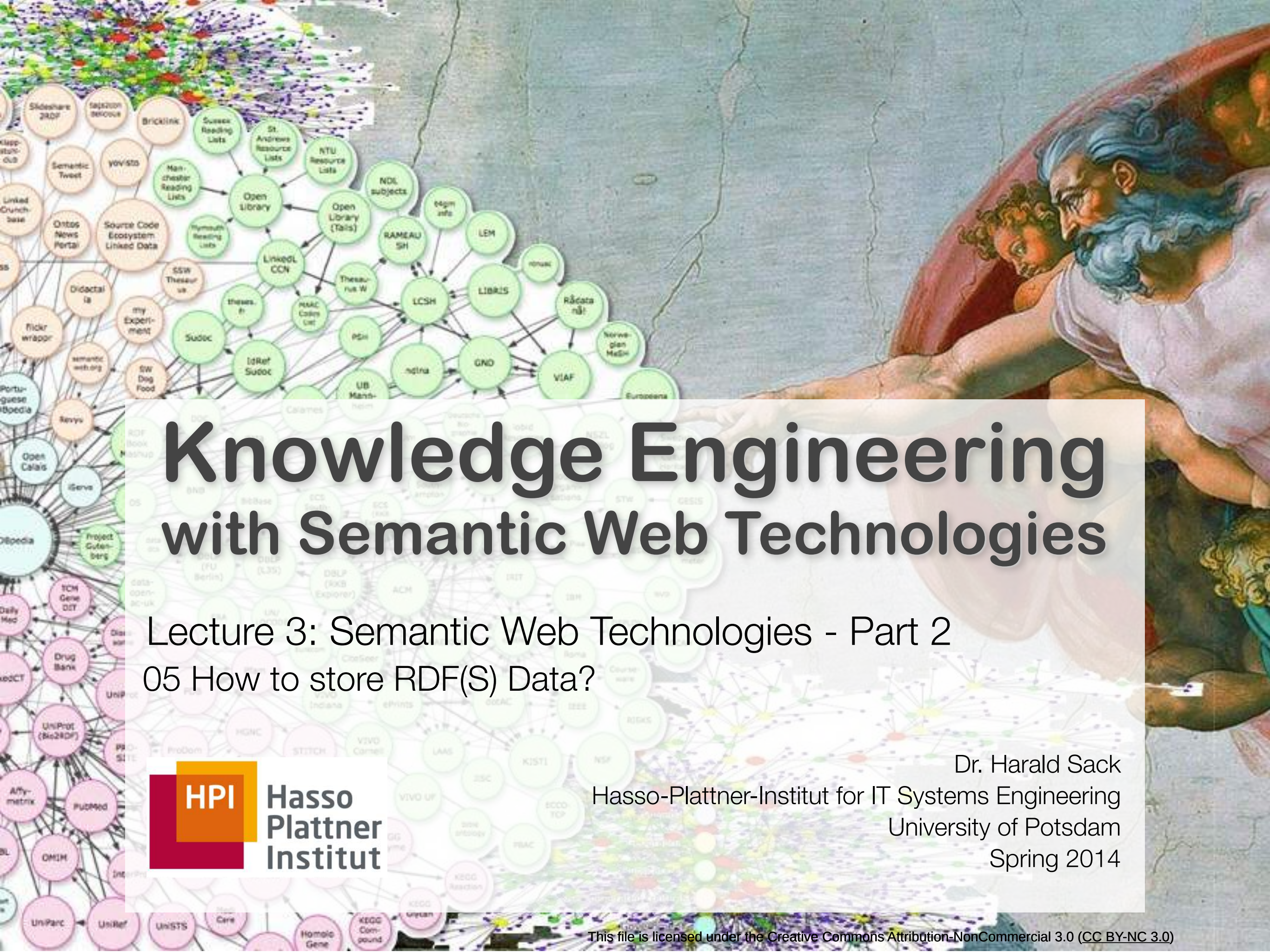




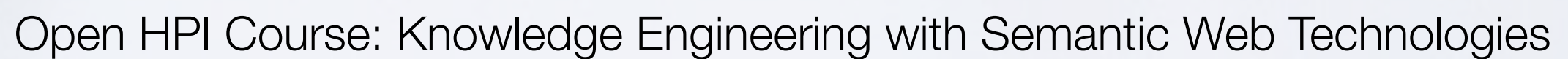
Knowledge Engineering with Semantic Web Technologies

Lecture 3: Semantic Web Technologies - Part 2

05 How to store RDF(S) Data?



Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014



How to store RDF Data?

05 How to store RDF(S) Data?

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 03: Semantic Web Technologies Pt.2

RDF in RDBMS

- RDF Graph can be represented by a set of Triples (s,p,o)
- Triples can simply be stored in a **relational database management system** (RDBMS)
- **Motivation:** Use a specific relational schema for RDF data and benefit from 40 years of research in the DB community
- **3 steps for SPARQL query processing:**
 - (1) Convert SPARQL query to SQL query (w.r.t. the schema)
 - (2) Use RDBMS to answer SQL query
 - (3) Generate SPARQL query result from SQL query result

Problem

- How to store RDF triples to carry out efficient SPARQL queries?
- 4 alternatives:
 1. Giant Triple Storage
 2. Property Tables
 3. Vertically Partitioned Tables (Binary Tables)
 4. Hexastore

Giant Triple Storage

- Basic idea:
 - Store all RDF triples in a single table
 - performance depends on efficient indexing

Subj	Property	Obj
ID1	type	FullProfessor
ID1	teacherOf	'AI'
ID1	bachelorFrom	'MIT'
ID1	mastersFrom	'Cambridge'
ID1	phdFrom	'Yale'
ID2	type	AssocProfessor
ID2	worksFor	'MIT'
ID2	teacherOf	'DataBases'
ID2	bachelorsFrom	'Yale'
ID2	phdFrom	'Stanford'
ID3	type	GradStudent
ID3	advisor	ID2
ID3	teachingAssist	'AI'
ID3	bachelorsFrom	'Stanford'
ID3	mastersFrom	'Princeton'
ID4	type	GradStudent
ID4	advisor	ID1
ID4	takesCourse	'DataBases'
ID4	bachelorsFrom	'Columbia'

Pros:

- easy to implement
- works for huge numbers of properties, if indexes are chosen with care

Cons:

- many self joins

Giant Triple Storage

- Basic idea:
 - Store all RDF triples in a single table
 - performance depends on efficient indexing

Subj	Property	Obj
ID1	type	FullProfessor
ID1	teacherOf	'AI'
ID1	bachelorFrom	'MIT'
ID1	mastersFrom	'Cambridge'
ID1	phdFrom	'Yale'
ID2	type	AssocProfessor
ID2	worksFor	'MIT'
ID2	teacherOf	'DataBases'
ID2	bachelorsFrom	'Yale'
ID2	phdFrom	'Stanford'
ID3	type	GradStudent
ID3	advisor	ID2
ID3	teachingAssist	'AI'
ID3	bachelorsFrom	'Stanford'
ID3	mastersFrom	'Princeton'
ID4	type	GradStudent
ID4	advisor	ID1
ID4	takesCourse	'DataBases'
ID4	bachelorsFrom	'Columbia'

```
SELECT ?university WHERE {  
    ?v rdf:type :GradStudent;  
    :bachelorsFrom ?university . }  
SPARQL
```

```
SELECT t2.o AS university  
FROM triples AS t1, triples AS t2  
WHERE t1.p='type' AND  
      t1.o='GradStudent' AND  
      t2.p='bachelorsFrom' AND  
      t1.s=t2.s  
SQL
```

ID Based Triple Storage

- Use numerical identifier for each RDF term in the dataset
- saves space and enhances efficiency

RDF Term	ID
:ID1	1
rdf:type	2
:FullProfessor	3
:teacherOf	4
“AI”	5
:bachelorsFrom	6
“MIT”	7
:mastersFrom	8
“Cambridge	9

s	p	o
1	2	3
1	4	5
1	6	7
1	8	9
1	10	11
12	13	14
12	15	7
12	4	16

Quad Tables

- Storing **multiple** RDF graphs
- used for provenance, versioning, contexts, etc.

RDF Term	ID
:ID1	1
rdf:type	2
:FullProfessor	3
:teacherOf	4
"AI"	5
:bachelorsFrom	6
"MIT"	7
:mastersFrom	8
"Cambridge"	9

g	s	p	o
100	1	2	3
100	1	4	5
101	12	16	7
100	1	8	9
102	23	21	11
100	1	6	7
102	23	22	8
101	12	4	16

Property Tables

- Combining all (or some) properties of similar subjects in n-ary tables
- Use ID based encoding for efficiency

Professors						
ID	type	teacherOf	bachelorsFrom	mastersFrom	phdFrom	worksFor
ID1	FullProfessor	'AI'	'MIT'	'Cambridge'	'Yale'	NULL
ID2	AssocProfessor	'DataBases'	'Yale'	NULL	'Stanford'	'MIT'
Students						
ID	type	advisor	bachelorsFrom	mastersFrom	teachingAssist	takesCourse
ID3	GradStudent	ID2	'Stanford'	'Princeton'	'AI'	NULL
ID4	GradStudent	ID1	'Columbia'	NULL	NULL	'DataBases'

Pros:

- Fewer joins
- If the data is structured, we have a relational DB

Cons:

- Potentially a lot of NULLs
- Clustering is not trivial
- Multi-value properties are complicated

Vertically Partitioned Tables (Binary Tables)

- for each unique property create a two column table
- Use ID based encoding for efficiency

type	
ID1	FullProfessor
ID2	AssocProfessor
ID3	GradStudent
ID4	GradStudent

teacherOf	
ID1	'AI'
ID2	'DataBases'

bachelorsFrom	
ID1	'MIT'
ID2	'Yale'
ID3	'Stanford'
ID4	'Columbia'

mastersFrom	
ID1	'Cambridge'
ID3	'Princeton'

phdFrom	
ID1	'Yale'
ID2	'Stanford'

worksFor	
ID2	'MIT'

advisor	
ID3	ID2
ID4	ID1

bachelorsFrom	
ID1	'MIT'
ID2	'Yale'
ID3	'Stanford'
ID4	'Columbia'

teachingAssist	
ID3	'AI'

takesCourse	
ID4	'DataBases'

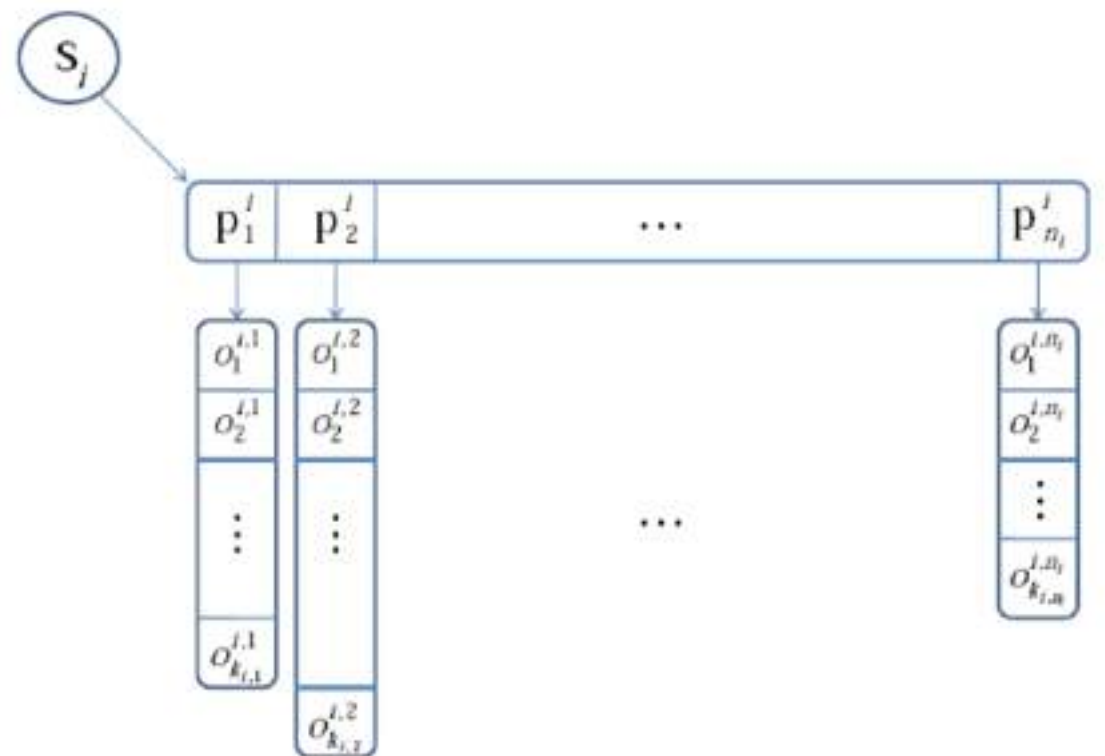
Pros:

- Supports multi-value properties
- No NULLs
- Read only needed attributes (i.e. less I/O)
- No clustering
- Excellent performance (if number of properties is small, queries with bounded properties)

Cons:

- Expensive inserts
- Bad performance (large number of properties, queries with unbounded properties)

- Create an index for every possible combination to enable efficient processing
 - spo, pos, osp, sop, pso, ops



The **spo** index :

Subject key S_i points to sorted vector of n_i property keys, $\{p_1^i, p_2^i, \dots, p_{n_i}^i\}$.

Each property key p_j^i is linked to a sorted list of object keys.

The object key lists are shared with the **pso** index.

Hexastores

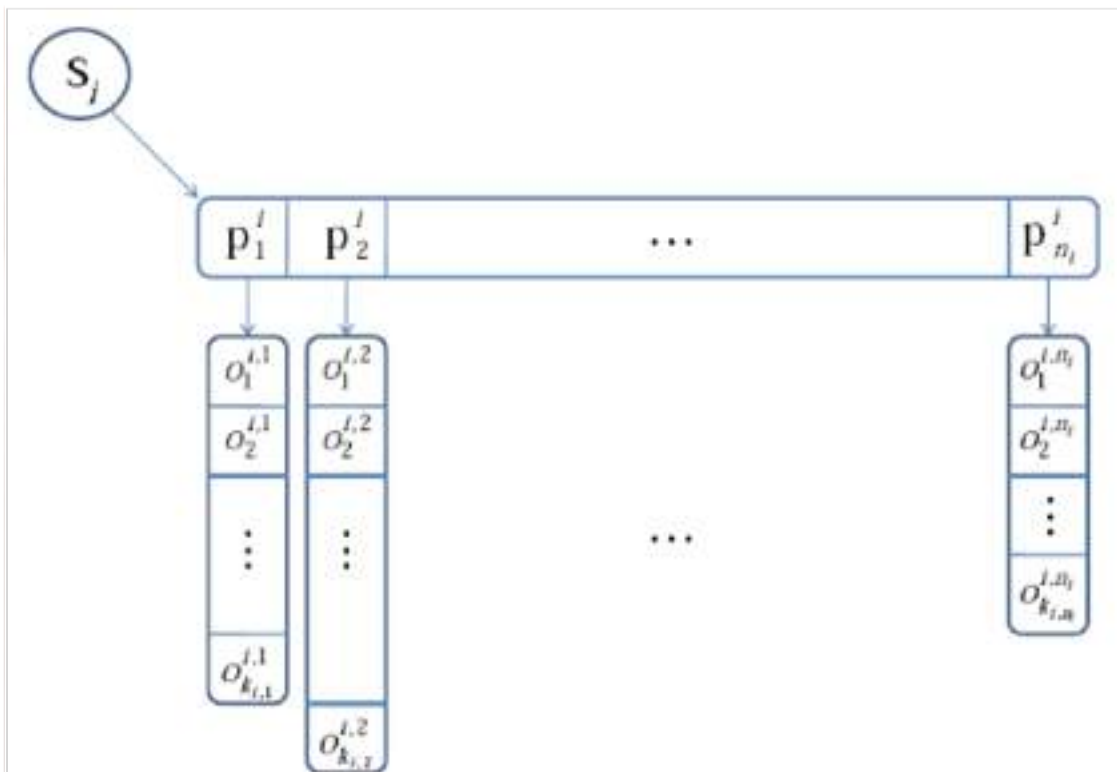
- Create an index for every possible combination to enable efficient processing
 - spo, pos, osp, sop, pso, ops

Pros:

- fast joins (in the beginning)

Cons:

- 5 times more storage
- weak performance, when disk access is necessary



Triple Stores

- Apache Jena - <http://jena.apache.org>
 - Open Link Virtuoso - <http://virtuoso.openlinksw.com>
 - Sesame - <http://www.openrdf.org>
 - SwiftOWLIM - <http://www.ontotext.com/owlim>
 - AllegroGraph - <http://www.franz.com/agraph/allegrograph/>
 - and many more...
-
- W3C maintains a list of triple stores:
http://www.w3.org/wiki/SemanticWebTools#RDF_Triple_Store_Systems



06 Semantic Meta Data and the Web

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 03: Semantic Web Technologies Pt.2

