GIGASPACES

WRITE ONCE.
SCALE ANYWHERE.

GigaSpaces Technologies

Yes, SQL!

QCon

Uri Cohen

```
> SELECT * FROM qcon2010.speakers WHERE
  name=‘Uri Cohen’
+----------------------------------------------------+
| Name       | Company      | Role              | Twitter  |
+----------------------------------------------------+
| Uri Cohen | GigaSpaces | Product Manager | @uri1803 |
+----------------------------------------------------+
```

```
> db.speakers.find({name:”Uri Cohen”})
{
   “name”:”Uri Cohen”,
   “company”: {
                   name:”GigaSpaces”,
                   products:[“XAP”, “IMDG”]
                   domain: “In memory data grids”
               }
   “role”:”product manager”,
   “twitter”:”@uri1803”
}
```

GIGASPACES

Agenda

- SQL
  - What it is and isn't good for
- NoSQL
  - Motivation & Main Concepts of Modern Distributed Data Stores
  - Common interaction models
    - Key/Value, Column, Document
    - NOT consistency and distribution algorithms
- One Data Store, Multiple APIs
  - Brief intro to GigaSpaces
  - Key/Value challenges
  - SQL challenges: Add-hoc querying, Relationships (JPA)

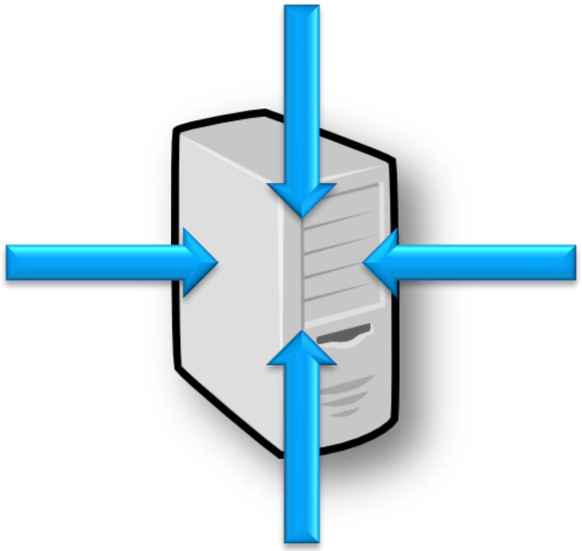
A FEW (MORE) WORDS ABOUT SQL

SQL

(Usually) Centralized

→ Transactional, consistent

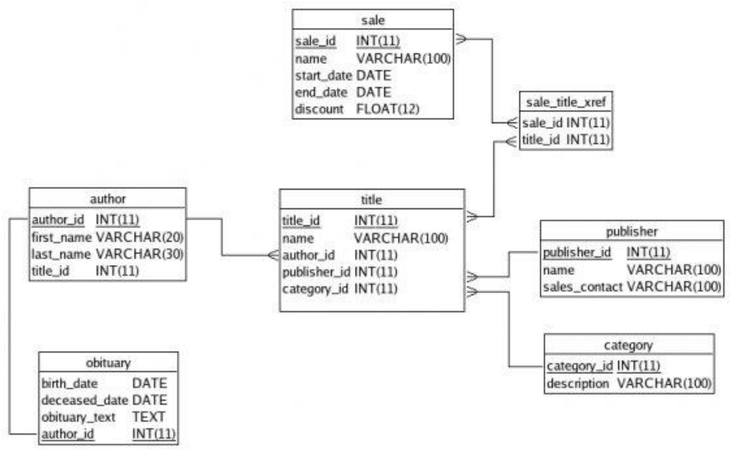
→ Hard to Scale



SQL

Static, normalized data schema

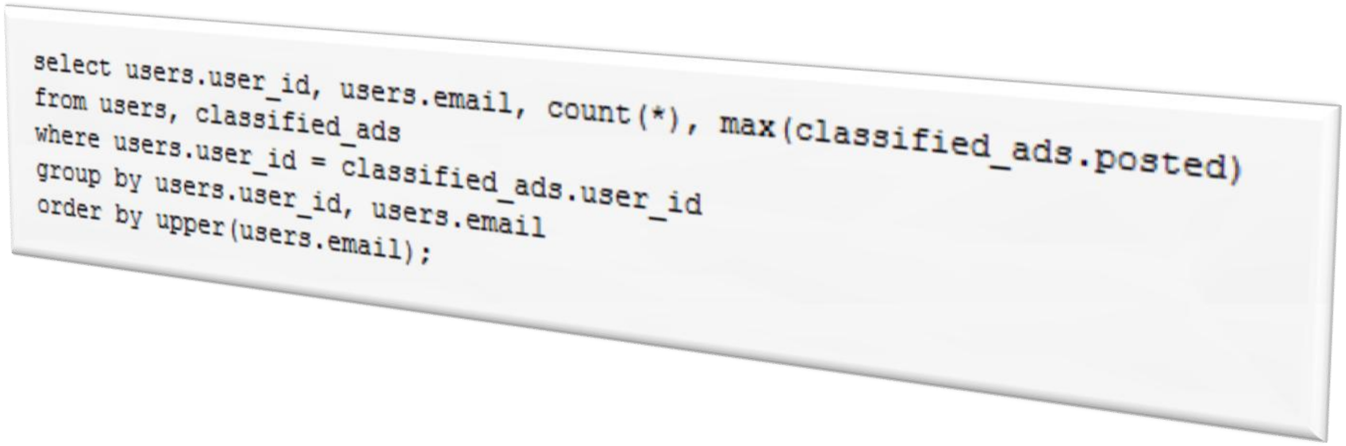
- Don’t duplicate, use FKs



SQL

Add hoc query support

→ Model first, query later



```
select users.user_id, users.email, count(*), max(classified_ads.posted)
from users, classified_ads
where users.user_id = classified_ads.user_id
group by users.user_id, users.email
order by upper(users.email);
```

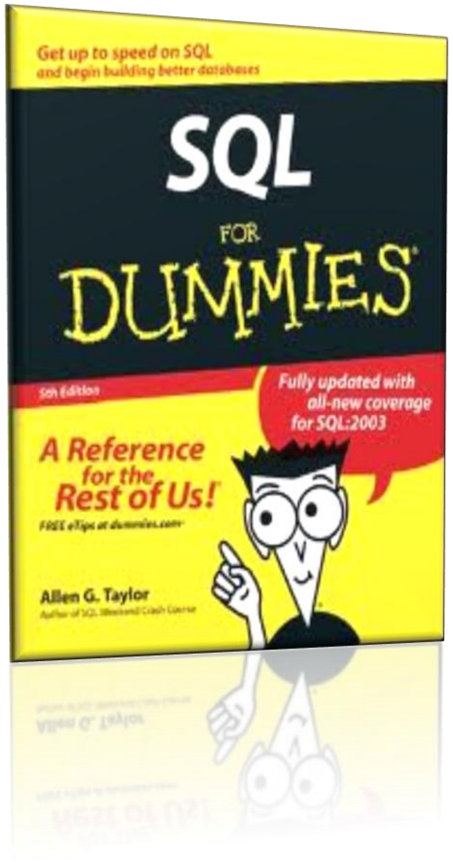
GIGASPACES

SQL

Standard

→ Well known

→ Rich ecosystem


Get up to speed on SQL and begin building better databases
SQL FOR DUMMIES
5th Edition
Fully updated with all-new coverage for SQL:2003
A Reference for the Rest of Us!
FREE eTips at dummies.com
Allen G. Taylor

(BRIEF) NOSQL RECAP

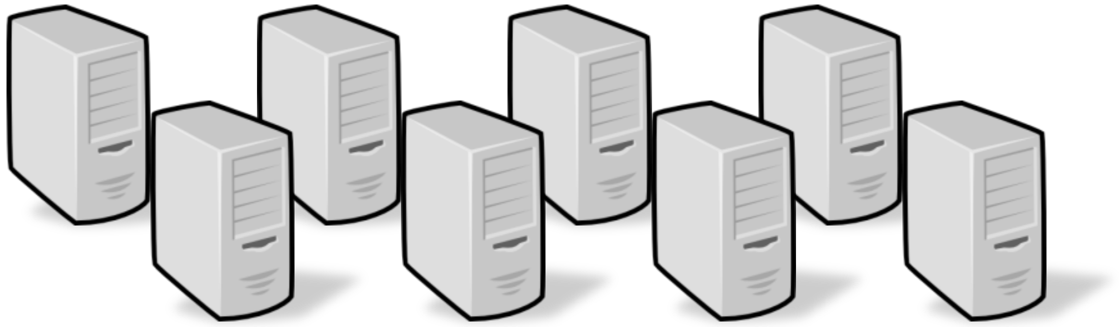
NoSql (or a Naive Attempt to Define It)

A loosely coupled collection of **non-relational data stores**

NoSql (or a Naive Attempt to Define It)

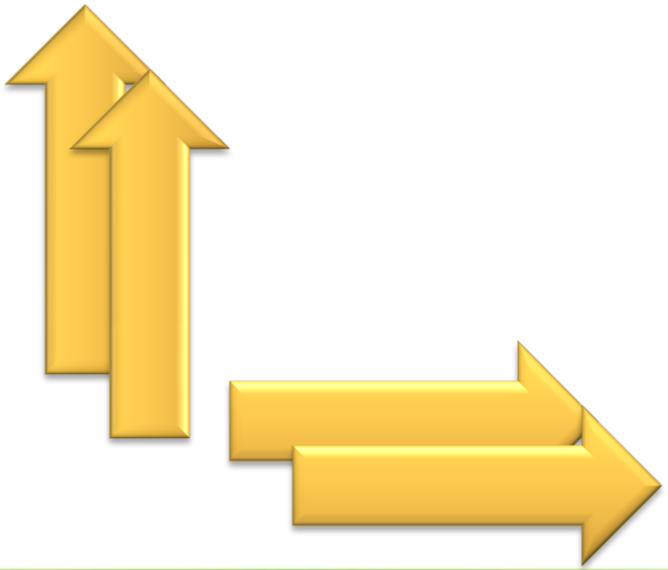
(Mostly)

d i s t r i b u t e d



NoSql (or a Naive Attempt to Define It)

scalable (Up & Out)



NoSql (or a Naive Attempt to Define It)

Not (always) ACID

- **BASE anyone?**

Why Now?

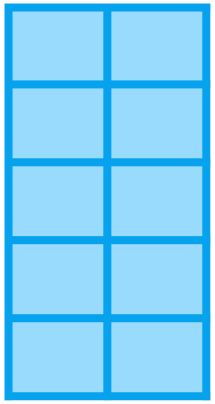
Timing is everything…

- Exponential Increase in data & throughput
- Non or semi structured data that changes frequently

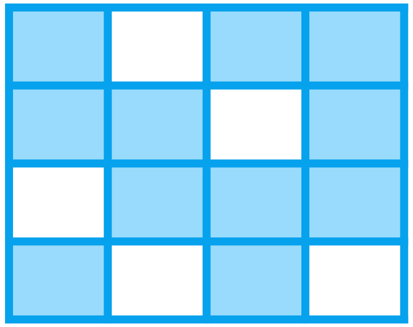
GIGASPACES

A Universe of Data Models

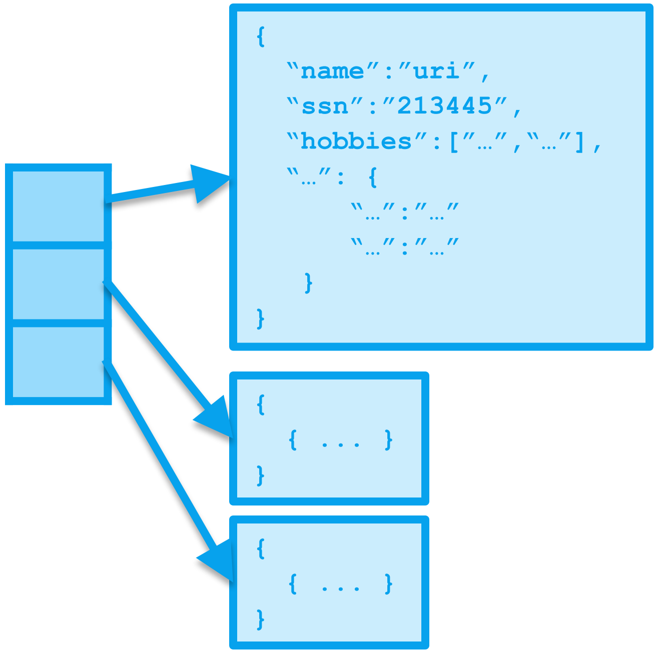
Key / Value



Column



Document



```
{
  "name":"uri",
  "ssn":"213445",
  "hobbies":["…","…"],
  "…": {
       "…":"…"
       "…":"…"
   }
}
```

```
{
  { ... }
}
```

```
{
  { ... }
}
```

GIGASPACES

Key/Value

- Have the key? Get the value
  - That's about it when it comes to querying
  - Map/Reduce (sometimes)
  - Good for
    - cache aside (e.g. Hibernate 2nd level cache)
    - Simple, id based interactions (e.g. user profiles)
- In most cases, values are Opaque

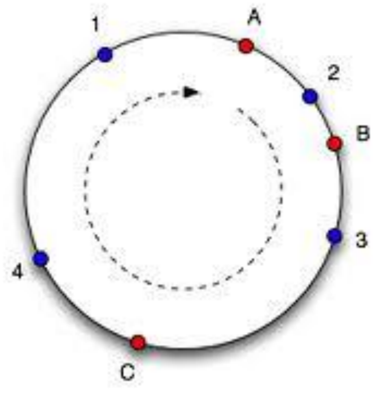
K1	V1
K2	V2
K3	V3
K4	V1

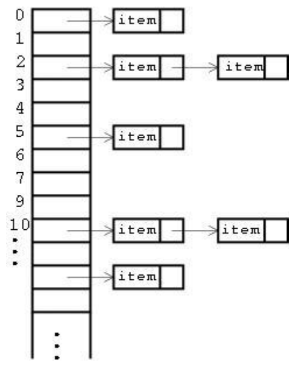
Key/Value

Scaling out is relatively easy

(just hash the keys)

- Some will do that automatically for you
- Fixed vs. consistent hashing





Key/Value

- **Implementations:**
  - Memcached, Redis, Riak
  - In memory data grids (mostly Java-based) started this way
    - GigaSpaces, Oracle Coherence, WebSphere XS, JBoss Infinispan, etc.

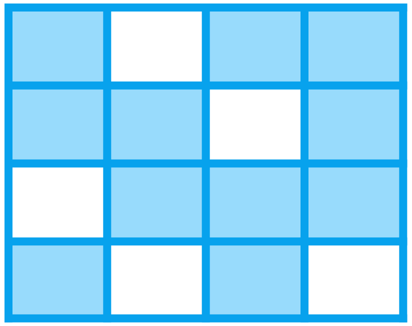
Column Based

H·BASE

Cassandra

amazon web services™

SimpleDB

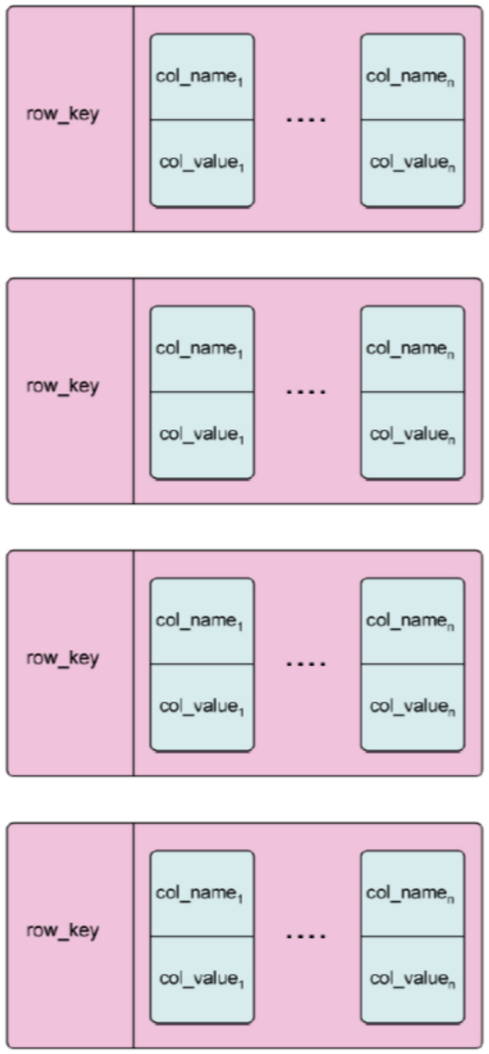


Column Based

- Mostly derived from Google's BigTable / Amazon Dynamo papers
- One giant table of rows and columns
  - Column == pair (name and a value, sometimes timestamp)
  - Each row can have a different number of columns
  - Table is sparse:
    `(#rows) × (#columns) ≥ (#values)`

Column Based

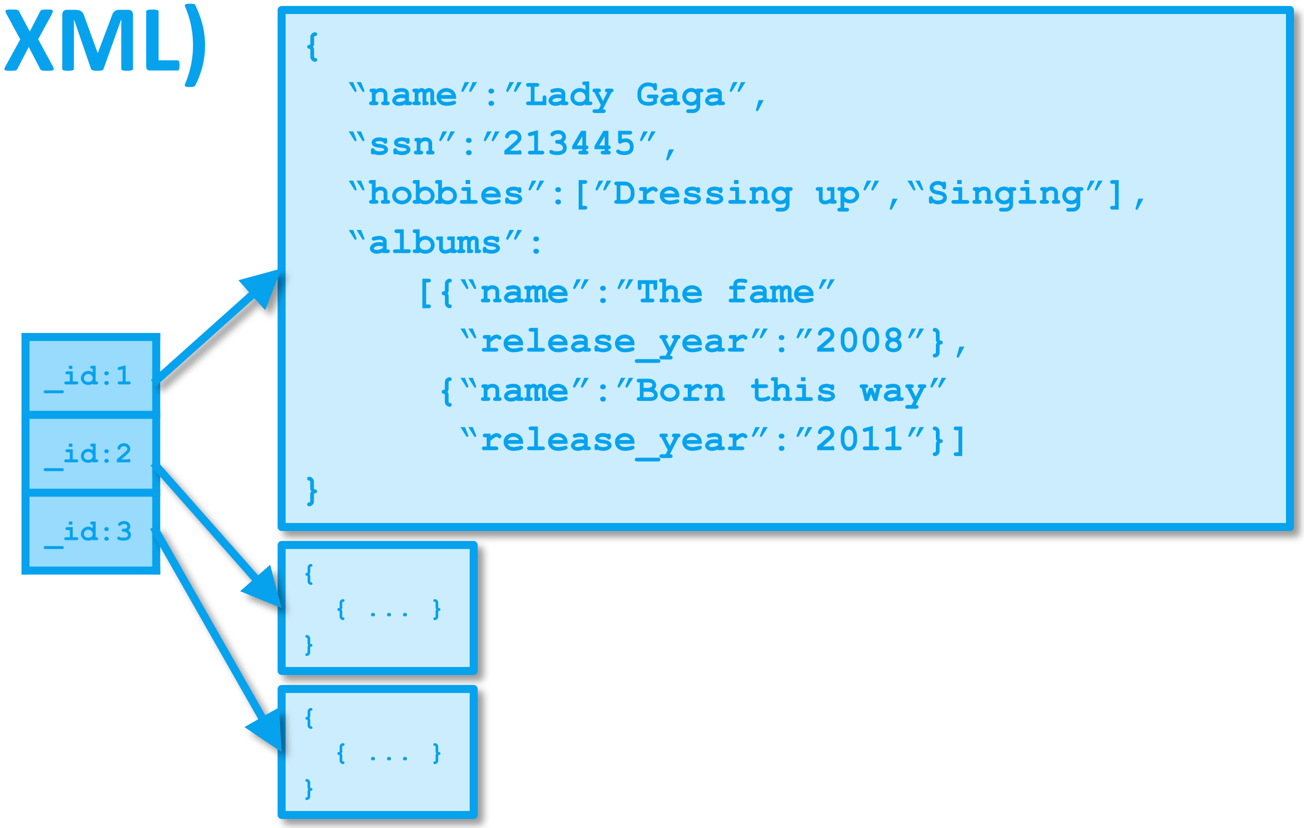
- Query on row key
  - Or column value (aka secondary index)
- Good for a constantly changing, (albeit flat) domain model



Document

Think JSON

(or BSON, or XML)



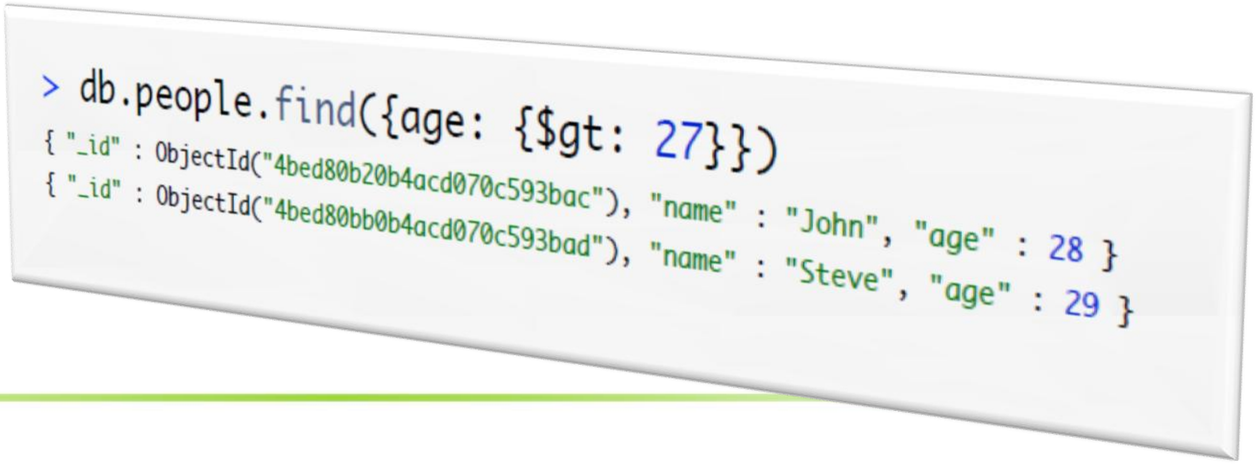
```
{
  "name":"Lady Gaga",
  "ssn":"213445",
  "hobbies":["Dressing up","Singing"],
  "albums":
      [{"name":"The fame"
        "release_year":"2008"},
       {"name":"Born this way"
        "release_year":"2011"}]
}
```

```
{
  { ... }
}
```

```
{
  { ... }
}
```

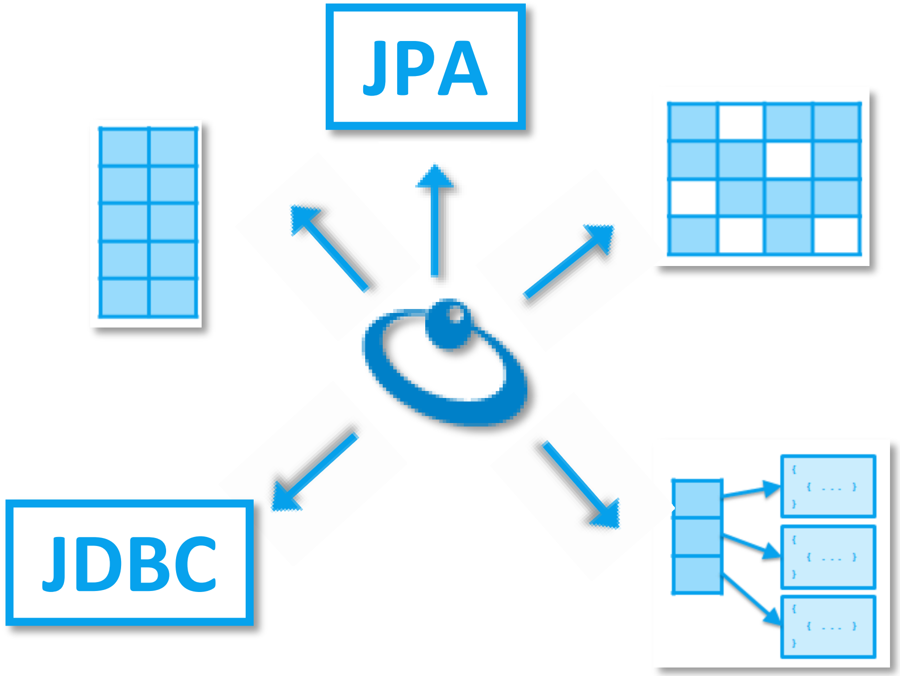
Document

- Model is not flat, data store is aware of it
  - Arrays, nested documents
- Better support for ad hoc queries
  - MongoDB excels at this
- Very intuitive model
- Flexible schema



```
> db.people.find({age: {$gt: 27}})
{ "_id" : ObjectId("4bed80b20b4acd070c593bac"), "name" : "John", "age" : 28 }
{ "_id" : ObjectId("4bed80bb0b4acd070c593bad"), "name" : "Steve", "age" : 29 }
```

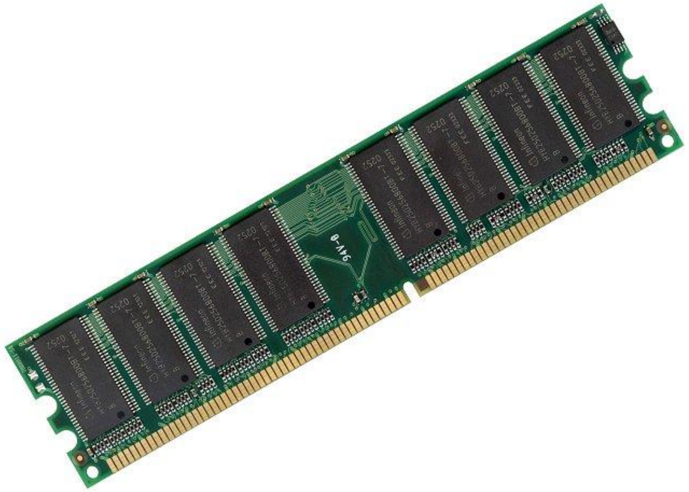
What if you didn't have to choose?



A Brief Intro to GigaSpaces

In Memory Data Grid

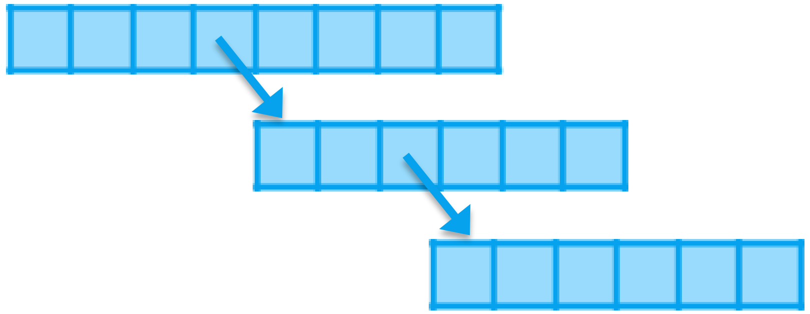
- With optional write behind to a secondary storage



A Brief Intro to GigaSpaces

Tuple based

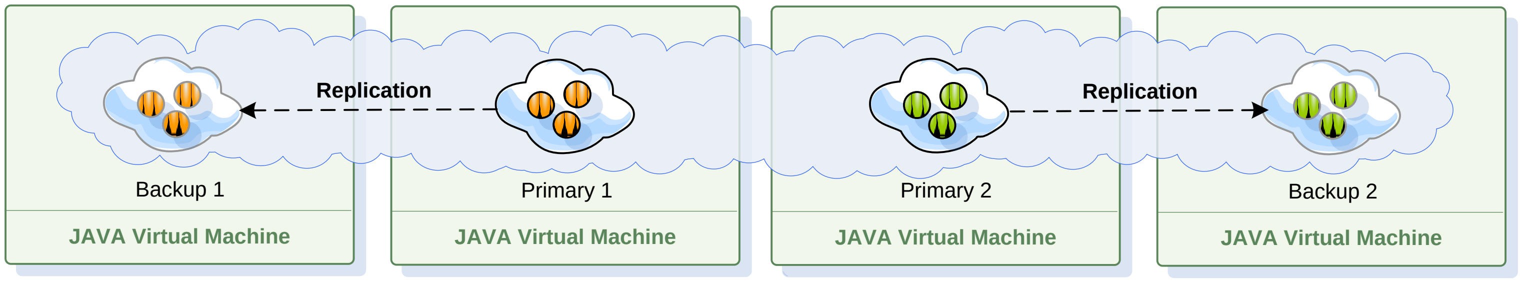
- **Aware of nested tuples (and soon collections)**
  - Document like
- **Rich querying and map/reduce semantics**



A Brief Intro to GigaSpaces

Transparent partitioning & HA

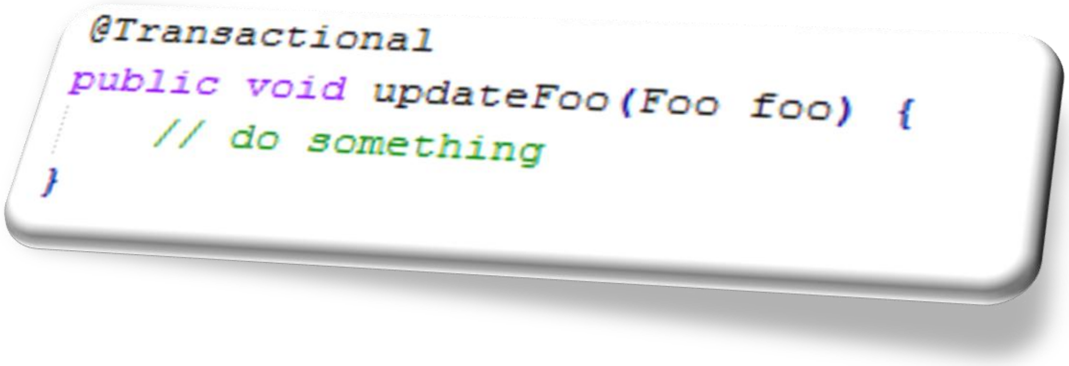
- Fixed hashing based on a chosen property



A Brief Intro to GigaSpaces

Transactional (Like, ACID)

- Local (single partition)
- Distributed (multiple partitions)



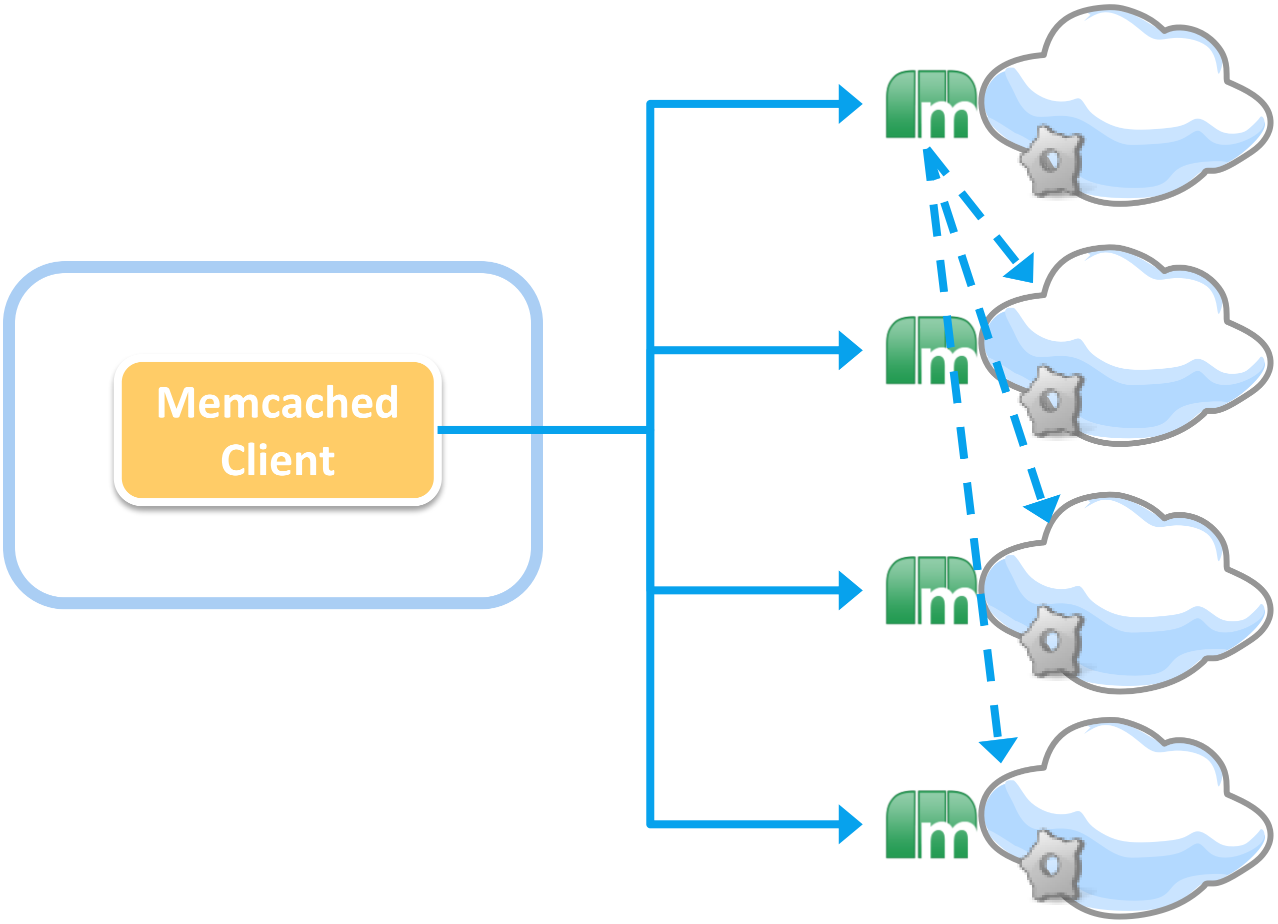
```
@Transactional
public void updateFoo(Foo foo) {
    // do something
}
```

Use the Right API for the Job

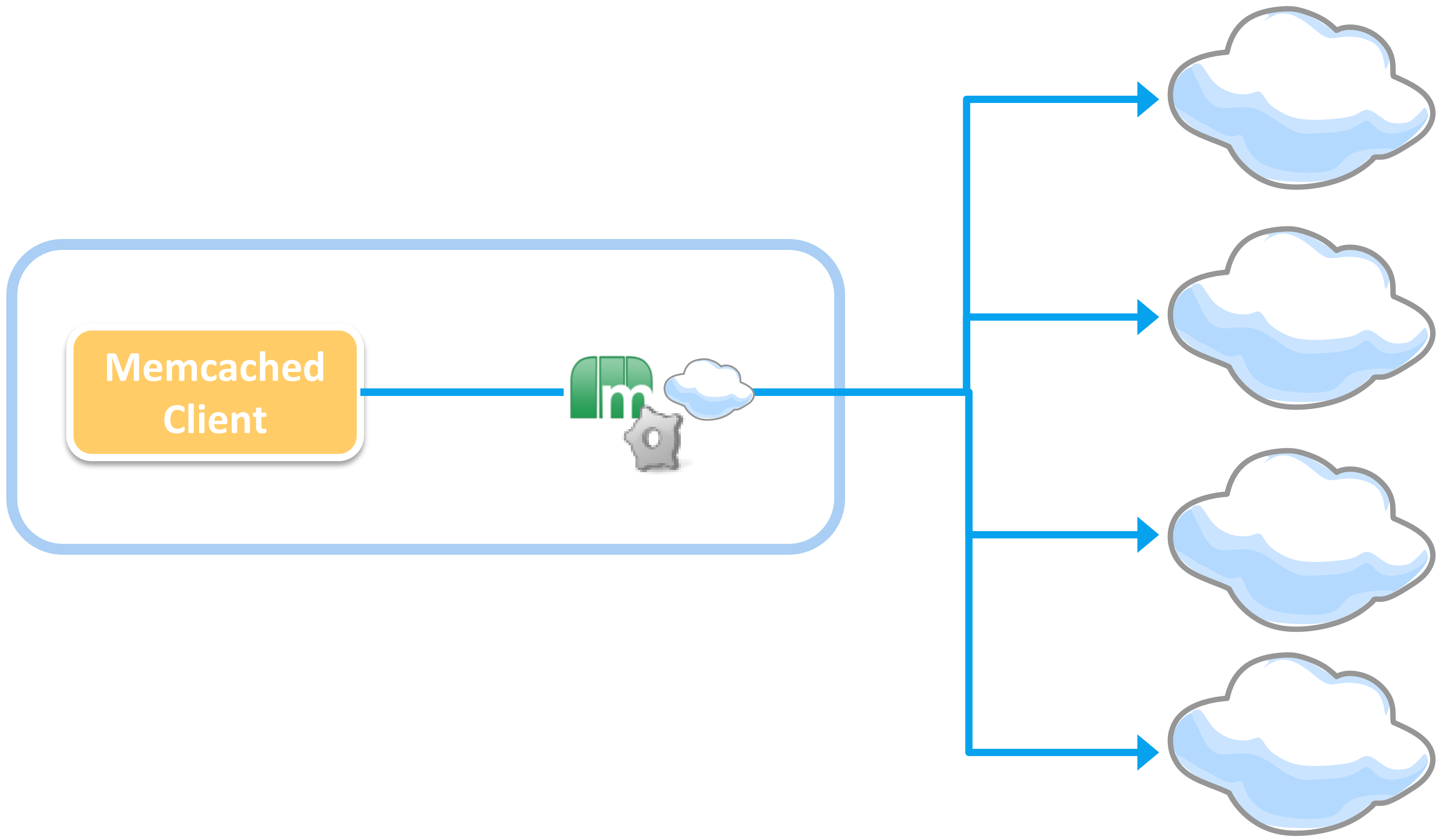
- Even for the same data…
  - **POJO & JPA** for Java apps with complex domain model
  - **Document** for a more dynamic view
  - **Memcached** for simple, language neutral data access
  - **JDBC** for:
    - Interaction with legacy apps
    - Flexible ad-hoc querying (e.g. projections)

GIGASPACES

Memcached (the Daemon is in the Details)



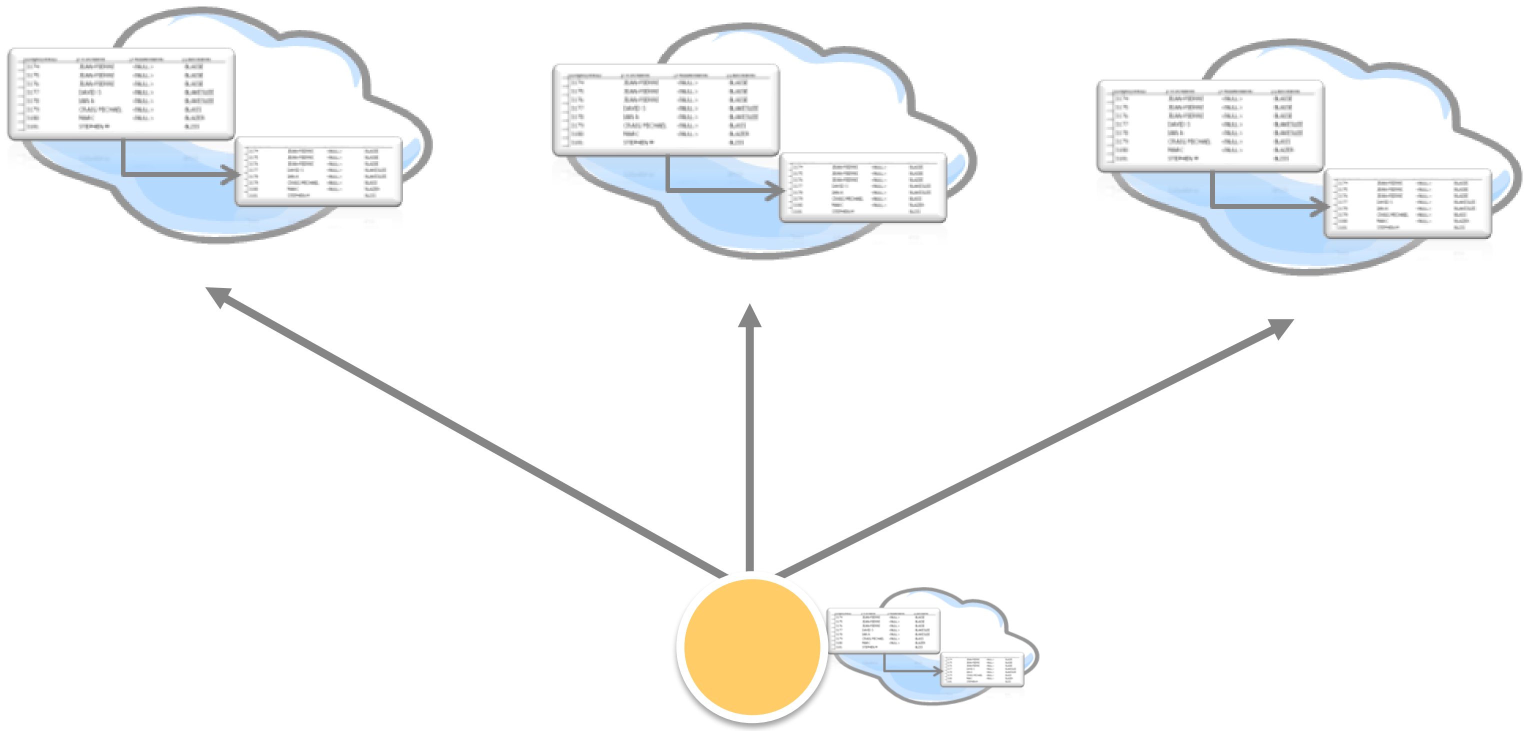
Memcached (the Daemon is in the Details)



SQL/JDBC – Query Them All

Query may involve Map/Reduce

- Reduce phase includes merging and sorting



GIGASPACES

SQL/JDBC – Things to Consider

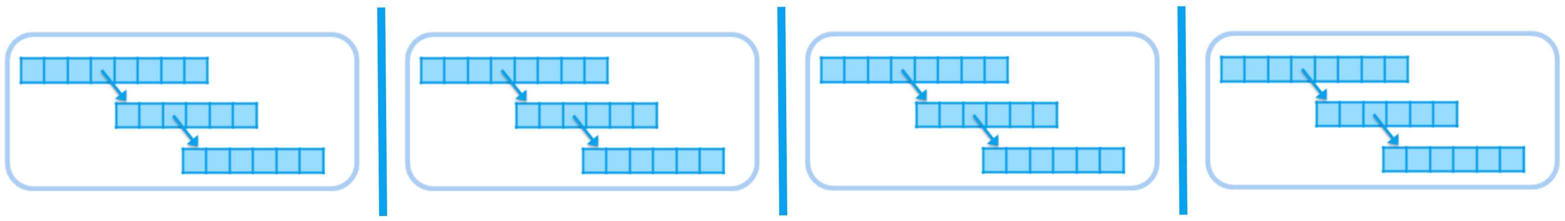
- Unique and FK constraints are not practically enforceable
- Sorting and aggregation may be expensive
- Distributed transactions are evil
  - Stay local…

JPA

It’s all about relationships…

JPA Relationships

To **embed** or not to embed, that is the question….



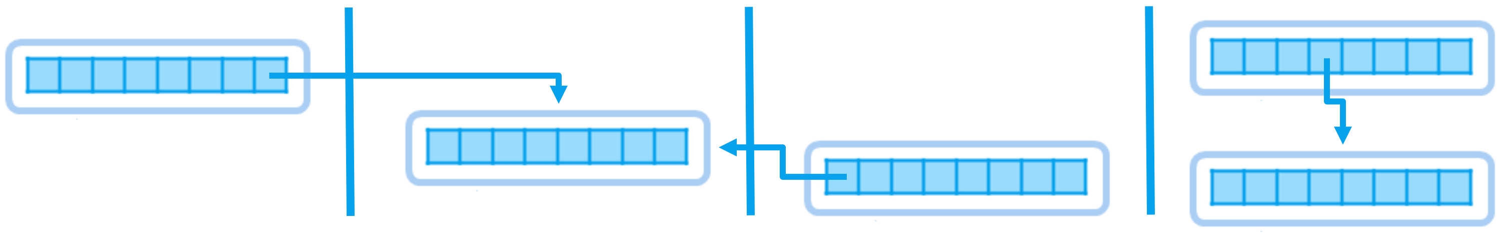
- ✓ Easy to partition and scale
- ✓ Easy to query:

```
user.accounts[*].type = 'checking'
```

- × Owned relationships only

JPA Relationships

To embed or **not to embed**, that is the question….



- ✓ Any type of relationship
- × Partitioning is hard
- × Querying involves joining

Summary

- One API doesn't fit all
  - Use the right API for the job
- Know the tradeoffs
  - Always ask what you're giving up, not just what you're gaining

THANK YOU!

@uri1803
http://blog.gigaspaces.com