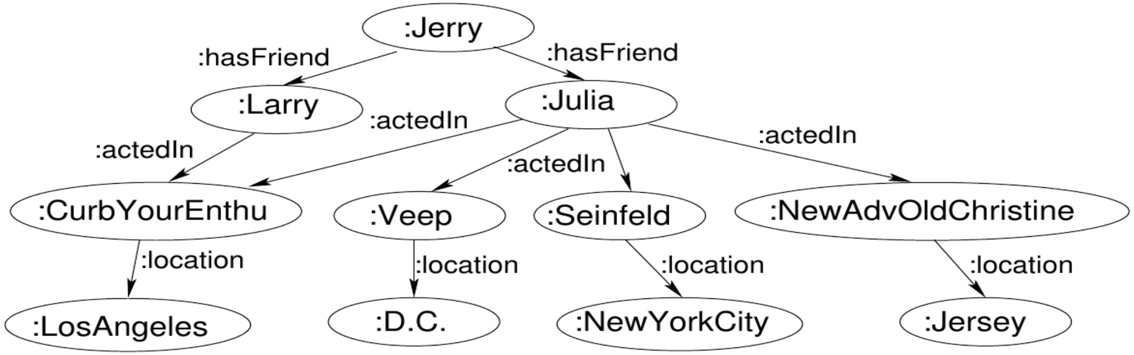
CS698F Advanced Data Management

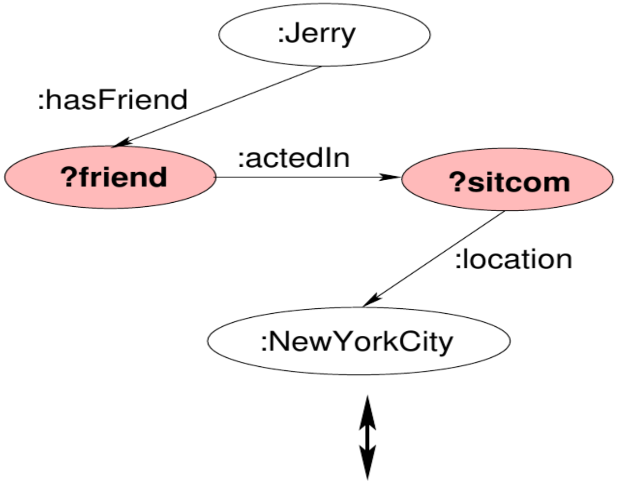
Instructor: Medha Atre

Reminder and Recap

- Reminder – Assignment-1 papers/topics due tonight (23:59) by email.
- Graphs can be represented by tables or adjacency matrices
- Join operator can be abstracted out to make it work with different underlying data structures.

A graph pattern query





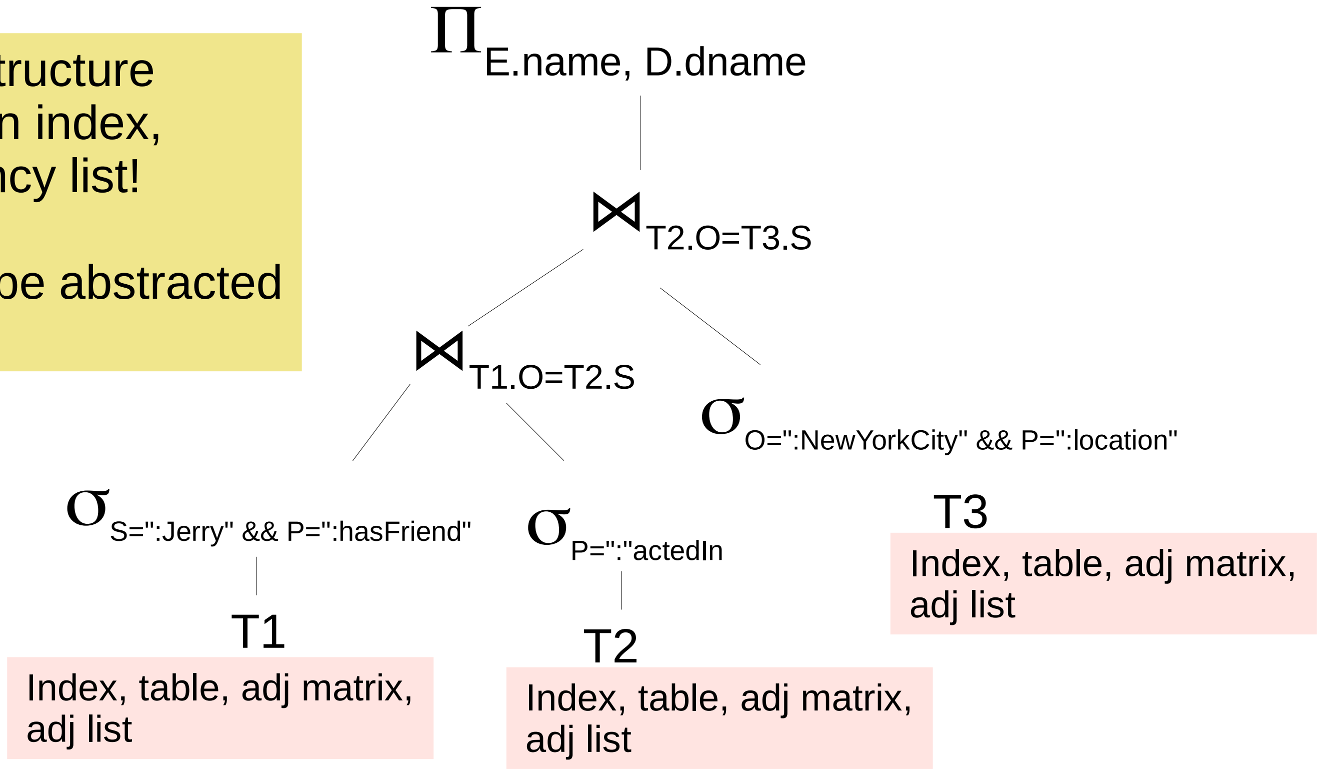
SPARQL BGP

```
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.



$\Pi_{\text{E.name, D.dname}}$

$\bowtie_{\text{T2.O=T3.S}}$

$\bowtie_{\text{T1.O=T2.S}}$

$\sigma_{\text{O=":NewYorkCity" \&\& P=":location"}}$

$\sigma_{\text{S=":Jerry" \&\& P=":hasFriend"}}$

$\sigma_{\text{P=":"actedIn}}$

T3

Index, table, adj matrix, adj list

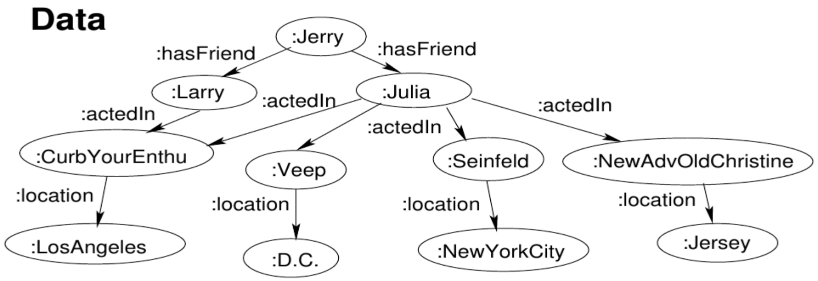
T1

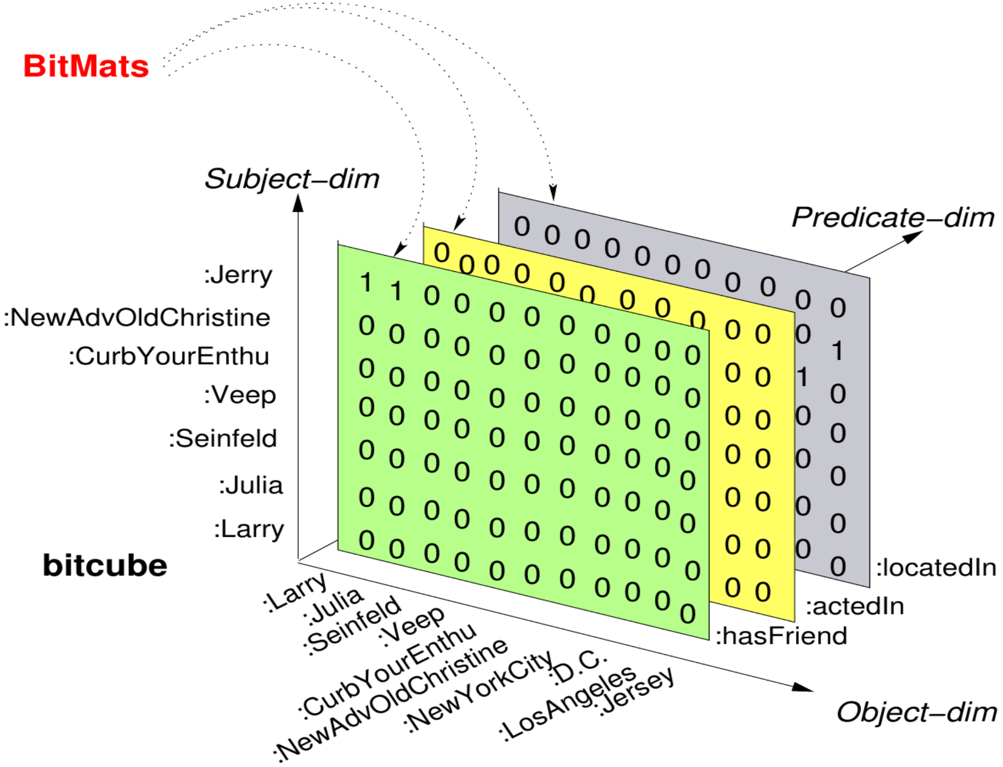
Index, table, adj matrix, adj list

T2

Index, table, adj matrix, adj list

Indexing adjacency matrices





:actedIn

?friend

: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
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

:hasFriend

?sitcom

?friend

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
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	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

:NewYorkCity

?sitcom

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	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	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

:locatedIn

Multi-way join

?friend

: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
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

?sitcom

?friend

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
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	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

:NewYorkCity

?sitcom

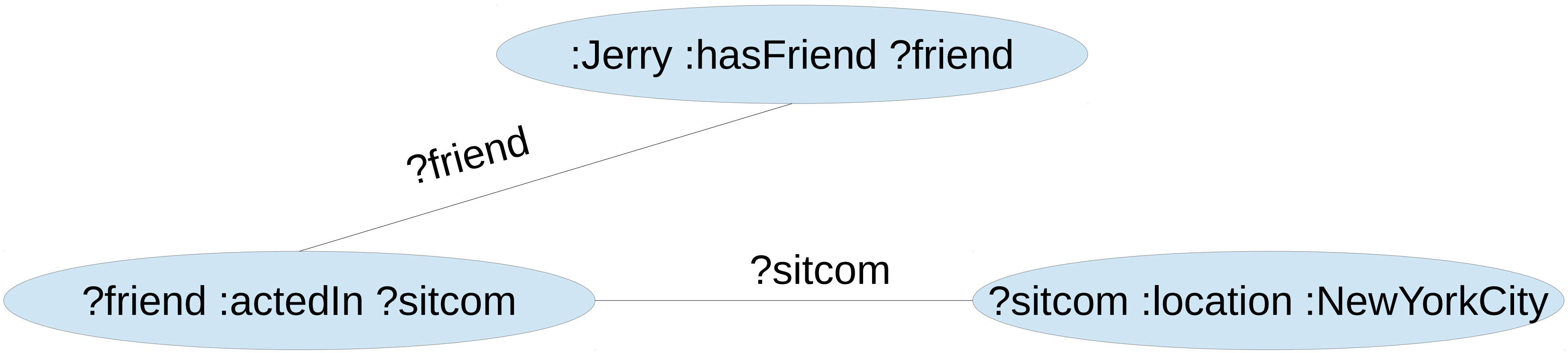
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

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

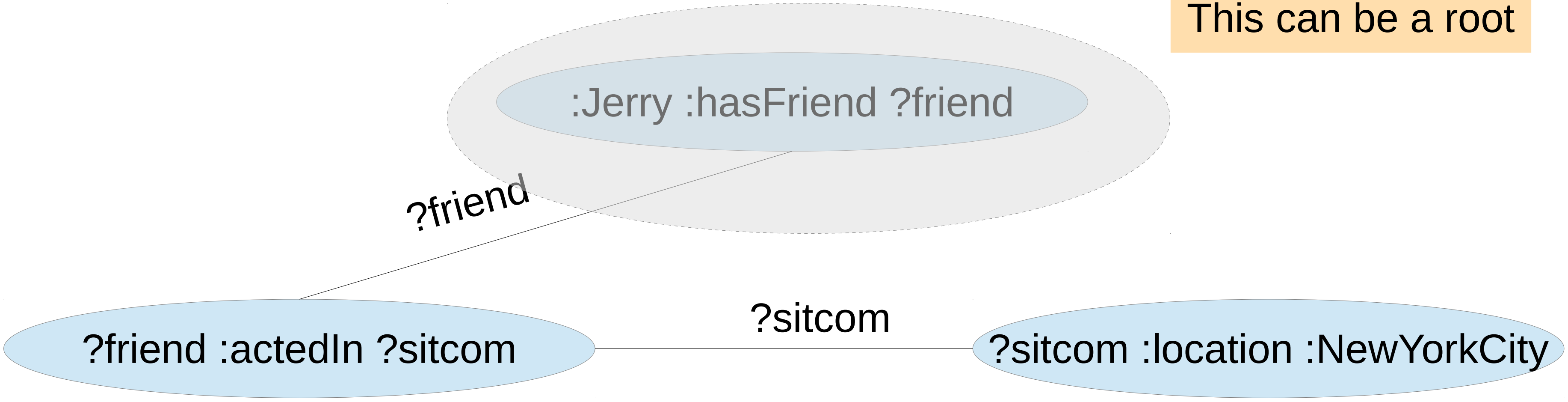
- Similar to nested-loop joins
- All of which are executed in *pipelined* fashion!
- Assumes that all the data is in memory?
  - Can you make some exceptions to this requirement?
- 2D matrix is like an index
  - Since we do semi-joins, it remains in tact despite semi-joins.
  - Does not happen so with joins.

Graph of Tables



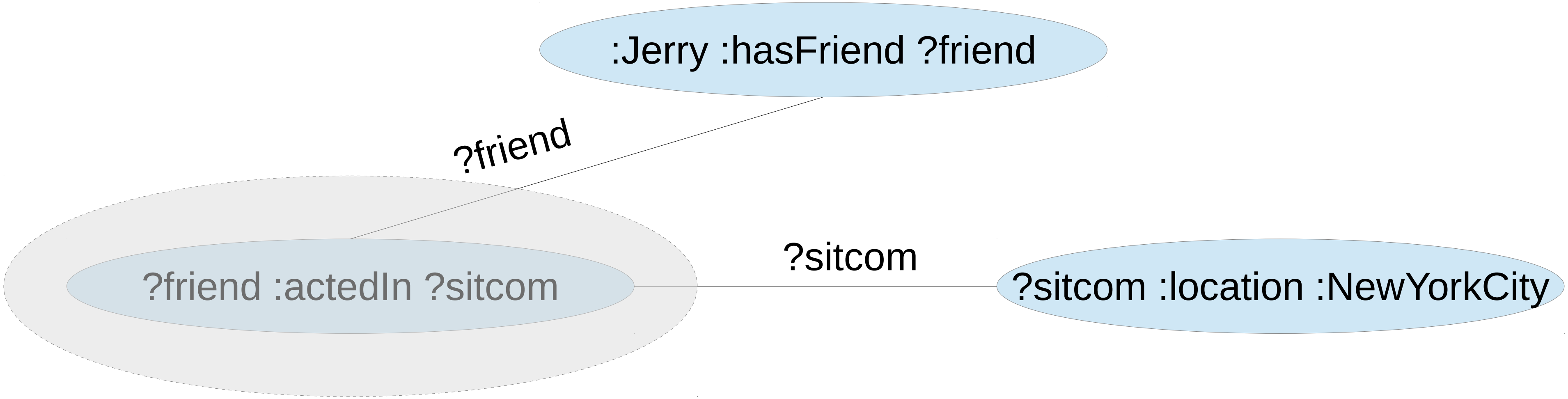
If this graph is *acyclic* construct a rooted spanning tree over it, such that the tables with smaller number of tuples are leaves.
Then start with the leaves and their neighbors and perform semi-joins

Graph of Tables



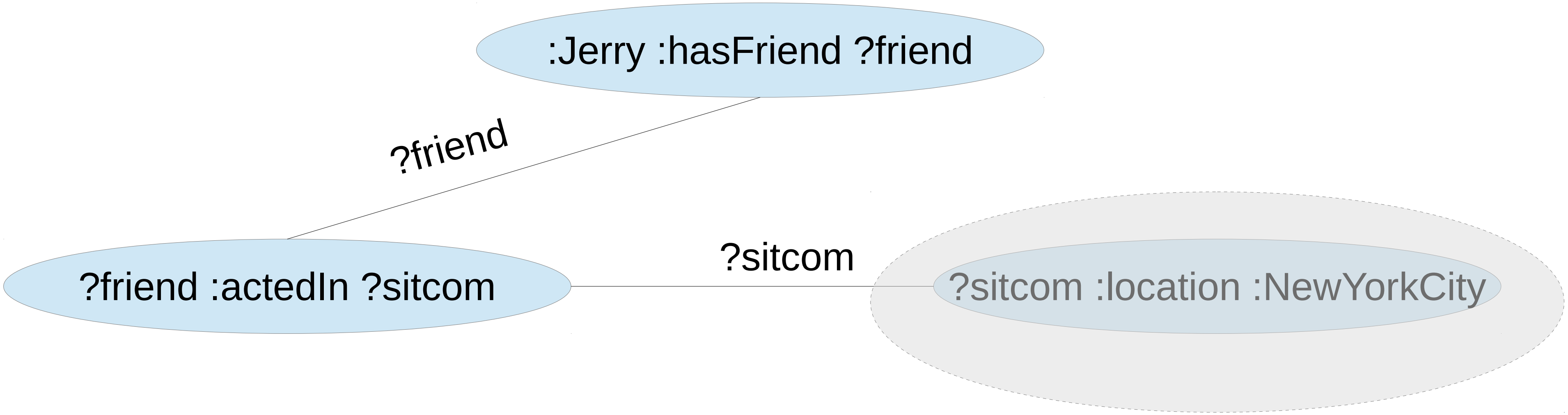
This can be a root

Graph of Tables



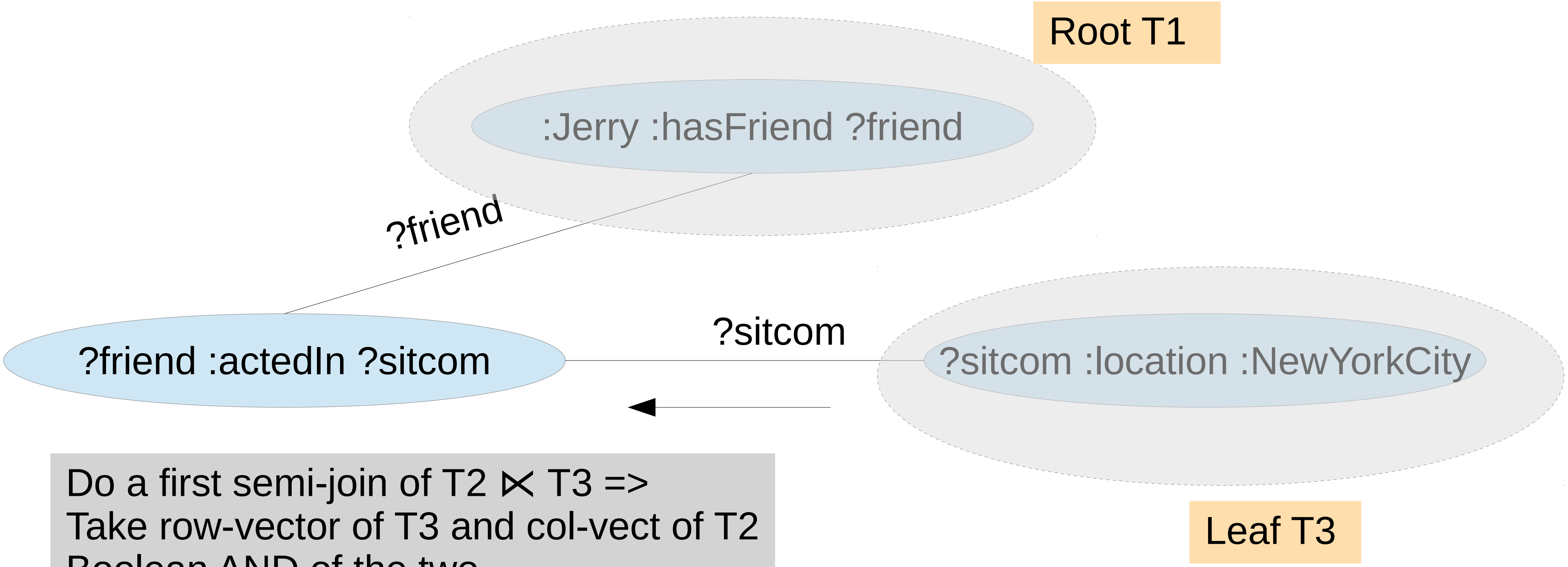
This can be a root

Graph of Tables



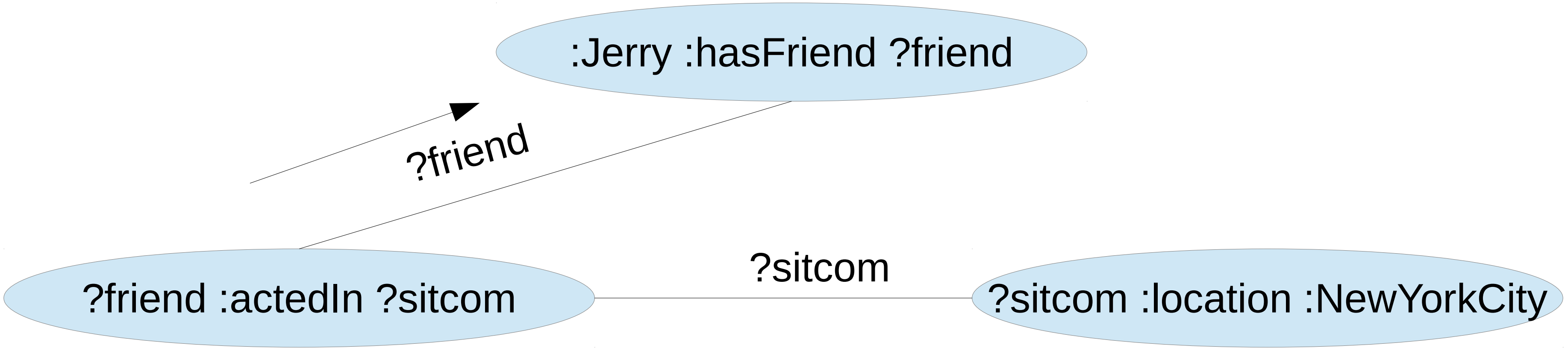
This can be a root

Graph of Tables



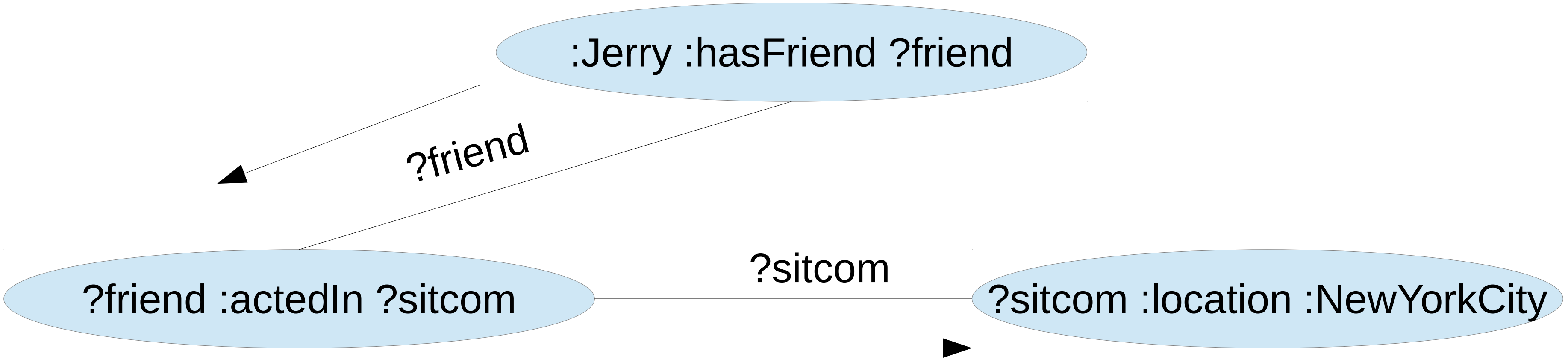
Do a first semi-join of T2 ⋉ T3 =>
Take row-vector of T3 and col-vect of T2
Boolean AND of the two
Unfold the results on T2

Graph of Tables



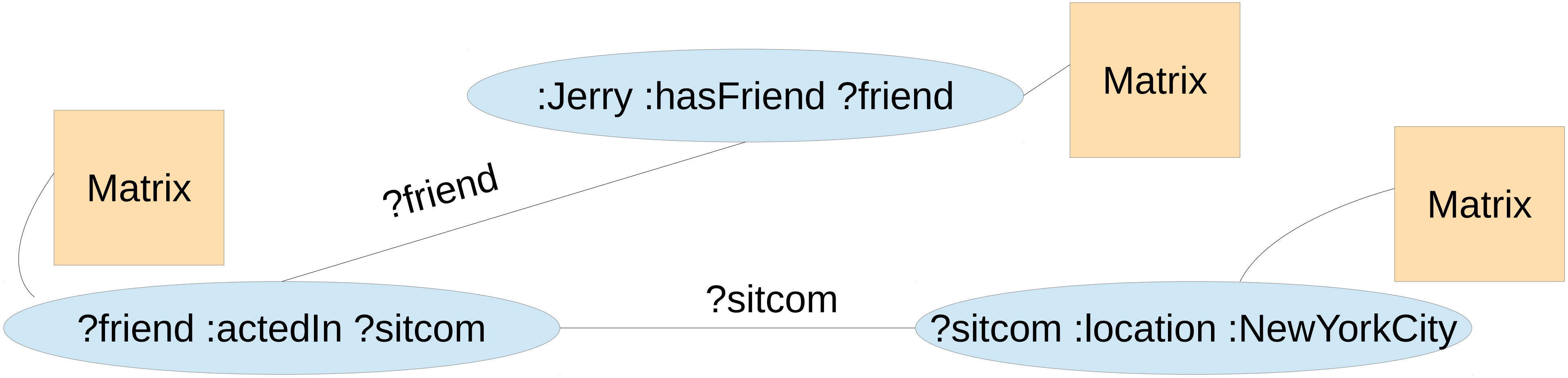
Do the second semi-join of T1 ⋉ T2 =>
Take row-vector of T2 and col-vect of T1
Boolean AND of the two
Unfold the results on T1

Graph of Tables



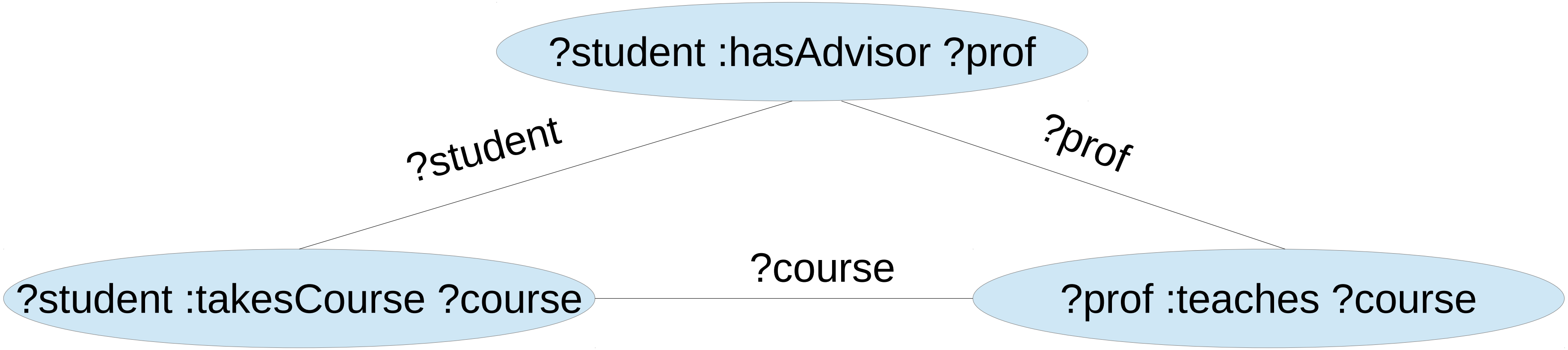
Do the third semi-join of T2 ⋉ T1 => Take row-vector of T2 and col-vect of T1 Boolean AND of the two, unfold the results on T2, then do the same with T2 and T3

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

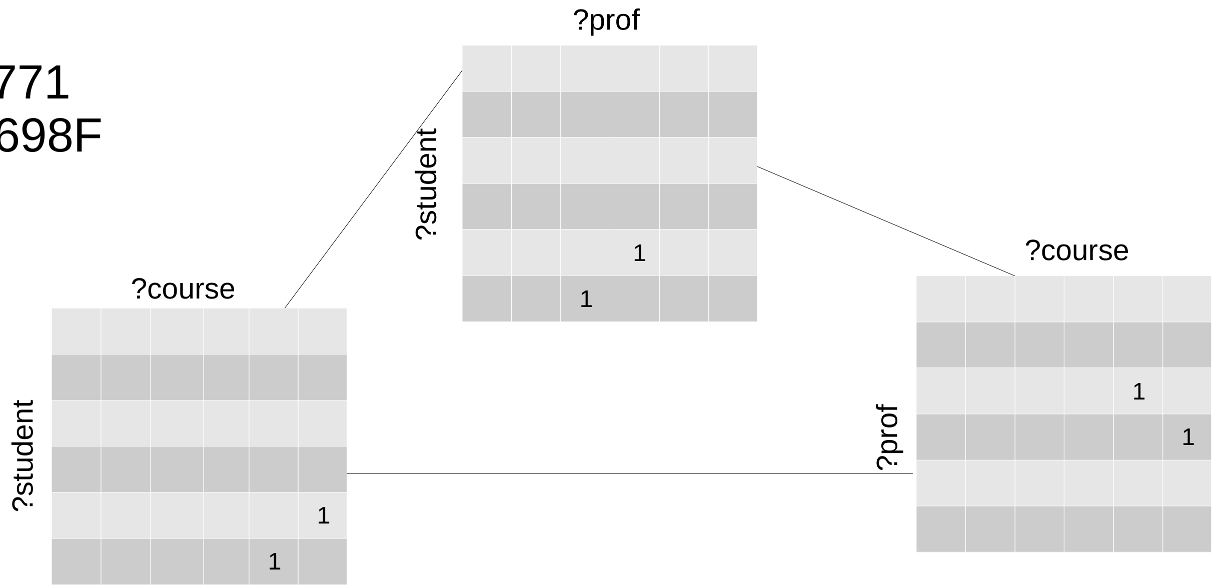
Cyclic graph of tables



Cyclic graph of tables

:Rajesh :hasAdvisor :Atre
:Suresh :hasAdvisor :Ganguly
:Atre :teaches :CS698F
:Ganguly :teaches :CS771
:Rajesh :takesCourse :CS771
:Suresh :takesCourse :CS698F

:Rajesh → 1, :Suresh → 2, :Atre → 3,
:Ganguly → 4, :CS771 → 5, :CS698F → 6

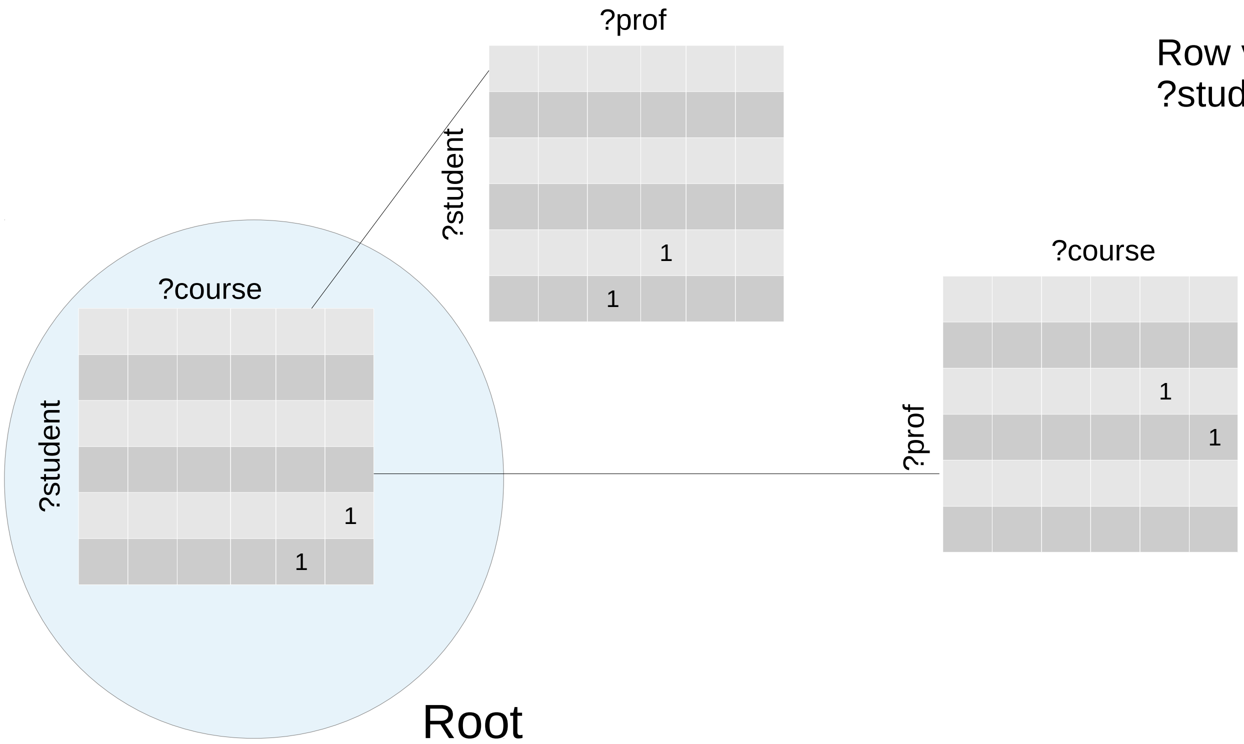


			1		
		1			

					1
				1	

				1	
					1

Cyclic graph of tables



Row vecs for ?student

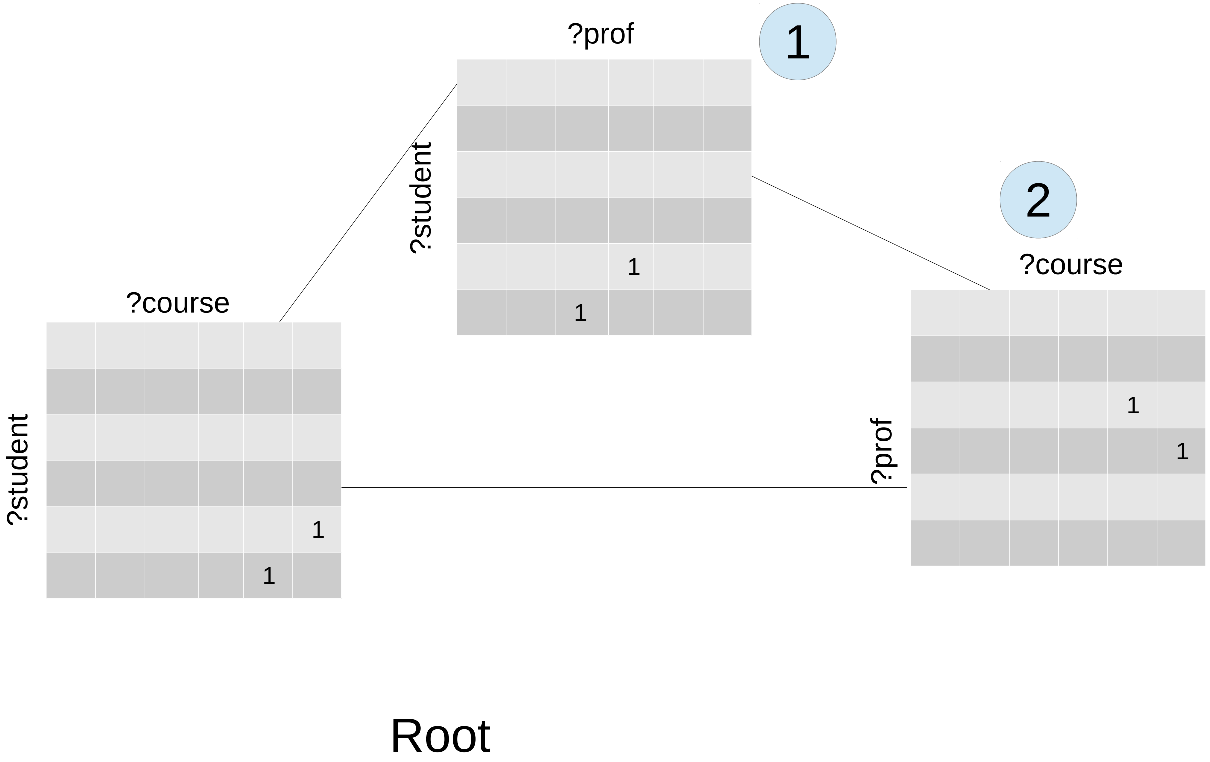
1	1				
1	1				

Col vecs for ?course

				1	1
				1	1

Nothing changes!

Multi-way-join cyclic queries



			1		
		1			

					1
				1	

				1	
					1

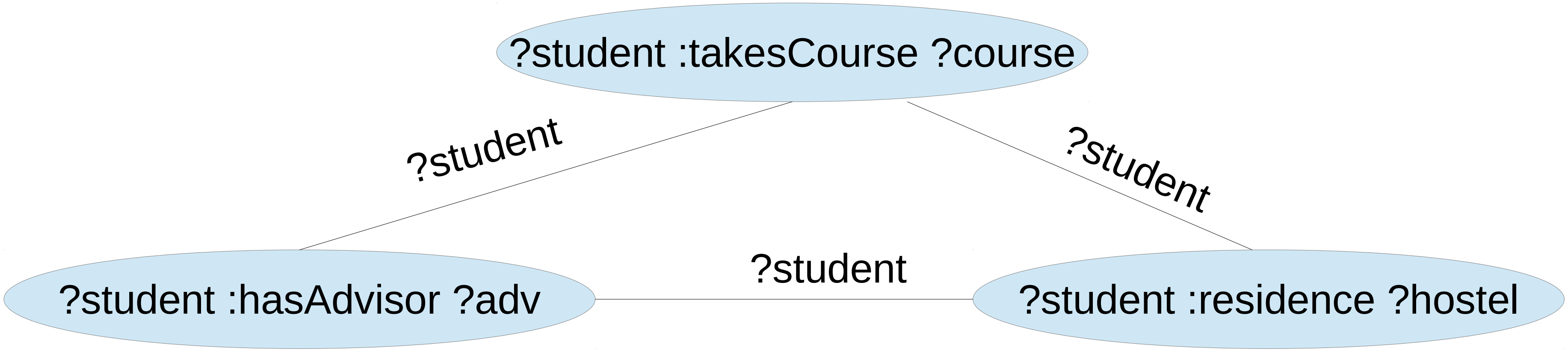
(1, 3) match (3,…) → (1, 3), (3, 6)
Match (3, 6) to (…., 6)
(1, 3), (3, 6), (2, 6)

WAIT!
Mismatch in (1, 3) (2, 6)
Discard the match, and backtrack.

3rd row in mat-2 has only 1 bit, so again backtrack.

(2, 4) match (4,…) → (2, 4), (4, 5)
Match (4, 5) to (…, 5)
(2, 4) (4, 5) (1, 5) mismatch!

Redundant cycles



?student :takesCourse ?course

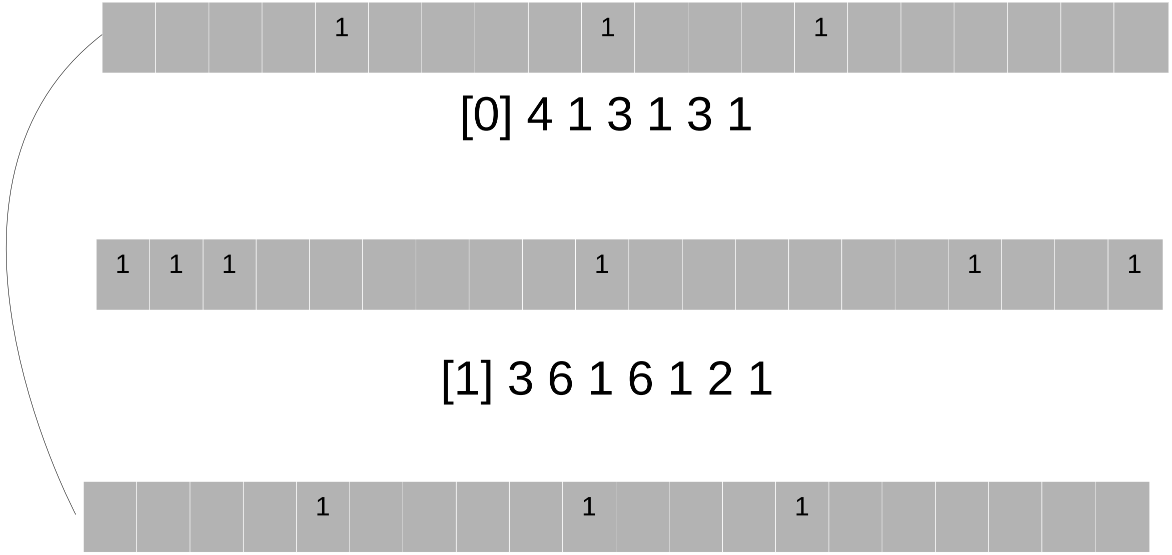
?student :hasAdvisor ?adv

?student :residence ?hostel

Data compression

- Adjacency matrices are very sparse.
- Few 1 bits and lot of 0 bits.
- Compression techniques
  - Run-length-encoding
  - Byte-aligned Bitmap Code (BBC)
  - Word Aligned Hybrid (WAH)
  - Patitioned Word-Aligned Hybrid (PWAH)
  - Others

Run-length-encoding



[0] 4 1 3 1 3 1

[1] 3 6 1 6 1 2 1

5, 10, 14

Delta-encoding

1234, 1236, 1240, 2000, 2011, 2015……

1234, 2, 4, 760, 11, 4……

Only very first integer requires 4 bytes. The following integers can be stored using 2 bytes.

Used in B+ tree clustered indexes

Can you use it in unclustered indexes?

Can you use it in hash-indexes?

Handling compressed data

- How to do Boolean AND/OR on compressed bitvector?
  - Without uncompressing, go on reading run-lengths
  - e.g. [0] 3 1 3 AND [1] 1 3 1 => [0] 3… slide the window
  - [1] 1 3... AND [0] 1 1 => [0] 1 add to the prev => [0] 4… so on
  - For very sparse vs dense vector, go over set bits in sparse vector and check respective set bits in dense one (AND)
  - OR on dense vectors expensive
- How to do a join on delta-encoded index?