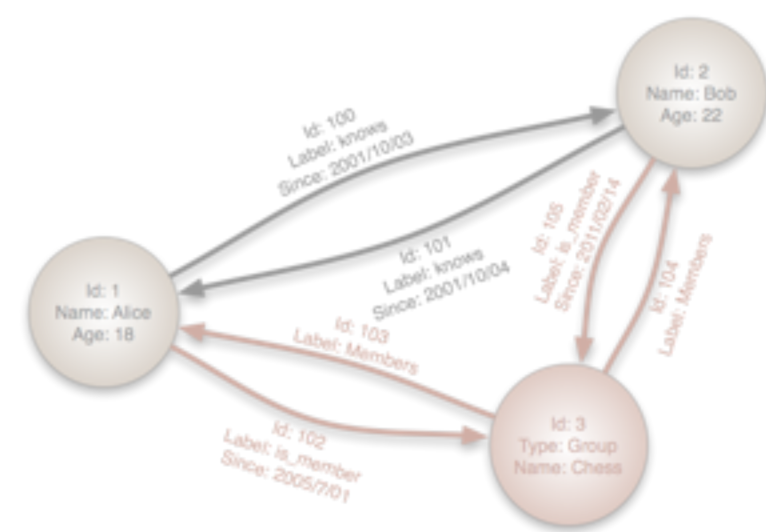
Graph Database Introduction

Meetup
Juni 2014

Michael Hunger
michael@neotechnology.com
@mesirii
@neo4j



Agenda

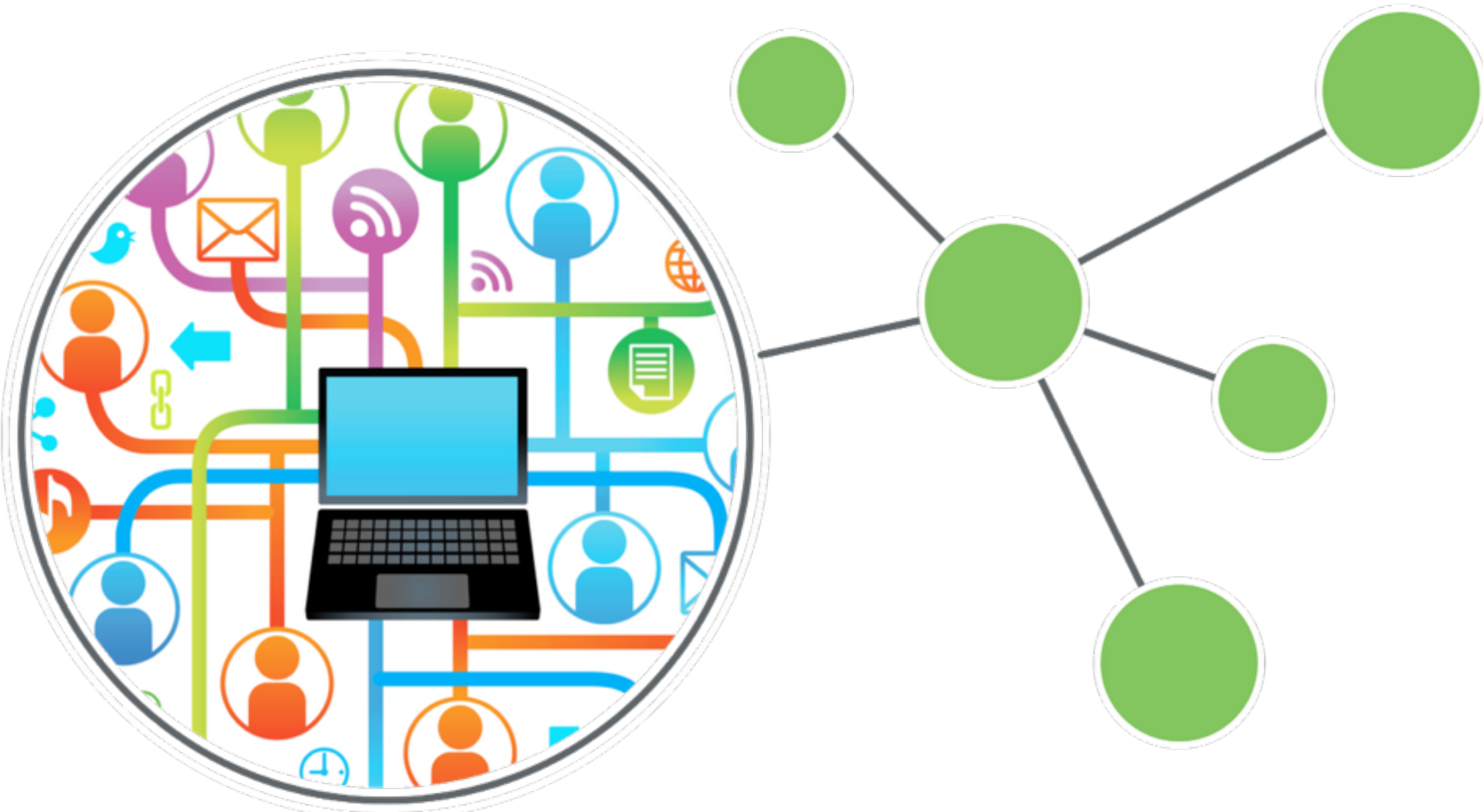
1. Why Graphs, Why Now?
2. What Is A Graph, Anyway?
3. Graphs In The Real World
4. The Graph Landscape
   i) Popular Graph Models
   ii) Graph Databases
   iii) Graph Compute Engines

Why Graphs?

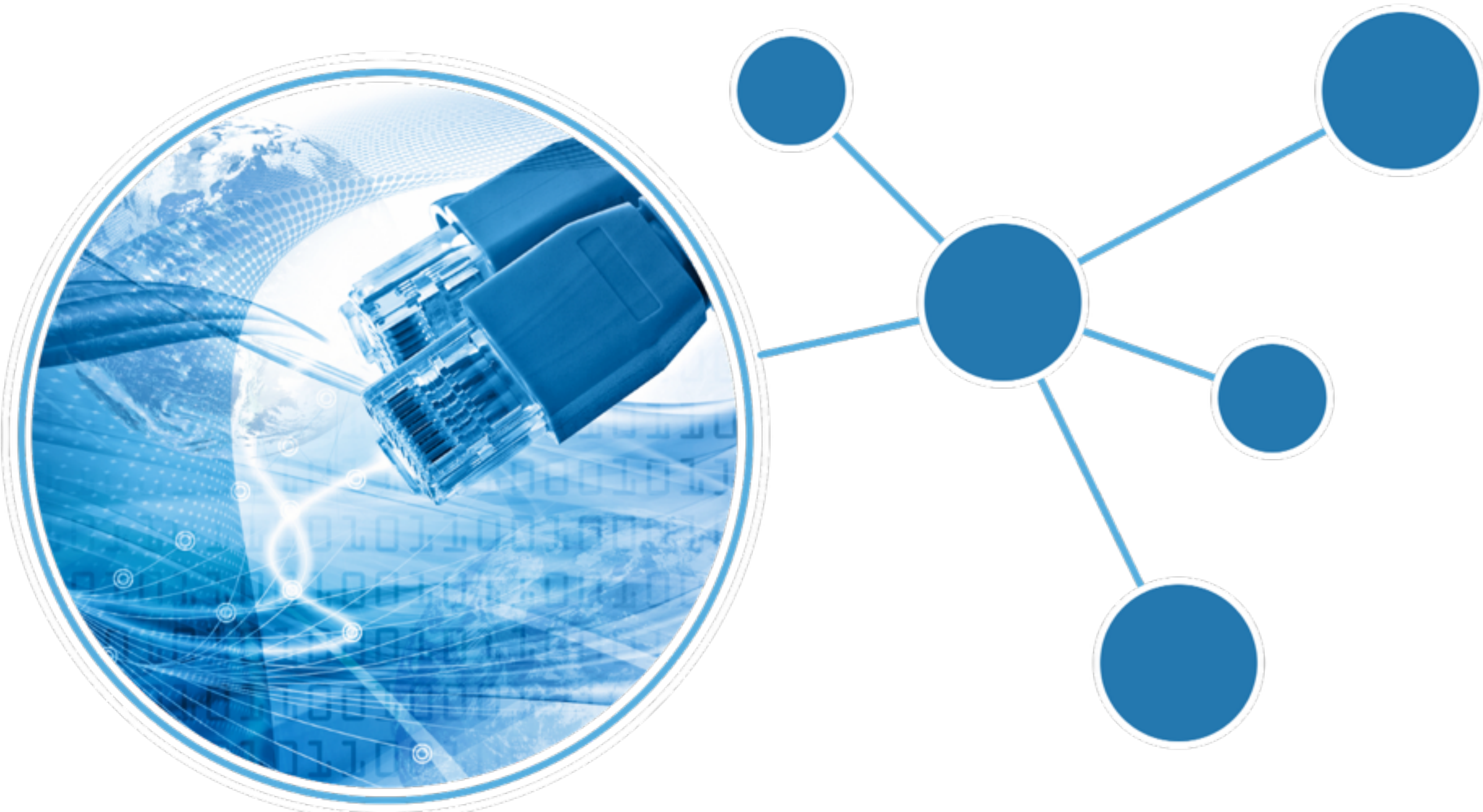
The World is a Graph

Some Use-Cases

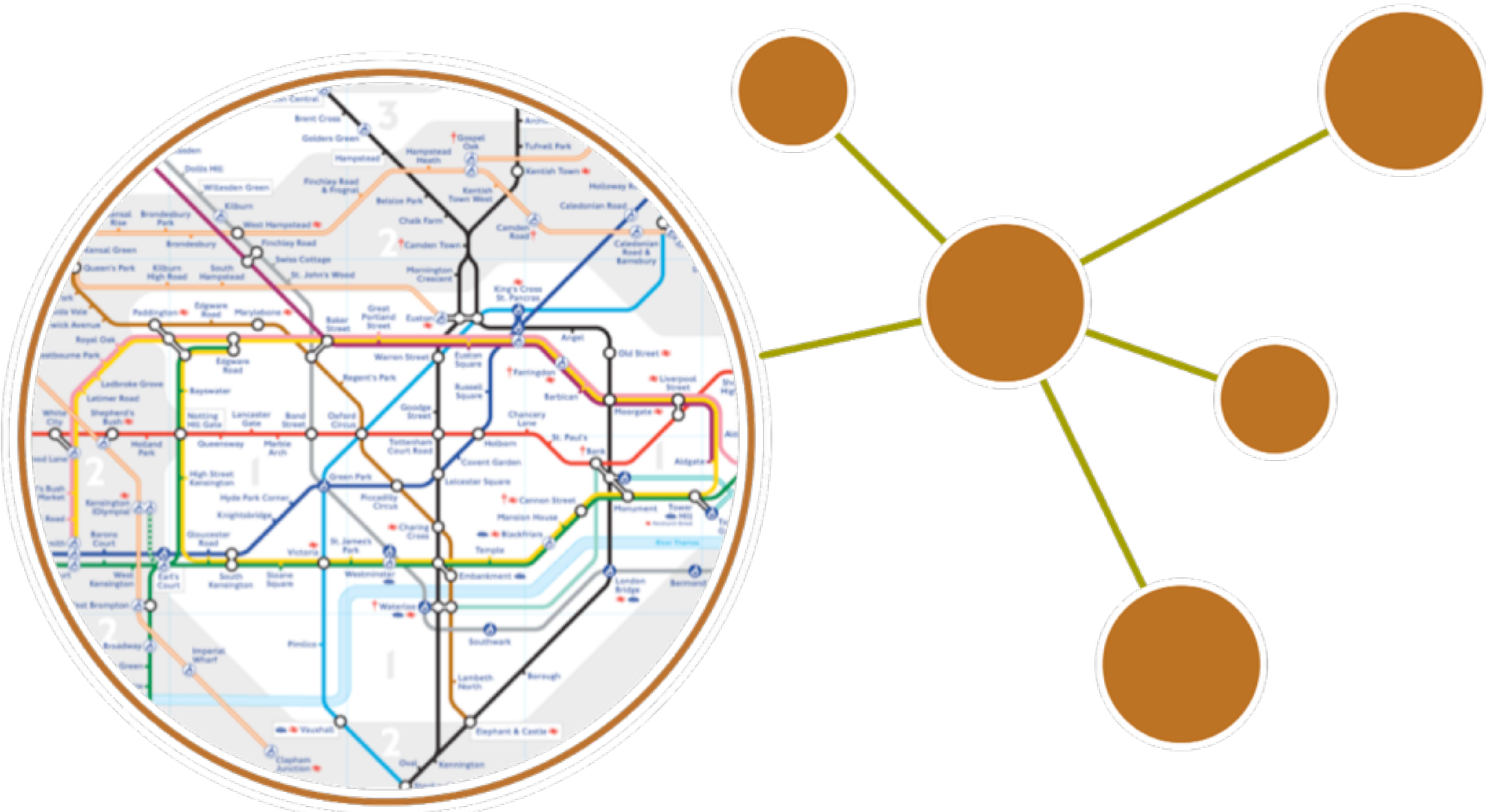
Social Network



(Network) Impact Analysis



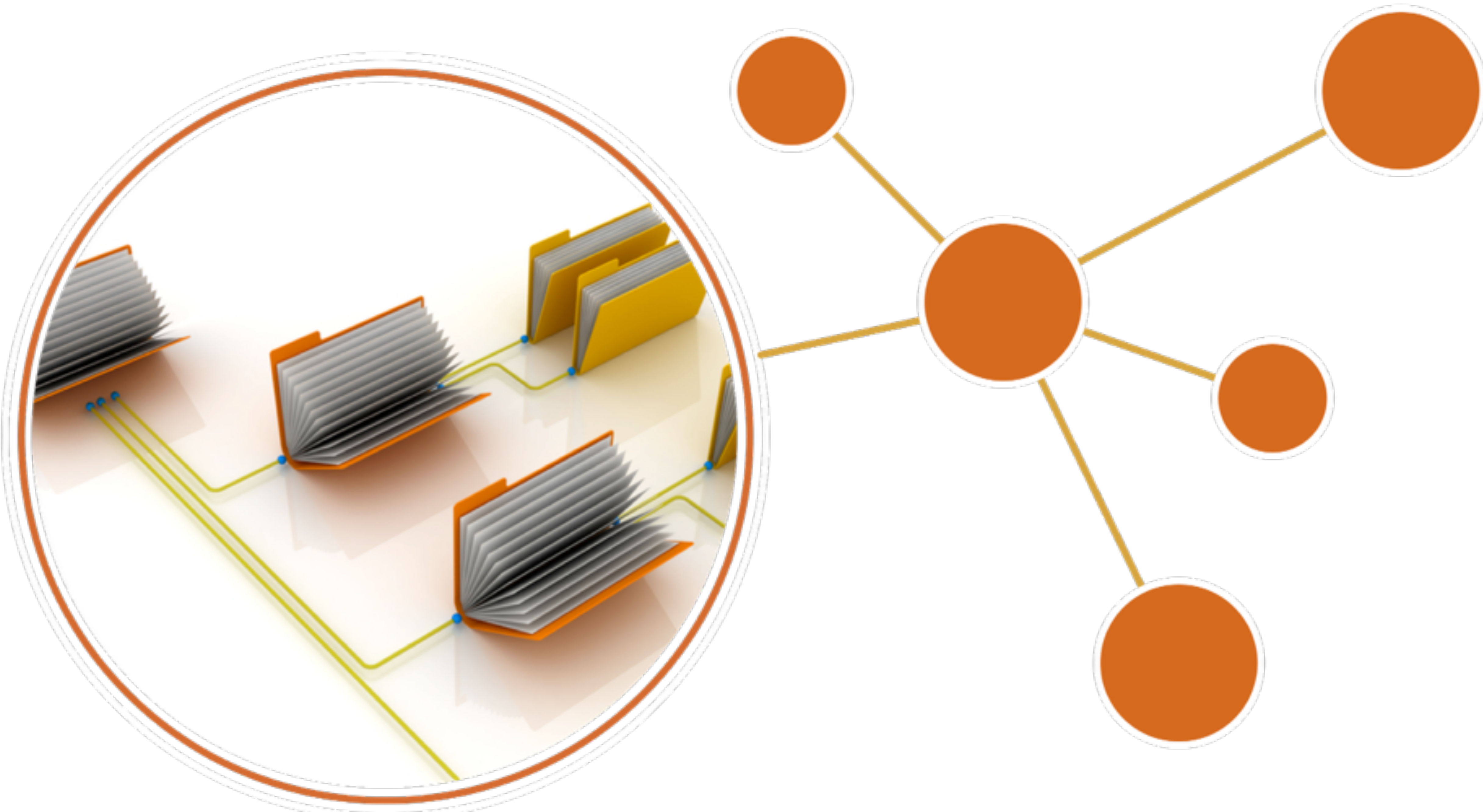
Route Finding



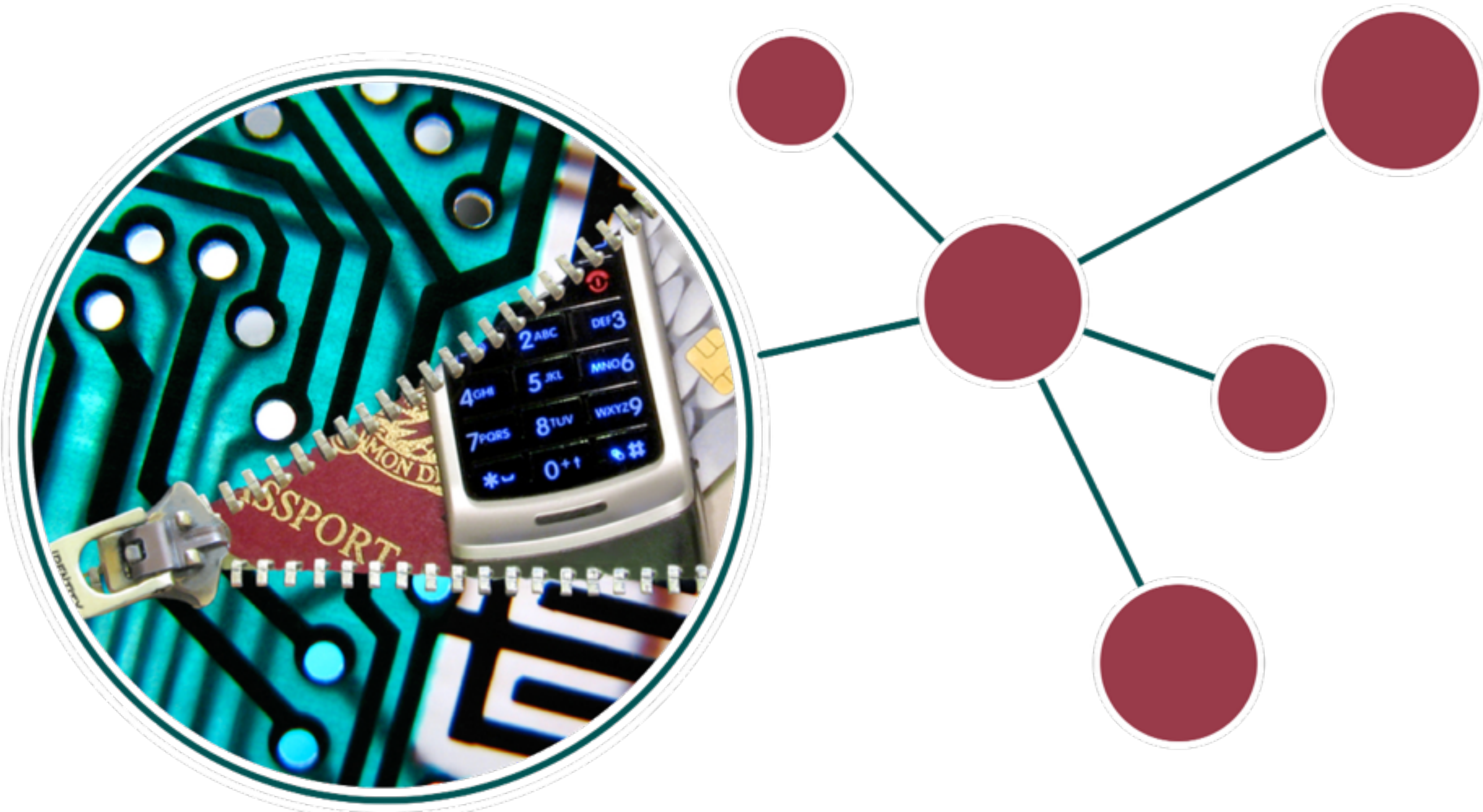
Recommendations

Logistics

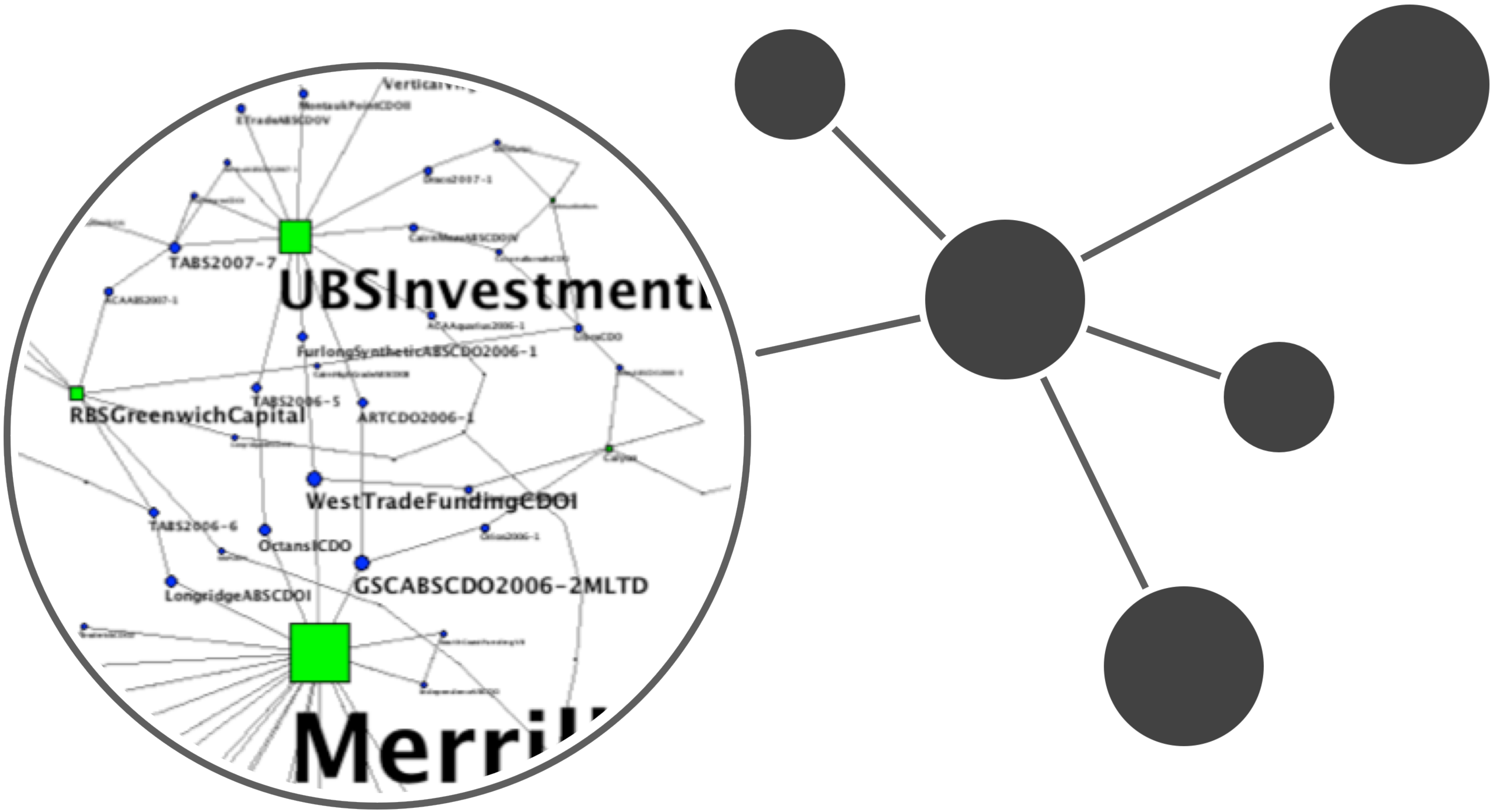
Access Control



Fraud Analysis

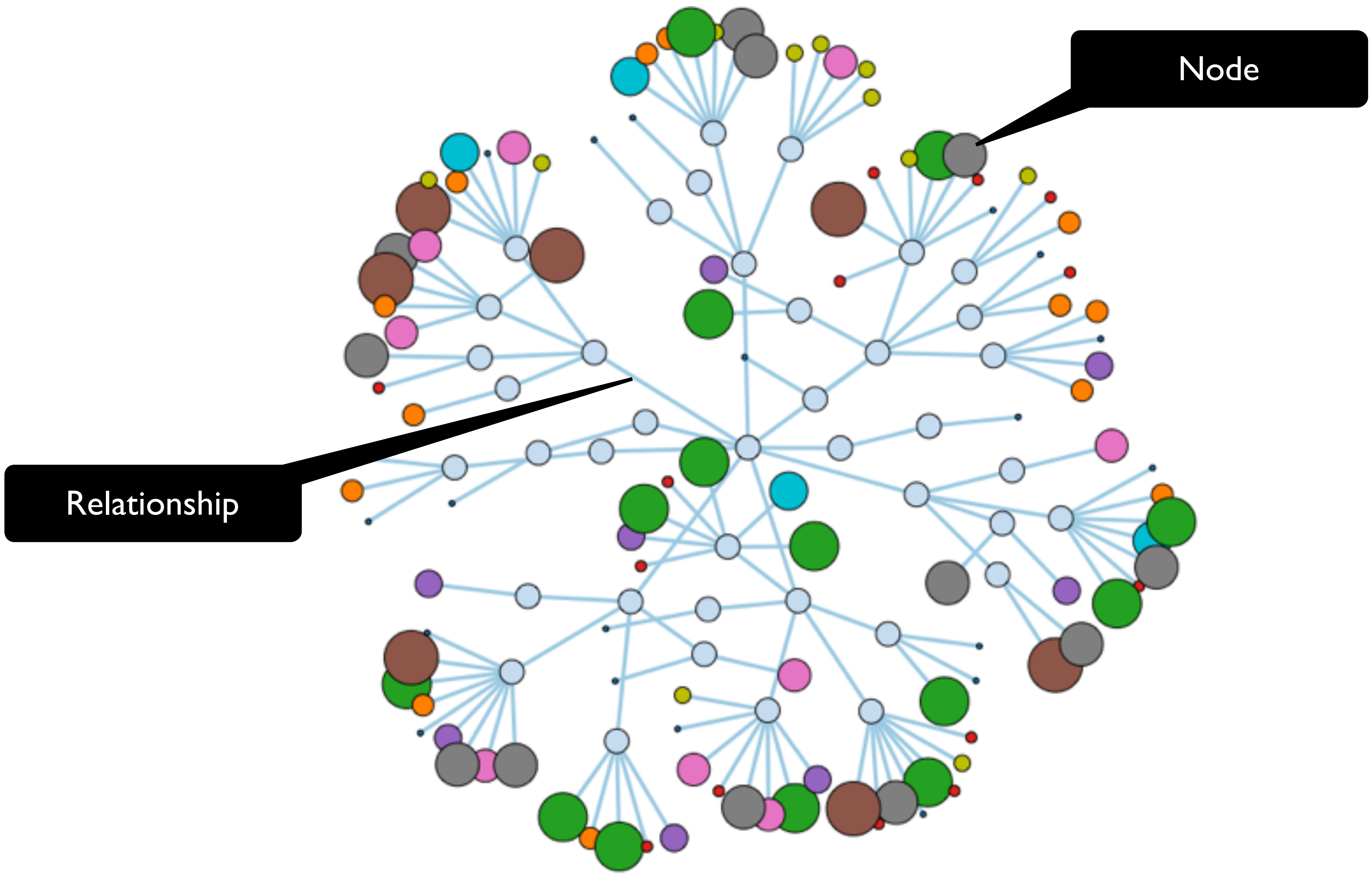


Securities & Debt



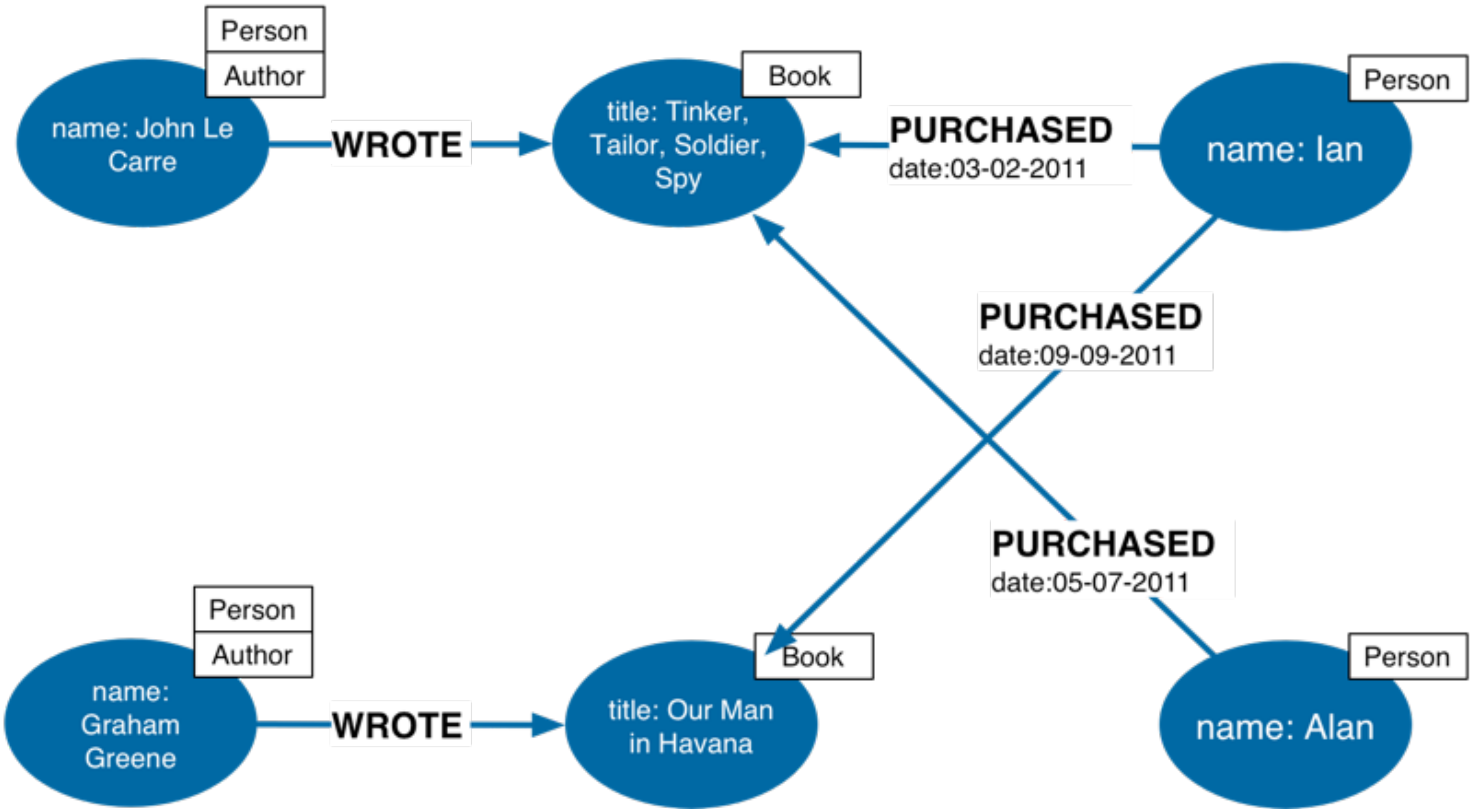
What Is A Graph, Anyway?

A Graph

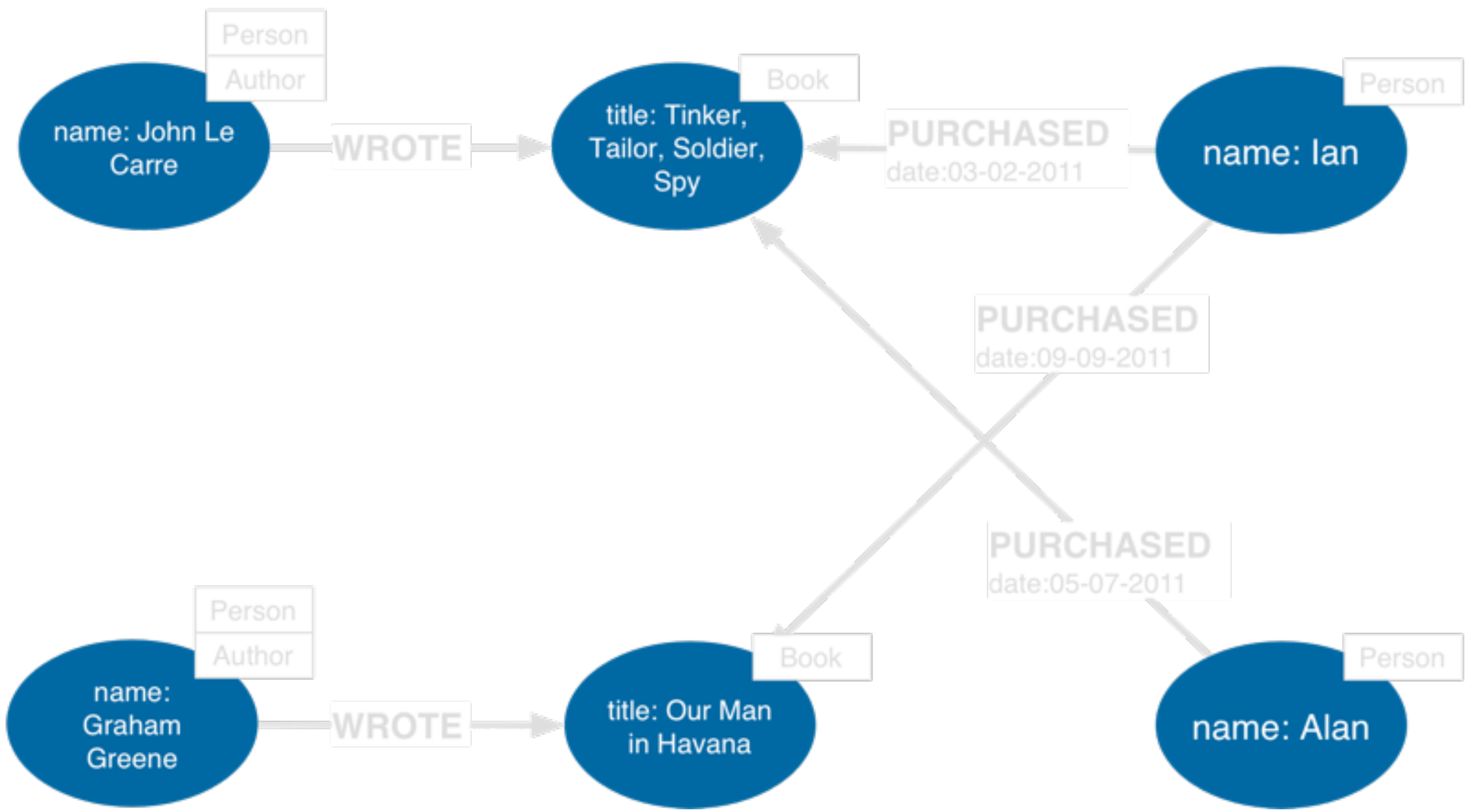


Four Graph Model Building Blocks

Property Graph Data Model

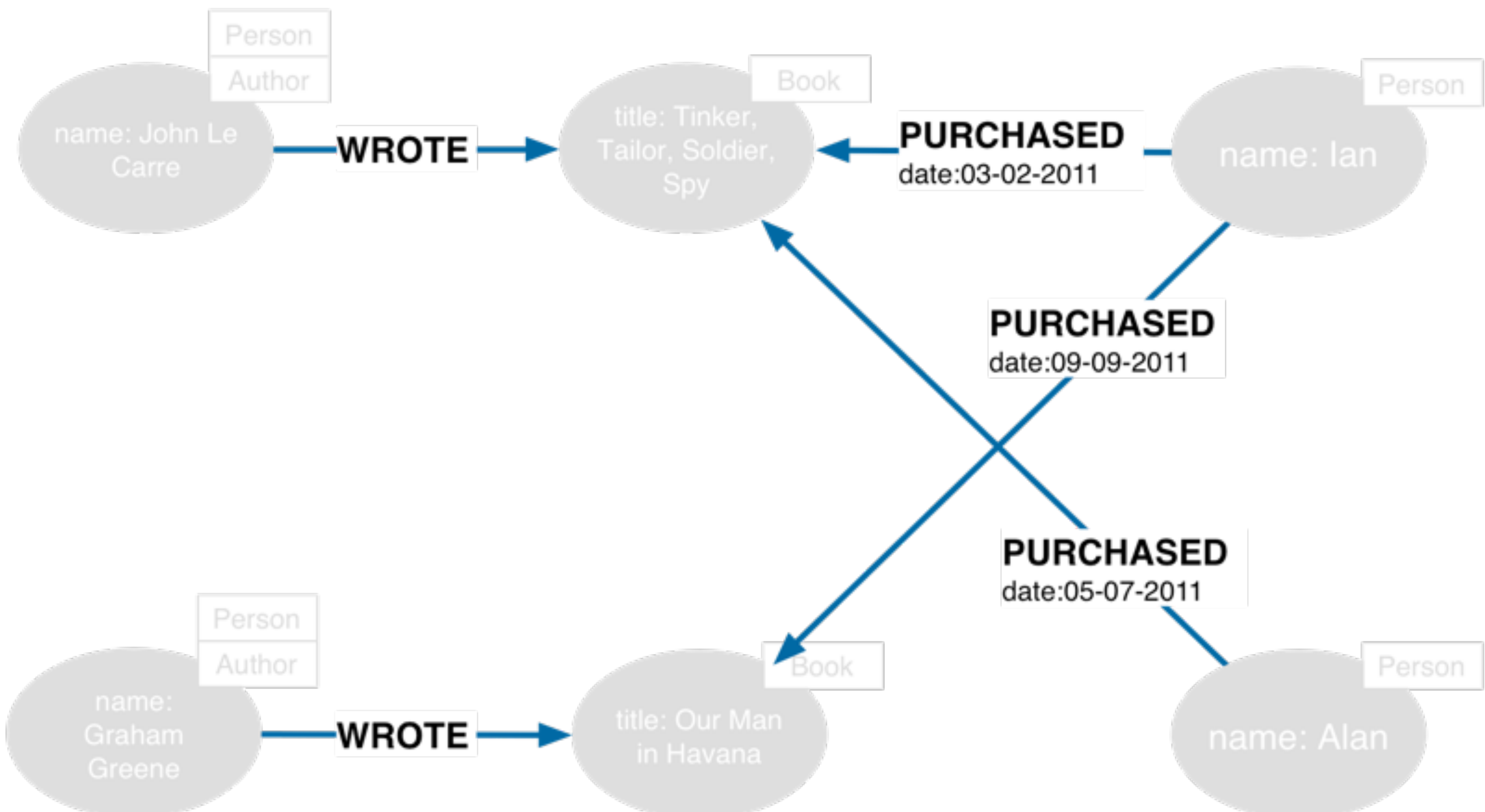


Nodes

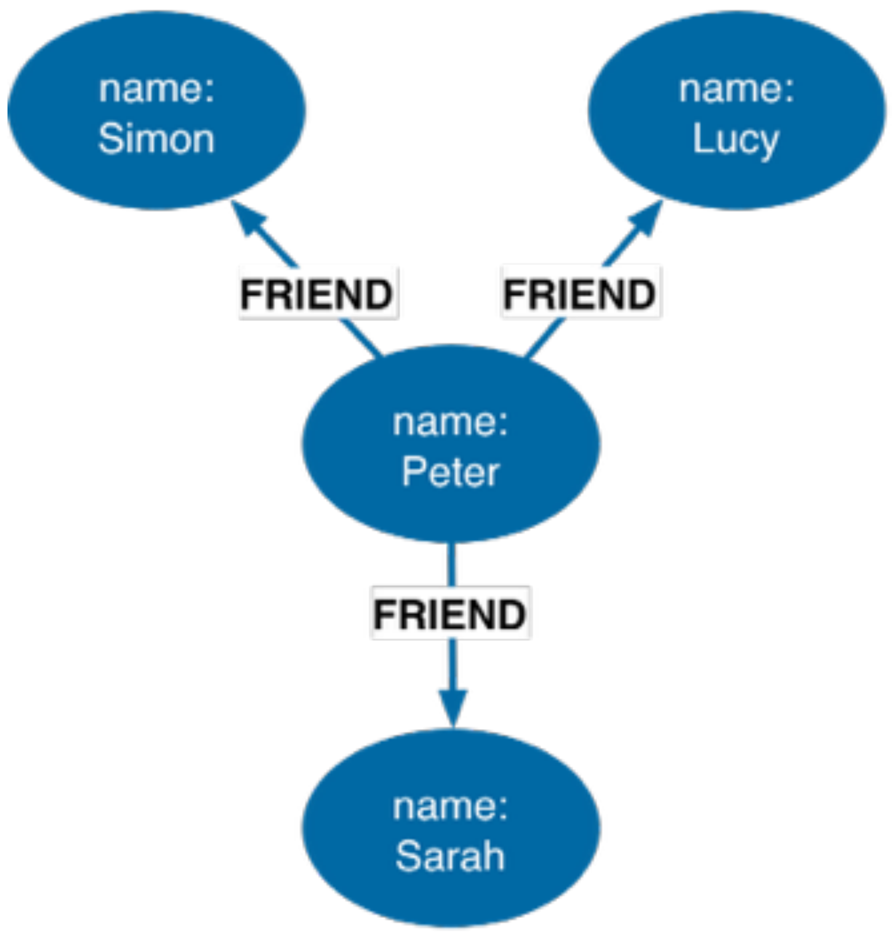


Person
Author
name: John Le Carre
WROTE
Book
title: Tinker, Tailor, Soldier, Spy
PURCHASED
date:03-02-2011
Person
name: Ian
PURCHASED
date:09-09-2011
PURCHASED
date:05-07-2011
Person
Author
name: Graham Greene
WROTE
Book
title: Our Man in Havana
Person
name: Alan

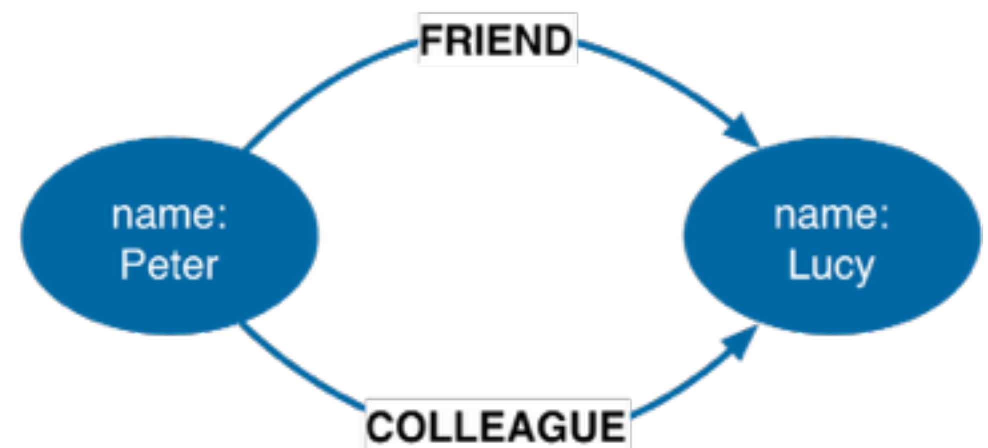
Relationships



Relationships (continued)



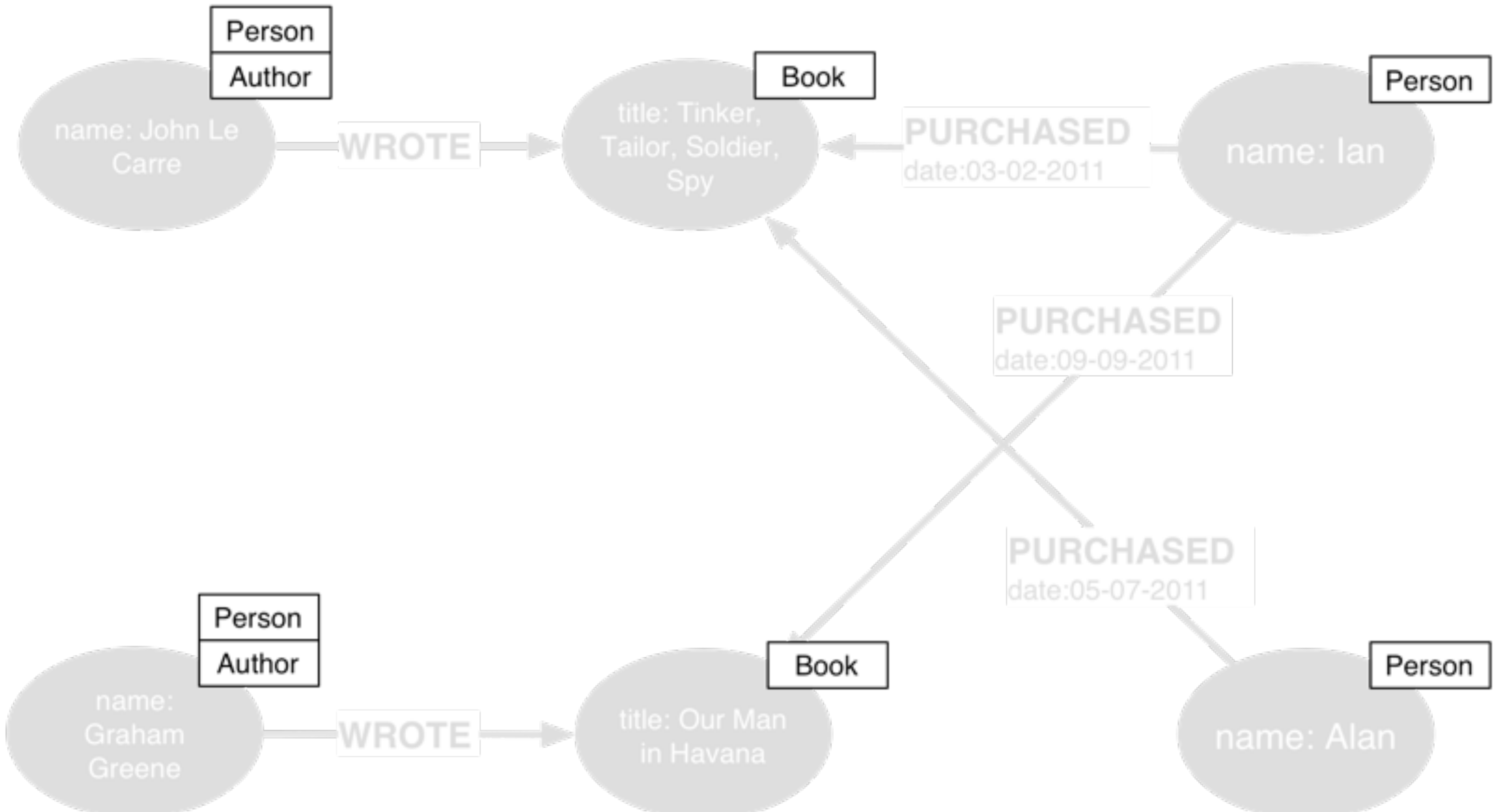
Nodes can have more than one relationship



Nodes can be connected by more than one relationship

Self relationships are allowed

Labels



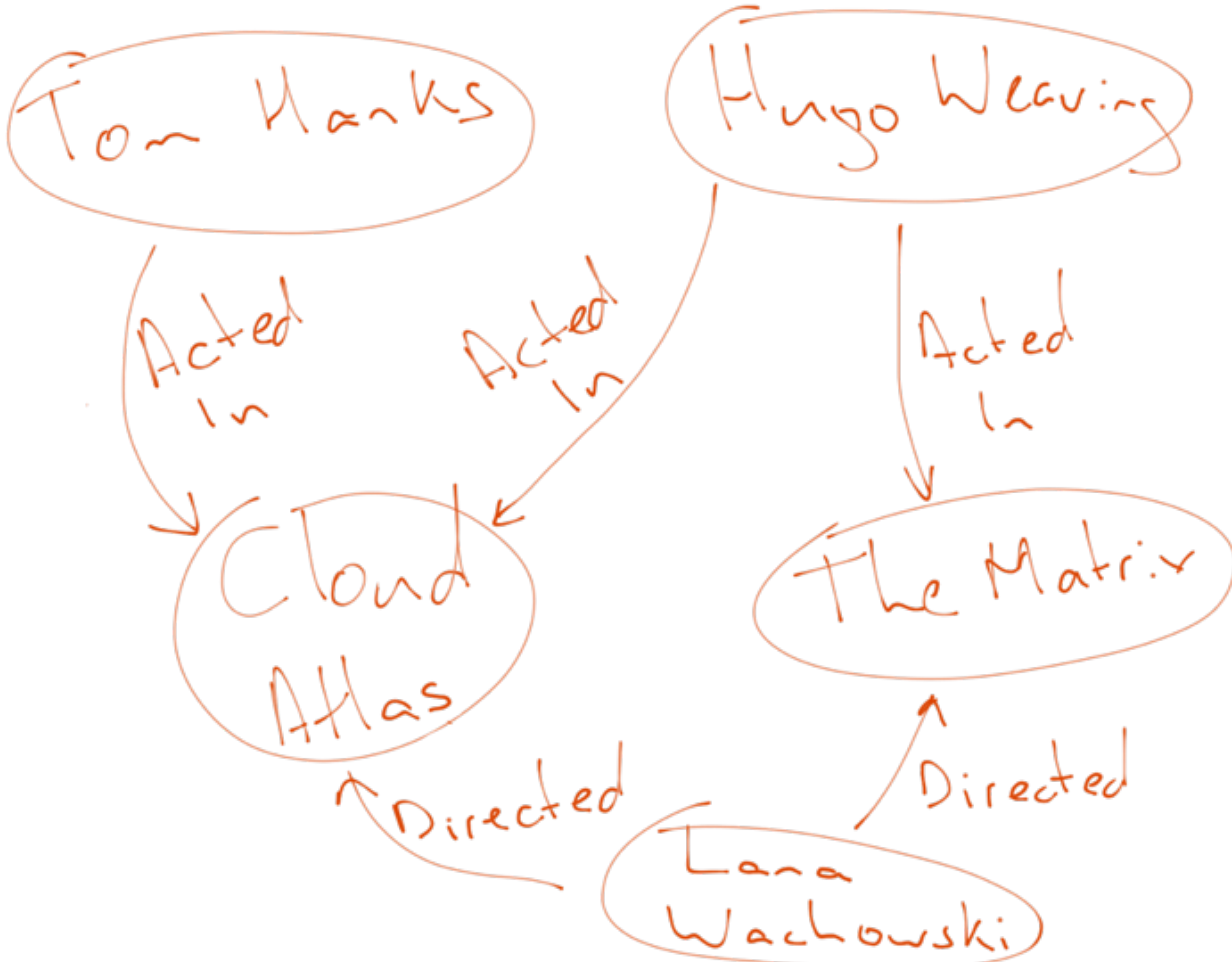
Person
Author
name: John Le Carre
WROTE
Book
title: Tinker, Tailor, Soldier, Spy
PURCHASED
date:03-02-2011
Person
name: Ian
PURCHASED
date:09-09-2011
PURCHASED
date:05-07-2011
Person
Author
name: Graham Greene
WROTE
Book
title: Our Man in Havana
Person
name: Alan

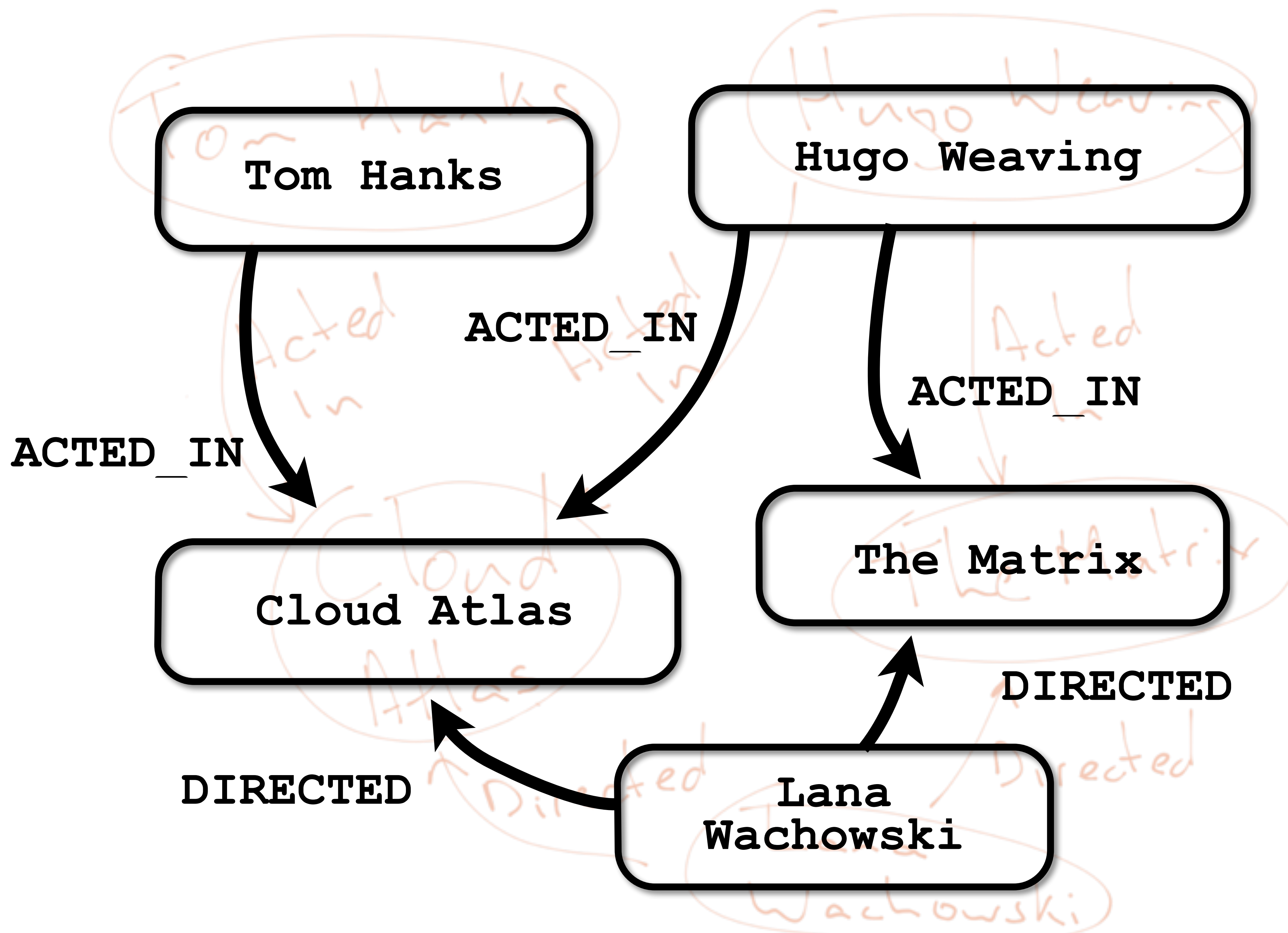
Four Building Blocks

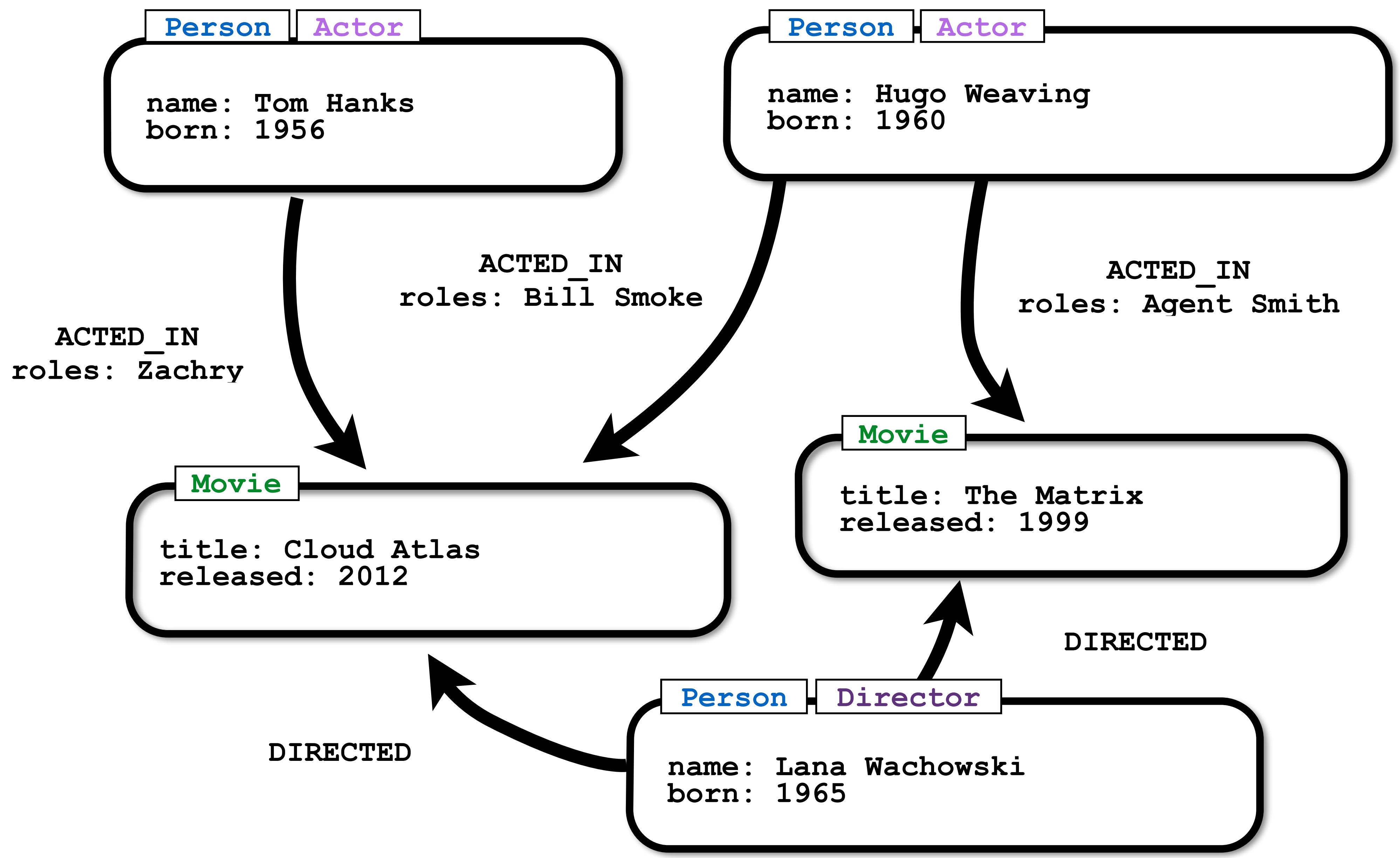
- Nodes
  - Entities
- Relationships
  - Connect entities and structure domain
- Properties
  - Attributes and metadata
- Labels
  - Group nodes by role

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


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


Person
Actor
name: Tom Hanks
born: 1956
Person
Actor
name: Hugo Weaving
born: 1960
ACTED_IN
roles: Zachry
ACTED_IN
roles: Bill Smoke
ACTED_IN
roles: Agent Smith
Movie
title: Cloud Atlas
released: 2012
Movie
title: The Matrix
released: 1999
DIRECTED
DIRECTED
Person
Director
name: Lana Wachowski
born: 1965


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

Aggregate vs. Connected Data-Model

What is NOSQL?

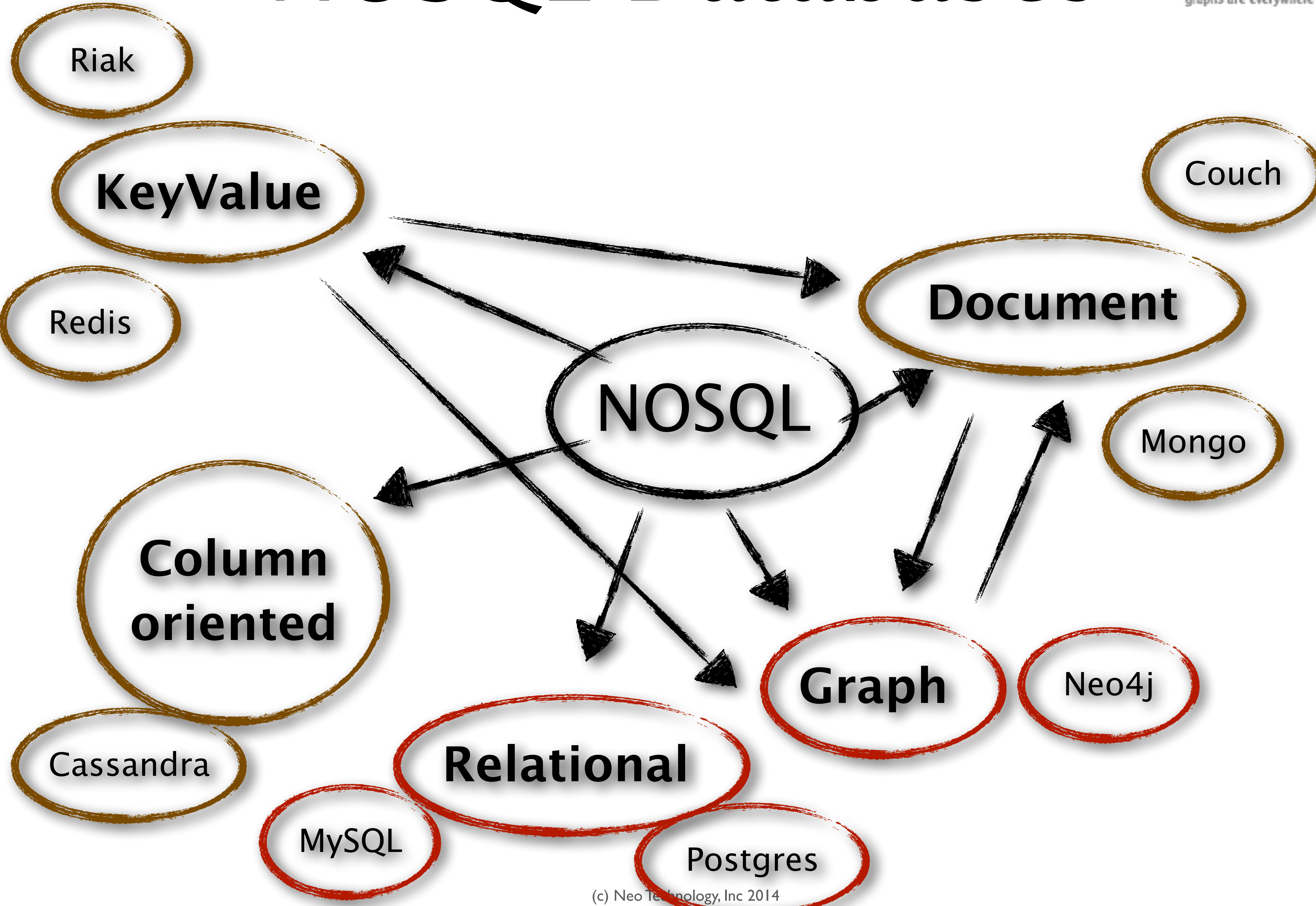
It's not "No to SQL"

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



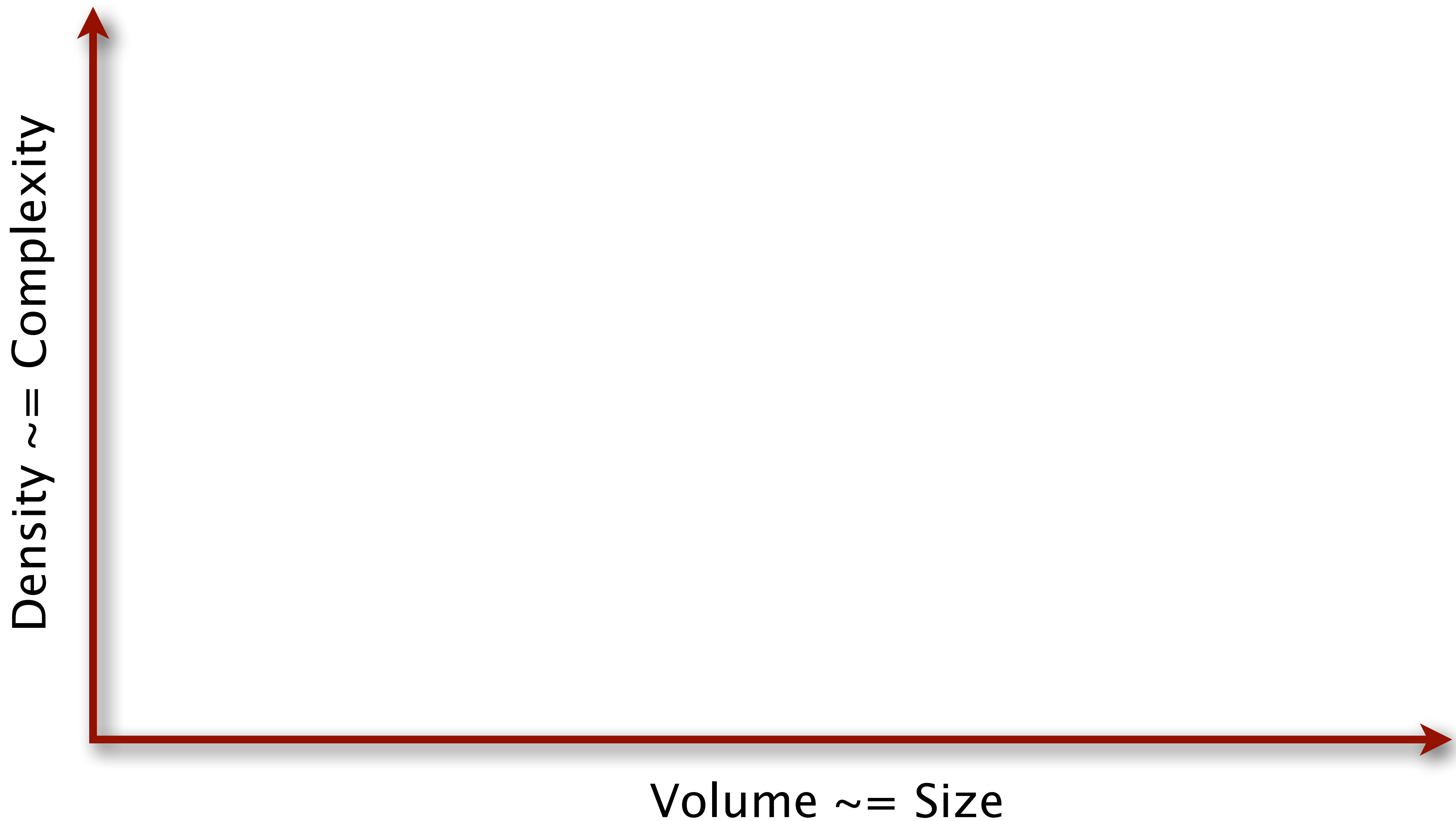
Living in a NOSQL World

Living in a NOSQL World

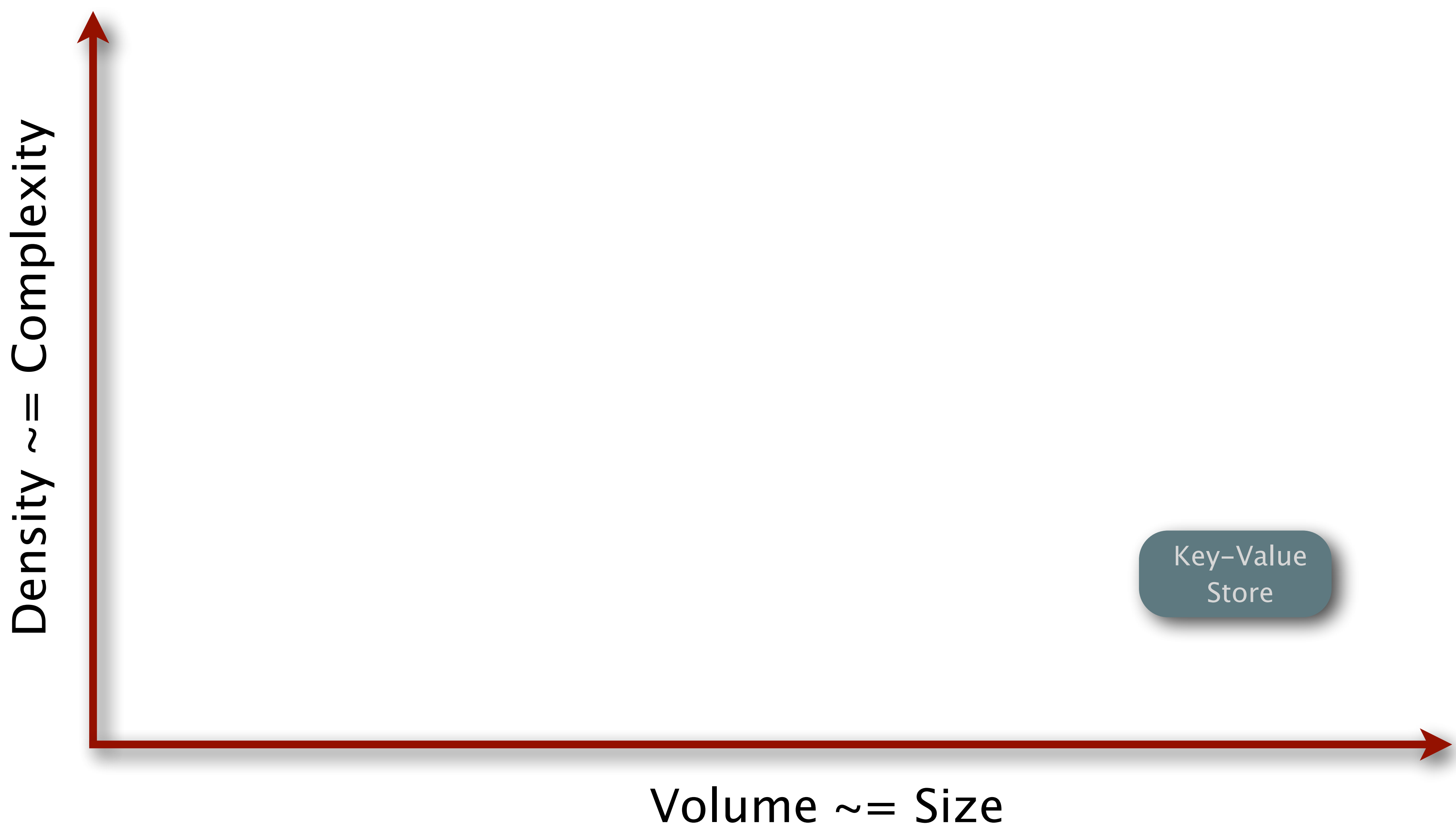
Volume ~= Size

neotechnology
graphs are everywhere

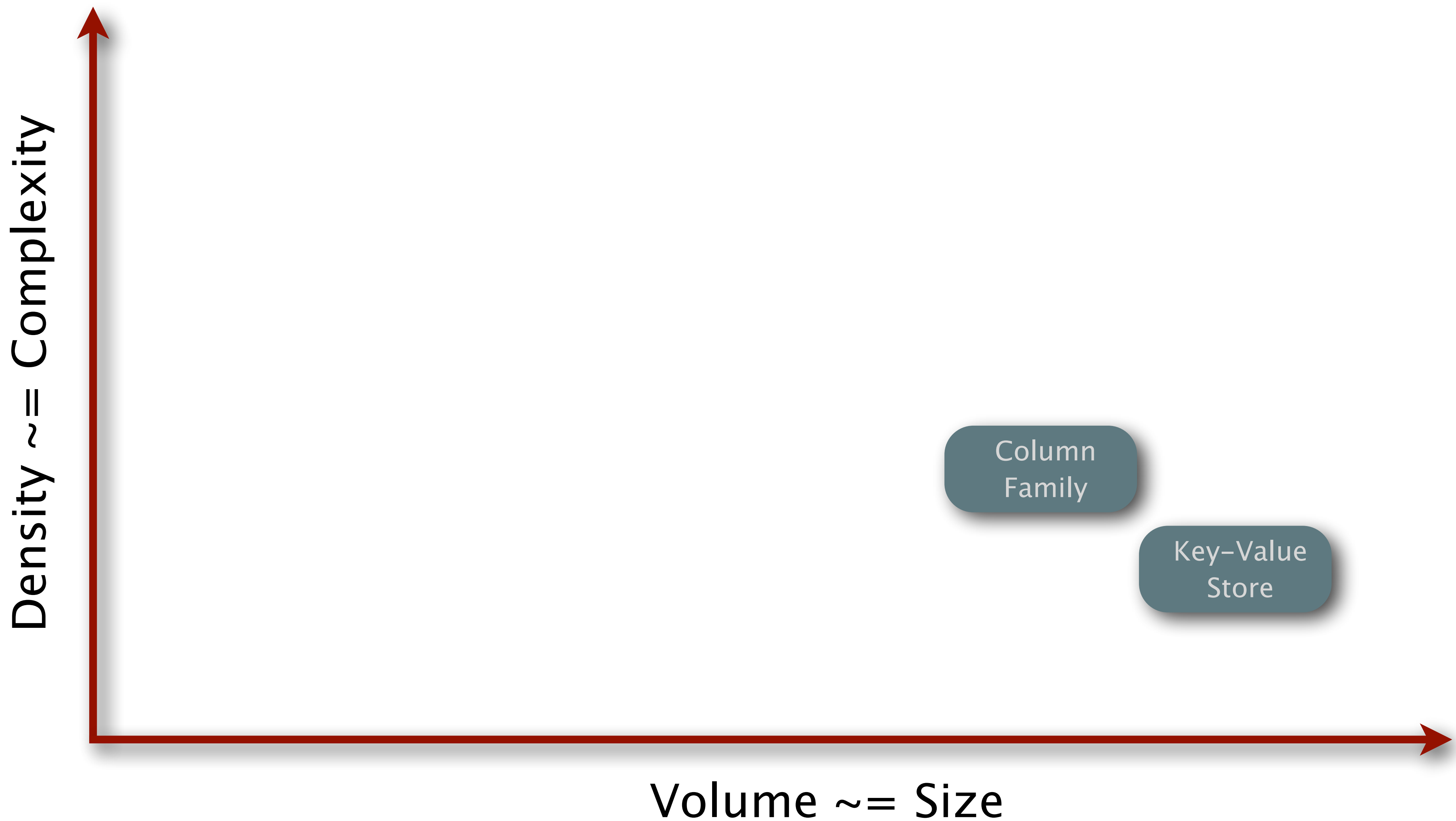
Living in a NOSQL World



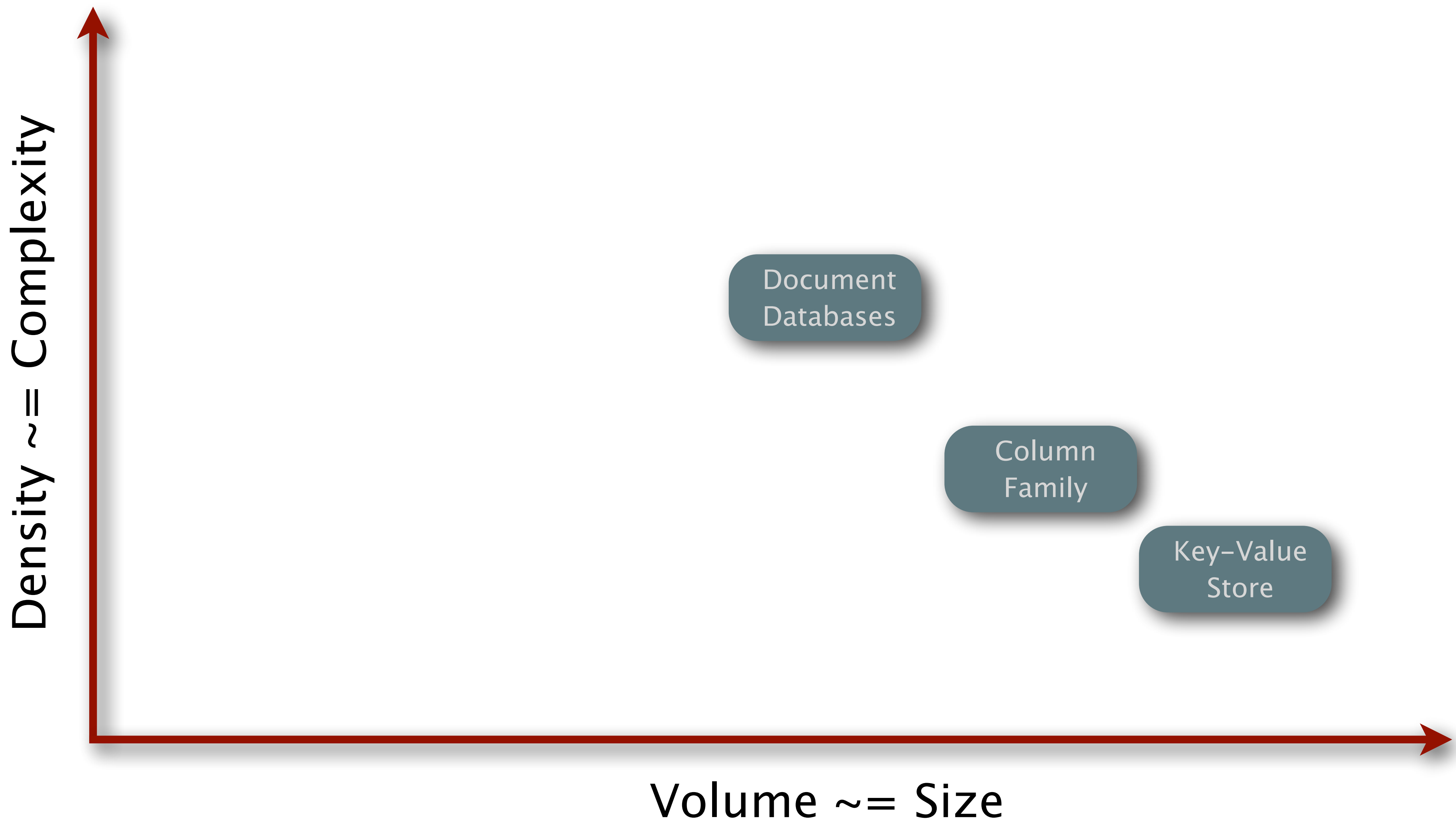
Living in a NOSQL World



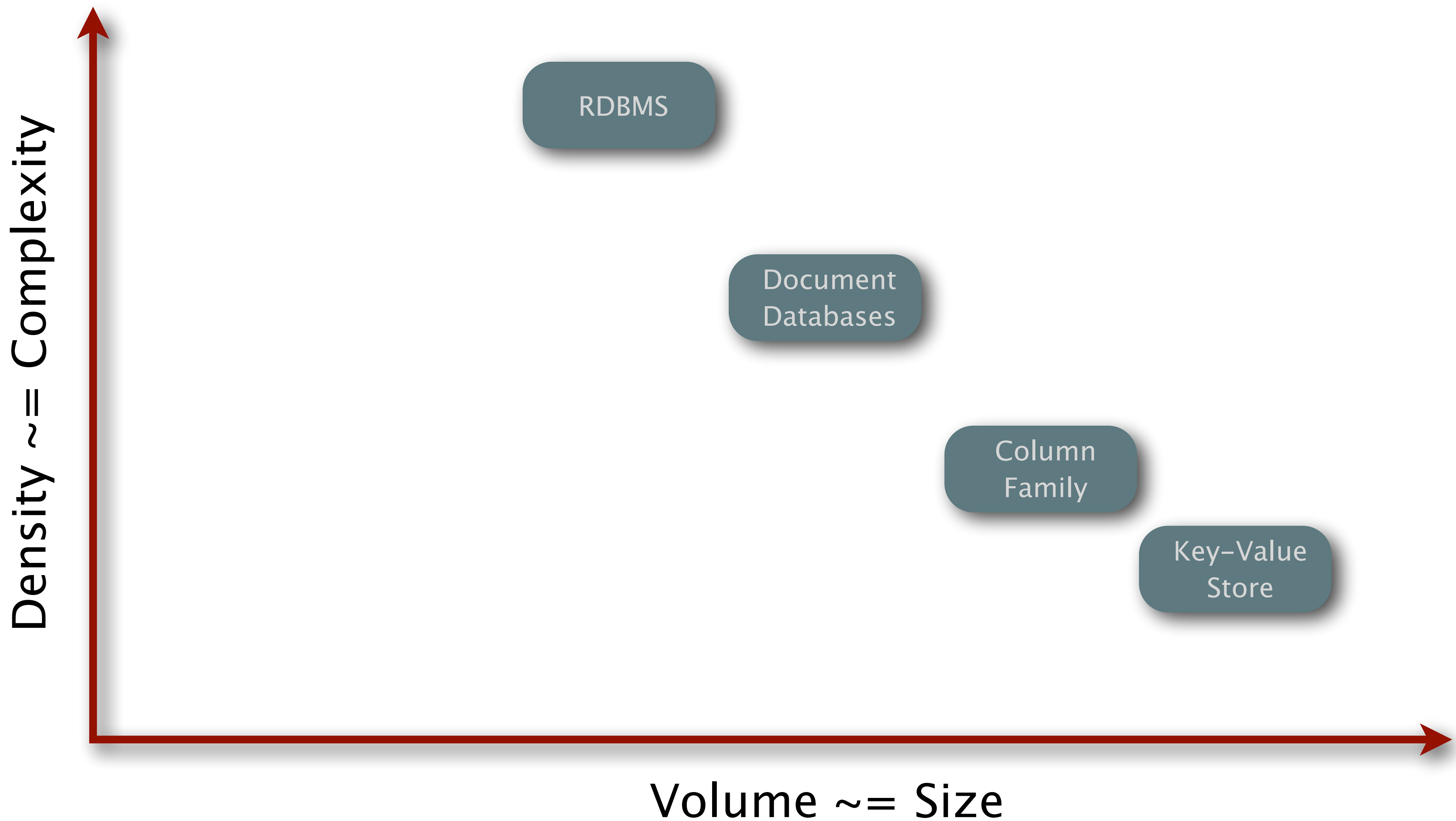
Living in a NOSQL World



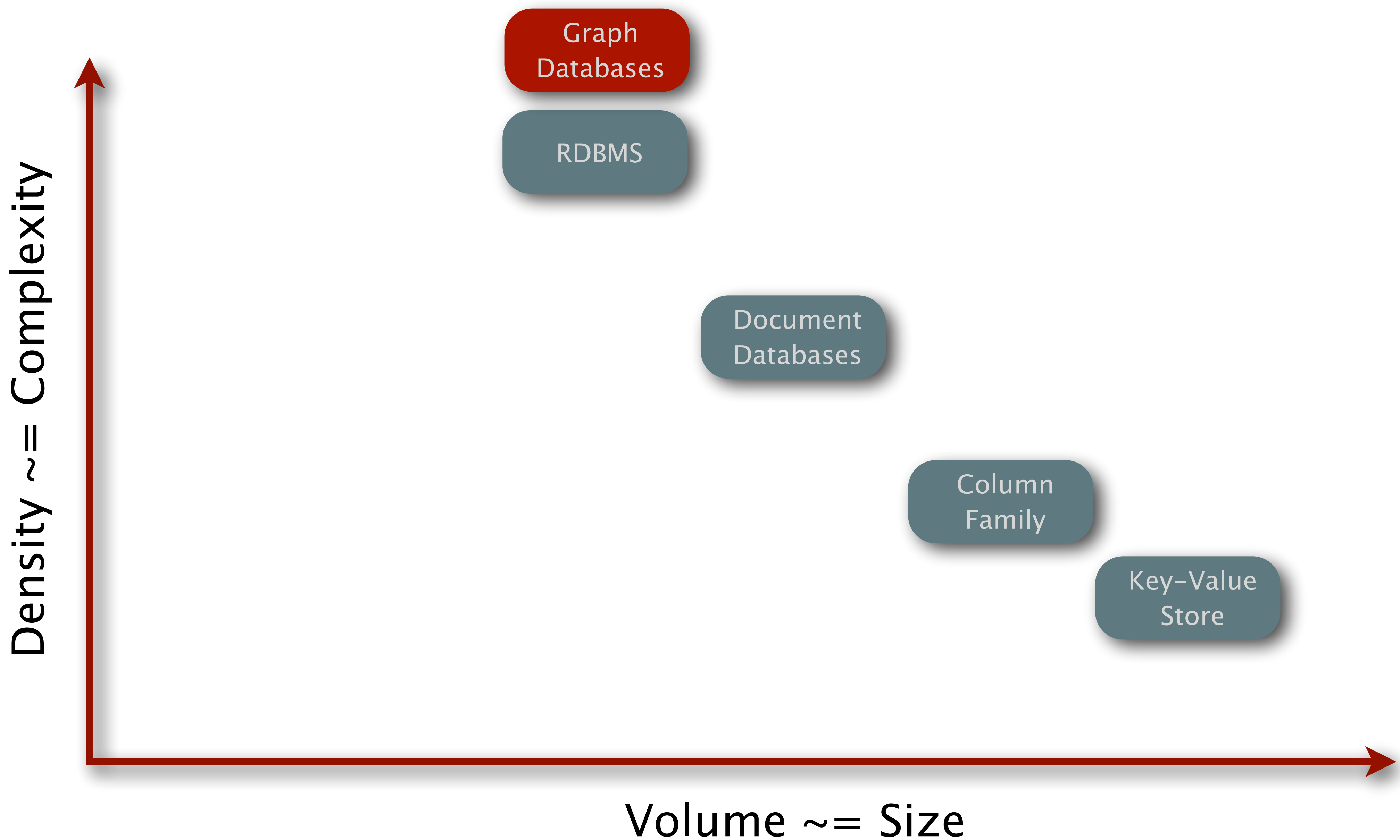
Living in a NOSQL World



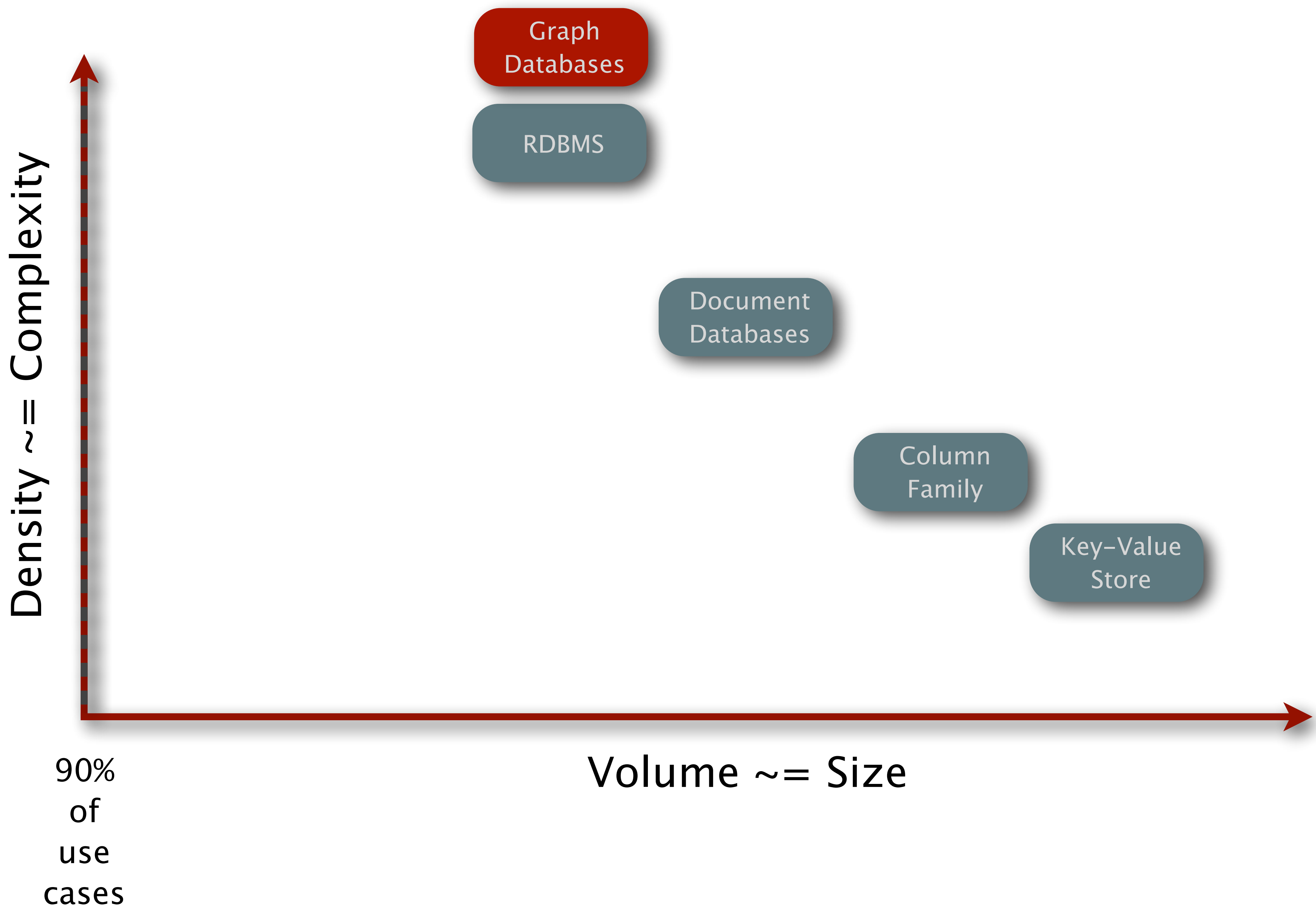
Living in a NOSQL World



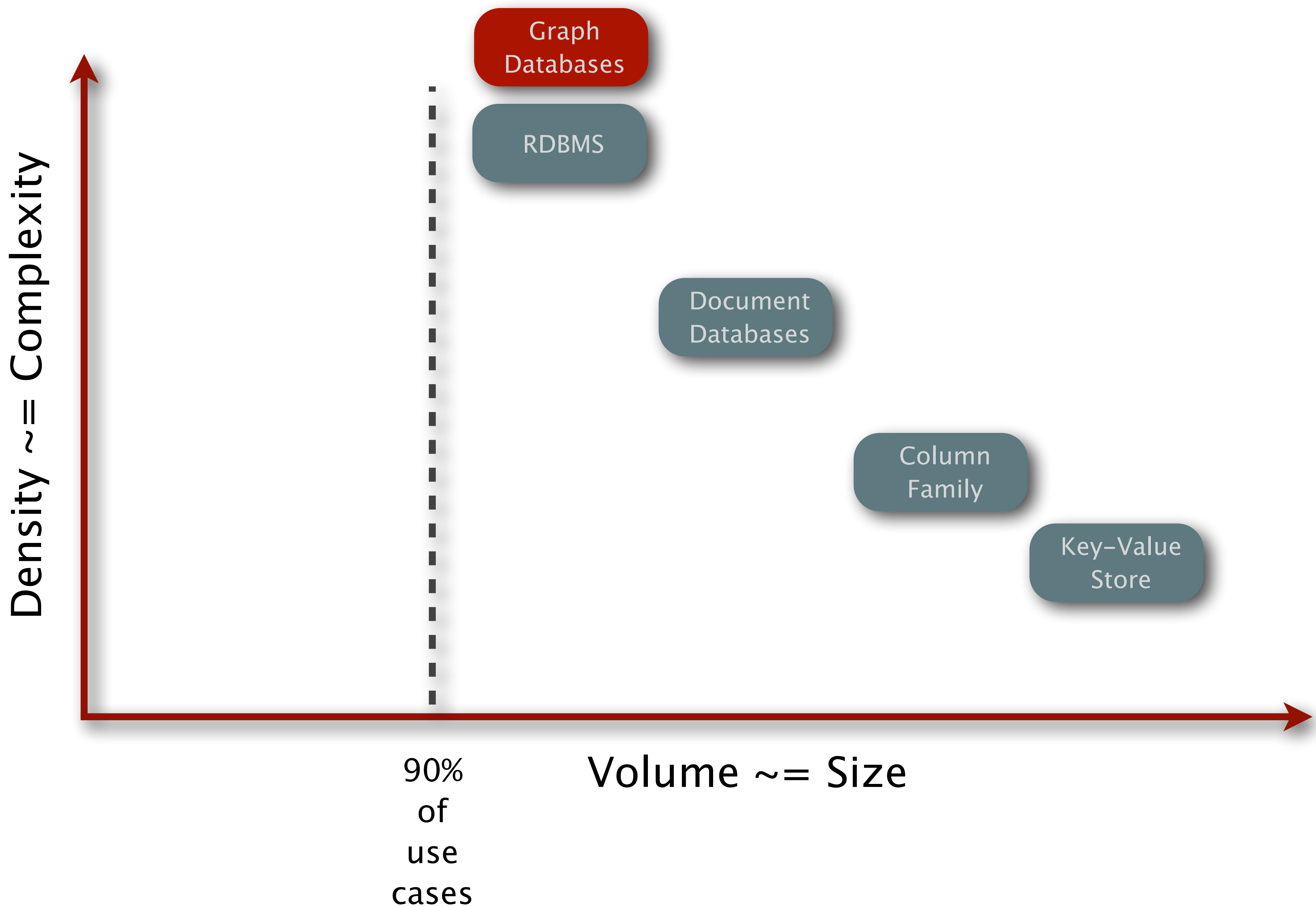
Living in a NOSQL World



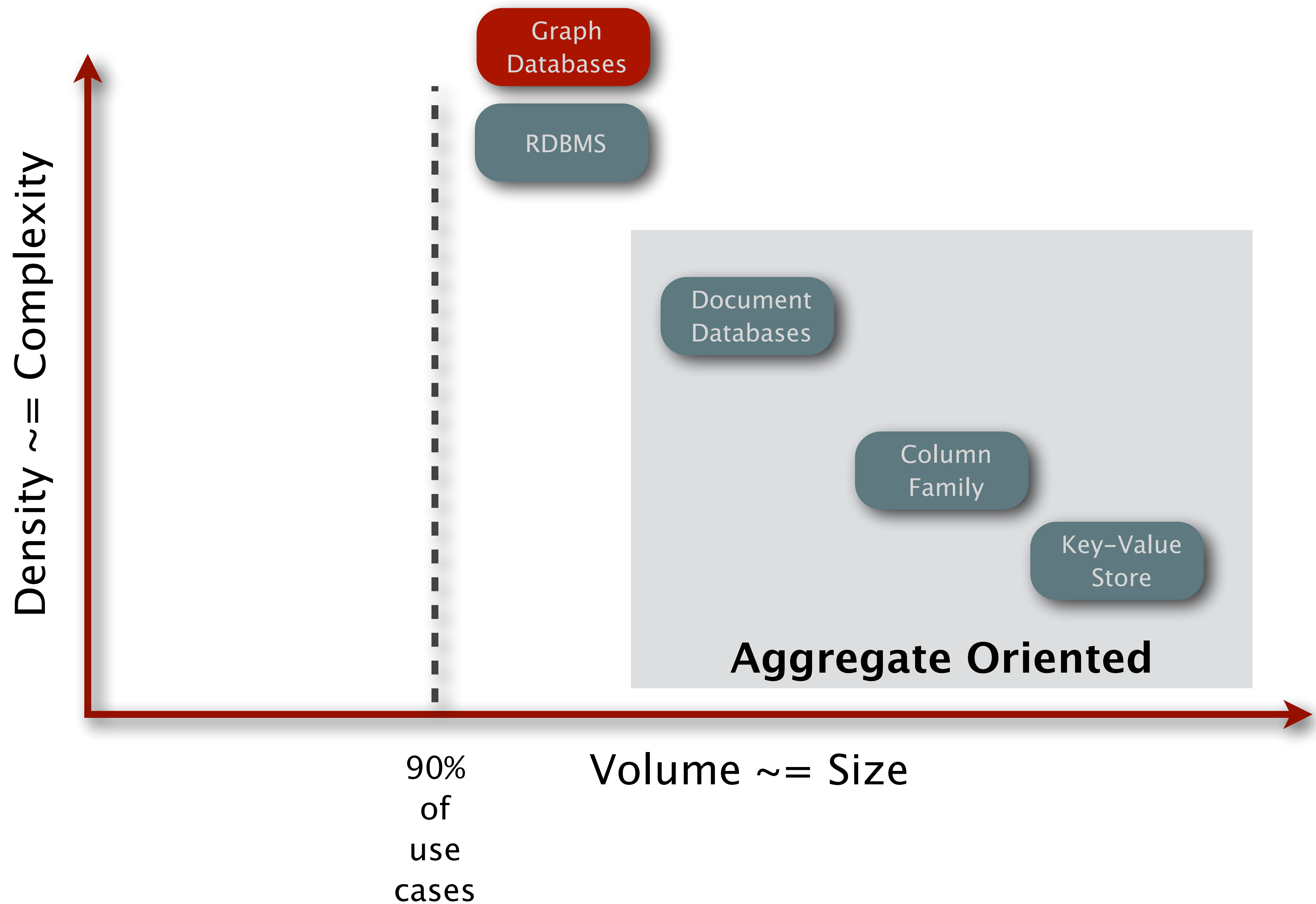
Living in a NOSQL World



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

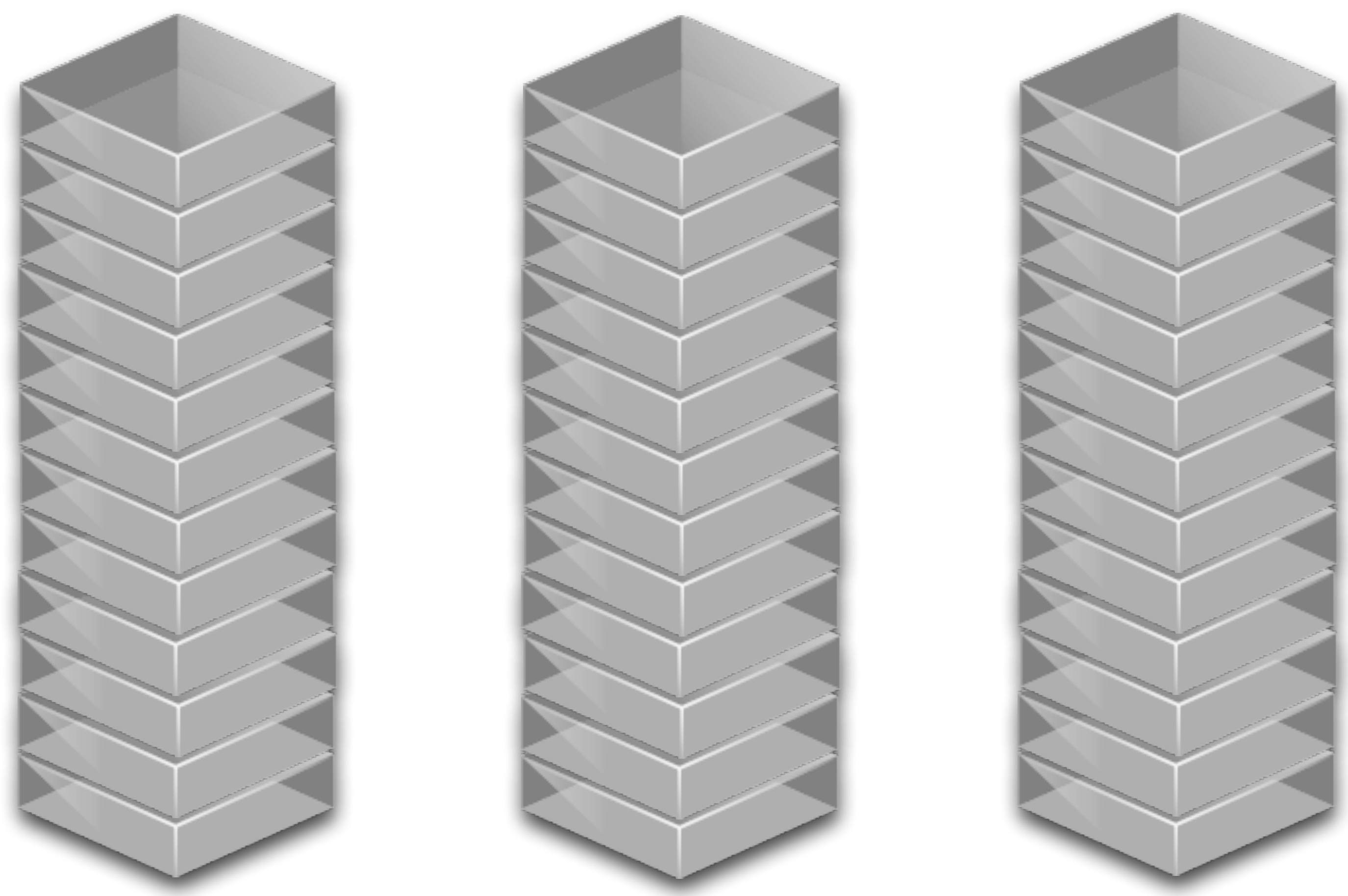
Relational vs. Graph

Relational vs. Graph

You know relational

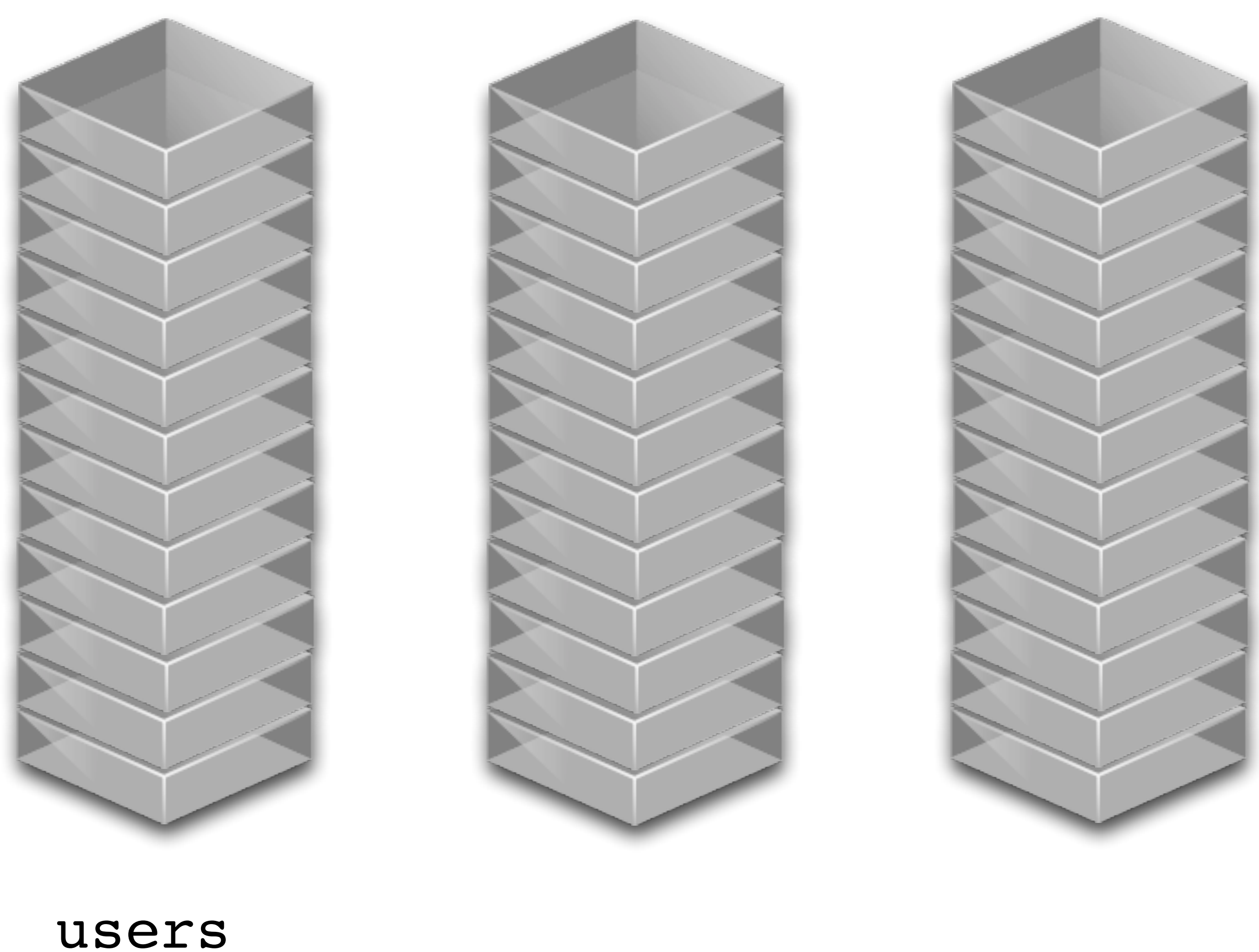
Relational vs. Graph

You know relational



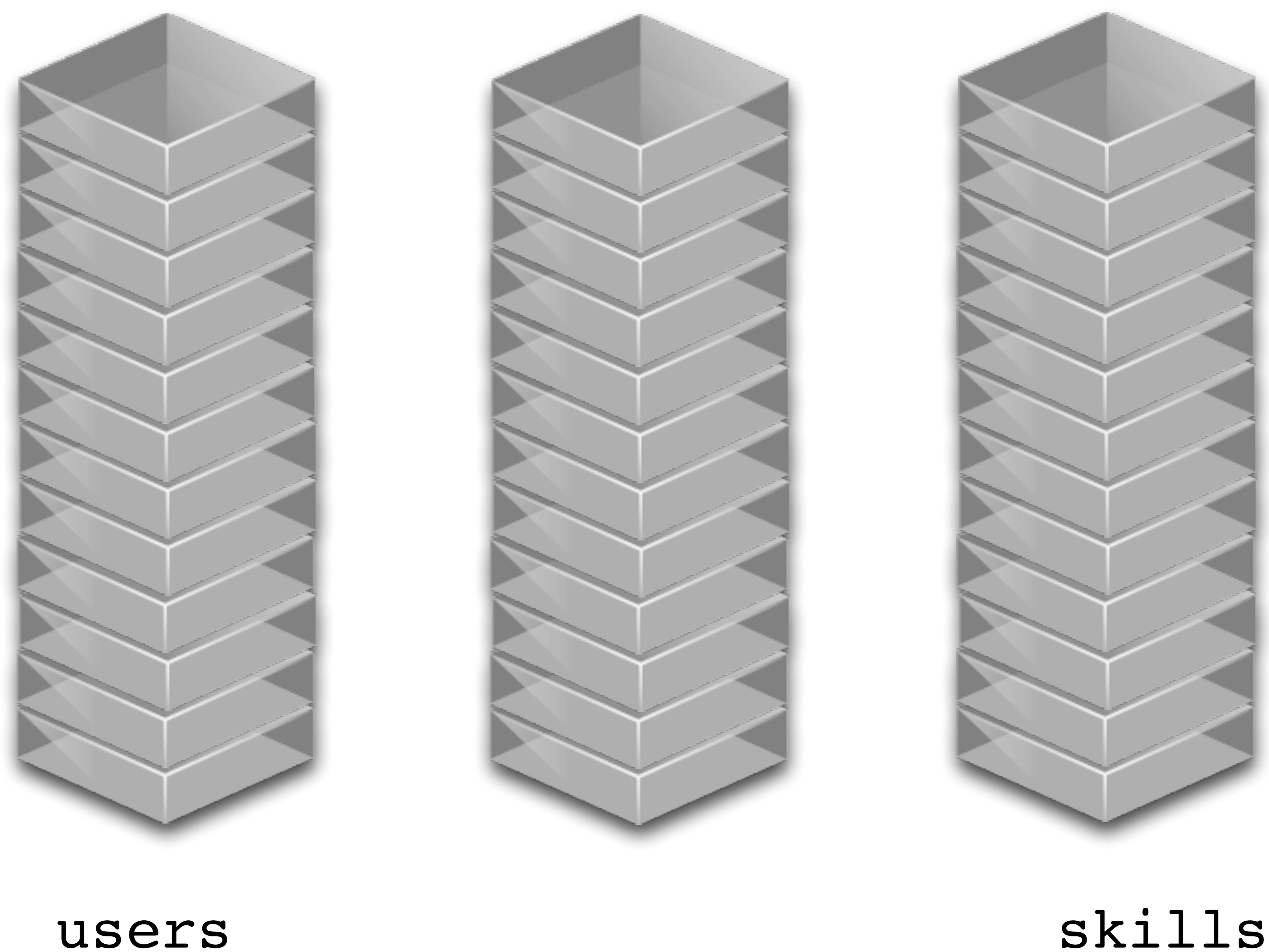
Relational vs. Graph

You know relational



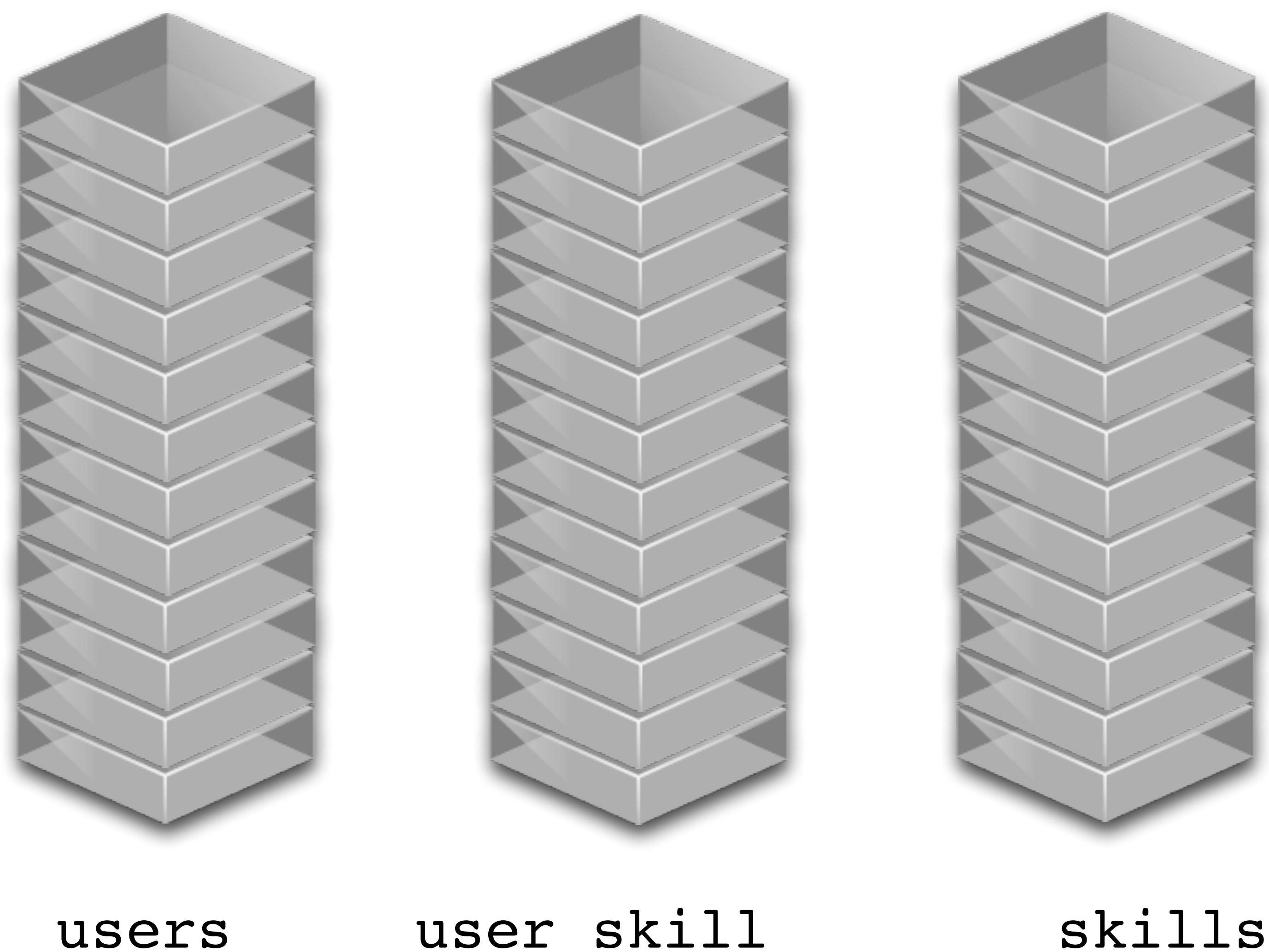
Relational vs. Graph

You know relational



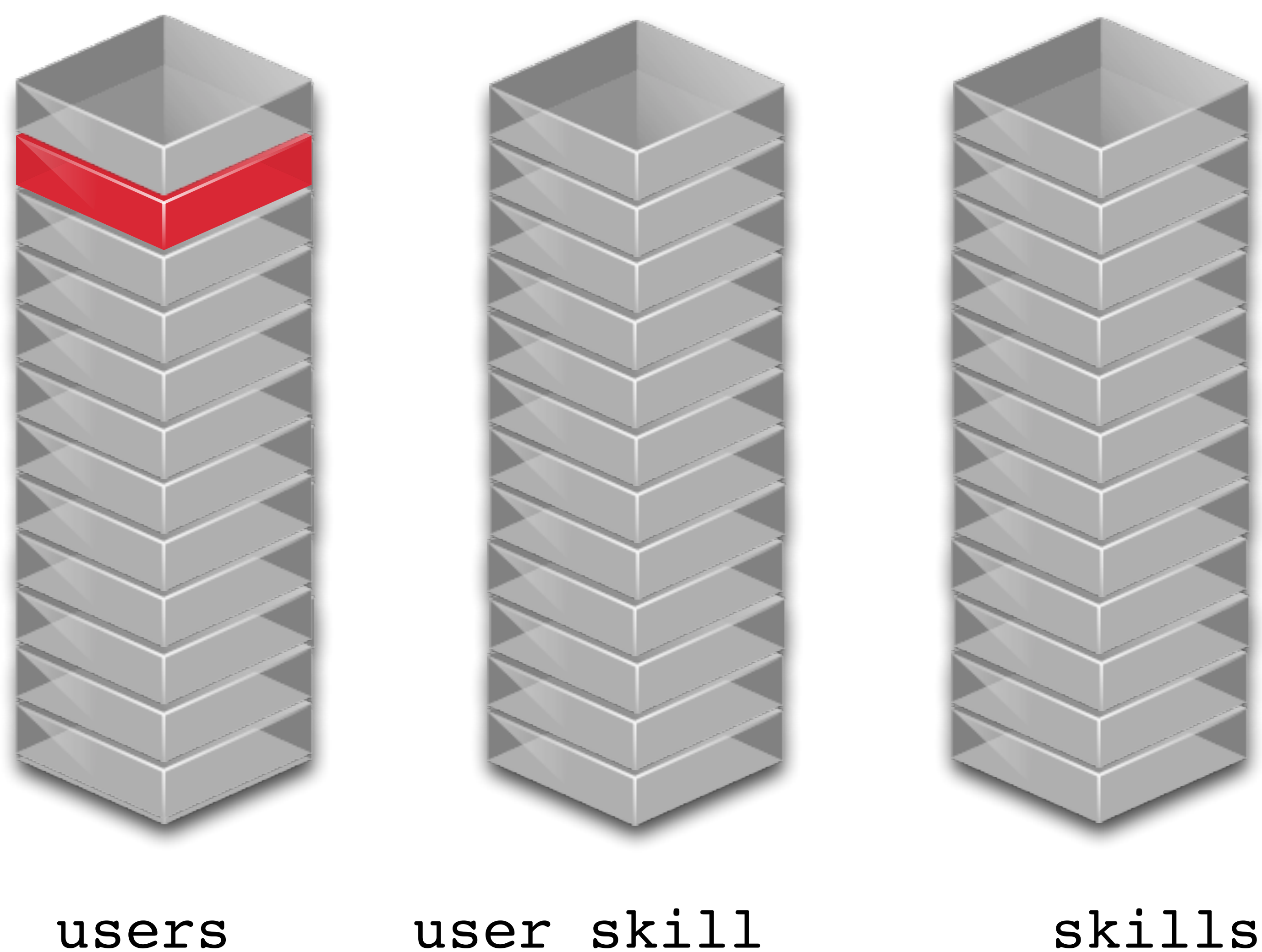
Relational vs. Graph

You know relational



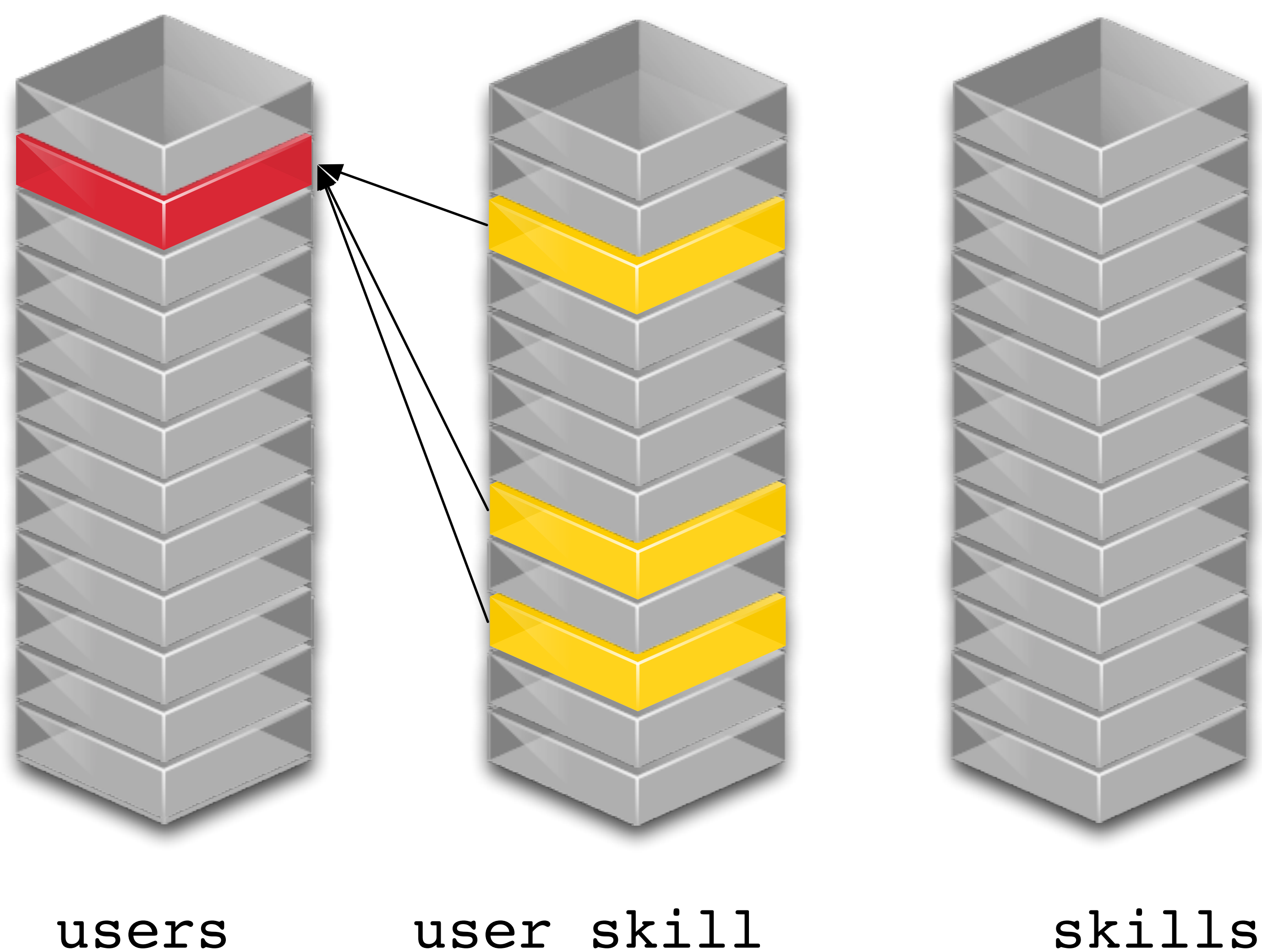
Relational vs. Graph

You know relational



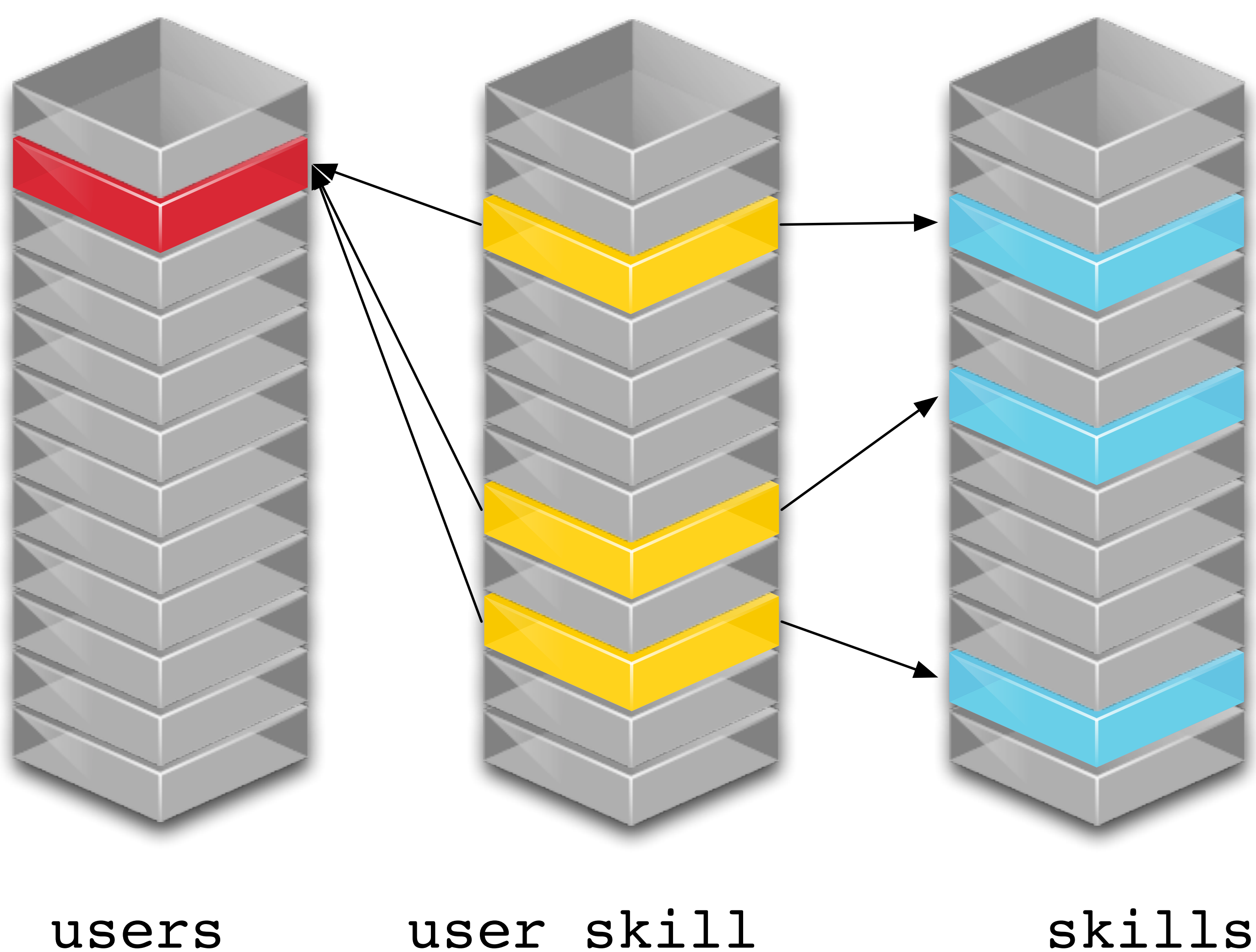
Relational vs. Graph

You know relational



Relational vs. Graph

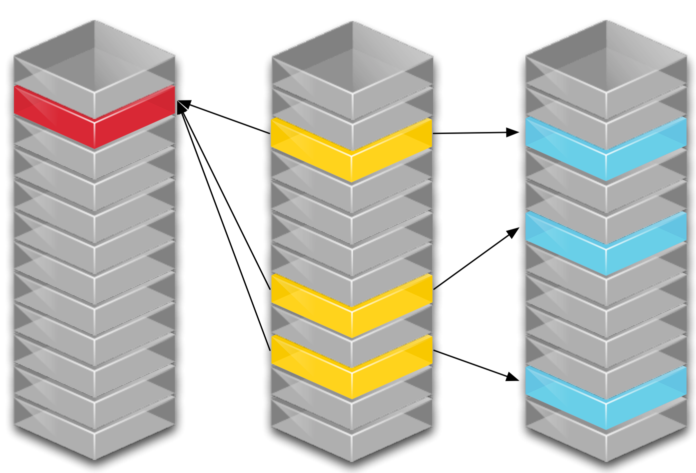
You know relational



Relational vs. Graph

You know relational

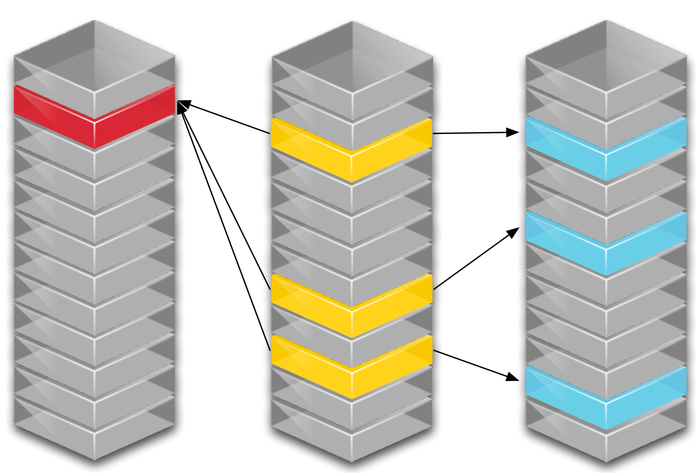
now consider relationships...

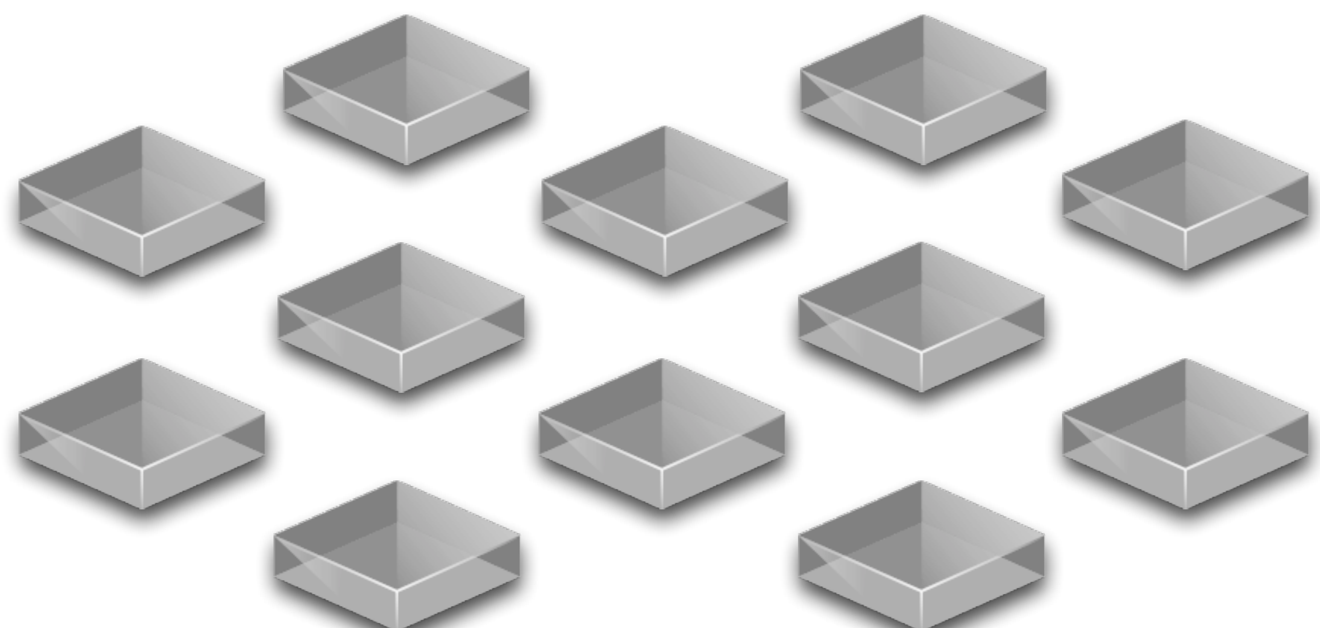


Relational vs. Graph

You know relational

now consider relationships...

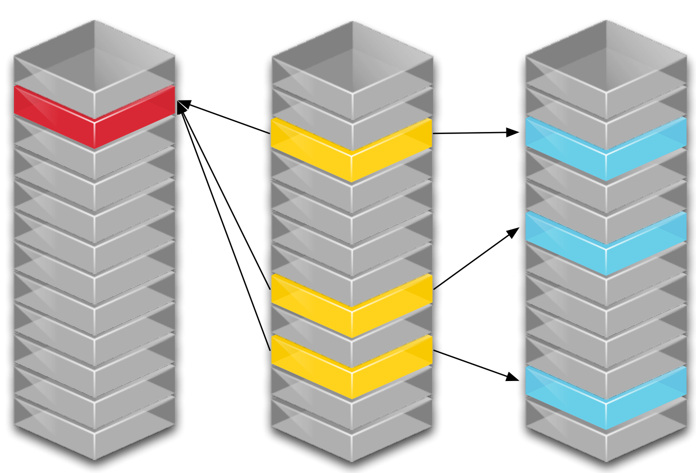


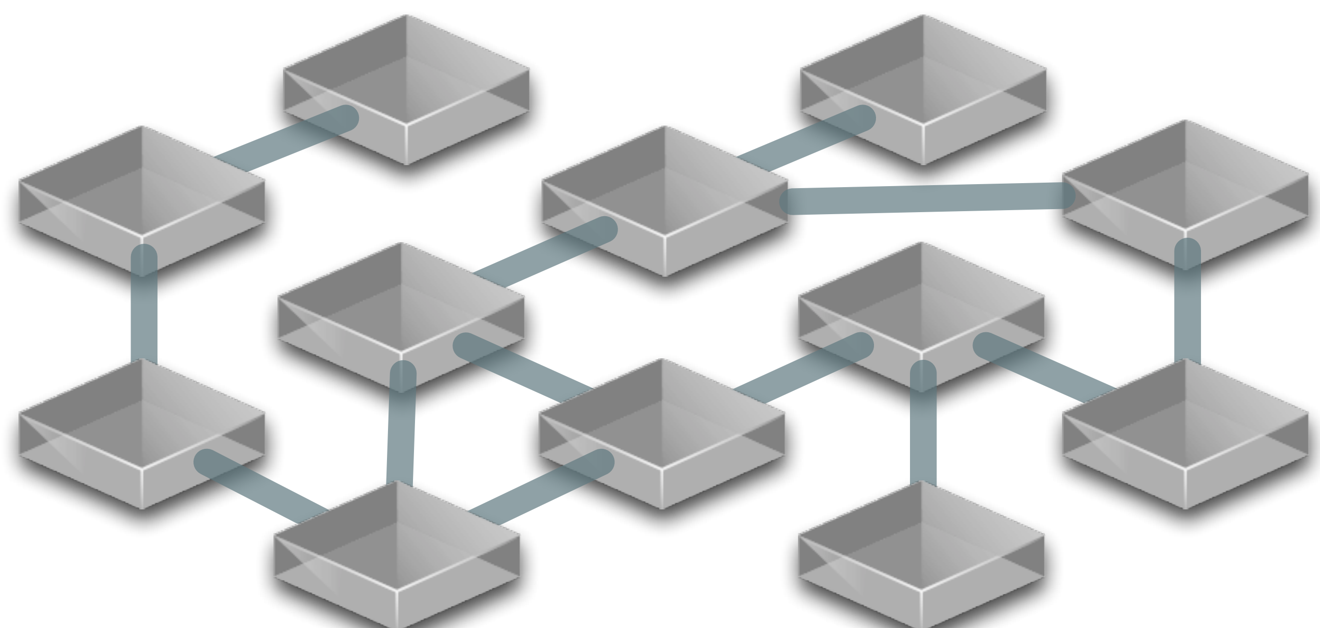


Relational vs. Graph

You know relational

now consider relationships...

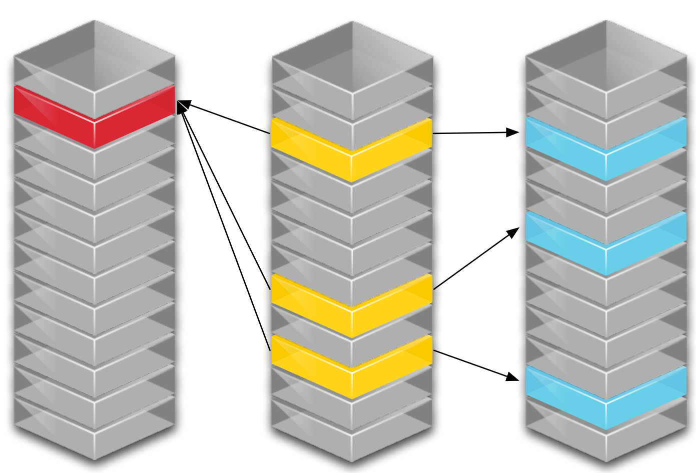


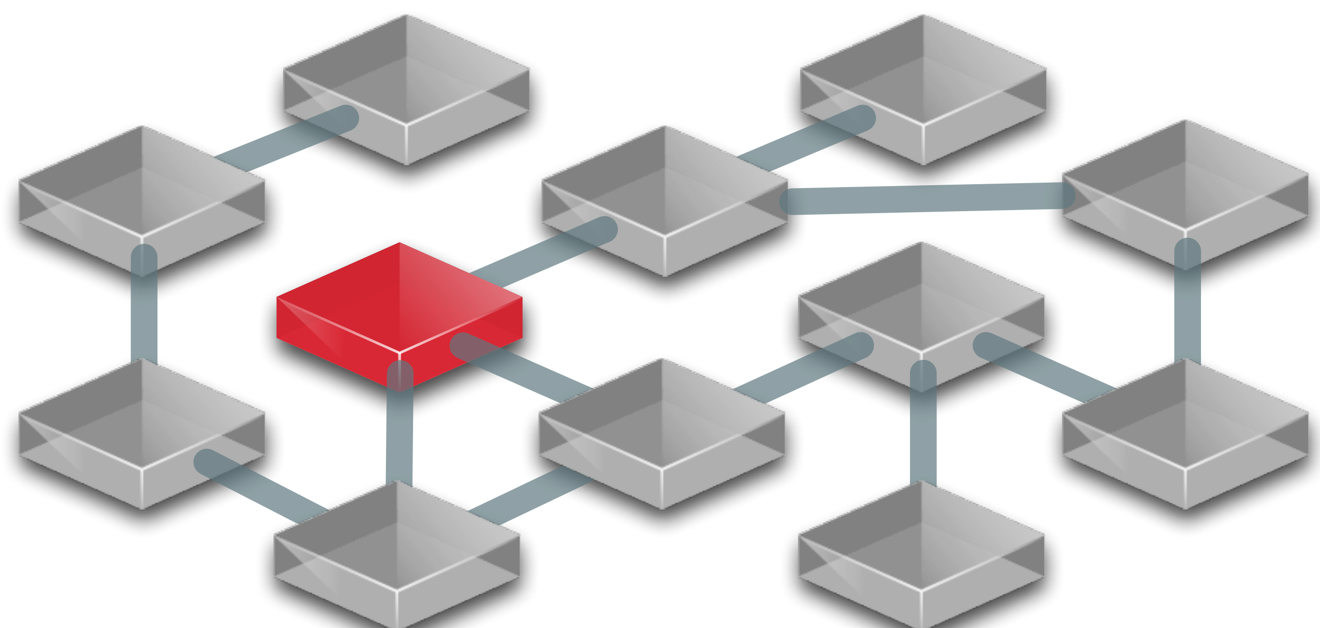


Relational vs. Graph

You know relational

now consider relationships...

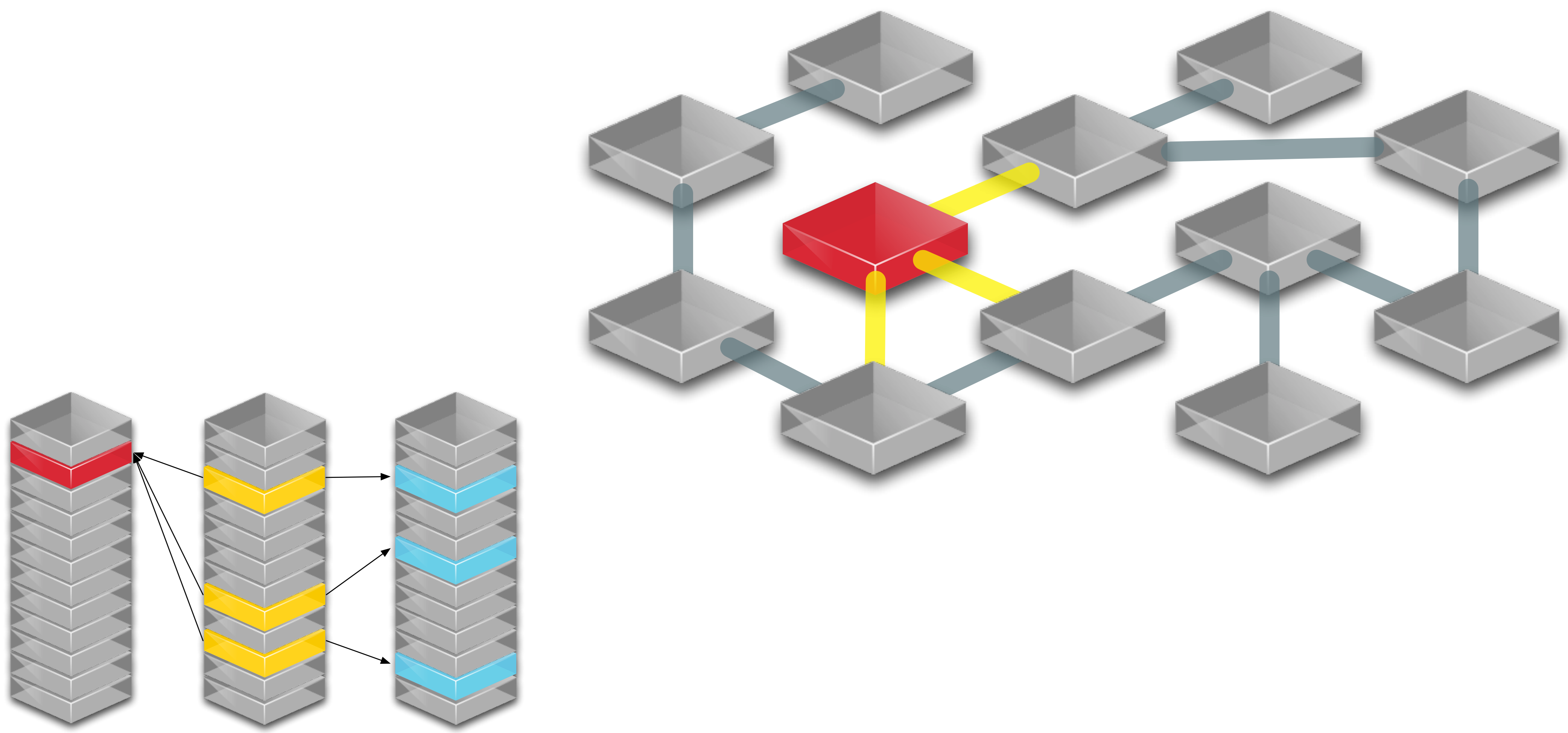




Relational vs. Graph

You know relational

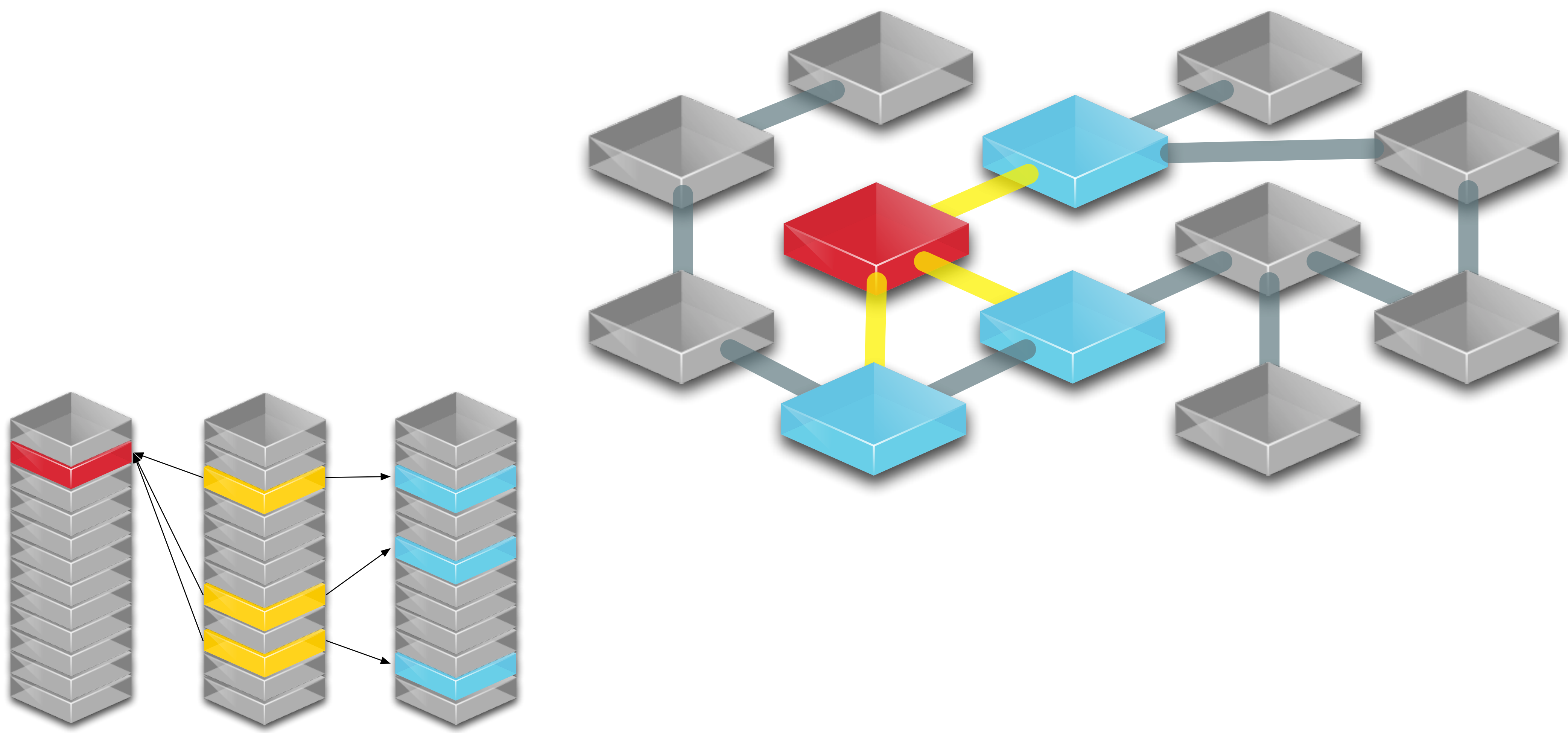
now consider relationships...



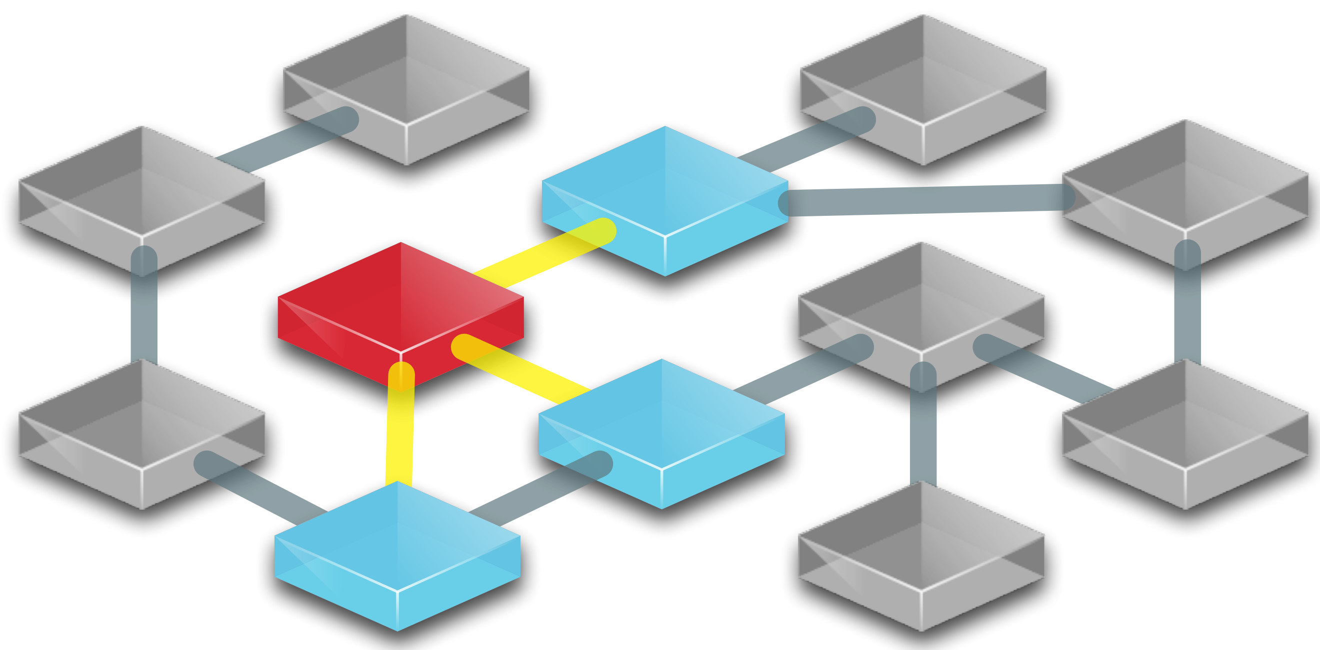
Relational vs. Graph

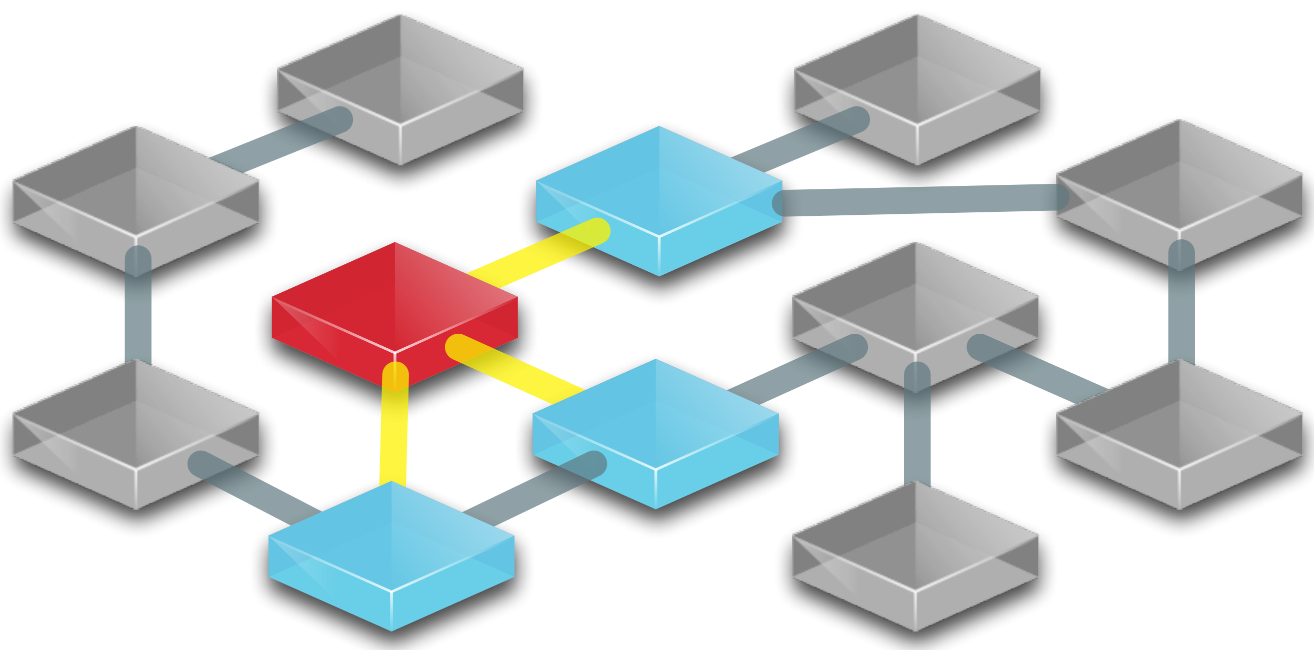
You know relational

now consider relationships...

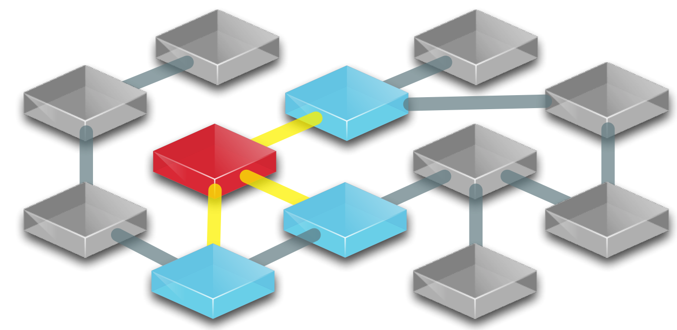


Relational vs. Graph



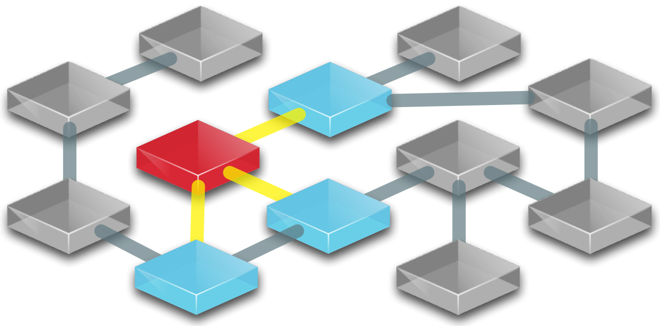


Looks different, fine. Who cares?



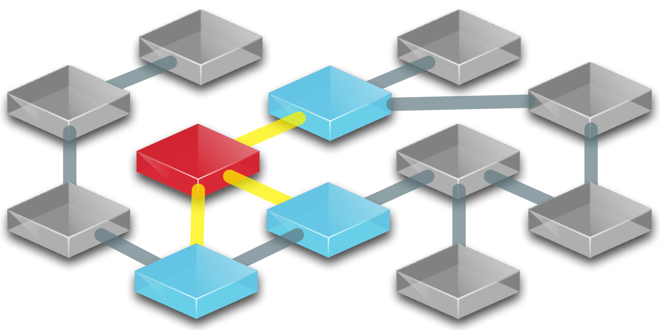
Looks different, fine. Who cares?

- a sample social graph



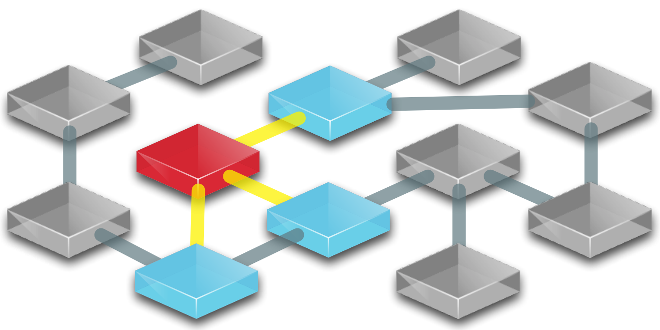
Looks different, fine. Who cares?

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



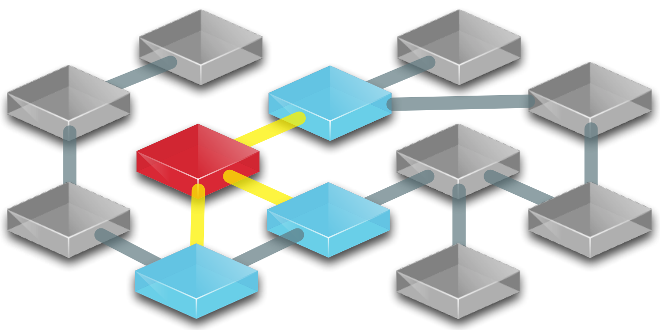
Looks different, fine. Who cares?

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



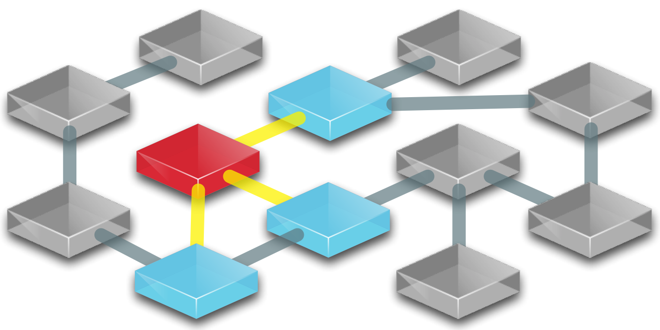
Looks different, fine. Who cares?

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



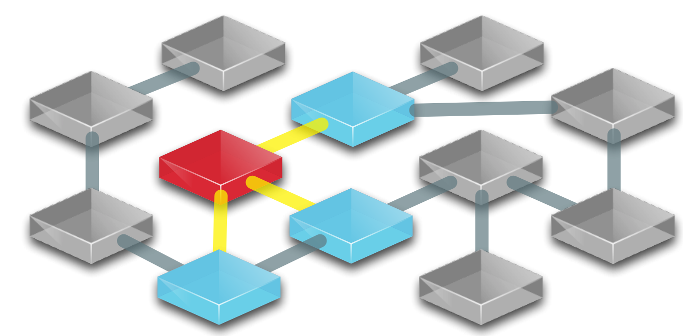
Looks different, fine. Who cares?

- 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



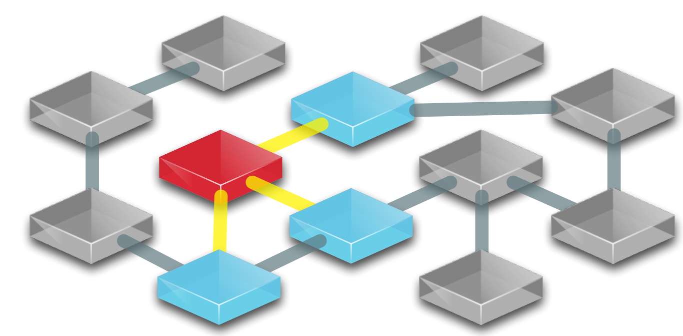
Looks different, fine. Who cares?

- 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, fine. Who cares?

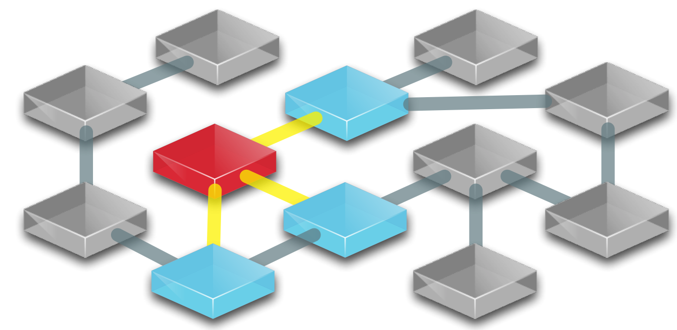


- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- pathExists(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O

	# persons	query time
Relational database	1.000	2000ms
Neo4j	1.000	2ms

Looks different, fine. 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



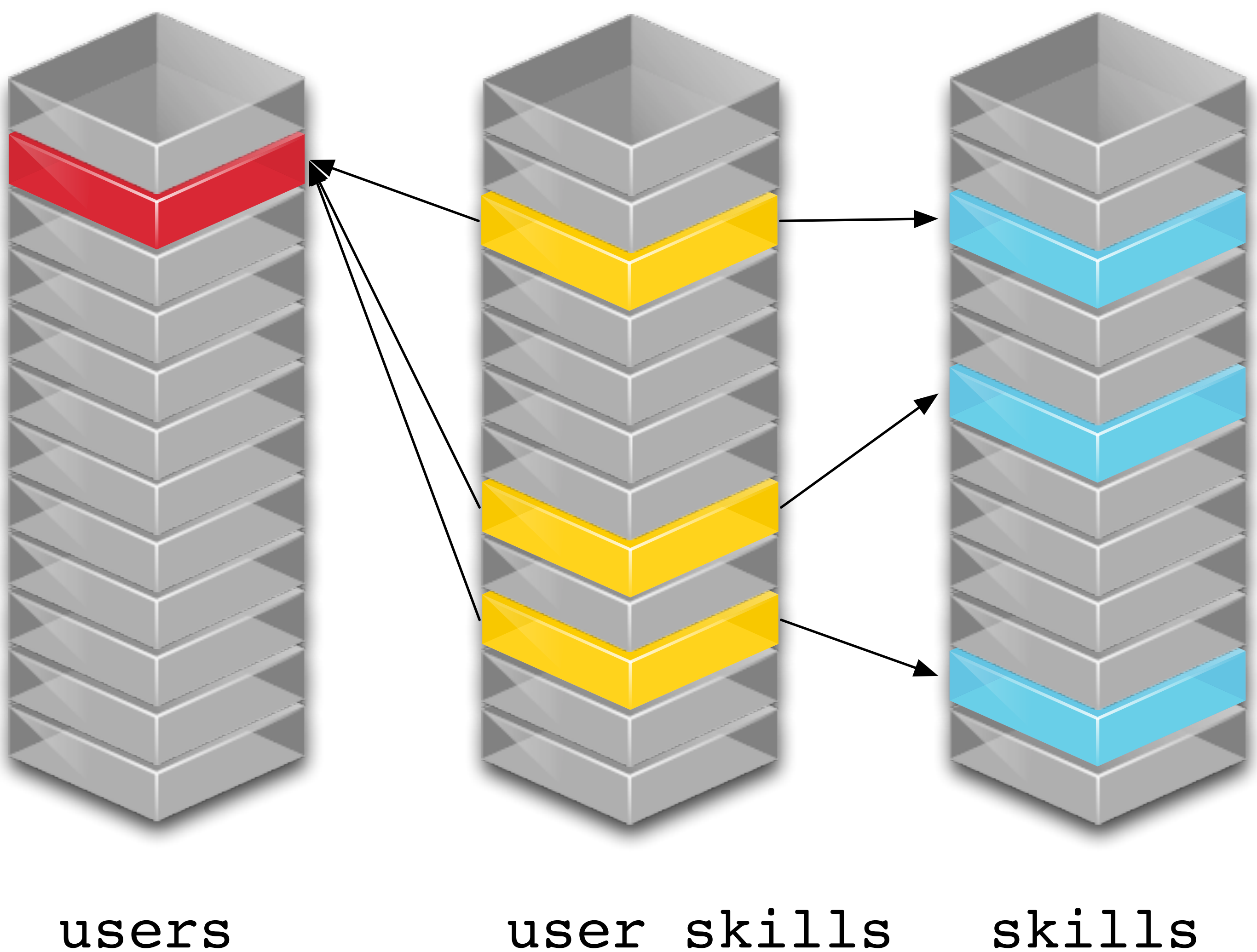
	# persons	query time
Relational database	1.000	2000ms
Neo4j	1.000	2ms
Neo4j	1.000.000	2ms

Graph Querying

You know how to query a relational database!

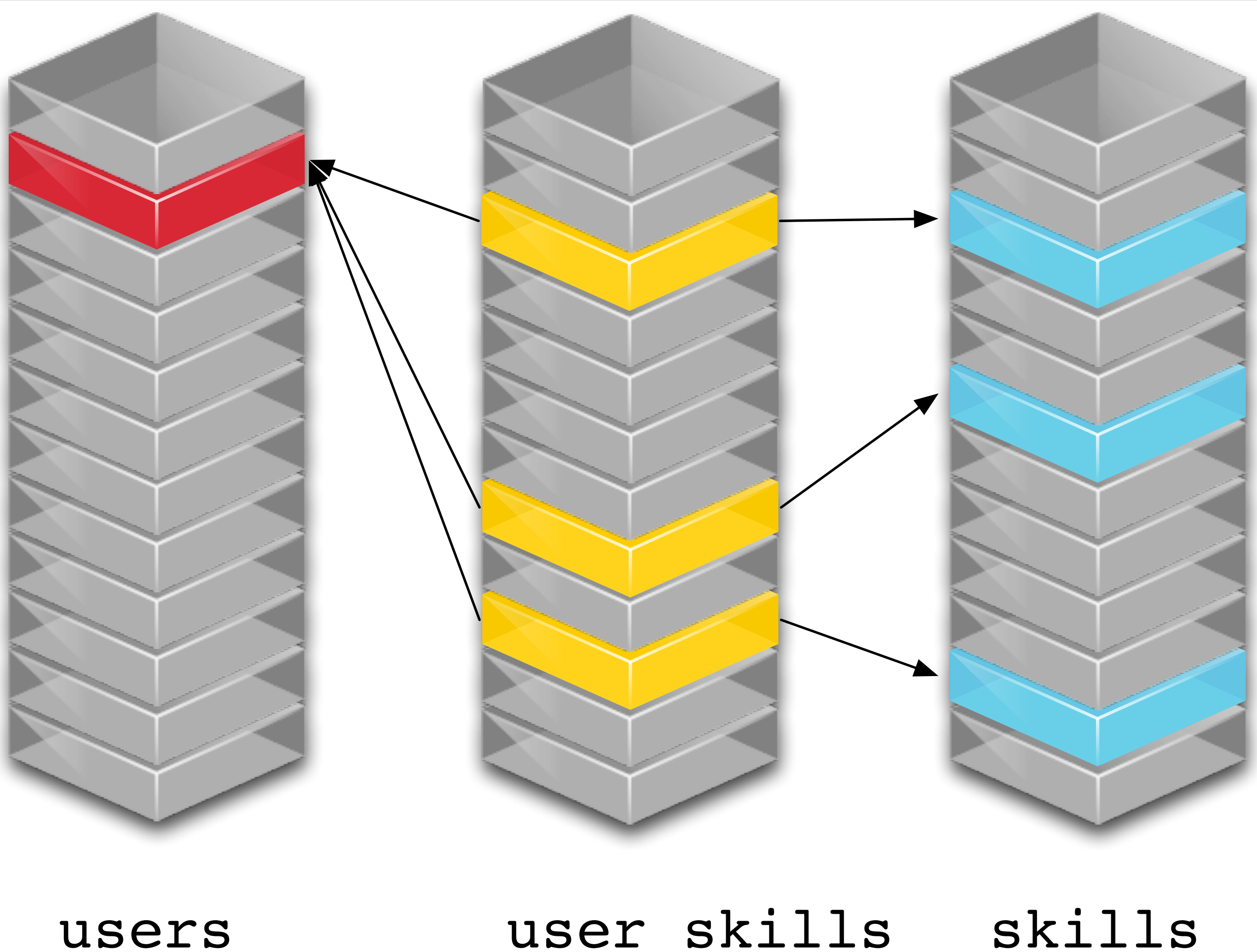
Just use SQL

Just use SQL



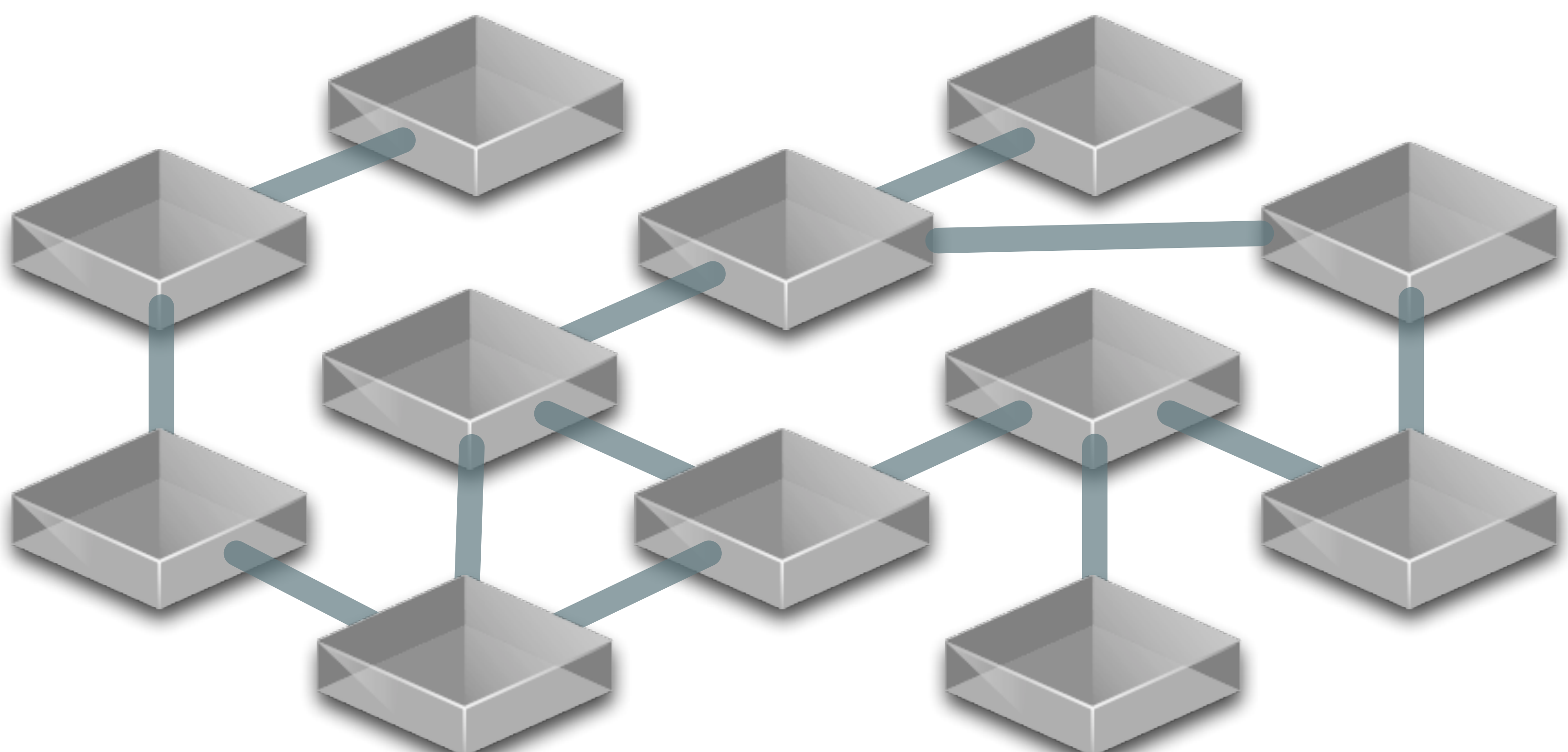
Just use SQL

```
select skills.name
from users join user_skills on (...) join skills on (...)
where users.name = "Michael"
```



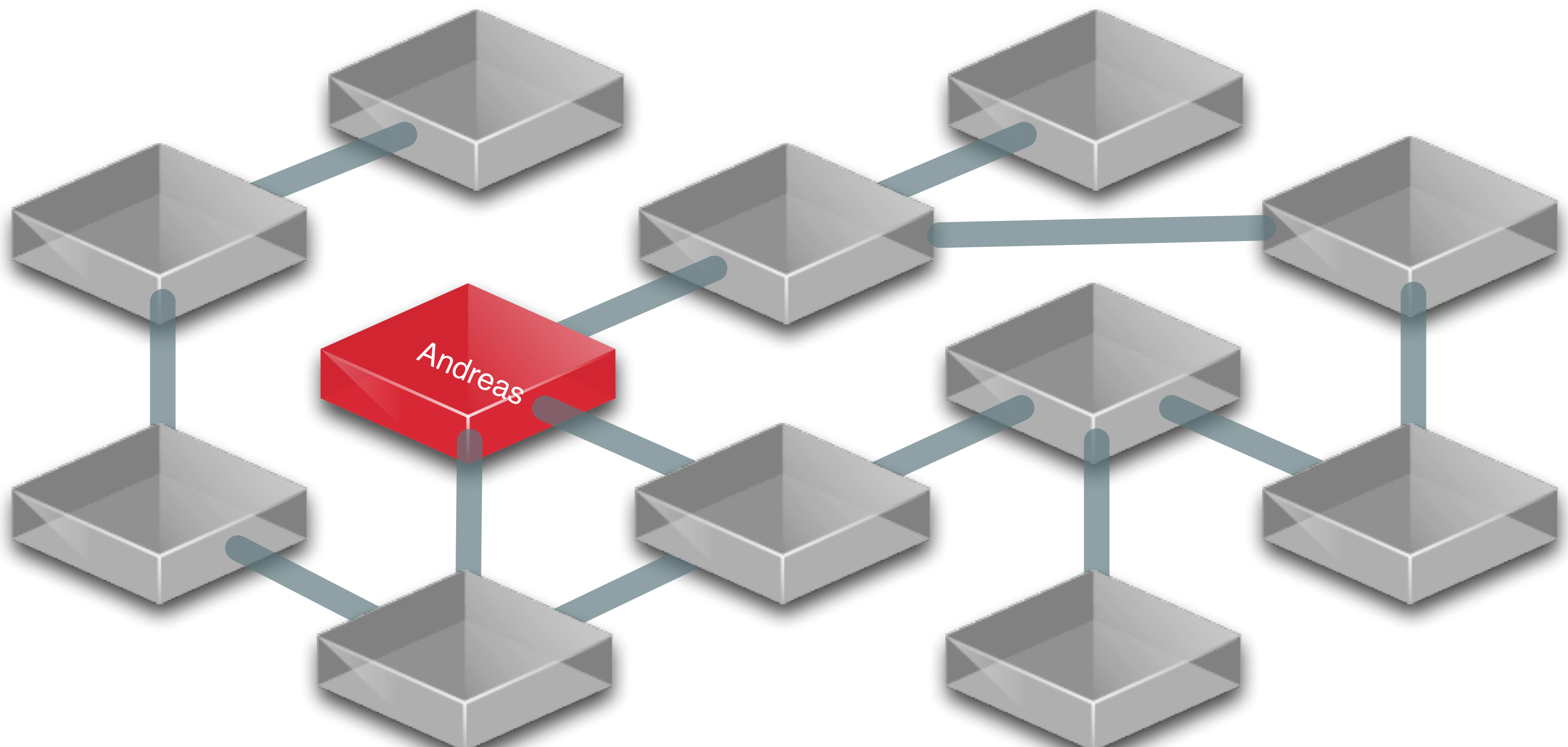
How to query a graph?

You traverse the graph



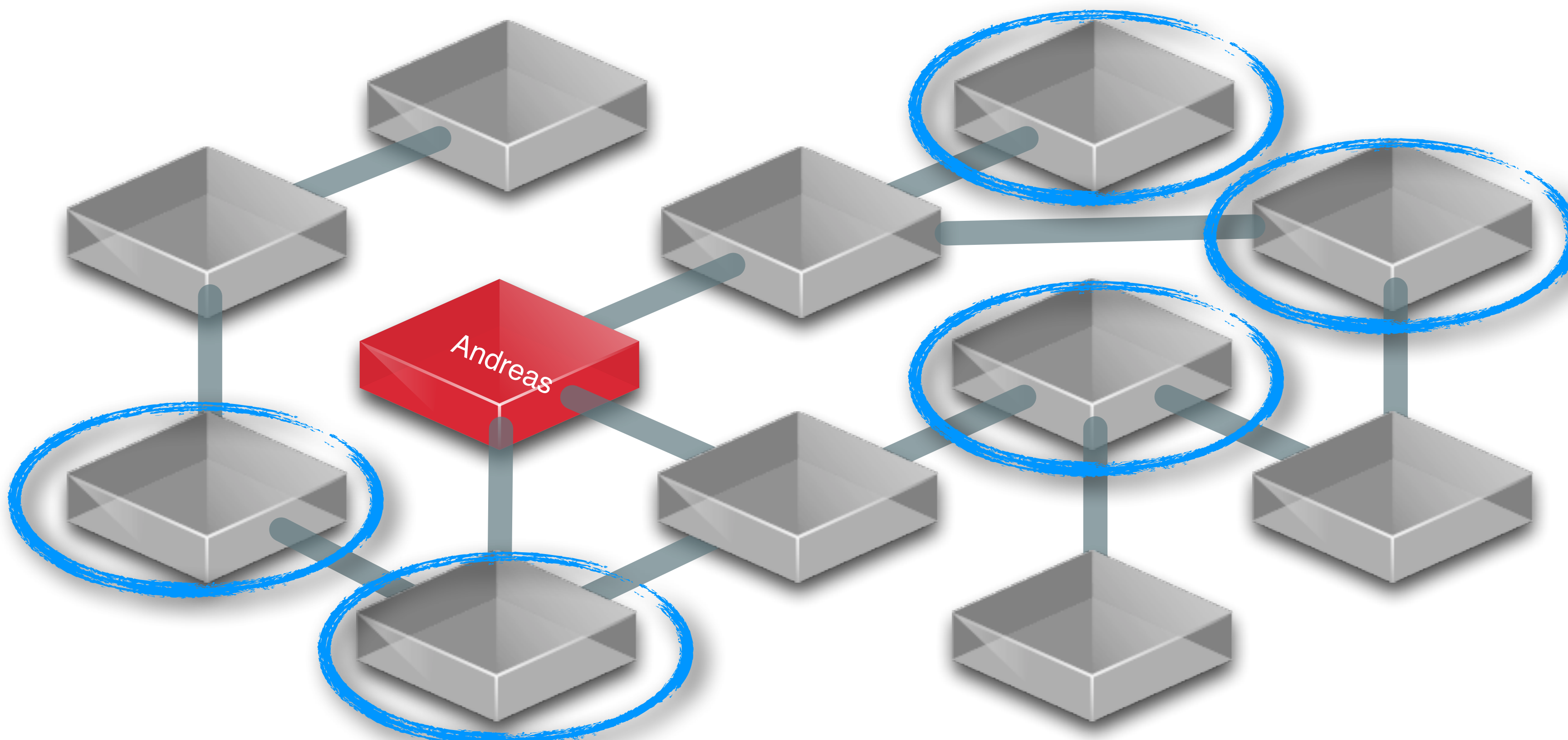
You traverse the graph

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



You traverse the graph

```
// then traverse the relationships
MATCH (me:Person {name:'Andreas'})-[:FRIEND]-(friend)
                                  -[:FRIEND]-(friend2)
RETURN friend2
```



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 (a:Actor)-[r:ACTS_IN]->(m:Movie)
RETURN a.name, r.role, m.title
```

Cypher attributes

#3 Pattern Matching

Patterns are easy for your human brain

Query Structure

Query Structure

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
LIMIT 10
```

MATCH

describes the pattern

MATCH - Pattern

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
SKIP 5 LIMIT 10
```

WHERE

filters the result set

WHERE - filter

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
SKIP 5 LIMIT 10
```

RETURN

returns the result rows

RETURN - project

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
SKIP 5 LIMIT 10
```

ORDER BY
LIMIT SKIP

sort and paginate

ORDER BY LIMIT - Paginate

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
SKIP 5 LIMIT 10
```

WITH

combines query parts

like a pipe

WITH + WHERE = HAVING

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
SKIP 5 LIMIT 10
```

Collections

powerful datastructure handling

Collections

```
MATCH (n:Label)-[:REL]->(m:Label)
WHERE n.prop < 42
WITH n, count(m) as cnt,
     collect(m.attr) as attrs
WHERE cnt > 12
RETURN n.prop,
       extract(a2 in
          filter(a1 in attrs
             WHERE a1 =~ "...-.*")
       | substr(a2,4,size(a2)-1)]
       AS ids
ORDER BY length(ids) DESC
LIMIT 10
```

Concrete Example

```
MATCH (:Country {name:"Sweden"})
      <-[:REGISTERED_IN]-(c:Company)
      <-[:WORKS_AT]-(p:Person:Developer)
WHERE p.age < 42
WITH c, count(p) as cnt,
     collect(p.empId) as emp_ids
WHERE cnt > 12
RETURN c.name AS company_name,
       extract(id2 in
          filter(id1 in emp_ids
             WHERE id1 =~ "...-.*")
       | substr(id2,4,size(id2)-1)]
       AS last_emp_id_digits
ORDER BY length(last_emp_id_digits) DESC
SKIP 5 LIMIT 10
```

CREATE

creates nodes, relationships and patterns

CREATE - nodes, rels, structures

```
CREATE (y:Year {year:2014})
FOREACH (m IN range(1,12) |
  CREATE
   (:Month {month:m})-[:IN]->(y)
)
```

MERGE

matches or creates

MERGE - get or create

```
MERGE (y:Year {year:2014})
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

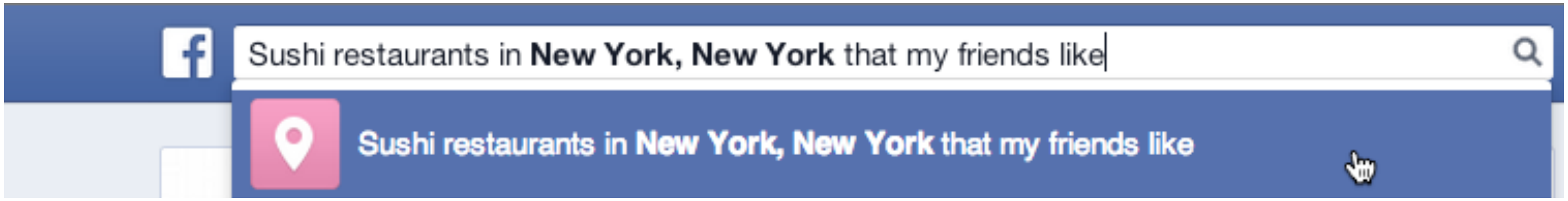
represent optional schema

INDEX / CONSTRAINT

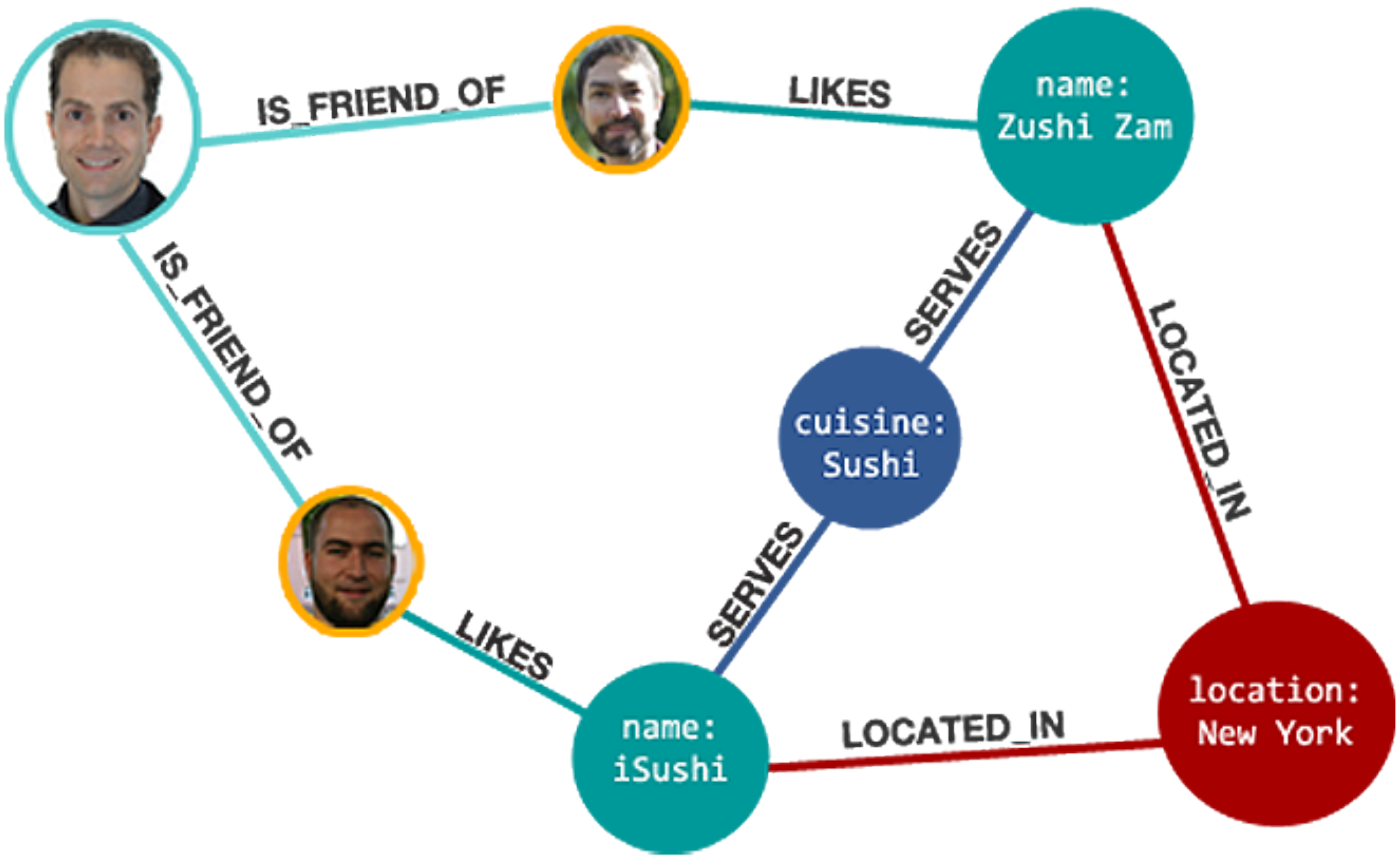
```
CREATE CONSTRAINT ON (y:Year)
   ASSERT y.year IS UNIQUE

CREATE INDEX ON :Month(month)
```

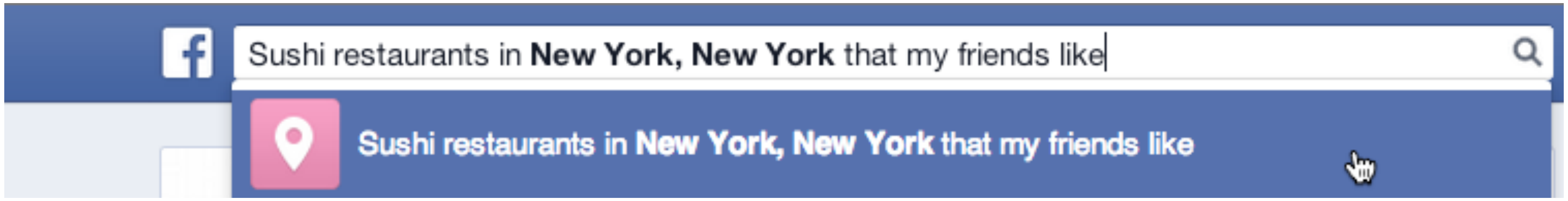
Graph Query Examples

Social Recommendation


Sushi restaurants in **New York, New York** that my friends like
Sushi restaurants in **New York, New York** that my friends like


Sushi restaurants in **New York, New York** that my friends like
Sushi restaurants in **New York, New York** that my friends like


IS_FRIEND_OF
LIKES
name: Zushi Zam
IS_FRIEND_OF
SERVES
cuisine: Sushi
LOCATED_IN
SERVES
LIKES
name: iSushi
LOCATED_IN
location: New York

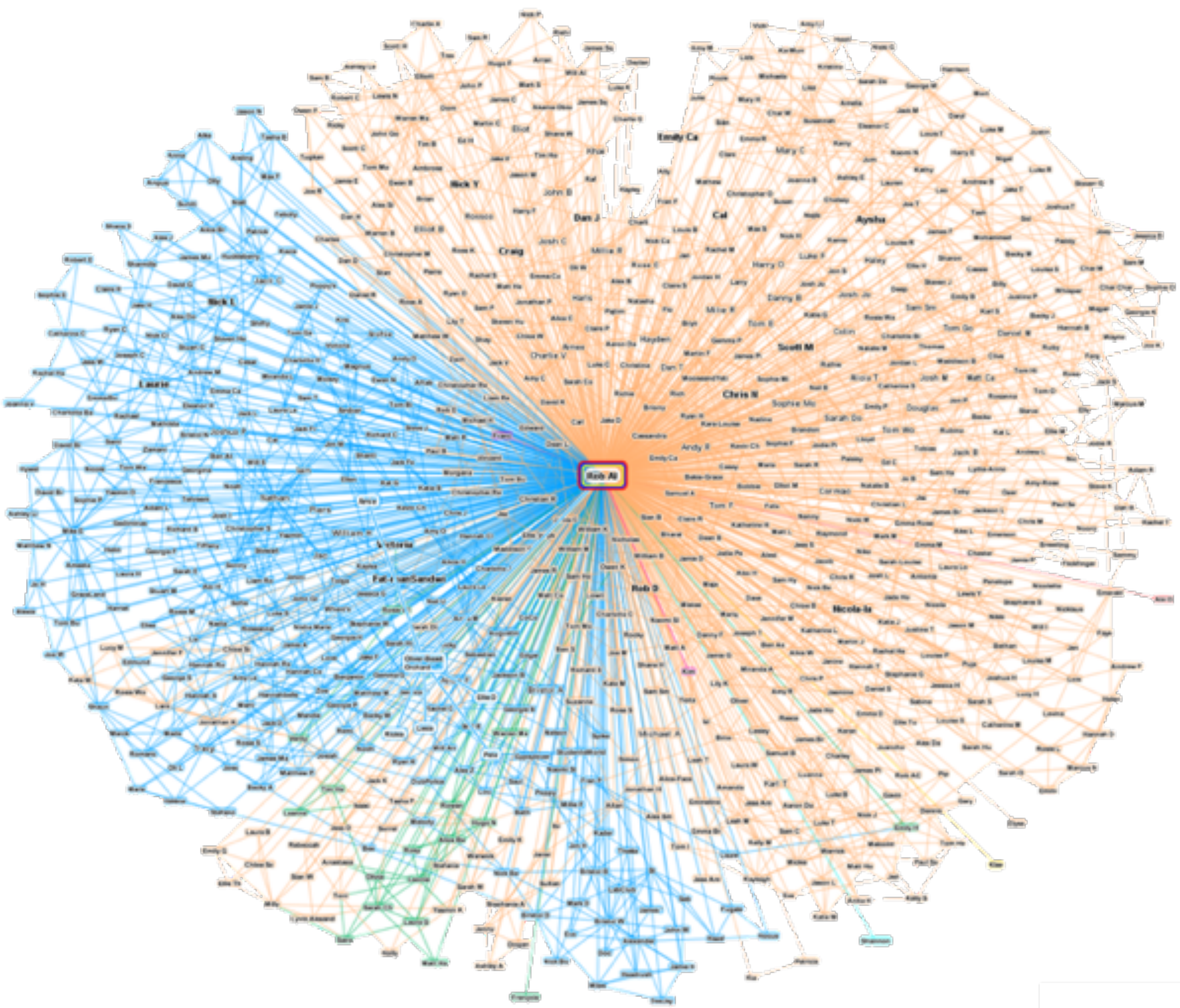


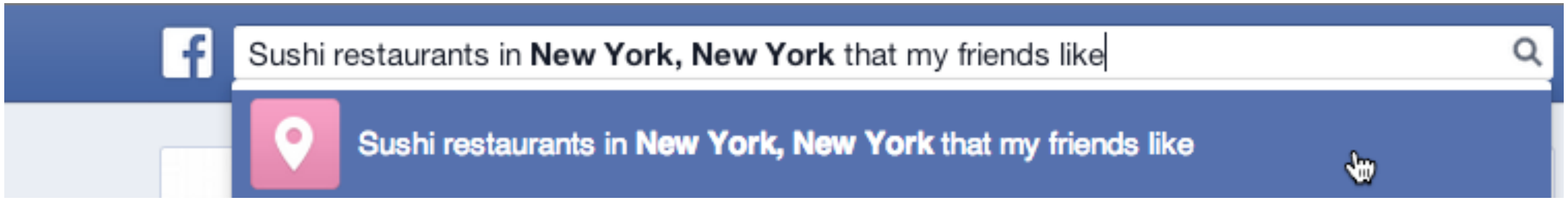

```
MATCH  (person:Person)-[:IS_FRIEND_OF]->(friend),
        (friend)-[:LIKES]->(restaurant),
       (restaurant)-[:LOCATED_IN]->(loc:Location),
       (restaurant)-[:SERVES]->(type:Cuisine)

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

RETURN restaurant.name
```

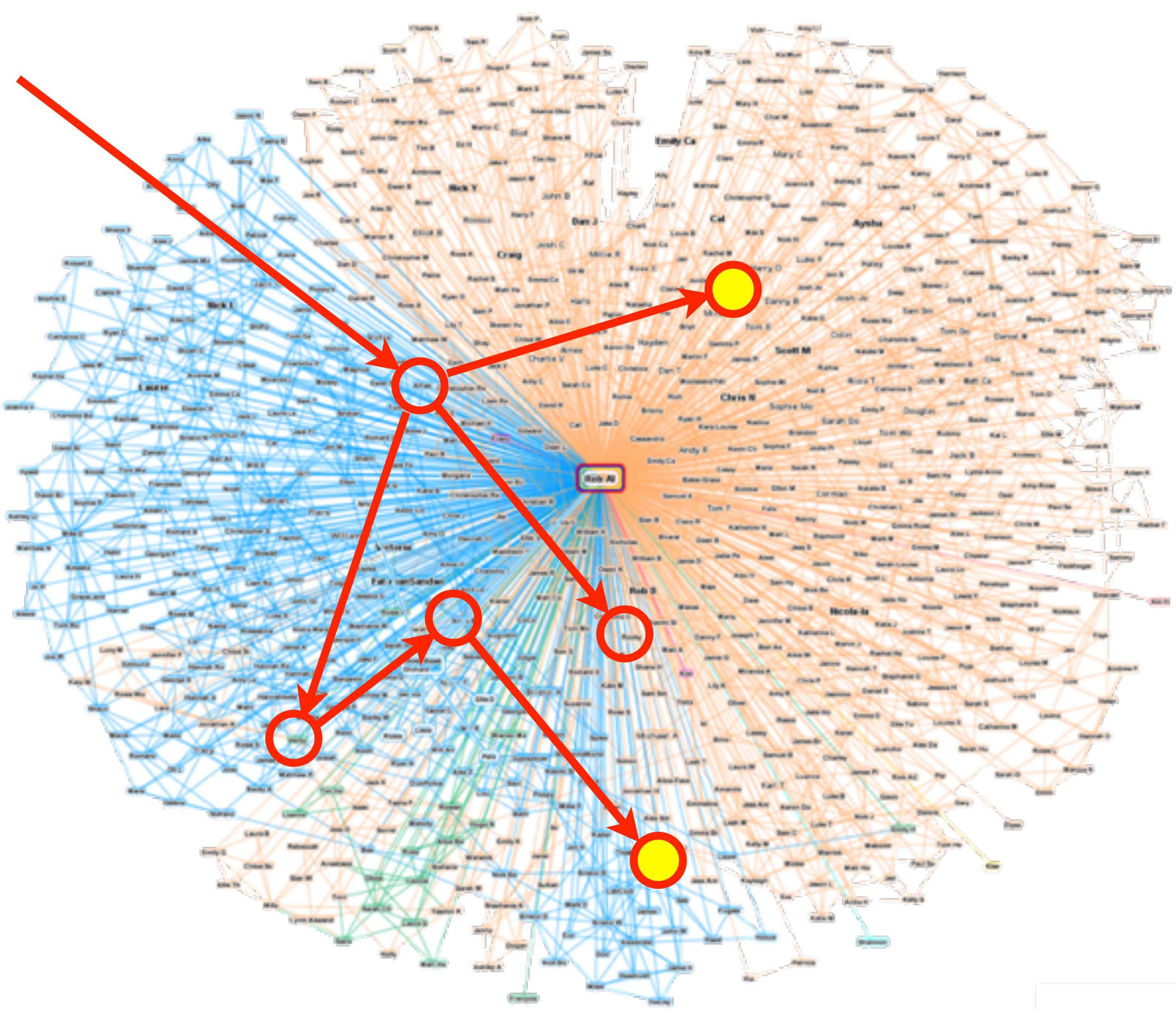
http://maxdemarzi.com/?s=facebook

** Cypher query language example*


Sushi restaurants in **New York, New York** that my friends like
Sushi restaurants in **New York, New York** that my friends like




Sushi restaurants in **New York, New York** that my friends like
Sushi restaurants in **New York, New York** that my friends like

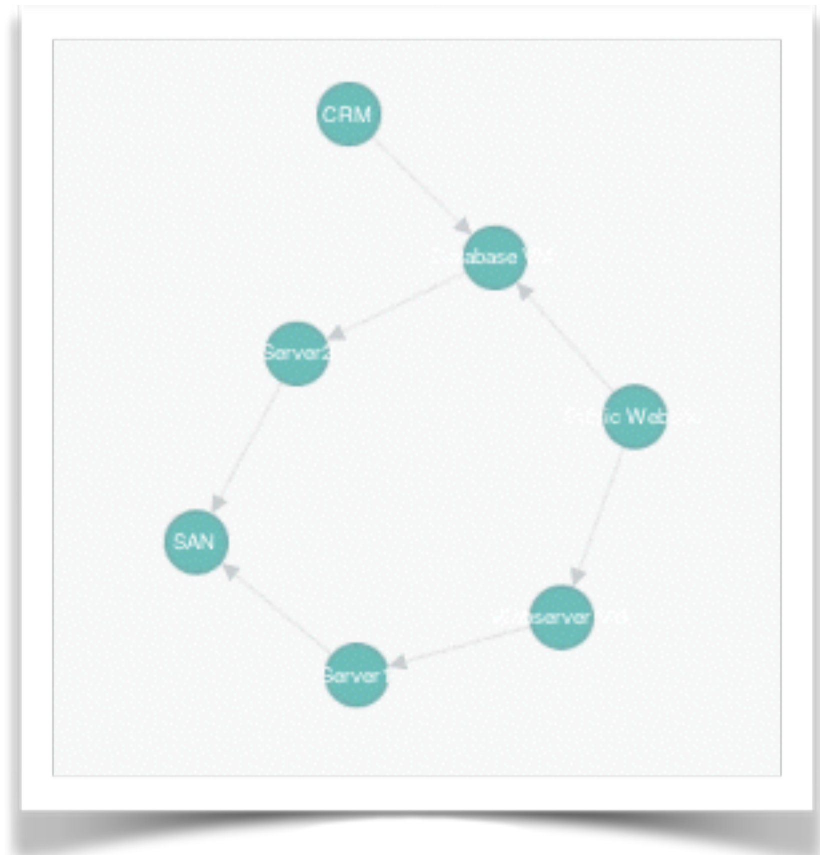


Network Management Example

Practical Cypher

Network Management - Create

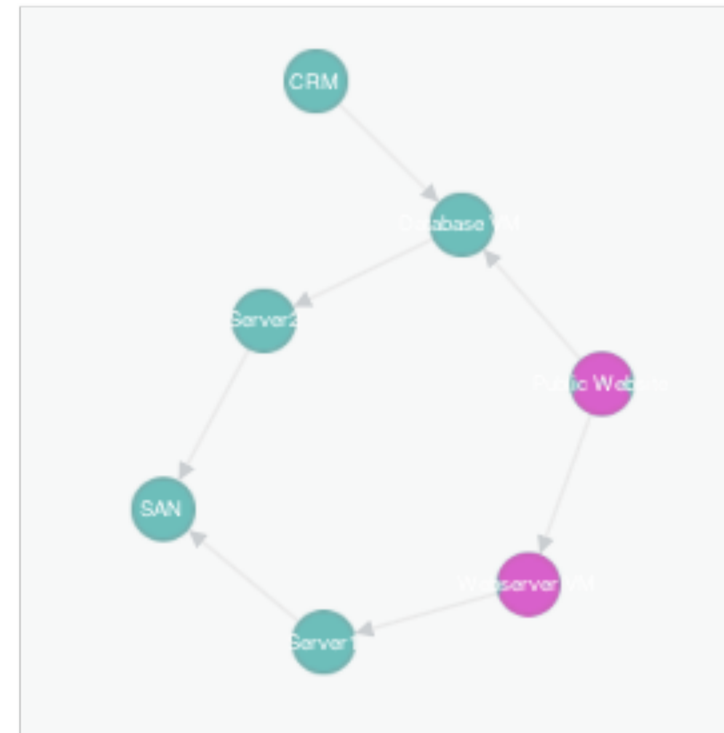
```
CREATE
    (crm {name:"CRM"}),
    (dbvm {name:"Database VM"}),
    (www {name:"Public Website"}),
    (wwwvm {name:"Webserver VM"}),
    (srv1 {name:"Server 1"}),
    (san {name:"SAN"}),
    (srv2 {name:"Server 2"}),

    (crm)-[:DEPENDS_ON]->(dbvm),
    (dbvm)-[:DEPENDS_ON]->(srv2),
    (srv2)-[:DEPENDS_ON]->(san),
    (www)-[:DEPENDS_ON]->(dbvm),
    (www)-[:DEPENDS_ON]->(wwwvm),
    (wwwvm)-[:DEPENDS_ON]->(srv1),
    (srv1)-[:DEPENDS_ON]->(san)
```



Practical Cypher

Network Management - Impact Analysis

```
// Server 1 Outage
MATCH (n)<-[:DEPENDS_ON*]-(upstream)
WHERE n.name = "Server 1"
RETURN upstream
```

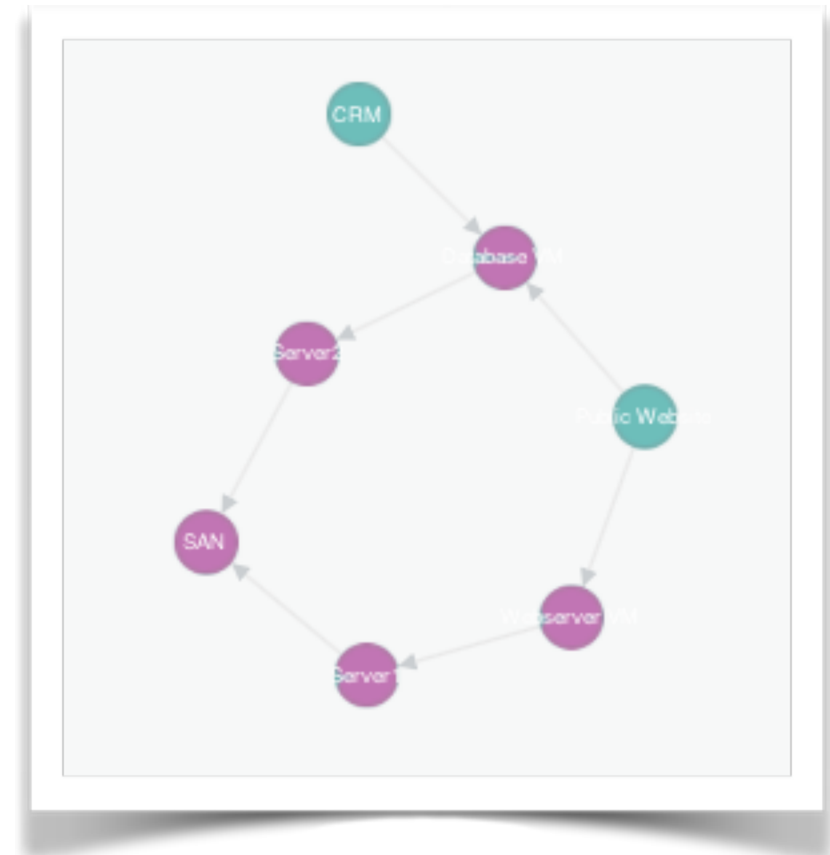


upstream
{name:"Webserver VM"}
{name:"Public Website"}

Practical Cypher

Network Management - Dependency Analysis

```
// Public website dependencies
MATCH (n)-[:DEPENDS_ON*]->(downstream)
WHERE n.name = "Public Website"
RETURN downstream
```

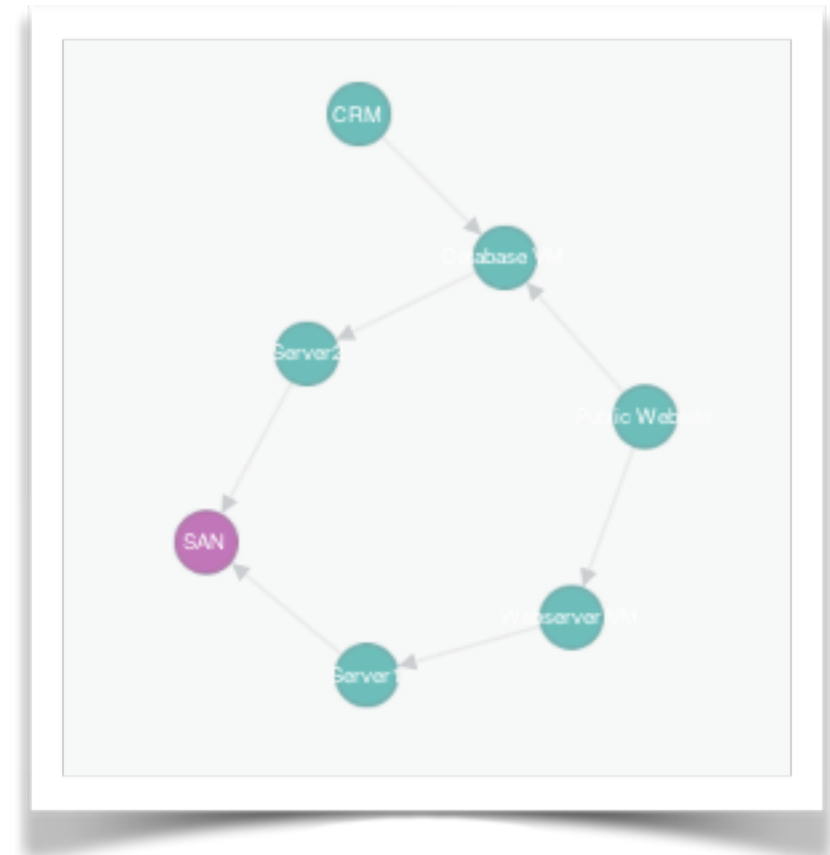


downstream
{name:"Database VM"}
{name:"Server 2"}
{name:"SAN"}
{name:"Webserver VM"}
{name:"Server 1"}

Practical Cypher

Network Management - Statistics

```
// Most depended on component
MATCH (n)<-[:DEPENDS_ON*]-(dependent)
RETURN n,
  count(DISTINCT dependent)
    AS dependents
ORDER BY dependents DESC
LIMIT 1
```



n	dependents
{name:"SAN"}	6

How to get started?

- Full day Neo4j Training & Online Training
- Free e-Books
  - Graph Databases, Neo4j 2.0 (DE)
- neo4j.org
  - http://neo4j.org/develop/modeling
- docs.neo4j.org
  - Data Modeling Examples
- http://console.neo4j.org
- http://gist.neo4j.org
- Get Neo4j
  - http://neo4j.org/download
- Participate
  - http://groups.google.com/group/neo4j

neotechnology
graphs are everywhere

Brown Bag Lunch

By request only!

- *you* bring 10+ colleagues
- *you* provide a room with a projector + screen

- *we* bring a bag lunch
- *we* introduce Neo4j to your team in 45 min + 15 min for Q&A

http://neotechnology.com/brownbag

Schedule your Neo4j Intro now!

Thank You

Time for Questions!