O'REILLY®

oscon

open source convention

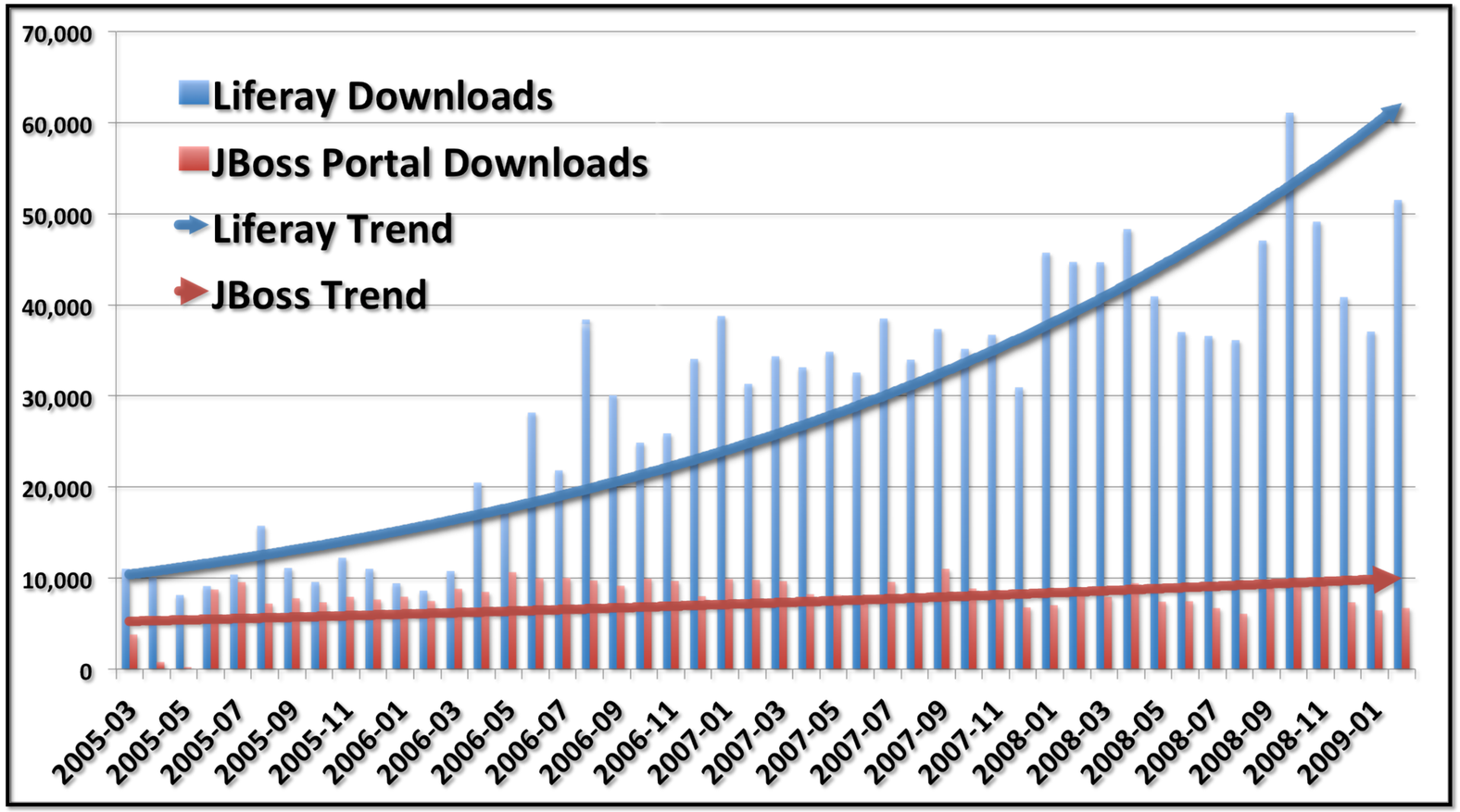

oscon.com

# NoSQL @ Liferay

James Falkner
Community Manager
Liferay, Inc.

# Liferay

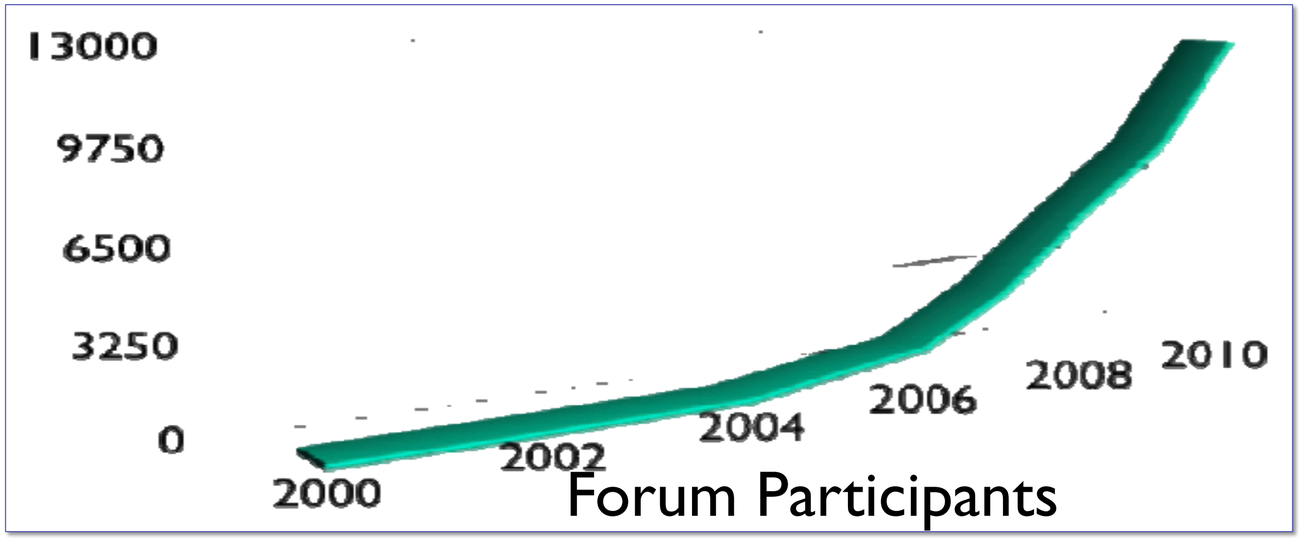
- 10 Years of open source
- Most popular and widely downloaded open source portal
- Over 4 Million downloads and 50,000 downloads per month
- 46,053 registered users on liferay.org
- 16,343 forum participants
- 164,342 forum posts
- 50+ Active contributors (excluding employees)
- ~400k CE Deployments
- ~6M LOC
- LGPL
- http://github.com/liferay
- http://liferay.com/downloads

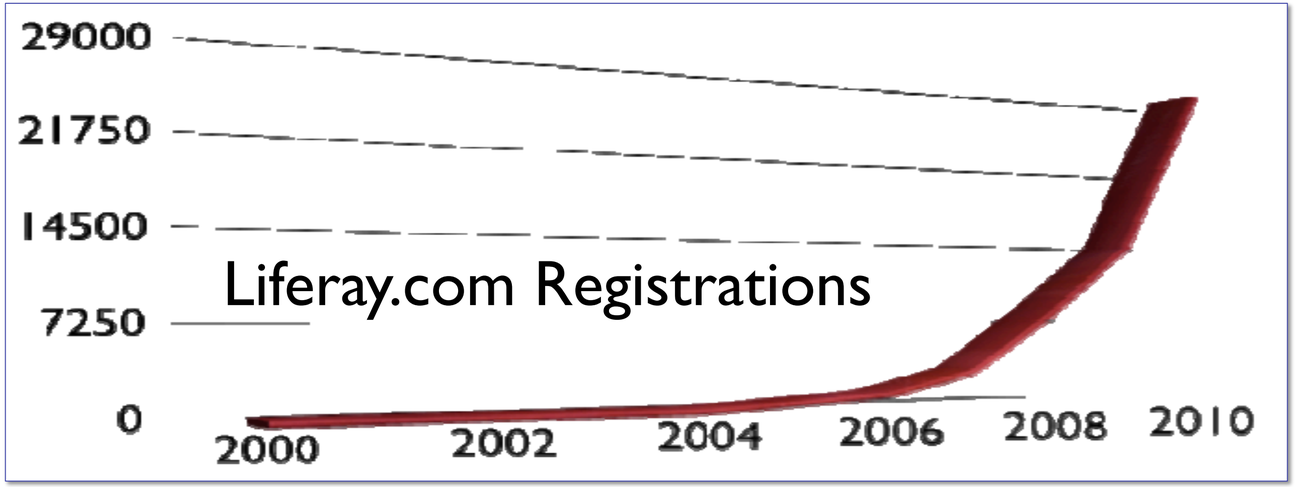



*statistics taken from SourceForge.net

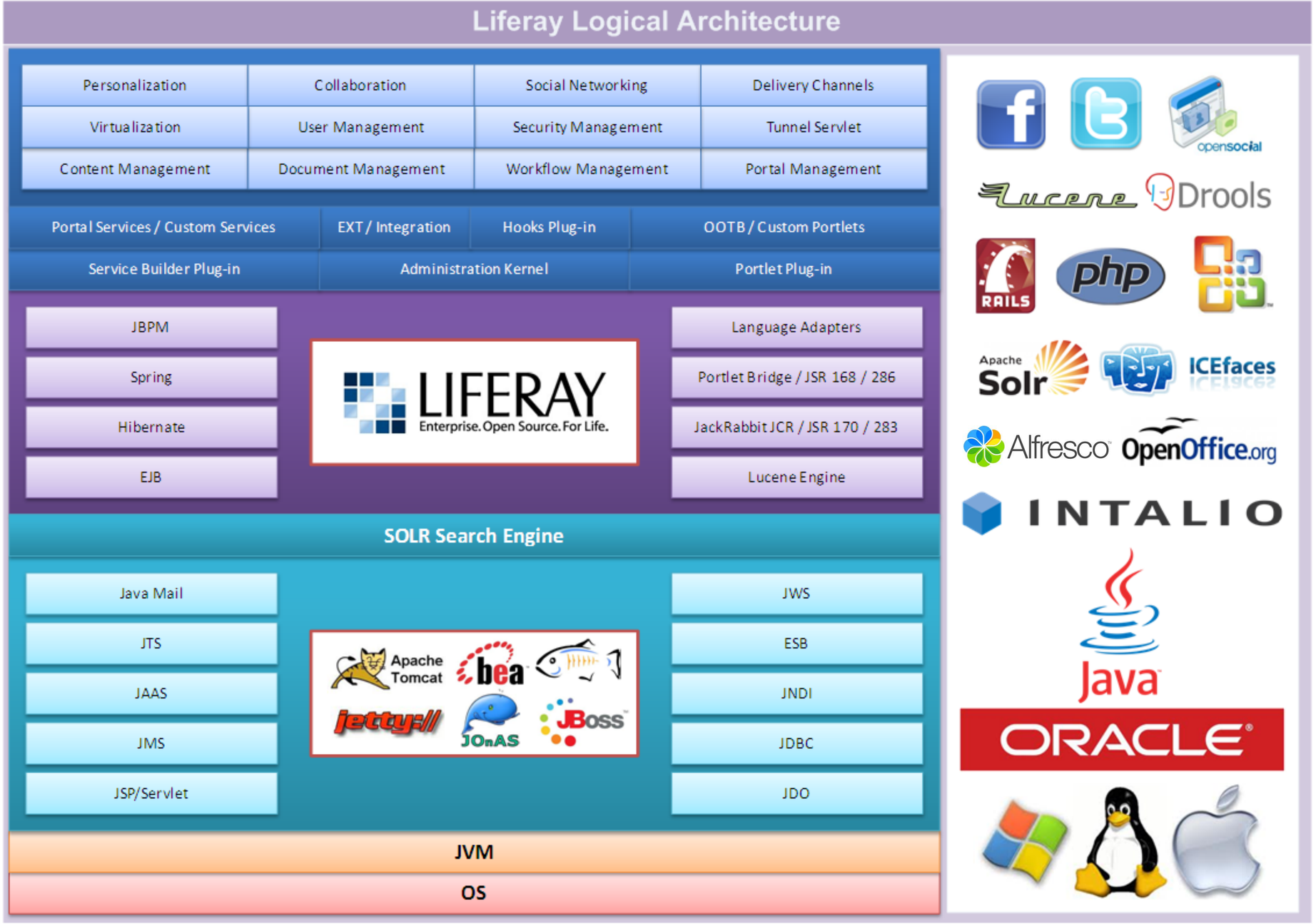
# Community Contributions

- Communities
- Virtual Hosts
- Page & Organizational Hierarchy
- EXT Plugins
- Translations
- Control Panel
- Improved SEO
- Flag As Inappropriate
- Wiki Advancements
- Netvibes, iGoogle Integration
- Reporting Portlet
- Read/Write Database Splitting
- Testing, Bug Reporting, Help Desk, Bug Fixing, Documentation



Liferay




Marketecture

# It’s all about the DATA!

# Data

- Comes in all shapes and sizes and we deal with it in different ways based on its characteristics: size, number, volatility, longevity, updateability, complexity, relations, etc
- Data Store: “space” where data is “stored” for the duration of its CRUD lifecycle

# Pigeonholed Data

Filesystem

OR

RDBMS (SQL)

# Other Common Data Stores

- In recent years, different types of data stores have become more prolific
- Cache?
  - If the rule of a data store is defined by CRUD lifecycle, the yes, caches are data stores!
    - Liferay: memcached, ehcache, request/session caching, or Terracotta
  - Is one of the most prominent concerns in scaling for performance in web-based systems

# Still More Data Stores

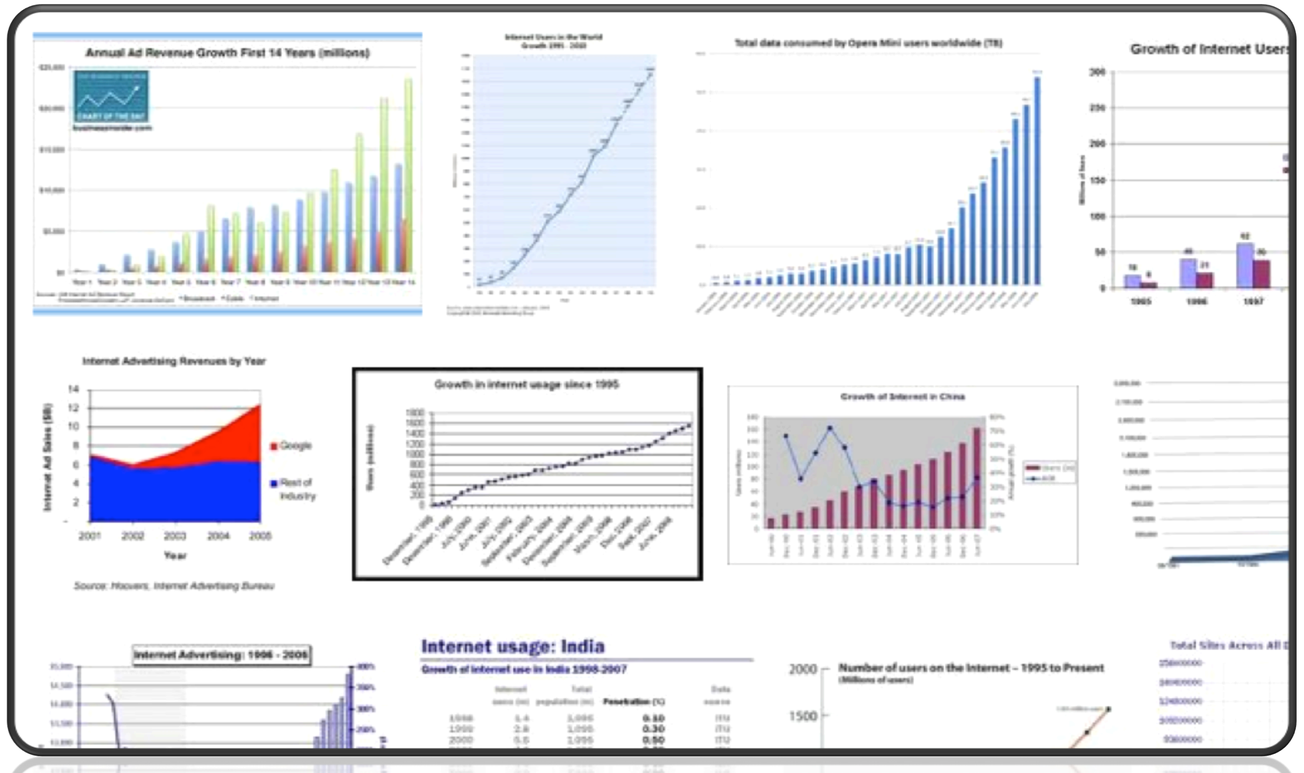
- Most such applications today (including Liferay) many types of data stores involved in any single user operation:
  - hardware (cpu/gpu, memory, hard drives, etc)
  - file system (static resources)
  - relational database
  - cache engines (session, persistence, cluster)
  - indexing engine
  - directory server
- Know your data before you choose your implementation

# Why so Many?

- Each type of store specializes in handling particular types of data in specific ways
- Takes advantage of the characteristics of data to fulfill underlying requirements
  - Many Directory Servers: Berkeley DB (non-relational)
  - Liferay: memcached (non-relational), Entity storage (relational)
  - Linux: vfs Page Cache (non-relational)
  - Facebook, etc

# Still More Data Stores

- Even more recent history has shown that there are even more scenarios where traditional "data stores" simply don't live up to the demands of modern applications



# Characteristics of Data “Handling” Systems

- ACID (Atomicity, Consistency, Isolation, Durability)
- Expressiveness of language
- Flexibility of schema
- Performance
- Scalability
- Availability
- Fault Tolerance
- Agility (ease of development)
- Cost

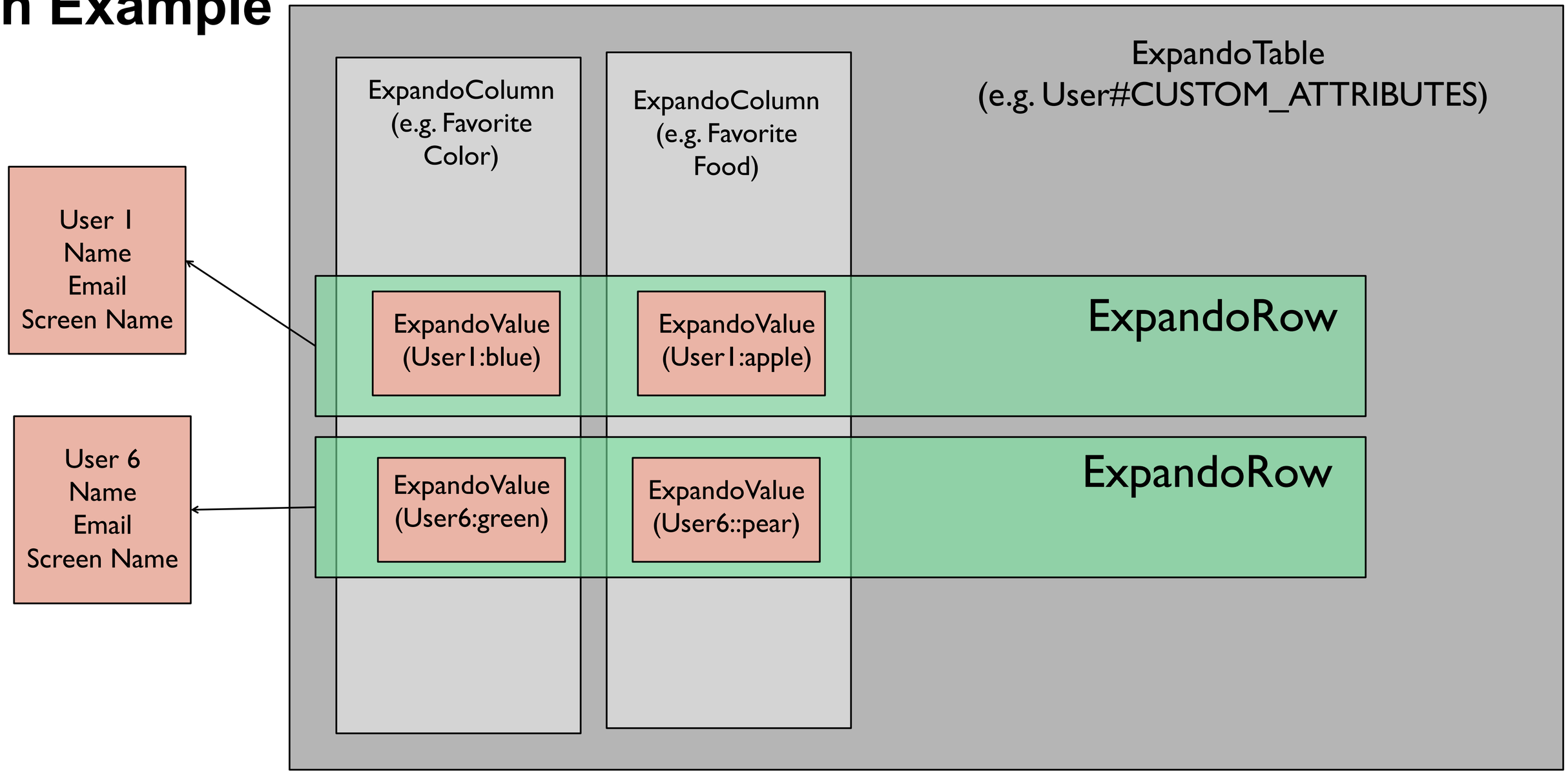
# Liferay Data

- Liferay built on top of RDBMS
  - Main reason: it was an easy choice in 2001
    - Liferay is JavaEE-based
    - Well known standard (SQL)
    - Many language bindings
    - Many vendors
    - Cheap
  - Lots of potential customers already invested in RDBMS (Oracle, DB2, SQL Server)

# An Example

- Expando
  - Name comes from JavaScript expando properties
  - *Homomorphic* data pattern (schema is data)
  - Large scales may introduce performance issues due to impedance mismatch
  - How to provide for greater scalability?
    - Use a storage engine suited to its characteristics
      - Scales well
      - Flexibile Schema
      - High Performance

# An Example

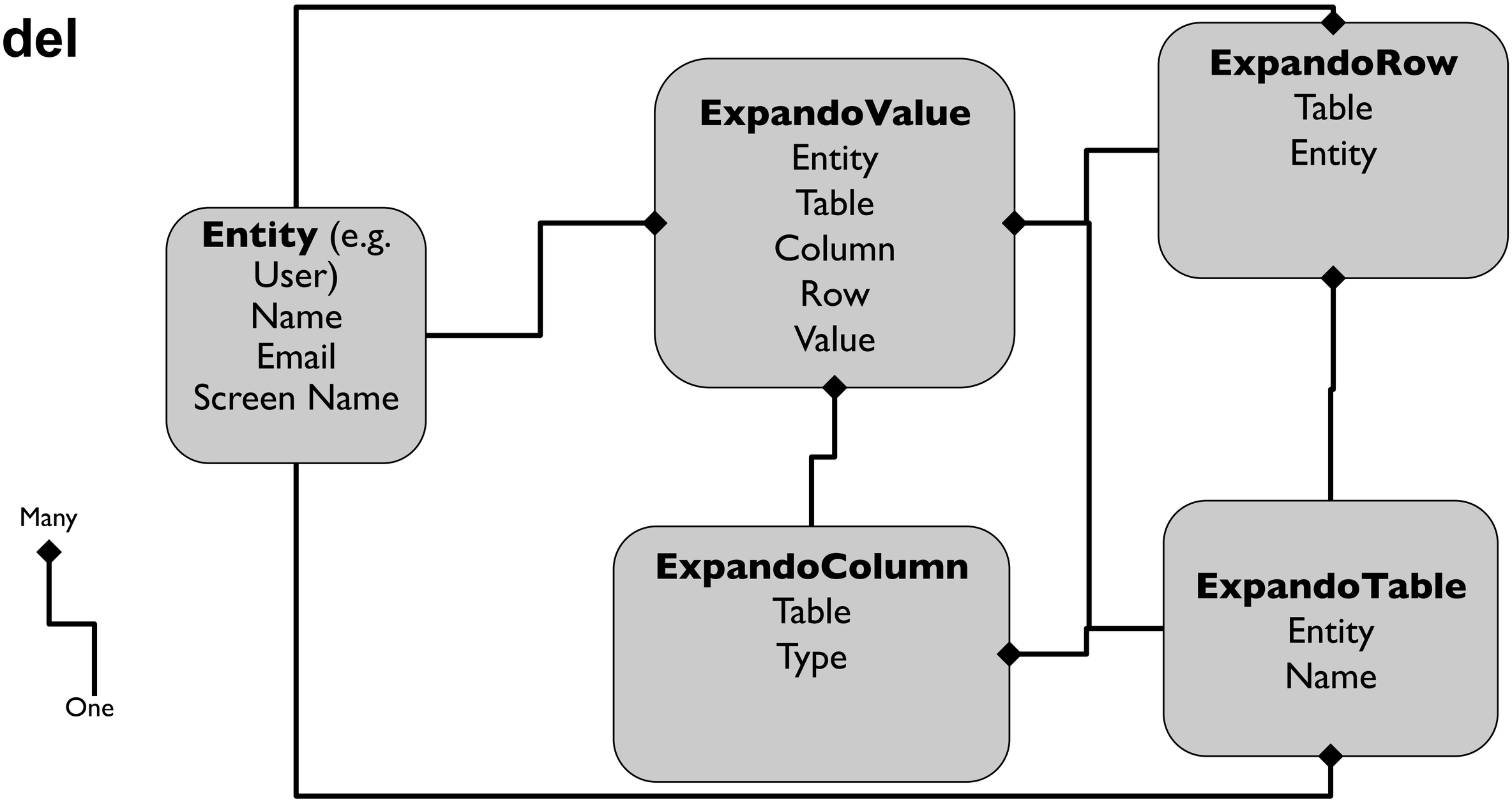

ExpandoTable
(e.g. User#CUSTOM_ATTRIBUTES)
ExpandoColumn
(e.g. Favorite
Color)
ExpandoColumn
(e.g. Favorite
Food)
User 1
Name
Email
Screen Name
User 6
Name
Email
Screen Name
ExpandoValue
(User1:blue)
ExpandoValue
(User1:apple)
ExpandoRow
ExpandoValue
(User6:green)
ExpandoValue
(User6::pear)
ExpandoRow

# Expando and SQL

- Expando
  - Non-relational, dynamic schema
  - But defined as a relational model using RDBMS
  - Dynamic Queries not supported
  - Finding expando involves a JOIN across 4 or more tables
  - Contention at high query/update rates

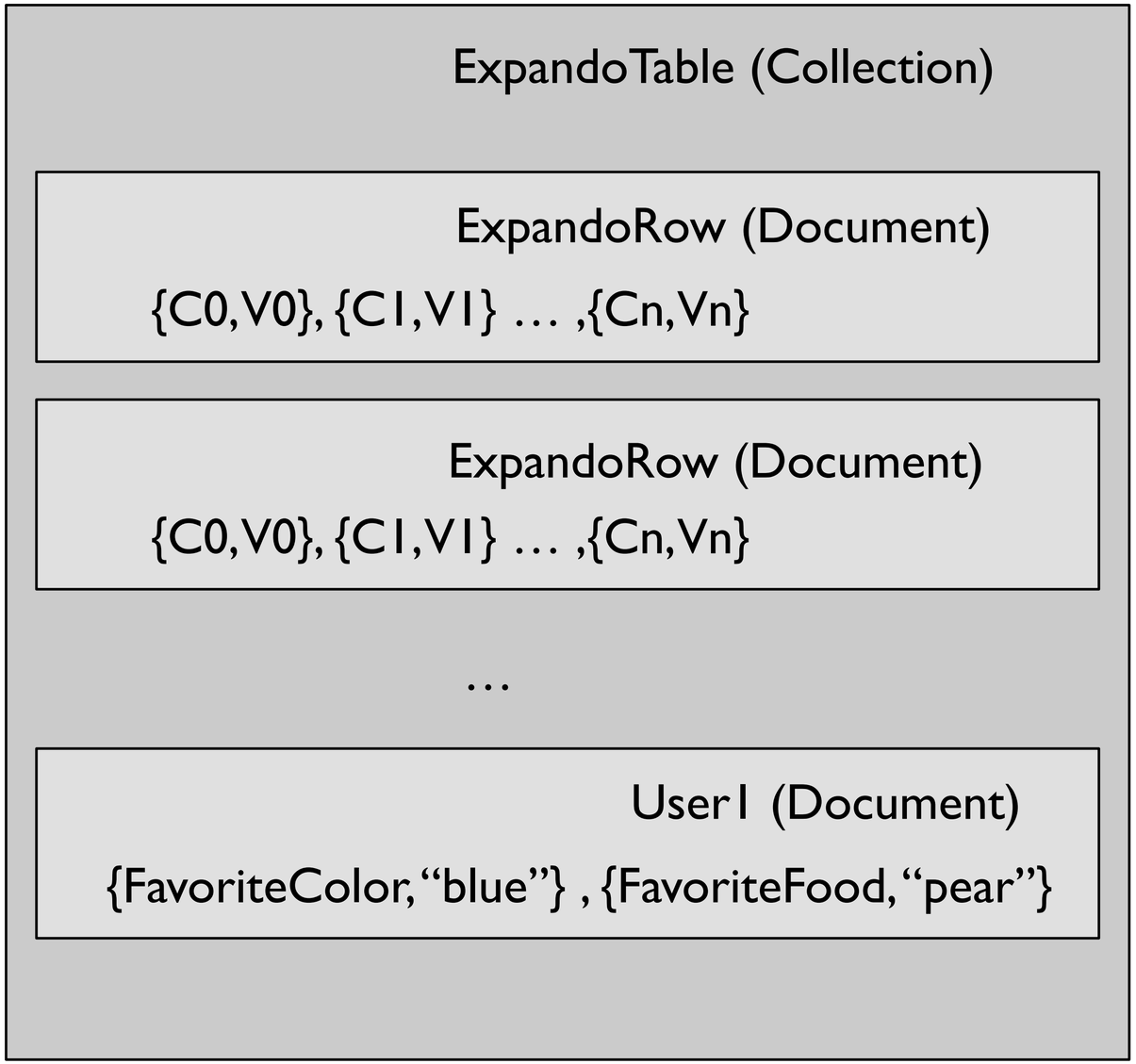
```
select
     BlogsEntry.entryId,
     BlogsEntry.title,
     ExpandoValue.data_
from
     BlogsEntry
INNER JOIN
     ExpandoTable
ON (
     ExpandoTable.companyId = BlogsEntry.companyId AND
     ExpandoTable.classNameId = (select classNameId
from ClassName_ where value =
'com.liferay.portlet.blogs.model.BlogsEntry') AND
     ExpandoTable.name = 'CUSTOM_FIELDS'
)
INNER JOIN
     ExpandoColumn
ON (
     ExpandoColumn.tableId = ExpandoTable.tableId AND
     ExpandoColumn.name like 'ListPriceAmount'
)
INNER JOIN
     ExpandoValue
ON (
     ExpandoValue.columnId=ExpandoColumn.columnId AND
     ExpandoValue.classPK = BlogsEntry.entryId
)
```

# Model


**Entity** (e.g. User)
Name
Email
Screen Name
**ExpandoValue**
Entity
Table
Column
Row
Value
**ExpandoRow**
Table
Entity
**ExpandoColumn**
Table
Type
**ExpandoTable**
Entity
Name
Many
One

# Expando using mongoDB

- Schemaless nature of Expando
- Less contention
  - High write load
  - Auto sharding
  - Collection-level locking (when available)
  - Language drivers available, tooling simple, query expressions simple
  - Further refinements possible



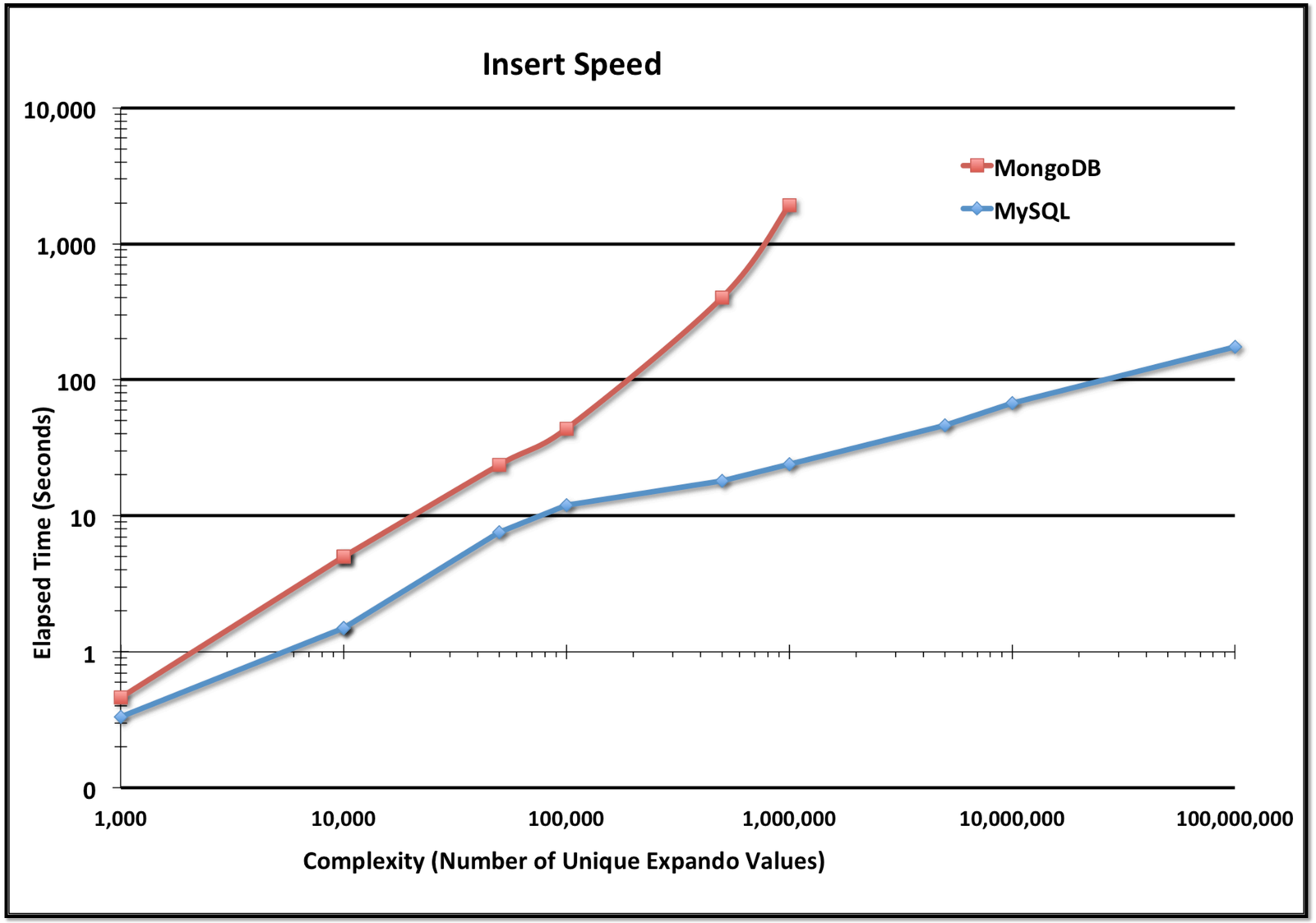

https://github.com/liferay/liferay-plugins/tree/master/hooks/mongodb-hook

# Comparison: MySQL vs mongoDB (as applied to Expando)

- Insert Performance
  - 100 to 10M Expando values, varying complexity
- Query Performance
  - 100k random queries using various data set complexity
- Query+Update Performance
  - 100k random queries or updates using various data set complexity

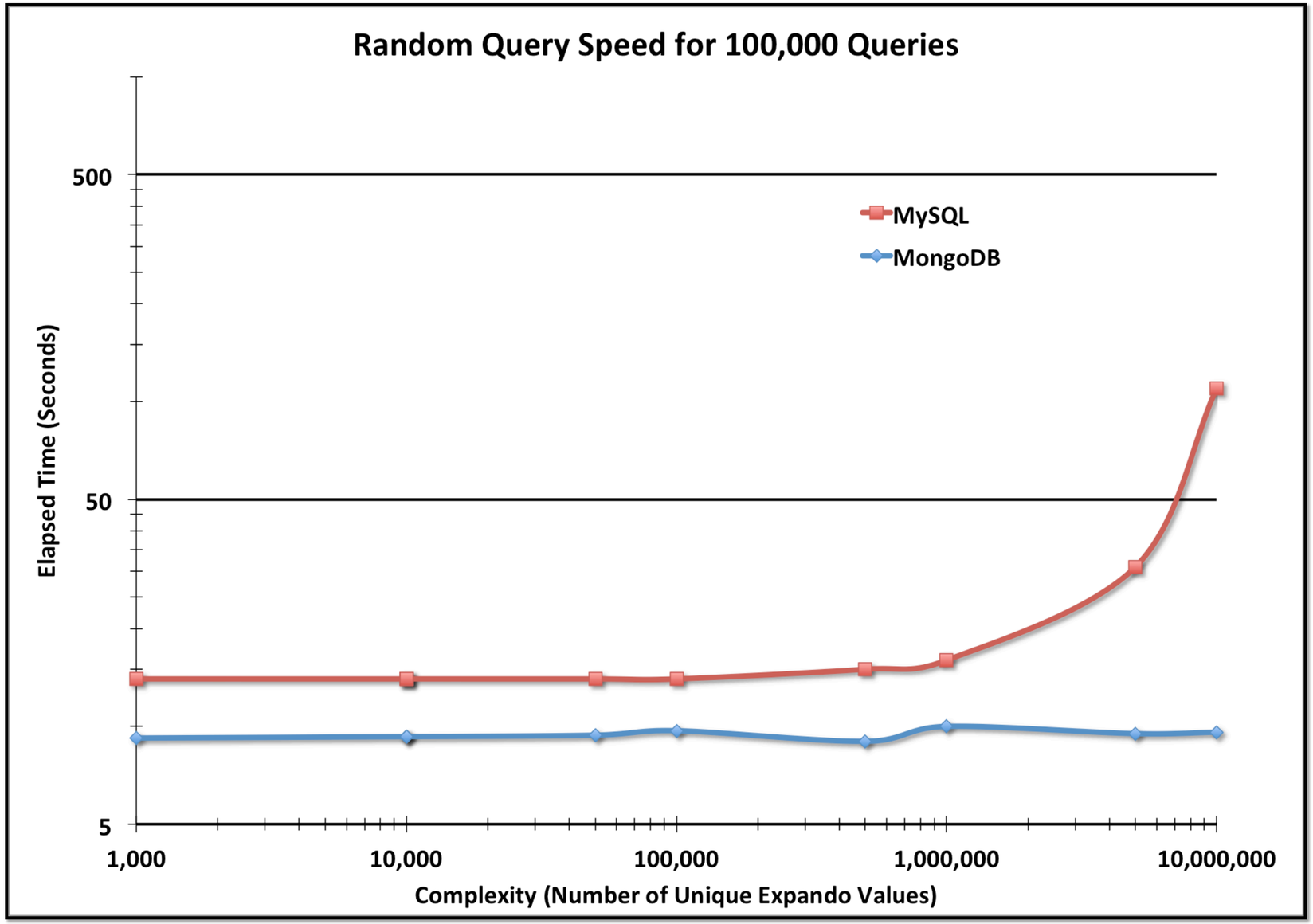
- MySQL: “Large” profile, no replication or partitioning, optimized InnoDB settings, stored procedures
- mongoDB: No sharding, no replication sets
- 20 threads per test
- Hardware: MBP dual-core Intel Core i7, 8GB

# Comparison: MySQL vs MongoDB (as applied to Expando)

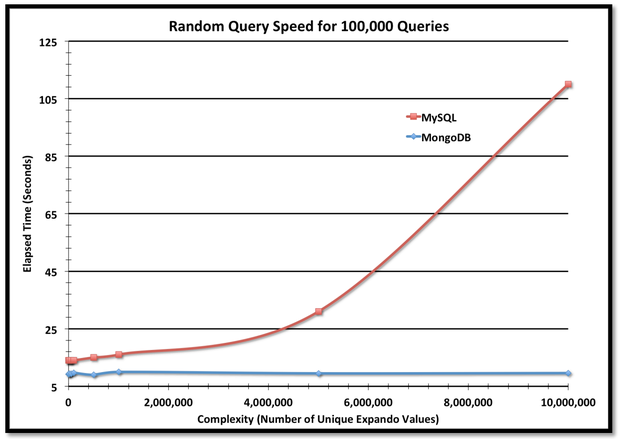



- At high complexity, successive inserts, updates, deletes require many table queries and updates for MySQL
- MongoDB Document-oriented structures can be written quickly as individual documents for entire rows, within a separate collection for each Entity-Table
- Optimizations can help (partitioning/sharding, memory tables, etc)

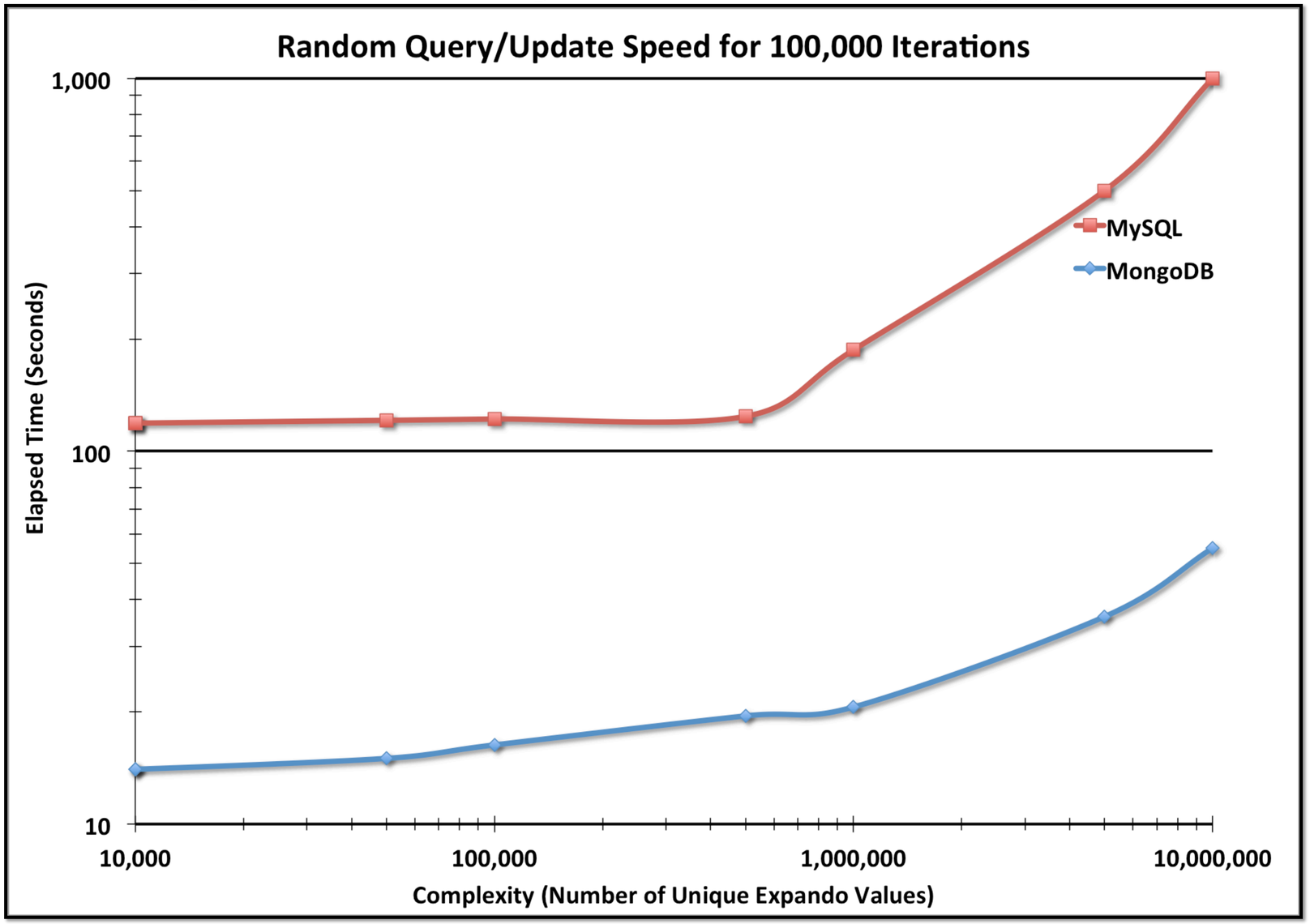
# Comparison: MySQL vs mongoDB (as applied to Expando)



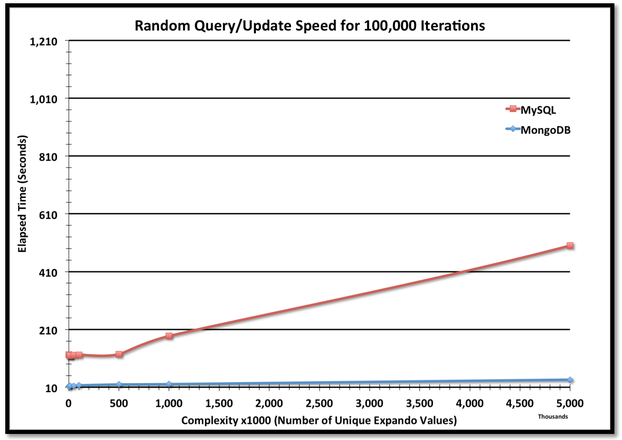

- Reasonable at lower complexity
- 1M entries = 100 tables x 100 attributes x 100 users – not that many
- Optimizations can help (e.g. partitioning and reorganization of tables)



# Comparison: MySQL vs mongoDB (as applied to Expando)

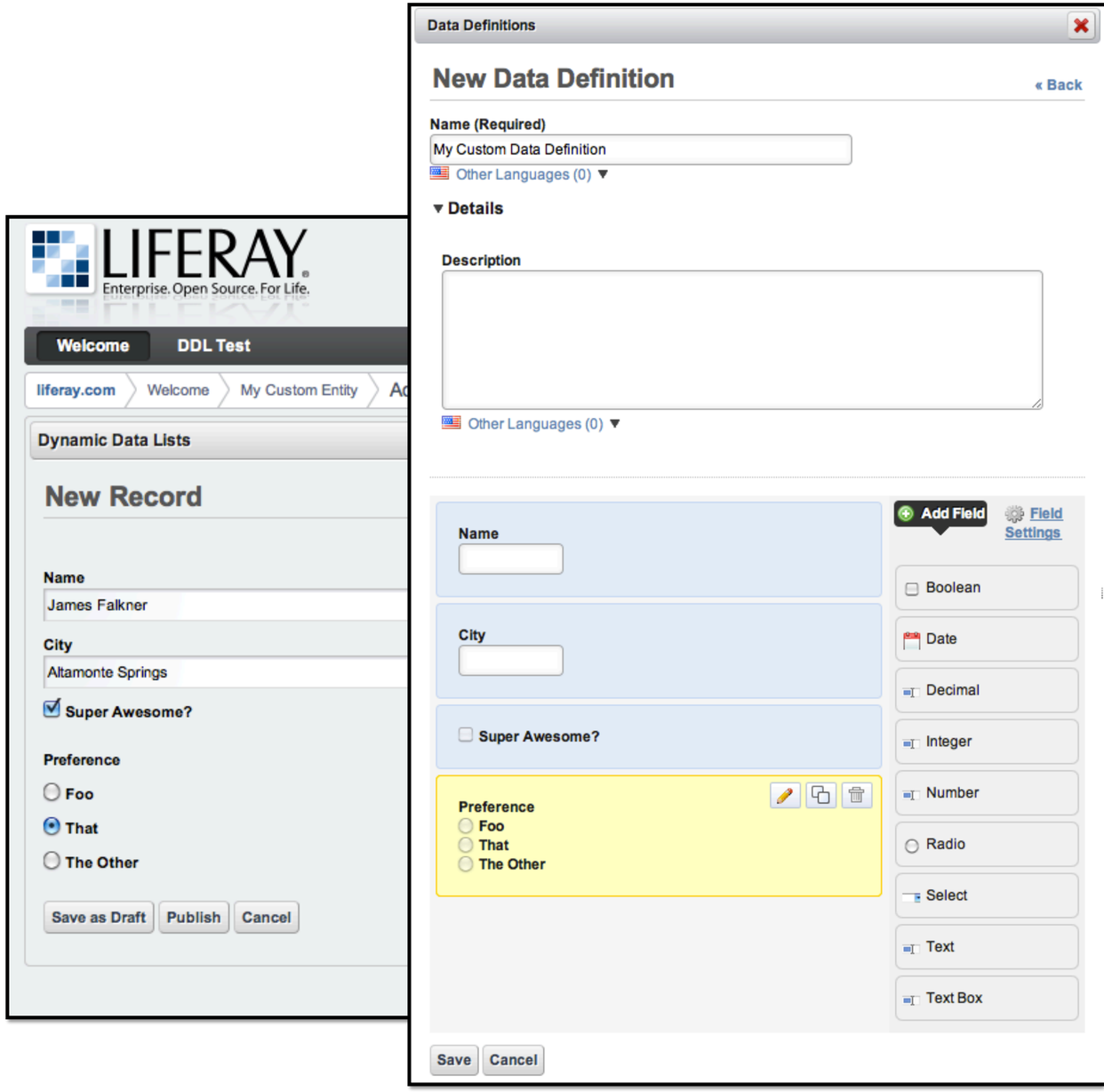



- High contention for both
- In-place writing of mongoDB performed well, when # writers ~ # readers (think Facebook, Twitter)
- Optimizations can help (partitioning, memory tables, etc)



# Another Example

- Dynamic Data Lists
  - End Users define custom entities to store into database, forms for capture and display
    - Project Status
    - Surveys/Polls
    - Arbitrary, exensible, typed data entry
- CMS, Document Library



# More Examples

- mongoj – Java persistence service generator and ORM for mongoDB
  - Generates mongoDB mapping for entity definitions in xml
    - Indexing, Spring Config
    - http://github.com/pdd/mongoj

```
user = {
    "firstName" : "Joe",
    "lastName" : "Black",
    "image" : <binary image data>,
    "info" : {
        "dob" : <date object>,
        "address" : {
            "street" : "10 main street",
            "zip" : 12345
        },
        "phone" : "102345679",
        "reminders" : [
            <date1>, <date2>,...
        ]
    },
    "active" : true
}
```

# Choose Wisely

- The nature of data is constantly changing
  - Dynamic content demands dynamic data storage techniques
  - Massive scaling
- Know your data before you choose your implementation
  - SQL/RDBMS is a better choice in many situations
- Consider implementation characteristics