Graph Databases

Knowledge Objectives

1. Describe what a graph database is
2. Explain the basics of the graph data model
3. Enumerate the best use cases for graph databases
4. Pros and cons of graph databases in front of relational databases
5. Pros and cons of graph databases in front of other NOSQL options

Understanding Objectives

1. Simulate traversing processing in a relational database and compare it with a graph database

Application Objectives

1. Model a simple graph databases following the property graph data model
2. Implement graphs in Neo4J and use Cypher for traversing them

Graph Databases in a Nutshell

- Occurrence-oriented
  - May contain millions of instances
    - Big Data!
  - It is a form of schemaless databases
    - There is no explicit database schema
    - Data (and its relationships) may quickly vary
  - Objects and relationships as first-class citizens
    - *An object o relates (through a relationship r) to another object o'*
    - Both objects and relationships may contain properties
  - Built on top of the graph theory
    - Euler (18^{th} century)
    - More natural and intuitive than the relational model

Notation (I)

- A **graph** *G* is a set of nodes and edges: *G (N, E)*
- *N* - **Nodes** (or vertices): n1, n2, … Nm
- *E* - **Edges** are represented as pairs of nodes: (n1, n2)
  - An edge is drawn as a line between n1 *and* n2
  - An edge is said to be **incident** to n1 and n2
  - Also, n1 and n2 are said to be **adjacent**
  - **Directed edges** entail direction: *from* n1 *to* n2
  - An edge is said to be **multiple** if there is another edge exactly relating the same nodes
  - An **hyperedge** is an edge inciding in more than 2 nodes.
- **Simple graph**: If it does not contain multiple edges.
- **Multigraph**: If it contains at least one multiple edge.
- **Hypergraph**: A graph allowing hyperedges.

Notation (II)

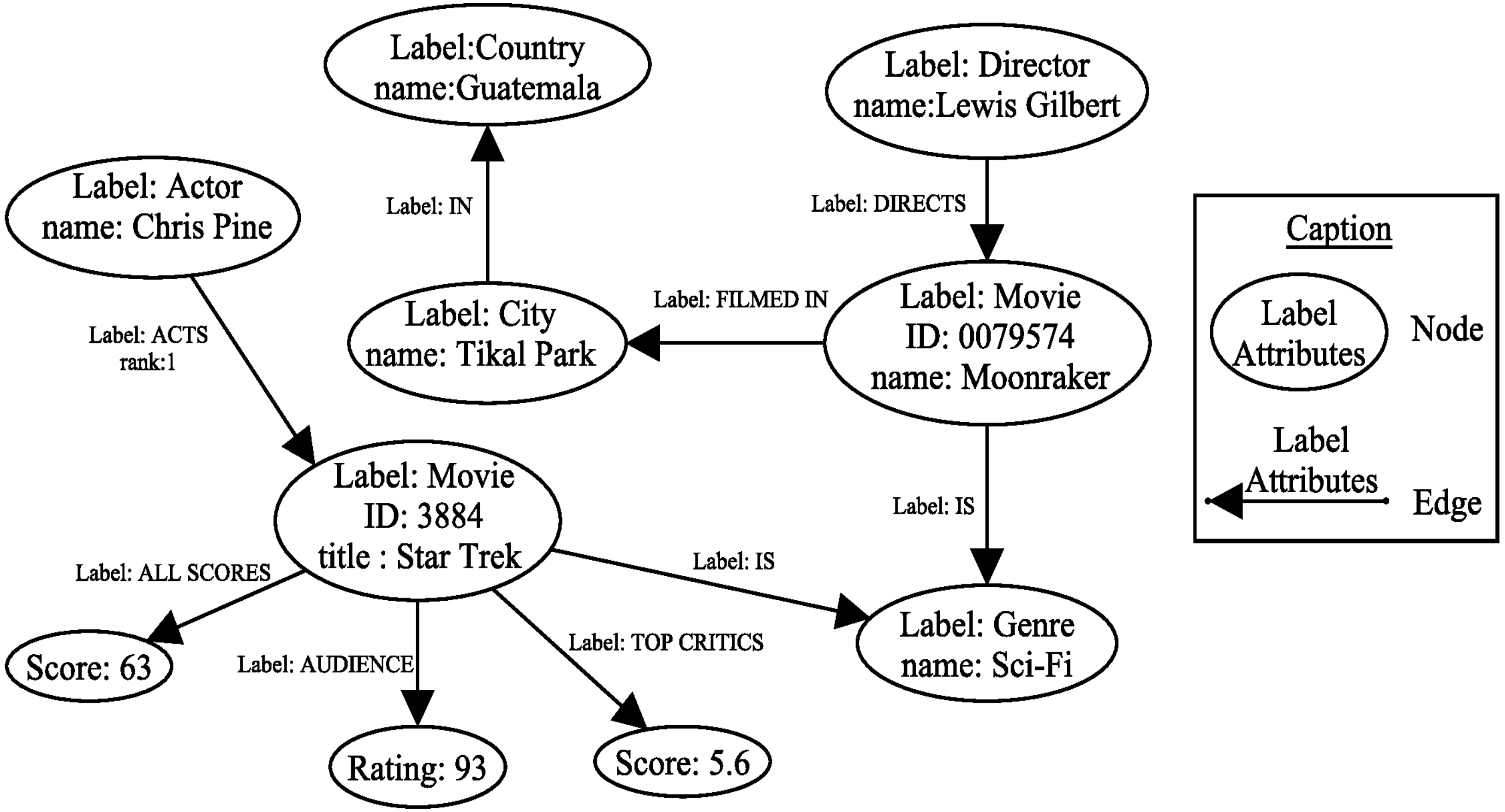
- **Size** (of a graph): #edges
- **Degree** (of a node): #(incident edges)
  - The degree of a node denotes the node adjacency
  - The neighbourhood of a node are all its adjacent nodes
- **Out-degree** (of a node): #(edges leaving the node)
  - Sink node: A node with 0 out-degree
- **In-degree** (of a node): #(incoming edges reaching the node)
  - Source node: A node with 0 in-degree
- Cliques and trees are specific kinds of graphs
  - **Clique**: Every node is adjacent to every other node
  - **Tree**: A connected acyclic simple graph

The Property Graph Data Model

- Two main constructs: nodes and edges
  - Nodes represent entities
  - Edges relate pairs of nodes, and may represent different types of relationships.
- Nodes and edges might be labeled
  - and may have a set of properties represented as attributes (key-value pairs)***
- Further assumptions:
  - Edges are directed
  - Multi-graphs are allowed.

*** *Note: in some definitions edges are not allowed to have attributes*

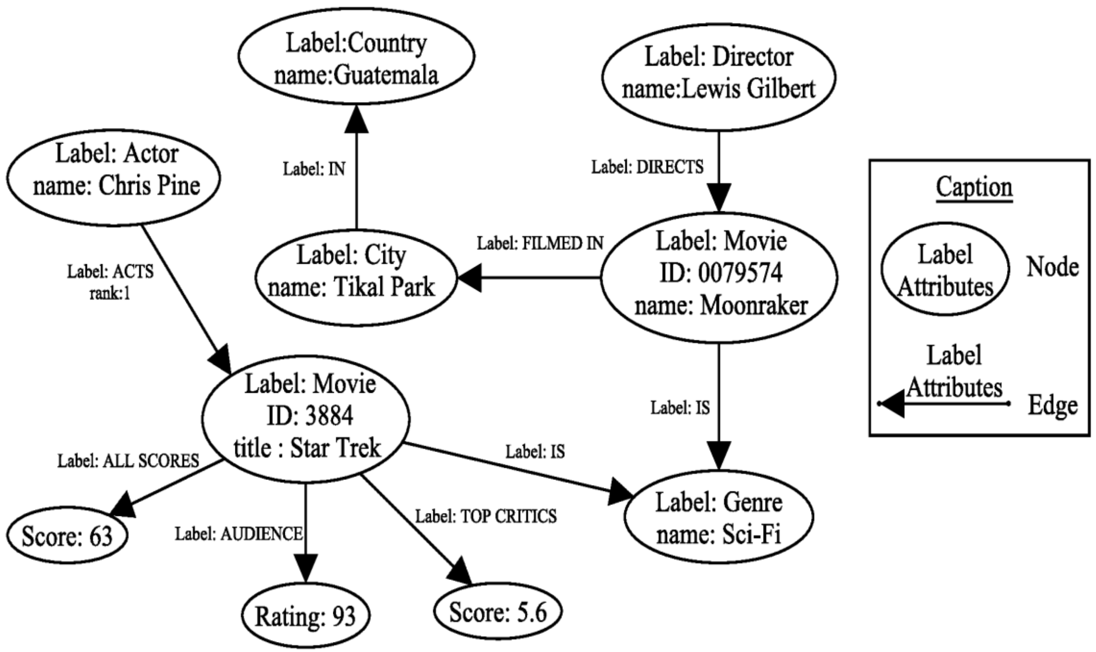
Example of Graph Database



http://grouplens.org/datasets/movielens/

Activity: Querying Graph Databases

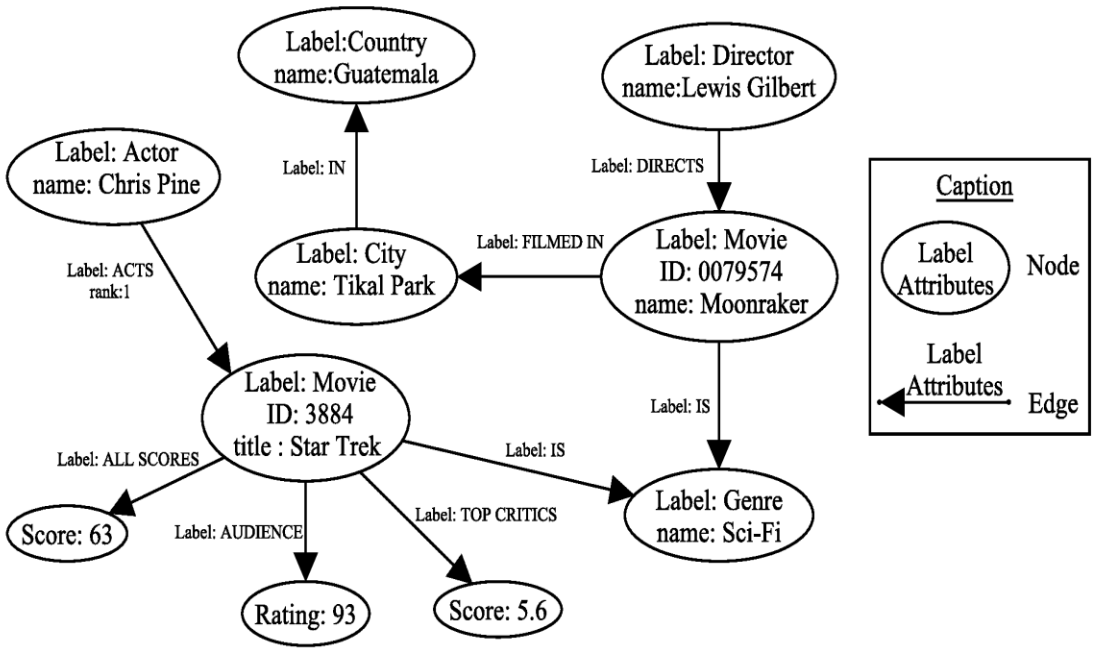
- *Objective: understand the main differences with regard to RDBMS*



- What movies did Lewis Gilbert direct?
  - How would a RDBMS perform this query?

Activity: Querying Graph Databases

- *Objective: understand the main differences with regard to RDBMS*



- What movies did receive a higher rating by the audience?
  - How would a RDBMS perform this query?

Activity: The Graph Data Model

- *Objective: Understand the graph data model*
- *Tasks:*
  1. *(5') With a teammate think of the following:*
     I. *Think of a couple of queries that naturally suit the graph data model*
     II. *Think of a couple of queries that do not suit the graph data model that nicely*
  2. *(5') Think tank: So, what kind of queries graph databases are thought for?*

GDBs Keystone: Traversal Navigation

- We define the graph traversal pattern as: *"the ability to* ***rapidly*** *traverse structures to an* ***arbitrary depth*** *(e.g., tree structures, cyclic structures) and with an* ***arbitrary path description*** *(e.g. friends that work together)"* [Marko Rodriguez]
- Totally opposite to set theory (on which relational databases are based on)
  - Sets of elements are operated by means of the relational algebra

Traversing Data in a RDBMS

- In the relational theory, it is equivalent to joining data (schema level) and select data (based on a value)

User	Address	Phone	Email
Alice			
Bob	456 Bar Ave.		bob@example.org
…	…	…	…
Zach	99 South St.		zach@example.org

Order	Date	Status	UserId
1234	20120808	delivered	Alice
5678	20120816	dispatched	Alice
…	…	…	…
5588	20120613	delivered	Zach

Id	Description	Handling
abcd	Strawberry ice-cream	freezer
efab	Brussels sprouts	
cdef	Espresso beans	
…	…	…

OrderId	ItemId
1234	abcd
1234	efab
1234	cdef
5678	cdef
…	
5588	hijk

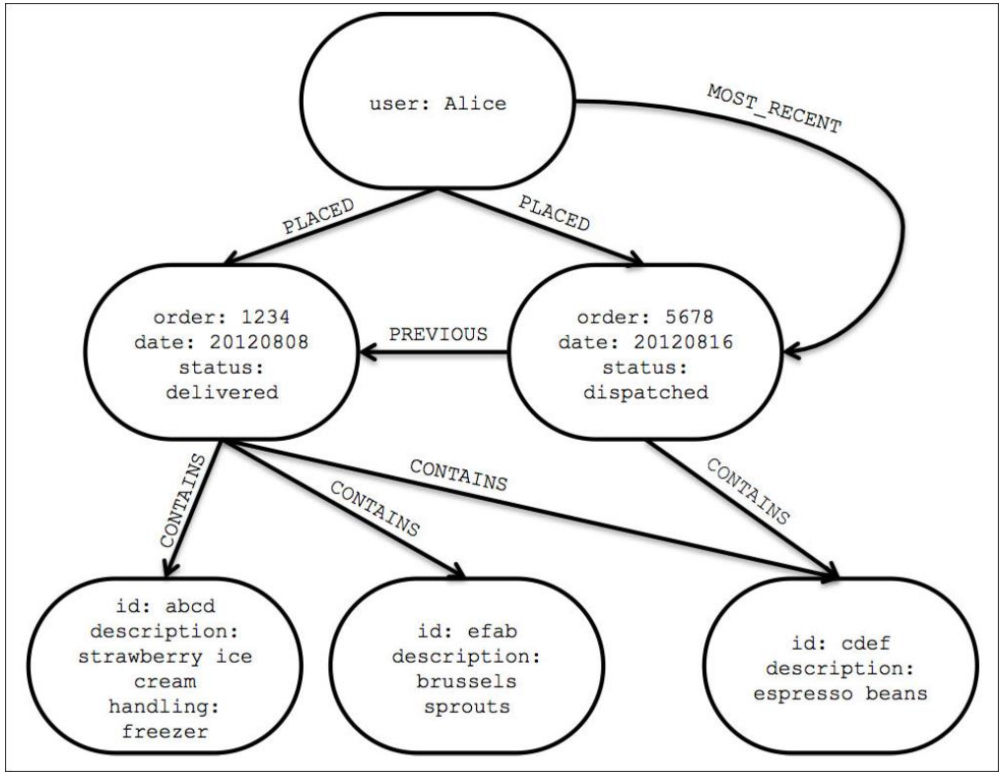
```
SELECT *
FROM users u, orders o,
oder_items oi, items i
WHERE u.user = 'Alice' and
u.user = o.userId AND
o.order = oi.orderId and
oi.itemId = i.id
```

Cardinalities:

|Users|: 5.000.000
|Orders|: 1.000.000.000
|OrderItems|: 3.000.000.000
|Items|: 35.000

Query Cost?!

Traversing Data in a Graph Database



Cardinalities:

|User|: 5.000.000
|Orders|: 1.000.000.000
|Item|: 35.000

Query Cost?!

O(N)

Graph DBs vs Relational DBs (experim.)

- The following table reports result of an experiment aimed to finding friends-of-friends in a social network, to a maximum depth of 5
  - For a social network containing 1,000,000 people, each with approximately 50 friends.

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

From *Neo4j in Action. Jonas Partner, Aleksa Vukotic, and Nicki Watt. MEAP. 2012*

Typical Graph Operations

Refreshing some basics on graphs

Typical Graph Operations

- Content-based queries
  - The value is relevant
    - Get a node, get the value of a node / edge attribute, etc.
    - A typical case are summarization queries (i.e., aggregations)
- Topological queries
  - Only the graph topology is considered
  - Typically, several business problems (such as fraud detection, trend prediction, product recommendation, network routing or route optimization) are solved using graph algorithms exploring the graph topology
    - Computing the betweenness centrality of a node in a social network an analyst can detect influential people or groups for targeting a marketing campaign audience.
    - For a telecommunication operator, being able to detect central nodes of a network helps optimizing the routing and load balancing across the infrastructure.
- Hybrid approaches

Topological Queries (I)

- Categories of queries

  - Adjacency queries
    - Basic node / edge adjacency
    - K-neighbourhood of a node

    Linear cost (on the number of edges to explore)

    Examples: Return all the friends of a person

  - Reachability queries (formalized as a traversal)
    - Fixed-length paths (fixed #edges and nodes)
    - Regular simple paths (restrictions as regular expressions) – Hybrid if the restriction is in the *content*
    - Shortest path

    Hard, to compute, in general, NP-complete

    Examples: Friend-of-a-friend

Topological Queries (II)

- Categories of queries (cont'd)

  - Pattern matching queries (formalized as the graph isomorphism problem)

    Hard, to compute, in general, NP-complete

    Examples: Customers without orders

  - Graph metrics
    - Compute the graph / node order, the min / max degree in the graph, the graph diameter, the graph density, closeness / betwenness of a node, the pageRank of a node, etc.

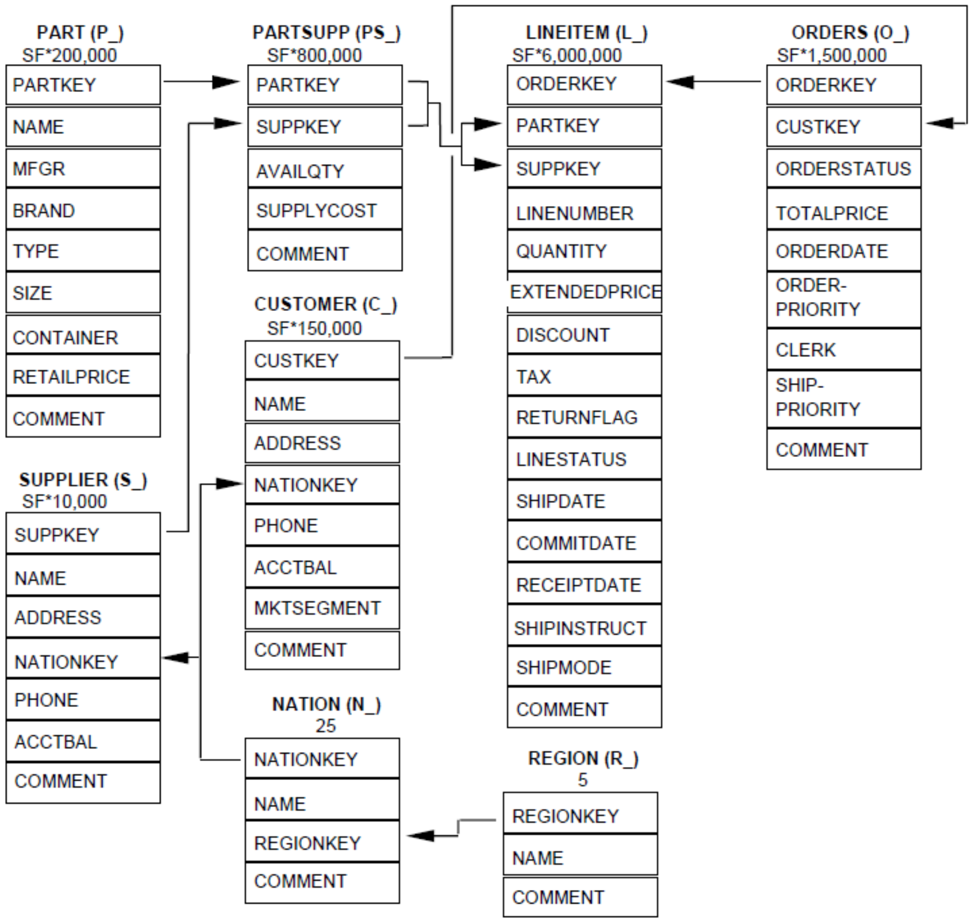
    Cost depends on the metric to be computed.

    Examples: Compute business processes bottlenecks (betweenness)

Activity: Modeling in Graph DBs

- *Objective: Learn how to model graphs*
- *Tasks:*
  - *(15') Model the TPC-H database as a graph*
  - *(5') Discussion*

```
SELECT l_orderkey,
sum(l_extendedprice*(1-
l_discount)) as revenue,
o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]'
AND c_custkey = o_custkey AND
l_orderkey = o_orderkey AND
o_orderdate < '[DATE]' AND
l_shipdate > '[DATE]'
GROUP BY l_orderkey,
o_orderdate, o_shippriority
ORDER BY revenue desc,
o_orderdate;
```



Internals of Graph Databases

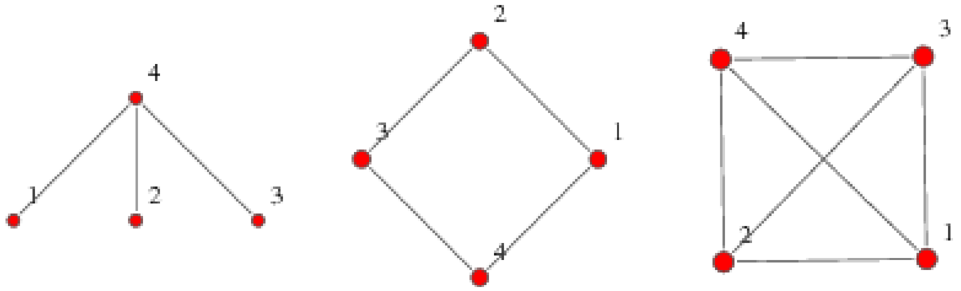
Implementation of Graphs

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

Implementation of Graphs

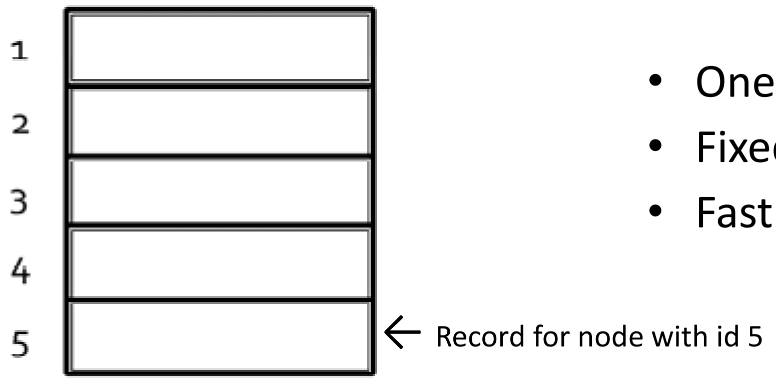
- Adjacency matrix (baseline)



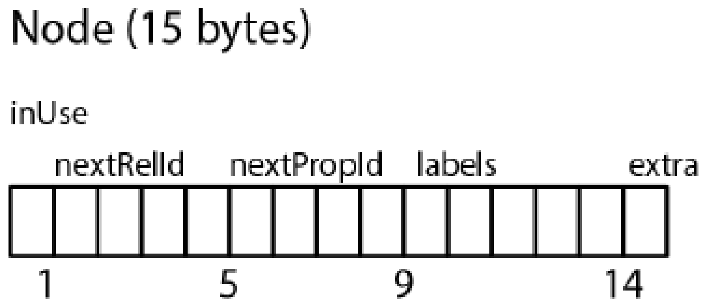
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Incidence Lists – Neo4J

- Implemented with linked lists



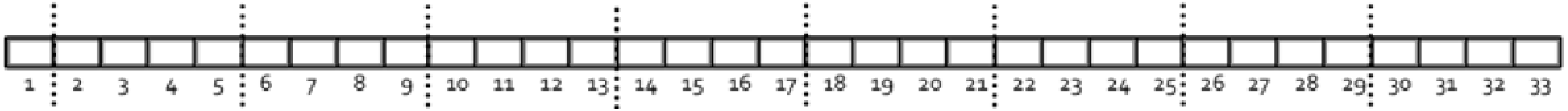
- One physical file to store all nodes (in-memory, or partly cached)
- Fixed-size record: 15 bytes in length
- Fast look-up: O(1)



- Each record is as follows:
- Byte 1 (metadata; e.g., in-use)
- Bytes 2-5: id first relationship
- Bytes 6-9: id first property
- Bytes 10-14: labels
- Byte 15: extra information

Linked Lists – Neo4J

- Two files: relationship and property files.
  - Both contain records of fixed size
  - Cache with *Least Frequently Used (LRU)* policy
- Relationship file (similarly for properties)
  - Metadata, id starting node, id end node, id type, ids of the previous and following relationship of the starting node and ending node, id first property

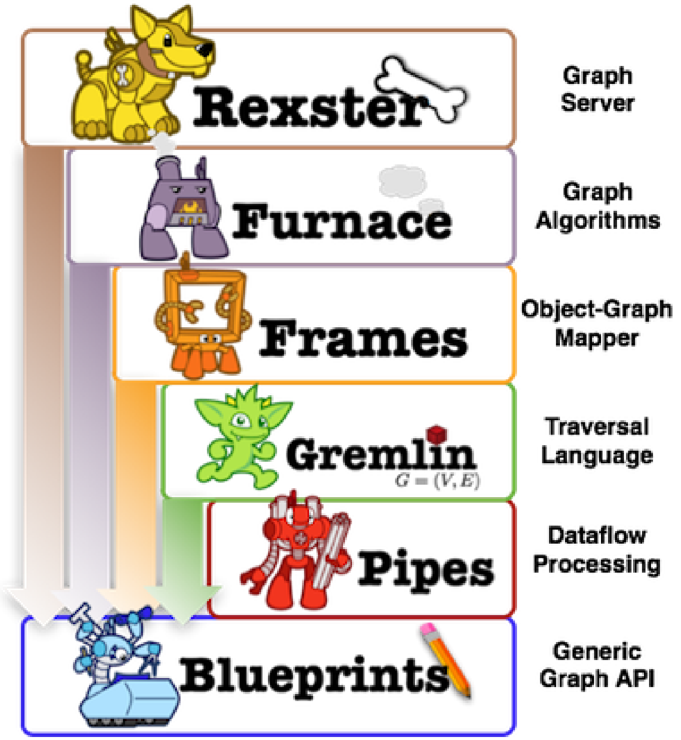


Other Issues

- Distributed Graph Databases
  - Graph databases, by nature, do not distribute well (need to identify graph components)
  - Most popular prototypes:
    - GraphX (Pregel on top of Spark): Discontinued
    - Titan
- Graphs on top of column-oriented databases
    - Vertexica (based on Vertica): http://people.csail.mit.edu/alekh/vertexica.html
    - SynopSys (on top of SAP HANA): http://event.cwi.nl/grades2013/16-Rudolf.pdf

Common Interfaces for GDBs

- Apache TinkerPop
  - Interfaces for the property graph data model
  - It is a stack of technologies
    - Blueprints: Generic graph API
    - Pipes: A data flow framework
    - Gremlin: A graph traversal language
    - Frames: A object-to-graph mapper
    - Furnace: A graph algorithms package
    - Rexster: A graph server



- TinkerPop-compliant graph databases:
  - Neo4j
  - Sparksee
  - OrientDB
  - Titan
  - And many others (see https://tinkerpop.apache.org/)
- TinkerPop enables storage-agnostic applications for graph databases

Neo4j

An example of Graph Database

Neo4J Data Model

- It is a graph!
  - Use nodes to represent entities
  - Use relationships to represent the semantic connection between entities
  - Use node properties to represent entity attributes plus any necessary entity metadata such as timestamps, version numbers, etc.
  - Use relationship properties to represent connection attributes plus any necessary relationship metadata, such as timestamps, version numbers, etc.
- Unique constraints (~PK) can be asserted
  - `CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS UNIQUE`
  - An index is also added to the property (in all the nodes with the indicated label)

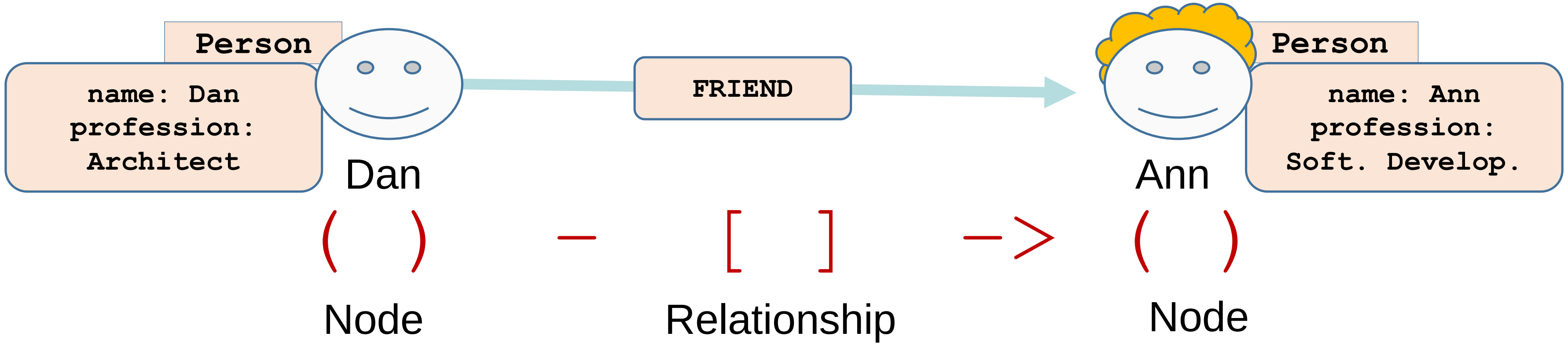
Neo4J: Querying

- The Traversal Framework (from any Neo4J API)
- Cypher
  - High-level, declarative language. It is both a DDL and a DML
    - Opposite idea to Gremlin
    - Query optimizer, welcome!
  - Internally, it uses the traversal framework
  - It can be used from the Shell
  - Contains several clauses:
    - CREATE: Creates nodes and relationships.
    - DELETE: Removes nodes, relationships and properties.
    - SET: Set values to properties.

    - MATCH: The graph pattern to find.
    - WHERE: Filtering criteria.
    - RETURN: What to return.
    - WITH: Divides a query into multiple, distinct parts.
    - FOREACH: Performs updating actions once per element in a list.

https://neo4j.com/docs/cypher-manual
https://neo4j.com/docs/cypher-refcard

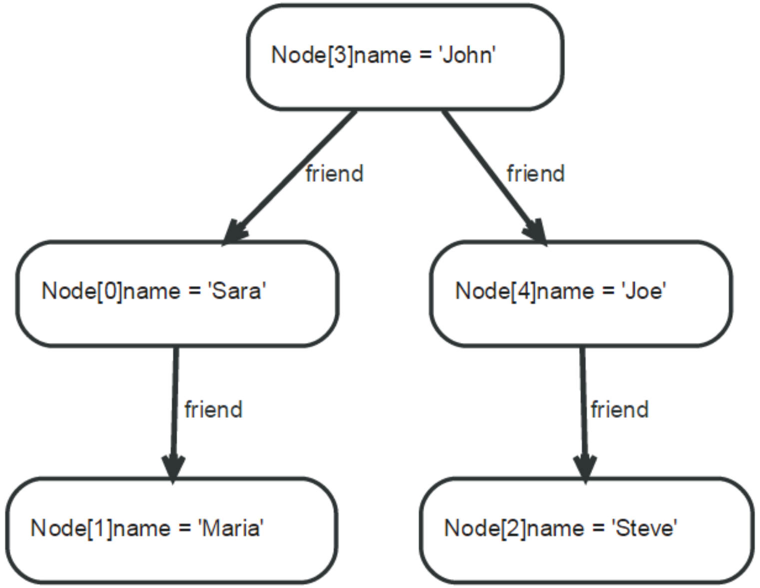
Cypher – powerful and expressive language




```
MATCH (:Person {name:"Dan"})-[:FRIEND]->(:Person {name:"Ann"})
```

Label Property Label Property

Cypher




```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john, fof
```

john	fof
Node[3]{name:"John"}	Node[1]{name:"Maria"}
Node[3]{name:"John"}	Node[2]{name:"Steve"}

2 rows

Cypher: Main clauses

- `MATCH` - allows you to specify the **patterns** Neo4j will search for in the database
- `WHERE` - adds **constraints** to the patterns in a MATCH, or **filters** the results of a WITH clause
- `RETURN` - defines **what to include** in the query result set
- `WITH` - allows query parts to be **chained together**, **piping** the results from one to be used as starting points or criteria in the next

Documentation: https://neo4j.com/docs/cypher-manual/current/clauses/

Cypher: Examples

- **MATCH** (n) **RETURN** n;

  Returns all the nodes in the graph

- **MATCH** (n:T) **RETURN** n;

  Returns all nodes with label of type T

- **MATCH** (n) **RETURN** n.p;

  Returns the property p of all the nodes in the graph

- **MATCH** (n)-[r:R1|R2]->(m) **RETURN** n,m;

  Returns all the pairs (n,m) of nodes that have a relationship of type R1 or R2 from n to m.

Cypher: Examples

```
MATCH (n)-[r]->() RETURN n, count(*);
```

Returns the number of nodes related to each node n

```
MATCH (n) RETURN n.name, count(*);
```

Groups n by name and returns the number

```
MATCH (david {name: 'David'})--(otherPerson)-->()
WITH otherPerson, count(*) AS foaf
WHERE foaf > 1
RETURN otherPerson.name
```

Returns the name(s) of the person(s) connected to 'David' with more than one outgoing relationship

Aggregates: https://neo4j.com/docs/cypher-manual/current/functions/aggregating

Summary

- A graph database <u>is a database</u>
  - Schema-less, instance-oriented
  - Two kinds of constructs: nodes and edges
- Graph databases do not scale well
  - Since there is no schema, edges are implemented as pointers and thus affected by data distribution
    - Algorithms to fragment graphs and distribute
- Many graph databases but Neo4j is the most popular (https://neo4j.com/)
  - Declarative language (Cypher) – **VERY interesting!**
  - Traversal Framework - procedural
  - Graph Data Science Library – graph algorithms
- Be aware of graph databases in the near future. They are the de-facto standard for Linked Data
  - Becoming more and more popular for Open Data

Bibliography

- Ian Robinson et al. Graph Databases. O'Reilly. 2013 (http://graphdatabases.com/)
- http://linkeddata.org/
- Resource Description Framework (RDF). W3C Recommendation. Latest version is available at http://www.w3.org/TR/rdf-concepts/
- OWL 2 Web Ontology Language (OWL). W3C Recommendation. Latest version is available at http://www.w3.org/TR/owl2-overview/