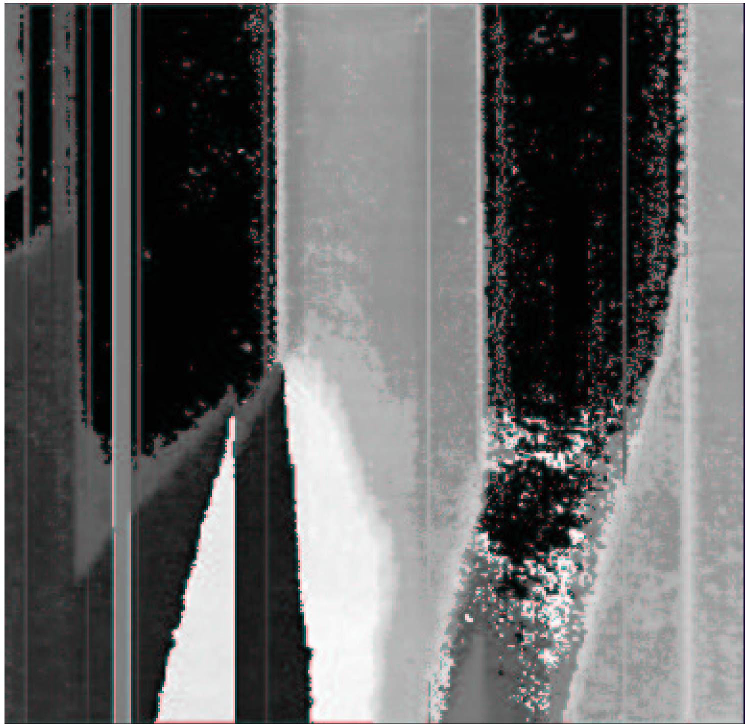


XQuery midflight: Emerging Database-Oriented Paradigms and a Classification of Research Advances

Ioana Manolescu
INRIA, France
Yannis Papakonstantinou
UCSD, USA

April 5, 2005

UCSD Jacobs School
CSE Department

Outline

Tuple-based evaluation of XQuery

XQDMA: an abstraction of the XQuery data model

Unified tuple-based algebra

Related work

Tuple-based Evaluation of XQuery

UCSD Jacobs School
CSE Department
INRIA

Database Research Meets XQuery Processing

Database research has succeeded on querying large data volumes

Tuple-based algebras have been key ingredient

- Relational, OQL
- Algebraic cost-based optimization
- Set-at-a-time query processing primitives (eg join)

Several tuple-based algebras for XQuery

- similar, yet different

PRIMARY GOAL: Formal definition of a unifying tuple-based algebra for processing XQuery

- functional coverage

Better communication in research and education

The Need for XQuery Data Model Abstraction (XQDMA)

Formal XML query languages and algebras have used labeled tree abstraction

XQuery data model is extremely complex for research

SECONDARY GOAL: XQuery Data Model Abstraction

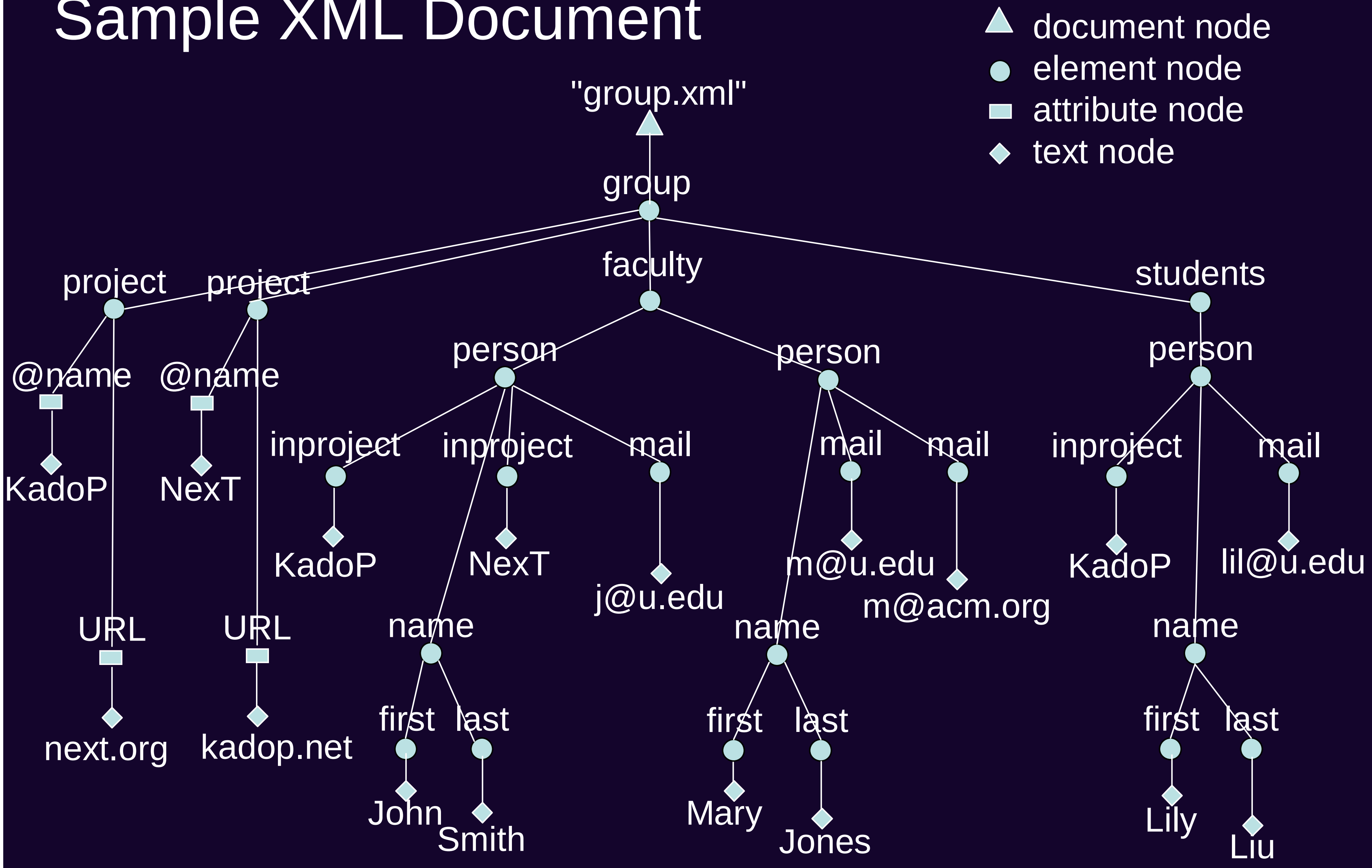
- Faithful to XQuery Data Model, leads to relevance to XQuery itself
- Yet modularization of issues enables redacted models

Abstraction of the XQuery Data Model

Sample XML Document

```
<department>
    <project name="NexT" url="next.org"/>
    <project name="KadoP" url="kadop.net"/>
    <faculty>
        <person> <inproject>NexT</inproject><inproject>KadoP</inproject>
            <name><first>John</first><last>Smith</last></name>
            <mail>j@u.edu</mail>
        </person>
        <person>
            <name> <first>Mary</first> <last>Jones</last></name>
            <mail>m@u.edu</mail> <mail>m@acm.org</mail>
        </person>
    </faculty>
    <students>
        <person> <inproject>KadoP</inproject>
            <name> <first>Lily</first> <last>Liu</last></name> <mail>lil@u.edu</mail>
        <person>
</students>
</department>
```

Sample XML Document



XQDMA Definition

Labeled (ordered) trees

Nodes

- Four kinds: Document, element, attribute, text
- Single document node
- Element/attribute nodes labeled with XQDM element/attribute names
  - by convention, attribute names start with @
- Text nodes labeled with XQDM values
- Nodes have unique *identities*
- *Type function* $\mathcal{T}$ labels every node with XQDM type

Edges

- Document node is root and has exactly one child, which is element
- Attribute nodes may appear only as children of element nodes
- Attribute nodes may only have text node children
- Text nodes may only be leaves

Equality Relationships (1/2)

Node ID-based equality $=_{id}$ (XQuery *is*)

- Two nodes are id-equal if they are the same

Value-based equality $=_{v}$ (XQuery *eq*)

- Two values are equal if the results of *casting* one or both into a *common domain* are equal. Depends on values' types.
- 24 atomic types [XQDM,XSch]; casting rules in [XQFO]

Equality Relationships (2/2)

Node ID-based equality $=_{id}$ (XQuery *is*)

Value-based equality $=_{v}$ (XQuery *eq*)

Limited value-based equality

- Text nodes are value-based equal if their labels are equal
- Attribute/Element nodes n_1 and n_2 are value-based equal if their labels are equal and
  - (unordered) for every child of n_1 there is an equal child of n_2 and vice versa
  - (obvious generalization to ordered)

Limited value-based equality implies Value-based equality

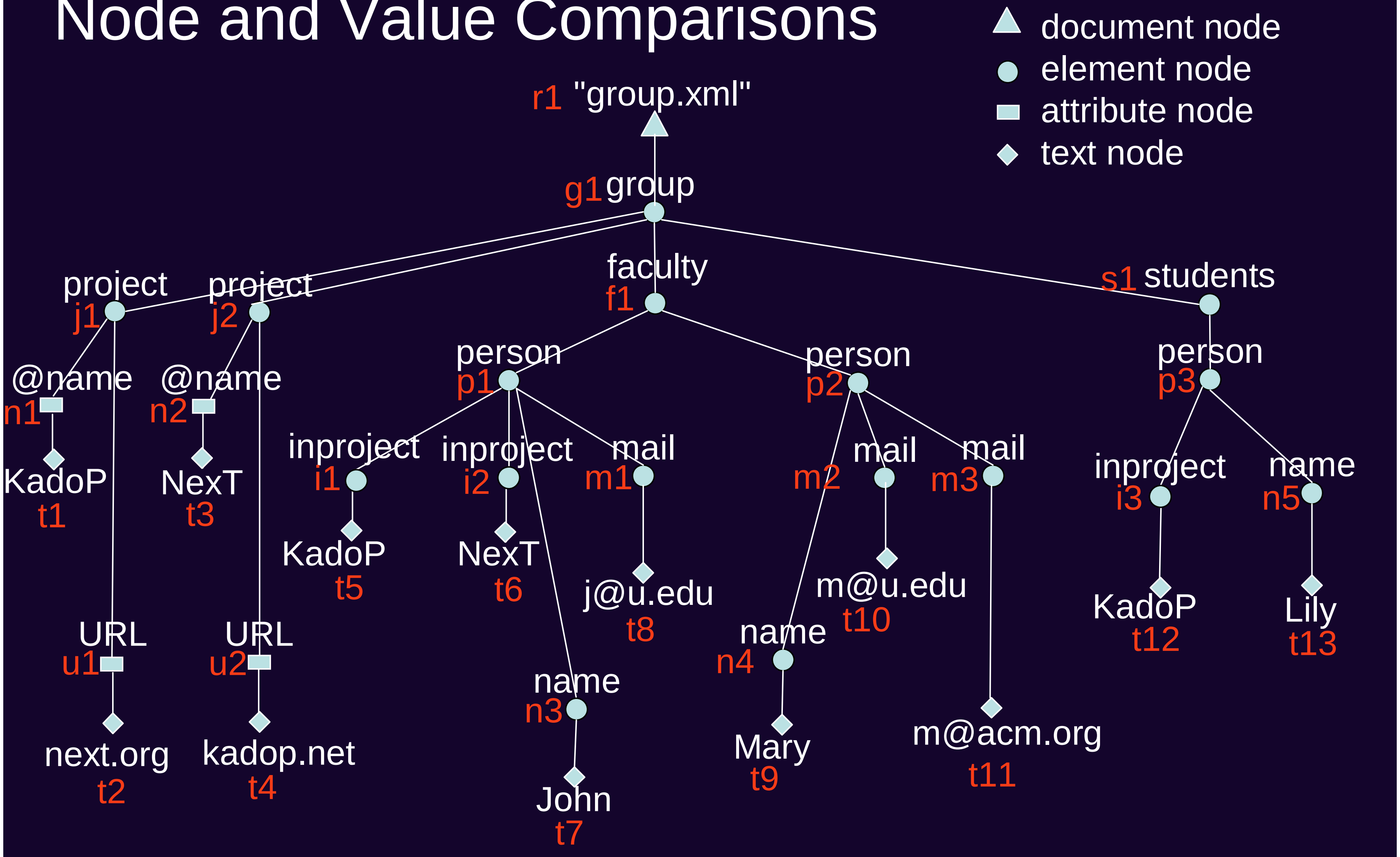
Not vice-versa (eg, text node with "005" is equal to text node with "05", considering typing and coercion)

Order Relationship

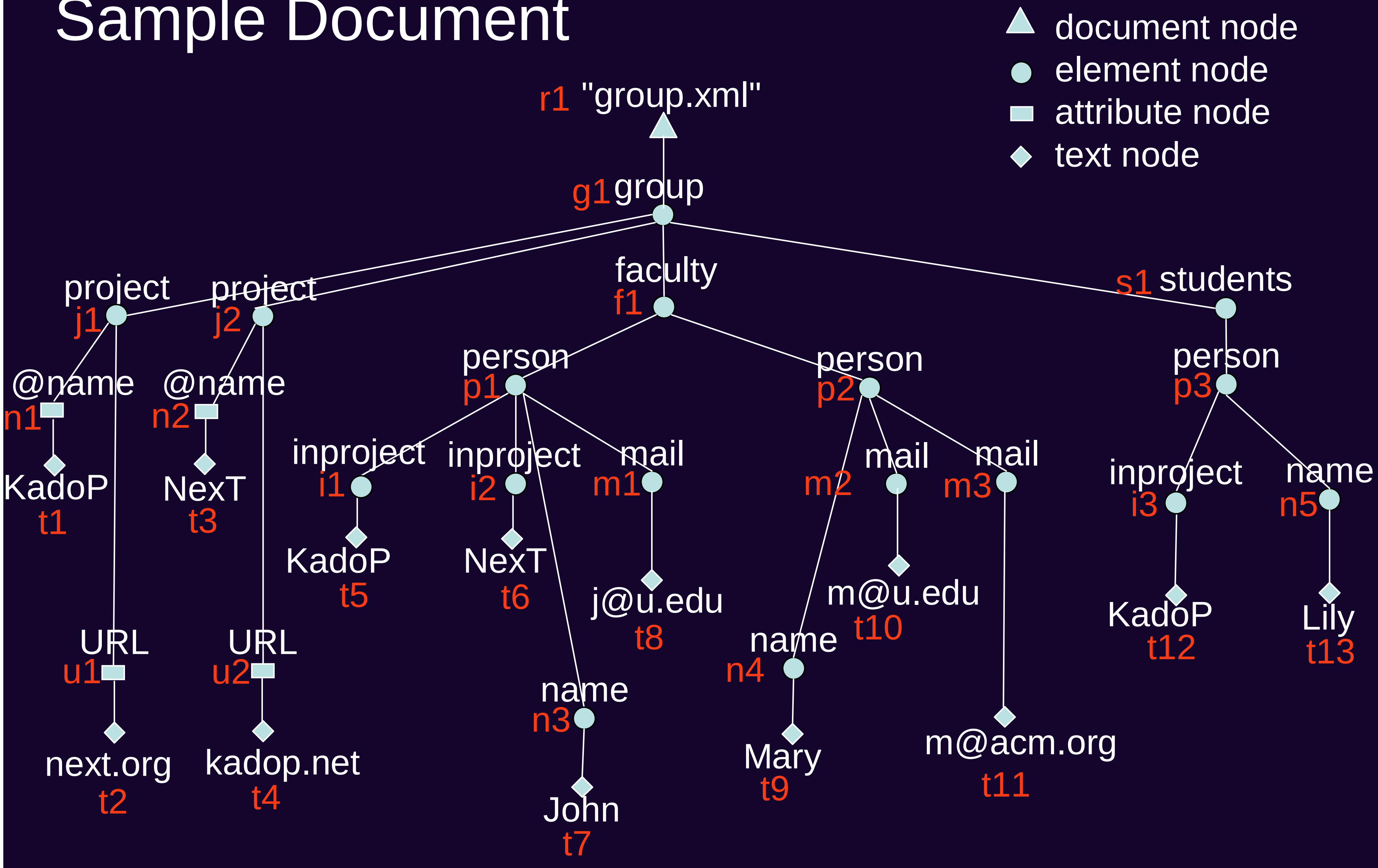
Node order relationship << (XQuery *before*) in ordered

- Parent before children
- Attribute nodes directly follow parent (precede non-attribute nodes)
- Undefined order between attributes

Node and Value Comparisons

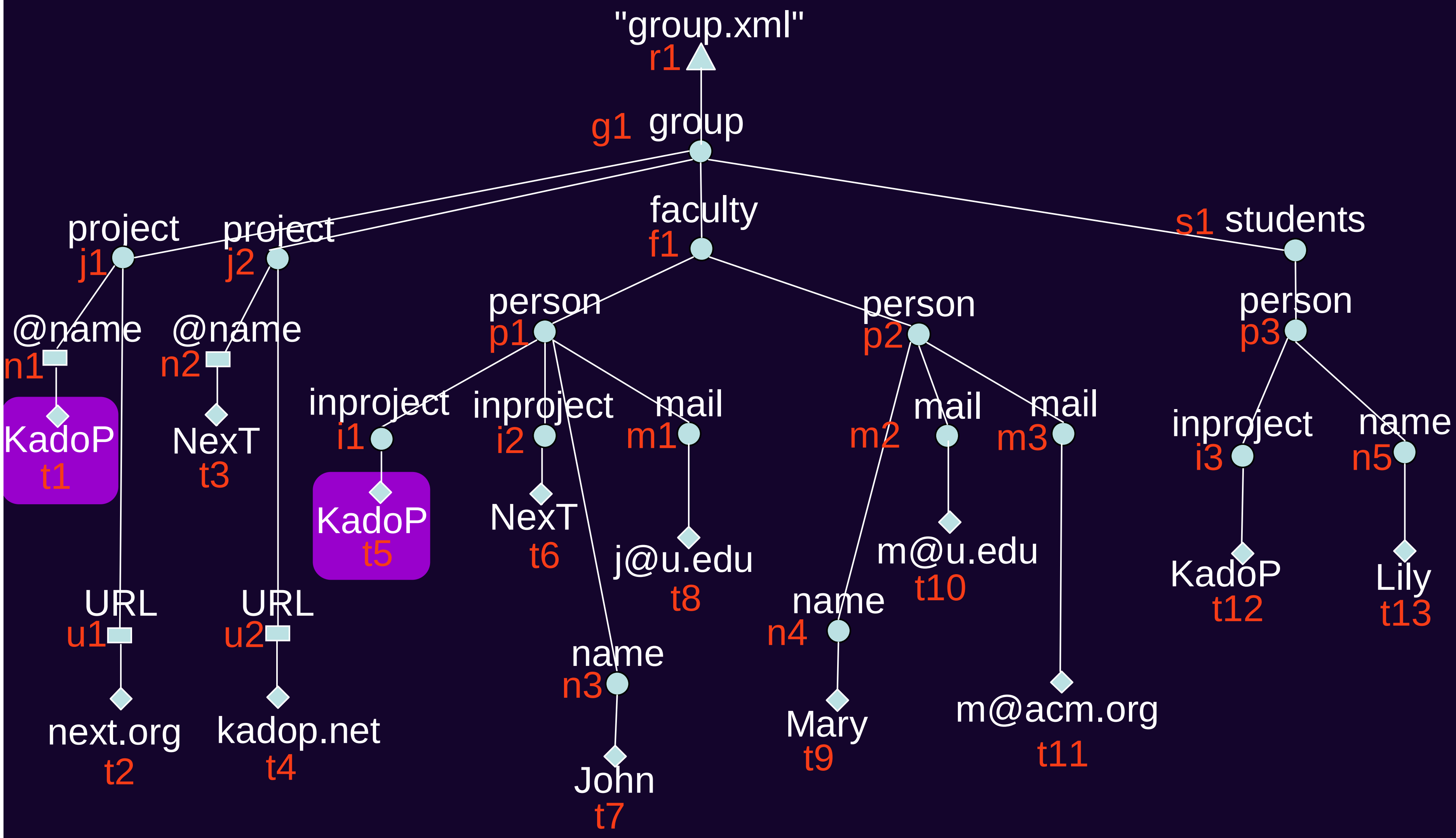


Sample Document



Equality Comparisons

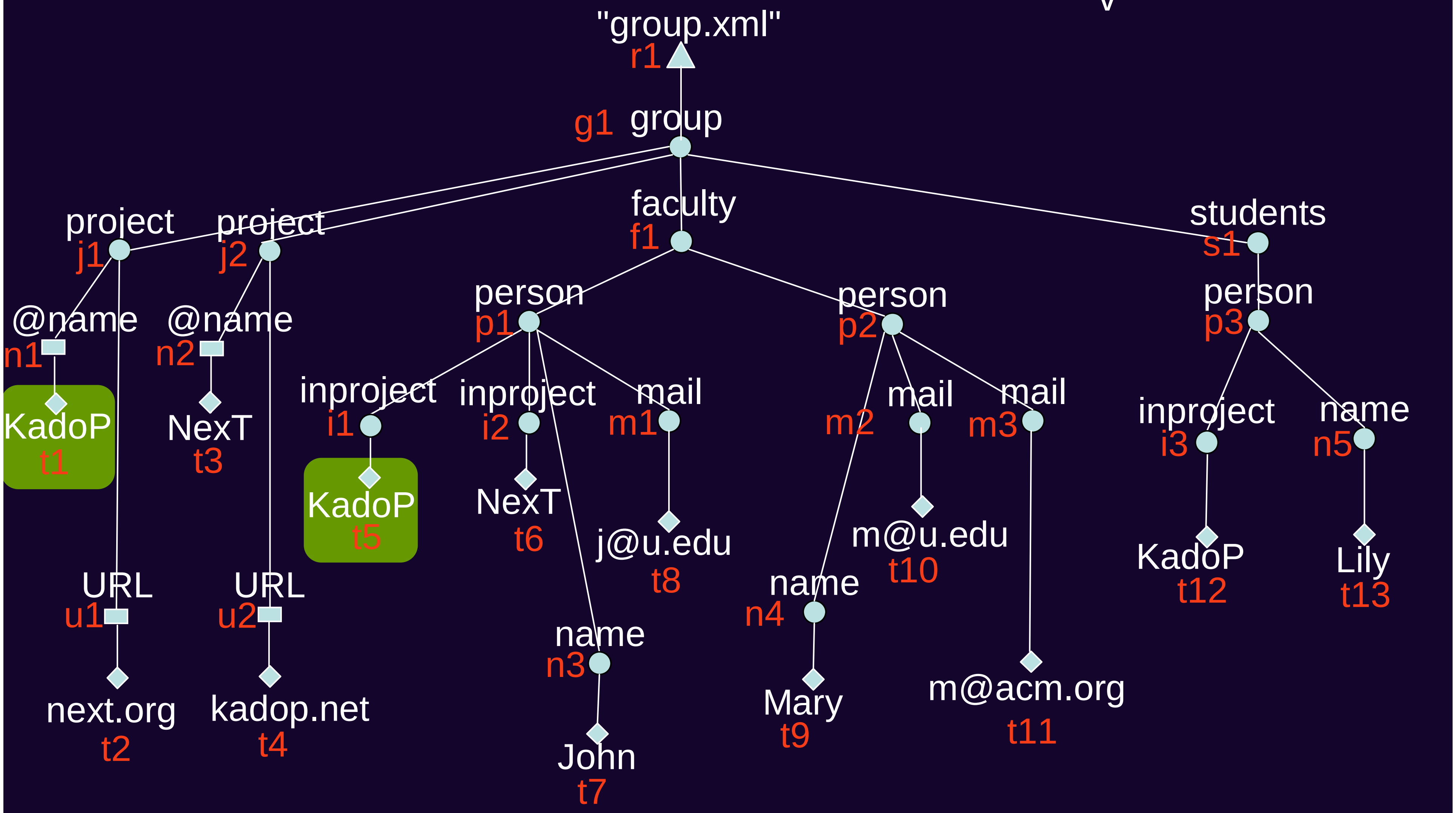
$t1 \neq_{id} t5$



Equality Comparisons

$t1 \neq_{id} t5$

$t1 =_{v} t5$

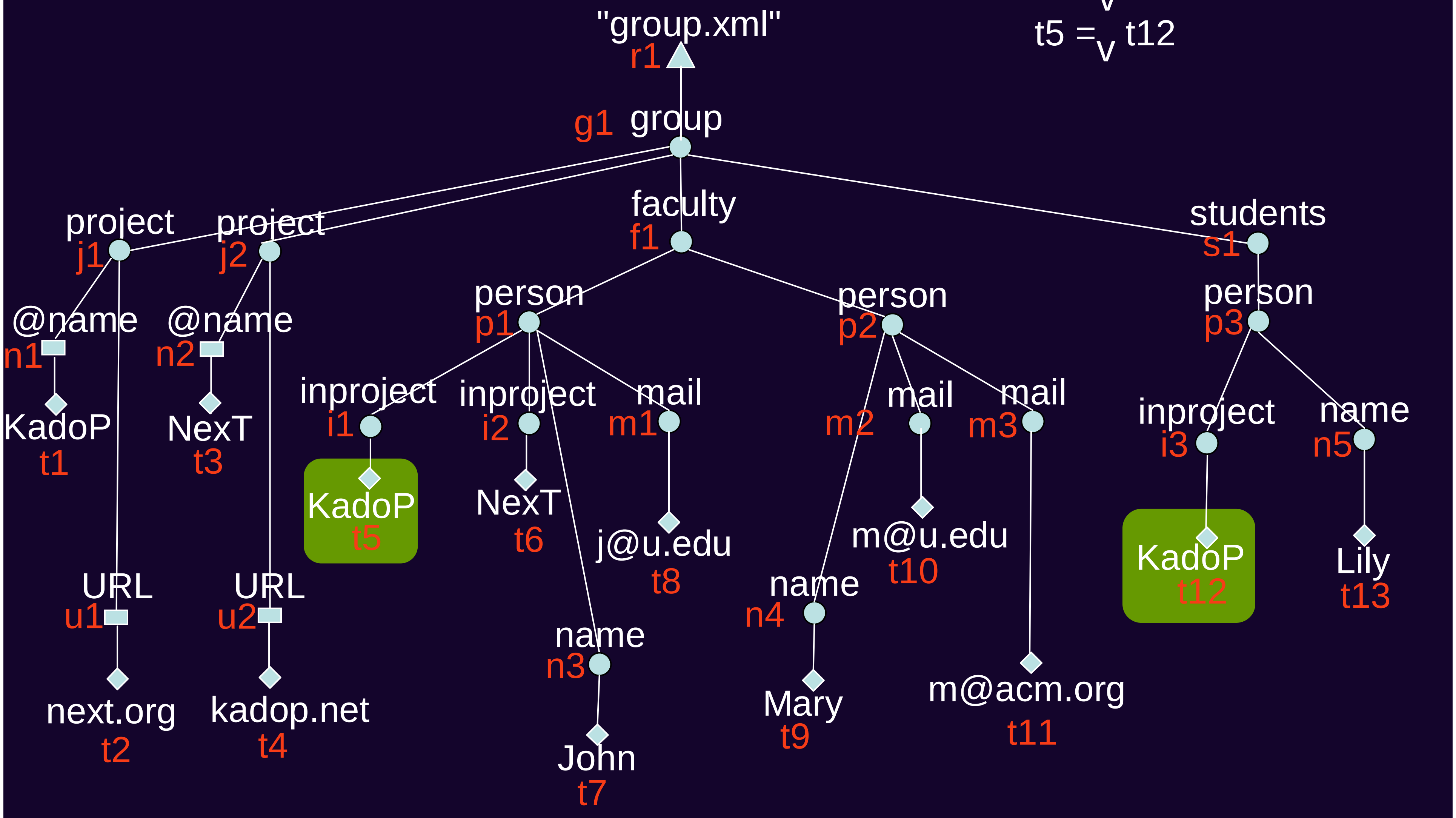


Equality Comparisons

$t1 \neq_{id} t5$

$t1 =_{v} t5$

$t5 =_{v} t12$



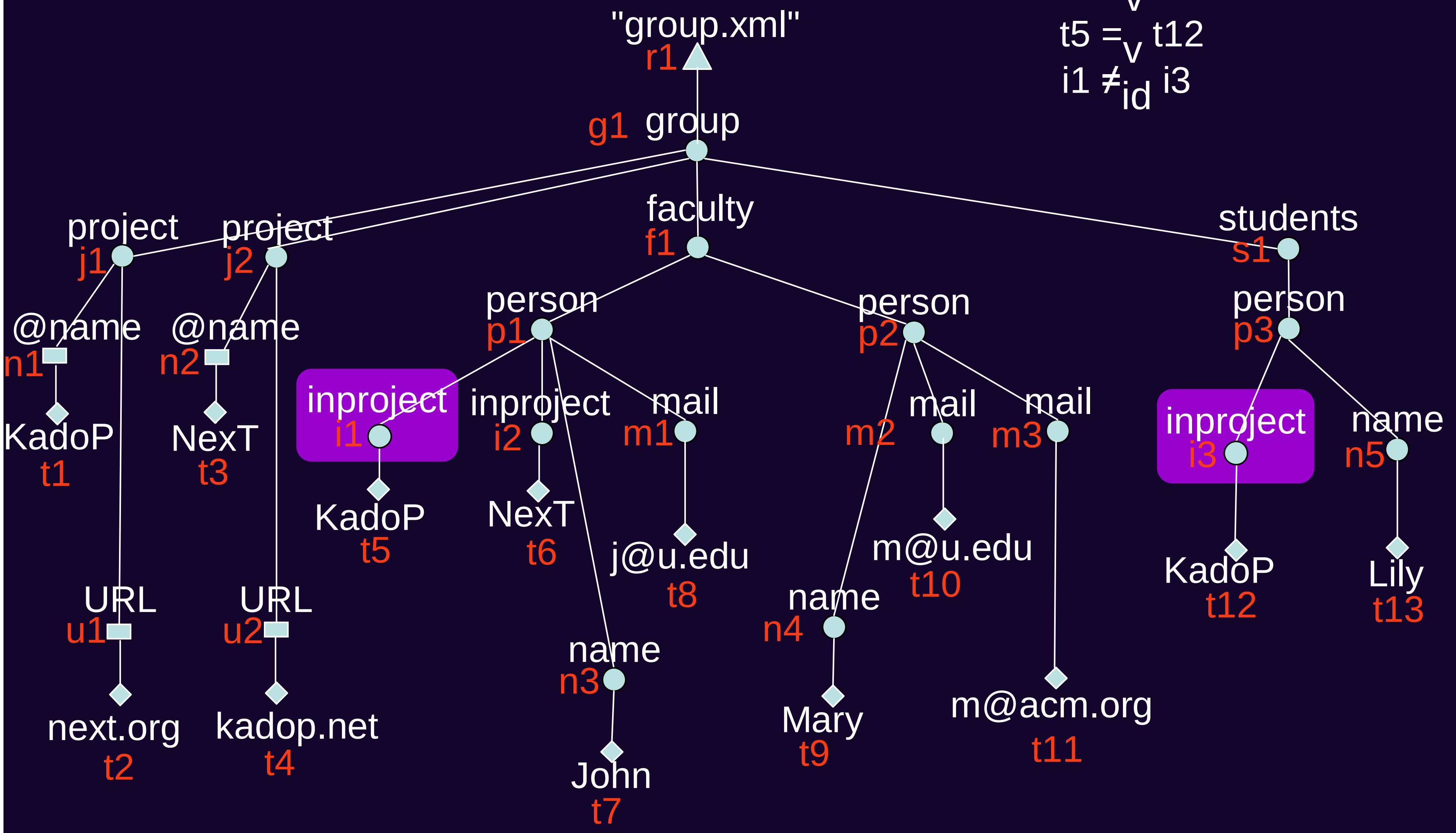
Equality Comparisons

$t1 \neq_{id} t5$

$t1 =_{v} t5$

$t5 =_{v} t12$

$i1 \neq_{id} i3$



Equality Comparisons

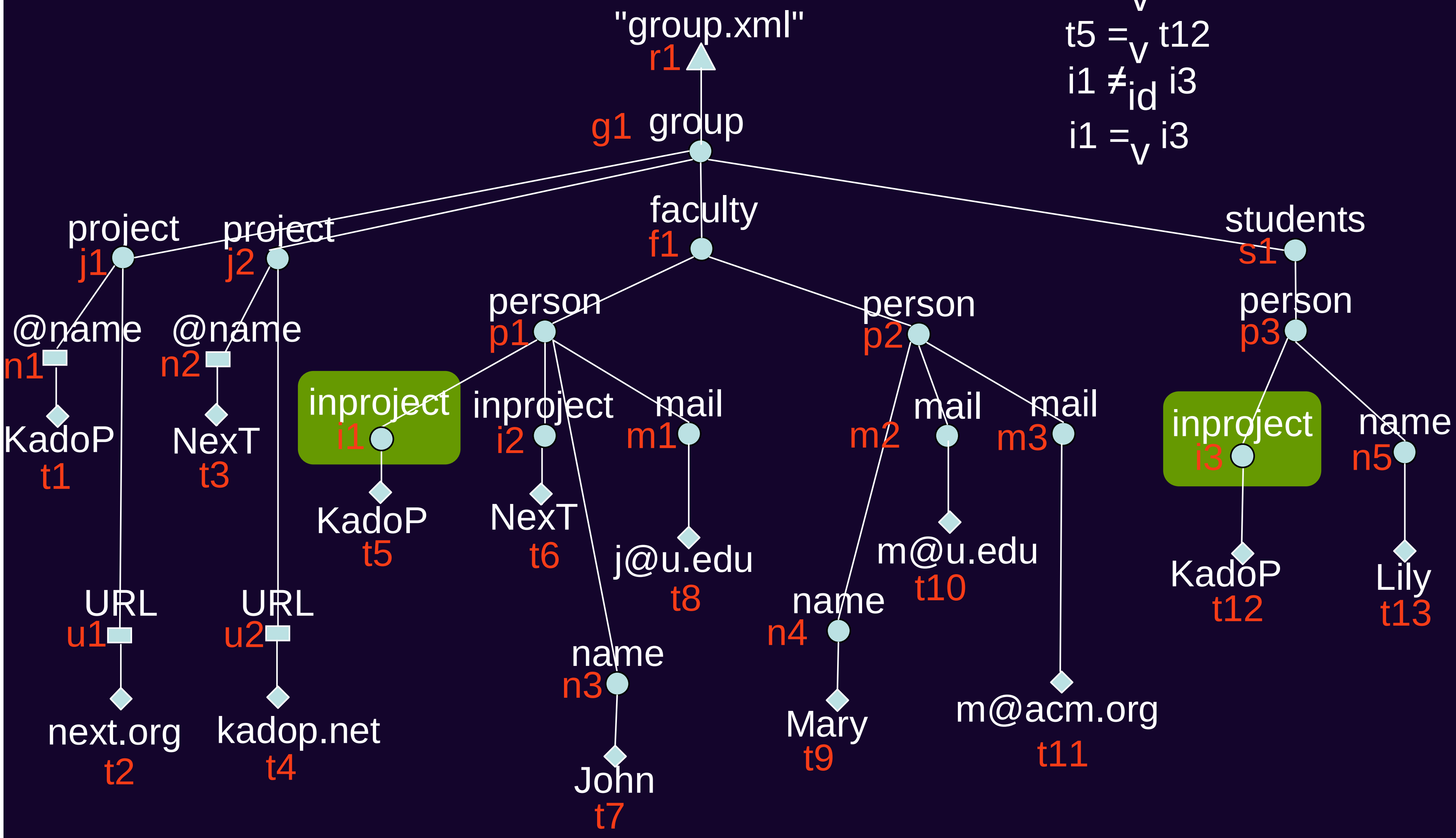
$t1 \neq_{id} t5$

$t1 =_{v} t5$

$t5 =_{v} t12$

$i1 \neq_{id} i3$

$i1 =_{v} i3$



Equality Comparisons

$t1 \neq_{id} t5$

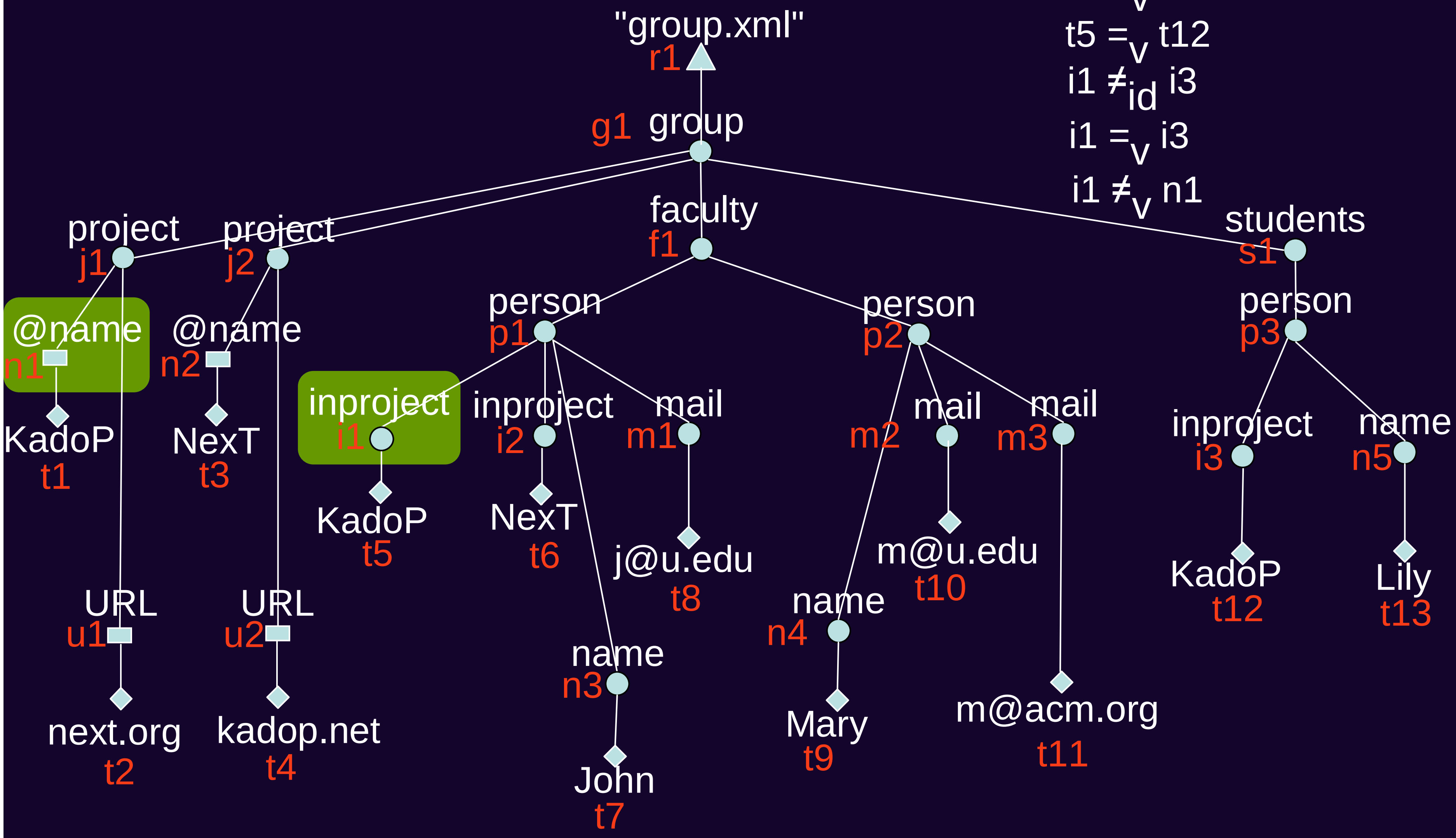
$t1 =_{v} t5$

$t5 =_{v} t12$

$i1 \neq_{id} i3$

$i1 =_{v} i3$

$i1 \neq_{v} n1$

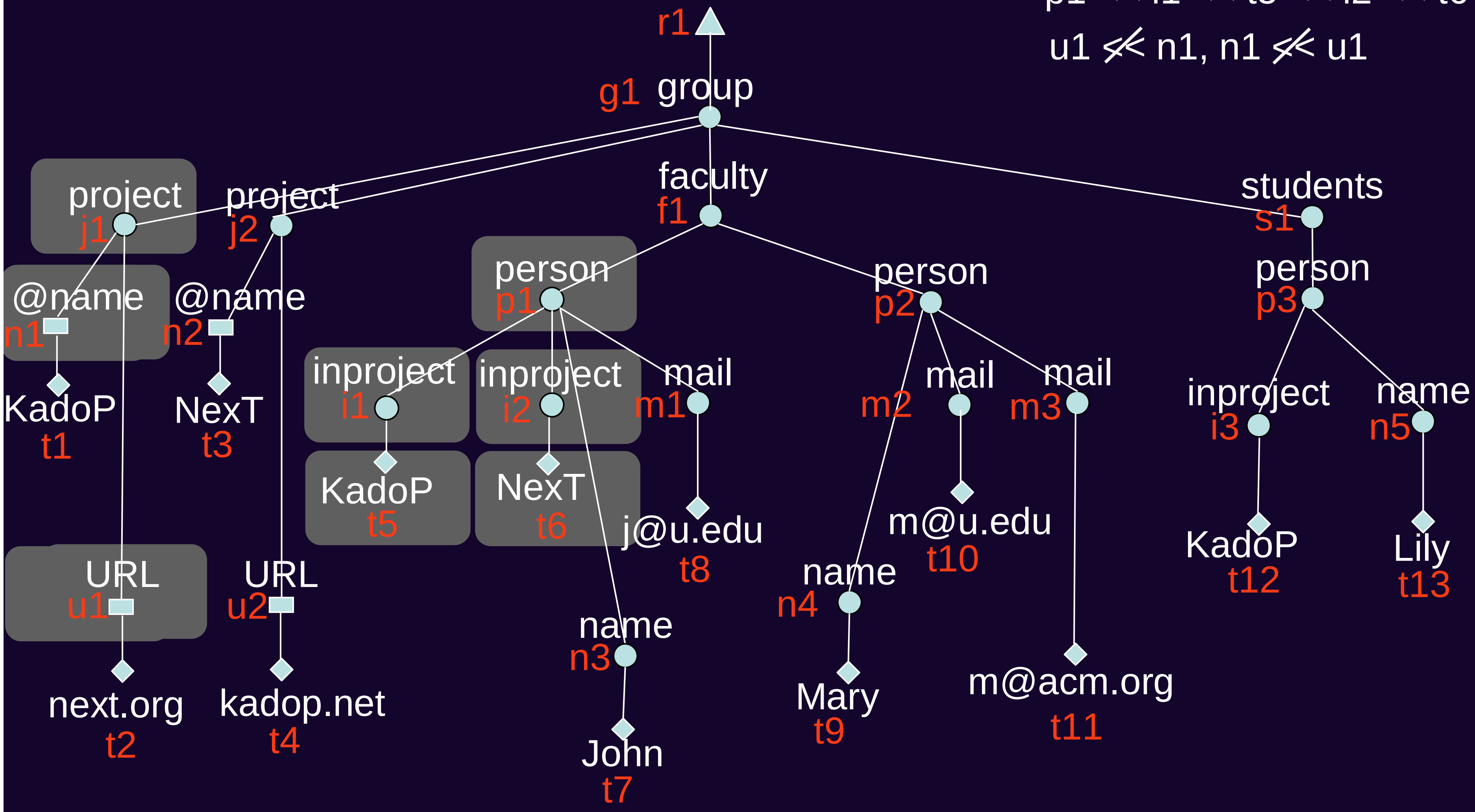


Order Comparisons

$j1 \ll n1,\ j1 \ll u1$

$p1 \ll i1 \ll t5 \ll i2 \ll t6$

$u1 \not\ll n1,\ n1 \not\ll u1$



Equalities on XQDMA Lists

Deep-equal =

- Two lists are deep-equal if they have the same length and their items at corresponding positions are value-based equal
- The "=" of XQuery does not translates to existential equality comparison

Existential list equality (*ele*) comparison $=_\exists$

- l1 $=_\exists$ l2 iff $\exists$ o1 $\in$ l1, o2 $\in$ l2 such that o1 $=_v$ o2
- *ele* not transitive

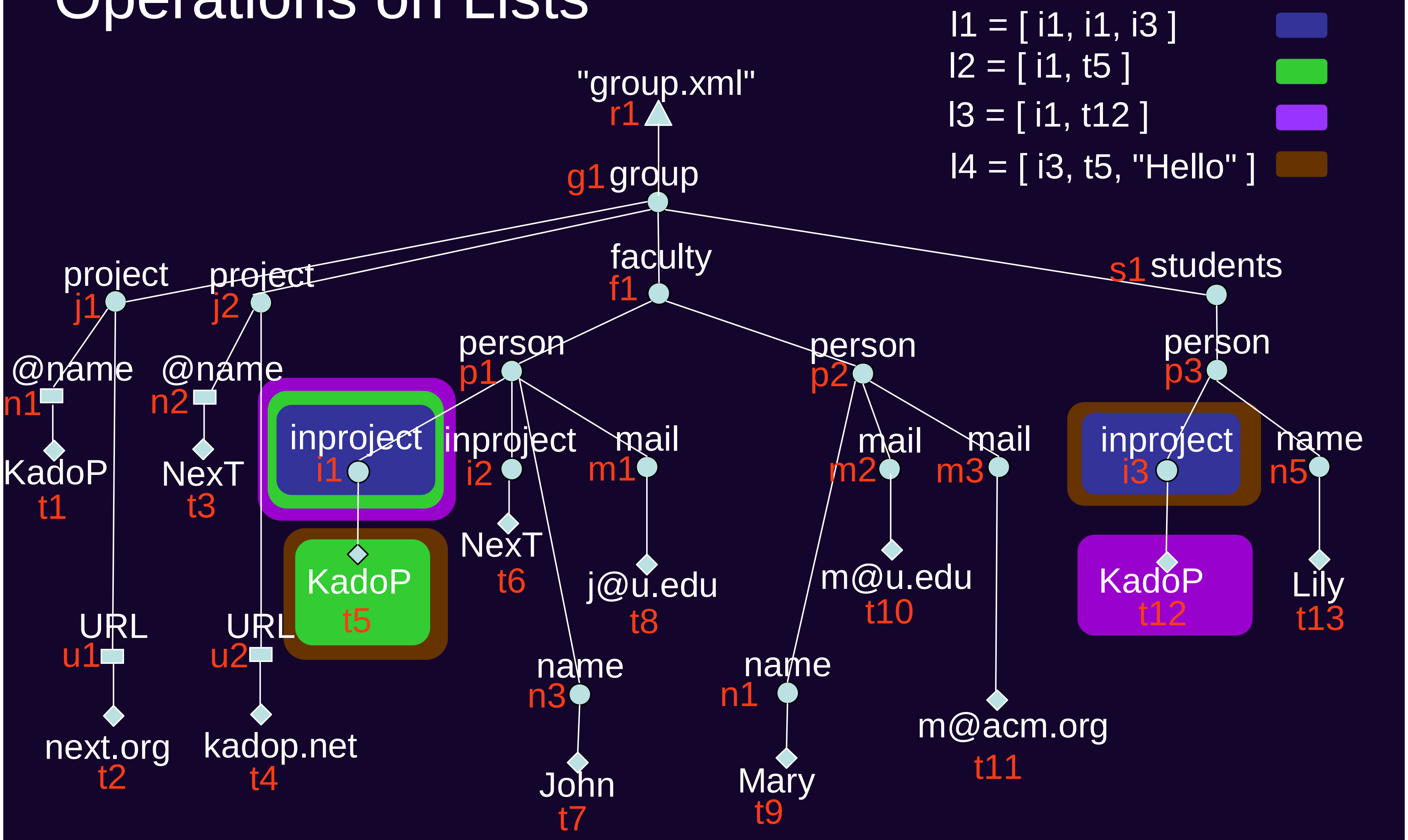
Generalization to Other Operators

Other relationships over nodes and lists

≠, >, <, ...

In limited value-based comparisons are induced by corresponding relationships on values

Operations on Lists



Operations on Lists

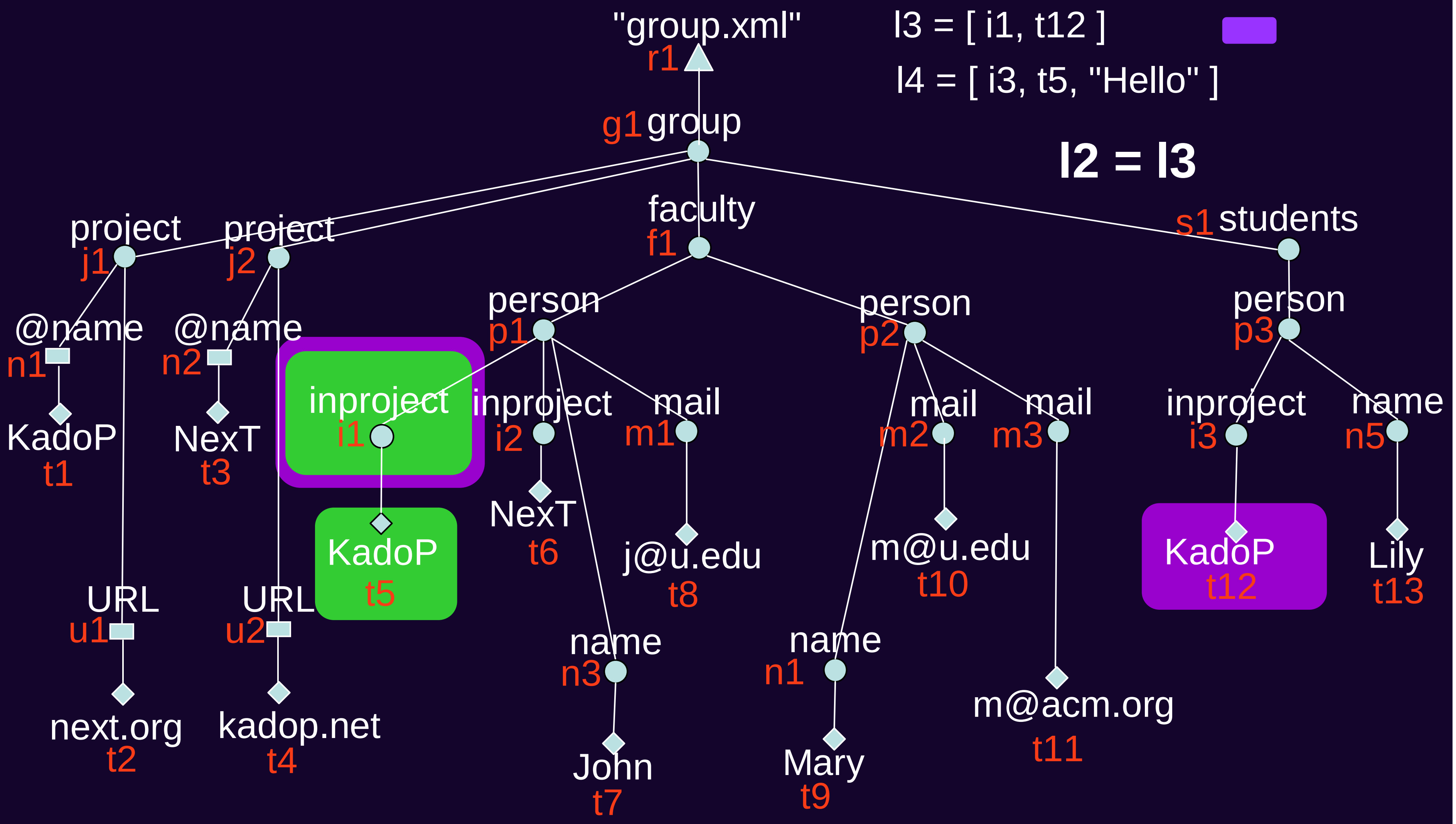
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

l2 = l3



Operations on Lists

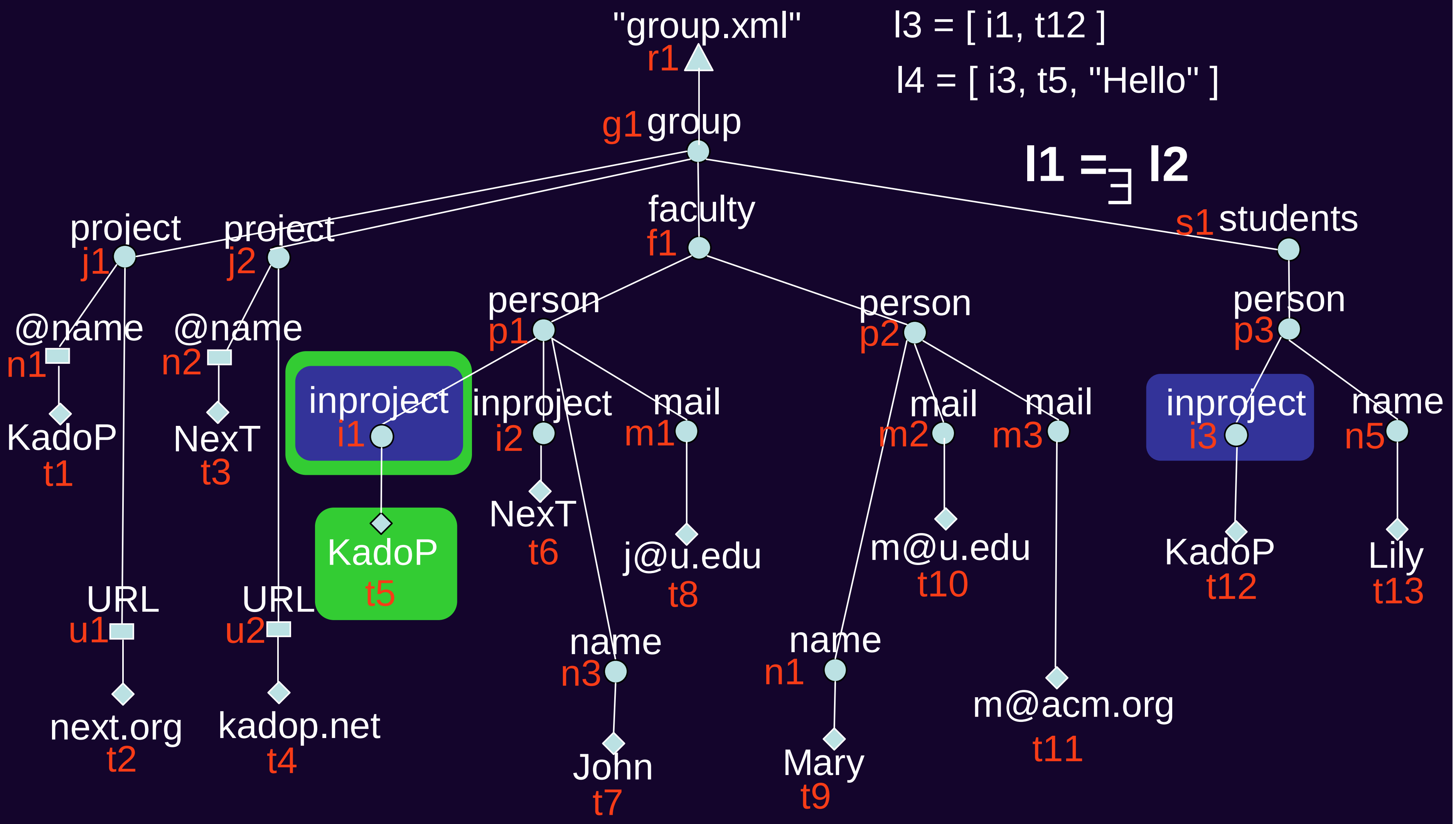
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

$$l1 =_{\exists} l2$$



Operations on Lists

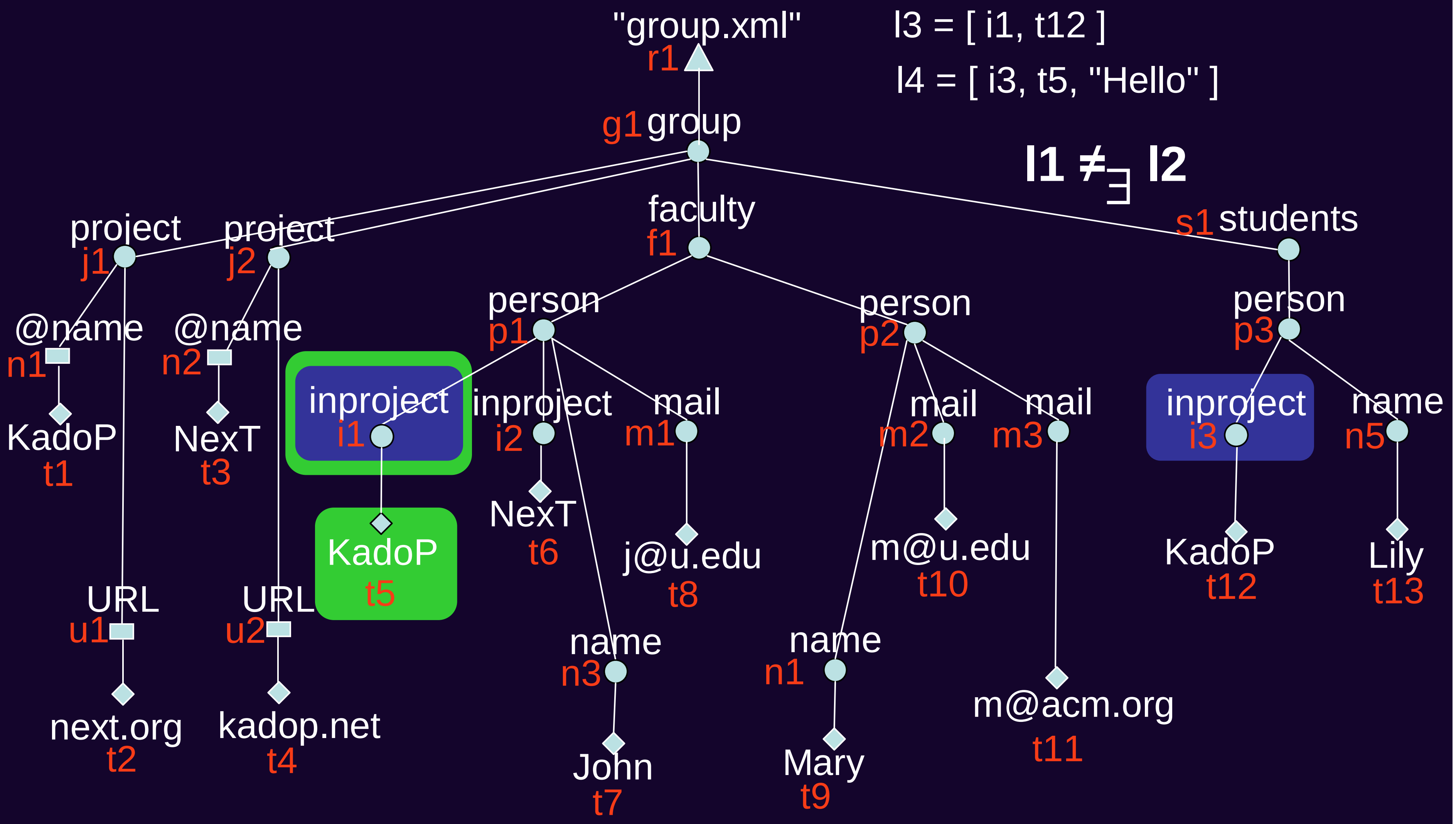
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

l1 $\neq_{\exists}$ l2



Operations on Lists

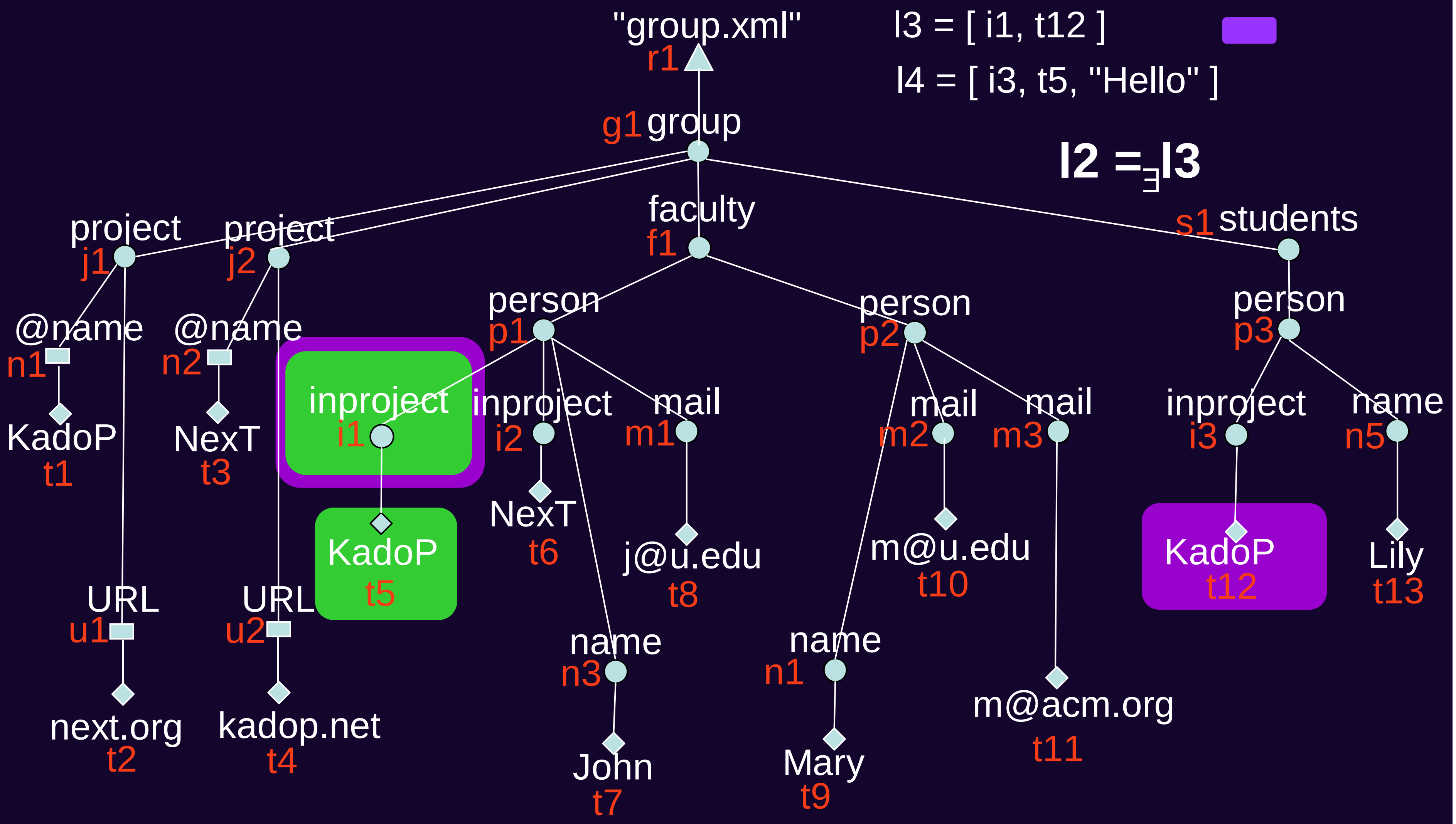
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

$l2 =_{\exists} l3$



Operations on Lists

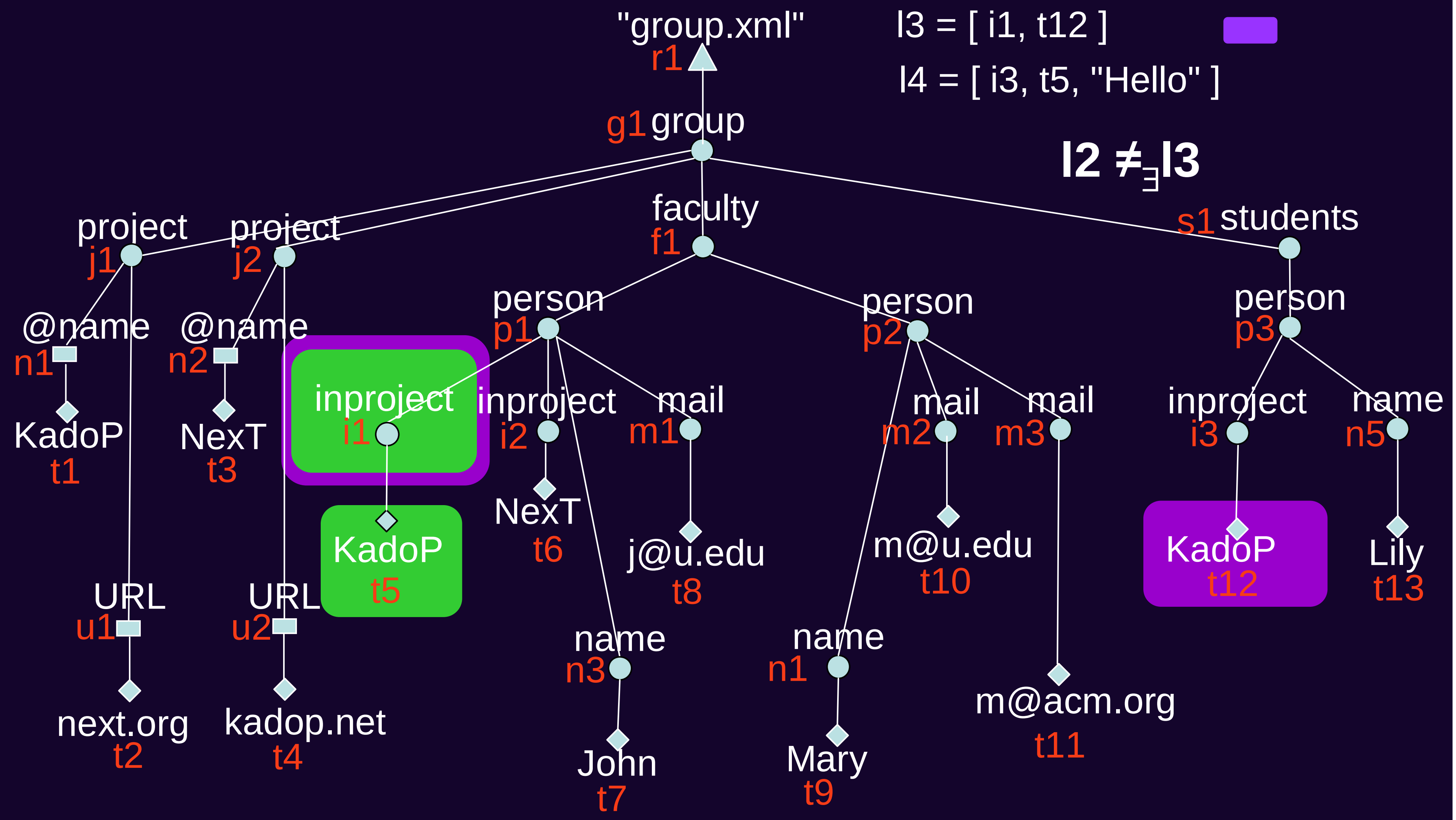
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

$$l2 \neq_{\exists} l3$$



Operations on Lists

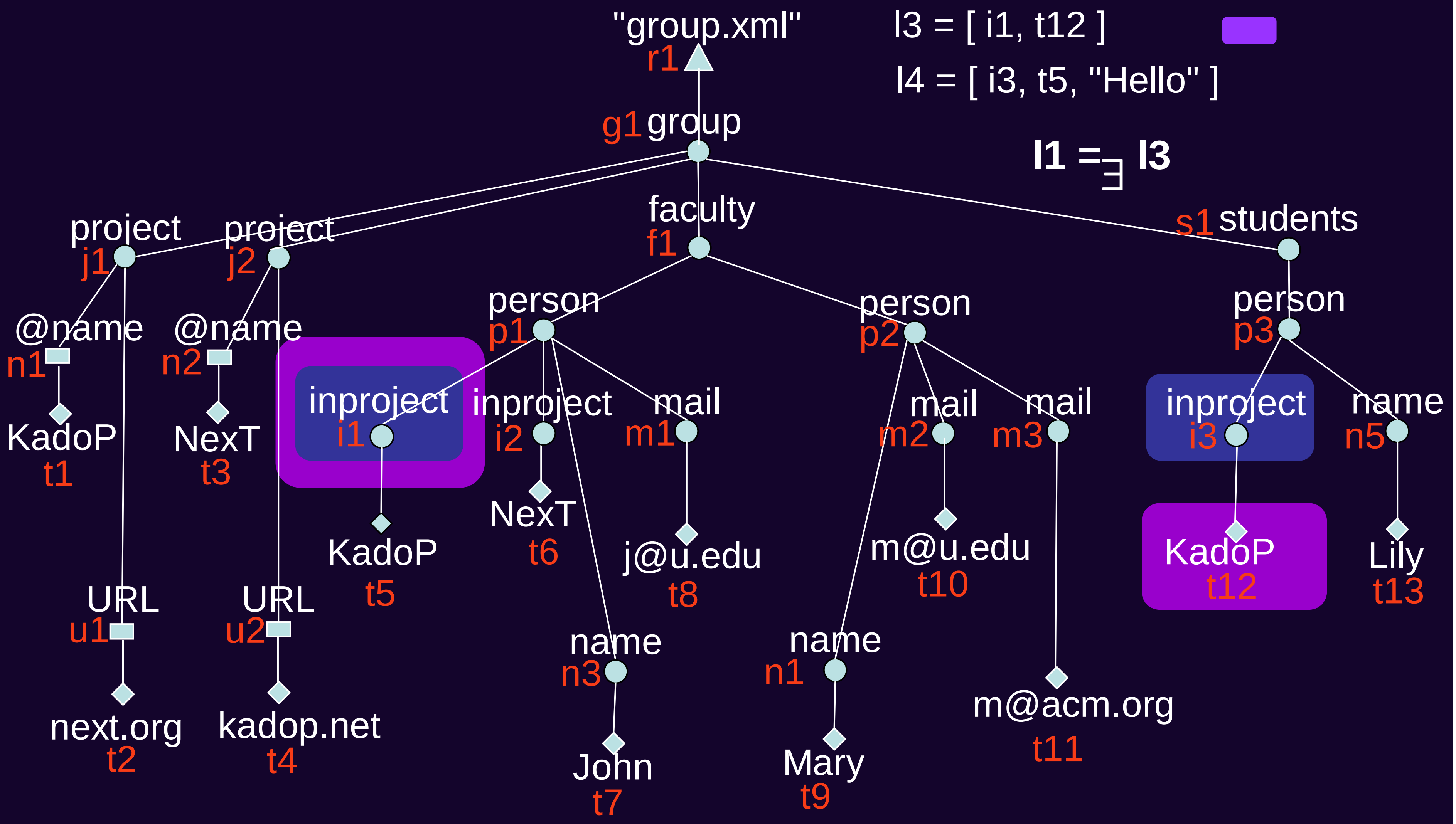
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

l1 $=_{\exists}$ l3



Operations on Lists

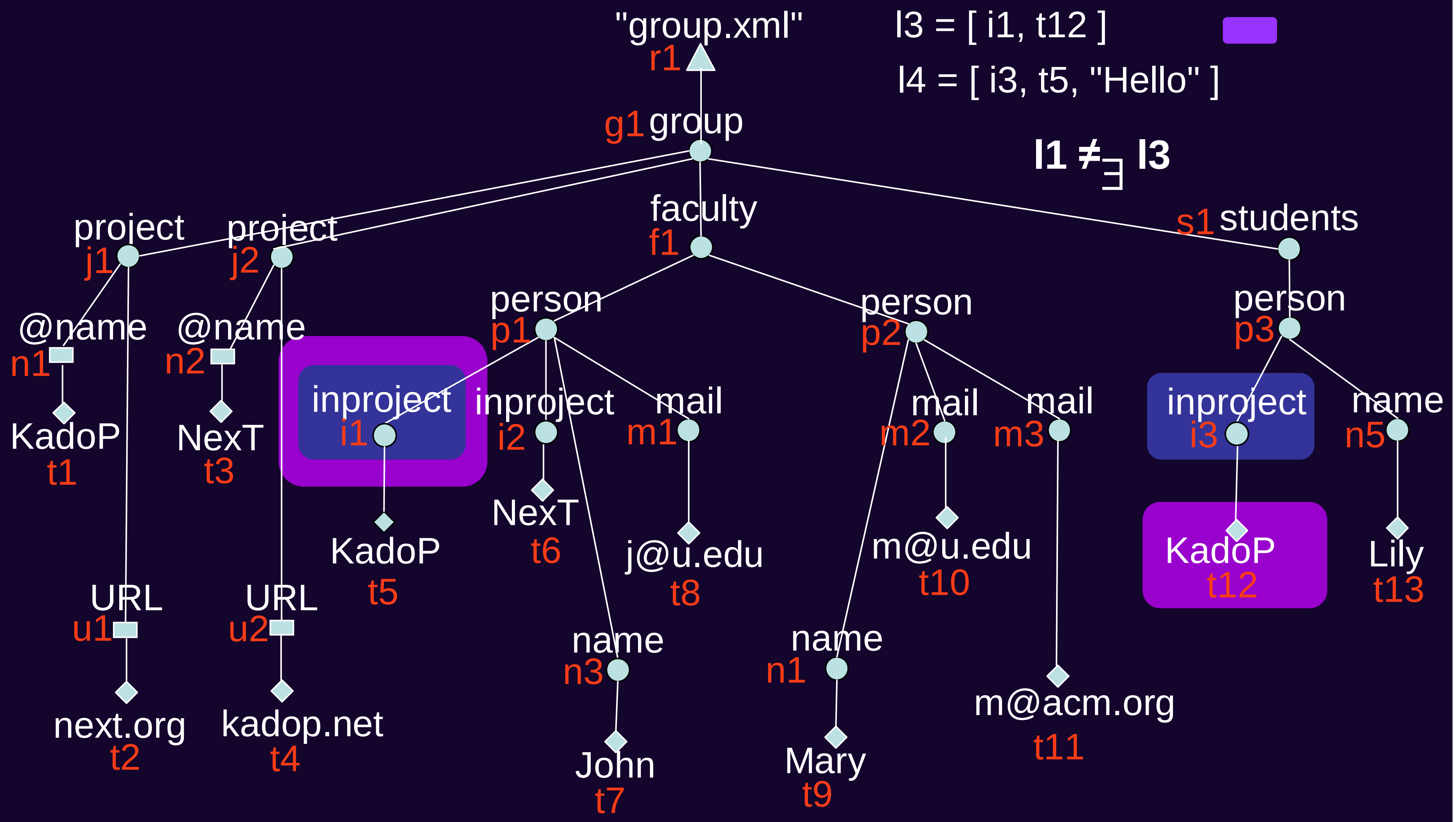
l1 = [i1, i1, i3]

l2 = [i1, t5]

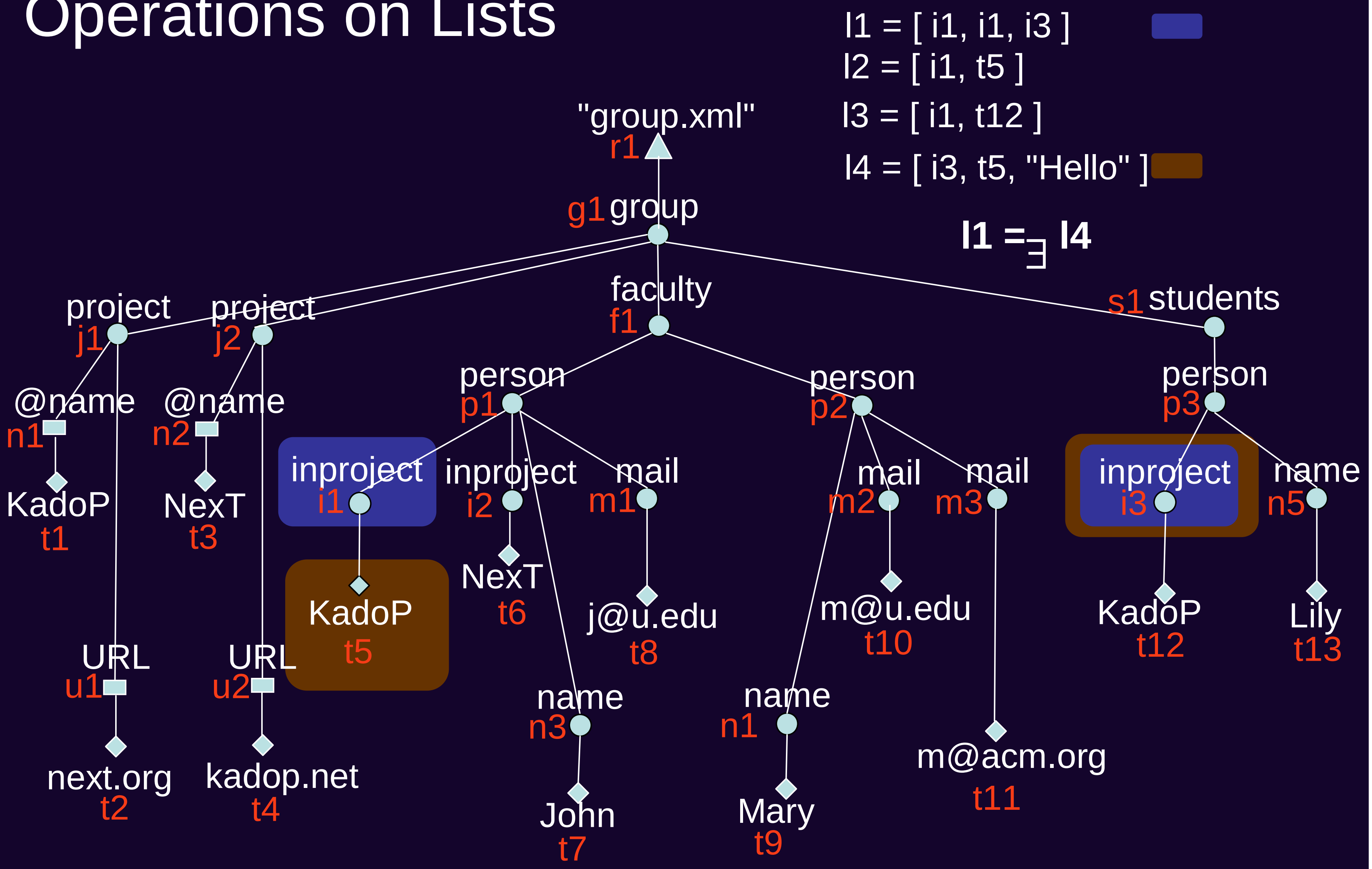
l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

l1 $\neq_{\exists}$ l3

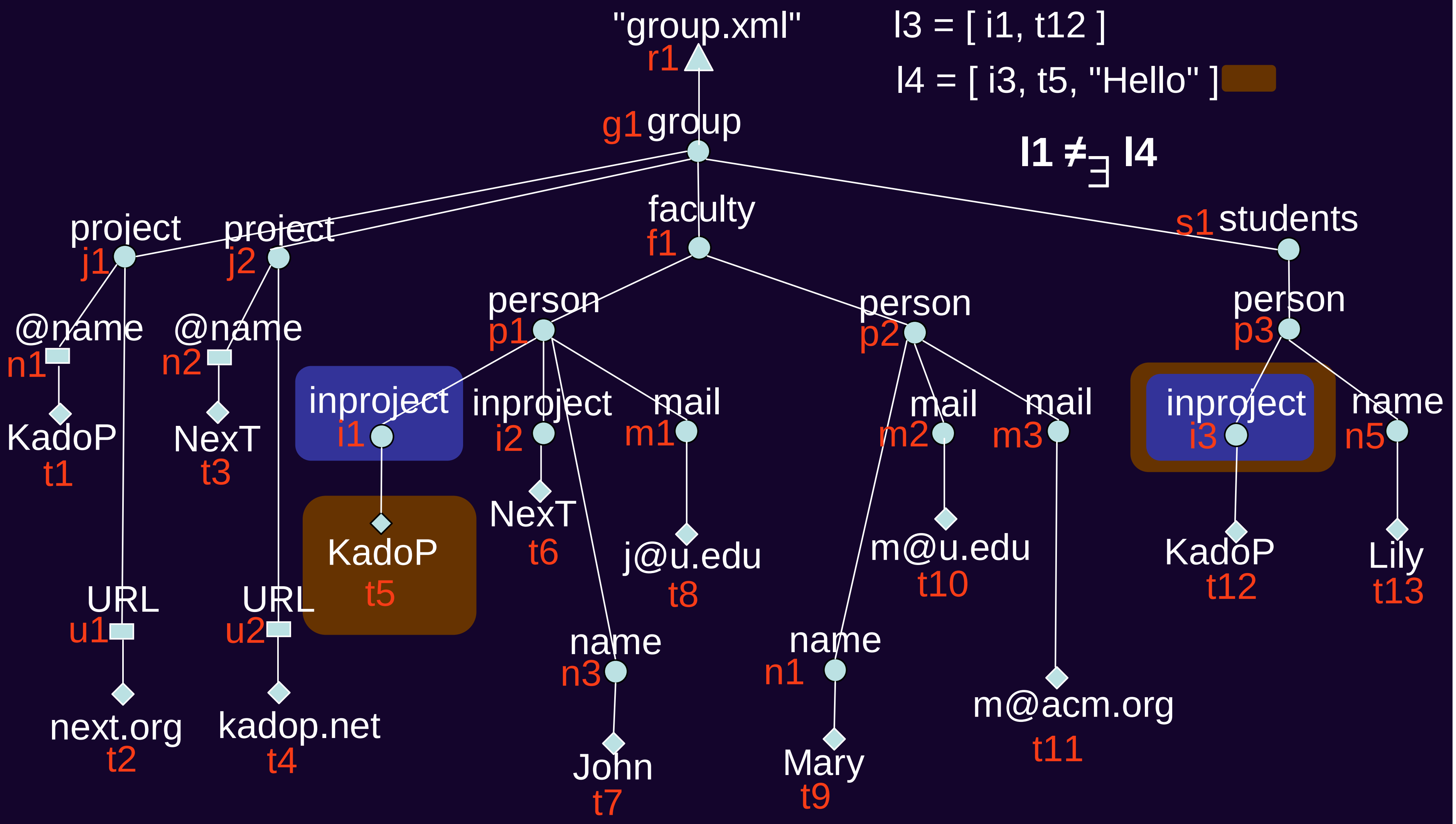


Operations on Lists



Operations on Lists

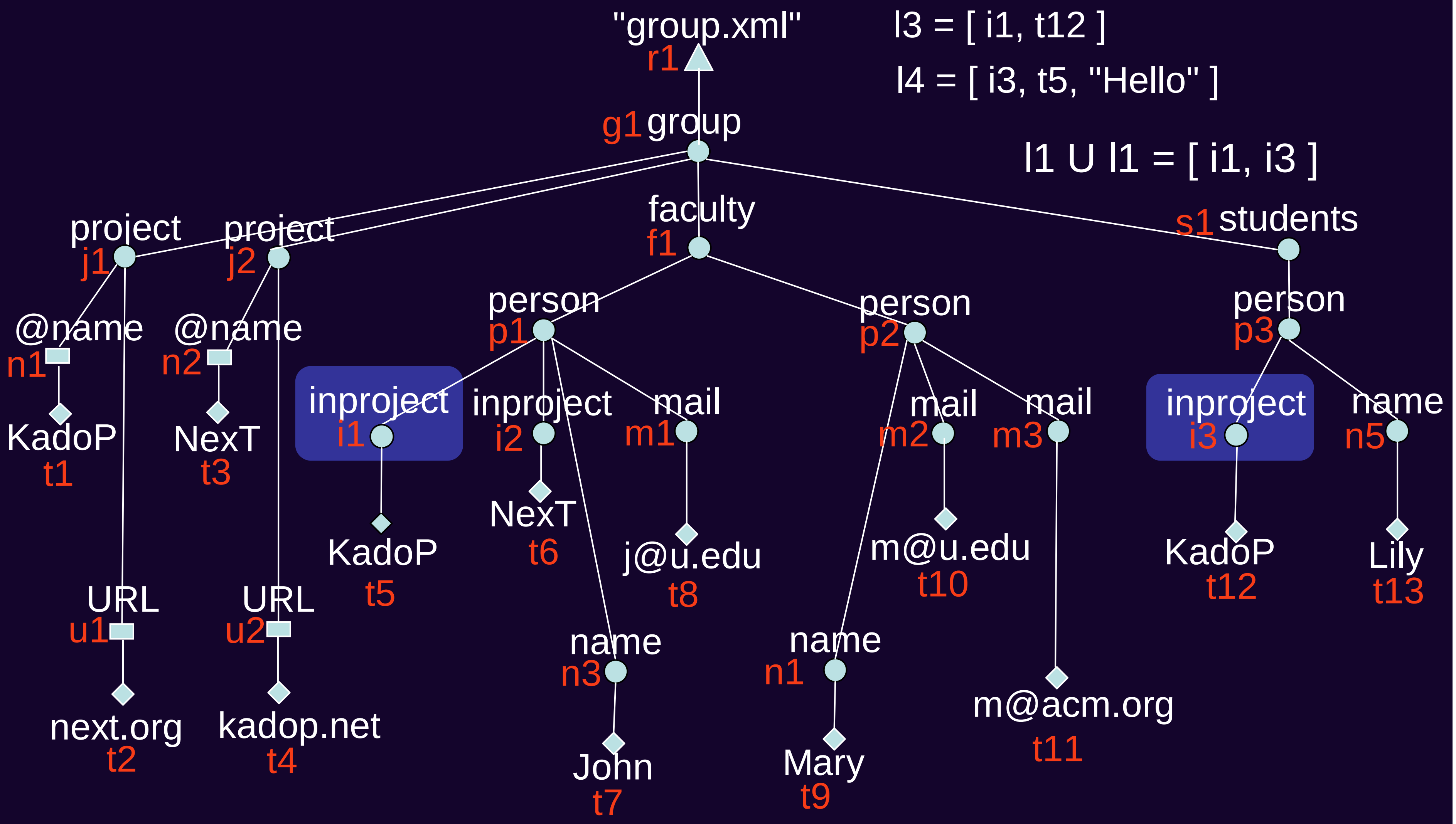
l1 = [i1, i1, i3]
l2 = [i1, t5]
l3 = [i1, t12]
l4 = [i3, t5, "Hello"]



Operations on Lists

l1 = [i1, i1, i3]
l2 = [i1, t5]
l3 = [i1, t12]
l4 = [i3, t5, "Hello"]

l1 U l1 = [i1, i3]



Operations on Lists

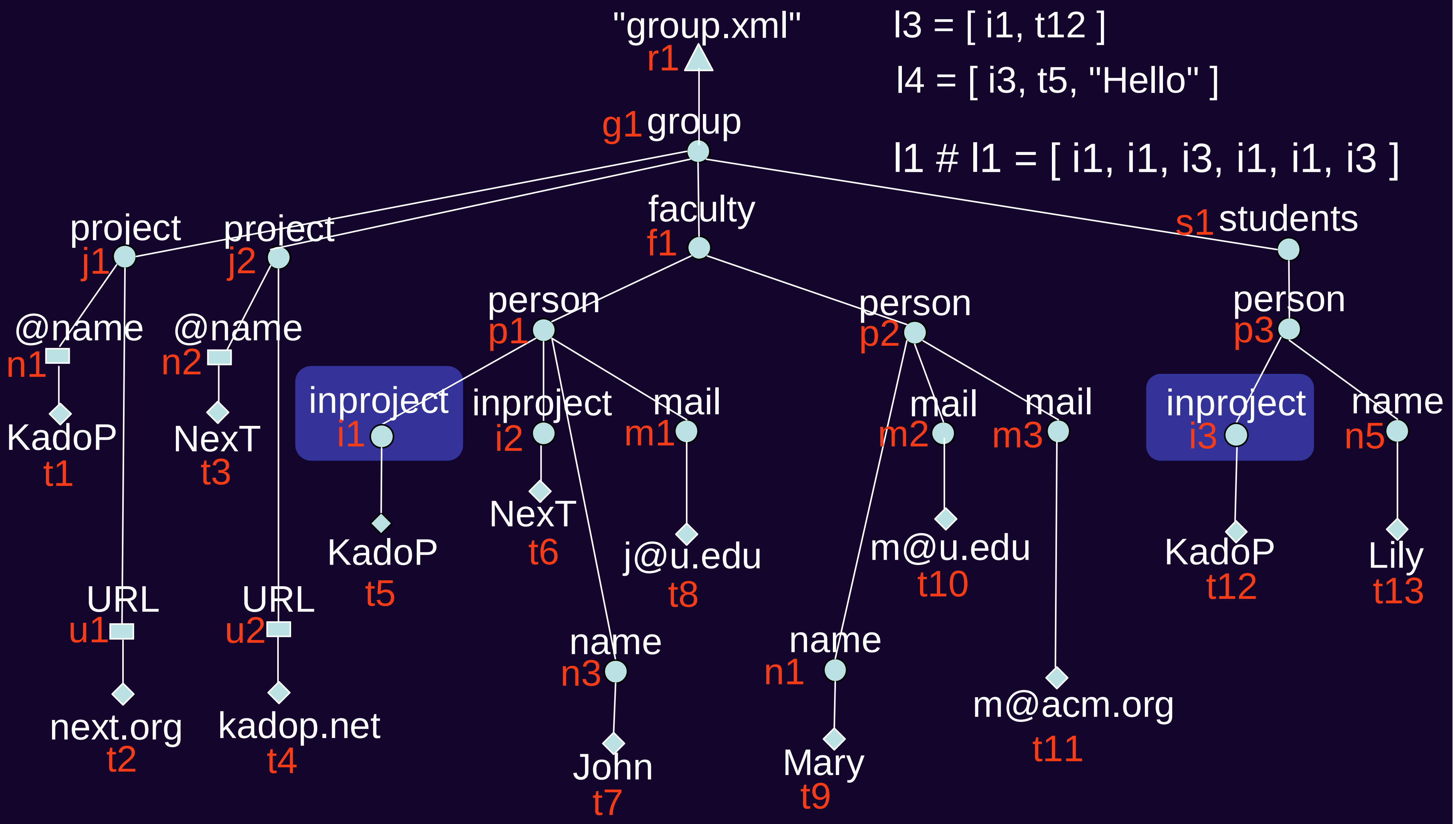
l1 = [i1, i1, i3]

l2 = [i1, t5]

l3 = [i1, t12]

l4 = [i3, t5, "Hello"]

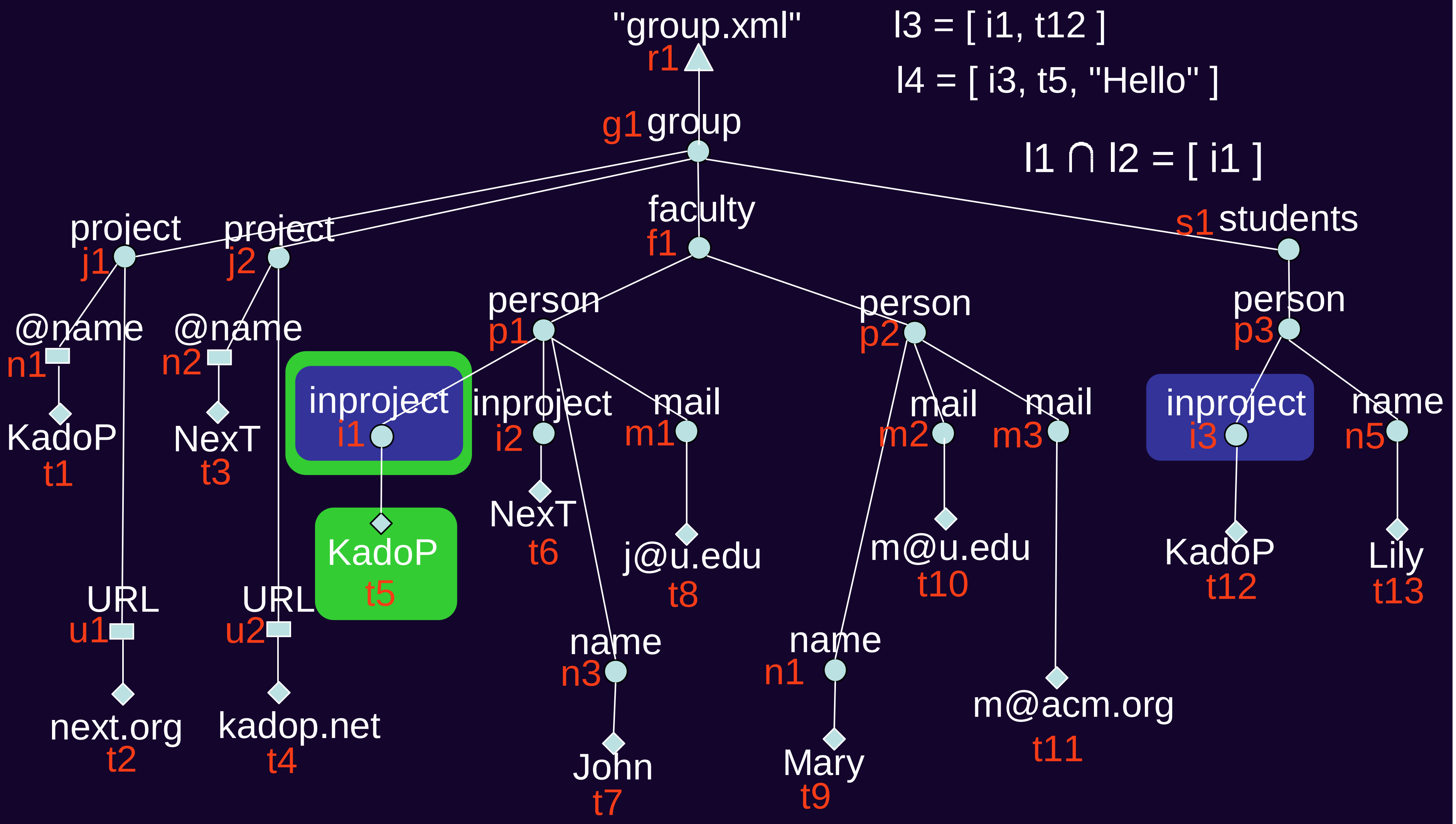
l1 # l1 = [i1, i1, i3, i1, i1, i3]



Operations on Lists

l1 = [i1, i1, i3]
l2 = [i1, t5]
l3 = [i1, t12]
l4 = [i3, t5, "Hello"]

l1 ∩ l2 = [i1]

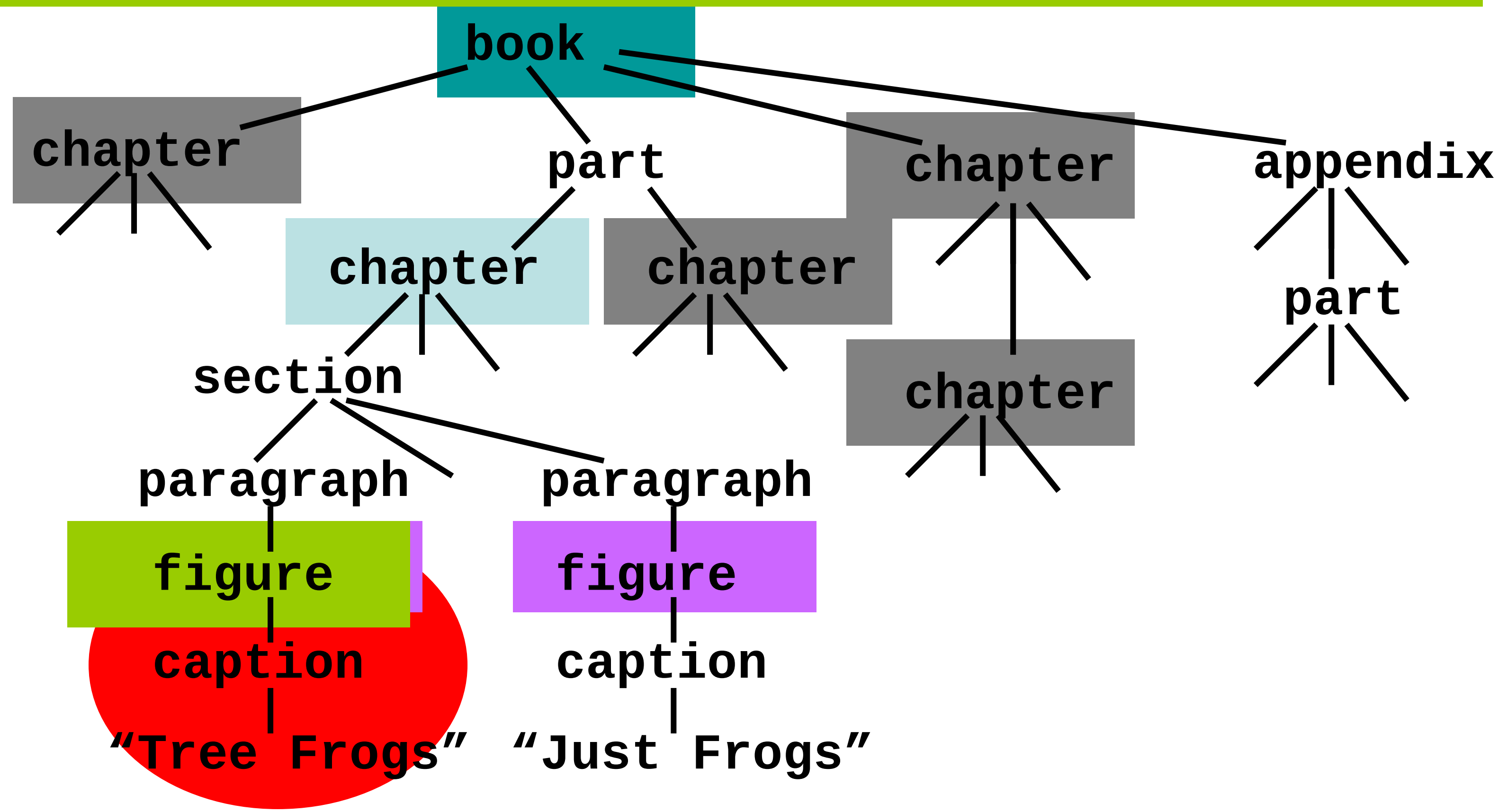


A Quick Refresh of Your XPath and XQuery

Path Expressions

In the second `chapter` of the document `zoo.xml` find the figures with caption “Tree Frogs”

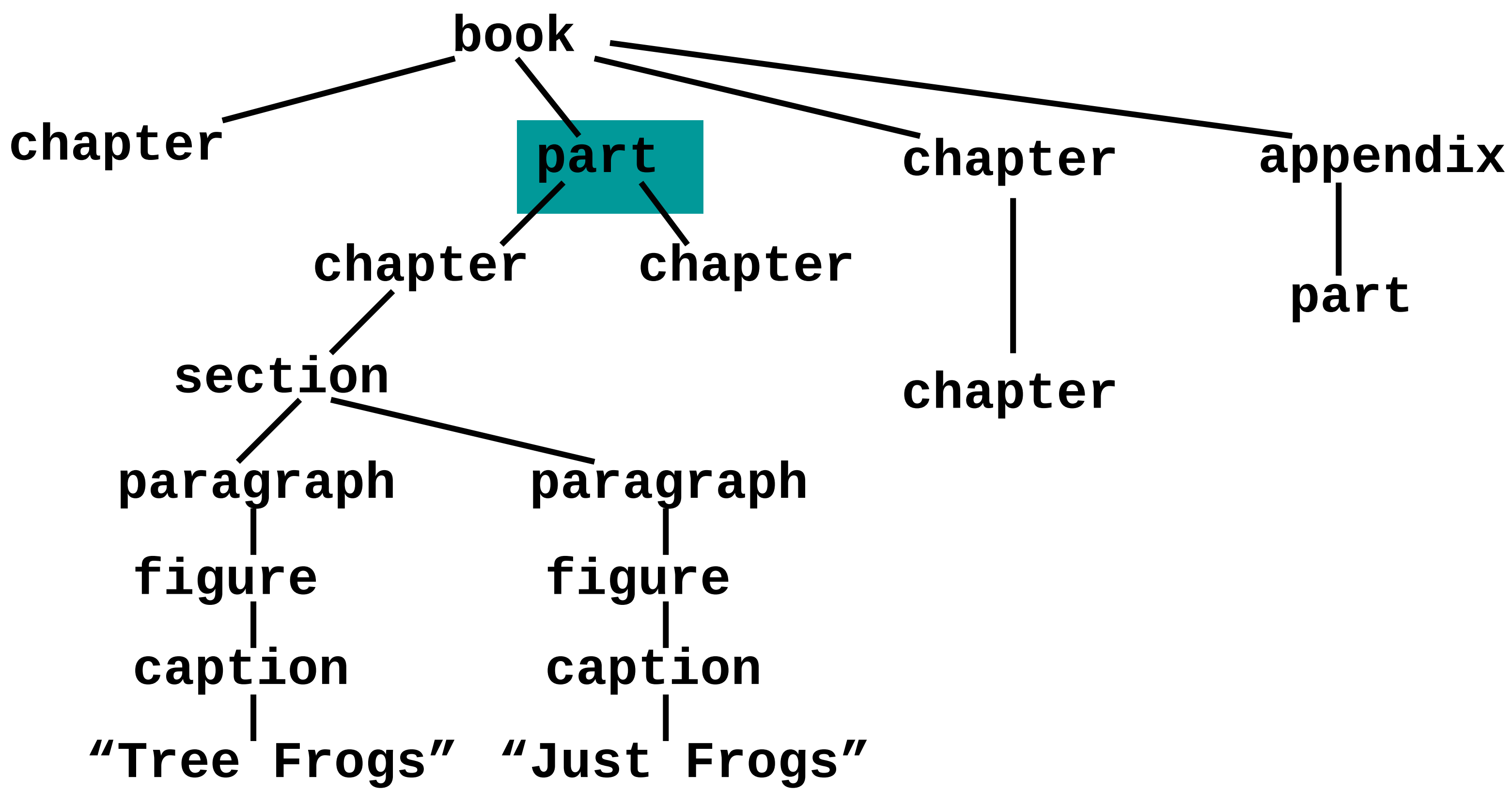
`./book//chapter[2]//figure[caption=“tree frogs”]`



More Path Expressions

Find the first immediate `chapter` subelements of immediate `part` subelements of the document `zoo.xml` and retrieve figures that have …

```
doc("zoo.xml")/part/chapter[1]//figure[caption="Tree Frogs"]
```



Element Construction

In the second chapter of the document zoo.xml find the figures with caption "Tree Frogs" and place them into an element called result

```
<result>
 {
 doc(“zoo.xml”)//chapter[2]//figure[caption=“Tree Frogs”]
 }
</result>
```

```
     result
        |
     figure
        |
     caption
        |
  “Tree Frogs”
```

Bibliography Example Data Set

```
<bib>
 <book>
  <author> Aho </author>
  <author> Hopcroft </author>
  <author> Ullman </author>
  <title> Automata Theory </title>
  <publisher> Morgan Kaufmann </publisher>
  <year> 1998 >/year>
 </book>
 <book>
  <author> Ullman </author>
  <title> Database Systems </title>
  <publisher> Morgan Kaufmann </publisher>
  <year> 1998 >/year>
 </book>
 <book>
  <author> Abiteboul </author>
  <author> Buneman </author>
  <author> Suciu </author>
  <title> Automata Theory </title>
  <publisher> Prentice Hall </publisher>
  <year> 1998 >/year>
 </book>
</bib>
```

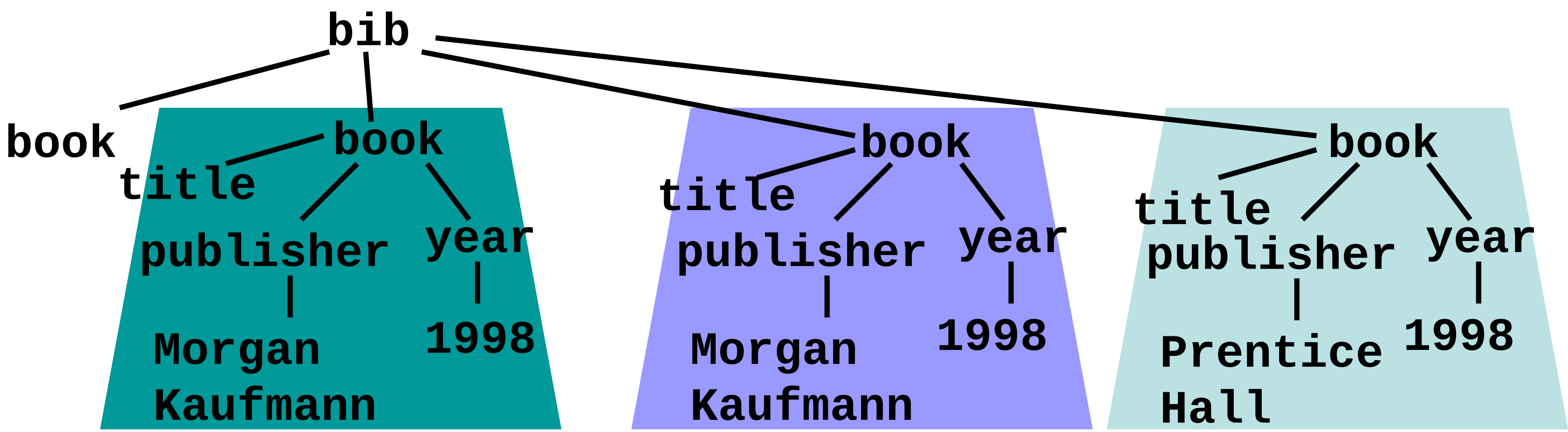
Reviews Example Data Set

```
<reviews>
 <review>
   <title> Automata Theory </title>
   <comment> It’s the best in automata theory </comment>
   <comment> A definitive textbook </comment>
 </review>
 …
</reviews>
```

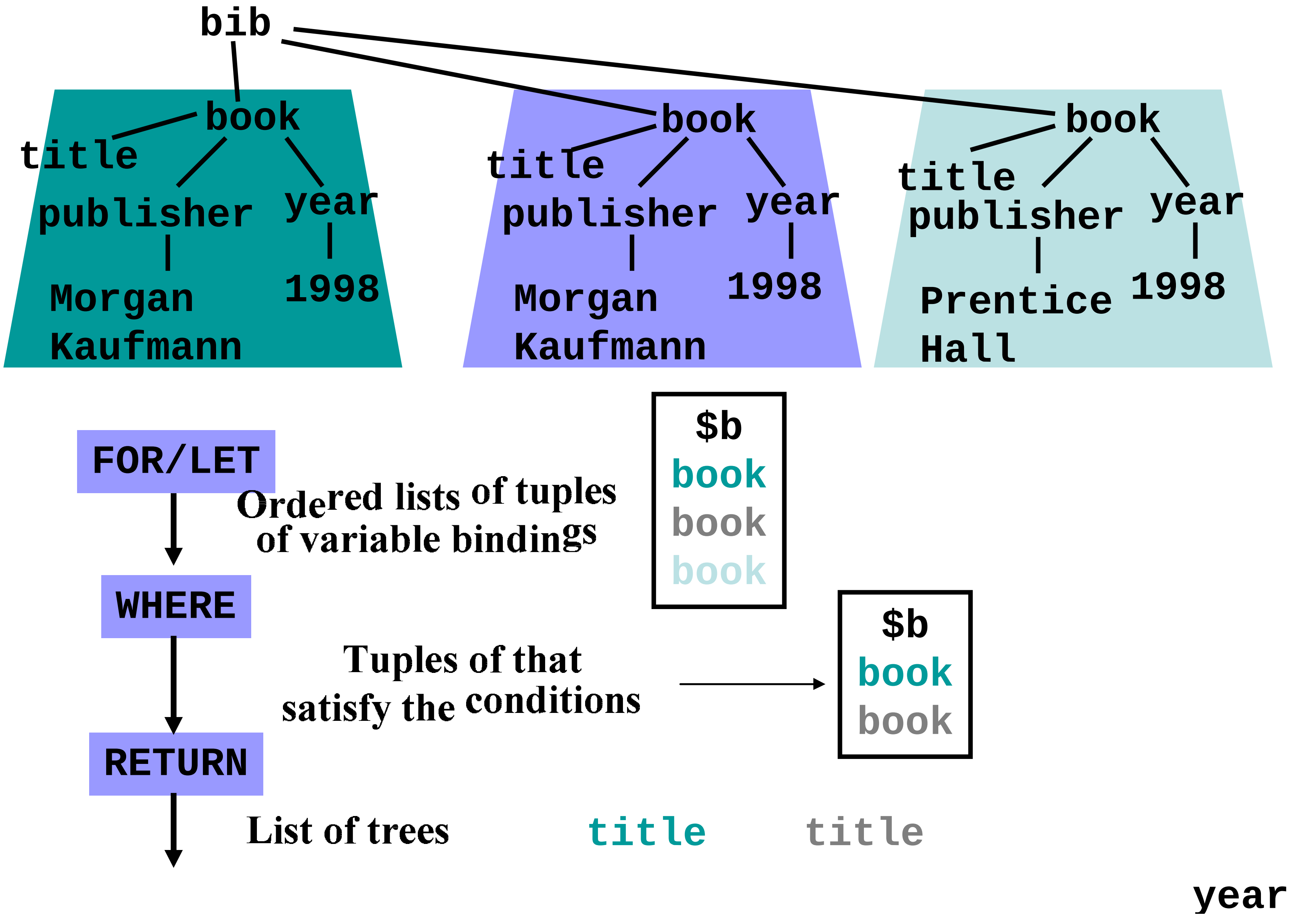
For-Let-Where-Return (FLWR)

List the titles of books published by "Morgan Kaufmann"

```
FOR $b in doc("bib.xml")//book
WHERE $b/publisher = "Morgan Kauffman"
RETURN $b/title
```



Think (tuples of) variable bindings



```
FOR $b in doc("bib.xml")//book
WHERE $b/year > 1990
RETURN $b/author
```

Return the list of authors who published after 1990

Tuples

List publishers who have published more than 1 book

```
FOR $p in distinct(doc(“bib.xml”)//publisher)
LET $b := document(“bib.xml”)//book[publisher = $p]
WHERE count($b) > 1
RETURN $p
```

Tuples ($p, $b) are formulated

Boolean Expressions in WHERE

List the titles of `books` published by `“Morgan Kaufmann”` in `1998`

```
FOR $b in doc(“bib.xml”)//book
WHERE $b/publisher = “Morgan Kaufmann”
 AND $b/year = “1998”
RETURN $b/title
```

An Unified Tuple-Based Algebra for XQuery

Unified Data model (UDM) Extending XQDMA

(XQDMA) lists l = [o1, o2, ..., on]

ois are XQDMA nodes

Tuples t = ($\$v_1$=$a_1$, $\$v_2$=$a_2$, ..., $\$v_n$=$a_n$)

"(in t) the variable $\$v_i$ binds to variable binding a_i"

t.$\$v_i$ may be: an XQDMA list, or

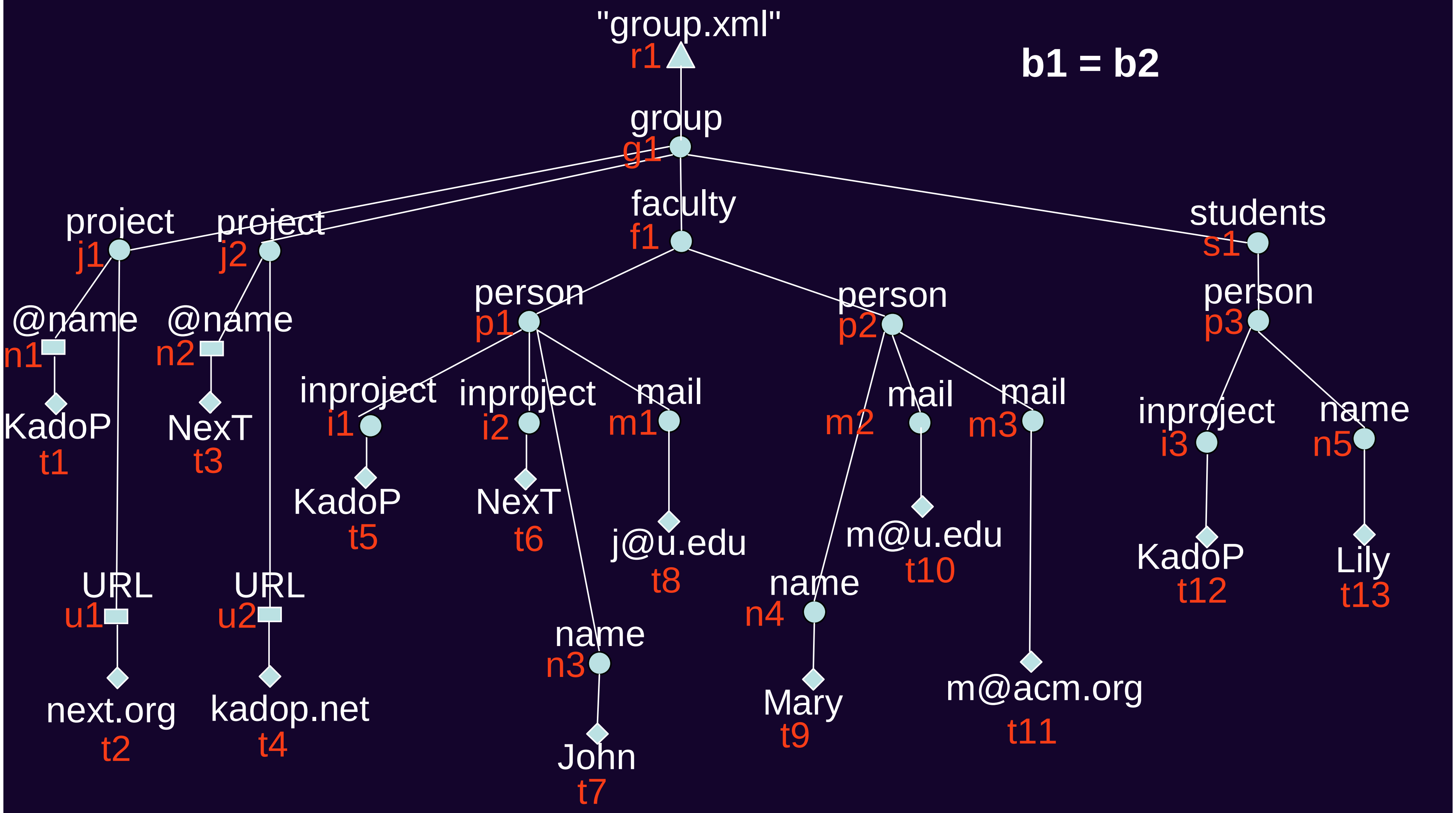
a set/bag/list (collection) of homogenous tuples

($\$v_1$, $\$v_2$, ..., $\$v_n$): tuple schema

UDM Tuple Equality

b1 = ($v1=[p1], $v2=[t4], $v3=[i1, t5])
b2 = ($v1=[p1], $v2=[t11], $v3=[i3, t14])

b1 = b2



UDM Operations on Tuples

$t_1 = (\$v_1 = a_1, ..., \$v_n=a_n), t_2 = (\$u_1=b_1, ..., \$u_n=b_n)$

$t1 + (\$vn+1=an+1) = (\$v_1 = a_1, ..., \$v_n=a_{n,}, \$v_{n+1}=a_{n+1})$

$t_1 = t_2$ iff:

- n = m
- for every i = 1,..., m, one of the following holds:
  - $t1.\$v_i$ and $t_2.\$u_i$ are *values* and $t_1.\$v_i =_v t_2.\$u_i$
  - $t1.\$v_i$ and $t_2.\$u_i$ are *nodes* and $t_1.\$v_i =_{id} t_2.\$u_i$
  - $t1.\$v_i$ and $t_2.\$u_i$ are *lists* | *sets* | *bags* and $t_1.\$v_i = t_2.\$u_i$

Unified Tuple-Based Algebra Operators

Unified tuple-based algebra operators

Navigation

XML construction

Nested plans

Relational-style operators

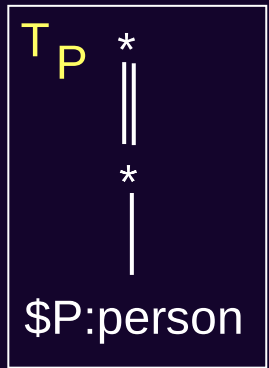
Other operators

XPath navigation

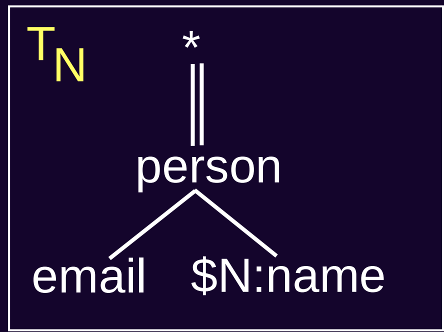
Based on tree patterns

$R//faculty/person

$R/*/person

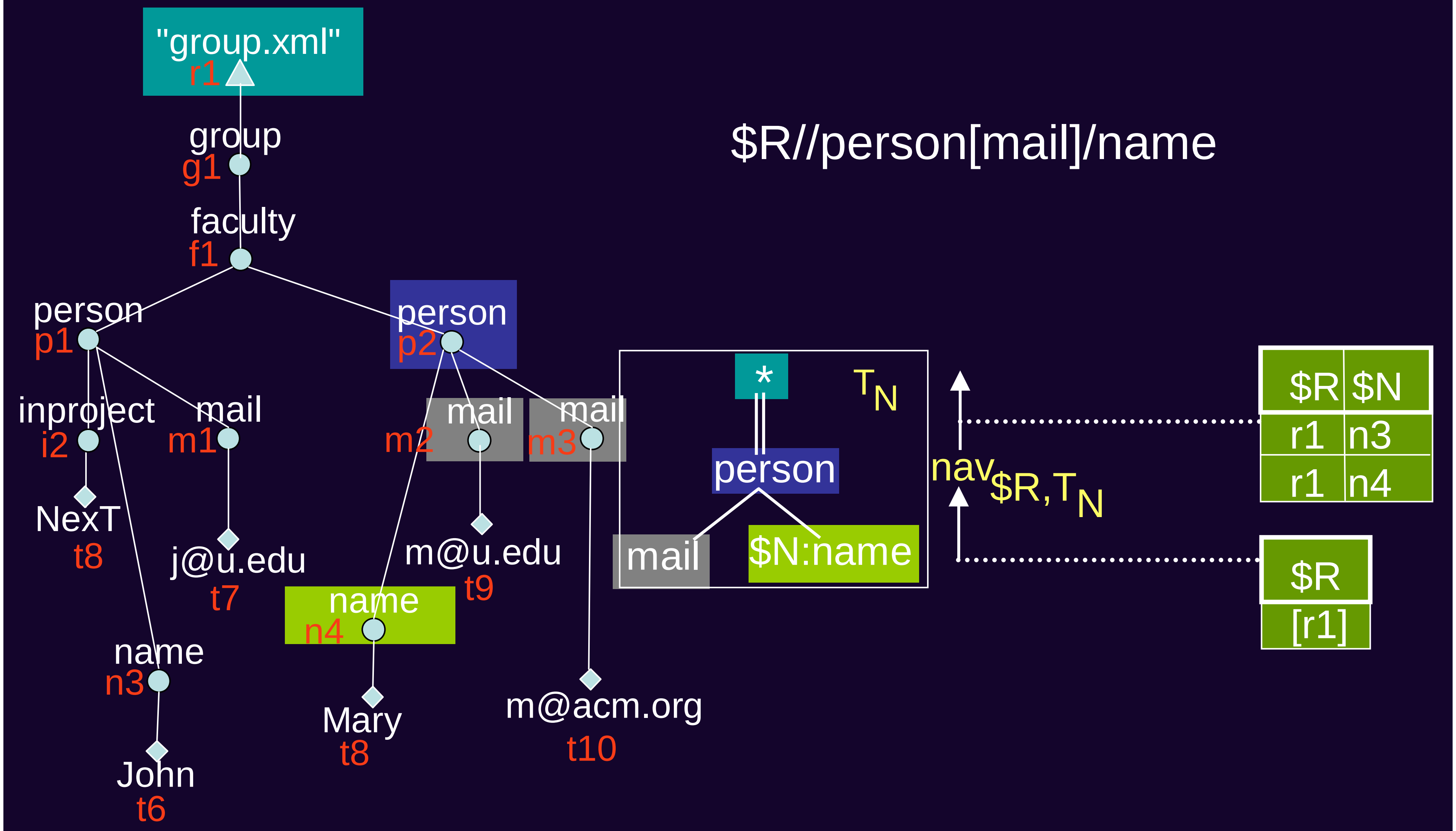


$R//person[email]/name



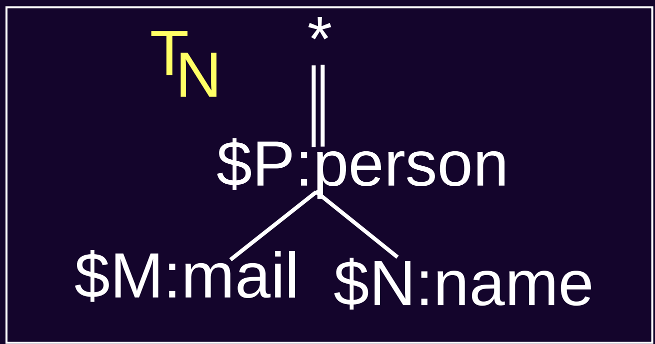
Navigation operator $nav_{\$R,T_F}$(Tuple Collection): Tuple Collection

XPath navigation



Tree patterns capture navigation of "for"

for $P in $R//person, $M in $P/mail, $N in $P/name
return { $M, $N }



$nav_{\$R, T_N}$

$R	$P	$M	$N
r1	p1	m1	n3
r1	p2	m2	n4
r1	p2	m3	n4

$R
r1

Generalized "for" navigation

for $P in $R//person, $M in $P/mail, $N in $P/name
return { $M, $N }

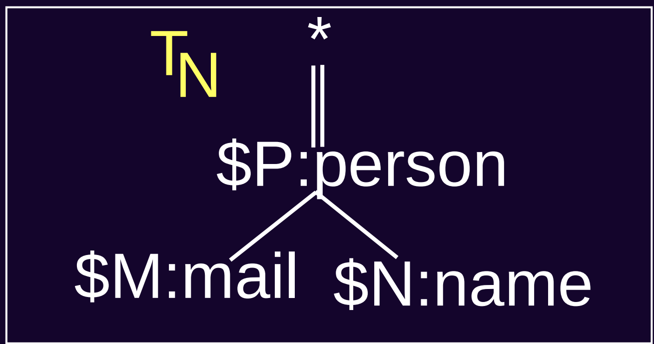
$P	$M	$N
p1	m1	n3
p2	m2	n4
p2	m3	n4

$\pi_{\$P,\$M,\$N}$

$R	$P	$M	$N
r1	p1	m1	n3
r1	p2	m2	n4
r1	p2	m3	n4

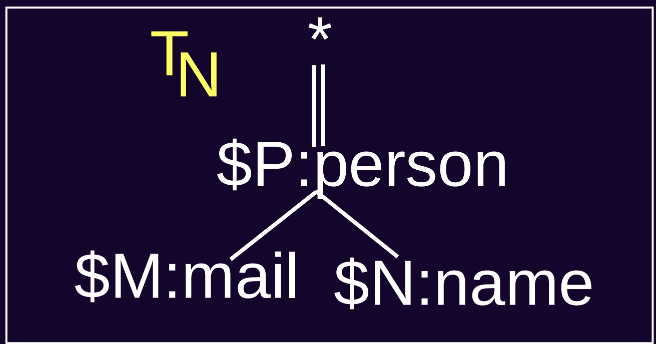
$nav_{\$R,T_N}$

$R
r1



Generalized "for" navigation

for $P in $R//person, $M in $P/mail, $N in $P/name
return { $M, $N }

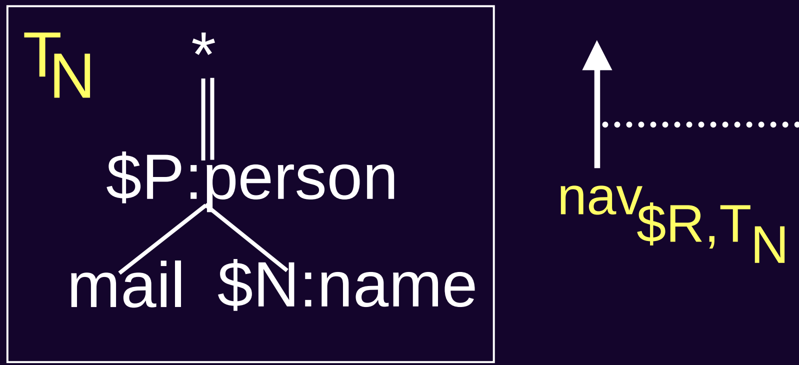


$P	$M	$N
p1	m1	n3
p2	m2	n4
p2	m3	n4

$R
r1

Generalized "for" and "where" navigation

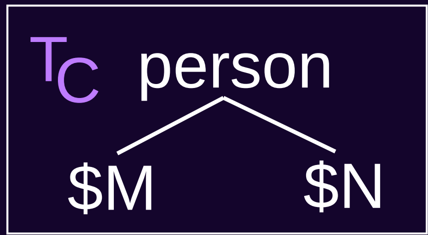
for $P in $R//person, $N in $P/name
where $P/email
return { $M, $N }

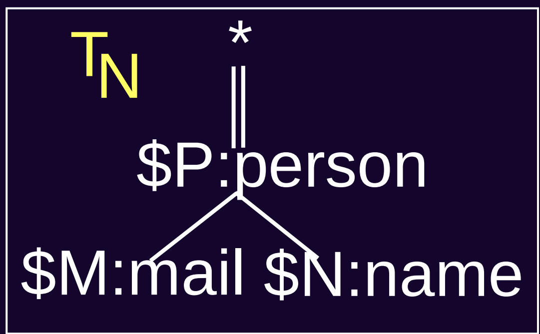


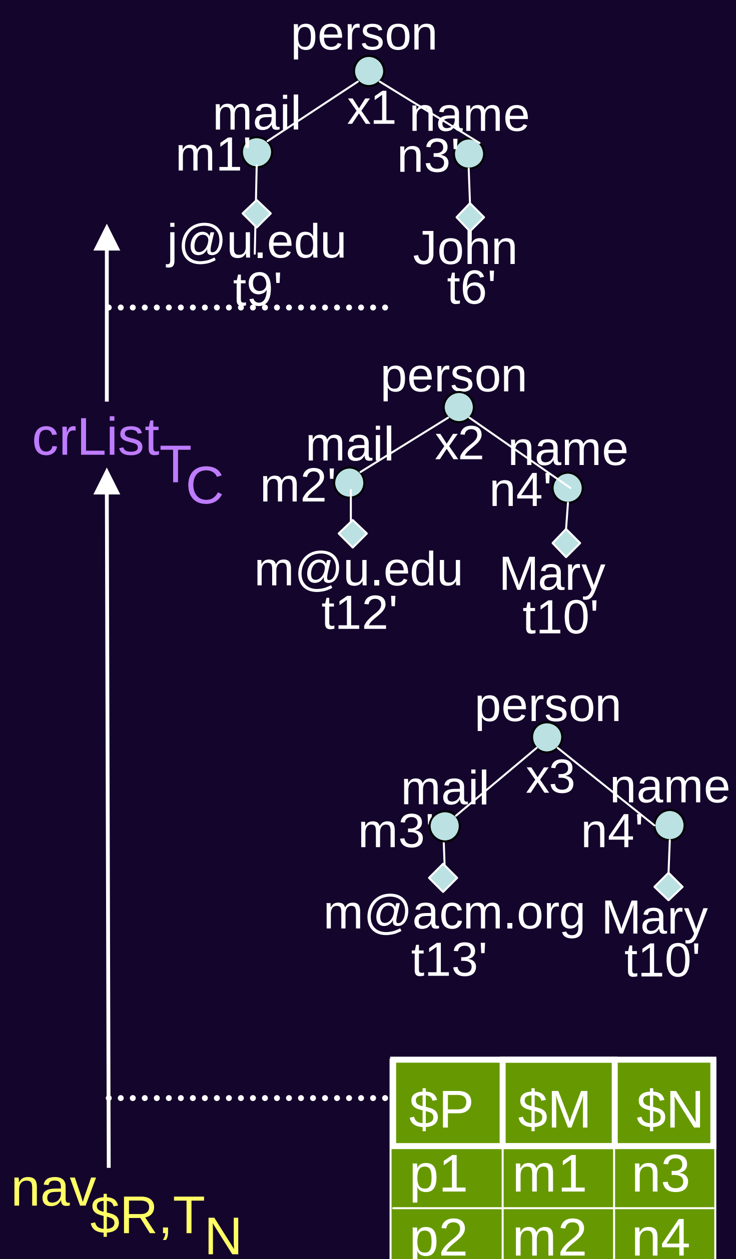
$P	$N
p1	n3
p2	n4
p2	n4

XML result construction

```
for $P in $R//person,
    $N in $P/name,
    $M in $P/mail
return <person>
         { $M, $N }
       </person>
```





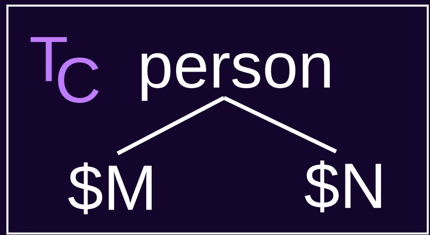


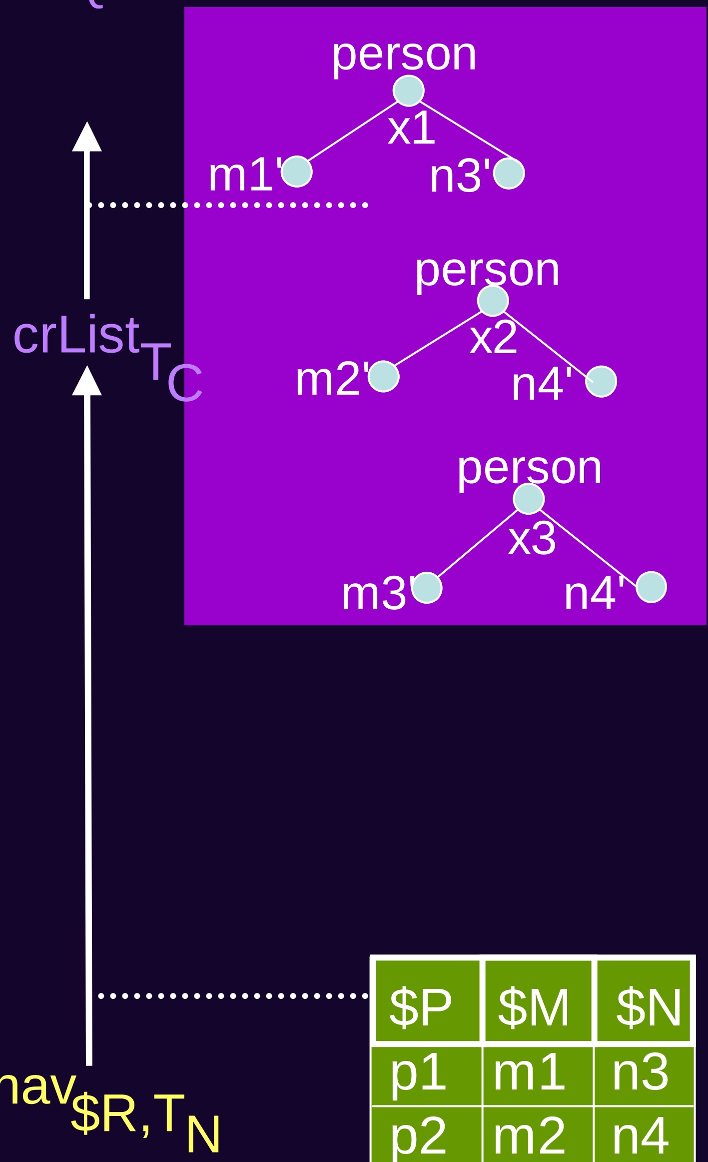
$P	$M	$N
p1	m1	n3
p2	m2	n4
p2	m3	n4

XML result construction

Back to XQDMA

```
for $P in $R//person,
    $N in $P/name,
    $M in $P/mail
return <person>
          { $M, $N }
       </person>
```

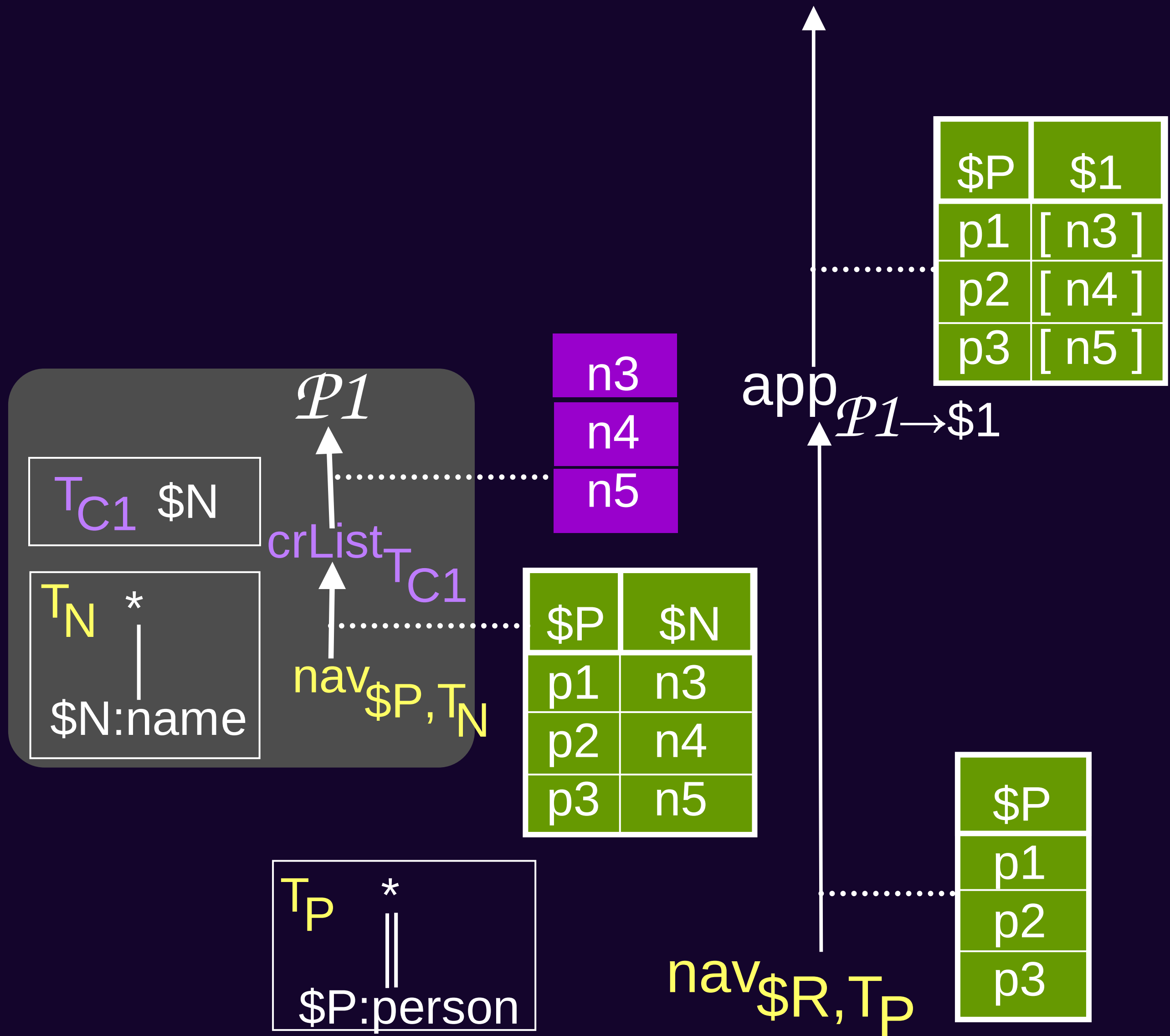




$P	$M	$N
p1	m1	n3
p2	m2	n4
p2	m3	n4

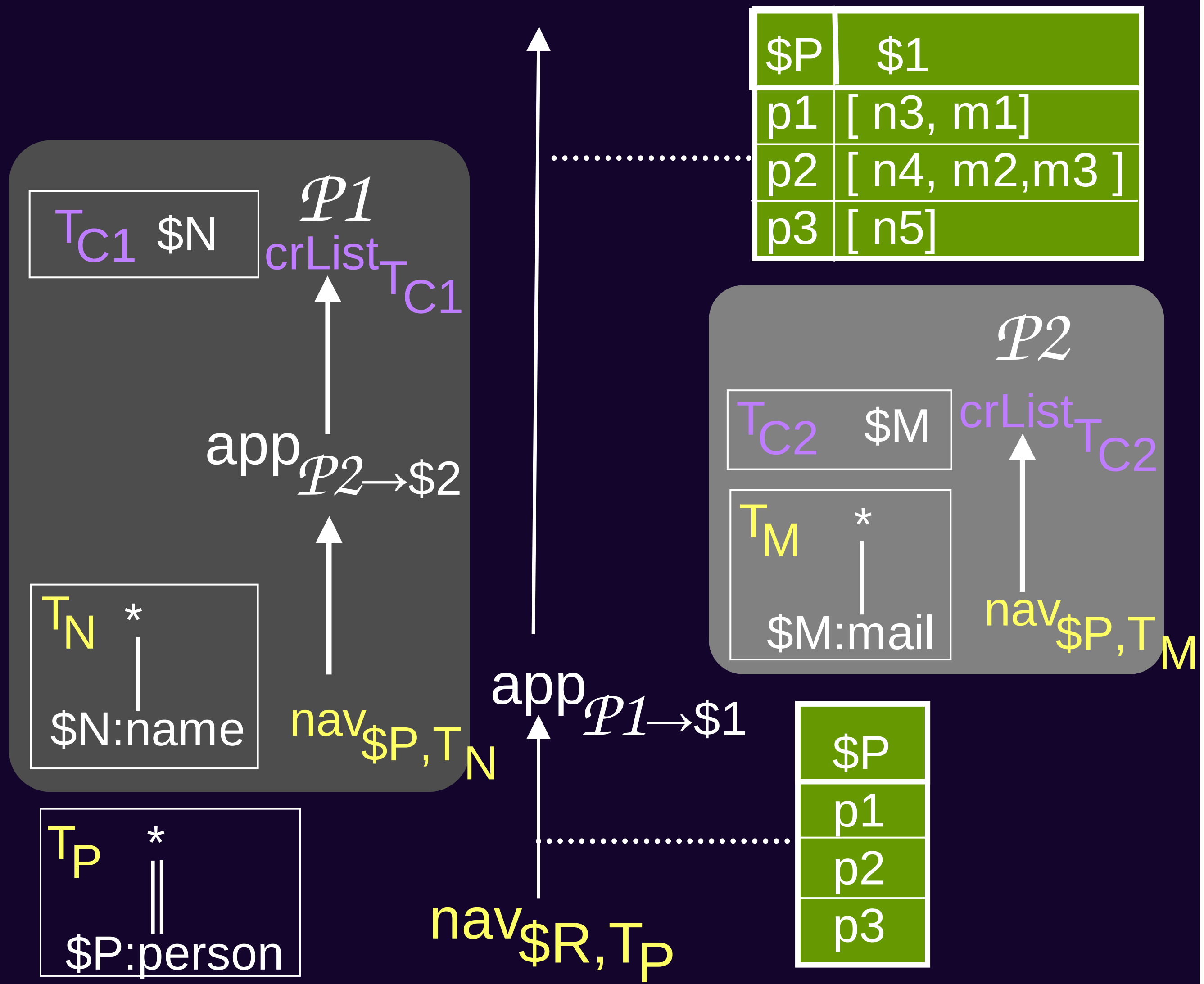
Nested plans and the *apply* operator (1)

```
for $P in $R//person
return
<person>
  { for $N in $P/name
    return
      { $N                    }
  }
</person>
```



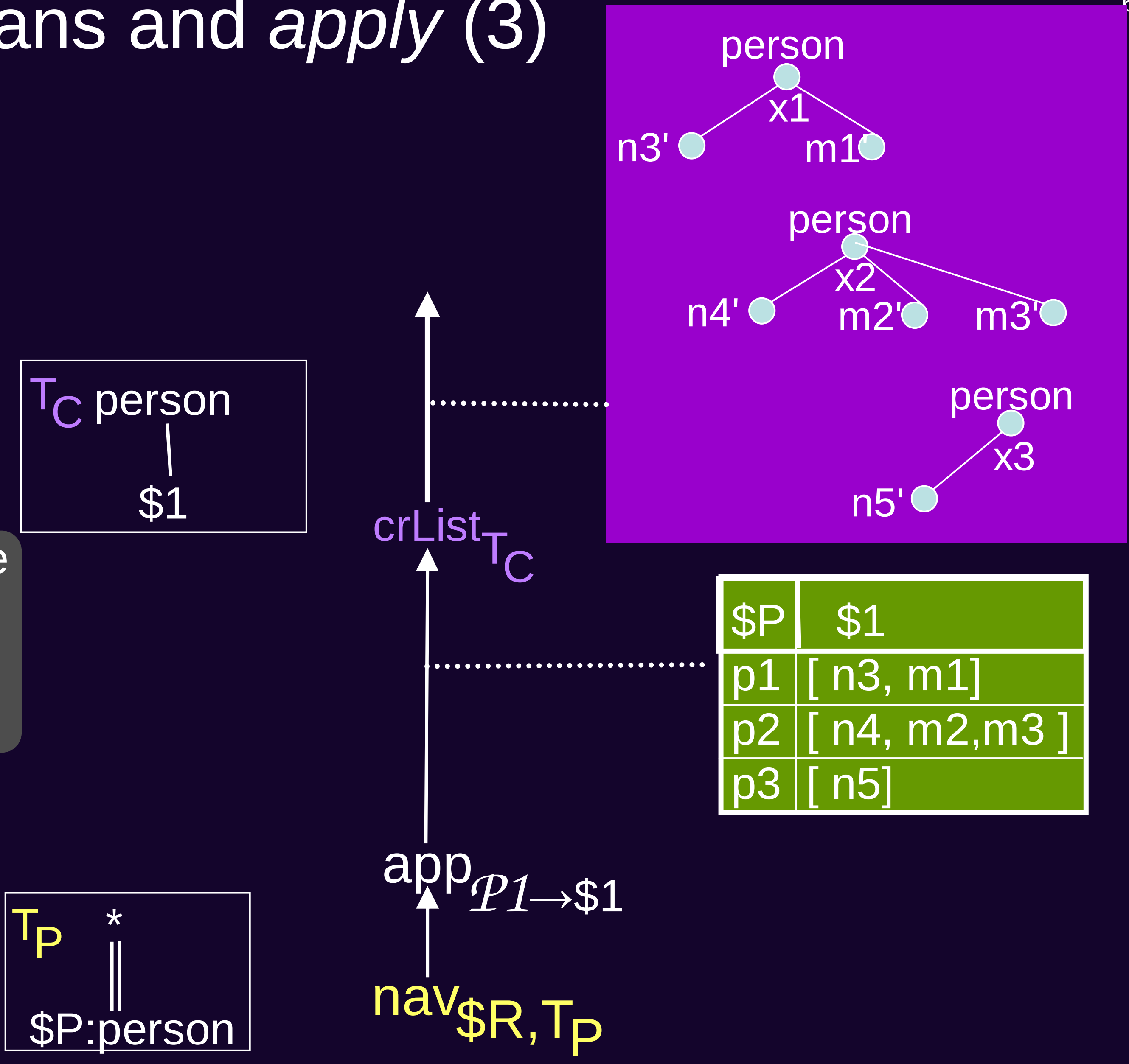
Nested plans and *apply* (2)

```
for $P in $R//person
return
<person>
    { for $N in $P/name
      return
        { $N, $P/mail }
    }
</person>
```



Nested plans and *apply* (3)

```
for $P in $R//person
return
<person>
  { for $N in $P/name
    return
      { $N, $P/mail }
  }
</person>
```



$P	$1
p1	[n3, m1]
p2	[n4, m2,m3]
p3	[n5]

Nested plans and *let* clauses

Previous query:

```
for $P in $R//person
return
<person>
    { for $N in $P/name
      return
         { $N, $P/mail }
     }
</person>
```

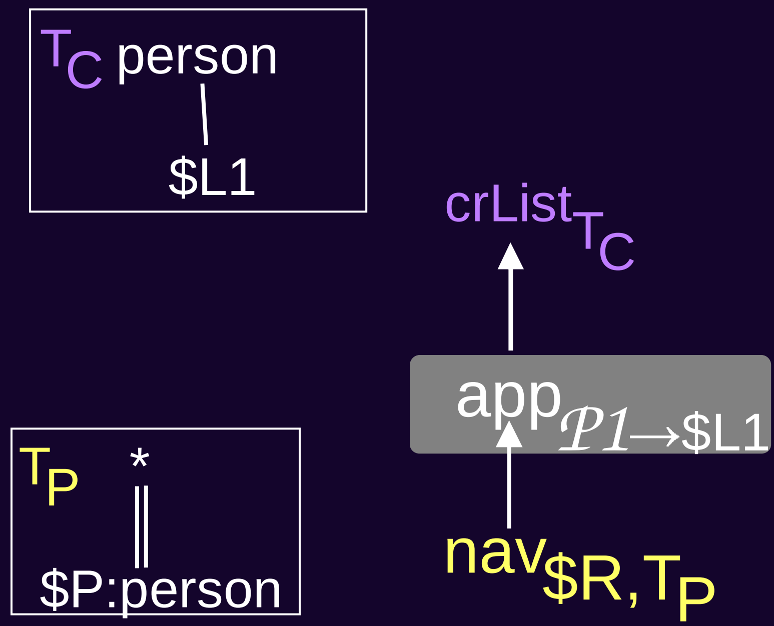
Same query with *let* clauses:

```
for $P in $R//person
let $L1 =
    { for $N in $P/name
      let $L2 = $P/mail
      return { $N, $L2 }
    }
return
<person>
     { $L1 }
</person>
```

Nested plans and *let* clauses

Same query with *let*:

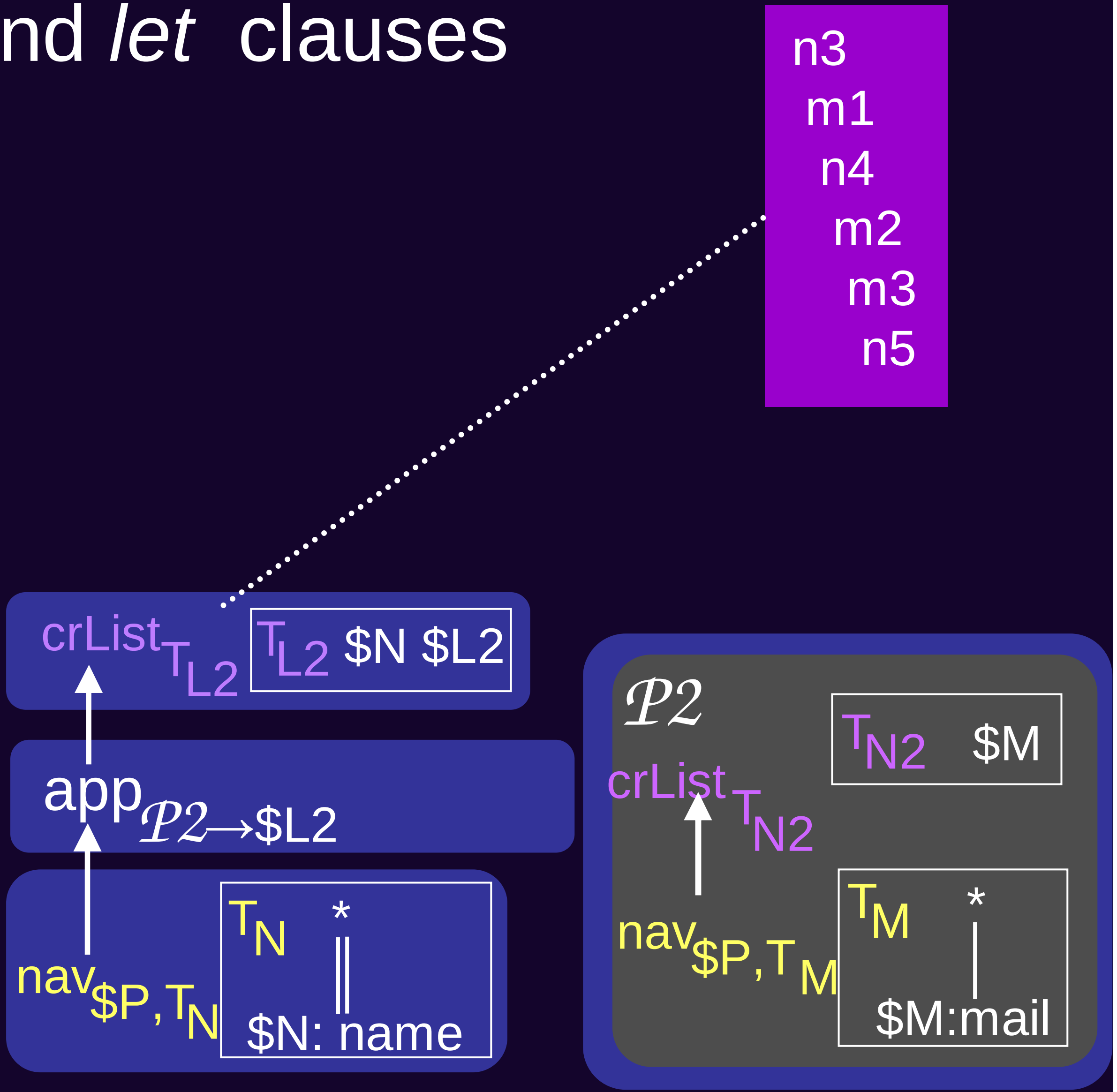
```
for $P in $R//person
let $L1 =
P1 { for $N in $P/name
      let $L2 =
          { $N, $P/mail }
       return { $L2 }
   }
return
<person>
    { $L1 }
</person>
```



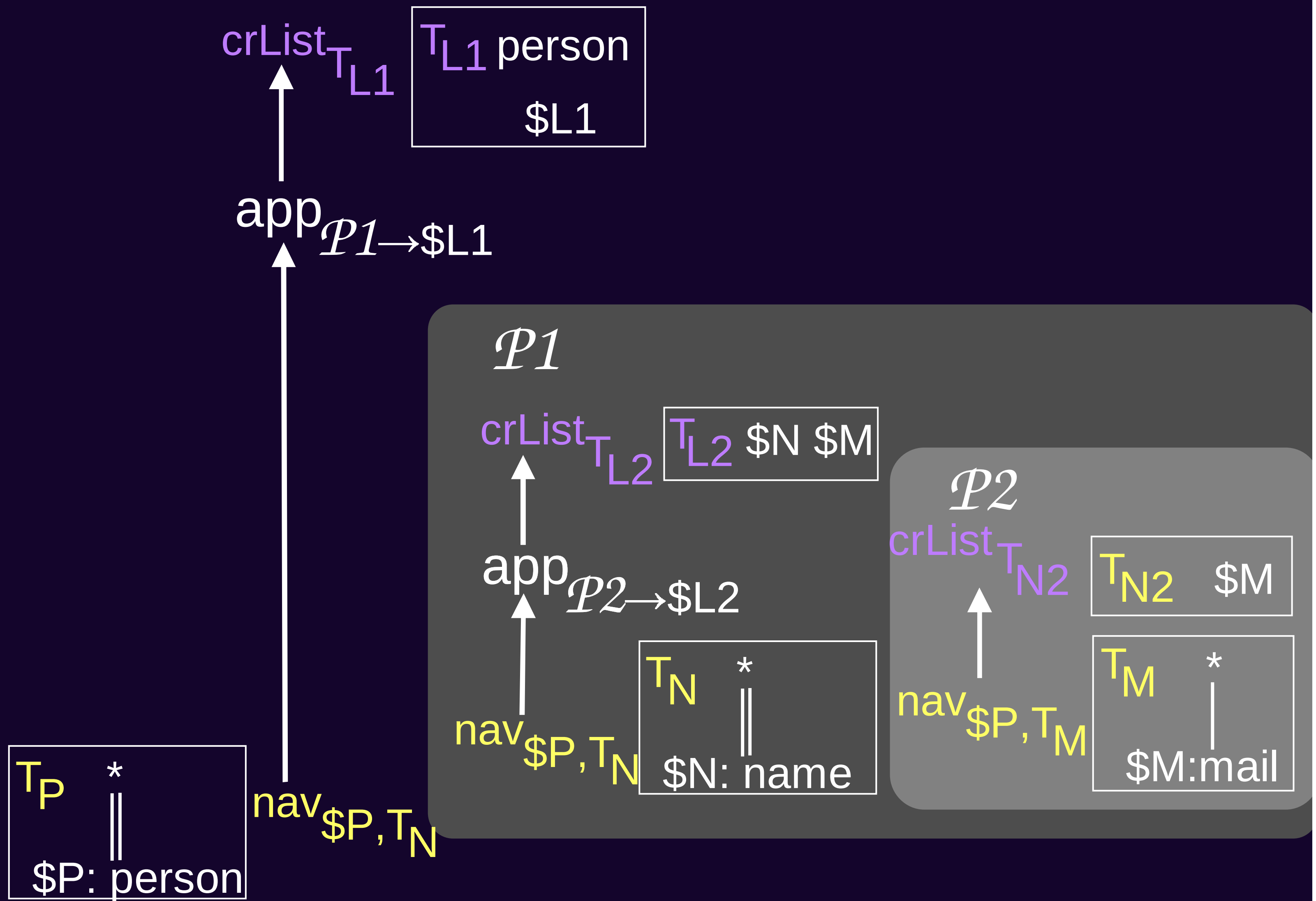
Nested plans and *let* clauses

Same query with *let*:

```
for $P in $R//person
let $L1 =
   { for $N in $P/name
     let $L2 = $P/mail
      return { $N, $L2 }
   }
return
<person>
   { $L1 }
</person>
```



Nested plans and *let* clauses



Nested plans and *let* clauses

Nested queries can be equivalently rewritten using *let* clauses until return clauses reach the form:

{ $V1, $V2, ..., $VK}   or   <tag> { $V1, $V2, ..., $Vk } </tag>

Nested queries with let can be "automatically" translated

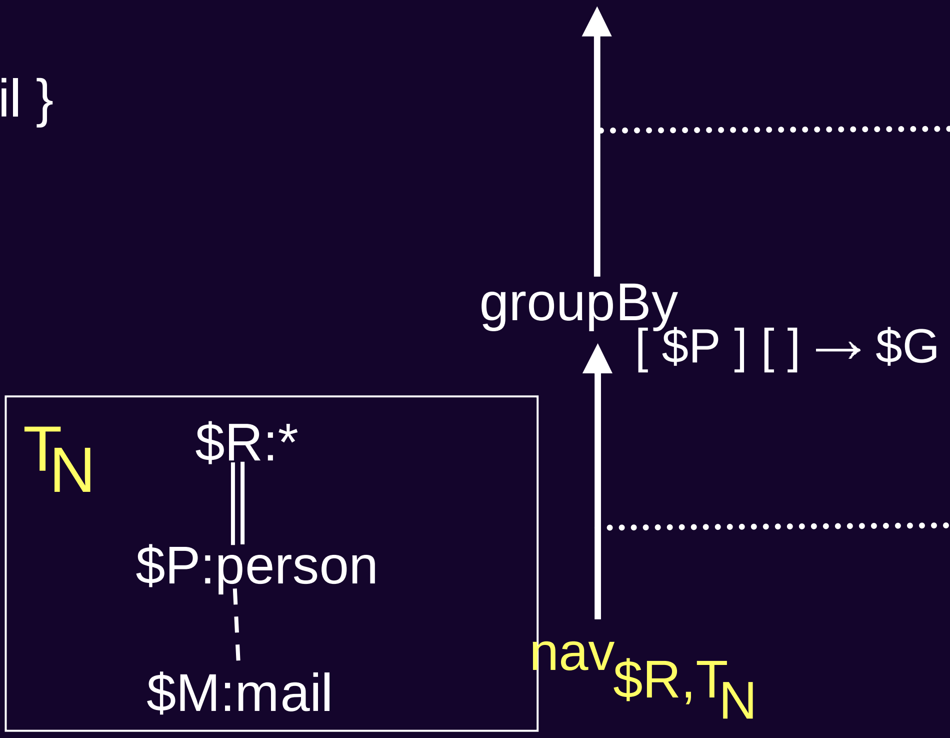
- for → nav
- let → apply
- return → crList

Nested plans and optional navigation

Capture all navigation with a single pattern

- optional edges
- null (⊥) variable values
- groupBy [by id] [by value]

```
for $P in $R//person
return
<person>
   { $P/mail }
</person>
```



$P	$G [$P | $M]
p1	[p1 | m1]
p2	[p2 | m2, p2 | m3]
p3	[p3 | ⊥]

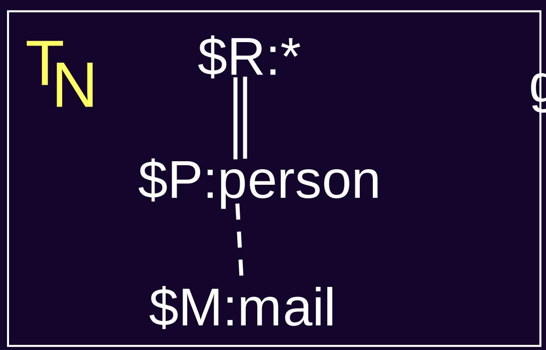
$P	$M
p1	m1
p2	m2
p2	m3
p3	⊥

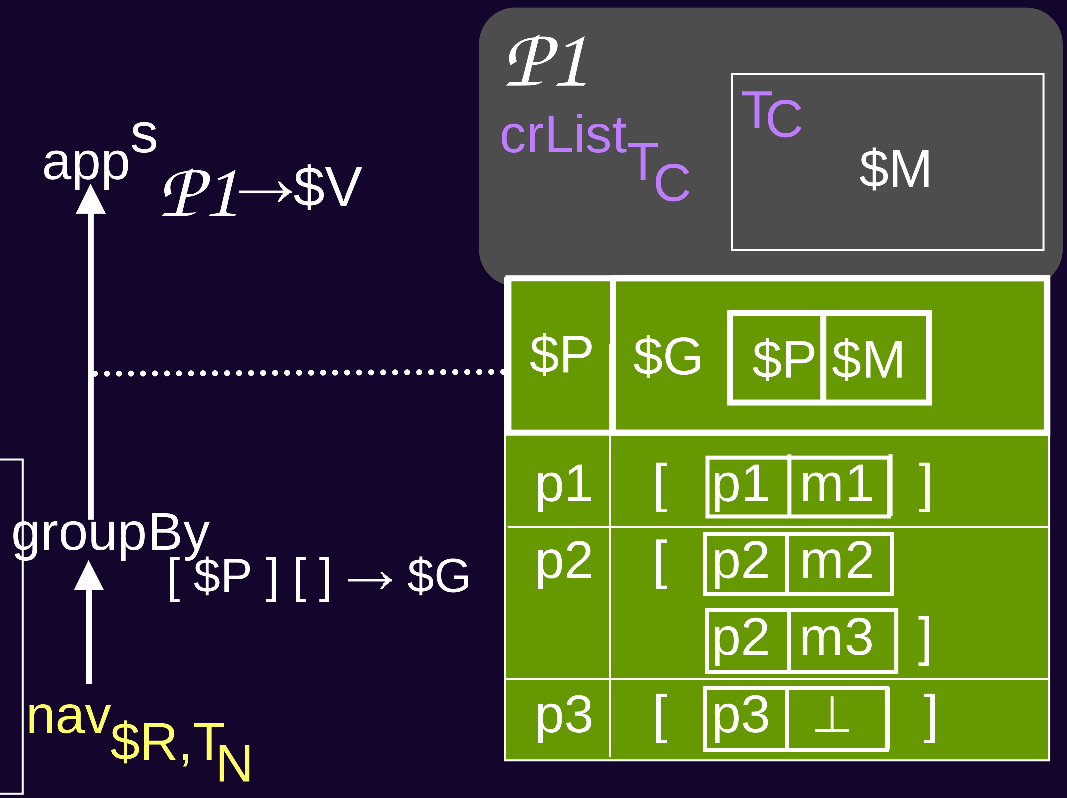
Nested plans and optional navigation

Capture all navigation with a single pattern

- optional edges
- null (⊥) variable values
- groupBy [variables] [values]
- apply on set: $app^{s}_{\mathcal{P}} \rightarrow \$V$

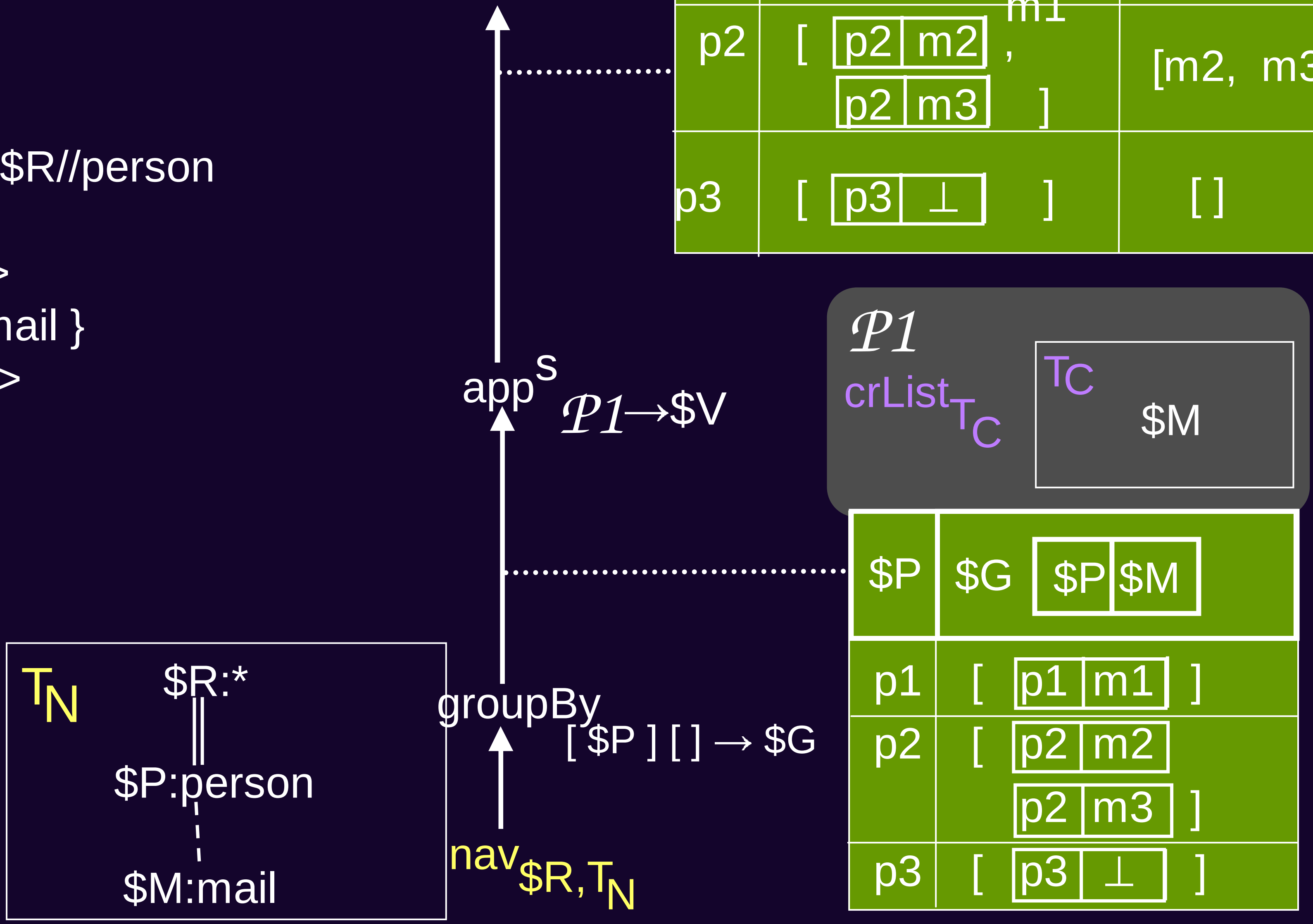
for $P in $R//person
return
<person>
{ $P/mail }
</person>





Nested plans and optional navigation

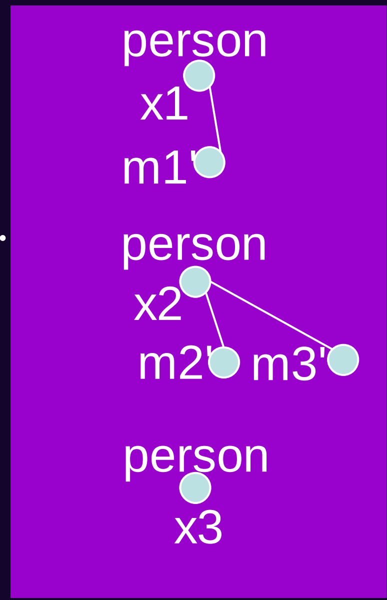
```
for $P in $R//person
return
<person>
    { $P/mail }
</person>
```



$P	$G ($P | $M)	$V
p1	[p1 | m1]	[m1]
p2	[p2 | m2 , p2 | m3]	[m2, m3]
p3	[p3 | ⊥]	[ ]

$P	$G ($P | $M)
p1	[p1 | m1]
p2	[p2 | m2 , p2 | m3]
p3	[p3 | ⊥]

Nested plans and optional navigation



T_V | $V

$crList_{T_V}$

for $P in $R//person
return
<person>
{ $P/mail }
</person>

app^s $\mathcal{P}1 \rightarrow$ $V

groupBy [$P] [ ] → $G

$nav_{\$R, T_N}$

T_N: $R:* ‖ $P:person ¦ $M:mail

$P	$G [$P | $M]	$V
p1	[p1 | m1]	[m1]
p2	[p2 | m2 , p2 | m3] m1	[m2, m3]
p3	[p3 | ⊥]	[ ]

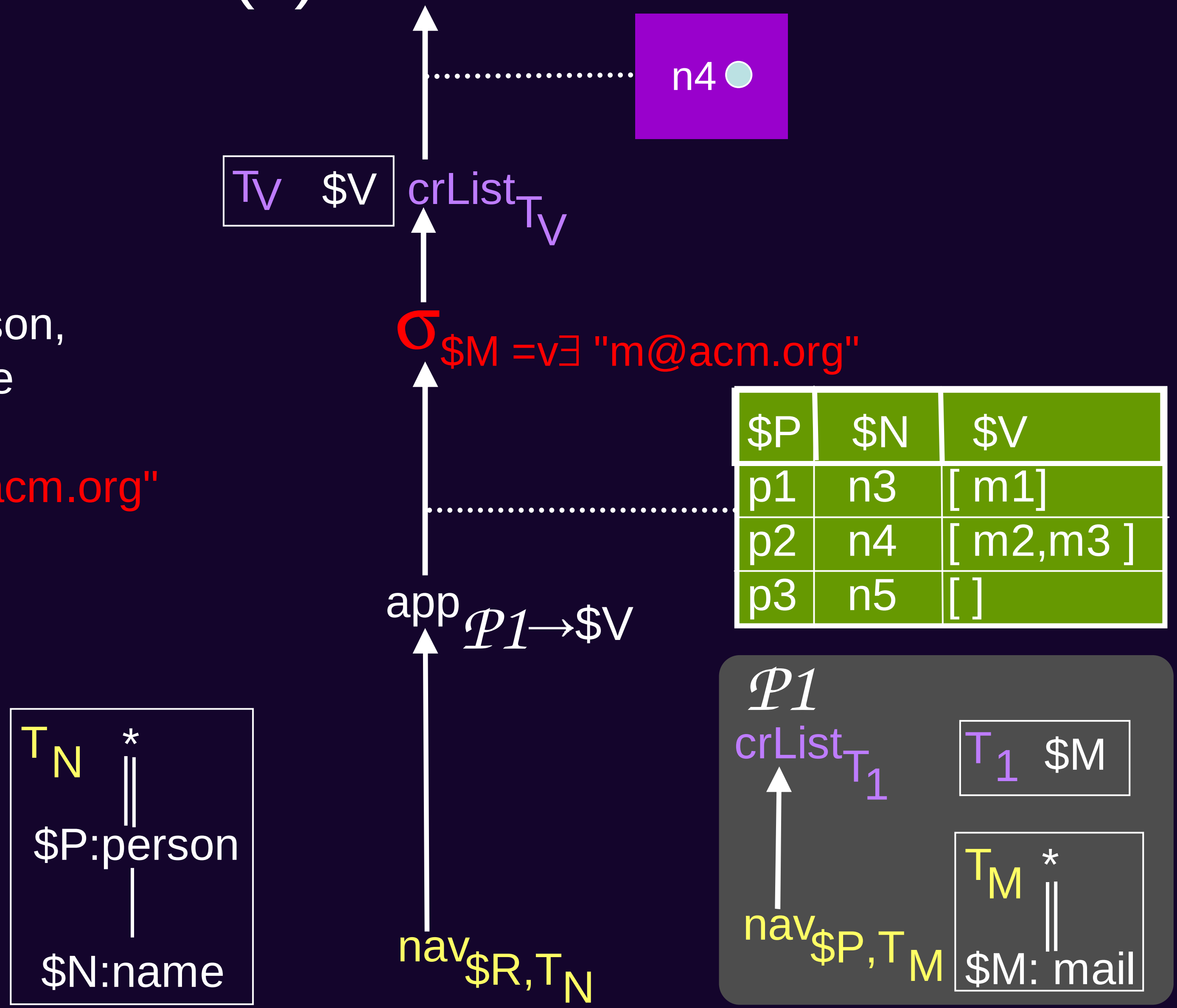
Selection Predicates

Predicate	Meaning	XQuery notation / fn or op [XQFO]
=id	same node	is / op:is-same-node
=v	same value	eq / fn:compare, op:numeric-equal...
<v	smaller value	lt / fn:compare, op:num-less-than...
<<	node before	<< / op:node-before
=	list equality	eq / fn:deep-equal tuple equality
=∃	existential list equality	= / fns. backing eq, tuple equality

Selection Plan (1)

$\sigma_{\theta}(p)$

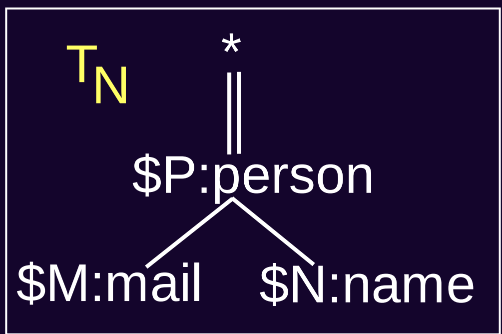
for $P in $R//person,
    $N in $P/name
where
$P/email = "m@acm.org"
return { $N }



Selection Plan (2)

for $P in $R//person,
    $N in $P/name
where
$P/mail = "m@acm.org"
return { $N }

$\mathcal{P}2$
crList_{T_1} [T_1 $M]



app^s $\mathcal{P}2 \rightarrow$ $M2

groupBy [$P,$N] [ ] → $G

$\text{nav}_{\$R,T_N}$

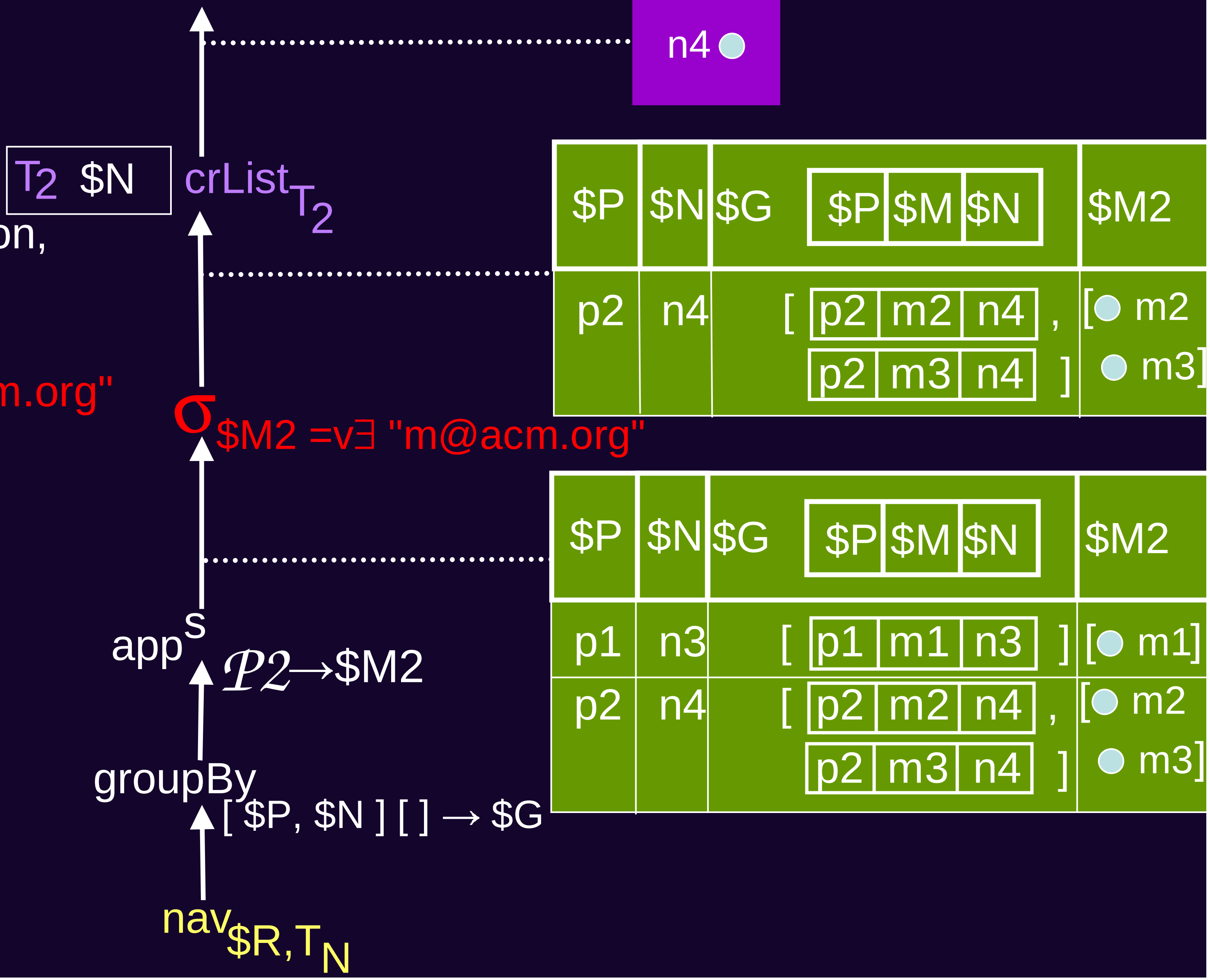
$P	$N	$G [$P | $M | $N]	$M2
p1	n3	[p1 | m1 | n3]	[● m1]
p2	n4	[p2 | m2 | n4 , p2 | m3 | n4]	[● m2 ● m3]

$P	$N	$G [$P | $M | $N]
p1	n3	[p1 | m1 | n3]
p2	n4	[p2 | m2 | n4 , p2 | m3 | n4]

$P	$M	$N
p1	m1	n3
p2	m2	n4
p2	m3	n4

Selection Plan (2)

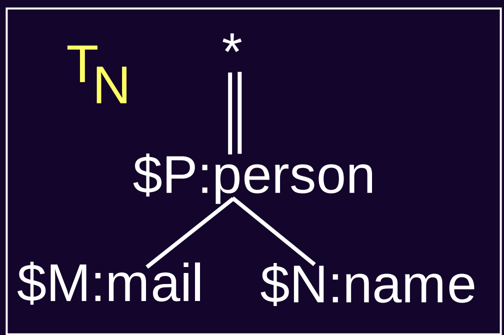
for $P in $R//person,
    $N in $P/name
where
$P/mail = "m@acm.org"
return { $N }



Selection plan (3)

$\sigma_{\theta}(p)$

for $P in $R//person,
$N in $P/name
where
$P/mail = "m@acm.org"
return { $N }



n4

T_C	$N

$crList_{T_C}$

$P	$N	$G ($P | $M | $N)
p2	n4	[(p2 | m2 | n4), (p2 | m3 | n4)]

$groupBy_{[\$P,\$N]\,[\,]\rightarrow \$G}$

$P	$M	$N
p2	m3	n4

$\sigma_{\$M =v \text{"m@acm.org"}}$

$P	$M	$N
p1	m1	n3
p2	m2	n4
p2	m3	n4

$nav_{\$R,T_N}$

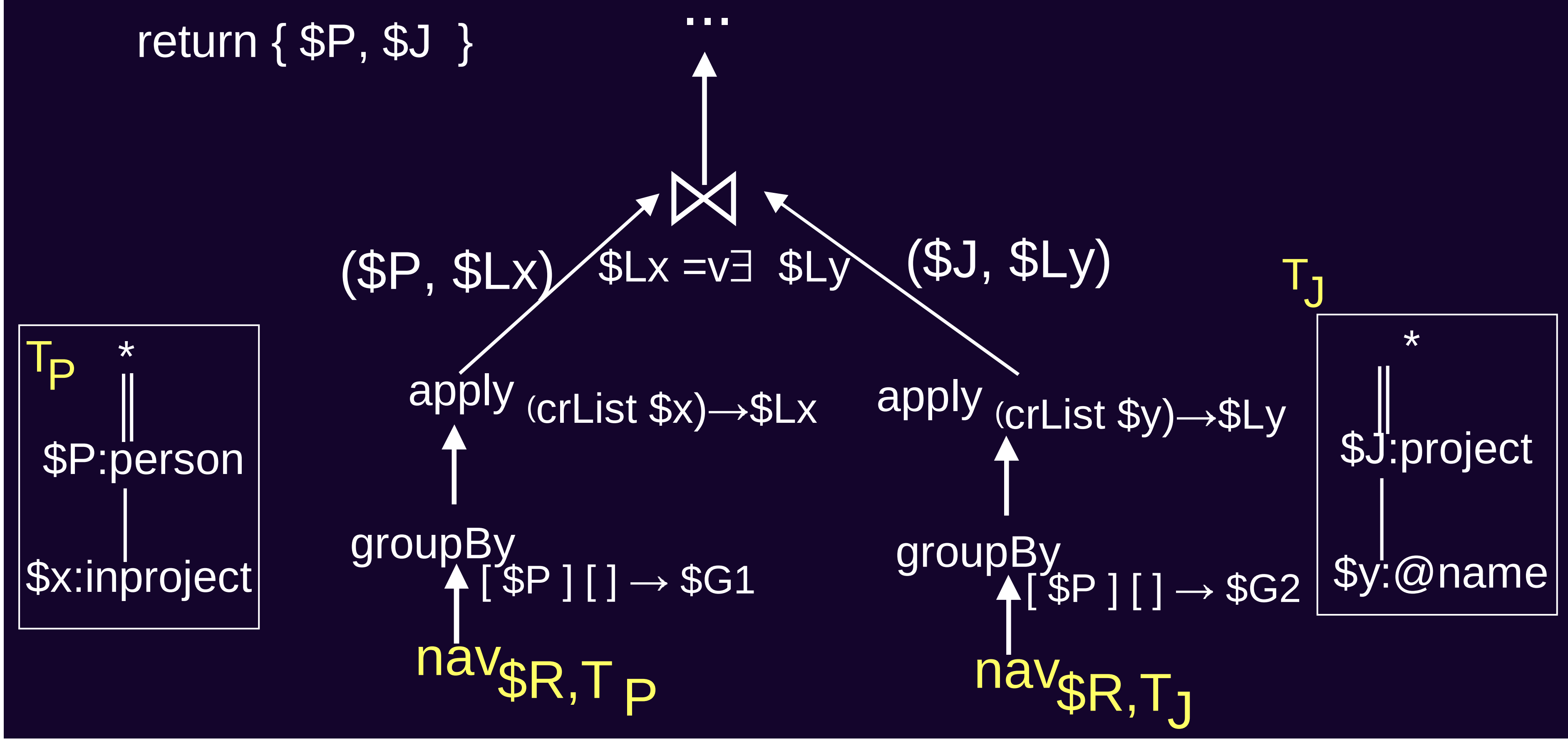
Joins

Value-based join:

for $P in $R//person, $J in $R//projects

where $P/inproject = $J/@name

return { $P, $J }



Joins and nesting

for $J in $R//project

return

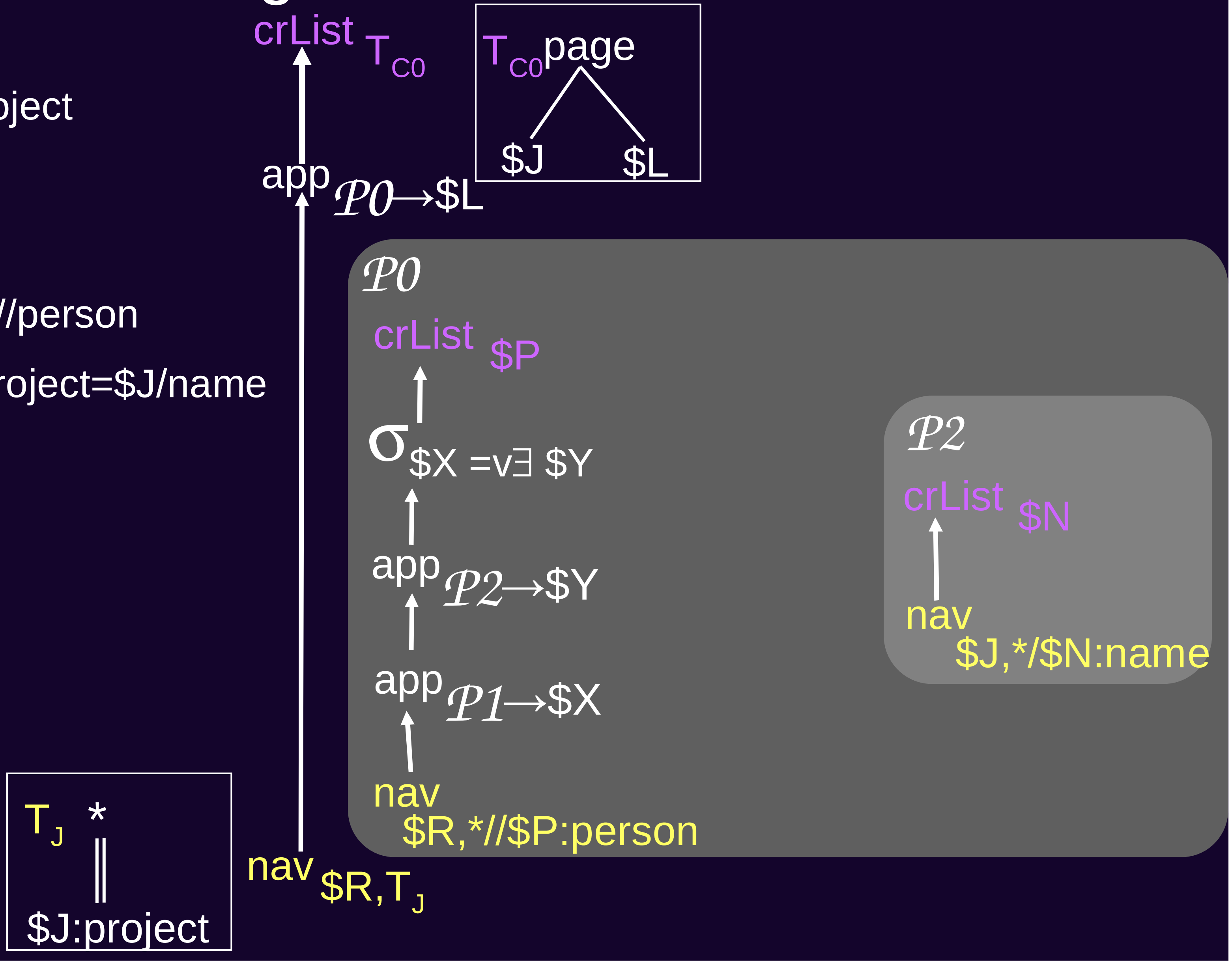
<page> { $J },

{ for $P in $R//person

where $P/inproject=$J/name

return { $P } }

</page>



Joins and nesting

Group people by the project

1) (N) from the notebook cust-->person, order-->project

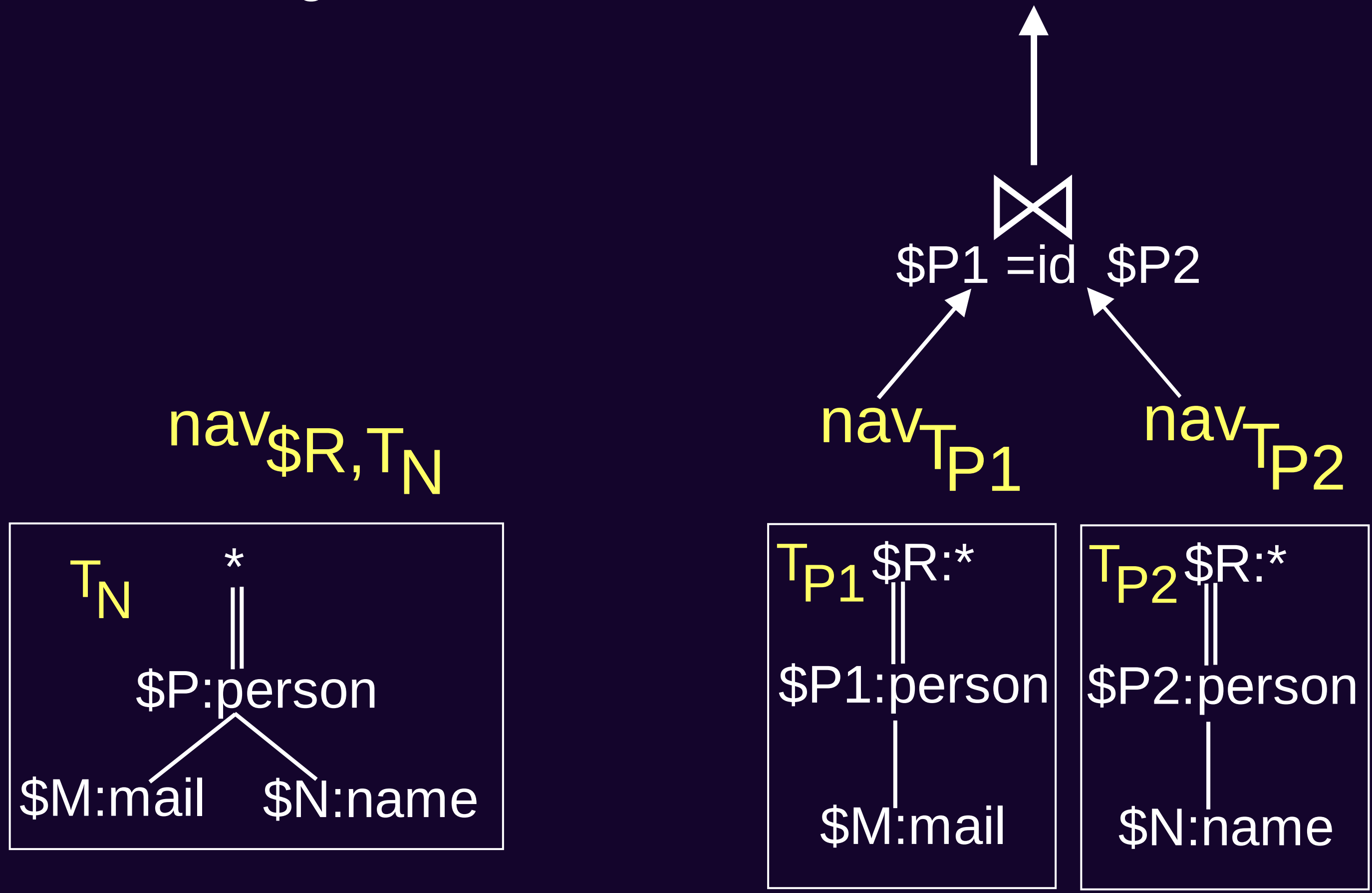
doing many apply plans

2) Same with outerjoins and group the project (which was the outer)

3) now nest outer join, show in the example and then definition

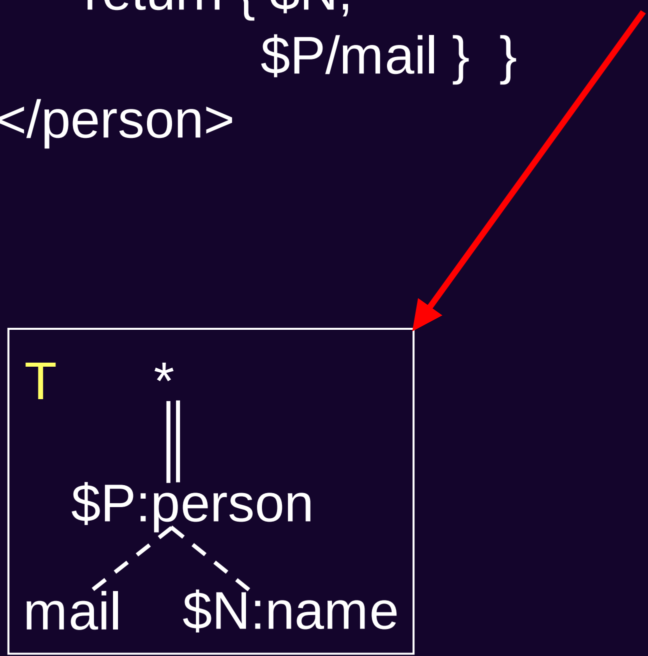
Joins

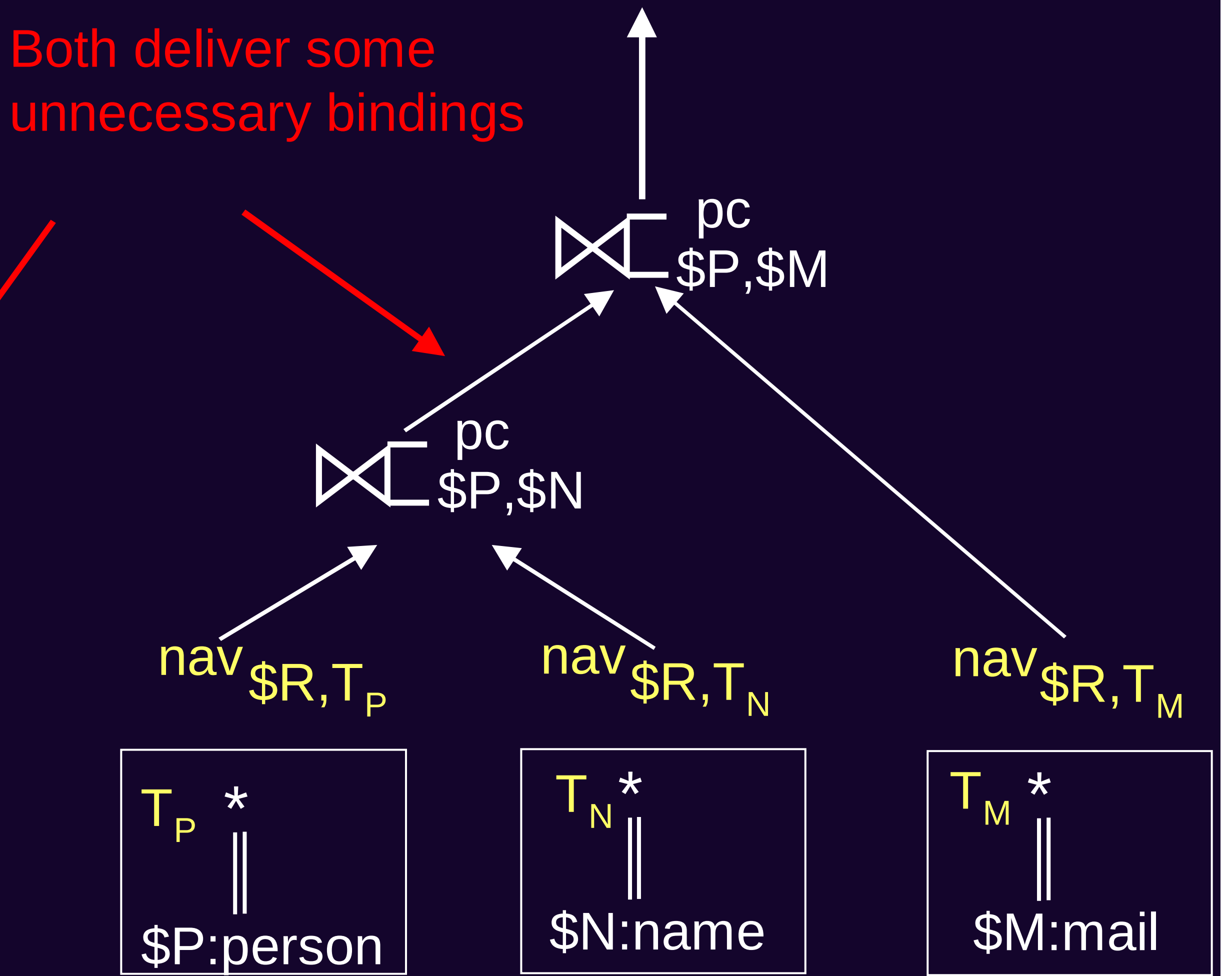
ID-based join may be used to decompose tree pattern navigation into linear navigations:



Structural Joins and Optional Navigation

for $p in //person
return <person>
  { for $n in $p//name
    return { $N,
            $P/mail } }
</person>

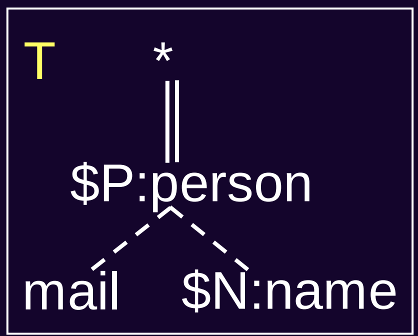




Structural Joins and Optional Navigation:

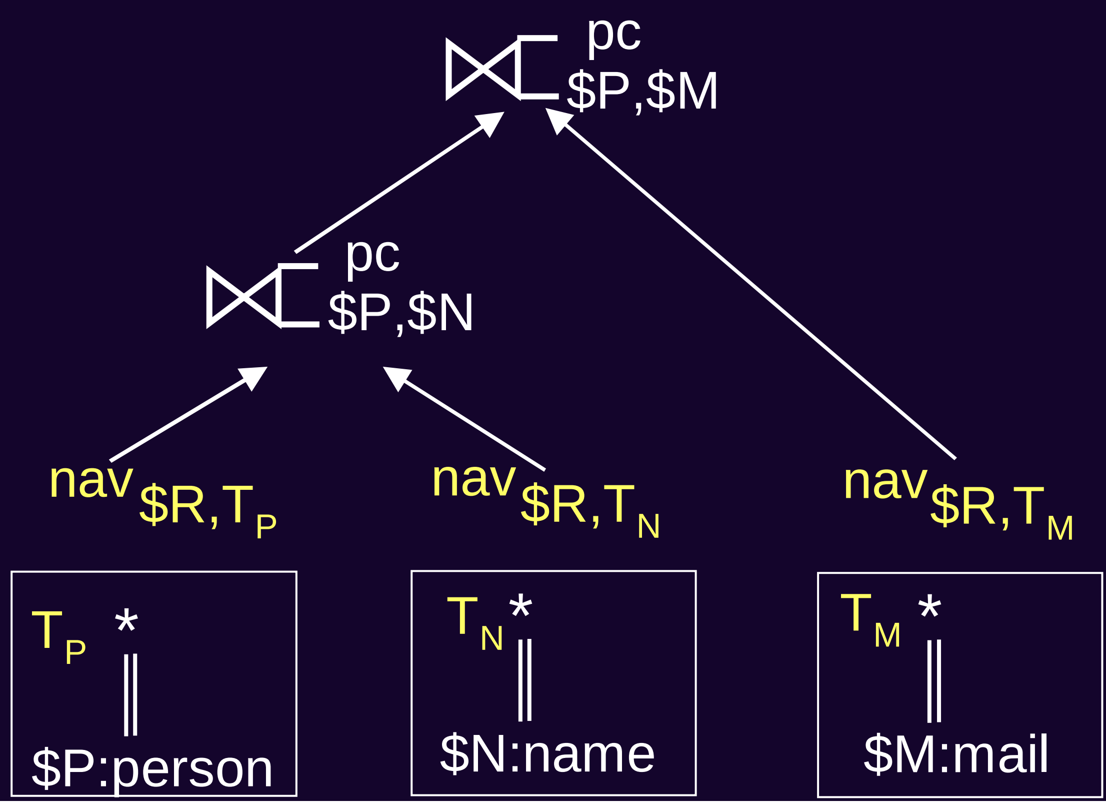
Capturing navigation of:

```
for $p in //person
return <person>
  { for $n in $p//name
    return { $N,
             $P/mail }  }
</person>
```



Structural joins capture *data-driven nesting*

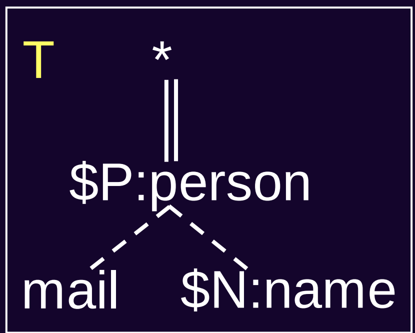
Algebra needs *query-driven nesting*

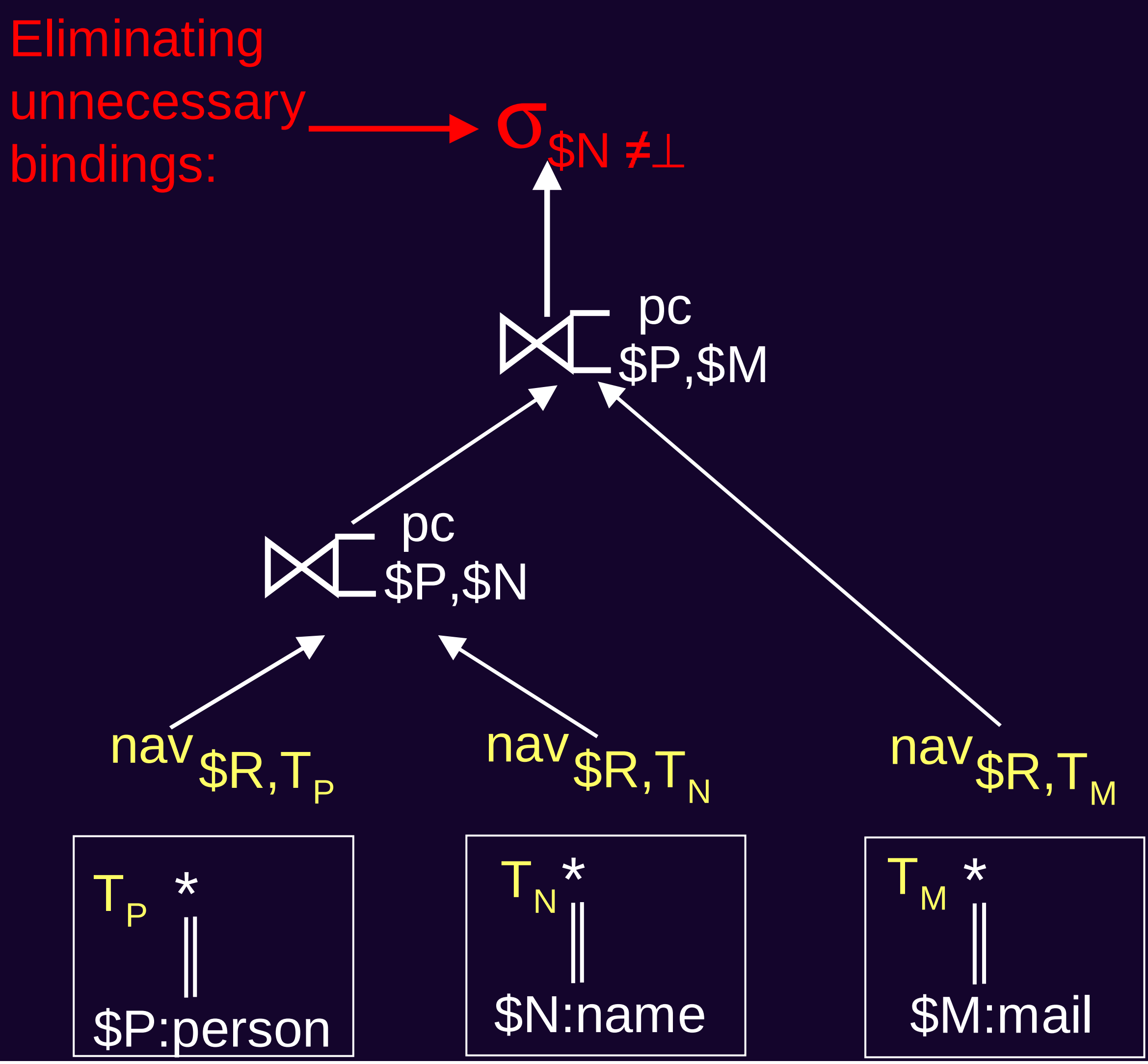


Structural Joins and Optional Navigation

Capturing navigation of:

```
for $p in //person
return <person>
   { for $n in $p//name
     return { $N,
                 $P/mail }  }
</person>
```

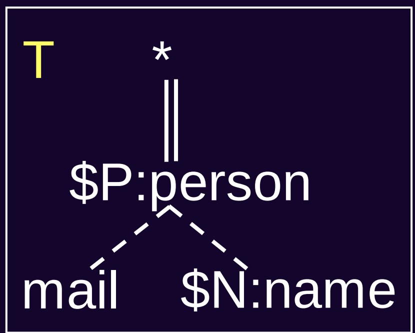


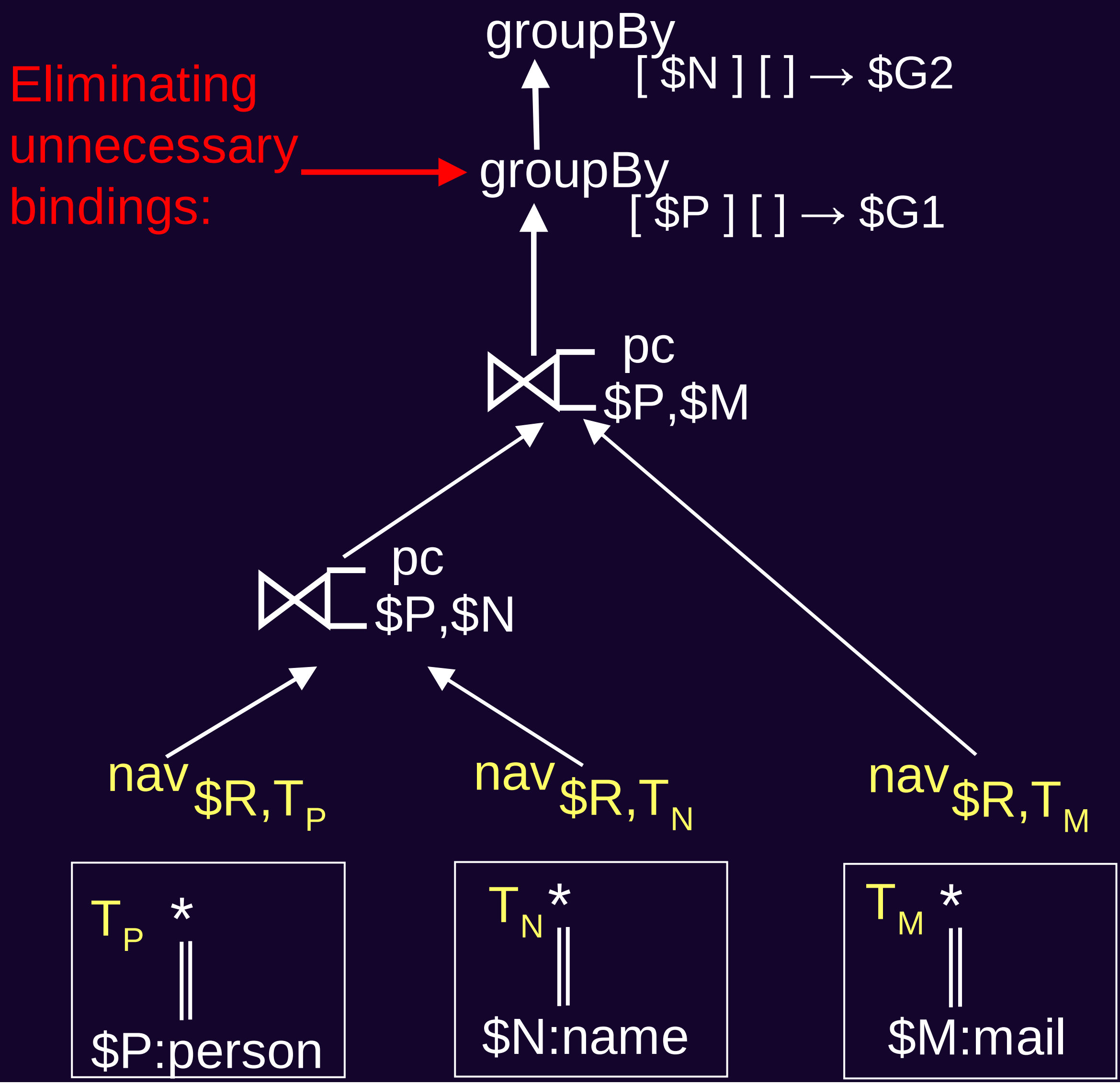


Structural Joins and Optional Navigation

Capturing navigation of:

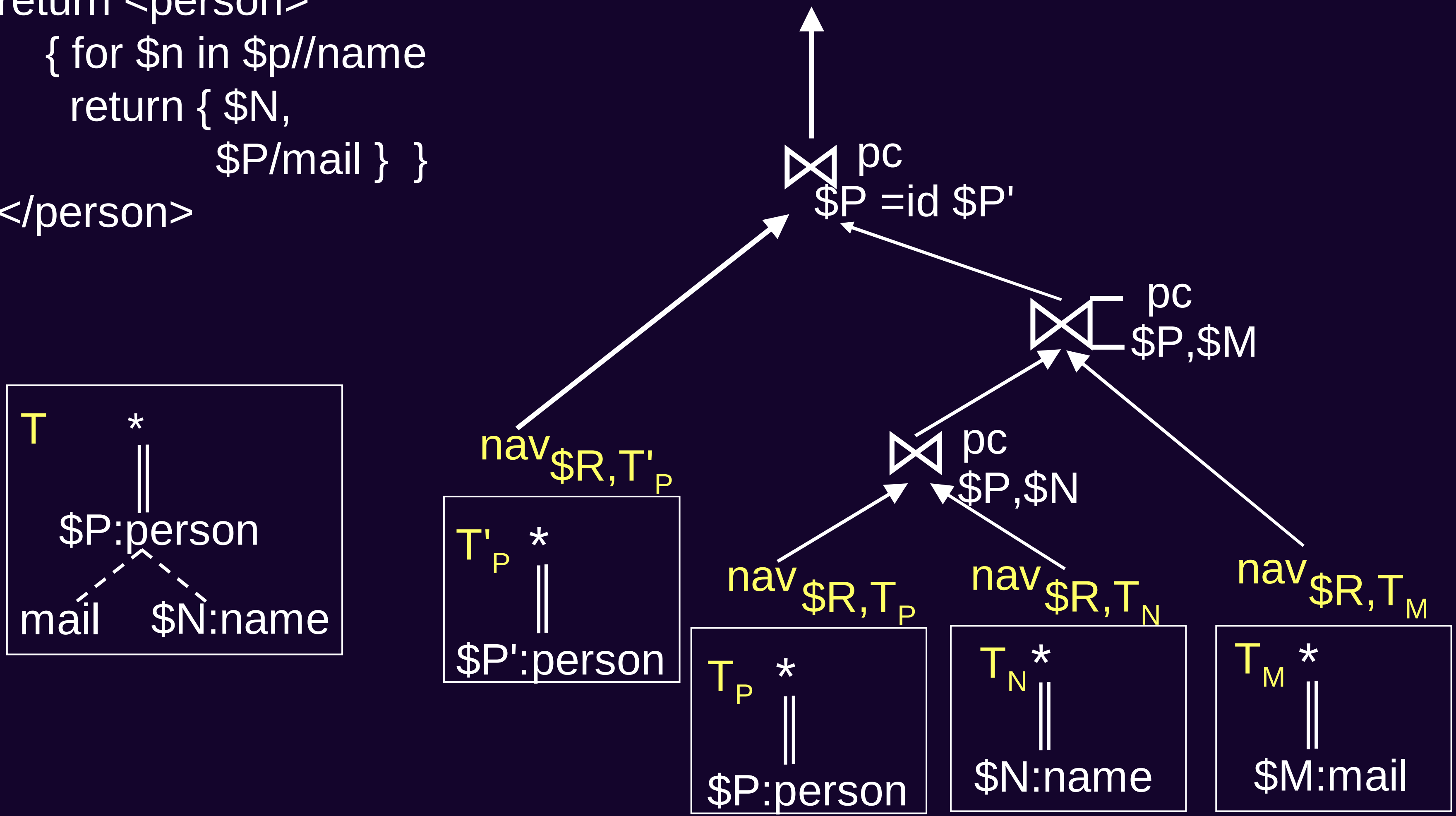
```
for $p in //person
return <person>
   { for $n in $p//name
     return { $N,
              $P/mail }  }
</person>
```



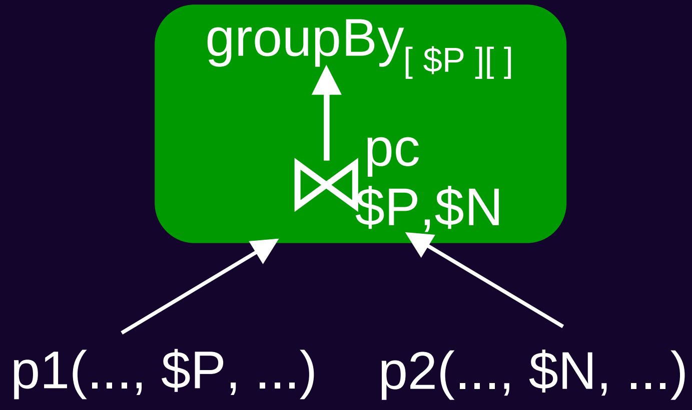


Structural Joins and Optional Navigation: Equivalent Expression

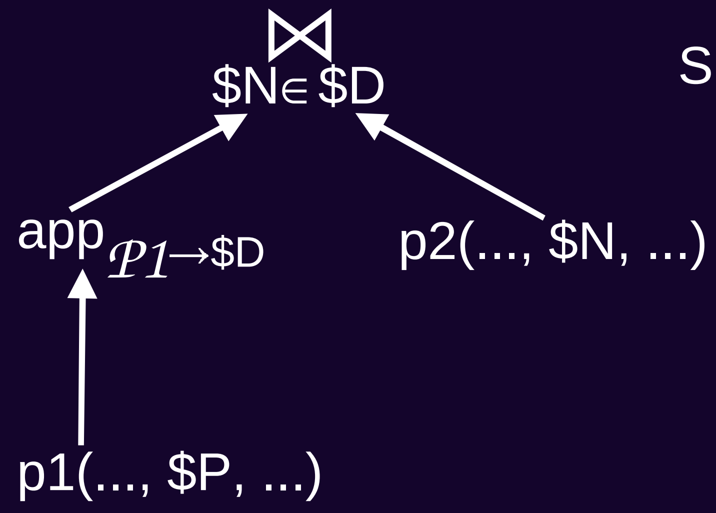
```
for $p in //person
return <person>
   { for $n in $p//name
     return { $N,
                  $P/mail }  }
</person>
```

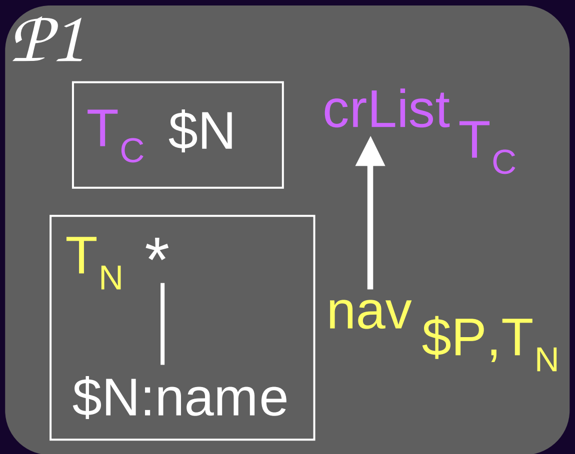


Structural Joins vs. Simple Joins With Navigation



equivalent to:





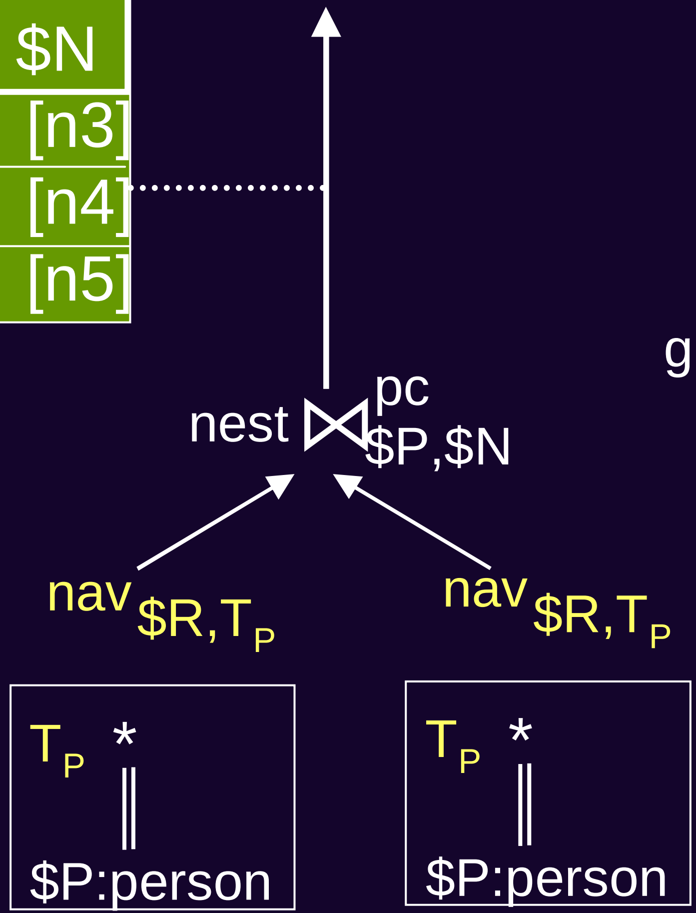
Remark:
groupBy re-builds
original nesting...

Nest structural joins
structural join +
group by the left hand side
variables

Such grouping often
comes cheap

Nest Structural Joins

$P	$N
p1	[n3]
p2	[n4]
p3	[n5]



Remark:
groupBy re-builds
original nesting...

Nest structural joins
structural join +
group by the left hand side
variables

Such grouping often
comes cheap

Related works

Existing tuple-based algebras

Tsimmis and YAT algebras for semistructured data [Cluet et al. 1998]

- Introduced navigation and construction patterns

SAL from U. Tel Aviv [Beeri et al. 1999]

- Close relative of OQL

TAX [Jagadish et al. 2001], Generalized Tree Patterns [Chen et al. 2003], Tree Logical Classes [Paparizos et al. 2004] from Michigan

- Navigation extracts bindings to ‘hidden’ tuples (packaged as trees)
- Grouping tracking navigation
- Also operators for updates (not covered here)

Enosys algebra [Papakonstantinou et al. 2003]

- Collect bindings (nav and join), do nested plans, create XML
- Also present in NEXT system [Deutsch et al. 2004]

Xstasy from U. Pisa [Sartiani et al. 2002]

Context-based algebra [Viglas et al. 2002]

Rainbow algebra from Worcester Polytechnic Inst. [Rundensteiner et al. 2002]

References

References

S. Abiteboul and N. Bidoit. "Non First Normal Form Relations: An Algebra Allowing Data Restructuring", JCSS 1986

A.Arion, A.Bonifati, I.Manolescu and A.Pugliese. "XQueC: Pushing Compression into XML Databases", EDBT 2004 (extended v. 2005)

S.Al-Khalifa, H.Jagadish, J.Patel, Y.Wu, N.Koudas and D.Srivastava. "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", ICDE 2002

C.Beeri and Y.Tzaban. "SAL: An Algebra for Semistructured Data and XML", WebDB 1999

K.Beyer, R.Cochrane, L.Colby, F.Ozcan and H.Pirahesh. "XQuery for Analytics: Chellanges and Requirements", XIME-P 2004

References (2)

N.Bruno, N.Koudas and D.Srivastava. "Holistic Twig Joins: Optimal XML Pattern Matching", SIGMOD 2002

Z.Chen, H.Jagadish, L.Lakshmanan and S.Paparizos. "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery", VLDB 2003

S.Cluet, C.Delobel, J.Simeon and K.Smaga. "Your Mediator Needs Data Conversion !", SIGMOD 1998

S.Cluet and G.Moerkotte. "Nested Queries in Object Bases", tech. report, 1995

A.Deutsch, Y.Papakonstantinou and Y.Xu. "The NEXT logical framework for XQuery", VLDB 2004

References (3)

T.Fiebig, S.Helmer, C.Kanne, G.Moerkotte, J.Neumann, R.Schiele and T.Westmann. "Anatomy of a Native XML Base Management System", VLDB Journal 2002

D.Florescu, C.Hillery, D.Kossmann, P.Lucas, F.Riccardi, T.Westmann, M.Carey and A.Sundararajan. "The BEA Streaming XQuery Processor", VLDB Journal 2004

H.V.Jagadish, L.Lakshmanan, D.Srivastava and K.Thompson. "TAX: A Tree Algebra for XML". DBPL, 2001

I. Manolescu and Y.Papakonstantinou. "Report on the first XIME-P Workshop", SIGMOD Record, 2004

P.O'Neil, E.O'Neil, S.Pal, I.Cseri, G.Schaller and N.Westbury. "ORDPATHs: Insert-friendly XML Node Labels", SIGMOD 2004

References (4)

Y.Papakonstantinou, V.Borkar, M.Orgiyan, K.Stathatos, L.Suta, V.Vassalos and P.Velikhov. "XML Queries and Algebra in the Enosys Integration Platform", DKE 2003

S.Paparizos, Y.Wu, L.Lakshmanan and H.Jagadish. "Tree Logical Classes for the Efficient Evaluation of XQuery", SIGMOD 2004

C.Sartiani and A.Albano. "Yet Another Query Algebra for XML Data", IDEAS 2002

S.Viglas, L.Galanis, D.DeWitt, D.Maier and J.Naughton. "Putting XML Query Algebras into Context", tech. report, 2002

References (5)

XQuery 1.0 and XPath 2.0 Data Model
www.w3.org/TR/xpath-datamodel

XQuery 1.0 and XPath 2.0 Functions and Operators

www.w3.org/TR/xpath-functions

XQuery 1.0 Formal Semantics
www.w3.org/TR/2005/WD-query-semantics

Thank you