



# OWLIM

## Basics and New Features

---

Mar, 2010

# Presentation Outline

---

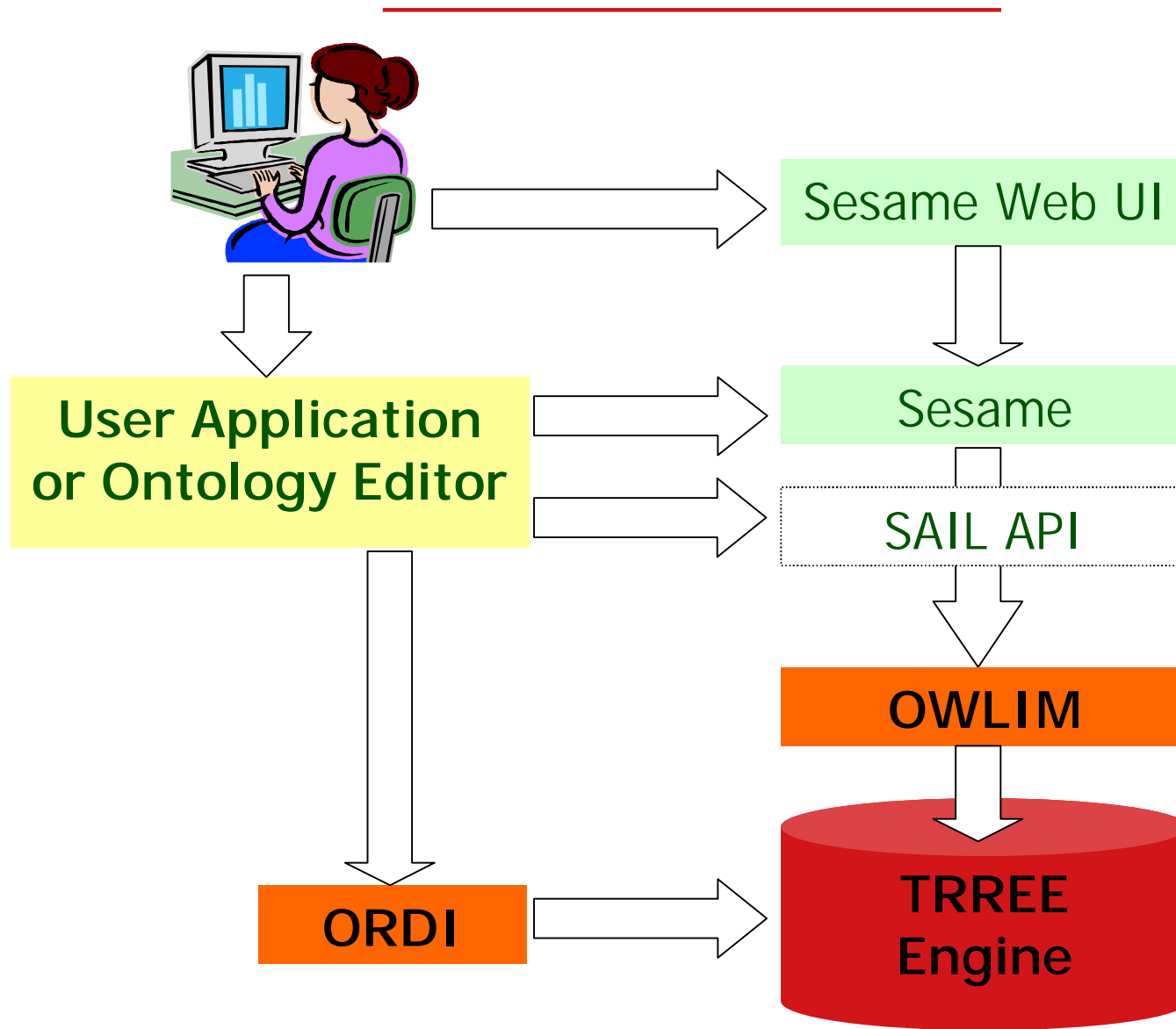
- Introduction
- New features
  - “Smooth” invalidation
  - Inconsistency checking
  - RDFRanks
  - Notifications
  - Replication Cluster

# Semantic Repository for RDFS and OWL

---

- OWLIM is a **scalable semantic repository** which allows
  - Management, integration, and analysis of heterogeneous data
  - Combined with light-weight reasoning capabilities
- OWLIM is RDF database with **high-performance reasoning**:
  - The inference is based on logical **rule-entailment**
  - **Full RDFS and limited OWL Lite and Horst** are supported
  - **Custom semantics** defined via rules and axiomatic triples

# Sesame, TRREE, ORDI, and OWLIM



# “Smooth” Invalidation

---

- Before Jun'09, whenever a statement was deleted from OWLIM, it was deleting the entire deductive closure
  - Which was triggering full re-inference
- A partial invalidation mechanism was implemented:
  - It performs a sequence of backward and forward-chaining iterations to figure out what part of the deductive closure is no longer supported
  - It requires no “truth maintenance information”
- The complexity of the invalidation is comparable to the complexity of the addition (doesn't work for owl:sameAs)
  - It is slower, but it is still in the same order of magnitude
  - Removing “key” statements can be still painful

# Inconsistency Checking

---

- OWLIM supports inconsistency checking rules (two kinds)

**IF <premises> CHECK <constraints>**

- They work like the entailment rules, with the difference that:
  - Whenever OWLIM finds a binding for the variables in the premises (the body) of the rule
  - It checks whether the triples in the head of the rule, given such assignment of the variables, exist in the repository
    - And “reports” inconsistency, if they don’t
  - Instead of adding the new statements from the head of the rule as newly entailed facts in the repository
  - Inconsistency checking is performed at the end of the transaction commit

# RDF Ranking

---

- OWLIM includes a routine which allows for efficient calculation of a modification of PageRank over RDF graphs
- The computation of the RDFRanks for LDSR (400M LOD statements) takes **310 sec**
  - **201 sec.** for reading the RDF graph from disk-based structures into specific in-memory representation
  - **98s** were spent in 27 PageRank iterations
- Results are available through a system predicate
- Example: get the 100 most important nodes in the RDF graph

```
SELECT ?n {?n onto:hasPageRank ?r}  
  
ORDER BY DESC(?r) LIMIT 100
```

# Naso's Favorite Query

---

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX opencyc: <http://sw.opencyc.org/2008/06/10/concept/en/>
PREFIX dbp-ont: <http://dbpedia.org/ontology/>
PREFIX owl: <http://www.ontotext.com/>
PREFIX geo-ont: <http://www.geonames.org/ontology#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
```

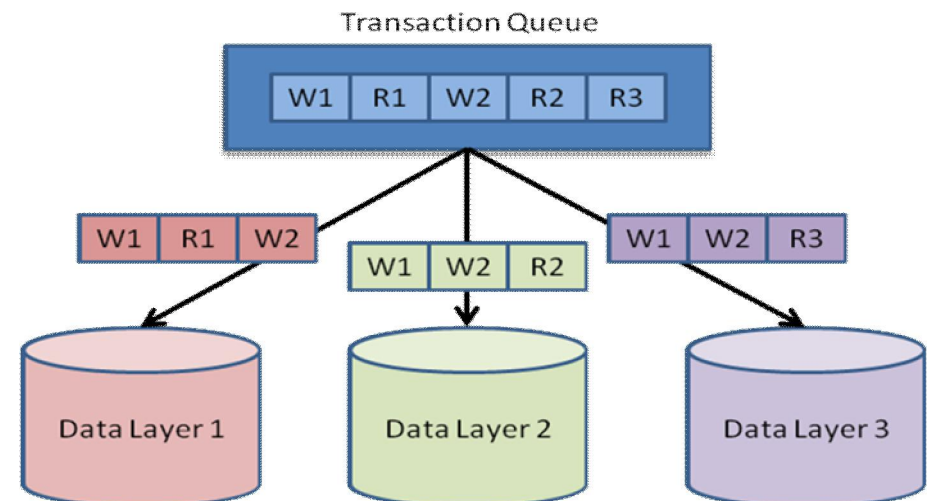
```
SELECT * WHERE {
    ?Person dbp-ont:birthPlace ?BirthPlace ;
            rdf:type opencyc:Entertainer ;
            owl:hasPageRank ?RR .
    ?BirthPlace geo-ont:parentFeature dbpedia:Bavaria .
} ORDER BY DESC(?RR) LIMIT 100
```

- This query involves data from DBPedia, Geonames, and UMBEL (OpenCyc)
- It involves inference over types, sub-classes, and transitive relationships
- Ranking of the results by “importance”
- Without setup like LDSR, getting an answer in real time, would be impossible

# OWLIM Replication Cluster

---

- Distribution through data replication is used to ensure:
  - Better handling of concurrent user requests
  - Failover support
- How does it work?
  - Every user write request is pushed in a transaction queue
  - Each data write request is multiplexed to all repository instances
  - Each read request is dispatched to one of the instance only
  - To ensure load-balancing, each read requests is send to the instance with smallest execution queue at this point in time



## OWLIM Replication Cluster (2)

---

- The total loading/modification performance of the cluster is equal to this of one instance
- The data scalability of the cluster is determined by the amount of RAM of the weakest instance
- The query performance of the cluster represents the sum of the throughputs which can be handled by each of the instances
- Failover:
  - In case of failure of one or more instances, the performance degradation is graceful
  - The cluster is fully operation even when there is only one instance working

# Notifications

---

- The client can subscribe for **notifications for incoming statements matching certain graph pattern**
- In order to define the graph pattern of interest the client provides a SPARQL SELECT query a
  - It should be noted though that such a representation is meant for simplicity and future extensions compatibility rather than to imply full SPARQL support
  - The SPARQL query is expanded into a set of graph patterns which are then used to filter incoming statements and notify the subscriber about those of them that help form a new solution of at least one of the graph patterns.
  - At the current stage, more complicated SPARQL constructs like FILTER, OPTIONAL, DISTINCT, LIMIT, ORDER BY, are to be ignored by the notification service
  - The subscriber should **not rely on any particular order or distinctness** of the statement notifications to follow her subscription

## Notifications (2)

---

- Leaving out the FILTER support implies several limitations
  - SPARQL achieves negative constraints only by means of FILTER(!BOUND(?))
- The implementation of notifications is **monotonous** system
  - A statement that was once reported to meet the requirements of a subscription will continue to meet them forever (or until it is removed from the triple store where it resides)
  - In contrast, a non-monotonous notification system would be a lot more complicated due to the need to report not only statements that compounded the solution of a query but also the statements that stopped compounding any
  - In terms of implementation efficiency such a non-monotonous behavior would require a lot more computational resources and is of dubious value which brought the choice of a monotonous system and therefore the disabling the FILTER capabilities of graph patterns