Centre for
Data Analytics

Insight

Gagg: A Graph Aggregation Operator

Fadi Maali*, Stephane Campinas, Stefan Decker

June 2nd 2015

ESWC2015

* Funded by the Irish Research Council

IRISH RESEARCH COUNCIL
An Chomhairle um Thaighde in Éirinn

DCU

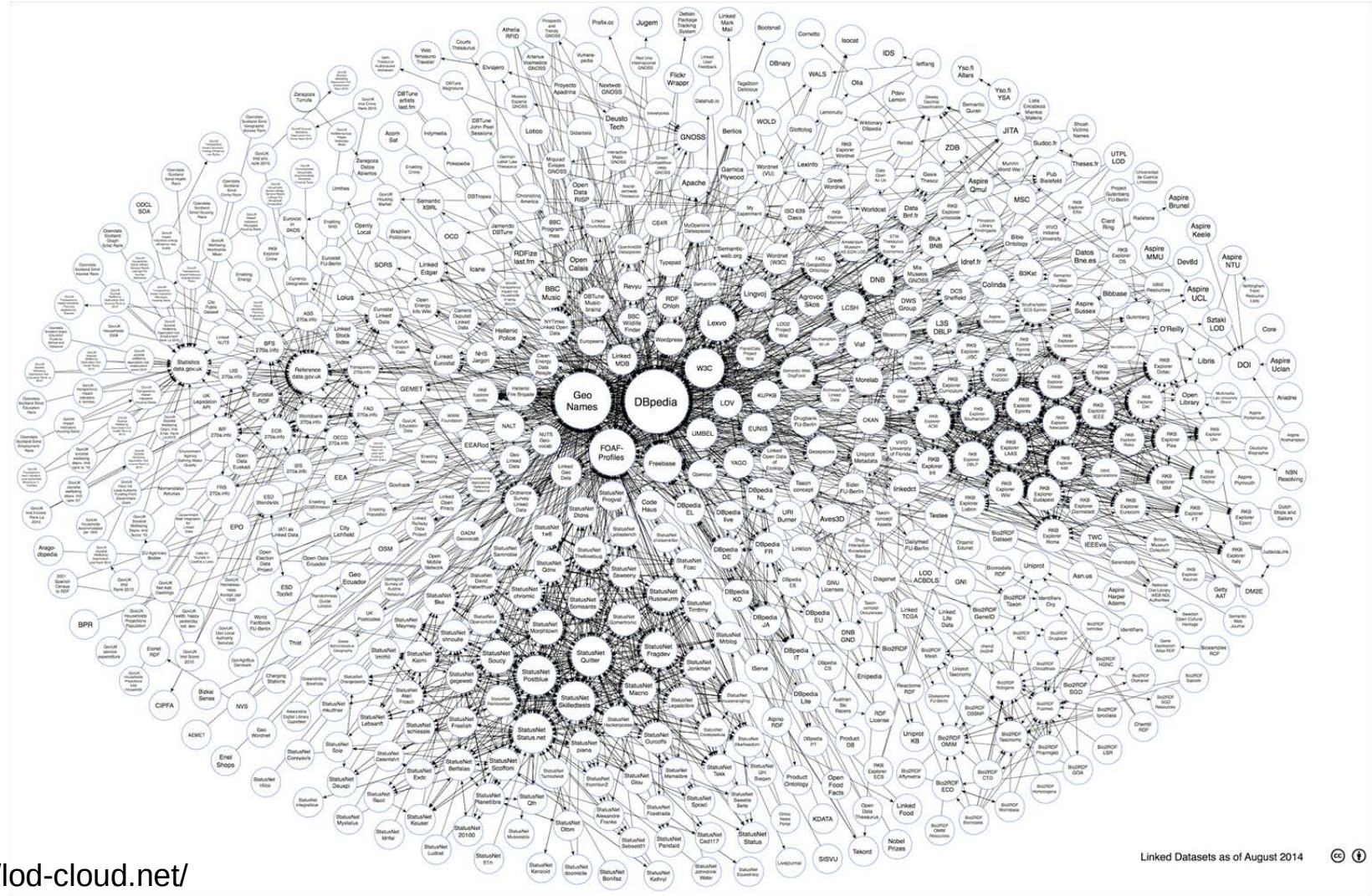
NUI Galway
OÉ Gaillimh

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

UCD
DUBLIN

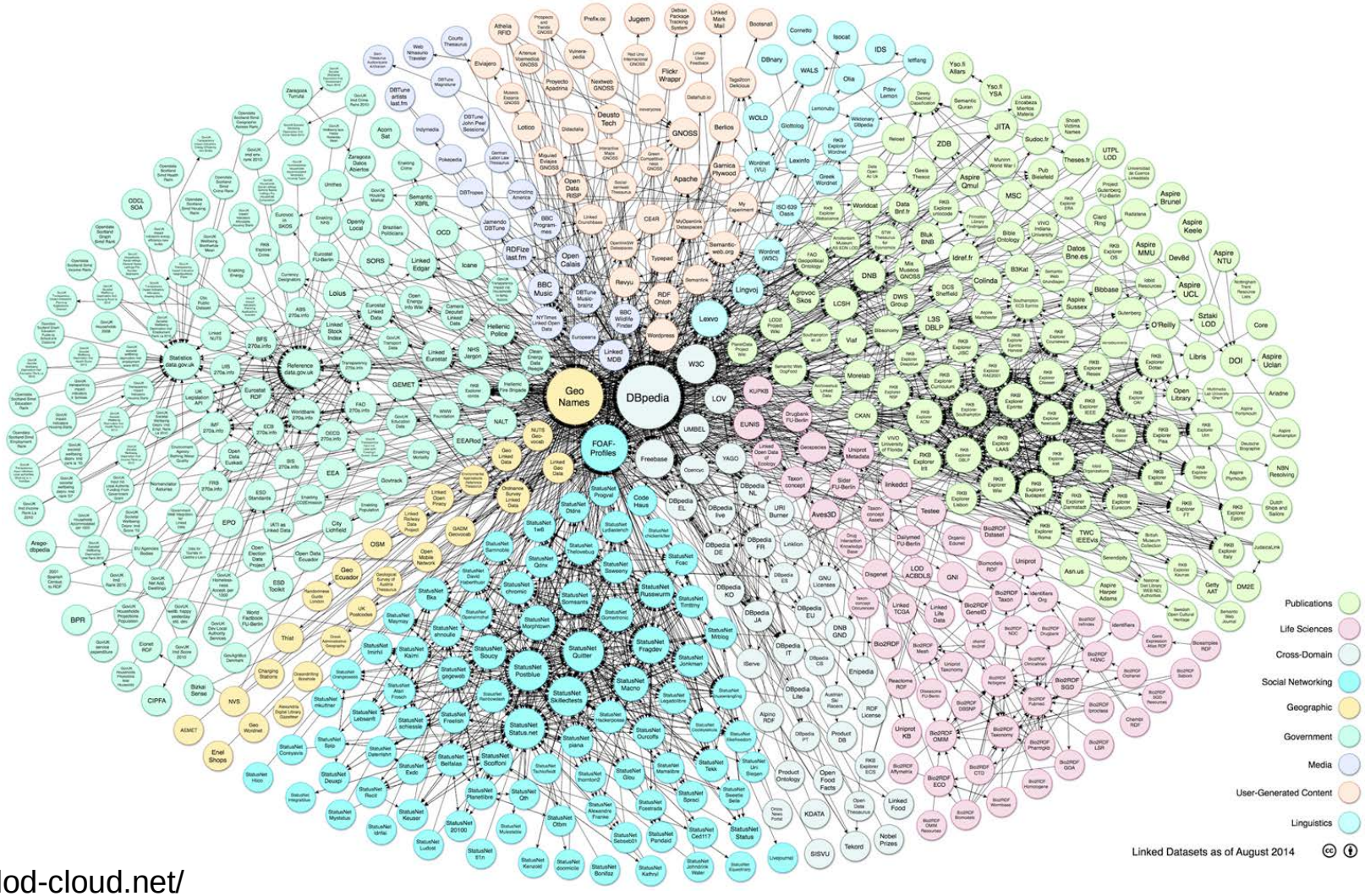
sfi

The Famous LOD Cloud



http://lod-cloud.net/

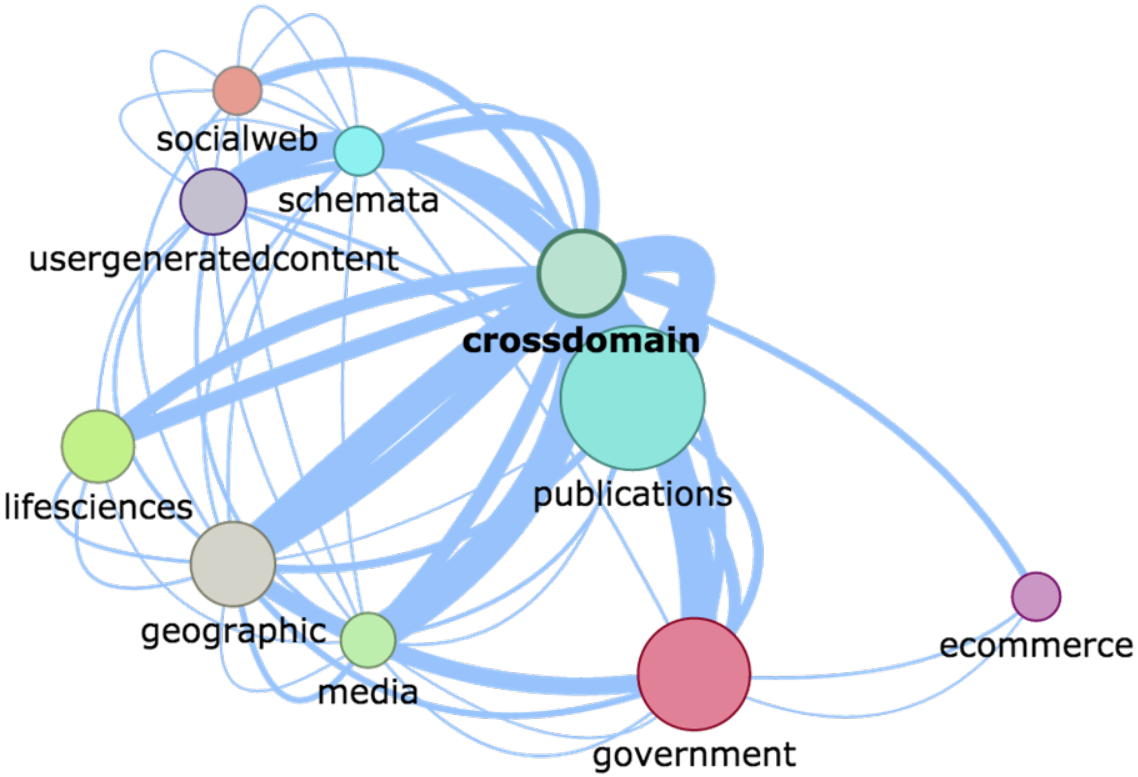
The Famous LOD Cloud - COLOURED



http://lod-cloud.net/

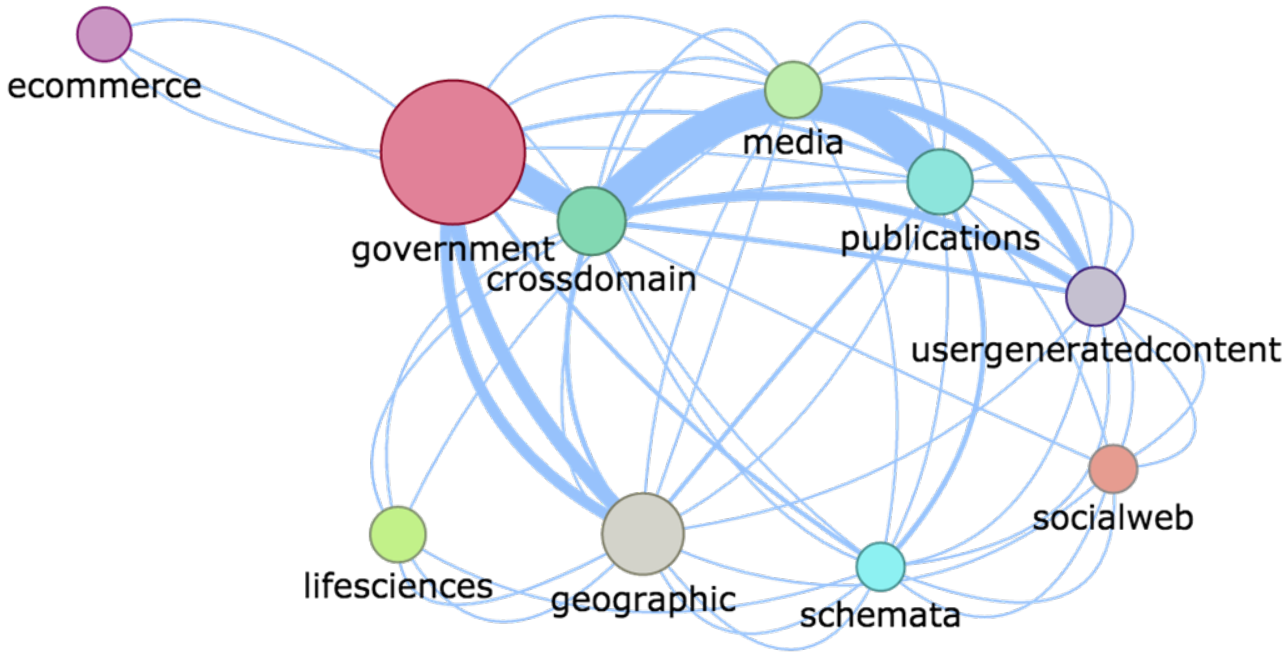
The Famous LOD Cloud

(from a different Angel)



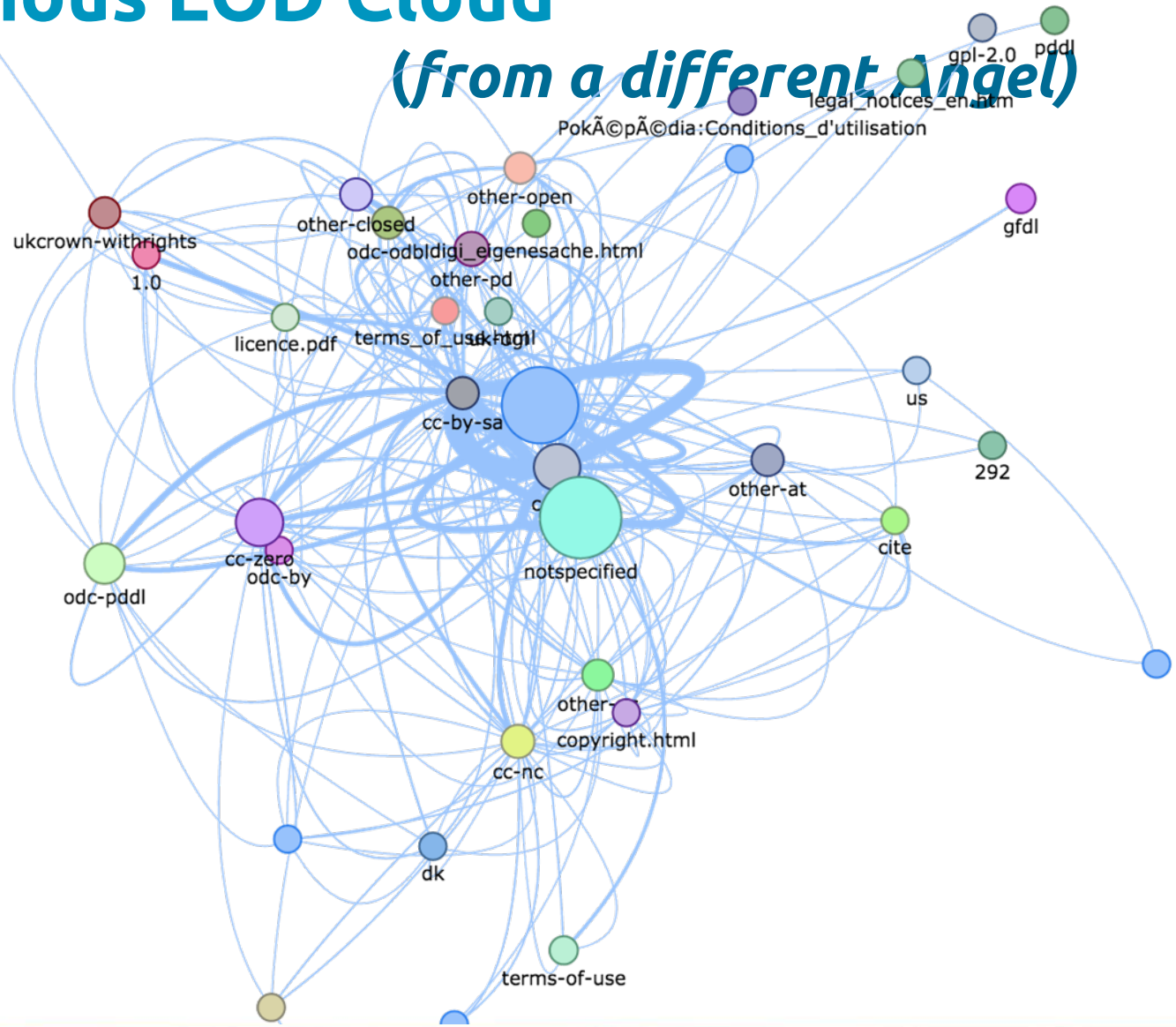
The Famous LOD Cloud

(from a different Angel)



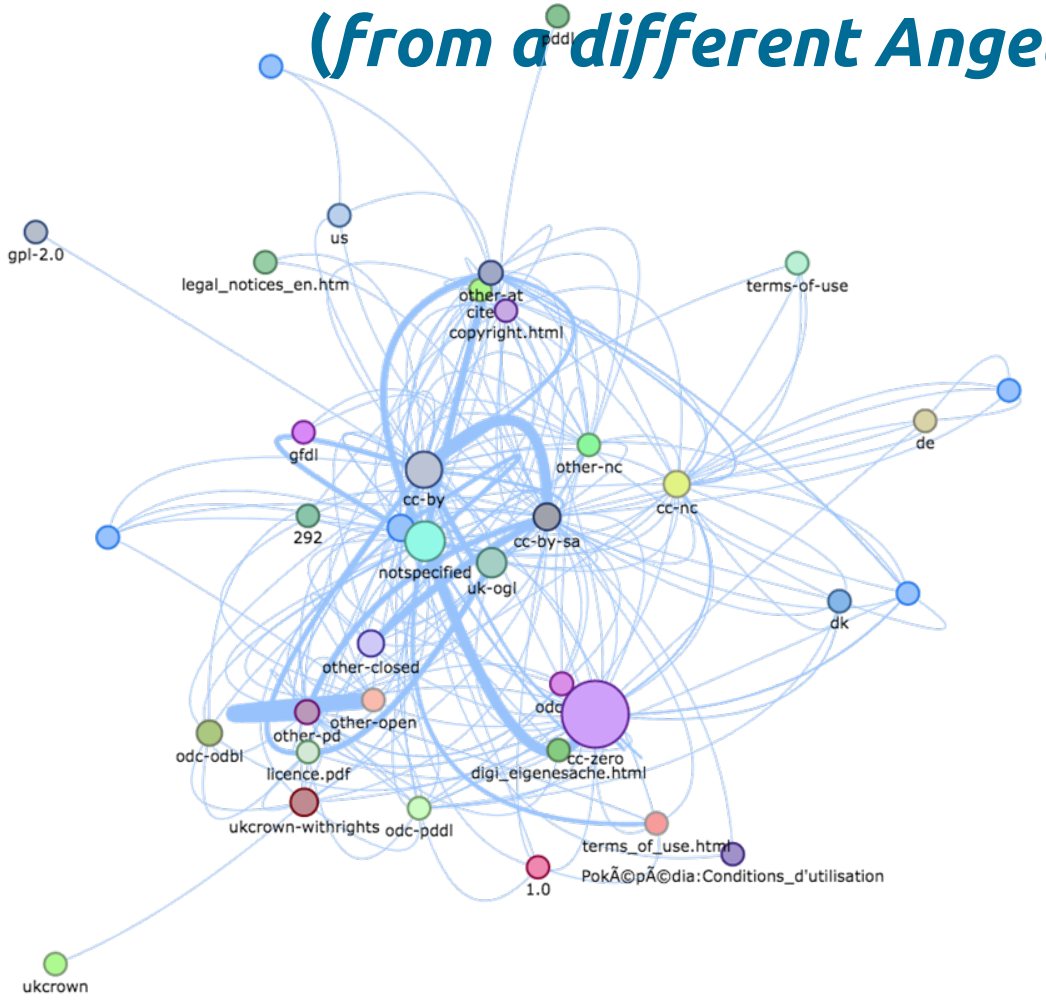
The Famous LOD Cloud

(from a different Angel)



The Famous LOD Cloud

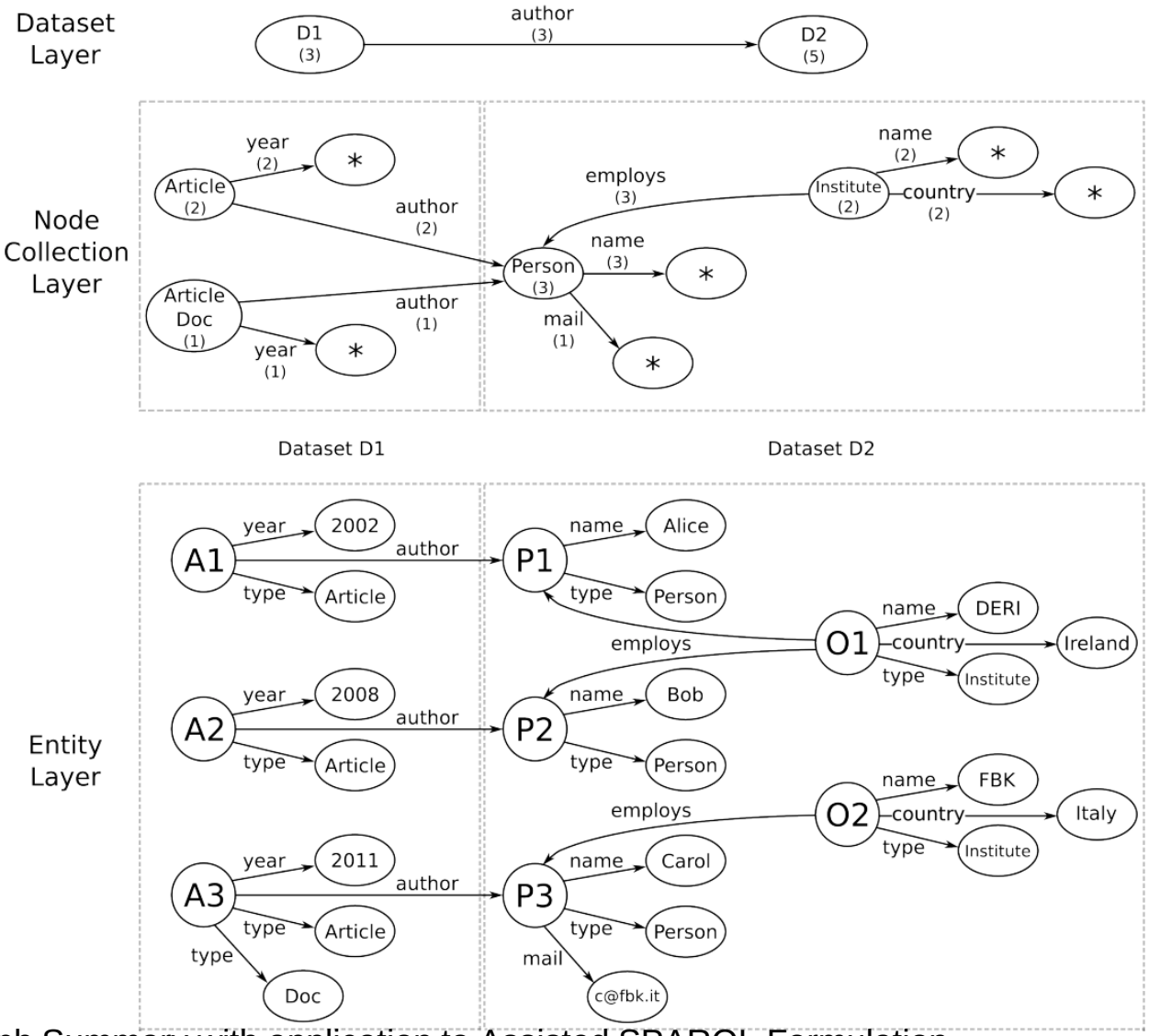
(from a different Angel)



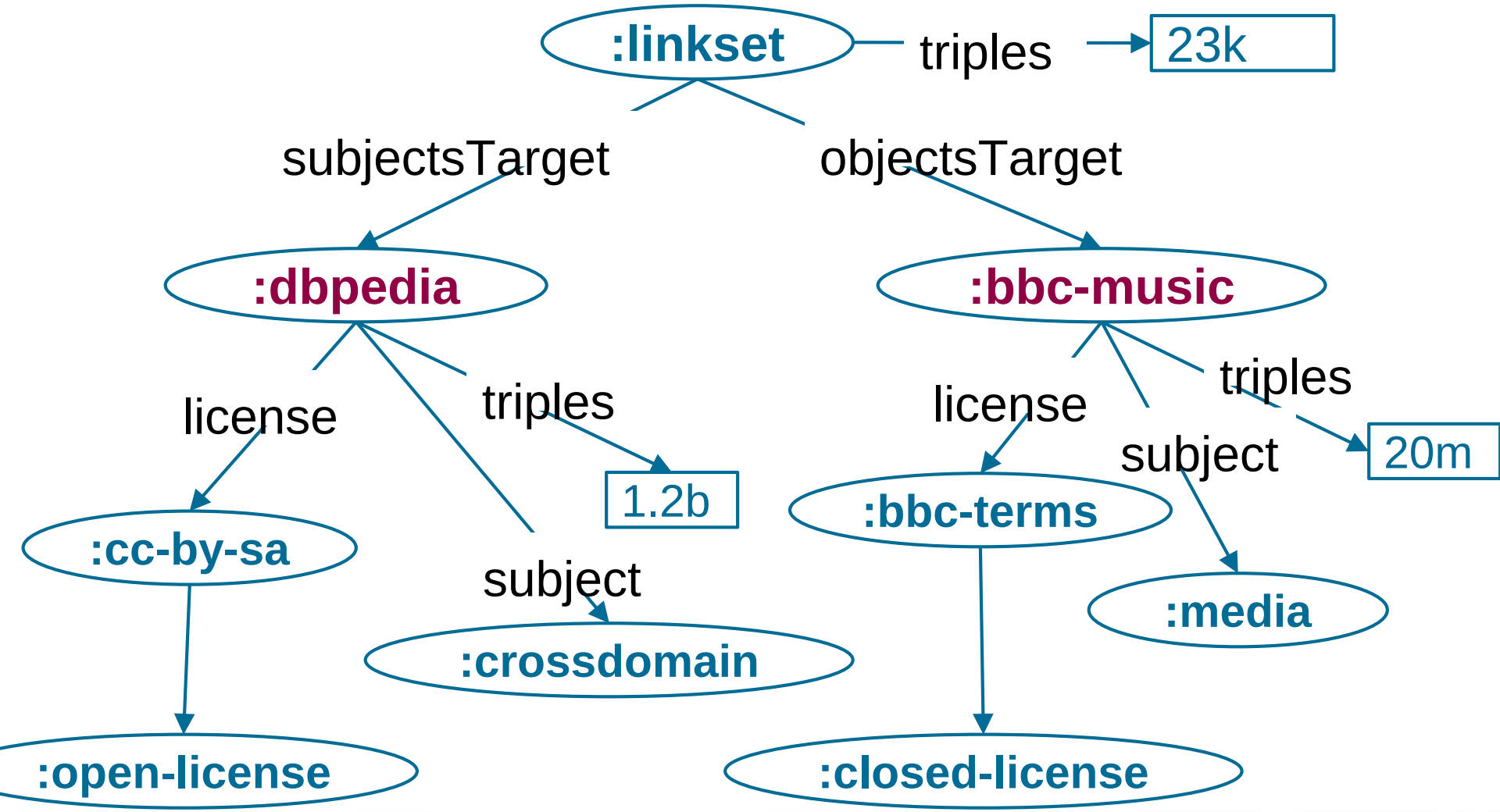
Graph Aggregation

Condenses a large graph into a structurally similar but smaller graph by collapsing vertices and edges

Graph Aggregation - Schema Discovery



Graph Aggregation - Requirements



Graph Aggregation Methods

1. **Custom Code**
   error prone, time, efficiency…

2. **SPARQL**
   error prone, time, efficiency…

3. **Graph Databases**
   expressivity, optimisation…

4. **Gagg, a first-class operator**

Graph Aggregation Methods

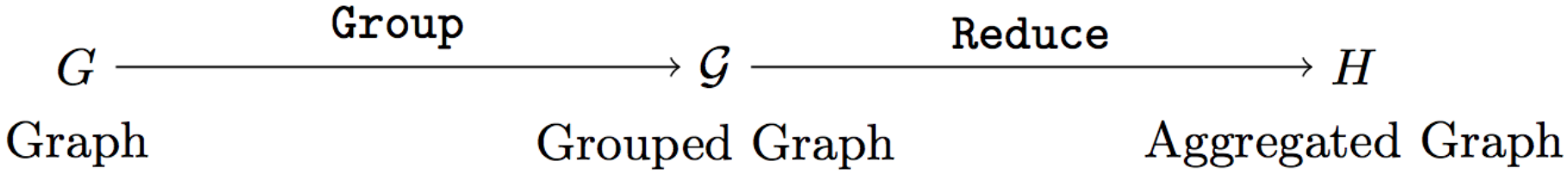
1. **Custom Code**
   error prone, time, efficiency…

2. **SPARQL**
   error prone, time, efficiency…

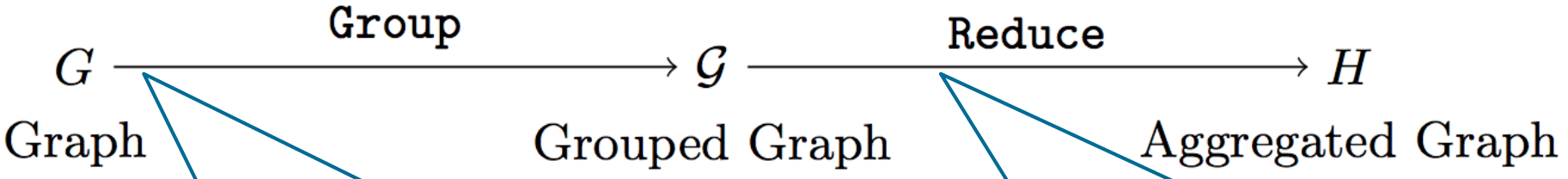
3. **Graph Databases**
   expressivity, optimisation…

4. **Gagg, a first-class operator**
   Operational Semantics
   In-memory evaluation algorithm
   Experimental evaluation

Gagg: Two-steps Aggregation



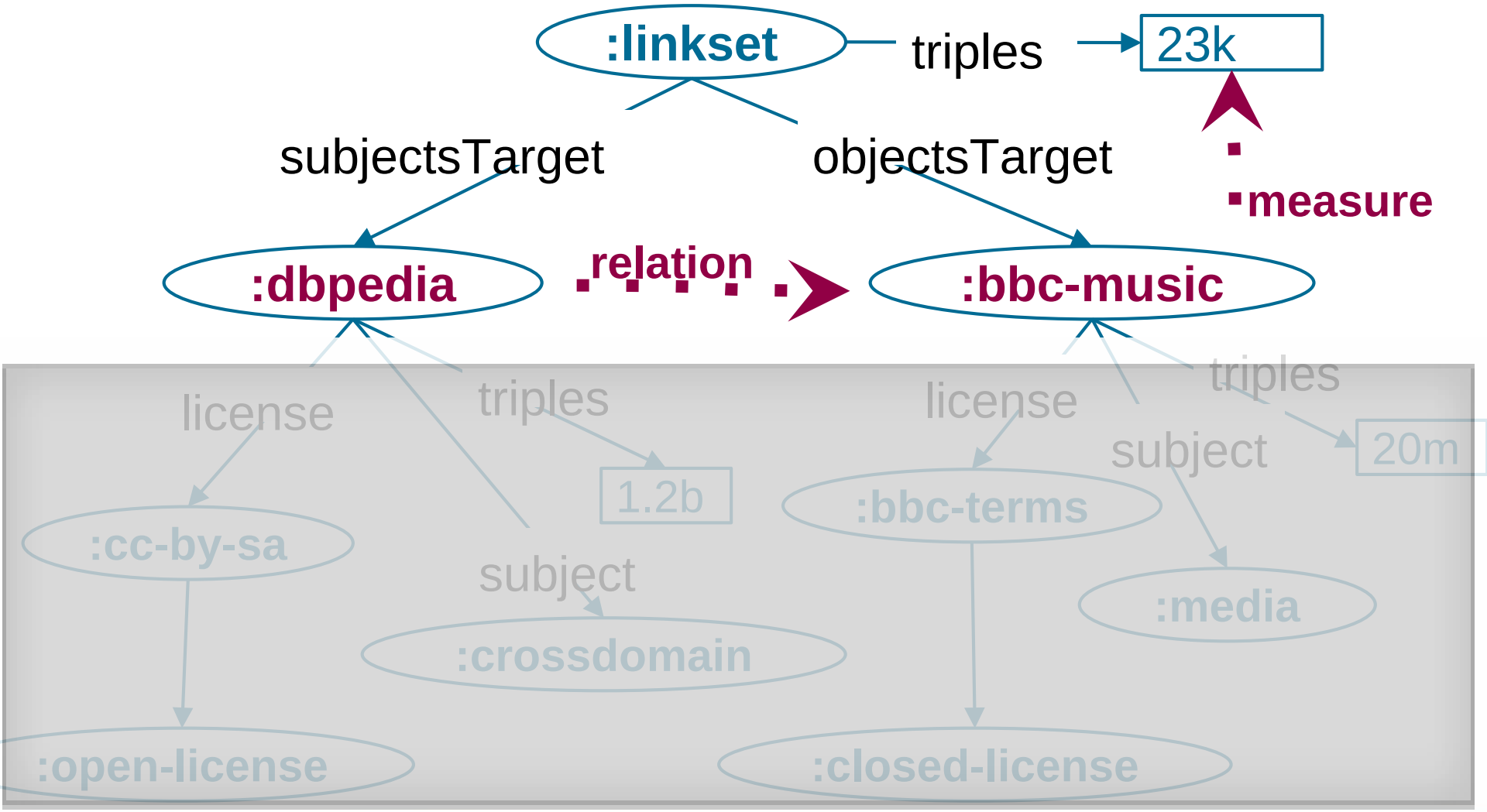
Gagg: Two-steps Aggregation



- Relation & measure
- Subject dimension(s) & measure
- Object dimension(s) & measure

Uses aggregation functions and a template similar to CONSTRUCT queries

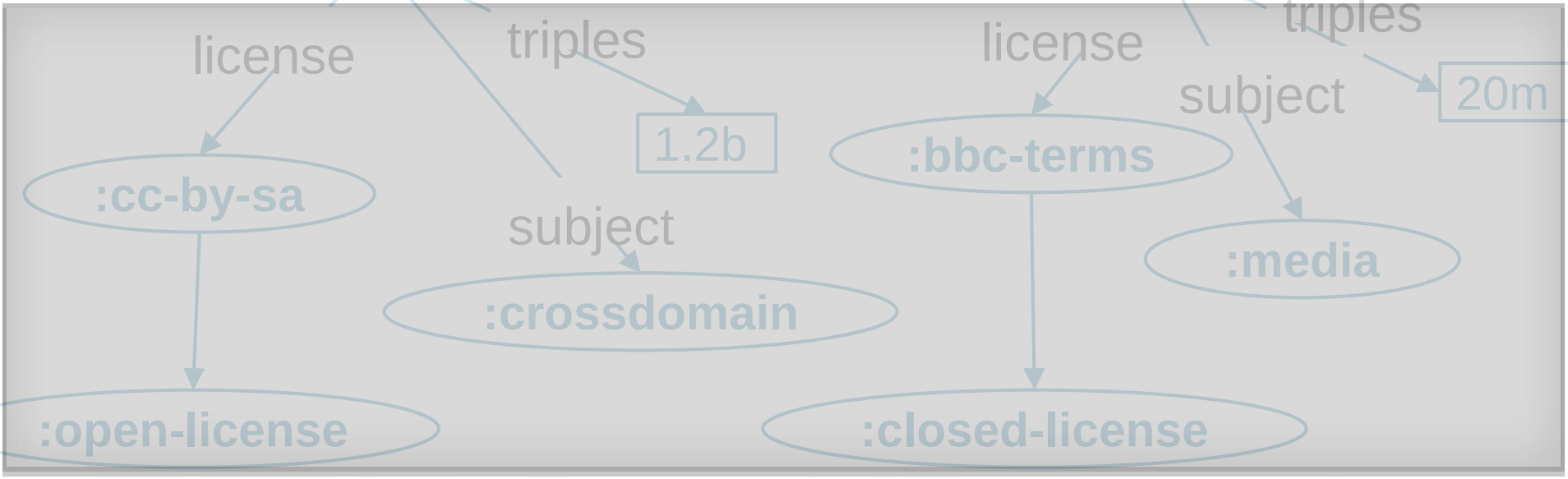
Graph Aggregation - Requirements



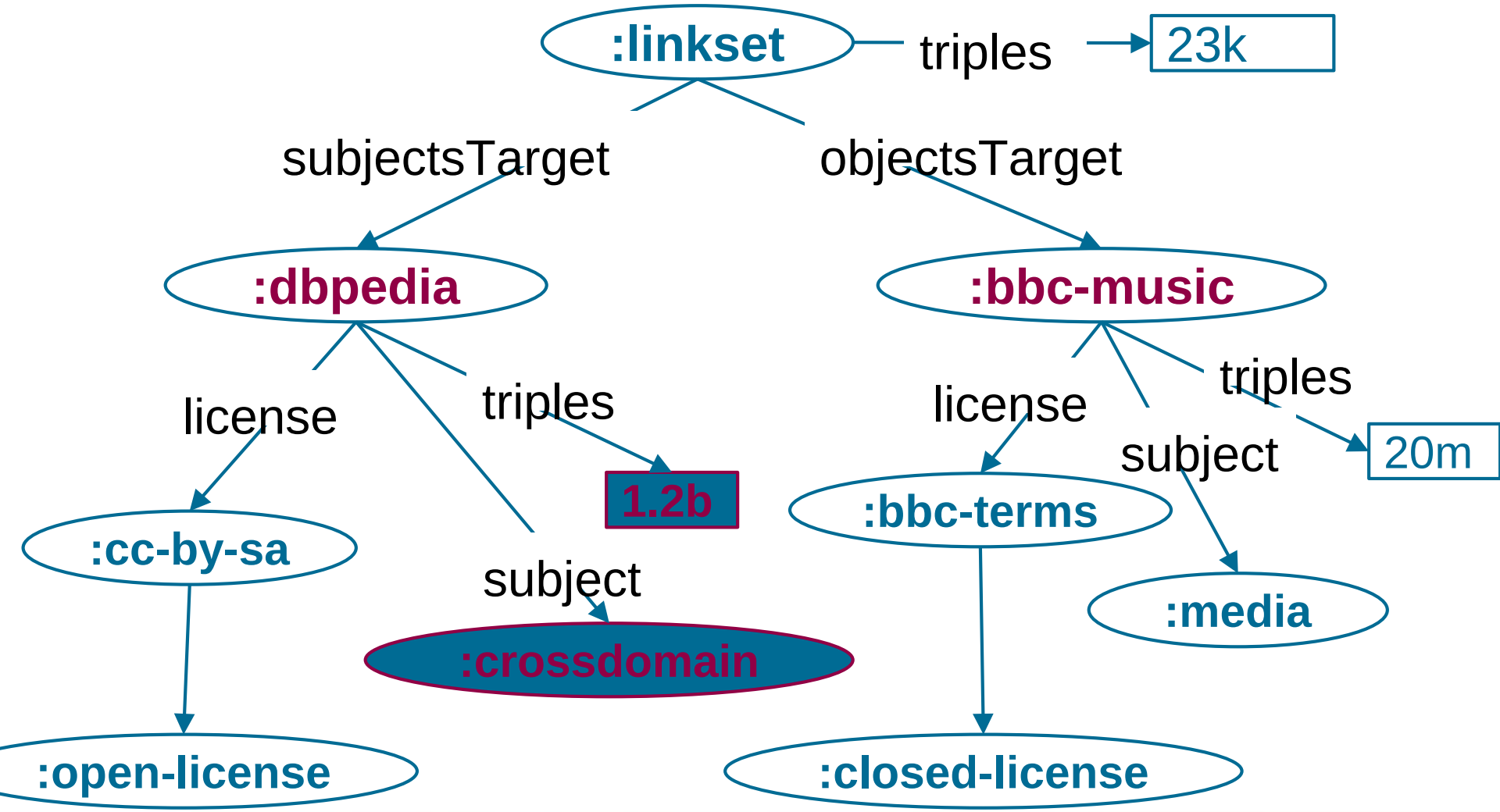
Graph Aggregation - Requirements

```
?l a void:LinkSet ;
   void:subjectsTarget ?s ;
   void:objectsTarget ?o ;
   void:triples ?m .
```

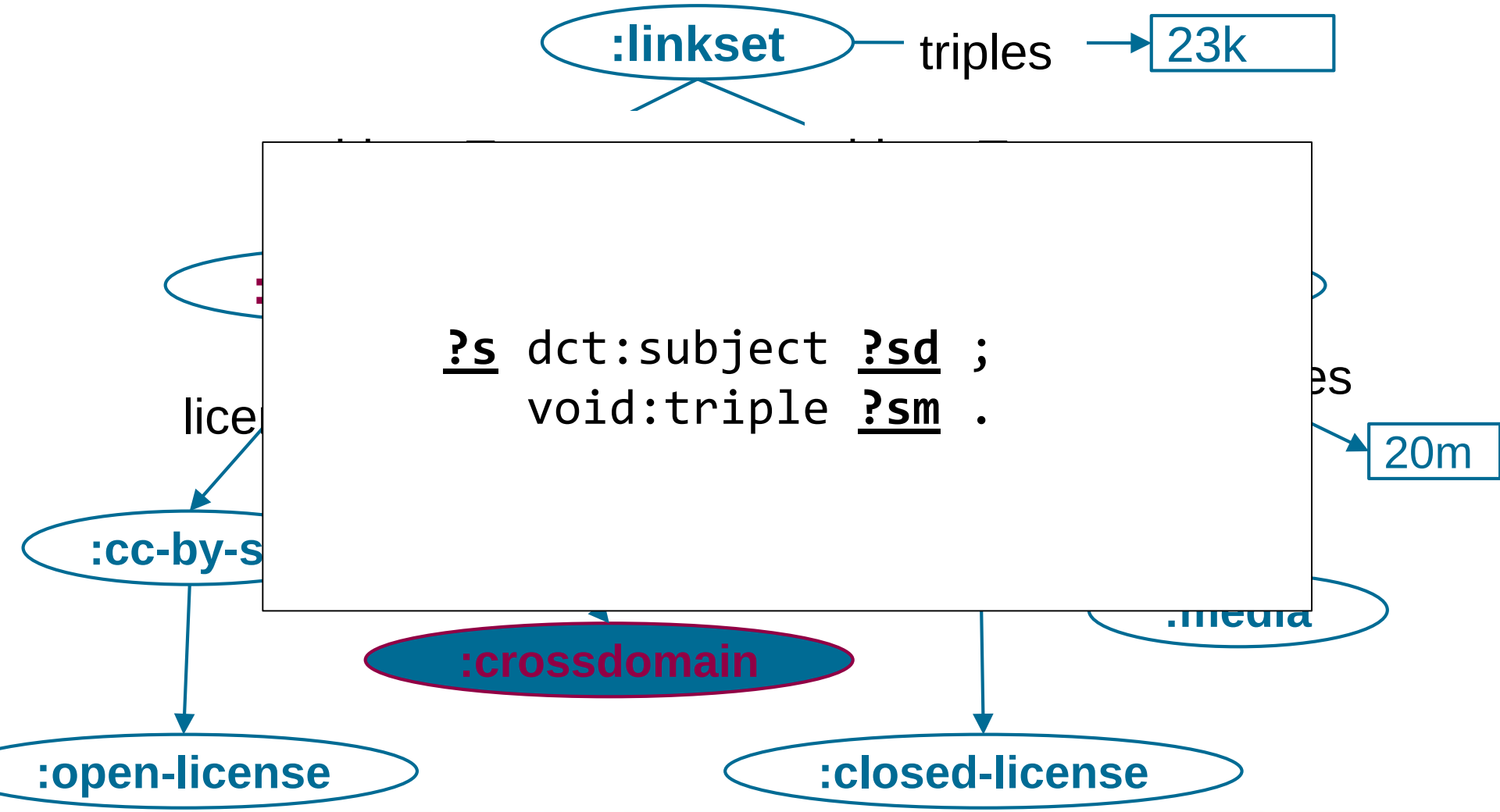
asure



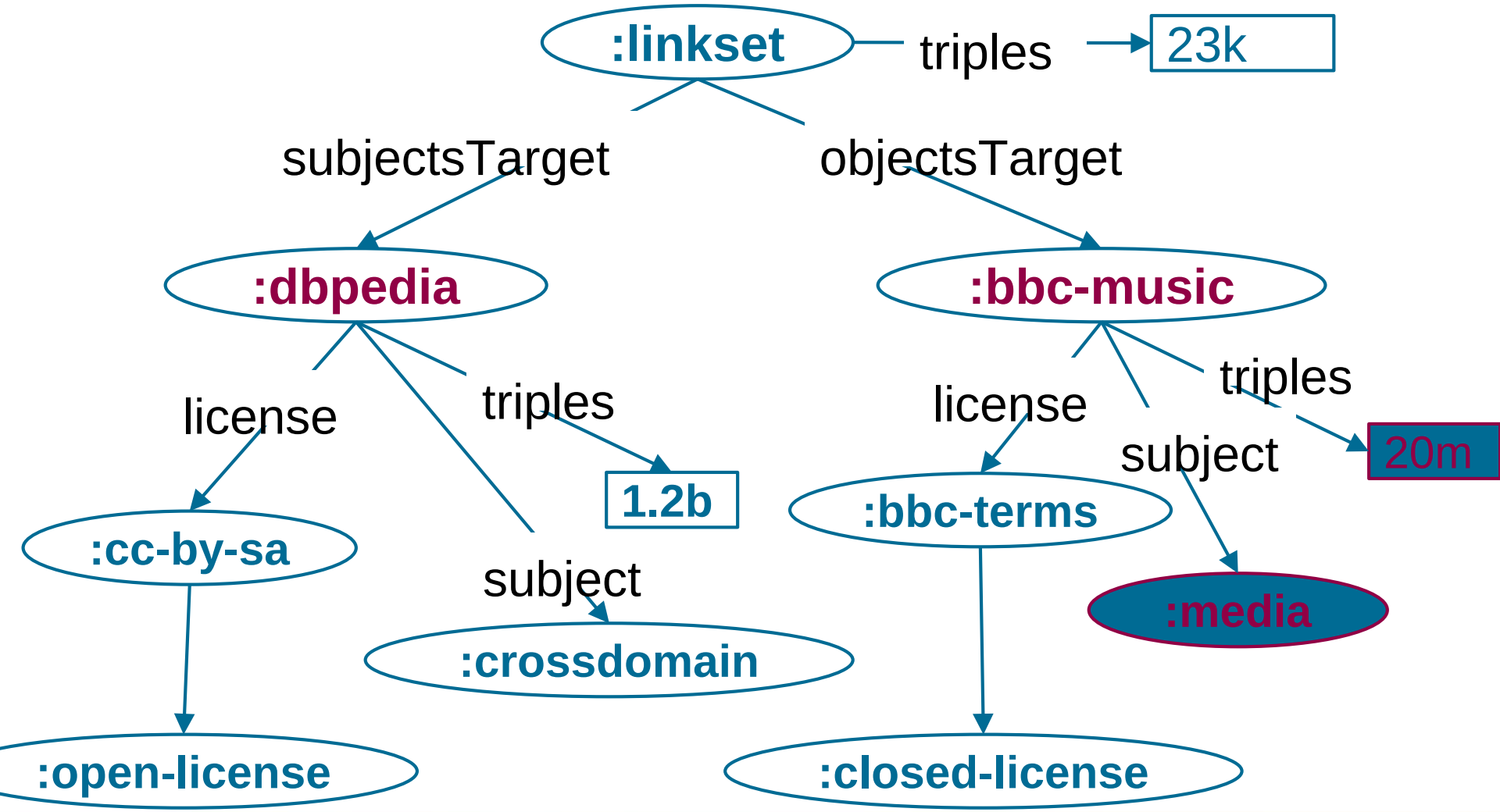
Graph Aggregation - Requirements



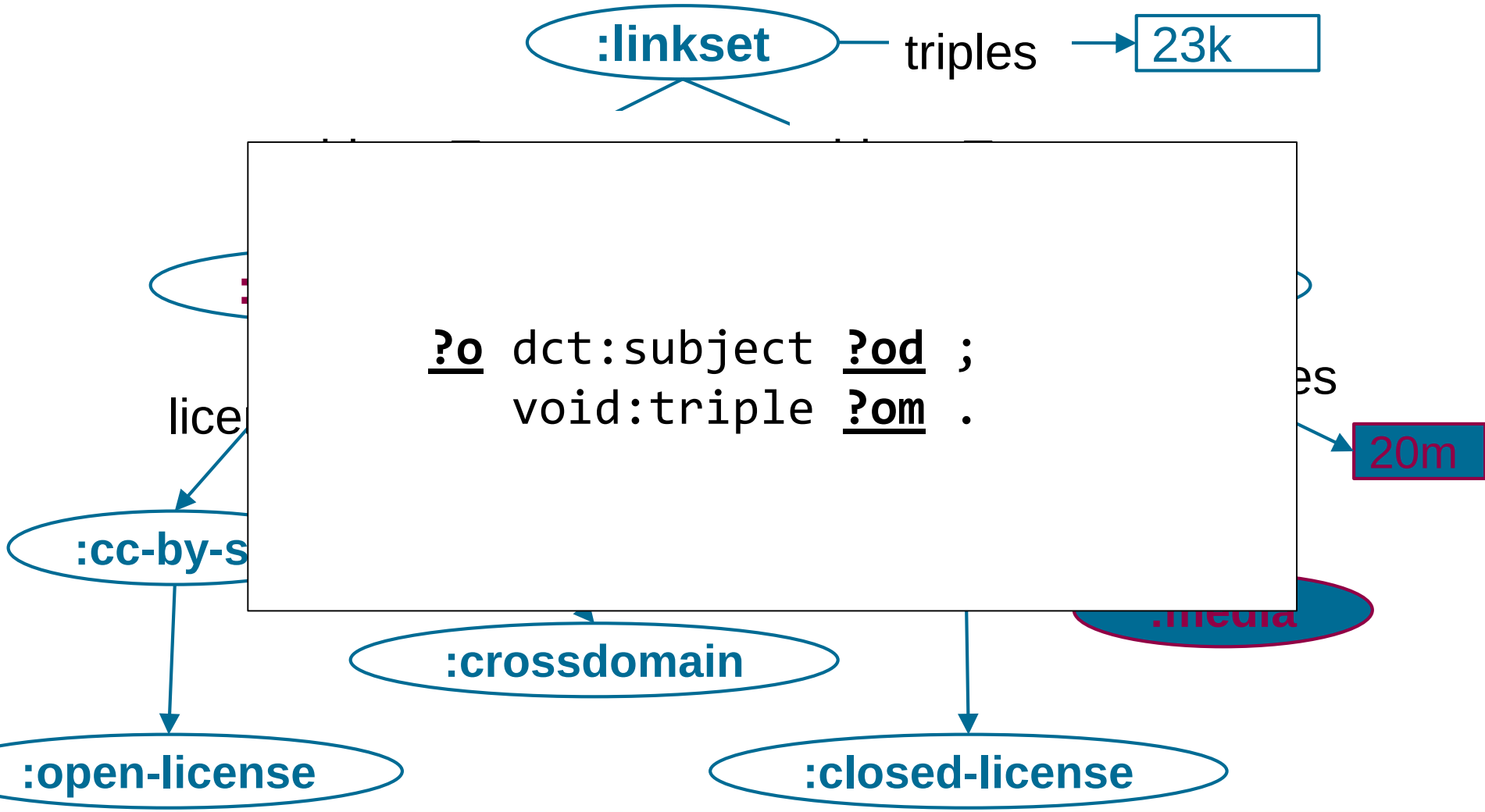
Graph Aggregation - Requirements



Graph Aggregation - Requirements



Graph Aggregation - Requirements



Graph Aggregation - Definition

Q=(D,M,E,N,R,f)

D: subject dimensions

?x	?sd

M: subject measure

?x	?sm

E: object dimensions

?y	?od

N: object measure

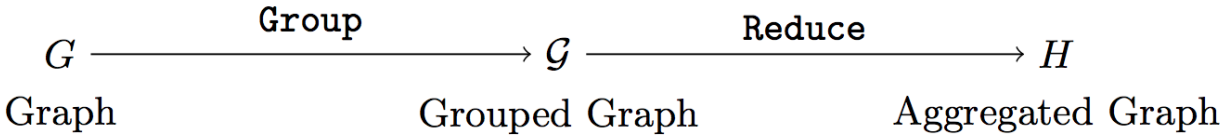
?y	?om

R: relation query

?x	?p	?y	?m

f: reduce function

Graph Aggregation - Grouped Graph



10k	33k	…

3k	1.2M	…

Graph Aggregation - Evaluation

- **Build a binding table**

?x	?p	?y	?m	?sd	?sm	?od	?om

- **Build the Grouped Graph**

  O(|B|) algorithm where B is the size of the binding table

- **Apply the reduction function**

Graph Aggregation - Experiment Setup

- **Extended In-memory Apache Jena**
  using SSE

- **BSBM and SP^{2}B**

- **Type Summary and Bibliometrics**

Graph Aggregation - Experiment Results

Data size (#triples)	fullSPARQL	3SPARQLs	reduced	Gagg
5k	0.08	0.06	0.01	0.03
190k	9.84	1.25	0.42	0.55
370k	31.88	2.82	1.00	1.13
1.8M	454.07	13.48	4.37	5.61

Type Summary on BSBM data

Conclusion

Graph aggregation as a first-class operator

- Easier for users to express
- Easier for engines to support and optimise
- Easier for further research and study

Further Questions:

- Syntax and effect on SPARQL
- Distributed implementation

```
PREFIX : <http://example.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>

CONSTRUCT {
    _:b0 a ?t1; :count COUNT(?sub.s) .
    _:b1 a ?t2; :count COUNT(?obj.o).
    _:b2 a rdf:Statement; rdf:predicate ?p; rdf:subject
   _:b0; rdf:object _:b1; :count ?prop_count
} WHERE {
    GRAPH_AGGREGATION  {
        ?s ?p ?o
        {?s a ?t1} GROUP BY ?t1 AS ?sub
        {?o a ?t2} GROUP BY ?t2 AS ?obj
    }
}
```

```
  SELECT ?t1 ?count_s ?subId ?t2 ?count_o ?objId ?p (COUNT(*) AS
?rel_count){
    {
      SELECT ?t1 ?count_s ?subId ?t2 ?count_o ?objId ?p {
        ?s a ?t1 . ?s ?p ?o . ?o a ?t2 .
        {
          SELECT ?t1 ?subId (COUNT(DISTINCT ?s) AS ?count_s){
            ?s a ?t1 . ?s ?p ?o .?o a ?t2 .
            BIND (iri(CONCAT(str(?t1), "_s")) AS ?subId)
          } GROUP BY ?t1 ?subId
        }
        {
          SELECT ?t2 ?objId (COUNT(DISTINCT ?t2) AS ?count_o){
            ?s a ?t1 .
            ?s ?p ?o .
            ?o a ?t2 .
            BIND (iri(CONCAT(str(?t2), "_o")) AS ?objId)
          } GROUP BY ?t2 ?objId
        }
      }
    }
  } GROUP BY ?t1 ?count_s ?t2 ?count_o  ?subId ?objId
}
```