Introduction to Graph Databases

Fundamentals & Implementations

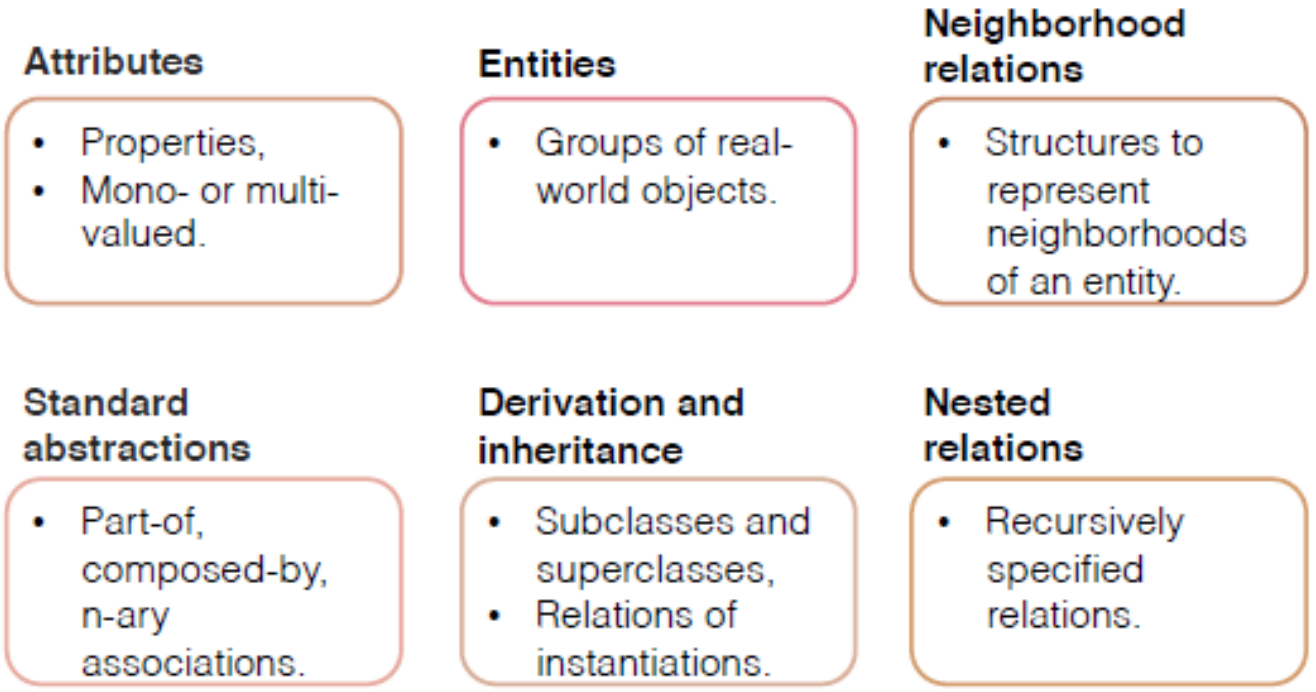
Alejandro Vaisman
avaisman@itba.edu.ar

Agenda

- 10.10.22. Introduction – Graph data models
- 13.10.22. **Graph DB internals. Introduction to Neo4j**
- 17.10.22. Querying Neo4j databases
- 20.10.22. Assignment 1. Graphs in relational databases
- 24.10.22. Assignment 2. Basic Cypher queries
- 27.10.22. Assignment 3. Advanced Cypher queries

Graph database models

- Types of relationships supported by graph data models



The abstract data type Graph (w/properties)

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, …}

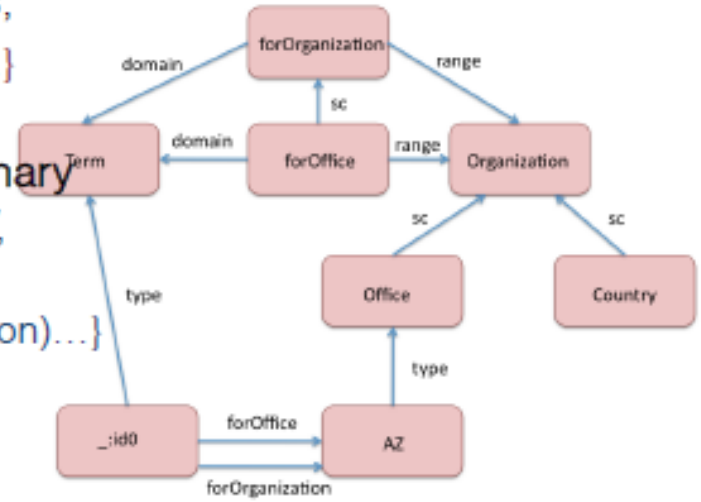
- L is a function: V x V ➔ Σ ,

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

The abstract data type Multigraph

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

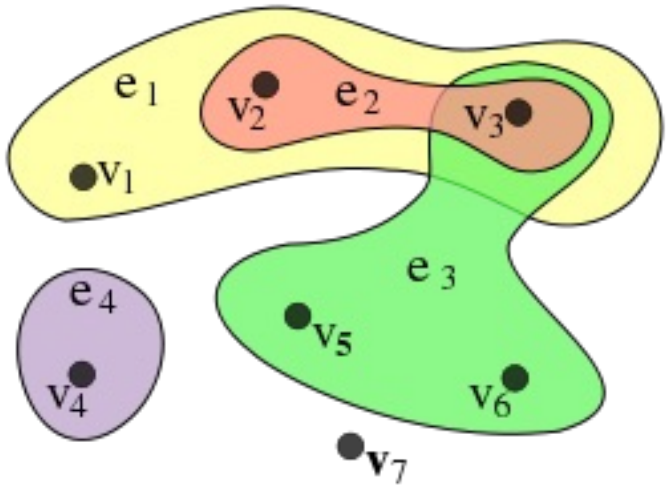
Given a graph G, the following are operations over G:

- 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 an edge e 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.

Graph generalization: (multi)Hypergraphs

H = (X , E), where X is a set of *nodes*, and E is a set of non-empty subsets of X called hyperedges **=>**
E ⊆ P (X), where P (X) is the power set of X.

Undirected



$$X = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$

$$E = \{e_1, e_2, e_3, e_4\} = \{\{v_1, v_2, v_3\}, \{v2, v3\}, \{v3, v5, v6\}, \quad \{v_4\}\}$$

Let X = (v*1*,…,v*n*), E = (*e1*,…,e*m*).
Every hypergraph has an *m* x *n* incidence matrix $A = (a_{ij})$ where

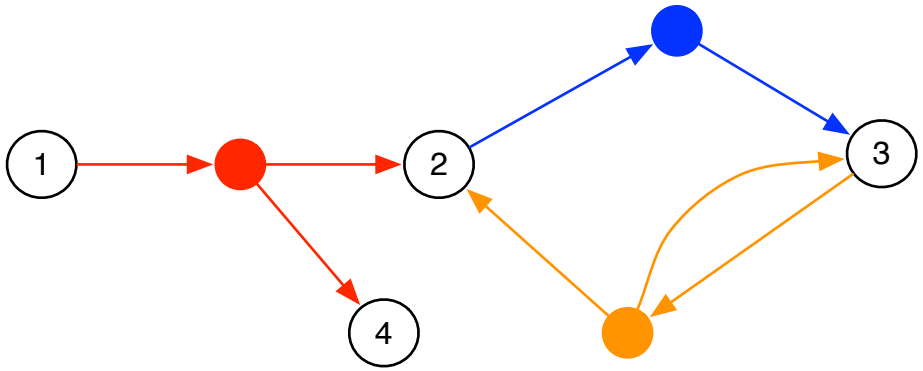
$$a_{ij} = \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{otherwise.} \end{cases}$$

e1	1	1	1	0	0	0	0
e2	0	1	1	0	0	0	0
e3	0	0	1	0	1	1	0
e4	0	0	0	1	0	0	0
	V1	v2	v3	v4	v5	v6	v7

Graph generalization: (multi)Hypergraphs

H = (X , E), where X is a set of *nodes*, and E is a set of non-empty subsets of X called hyperedges **=>**
E is a subbag of P (X) x P(X), where P (X) is the power set of X.

Directed

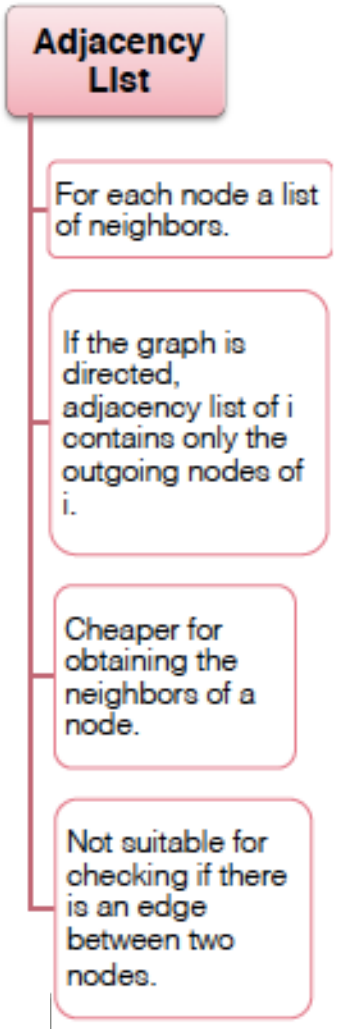


Graphically, S,T ⊆ X; A hyperedge is denoted S -> T

In the example:

X = {1,2,3,4}
E = {{1}->{2,4}, {2} -> {3}, {3} -> {2,3}}

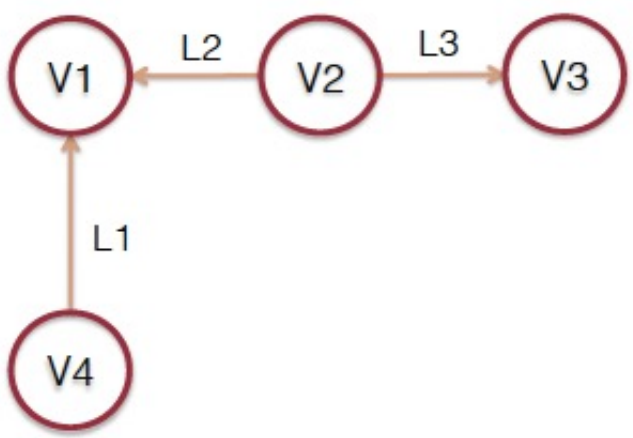
Implementation

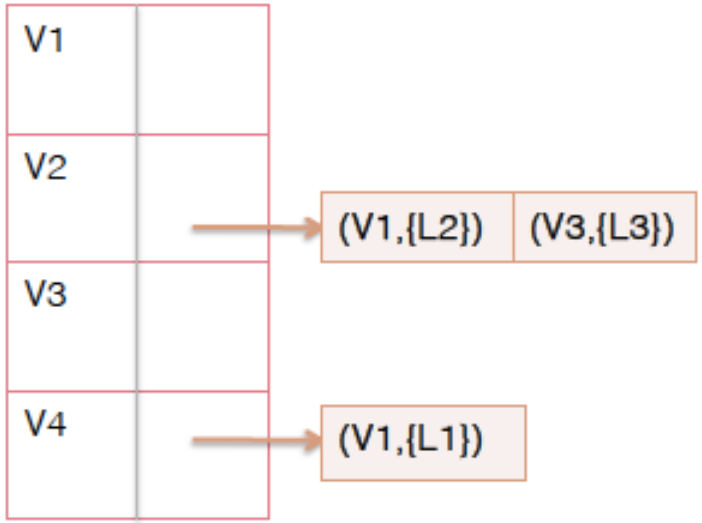


Implementation: adjacency list

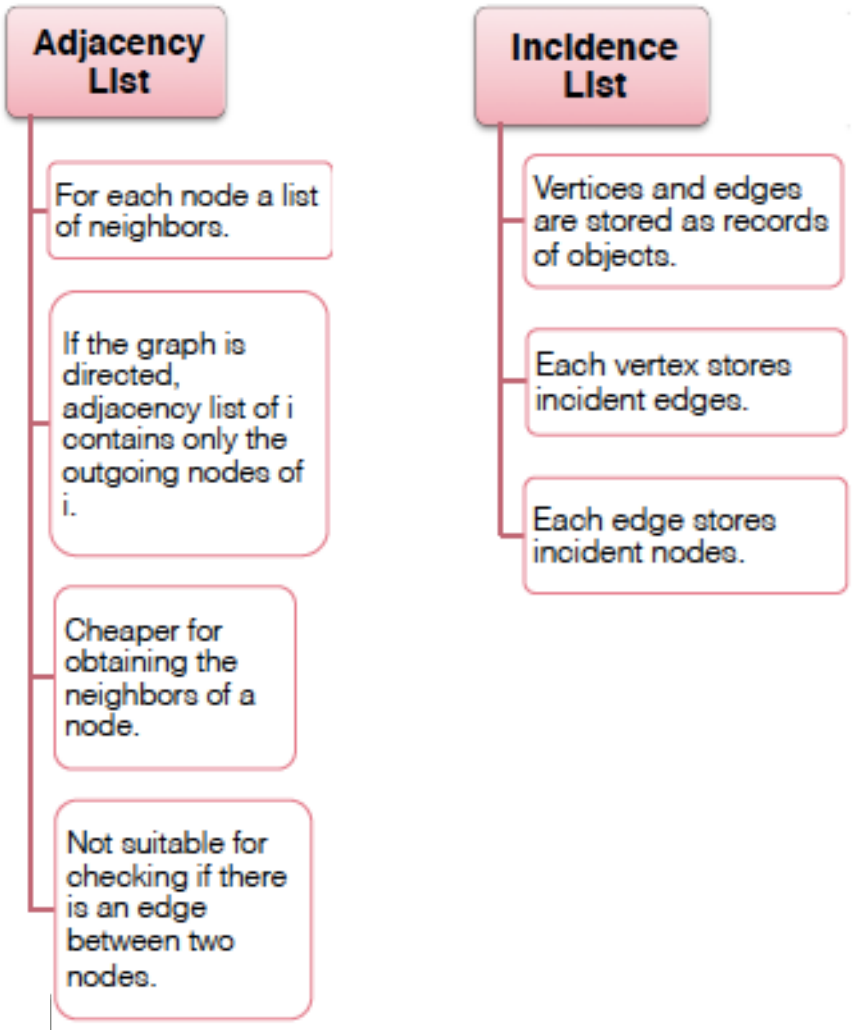
Adjacency List

- 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.





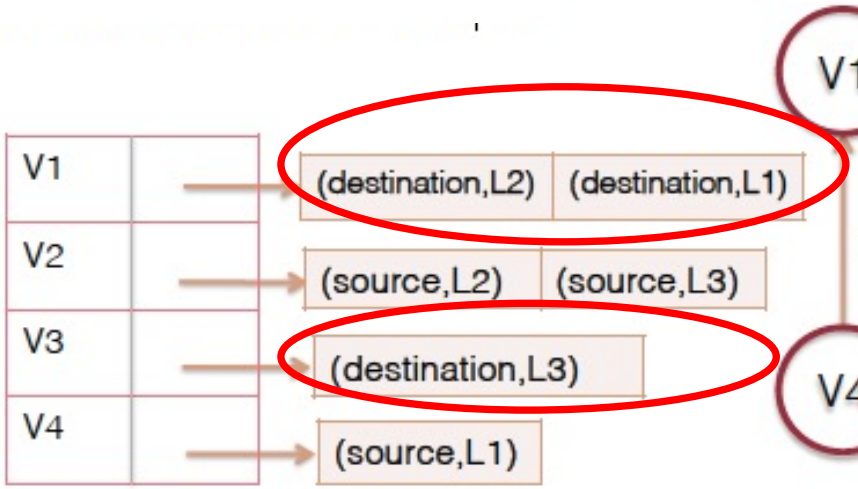
Implementation

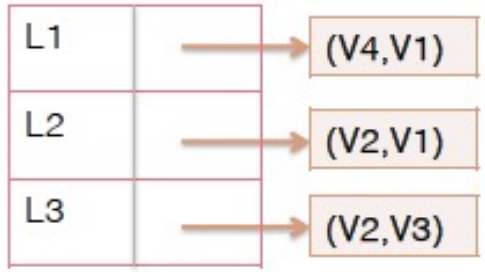


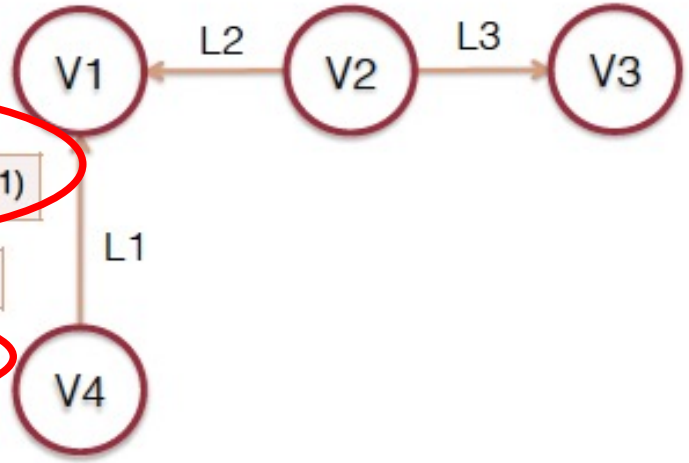
Implementation: incidence list

Incidence List

- Vertices and edges are stored as records of objects.
- Each vertex stores incident edges.
- Each edge stores incident nodes.



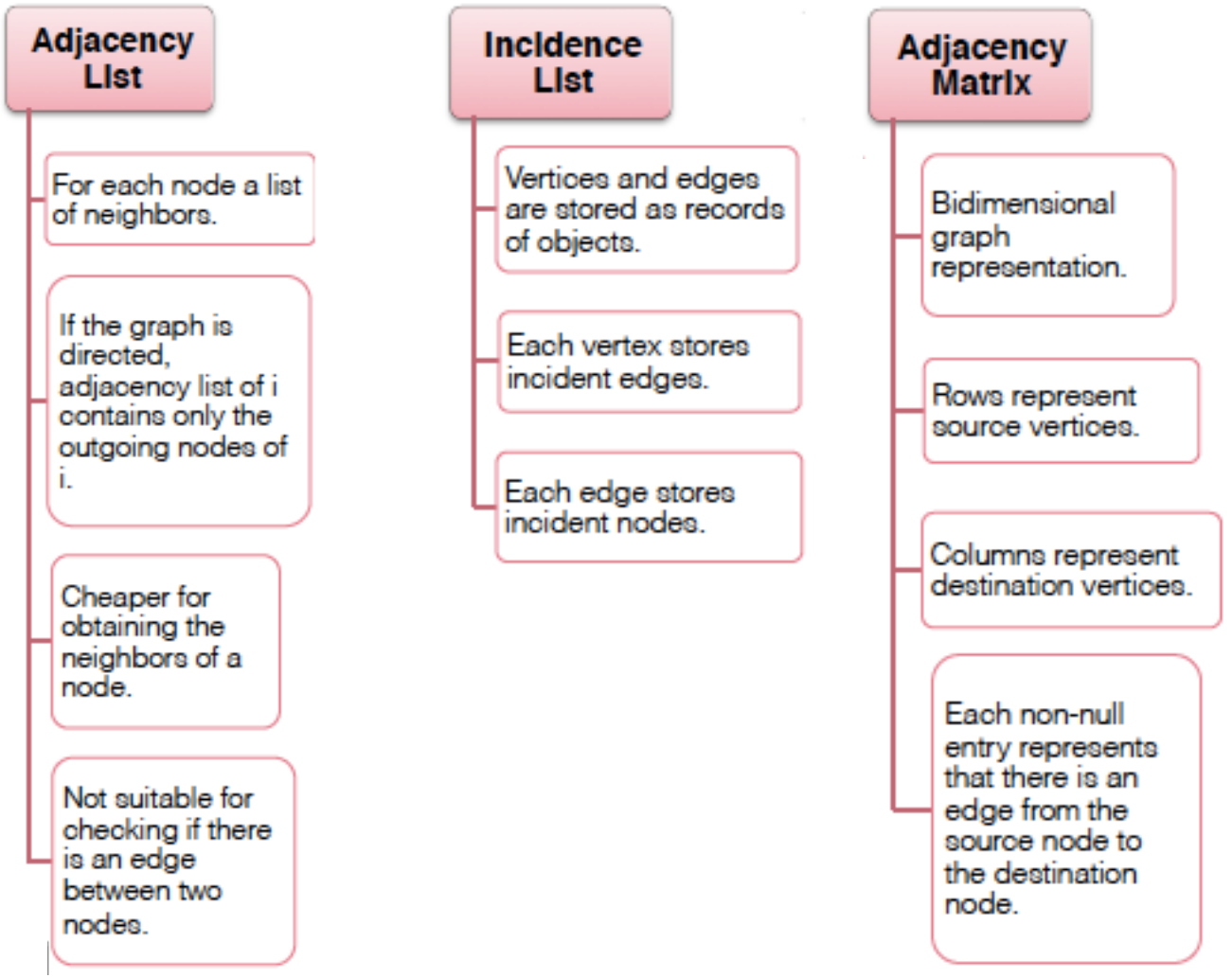




Properties:

- 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

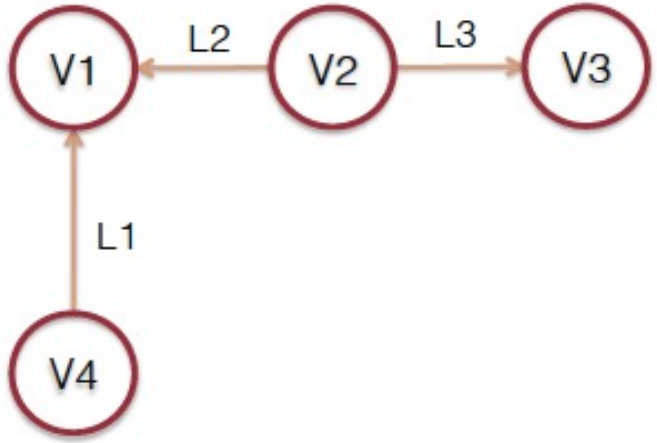


Implementation: adjacency matrix

Adjacency Matrix

- Bidimensional graph representation.
- Rows represent source vertices.
- Columns represent destination vertices.
- Each non-null entry represents that there is an edge from the source node to the destination node.

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



Implementation: adjacency matrix

- Complexity
- Storage

Answer : |V| X |V|

- Is there an edge from X to Z?

Answer : O(1)

- Compute the out-degree of Z

Answer: O(|V|)

- Compute the in-degree of Z

Answer: O(|V|)

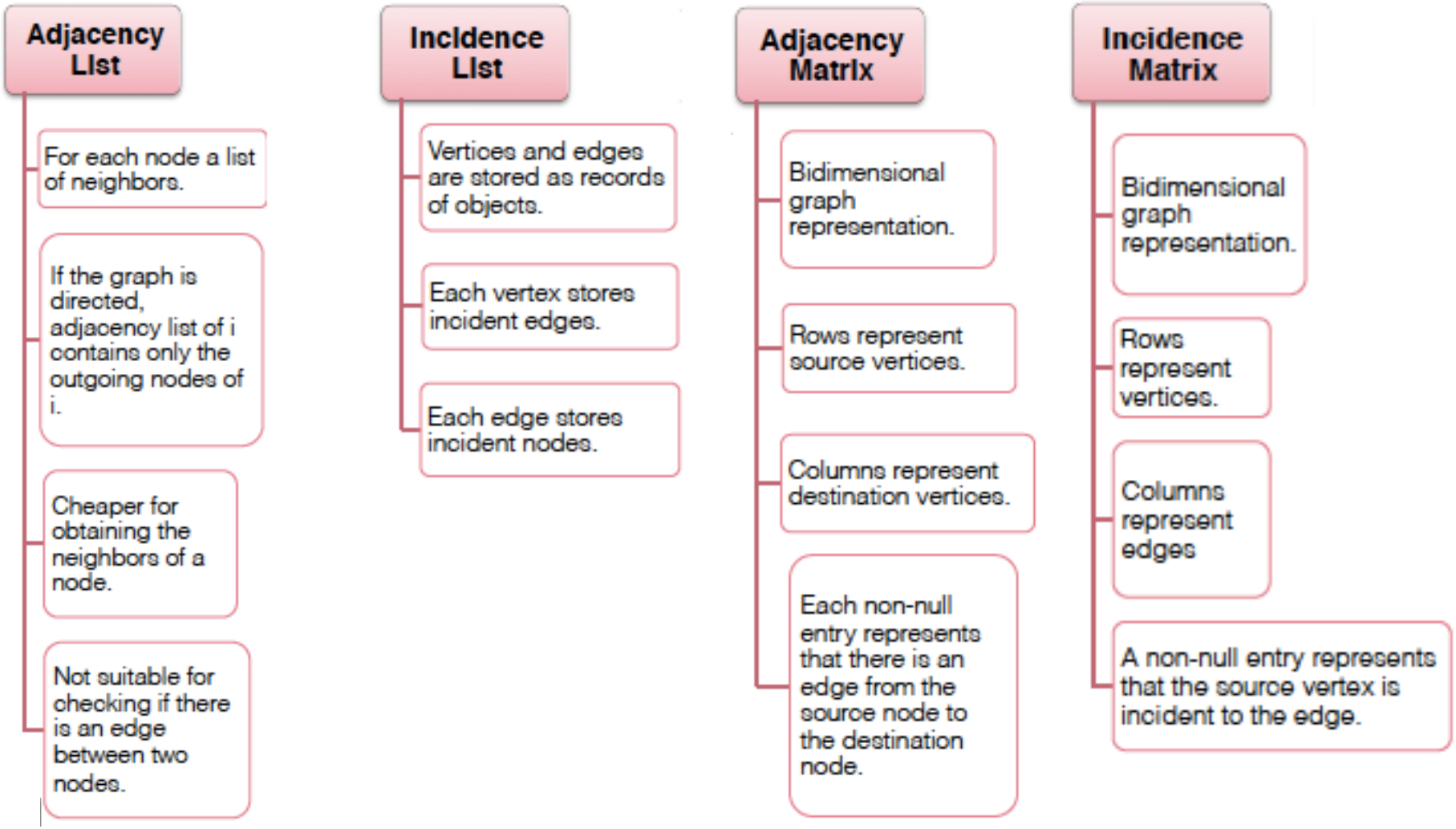
- Add an edge between two nodes

Answer: O(1)

- Compute all paths of length 4 between any pair of nodes (4-hop)

Answer: O($|V|^4$).

Implementation

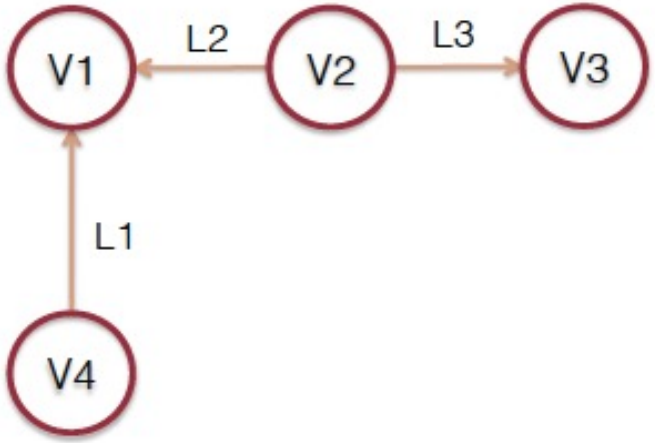


Implementation: incidence matrix

Incidence Matrix

- Bidimensional graph representation.
- Rows represent vertices.
- Columns represent edges
- A non-null entry represents that the source vertex is incident to the edge.

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

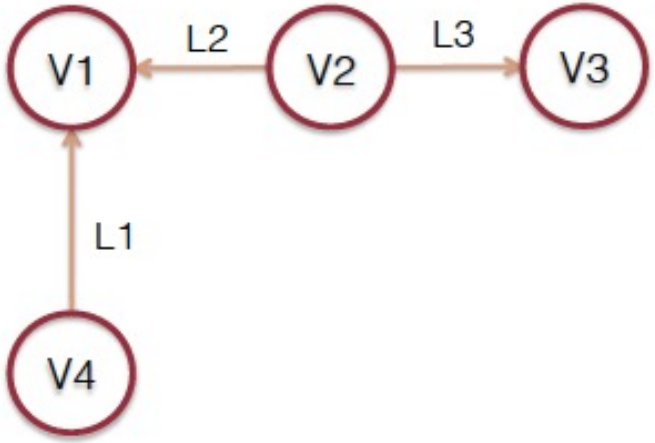


Implementation: incidence matrix

Incidence Matrix

- Bidimensional graph representation.
- Rows represent vertices.
- Columns represent edges
- A non-null entry represents that the source vertex is incident to the edge.

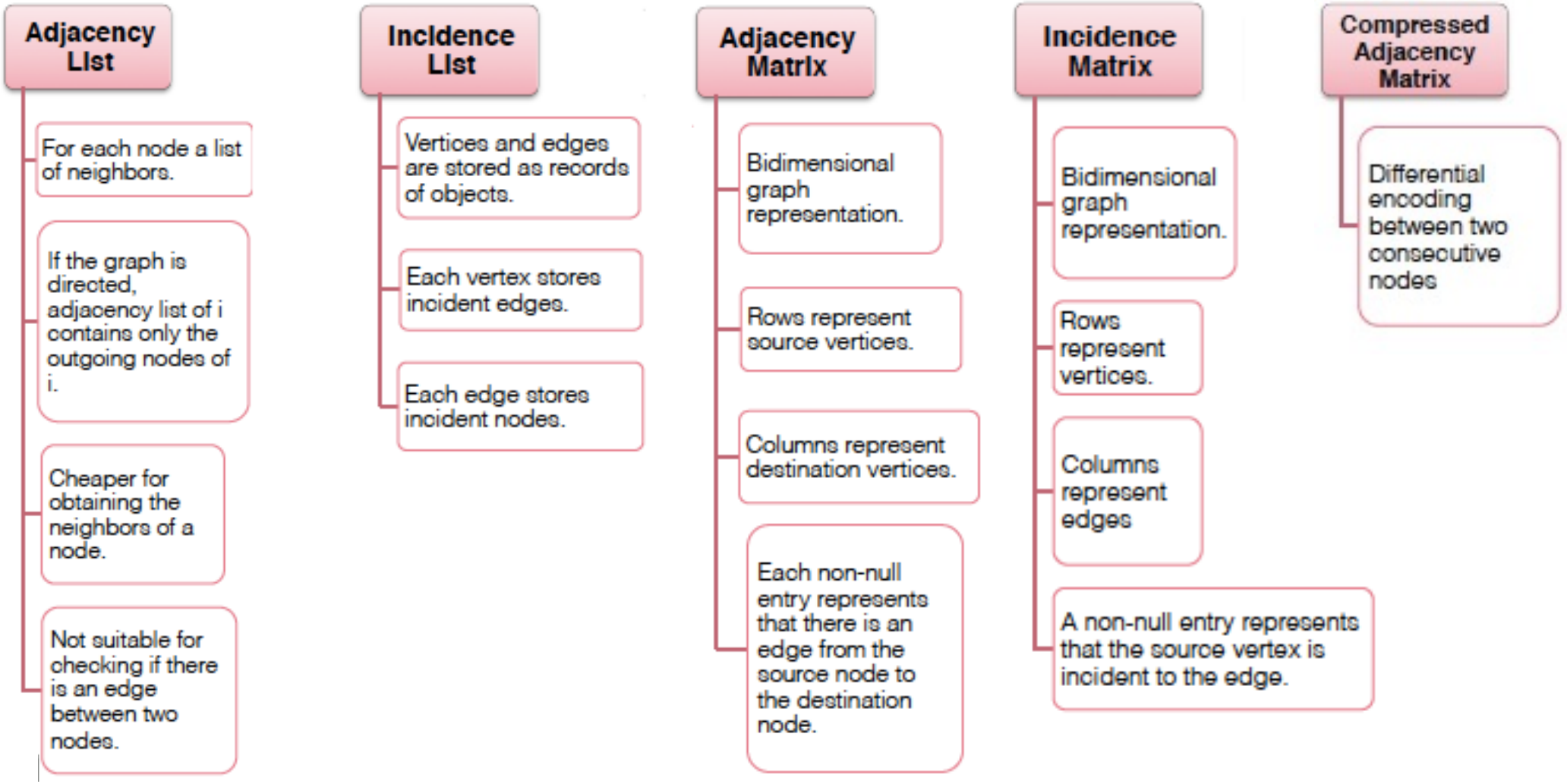
	L1	L2	L3
V1	destination	destination	
V2		source	source
V3			destination
V4	source		



Properties:

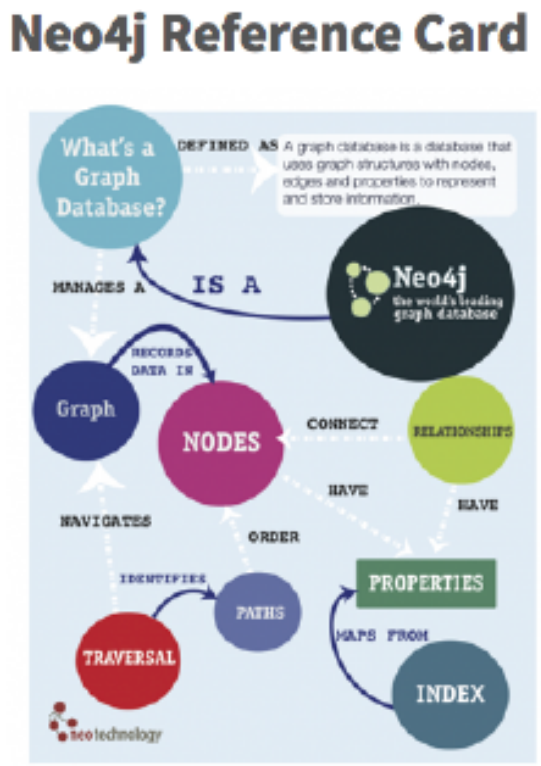
- Storage: O(|V|x|E|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|V|x|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|V|)
- Delete(G,x,y,l): O(|V|)

Implementation



Implementations

Graph databases – Representative approaches



http://www.neo4j.org

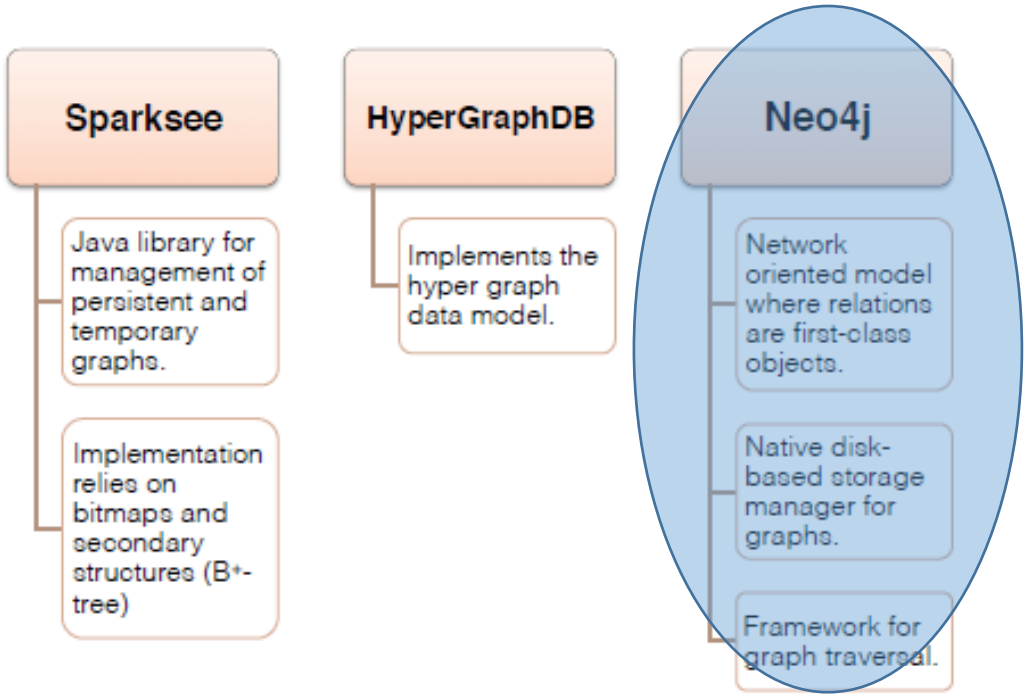
HyperGraphDB

http://www.hypergraphdb.org

*Sparksee

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

Some graph databases



- Some graph db implement an API rather than a query language

Neo4j (Robinson et al., 2013)

- Labelled attributed multigraph
- Nodes and edges can have properties (property graphs)
- No restrictions on the # of edges between nodes
- Loops allowed
- Different types of traversal strategies
- APIs for Java and Python
- Embeddable and server
- Full ACID transactions

Neo4j (Robinson et al., 2013)

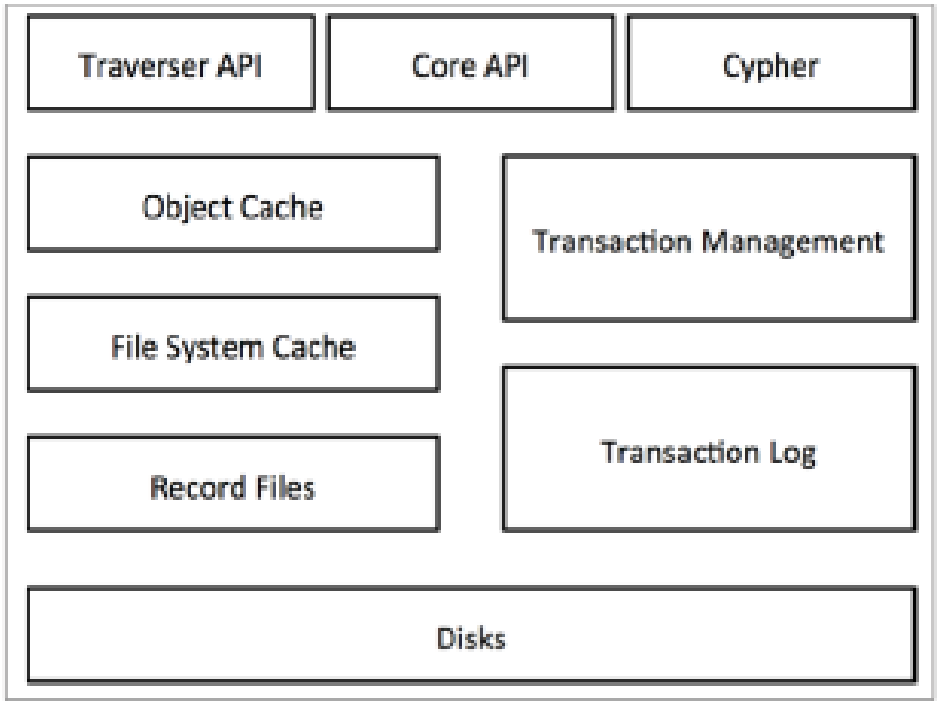
- Native graph processing and storage
  - Characterized by index-free adjacency:
    - Node keeps direct reference to adjacent nodes
    - Acts like a micro-index (or local index)
    - Makes query time independent from graph size for many queries
  - Joins are “precomputed” and stored as relationships
    - In non-native graph DBs, joins must be computed

Neo4j (Robinson et al., 2013)

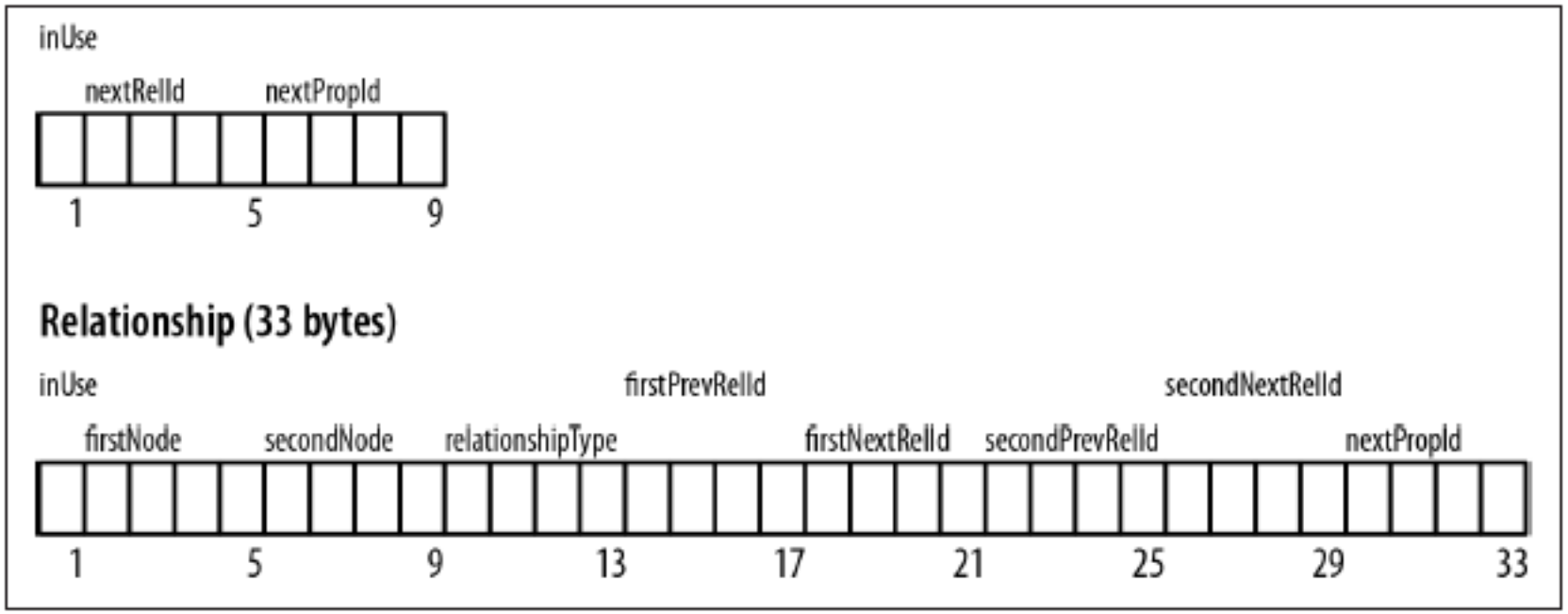
- Native graph storage
  - Storing graphs in files
  - Loading graphs into main memory
  - Caching graphs for fast querying

Neo4j - architecture

Robinson et al., 2013

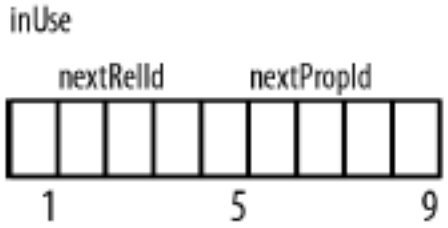


File storage



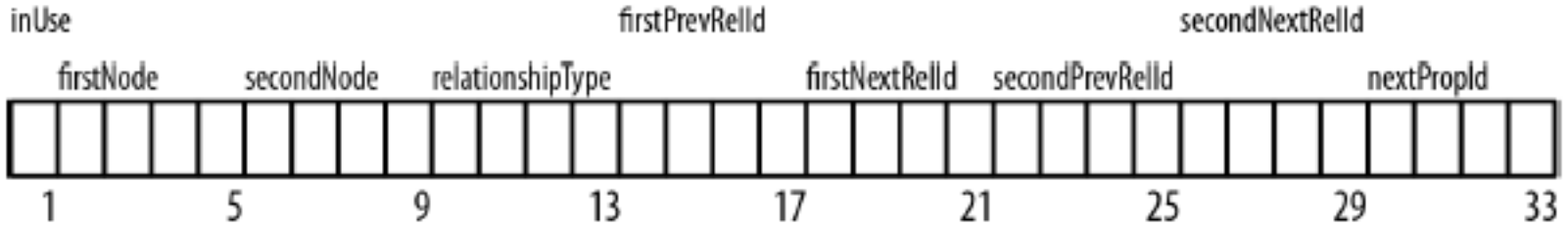
- Graphs stored in store files
  - Nodes (neostore.nodestore.db)
  - Relationships (neostore.relationshipstore.db)
  - Properties (neostore.propertystore.db)

File storage: nodes



- Stored in node records
  - Fixed length (9 bytes) to make search performant (find records with an offset from the node id)
    - Finding a node is O(1)
  - First byte: in-use flag
  - 4 bytes for the address of the first relationship
  - 4 bytes for the first property

File storage: relationships

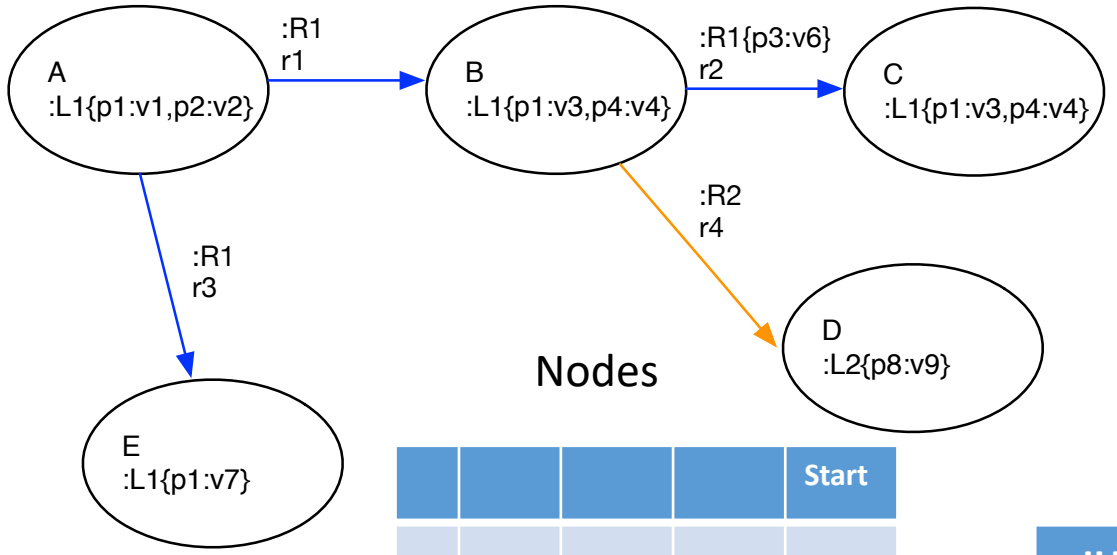


- Stored in relationship records
  - Fixed length (33 bytes)
  - First byte: in-use flag
  - Organized as a double-linked list
  - Each record contains the IDs of the two nodes in a relationship (start and end nodes)
  - A pointer to the relationship type
  - For each node, there is a pointer to the previous and next relationship records
    - E.g.: firstPrevRelID: **previous relationship of the start node**; firstNextRelID**: next relationship of the start node** (the one after the current relationship)
    - These form the relationship chain

File storage: properties

- Stored in property records
  - Fixed length
  - Each record consists of 32 bytes divided in blocks of 8 bytes
  - Include the ID of the next property in the property chain
  - Property chains: single-linked list
  - Each property record holds:
    - Property type
    - Pointer to the property index file, holding the property name
    - A value, or a pointer to a dynamic structure (string or array store) – for long strings or arrays

File storage: example



Nodes

				Start
1	A	np1	..	r1
2	B	..	..	r2
3	C	..	..	Nil
4	D	..	..	Nil
5	E	np7	..	Nil

Relationship Types

ID1	R1
ID2	R2

Properties

rp3	p3	v6	NIL
np1	p1	v1	np2
np2	p2	v2	NIL
np7	p1	v7	NIL
	...	...	...
	...	...	...

Relationships

	IU	Fst	Snd	RT	FPrev	FNext	SPrev	SNext	NP
r1	1	A	B	ID1	NIL	r3	NIL	r2	NIL
r2	1	B	C	ID1	NIL	r4	NIL	NIL	rp3
r3	1	A	E	ID1	r1	NIL	NIL	NIL	NIL
r4	1	B	D	ID2	r2	NIL	NIL	NIL	NIL

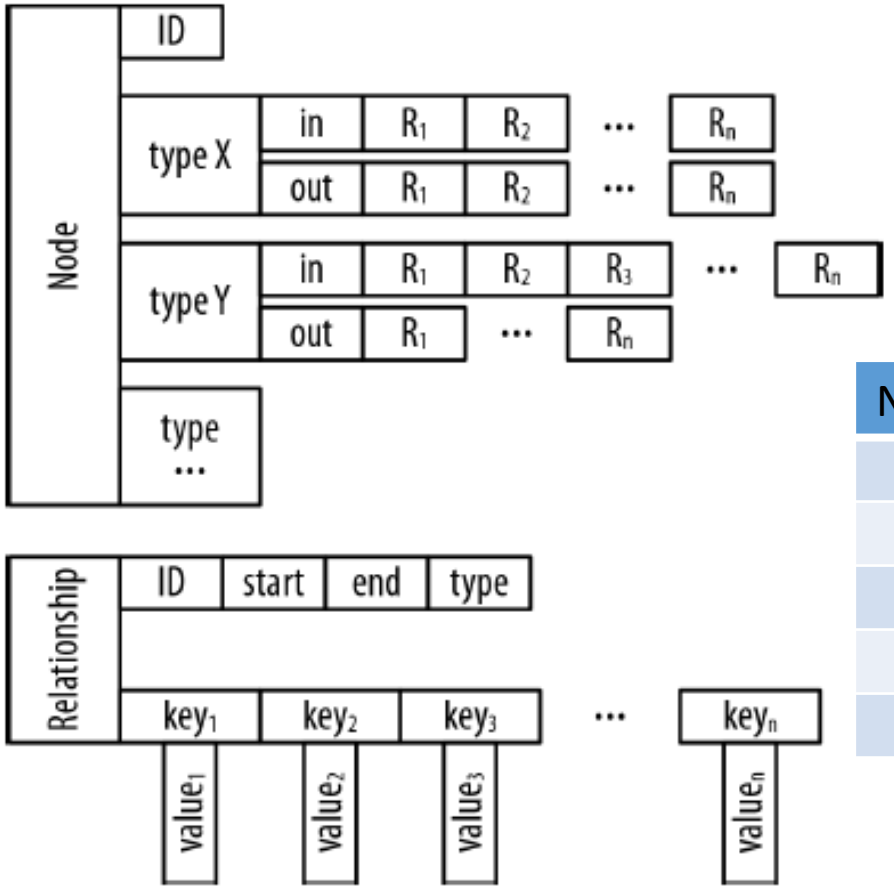
In a DFS, start from r1, then r2, r4, r3 (see table "Relationships"). We have all the information.

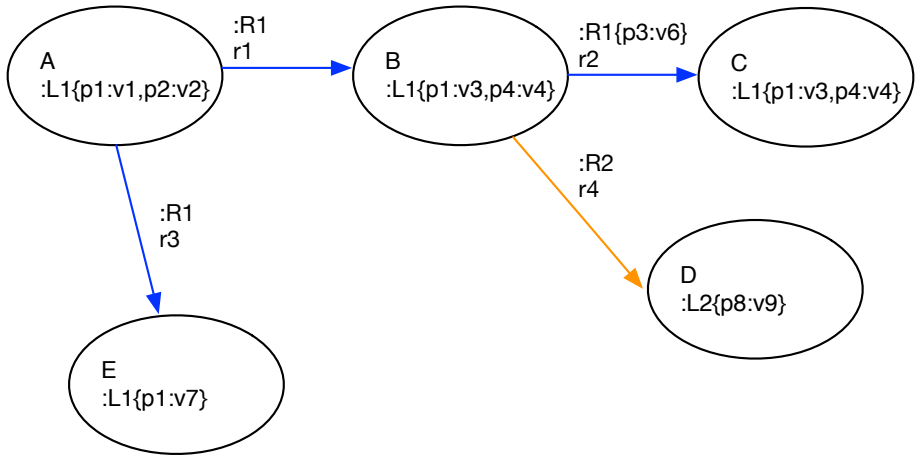
Caching

- File system cache (writing)
  - Cache divides each store into regions (pages)
  - Stores a fixed number of pages per file
  - Pages are replaced using Least Frequently Used pages

- Object cache
  - Optimized for reading
  - Stores object representations of nodes, relationships, and properties for fast path traversal
  - Node objects: contain properties and references to relationships
  - Relationship objects: contain only their properties
  - This is opposite to what happens in disk storage, where most information is in the relationship records

Object cache

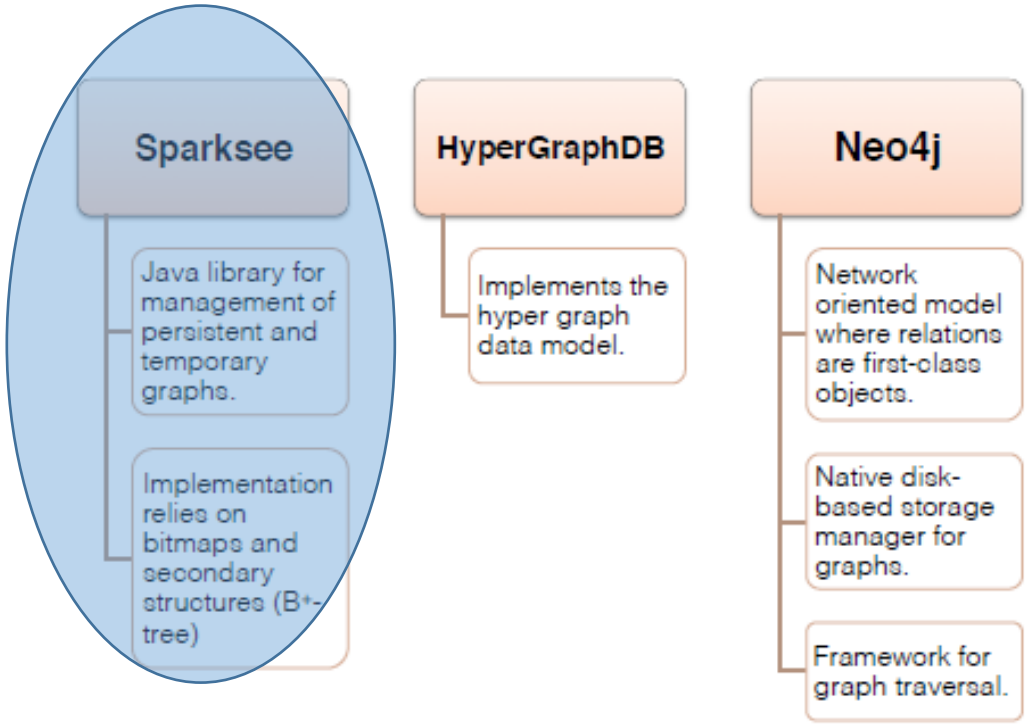




Node	Type	REL
n2	B	IN: r1
	B	OUT: r2
	B	OUT: r4
n3	C	IN: r2
...	...	...

REL	Start	End	Type
r1	A	B	R1
r2	B	C	R1
	(p3,v6)		
r3	A	E	R1
r4	B	D	R2

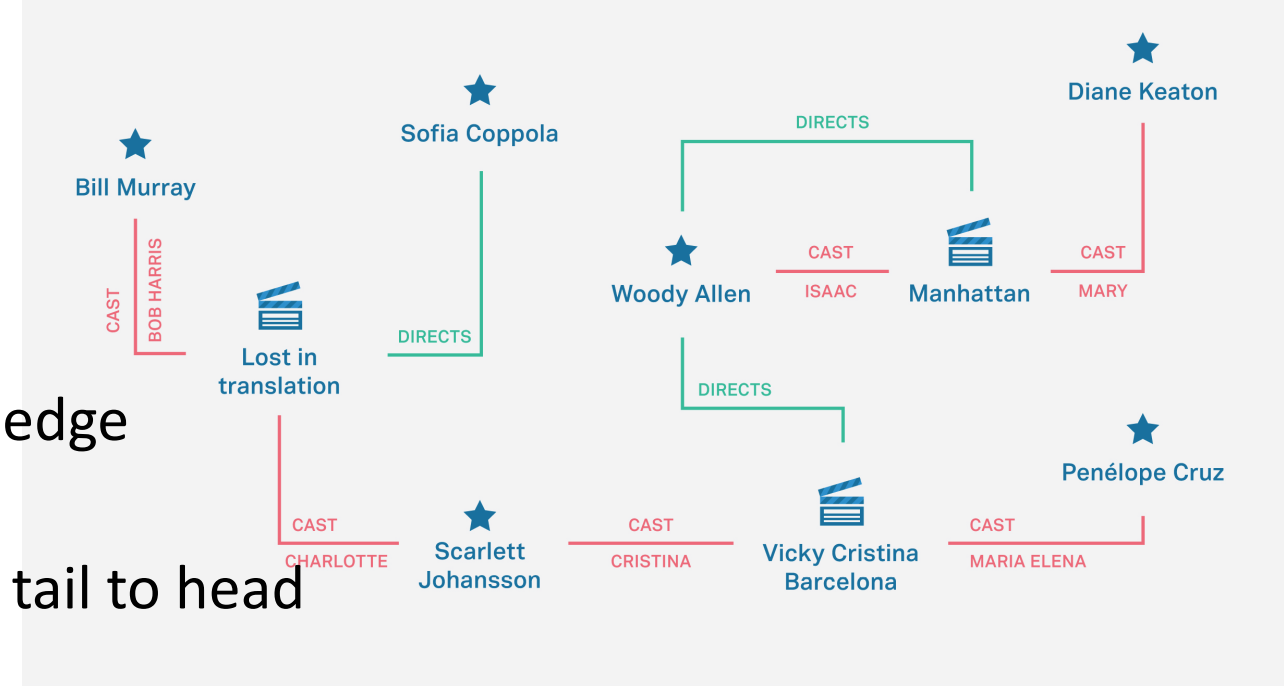
Some graph databases



- Some graph db implement an API rather than a query language

Sparksee

- Logical model
  - Labeled
    - a label for each vertex and edge
  - Directed
    - fixed direction edges, from tail to head
  - Attributed
    - variable # for each vertex)
  - Multigraph
    - possibly more than one edge between nodes
  - Embedded graph dbms
    - tightly integrated with the application at code level



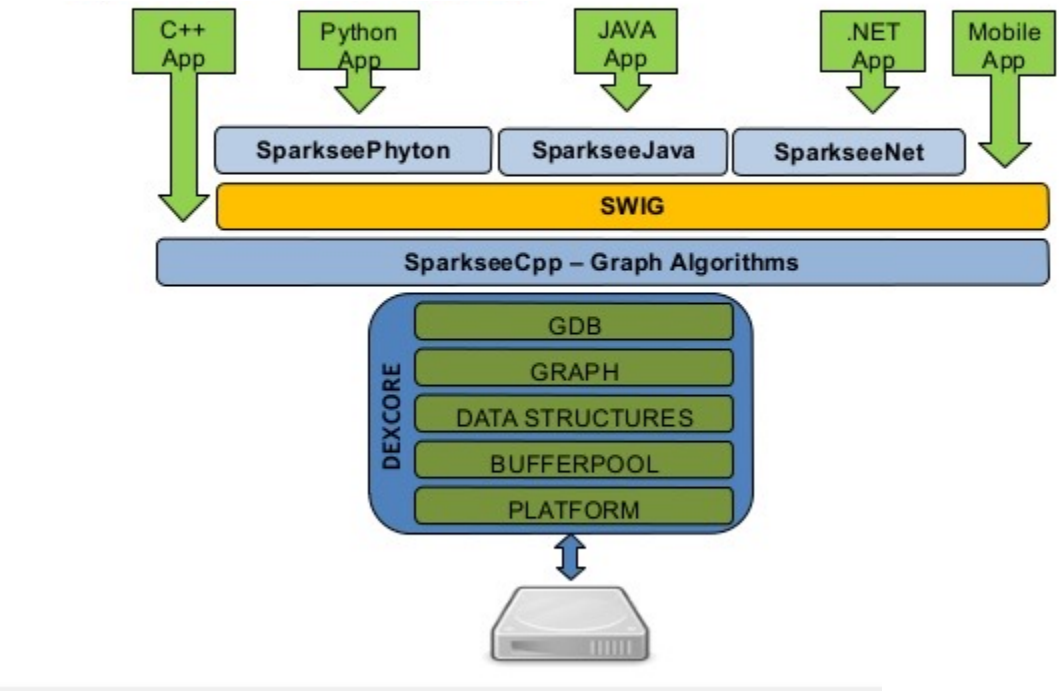
Sparksee

- Nodes and edges have a sparksee-generated OID
- Node, edge and global attributes
  - Not restricted to an edge or node type (e.g., NAME can belong to all node objects)
  - Global attributes belong to the graph
- Attributes can have different **indexes**
    - Basic attributes
    - Indexed attributes
    - Unique attributes
    - Neighborhood index
- Persistent database in a single file
- Can manage very large graphs

Sparksee

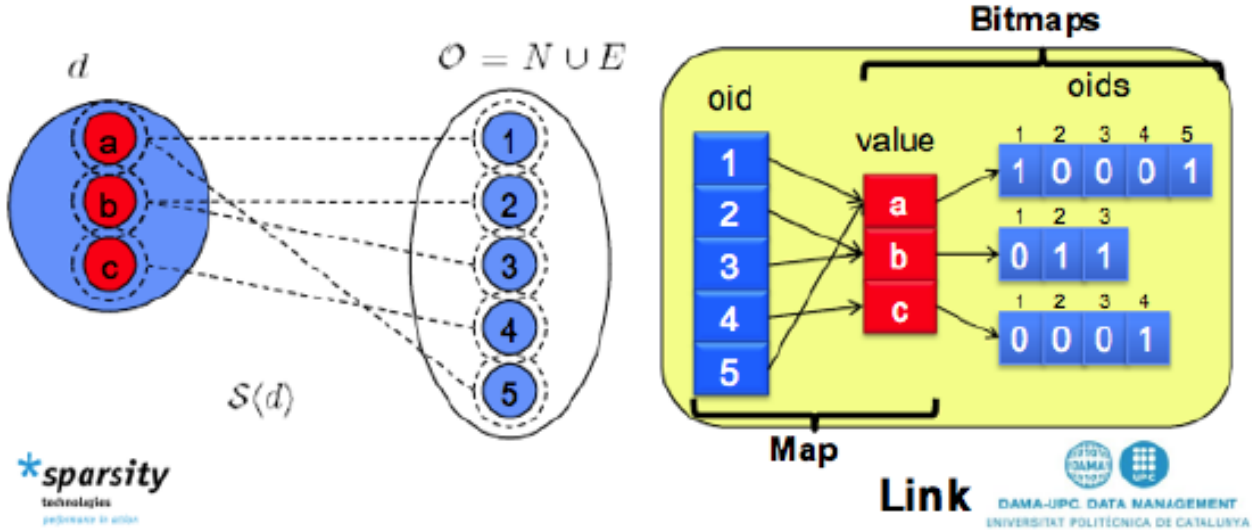
- A graph G= (V,E,L,T,H, A1,……,An) is defined as:
  - Labels L= {(*o,l*)| *o* ∈ (V ∪ E) ∧ *l* ∈ *string*}
  - Heads H= {(*e,h*)| *e* ∈ E ∧ *h* ∈ V}
  - Tails T={(*e,t*)| *e* ∈ E ∧ *t* ∈ V}
  - Attributes Ai={(*o,c*)| *o* ∈ (V ∪ E) ∧ *c* ∈ (*int,string,…*}
  - The graph is split into multiple lists of pairs
  - The first element in a pair is always an edge or a vertex

Sparksee - architecture



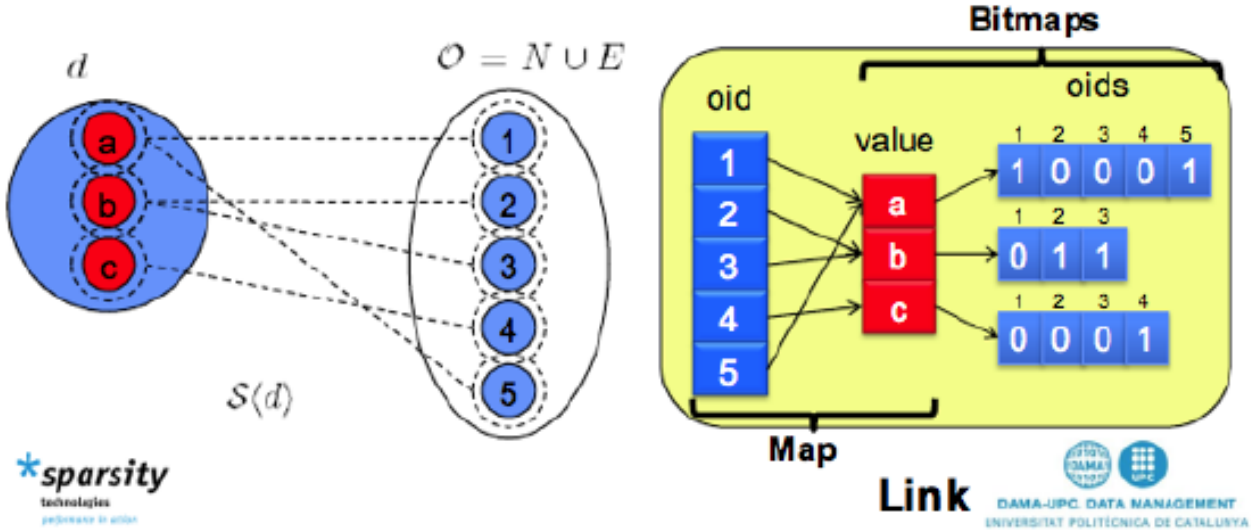
*SWIG = Simplified Wrapper and Interface Generator. Open source tool used to connect programs/libraries written in C/C++ with other languages.

Sparksee – internal representation



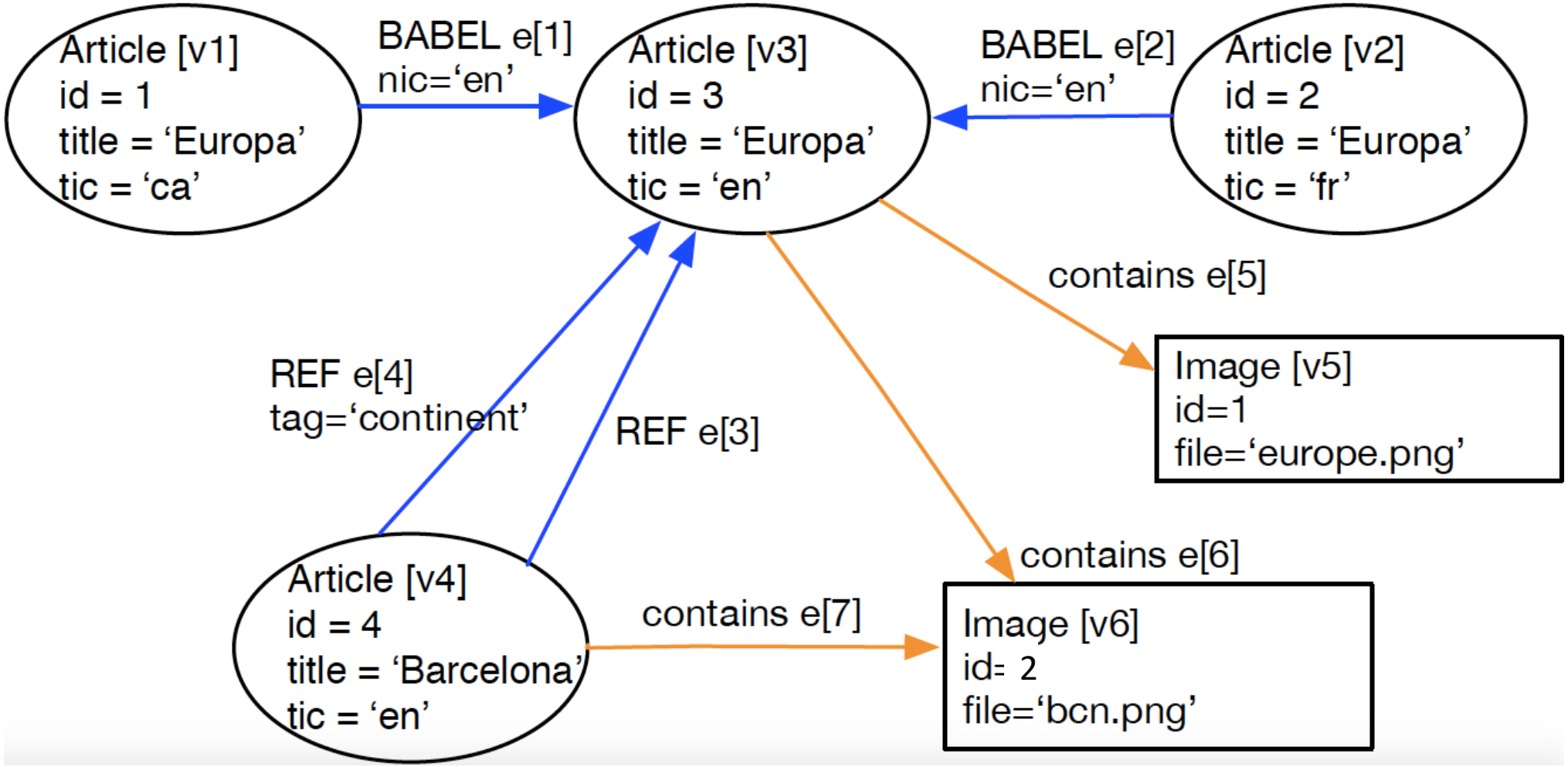
- Each vertex/edge is identified with an immutable oid.
- Links: bidirectional
  - Value - > set of OIDs.
  - Given an OID -> a value.
- Two maps: (a) from a value to a vertex or edge set; (b) from a vertex or edge to an oid.
- Maps are B-trees.

Sparksee – internal representation

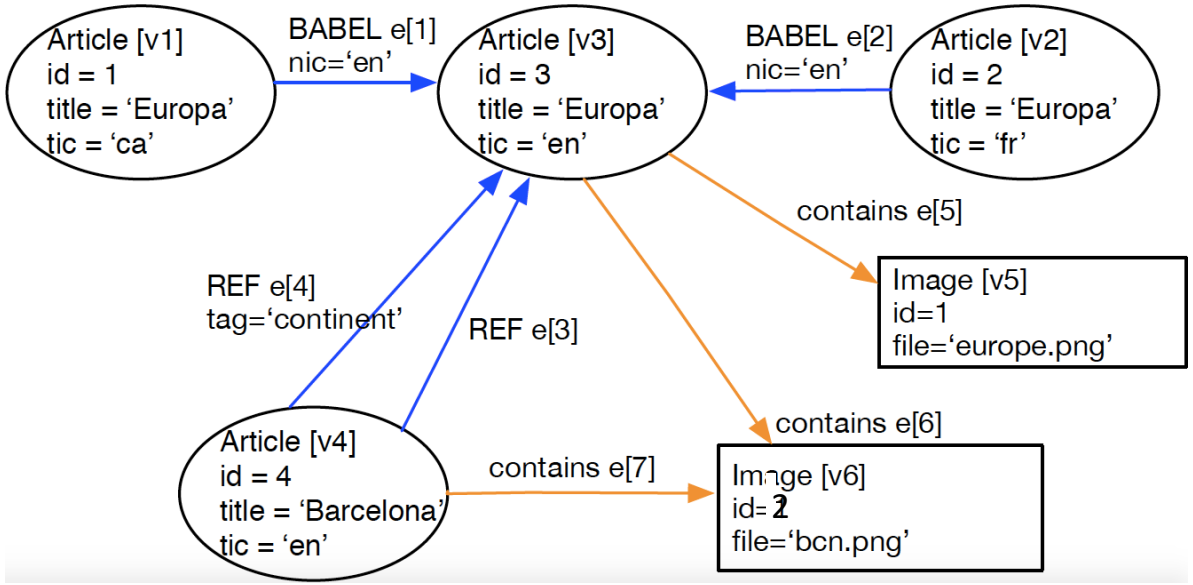


- A Sparksee Graph is a combination of **Bitmaps**:
  - Bitmap for each node or edge set (type).
  - Each position in the bitmap corresponds to the oid.
  - One link for each attribute.
  - Two links for each type: Outgoing and in-going edges.
- Maps are B+trees
  - A compressed UTF-8 storage for UNICODE string.

Sparksee – example



Sparksee – example



Value sets: group all pairs of the original set with the same value, as a pair between the value and the set of objects with that value

L	v_1, ARTICLE), (v_2, ARTICLE), (v_3, ARTICLE), (v_4, ARTICLE), (v_5, IMAGE), (v_6, IMAGE), (e_1, BABEL), (e_2, BABEL), (e_3, REF), (e_4, REF), (e_5, CONTAINS), (e_6, CONTAINS), (e_7, CONTAINS)	(ARTICLE, {v_1, v_2, v_3, v_4}), (BABEL, {e_1, e_2}), (CONTAINS, {e_5, e_6, e_7}), (IMAGE, {v_5, v_6}), (REF, {e_3, e_4})
T	(e_1, v_1), (e_2, v_2), (e_3, v_4), (e_4, v_4), (e_5, v_3), (e_6, v_3), (e_7, v_4)	(v_1, {e_1}), (v_2, {e_2}), (v_3, {e_5, e_6}), (v_4, {e_3, e_4, e_7})
H	(e_1, v_3), (e_2, v_3), (e_3, v_3), (e_4, v_3), (e_5, v_5), (e_6, v_6), (e_7, v_6)	(v_3, {e_1, e_2, e_3, e_4}), (v_5, {e_5}), (v_6, {e_6, e_7})
Aid	(v_1, 1), (v_2, 2), (v_3, 3), (v_4, 4), (v_5, 1), (v_6, 2)	(1, {v_1, v_5}), (2, {v_2, v_6}), (3, {v_3}), (4, {v_4})
Atitle	(v_1, Europa), (v_2, Europe), (v_3, Europe), (v_4, Barcelona)	(Barcelona, {v_4}), (Europa, {v_1}), (Europe, {v_2, v_3})
Anlc	(v_1, ca), (v_2, fr), (v_3, en), (v_4, en), (e_1, en),(e_2, en)	(ca, {v_1}), (en, {v_3, v_4, e_1, e_2}), (fr, {v_2})
Afilename	(v_5, europe.png), (v_6, bcn.jpg)	(bcn.jpg, {v_6}), (europe.png, {v_5})
Atag	(e_4, continent)	(continent, {e4})

Sparksee – example