NOSQL

the past, present and future

Emil Eifrem
emil@neotechnology.com
@emileifrem
#neo4j

So what's the plan?

- The title page already told you, but out of order(*). Actual order:
  - The **Present**
  - The **Past**
  - And **The Future** of NOSQL
- Then lunch.

(*) Out of order? Well, the title page did *inorder* traversal (left node, root, right node), and since most folks aren't graph geeks I'm going to go with the more intuitive *preorder* traversal (root, left, right) in the actual presentation.

Anyway...

NOSQL: The Present

First off: the name

- WE ALL HATES IT, M'KAY?

NOSQL is NOT...

- NO to SQL
- NEVER SQL

NOSQL is simply

Not **O**nly **SQL**

NOSQL - Why now?

Four trends

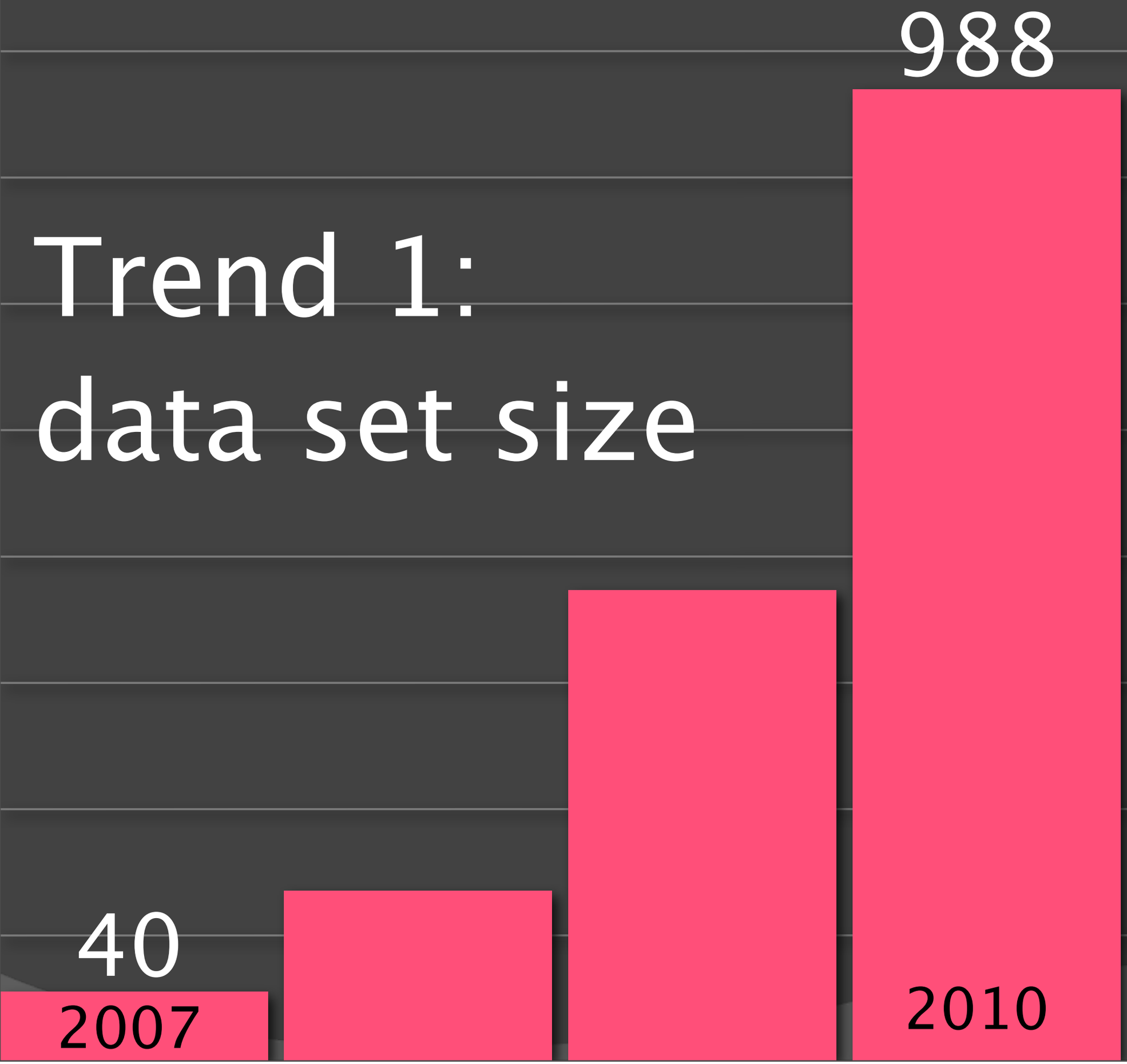
Trend 1:
data set size

40

2007

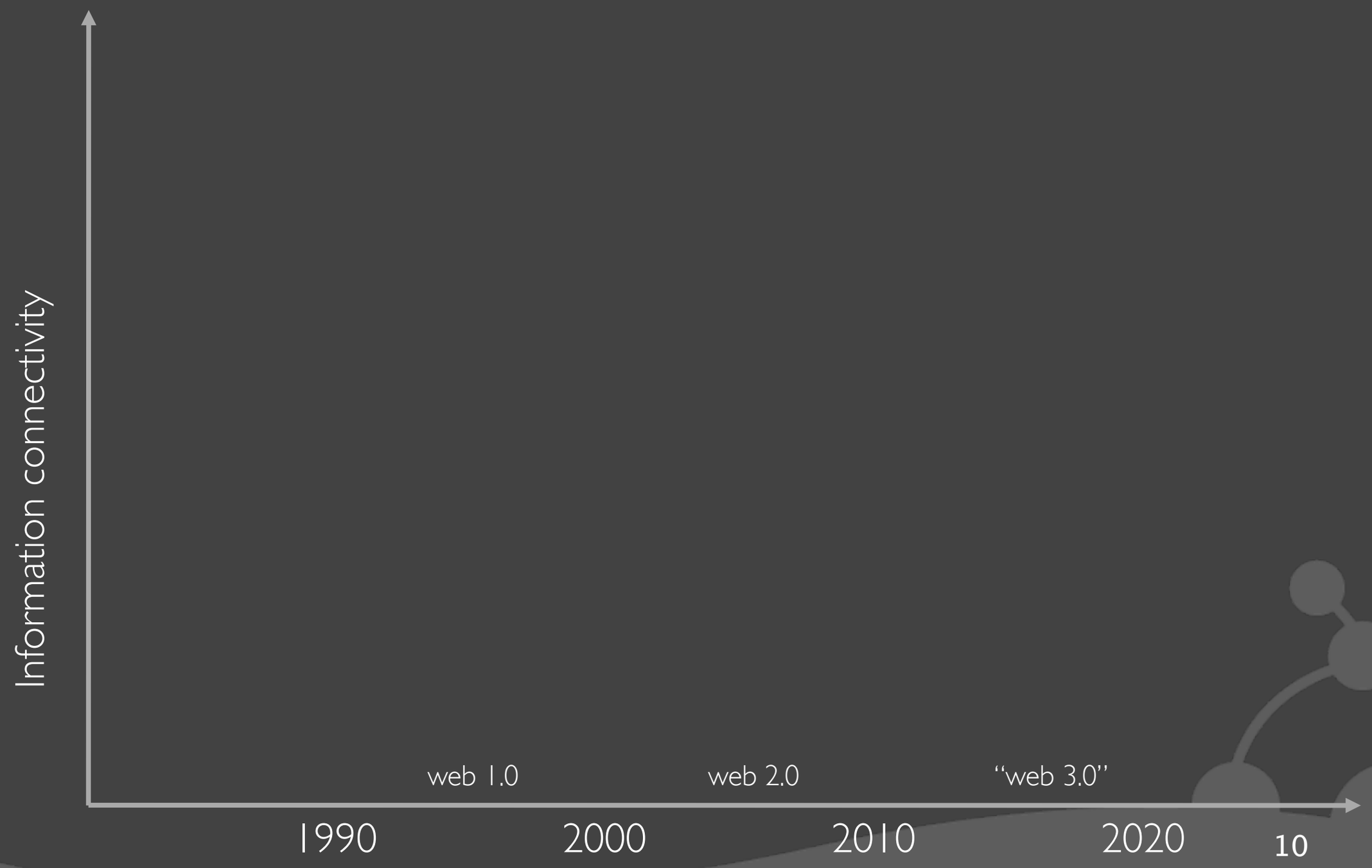
Source: IDC 2007

Trend 1: data set size

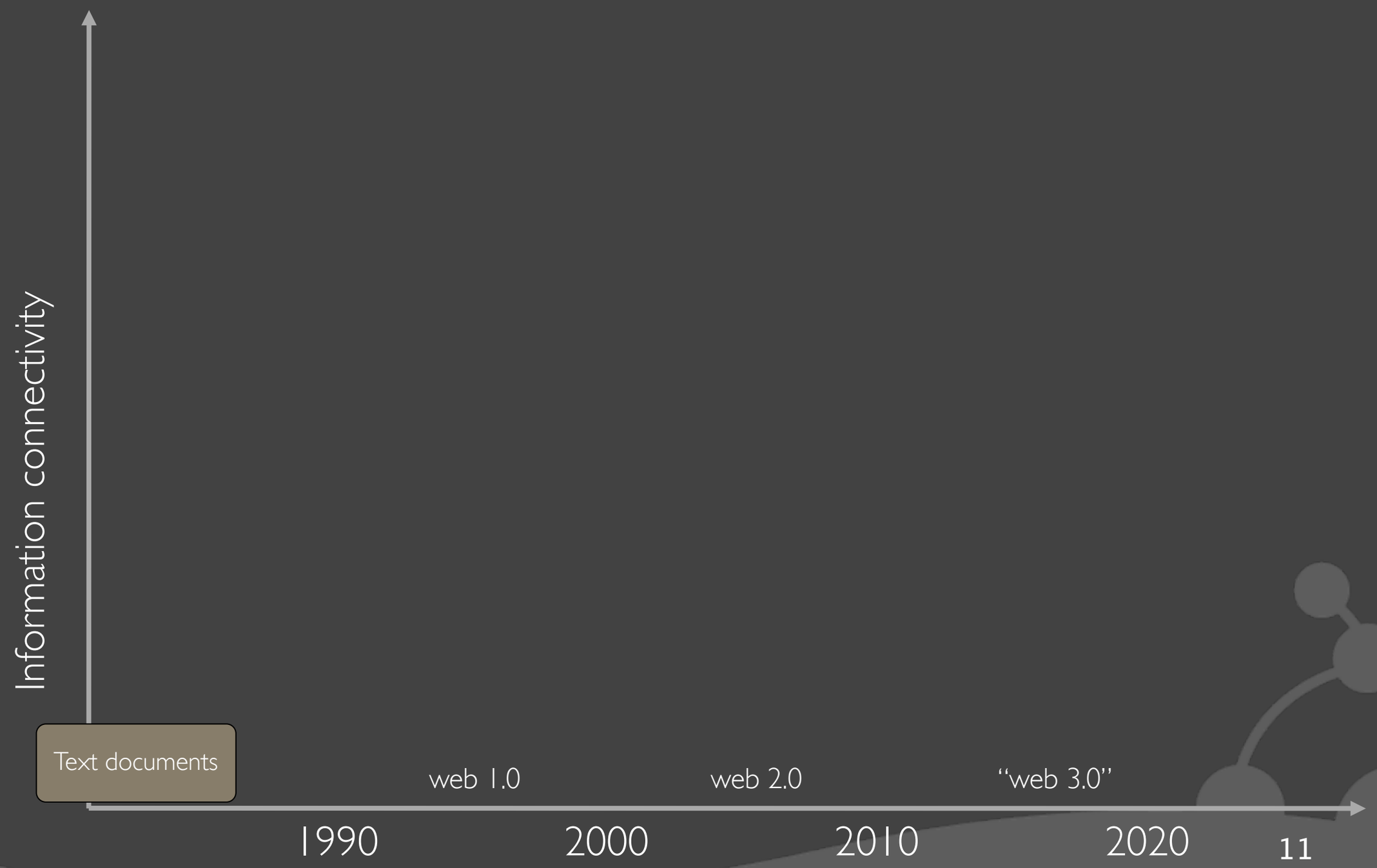


Source: IDC 2007

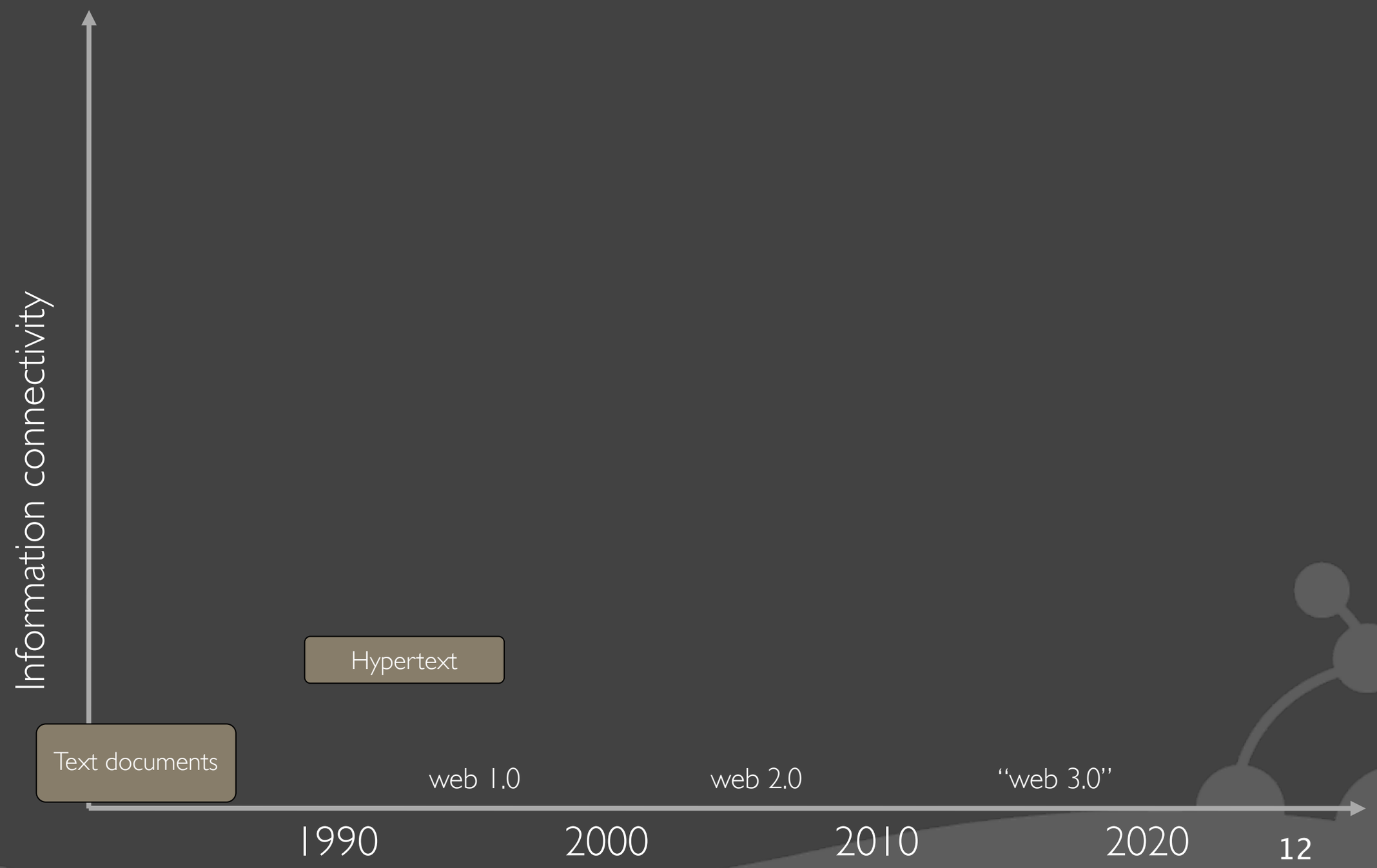
Trend 2: Connectedness



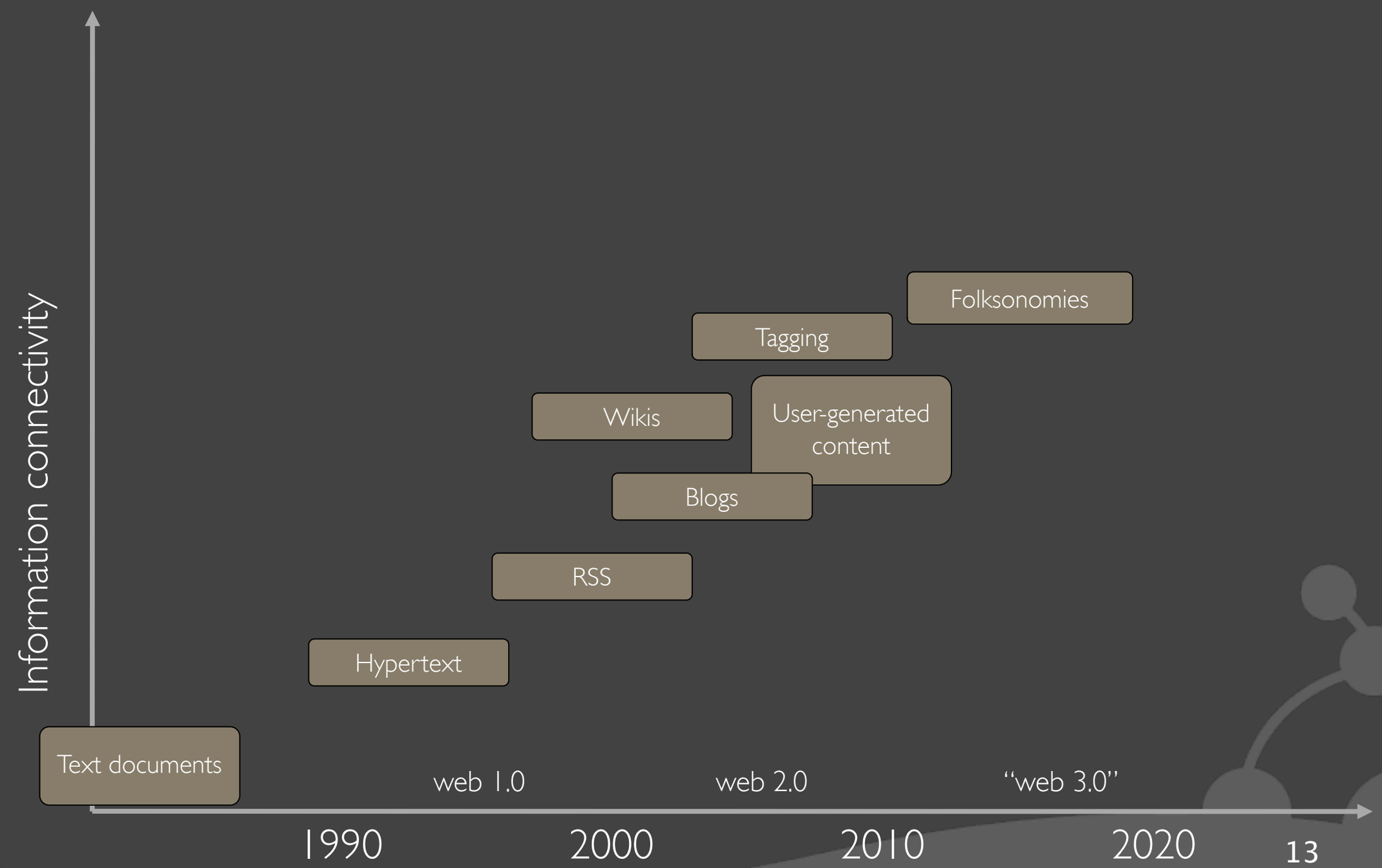
Trend 2: Connectedness



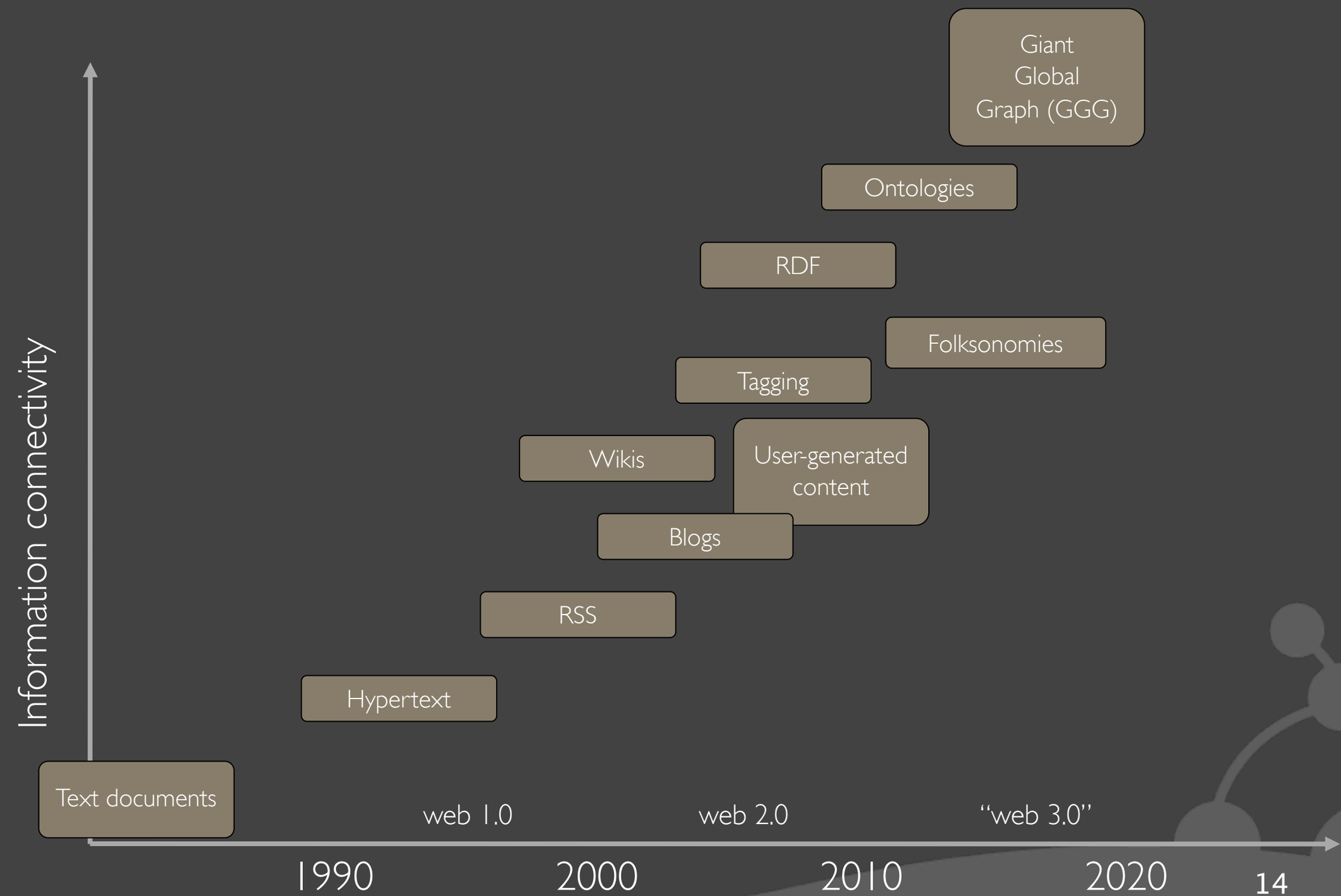
Trend 2: Connectedness



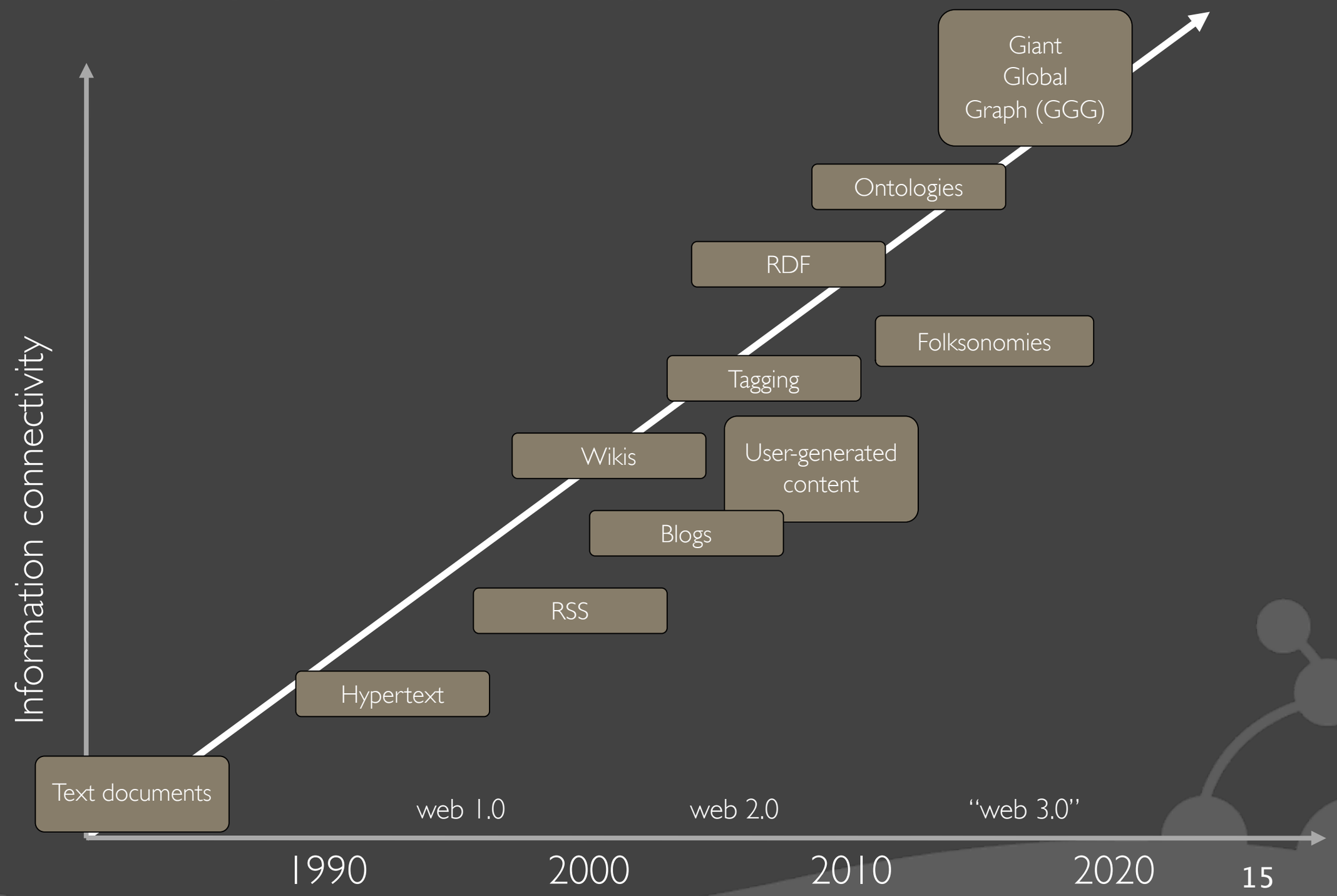
Trend 2: Connectedness



Trend 2: Connectedness



Trend 2: Connectedness

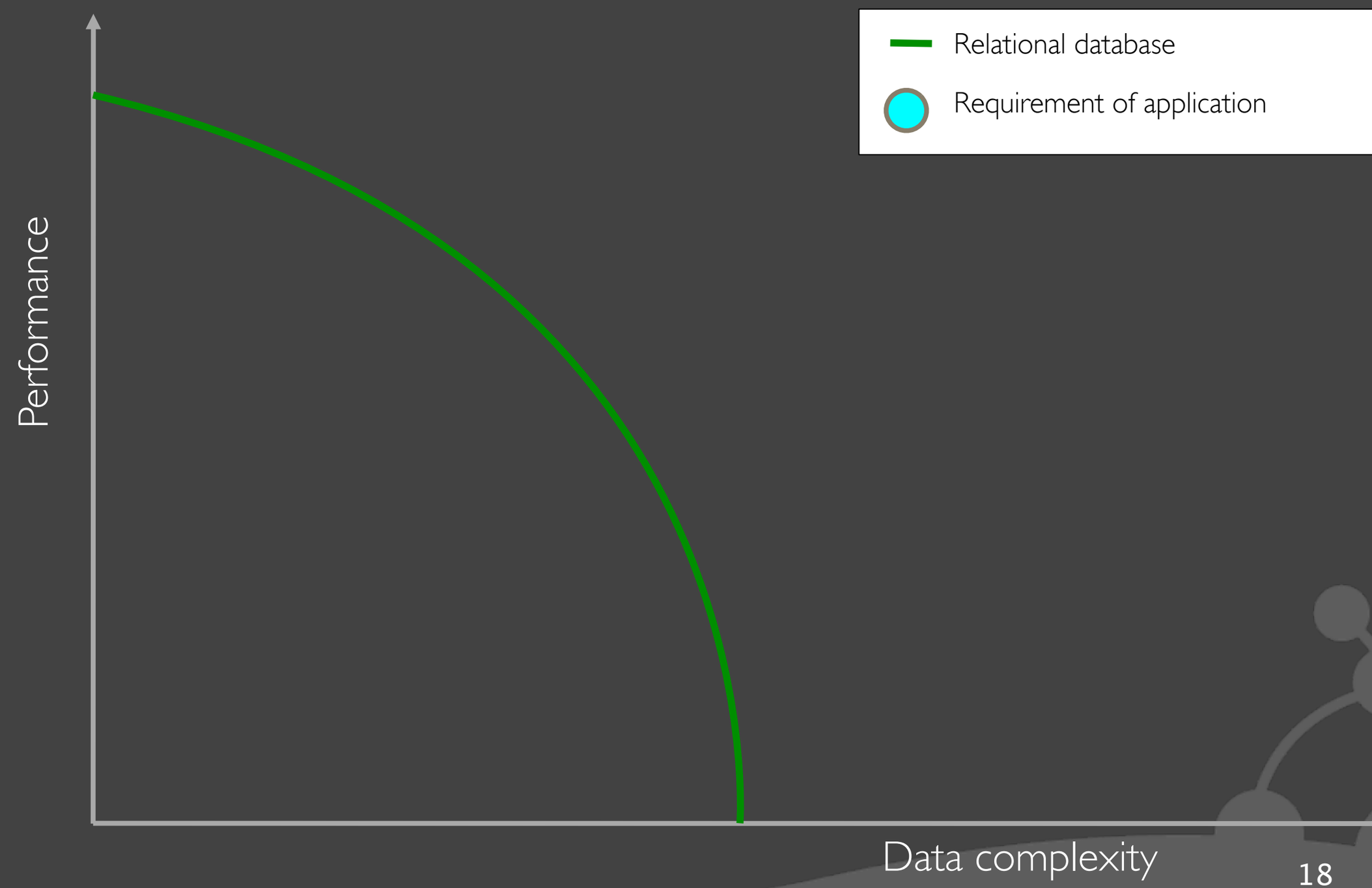


Trend 3: Semi-structure

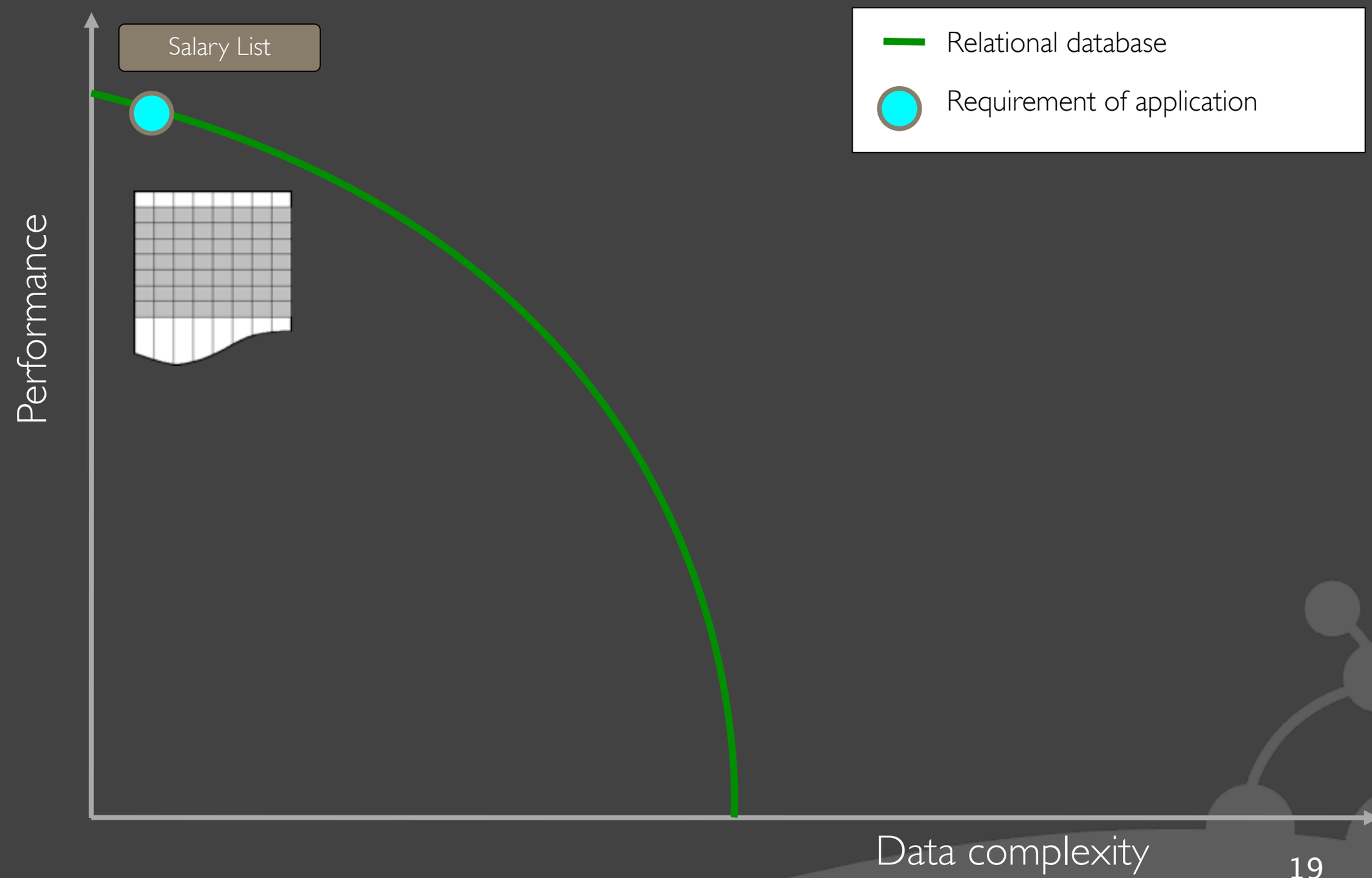
- Individualization of content
  - In the salary lists of the 1970s, all elements had exactly one job
  - In the salary lists of the 2000s, we need 5 job columns! Or 8? Or 15?
- All encompassing "entire world views"
  - Store more data about each entity
- Trend accelerated by the decentralization of content generation that is the hallmark of the age of participation ("web 2.0")

Aside: RDBMS performance

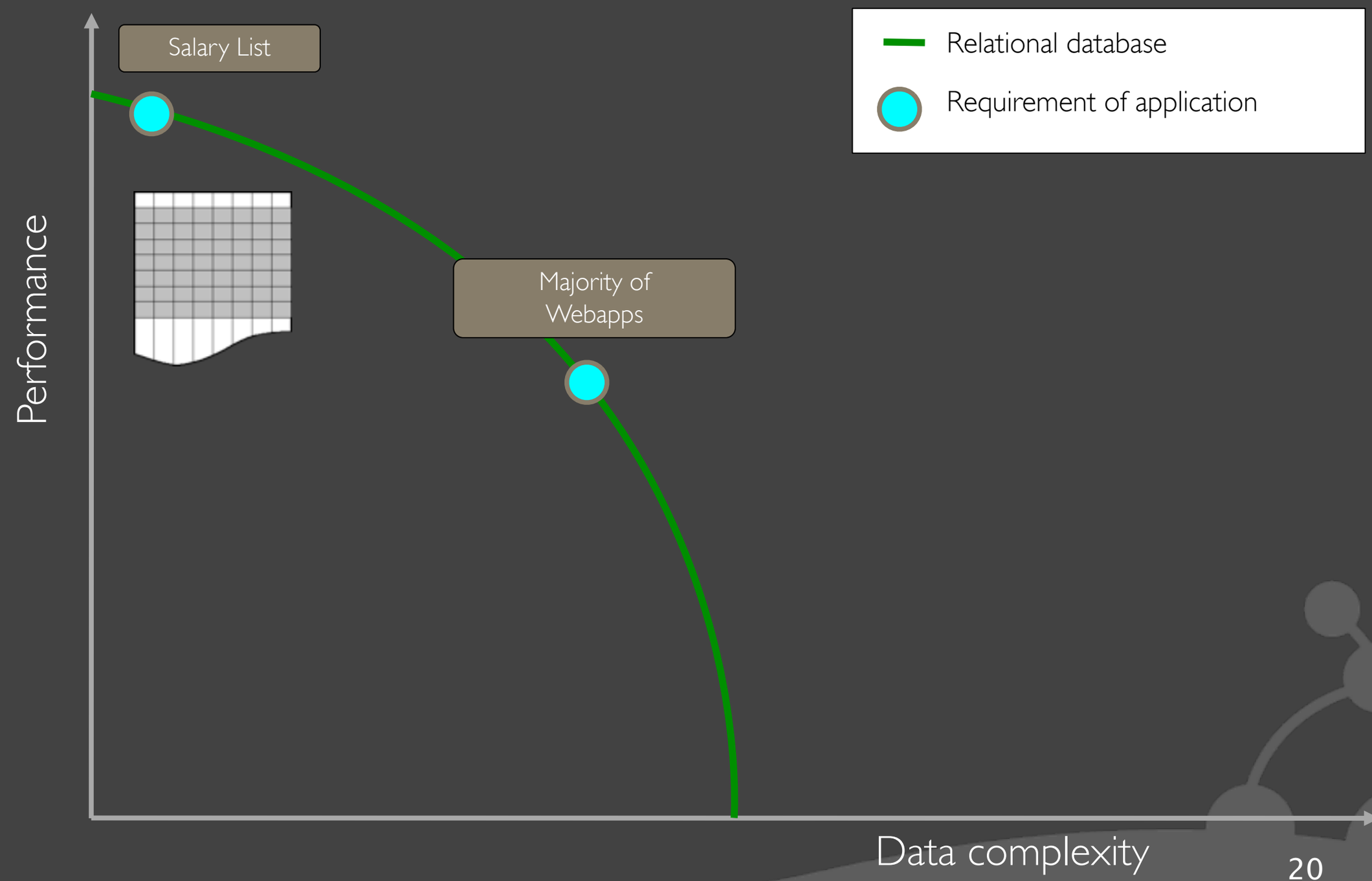
Aside: RDBMS performance



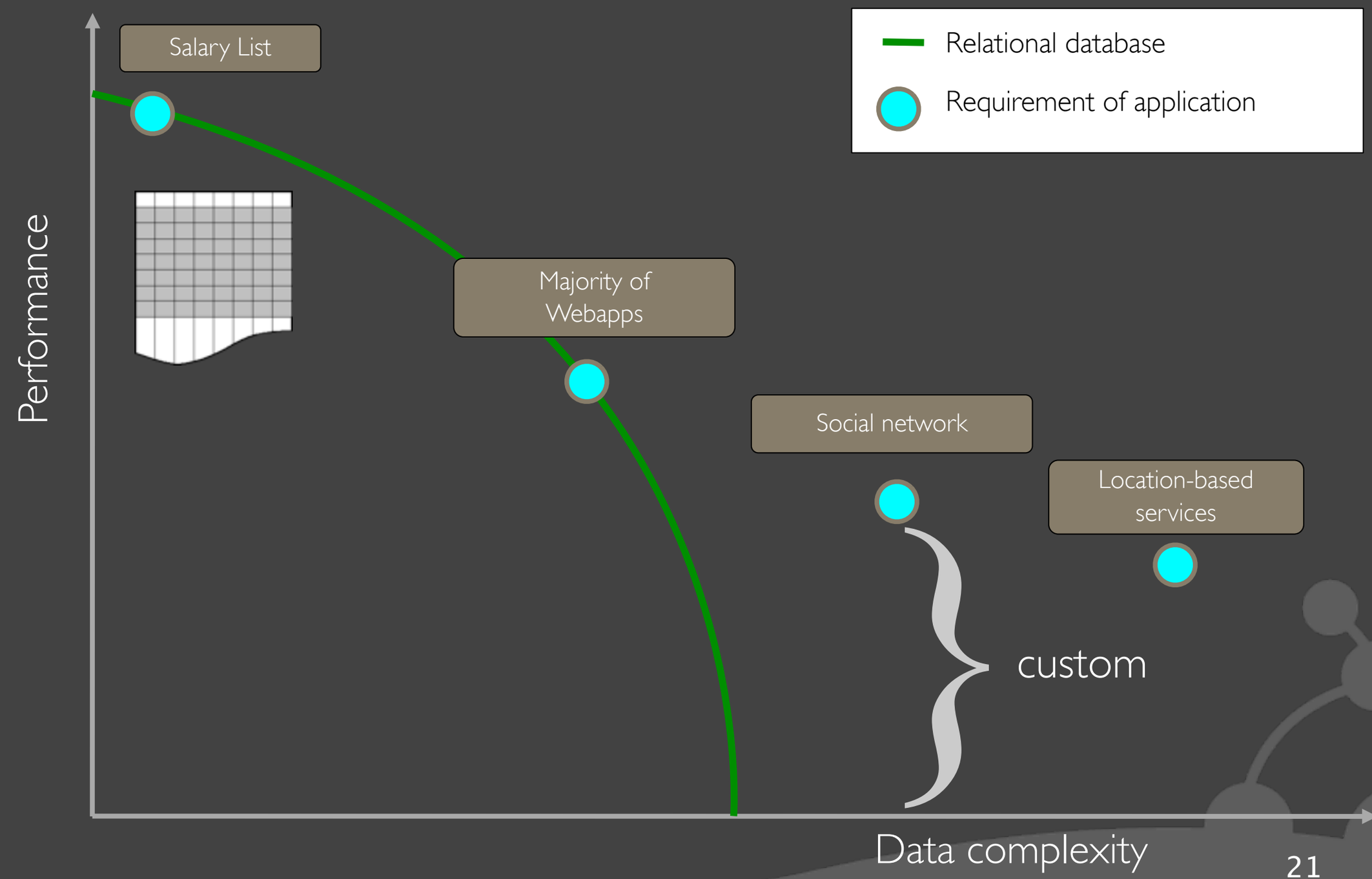
Aside: RDBMS performance



Aside: RDBMS performance

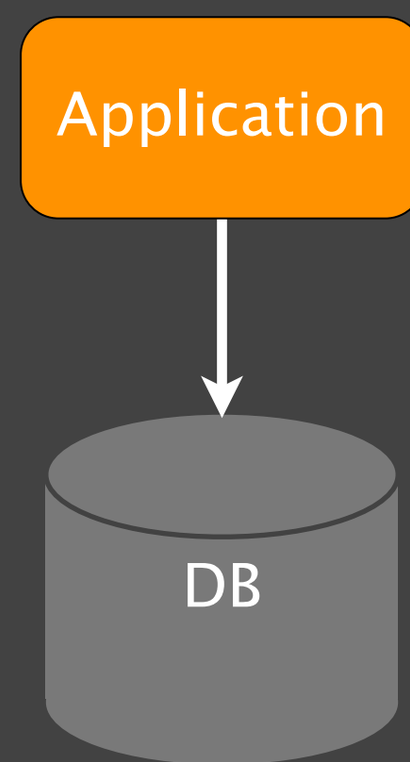


Aside: RDBMS performance



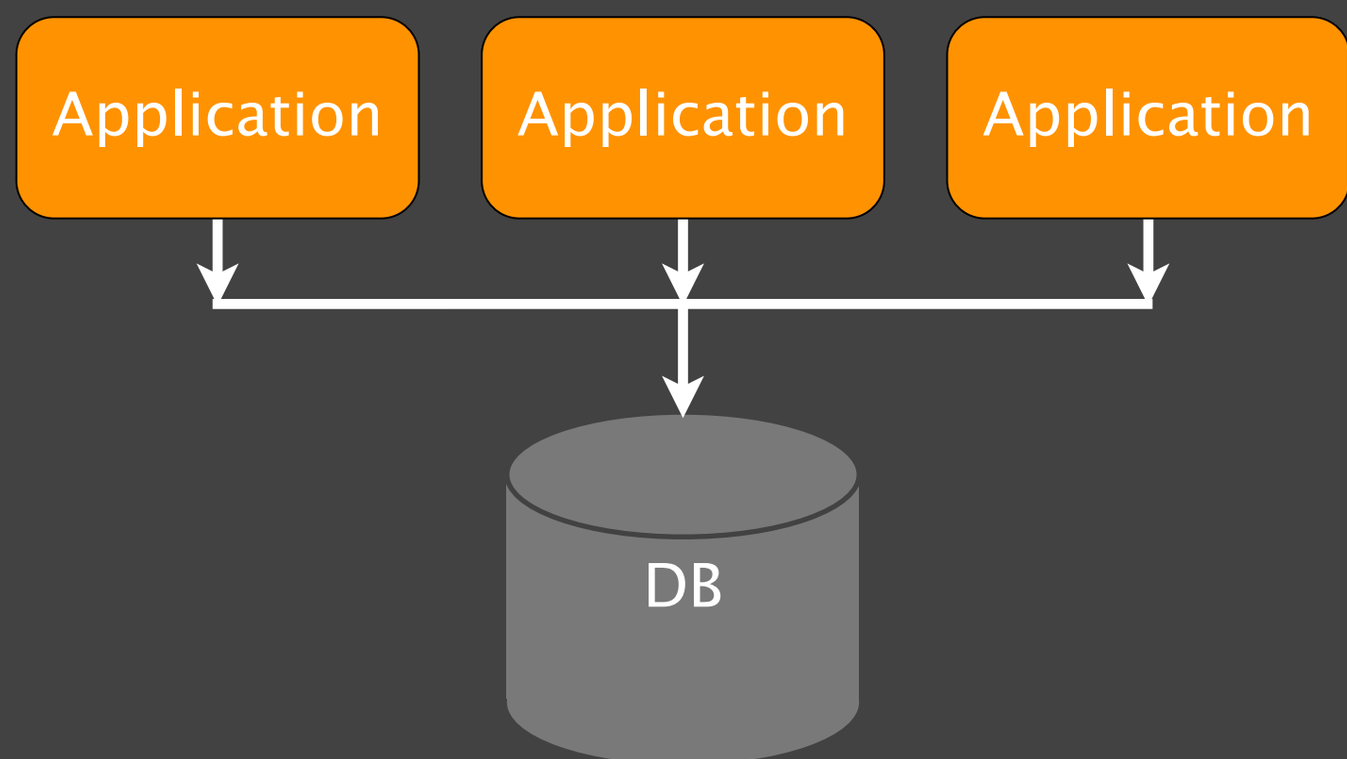
Trend 4: Architecture

1980s: Application (<-- note lack of s)



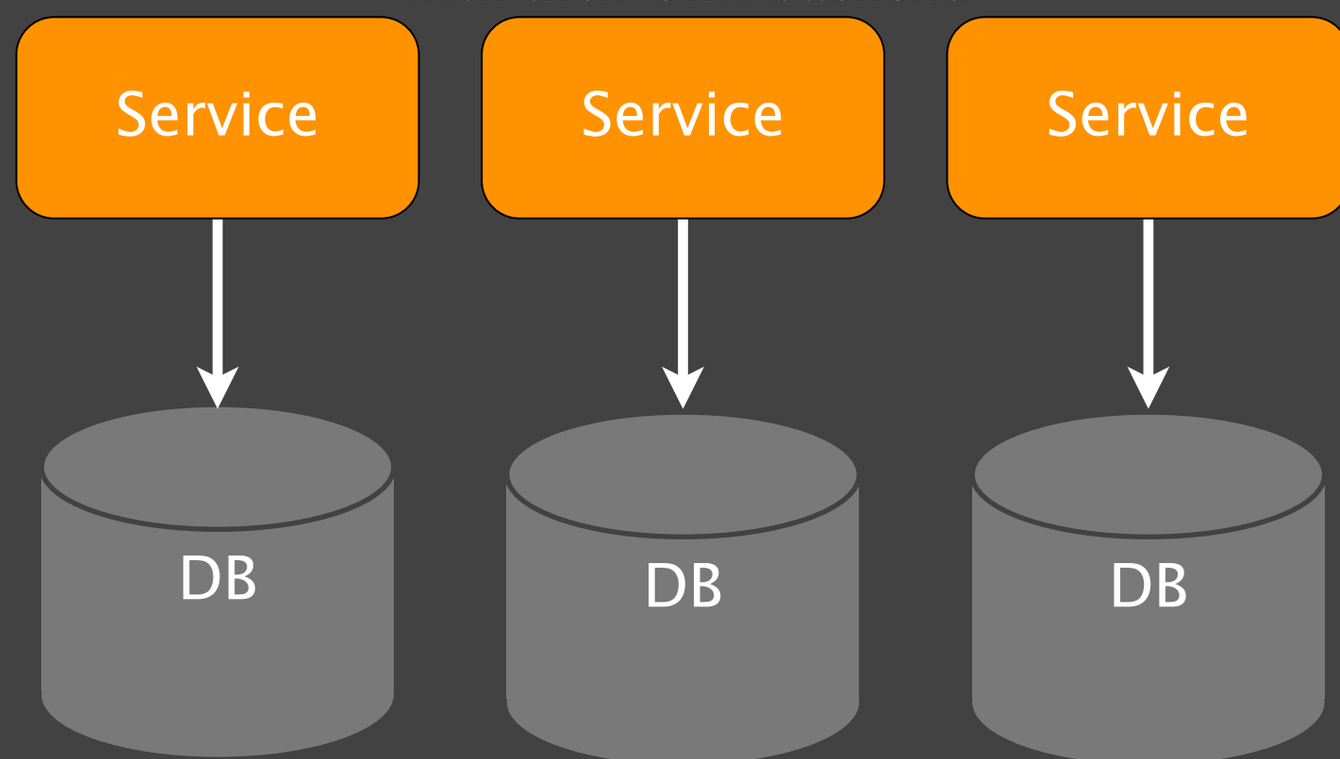
Trend 4: Architecture

1990s: Database as integration hub



Trend 4: Architecture

2000s: (moving towards) Decoupled services
with their own backend



Why NOSQL Now?

- Trend 1: Size
- Trend 2: Connectedness
- Trend 3: Semi-structure
- Trend 4: Architecture

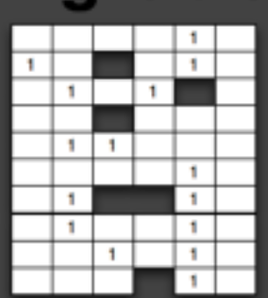
Key-Value

Graph DB

NOSQL

Four product categories

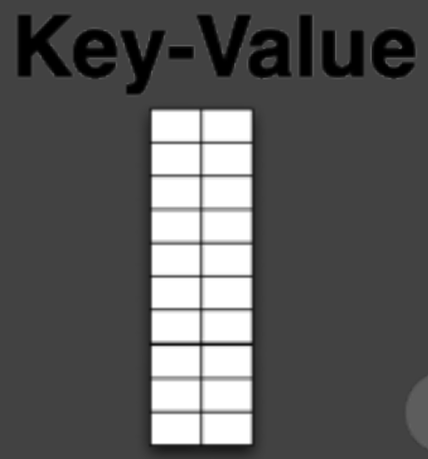
BigTable



Document

Category 1: Key-Value stores

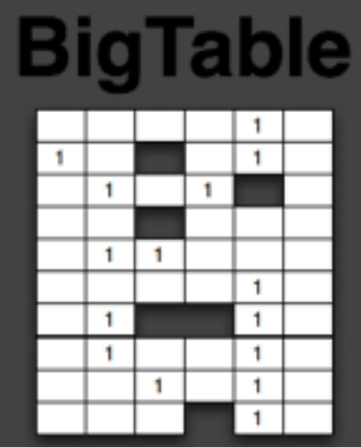
- Lineage:
  - "Dynamo: Amazon's Highly Available Key-Value Store" (2007)
- Data model:
  - Global key-value mapping
  - Think: Globally available HashMap/Dict/etc



- Examples:
  - Project Voldemort
  - Riak
  - ORACLE

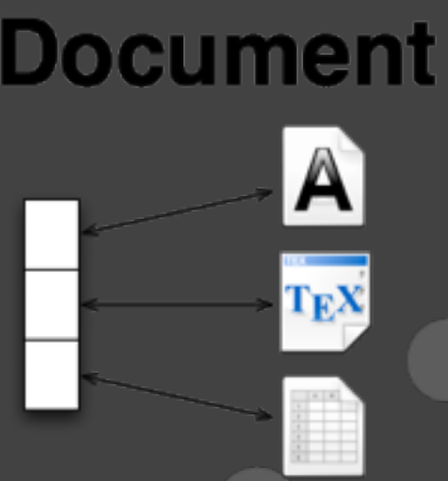
Category II: ColumnFamily (BigTable) stores

- Lineage:
  - "Bigtable: A Distributed Storage System for Structured Data" (2006)
- Data model:
  - A big table, with column families
- Examples:
  - HBase
  - HyperTable
  - Cassandra



Category III: Document databases

- Lineage:
  - Lotus Notes
- Data model:
  - Collections of documents
  - A document is a key-value collection
- Examples:
  - CouchDB
  - MongoDB



Document db: An example

- How would we model a blogging software?

- One stab:
  - Represent each Blog as a *Collection* of Post *documents*
  - Represent Comments as *nested documents* in the Post documents

Document db: Creating a blog post

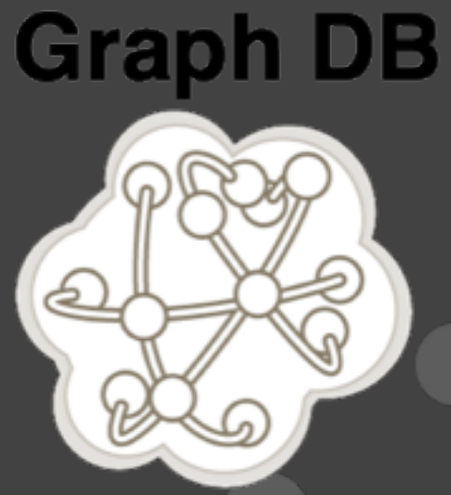
```
import com.mongodb.Mongo;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
// ...
Mongo mongo = new Mongo( "localhost" ); // Connect to MongoDB
// ...
DB blogs = mongo.getDB( "blogs" ); // Access the blogs database
DBCollection myBlog = blogs.getCollection( "Thobe's blog" );

DBObject blogPost = new BasicDBObject();
blogPost.put( "title", "ApacheCon 2011" );
blogPost.put( "pub_date", new Date() );
blogPost.put( "body", "Publishing a post about ApacheCon in my
   MongoDB blog!" );
blogPost.put( "tags", Arrays.asList( "conference", "names" ) );
blogPost.put( "comments", new ArrayList() );

myBlog.insert( blogPost );
```

Retrieving posts

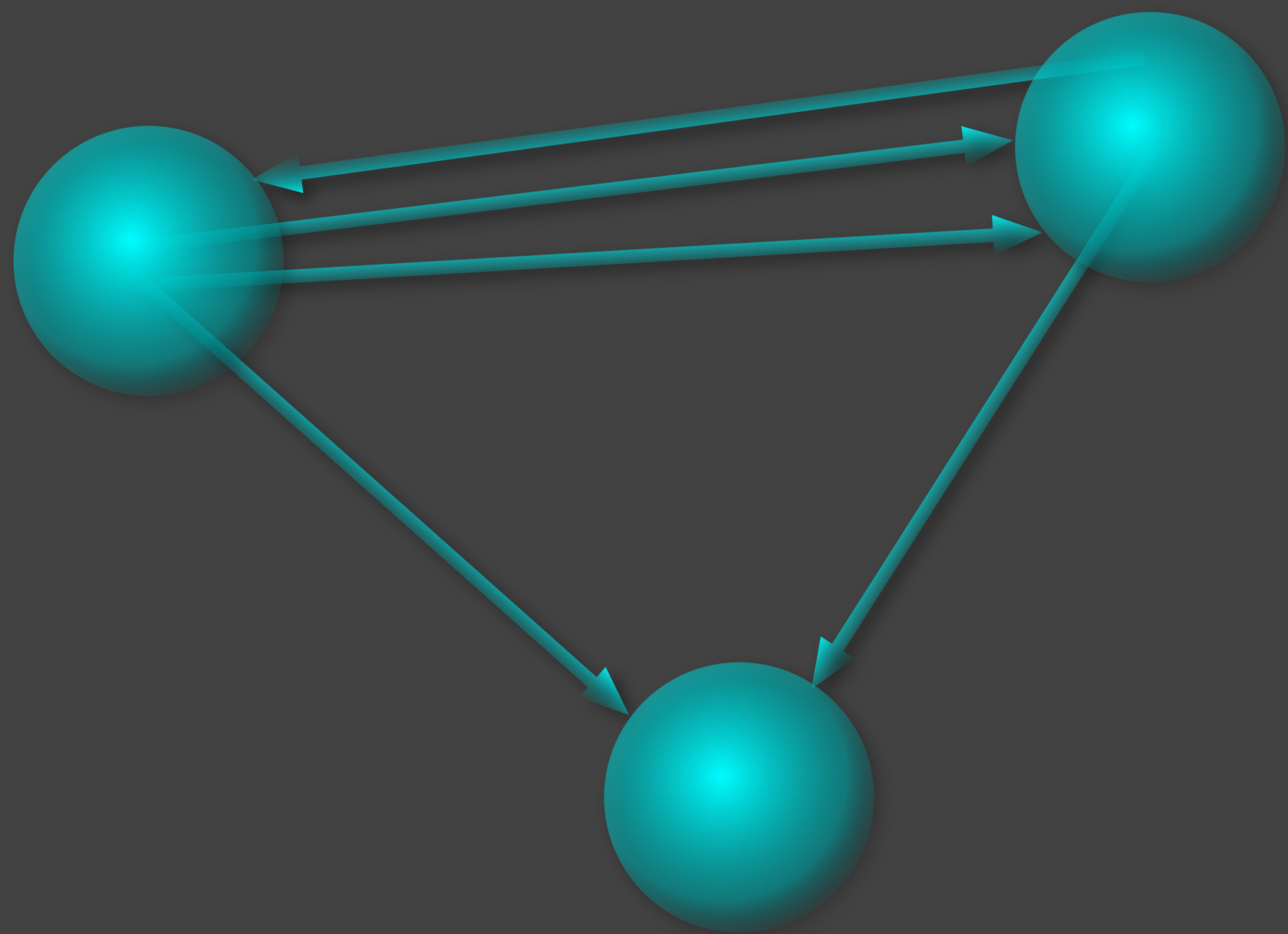
```
// ...
import com.mongodb.DBCursor;
// ...

public Object getAllPosts( String blogName ) {
    DBCollection blog = db.getCollection( blogName );
    return renderPosts( blog.find() );
}

private Object renderPosts( DBCursor cursor ) {
    // order by publication date (descending)
    cursor = cursor.sort( new BasicDBObject( "pub_date", -1 ) );
    // ...
}
```

Category IV: Graph databases

- Lineage:
  - Euler and graph theory
- Data model:
  - Nodes with properties
  - Typed relationships with properties
- Examples:
  - InfiniteGraph
  - Neo4j

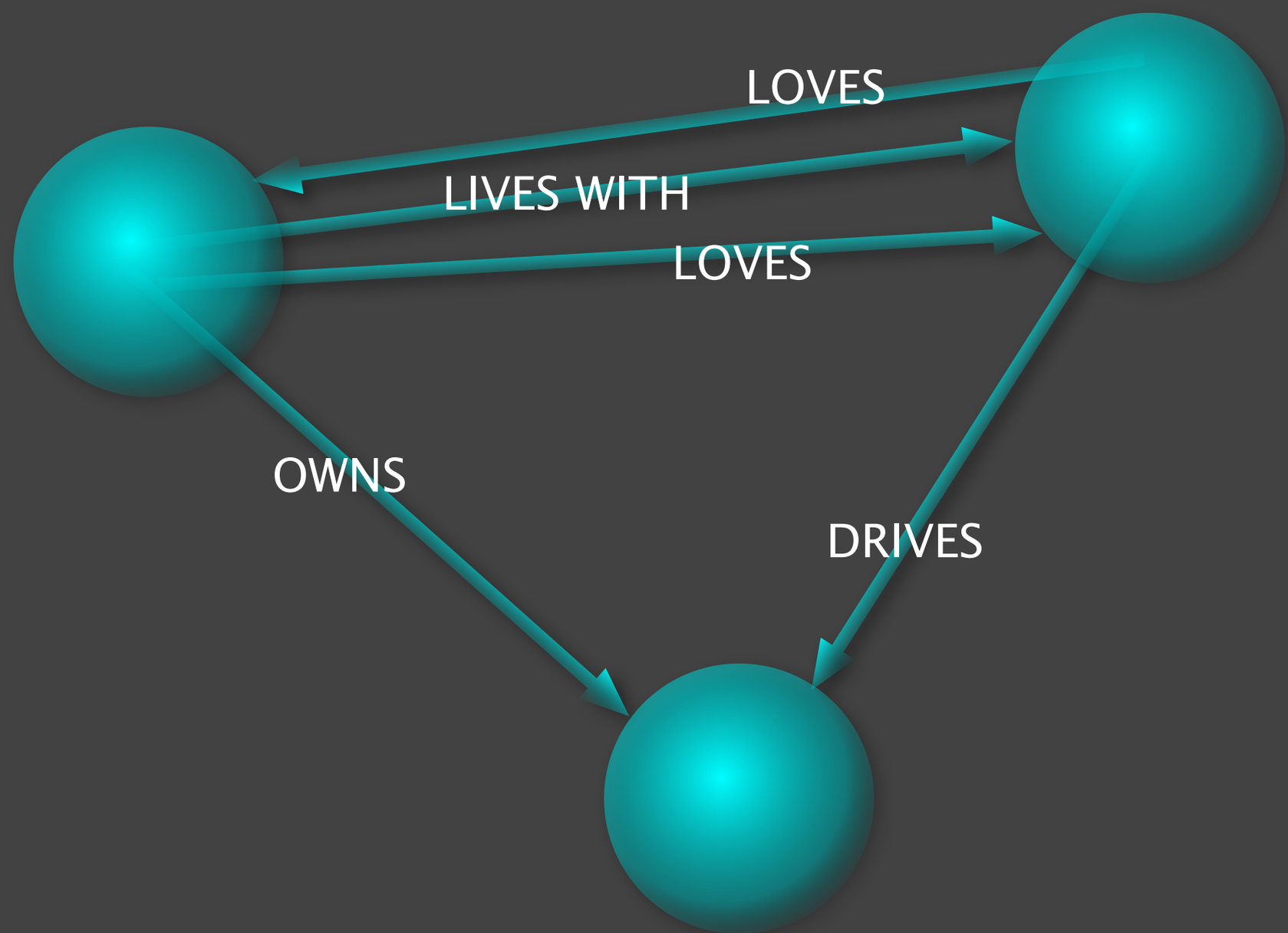


Property Graph model



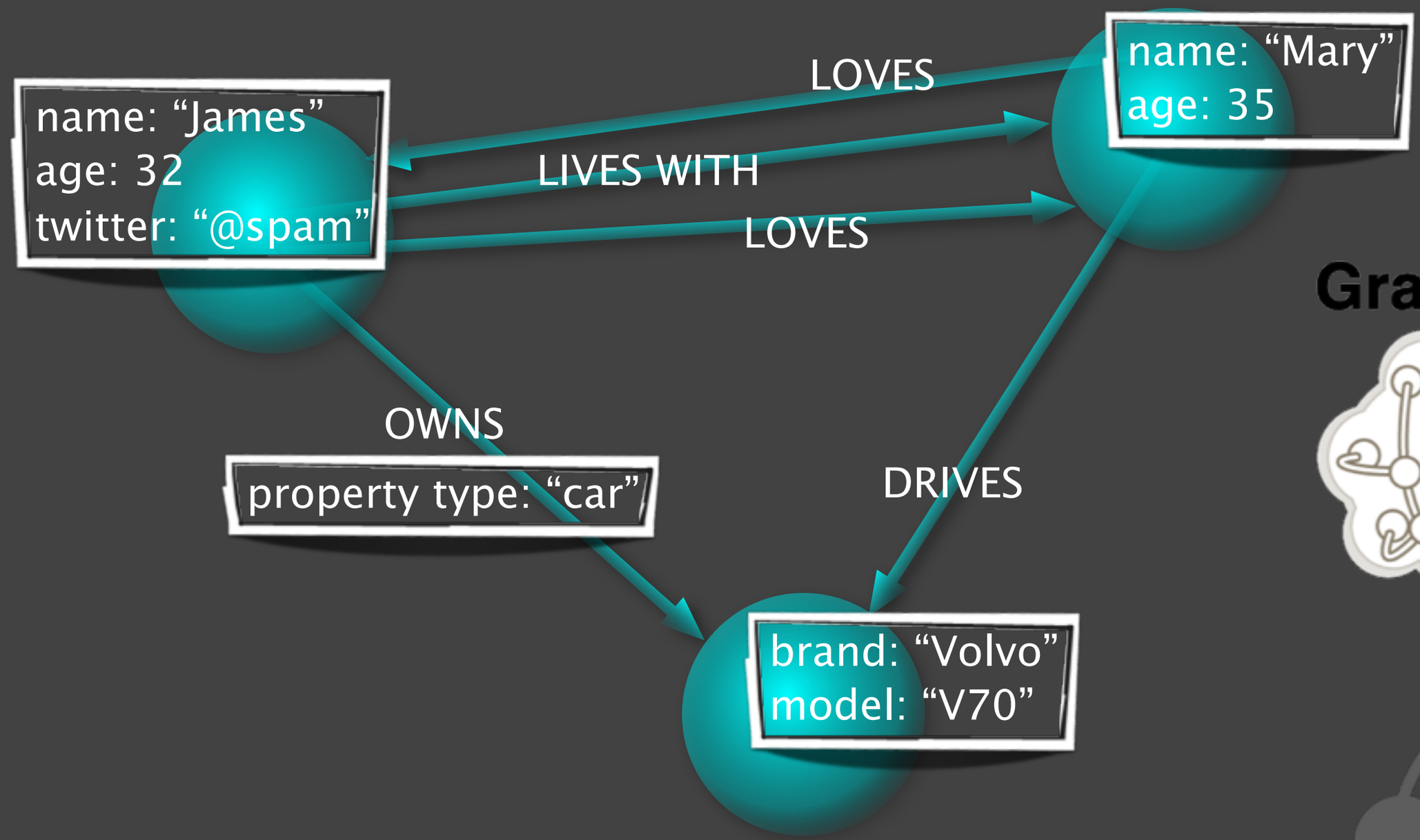
Graph DB

Property Graph model



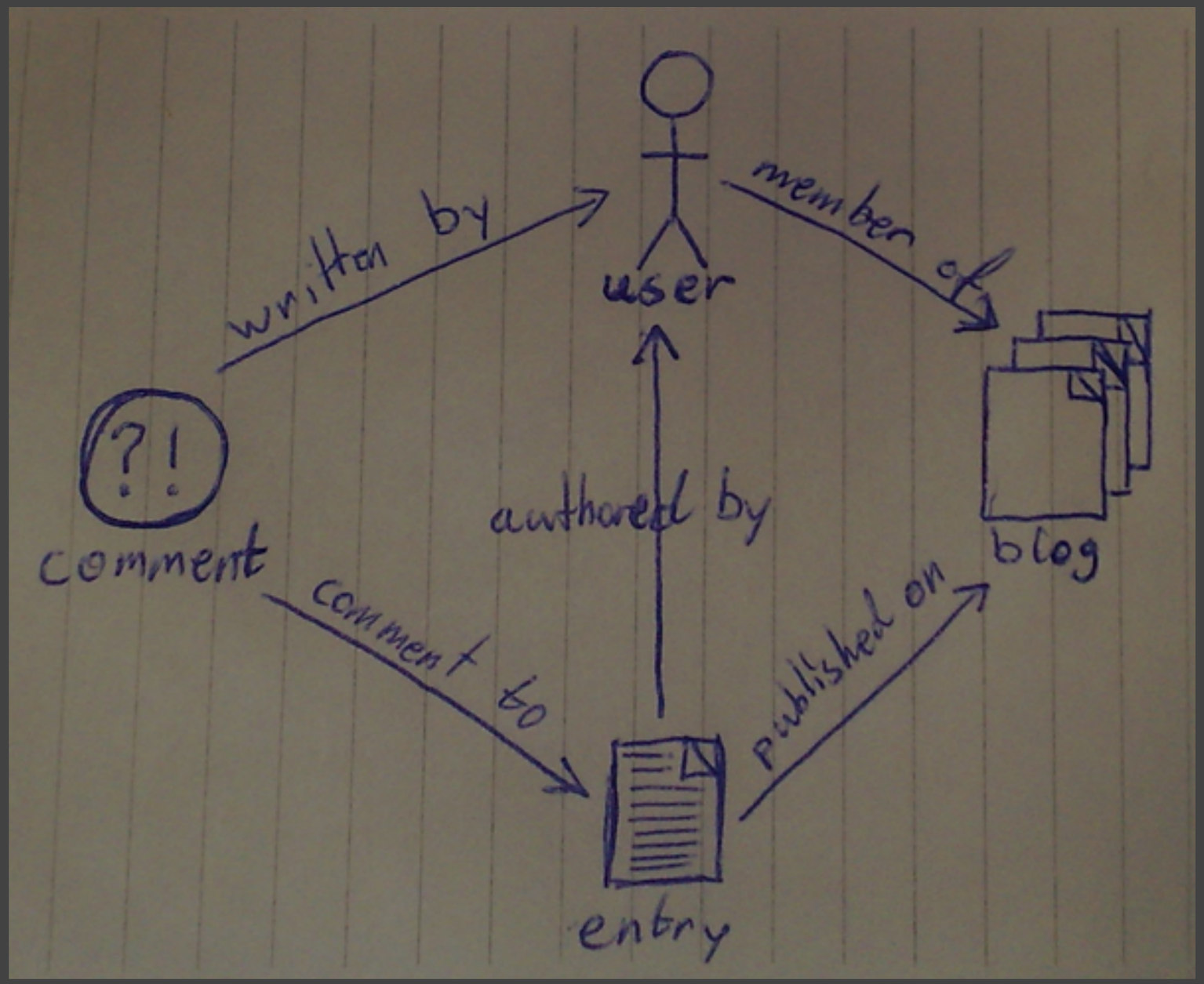
Graph DB

Property Graph model



Graph DB

Graphs are whiteboard friendly



An application domain model outlined on a whiteboard or piece of paper would be translated to an ER-diagram, then normalized to fit a Relational Database. With a Graph Database the model from the whiteboard is implemented directly.

Graph DB

Image credits: Tobias Ivarsson

Graphs are whiteboard friendly

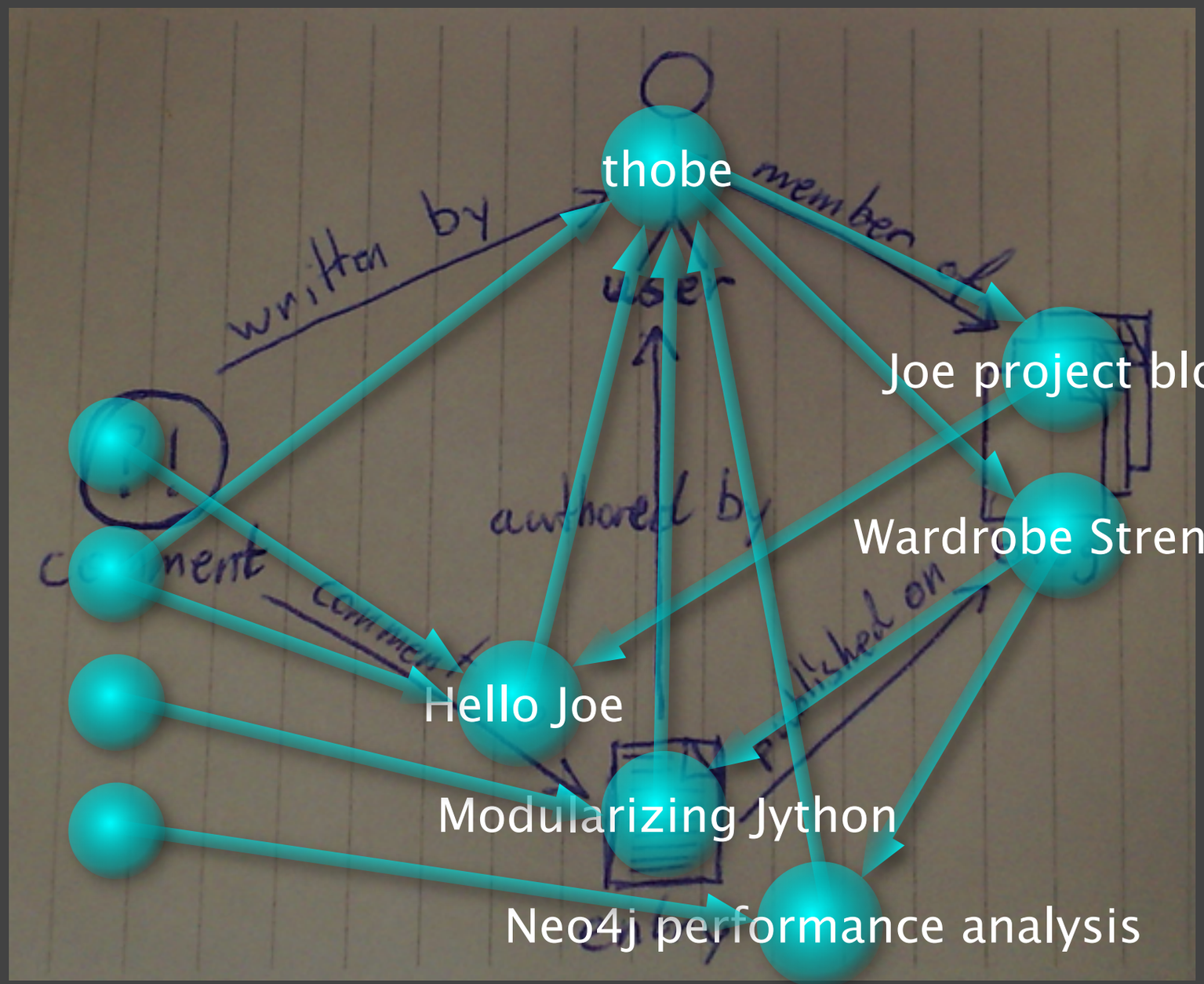


Image credits: Tobias Ivarsson

An application domain model outlined on a whiteboard or piece of paper would be translated to an ER-diagram, then normalized to fit a Relational Database. With a Graph Database the model from the whiteboard is implemented directly.

Graph DB

Graph db: Creating a social graph

```
GraphDatabaseService graphDb = new EmbeddedGraphDatabase(
   GRAPH_STORAGE_LOCATION );

Transaction tx = graphDb.beginTx();
try {
    Node mrAnderson = graphDb.createNode();
    mrAnderson.setProperty( "name", "Thomas Anderson" );
    mrAnderson.setProperty( "age", 29 );

    Node morpheus = graphDb.createNode();
    morpheus.setProperty( "name", "Morpheus" );
    morpheus.setProperty( "rank", "Captain" );

    Relationship friendship = mrAnderson.createRelationshipTo(
      morpheus, SocialGraphTypes.FRIENDSHIP );

    tx.success();
} finally {
    tx.finish();
}
```

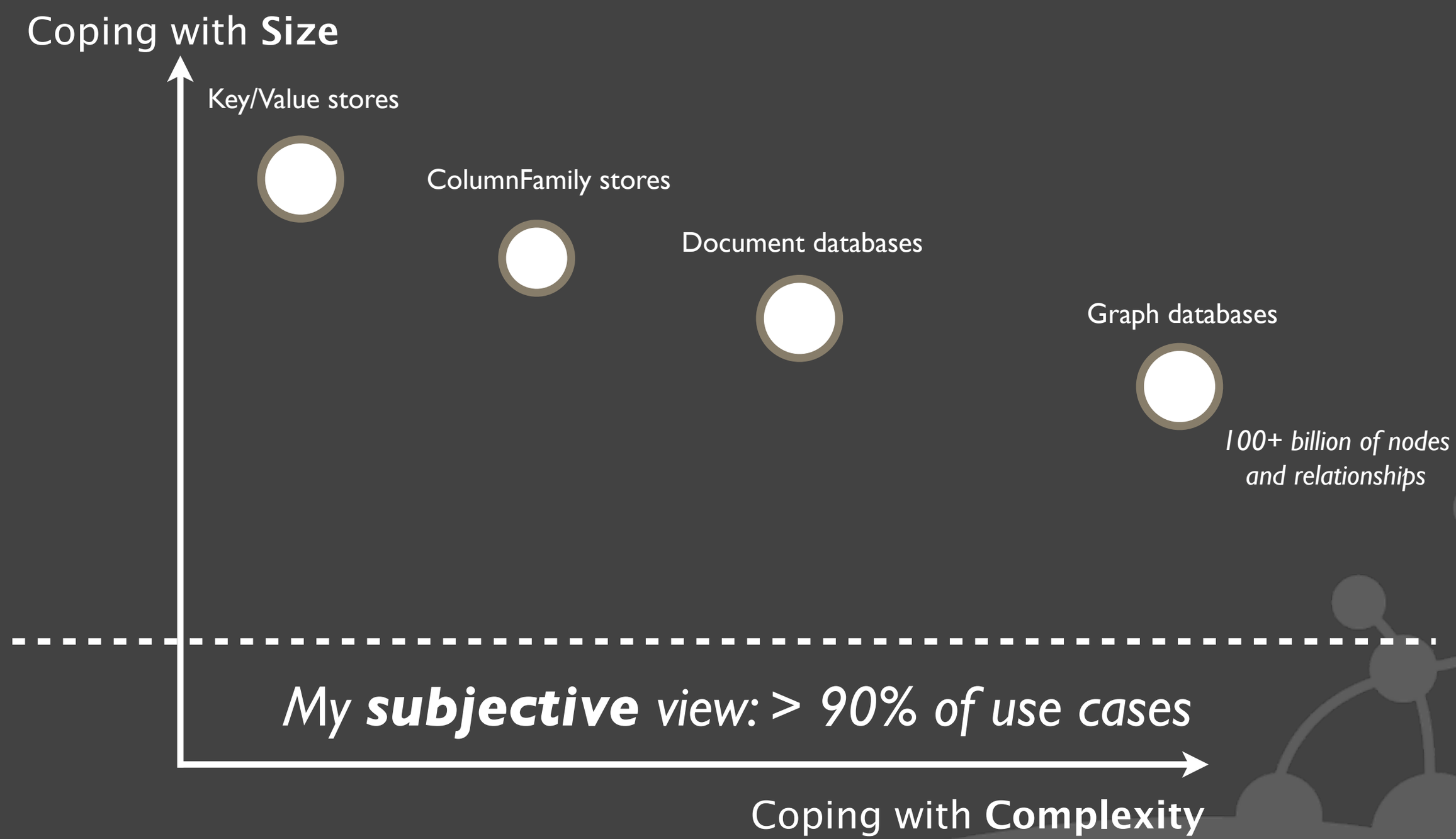
Graph db: How do I know this person?

```
Node me = ...
Node you = ...

PathFinder shortestPathFinder = GraphAlgoFactory.shortestPath(
    Traversals.expanderForTypes(
        SocialGraphTypes.FRIENDSHIP, Direction.BOTH ),
    /* maximum depth: */ 4 );

Path shortestPath = shortestPathFinder.findSinglePath(me, you);

for ( Node friend : shortestPath.nodes() ) {
    System.out.println( friend.getProperty( "name" ) );
}
```

Four emerging NOSQL categories

- Key-Value stores
- ColumnFamiy stores
- Document databases
- Graph databases

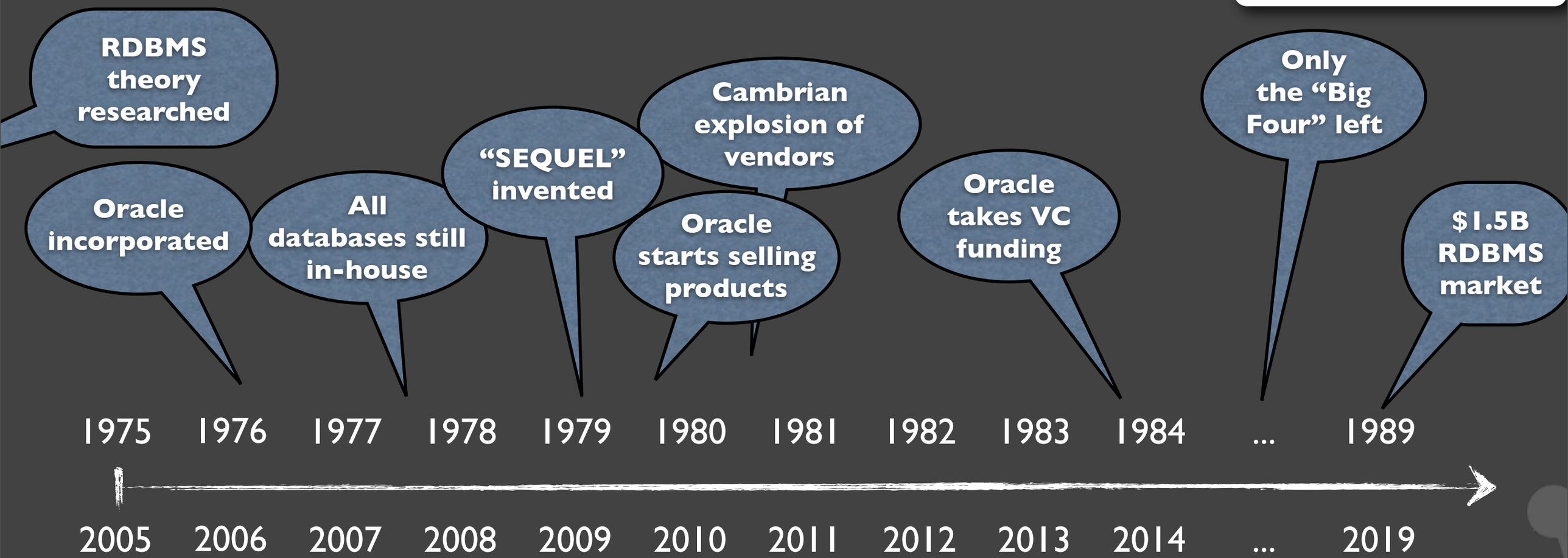
Scaling to size vs. Scaling to complexity

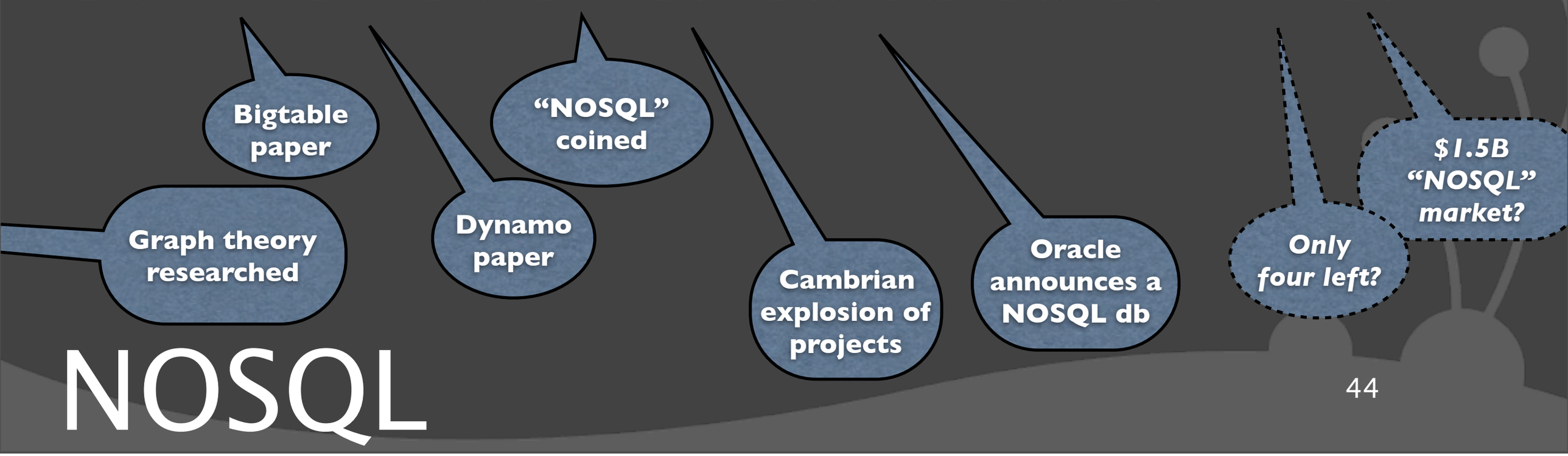


NOSQL

a brief excursion into the past

SQL


RDBMS theory researched
Oracle incorporated
All databases still in-house
"SEQUEL" invented
Oracle starts selling products
Cambrian explosion of vendors
Oracle takes VC funding
Only the "Big Four" left
$1.5B RDBMS market
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
...
1989
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
...
2019


Graph theory researched
Bigtable paper
Dynamo paper
"NOSQL" coined
Cambrian explosion of projects
Oracle announces a NOSQL db
Only four left?
$1.5B "NOSQL" market?

NOSQL

NOSQL

The Future

Four trends

- **More ACIDity**
  - Mongo adding durable logging storage in 1.7
  - Tunable consistency in Apache Cassandra
  - Roger Bodamer
    - `uptime ( CP + average developer )`
      `>=`
      `uptime ( AP + average developer )`
      http://www.slideshare.net/iammutex/q-con-sf10rogerbodamer
  - **Makes sense - why push the burden to the developer when eventually consistency is not needed in most scenarios?**

Four trends

- **More query languages**
  - In the past year, many prominent NOSQL databases have invested heavily in query languages
  - **Cassandra**: CQL
  - **Couchbase**: UnQL
  - **Neo4j**: Cypher
  - Mongo's had it from the get go? <--- One reason for their popularity?

Four trends

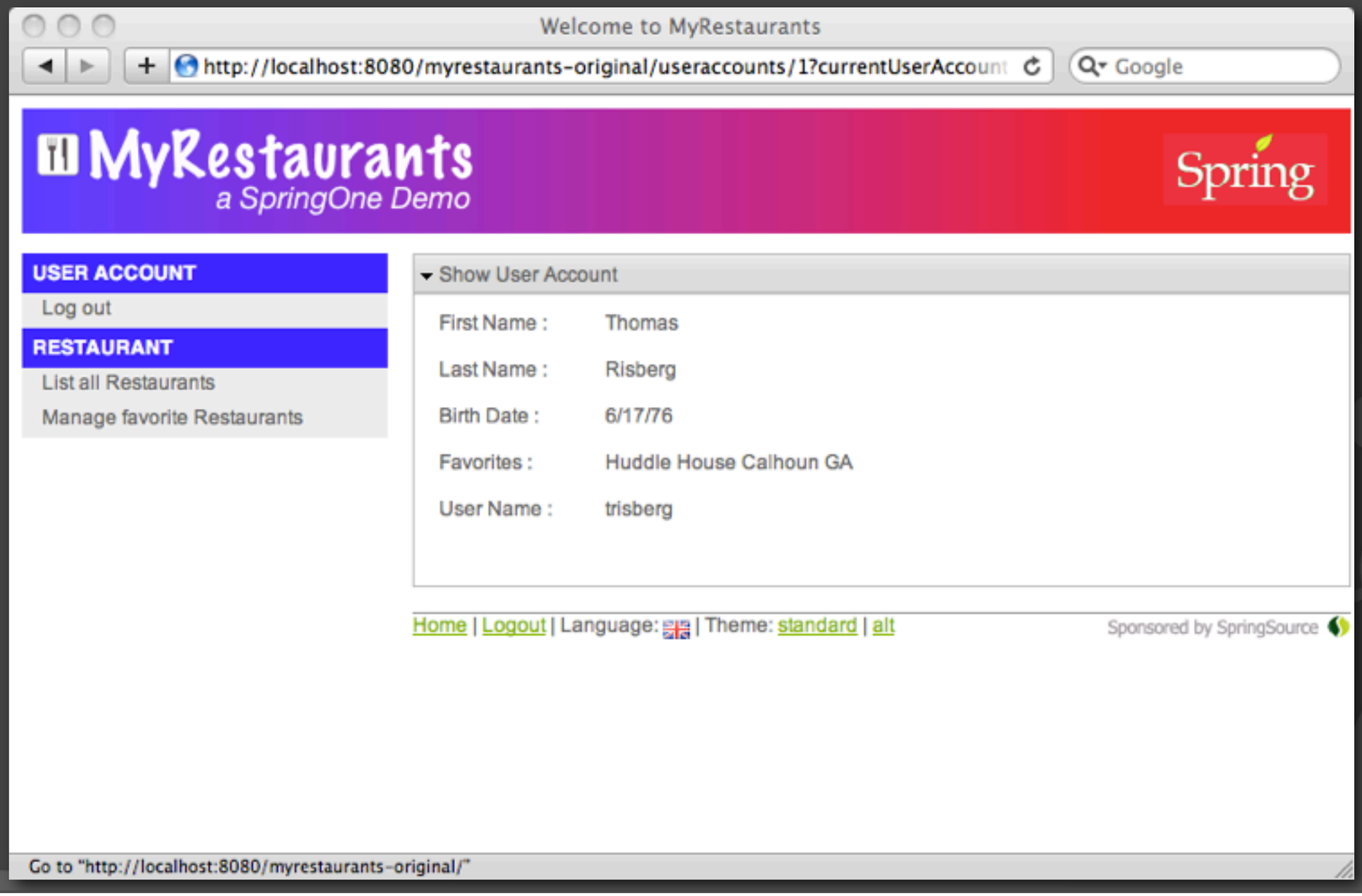
- **More schemas?**
  - Analogously, why push the full burden of schema freedom to the developer?
  - Over time, I believe we will see more schema-like support in most NOSQL stores
  - At least in document databases and graph databases, who have the richest models
  - Granted, we haven't really seen that yet

Four trends

- **Polyglot persistence will drive middleware support**
  - The era of the One Size Fits All Database is over
  - Ergo, any given system will typically work at runtime with multiple databases
  - That's all fine and dandy, except it's not because it's a pain
  - This trend will demand a lot of middleware support

Middleware support?

- Lemme tell you the story about Mike and his restaurant site



Domain & data model

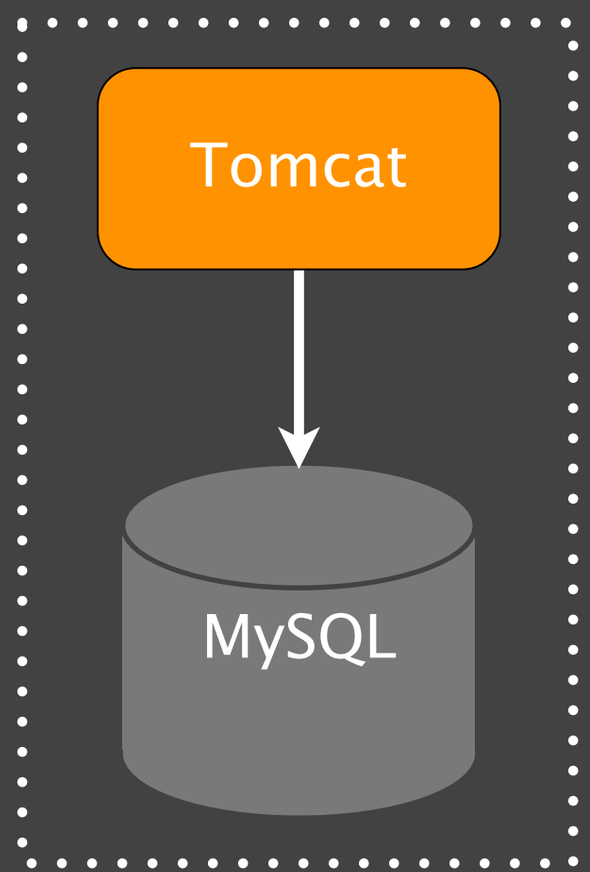
Restaurant

```
@Entity
public class Restaurant {
    @Id @GeneratedValue
    private Long id;
    private String name;
    private String city;
    private String state;
    private String zipCode;
```

UserAccount

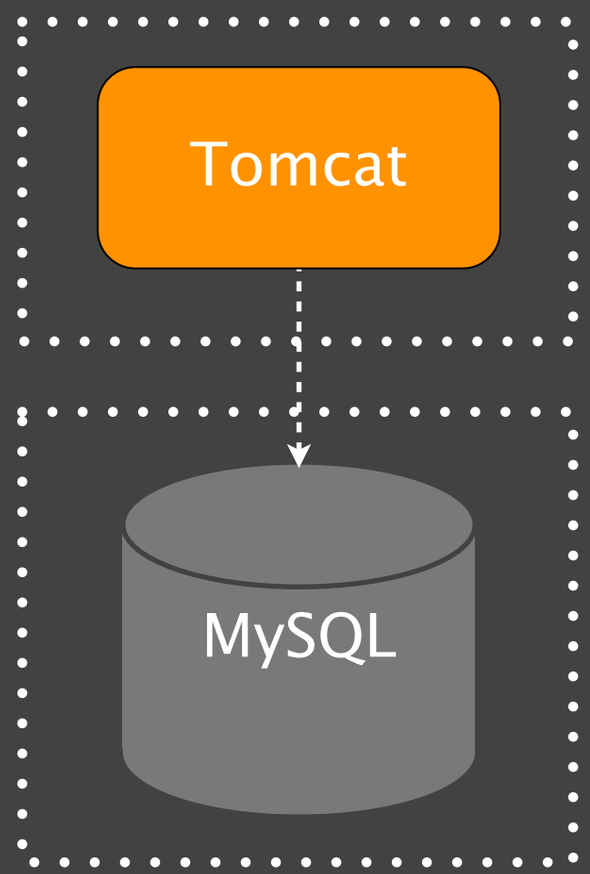
```
@Entity
@Table(name = "user_account")
public class UserAccount {
    @Id @GeneratedValue
    private Long id;
    private String userName;
    private String firstName;
    private String lastName;
    @Temporal(TemporalType.TIMESTAMP)
    private Date birthDate;
    @ManyToMany(cascade = CascadeType.ALL)
    private Set<Restaurant> favorites;
```

Step 1: Buildsing a web site



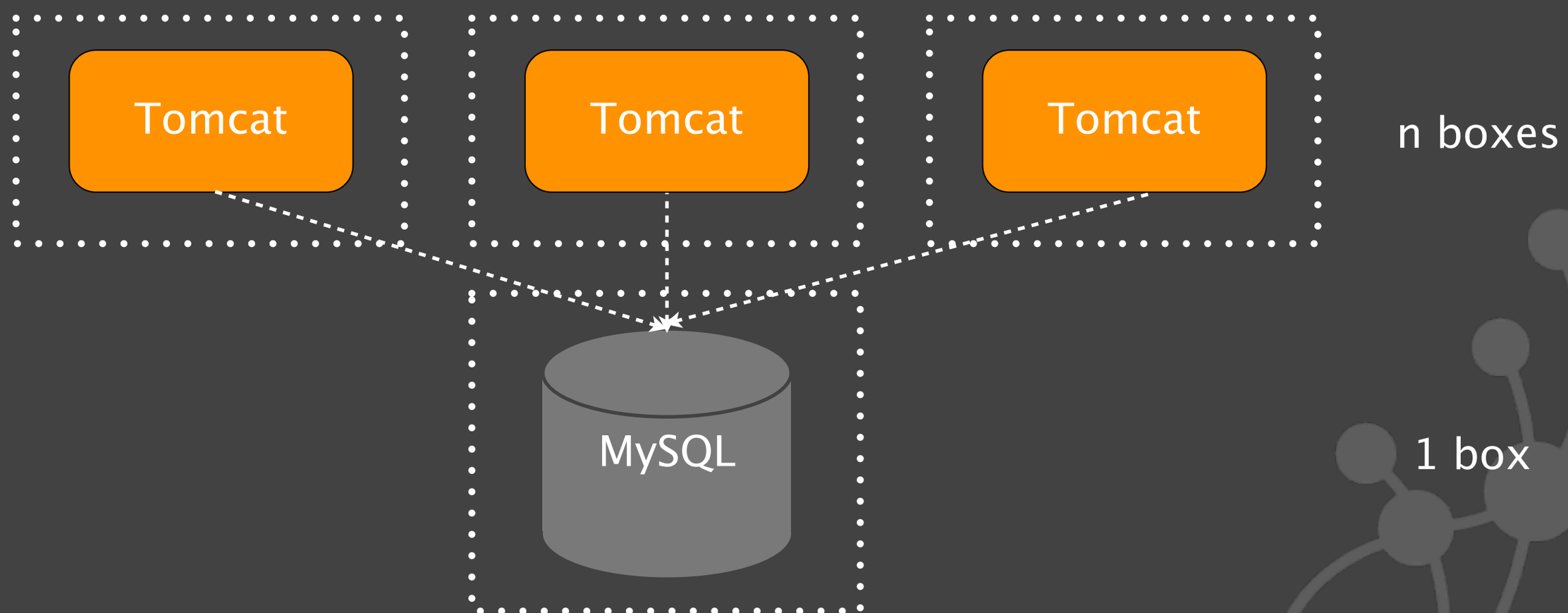
One box

Step II: Whoa, ppl are actually using it?

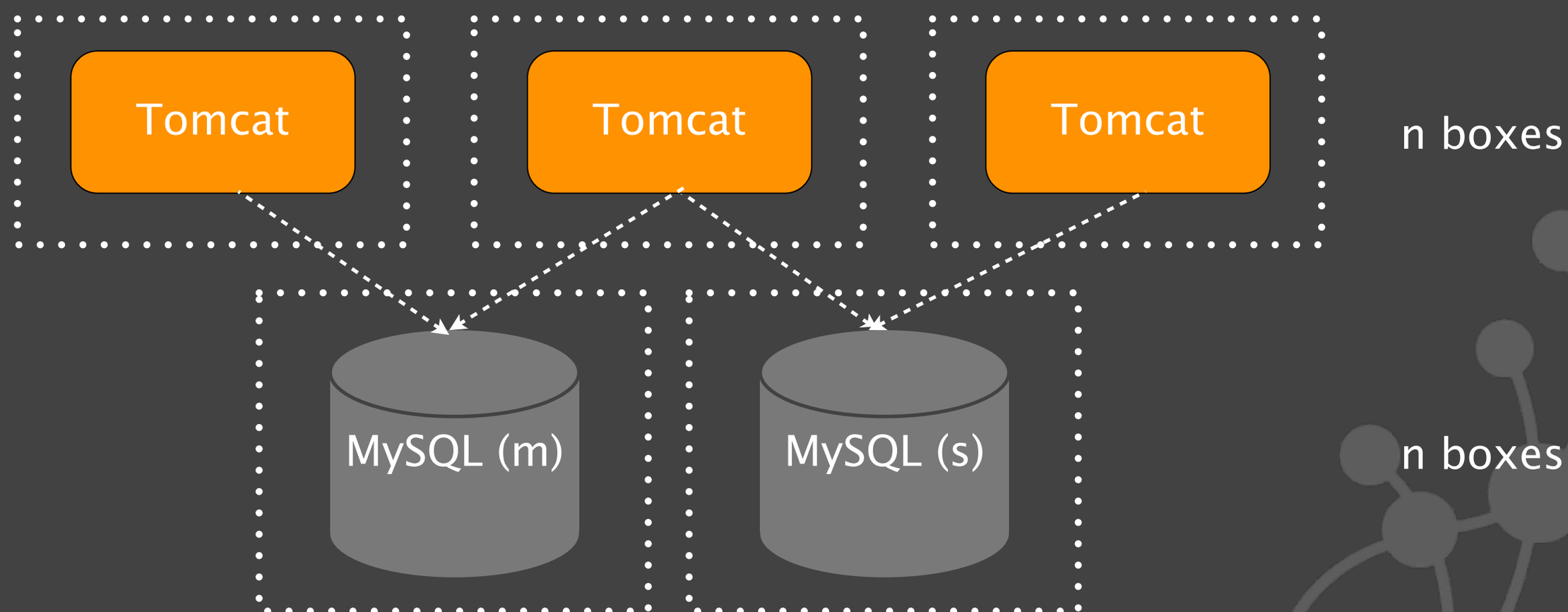


Two boxes

Step III: That's a LOT of pages served...



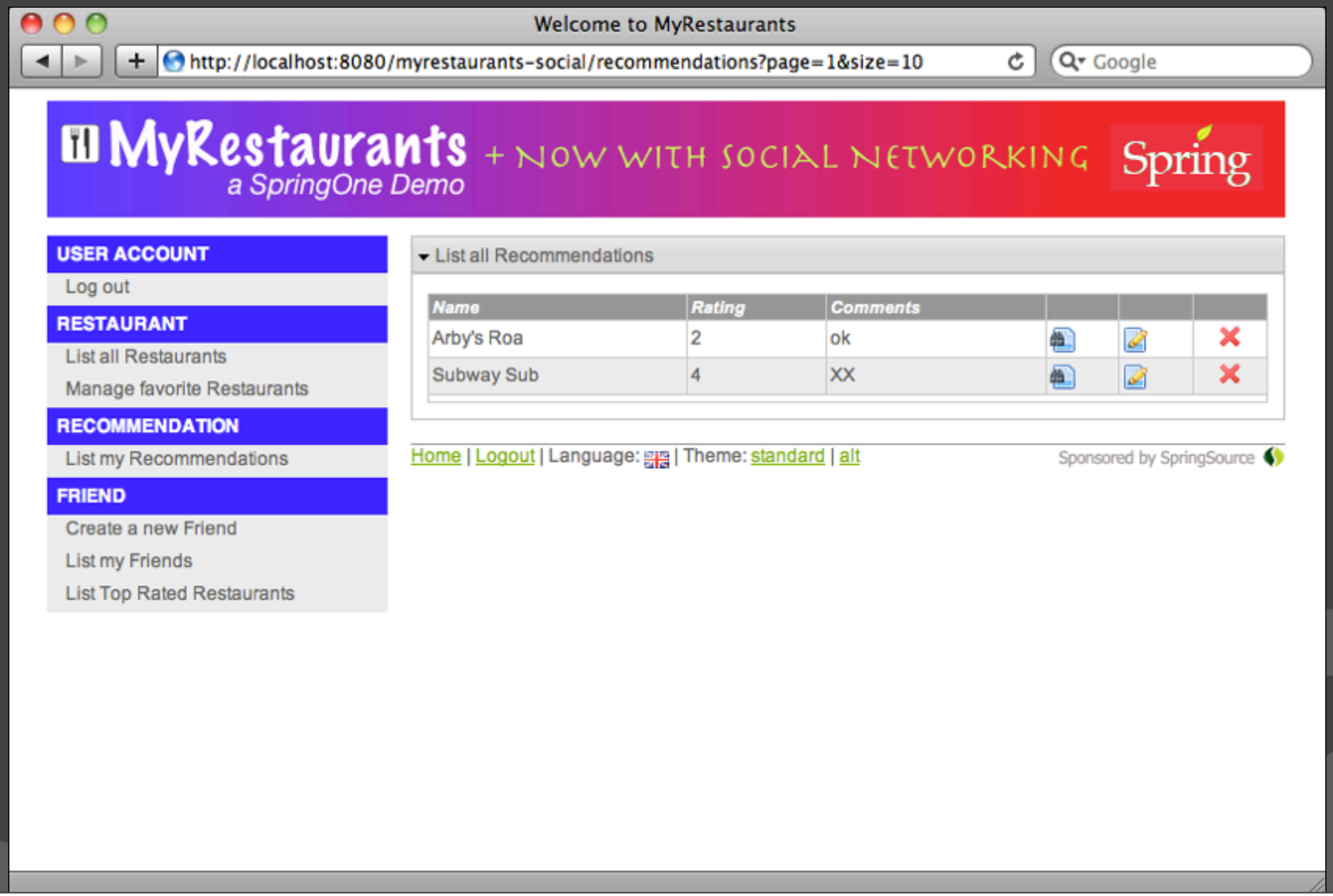
Step IV: Our DB is completely overwhelmed...



Step V: Our DBs are STILL overwhelmed

?

What does the site look like now?



Step V: Our DBs are STILL overwhelmed

- Turns out the problem is due to joins

- A while back Mike introduced a new feature
  - Recommend restaurants based on the user's friends (and friends of friends)
  - Whoa, recommendations aren't just simple get and put!
  - They're killing us with joins

- What about sharding?
- What about SSDs?

Polyglot persistence (Not Only SQL)

- How did we get into this situation?

- Well, data sets are increasingly less uniform
  - Parts of Mike's data fits well in an RDBMS
  - But parts of it is graph-shaped
    - It fits much better in a graph database!
    - And I'm sure that there is or will be very key-value-esque parts of the dataset

- Simple, just store some of it in a graph db and some of it in MySQL! But what does the code look like?

We were here

Restaurant

```
@Entity
public class Restaurant {
    @Id @GeneratedValue
    private Long id;
    private String name;
    private String city;
    private String state;
    private String zipCode;
```

UserAccount

```
@Entity
@Table(name = "user_account")
public class UserAccount {
    @Id @GeneratedValue
    private Long id;
    private String userName;
    private String firstName;
    private String lastName;
    @Temporal(TemporalType.TIMESTAMP)
    private Date birthDate;
    @ManyToMany(cascade = CascadeType.ALL)
    private Set<Restaurant> favorites;
```

Then we added recommendations

Restaurant

```
@Entity
@NodeEntity(partial = true)
public class Restaurant {
  @Id @GeneratedValue
  private Long id;
  private String name;
  private String city;
  private String state;
  private String zipCode;
```

UserAccount

```
@Entity
@Table(name = "user_account")
@NodeEntity(partial = true)
public class UserAccount {
  @Id @GeneratedValue
  private Long id;
  private String userName;
  private String firstName;
  private String lastName;
  @Temporal(TemporalType.TIMESTAMP)
  private Date birthDate;
  @ManyToMany(cascade = CascadeType.ALL)
  private Set<Restaurant> favorites;

  @GraphProperty
  String nickname;
  @RelatedTo(type = "friends",
       elementClass = UserAccount.class)
  Set<UserAccount> friends;
  @RelatedToVia(type = "recommends",
       elementClass = Recommendation.class)
  Iterable<Recommendation> recommendations;
```

Recommendation

```
@RelationshipEntity
public class Recommendation {
  @StartNode
  private UserAccount user;
  @EndNode
  private Restaurant restaurant;
  private int stars;
  private String comment;
```

Then we added recommendations

Restaurant

```
@Entity
@NodeEntity(partial = true)
public class Restaurant {
  @Id @GeneratedValue
  private Long id;
  private String name;
  private String city;
  private String state;
  private String zipCode;
```

UserAccount

```
@Entity
@Table(name = "user_account")
@NodeEntity(partial = true)
public class UserAccount {
  @Id @GeneratedValue
  private Long id;
  private String userName;
  private String firstName;
  private String lastName;
  @Temporal(TemporalType.TIMESTAMP)
  private Date birthDate;
  @ManyToMany(cascade = CascadeType.ALL)
  private Set<Restaurant> favorites;
```

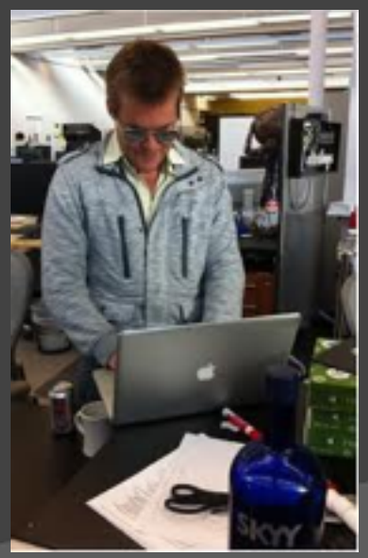
```
@GraphProperty
String nickname;
@RelatedTo(type = "friends",
     elementClass = UserAccount.class)
Set<UserAccount> friends;
@RelatedToVia(type = "recommends",
     elementClass = Recommendation.class)
Iterable<Recommendation> recommendations;
```

Recommendation

```
@RelationshipEntity
public class Recommendation {
  @StartNode
  private UserAccount user;
  @EndNode
  private Restaurant restaurant;
  private int stars;
  private String comment;
```

And we're back to talking about Middleware

- This example was from the Spring Data project
  - Specifically Spring Data Neo4j
  - Available at: http://www.springsource.org/spring-data/neo4j
  - Spring Data Neo4j 2.0 will be released RSN, RC out **NOW**
- But this really should be available in any middleware stack
  - Ask Redhat / JBoss
  - Ask David & the brogrammers
  - Ask Sun / Oracle
  - Ask Microsoft



Four NOSQL trends

- More ACIDity
- More query languages
- More schemas
- More middleware support

Conclusion

Ait, what's your point?

- There's an explosion of 'nosql' databases out there
  - Some are immature and experimental
  - Some are coming out of years of battle-hardened production
- NOSQL is about finding the right tool for the job
  - Sometimes that's an RDBMS
  - But increasingly commonly a NOSQL db is the perfect fit
- We **will** have heterogenous data backends in the future
  - Now the rest of the stack needs to step up and help developers cope with that

Key takeaway

Not **O**nly **SQL**

is here

Key takeaway

Not **O**nly **SQL**

is here to stay

Key takeaway

Not **O**nly **SQL**
is FUN - dl & play
around now!

Questions?

Image credits: Lost! Sorry... :(

http://neotechnology.com