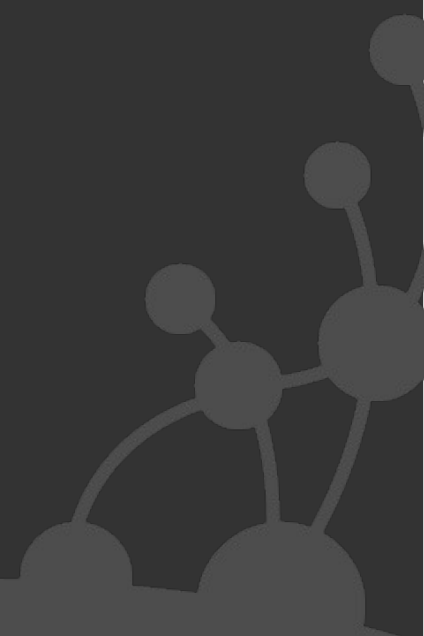


Neo4j

or: why graph dbs
kick ass

Emil Eifrem
CEO, Neo Technology



Death?

Community experimentation:

- CouchDB
- SimpleDB
- Hypertable
- Cassandra
- Scalaris
- ...

Tuesday, February 5, 2008

The Death of the Relational Database

The relational database is becoming increasingly less useful in a web 2.0 world. The reason for this is that, while the relational database model is great for storing information, it is horrible for storing knowledge. By knowledge I mean information that has value beyond the narrow current conception of the given application. I mean information that can have enduring value. In this context, one might say knowledge is information in a disposable form.

The RDBMS is not enough.

Posted by **Sebastien Auvray** on Nov 26, 2007 09:30 AM

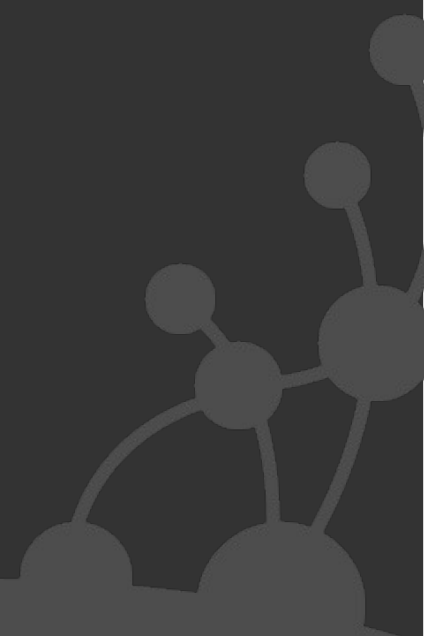
Community **Architecture, Ruby** Topics **Performance & Scalability, Data Access, Database Design**

Tags: **CouchDB, Database, Database Management, Distributed Document Oriented Database, S3, RDBMS, Relational Databases, Scalability**

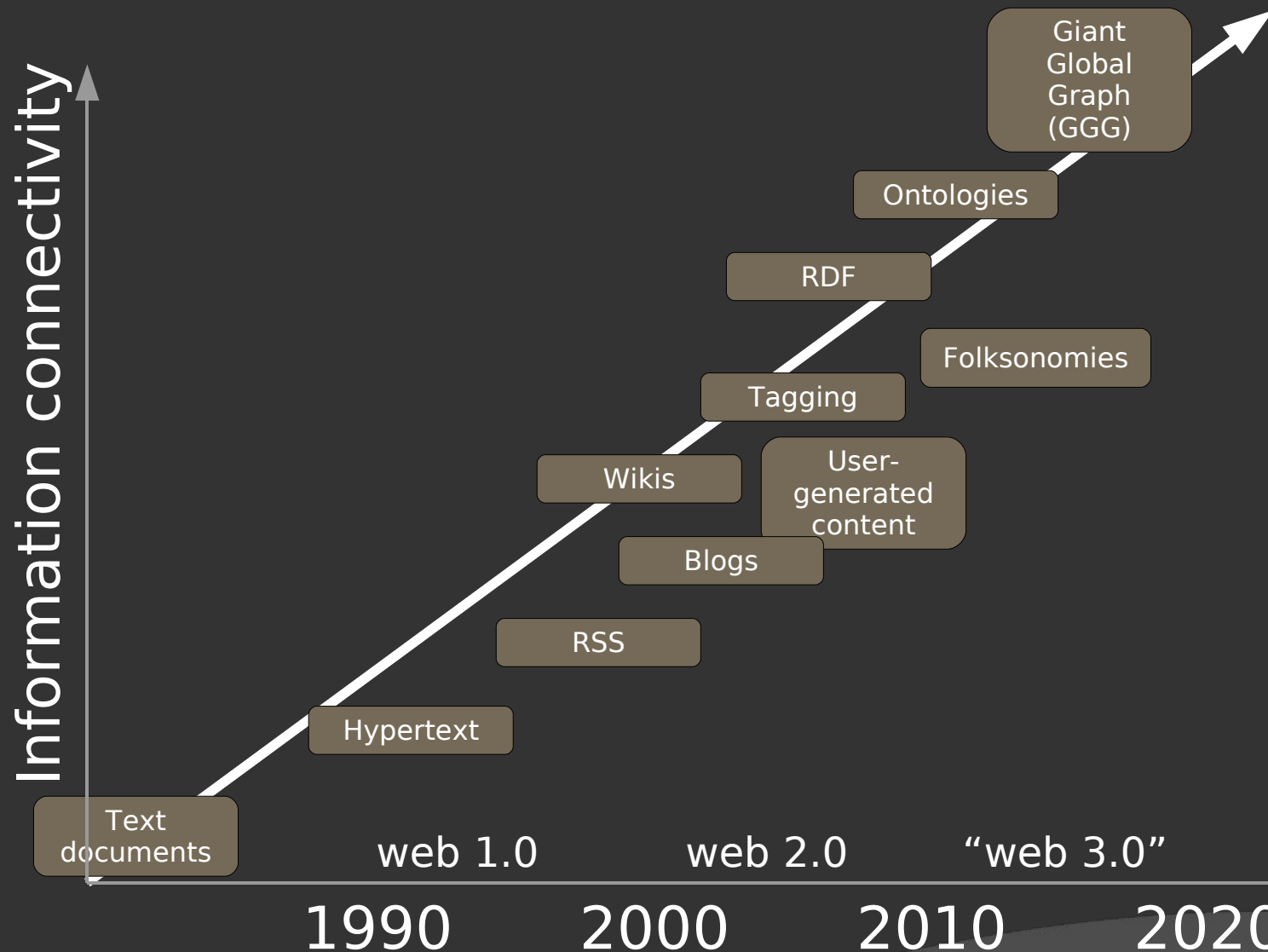
While Relational Databases fit a client-server model, in a [world of services new solutions are needed](#), RDBMS are subject to scalability issues: [How to create redundancy, parallelism ?](#)

CouchDB: Thinking beyond the RDBMS

September 2nd, 2007



Trend 1: data is getting more connected

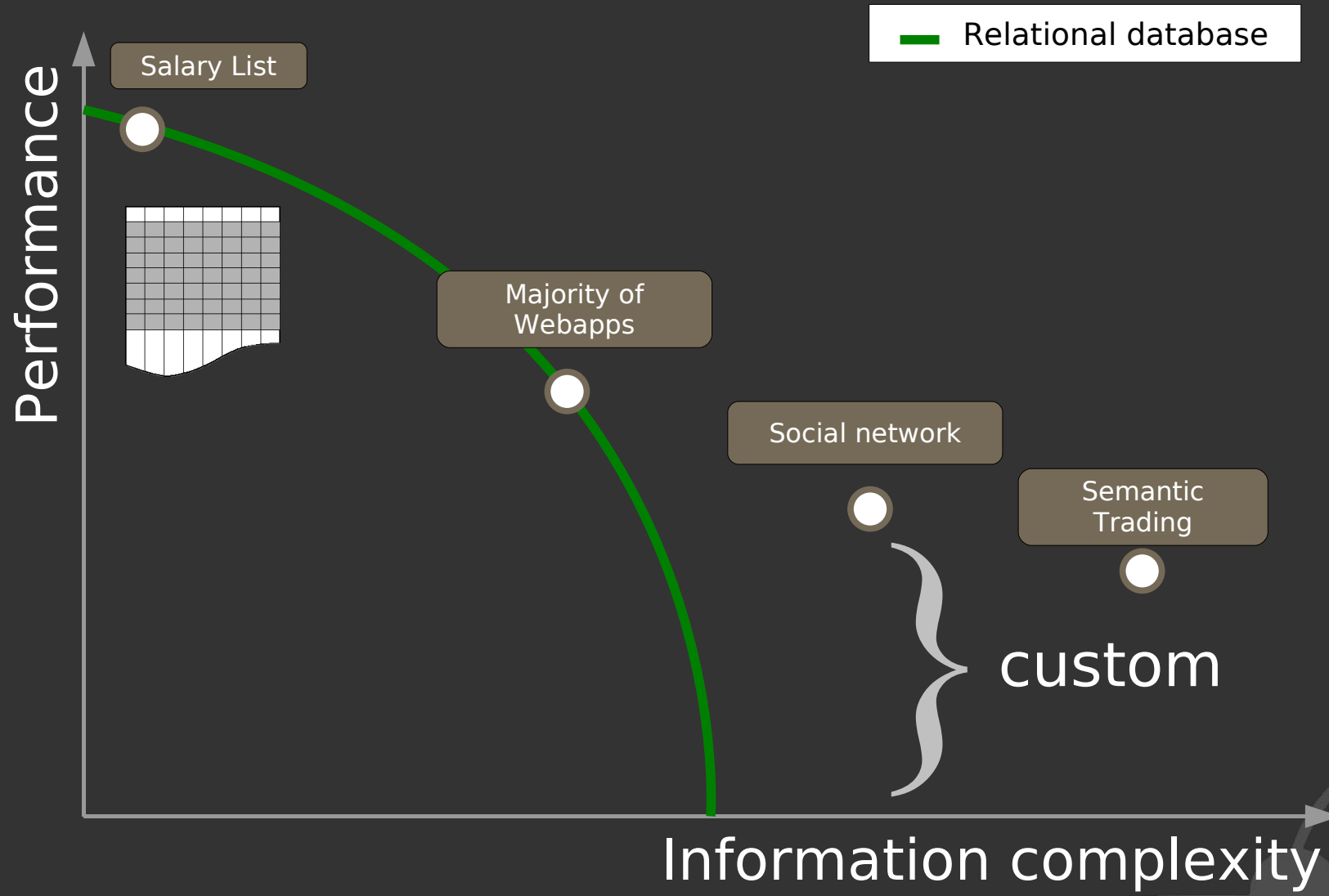


Trend 2: ... and more semi-structured

◎ Individualization of content!

- In the salary lists of the 1970s, all elements had exactly one job
- In the salary lists of the 2000s, we need 5 job columns! Or 8? Or 15?

◎ Trend accelerated by the decentralization of content generation that is the hallmark of the age of participation (“web 2.0”)

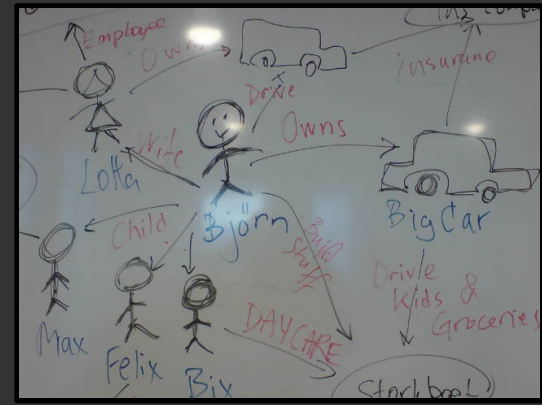


We = hackers!

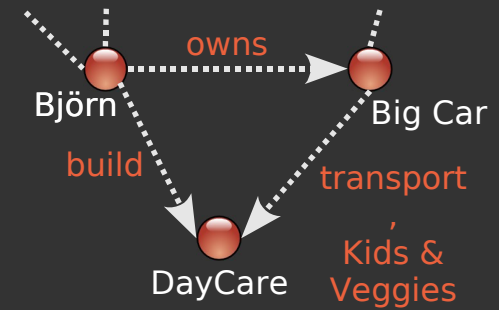
So that's v_{CPU} ...

what about v_{hackers} ?

Whiteboard friendly?

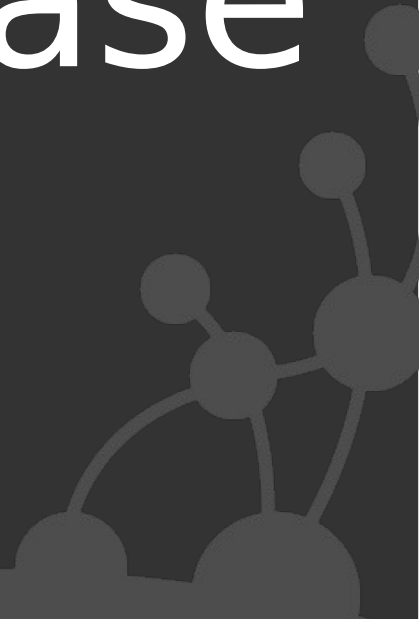


?

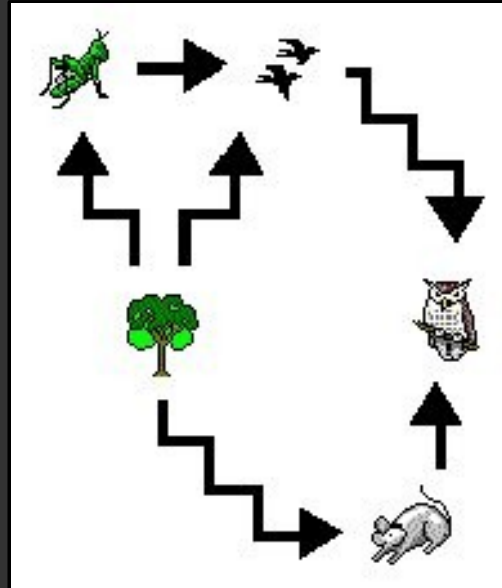


Alternative?

a graph database

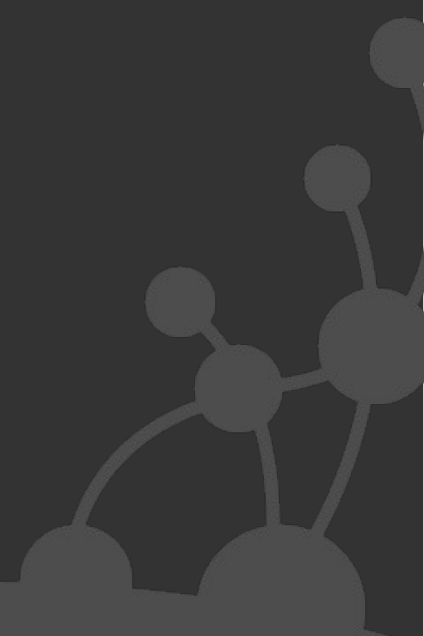


A graph

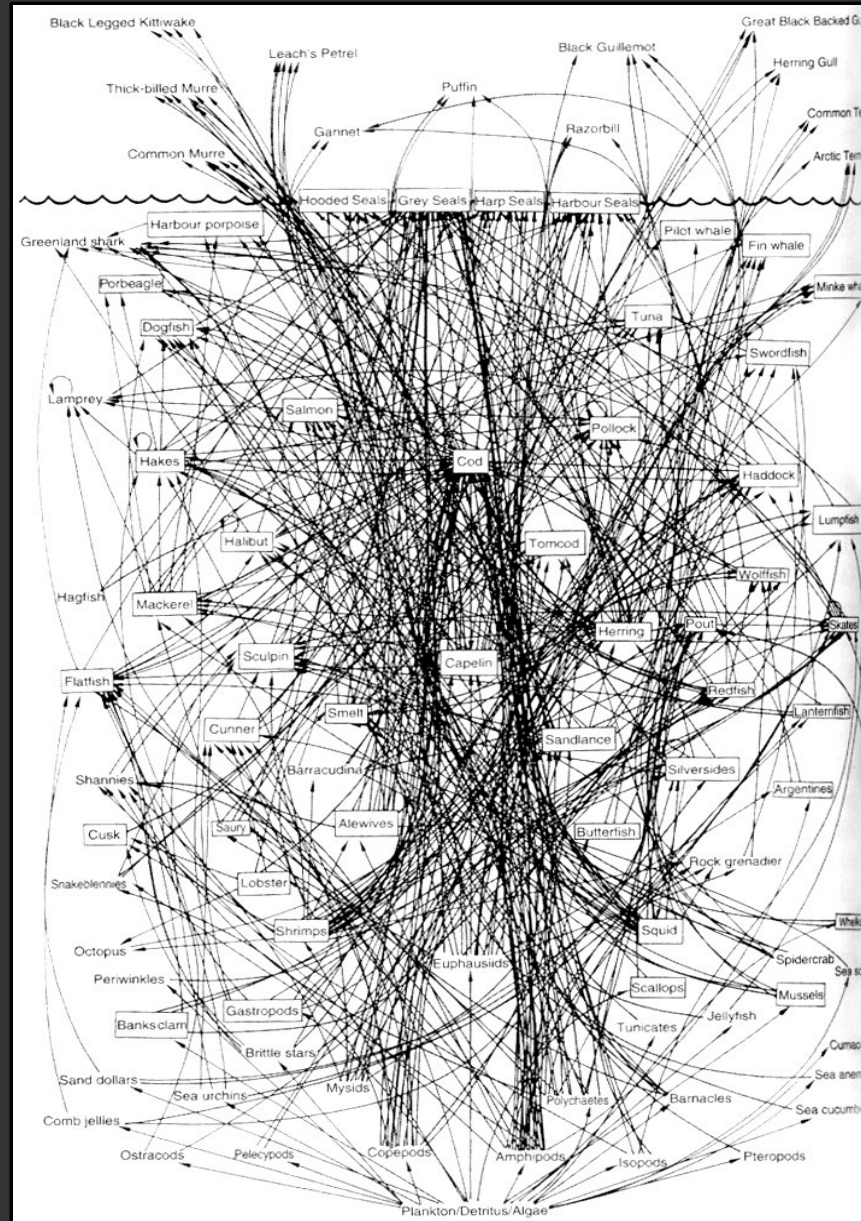


A simple food web

(image from vtaide.com)



A big graph

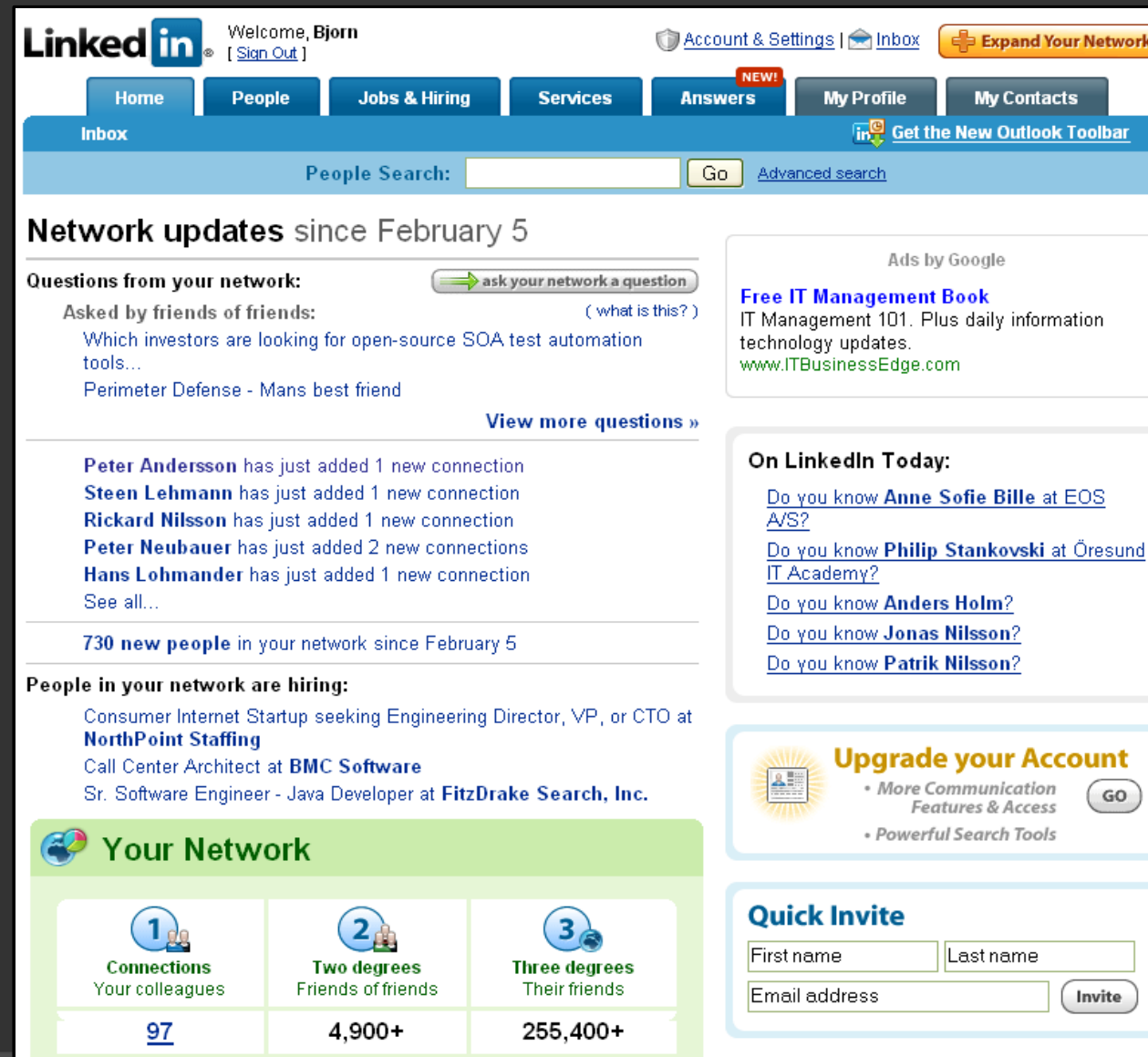


Part of the food
web of the North
Atlantic

(image from jeffkenedyassociates.com)

A social graph

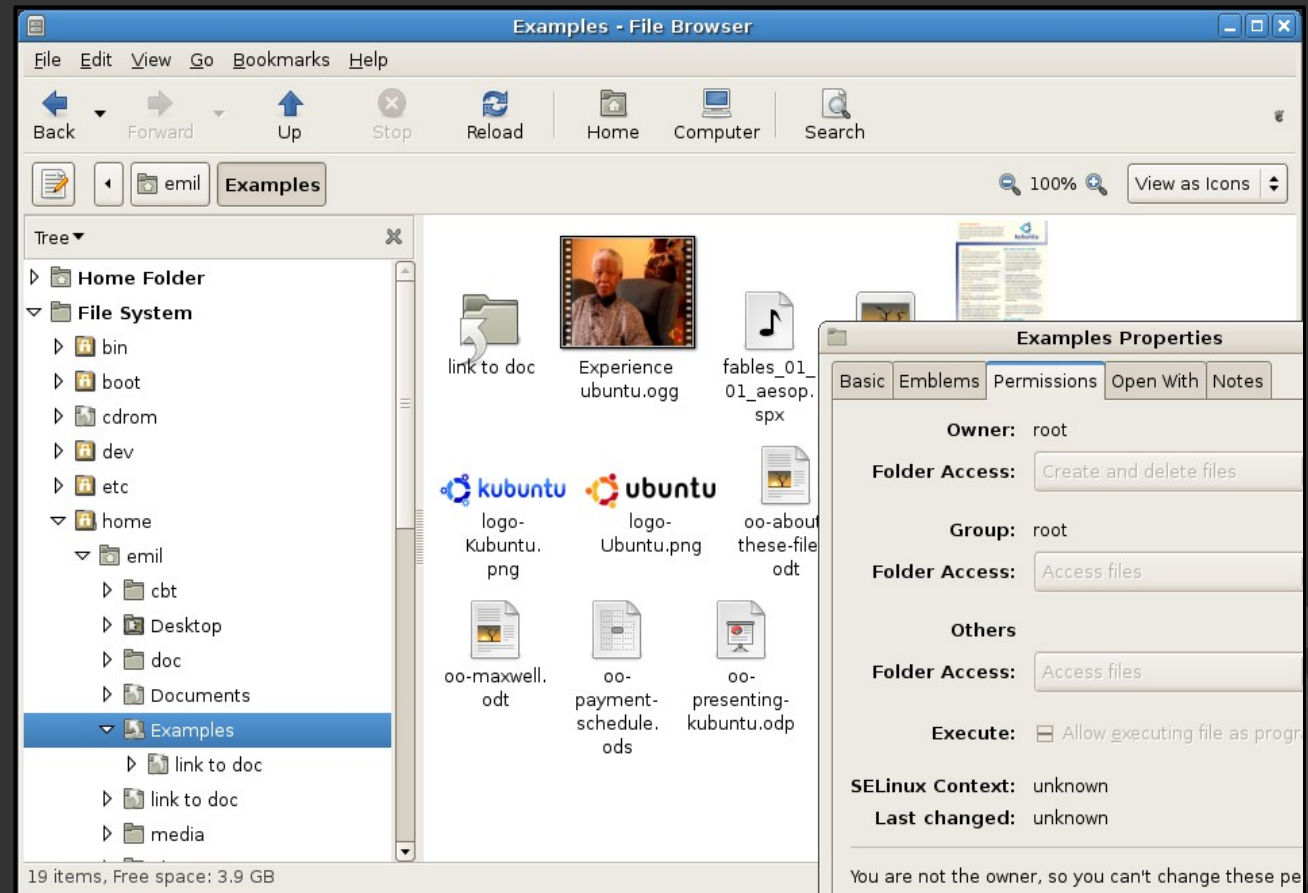
- LinkedIn
- Facebook
- Orkut
- Hi5
- Friendster
- Dopplr
- ...



The screenshot shows the LinkedIn homepage for a user named Bjorn. The page features a navigation bar with links for Home, People, Jobs & Hiring, Services, Answers, My Profile, and My Contacts. Below the navigation bar is an 'Inbox' section with a 'People Search' field and a 'Go' button. The main content area is titled 'Network updates since February 5' and includes sections for 'Questions from your network', 'Peter Andersson has just added 1 new connection', '730 new people in your network since February 5', and 'People in your network are hiring:'. A 'Your Network' section at the bottom shows statistics for connections, two degrees, and three degrees. On the right side, there are advertisements for 'Free IT Management Book' and 'Upgrade your Account', along with a 'Quick Invite' form.

Your file system

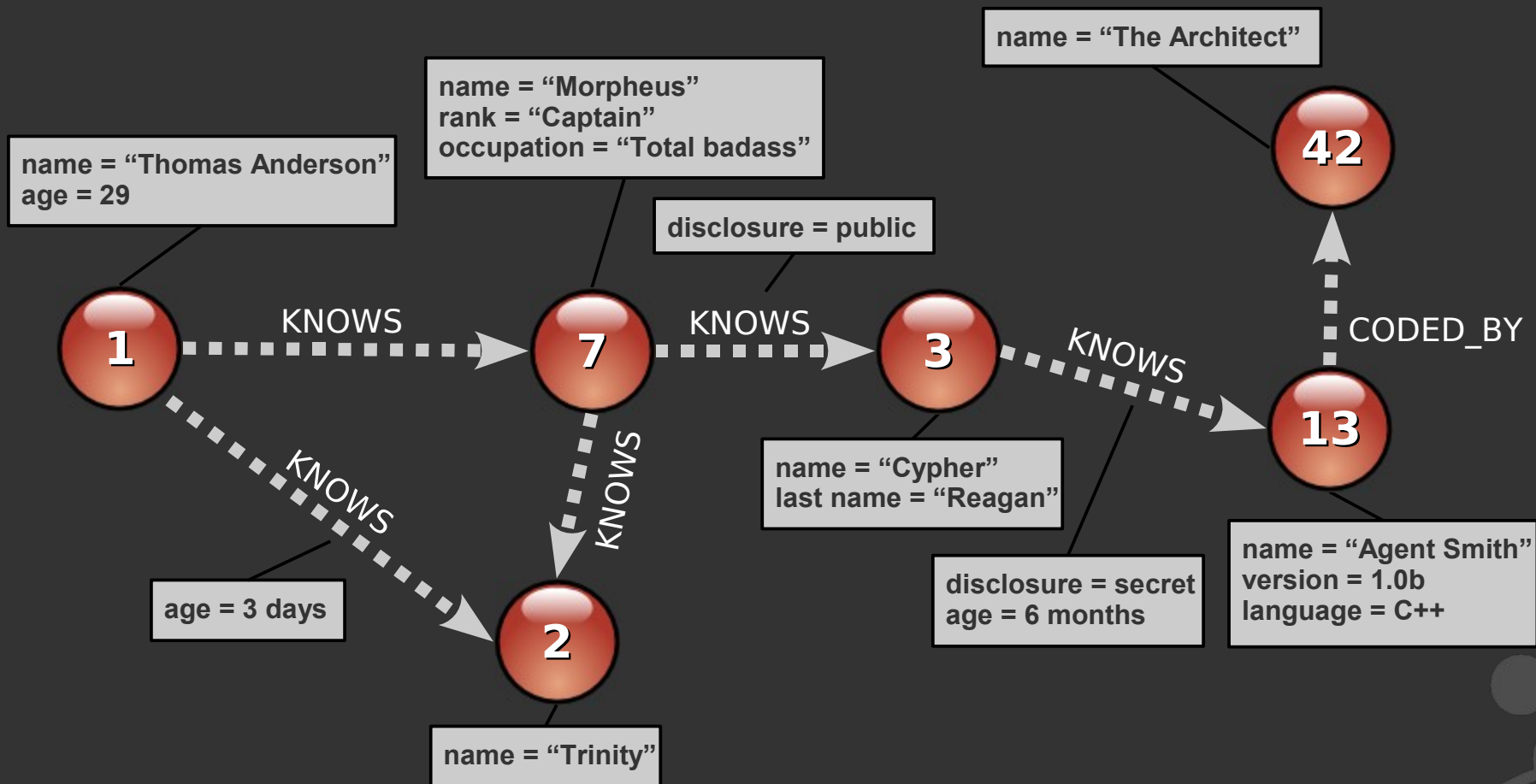
- Files & folders
- Alias, links
- Read & write
- Roles, groups
- ...



Shut up and
show us the
code!



Example: The Matrix



Code (1): Building a node space

```
NeoService neo = ... // Get factory

// Create Thomas 'Neo' Anderson
Node mrAnderson = neo.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = neo.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly
```

Code (1): Building a node space

```
NeoService neo = ... // Get factory
Transaction tx = neo.begin();

// Create Thomas 'Neo' Anderson
Node mrAnderson = neo.createNode();
mrAnderson.setProperty( "name", "Thomas Anderson" );
mrAnderson.setProperty( "age", 29 );

// Create Morpheus
Node morpheus = neo.createNode();
morpheus.setProperty( "name", "Morpheus" );
morpheus.setProperty( "rank", "Captain" );
morpheus.setProperty( "occupation", "Total bad ass" );

// Create a relationship representing that they know each other
mrAnderson.createRelationshipTo( morpheus, RelTypes.KNOWS );
// ...create Trinity, Cypher, Agent Smith, Architect similarly

tx.commit();
```

Code (1b): Defining RelationshipTypes

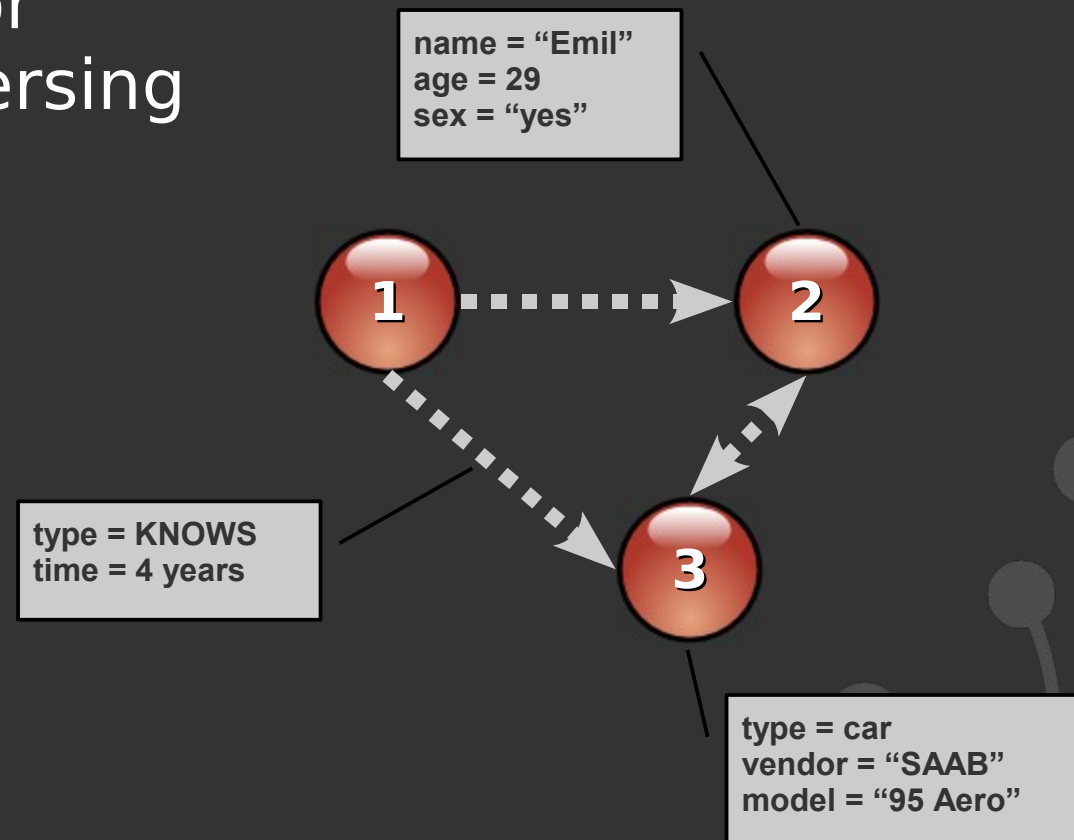
```
// In package org.neo4j.api.core
public interface RelationshipType
{
    String name();
}

// In package org.yourdomain.yourapp
// Example on how to roll dynamic RelationshipTypes
class MyDynamicRelType implements RelationshipType
{
    private final String name;
    MyDynamicRelType( String name ){ this.name = name; }
    public String name() { return this.name; }
}

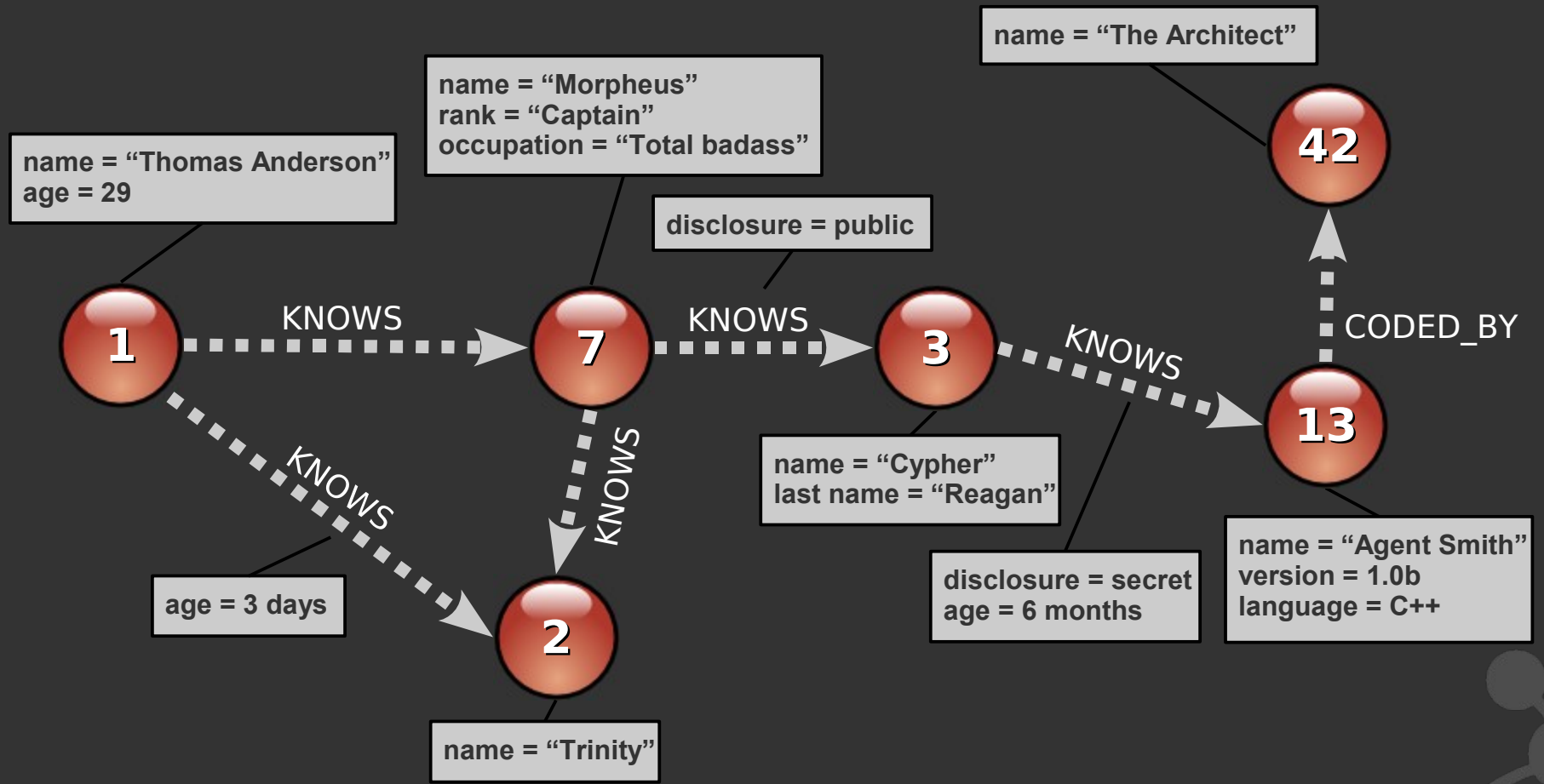
// Example on how to kick it, static-RelationshipType-like
enum MyStaticRelTypes implements RelationshipType
{
    KNOWS,
    WORKS_FOR,
}
```

The Graph DB model: traversal

- ◎ Traverser framework for high-performance traversing across the node space



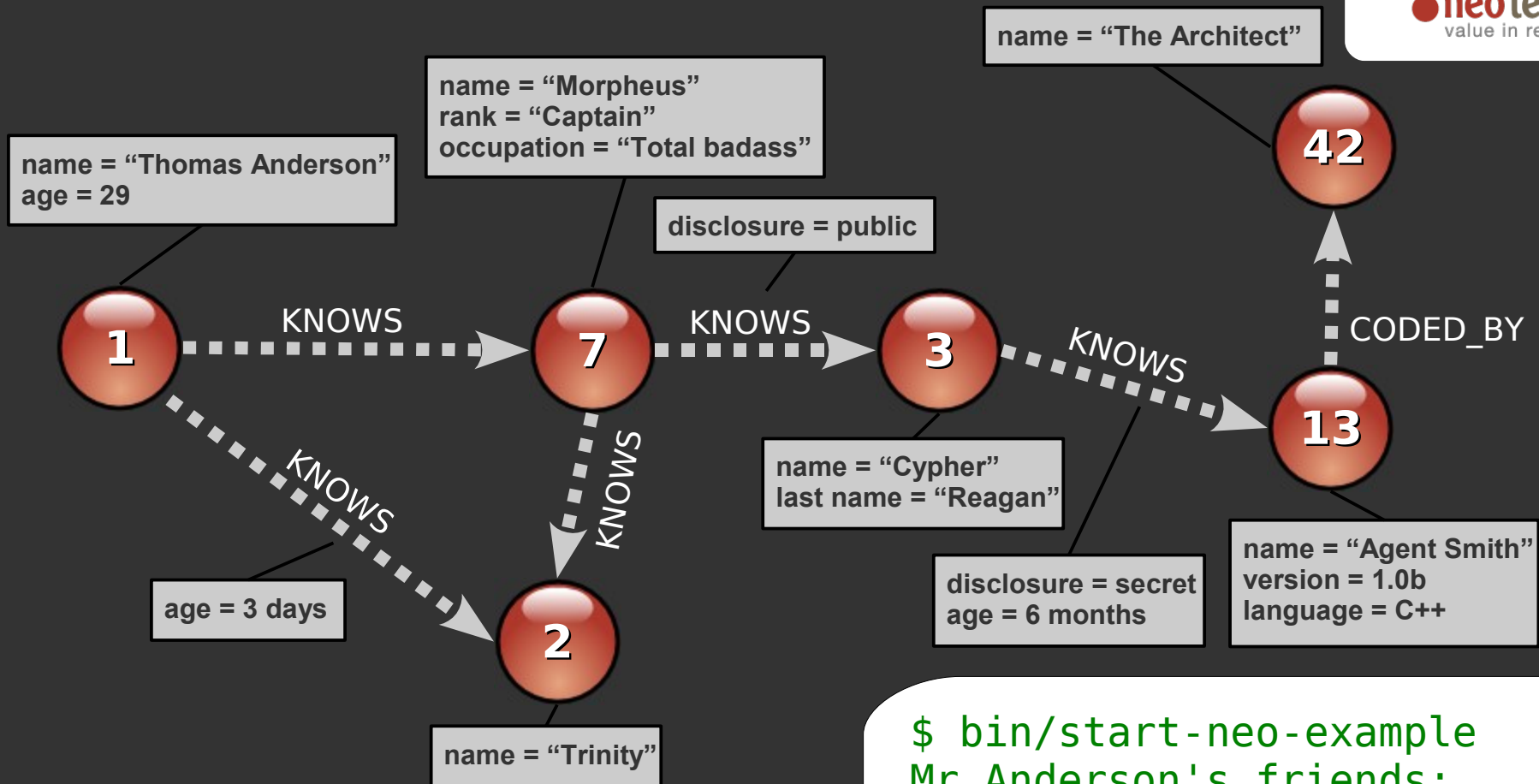
Example: Mr Anderson's friends



Code (2): Traversing a node space

```
// Instantiate a traverser that returns Mr Anderson's friends
Traverser friendsTraverser = mrAnderson.traverse(
    Traverser.Order.BREADTH_FIRST,
    StopEvaluator.END_OF_GRAPH,
    ReturnableEvaluator.ALL_BUT_START_NODE,
    RelTypes.KNOWS,
    Direction.OUTGOING );

// Traverse the node space and print out the result
System.out.println( "Mr Anderson's friends:" );
for ( Node friend : friendsTraverser )
{
    System.out.printf( "At depth %d => %s%n",
        friendsTraverser.currentPosition().getDepth(),
        friend.getProperty( "name" ) );
}
```



```

friendsTraverser = mrAnderson.traverse(
  Traverser.Order.BREADTH_FIRST,
  StopEvaluator.END_OF_GRAPH,
  ReturnableEvaluator.ALL_BUT_START_NODE,
  RelTypes.KNOWS,
  Direction.OUTGOING );
  
```

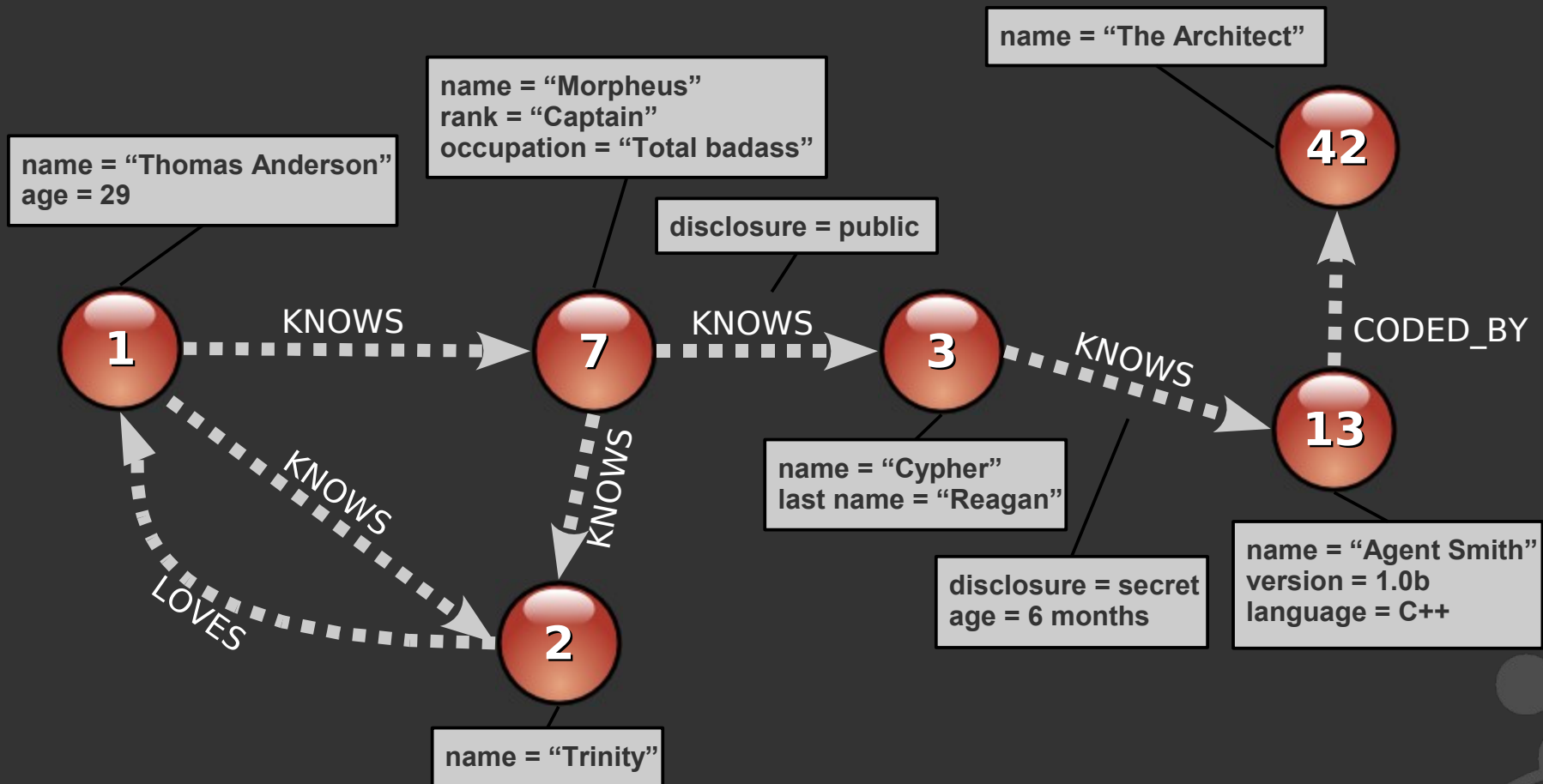
```

$ bin/start-neo-example
Mr Anderson's friends:
  
```

```

At depth 1 => Morpheus
At depth 1 => Trinity
At depth 2 => Cypher
At depth 3 => Agent Smith
$
  
```

Example: Friends in love?

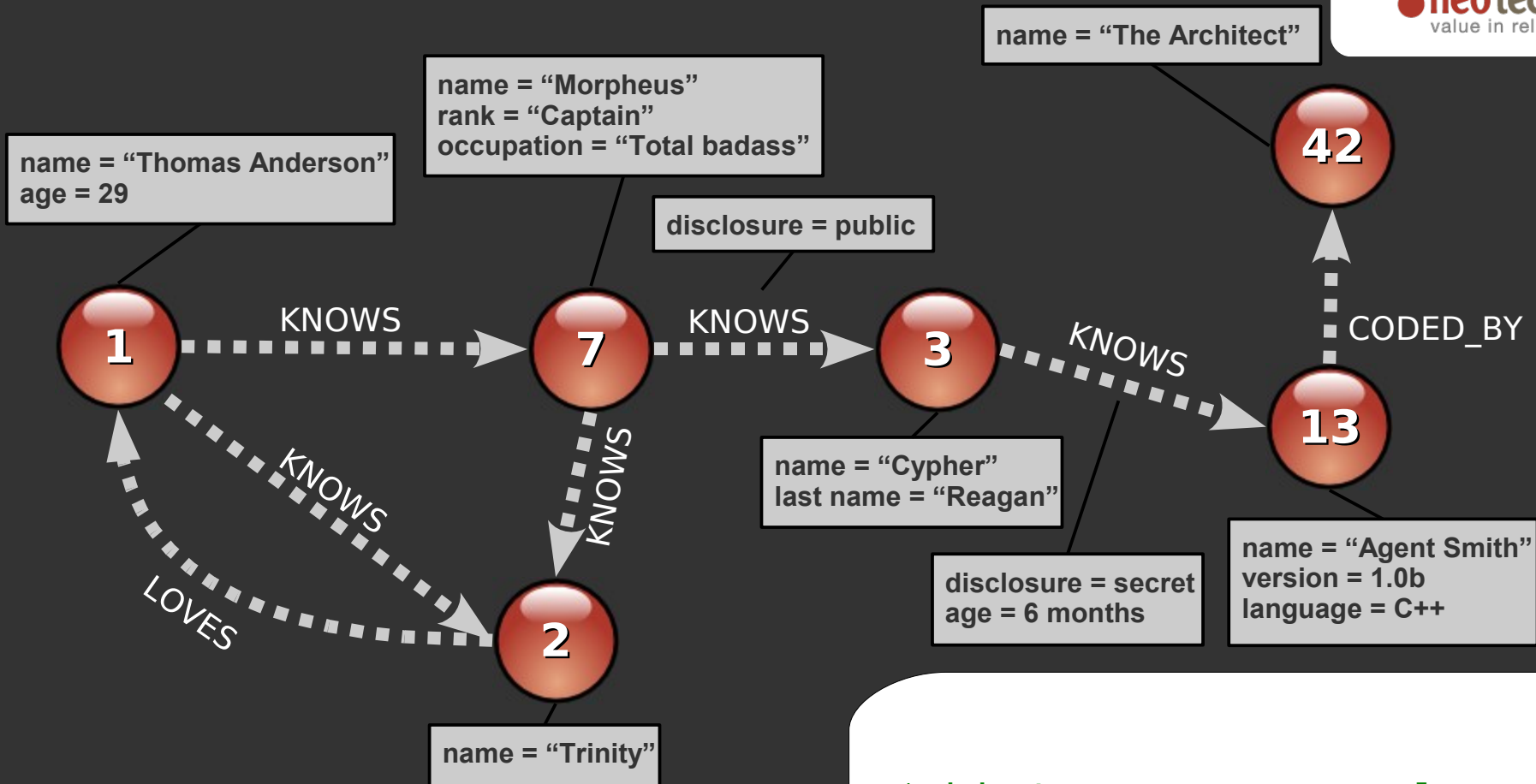


Code (3a): Custom traverser

```
// Create a traverser that returns all "friends in love"
Traverser loveTraverser = mrAnderson.traverse(
    Traverser.Order.BREADTH_FIRST,
    StopEvaluator.END_OF_GRAPH,
    new ReturnableEvaluator()
    {
        public boolean isReturnableNode( TraversalPosition pos )
        {
            return pos.currentNode().hasRelationship(
                RelTypes.LOVES, Direction.OUTGOING );
        }
    },
    RelTypes.KNOWS,
    Direction.OUTGOING );
```

Code (3a): Custom traverser

```
// Traverse the node space and print out the result  
System.out.println( "Who's a lover?" );  
for ( Node person : loveTraverser )  
{  
    System.out.printf( "At depth %d => %s%n",  
        loveTraverser.currentPosition().getDepth(),  
        person.getProperty( "name" ) );  
}
```



```

new ReturnableEvaluator()
{
  public boolean isReturnableNode(
    TraversalPosition pos)
  {
    return pos.currentNode().
      hasRelationship( RelTypes.LOVES,
        Direction.OUTGOING );
  }
},

```

```

$ bin/start-neo-example
Who's a lover?

```

```

At depth 1 => Trinity
$

```

Bonus code: domain model

- ◎ How do you implement your domain model?
- ◎ Use the delegator pattern, i.e. every domain entity wraps a Neo4j primitive:

```
// In package org.yourdomain.yourapp
class PersonImpl implements Person
{
    private final Node underlyingNode;
    PersonImpl( Node node ){ this.underlyingNode = node; }

    public String getName()
    {
        return this.underlyingNode.getProperty( "name" );
    }
    public void setName( String name )
    {
        this.underlyingNode.setProperty( "name", name );
    }
}
```

Domain layer frameworks

◎ Qi4j (www.qi4j.org)



- Framework for doing DDD in pure Java5
- Defines Entities / Associations / Properties
 - Sound familiar? Nodes / Rel's / Properties!
- Neo4j is an “EntityStore” backend

◎ NeoWeaver (<http://components.neo4j.org/neo-weaver>)

- Weaves Neo4j-backed persistence into domain objects in runtime (dynamic proxy / cglib based)
- Veeeery alpha, but veeery cool

Neo4j system characteristics

◎ Disk-based

- Native graph storage engine with custom (“SSD-ready”) binary on-disk format

◎ Transactional

- JTA/JTS, XA, 2PC, Tx recovery, deadlock detection, etc

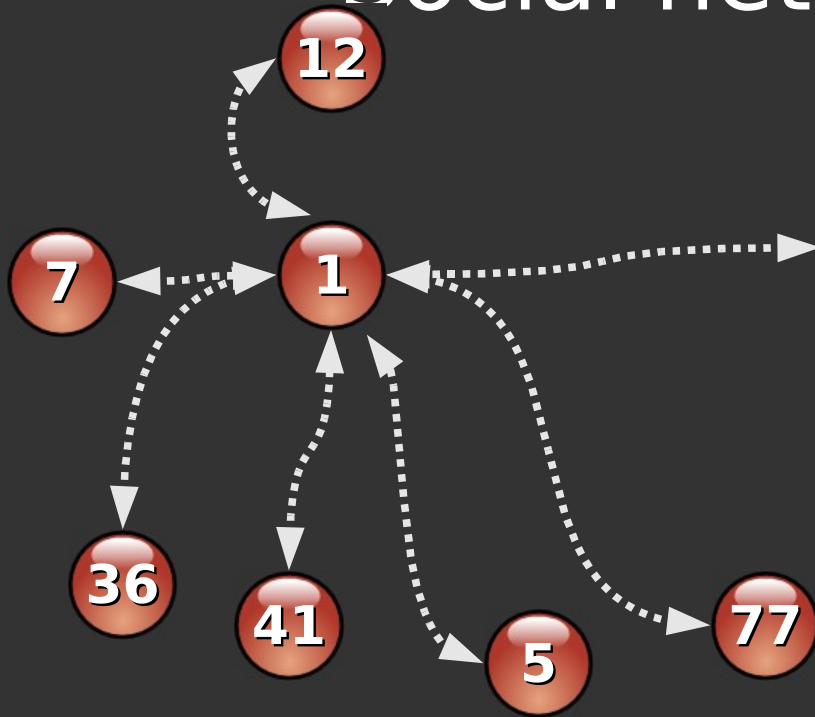
◎ Scalable

- Several billions of nodes/rels/props on single JVM

◎ Robust

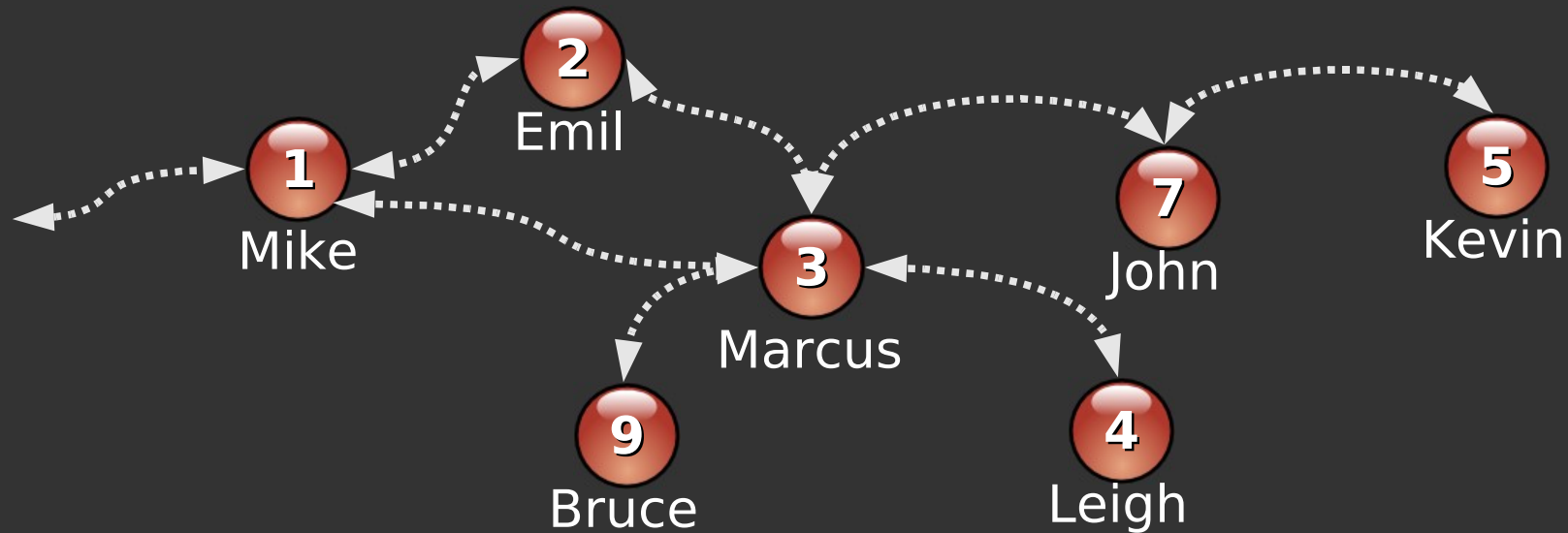
- 5+ years in 24/7 production

Social network *pathExists()*



- ~1k persons
- Avg 50 friends per person
- *pathExists(a, b)* limit depth 4
- Two backends
- Eliminate disk IO so warm up caches

Social network *pathExists()*



Relational database (MySQL)
Graph database (Neo4j)
Graph database (Neo4j)

persons query time

Pros & Cons compared to RDBMS

- + No O/R impedance mismatch (*whiteboard friendly*)
- + Can easily evolve schemas
- + Can represent semi-structured info
- + Can represent graphs/networks (*with* performance)
- Lacks in tool and framework support
- No other implementations => potential lock in
- + ~~No support for ad-hoc queries~~

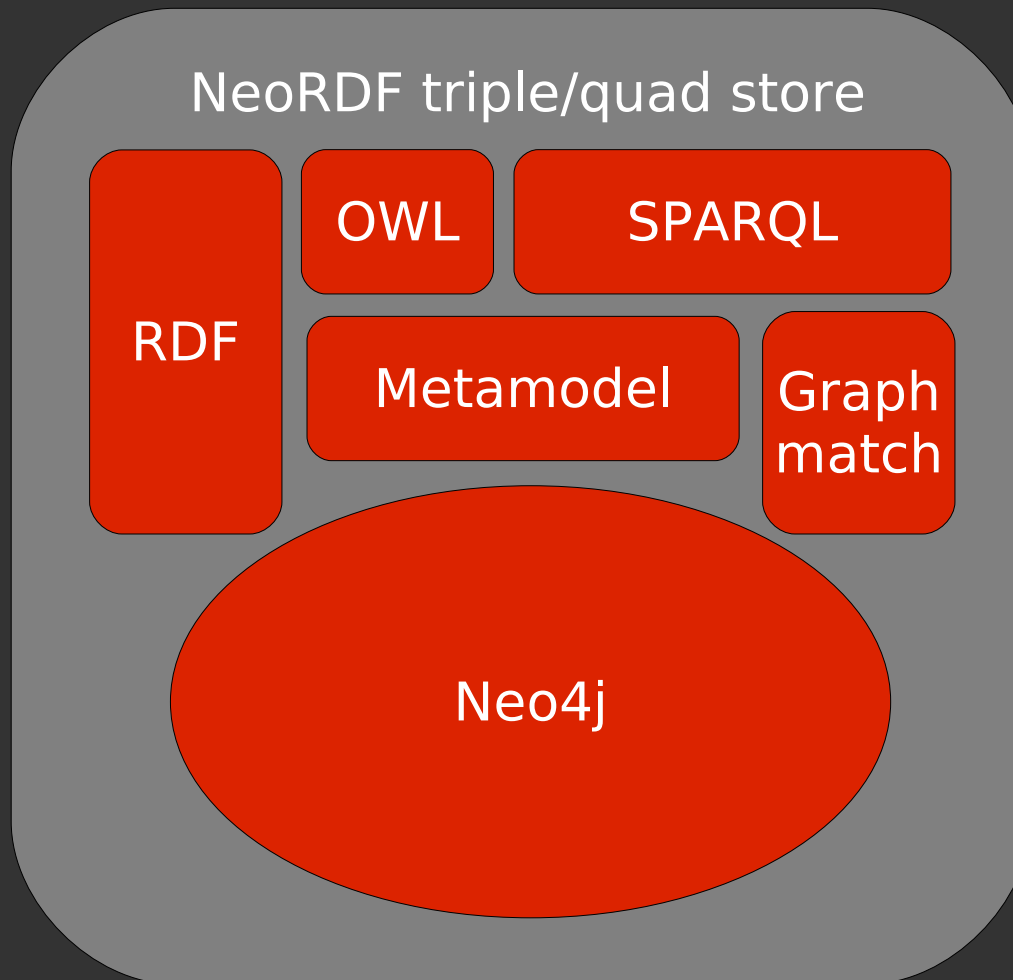
More consequences

- ◎ Ability to capture semi-structured information
 - => allowing individualization of content
- ◎ No predefined schema
 - => easier to evolve model
 - => can capture ad-hoc relationships
- ◎ Can capture non-normative relations
 - => easy to model specific links to specific sets
- ◎ All state is kept in transactional memory
 - => improves application concurrency

The Neo4j ecosystem

- ◎ Neo4j is an embedded database
 - Tiny teeny lil jar file
- ◎ Component ecosystem
 - index-util
 - neo-meta
 - neo-utils
 - owl2neo
 - sparql-engine
 - ...
- ◎ See <http://components.neo4j.org>

Example: NeoRDF




Future development

- ◎ Productify RDF support (Neo4j 1.1)
- ◎ Tool support (Neo4j 1.1 and onwards)
- ◎ Language bindings
 - Currently Jython, Python, Ruby
 - Probably works well with Groovy, Beanshell, etc
 - Tomorrow Scala? .NET? Erlang?
- ◎ Standalone server
 - Experimental RemoteNeo in laboratory right now
 - How standardize REST API? SPARQL protocol?

Future development

◎ Distribution (Neo4j 2.0), current thoughts:

- Best bet today: sharding on top of  Newton.
(Infiniflow) from Paremus: www.codecauldron.org
- Fundamentals:
 - CAP theorem
 - BASE (“ACID 2.0”)
 - Eventual consistency
- Separate HA and data partitioning
- Generic clustering algorithm as base case, but give lots of knobs for developers

How ego are you? (aka other impls?)

- ◎ Franz' **AllegroGraph** (<http://agraph.franz.com>)
 - Proprietary, Lisp, RDF-oriented but real graphdb
- ◎ FreeBase **graphd** (<http://blog.freebase.com/2008/04/09/a-brief-tour-of-graphd/>)
 - In-house at Metaweb
- ◎ **Kloudshare** (<http://whydoeseverythingsuck.com>)
 - Graph database in the cloud, still stealth mode
- ◎ Some academic papers from ~10 years ago
 - $G = \{V, E\}$

Conclusion

- ◎ Graphs & Neo4j => teh awesome!
- ◎ Available NOW under AGPLv3 / commercial license
 - AGPLv3: “if you’re open source, we’re open source”
 - If you have proprietary software? Must buy a commercial license
 - But up to 1M primitives it’s free for all uses!
- ◎ Download
 - <http://neo4j.org>
- ◎ Feedback
 - <http://lists.neo4j.org>

Questions?



Image credit: lost again! Sorry :(



<http://neotechnology.com>

Neo4j architecture gotchas

- ◎ Focus on the domain (whiteboard friendly – domain-first development)
- ◎ Purpose of the domain layer:
 - “an adaptation of the generic node space to a type-safe, object-oriented abstraction expressed in the vocabulary of our domain” (!)
- ◎ Implementation mindset
 - Assume the node space is always in memory
 - Assume everything is automatically persistent
 - Focus on logical transactions instead of artificial load/stores

Implementation pointers

- ◎ Use the delegator pattern, i.e. every domain entity wraps a Neo4j primitive as follows:
 - `Actor actor = new ActorImpl(underlyingNode);`
- ◎ Remember that the wrappers are extremely lightweight – they contain no state except for a reference – so create them freely!
- ◎ It is good practice to override equals/hashCode
 - Delegate to `underlying{Node, Relationship}`'s `equals()` or `hashCode()` implementation