Introduction to Neo4j Graph Data Science

Jean-Marc Guerin
Neo4j Sales Engineer - UK
jean-marc.guerin@neo4j.com

Agenda

Introduction to Neo4j Graph Data Science

1. What is Graph Data Science and why do you need it?
2. Graph Algorithms & How to use them
3. Graph Machine Learning and Integrations with ML frameworks
4. Applied examples

Break

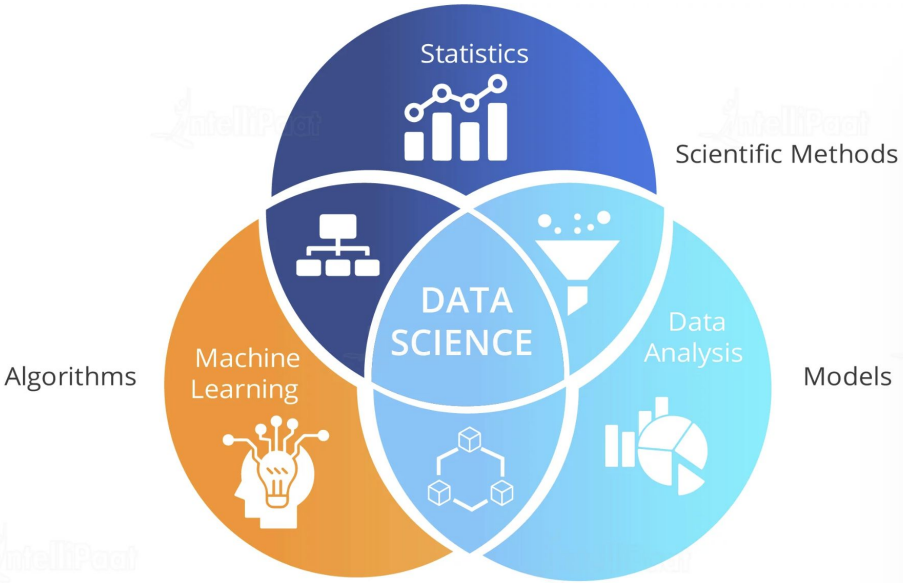
Hands-on:

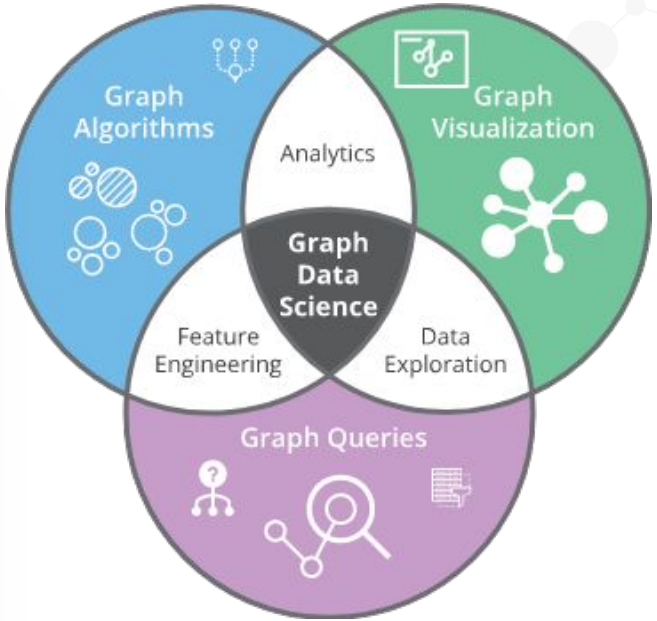
- Fraud detection and Investigation in Python Notebook
- Machine learning pipeline: node classification

Q&A

Neo4j Graph Data Science #1: What is Graph Data Science and Why do you need it?

What is *Graph* data science?

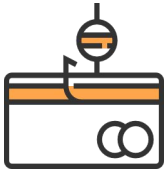




.. relationships to answer questions.

Where is *Graph* data science applied?

Fraud Detection



Disambiguation & Segmentation

Personalized Recommendations

Life Sciences

Churn Prediction

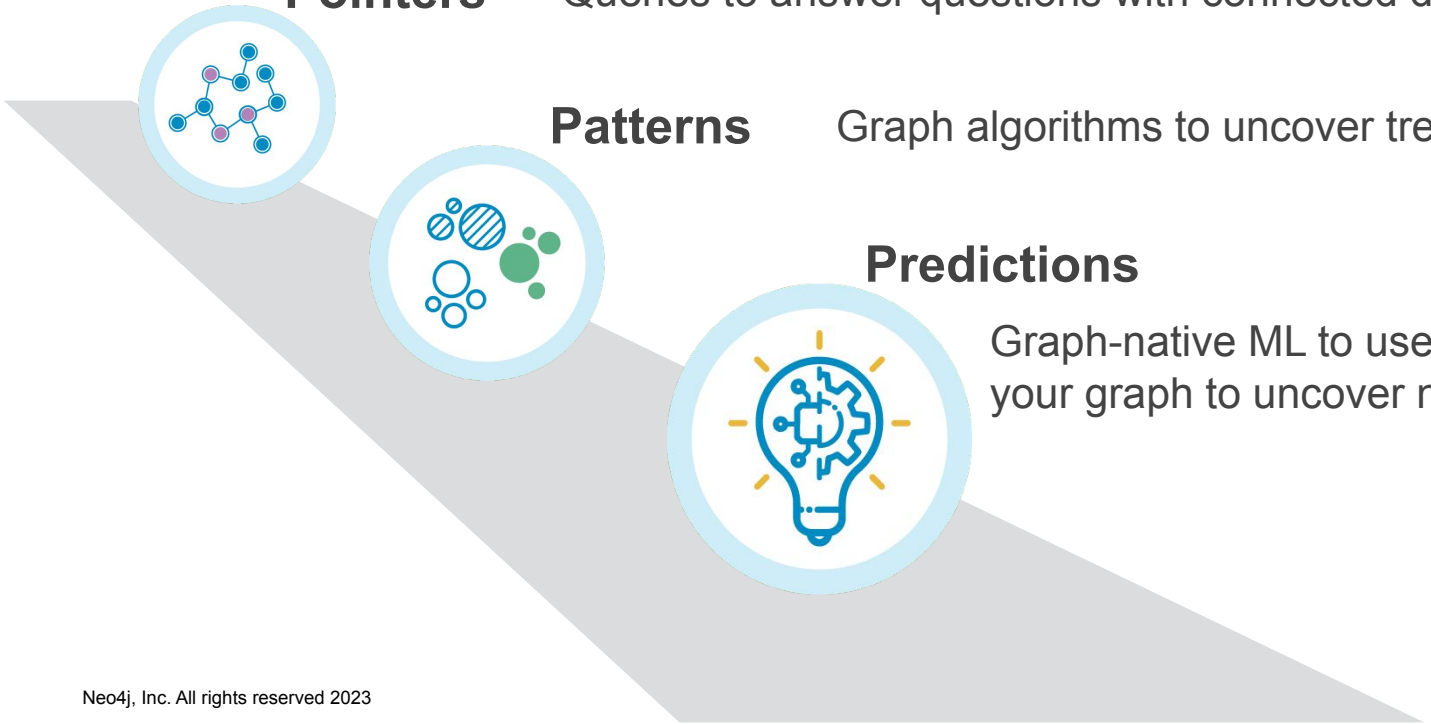
Search & Master Data Mgmt.

Predictive Maintenance

Cybersecurity

Neo4j Graph Data Science

From Analytics to Graph-Native Machine Learning



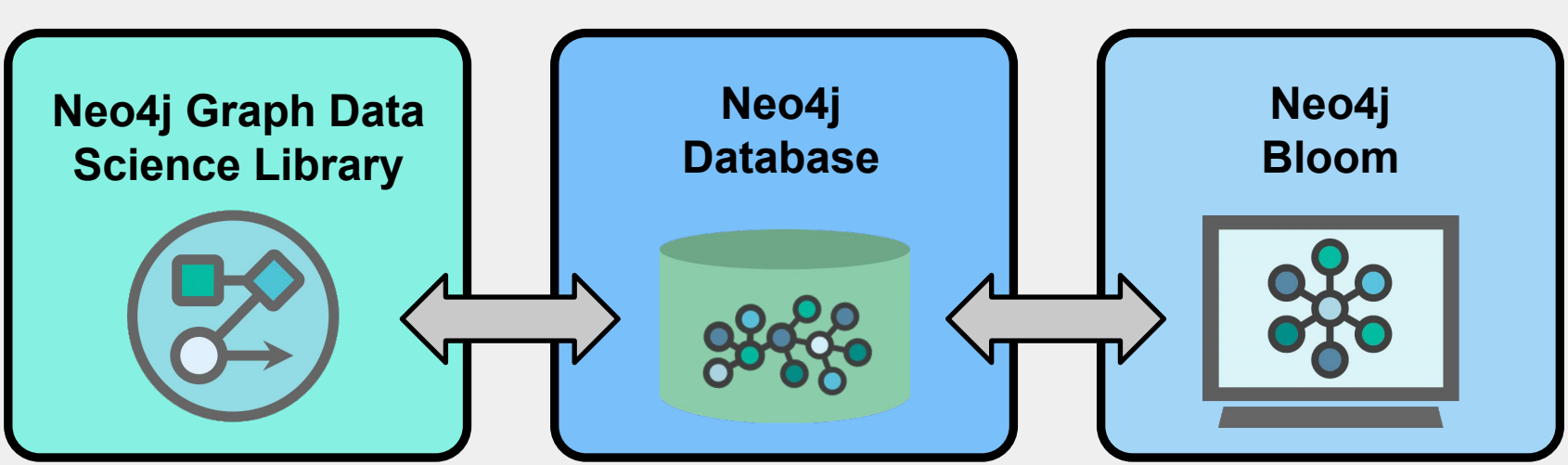
Pointers Queries to answer questions with connected data

Patterns Graph algorithms to uncover trends and patterns

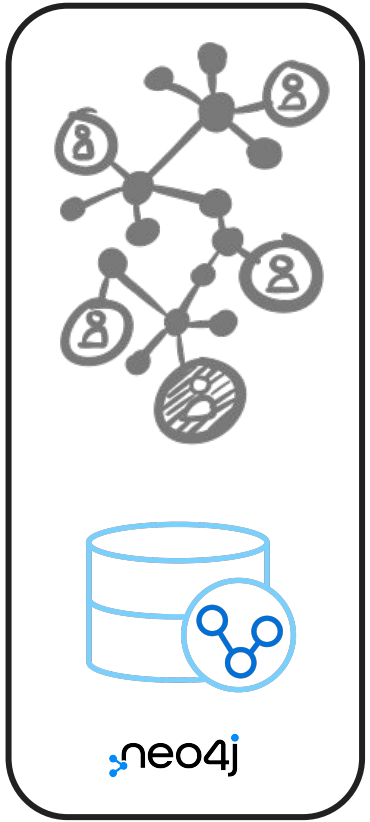
Predictions

Graph-native ML to use the topology of your graph to uncover new facts

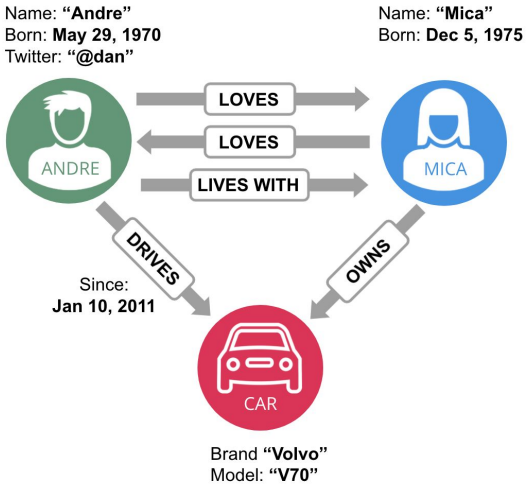
Neo4j Graph Data Science



Neo4j Graph Database



DATA



GRAPH QUERIES

```
MATCH (a)-[:KNOWS]->(b)
RETURN b.name
UNION ALL
MATCH (a)-[:LOVES]->(b)
RETURN b.name
```

CYPHER

https://neo4j.com/developer/cypher/

neo4j

Graph Algorithms in Neo4j GDS

Pathfinding & Search

- A* Shortest Path
- All Pairs Shortest Path
- Breadth & Depth First Search
- Delta-Stepping Single-Source
- Dijkstra Single-Source
- Dijkstra Source-Target
- Minimum Spanning Tree & K-Spanning Tree
- Random Walk
- Yen's K Shortest Path
- Minimum Directed Steiner Tree

Centrality & Importance

- ArticleRank
- Betweenness Centrality & Approx.
- Closeness Centrality
- Degree Centrality
- Eigenvector Centrality
- Harmonic Centrality
- Hyperlink Induced Topic Search (HITS)
- Influence Maximization (CELF)
- PageRank
- Personalized PageRank

Community Detection

- Conductance Metric
- K-1 Coloring
- K-Means Clustering
- Label Propagation
- Leiden Algorithm
- Local Clustering Coefficient
- Louvain Algorithm
- Max K-Cut
- Modularity Optimization
- Speaker Listener Label Propagation
- Strongly Connected Components
- Triangle Count
- Weakly Connected Components

Supervised Machine Learning

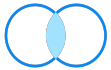
- Link Prediction Pipelines
- Node Classification Pipelines
- Node Regression Pipelines

… and more!

- Collapse Paths
- Graph Sampling
- Graph Stratified Sampling
- One Hot Encoding
- Pregel API (write your own algos)
- Property Scaling
- Split Relationships
- Synthetic Graph Generation

Heuristic Link Prediction

- Adamic Adar
- Common Neighbors
- Preferential Attachment
- Resource Allocations
- Same Community
- Total Neighbors



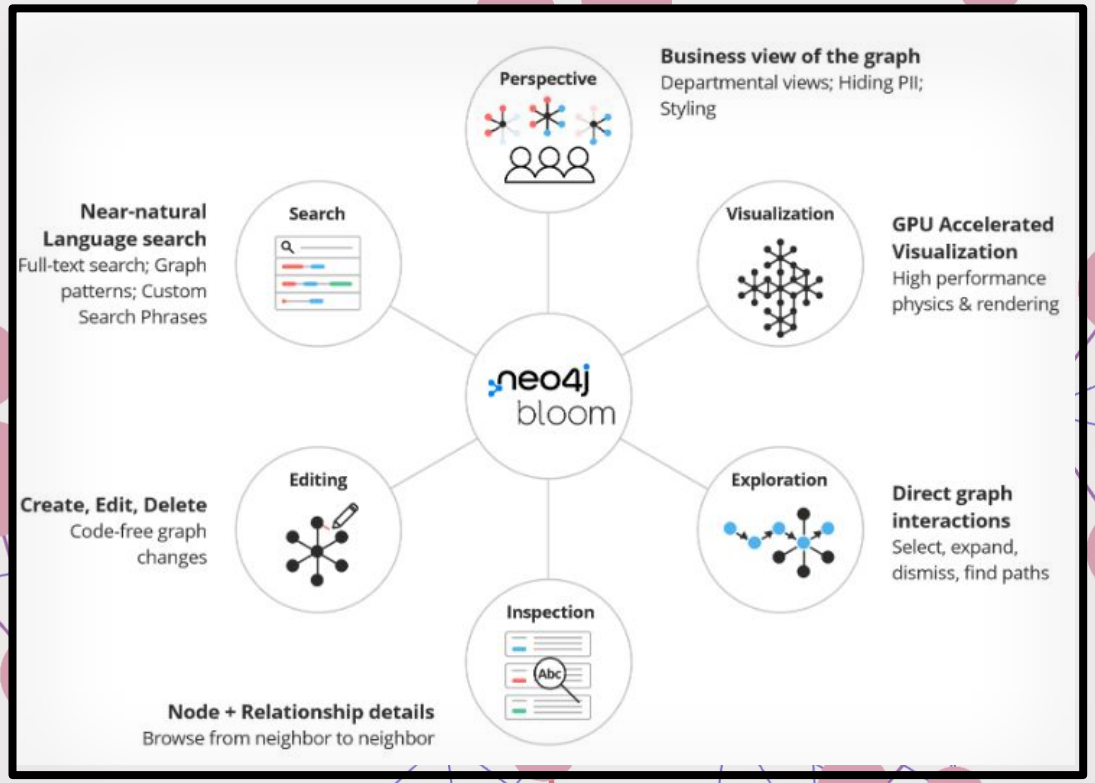
Similarity

- K-Nearest Neighbors (KNN)
- Node Similarity
- Filtered KNN & Node Similarity
- Cosine & Pearson Similarity Functions
- Euclidean Distance Similarity Function
- Euclidean Similarity Function
- Jaccard & Overlap Similarity Functions

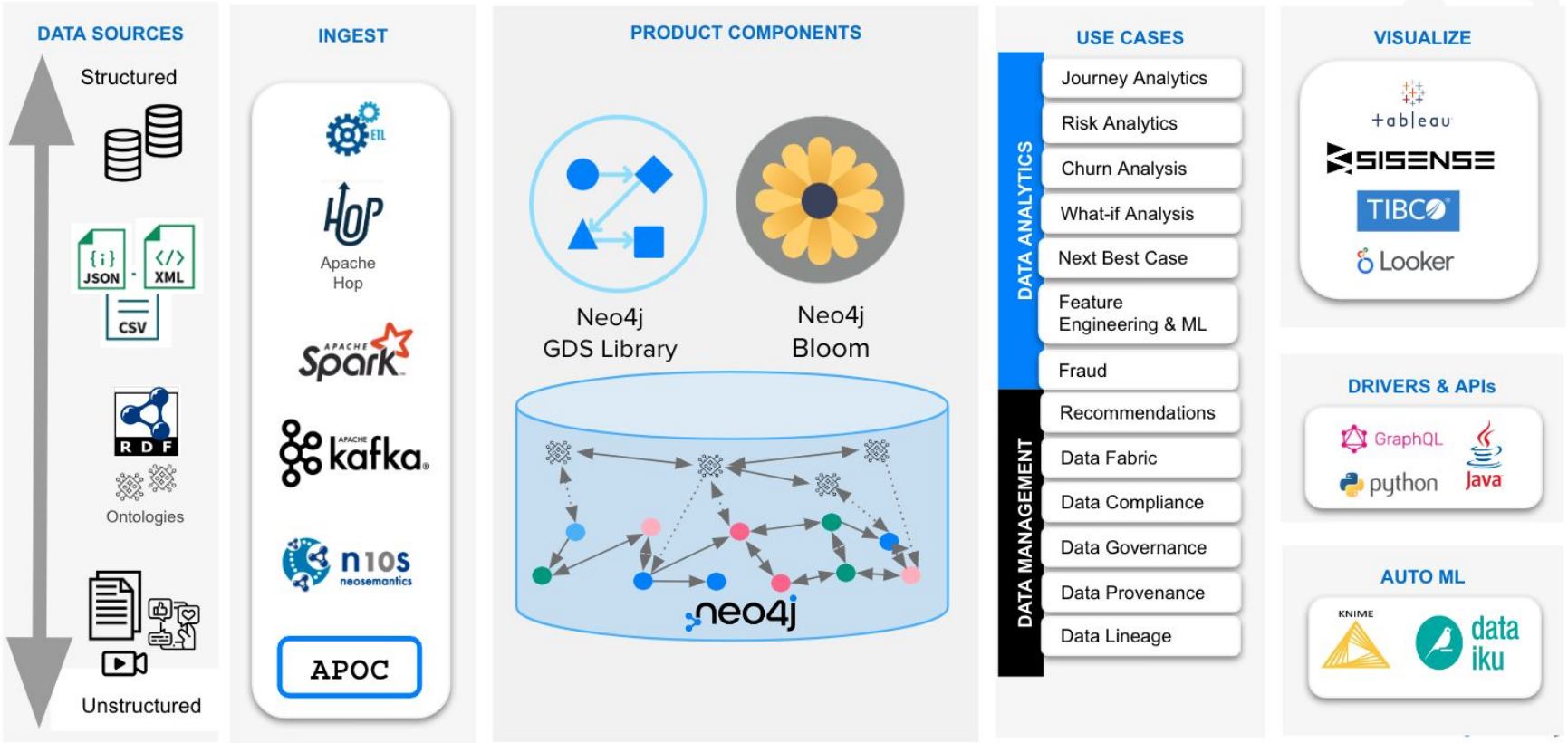
Graph Embeddings

- Fast Random Projection (FastRP)
- FastRP with Property Weights
- GraphSAGE
- Node2Vec
- HashGNN (Knowledge Graph Embedding)

Neo4j Bloom

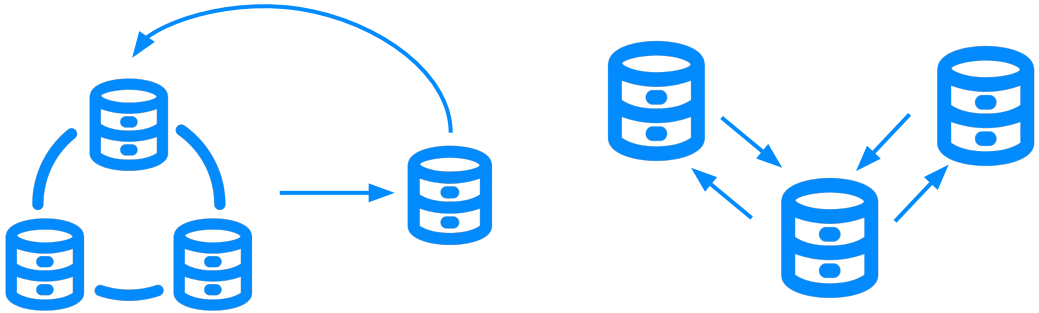

Perspective
Business view of the graph
Departmental views; Hiding PII; Styling
Search
Near-natural Language search
Full-text search; Graph patterns; Custom Search Phrases
Visualization
GPU Accelerated Visualization
High performance physics & rendering
neo4j bloom
Editing
Create, Edit, Delete
Code-free graph changes
Exploration
Direct graph interactions
Select, expand, dismiss, find paths
Inspection
Abc
Node + Relationship details
Browse from neighbor to neighbor

How does Neo4j GDS fit in the data ecosystem



How do we deploy Neo4j GDS

Self-hosted option on any cloud - available via marketplaces



Neo4j's fully managed cloud service

neo4j aura DS

Easy to Use

Quickly transform your data into a graph and begin analysis. First-time users are set up for success with tooltips, guides, and pop-up hints.

Built for Data Scientists

Access over 65 graph algorithms in a single workspace to experiment faster and increase productivity. Use in-graph ML models and the native Python client to simplify your workflows.

Partners and Connectors

Streamline your data pipeline with connectors to data tools like Spark and Kafka. Use your committed spend on Google Cloud Platform to pay for AuraDS.

Enterprise-Ready

Take your analysis from an experiment to deployment in production without having to worry about infrastructure, security, or data limits.

Neo4j is a cloud-friendly database

neo4j

Part 2 of Neo4j Graph Data Science: Graph Algorithms & How to Use Them

Neo4j Graph Data Science: Graph Algorithms

When do you need Graph Algorithms?

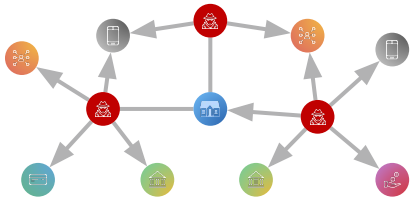
Query (e.g. Cypher/Python)

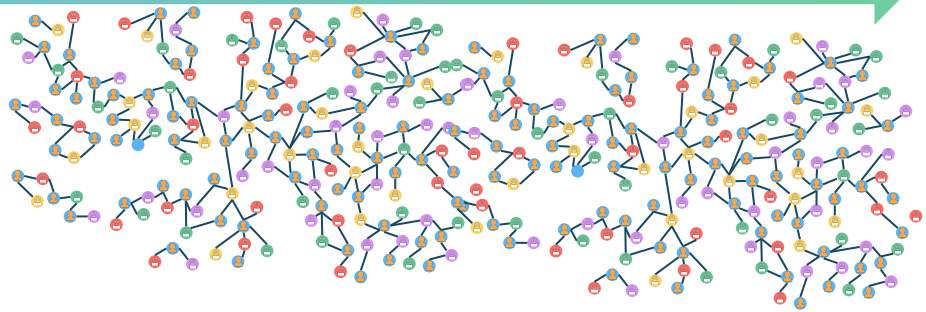
Real-time, local decisioning and pattern matching

Graph Algorithms

Global analysis and iterations

Local Patterns ⟷ **Global Computation**





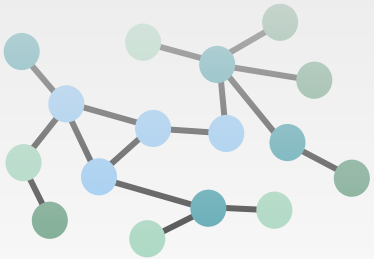
You know what you're looking for and making a decision

You're learning the overall structure of a network, updating data, and predicting

neo4j

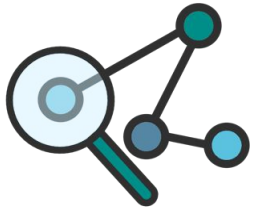
Graphs & Data Science

Knowledge Graphs



Find the patterns you're looking for in connected data

Graph Algorithms



Unsupervised machine learning techniques to identify communities, associations and anomalies

Graph Native Machine Learning

Combine graph metrics and embeddings with supervised machine learning to predict links, labels, and missing data.

Neo4j Graph Data Science Algorithms

Pathfinding & Search

- A* Shortest Path
- All Pairs Shortest Path
- Breadth & Depth First Search
- Delta-Stepping Single-Source
- Dijkstra Single-Source
- Dijkstra Source-Target
- K-Spanning Tree (MST)
- Minimum Weight Spanning Tree
- Random Walk
- Yen's K Shortest Path
- Minimum Directed Steiner Tree

Centrality & Importance

- ArticleRank
- Betweenness Centrality & Approx.
- Closeness Centrality
- Degree Centrality
- Eigenvector Centrality
- Harmonic Centrality
- Hyperlink Induced Topic Search (HITS)
- Influence Maximization (Greedy, CELF)
- PageRank
- Personalized PageRank

Community Detection

- Conductance Metric
- K-1 Coloring
- K-Means Clustering
- Label Propagation
- Leiden Algorithm
- Local Clustering Coefficient
- Louvain Algorithm
- Max K-Cut
- Modularity Optimization
- Speaker Listener Label Propagation
- Strongly Connected Components
- Triangle Count
- Weakly Connected Components

Supervised Machine Learning

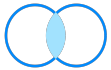
- Link Prediction Pipelines
- Node Classification Pipelines
- Node Property Regression Pipelines
- Logistic Regression, RandomForest and MLPs

… and more!

- Collapse Paths
- One Hot Encoding
- Pregel API (write your own algos)
- Property Scaling
- Split Relationships
- Synthetic Graph Generation

Heuristic Link Prediction

- Adamic Adar
- Common Neighbors
- Preferential Attachment
- Resource Allocations
- Same Community
- Total Neighbors



Similarity

- K-Nearest Neighbors (KNN)
- Node Similarity
- Filtered KNN & Node Similarity
- Cosine & Pearson Similarity Functions
- Euclidean Distance Similarity Function
- Euclidean Similarity Function
- Jaccard & Overlap Similarity Functions

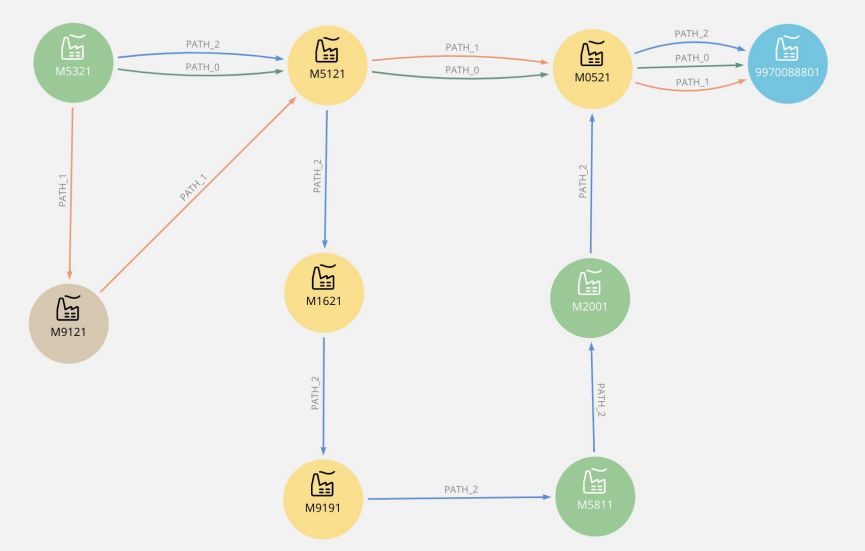
Graph Embeddings

- Fast Random Projection (FastRP)
- GraphSAGE
- Node2Vec
- HashGNN

Pathfinding

Pathfinding and Graph Search algorithms are used to identify optimal routes, and they are often a required first step for many other types of analysis.

Applications: Shortest path, Optimal paths, route availability, What-if analysis, Alternate Routing, Disaster Recovery



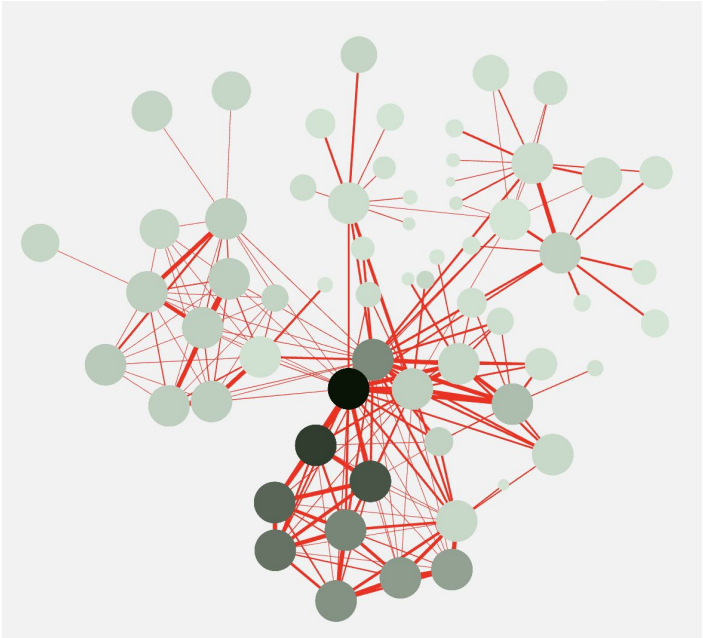
https://neo4j.com/docs/graph-data-science/current/algorithms/pathfinding/

Centrality

Finds important nodes based on relationships to other nodes in the graph

Applications: Outlier detection, preprocessing, Influencer detection, Bridge points, points of failure, vulnerabilities

https://neo4j.com/docs/graph-data-science/current/algorithms/centrality/



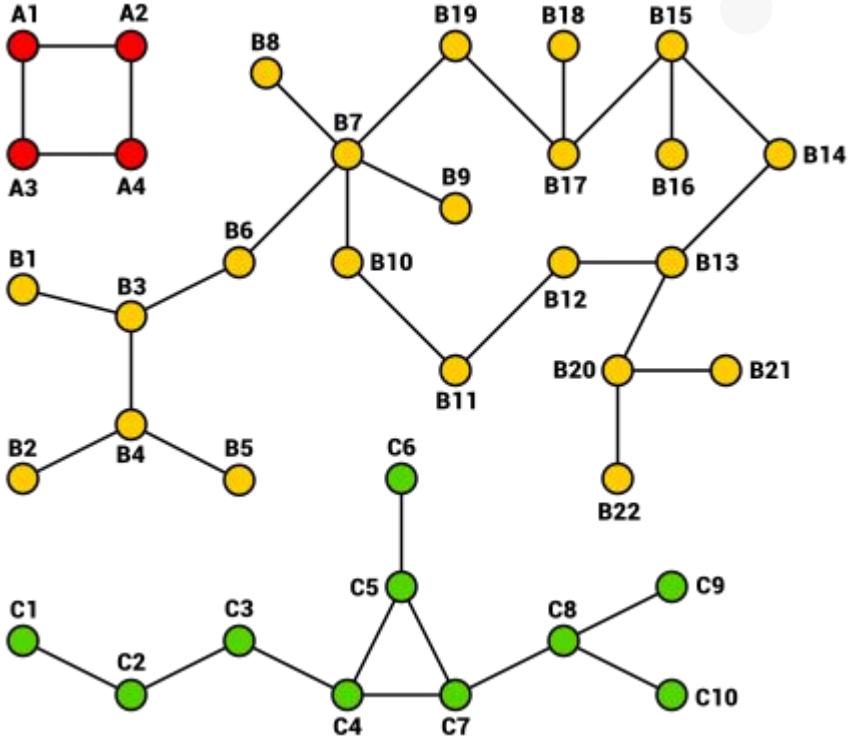
Color and size represent influence based on centrality scores

Community Detection

Evaluates how groups of nodes are clustered or partitioned, as well as their tendency to strengthen or break apart

Applications: Recommendations, homogeneity, disjoint communities, outlier detection, preprocessing

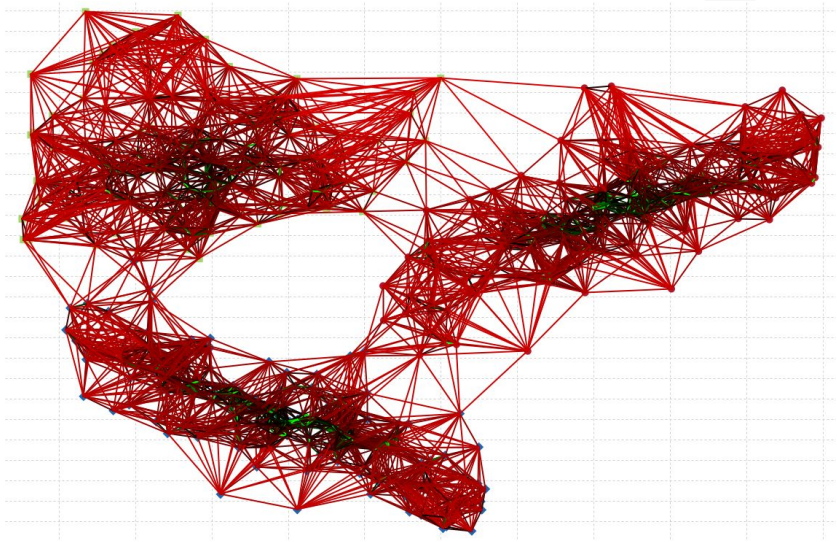
https://neo4j.com/docs/graph-data-science/current/algorithms/community/



Similarity

Evaluates how alike nodes are at an individual level either based on node attributes, neighboring nodes, or relationship properties.

Applications: Recommendations, What-if analysis, Disambiguation

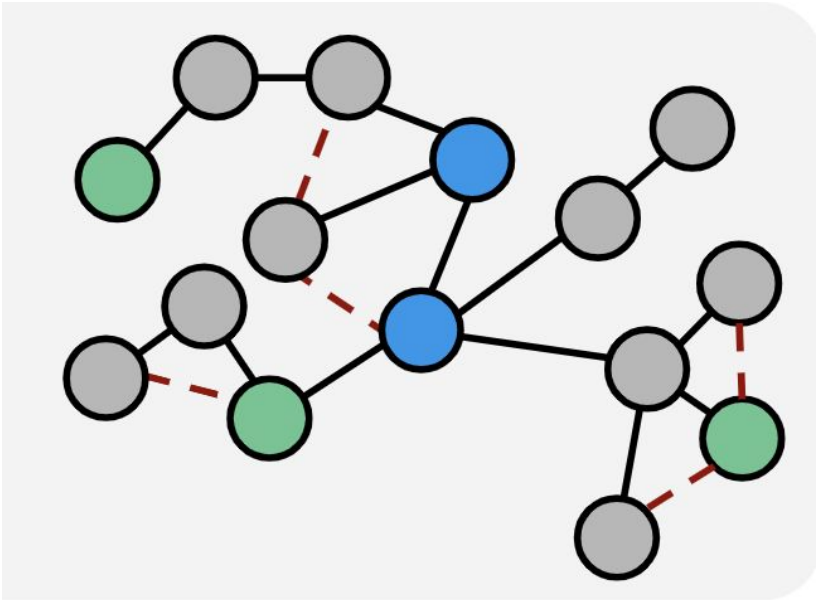


https://neo4j.com/docs/graph-data-science/current/algorithms/similarity/

Link Prediction

These methods compute a score for a pair of nodes, where the score could be considered a **measure of proximity** or “similarity” between those nodes **based on the graph topology**.

Applications: Context-enrichment, contact tracing, spam detection, association in social networks etc.

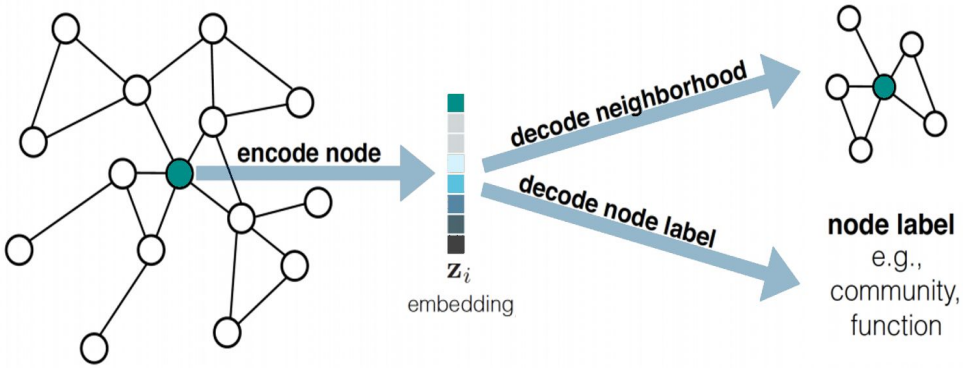


https://neo4j.com/docs/graph-data-science/current/algorithms/linkprediction/

Embeddings

A ***graph embedding*** is a way of representing each node in your graph as a fixed-length vector.

- Preserves key features
- Reduces dimensionality
- Can be decoded



Different techniques may represent different aspects of a graph, and may use different approaches to learn that representation

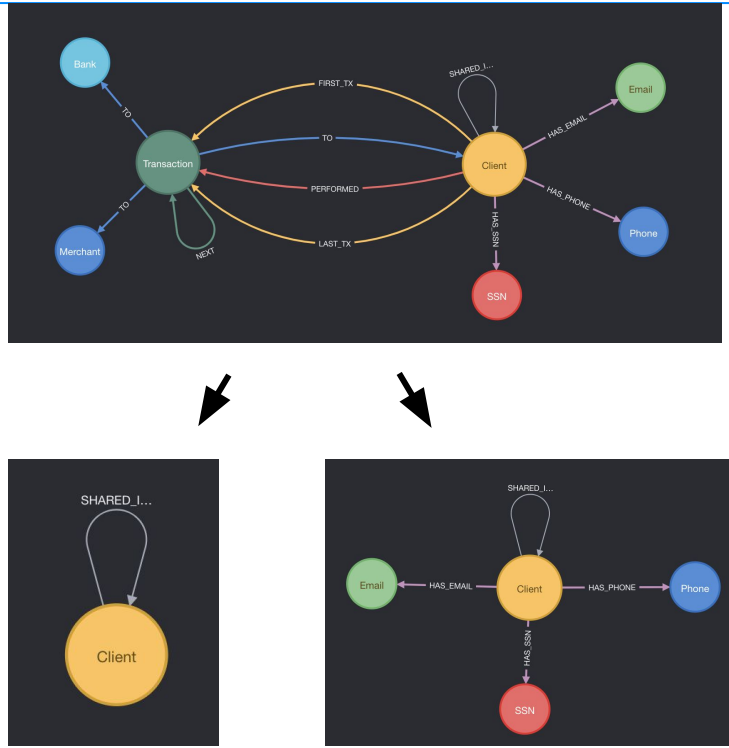
Neo4j Graph Data Science: Internals & Running Algorithms

Graph projections

Graph algorithms run on a *projection* of the stored graph.

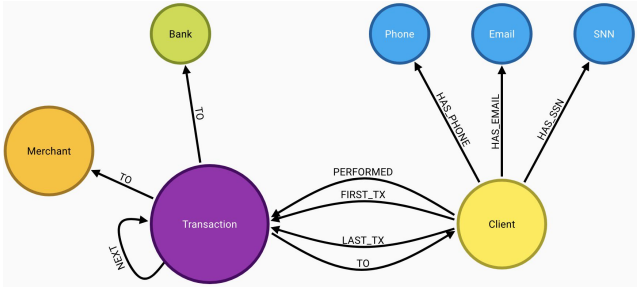
Graph projection

- is a materialized view/snapshot over the stored graph
- contains only analytically relevant topological and property information
- is stored entirely in-memory using compressed data
- is optimized for topology and property lookup operations



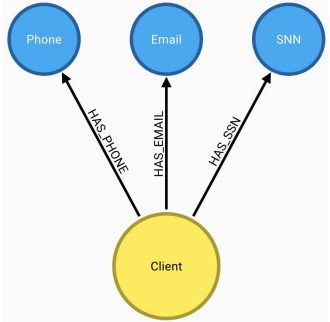
Graph projections

Multipartite



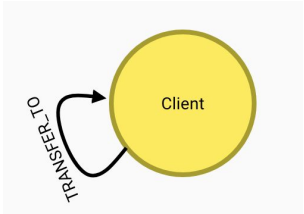
Contains data with multiple node and relationship types

Bipartite



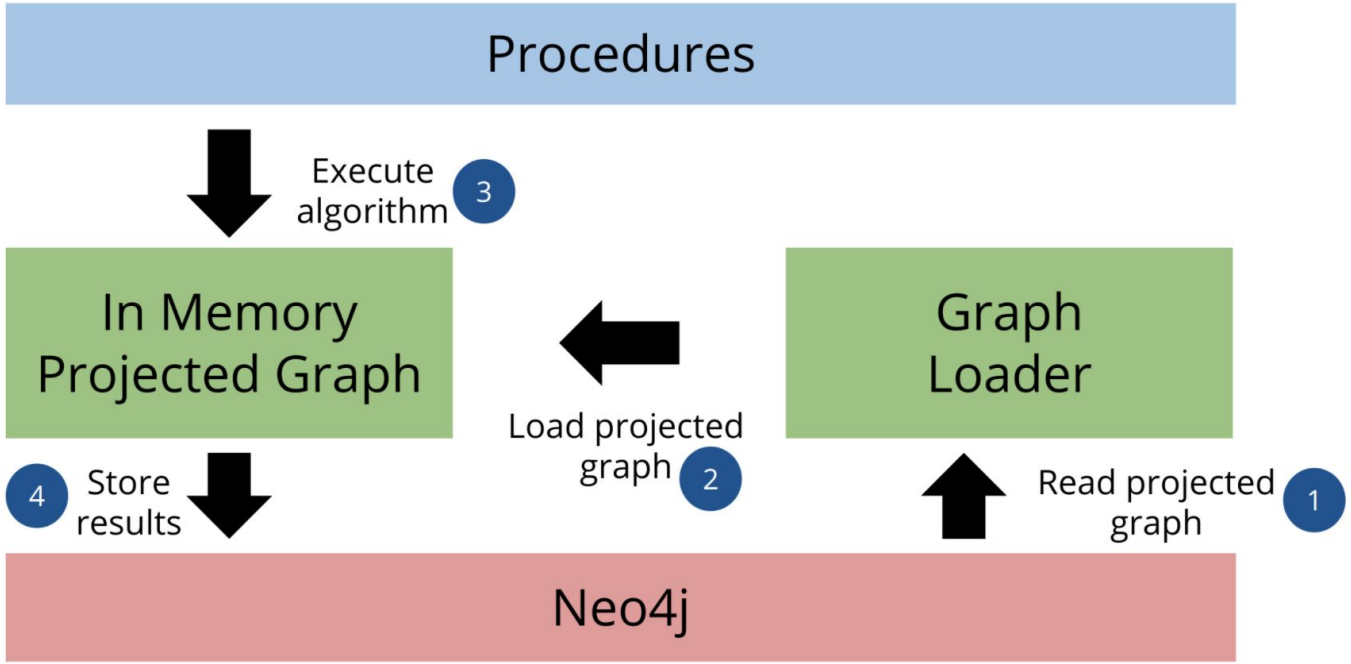
Contains nodes that can be divided into two sets, such that relationships only exists between sets but not within each set

Monopartite



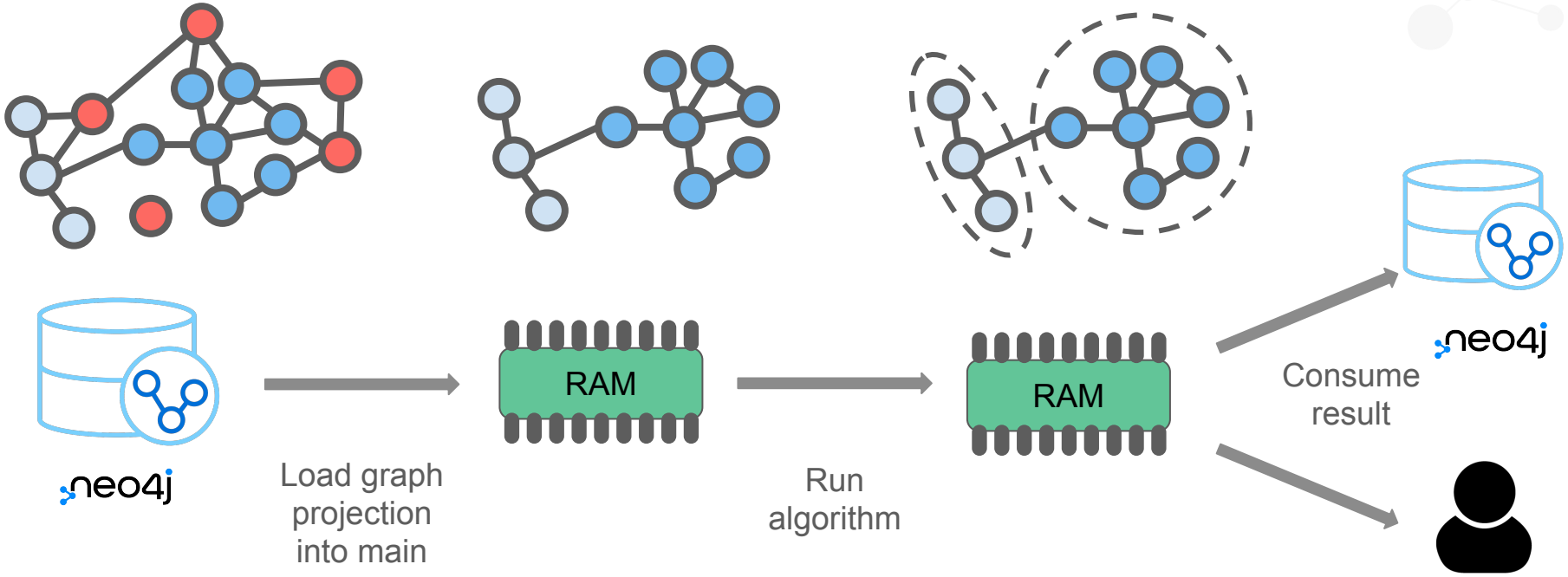
Contains one node label and relationship. Most algorithms rely on this type of graph

Graph Algorithms Execution Workflow

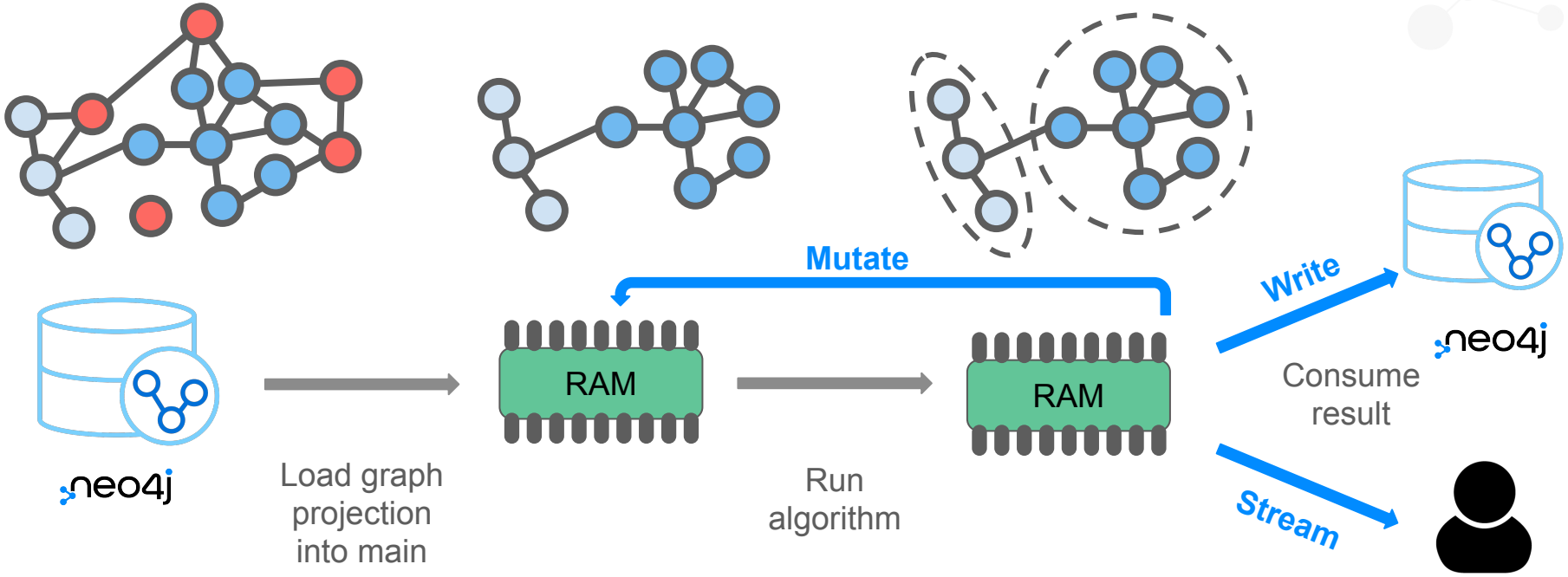


https://neo4j.com/docs/graph-data-science/current/common-usage/

Graph Algorithms Execution Workflow



Graph Algorithms Execution Workflow



Graph Catalog

Allows managing multiple named graphs

A Named graph

- is a graph projection with a name
- can be generated by Native projection or a Cypher Projection
- Stored in the catalog and are accessible by its name
- can be used many times in the analytical workflow
- can be mutated to store intermediate results

https://neo4j.com/docs/graph-data-science/current/management-ops/graph-catalog-ops/

Native and Cypher Projection

Native projections provide the best performance by reading from the Neo4j store files

```
CALL gds.graph.create(
    graphName: String,
    nodeProjection:
                String or List or Map,
    relationshipProjection:
                String or List or Map,
    configuration: Map
)
YIELD
  graphName: String,
```

Cypher projections is a more flexible and expressive approach with diminished focus on performance

```
CALL gds.graph.create.cypher(
    graphName: String,
    nodeQuery: String,
    relationshipQuery: String,
    configuration: Map
) YIELD
    graphName: String
```

https://neo4j.com/docs/graph-data-science/current/management-ops/projections/graph-project/

Graph Export

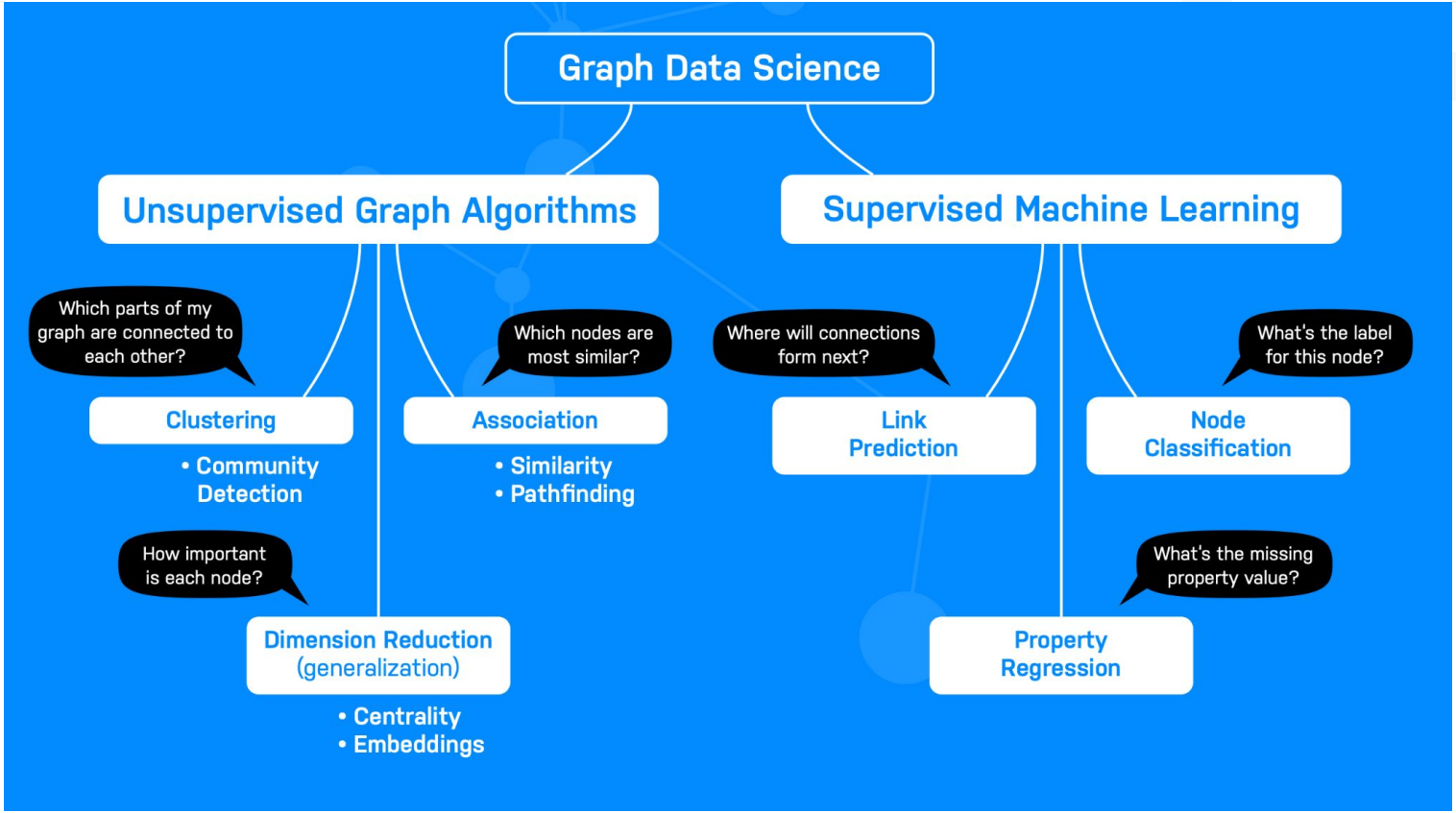
Creates new Neo4j databases from named in-memory graphs or saves in-memory graphs to disk

- Avoid heavy write load on the operational system by exporting the data instead of writing back
- Create an analytical view of the operational system that can be used as a basis for running algorithms
- Produce snapshots of analytical results and persistent them for archiving and inspection
- Share analytical results within the organization

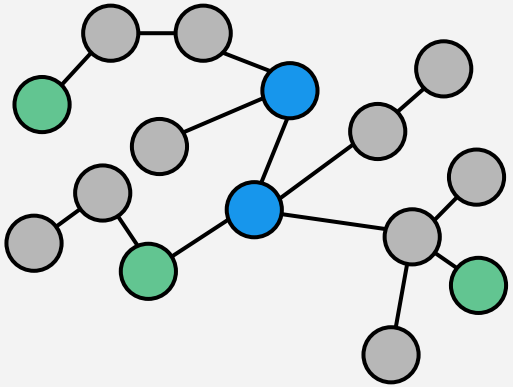
https://neo4j.com/docs/graph-data-science/current/graph-catalog-export-ops/

Neo4j Graph Data Science #3: Graph Machine Learning and Integrations with ML frameworks

Graph Algorithms and Supervised ML



In-graph Machine Learning

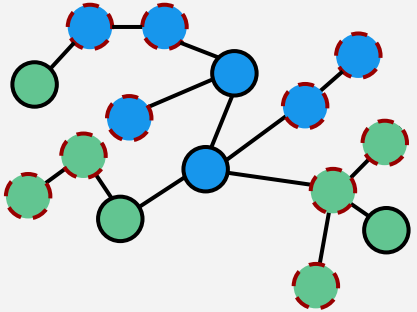


Labeled data for link prediction: Pairs of nodes that are either linked or not

Labeled data for node classification: key-value pairs as node properties

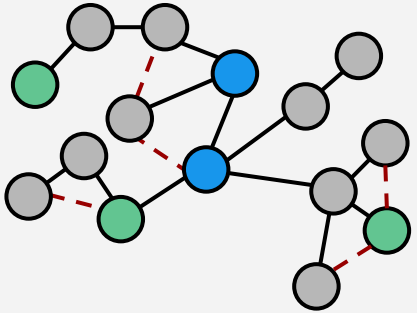
$f(x)$

Node classification:
"What kind of node is this?"



$f(x)$

Link prediction:
"Should there be a relationship between these nodes?"



neo4j

Graphy features

Queries

Human-crafted query, human-readable result

```
MATCH (p1:Person)-[:ENEMY]->(:Person)<-[:ENEMY]-(p2:PERSON)
MERGE (p1)-[:FRIEND]->(p2)
```

Algorithms

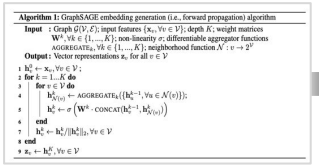
Predefined formula, human-readable result

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

```
PageRank(Emil)    = 13.25
PageRank(Amy)     =  4.83
PageRank(Alicia)  =  4.75
```

Embeddings

AI-learned formula, machine-readable result



Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

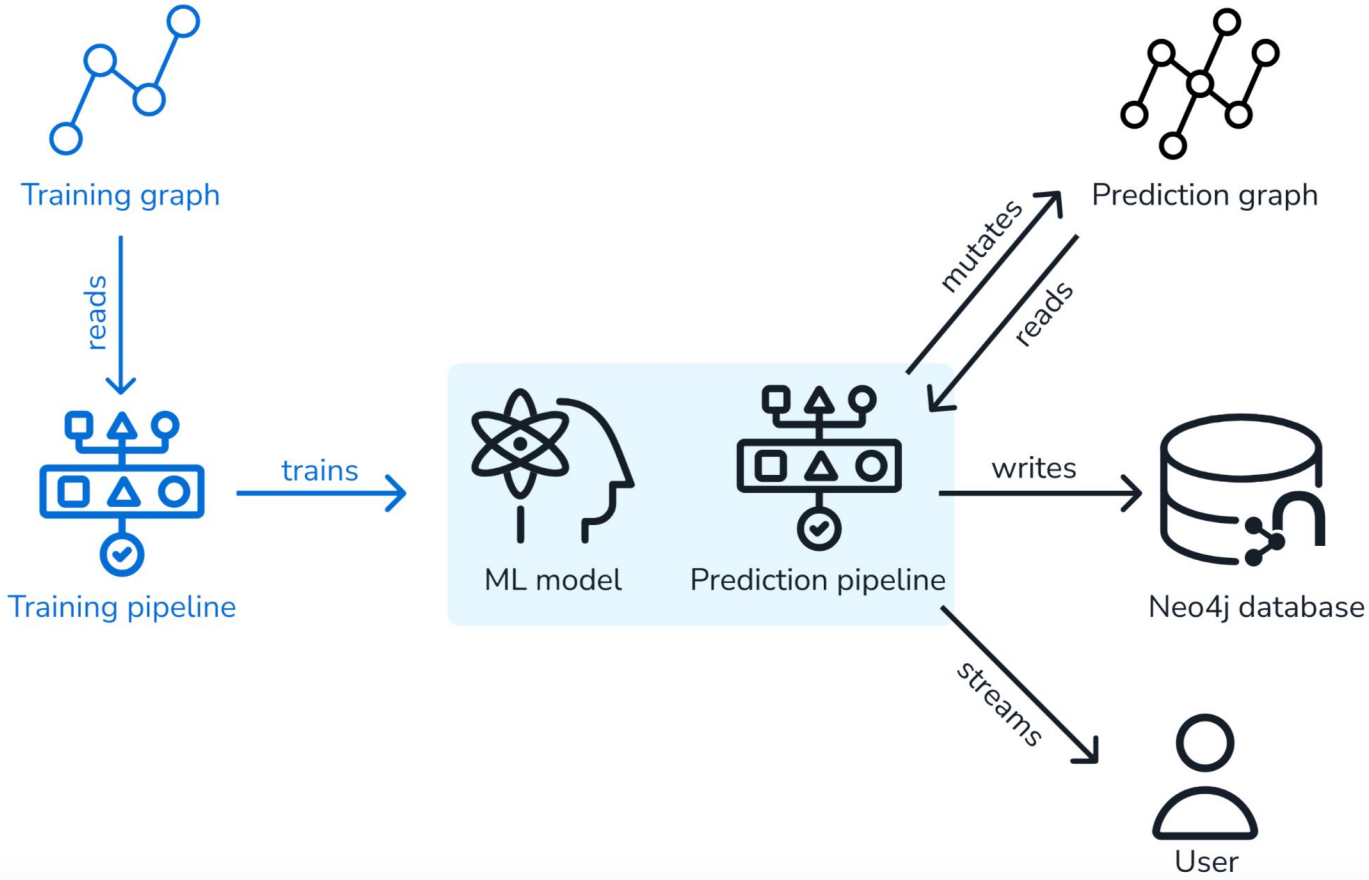
```
Node2Vec(Emil)    =[5.4 5.1 2.4 4.5 3.1]
Node2Vec(Amy)     =[2.8 1.8 7.2 0.9 3.0]
Node2Vec(Alicia)  =[1.4 5.2 4.4 3.9 3.2]
```

Machine Learning Workflows

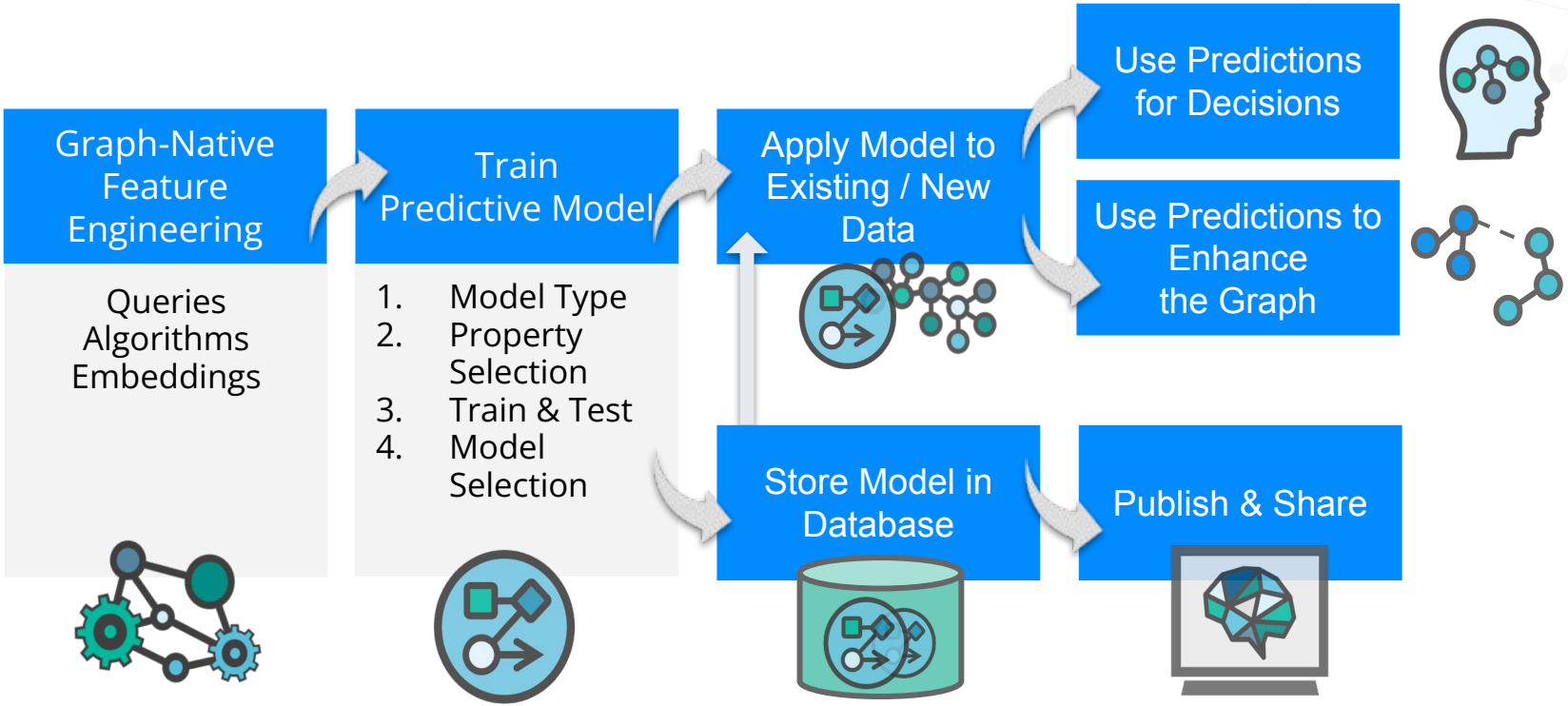
Train ML models based on results

$$f(x)$$

ML Pipelines in Neo4j



Supervised ML in Neo4j

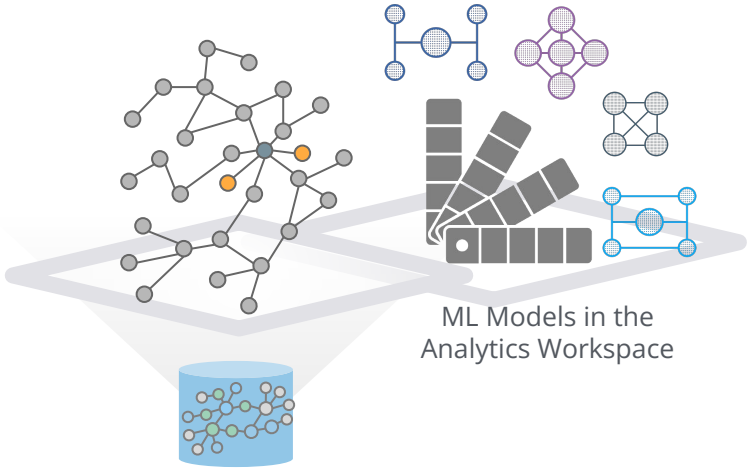


The Only Completely In-Graph, ML Workflow

Model Catalog

Allows storing and managing multiple trained models by name

- Lives in the Neo4j analytics workspace in a model catalog
- Model catalog is lost when the database is restarted



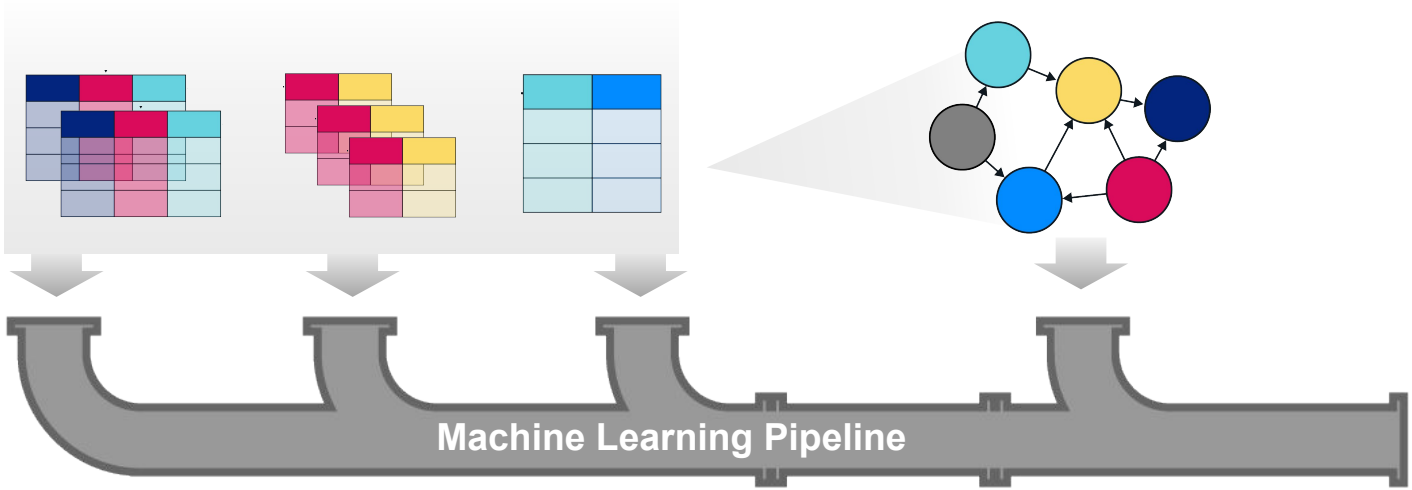
https://neo4j.com/docs/graph-data-science/current/model-catalog/

Contains versioning information, Time stamps, Model names and Metadata

neo4j

Integration with ML frameworks

- Traditional ML ignore network structure because it's difficult to extract
- Add graphy data to existing ML pipelines to increase accuracy, *or*
- Graphs use relationships to unlock otherwise unattainable predictions



GDS Python Client

Built for Data Scientists

- **Native Python Client**
- 65+ Pretuned Algos
- Single API for load, analysis & writeback
- Production tier graduation

Connect to Graph Data Science (GDS)

```
In [3]: from graphdatascience import GraphDataScience

        # Use Neo4j URI and credentials according to your setup
        gds = GraphDataScience(HOST, auth=(USERNAME, PASSWORD), aura_ds=True)
```

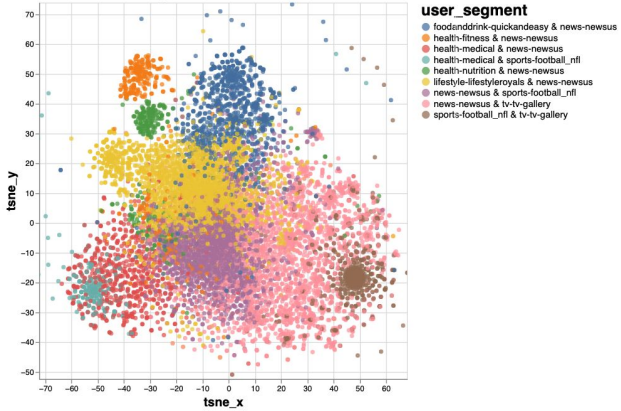
Apply GDS for Automated Graph Feature Engineering

FastRP transforms the graph of interconnected user activity into embedding features for EDA & ML applications

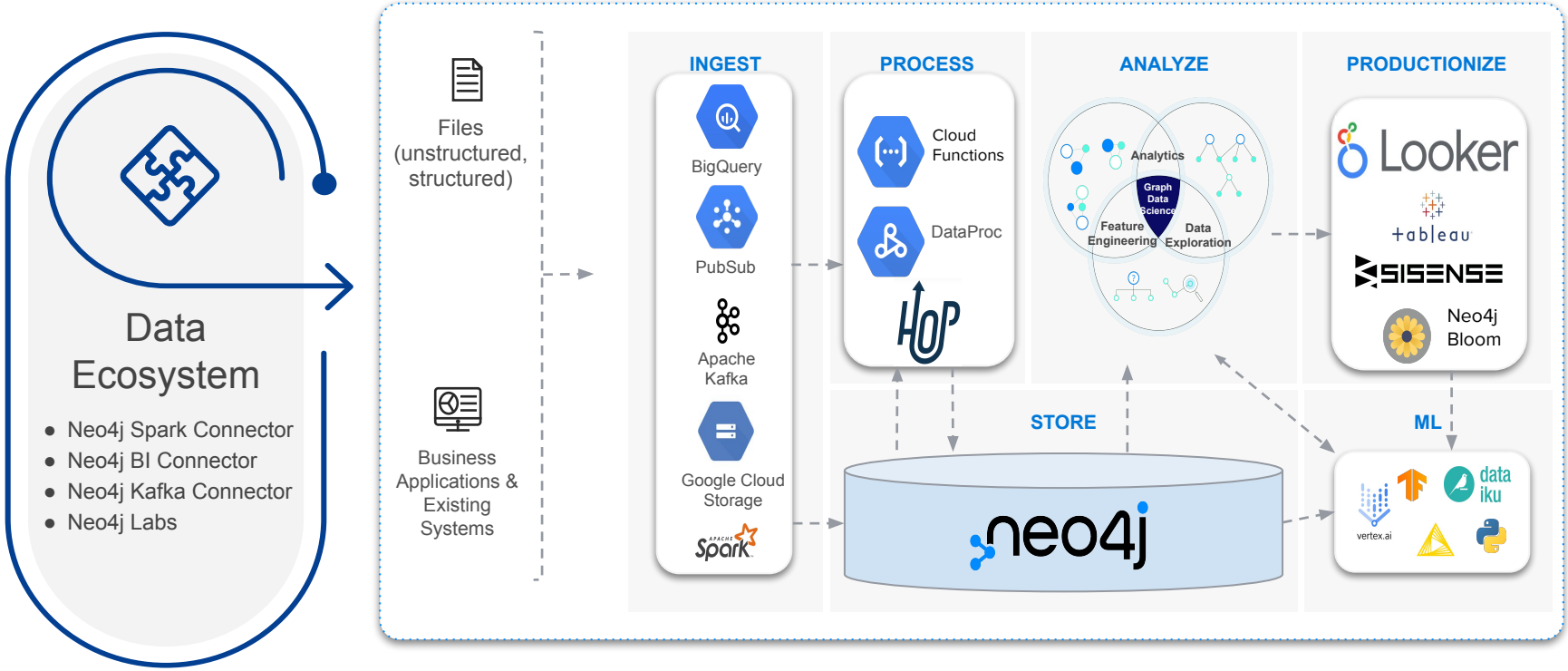
```
In [4]: g, _ = gds.graph.project('proj',['User','News'], {'CLICKED': {'orientation': 'UNDIRECTED'}})
        gds.fastRP.mutate(g,mutateProperty='segmentEmbedding',embeddingDimension=128,randomSeed=7474)
        gds.graph.writeNodeProperties(g, ['segmentEmbedding'], ['User'])

Out[4]: writeMillis                     1212
        graphName                       proj
        nodeProperties       [segmentEmbedding]
        propertiesWritten              750434
        Name: 0, dtype: object
```

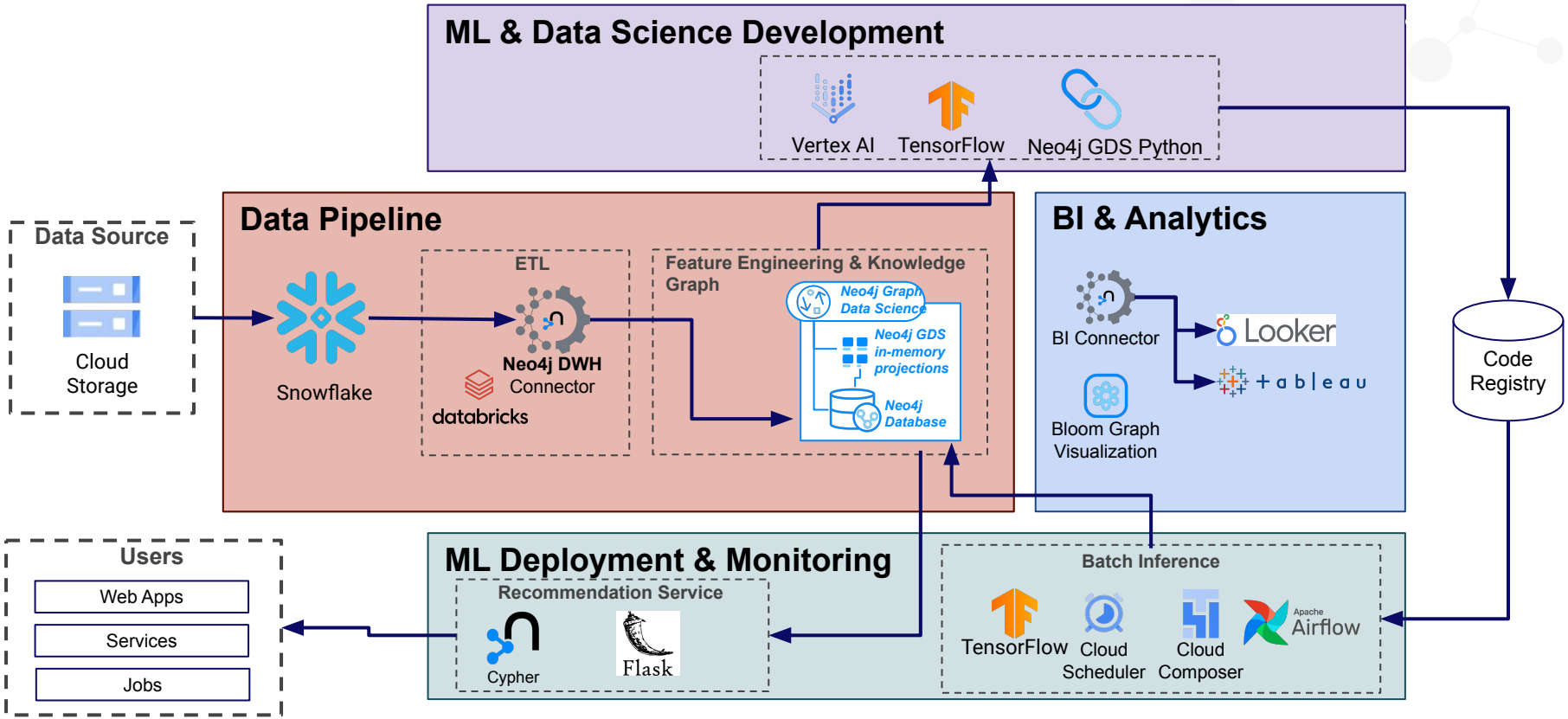
User Segmentation Clusters



Connector ecosystem



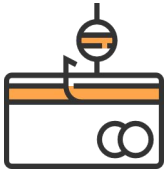
Integrated architecture for Contextual ML predictions



Part 4 of Neo4j Graph Data Science: Applied Examples

Graph data science applications

Fraud Detection



Disambiguation & Segmentation

Personalized Recommendations

Life Sciences

Churn Prediction

Search & Master Data Mgmt.

Predictive Maintenance

Cybersecurity

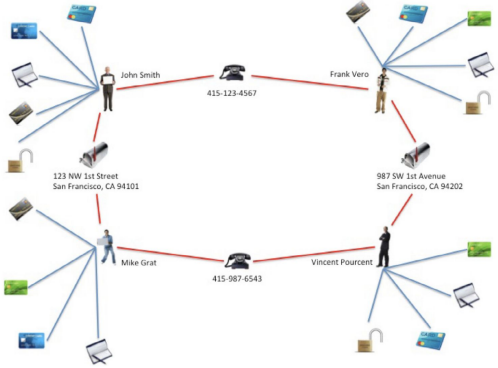
Top 10 US Bank

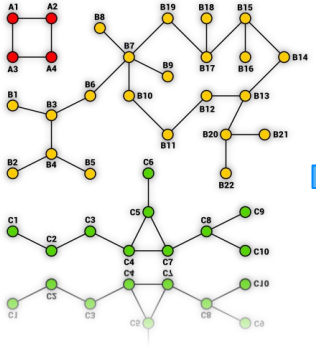
- **Challenge:** Hundreds of millions of dollars in fraud goes undetected and unstopped. Previously, they were using SQL and manual review to identify suspicious patterns.
- **Solution:** Use Neo4j to identify contacts of fraudsters, first party fraud, fraud rings.
- **Results:**
  - Identified **tens of millions of dollars** in previously undetectable fraud
  - Enabled analysts to review results directly in the graph and proactively stop fraud.

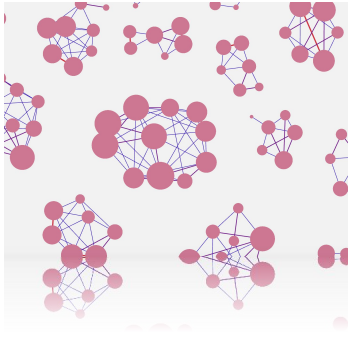
neo4j

Fraud Detection

Graph algorithms for entity link analysis detect first party and synthetic identity fraud across channels in financial sector and other industries







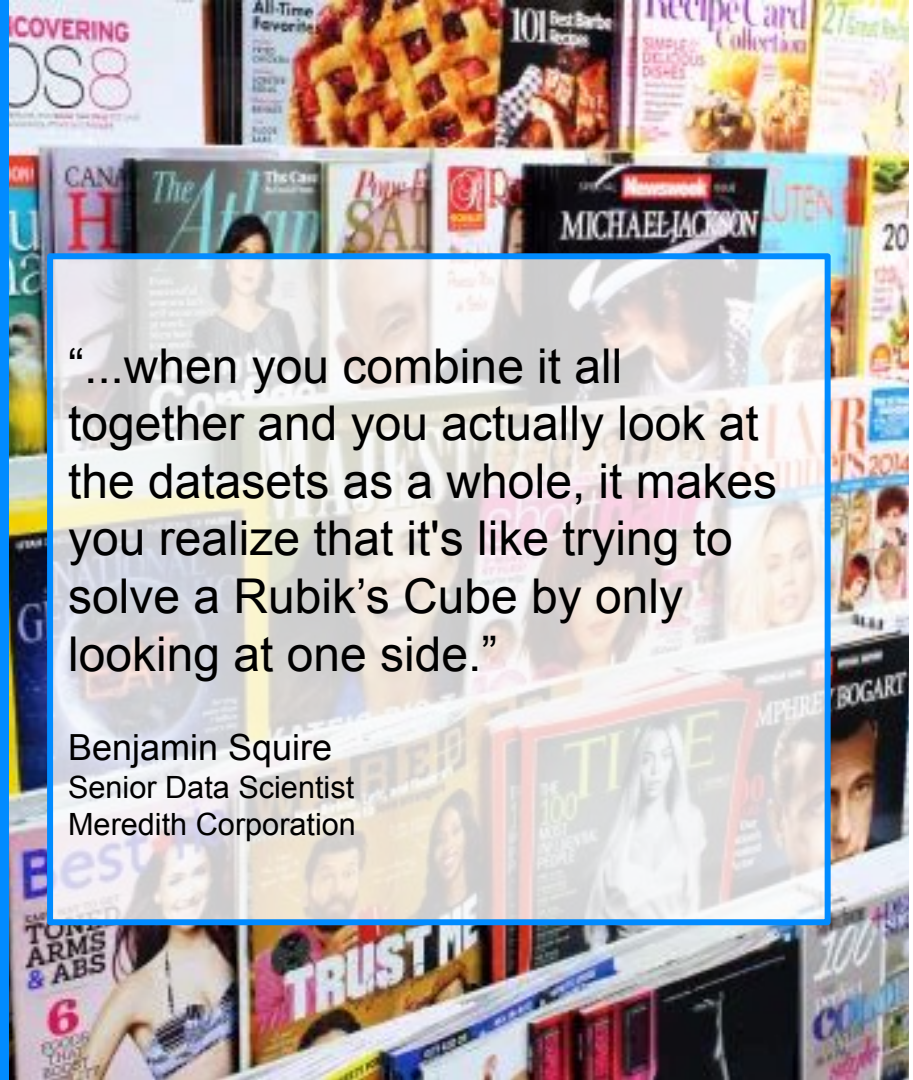
Detect patterns of fraudulent activity across channels

Isolate fraudsters by running community detection, centrality and embedding algorithms

Identify potential fraudsters by computing similarities between known fraudsters and customers

Prevent fraud by flagging transactions and clients with higher risk

neo4j



"...when you combine it all together and you actually look at the datasets as a whole, it makes you realize that it's like trying to solve a Rubik's Cube by only looking at one side."

Benjamin Squire
Senior Data Scientist
Meredith Corporation

meredith

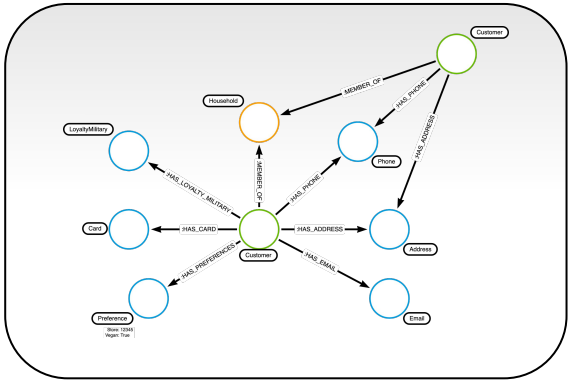
Meredith Corp
Identifying the Anonymous

- **Challenge:** It's hard to make recommendations to anonymous users
- **Solution:** Connected first and third party cookies using graph algorithms to create unique profiles
- **Results:**
  - Converted 14B anonymous data points into **163M unique user profiles**
  - User profiles - and personalized recommendations - **drove 612% increase in web traffic**

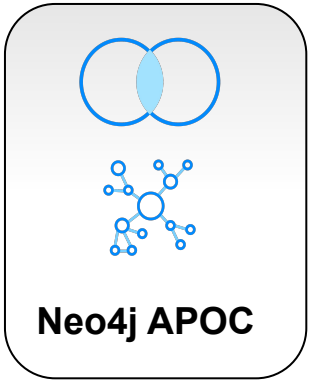
neo4j

Identity Management / Entity Resolution

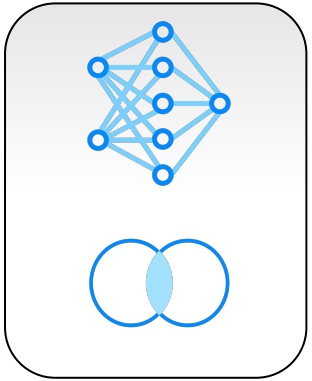
Graph algorithms and graph embeddings are used for generating context and resolving identities/entities



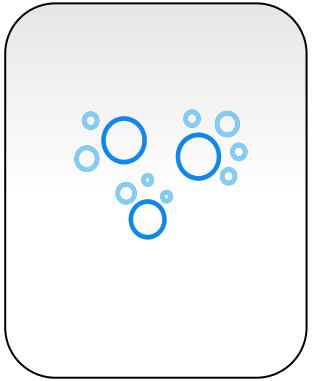
Capture relationships between entities across data sources using a knowledge graph



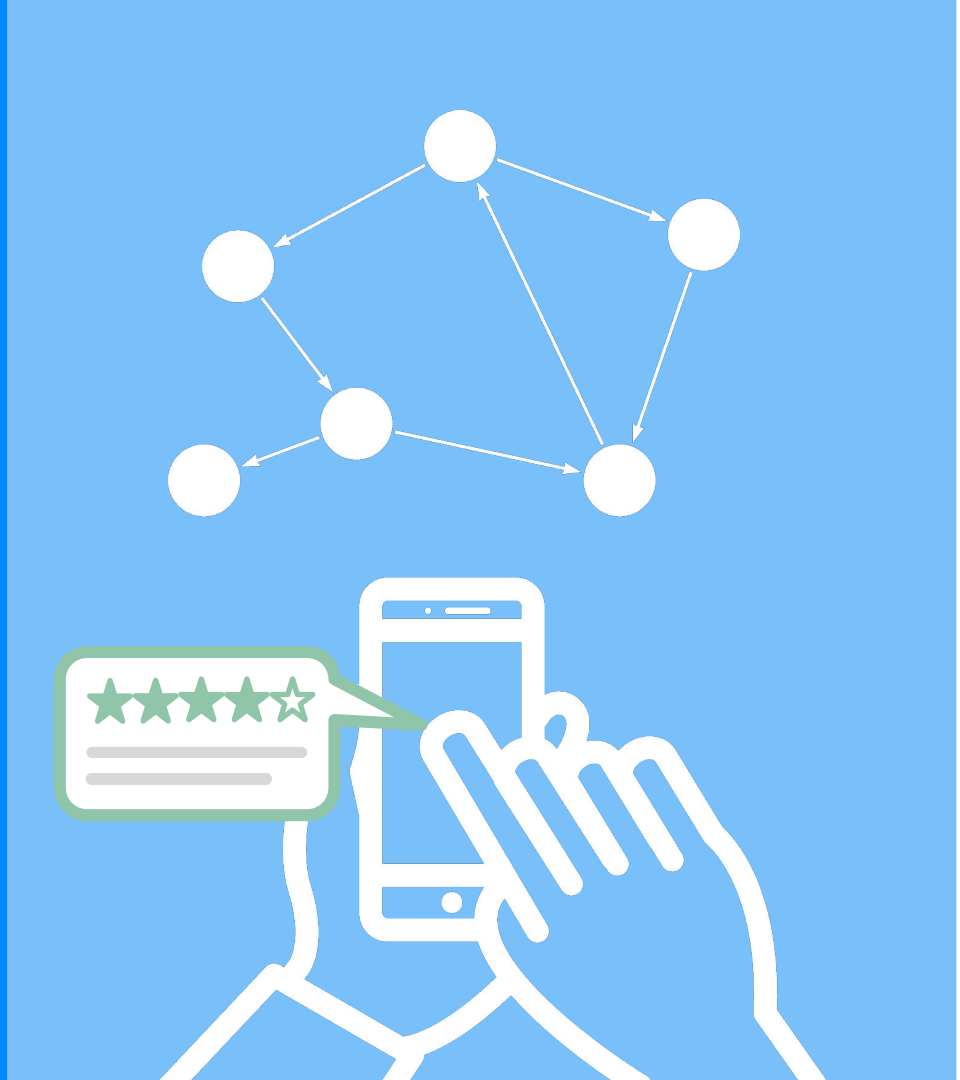
Create additional weighted relationships based on similar text description and/or other similar metadata



Construct node embeddings and resolve entities based on weighted pairwise similarity between various entities



Identify communities of entities based on distance between node embeddings

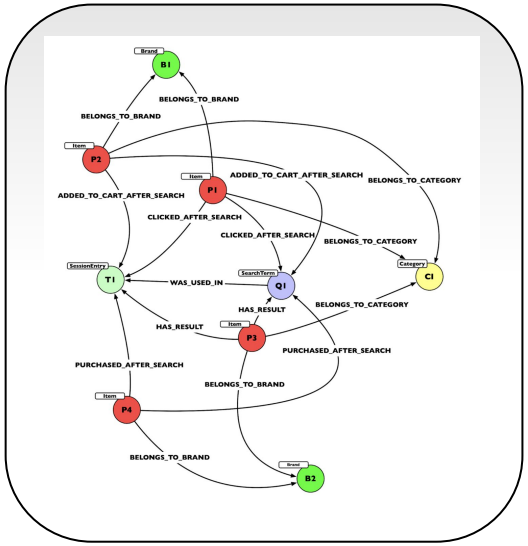


Top US Retailer: Improving Recommendations

- **Challenge:** Providing relevant search results and recommendations when sparse results
- **Solution:** Represent historical customer purchase behavior (views, clicks, purchases) alongside product ontology, train graph embeddings to provide recommendations based on graph topology
- **Results:**
  - Recommendations that **can account for behavior and preferences** of similar users
  - **Relevant recommendations or sparse and unusual search terms -** that might have had few results, historically

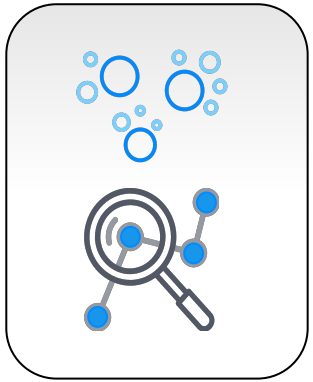
Personalized Recommendations

Graph algorithms and graph embeddings are used for generating product recommendations and improving search relevance

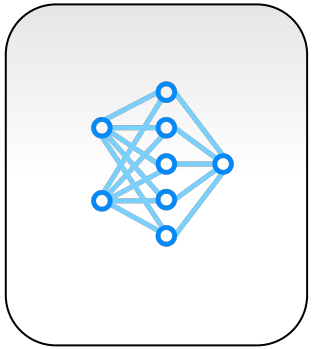


Capture customer interactions and customer journey using a knowledge graph

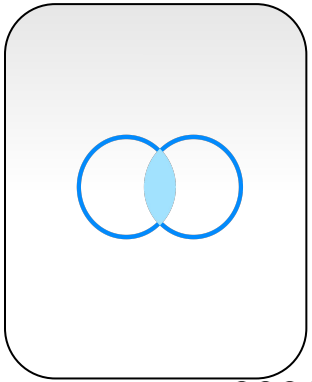
Analyze customer interactions using graph queries and find customer communities based on common purchase behavior

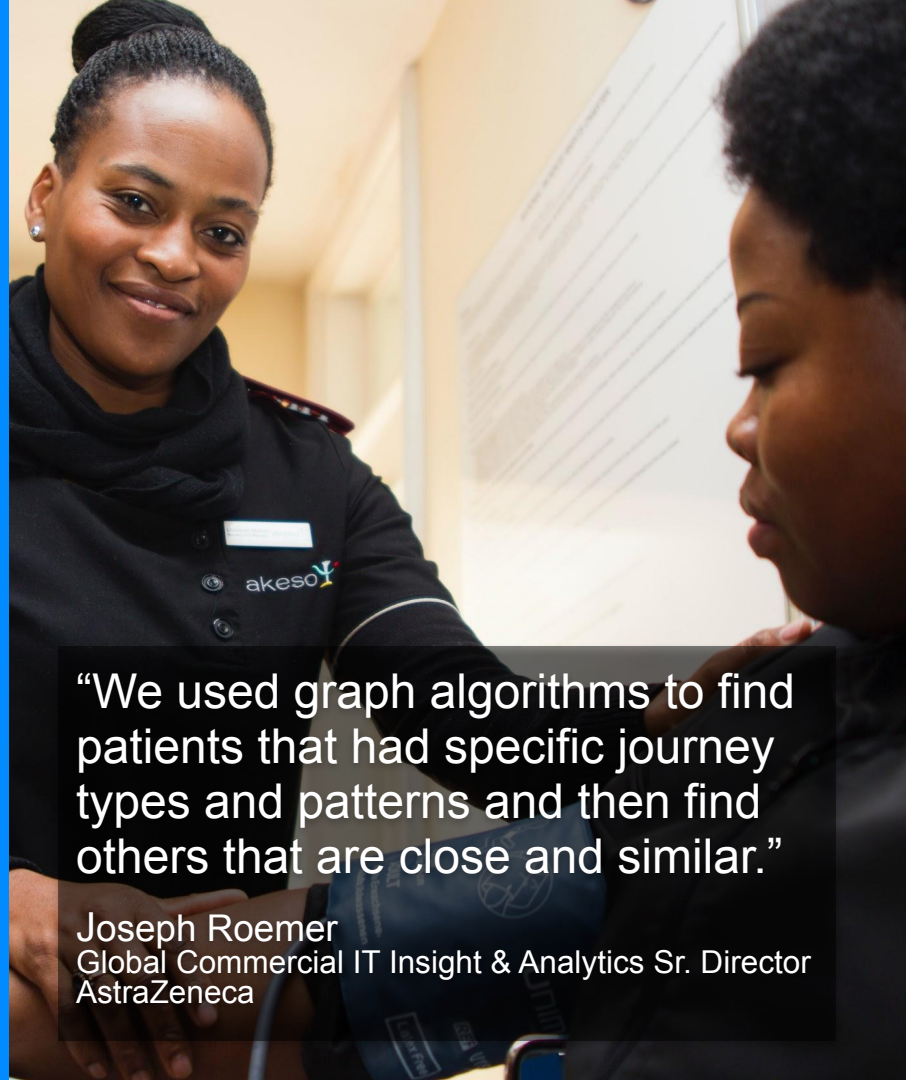


Construct node embeddings and resolve entities based on weighted pairwise similarity between various entities



Generate product recommendations based on correlations between products, search queries and historical purchases





"We used graph algorithms to find patients that had specific journey types and patterns and then find others that are close and similar."

Joseph Roemer
Global Commercial IT Insight & Analytics Sr. Director
AstraZeneca

AstraZeneca Patient Journey

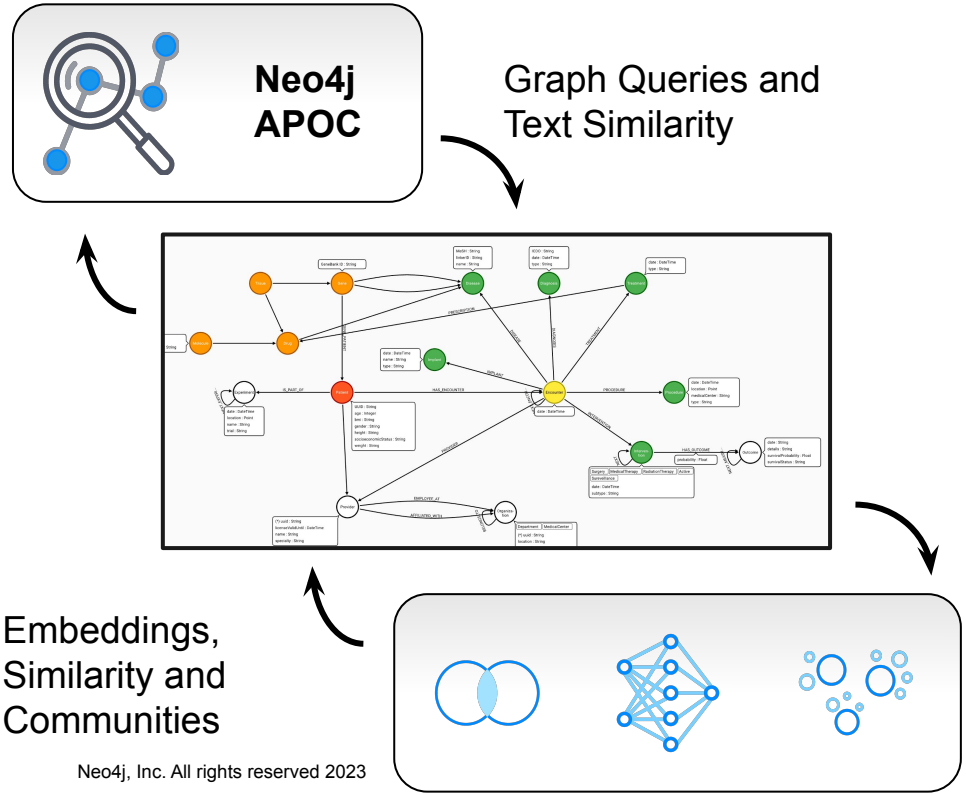
AstraZeneca

- **Challenge:** How to best intervene sooner for complex diseases that develop over years
- **Solution:** Neo4j knowledge graph of 3 yrs of visits, tests, & diagnosis with 10's Bn of records. Using graph algorithms and machine learning together.
- **Results:**
  - **Identified journey archetypes and patterns** using graph feature engineering as input to ML
  - Revealed journey **similarities over time** with community detection
  - **Found influential touch-points** in the journey using graph algorithms

neo4j

Life Sciences: Patient care

Graphs and graph algorithms are used for capturing the events in patient journeys and analyze them for proactive monitoring



- Build heterogeneous knowledge graphs with rich metadata by ingesting from structured and unstructured data sources
- Define and capture significant events in patient journeys and connect them to build patient medical profiles
- Preprocess the data by building additional relationships based on text description and/or other metadata (Context-enrichment)
- Identify similar patients based on their medical profiles, generate embeddings and find communities of patients based on similar profiles using community detection algorithms

Top Graph Data Science Applications in Supply Chain

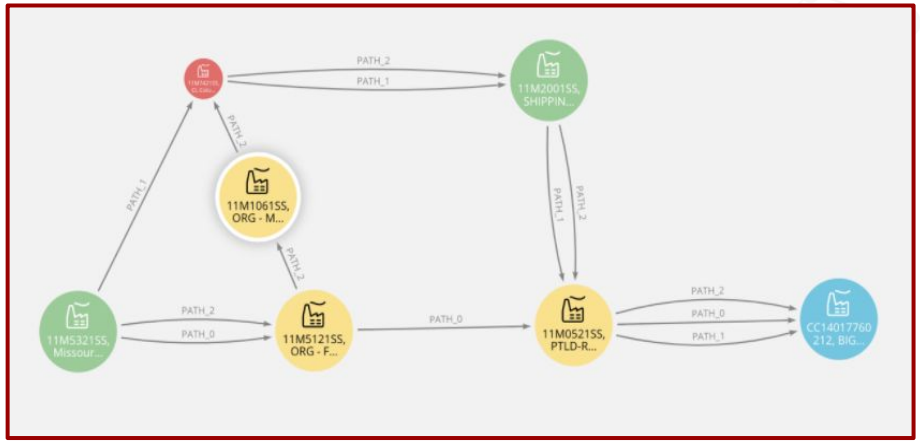
Supply Chain Optimization
Logistics
BoM Management

- Route optimization
- Predictive fulfillment
- Risk identification and diversity planning
- Quality recommendation
- Supply chain driven product design

Graph Algorithms in Supply Chains

Graph algorithms enable reasoning about **network structure**

K-Shortest Paths to identify the best alternative routes



"Site IDs"	"costs"	"totalCost"
["M5321","M5121","M0521","9970088801"]	[442.8,2671.2000000000003,198.0]	3312.0000000000005
["M5321","M7421","M2001","M0521","9970088801"]	[1810.8,1098.0,1575.0,198.0]	4681.8
["M5321","M7421","M1061","M0831","P1185","M2001","M0521","9970088801"]	[1810.8,810.0,352.8,0.0,657.0,1575.0,198.0]	5403.6

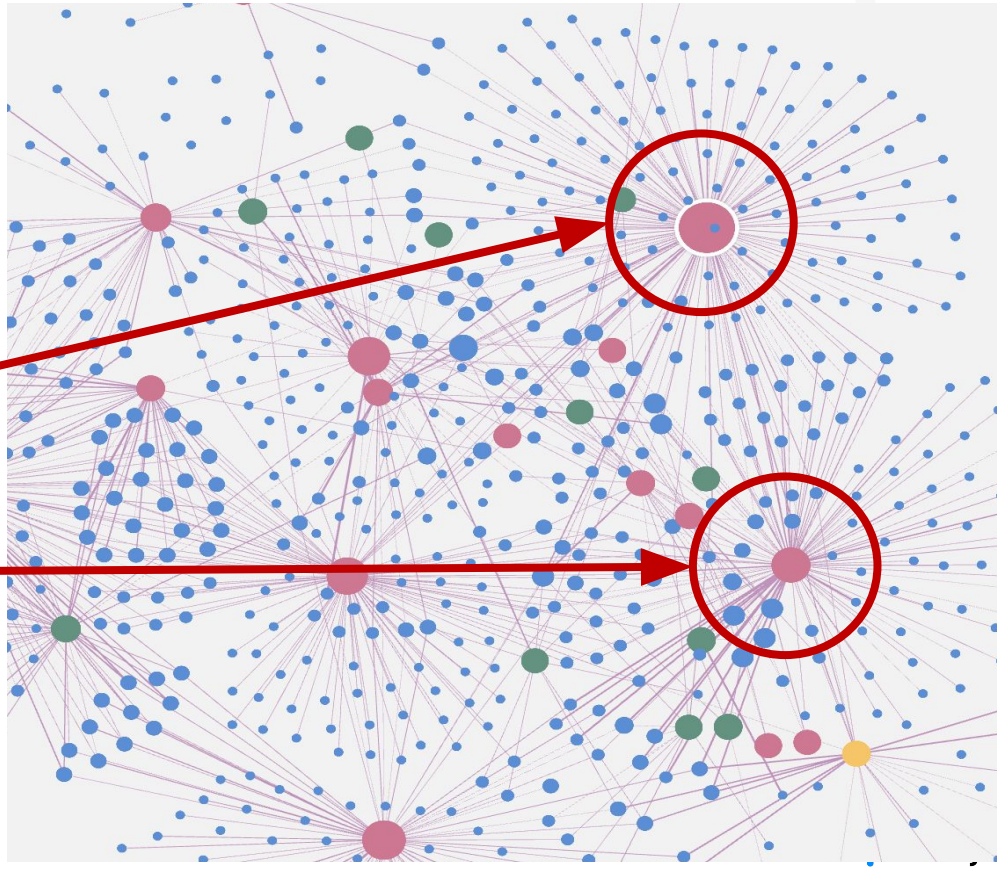
Graph Algorithms in Supply Chain

Graph algorithms enable reasoning about **network structure**

K-Shortest Paths to identify the best alternative routes

Betweenness Centrality to find critical bottlenecks or risk points

Degree Centrality to see distribution centers with high use



Graph Algorithms in Supply Chain

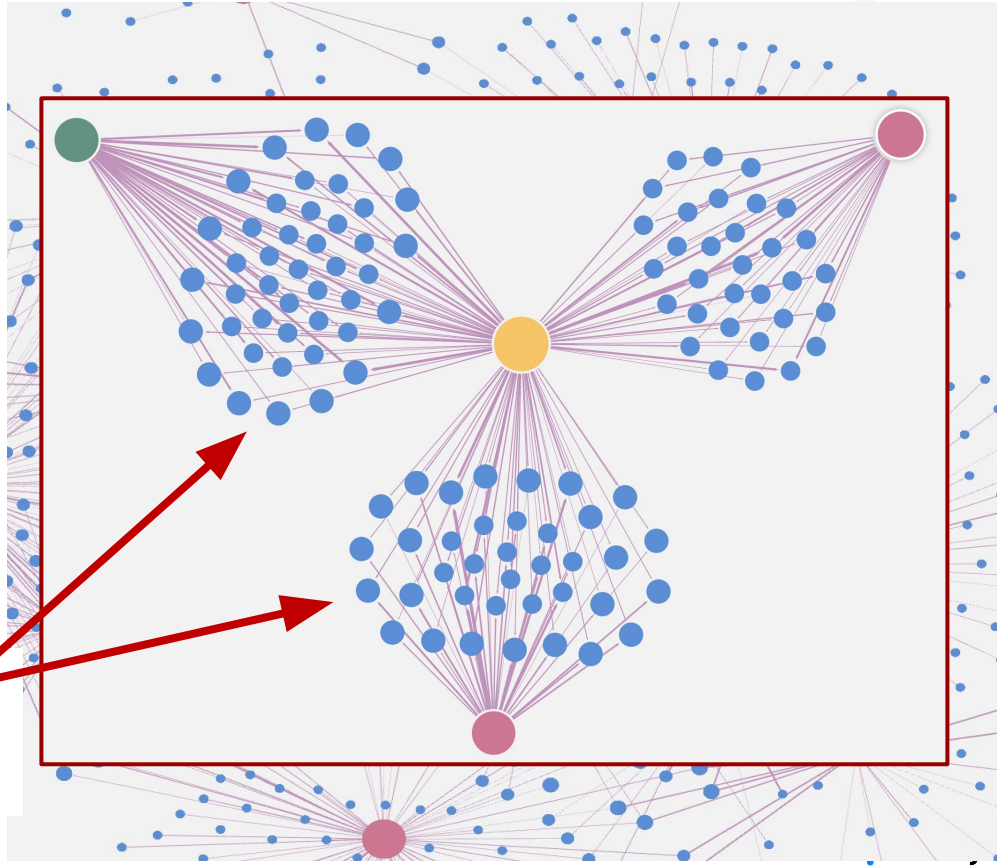
Graph algorithms enable reasoning about **network structure**

K-Shortest Paths to identify the best alternative routes

Betweenness Centrality to find critical bottlenecks or risk points

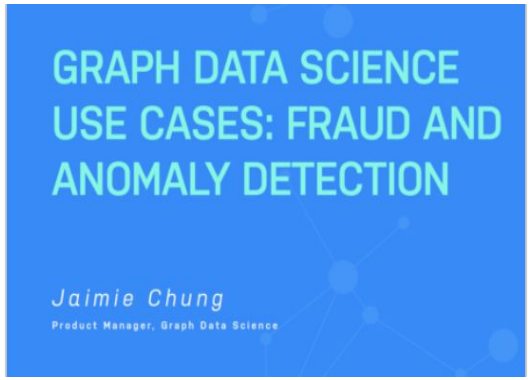
Degree Centrality to see distribution centers with high use

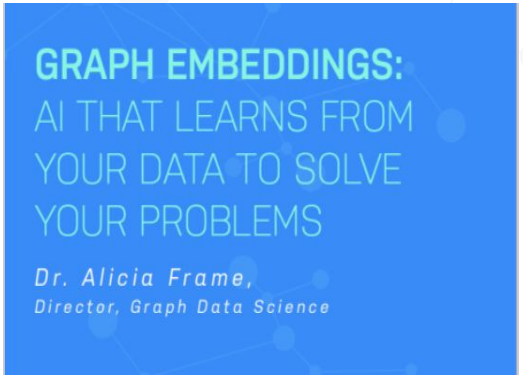
Similarity to find providers that can step in during a disruption

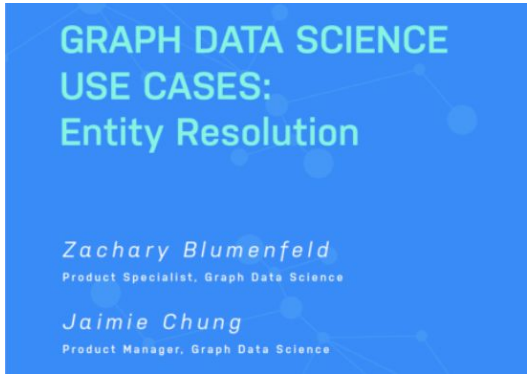


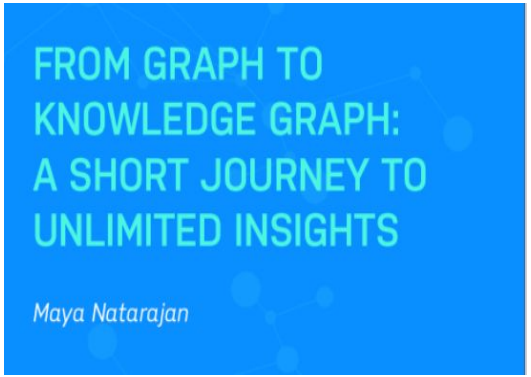
Resources

https://neo4j.com/product/graph-data-science/









Thank you!

https://neo4j.com/product/graph-data-science/