Scaling with HiveDB

Project Genesis

- Cafepress.com Product Catalog
  - Hundreds of Millions of Products
  - Millions of new products every week
  - Accelerating growth

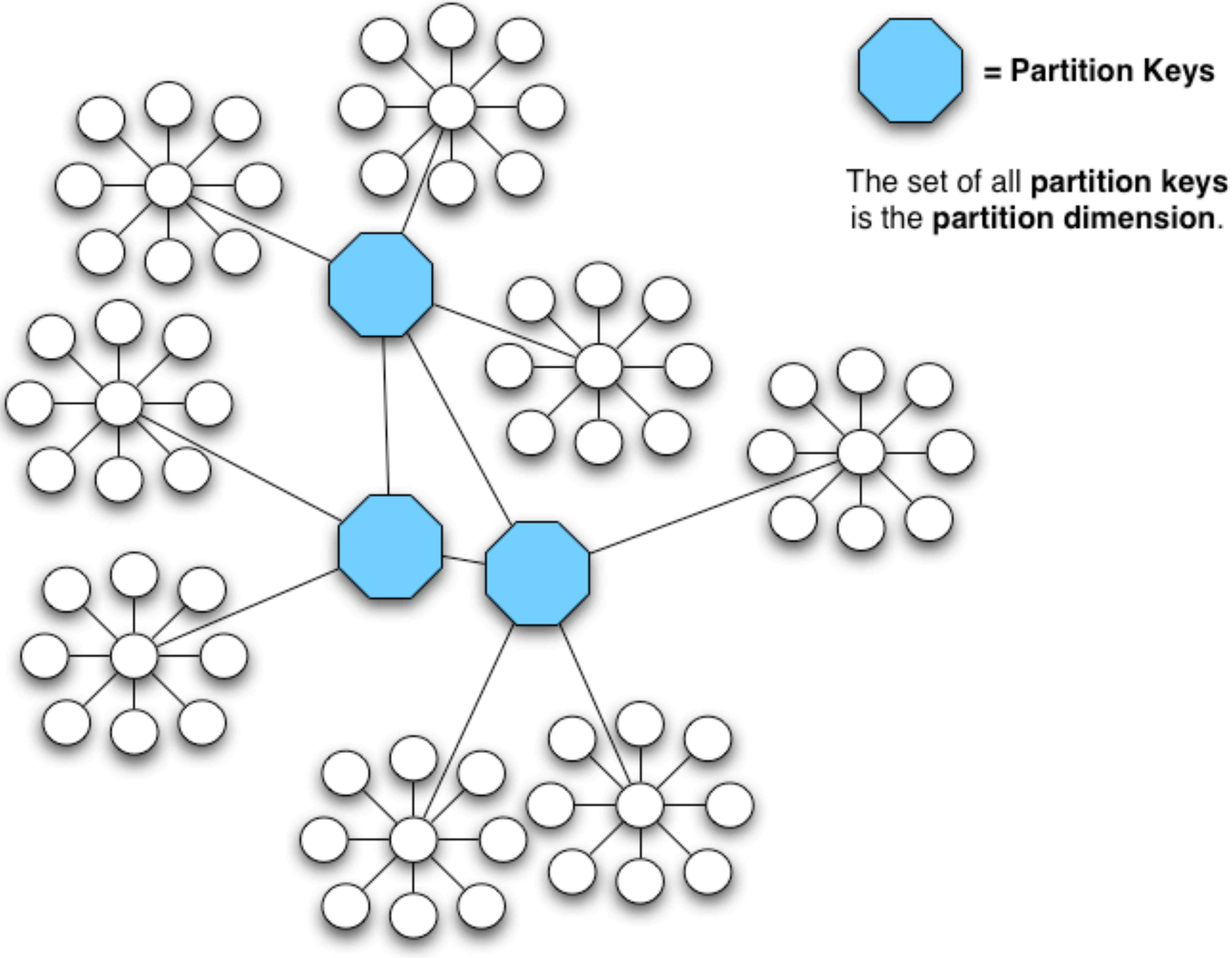
Enter Jeremy and HiveDB

Proven Scaling

Our Requirements

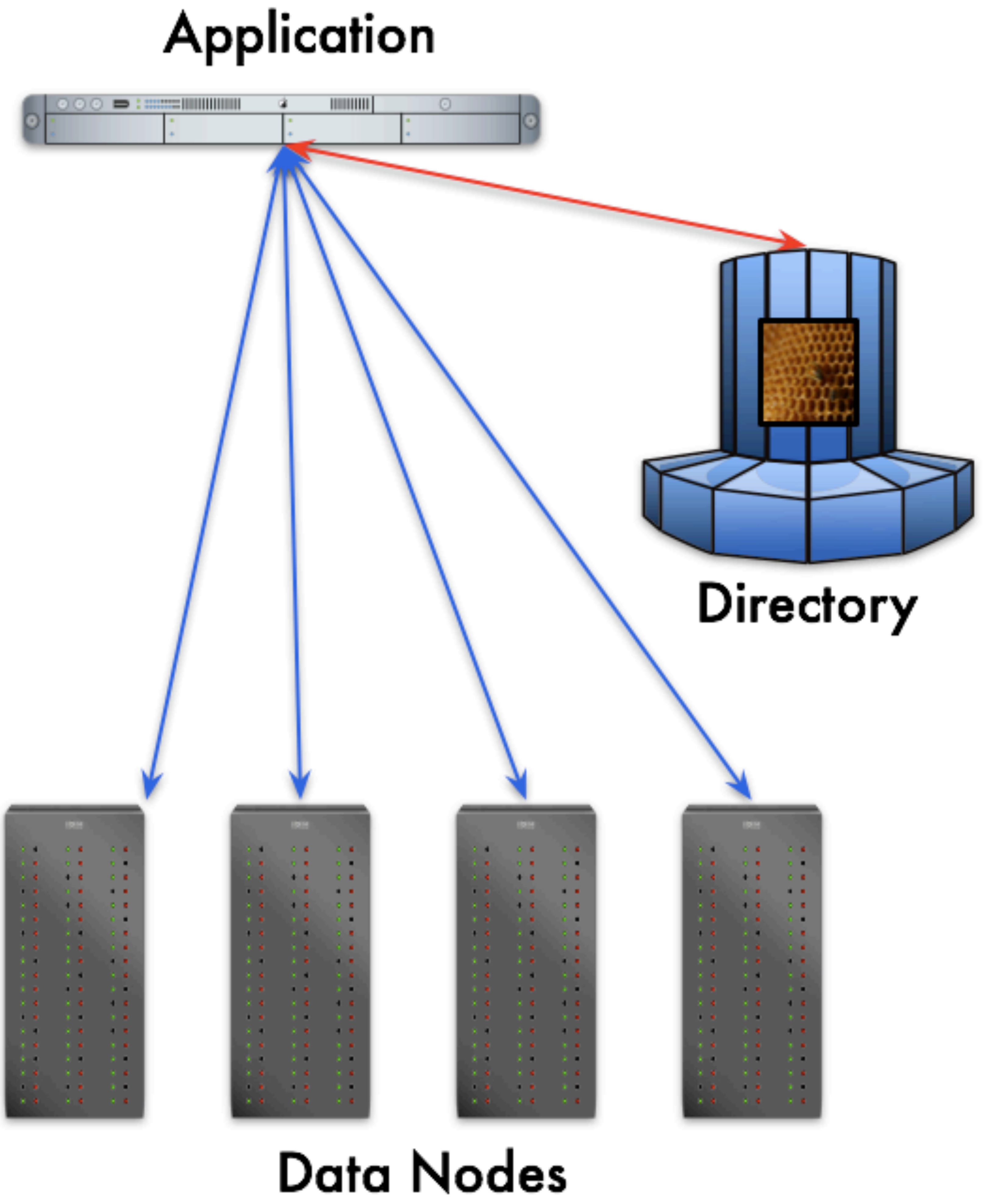
- OLTP Optimized
- Constant response time is more important than low latency
- Related sets vary wildly in size
- Growth hotspots
- Usage hotspots

Partition by key


= Partition Keys
The set of all **partition keys** is the **partition dimension**.

Directory

- No broadcasting
- No re-partitioning
- Easy to relocate records
- Easy to add capacity


Application
Directory
Data Nodes

Disadvantages

- Intelligence moved from the database to the application (Queries have to be planned and indexed)
- Can’t join across partitions
- NO OLAP! (We consider this a benefit.)
- Directory is a bottleneck

Original Design

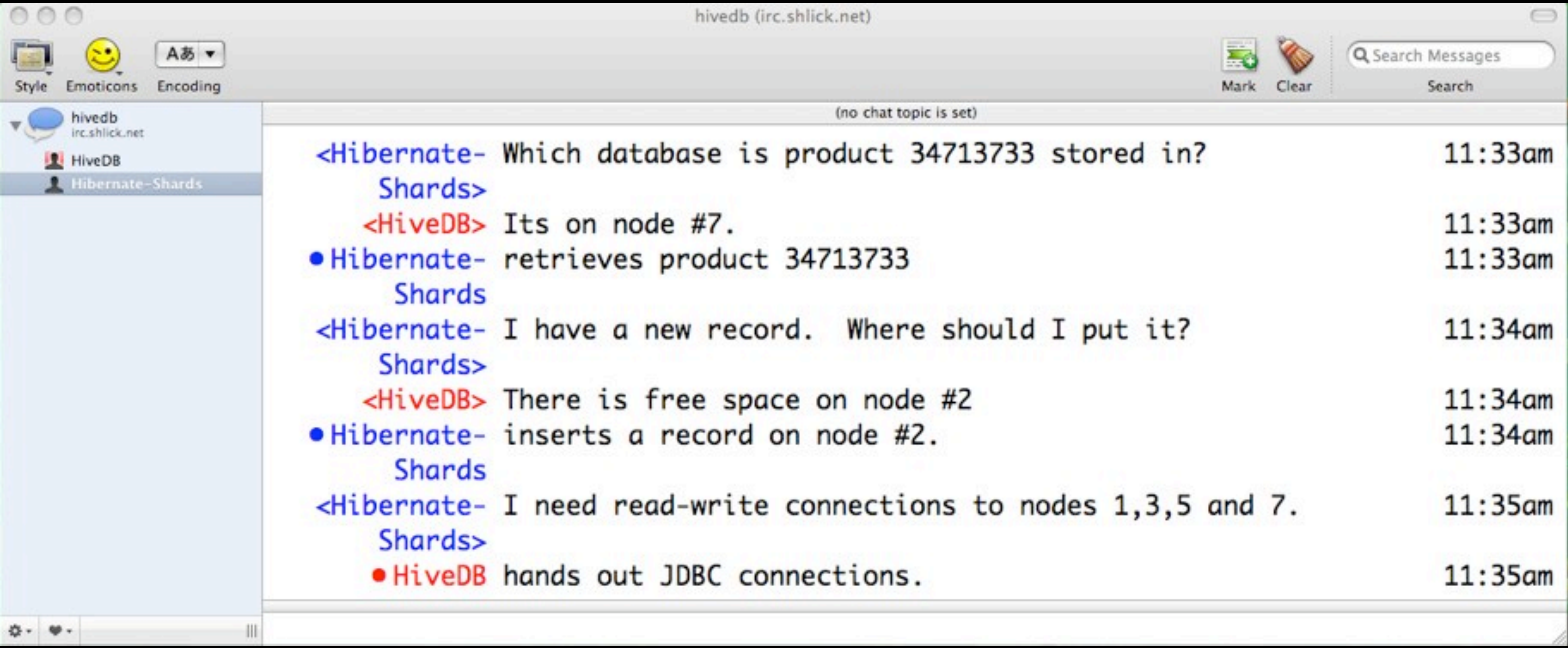
- Smallest possible implementation
- HiveDB was just a JDBC Gatekeeper for applications.

Development Complexity Problem

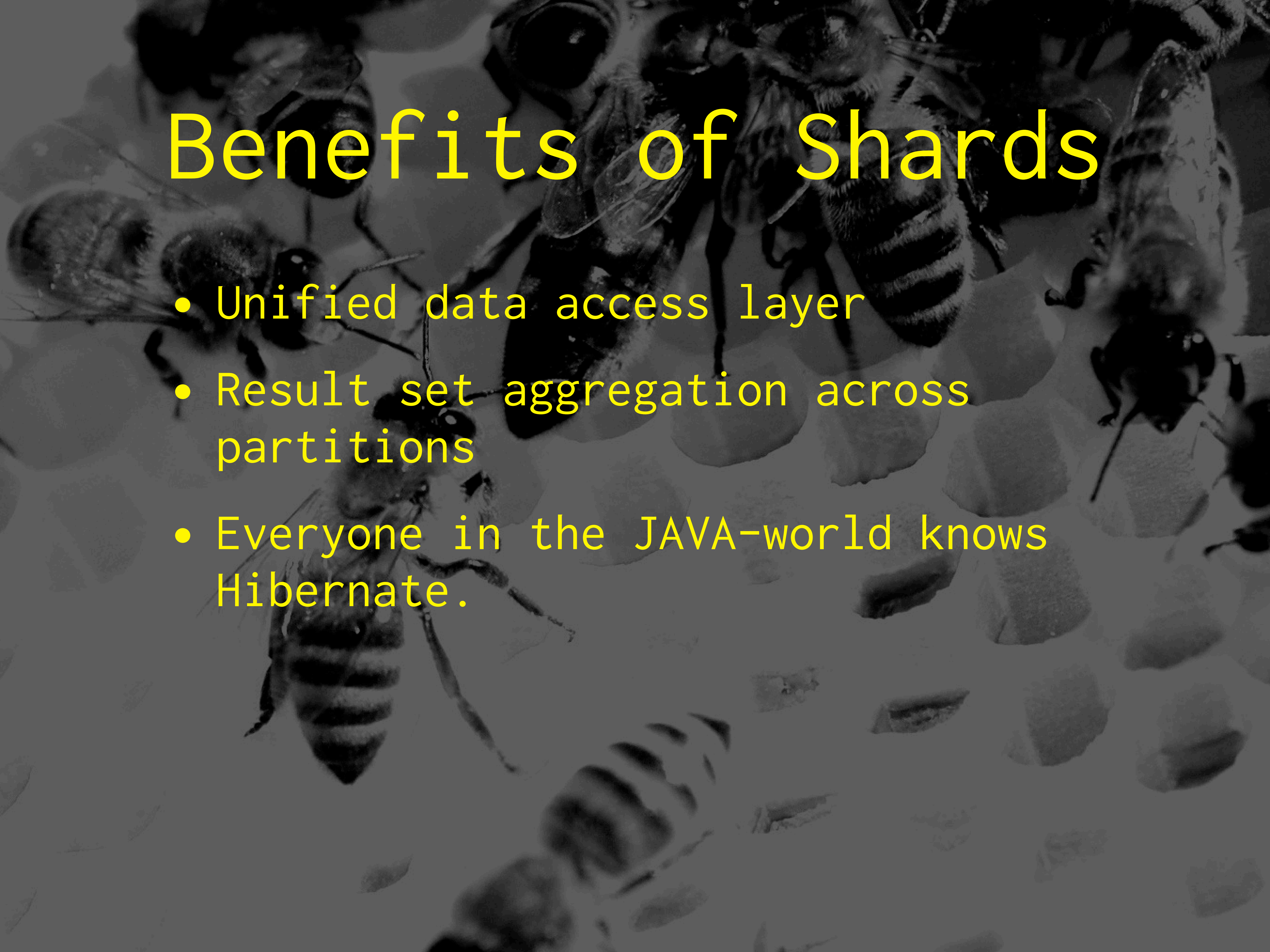
- You have to maintain synchronization between the directory and the data nodes.
- Lots of code for simple operations
- Data access objects have to be re-implemented

Enter Hibernate Shards

- Partitioned Hibernate from Google
- Why did we write this thing again?
- Oh wait, you have to tell it how to look things up...we’re good at that.



```
 <Hibernate-Shards> Which database is product 34713733 stored in?    11:33am
           <HiveDB> Its on node #7.                                  11:33am
 • Hibernate-Shards retrieves product 34713733                       11:33am
 <Hibernate-Shards> I have a new record.  Where should I put it?     11:34am
           <HiveDB> There is free space on node #2                   11:34am
 • Hibernate-Shards inserts a record on node #2.                     11:34am
 <Hibernate-Shards> I need read-write connections to nodes 1,3,5 and 7.  11:35am
           • HiveDB hands out JDBC connections.                      11:35am
```



Benefits of Shards

- Unified data access layer
- Result set aggregation across partitions
- Everyone in the JAVA-world knows Hibernate.

Show don’t tell!

Competitive Landscape

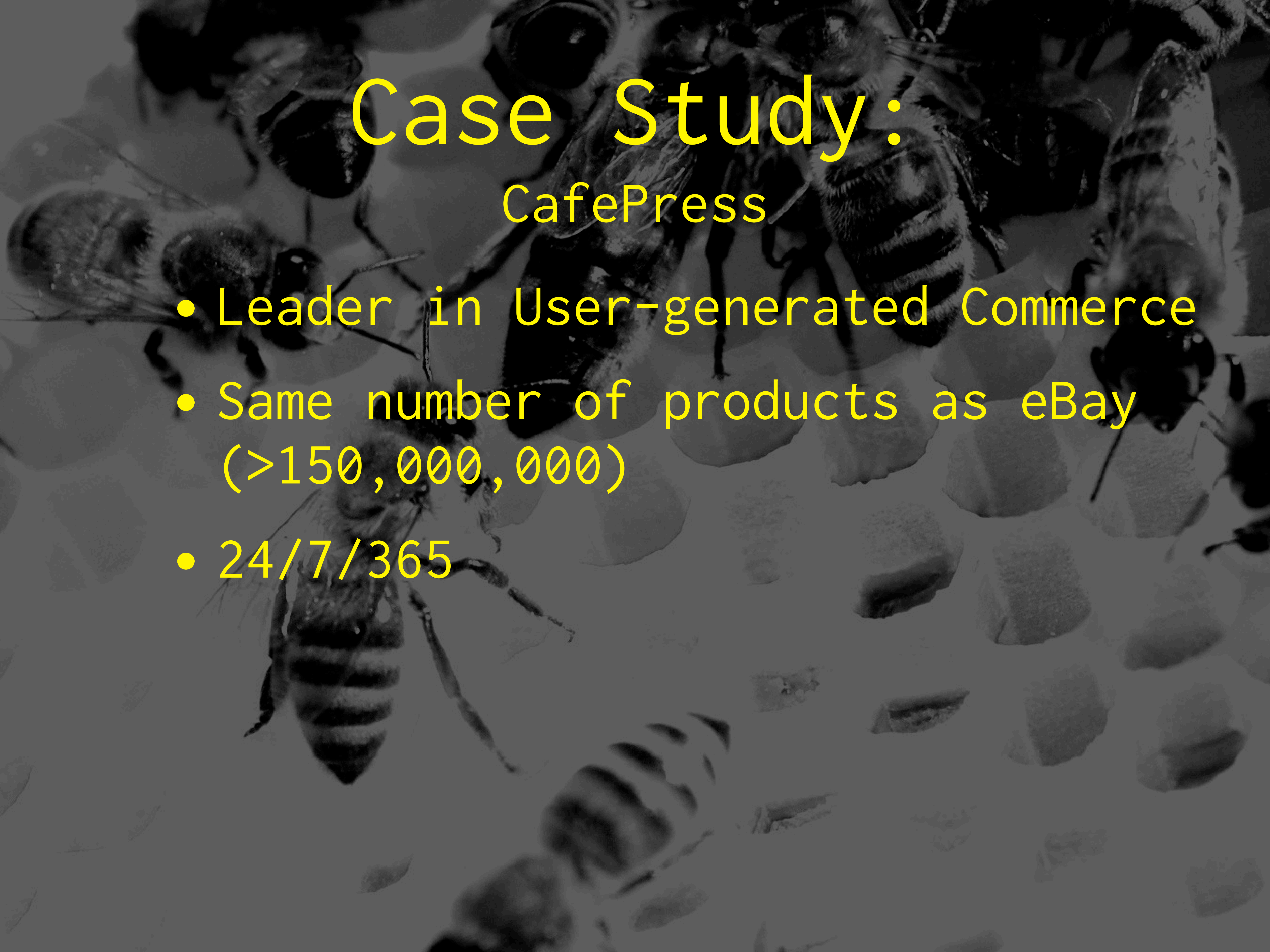
- Clustered Relational Databases
- Non-relational Structured Databases
- Non-relational, Unstructured Storage

Competitive Landscape

Clustered	Non-relational	Unstructured
Oracle RAC MS SQL Server MySQL Cluster DB2 Teradata (OLAP) HiveDB	Hypertable HBase CouchDB SimpleDB	Hadoop MogileFS S3

Competitive Landscape

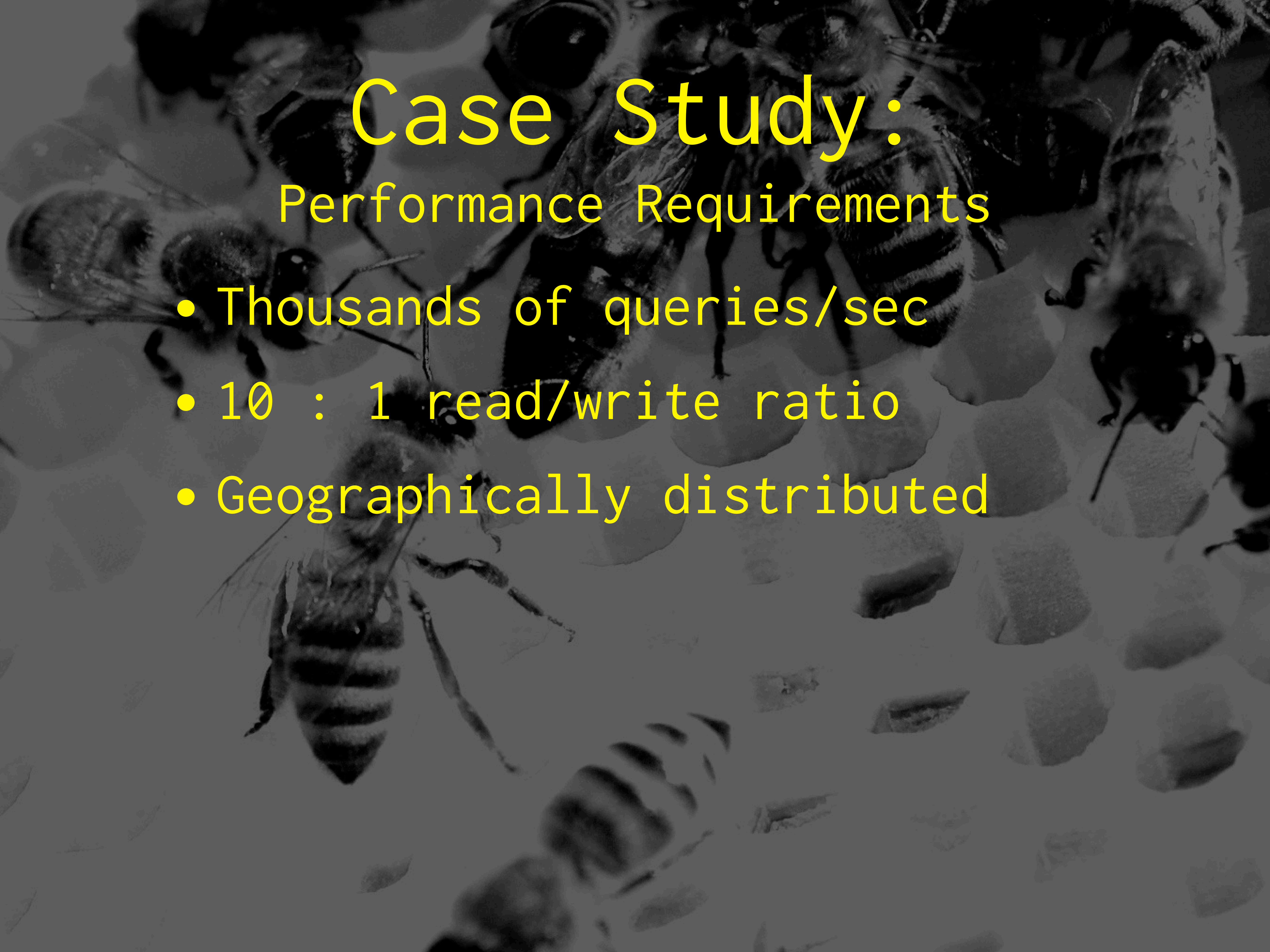
	Storage	Interface	Partitioning	Expansion	Node Types	Maturity
Oracle RAC	Shared	SQL	Transparent	No downtime	Identical	7 years
MySQL Cluster	Memory / Local	SQL	Transparent	Requires Restart	Mixed	3 years
Hypertable	Local	HQL	Transparent	No downtime	Mixed	? (released 2/08)
DB2	Local	SQL	Fixed Hash	Degraded Performance	Identical	3 years (25 years total)
HiveDB	Local	SQL	Key-based Directory	No downtime	Mixed	18 months (+13 years!)



Case Study:

CafePress

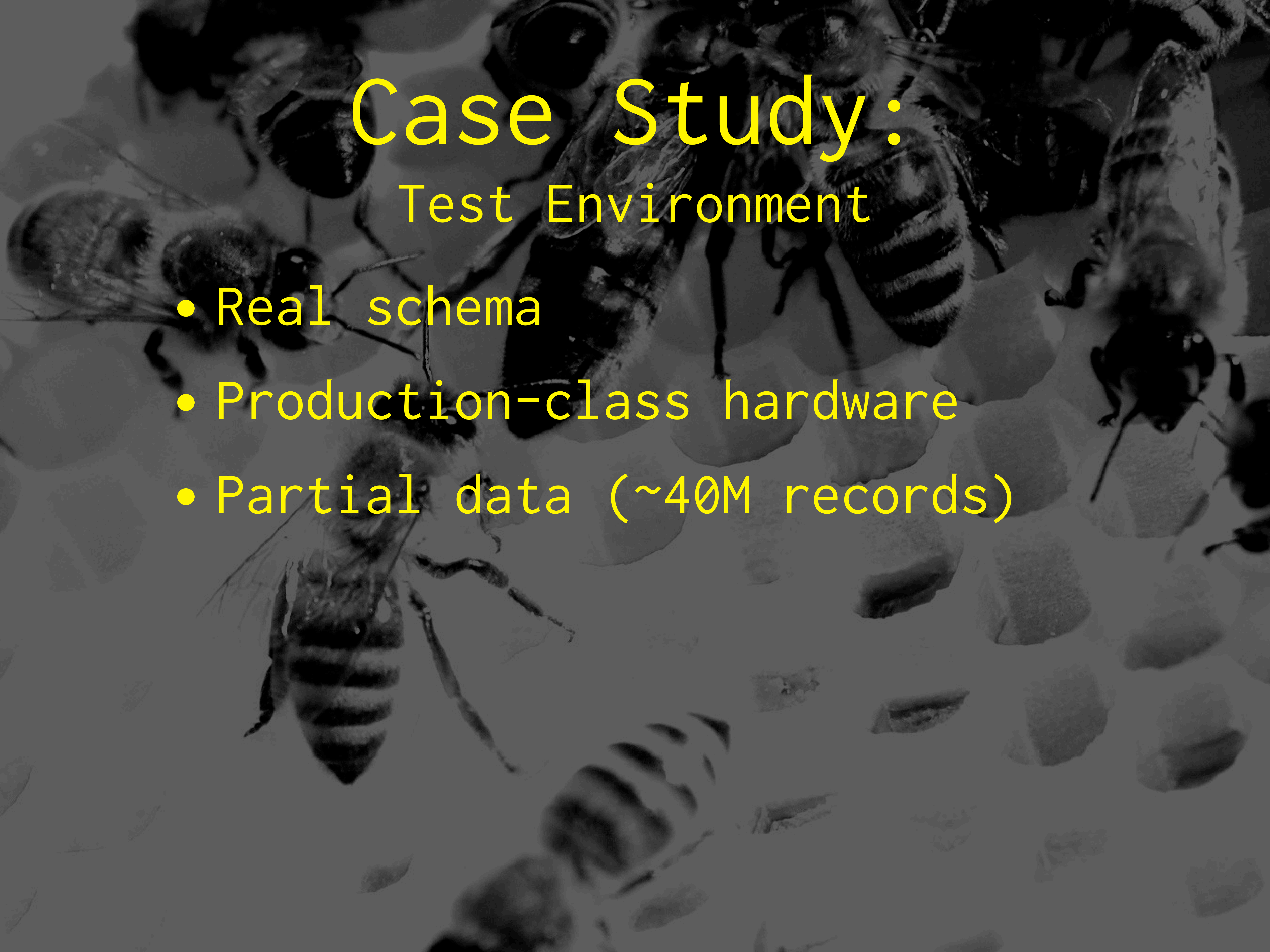
- Leader in User-generated Commerce
- Same number of products as eBay (>150,000,000)
- 24/7/365



Case Study:

Performance Requirements

- Thousands of queries/sec
- 10 : 1 read/write ratio
- Geographically distributed



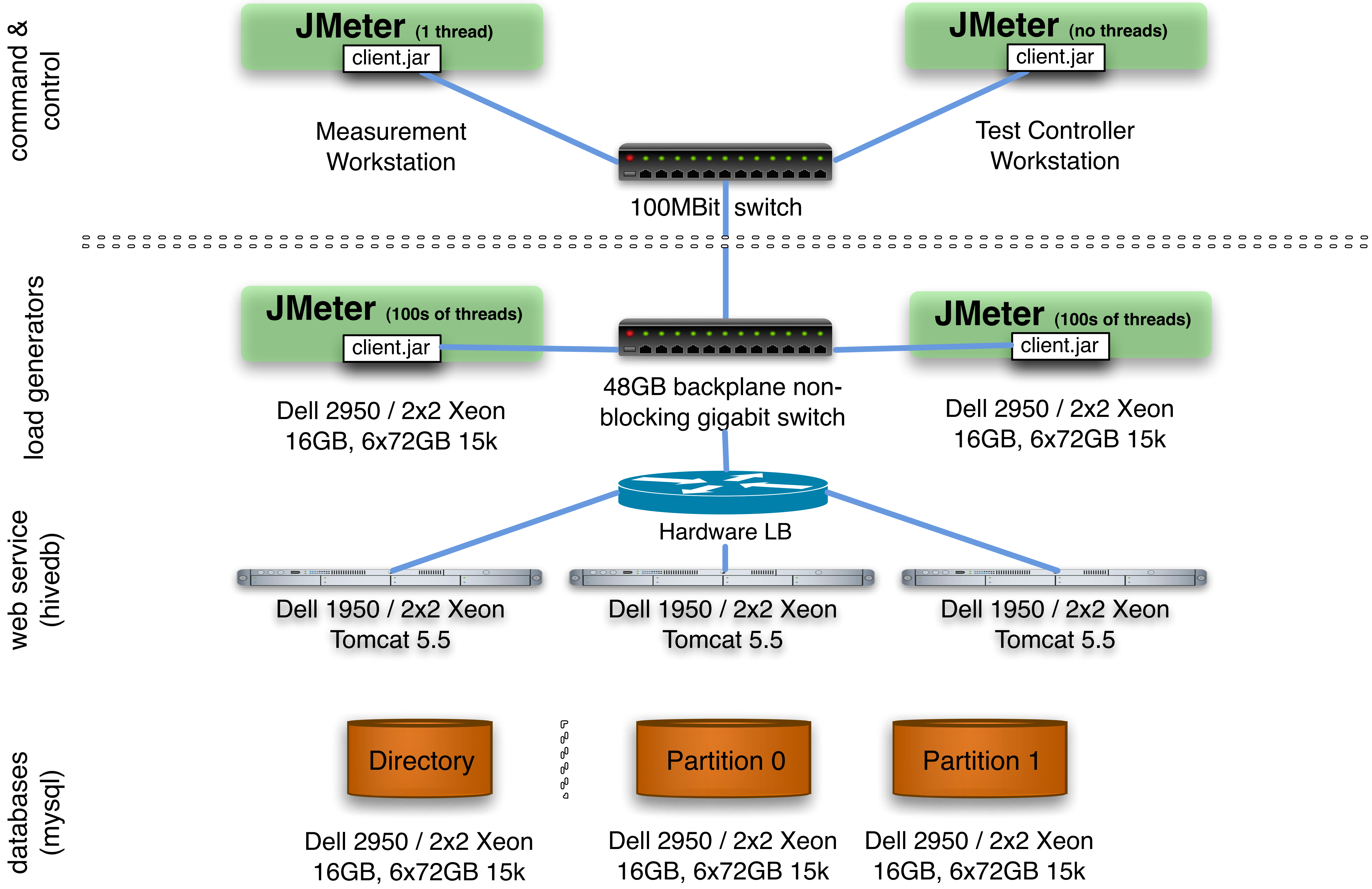
Case Study:

Test Environment

- Real schema
- Production-class hardware
- Partial data (~40M records)

CafePress HiveDB 2007

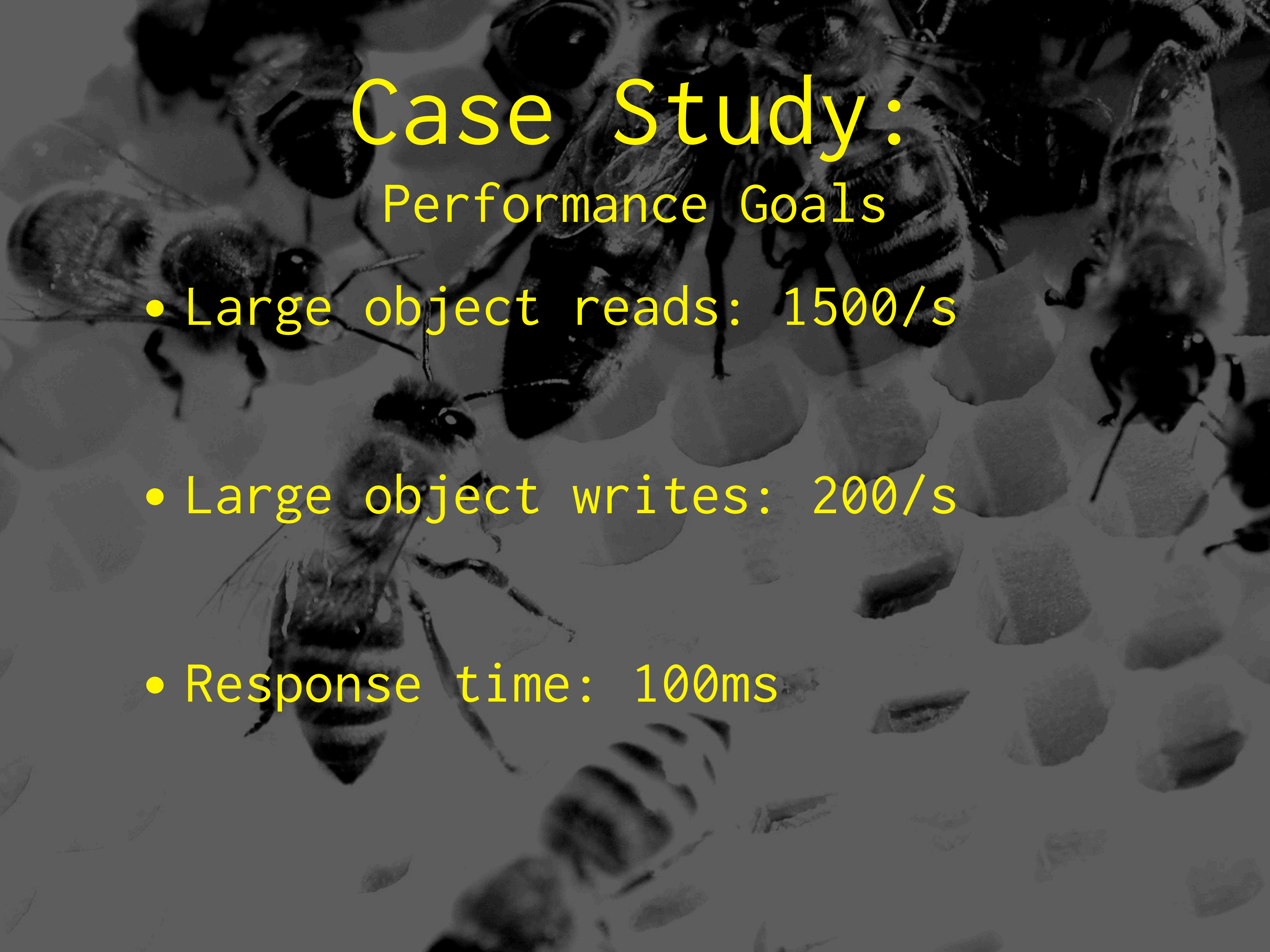
Performance Test Environment



command & control
JMeter (1 thread)
client.jar
Measurement Workstation
JMeter (no threads)
client.jar
Test Controller Workstation
100MBit switch
load generators
JMeter (100s of threads)
client.jar
Dell 2950 / 2x2 Xeon 16GB, 6x72GB 15k
48GB backplane non-blocking gigabit switch
JMeter (100s of threads)
client.jar
Dell 2950 / 2x2 Xeon 16GB, 6x72GB 15k
web service (hivedb)
Hardware LB
Dell 1950 / 2x2 Xeon Tomcat 5.5
Dell 1950 / 2x2 Xeon Tomcat 5.5
Dell 1950 / 2x2 Xeon Tomcat 5.5
databases (mysql)
Directory
Dell 2950 / 2x2 Xeon 16GB, 6x72GB 15k
Partition 0
Dell 2950 / 2x2 Xeon 16GB, 6x72GB 15k
Partition 1
Dell 2950 / 2x2 Xeon 16GB, 6x72GB 15k

jmccarthy@cafepress.com

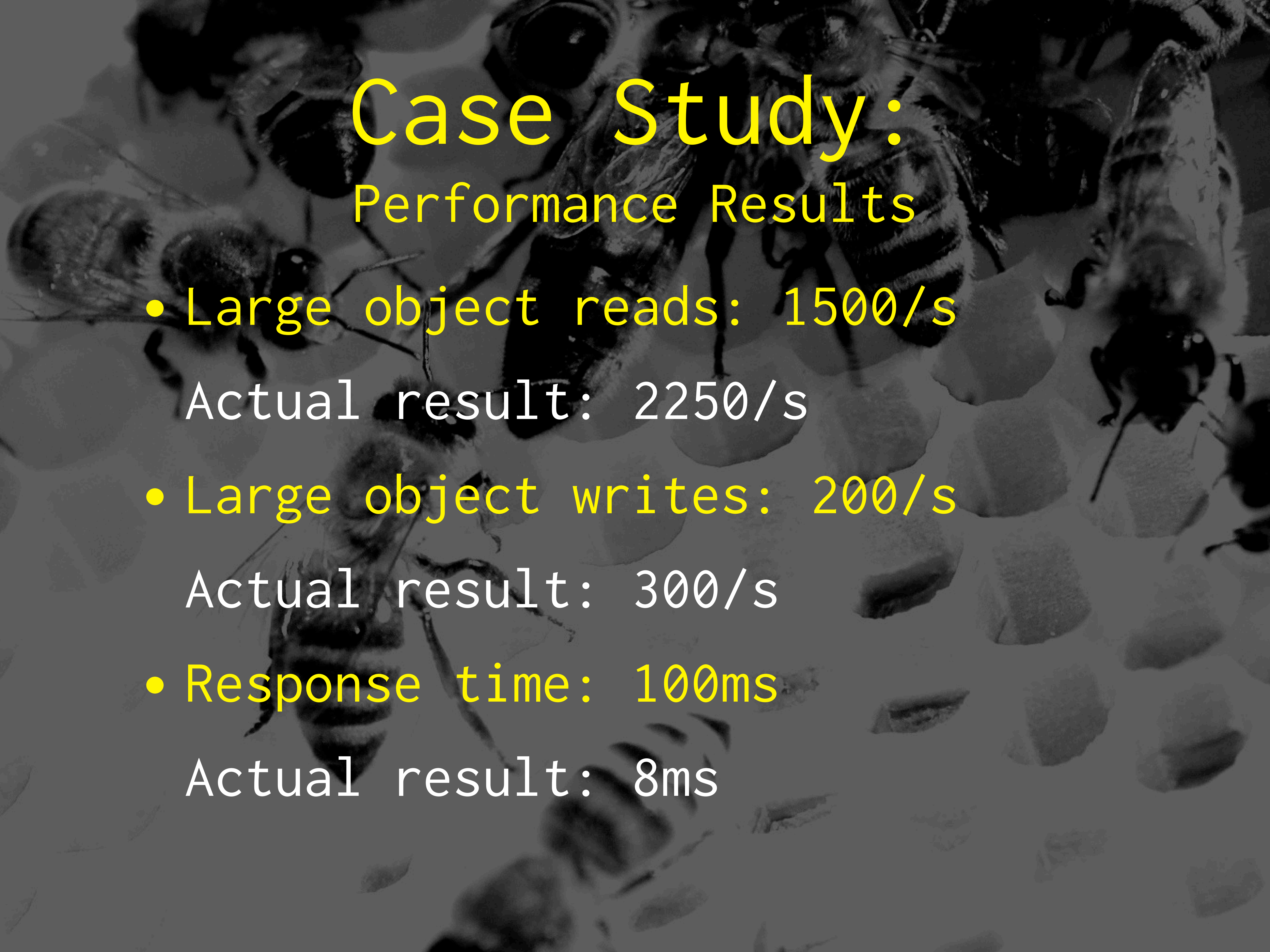
Modified on April 09 2007



Case Study:

Performance Goals

- Large object reads: 1500/s
- Large object writes: 200/s
- Response time: 100ms



Case Study:

Performance Results

- Large object reads: 1500/s

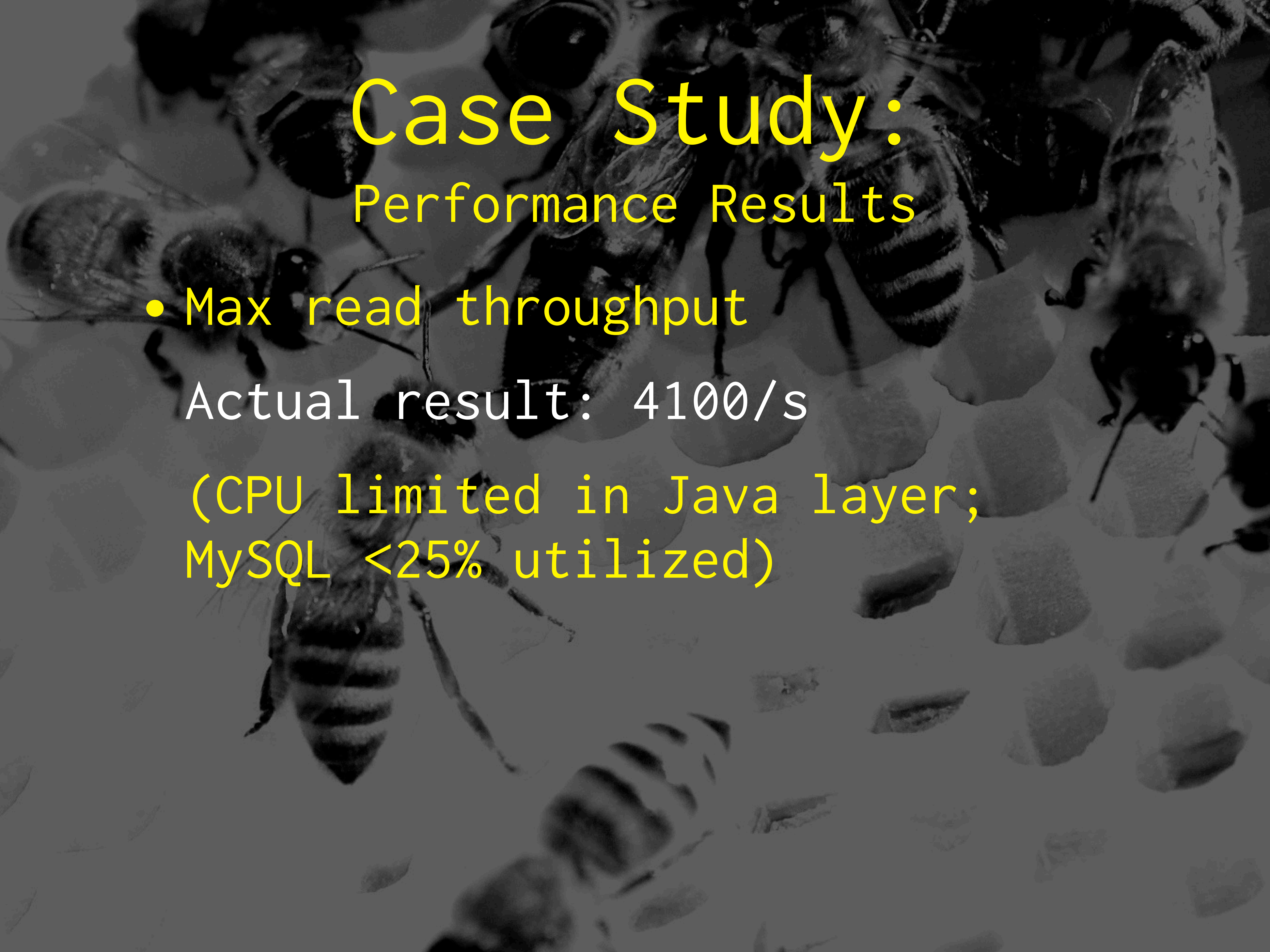
  Actual result: 2250/s

- Large object writes: 200/s

  Actual result: 300/s

- Response time: 100ms

  Actual result: 8ms



Case Study:

Performance Results

- Max read throughput

Actual result: 4100/s

(CPU limited in Java layer; MySQL <25% utilized)

Case Study:

Organizational Results

- Billions of queries served
- Highest DB uptime at CafePress
- Hundreds of millions of updates performed

High Availability & Replication

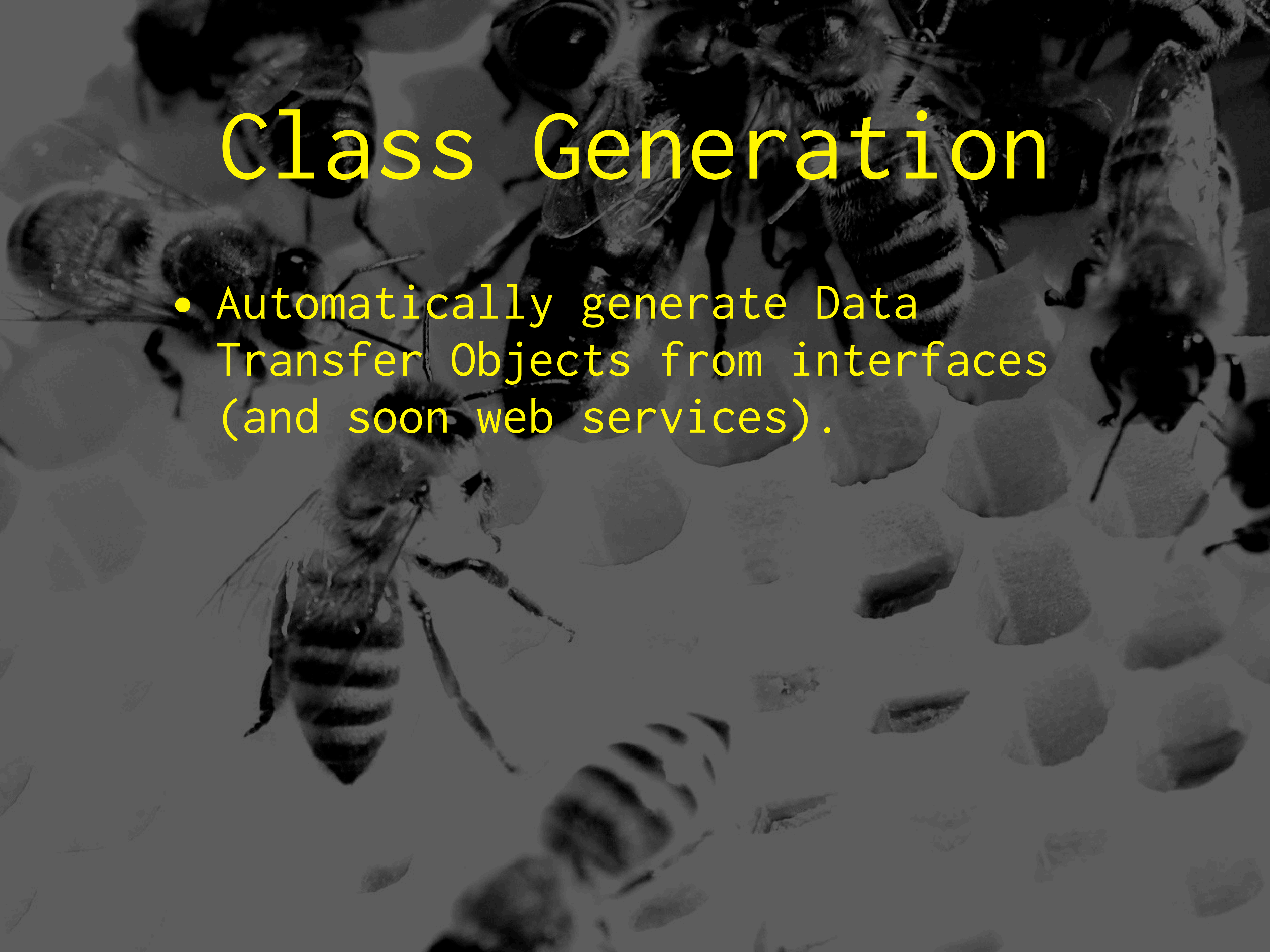
- We don’t specify a fail over strategy
- We delegate to MySQL replication

Non-JAVA Deployment Options

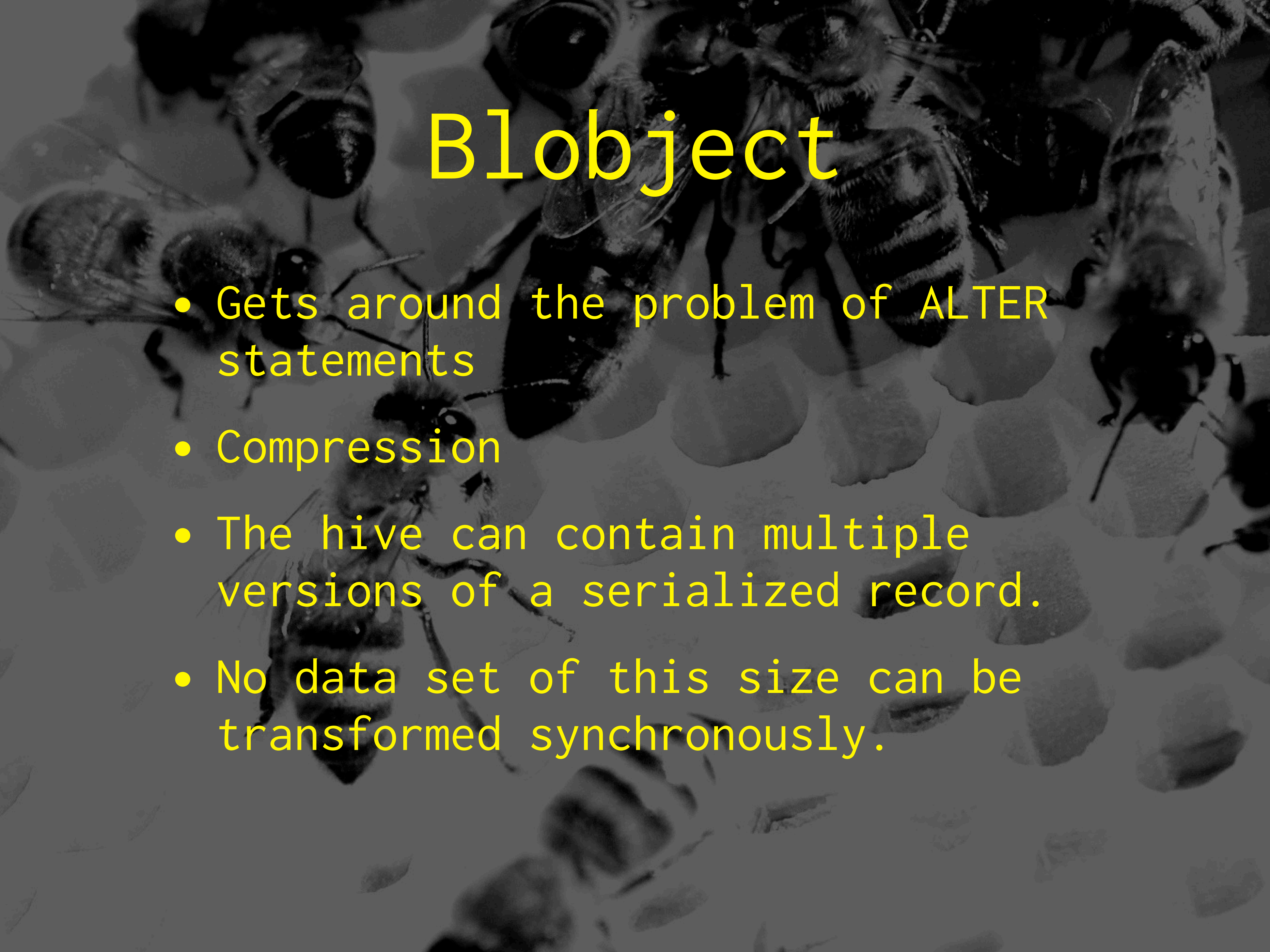
- Web service
- JVM Dynamic Languages

HiveDB Accessories

Class Generation



- Automatically generate Data Transfer Objects from interfaces (and soon web services).



Blobject

- Gets around the problem of ALTER statements
- Compression
- The hive can contain multiple versions of a serialized record.
- No data set of this size can be transformed synchronously.

Features Teaser

- We're taking over HA...you're still on your own for replication.
- Generated Web Services
- Monitoring & RRD stats (with graphs!)
- Query/transform tool
- Record migration & balancing tools

Contributing

- Post to the mailing list

  http://groups.google.com/group/hivedb-dev

- Comment on our site

  http://www.hivedb.org

- File a bug

  http://hivedb.lighthouseapp.com

- Submit a patch / pull request

  ```
  git clone git://github.com/britt/hivedb.git
  ```

Photo Credits

- http://www.flickr.com/photos/7362313@N07/1240245941/sizes/o
- http://www.flickr.com/photos/99287245@N00/2229322675/sizes/o