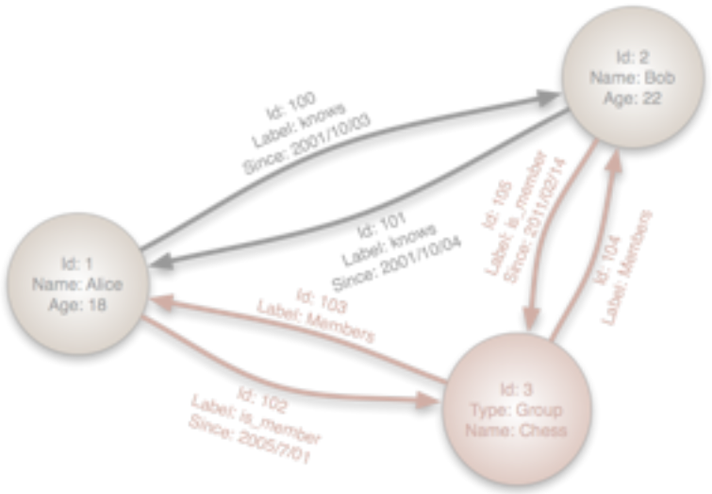
Intro to Graph Databases with Neo4j

University of Lugano
May 2016

Michael Hunger
michael@neo4j.com
@mesirii
@neo4j

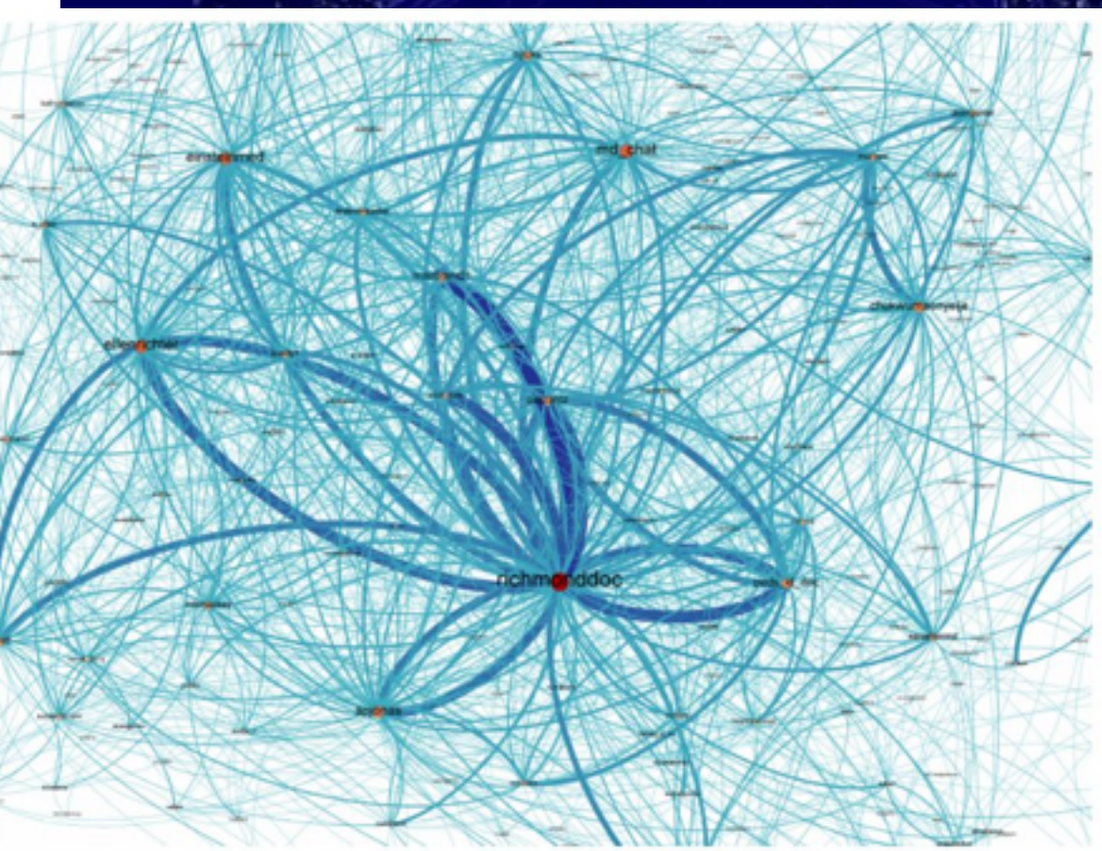


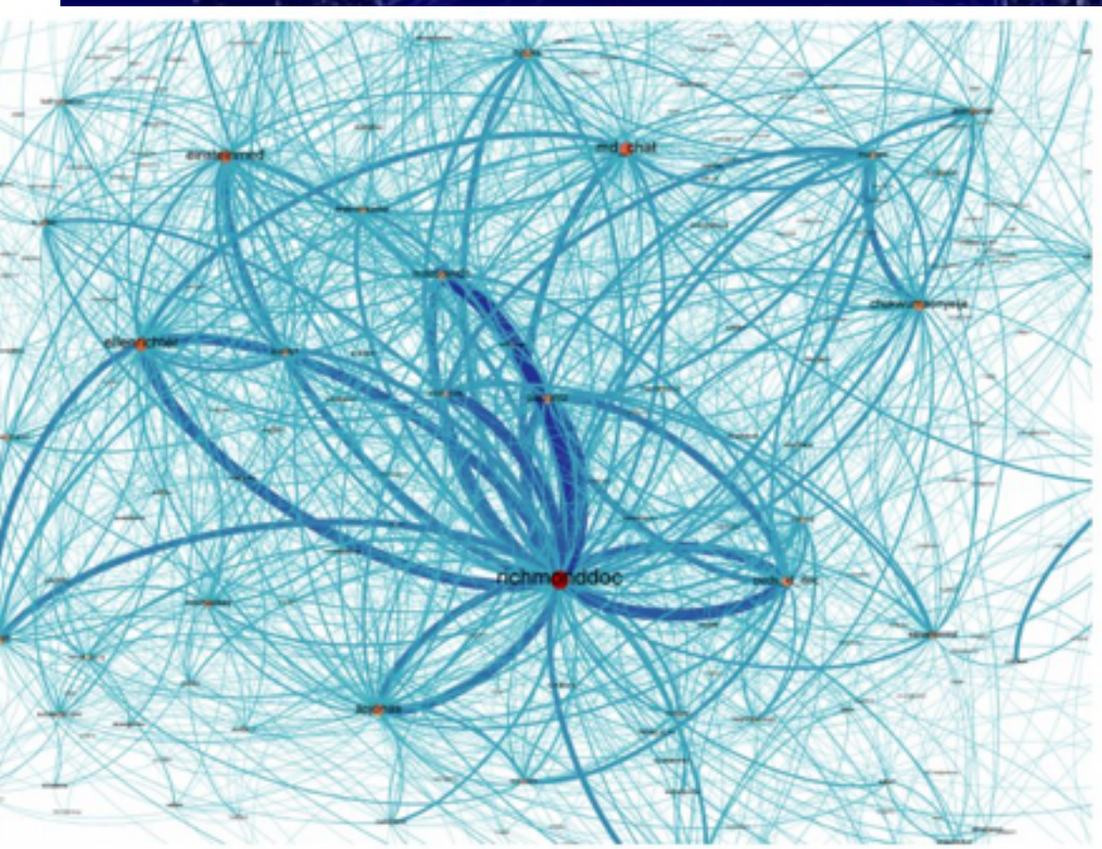
Agenda

1. Why Graphs, Why Now?
2. What Is A Graph, Anyway?
3. Neo4j as a Graph Database
4. Graph Querying
    1. Cypher
    2. Examples

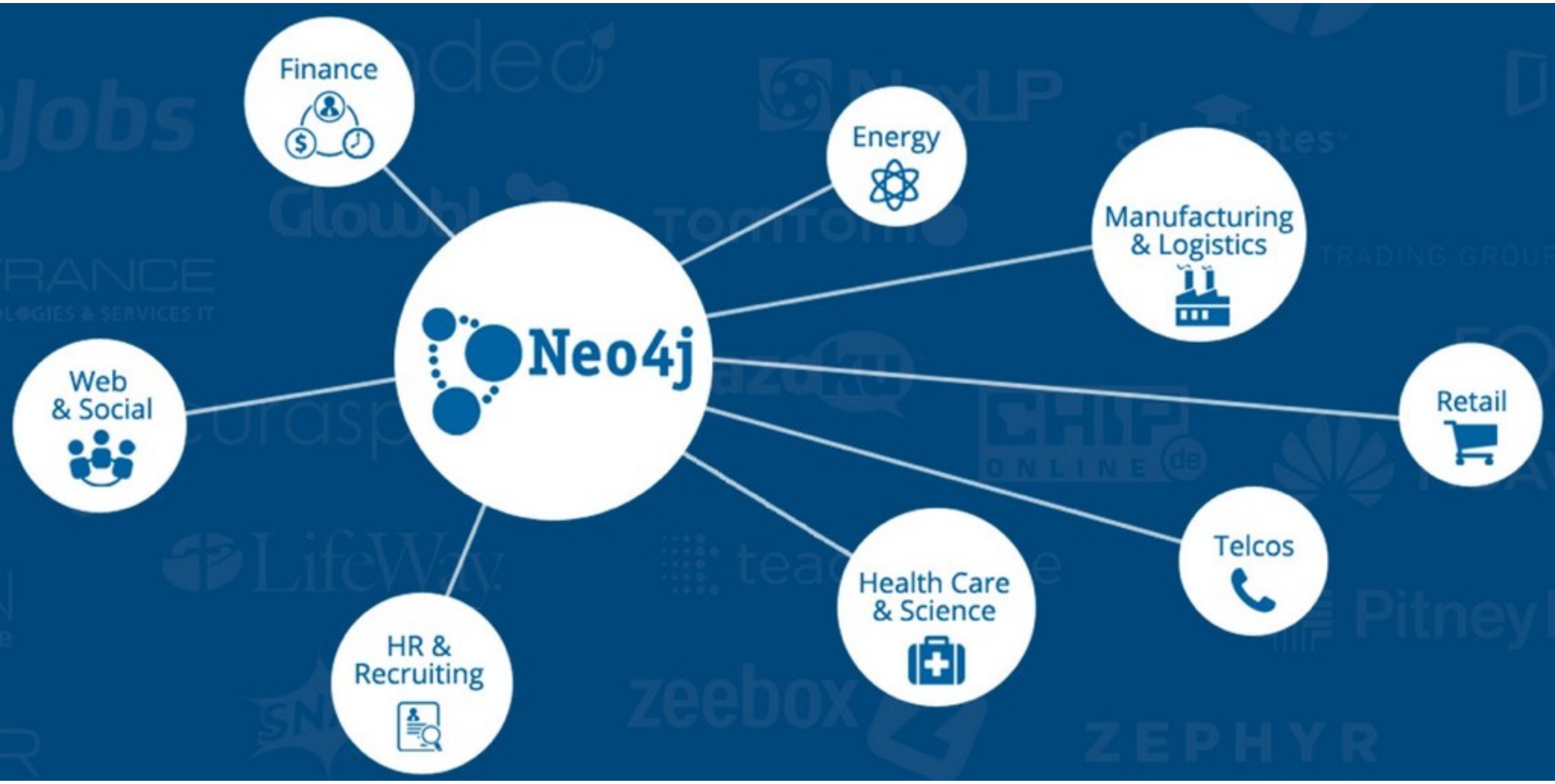
Why Graphs?

The World is a Graph



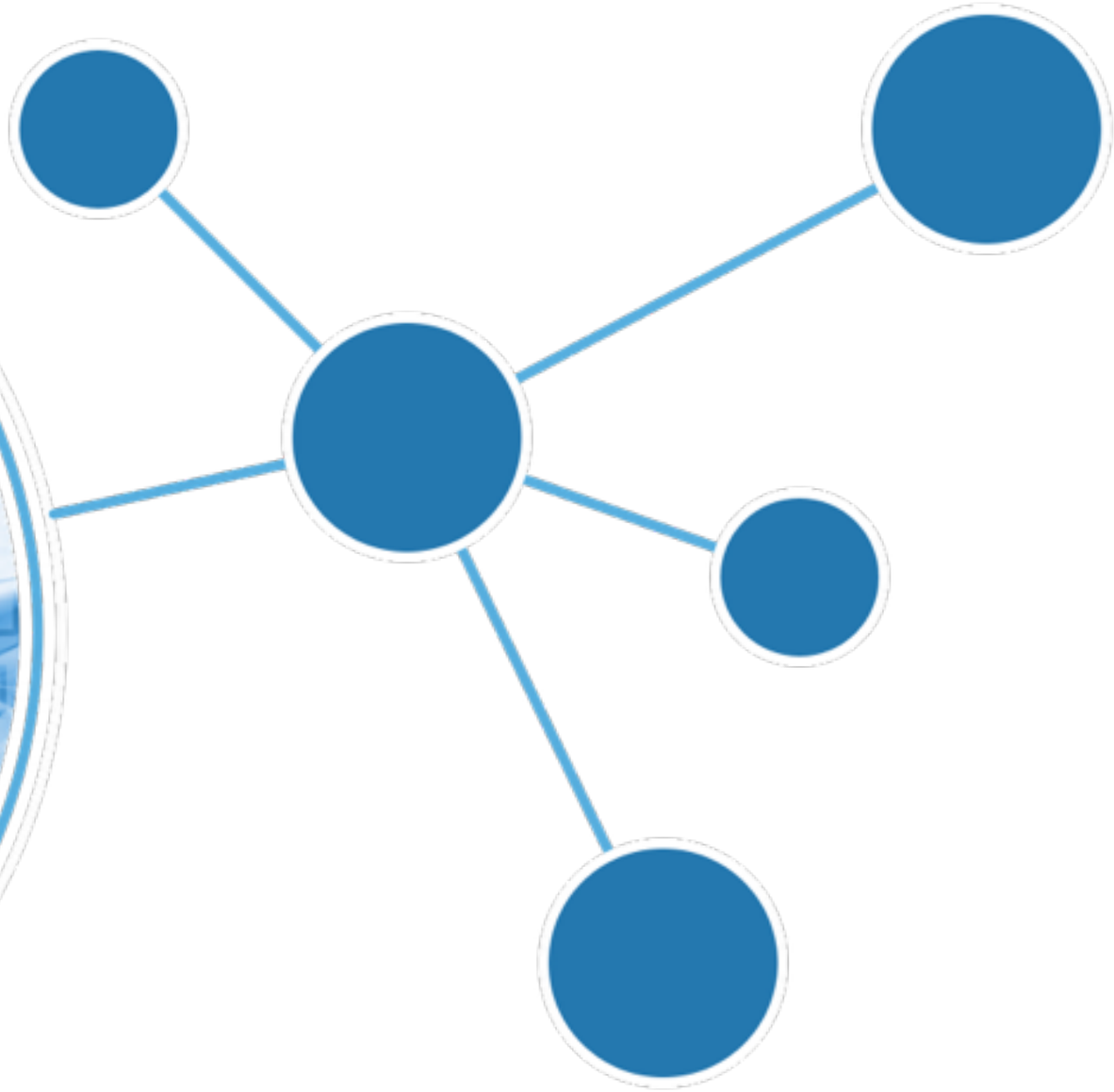


Some Use-Cases



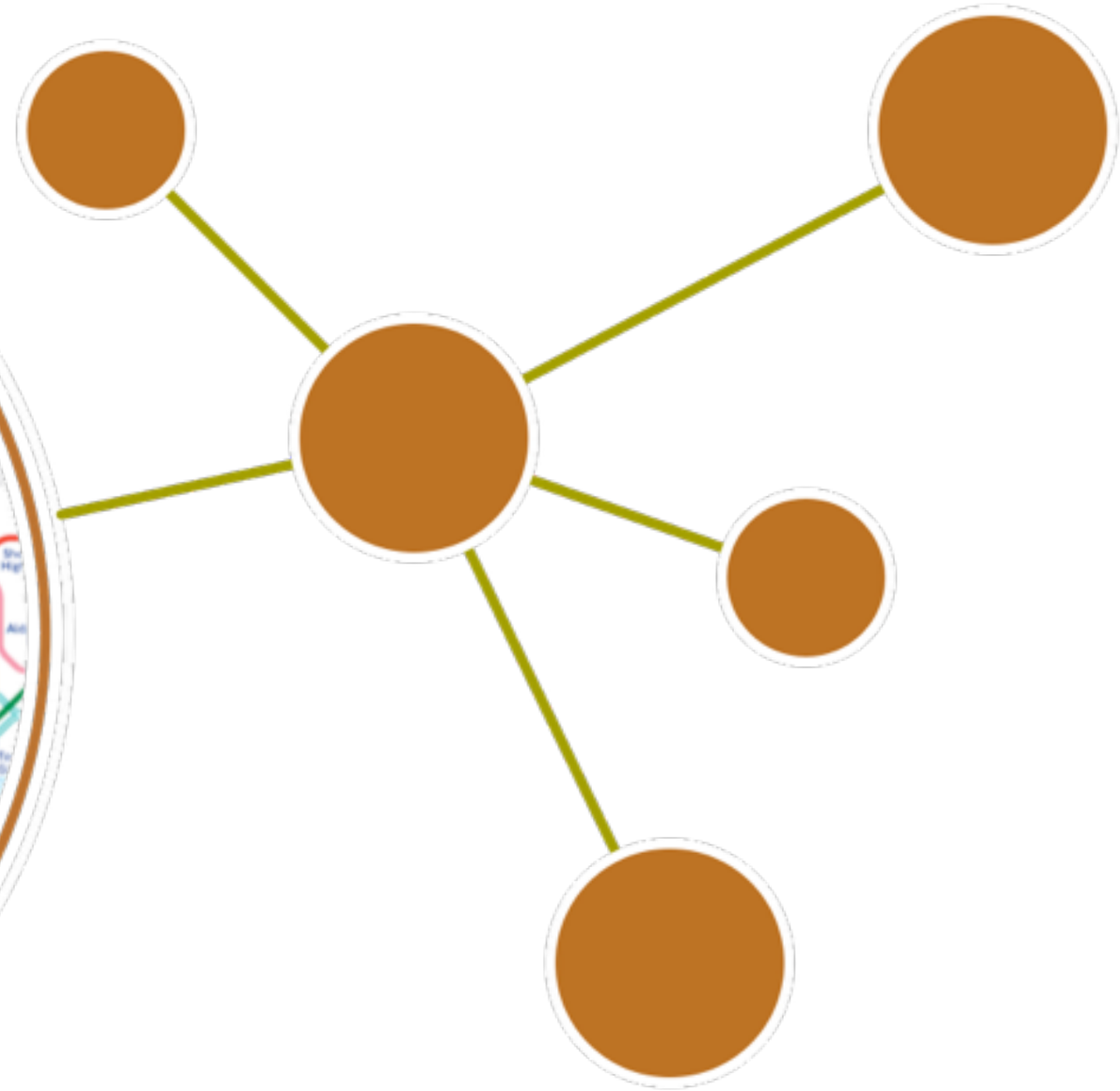
Social Network

(Network) Impact Analysis

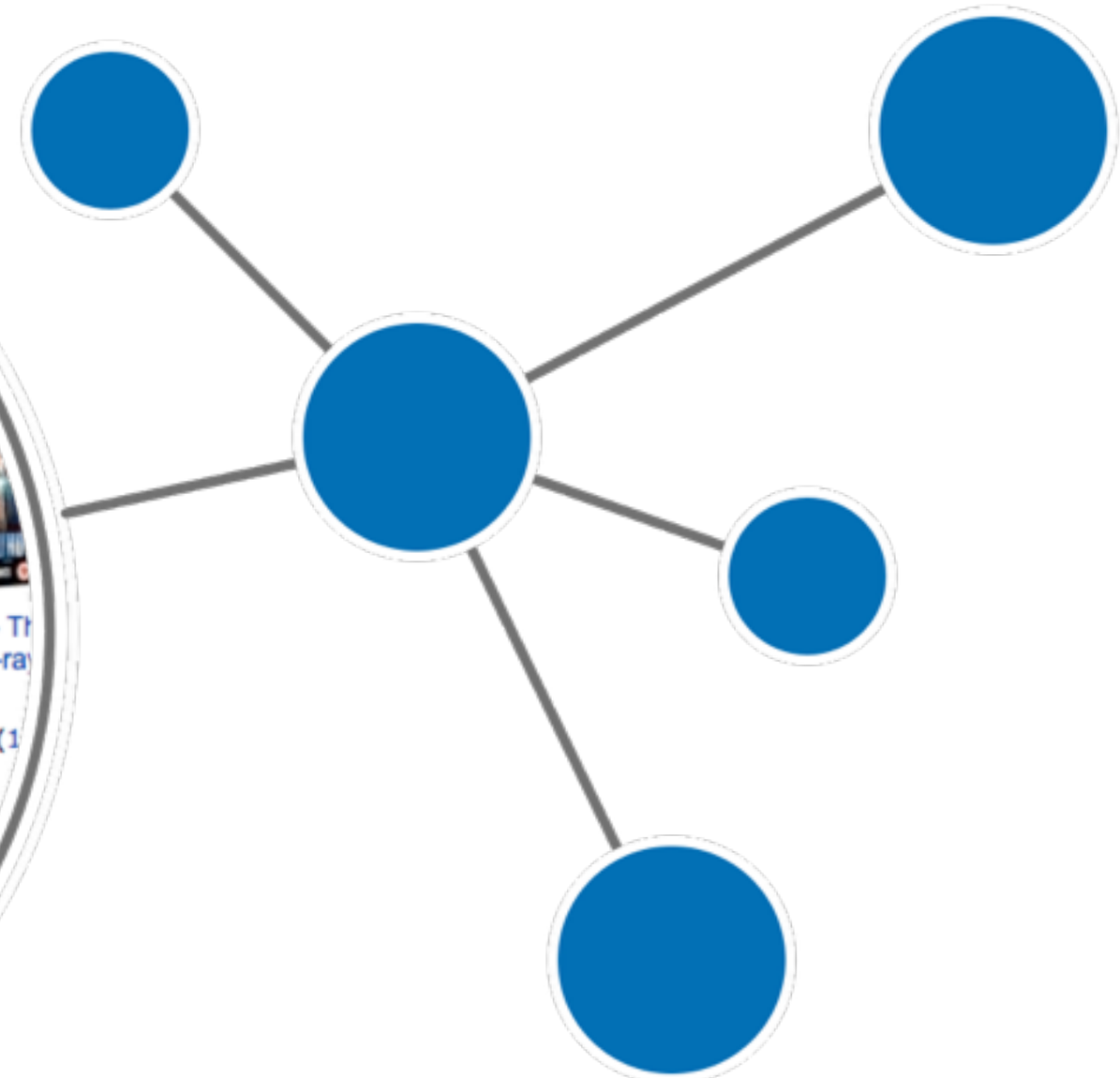


Logistics & Routing

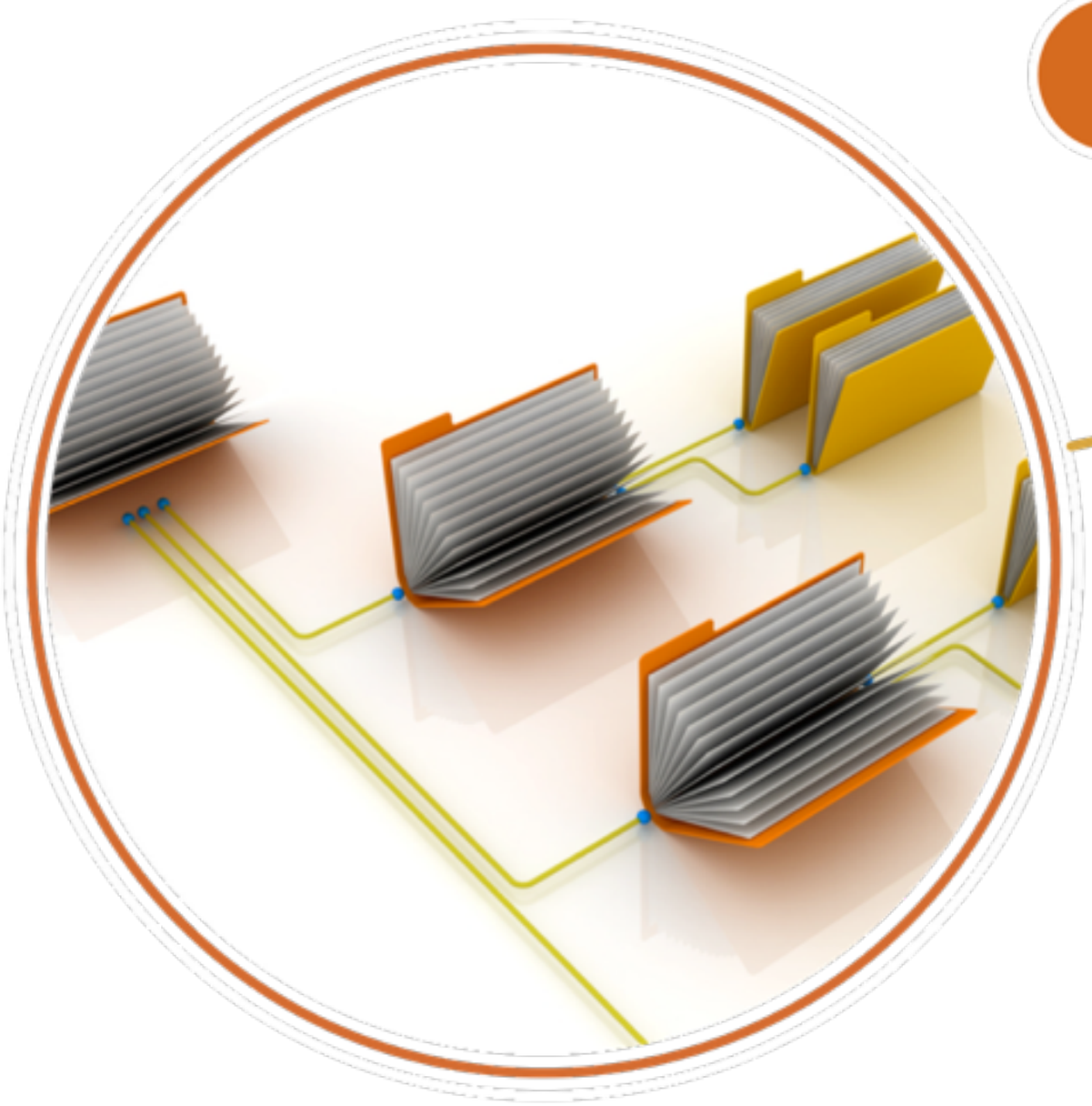
Route Finding



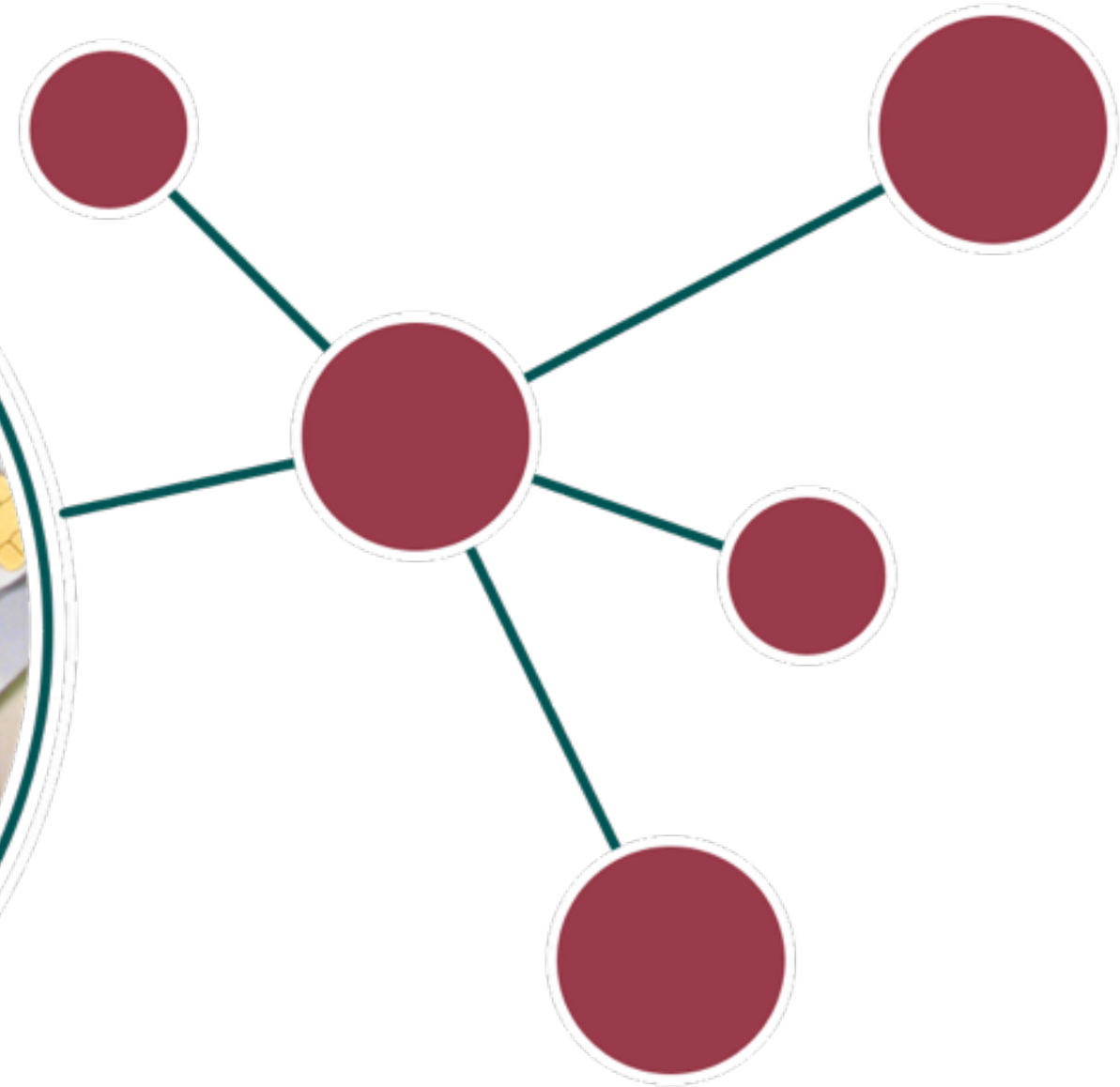
Recommendations



Access Control



Fraud Analysis



The NoSQL Space

What is NOSQL?

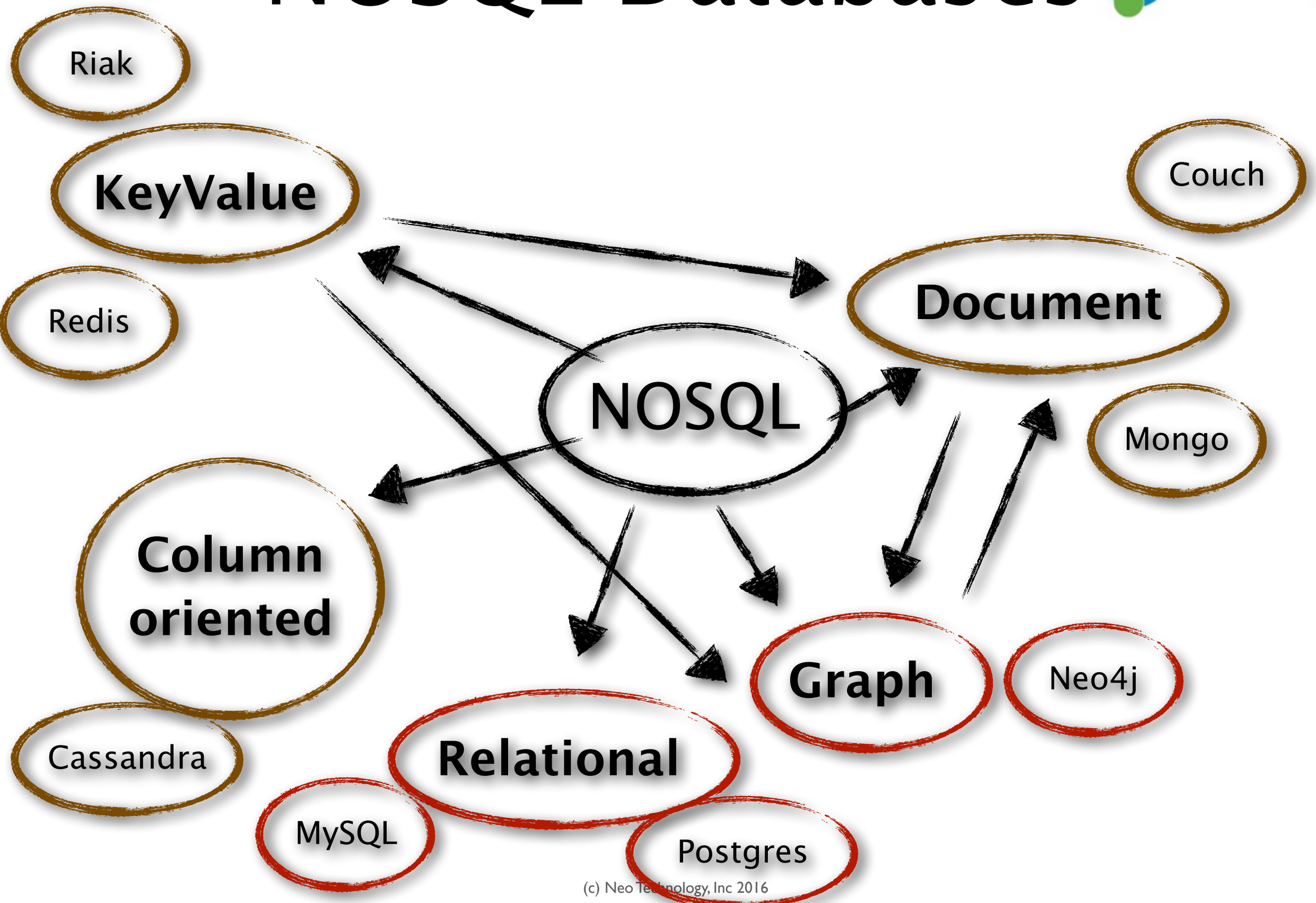
It's not "No to SQL"

It's not "Never SQL"

It's "**Not Only SQL**"

NOSQL \no-seek-wool\ *n.* Describes ongoing trend where developers increasingly opt for non-relational databases to help solve their problems, in an effort to use the right tool for the right job.

NOSQL Databases

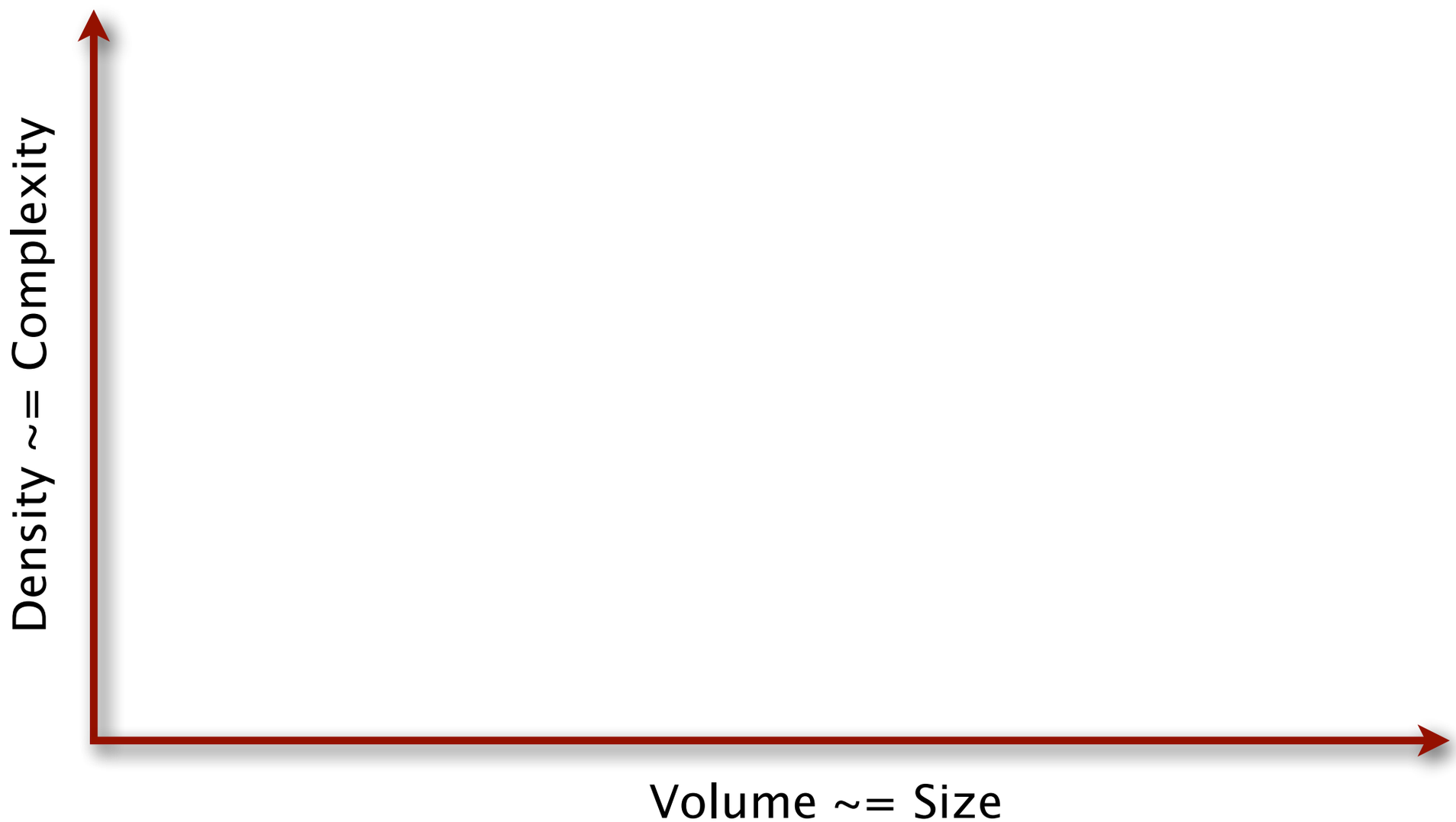


Living in a NOSQL World

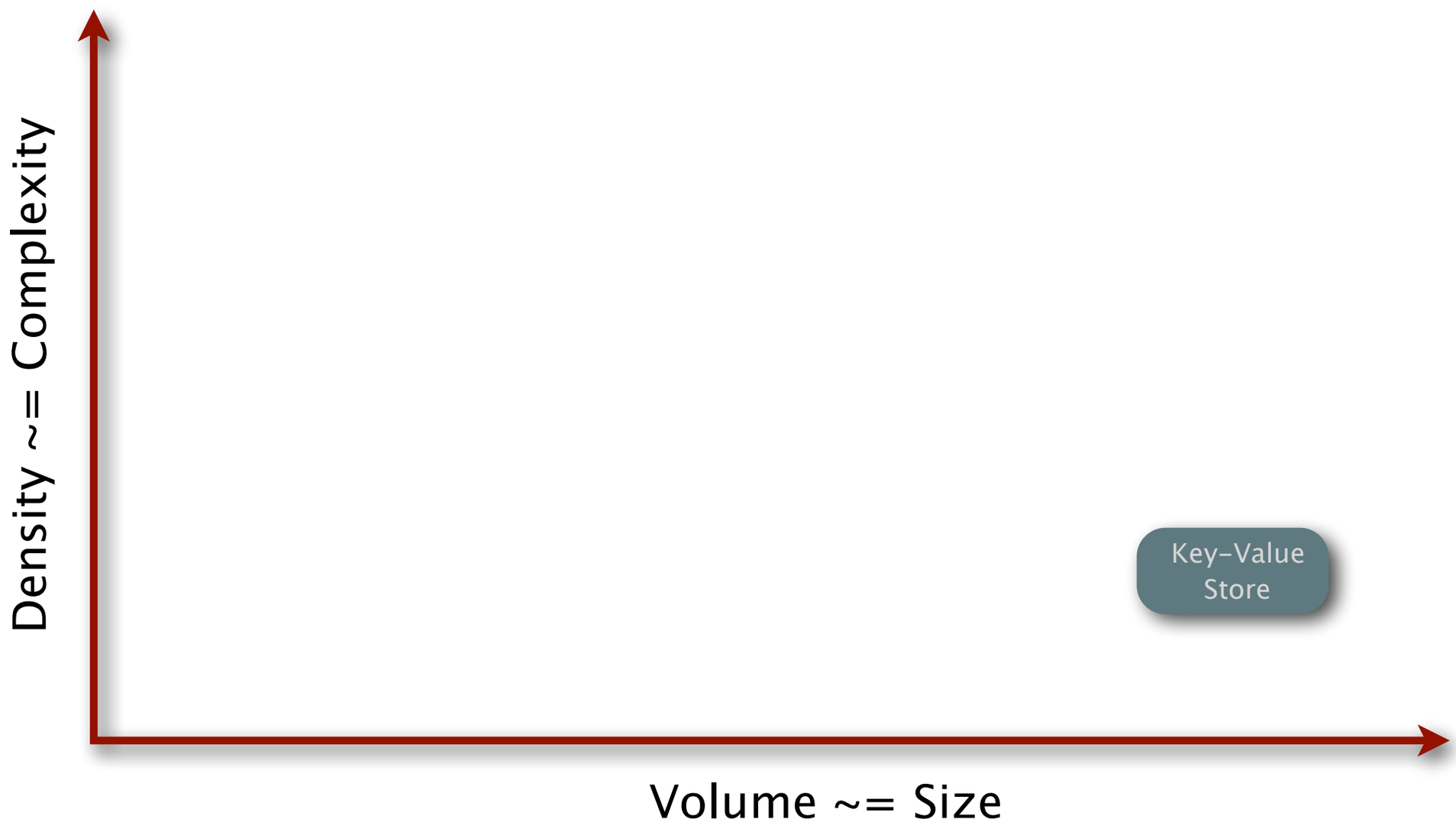
Living in a NOSQL World

Volume ~= Size

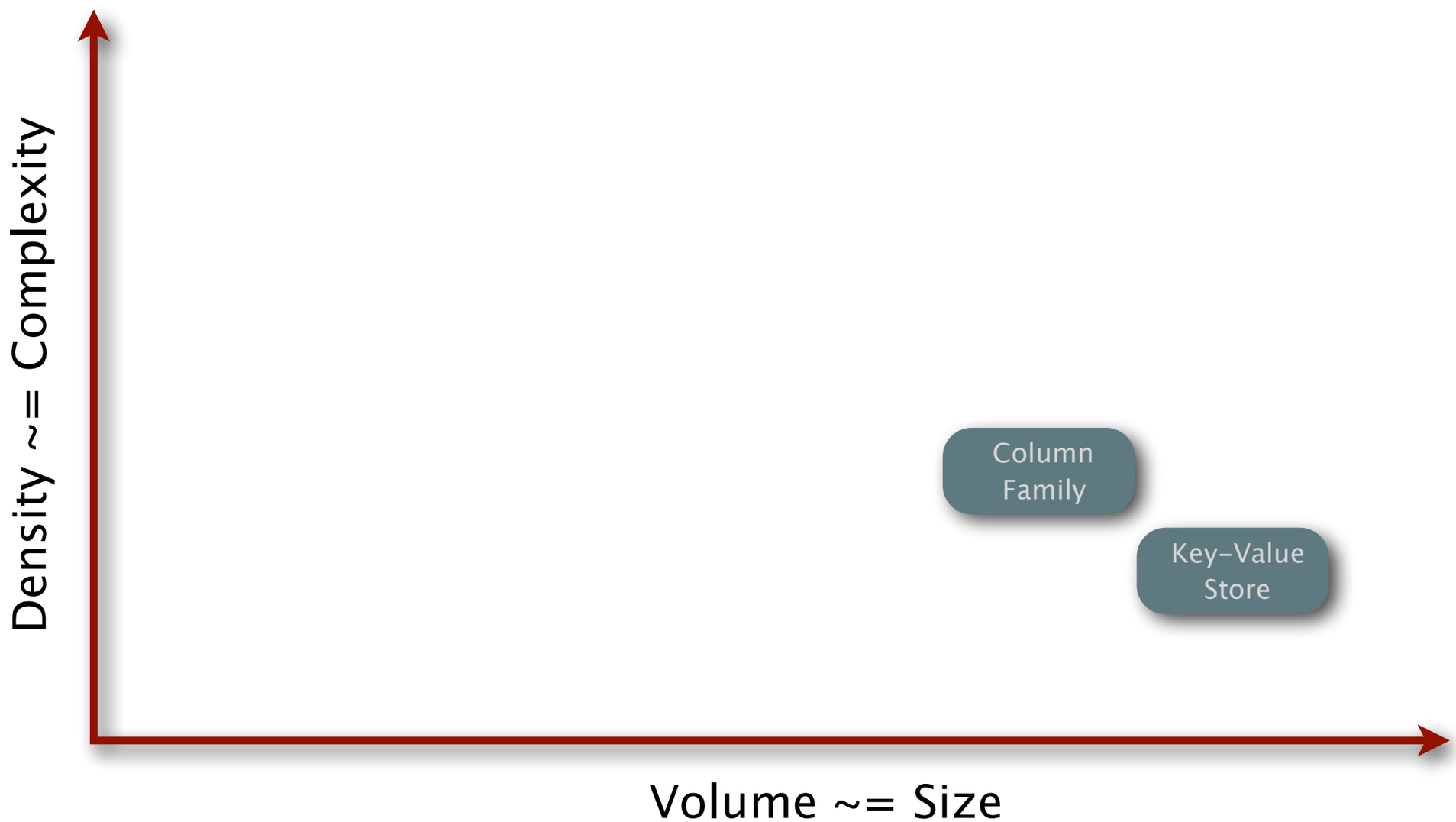
Living in a NOSQL World



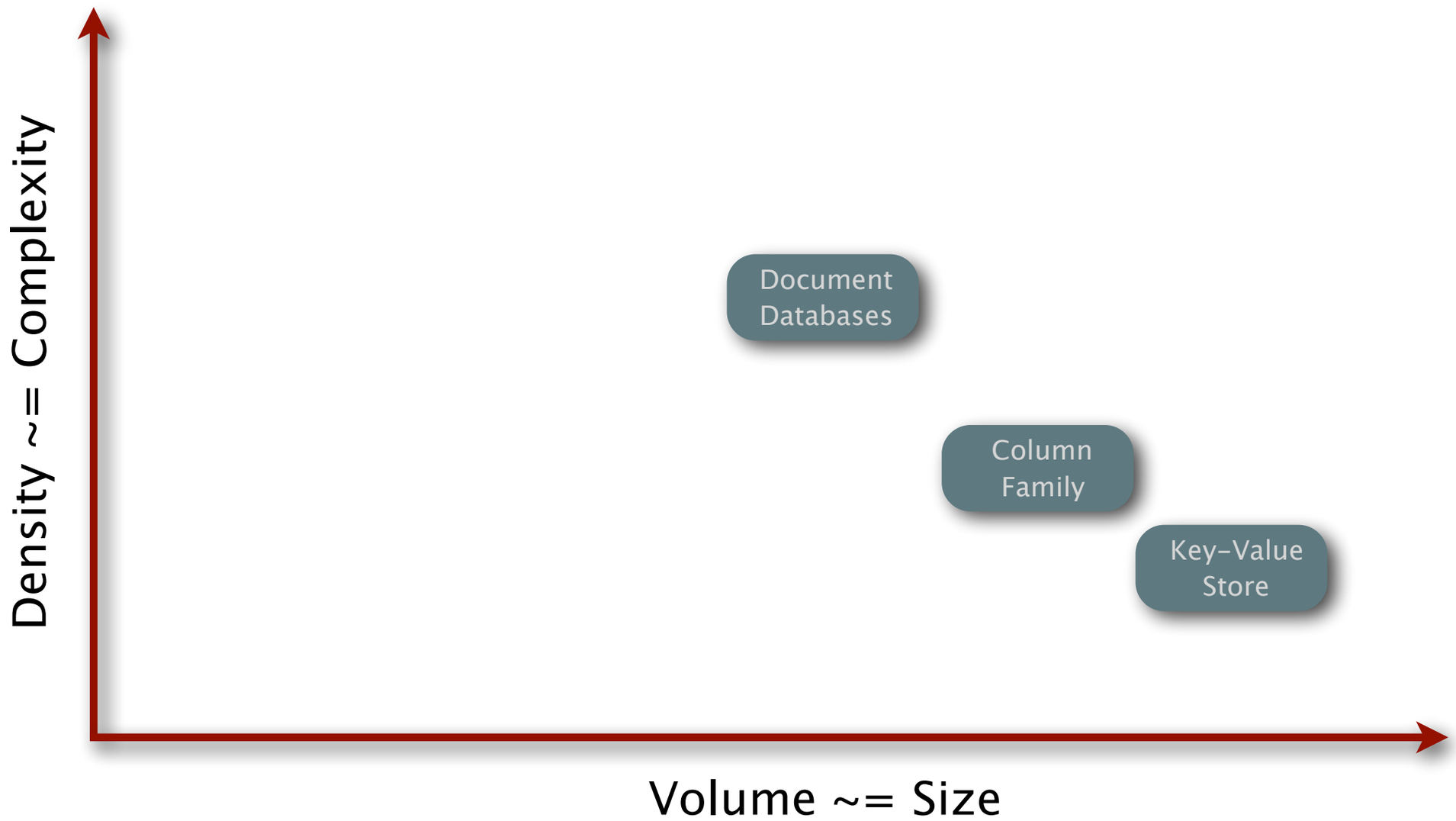
Living in a NOSQL World



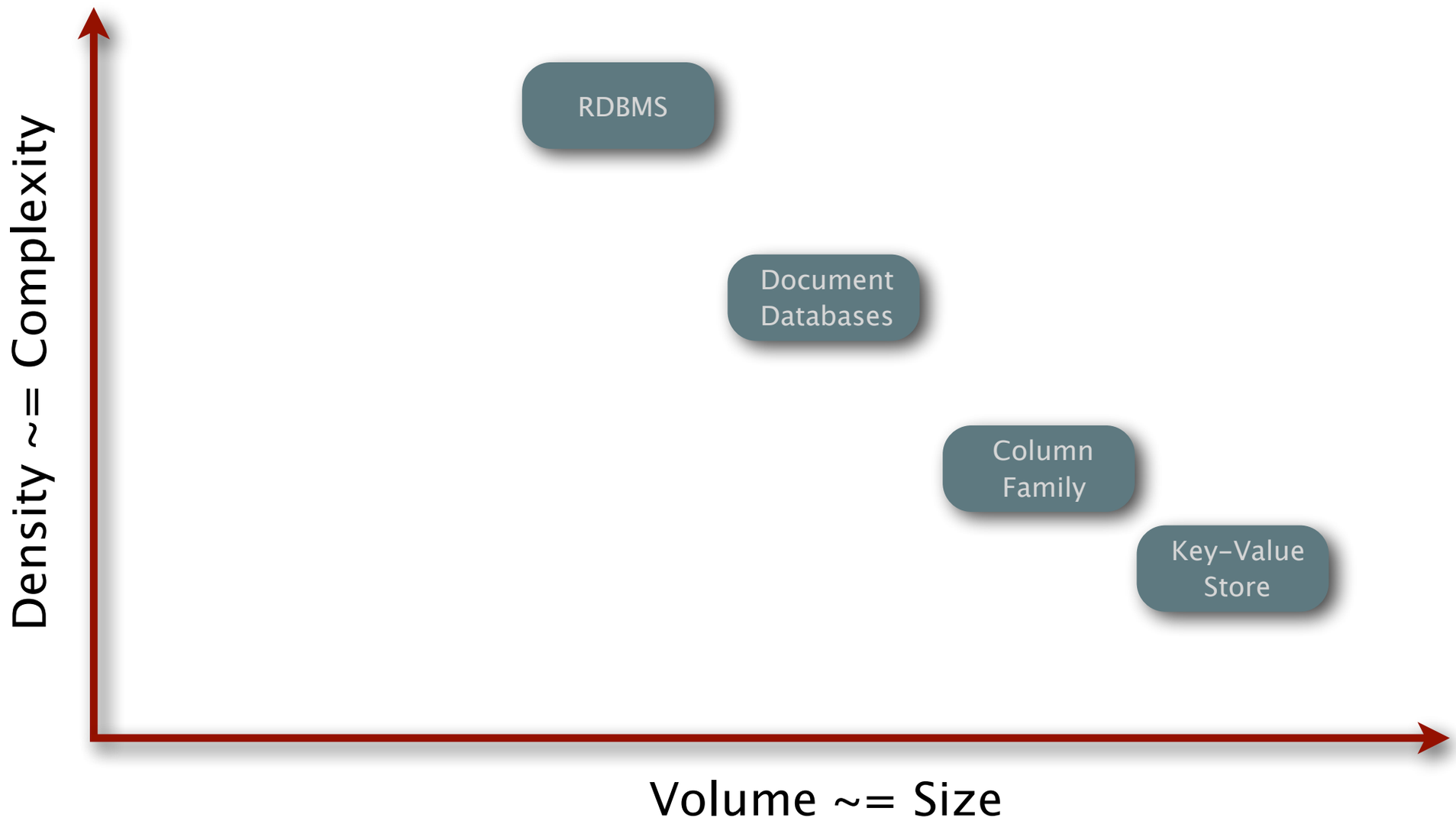
Living in a NOSQL World



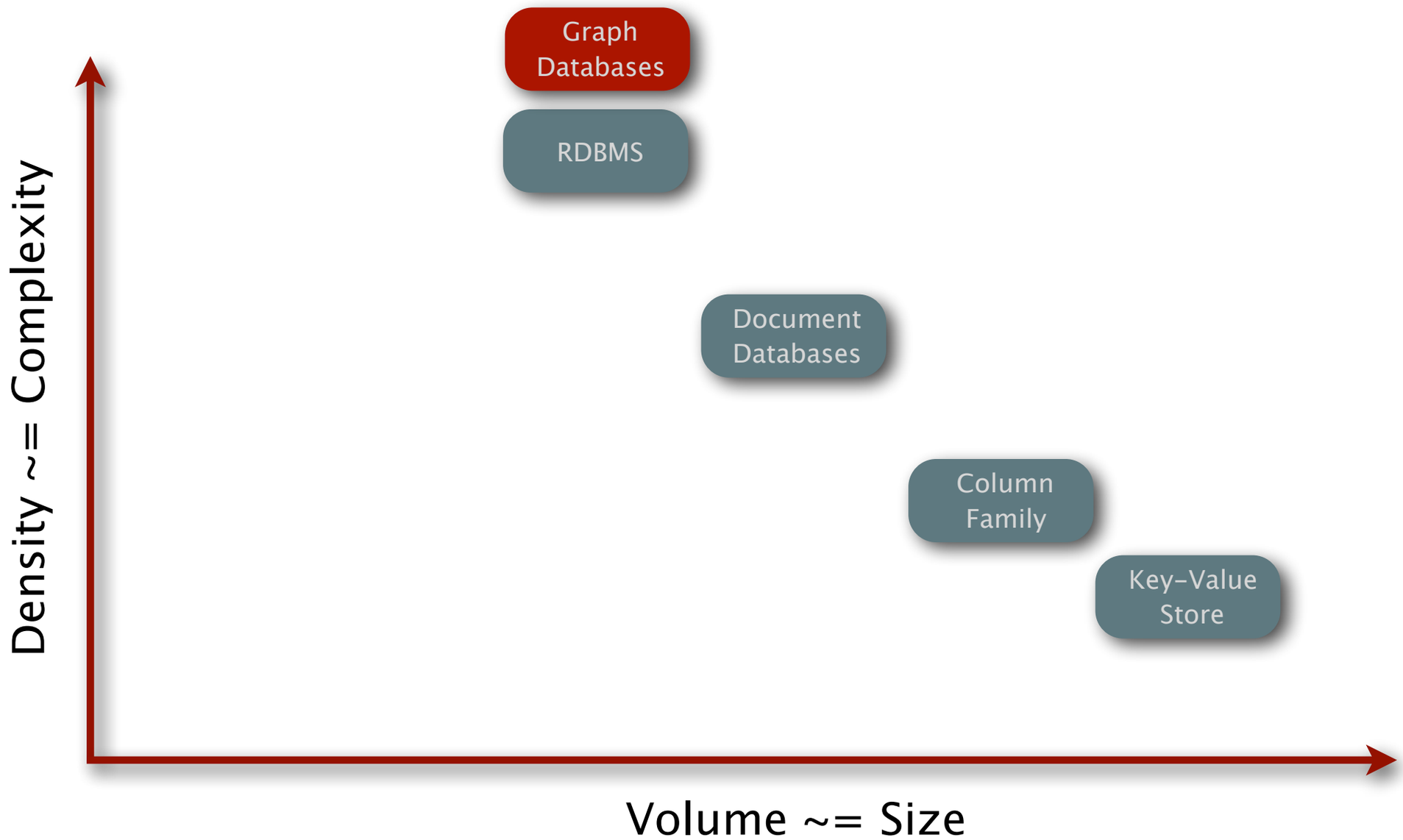
Living in a NOSQL World



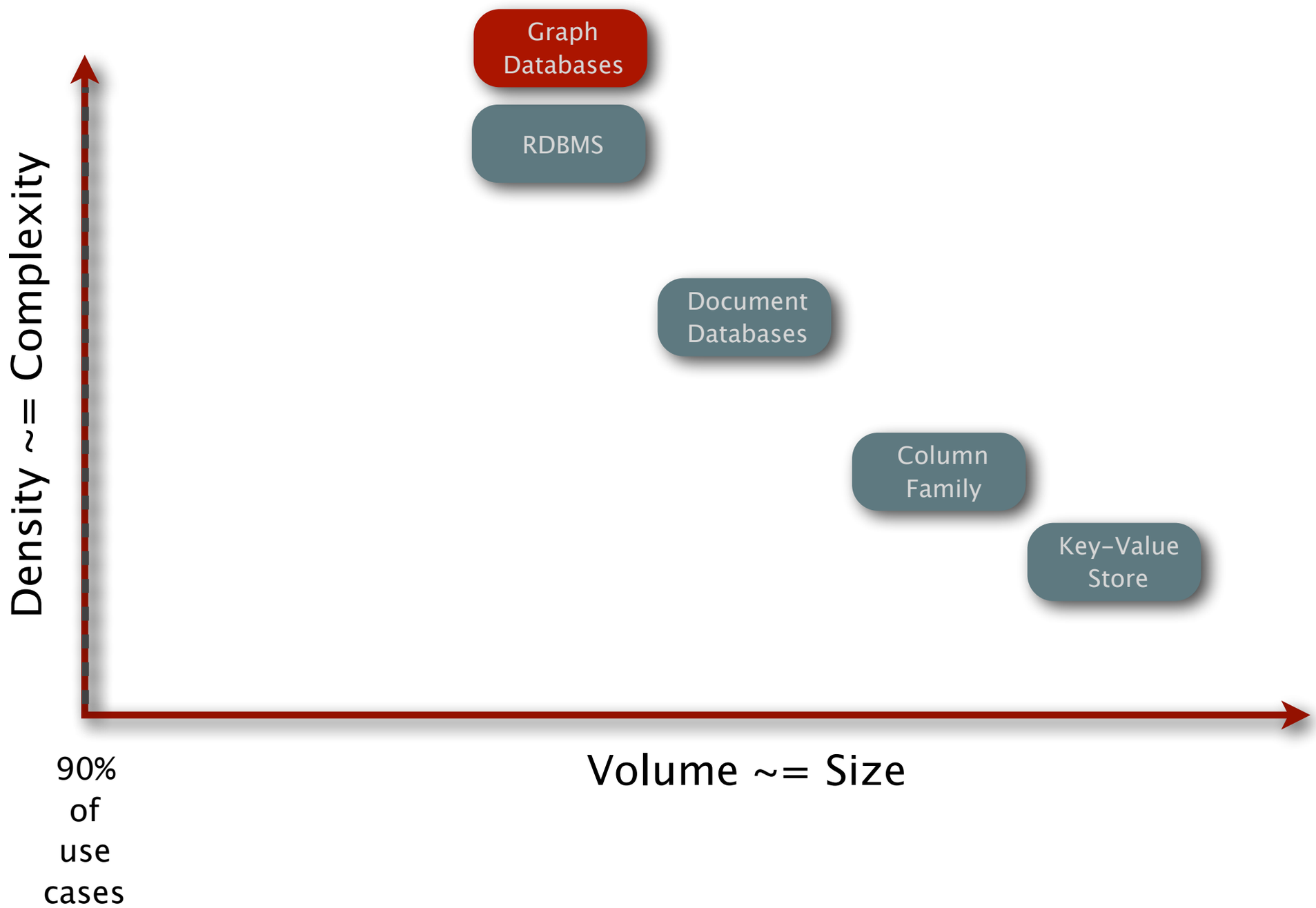
Living in a NOSQL World



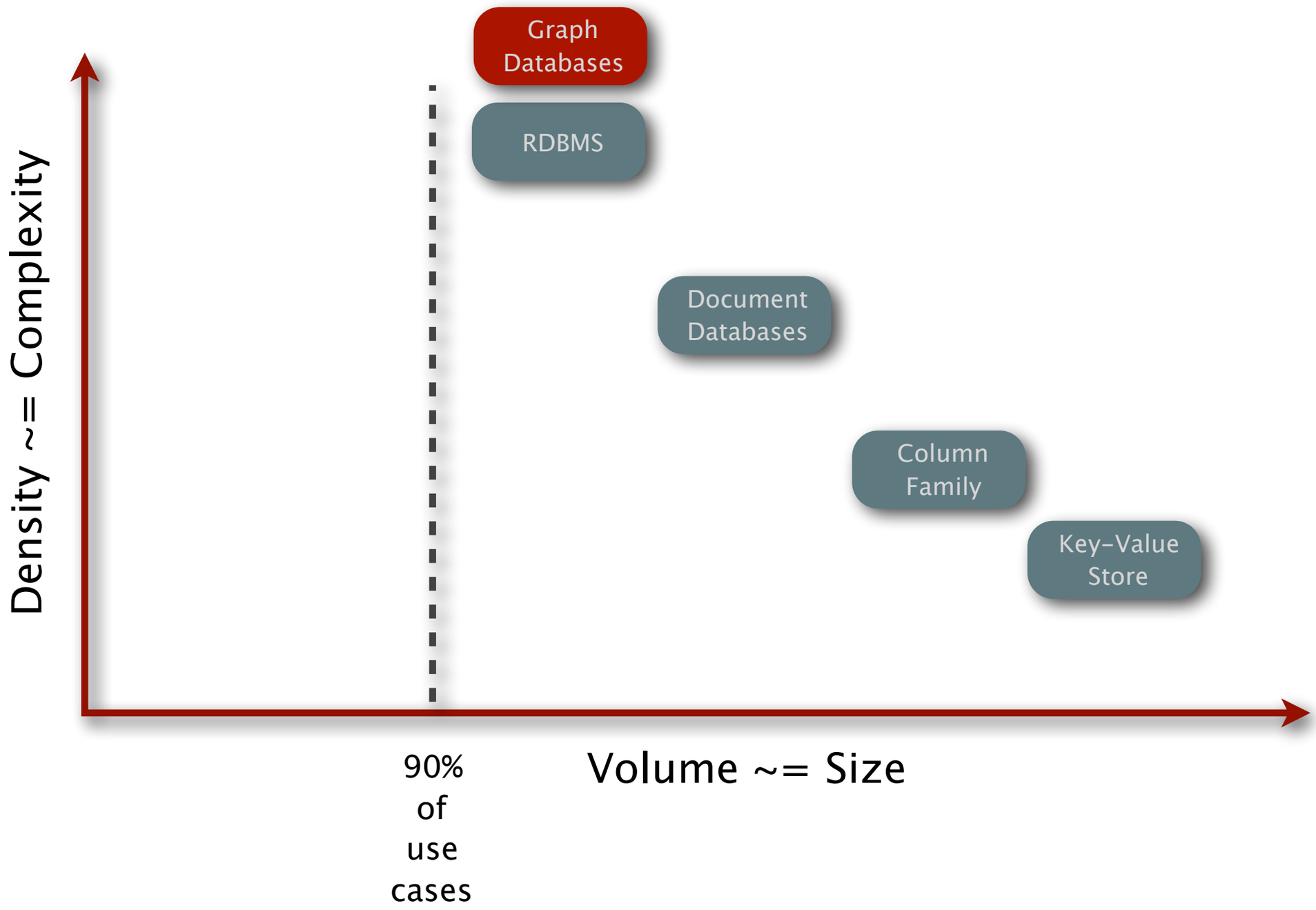
Living in a NOSQL World



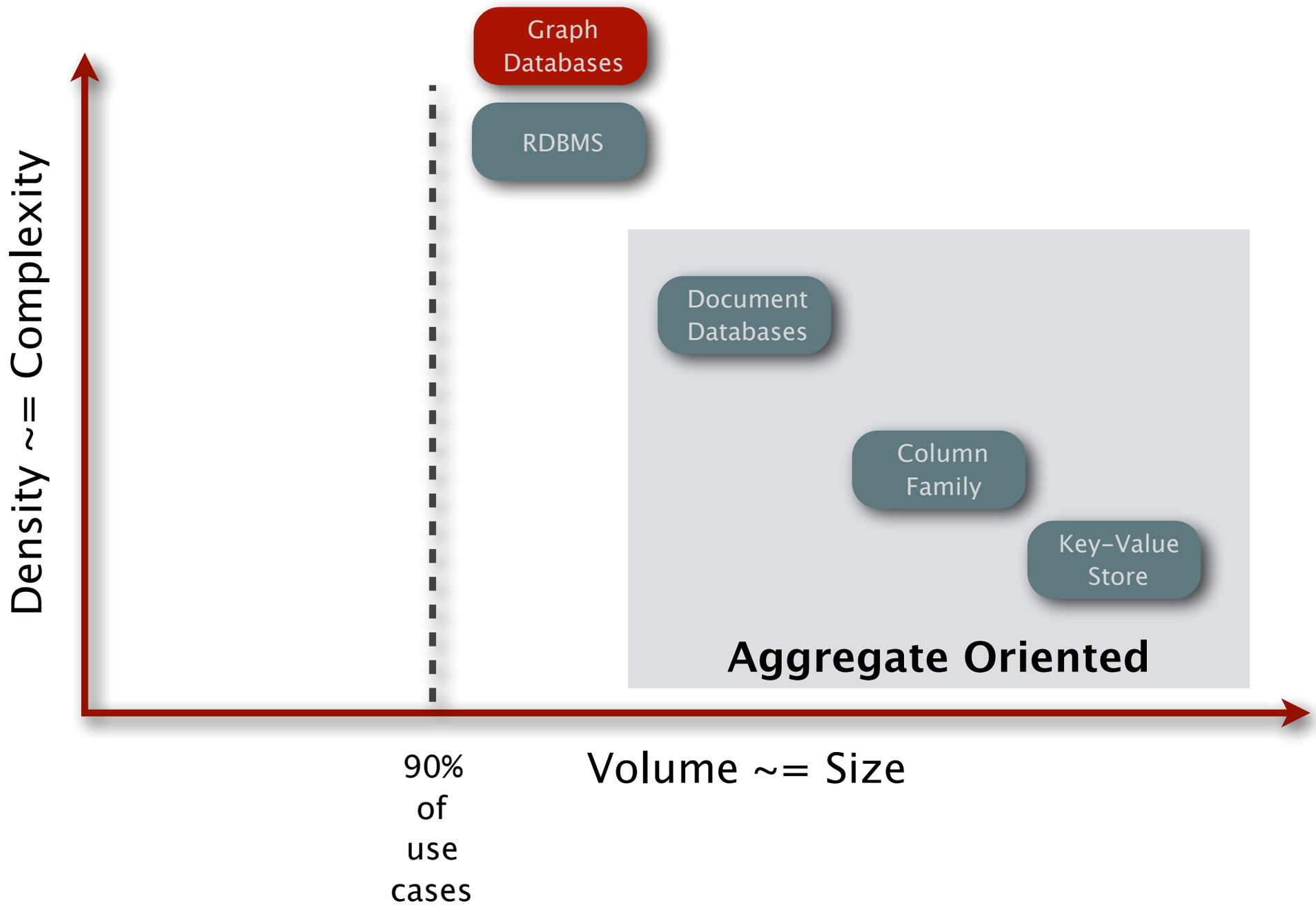
Living in a NOSQL World



Living in a NOSQL World



Living in a NOSQL World



Aggregate Oriented Model

"There is a significant downside - the whole approach works really well when data access is aligned with the aggregates, but what if you want to look at the data in a different way? Order entry naturally stores orders as aggregates, but analyzing product sales cuts across the aggregate structure. The advantage of not using an aggregate structure in the database is that it allows you to slice and dice your data different ways for different audiences.

This is why aggregate-oriented stores talk so much about map-reduce."

Martin Fowler

Connected Data Model

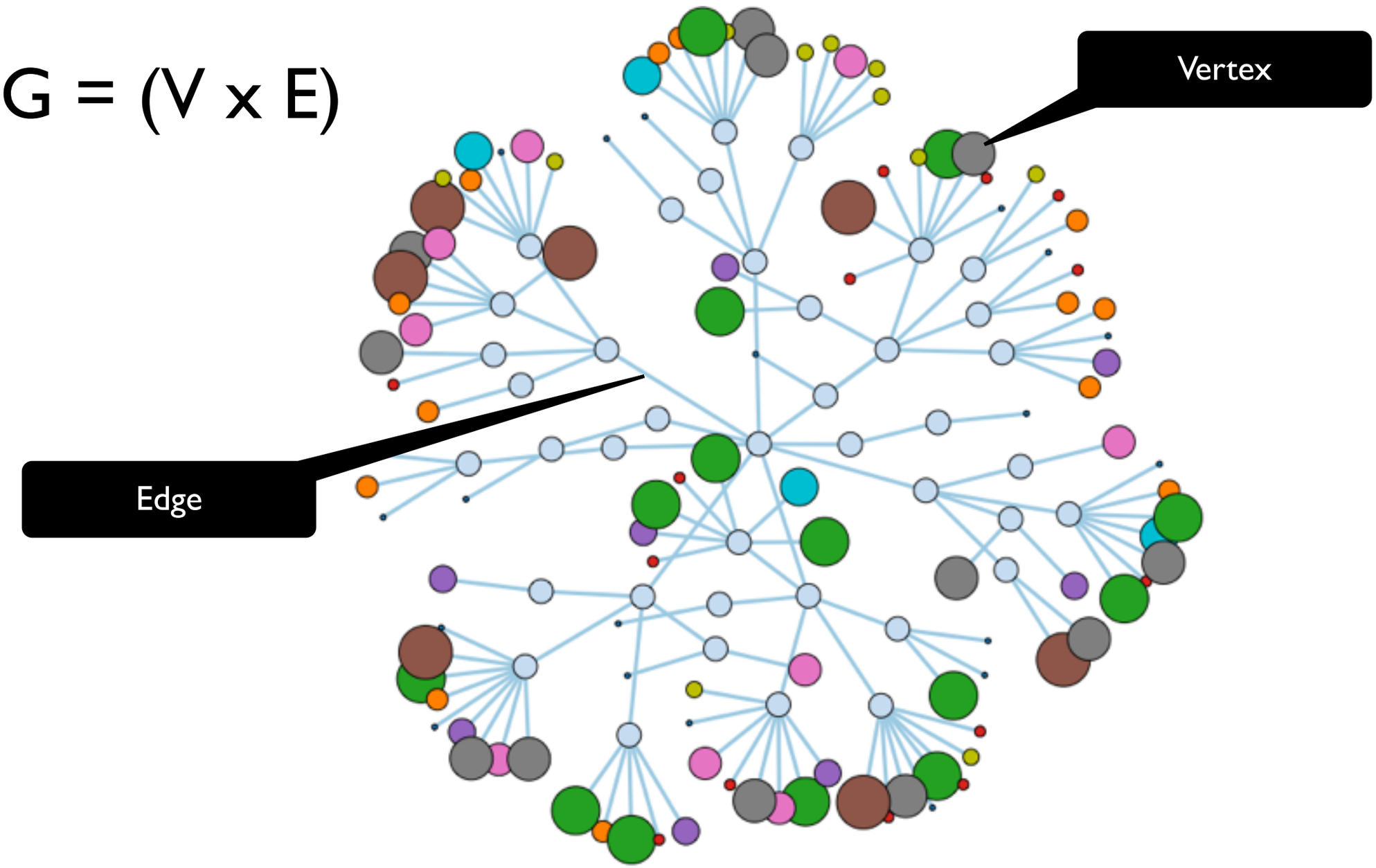
The connected data model is based on fine grained elements that are richly connected, the emphasis is on extracting many dimensions and attributes as elements.
Connections are cheap and can be used not only for the domain-level relationships but also for additional structures that allow efficient access for different use-cases. The fine grained model requires a external scope for mutating operations that ensures Atomicity, Consistency, Isolation and Durability - ACID also known as Transactions.

Michael Hunger

The Graph Landscape

A Graph

$G = (V \times E)$



Graph Ecosystem

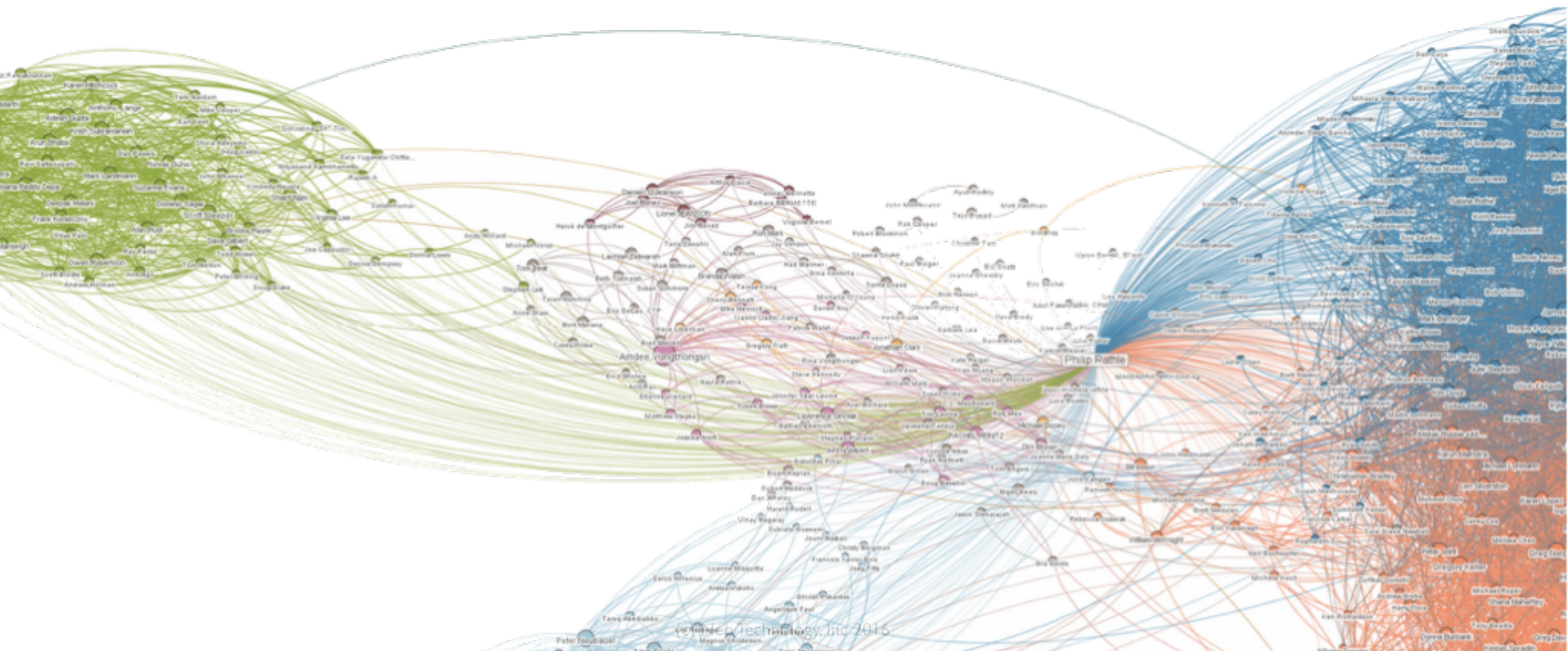
1. Graph Databases

2. Graph Compute Engines

Real-Time/ OLTP

Offline/ Batch

Connected Data



What is a Graph Database

A **graph database** is an online ("real-time") database management system with CRUD methods that expose a graph data model[1]

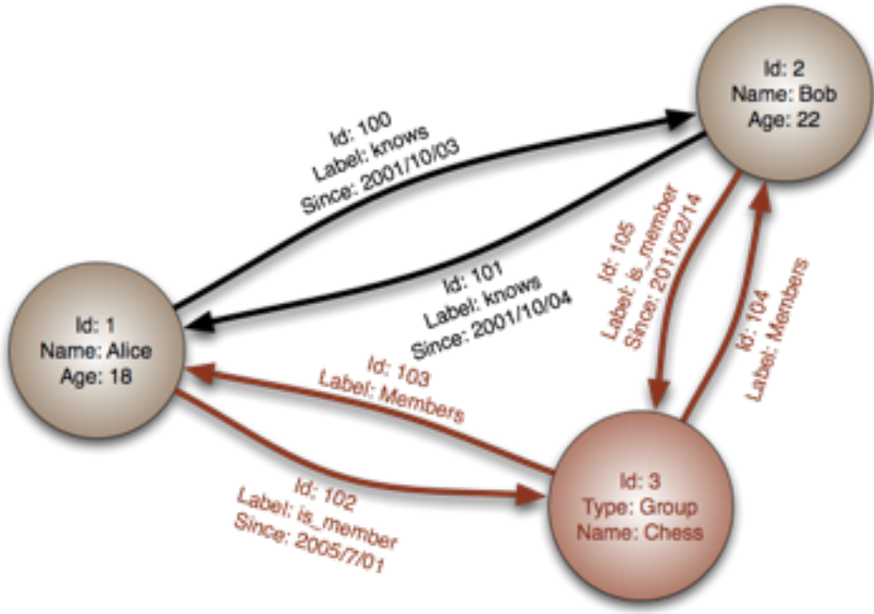
- Two important properties:
  - **Native graph processing**, including *index-free adjacency* to facilitate traversals
  - **Native graph storage** engine, i.e. written from the ground up to be optimized for managing graph data

1] Robinson, Webber, Eifrem. *Graph Databases*. O'Reilly, 2013. p. 5. ISBN-10: 1449356265

Overview of Popular Graph Data Models

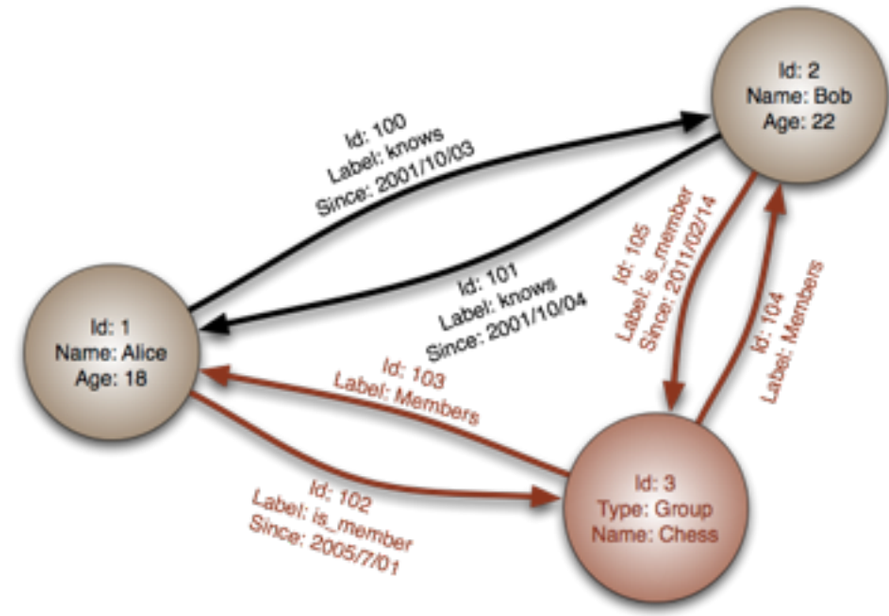
Overview of Popular Graph Data Models

- **Property Graph**
  - **Description:** A "directed, labeled, attributed, multi-graph" which exposes three building blocks: *nodes*, *typed relationships* and *key-value properties* on both nodes and relationships
  - **Vendors:** Neo4j, OrientDB, InfiniteGraph, Dex

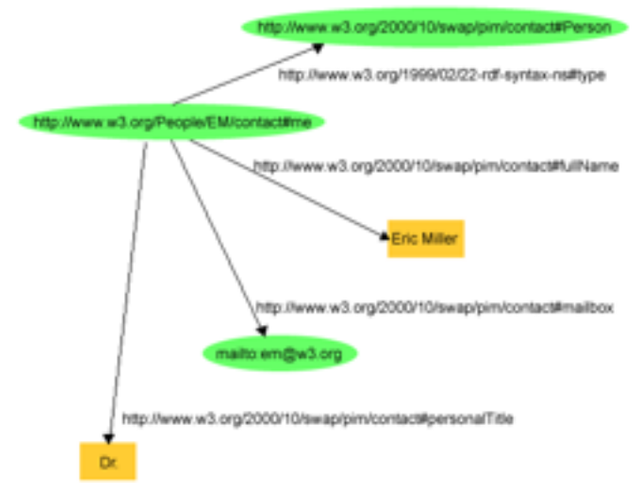


Overview of Popular Graph Data Models

- **Property Graph**
  - **Description:** A "directed, labeled, attributed, multi-graph" which exposes three building blocks: *nodes*, *typed relationships* and *key-value properties* on both nodes and relationships
  - **Vendors:** Neo4j, OrientDB, InfiniteGraph, Dex

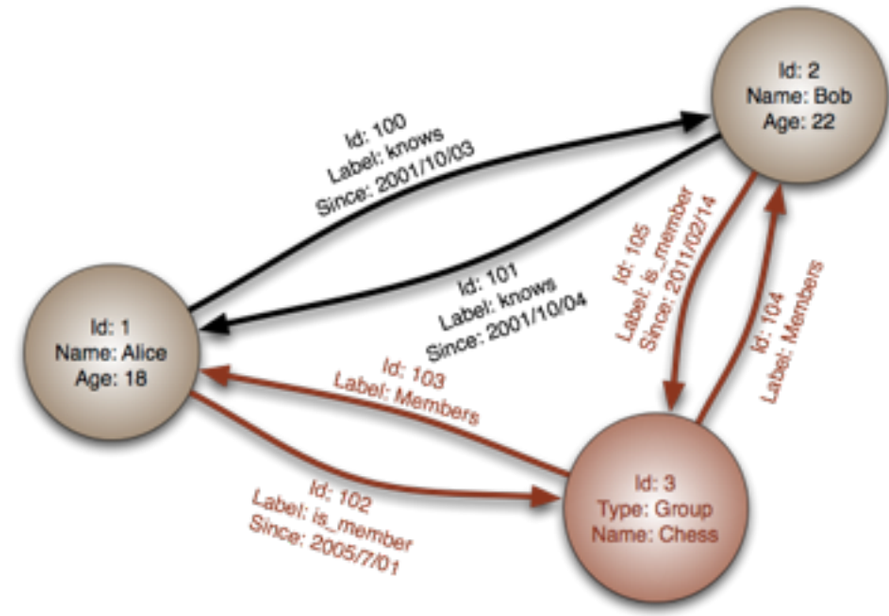


- **RDF Triples**
  - **Description:** URI-centered subject-predicate-object[2] triples as pioneered by the semantic web movement
  - **Vendors:** AllegroGraph, Sesame

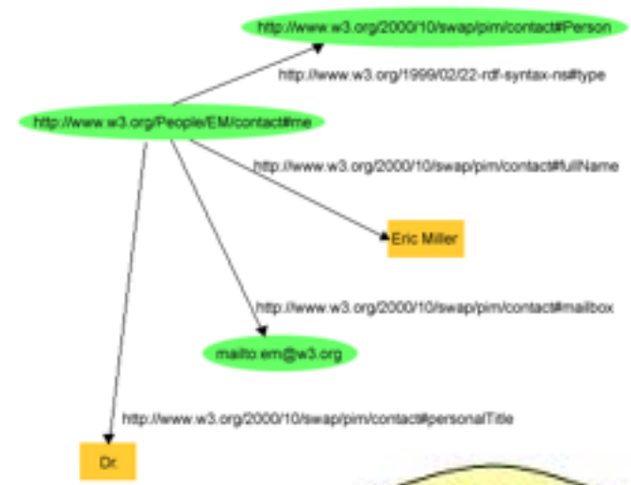


Overview of Popular Graph Data Models

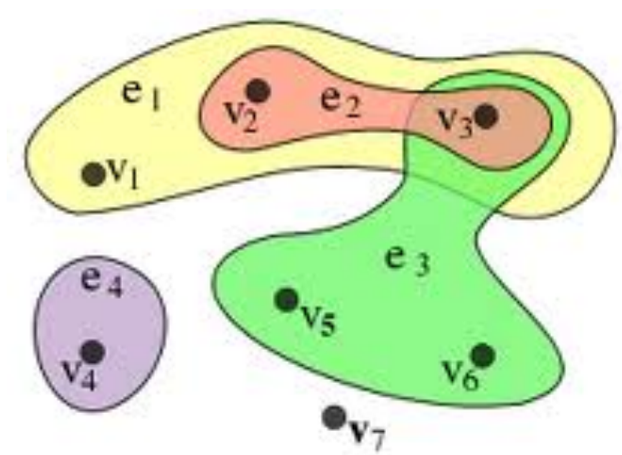
- **Property Graph**
  - **Description:** A "directed, labeled, attributed, multi-graph" which exposes three building blocks: *nodes*, *typed relationships* and *key-value properties* on both nodes and relationships
  - **Vendors:** Neo4j, OrientDB, InfiniteGraph, Dex



- **RDF Triples**
  - **Description:** URI-centered subject-predicate-object[2] triples as pioneered by the semantic web movement
  - **Vendors:** AllegroGraph, Sesame



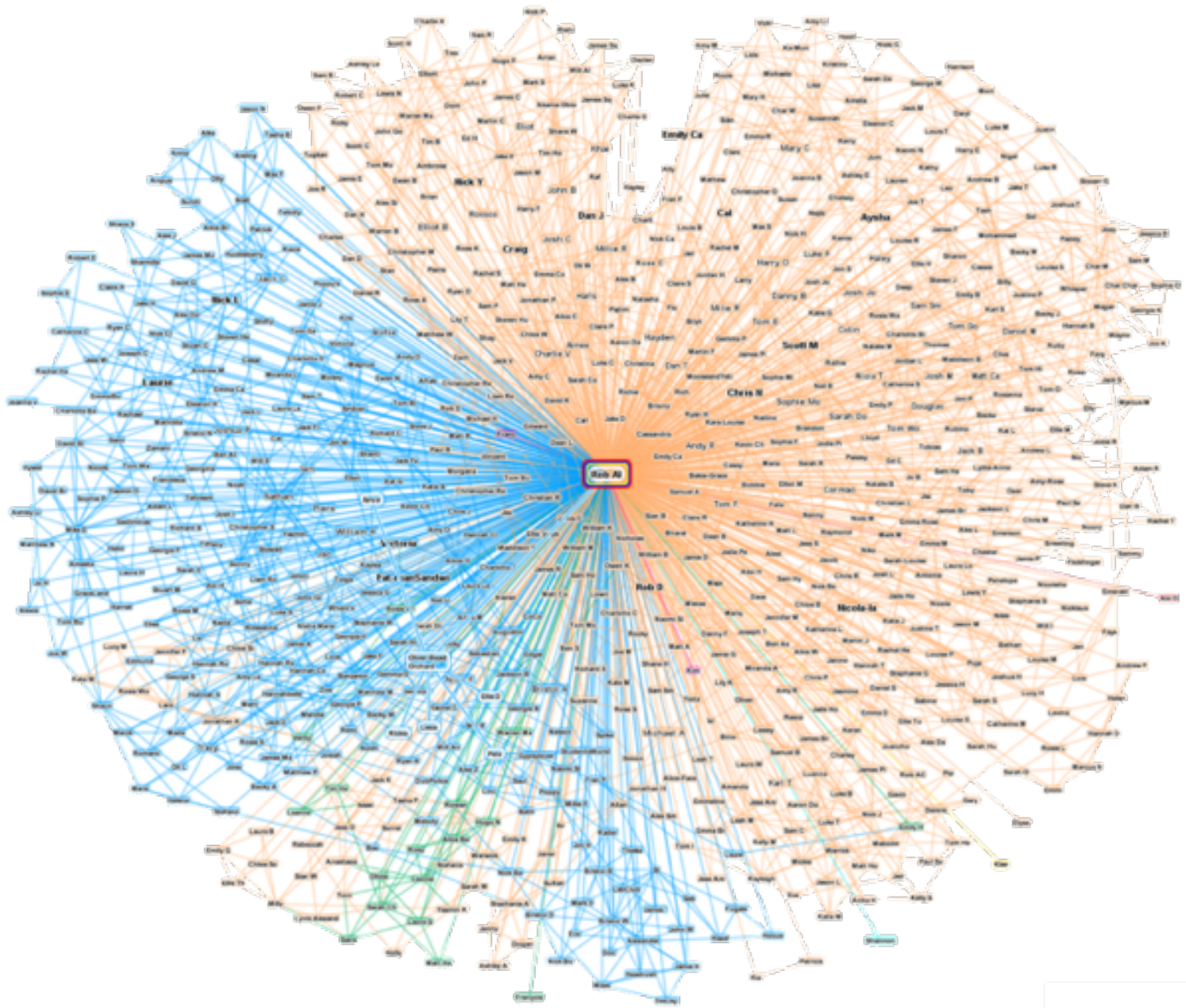
- **HyperGraph**
  - **Description:** A generalized graph where a relationship can connect an arbitrary amount of nodes[3] (compared to the more common binary graph models)
  - **Vendors:** HyperGraphDB, TrinityDB



1] Rodriguez, M.A., Neubauer, P., "Constructions from Dots and Lines," 2010, http://arxiv.org/abs/1006.2361
2] W3C, "The Resource Description Framework (RDF)," 2004, http://www.w3.org/RDF/
3] Wikipedia, http://en.wikipedia.org/wiki/Hypergraph

Graph Local Queries

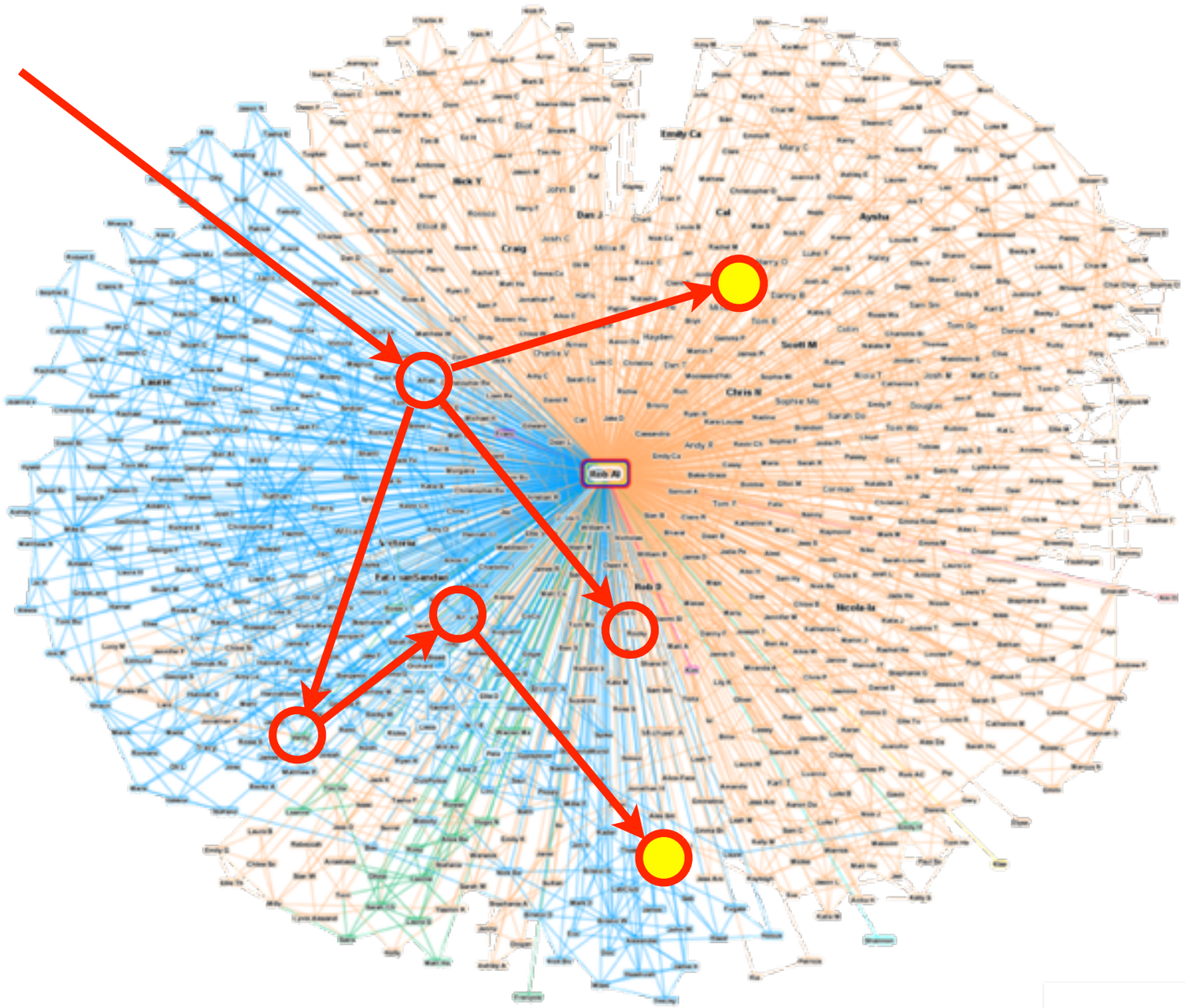
Sweet Spot for Graph Databases



e.g. Recommendations, Friend-of-Friend, Shortest Path

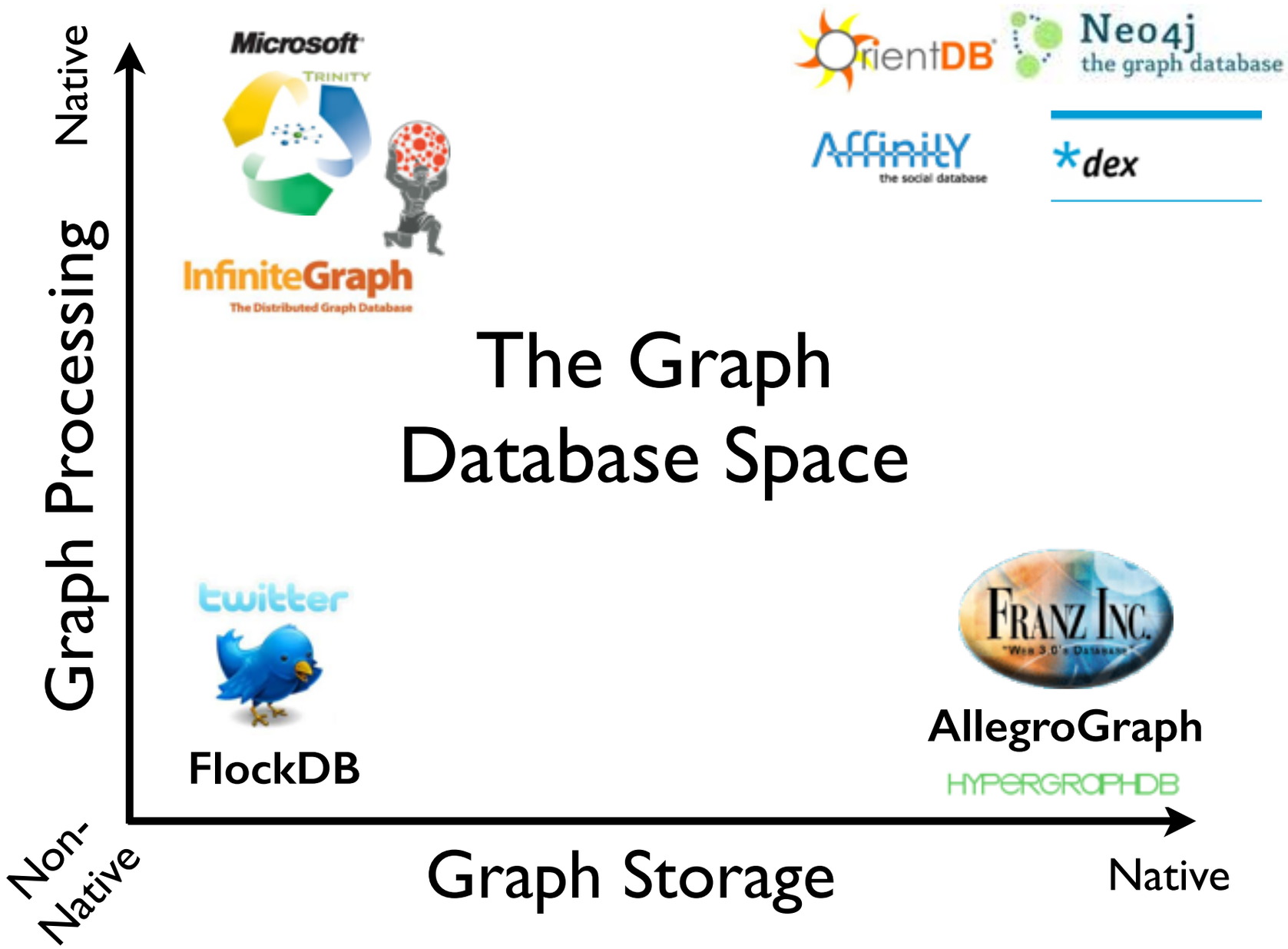
Graph Local Queries

Sweet Spot for Graph Databases



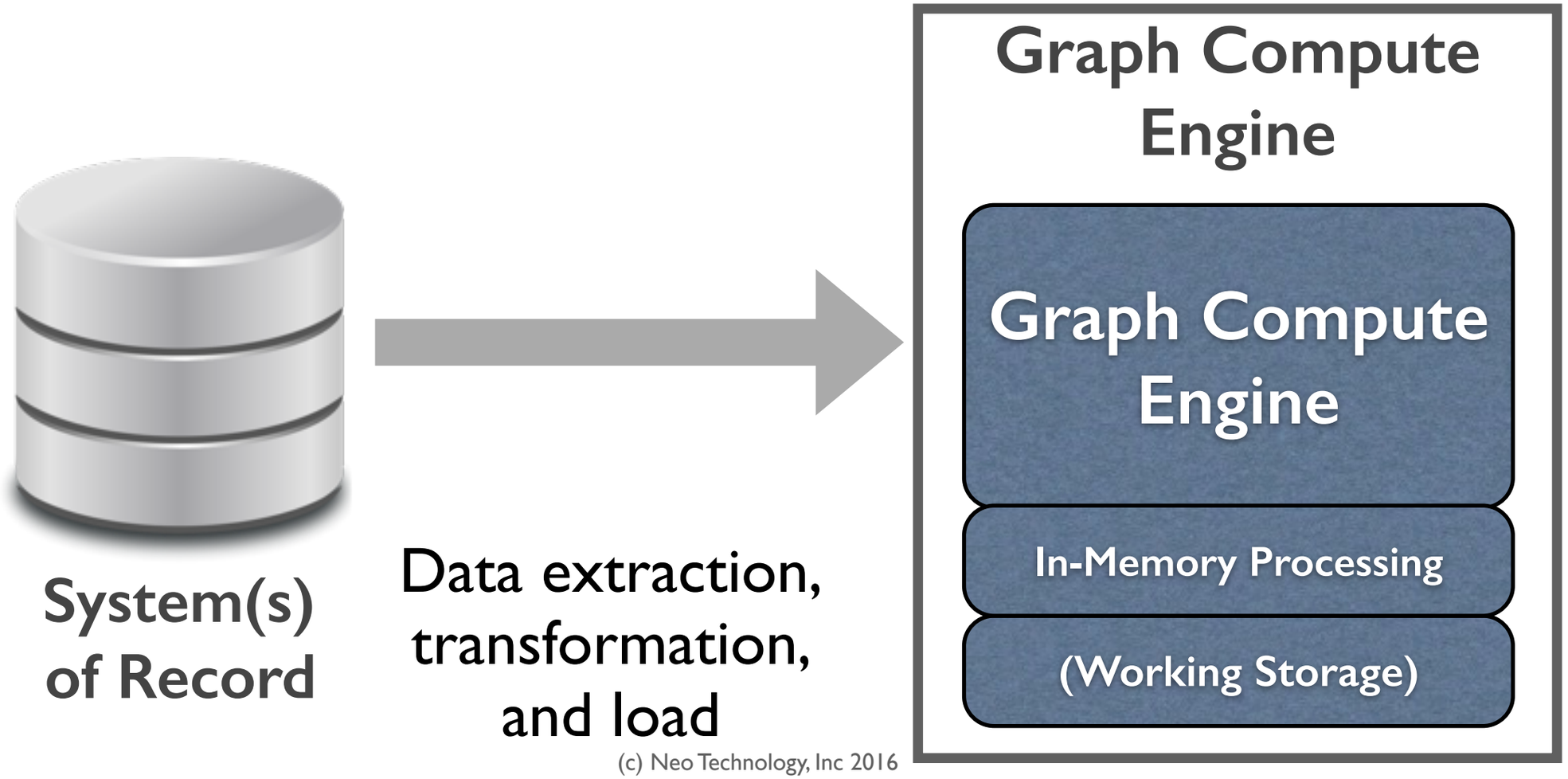
e.g. Recommendations, Friend-of-Friend, Shortest Path

The Emerging Graph Database Space



2. What is a Graph Compute Engine

Processing platforms that enable graph global computational algorithms to be run against large data sets



Graph Compute Engines

Largely fall into one of these patterns:

- In-Memory / Single Machine
  - Cassovary , GraphChi, YarcData uRiKA
- Distributed - most common of which is the "Bulk Synchronous Parallel Model" (aka Pregel clone)
  - Giraph, GraphLab, Pegasus
- Batch or Stream Processing
  - Hadoop, Apache Spark, Apache Flink

What Is A Graph, Anyway?

Graph Theory

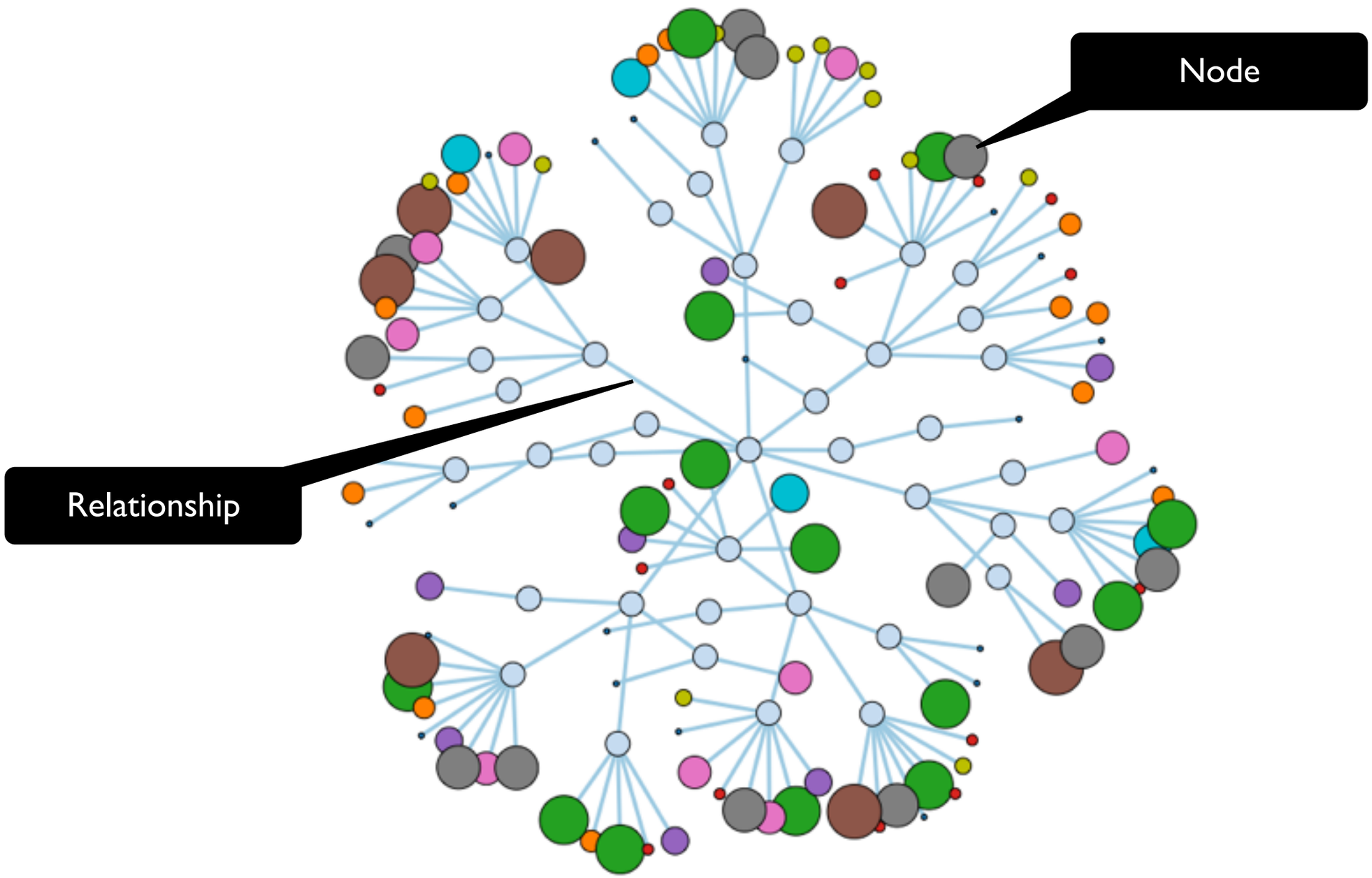
- Long History since Leonard Euler in Königsberg
- Field of Mathematics
- We use a similar underlying datamodel
- But this is about databases, not just algorithms

Graph Models

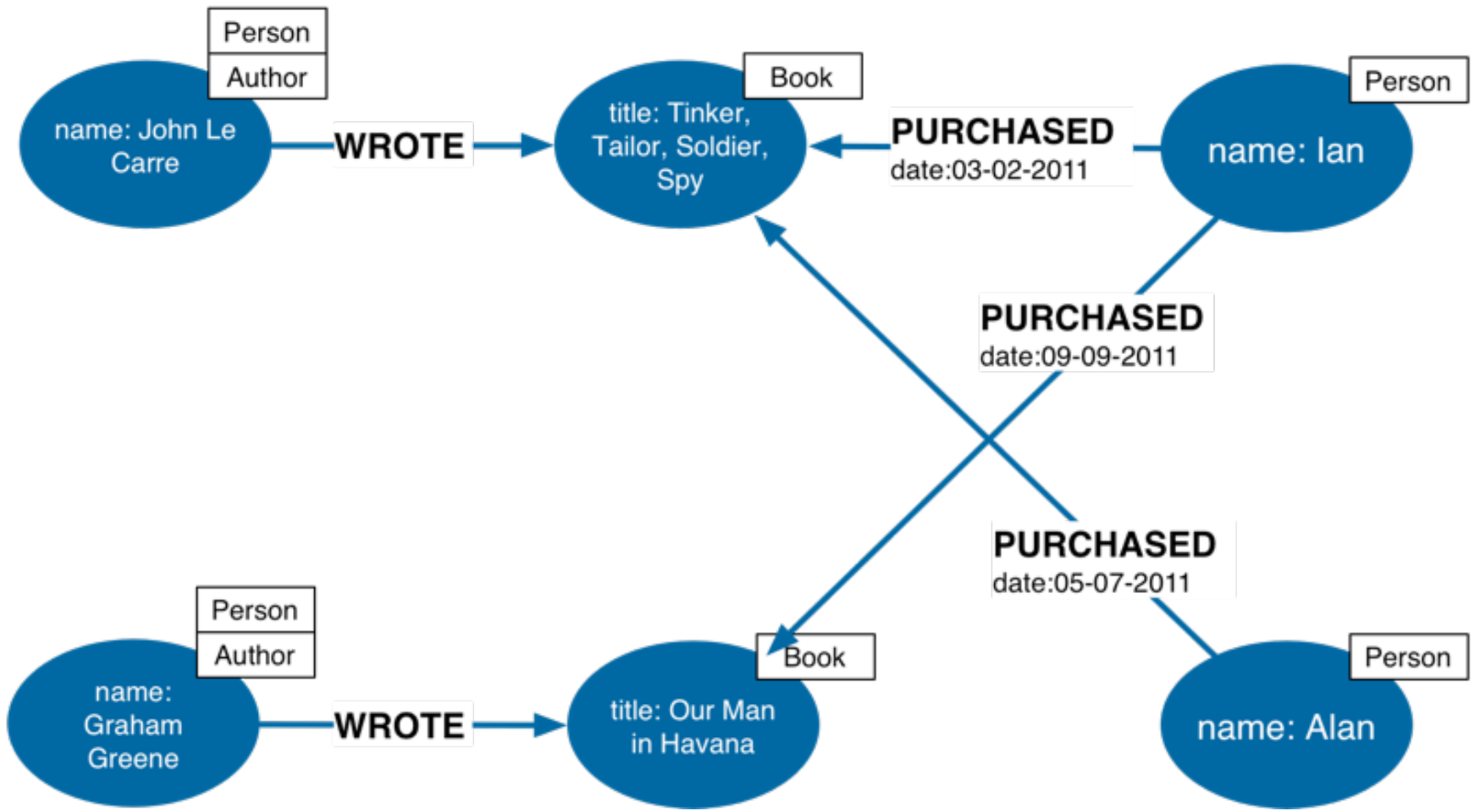
- Mathematical, Adjacency Matrix
- RDF - Triples
- HyperGraph
- **Property Graph**

Property Graph Model
Building Blocks

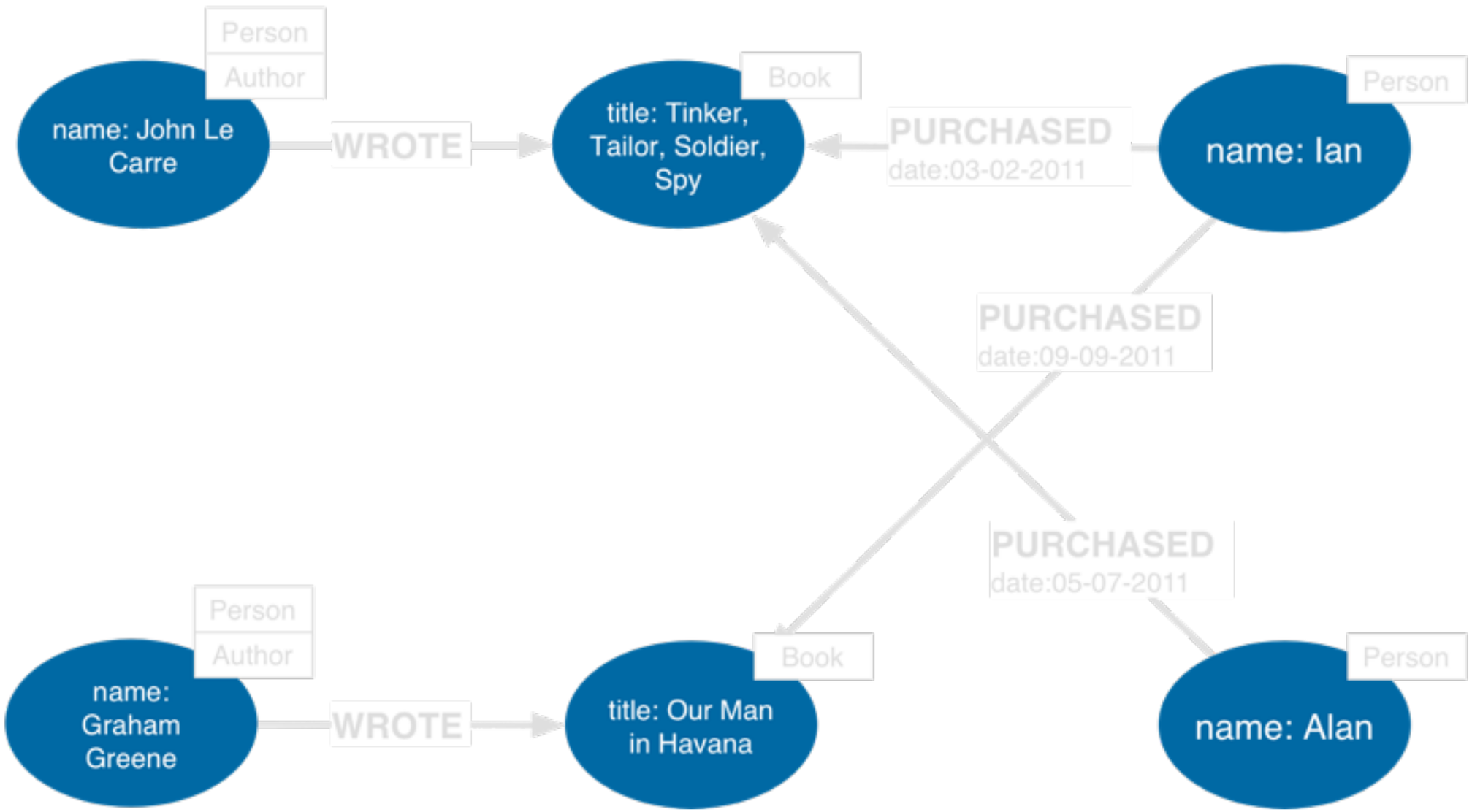
A Graph



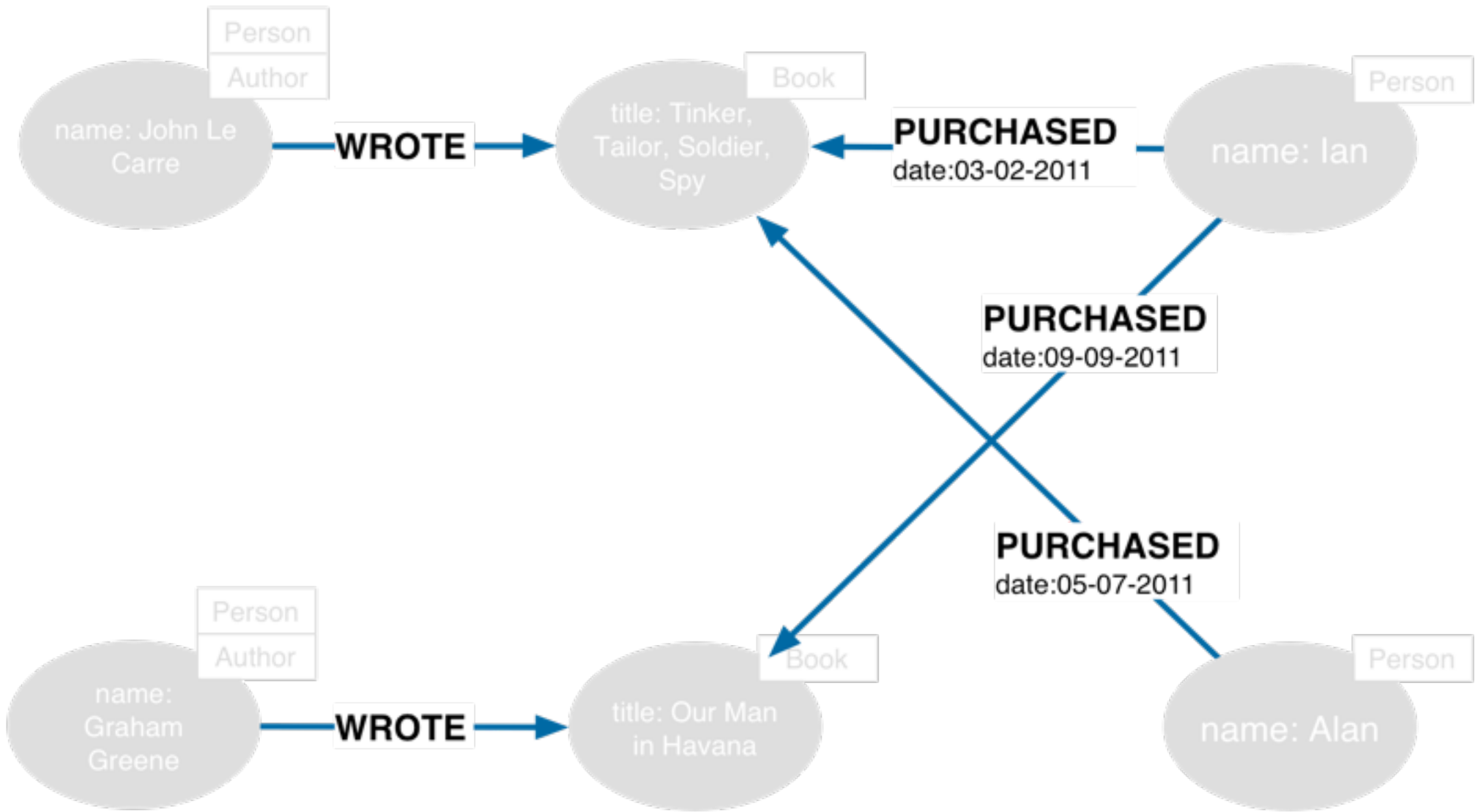
Property Graph Data Model



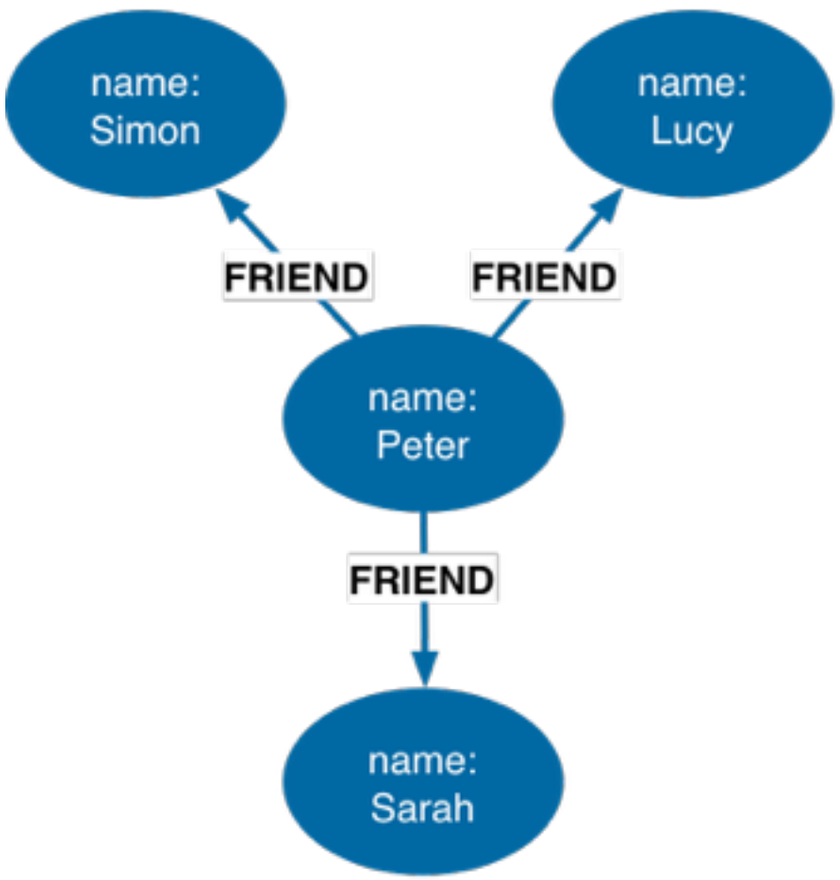
Nodes



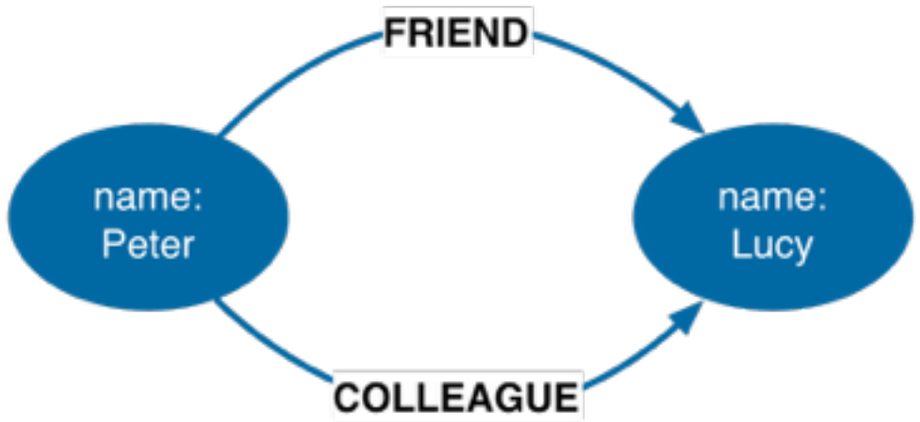
Relationships



Relationships (continued)



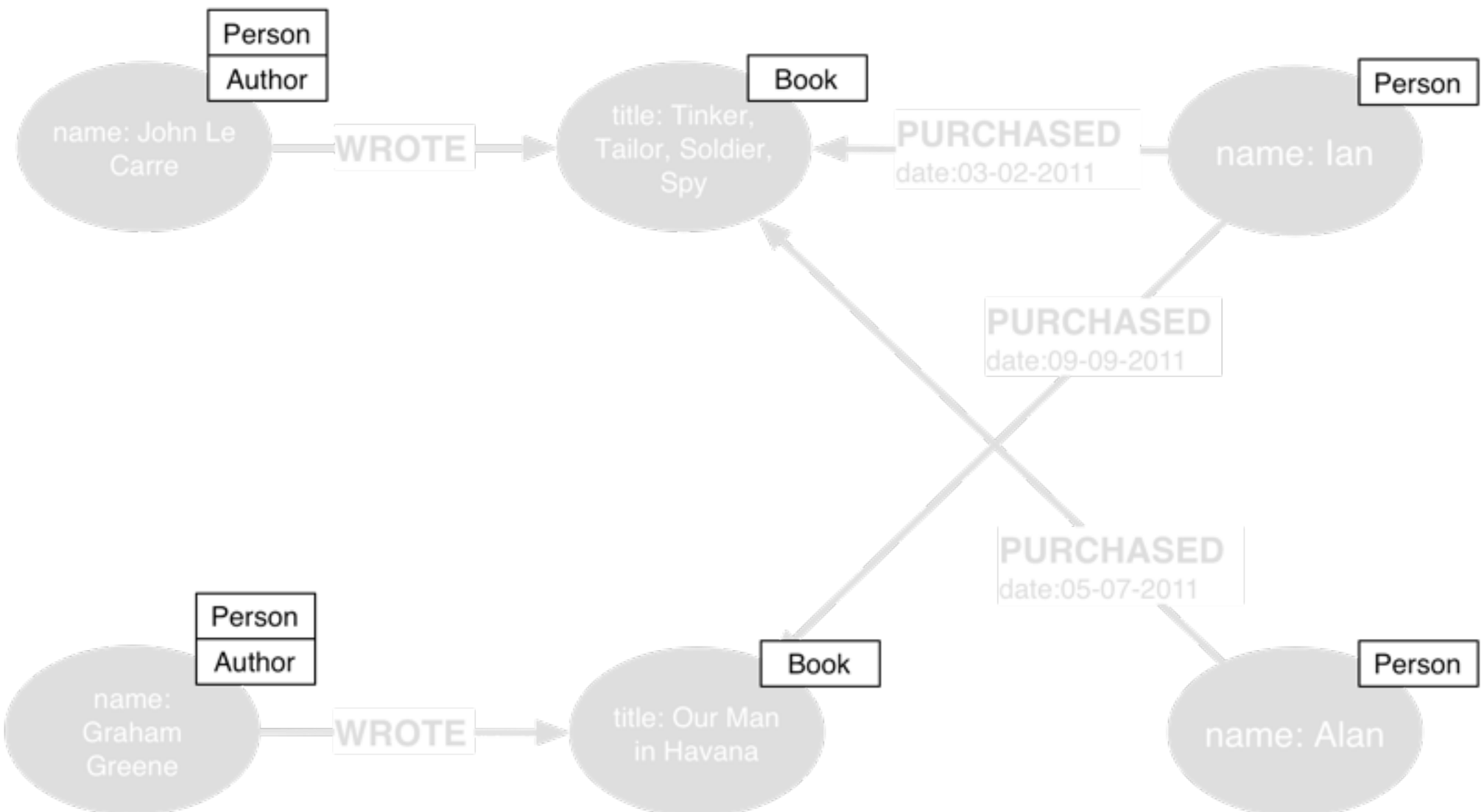
Nodes can have more than one relationship



Nodes can be connected by more than one relationship

Self relationships are allowed

Labels



Four Building Blocks

- Nodes
  - Entities & complex value types
- Relationships
  - Connect entities and structure domain
- Properties
  - Attributes and metadata
- Labels
  - Group nodes by role

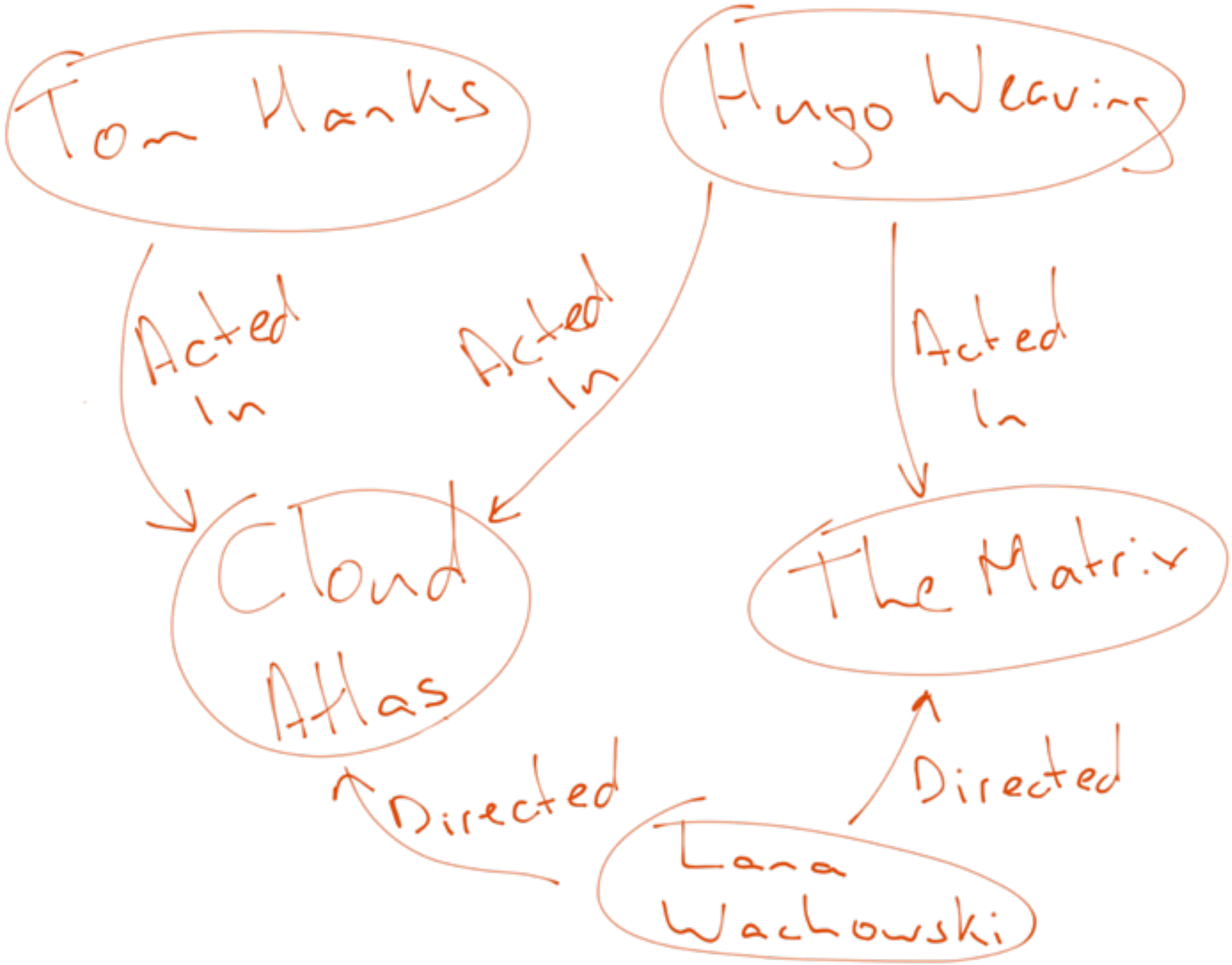
What do you need?

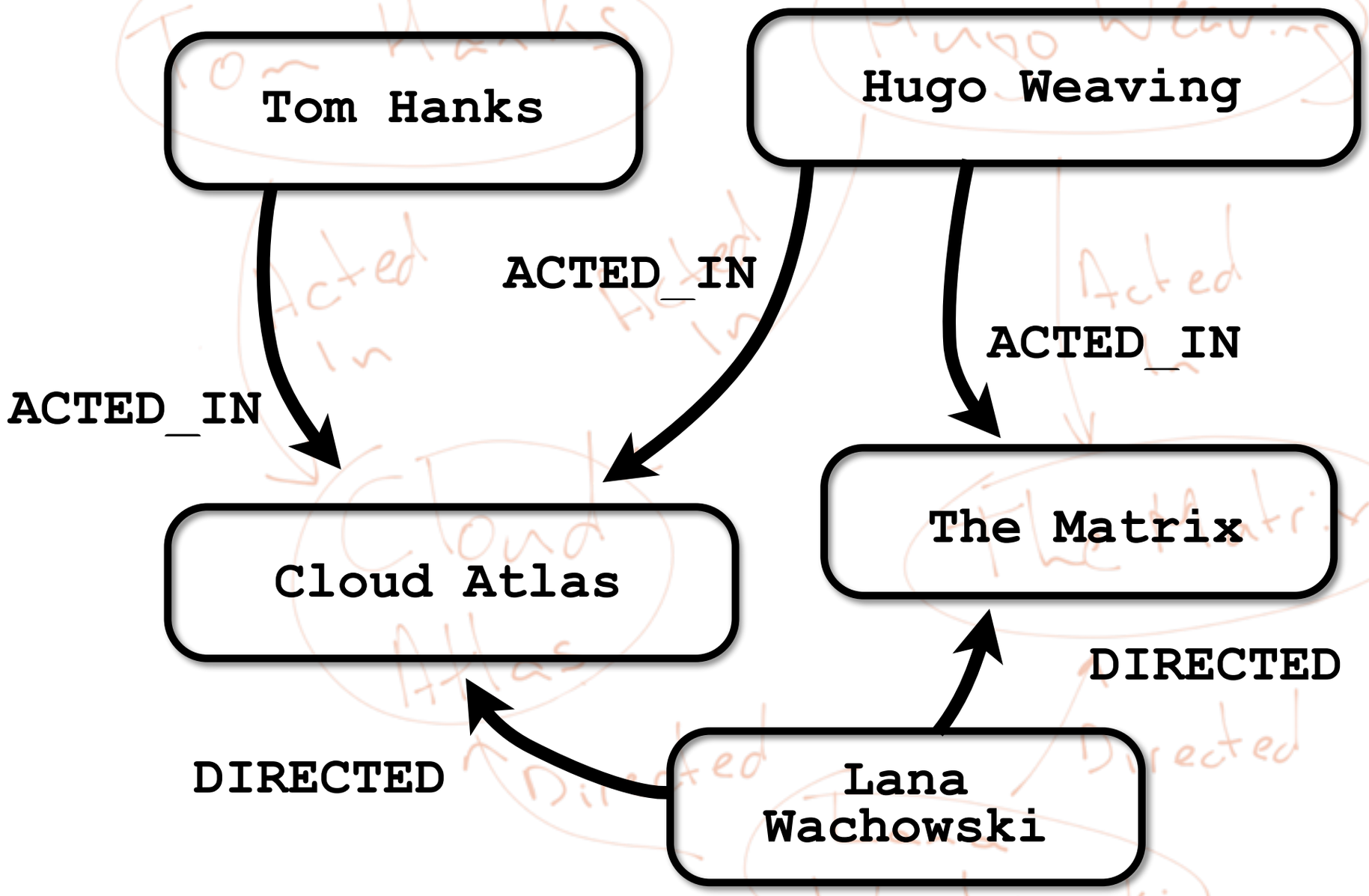
What do you need?

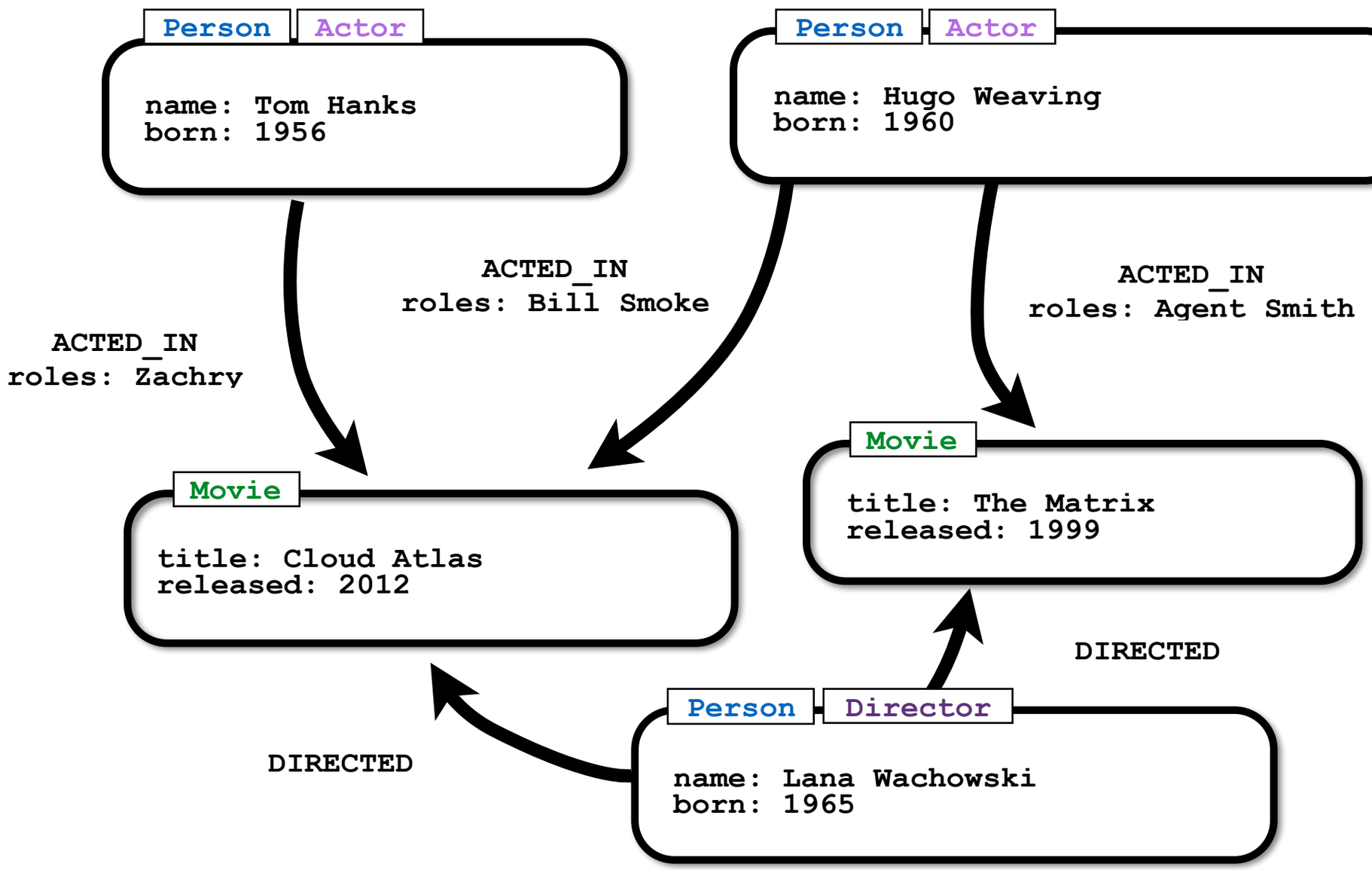
A whiteboard!

Whiteboard Friendlyness

Easy to design and model
direct representation of the model








Tom Hanks
Hugo Weaving
ACTED_IN
ACTED_IN
ACTED_IN
Cloud Atlas
The Matrix
DIRECTED
DIRECTED
Lana Wachowski

Relational to Graph

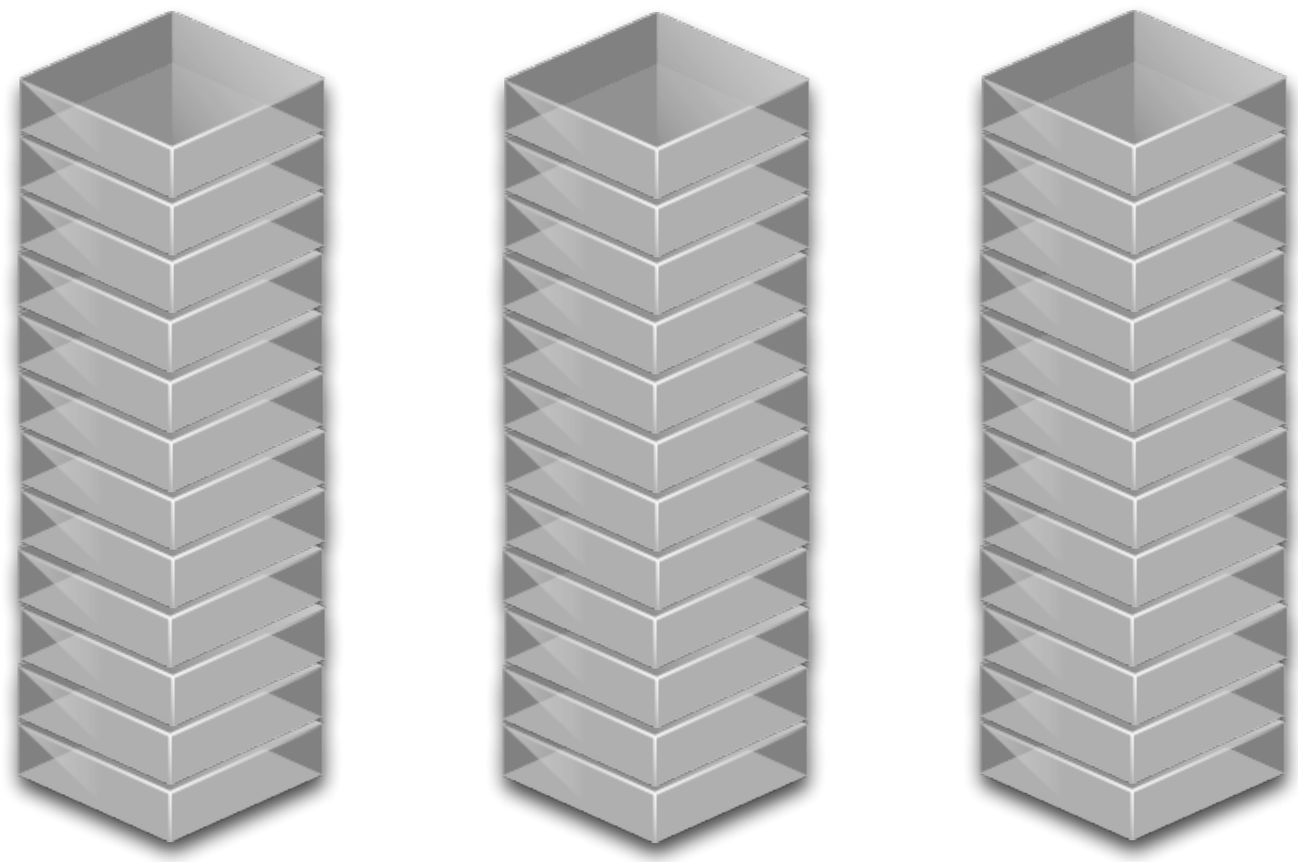
Relational to Graph

Relational to Graph

Relational is simple

Relational to Graph

Relational is simple



Relational to Graph

Relational is simple

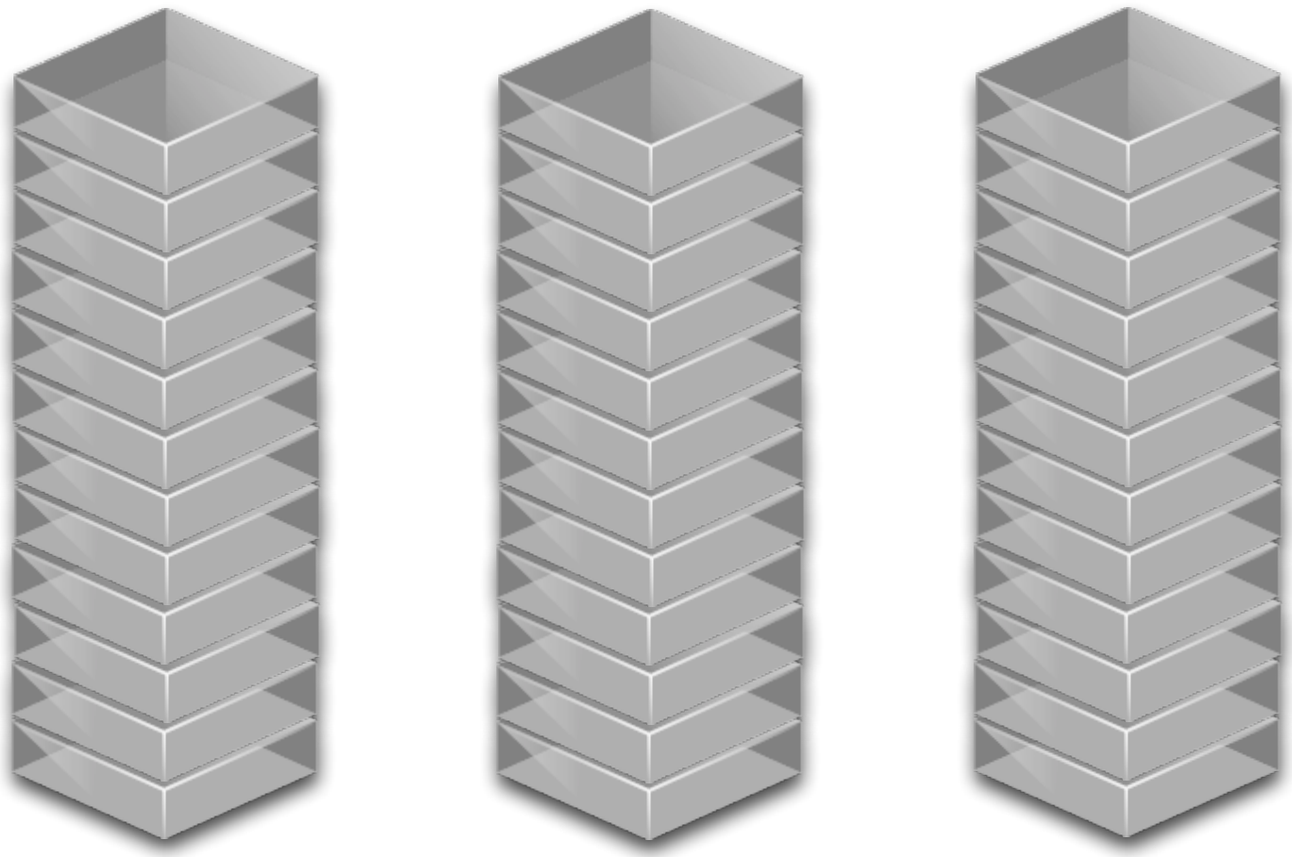
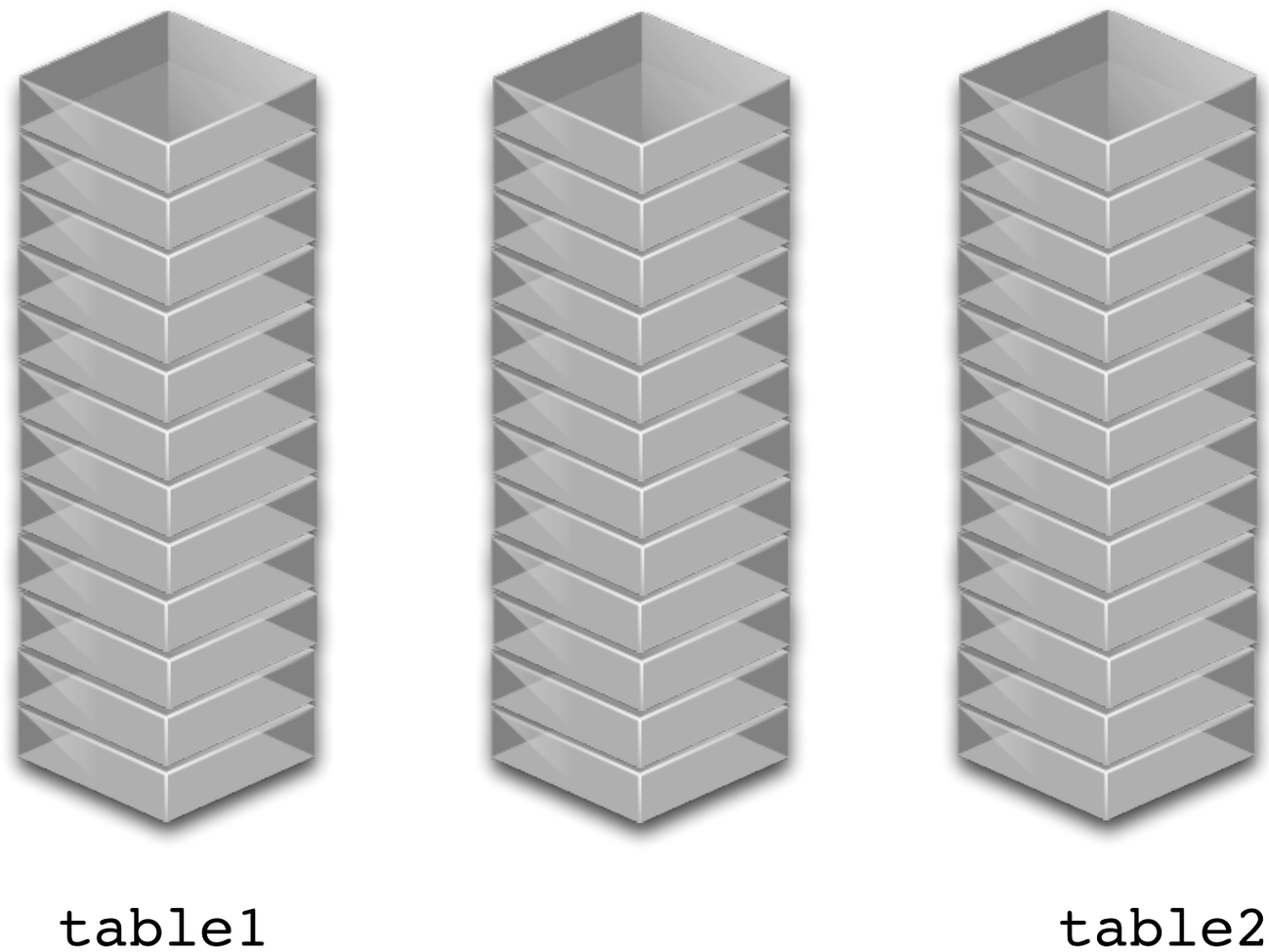


table1

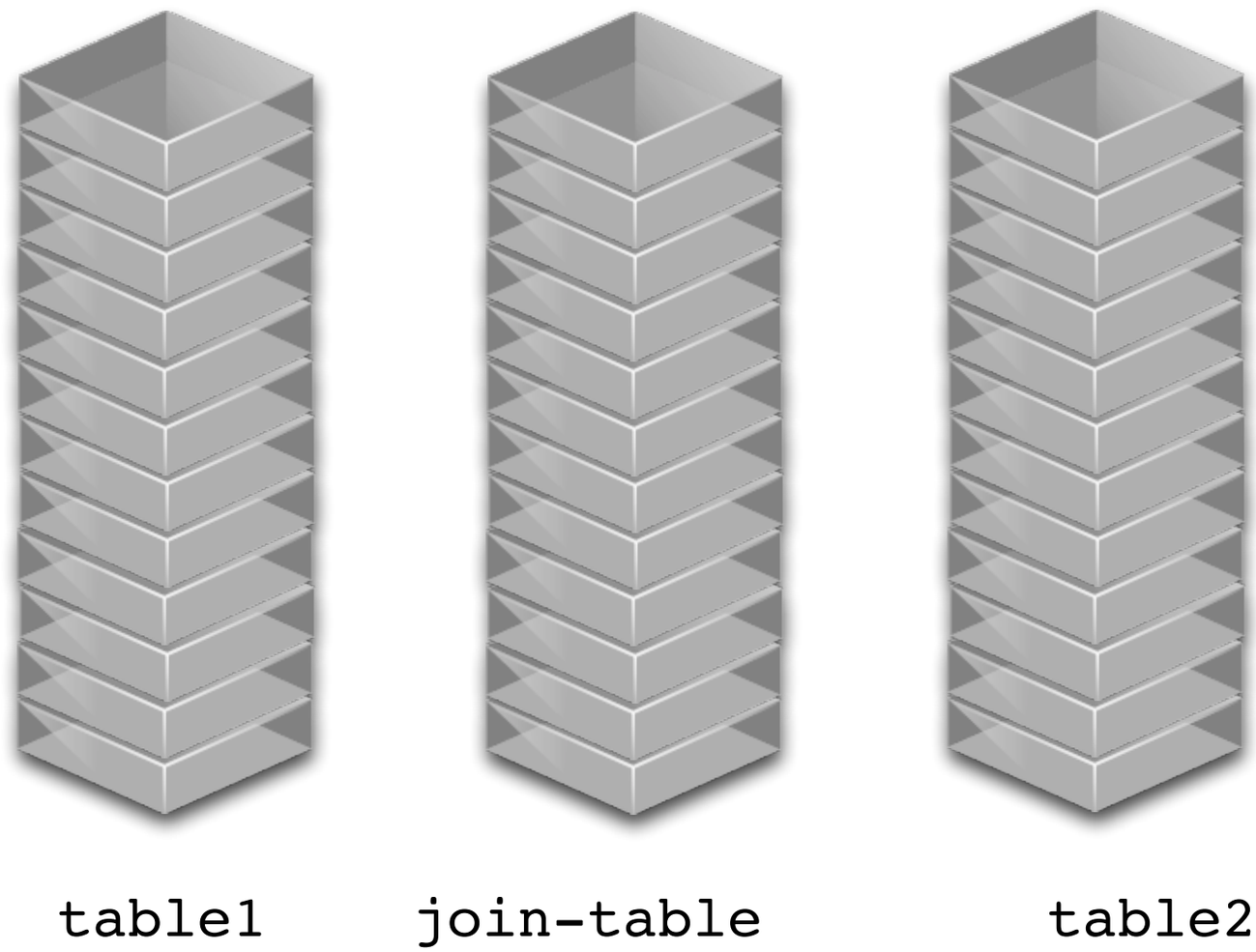
Relational to Graph

Relational is simple



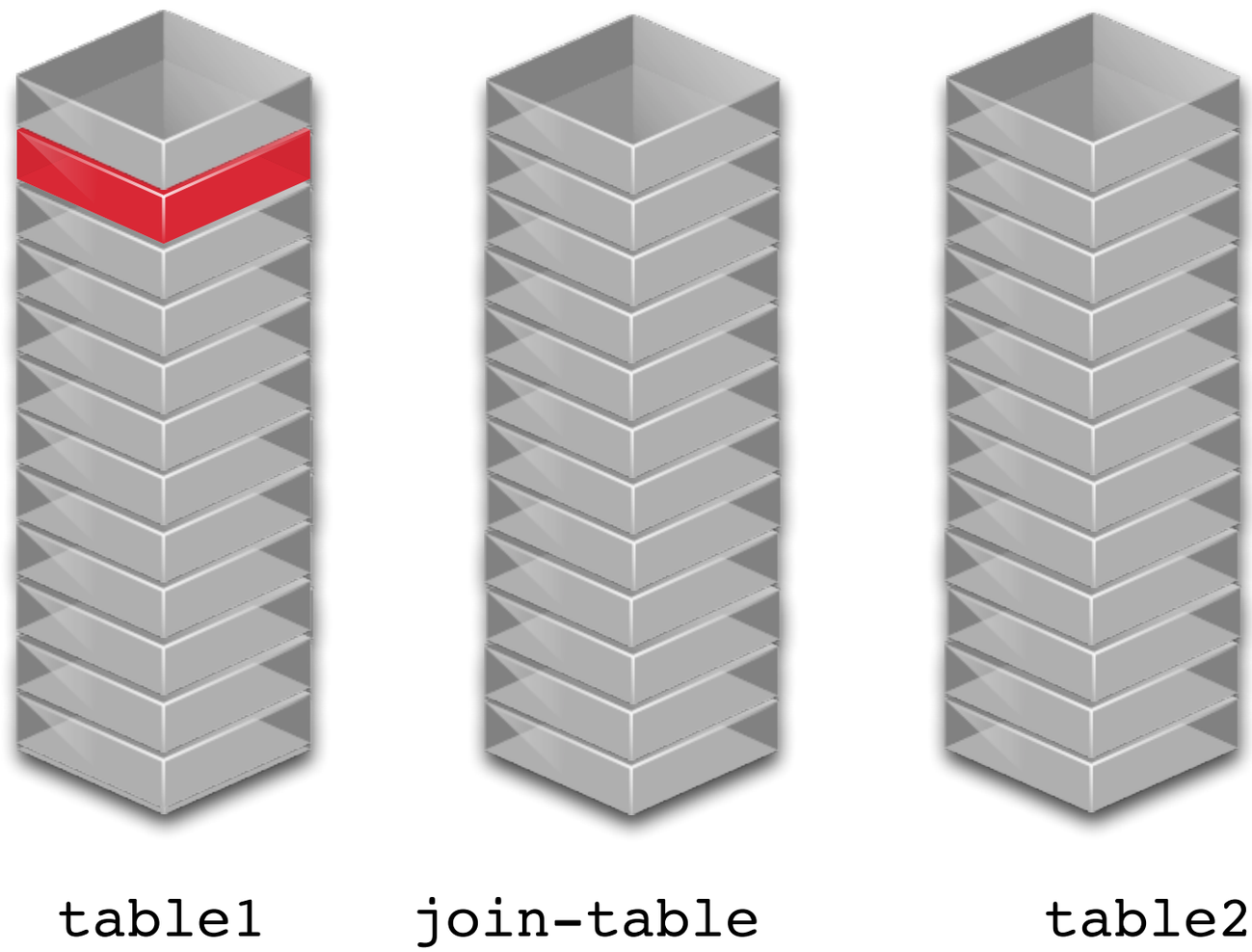
Relational to Graph

Relational is simple



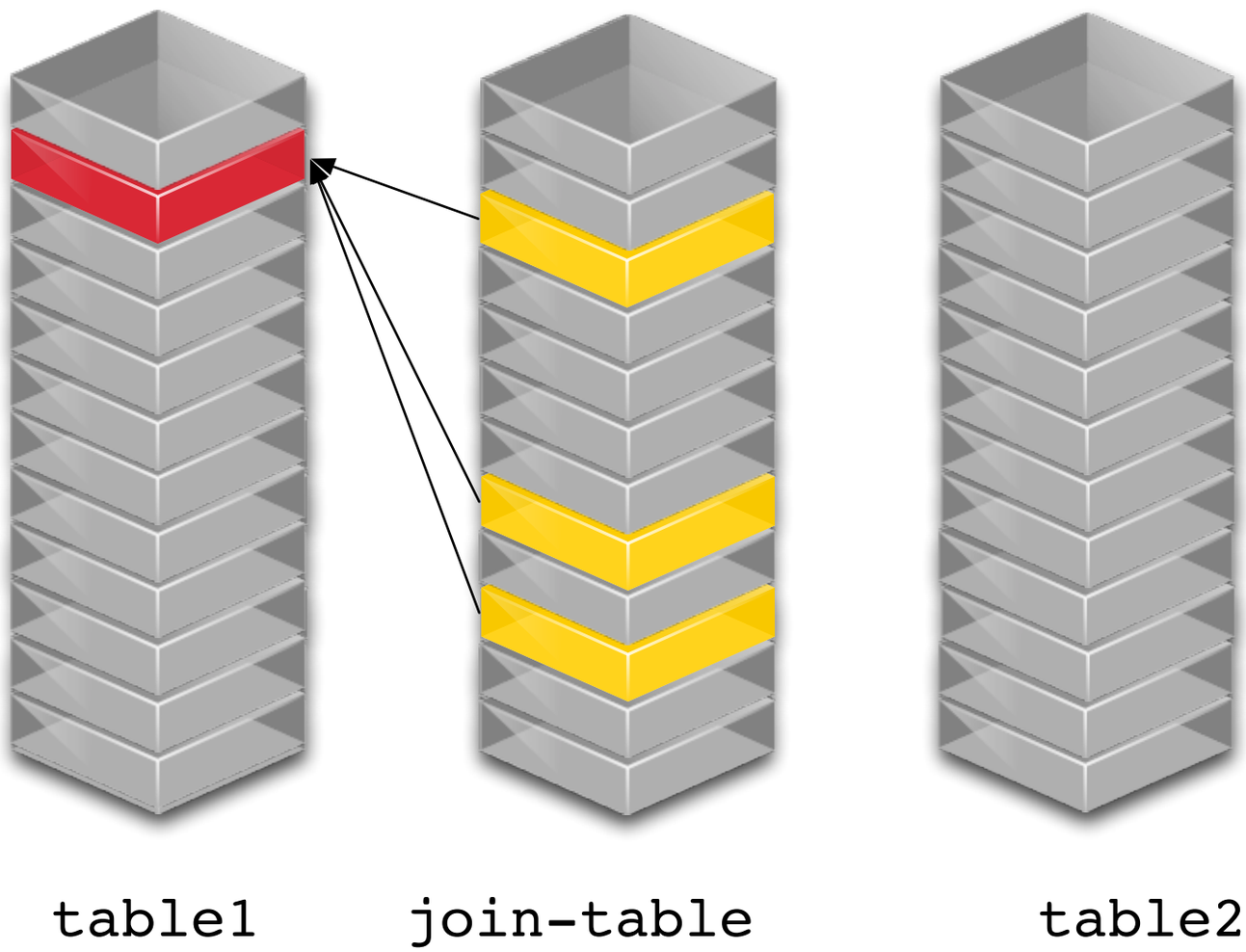
Relational to Graph

Relational is simple



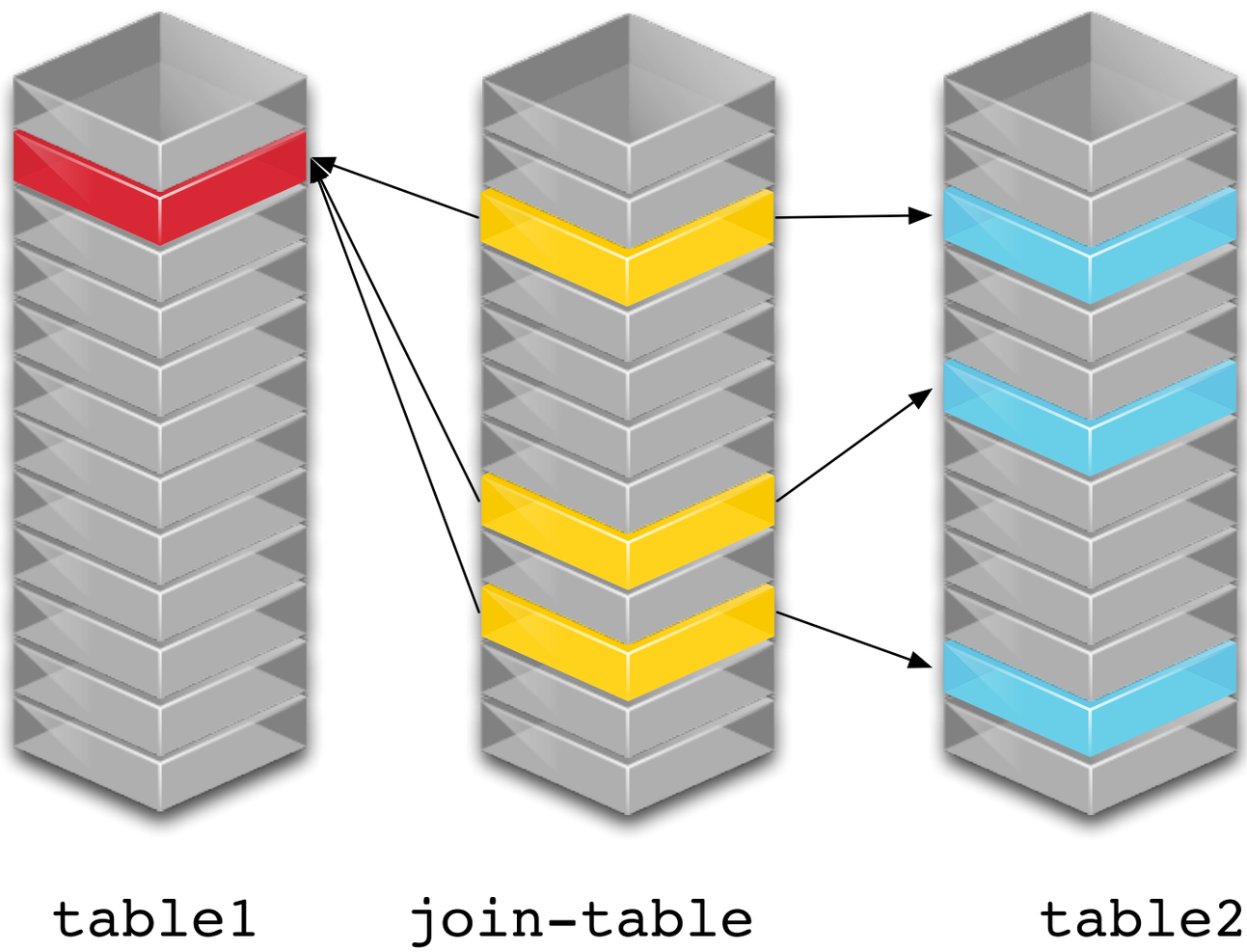
Relational to Graph

Relational is simple



Relational to Graph

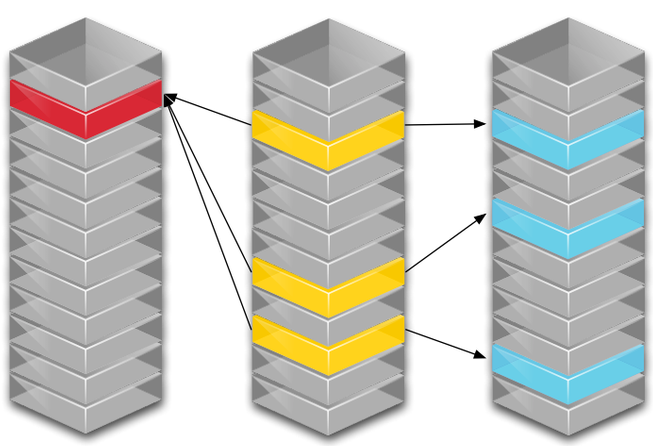
Relational is simple

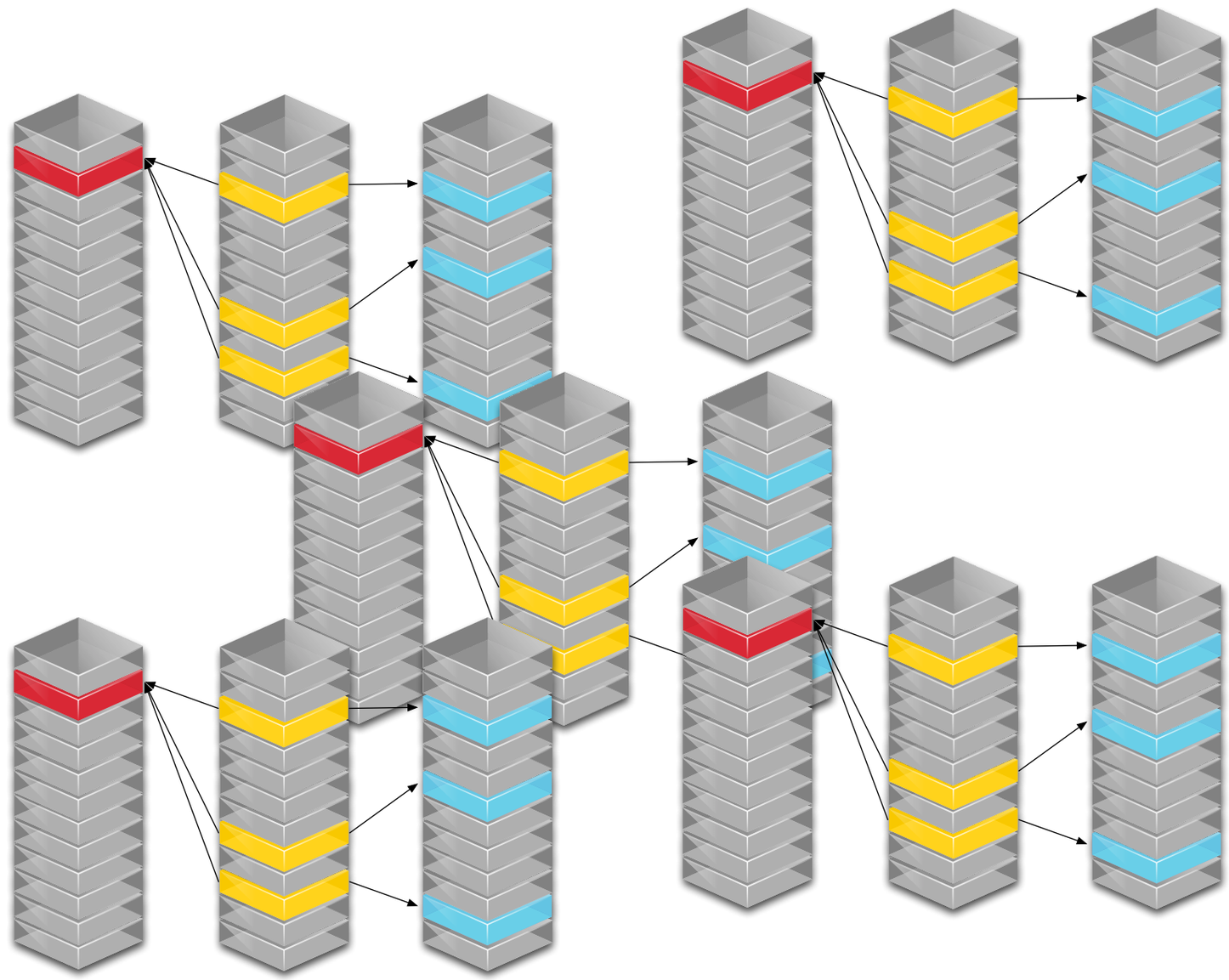


Relational to Graph

Relational is simple

until it gets complicated ...





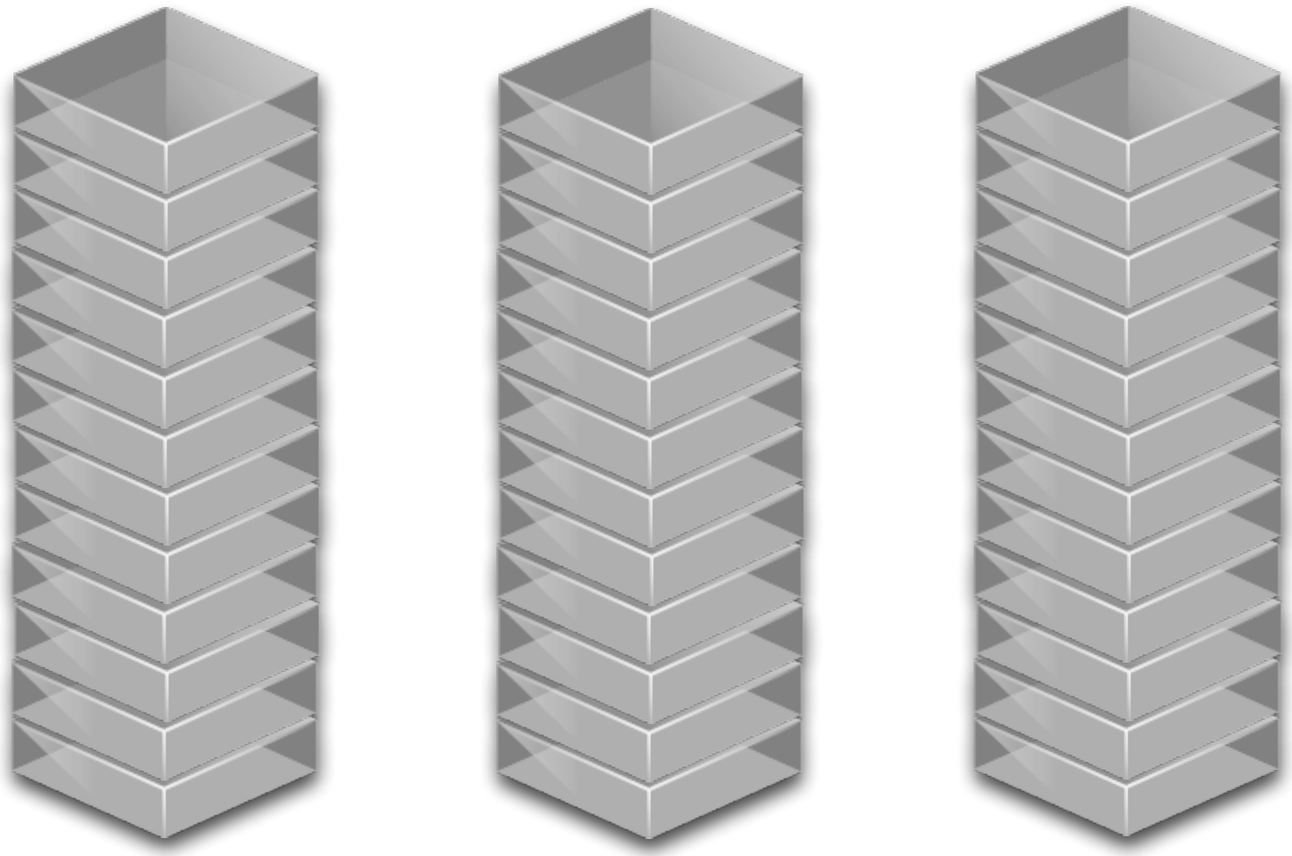
Relational to Graph

Relational to Graph

You know relational

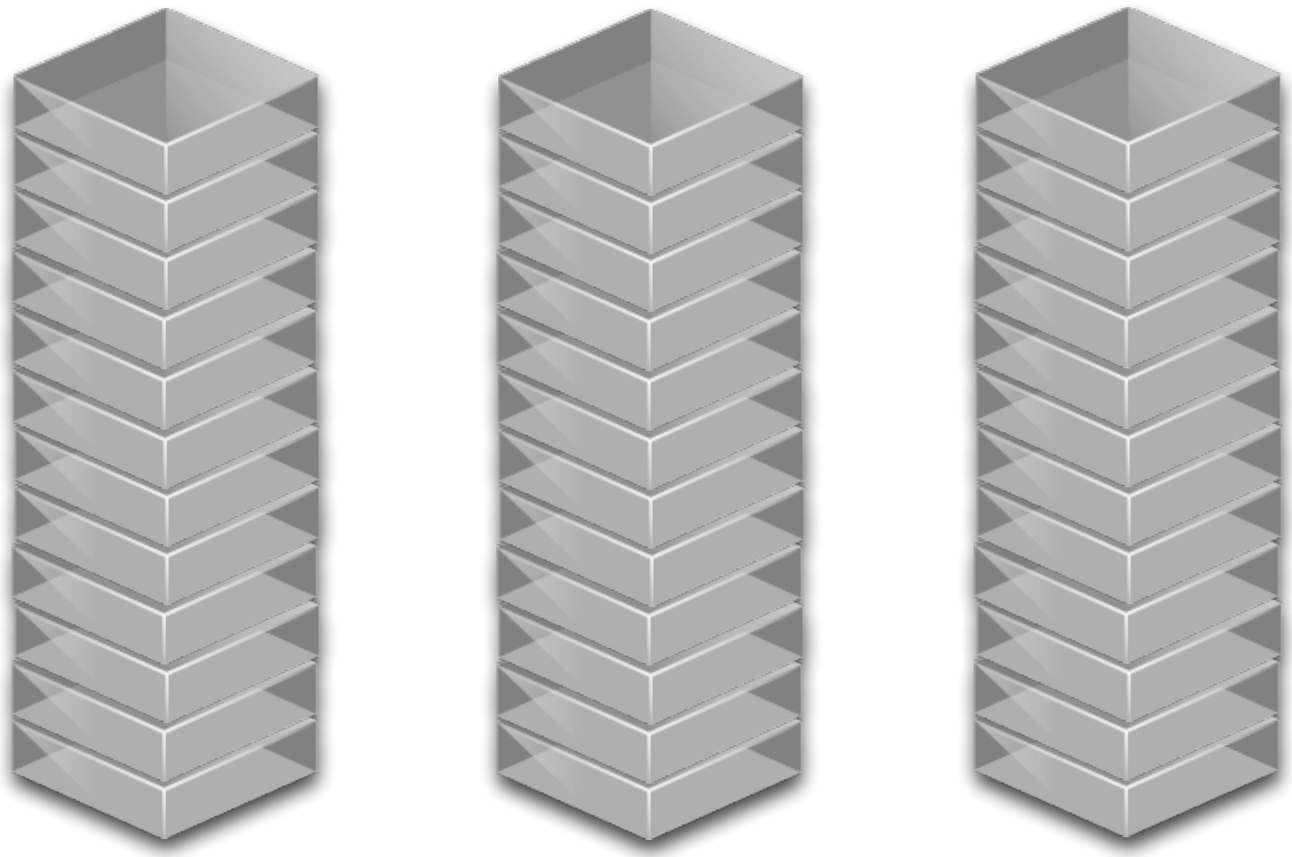
Relational to Graph

You know relational



Relational to Graph

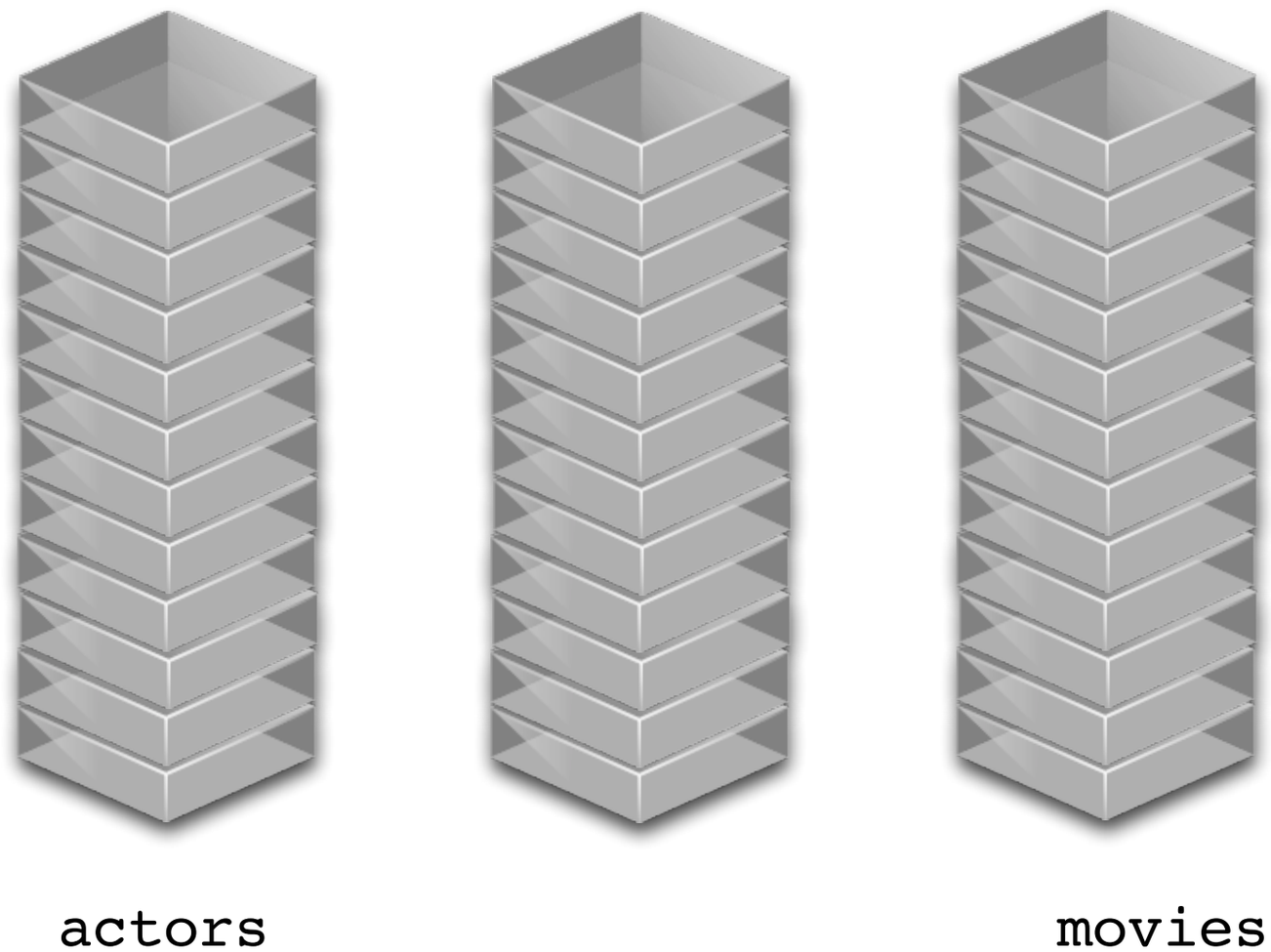
You know relational



actors

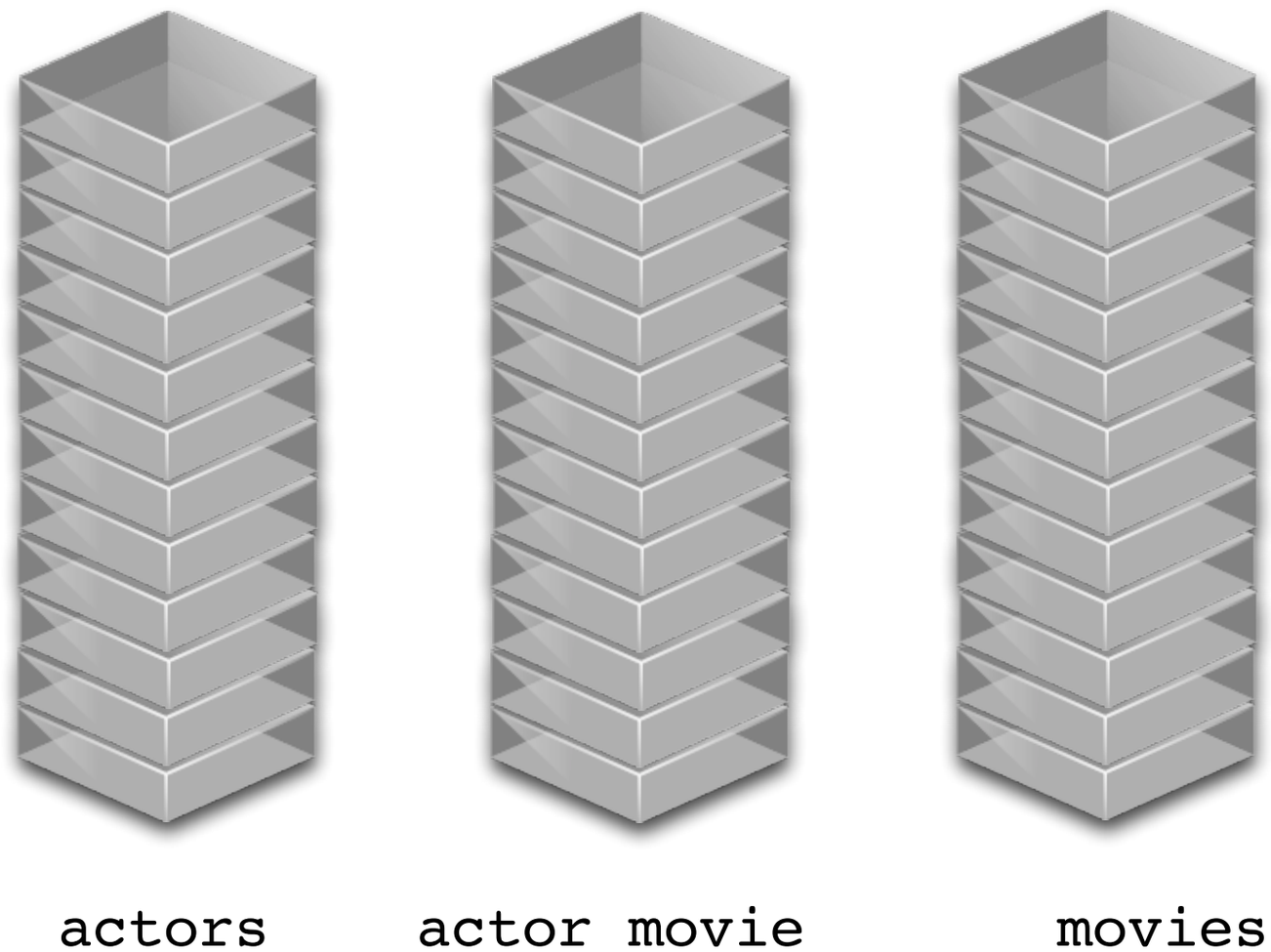
Relational to Graph

You know relational



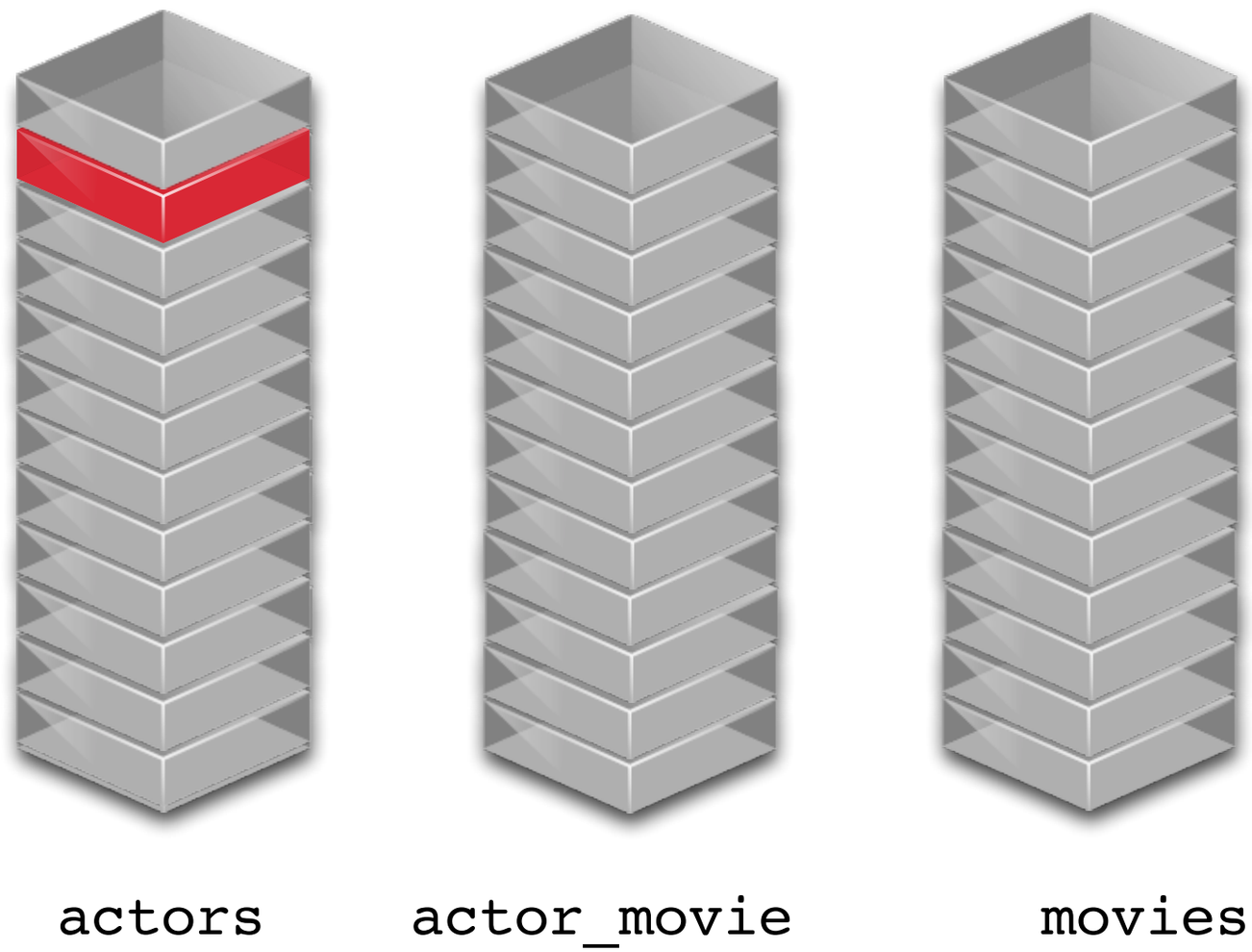
Relational to Graph

You know relational



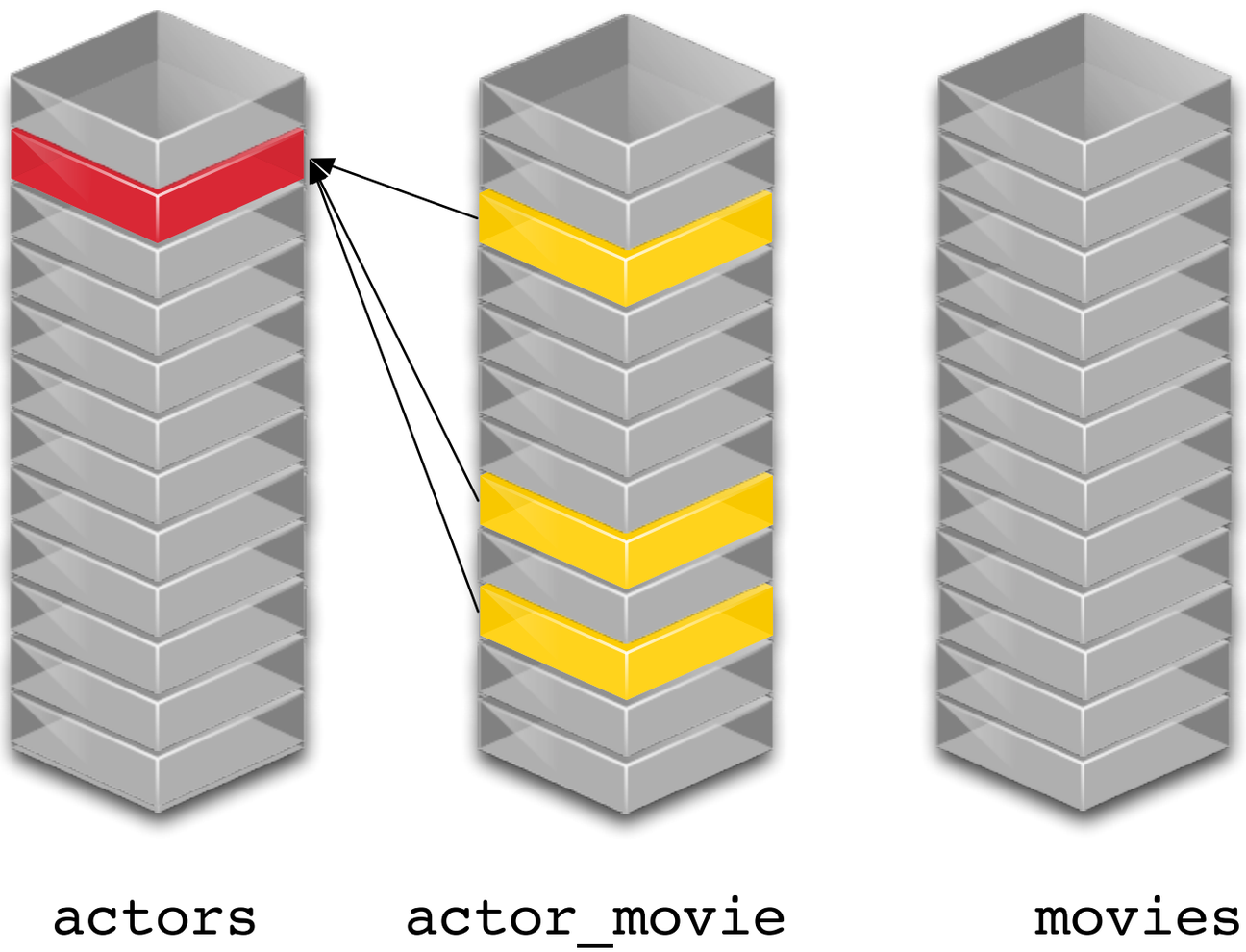
Relational to Graph

You know relational



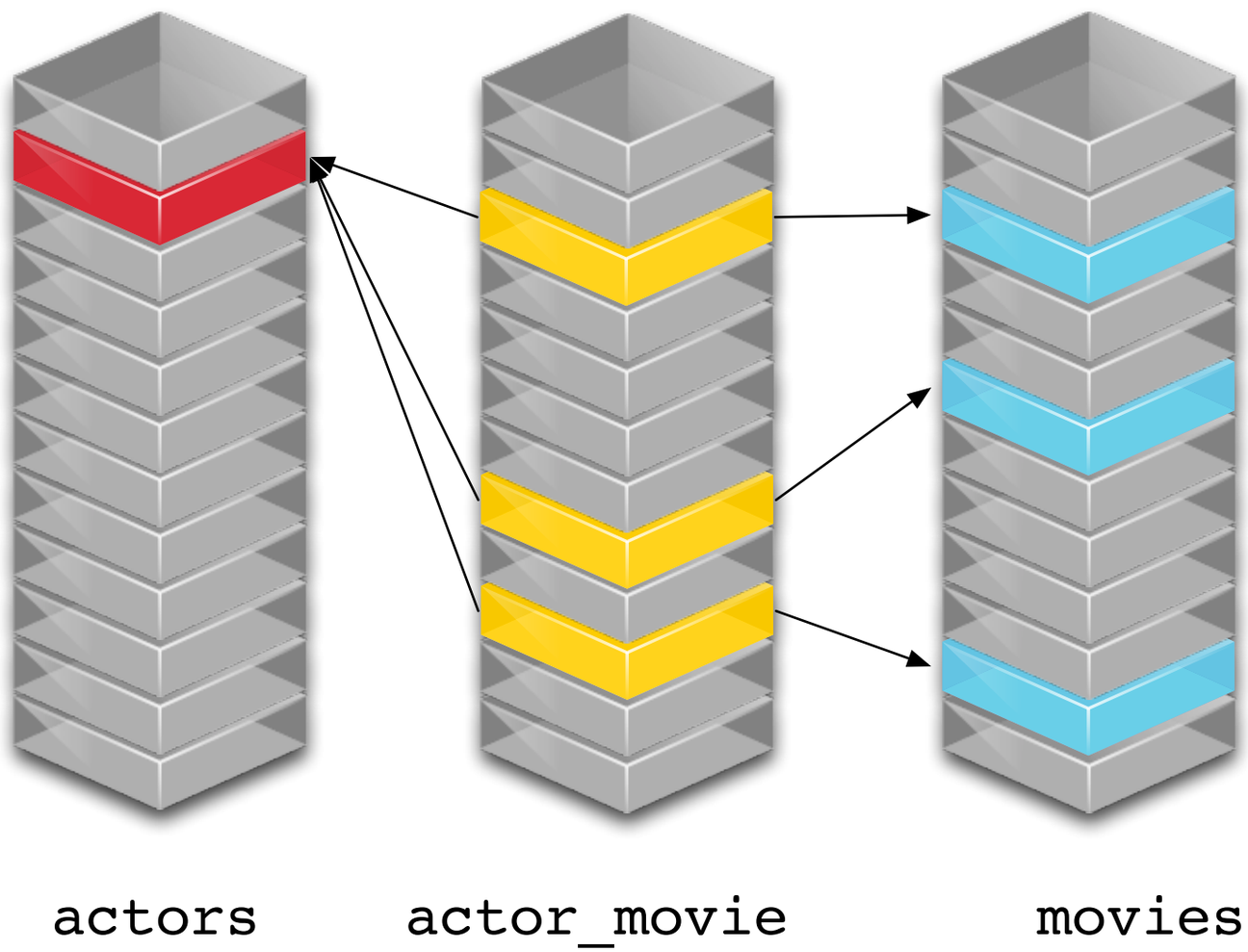
Relational to Graph

You know relational



Relational to Graph

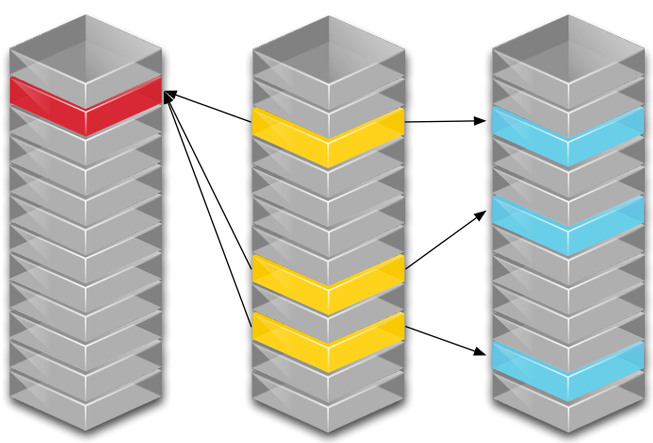
You know relational



Relational to Graph

You know relational

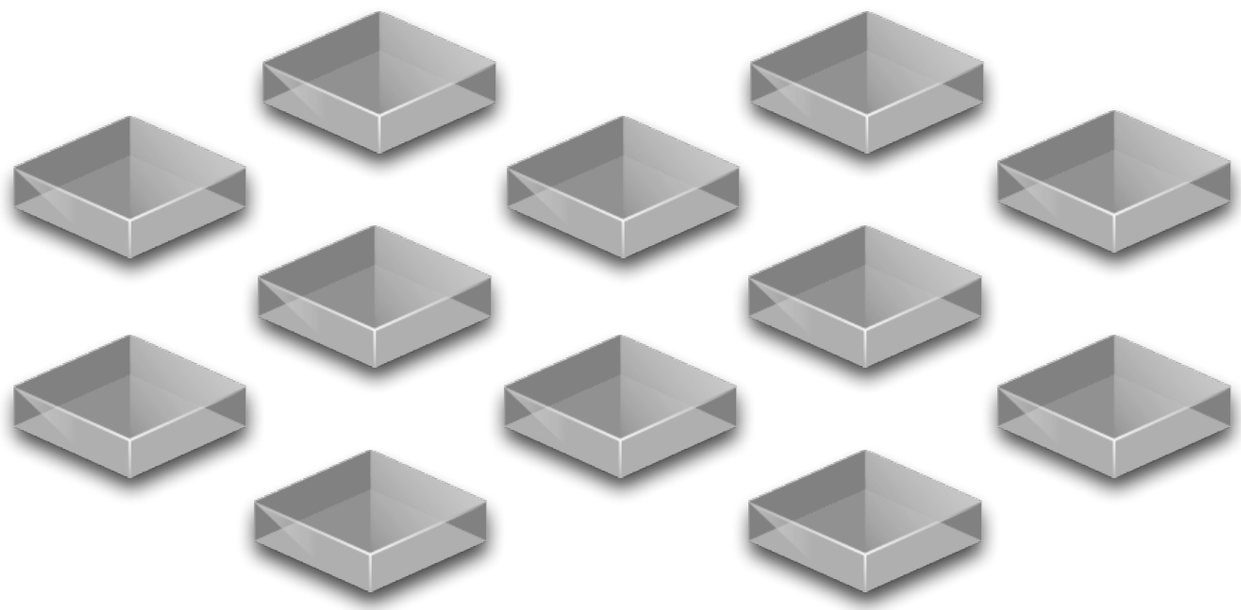
now consider relationships...

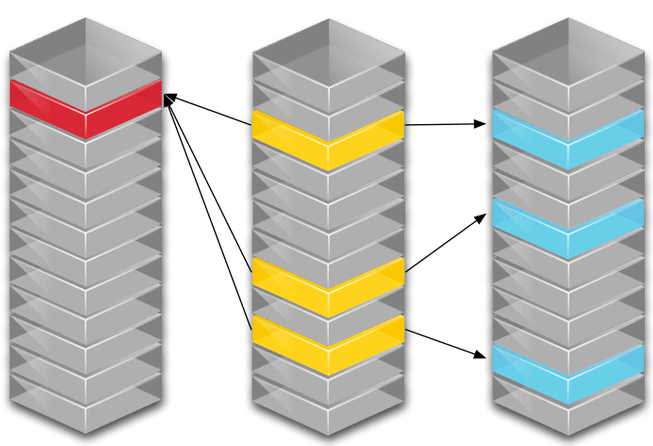


Relational to Graph

You know relational

now consider relationships...

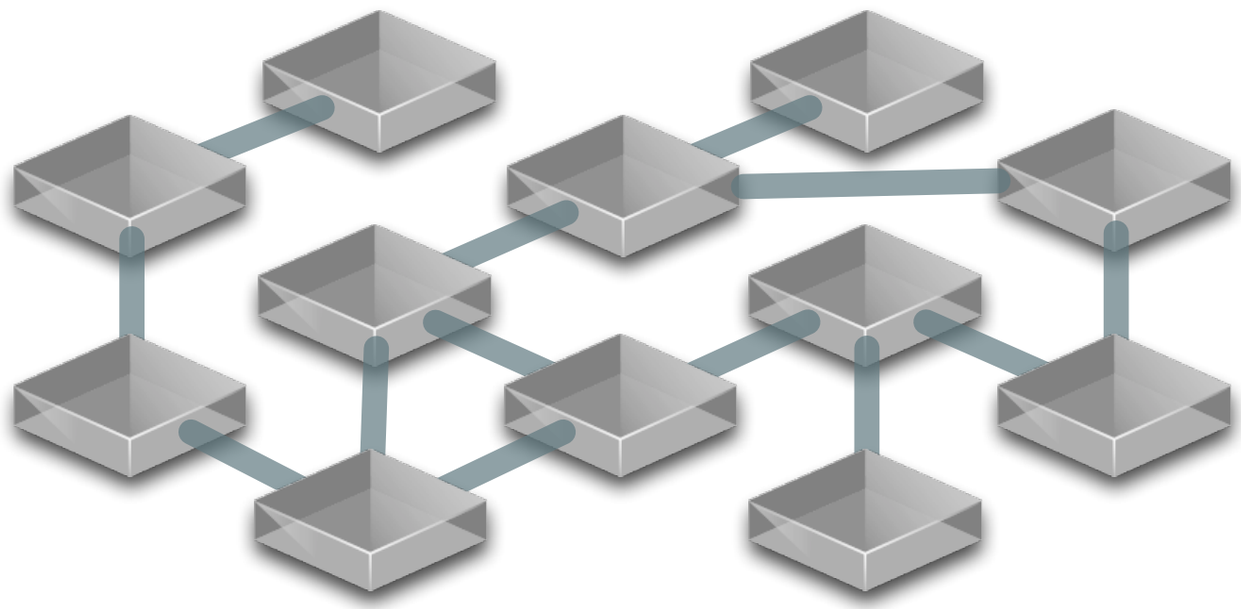


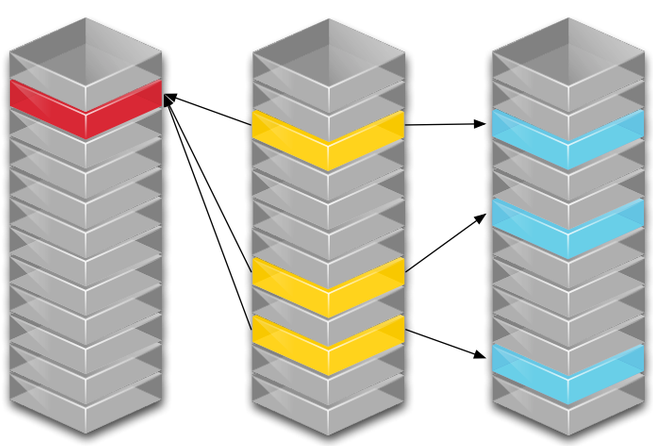


Relational to Graph

You know relational

now consider relationships...

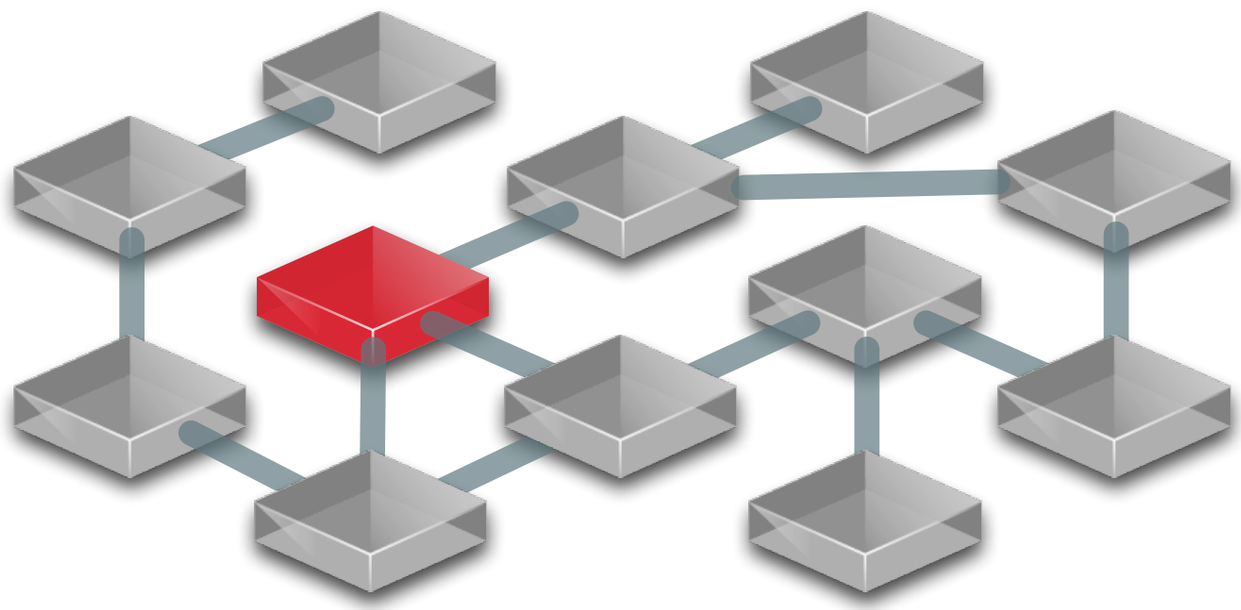


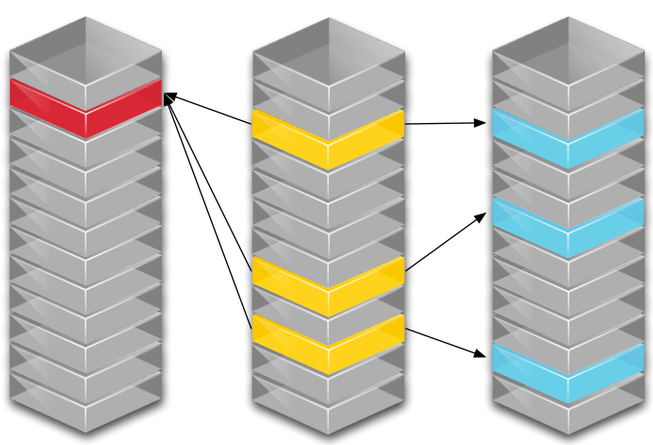


Relational to Graph

You know relational

now consider relationships...

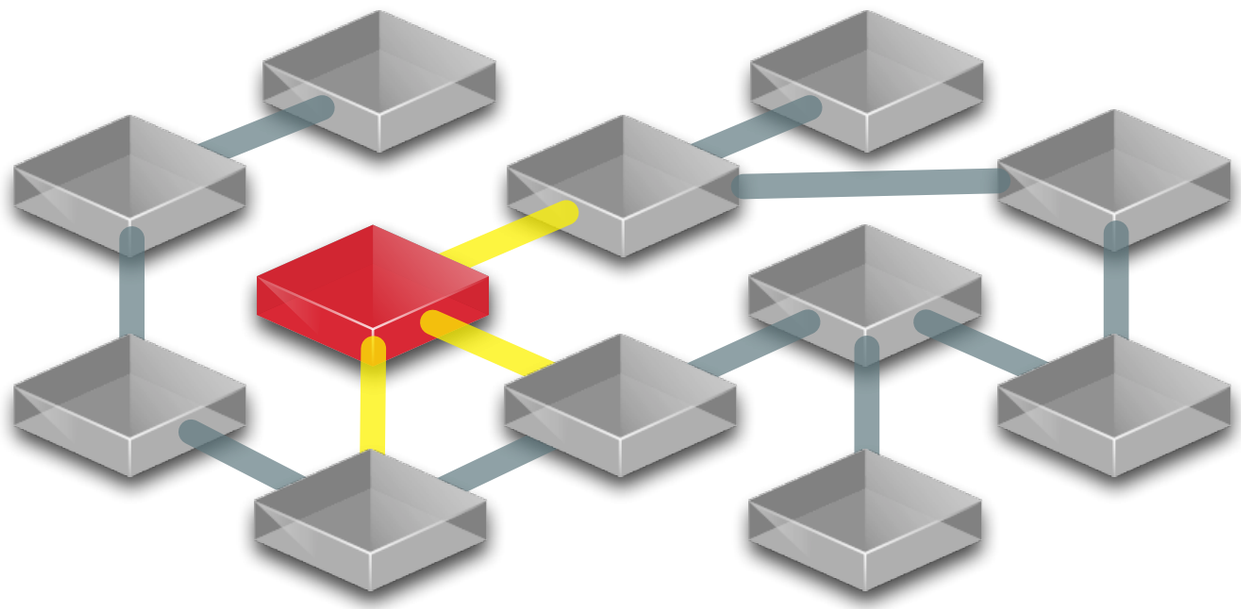


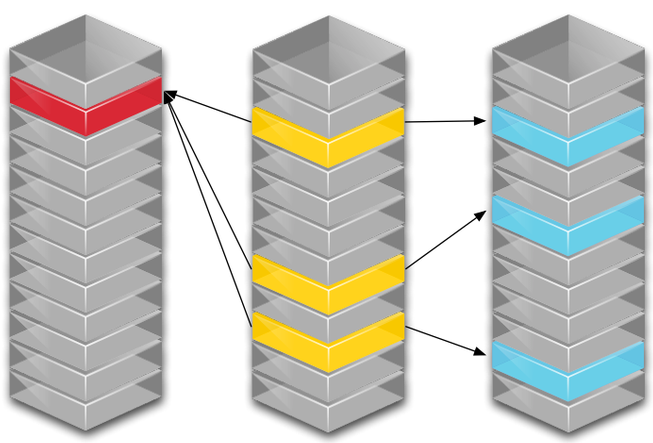


Relational to Graph

You know relational

now consider relationships...

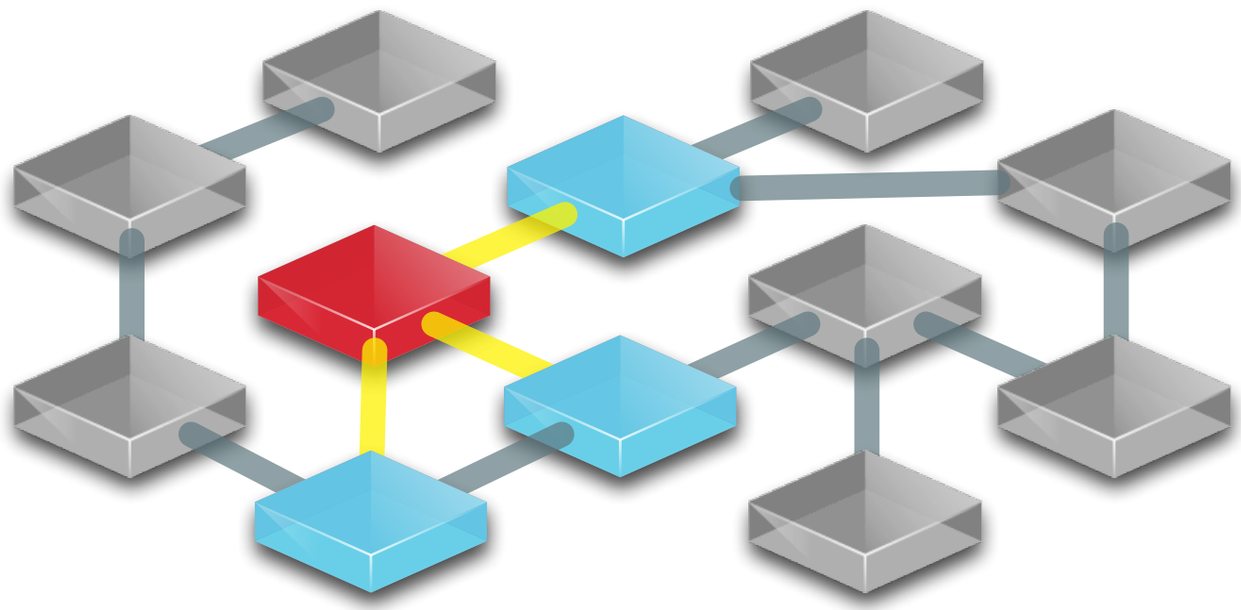


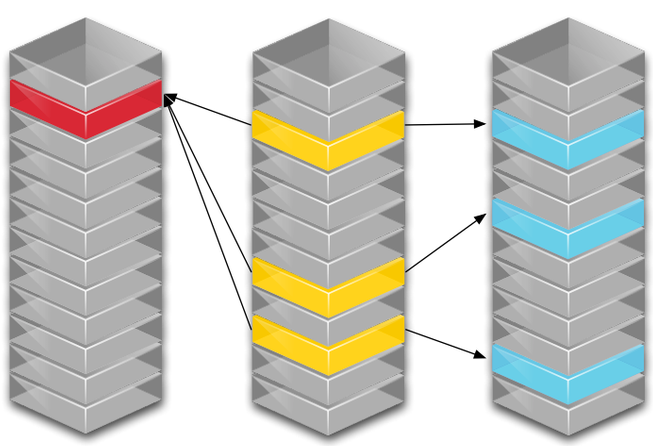


Relational to Graph

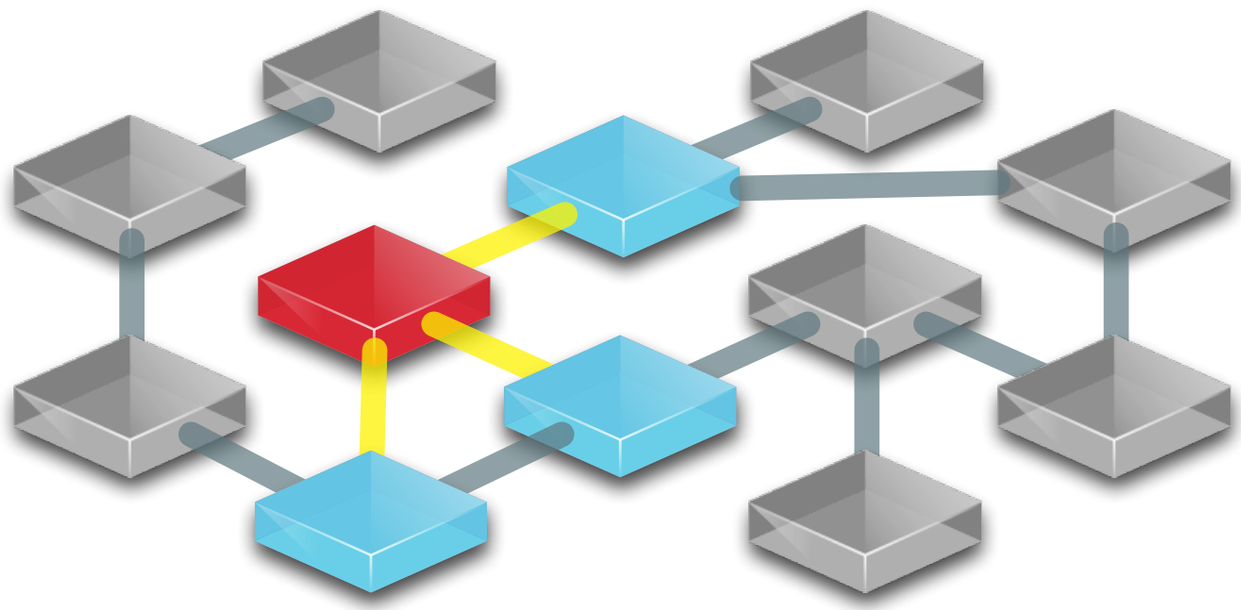
You know relational

now consider relationships...



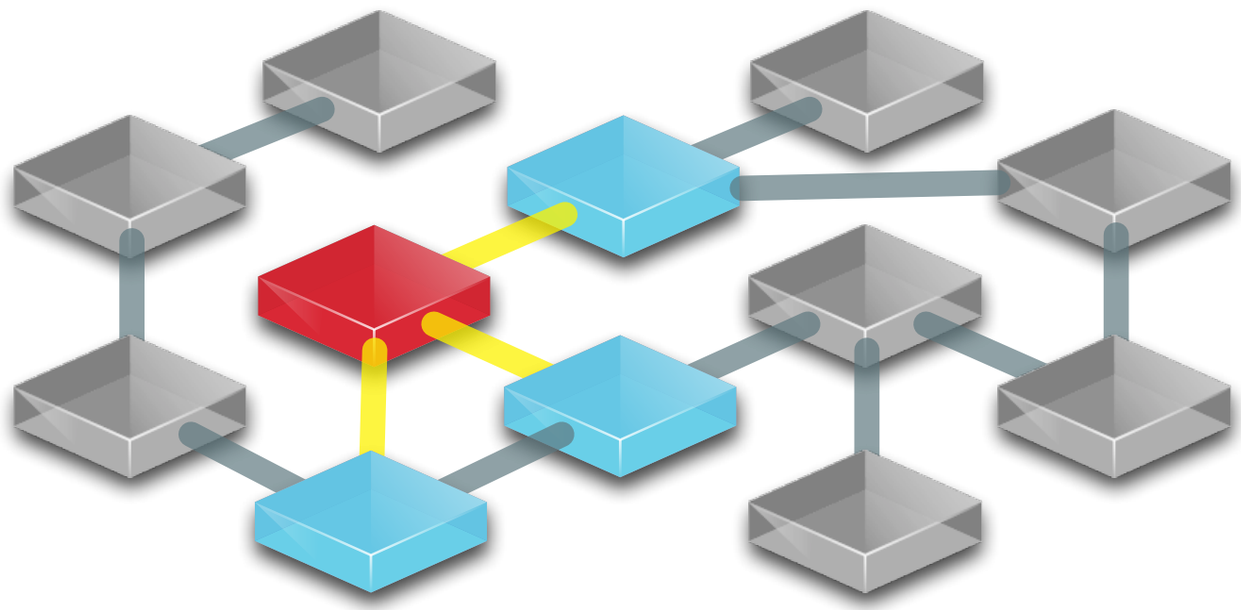


Relational to Graph

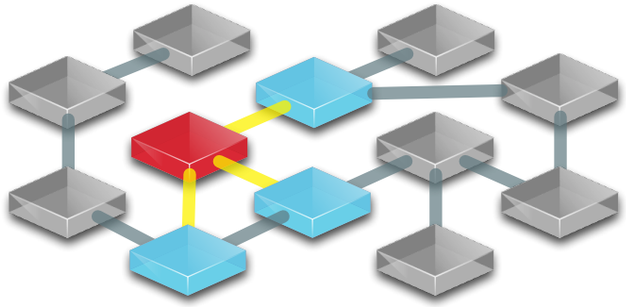


Migration Relational to Graph

- **Entity-Tables** become **Nodes**
- **Foreign Keys** become **Relationships**
- **Link Tables** become **Relationships**
- Remove artificial **Primary Keys and Foreign Keys**

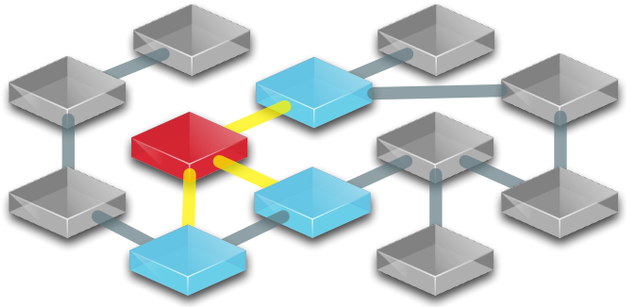


Looks different. Who cares?



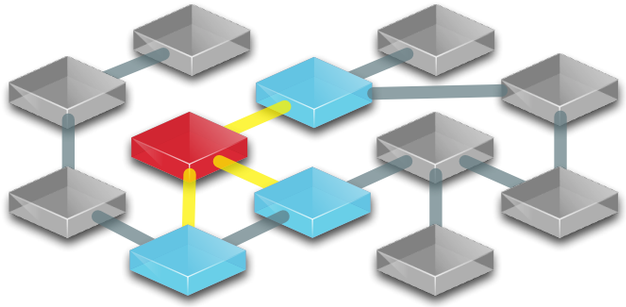
Looks different. Who cares?

- a sample social graph



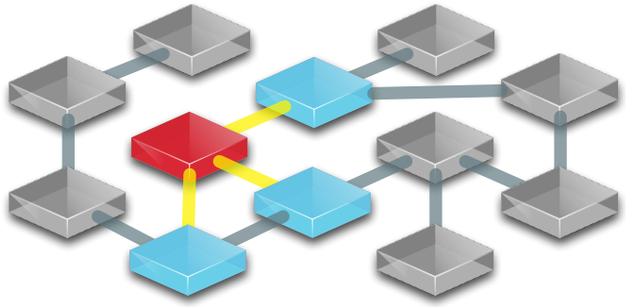
Looks different. Who cares?

- a sample social graph
  - with ~1,000 persons



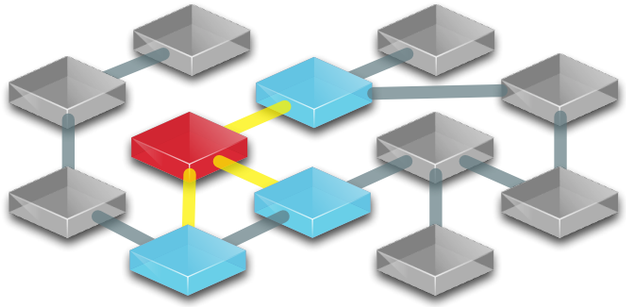
Looks different. Who cares?

- a sample social graph
  - with ~1,000 persons
- average 50 friends per person



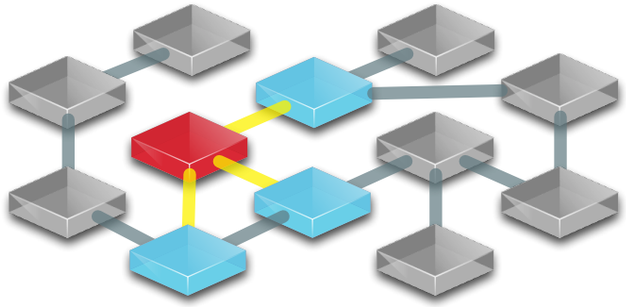
Looks different. Who cares?

- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4



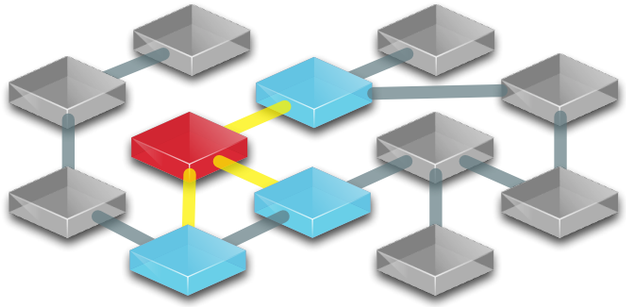
Looks different. Who cares?

- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O



Looks different. Who cares?

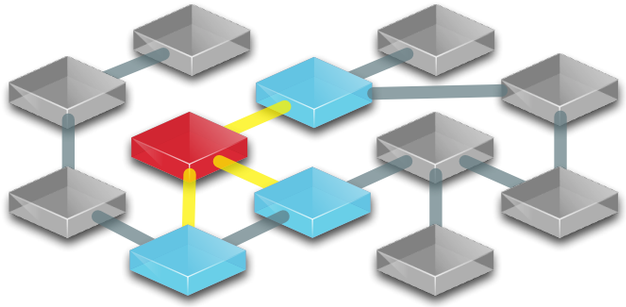
- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O



	# persons	query time
Relational database	1,000	2000ms

Looks different. Who cares?

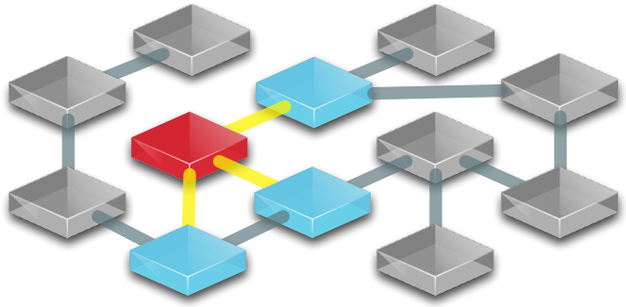
- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O



	# persons	query time
Relational database	1,000	2000ms
Neo4j	1,000	2ms

Looks different. Who cares?

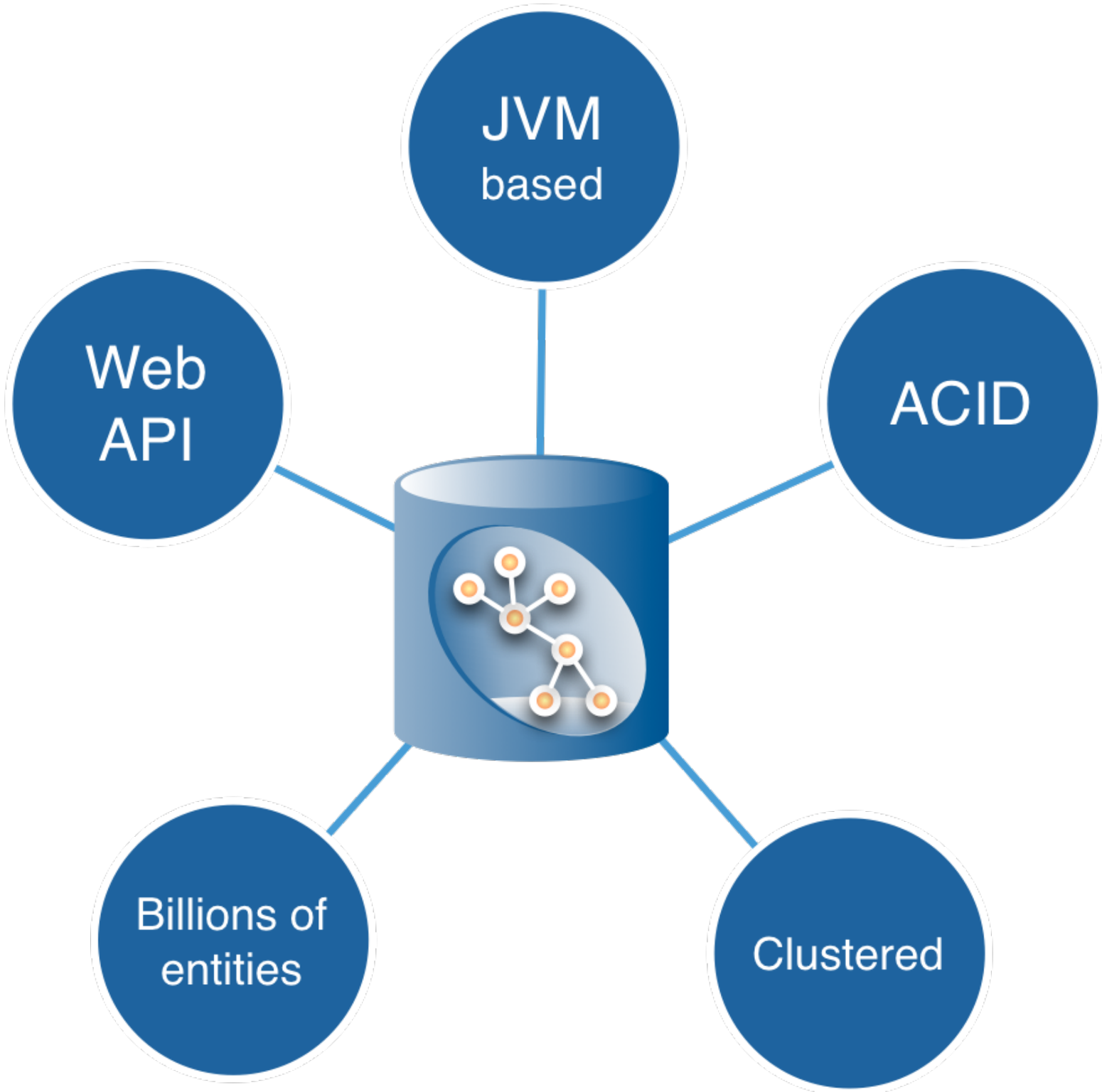
- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O



	# persons	query time
Relational database	1,000	2000ms
Neo4j	1,000	2ms
Neo4j	1,000,000	2ms

Neo4j is a Graph Database

Neo4j is a Graph Database



Neo4j is a Graph Database

Neo4j is a Graph Database

- A **Graph** Database:

Neo4j is a Graph Database

- A **Graph** Database:
    - a schema-free labeled Property Graph

Neo4j is a Graph Database

- A **Graph** Database:
    - a schema-free labeled Property Graph
    - perfect for complex, highly connected data

Neo4j is a Graph Database

- A **Graph** Database:
    - a schema-free labeled Property Graph
    - perfect for complex, highly connected data
- A Graph **Database**:

Neo4j is a Graph Database

- A **Graph** Database:
  - a schema-free labeled Property Graph
  - perfect for complex, highly connected data
- A Graph **Database**:
  - reliable with real ACID Transactions

Neo4j is a Graph Database

- A **Graph** Database:
  - a schema-free labeled Property Graph
  - perfect for complex, highly connected data
- A Graph **Database**:
  - reliable with real ACID Transactions
  - scalable: Billions of Nodes and Relationships, Scale out with highly available Neo4j-Cluster

Neo4j is a Graph Database

- A **Graph** Database:
  - a schema-free labeled Property Graph
  - perfect for complex, highly connected data
- A Graph **Database**:
  - reliable with real ACID Transactions
  - scalable: Billions of Nodes and Relationships, Scale out with highly available Neo4j-Cluster
  - fast with more than 2-4M traversals / second per core

Neo4j is a Graph Database

- A **Graph** Database:
  - a schema-free labeled Property Graph
  - perfect for complex, highly connected data
- A Graph **Database**:
  - reliable with real ACID Transactions
  - scalable: Billions of Nodes and Relationships, Scale out with highly available Neo4j-Cluster
  - fast with more than 2-4M traversals / second per core
  - Server with HTTP API, or Embeddable on the JVM

Neo4j is a Graph Database

- A **Graph** Database:
  - a schema-free labeled Property Graph
  - perfect for complex, highly connected data
- A Graph **Database**:
  - reliable with real ACID Transactions
  - scalable: Billions of Nodes and Relationships, Scale out with highly available Neo4j-Cluster
  - fast with more than 2-4M traversals / second per core
  - Server with HTTP API, or Embeddable on the JVM
  - Declarative Query Language

Graph Database: Why ?

- Powerful data model, as general as RDBMS
- Whiteboard friendly, agile development
- Fast, for connected data
- Easy to query

Graph Database: Why Not?

- Global Queries / Number Crunching
- Binary Data / Blobs
- Requires conceptual shift
  - graph-like thinking is ***addictive***

ACID Transactions.

Why?

Consistency Model

CAP

Consistency Model

C A P

C A P *(single, master)*

Consistency Model

C A P

C A P *(single, master)*

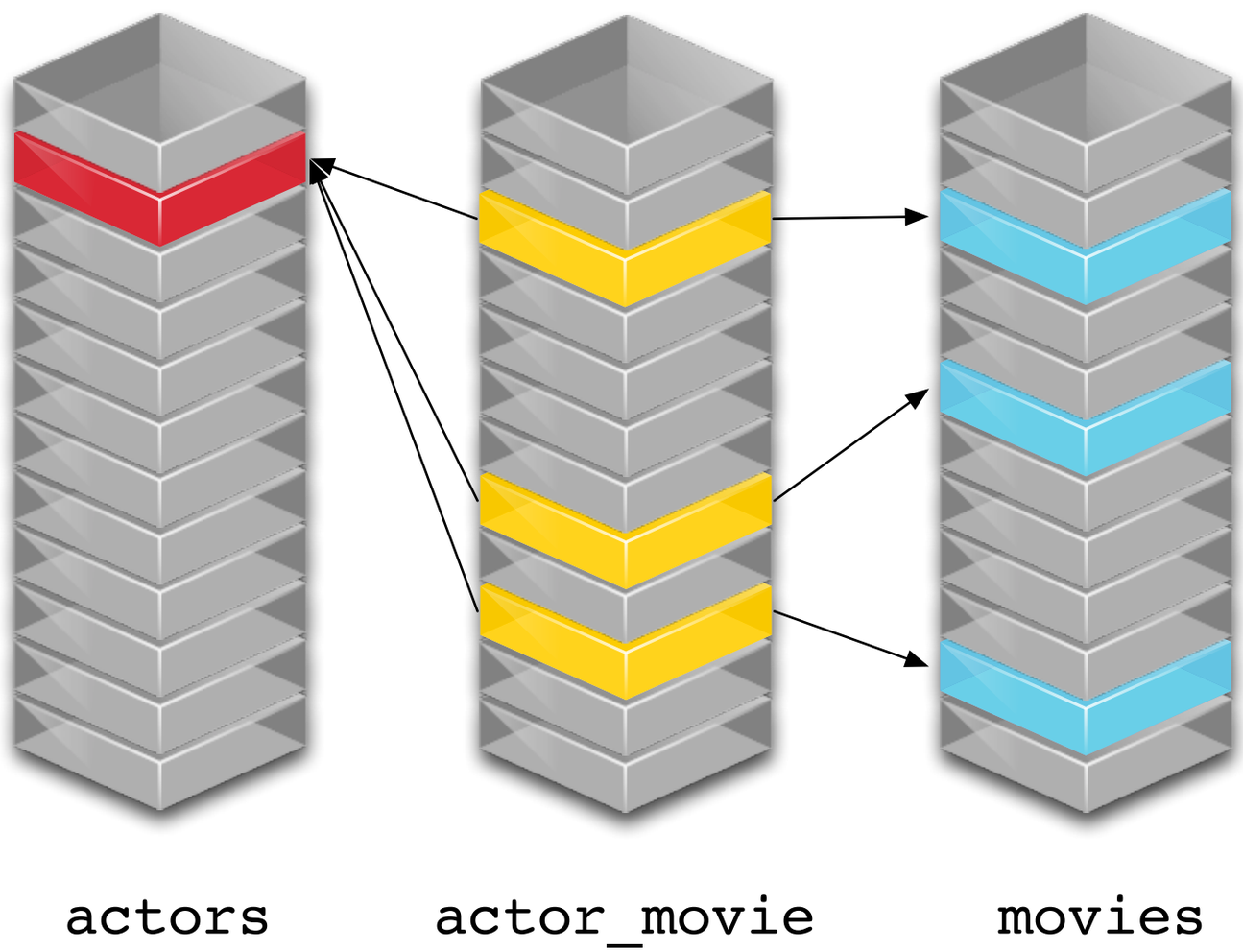
C **A P** *(cluster)*

Graph Querying

You know how to query a Relational Database?

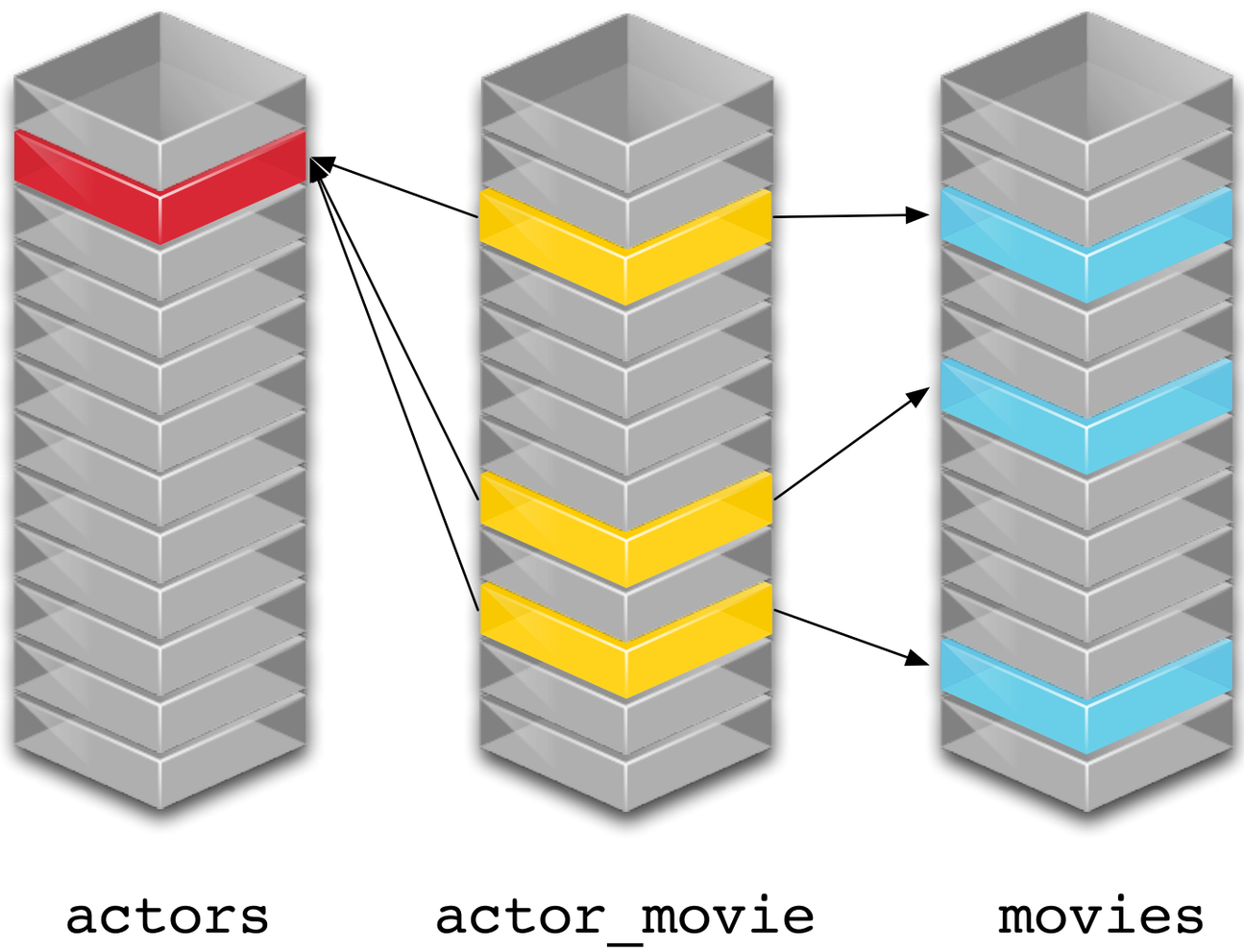
Just use SQL

Just use SQL



Just use SQL

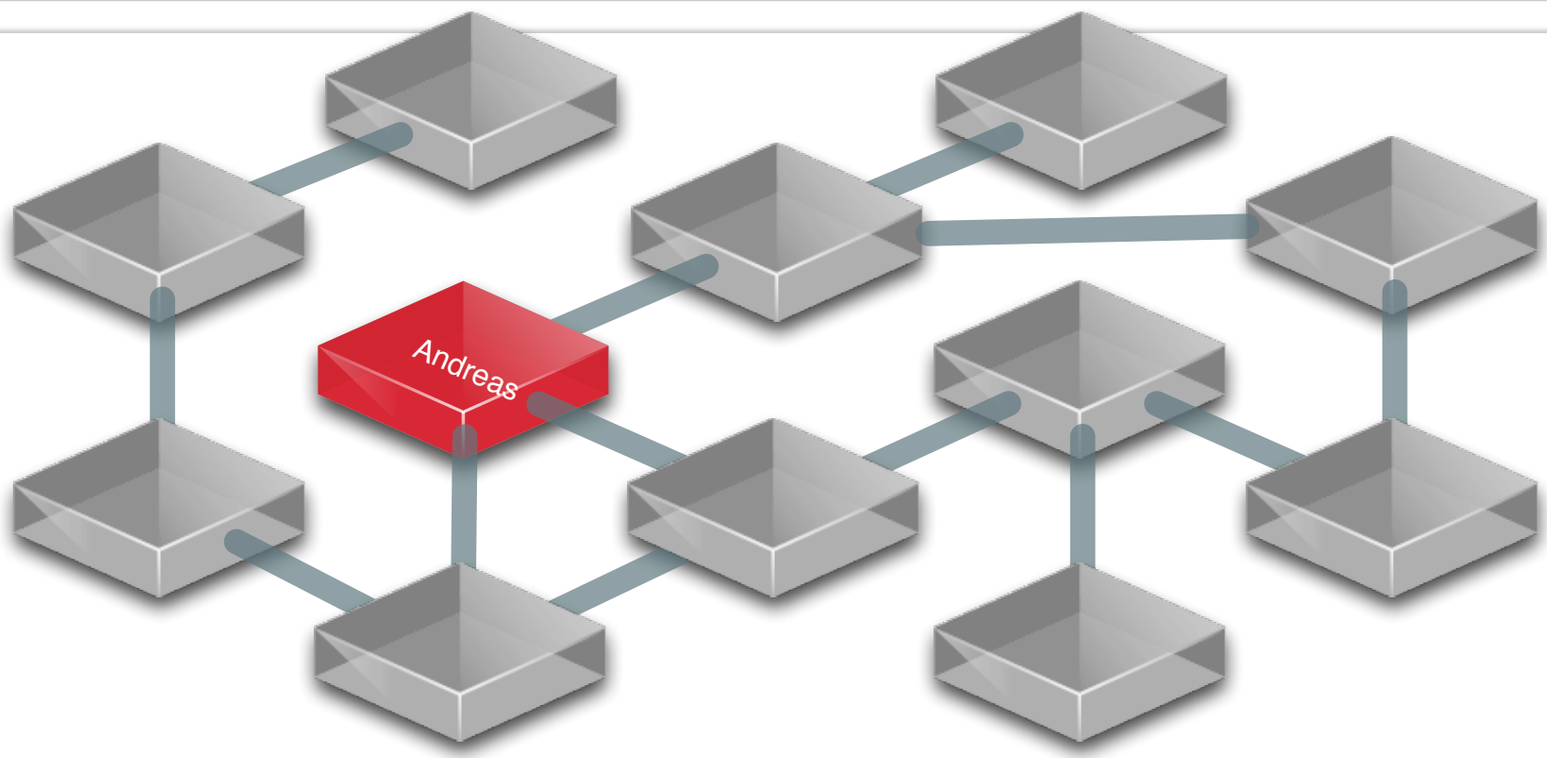
```
select movie.title
from actors join actor_movie on (...) join movies on (...)
where actors.name = "Andreas"
```



How to query a graph?

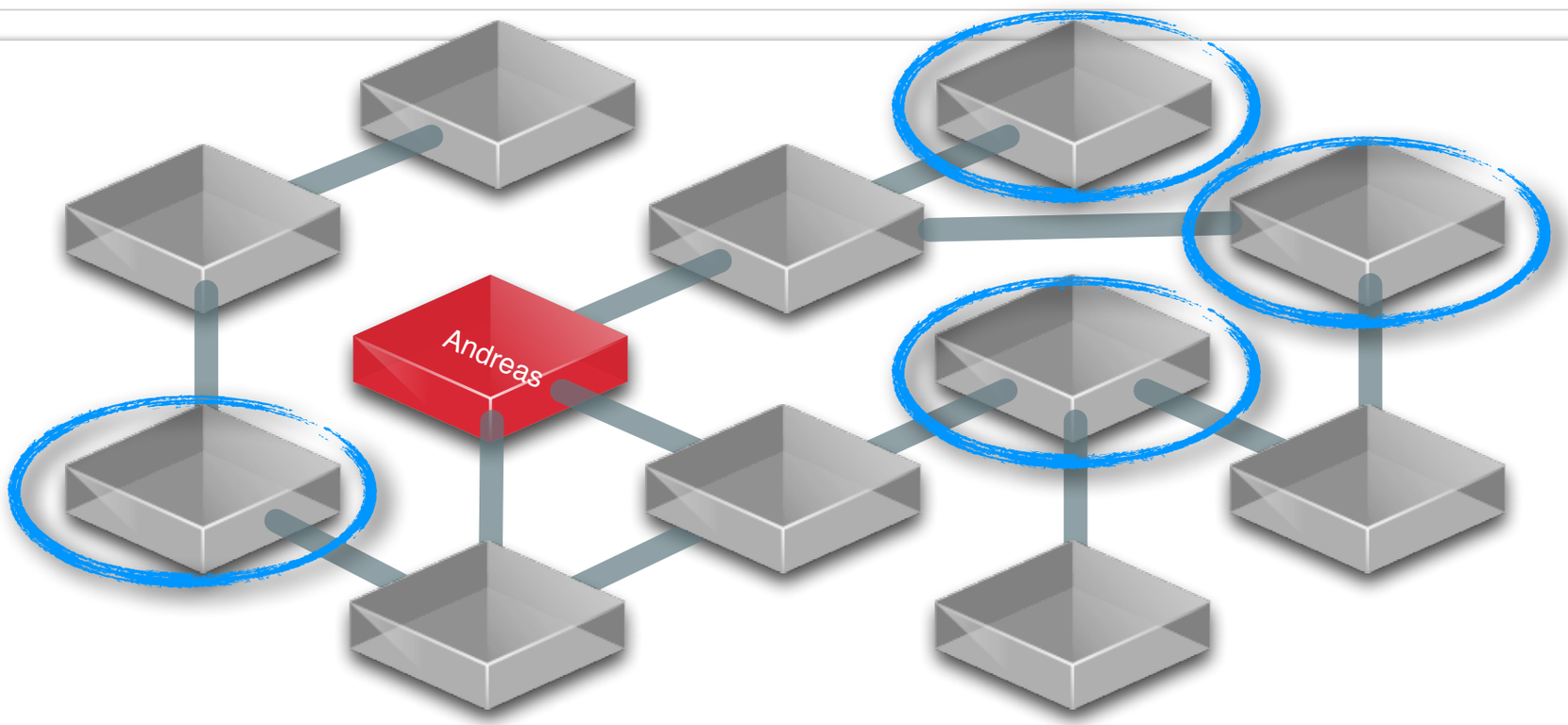
You traverse the graph

```
// find starting points
MATCH (:Person {name:'Andreas'})
```



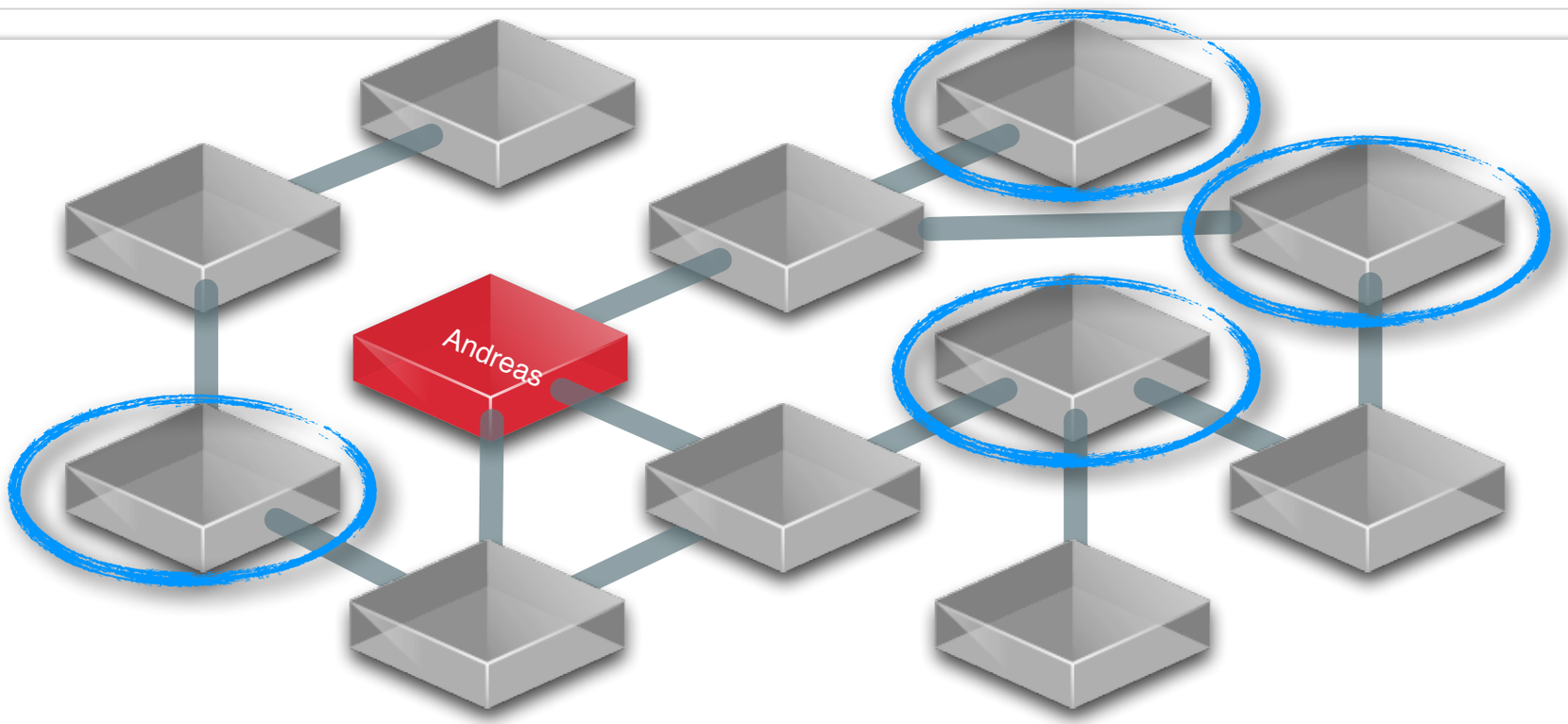
You traverse the graph

```
// find starting points
MATCH (:Person {name:'Andreas'})
// then traverse the relationships
      -[:ACTED_IN]->(movie:Movie)
     <-[:ACTED_IN]- (other:Person)
```



You traverse the graph

```
// find starting points
MATCH (:Person {name:'Andreas'})
// then traverse the relationships
      -[:ACTED_IN]->(movie:Movie)
     <-[:ACTED_IN]- (other:Person)
// and return results
RETURN other
```



(open)Cypher

a pattern-matching query language for graphs

Cypher attributes

#1 Declarative

You tell Cypher what you want, not how to get it

Cypher attributes

#2 Expressive

Optimize syntax for reading

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
RETURN a.name, r.role, m.title
```

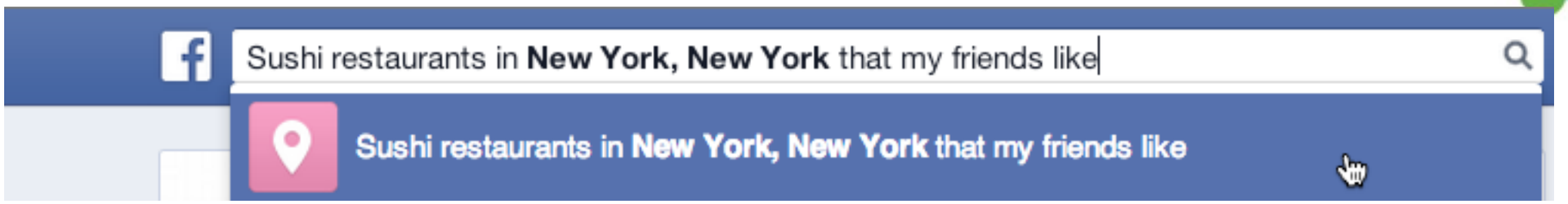
Cypher attributes

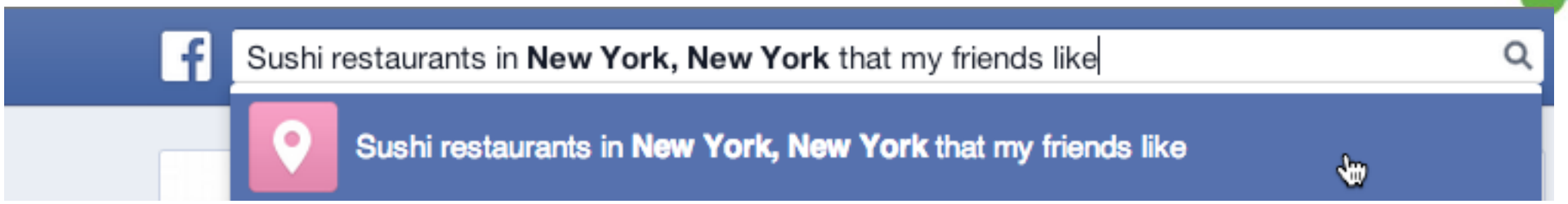
#3 Pattern Matching

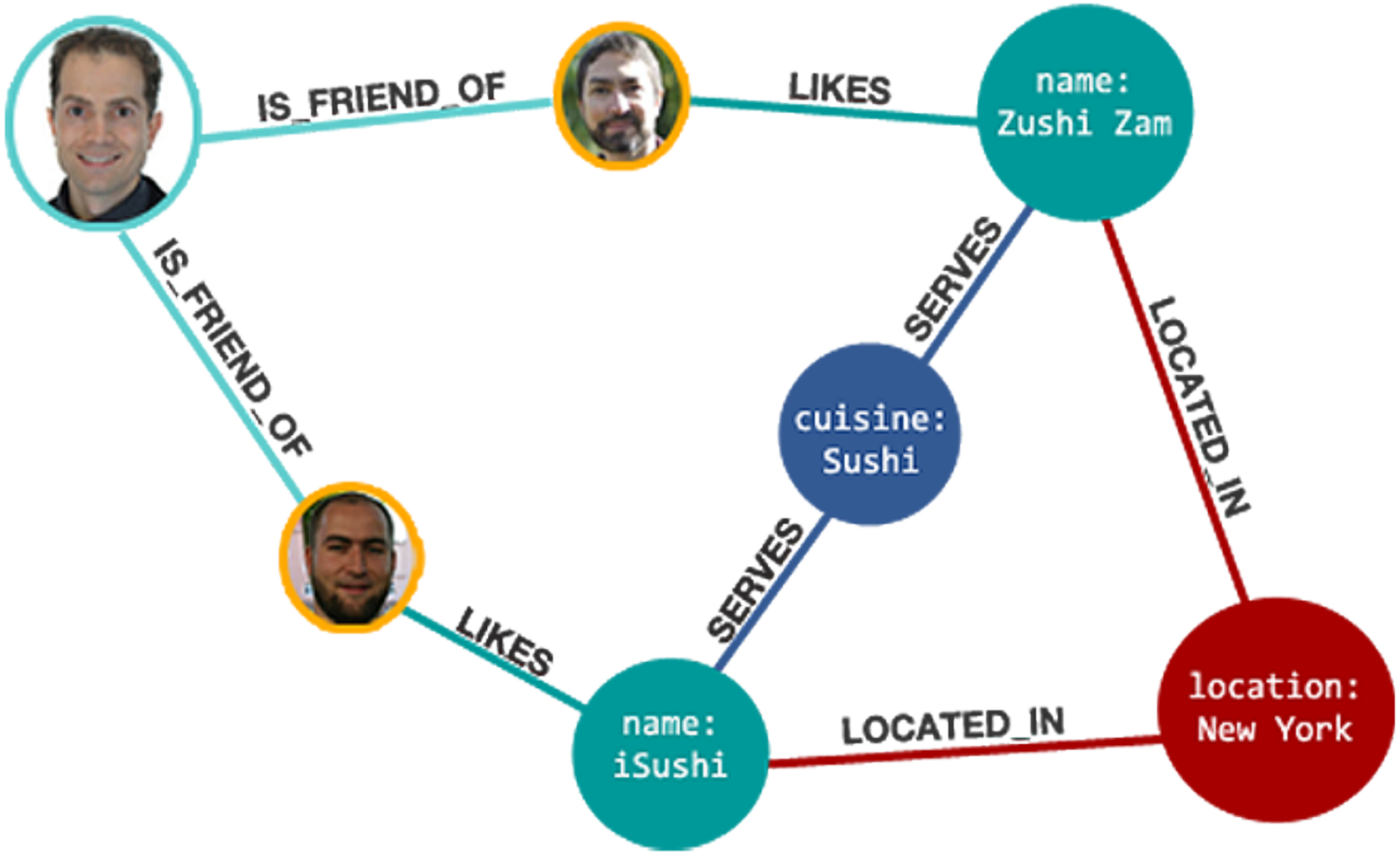
Patterns are easy for your human brain

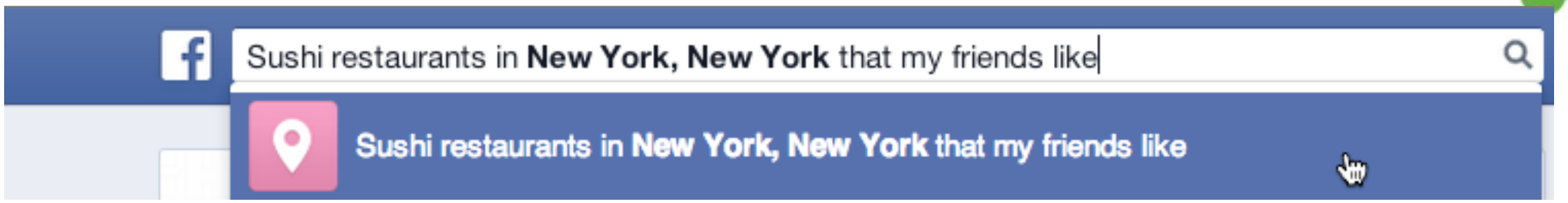
Graph Query Examples

Social Recommendation





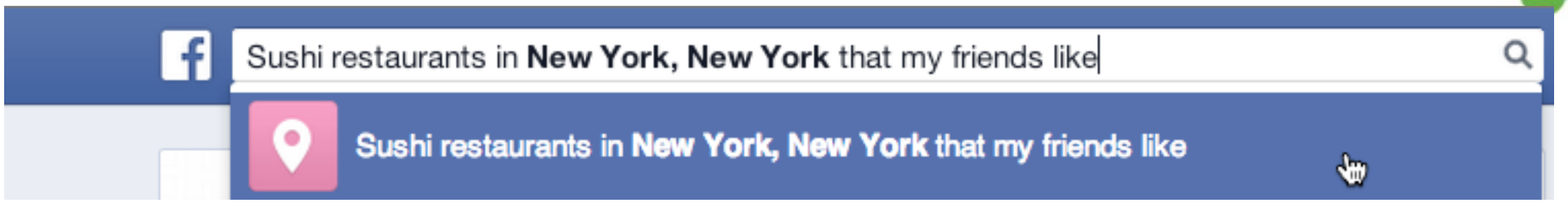


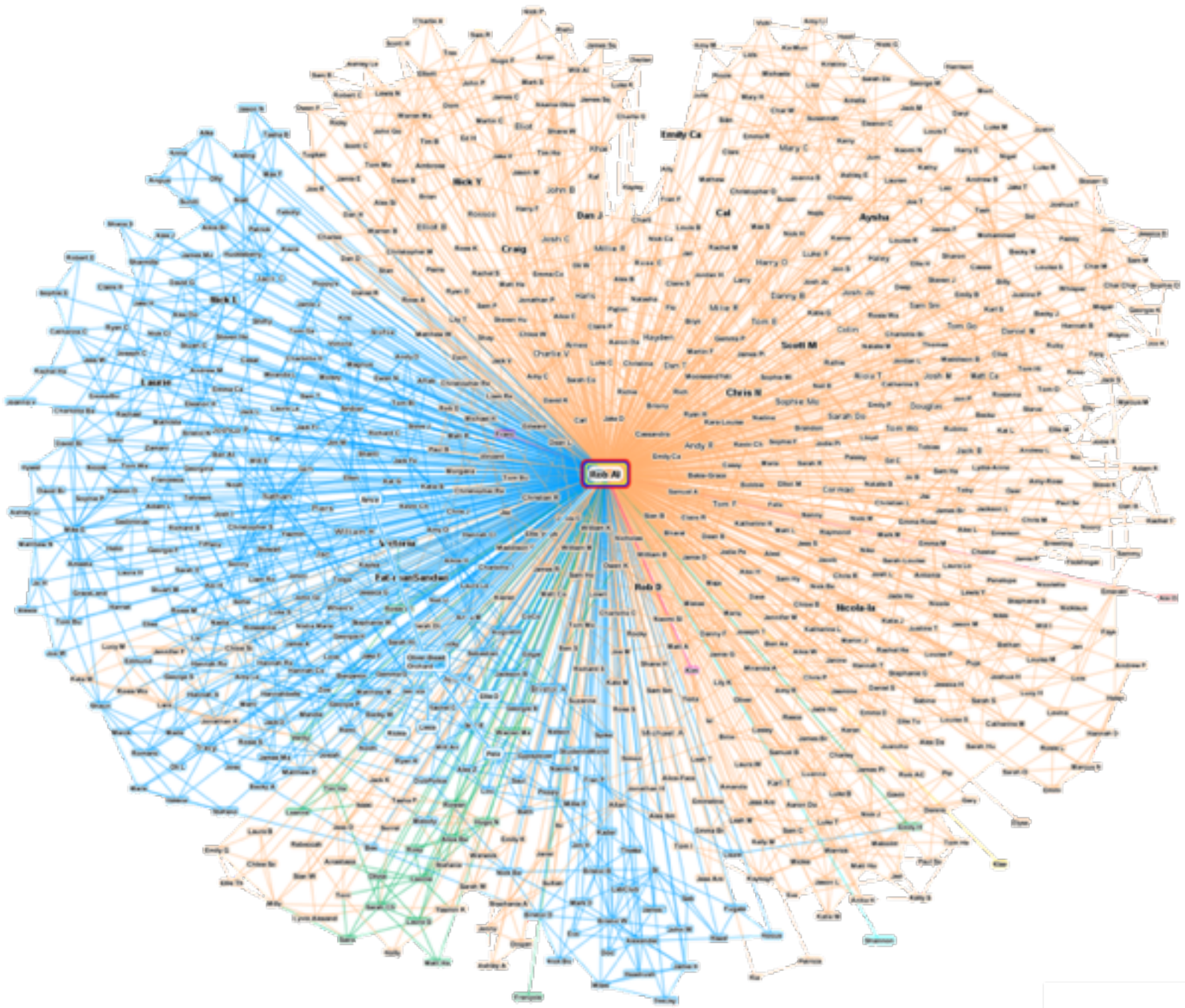


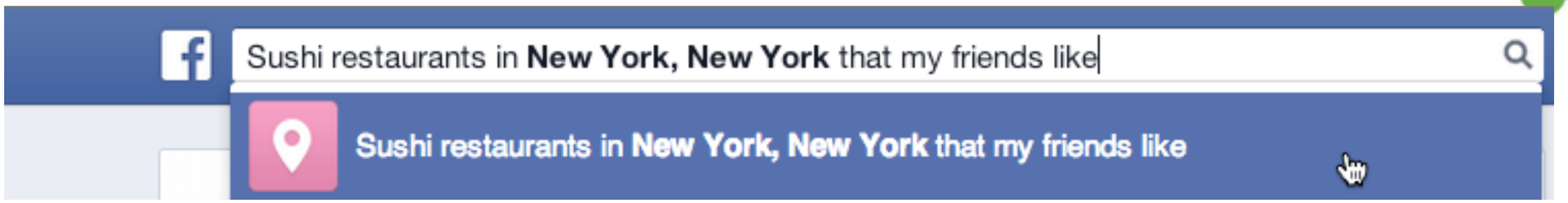

```
MATCH (person:Person)-[:IS_FRIEND_OF]->(friend),
      (friend)-[:LIKES]->(restaurant),
      (restaurant)-[:LOCATED_IN]->(loc:Location),
      (restaurant)-[:SERVES]->(type:Cuisine)

WHERE person.name = 'Philip' AND loc.location='New York' AND
      type.cuisine='Sushi'

RETURN restaurant.name
```

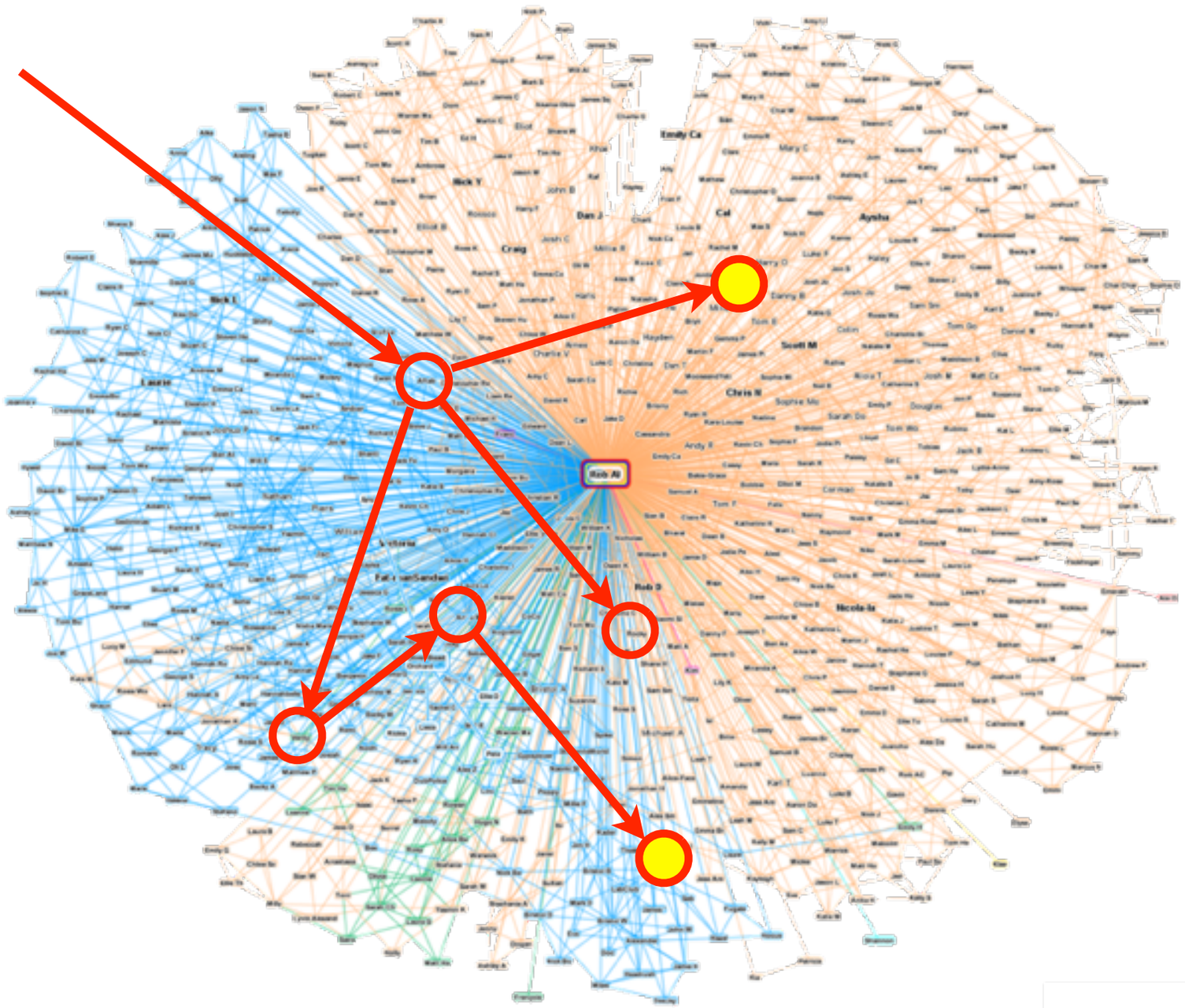
http://maxdemarzi.com/?s=facebook

** Cypher query language example*









Query Structure

Our Question!

```
Find all Actors and Movies they acted in
Whose name starts with "T"
Aggregate their activity
          and movie titles
Who acted in more than 5 movies
Return their name, birth-year and movie-
titles
Ordered by number of movies
Limited to top 10
```

In SQL

```
SELECT a.name, a.born,
       group_concat(m.title) as movies,
       count(*) as cnt
FROM   actors as a JOIN actor_movie
  ON  (a.id = actor_movie.actor_id)
JOIN   movies as m
  ON  (actor_movie.movie_id = m.id)
WHERE  a.name LIKE "T%"
GROUP BY a.name, a.born
HAVING cnt > 5
ORDER BY cnt DESC
```

Demo
Neo4j Browser

Cypher Query Structure

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

Breakdown

MATCH

describes the pattern

MATCH - Pattern

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

WHERE

filters the result set

WHERE - filter

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

WITH

computes intermediate results

WITH - intermediate projection

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

RETURN

returns the final results

RETURN - results

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

ORDER BY
LIMIT SKIP

sort and paginate

ORDER BY LIMIT - Paginate

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

Collections

powerful datastructure handling

Collections

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
     collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

More Cypher

UPDATE OPERATIONS

create & update nodes, relationships and patterns

CREATE

creates nodes, relationships and patterns

CREATE - nodes, rels, structures

```
CREATE (y:Year {year:2016})
FOREACH (m IN range(1,12) |
  CREATE
   (:Month {month:m})-[:IN]->(y)
)
```

MERGE

matches or creates

MERGE - get or create

```
MERGE (y:Year {year:2016})
ON CREATE
  SET y.created = timestamp()
FOREACH (m IN range(1,12) |
  MERGE
    (:Month {month:m})-[:IN]->(y)
)
```

SET, REMOVE

update attributes and labels

SET, REMOVE, DELETE

```
MATCH (year:Year)
WHERE year.year % 4 = 0 OR
      year.year % 100 <> 0 AND
        year.year % 400 = 0
SET year:Leap
WITH year
MATCH (year)<-[:IN]-(feb:Month {month:2})
SET feb.days = 29
CREATE (feb)<-[:IN]-(:Day {day:29})
```

INDEX, CONSTRAINTS

represent optional schema

INDEX / CONSTRAINT

```
CREATE INDEX ON :Month(month)

CREATE CONSTRAINT ON (y:Year)
   ASSERT y.year IS UNIQUE

CREATE CONSTRAINT ON (p:Person)
   ASSERT exists(p.name)
```

Demo
Neo4j Browser

Neo4j APIs

Cypher Query Language

Transactional via HTTP

binary via bolt

Neo4j 3.0

Java embedded
e.g for Neo4j Extensions

Apache Tinkerpop 3.x
Gremlin

Data In

LOAD CSV

from URL & Cypher

up to 0.5m rec/s

LOAD JSON

with Script & Cypher

Your Importer

any language

Cypher / API

Data Generator

LDBC

CSV Parallel Batch Import

1m rec/s for billions of rows

Data Out

CYPHER

via HTTP

ALL LANGUAGE

with Drivers

Visualization

JavaScript - D3 / Sigma

Tool Integration

via JDBC

Spark Integration

via MazeRunner

Resources on neo4j.com

community

Graph Academy
Learn. Graph. Deploy.

On-site Training

BOOKS

Documentation

GraphGist

Built-in Guides

Online Training

Thank You

Time for Questions!