GRADES 2014

Using semijoin programs to solve traversal queries in graph databases

Norbert Martínez, David Domínguez

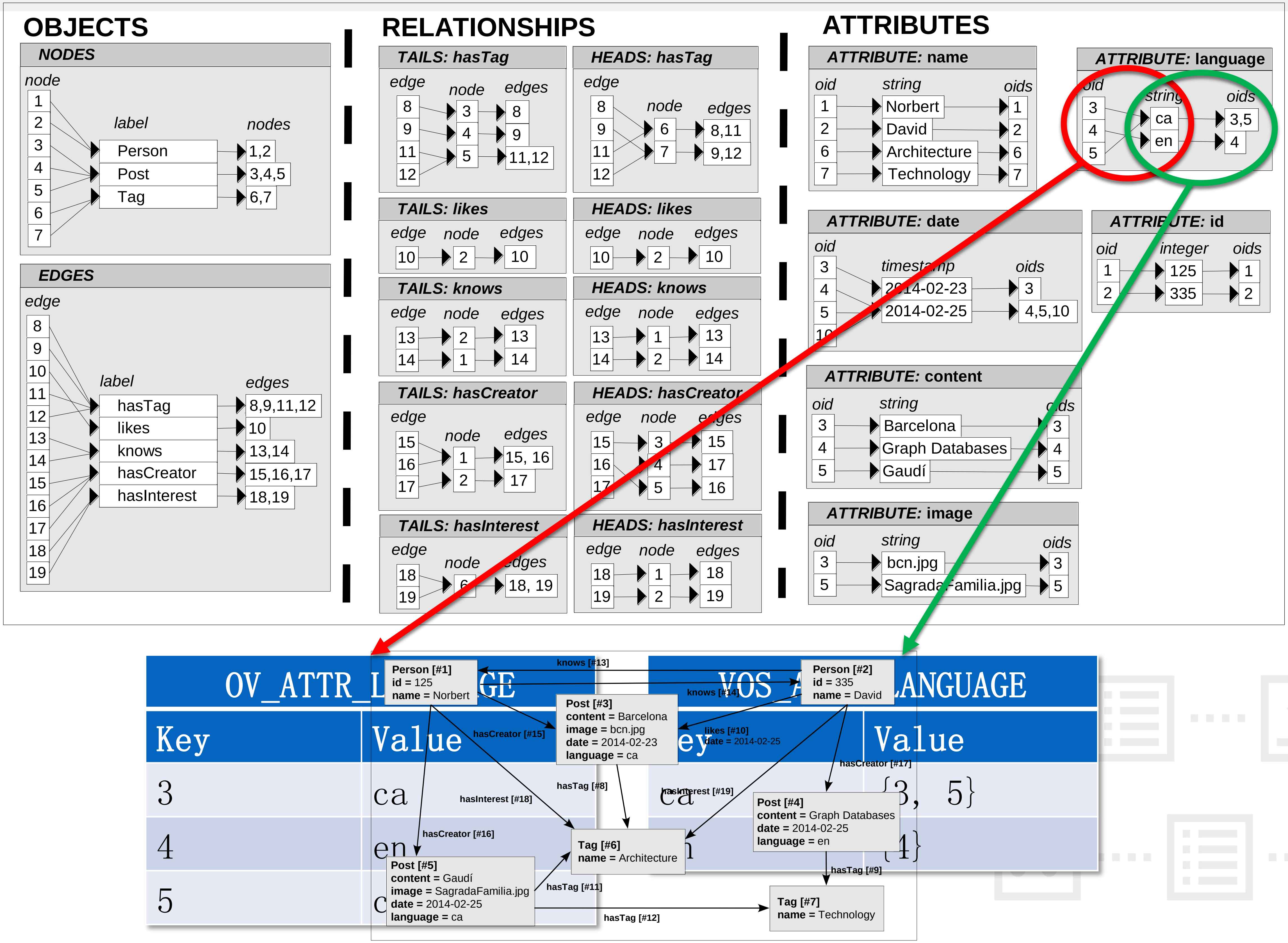
June 22, 2014 - Snowbird, Utah, USA

State-of-the-art in the management of large graphs

Technology	Solutions	Querying systems
RDF	Virtuoso RDF-3X …	SPARQL
Distributed frameworks	Pregel GraphLab …	API DSL (e.g. Green-Marl)
Graph Databases	Sparksee (DEX) Neo4J Titan …	API (e.g. BluePrints) Gremlin Cypher GQL …

- Querying graph databases
  - lack of standards
  - very efficient proprietary APIs for procedural languages
  - DSL or languages that combine data flows of results from APIs
- Open challenges for GDB querying systems
  - define an algebra of graph query operations
  - optimization of graph query plans
  - generic solution to solve different graph queries
    - edge traversals
    - attribute filters and aggregation
    - graph pattern matching
    - graph algorithms

- An algebra with a set of operations to solve graph queries in Sparksee
  - adapted from the relational algebra
    - *select*, *semijoin*, *project*, *group*, *sort*, …
  - extensions
    - *foreach*: collection-oriented procedures (parallel)
    - *while-do*: iterative (recursive) programs
  - data structures in the form of key-value pairs
  - compatible with the current Sparksee API
- Optimization of graph query plans
  - classical database query optimization techniques
  - new specific for graph patterns and graph traversals
  - take advantage of Sparksee's compressed bitmap


OBJECTS
NODES
node
1
2
3
4
5
6
7
label
nodes
Person
Post
Tag
1,2
3,4,5
6,7
EDGES
edge
8
9
10
11
12
13
14
15
16
17
18
19
label
edges
hasTag
likes
knows
hasCreator
hasInterest
8,9,11,12
10
13,14
15,16,17
18,19
RELATIONSHIPS
TAILS: hasTag
edge
node
edges
8
9
11
12
3
4
5
8
9
11,12
HEADS: hasTag
edge
node
edges
8
9
11
12
6
7
8,11
9,12
TAILS: likes
edge
node
edges
10
2
10
HEADS: likes
edge
node
edges
10
2
10
TAILS: knows
edge
node
edges
13
14
2
1
13
14
HEADS: knows
edge
node
edges
13
14
1
2
13
14
TAILS: hasCreator
edge
node
edges
15
16
17
1
2
15, 16
17
HEADS: hasCreator
edge
node
edges
15
16
17
3
4
5
15
17
16
TAILS: hasInterest
edge
node
edges
18
19
6
18, 19
HEADS: hasInterest
edge
node
edges
18
19
1
2
18
19
ATTRIBUTES
ATTRIBUTE: name
oid
string
oids
1
2
6
7
Norbert
David
Architecture
Technology
1
2
6
7
ATTRIBUTE: language
oid
string
oids
3
4
5
ca
en
3,5
4
ATTRIBUTE: date
oid
timestamp
oids
3
4
5
10
2014-02-23
2014-02-25
3
4,5,10
ATTRIBUTE: id
oid
integer
oids
1
2
125
335
1
2
ATTRIBUTE: content
oid
string
oids
3
4
5
Barcelona
Graph Databases
Gaudí
3
4
5
ATTRIBUTE: image
oid
string
oids
3
5
bcn.jpg
SagradaFamilia.jpg
3
5
OV_ATTR_LANGUAGE
Key
Value
3
4
5
ca
en
ca
VOS_ATTR_LANGUAGE
Key
Value
ca
en
{3, 5}
{4}
Person [#1]
id = 125
name = Norbert
Person [#2]
id = 335
name = David
knows [#13]
knows [#14]
Post [#3]
content = Barcelona
image = bcn.jpg
date = 2014-02-23
language = ca
hasCreator [#15]
likes [#10]
date = 2014-02-25
hasCreator [#17]
hasTag [#8]
hasInterest [#18]
hasInterest [#19]
Post [#4]
content = Graph Databases
date = 2014-02-25
language = en
hasCreator [#16]
Tag [#6]
name = Architecture
Post [#5]
content = Gaudí
image = SagradaFamilia.jpg
date = 2014-02-25
language = ca
hasTag [#11]
hasTag [#9]
Tag [#7]
name = Technology
hasTag [#12]

Sparksee APIs

```
NODES(T) ::= select(scan("VOS_LABELS"), k=T)
NEIGHBORS_OUT(N, E) ::= semijoin(N, scan("OUT_" || E), k=k)
DEGREE_OUT(N, E) ::= group(NEIGHBORS_OUT(N, E), [], [sum(length(v))])
```

Graph Algorithms

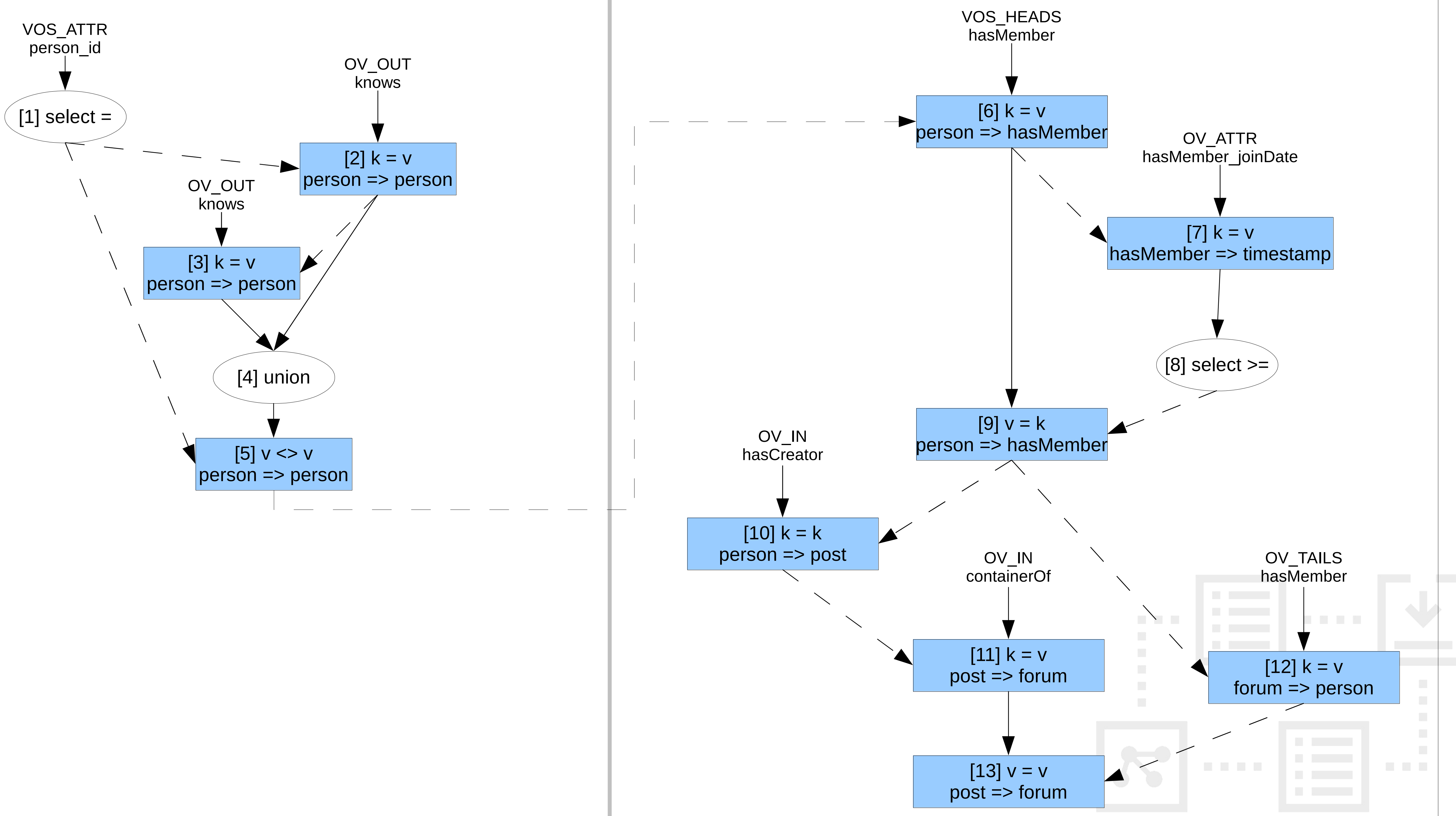
```
HUB(N, E) ::= let D = foreach X in NODES(N)
                          DEGREE_OUT(X, E)
               in semijoin(D, group(D, [], [max(v)]), v=v)


BFS(N, E, K) ::= with R=N
                 do S=R,
                    R=NEIGHBORS_OUT(S, E)
                 until semijoin(R, S, k<>k)
                 steps K
                 return R
```

LDBC SNB summary of operation usage

Operation	Q1	Q2	Q3	Q4	Q5	Q6	Total	%
EQUIJOIN	16	6	4	1	1	1	29	11.2%
FOREACH	3	4	1				8	3.1%
GROUP	4		2	1	1	1	9	3.5%
LIMIT	1	1		1	1	1	5	1.9%
PRODUCT	4	3	2				9	3.5%
PROJECT	12	7	5	1	1	1	27	10.4%
SCAN	23	14	7	6	7	6	63	24.2%
SELECT	3	3	4	3	2	2	17	6.5%
SEMIJOIN	30	13	11	7	10	10	81	31.2%
SORT	1	1		1	1	1	5	1.9%
UNION	1	3	1		1	1	7	2.7%

Friends and friends of friends

Forum membership and posts



semijoin(R, S, R.x = S.y):

- R.x is an oid (key *k*)
  - usually R is a persistent KVP and R.x is indexed
  - S.y is a collection of oids: bitmap union and test
- R.x is a collection of oids (values *v*)
  - in general R is a temporary result inside the query pipeline
  - bitmap intersections

semijoin	Q1	Q2	Q3	Q4	Q5	Q6	Total	%
k=k	23	6	2	4	1	2	38	46.9%
k=v	7	7	8	3	6	5	36	44.5%
v=k					1	2	4	4.9%
v=v			1		1	1	3	3.7%

- Our proposal is a compromise between:
  - the relational model and the noSQL approach
  - key-value pairs storages and graph querying systems
- The main operation is the semijoin:
  - widely used in database technology for distributed systems
  - semijoin program optimization has been studied in detail
  - specific semijoin optimizations on compressed collections of oids
- Procedural extensions for parallel subquery execution and recursion
  - used to simulate graph analytical frameworks for the computation of complex graph algorithms

Thanks!

Q&A