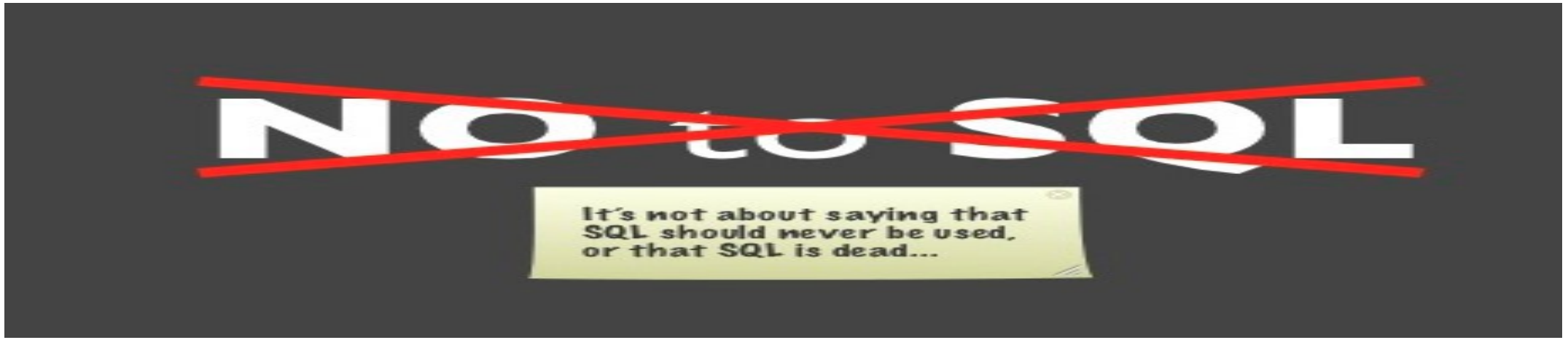
Graph database Introduction

Agenda

- What is NoSQL?
- What is a Graph, Anyway?
- What is a Graph Database?
- Neo4J Graph Database

What is NoSQL?



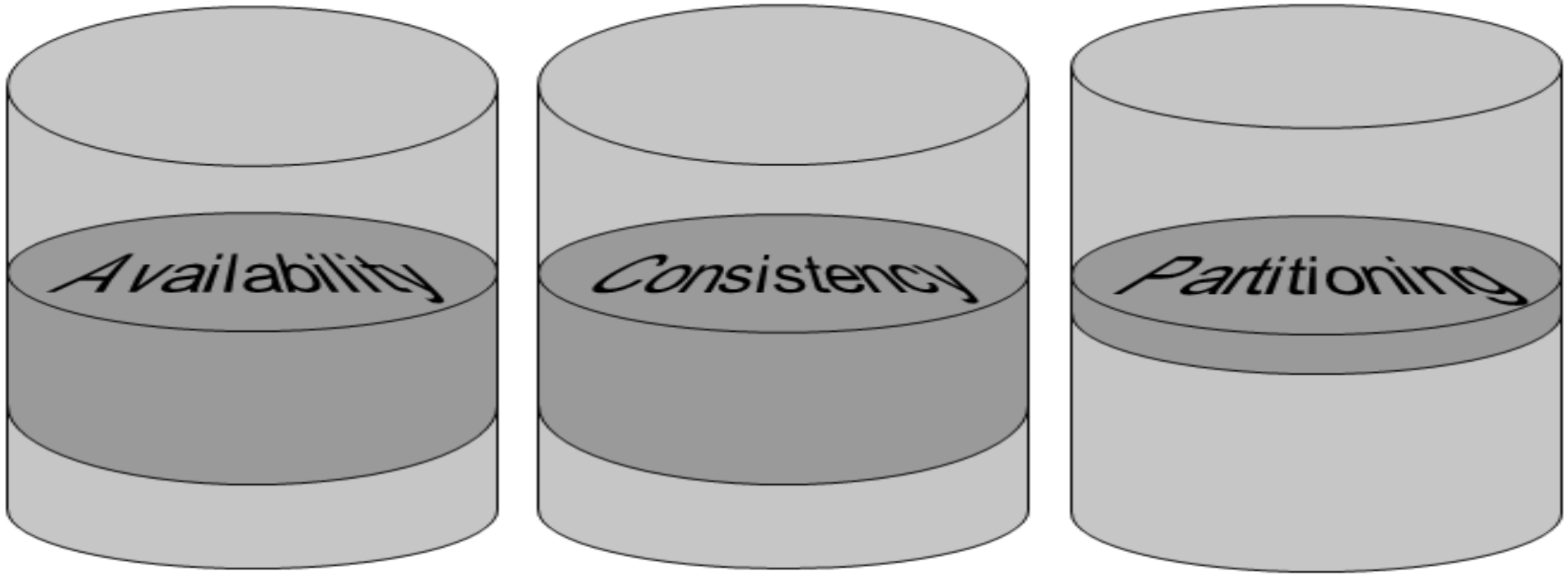
- Stands for Not Only SQL
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they user the concept of joins, group by, order by and so on.
- All NoSQL offerings relax one or more of the ACID (Atomicity, Consistency, Isolation, Durability)properties

What is NoSQL?

- Next generation databases
- Characteristics:
  - Large Data Volumes
  - Non-relational
  - Semi-structured data
  - Distributed
  - Less restrictive
  - Scalable replication and distribution

ACID vs BASE

- ACID = **A**vailability, **C**onsistency, **I**solation, **D**urability
- BASE = **B**asically **A**vailable **S**oft-state services with **E**ventual-consistency



Popular NoSQL Databases

Who is using them?

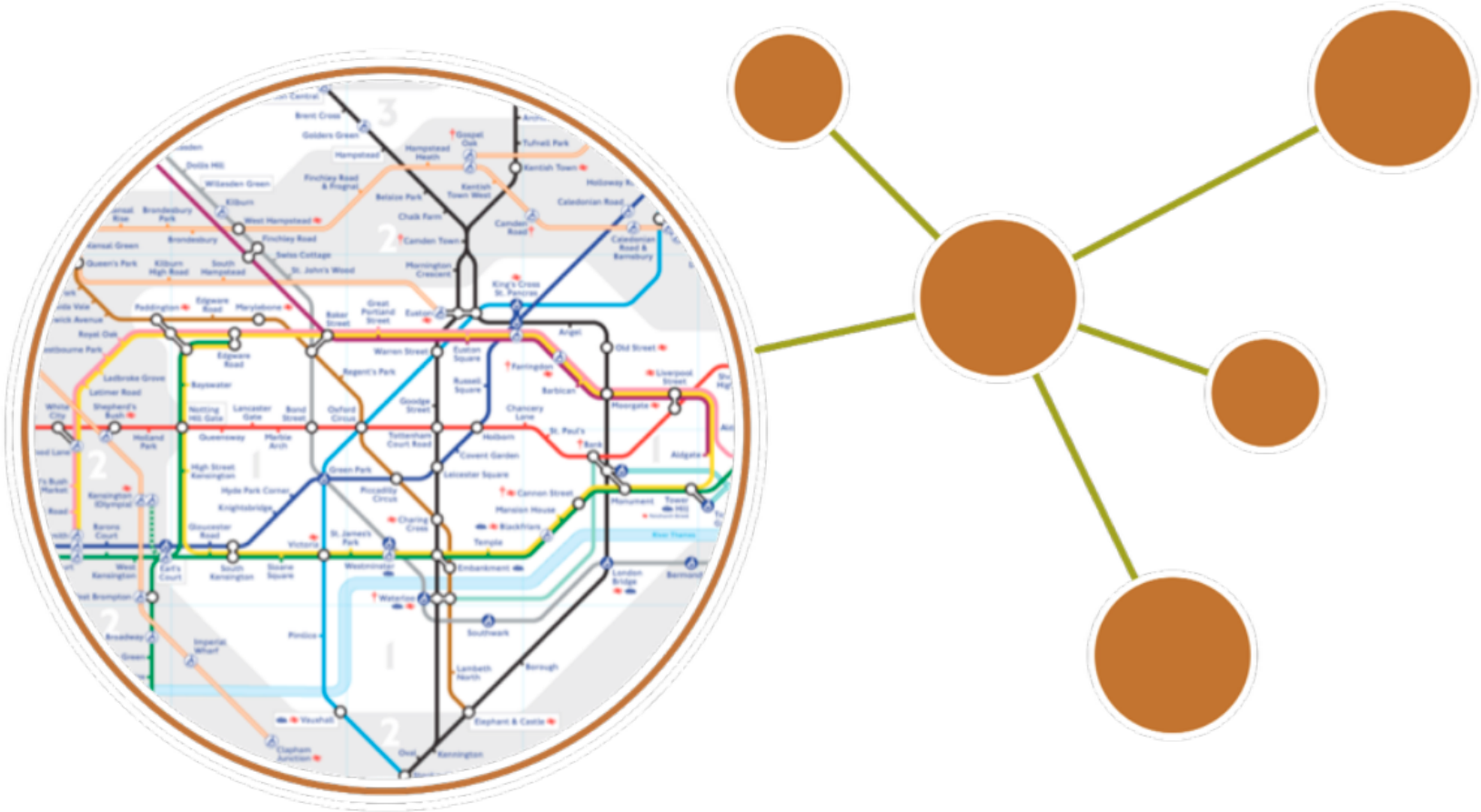
Why Graphs?

- Ideal data structures to represent relationships
- Provide visual representation of inter-connections
- White-board friendly
- Semi structured – represent a wide range of data
- Variations – (non)attributed / (un)directed / (non)weighted, etc.,.

Some Use Cases

Social Network

Route Finding

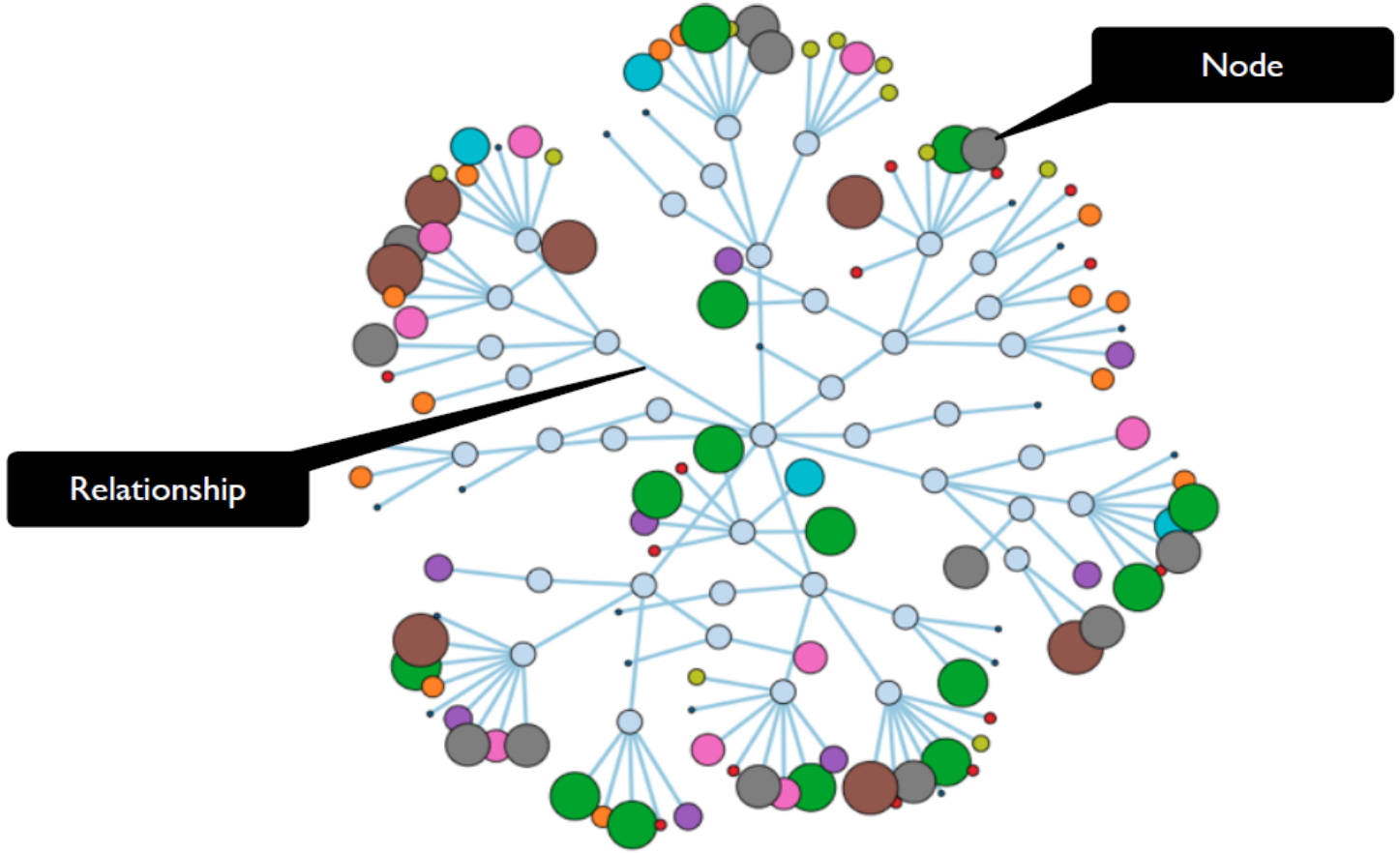


Recommendations

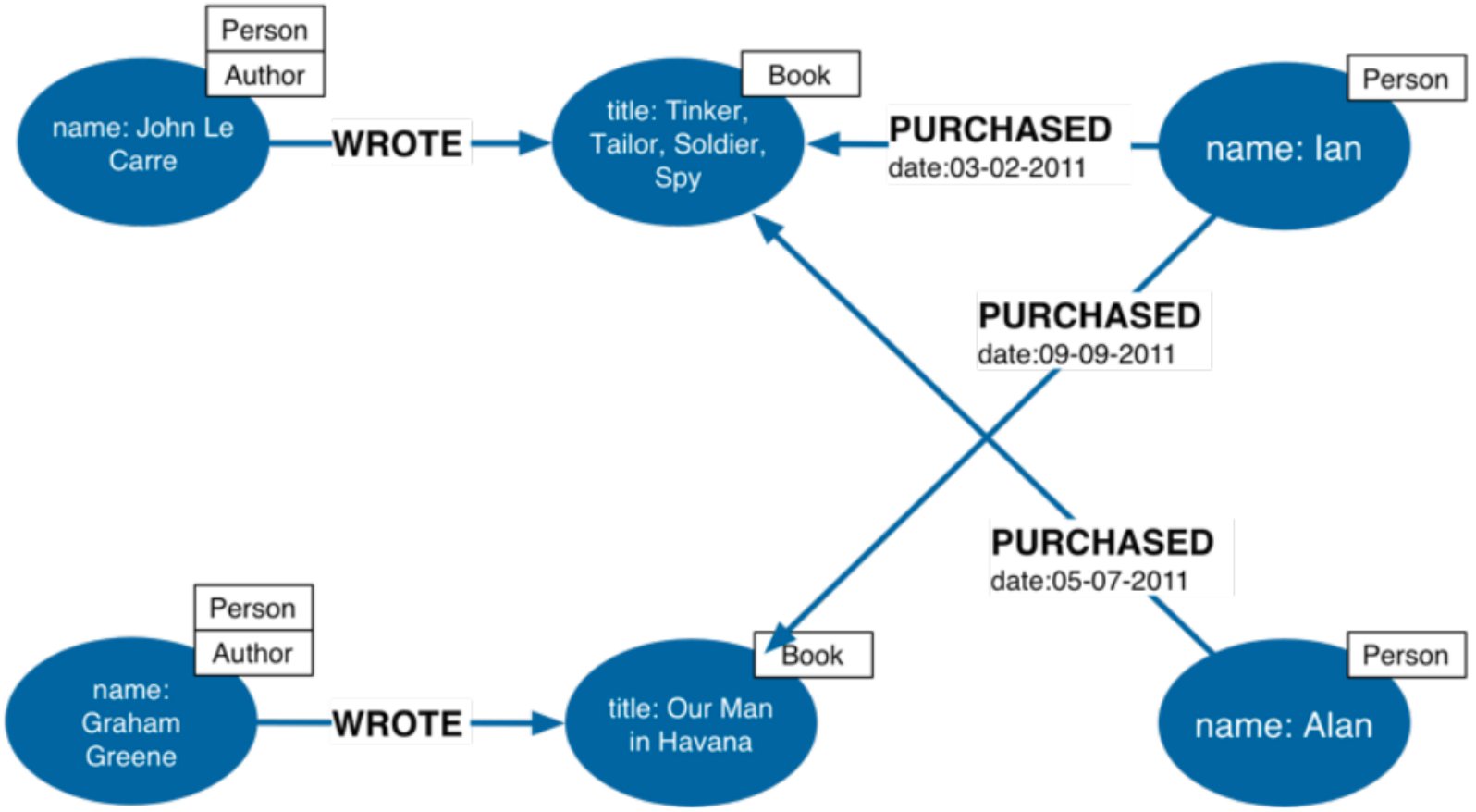
Logistics

What is a Graph Anyway?

A Graph

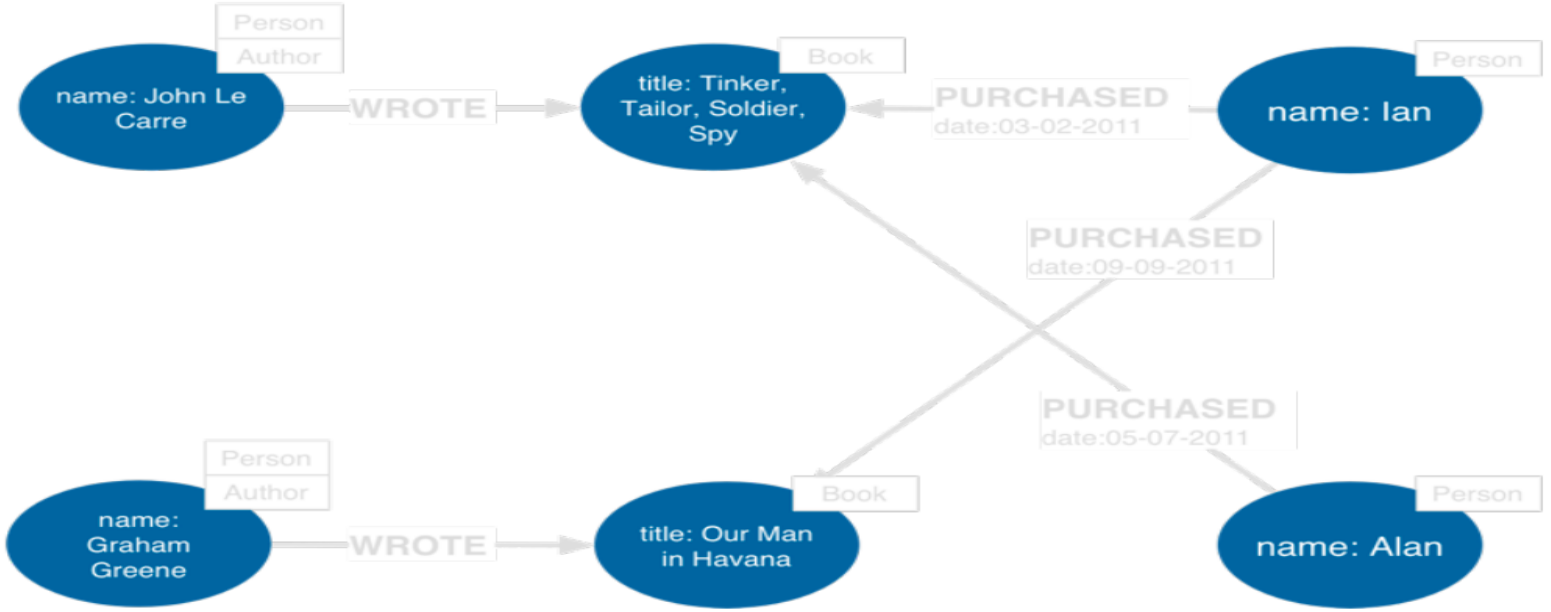


Property Graph Data Model



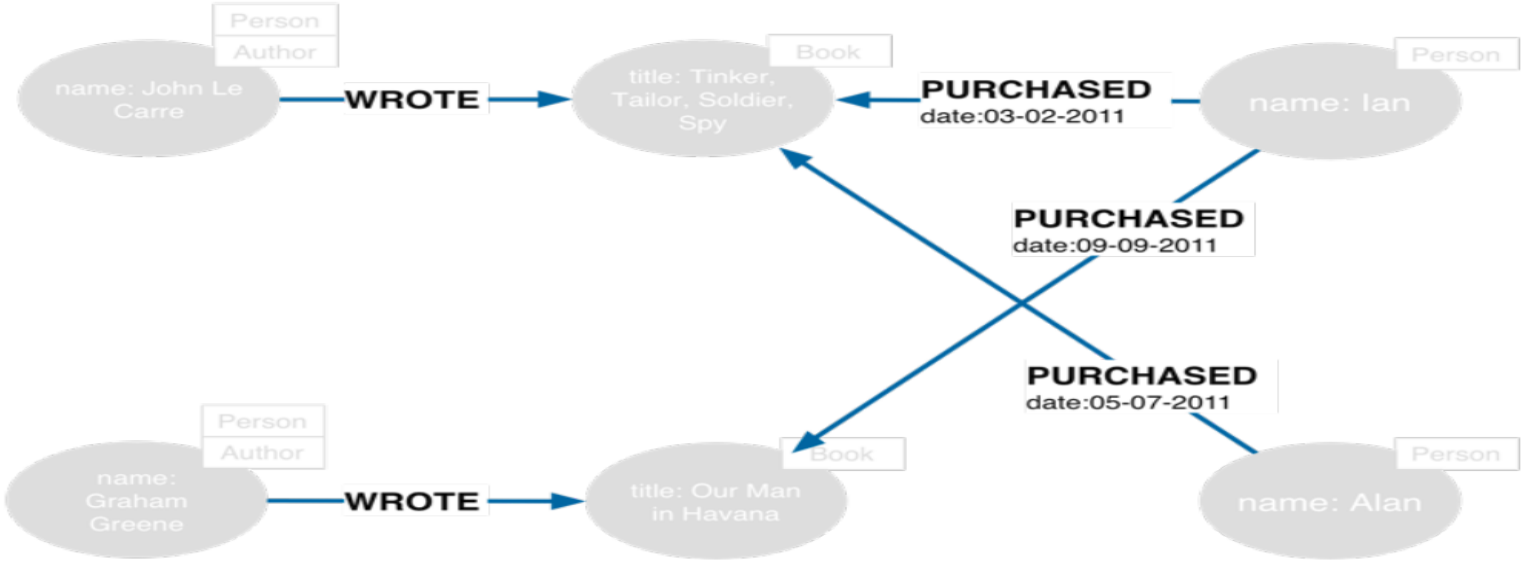
Nodes

- Used to represent entities.
- Can have properties

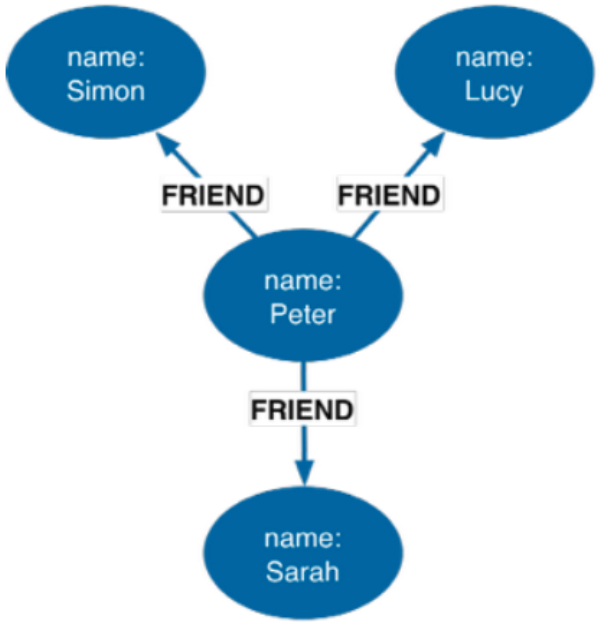


Relationships

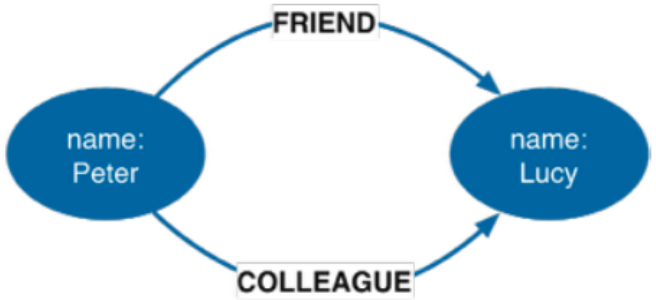
- Used to represent connections between entities.
- Can be directed or undirected
- Can have properties



Relationships (continued)



Nodes can have more than one relationship

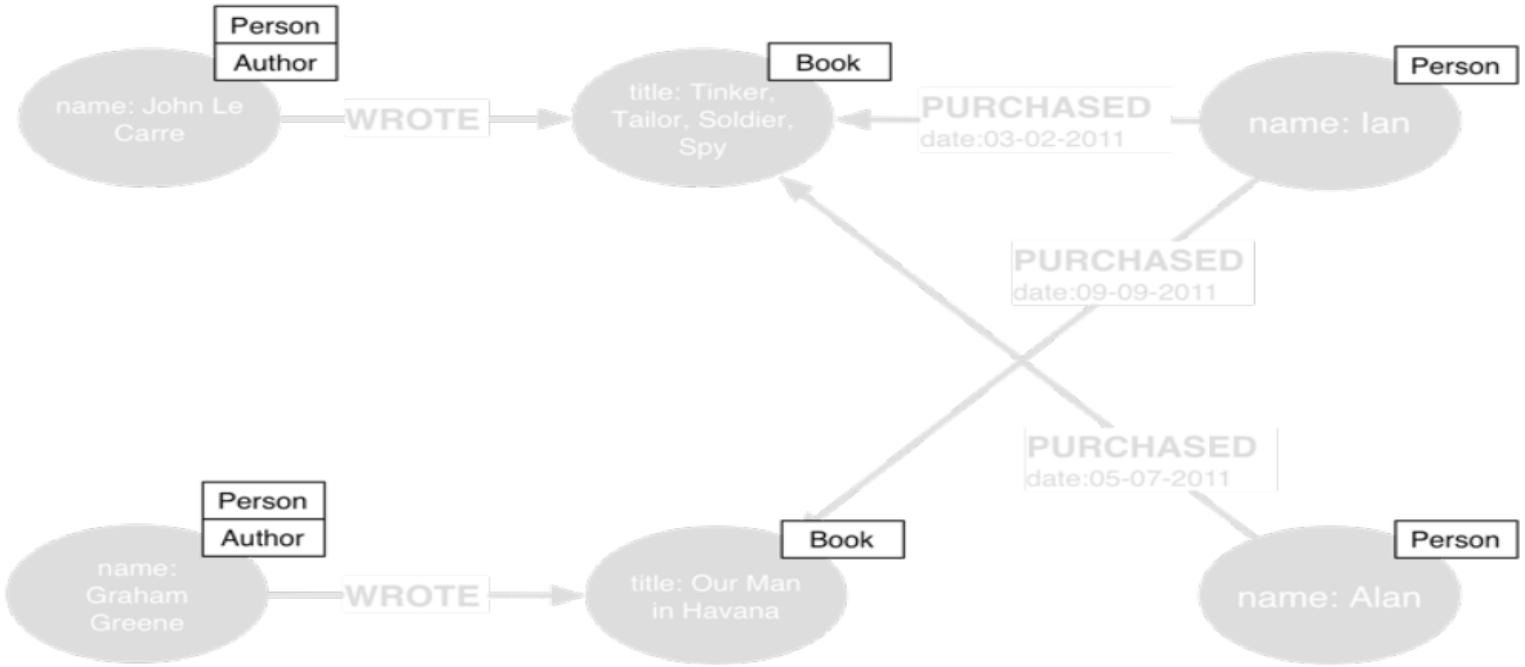


Nodes can be connected by more than one relationship

Self relationships are allowed

Labels

- Labels identify the node and edge types



Why Graph Databases?

A relational database may tell you the average age of everyone in this session, but a graph database will tell you who is most likely to buy you a beer!

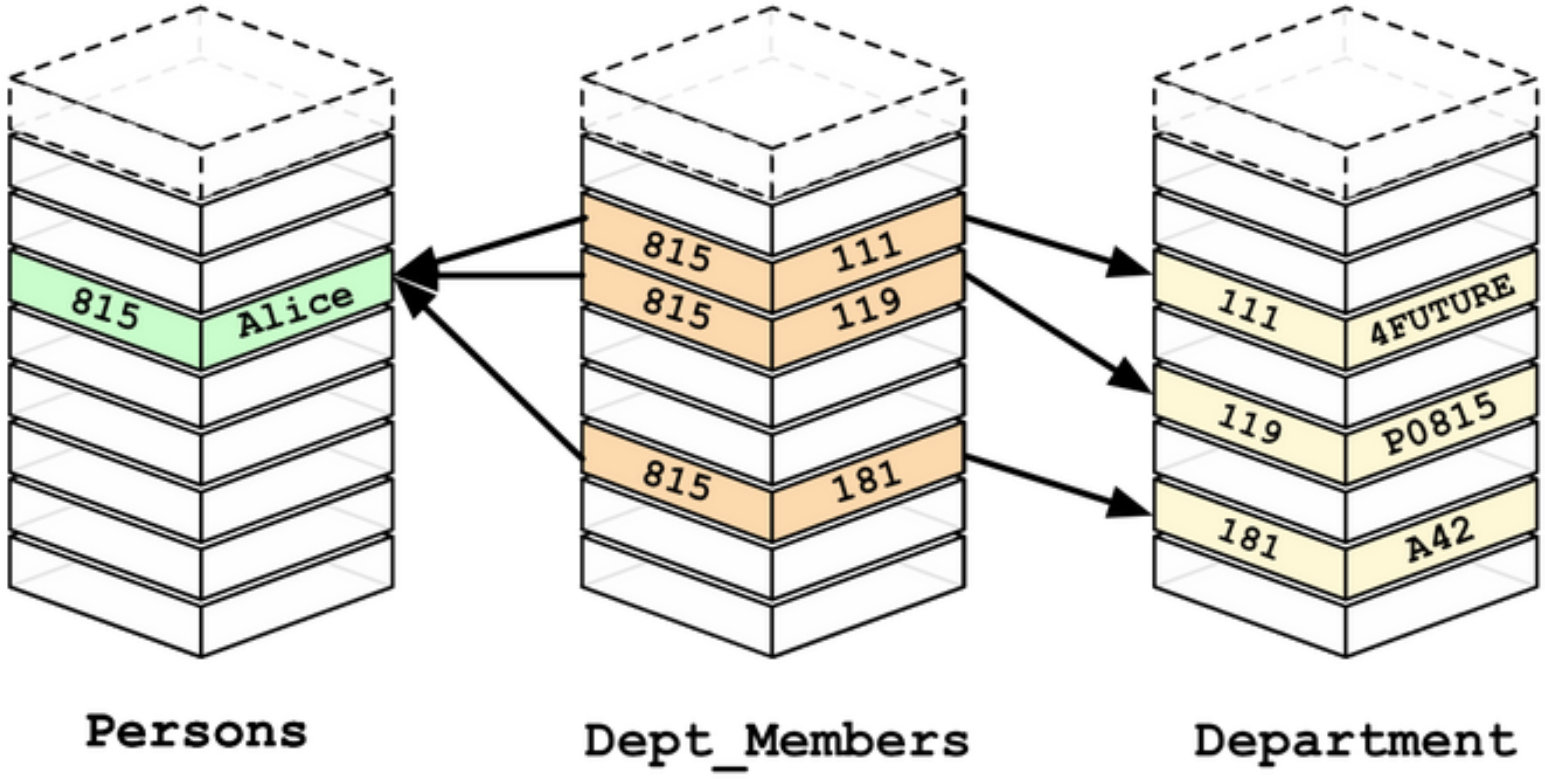
Graphs are unique

- Based on relationships
- Structure as important as the data itself
- Difficult to capture in a one-dimensional space
- Most graph queries have a tight coupling between the structure and data
- Existing storage formats incapable of optimizing storage to both structure and data

RDBMS and Relationships

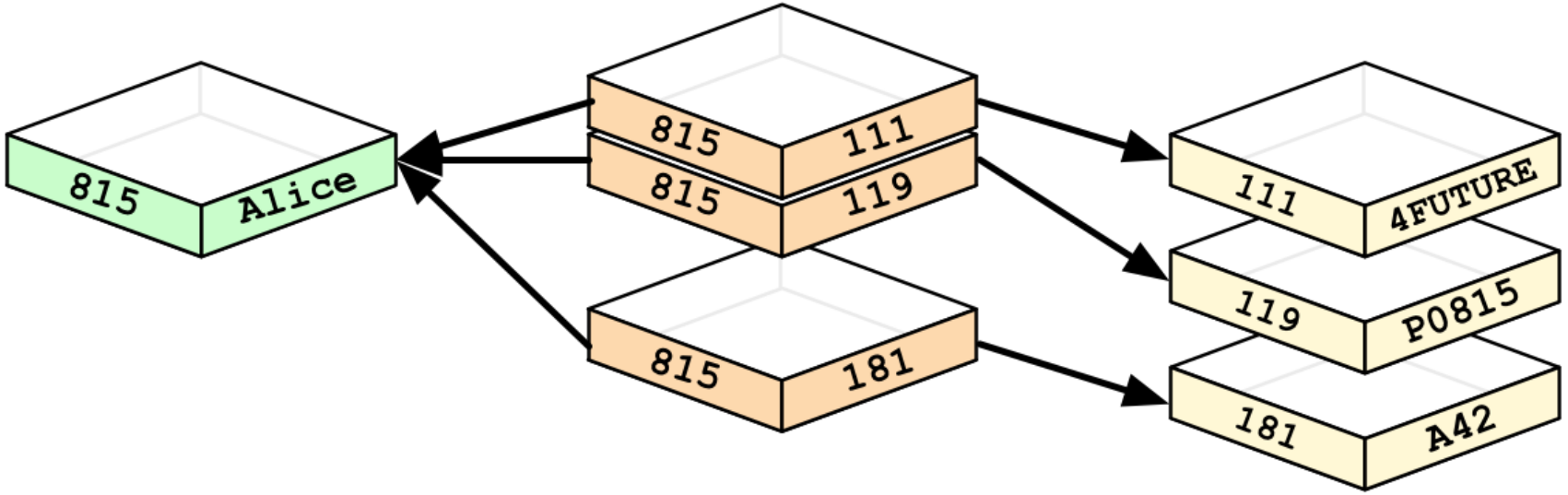
- Relational ≠ Relationships
- RDBMS not suitable graphs because relationships uncovered through joins
- No. of relationships traversed = No. of Joins
- Joins are expensive operations
- Not practical for large datasets

RDBMS and Relationships



2 Joins for 2 relationships!

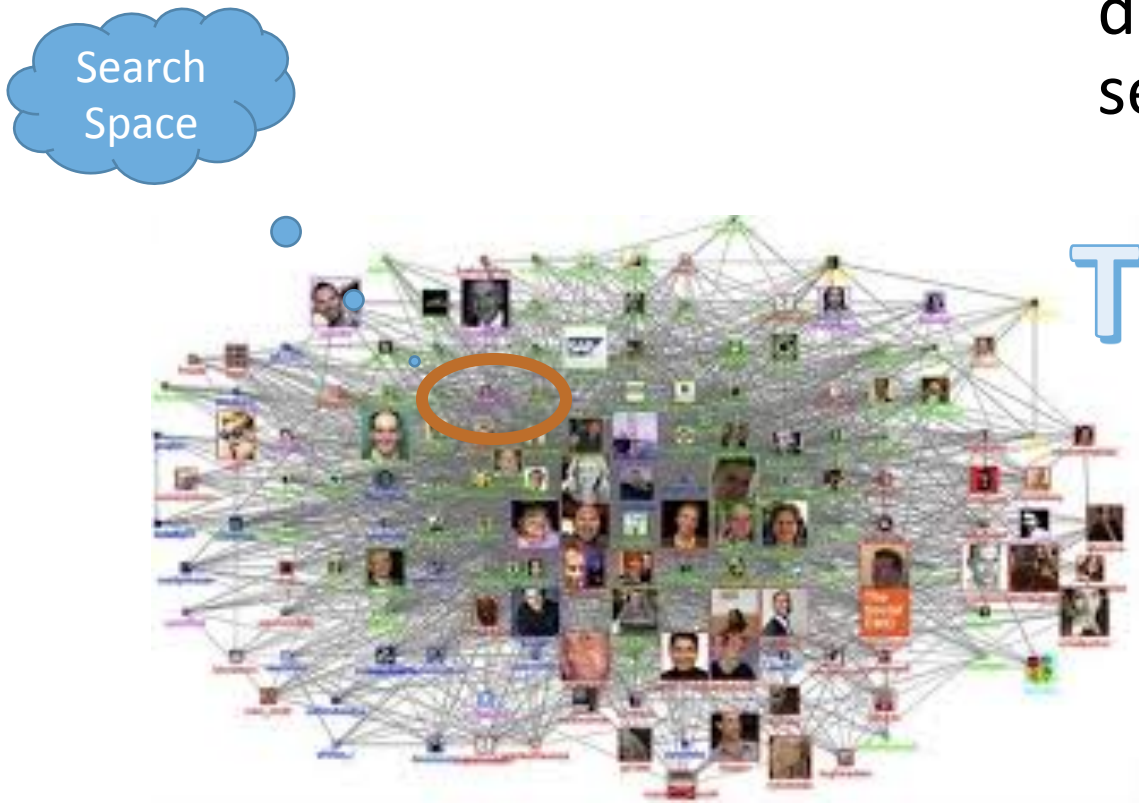
Graphs and Relationships



Relationships explored through graph traversals!

Graph Databases Advantage

- Graph Databases allow a drastic reduction in search space

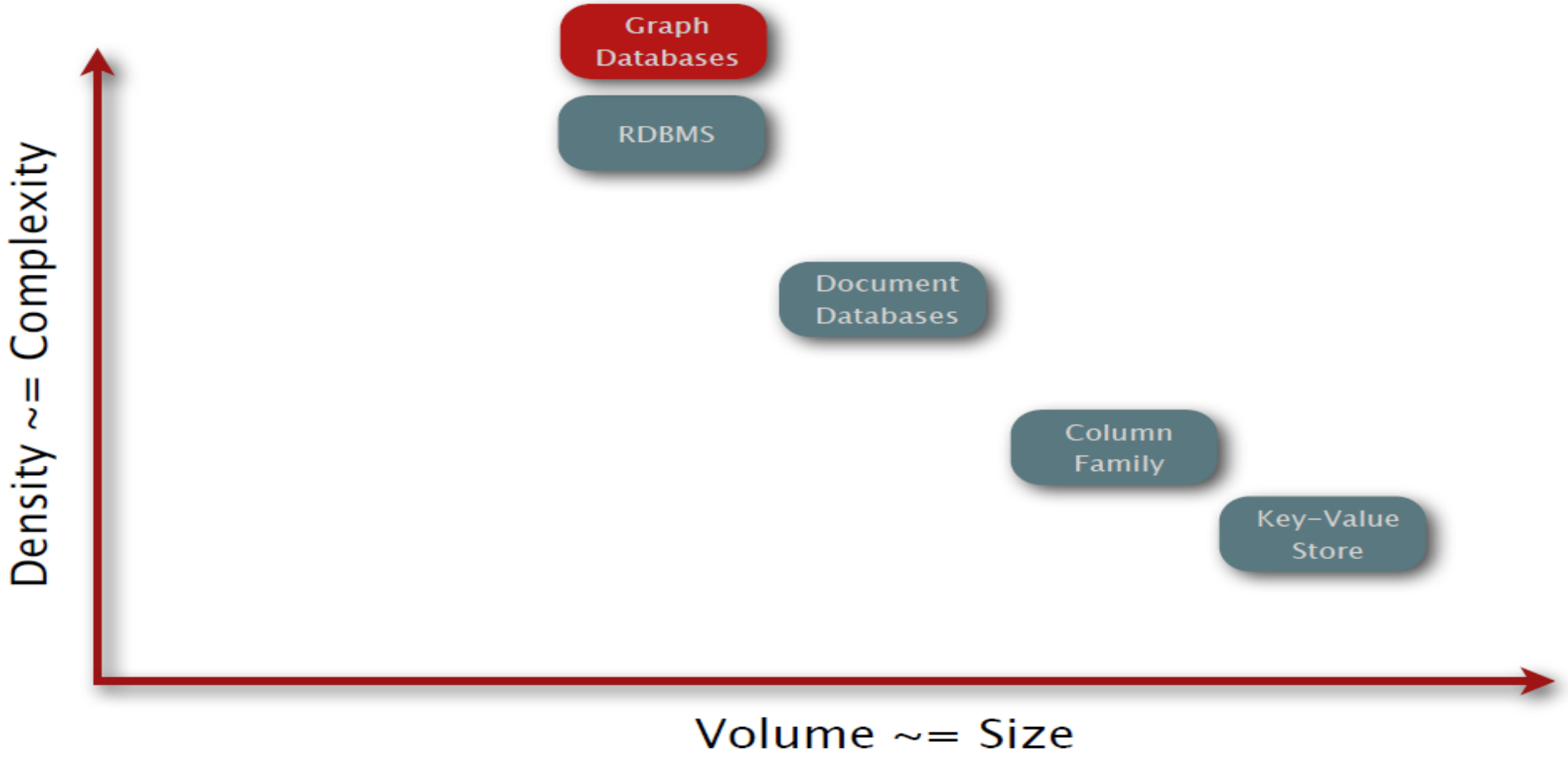


Typical Graph Size!!

Graph Databases Advantage

- No joins! – Relationships explored though graph traversals
- Optimized for storage and querying graphs
- Querying – APIs or dedicated languages. Better expressiveness

Graph Databases in the NoSQL world



Neo4J Graph Database

Neo4J - Introduction

- Robust and high performance native graph database
- Optimized for connections between records
- True ACID transactions
- High availability
- Scales to billions of nodes and relationships
- High speed querying through traversals

Neo4J - Introduction

- Uses the property graph model
- Graph queried through traversals
- Traversal
  - Visiting the nodes of the graph, following the edges according to specific rules
  - Depth first / Breadth first
- Paths
  - Set of nodes with connecting relationships
  - Fundamental units for graph queries and query results

Querying Method- Cypher

- Cypher
  - Declarative, SQL-like
  - Emphasizes on the “WHAT” instead of the “HOW”
  - Major constructs:
    - START
    - MATCH
    - WHERE
    - RETURN
    - CREATE
    - SET
    - DELETE
    - FOREACH

CYPHER – Code examples

- CREATE
  - Creates a graph object of a given type and given attributes
  - Eg: CREATE (**node1**:*Person*{name:”John”, age:20})
    - This statement creates a node node1 of type *Person* with attributes name as ”John” and age as 20.
    - Naming the created object (node1 in this case) is optional.
- RETURN
  - Returns a specific graph object
  - Used in conjunction with CREATE/MATCH/DELETE/SET
  - Eg: CREATE (**node1**:*Person*{name:”John”, age:20}) RETURN (**node1**)
    - This statement returns the **node1** object that was created

CYPHER – Code examples

- MATCH
  - Specifies the graph pattern to match. Forms the graph query.
  - Eg: MATCH (**node2**:*Person*{name:"Mary", age:23})
    - This statement matches the graph object with name Mary, of age 23 and belonging to the type "Person"
  - Complex path queries can also be given the MATCH clause.
  - Eg: MATCH (**node2**:*Person*{name:"Mary", age:23})-[**rel1**:*knows*]->(**node3**:*Person*{city:"NY"})
    - This statement matches all paths in the graph such that a node of type "*Person*" with name "Mary" and age 23 is connected to another node of type "*Person*" with attribute city as "NY" through a *knows* relationship.
  - Naming and returning the objects is optional and is generally used for further querying

Try out at http://console.neo4j.org/

Querying Method – as an embedded database in JAVA application

```
GraphDatabaseService graphDB = new EmbeddedGraphDatabase(“var/neo4j”); // start the database
Transaction tx = graphDb.beginTx(); // Neo4j supports transactions. include all accesses inside a transaction
try{
    // create nodes
    Node node1 = graphDb.createNode();
    Node node2 = graphDb.createNode();
    // set properties for nodes
    node1.setProperty(“name”,”john”);
    node2.setProperty(“name”,”mary”);
    // create relationships. Reltypes is an enum of the relationship types allowed. This needs to be predefined
    Relationship rel1 = node1.createRelationshipTo(node2, Reltypes.KNOWS);
    rel1.setProperty(“date”,today); // set properties for the relationship
    System.out.printl(“node 1 name =“+node1.getProperty(“name”)+” “+
            (“node 2 name =“+node2.getProperty(“name”)); // print the results
    tx.success();
}   finally{
    tx.finish(); // close the transactio
}
```

More code examples and tutorials at http://neo4j.com/docs/stable/introduction.html