Realizing OBDA with data dependencies

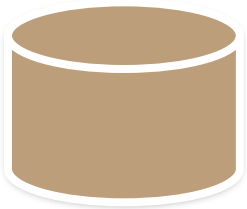
Mariano Rodríguez-Muro and Diego Calvanese
KRDB Research Group
Free University of Bozen-Bolzano

Semantic Technologies 2012
6-7 Sept. 2012, Sogndal Norway

ABOX CONSTRAINTS

Enforcing dependencies a.k.a completing the data

Enforcing dependencies



Manipulate the data

Inference materialization
Forward chaining (Sesame, OWLim, etc.)

Not the only way,
often, not the best way

Man subClass

$\equiv_{\mathcal{A}} Human$

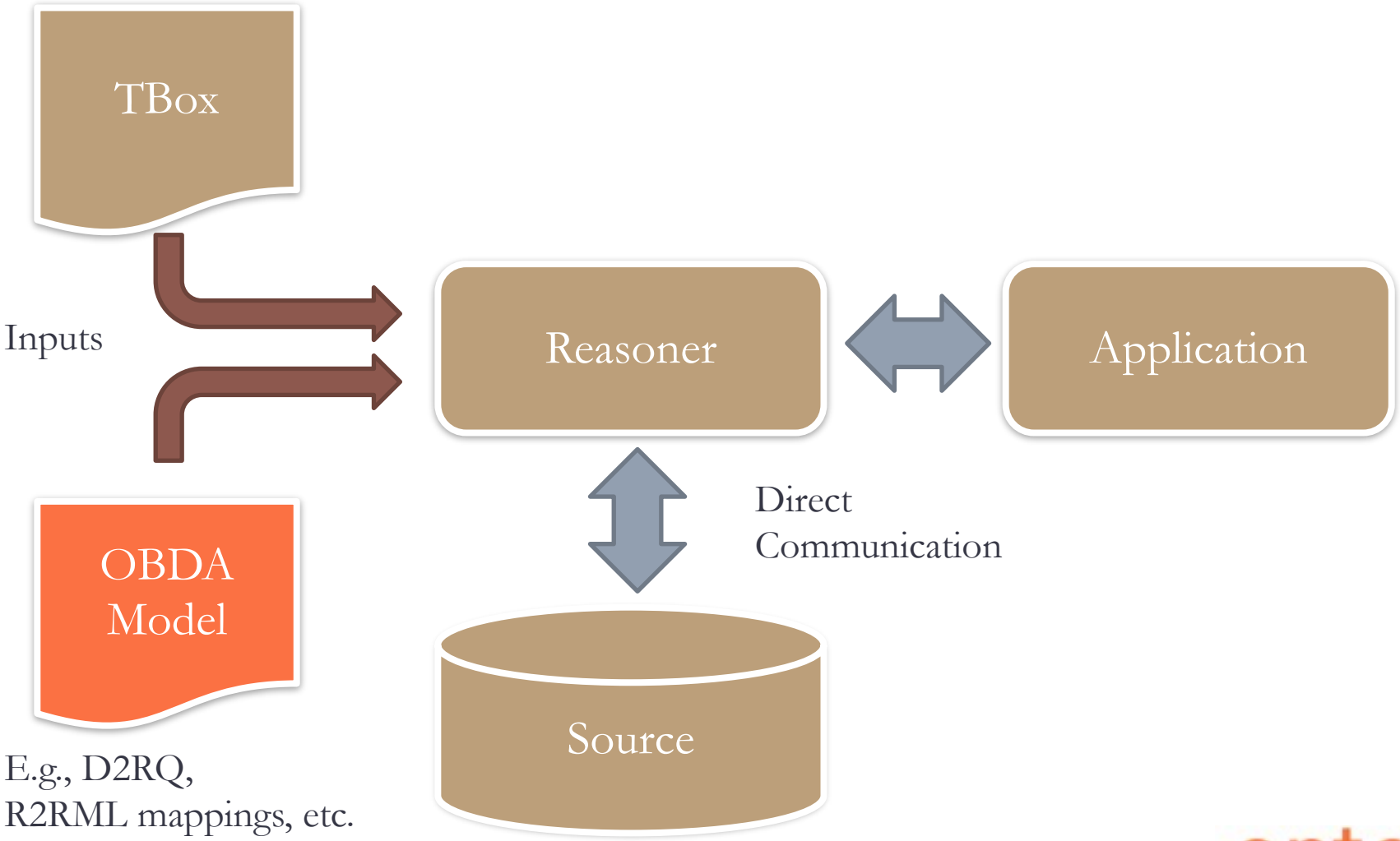
inverseOf(hasParent, hasChild) John hasParent Mary

Mary hasChild John

$hasParent \equiv_{\mathcal{A}} hasChild$

ontop

Case 1:Virtual RDF Graphs



ontop

T-Mappings

- Manipulate the mappings to enforce the semantics of T
- if $\mathcal{T} \models A \sqsubseteq B$ ensure that the mapping(s) for B include the data from the mappings for A

ontop

T-Mappings

- Manipulate the mappings to enforce the semantics of T
- if $\mathcal{T} \models A \sqsubseteq B$ ensure that the mapping(s) for B include the data from the mappings for A

SELECT id FROM t WHERE a = ‘a‘ $\rightsquigarrow$?id rdf:type A

SELECT id FROM t WHERE a = ‘b‘ $\rightsquigarrow$?id rdf:type B

SELECT id FROM t $\rightsquigarrow$?id rdf:type B

WHERE a = ‘b‘ OR a = ‘a’

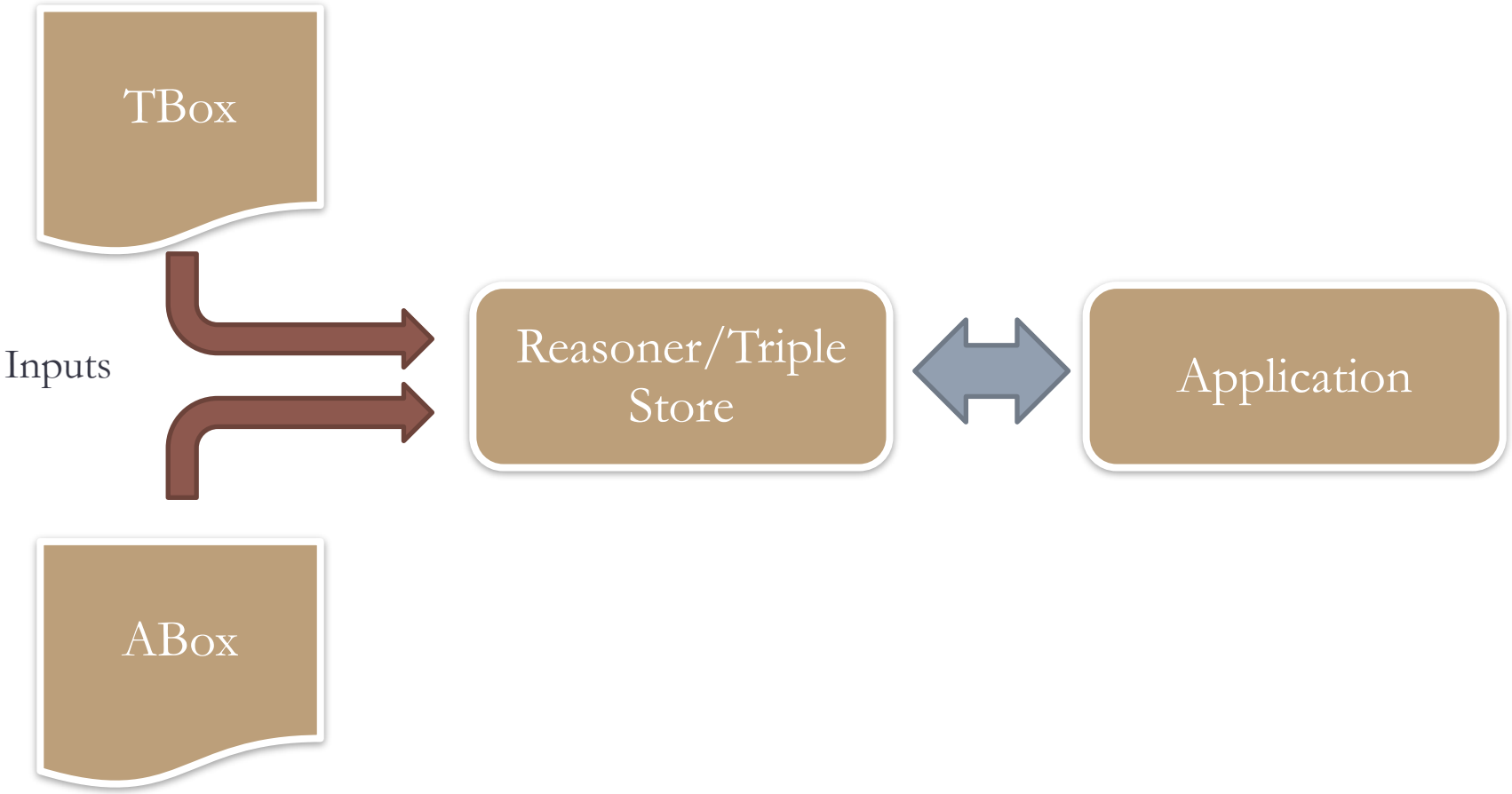
- Use ‘completed’ mappings for query translation
- Enough for OWL2QL except: $A \sqsubseteq \exists R$

ontop

T-Mappings

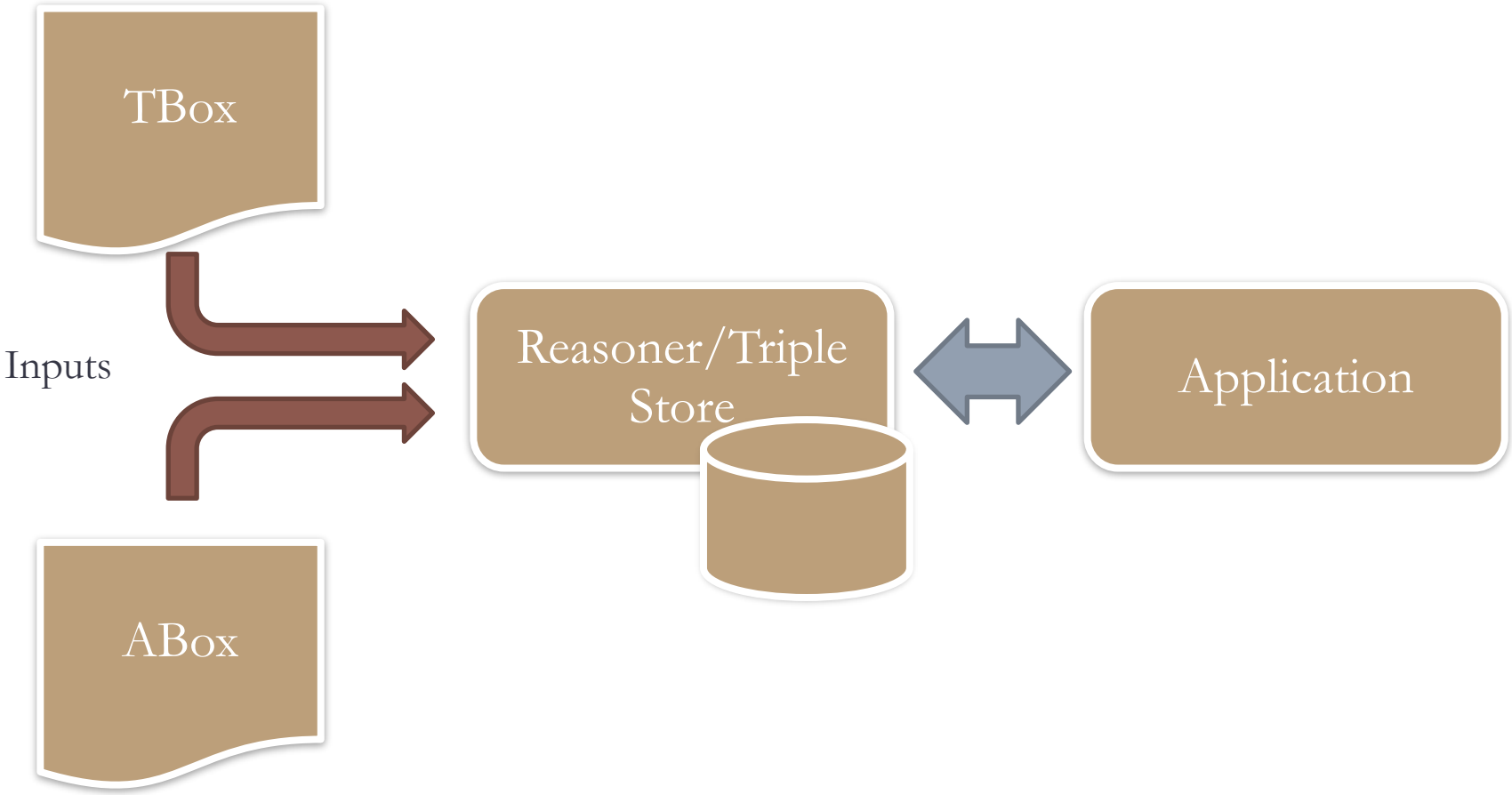
- Things to care about (for SQL data sources):
  - SQL structure (no sub-queries, no large SQL, 'natural SQL')
  - Redundant JOINs
  - Impedance mismatch: URI's and RDF/OWL datatypes, not SQL types
- Available tools:
  - Partial evaluation
  - Static analysis of SQL (for mappings) and SPARQL (for queries)
- Results in the next slides

Case II: Reasoners/Triple stores



ontop

Case II: Reasoners/Triple stores



ontop

Case II: Reasoners/Triple stores


```
SELECT ?x ?z WHERE { ?x foaf ?z}
```

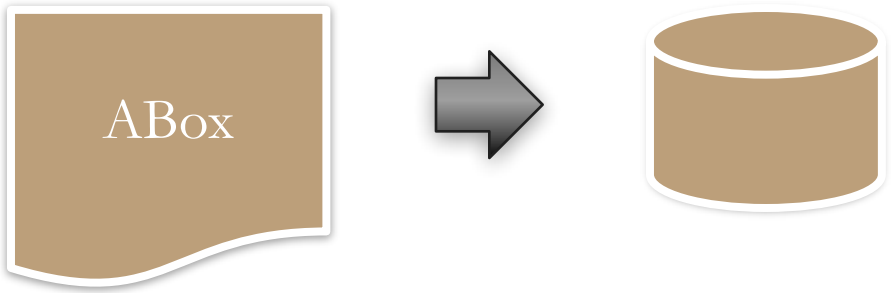
```
SELECT s as x, o as z
FROM triple WHERE p = 'foaf'
```

```
SELECT s as x, o as z
FROM triple WHERE p = 227
```

```
SELECT s as x, o as z
FROM foaf
```

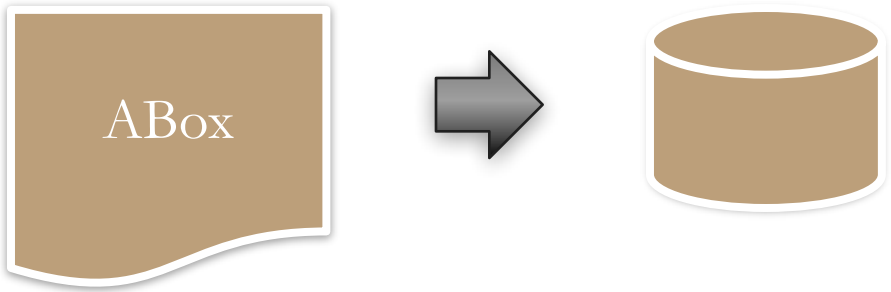
ontop

Creating semantic triple storage

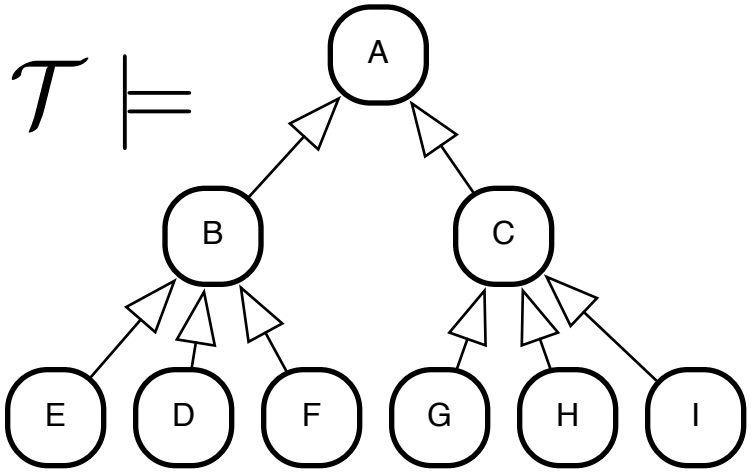


ontop

Creating semantic triple storage

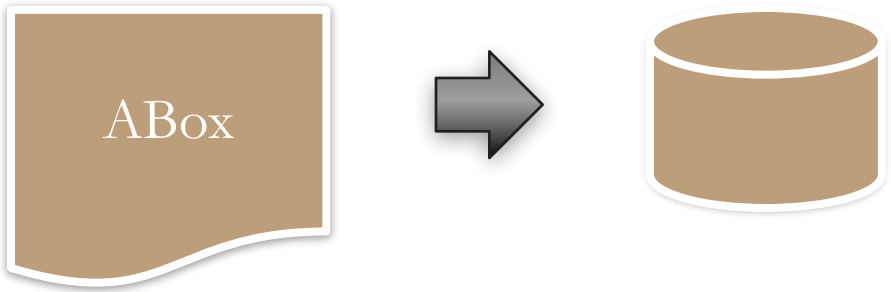


1. Encode the hierarchies of T in indexes

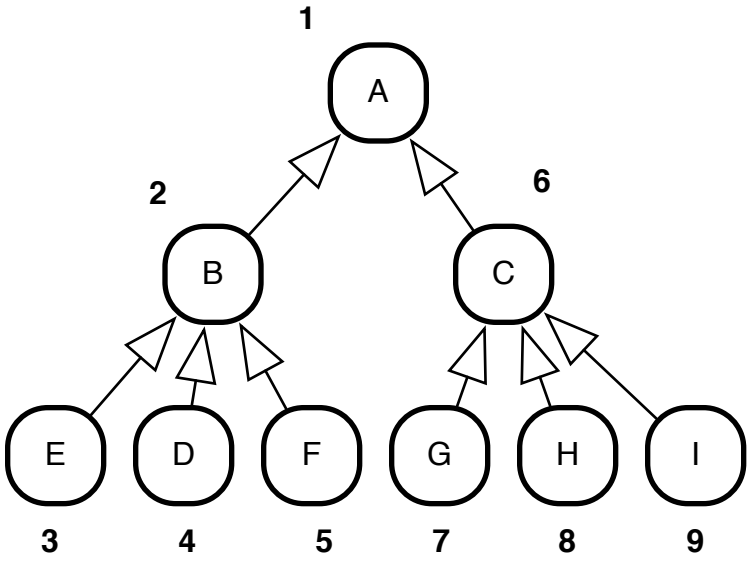


ontop

Creating semantic triple storage



1. Encode the hierarchies of T in **indexes**



ABox:

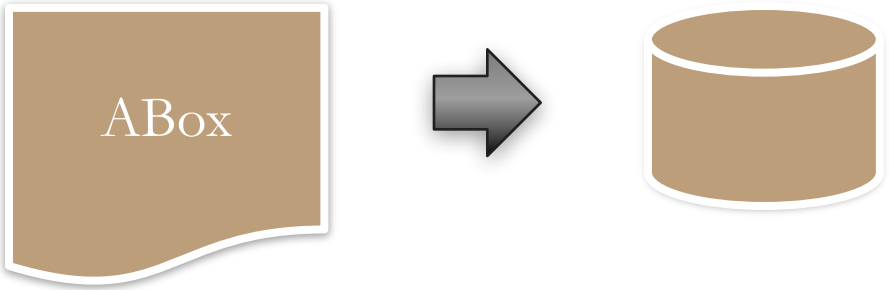
```
d rdf:type D
h rdf:type H
```

Insert into your data structure using those indexes

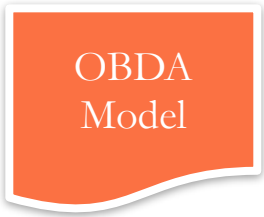
s	p	o
d	rdf:type	4
h	rdf:type	8

ontop

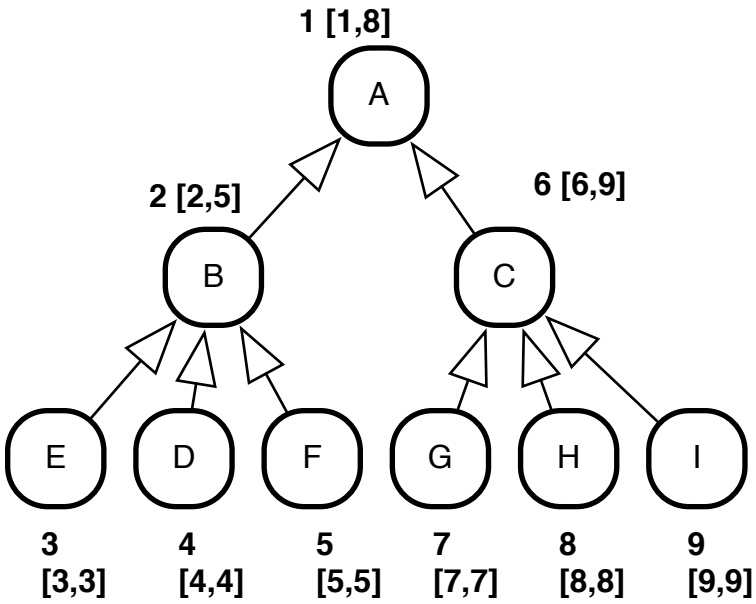
Creating semantic triple storage



4. Create mappings using those intervals



3. Define **intervals** to retrieve your data




```
?s rdf:type A
```

SELECT s FROM t WHERE IDX >= 1 AND
IDX <= 8

5. Complement with **T-mappings** to cover domain, range and inverse inferences

ontop

EXPERIMENTS-BENCHS

Query rewriting with Semantic Indexes

Experiments - LUBM

- LUBM3X-Lite - Larger LUBM in OWL 2 QL fragment
- 14 original LUBM queries
- Consumer hardware (Macbook Pro)
- 50 Universities
  - 6,863,227 data triples (ABox assertions)
- Systems tested:
  - Quest - (Over PostgreSQL)
  - Sesame
  - Requiem
  - OwlGres
  - Presto
- Measures: Rewriting size/complexity, Loading time, Execution

Experiments - LUBM - Rewriting

Q	Qs(b)	Qs(f)	Rq(n)	Rq(g)	Pr(d)	Pr(c)
1	9	**1**	42	3	4	3
2	17496	**1**	17496	972	22	729
3	153	**1**	150	3	4	3
4	162	**1**	1356	27	28	27
5	1476	**1**	5322	69	67	66
6	12	**1**	12	12	13	12
7	1620	**1**	1512	108	19	108
8	3888	**1**	1296	1296	31	2916
9	-	**1**	-	-	22	324
10	36	**1**	111	12	13	12
11	18	**1**	18	18	10	18
12	162	**1**	162	162	16	162
13	1845	**1**	1560	15	16	15
14	3	**1**	3	3	4	3

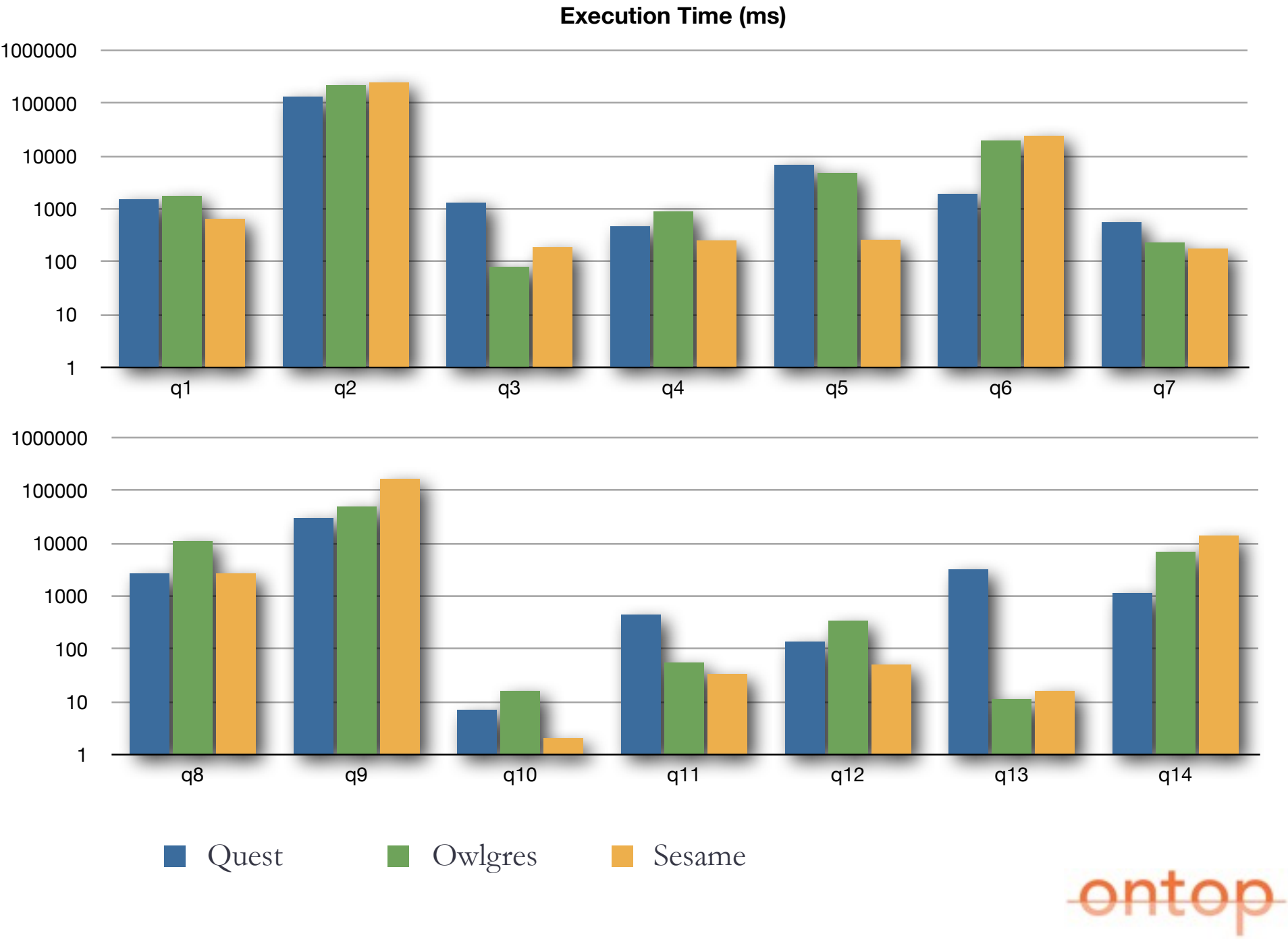
Table 2: Reformulation size (# CQs/Rules), smallest in bold

ontop

Experiments - LUBM - Loading

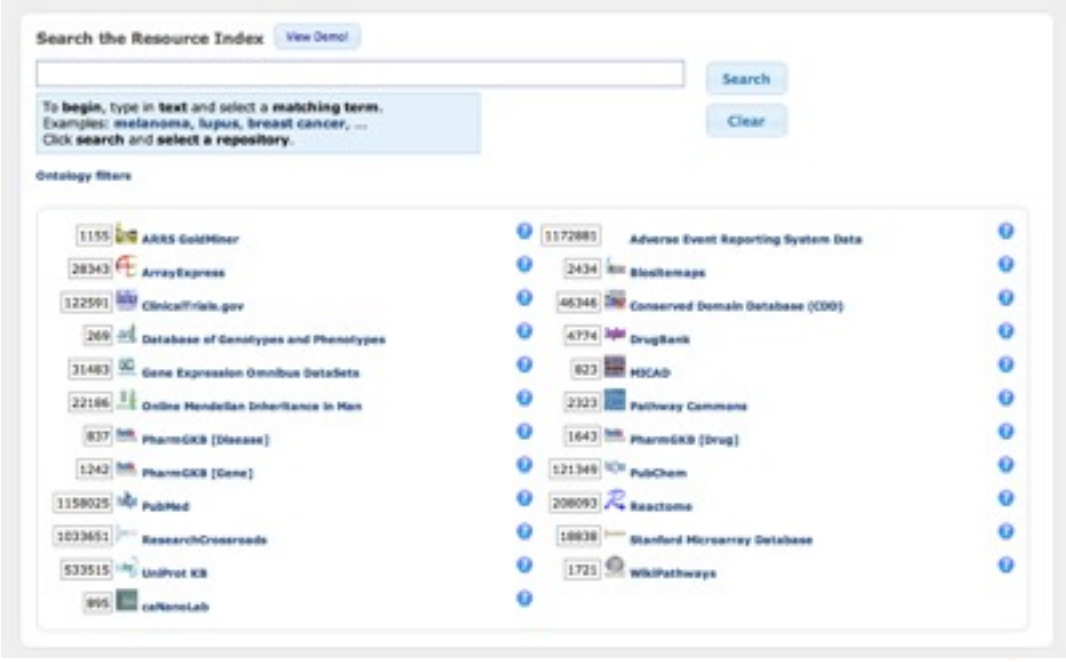
- Time required to load and process all triples
  - Jena: 5 hrs timeout
  - Sesame: 2 hrs 20 m
  - Owlgres: 1 hr 25 m
  - Quest: 104 s

ontop


Execution Time (ms)
1000000
100000
10000
1000
100
10
1
q1
q2
q3
q4
q5
q6
q7
1000000
100000
10000
1000
100
10
1
q8
q9
q10
q11
q12
q13
q14
Quest
Owlgres
Sesame

Experiments - Resource Index

- Experimentation using Stanford's "**Resource Index**"
- **Semantic search** over **annotated documents**
- 200 ontologies from Bio-portal (only hierarchies 200k concepts, millions of subClassOf)



Experiments - Resource Index

- Current system uses **forward chaining**
  - Naive chase: 7 days
  - Optimized chase: 40 mins
  - Cost 16 GB + 70 GB chase data
  - Split second responses
- **Pure query rewriting** approaches

- **Semantic index**-based rewriting
  - DAG computation and indexing 5 mins
  - Cost: 16 GB
  - Single queries, split second responses

ontop

```
SELECT ?x WHERE {
   ?x a DNA_Repair_Gene.
   ?x a Antigen_Gene.
   ?x a Cancer_Gene
   }
```

- Materialized, 3s execution time)
- Pure query rewriting: not-feasible (467k UCQs or 250 datalog rules)
- Semantic Index based rewriting, 3 s

```
SELECT DISTINCT r0.element_id as element_id
FROM RI.CT_ANN r0
  JOIN RI.CT_ANN r1 ON r0.element_id = r1.element_id
  JOIN RI.CT_ANN r2 ON r1.element_id = r2.element_id
WHERE
  ((r0.idx >= 1783559 AND r0.idx <= 1783657)) AND
  ((r1.idx >= 1782996 AND r1.idx <= 1783029)) AND
  ((r2.idx >= 1783115 AND r2.idx <= 1783253));
```

ontop

EXPERIMENTS-BENCHS

Virtual RDF graphs with SPARQL

Experiments - Fish Deli (virtual graph)

- By Parsia et. al [SSWS12]
- FishBase: Fish species database, currently deployed. ETL based.
- FishMark:
  - FishBank data set 20,186,775 triples
  - 22 queries (based on queries from user logs, building a use scenario)
- Systems involved:
  - Virtuoso
  - MySQL 5.5
  - Quest 1.7 alpha (July 12)
- Measure: Queries per second (each query 100 times)
- Machine: Consumer PC (Mac mini, 2.6 GHz + 16 GB Ram)

ID	Query name	Virtuoso	Quest	MySQL
1	CommonName	132	850	1262
2	SpeciesPage	1	552	840
3	Genus	167	753	1025
4	Species	192	870	1249
5	FamilyInformation	17	572	840
6	FamilyAllfish	141	704	1113
7	FamilyNominalSpecies	161	773	1060
8	FamilyListOfPictures	50	578	742
9	CollaboratorPage	27		824
10	PicturePage	68	711	1166
11	CAllFish	24	316	629
12	CSpeciesInformation	7	598	1009
13	CFreshwater	6	201	212
14	CIntroduced	9	303	479
15	CEndemic	13	656	985
16	CReefAssociated	4	266	378
17	CPelagic	7	337	532
18	CGameFish	9	580	845
19	CCommercial	12	556	748
20	CUsedForAquaculture	12	779	1229
21	CPotentialAquaculture	34	694	957
22	CAquariumTrade	66	815	1251

Table 4. Query results of the 22 queries in qps (queries per second)

```
SELECT *  WHERE {
            $x a :Species .
           $x :genus "Alosa"^^xsd:string .
           $x :species "alosa"^^xsd:string .
           $x :isFoundInCountry ?id .
           $countryID :countryName "algeria"^^xsd:string .
           $countryID :factbook ?Factbook .
            $countryID :refersToCountry ?id .
            $id :status ?occurence .
           $id :inFreshwater ?freshwater .
           $id :inBrackish ?brackish .
           $id :inSaltwater ?marine .
           $id :importance ?importance .
           $id :abundance ?abundance .
           $id :abundanceRef ?abundanceRef .
           $id :regulations ?regulations .
           $id :regulationsRef ?regulationsRef .
           $id :importanceRef ?importanceRef .
           $id :comments ?comments .
}
```

ontop

```
SELECT
   4 AS "freshwaterQuestType", '' AS "freshwaterLang", QVIEW1.`Freshwater` AS `freshwater`,
   4 AS "marineQuestType", '' AS "marineLang", QVIEW1.`Saltwater` AS `marine`,
   4 AS "brackishQuestType", '' AS "brackishLang", QVIEW1.`Brackish` AS `brackish`,
  ...
   3 AS "commentsQuestType", '' AS "commentsLang", QVIEW1.`Comments` AS `comments`
FROM
    `species` QVIEW0,
    `country` QVIEW1,
    `countref` QVIEW2
WHERE
    (QVIEW1.`C_Code` = QVIEW2.`C_Code`)
     AND (QVIEW0.`SpecCode` = QVIEW1.`SpecCode`)
     AND (QVIEW0.`Genus` = 'Alosa')
     AND (QVIEW0.`Species` = 'alosa')
     AND (QVIEW2.`PAESE` = 'algeria')
```

ontop

CONCLUSIONS

Conclusions

- Introduced **ABox dependencies** to describe the structure of data
- Showed how to **optimize TBoxes w.r.t. dependencies**,
- Introduced "**Semantic Index**" repositories
- Introduced "**T-Mappings**" for mapping processing
- Not detailed:
  - How to translate from SPARQL-SQL for query rewriting
  - How to generate efficient SQL (partial evaluation)
  - How to optimize efficiently with static analysis
  - How to deal with impedance mismatch

ontop

Conclusions

- Query answering by query rewriting can be done efficiently
- Current prototypes outperform well stablished commercial systems
  - See **-ontop-** and **Quest**
    - **http://obda.inf.unibz.it/protege-plugin/**
- Further developments in: Optique™

ontop


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A
0x6339392C,
0x7363682E.
I HATE YOU.

THANK YOU