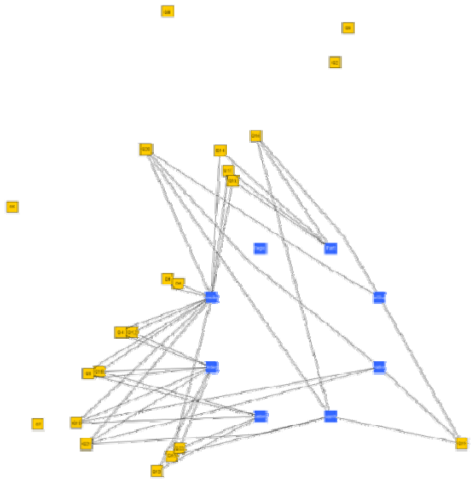
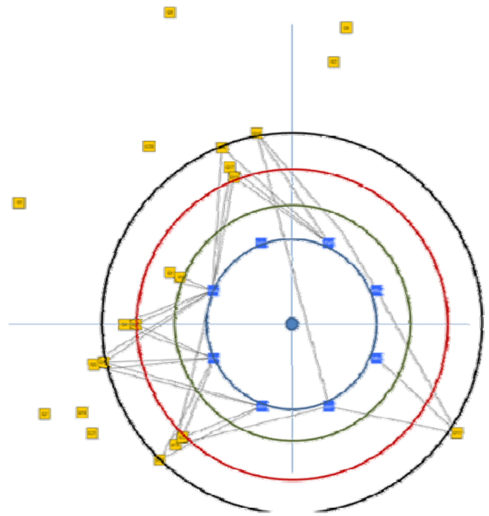
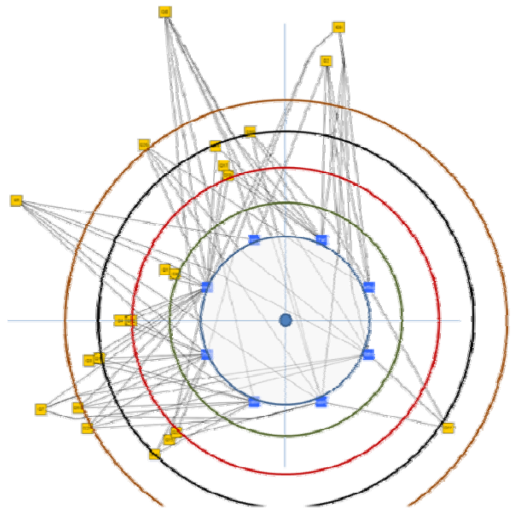




RADAR: Radial Applications' Depiction Around Relations For Data-Centric Ecosystems

Panos Vassiliadis

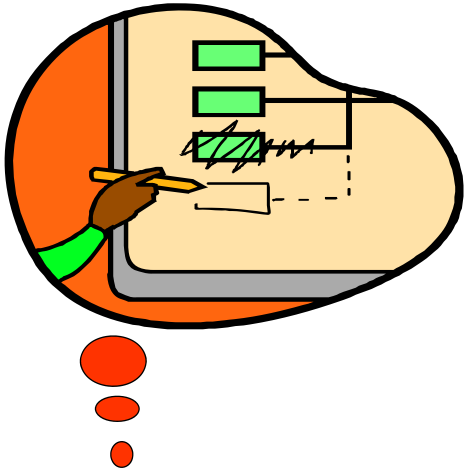
Univ. of Ioannina

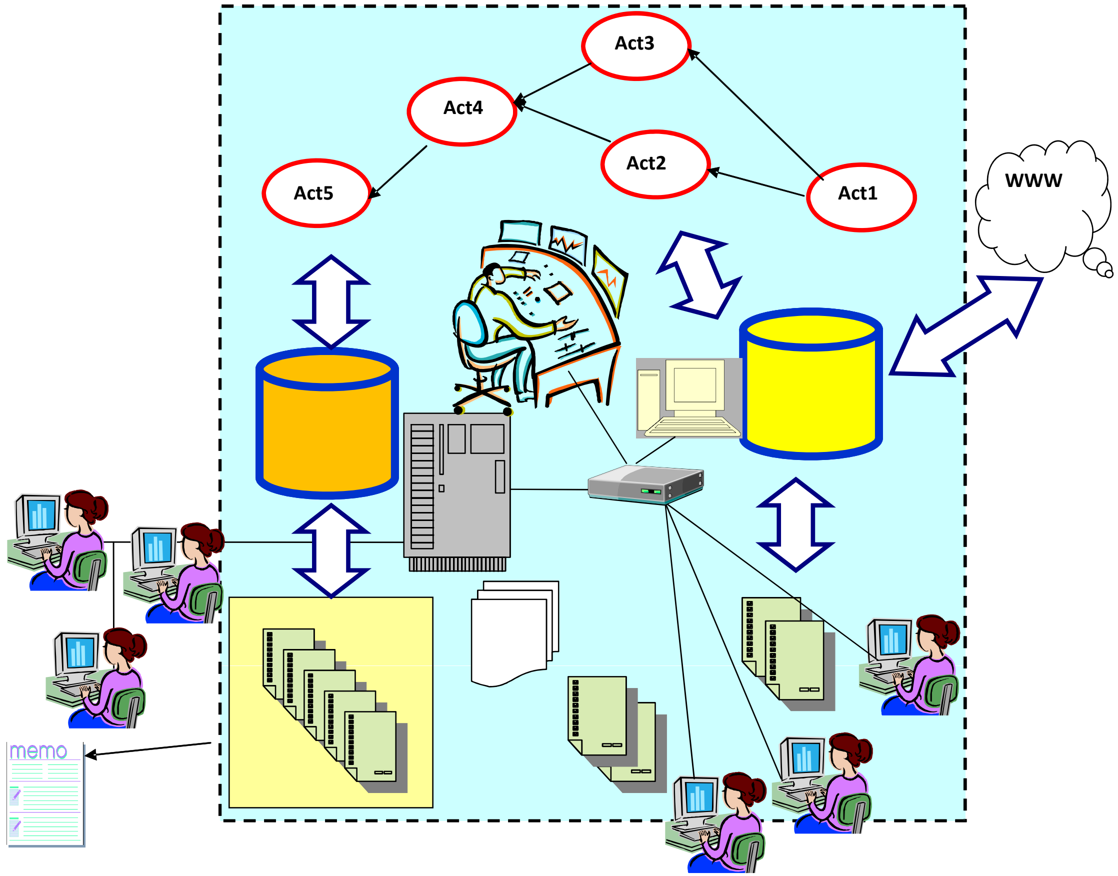


Roadmap

- Motivation & Related Work
- Problem Definition & Reference example
- Barycenter Methods
- Radial Methods
- Conclusions & Thoughts for the future

Data centric ecosystems & the need for a map





Main drive

We would like the administrators of the database, as well as the developers of all the software modules of the ecosystem to be able to have a visual overview of the ecosystem and to be able to understand the interdependencies between its parts. In simple words, the question can be expressed as: "how do we visualize a data-centric ecosystem?"

Requirements and Challenges

- Clear separation of relations and queries
- Spatial memory
  - placing graph’s nodes that have semantic relationships closely over the diagram
- Simple representation

PROBLEM DEFINITION

Graph model

- The ecosystem is a bipartite graph G(V,E)
  - Nodes: relations and queries
  - Edges: query q uses a relation r in any way

- Simplest possible model: we only care for the usage of a relation by a query

- For the future
  - Query semantics
  - Views & constraints

Problem Definition

- We need to lay the bipartite graph of the ecosystem on a 2D surface (the screen) such that
  - Similar nodes are placed nearby (spatial memory)
    - We need to measure similarity
  - Dependency of queries over relations is easily traceable
    - Place queries as close to their defining relations
  - Clearly separate queries and relations

"Similar"? => Distance/Similarity function

- Must precompute all possible distances between
  - two queries
  - two relations
- We adopt a simple modeling and use Jaccard as the distance function

N_T	Number of relations of the ecosystem
N_Q	Number of queries of the ecosystem
relations(q)	the set of relations that are utilized by query q
queries(r)	the set of queries that access relation r

$$JaccQ[q_i][q_j] = \frac{|relations(q_i) \cap relations(q_j)|}{|relations(q_i) \cup relations(q_j)|}$$

$$JaccT[r_i][r_j] = \frac{|queries(r_i) \cap queries(r_j)|}{|queries(r_i) \cup queries(r_j)|}$$

TPC-H as the reference example

	Region	*Nation*	*Supplier*	*Customer*	*Part*	*Partsupp*	*Lineitem*	*Orders*
Q1	0	0	0	0	0	0	1	0
Q2	1	1	1	0	1	1	0	0
Q3	0	0	0	1	0	0	1	1
Q4	0	0	0	0	0	0	1	1
Q5	1	1	1	1	0	0	1	1
Q6	0	0	0	0	0	0	1	0
Q7	0	1	1	1	0	0	1	1
Q8	1	1	1	1	1	0	1	1
Q9	0	1	1	0	1	1	1	1
Q10	0	1	0	1	0	0	1	1
Q11	0	1	1	0	0	1	0	0
Q12	0	0	0	0	0	0	1	1
Q13	0	0	0	1	0	0	0	1
Q14	0	0	0	0	1	0	1	0
Q15	0	0	1	0	0	0	1	0
Q16	0	0	1	0	1	1	0	0
Q17	0	0	0	0	1	0	1	0
Q18	0	0	0	1	0	0	1	1
Q19	0	0	0	0	1	0	1	0
Q20	0	1	1	0	0	1	1	0
Q21	0	1	1	0	0	0	1	1
Q22	0	0	0	1	0	0	0	1

	Region	*Nation*	*Supplier*	*Customer*	*Part*	*PartSupp*	*LineItem*	*Orders*
Region	**1,00**	0,33	0,30	0,22	0,25	0,14	0,11	0,15
Nation	0,33	**1,00**	0,73	0,31	0,23	0,40	0,37	0,40
Supplier	0,30	0,73	**1,00**	0,20	0,31	0,50	0,35	0,29
Customer	0,22	0,31	0,20	**1,00**	0,07	0,00	0,32	0,67
Part	0,25	0,23	0,31	0,07	**1,00**	0,33	0,26	0,12
PartSupp	0,14	0,40	0,50	0,00	0,33	**1,00**	0,10	0,06
LineItem	0,11	0,37	0,35	0,32	0,26	0,10	**1,00**	0,53
Orders	0,15	0,40	0,29	0,67	0,12	0,06	0,53	**1,00**

	Q1	*Q2*	*Q3*	*Q4*	*Q5*	*Q6*	*Q7*	*Q8*	*Q9*	*Q10*	*Q11*	*Q12*	*Q13*	*Q14*	*Q15*	*Q16*	*Q17*	*Q18*	*Q19*	*Q20*	*Q21*	*Q22*
Q1	1,00																					
Q2	0,00	1,00																				
Q3	0,33	0,00	1,00																			
Q4	0,50	0,00	0,67	1,00																		
Q5	0,17	0,38	0,50	0,33	1,00																	
Q6	1,00	0,00	0,33	0,50	0,17	1,00																
Q7	0,20	0,25	0,60	0,40	**0,83**	0,20	1,00															
Q8	0,14	0,50	0,43	0,29	**0,86**	0,14	0,71	1,00														
Q9	0,17	0,57	0,29	0,33	0,50	0,17	0,57	0,63	1,00													
Q10	0,25	0,13	**0,75**	0,50	0,67	0,25	**0,80**	0,57	0,43	1,00												
Q11	0,00	0,60	0,00	0,00	0,29	0,00	0,33	0,25	0,50	0,17	1,00											
Q12	0,50	0,00	0,67	1,00	0,33	0,50	0,40	0,29	0,33	0,50	0,00	1,00										
Q13	0,00	0,00	0,67	0,33	0,33	0,00	0,40	0,29	0,14	0,50	0,00	0,33	1,00									
Q14	0,50	0,17	0,25	0,33	0,14	0,50	0,17	0,29	0,33	0,20	0,00	0,33	0,00	1,00								
Q15	0,50	0,17	0,25	0,33	0,33	0,50	0,40	0,29	0,33	0,20	0,25	0,33	0,00	0,33	1,00							
Q16	0,00	0,60	0,00	0,00	0,13	0,00	0,14	0,25	0,50	0,00	0,50	0,00	0,00	0,25	0,25	1,00						
Q17	0,50	0,17	0,25	0,33	0,14	0,50	0,17	0,29	0,33	0,20	0,00	0,33	0,00	1,00	0,33	0,25	1,00					
Q18	0,33	0,00	1,00	0,67	0,50	0,33	0,60	0,43	0,29	**0,75**	0,00	0,67	0,67	0,25	0,25	0,00	0,25	1,00				
Q19	0,50	0,17	0,25	0,33	0,14	0,50	0,17	0,29	0,33	0,20	0,00	0,33	0,00	1,00	0,33	0,25	1,00	0,25	1,00			
Q20	0,25	0,50	0,17	0,20	0,43	0,25	0,50	0,38	0,67	0,33	**0,75**	0,20	0,00	0,20	0,50	0,40	0,20	0,17	0,20	1,00		
Q21	0,25	0,29	0,40	0,50	0,67	0,25	**0,80**	0,57	0,67	0,60	0,40	0,50	0,20	0,20	0,50	0,17	0,20	0,40	0,20	0,60	1,00	
Q22	0,00	0,00	0,67	0,33	0,33	0,00	0,40	0,29	0,14	0,50	0,00	0,33	1,00	0,00	0,00	0,00	0,00	0,67	0,00	0,00	0,20	1,00

BARYCENTER METHODS

Reference method for deploying bipartite graphs

1. Decide layers
2. Decide position of each node in its layer
   – Such that the number of edge crossings is minimized
3. Assign coordinates in the 2D canvas

Reference method for deploying bipartite graphs

1. Decide layers
2. Decide position of each node in its layer
   - Such that the number of edge crossings is minimized
3. Assign coordinates in the 2D canvas

- In our case, steps 1 and 3 are easy; however step 2 is already NP-complete…

Barycenter methods

- Recursive iterate the following over each layer of the bipartite graph
  - Given two layers L and L’, where L’ has an ordering for its nodes
  - Place each node of L as close as possible to the “middle” of the range of the nodes it accesses in layer L’,
  - “middle” can be
    - the median
    - the average (barycenter) of their positions

In our case

- 2 layers: queries and relations
- First align the relations by order of similarity
- Then align the queries with respect to the placement of tables

- We present only the sandwich extension: split the line of queries in two, surrounding the line of relations; place queries alternatively to the two lines

Align Tables as L1

- Goal: place similar tables as close as possible
  - "similar": hit by as many as possible common queries
  - Why? To reduce the span of these common queries

Align Tables as L1

- While there exist relations not assigned to a position in the output list, the algorithm tries to find the two most similar relations out of the remaining ones, r_i and r_j, and once it finds them, it marks r_i as *maxI* and r_j as *maxJ*.
- Then, it places relation *maxI* to the sorted list's next free slot and adds right next to it all the relations whose similarity to *maxI* is larger than a user-defined threshold τ (which is given as input parameter to the algorithm).
- Practically, the sorted list contains sub-lists of relations, each with decreasing maximum similarity among its members.

Align Tables as L1

Input: the graph $G(V,E)$, $V=V_R \cup V_Q$, $E \subseteq V_Q \times V_R$; the values X_{low}, X_{upper}, Y_{low}, Y_{upper}, and, an offset dY for the representation of the line of relations

Output: a sorted list L of these relations, with the x,y coordinates of each relation computed

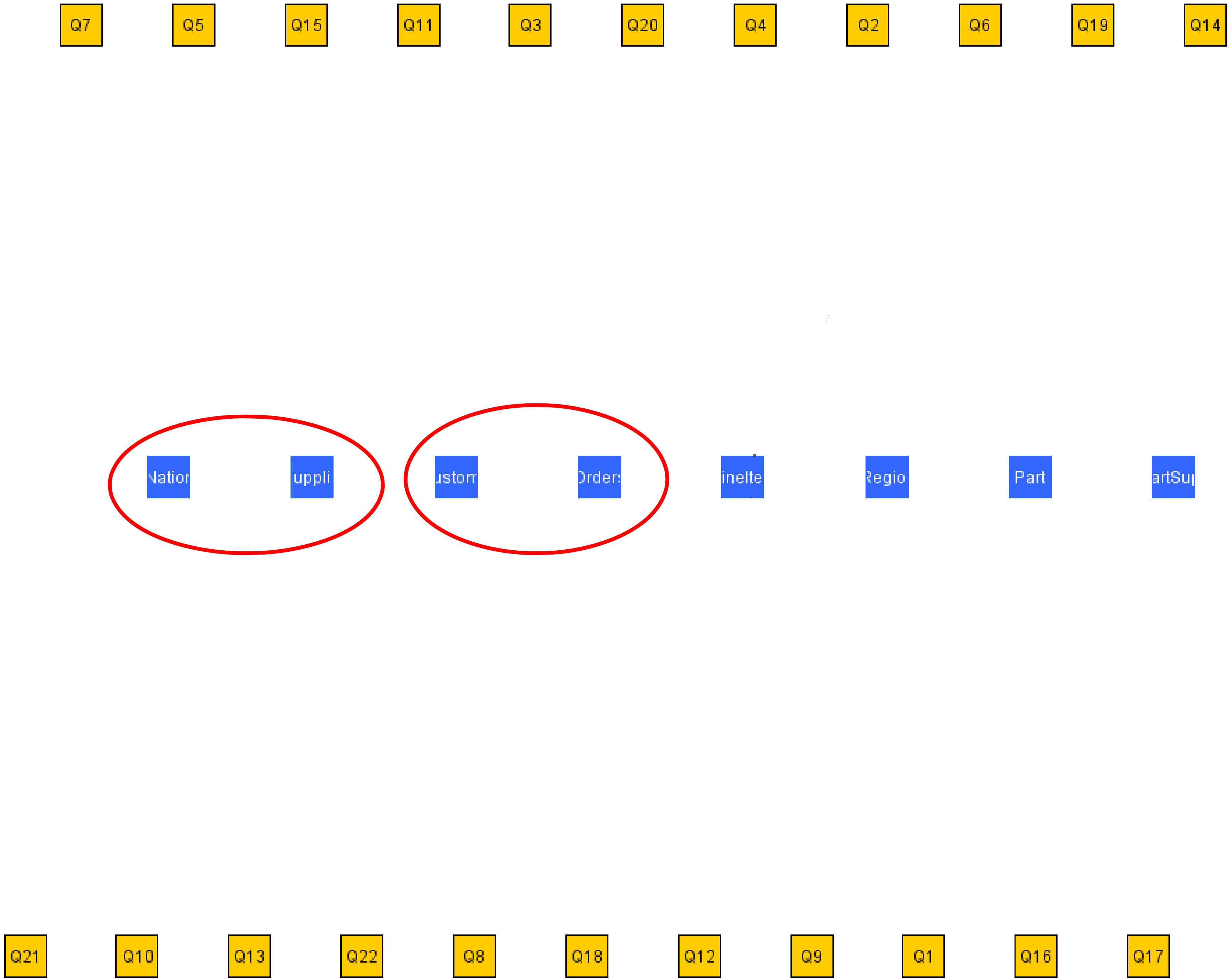
Begin

```
While there exist relations not visited yet{
        For every relation r_i that has not been visited yet{
                For every other relation r_j
                        If JaccT[r_i][r_j] is the current max
                          similarity, maxI := r_i & maxJ := r_j ;
                Place maxI in the next slot in the sorted list L;
                For every other not visited relation r_j {
                        If (JaccT[r_i][r_j] > threshold τ) place r_j
                                next to maxI;
                        Set JaccT[r_i][r_j] = 0;
//to avoid reconsidering it in the future
                }
        }
        If the last relation is not assigned, place it appropriately;
}//all relations are placed in the sorted list by now
Compute a dX common to all relations, dX = (X_upper – X_low) / (N_T + 1)
For every relation r_i assign coordinates x and y as follows:
        r_i.x = X_low +dX * i;  //relations in a line parallel to x-axis
        r_i.y = Y_upper – dY;             //dY is input to the algorithm
```

End.

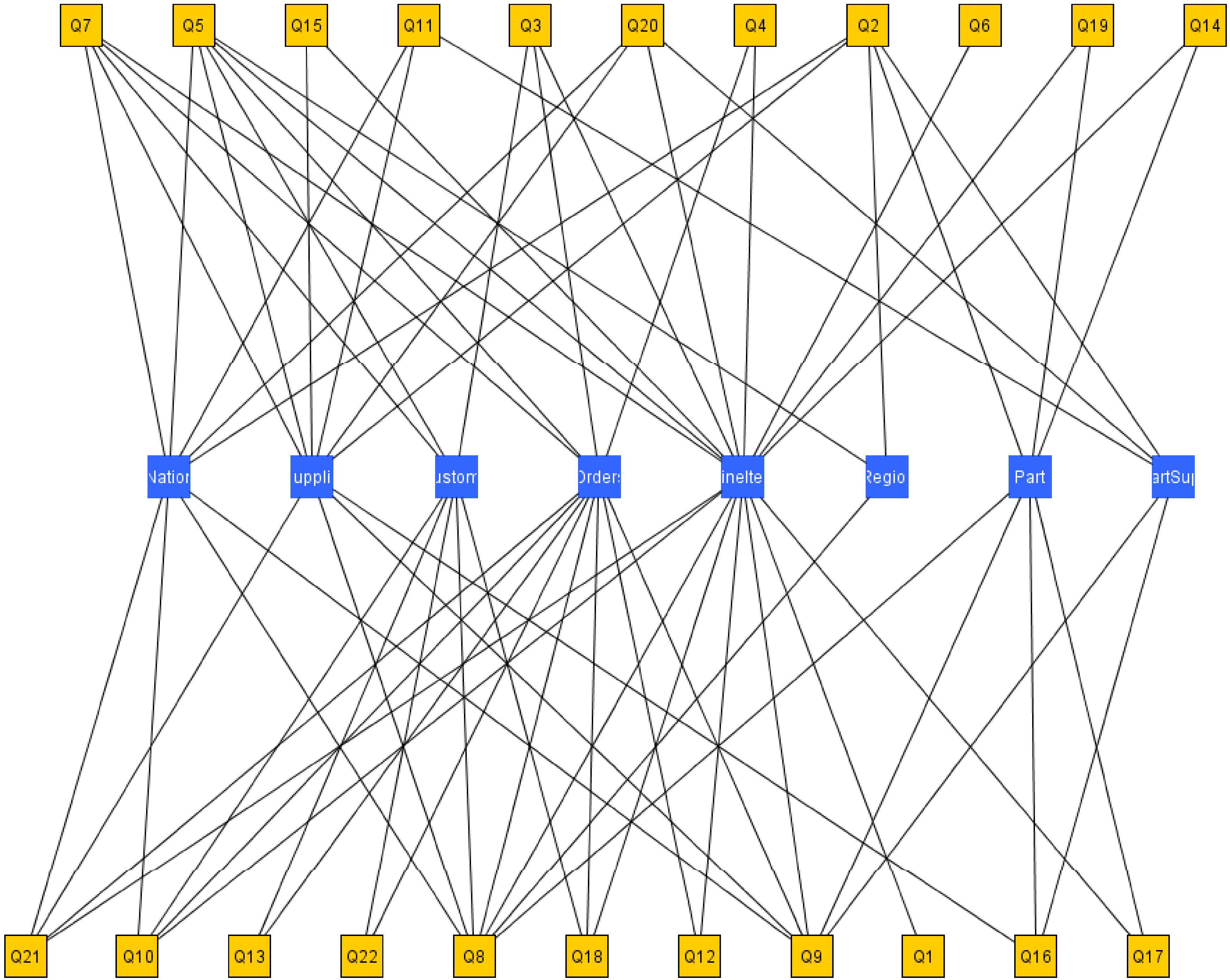
Nation-Supplier & Customer-Orders

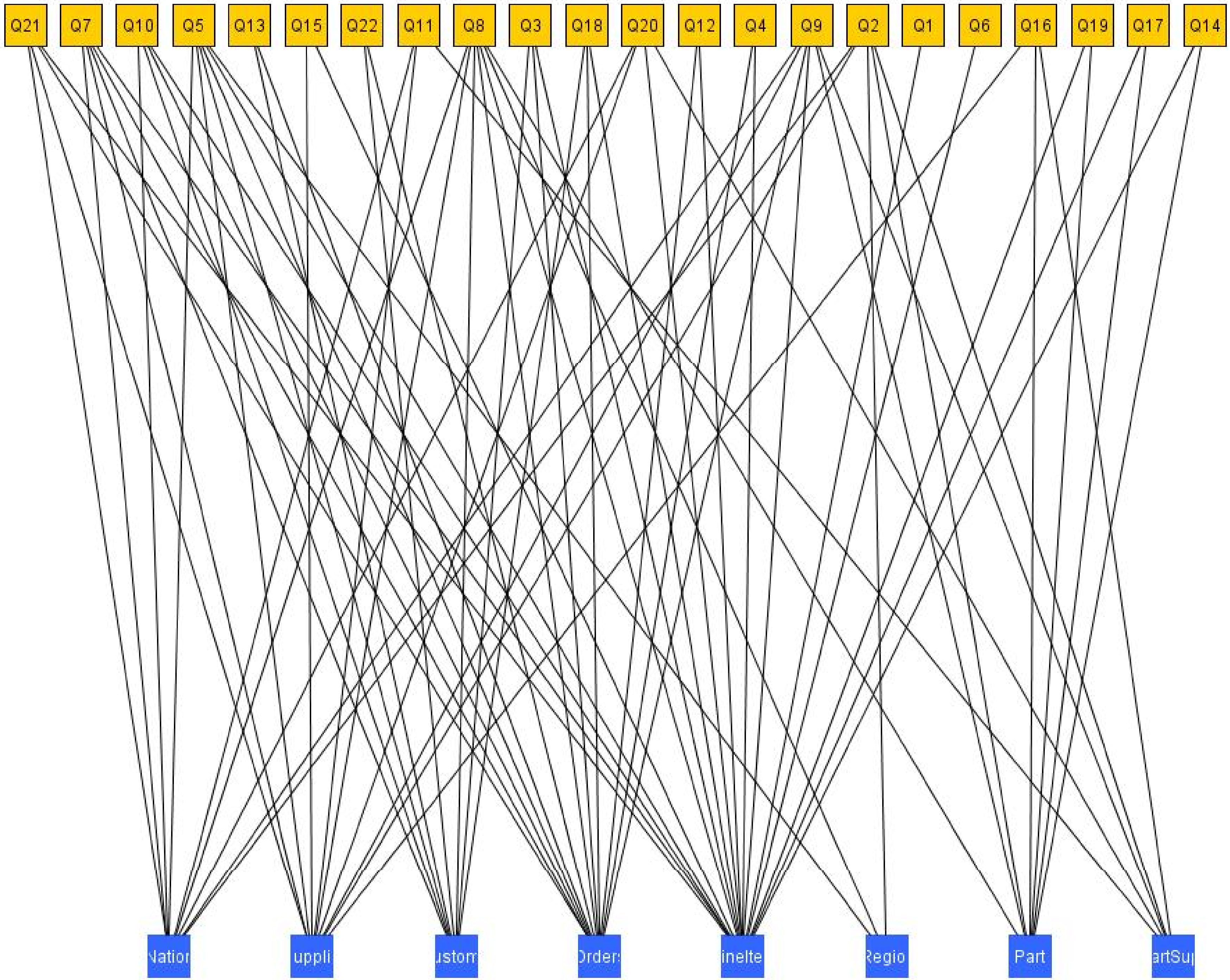
	Region	*Nation*	*Supplier*	*Customer*	*Part*	*Partsupp*	*Lineitem*	*Orders*
Q1	0	0	0	0	0	0	1	0
Q2	1	1	1	0	1	1	0	0
Q3	0	0	0	1	0	0	1	1
Q4	0	0	0	0	0	0	1	1
Q5	1	1	1	1	0	0	1	1
Q6	0	0	0	0	0	0	1	0
Q7	0	1	1	1	0	0	1	1
Q8	1	1	1	1	1	0	1	1
Q9	0	1	1	0	1	1	1	1
Q10	0	1	0	1	0	0	1	1
Q11	0	1	1	0	0	1	0	0
Q12	0	0	0	0	0	0	1	1
Q13	0	0	0	1	0	0	0	1
Q14	0	0	0	0	1	0	1	0
Q15	0	0	1	0	0	0	1	0
Q16	0	0	1	0	1	1	0	0
Q17	0	0	0	0	1	0	1	0
Q18	0	0	0	1	0	0	1	1
Q19	0	0	0	0	1	0	1	0
Q20	0	1	1	0	0	1	1	0
Q21	0	1	1	0	0	0	1	1
Q22	0	0	0	1	0	0	0	1


Q7
Q5
Q15
Q11
Q3
Q20
Q4
Q2
Q6
Q19
Q14
Natior
uppli
ustom
Orders
inelte
Regio
Part
artSup
Q21
Q10
Q13
Q22
Q8
Q18
Q12
Q9
Q1
Q16
Q17

Coordinates

- Once relations and queries are ordered, we give them coordinates
- Y-coordinates fixed for both queries and relations
- X-coordinates:
  - Relations: each relation is placed in the next slot (computed as a *dX* step) in the x-axis
  - Queries have a barycenter computation: we add the x coordinates of all the relations accessed by the query and we divide this sum by the out-degree of the query.

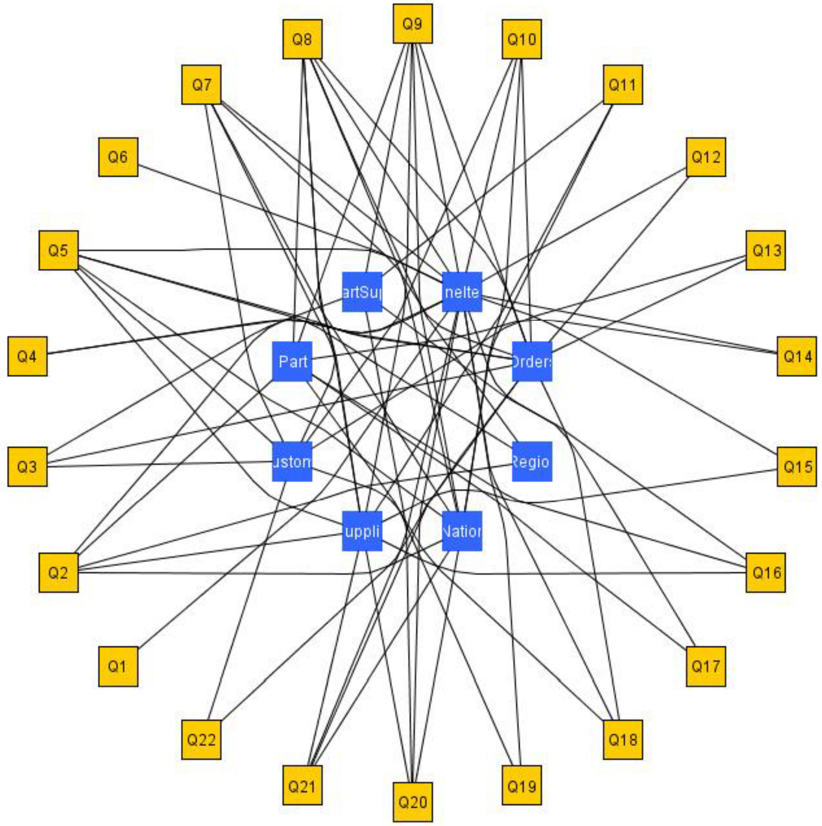

Q7
Q5
Q15
Q11
Q3
Q20
Q4
Q2
Q6
Q19
Q14
Natior
uppli
ustom
Orders
inelte
Regio
Part
artSup
Q21
Q10
Q13
Q22
Q8
Q18
Q12
Q9
Q1
Q16
Q17


Q21
Q7
Q10
Q5
Q13
Q15
Q22
Q11
Q8
Q3
Q18
Q20
Q12
Q4
Q9
Q2
Q1
Q6
Q16
Q19
Q17
Q14
Natior
uppli
ustom
Orders
inelte
Regio
Part
artSup

RADIAL LAYOUTS

Basic idea

- What if we bend the lines, to exploit the extra space?
- Simple radial:
  - Inner circle for relations
  - Outer circle for queries



Simple Radial

Concentric circles

- What –if, instead of 1 query circle, we have many concentric circles?

- Keeps clear separation of queries and relations
- Scales better
- Adapts to different semantics
  - One circle per query class
  - … per source file / package / module
  - … per developer
  - …
  - here: per query fan-out

Extra issues

- Place each query as close to the relations if accesses
- Handle gracefully the issue of large values for the range of accessed relations
- Handle conflicts of nodes falling one on top of another

Input: A graph $G(V,E)$, $V=V_R \cup V_O$, $E \subseteq V_O \times V_R$; a default radius for the relations' circle, ρ and a center (x_0,y_0); the empty space radius between the concentric circles Δ_ρ; a small delta radius for conflicting queries $\Delta_\rho^{conflict}$

Output: an assignment of coordinates to every node v in V; specifically, for every relation r_i, a radius ρ_i and an angle θ_i, resulting in coordinates $r_i.x$ and $r_i.y$; we handle every query q_i similarly, too.

Begin

$\theta_0 = 180 / |V_R|$; *// sector (pizza slice) per relation*

AlignTablesAsL1; *// sort tables by similarity;*

For every relation r_i {

$$\theta_i = \frac{\pi \cdot \theta \cdot (\frac{1}{2} + i)}{180}$$

$r_i.x = x_0 + \rho \cdot cos(\theta_i)$; $r_i.y = y_0 + \rho \cdot sin(\theta_i)$;

}

For every query q_i {

$\theta_{Arc}=0$

For every relation r_i s.t. $(q_i,r_i) \in E${

$$\theta_{Arc} = \theta_{Arc} + \begin{cases} \vartheta_i, \vartheta_{Arc} = 0 \\ \vartheta_i, |\vartheta_{Arc} - \theta_i| < \pi \\ 2\pi - \theta_i, else \end{cases}$$

$\theta_{Arc}=\theta_{Arc}/2$; *//place the node in the bisector of the arc*

}

$\theta_i = \theta_{Arc}$;

$\rho_i = degree(q_i) \cdot \Delta_\rho + \rho$;

}

For every query q_j { *//resolve conflicts*

For every query q_k s.t. $conflict(q_j,q_k)$, $count_j^{conflict}$++; *//count conflicts of q_j*

$\rho_j = \rho_j + \cdot \Delta_\rho^{conflict} \cdot count_j^{conflict}$

$q_i.x = x_0 + \rho_i \cdot cos(\theta_i)$; $q_i.y = y_0 + \rho_i \cdot sin(\theta_i)$;

}

End.

Algorithm Concentric Radial

Concentric Radial

- We take as input
  - the graph of the ecosystem
  - the radius of the starting relations circle
  - the empty space among circles
  - a delta for conflicting queries
- The output is a placement of queries and relations in a 2D canvas

Concentric Radial

- First, Align Tables as L1, such that similar relations are placed one next to the other
- Concert the relations' line to circle, by computing the angle per relation
- For each relation belonging to the next circle (here: depending on the fan-out), compute the arc of its accessing relations & place it in the bisector
  - If some relations are further than 180^o then, count the remainder angle
- If there are conflicts, find the groups of conflicting nodes and push each of them by a small delta radius

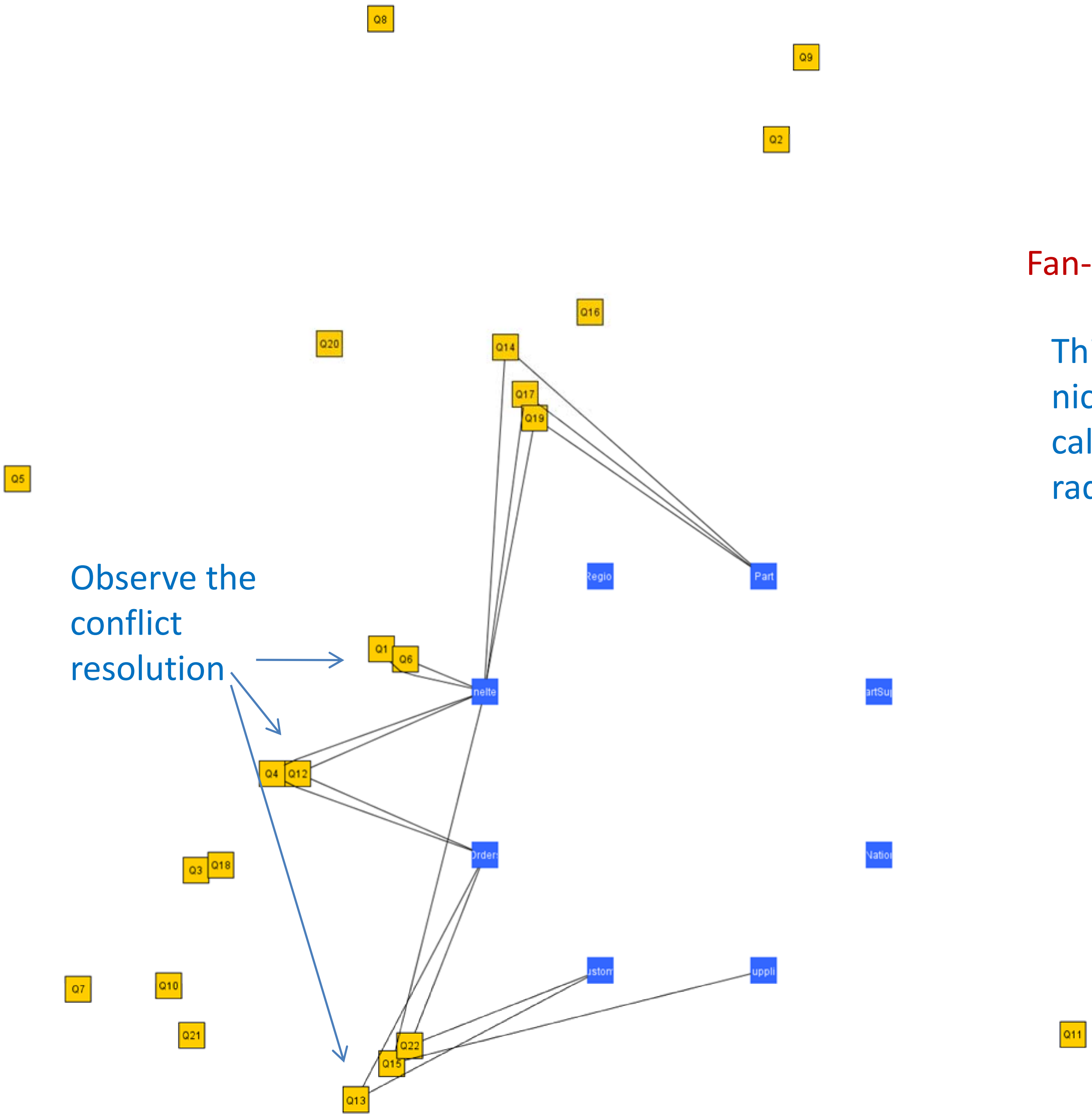
Surprisingly, the most interesting part is the “movie”

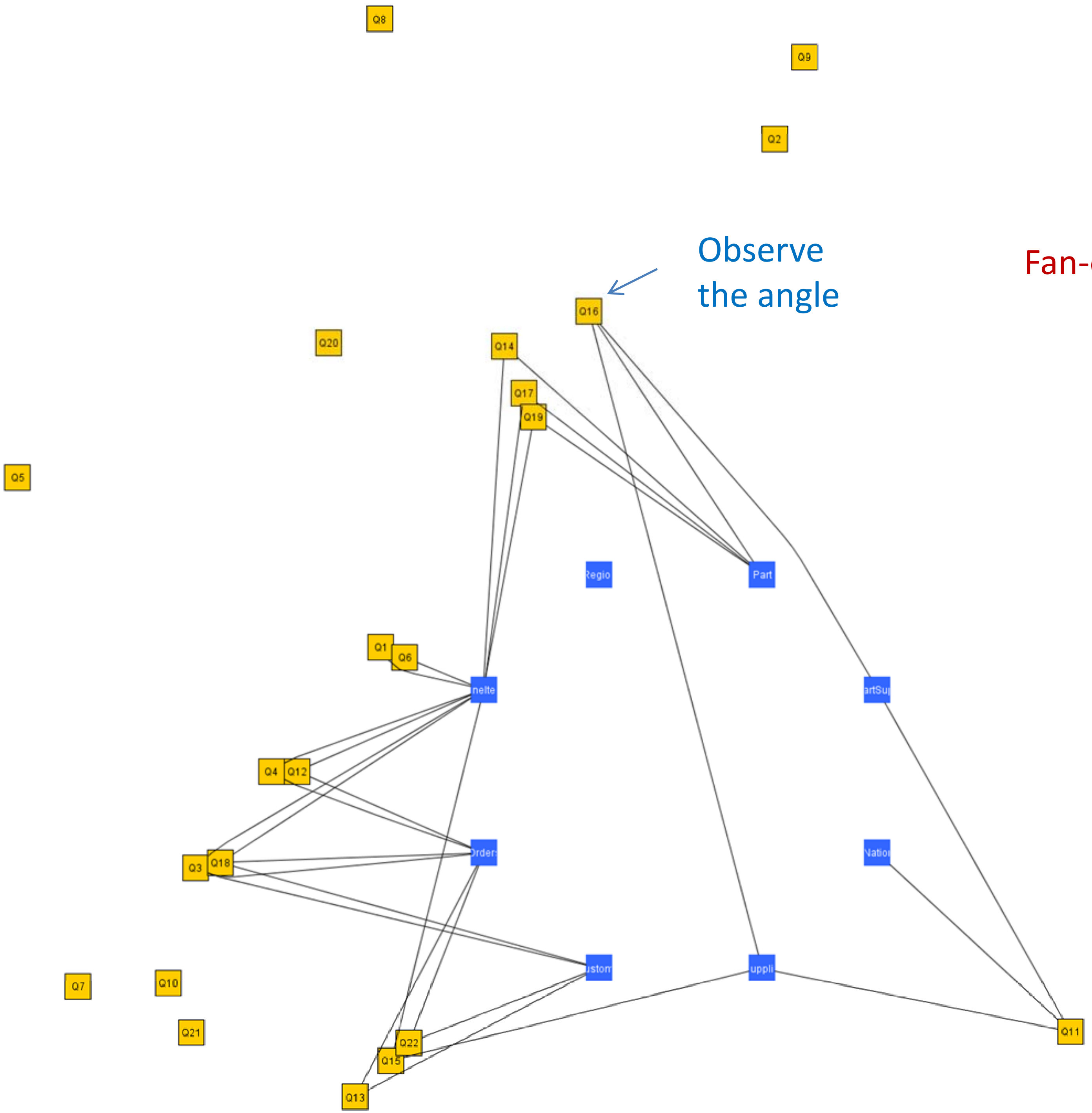
…MOVIE WITH SUBTITLES…

Fan-out = 1,2

Things are nice and calm in radar city

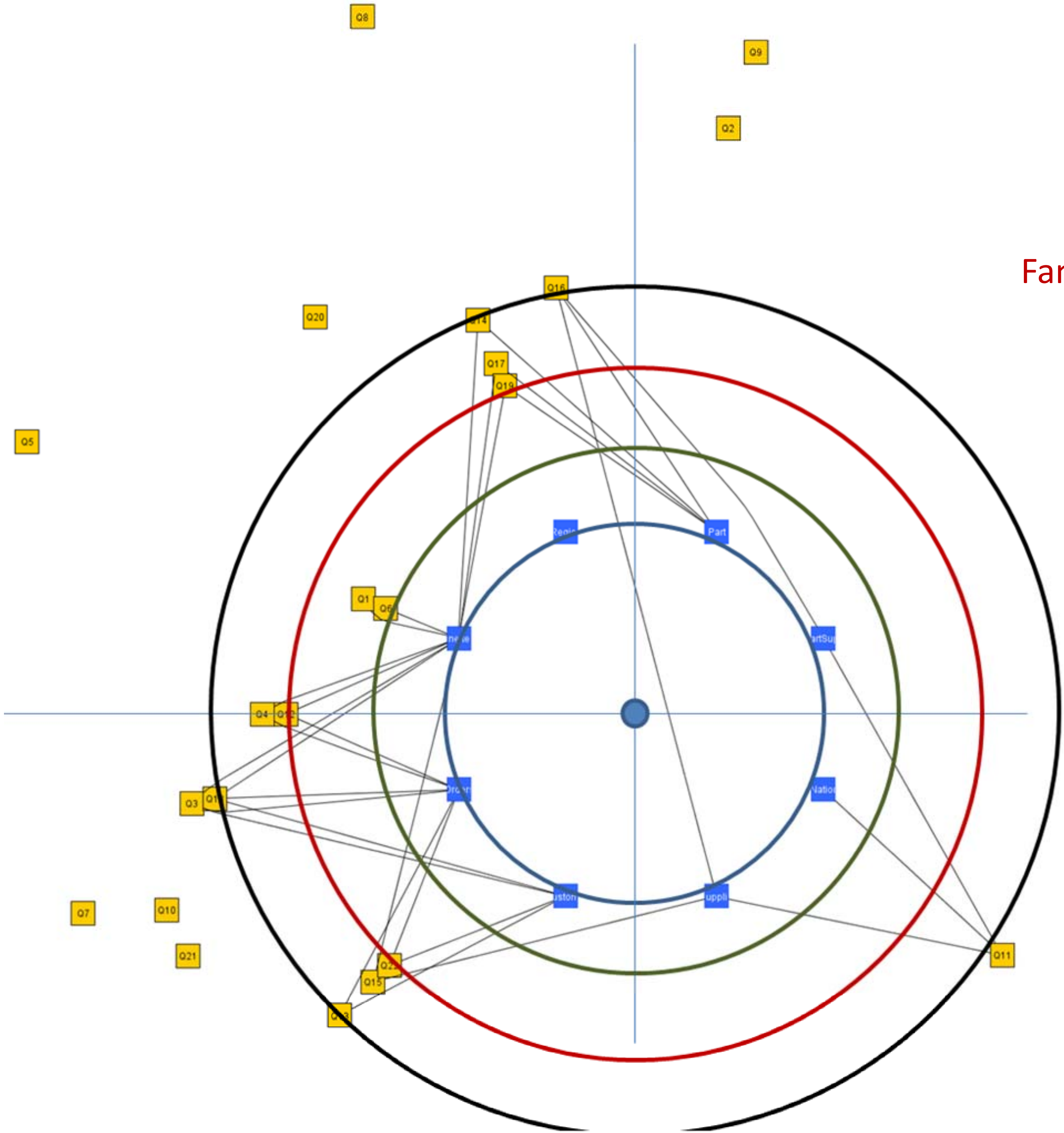
Observe the conflict resolution





Observe the angle

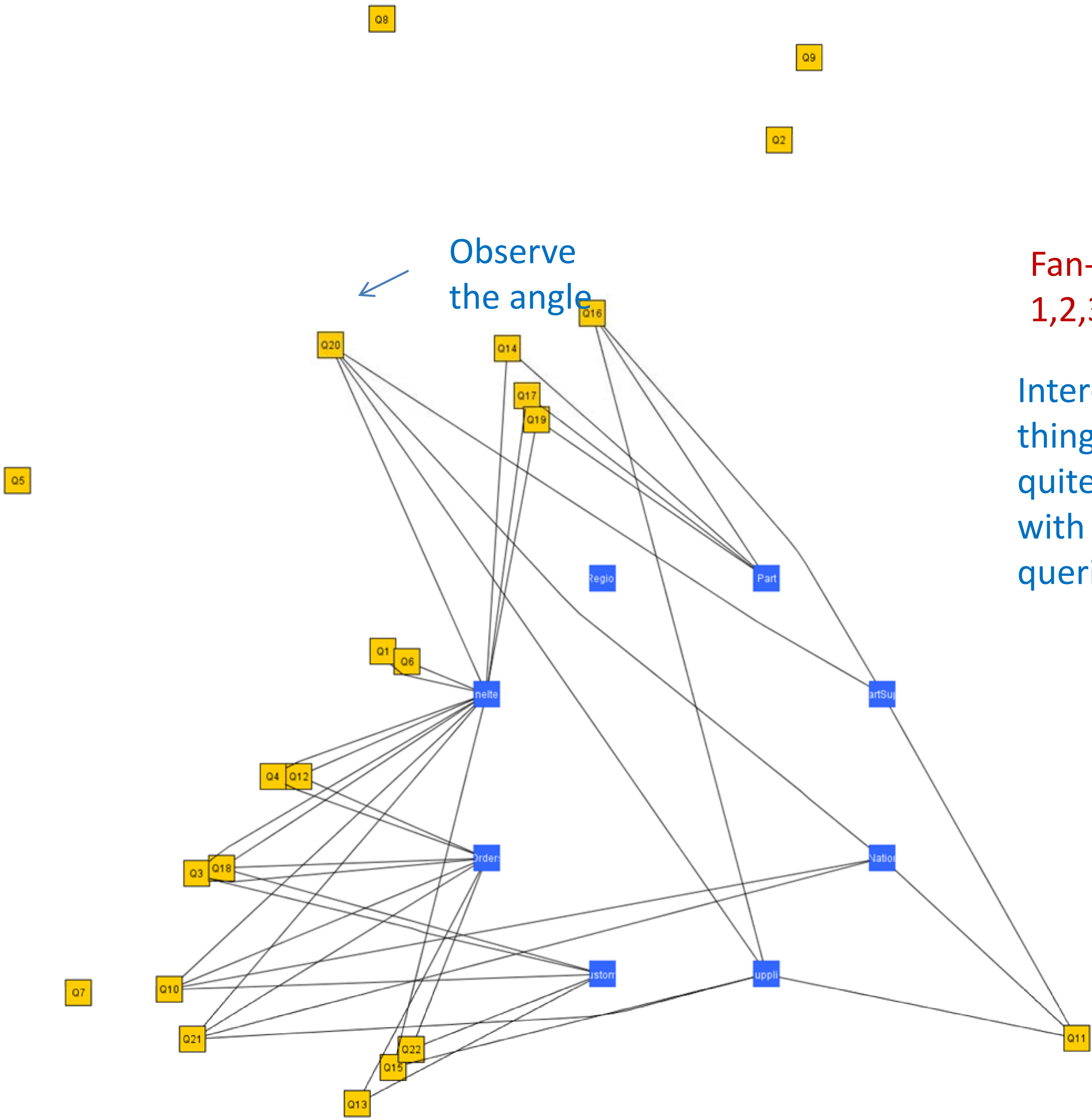
Fan-out = 1,2,3



Fan-out = 1,2,3

See how the concentric circles work

It's called RADAR, remember?



Observe the angle

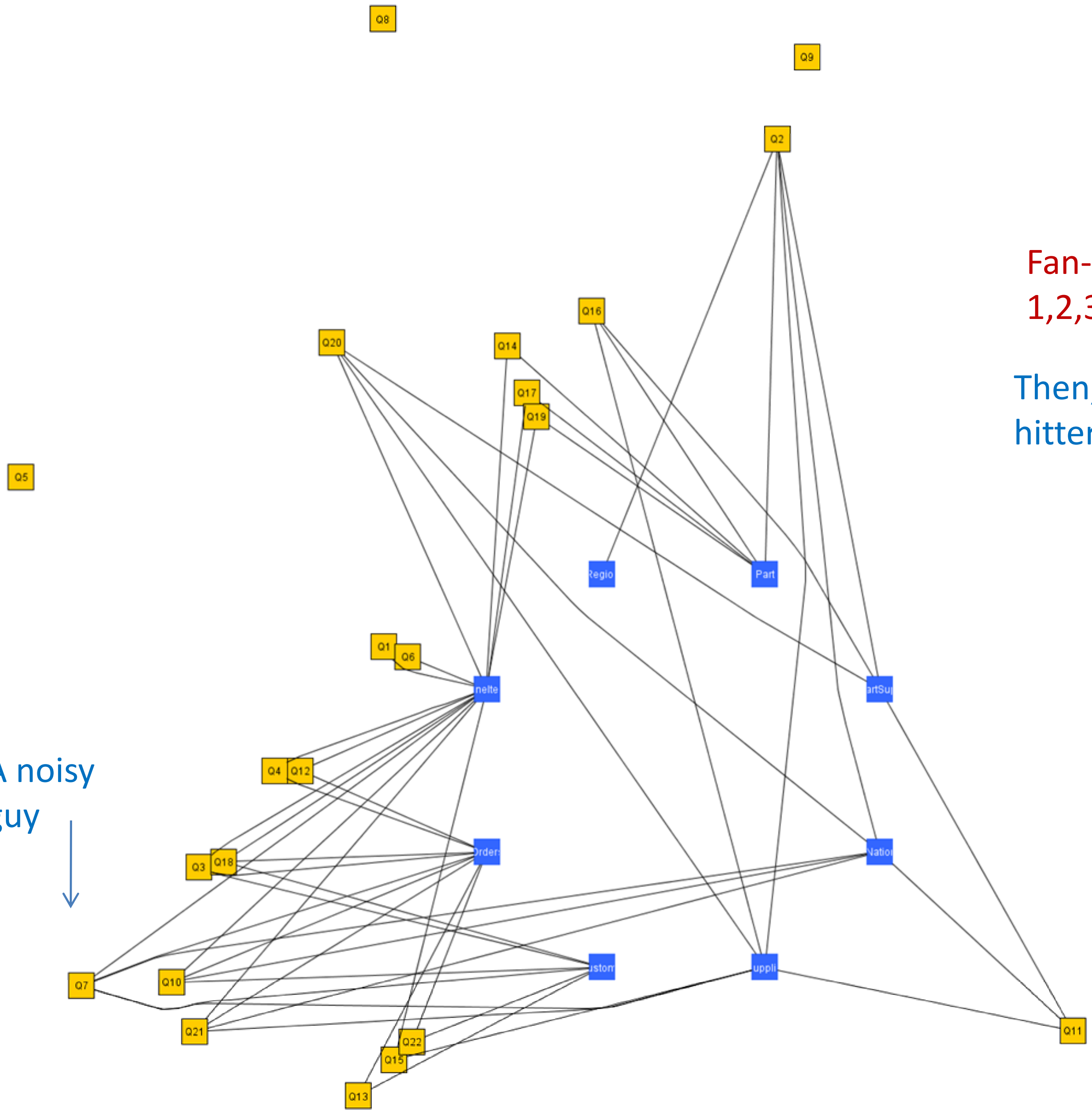
Fan-out = 1,2,3,4

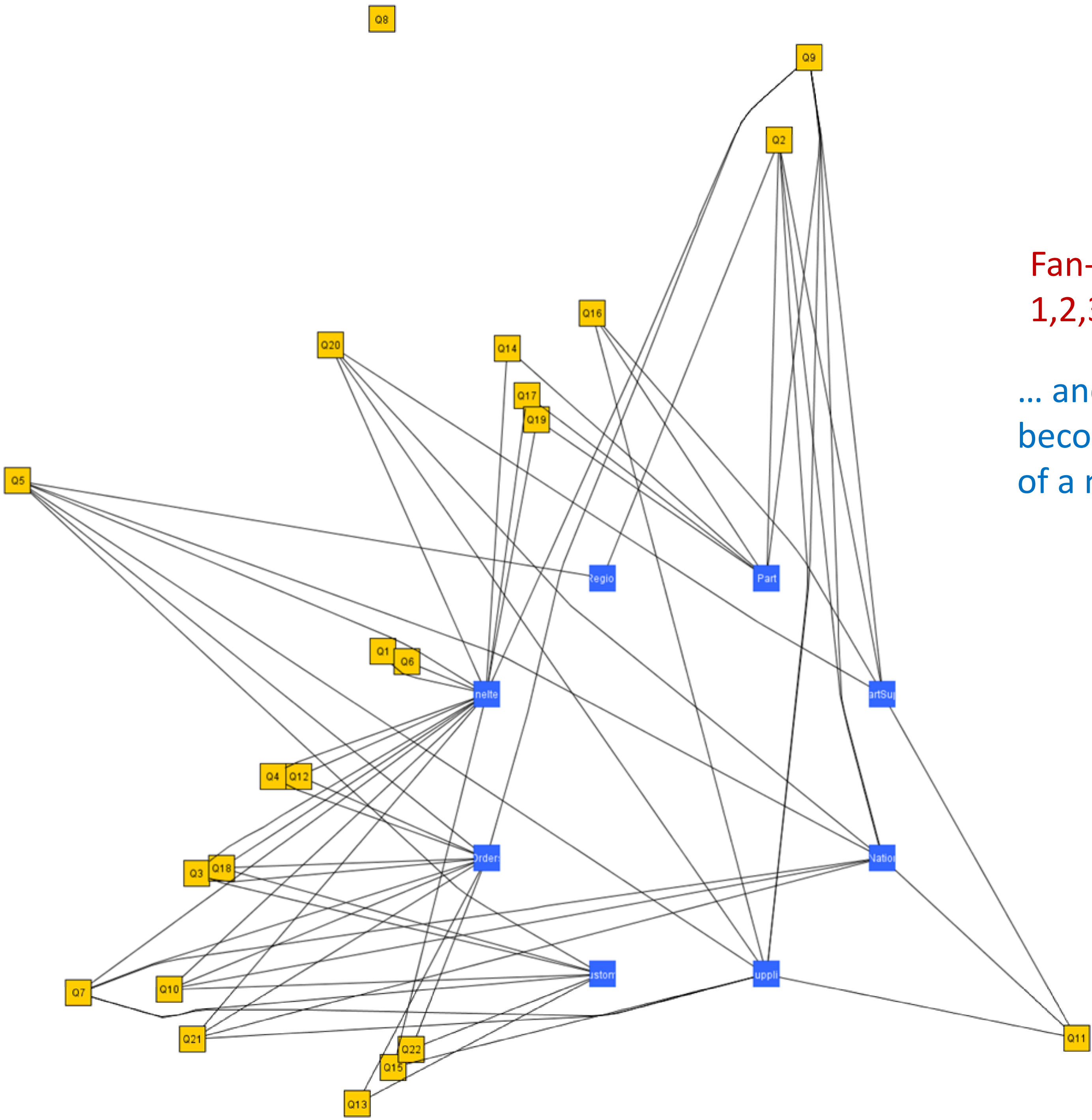
Interestingly, things are still quite clear with 17/22 queries

Fan-out = 1,2,3,4,5

Then, heavy hitters come…

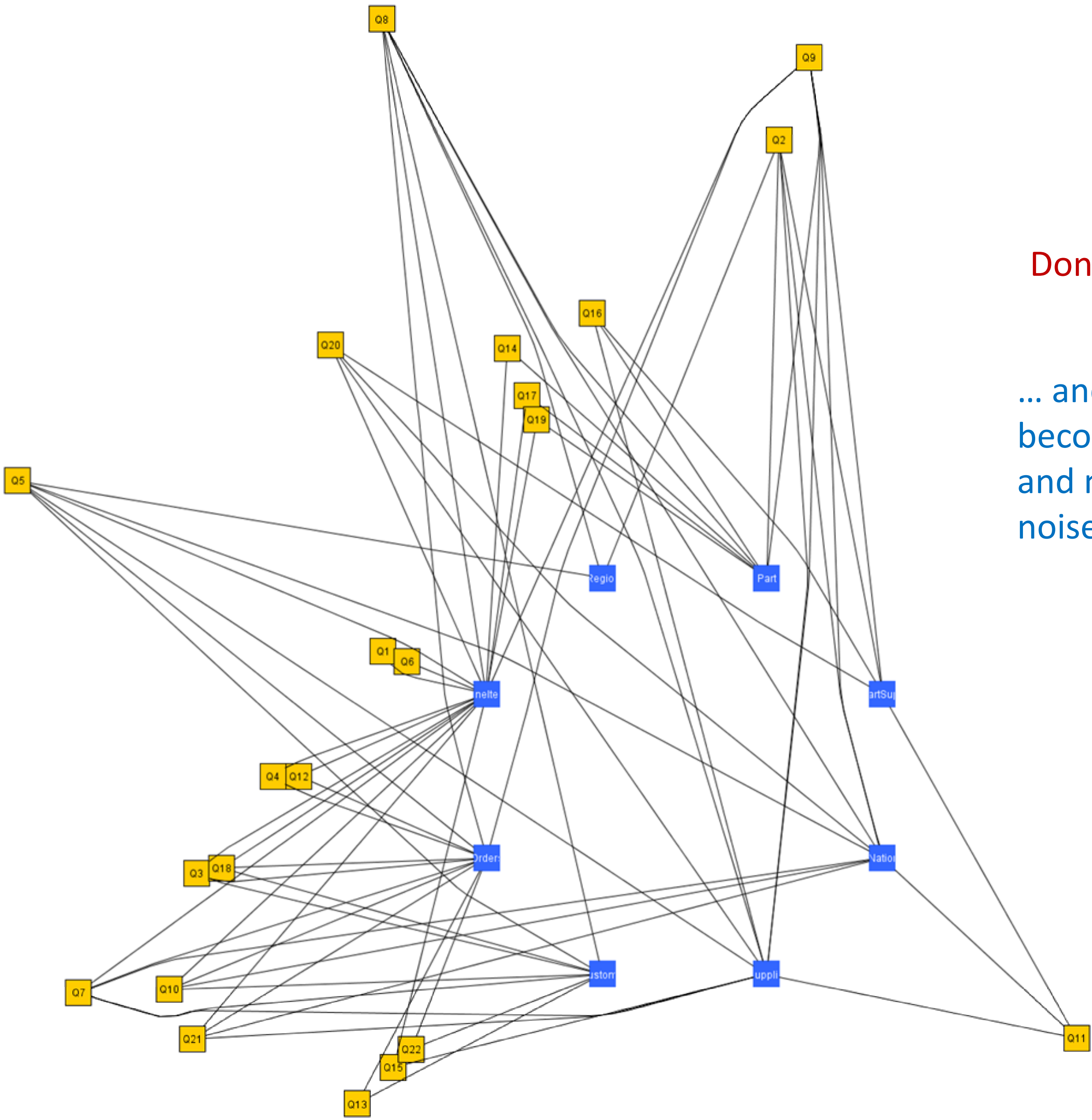
A noisy guy





Fan-out =
1,2,3,4,5,6

… and edges
become more
of a noise …

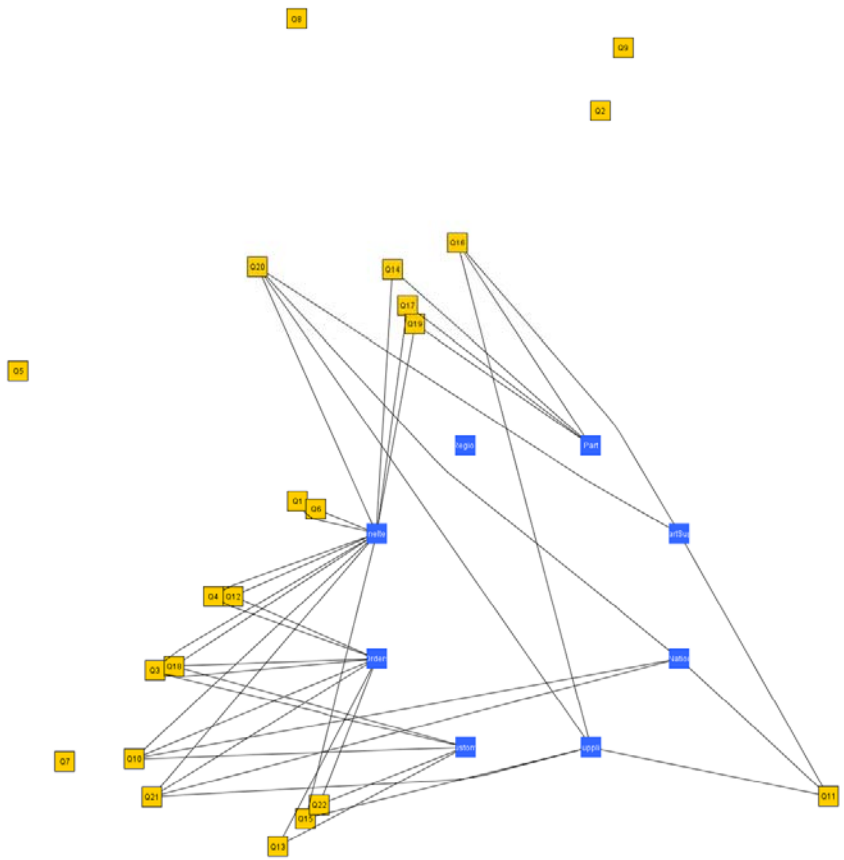


Done

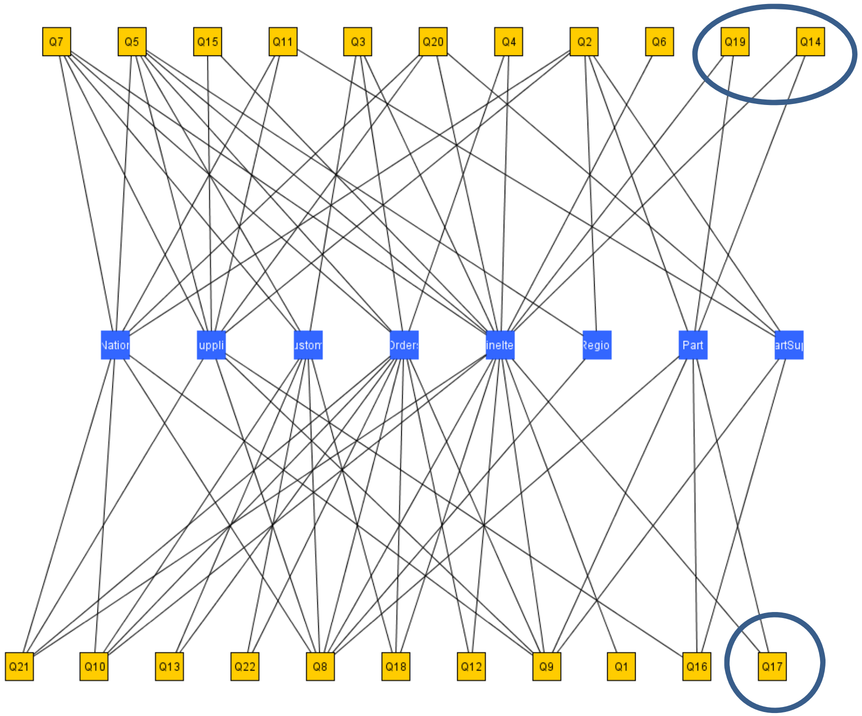
... and edges become more and more of a noise ...

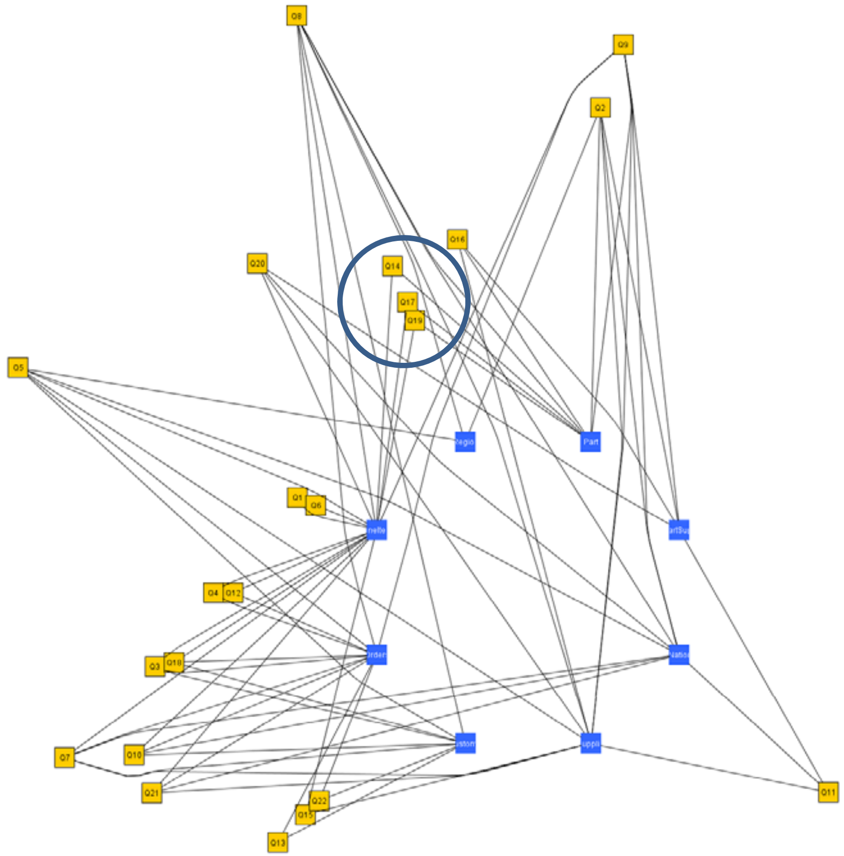
What have we gained?

- Clear diagram
  - (to a large, but not full) extent
- Less noise than barycenter
- Scalability
- … the movie … (!)
  - try it backwards ☺
- The visual clustering of queries (!)



Cluster: Q14, Q17, Q19





CONCLUSIONS AND THOUGHTS FOR THE FUTURE

What we have done

- Two methods for the visualization of data centric ecosystems, both with 2 variants
  - barycentric (here shown only the sandwich variant)
  - Radial – mainly interested to the concentric radial (or RADAR)

- RADAR gives us better scalability, uses the empty space better, has multiple usages for its circles, clusters queries nicely and provides spatial memory

…but…

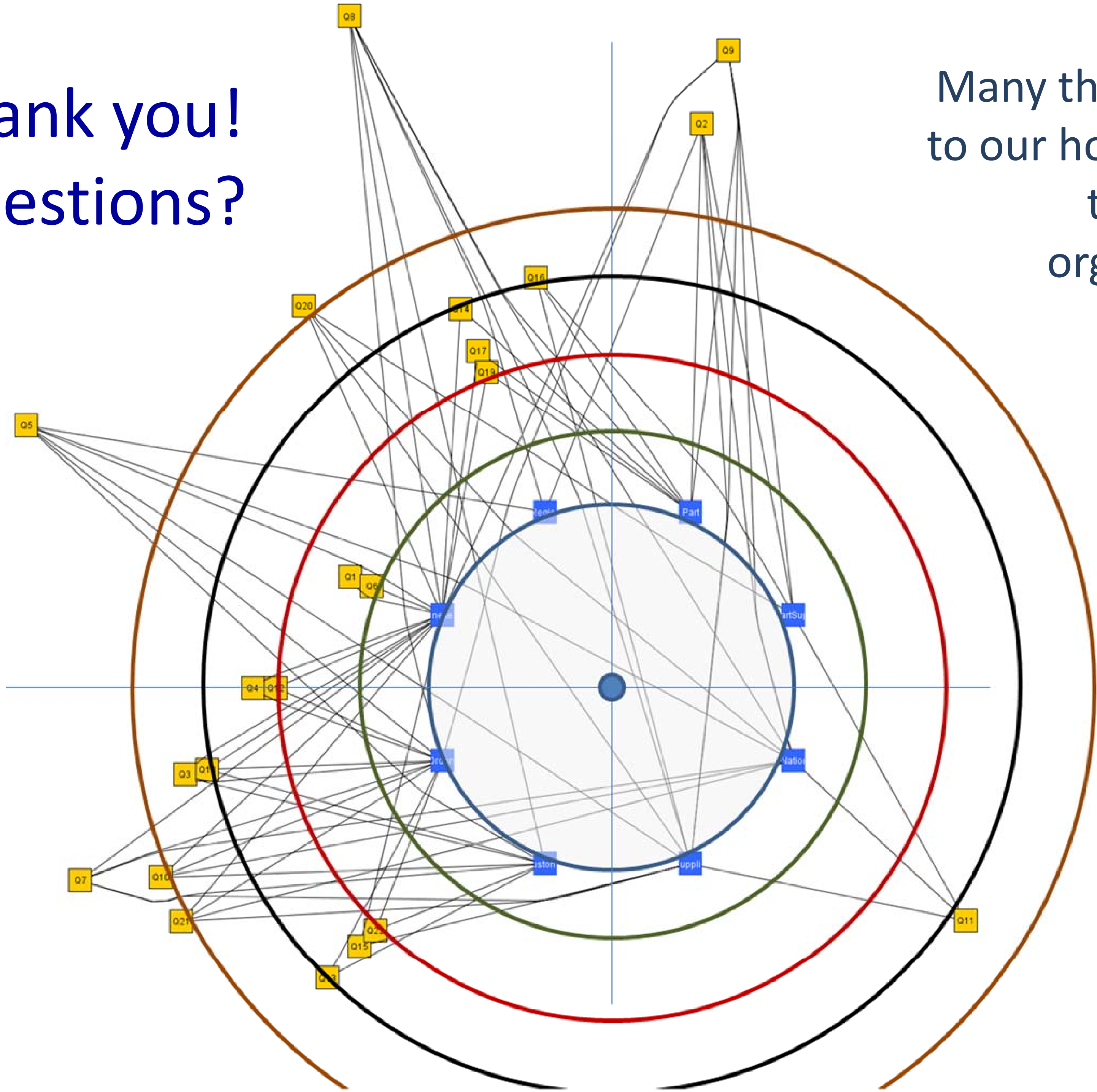
- Heavy hitters make too much noise
  - Possibly stop adding edges after a threshold circle (remember: here fan-out determines the circle)
  - Unfortunately, agglomerative clustering of queries soon creates heavy hitters, too ☹
- Need for many real world examples to test scalability
  - Any input/suggestion/… is most welcome!!
- Not yet incorporated in our impact analysis tool, HECATAEUS
  - http://www.cs.uoi.gr/~pvassil/projects/hecataeus

…and…

- More sophisticated modeling
  – of the graph (e.g., with query semantics incorporated)
  – of the distance function
  – with views included
  – of both the logical (current) graph with its physical counterparts (i.e., info on scripts, stored procedures, triggers, …)

Thank you!
Questions?

Many thanks go to our hosts and the W/S organizers



http://www.cs.uoi.gr/~pvassil/projects/hecataeus