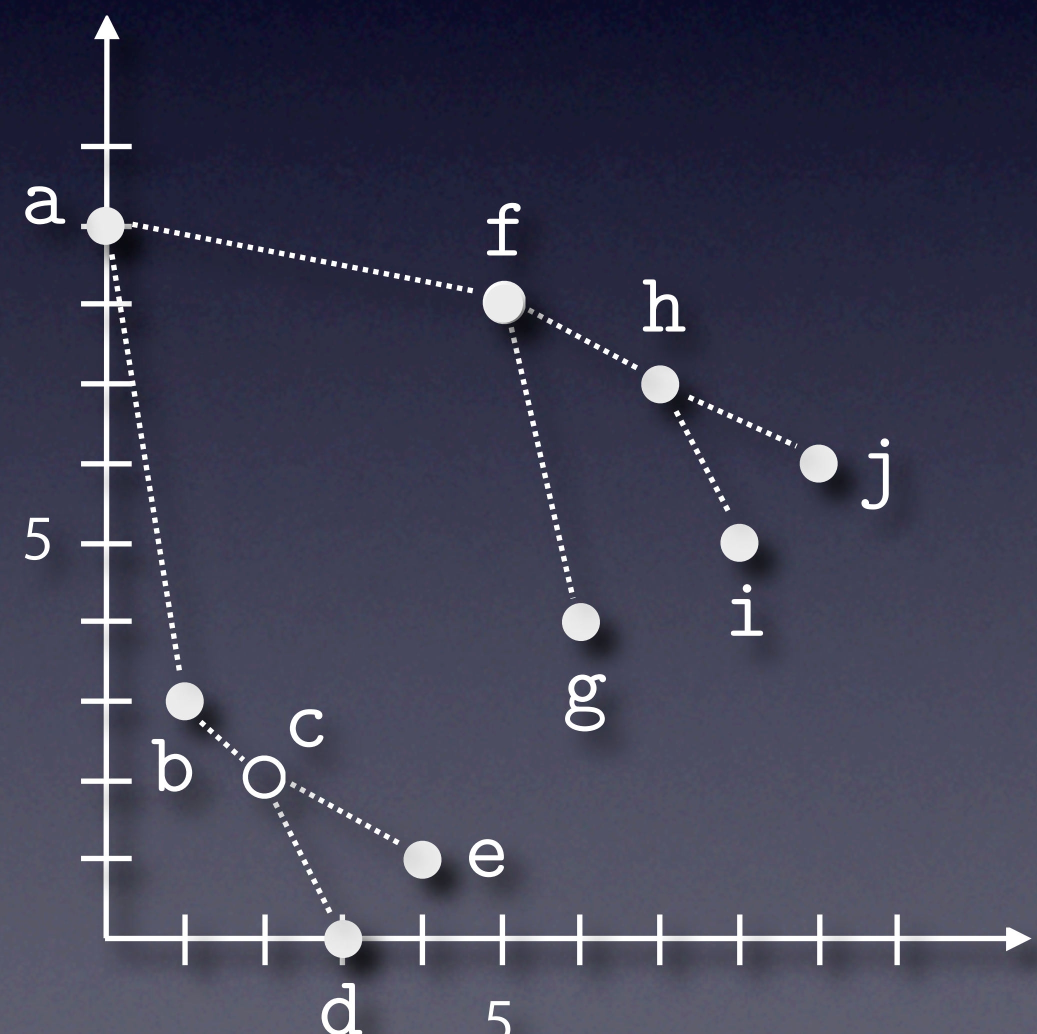
B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



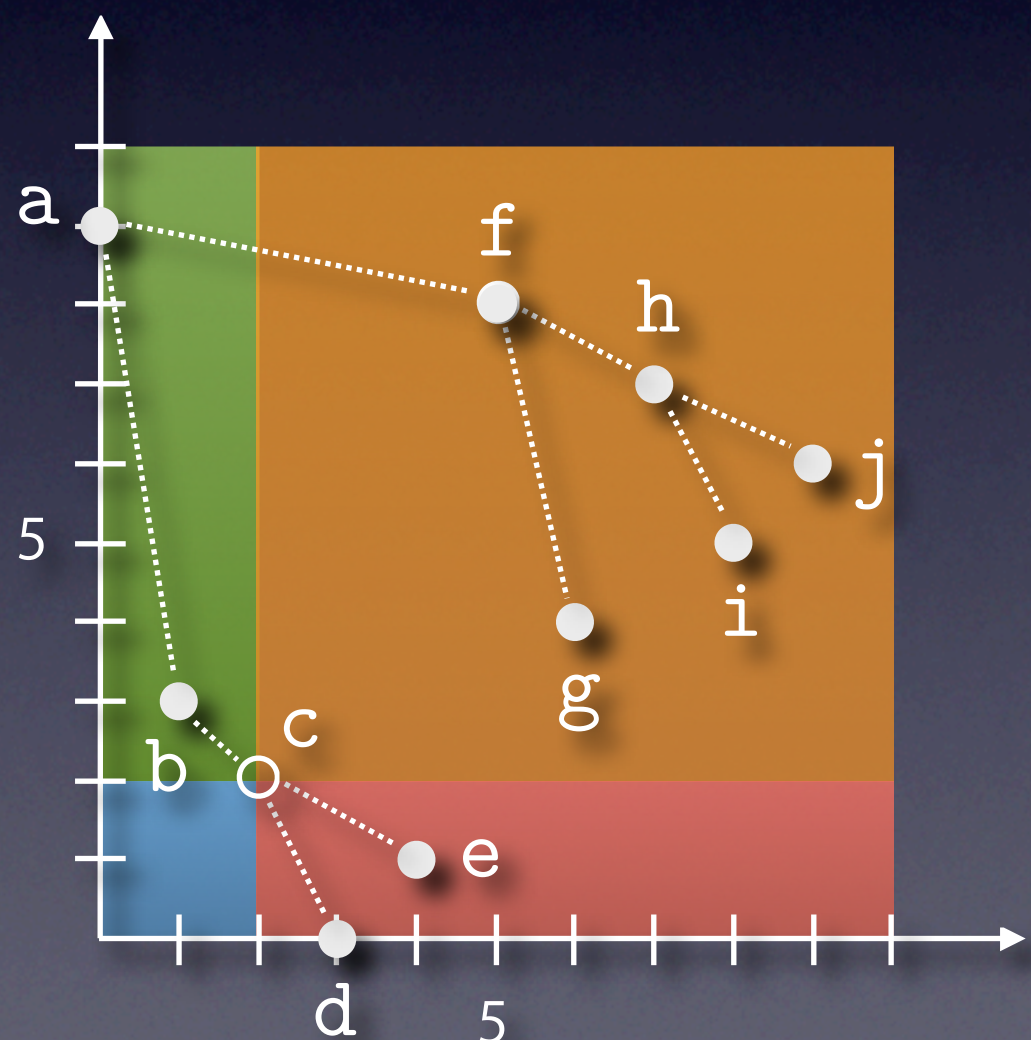
B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



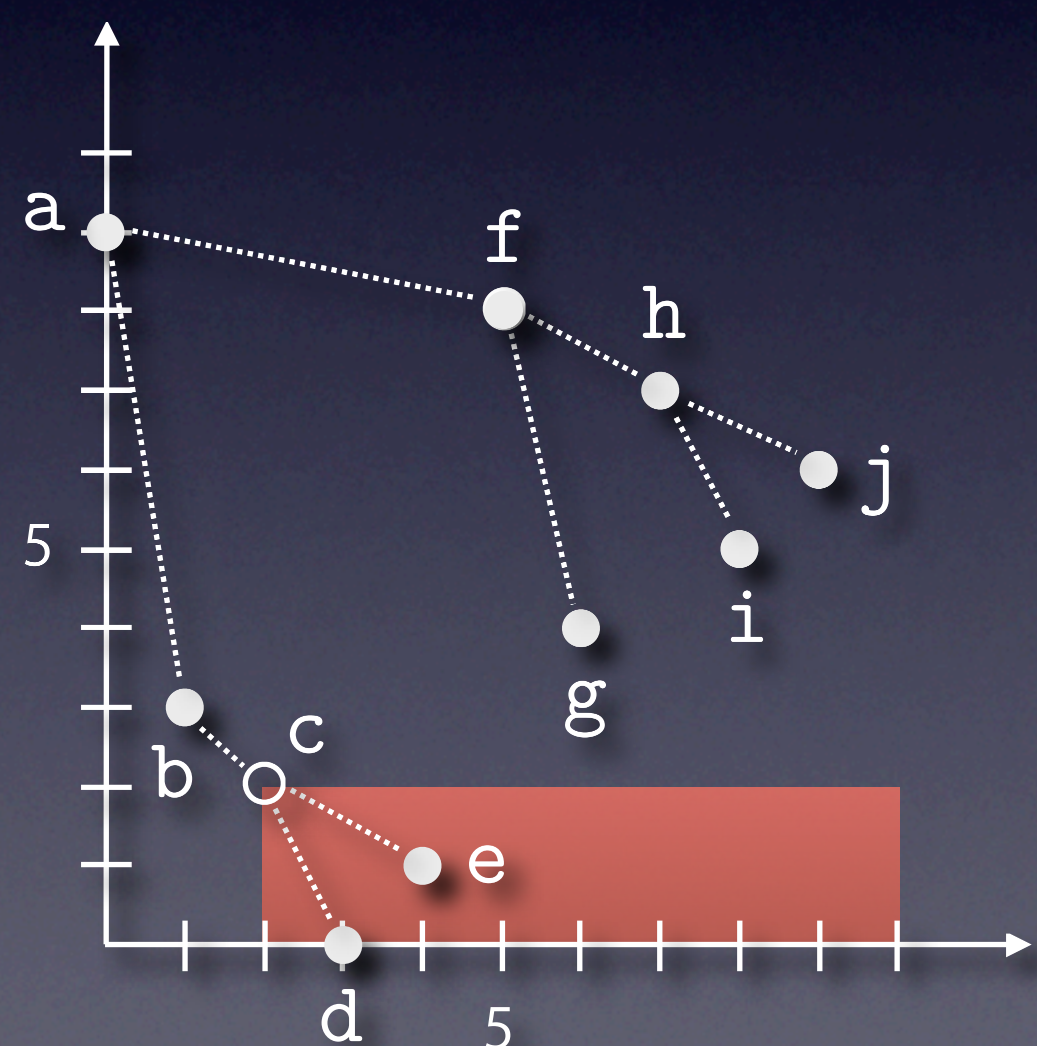
B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



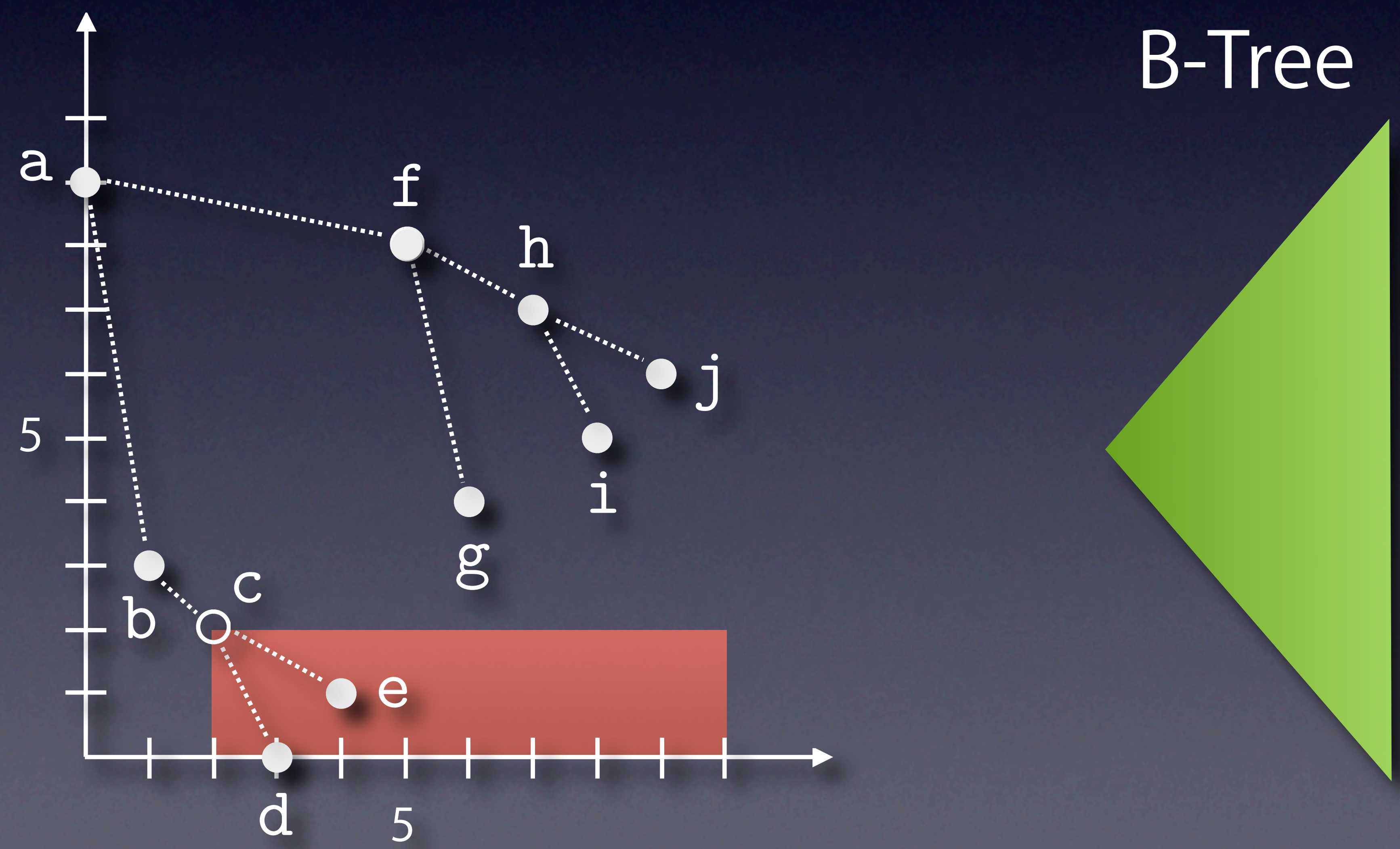
B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



B-Trees Accelerate XPath Location Steps

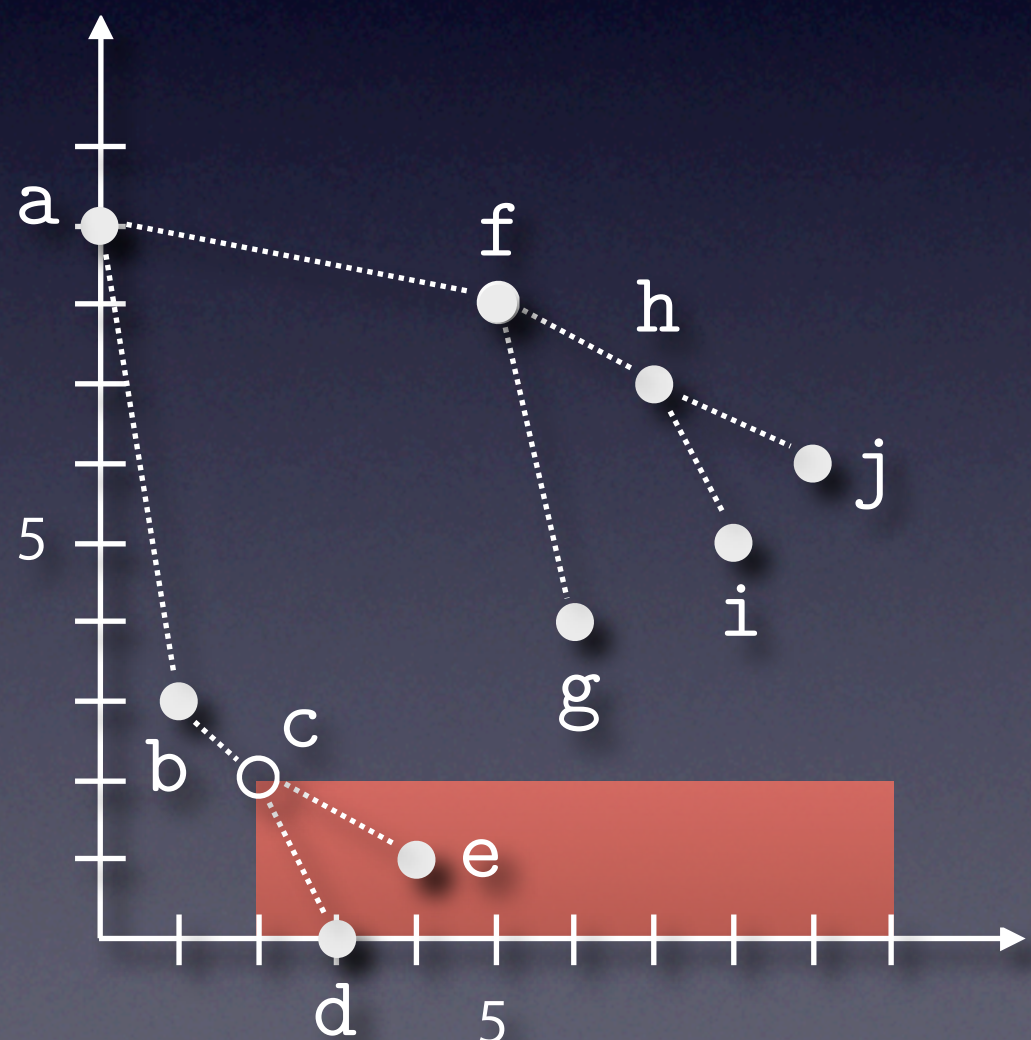
- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



pre	post	node
0	9	a
1	3	b
2	2	c
3	0	d
4	1	e
5	8	f
6	4	g
7	7	h
8	5	i
9	6	j

B-Trees Accelerate XPath Location Steps

- The XPath axes `ancestor`, `descendant`, `preceding`, `following` partition any XML document:



pre	post	node
0	9	a
1	3	b
2	2	c
3	0	d
4	1	e
5	8	f
6	4	g
7	7	h
8	5	i
9	6	j

Partitioned B-Trees

Partitioned B-Trees

- Use *low* selectivity key prefixes to partition the XML nodes in a B-Tree according to various criteria:

Partitioned B-Trees

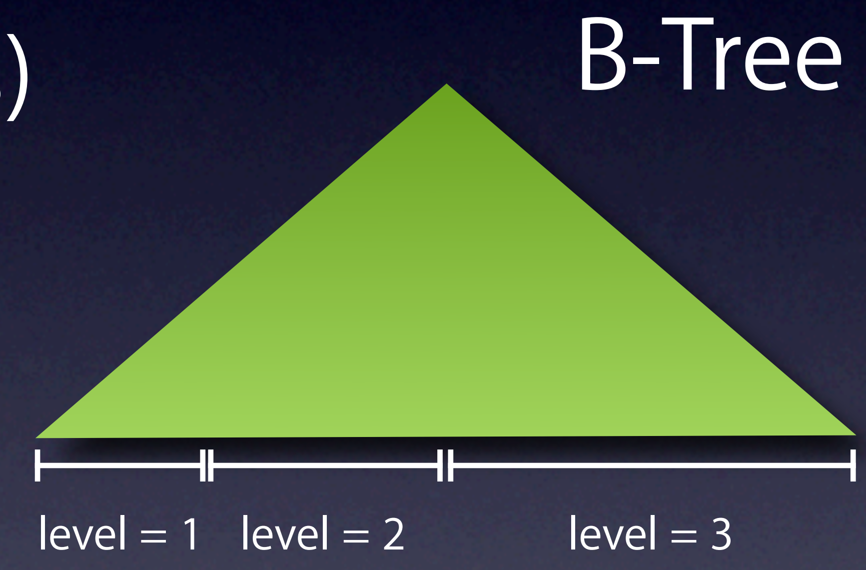
- Use *low* selectivity key prefixes to partition the XML nodes in a B-Tree according to various criteria:

  1. level (support XPath `child` axis)

Partitioned B-Trees

- Use *low* selectivity key prefixes to partition the XML nodes in a B-Tree according to various criteria:

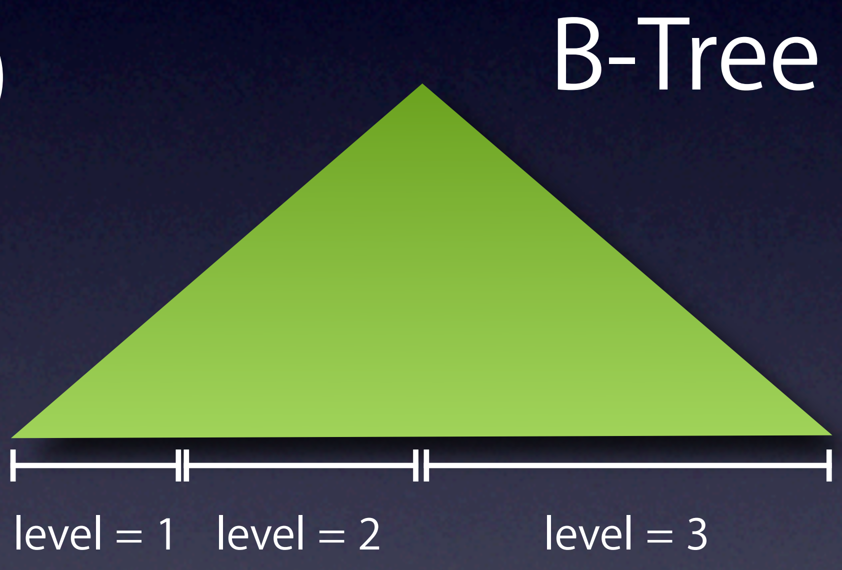
  1. level (support XPath `child` axis)



Partitioned B-Trees

- Use *low* selectivity key prefixes to partition the XML nodes in a B-Tree according to various criteria:

  1. level (support XPath `child` axis)
  2. element tag name
  3. path to node (*"Node GPS"*)



- Given a Pathfinder-generated workload, the index advisor of IBM DB2 V9 proposes such partitionings automatically.

Injecting More Tree Awareness

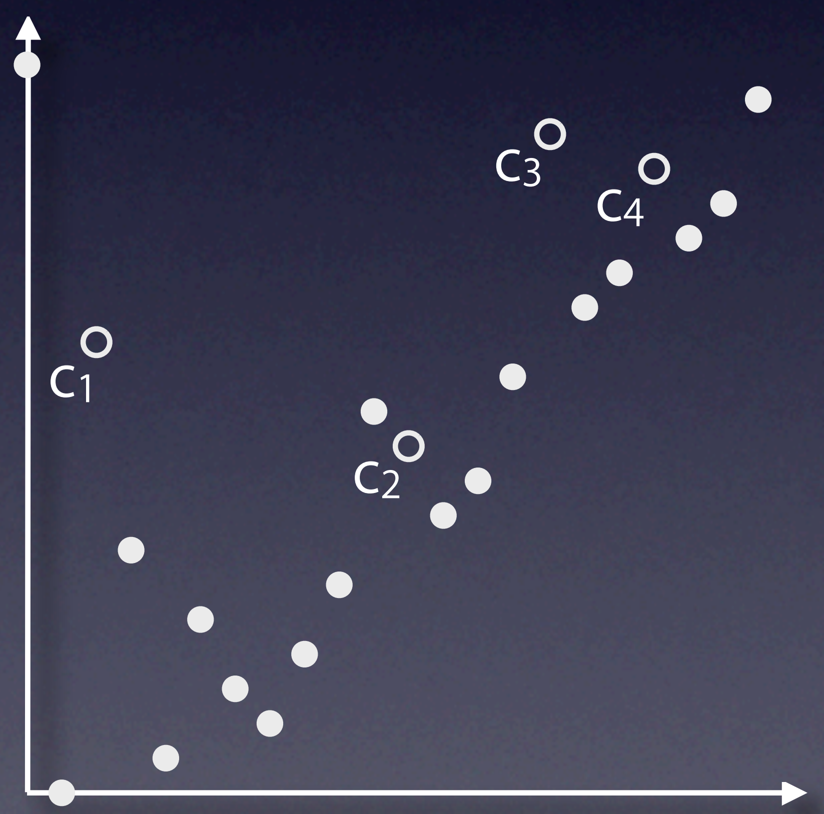
- In XQuery, an XPath location step originates in a sequence of context nodes.

Duplicate nodes, out of order results (XPath semantics!), wasted work.

Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

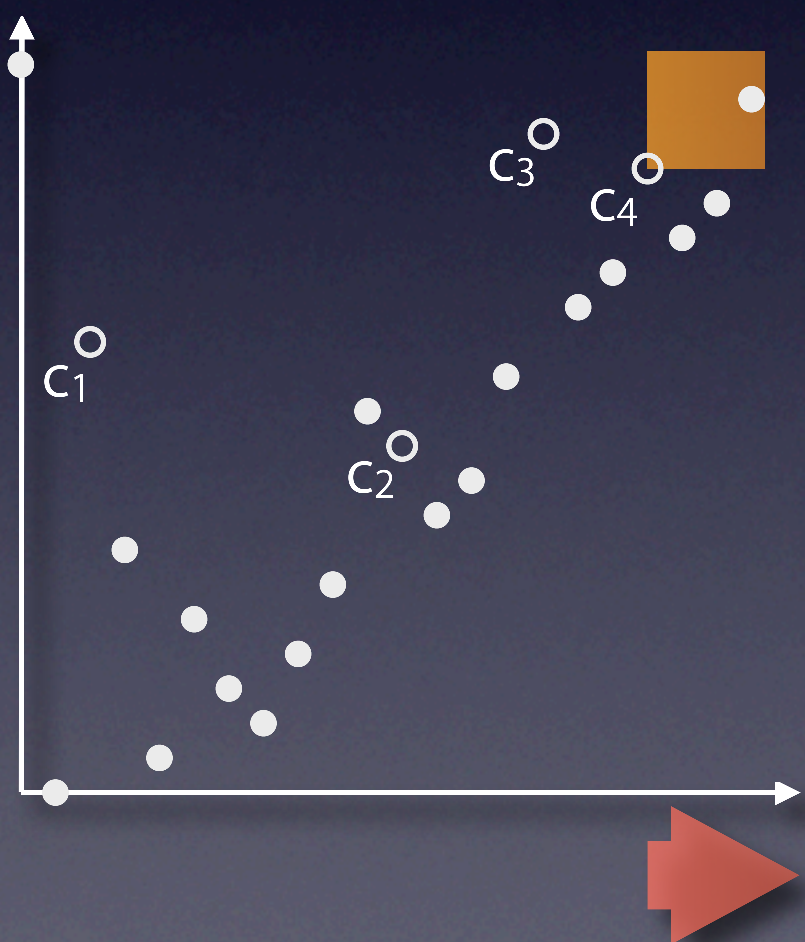
Duplicate nodes, out of order results (XPath semantics!), wasted work.



Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

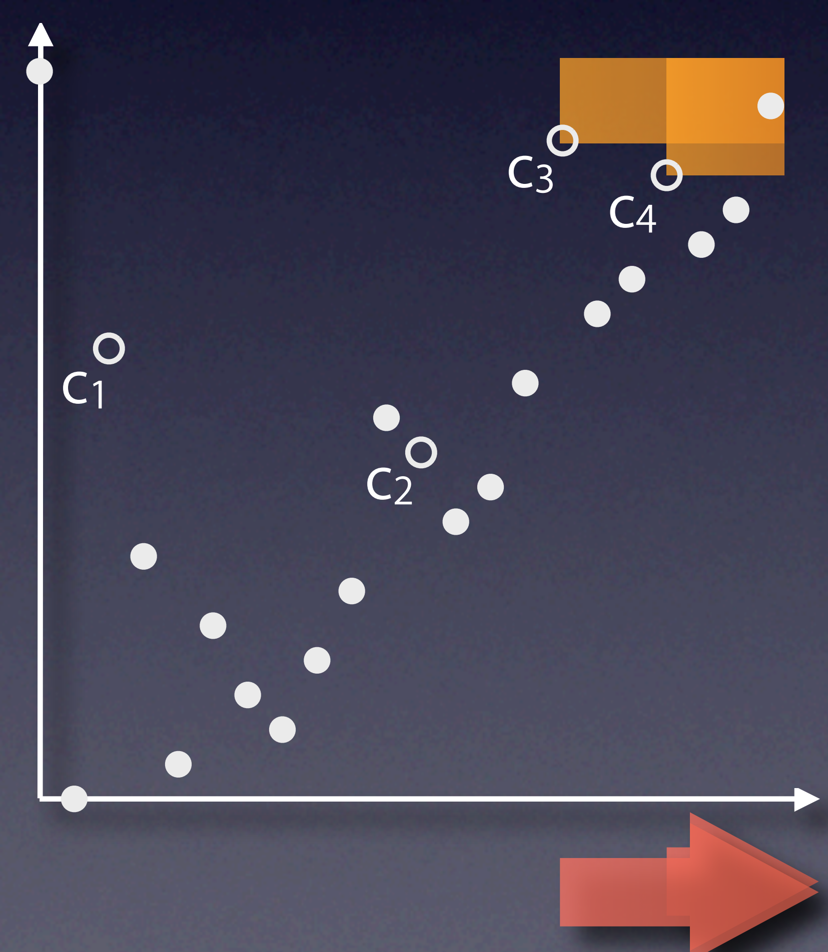
Duplicate nodes, out of order results (XPath semantics!), wasted work.



Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

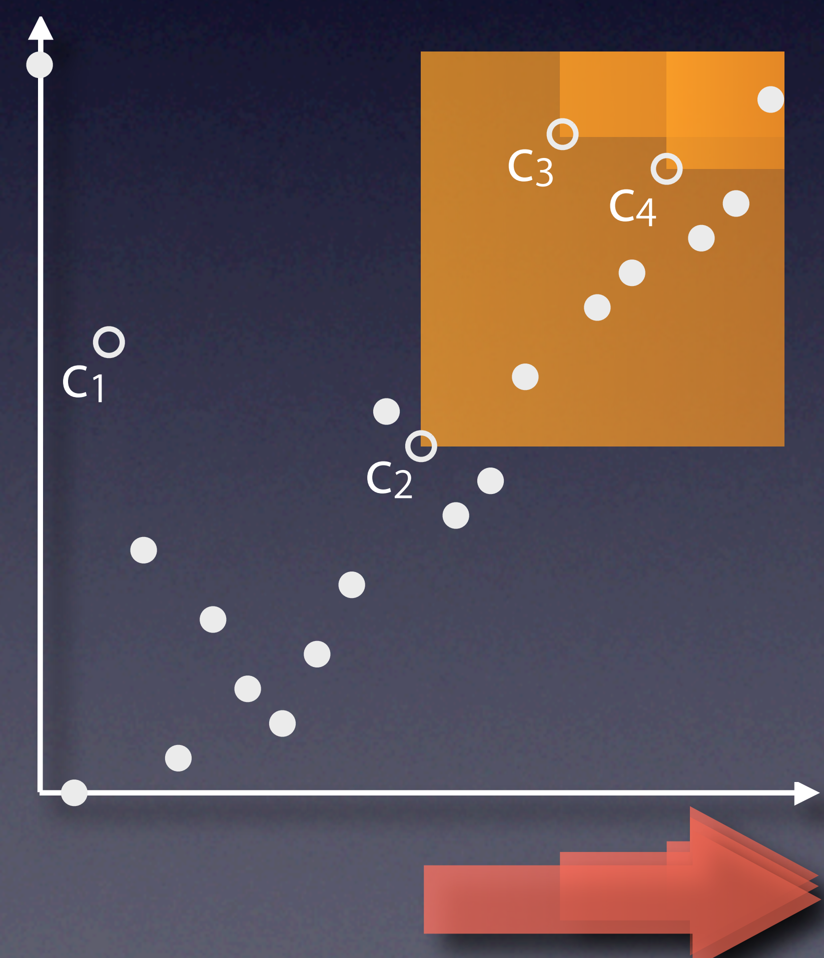
Duplicate nodes, out of order results (XPath semantics!), wasted work.



Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

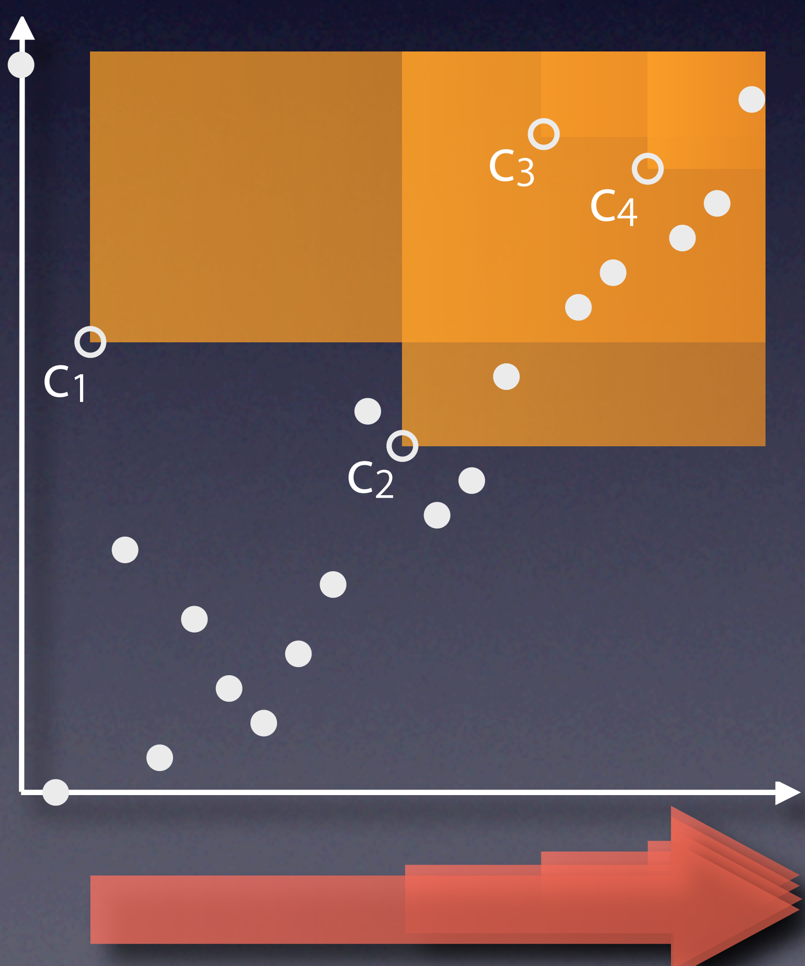
Duplicate nodes, out of order results (XPath semantics!), wasted work.



Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

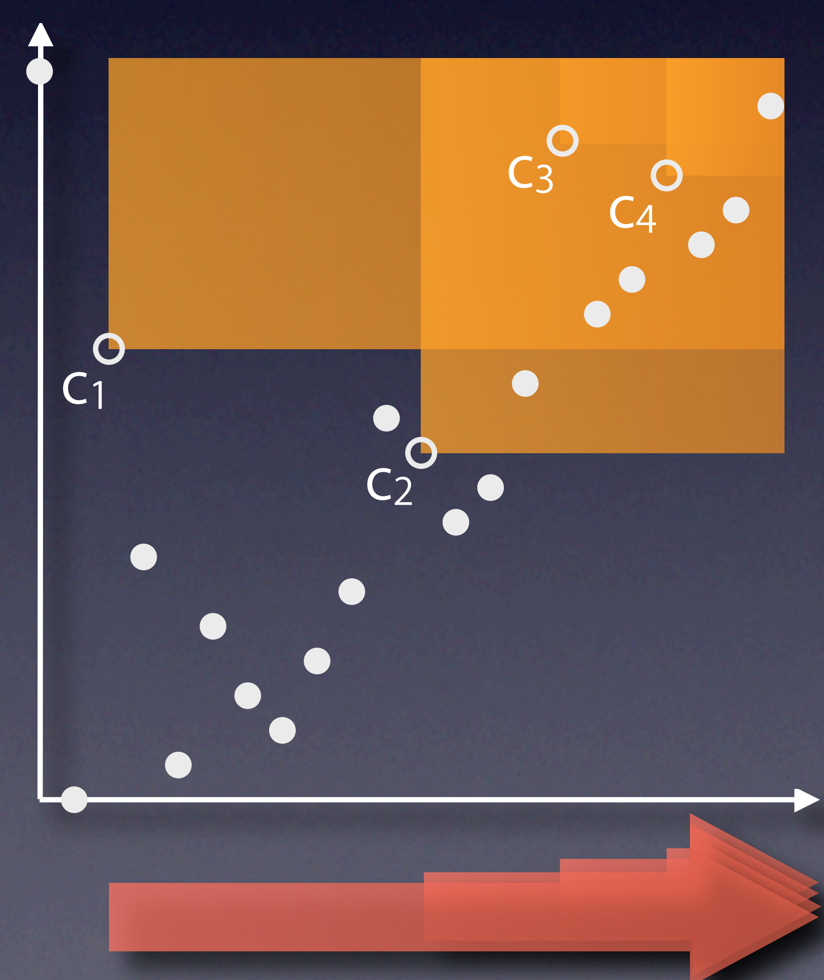
Duplicate nodes, out of order results (XPath semantics!), wasted work.

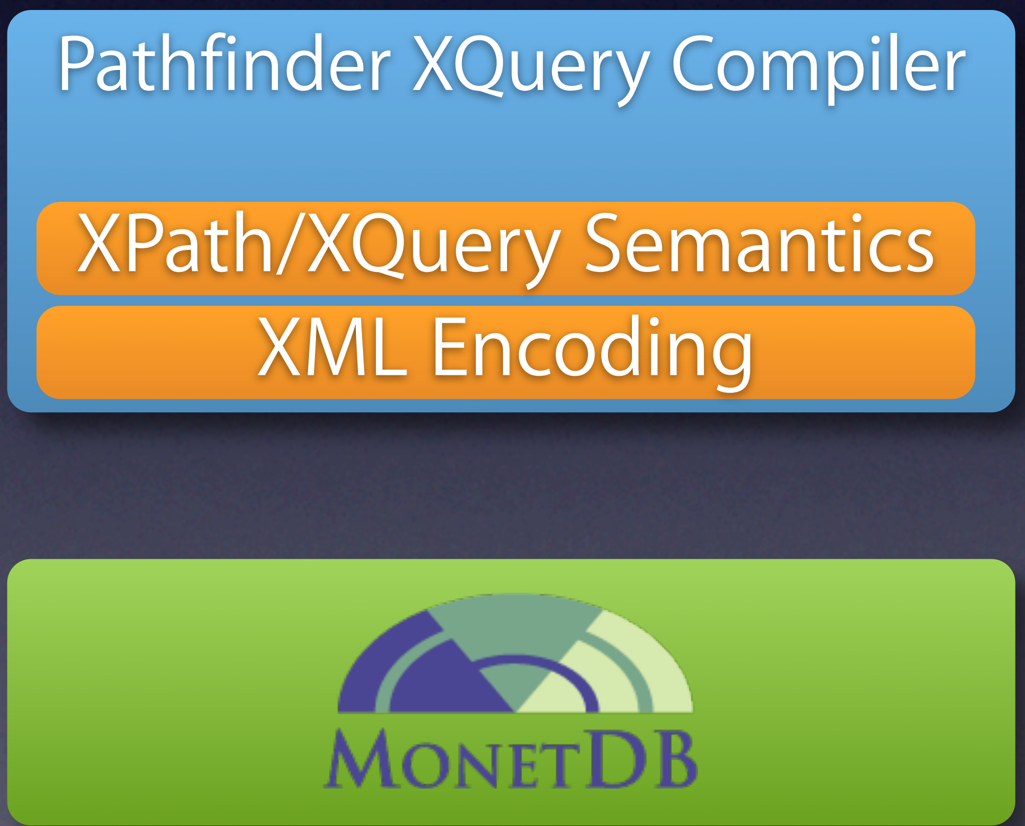


Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

Duplicate nodes, out of order results (XPath semantics!), wasted work.

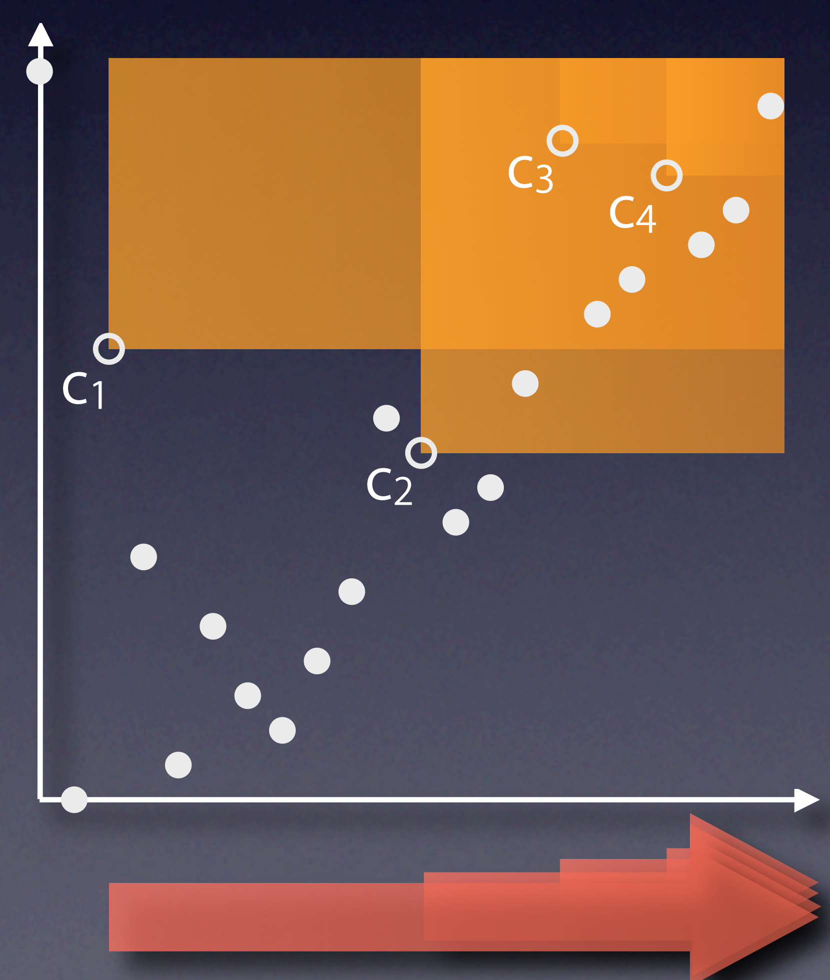


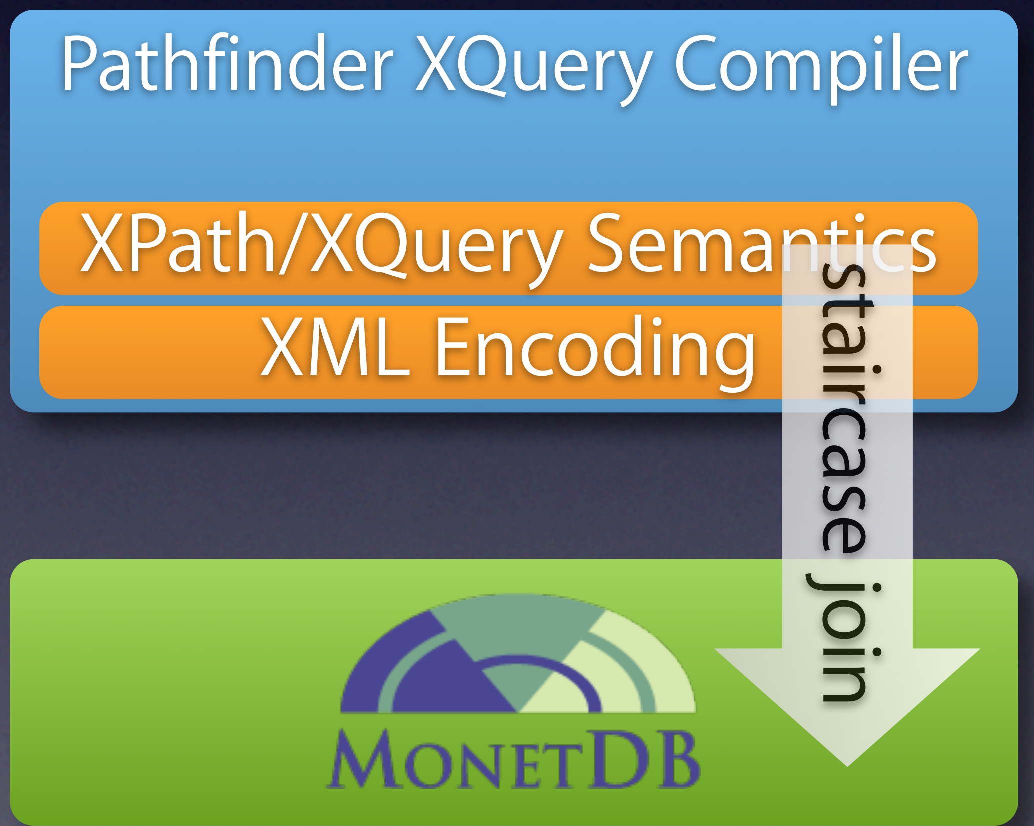


Injecting More Tree Awareness

- In XQuery, an XPath location step originates in a sequence of context nodes.

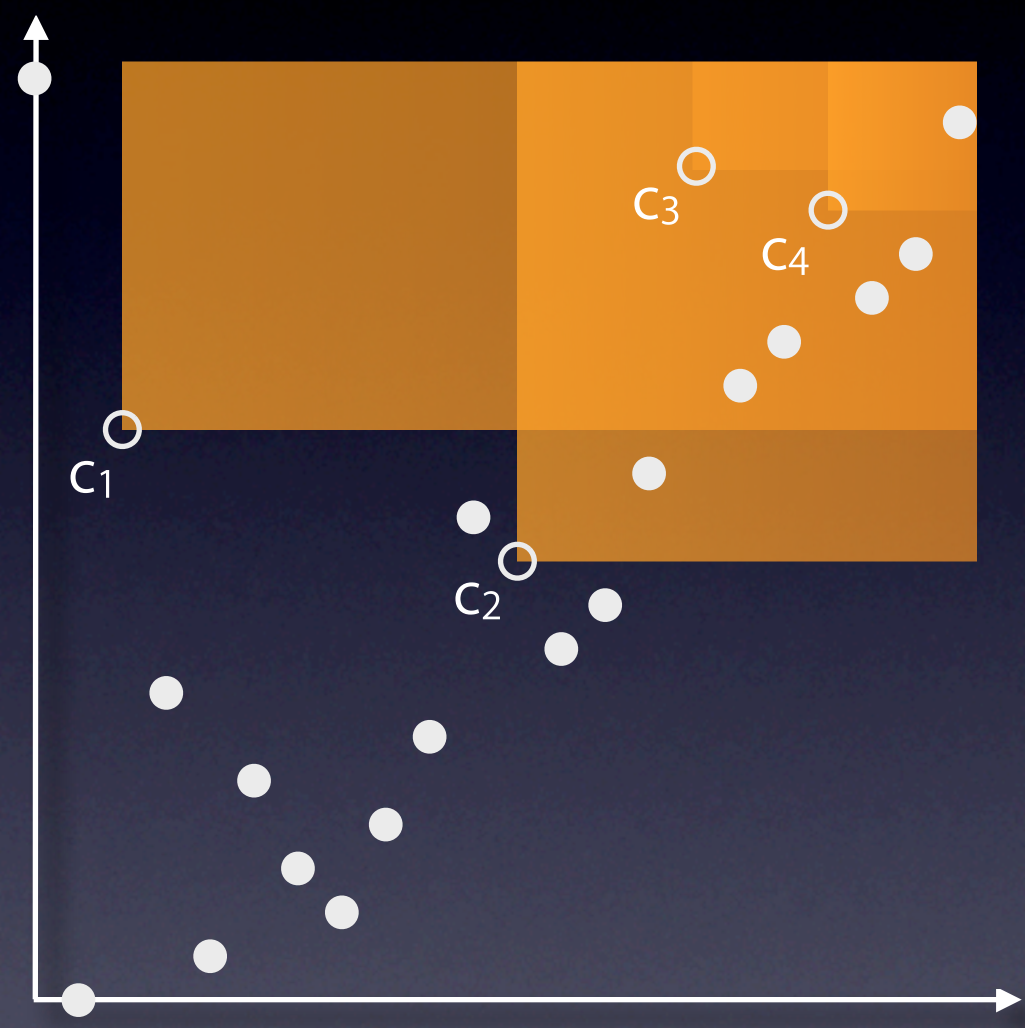
Duplicate nodes, out of order results (XPath semantics!), wasted work.





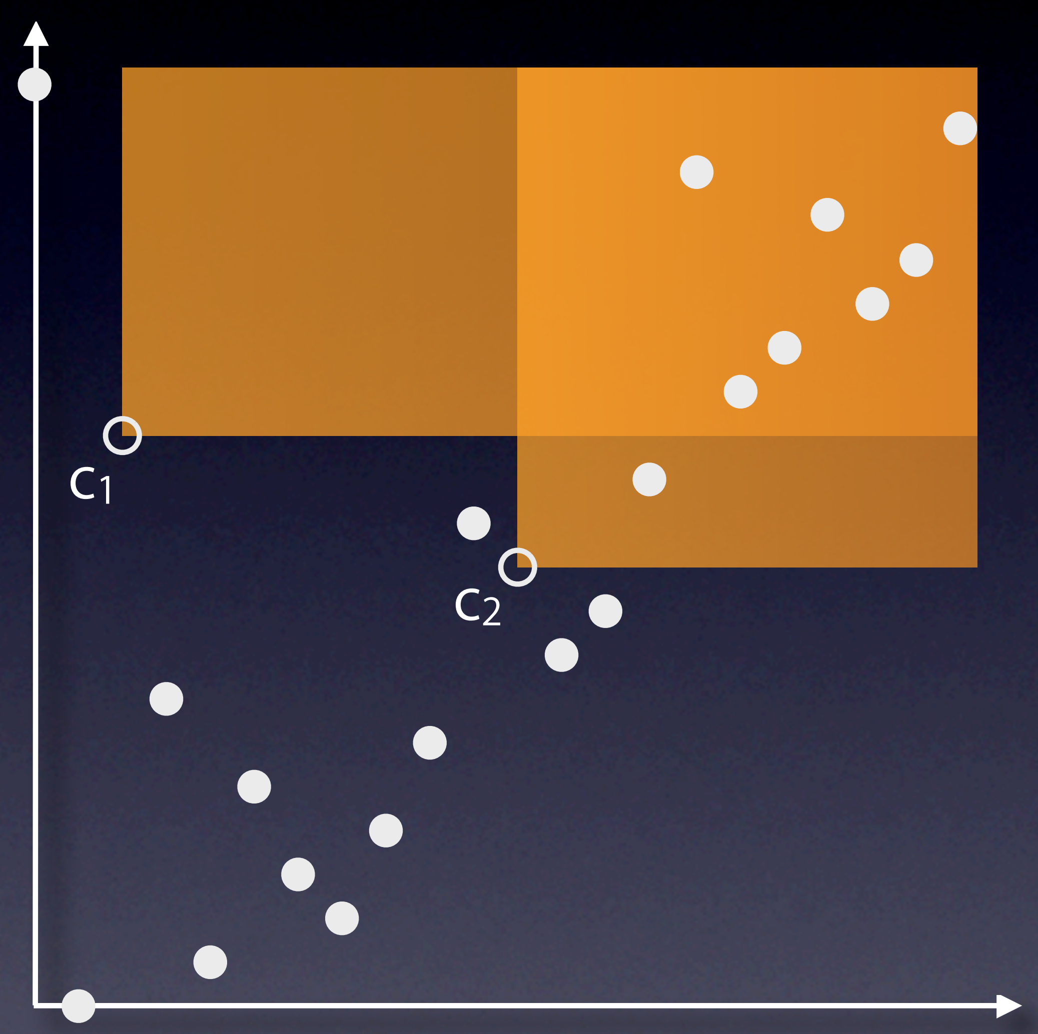
Staircase Join: Context Pruning

- XPath `following` axis:



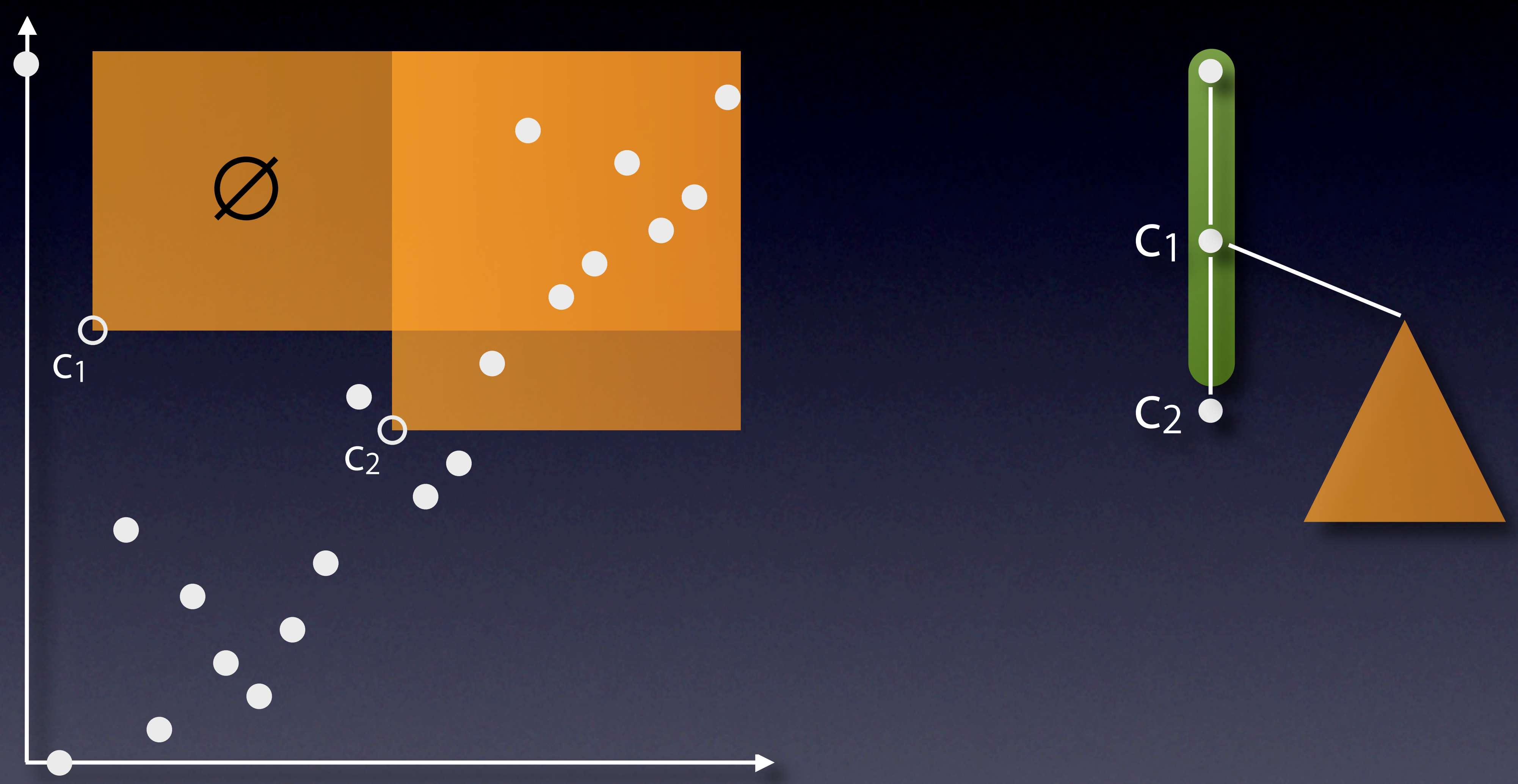
Staircase Join: Context Pruning

- XPath `following` axis:



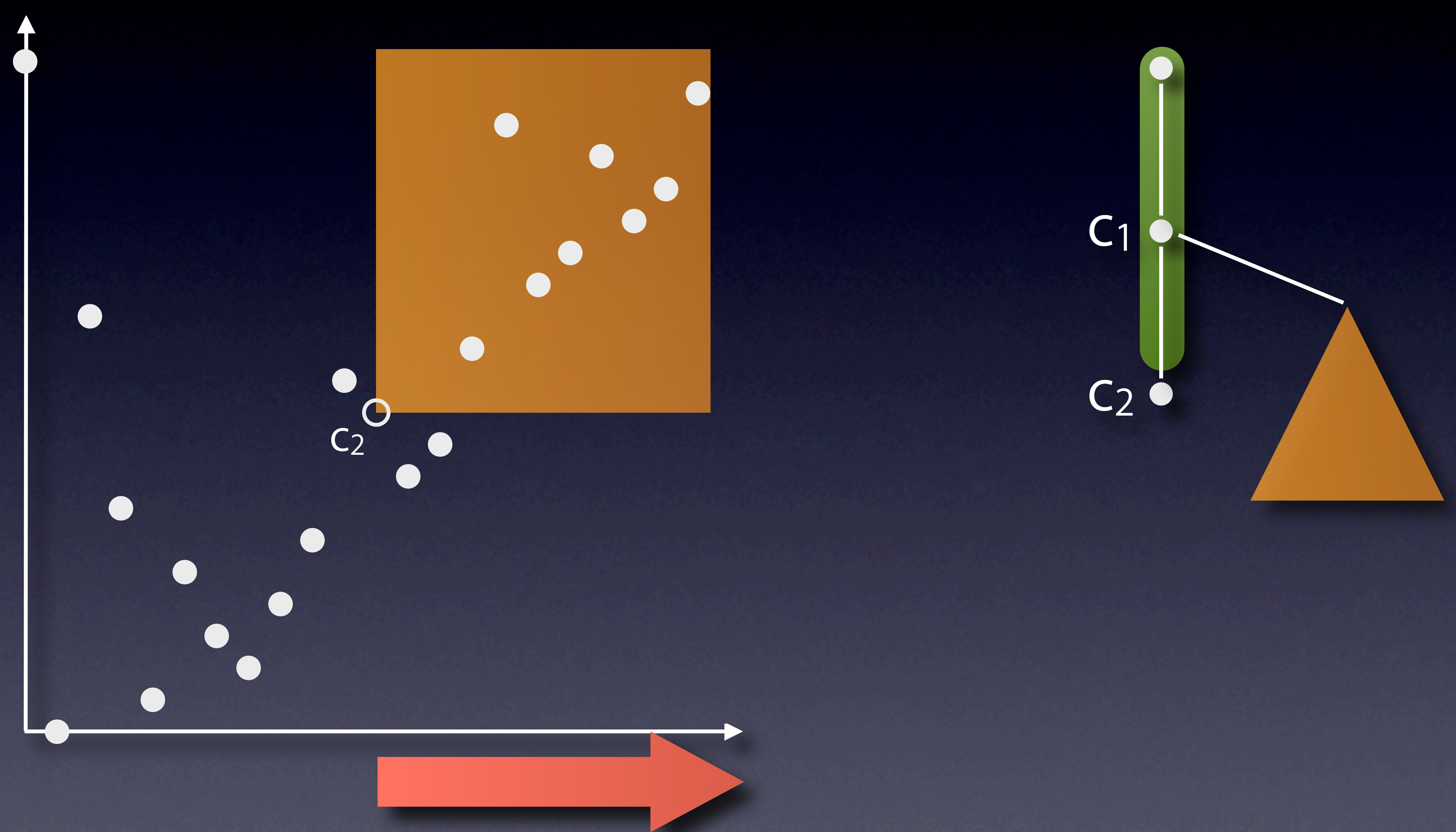
Staircase Join: Context Pruning

- XPath `following` axis:



Staircase Join: Context Pruning

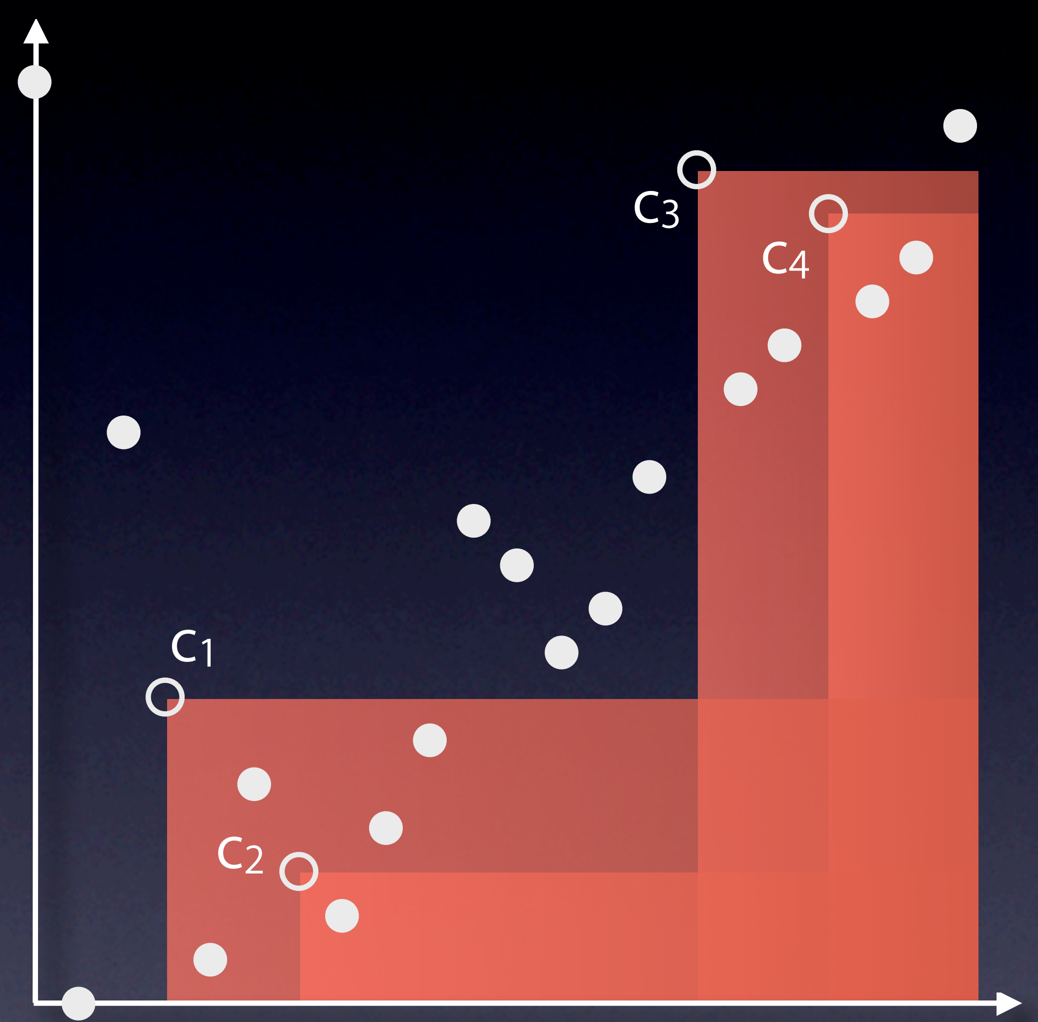
- XPath `following` axis:



- Index scan yields duplicate free result in document order.

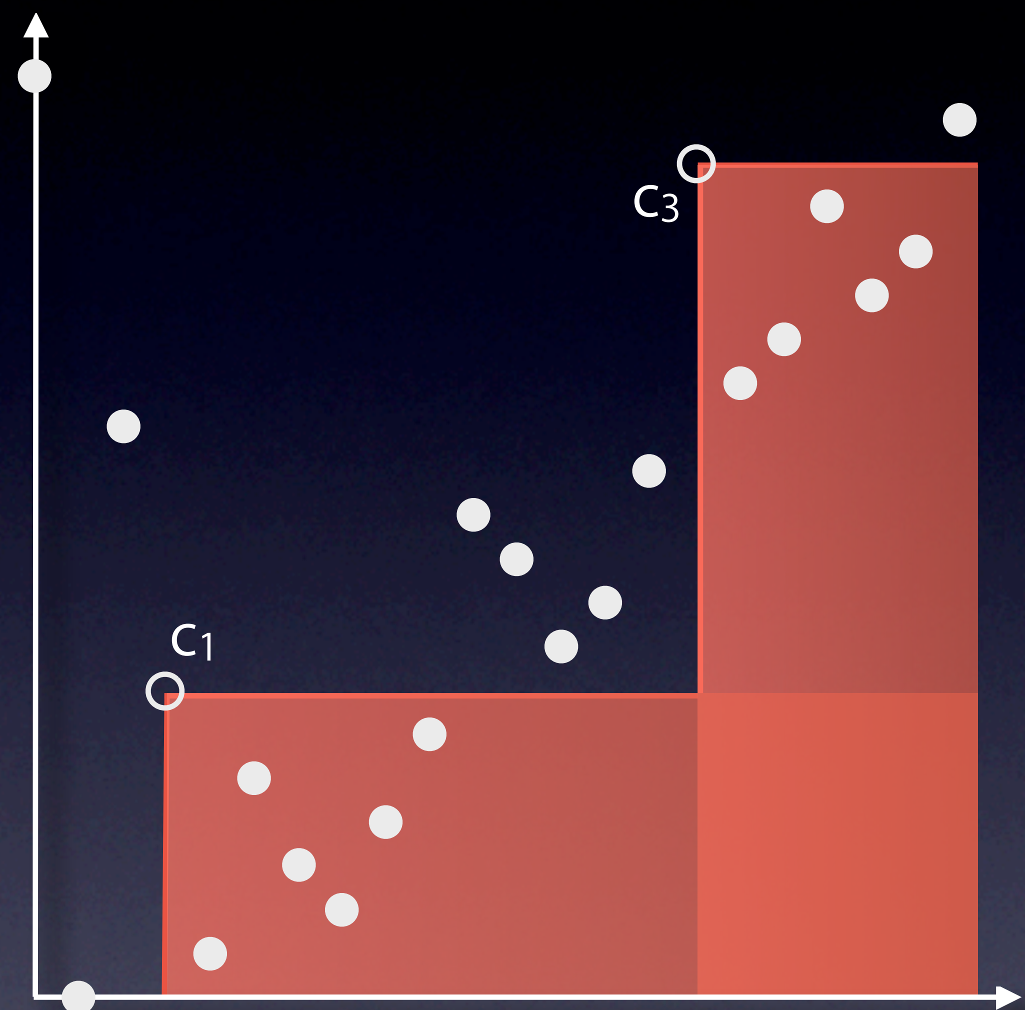
Staircase Join: Skipping

- XPath `descendant` axis:



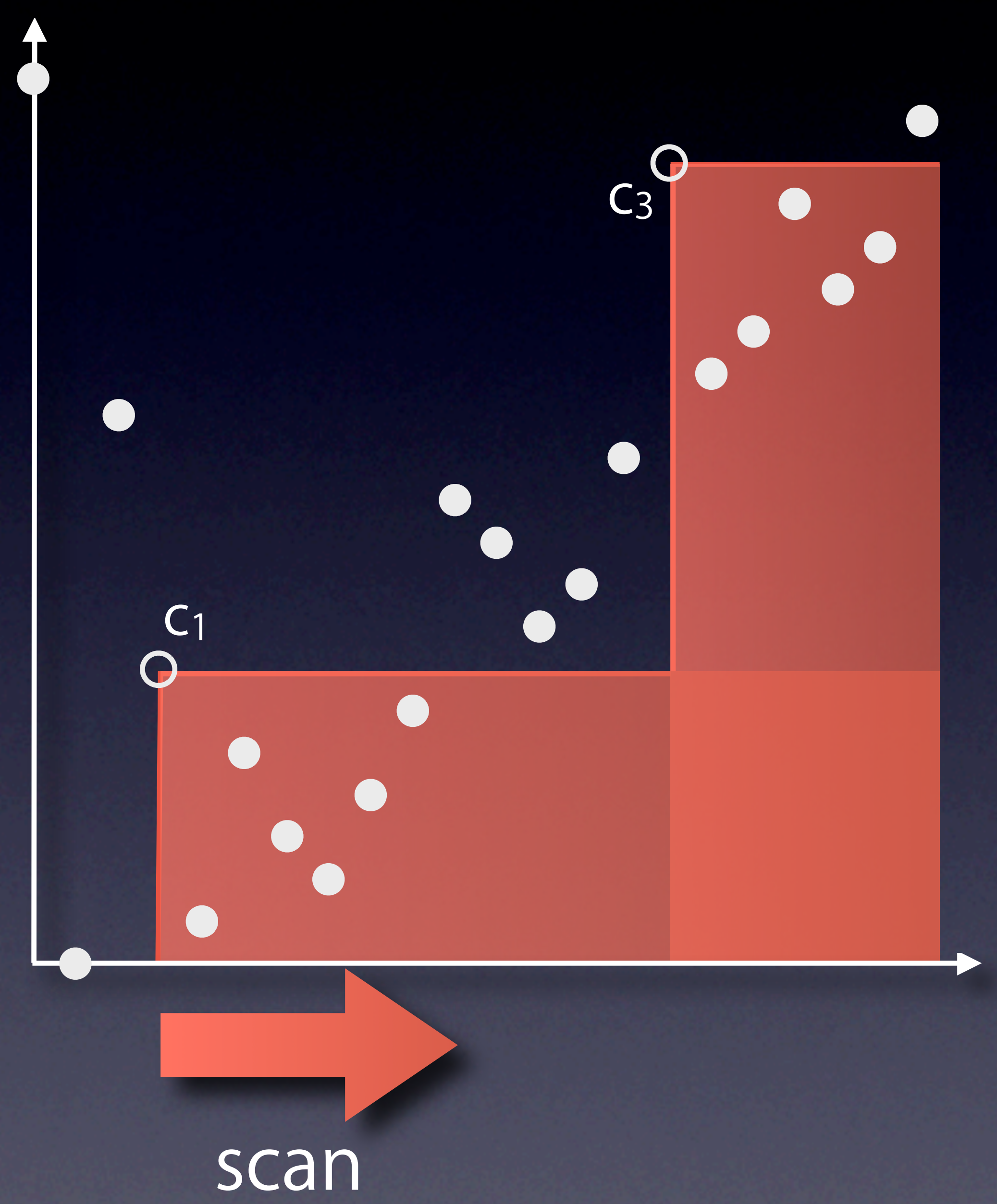
Staircase Join: Skipping

- XPath `descendant` axis:



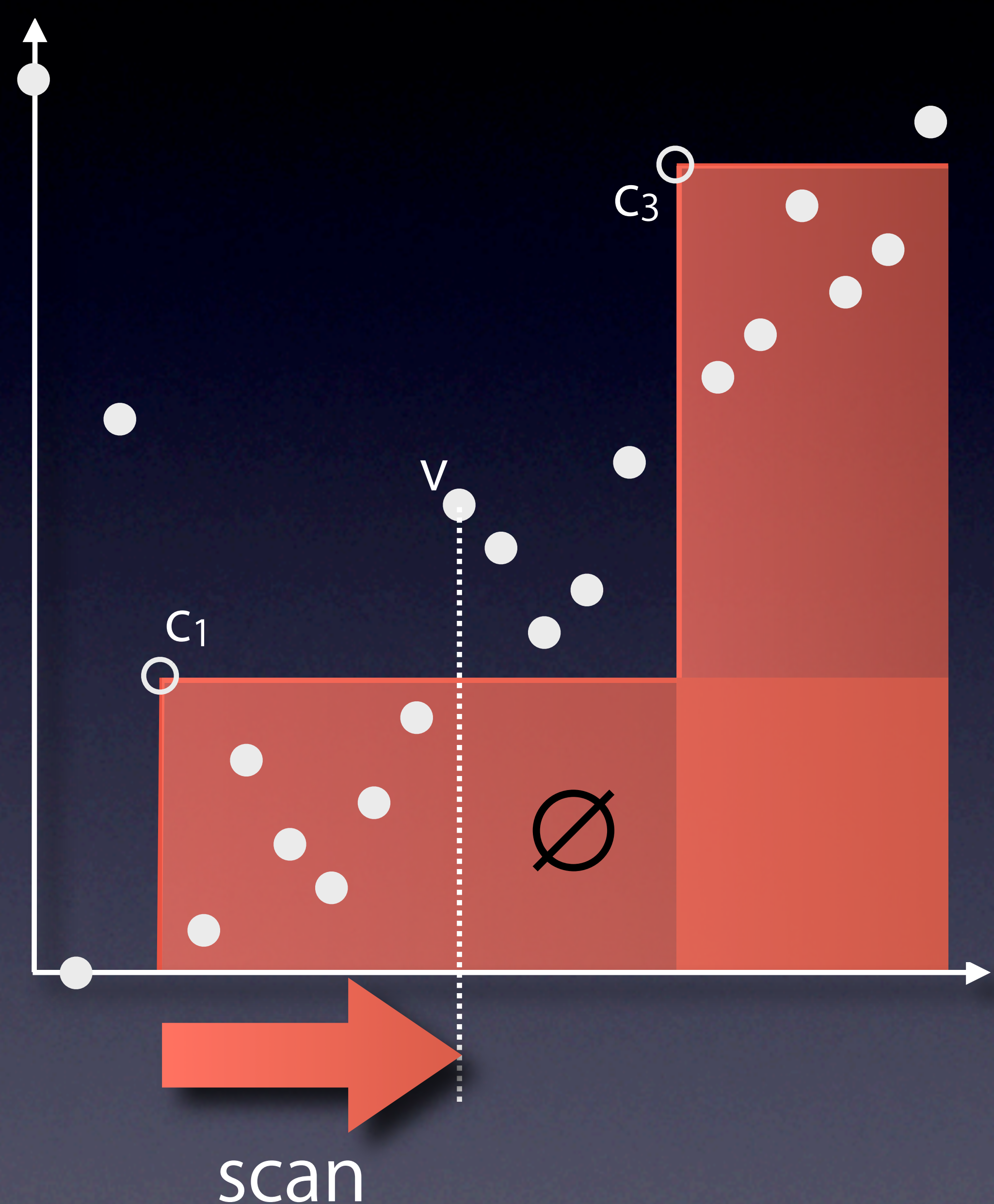
Staircase Join: Skipping

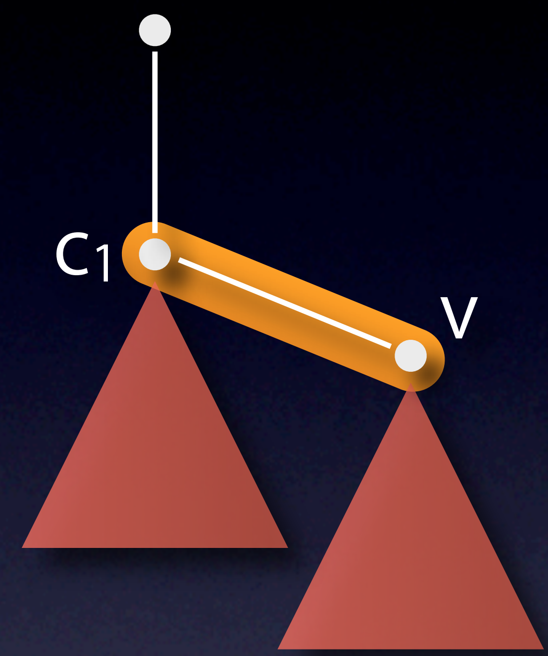
- XPath `descendant` axis:



Staircase Join: Skipping

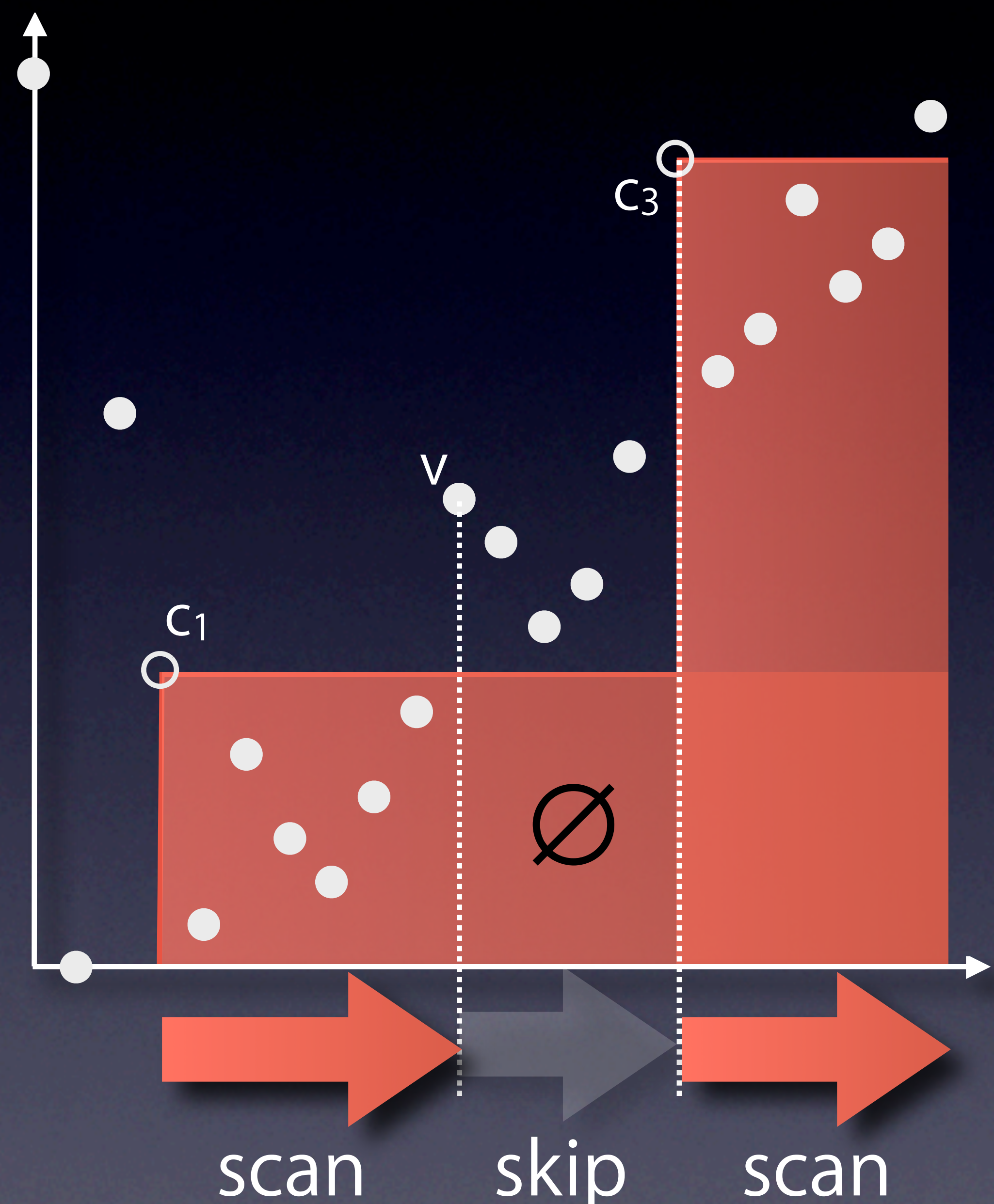
- XPath `descendant` axis:

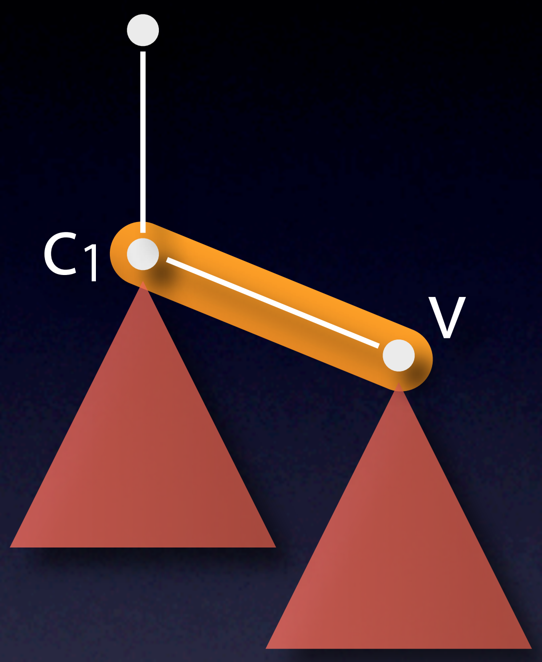




Staircase Join: Skipping

- XPath `descendant` axis:





- Index scan yields duplicate free result in document order.

MonetDB/XQuery

MonetDB/XQuery

- MonetDB: Extensible relational database kernel, optimized for query processing close to the CPU.
  - Full vertical fragmentation (column store).

pre	post	node
0	9	a
1	3	b
2	2	c
3	0	d
4	1	e
5	8	f
6	4	g
7	7	h
8	5	i
9	6	j

MonetDB/XQuery

- MonetDB: Extensible relational database kernel, optimized for query processing close to the CPU.
  - Full vertical fragmentation (column store).

pre	post
0	9
1	3
2	2
3	0
4	1
5	8
6	4
7	7
8	5
9	6

pre	node
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j

MonetDB/XQuery

- MonetDB: Extensible relational database kernel, optimized for query processing close to the CPU.
  - Full vertical fragmentation (column store).

pre	post
0	9
1	3
2	2
3	0
4	1
5	8
6	4
7	7
8	5
9	6

pre	node
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j

MonetDB/XQuery

- MonetDB: Extensible relational database kernel, optimized for query processing close to the CPU.
  - Full vertical fragmentation (column store).
- Pathfinder + MonetDB = MonetDB/XQuery.
  - Implementation of XQuery 1.0.
  - Evaluates queries against GB-range XML instances in interactive time.

pre	post
0	9
1	3
2	2
3	0
4	1
5	8
6	4
7	7
8	5
9	6

pre	node
0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j

\+ XQuery Update, XQuery RPC (`execute at` *uri* { *e* }), support for multi-dimensional XML, ...

“DB2’s XML support doesn’t beat its relational self”.

“DB2’s XML support doesn’t beat its relational self”.

- Pathfinder builds on 30+ years of development of relational database technology.

“DB2’s XML support doesn’t beat its relational self”.

- Pathfinder builds on 30+ years of development of relational database technology.
  - Outperforms the built-in DB2 V9 pureXML® engine, especially for large XML input instances.

"DB2's XML support doesn't beat its relational self".

- Pathfinder builds on 30+ years of development of relational database technology.
  - Outperforms the built-in DB2 V9 pureXML® engine, especially for large XML input instances.
- Covers other data-intensive languages:

1. SQL/XML

“DB2’s XML support doesn’t beat its relational self”.

- Pathfinder builds on 30+ years of development of relational database technology.
  - Outperforms the built-in DB2 V9 pureXML® engine, especially for large XML input instances.
- Covers other data-intensive languages:

1. SQL/XML
2. LINQ + other “*Nested Loop*” languages.

grust@in.tum.de
http://www-db.in.tum.de/~grust/