Graphdatenbanksysteme

Ein Überblick

Benjamin Gehrels
benjamin@gehrels.info
GitHub: @BGehrels

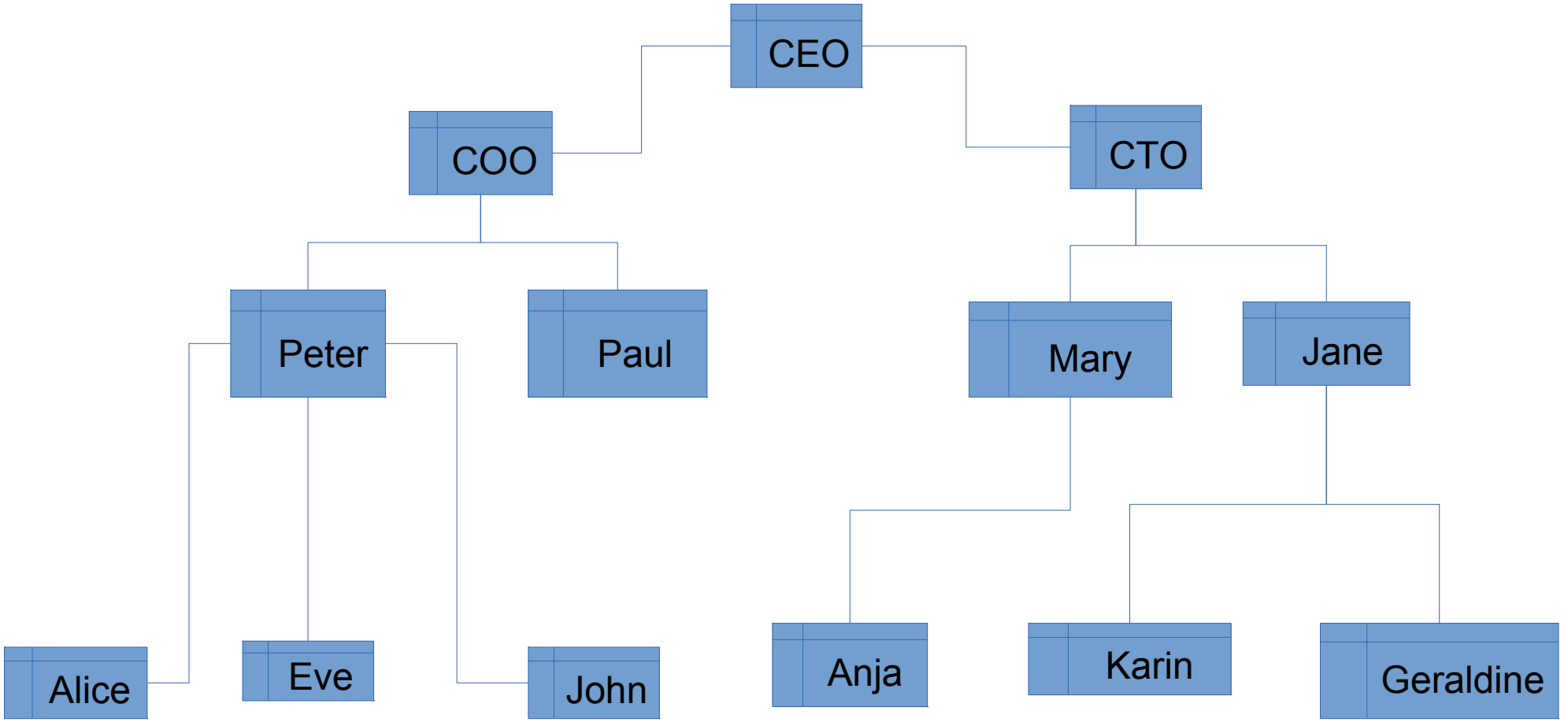
Was ist das?

```
WITH RECURSIVE manager ( level, managerId) AS (
   SELECT
      1                       AS depth,
      employees.managerId     AS managerId
   FROM employees
   WHERE employees.employeeId=13
    UNION ALL
   SELECT
      manager.depth+1,
      employees.managerId
   FROM manager
   INNER JOIN employees
   ON employees.employeeId=manager.managerId
)
SELECT
manager.depth,
manager.employeeId
FROM manager
ORDER BY manager.depth ASC;
```

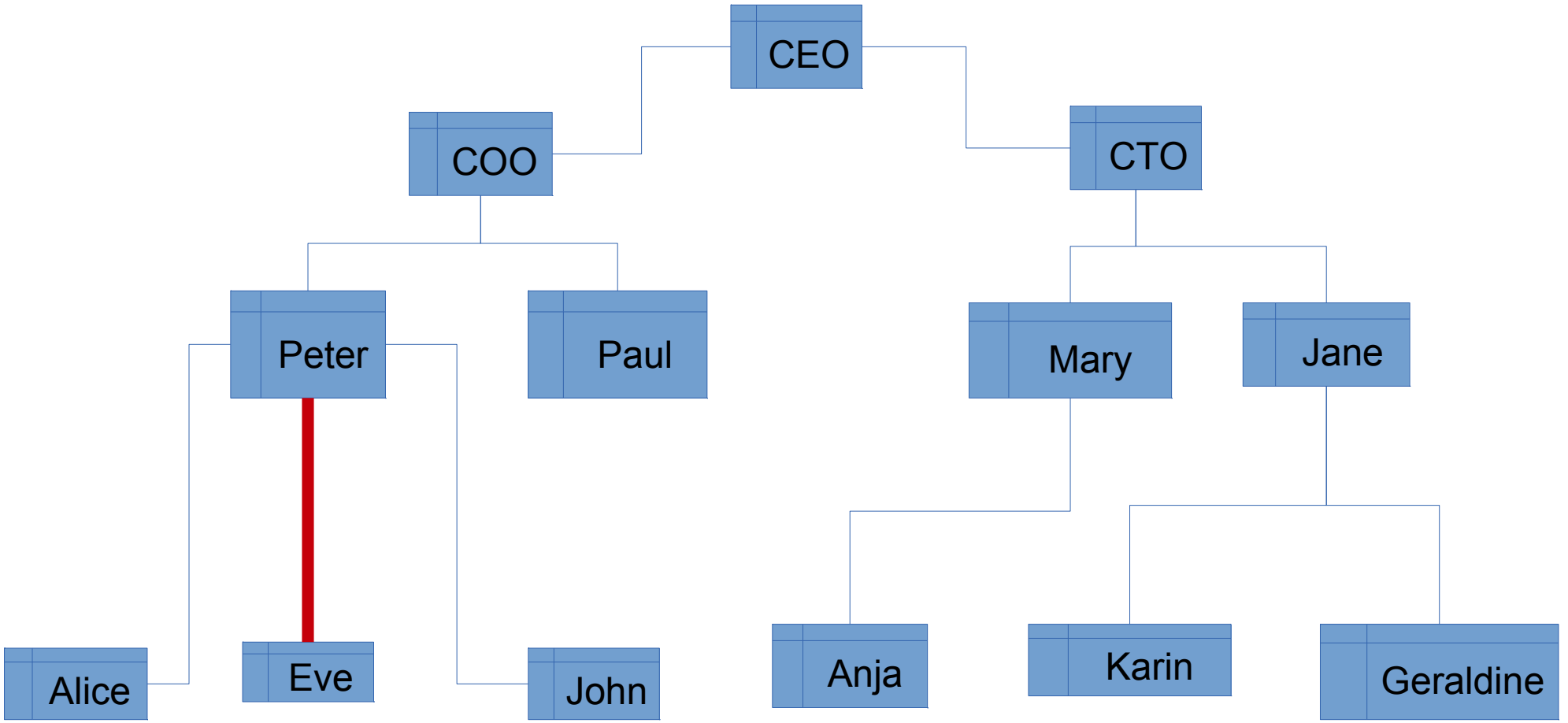
Relationale Datenbank

employeeId	*employeeName*	*managerId*
1	Alice	6
2	Anja	11
3	CEO	null
4	COO	3
5	CTO	3
6	Peter	4
7	Geraldine	8
8	Jane	5
9	John	6
10	Karin	8
11	Mary	11
12	Paul	4
13	Eve	6

Graph


CEO
COO
CTO
Peter
Paul
Mary
Jane
Alice
Eve
John
Anja
Karin
Geraldine

Graph


CEO
COO
CTO
Peter
Paul
Mary
Jane
Alice
Eve
John
Anja
Karin
Geraldine

Relationale Datenbank

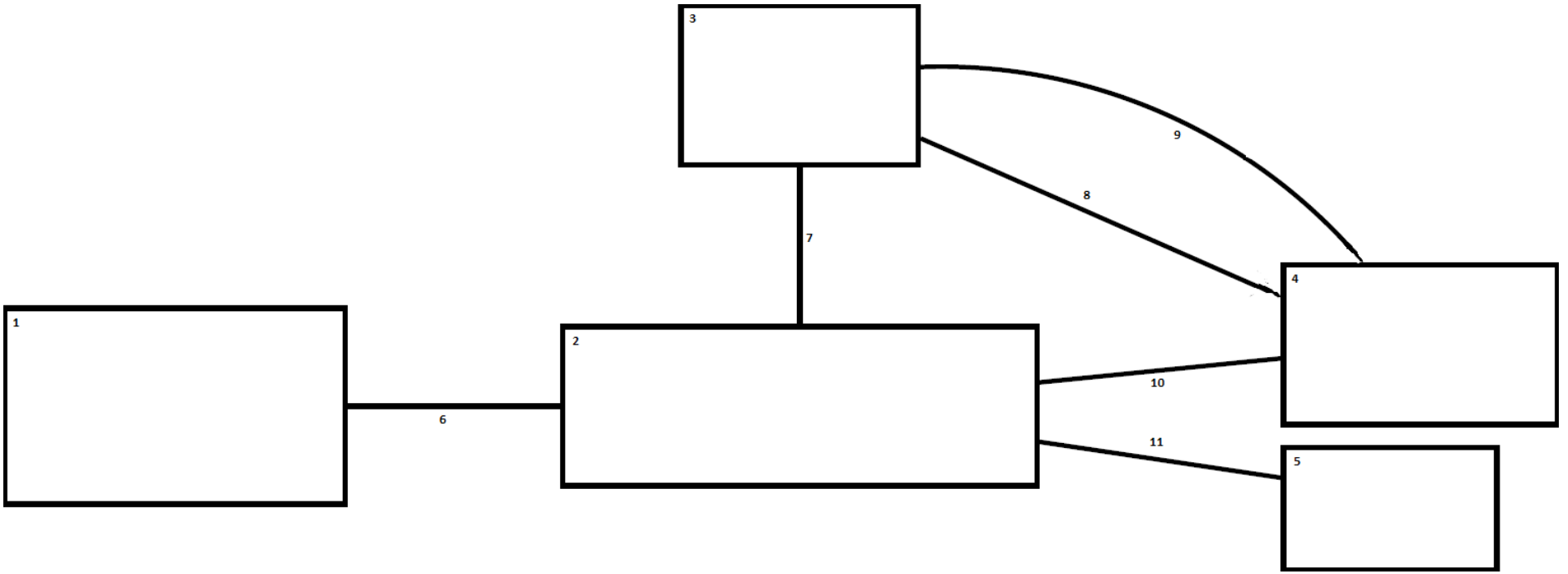
employeeId	*employeeName*	*managerId*
1	Alice	6
2	Anja	11
3	CEO	null
4	COO	3
5	CTO	3
6	Peter	4
7	Geraldine	8
8	Jane	5
9	John	6
10	Karin	8
11	Mary	11
12	Paul	4
13	Eve	6

Relationale Datenbank

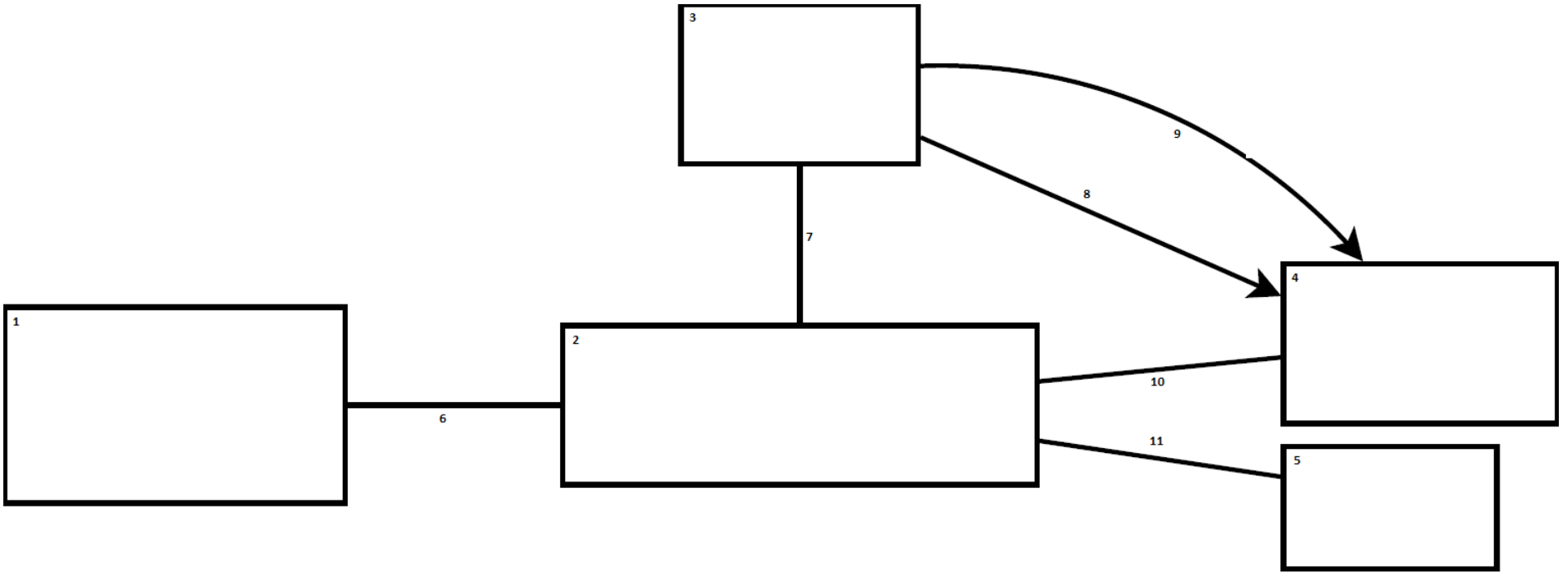
employeeId	*employeeName*	*managerId*
1	Alice	6
2	Anja	11
3	CEO	null
4	COO	3
5	CTO	3
6	Peter	4
7	Geraldine	8
8	Jane	5
9	John	6
10	Karin	8
11	Mary	11
12	Paul	4
13	Eve	6

Graphdatenbanken

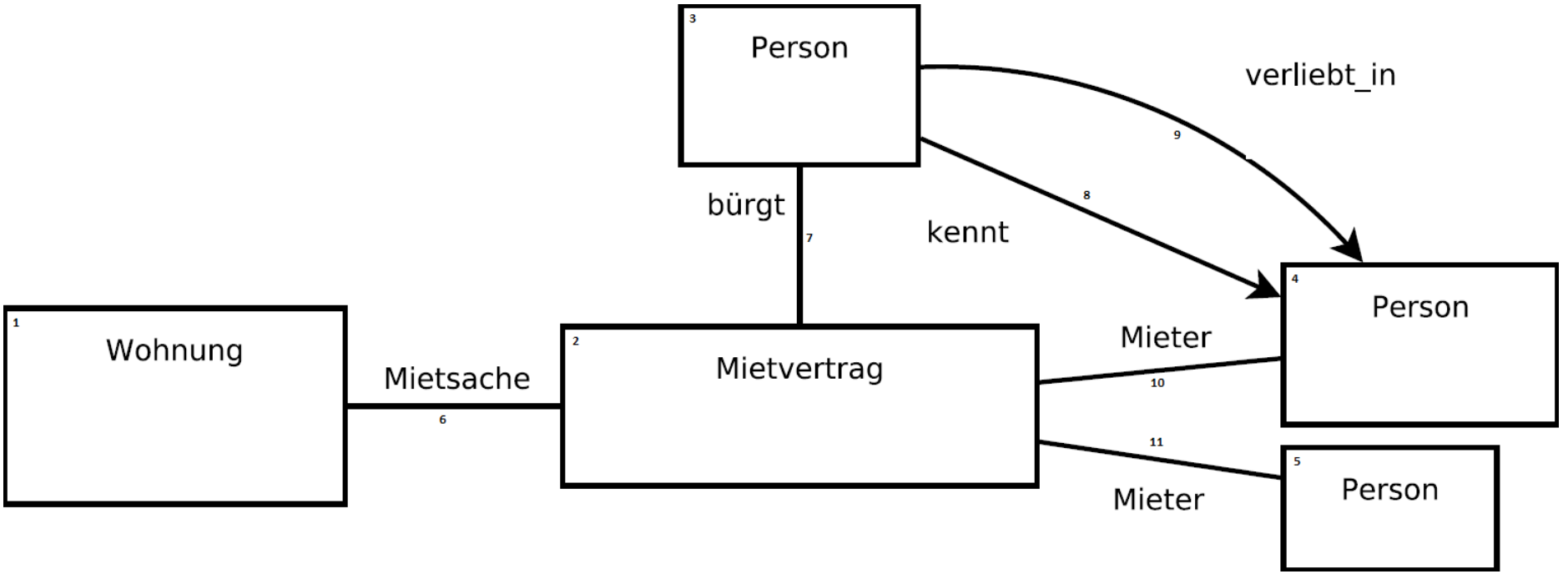
Datenmodell



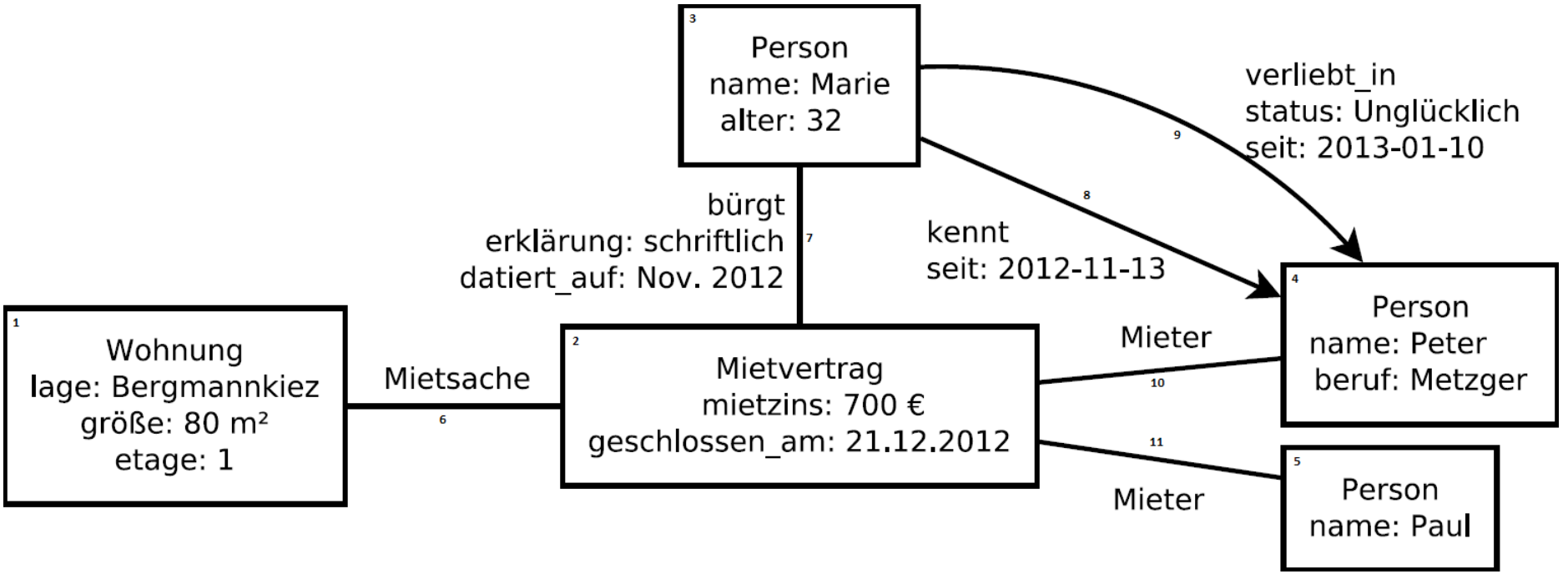
Datenmodell



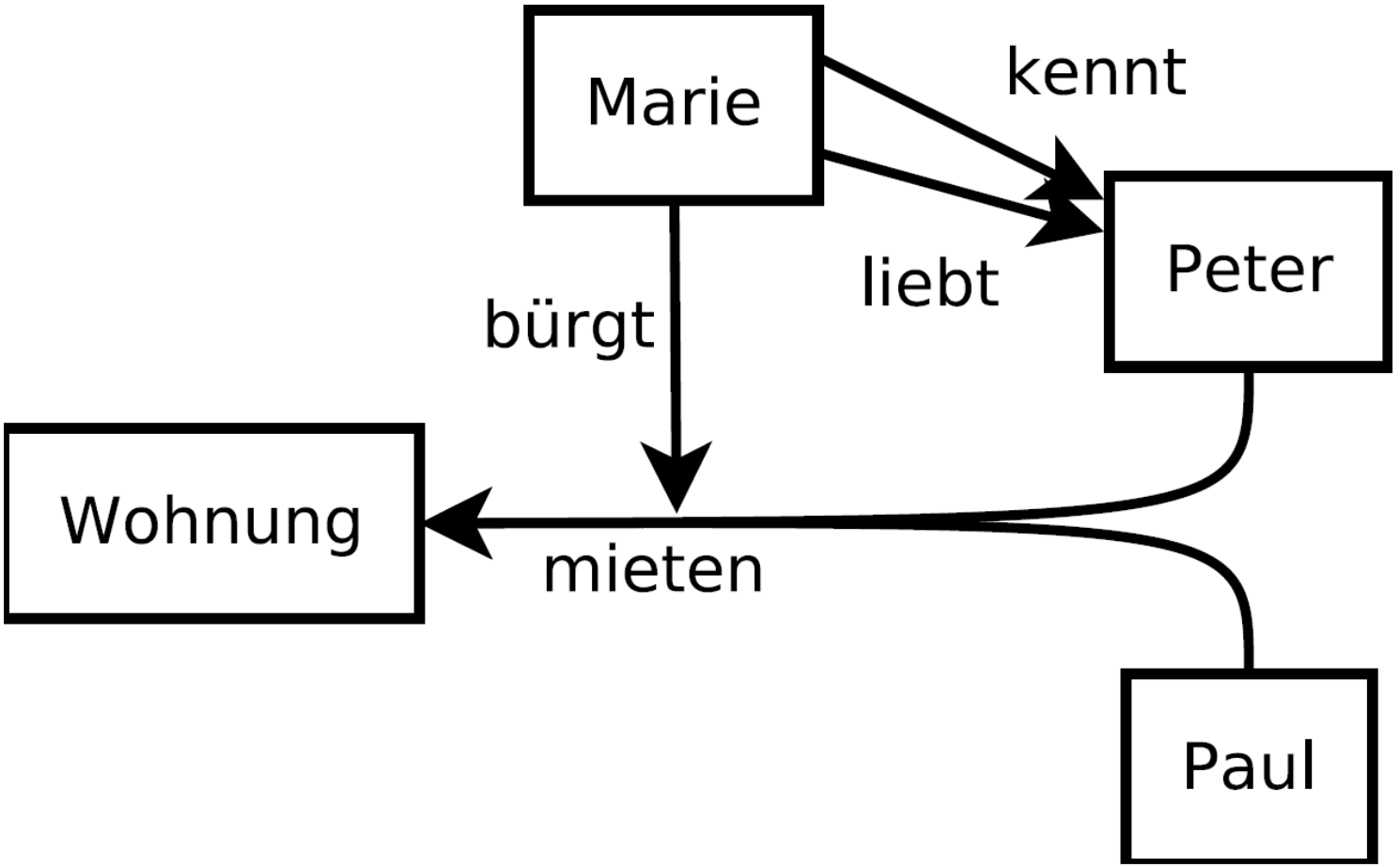
Datenmodell



Datenmodell



Datenmodell



Graphdatenbanksysteme

- Neo4j
- DEX
- HyperGraphDB
- FlockDB
- OrientDB
- InfoGrid
- InfiniteGraph
- Allegro Graph
- Sones GraphDB
- VertexDB
- Apache Giraph
- GraphLab

Graphdatenbanksysteme

- Neo4j
- DEX
- HyperGraphDB
- FlockDB
- OrientDB
- InfoGrid
- InfiniteGraph
- Allegro Graph
- Sones GraphDB
- VertexDB

- Apache Giraph
- GraphLab

Anfragemechanismen

Anfragemechanismen

Elementare Operationen

Lesend

- node by id
- edge by id
- Attribute
- Eingehende Kanten
- Ausgehende Kanten
- Nachbarknoten
- Indizes

Schreibend

- CRUD node
- CRUD edge
- CRUD attribute
- CRUD label
- Indexupdates

Anfragemechanismen

Mengenbasierte Anfragen

```
List<PagedNodeIdList> result = myFlockDBConnection
.select(
  difference(
    intersect(
      simpleSelection(personAId, FOLLOWS, INCOMING),
      simpleSelection(personBId, FOLLOWS, INCOMING)
    ),
    simpleSelection(personCId, BLOCKS, OUTGOING)
  )
)
.execute();
```

Anfragemechanismen

Traversierende Anfragen

```
Iterable<Path> traverser = traversal().breadthFirst()
  .relationships(withName("LIKE"), OUTGOING)
  .relationships(withName("FRIEND"), INCOMING)
  .relationships(withName("LOVES"), BOTH)
  .evaluator(includingDepths(1,3))
  .traverse(startNodes);


for (Path traversedPath : traverser)
  System.out.println(traversedPath);
```

Anfragemechanismen
SPARQL

- Für RDF und Semantic Web entwickelt

```
PREFIX abc: <http://example.com/ontology#>
SELECT ?capital ?country
  WHERE {
    ?x abc:cityname ?capital ;
       abc:isCapitalOf ?y .
    ?y abc:countryname ?country ;
       abc:isInContinent abc:Africa .
  }
```

Anfragemechanismen

Gremlin

```
g.v(1).outE.filter{it.weight<1.0}.inV
```

Anfragemechanismen

Cypher

```
START employee=node(13)
MATCH manager-[pfad:MANAGES*]->employee
RETURN manager.id, length(pfad) AS depth
ORDER BY depth ASC
```

Anfragemechanismen

Cypher

```
START person = node:personen(name="John Doe"),
      c = node(15)
MATCH a-[:LIEBT]->b,
      b-[:LIEBT]->c,
      c-[:LIEBT]->a,
      p=shortestPath(person-[:FREUND_VON*3]-a)
WHERE c.alter < 18
RETURN a, b, c, c.alter, LENGTH(p) AS abstand
ORDER BY abstand ASC
```

Die Graphdatenbanksysteme im Einzelnen

HyperGraphDB

- Entwicklerin: Kobrix Software
- Embedded (Java); LGPL

- Gerichteter Multihypergraph (wow)
- Forciert Schemata
- Automatisches Mapping von Java-Beans
- Mächtige Indizierungsmöglichkeiten
- Unterstützt Traversierung und Mengenbasierte Anfragen

DEX

- Entwicklerin: Sparsity Technologies
- Embedded (Java, C++, .Net); Kommerziell

- Unglaublich schnell.
- Unterstützt Atomare Anfragen, Traversierung und einige Mitgelieferte Graphalgorithmen
- Master-Slave-Replikation
- Schlechte Transaktionskontrolle
- Forciert Schema

Neo4j

- Entwicklerin: Neo Technologies
- Embedded (Java) & REST;
  GPL/AGPL/Kommerziell

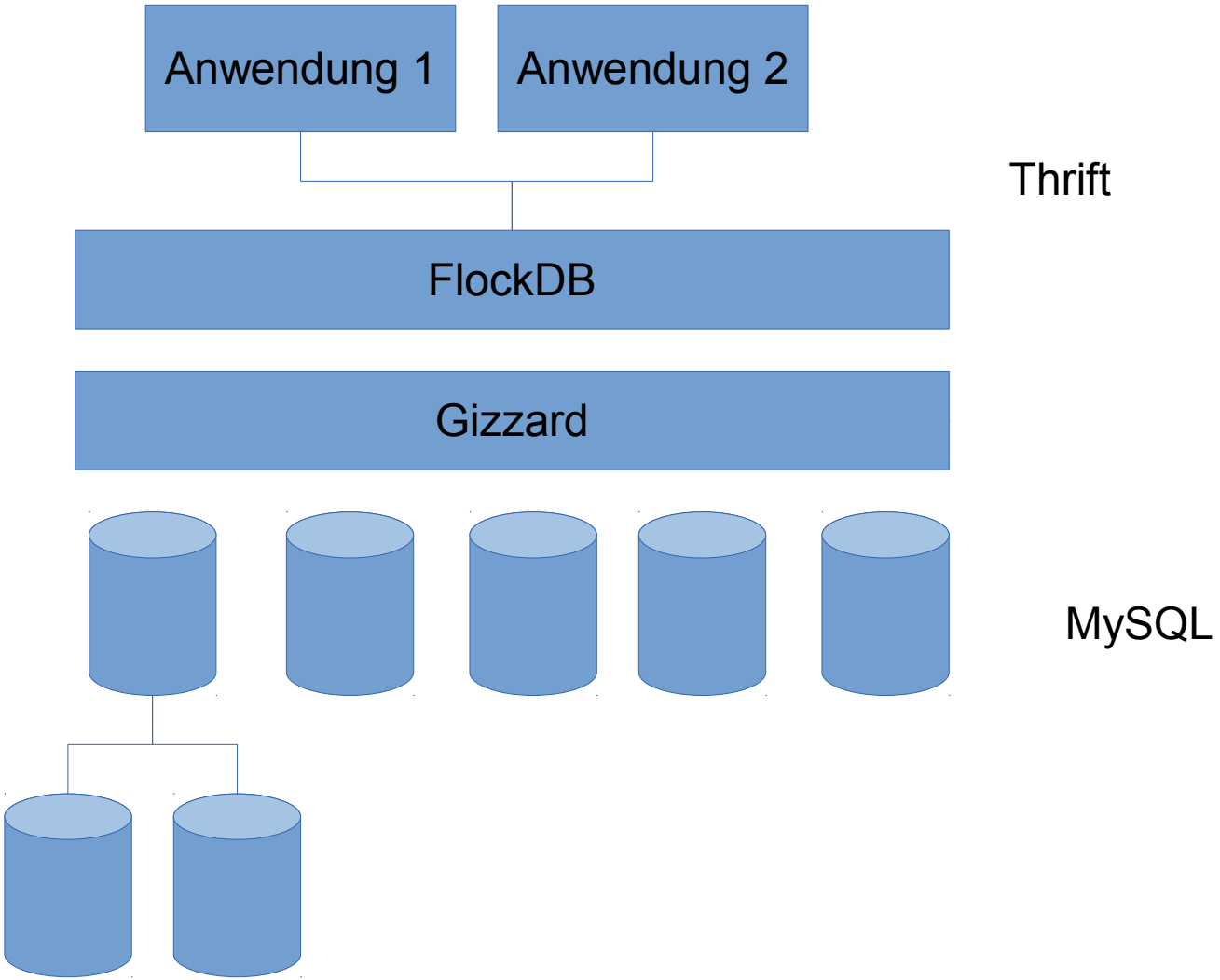
- Traversierende Anfragen, Cypher (!)
- Master-Slave-Replikation
- Leider kaum Anfrageoptimierung bei Cypher
- Schemalos, ACID-Transaktionen

FlockDB

- Entwicklerin: Twitter Inc.
- Thrift-API; Apache License 2.0

- Skaliert Horizontal bzgl. Schreiblast (Sharding)
- Sehr mageres Datenmodell (gelabelte Graphen)
- Basiert auf MySQL

FlockDB



FlockDB

- Sharding anhand von
  - Kantenlabel
  - Knoten-ID ranges
- Eingehende und ausgehende Kanten stets im selben Shard
  - Redundant

Anfragemechanismen

Mengenbasierte Anfragen

```
List<PagedNodeIdList> result = myFlockDBConnection
.select(
  difference(
    intersect(
      simpleSelection(personAId, FOLLOWS, INCOMING),
      simpleSelection(personBId, FOLLOWS, INCOMING)
    ),
    simpleSelection(personCId, BLOCKS, OUTGOING)
  )
)
.execute();
```

Graphen & Big Data

Pregel, GraphLab und Apache Giraph

- Verlagerung des Algorithmus in die Knoten

I. (Neu-) Berechnung des Knotenwertes
II. Voting: Berechnung beenden?
III. Messaging: Nachrichten an Nachbarknoten
IV. Zurück zu I.

- Dezentrale Berechnung direkt auf den Verteilten Daten
- Flüsterpost-Algorithmen
- Giraph basiert auf Hadoop, bestehende Infrastruktur nutzbar

Graphen & Big Data

PageRank

```
public void compute(Iterable<DoubleWritable> msgIterator) {
    if (getSuperstep() >= 1) {
        double sum = 0;
        for ( DoubleWritable msg : msgIterator ) {
            sum += msg.get();
        }
        DoubleWritable vertexValue = new DoubleWritable( (0.15f/getTotalNumVertices()) + 0.85f*sum );
        setValue(vertexValue);
    }

    if (getSuperstep() < NUM_ITERATIONS) {
        long edges = getNumEdges();
        sendMessageToAllEdges(new DoubleWritable(getValue().get() / edges));
    } else {
        voteToHalt();
    }
}
```

Framework-Unterstützung

Spring Data Neo4j

```
@NodeEntity
public class Movie {
   @GraphId Long id;

   @Indexed(type = FULLTEXT, indexName = "search")
   String title;

   Person director;

   @RelatedTo(type="ACTS_IN", direction = INCOMING)
   Set<Person> actors;

   @Query("start movie=node({self}) match
           movie-->genre<--similar return similar")
   Iterable<Movie> similarMovies;
}
```

Spring Data Neo4j

```
interface MovieRepository extends GraphRepository<Movie> {
  @Query("start movie={0} match m<-[rating:RATED]-user
        return rating")
  Iterable<Rating> getRatings(Movie movie);
}
```

```
@Autowired MovieRepository repo;

Iterable<Movie> movies = repo.findAll();
Movie movie = repo.findByPropertyValue("title","Matrix");
repo.save(movie);

Iterable<Rating> ratings = repo.getRatings(movie);
```

Tinkerpop Blueprints

```
Graph graph = new Neo4jGraph("/tmp/my_graph");
Vertex a = graph.addVertex(null);
Vertex b = graph.addVertex(null);
a.setProperty("name","marko");
b.setProperty("name","peter");
Edge e = graph.addEdge(null, a, b, "knows");
e.setProperty("since", 2006);
graph.shutdown();
```

Tinkerpop Blueprints

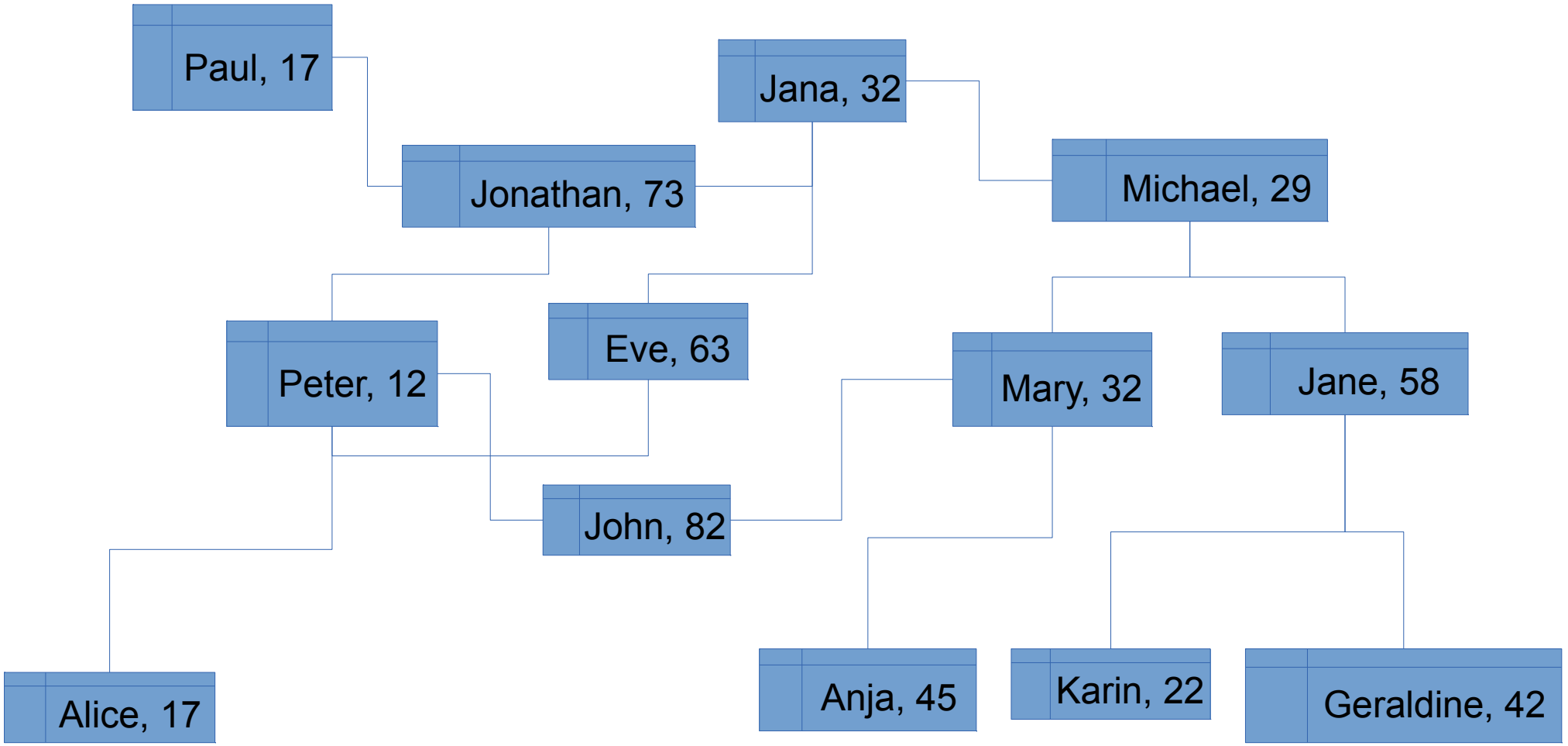
- Besteht aus mehreren Komponenten:
  - Blueprints: Einheitliche low-level API
  - Frames: OO-Mapping
  - Gremlin: Anfragesprache
  - Furnace: Sammlung von Graphalgorithmen
  - Rexter: REST-Server

Tinkerpop Blueprints Frames

```
public interface Person {
  @Property("name")
  public String getName();
  @Adjacency(label="knows")
  public Iterable<Person> getKnowsPeople();
  @Adjacency(label="knows")
  public void addKnowsPerson(final Person person);
  @GremlinGroovy("it.out('knows').out('knows').dedup")
  public Iterable<Person> getFriendsOfAFriend()
}

public class Frames {
  public Frames() {
   FramedGraph<NeoGraph> graph = new FramedGraph<NeoGraph>(new NeoGraph("local:/path/to/db"));
   Person person = graph.getVertex(1, Person.class);
   person.getName(); // equals "marko"
  }
}
```

Eine Letzte Aufgabe



Danke!

Benjamin Gehrels
benjamin@gehrels.info
GitHub: @BGehrels