Knowledge Graphs: The Power of Graph-Based Search

Petra Selmer

Query Languages Standards & Research Group

petra.selmer@neo4j.com

19 September 2018

neo4j

Outline

The property graph data model

Introduction to Neo4j

Knowledge Graphs

- Background
- What is a Knowledge Graph?
- How do they work in Neo4j?
- Tools and techniques: a quick tour

Graph Search through Cypher

The property graph data model

neo4j

The property graph data model

Underlying construct is a *graph*

Nodes: represent entities of interest (vertices)

Relationships: connections between nodes (edges)

- Relate nodes by type and direction
- Add structure to the graph
- Provide semantic context for nodes

Properties: key-value pairs (map) representing the data

The property graph data model

A node may have 0 or more **labels**

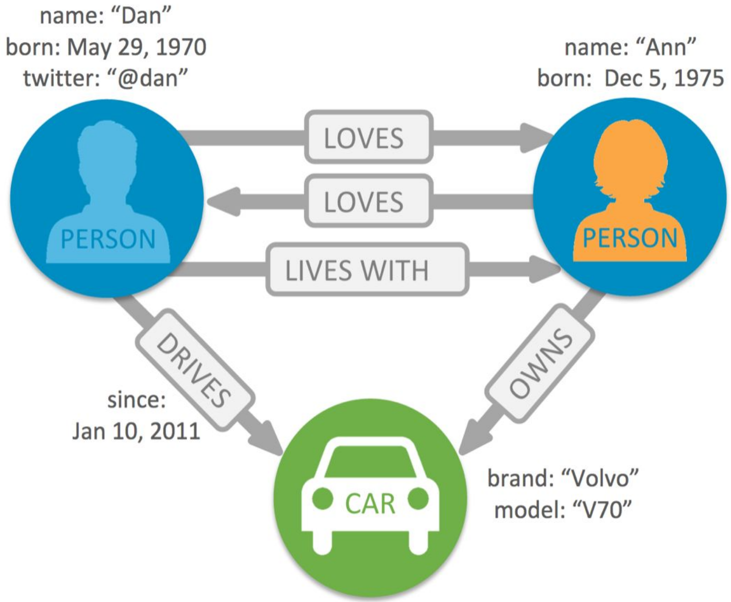
- E.g. `Person`, `Appliance`, `Teacher`

A relationship must have a **type** and **direction**

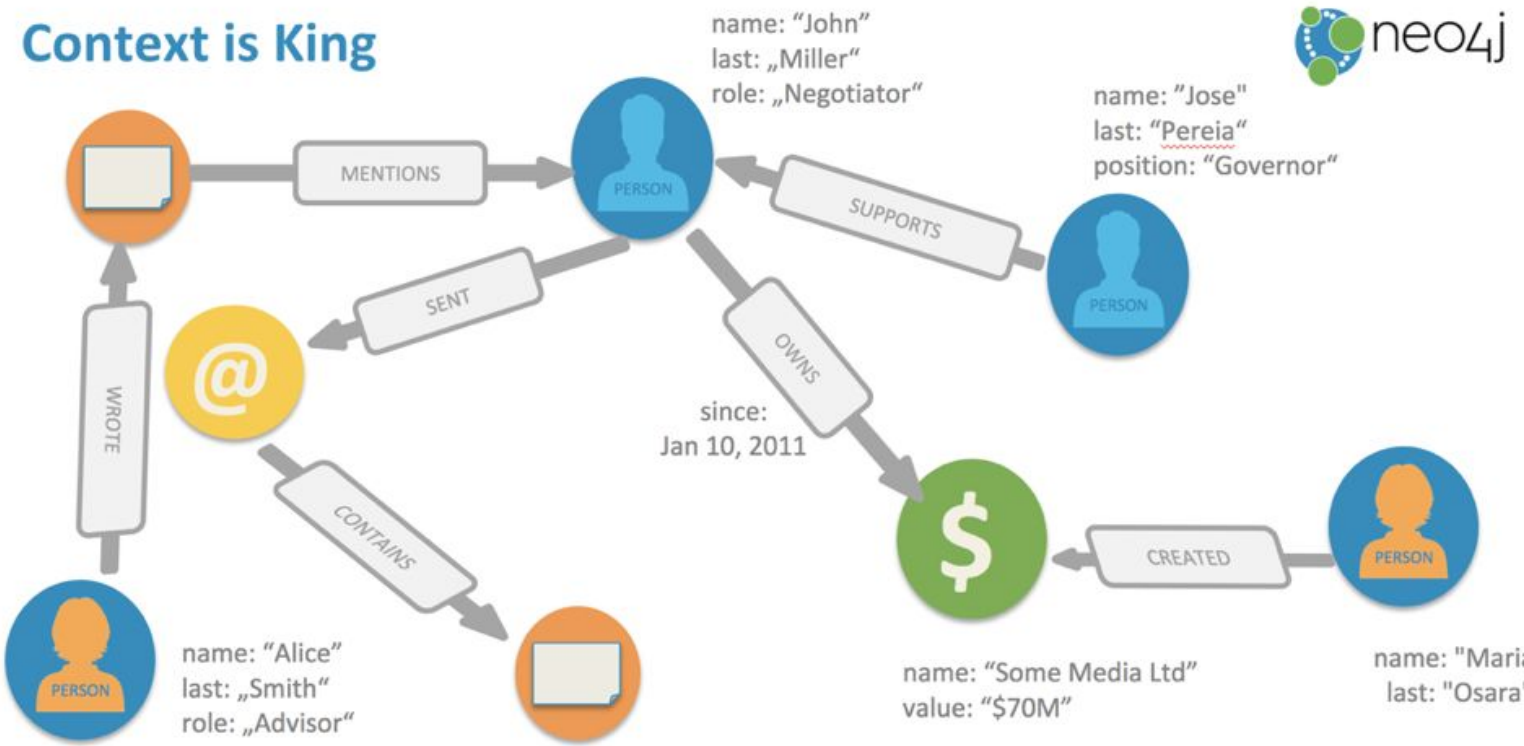
- E.g. `KNOWS`, `LIKES`

A node or relationship may have 0 or more properties

- E.g. `name: 'John'`



The topology is as important as the data



Use cases


Impact Analysis
Logistics and Routing
Recommendations
Access Control
Fraud Analysis
Social Network

Some well-known use cases

NASA

Knowledge repository for previous missions - root cause analysis

Panama Papers

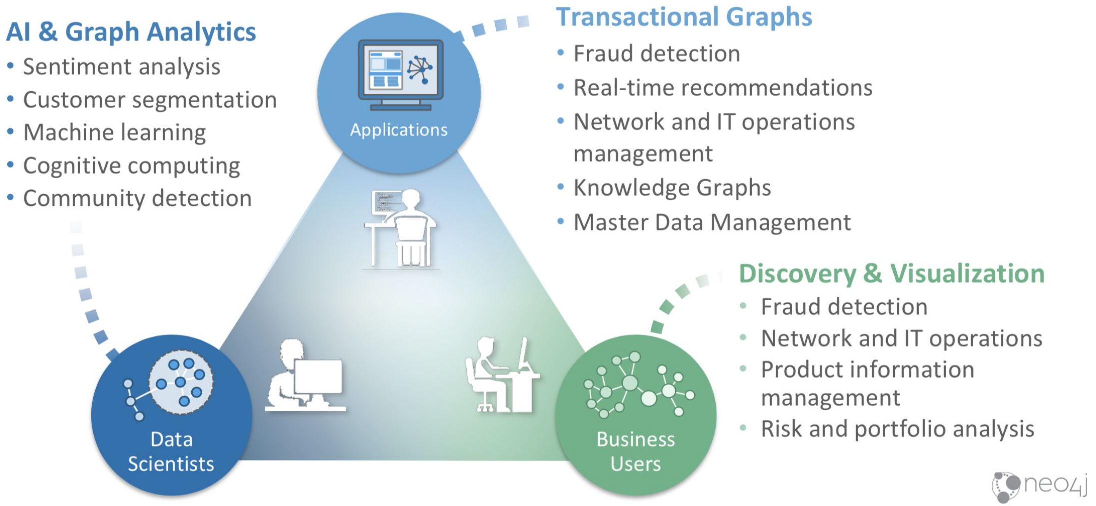
How was money flowing through companies and individuals?

Introduction to Neo4j

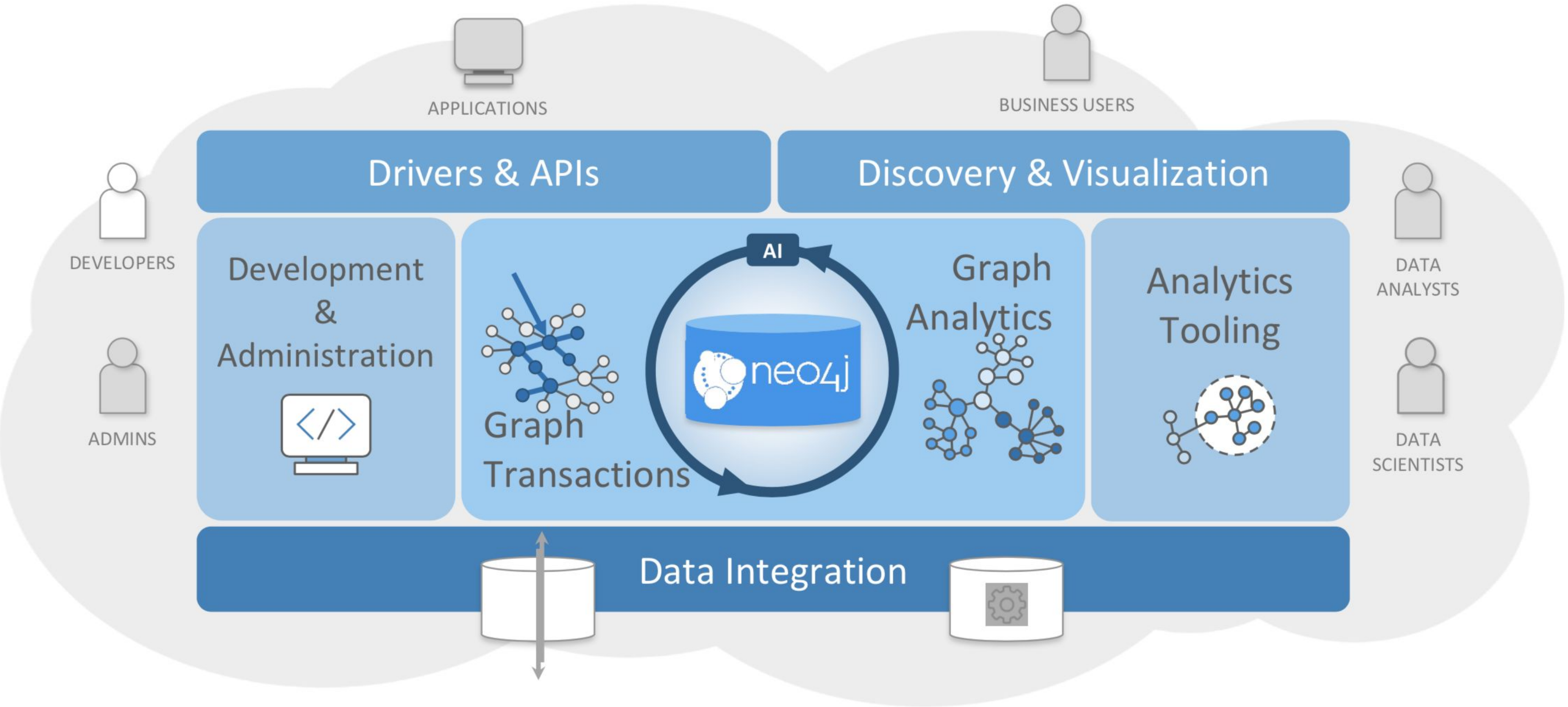
Consumers of connected data

Neo4j is an *enterprise-grade native graph platform* enabling you to:

- **Store, reveal and query** connections within your data
- **Traverse and analyze** any level of depth in real time
- **Add context and connect** new data on the fly



Neo4j graph platform



Knowledge Graphs

1. Background…

Relationship-driven applications

Customer Engagement

Real-Time Recommendations

Fraud Detection

Network Management

Dynamic Pricing

Artificial Intelligence & IoT-applications

Identity and Access Management

Supply Chain Efficiency

The Knowledge Graph problem

Organizations have difficulty maintaining their corporate memory owing to a variety of reasons:

- Growth which drives need for new and continuous education
- Turnover where long-term knowledge is lost
- Ageing infrastructures and siloed information

Negative consequences

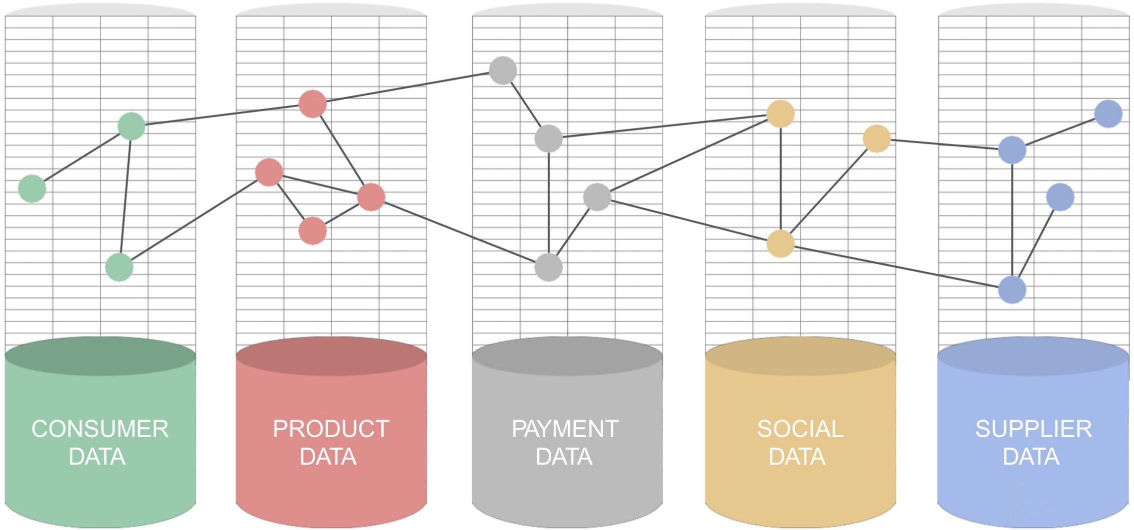
Lack of knowledge sharing slows project progress, and creates inconsistencies even among team members.

Organizations **don't know what they don't know**, nor do they know what they know.

Data scientists, and therefore the organization, are slow to recognize or react to changing market conditions, so they miss opportunities to innovate

Bad information is spread inadvertently which erodes corporate trust

Knowledge Graphs in the Age of Connections



The next wave of competitive advantage will be all about using **connections** to **identify** and **build knowledge**

2. What is a Knowledge Graph?

neo4j

Enriching graphs with more and more data (raw or derived) over time...

Enriching graphs with more and more data (raw or derived) over time...

...resulting in a graph that has more detail, context, truth, intelligence and semantics...

Enriching graphs with more and more data (raw or derived) over time...

...resulting in a graph that has more detail, context, truth, intelligence and semantics...

...capturing the real world (your ecosystem) more precisely and more comprehensively...

Enriching graphs with more and more data (raw or derived) over time...

...resulting in a graph that has more detail, context, truth, intelligence and semantics...

...capturing the real world (your ecosystem) more precisely and more comprehensively...

...so that the information captured in the graph can be searched for in a meaningful way...

Enriching graphs with more and more data (raw or derived) over time...

...resulting in a graph that has more detail, context, truth, intelligence and semantics...

...capturing the real world (your ecosystem) more precisely and more comprehensively...

...so that the information captured in the graph can be searched for in a meaningful way...

...yielding **KNOWLEDGE**, both directly and indirectly (new insights are discovered)

Knowledge Graphs provide:

A 360 degree-view of:

- any entity of interest,
- auxiliary entities,
- and the processes that surround these

Example:

- Customers
- Products, orders, reviews, delivery, ...
- Purchasing, fulfilment, order management, shipping, ...

Knowledge Graphs can:

Give answers

→ about things it knows about

Explain why and how the answer was returned

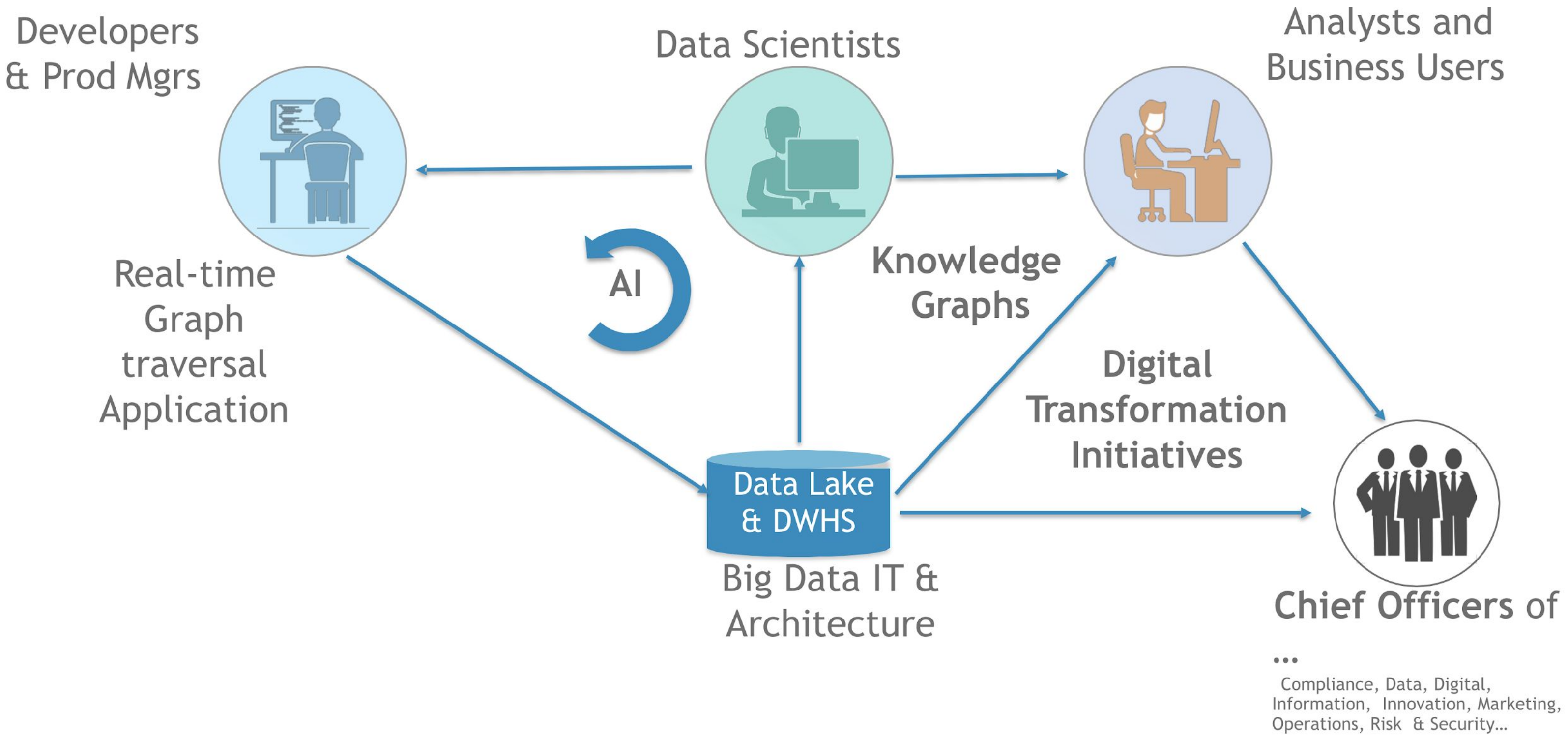
→ from the context and structure within the graph

Knowledge Graphs must be <u>searchable</u>*

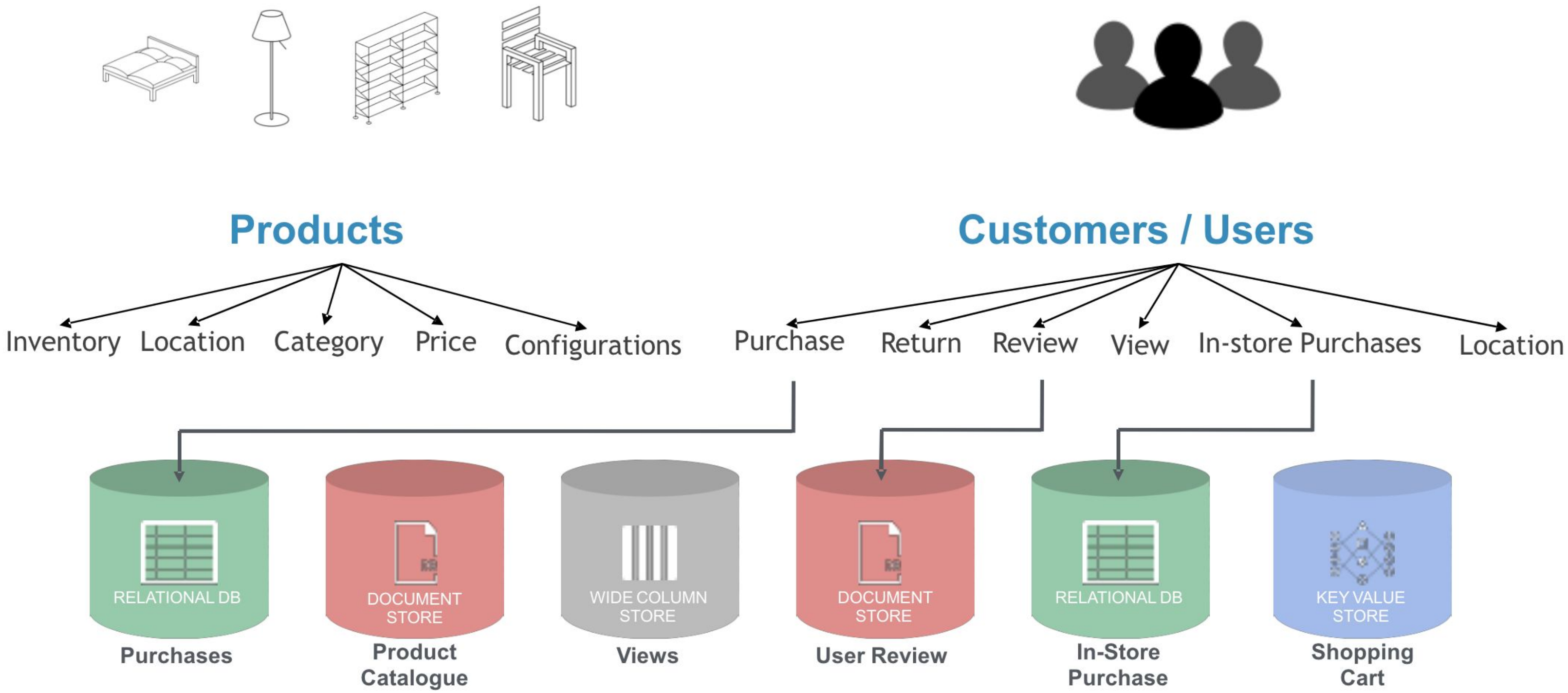
*more on this later

3. How do they work in Neo4j?

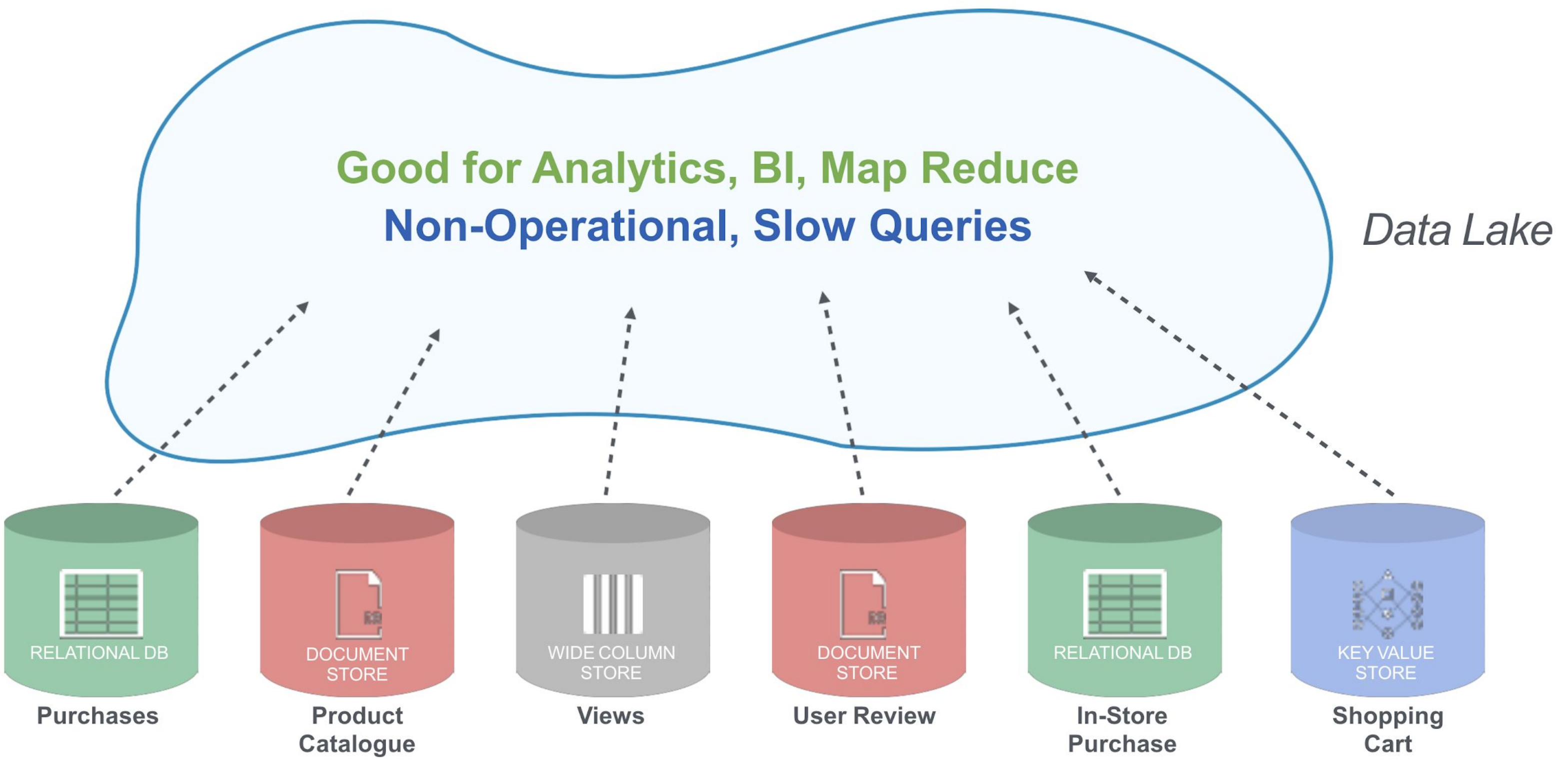
Connecting roles in the enterprise

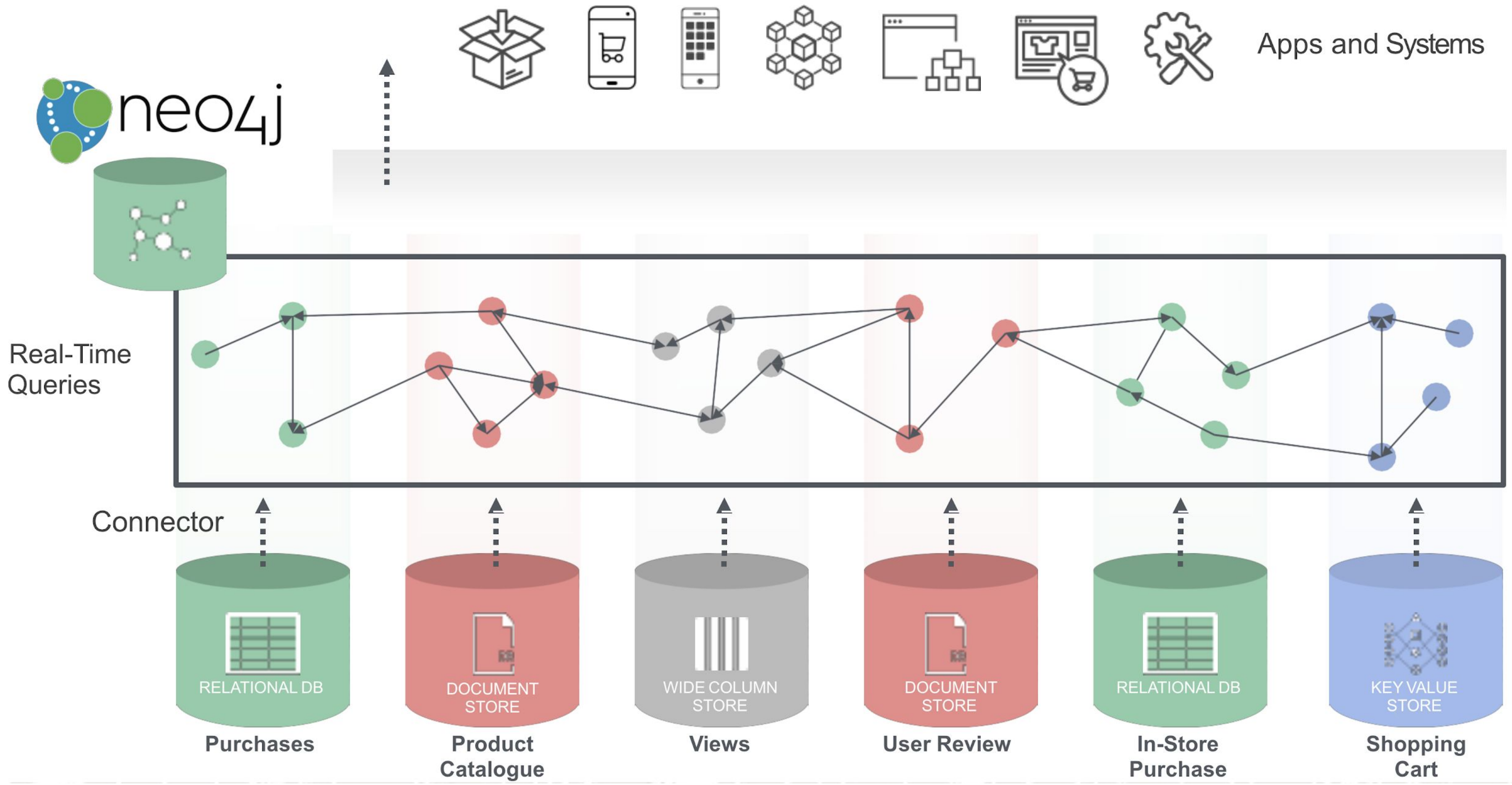


Data lives across the enterprise

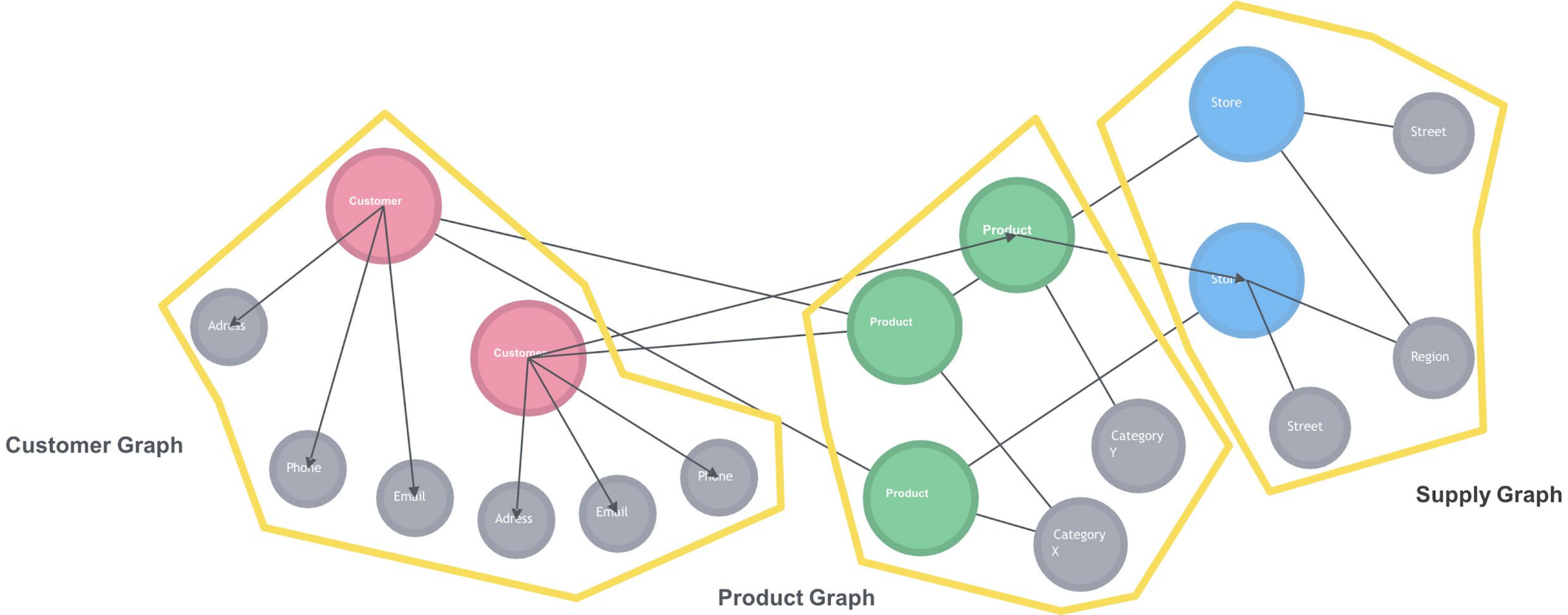


Recommendations require an operational workload

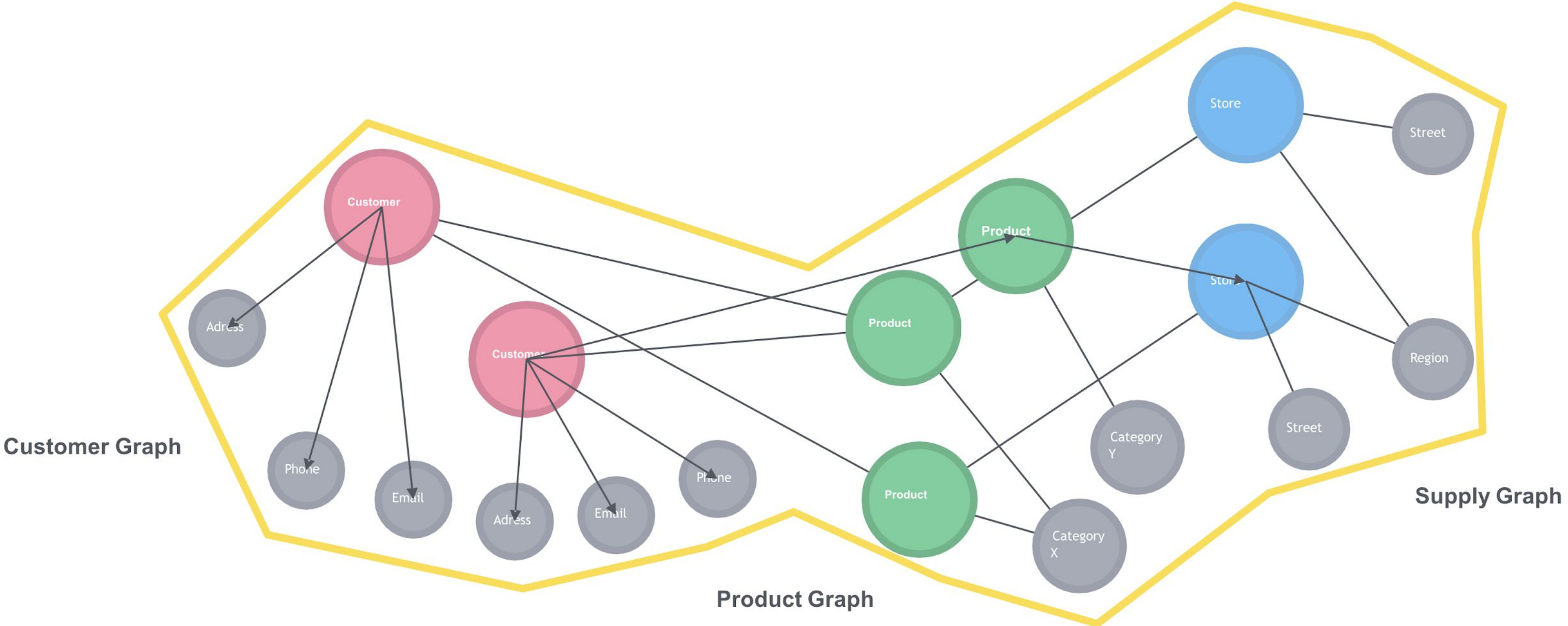



Apps and Systems
neo4j
Real-Time Queries
Connector
RELATIONAL DB
Purchases
DOCUMENT STORE
Product Catalogue
WIDE COLUMN STORE
Views
DOCUMENT STORE
User Review
RELATIONAL DB
In-Store Purchase
KEY VALUE STORE
Shopping Cart

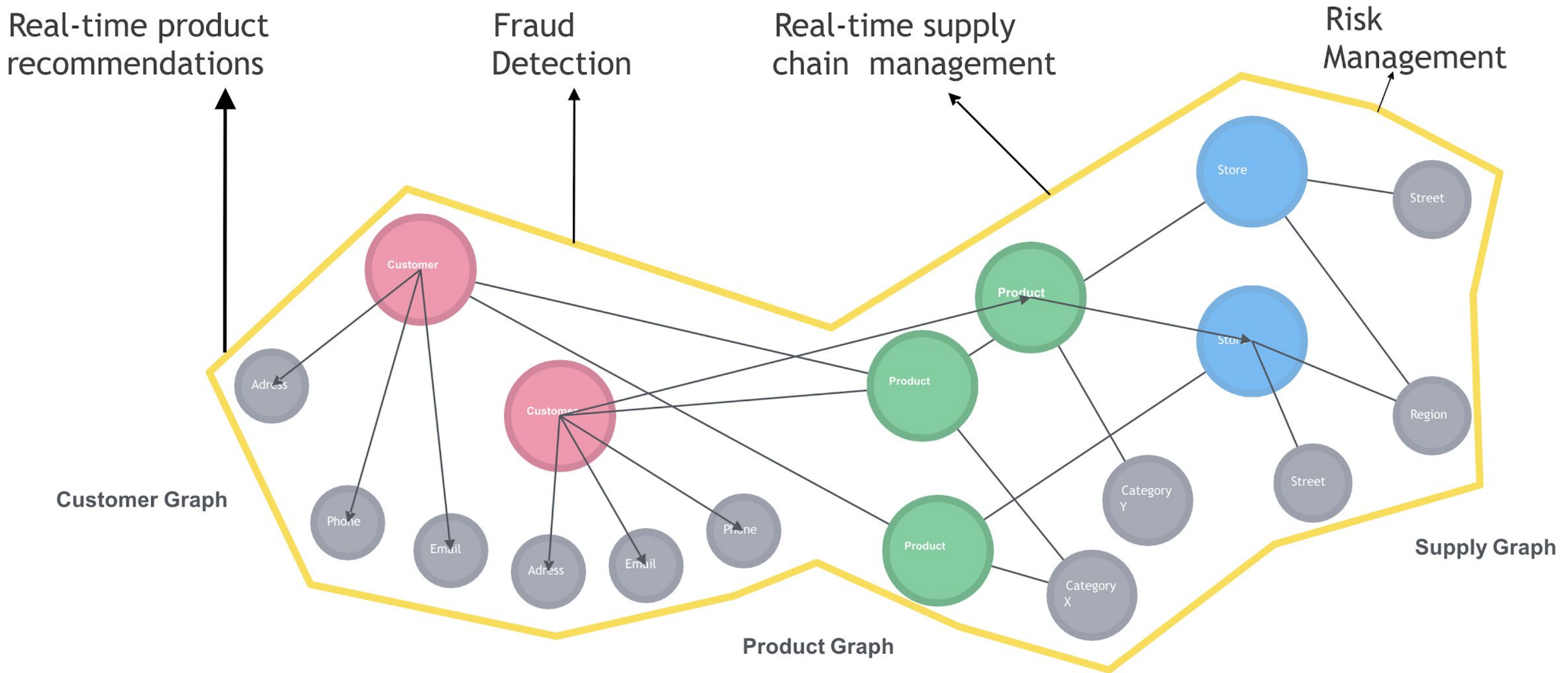
Simple enterprise Knowledge Graphs



Simple enterprise Knowledge Graph



Unlock the institutional memory



The goals

Information, especially in analytics, research departments and customer service should have a searchable, consistent repository, or representation of a repository, from which to store and draw institutional knowledge.

Corporations who maintain a knowledge graph will develop higher degrees of consistency across all areas of business.

What’s required to get there

Institutional memory requires a solution that can **integrate** diverse data sets, often in text due to the legacy nature of that information and return “context” as a result

Connections and **relationships**, cause and effect correlation needs to be materialized and persisted permanently

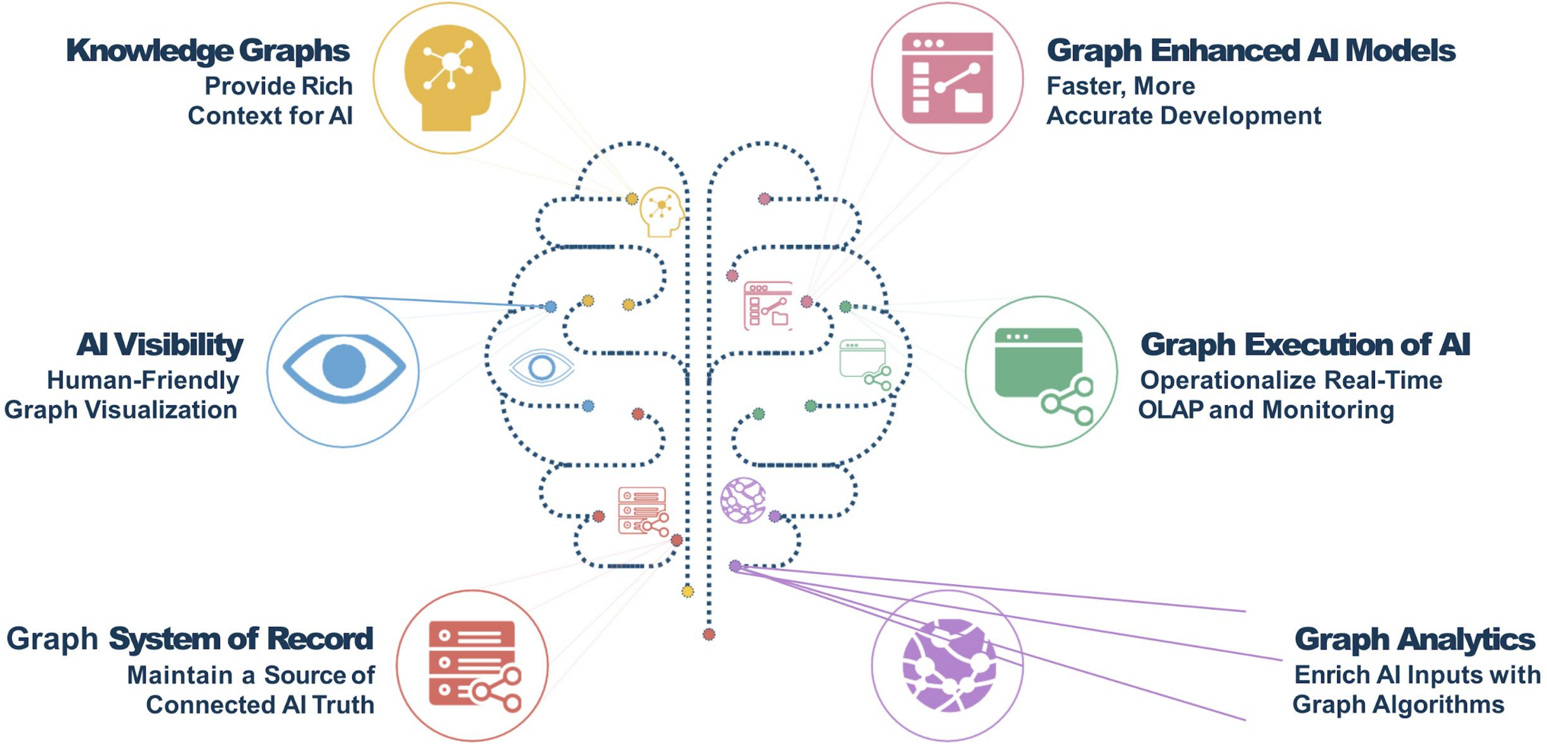
All information must be indexed, **searchable** and shareable

The solution must be agile, **easily expandable and adaptable** to changing business conditions

The solution needs to be a combination of text-based NLP, ElasticSearch and Graphs.

Information must be easy to visualize and leverage in your processes and workflows

Graph-boosted Artificial Intelligence



4. Tools & techniques: a whistle-stop tour

neo4j

Tools & techniques: enriching the data

NLP (Natural Language Processing): understanding human language (for the purposes of information extraction)

Parsing unstructured data (usually text) to extract entities, their attributes or properties and the relationships between entities

Named entity recognition: identify entities and categorise them; e.g. “Daniel Craig” is a Person and an Actor

Tools & techniques: enriching the data

NLP (Natural Language Processing): [continued...]

Entity disambiguation: how to determine which entity is being referred to; e.g. “Amazon” can be the rainforest in South America, or the online store

Sentiment analysis: identifying the mood/attitude/emotion of a statement

Tools & techniques: enriching the data

Provenance and lineage of data:

- How/from where the data was derived or obtained
- This adds to data veracity and quality

Tools & techniques: enriching the data

Inferencing / deriving new facts from machine learning

APOC: ~450 procedures and user-defined functions helping with data integration, graph algorithms, data conversions etc
https://github.com/neo4j-contrib/neo4j-apoc-procedures

Graph algorithms library (highly-parallelized & efficient): centralities (PageRank, betweenness…), community detection, path finding etc
https://github.com/neo4j-contrib/neo4j-graph-algorithms

In constant development; links with academic institutions for cutting-edge optimization techniques

Tools & techniques: enriching the data

Inferencing / deriving new facts: deductive method (from a knowledge base, rules engine or ontology)

Rules:

If a node **X** has
(i) a `:Person` label and
(ii) there is an outgoing `TEACHES` relationship,
then add a `:Teacher` label to **X**

Tools & techniques: enriching the data

Inferencing / deriving new facts: deductive method [continued..]

Use external data sources to obtain new facts; e.g. hook into a thesaurus to deduce/derive synonyms (ConceptNet, WordNet, ..)

Use the graph to deduce a set of rules (or begin with a seed set, using ML) which enrich the graph, and subsequently enrich the rules as more data gets added to the graph: feedback loop

Graph search through Cypher

Introducing Cypher

A declarative **graph pattern matching** language for property graphs

This is at the heart of Cypher

SQL-like syntax

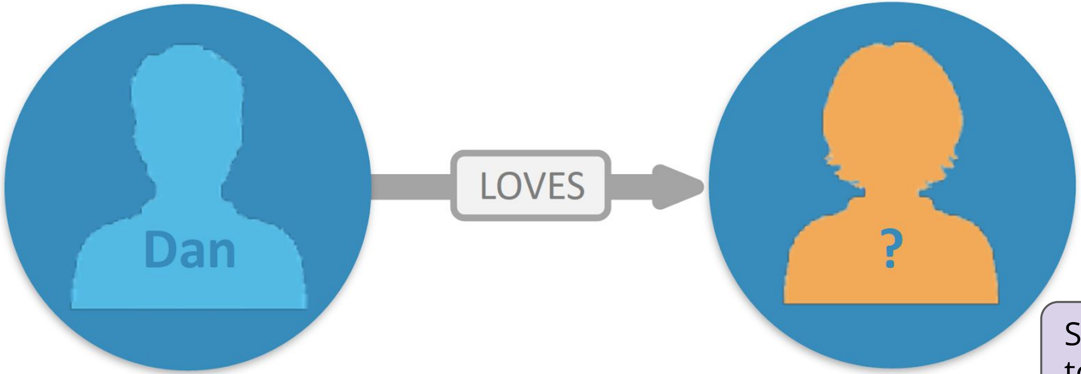
- DQL for reading data (focus of this section)
- DML for creating, updating and deleting data
- DDL for creating constraints and indexes

Relationship-centric querying

- Recursive querying
- Variable-length relationship chains
- Returning *paths*

http://www.opencypher.org/cips

Searching for (matching) graph patterns



Specifying the pattern to match

Specifying the data to be returned

NODE | Relationship | NODE

```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```

LABEL | PROPERTY | VARIABLE

DQL: reading data

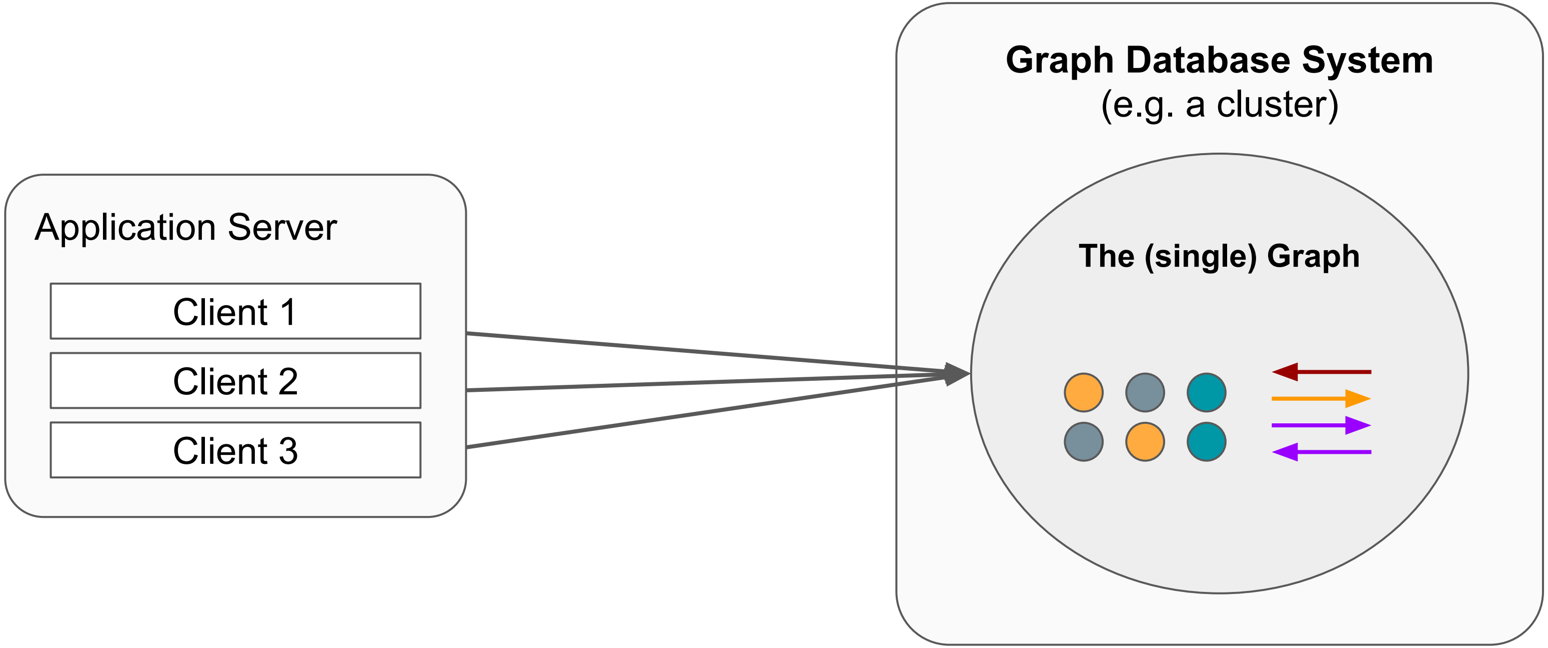
Input: a property graph
Output: a table

```
// Pattern description (ASCII art)
MATCH (me:Person)-[:FRIEND]->(friend)
// Filtering with predicates
WHERE me.name = 'Frank Black' AND friend.age > me.age
// Projection of expressions
RETURN toUpper(friend.name) AS name, friend.title AS title
// Order results
ORDER BY name, title DESC

Variable-length relationship patterns
MATCH (me)-[:FRIEND*]-(foaf)       // Traverse 1 or more FRIEND relationships
MATCH (me)-[:FRIEND*2..4]-(foaf)   // Traverse 2 to 4 FRIEND relationships

// Path binding returns all paths (p)
MATCH p = (a)-[:ONE]-()-[:TWO]-()-[:THREE]-()
// Each path is a list containing the constituent node
RETURN p
```

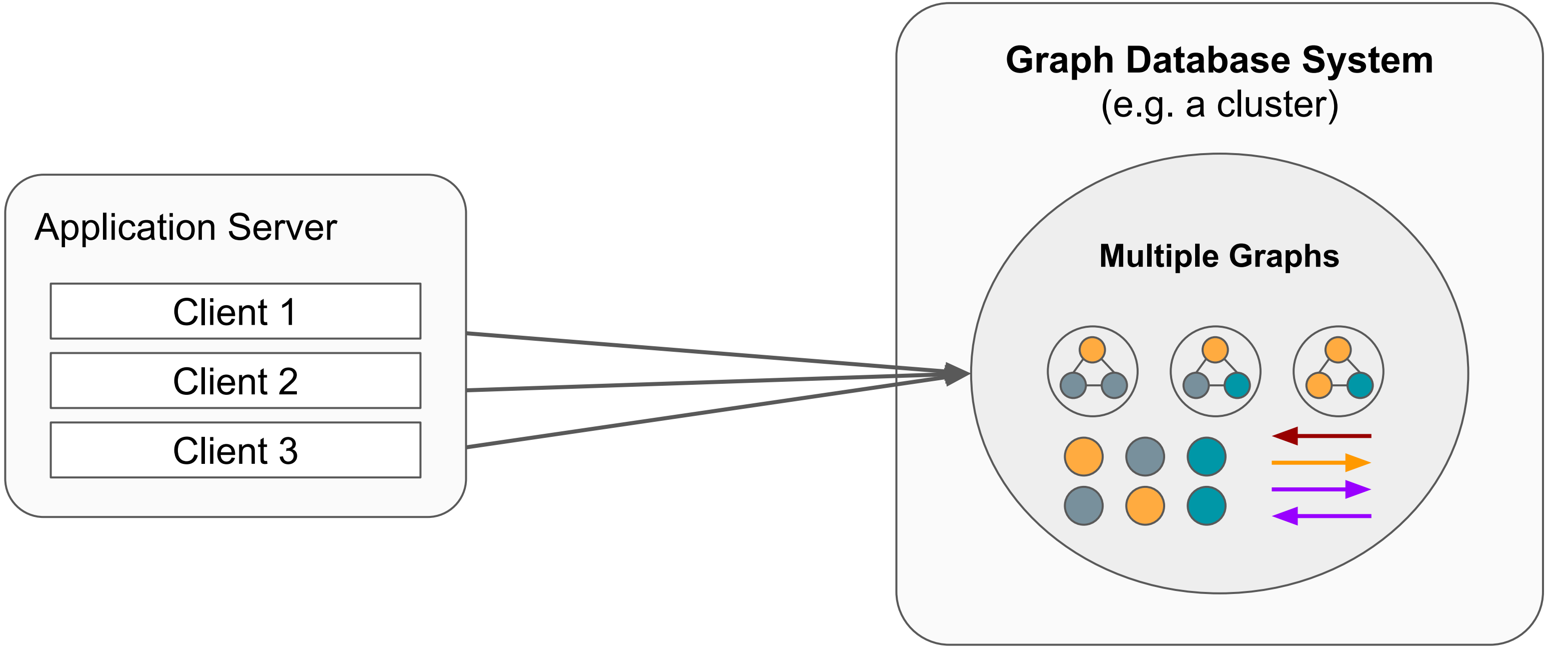
This tells you why the answer was returned (going back to the context a Knowledge Graph provides)

Extension: Multiple graphs and query composition

Cypher today: a single graph model



Cypher: multiple graphs model



Multiple graphs & query composition

Accept multiple graphs and a table as input

Return multiple graphs and a table

Chain queries to form a query pipeline

Subquery support

Use cases: create new graphs (either from scratch or from existing graphs), combining and transforming graphs from multiple sources, views, access control, snapshot graphs, roll-up and drill-down operations, ...

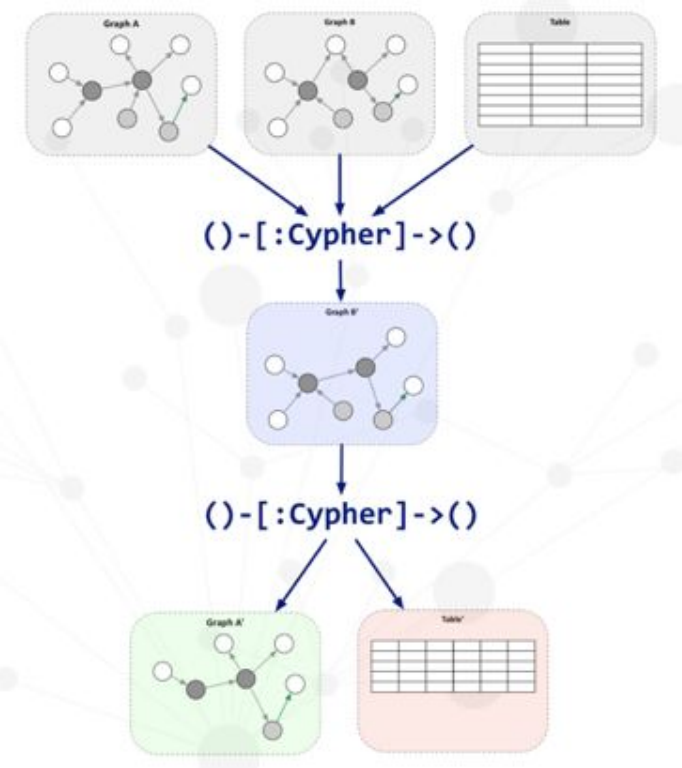
Query composition

The **output** of one query can be used as the **input** to another

Organize a query into multiple parts

- Extract parts of a query to a view for re-use
- Replace parts of a query without affecting other parts

Build complex workflows programmatically



Extension: Complex path patterns

Complex path patterns

Concatenation
a.b - a is followed by b
Alternation
a|b - either a <u>or</u> b
Transitive closure
***** - 0 or more
+ - 1 or more
{m, n} - at least m, at most n
Optionality:
? - 0 or 1
Grouping/nesting
() - allows nesting/defines scope

"Patterns of patterns"

Regular path queries (RPQs)

Long academic history

Can specify a path by means of a regular expression over the relationship types (allowing for nesting of patterns)

```
X, knows.(likes.eats|drinks)+, Y
```

Find a path whose edge labels conform to the regular expression, starting at node X and ending at node Y

Path Pattern Queries in Cypher

Find complex connections

Increase in the need to express "nested patterns"; in particular:
`(a.b)*`

Property graph data model:

- *Properties* need to be considered
- *Node labels* need to be considered

Specifying a cost for paths (ordering and comparing)

https://github.com/thobe/openCypher/blob/rpq/cip/1.accepted/CIP2017-02-06-Path-Patterns.adoc

Path Pattern: example

Illustrative syntax only!

```
PATH PATTERN
  older_friends = (a)-[:FRIEND]-(b) WHERE b.age >
 a.age


MATCH p=(me)-/~older_friends+/-(you)
WHERE me.name = $myName AND you.name = $yourName
RETURN p AS friendship
```

Path Patterns can be composed (nested)

Extension: Configurable pattern-matching semantics

Cypher today

Usefulness proven **in practice** over multiple industrial verticals

Pattern matching today uses **relationship isomorphism** (no repeated relationships)


```
MATCH (p:Person {name: Jack})-[r1:FRIEND]-()-[r2:FRIEND]-(friend_of_a_friend)
RETURN friend_of_a_friend.name AS fofName
+----------+
| fofName  |
+----------+
| “Tom”    |
+----------+
```

r1 and **r2** may not be bound to the same relationship *within the same pattern*

Rationale was to avoid **potentially** returning infinite results for varlength patterns when matching graphs containing cycles (this would have been different if we were just checking for the *existence* of a path)

All the **morphisms*

All forms are valid in different scenarios

The user can configure which semantics they wish to use at a query level

Node isomorphism:

No node occurs in a path more than once
Most restrictive

Relationship isomorphism (Cypher today):

No relationship occurs in a path more than once
Proven in practice

Homomorphism:

A path can contain the same nodes and relationships more than once
Most efficient for some regular path queries
Least restrictive

Pattern quantifiers and length restrictions

Configurable **pattern quantifiers** controlling how many matches are returned

ANY At most one match (existence checking)

EACH All matches

Illustrative syntax only!

Configurable **pattern length restrictions** limits the length and nature of matches

SHORTEST Consider only shortest path (determined by length of path)

CHEAPEST Consider only cheapest path (determined by cost function in Path Pattern)

UNRESTRICTED Consider all possible paths

***Extension:* Schema**

neo4j

Constraints

Principle of schema optionality

Extending the schema to ‘tighten up’ the data model

Some examples:

- Endpoint requirements; e.g. an `:OWNS` relationship may only end at a node labelled with either `:Vehicle` or `:Building`
- Cardinality constraints; e.g. a `:KNOWS` relationship must occur no more than 3 times between any two `:Person`-labelled nodes

Thank you!

petra.selmer@neo4j.com

neo4j