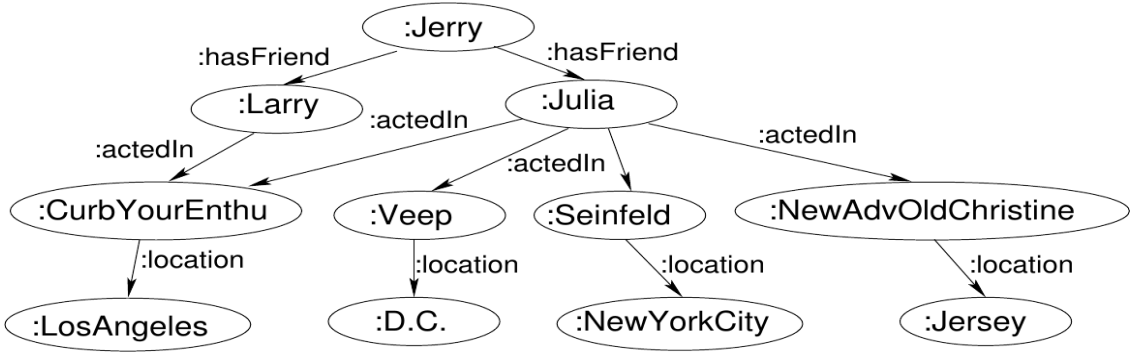
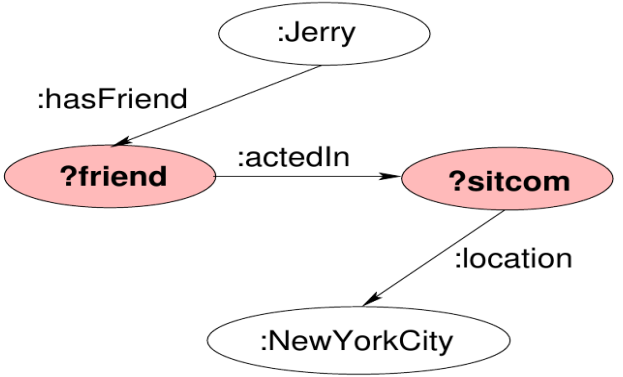
CS698F Advanced Data Management

Instructor: Medha Atre

A graph pattern query





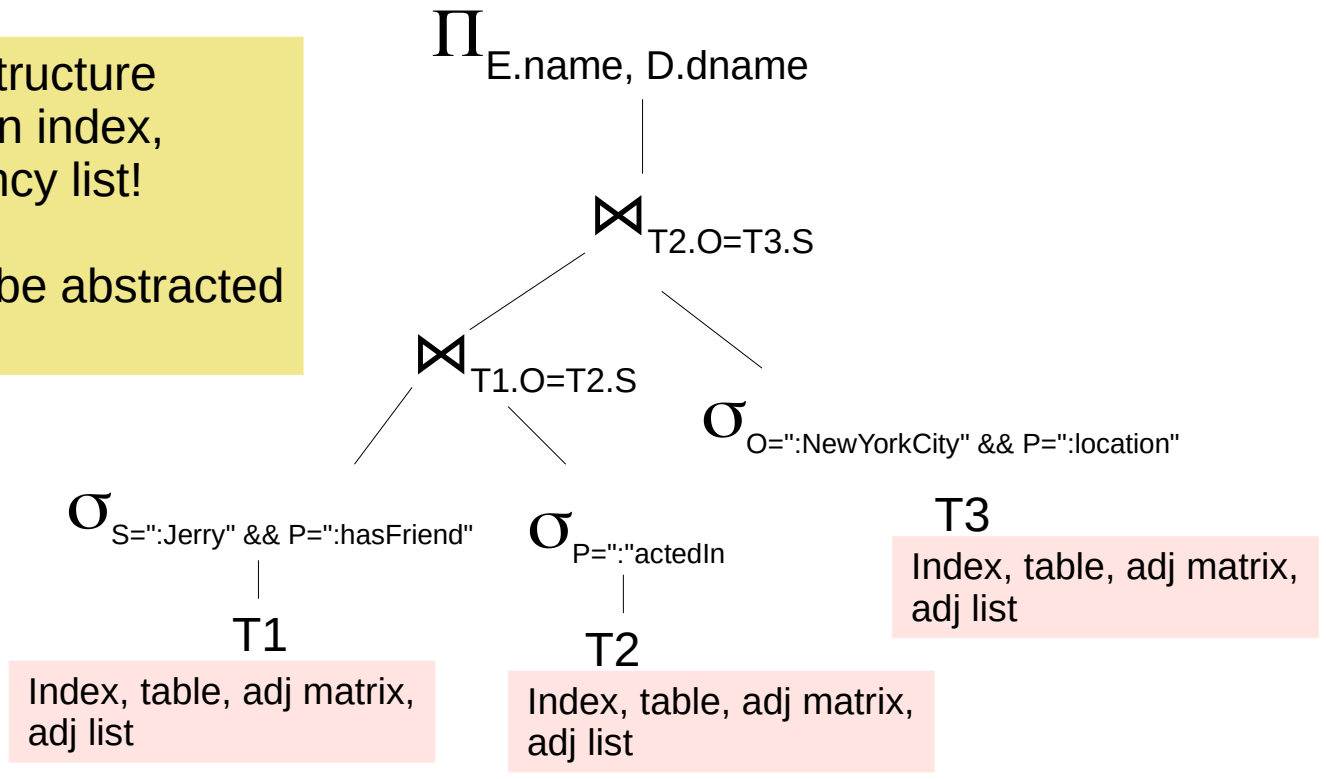
SPARQL BGP

```
SELECT ?friend ?sitcom
WHERE {
      :Jerry :hasFriend  ?friend .
      ?friend :actedIn ?sitcom .
      ?sitcom :location :NewYorkCity .
}
```

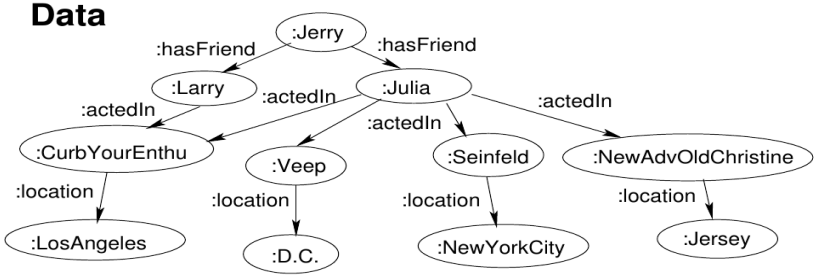
Abstraction of join queries

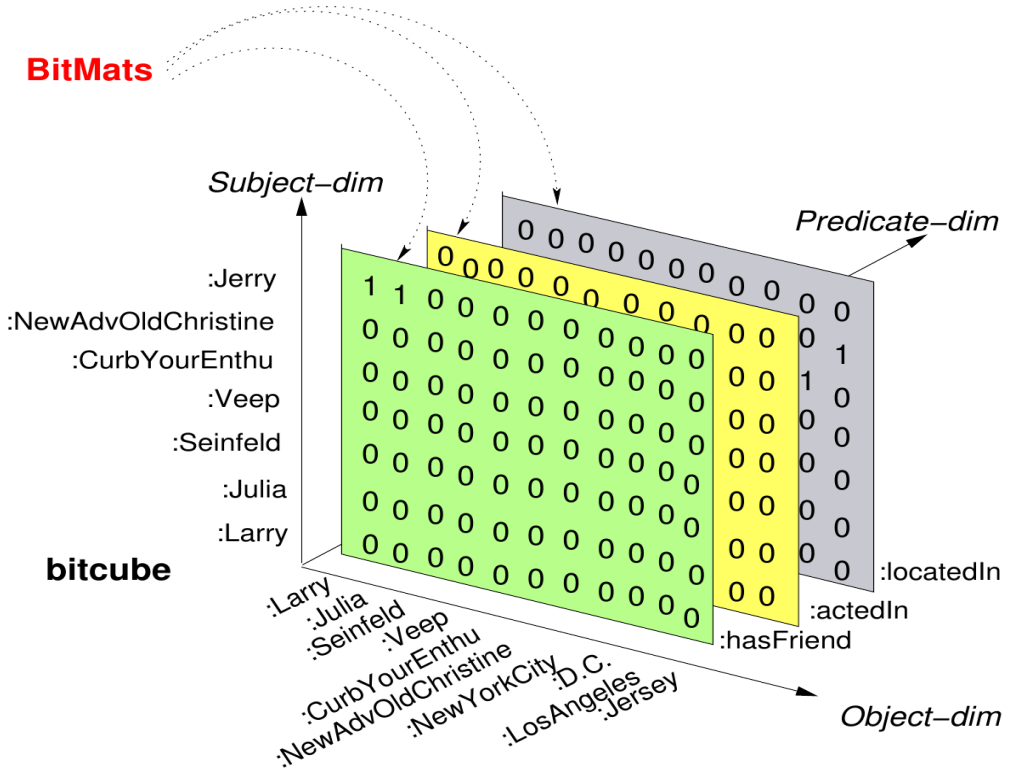
The access data structure can be anything, an index, a table, an adjacency list!

Join methods can be abstracted out accordingly.



Indexing adjacency matrices





Joins with adjacency matrix

- Each matrix is like a 2-column table
  - So serialize the matrix as a table and join – no benefit of 2D matrix!
- Create a *row-vector* and *column-vector*
  - Column-vect = Boolean OR of all the rows
  - Row-vect = Boolean OR of all the columns
- Joining two matrices is equivalent to
  - *Intersection* of row/column vectors of two matrices
  - Removing matrix entries that have the values eliminated in the intersection!
  - This is called a *semi-join*!

Joins with adjacency matrices

	:Larry	:Julia									
:Jerry	1	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	0	0	0	0	0	0	0	0	0
:Larry	0	0	0	0	0	0	0	0	0	0	0

:hasFriend

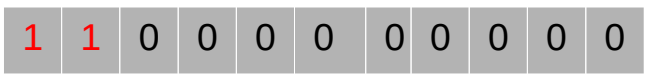
We want to join columns of left with rows of right

	:Seinfeld	:Veep	:CurbUourEnthu								
:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	1	1	1	1	0	0	0	0	0
:Larry	0	0	0	0	1	0	0	0	0	0	0

:actedIn

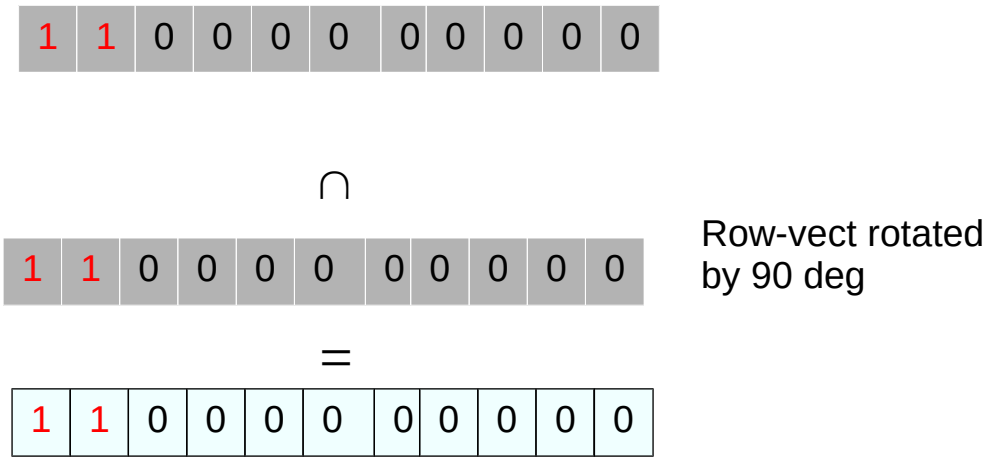
Joins with adjacency matrices

- Column values on LHS join with row values on RHS
  - *col-vect (mat1) AND row-vect(mat2) = partial-join-res*
  - For each 0 bit in *partial-join-res*, remove all matrix cells that contain 1 in the respective position



Joins with adjacency matrices

- Column values on LHS join with row values on RHS
  - *col-vect (mat1) AND row-vect(mat2) = partial-join-res*
  - For each 0 bit in *partial-join-res*, remove all matrix cells that contain 1 in the respective position



Nothing changes here, hence no matrix cells removed!

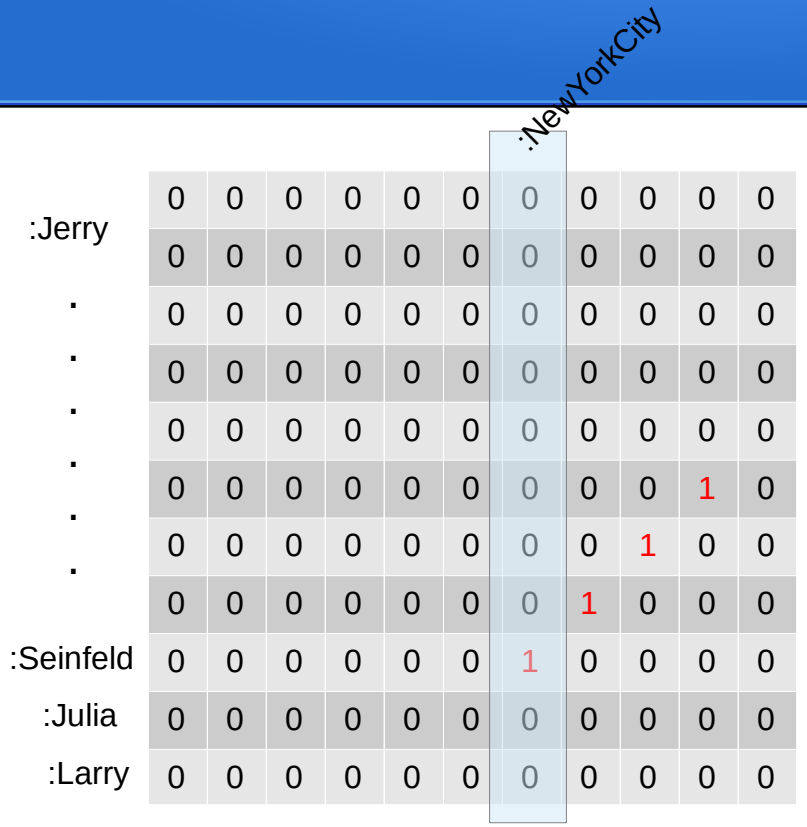
Joins with adjacency matrices

:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	1	0
.	0	0	0	0	0	0	0	0	1	0	0
.	0	0	0	0	0	0	0	1	0	0	0
.	0	0	0	0	0	0	1	0	0	0	0
:Julia	0	0	0	0	0	0	0	0	0	0	0
:Larry	0	0	0	0	0	0	0	0	0	0	0

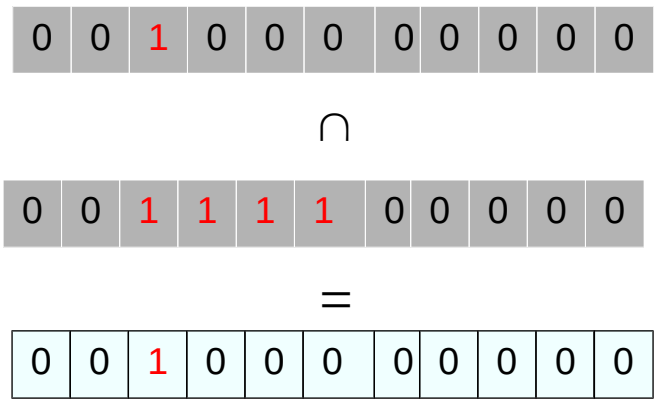
:locatedIn

			:Seinfeld	:Veep	:CurbUourEnthu	:NewAdvOldChri					
:Jerry	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0
:Julia	0	0	1	1	1	1	0	0	0	0	0
:Larry	0	0	0	0	1	0	0	0	0	0	0

:actedIn

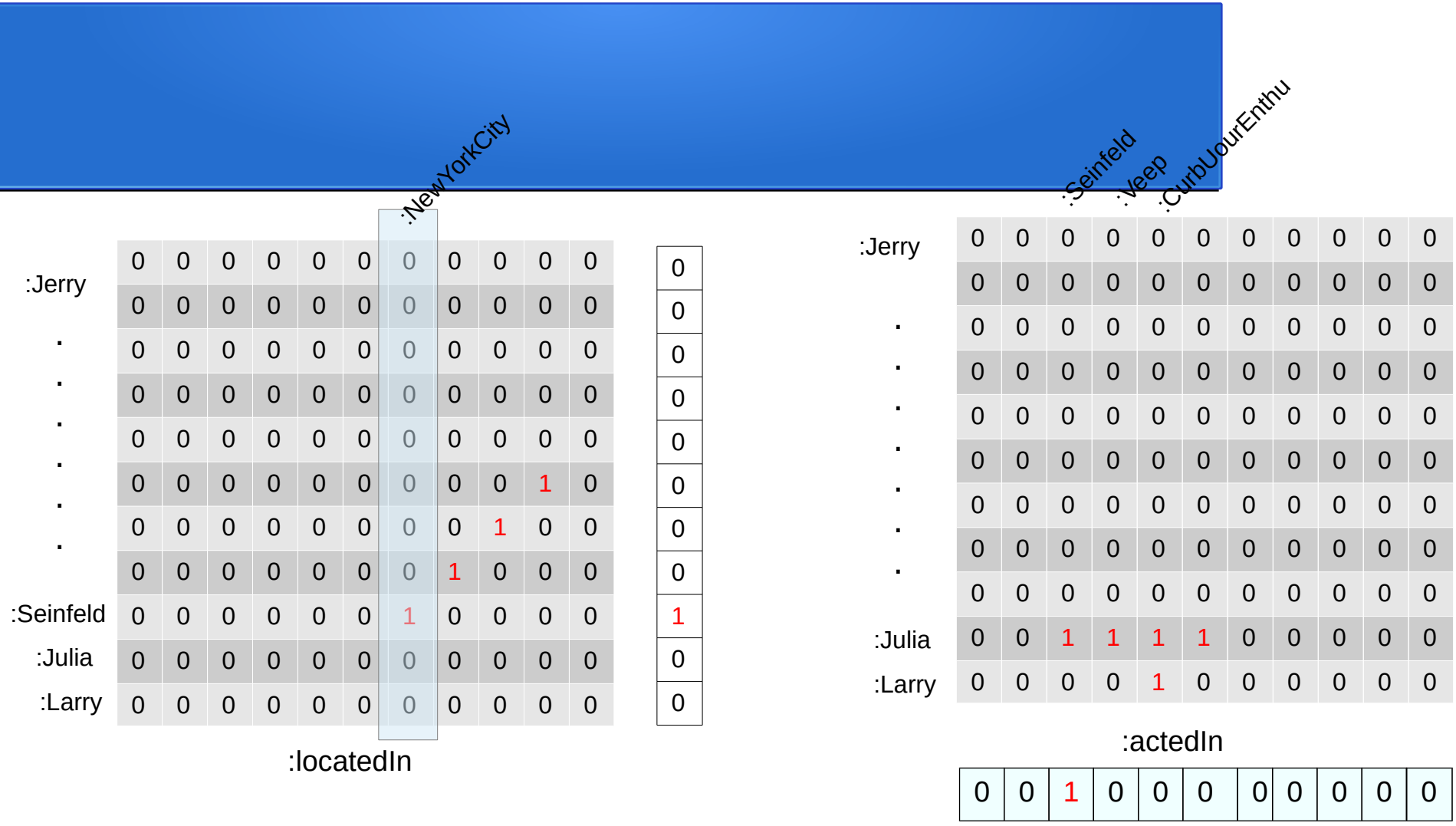

:NewYorkCity
:Jerry
:Seinfeld
:Julia
:Larry
:locatedIn

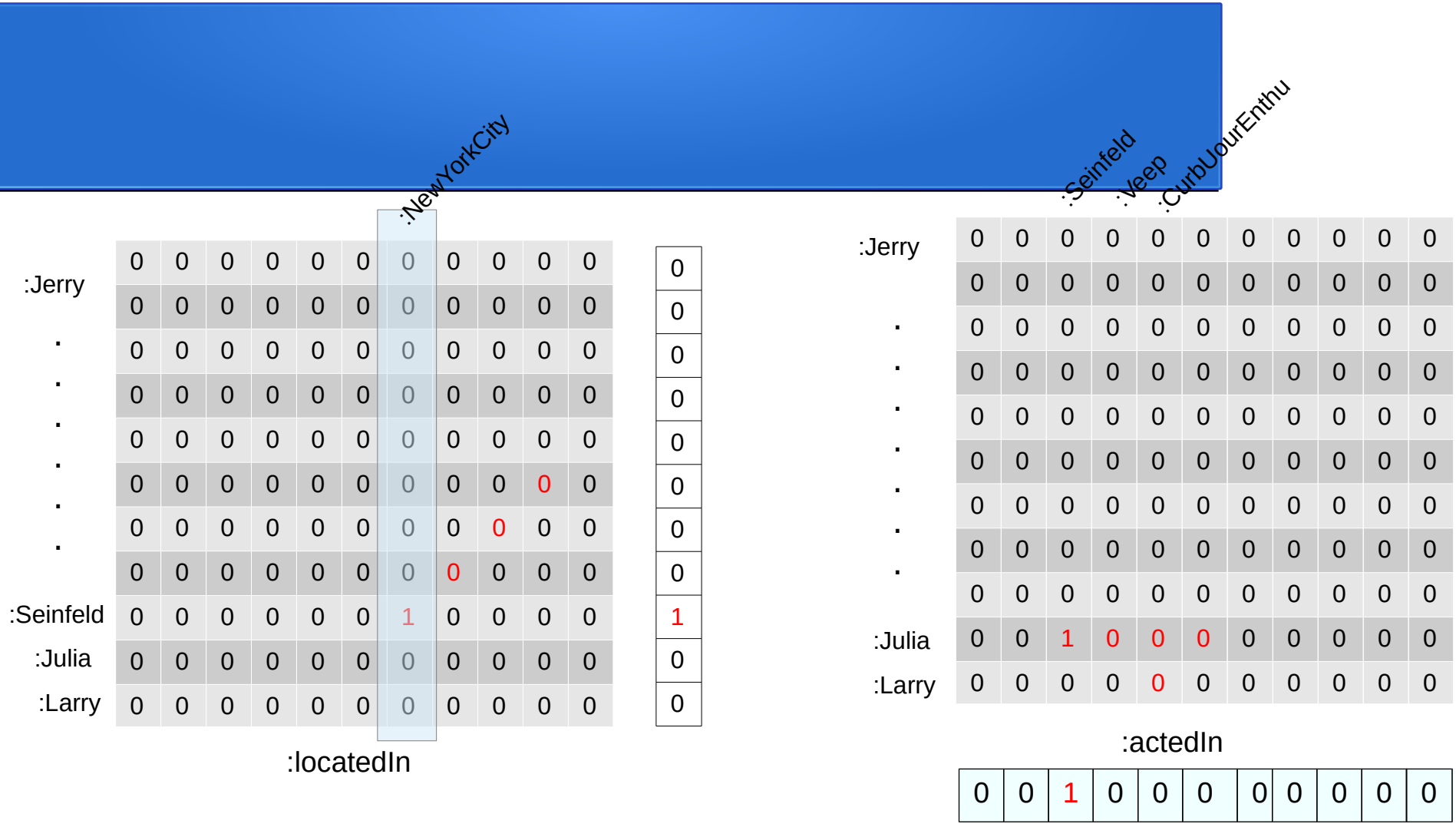

:Seinfeld
:Veep
:CurbUourEnthu
:Jerry
:Julia
:Larry
:actedIn

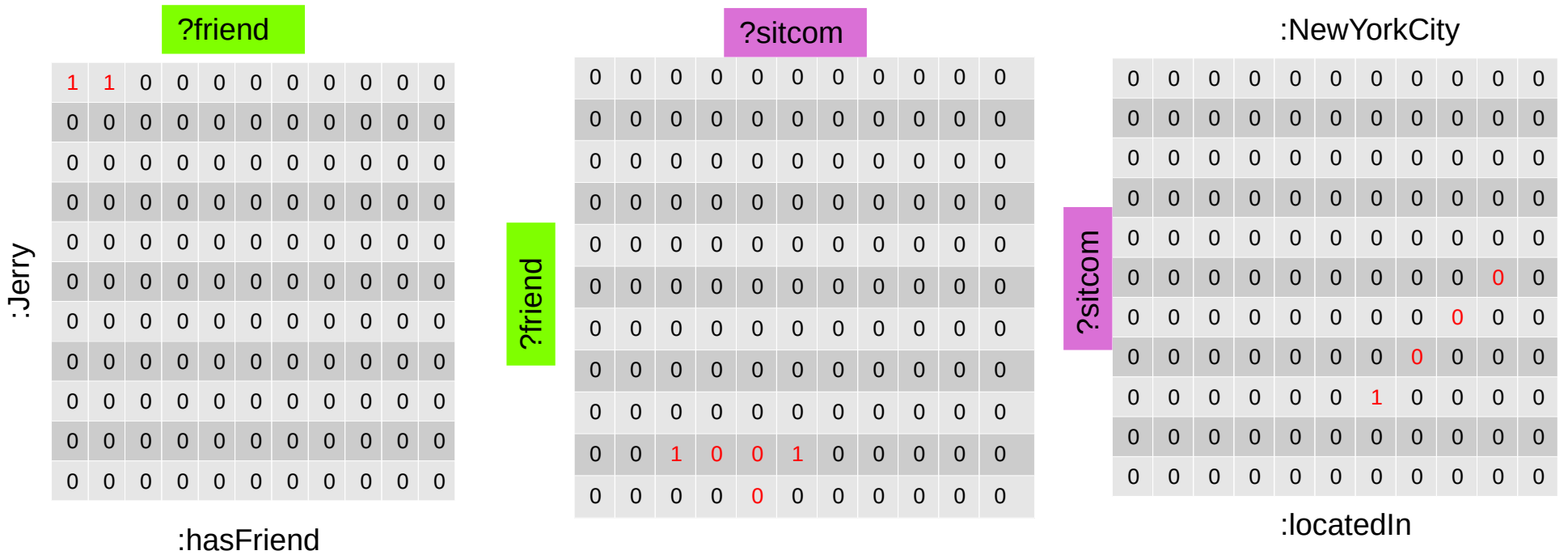


Col-vect rotated
by 90 deg clockwise

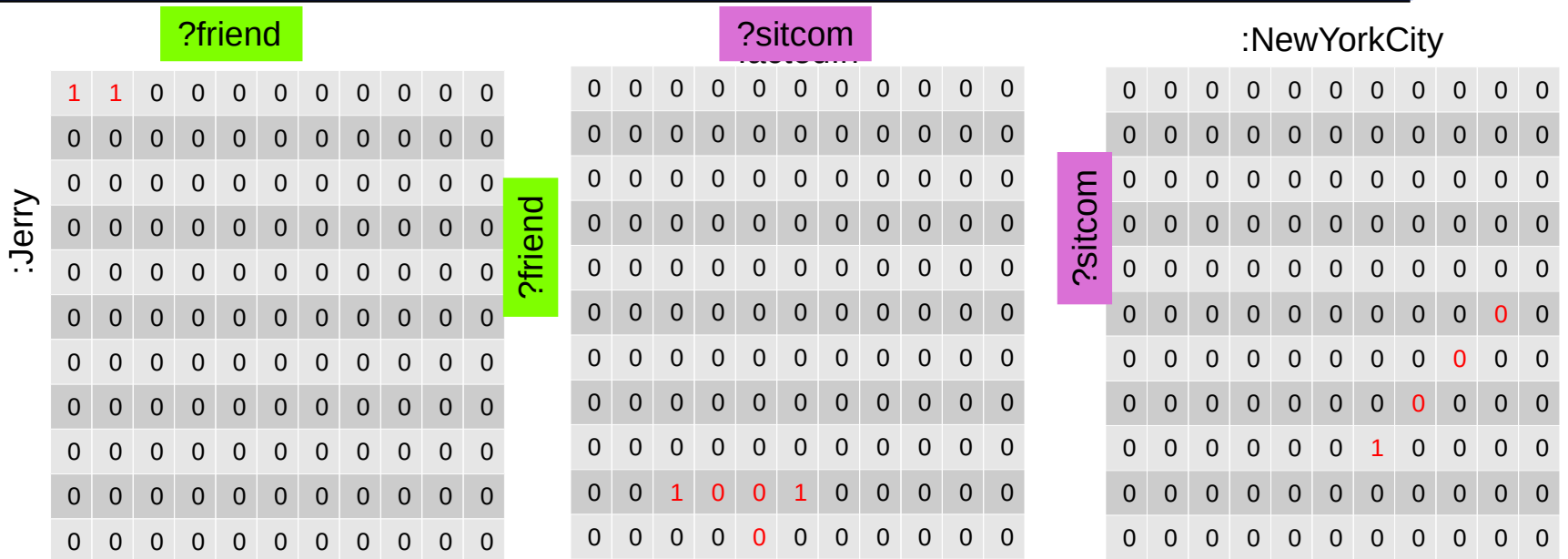
Bits corresponding to :Veep, :CurbYourEnthu, and :NewAdvOldChristine removed, hence remove respective matrix entries from both sides


:NewYorkCity
:Jerry
:Seinfeld
:Julia
:Larry
:locatedIn
:Seinfeld
:Veep
:CurbUourEnthu
:Jerry
:Julia
:Larry
:actedIn


:NewYorkCity
:Jerry
:Seinfeld
:Julia
:Larry
:locatedIn
:Seinfeld
:Veep
:CurbUourEnthu
:Jerry
:Julia
:Larry
:actedIn


:actedIn
?friend
?sitcom
:NewYorkCity
:Jerry
?friend
?sitcom
:hasFriend
:locatedIn

Multi-way join



Match (11, 1) from first matrix to (1,…) in the second matrix => (11, 1), (nothing) => Because first row in second matrix is empty
So *backtrack,* match (11, 2) from the first matrix to (2,…) in the second => (11, 2), (2, 3)
Now match (2, 3) from second matrix to (3,…) from the third => (3, 7) => All matrices matched, so we have one result
(11, 2), (2, 3) (3, 7) => (:Jerry, Julia), (:Julia, :Seinfeld), (:Seinfeld, :NewYorkCity)

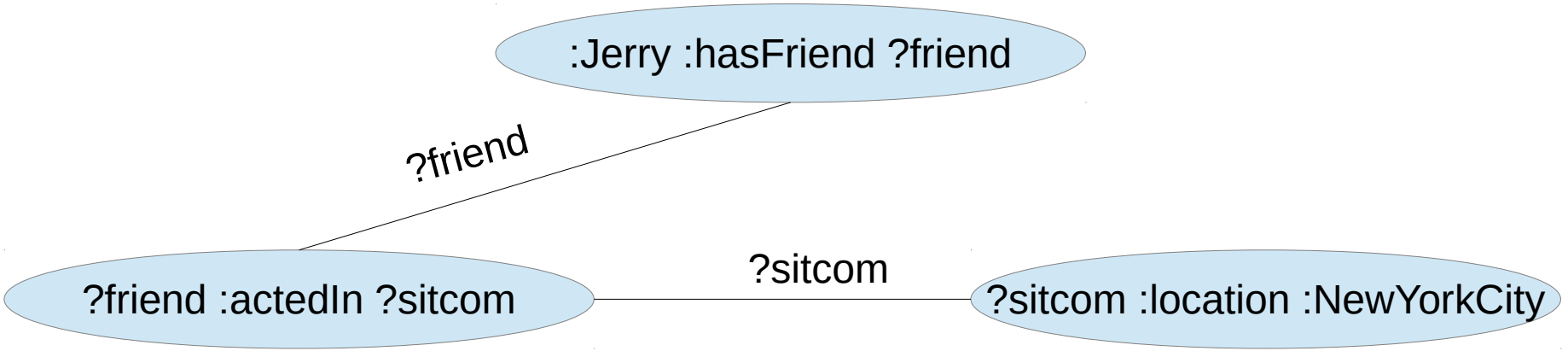
Multi-way join

- What does multi-way join resemble?
  - Nested-loop joins
  - All of which are executed in *pipelined* fashion!
  - First variable bindings from first matrix generated => Pick one tuple from one table
  - Matching bindings from second matrix => join this tuple to the second table
  - Matching bindings from the third table => join the combined tuple of first and second matrix to the third one.

Alternate ways of joining

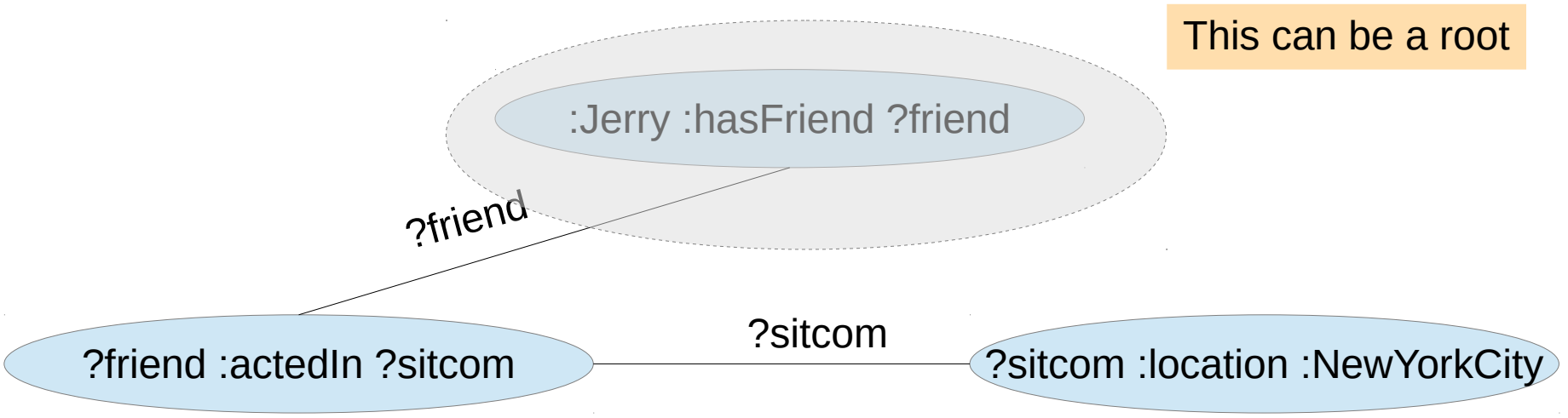
- First prune by taking intersection of bitvectors
  - Semi-joins
- Then join all matrices together – multi-way join
  - Similar to *pipelined* join
- How to decide which matrices to join in which order?
  - Naïve way – try to match each cell in each matrix – $O(N^T)$ => *N* #of tuples/triples, *T* #tables/matrices
  - Or – start with a matrix with least number of 1s, then go to the next smallest one etc.

Graph of Tables

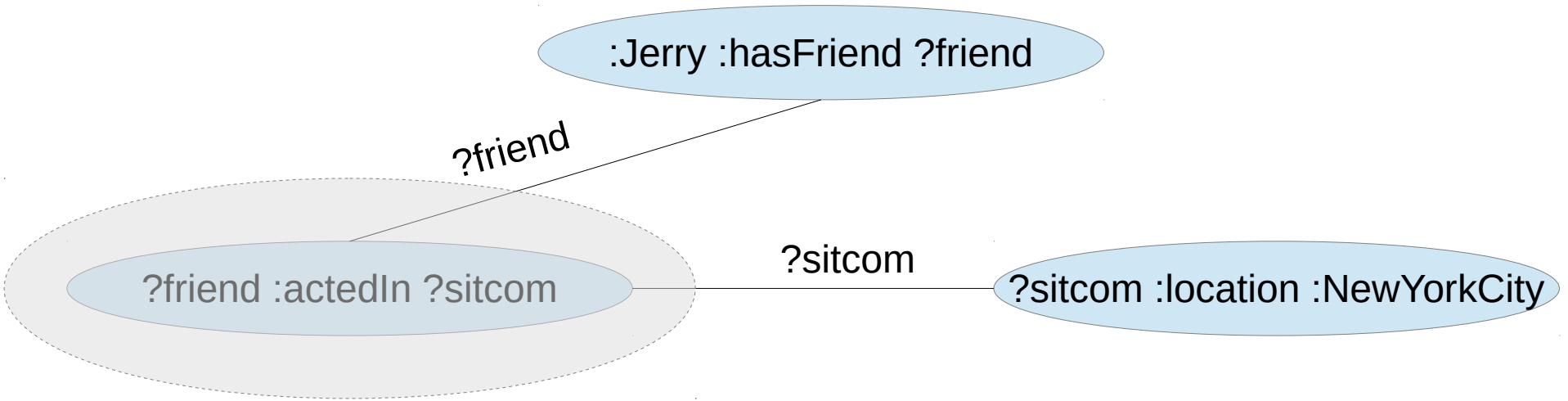


If this graph is *acyclic* construct a rooted spanning tree over it, such that the tables with smaller number of tuples are leaves.
Then start with the leaves and their neighbors and perform semi-joins

Graph of Tables

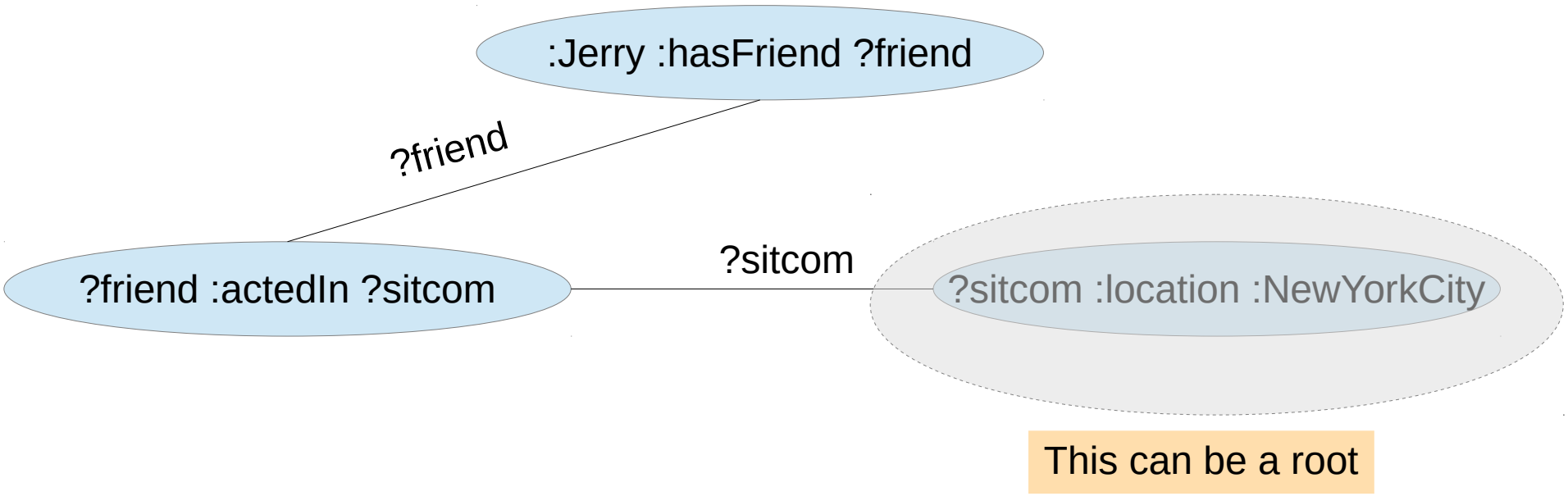


Graph of Tables

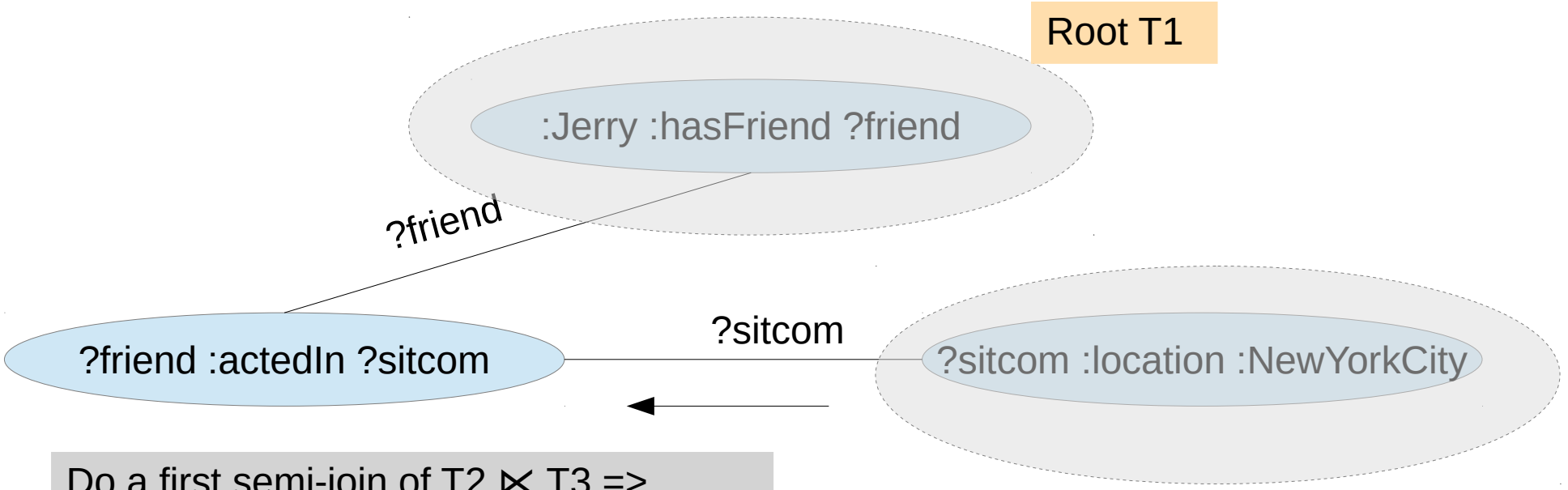


This can be a root

Graph of Tables

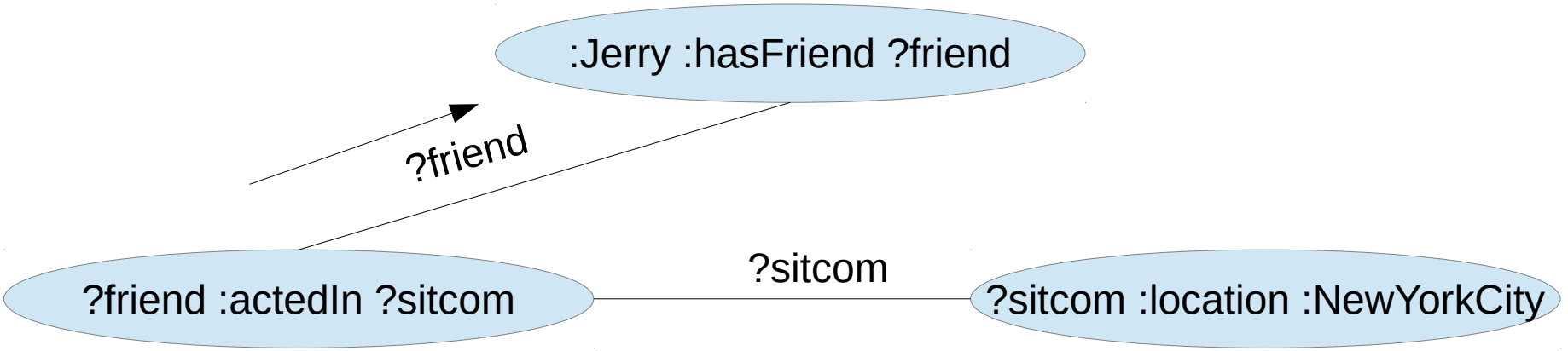


Graph of Tables



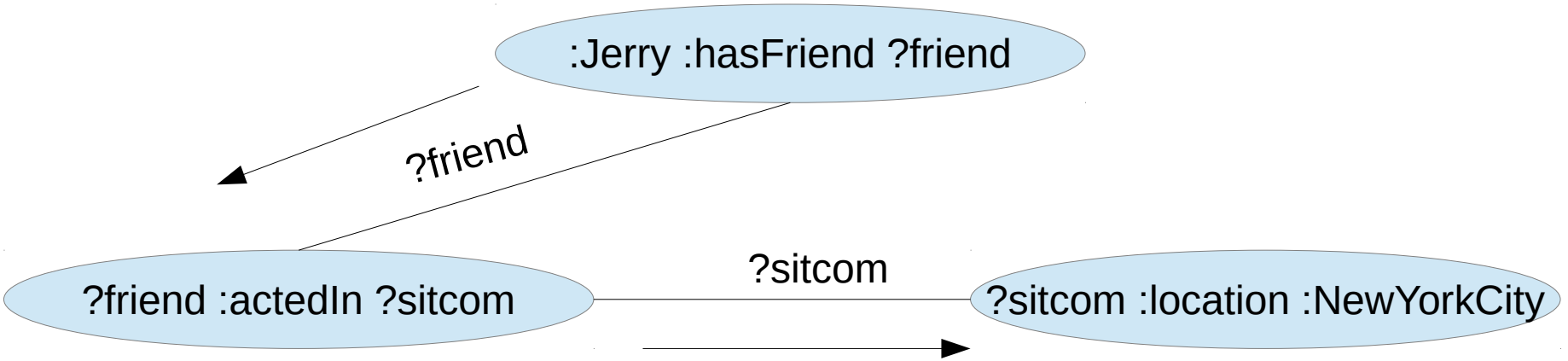
Do a first semi-join of T2 ⋉ T3 =>
Take row-vector of T3 and col-vect of T2
Boolean AND of the two
Unfold the results on T2

Graph of Tables



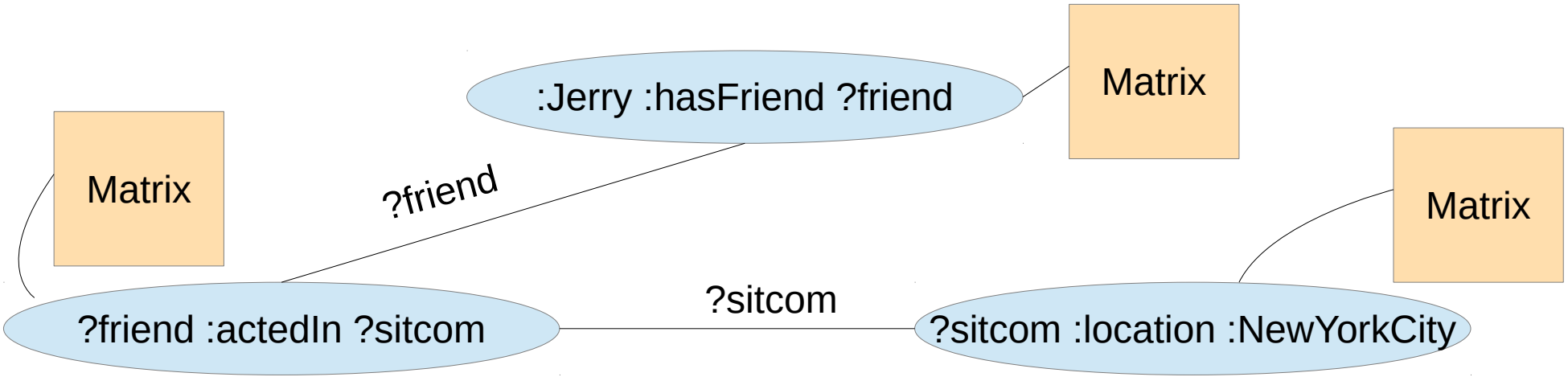
Do the second semi-join of T1 ⋉ T2 =>
Take row-vector of T2 and col-vect of T1
Boolean AND of the two
Unfold the results on T1

Graph of Tables



Do the third semi-join of T2 ⋉ T1 => Take row-vector of T2 and col-vect of T1
Boolean AND of the two, unfold the results on T2, then do the same with T2 and T3

Graph of Tables



Once done with semi-joins, perform multi-way-pipelined join. Starting from any table/matrix, continue recursively matching the cells from its neighbors, output one result when done matching across all matrices.
When matched **all** the cells in **all** the matrices → you have generated all the results

Graph of Tables

- Can the graph of tables be *cyclic*?
- Can there be *redundant* cycles?
- If it is acyclic, does it have any special benefits?
- Why are semi-joins nicer than normal joins?
- How do you do page-wise loading in case of matrices?