Evaluating Cypher queries as algebraic expressions within RedisGraph

Roi Lipman
RedisLabs

RedisGraph

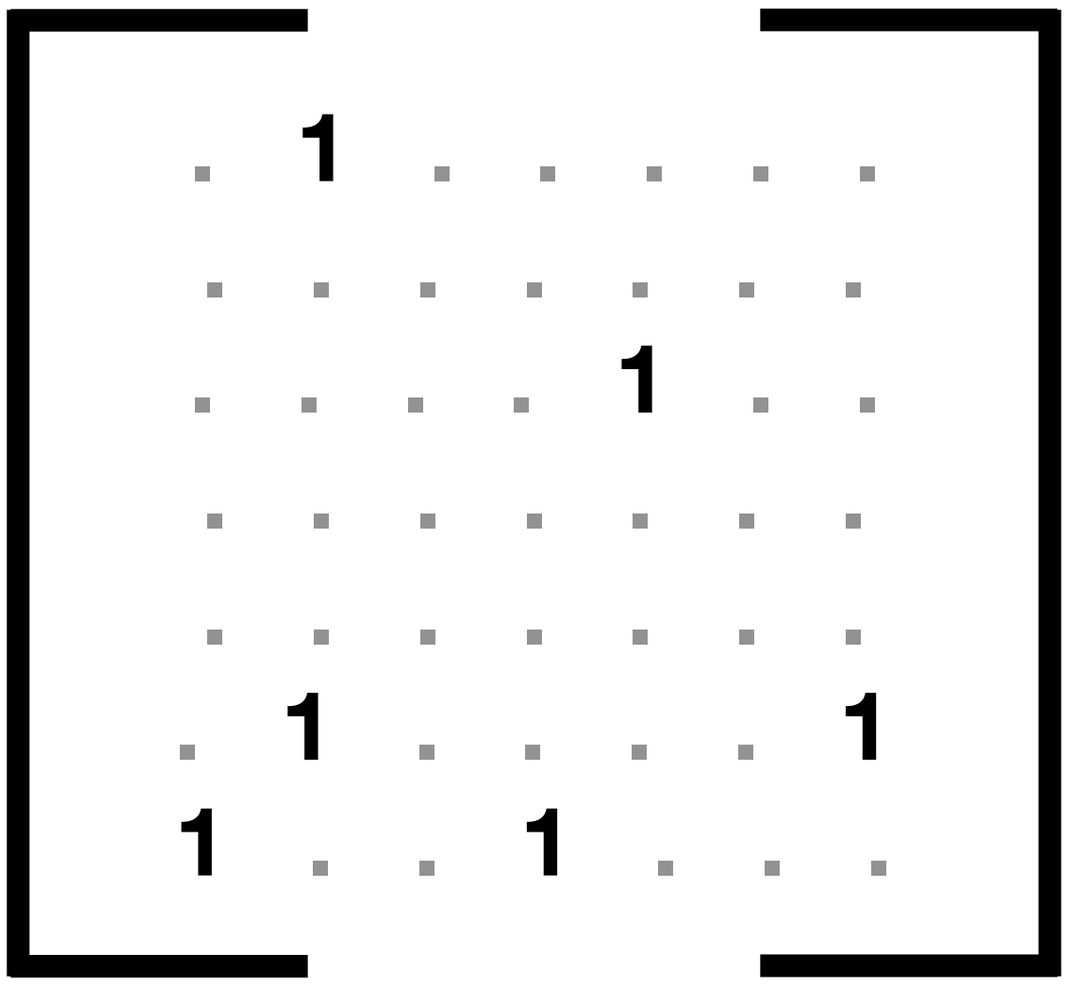
property graph representation

4 types of matrices

- THE adjacency matrix
- Label matrices
- Relation matrices
- Relationship mapping matrices

  All matrices share the same dimensions
  #rows = #columns = #nodes

THE Adjacency matrix



THE Adjacency matrix

- **Boolean**

THE Adjacency matrix

- Boolean
- **Relation type agnostic**

THE Adjacency matrix

- Boolean
- Relation type agnostic
- **Label agnostic**

THE Adjacency matrix

- **Boolean**
- **Relation type agnostic**
- **Label agnostic**
- **Directional, M[i, j] = 1**

  Node i connected to Node j
  ***might be multiple times with different relationship types**

Label matrix

$$
\begin{bmatrix}
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & 1 & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & 1 & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 1
\end{bmatrix}
$$

Label matrix

- **Boolean**

Label matrix

- Boolean
- **One for each node type**

Label matrix

- **Boolean**
- **One for each node type**
- **Diagonal**
  L[i, i] = 1
  Node with ID i is labeled as L

Relation matrix

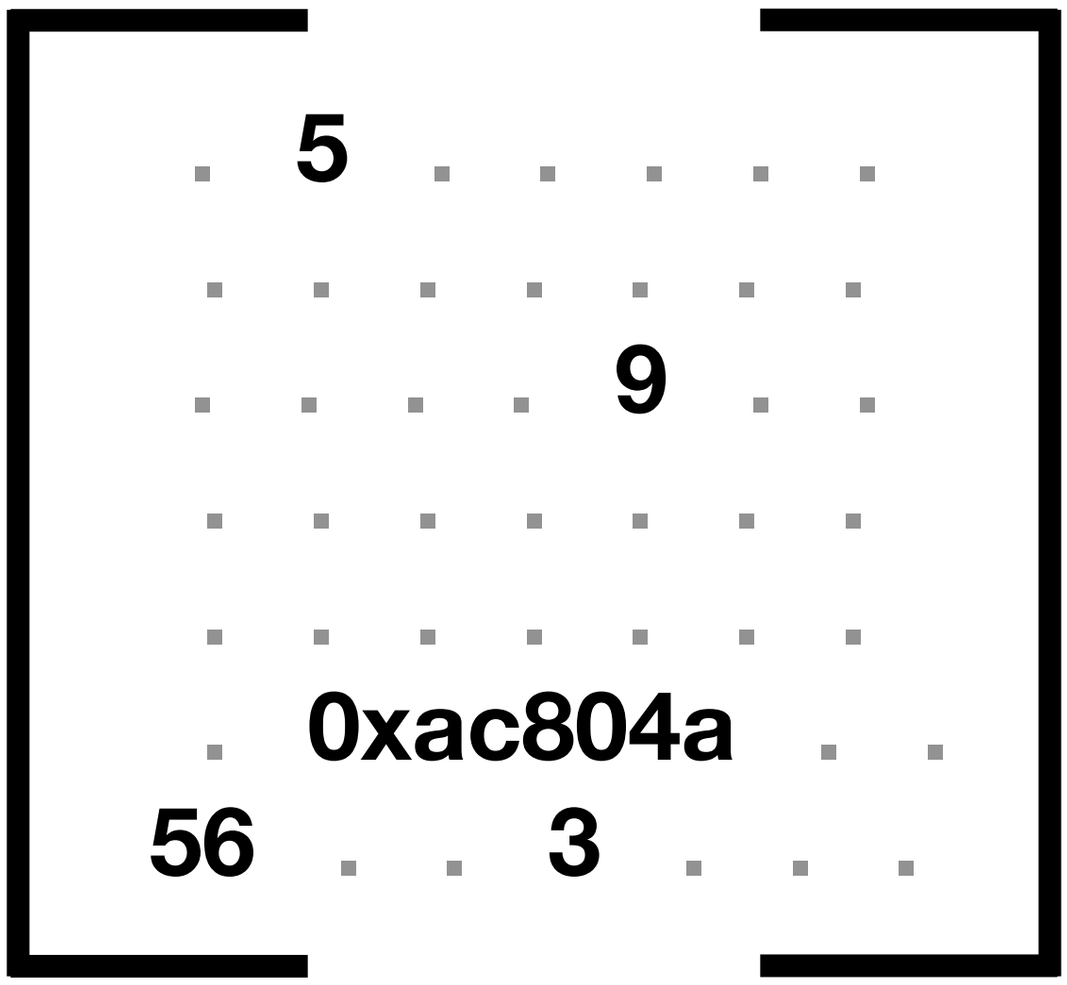
$$
\begin{bmatrix}
 & 1 & & & & & \\
 & & & & & & \\
 & & & & 1 & & \\
 & & & & & & \\
 & & & & & & \\
 & & & & & & \\
 & 1 & & & & 1 & \\
1 & & & 1 & & & \\
 & & & & & &
\end{bmatrix}
$$

Relation matrix

- **Boolean**
- **Label agnostic**
- **Directional, R[i, j] = 1**

  Node i connected to Node j
  ***might be multiple times with relationship type R**

Relation mapping matrix



Relation mapping matrix

- **64bit entries**

  RM[i, j] - either edge ID if node i is connected to node j with a single edge of type R

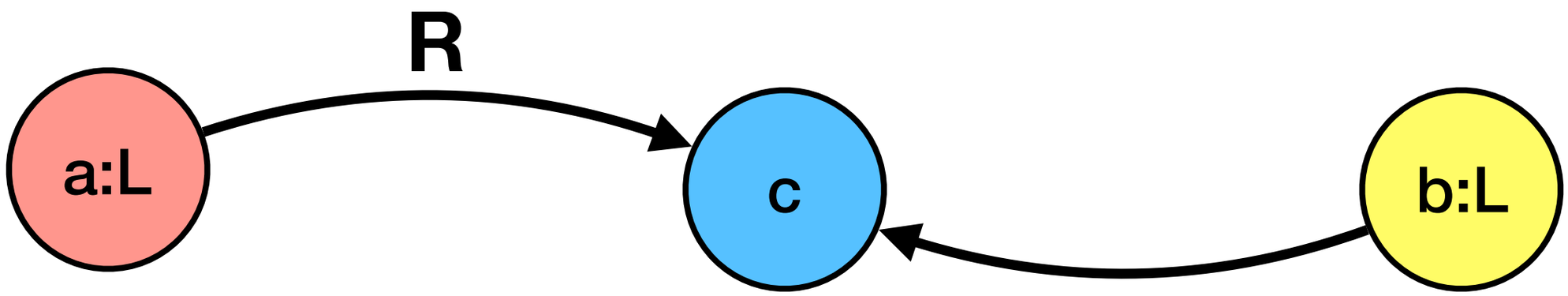
  Pointer to edge IDs array if node i is connected to node j with multiple edges of type R

Query Execution Traversals

Chain:

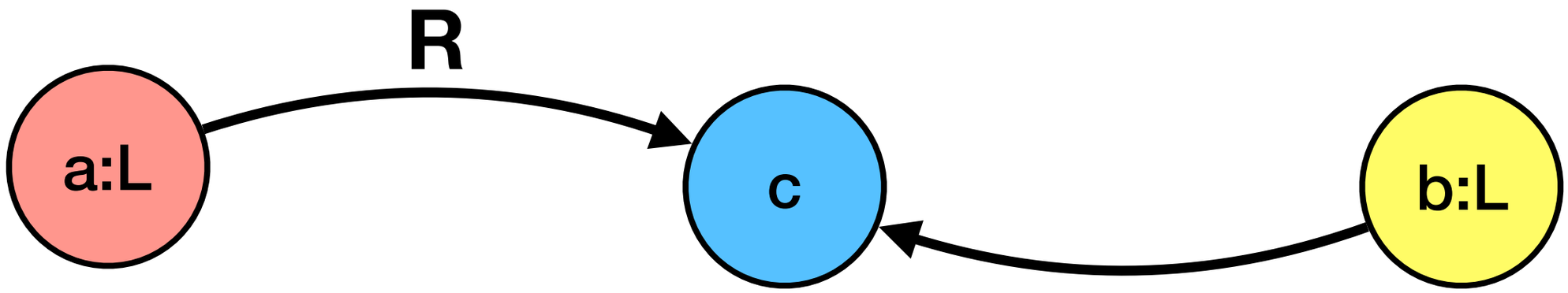
A path where each node on the path appears only once and the sum of its In/Out degrees < 3

Query Execution Traversals



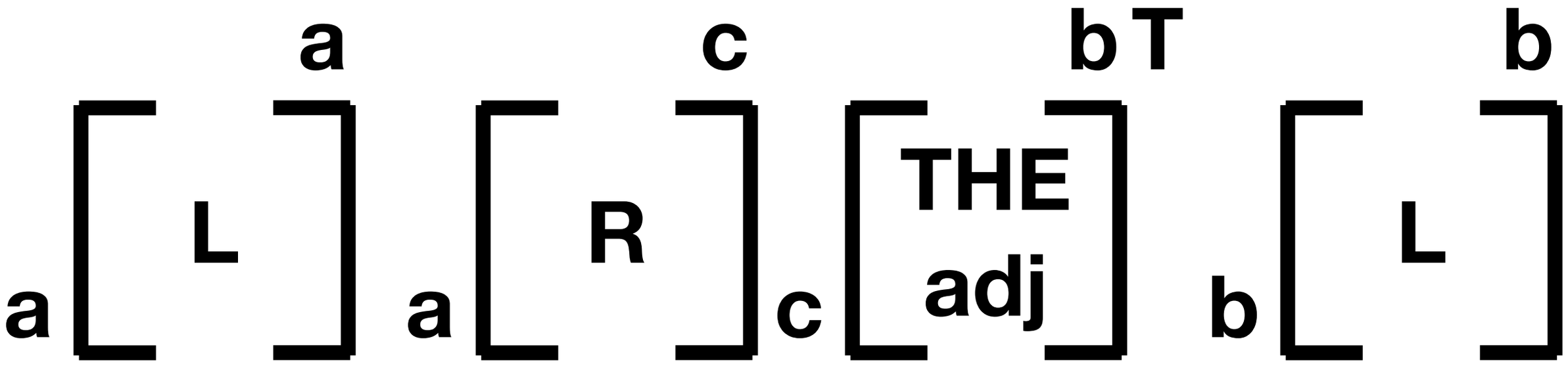
MATCH (a:L)-[:R]->(c)<-[]-(b:L)
RETURN b

Algebraic expression



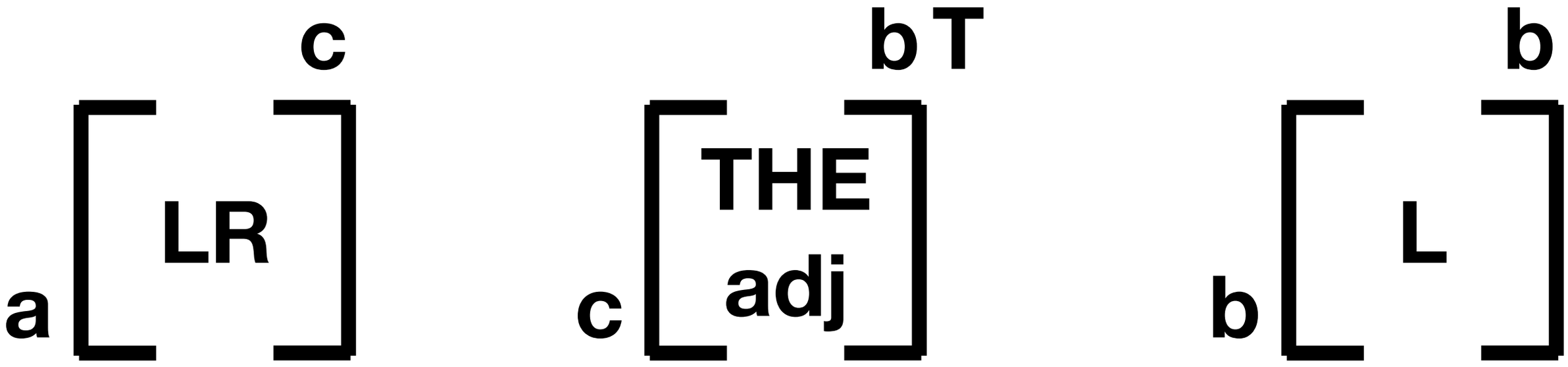
$$a\left[\begin{matrix}L\end{matrix}\right]_{a}\quad a\left[\begin{matrix}R\end{matrix}\right]_{c}\quad c\left[\begin{matrix}\mathrm{THE}\\ \mathrm{adj}\end{matrix}\right]_{b}\quad b\left[\begin{matrix}L\end{matrix}\right]_{b}$$

Evaluation

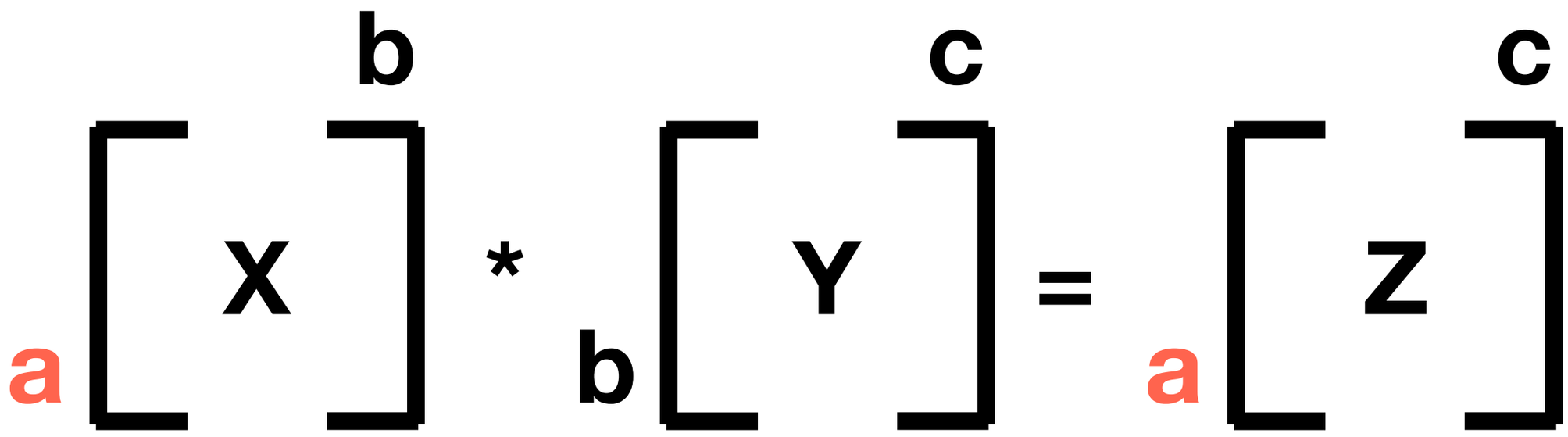


Matrix multiplication is associative
Motivation keep sparsity

Evaluation

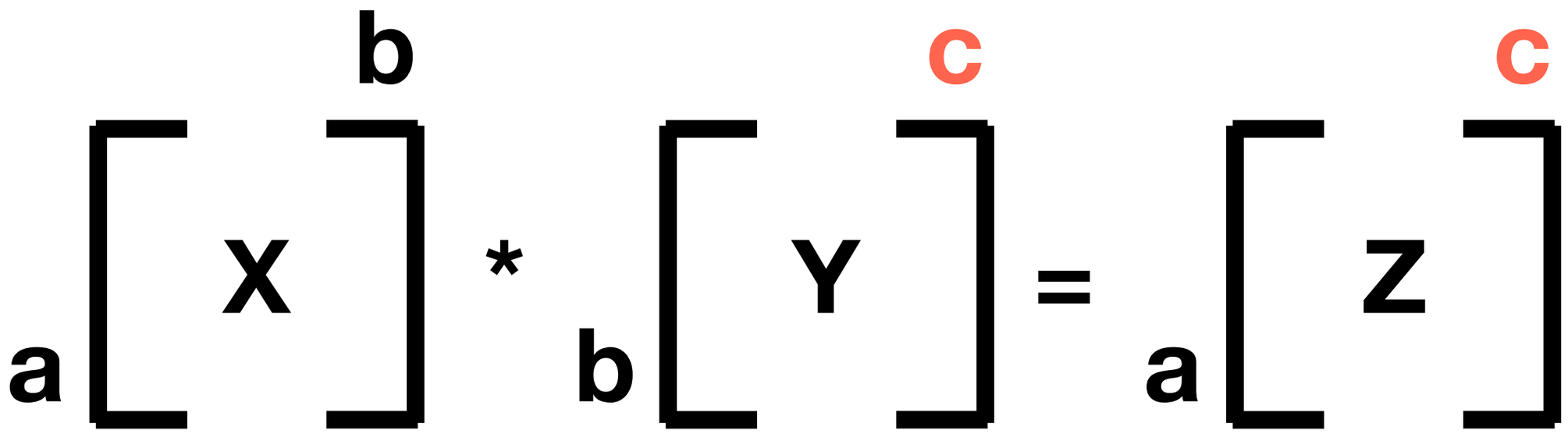


Evaluation



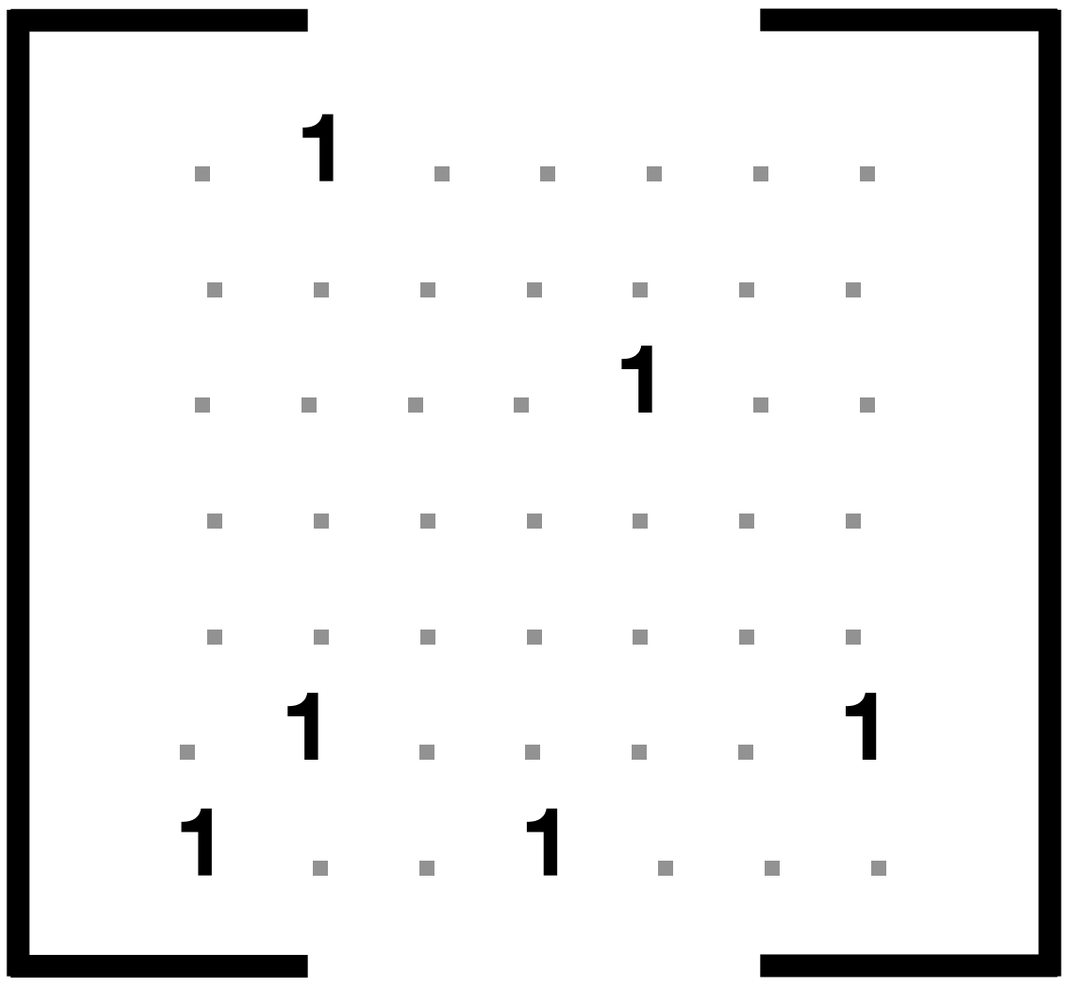
Row domain is “Sticky”

Evaluation



Column domain changes with every “hop”

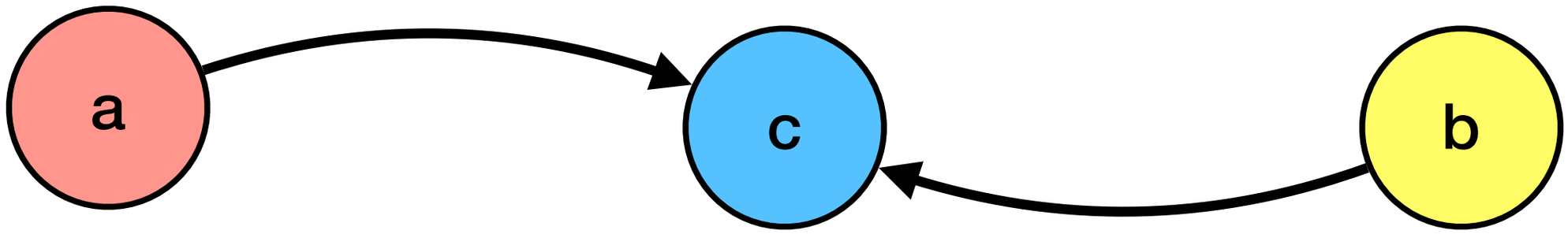
“2D”

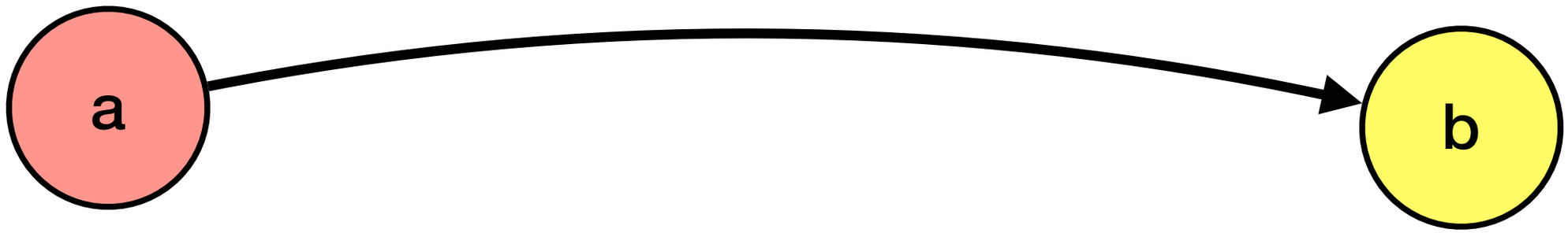


MxM
Lose of information

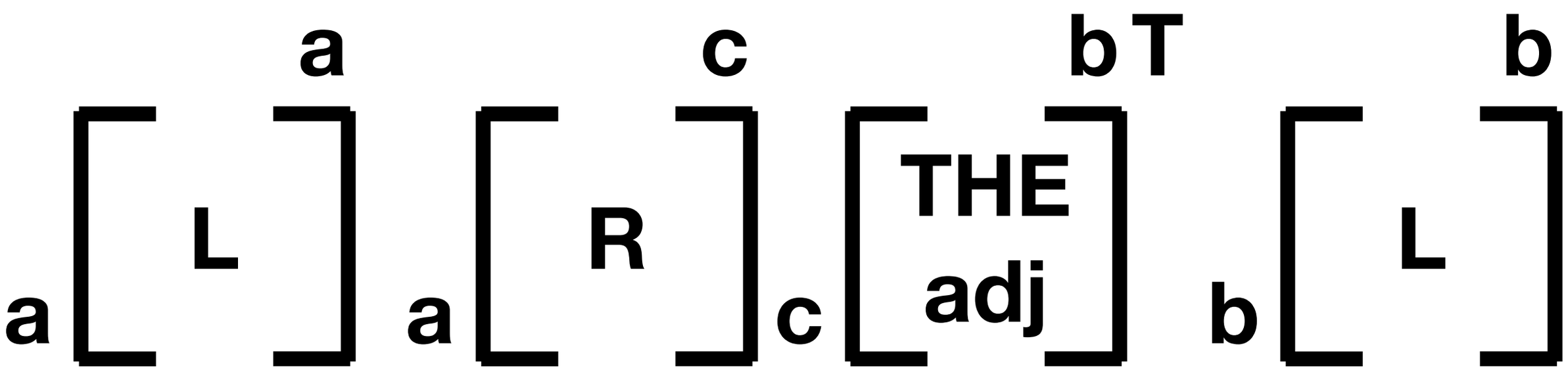
$$
\begin{matrix} & & b & & & & c & & & & c & \\ & \begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix} & & * & & \begin{bmatrix} & & \\ & Y & \\ & & \end{bmatrix} & & = & & \begin{bmatrix} & & \\ & Z & \\ & & \end{bmatrix} & \\ a & & & & b & & & & a & & \end{matrix}
$$

a is connected to b
But how?





```
MATCH (a:L)-[:R]->(c)<-[]-(b:L)
RETURN c
```



Intermediate entities

- Referred node/edge

  RETURN n,e

Intermediate entities

- Variable length edges

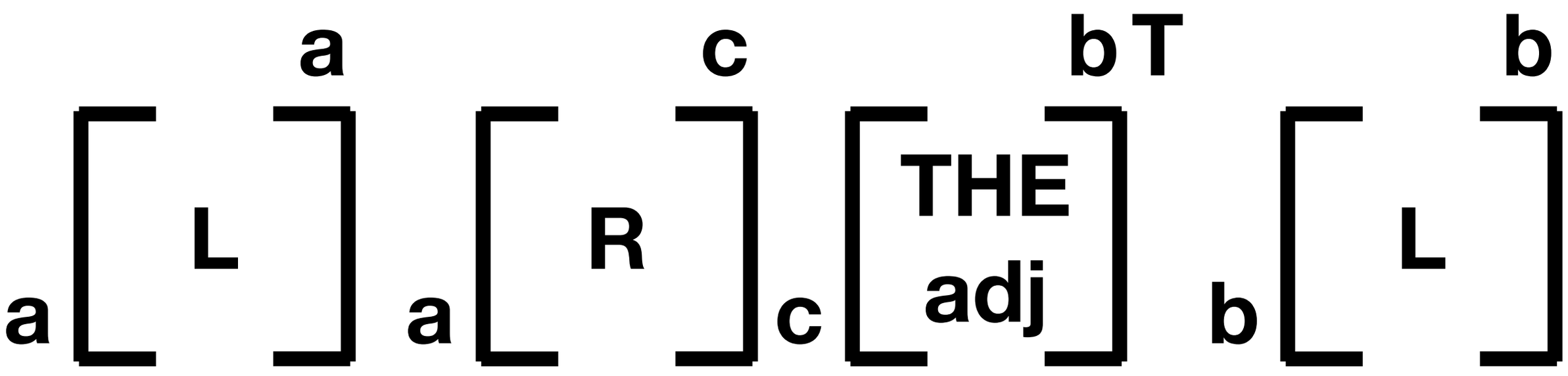
  MATCH (a)-[e:*2..4]->(b)

Intermediate entities

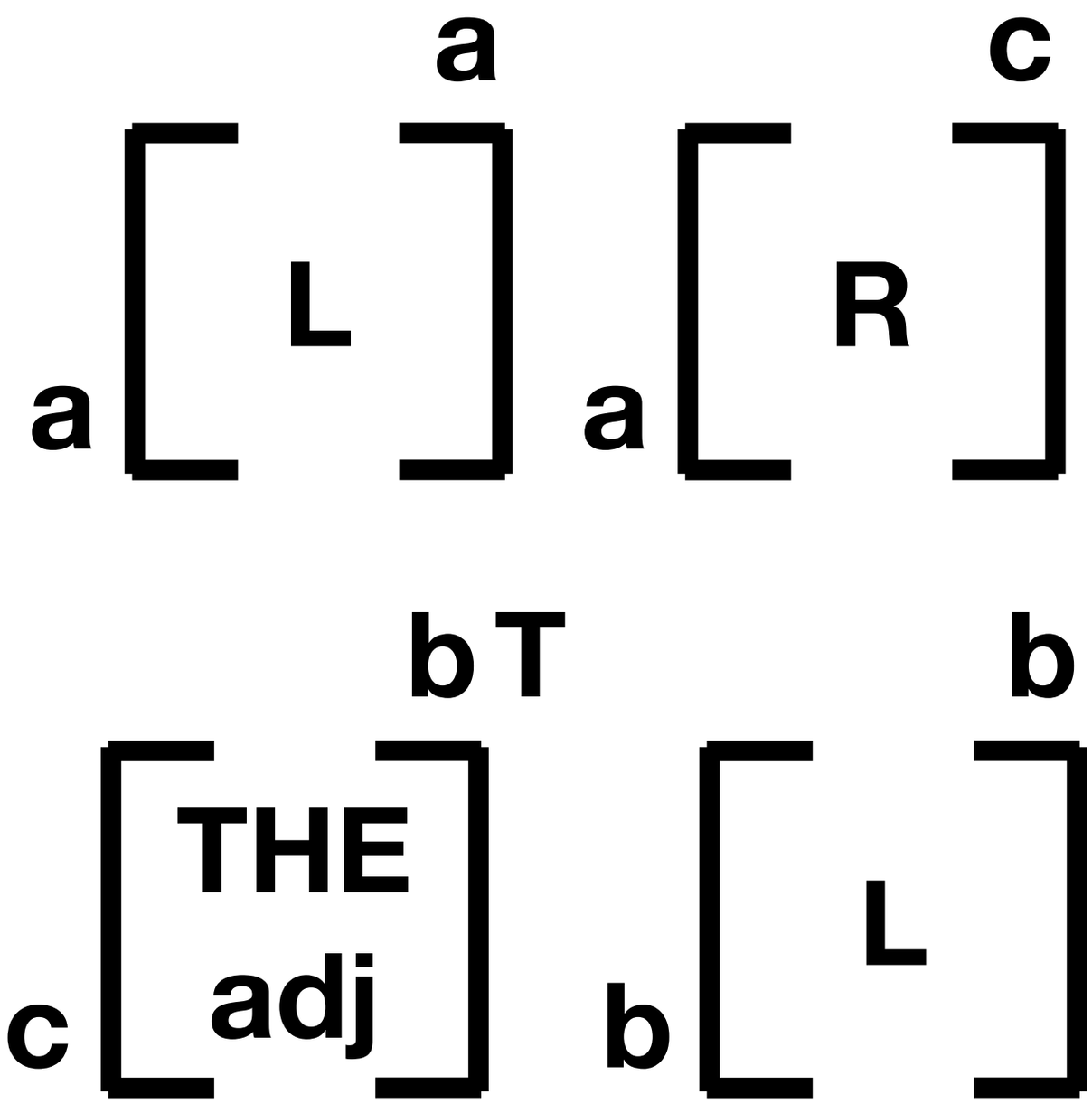
- Filtered entities

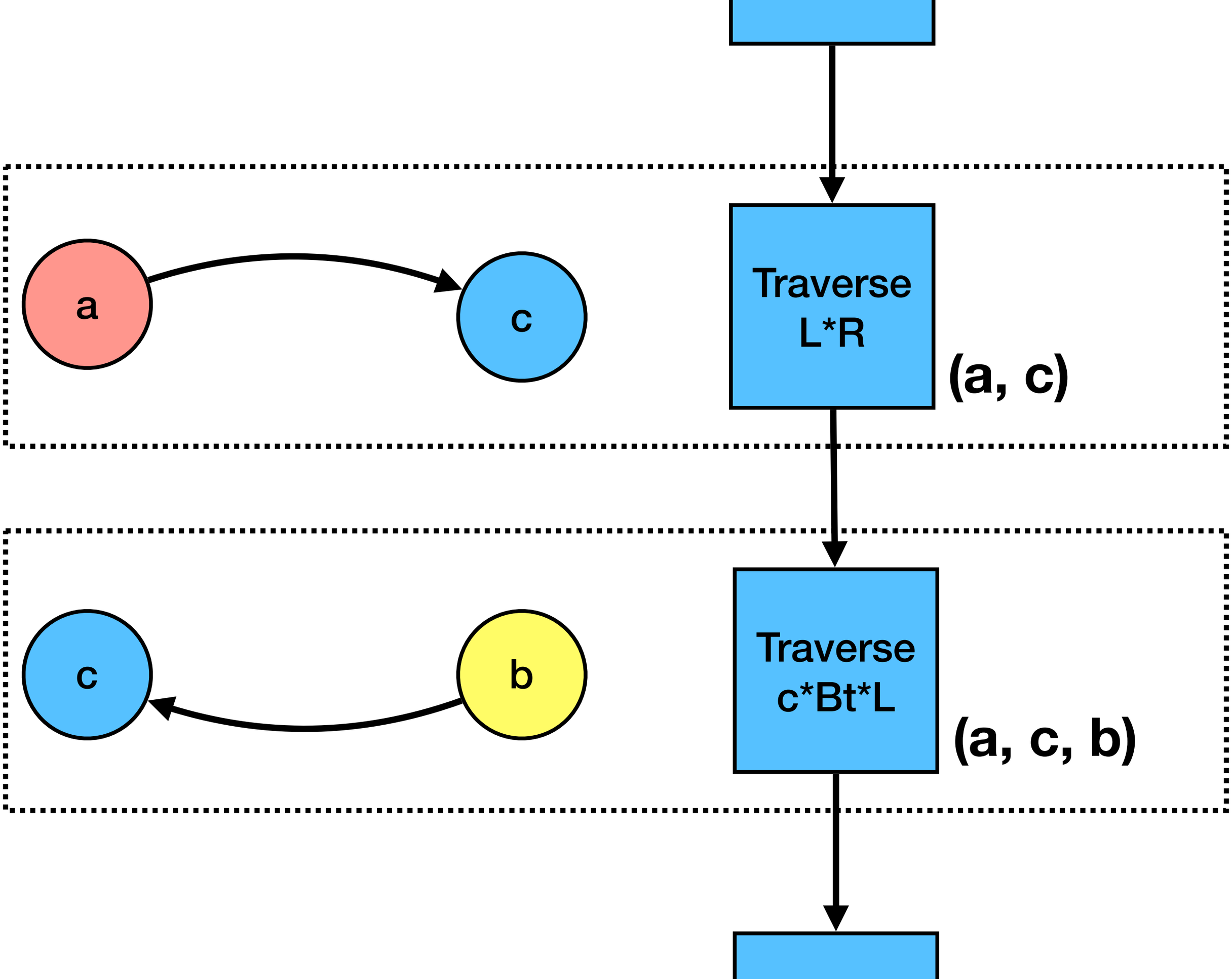
  WHERE n.v = 34

Break on intermediate

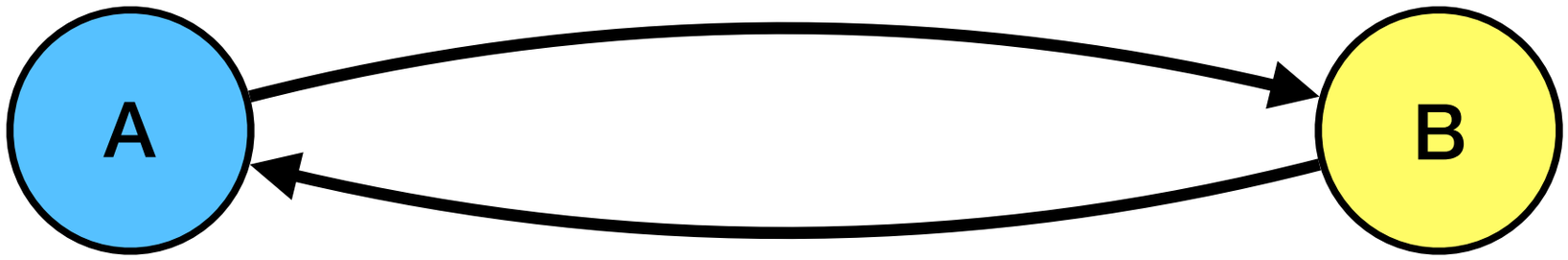


Break on intermediate



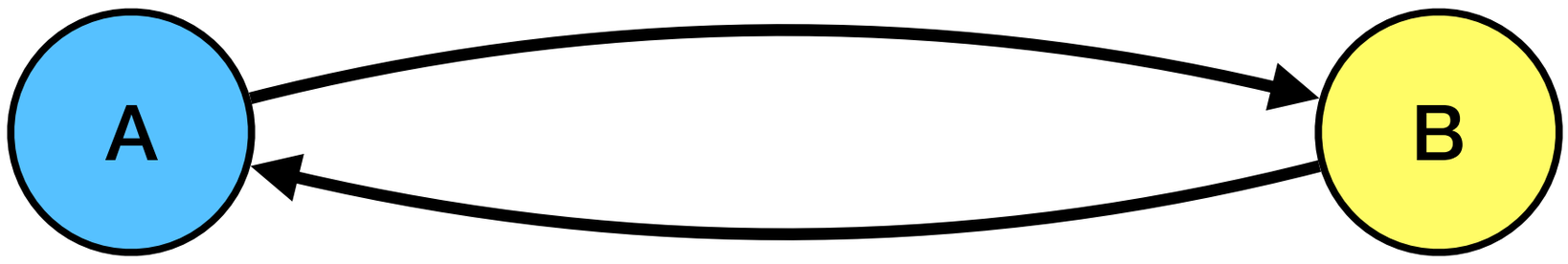

a
c
Traverse
L*R
(a, c)
c
b
Traverse
c*Bt*L
(a, c, b)

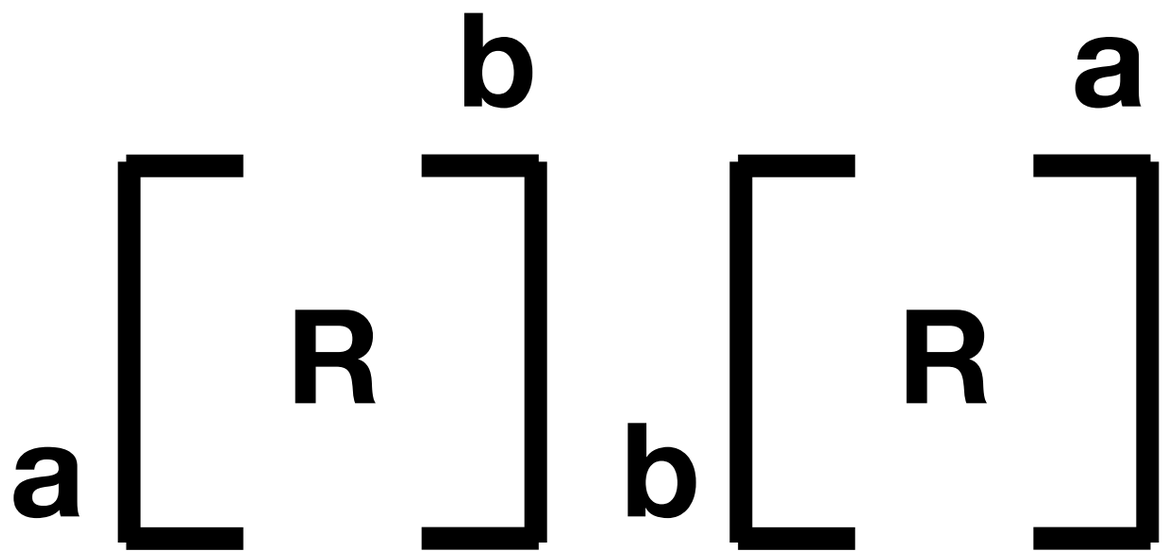
High degree nodes & Cycles



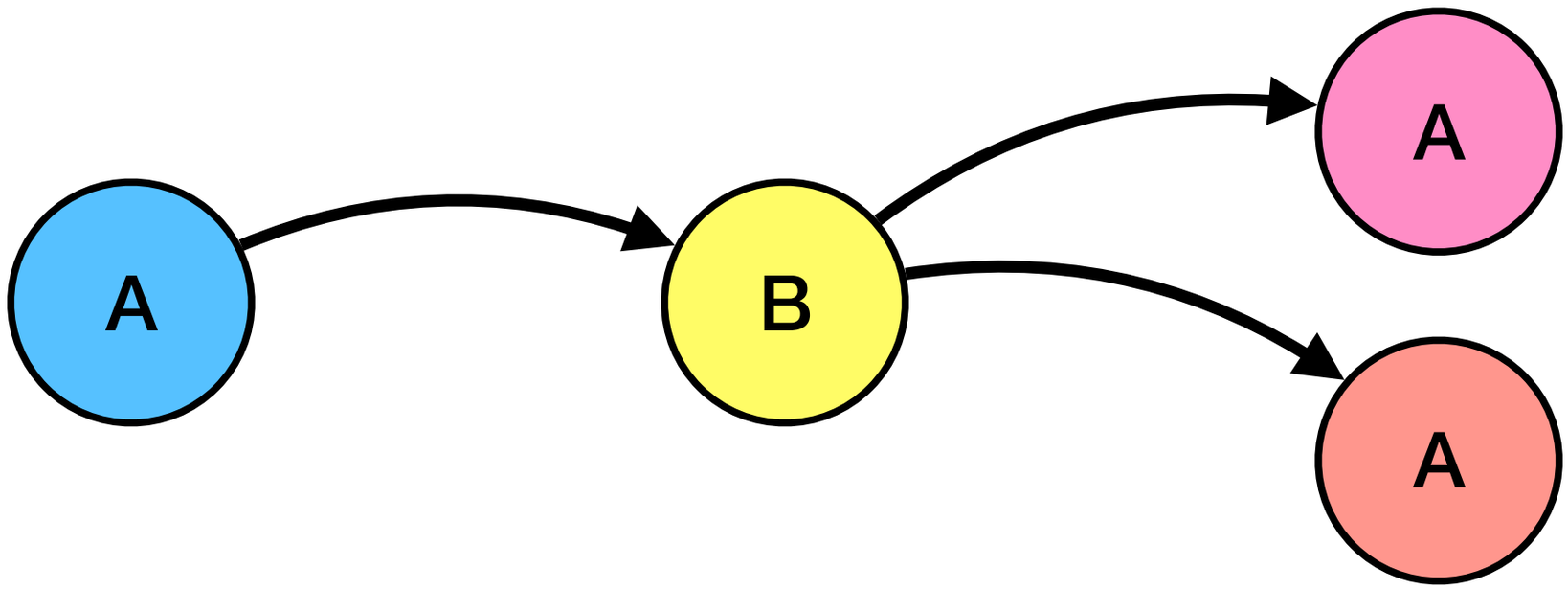

```
MATCH (a)-[]->(b)-[]->(a)
RETURN a
```

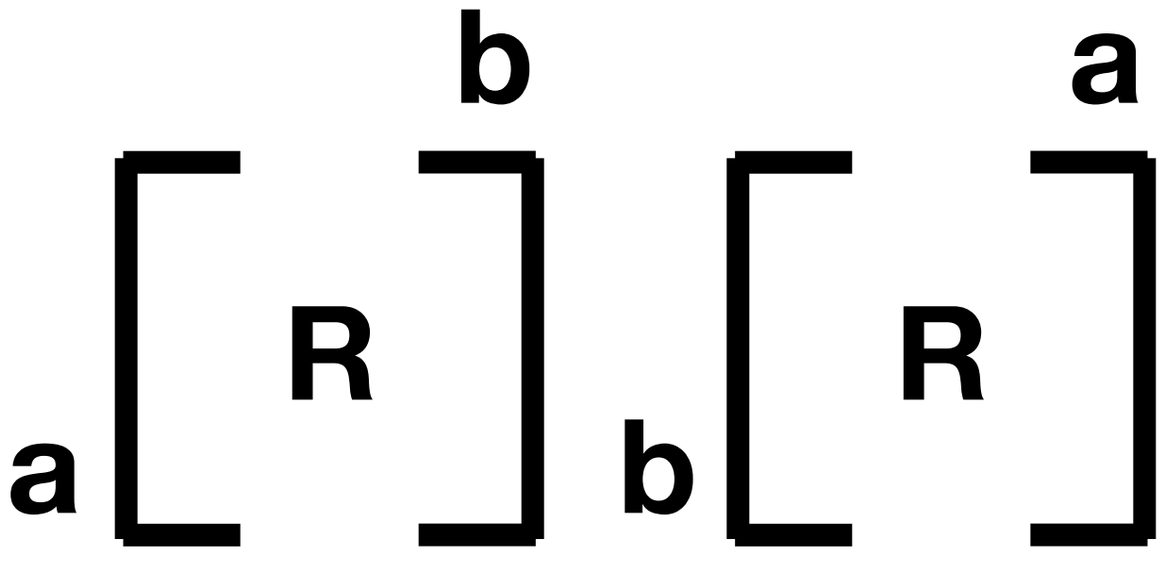
Cycle





Cycle





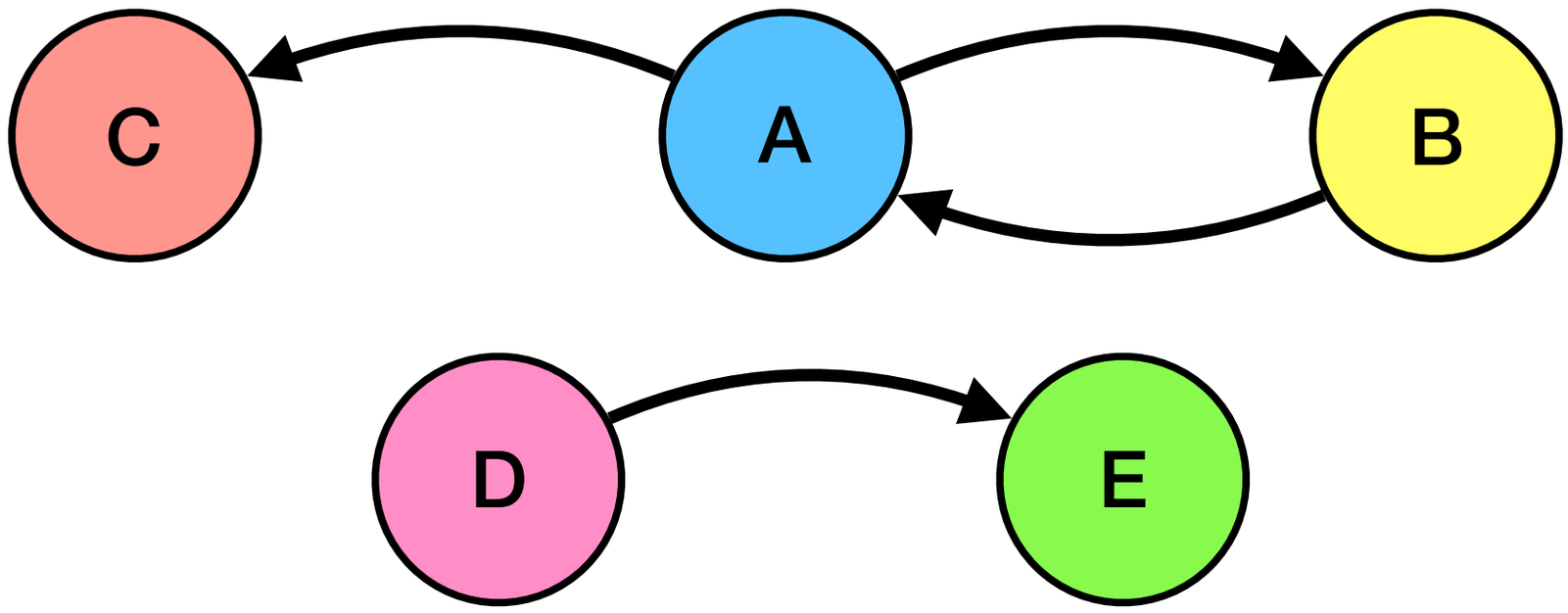
Improved Algebraic expressions construction

```
Exps = []
QG = QueryGraph(AST)
CCS = ConnectedComponents(QG)

For CC in CCS
    while(CC not empty)
        P = LongestPath(CC)
        E = AlgebricExpressionFromPath(P)
        Es = AlgebricExpressionBreakInter(E)
        Exps += Es
        CC -= P
```

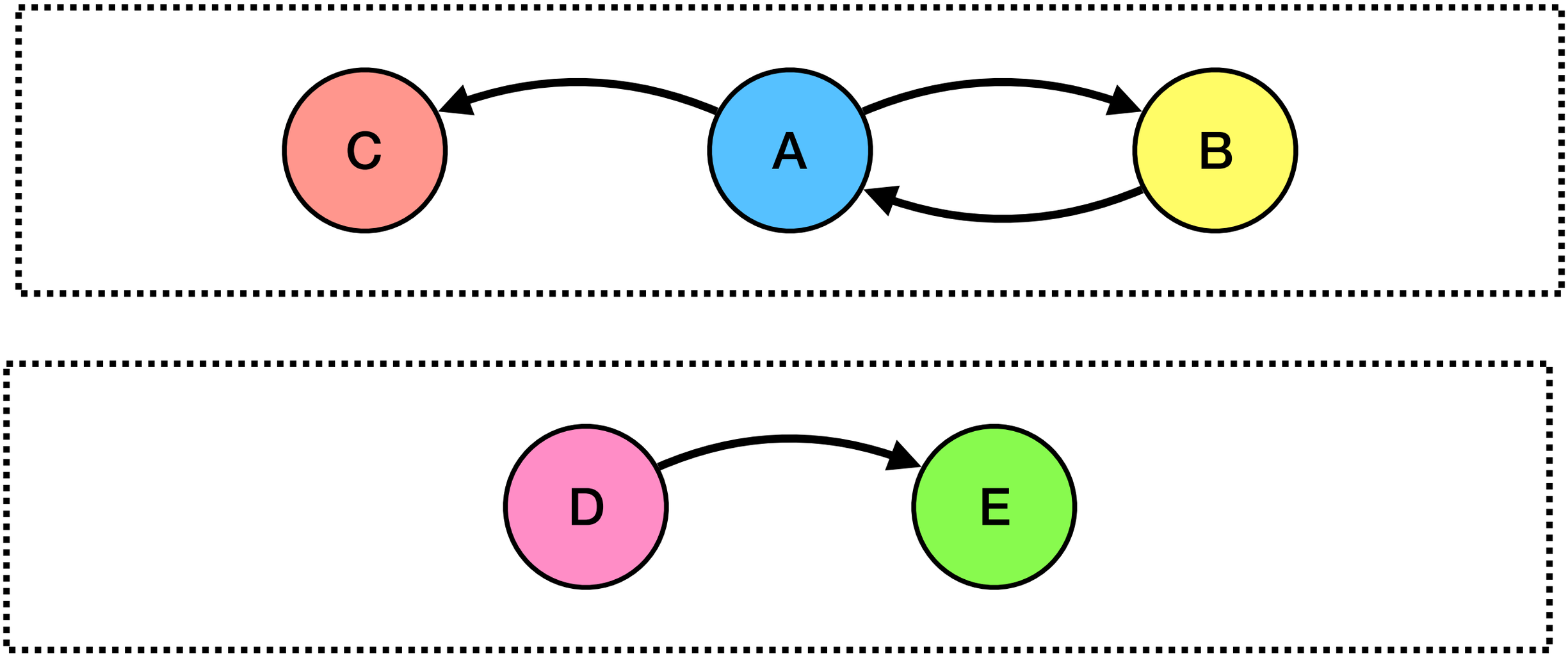
Pseudo code

Query Graph

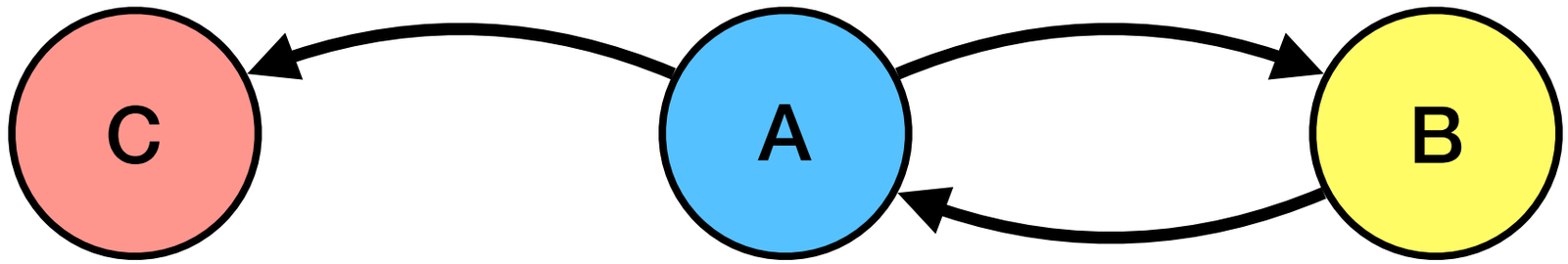


MATCH (a)-[]->(b)-[]->(a),
(a)-[]->(c), (d)-[]->(e)

Connected Components



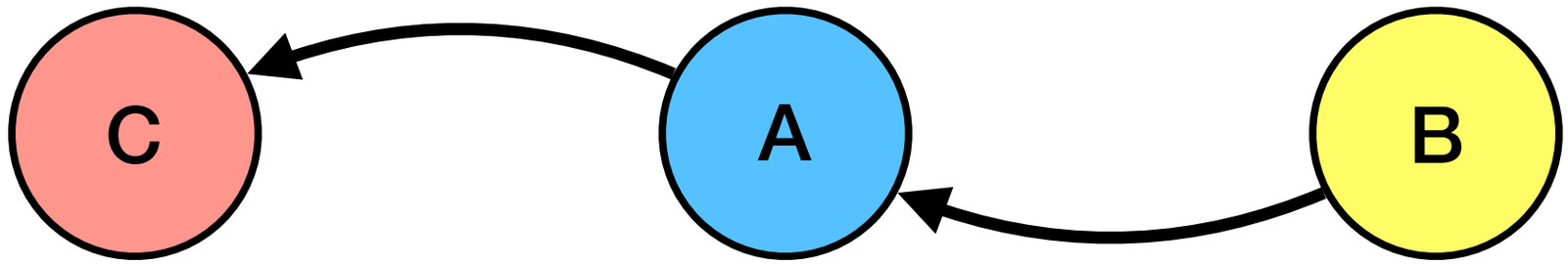
Longest path



P0 = {C, A, B}

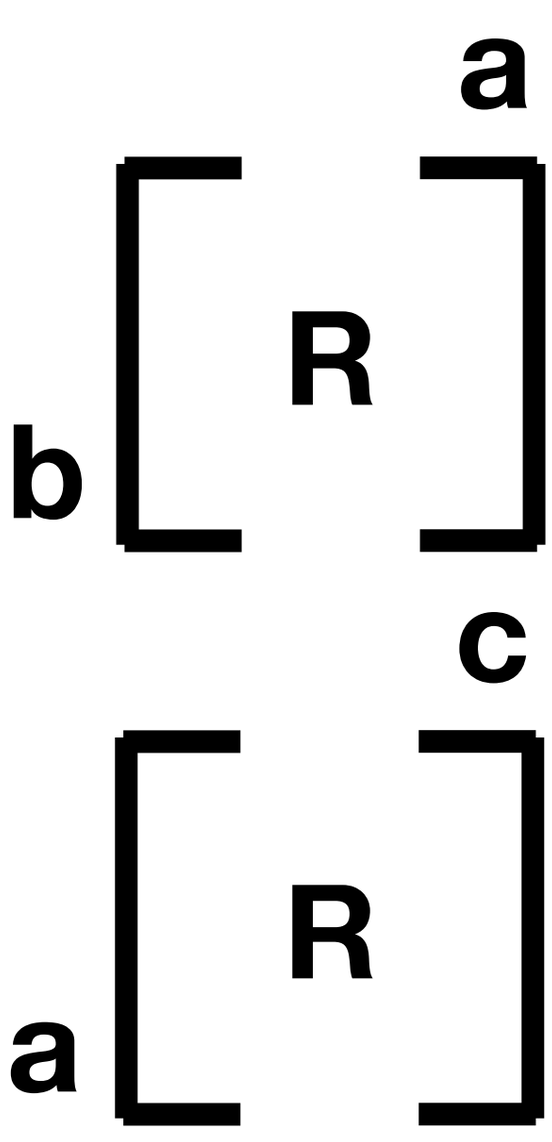
P1 = {B, A, C}

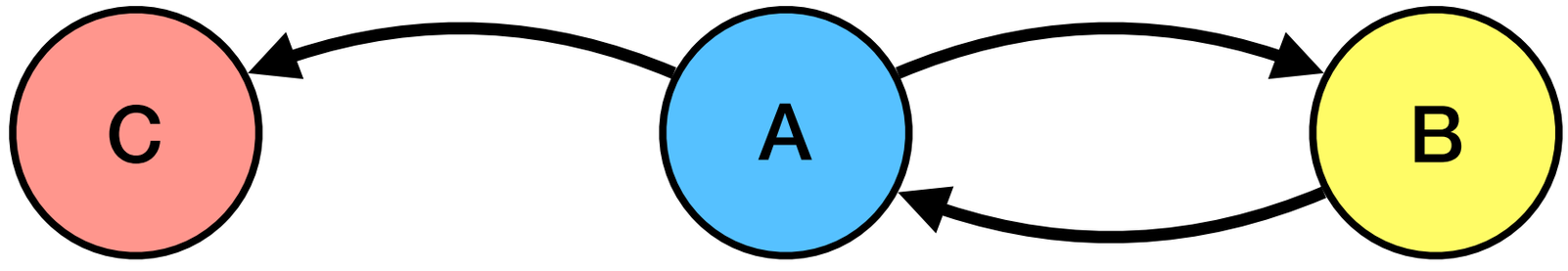
Algebraic expression from path (chain)



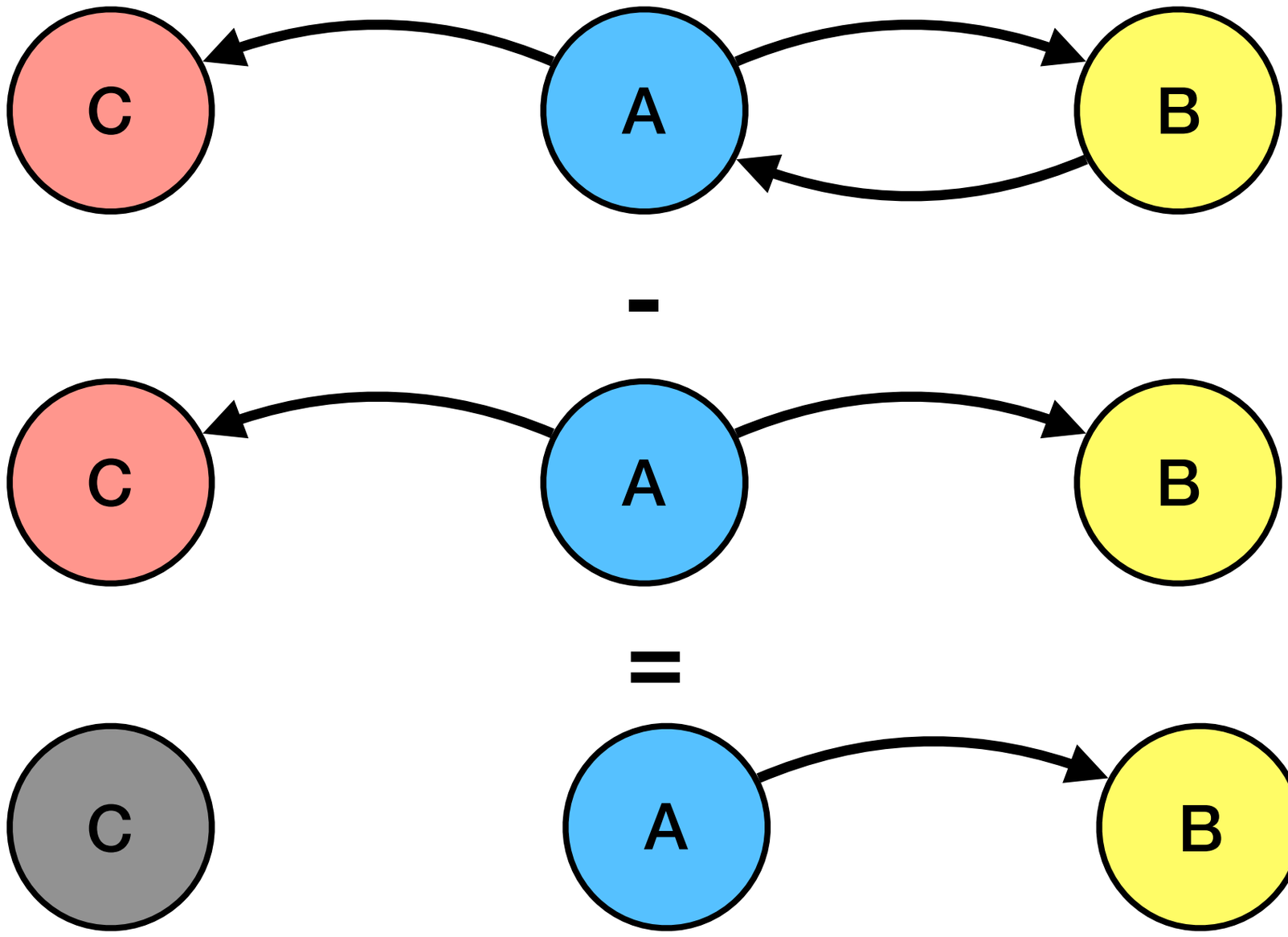
$$b\left[\,a\,\right]R\;a\left[\,c\,\right]R$$

Break on intermediates

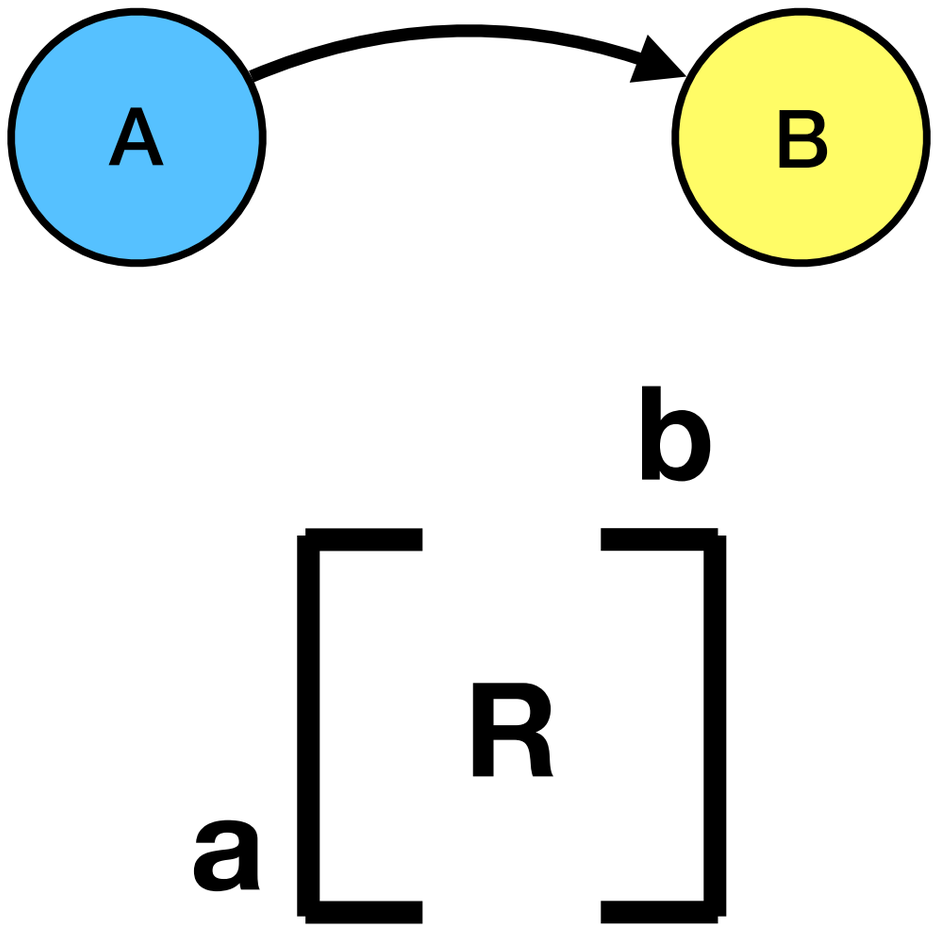




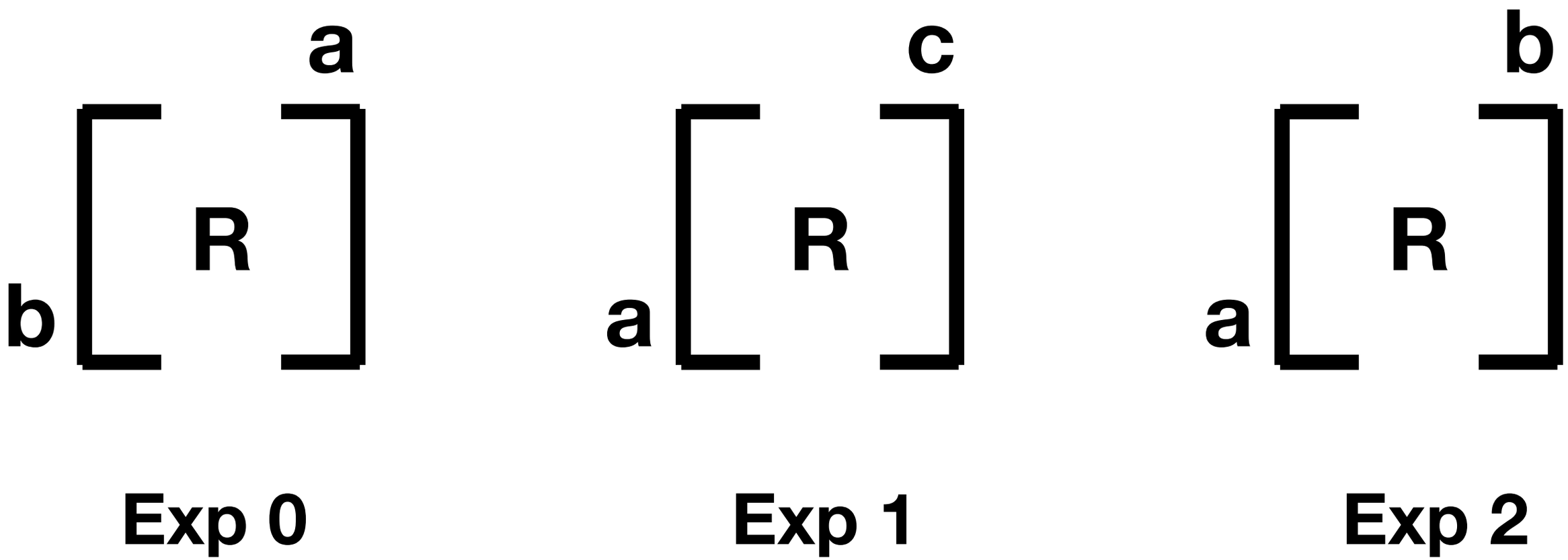
Remove path from Query Graph



Algebraic expression from path (chain)



Expressions



Exp 0 **Exp 1** **Exp 2**

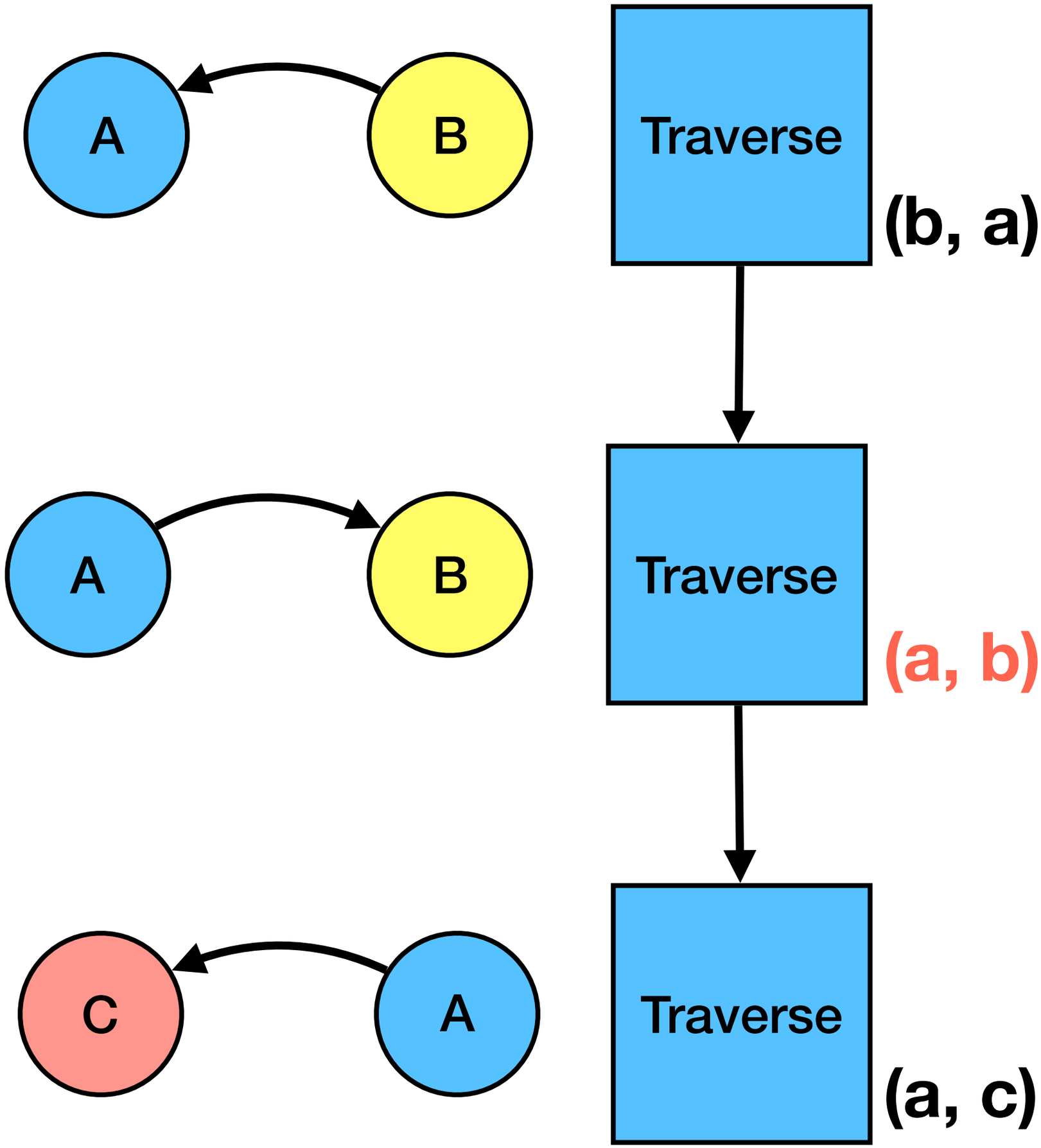
Arrangements

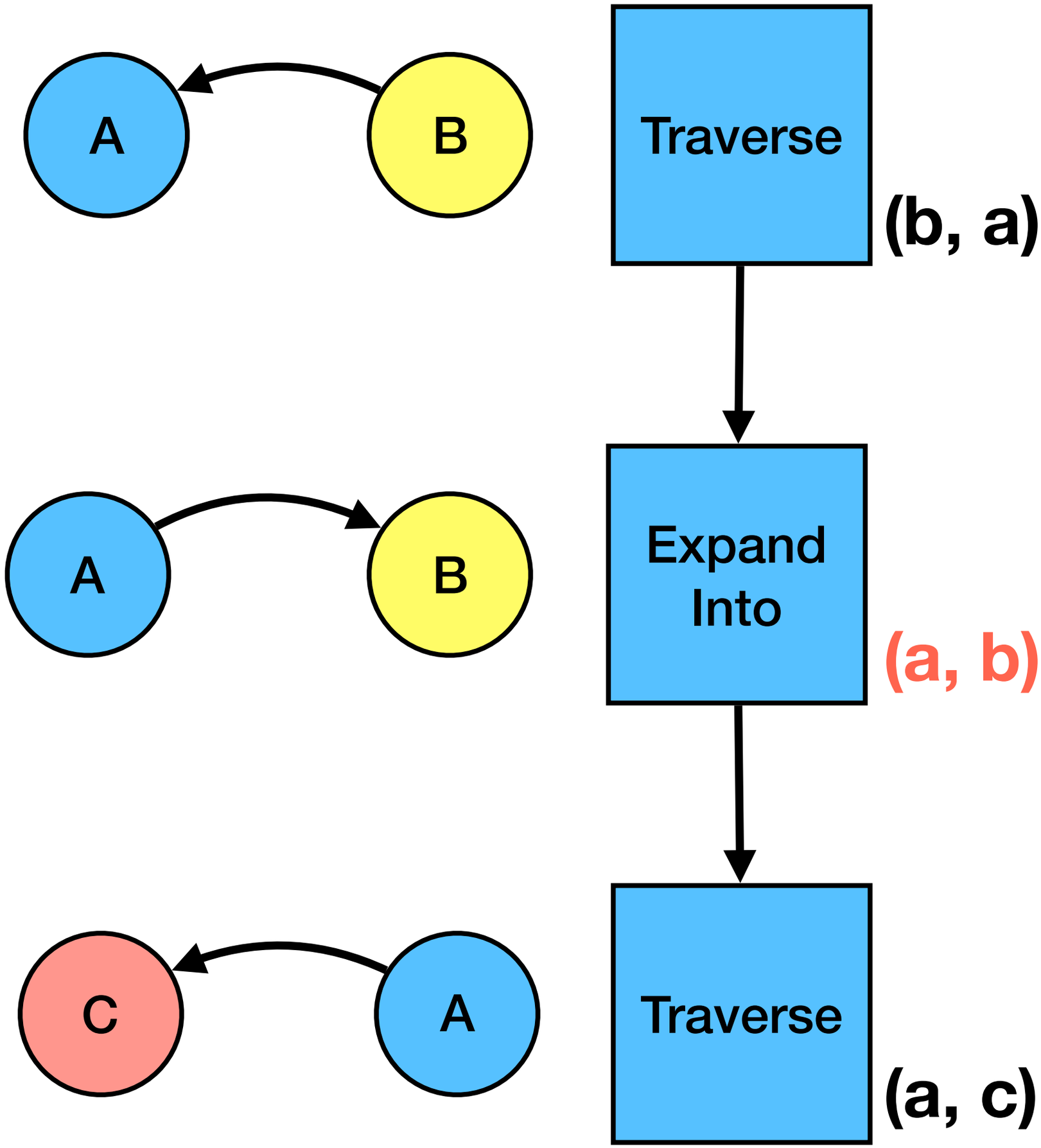
Permute ([Exp0, Exp1, Exp2]), N!

Pick best arrangement to be executed in order

Valid arrangement:
1. Expi either src or dest node is already resolved by previous expression.

2. Early filters


A
B
Traverse
(b, a)
A
B
Traverse
(a, b)
C
A
Traverse
(a, c)


A
B
Traverse
(b, a)
A
B
Expand
Into
(a, b)
C
A
Traverse
(a, c)

Thank you

Roi Lipman
roi@redislabs.com