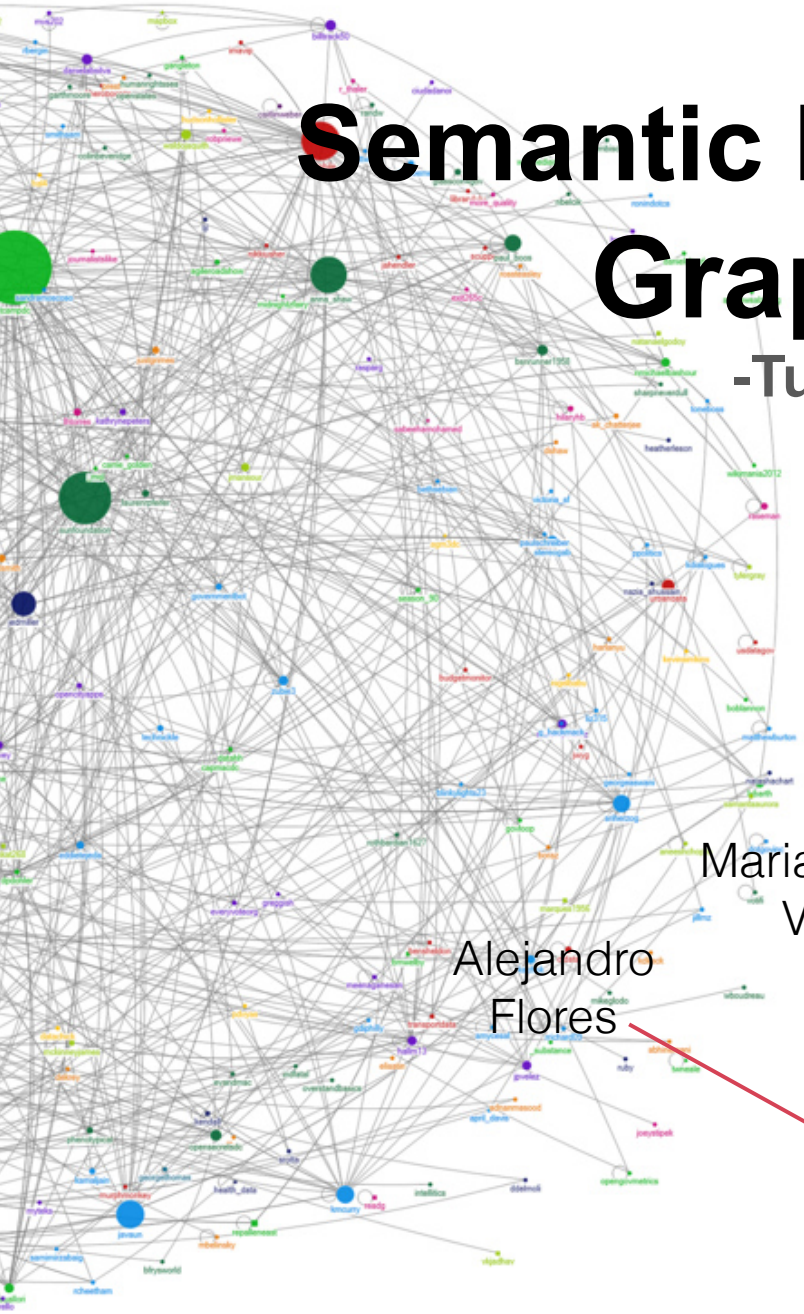


# Semantic Data Management in Graph Databases

## -Tutorial at ESWC 2014-


Maria-Esther Vidal

Edna Ruckhaus

Maribel Acosta

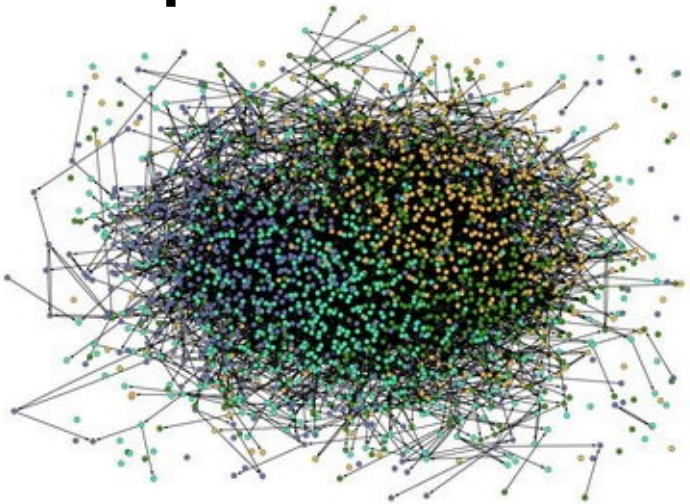
Cosmin Basca

Alejandro Flores

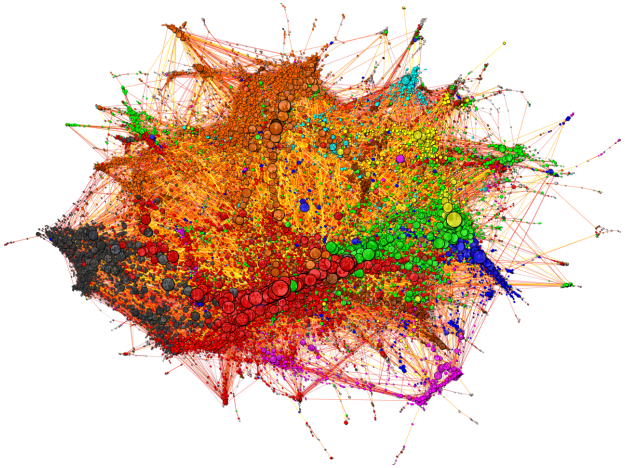
USB

Karlsruhe Institute of Technology

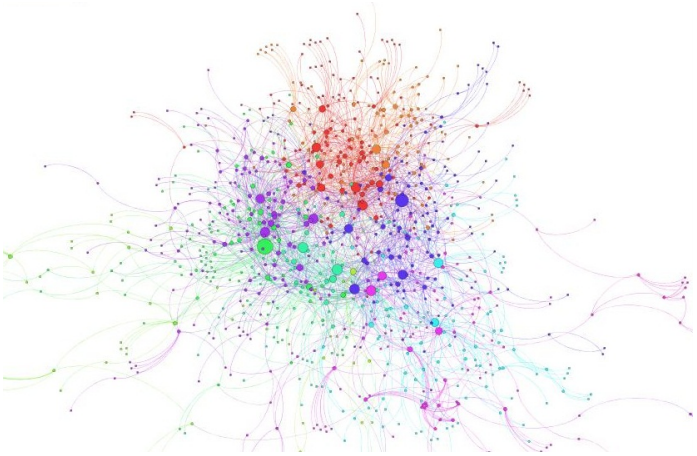
# Graphs …



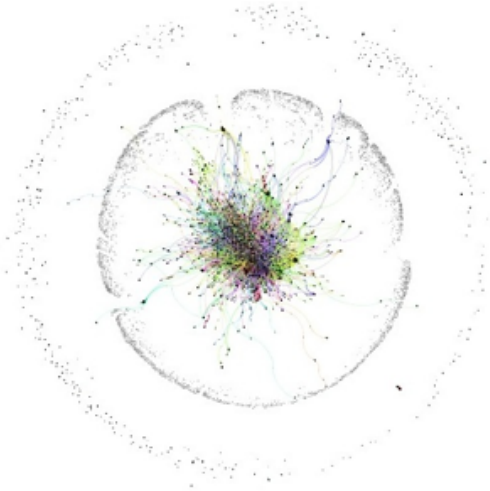
Network of Friends in a High School



Relationships among artists in Last.fm
http://sixdegrees.hu/last.fm/

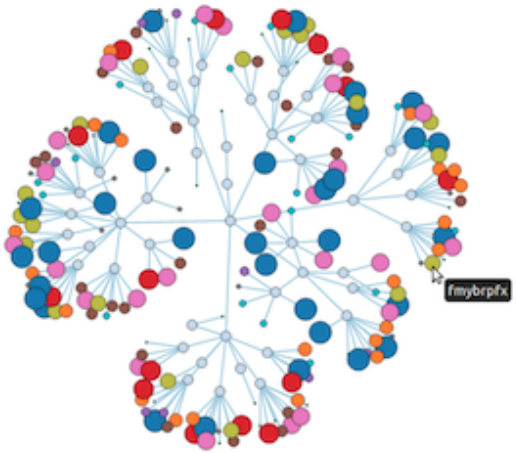


Network structure of music genres and their stylistic origin
http://www.infosysblogs.com/web2/2013/01/network_structure_of_music_gen.html

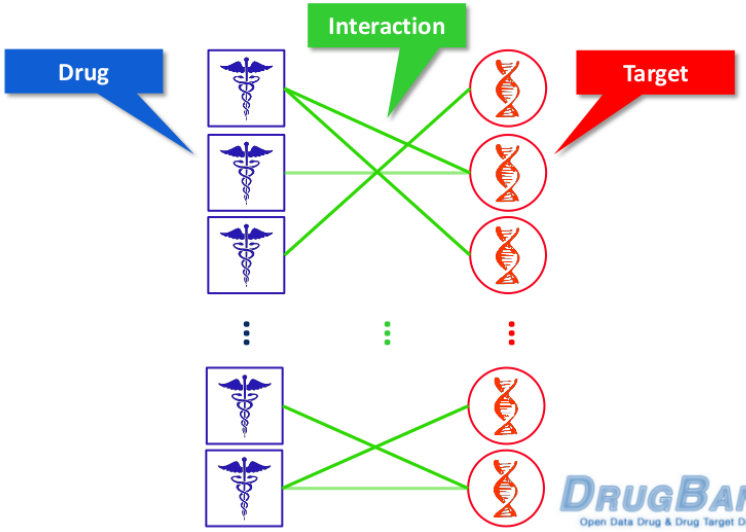


Network structure of Patent Citations
http://www.infosysblogs.com/web2/2013/07/

# Tasks to be Solved …



**Patterns** of **connections**
between people to understand functioning of society.

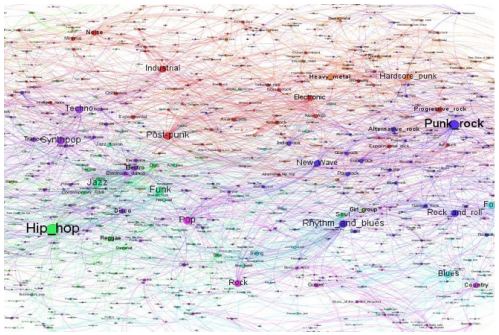



**Topological properties**
of graphs can be used to identify patterns that reveal phenomena, anomalies and potentially lead to a discovery.

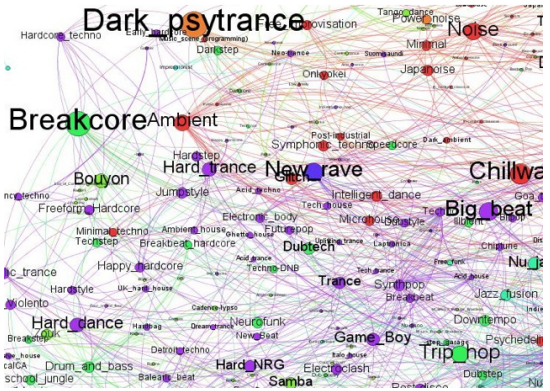
A significant increase of **graph data** in the form of social & biological information.

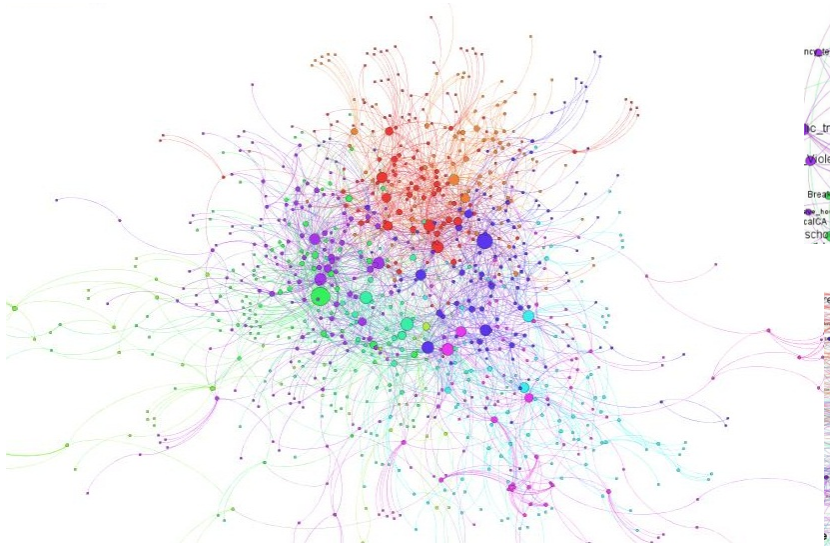
# Tasks to be Solved … (2)

## Degree

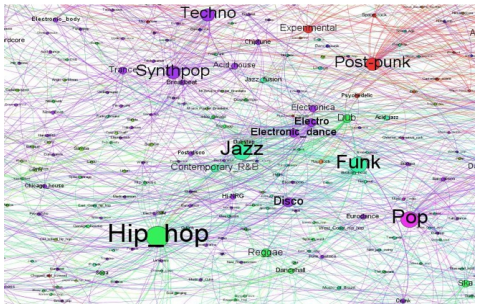



## In-degree

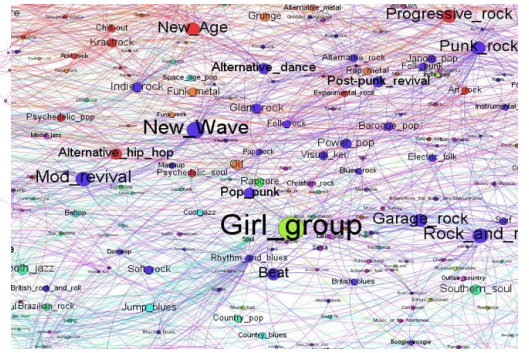


## Out-Degree




## Hubs




The **importance** of the information relies on the **relations** more or equal than on the **entities**.

http://www.infosysblogs.com/web2/2013/01/network_structure_of_music_gen.html

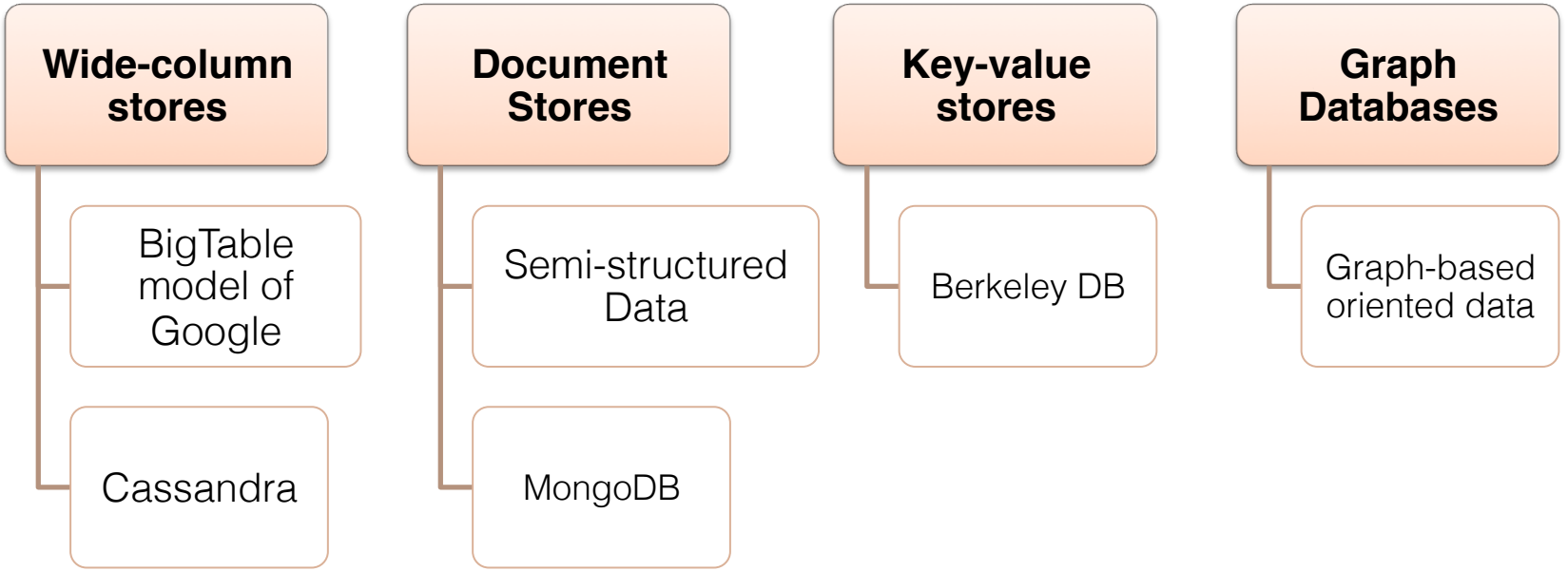
# Basic Graph Operations

**Basic Graph operations** that need to be **efficiently** performed:

- Graph traversal.
- Measuring path length.
- Finding the most interesting central node.
- Graph Invariants.

# NoSQL-based Approaches

These approaches can handle unstructured, unpredictable or messy data.



# Agenda Morning Session

1. Basic Concepts & Background (60 min)
2. The Graph Data Management Paradigm ( 35 min.)
3. Coffee Break (30 min.)
4. Existing Graph Database Engines (75 min.)
5. Questions & Discussion (15 min.)
6. Lunch Break (90 min.)

# Agenda Afternoon Session

1. RDF-Based Engines (45 min)
2. Empirical Evaluation & Hands-On Description( 45 min.)
3.  Coffee Break (30 min.)
4. Hands-On (75 min.)
5. Questions & Discussion (15 min.)

# 1 BASIC CONCEPTS & BACKGROUND

# Abstract Data Type Graph

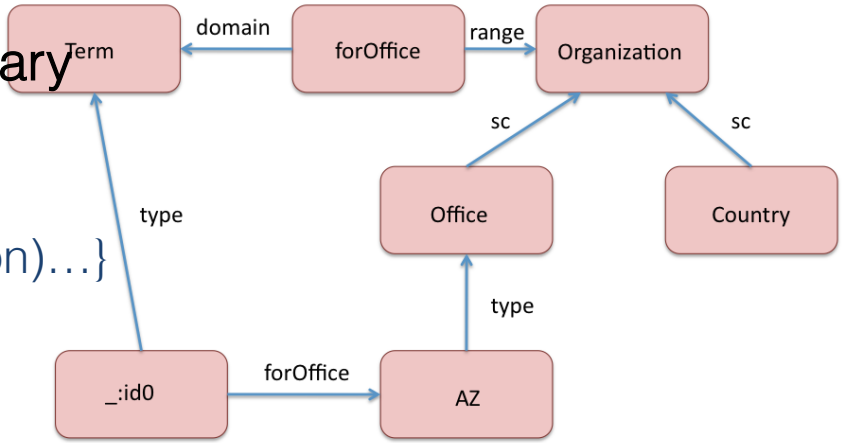
G=(V, E, Σ ,L) is a graph:

- V is a finite set of nodes or vertices,

  e.g. V={Term, forOffice, Organization,…}

- E is a set of edges representing **binary** relationship between elements in V,

  e.g. E={(forOffice,Term) (forOffice,Organization),(Office,Organization)…}




- Σ is a set of labels,

  e.g., Σ ={domain, range, sc, type, …}

- L is a function: V x V ➔ Σ ,

  e.g., L={((forOffice,Term),domain), ((forOffice,Organization),range)… }

# Abstract Data Type Multi-Graph

G=(V, E, Σ ,L) is a multi-graph:

- V is a finite set of nodes or vertices,

  e.g. V={Term, forOffice, Organization,…}

- E is a set of edges representing binary relationship between elements in V,

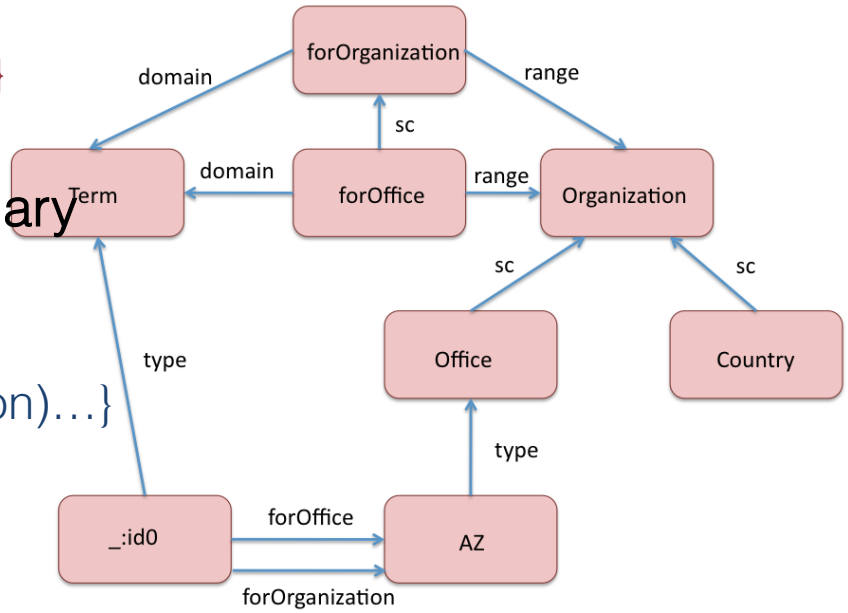
  e.g. E={(forOffice,Term) (forOffice,Organization),(Office,Organization)…}

- Σ is a set of labels,

  e.g., Σ ={domain, range, sc, type, …}

- L is a function: V x V ➔ PowerSet( Σ ),

  e.g., L={((forOffice,Term),{domain}), ((forOffice,Organization),{range}), ((_id0,AZ),{forOffice, forOrganization})… }



# Basic Operations

Given a graph G, the following are operations over G:

- AddNode(G,x): adds node x to the graph G.
- DeleteNode(G,x): deletes the node x from graph G.
- Adjacent(G,x,y): tests if there is an edge from x to y.
- Neighbors(G,x): nodes y s.t. there is a node from x to y.
- AdjacentEdges(G,x,y): set of labels of edges from x to y.
- Add(G,x,y,l): adds an edge between x and y with label l.
- Delete(G,x,y,l): deletes an edge between x and y with label l.
- Reach(G,x,y): tests if there a path from x to y.
- Path(G,x,y): a (shortest) path from x to y.
- 2-hop(G,x): set of nodes y s.t. there is a path of length 2 from x to y, or from y to x.
- n-hop(G,x): set of nodes y s.t. there is a path of length n from x to y, or from y to x.

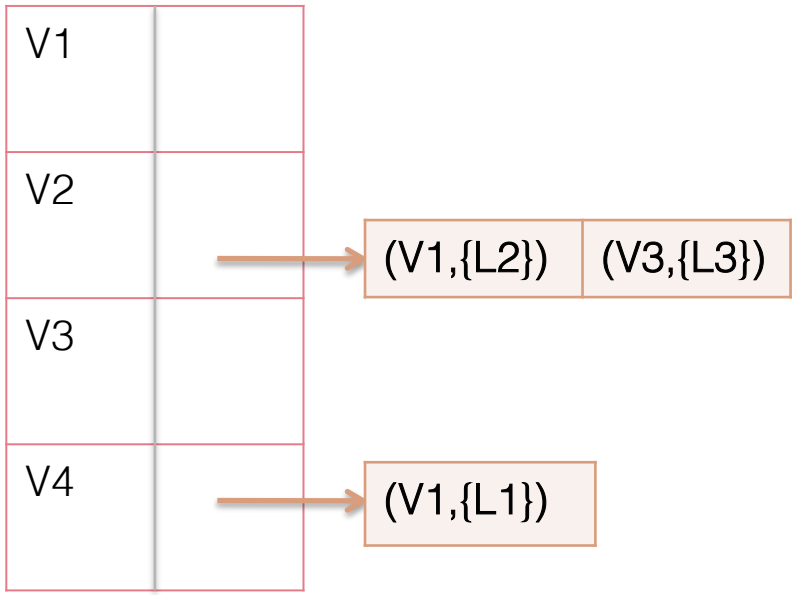
# Implementation of Graphs

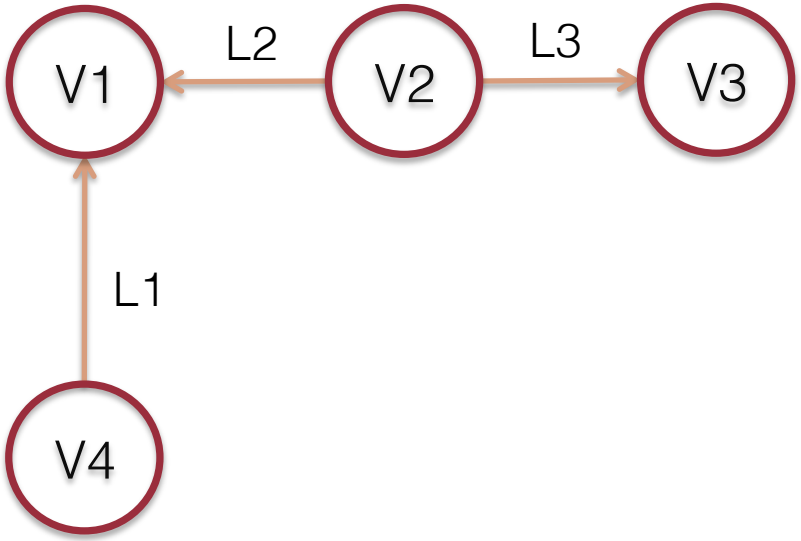
[Sakr and Pardede 2012]

**Adjacency List**

- For each node a list of neighbors.
- If the graph is directed, adjacency list of i contains only the outgoing nodes of i.
- Cheaper for obtaining the neighbors of a node.
- Not suitable for checking if there is an edge between two nodes.

# Adjacency List



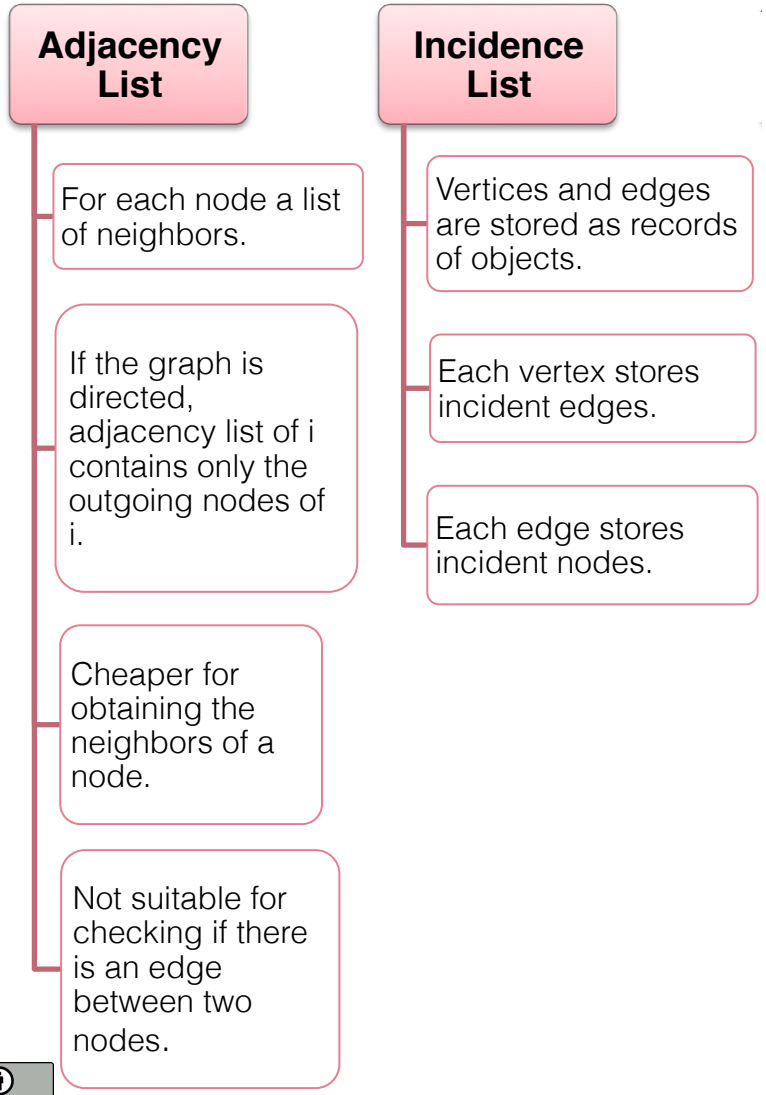




**Properties:**

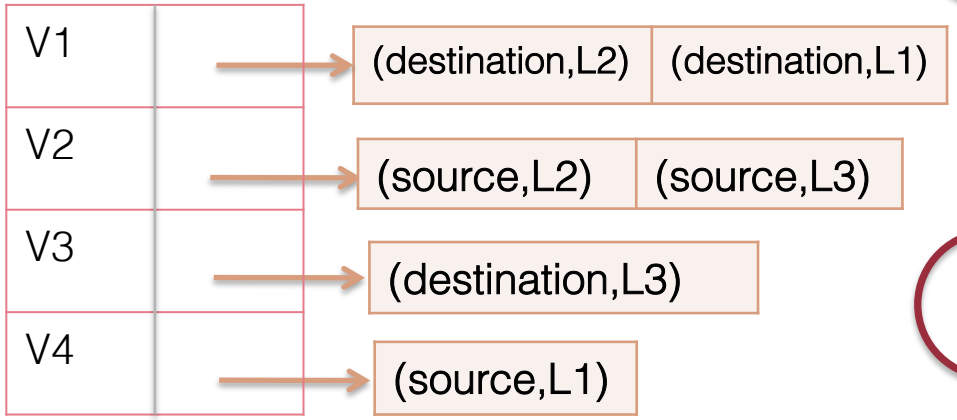
- Storage: O(|V|+|E|+|L|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|E|)
- Delete(G,x,y,l): O(|E|)

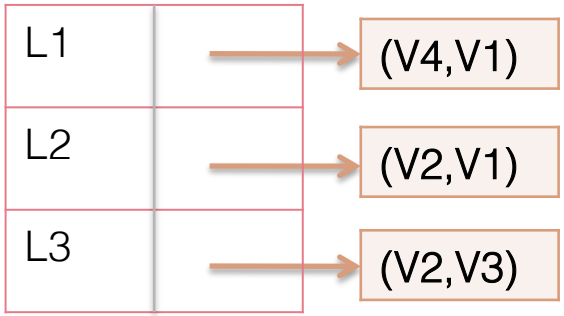
# Implementation of Graphs

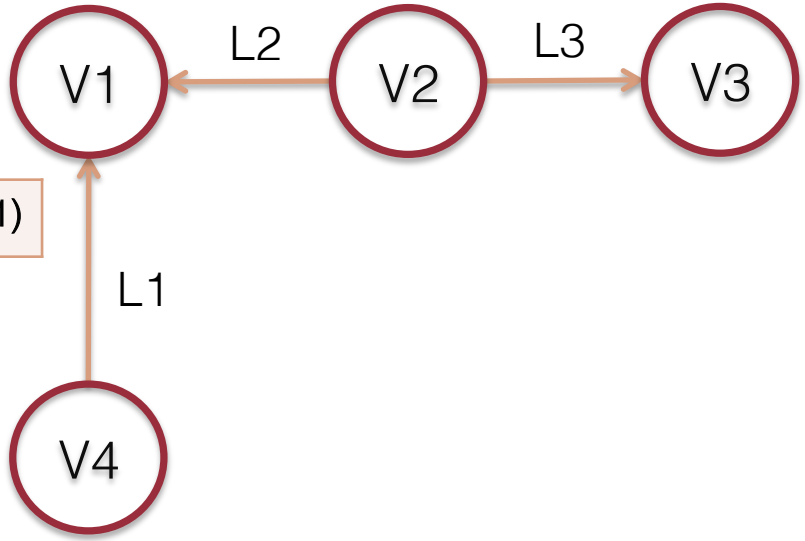
[Sakr and Pardede 2012]



# Incidence List



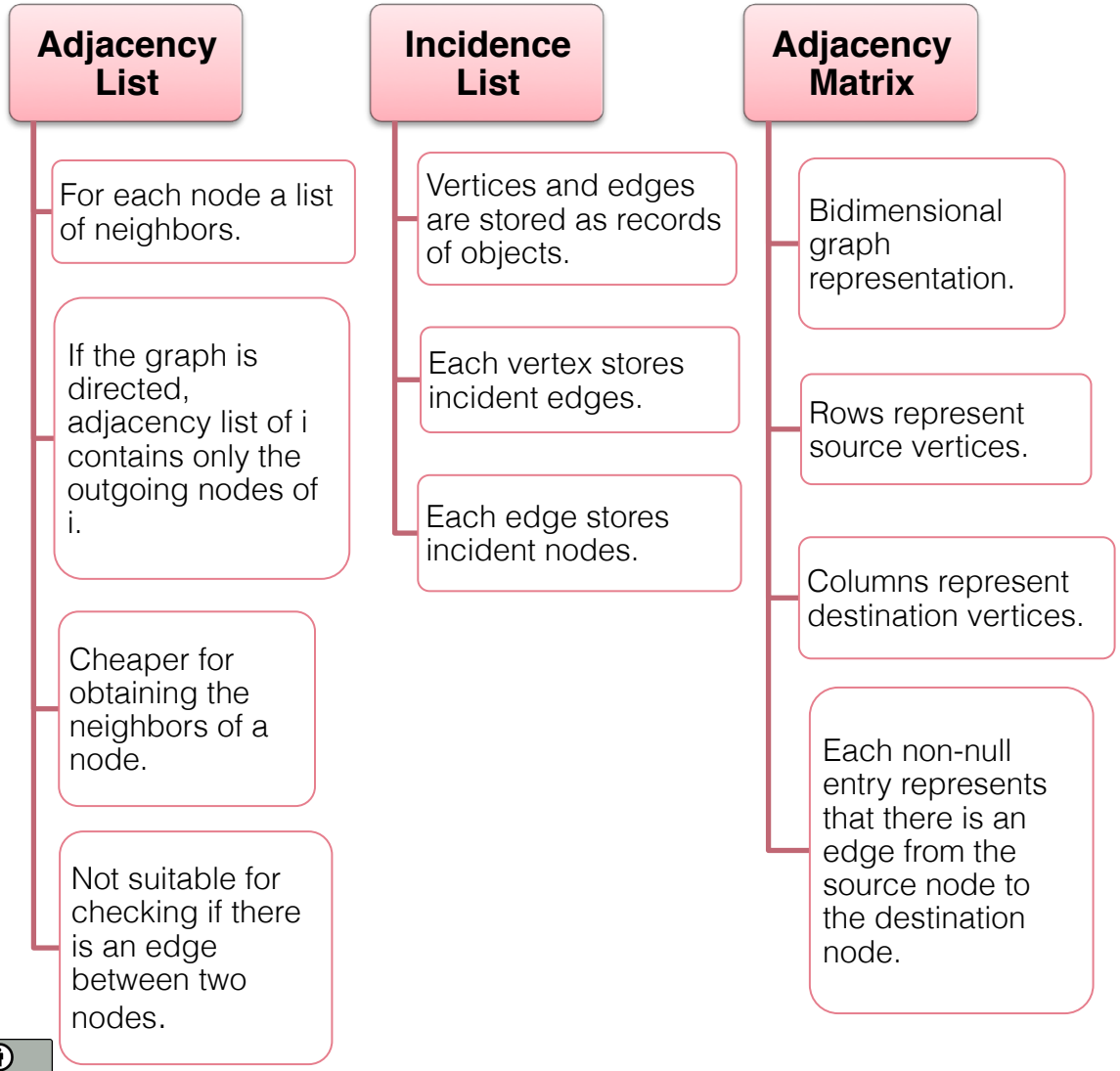




**Properties:**

- Storage: O(|V|+|E|+|L|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|E|)
- Delete(G,x,y,l): O(|E|)

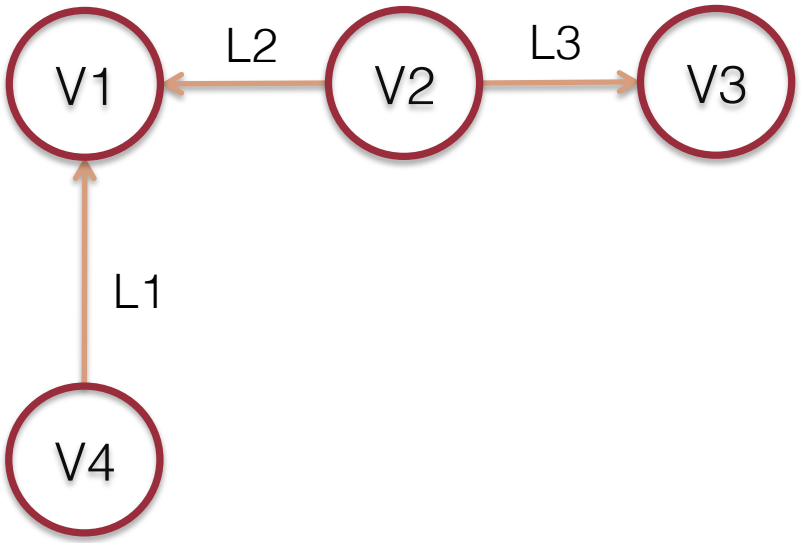
# Implementation of Graphs

[Sakr and Pardede 2012]



# Adjacency Matrix

| | V1 | V2 | V3 | V4 |
|---|---|---|---|---|
| V1 | | | | |
| V2 | {L2} | | {L3} | |
| V3 | | | | |
| V4 | {L1} | | | |

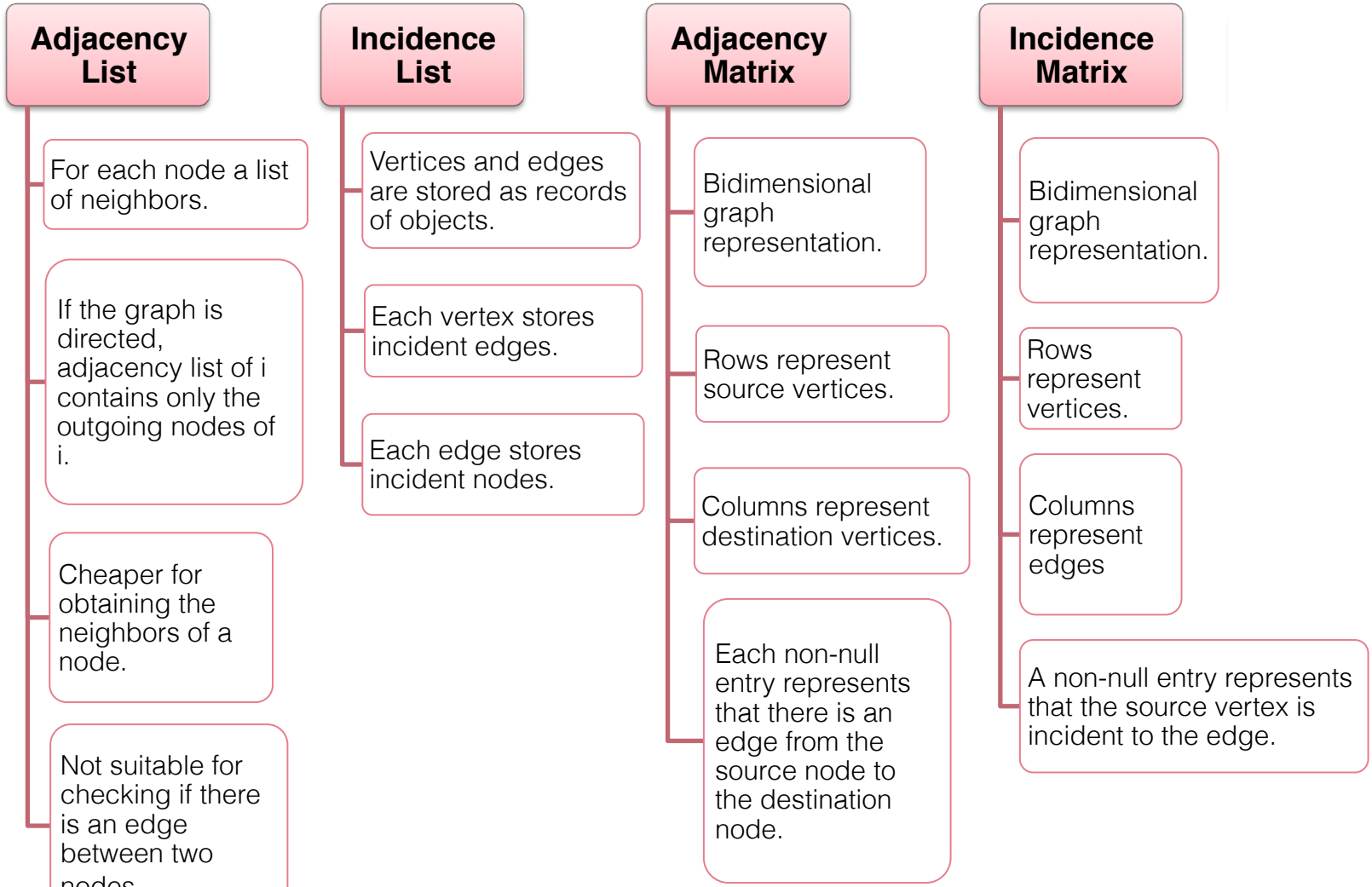



**Properties:**

- Storage: $O(|V|^2)$
- Adjacent(G,x,y): O(1)
- Neighbors(G,x): O(|V|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|E|)
- Delete(G,x,y,l): O(|E|)

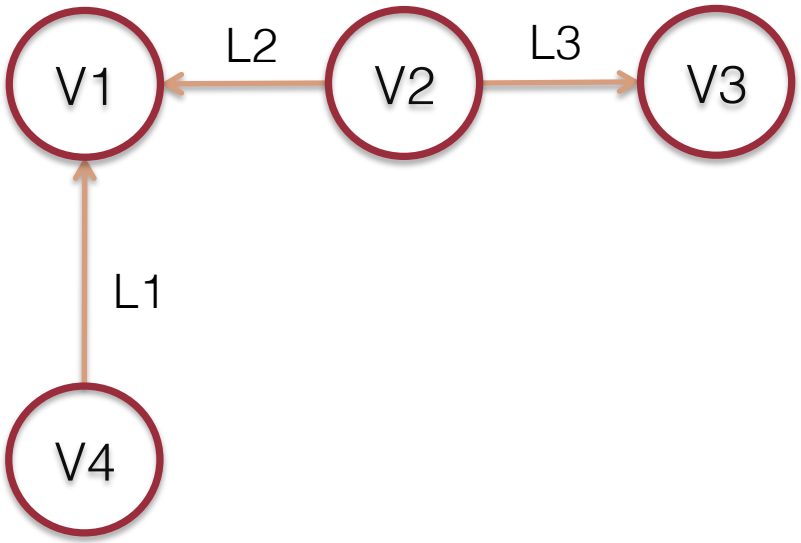
# Implementation of Graphs

[Sakr and Pardede 2012]



# Incidence Matrix

| | L1 | L2 | L3 |
|---|---|---|---|
| V1 | destination | destination | |
| V2 | | source | source |
| V3 | | | destination |
| V4 | source | | |

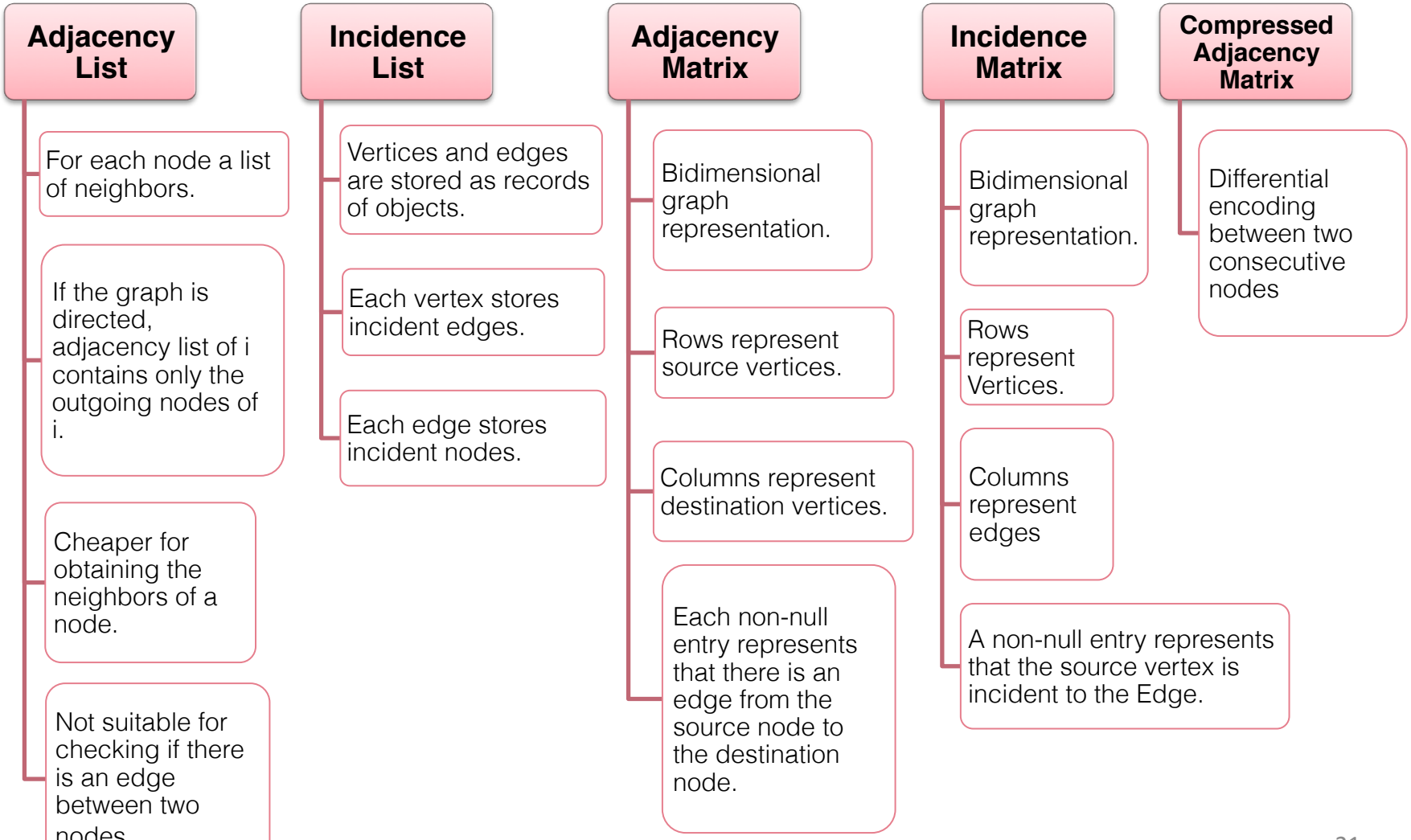



**Properties:**

- Storage: O(|V|x|E|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|V|x|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|V|)
- Delete(G,x,y,l): O(|V|)

# Implementation of Graphs

[Sakr and Pardede 2012]



# Compressed Adjacency List
# Gap Encoding

**Original Adjacency List**

| Node | Neighbors |
|---|---|
| 1 | 20,23,24,25,27 |
| 2 | 30,32,33,34 |
| 3 | 40,42,45,46,47,48 |
| ….. | |

**Compressed Adjacency List**

| Node | Successors |
|---|---|
| 1 | 38,2,0,0,1 |
| 2 | 56,1,0,0 |
| 3 | 74,1,2,0,0,0 |
| ….. | |

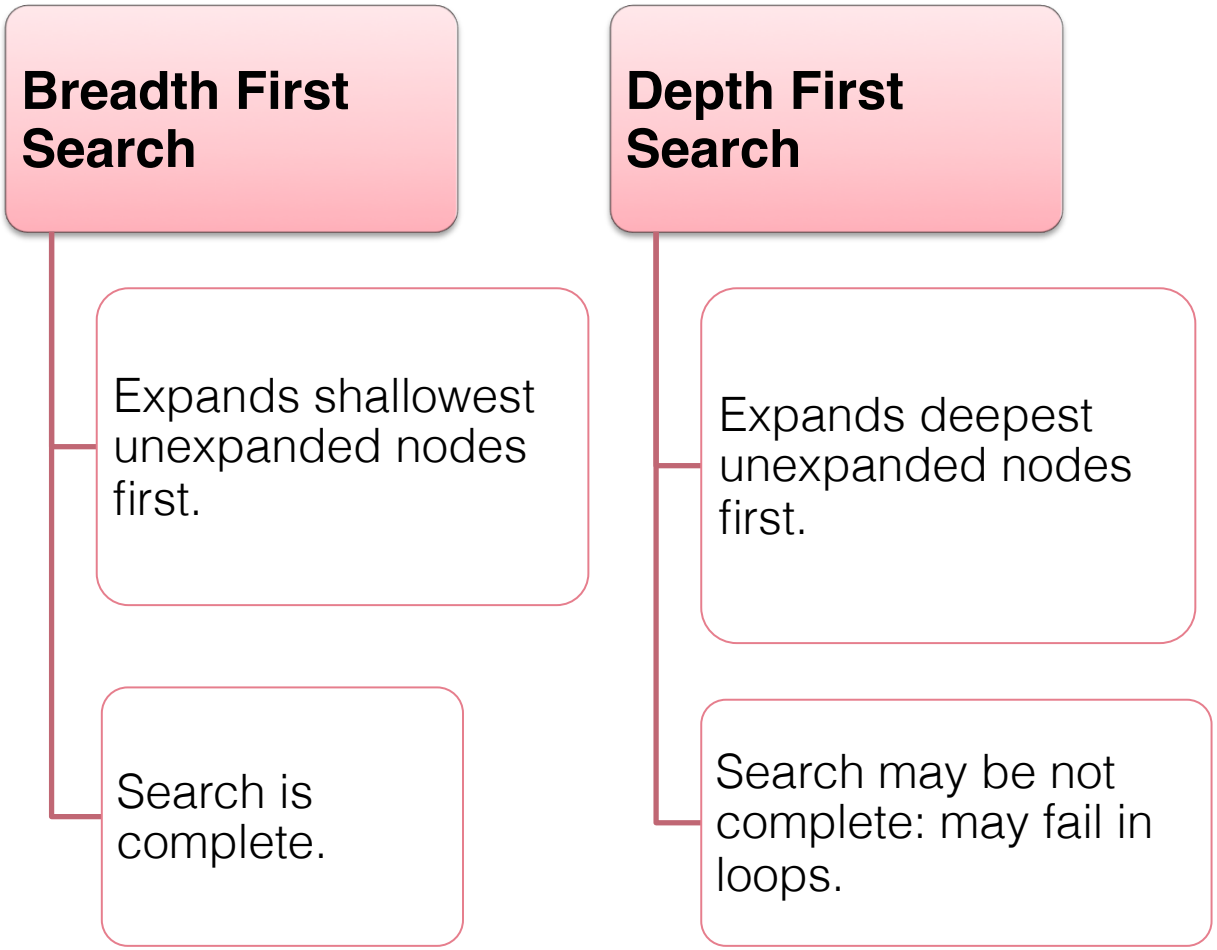
The **ordered** adjacent list:
$A(x)=(a_1,a_2,a_3,\ldots,a_{n)}$
is encoded as
$(v(a_1\text{-}x),a_2\text{-}a_1\text{-}1,a_3\text{-}a_2\text{-}1,\ldots,\ a_n\text{-}a_{n\text{-}1}\ \text{-}1)$
Where:

$$v(x)=\begin{cases}2x & \text{if x} \geq 0\\ 2\ |\text{x}|\text{-}1 & \text{if x} < 0\end{cases}$$

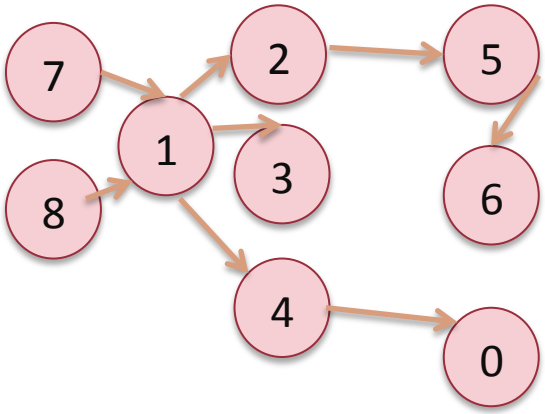
**Properties:**

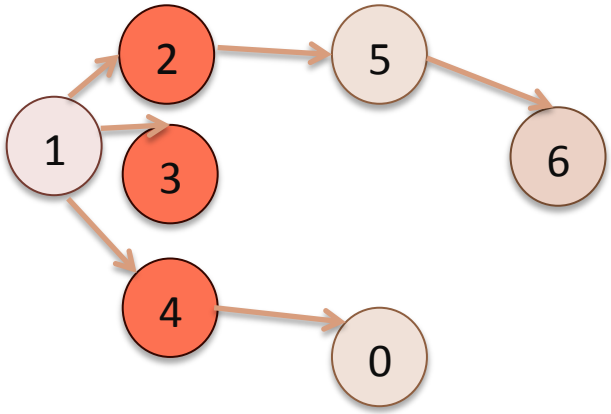
- Storage: O(|V|+|E|+|L|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|E|)
- Delete(G,x,y,l): O(|E|)

# Traversal Search



# Breadth First Search




**Notation:**

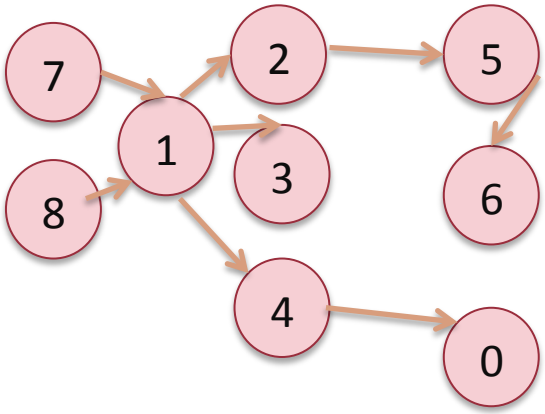
1 Starting Node

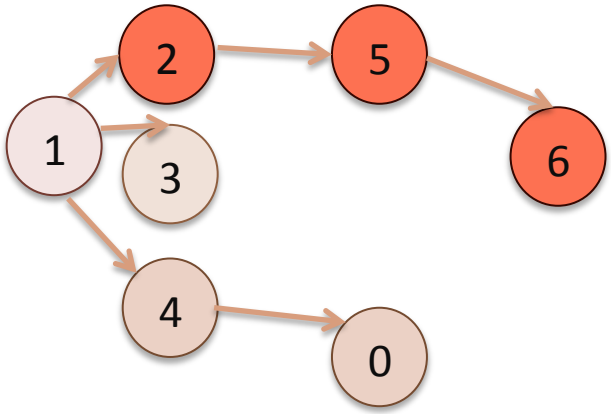
First Level Visited Nodes

Second Level Visited Nodes

Third Level Visited Nodes

# Depth First Search




**Notation:**

1 Starting Node

First Level Visited Nodes

Second Level Visited Nodes

Third Level Visited Nodes

# 2 THE GRAPH DATA MANAGEMENT PARADIGM

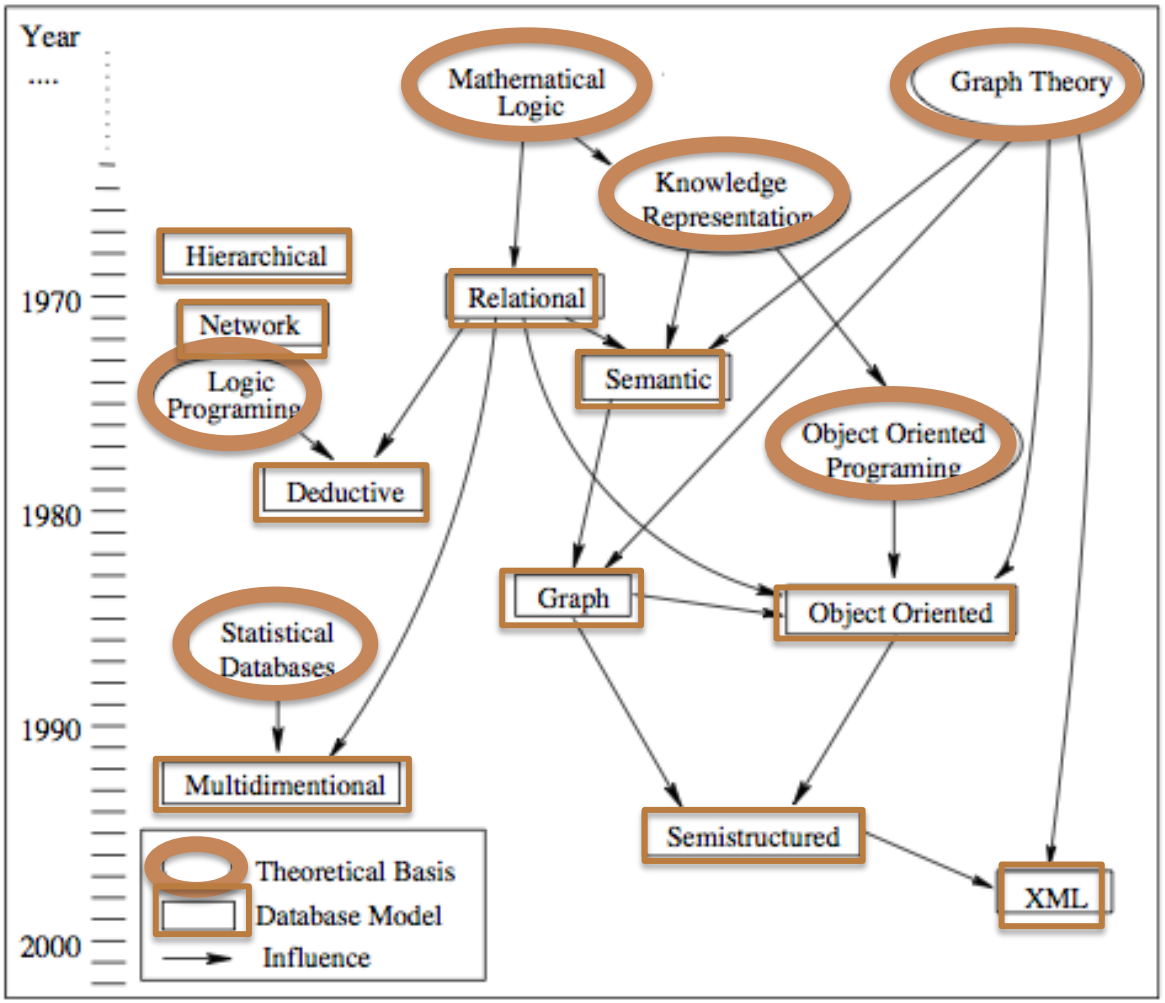
# The Graph Data Management Paradigm

Graph database models:

- **Data models** for schema and instances are graph models or generalizations of them.
- **Data manipulation** is expressed as graph-based operations.

# Survey of Graph Database Models

[Renzo and Gutiérrez 2008]




A **graph data model** represents data and schema:

- **Graphs** or
- More general, the **notion of graphs**: Hypergraphs, Digraphs.

# Survey of Graph Database Models

[Renzo and Gutiérrez 2008]

Types of relationship supported by graph data models:

**Attributes**

- Properties,
- Mono- or multi-valued.

**Entities**

- Groups of real-world objects.

**Neighborhood relations**

- Structures to represent neighborhoods of an entity.

**Standard abstractions**

- Part-of, composed-by, n-ary associations.

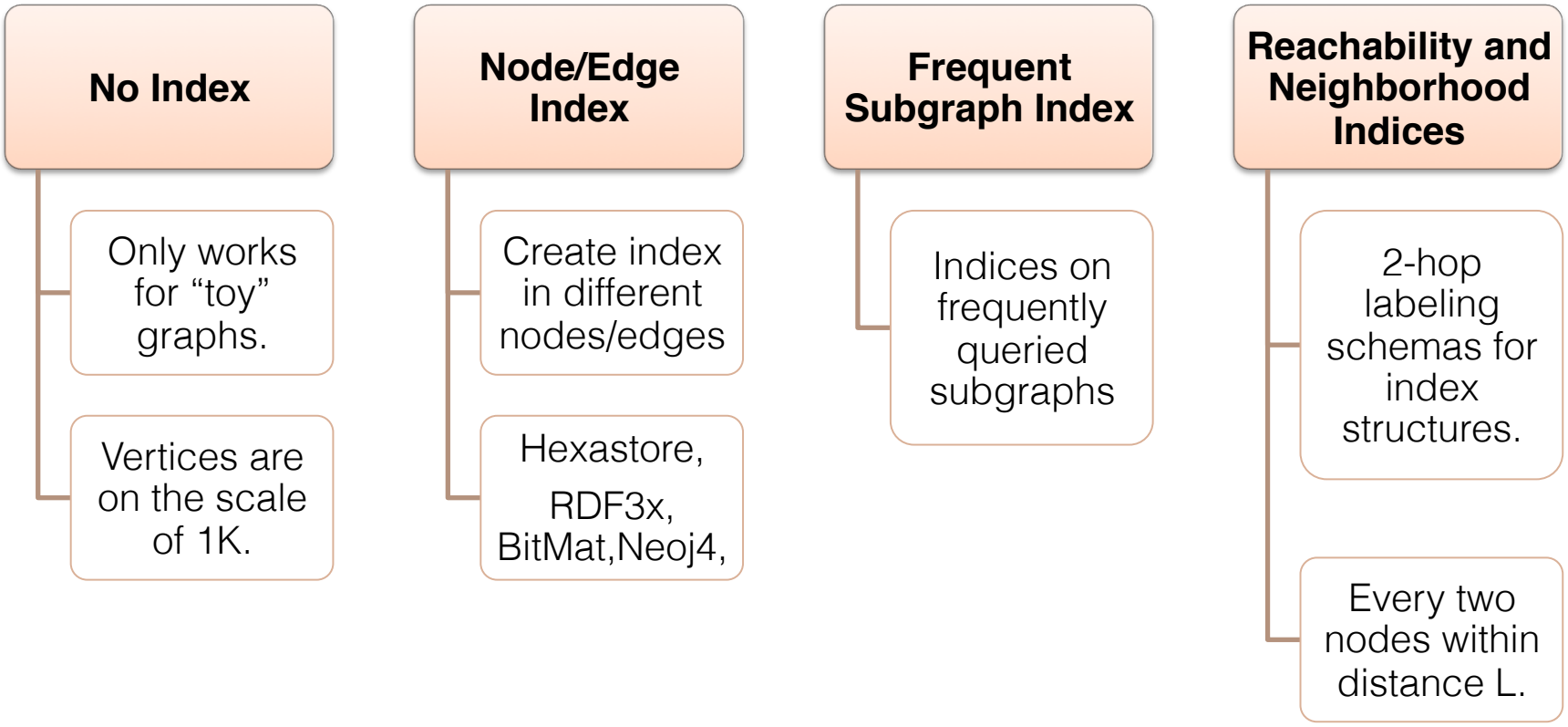
**Derivation and inheritance**

- Subclasses and superclasses,
- Relations of instantiations.

**Nested relations**

- Recursively specified relations.

# Representative Approaches



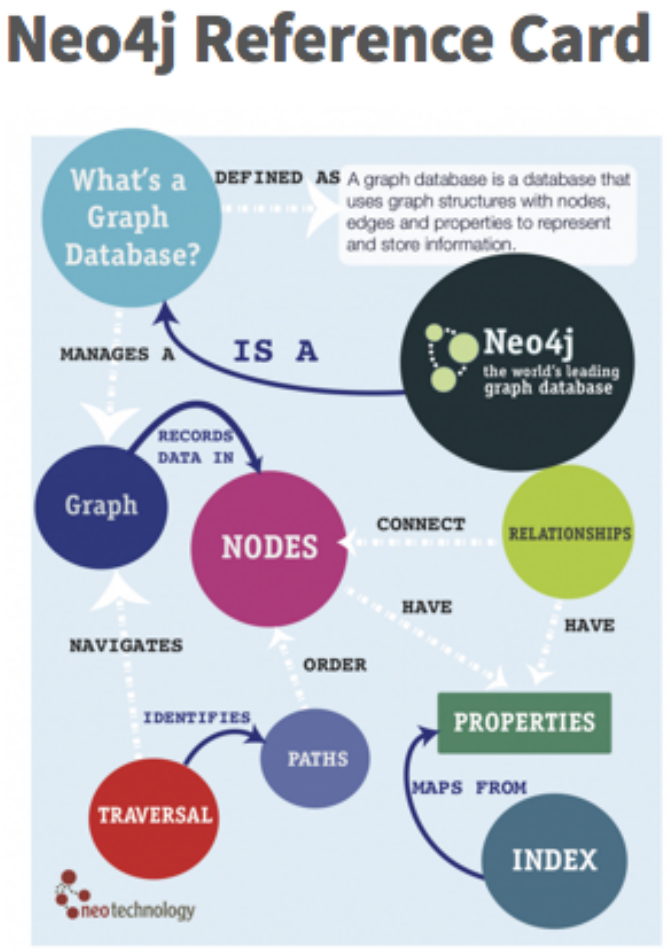
# COFFEE BREAK

# 3 GRAPH DATABASES ENGINES

# Graph Databases Advantages

- ✓ Natural modeling of highly **connected data**.
- ✓ Special graph **storage** structure
- ✓ Efficient **schemaless** graph algorithms.
- ✓ Support for **query languages**.
- ✓ **Operators** to query the graph structure.

# Graph Databases

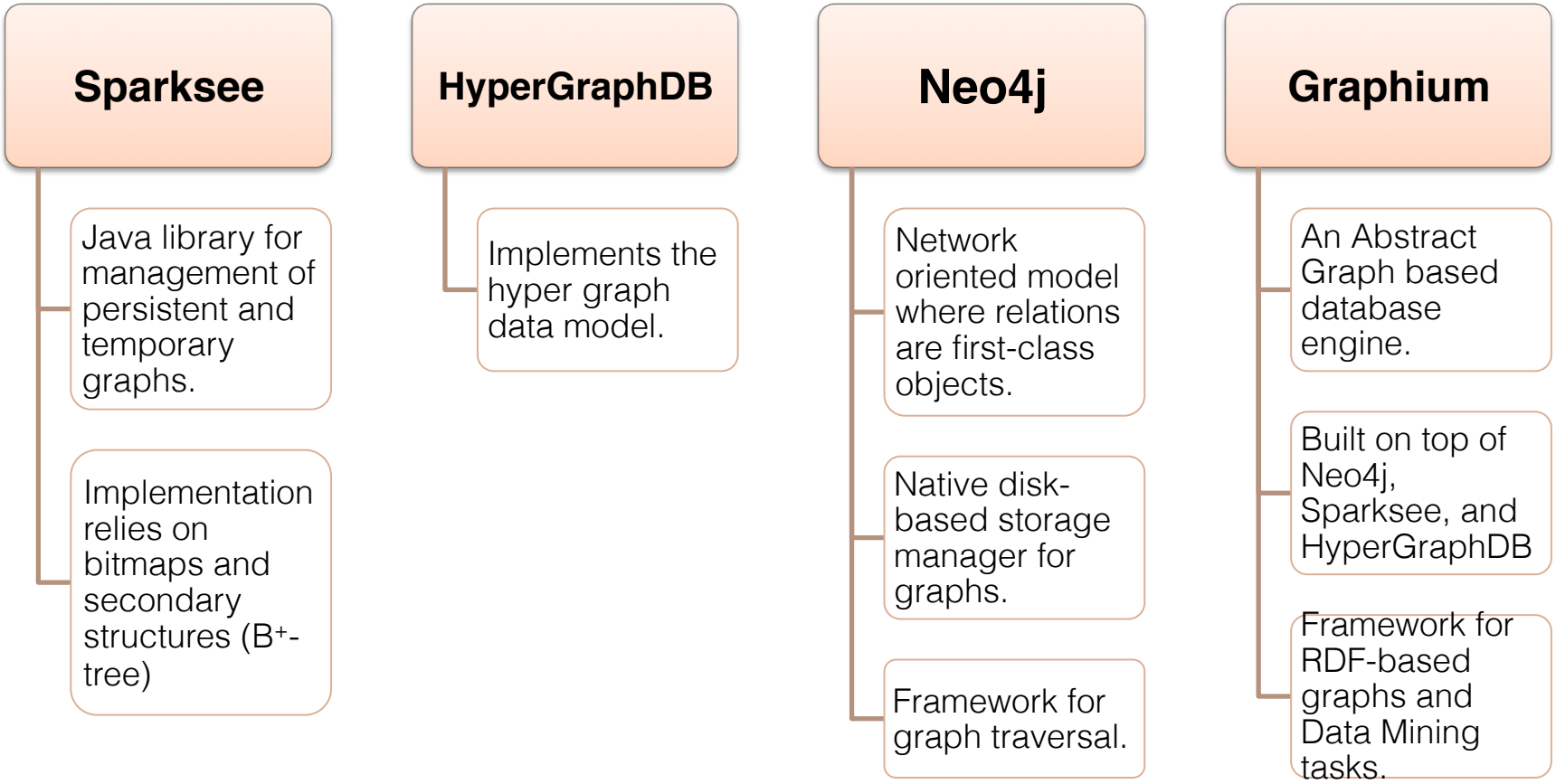



http://www.neo4j.org


http://www.hypergraphdb.org


http://www.sparsity-technologies.com/

# Existing General Graph Database Engines

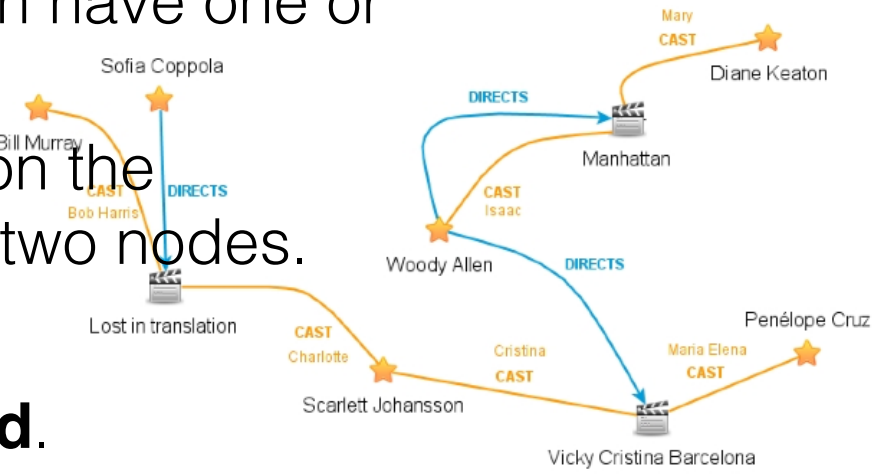



Most graph databases implement an API instead of a query language.

# Sparksee(1)

[Martínez-Basan et al. 2007]

- Labeled attribute **multi-graph**.
- All node and edge objects belong to a **type**.
- Edges have a **direction**:
  - Tail: corresponds to the source of the edge.
  - Head: corresponds to the destination of the edge.
- Node and edge objects can have one or more **attributes**.
- There are **no restrictions** on the **number** of edges between two nodes.
- **Loops** are allowed.
- Attributes are **single valued**.

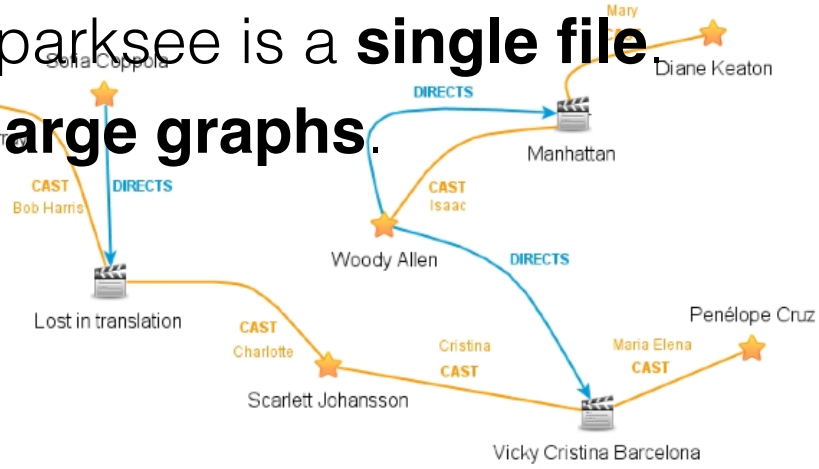



http://www.sparsity-technologies.com/dex_tutorials

# Sparksee (2)

[Martínez-Basan et al. 2007]

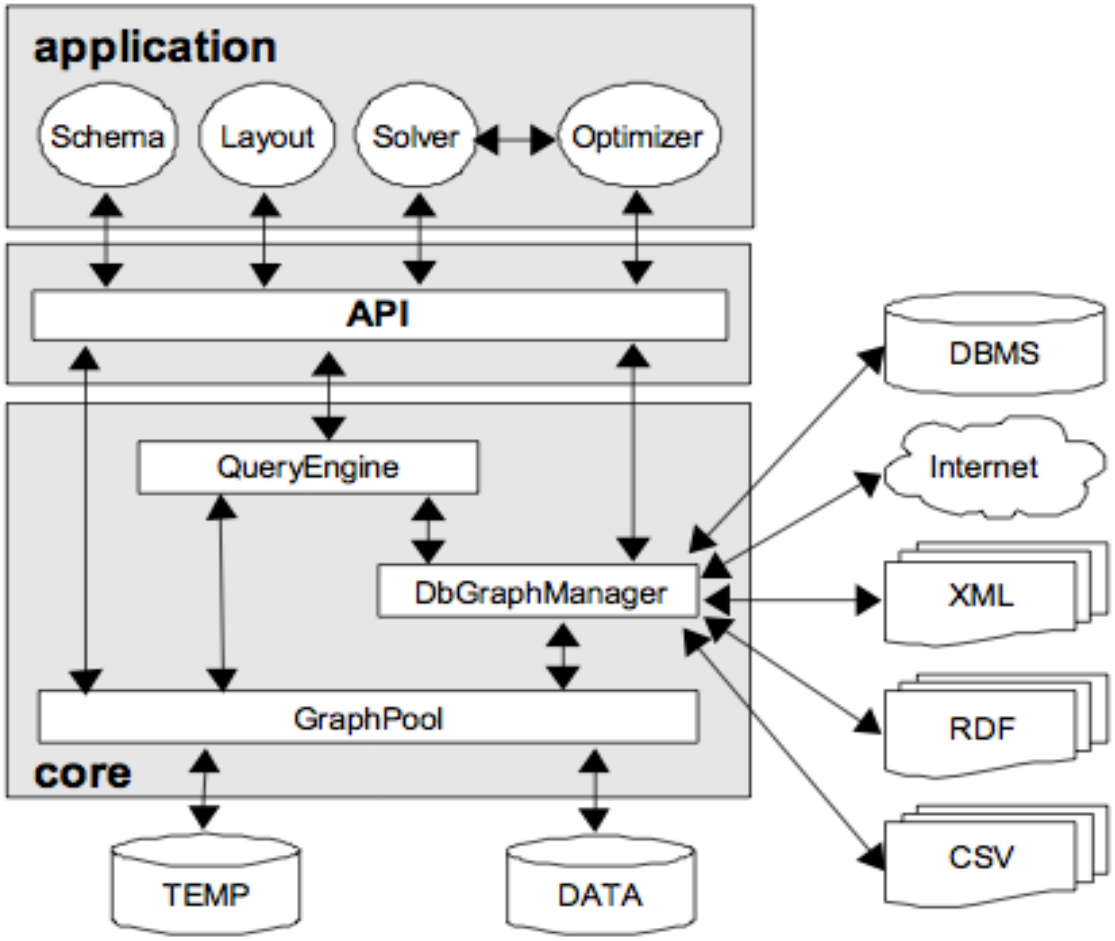
- Attributes can have different **indexes**:
  - Basic attributes.
  - Indexed attributes.
  - Unique attributes.
  - Edges  can index neighborhoods.
  - Neighborhood index.
- The persistent database of Sparksee is a **single file**.
- Sparksee can manage **very large graphs**.




http://www.sparsity-technologies.com/dex_tutorials

# Sparksee Architecture

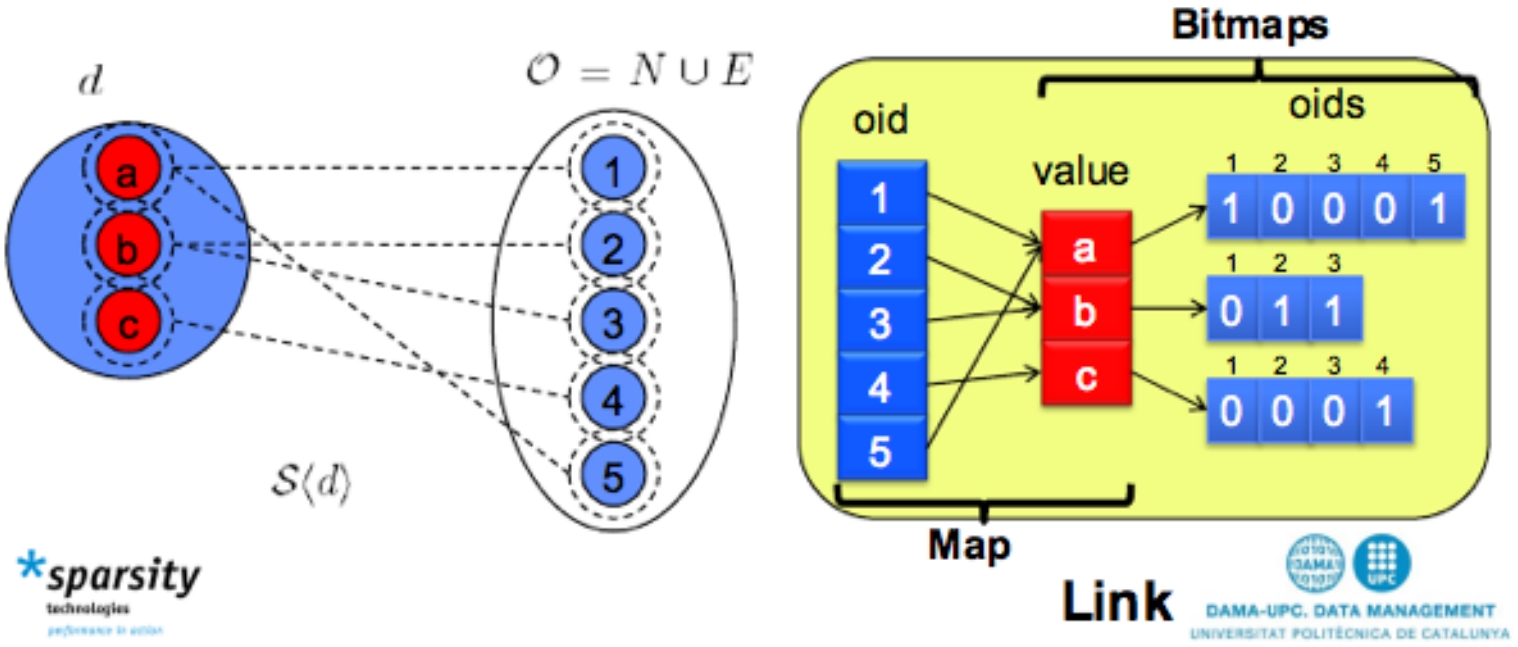
[Martínez-Basan et al. 2007]




http://www.sparsity-technologies.com/dex_tutorials

# Sparksee Internal Representation (1)

[Martínez-Basan et al. 2007]




**Sparksee Approach:** Map + Bitmaps ➔ Links

Link: bidirectional association between values and OIDs

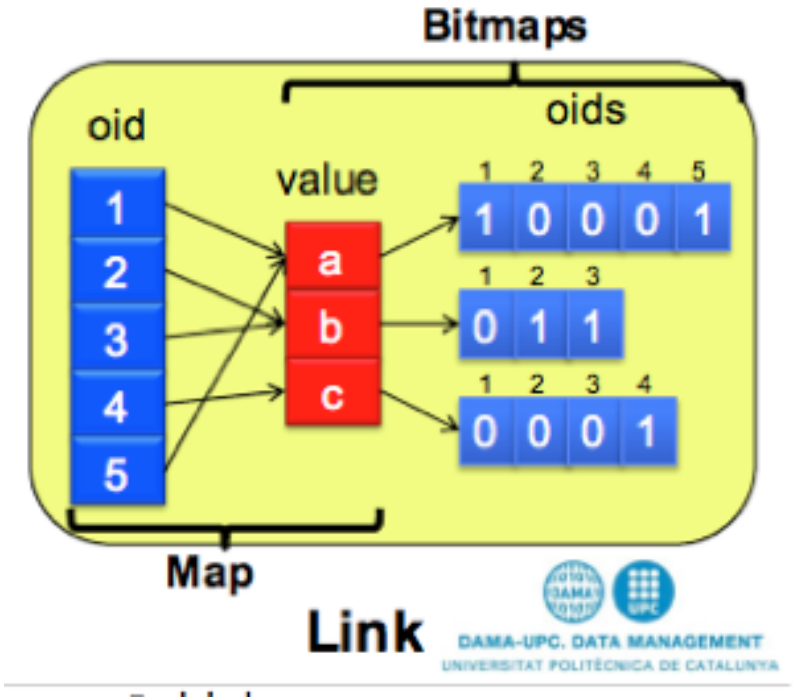
- Given a value ➔ a set of OIDs
- Given an OID ➔ the value

http://www.sparsity-technologies.com/

# Sparksee Internal Representation (2)

[Martínez-Basan et al. 2007]

- A Sparksee Graph is a combination of **Bitmaps**:
  - Bitmap for each node or edge type.
  - One link for each attribute.
  - Two links for each type: Out-going and in-going edges.
- Maps are B+trees
  - A compressed UTF-8 storage for UNICODE string.




http://www.sparsity-technologies.com/

# Sparksee API

Operations of the Abstract Data Type Graph:

- Graph construction.
- Definition of node and edge types.
- Creation of node and edge objects.
- Definition and use of attributes.
- Query nodes and edges.
- Management of edges and nodes.

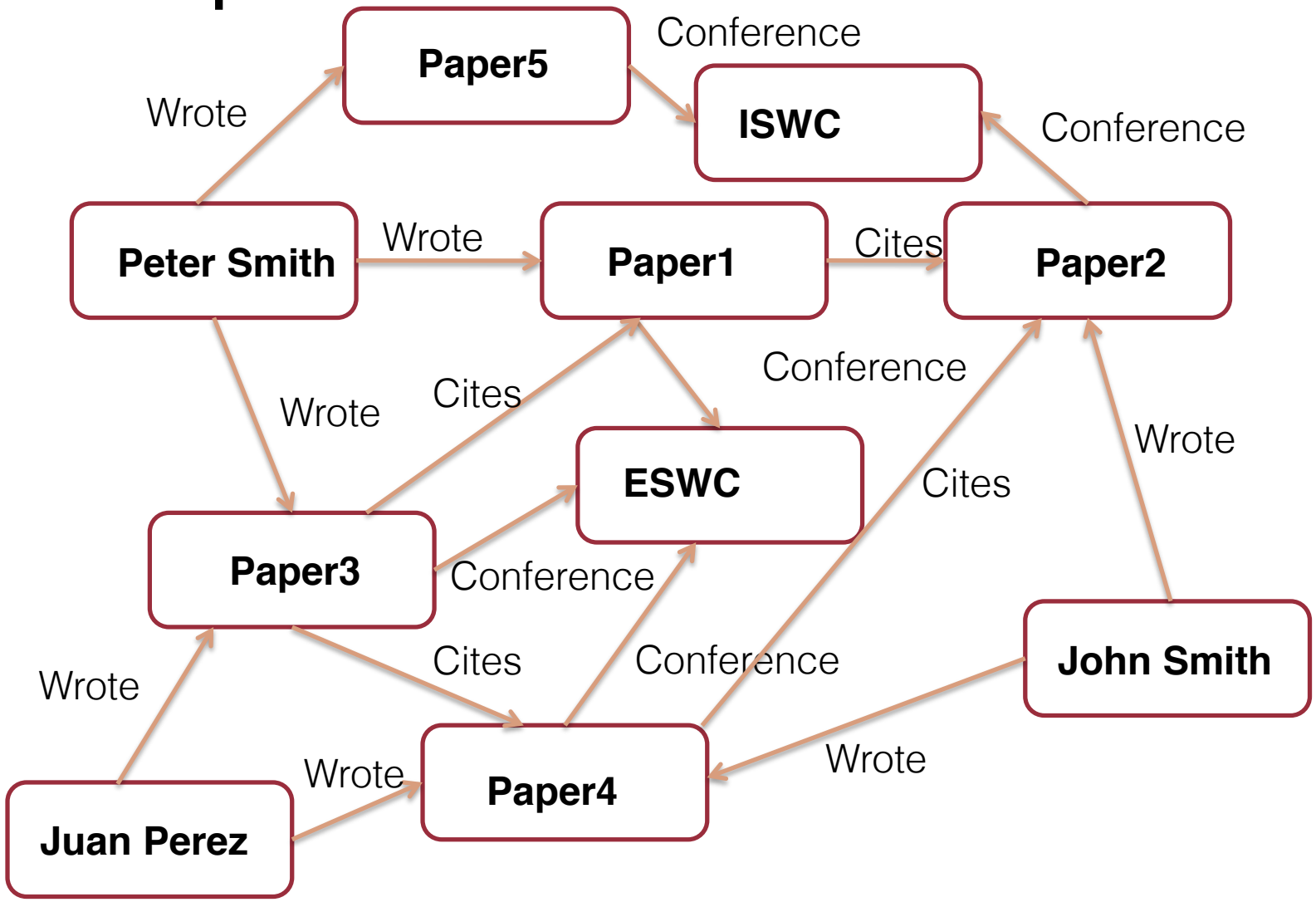
# Sparksee API (2)

## Graph construction

```
SparkseeConfig cfg = new SparkseeConfig();
cfg.setCacheMaxSize(2048); // 2 GB
Sparksee sparksee= new Sparksee(cfg);
Database db = sparksee.create(”Publications.dex",
  ”PUBLICATIONSDex");
Session sess = db.newSession();
Graph graph = sess.getGraph();
 ……
sess.close();
db.close();
sparksee.close();
```

Operations on the graph database will be performed on the object graph

# Example


Paper5
Conference
ISWC
Conference
Wrote
Peter Smith
Wrote
Paper1
Cites
Paper2
Wrote
Cites
Conference
ESWC
Wrote
Cites
Paper3
Conference
Cites
Conference
John Smith
Wrote
Wrote
Paper4
Wrote
Juan Perez

# Sparksee API (3)

## Adding nodes to the graph

```
int authorTypeId = graph.newNodeType(”AUTHOR");
int paperTypeId = graph.newNodeType(”PAPER");
int conferenceTypeId = graph.newNodeType(”CONFERENCE");

long author1 = graph.newNode(authorTypeId);
long author2 = graph.newNode(authorTypeId);
long author3 = graph.newNode(authorTypeId);

long paper1 = graph.newNode(paperTypeId);
long paper2 = graph.newNode(paperTypeId);
long paper3 = graph.newNode(paperTypeId);
long paper4 = graph.newNode(paperTypeId);
long paper5 = graph.newNode(paperTypeId);

long conference1 = graph.newNode(conferenceTypeId);
long conference2 = graph.newNode(conferenceTypeId);
```

# Sparksee API (4)

## Adding edges to the graph

```
int wroteTypeId = graph.newEdgeType(”WROTE", true,true);
int isWrittenTypeId = graph.newEdgeType(”Is-Written", true,true);
int citeTypeId = graph.newEdgeType(”CITE", true,true);
int isCitedByTypeId = graph.newEdgeType(”IsCitedBy", true,true);
int conferenceTypeId = graph.newEdgeType(”IsConference", true,true);
```

```
long wrote1 = graph.newEdge(wroteTypeId, author1, paper1);
long wrote2 = graph.newEdge(wroteTypeId, author1, paper3);
long wrote3 = graph.newEdge(wroteTypeId, author1, paper5);
long wrote4 = graph.newEdge(wroteTypeId, author2, paper3);
long wrote5 = graph.newEdge(wroteTypeId, author2, paper4);
long wrote4 = graph.newEdge(wroteTypeId, author3, paper2);
long wrote5 = graph.newEdge(wroteTypeId, author3, paper4);
```

# Sparksee API (5)

## Indexing

- Basic attributes: There is no index associated with the attribute.
- Indexed attributes: There is an index automatically maintained by the system associated with the attribute.
- Unique attributes: The same as for indexed attributes but with an added integrity restriction: two different objects cannot have the same value, with the exception of the null value.
- Sparksee operations: Accessing the graph through an attribute will automatically use the defined index, significantly improving the performance of the operation.
- A specific index can also be defined to improve certain navigational operations, for example, Neighborhood.
- There is the AttributeKind enum class which includes basic, indexed and unique attributes.

# Sparksee API (6)

Adding attributes to nodes of the graph

```
nameAttrId = graph.newAttribute(authorTypeId, "Name",
DataType.String, AttributeKind.Indexed);

conferenceNameAttrId = graph.newAttribute(conferenceTypeId,
”ConferenceName", DataType.String, AttributeKind.Indexed);

Value v = new Value();
graph.setAttribute(author1, nameAttrId, v.setString(Peter Smith"));
graph.setAttribute(author2, nameAttrId, v.setString(”Juan Perez"));
graph.setAttribute(author2, nameAttrId, v.setString(”John Smith"));
graph.setAttribute(conference1, conferenceNameAttrId, v.setString
(“ESWC”)));
graph.setAttribute(conference2, conferenceNameAttrId, v.setString
(“ISWC"));
```

# Sparksee API (7)

## Searching in the graph

```
Value v = new Value();
// retrieve all 'AUTHOR' node objects
Objects authorObjs1 = graph.select(authorTypeId);
...// retrieve Peter Smith from the graph, which is a "AUTHOR"
Objects authorObjs2 = graph.select(nameAttrId, Condition.Equal,
v.setString("Peter Smith"));
...// retrieve all 'author' node objects having "Smith" in the name.
It would retrieve
// Peter Smith and John Smith
Objects authorObjs3 = graph.select(nameAttrId, Condition.Like,
v.setString("Smith"));
...
authorObjs1.close();
authorObjs2.close();
authorObjs3.close();
```

# Sparksee API (8)

## Searching in the graph

```
Graph graph = sess.getGraph();
Value v = new Value();
…
// retrieve all 'author' node objects having a value for the 'name' attribute
// satisfying the '^J[^]*n$'
Objects authorObjs = graph.select(nameAttrId,
Condition.RegExp, v.setString("^J[^]*n$"));
...
authorObjs.close();
```

# Sparksee API (9)

## Searching in the graph

```
Graph graph = sess.getGraph();
sess.begin();
Value v = new Value();

int authorTypeId = graph.findType(”AUTHOR");
int nameAttrId = graph.findAttribute(authorTypeId, "NAME");
v.setString(”Peter Smith");
Long peterSmith = graph.findObject(nameAttrId, v);
sess.commit();
```

# Sparksee API (10)

Traversing the graph

- **Navigational direction** can be restricted through the edges
- The **EdgesDirection** enum class:
  - Outgoing: Only out-going edges, starting from the source node, will be valid for the navigation.
  - Ingoing: Only *in-coming* edges will be valid for the navigation.
  - Any: Both out-going and in-going edges will be valid for the navigation.

# Sparksee API (11)

**Explode-based** methods visit the edges of a given node
**Neighbor-based** methods visit the neighbor nodes of a given node identifier

## Traversing the graph

```
Graph graph = sess.getGraph();
...
Objects authorObjs1 = graph.select(authorTypeId);
...// retrieve Peter Smith from the graph, which is a ”AUTHOR"
Objects node1 = graph.select(nameAttrId, Condition.Equal,
v.setString(”Peter Smith"));
Objects node2 = graph.select(nameAttrId, Condition.Like, v.setString
(”Paper"));
Int wroteTypeId = graph.findType(”WROTE");
Int citeTypeId = graph.findType(”CITE");
...// retrieve all in-comings WROTE
Objects edges = graph.explode(node1, wroteTypeId,
EdgesDirection.Ingoing);
...// retrieve all nodes through CITE
Objects cites = graph.neighbors(node2, citeTypeId,
EdgesDirection.Any);
...
edges.close();
cites.close();
```

# Sparksee API (12)

## Traversing the graph

```
Graph graph = sess.getGraph();
...
Objects node2 = graph.select(nameAttrId, Condition.Like,
v.setString(”Paper"));
int citeTypeId = graph.findType(”CITE");
// 1-hop cites
Objects cites1 = graph.neighbors(node2, citeTypeId,
EdgesDirection.Any);
// cites of cites (2-hop)
Objects cites2 = graph.neighbors(cites1, citeTypeId,
EdgesDirection.Any);
.....
edges.close();
cites.close();
```

# Sparksee API Summary

| Operation | Description |
|---|---|
| CreateGraph | Creates a empty graph |
| DropGraph | Removes a graph |
| RegisterType | Registers a new node or edge type |
| GetTypes | Returns the list of all node or edge types |
| FindType | Returns the id of a node or edge type |
| NewNode | Creates a new empty node of a given node type |
| AddNode | Copies a node |
| AddEdge | Adds a new edge of some edge type |
| DropObject | Removes a node or an edge |
| Scan | Returns all nodes or edges of a given type |
| Select | Returns the nodes or edges of a given type which have an attribute that evaluates true to a comparison operator over a value |
| Related | Returns all neighbors of a node following edges members of an edge type |
| Neighbors | Returns all neighbors |

# Neo4J

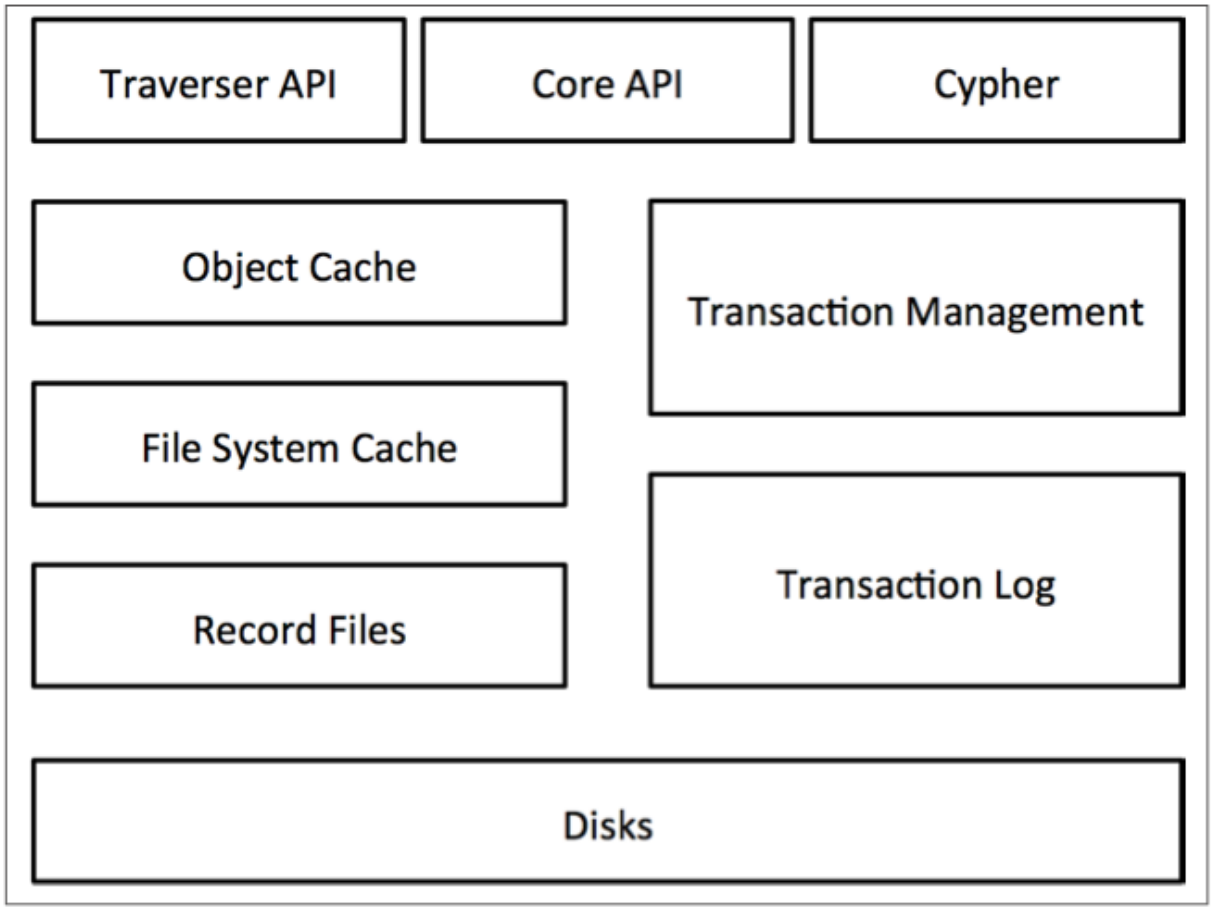
[Robinson et al. 2013]

- Labeled attribute **multigraph**.
- Nodes and edges can have **properties**.
- There are **no restrictions** on the number of edges between two nodes.
- **Loops** are allowed.
- Attributes are **single valued**.
- Different types of **indexes**: Nodes & relationships.
- Different types of **traversal strategies**.
- API for Java, Python.

http://neo4j.org/

# Neo4J Architecture

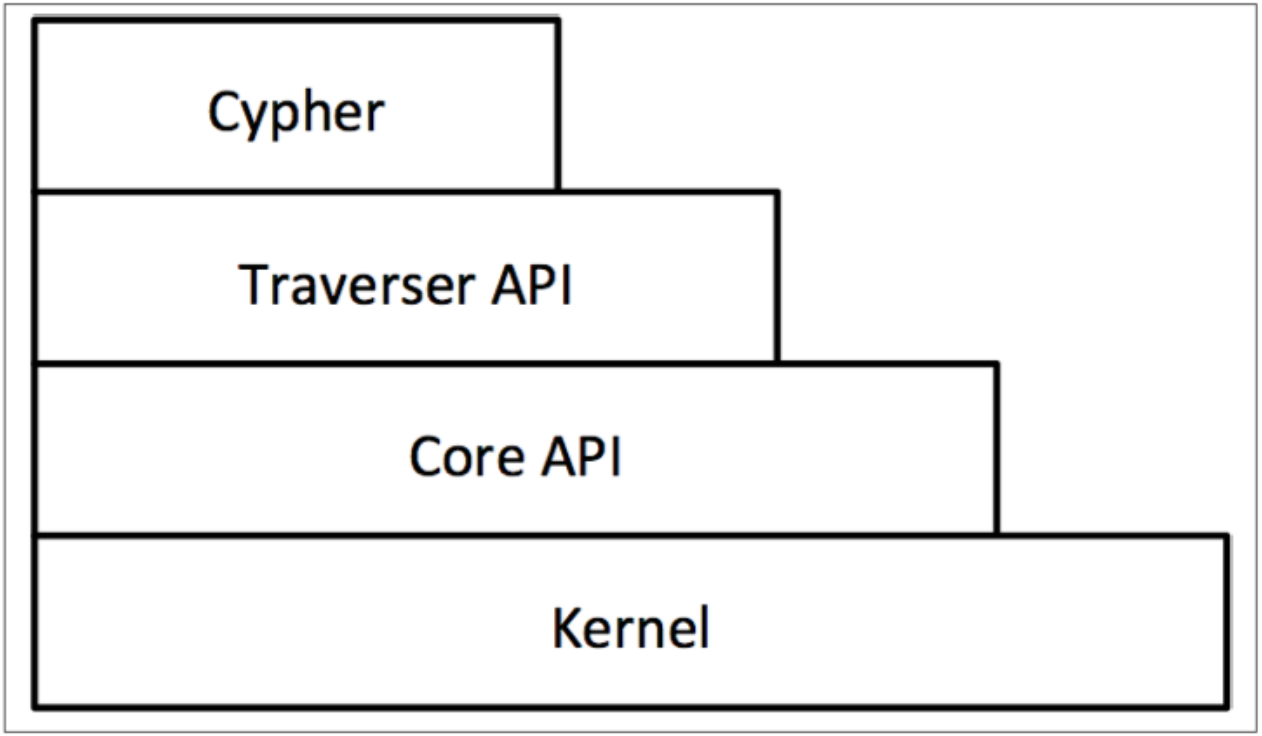
[Robinson et al. 2013]




http://neo4j.org/

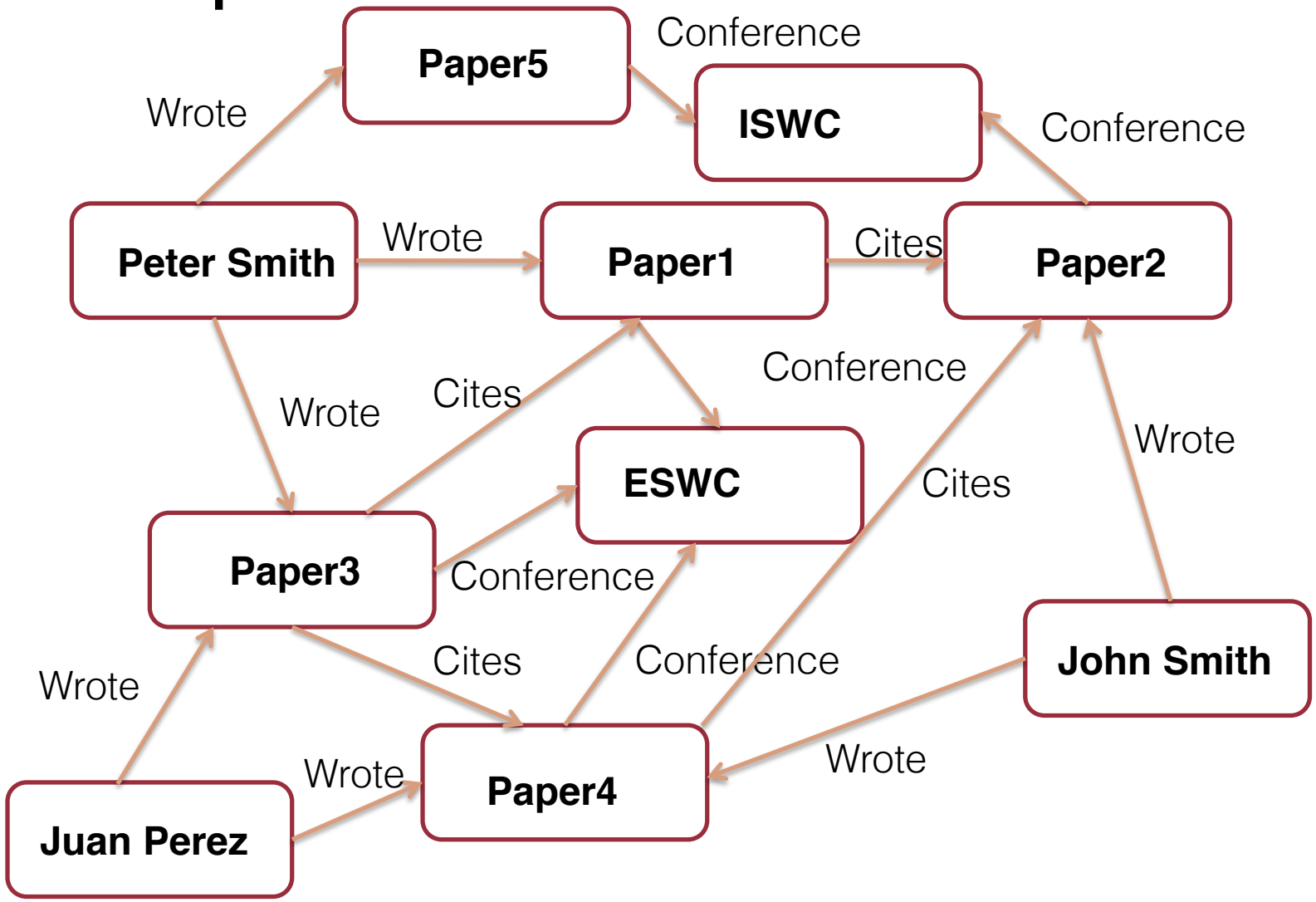
# Neo4J Logical View

[Robinson et al. 2013]




http://neo4j.org/

# Example


Paper5
Conference
ISWC
Conference
Wrote
Peter Smith
Wrote
Paper1
Cites
Paper2
Wrote
Cites
Conference
ESWC
Wrote
Cites
Paper3
Conference
Cites
Conference
John Smith
Wrote
Wrote
Paper4
Wrote
Juan Perez

# Neo4J API (1)

## Creating a graph and adding nodes

```
db = new GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );

Node author1 = db.createNode();
Node author2 = db.createNode();
Node author3 = db.createNode();
Node paper1 = db.createNode();
Node paper2 = db.createNode();
Node paper3 = db.createNode();
Node paper4 = db.createNode();
Node paper5 = db.createNode();
Node conference1 = db.createNode();
Node conference2= db.createNode();
```

# Neo4J API (2)

## Adding attributes to nodes

```
author1.setProperty(“firstname”,”Peter”);
author1.setProperty(“lastname”,”Smith”);

author2.setProperty(“firstname”,”Juan”);
author2.setProperty(“lastname”,”Perez”);

author3.setProperty(“firstname”,”John”);
author3.setProperty(“lastname”,”Smith”);


conference1.setProperty(“name”,”ESWC”);
conference2.setProperty(“name”,”ISWC”);
```

# Neo4J API (3)

## Adding relationships

```
RelationshipType relWrote = DynamicRelationshipType.withName(”WROTE”);
Relationship rel1=author1.createRelationshipTo(paper1,relWrote);
Relationship rel2=author1.createRelationshipTo(paper5,relWrote);
Relationship rel3=author1.createRelationshipTo(paper3,relWrote);
…
RelationshipType relCites = DynamicRelationshipType.withName(”CITES”);
Relationship rel8=paper1.createRelationshipTo(paper2,relCites);
Relationship rel9=paper3.createRelationshipTo(paper1,relCites);
Relationship rel10=paper3.createRelationshipTo(paper4,relCites);
Relationship rel11=paper4.createRelationshipTo(paper2,relCites);
…
RelationshipType relConf = DynamicRelationshipType.withName(”Conference”);
Relationship rel12=paper1.createRelationshipTo(conference1,relConf);
Relationship rel13=paper2.createRelationshipTo(conference2,relConf);
Relationship rel14=paper3.createRelationshipTo(conference1,relConf);
Relationship rel15=paper4.createRelationshipTo(conference1,relConf);
Relationship rel16=paper5.createRelationshipTo(conference2,relConf);
```

# Neo4J API (4)

## Creating indexes

```
IndexManager index = db.index();

Index<Node> authorIndex = index.forNodes( ”authors" );
//”authors” index name
Index<Node> paperIndex = index.forNodes( ”papers" );
Index<Node> conferenceIndex = index.forNodes( ”conferences" );

RelationshipIndex wroteIndex = index.forRelationships( ”write" );

RelationshipIndex citeIndex = index.forRelationships( ”cite" );

RelationshipIndex conferenceRelIndex = index.forRelationships
( ”conferenceRel" );
```

# Neo4J API (5)

## Indexing nodes

```
author1.setProperty("firstname","Peter");
author1.setProperty("lastname","Smith");

authorIndex.add(author1,lastname,author1.getProperty(lastname));

author2.setProperty("firstname","Juan");
author2.setProperty("lastname","Perez");

authorIndex.add(author2,lastname,author2.getProperty(lastname));

author3.setProperty("firstname","John ");
author3.setProperty("lastname","Smith");
authorIndex.add(author3,lastname,author3.getProperty(lastname));

conference1.setProperty("name","ESWC");
conferenceIndex.add(conference1,name,conference1.getProperty(name));

conference2.setProperty("name","ISWC");
conferenceIndex.add(conference2,name,conference2.getProperty(name));
```

# Neo4J: Cypher Query Language (1)

- START:
  - Specifies one or more starting points in the graph.
  - Starting points can be defined as index lookups or just by an element ID.
- MATCH:
  - Pattern matching to match based on the starting point(s)
- WHERE:
  - Filtering criteria.
- RETURN:
  - What is projected out from the evaluation of the query.

# Neo4J: Cypher Query Language (2)

Query: Is *Peter Smith connected to John Smith*.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[R*]->(m)
RETURN count(R)
```

# Neo4J: Cypher Query Language (2)

**Query**: Papers written by *Peter Smith*.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->(papers)
RETURN papers
```

# Neo4J: Cypher Query Language (3)

**Query**: Papers cited by a paper written by *Peter Smith*.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->()-[:CITES](papers)
RETURN papers
```

# Neo4J: Cypher Query Language (3)

**Query**: Papers cited by a paper written by *Peter Smith that have at most 20 cites*.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->()-[:CITES]->()-[:CITES]->(papers)
WITH COUNT(papers) as cites
WHERE cites < 21
RETURN papers
```

# Neo4J: Cypher Query Language (4)

**Query**: Papers cited by a paper written by *Peter Smith* or cited by papers cited by a paper written by *Peter Smith*.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->()-[:CITES*1..2](papers)
RETURN papers
```

# Neo4J: Cypher Query Language (5)

**Query**: Number of papers cited by a paper written by *Peter Smith* or cited by papers cited by a paper written by *Peter Smith*.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->()-[:CITES*1..2](papers)
RETURN count(papers)
```

# Neo4J: Cypher Query Language (6)

**Query**: Number of papers cited by a paper written by *Peter Smith* or cited by papers cited by a paper written by *Peter Smith*, and have been published in ESWC.

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->()-[:CITES*1..2](papers)
WHERE papers.conference! =‘ESWC’
RETURN count(papers)
```

Exclamation mark in `papers.conference` ensures that papers without the conference attribute are not included in the output.

# Neo4J: Cypher Query Language (7)

**Query**: Number of papers cited by a paper written by *Peter Smith* or cited by papers cited by a paper written by *Peter Smith*, have been published in ESWC and have at most 4 co-authors .

```
START author=node:authors(
                'firstname:”Peter" AND lastname:”Smith"')
MATCH (author)-[:WROTE]->()-[:CITES*1..2](papers)-
                                [:Was-Written]->authorFinal
WHERE papers.conference! =‘ESWC’
WITH author, count(authorFinal) as authorFinalCount
WHERE authorFinalCount < 4
RETURN author, authorFinalCount
```

Exclamation mark in `papers.conference` ensures that papers without the conference attribute are not included in the output.

# Neo4J API-Cypher

Query: *Peter Smith Neighborhood*

```
static Neo4j TestGraph;
String Query = "start n=node:node_auto_index(idNode='PeterSmith')" +
               "match n-[r]->m return m.idNode";
ExecutionResult result = TestGraph.query(Query);
boolean res = false;
for ( Map<String, Object> row : result ){
 for ( Map.Entry<String, Object> column : row.entrySet() ){
     print(column.getValue().toString());
     res = true;
}
}
if (!res) print("  No result");
```

# Neo4J API Summary

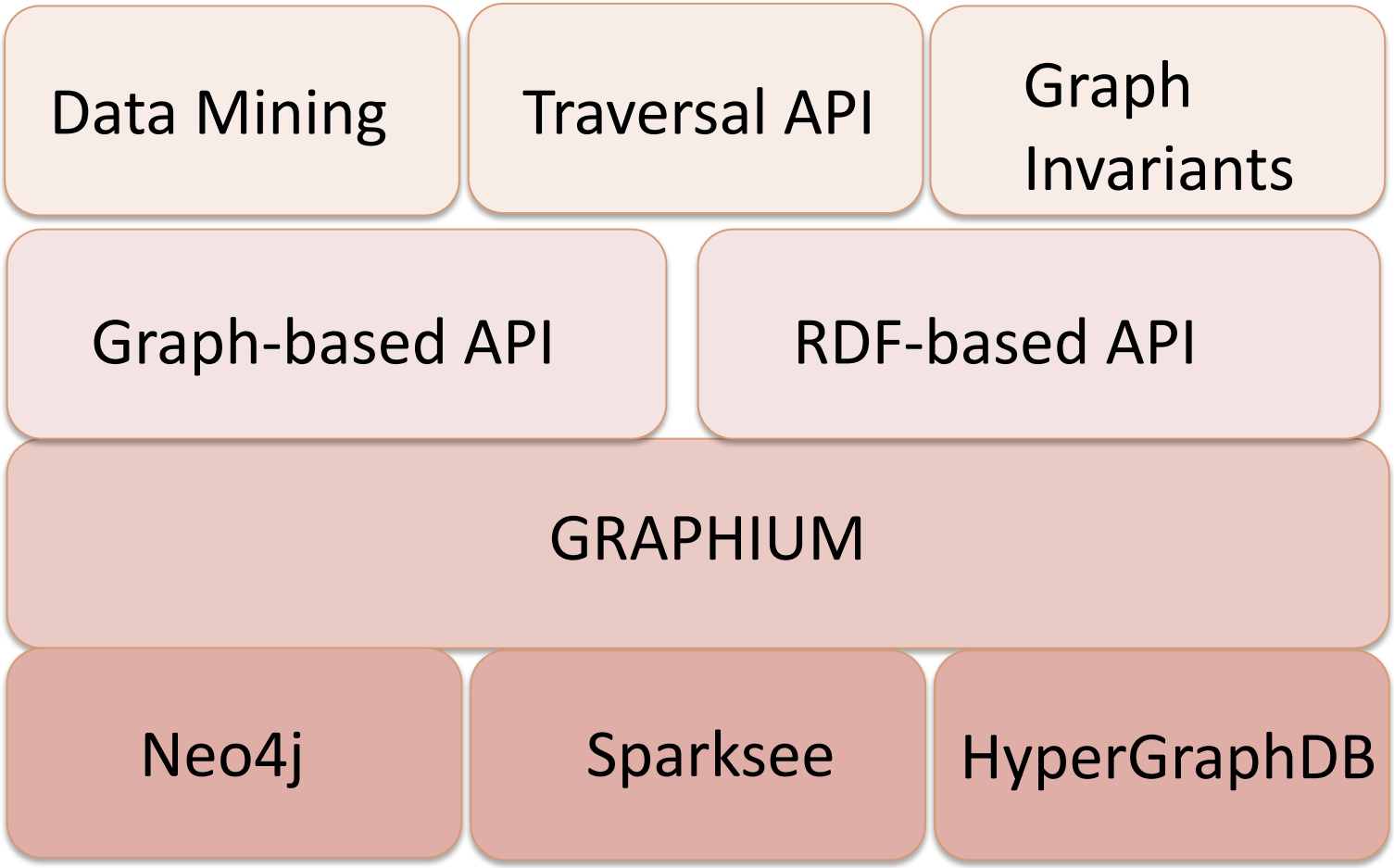
| Operation | Description |
|---|---|
| GraphDatabaseFactory | Creates a empty graph |
| registerShutdownHook | Removes a graph |
| createNode | Creates a new empty node of a given node type |
| AddNode | Copies a node |
| createRelationshipTo | Adds a new edge of some edge type |
| forNodes | Index for a Node |
| IndexManager | Index Manager |
| forRelationships | Index for a Relationship |
| add | Add an object to an index |
| TraversalDescription | Different types of strategies to traverse a graph |
| Traversal Branch Selector | preorderDepthFirst, postorderDepthFirst, preorderBreadthFirst, postorderBreadthFirst |

# Graphium (http://graphium.ldc.usb.ve)

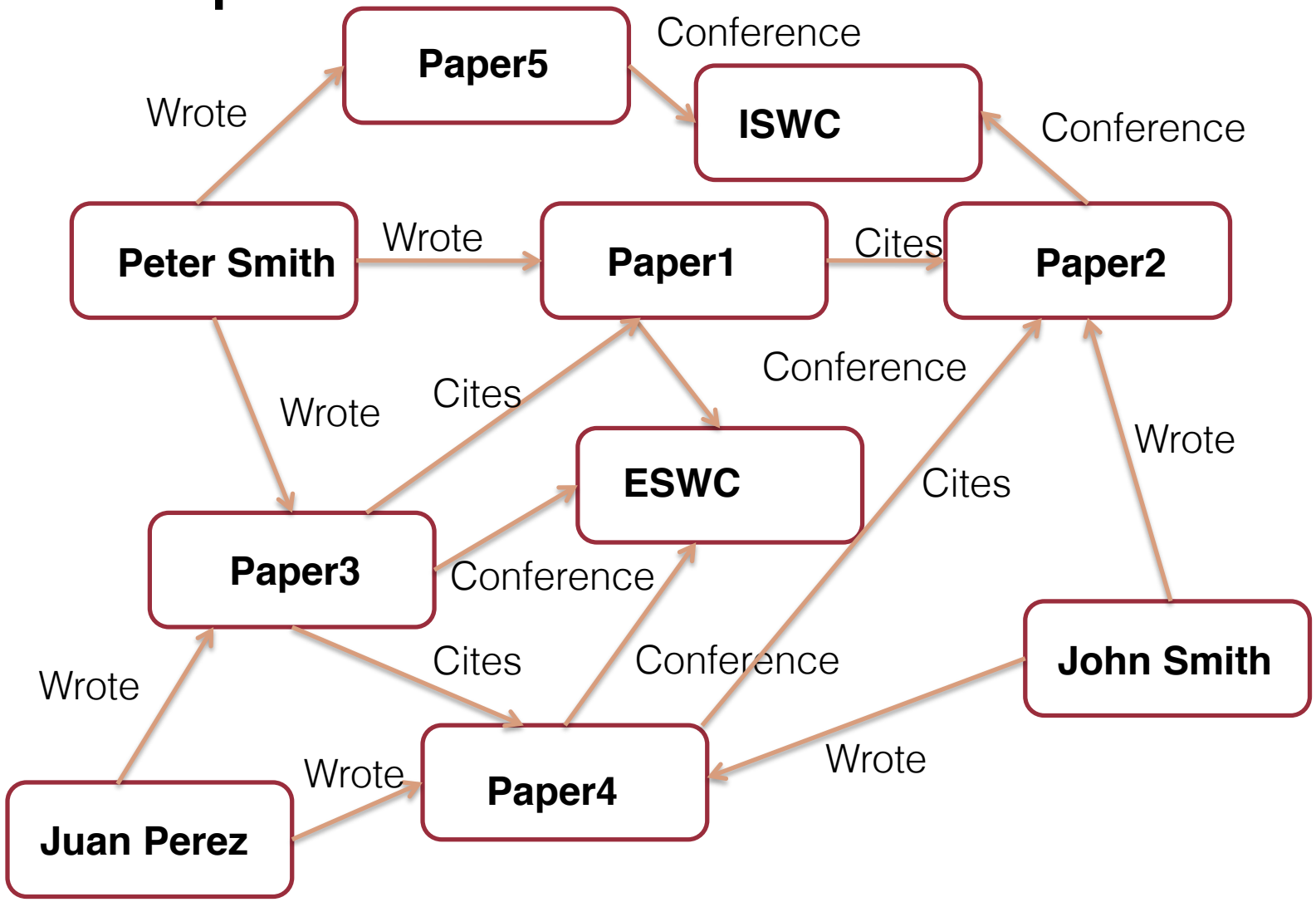
[ Flores et al. 2013, De Abreu et al. 2013]

- Abstract **Graph Database Engine** built on top of: Neo4j, Sparksee, and HyperGraphDB.
- Java **Graph-based** and **RDF-based** API.
- Labeled attribute **multigraph**.
- Nodes and edges can have **properties**.
- There are **no restrictions** on the number of edges between two nodes.
- **Loops** are allowed.
- Attributes are **single valued**.
- Different types of **traversal strategies**.
- Different **data mining strategies**.

# Graphium Architecture



# Example


Paper5
Conference
ISWC
Conference
Wrote
Peter Smith
Wrote
Paper1
Cites
Paper2
Conference
Wrote
Cites
Wrote
ESWC
Cites
Paper3
Conference
Cites
Conference
John Smith
Wrote
Wrote
Paper4
Wrote
Juan Perez

# Graphium *Graph-based* API (1)

## Creating a graph

```
GraphDB db;

db = new Neo4j();
// or…
db = new DEXDB(); // Sparksee was previously known as DEX
// or…
db = new HyperGraphDB();
```

# Graphium *Graph-based* API (2)

## Adding nodes

```
db.addNode(”Peter Smith”);
db.addNode(”Juan Perez”);
db.addNode(”John Smith”);

db.addNode(”ESWC”);
db.addNode(”ISWC”);

db.addNode(”Paper1”);
db.addNode(”Paper2”);
db.addNode(”Paper3”);
db.addNode(”Paper4”);
db.addNode(”Paper5”);
```

# Graphium *Graph-based* API (3)

## Adding relationships

```
db.addEdge(new Edge(“WROTE”, “Peter Smith”, “Paper1”));
db.addEdge(new Edge(“WROTE”, “Peter Smith”, “Paper3”));
db.addEdge(new Edge(“WROTE”, “Peter Smith”, “Paper5”));
…

db.addEdge(new Edge(“CITES”, “Paper1”, “Paper2”));
db.addEdge(new Edge(“CITES”, “Paper3”, “Paper1”));
db.addEdge(new Edge(“CITES”, “Paper3”, “Paper4”));
db.addEdge(new Edge(“CITES”, “Paper4”, “Paper2”));
…

db.addEdge(new Edge(“Conference”, “Paper1”, “ESWC”));
db.addEdge(new Edge(“Conference”, “Paper2”, “ISWC”));
db.addEdge(new Edge(“Conference”, “Paper3”, “ESWC”));
db.addEdge(new Edge(“Conference”, “Paper4”, “ESWC”));
db.addEdge(new Edge(“Conference”, “Paper5”, “ISWC”));
```

# Graphium *Graph-based* API (4)

## Adjacency: "*Papers cited by Paper3*"

```
GraphIterator<String> it;

it = db.adj("Paper3", "CITES");
while (it.hasNext()) {
    … it.next() …
}
it.close();
```

# Graphium *Graph-based* API (5)

Traversing the graph: "*2-Hops from Peter Smith*"

```
GraphIterator<String> it;

it = db.kHops("Peter Smith", 2);
while (it.hasNext()) {
    … it.next() …
}
it.close();
```

# Graph Mining Tasks

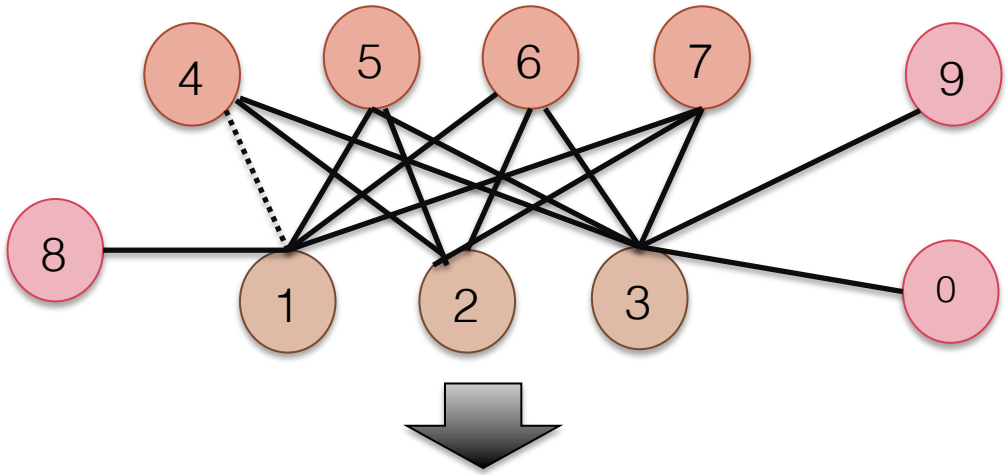
These tasks evaluate the engine performance while executing the following graph mining:

- Graph summarization.
- Dense subgraph.
- Shortest path.

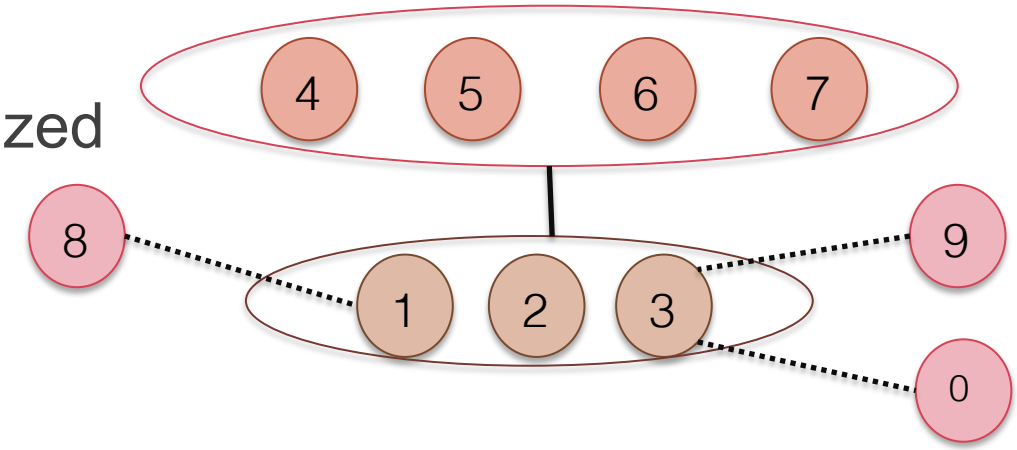
# Graph Summarization

[Navlakha et al. 2008]

Original Graph




Summarized Graph



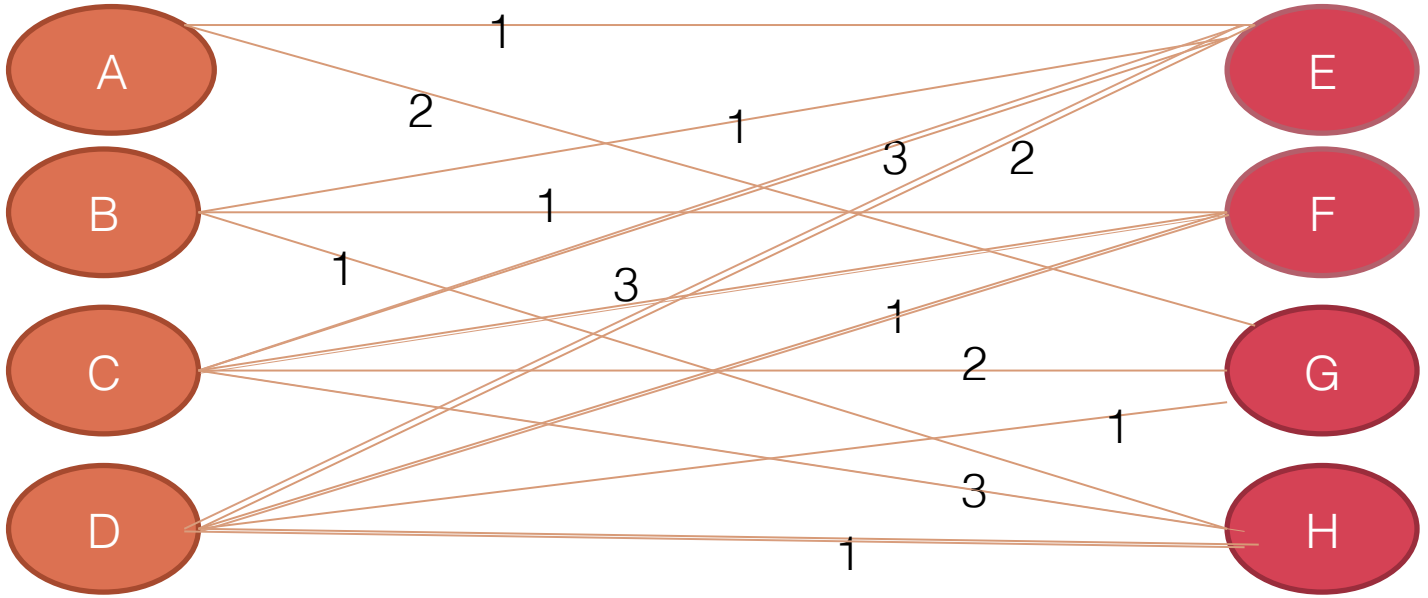

Corrections

Add(8,1)
Add(9,3)
Add(0.3)
Del(4,1)

Cost = 1(superedge) + 4(corrections)= 5

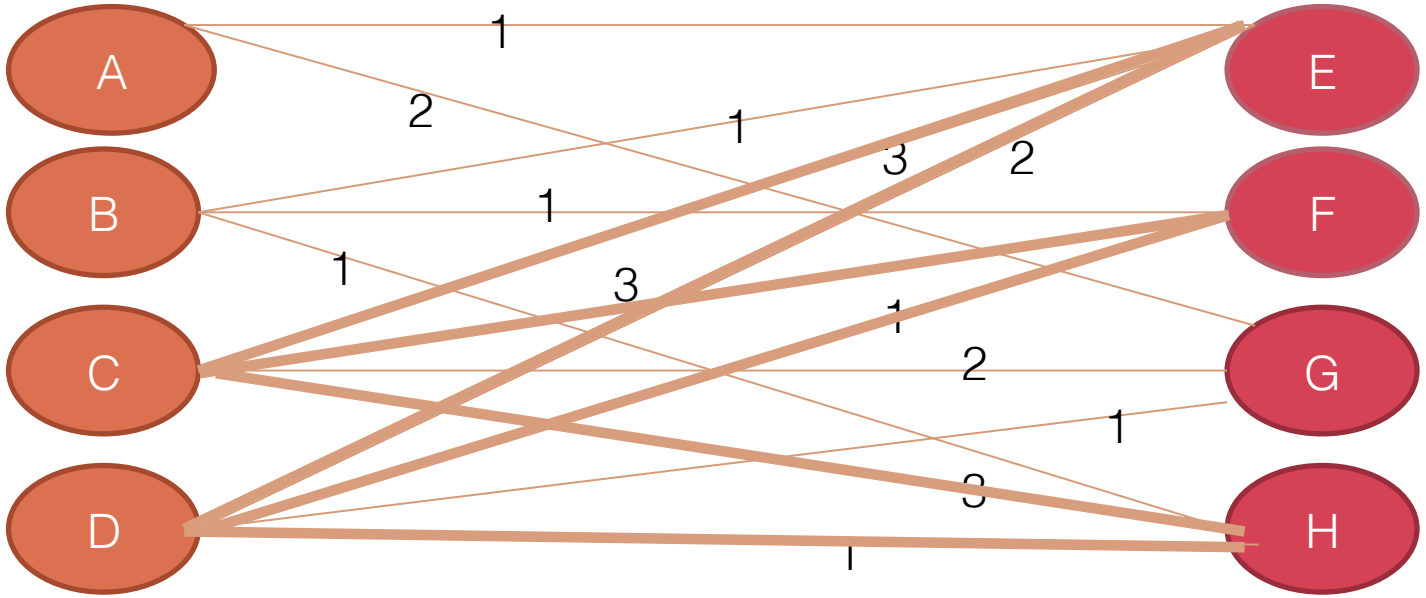
# Dense Subgraph

[Khuller et al. 2010]



# Dense Subgraph

[Khuller et al. 2010]




Given a subset of nodes E' compute the densest subgraph containing them, i.e., density of E' is maximized:

$$density(E') = \sum_{(a,b)\ in\ E'} \frac{weight(a,b)}{|E'|}$$

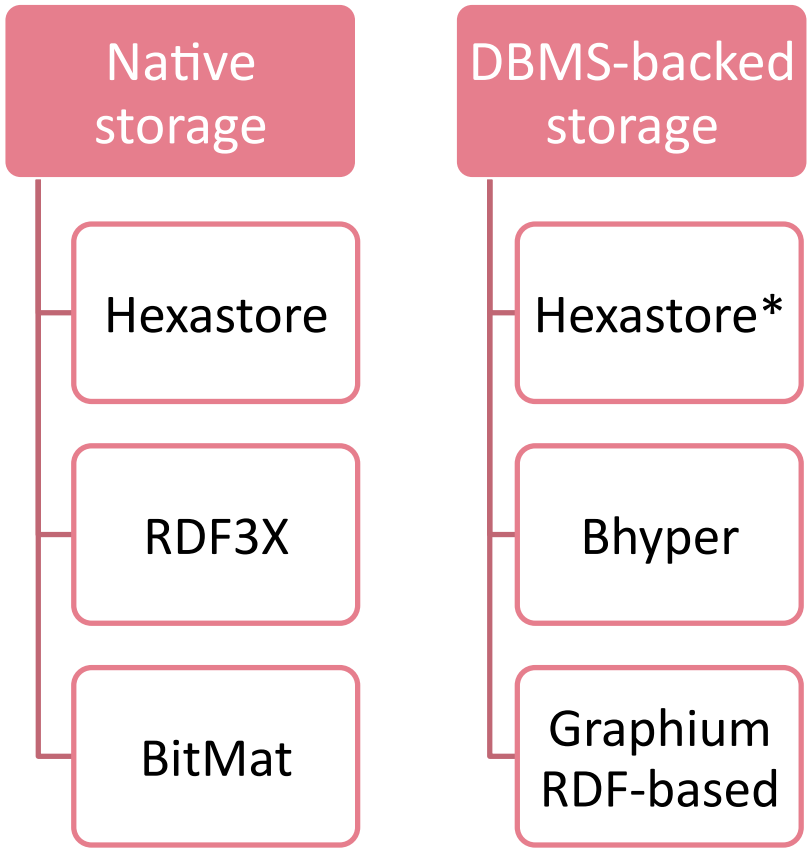
# Shortest Path

- Given two nodes finds the shortest path between them.
- Implemented using Iterative **Depth First Search** (DFID).
- Ten node pairs are taken at random from each graph, and then the Shortest Path algorithm is executed on each pair of nodes

# LUNCH BREAK

# 4 RDF GRAPH ENGINES

# RDF-Based Graph Database Engines



# In the beginnings …

… there were dinosaurs

# In the beginnings …

- Traditionally:
  - Use Relational DBMS' to store data
  - Early Approach: **one large SPO table** (physical)
    - Pros: simple schema, fast updates (when no index present), fast single triple pattern queries
    - Cons: not suitable for multi-join queries, results in multiple self-joins

# In the beginnings …

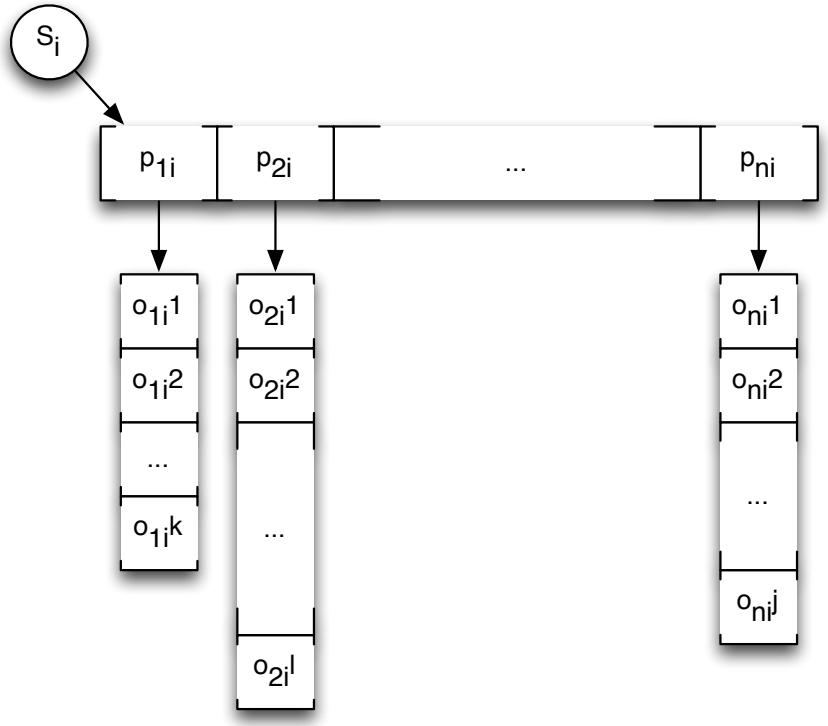
- Traditionally:
  - Use Relational DBMS' to store data
  - Further Approaches: **property tables** (multiple flavors)
    - Pros: fast retrieval of subject / object pairs for given predicates
    - Cons: not easy to solve queries where predicate is unbound & harder to maintain schema (a table for each known predicate)

# RDF-tailored management approaches

- Same conceptual approach: **one large SPO table** (but with RDF-specific physical organization):
  - Triples are indexed in all 3!=6 possible permutations: SPO, PSO, OSP, OPS, PSO, POS
  - Logical extension of the vertical partitioning proposed in [Abadi et al 2008] (the PSO index = generalization of the SO table)
  - Each index (i.e. SPO) stores partial triple pattern cardinalities
  - Resulting in 6 tree-based indexes, each 3 levels deep
- First proposed in **Hexastore**

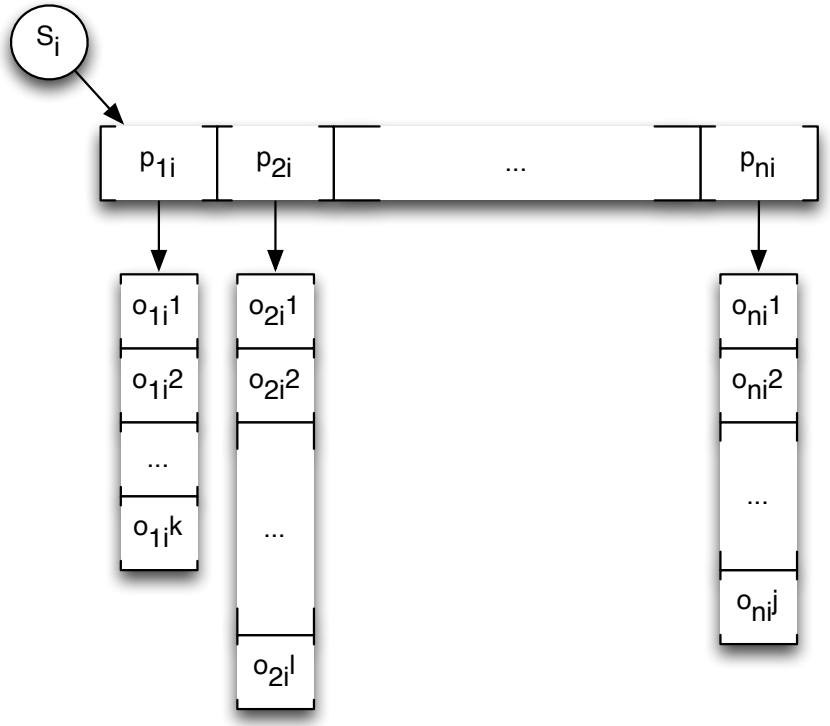
  [Weiss et al. 2008]

# Hexastore / Conceptual Model



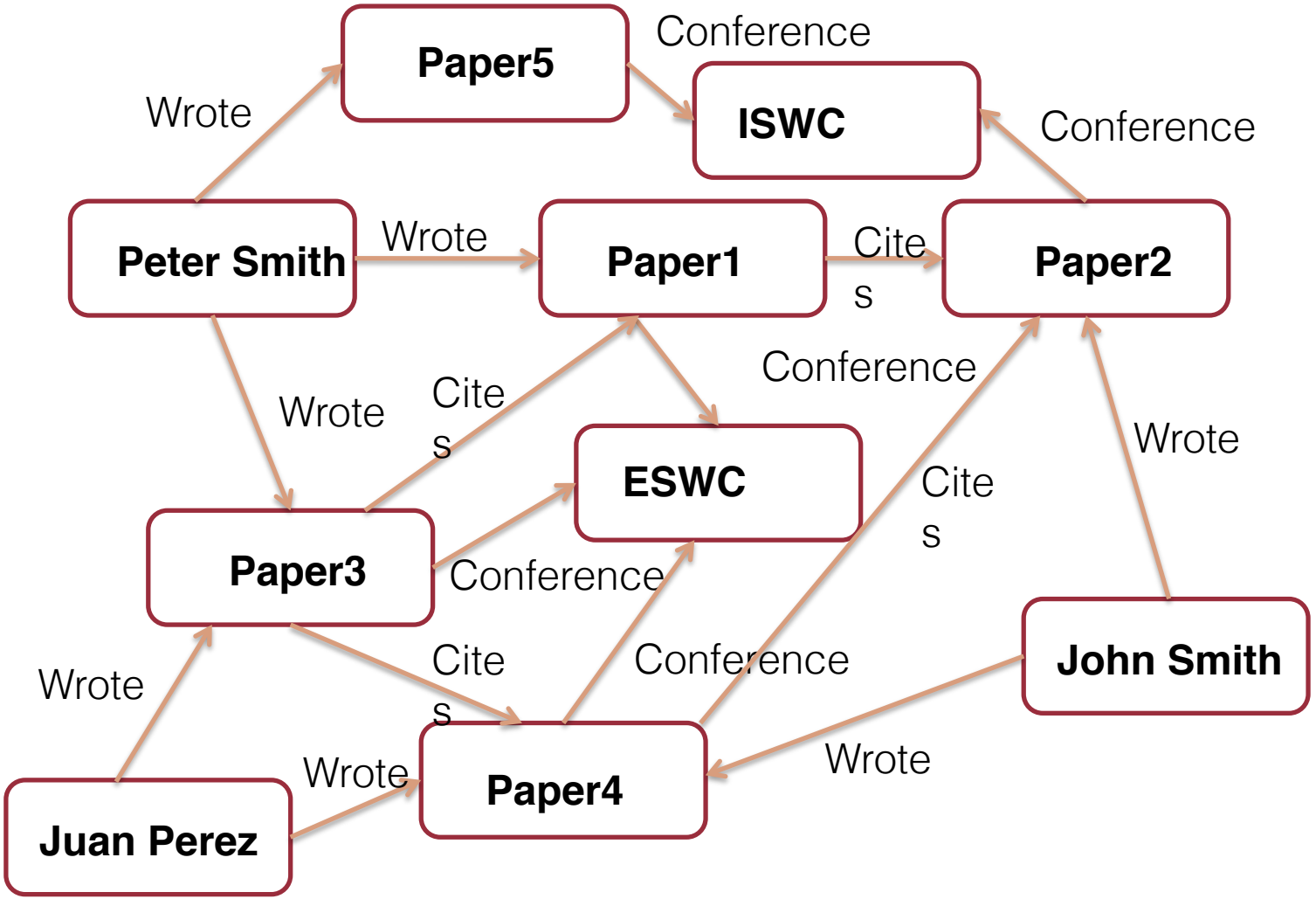

- 2 main operations: (input = **tp**: partially bounded triple pattern, i.e. *<bound_subject> <some predicate> ?o*)
  - *cardinality* = *getCardinality*(**tp**)
  - *term_set* = *getSet*(**tp**, **set**)
- Resulting *term_set* is sorted

# Hexastore / Conceptual Model

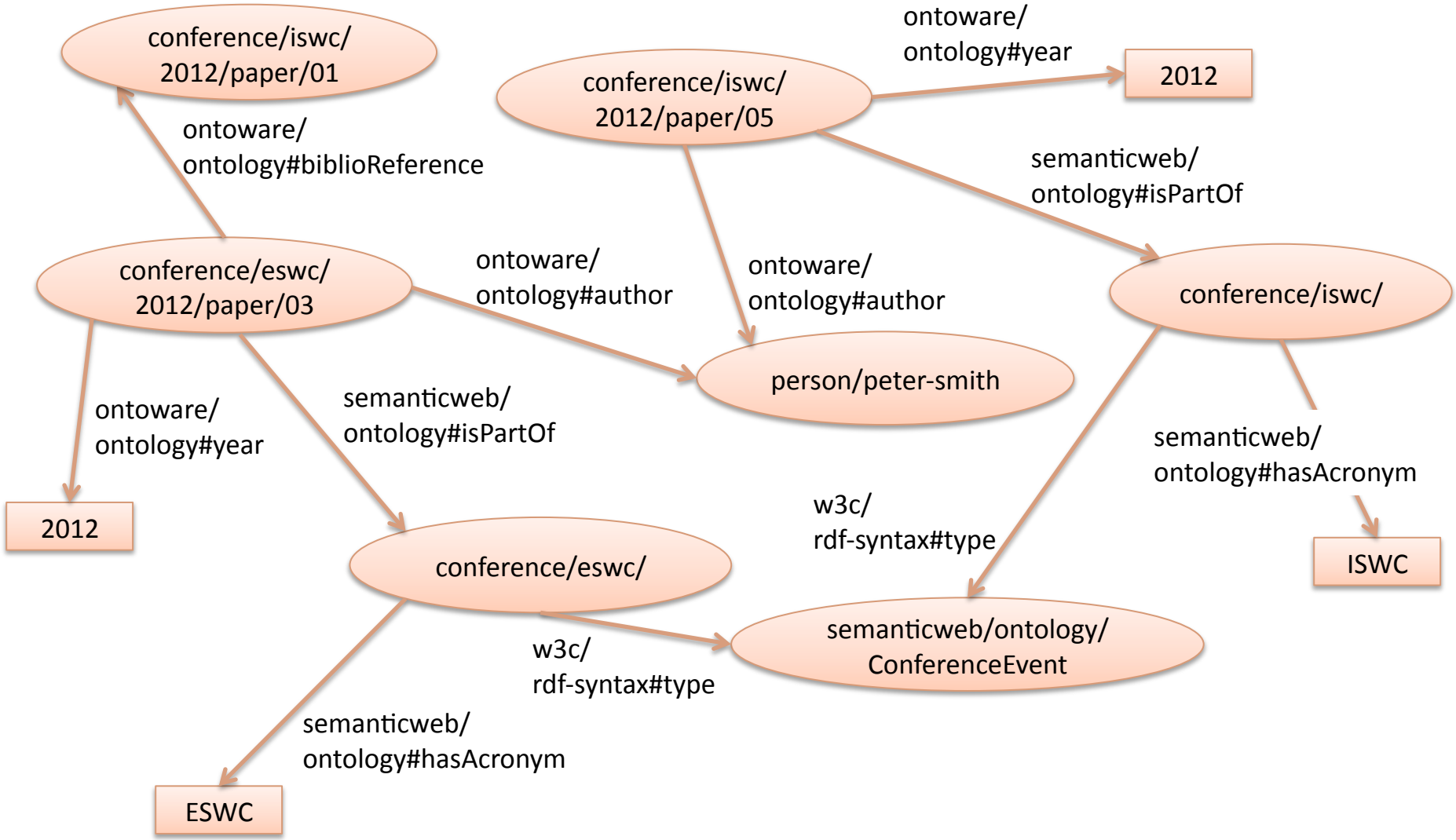



- Pros:
  - The optimal index permutation & level is used (i.e. <bound_s> ?p ?o : SPO level 1)
  - Ability to make use of fast merge-joins
- Cons:
  - Higher upkeep cost: a theoretical upper bound of 5x space increase over original (encoded) data

# An Example (Publications): logical representation



# RDF Subgraph for the Publication Graph



# An Example (Publications): first step, encode strings

Hexastore Mapping Dictionary

| | |
|---|---|
| *http://data.semanticweb.org/conference/eswc/2012/paper/01* | ***1*** |
| *http://data.semanticweb.org/conference/iswc/2012/paper/02* | ***2*** |
| *http://data.semanticweb.org/conference/eswc/2012/paper/03* | ***3*** |
| *http://data.semanticweb.org/conference/eswc/2012/paper/04* | ***4*** |
| *http://data.semanticweb.org/conference/iswc/2012/paper/05* | ***5*** |
| *"ESWC2012"* | ***6*** |
| *"ISWC2012"* | ***7*** |
| *http://data.semanticweb.org/conference/iswc/2012* | ***8*** |
| *http://data.semanticweb.org/conference/eswc/2012* | ***9*** |
| *http://data.semanticweb.org/ns/swc/ontology#ConferenceEvent* | ***10*** |
| *http://data.semanticweb.org/ns/swc/ontology#isPartOf* | ***11*** |
| *http://swrc.ontoware.org/ontology#author* | ***12*** |
| *http://swrc.ontoware.org/ontology#year* | ***13*** |
| *http://swrc.ontoware.org/ontology#biblioReference* | ***14*** |
| *"2012"* | ***15*** |
| *http://www.w3.org/1999/02/22-rdf-syntax-ns#type* | ***16*** |
| *http://data.semanticweb.org/ns/swc/ontology#hasAcronym* | ***17*** |
| *http://data.semanticweb.org/person/peter-smith* | ***18*** |
| *http://data.semanticweb.org/person/juan-perez* | ***19*** |
| *http://data.semanticweb.org/person/john-smith* | ***20*** |

# An Example (Publications): first step, encode strings

Hexastore Mapping Dictionary

| | |
|---|---|
| *http://data.semanticweb.org/conference/eswc/2012/paper/01* | ***1*** |
| *http://data.semanticweb.org/conference/iswc/2012/paper/02* | ***2*** |
| *http://data.semanticweb.org/conference/eswc/2012/paper/03* | ***3*** |
| *http://data.semanticweb.org/conference/eswc/2012/paper/04* | ***4*** |
| *http://data.semanticweb.org/conference/iswc/2012/paper/05* | ***5*** |
| *"ESWC2012"* | ***6*** |
| *"ISWC2012"* | ***7*** |
| *http://data.semanticweb.org/conference/iswc/2012* | ***8*** |
| *http://data.semanticweb.org/conference/eswc/2012* | ***9*** |
| *http://data.semanticweb.org/ns/swc/ontology#ConferenceEvent* | ***10*** |
| *http://data.semanticweb.org/ns/swc/ontology#isPartOf* | ***11*** |
| *http://swrc.ontoware.org/ontology#author* | ***12*** |
| *http://swrc.ontoware.org/ontology#year* | ***13*** |
| *http://swrc.ontoware.org/ontology#biblioReference* | ***14*** |
| *"2012"* | ***15*** |
| *http://www.w3.org/1999/02/22-rdf-syntax-ns#type* | ***16*** |
| *http://data.semanticweb.org/ns/swc/ontology#hasAcronym* | ***17*** |
| *http://data.semanticweb.org/person/peter-smith* | ***18*** |
| *http://data.semanticweb.org/person/juan-perez* | ***19*** |
| *http://data.semanticweb.org/person/john-smith* | ***20*** |

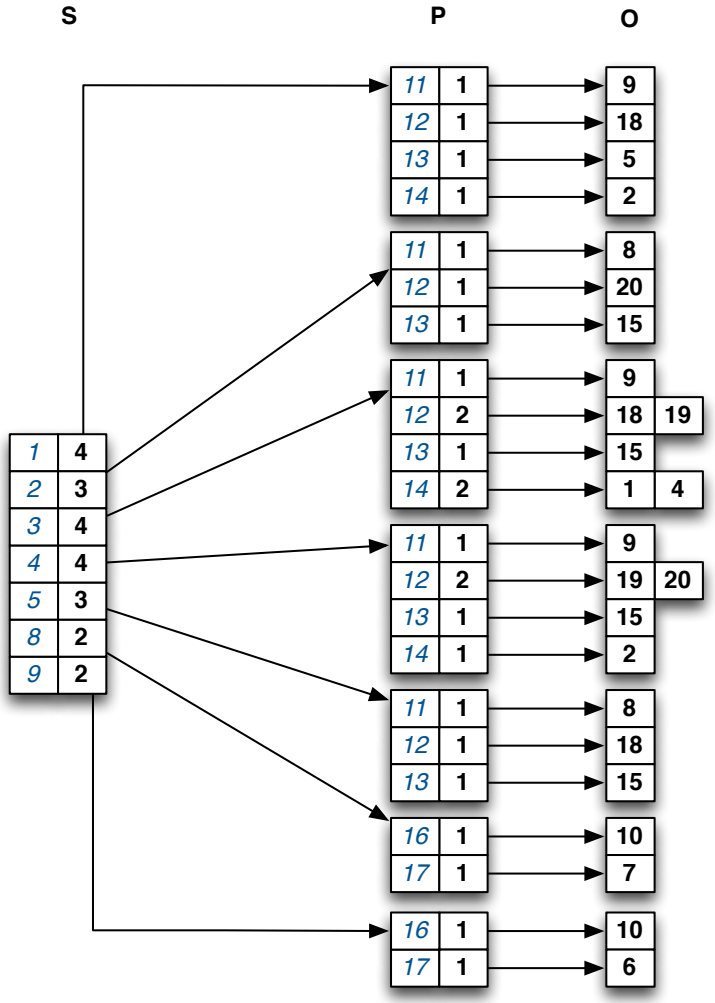
- Next, index original encoded RDF file …

# Hexastore / Physical implementation. Native storage

- Uses **vector-storage** [Weiss et al 2009]
- Each index level:
  - partially sorted vector (on disk, one file per level)
  - each record:

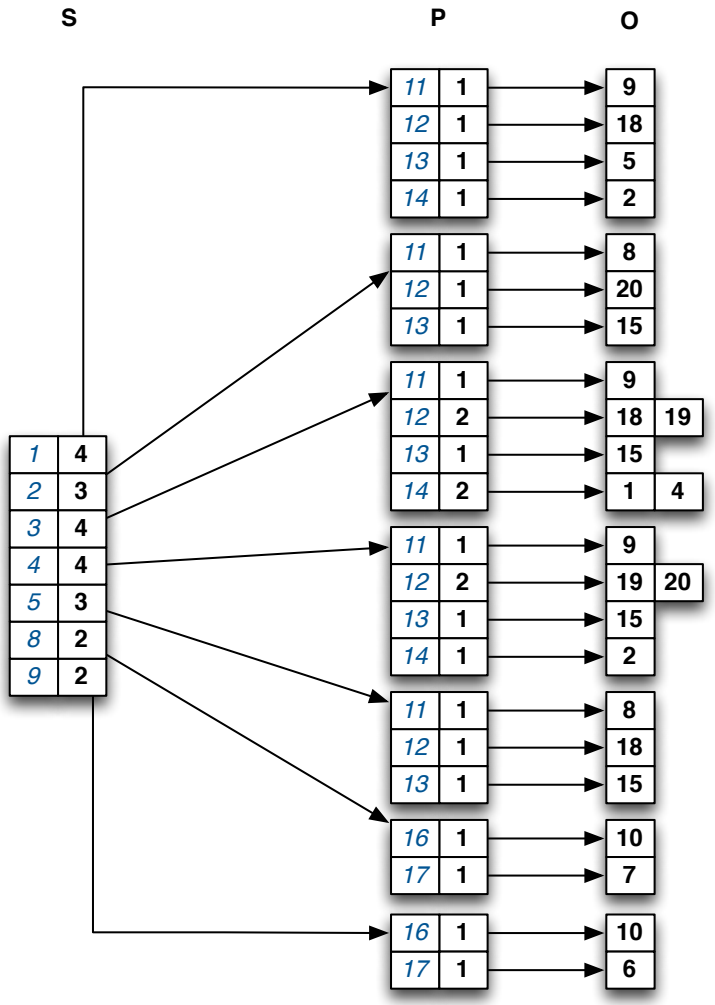
<rdf term id, cardinality, pointer to set>

  - first level gets special treatment:
    - fully sorted vector: search = binary search, *O(log(N))*
    - Hint: using an additional hash can circumvent the binary search



# Hexastore / Physical implementation. Native storage

- **Pros**:
  - Optimized for read-intensive / static workloads
  - Optimal / fast triple pattern matching
  - No comparisons when scanning since each (partial) cardinality is known !
- **Cons**:
  - Not suitable for write intensive workloads (worst case can result in rewrite of entire index!)
  - Hypothesis (not yet tested): SSD media may be adequate…



# Hexastore / Physical implementation. DBMS (managed) storage

- Builds on top of sorted indexes:
  - B+Tree, LSM Tree, etc
- Each index level:
  - Same sorted index structure (but not necessarily)
  - each record:

<rdf term id list, cardinality>

  - Search cost = as provided by the managed index structure

**S**

| | |
|---|---|
| 1 | 4 |
| 2 | 3 |
| 3 | 4 |
| 4 | 4 |
| 5 | 3 |
| 8 | 2 |
| 9 | 2 |

**SP**

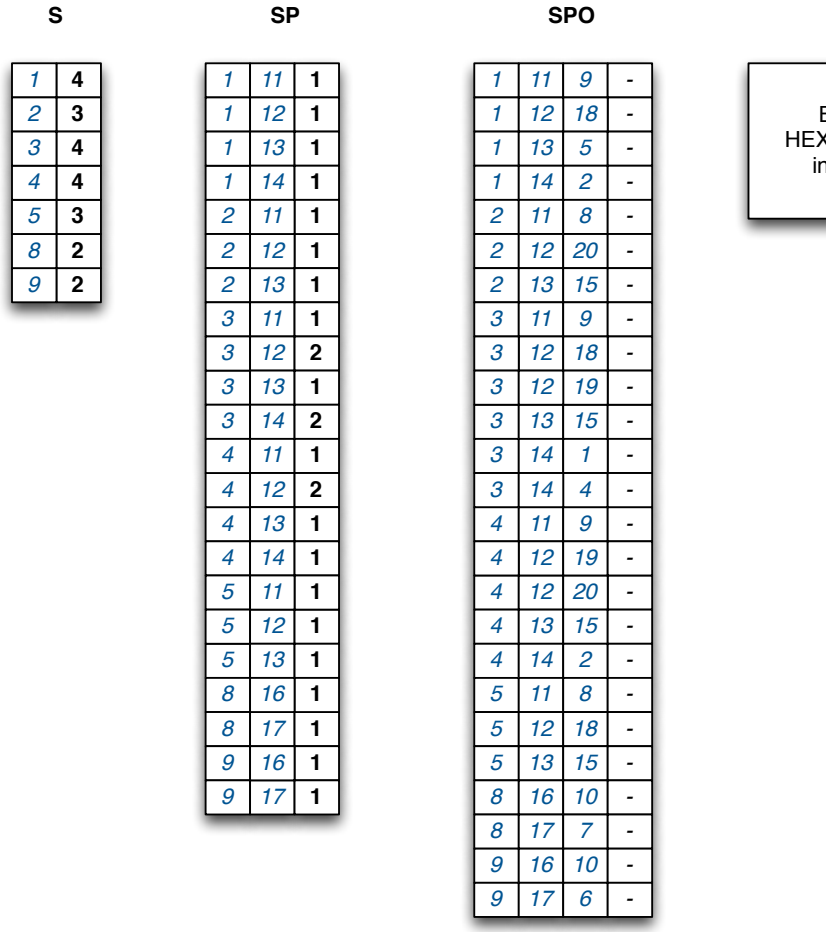
| | | |
|---|---|---|
| 1 | 11 | 1 |
| 1 | 12 | 1 |
| 1 | 13 | 1 |
| 1 | 14 | 1 |
| 2 | 11 | 1 |
| 2 | 12 | 1 |
| 2 | 13 | 1 |
| 3 | 11 | 1 |
| 3 | 12 | 2 |
| 3 | 13 | 1 |
| 3 | 14 | 2 |
| 4 | 11 | 1 |
| 4 | 12 | 2 |
| 4 | 13 | 1 |
| 4 | 14 | 1 |
| 5 | 11 | 1 |
| 5 | 12 | 1 |
| 5 | 13 | 1 |
| 8 | 16 | 1 |
| 8 | 17 | 1 |
| 9 | 16 | 1 |
| 9 | 17 | 1 |

**SPO**

| | | | |
|---|---|---|---|
| 1 | 11 | 9 | - |
| 1 | 12 | 18 | - |
| 1 | 13 | 5 | - |
| 1 | 14 | 2 | - |
| 2 | 11 | 8 | - |
| 2 | 12 | 20 | - |
| 2 | 13 | 15 | - |
| 3 | 11 | 9 | - |
| 3 | 12 | 18 | - |
| 3 | 12 | 19 | - |
| 3 | 13 | 15 | - |
| 3 | 14 | 1 | - |
| 3 | 14 | 4 | - |
| 4 | 11 | 9 | - |
| 4 | 12 | 19 | - |
| 4 | 12 | 20 | - |
| 4 | 13 | 15 | - |
| 4 | 14 | 2 | - |
| 5 | 11 | 8 | - |
| 5 | 12 | 18 | - |
| 5 | 13 | 15 | - |
| 8 | 16 | 10 | - |
| 8 | 17 | 7 | - |
| 9 | 16 | 10 | - |
| 9 | 17 | 6 | - |

# Hexastore / Physical implementation. DBMS (managed) storage

- **Pros**:
  - Flexible design: each index can be configured for a different physical data-structure:
    - i.e. PSO=B+Tree, OSP=LSM Tree
  - Separation of concerns (the underlying DBMS manages and optimizes for access patterns, cache, etc…)
- **Cons**:
  - Less control over fine-grained data storage (up to the level permitted by the DBMS API)
  - Usually less potential for native optimization



# RDF3x (1)

[Neumann and Weikum 2008]

- **RISC** style operators.
- Implements the traditional **Optimize-then-Execute** paradigm, to identify left-linear plans of star-shaped sub-queries of basic triple patterns.
- Makes use of **secondary memory** structures to locally store RDF data and reduce the overhead of I/O operations.
- Registers aggregated count or **number of occurrences**, and is able to detect if a query will return an empty answer.

# RDF3x (2)

[Neumann and Weikum 2008]

- **Triple table**, each row represents a triple.
- **Mapping dictionary**, replacing all the literal strings by an id; two dictionary indices.
- Compressed clustered **B$^+$-tree** to index all triples:
  - Six permutations of subject, predicate and object; each permutation in a different index.
  - Triples are sorted lexicographically in each B$^+$-tree.

# RDF3x (3)

**Mapping dictionary**
**Triple Table**

| Subject | Property | Object |
|---|---|---|
| _:id0 | type | Term |
| _:id0 | forOffice | AZ |
| AZ | type | Office |
| _:id1 | sponsor | HR45 |
| Wigle | sponsor | HR44 |

Dictionary Table

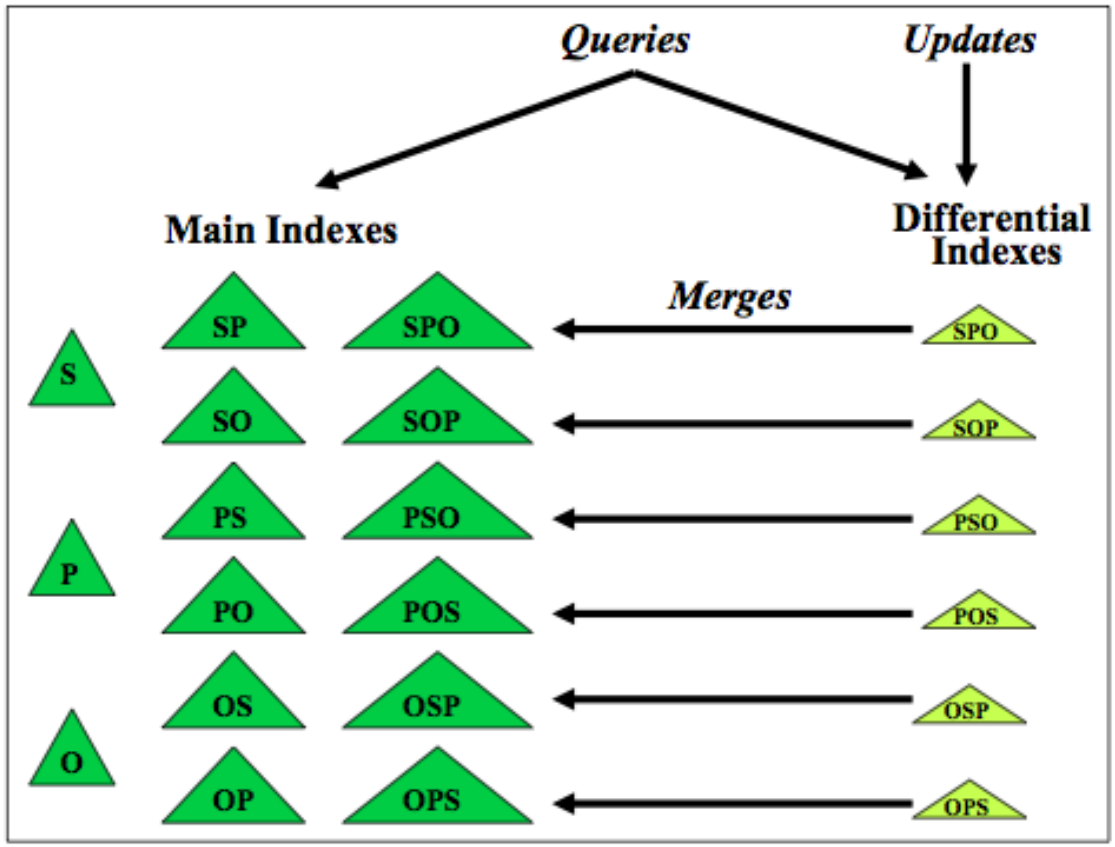
| ID | Value |
|---|---|
| 0 | _:id0 |
| 1 | AZ |
| 2 | _:id1 |
| 3 | Wigle |
| 4 | type |
| 5 | forOffice |
| 6 | sponsor |
| 7 | Term |
| 8 | Office |
| 9 | HR45 |
| 10 | HR44 |

Triples Table

| Subject | Property | Object |
|---|---|---|
| 0 | 4 | 7 |
| 0 | 5 | 1 |
| 1 | 4 | 8 |
| 2 | 6 | 9 |
| 3 | 6 | 10 |

# RDF3x (4)

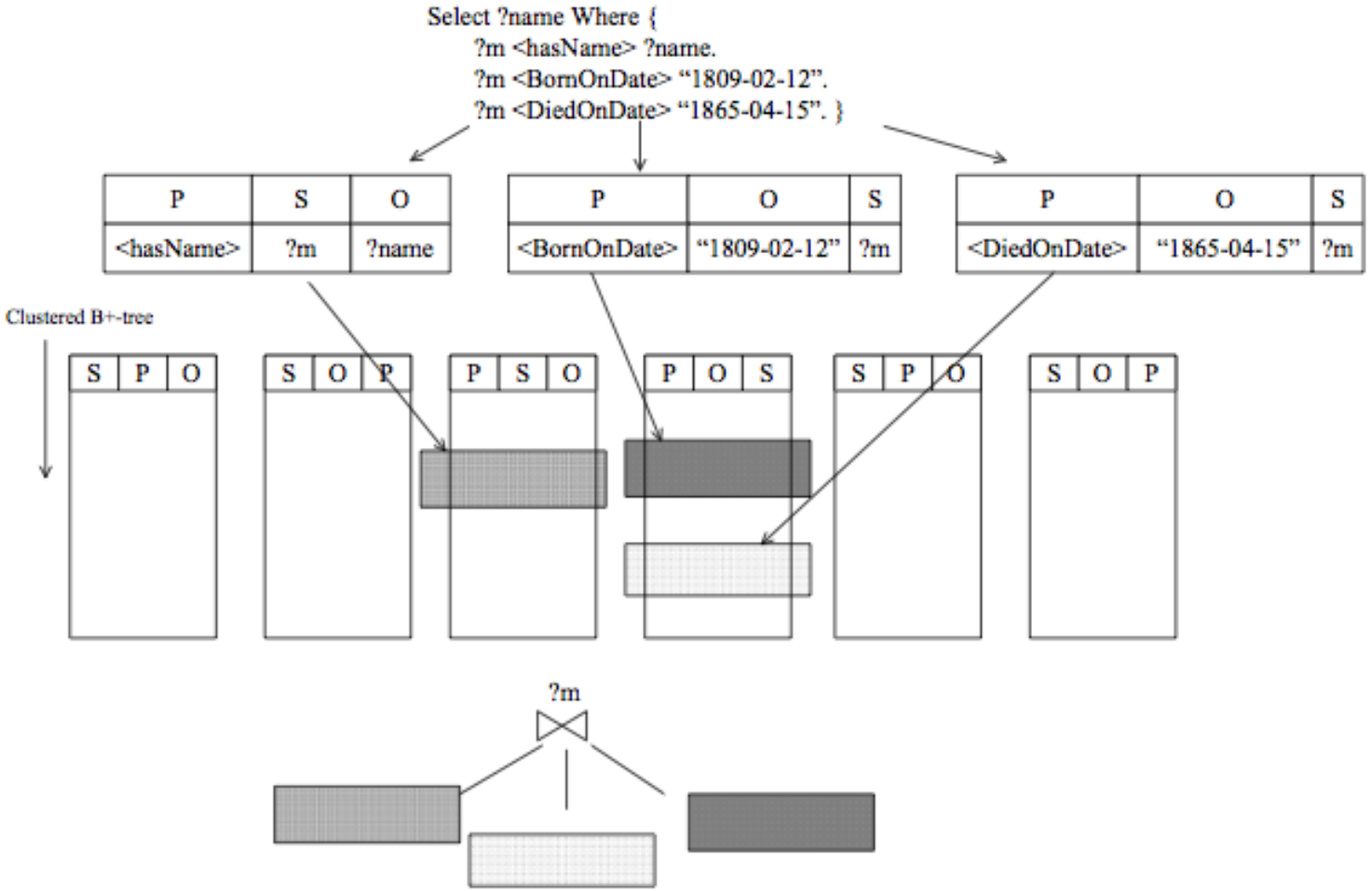
[Figure from Neumann et al 2010]



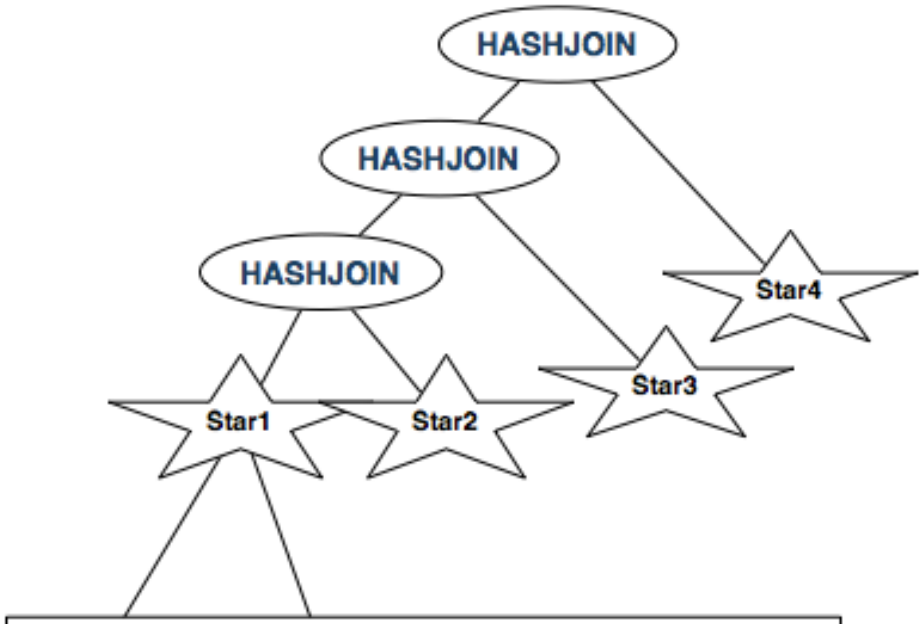

- Compressed B⁺-trees.
- One B⁺-tree per pattern.
- Structure is updatable.

# RDF3x (5)

[Figure from Lei Zou http://hotdb.info/slides/RDF-talk.pdf]

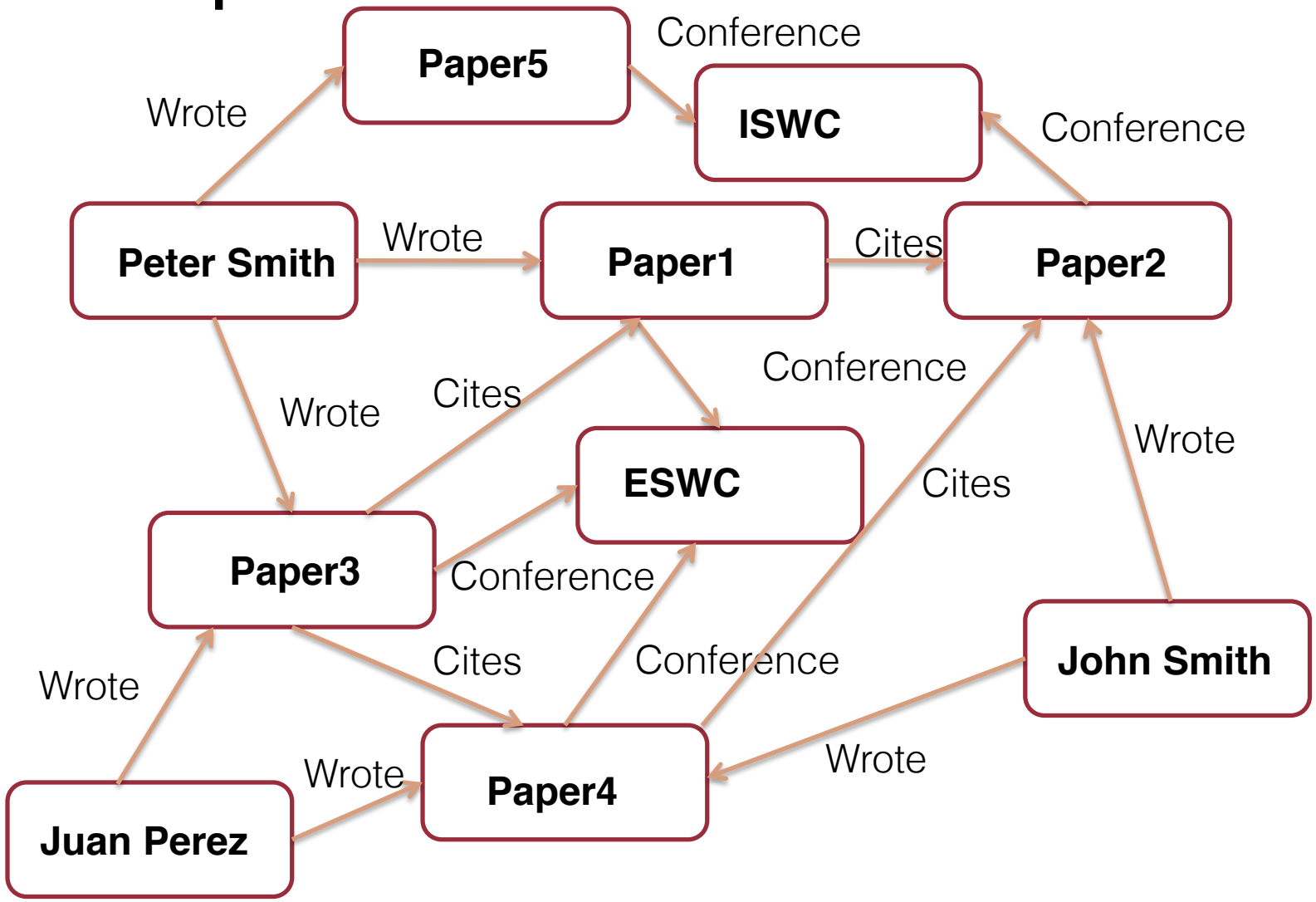


# RDF3x (6)

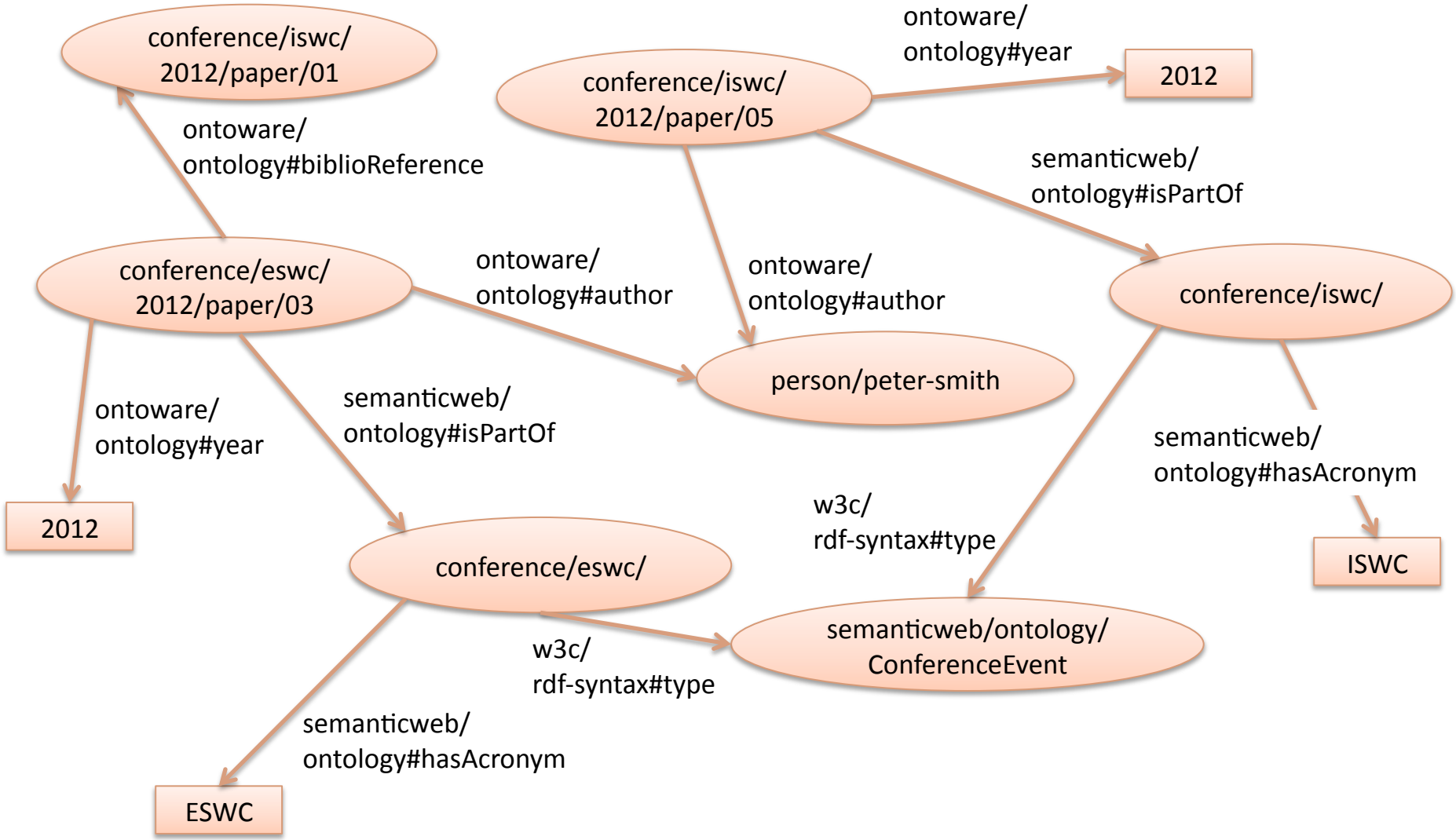



```
MergeJoin
 IndexScan PredicateSubjectObject
   ?A1 yago:hasFamilyName ?fn1.
 MergeJoin
    IndexScan PredicateSubjectObject
    ?A1 yago:hasGivenName ?gn1 .
    MergeJoin
       IndexScan PredicateObjectSubject
       ?person1 yago:influences ?A1.
       IndexScan ObjectPredicateSubject
       ?A1 rdf:type yago:wordnet_artist_109812338.
```

# Example


Paper5
Conference
ISWC
Conference
Wrote
Peter Smith
Wrote
Paper1
Cites
Paper2
Wrote
Cites
Conference
ESWC
Wrote
Cites
Paper3
Conference
John Smith
Wrote
Cites
Conference
Wrote
Paper4
Wrote
Juan Perez

# RDF Subgraph for the Publication Graph



# RDF3x: Representation of the Publication Graph

RDF3x Mapping Dictionary

| OID | Resource |
|---|---|
| 0 | http://www.w3.org/1999/02/22-rdf-syntax-ns#type |
| 1 | http://data.semanticweb.org/ns/swc/ontology#hasAcronym |
| 2 | http://data.semanticweb.org/ns/swc/ontology#isPartOf |
| 3 | http://swrc.ontoware.org/ontology#author |
| 4 | http://swrc.ontoware.org/ontology#year |
| 5 | http://swrc.ontoware.org/ontology#biblioReference |
| 6 | http://data.semanticweb.org/conference/iswc/2012 |
| 7 | http://data.semanticweb.org/ns/swc/ontology#ConferenceEvent |
| 8 | ISWC2012 |
| 9 | http://data.semanticweb.org/conference/eswc/2012 |
| 10 | ESWC2012 |
| 11 | http://data.semanticweb.org/conference/iswc/2012/paper/05 |
| 12 | http://data.semanticweb.org/conference/iswc/2012/paper/02 |
| 13 | http://data.semanticweb.org/conference/eswc/2012/paper/01 |
| 14 | http://data.semanticweb.org/conference/eswc/2012/paper/03 |
| 15 | http://data.semanticweb.org/conference/eswc/2012/paper/04 |
| 16 | http://data.semanticweb.org/author/peter-smith |
| 17 | http://data.semanticweb.org/author/juan-perez |
| 18 | http://data.semanticweb.org/author/john-smith |
| 19 | 2012 |

Triple Table

| S | P | O |
|---|---|---|
| 6 | 0 | 7 |
| 9 | 0 | 7 |
| 6 | 0 | 7 |
| 9 | 1 | 8 |
| 11 | 1 | 10 |
| 12 | 2 | 6 |
| 13 | 2 | 6 |
| 14 | 2 | 9 |
| 15 | 2 | 9 |
| 11 | 3 | 16 |
| 12 | 3 | 18 |
| 13 | 3 | 16 |
| 14 | 3 | 16 |
| 14 | 3 | 17 |
| 15 | 3 | 17 |
| 15 | 3 | 18 |
| 11 | 4 | 19 |
| 12 | 4 | 19 |
| 13 | 4 | 19 |
| 14 | 4 | 19 |
| 15 | 4 | 19 |
| 13 | 5 | 12 |
| 14 | 5 | 13 |
| 14 | 5 | 15 |
| 15 | 5 | 12 |

# Graphium (http://graphium.ldc.usb.ve)

[ Flores et al. 2013, De Abreu et al. 2013]

- Abstract **Graph Database Engine** built on top of: Neo4j, Sparksee, and HyperGraphDB.
- Java **Graph-based** and **RDF-based** API.
- Labeled attribute **multigraph**.
- Nodes and edges can have **properties**.
- There are **no restrictions** on the number of edges between two nodes.
- **Loops** are allowed.
- Attributes are **single valued**.
- Different types of **traversal strategies**.
- Different **data mining strategies**.

# Graphium *RDF-based* API (1)

## Creating a graph

Just use the command-line tool to load your RDF graph from an NT file, and create the database using Neo4j or Sparksee.

```
./create <GDBM (Sparksee or Neo4j)> <NT file> <DB location>
```

For example…

```
> ./create Neo4j publications.nt pubDB-neo4j/
> ./create Sparksee publications.nt pubDB-sparksee/
```

# Graphium *RDF-based* API (2)

## Loading the graph in Java

```
GraphRDF g;

g = new Neo4jRDF( “pubDB-neo4j/” );

// or…

g = new SparkseeRDF( “pubDB-sparksee/” );
```

# Graphium *RDF-based* API (3)

## Searching for nodes…

```
Vertex v;
```

If the node you are searching for is an URI, use:

```
v = g.getVertexURI("http://data.semanticweb.org/author/peter-smith");
```

If it is a Blank Node, use:

```
v = g.getVertexBlankNode("blanknodeid");
```

# Graphium *RDF-based* API (4)

From each vertex you can…

(1) Get the RDF object it represents:

```
// If you don't know what object it represents,
// use the superclass RDFobject…
RDFobject v_any = v.getAny();

// If you know, use any of the subclasses…
URI v_uri      = v.getURI();
BlankNode v_bn = v.getBlankNode();
Literal v_lit  = v.getLiteral();
```

# Graphium *RDF-based* API (5)

From each vertex you can…

(2) Get its in/out relationships:

```
GraphIterator<Edge> it;

// For out-going relationships…
it = v.getEdgesOut();

// For in-coming relationships…
it = v.getEdgesIn();

while (it.hasNext()) {
    Edge rel = it.next();
    …
}
it.close();
```

# Graphium *RDF-based* API (6)

From each edge you can…

(1) Get its *predicate*/URI:

```
URI predicate = rel.getURI();
```

(2) Get the *start*/*end* of the edge (the *subject* and *object* of each triple):

```
Vertex start_v, end_v;

start_v = rel.getStart();
end_v   = rel.getEnd();
```

# Graphium *RDF-based* API (7)

From these building blocks you can start traversing the graph…

"*2-Hops from Peter Smith*"

```
Vertex v;
GraphIterator<Edge> it1, it2;

v = g.getVertexURI("http://data.semanticweb.org/author/peter-smith");
it1 = v.getEdgesOut();
while (it1.hasNext()) {
    it2 = it1.next().getEnd().getEdgesOut();
    while (it2.hasNext()) {
        … it2.next() …
    }
    it2.close();
}
it1.close();
```

# 5 EMPIRICAL EVALUATION

# Experimental Set-Up

Machine:

- Model: Sun Fire x4100
- Quantity of processors: 2
- Processor specification: Dual-Core AMD Opteron™ Processor 2218 64 bits
- RAM memory: 16GB
- Disk capacity: 2 disks, 135GB e/o
- Network interface: 2 + ALOM
- Operating System: CentOS 5.5. 64 bits

Engines:

- RDF3x version: 0.3.7
- Sparksee version: 4.8 Very Large Databases
- HyperGraphDB version: 1.2
- Neo4J version: 1.9

Timeout: 3,600 secs.

Implementation:

We provide both **internal** implementation for traversal methods using the API's of each engine, and the **external** implementation using the generic algorithm through the core-graph API methods.

Runs: Up to 10 times for each test. Average is reported. All experiments were run in cold cache.

The graphs were loaded in *sif* format for the general purpose graph engines and in *nt* format for RDF-3x.

Only RDF-3x could load graphs from Berlin Benchmark of over 50 million edges.

Download the code and datasets at ***https://github.com/gpalma/gdbb/***

# Benchmark Graphs

| Graph Name | #Nodes | #Edges | Description |
|---|---|---|---|
| DSJC1000.1 | 1,000 | 49,629 | Graph Density:0.1 [Johnson91] |
| DSJC1000.5 | 1,000 | 249,826 | Graph Density:0.5 [Johnson91] |
| DSJC1000.9 | 1,000 | 449,449 | Graph Density:0.9 [Johnson91] |
| SSCA2-17 | 131,072 | 3,907,909 | GTgraph 1) |

[Johnson91] Johnson, D., Aragon, C., McGeoch, L., and Schevon, C. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. Operations research 39, 3 (1991), 378–406.

GTgraph: A suite of three synthetic graph generators:

1) SSCA2: generates graphs used as input instances for the DARPA. High Productivity Computing Systems SSCA#2 graph theory benchmark
   http://www.cse.psu.edu/~madduri/software/GTgraph/index.html



# Benchmark Tasks

The benchmark tasks evaluate the engine performance while executing the following graph operations:

| Task | Description |
|---|---|
| adjacentXP | Given a node X and an edge label P, find adjacent nodes Y. |
| adjacentX | Given a node X, find adjacent nodes Y. |
| edgeBetween | Given two nodes X and Y, find the labels of the edges between X and Y. |
| 2-hopX | Given a node X, find the 2-hop Y nodes. |
| 3-hopX | Given a node X, find the 3-hop Y nodes. |
| 4-hopX | Given a node X, find the 4-hop Y nodes. |

# Benchmark Tasks: RDF Engines

The benchmark tasks were implemented in the RDF engines with the following SPARQL queries:

| Task | SPARQL Query |
|---|---|
| adjacentXP | `Select ?y`<br>`where {`<br>`<http://graphdatabase.ldc.usb.ve/resource/6> <http://graphdatabase.ldc.usb.ve/resource/pr> ?y}` |
| adjacentX | `Select ?y`<br>`where {`<br>`<http://graphdatabase.ldc.usb.ve/resource/6> ?p ?y}` |
| edgeBetween | `Select ?p`<br>`where {`<br>`<http://graphdatabase.ldc.usb.ve/resource/6> ?p <http://graphdatabase.ldc.usb.ve/resource/8>` |
| 2-hopX | `Select ?z`<br>`where {`<br>`<http://graphdatabase.ldc.usb.ve/resource/6> ?p1 ?y1.`<br>`?y1 ?p2 ?z}` |
| 3-hopX | `Select ?z`<br>`where {`<br>`<http://graphdatabase.ldc.usb.ve/resource/6> ?p1 ?y1.`<br>`?y1 ?p2 ?y2.`<br>`?y2 ?p3 ?z}` |
| 4-hopX | `Select ?z`<br>`where {`<br>`<http://graphdatabase.ldc.usb.ve/resource/6> ?p1 ?y1.`<br>`?y1 ?p2 ?y2.`<br>`?y2 ?p3 ?y3.`<br>`?y3 ?p4 ?z}` |

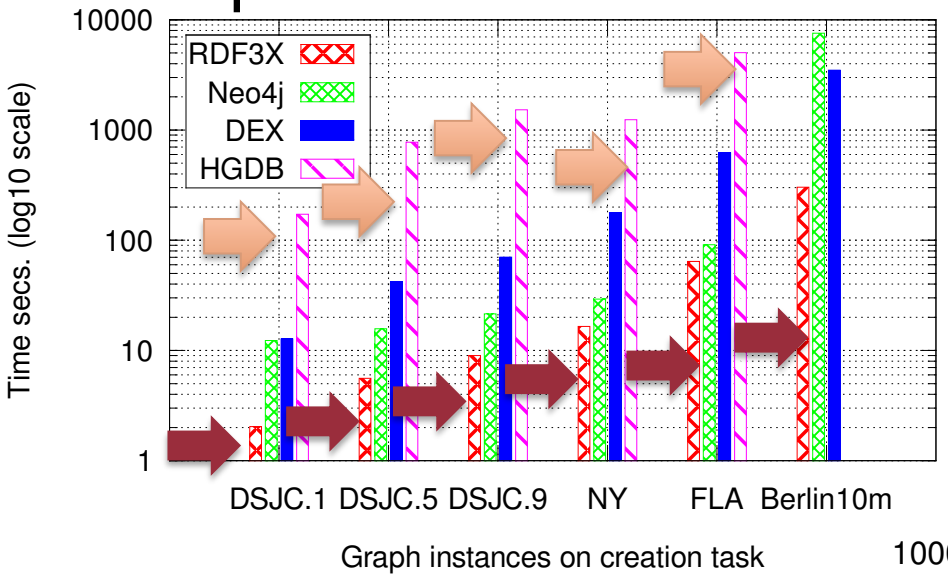
# Benchmark of Graph

| Graph Name | #Nodes | #Edges | Density | #Labels | |
|---|---|---|---|---|---|
| DSJC1000.1 [Johnson91] | 1,000 | 99,258 | 0.099 | 1 | ✔ |
| DSJC1000.5 [Johnson91] | 1,000 | 499,652 | 0.50 | 1 | ✔ |
| DSJC1000.9 [Johnson91] | 1,000 | 898,898 | 0.899 | 1 | ✔ |
| USA-road-d.NY | 264,346 | 730,100 | 0.00001045 | 7,970 | ✔ |
| USA-road-d.FLA | 1,070,376 | 2,687,902 | 0.00000235 | 22,704 | ✔ |
| Berlin10M | 2,743,235 | 9,709,119 | 0.00000129 | 40 | ✔ |

Medium-Size Graphs

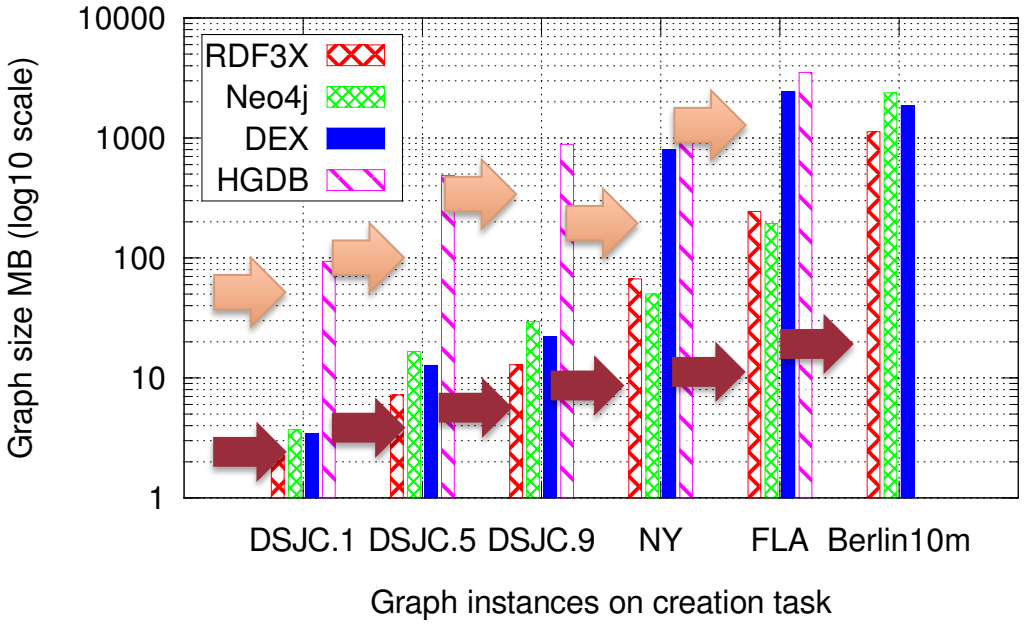
[Johnson91] Johnson, D., Aragon, C., McGeoch, L., and Schevon, C. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. Operations research 39, 3 (1991), 378–406.
USA-road-d* Graphs 9th DIMACS Implementation Challenge - Shortest Paths http://www.dis.uniroma1.it/challenge9/download.shtml
Berlin10M: Berlin Bechmark-http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

# Graph Creation-Time and Memory

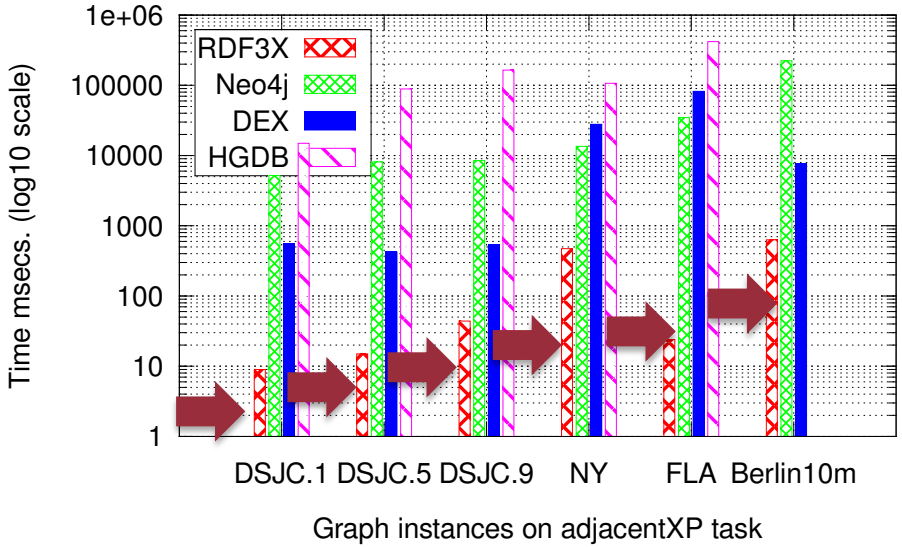


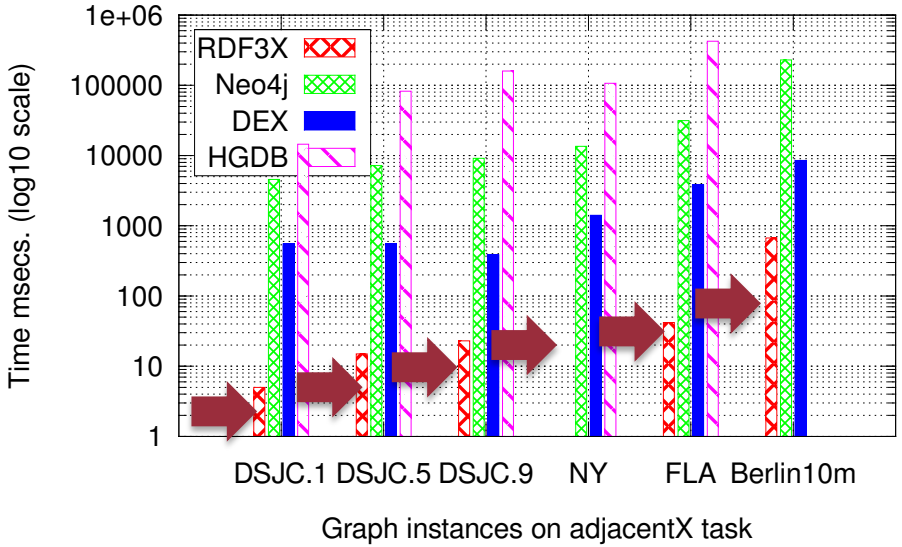

- HyperGraphDB timed out after 24 hours (Berlin10M).
- RDF-3x is faster than the rest of the engines.

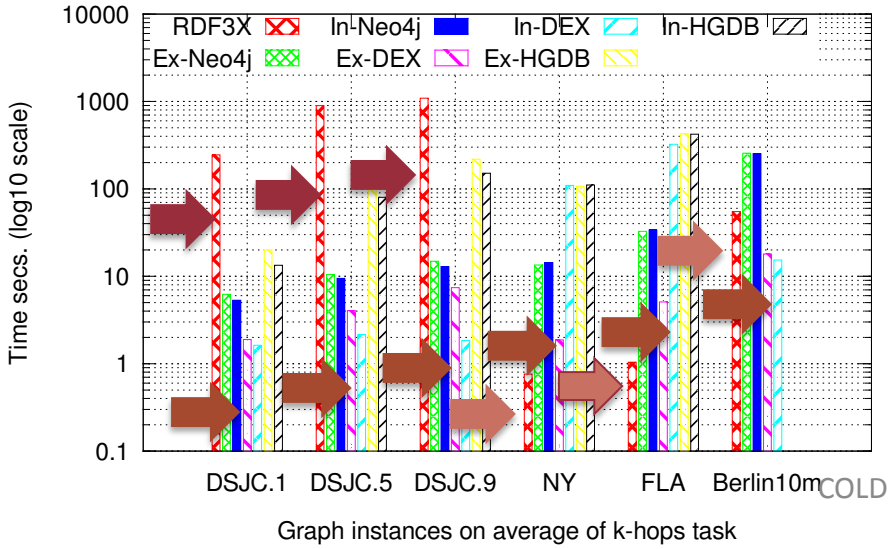
- Secondary Memory grows as the size of the graph.
- In general, RDF-3x requires less memory than the rest of the engines.



- RDF-3x exploits internal structures and overcomes the rest of the graph engines.
- Neo4j and Sparksee are competitive.
- HyperGraphDB always consumes at least one order of magnitude more time than the rest of the engines.

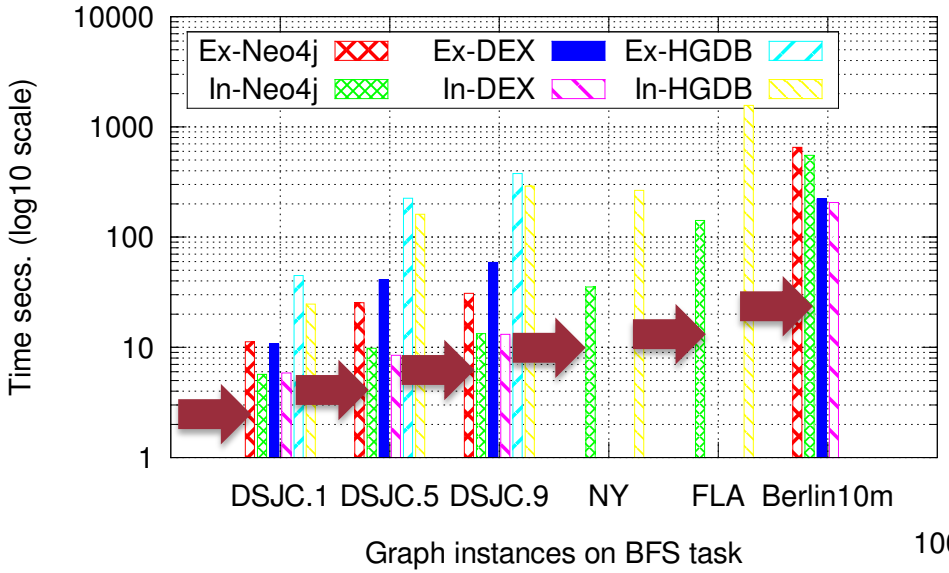


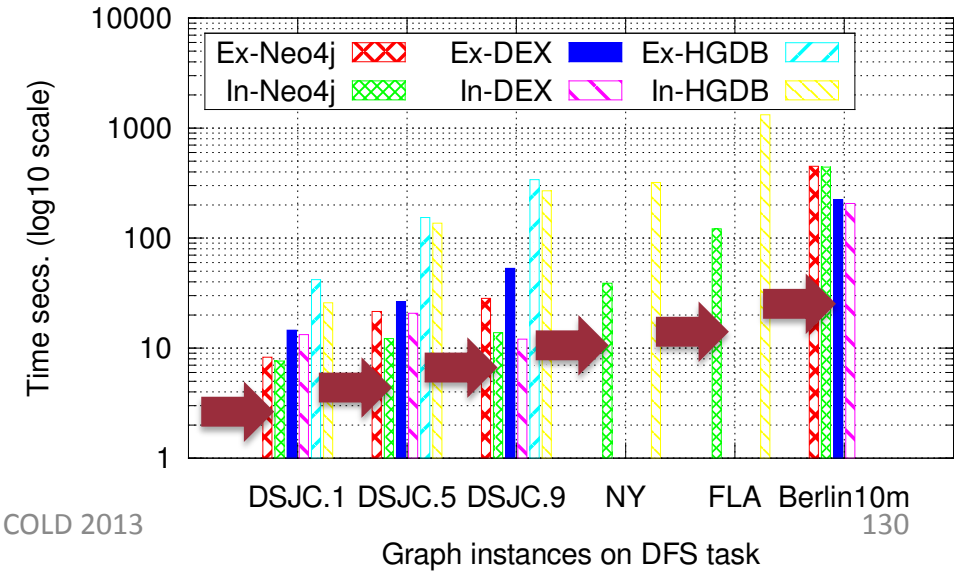




- RFD-3x is impacted by the graph density; it timed out for 4-hops in DSJC.5 and DSJC.9.
- RDF-3x exploits internal structures in sparse graphs.
- Sparksee outperforms all the engines in dense graph with few label; internal and external implementations are competitive.
- Number of label negatively impacts Sparksee.

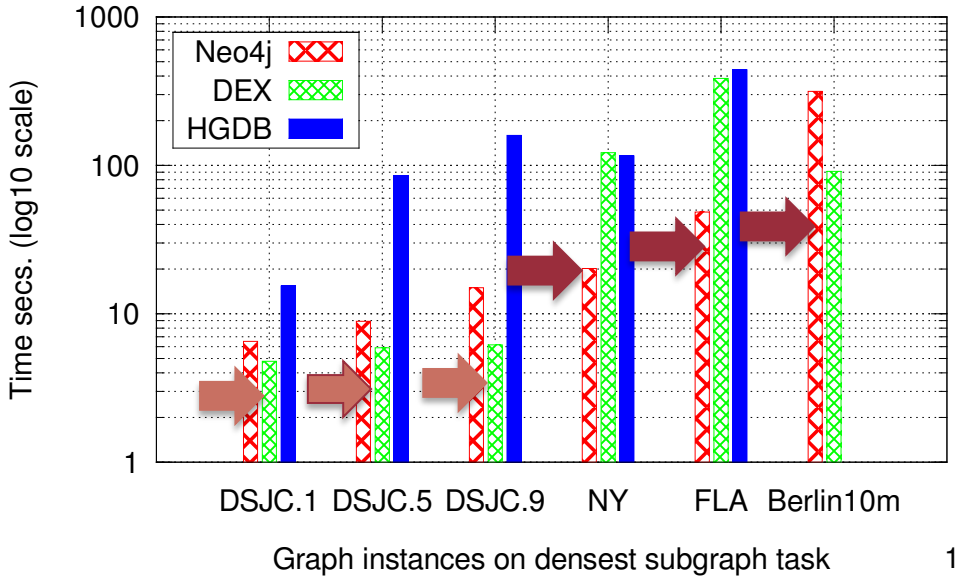
# Traversals-BFS and DFS (Internal and External)




- Internal representations outperform external representations.
- Neo4j exploits internal representation of graph, neighborhood indices, and main memory-structures and overcomes the rest of the engines.

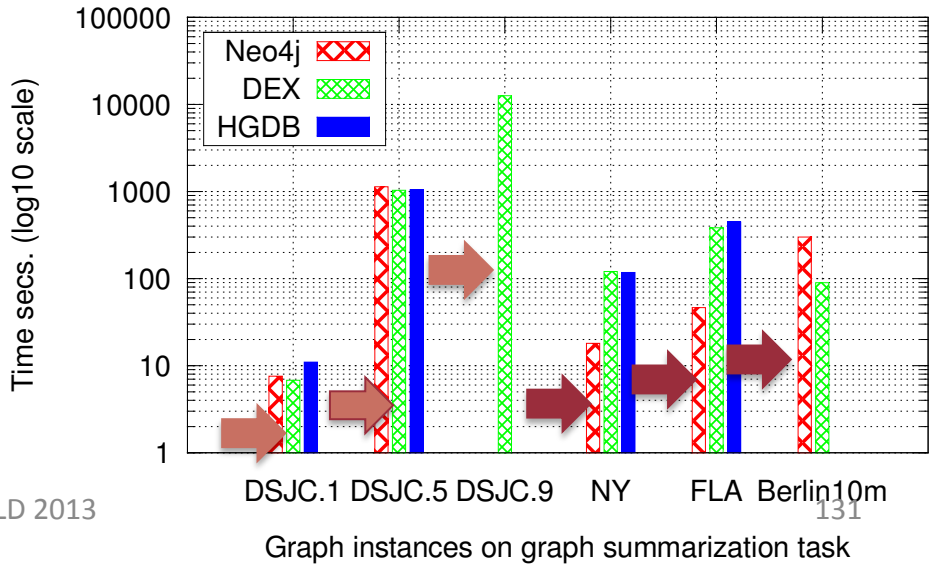


# Mining Tasks



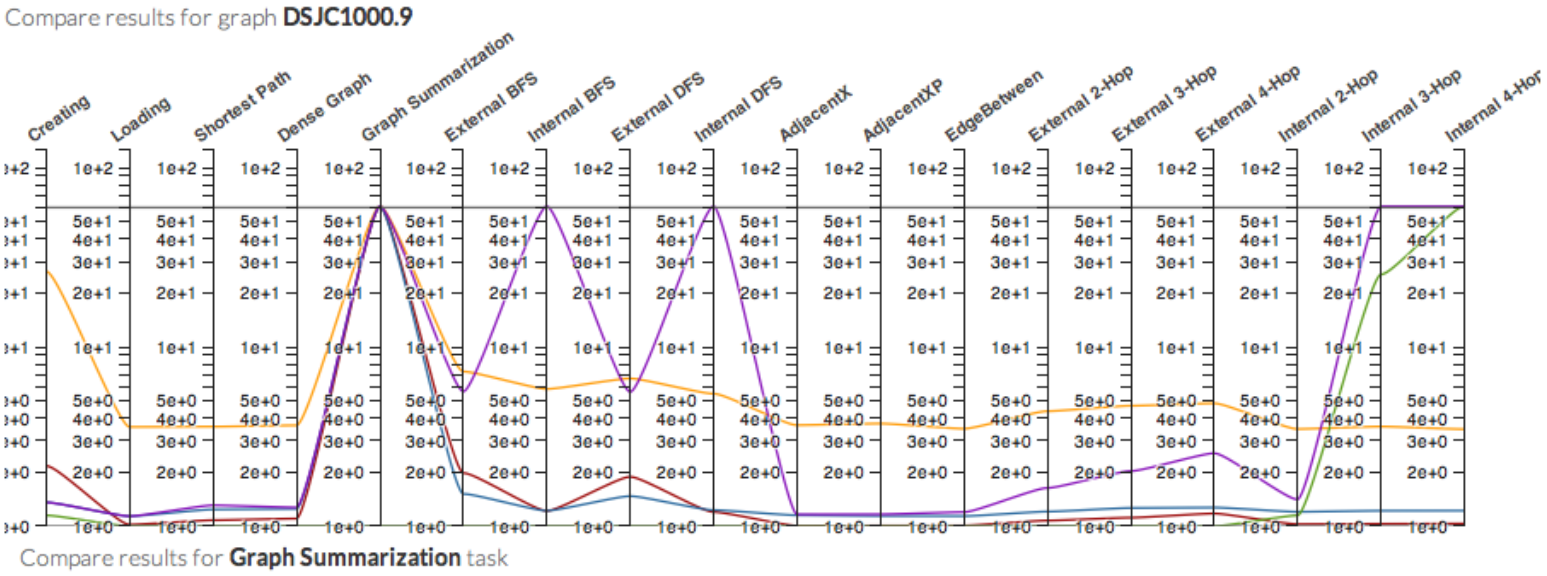

- Sparksee overcomes the rest of engines when the graphs are dense.
- Neo4j overcomes the rest of the engines in sparse graphs.

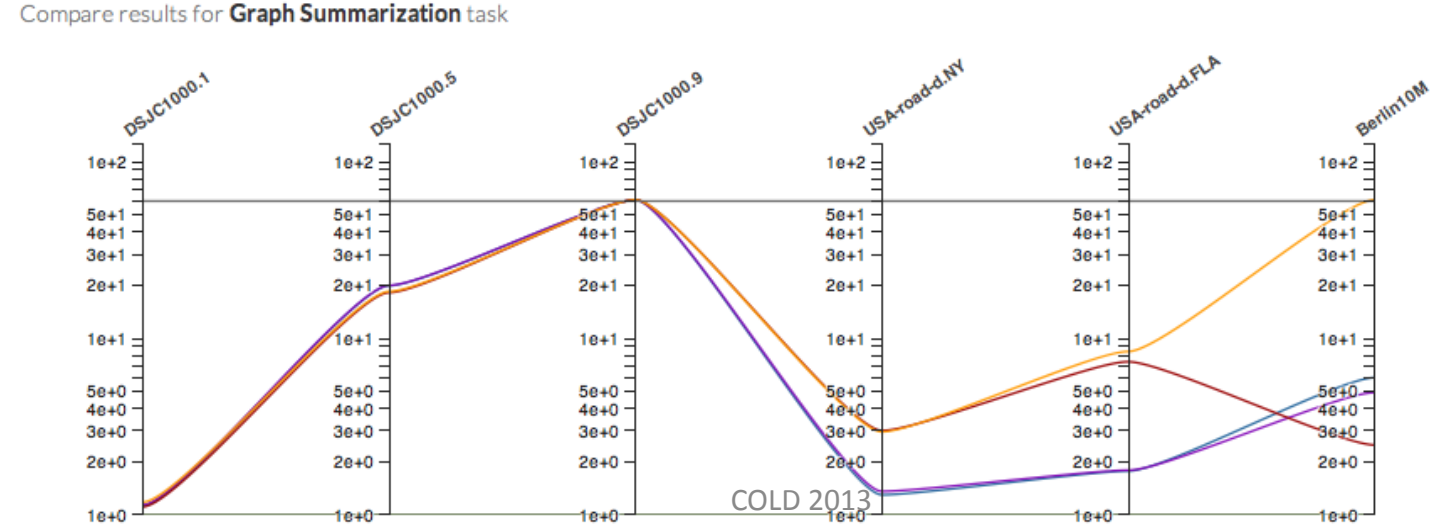
- Graph Summarization seems to be more affected by graph density, size of the graph and number of labels.
- Sparksee overcomes the rest of engines when the graphs are dense.
- Neo4j overcomes the rest of the engines in sparse graphs.

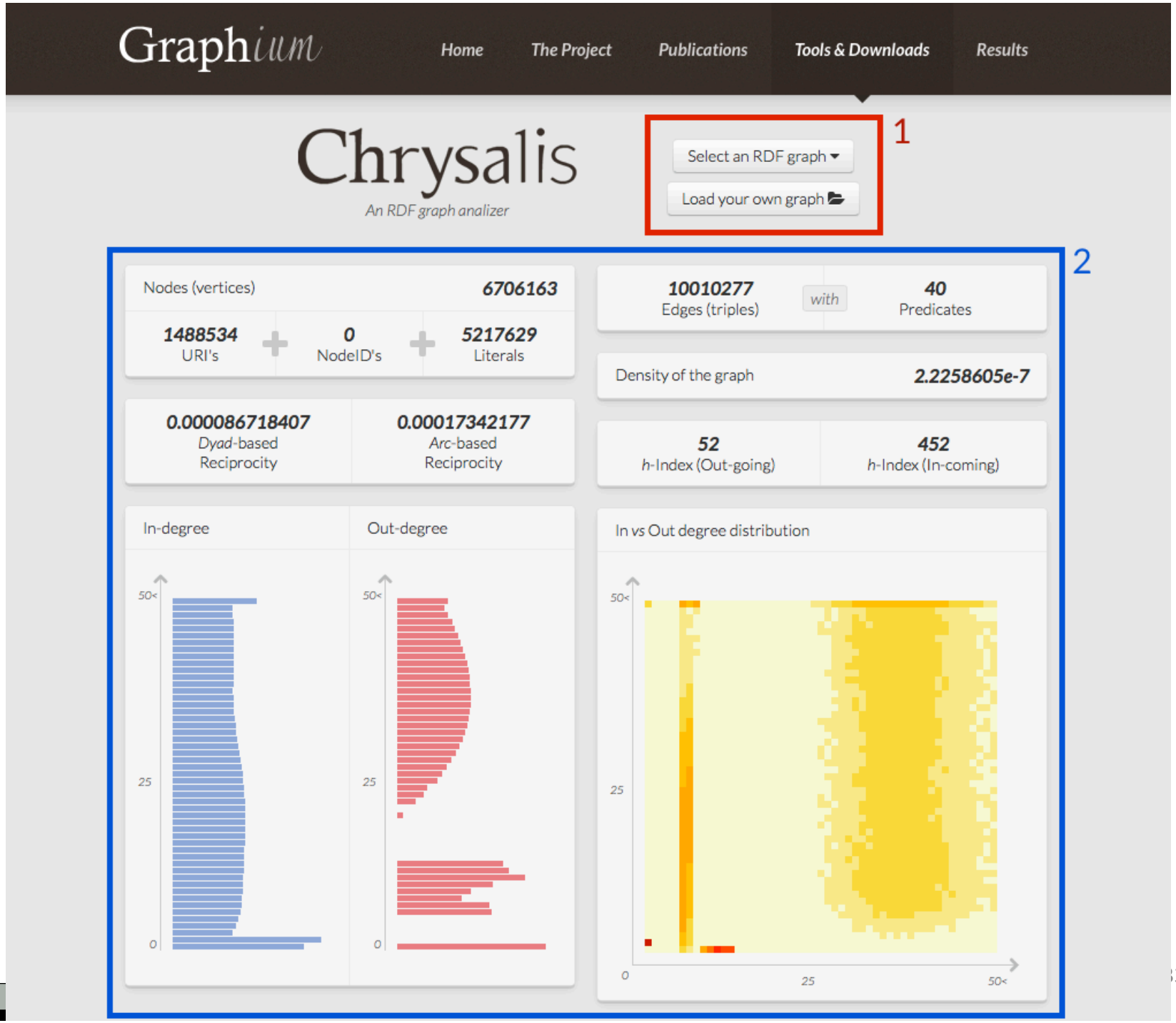


Graphium
GRAPH DATABASE
BENCHMARK

http://graphium.ldc.usb.ve/




Graphium
Home
The Project
Publications
Tools & Downloads
Results
Chrysalis
An RDF graph analizer
Select an RDF graph
Load your own graph
1
2
Nodes (vertices)
6706163
1488534
URI's
0
NodeID's
5217629
Literals
10010277
Edges (triples)
with
40
Predicates
Density of the graph
2.2258605e-7
0.000086718407
Dyad-based
Reciprocity
0.00017342177
Arc-based
Reciprocity
52
h-Index (Out-going)
452
h-Index (In-coming)
In-degree
Out-degree
In vs Out degree distribution
50<
25
0
50<
25
0
50<
25
0
25
50<

# COFFEE BREAK

# 6 HANDS-ON SESSION

# Hands-on Goals

- Implement simple graph-based queries by using Neo4j and Sparksee API.
- Implement simple RDF-based queries by using Graphium API.
- Implement graph invariants by using Graphium API.

# Assignments:

1) Implement the following query: "*Papers written by Peter Smith*". Use the Sparksee API[1] and Neo4j[2] API.

[1] http://sparsity-technologies.com/downloads/javadoc-java/index.html

[2] http://api.neo4j.org/2.1.0-M01/

# Assignments

2) Implement the following queries using the Graphium API[3]:

a) "*Papers written by Peter Smith*".

b) "*Papers cited by a paper written by Peter Smith that have at most 20 cites.*".

c) "*Papers cited by a paper written by Peter Smith or cited by papers cited by a paper written by Peter Smith.*"

d) "Number of papers cited by a paper written by *Peter Smith or cited by papers cited by a paper written by Peter Smith. "*

e) "Number of papers cited by a paper written by *Peter Smith or cited by papers cited by a paper written by Peter Smith, and have been published in ESWC."*

f) "*Number of papers cited by a paper written by Peter Smith or cited by papers cited by a paper written by Peter Smith, have been published in ESWC and have at most 4 co-authors ".*

[3] http://graphium.ldc.usb.ve

# Assignments

## 3) Implement graph invariants using the Graphium API[3]

| Invariant | Description |
|---|---|
| Vertex and Edge Count | number of vertices and edges in the graph. |
| Graph Density | number of edges in the graph divided by the number of possible edges in a complete digraph. |
| Reciprocity | Reciprocity measures the extend to which a triple that relates resources A and B is reciprocated by a another triple that relates B with A too. |
| In- and Out-degree Distribution | Distribution of the number of in-coming and out-going edges of the vertices of a graph. |
| In-coming and Out-going h-index | h is the maximum number, such that h vertices have each at least h in-coming neighbors (resp., out-going neighbors) in the graph. |

[3] http://graphium.ldc.usb.ve

# Assignments

4) Evaluate Berlin 50K in Graphium Chrisalys[4] and upload your results.

[4] http://graphium.ldc.usb.ve/chrysalis/

# 6 QUESTIONS & DISCUSSION

# Graph Data Storing Features

| Graph Database | Main Memory | Persistent Memory | Backend Storage/ Implementation | Indexing |
|---|---|---|---|---|
| Sparksee | X | X | | X |
| HyperGraphDB[1] | X | X | X | X |
| Neo4j | X | X | | X |
| Hexastore[2] | X | X | X | X |
| RDF3x | | X | | X |
| Graphium[3] Graph-based | X | X | X | X |
| Graphium[3] RDF-based | X | X | X | X |

[1]Backend Storage: Berkeley DB

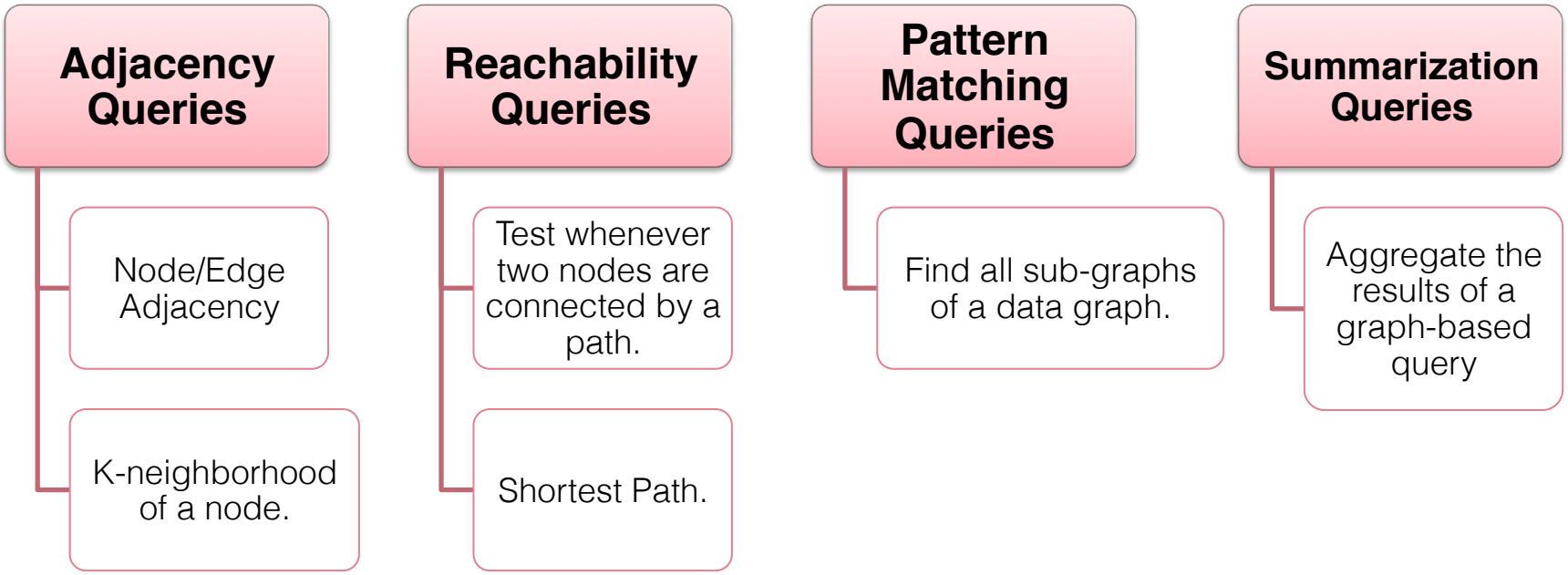
[2]Backend Storage: Tokyo Cabinet

[3]Backend Implementation: Neo4j and Sparksee

# Graph Data Management Features

| Graph Database | Graph-based API/ RDF-based API | | | Query Language | Data Visualization |
|---|---|---|---|---|---|
| | Data Definition | Data Manipulation | Standard Compliance | | |
| Sparksee | X | X | X | | X |
| HyperGraphDB | X | X | X | | |
| Neo4j | X | X | X | X | X |
| Hexastore | X | X | X | X | |
| RDF3x | | | | X | |
| Graphium Graph-based | X | X | X | | |
| Graphium RDF-based | | X | X | | X |

# Graph-Based Operations



# Graph-Based Operations

| Graph Database | Node/Edge adjacency | K-neighborhood | Fixed-length paths | Regular Simple Paths | Shortest Path | Pattern Matching | Data Mining Tasks |
|---|---|---|---|---|---|---|---|
| Sparksee | X | X | X | X | X | X | |
| HyperGraphDB | X | X | X | | X | X | |
| Neo4j | X | X | X | X | X | X | |
| Hexastore | X | | | | | X | |
| RDF3x | X | | | | | X | |
| Graphium Graph-based | X | X | X | X | X | X | X |
| Graphium RDF-based | X | X | X | | | X | |

# 7 SUMMARY & CLOSING

# Conclusions (1)

## Data Storing Features:

- Engines implement a **wide variety of structures** to efficiently represent large graphs.
- **RDF engines** implement **special-purpose structures** to efficiently store RDF graphs.

# Conclusions (2)

## Data Management Features:

- General-purpose graph database engines provide APIs to manage and query data.
- RDF engines support general data management operations.

# Conclusions (3)

## Graph-Based Operations Support:

- General-purpose graph database engines provide **API methods** to implement a wide variety of graph-based operations.
- Only few offer **query languages** to express pattern matching.
- RDF engines are **not customized** for basic graph-based operations, however,
  - They efficiently implement the pattern matching-based language SPARQL.

# Future Directions

- **Extension** of existing **engines** to support specific tasks of graph traversal and mining.

- Definition of **benchmarks** to study the performance and quality of existing graph database engines, e.g.,
  - Graph traversal, link prediction, pattern discovery, graph mining.

- **Empirical evaluation** of the performance and quality of existing graph database engines

# References


- P. Anderson, A. Thor, J. Benik, L. Raschid, and M.-E. Vidal. Pang: finding patterns in anno- tation graphs. In SIGMOD Conference, pages 677–680, 2012.
- R. Angles and C. Gutiérrez. Survey of graph database models. ACM Comput. Surv., 40(1), 2008.
- M. Atre, V. Chaoji, M. Zaki, and J. Hendler. Matrix Bit loaded: a scalable lightweight join query processor for RDF data. In International Conference on World Wide Web, Raleigh, USA, 2010. ACM.
- D. De Abreu, A. Flores, G. Palma, V. Pestana, J. Piñero, J. Queipo, J. Sánchez, M.E. Vidal. Choosing Between Graph Databases and RDF Engines for Consuming and Mining Linked Data. COLD 2013 in Conjunction with ISWC 2013.
- A.Flores, G. Palma, M.E. Vidal, D. De Abreu, V. Pestana, J. Piñero, J. Queipo, J. Sánchez, GRAPHIUM: Visualizing Performance of Graph and RDF Engines on Linked Data. Poster & Demo ISWC 2013.
- A. Flores, G. Palma, M.E. Vidal, D. De Abreu, V. Pestana, J. Piñero, J. Queipo, J. Sánchez, Graphium Chrysalis: Exploiting Graph Database Engines to Analyze RDF Graphs. Demo ESWC 2014.

# References


- B.Iordanov.Hypergraphdb:Ageneralizedgraphdatabase.InWAIMWorkshops,pages25–36, 2010.

- N. Martíınez-Bazan, V. Muntés-Mulero, S. G ómez-Villamor, J. Nin, M.-A. Sá nchez-Martíınez, and J.-L. Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In CIKM, pages 573–582, 2007.

- T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. Proc. VLDB, 1(1), 2008.

- M. K. Nguyen, C. Basca, and A. Bernstein. B+hash tree: optimizing query execution times for on-disk semantic web data structures. In Proceedings Of The 6th International Workshop On Scalable Semantic Web Knowledge Base Systems (SSWS2010), 2010.

- I Robinson, J. Webber, and E. Eifrem. Graph Databases. O'Reilly Media, Incorporated, 2013.

# References


- S. Sakr and E. Pardede, editors. Graph Data Management: Techniques and Applications. IGI Global, 2011.

- D.Shimpiand S.Chaudhari. An overview of graph databases .In IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science 2012, 2013.

- M.-E. Vidal, A. Martínez, E. Ruckhaus, T. Lampo, and J. Sierra. On the efficiency of querying and storing rdf documents. In Graph Data Management, pages 354–385. 2011.

- C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple indexing for semantic web data management. Proc. of the 34th Intl Conf. on Very Large Data Bases (VLDB), 2008.

- C Weiss, A Bernstein, On-disk storage techniques for Semantic Web data - Are B-Trees always the optimal solution?, Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems, October 2009.