ArangoDB

Polyglot Persistence & Multi-Model Databases

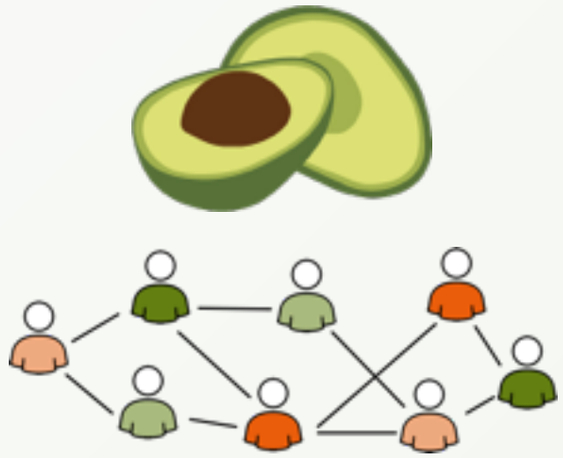
Java User Group Hessen

Michael Hackstein
@mchacki

www.arangodb.com

Michael Hackstein

- ArangoDB Core Team
  - Web Frontend
  - Graph visualisation
  - Graph features

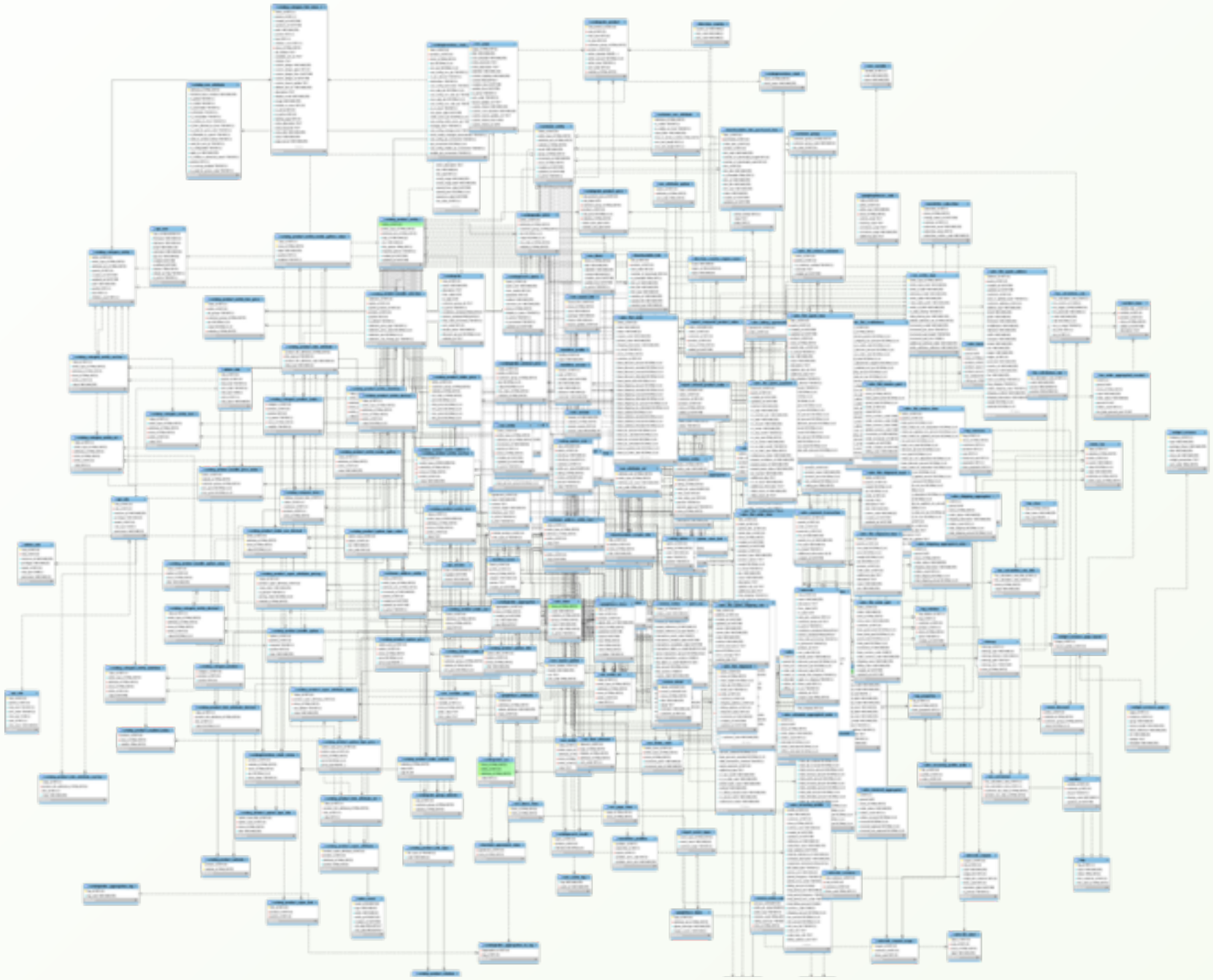


- Host of cologne.js

- Master's Degree
  (spec. Databases and
  Information Systems)

The Single Model era is over

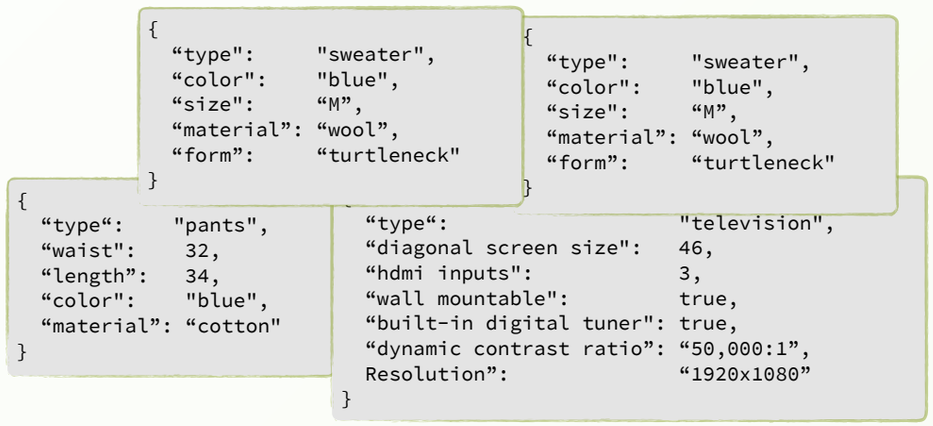
Relational World



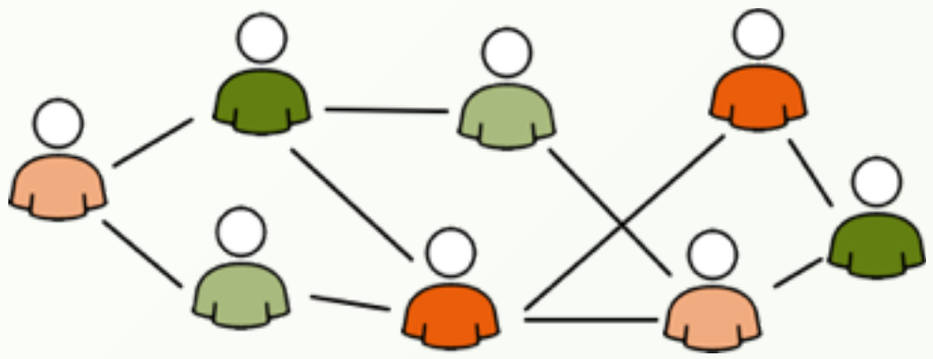
The Multi Model era begins

NoSQL World

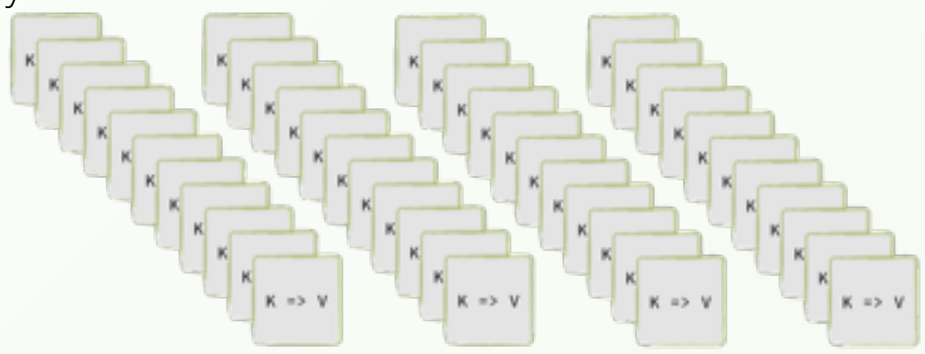
Documents - JSON



Graphs



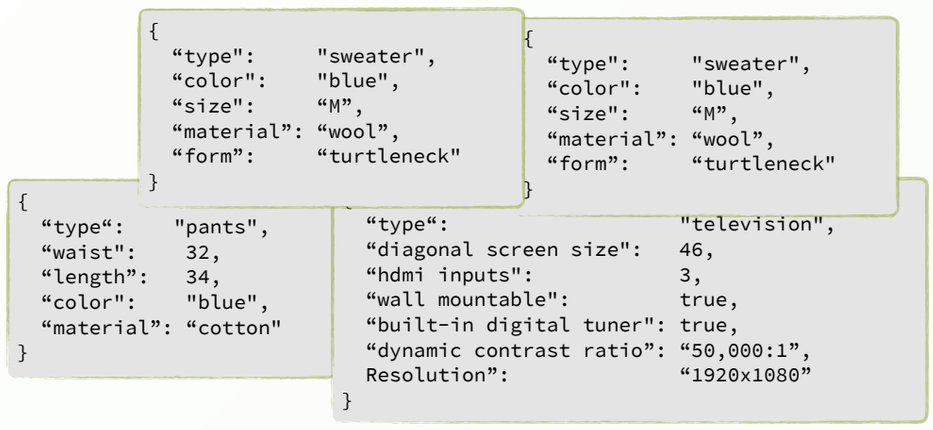
Key Value



The Multi Model era begins

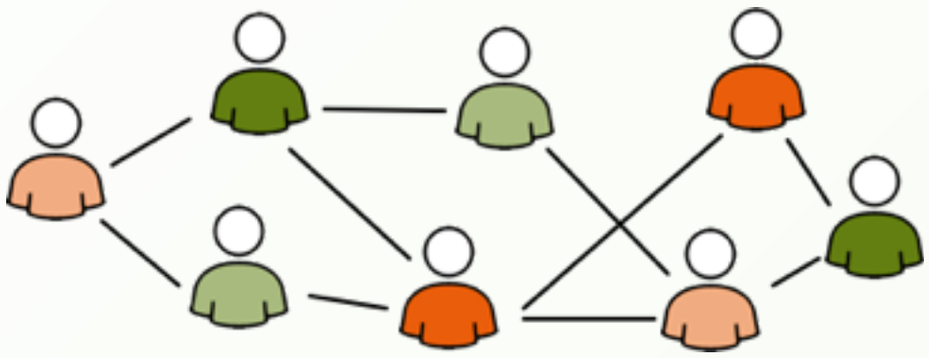
NoSQL World

Documents - JSON

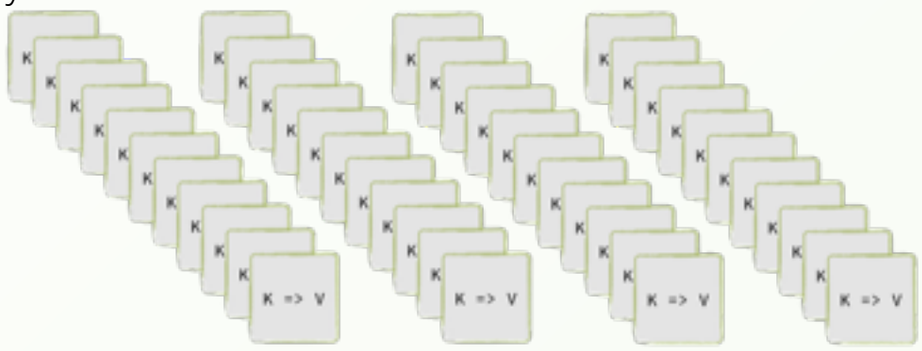


- Normally based on key-value stores (each document still has a unique key)
- Allow to save documents with logical similarity in “collections"
- Treat data records as attribute-structured documents (data is no more opaque)
- Often allow querying and indexing document attributes

Graphs



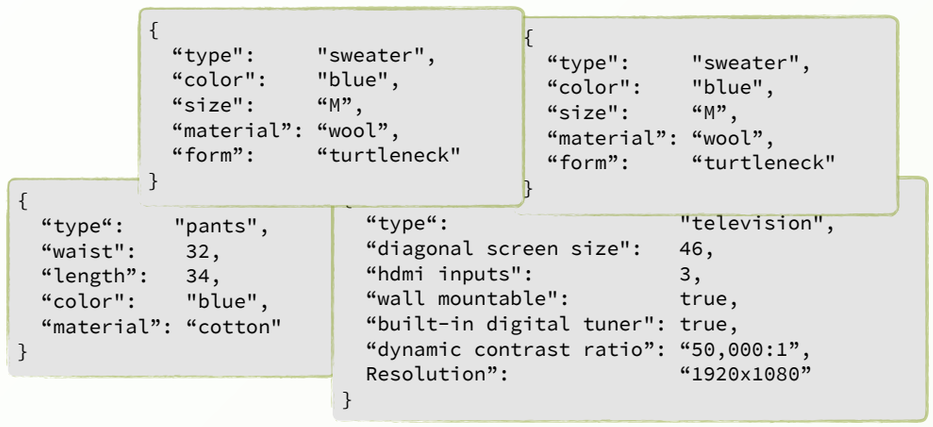
Key Value



The Multi Model era begins

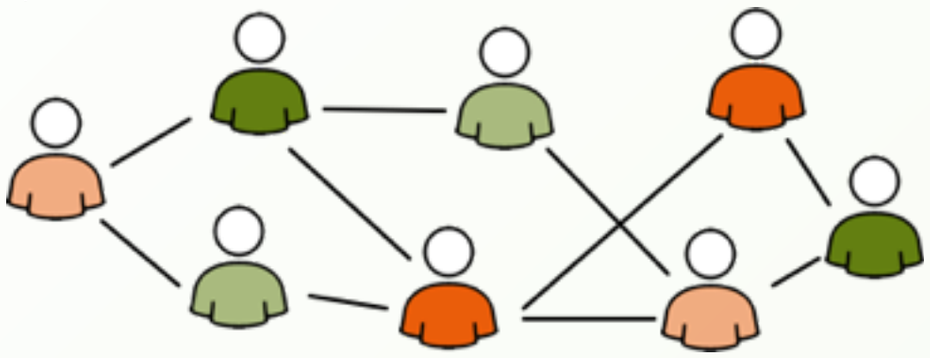
NoSQL World

Documents - JSON



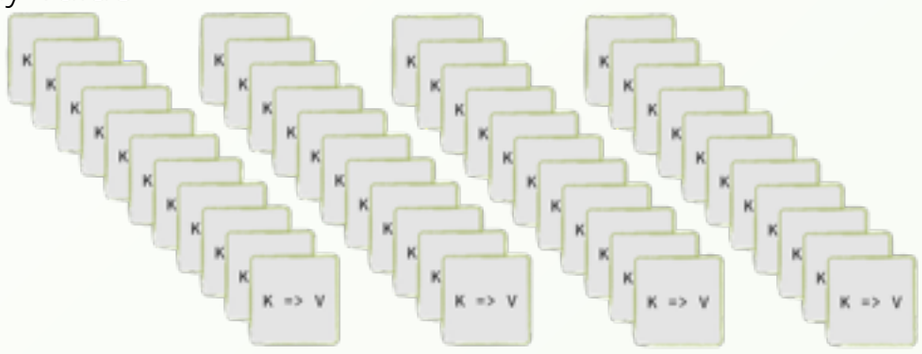
- Normally based on key-value stores (each document still has a unique key)
- Allow to save documents with logical similarity in “collections"
- Treat data records as attribute-structured documents (data is no more opaque)
- Often allow querying and indexing document attributes

Graphs



- Focussed on m-to-n relations between entities
- Stores property graphs: entities and edges can have attributes
- Easily query paths of variable length

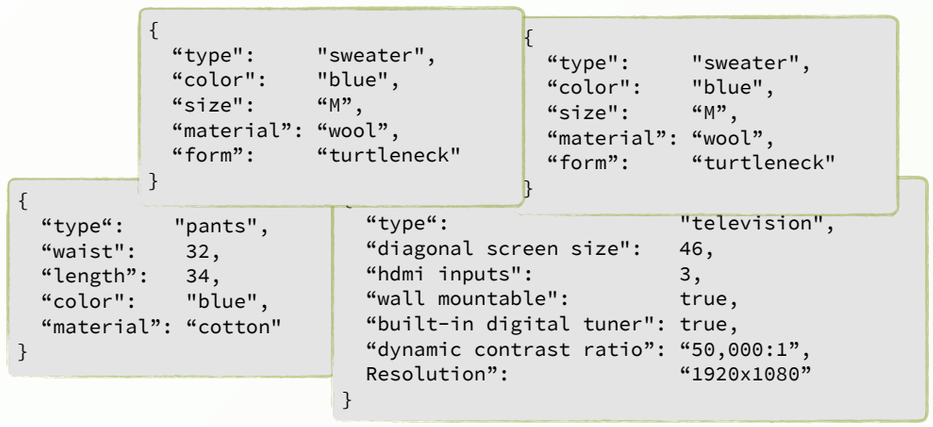
Key Value



The Multi Model era begins

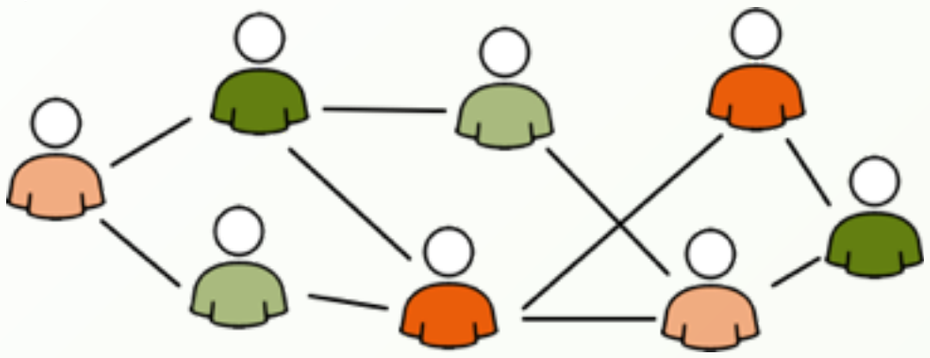
NoSQL World

Documents - JSON



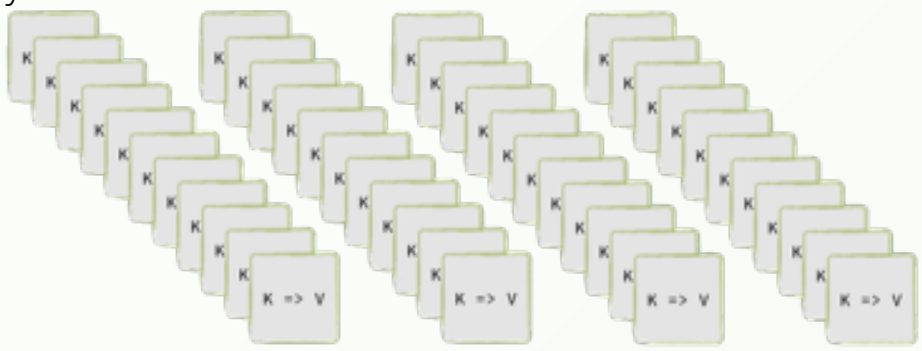
- Normally based on key-value stores (each document still has a unique key)
- Allow to save documents with logical similarity in “collections"
- Treat data records as attribute-structured documents (data is no more opaque)
- Often allow querying and indexing document attributes

Graphs



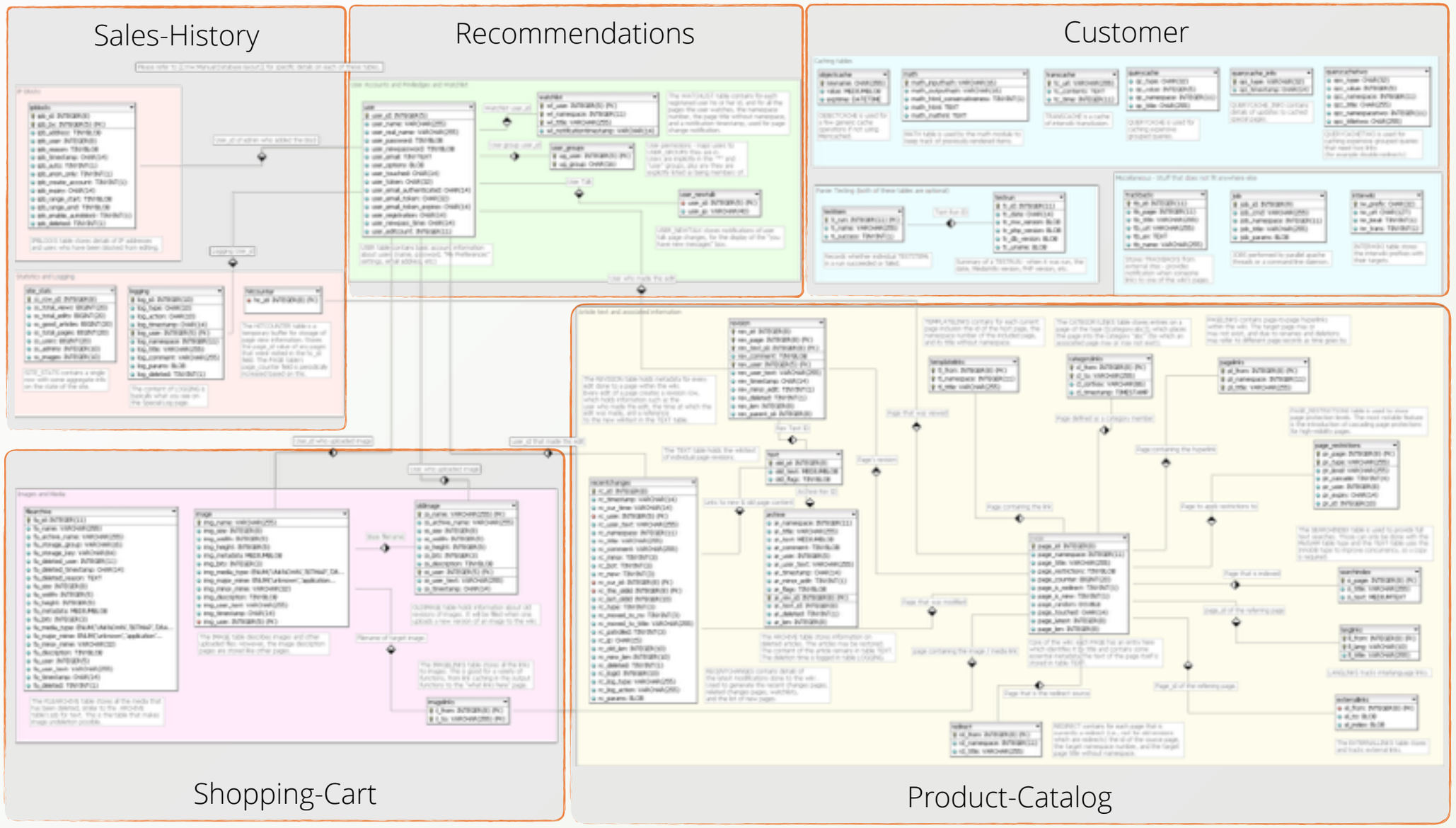
- Focussed on m-to-n relations between entities
- Stores property graphs: entities and edges can have attributes
- Easily query paths of variable length

Key Value

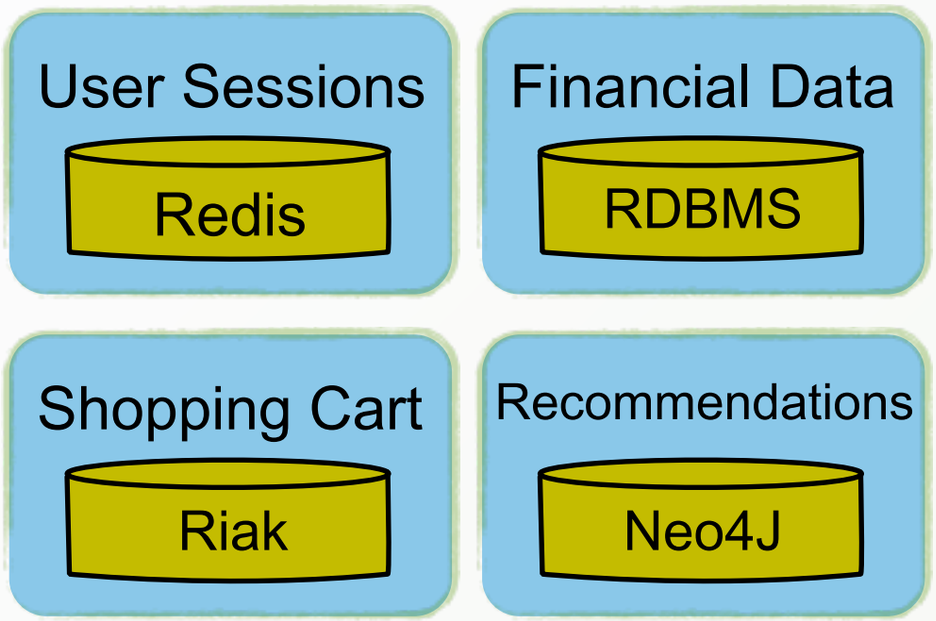


- Map value data to unique string keys (identifiers)
- Treat data as opaque (data has no schema)
- Can implement scaling and partitioning easily

e-commerce system in Relational World

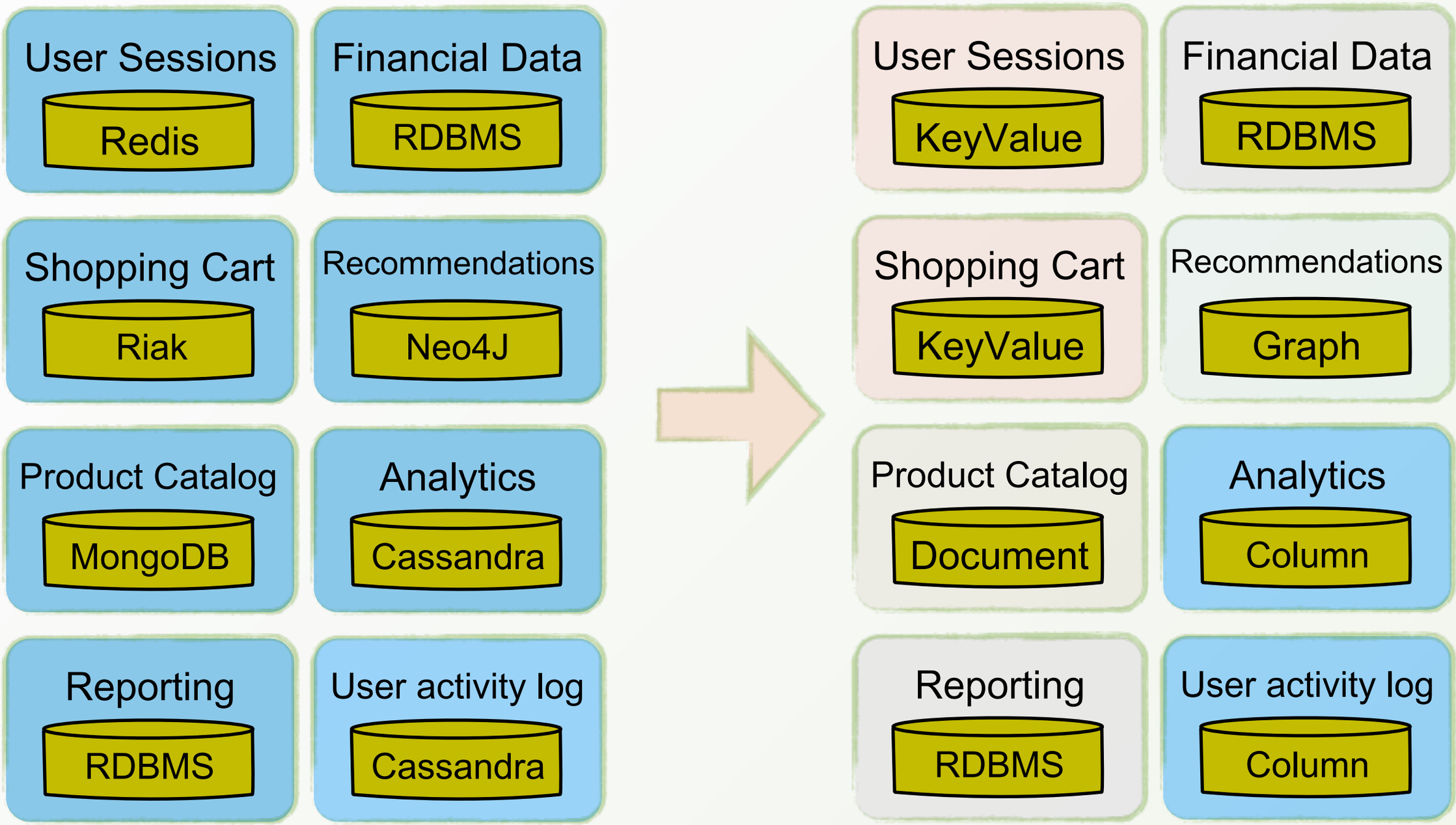


Polyglot Persistence



Source: Martin Fowler, http://martinfowler.com/articles/nosql-intro.pdf

Polyglot Persistence



Source: Martin Fowler, http://martinfowler.com/articles/nosql-intro.pdf

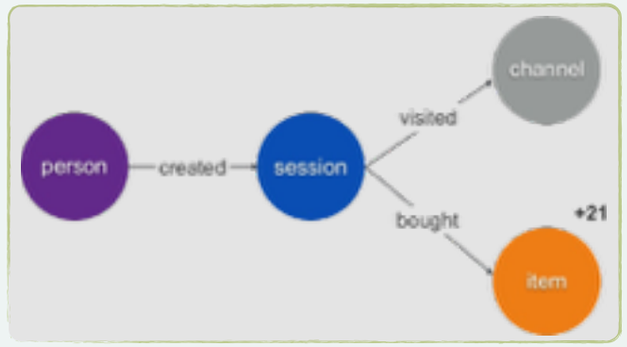
Single Model Databases

Sales-History

```
{
  “userID”:    239178239,
  “productID”: 128623883,
  “number”:    5,
  “price”:     12.20,
}
```

```
{
  “userID”:    239178239,
  “productID”: 128623883,
  “number”:    5,
  “price”:     12.20,
}
```

Recommendations



Customer

```
{
  “Name”:              "Smith",
  “lastLogin”:         “2012-11-01",
  “Visits”:            121,
  “shipping address”: “abc”,
  “shipping address”: “def”
}
```

```
{
  “Name”:              "Meyer",
  “lastLogin”:         “2012-11-21",
  “Visits”:            20,
  “shipping address”: “xyz”,
}
```

Shopping-Cart

```
423453453=>
4328, “shirt”, “L”, 1, 12.99
6378, “sweater”, “M”, 2, 37.95
3245, “sweater”, “blue”, 1, 99.95
3245, “pants”, “32/34”, “black”, 1, 99.95
```

```
874365563=>
5463, “shirt”, “S”, 1, 9.99
6378, “sweater”, “M”, 2, 37.95
3245, “pants”, “32/34”, “black”, 1, 99.95
```

Product-Catalog

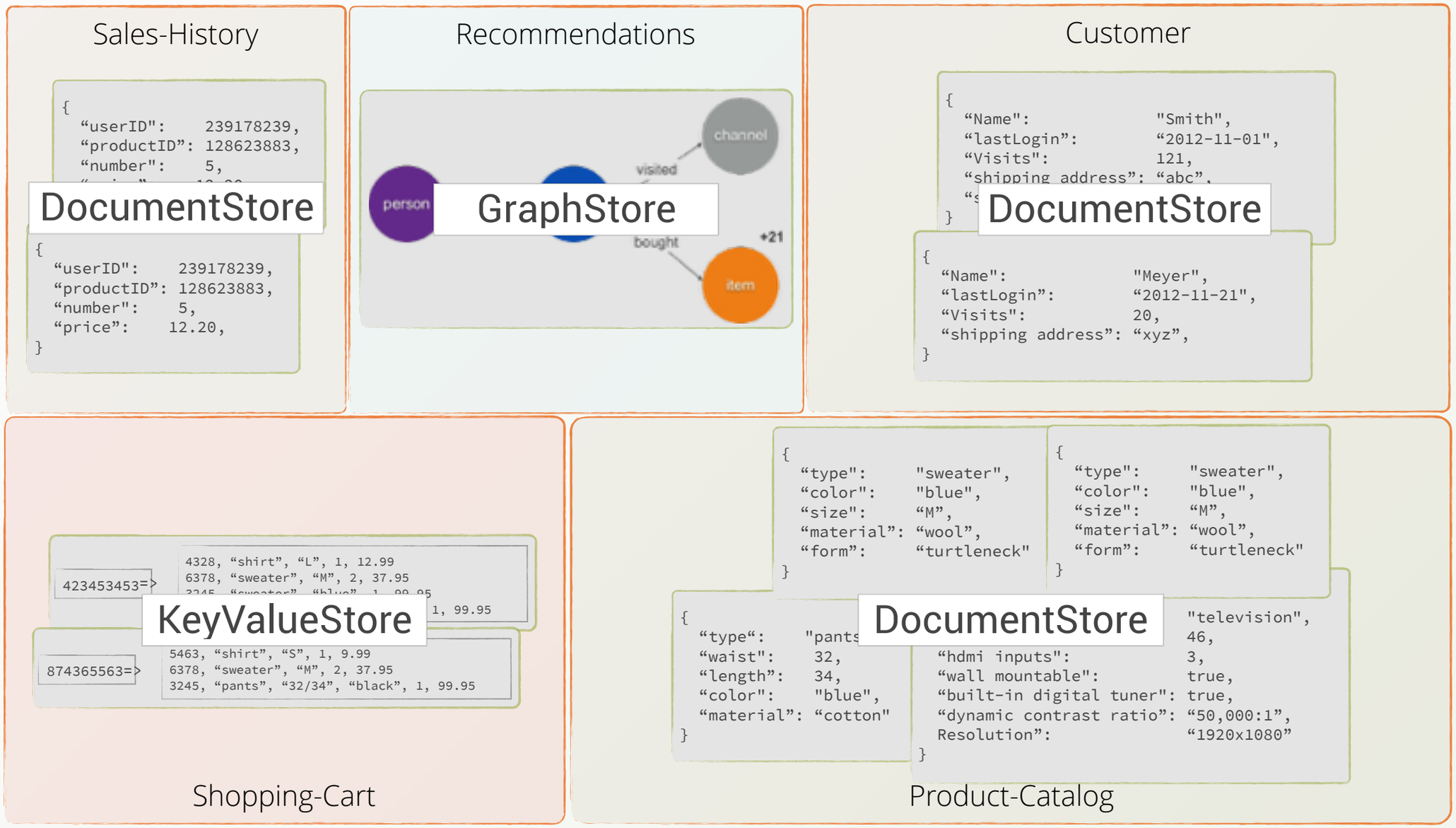
```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”: “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”: “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type“:    "pants",
  “waist”:    32,
  “length”:   34,
  “color”:    "blue",
  “material”: “cotton"
}
```

```
{
  “type”:                          "television",
  “diagonal screen size”:          46,
  “hdmi inputs":                   3,
  “wall mountable":                true,
  “built-in digital tuner":        true,
  “dynamic contrast ratio”:        “50,000:1”,
  Resolution”:                     “1920x1080”
}
```

Single Model Databases



Single Model Databases

mongoDB

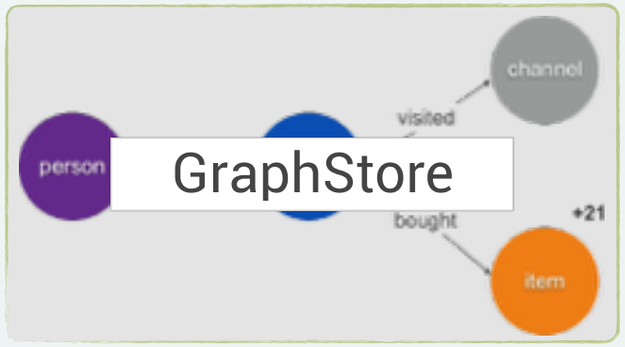
Neo4j

Sales-History

```
{
  “userID”:    239178239,
  “productID”: 128623883,
  “number”:    5,
  “price”:     12.20,
}
```

DocumentStore

Recommendations



GraphStore

Customer

```
{
  “Name”:              "Smith",
  “lastLogin”:         “2012-11-01",
  “Visits”:            121,
  “shipping address”: “abc”,
}
```

DocumentStore

```
{
  “Name”:              "Meyer",
  “lastLogin”:         “2012-11-21",
  “Visits”:            20,
  “shipping address”: “xyz”,
}
```

Shopping-Cart

```
423453453=>
4328, “shirt”, “L”, 1, 12.99
6378, “sweater”, “M”, 2, 37.95
3245, “sweater”, “blue”, 1, 99.95

874365563=>
5463, “shirt”, “S”, 1, 9.99
6378, “sweater”, “M”, 2, 37.95
3245, “pants”, “32/34”, “black”, 1, 99.95
```

KeyValueStore

redis

Product-Catalog

```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”: “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”: “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type“:     "pants",
  “waist”:     32,
  “length”:    34,
  “color”:     "blue",
  “material”: “cotton"
}
```

```
  “type”:                  "television",
  “hdmi inputs”:           46,
  “wall mountable”:        3,
  “built-in digital tuner”: true,
  “dynamic contrast ratio”: true,
  “50,000:1”,
  Resolution”:             “1920x1080”
}
```

DocumentStore

Couchbase

Benefits & Overhead

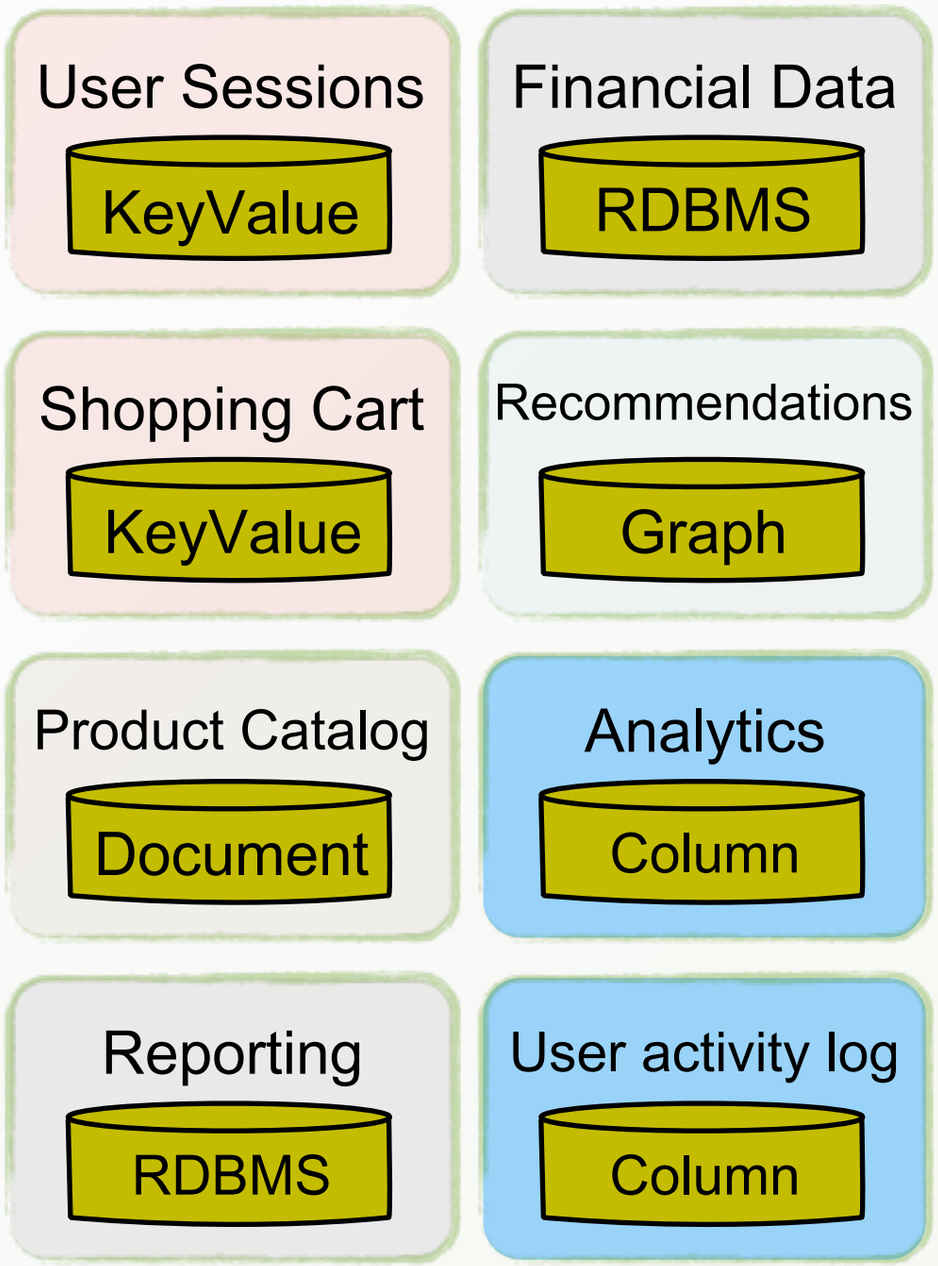
- Natural mapping of data into DB
- DB optimized for the data format
- Queries are tailored for your data format
- Focus on writing business logic

- Data has to be stored redundantly and has to be kept in sync
- Several technologies involved
- Administration effort is huge

Solution: Multi Model Database

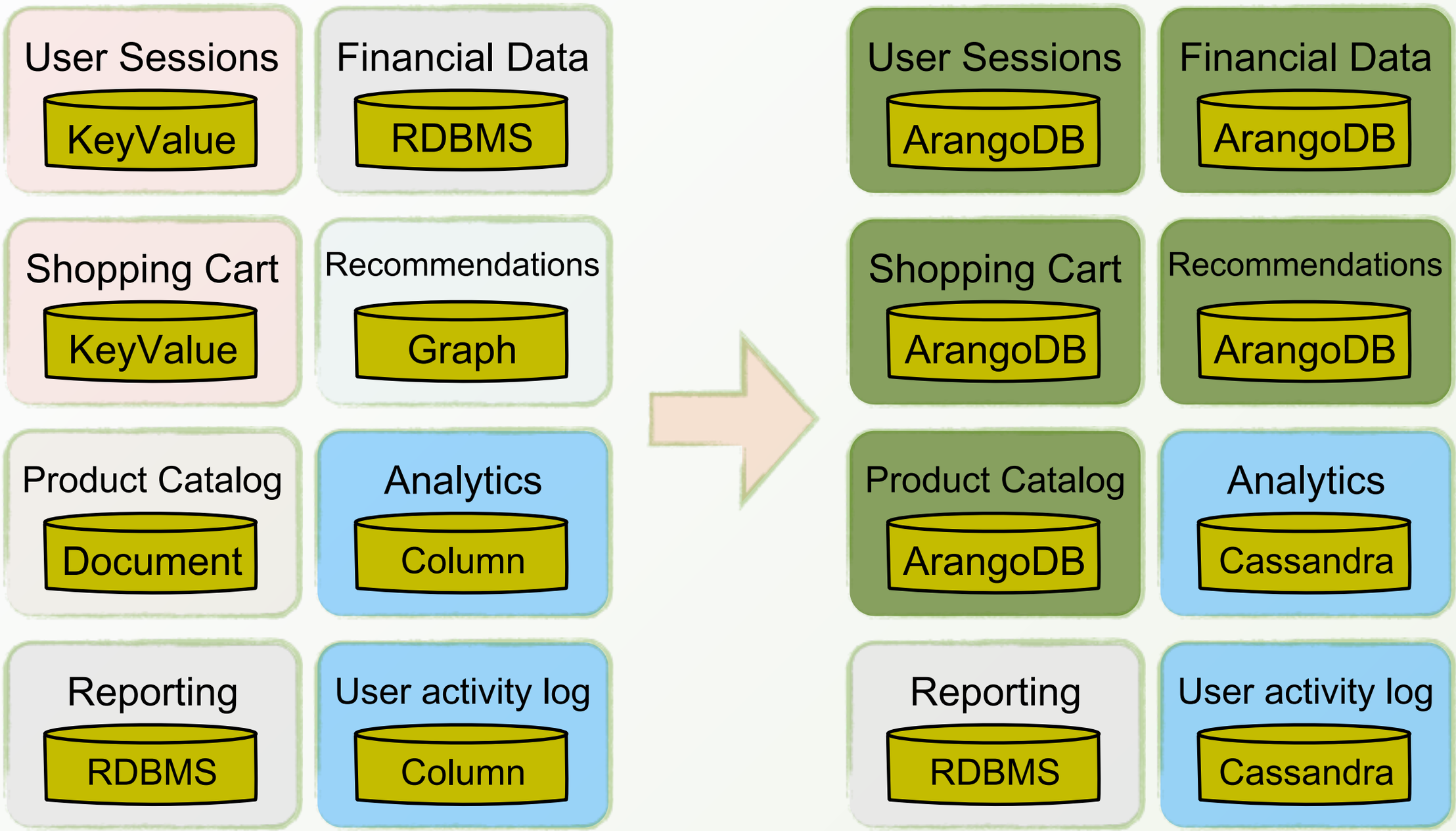
- Can natively store several kinds of data models:
  - Key-value pairs
  - Documents
  - Graphs
- Delivers query mechanisms for all data models

Polyglot Persistence Revisited



Source: Martin Fowler, http://martinfowler.com/articles/nosql-intro.pdf

Polyglot Persistence Revisited



Source: Martin Fowler, http://martinfowler.com/articles/nosql-intro.pdf

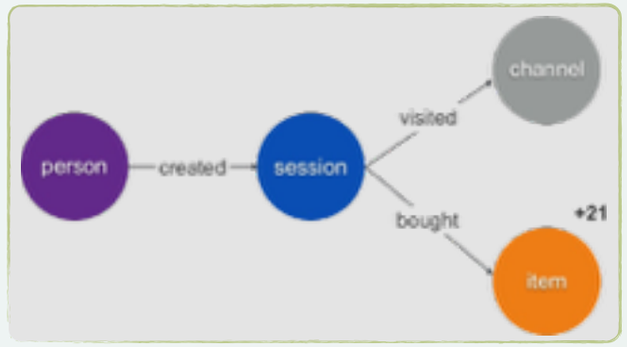
Use Case: Multi-Model-Databases

Sales-History

```
{
  “userID”:    239178239,
  “productID”: 128623883,
  “number”:    5,
  “price”:     12.20,
}
```

```
{
  “userID”:    239178239,
  “productID”: 128623883,
  “number”:    5,
  “price”:     12.20,
}
```

Recommendations



Customer

```
{
  “Name”:              "Smith",
  “lastLogin”:         “2012-11-01",
  “Visits”:            121,
  “shipping address”: “abc”,
  “shipping address”: “def”
}
```

```
{
  “Name”:              "Meyer",
  “lastLogin”:         “2012-11-21",
  “Visits”:            20,
  “shipping address”: “xyz”,
}
```

Shopping-Cart

```
423453453=>
4328, “shirt”, “L”, 1, 12.99
6378, “sweater”, “M”, 2, 37.95
3245, “sweater”, “blue”, 1, 99.95
3245, “pants”, “32/34”, “black”, 1, 99.95
```

```
874365563=>
5463, “shirt”, “S”, 1, 9.99
6378, “sweater”, “M”, 2, 37.95
3245, “pants”, “32/34”, “black”, 1, 99.95
```

Product-Catalog

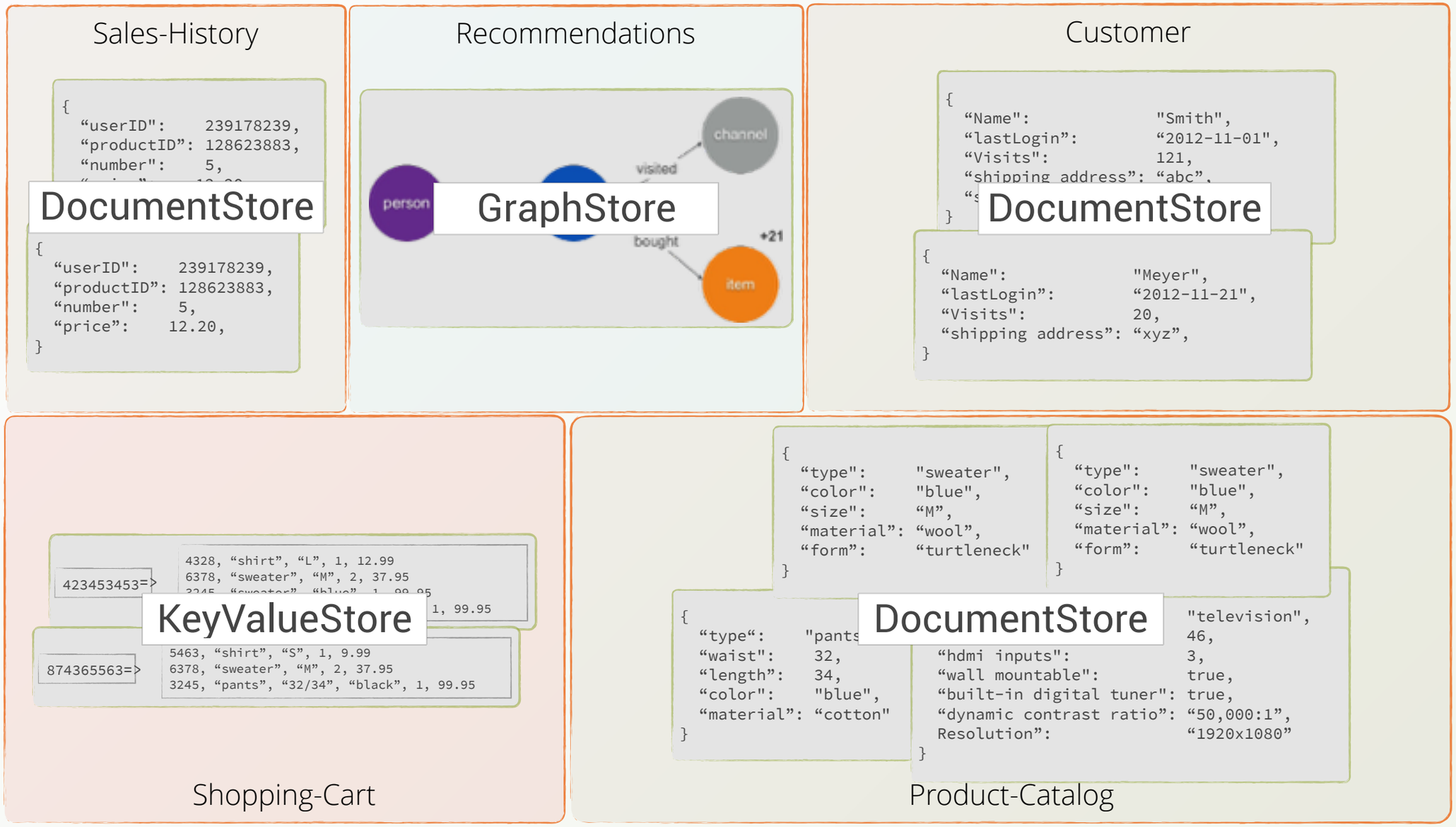
```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”: “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”: “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type“:    "pants",
  “waist”:    32,
  “length”:   34,
  “color”:    "blue",
  “material”: “cotton"
}
```

```
{
  “type“:                          "television",
  “diagonal screen size”:          46,
  “hdmi inputs":                   3,
  “wall mountable":                true,
  “built-in digital tuner":        true,
  “dynamic contrast ratio”:        “50,000:1”,
  Resolution”:                     “1920x1080”
}
```

Use Case: Multi-Model-Databases



Use Case: Multi-Model-Databases

ArangoDB

Sales-History

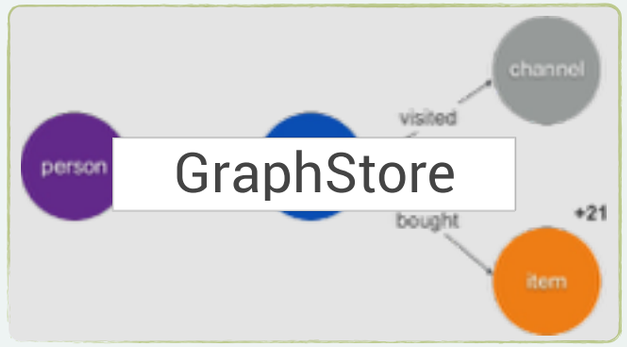
DocumentStore

```
{
  “userID”:    239178239,
  “productID”: 128623883,
  “number”:    5,
  “price”:     12.20,
}
```

ArangoDB

Recommendations

GraphStore



ArangoDB

Customer

DocumentStore

```
{
  “Name”:              "Smith",
  “lastLogin”:         “2012-11-01",
  “Visits”:            121,
  “shipping address”: “abc”,
  “
}
```

```
{
  “Name”:              "Meyer",
  “lastLogin”:         “2012-11-21",
  “Visits”:            20,
  “shipping address”: “xyz”,
}
```

KeyValueStore

```
423453453=>  4328, “shirt”, “L”, 1, 12.99
             6378, “sweater”, “M”, 2, 37.95
             3245, “sweater”, “blue”, 1, 99.95

874365563=>  5463, “shirt”, “S”, 1, 9.99
             6378, “sweater”, “M”, 2, 37.95
             3245, “pants”, “32/34”, “black”, 1, 99.95
```

Shopping-Cart

ArangoDB

DocumentStore

```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”:  “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type”:      "sweater",
  “color”:     "blue",
  “size”:      “M”,
  “material”:  “wool”,
  “form”:      “turtleneck"
}
```

```
{
  “type“:     "pants",
  “waist”:    32,
  “length”:   34,
  “color”:    "blue",
  “material”: “cotton"
}
```

```
  “hdmi inputs”:                3,
  “wall mountable”:             true,
  “built-in digital tuner”:     true,
  “dynamic contrast ratio”:     “50,000:1”,
  Resolution”:                  “1920x1080”
}
```

Product-Catalog

ArangoDB

My four favorite features of ArangoDB

- AQL offering joins & traversals
- ACID including Multi Collection Transactions
- MULTI-MODEL stores graphs and documents

My four favorite features of ArangoDB

- AQL offering joins & traversals
- ACID including Multi Collection Transactions
- MULTI-MODEL stores graphs and documents
- FOXX extend the API and adapt it to your needs

AQL

```
FOR user IN users
    RETURN user
```

AQL

```
FOR user IN users
    FILTER user.name == "alice"
    RETURN user
```

AQL

```
FOR user IN users
  FILTER user.name == "alice"
  FOR invoice IN invoices
    FILTER user._key == invoice.customer
    RETURN {
      user: user,
      invoice: invoice
    }
```

AQL

```
FOR user IN users
   FILTER user.name == "alice"
   RETURN {
      user: user,
      invoices: (
        FOR invoice IN invoices
          FILTER user._key == invoice.customer
          RETURN invoice
      )
   }
```

AQL

```
FOR user IN users
  FILTER user.name == "alice"
  RETURN {
    user: user,
    hasToPay: SUM(
      FOR invoice IN invoices
        FILTER user._key == invoice.customer
        FILTER !invoice.payed
        RETURN invoice.price
    )
  }
```

AQL - Alternative

```
FOR user IN users
  FILTER user.name == "alice"
  FOR invoice IN invoices
    FILTER user._key == invoice.customer
    FILTER !invoice.payed
    COLLECT u = user AGGREGATE toPay = SUM(invoice.price)
    RETURN {
      user: u,
      hasToPay: toPay
    }
```

AQL

```
FOR user IN users
    FILTER user.name == "alice"
    FOR product IN OUTBOUND user has_bought
    RETURN product
```

AQL

```
FOR user IN users
    FILTER user.name == "alice"
    FOR product IN OUTBOUND user has_bought
    LIMIT 10
    RETURN product
```

AQL

```
FOR user IN users
    FILTER user.name == "alice"
    FOR product, action IN OUTBOUND user has_bought
    SORT action.timestamp DESC
    LIMIT 10
    RETURN product
```

AQL

```
FOR user IN users
    FILTER user.name == "alice"
    FOR recommendation IN 3 ANY user has_bought
        OPTIONS { bfs: true, uniqueVertices: "global" }
    LIMIT 10
    RETURN recommendation
```

AQL

```
FOR user IN users
  FILTER user.name == "alice"
  FOR recommendation, action, path IN 3 ANY user has_bought
    FILTER path.vertices[2].age <= user.age + 5
    AND path.vertices[2].age >= user.age - 5
  LIMIT 10
  RETURN recommendation
```

AQL

```
FOR user IN users
  FILTER user.name == "alice"
  FOR recommendation, action, path IN 3 ANY user has_bought
    FILTER path.vertices[2].age <= user.age + 5
    AND path.vertices[2].age >= user.age - 5
    FILTER recommendation.price < 25
    LIMIT 10
    RETURN recommendation
```

ACID - Transactions

- Invoke a transaction:

```
db._executeTransaction({
  collections: {
    write: ["users", "products"],
    read: "has_bought"
  },
  action: function() {
    // all operations go here
  }
});
```

ACID - Transactions

- Invoke a transaction:

```
db._executeTransaction({
  collections: {
    write: ["users", "products"],
    read: "has_bought"
  },
  action: function() {
    // all operations go here
    throw "failure"; // Triggers rollback
  }
});
```

Benefits & Overhead

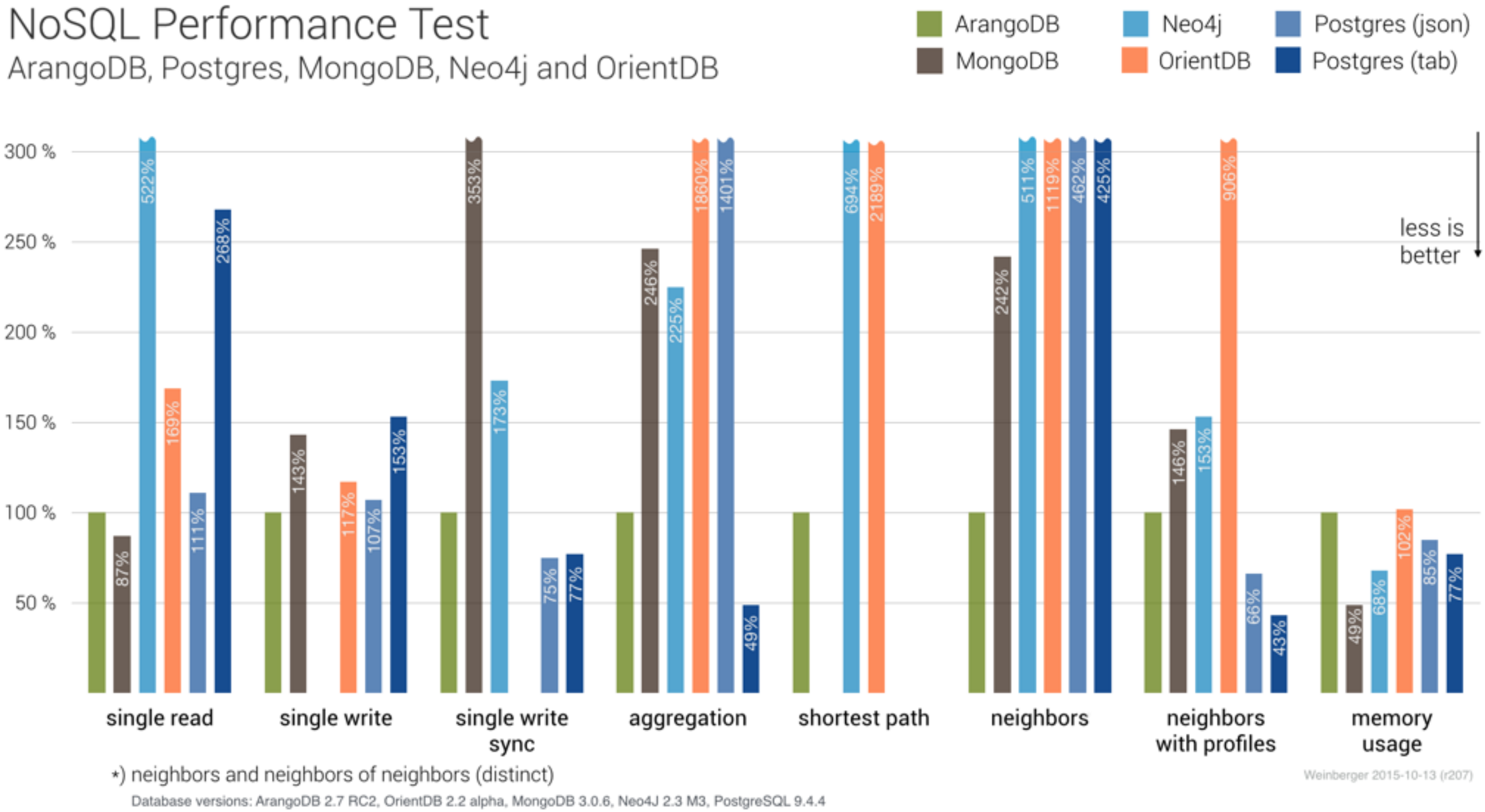
- Native mapping of data into DB
- DB optimized
- Queries are tailored for your data format
- Focus on writing business logic

- Data has to be stored redundantly and has to be kept in sync
- Several technologies involved
- Administration effort is huge

Benefits & Overhead

- Native mapping of data into DB
- DB optimized
- Queries are tailored for your data format
- Focus on writing business logic
- One technology involved

Benchmark Comparison



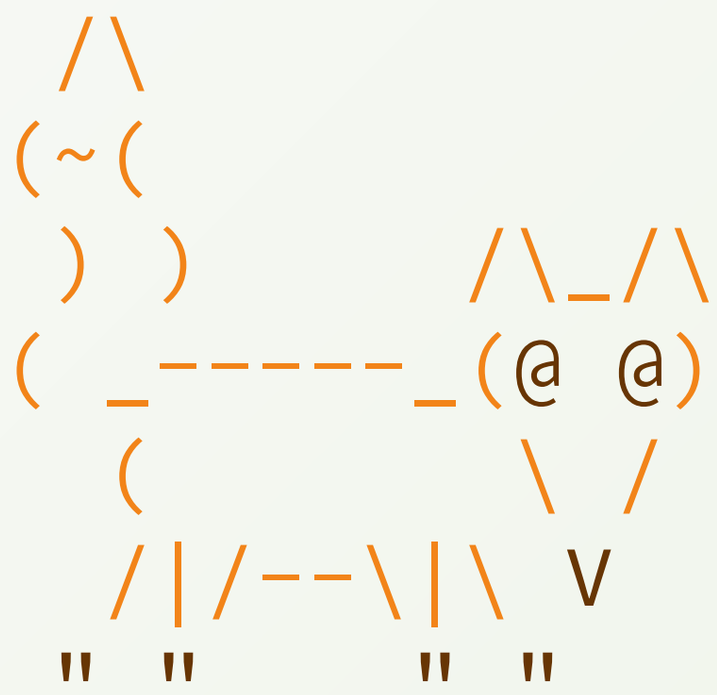
Source: https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/

Foxx

- Add your own customized and versioned REST-API on top of ArangoDB in JavaScript
  - Include as a microservice in Rails, Node.js etc.
  - Ship an administration fronted with it
- Built-in authentication using OAuth2.0 or HTTP-Basic Auth
- Operations are encapsulated in the database
  - low network traffic, direct data access
  - increases data privacy

➡ Multi-device setups

➡ Microservice Architectures



Foxx Example

```
router.get("/recommend/:price", function(req, res) {
  res.json(db._query(
    `FOR user IN users
      FILTER user.name == @name
      FOR recommendation, action, path IN 3 ANY user has_bought
        OPTIONS { bfs: true, uniqueVertices: "global" }
       FILTER path.vertices[2].age <= user.age + 5
        AND path.vertices[2].age >= user.age - 5
        FILTER recommendation.price < @maxPrice
        LIMIT 10
        RETURN recommendation`,
    {
      maxPrice: req.params("price"),
      name: req.session.username
    }).toArray());
});
```

Does it Scale?

- Sharding (Huge Dataset, Write-Scaling)
  - Collection: distributed across several servers
  - Distributed by: List of Attributes (default: _key)
  - Number of Shards immutable
    - Has to be defined on creation
    - Can be different for all collections
    - Suggested: #Servers^2

Does it Scale?

- Sharding (Huge Dataset, Write-Scaling)
  - Collection: distributed across several servers
  - Distributed by: List of Attributes (default: _key)
  - Number of Shards immutable
    - Has to be defined on creation
    - Can be different for all collections
    - Suggested: #Servers^2
- Replication (Failover, Read-Scaling)
  - Shard: can have **n** Followers (Replicas)
  - Followers:
    - do not accept writes
    - synchronous (identical data)
    - placed on different machine (if possible)

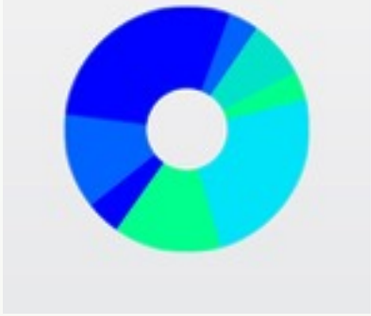
Speaking of Clusters ...

ArangoDB

MESOSPHERE

MESOSPHERE

Apache MESOS™



Marathon

DC/OS

Apache MESOS™

- Connects the cluster
- Ressource management
  - Knows which servers are online
  - Knows which resources they have

Marathon

- Builds on top of Mesos
- Application Monitor
  - Can start new applications
  - "Monitors" running applications
  - If a machine/process dies Marathon restarts it
  - Can stop applications
  - ➡ Can scale up / down running applications within a cluster
- Offers available resources to new applications
- If Marathon dies it is restarted somewhere else

- Builds on top of Marathon
- "Data-Center-Operating-System"
- Offers one-click installers for applications (DC/OS Universe)
- Includes Service Discovery
- Includes Reverse Proxying

➡ You just run an Application through DC/OS and you can communicate with it without ever knowing any IP address

Frameworks

- Marathons little helpers
- Know details about complete Applications
- Communicate with Marathon to order sub-processes
- Can describe additional conditions for processes
  - Like: "Do not run all DB instances on the same Machine"
- Can trigger marathon to scale up/down processes

➡ ArangoDB has a Marathon Framework

Live Demo

ArangoDB on DC/OS

Is Mesosphere required?

- ArangoDB can run clusters without it
  - Setup Requires manual effort (can be scripted):
    - Configure IP addresses
    - Correct startup ordering

- This works:
  - Automatic Failover (Follower takes over if leader dies)
  - Rebalancing of shards
  - Everything inside of ArangoDB
- This is based on Mesos:
  - Complete self healing
  - Automatic restart of ArangoDBs (on new machines)

➡We suggest you have someone on call

Does your data-model scale?

- Rule of thumb: "The more complex the query, the less it scales".
- Indexed-Attribute lookups:
  - Scale "infinitely"
  - Examples
    - Key/Value
    - FOR doc IN sharded FILTER doc.name == 'Michael'
- Joins:
  - Each join operation requires one additional round-trip (find left side, than find right side)
  - Scale "okayish"
- Graphs:
  - Each search depth potentially requires 2 join-like operations
  - A single-server needs to hold a large intermediate result
  - Does not scale too good

An overview of other  ArangoDB features

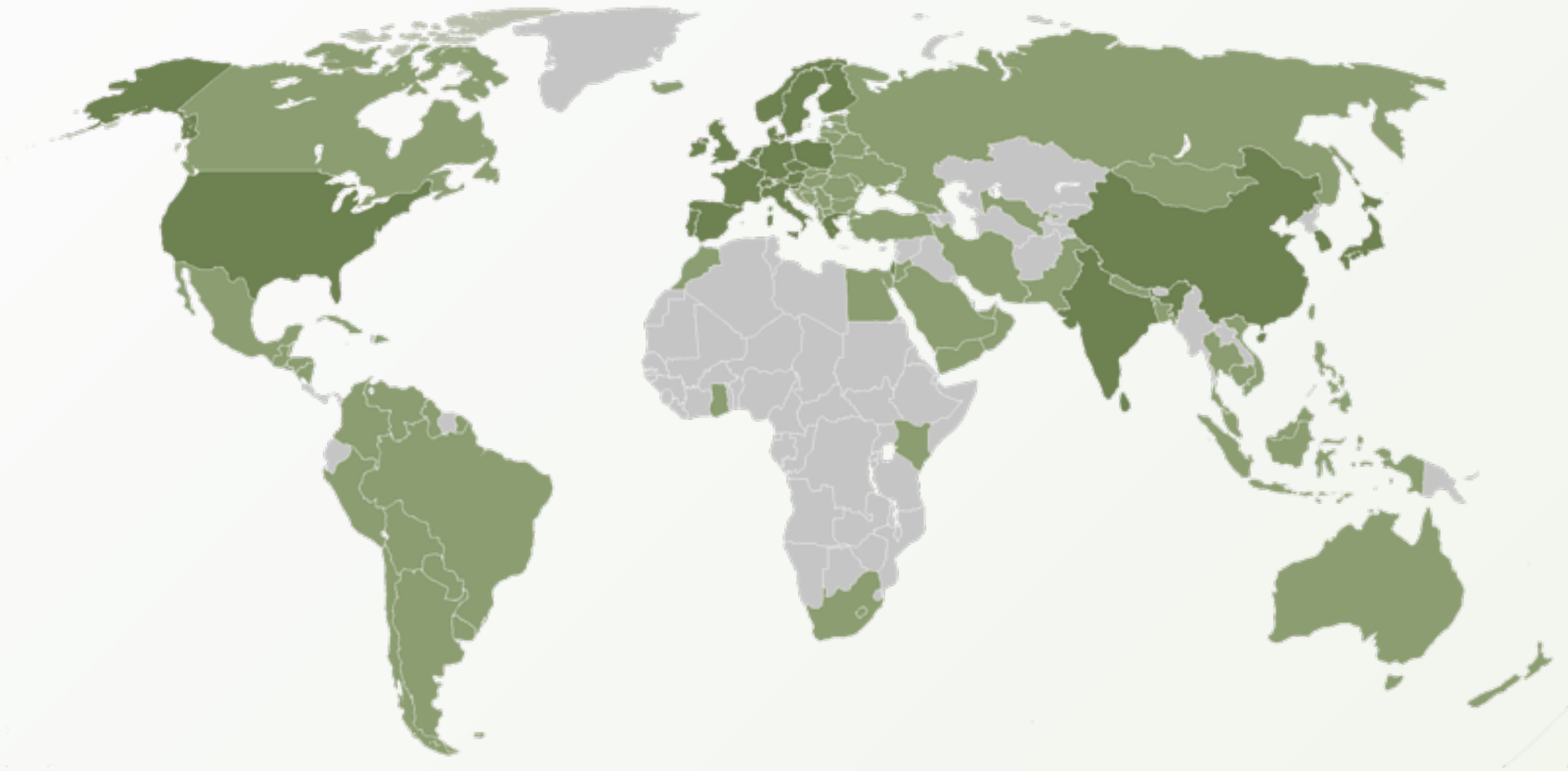
- open source and free (Apache 2 license)
- sharding & replication
- JavaScript throughout (V8 built into server)
- drivers for a wide range of languages
- web frontend
- good & complete documentation
- professional as well as community support

What about Java?

- Basic Driver developed in-house
  - Tutorial: https://www.arangodb.com/tutorials/tutorial-java/
- We are working on a Spring-Data integration
- We are very happy to get input on this
  - Feedback
  - Feature Requests
  - Personal Opinion on other Drivers/Integration you worked with
  - Development

Join our growing community

.. working on the geo index, the full text search and many APIs: Ruby, Python, PHP, Java, D, Javascript, ...



Thank you

- Further questions?
  - Follow us on twitter: @arangodb
  - Join or community slack: arangodb-community.slack.com
  - Join or google group: https://groups.google.com/forum/#!forum/arangodb

- Follow me on twitter/github: @mchacki
- Write me a mail: michael@arangodb.com