What is Big Data

Big Data

- Collection of dataset with:
    - High in Volume, velocity, variety — Difficult to store & analyze using traditional data processing tools.
- In recent years, exponential growth in data
- A recent report by DOMO estimates the amount of data generated every minute:
    - Facebook users share nearly 4 million pieces of content every minute
    - Twitter users send nearly 300,000 tweets every min.
    - Amazon receives over 4000 new visitors
    - Instagram users like nearly 1.73 million photos
    - Netflix subscribers stream nearly 77,000 hours of video
    - Youtube users upload 300 hours of new video content

Characteristics of Big Data

- Volume
- Velocity
- Variety
- Veracity — Refers to the accuracy of data. Data needs to be cleaned.
- Value — Extracting the value out of the data

Big Data Examples

- Data generated by social networks
- Click-stream data generated by Web application
  - Such as e-Commerce to analyze user behavior
- Machine sensor data collected from sensors
- Healthcare data collected in EHR
- Logs generated by application
- Stock market data

Big Data

- Traditionally users to know what kind of questions they wanted to ask beforehand.
  - Determined how data was stored, collected
- Big Data — Organizations are collecting data before know what questions they will be asking.
  - Difficult to transform every piece of data
  - Stored in the form it was received

Big Data Analytics Flow

- Data Collection
  - Data collection depends upon the source
  - Before data is analyzed — needs to be ingested
- Data Preparation
  - Data is often dirty (corrupt/missing values)
  - Cleaning, wrangling, de-duplication, normalizing, etc.

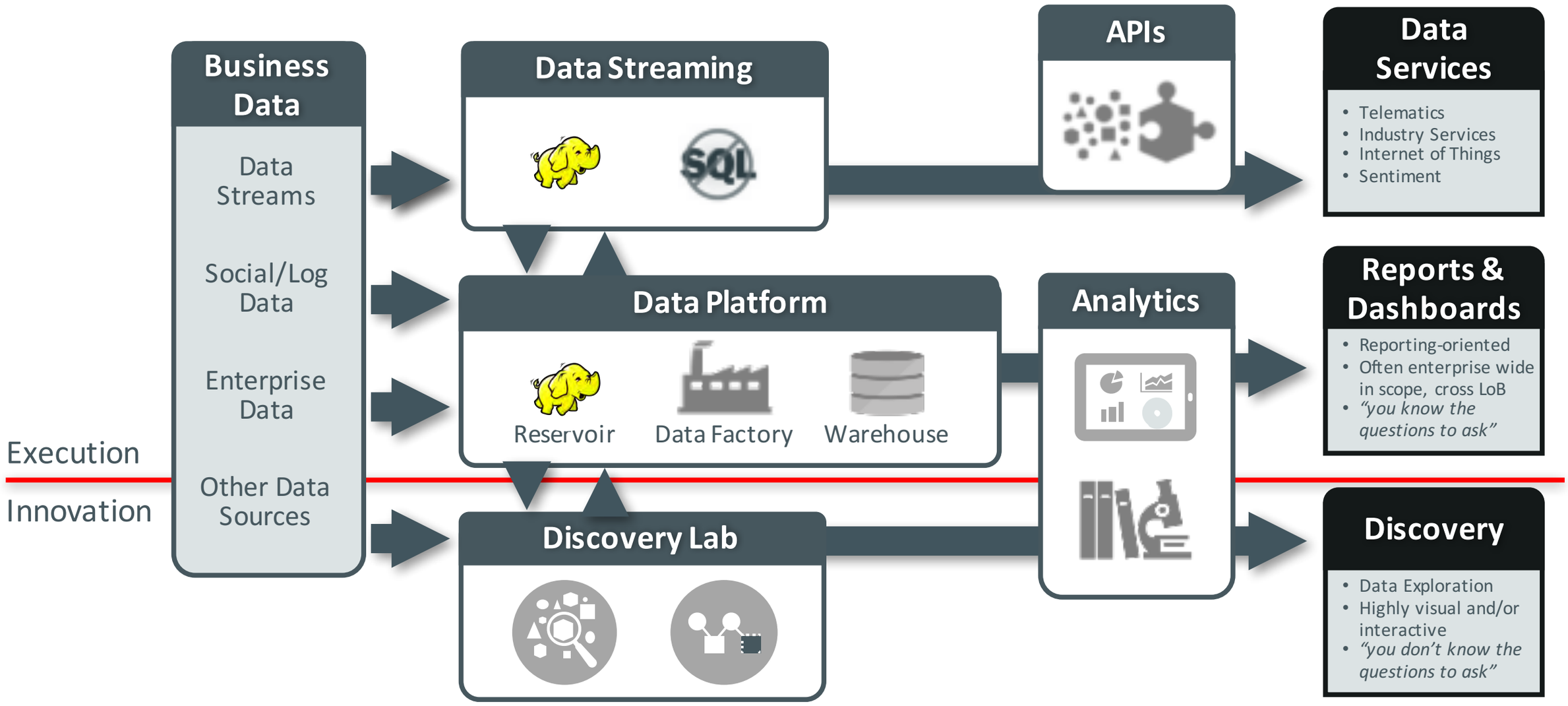
Big Data Analytics Flow

- Analysis Types
  - Need to determine the analysis types for applications
  - Understanding the data and what you want to get out of it
- Analysis Modes
  - Batch, real-time — depends upon requirement of application
- Visualization

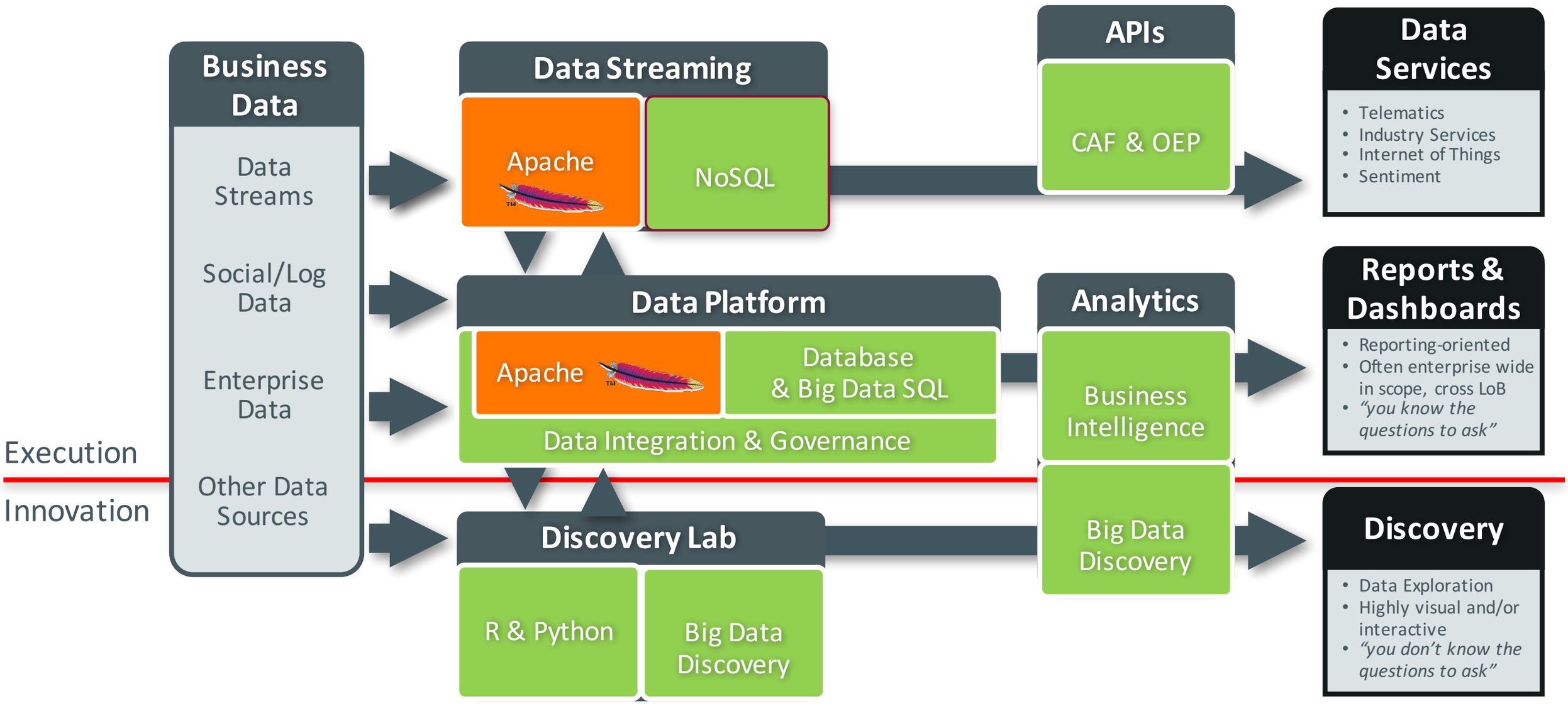
Big Data in the Cloud

- Customers are innovating their businesses with new data.
- Cloud providers are enabling customers:
  - To focus on creating new data-driven services, and data-centric products
    - Instead of having to focus on just the raw technology nuts and bolts underneath them.
- Most of the data is outside of the enterprise
  - Data movement is still very costly and time consuming

Big Data Architecture



Comprehensive Cloud Solution for Big Data



NoSQL Databases

Limitations of RDBMS

- Lage data sizes
- Semi-structured and un-structured data
- Scalability — How to scale up with large data

NoSQL Movement: What is it all about?

- NoSQL is term for a movement in database design away from traditional relational database models.

- With the emergence of big data and cloud computing

  - Traditional databases and schema driven design becoming constraining.

Motivation for NoSQL

- Scalability
- Schema-less data storage
- Quick data storage and traversal
- Easier to program
- Better performance
- Easily distributed
- Not a replacement for RDBMS
    - Complements it

Popular NoSQL Databases

- Key-value Store
- Document Databases
- Column Family
- Graph databases

Key-value store

- Key / Value store databases allow for values to be associated with and looked up by a key.
- Keys can be associated with more than one value.
- Value is a blob without the knowledge of what is inside
  - Application is responsible for understanding what is stored
- Data can be stored in the native data type of a particular programming language.
- Amazon DynamoDB

Key-Value Database

- Key-Value databases are the simplest of the NoSQL
- Data store is based upon storing data identifier called keys
- Think of arrays

- Has keys and values — but has some limitations
  - Limitations:
    - ✓ Index are always integer
    - ✓ Values are all of the same type

Key-Value Database

- Associative Arrays
  - Shares some characteristics of array but fewer limitations
  - Keys and Values can vary

- Key-Value database builds on the concept of Associated array.

Essential Features of Key-Value Databases

- Simplicity
  - Flexible and forgiving
    - Assign wrong types of data (good & bad)
- Speed
  - Simple associative array data structure and design make it efficient
- Scalability
  - Ability to add/remove servers without impacting the application.

Graph Database

- Graph databases store all of their information in nodes (vertices) and edges.
- Graph traversal is how you “query” the database.
- Relationship information about nodes is stored in the edges.

Column Family Databases

- Column — stores a single value in the DB
- Column families — are collection of related columns
  - Columns frequently used together should be grouped into the same column family
    - Each row in the column family is uniquely identified by a row key
- Keyspace — top-level data structure, other data structure contains within the Keyspace

Column Family Databases

Street	City	State	Province	Zip	Postal Code	Country
178 Main St.	Boise	ID		83701		U.S.
89 Woodridge	Baltimore	MD		21218		U.S.
293 Archer St.	Ottawa		ON		K1A 2C5	Canada
8713 Alberta DR	Vancouver		BC		VSK 0AI	Canada

Distributed Databases

- Often times, as databases grow larger, it is necessary to expand the hardware powering them

- Distributed databases take advantage of cheaper hardware by having multiple computers work together rather than building one large machine.

- NoSQL DBs are commonly designed to use distributed servers

  - Not a requirement

  - There are some issues managing data in a distributed environment

Scaling Out

- RDBMS are usually “scaled up” by adding hardware or processing power
- NoSQL is “scaled out” by spreading the load
- Sharding divides the data inside the database and partitions pieces of it to different nodes.
- Databases can be sharded horizontally (by rows) or vertically (by columns).

Sharding

Customer

Customer	Name
123	Mary
456	John
999	Peter

Orders

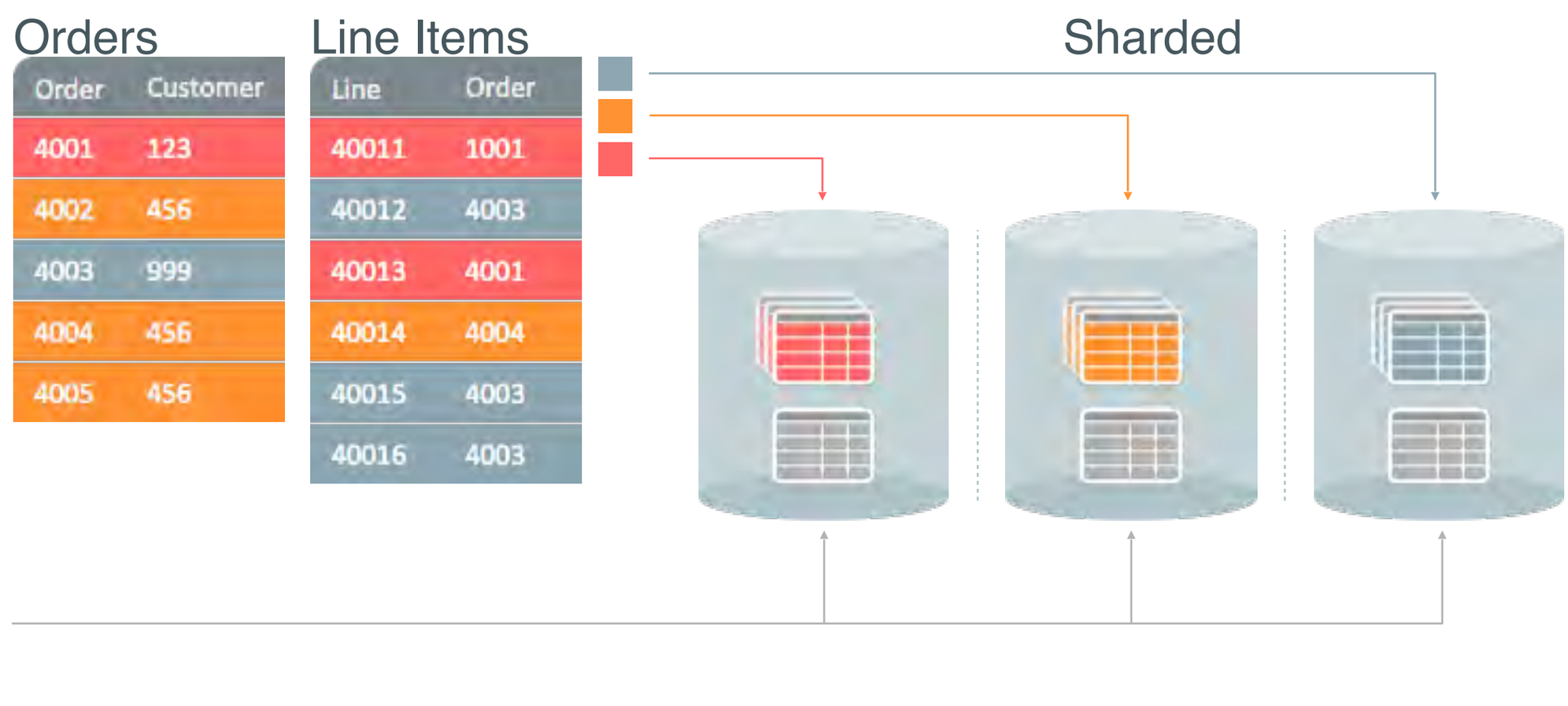
Order	Customer
4001	123
4002	456
4003	999
4004	456
4005	456

Line Items

Line	Order
40011	1001
40012	4003
40013	4001
40014	4004
40015	4003
40016	4003

Products

SKU	Product
100	Coil
101	Piston
102	Belt



Summary

- NoSQL is a movement away from RDBMS
- NoSQL is easier for programming and more efficient
- Schema-less leans favorably to semi and un-structured data
- RDBMS still plays a pivotal role in Enterprises datastore
  - RDBMS vendors adding features such as sharding for enhance scalability

Document DB

Design Philosophy

- Not everything for everyone
- Database that worked with documents rather than rows
- Fast
- Scalable
- Easy to Use

DocumentDB is a cross platform schema-less, document-oriented NoSQL database that provides easy horizontal scalability

Essential Elements of DocumentDB

- Document
- Collections

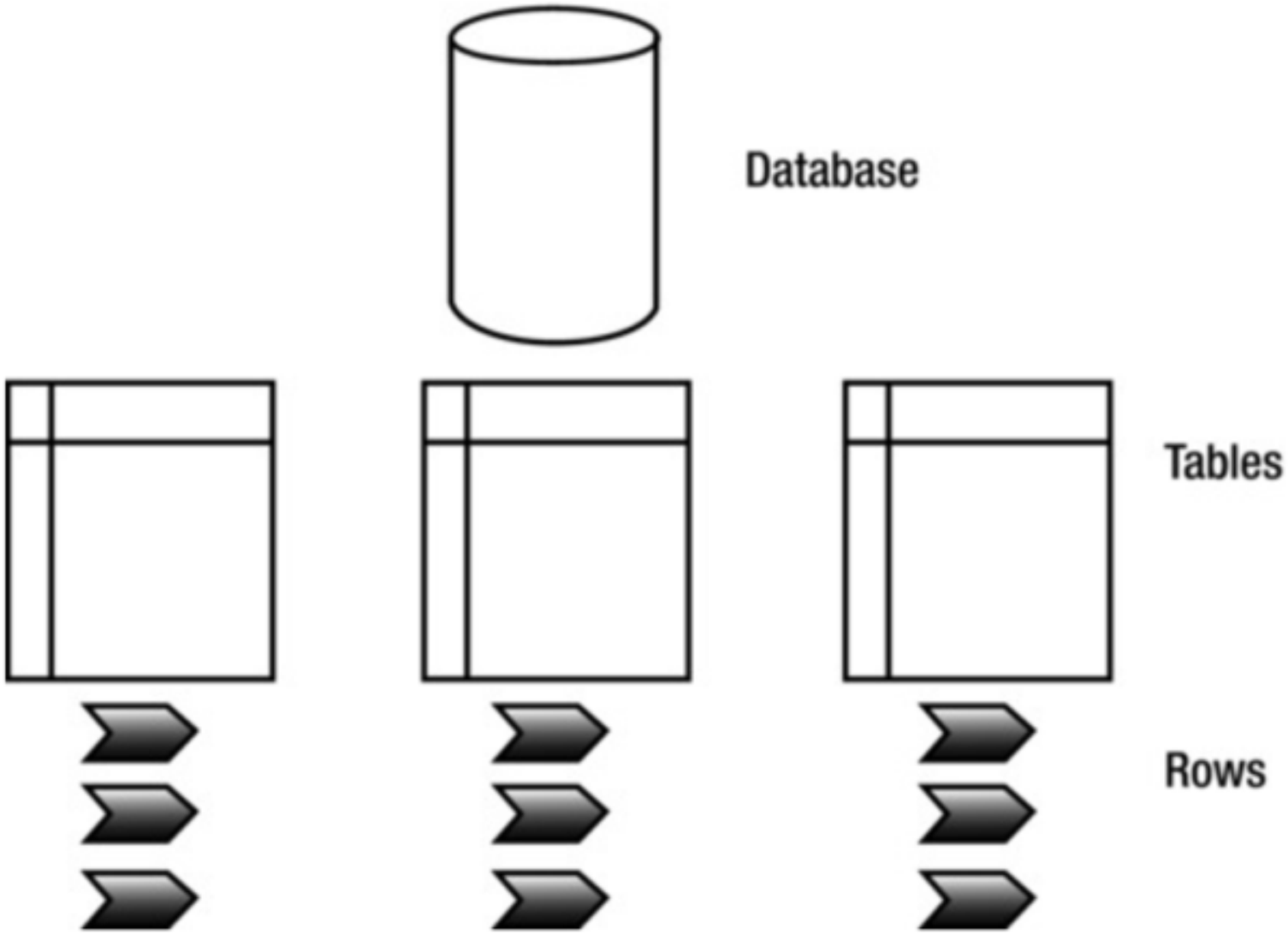
What is a Document

- A record in MongoDB is a document
  - Composed of fields and value pairs
  - Similar to JSON objects.
    - Value of fields may include other documents, arrays and arrays of documents
      - Big differentiation from RDBMS

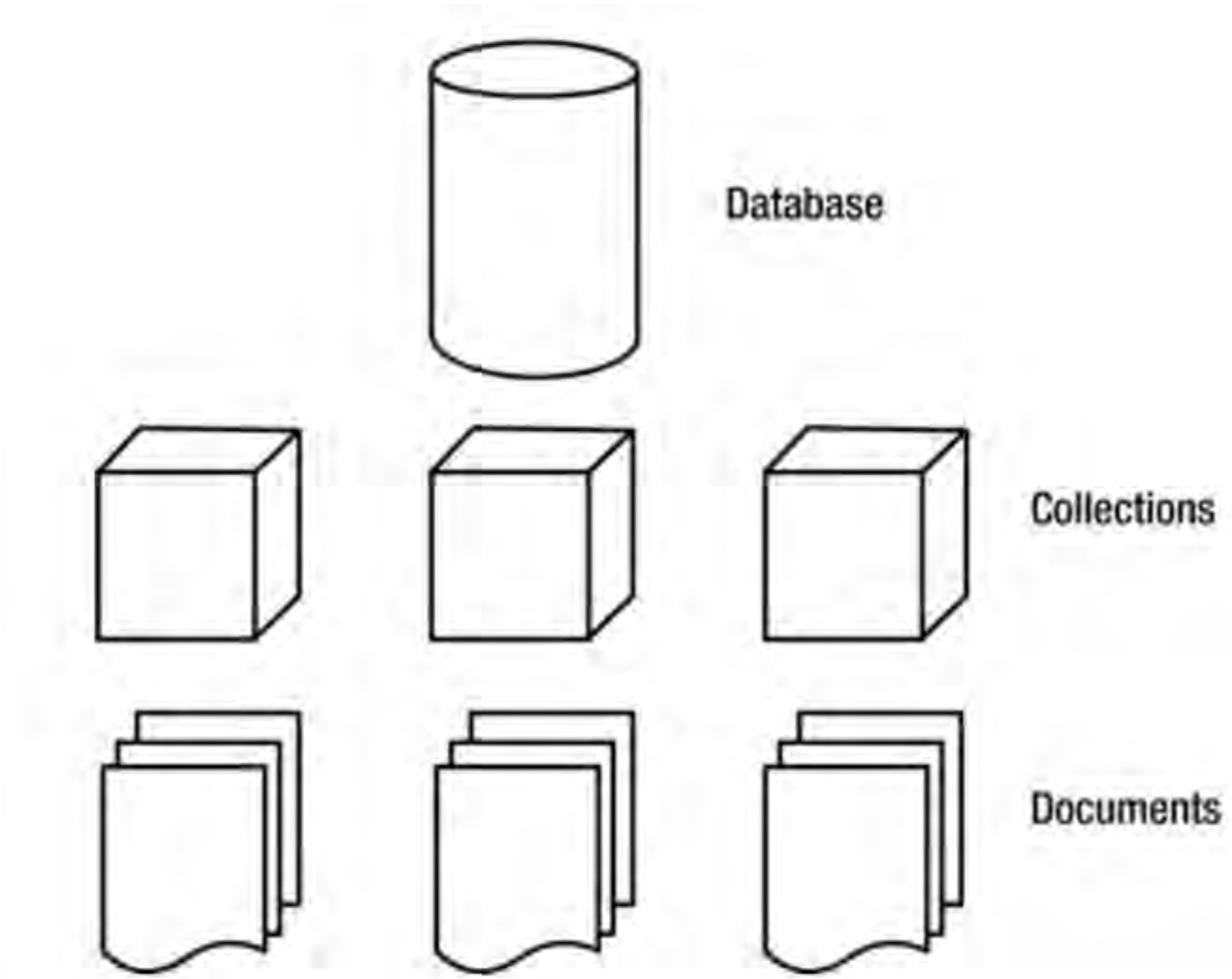
What are Collections?

- MongoDB stores documents in collections.
- Collections are analogous to tables in relational databases.
- In RDMS all tables in a database must have the same schema,
  - In MongoDB there is no such requirement.
  - A collection may store documents that are not same in structure
  - This schema-less design
  - Documents stored in a collection must have a unique _id field that acts as a primary key.
- Documents in a collection can be stored either in Normalized form or embedded into another documents.

Relational Database Model



DocumentDB Database Model

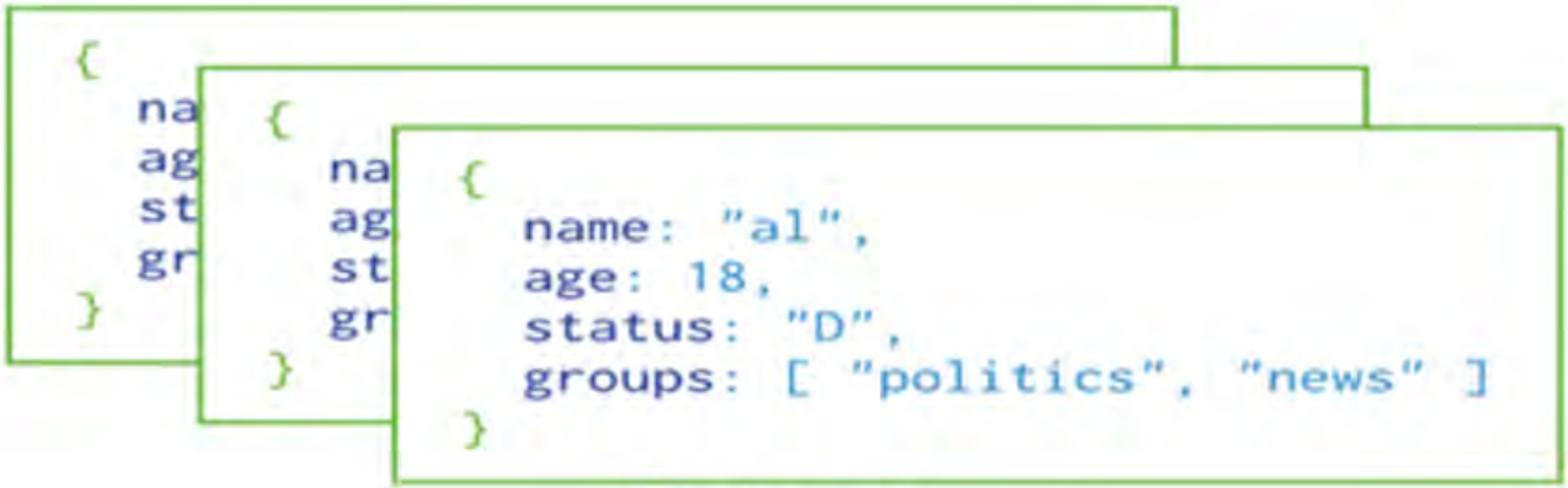


RDBMS vs DocumentDB

- The general structure is the same between the two types of databases;
- Not use them in even remotely similar manners.
- There are several types of collections in MongoDB.
- The default collection type is expandable in size:
- It’s also possible to define collections that are capped.
    - These capped collections can only contain a certain amount of data before the oldest document is replaced by a newer document

RDBMS vs DocumentDB

RDBMS	MongoDB
Table	Collection
Column	Key
Value	Value
Records / Rows	Document / Object



Collection

JSON

- JSON is more than a great way to exchange data
  - Nice way to store data
- RDBMS is highly structured, with multiple tables that store the individual pieces
  - MongoDB stores everything together in a single document
- JSON describes all the content in a given document
  - No need to specify the structure of the document in advance
  - JSON is effectively schema less
  - JSON also provides excellent performance by keeping all of the related data in place

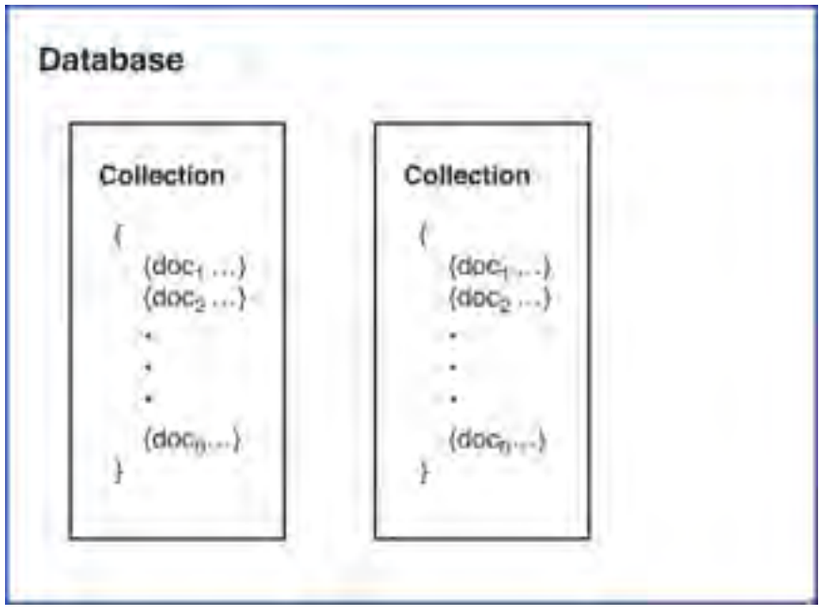
```
{
    "Type": "CD",
    "Artist": "Nirvana",
    "Title": "Nevermind",
    "Genre": "Grunge",
    "Releasedate": "1991.09.24",
    "Tracklist": [
        {
        "Track" : "1",
        "Title" : "Smells Like Teen Spirit",
        "Length" : "5:02"
        },
        {
        "Track" : "2",
        "Title" : "In Bloom",
        "Length" : "4:15"
        }
    ]
}
```

BSON

- MongoDB does not use JSON to store data
- MongoDB stores data in the form of BSON -Binary encoded JSON documents which supports a rich collection of types.
- Fields in BSON documents may hold arrays of values or embedded documents.
- In MongoDB, the database construct is a group of related collections.
- Each database has a distinct set of data files and can contain a large number of collections.
- A single MongoDB deployment may have many databases.
- BSON also adds a couple of features that aren't available in standard JSON
    - Ability to add types for handling binary data

Database Container

- Highest level of logical structure in document DB
- DB Container is referred to as “db”
- db is prefixed to the collection name for basic operations



Basic Operations

Inserting a document

- Collections can perform a number of different operations.
- The insert method adds documents to a collection.
- For example, the following adds a single document describing a book by Malcolm Gladwell to the books collection:

db.books.insert({"title":" The Tipping Point", "author": "Malcolm Gladwell."})

Inserting a document

- Better option is to include unique identifier:

  db.books.insert({book_id: 051717,"title":" The Tipping Point", "author": "Malcolm Gladwell."})

- MongoDB would add unique identifier if not provided
- Efficient to bulk insert instead of individual inserts:

```
db.books.insert(
  [
            {"book_id": 1298747,
             "title":"Mother Night",
             "author": "Kurt Vonnegut, Jr."},

            {"book_id": 639397,
             "title":"Science and the Modern World",
             "author": "Alfred North Whitehead"},

            {"book_id": 1456701,
             "title":"Foundation and Empire",
             "author": "Isaac Asimov"}
  ]
```


Basic Operations

```
db.books.remove({"book_id": 051717})

db.books.update ({"book_id": 051717},
                 {$inc  {"quantity" : 5 }})

db.books.find({"author": "Malcolm Gladwell."},
{"title" : 1} )
```

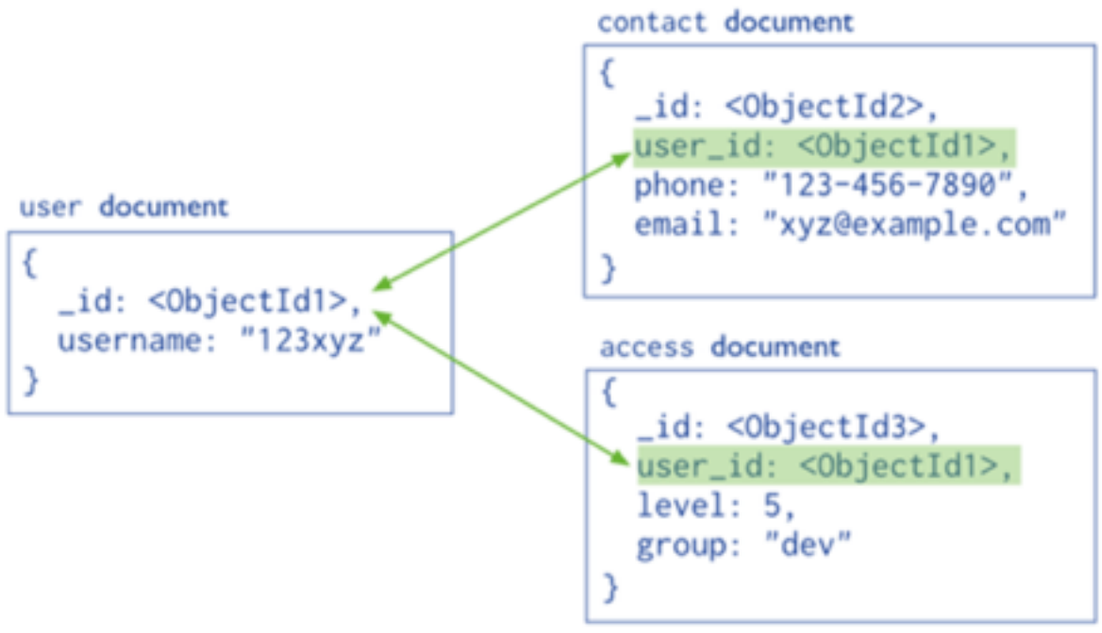

Designing for Document DB

Embedding vs Referencing Info in Document

- Embedding
    - Means that you place a certain type of data into the document itself.
- Referencing
    - Create a reference to another document that contains that specific data

Normalized Data Models

- The relationships between data is stored by links (references)
  - One document to another
  - References are resolved by the application



Normalized Data Models

- Need to perform two lookups

```
{
order_item_ID : 834838,
   order_ID: 8827,
   quantity: 3,
   cost_per_unit: 8.50,
   product_ID: 3648
}
```

```
{
   product_ID: 3648,
   product_description: "1 package laser printer paper.
     100% recycled.",
   product_name : "Eco-friendly Printer Paper",
   product_category : "office supplies",
   list_price : 9.00
}
```


Denormalized

```
 {
order_item_ID : 834838,
   order_ID: 8827,
   quantity: 3,
   cost_per_unit: 8.50,
   product :
        {
            product_description: "1 package laser printer
              paper. 100% recycled.",
            product_name : "Eco-friendly Printer Paper",
            product_category : "office supplies",
            list_price : 9.00
        }
   }
```

- No longer need to maintain product_ID field
    - Used as DB reference in the Order_Items document

Denormalized

- Keep frequently used data together in a document
- Minimize persistence storage reads
- Unnecessary denormalizing results in large document
  - Fewer documents stay in memory
- How much is enough?
  - Depends upon queries

Mutable Documents

- Take an example of trucks in a company fleet transmit location, fuel consumption, etc. every 3 mins

- One option — create new document every time

```
{   truck_id: 'T87V12',
    time: '08:10:00',
    date :  '27-May-2015',
    driver_name: 'Jane Washington',
    fuel_consumption_rate: '14.8 mpg',
}
```

- Truck id, date, and driver name is same for one day

Mutable Document: Embedded

```
{
   truck_id: 'T87V12',
   date :  '27-May-2015',
   driver_name: 'Jane Washington',
   operational_data:
            [
               {time : '00:01',
                fuel_consumption_rate: '14.8 mpg',
                ...},
                {time : '00:04',
               fuel_consumption_rate: '12.2 mpg',
               ...},
                {time : '00:07',
               fuel_consumption_rate: '15.1 mpg',
               ...},
            ...]
}
```


Performance Issue

- DB Management system allocates certain amount of space
  - Usually enough to fit the document as it exists
  - If document grows larger than the size
    - Must be moved to another location.
  - Allocate sufficient space for the document at the creation time

Preconfigured document

200 Embedded Documents with Default Values

```
{truck_id: 'T8V12'
 date: '27-May-2015'
 operational_data:
    [{time: '00 : 00',
      fuel_consumption_rate: 0.0}
     {time: '00 : 00',
      fuel_consumption_rate: 0.0}
      .
      .
      .
      .
    {time: '00 : 00',
     fuel_consumption_rate: 0.0}
     ]
```

Indexing

- Read-heavy — High percentage of read operations
  - Mostly analytical application fall into this category
  - Should have indexes on all the fields used to filter results
- Write-Heavy — High percentage of write operations
  - E.g., Truck sensor data would fall in write heavy
  - Minimize indexes
    - Every insert — indexes has to be created, and updated