Semantic Web Technologies

Topic: RDF Triple Stores

Olaf Hartig

olaf.hartig@liu.se

Acknowledgement: Some slides in this slide set are adaptations of slides of Olivier Curé (University of Paris-Est Marne la Vallée, France)

LINKÖPING UNIVERSITY

Overview

- Classification of Triple Stores
- Production-Ready Triple Stores
- Full-Text Search in Triple Stores
- Automated Reasoning in Triple Stores

image source: http://maxpixel.freegreatpicture.com/Watch-To-Find-See-Overview-Search-Binoculars-1026424

Before we begin …

… a reminder of database-related terminology

- **Data**: known facts that can be recorded and that have implicit meaning
- **Database**: logically coherent collection of related data
  - Built for a specific purpose
  - Represents some aspects of the real world
- **Database management system** (**DBMS**): collection of computer programs to create and maintain a database
  - Protects DB against unauthorized access and manipulation
  - Examples of relational DBMSs: *Microsoft's SQL Server, IBM's DB2, Oracle, MySQL, PostgreSQL*
- Now, DBMSs for RDF data are called **triple stores**

Classification of *RDF Triple Stores**

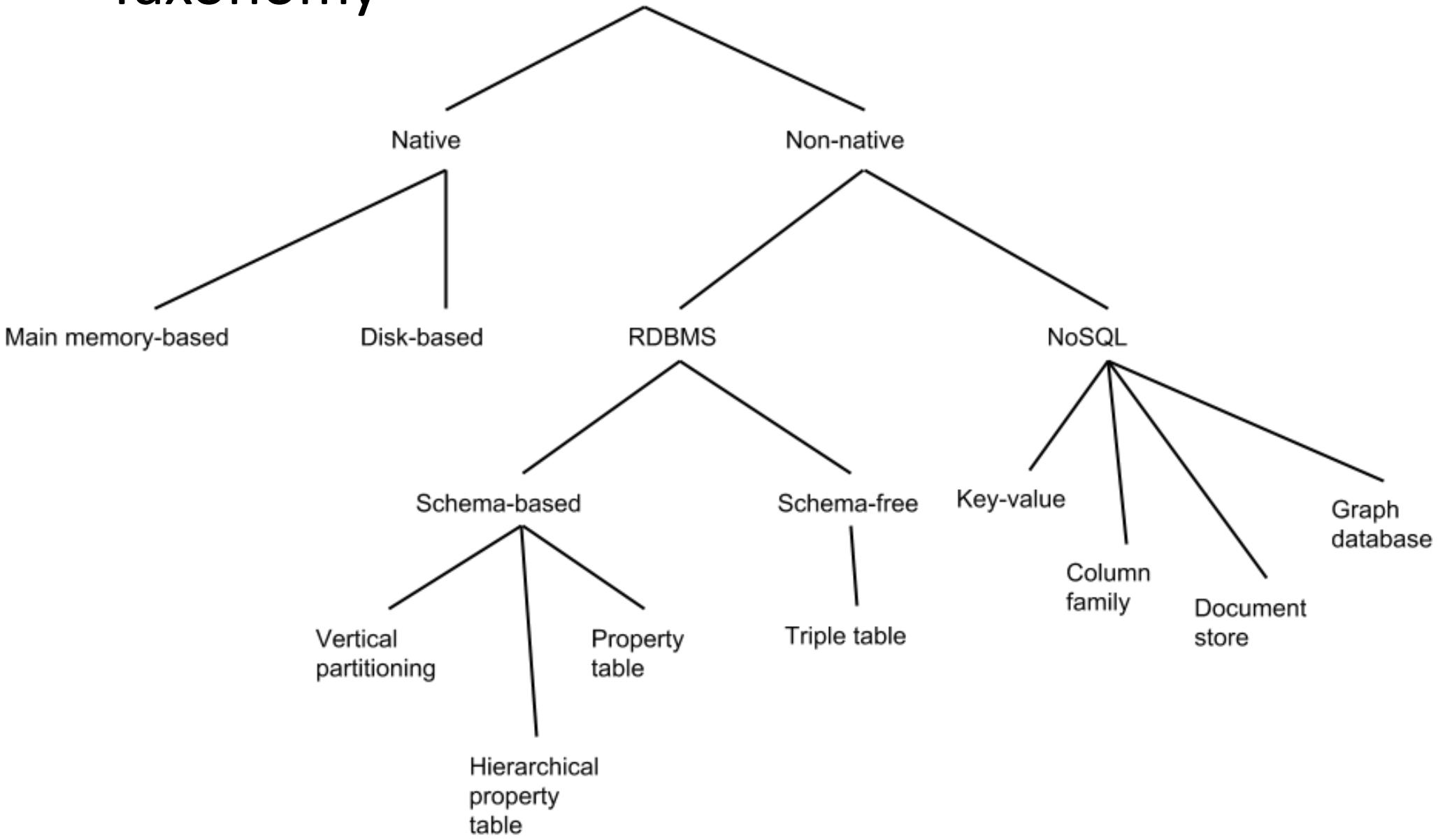
**Triple store* = DBMS for RDF data

RDF Storage

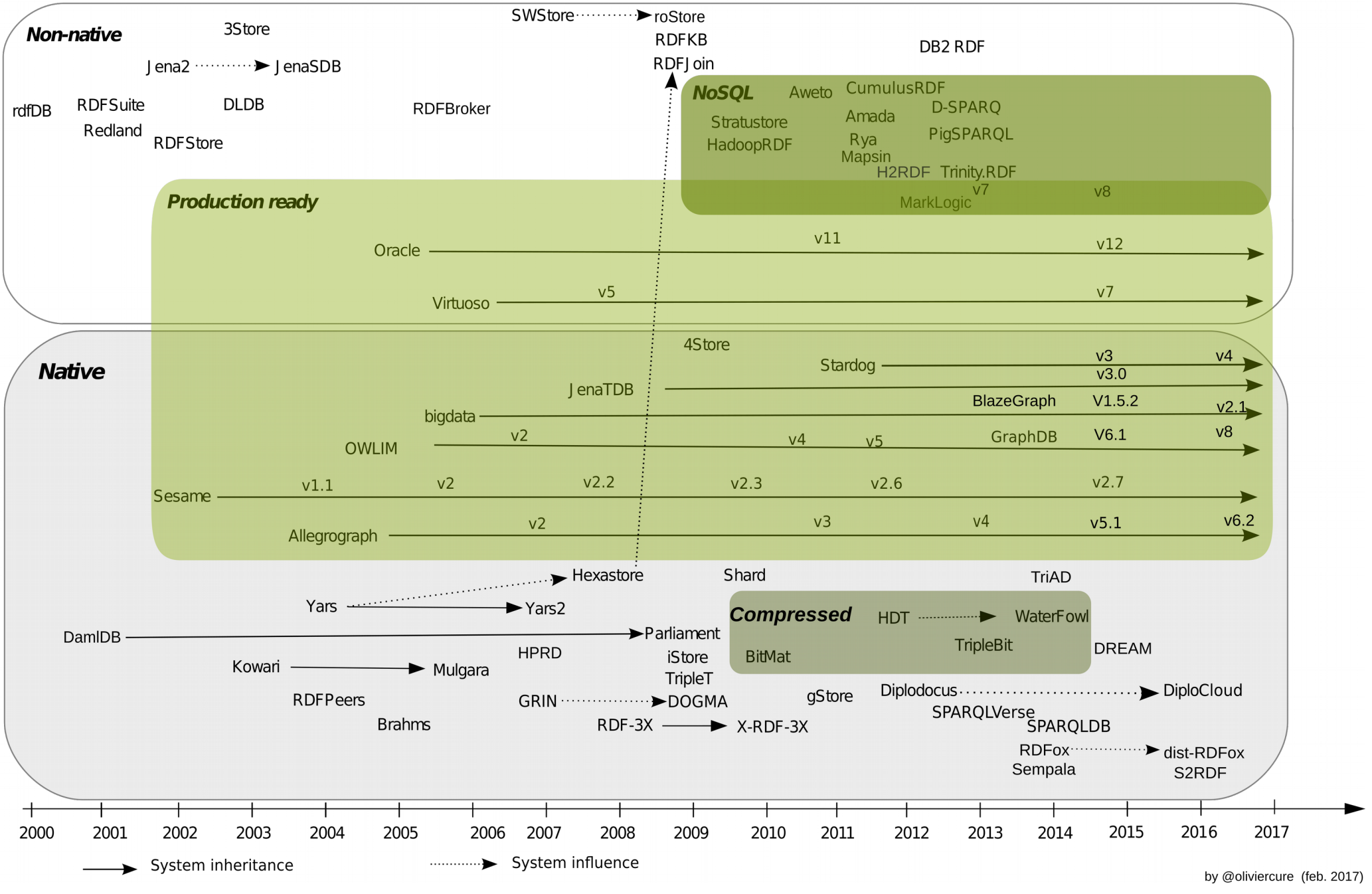
- RDF is a logical data model and, thus, does not impose any physical storage solution
- Existing triple stores are either
  - designed from scratch (“native”)

    or
  - based on an existing DBMS
    - Relational model, e.g., PostgreSQL
    - NoSQL, e.g., Cassandra

Taxonomy

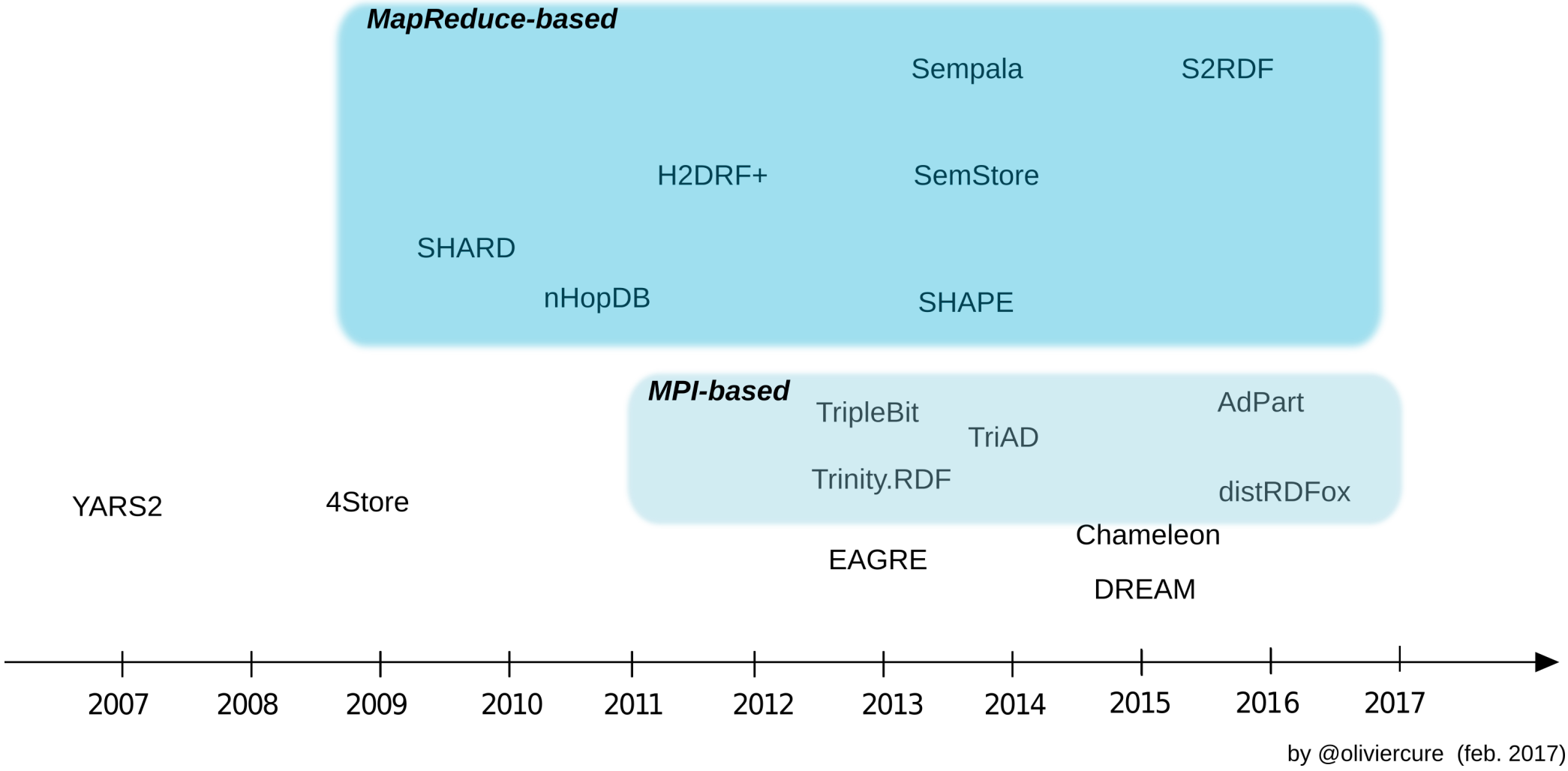

Native
Non-native
Main memory-based
Disk-based
RDBMS
NoSQL
Schema-based
Schema-free
Key-value
Graph database
Column family
Document store
Vertical partitioning
Property table
Triple table
Hierarchical property table

Timeline of Triple Store Proposals



by @oliviercure (feb. 2017)

Prototypes of Distributed Triple Stores



by @oliviercure (feb. 2017)

Production-Ready Triple Stores

Overview

Name
Allegrograph
Blazegraph
GraphDB
MarkLogic
Oracle
Stardog
Virtuoso

AllegroGraph
Franz Inc.

Transactions with ACID Properties

Name
Allegrograph ✓
Blazegraph ✓
GraphDB ✓
MarkLogic ✓
Oracle ✓
Stardog ✓
Virtuoso ✓

- **A**tomicity: a transaction (TA) is an atomic unit of processing; it is either performed in its entirety or not performed at all
- **C**onsistency preservation: a correct execution of a TA must take the DB from one consistent state to another
- **I**solation: even if TAs are executing concurrently, they should appear to be executed in isolation; that is, their final effect should be as if each TA was executed alone from start to end
- **D**urability: once a TA is committed, its changes applied to the database must never be lost due to subsequent failure

Cluster Setups

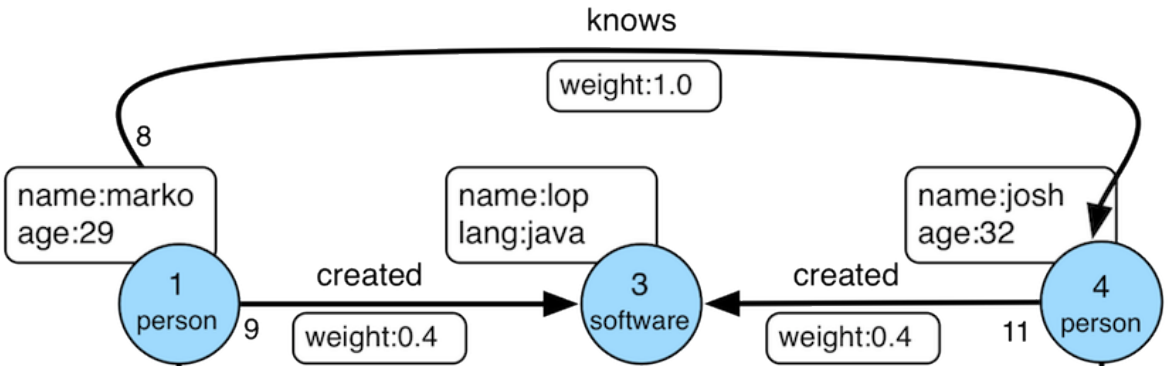
Name
Allegrograph ✓
Blazegraph ✓
GraphDB ✓
MarkLogic ✓
Oracle ✓
Stardog ✓
Virtuoso ✓

- Replication: mostly master-slave, some master-master
- Partitioning: range, hash

Support for other Data Models (besides RDF)

Name
Allegrograph
Blazegraph ✓
GraphDB ✓
MarkLogic ✓
Oracle ✓
Stardog ✓
Virtuoso ✓

- Relational Model (with SQL)
  - Virtuoso, Oracle
- XML (with XQuery)
  - MarkLogic, Virtuoso
- Document Model
  - MarkLogic
- Property Graphs (with Gremlin)
  - Blazegraph, GraphDB, Stardog



Licenses

Name
Allegrograph
Blazegraph
GraphDB
MarkLogic
Oracle
Stardog
Virtuoso

- Most of these systems have a free-to-use edition, some even have a feature-limited free software version (open source)
  - Allegrograph: free (of charge) edition with 5M triples limit
  - Blazegraph: free for a single machine
  - GraphDB: free (of charge) edition without clustering and replication
  - MarkLogic: dev license is free for up to 1TB and max 10 months
  - Stardog: community ed. (max 10 DBs with max 25M triples per DB, 4 users)
  - Virtuoso: free w/o clustering and replication
- All have commercial editions

Full-Text Search Support

Name	Full-text search
Allegrograph	Integrated + Solr
Blazegraph	Integrated + Solr
GraphDB	Integrated + Solr + ElasticSearch (enterp.)
MarkLogic	Integrated
Oracle	Integrated
Stardog	Integrated + Lucene
Virtuoso	Integrated

Cloud Readyness

Name	Full-text search	Cloud-ready
Allegrograph	Integrated + Solr	AMI
Blazegraph	Integrated + Solr	AMI
GraphDB	Integrated + Solr + ElasticSearch (enterp.)	AMI
MarkLogic	Integrated	AMI
Oracle	Integrated	
Stardog	Integrated + Lucene	AMI
Virtuoso	Integrated	AMI

AMI: Amazon Machine Image

Other, cloud-native options:

- Dydra
- Amazon Neptune

Other Features

Name	Full-text search	Cloud-ready	Extra features
Allegrograph	Integrated + Solr	AMI	
Blazegraph	Integrated + Solr	AMI	“Reification done right” (RDF*)
GraphDB	Integrated + Solr + ElasticSearch (enterp.)	AMI	RDF ranking
MarkLogic	Integrated	AMI	XQuery; Javascript
Oracle	Integrated		Inline in SQL
Stardog	Integrated + Lucene	AMI	Integrity constraints; explanations
Virtuoso	Integrated	AMI	Inline in SQL

Full-Text Search in Triple Stores

Goal

- Query a dataset by using keywords
  - Full-text search

- Typical use cases are related to datasets that contain literals with (large) texts

image source: https://www.flickr.com/photos/projectart69/6704263029

What is Full-Text Search?

- Retrieve text documents out of a large collection
- Query is an unordered set of tokens (seq. of chars)
- Result is a set of documents relevant to the query
- Relevance may be *boolean*
  - i.e., document contains all tokens or not

  or it is *degree-based*

  - relevance of a document usually measured by taking into account the frequency of tokens in it, normalized by frequency in all documents
  - in this case, result set is ordered

Options

1. Use a full-text search engine
   (as a separate component in the software stack)
2. Use full-text search features built into triple stores
   - Native full-text search functionality
   - Integration of external search engines

Popular Full-Text Search Engines

- Apache Lucene is a Java-based full-text indexing and search library with a lot of features

- ElasticSearch is a distributed full-text search engine built on Lucene

- Apache Solr is another distributed full-text search engine, also built on Lucene

Options

1. Use a full-text search engine
   (as a separate component in the software stack)
2. Use full-text search features built into triple stores
   - Native full-text search functionality
   - Integration of external search engines

Triple store	Integrated	external
MarkLogic	x	
Virtuoso	x	
Allegrograph	x	Solr (1.5.2)
Stardog	x	Lucene
GraphDB SE	x	Lucene Solr, ElasticSearch (Enterprise)
BlazeGraph	x	SolR
Oracle 12c	x	

Native Full-Text Search in Blazegraph

- Built-in full-text search feature is custom-built
- Enabled by default in the configuration file
  `com.bigdata.rdf.store.AbstractTripleStore.textIndex=true`
- B+Tree over tokens extracted from each RDF literal added to the database
  - Fast exact match on tokens
  - Fast prefix match
  - Fast match on multiple tokens
  - No performance gains for arbitrary regular expressions

Native Full-Text Search in Blazegraph

- Integration into SPARQL via the **`bds:search`** predicate

```
prefix bds: <http://www.bigdata.com/rdf/search#>
SELECT ?s ?p ?o WHERE {
  ?o bds:search "dog" .
  ?s ?p ?o .
}
```

Native Full-Text Search in Blazegraph

- Integration into SPARQL via the **`bds:search`** predicate and other related predicates

```
prefix bds: <http://www.bigdata.com/rdf/search#>
SELECT ?s ?p ?o ?score ?rank WHERE {
   ?o bds:search "dog cat" .
   ?o bds:matchAllTerms "true" .
   ?o bds:minRelevance "0.25" .
   ?o bds:relevance ?score .
   ?o bds:maxRank "1000" .
   ?o bds:rank ?rank .
   ?s ?p ?o .
}
```

Only literals that contain all of the specified search terms are to be considered

Using External Solr Services in Blazegraph

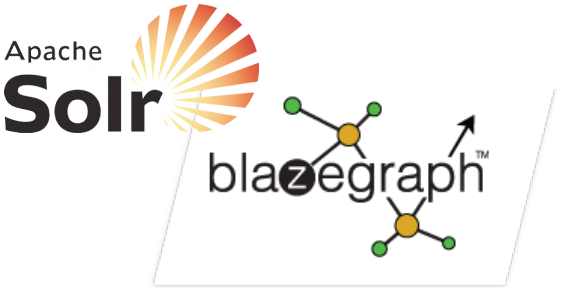
- Access to an external Solr service from within a SPARQL query is supported out of the box

```
prefix fts: <http://www.bigdata.com/rdf/fts#>
SELECT ?person ?kwDoc ?snippet WHERE {
  ?person rdf:type ex:Artist .
  ?person rdfs:label ?label .
  SERVICE <http://www.bigdata.com/rdf/fts#search> {
   ?kwDoc fts:search ?label .
   ?kwDoc fts:endpoint "http://my.solr.server/solr/select" .
   ?kwDoc fts:params "fl=id,score,snippet" .
   ?kwDoc fts:scoreField "score" .
   ?kwDoc fts:score ?score .
   ?kwDoc fts:snippetField "snippet" .
   ?kwDoc fts:snippet ?snippet .  }
} ORDER BY ?person ?score
```

Using External Solr Services in Blazegraph

- Use BIND to construct more complex search query

```
...
?person rdfs:label ?label .
BIND(CONCAT("\"", ?label,
            "\" AND -\"expressionism\"") AS ?search)
SERVICE <http://www.bigdata.com/rdf/fts#search> {
 ?kwDoc fts:search ?search .
 ...
```



Native Full-Text Search in Virtuoso

- Objects of RDF triples with a given predicate or in a given graph can get indexed for full-text search
- Full-text index is in batch mode by default
  - Changes in triples are reflected in the index periodically (i.e., no strict synchronization)
  - Configuration option to enforce synchronization
- Powerful grammar for full-text queries – examples:

```
dogs AND cats
vet AND (dog OR cat)
dog AND NOT (dog NEAR cat)
"dog h*"
```

Native Full-Text Search in Virtuoso

- RDF triples whose object has been indexed can be found in SPARQL using the predicate **`bif:contains`**
  - ```
    SELECT * WHERE {
       ?s foaf:name ?name .
       ?name bif:contains '"rich*"'.
    }
    ```
  - ```
    SELECT * WHERE {
       ?s ?p ?o .
       ?o bif:contains 'New AND York'
       OPTION (score ?sc) .
    }
    ORDER BY DESC (?sc)
    LIMIT 10
    ```

Native Full-Text Search in AllegroGraph

- Full-text search via API and in SPARQL queries
- Syntax of full-text search queries:
  - Wildcards: `?` (single char.), `*` (multiple chars)
  - Boolean operators: `and`, and `or`
  - Double quotes around an exact phrase to match
- Multiple full-text indexes possible
- Each index works with one or more predicates, including an option to index all predicates
- Each index can be configured to include:
  - All literals, no literals, or specific types of literals
  - Full URI, just the local part, or ignore URIs entirely
  - Any combination of the four parts of a triple (incl. G)

AllegroGraph
Franz Inc.

Solr Integration in AllegroGraph

- External full-text search by using Apache Solr
  - Solr server must be installed and started separately
  - Inserts, updates, and deletes in the Solr database must be done in the application logic
- Solr features that the native solution does not have:
  - Faceted search
  - Finding words close together
  - Relevancy ranking and word boosting
  - Text clustering
  - Hit highlighting

AllegroGraph
Franz Inc.

Apache Solr

Solr Integration in AllegroGraph (cont'd)

- Storage strategy for an RDF triple such as:

```
ex:someSubj ex:somePred "text to index"
```

  - Tell Solr to associate `"text to index"` with a new `id`
  - Then, add a new triple into AllegroGraph:

```
ex:someSubj <http://www.franz.com/solrDocId> id
```

- Now, you may write a SPARQL query such as:

```
PREFIX solr: <http://www.franz.com/ns/allegrograph/4.5/solr/>
PREFIX franz: <http://franz.com/ns/allegrograph/4.5/>
SELECT * WHERE {
  ?s solr:match 'medicate disastrous' .
  ?s franz:text ?text .
  ?s otherProperty ?other . }
```

- Solr can also be used from the API and the CLI

Native Full-Text Search in Stardog

- Based on Lucene
- Creation of a “search document” per RDF literal
- Disabled by default, must be enabled:

```
stardog-admin db create -o
                    search.enabled=true -n myDb
```

- Three modes for rebuilding indexes, configured by setting `search.reindex.mode` to:
  - *sync* (synchronous rebuild with a transacted write, dflt.)
  - *async* (asynchronous rebuild “as soon as possible”), or
  - *scheduled* (cron expression specifies when to rebuild)

Native Full-Text Search in Stardog (cont'd)

- Search syntax as in Lucene:
  - e.g., wildcards `?` and `*`, fuzzy with similarity `~0.5`
- Use it on the command line:

```
stardog query search -q "html" -l 10 myDb
```

- Use it in SPARQL:

```
SELECT DISTINCT ?s ?score WHERE {
?s ?p ?l.
(?l ?score) <tag:stardog:api:property:textMatch>
                                        "html"}

SELECT DISTINCT ?s ?score WHERE {
?s ?p ?l.
(?l ?score) <tag:stardog:api:property:textMatch>
                                  ("html" 0.5 10)}
```

Native Full-Text Search in GraphDB

- Based on Lucene
- For each RDF node, text document that is made up of other nodes reachable from the node ("molecule")

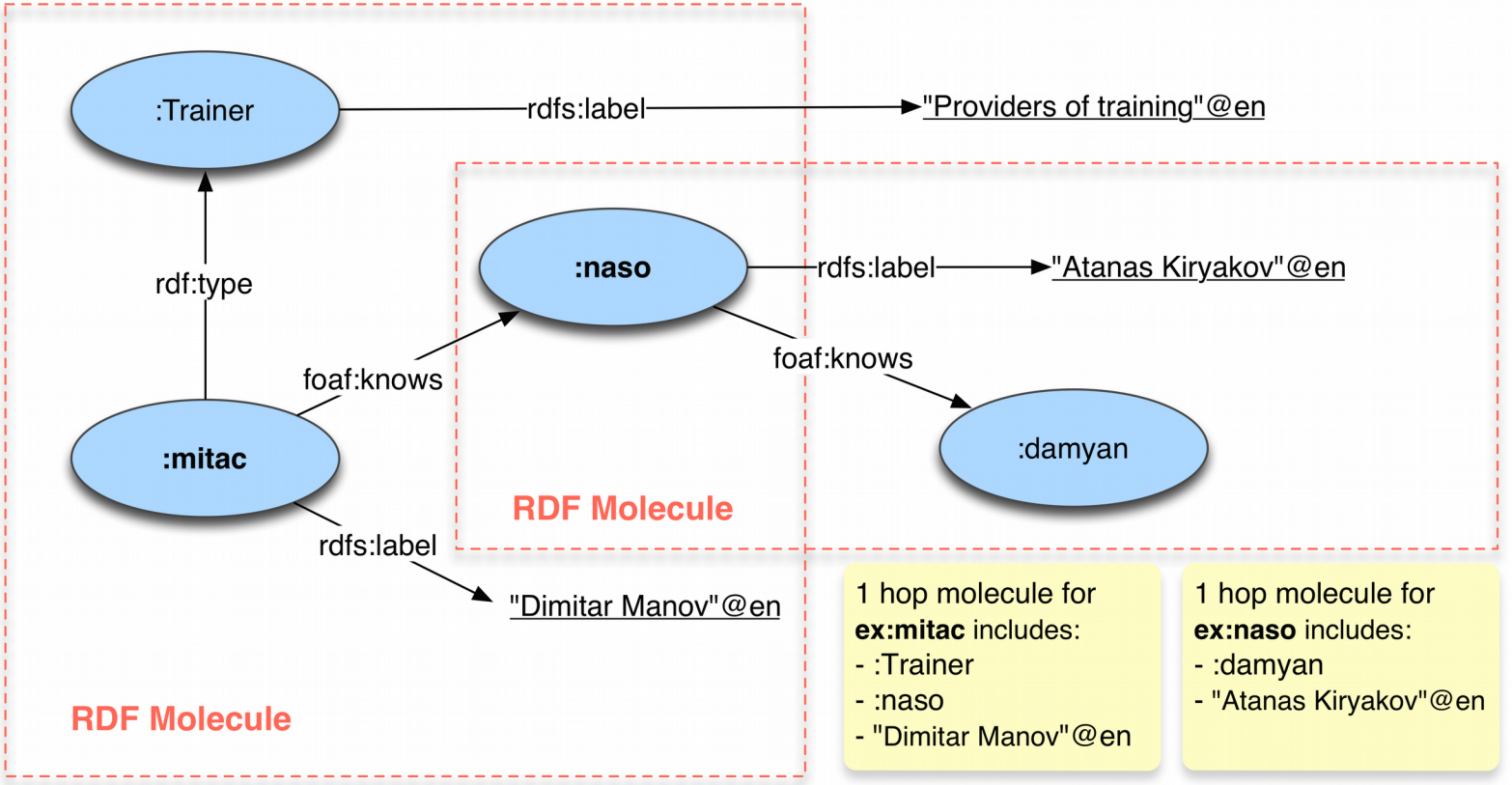


Image source: http://graphdb.ontotext.com/documentation/free/full-text-search.html

Native Full-Text Search in GraphDB

- Based on Lucene
- For each RDF node, text document that is made up of other nodes reachable from the node ("molecule")
- Indexes can be parameterized
  - what kinds of nodes are indexed (URIs / literals)
  - literals with specific language tags only
  - what is included in the notion of "molecule"
  - size of the "molecule" to index
  - relevance of nodes boosted by RDF Rank values
  - alternative analyzers
  - alternative scorers
- Multiple, differently configured full-text indexes possible

Native Full-Text Search in GraphDB (cont'd)

- Setting up an (example) configuration for full-text indexes:

```
PREFIX luc: <http://www.ontotext.com/owlim/lucene#>
INSERT DATA {
  luc:index          luc:setParam  "uris" .
  luc:include        luc:setParam  "literals" .
  luc:moleculeSize   luc:setParam  "1" .
  luc:includePredicates  luc:setParam
     "http://www.w3.org/2000/01/rdf-schema#label" .
}
```

- Creating a new index (uses the previous configuration):

```
PREFIX luc: <http://www.ontotext.com/owlim/lucene#>
INSERT DATA {
  luc:myTestIndex luc:createIndex "true" .
}
```

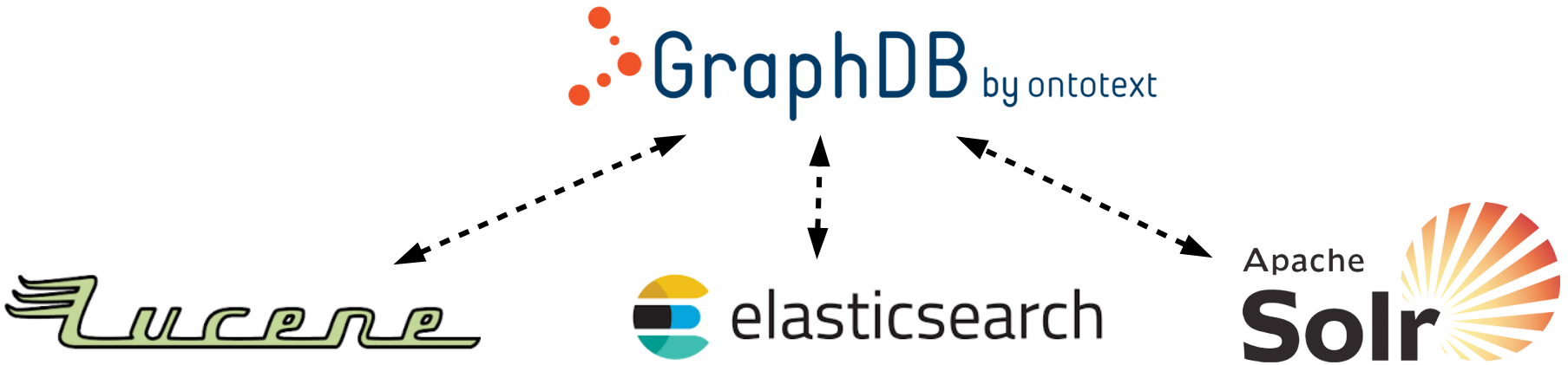
Native Full-Text Search in GraphDB (cont'd)

- Use the index in a query
  - ```
    PREFIX luc: <http://www.ontotext.com/owlim/lucene#>
    SELECT * { ?id luc:myTestIndex "ast*" }
    ```
  - ```
    PREFIX luc: <http://www.ontotext.com/owlim/lucene#>
    SELECT * {
      ?id luc:myTestIndex "lucene query string" .
      ?node luc:score ?score .
    } ORDER BY ( ?score )
    ```
- Incremental update

```
PREFIX luc: <http://www.ontotext.com/owlim/lucene#>
INSERT DATA {
  luc:myTestIndex luc:addToIndex ex:newURI .
}
```

Connecting External Search to GraphDB

- Connectors for Lucene, Solr, and Elasticsearch (the latter two only in the enterprise edition of GraphDB)
- Similar to how Blazegraph supports access to an external Solr service from within a SPARQL query



Native Full-Text Search in MarkLogic

- Built-in full-text search feature is custom-built
- Full-text indexes created when loading a document
- Powerful grammar for string queries
  - Examples:

```
(cat OR dog) NEAR vet
dog NEAR/30 vet
cat -dog
"cats and dogs"
dog NOT_IN "dog house"
dog BOOST cat
```

Automated Reasoning in Triple Stores

Approach 1: *Materialization*

aka *forward reasoning* or *closure*

- Idea: make explicit all inferences in the store
- Pros:
  - Efficient query processing
    (no reasoning at query runtime)
- Cons:
  - Slow data loading
  - Data volume expansion
  - Tricky update management

Approach 2: *Query Rewriting*

aka *backward reasoning* or *query reformulation*

- Idea: reformulate the original query such that all answers can be retrieved
- Pros:
  - No preprocessing overhead
  - No expansion of stored data volume
  - Easy update management
- Cons:
  - Slow query processing due to cost of of reasoning at query runtime

Reasoning in the Production-Ready Systems

Triple store	Materialization	Query rewriting
Allegrograph	OWLRL	RDFS++, Prolog
Blazegraph	RDFS, OWL Lite	
GraphDB	RDFS, OWL Horst, OWLRL, OWLQL	
MarkLogic		RDFS, RDFS++, OWL Horst
Oracle	RDFS, OWLRL, OWLQL	
Stardog	All OWL2	
Virtuoso		RDFS++

Ontology-Based Data Access (OBDA)

Ontology-Based Data Access (OBDA)

- Relevant if you want to access an existing (relational) database in terms of an ontology
- Ontology models the domain, hides the structure of the database, and enriches incomplete data
- Mappings associate concepts and properties of the ontology with SQL views over the database
- Queries expressed in terms of the ontology (using SPARQL) translated into source queries (SQL)
- State-of-the-art systems: Ontop, Capsenta's Ultrawrap

www.liu.se

LINKÖPING UNIVERSITY