Data Models Matter Less Than You Think

Peter Boncz (CWI and VU University)

Keynote Graph-TA (March 4, 2016)

Keynote Statements

(1) Even if data models seem to be very different,
the techniques to manage data are common among them

(at least from a database architect perspective)

(2) some datasets often assumed to belong to very different models are structurally very similar (RDF, graph, relational)

History

- 1970-
  Relational Data Model, SQL Query Language, Entity Relationship Modeling

- 1980-
  Object Oriented Data models, OQL query language, UML
  - Object-Relational

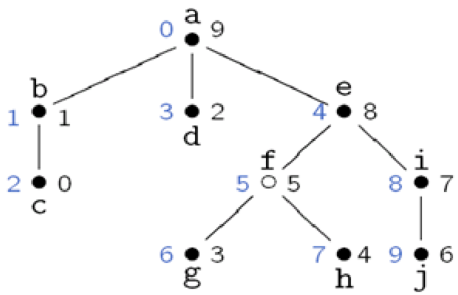
- 1990-
  XML, XPath & XQuery query languages, XML Schema
  - JSON

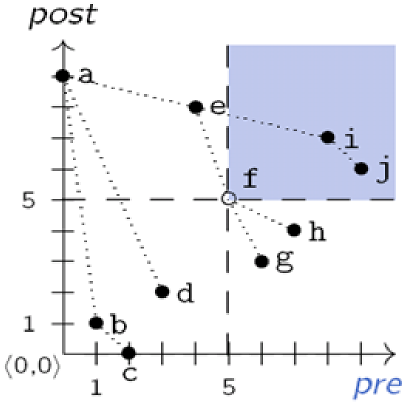
- 2000-
  RDF, SPARQL query Languages, Ontologies, OWL
  - Graph Data Models, Cypher query language

11 years ago

Tree Encoding: XPath Accellerator

Node-based relational encoding of XQuery's data model





Pre	Size	Level
0	9	0
1	1	1
2	0	2
3	0	1
4	5	1
5	2	2
6	0	3
7	0	3
8	1	2
9	0	3

pre + size – level = post

Lessons

- a new data model does **not imply** a necessity for “everything new” (storage, compression, query optimization, execution)
  - Virtuoso SPARQL → a SQL system
  - MonetDB XQuery → XPath on top of relational algebra

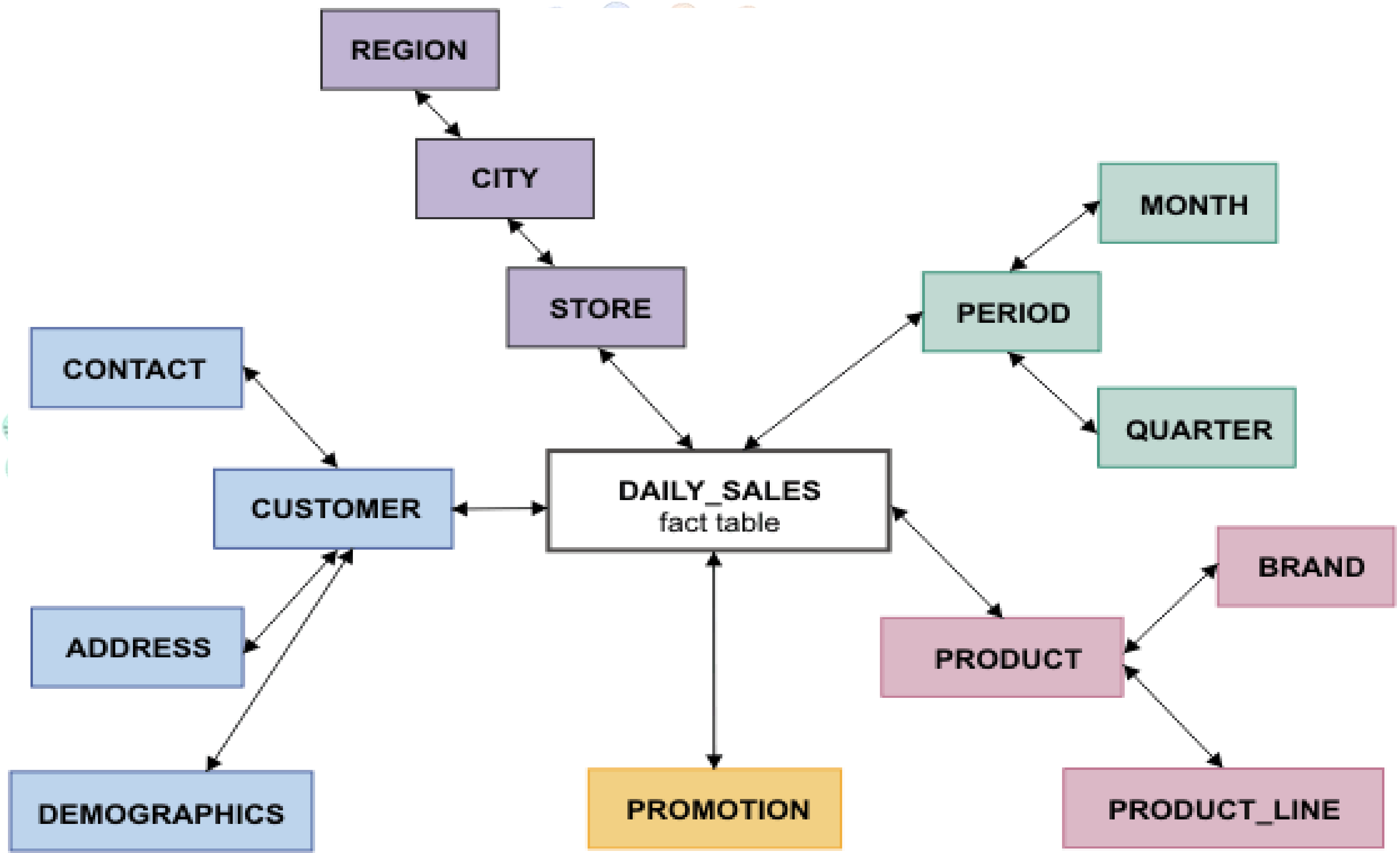
  “pointer based navigation is more efficient than relational join?” NO!

  pointer swizzling de/serialization, join index, row-IDs

  graph navigation = relational join = graph navigation = relational join = …

- proven techniques used and often invented in relational data management systems are not by themselves “relational”. They are **data management** techniques, widely applicable (not to be dismissed).
  - relational hash-join? B-trees? Bloom-filters?
  - relational dynamic-programming bottom-up enumeration?
  - relational query algebra?

Are LOD Knowledge Graphs proper graphs?



WWW2015

Deriving an Emergent Relational Schema from RDF Data

Minh-Duc Pham[△] m.d.pham@vu.nl
Linnea Passing[□] passing@in.tum.de
Orri Erling[◇] oerling@openlinksw.com
Peter Boncz[⊚] boncz@cwi.nl

[△]Vrije Universiteit Amsterdam, The Netherlands
[□]Technische Universität München, Germany
[◇]OpenLink Software, UK [⊚]CWI, The Netherlands

ABSTRACT

We motivate and describe techniques that allow to detect an "emergent" *relational schema* from RDF data. We show that on a wide variety of datasets, the found structure explains well over 90% of the RDF triples. Further, we also describe technical solutions to the semantic challenge to give short names that humans find logical to these emergent tables, columns and relationships between tables. Our techniques can be exploited in many ways, e.g., to improve the efficiency of SPARQL systems, or to use existing SQL-based applications on top of any RDF dataset using a RDBMS.

O (subject, property, object) columns[1]. SQL systems tend to be more efficient than triple stores, because the latter need query plans with many self-joins – one per SPARQL triple pattern. Not only are these extra joins expensive, but because the complexity of query optimization is exponential in the amount of joins, SPARQL query optimization is much more complex than SQL query optimization. As a result, large SPARQL queries often execute with a suboptimal plan, to much performance detriment. RDBMS's can further store data efficiently e.g. using advanced techniques such as column-wise compression, table partitioning, materialized views and multi-dimensional data clustering. These techniques require insight in the (tabular) structure of the

Main Problems in RDF Data Management

- **Bad query plans**
- Low storage locality
- Lack of user schema insight

RDF Triple Indexing

S	P	O
book1	has_title	“Pride & Prejudice”
book1	has_author	“Austen”
book1	isbn_no	“960-425-059-0”
book2	has_title	
book2	has_author	“Pecker”
book2	isbn_no	

SPO Index

P	S	O
has_author	book0	
has_author	book1	“Austen”
has_author	book2	“Pecker”
isbn_no	book0	
isbn_no	book1	“960-425-059-0”
isbn_no	book2	

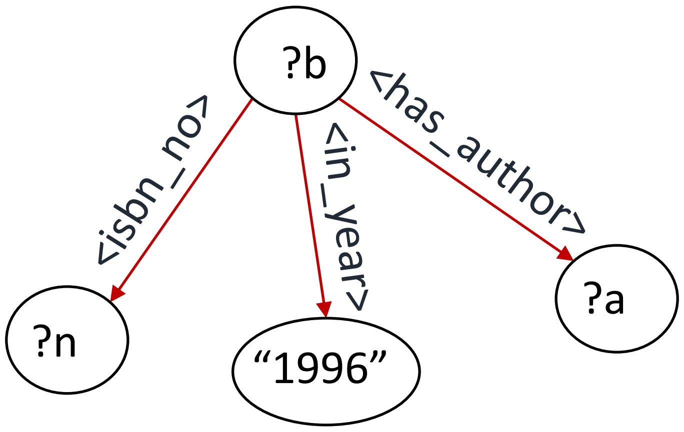
PSO Index

- Most current RDF systems store data with triples sorted on various permutations
  - SPO, PSO, OPS, POS, OSP, SOP,

    PSO – a bit like relational “column store”

    SPO – a bit like relational “row store”

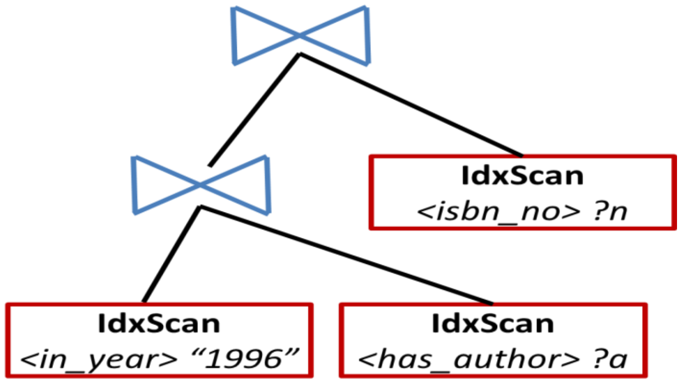
Bad Query Plans



< Query graph>

```
SELECT ?a ?n WHERE
{
        ?b      <has_author> ?a.
        ?b      <in_year>    "1996".
        ?b      <isbn_no>     ?n
}
```

< SPARQL query >



< Example query plan >

- **Have unnecessary joins**
  - All subject having property <isbn_no> always has property <has_author>, but query plan still needs a join for these properties to construct the answer

  problems: **query optimization explosion** + **costly join operations**
- **Hard to find the optimal join order**
  - Being unaware of **structural correlations** makes it difficult to **estimate the join hit ratio** between triple patterns
  - SPARQL queries are very join-intensive

Main Problems in RDF Data Management

- Bad query plans
- **Low storage locality**
- Lack of user schema insight

Low Storage Locality

- Impossible to formulate **clustered index** or **partitioning scheme** without the notion of classes/tables (DBA would say “store all Book triples clustered by Year”)

- Exhaustive indexes for all permutations of S, P, O do not create real locality (contrary to common belief)

```
SELECT ?a, ?n WHERE
{
        ?b      <has_author>    ?a.
        ?b      <in_year>       ?y.
        ?b      <isbn_no>       ?n.
        FILTER (?y = 1997)
}
```

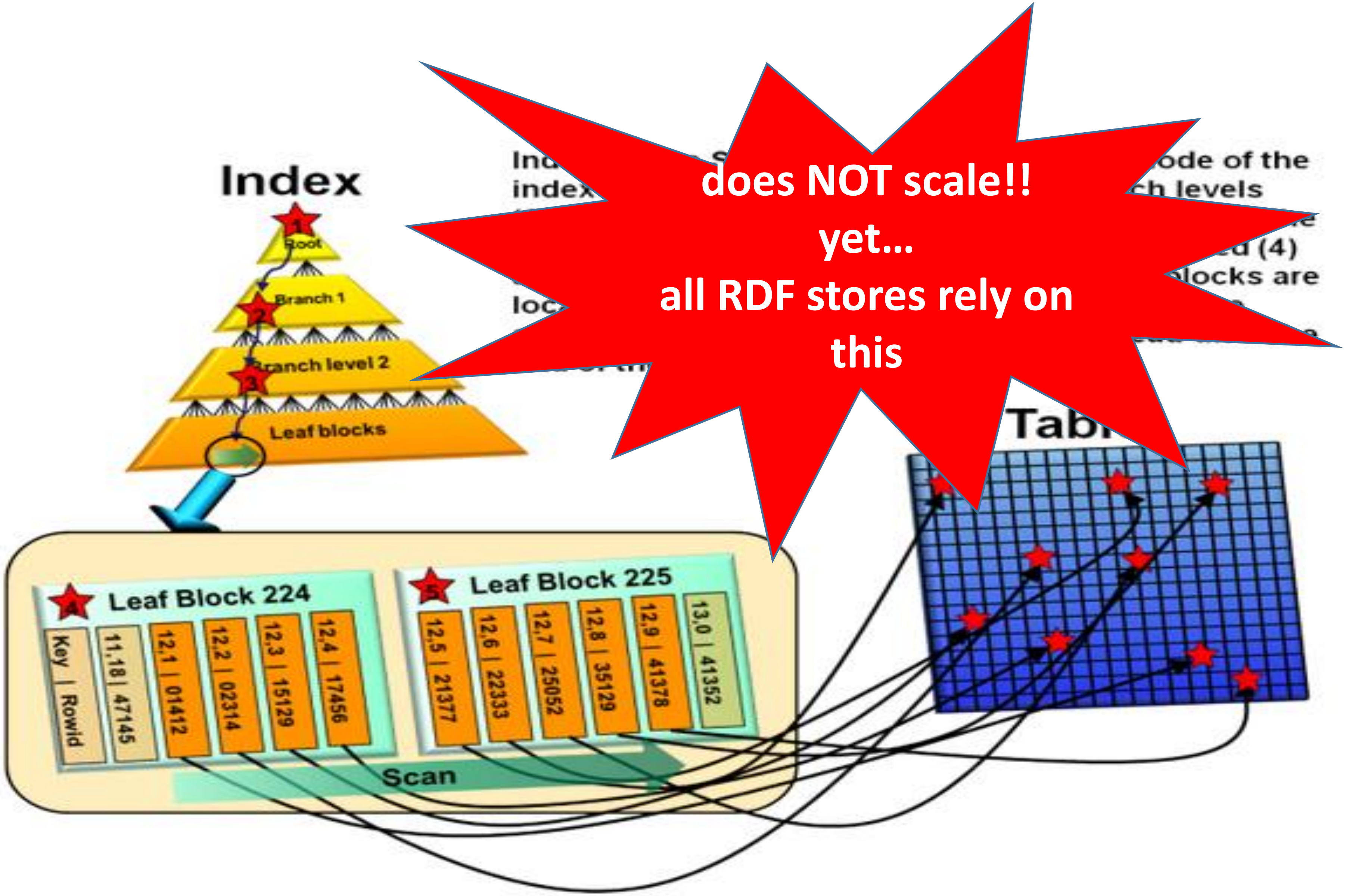
SPARQL

Using **POS index** for quick range selection (in_year,1997,?s)

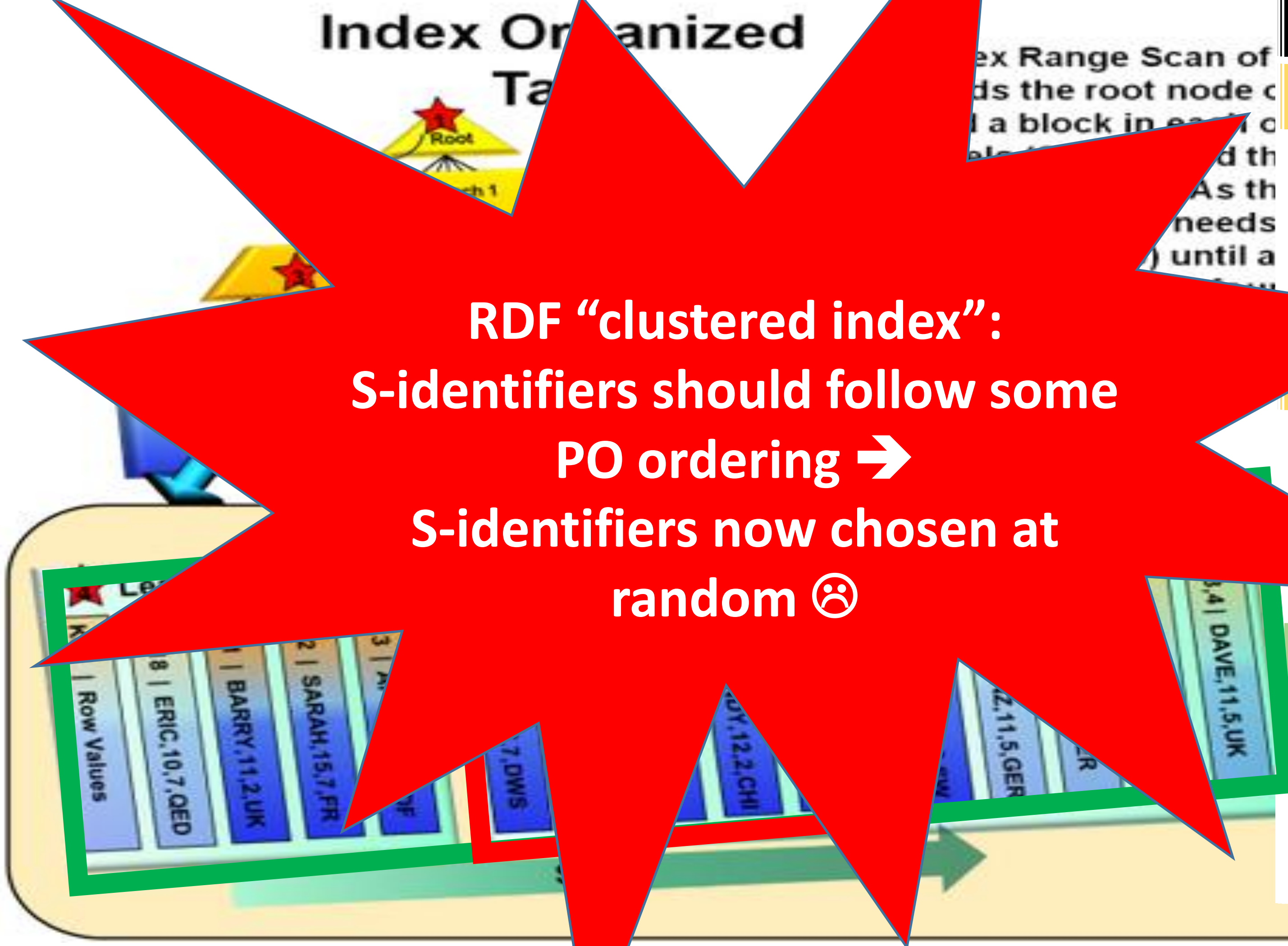
Need **repeated lookups** into a **PSO index** for each attribute

→ **No locality**

Unclustered Index: Random Access Horrors



Clustered Index



RDF "clustered index":
S-identifiers should follow some PO ordering ➔
S-identifiers now chosen at random ☹

S	P	O
Book1	has_title	"Pride & Prejudice"
Book1	has_author	"Austen"
Book1	isbn_no	"960-425-059-0"
Book1	in_year	1996
Book2	has_title	
Book2	has_author	"John Doe"
Book2	isbn_no	"960-477-109-1"
Book2	in_year	1997
Book3	has_title	
Book3	has_author	"Pecker"
Book3	isbn_no	
Book3	in_year	2001

Main Problems in RDF Data Management

- Bad query plans
- Low storage locality
- **Lack of user schema insight**

Lack of User Schema Insight

- RDF data does not have explicit schema
  - **difficult** to formulate SPARQL queries
  - would be good to get a schema (summary)

- Many more tools for relational data access, than for RDF
  - try to **expose** the **regular part** of RDF triple set **as SQL**

Emergent Schema

Recovering the Emergent Schema of RDF data

Emergent schema = “rough” schema to which the majority of triples conforms

Recognize:

- **Classes** (CS) – recognize “classes” of often co-occurring properties
- **Relationships** (CS) – recognize often-occurring references between such classes

+give logical names to these

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://rdfs.org/sioc/ns#num_replies>
<http://purl.org/dc/terms/title>
<http://rdfs.org/sioc/ns#has_creator>
<http://purl.org/dc/terms/date>
<http://purl.org/dc/terms/created>
<http://purl.org/rss/1.0/modules/content/encoded>

“Book”

<has_creator>

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/name>
<http://xmlns.com/foaf/0.1/page>

“Author”

SQL vs RDF: Schema-first vs Schema-last

What does "schema" mean?

Relational Schema

Describes the structure of the occurring data

Concept mixing (for convenience)

Designed for one database (=dataset)

Semantic Web Schema

Purpose: knowledge representation

Describing a concept universe (regardless data)

Designed for interoperability in many contexts

Statement: it is useful to have **both** an (Emergent) Relational and Semantic Schema for RDF data

- useful for systems (higher efficiency)
- useful for humans (easier query formulation)

When is a Emergent Schema of RDF data useful?

- **Compact** Schema
  - as few tables as possible
  - homogeneous literal types (few NULLs in the tables)
- **Human-friendly** “Labels”
  - URIs + human-understandable table/column/relationship names
- High “**Coverage**”
  - the schema should match almost all triples in the dataset
- **Efficient** to compute
  - as fast as data import

Step 1:
Basic CS
discovery

(s1, offers, offer1)
(s1, region, region1)
(s2, offers, offer2)
(s2, offers, offer3)
(s2, region, region1)
...
(offer1, availableDeliveryMethods, DHL)
(offer1, description, "Offer data")
(offer1, hasBusinessFunction, "Sell")
(offer1, hasEligibleQuantity, 1)
(offer1, hasInventoryLevel, 1)
(offer1, hasStockKeepingUnit, 112)
(offer2, availableDeliveryMethods, DHL)
(offer2, hasPriceSpec, price1)
(offer2, hasStockKeepingUnit, 112)
(offer2, type, Offering)
...
(price1, hasCurrency, "EUR")
(price1, hasCurrencyValue, "35.99")
(price1, hasUnitOfMeasurement, "C62")
(price1, valueAddedTaxIncluded, "false")
(price1, eligibleTransactionVolume, 0)
(price1, ...
...

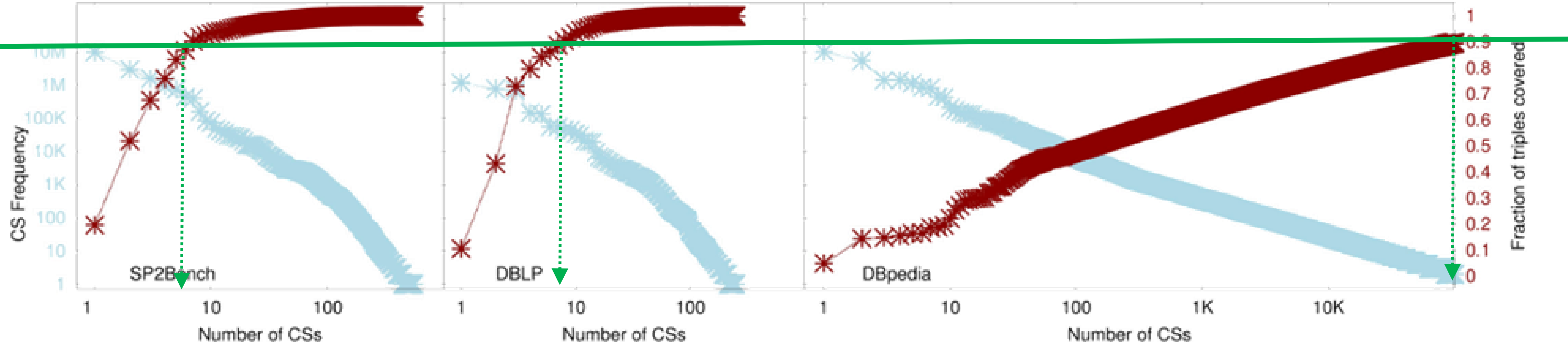
<Example RDF triples>

Characteristic Sets in some well-known RDF datasets

Datasets	#triples*	#CS's	#CS's to cover 90%	Avg. #prop.	#multi-type properties
LUBM	100M	17	7	5.71	0
BSBM	100M	51	14	12.35	0
SP2Bench	100M	554	7	9.8	0
synthetic	*data created by benchmark data generator*				
MusicBrainz	179M	27	10	4.7	0
EuroStat	70K	44	8	7.77	0
DBLP	56M	251	8	13.61	0
PubMed	1.82B	3340	35	19.27	0
relational	*RDF data from a relational database dump*				
WebData.	90M	13914	993	7.79	543
DBpedia	404M	472270	109831	24.02	1601
native	*real data originating as RDF*				

Characteristic Sets in some well-known RDF datasets

- frequency distribution
  - **how many CS's do I need to represent 90% of the triples?**



Partial and Mixed Use of Ontologies

dataset	partial %ontology class properties used per CS
LUBM	37%
BSBM	3%
SP2Bench	4%
MusicBrainz	1%
EuroStat	84%
DBLP	8%
PubMed	-
WebData.	33%
DBpedia	5%

cs_4
dc:description
gor:validFrom
gor:validThrough
gor:hasCurrency
gor:hasCurrencyValue
gor:hasUnitOfMeasurement
gor:valueAddedTaxIncluded
gor:eligibleTransactionVolume

(prefix gor: http://purl.org/goodrelations/v1# prefix dc: http://purl.org/dc/elements/1.1/)

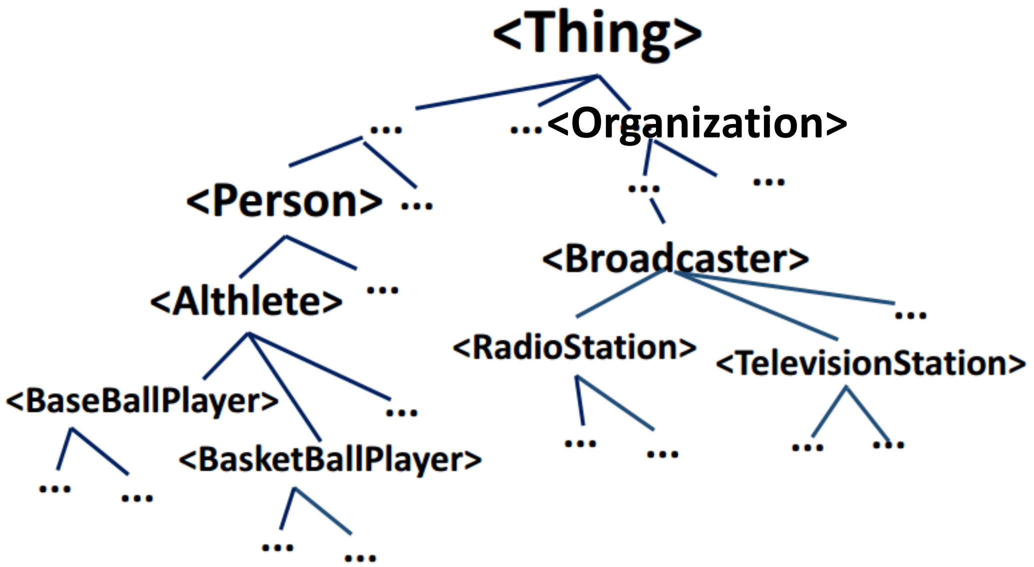
PriceSpecification
gor:description
gor:name
gor:eligibleTransactionVolume
gor:validFrom
gor:validThrough
gor:hasCurrency
gor:hasCurrencyValue
gor:hasUnitOfMeasurement
gor:valueAddedTaxIncluded
gor:hasMaxCurrencyValue
gor:hasMinCurrencyValue

Step 2: Labeling

Using ontologies to get class and property labels:

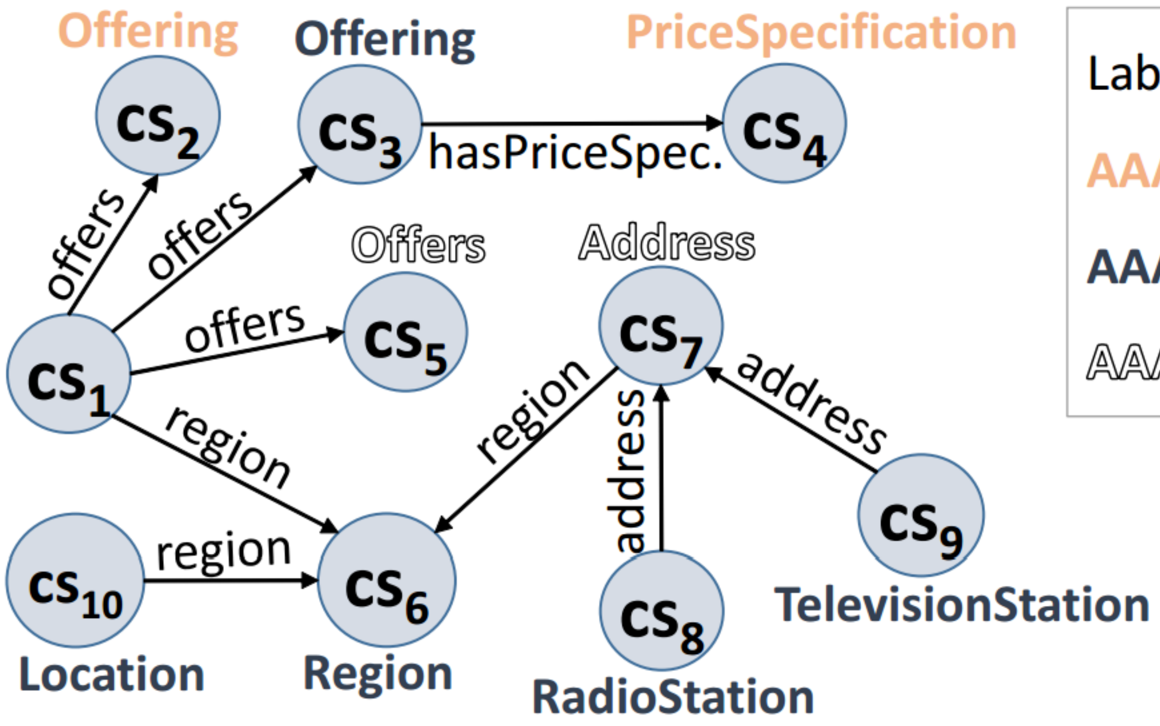
- exploit subclass hierarchy
- TF/IDF: how frequent is the label inside the CS divided by global frequency

label of rdf:type	subjects CS %	subjects all %
Thing	100	83
Organization	100	7
RadioStation	**97**	**0.2**
Company	1	4



We try to associate each CS with an ontology class (will not work always)

Step 2: Labeling



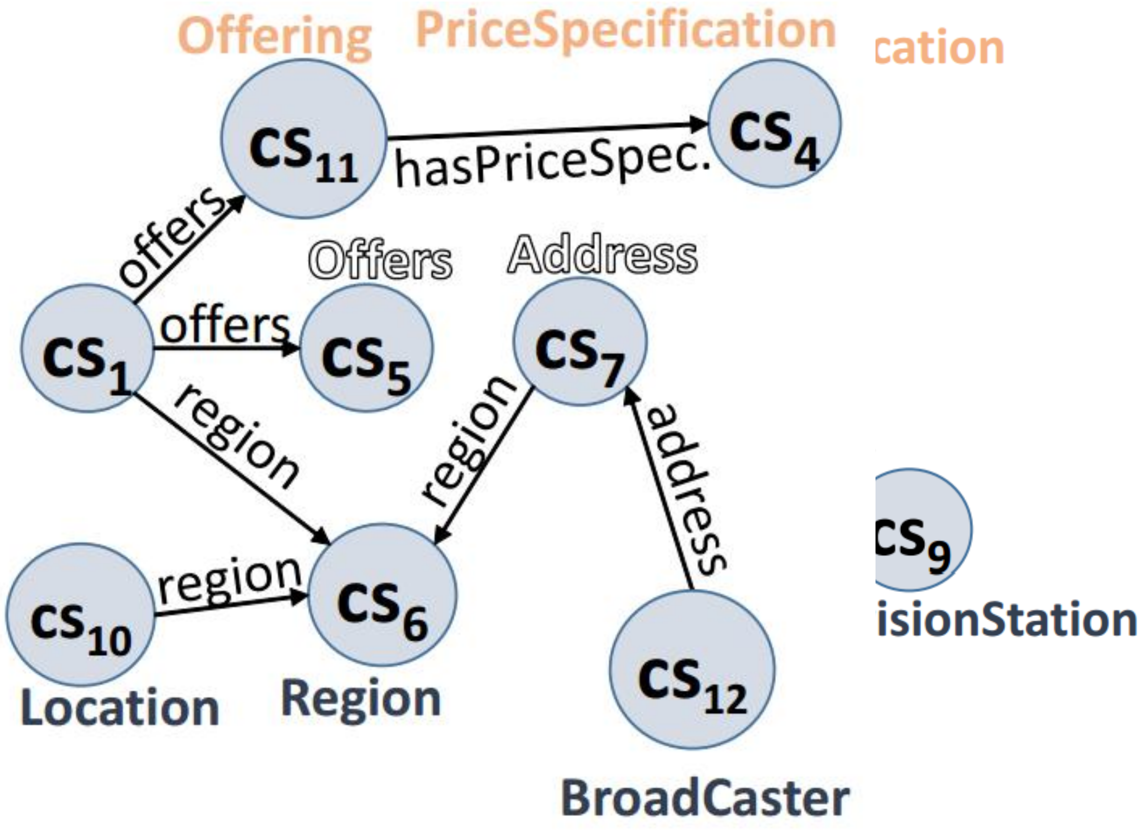
Labels assigned by

AAA: using ontologies

AAA: using **discriminative** properties

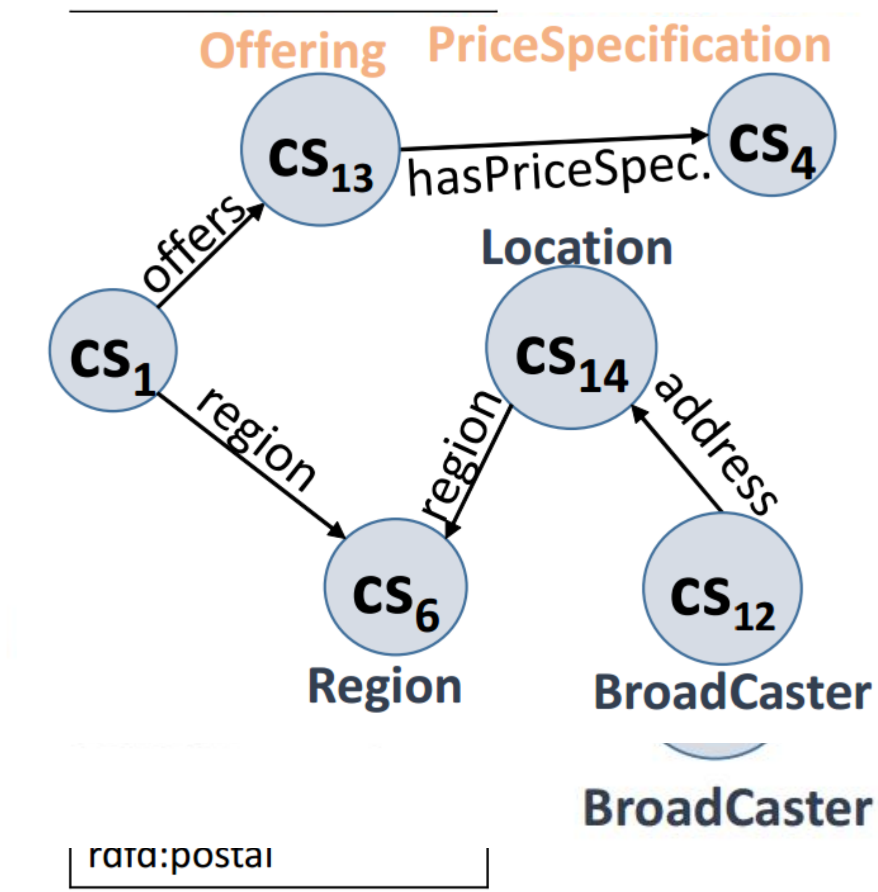
AAA: using CS's relationships

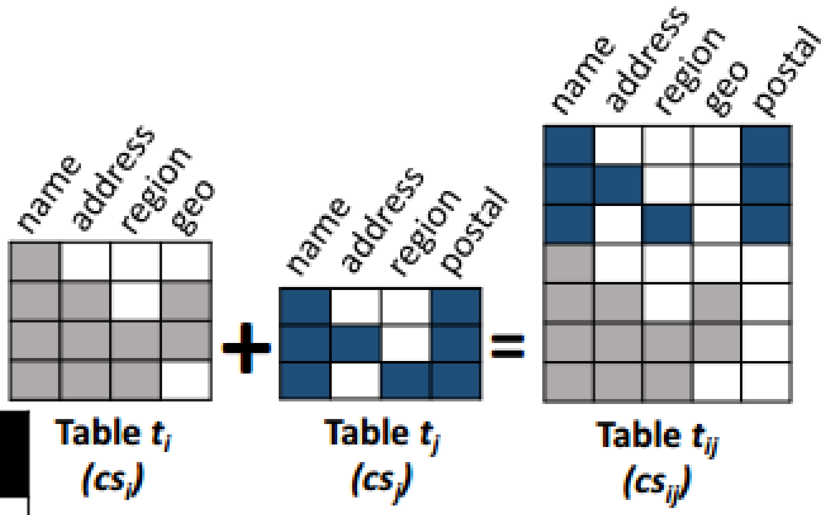
Step 3: CS Merging



Semantic merging: based on ontology correspondences (found during labeling)

Step 3: CS Merging





Structural merging: based on class structure and **discriminative** properties

Step 4: Schema Filtering

Goal: make the schema more compact

- remove **infrequent CS**'s (small tables)
  - except "Dimension Tables"
    - CS that is small but is referred to very often
    - Run **PageRank** on the emergent schema
      - weight is initial frequencies
- remove **infrequent properties**
  - and infrequent relationships

Step 5: Instance Filtering

Reduce the amount of NULLs in relational table representation

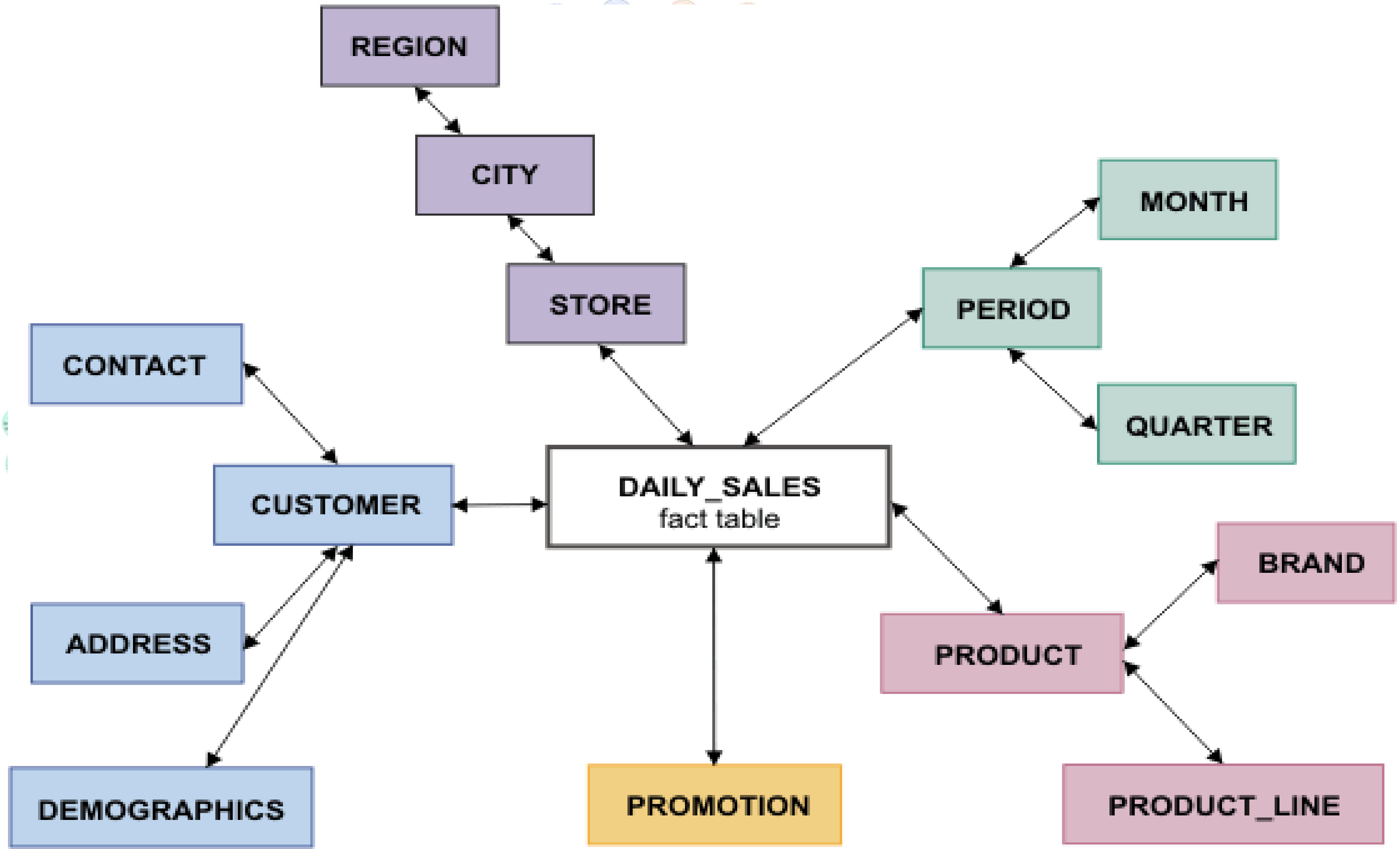
- remove infrequent **literal types**
  - e.g. Person.name is a string, but sometimes a number (remove)
- remove infrequent **multi-valued properties**
  - e.g. Person usually has one birthdate (but a few have multiple)
- remove triples to improve **relationship cardinalities**
  - Car usually has 0 or at most one Brand
    - but some have multiple Brands (remove)

Results: compact schemas with high coverage

Datasets	Number of tables			Coverage – Metric C (%)		
	before merging	after merging	remove small tables	remove small tables	prune infreq. prop.	final schema
LUBM	17	13	12	100	100	**100.00**
BSBM	51	8	8	100	100	**100.00**
SP2B	554	13	10	99.99	99.65	**99.65**
MusicBrainz	27	12	12	100	99.9	**99.60**
EuroStat	44	10	5	99.73	99.53	**99.53**
DBLP	251	9	6	100	99.68	**99.60**
PubMed	3340	14	12	100	99.75	**99.73**
WebData.	13914	3000	253	98.17	94.37	**92.79**
DBpedia	472270	542	234	99.12	96.68	**95.82**

!

Are LOD Knowledge Graphs proper graphs?



Results: understandable labels & performance

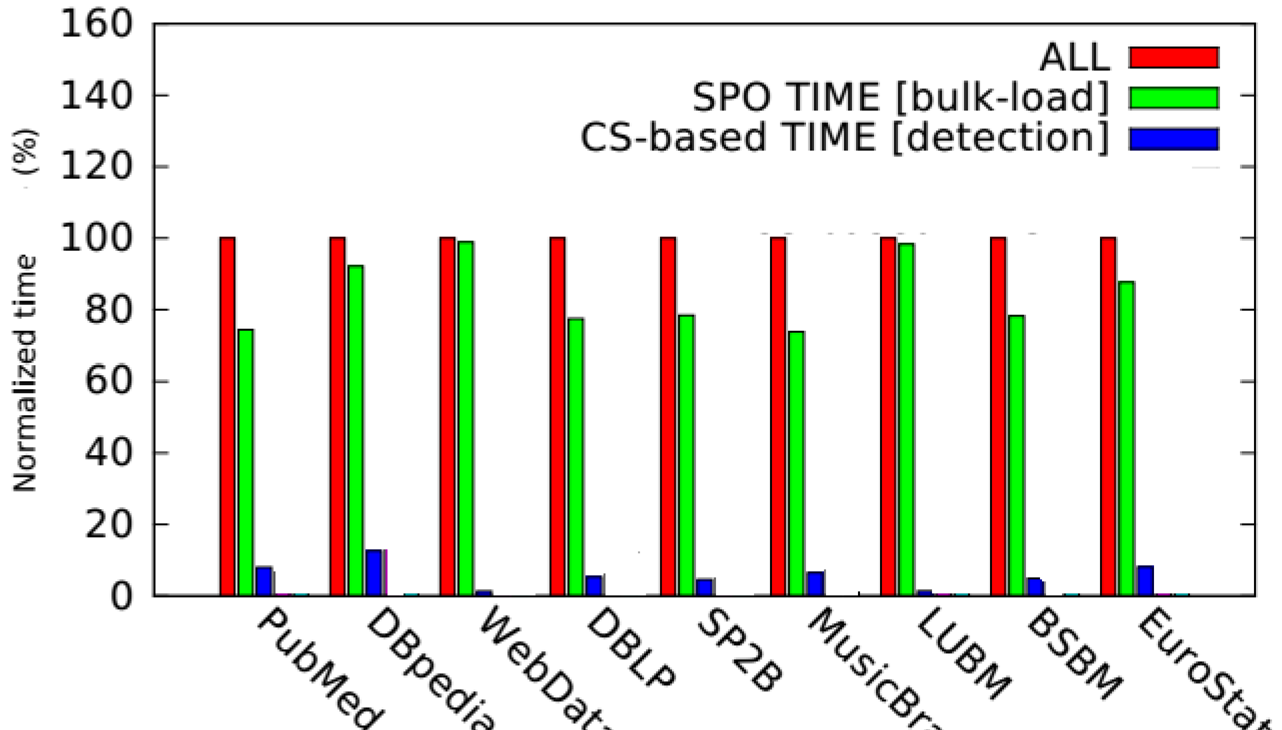
labels	WebData.	DBpedia
top 3	3.6	3.8
final	4.1	4.6

Table 3: Human survey results on Likert scale

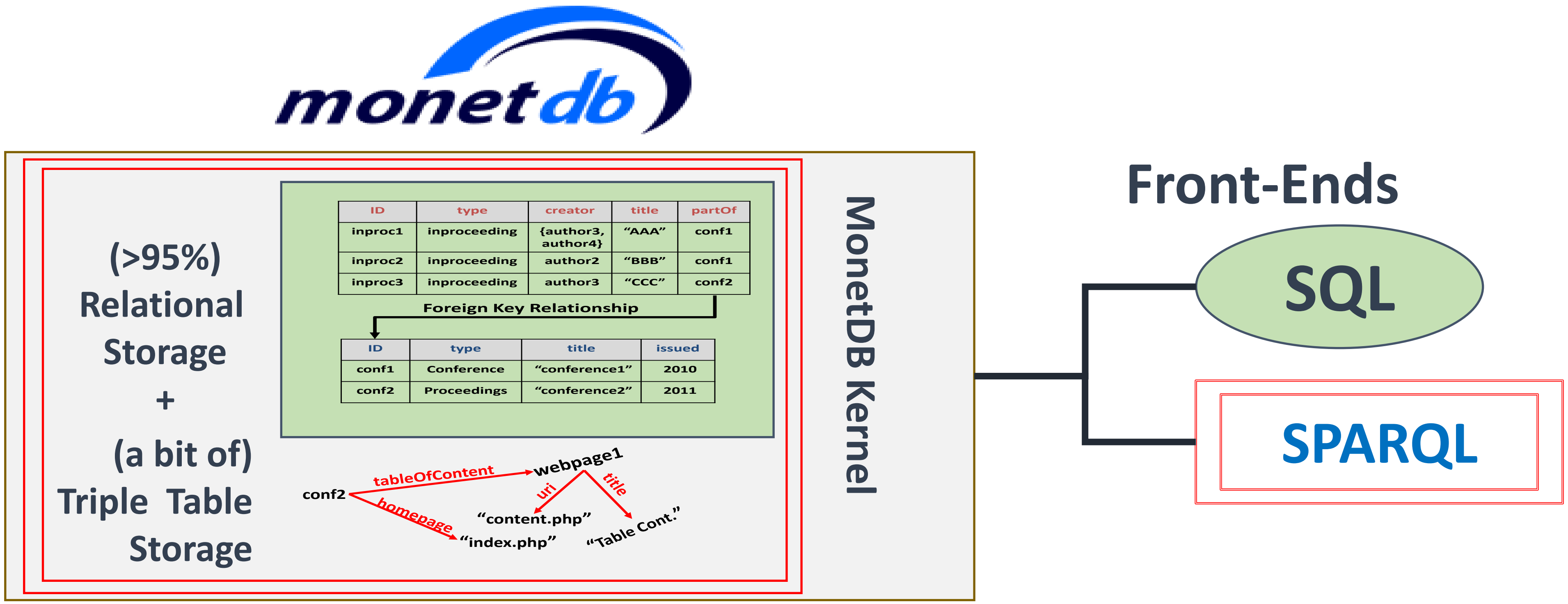
Likert Score: 1=bad ….. 5=excellent

RDF Store	Query 3			Query 5		
	Cold	Hot	Opt. Time	Cold	Hot	Opt. Time
Virt-Quad	4210	53	40.2	3842	1350	18.6
Virt-CS	2965	9	5.4	2130	712	4.2

Table 5: Query time (msecs) w/wo the recognized schema
(Cold: First query runtime after re-starting the server
Hot : Run the query 3 times and get the last runtime
Opt. Time: Query optimization time)



MonetDB/RDF


monetdb
(>95%) Relational Storage
+
(a bit of) Triple Table Storage
ID | type | creator | title | partOf
inproc1 | inproceeding | {author3, author4} | "AAA" | conf1
inproc2 | inproceeding | author2 | "BBB" | conf1
inproc3 | inproceeding | author3 | "CCC" | conf2
Foreign Key Relationship
ID | type | title | issued
conf1 | Conference | "conference1" | 2010
conf2 | Proceedings | "conference2" | 2011
conf2
tableOfContent
webpage1
homepage
uri
title
"content.php"
"index.php"
"Table Cont."
MonetDB Kernel
Front-Ends
SQL
SPARQL

Conclusion

- identified main **RDF Store problems**
  - data locality, query optimization, query formulation

- Identified different **notion of "schema"** in relational vs semantic web
  - argument: we need both **relational schema** and **semantic schema**
  - can bring relational and semantic data management closer together

- Outlined an algorithm for **Emergent Schema** detection in RDF
  - compact, high coverage, understandable labels, efficient

Keynote Statements

(1) Even if data models seem to be very different,
the techniques to manage data are common among them

(at least from a database architect perspective)

(2) some datasets often assumed to belong to very different models are structurally very similar (RDF, graph, relational)