ThingWorx

www.thingworx.com

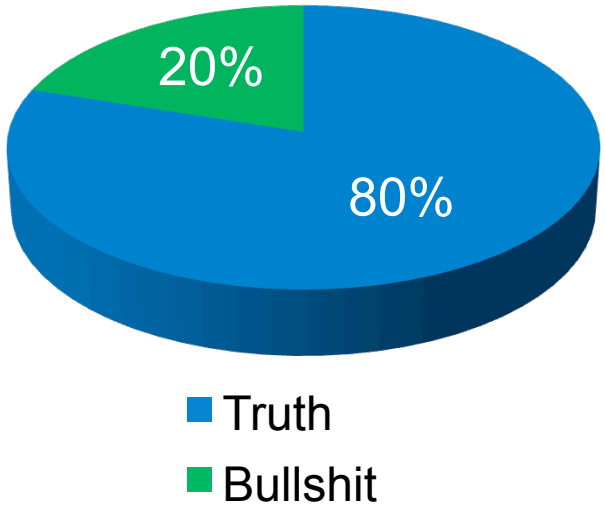
neo technology
value in relationships

www.neotechnology.com

Using A Graph Database To Power The "Web of Things"

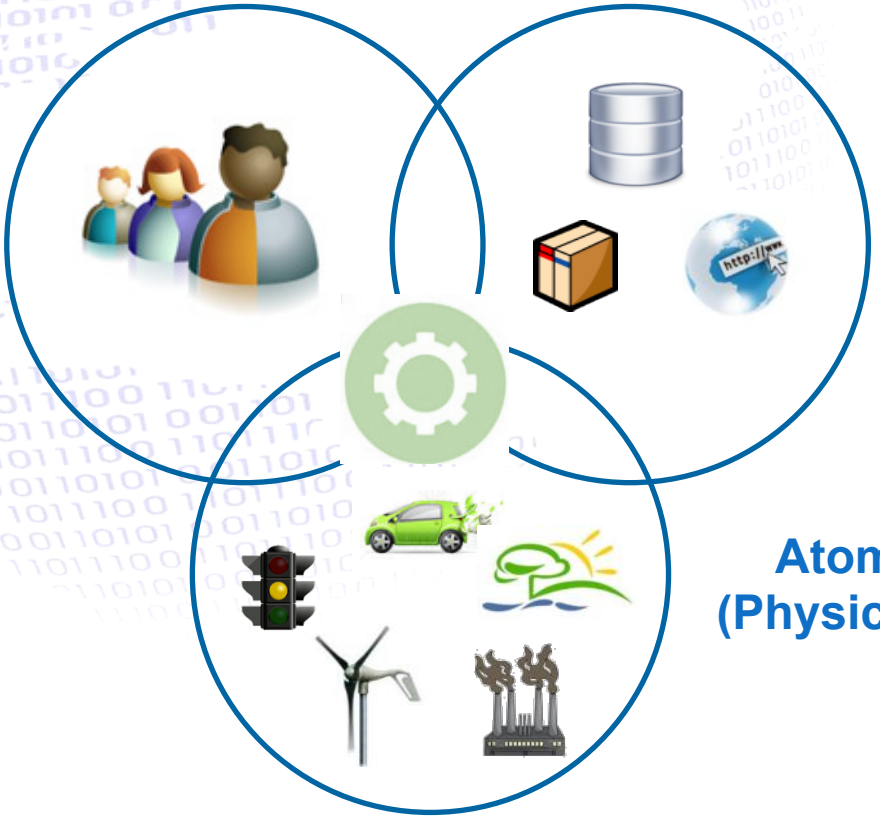
Rick Bullotta, CTO at ThingWorx
Emil Eifrem, CEO at Neo Technology

Standard Disclaimer

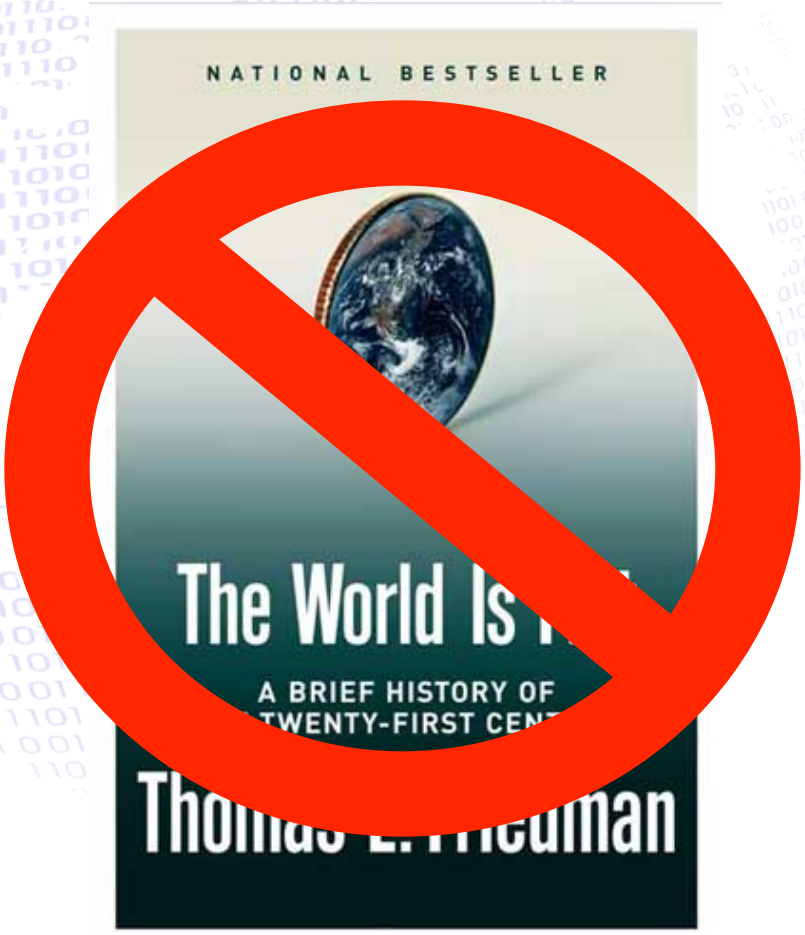


We're Solving a Large, Difficult Problem

Built the 1st platform designed for applications that integrate people, systems and the physical world

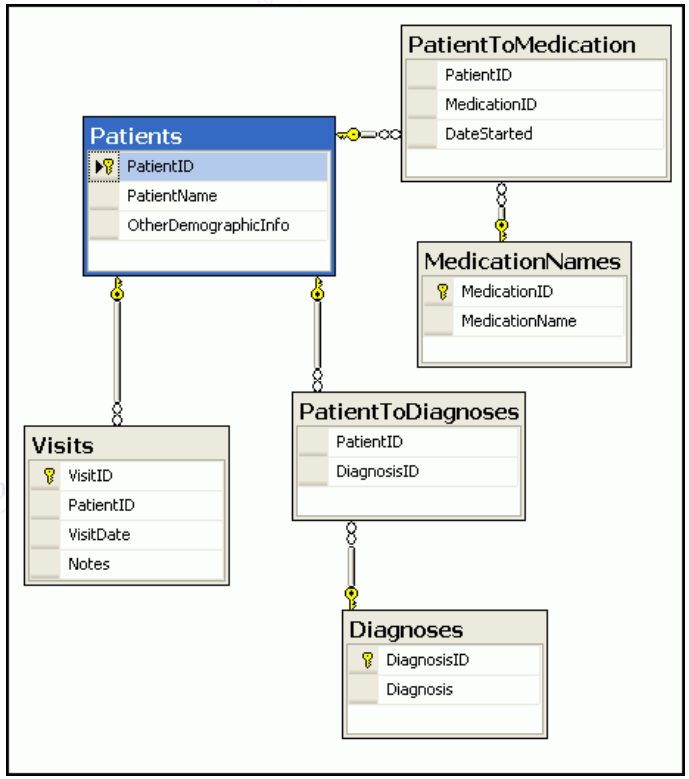


The world is definitely not flat…

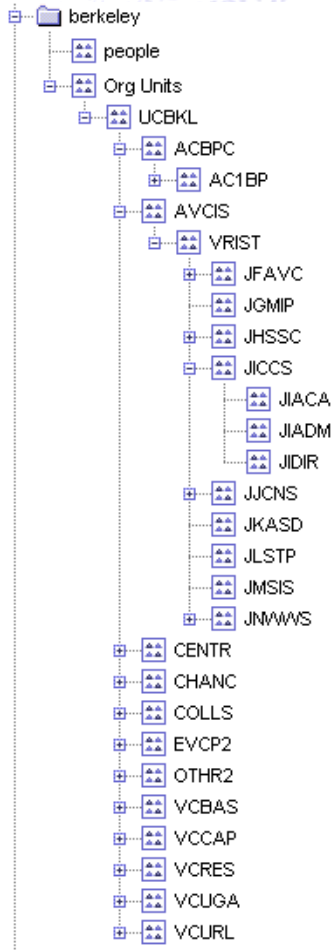


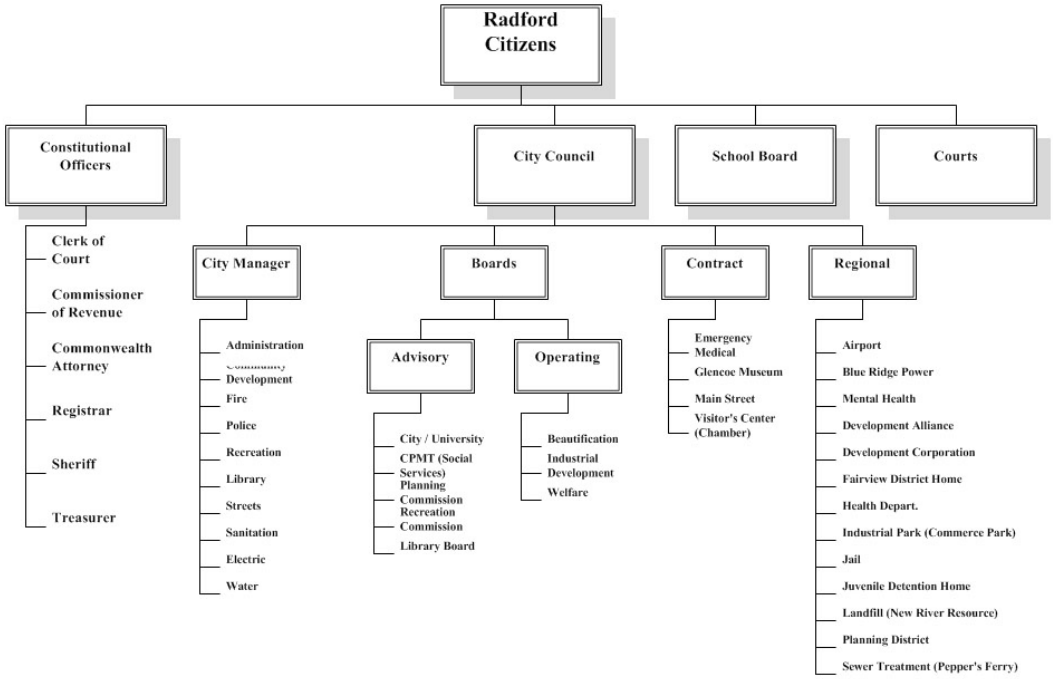
It's also not (only) tabular…

	Operation Date	Operation Direction	Operation Type	Release Time	Close Time	Maturity Start	Maturity End	# Issues	Total Accepted (M)	Total Submitted (M)
2	8/25/2005 0:00	P	COUPON	10:30 AM	11:00 AM	9/30/2005 0:00	8/15/2006 0:00	22	1298	11683.5
3	9/7/2005 0:00	P	COUPON	10:30 AM	11:00 AM	11/15/2008 0:00	5/15/2009 0:00	9	1201	6597
4	9/8/2005 0:00	P	TIPS	11:00 AM	11:30 AM	1/15/2007 0:00	4/15/2032 0:00	17	450	1908
5	9/12/2005 0:00	P	COUPON	10:30 AM	11:00 AM	11/15/2007 0:00	10/15/2008 0:00	13	1100	6029
6	9/21/2005 0:00	P	BILL	10:30 AM	11:00 AM	1/12/2006 0:00	3/16/2006 0:00	10	1247	11445
7	9/22/2005 0:00	P	COUPON	10:45 AM	11:15 AM	6/15/2009 0:00	1/15/2010 0:00	11	1104	6491
8	10/4/2005 0:00	P	COUPON	10:30 AM	11:00 AM	10/31/2006 0:00	7/31/2007 0:00	18	1193	8334.8
9	10/25/2005 0:00	P	COUPON	10:30 AM	11:00 AM	1/31/2006 0:00	10/31/2006 0:00	18	1000	9746
10	10/26/2005 0:00	P	COUPON	10:45 AM	11:15 AM	11/15/2013 0:00	2/15/2026 0:00	35	902	10949
11	11/4/2005 0:00	P	COUPON	10:30 AM	11:00 AM	3/15/2010 0:00	8/15/2013 0:00	17	800	9652
12	11/14/2005 0:00	P	COUPON	10:30 AM	11:00 AM	1/31/2006 0:00	10/31/2006 0:00	18	1096	14944.9
13	11/18/2005 0:00	P	COUPON	10:30 AM	11:00 AM	6/15/2009 0:00	4/15/2010 0:00	13	1095	9182
14	1/4/2006 0:00	P	COUPON	10:30 AM	11:00 AM	5/15/2010 0:00	2/15/2013 0:00	15	844	9413
15	1/5/2006 0:00	P	TIPS	10:30 AM	11:00 AM	1/15/2007 0:00	4/15/2032 0:00	17	448	2550
16	1/12/2006 0:00	P	BILL	10:30 AM	11:00 AM	2/23/2006 0:00	5/4/2006 0:00	11	1249	10480
17	1/23/2006 0:00	P	COUPON	10:30 AM	11:00 AM	2/15/2014 0:00	2/15/2026 0:00	35	947	8191.5
18	1/25/2006 0:00	P	COUPON	10:30 AM	11:00 AM	11/15/2008 0:00	7/15/2009 0:00	13	1090	7008
19	1/26/2006 0:00	P	COUPON	10:30 AM	11:00 AM	9/30/2007 0:00	10/15/2008 0:00	15	1190	9743
20	2/8/2006 0:00	P	COUPON	10:15 AM	10:45 AM	11/30/2006 0:00	8/31/2007 0:00	18	1250	13775
21	2/10/2006 0:00	P	BILL	10:30 AM	11:00 AM	5/11/2006 0:00	8/10/2006 0:00	14	1244	12935
22	2/21/2006 0:00	P	COUPON	10:30 AM	11:00 AM	2/15/2007 0:00	11/30/2007 0:00	19	1248	8695

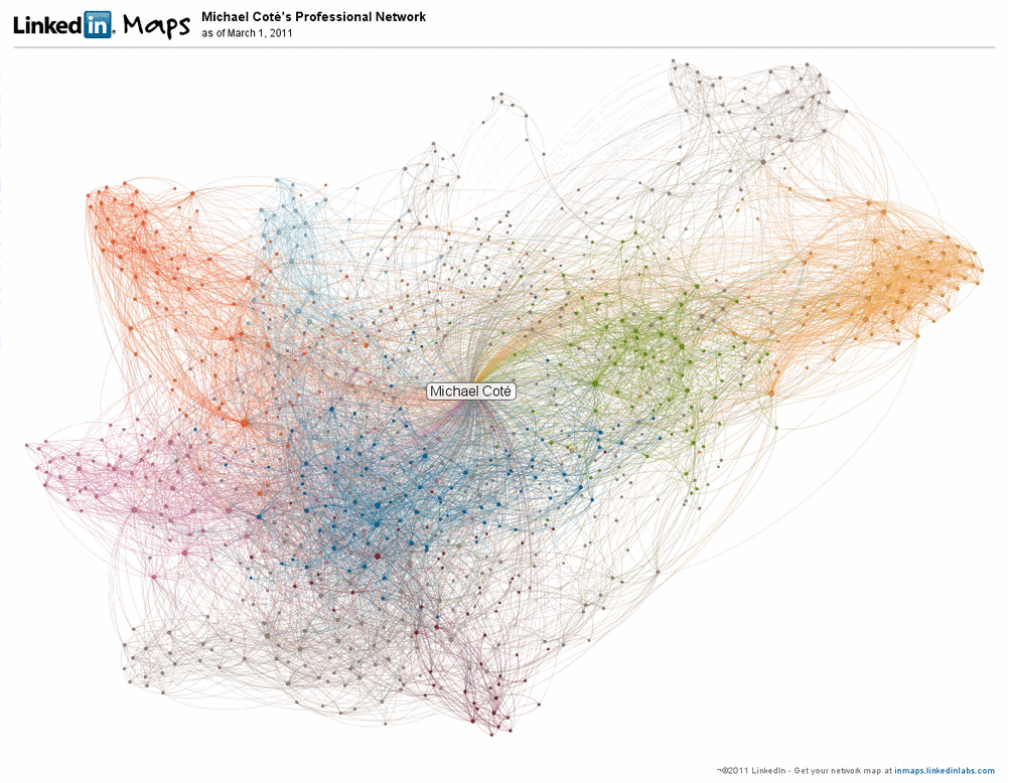


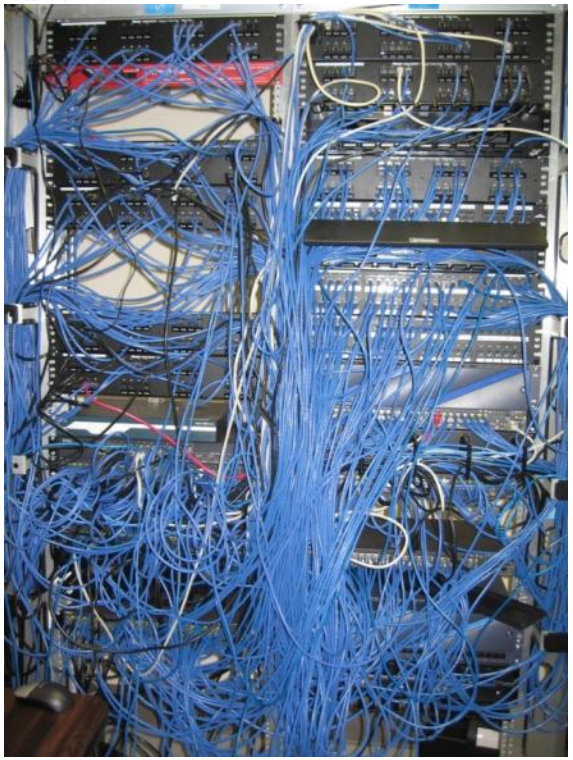
It’s certainly not (only) hierchial…





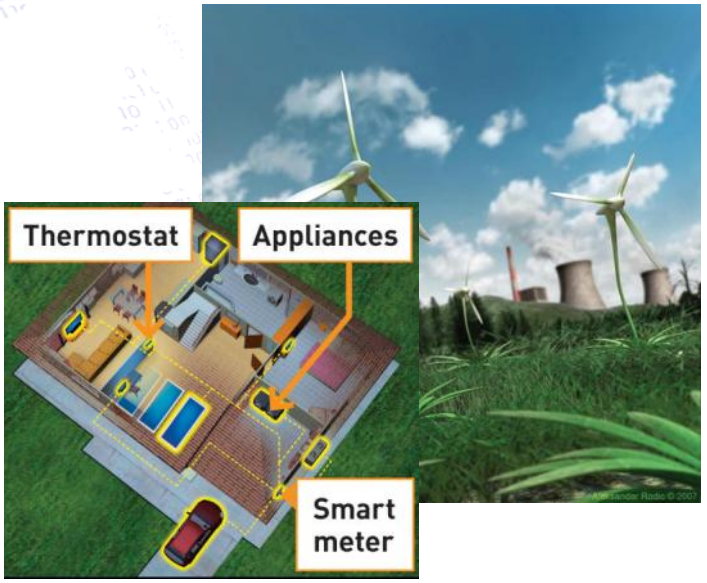
It's actually very complex, interconnected, constantly changing, and extremely large





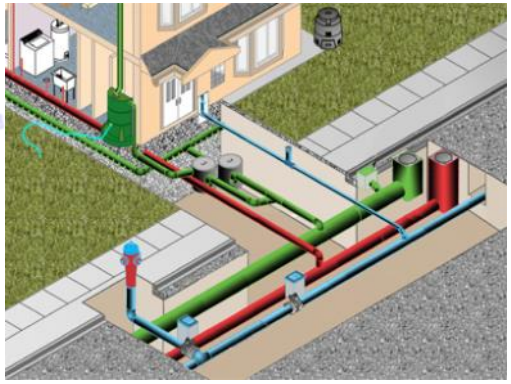
Some Factoids

A single large chemical plant generates more raw data in a day than the NYSE and AMEX combined



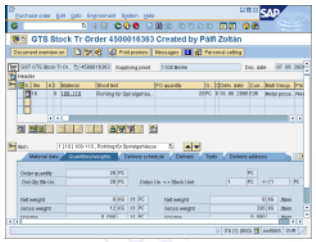
Estimates of so-called “Smart Grid”data are on the order of 35-1000 Petabytes/year

Homes, Cities, and Infrastructure Generate Lots of Data and Events, Too…



Need to Consider The New Modalities of Working and Interacting With Data

- Less structure, more ad-hoc and dynamic
- More self-service , search as an entry point
- Ubiquitous access – mobility implications



Objective:
A Platform and API for the “Real World”

- Mashup the planet
- Make every(thing) discoverable & consumable
- Augment human decision making
- Enable fluid app-to-app interaction

What's A Thing?

Atomspace

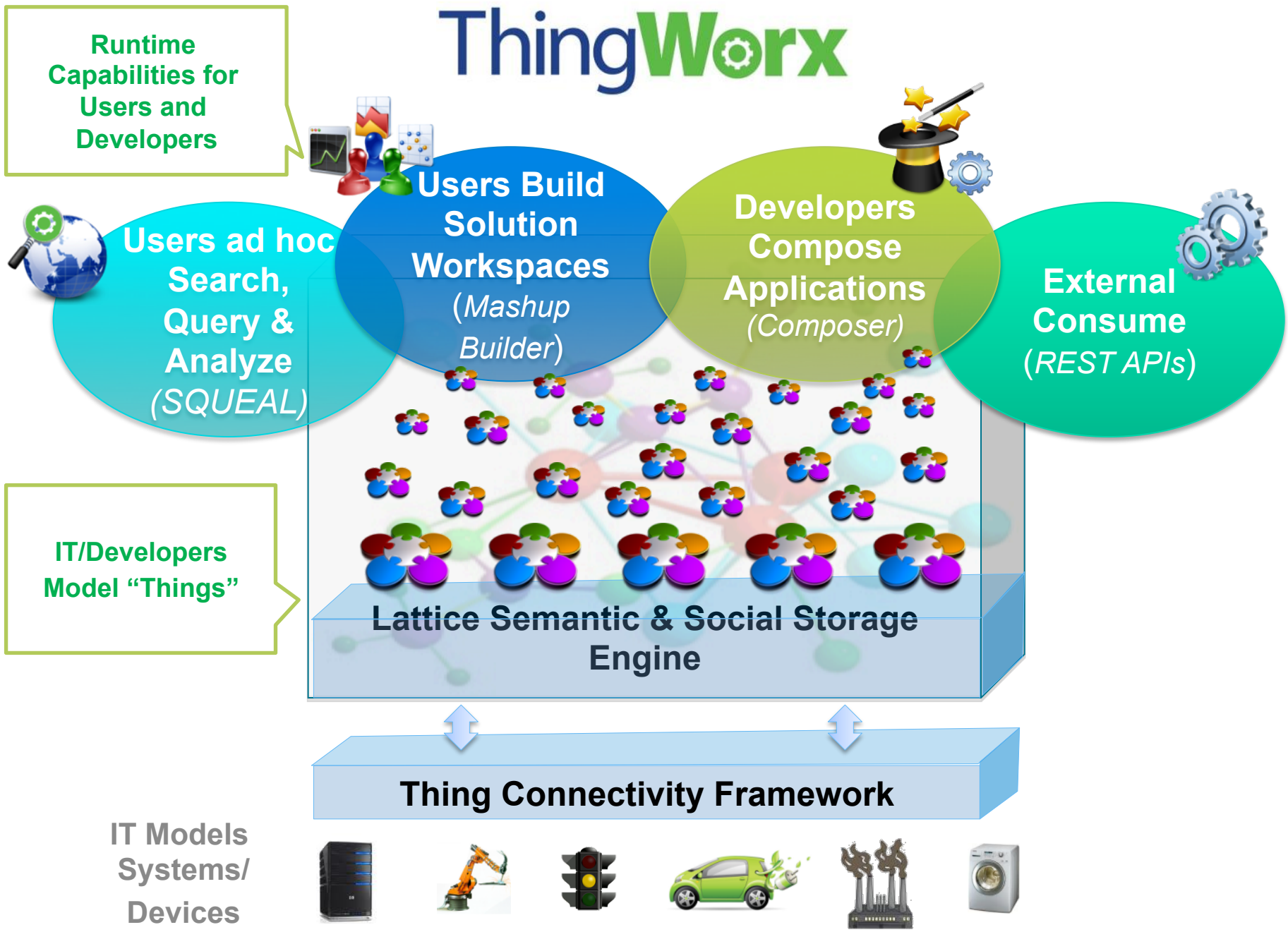
- Refrigerated Truck, Electric Car
- Filling Machine, Industrial Robot
- Power Transformer, Wind Turbine
- Shipping Container
- Building, Floor, Office
- EKG Machine

Meatspace

- People

Cyberspace

- Web platform (Blog, Twitter, S3)
- Line of business system (ERP)
- Web service, database
- E-mail or SMS gateway


Runtime Capabilities for Users and Developers
ThingWorx
Users ad hoc Search, Query & Analyze (SQUEAL)
Users Build Solution Workspaces (Mashup Builder)
Developers Compose Applications (Composer)
External Consume (REST APIs)
IT/Developers Model "Things"
Lattice Semantic & Social Storage Engine
Thing Connectivity Framework
IT Models Systems/ Devices

Requirement: Deploy Anywhere

ThingWorx scales large to small, deploys cloud, hybrid, on-premise and federated

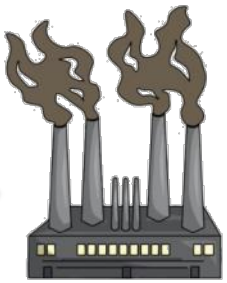
ThingWorx
Embedded

ThingWorx
Cloud

ThingWorx
Embedded

ThingWorx
Gateway

ThingWorx
Server



QCon

ThingWorx

neotechnology

Additional Design Goals

- Deal with "**Big Data**" – massive amounts of "feeds" put into the right context (Lattice), at high burst rates
- Enable answering **new kinds of questions to new kinds of problems** – unbounded domain (SQUEAL)
- **Respond quickly and flexibly to changes** and opportunities (dynamic platform)
- Support **modeling of any "real world" scenario**
- **Integrate with external data sources** on an almost-equal basis

Types of Data Storage Required for ThingWorx

- Model data
  - Persisted objects/configuration data
  - Metadata, relationships between entities
- Storage of collected data
  - Activity streams w/structured data
  - Collaboration streams
  - Table-like data
- Relationships between data and entities
  - UAC/permission models
  - “Where used”, “is a”, containment, app-specific
  - Tagging, source, time indexing

Path to a Graph Database

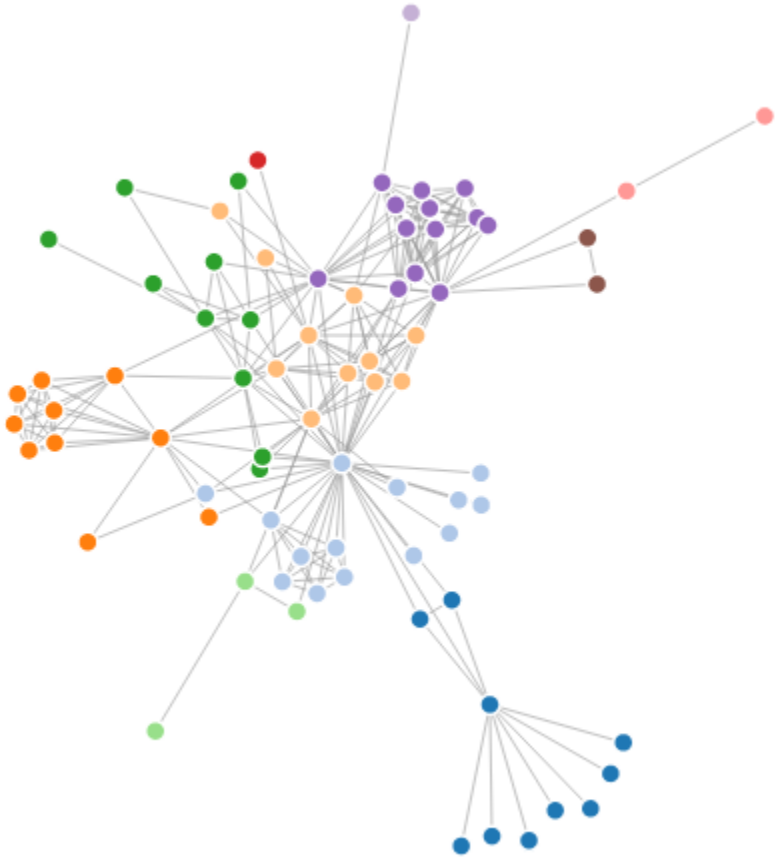
- Started with traditional RDBMS's
  - "Affordable" performance was hard to achieve
  - Not friendly to dynamically changing models
  - Not easy to model loosely coupled relationships
  - Difficult to model many of our real-world use cases, particularly relationships
  - Extremely awkward to query complex models
- Saw mention of Neo4J on Twitter!

Quick Intro To Graph Databases

QCon ThingWorx neotechnology

Graphs Can Be "Elegant In Their Simplicity"

- Nodes (Vertices)
- Relationships (Edges)
- Properties
- Can model almost anything



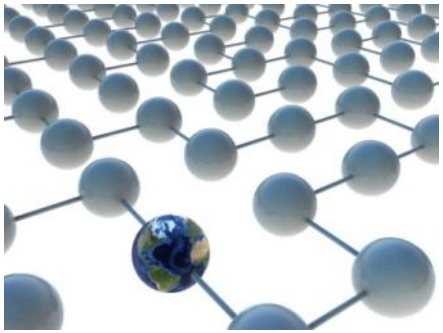
…but they can also be "Challenging In Their Simplicity"

- Not all graph databases support relationship/edge properties and types
- How can regular people, much less developers, query useful information?
- What are the "best practices" for persisting domain objects and data storage?
- Can be tricky to debug at runtime

Examples of Real-World Relationships

- That car is an electric vehicle and contains battery C91910A (all EV's contain a battery)
- My town is in Chester County which is in Pennsylvania which is in the USA
- The distance between cell towers 101 and 109 is 5 kilometers
- Pallet HD104 is in shipping container SC1101 on train car TC87154

Examples of Real-World"Networks"



- Social graphs: Twitter, Facebook, etc.
  - These represent ***extremely*** simplistic data models
  - Are your "life" networks that simple?
- Machines in a manufacturing plant
- Production, transmission, and consumption in a utility network
- A mobile communications network
- Transportation networks (air, land, sea)
- ***Mega value in "networks of networks"***

Unique Challenges of Using A Graph Database

- Inherent lack of “structure”
- Lack of a general purpose query language
- Very few domain model examples or standards
- Minimal standards on implementation or terminology
- Not all graph databases created equal

Why Neo4J?

- Supported all of our deployment scenarios
- Relationships can have types/properties
- Performance and reliability
- Platform neutral, many language bindings
- Open source and extensible
- Enabled REST API and/or embeddable
- Reasonable set of admin tools
- Very responsive dev team & community

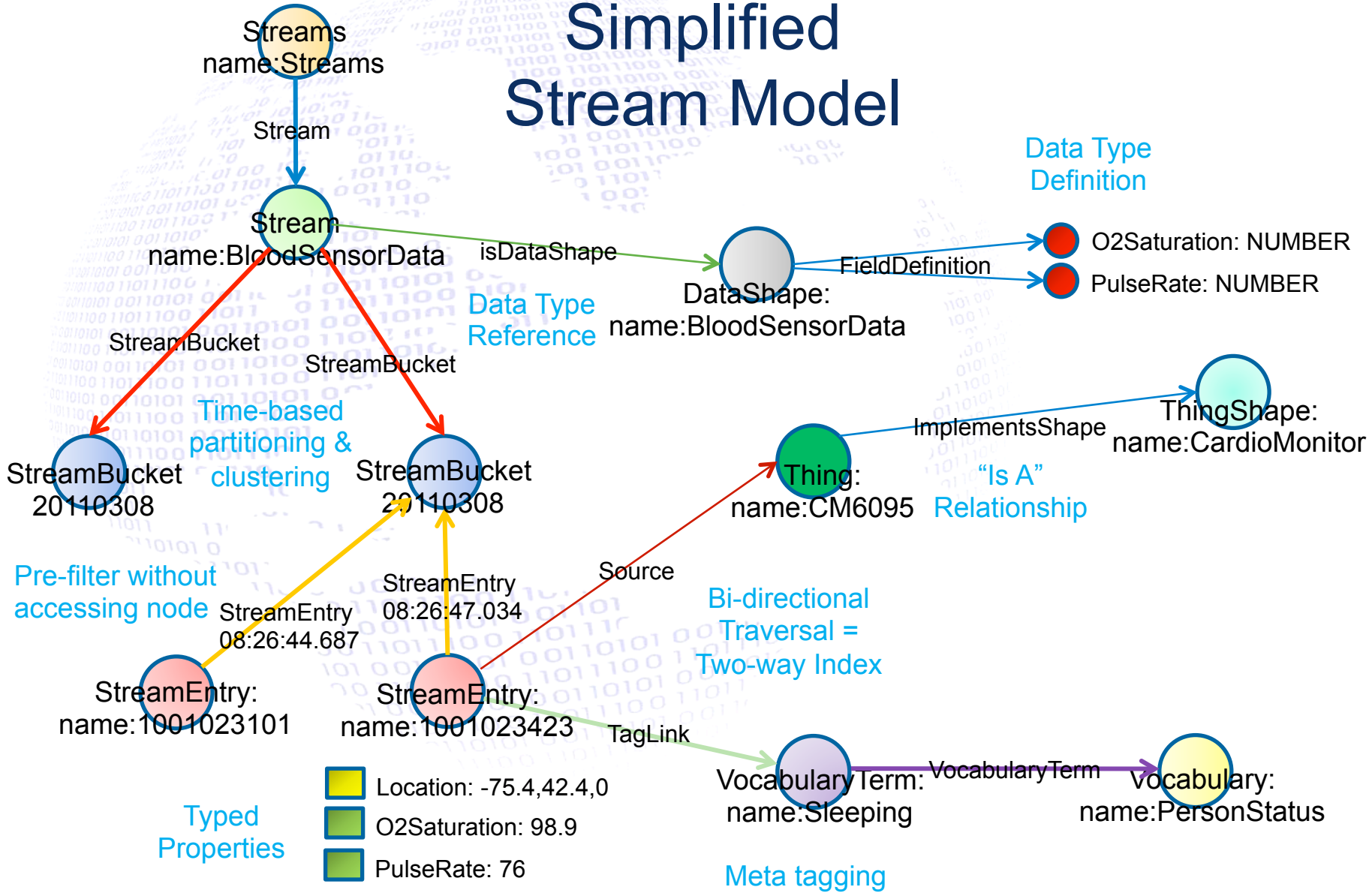
Example: Streams

- Activity streams plus more
  - Structured data packets w/richer metatagging
  - Implicit linking to hierarchy of things
- Performance was critical
  - Storage : buffered block writes (up to 10K/second)
  - Retrieval : optimized for time-based query
- Multiple perspectives
  - Cross-stream (similar to “following”)
    - Follow machines, orders, people, anything
    - Apply filters and transforms
    - Building iPhone/Android/browser apps
  - Within stream (faceted search/query/aggregates)

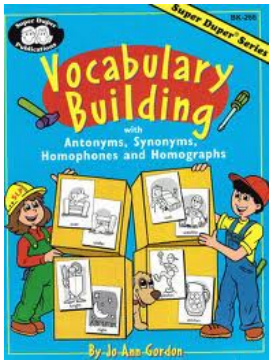
Implementation: Streams

- Data Shapes
  - Meta data for data elements/values
  - Relationship to data shape
- Time indexing
  - Buckets
  - Ordered traversal relationships
- Implicit Relationships
  - Source
  - Tagging to Vocabularies

Simplified Stream Model

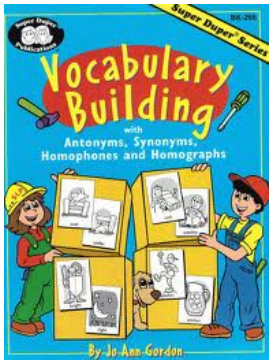


Example: Vocabularies



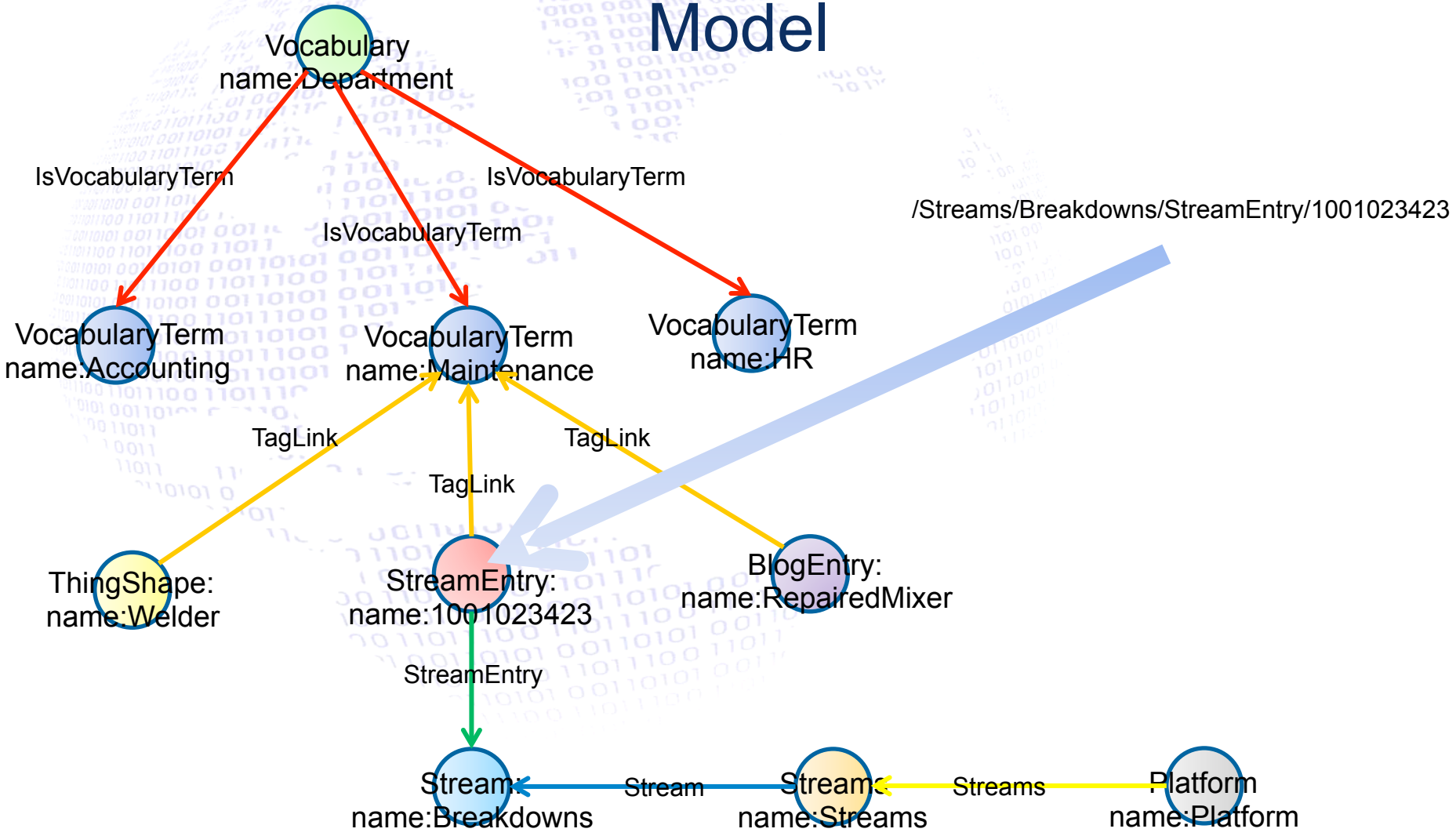
- Wanted to enable tagging of:
  - Model entities
  - Activity streams and data table rows
  - Collaboration entries (blog, wiki, forum)
  - External references (real or virtual URIs)
- Wanted some structure
  - Static versus dynamic, multiple tags
- Wanted to use these as search criteria

Solution: Vocabularies



- Leverages Relationships
- Node typing was key
  - What am I linked to?
- Dynamic URI from node hierarchy
  - Node/relationship/node/relationship
  - The model drives the REST API

Simplified Vocabulary Model



SQUEAL
(Search, Query, and Analysis)

- Synthesis of a few concepts:
  - Semantic search and keyword search
  - Faceted queries
  - Aggregation and transformation
  - Relationship search
- Required us to:
  - Leverage set processing
  - “Square up” data and
  - Develop in-memory engine on top of graph

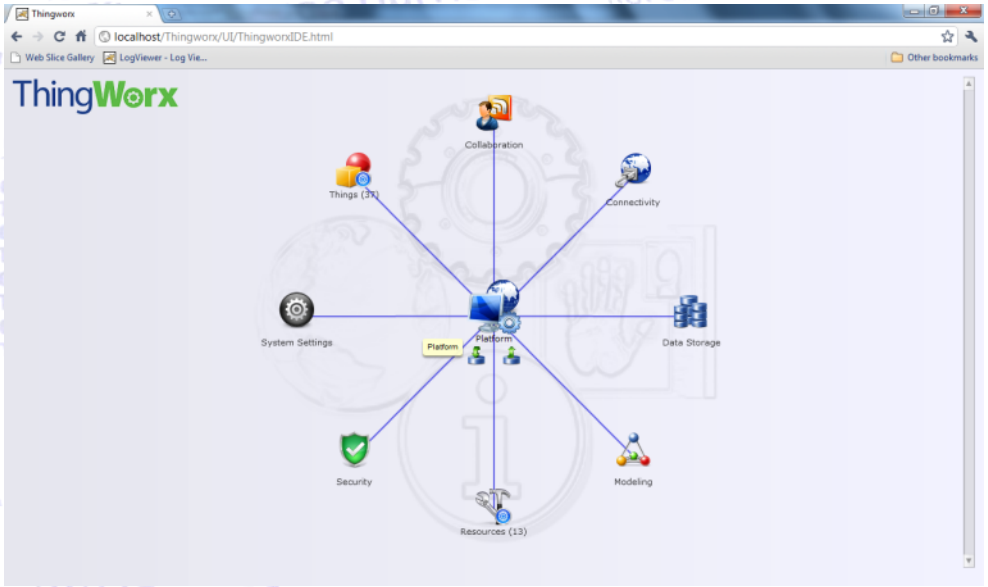
Case Study: SQUEAL

- Queries ultimately return either:
  - A set of nodes
  - A set of typed data, reflected as an “InfoTable”
- Needed to support SQL-like functions
  - Filter, sort, aggregate, union, intersection…
  - …plus location, time aggregates, interpolation
- Broad set of query criteria
  - “Is A”, “Is Near”, “Written By”, “Tagged With”, etc.
  - Full-text search

Implementation: SQUEAL

- Metadata from “shapes” (data shapes, thing shapes) defines facets
- Traversals & Lucene full-text searches for node selection
- Optimized dataset reduction where possible
- Set functions for nodes, in-memory SQL-ish functions for data
- “InfoTable” construct is in-memory “data” structure: data shape + rows w/values

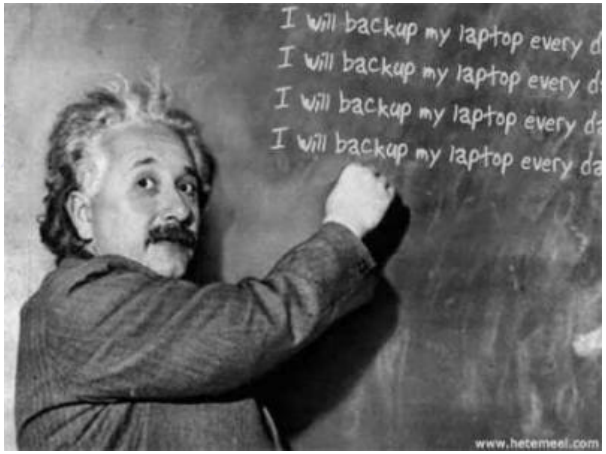
Brief Demo



amazon web services™

Lessons Learned

- Batch writes where possible for maximum insert performance
- Refactoring data models can be a bit trickier than with other DB's
- It is a good "best practice" to stamp each node/vertex with a "type" property
- Use different approaches when indexing for ordering versus indexing for searching
- Graph databases rock



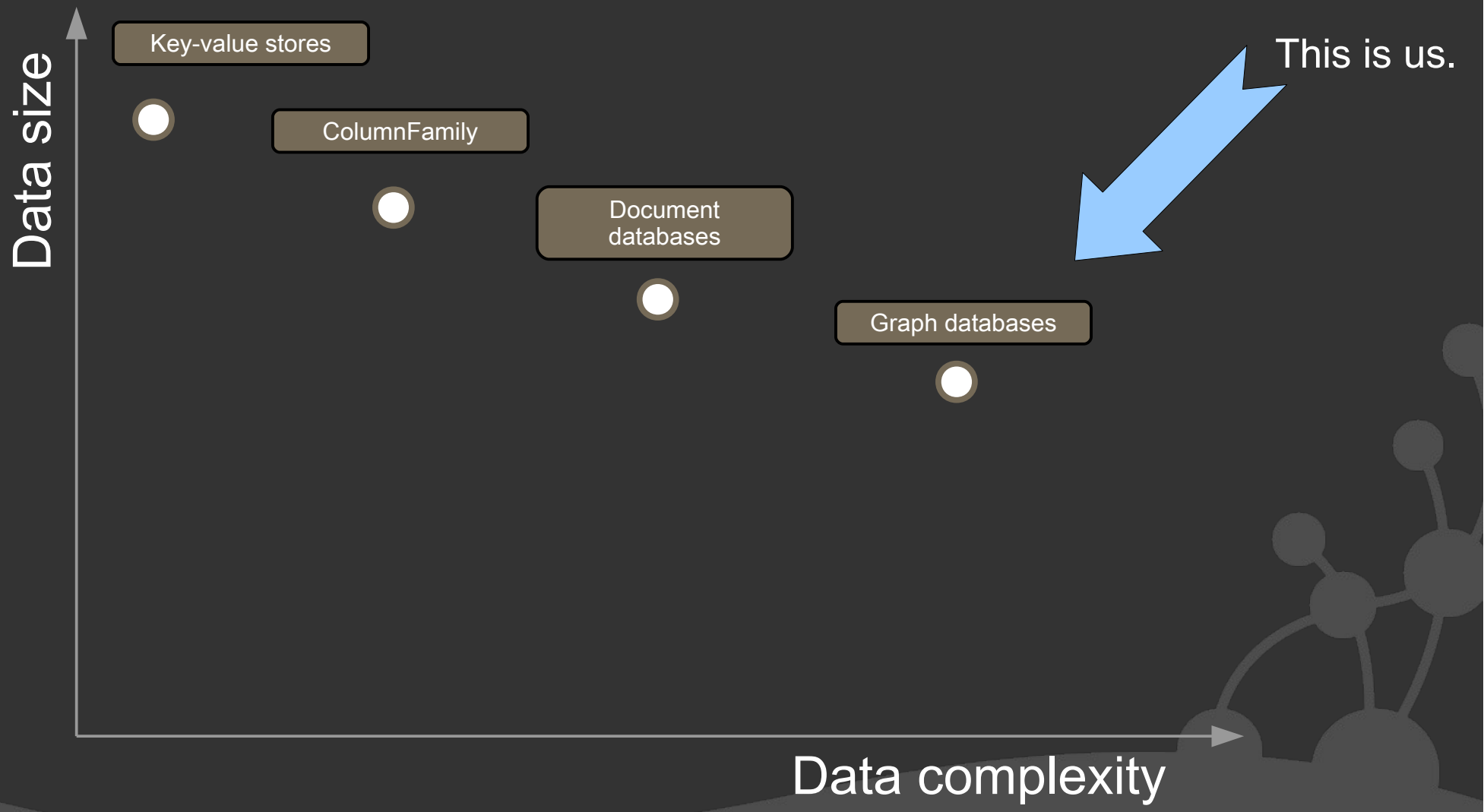
Questions?

“Far better is it to dare mighty things, to win glorious triumphs, even though checked by failure...than to rank with those poor spirits who neither enjoy much nor suffer much, because they live in a gray twilight that knows not victory nor defeat.”

- Theodore Roosevelt


neotechnology
value in relationships

(graphdb) – BELONGS_TO → (nosql)



No SQL?

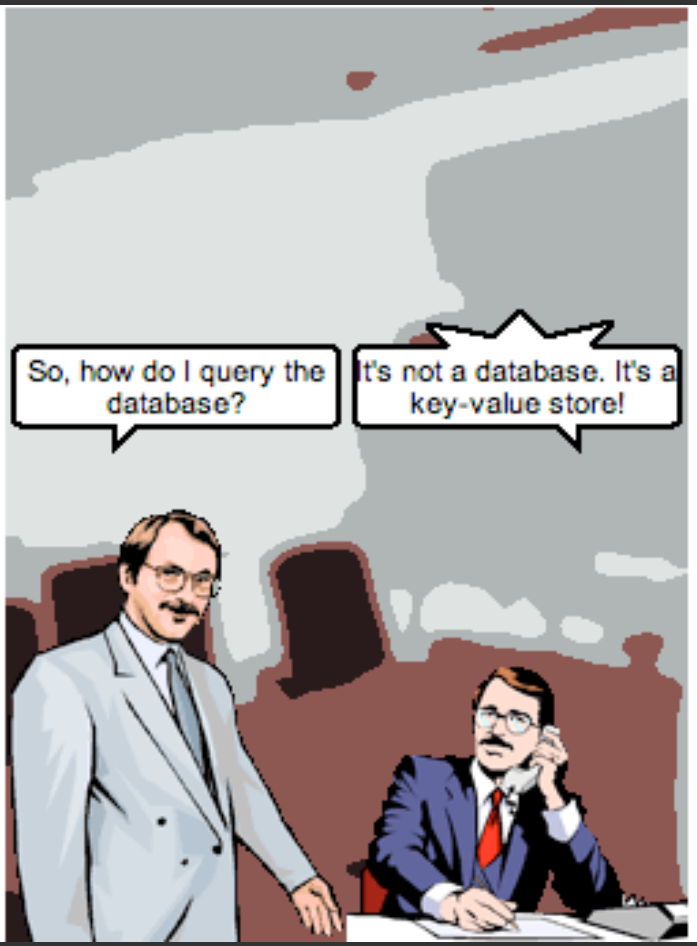


Image credit: http://browsertoolkit.com/fault-tolerance.png

No SQL?

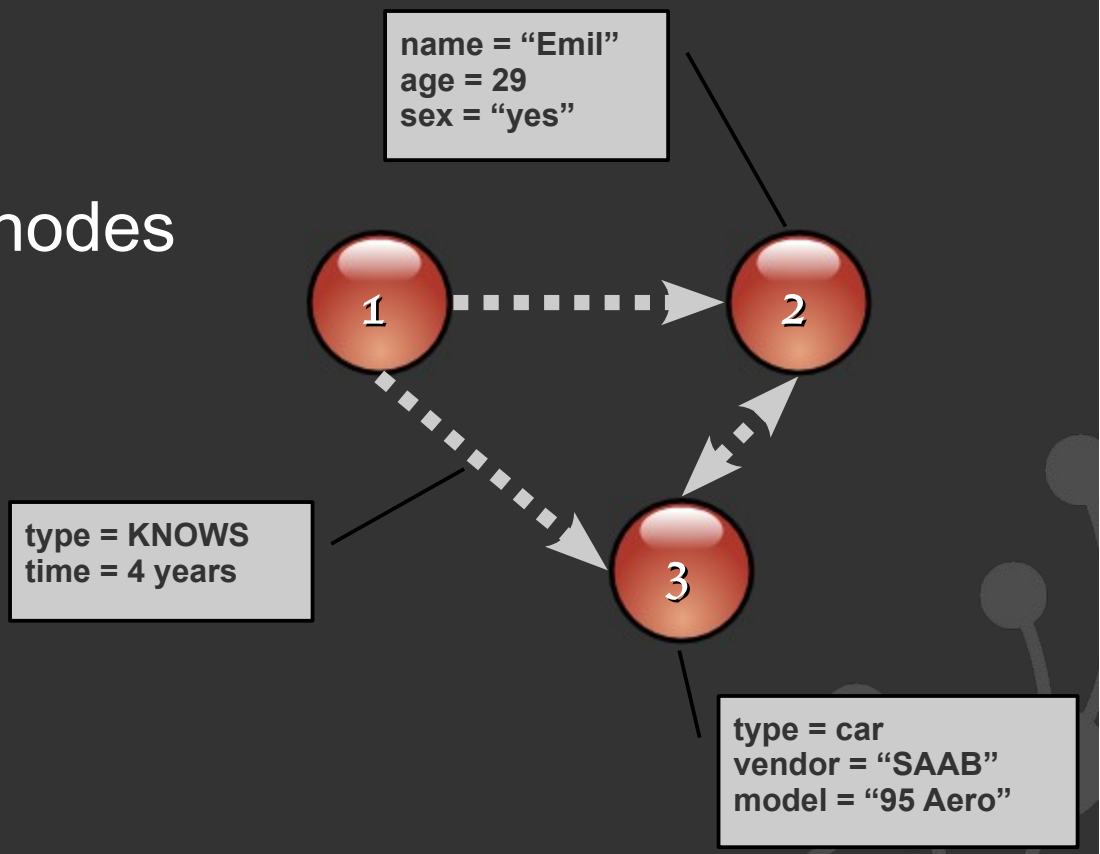
Image credit: http://browsertoolkit.com/fault-tolerance.png

No SQL?

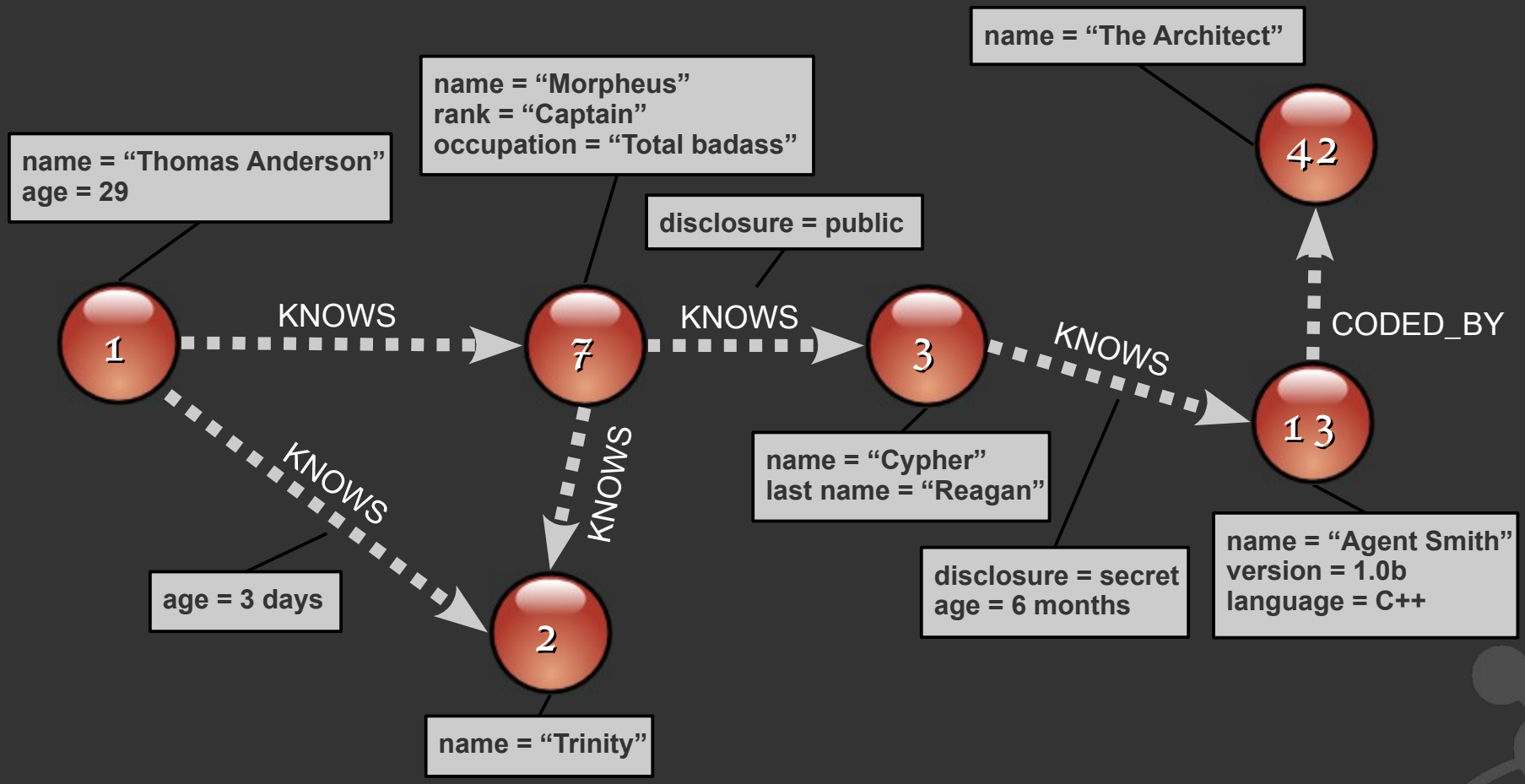
Image credit: http://browsertoolkit.com/fault-tolerance.png

The Graph DB model: representation

- Core abstractions:
  - Nodes
  - Relationships between nodes
  - Properties on both



Example: The Matrix



Code (1): Building a node space

```
GraphDatabaseService graphDb = ... // Get factory


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

// Create Morpheus
Node morpheus = graphDb.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

```
GraphDatabaseService graphDb = ... // Get factory
Transaction tx = graphDb.beginTx();

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

// Create Morpheus
Node morpheus = graphDb.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();
```

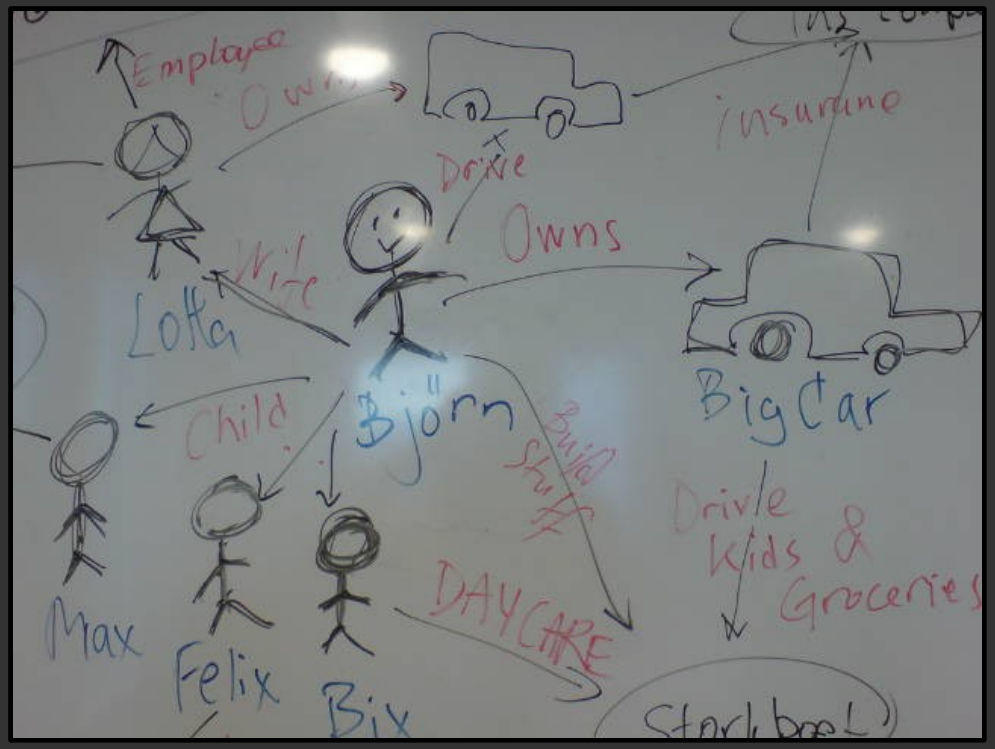
Code (1b): Defining RelationshipTypes

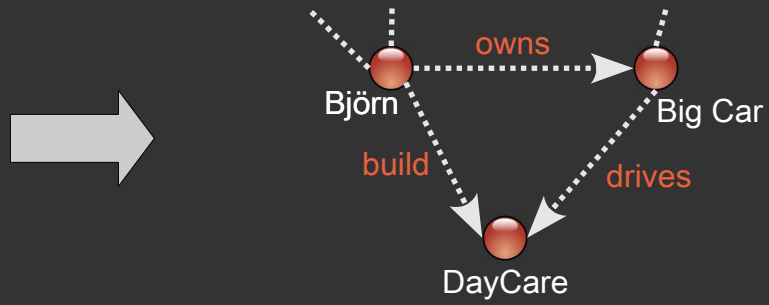
```
// In package org.neo4j.graphdb
public interface RelationshipType
{
    String name();
}

// 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,
}
```

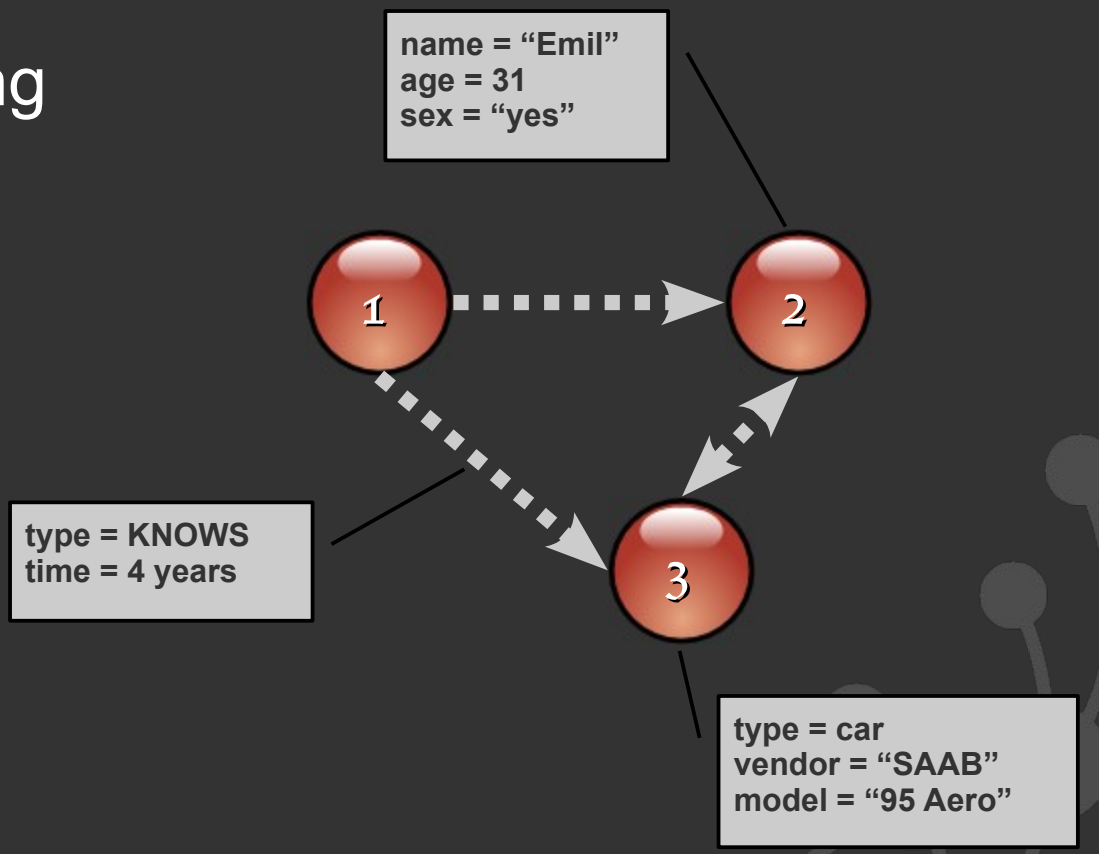
Whiteboard friendly



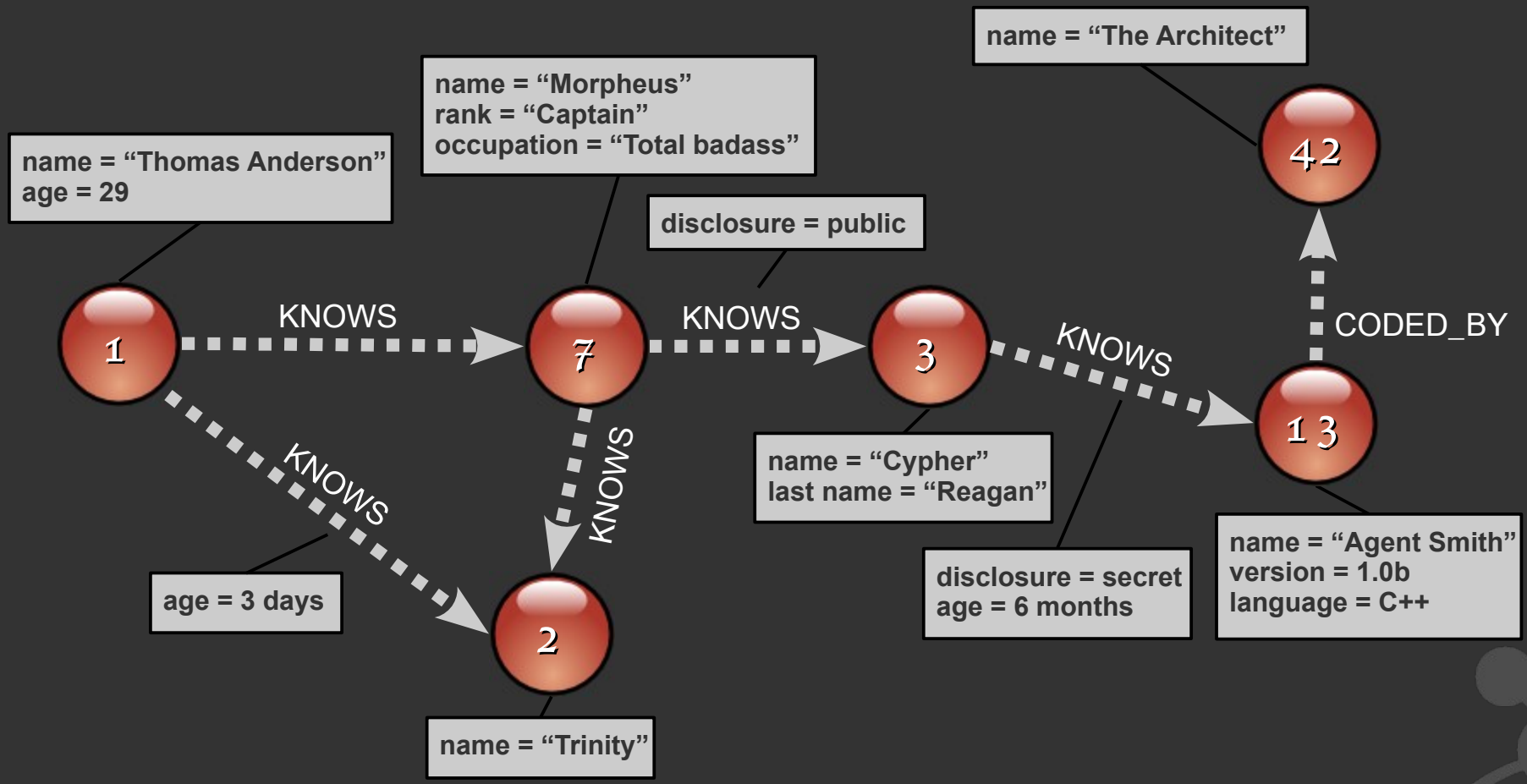


The Graph DB model: traversal

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



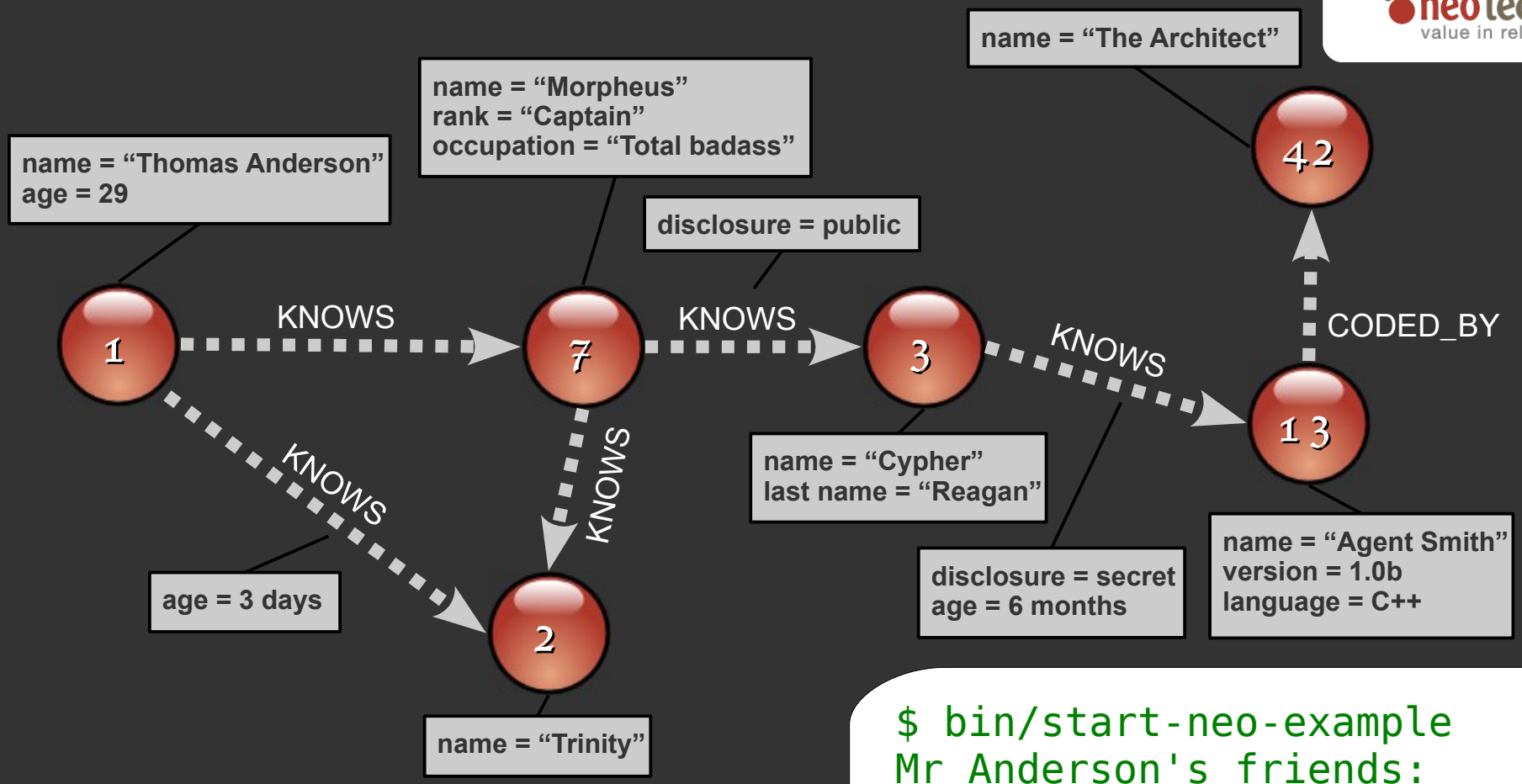
Example: Mr Anderson's friends



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 );
```

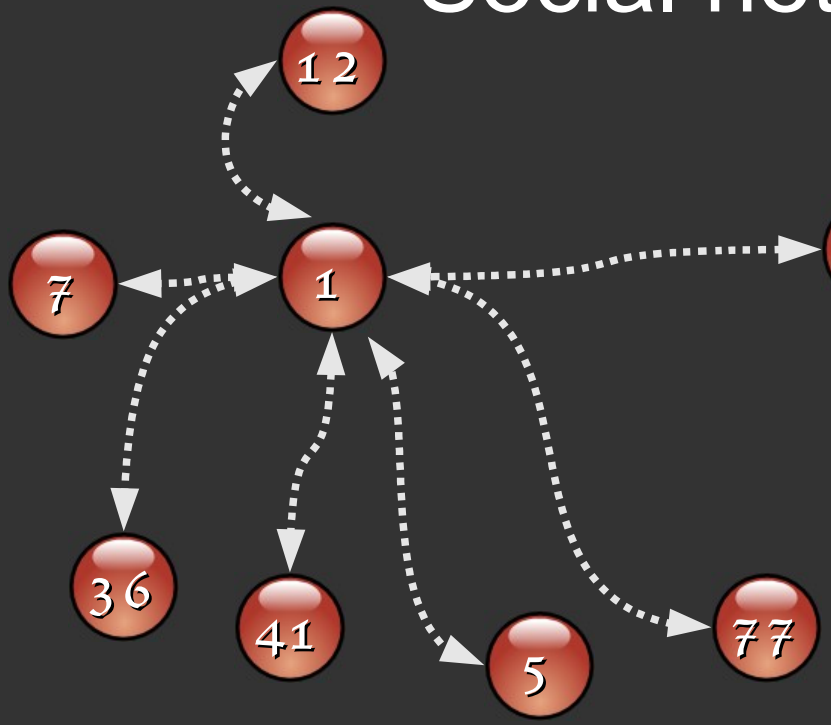
```
$ bin/start-neo-example
Mr Anderson's friends:

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

Neo4j system characteristics

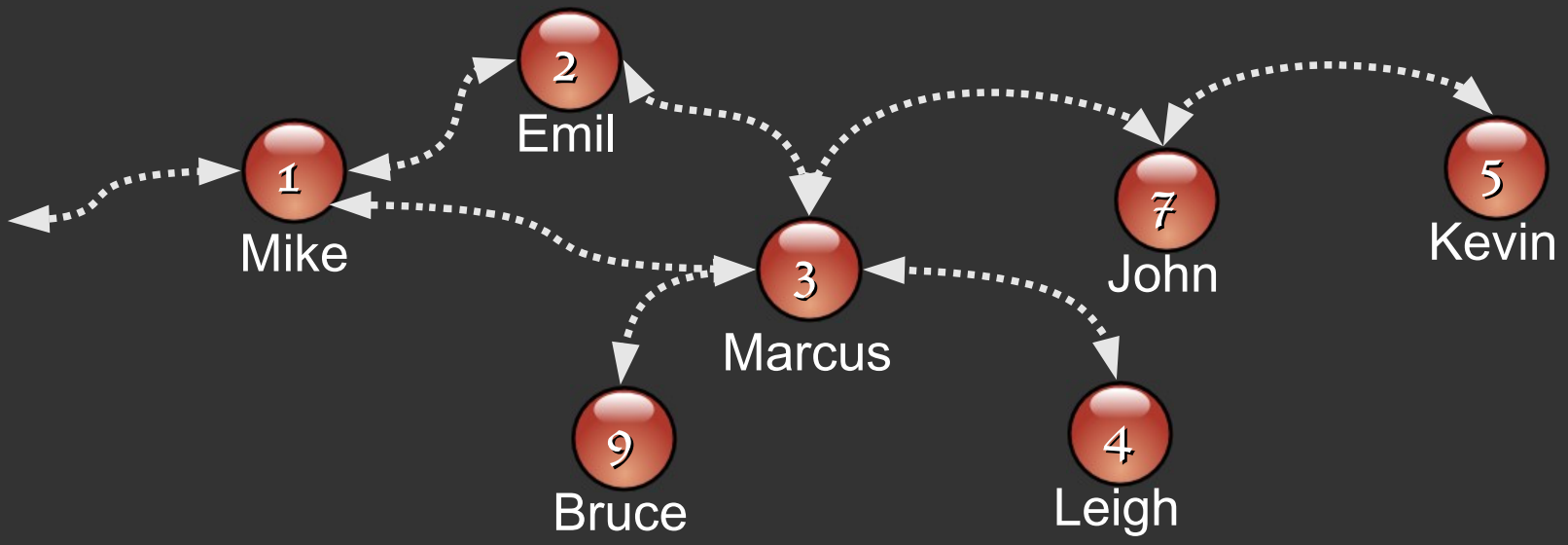
- Disk-based
  - Native graph storage engine with custom binary on-disk format
- Transactional
  - JTA/JTS, XA, 2PC, Tx recovery, deadlock detection, MVCC, etc
- Scales up
  - Many billions of nodes/rels/props on single JVM
- Robust
  - 7+ years in 24/7 production

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
Graph database (Neo4j)
Graph database (Neo4j)

# persons	*query time*


neo technology
value in relationships