Introduction to Neo4j

Stefan Armbruster
@darthvader42
stefan.armbruster@neotechnology.com

most Slides from:
Michael Hunger

The Path Forward

1. No .. NO .. NOSQL
2. Why graphs?
3. What's a graph database?
4. Some things about Neo4j.
5. How do people use Neo4j?

NOSQL

Neo4j

What is NOSQL?

Neo4j

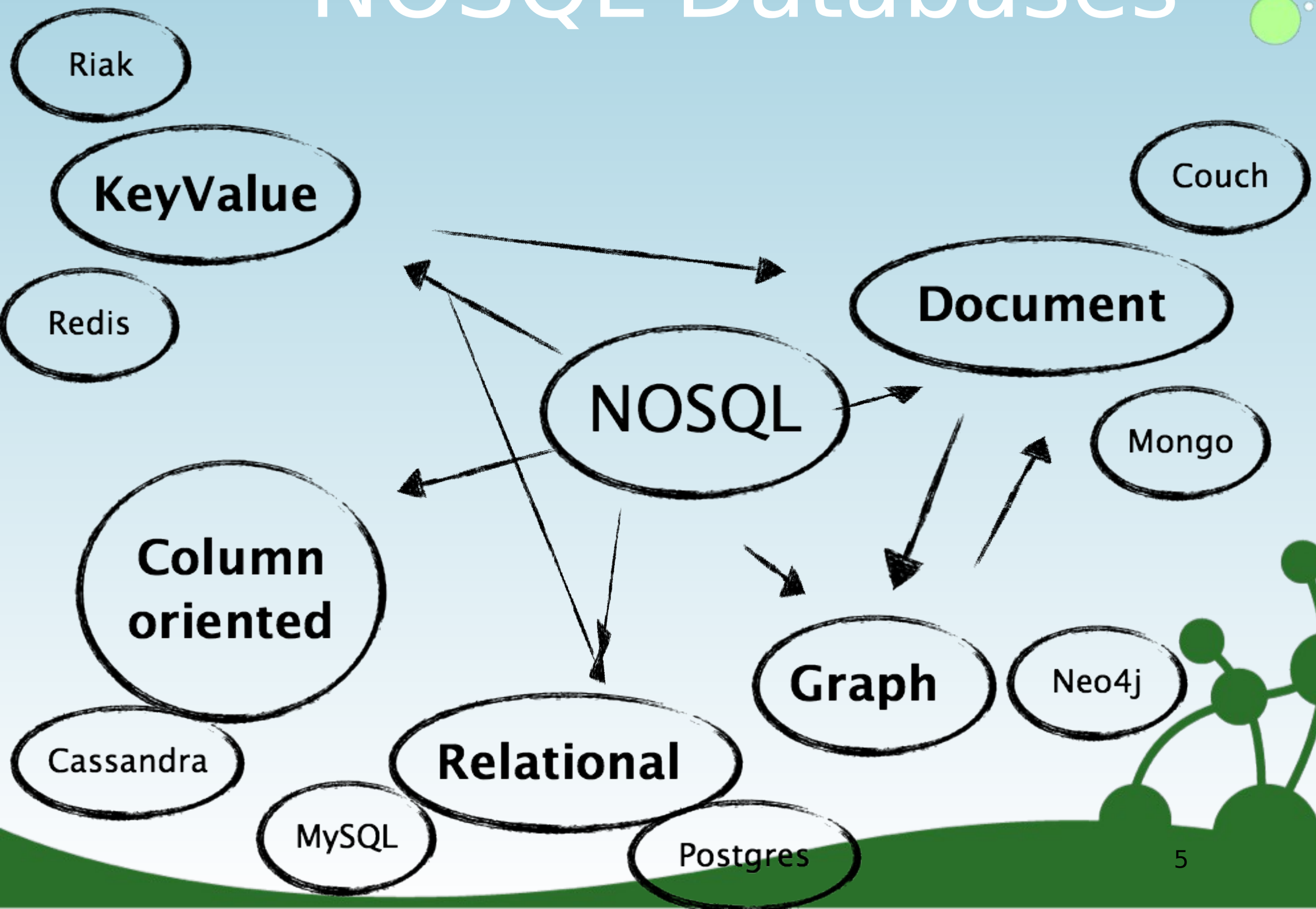
It's not "No to SQL"

It's not "Never SQL"

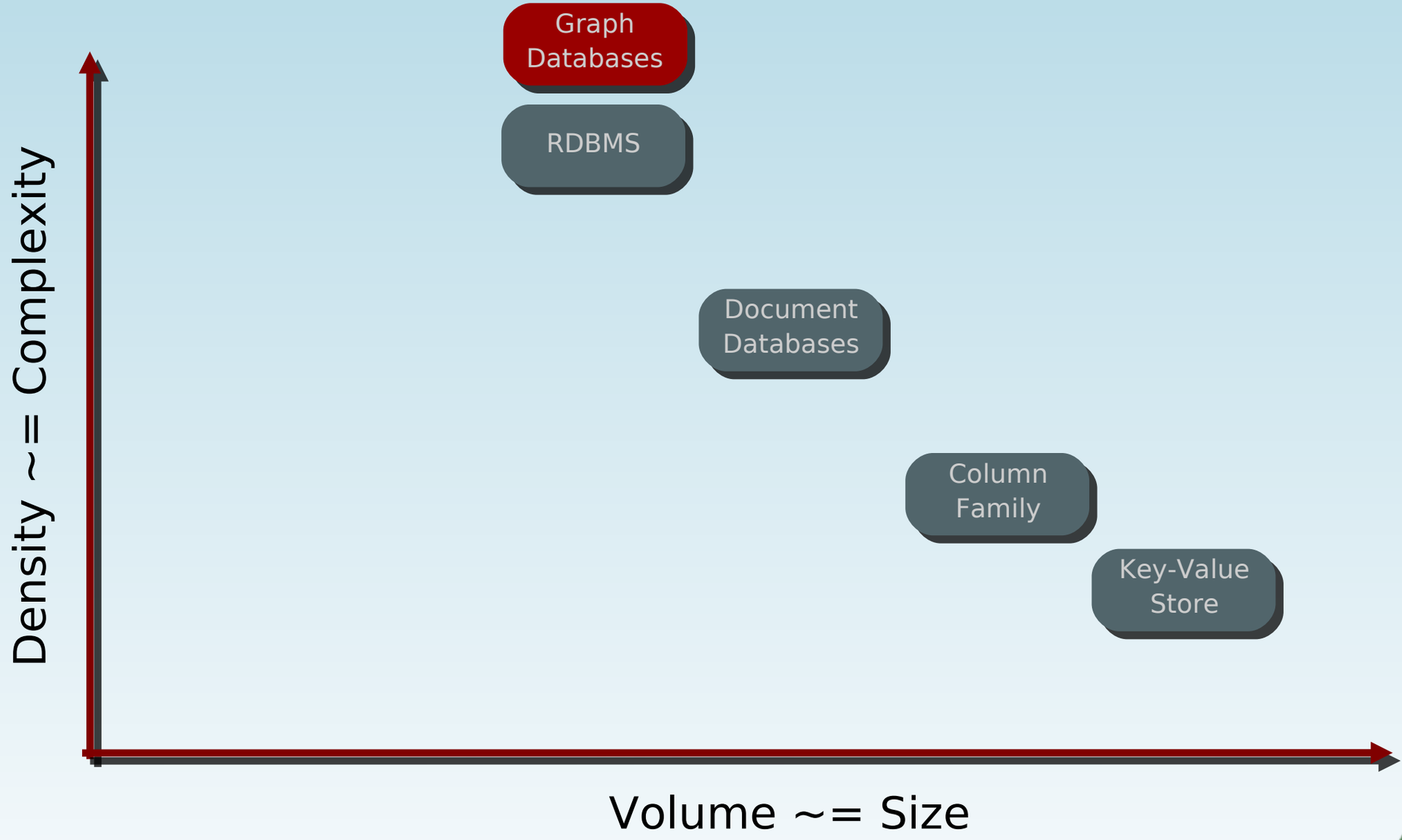
It's "Not Only SQL"

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



Trends in BigData & NOSQL

1. increasing data size (big data)

- *"Every 2 days we create as much information as we did up to 2003"*
  - Eric Schmidt

2. increasingly connected data (graph data)

- *for example, text documents to html*

3. semi-structured data

- *individualization of data, with common sub-set*

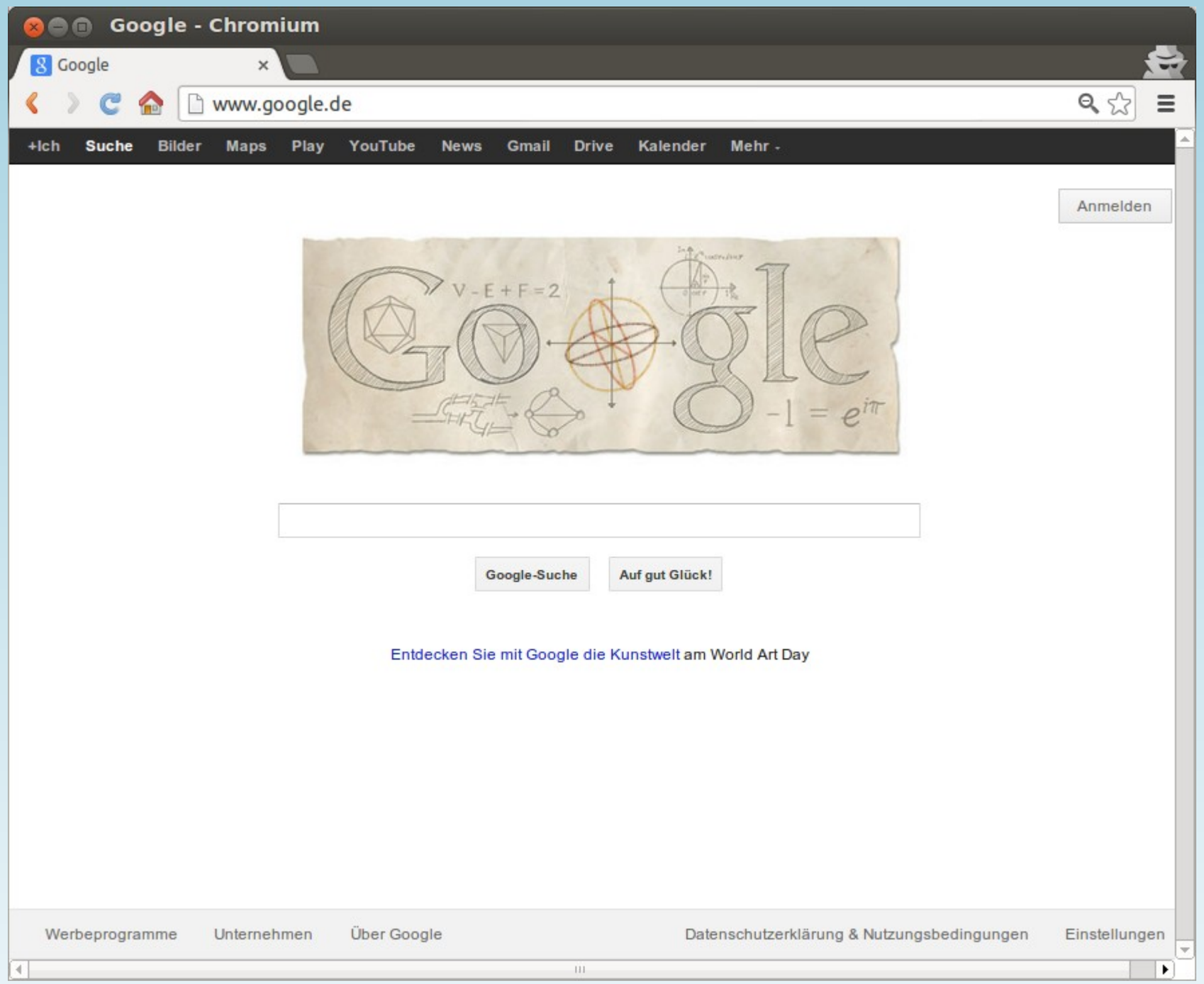
4. architecture - a facade over multiple services

- *from monolithic to modular, distributed applications*

A Graph?

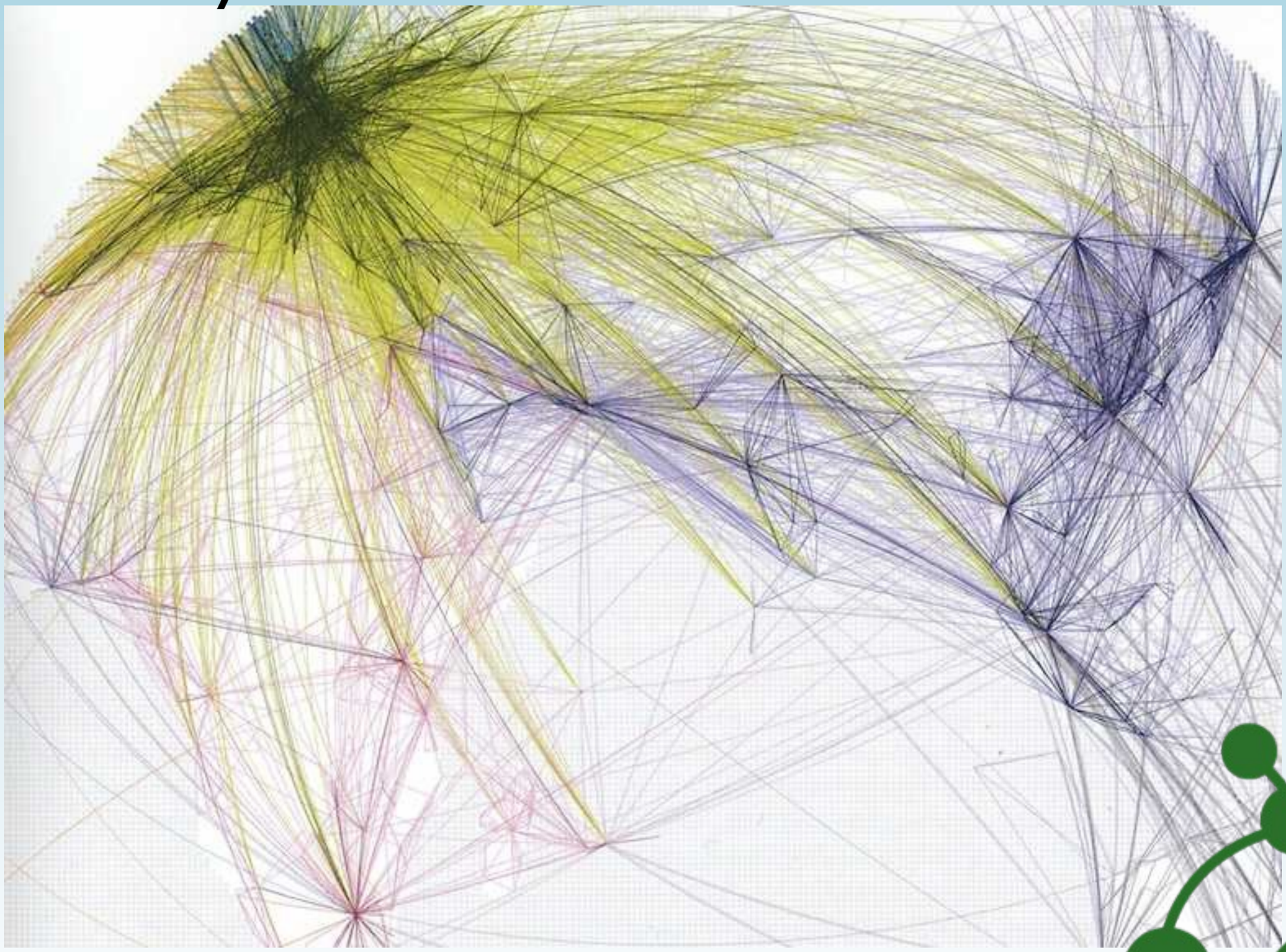
Yes, a graph

Leonhard Euler 1707-1783



Apr 15th 2013: Google Doodle for Euler's birthday

They are everywhere



Flight Patterns

Graphs Everywhere

- *Relationships in*
  - *Politics, Economics, History, Science, Transportation*
- *Biology, Chemistry, Physics, Sociology*
  - *Body, Ecosphere, Reaction, Interactions*
- *Internet*
  - *Hardware, Software, Interaction*
- *Social Networks*
  - *Family, Friends*
  - *Work, Communities*
  - *Neighbours, Cities, Society*

Good Relationships

- *the world is rich, messy and related data*
- *relationships are as least as important as the things they connect*
- *Graphs = Whole > Σ parts*
- *complex interactions*
- *always changing, change of structures as well*
- *Graph: Relationships are part of the data*
- *RDBMS: Relationships part of the fixed schema*

Questions and Answers

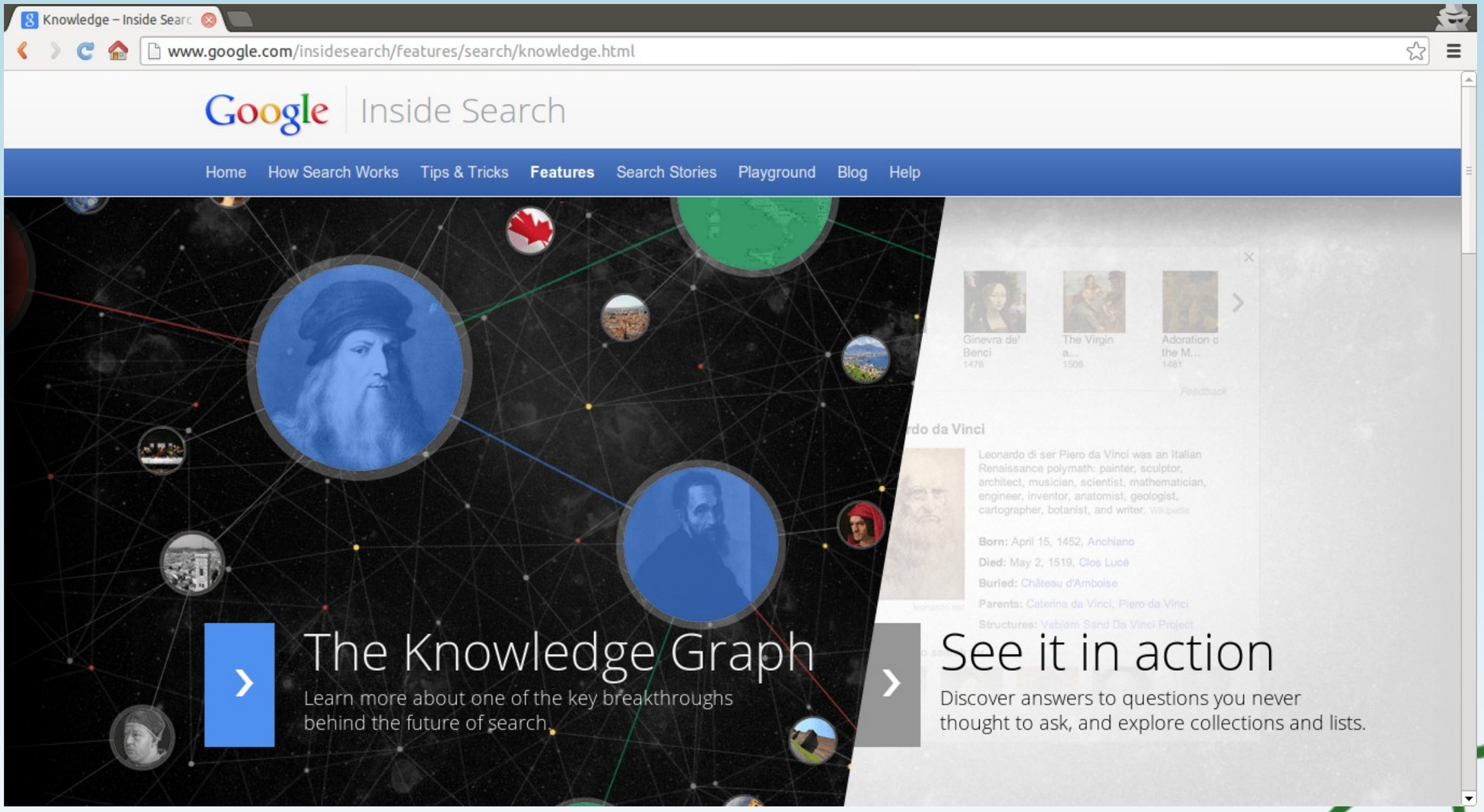
- *Complex Questions*
- *Answers lie between the lines (things)*
- *Locality of the information*
- *Global searches / operations very expensive*
- *constant query time, regardless of data volume*

Categories ?

- *Categories == Classes, Trees ?*
- *What if more than one category fits?*
- *Tags*
- *Categories vi relationships like „IS_A“*
- *any number, easy change*
- *„virtual“ Relationships - Traversals*
- *Category dynamically derived from queries*

Everyone is talking about graphs...

Everyone is talking about graphs...



Each of us has not only one graph, but many!

Graph DB 101

A graph database...

***NO**: not for charts & diagrams, or vector artwork*

***YES**: for storing data that is structured as a graph*

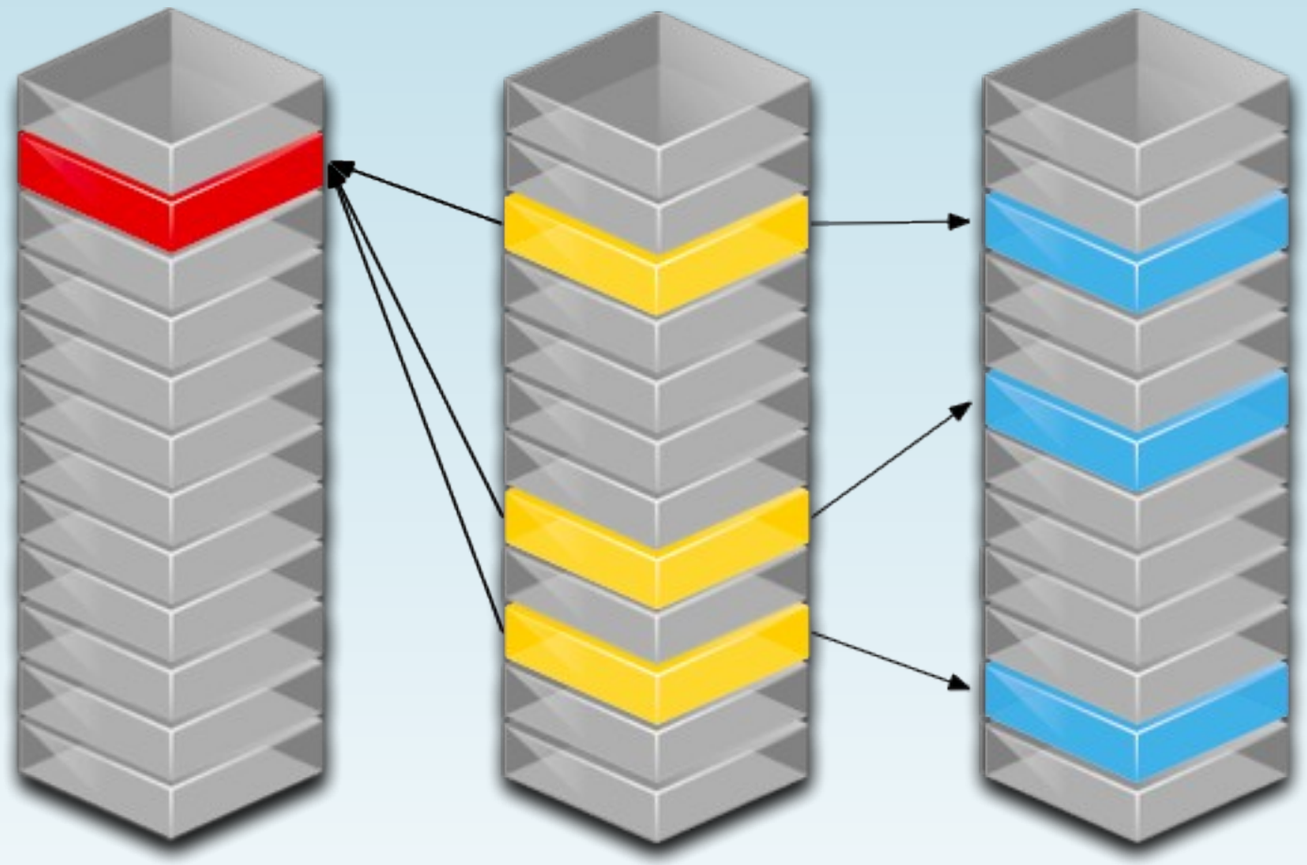
remember linked lists, trees?

graphs are the general-purpose data structure

"A relational database may tell you the average age of everyone in this session,

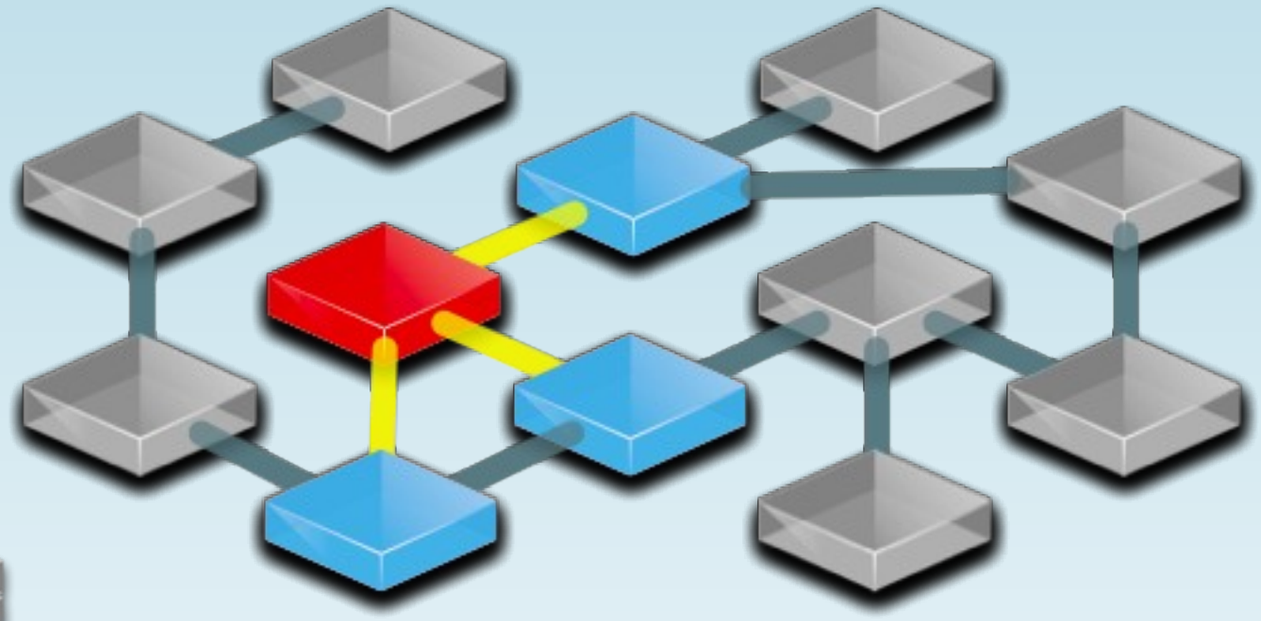
but a graph database will tell you who is most likely to buy you a beer."

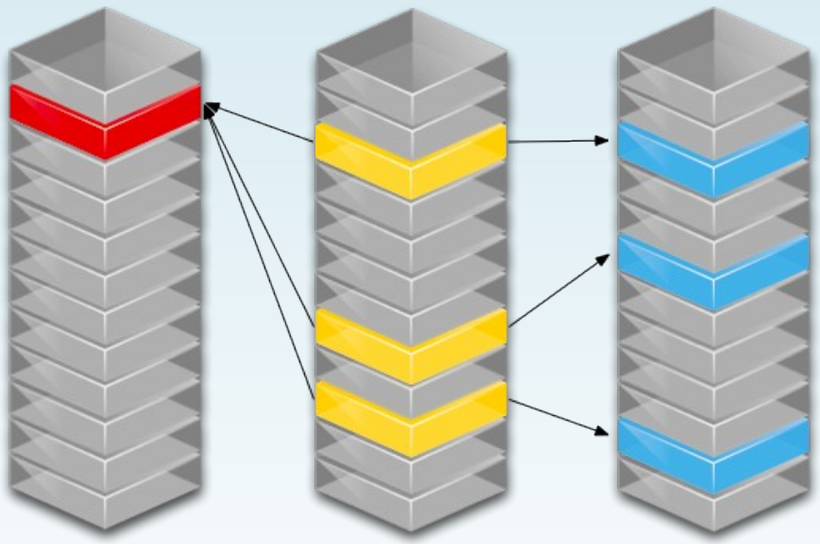
You know relational



foo foo_bar bar

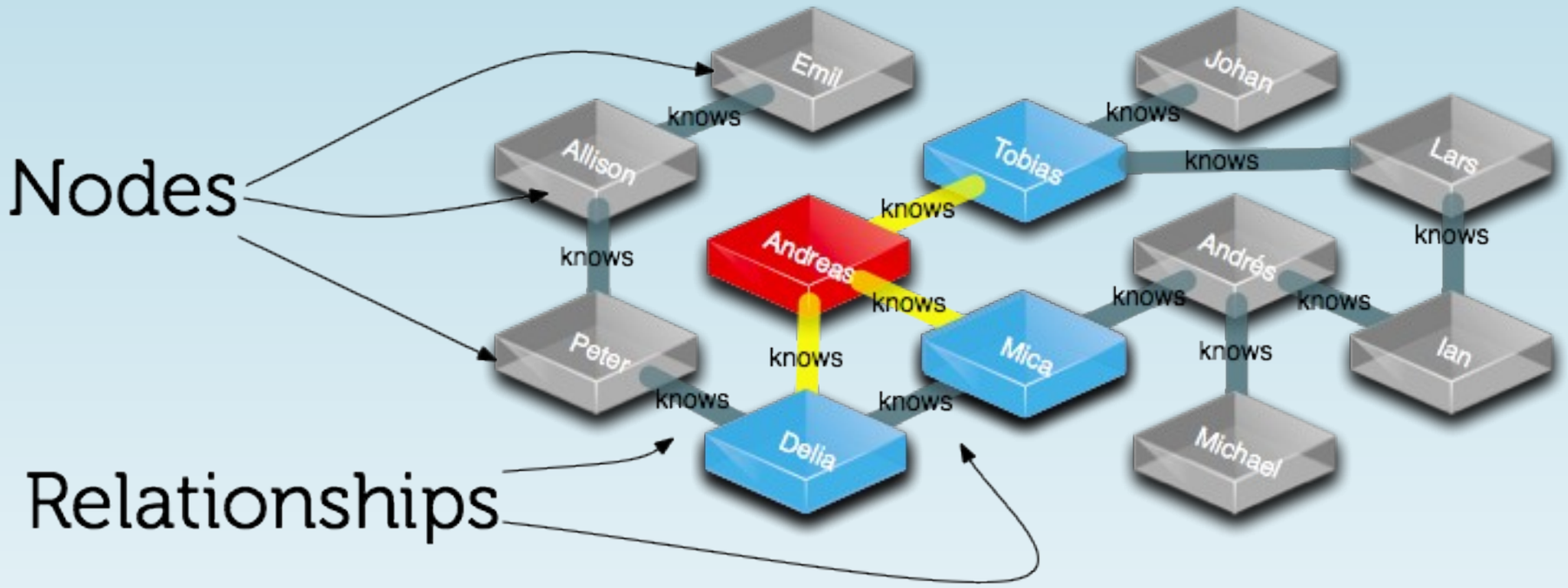
now consider relationships...





foo foo_bar bar

We're talking about a Property Graph



Properties (each a key+value)

+ Indexes (for easy look-ups)

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

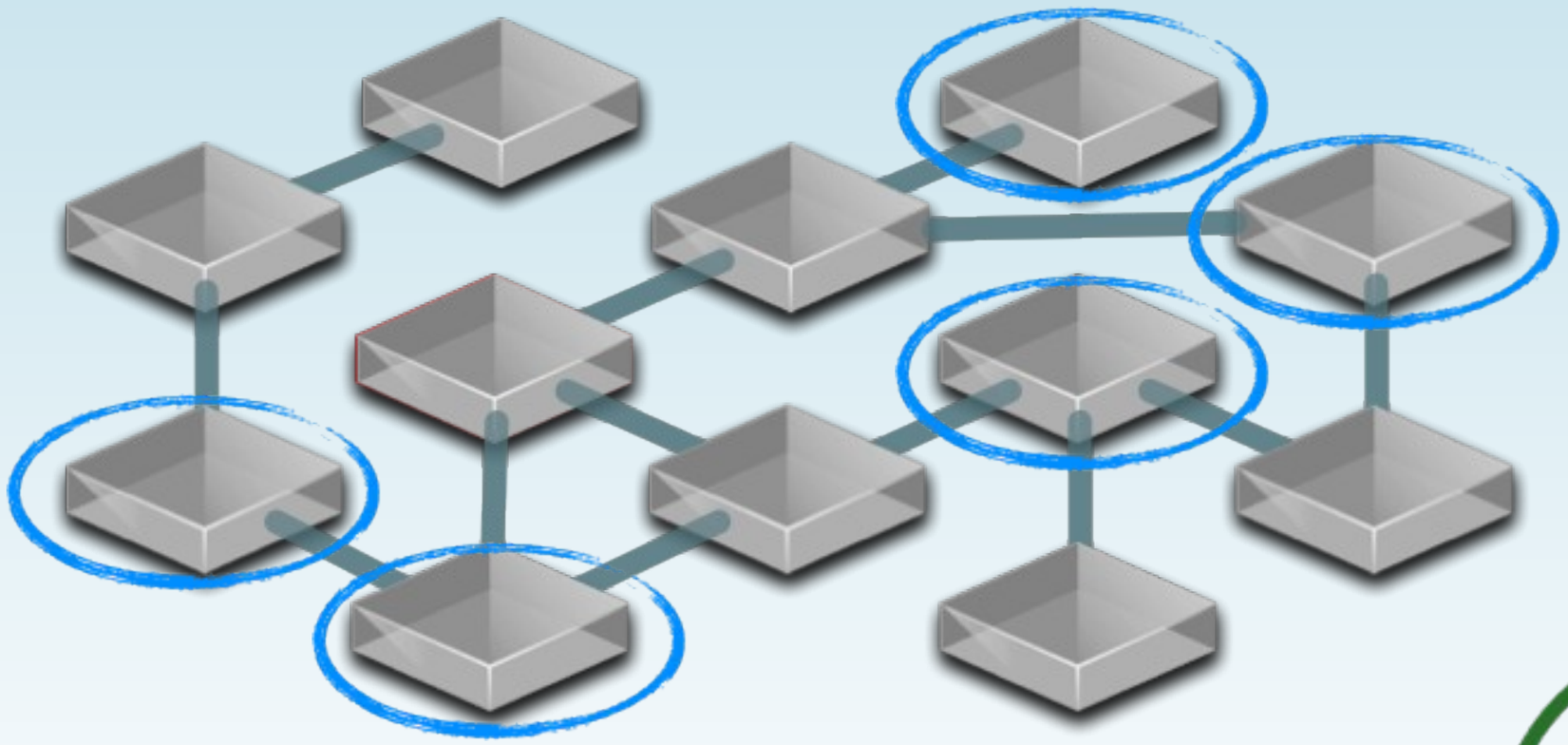
Graph Database: Pros & Cons

- *Strengths*
  - *Powerful data model, as general as RDBMS*
  - *Fast, for connected data*
  - *Easy to query*
- *Weaknesses:*
  - *Sharding (though they can scale reasonably well)*
    - *also, stay tuned for developments here*
  - *Requires conceptual shift*
    - *though graph-like thinking becomes addictive*

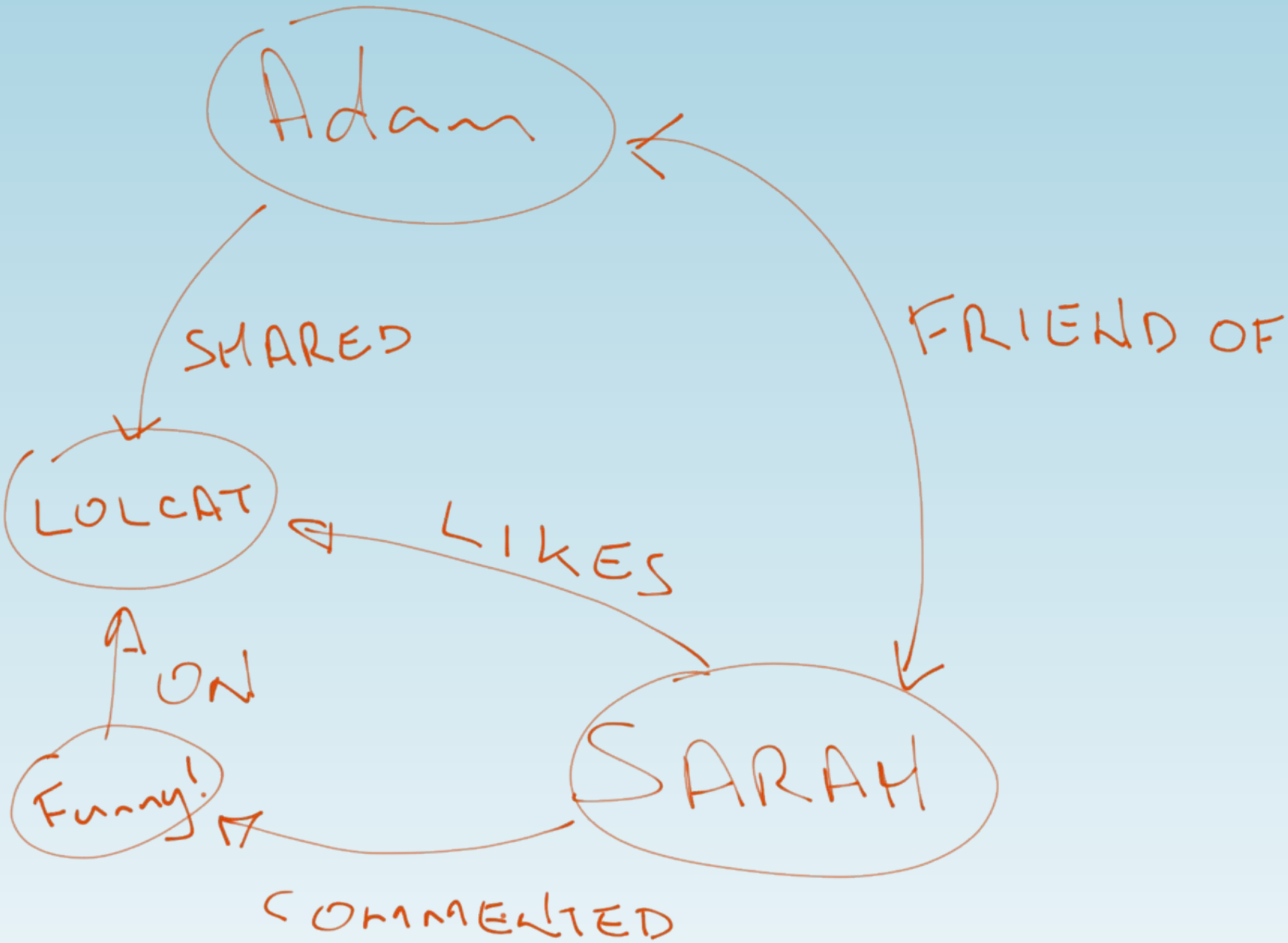
And, but, so how do you query this "graph" database?

Query a graph with a traversal

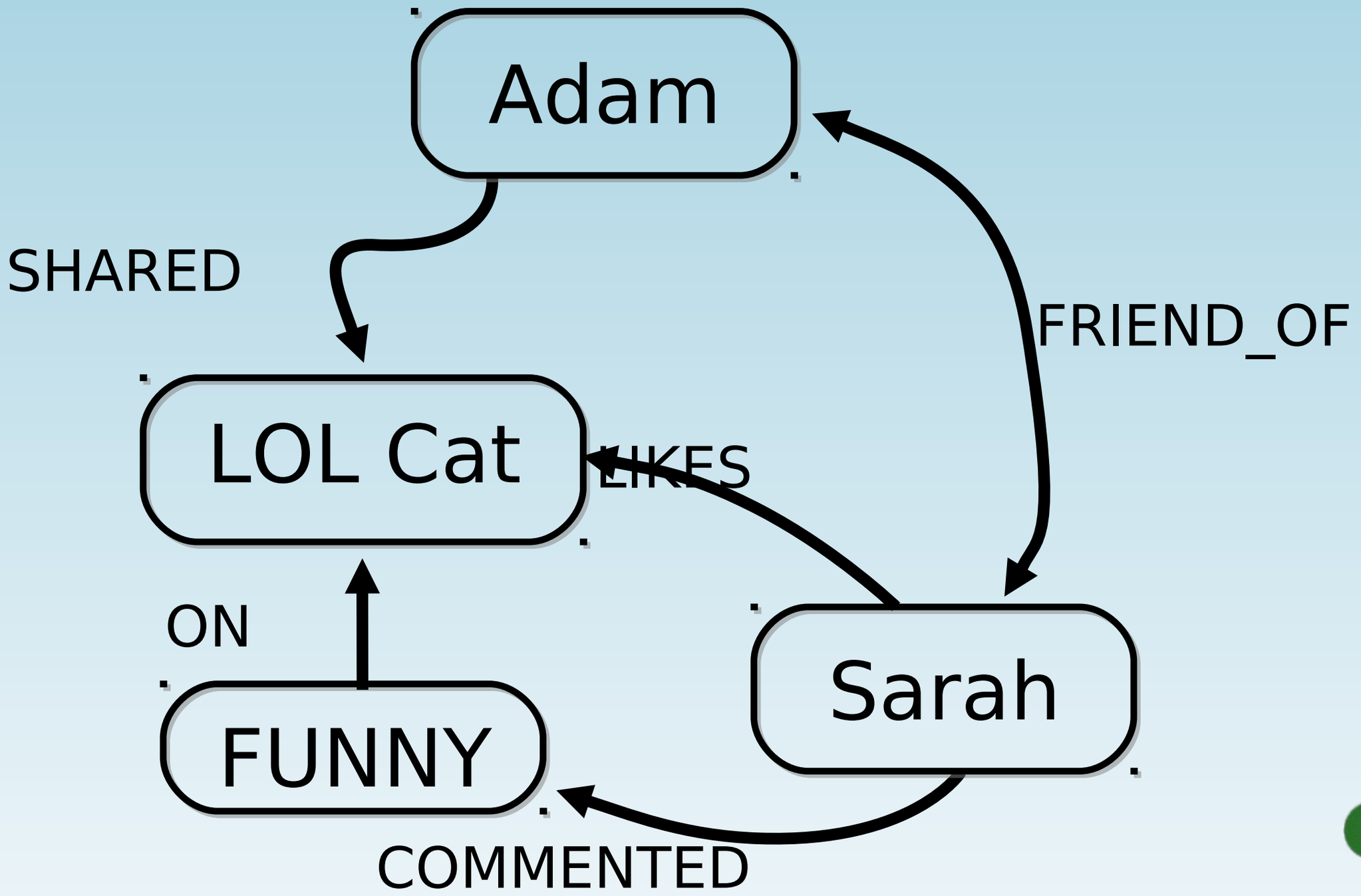
```
// then traverse to find results
start n=node:People(name = 'Andreas')
match (n)--()--(foaf) return foaf
```



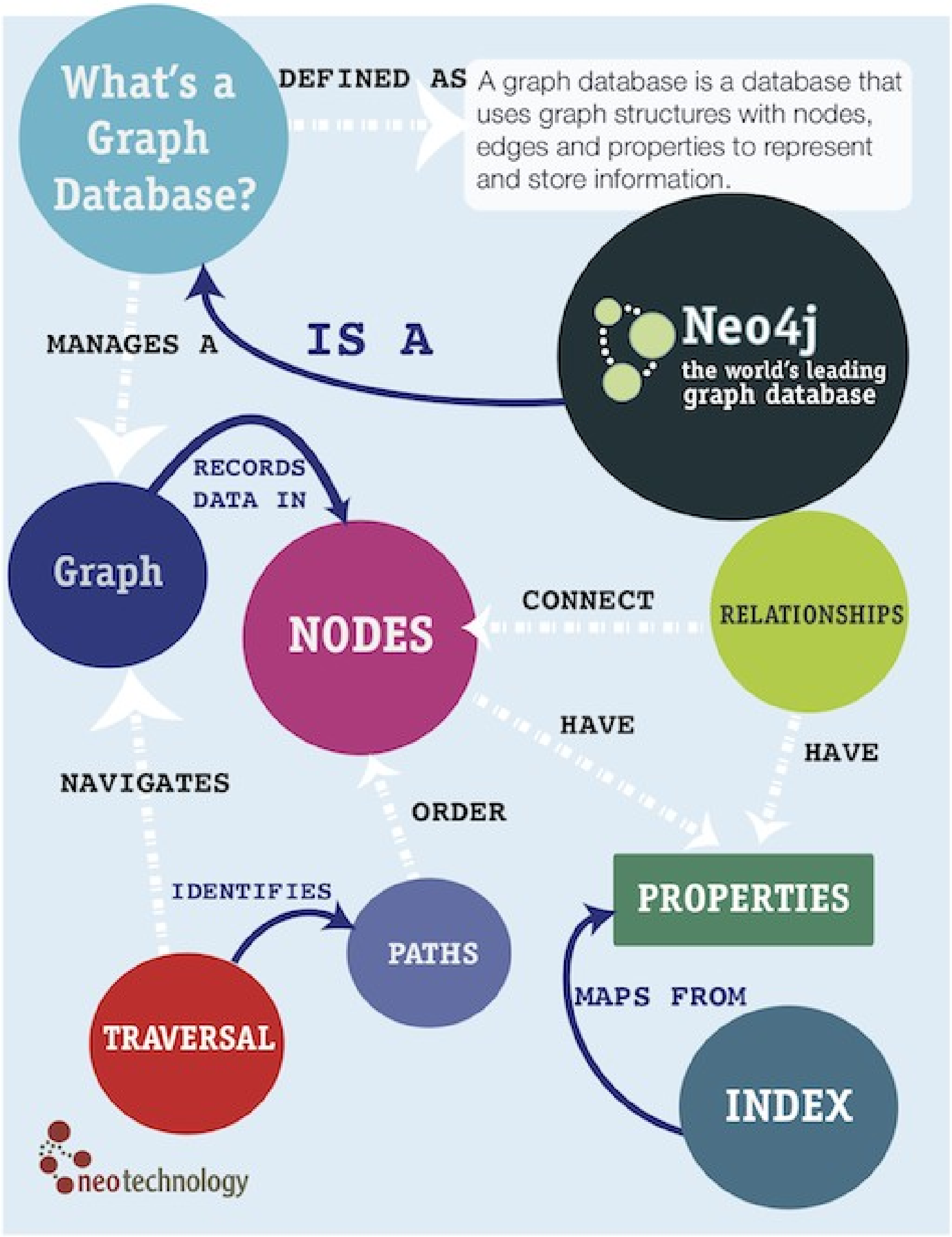
Modeling for graphs


Adam
SHARED
FRIEND OF
LOLCAT
LIKES
ON
Funny!
SARAH
COMMENTED


Adam
SHARED
FRIEND_OF
LOL Cat
LIKES
ON
Sarah
FUNNY
COMMENTED


Adam
SHARED
FRIEND_OF
LOL Cat
LIKES
ON
Sarah
FUNNY
COMMENTED

Neo4j - the Graph Database


What's a Graph Database?
DEFINED AS
A graph database is a database that uses graph structures with nodes, edges and properties to represent and store information.
MANAGES A
IS A
Neo4j
the world's leading graph database
RECORDS DATA IN
Graph
NODES
CONNECT
RELATIONSHIPS
HAVE
HAVE
NAVIGATES
ORDER
IDENTIFIES
PATHS
PROPERTIES
MAPS FROM
TRAVERSAL
INDEX
neotechnology

(Neo4j) -[:IS_A]-> (Graph Database)

Shardin g

Master/Sl ave

1 M/s

Lucene

Index

HIGH_AVAIL.

TRAVERSALS

INTEGRATES

PROVIDES

ACID TX

Server

RUNS_AS

Neo4j

LICENSED_LIKE

Ruby

JS

Clojure

.net

RUNS_AS

SCALES_TO

RUNS_ON

MySQL

Mongo

embedded

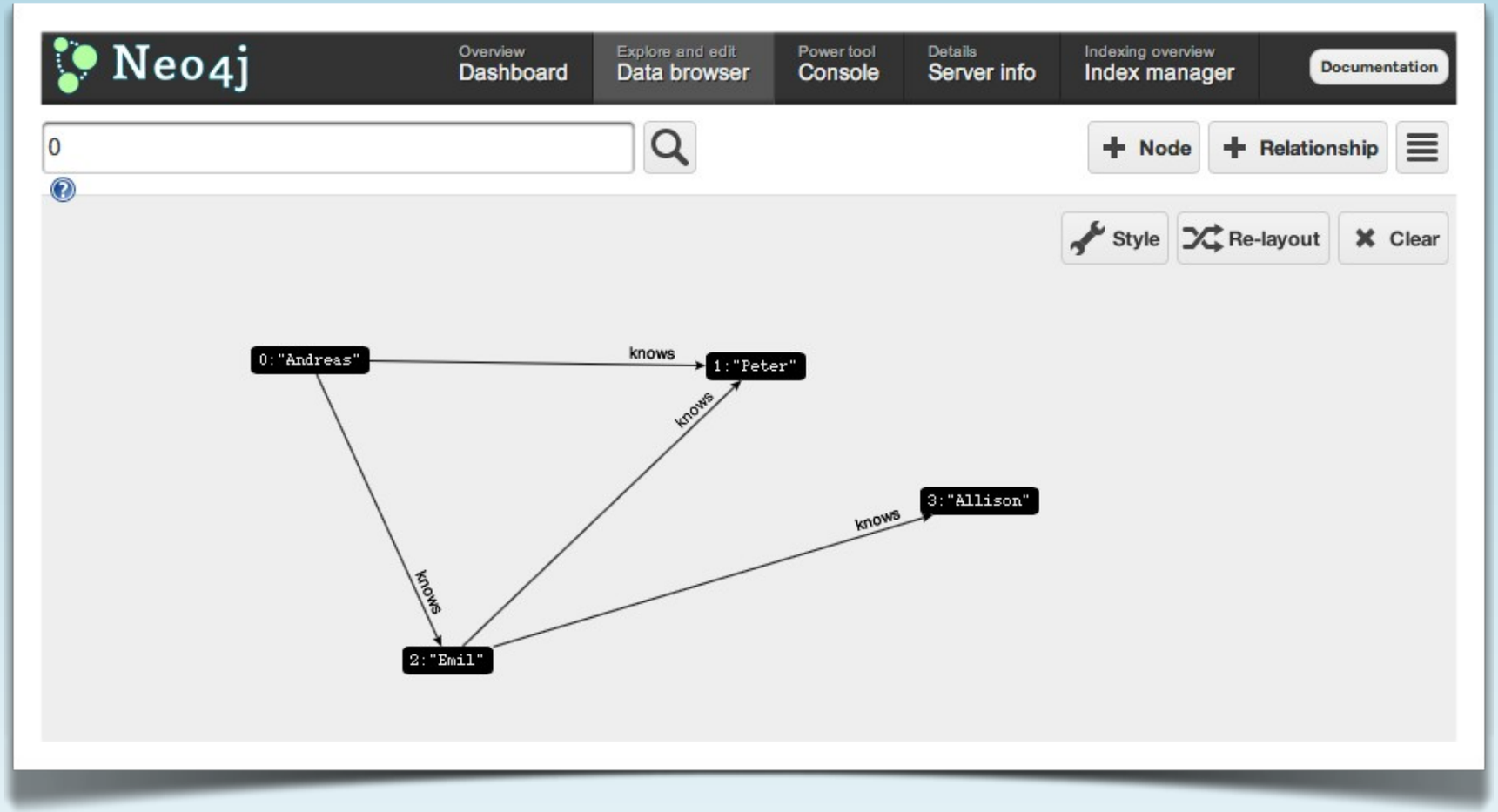
34bn Nodes

Heroku

Neo4j is a Graph Database

- *A **Graph** Database:*
  - *a schema-free Property Graph*
  - *perfect for complex, highly connected data*
- *A Graph **Database**:*
  - *reliable with real ACID Transactions*
  - *scalable: 32 Billion Nodes, 32 Billion Relationships, 64 Billion Properties*
  - *fast with more than 1M traversals / second*
  - *Server with REST API, or Embeddable on the JVM*
  - *higher-performance with High-Availability (read scaling)*

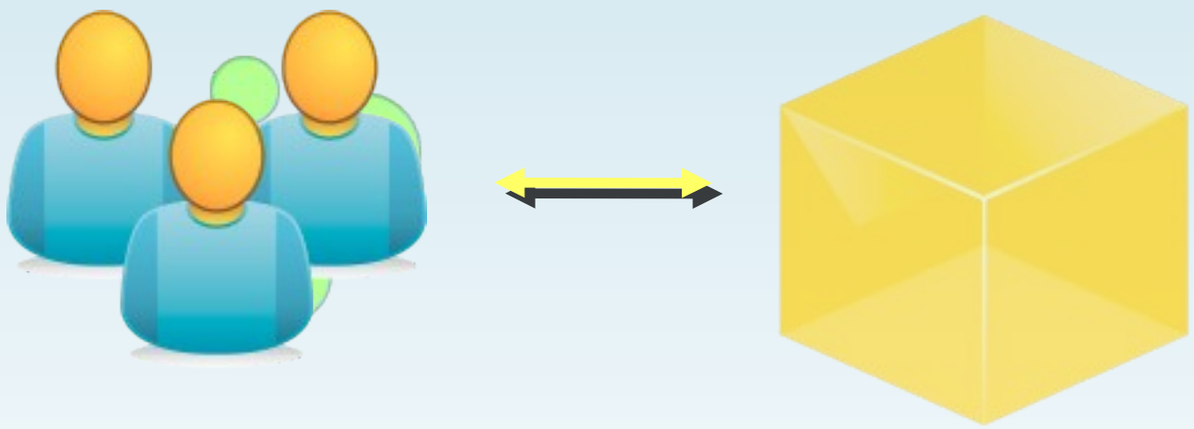
Whiteboard --> Data




```
// Cypher query - friend of a friend
start n=node(0)
match (n)--()--(foaf)
return foaf
```

Two Ways to Work with Neo4j

- *1. Embeddable on JVM*
  - *Java, JRuby, Scala...*
  - *Tomcat, Rails, Akka, etc.*
  - *great for testing*



Show me some code, please

```
GraphDatabaseService graphDb =
    new EmbeddedGraphDatabase(“var/neo4j”);
Transaction tx = graphDb.beginTx();
try {
  Node steve = graphDb.createNode();
  Node michael = graphDb.createNode();

  steve.setProperty(“name”, “Steve Vinoski”);
  michael.setProperty(“name”, “Michael Hunger”);

  Relationship presentedWith = steve.createRelationshipTo(
  michael, PresentationTypes.PRESENTED_WITH);
  presentedWith.setProperty(“date”, today);
  tx.success();
} finally {
  tx.finish();
}
```

Spring Data Neo4j

```
@NodeEntity
public class Movie {
  @Indexed private String title;
  @RelatedToVia(type = “ACTS_IN”, direction=INCOMING)
  private Set<Role> cast;
  private Director director;
}

@NodeEntity
public class Actor {
  @RelatedTo(type = “ACTS_IN”)
  private Set<Movies> movies;
}

@RelationshipEntity
public class Role {
  @StartNode private Actor actor;
  @EndNode private Movie movie;
  private String roleName;
}
```

Cypher Query Language

- *Declarative query language*
  - *Describe what you want, not how*
  - *Based on pattern matching*
- *Examples:*

```
START david=node:people(name="David")   # index lookup
MATCH david-[:knows]-friends-[:knows]-new_friends
WHERE new_friends.age > 18
RETURN new_friends

START user=node(5, 15, 26, 28)   # node IDs
MATCH user--friend
RETURN user, COUNT(friend), SUM(friend.money)
```

Create Graph with Cypher

```
CREATE
  (steve {name: “Steve Vinoski”})
    -[:PRESENTED_WITH {date:{day}}]->
  (michael {name: “Michael Hunger”})
```

Two Ways to Work with Neo4j

◎ *2. Server with REST API*

- *every language on the planet*
- *flexible deployment scenarios*
- *DIY server, or cloud managed*

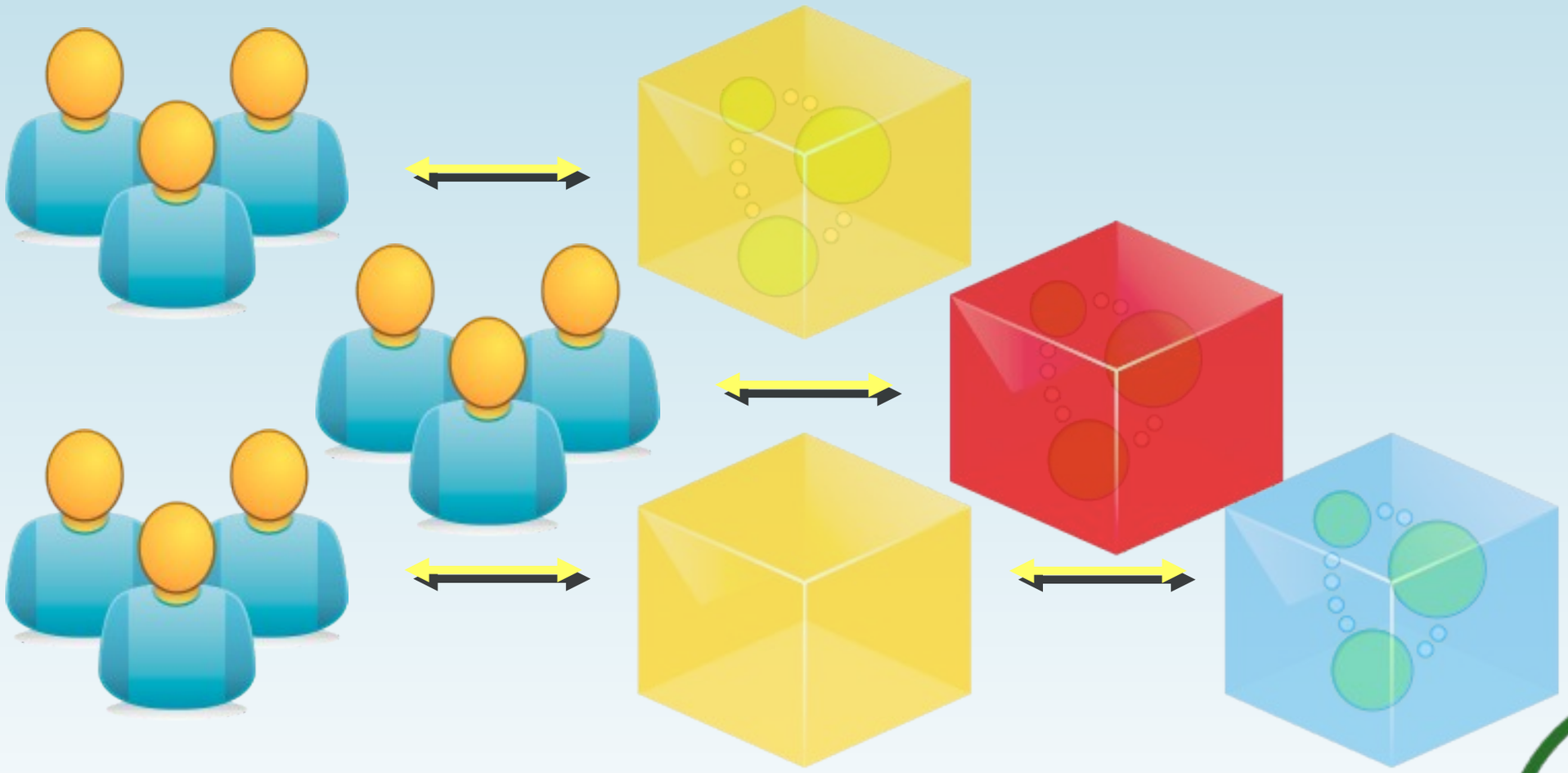
Bindings

REST://

Two Ways to Work with Neo4j

◎ *Server capability == Embedded capability*

- *same scalability, transactionality, and availability*



How to get started?

- Documentation
  - docs.neo4j.org – tutorials+reference
  - http://www.graphdatabases.com
  - Neo4j in Action
  - Good Relationships
- Get Neo4j
  - http://neo4j.org/download
  - http://addons.heroku.com/neo4j/
- Participate
  - http://groups.google.com/group/neo4j
  - http://neo4j.meetup.com
  - a session like this one ;)

Thank you!

Cypher

a

pattern-matching query language for graphs

Cypher - overview

- a pattern-matching query language
- declarative grammar with clauses (like SQL)
- aggregation, ordering, limits
- create, update, delete

Cypher: START + RETURN

- START <lookup> RETURN <expressions>
- START binds terms using simple look-up
  - directly using known ids
  - or based on indexed Property
- RETURN expressions specify result set

```
// lookup node id 0, return that node
start n=node(0) return n
// lookup node in Index, return that node
start n=node:Person(name="Andreas") return n
// lookup all nodes, return all name properties
start n=node(*) return n.name
```

Cypher: MATCH

- START <lookup> MATCH <pattern> RETURN <expr>
- MATCH describes a pattern of nodes+relationships
  - node terms in optional parenthesis
  - lines with arrows for relationships

```
// lookup 'n', traverse any relationship to some 'm'
start n=node(0) match (n)--(m) return n,m
// any outgoing relationship from 'n' to 'm'
start n=node(0) match n-->m return n,m
// only 'KNOWS' relationships from 'n' to 'm'
start n=node(0) match n-[:KNOWS]->m return n,m
// from 'n' to 'm' and capture the relationship as 'r'
start n=node(0) match n-[r]->m return n,r,m
// from 'n' outgoing to 'm', then incoming from 'o'
start n=node(0) match n-->m<--o return n,m,o
```

Cypher: WHERE

- START <lookup> [MATCH <pattern>]
- WHERE <condition> RETURN <expr>
- WHERE filters nodes or relationships
  - uses expressions to constrain elements

```
// lookup all nodes as 'n', constrained to name 'Andreas'
start n=node(*) where n.name='Andreas' return n
// filter nodes where age is less than 30
start n=node(*) where n.age<30 return n
// filter using a regular expression
start n=node(*) where n.name =~ /Tob.*/ return n
// filter for a property exists
start n=node(*) where has(n.name) return n
```

Cypher: CREATE

- CREATE <node>[,node or relationship] RETURN <expr>
  - create nodes with optional properties
  - create relationship (must have a type)

```
// create an anonymous node
create n
// create node with a property, returning it
create n={name:'Andreas'} return n
// lookup 2 nodes, then create a relationship and return it
start n=node(0),m=node(1) create n-[r:KNOWS]-m return r
// lookup nodes, then create a relationship with properties
start n=node(1),m=node(2) create n-[r:KNOWS {since:2008}]->m
```

Cypher: SET

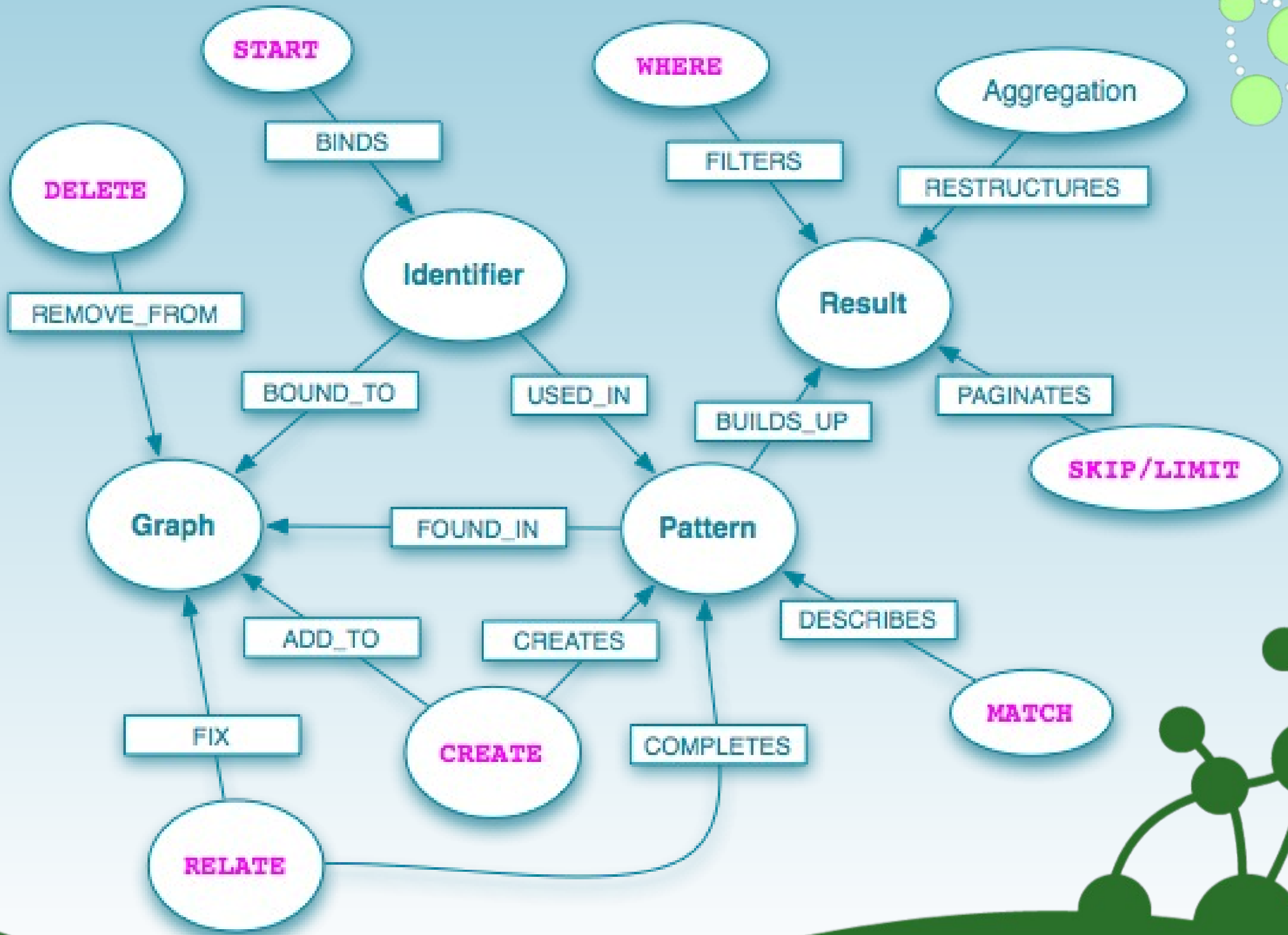
- SET [<node property>] [<relationship property>]
  - update a property on a node or relationship
  - must follow a START

```
// update the name property
start n=node(0) set n.name='Peter'
// update many nodes, using a calculation
start n=node(*) set n.size=n.size+1
// match & capture a relationship, update a property
start n=node(1) match n-[r]-m set r.times=10
```

Cypher: DELETE

- DELETE [<node>|<relationship>|<property>]
  - delete a node, relationship or property
  - must follow a START
  - to delete a node, all relationships must be deleted first

```
// delete a node
start n=node(5) delete n
// remove a node and all relationships
start n=node(3) match n-[r]-() delete n, r
// remove a property
start n=node(3) delete n.age
```


START
WHERE
Aggregation
BINDS
FILTERS
RESTRUCTURES
DELETE
Identifier
Result
REMOVE_FROM
BOUND_TO
USED_IN
BUILDS_UP
PAGINATES
SKIP/LIMIT
Graph
FOUND_IN
Pattern
ADD_TO
CREATES
DESCRIBES
MATCH
FIX
CREATE
COMPLETES
RELATE

The Rabbithole

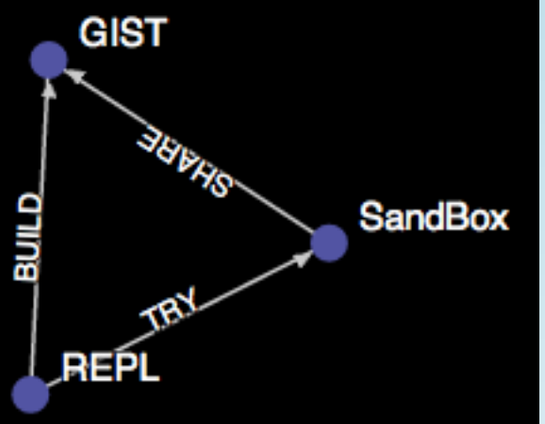
You can modify and query this graph by entering statements in the input field at the bottom.
For some syntax help hit the button. If you want to share your graph, just do it with

```
start n=node(*)
match n-[r]-m
return n,r
```

n	r
Node[0]{name->"REPL"}	:TRY[0] {}
Node[0]{name->"REPL"}	:BUILD[2] {}
Node[1]{name->"GIST"}	:SHARE[1] {}
Node[1]{name->"GIST"}	:BUILD[2] {}
Node[2]{name->"SandBox"}	:TRY[0] {}
Node[2]{name->"SandBox"}	:SHARE[1] {}

6 rows
0 ms

```
start n=node(*) match n-[r]-m return n,r
```



http://console.neo4j.org

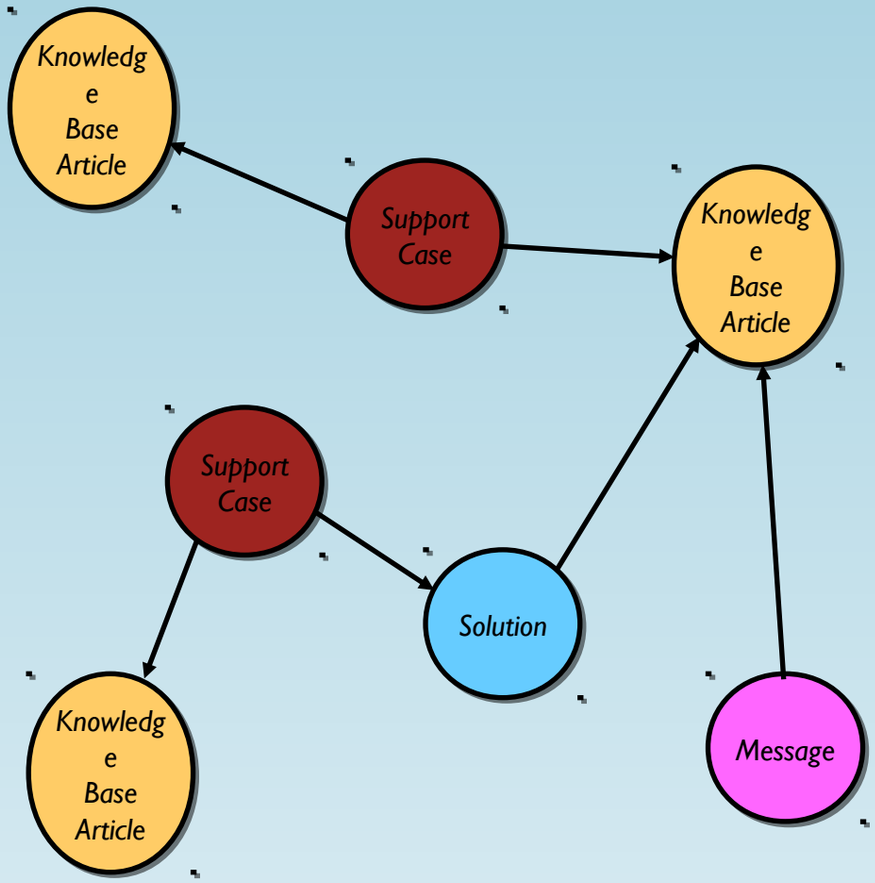
This Graph: http://tinyurl.com/7cnvmlq

the Real World

Industry: Communications *Use case: Recommendations*

San Jose, CA

Cisco.com



- *Cisco.com serves customer and business customers with Support Services*
- *Needed real-time recommendations, to encourage use of online knowledge base*
- *Cisco had been successfully using Neo4j for its internal master data management solution.*
- *Identified a strong fit for online recommendations*

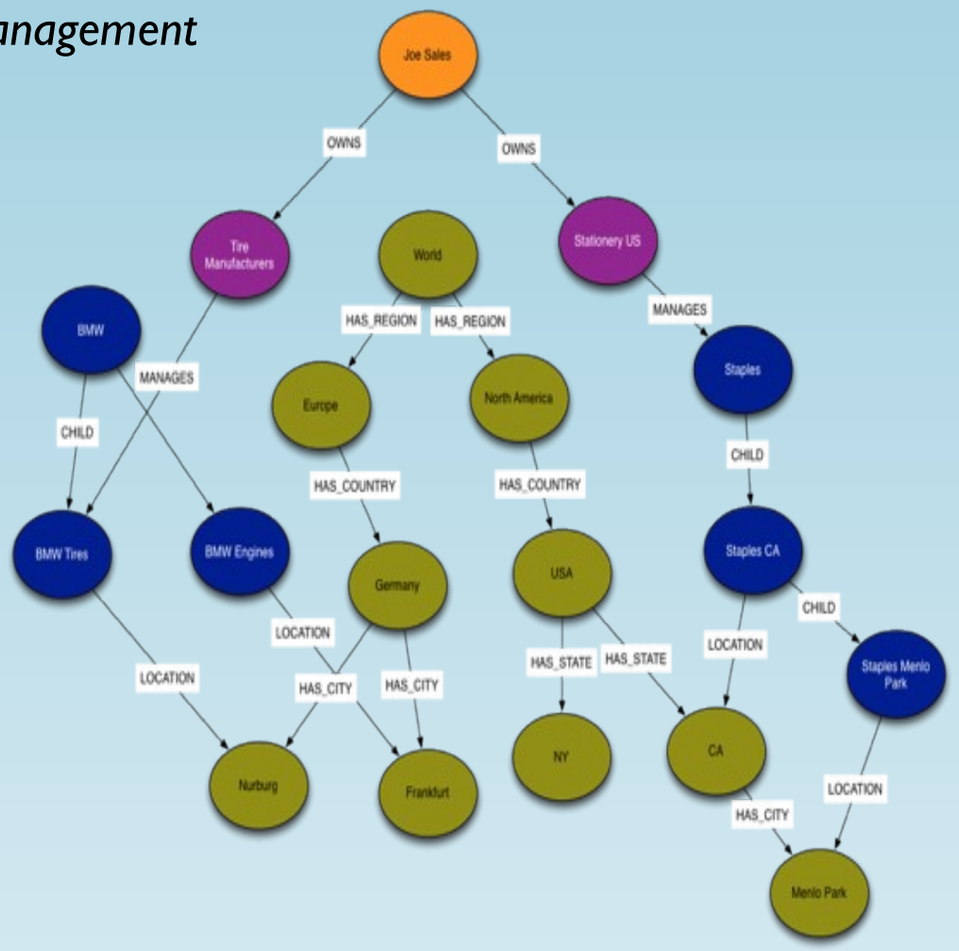
- *Call center volumes needed to be lowered by improving the efficacy of online self service*
- *Leverage large amounts of knowledge stored in service cases, solutions, articles, forums, etc.*
- *Problem resolution times, as well as support costs, needed to be lowered*

- *Cases, solutions, articles, etc. continuously scraped for cross-reference links, and represented in Neo4j*
- *Real-time reading recommendations via Neo4j*
- *Neo4j Enterprise with HA cluster*
- *The result: customers obtain help faster, with decreased reliance on customer support*

Industry: Communications *Use case: Master Data Management*

San Jose, CA

Cisco HMP



- *One of the world's largest communications equipment manufacturers#91 Global 2000. $44B in annual sales.*
- *Needed a system that could accommodate its master data hierarchies in a performant way*
- *HMP is a Master Data Management system at whose heart is Neo4j. Data access services available 24x7 to applications companywide*

- *Sales compensation system had become unable to meet Cisco's needs*
- *Existing Oracle RAC system had reached its limits:*
  - *Insufficient flexibility for handling complex organizational hierarchies and mappings*
  - *"Real-time" queries were taking > 1 minute!*
- *Business-critical "P1" system needs to be continually available, with zero downtime*

- *Cisco created a new system: the Hierarchy Management Platform (HMP)*
- *Allows Cisco to manage master data centrally, and centralize data access and business rules*
- *Neo4j provided "Minutes to Milliseconds" performance over Oracle RAC, serving master data in real time*
- *The graph database model provided exactly the flexibility needed to support Cisco's business rules*
- *HMP so successful that it has expanded to include product hierarchy*

Industry: Logistics *Use case: Parcel Routing*

accenture

- *One of the world's largest logistics carriers*
- *Projected to outgrow capacity of old system*
- *New parcel routing system*
- *Single source of truth for entire network*
- *B2C & B2B parcel tracking*
- *Real-time routing: up to 5M parcels per day*

- *24x7 availability, year round*
- *Peak loads of 2500+ parcels per second*
- *Complex and diverse software stack*
- *Need predictable performance & linear scalability*
- *Daily changes to logistics network: route from any point, to any point*

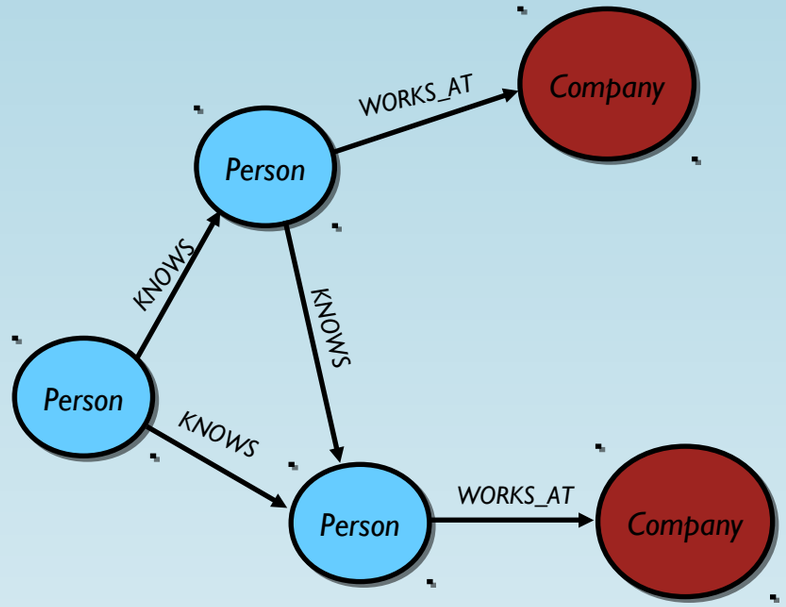
- *Neo4j provides the ideal domain fit:*
- *a logistics network is a graph*
- *Extreme availability & performance with Neo4j clustering*
- *Hugely simplified queries, vs. relational for complex routing*
- *Flexible data model can reflect real-world data variance much better than relational*
- *"Whiteboard friendly" model easy to understand*

Industry: Online Job Search *Use case: Social / Recommendations*

Sausalito, CA

GlassDoor

- *Online jobs and career community, providing anonymized inside information to job seekers*



- *Wanted to leverage known fact that most jobs are found through personal & professional connections*
- *Needed to rely on an existing source of social network data. Facebook was the ideal choice.*
- *End users needed to get instant gratification*
- *Aiming to have the best job search service, in a very competitive market*

- *First-to-market with a product that let users find jobs through their network of Facebook friends*
- *Job recommendations served real-time from Neo4j*
- *Individual Facebook graphs imported real-time into Neo4j*
- *Glassdoor now stores > 50% of the entire Facebook social graph*
- *Neo4j cluster has grown seamlessly, with new instances being brought online as graph size and load have increased*

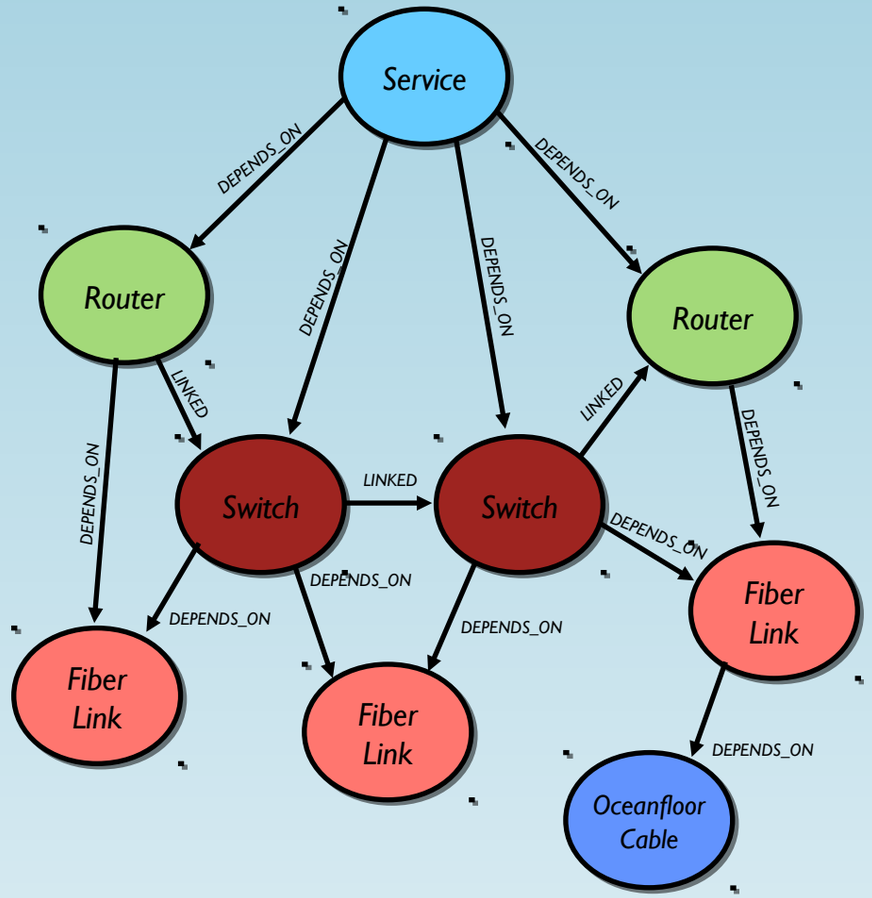
Industry: Communications

Use case: Network Management

Paris, France

SFR

- *Second largest communications company in France*
- *Part of Vivendi Group, partnering with Vodafone*



- *Infrastructure maintenance took one full week to plan, because of the need to model network impacts*
- *Needed rapid, automated "what if" analysis to ensure resilience during unplanned network outagesIdentify weaknesses in the network to uncover the need for additional redundancy*
- *Network information spread across > 30 systems, with daily changes to network infrastructureBusiness needs sometimes changed very rapidly*

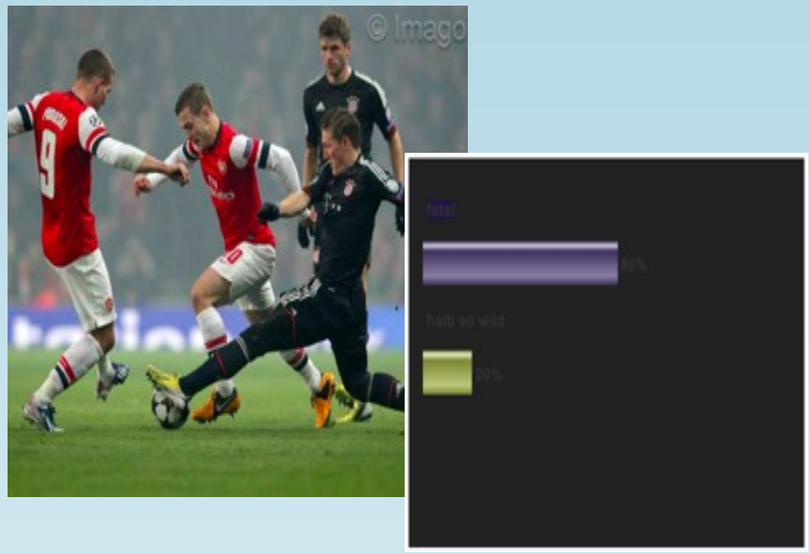
- *Flexible network inventory management system, to support modeling, aggregation & troubleshooting*
- *Single source of truth (Neo4j) representing the entire network*
- *Dynamic system loads data from 30+ systems, and allows new applications to access network data*
- *Modeling efforts greatly reduced because of the near 1:1 mapping between the real world and the graph*
- *Flexible schema highly adaptable to changing business requirements*

Industry: *Communications* *Use case:* *Social gaming*

Frankfurt, Germany

Deutsche Telecom

Interactive Television Programming



- *Europe's largest communications company*
- *Provider of mobile & land telephone lines to consumers and businesses, as well as internet services, television, and other services*

> 236,000	50	> 58 bn. €
Employees worldwide in 2011	Countries	Revenue in 2011

- *The Fanorakel application allows fans to have an interactive experience while watching sports*
- *Fans can vote for referee decisions and interact with other fans watching the game*
- *Highly connected dataset with real-time updates*
- *Queries need to be served real-time on rapidly changing data*
- *One technical challenge is to handle the very high spikes of activity during popular games*

- *Interactive, social offering gives fans a way to experience the game more closely*
- *Increased customer stickiness for Deutsche Telekom*
- *A completely new channel for reaching customers with information, promotions, and ads*
- *Clear competitive advantage*

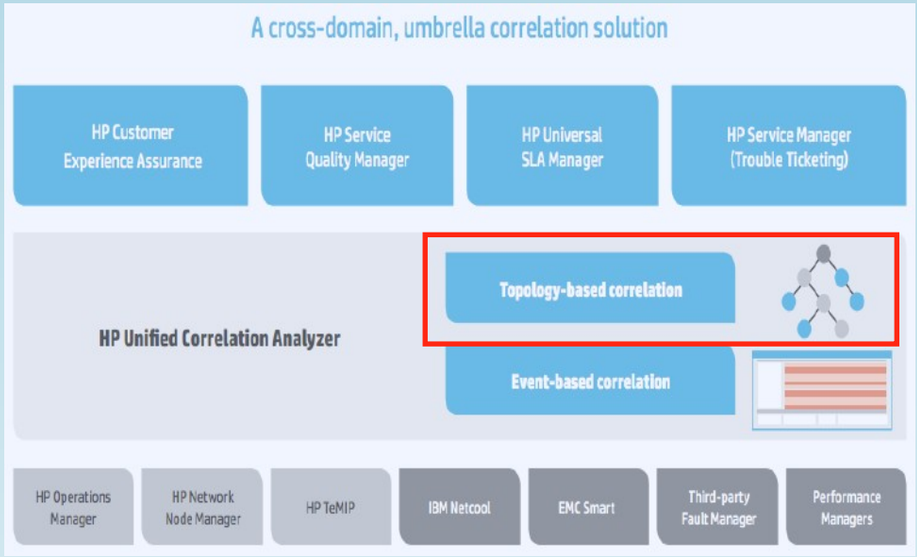
Industry: Web/ISV, Communications

Global (U.S., France)

Hewlett Packard

Use case: Network Management

- *World's largest provider of IT infrastructure, software & services*
- *HP's Unified Correlation Analyzer (UCA) application is a key application inside HP's OSS Assurance portfolio*
- *Carrier-class resource & service management, problem determination, root cause & service impact analysis*
- *Helps communications operators manage large, complex and fast changing networks*



- *Use network topology information to identify root problems causes on the network*
- *Simplify alarm handling by human operators*
- *Automate handling of certain types of alarms*
- *Help operators respond rapidly to network issues*
- *Filter/group/eliminate redundant Network Management System alarms by event correlation*

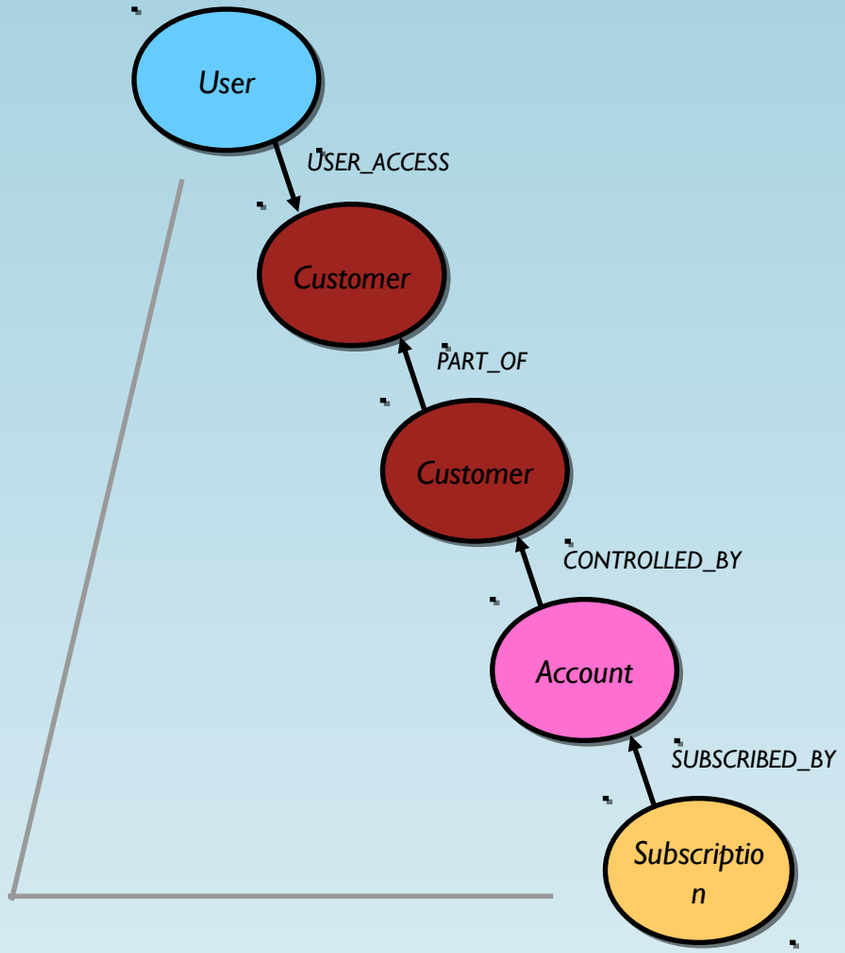
- *Accelerated product development time*
- *Extremely fast querying of network topology*
- *Graph representation a perfect domain fit*
- *24x7 carrier-grade reliability with Neo4j HA clustering*
- *Met objective in under 6 months*

Industry: Communications

Use case: Resource Authorization & Access Control

Oslo, Norway

Telenor



- *10th largest Telco provider in the world, leading in the Nordics*
- *Online self-serve system where large business admins manage employee subscriptions and plans*
- *Mission-critical system whose availability and responsiveness is critical to customer satisfaction*

- *Degrading relational performance. User login taking minutes while system retrieved access rights*
- *Millions of plans, customers, admins, groups. Highly interconnected data set w/massive joins*
- *Nightly batch workaround solved the performance problem, but meant data was no longer current*
- *Primary system was Sybase. Batch pre-compute workaround projected to reach 9 hours by 2014: longer than the nightly batch window*

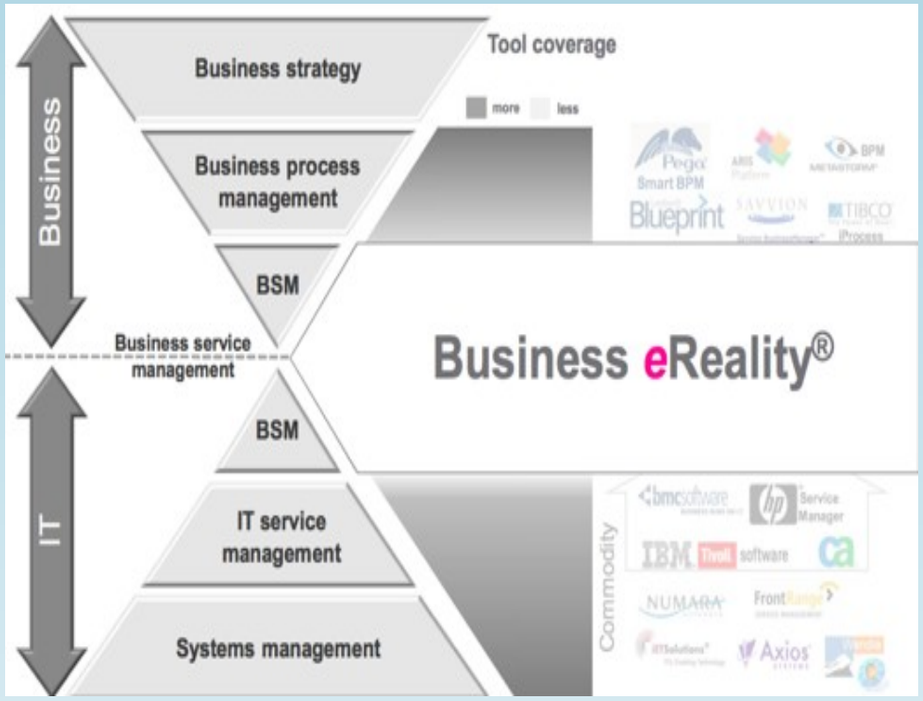
- *Moved authorization functionality from Sybase to Neo4j*
- *Modeling the resource graph in Neo4j was straightforward, as the domain is inherently a graph*
- *Able to retire the batch process, and move to real-time responses: measured in milliseconds*
- *Users able to see fresh data, not yesterday's snapshotCustomer retention risks fully mitigated*

Industry: Web/ISV, Communications *Use case: Data Center Management*

Zürich, Switzerland

Junisphere

- *Junisphere AG is a Zurich-based IT solutions provider*
- *Founded in 2001.*
- *Profitable.*
- *Self funded.*
- *Software & services.*
- *Novel approach to infrastructure monitoring:*
  Starts with the end user, mapped to business processes and services, and dependent infrastructure



- *"Business Service Management" requires mapping of complex graph, covering: business processes--> business services--> IT infrastructure*
- *Embed capability of storing and retrieving this information into OEM application*
- *Re-architecting outdated C++ application based on relational database, with Java*

- *Actively sought out a Java-based solution that could store data as a graph*
- *Domain model is reflected directly in the database:"No time lost in translation"*
- *"Our business and enterprise consultants now speak the same language, and can model the domain with the database on a 1:1 ratio."*
- *Spring Data Neo4j strong fit for Java architecture*

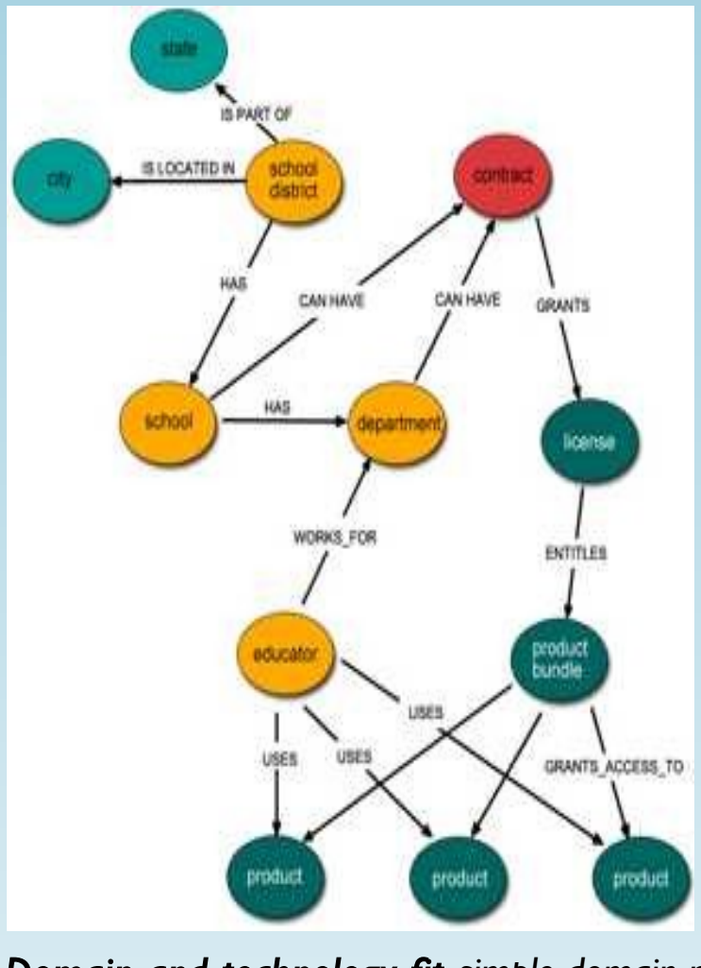
Industry: Education *Use case: Resource Authorization & Access Control*

San Francisco, CA

Teachscape

- *Teachscape, Inc. develops online learning tools for K-12 teachers, school principals, and other instructional leaders.*
- *Teachscape evaluated relational as an option, considering MySQL and Oracle.*
- *Neo4j was selected because the graph data model provides a more natural fit for managing organizational hierarchy and access to assets.*

- *Neo4j was selected to be at the heart of a new architecture. The user management system, centered around Neo4j, will be used to support single sign-on, user management, contract management, and end-user access to their subscription entitlements.*



- ***Domain and technology fit*** *simple domain model where the relationships are relatively complex.*
- *Secondary factors included support for transactions, strong Java support, and well-implemented Lucene indexing integration*
- ***Speed and Flexibility*** *The business depends on being able to do complex walks quickly and efficiently. This was a major factor in the decision to use Neo4j.*
- ***Ease of Use*** *accommodate efficient access for home-grown and commercial off-the-shelf applications, as well as ad-hoc use.*
- *Extreme availability & performance with Neo4j clustering*
- *Hugely simplified queries, vs. relational for complex routing*
- *Flexible data model can reflect real-world data variance much better than relational*
- *"Whiteboard friendly" model easy to understand*

Really, once you start

thinking in graphs

it's hard to stop

What will you build?

Recommendations MDM

Geospatial

Business intelligence

catalogs

Systems

access control

Social computing

Management

your brain

Biotechnology

routing

genealogy

linguistics

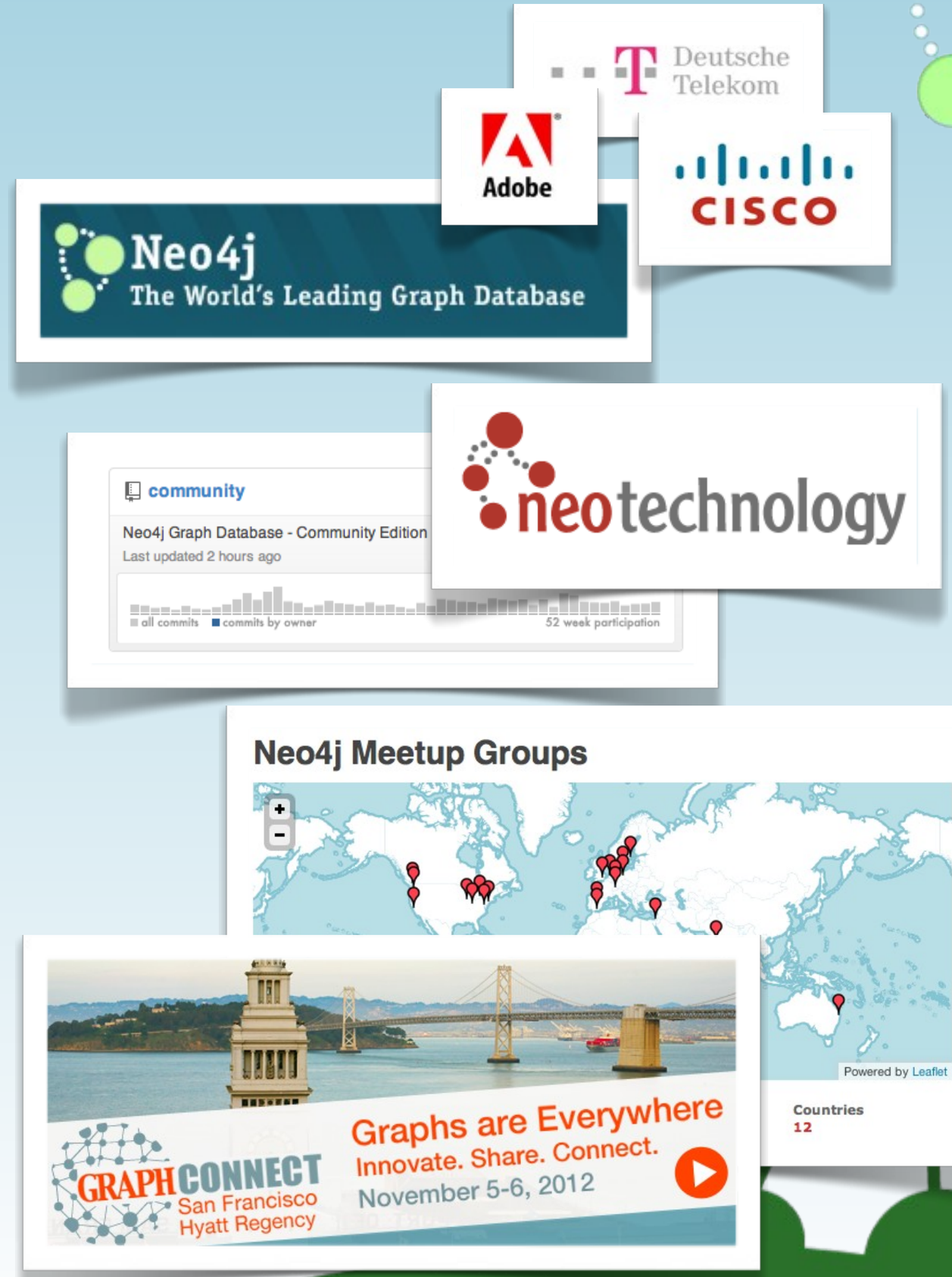
Making Sense of all that

compensation data

market vectors

Google "neo4j"

- [docs.]neo4j.org
- [news.]neotechnology.com
- github.com/neo4j
- neo4j.meetup.com
- graphconnect.com

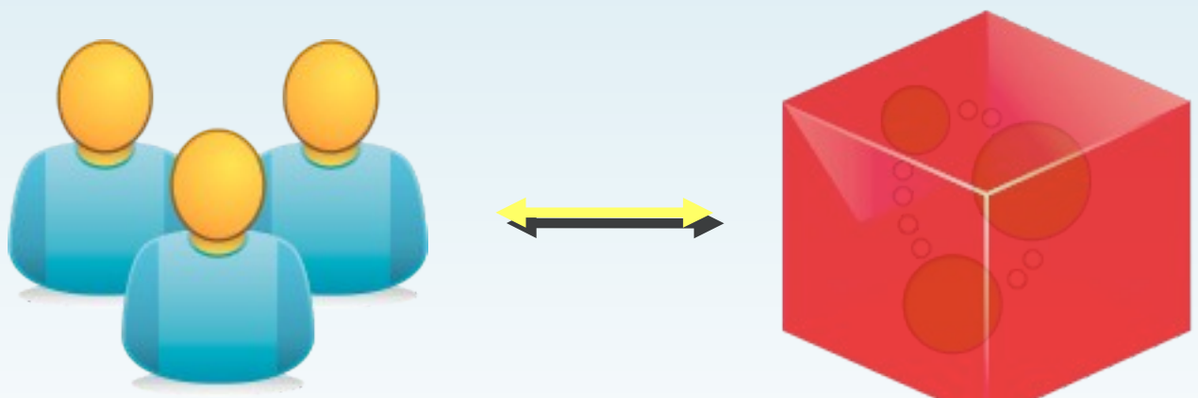


Neo4j

High Availability

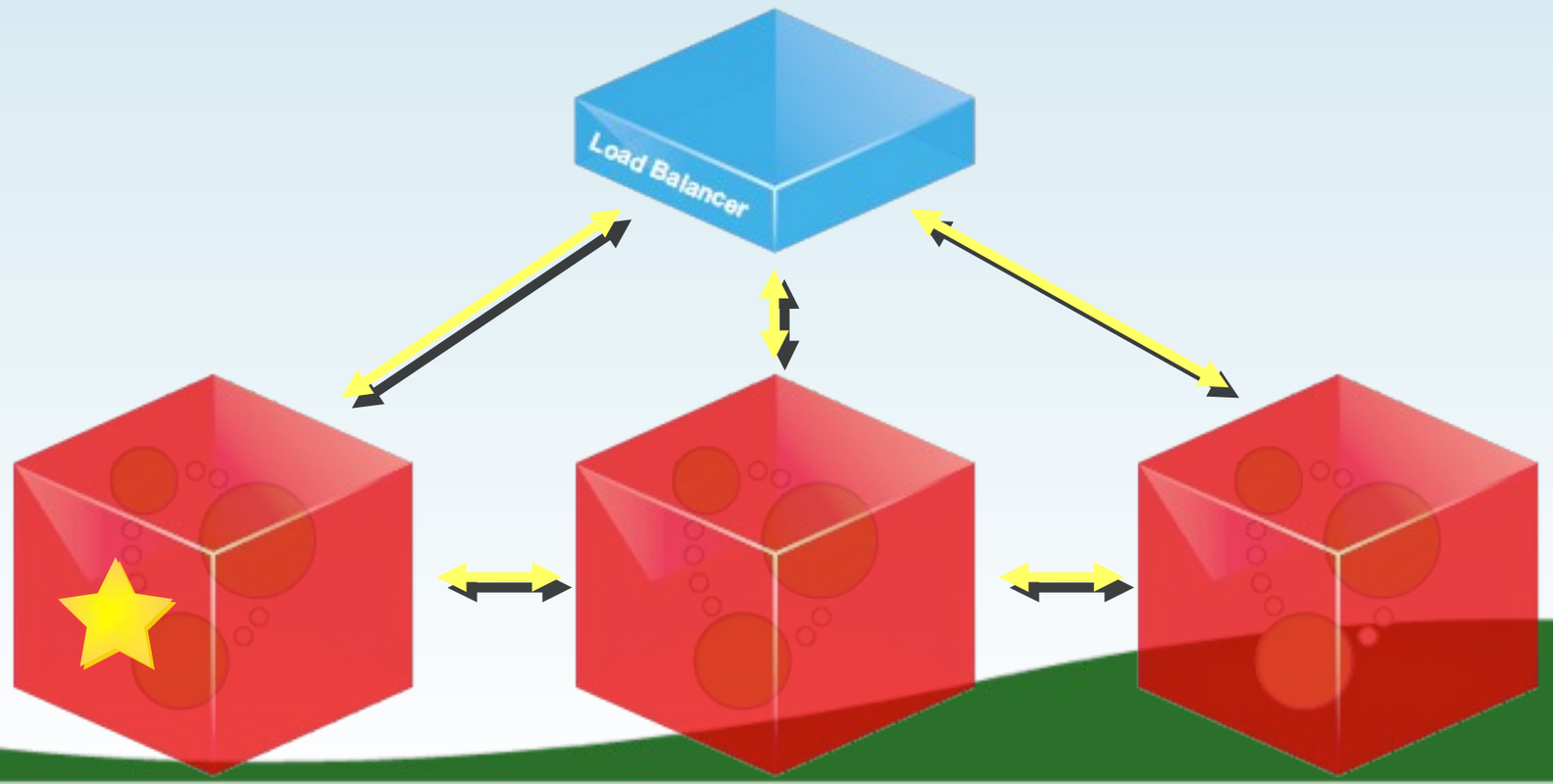
Scaling on a single server

- *data size can increase into the billions*
- *however*
  - *performance relies on memory caches*
  - *server must be taken offline for backups*
  - *single point of failure*
- *For 24x7 production, it's time to introduce HA*



High Availability

- *master-slave replication*
  - *read/write to any cluster member*
  - *slave writes commit to master first (redundancy)*
  - *master writes are faster*
  - *all writes propagate to slaves (polling interval)*

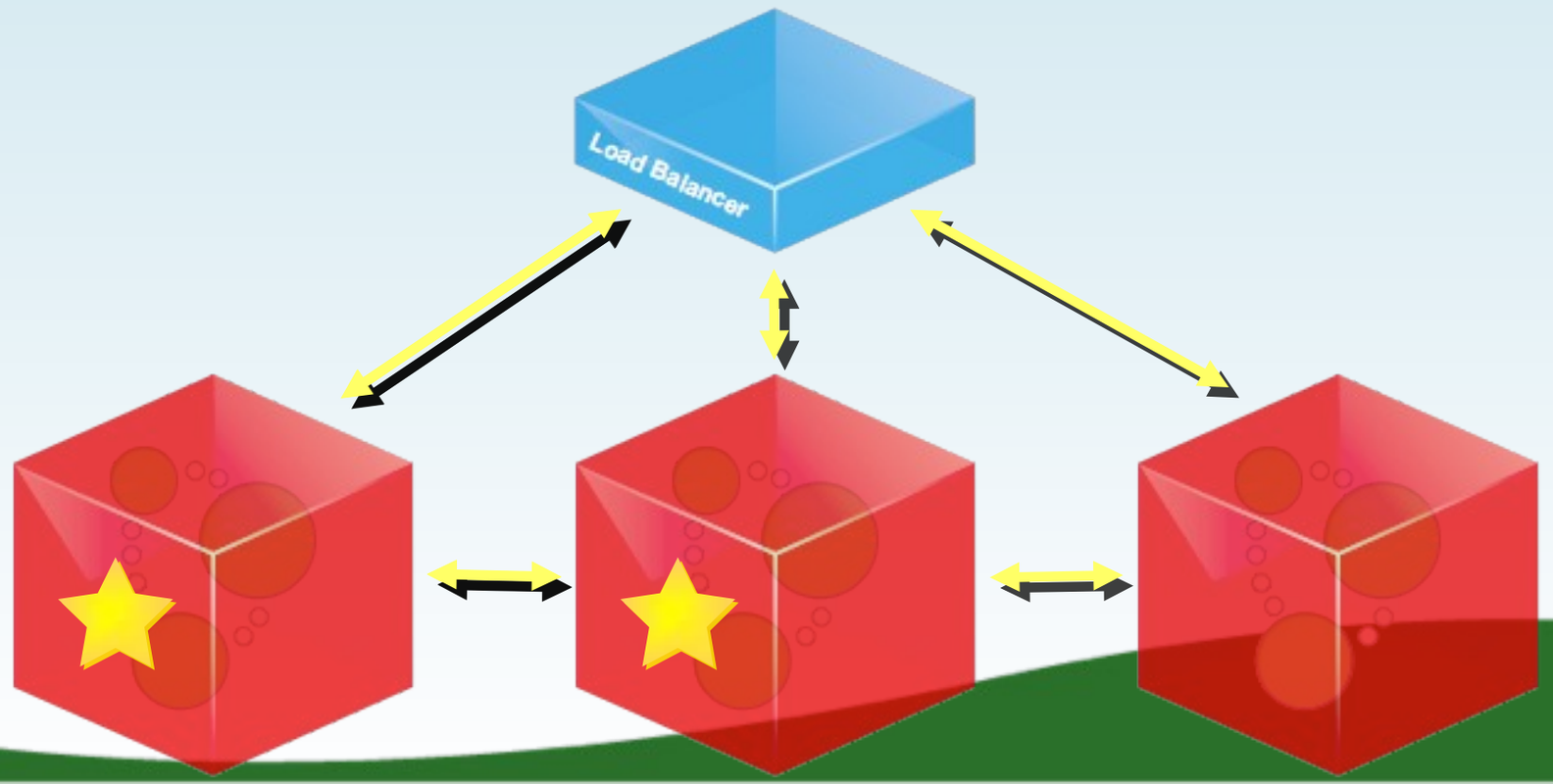


High Availability

◎ *automatic fail-over*

- *any cluster member can be elected master*
- *on failure, a new master will be automatically elected*
- *a failed master can re-join as a slave*

- *automatic branch detection & resolution*

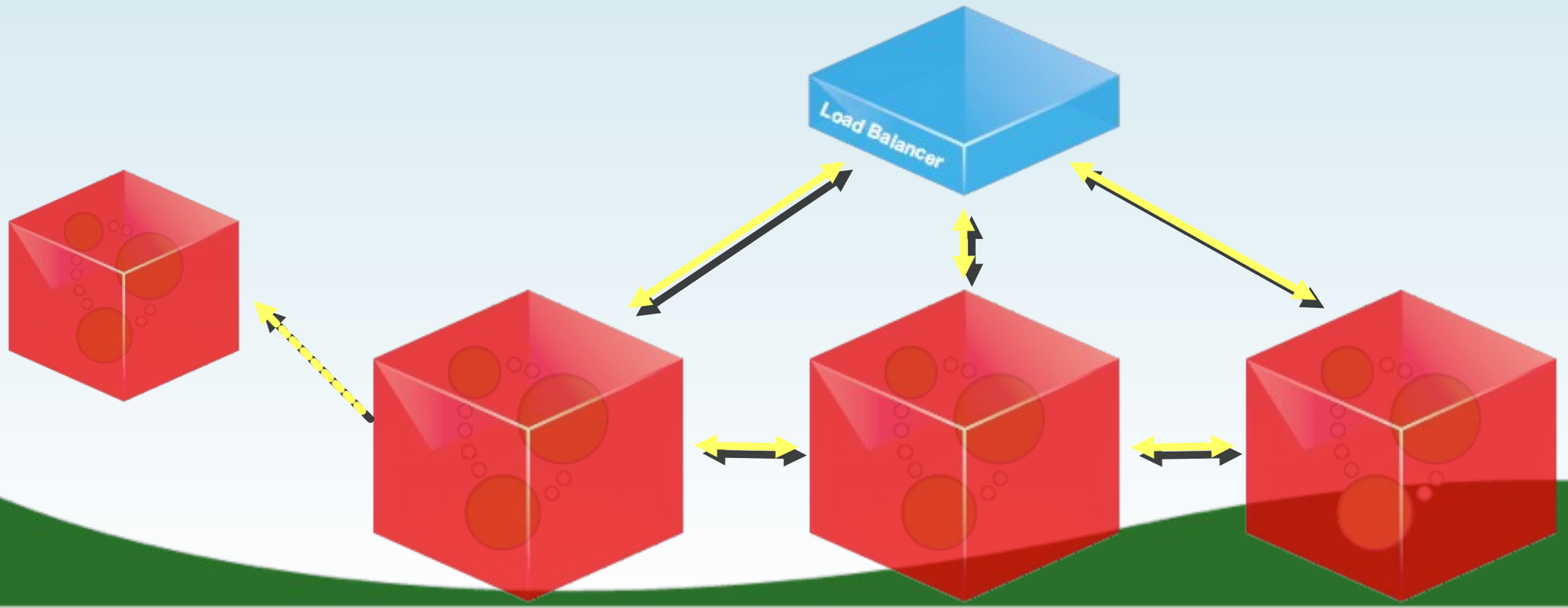


High Availability

◎ *online backups*

- *backup pulls updates directly from live cluster*
- *backup is a full, directly useable Neo4j database*

◎ *to restore: shutdown cluster, distribute backup, restart*

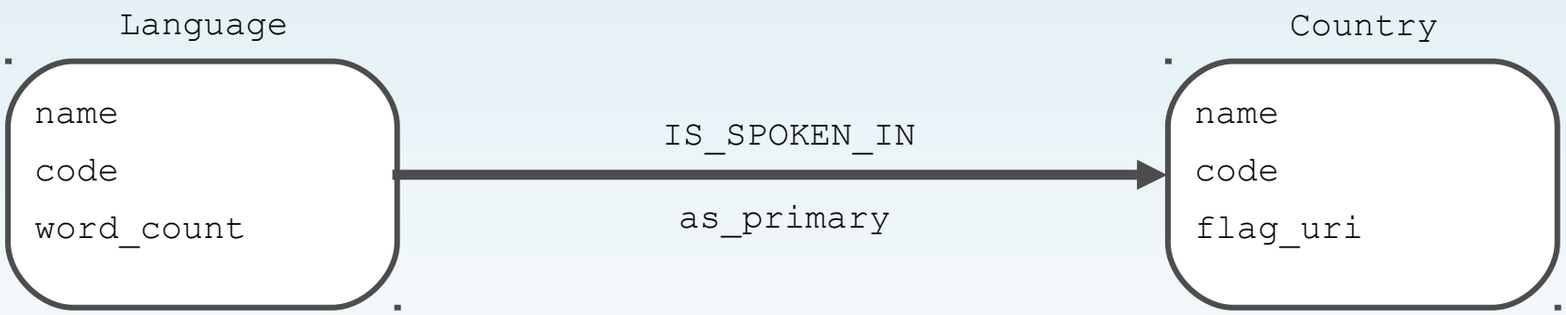


3 Lessons Learned

1. Healthy Relationships

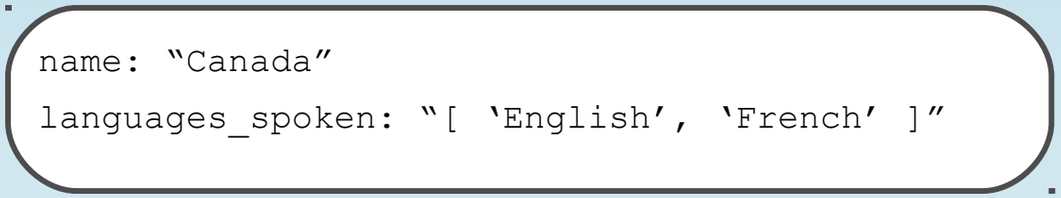
◎ *replace many-to-many join tables...*

◎ *...with a relationship in the graph*

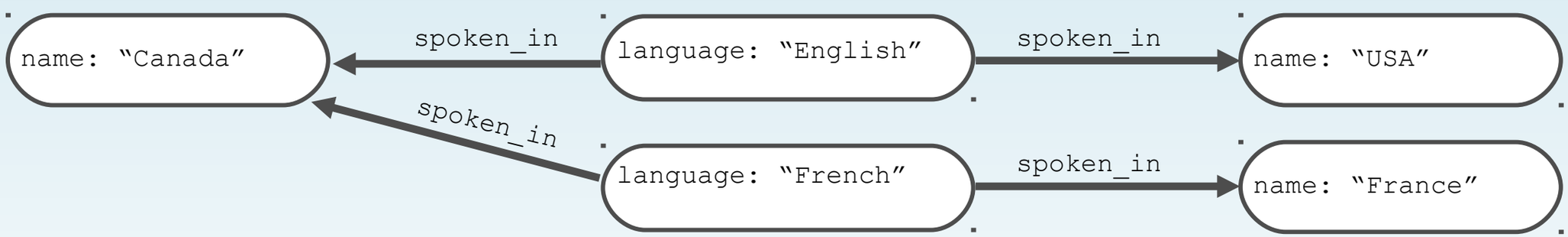


2. Property Lists

- *Don't try to embed multiple values into a single property*
- *That makes it harder to traverse using these values*



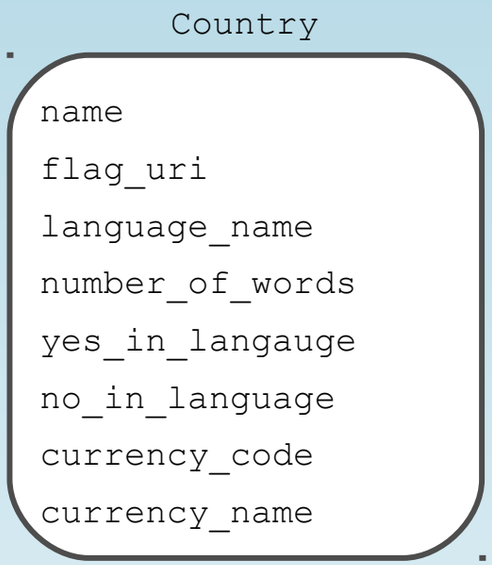
- *Instead, extract "list" values into separate nodes*

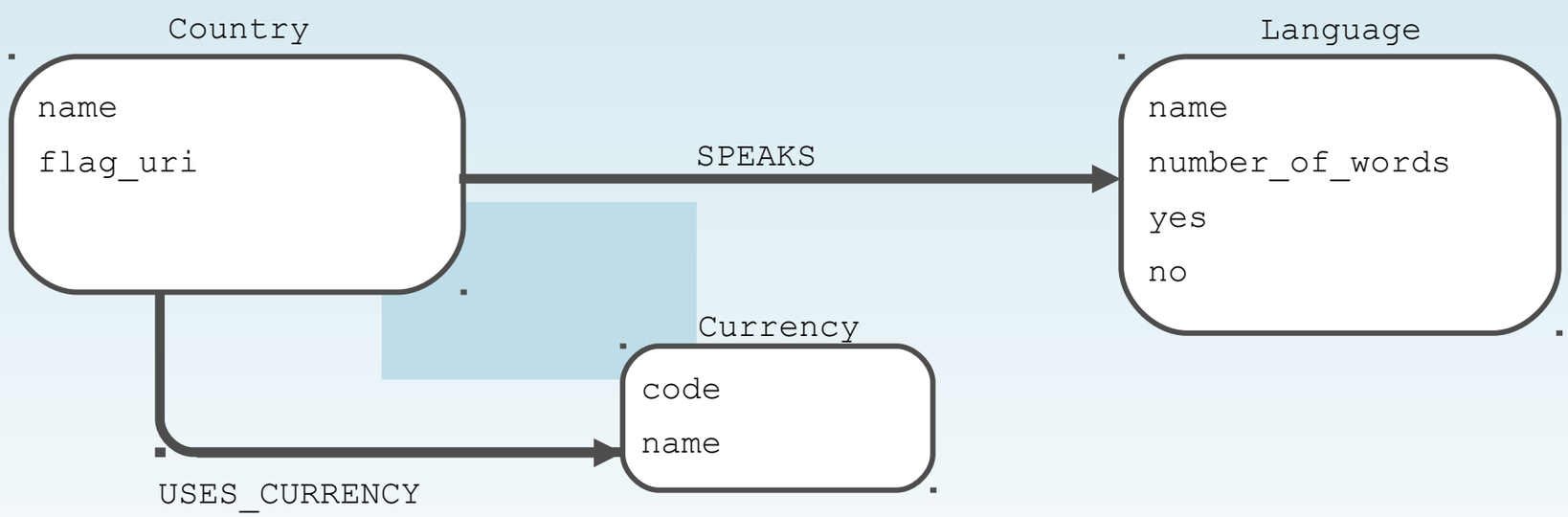


3. One Concept Per Node

◎ *Don't bundle multiple concepts*

stead, break out the separate concepts...





3 Lessons Learned

- *Use Relationships*
- *Use Relationships*
- *Use Relationships*