Optimization Approaches for SPARQL and Storage Schemes for RDF
- An Experimental Analysis -

Abschlusskolloquium Graduiertenkolleg 806/3
"Mathematische Logik und Anwendungen"
September 12, 2008

Speaker: *Michael Schmidt*
(joint work with T. Hornung, N. Kuechlin, G. Lausen, and C. Pinkel)

Table of Content

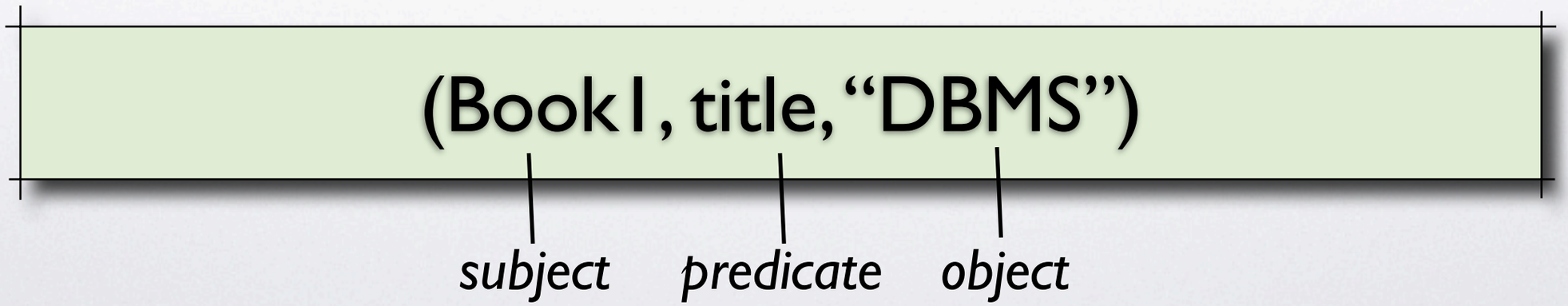
- Introduction to RDF and SPARQL
- Motivation
- The SP^{2}Bench Framework
- Experimental Analysis of RDF Storage Schemes

RDF and SPARQL

- *RDF*
  - Data format for encoding information/knowledge
  - W3C Recommendation since February 2004
- *SPARQL*
  - "SPARQL Protocol and Query Language" for RDF
  - Declarative Language
  - W3C Recommendation since January 2008

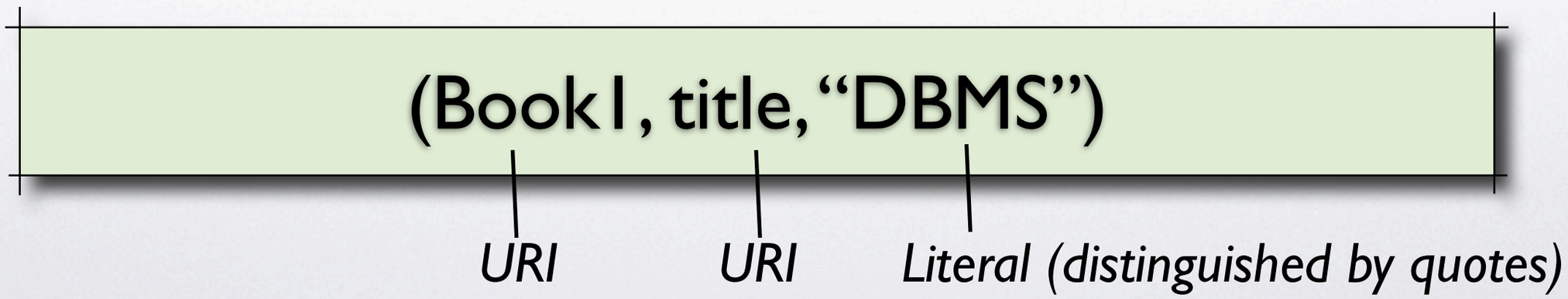
The RDF Data Format

- RDF Triples encode knowledge facts
- Each triple of the form *(subject, predicate, object)*
- Encodes binary relation *predicate* between *subject* and *object*
- Example:



The RDF Data Format

- Three basic sets of elements
  - *U* - URIs (Uniform Resource Identifiers)
  - *B* - Blank Nodes
  - *L* - Literals
- RDF triple t ∈ (U ∪ B) x U x (U ∪ B ∪ L)



The RDF Data Format

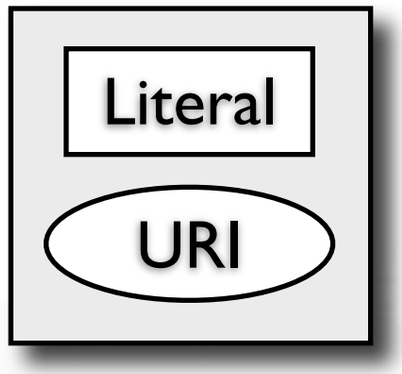
- An RDF Database is a set of triples
  $D \subseteq (U \cup B) \times U \times (U \cup B \cup L)$

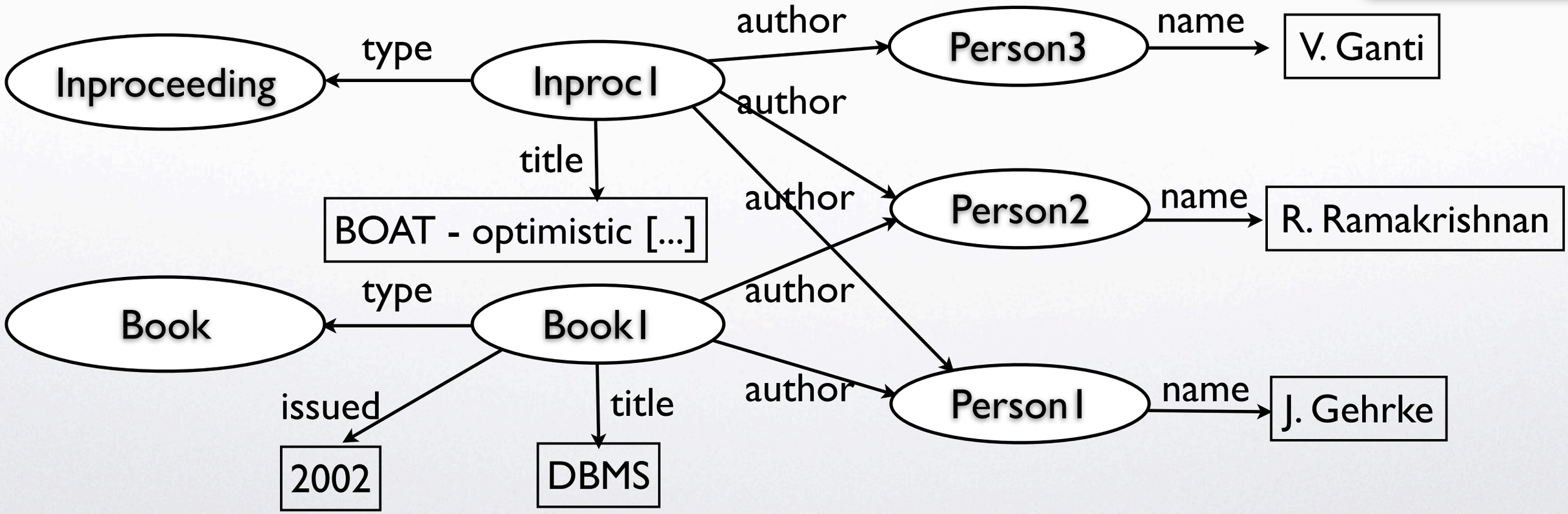
- Example:

{ (Book1, type, Book), (Book1, title, "DBMS"),
(Book1, author, Person1), (Book1, author, Person2),
(Book1,issued, "2002"), (Person1, name, "J. Gehrke"),
(Person2, name, "R. Ramakrishnan"),
(Inproc1, type, Inproceedings),
(Inproc1, title, "BOAT - optimistic decision tree construction"),
(Inproc1, author, Person1), (Inproc1, author, Person2),
(Inprocl, author, Person3), (Person3, name, "V. Ganti") }

RDF Graph Representation

- RDF Databases can be represented by directed, acyclic graphs





SPARQL Query Language

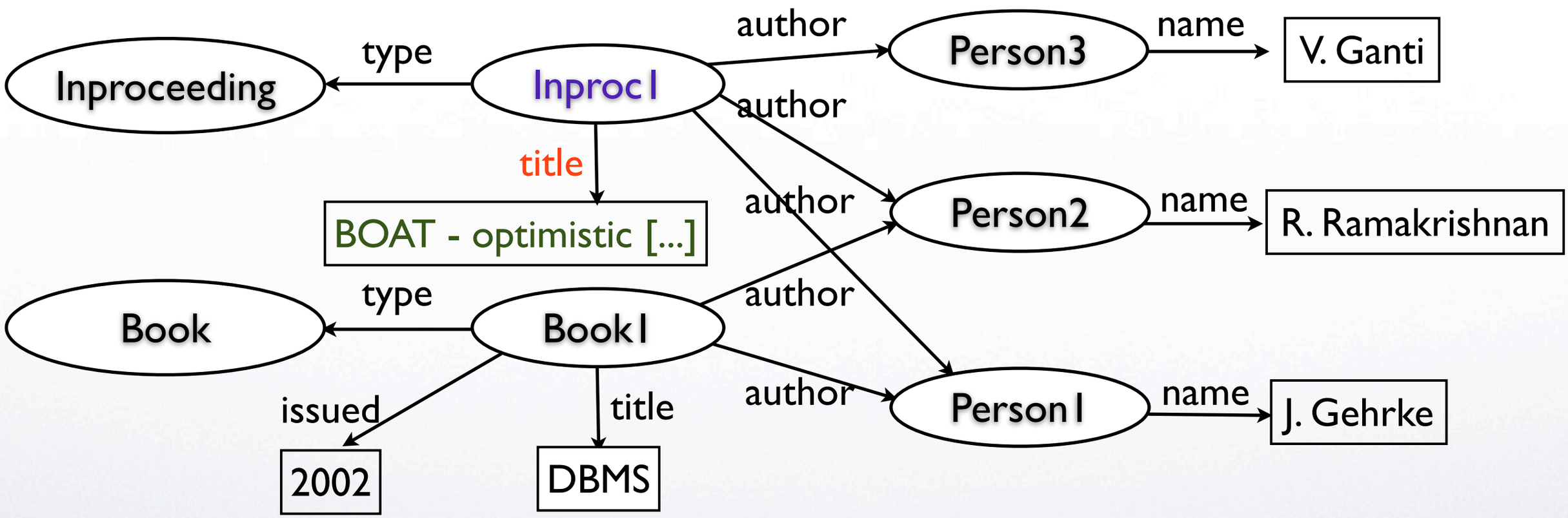
- Let V be a set of variables { ?x_1, ?x_2, ?x_3, ... }
- Basic construct: triple patterns of the form
  t ∈ (U ∪ B ∪ V) x (U ∪ V) x (U ∪ B ∪ L ∪ V)
- Example:

also written as:

```
(?e,title,?title)
```

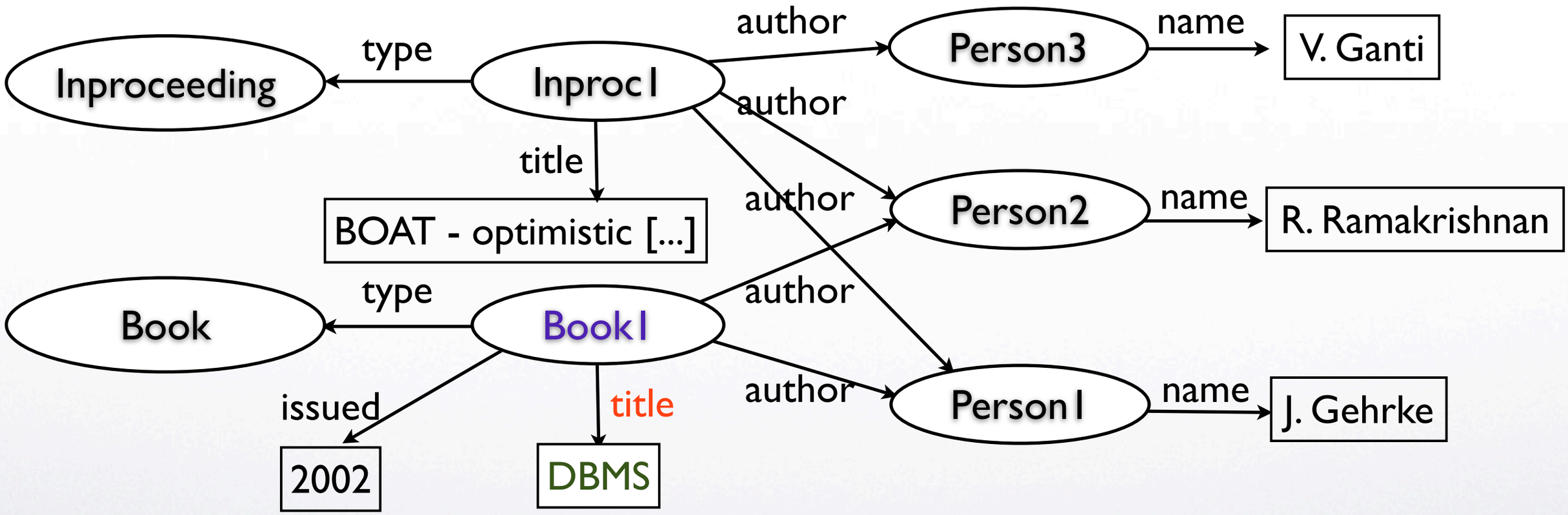
```
?e title ?title
```

SPARQL Query Language



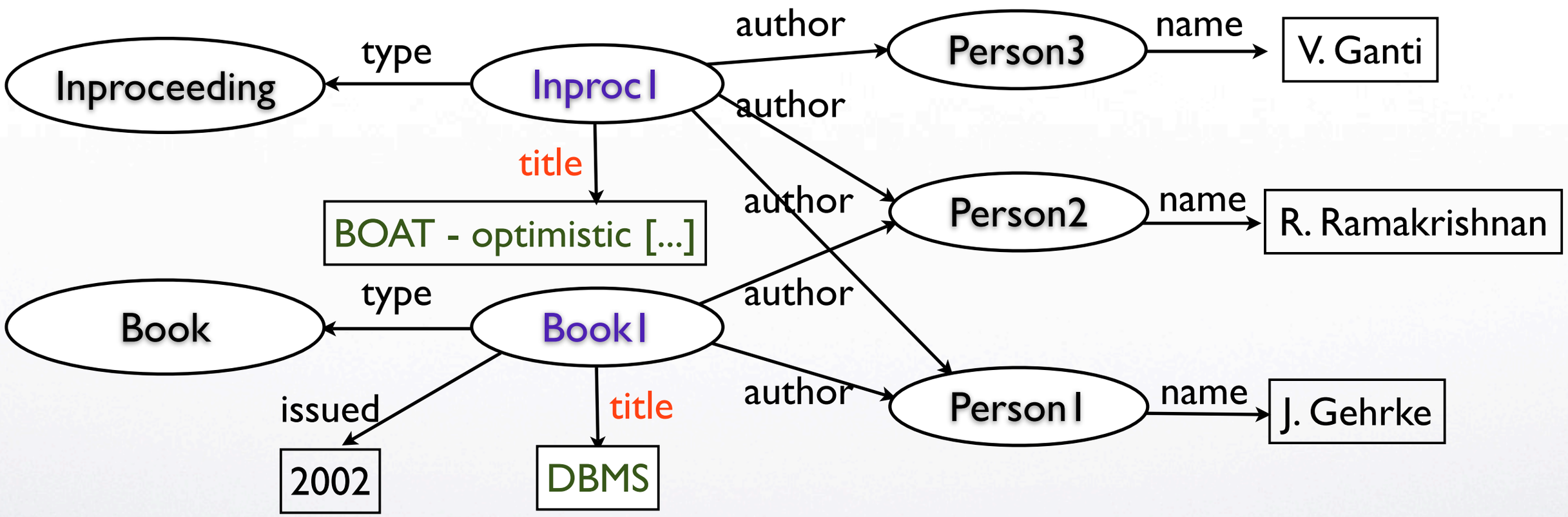

```
?e title ?title
```

SPARQL Query Language




```
?e title ?title
```

SPARQL Query Language



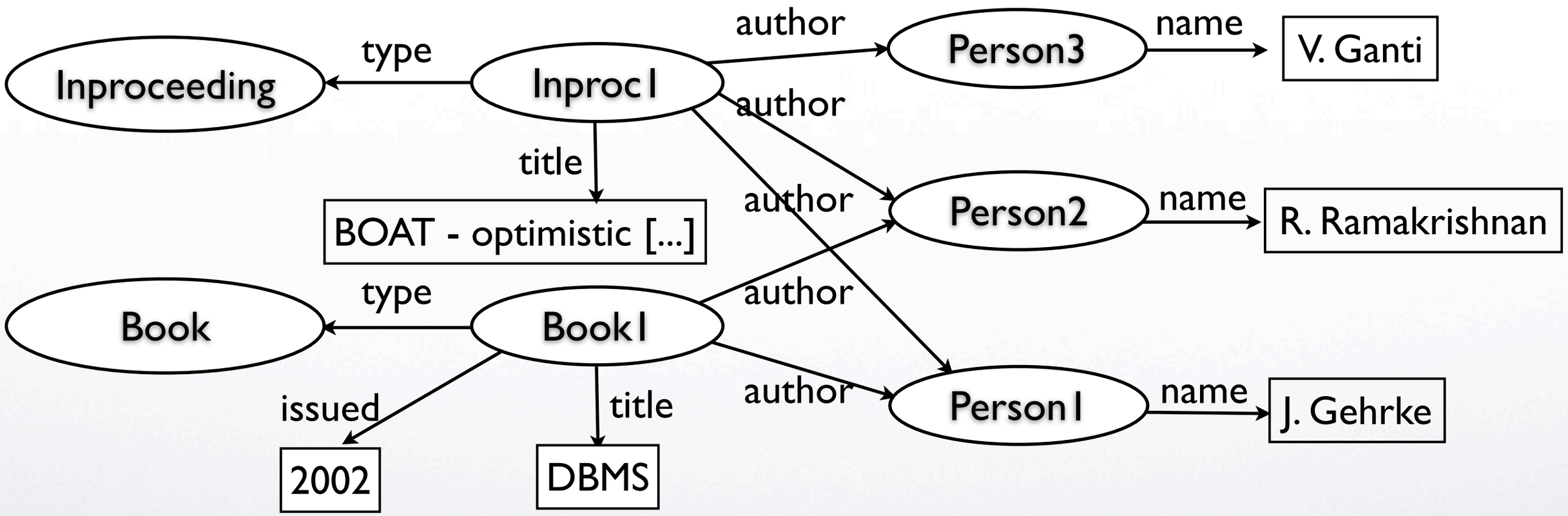

```
?e title ?title
```

S = {{ ?e → Inproc1, ?title -> "BOAT ..." },
{ ?e → Book1, ?title → "DBMS" }}

SPARQL Query Language

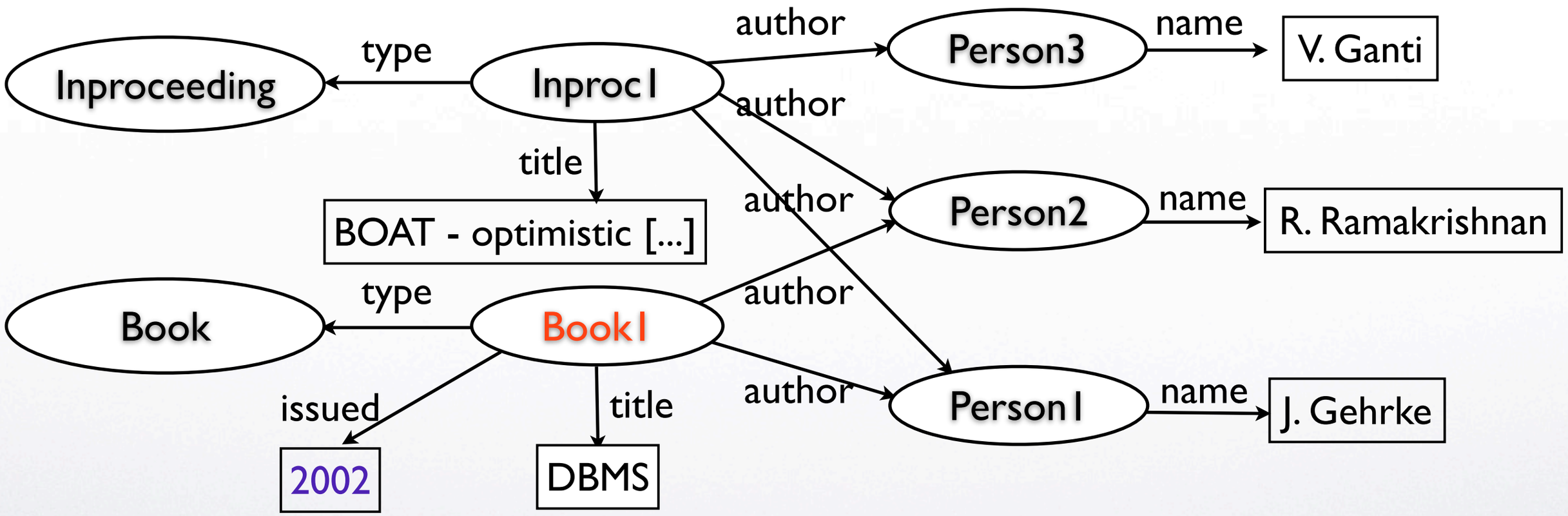
- Operators can be used to combine simple triple patterns to more complex queries
  - **`SELECT`**: projection for variables
  - **`AND`**: (denoted as “.”): join over shared variables
  - **`UNION`**: computes union of two result sets
  - **`OPTIONAL`**: optional selection of components
  - **`FILTER`**: filters for a specified condition

SPARQL Query Language




```
SELECT ?yr
WHERE { ?book type Book. ?book title "DBMS". ?book issued ?yr }
```

SPARQL Query Language



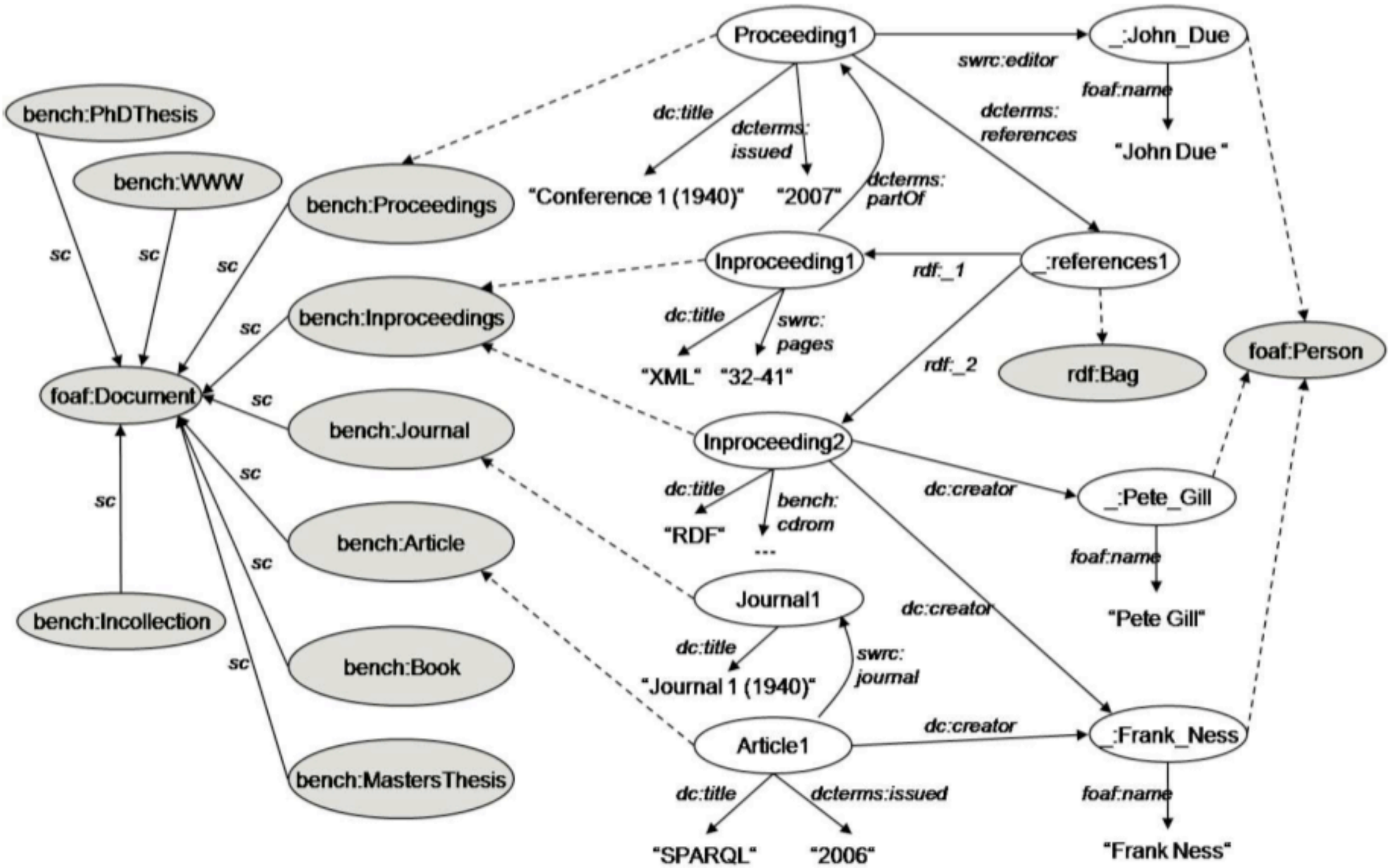

```
SELECT ?yr
WHERE { ?book type Book. ?book title "DBMS". ?book issued ?yr }
```

Motivation

- Efficient evaluation of SPARQL is a non-trivial task
  - SPARQL evaluation is **PSPACE**-complete (even for many subfragments)
  - Homogeneous data format poses potential for severe bottlenecks (as we will discuss later)
  - Several optimization approaches have been made, but use their own, user-defined experimental setting for verification

Motivation

- Need for a SPARQL benchmark framework that
  - ... poses various challenges to SPARQL engines
  - ... helps to identify strengths/weaknesses of approaches
  - ... allows to compare optimization approaches
- *SP²Bench Framework* satisfies these needs
  - Data generator for arbitrarily large bibliographic models in RDF format
  - Set of meaningful and challenging benchmark queries


Proceeding1
_:John_Due
swrc:editor
foaf:name
dc:title
dcterms:
issued
dcterms:
references
"John Due "
bench:PhDThesis
bench:WWW
bench:Proceedings
"Conference 1 (1940)"
"2007"
dcterms:
partOf
Inproceeding1
_:references1
rdf:_1
sc
sc
sc
sc
bench:Inproceedings
dc:title
swrc:
pages
"XML"
"32-41"
rdf:_2
rdf:Bag
foaf:Person
foaf:Document
sc
bench:Journal
Inproceeding2
sc
dc:title
bench:
cdrom
"RDF"
...
dc:creator
_:Pete_Gill
sc
bench:Article
foaf:name
sc
Journal1
dc:creator
bench:Incollection
dc:title
swrc:
journal
"Pete Gill"
sc
bench:Book
"Journal 1 (1940)"
dc:creator
_:Frank_Ness
Article1
bench:MastersThesis
dc:title
dcterms:issued
foaf:name
"SPARQL"
"2006"
"Frank Ness"

SP²Bench Queries

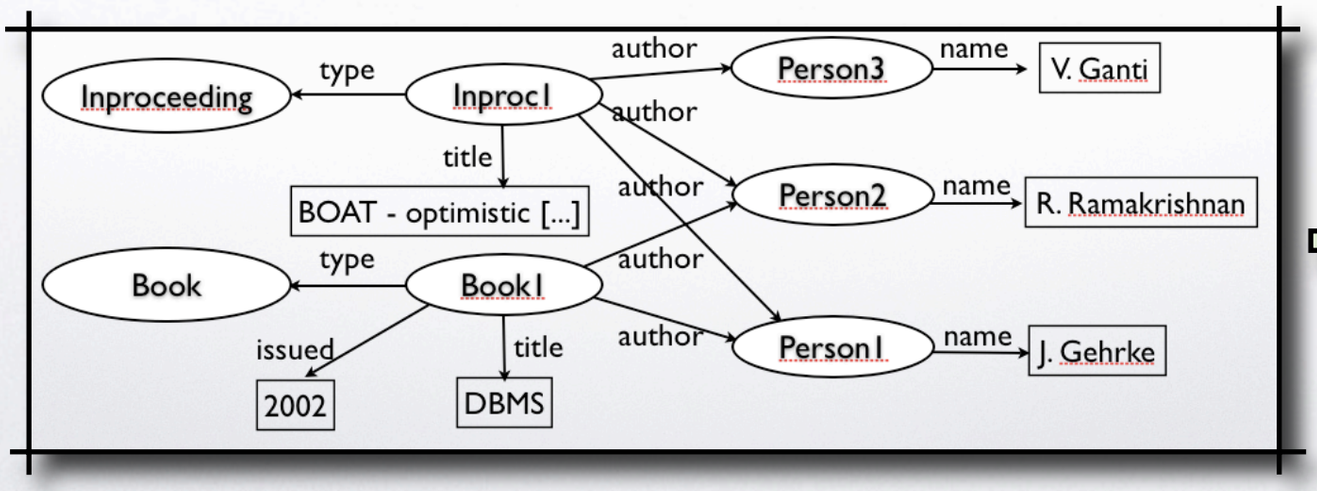
- Vary in a broad range of characteristics
  - Meaningful requests on top of the data
  - Different operator constellation, RDF access patterns, and complexity
  - Result size (small, large, linear, ...)
  - Number of variables
  - ...

Storage Schemes for RDF

- Idea: translation into Relational context and evaluation of queries with conventional SQL database systems
- We consider two different approaches
  - Simple Triple Table Approach
  - Vertical Partitioning

Triple Table Approach

- Simple and straightforward storage scheme for RDF data
- All data stored in a single relation Triples(*subject*, *predicate*, *object*)



Triples

subject	*predicate*	*object*
Book1	type	Book
Book1	title	"DBMS"
Book1	issued	"2002"
Book1	author	Person1
Book1	author	Person2
Person1	name	"J. Gehrke"
...	...	...

Triple Table Approach

- Systematic SPARQL-to-SQL rewriting to evaluate SPARQL queries on top of the triples table

```
SELECT ?yr
WHERE {
   ?book type Book.
   ?book title "DBMS".
   ?book issued ?yr
}
```

SPARQL-to-SQL translation

```
SELECT T3.object AS yr
FROM Triples T1,
     Triples T2,
     Triples T3
WHERE
   T1.predicate = "type" AND
   T1.object = "Book" AND
   T2.predicate = "title" AND
   T2.object = "DBMS" AND
   T3.predicate = "issued" AND
   T1.subject = T2.subject AND
   T2.subject = T3.subject
```

Triple Table Approach

- Main disadvantage: Resulting queries typically contain many self-joins

```
SELECT T3.object AS yr
FROM Triples T1,
     Triples T2,
     Triples T3
WHERE
  T1.predicate = “type” AND
  T1.object = “Book” AND
  T2.predicate = “title” AND
  T2.object = “DBMS” AND
  T3.predicate = “issued” AND
  T1.subject = T2.subject AND
  T2.subject = T3.subject
```

Dictionary Encoding

- URIs and Literals tend to be long strings
- Save space and accelerate joins through encoding

Triples

subject	*predicate*	*object*
Book1	type	Book
Book1	title	“DBMS”
Book1	issued	“2002”
Book1	author	Person1
Book1	author	Person2
Person1	name	“J. Gehrke”
…	…	…

Dictionary encoding

Triples

subject	*predicate*	*object*
1	2	3
1	4	5
1	6	7
1	8	9
1	8	10
9	11	12
…	…	…

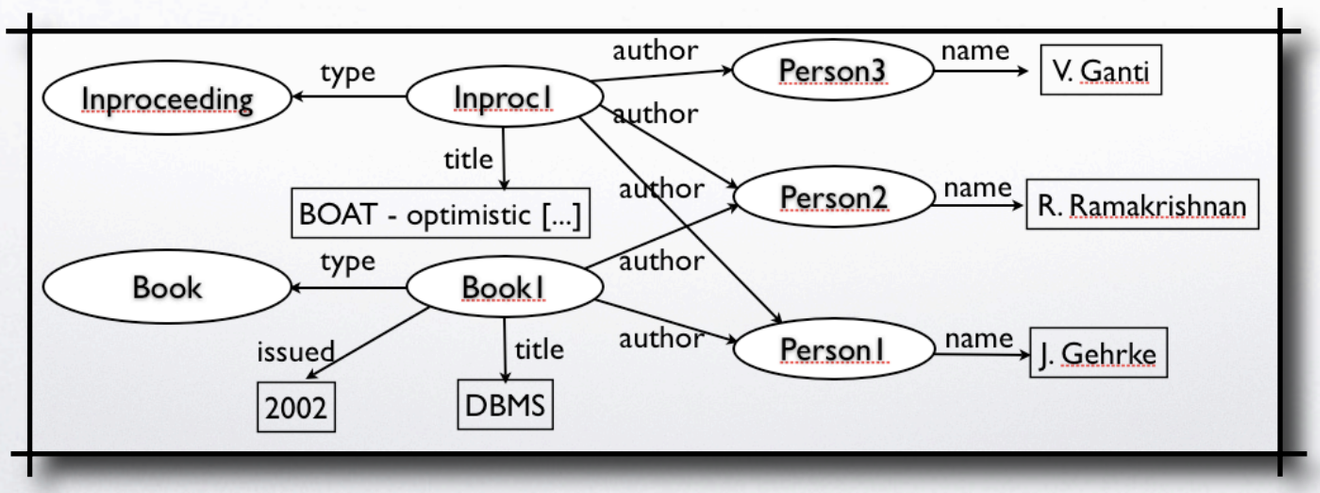
+

Dictionary

ID	*val*
1	Book1
2	type
3	Book
4	title
5	“DBMS”
6	issued
7	2002
8	author
9	Person1
10	Person2
11	name
12	J. Gehrke
…	…

Vertical Partitioning

- Set up one table for each distinct property (predicate) in the data
- Per table, store all tuples with the respective predicate



type

subject	*object*
Book1	Book
Inproc1	Inproceeding

author

subject	*object*
Book1	Person1
Book1	Person2
Inproc1	Person1
Inproc1	Person2
Inproc1	Person3

name

subject	*object*
Person1	"J. Gehrke"
Person2	"R. Ramakrishnan"
Person3	"V. Ganti"

...

Vertical Partitioning

- Systematic SPARQL-to-SQL rewriting also exists for this scheme
- Approach very similar to this for the Triple Table approach
- Resulting queries contain joins over the (typically smaller) predicate tables instead of the whole Triples table

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
   t.object="Book" AND
   t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Query: Select all book authors

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
   t.object="Book" AND
   t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2), (Person2)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2), (Person2), (Person3)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
   t.object="Book" AND
   t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2), (Person2), (Person3)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2), (Person2), (Person3), (Person4)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2), (Person2), (Person3), (Person4), (Person5)}

Efficient Merge Joins

```
SELECT a.object
FROM type t, author a
WHERE
  t.object="Book" AND
  t.subject=a.subject
```

type

subject	*object*
Book1	Book
Book2	Book
Book3	Book
...	...

author

subject	*object*
Book1	Person1
Book1	Person2
Book2	Person2
Book2	Person3
Book3	Person4
Book3	Person5
Book3	Person6
...	...

Result: {(Person1), (Person2), (Person2), (Person3), (Person4), (Person5), (Person6)}

Experimental Setting

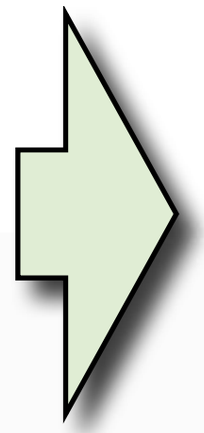
- Scenario ***TR***:
  - Simple Triple Table Approach
  - Physically sorted by (*predicate*, *subject*, *object*)
  - Combined with Dictionary Encoding
- Scenario ***VP***:
  - Vertical Partitioning as described before
  - Combined with Dictionary Encoding

Experimental Setting

- Scenario ***RS***:
  - Purely relational model of the scenario
  - Using flat tables for docs, authors, citations, etc.
  - Designed using ERM DB modeling techniques
- Scenario ***SP***:
  - Sesame native SPARQL engine
  - No RDF/SPARQL-to-SQL translation necessary

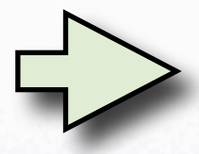
Settings Summary

- TR: Triple Table Approach
- VP: Vertical Partitioning
- RS: Purely Relational Schema



MonetDB mserver v5.5.0, using the new algebra frontend

- SP: SPARQL Engine Sesame



Sesame v2.0 coupled with its native SAIL

- Intel2 DuoCore 2.13GHz CPU, 3GB DDR2 RAM
- OS: Ubuntu v7.10 gutsy Linux
- 30min timeout per query, 2GB main memory limit

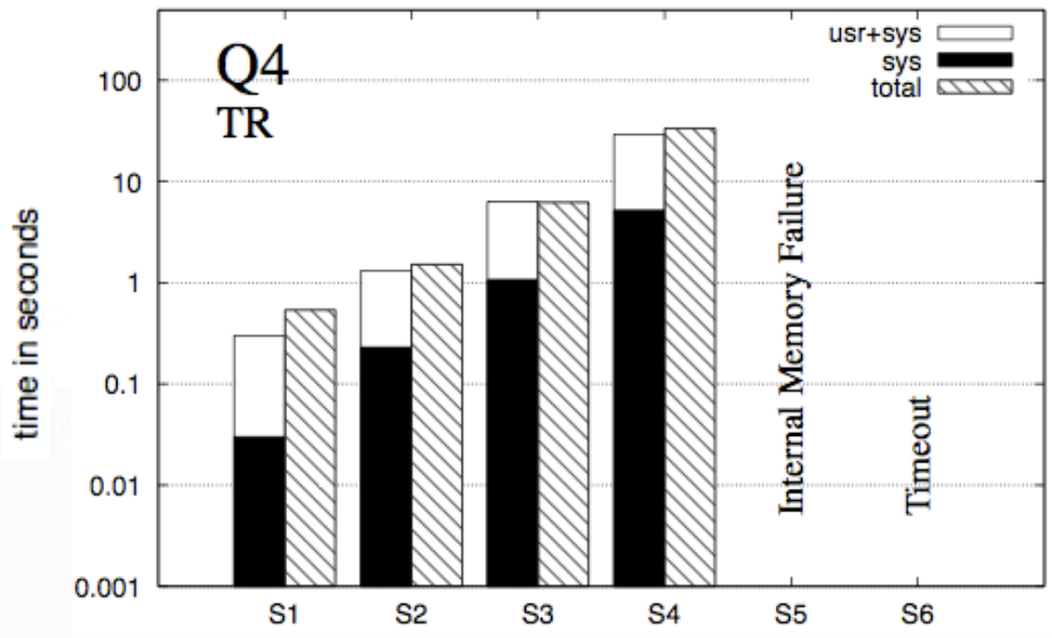
Experimental Results Q4

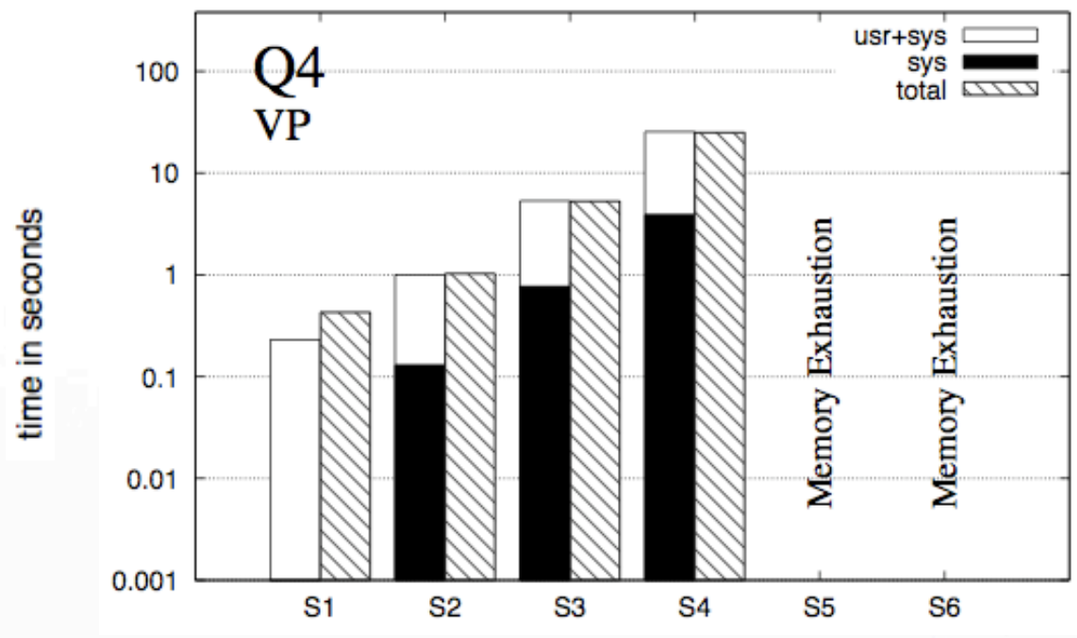
Select all distinct pairs of article author names that have published in the same journal

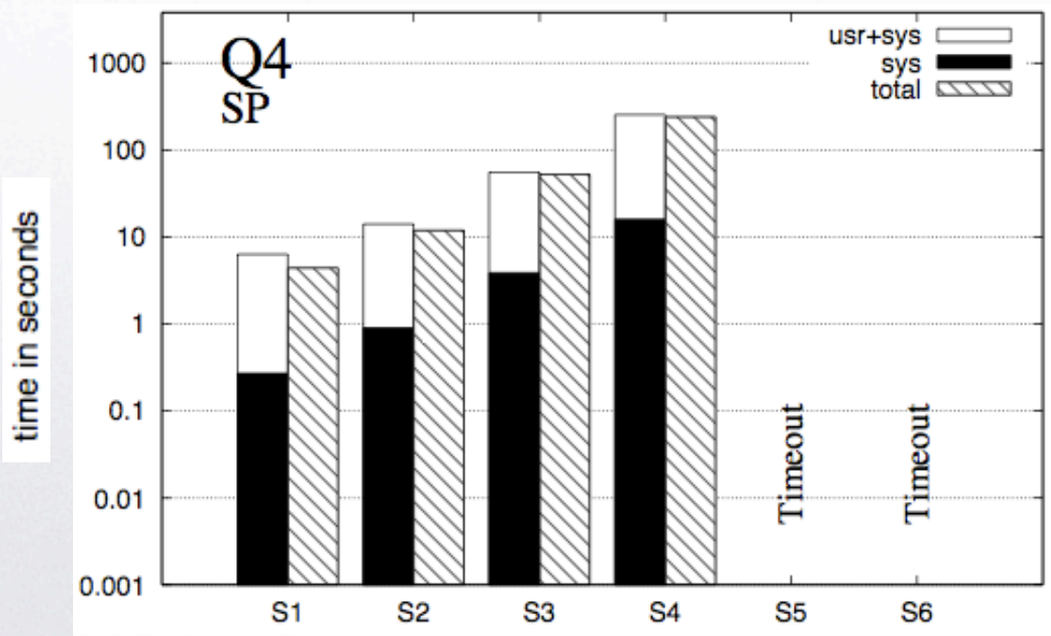
Q4

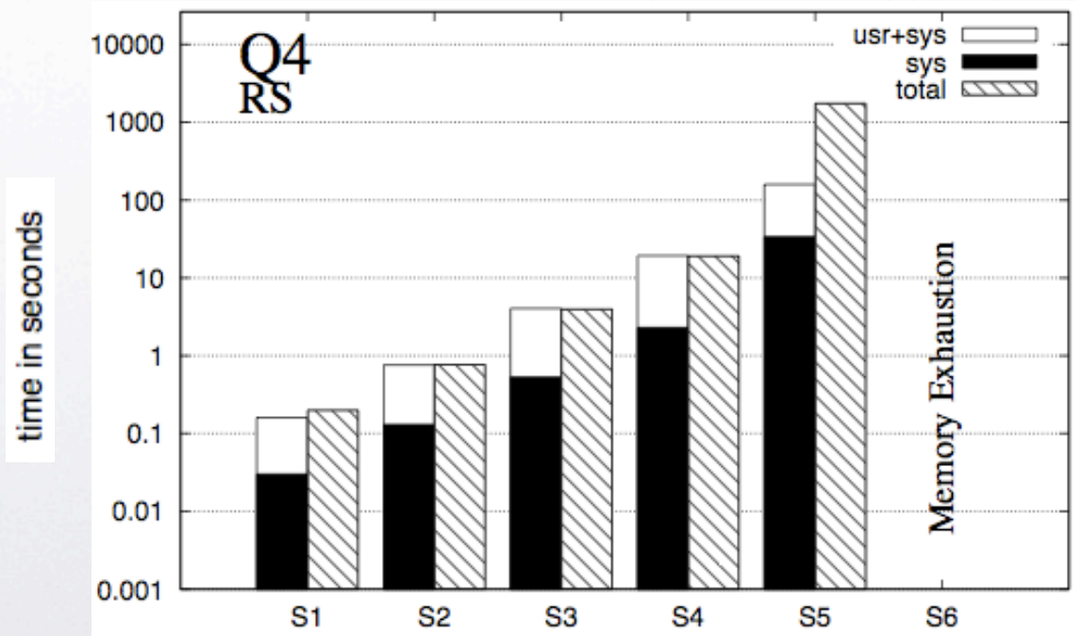
```
SELECT DISTINCT ?name1 ?name2
WHERE {
  ?article1 rdf:type bench:Article.
  ?article2 rdf:type bench:Article.
  ?article1 dc:creator ?author1.
  ?author1 foaf:name ?name1.
  ?article2 dc:creator ?author2.
  ?author2 foaf:name ?name2.
  ?article1 swrc:journal ?journal.
  ?article2 swrc:journal ?journal
  FILTER (?name1<?name2) }
```

#Triples: S1=10k / S2=50k / S3=250k / S4=1M / S5=5M / S6=25M









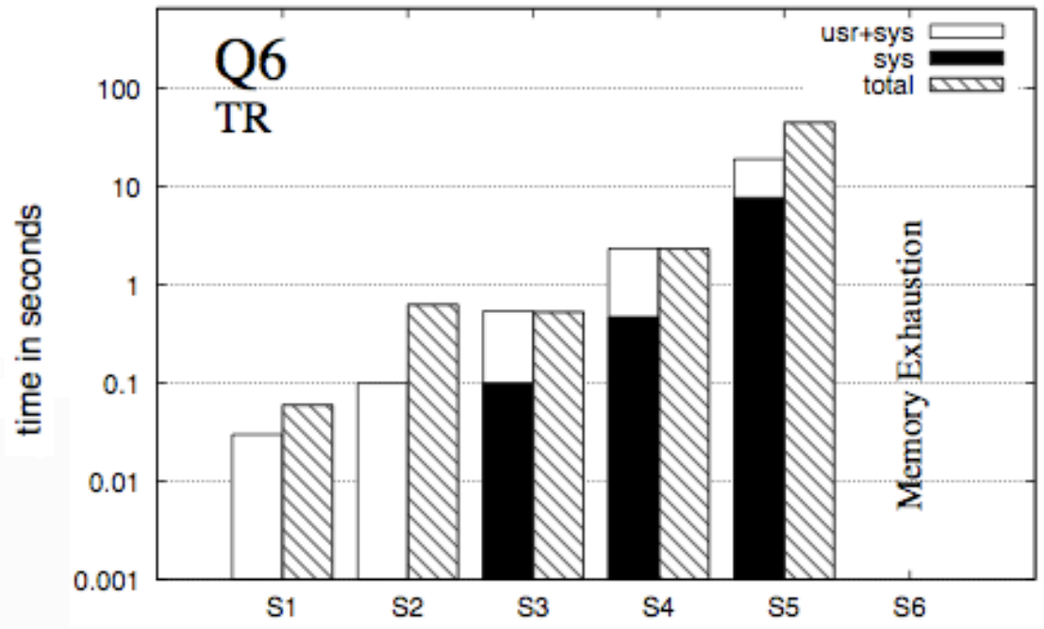
Experimental Results Q6

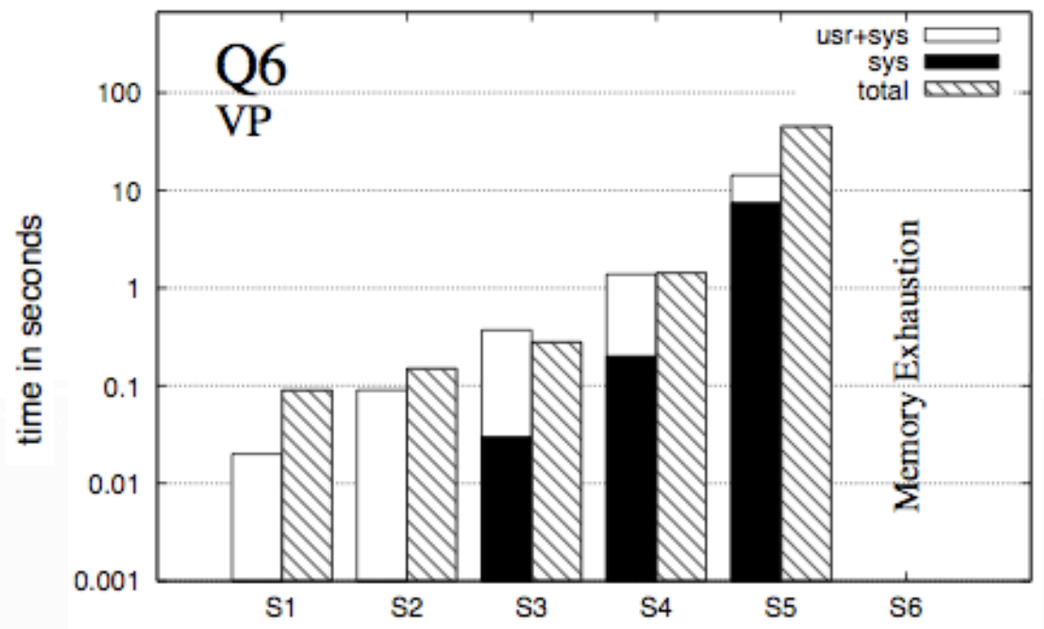
Return, for each year, the set of publications authored by persons that have not published in a year before

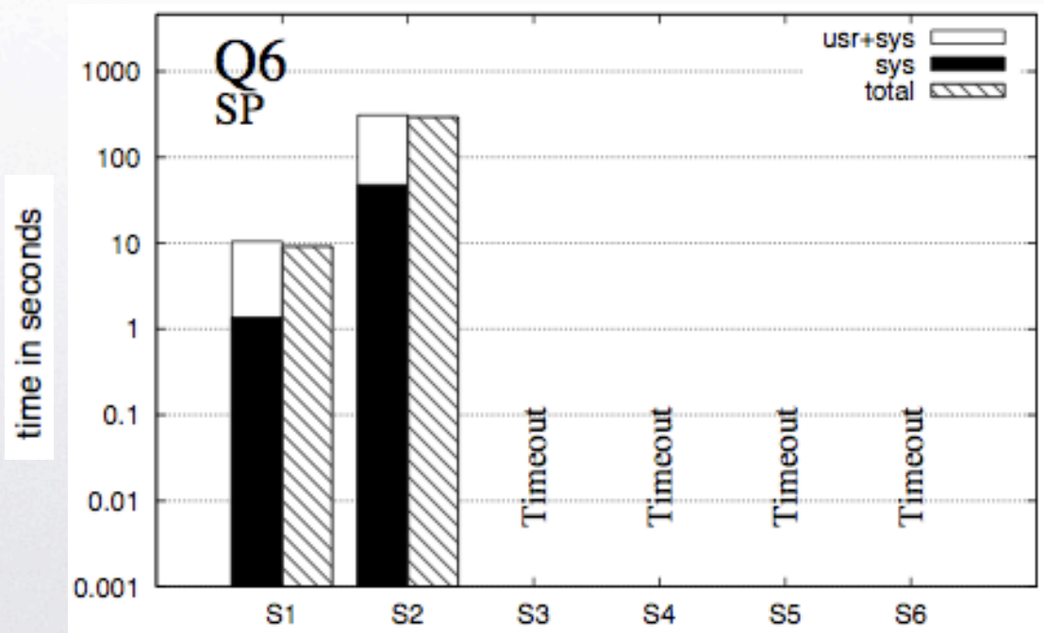
Q6

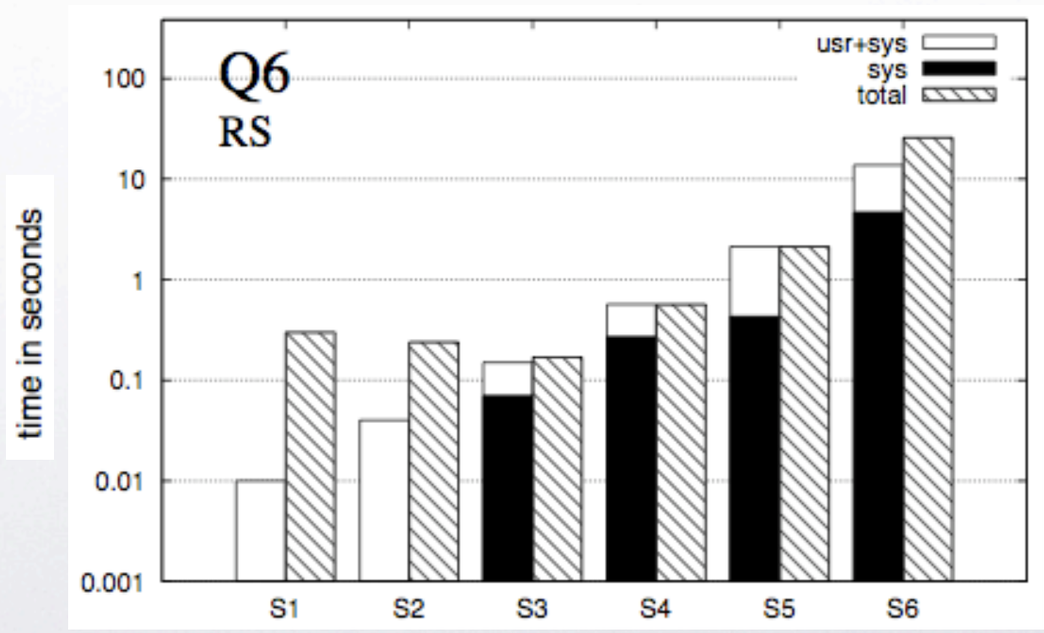
```
SELECT ?yr ?name ?doc
WHERE {
  ?class rdfs:subClassOf foaf:Document.
  ?doc rdf:type ?class.
  ?doc dcterms:issued ?yr.
  ?doc dc:creator ?author.
  ?author foaf:name ?name
  OPTIONAL {
    ?class2 rdfs:subClassOf foaf:Document.
    ?doc2 rdf:type ?class2.
    ?doc2 dcterms:issued ?yr2.
    ?doc2 dc:creator ?author2
    FILTER (?author=?author2 && ?yr2<?yr)
  } FILTER (!bound(?author2))
}
```

#Triples: S1=10k / S2=50k / S3=250k / S4=1M / S5=5M / S6=25M



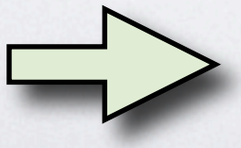






Conclusion

- Experiments identified previously unknown drawbacks
- Vertical Partitioning not a general solution
- Triple Store with different physical sort order competitive to Vertical Partitioning
- Gap of at least one order of magnitude compared to relational data processing, increasing with doc size



New storage schemes and query evaluation approaches need to be developed, to bring forward the evaluation of SPARQL queries

Selected References

M. Schmidt, T. Hornung, N. Kuechlin, G. Lausen, C. Pinkel. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In Proc. ISWC, 2008. To appear.
M. Schmidt, T. Hornung, G. Lausen, C. Pinkel. SP2Bench - A SPARQL Performance Benchmark. TR arXiv [DB]: 0806.4627v1.
G. Lausen, M. Meier, and M. Schmidt. SPARQLing Constraints for RDF. In *EDBT*, 2008.

A. Chebotko et al.. Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns. TR-DB-052006-CLJF.
C. Bizer and R. Cyganiak. D2R Server publishing the DBLP Bibliography Database, 2007. http://www4.wiwiss.fu-berlin.de/dblp/.
C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/.
R. Cyganiac. A relational algebra for SPARQL. Technical Report, HP Laboratories Bristol.
D.J. Abadi et al.. Scalable Semantic Web Data Management Using Vertical Partitioning. In *VLDB*, pages 411–422, 2007.
D..J. Abadi et al.. Using the Barton libraries dataset as an RDF benchmark. TR MIT-CSAIL-TR-2007-036, MIT.
J. Gray. The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, 1993.
S. Groppe, J. Groppe, and V. Linnemann. Using an Index of Precomputed Joins in order to speed up SPARQL Processing. In *ICEIS*, 2007.
S. Harris and N. Gibbins. 3store: Efficient Bulk RDF Storage. In *PSSS*, 2003.
A. Harth and S. Decker. Optimized Index Structures for Querying RDF from the Web. In *LA-WEB*, 2005.
O. Hartwig and R. Heese. The SPARQL Query Graph Model for Query Optimization. In *ESWC*, 2007.
M. Ley. DBLP Database. http://www.informatik.uni-trier.de/ ley/db/.
J. Perez, M. Arenas, and C. Gutierrez: Semantics and Complexity of SPARQL. In CoRR cs.DB/0605124, 2006.
A. Polleres: From SPARQL to Rules (and back). In WWW, pages 787-796, 2007.
L. Sidirourgos, R. Gocalves, M. Kerstin, N. Nes, and S. Manegold: Column-store support for rdf data management: not all swans are white. In VLDB 2008.

MonetDB: http://monetdb.cwi.nl/.
Sesame: http://www.openrdf.org/