Data Engineering

LTAT.02.007

Ass Prof. Riccardo Tommasini

Assistants: Fabiano Spiga, Mohamed Ragab, Hassan Eldeeb

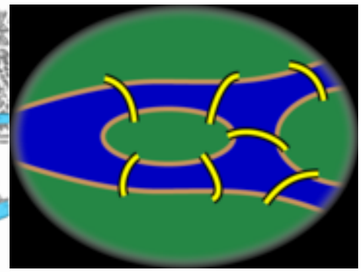
https://courses.cs.ut.ee/2020/dataeng

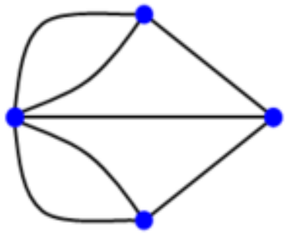
Forum

Moodle

History

Leonhard Euler's paper on "Seven Bridges of Königsberg", published in 1736.





Famous problems

- The traveling salesman problem: A traveling salesman is to visit a number of cities.
  - how to plan the trip so every city is visited once and just once and the whole trip is as short as possible ?
- Four color problem[100] : using only four colors, color any map of countries in such a way as to prevent two bordering countries from having the same color.
  - SOLVED ONLY 120 YEARS LATER!

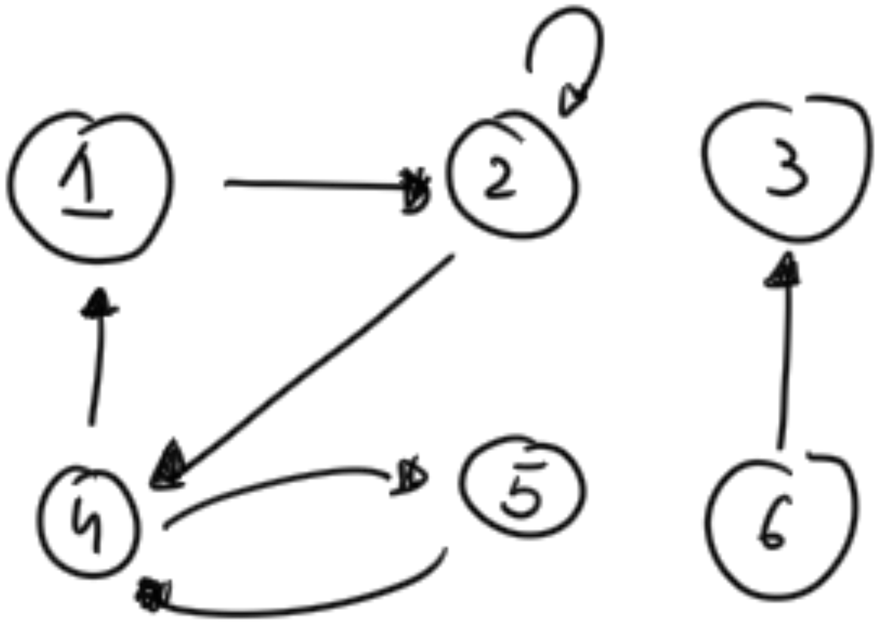
[100] Francis Guthrie,1852

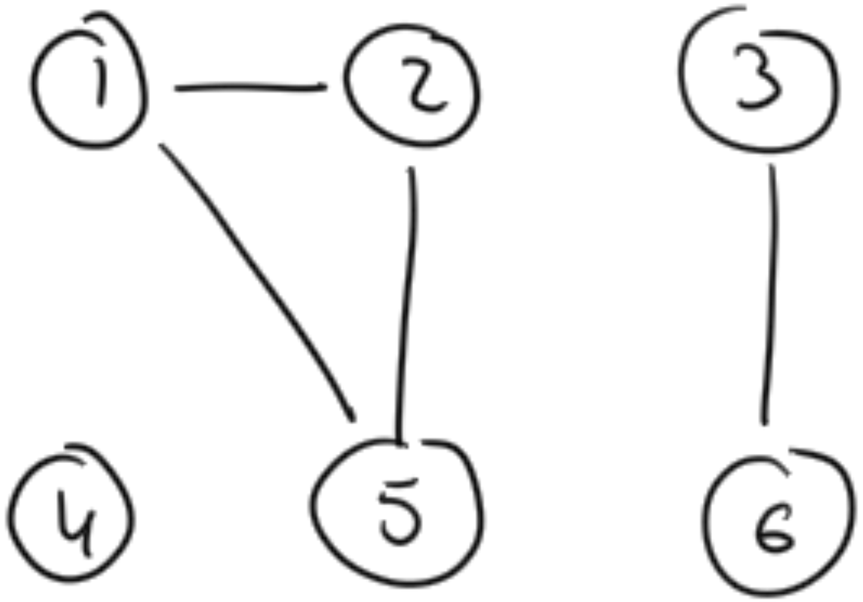
Other Examples of Graph Problems

- Cost of wiring electronic components
- Shortest route between two cities.
- Shortest distance between all pairs of cities in a road atlas.
- Matching / Resource Allocation
- Task scheduling
- Visibility / Coverage

What is a Graph?

Informally a *graph* is a set of nodes joined by a set of lines or arrows.





Graph

G is an ordered triple $G := (V, E, f)$

- V is a set of nodes, points, or vertices.
- E is a set, whose elements are known as edges or lines.
- f is a function
- maps each element of E
- to an unordered pair of vertices in V.

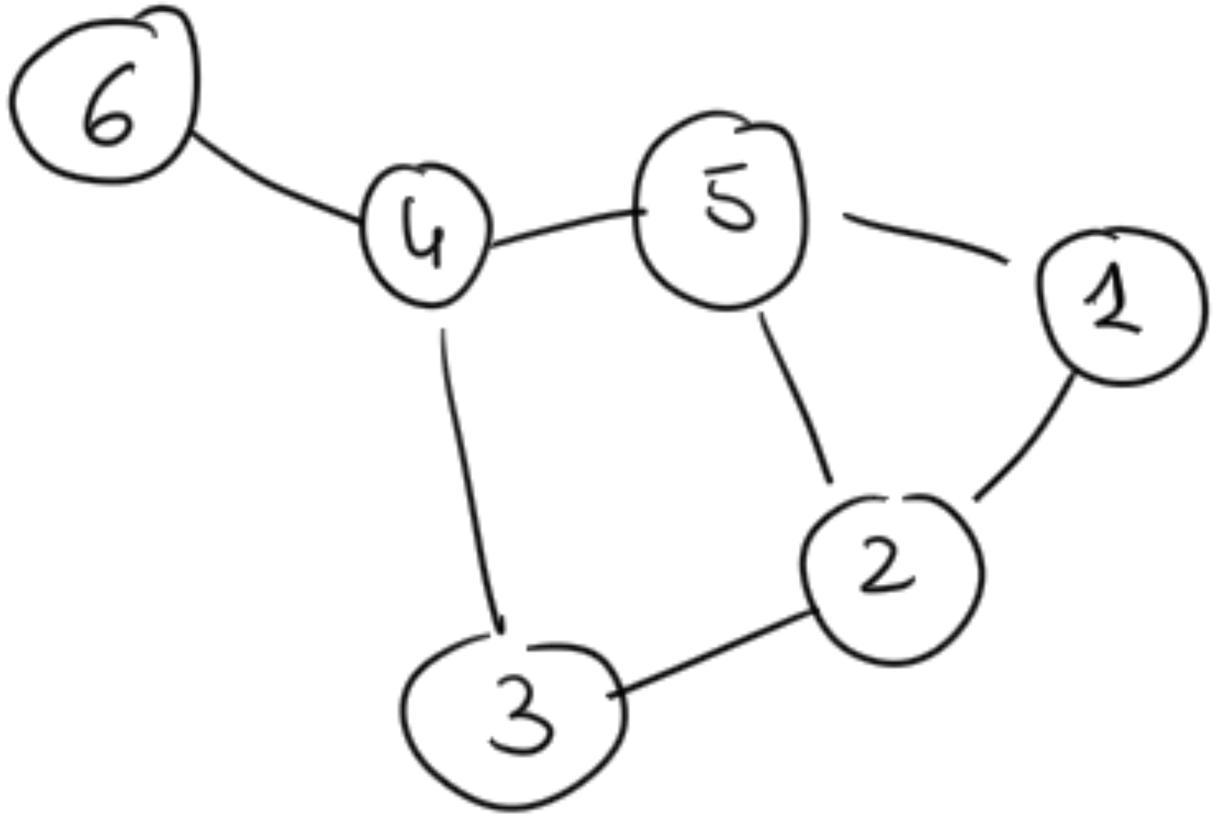
Vertexes and Edges

- A Vertex is a Basic Element
  - Drawn as a *node* or a *dot* .
  - The *Vertex set of a graph* *G is usually denoted by *V*
- An edge is *set* of two elements
  - Drawn as a line connecting two vertices, called end vertices, or endpoints.
  - The edge set of G is usually denoted by E(G), or E.

Example

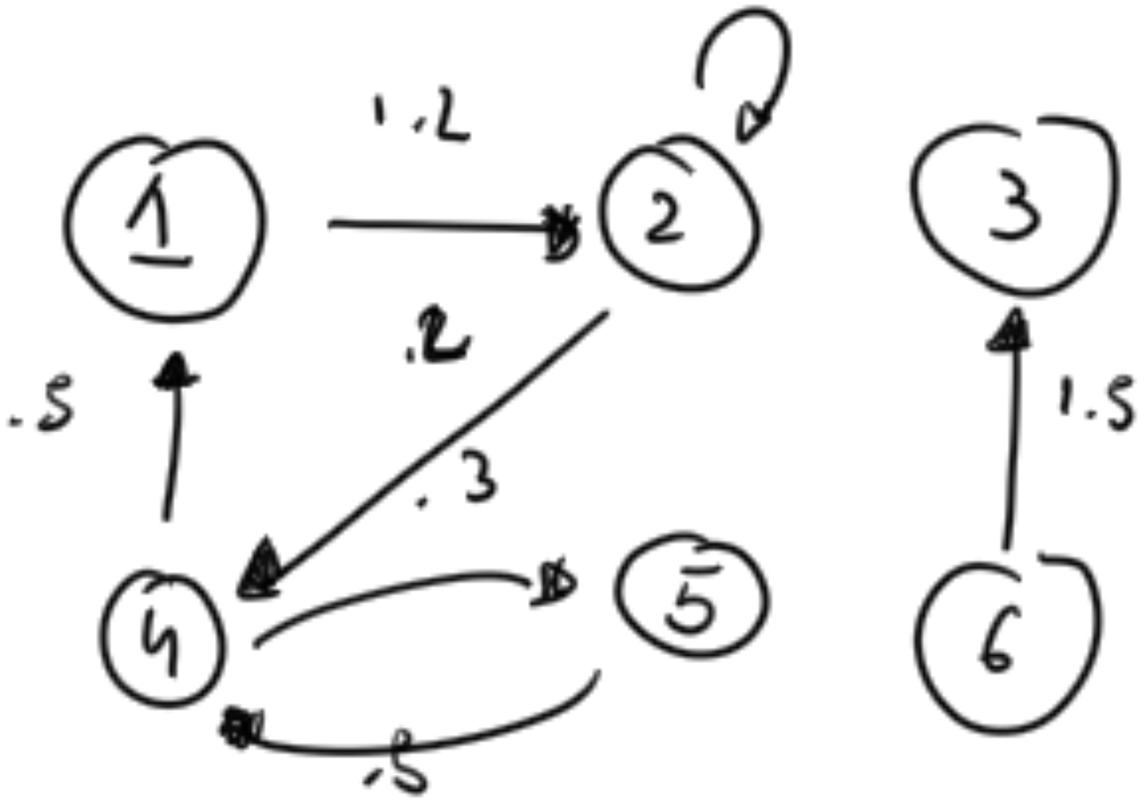
V:={1,2,3,4,5,6}

E:={{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}



Directed Graph (digraph)

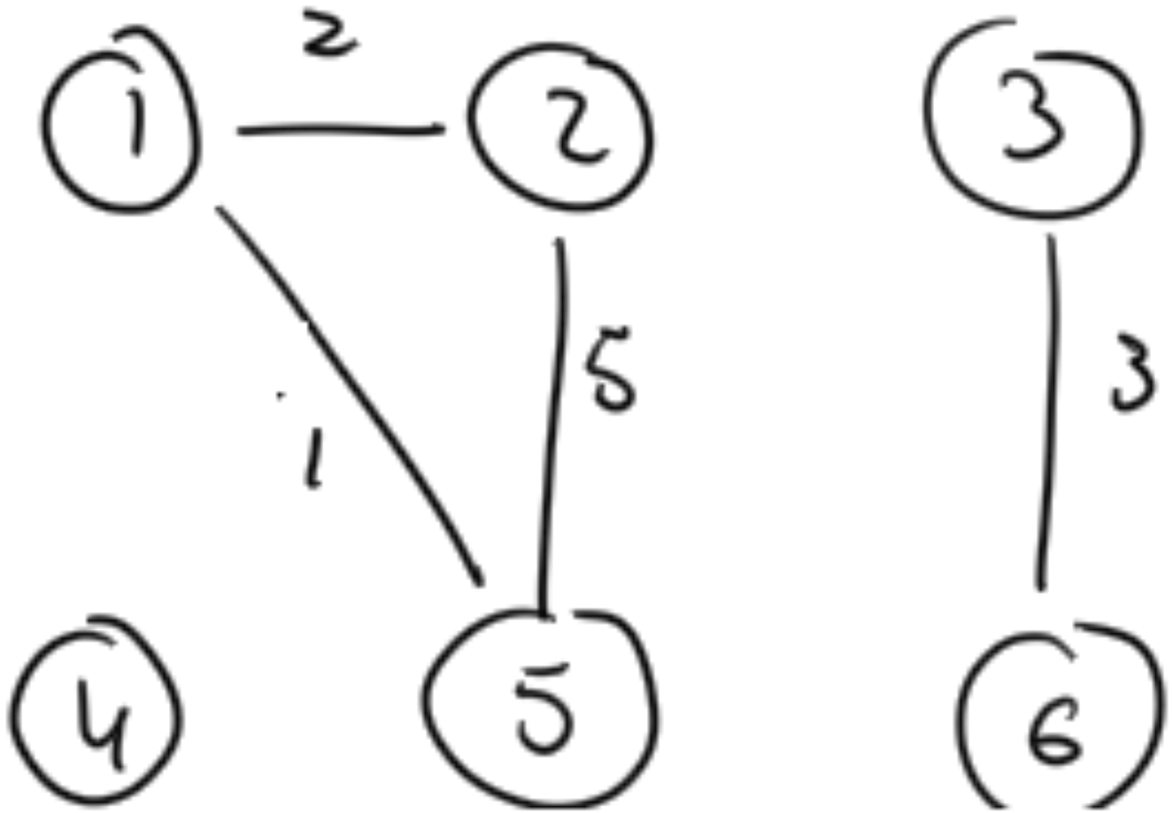
Edges have directions, i.e. an edge is an *ordered* pair of nodes



Weighted graphs

are graphs for which each edge has an associated *weight*, usually given by a _weight function

$$f_w : E \to R\,.$$

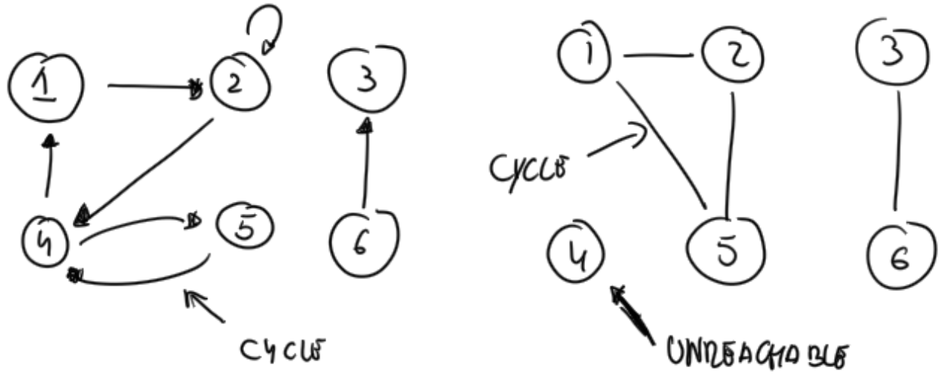


Path

A *path* is a sequence of vertices such that there is an edge from each vertex to its successor.

- A path is *simple* if each vertex is distinct.
- If there is path *p* from *u* to *v* then we say *v* is **reachable** from *u* via *p* .

Example: Simple path from 1 to 5= [1, 2, 4, 5]



Cycle

- A path from a vertex to itself is called a *cycle* .
- A graph is called *cyclic* if it contains a cycle;
  - otherwise it is called *acyclic*

Connectivity

- A graph is *connected* if
  - you can get from any node to any other by following a sequence of edges OR
  - any two nodes are connected by a path.
- A directed graph is *strongly connected* if there is a directed path from any node to any other node.

Sparsity/Density

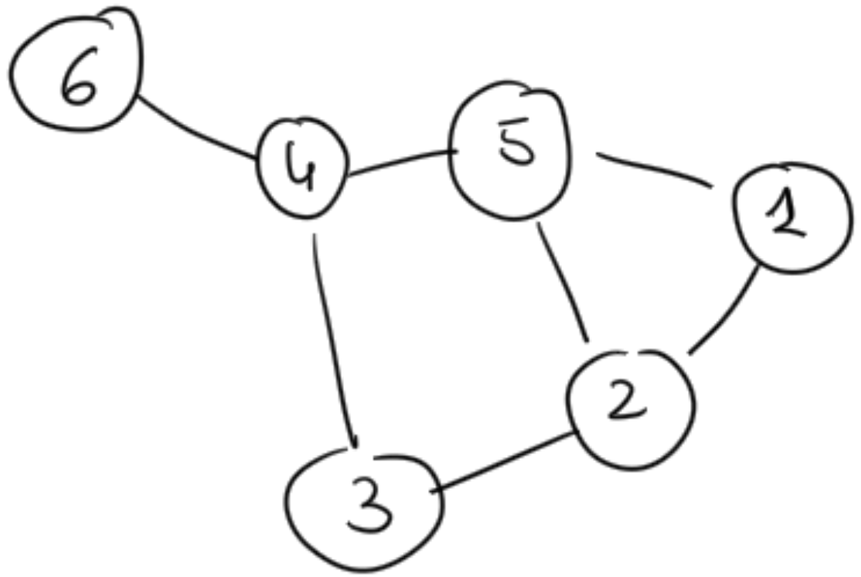
A graph is *sparse* if $|E| \approx |V|$

A graph is *dense* if $|E| \approx |V^2|$

Degree

Number of edges incident on a node

E.g., the degree of **5** is 3.



Degree (Directed Graphs)

In degree: Number of edges entering

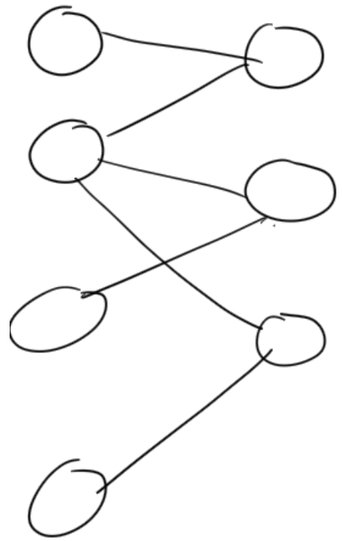
Out degree: Number of edges leaving

Degree =indegree+outdegree

Graph Types

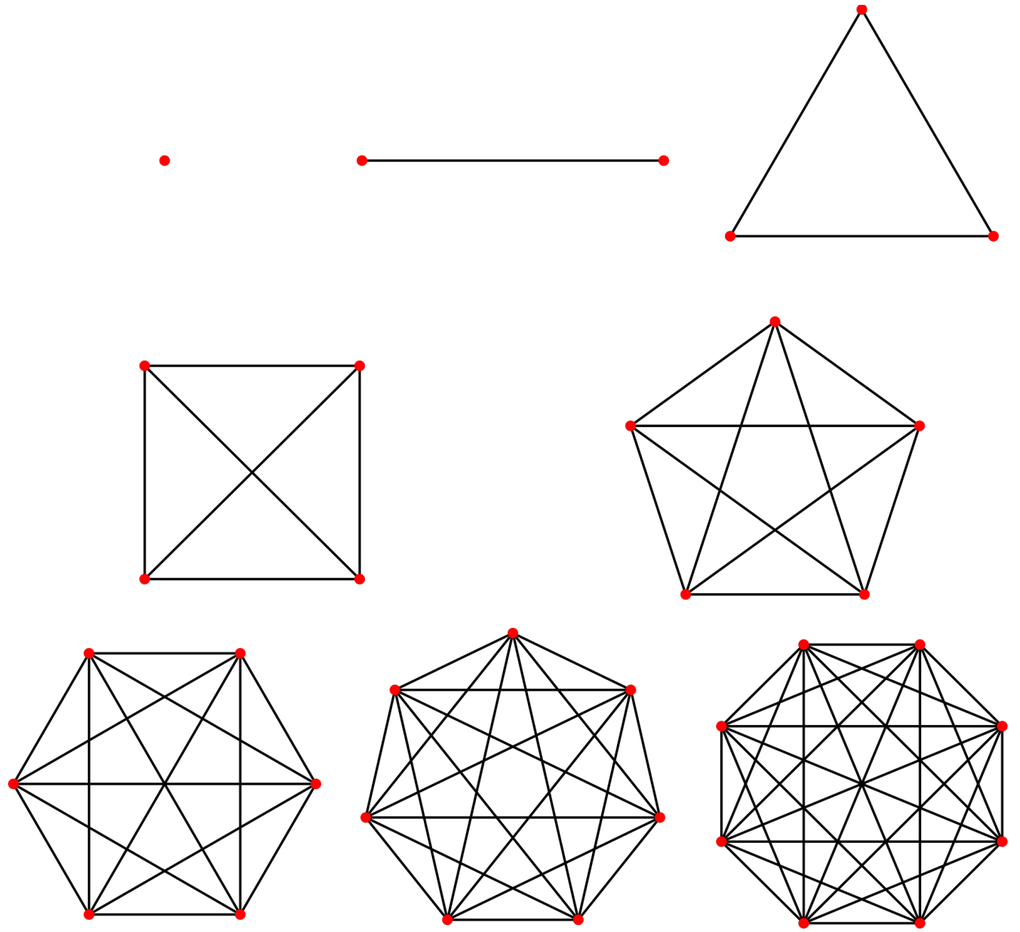
Bipartite graph

- *V* can be partitioned into 2 sets V_1 and V_2 such that $(u, v) \in E$ implies
  - either $u \in V_1$ and $v \in V_2$
  - or $^*v \in V_1$ and $u \in V_2$



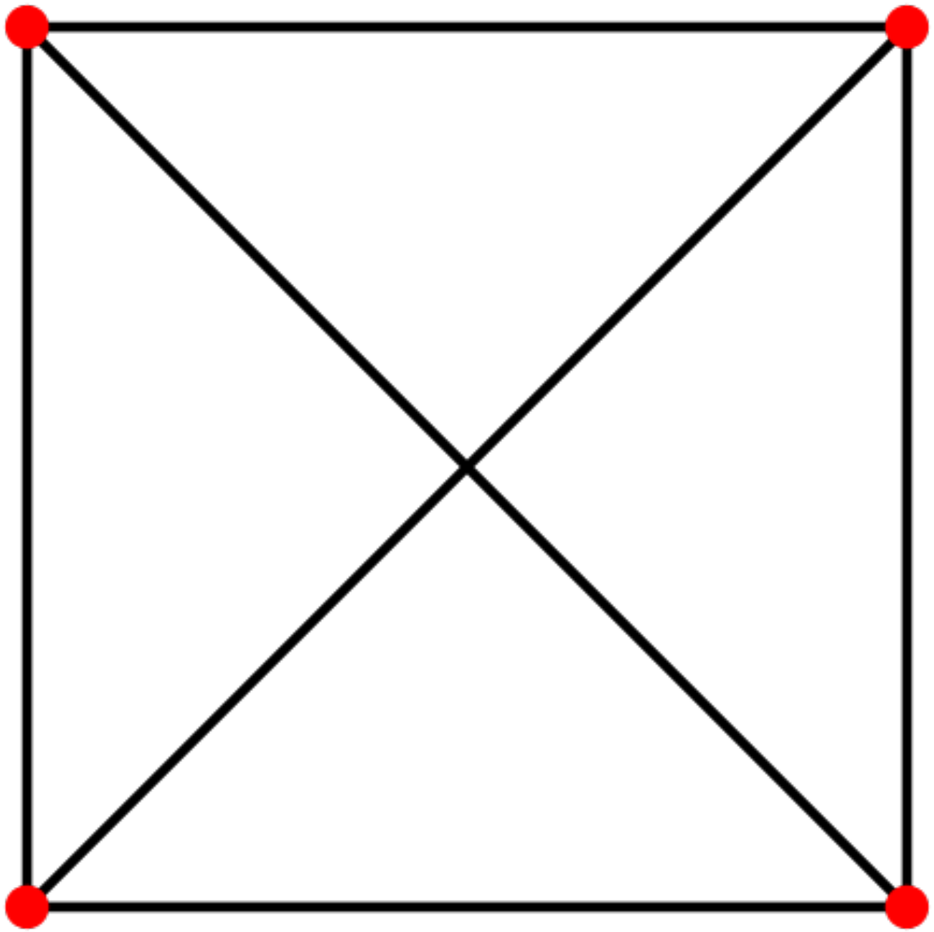
Complete Graph

- Denoted K_n
- Every pair of vertices are adjacent
- Has n(n-1) edges



Planar Graph

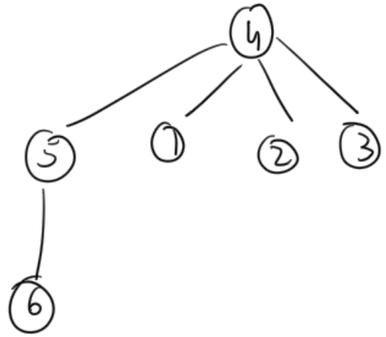
- Can be drawn on a plane such that no two edges intersect
- K_4 is the largest complete graph that is planar



Tree

Connected Acyclic Graph

Two nodes have *exactly* one path between them



Hypergraph

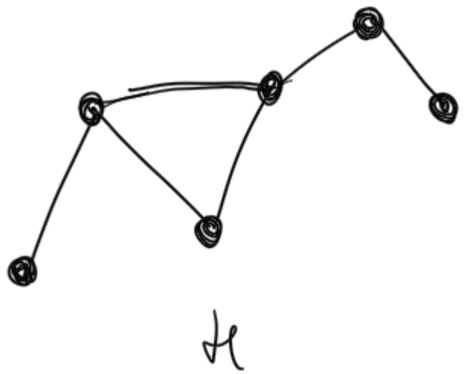
- Generalization of a graph,
  - edges can connect any number of vertices.
- Formally, an hypergraph is a pair (X,E) where
  - X is a set of elements, called nodes or vertices, and
  - E is a set of subsets of X, called hyperedges.
- Hyperedges are arbitrary sets of nodes,
  - contain an arbitrary number of nodes.

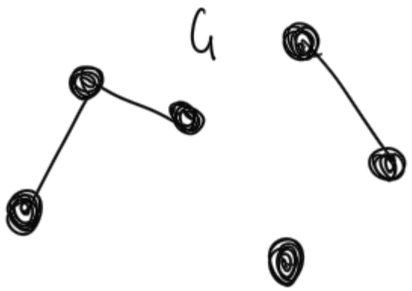
Subgraph

- Vertex and edge sets are subsets of those of G
  - a *supergraph* of a graph G is a graph that contains G as a subgraph.

Spanning subgraph

- Subgraph G has the same vertex set as H.
  - Possibly not all the edges
  - "G spans H".





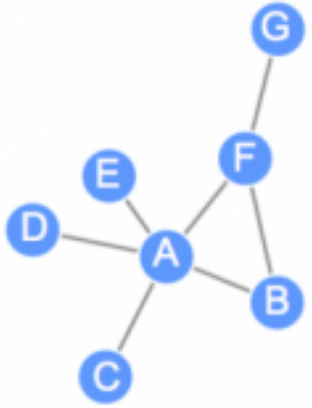
Graph ADT

- In computer science, a graph is an abstract data type (ADT)
- that consists of
  - a set of nodes and
  - a set of edges
  - establish relationships (connections) between the nodes.
- The graph ADT follows directly from the graph concept from mathematics.

Representation (Matrix)

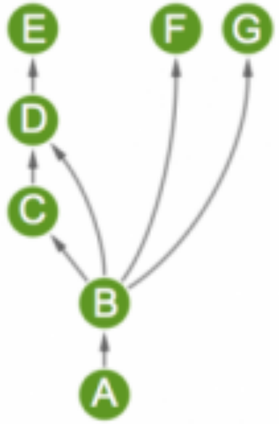
- Incidence Matrix
  - E x V
  - [edge, vertex] contains the edge's data
- Adjacency Matrix
  - V x V
  - Boolean values (adjacent or not)
  - Or Edge Weights

Undirected



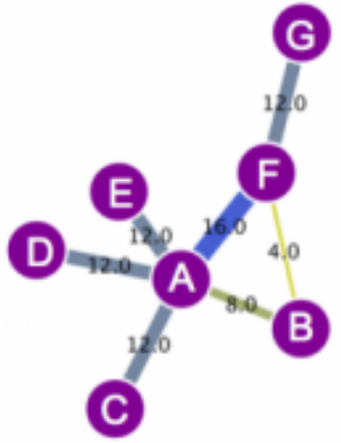
	A	B	C	D	E	F	G	Degree
A	0	1	1	1	1	1	0	5
B	1	0	0	0	0	1	0	2
C	1	0	0	0	0	0	0	1
D	1	0	0	0	0	0	0	1
E	1	0	0	0	0	0	0	1
F	1	1	0	0	0	0	1	3
G	0	0	0	0	0	1	0	1

Directed



	A	B	C	D	E	F	G	Out-degree
A	0	1	0	0	0	0	0	1
B	0	0	1	1	0	1	1	4
C	0	0	0	1	0	0	0	1
D	0	0	0	0	1	0	0	1
E	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0

Weighted



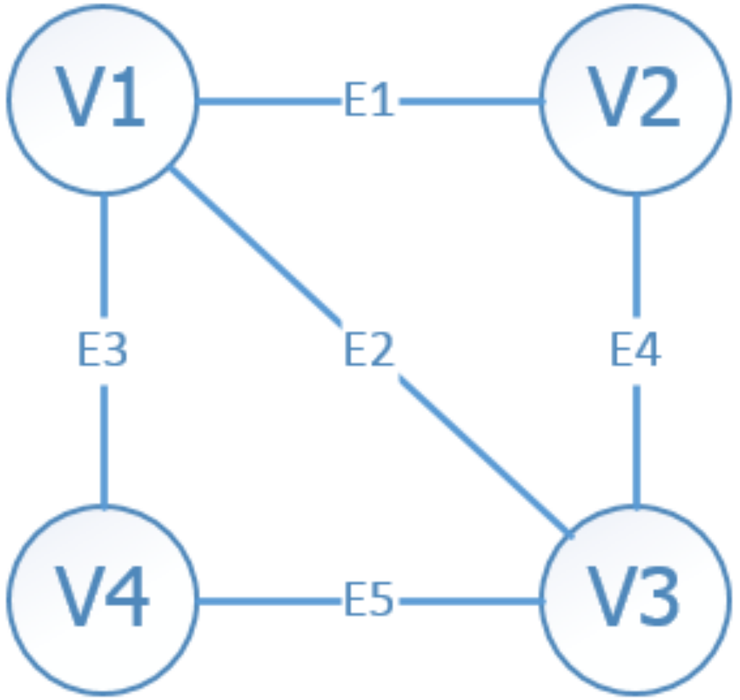
	A	B	C	D	E	F	G	Degree
A	0	8	12	12	12	16	12	72
B	8	0	0	0	0	4	0	12
C	12	0	0	0	0	0	0	12
D	12	0	0	0	0	0	0	12
E	12	0	0	0	0	0	0	12
F	16	4	0	0	0	0	12	32
G	12	0	0	0	0	12	0	24

Adjacency matrices

Representation (List)

- Edge List
  - pairs (ordered if directed) of vertices
  - Optionally weight and other data
- Adjacency List

Undirected Graph



Edge List

[[0,1],[0,2],[0,3],[1,2],[3,2]]

Adjacency Matrix

X	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

Adjacency List

[[1,2,3],
[0,2],
[0,1,3],
[0,2]]

Graph Algorithms

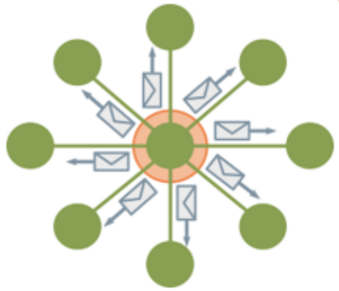
- Shortest Path
  - Single Source
  - All pairs (Ex. Floyd Warshall)
- Network Flow
- Matching
  - Bipartite
  - Weighted
- Topological Ordering
- Strongly Connected
- Biconnected Component / Articulation Point
- Bridge
- Graph Coloring
- Euler Tour
- Hamiltonian Tour
- Clique
- **Isomorphism**
- Edge Cover
- Vertex Cover
- Visibility

Graph Technologies

Graph Databases

- Technologies used primarily for transactional online graph persistence, typically accessed directly in real time from an application.
- They are the equivalent of “normal” online transactional processing (OLTP) databases in the relational world.

Offline Graph Analytics



- Technologies used primarily for offline graph analytics, typically performed as a series of batch steps.
- These technologies can be called graph compute engines.
- Used for analysis of data in bulk, such as data mining and online analytical processing (OLAP).

Graph Databases

Back to One Machine

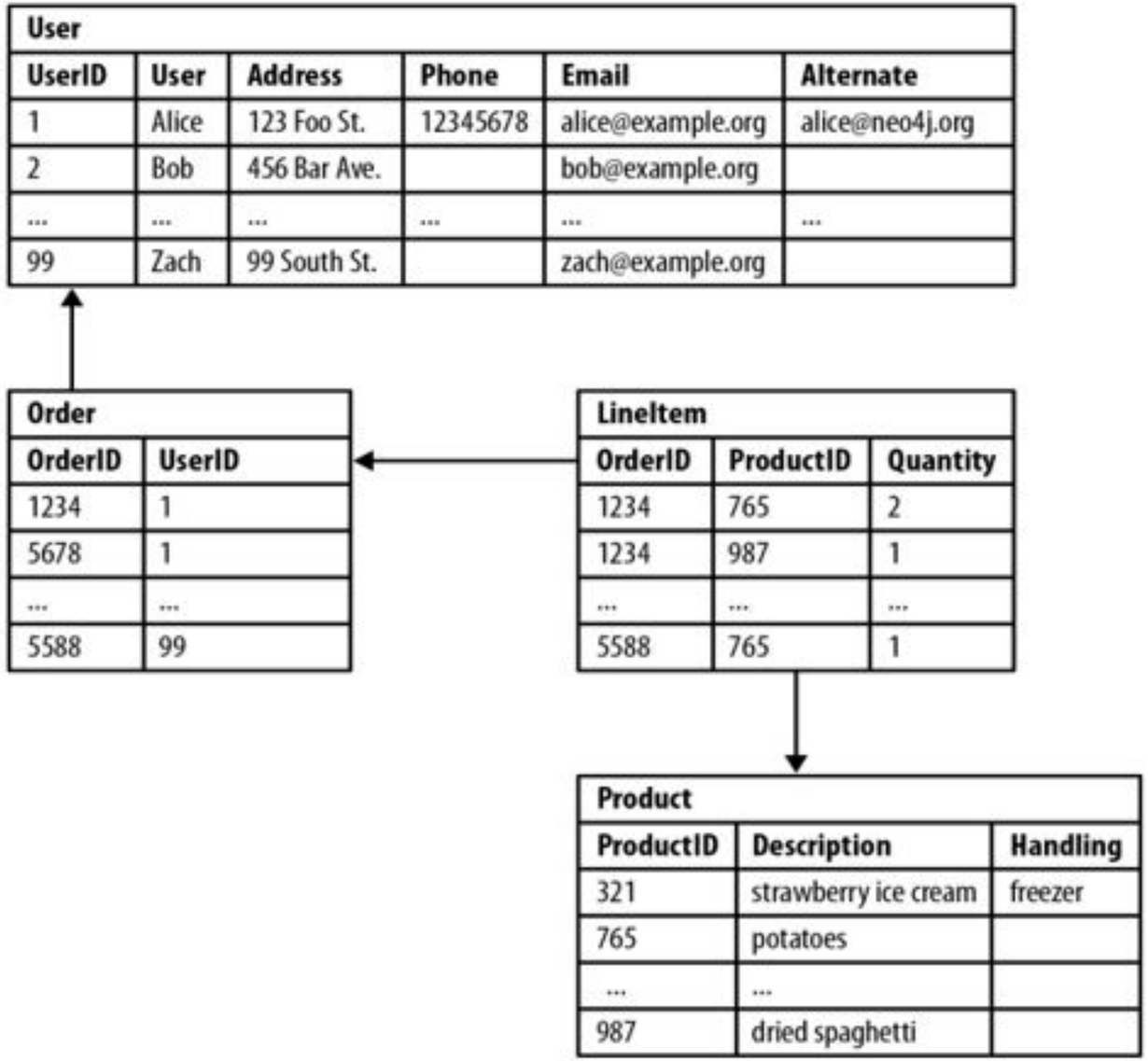
- Graph Databases are tailored for OLTP workloads.
- Typically, you are interested in selecting the subset of your graph based on a condition and then operate on that.
- Most of them work in a centralized fashion

The Case of Graph OLAP

- OLAP queries over the entire graph will not be so efficient (why?)
- Graph OLAP algorithms are ofte **iterative**, and need to process the whole graph.
- Hard to Scale out because graph are hard to partition
- If you're interested join our Spring courses LTAT.02.003 and LTAT.02.010

Graph DBs VS. RDBMSs

- RDBs are well fitted to find generice queries, thanks to the internal structure of the tables.
- Aggregations over a complete dataset are "easy".
- However, Relational databases struggle with highly connected domains.



User

UserID	User	Address	Phone	Email	Alternate
1	Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
2	Bob	456 Bar Ave.		bob@example.org	
...	...	...	...	...	...
99	Zach	99 South St.		zach@example.org	

Order

OrderID	UserID
1234	1
5678	1
...	...
5588	99

LineItem

OrderID	ProductID	Quantity
1234	765	2
1234	987	1
...	...	...
5588	765	1

Product

ProductID	Description	Handling
321	strawberry ice cream	freezer
765	potatoes	
...	...	
987	dried spaghetti	

Performance

In relational databases, the performance of join-intensive queries deteriorates as the dataset gets bigger.

On the other hand, graph database performance tends to remain relatively constant, even as the dataset grows.

(Nope, indexes)

Clarke's Third Law:
Any sufficiently advanced technology is indistinguishable from magic.

Agility

Despite their names though, relational databases are less suited for exploring relationships. Thus, the complexity is pushed on the query language.

In graph databses, relationships are first-class moreover, they have no schema. Thus, API and query language are much simpler and agile.

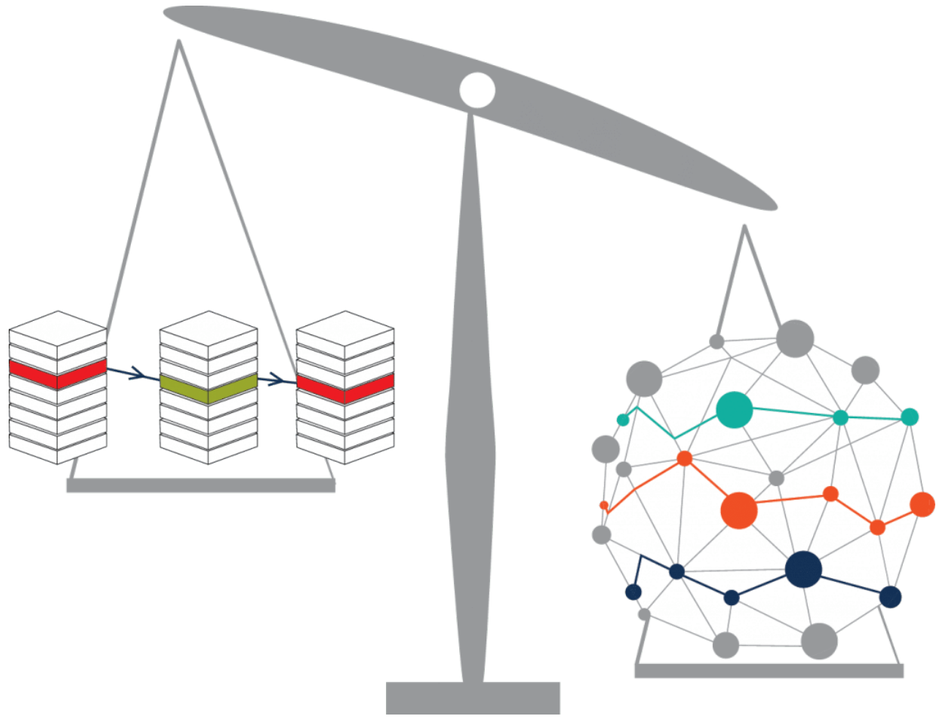
Flexibility

Changing schemas in Relational Databases may break queries and store procedures or require to change the integrity constraints.

Graphs are naturally additive, we can add new relationships or nodes without disturbing existing queries and application functionality.

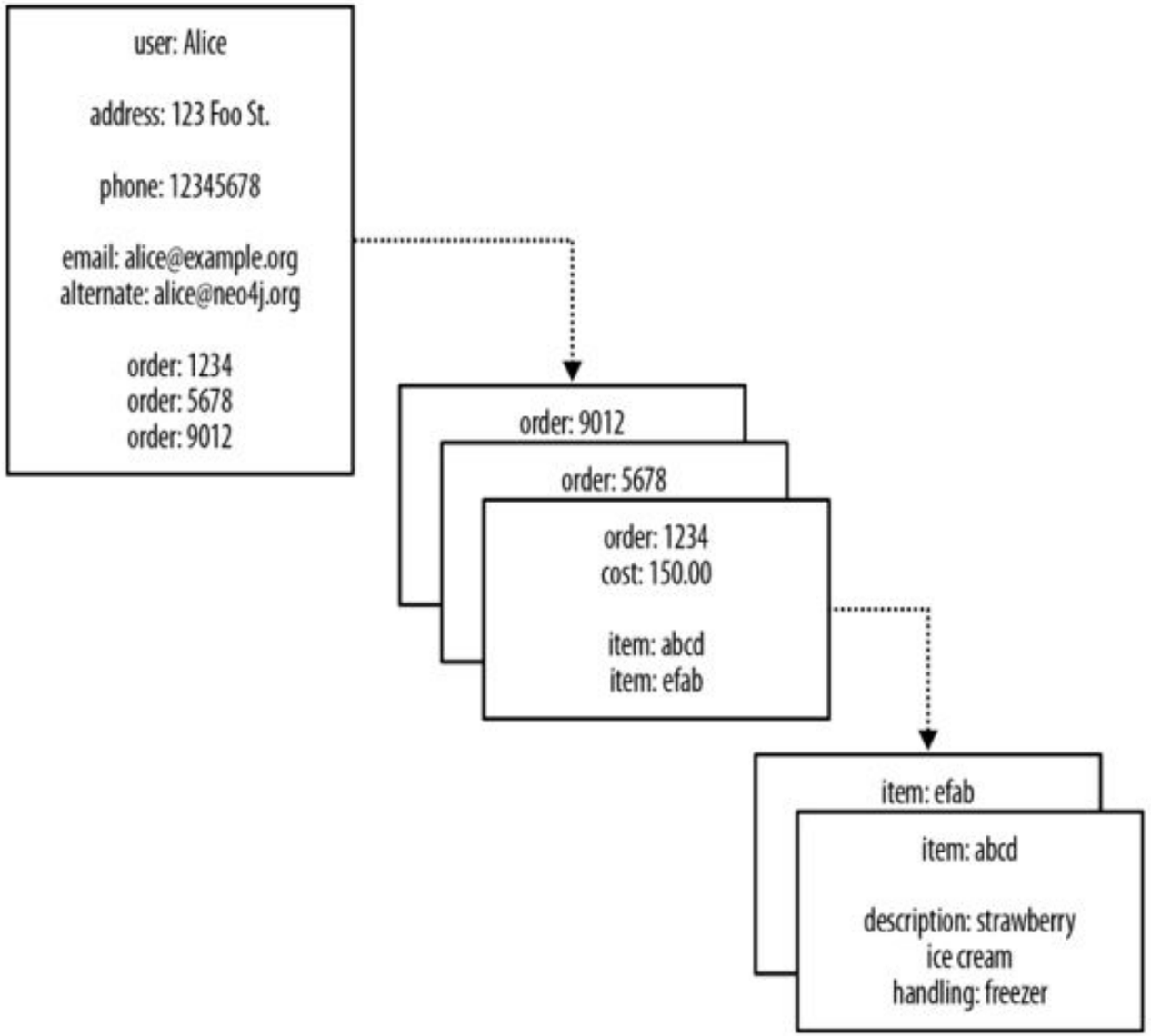
Graph DBs VS. NoSQL

- Are RelationalDB NoSQL?
  - In principles, yes. However they do not target OLAP...



Nosql also Lacks Relationships

- Most NOSQL databases whether key-value, document, or column oriented store sets of disconnected documents/ values/columns.
- This makes it difficult to use them for connected data and graphs.
- One well-known strategy for adding relationships to such stores is to embed an aggregate's identifier inside the field belonging to another aggregate.

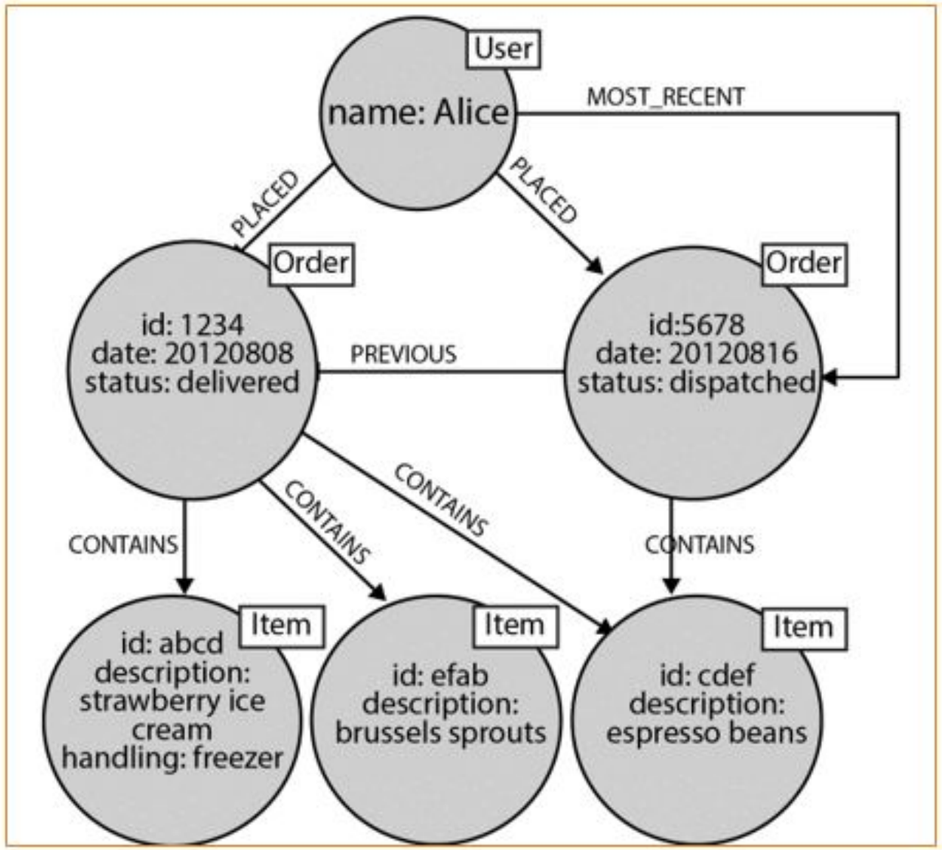


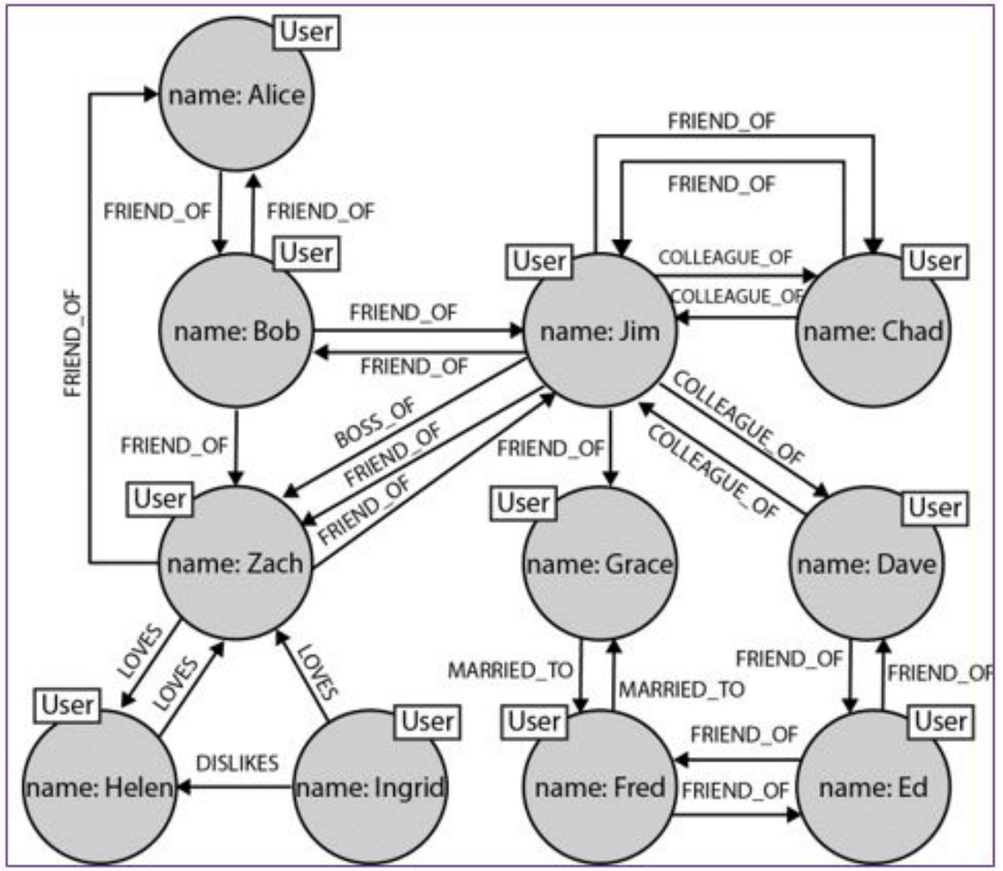

OH NO!
ANYWAY

Nosql also Lacks Relationships

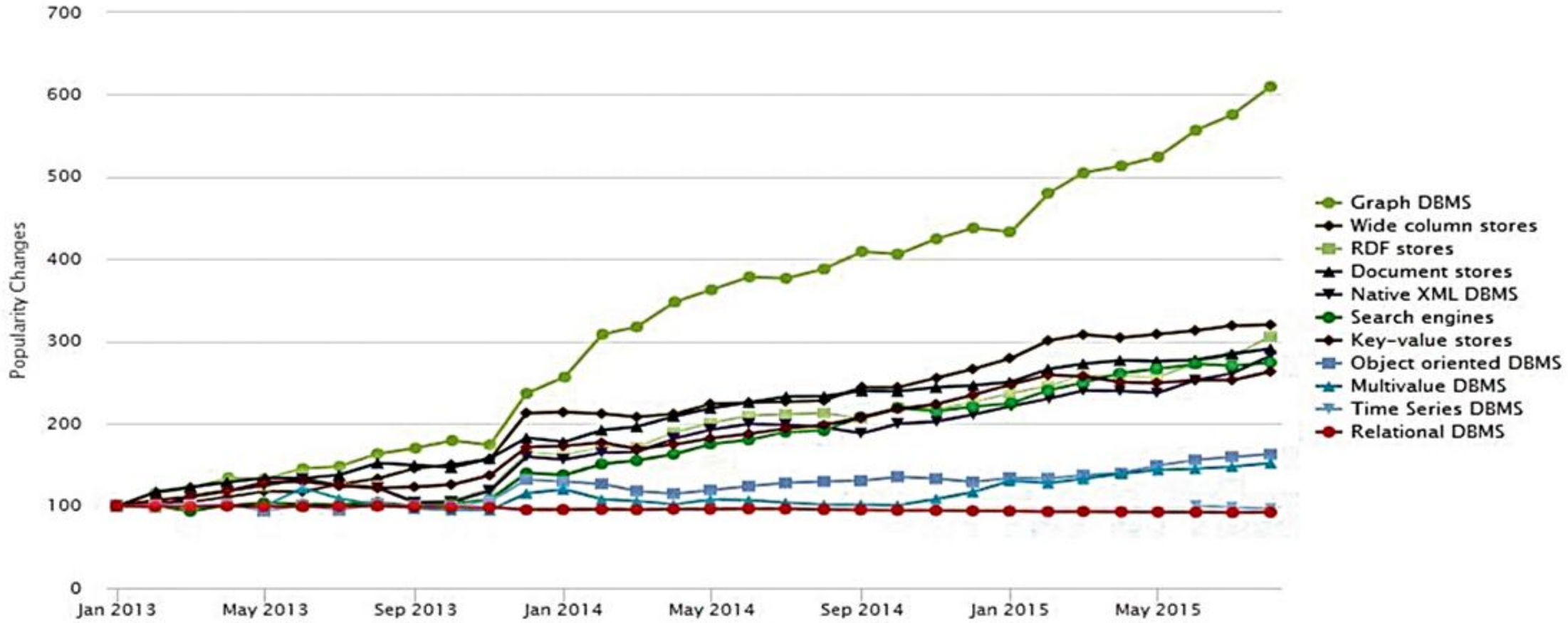
- We can **join aggregates** at the application level
  - Seeing a reference to order: 1234 in the record beginning user: Alice, we infer a connection between user: Alice and order: 1234.
- Because there are no identifiers that "point" backward (the foreign aggregate "links" are not reflexive.
  - How to answer: *Who customers that bought a particular product?*
- Aggregates quickly becomes prohibitively expensive.

Graph DBs embrace Relationships

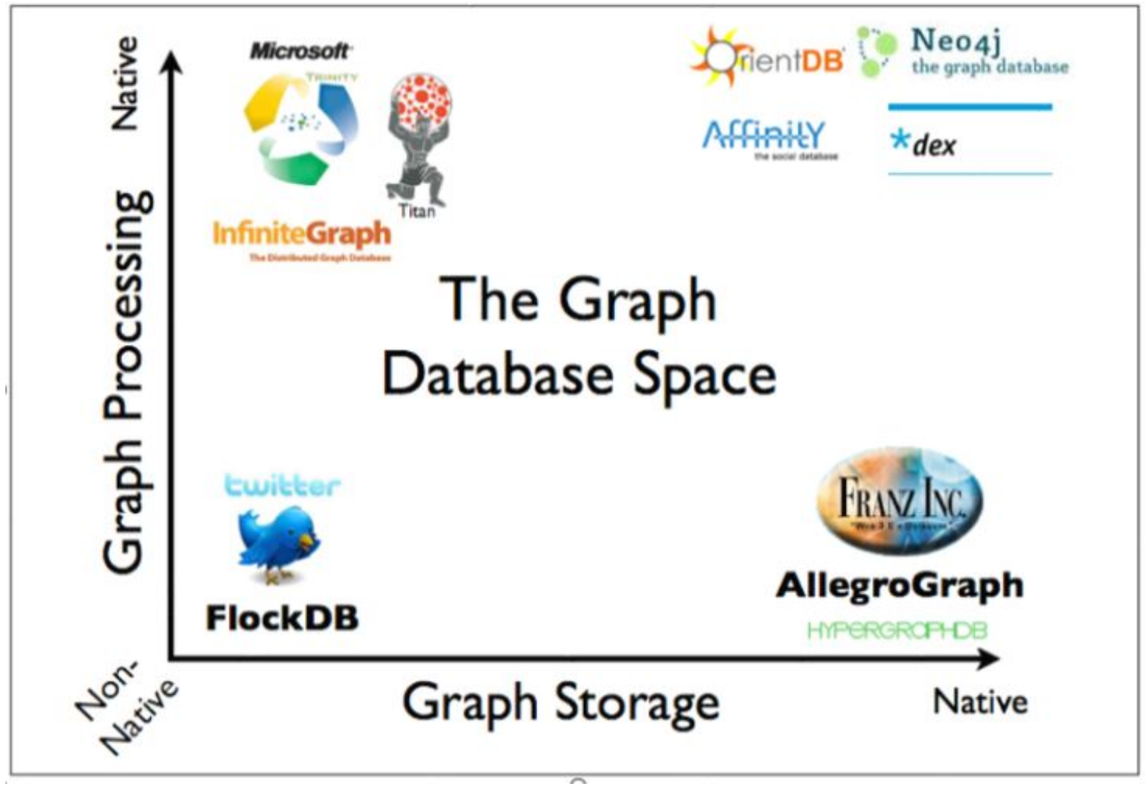

User
name: Alice
MOST_RECENT
PLACED
PLACED
Order
id: 1234
date: 20120808
status: delivered
PREVIOUS
Order
id:5678
date: 20120816
status: dispatched
CONTAINS
CONTAINS
CONTAINS
CONTAINS
Item
id: abcd
description:
strawberry ice
cream
handling: freezer
Item
id: efab
description:
brussels sprouts
Item
id: cdef
description:
espresso beans


User
name: Alice
FRIEND_OF
FRIEND_OF
FRIEND_OF
User
name: Bob
FRIEND_OF
FRIEND_OF
User
name: Jim
FRIEND_OF
FRIEND_OF
COLLEAGUE_OF
COLLEAGUE_OF
User
name: Chad
FRIEND_OF
BOSS_OF
FRIEND_OF
FRIEND_OF
FRIEND_OF
COLLEAGUE_OF
COLLEAGUE_OF
User
name: Zach
User
name: Grace
User
name: Dave
LOVES
LOVES
LOVES
MARRIED_TO
MARRIED_TO
FRIEND_OF
FRIEND_OF
User
name: Helen
DISLIKES
User
name: Ingrid
User
name: Fred
FRIEND_OF
FRIEND_OF
User
name: Ed

Popularity of Graph DBs



Which one to choose?![111]



[111] Ian Robinson, Jim Webber, and Emil Eifrem. 2013. Graph Databases. O'Reilly Media, Inc.

Graph Storage and Processing

- **Native Graph Storage** benefits traversal performance at the expense of making some queries that don't use traversals difficult or memory intensive.
- **Non-Native graph storage**, e.g., usuing a relational backend, is purpose-built stack and can be engineered for performance and scalability.

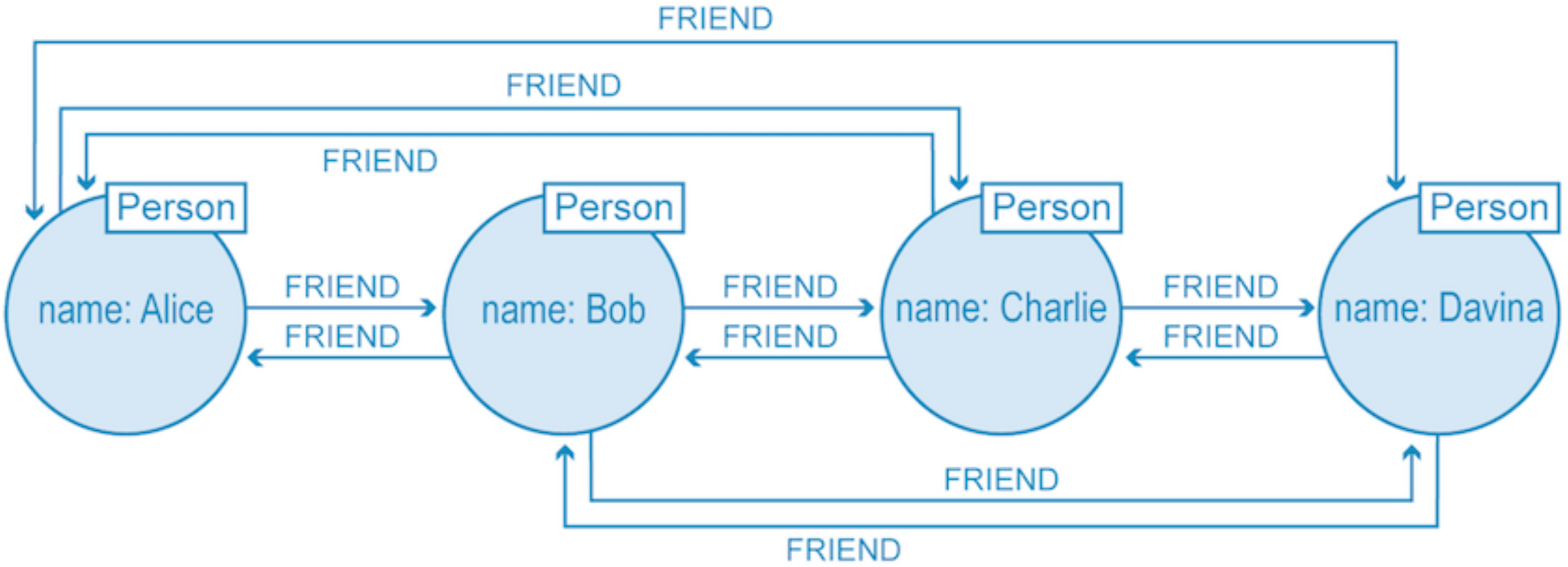
Native Graph Processing

A graph database has native processing capabilities if it uses index-free adjacency.

A node directly references its adjacent nodes, acting as a micro-index for all nearby nodes.

With index-free adjacency, bidirectional joins are effectively precomputed and stored in the database as relationships[1140].

[1140] It is cheaper and more efficient than doing the same task with indexes, because query times are proportional to the amount of the graph searched.


FRIEND
FRIEND
FRIEND
Person
name: Alice
FRIEND
FRIEND
Person
name: Bob
FRIEND
FRIEND
Person
name: Charlie
FRIEND
FRIEND
Person
name: Davina
FRIEND
FRIEND

Storage

Doubly Linked Lists in the Relationship Store

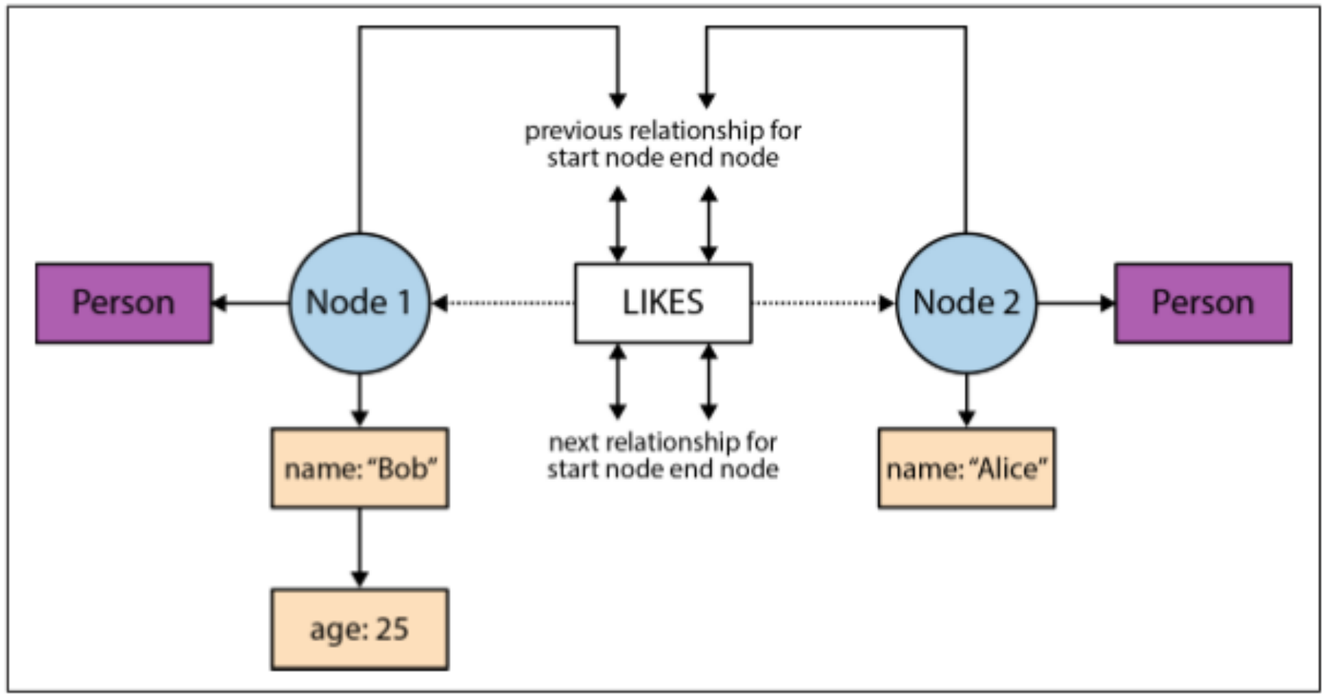
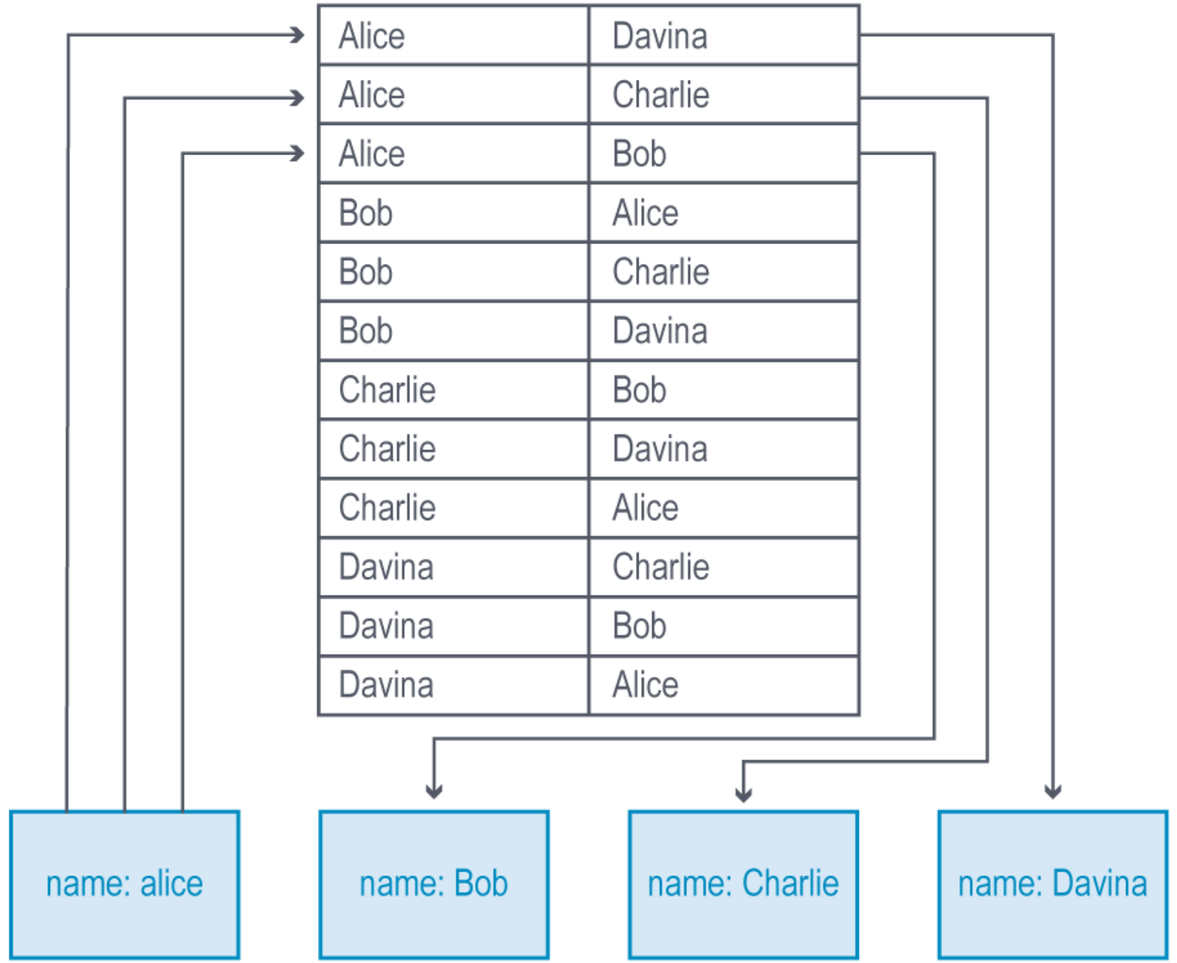


Figure 6-5. How a graph is physically stored in Neo4j

Non-native processing

- A nonnative graph database engine uses (global) indexes to link nodes together,
- Example:
  - To find Ali- ce's friends we have first to perform an index lookup, at cost O(log n).
  - If we wanted to find out who is friends with Alice, we would have to one lookup for each node that is potentially friends with Alice. This makes the cost O(m log n).



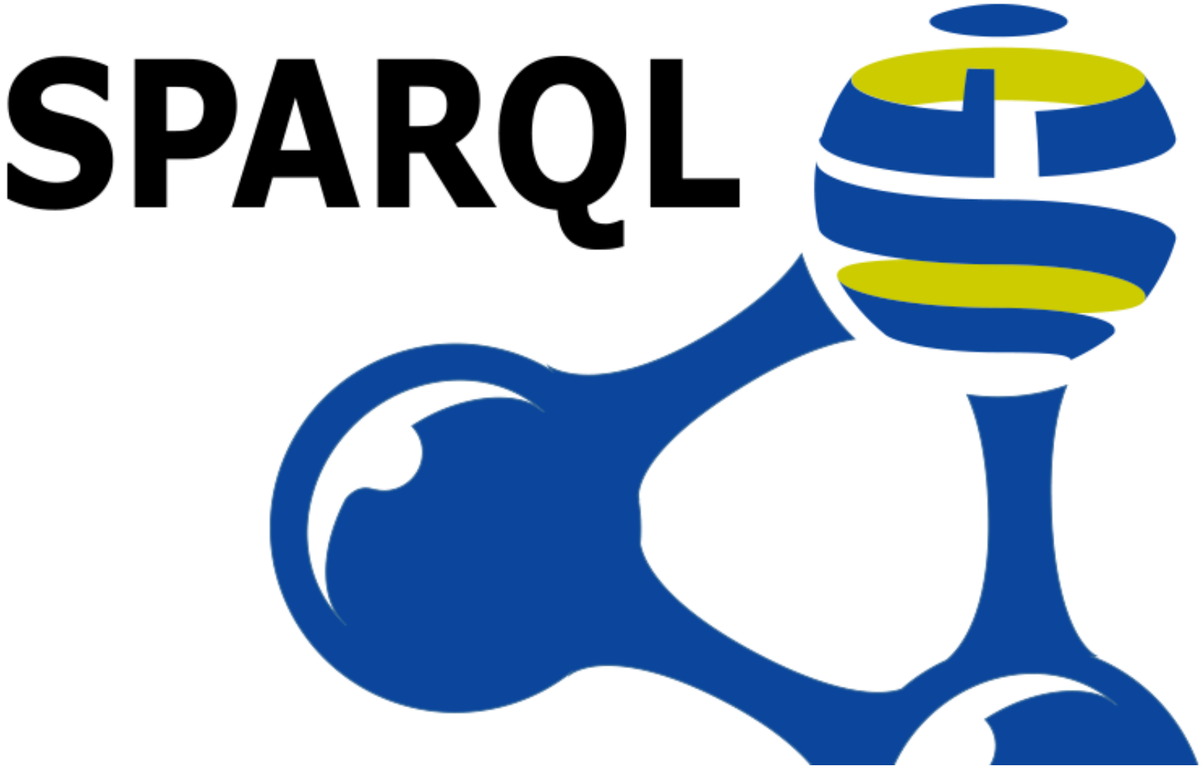
Neo4J Graph DB[112]

- It supports ACID transactions
- It implements a Property Graph Model efficiently down to the storage level.
- It is useful for single server deployments to query over medium sized graphs due to using memory caching and compact storage for the graph.
- Its implementation in Java also makes it widely usable.
- It provides master-worker clustering with cache sharding for enterprise deployment.
- It uses Cypher as a declarative query language.

[112] url

AllegroGraph Semantic Graph DB[114]

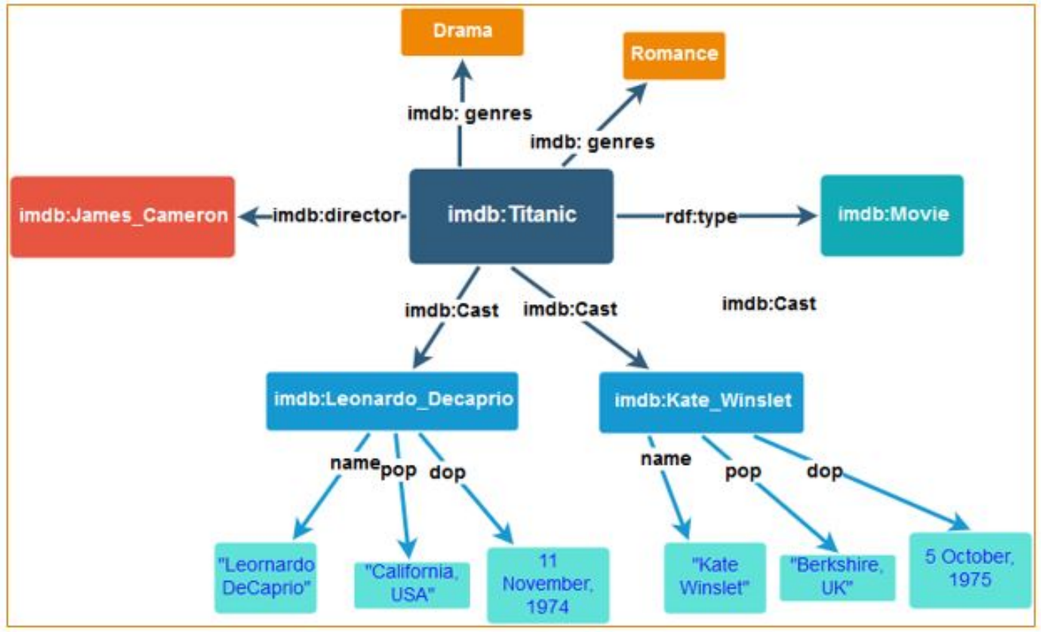
- AllegroGraph is a graph database and application framework for building Semantic Web applications.
- It can store data and meta-data as triples.
- It can query these triples through various query APIs like SPARQL (the standard W3C query language).
- It supports RDFS++ as well as Prolog reasoning with its built-in reasoner.
- AllegroGraph includes support for Federation, Social Network Analysis, Geospatial capabilities and Temporal reasoning.

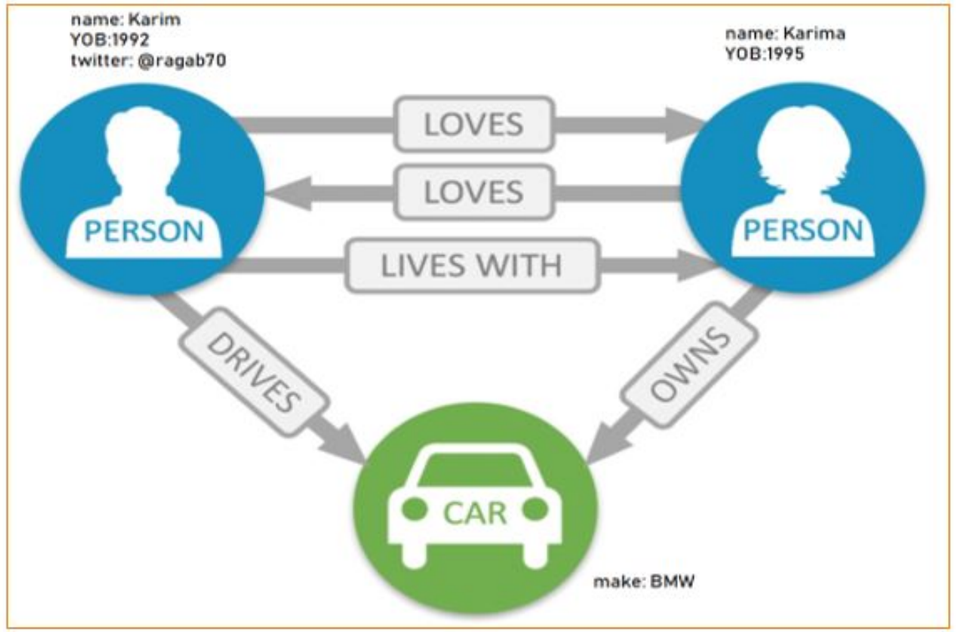


[114] https://franz.com/agraph/allegrograph/

Graph Data Models

- Two Popular Graph Data Models:
  - Edge-Labelled Graphs
  - Property Attributed Graphs

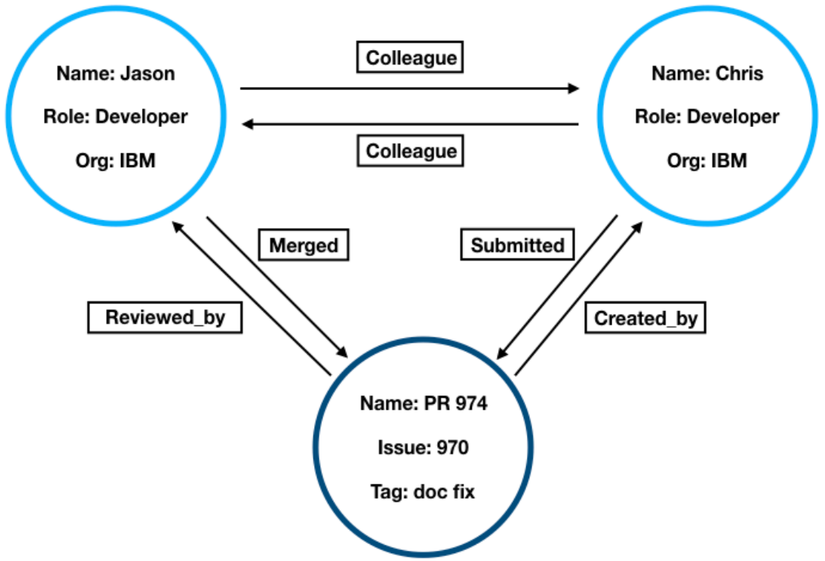




Property Graphs Vs. Edge-Labelled Graphs

- Edge-Labelled Graphs are widely adopted in practice. E.g. Resource Description Framework (RDF) (Figure in the previous slide).
- However, it is often cumbersome to add information about the edges to an edge-labelled graph.
- For example, if we wished to add the source of information, for example, that the acts-in relations were sourced from the web-site IMDb.
- Adding new types of information to edges in an edge-labelled graph may thus require a major change to the graph's structure, entailing a significant cost.

Property Graph Example



Variations of the Property Graph Data Model (PGM)

- **Direction.** A property graph is a directed graph; the PGM defines edges as ordered pairs of vertices.
- **Multi-graph.** A property graph is a multi-graph; the PGM allows multiple edges between a given pair of vertices.
- **Simple graphs** (in contrast to multi-graphs) additionally require to be injective (one-to-one).
- **Labels.** A property graph is a multi-labeled graph; the PGM allows vertices and edges to be tagged with zero or more labels.
- **Properties.** A property graph is a key-value-attributed graph; the PGM allows vertices and edges to be enriched with data in the form of key-value pairs.

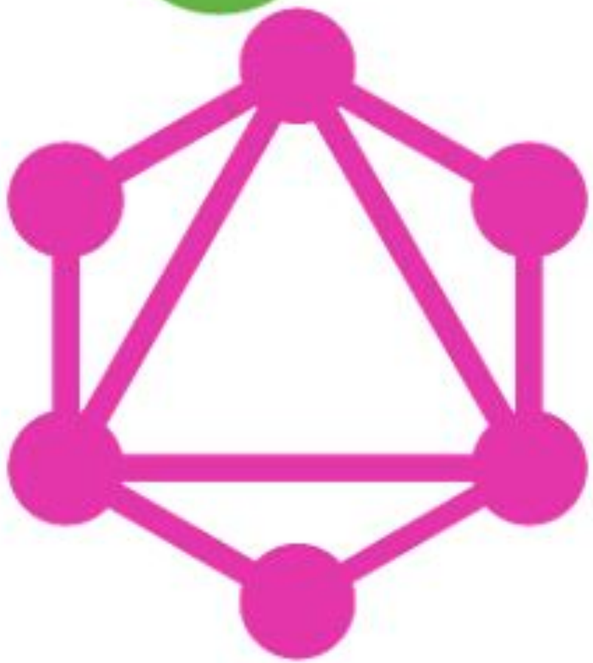
Graph Query Languages

How To Query Graph Databases!

- Although graphs can still be (and sometimes still are) stored in relational databases, the choice to use a graph database for certain domains has significant benefits in terms of querying.
- Where the emphasis shifts from joining various tables to specifying graph patterns and navigational patterns between nodes that may span arbitrary-length paths.
- A variety of graph database engines, graph data models, and graph query languages have been released over the past few years.
  - Examples of Graph DBs: Neo4j, OrientDB, AllegroGraph.
  - Graph data models: Property graphs, and edge labelled graphs and many other variations of them.
  - Different modern query languages also come to the scene such as Cypher, SPARQL, Gremlin and many more.

Graph Query Languages Core Features

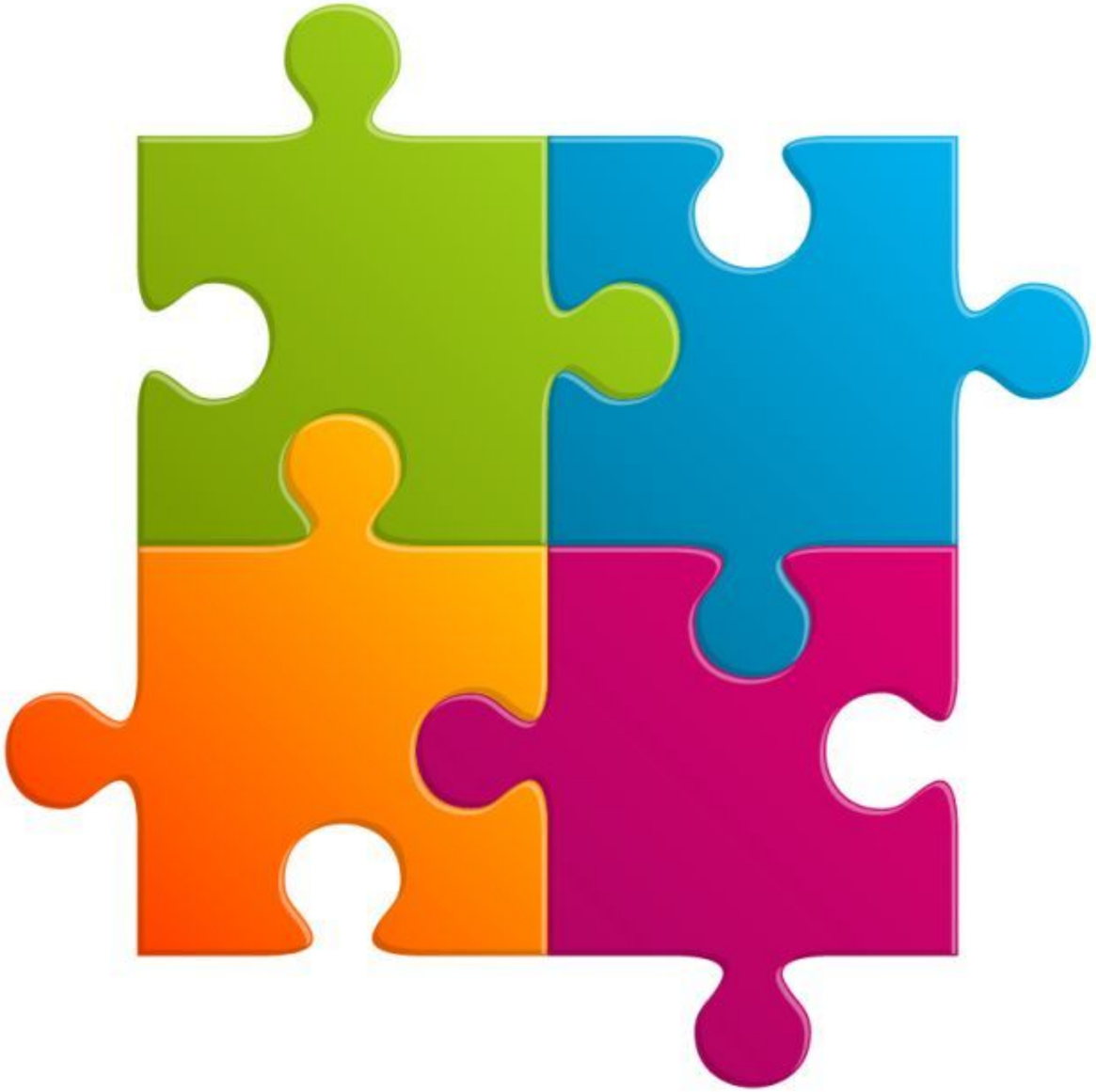
- Features:
  - Graph Patterns.
  - Navigational "Path" expressions.
  - Aggregation
  - Graph-to-Graph queries.
  - Path unwinding.
- Standardization:
  - (SPARQL/SPARQL 1.1) --- Yes
  - (Gremlin,G-Core,Gremlin,GraphQl,Cypher)--- No



Pattern Matching and Graph Navigation

Graph Pattern Matching VS. Graph Navigational

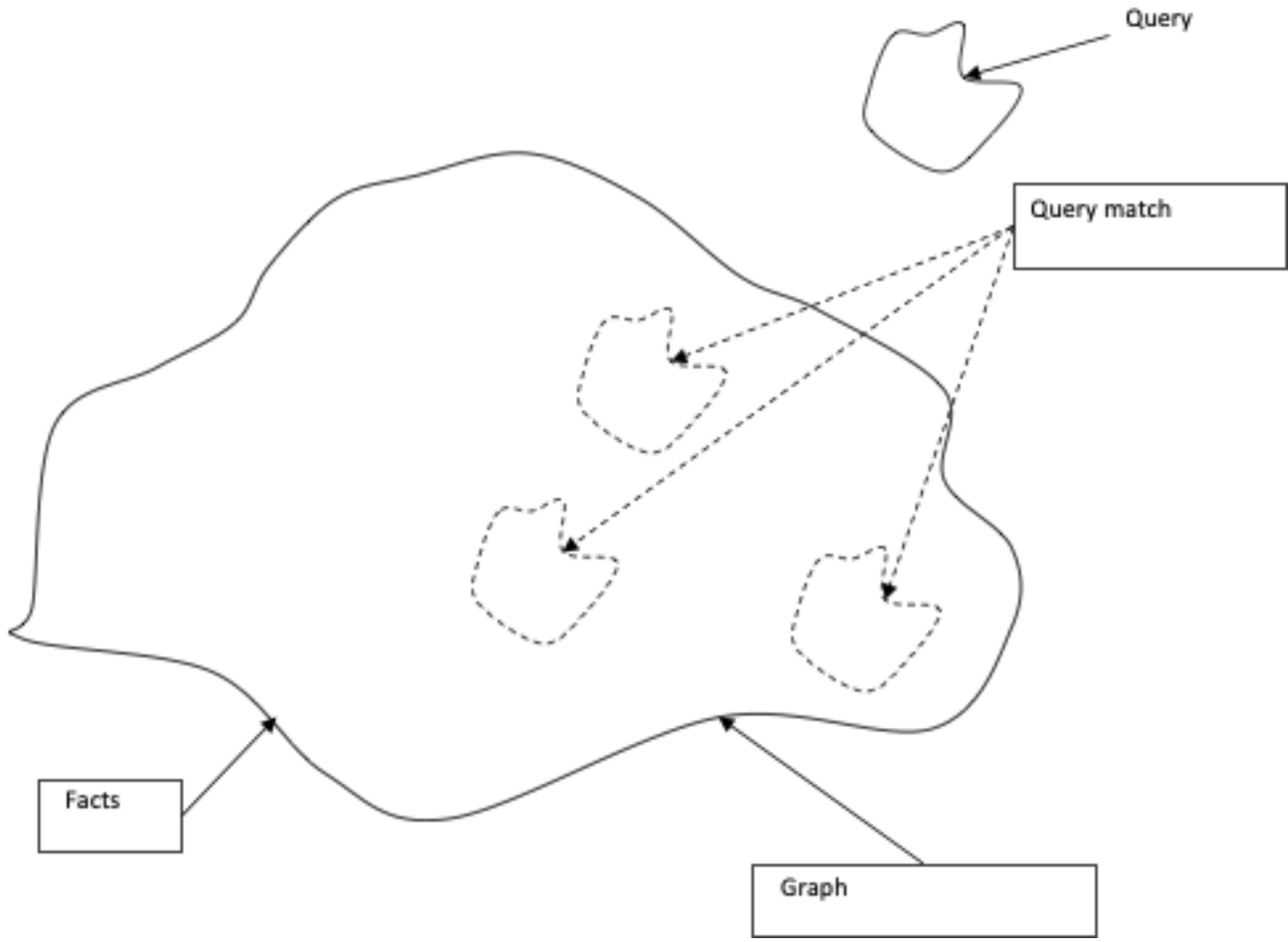
- Graph query languages vary significantly in terms of style, purpose, and expressivity.
- However, they share a common conceptual core:
  - **Graph pattern matching** consists of a graph-structured query that should be matched against the graph database
    - e.g. find all triangles of friendships in a social network.
  - **Graph navigation** is a more flexible querying mechanisms that allows to navigate the topology of the data.
    - e.g find all friends-of-a-friend of some person in a social network.



Graph Pattern Matching

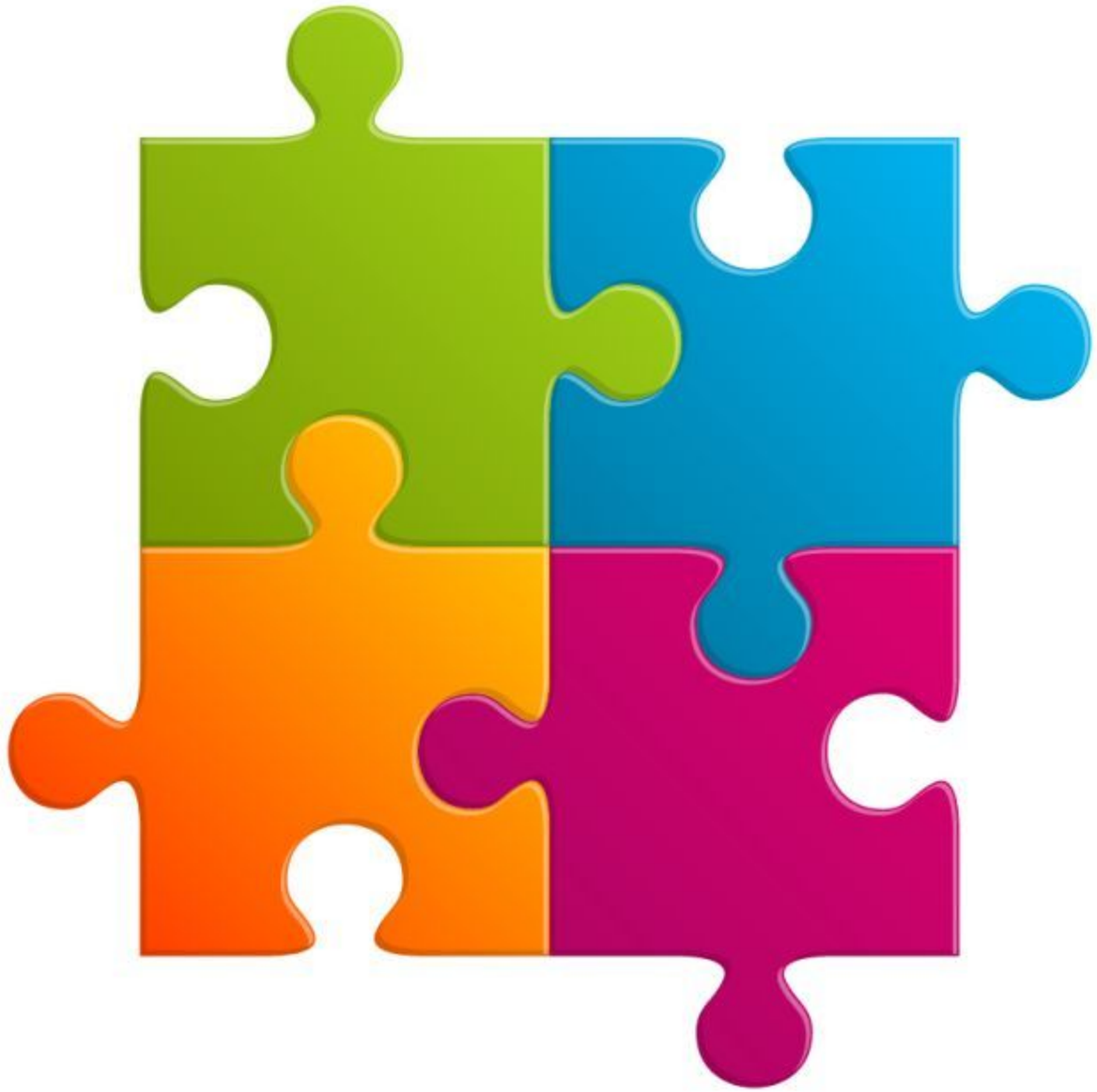
For matching graph patterns we classified the main proposals for the semantics into two categories:

- **Homomorphism-base**d: matching the pattern onto a graph with no restrictions.
- **Isomorphism-based**: one of the following restrictions is imposed on a match:
  - **No-repeated-anything**: no part of a graph is mapped to two different variables.
  - **No-repeated-node**: no node in the graph is mapped to two different variables.
  - **No-repeated-edge**: no edges in the graph is mapped to two different variables.



Basic Graph patterns VS. Complex Graph patterns

- Basic Graph Patterns (BGPs) are just graph to match within the bigger graph database. BGPs are the core of any graph query language.
- Complex Graph Patterns (CGPs) extend BGPs with some additional query features such as UNION, Difference, Projection, Optional (aka left-outer-join), and Filters.



CGPs Operators: Projection

- Like `SELECT` in SQL, is used also to select project on specific outputs.
- Example: retrieve only the names of actors who starred together in Unforgiven

CGPs Operators: Union

- Intended to merge the result of two queries
- Let $Q1$ and $Q2$ be two graph patterns. The union of $Q1$ and $Q2$ is a complex graph pattern whose evaluation is defined as the union of the evaluations.
- Example: *find the movies in which Clint Eastwood acted or which he directed*.

CGPs Operators: Difference

- The difference of $Q1$ and $Q2$ is also a complex graph pattern whose evaluation is defined as the set of matches in the evaluation of $Q1$ that do not belong to the evaluation of $Q2$.
- Logically a form of **negation**
- Example: * find the movies in which Clint Eastwood acted but did **not** direct*.

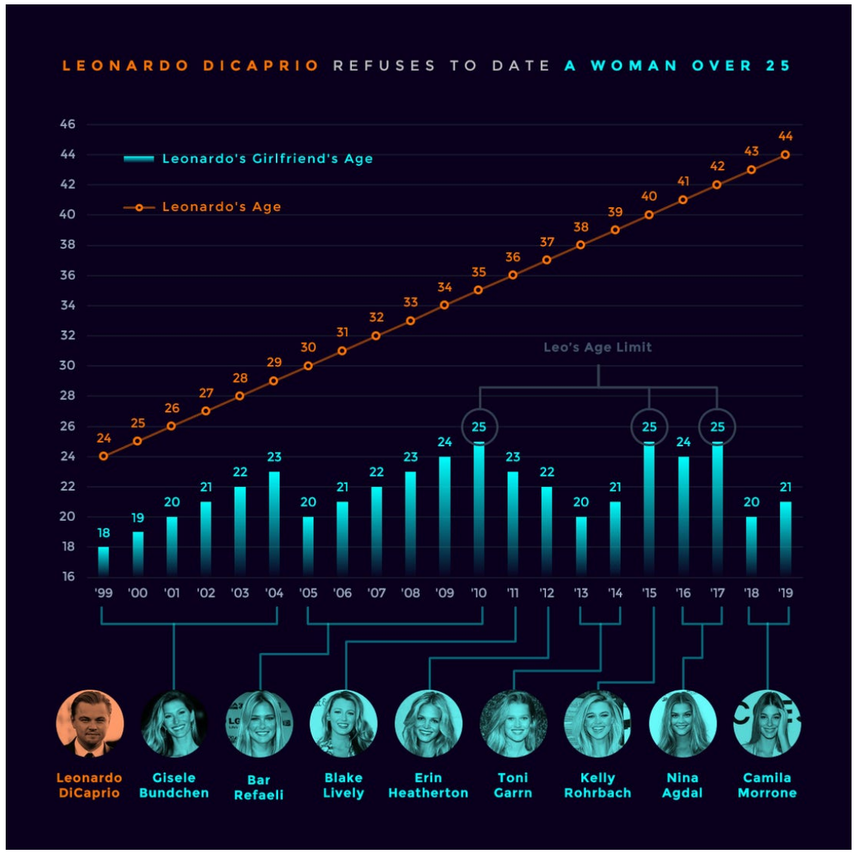
CGPs Operators: Optional

- This feature is particularly useful when dealing with incomplete information, or in cases where the user may not know what information is available.
- Essentially a Left-join
- Example: *Find the information relating to the gender of users is incomplete but may still be interesting to the client, where available.*

CGPs Operators: Filter

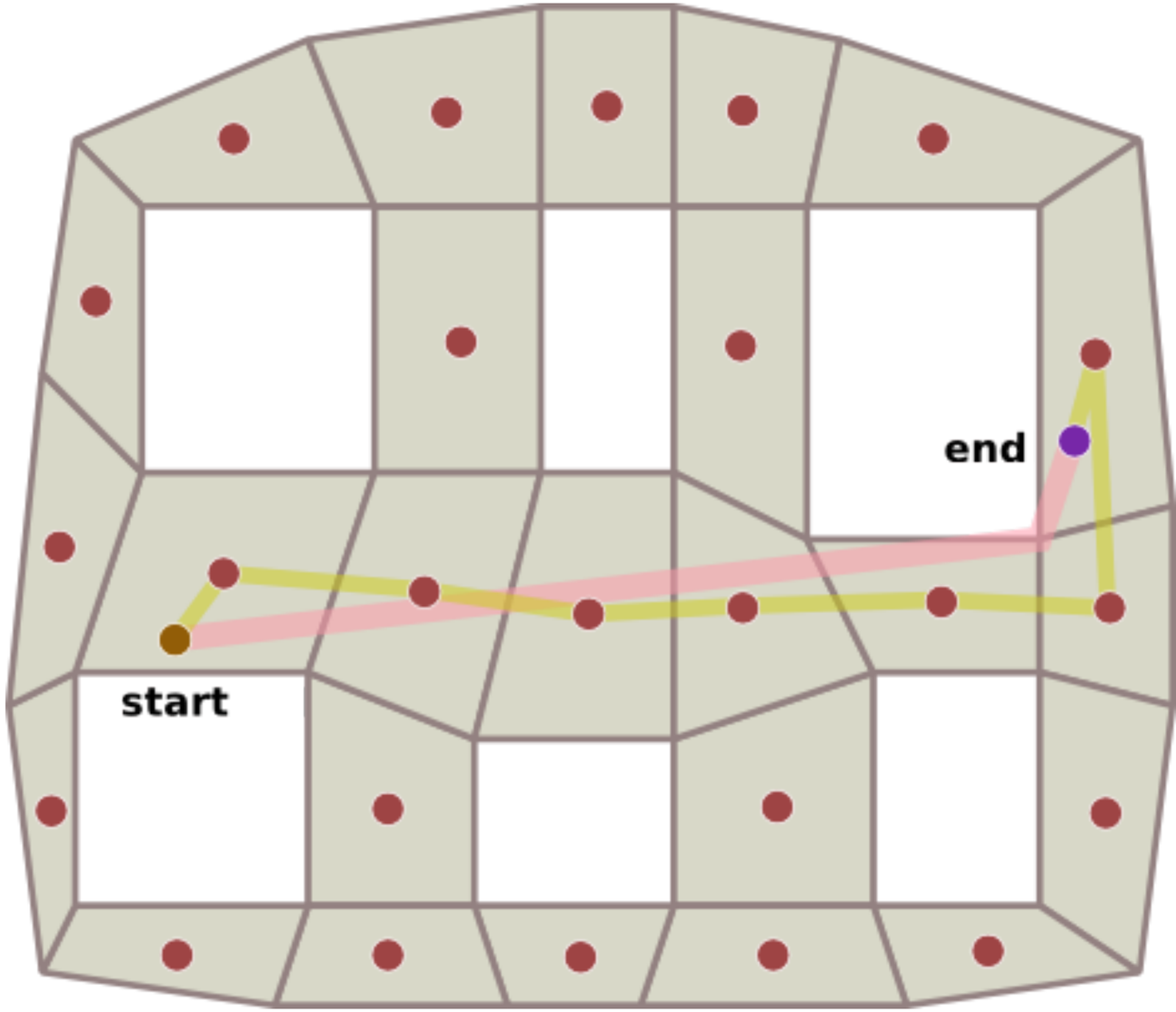
- Users may wish to restrict the matches of a cgp over a graph database G based on some of the intermediate values returned using, for example, inequalities, or other types of expressions.
- Equivalent to relational selection
- Example: *find all male actors that acted in a Clint Eastwood's movie*

Or find all Leonardo Di Caprio's ex girlfriends that are were above 25 yo.



Hint: None

Navigational (Path) Queries in Graphs



Navigational Path Queries

- Graph patterns allow for querying graph databases in a bounded manner.
- Navigational Path Queries provide a more flexible querying mechanisms (yet more expensive) that allow to navigate the topology of the data.
- One example of such a query is to find all friends-of-a-friend of some person in a social network.

Path under Set Semantics

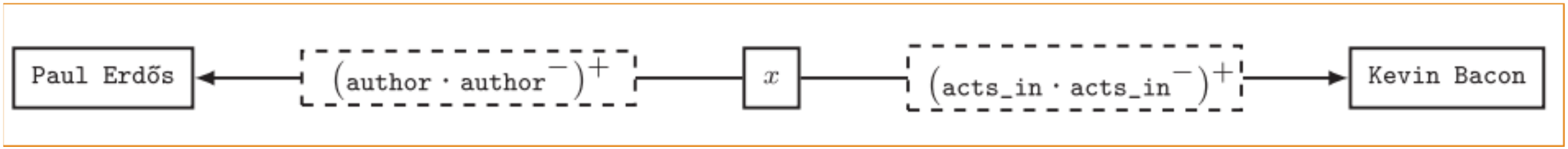
- **Arbitrary paths**: All paths are considered. More specifically, all paths in G that satisfy the constraints of P are included in P (G).
- **Shortest paths**: In this case, P (G) is defined in terms of shortest paths only, that is, paths of minimal length that satisfy the constraint specified by P.
- **No-repeated-node paths**: In this case, P (G) contains all matching paths where each node appears once in the path; such paths are commonly known as simple paths. This interpretation makes sense in some practical scenarios; for example, when finding a route of travel, it is often not desired to have routes that come to the same place more than once.
- **No-repeated-edge paths**: Under this semantics, P (G) contains all matching paths where each edge appears only once in the path. The Cypher query language of the Neo4j engine currently uses this semantics.

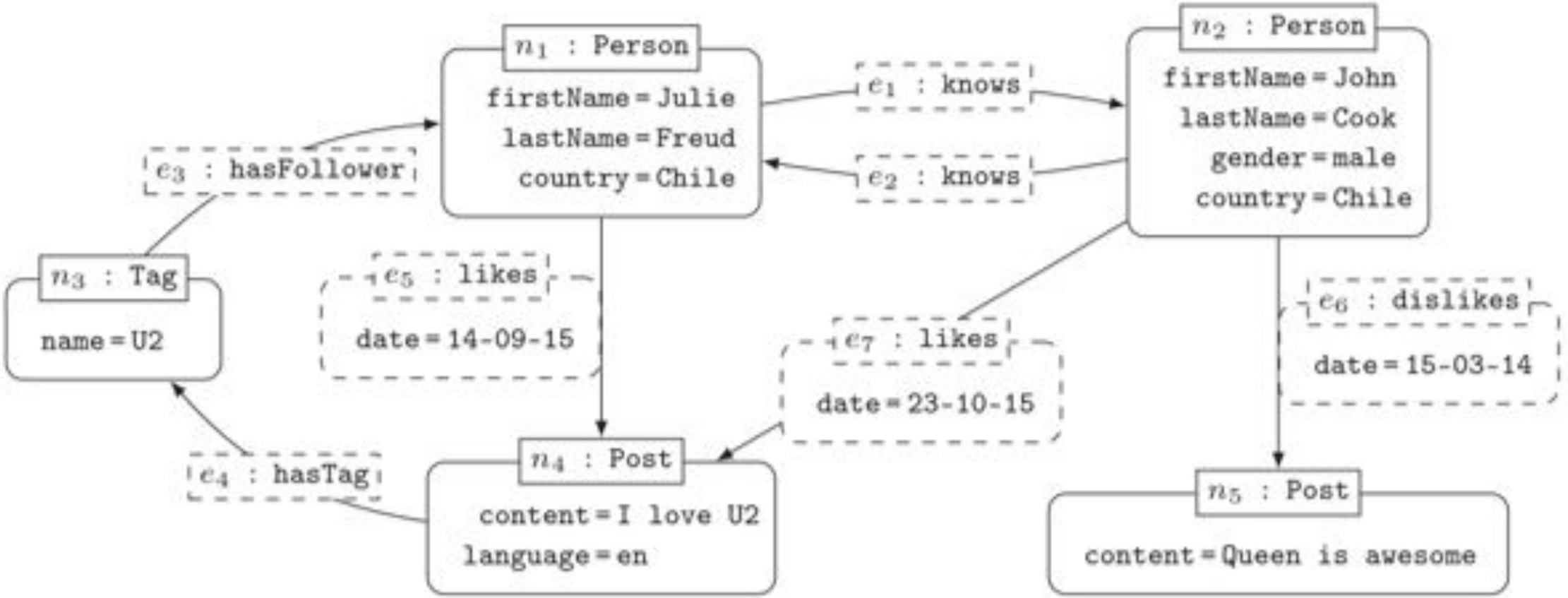
Output of Navigational Queries

- As hinted at previously, a user may have different types of questions with respect to the paths contained in the evaluation P(G), such as:
  - *Does there exist any such path*
  - *Is a particular path contained in P (G)*
  - *What are the pairs of nodes connected by a path in P (G)*
  - *What are (some of) the paths in P (G)*
- We can Categorize such questions by what they return as results:
  - Boolean --- (True / False) values.
  - Nodes --- are interested in the nodes connected by specific paths.
  - Paths --- some or all of the full paths are returned from P (G). Example:Some of the Shortest Paths.
  - Graphs --- is to offer a compact representation of the output as a graph

Navigational Graph Patterns (NGPs)

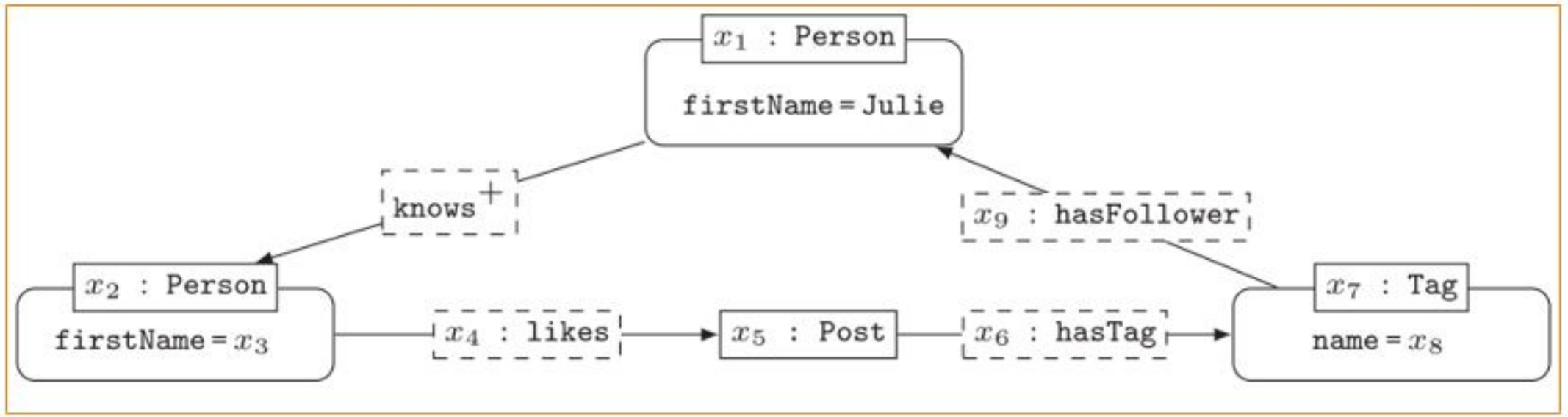
- Combining path queries with basic graph patterns (BGPs) gives rise to navigational graph patterns (NGPs).
- In particular, this language allows to express that some edges in a graph pattern should be replaced by a path (satisfying certain conditions) instead of a single edge.
- Example: Persons and movies are connected , while a person can also have an author edge connecting it to an article.
- In such a database we might be interested in finding people with finite Erdos-Bacon number, that is, people who are connected to Kevin Bacon through co-stars relations and are connected to Paul Erdos through co-authorship relations.




n_1 : Person
firstName = Julie
lastName = Freud
country = Chile
n_2 : Person
firstName = John
lastName = Cook
gender = male
country = Chile
e_1 : knows
e_2 : knows
e_3 : hasFollower
n_3 : Tag
name = U2
e_5 : likes
date = 14-09-15
e_7 : likes
date = 23-10-15
e_6 : dislikes
date = 15-03-14
e_4 : hasTag
n_4 : Post
content = I love U2
language = en
n_5 : Post
content = Queen is awesome

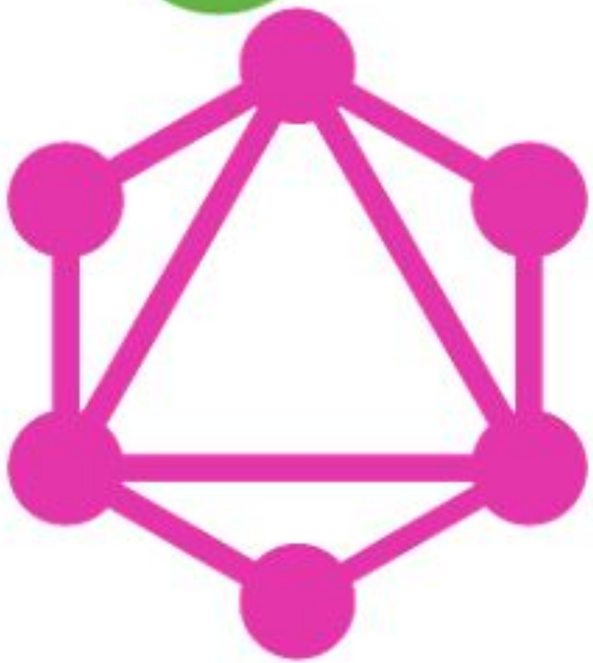
Navigational Graph Patterns (NGPs)

- Coming back to the social network, we might be interested in finding all friends of friends of Julie that liked a post with a tag that Julie follows. The navigational graph pattern in this Figure expresses this query over our social graph.
- Extending Navigational Graph patterns with the complex operators of "Projection", "Optional", "Filter", "Union" and "Difference" give the rise to another new type of them: (cngps).
- Example: Let's call these results the "recommended posts" for Julie. Now consider a copy of the same pattern to find the recommended posts for John.



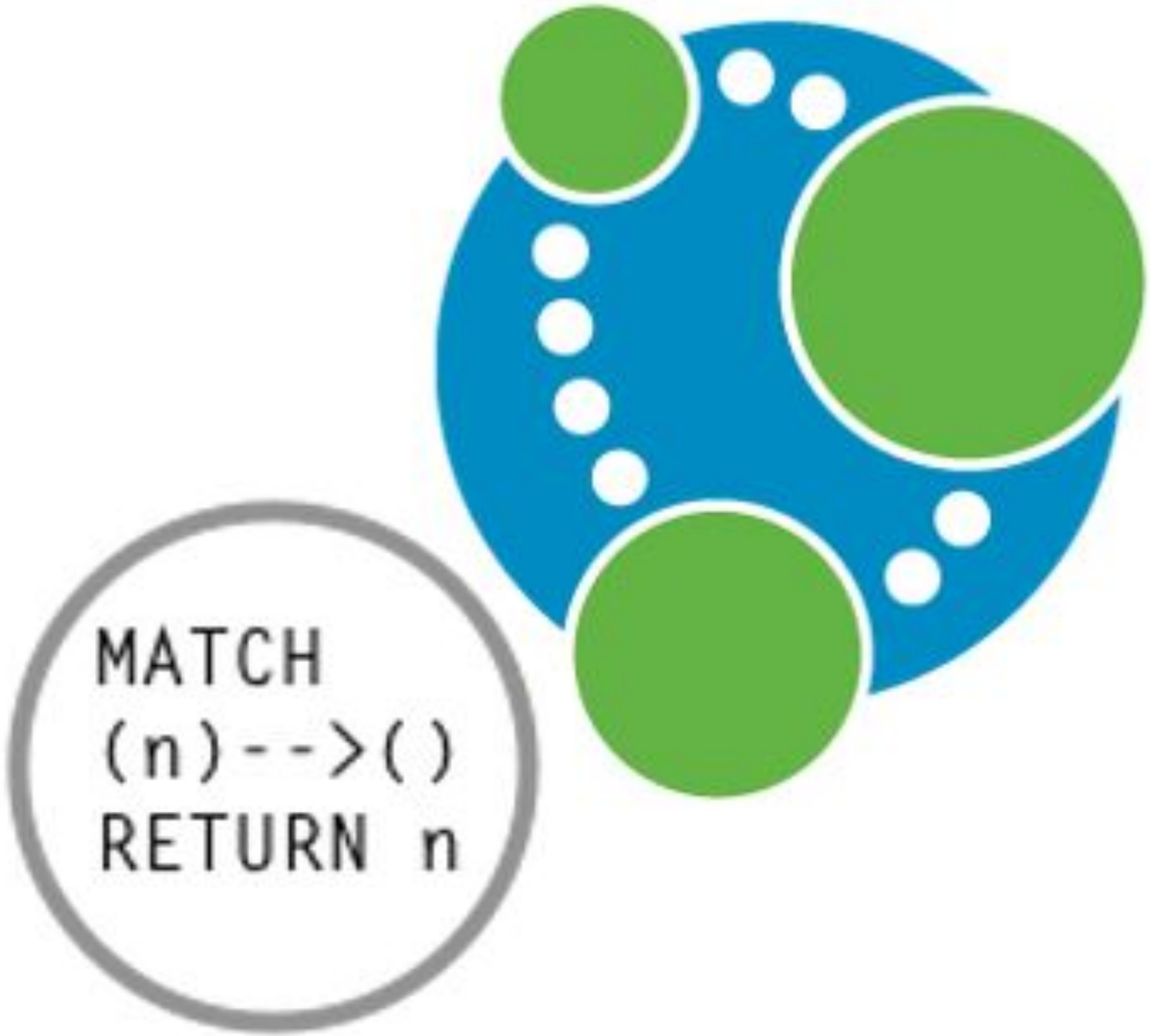
Graph Query Languages In Action

- **Cypher** --- Property Graphs
- Gremlin--- Property Graphs
- GraphQL --- Edge-Labelled multi Graphs
- **SPARQL** --- Edge-Labelled Graphs RDF
- G-Core --- Property Graphs



Cypher - The Neo4J DB Query Language

- Cypher is a declarative language for querying property graphs that uses "patterns" as its main building blocks.
- Cypher's declarative syntax provides a familiar way to match patterns of nodes and relationships in the graph.
- It is backed by several companies in the database space and allows implementors of databases and clients to freely benefit, use from and contribute to the development of the openCypher language.



Graph Patterns in Cypher (Projection)

- Patterns are expressed syntactically following a "pictorial" intuition to encode nodes and edges with arrows between them.
- The following queries ask for co-stars of the *"Unforgiven"* movie.

```
MATCH (x:Person)-[:acts_in]->
    (m:Movie {title: "Unforgiven"})
        <-[:acts_in]-(y:Person)
RETURN x,y```
[.column]

```sql

MATCH (x:Person)-[:acts_in]->(m:Movie
    {title: "Unforgiven"})
(y:Person)-[:acts_in]->(m)
RETURN x,y
```
```

Comple Graph Patterns in Cypher: Union

```
MATCH (:Person
    {name:"Clint Eastwood"})-[:acts_in]->(m:Movie)
RETURN m.title
UNION ALL
MATCH (:Person
    {name:"Clint Eastwood"})-[:directs]->(m:Movie)
RETURN m.title
```

Comple Graph Patterns in Cypher: Difference

```
MATCH (p:Person)-[:acts_in]->(m:Movie
    {title: "Unforgiven"})
WHERE NOT (p)-[:direct]->(m)
RETURN m.title
```

Comple Graph Patterns in Cypher: Optional

```
MATCH (p:Person)-[:acts_in]->(m:Movie)
OPTIONAL MATCH (p)-[x]->(m)
WHERE type(x) <> "acts_in"
RETURN p.name, m.title, type(x)
```

Navigational Queries in Cypher

- While not supporting full regular expressions, Cypher still allows transitive closure over a single edge label in a property graph.
- Since it is designed to run over property graphs, Cypher also allows the star to be applied to an edge property/value pair.
- **Example**: compute the friend-of-a-friend relation. The following query selects pairs of nodes that are linked by a path completely labelled by knows. To do this, it applies the star operator * over the label knows .

```
MATCH (x:Person)-[:knows*]->(y:Person)
RETURN x,y
```

Navigational Queries in Cypher

- Example 2. If we wanted to find friends of friends of Julie and return only the shortest witnessing path. This will return a single shortest witnessing path. If we wanted to return all shortest paths, then we could replace "shortestPath" with "allShortestPaths".

```
MATCH (x:Person {firstname:"Julie"}),
p = shortestPath( (x)-[:knows*]->(y:Person))
RETURN p
```

- Example 3. Coming back to the social network, if we want to find all friends of-friends of Julie that liked a post with a tag that Julie follows, we can use the following Cypher query:

```
MATCH (x:Person {firstname:"Julie"})-[:knows*]->(y:Person))
MATCH (y)-[:likes]->()->[:hasTag]->(z)
MATCH (z)-[:hasFollower]->(x)
RETURN y
```

Navigational Queries Cypher

- Another interesting feature available in Cypher is the ability to return paths.
- Example 4. If we wanted to return all friends of friends of Julie in the graph, together with a path witnessing the friendship, then we can use:

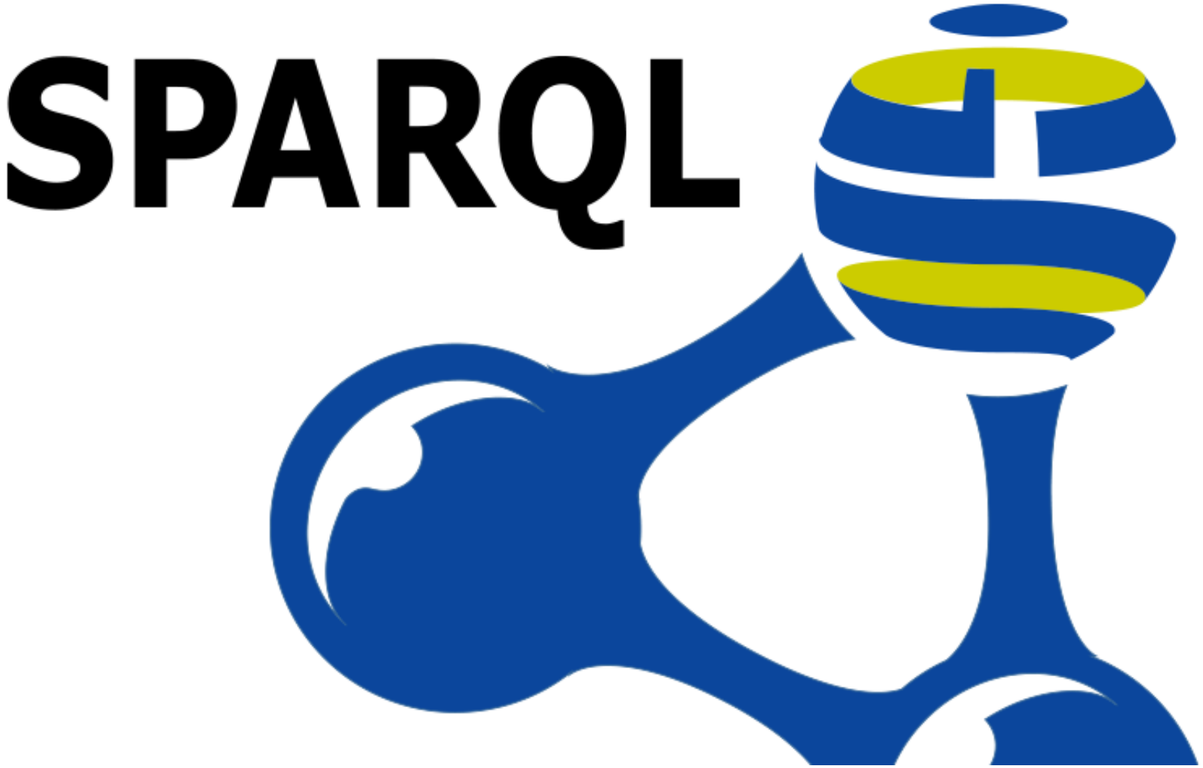
```
MATCH p = (:Person name:"Julie")-[:knows*]->(x:Person)
RETURN x,p
```

- Result will be:

x	p
Node[2]	[Node[1],:knows[1],Node[2]]
Node[1]	[Node[1],:knows[1],Node[2],:knows[2],Node[1]]

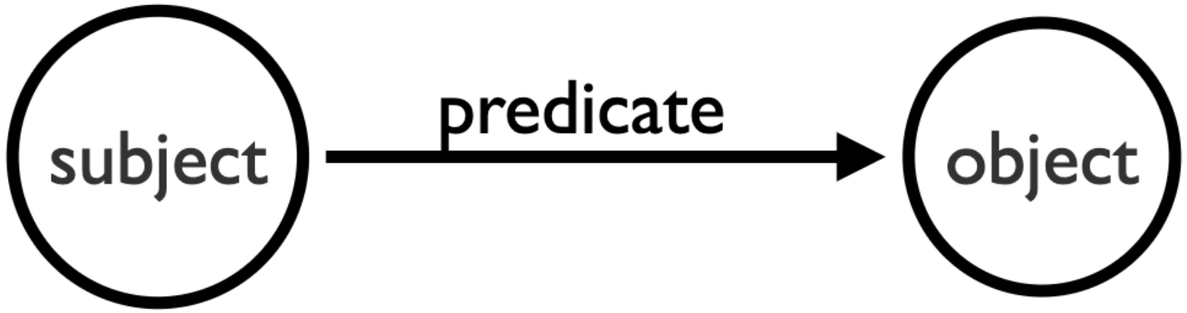
SPARQL - The RDF Query Language

- SPARQL is the standard query language of RDF and become official W3C recommendation since 2003.
- SPARQL is a pattern matching query language over the RDF graph. SPARQL queries contain a set of triple patterns (TPs), also known as Basic Graph Patterns (BGPs).
- Triple patterns are similar to RDF triple patterns, but each of the subject, predicate or object may be unbounded variable preceded by ("?") prefix.
- SPARQL mission is to bind those variable by matching the query patterns to triples in the RDF dataset.

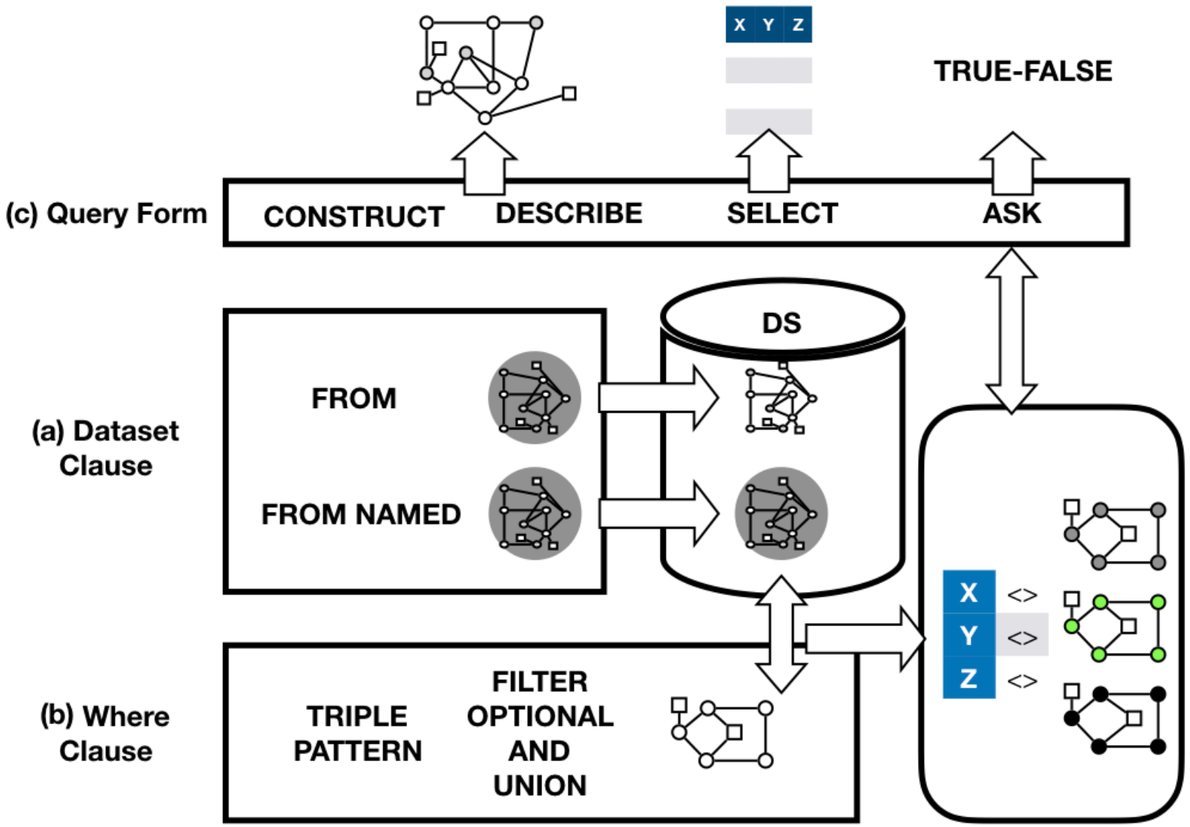


RDF Graphs

- RDF graphs are a special type of edge-labelled graph.
- The basic bloc is a triple `<subject> <predicate> <object>`
- Nodes and edges are identified using URIs
- Obejcts can be literals (Numbers, strings)



Anathomy of a SPARQL Query



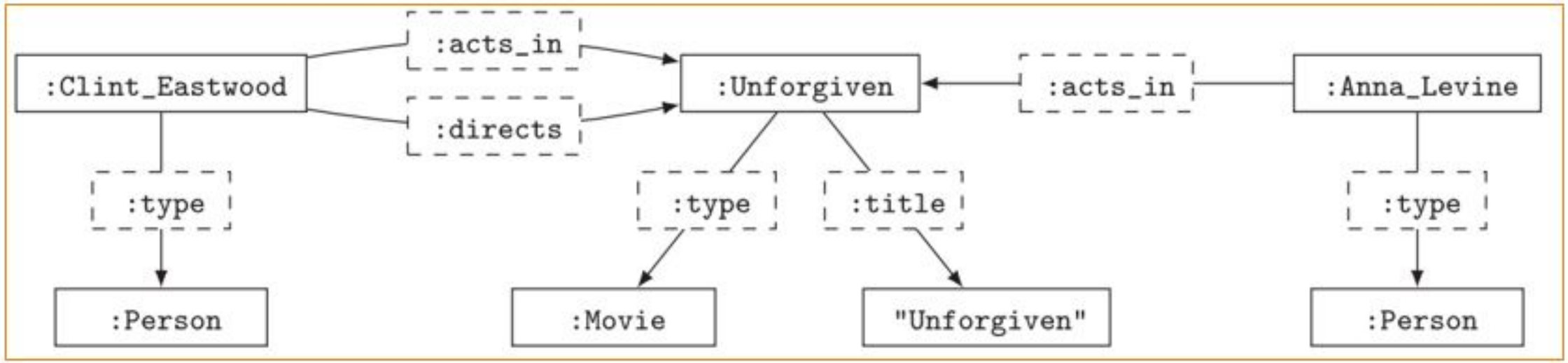
SPARQL Graph Patterns

Let us take a closer look at how graph patterns are applied in three practical query languages: SPARQL, Cypher, and Gremlin.

- SPARQL: Projection
  - The following SPARQL query represents a complex graph pattern that combines the basic graph pattern with a projection that asks to only return the co-stars and not the movie identifier.

```
PREFIX : <httpL//example.org#>
SELECT ?x ?y
WHERE {
    ?x :acts_in ?y ;
       :type :Person .
    ?z :acts_in ?y ;
       :type :Person .
    ?y :title "Unforgiven" ;
       :type :Movie .
    FILTER(?x!=?y)
}
```

?x	?y
:Clint_Eastwood	:Anna_Levine
:Anna_Levine	:Clint_Eastwood


:Clint_Eastwood
:acts_in
:directs
:Unforgiven
:acts_in
:Anna_Levine
:type
:type
:title
:type
:Person
:Movie
"Unforgiven"
:Person

Complex Graph Patterns in SPARQL (Union)

- This example of a union to find movies that Clint Eastwood has acted or directed in.

```
SELECT ?x
WHERE {
    {:Clint_Eastwood :acts_in ?x . }
    UNION
    {:Clint_Eastwood :directs ?x . }
}
```

?x
:Unforgiven

Complex Graph Patterns in SPARQL (Difference)

- SPARQL Difference We could use difference to ask for people who acted in the movie Unforgiven but who did not (also) direct.

```
SELECT ?x
WHERE {
    {?x :acts_in :Unforgiven . }
    MINUS
    {?x :directs :Unforgiven . }
}
```

?x
:Anna_Levin

Complex Graph Patterns in SPARQL (OPTIONAL)

- SPARQL: Optional
  Using optional, we could ask for movies that actors have appeared in, and any other participation they had with the movie besides acting in it

```
SELECT ?x ?y ?z
WHERE {
    {?x :acts_in ?y . }
    OPTIONAL
    {?x ?z ?y .
        FILTER(?x != :acts_in) }
}```

### Navigational Queries in Action: SPARQL

- Since Version 1.1 , SPARQL permits the use of property paths.

- SPARQL Property Paths are an extended form of regular expression.

- As a consequence, we can express any path query using SPARQL 1.1.

---
[.column]
### Example 1
-  Consider the following SPARQL query to find all pairs of actors who have finite collaboration distance, we can use


```SQL

SELECT ?x ?y
WHERE { ?x (:acts_in/acts_in*) ?y }
```

Example 2

- Consider the following SPARQL query with a negated property-set.
- This query will match :Unforgiven (the IRI) and "Unforgiven" (the title string) for ?y.

```
SELECT ?y
WHERE { :Clint_Eastwood (!{rdf:type,:directs})* ?y }
```
```

Navigational Queries in SPARQL

- Similarly, SPARQL can also express navigational graph patterns (ngps).
- **Example**: find all people with a finite Erdos-Bacon number can be expressed in SPARQL as in the query below, which is a conjunction of two RPQs, where the symbol "." denotes conjunction.

```
SELECT ?x
WHERE {
    ?x (:acts_in/^:acts_in)* :Kevin_Bacon .
    ?x (:author/^:author)* :Paul_Erdos .
}
```

Navigational Queries in SPARQL

- Likewise, SPARQL can express complex navigational graph patterns (cngps).
- **Example**. We can express an RDF version of the query for the posts recommended to Julie but not to John as follows:

```
SELECT ?x ?y ?z
WHERE {
    {:Julie :knows+/:likes ?x ;
            :hasTag/:hasFollower :Julie . }
    MINUS
    {:John :knows+/:likes ?x ;
           :hasTag/:hasFollower :John . }
}
```

Other Popular Query Languages.

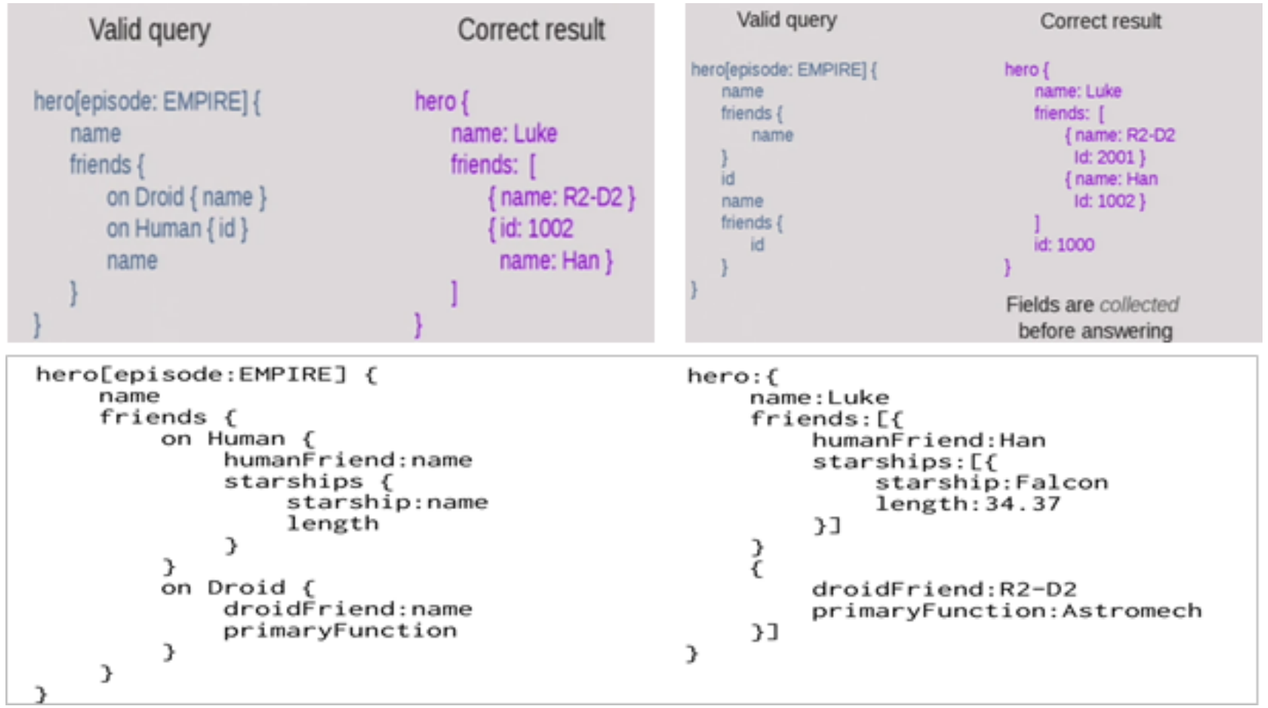
- G-Core[117]
  - Community effort between industry and academia to shape and standardize the future of graph query languages.
  - G-Core Features:
    - Composability: Graphs are inputs and outputs of the queries. Queries can be composed. The fact that G-CORE is closed on the PPG data model means that subqueries and views are possible.
    - Paths are First Class-Citizens: Paths can increase the expressivity of the language. G-Core extends graphs models with paths (PPG). Can have labels and prosperities.
    - Capture a core: Standards are difficult and politics, Take the successful functionalities with tractable evaluation of current languages as a base to develop

---

[117] Angles, Renzo, et al. *G-CORE: A core for future graph query languages*. Proceedings of the 2018 International Conference on Management of Data. ACM, 2018.

Other Popular Query Languages.

- GraphQL also removes redundancy, Another restriction is type restrictions.
- The following Figure (left) shows an example GraphQL query over the domain (F, A, T) and the response is in the right.



Graph Query Languages Features Comparison

	SPARQL	Cypher	Gremlin	G-Core	GraphQl	Notes
Focus	RDF, LOD Datasets	General	Navigation-Traversal	General & Graph Composability	Web Data Access	
Supported Graph Data model	RDF(Edge-labelled graph)	Property Graph	Property Graph	Property Graph	Edge- labelled graph	
Standardization	Yes "W3C"	NO	NO	NO (attempt to standardize)	NO	
Easy to learn	Yes	YES	NO	Yes	YES	
Syntax	SQL-like	SQL-like	Functional Programming	SQL-like	REST Like Query	
Composability	NO	YES Cypher 10	NO	YES	NO	
Paths Storage	NO	NO	NO	YES	NO	
GRAPH VIEWS & Subqueries	NO	NO	NO	YES	NO	* SPARQL may support subqueries but not Views.
Semantics of Pattern Matching	homomorphism-based, bags	no-repeated-edges, bags	homomorphism-based, bags	-	-	
Declarative	Declarative	Declarative	Declarative	Imperative	Declarative	
Output	Table of nodes or edges/ Boolean	Paths, Table nodes or edges/ Boolean	Nodes/ Paths	Always GRAPHS	Values	* GraphQl can work with SQL tables (RDBs) and also return tables.
Navigational Queries	Yes Using "Path Prosperities", Arbitrary Paths, Sets	YES using RPQs, no repeated edges, Bags	YES using RPQs, Arbitrary Paths, Sets	-	NO	* Supported Only from SPARQL 1.1