Data Models
Key/Value: Amazon Dynamo

Part 3

Thanks to David Maier, M. Grossniklaus & K. Tufte

1
1

Techniques Used in Dynamo

Problem	Techniques	Advantage
Partitioning	Consistent hashing	Incremental scalability
High availability for writes	Vector clocks with reconciliation during reads Buffered writes	Availability over consistency
Handling temporary failures	Sloppy quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background
Membership and failure detection	Gossip-based membership protocol and failure detection	Preserves symmetry and avoids having a centralized registry for storing membership and node liveliness information

2

Data Management in the Cloud

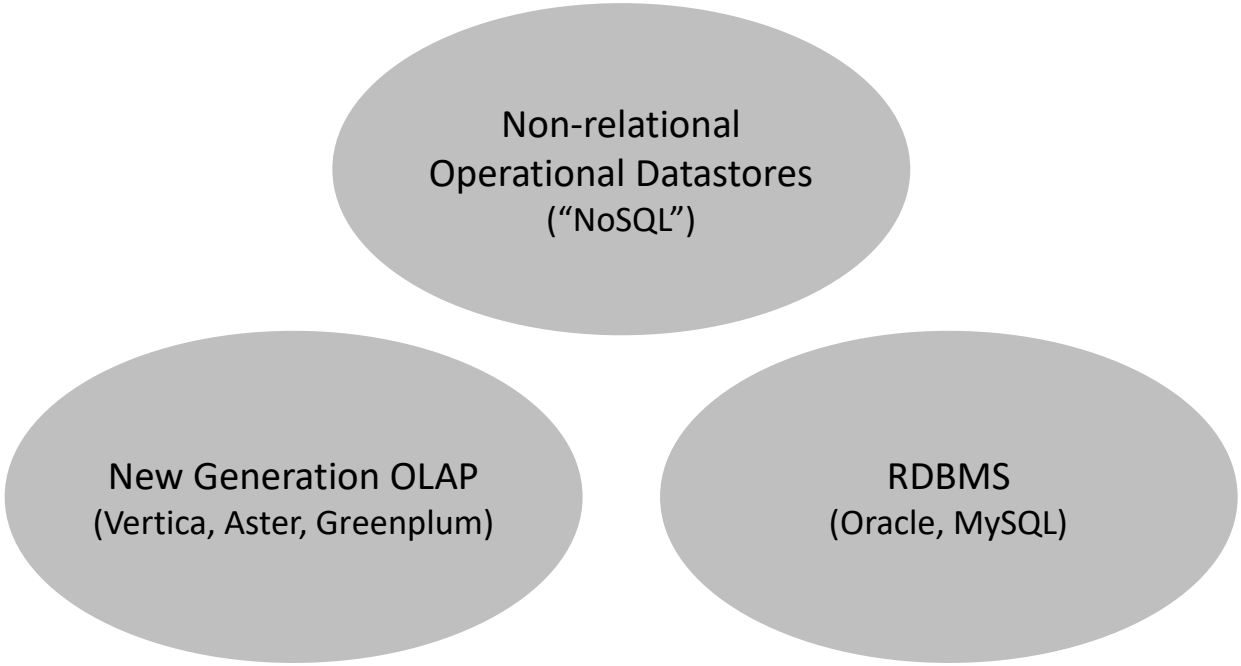
Part 1

Data Models

Document: MongoDB

3

Trend Towards Specialization
* no longer one size fits all *



MongoDB's definition of "NoSQL":
non-relational, next-generation operational datastores and databases

4

4

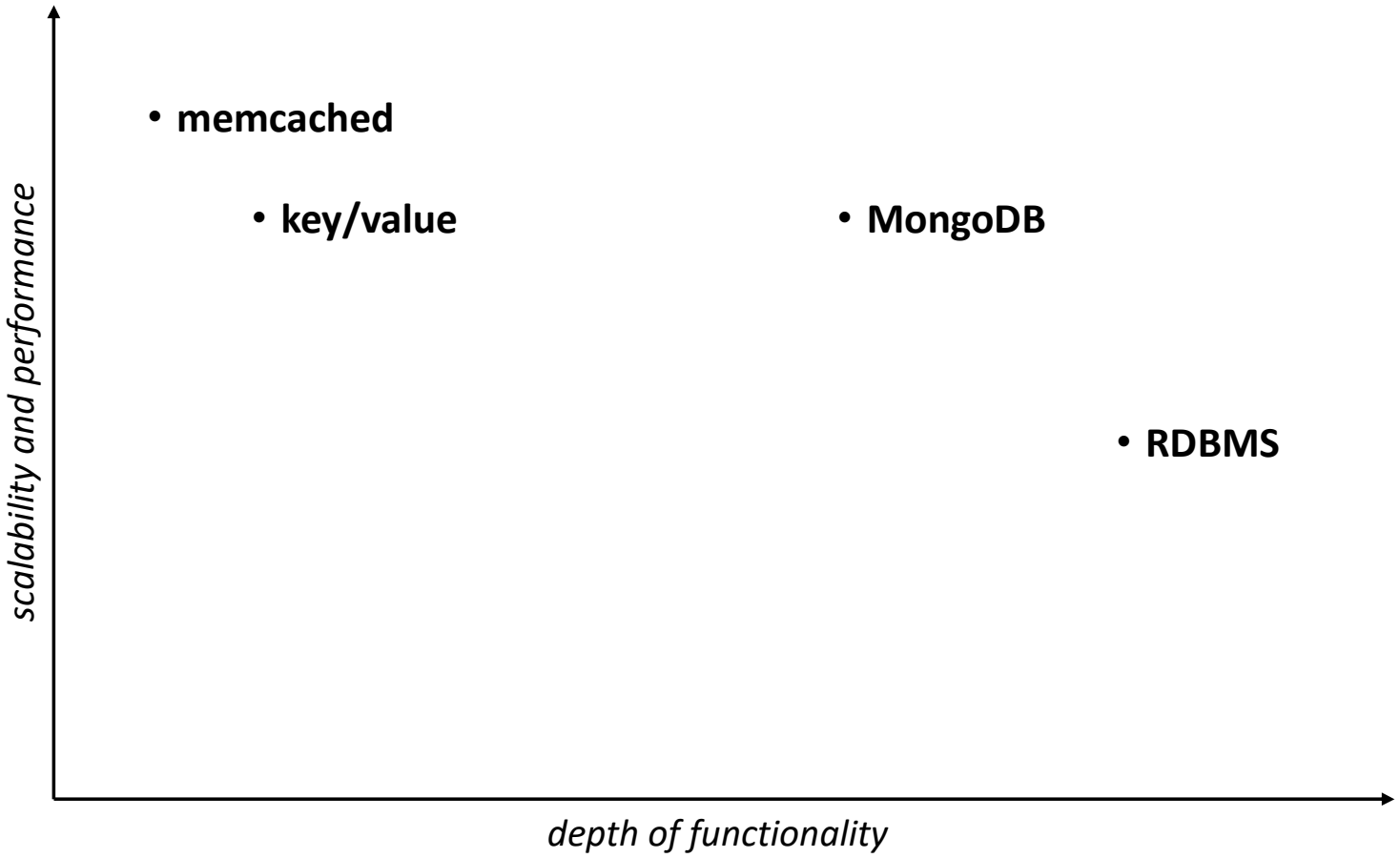
Unifying Theme of NoSQL Systems: Horizontally Scalable Architectures

- No joins + complex multi-row transactions
  - hard to implement both when scaling horizontally
  - abandon the relational data model
- New data models
  - **key/value**: memcached, Dynamo
  - **document-oriented**: MongoDB, CouchDB, JSON stores
  - **tabular**: BigTable
- Improved ways to develop applications?
  - "easier than a relational database management system"
  - **data modeling**: begin with normalized model and extend model by embedding documents

5

5

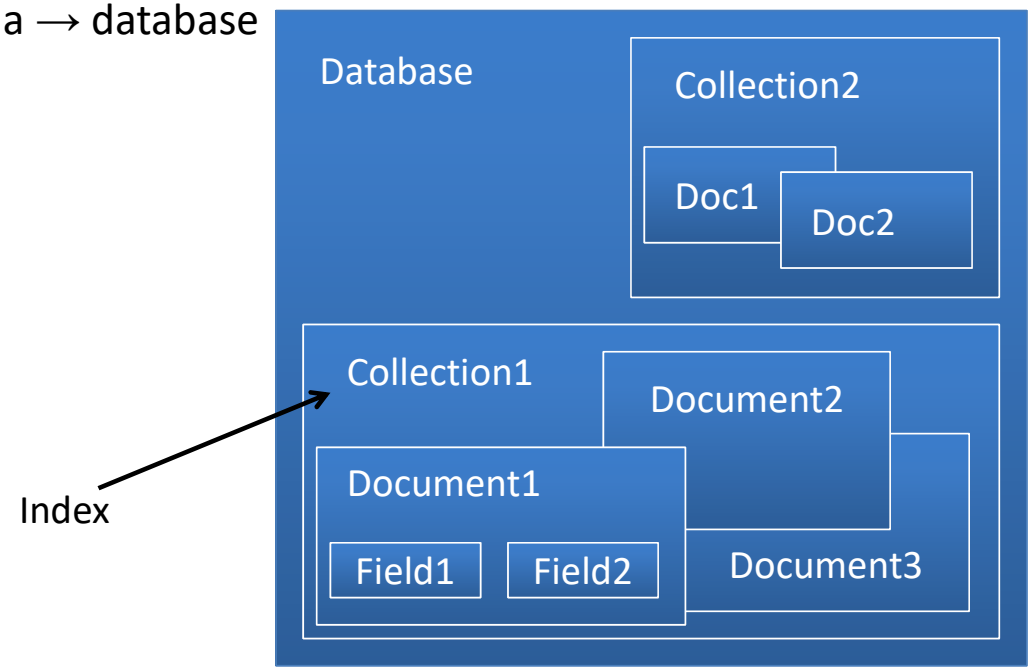
Spectrum of Systems



6

MongoDB vs. RDBMS

- What is missing?
  - joins
  - SQL (but MongoDB has document-based query API)
- Terminology
  - database or schema ⟶ database
  - table ⟶ collection
  - row ⟶ document
  - column ⟶ field



7

MongoDB vs. RDBMS

- "relational databases define columns at the table level whereas a document-oriented database defines its fields at the document level" - http://openmymind.net/mongodb.pdf
  - Table -> Collection; Row -> Document

Shoes Table

Brand	Size	Color
Vans	8	Red/Gold
DC	8	White/Blue/ Green
Nike	8.5	YellowGreen

Shoes Collection

```
{"Brand": "Vans",
 "Size": "8",
 "Color": "Red/Gold"}

{"Brand": "DC",
 "Size": "8",
 "Color": "White/Blue/Green"}

{"Brand": "Nike",
 "Size": "8.5",
 "Color": "YellowGreen"}
```


8

MongoDB vs. RDBMS

- "relational databases define columns at the table level whereas a document-oriented database defines its fields at the document level" - http://openmymind.net/mongodb.pdf
  - Table -> Collection; Row -> Document

Shoes Table

Brand	Size	Color
Vans	8	Red/Gold
DC	8	White/Blue/Green
Nike	8.5	YellowGreen

Shoes Collection

```
{"Brand": "Vans",
 "Size": "8",
 "Color": "Red/Gold",
 "Note": "Looks like mermaid"}

{"Brand": "DC",
 "Size": "8",
 "Color": "White/Blue/Green"}

{"Brand": "Nike",
 "Size": "8.5",
 "Color": "YellowGreen"}
```

9

No Schema

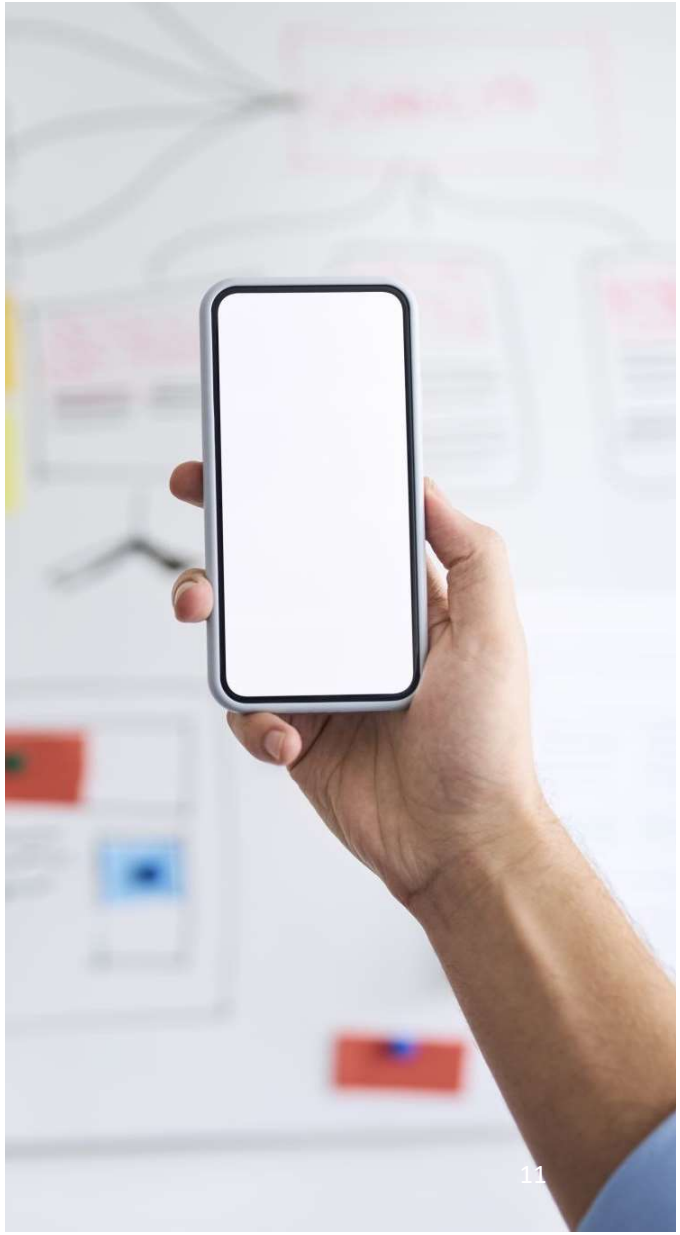
- Schema is not required before inserting data
- Flexible Data Structures:
  - Different documents in the same collection can have different fields
- Schema Validation:
  - Can use validators to enforce specific rules on insert/update operations

10

10

Participation Question

- Your team is designing a **real-time inventory tracking system** for an e-commerce store using MongoDB.
- **Discussion Points:**
  - How does MongoDB's **schema flexibility** benefit this application compared to a structured RDBMS?
  - Are there any challenges that come with allowing flexible schemas?
  - What strategies could you use to enforce some structure while keeping flexibility?



MongoDB Data Model & Features

- Advantages of document model
  - Documents (objects) correspond to native types in many programming languages
  - Embedded documents and arrays reduce need for expensive joins
- High Performance
  - Support for embedded data models reduces I/O activity
  - Indexes; can include keys from embedded documents and arrays
- High Availability
  - Automatic failover
  - Data redundancy
- Automatic Scaling
  - Automatic sharding
  - Replica sets for eventually consistent reads

http://docs.mongodb.org/manual/introduction/

12

12

Data Model

- MongoDB stores JSON-**style** documents
  - internally represented as BSON (binary JSON)
  - `{"hello": "world"}`

    ↓

    `\x16\x00\x00\x00\x02hello\x00`
    `\x06\x00\x00\x00world\x00\x00`
  - General-purpose serialization format
  - light-weight and traversable
  - driver converts programming language objects to BSON
  - document size limited to 16 MB
- Flexible "schemas"
  - the "big difference" between a collection and a table

document size

field type (string)

field name

field value

end of object

both documents can be stored in the same collection

```
{"author": "fred",
 "text": "LOL"}
```

```
{"author": "mary",
 "text": "FYI",
 "tags": ["mongodb"]}
```

this is a list

Source: http://bsonspec.org

13

Storing Documents

```
post = {author: "mike",
        date: new Date(),
        text: "my blog post...",
        tags: ["mongodb", "intro"]}

db.posts.save(post)
```

JavaScript object

tags embedded in post vs. many-to-many join

database

collection

- Example uses JavaScript
  - MongoDB supports JavaScript on the client and server side
  - other language bindings exist
- BSON is the basis for a rich type system
  - e.g., (binary) date values are not possible in JSON
- Collections are created implicitly on demand
  - Created first time a document is saved to that collection

14

14

CRUD Operations

- Create: `insert` (won't overwrite)

```
db.posts.insertOne({author: "Maria",
                    text: "My response ...})
```

- Read: `find` (examples later)
- Update

```
db.posts.updateMany({author: "mike"},
                    {$set: {status: "valid"})
```

- Delete

```
db.posts.deleteMany({status: "reject"})
```


Source: https://docs.mongodb.com/manual/crud/

15

Document Identifiers

- Special field **`_id`**
  - present in all documents: user or system-defined
  - unique across a collection
  - can be any type
- System-defined default **`_id`** is added if not specified by user
  - similar to GUID/UUID, lightweight (only 12 bytes) and fast to generate
  - generated at the client
  - **`ObjectId("4bface1a2231316e04f3c434")`**

timestamp

machine id

process id

incrementing counter

16

16

Queries

- Query evaluation invoked by method **`find()`** on collection
  - posts by author

```
db.posts.find({author: "mike"})
```

- Query language
  - query-by-example plus "$ modifiers": **`$gt`**, **`$lt`**, **`$gte`**, **`$lte`**, **`$ne`**, **`$all`**, **`$in`**, and **`$nin`**
  - age between 20 and 40

```
{author: "mike", age: {$gte: 20, $lt: 40}}
```

- Advanced queries
  - keyword **`$where`**

```
db.posts.find({$where: "this.author == 'mike' ||
                        this.title == 'dbms'"})
```


17

17

Complex Queries

- Queries involving dates
  - posts since April 1

JavaScript starts counting months at 0

```
april1 = new Date(2024, 4, 1)
db.posts.find({date: {$gt: april1}})
```

- Sorting and limit
  - last ten posts

all posts

descending

```
db.posts.find()
        .sort({date: -1})
        .limit(10)
```

  - lazy evaluation (cursors)
  - sort is defined on the cursor instance

18

18

Cursors

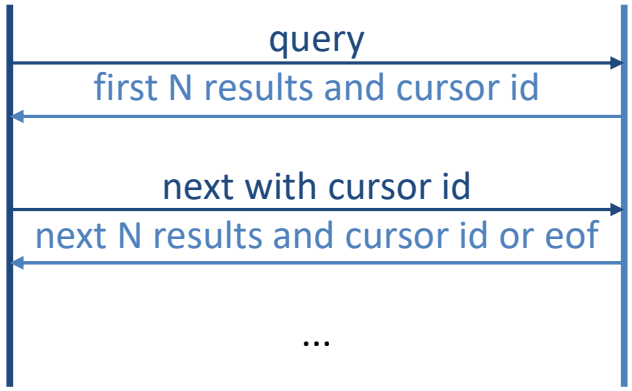
```
var c = db.posts.find({author: "mike"})
                .skip(20)
                .limit(10))

c.next()
c.next()
...
```

omit first 20 results

only return 10 results

next result



19

Queries over Text and Collections

- Use regular expression in find() method
  - posts ending with "Tech"

```
db.posts.find({text: /Tech$/})
```

Regular Expression

- Members of collection-valued attributes are queried in the same way as attributes with atomic values
  - posts with a specific tag

```
db.posts.find({tags: "mongodb"})
```

- Use of a multi-key index speeds up collection-oriented queries
  - `db.posts.ensureIndex({tags: 1})`
  - The '1' means increasing order

20

20

Aggregation Functions

- Counting the number of documents in a collection
  - total posts

```
db.posts.count()
```

  - total posts authored by Mike

```
db.posts.find({author: "mike"}).count()
```

- Method **`distinct()`** displays the distinct values found for a specified fieldin a collection
- Method **`group()`** groups documents in a collection by the specified key and performs simple aggregation
- More ways to compute aggregated values
  - **aggregation framework** using **`aggregate`** command
  - **map/reduce aggregation** for large data sets using **`mapReduce`** command

21

21

Data Models
Key/Value: Amazon Dynamo

Part 3

Thanks to David Maier, M. Grossniklaus & K. Tufte

1
1

Techniques Used in Dynamo

Problem	Techniques	Advantage
Partitioning	Consistent hashing	Incremental scalability
High availability for writes	Vector clocks with reconciliation during reads Buffered writes	Availability over consistency
Handling temporary failures	Sloppy quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background
Membership and failure detection	Gossip-based membership protocol and failure detection	Preserves symmetry and avoids having a centralized registry for storing membership and node liveliness information

2

Data Management in the Cloud

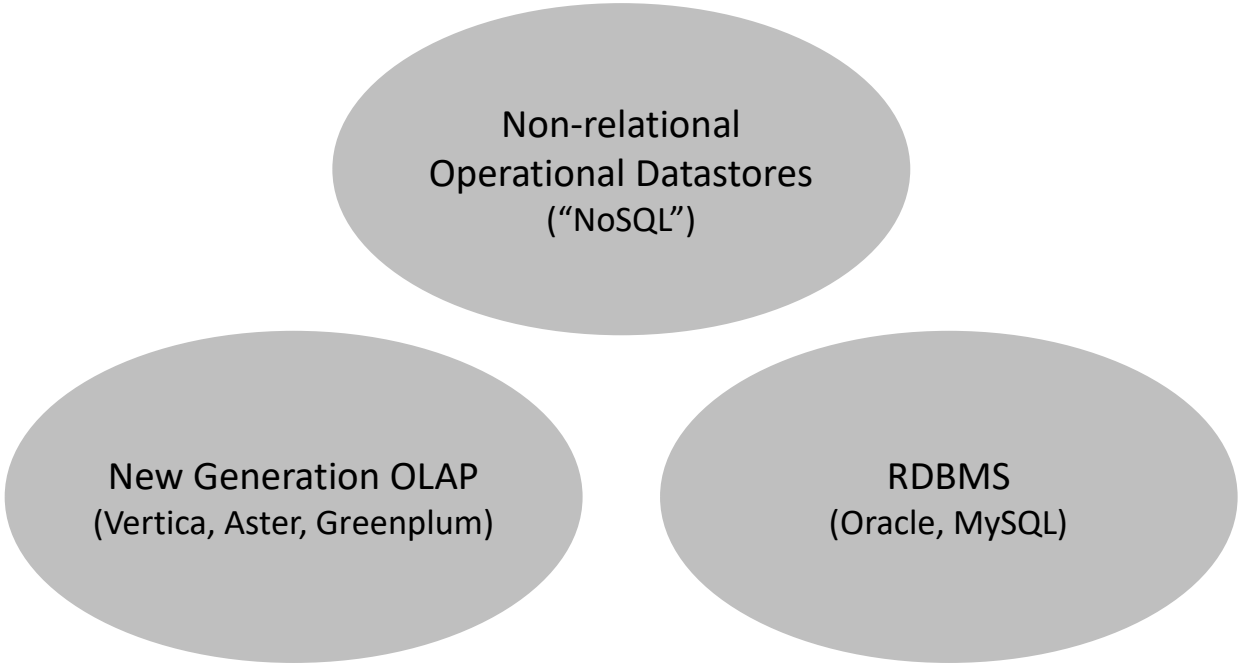
Part 1

Data Models

Document: MongoDB

3

Trend Towards Specialization
* no longer one size fits all *



MongoDB's definition of "NoSQL":
non-relational, next-generation operational datastores and databases

4

4

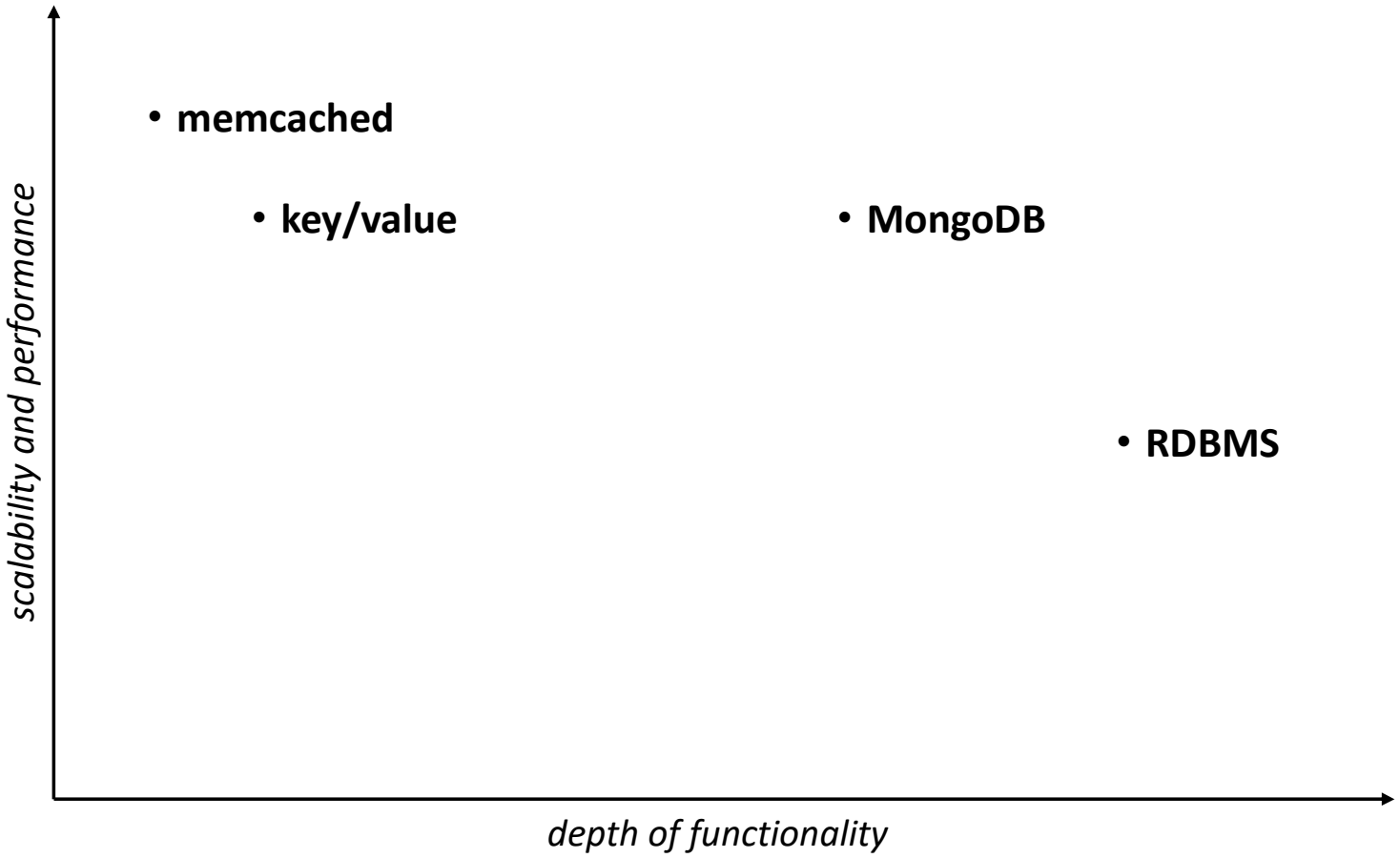
Unifying Theme of NoSQL Systems: Horizontally Scalable Architectures

- No joins + complex multi-row transactions
  - hard to implement both when scaling horizontally
  - abandon the relational data model
- New data models
  - **key/value**: memcached, Dynamo
  - **document-oriented**: MongoDB, CouchDB, JSON stores
  - **tabular**: BigTable
- Improved ways to develop applications?
  - "easier than a relational database management system"
  - **data modeling**: begin with normalized model and extend model by embedding documents

5

5

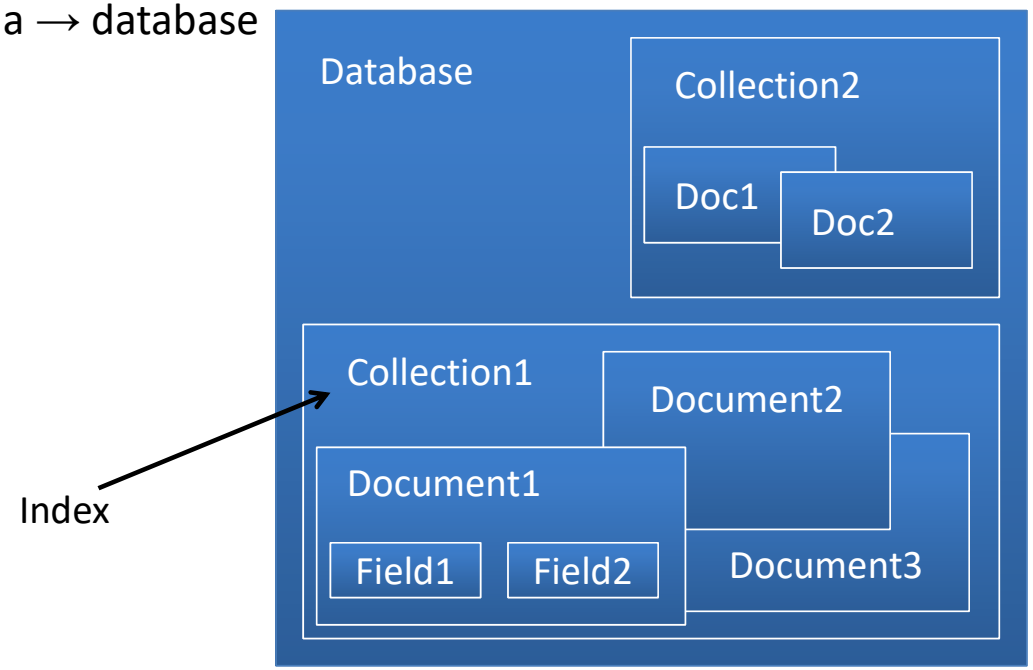
Spectrum of Systems



6

MongoDB vs. RDBMS

- What is missing?
  - joins
  - SQL (but MongoDB has document-based query API)
- Terminology
  - database or schema ⟶ database
  - table ⟶ collection
  - row ⟶ document
  - column ⟶ field



7

MongoDB vs. RDBMS

- "relational databases define columns at the table level whereas a document-oriented database defines its fields at the document level" - http://openmymind.net/mongodb.pdf
  - Table -> Collection; Row -> Document

Shoes Table

Brand	Size	Color
Vans	8	Red/Gold
DC	8	White/Blue/ Green
Nike	8.5	YellowGreen

Shoes Collection

```
{"Brand": "Vans",
 "Size": "8",
 "Color": "Red/Gold"}

{"Brand": "DC",
 "Size": "8",
 "Color": "White/Blue/Green"}

{"Brand": "Nike",
 "Size": "8.5",
 "Color": "YellowGreen"}
```


8

MongoDB vs. RDBMS

- "relational databases define columns at the table level whereas a document-oriented database defines its fields at the document level" - http://openmymind.net/mongodb.pdf
  - Table -> Collection; Row -> Document

Shoes Table

Brand	Size	Color
Vans	8	Red/Gold
DC	8	White/Blue/Green
Nike	8.5	YellowGreen

Shoes Collection

```
{"Brand": "Vans",
 "Size": "8",
 "Color": "Red/Gold",
 "Note": "Looks like mermaid"}

{"Brand": "DC",
 "Size": "8",
 "Color": "White/Blue/Green"}

{"Brand": "Nike",
 "Size": "8.5",
 "Color": "YellowGreen"}
```

9

No Schema

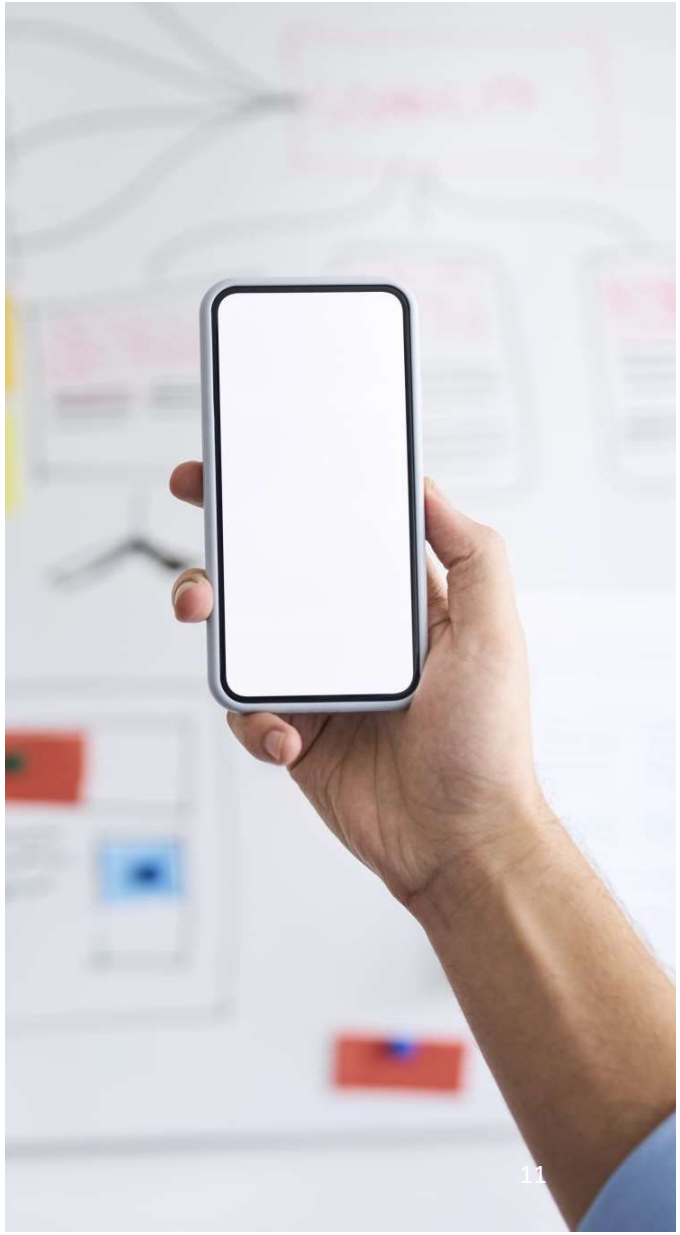
- Schema is not required before inserting data
- Flexible Data Structures:
  - Different documents in the same collection can have different fields
- Schema Validation:
  - Can use validators to enforce specific rules on insert/update operations

10

10

Participation Question

- Your team is designing a **real-time inventory tracking system** for an e-commerce store using MongoDB.
- **Discussion Points:**
  - How does MongoDB's **schema flexibility** benefit this application compared to a structured RDBMS?
  - Are there any challenges that come with allowing flexible schemas?
  - What strategies could you use to enforce some structure while keeping flexibility?



MongoDB Data Model & Features

- Advantages of document model
  - Documents (objects) correspond to native types in many programming languages
  - Embedded documents and arrays reduce need for expensive joins
- High Performance
  - Support for embedded data models reduces I/O activity
  - Indexes; can include keys from embedded documents and arrays
- High Availability
  - Automatic failover
  - Data redundancy
- Automatic Scaling
  - Automatic sharding
  - Replica sets for eventually consistent reads

http://docs.mongodb.org/manual/introduction/

12

12

Data Model

- MongoDB stores JSON-**style** documents
  - internally represented as BSON (binary JSON)
  - `{"hello": "world"}`

    ↓

    `\x16\x00\x00\x00\x02hello\x00`
    `\x06\x00\x00\x00world\x00\x00`
  - General-purpose serialization format
  - light-weight and traversable
  - driver converts programming language objects to BSON
  - document size limited to 16 MB
- Flexible "schemas"
  - the "big difference" between a collection and a table

document size

field type (string)

field name

field value

end of object

both documents can be stored in the same collection

```
{"author": "fred",
 "text": "LOL"}
```

```
{"author": "mary",
 "text": "FYI",
 "tags": ["mongodb"]}
```

this is a list

Source: http://bsonspec.org

13

Storing Documents

```
post = {author: "mike",
        date: new Date(),
        text: "my blog post...",
        tags: ["mongodb", "intro"]}

db.posts.save(post)
```

JavaScript object

tags embedded in post vs. many-to-many join

database

collection

- Example uses JavaScript
  - MongoDB supports JavaScript on the client and server side
  - other language bindings exist
- BSON is the basis for a rich type system
  - e.g., (binary) date values are not possible in JSON
- Collections are created implicitly on demand
  - Created first time a document is saved to that collection

14

14

CRUD Operations

- Create: `insert` (won't overwrite)

```
db.posts.insertOne({author: "Maria",
                    text: "My response ...})
```

- Read: `find` (examples later)
- Update

```
db.posts.updateMany({author: "mike"},
                    {$set: {status: "valid"})
```

- Delete

```
db.posts.deleteMany({status: "reject"})
```


Source: https://docs.mongodb.com/manual/crud/

15

Document Identifiers

- Special field **`_id`**
  - present in all documents: user or system-defined
  - unique across a collection
  - can be any type
- System-defined default **`_id`** is added if not specified by user
  - similar to GUID/UUID, lightweight (only 12 bytes) and fast to generate
  - generated at the client
  - **`ObjectId("4bface1a2231316e04f3c434")`**

timestamp

machine id

process id

incrementing counter

16

16

Queries

- Query evaluation invoked by method **`find()`** on collection
  - posts by author

```
db.posts.find({author: "mike"})
```

- Query language
  - query-by-example plus "$ modifiers": **`$gt`**, **`$lt`**, **`$gte`**, **`$lte`**, **`$ne`**, **`$all`**, **`$in`**, and **`$nin`**
  - age between 20 and 40

```
{author: "mike", age: {$gte: 20, $lt: 40}}
```

- Advanced queries
  - keyword **`$where`**

```
db.posts.find({$where: "this.author == 'mike' ||
                        this.title == 'dbms'"})
```


17

17

Complex Queries

- Queries involving dates
  - posts since April 1

JavaScript starts counting months at 0

```
april1 = new Date(2024, 4, 1)
db.posts.find({date: {$gt: april1}})
```

- Sorting and limit
  - last ten posts

all posts

descending

```
db.posts.find()
        .sort({date: -1})
        .limit(10)
```

  - lazy evaluation (cursors)
  - sort is defined on the cursor instance

18

18

Cursors

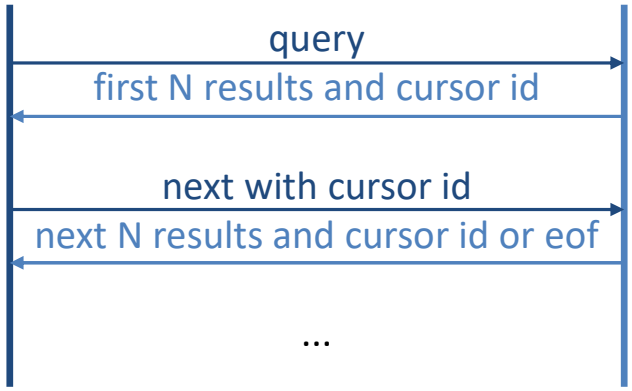
```
var c = db.posts.find({author: "mike"})
                .skip(20)
                .limit(10))

c.next()
c.next()
...
```

omit first 20 results

only return 10 results

next result



19

Queries over Text and Collections

- Use regular expression in find() method
  - posts ending with "Tech"

```
db.posts.find({text: /Tech$/})
```

Regular Expression

- Members of collection-valued attributes are queried in the same way as attributes with atomic values
  - posts with a specific tag

```
db.posts.find({tags: "mongodb"})
```

- Use of a multi-key index speeds up collection-oriented queries
  - `db.posts.ensureIndex({tags: 1})`
  - The '1' means increasing order

20

20

Aggregation Functions

- Counting the number of documents in a collection
  - total posts

```
db.posts.count()
```

  - total posts authored by Mike

```
db.posts.find({author: "mike"}).count()
```

- Method **`distinct()`** displays the distinct values found for a specified fieldin a collection
- Method **`group()`** groups documents in a collection by the specified key and performs simple aggregation
- More ways to compute aggregated values
  - **aggregation framework** using **`aggregate`** command
  - **map/reduce aggregation** for large data sets using **`mapReduce`** command

21

21