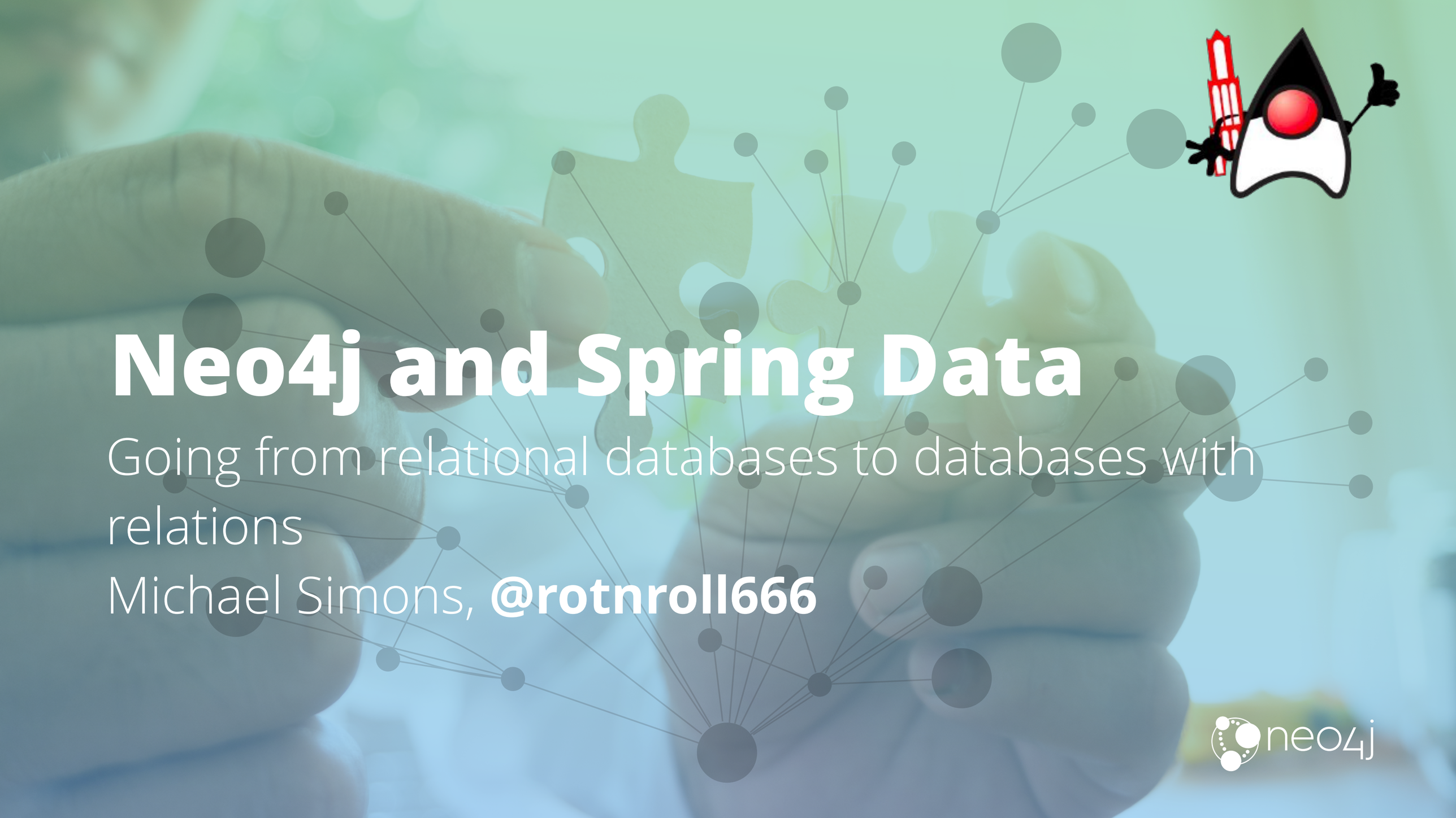


Neo4j and Spring Data

Going from relational databases to databases with relations

Michael Simons, **@rotnroll666**

neo4j

Agenda

- About Neo4j
- My „business“ domain
- Getting data into Neo4j
- Some options to access Neo4j on the JVM
- Spring Data Neo4j
- Some advanced queries

About Neo4j

Neo4j

- Neo4j is ***the #1 platform for connected data***.
- Neo4j powers the ***next generation*** of applications and analytics
- Prominent use cases are found in areas like machine learning, personalized recommendations, fraud detection, data governance and more.

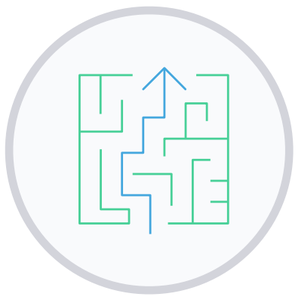
Neo4j

Mindset

"Graph Thinking" is all about considering connections in data as important as the data itself.

Native Graph Platform

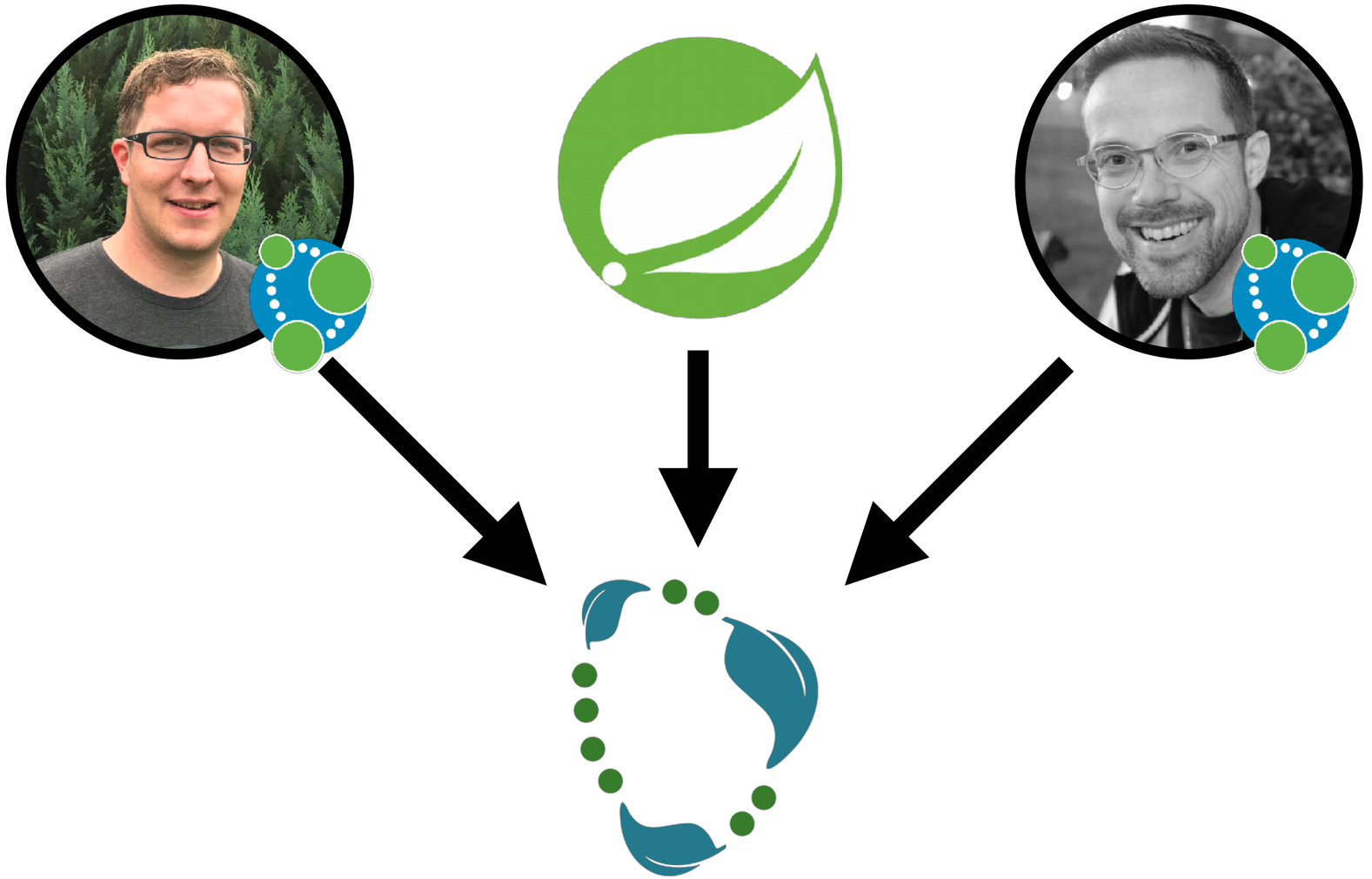
Neo4j is an internet-scale, native graph database which executes connected workloads faster than any other database management system.



Ecosystem

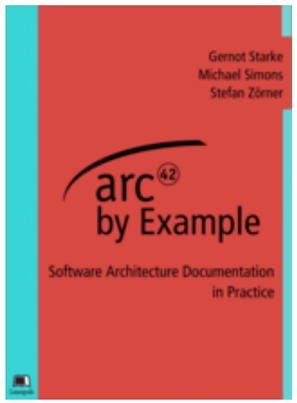
Neo4j Professional Services
300+ partners
47,000 group members
61,000 trained engineers
3.5M downloads

Spring Data and Neo4j



About me

- Neo4j since July 2018
- Java Champion
- Co-Founder and current lead of Java User Group **EuregJUG**
- Author (**Spring Boot 2** und **Arc42 by example**)

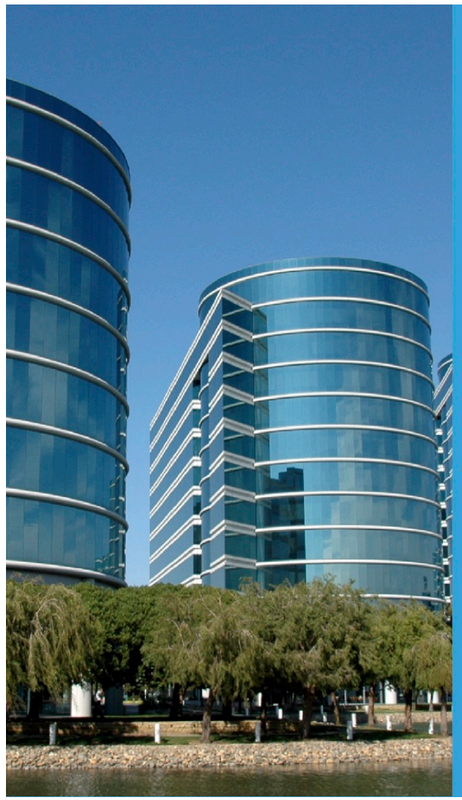


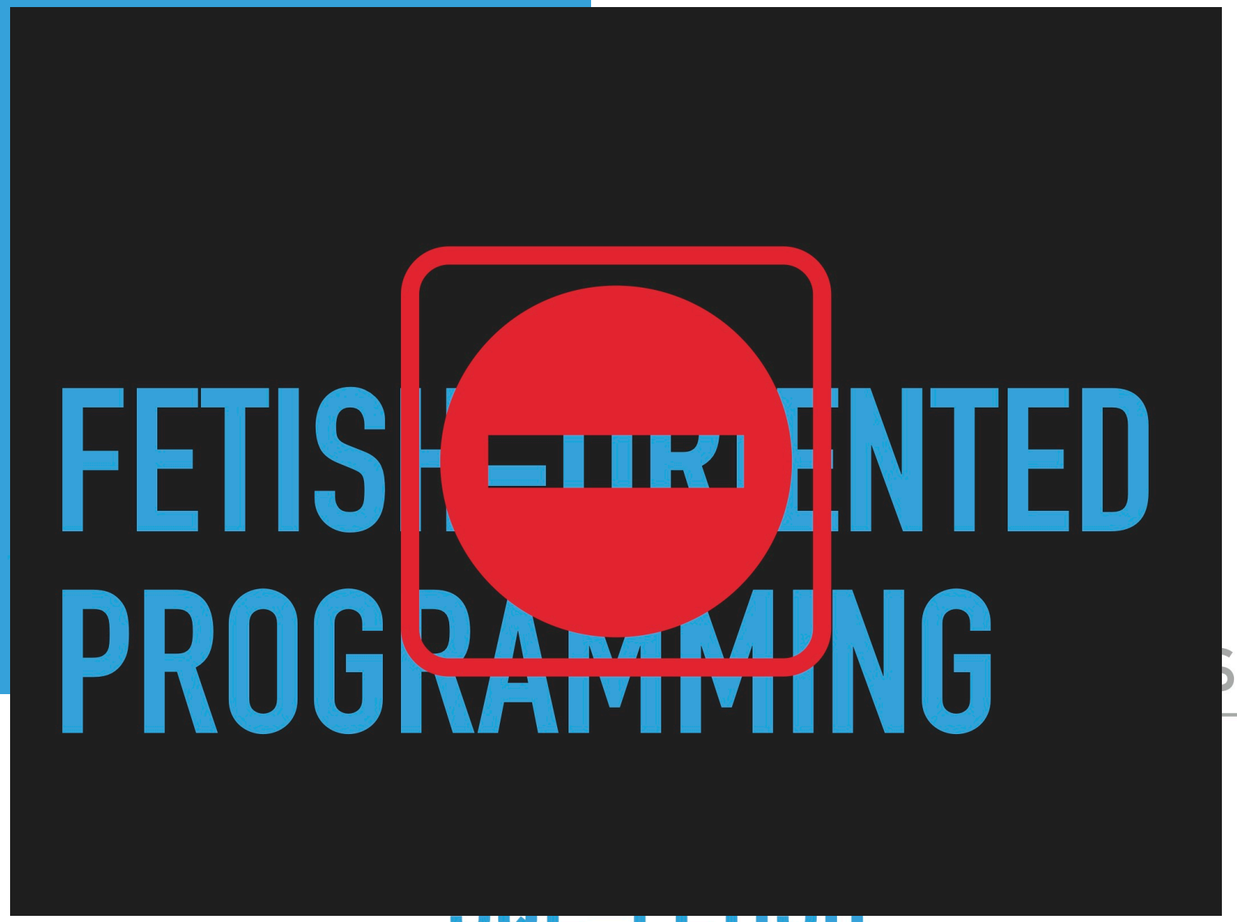
First contact to Neo4j through QAssistant

neo4j

30.12.2018 #festive500 2018 - Ronde van Nederland biking

Known for

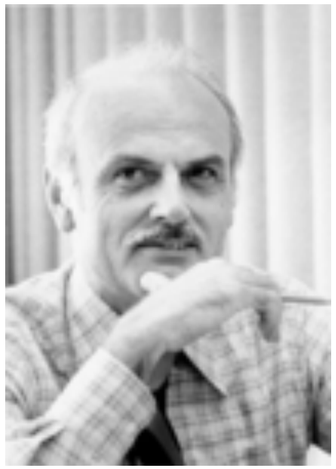


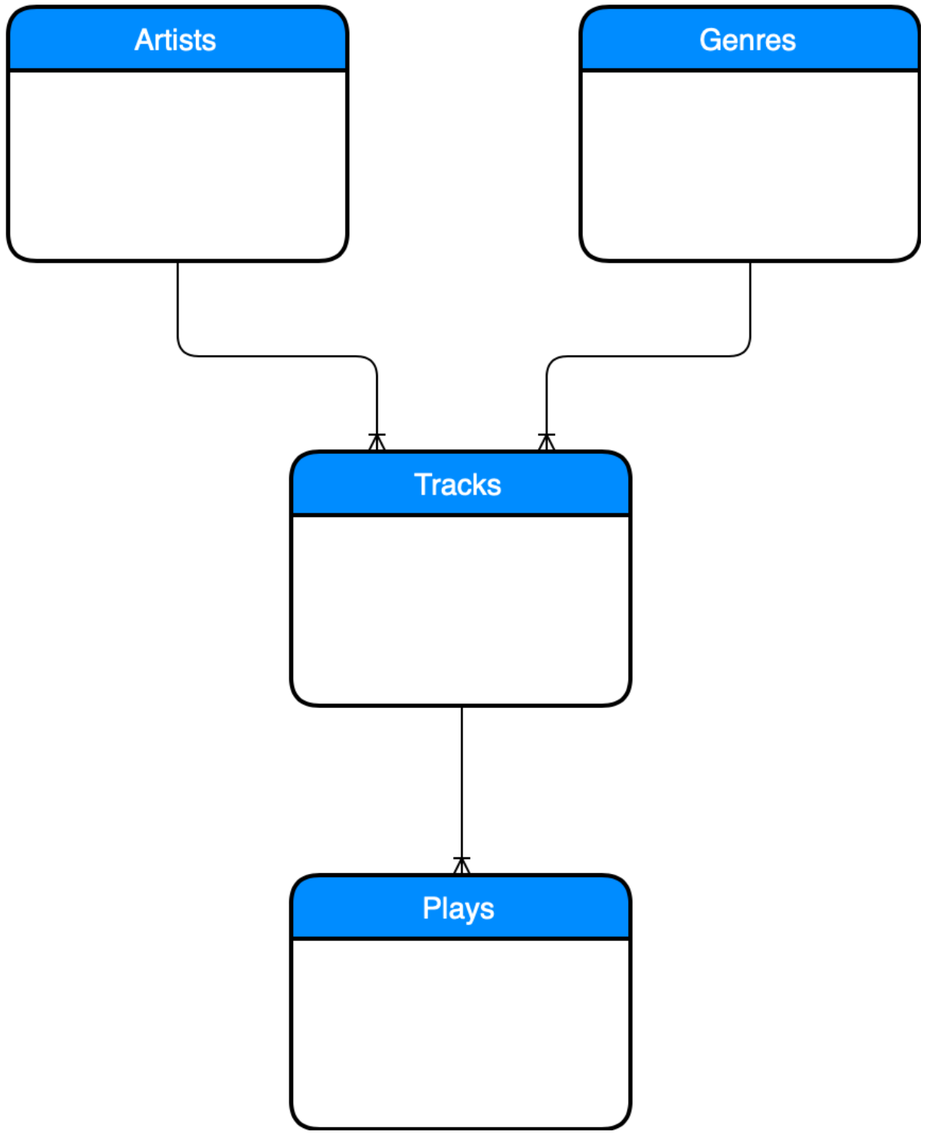


My „business“ domain

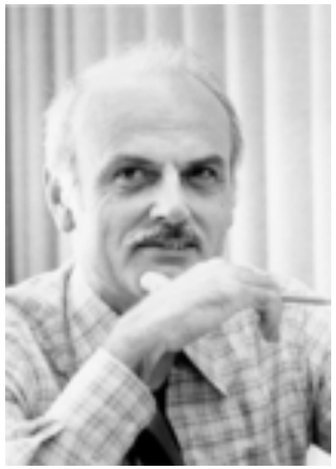
neo4j

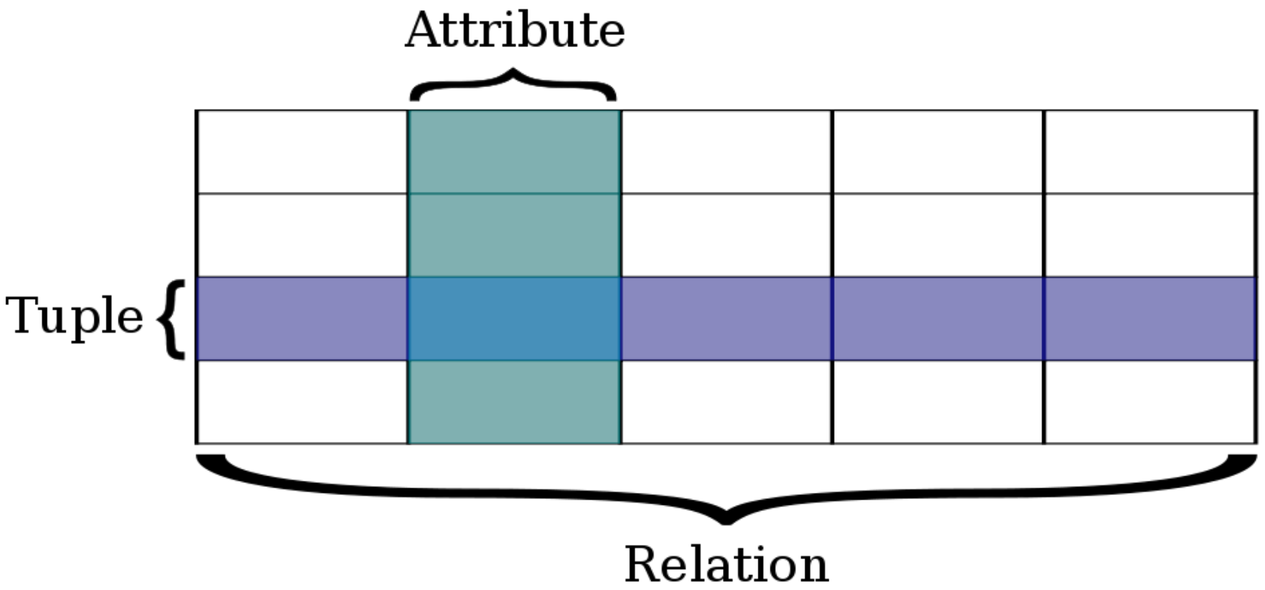
Tracking musical data





Tracking musical data





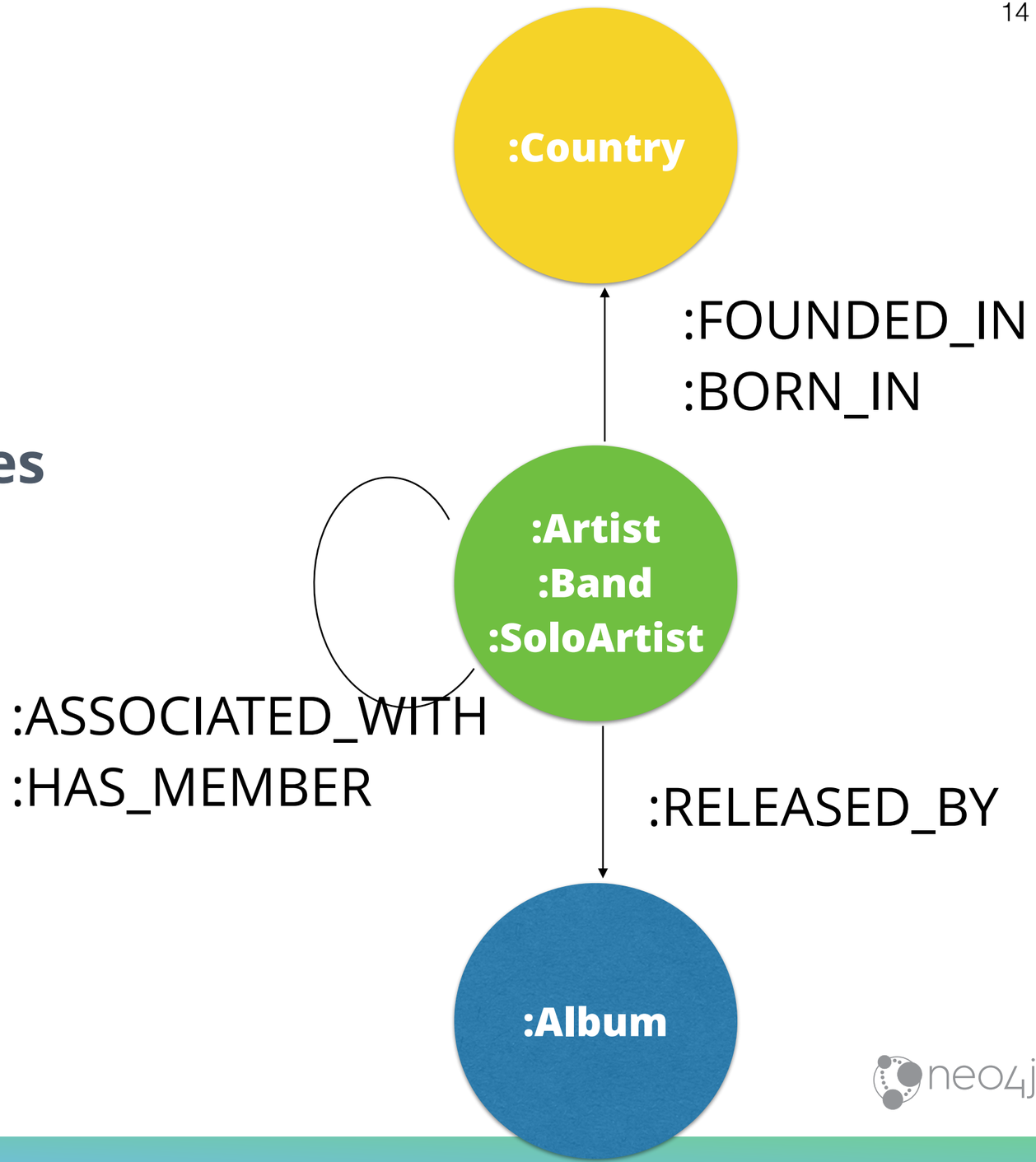
Logical vs physical model

- Logical model designed as ER diagram
- Then normalized
  - All about being free of redundancies
  - UNF (Unnormalized)
  - 1NF: Atomic
  - 2NF: + No partial dependencies
  - 3NF: + No transitive dependencies

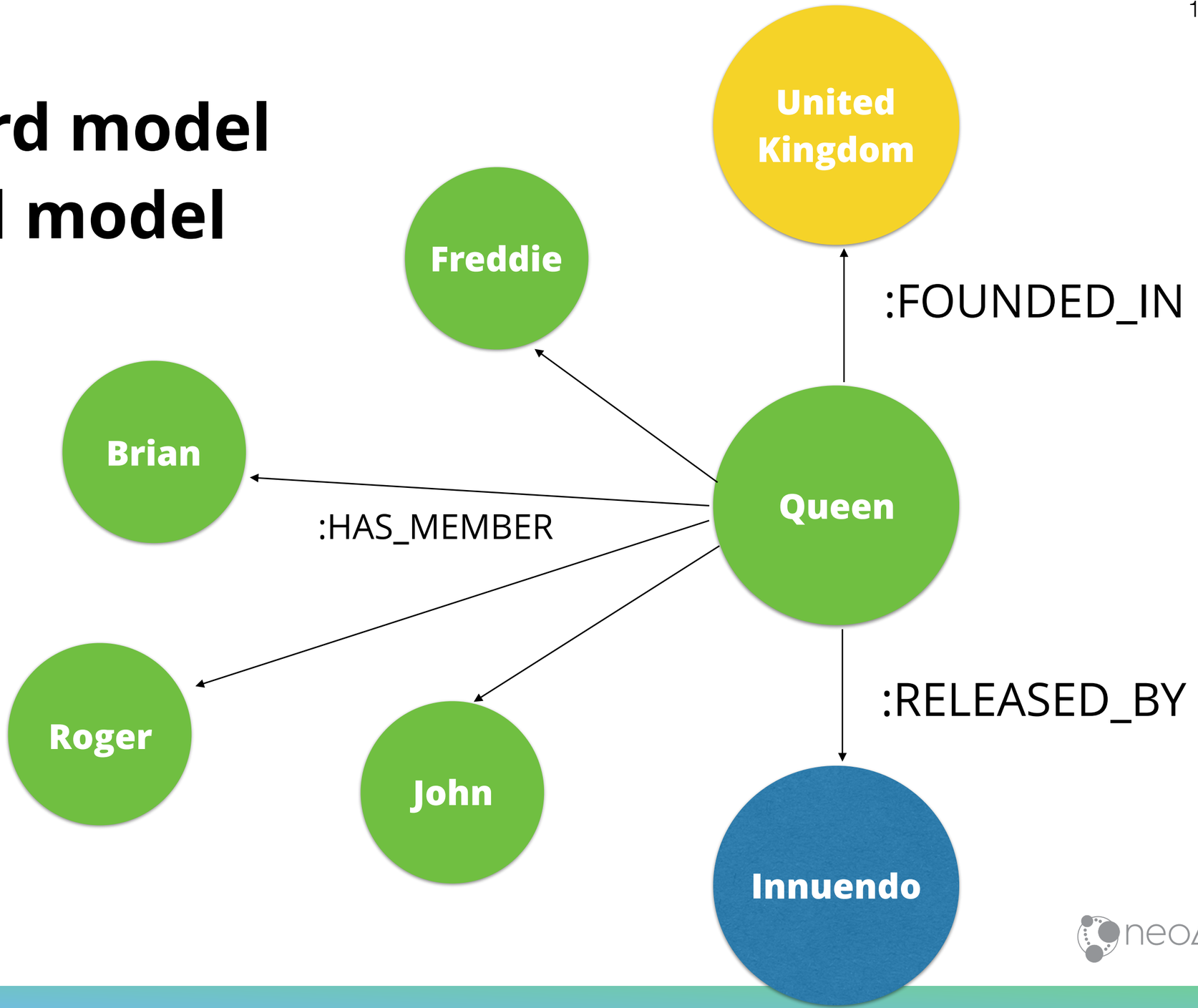
Foreign keys between tables aren't relations!
The tables itself and every query result are.

The whiteboard model IS the physical model

- Bands are **founded in** and solo artists are **born in countries**
- Sometimes Artists are **associated with** other Artists and bands **have member**
- Artists used to release **Albums**



The whiteboard model IS the physical model



A Property Graph

Relationships connect nodes
and represent actions (verbs)

name: Freddie
role: Lead Singer

Nodes represents objects
(Nouns)

Both nodes and relationships
can have properties

Querying

- Cypher is to Neo4j what SQL is to RDBMS:
  A declarative, powerful query language
- https://www.opencypher.org / The GQL Manifesto

```
MATCH (a:Album) -[:RELEASED_BY]→ (b:Band),
      (c) ←[:FOUNDED_IN]- (b) -[:HAS_MEMBER]→ (m) -[:BORN_IN]→ (c2)
WHERE a.name = 'Innuendo'
RETURN a, b, m, c, c2
```

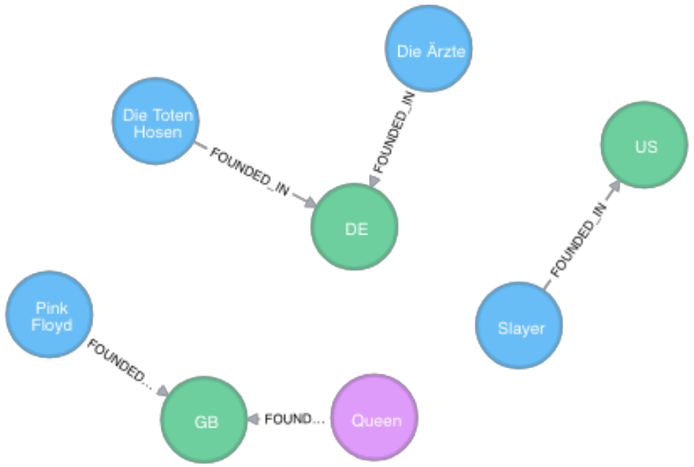
Demo

Getting data into Neo4j

neo4j

LOAD CSV

```
Name;Founded in
Slayer;US
Die Ärzte;DE
Die Toten Hosen;DE
Pink Floyd;GB
```




```
LOAD CSV WITH HEADERS FROM 'http://localhost:8001/data/artists.csv'
AS line FIELDTERMINATOR ';'
MERGE (a:Artist {name: line.Name})
MERGE (c:Country {code: line.`Founded in`})
MERGE (a) -[:FOUNDED_IN]→ (c)
RETURN *
```

Building your own importer

```
public class StatsIntegration {

  @Context public GraphDatabaseService db;

  @Procedure(name = "stats.loadArtistData", mode = Mode.WRITE)
  public void loadArtistData(
       @Name("userName") final String userName,
       @Name("password") final String password,
       @Name("url") final String url) {

    try (var connection = DriverManager.getConnection(url, userName, password);
        var neoTransaction = db.beginTx()) {

      DSL.using(connection)
           .selectFrom(ARTISTS)
           .forEach(a →
             db.execute("MERGE (artist:Artist {name: $artistName}) ", Map.of("artistName", a.getName()))
           );
      neoTransaction.success();
    } catch (Exception e) {}
  }
}
```

APOC

- Not only a guy from the movie „The Matrix“

APOC

- Not only a guy from the movie „The Matrix“
- Also not that guy
- „A Package Of Components“ for Neo4j
- „Awesome Procedures on Cypher“

A huge set of all kinds of extension for Neo4j
https://neo4j-contrib.github.io/neo4j-apoc-procedures/

APOC

- Import / Export
- Graph refactoring
- Job management
- Graph algorithms

apoc.load.jdbc

- Use with single tables
- Or custom SQL statements

apoc.load.jdbc

```
WITH "jdbc:postgresql://localhost:5432/bootiful-music?user=statsdb-dev&password=dev" as url,
     "SELECT DISTINCT a.name as artist_name, t.album, g.name as genre_name, t.year
      FROM tracks t JOIN artists a ON a.id = t.artist_id JOIN genres g ON g.id = t.genre_id
      WHERE t.compilation = 'f'" as sql
CALL apoc.load.jdbc(url,sql) YIELD row
MERGE (decade:Decade {value: row.year-row.year%10})
MERGE (year:Year {value: row.year})
MERGE (year) -[:PART_OF]→ (decade)
MERGE (artist:Artist {name: row.artist_name})
MERGE (album:Album {name: row.album}) -[:RELEASED_BY]→ (artist)
MERGE (genre:Genre {name: row.genre_name})
MERGE (album) -[:HAS]→ (genre)
MERGE (album) -[:RELEASED_IN]→ (year)
```

Demo

Using Neo4j from the JVM

neo4j

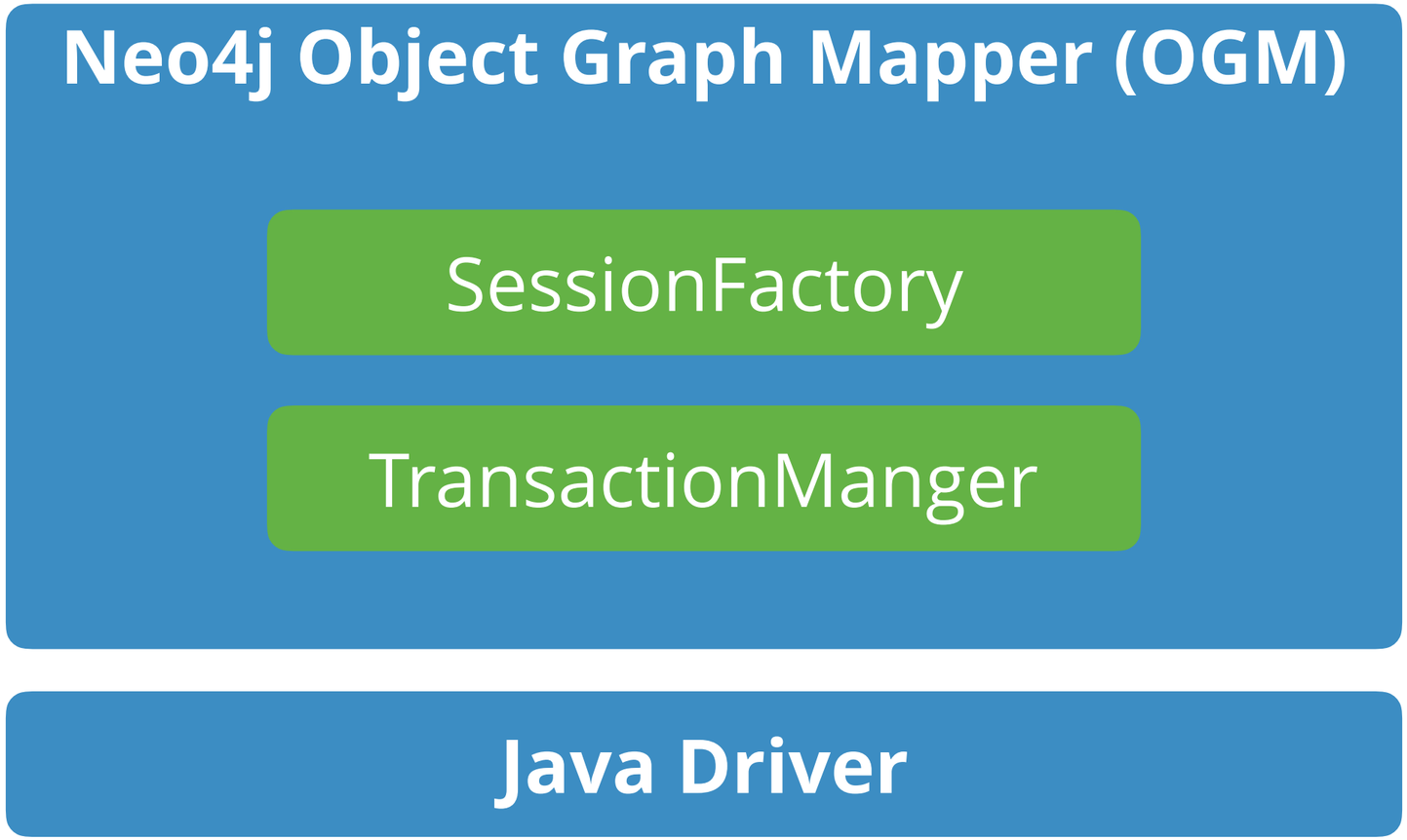
Different endpoints

- Neo4j can run embedded in the same VM
- Has an HTTP endpoint
- Offers the binary **Bolt protocol**
  - Drivers for Java, Go, C#, Seabolt (C), Python, JavaScipt

Working directly with the driver

```
try (
  Driver driver = GraphDatabase.driver(uri, AuthTokens.basic(user, password));
  Session session = driver.session()
) {
  List<String> artistNames =
    session
      .readTransaction(tx → tx.run("MATCH (a:Artist) RETURN a", emptyMap()))
      .list(record → record.get("a").get("name").asString());
}
```

Using Neo4j-OGM



Using Neo4j-OGM

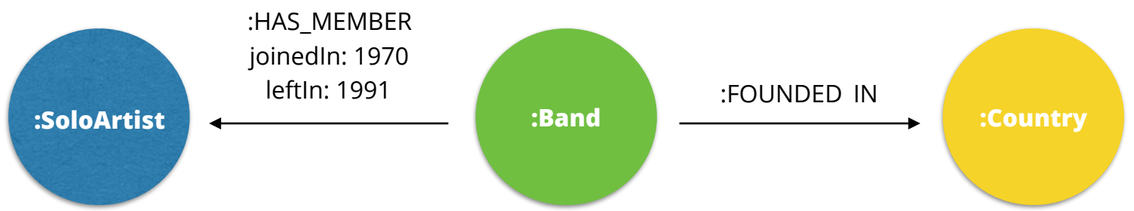
- Unified configuration
- Annotation based
  - Mapping between Classes and Graph Model
- Data access
  - Domain based
  - Through custom queries

```java
@NodeEntity("Band")
public class BandEntity extends ArtistEntity {

  @Id @GeneratedValue
  private Long id;

  private String name;

  @Relationship("FOUNDED_IN")
  private CountryEntity foundedIn;

  @Relationship("ACTIVE_SINCE")
  private YearEntity activeSince;

  @Relationship("HAS_MEMBER")
  private List<Member> member = new ArrayList<>();
}
```

Annotations

```
@RelationshipEntity("HAS_MEMBER")
public static class Member {
  @Id @GeneratedValue
  private Long memberId;

  @StartNode
  private BandEntity band;

  @EndNode
  private SoloArtistEntity artist;

  @Convert(YearConverter.class)
  private Year joinedIn;

  @Convert(YearConverter.class)
  private Year leftIn;
}
```

Annotations



Domain based data access

```
var artist = new BandEntity("Queen");
artist.addMember(new SoloArtistEntity("Freddie Mercury"));

var session = sessionFactory.openSession();
session.save(artist);
```

Domain based data access

```
var queen = session.load(BandEntity.class, 4711);

var allBands = session.loadAll(BandEntity.class);
```

Domain based data access

```
session.delete(queen);
// OR
session.delete(queen, 1);

session.deleteAll(BandEntity.class);
```

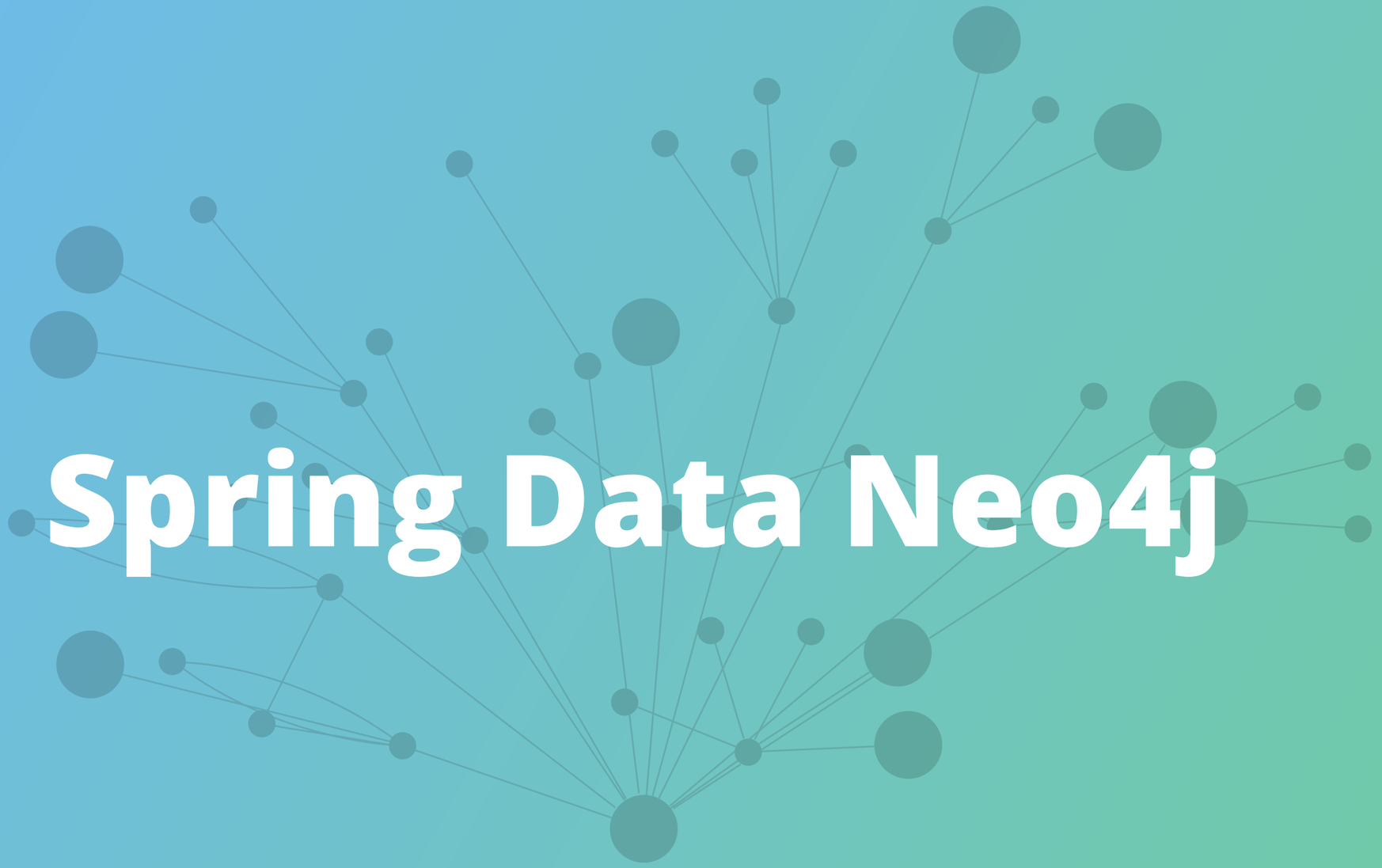
Data access with custom queries

```
var britishBands = session.query(
      ArtistEntity.class,
      "MATCH (b:Band) -[:FOUNDED_IN]→ (:Country {code: 'GB'})", emptyMap());

Result result = session.query(
  "MATCH (b:Artist) ←[r:RELEASED_BY]- (a:Album) -[:RELEASED_IN]→ () -
[:PART_OF]→ (:Decade {value: $decade})"
  "WHERE b.name = $name" +
  "RETURN b, r, a",
  Map.of("decade", 1970, "name", "Queen")
);
```

Works with

- „Plain“ Java
- Micronaut
- Spring
- Spring Boot

Spring Data Neo4j



neo4j

Spring Data Neo4j

- Very early Spring Data Module
  - First Version ~2010 (Emil Eifrem, Rod Johnson)
- Build on top of Neo4j-OGM
- Part of the Spring Data release trains
- Offers
  - Derived finder methods
  - Custom results and projections
  - Domain Events
- Integrated in Spring Boot

Spring Data Neo4j

- Can be used store agnostic
- Without Cypher
- Or „Graph aware“
  - limiting the fetch size
  - Custom Cypher

Domain based data access revised

```
interface BandRepository extends Repository<BandEntity, Long> {
}
```

Domain based data access revised

```
interface BandRepository extends Neo4jRepository<BandEntity, Long> {
}
```

- CRUD Methods
  - (save, findById, delete, count)
- Supports `@Depth` annotation as well as `depth` argument

Domain based data access revised

```
var artist = new BandEntity("Queen");
artist.addMember(new SoloArtistEntity("Freddie Mercury"));

artist = bandRepository.save(artist);
```

Domain based data access revised

```
var artist = bandRepository.findByName("Nickelback")

artist.ifPresent(bandRepository::delete);
```

Derived finder methods

```
interface AlbumRepository extends Neo4jRepository<AlbumEntity, Long> {

  Optional<AlbumEntity> findOneByName(String x);

  List<AlbumEntity> findAllByNameMatchesRegex(String name);

  List<AlbumEntity> findAllByNameMatchesRegex(
    String name, Sort sort, @Depth int depth);

  Optional<AlbumEntity> findOneByArtistNameAndName(
     String artistName, String name);
}
```

Custom queries

```
interface AlbumRepository extends Neo4jRepository<AlbumEntity, Long> {
  @Query(value
    = " MATCH (album:Album) - [:CONTAINS] → (track:Track)"
    + " MATCH p=(album) - [*1] - ()"
    + " WHERE id(track) = $trackId"
    + "   AND ALL(relationship IN relationships(p) "
    + "               WHERE type(relationship) <> 'CONTAINS')"
    + " RETURN p"
  )
  List<AlbumEntity> findAllByTrack(Long trackId);
}
```

Custom results

```
@QueryResult
public class AlbumTrack {
  private Long id;

  private String name;

  private Long discNumber;

  private Long trackNumber;
}
```

Custom results

```
interface AlbumRepository extends Neo4jRepository<AlbumEntity, Long> {
  @Query(value
    = " MATCH (album:Album) - [c:CONTAINS] → (track:Track) "
    + " WHERE id(album) = $albumId"
    + " RETURN id(track) AS id, track.name AS name, "
    + "          c.discNumber AS discNumber, c.trackNumber AS trackNumber"
    + " ORDER BY c.discNumber ASC, c.trackNumber ASC"
  )
  List<AlbumTrack> findAllAlbumTracks(Long albumId);
}
```

Spring Transactions

```
public class ArtistService {

  @Transactional
  public void deleteArtist(Long id) {

    this.bandRepository.findById(id).ifPresent(a → {
      session.delete(a);
      session.query("MATCH (a:Album) WHERE size((a)-[:RELEASED_BY]→(:Artist))=0 DETACH DELETE a", emptyMap());
      session.query("MATCH (t:Track) WHERE size((:Album)-[:CONTAINS]→(t))=0 DETACH DELETE t", emptyMap());
    });
  }
}
```

Spring Transactions

```
TransactionTemplate transactionTemplate;

return transactionTemplate.execute(t → {
  ArtistEntity artist = this.findArtistById(artistId).get();

  var oldLinks = artist.updateWikipediaLinks(newLinks);
  session.save(artist);
  oldLinks.forEach(session::delete);

  return artist;
});
```

With Spring Boot: Configuration properties and auto config

```
org.springframework.boot:spring-boot-starter-neo4j
```

```
spring.data.neo4j.username=neo4j
spring.data.neo4j.password=music
spring.data.neo4j.uri=bolt://localhost:7687

spring.data.neo4j.embedded.enabled=false
```

With Spring Boot: Test-Slice

```
@DataNeo4jTest
@TestInstance(Lifecycle.PER_CLASS)
class CountryRepositoryTest {

  private final Session session;

  private final CountryRepository countryRepository;

  @Autowired
  CountryRepositoryTest(Session session, CountryRepository countryRepository) {
    this.session = session;
    this.countryRepository = countryRepository;
  }

  @BeforeAll
  void createTestData() {}

  @Test
  void getStatisticsForCountryShouldWork() {}
}
```

Spring Data Neo4j: Don'ts

- Not for batch processing
- Don’t abuse derived method names

  i.e.
  ```
  Optional<AlbumEntity>
  findOneByArtistNameAndNameAndLiveIsTrueAndReleasedInValue(String artistName,
  String name, long year)
  ```
- Don’t follow your Graph model blindly while modeling the domain
  - Graph model usually tailored to answer specific question
  - Domain often follows a different use-case

Don’t follow your Graph model blindly while modeling the domain

```
@NodeEntity("Artist")
public class ArtistEntity {

  private String name;

  @Relationship(
    value = "RELEASED_BY",
    direction = INCOMING)
  private List<AlbumEntity> albums;
}
```

```
@NodeEntity("Album")
public class AlbumEntity {

  @Relationship("RELEASED_BY")
  private ArtistEntity artist;

  @Relationship("CONTAINS")
  private List<TrackEntity> tracks;
}

@NodeEntity("Track")
public class TrackEntity {

  @Relationship(
    value = "CONTAINS", direction = INCOMING)
  private List<AlbumEntity> tracks;
}
```

Better approach

```
@NodeEntity("Artist")
public class ArtistEntity {

  private String name;
}

@NodeEntity("Album")
public class AlbumEntity {

  @Relationship("RELEASED_BY")
  private ArtistEntity artist;
}

@QueryResult
public class AlbumTrack {

  private String name;

  private Long trackNumber;
}
```

```
interface AlbumRepository extends Repository<AlbumEntity, Long> {
  List<AlbumEntity> findAllByArtistNameMatchesRegex(
    String artistName,
    Sort sort);

  @Query(value
    = " MATCH (album:Album) - [c:CONTAINS] → (track:Track) "
    + " WHERE id(album) = $albumId"
    + " RETURN track.name AS name, c.trackNumber AS trackNumber"
    + " ORDER BY c.discNumber ASC, c.trackNumber ASC"
  )
  List<AlbumTrack> findAllAlbumTracks(long albumId);
}
```

Demo


What are your top devices?
Where are your users?
Sessions by country
Sessions by device
COHORT ANALYSIS REPORT
Czechia
Slovakia

Some advanced queries

neo4j

More Cypher

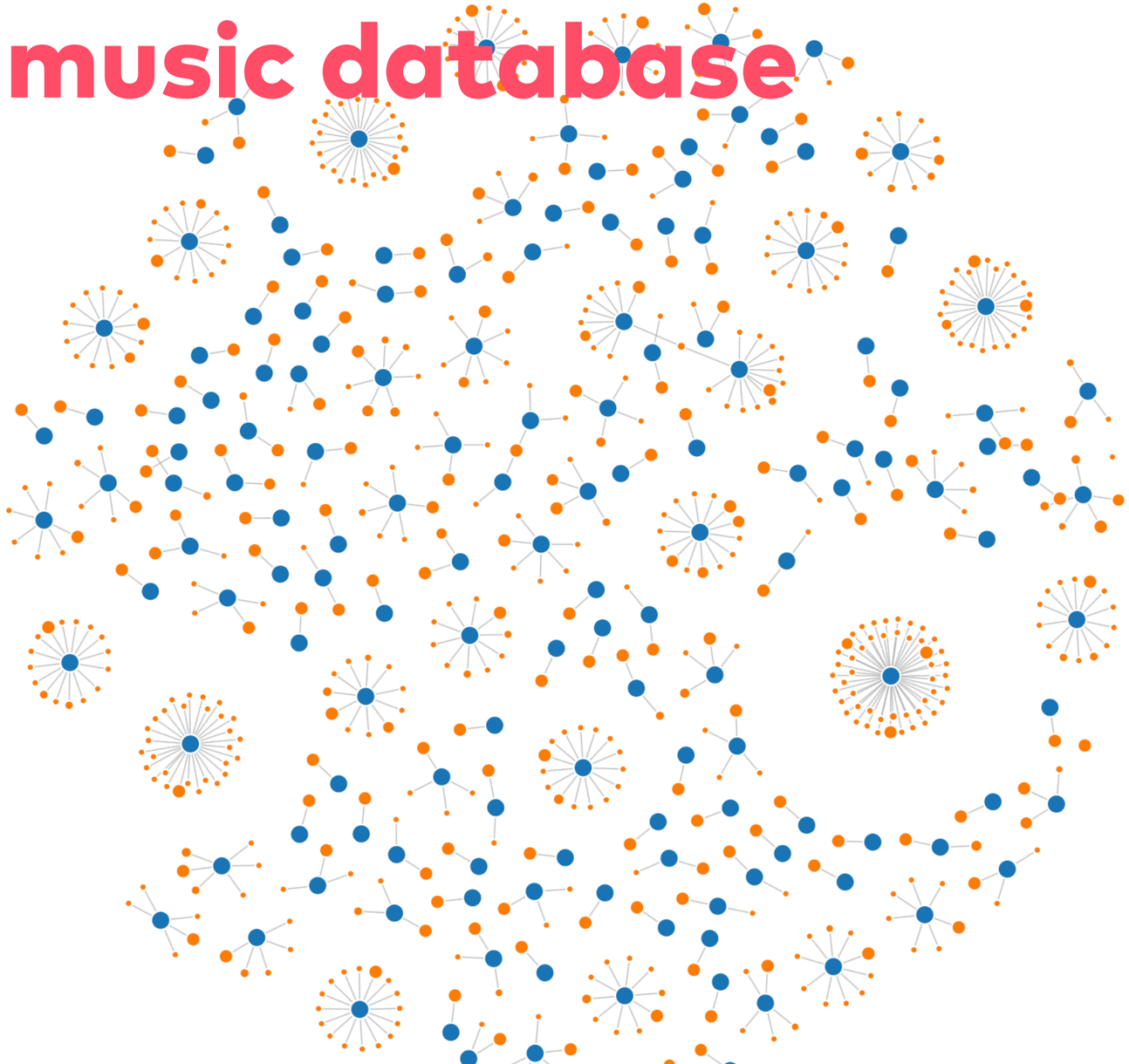
Putting it all together

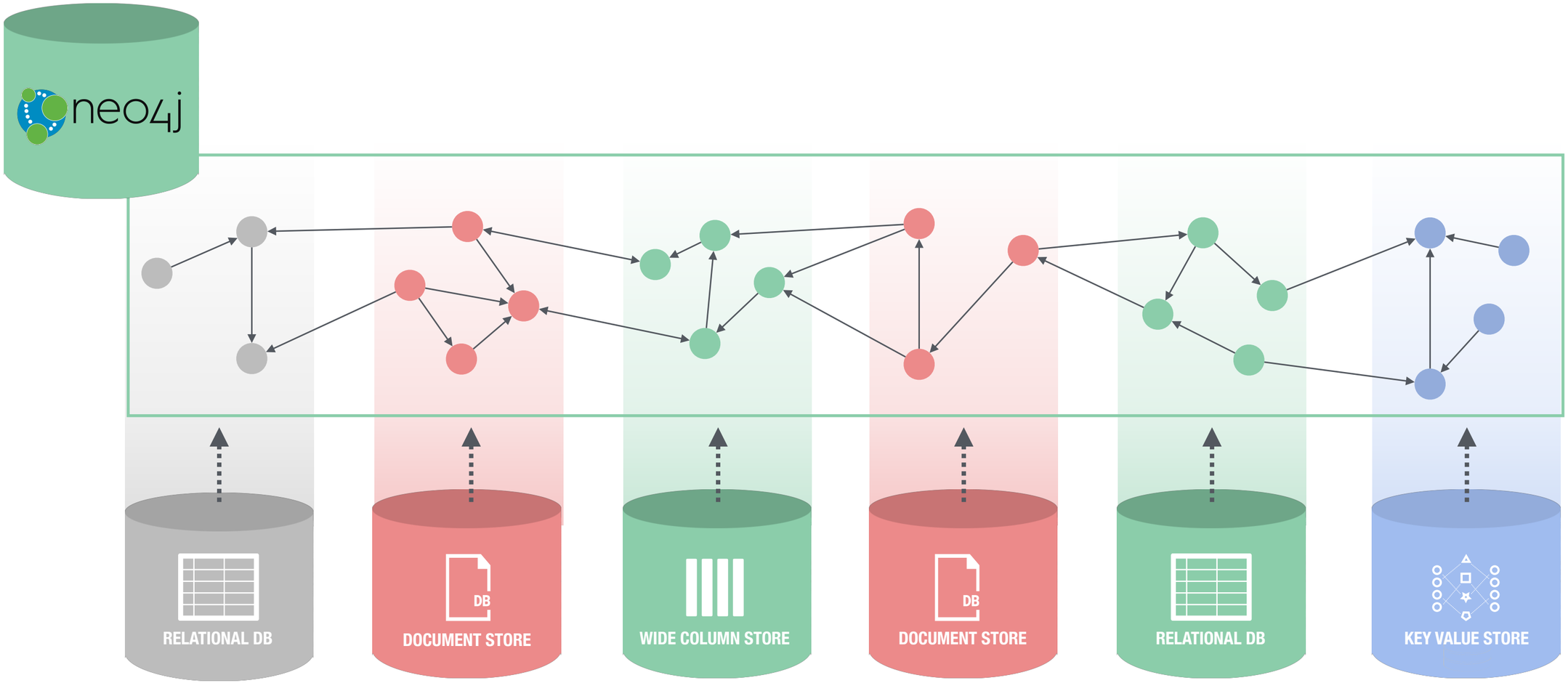
neo4j

Genre

My personal music database

Name	
2[illegible] Germany Punk Rock	174
2010 Germany Punk Rock	8271
1990 Germany Punk Rock	6919
2010 Germany Rock	6778
2000 United Kingdom Rock	5116
1970 United Kingdom Rock	3554
1980 United Kingdom Rock	3433
2010 United Kingdom Rock	3224
2000 United Kingdom Heavy Metal	2370
1990 United States Heavy Metal	2314



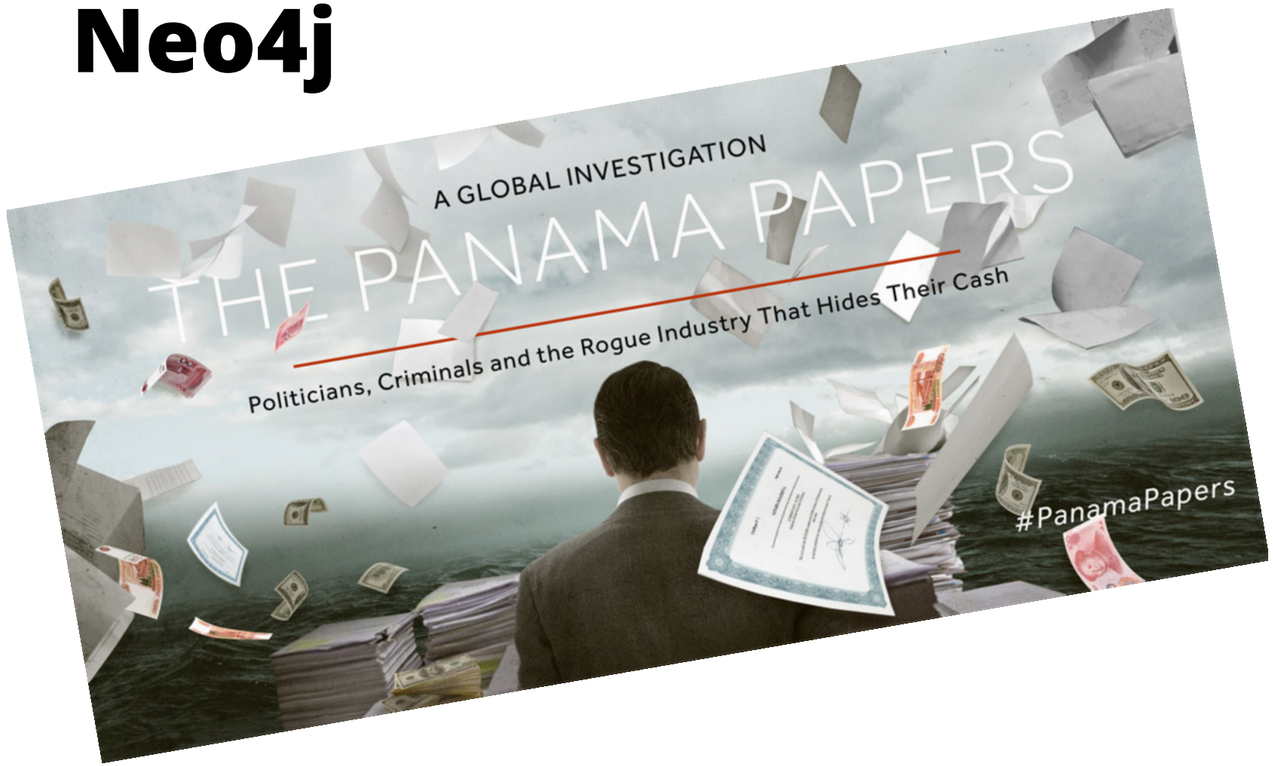


Leveraging Cross-Silo Connections

Real use-cases

neo4j

Neo4j



ICIJ - International Consortium of Investigative Journalists

https://neo4j.com/blog/icij-neo4j-unravel-panama-papers/

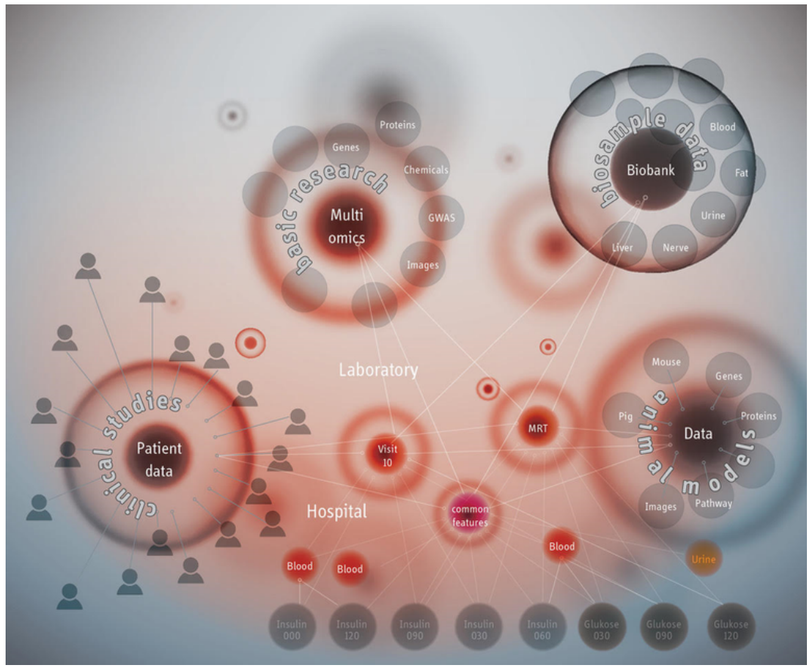
https://neo4j.com/blog/analyzing-panama-papers-neo4j/

https://neo4j.com/blog/analyzing-paradise-papers-neo4j/

Neo4j

„In biology or medicine, data is connected. You know that entities are connected -- they are dependent on each other. The reason why we chose graph technology and Neo4j is because all the entities are connected.“

Dr Alexander Jarasch, DZD German centre of diabetic research



https://www.zdnet.com/article/using-graph-database-technology-to-tackle-diabetes/

Try yourself

neo4j

neo4j.com/graphtour

News Blog Support Company Contact Us **Download**

neo4j PRODUCTS SOLUTIONS PARTNERS CUSTOMERS LEARN DEVELOPERS Search

@Neo4j **#GraphTour**

GraphTour 2019 Brings Neo4j to a City Near You

Neo4j is hitting the road to bring a full day of content-rich sessions on how graph databases are revolutionizing the modern enterprise. This one-day event will turn you into a graph expert — no matter your technical background or familiarity with graph technology.

Neo4j

- https://neo4j.com/download/
  - Neo4j Desktop (Analyst centric)
  - Neo4j Server (Community and Enterprise Edition)
    Community Edition: GPLv3
    Enterprise Edition: Proprietary

Neo4j Datasets

- https://neo4j.com/sandbox-v2/
  - Preconfigured instance with several different datasets
- https://neo4j.com/graphgists/
  - Neo4j Graph Gists, Example Models and Cypher Queries
- https://offshoreleaks.icij.org/
  - Data convolutes mentioned early

My „Bootiful Music“ project

- https://github.com/michael-simons/bootiful-music
  - Contains docker-compose-scripts for both relational database and Neo4j Instances
- Two Spring Boot applications
  - charts: the relational part of the application
  - knowledge: the graph application
- etl: the custom Neo4j plugin
- A Micronaut demo as well

Resources

- Demo:
  github.com/michael-simons/bootiful-music
- A series of blog posts: From relational databases to databases with relations
  https://info.michael-simons.eu/2018/10/11/from-relational-databases-to-databases-with-relations/
- Slides: speakerdeck.com/michaelsimons
- Curated set of SDN / OGM tips
  https://github.com/michael-simons/neo4j-sdn-ogm-tips
- GraphTour 2019: https://neo4j.com/graphtour/
- (German) Spring Boot Book
  @SpringBootBuch // springbootbuch.de

Thank you!

neo4j

Images

- Medical graph: DZD German centre of diabetic research
- Codd: Wikipedia
- Apoc and Cypher: Stills from the motion picture „The Matrix“
- Demo:
  https://unsplash.com/photos/Uduc5hJX2Ew
  https://unsplash.com/photos/FlPc9_VocJ4
  https://unsplash.com/photos/gp8BLyaTaA0