GIGASPACES

WRITE ONCE.
SCALE ANYWHERE.

GigaSpaces Technologies

Yes, SQL!

Mickey Alon

```
> SELECT * FROM qcon.speakers WHERE
name=‘Mickey Alon’
+-------------------------------------------------------+
| Name        | Company     | Role               | Twitter   |
+-------------------------------------------------------+
| Mickey Alon | GigaSpaces | Director |  @mickey_alon |
+-------------------------------------------------------+
```

```
> db.speakers.find({name:”Mickey Alon”})
{
  “name”:”Mickey Alon”,
  “company”: {
                name:”GigaSpaces”,
                products:[“XAP”, “IMDG”]
                domain: “In memory data grids”
             }
  “role”:”director”,
  “twitter”:”@mickey_alon”
}
```

GIGASPACES

Agenda

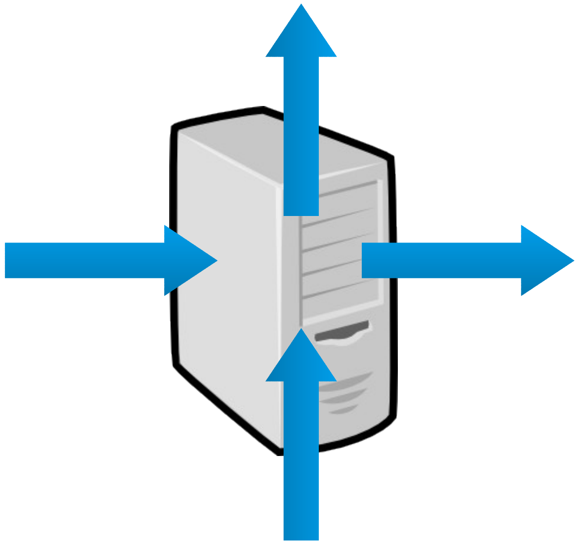
- SQL
  - What it is and isn't good for
- NoSQL
  - Motivation & Main Concepts of Modern Distributed Data Stores
  - Common interaction models
    - Key/Value, Column, Document
    - NOT consistency and distribution algorithms
- One Data Store, Multiple APIs
  - (Really) brief intro to GigaSpaces
  -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

4

A Few (more) Words About SQL

SQL

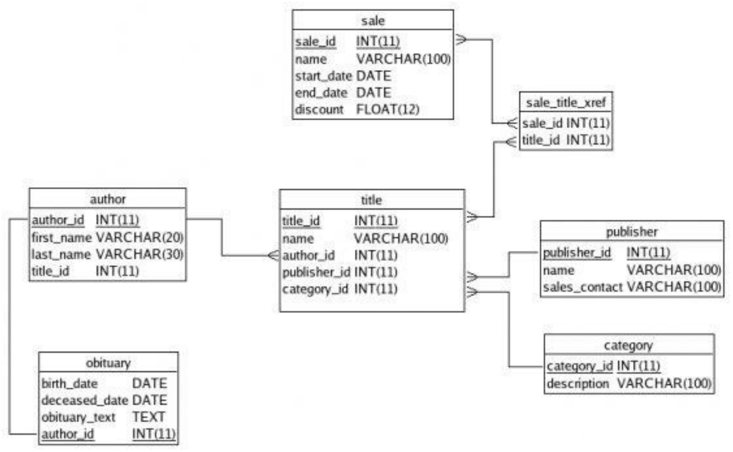
- **(Usually) Centralized** →
- Transactional, consistent
- Hard to Scale
  - Disk Based



SQL

Static, normalized data schema

- Don’t duplicate, use FKs



SQL

Add hoc query support

→ Model first, query later

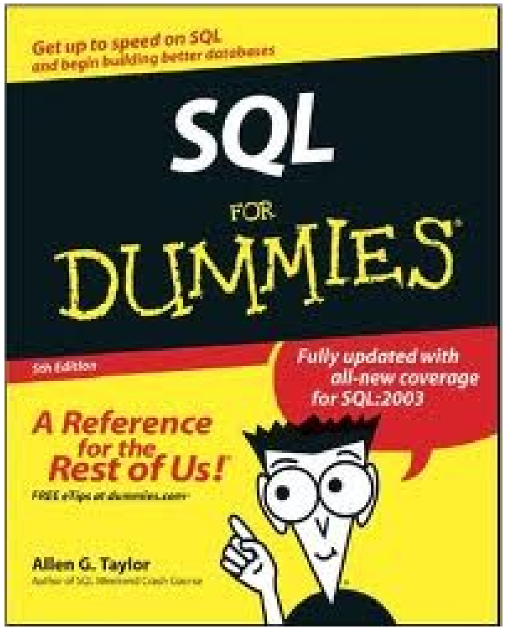
```
select users.user_id, users.email, count(*), max(classified_ads.posted)
from users, classified_ads
where users.user_id = classified_ads.user_id
group by users.user_id, users.email
order by upper(users.email);
```

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

Not Only SQL

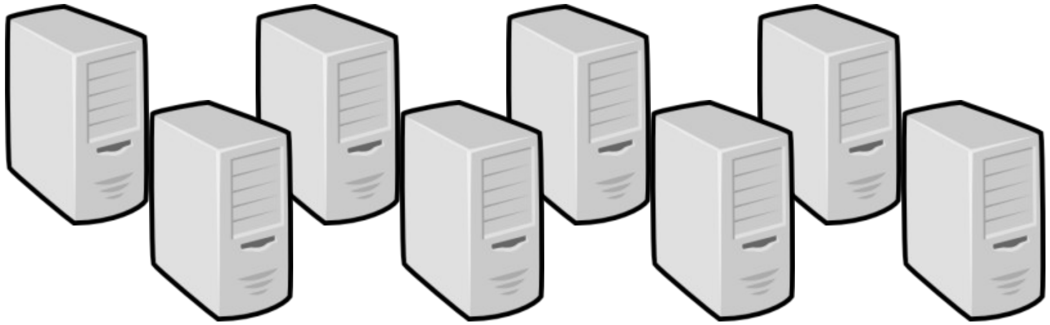
NoSql (or a Naive Attempt to Define It)

A loosely coupled collection of

non-relational multiple data stores

NoSQL – some key concepts

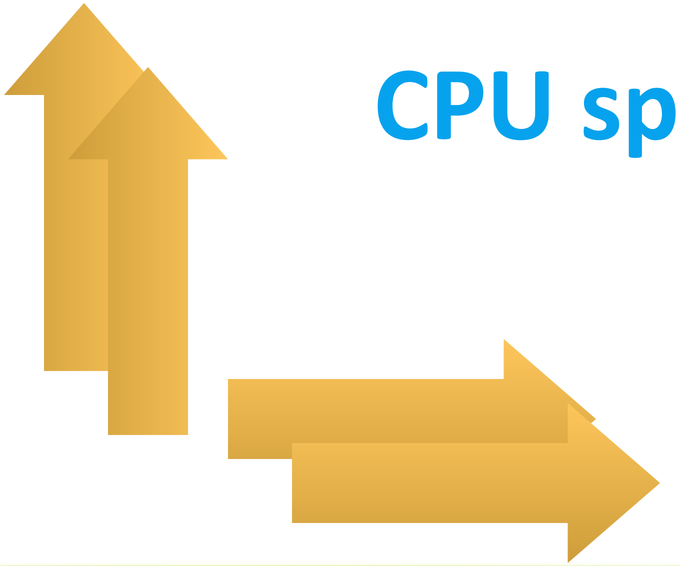
- Takes care of data-scaling
- Distributed by nature (mostly)
- Can scale up-to thousands of nodes
- Complements SQL – not replacing it



Few words about scaling

scalable (Up & Out)

!
Concurrency



CPU speed/Cores

What are the options?

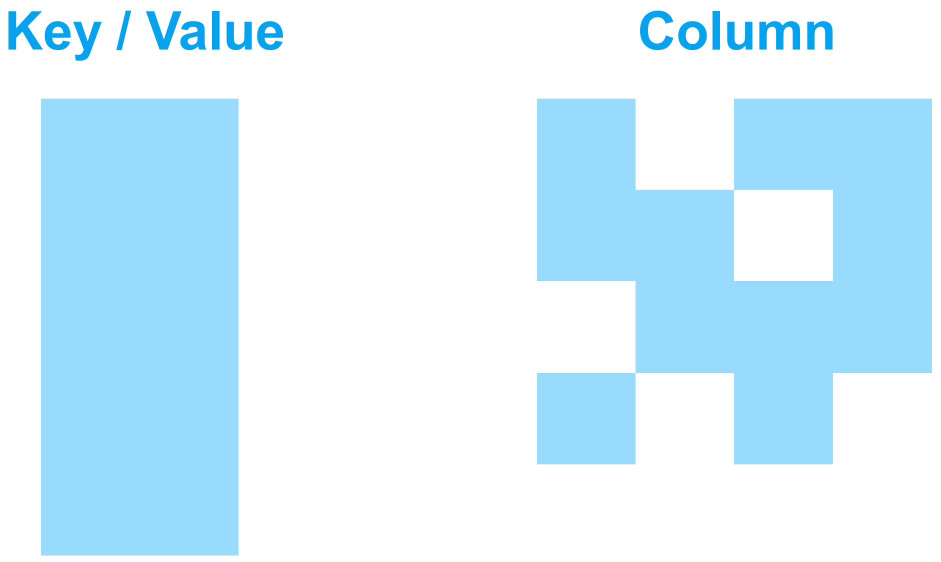
- Hardware based
  - Use extreme hardware such as RAC - 99,999% uptime
  - Non intrusive – minimal change
  - Can you afford it? How far does it scale?
- Software (NoSQL) based on commodity HW
  - Failures are more likely to happen (due to number of nodes)
  - Design for failure scenarios
    - Putting data in multiple nodes
    - Client support for transparent failover
  - Eventually consistent (CAP theorem)

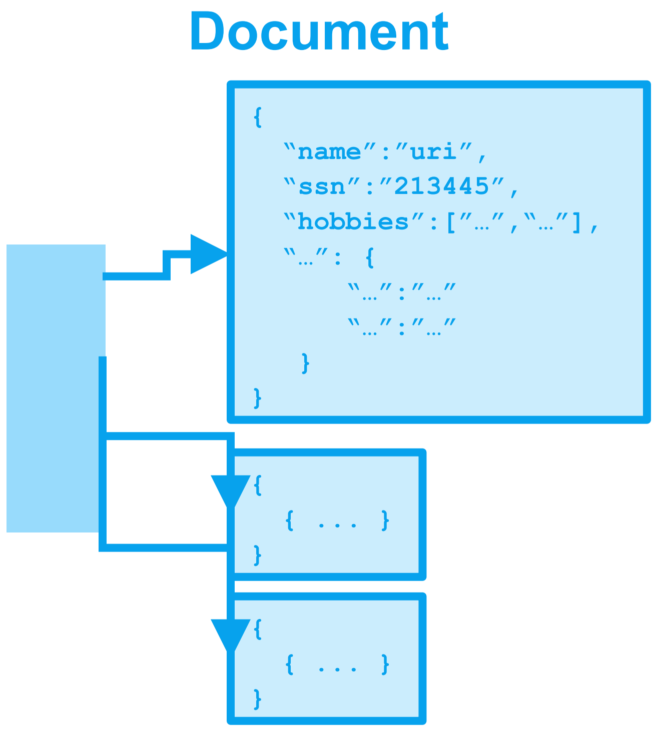
Why Now?

The Big Data Era

- Exponential Increase in data & throughput
  - We generate increasingly growing amount content
  - Data is being pushed to consumers
  - Caching ?
- Software is delivered as service
  - 24/7 + schema evolution + agility = leading constraints ?
- Competition is growing while price decrease

The Current Leading Data Models





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

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

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

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

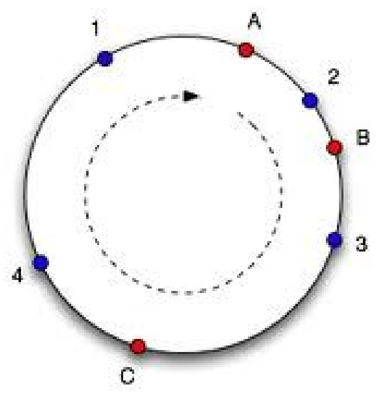
K1	V1
K2	V2
K3	V3
K4	V1

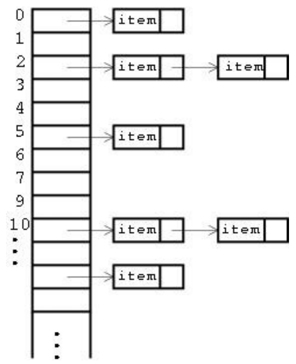
Key/Value

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 webservices™

SimpleDB

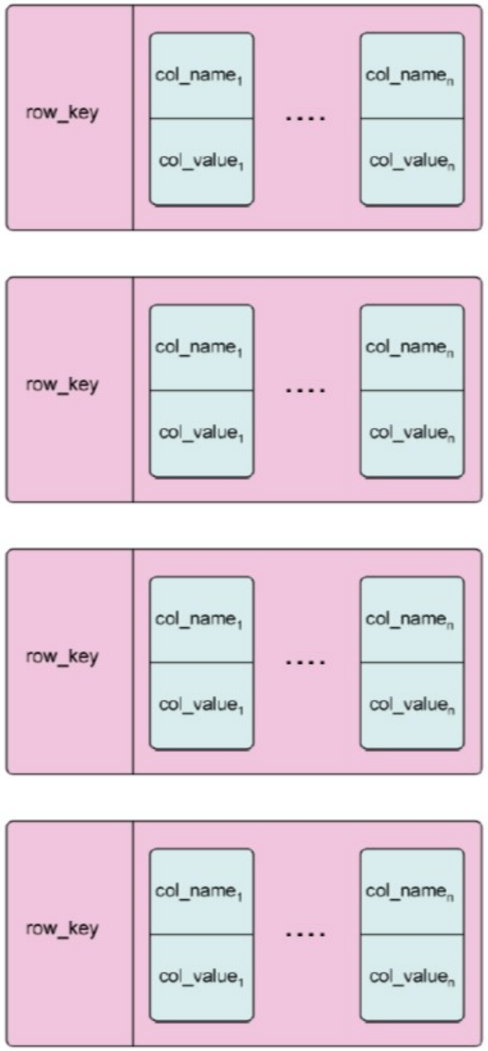
Column Based

- Google's BigTable / Amazon Dynamo
- One giant table of rows and columns
  - Column == pair (name and a value, sometimes timestamp)
  - Each row can have a different number of columns = flexible schema

```
Table ->* Rows ->* Columns ->* Values
```

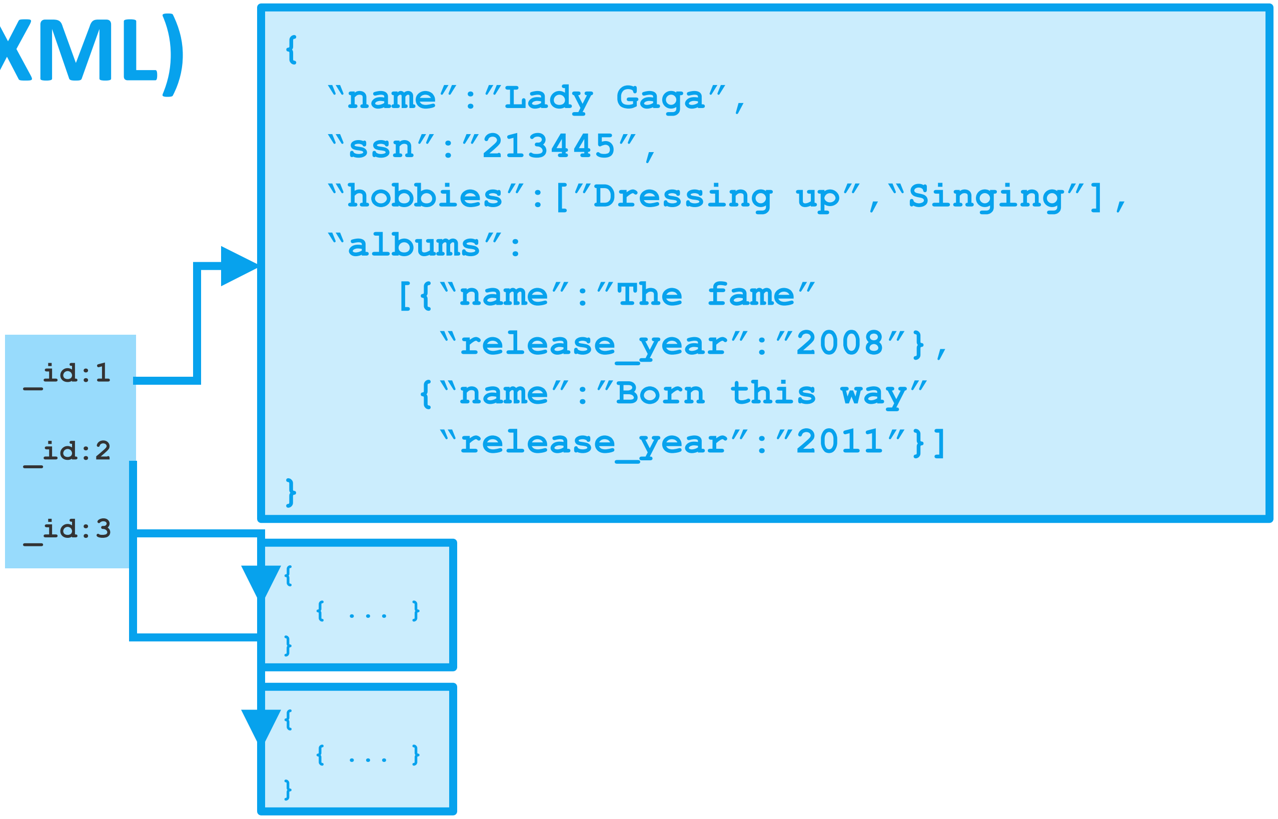
Better query support

- Query on row key
  - Or column value (aka secondary index)
- Good for a constantly changing, (albeit flat) domain model
- Can joins and relations be replaced by map/reduce?



Document

Think JSON
(or BSON, or XML)



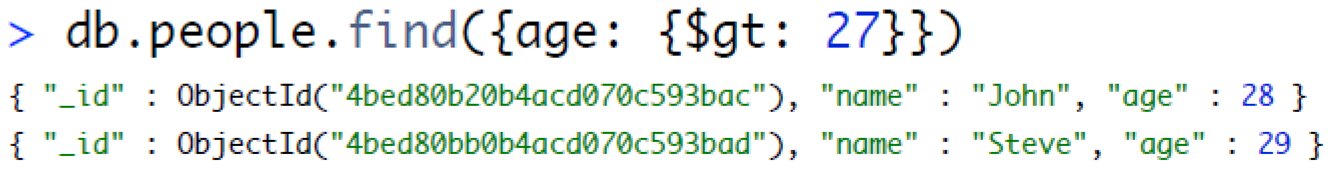
```
{
  "name":"Lady Gaga",
  "ssn":"213445",
  "hobbies":["Dressing up","Singing"],
  "albums":
     [{"name":"The fame"
       "release_year":"2008"},
      {"name":"Born this way"
       "release_year":"2011"}]
}
```

Document

- Built-in support for hierarchal model
  - Arrays, nested documents

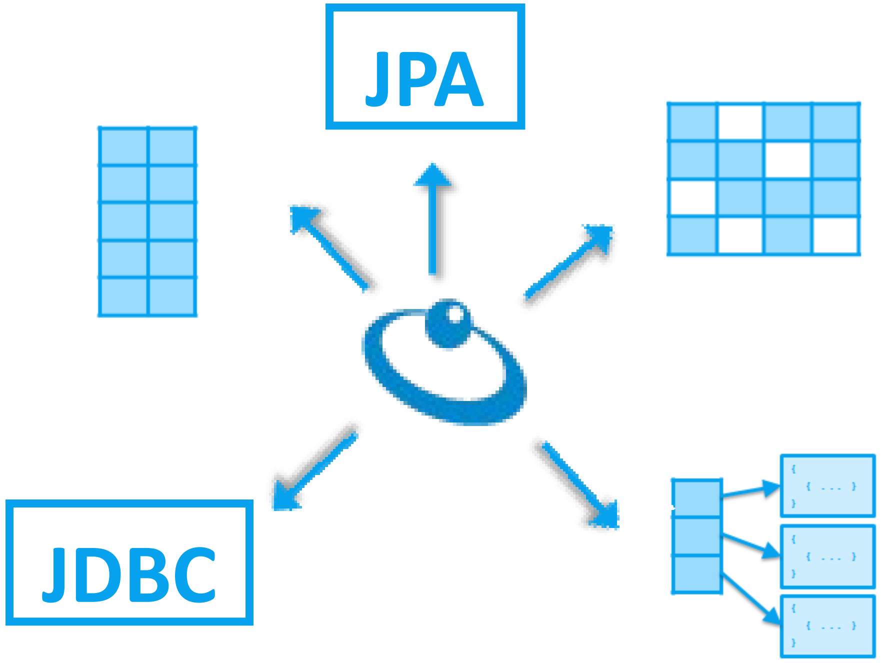
Great power comes with great responsibility!

  - normally comes with restful and map/reduce API
- Flexible schema



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

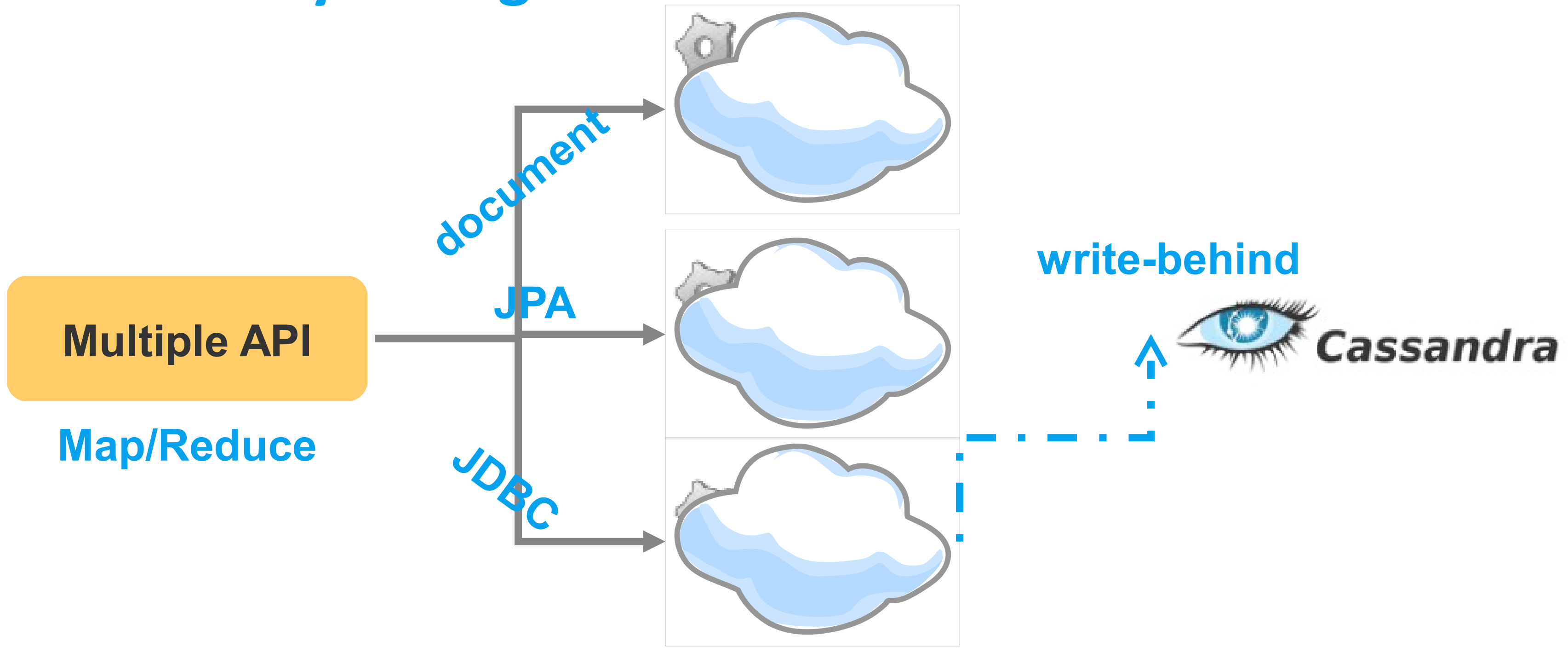
What if you didn't have to choose?



A Brief Intro to GigaSpaces

In Memory Data Grid

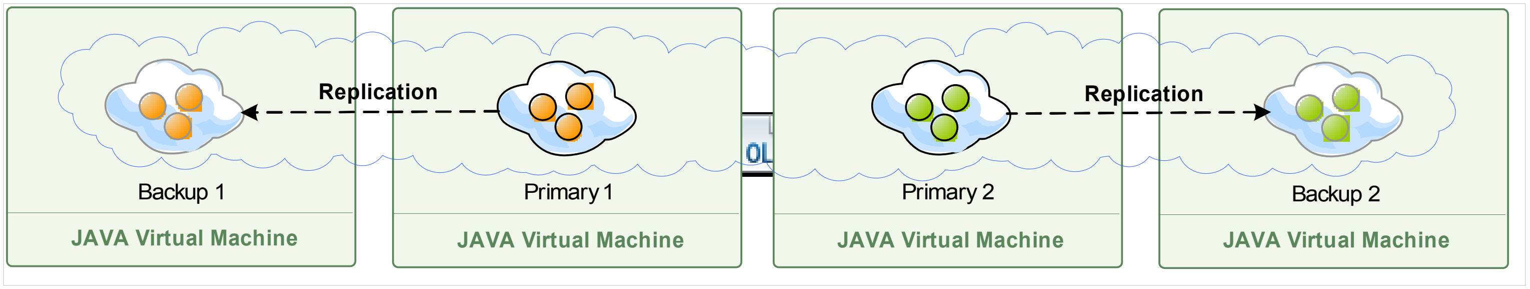
- With optional write behind to a secondary storage



A Brief Intro to GigaSpaces

Transparent partitioning & HA

- Fixed hashing based on a chosen property



A Brief Intro to GigaSpaces

Transactional (Like, ACID)

- Local (single partition)
- Distributed (multiple partitions)
- Durability via memory replication

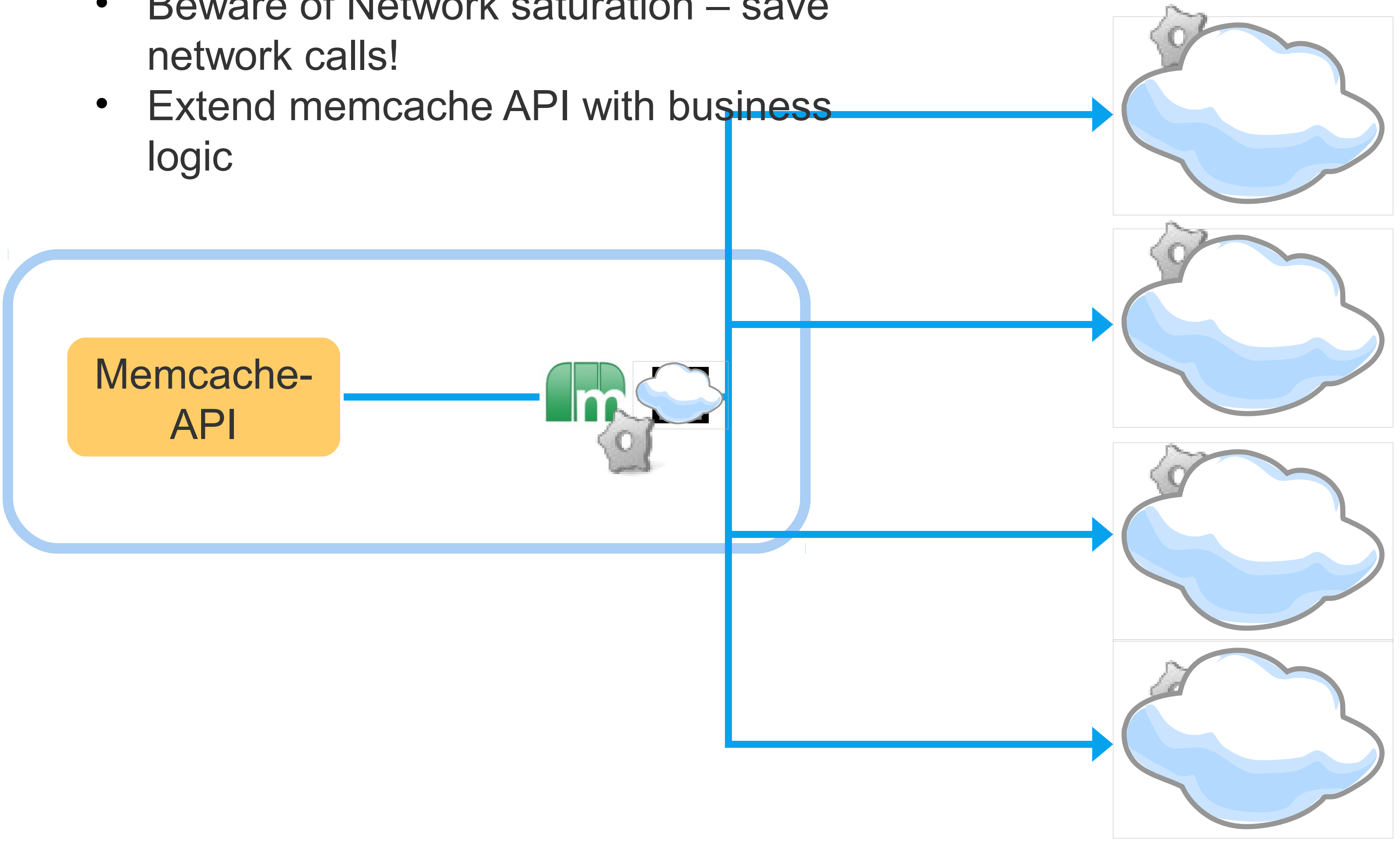
```
@Transactional
public void updateFoo(Foo foo) {
    // do something
}
```

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)

Memcached (the Daemon is in the Details)

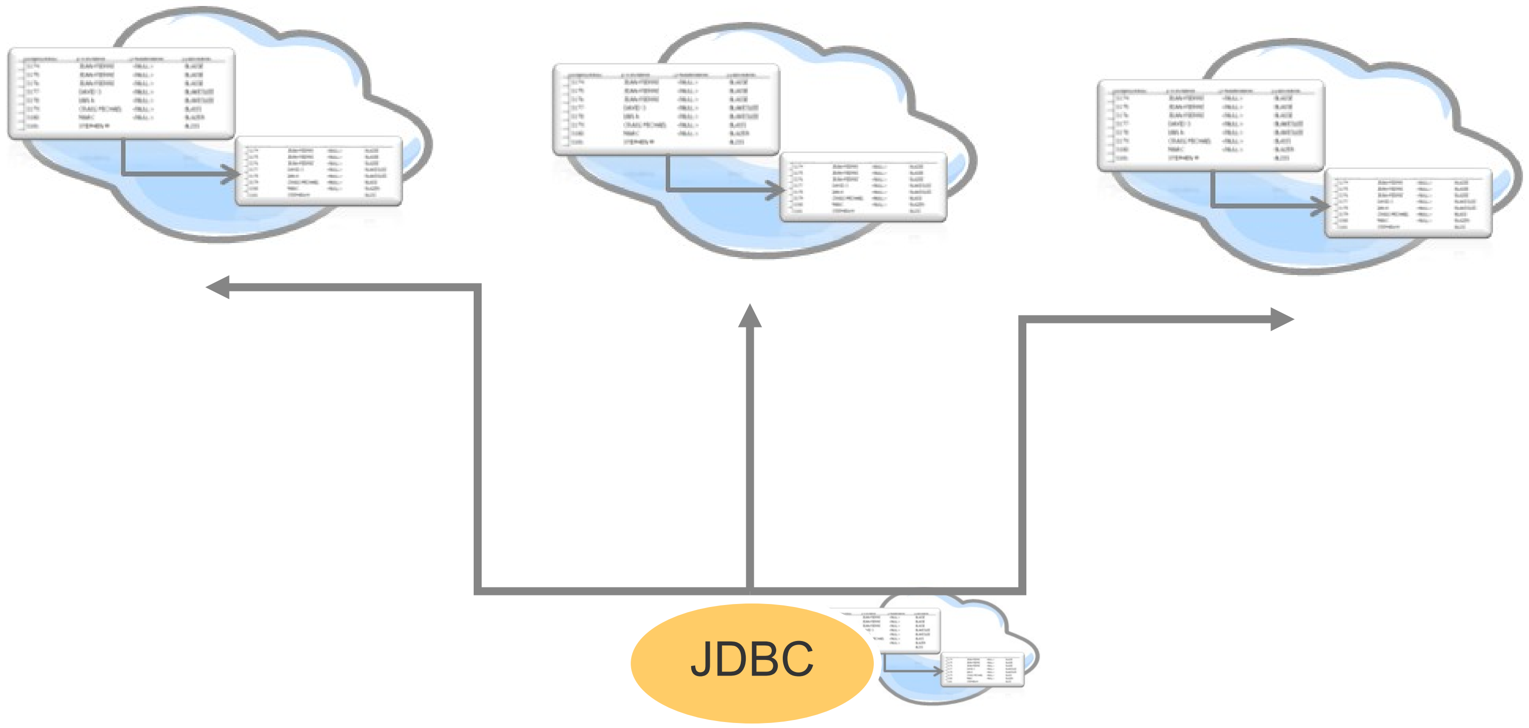
- Beware of Network saturation – save network calls!
- Extend memcache API with business logic



SQL/JDBC – Query Them All

Query may involve Map/Reduce

- Reduce phase includes merging and sorting



SQL/JDBC – Things to Consider

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

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!

@mickey_alon
http://blog.gigaspaces.com