NOSQL Databases

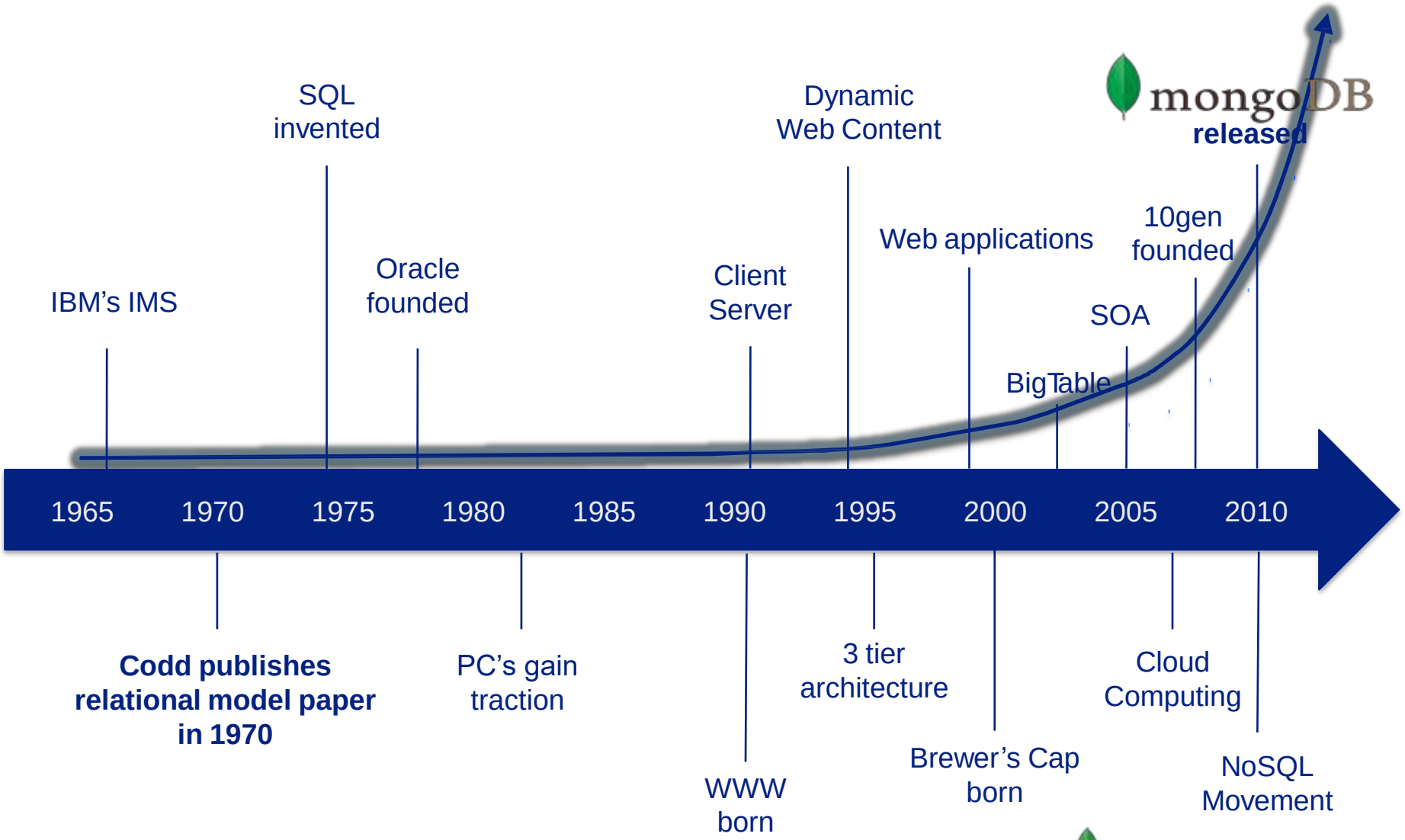
High performance. Delivered.

consulting | technology | outsourcing

Java 8 Goals

- What is a NoSQL database?
- NoSQL vs. SQL
- Types and examples
- When to use NoSQL
- Microservices, polyglot persistence and NoSQL
- NoSQL and Cloud

Dawn of Databases to Present



3

Relational Database Strengths

- Data stored in a RDBMS is very compact (disk was
- more expensive)
- SQL and RDBMS made queries flexible with rigid schemas
- Rigid schemas helps optimize joins and storage
- Massive ecosystem of tools, libraries and integrations
- Been around 40 years!

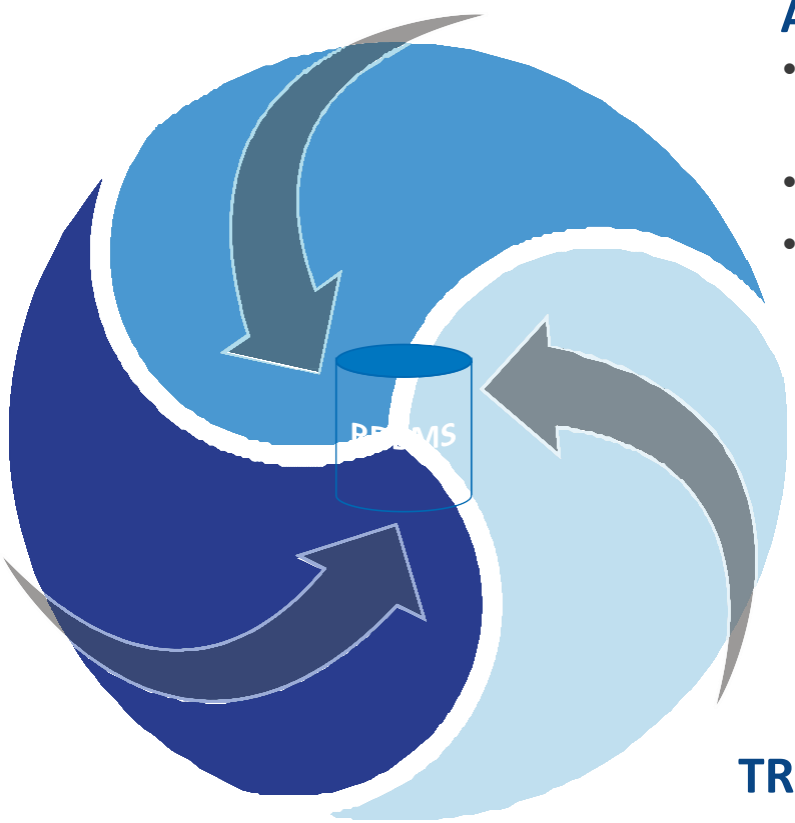
Enter Big Data

- Gartner uses the 3Vs to define
- Volume - Big/difficult/extreme volume is relative
- Variety
  - Changing or evolving data
  - Uncontrolled formats
  - Does not easily adhere to a single schema
  - Unknown at design time
- Velocity
  - High or volatile inbound data
  - High query and read operations
  - Low latency

RDBMS challenges

VOLUME & NEW ARCHITECTURES

- Systems scaling horizontally, not vertically
- Commodity servers
- Cloud Computing



DATA VARIETY & VOLATILITY

•Extremely difficult to find a single fixed schema

TRANSACTIONAL MODEL

- N x Inserts or updates
- Distributed transactions

Enter NoSQL

- Non-relational been hanging around (MUMPS?)
- Modern NoSQL theory and offerings started in early 2000s
- Modern usage of term introduced in 2009
- NoSQL = Not Only SQL
- A collection of very different products
- Alternatives to relational databases when they are a bad fit
  Motives
- Horizontally scalable (commodity server/cloud computing)
- Flexibility

What is NoSQL

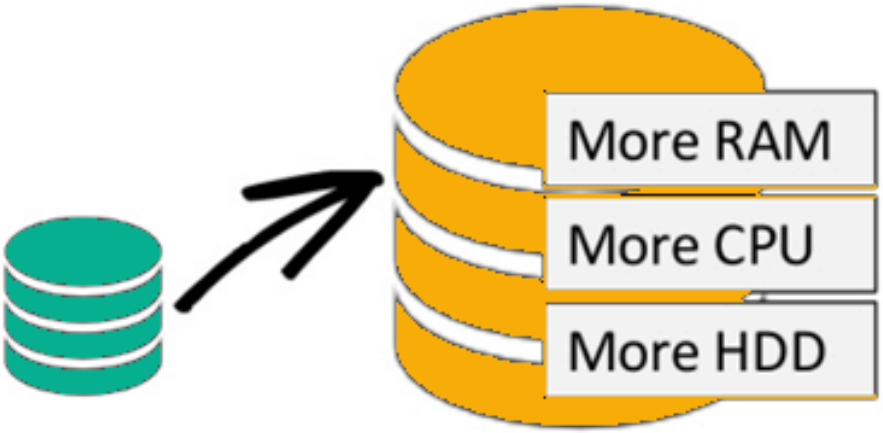
- NoSQL Database is a non-relational Data Management System, that does not require a fixed schema.
- It avoids joins, and is easy to scale.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps.
- For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.
- NoSQL database stands for "Not Only SQL" or "Not SQL."

Why NoSQL

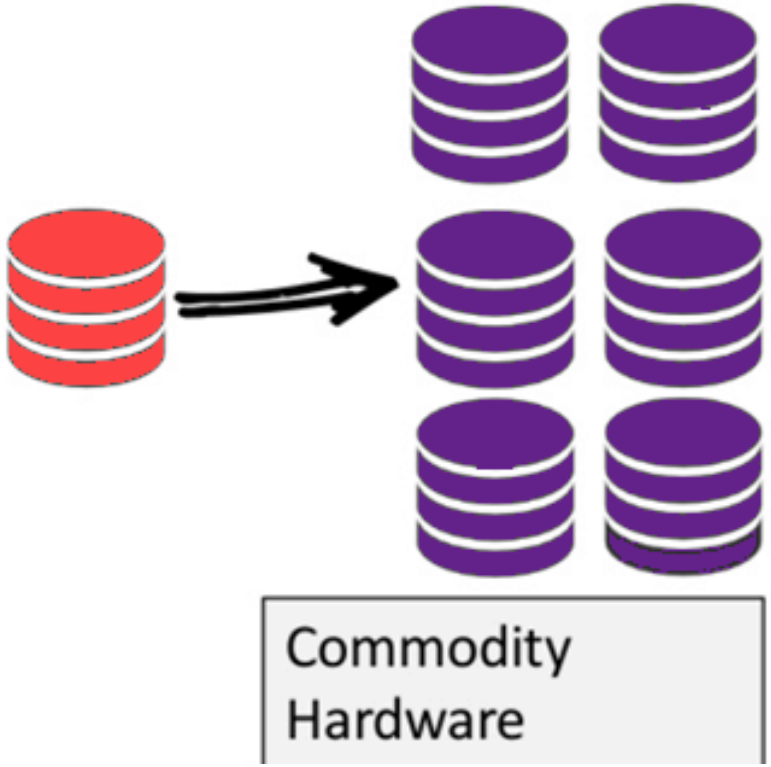
- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.
- The system response time becomes slow when you use RDBMS for massive volumes of data.
- To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

Why NoSQL

Scale-Up (*vertical* scaling):

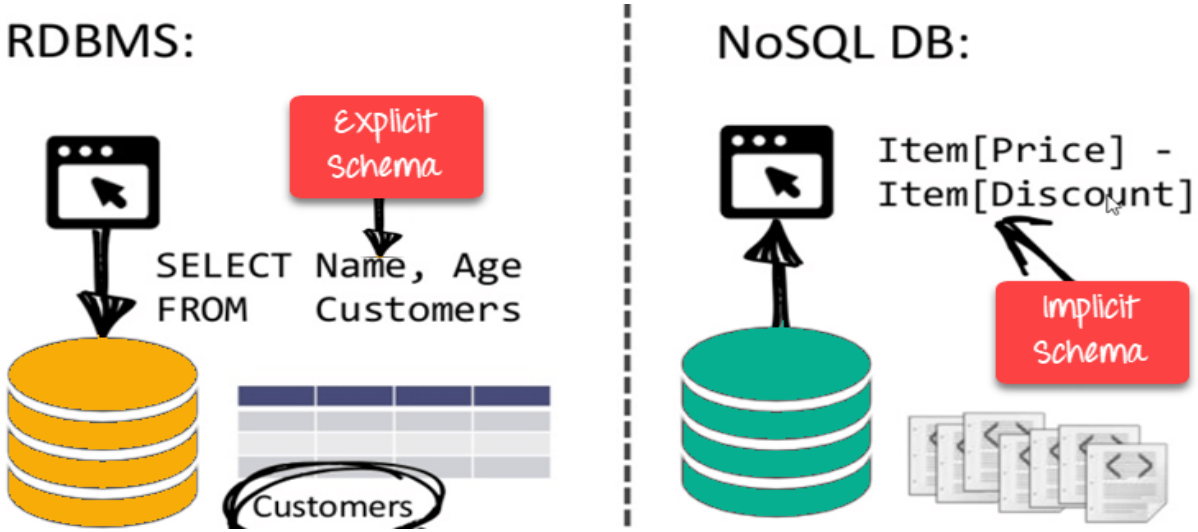


Scale-Out (*horizontal* scaling):



Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain



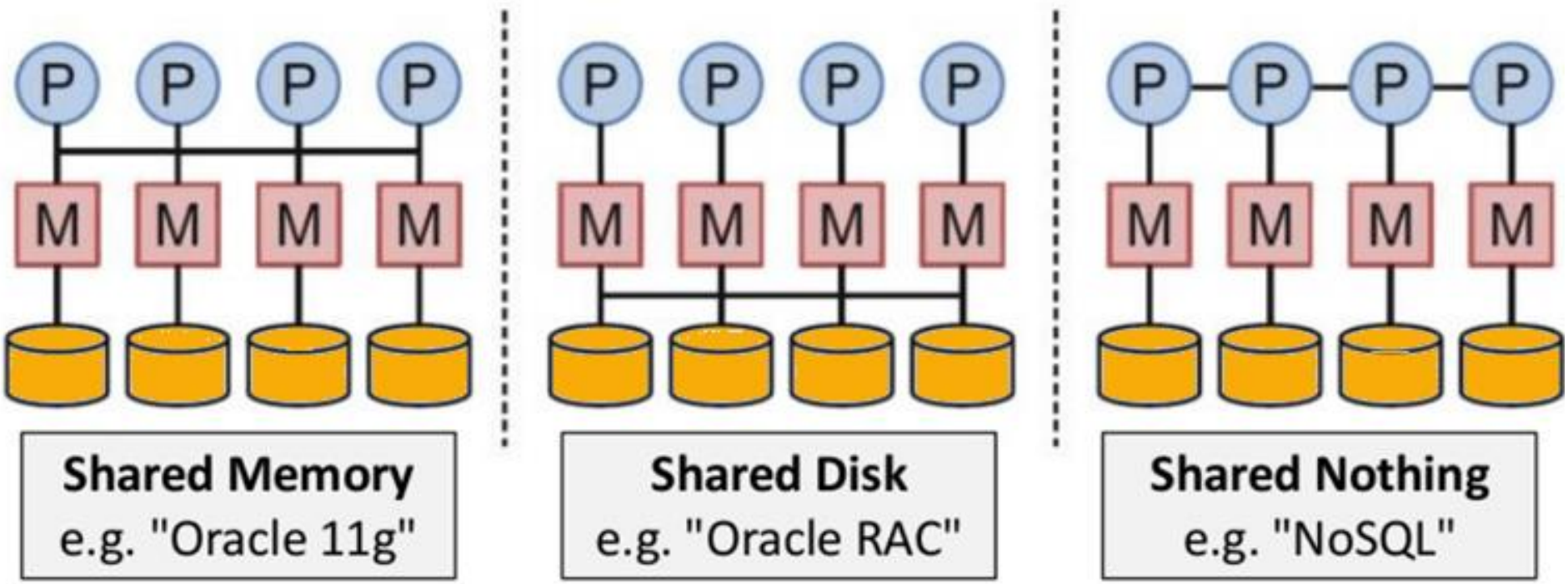
Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based NoSQL query language
- Web-enabled databases running as internet-facing services

Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput
- Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
- Only providing eventual consistency
- Shared Nothing Architecture. This enables less coordination and higher distribution.

Distributed



Types of NoSQL Databases

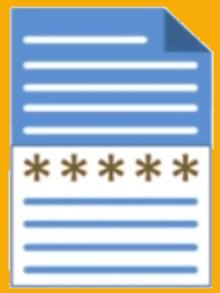
- NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented.
- Every category has its unique attributes and limitations.
- None of the above-specified database is better to solve all the problems.
- Users should select the database based on their product needs.
- Types of NoSQL Databases:
  - Key-value Pair Based
  - Column-oriented Graph
  - Graphs based
  - Document-oriented

Types of NoSQL Databases

Key Value

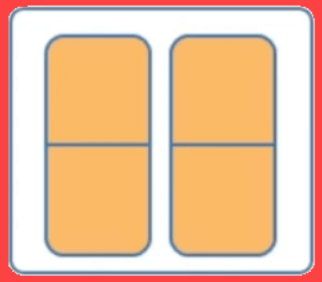
Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based



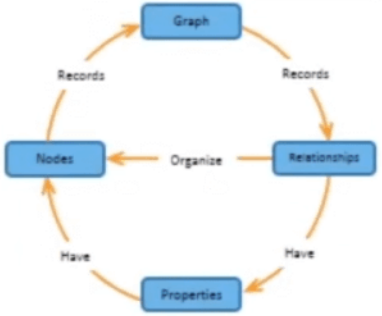
Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based



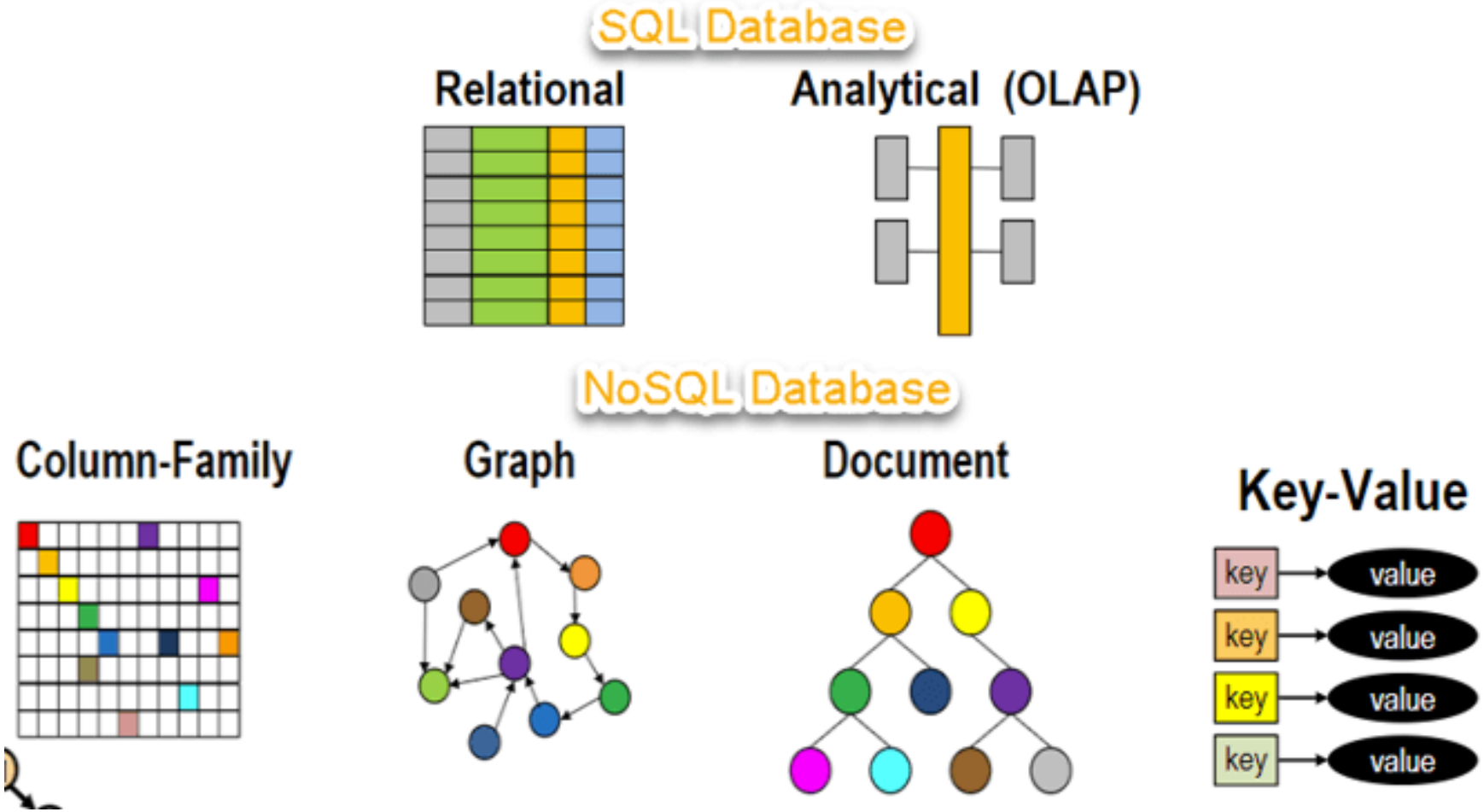
Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based



Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

What is NoSQL



Key Value Pair Based

- Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
- Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
- For example, a key-value pair may contain a key like "Website" associated with a value like “virtusa".

Key Value Pair Based

- It is one of the most basic NoSQL database example.
- This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc.
- Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
- Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.

Column-based

- Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column-based

- They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.
- Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.

Document-Oriented

- Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document.
- The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

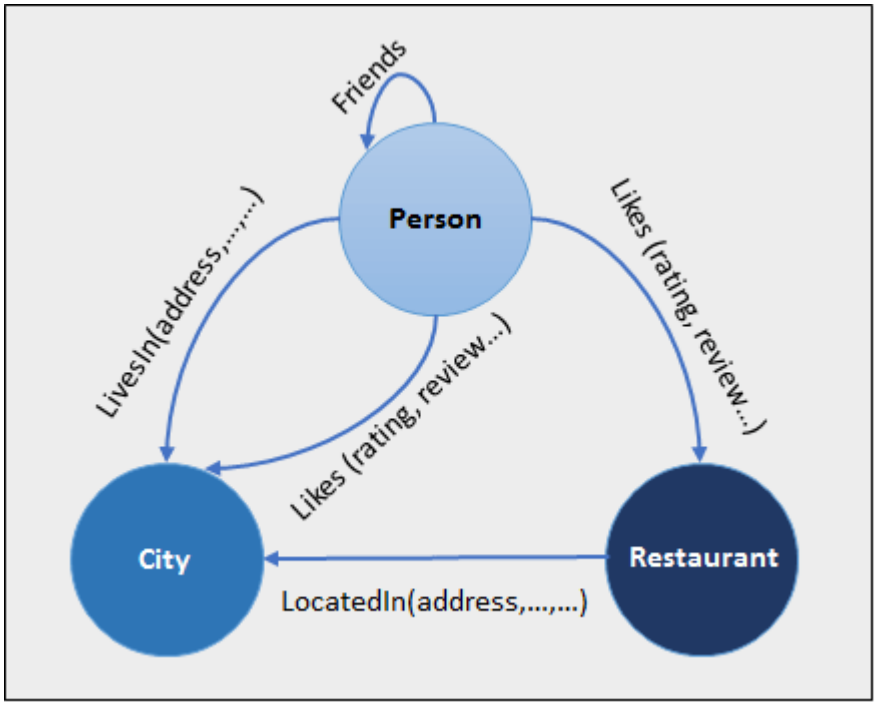
```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document-Oriented

- In the above diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON.
- Now for the relational database, you have to know what columns you have and so on.
- However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.
- The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications.
- It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.
- Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

Graph-Based

- A graph type database stores entities as well the relations amongst those entities.
- The entity is stored as a node with the relationship as edges.
- An edge gives a relationship between nodes. Every node and edge has a unique identifier.



Graph-Based

- Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature.
- Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.
- Graph base database mostly used for social networks, logistics, spatial data.
- Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

SQL vs NoSQL

RDBMS	NoSQL
•Data is stored in a relational model, with rows and columns. •A row contains information about an item while columns contain specific information, such as ‘Model’, ‘Date of Manufacture’, ‘Color’. •Follows fixed schema. Meaning, the columns are defined and locked before data entry. In addition, each row contains data for each column. •Supports vertical scaling. Scaling an RDBMS across multiple servers is a challenging and time-consuming process. •Atomicity, Consistency, Isolation & Durability(ACID) Compliant	• Data is stored in a host of different databases, with different data storage models. •Follows dynamic schemas. Meaning, you can add columns anytime. •Supports horizontal scaling. You can scale across multiple servers. Multiple servers are cheap commodity hardware or cloud instances, which make scaling cost-effective compared to vertical scaling. •Not ACID Compliant.

What is the CAP Theorem?

- CAP theorem is also called brewer's theorem. It states that is impossible for a distributed data store to offer more than two out of three guarantees
  1. Consistency
  2. Availability
  3. Partition Tolerance

What is the CAP Theorem?

- Consistency:
  - The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.
- Availability:
  - The database should always be available and responsive. It should not have any downtime.

What is the CAP Theorem?

- Partition Tolerance
  - Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable.
  - For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

What is the CAP Theorem?

- Partition Tolerance
  - Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable.
  - For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

Visual Guide to NoSQL Systems

Availability: Each client can always read and write.

A

Available, Partition-Tolerant (AP) Systems achieve "eventual consistency" through replication and verification

Data Models

- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Pick Two

Consistent, Available (CA) Systems have trouble with partitions and typically deal with it with replication

Consistent, Partition-Tolerant (CP) Systems have trouble with availability while keeping data consistent across partitioned nodes

C

P

Consistency: All clients always have the same view of the data.

CP

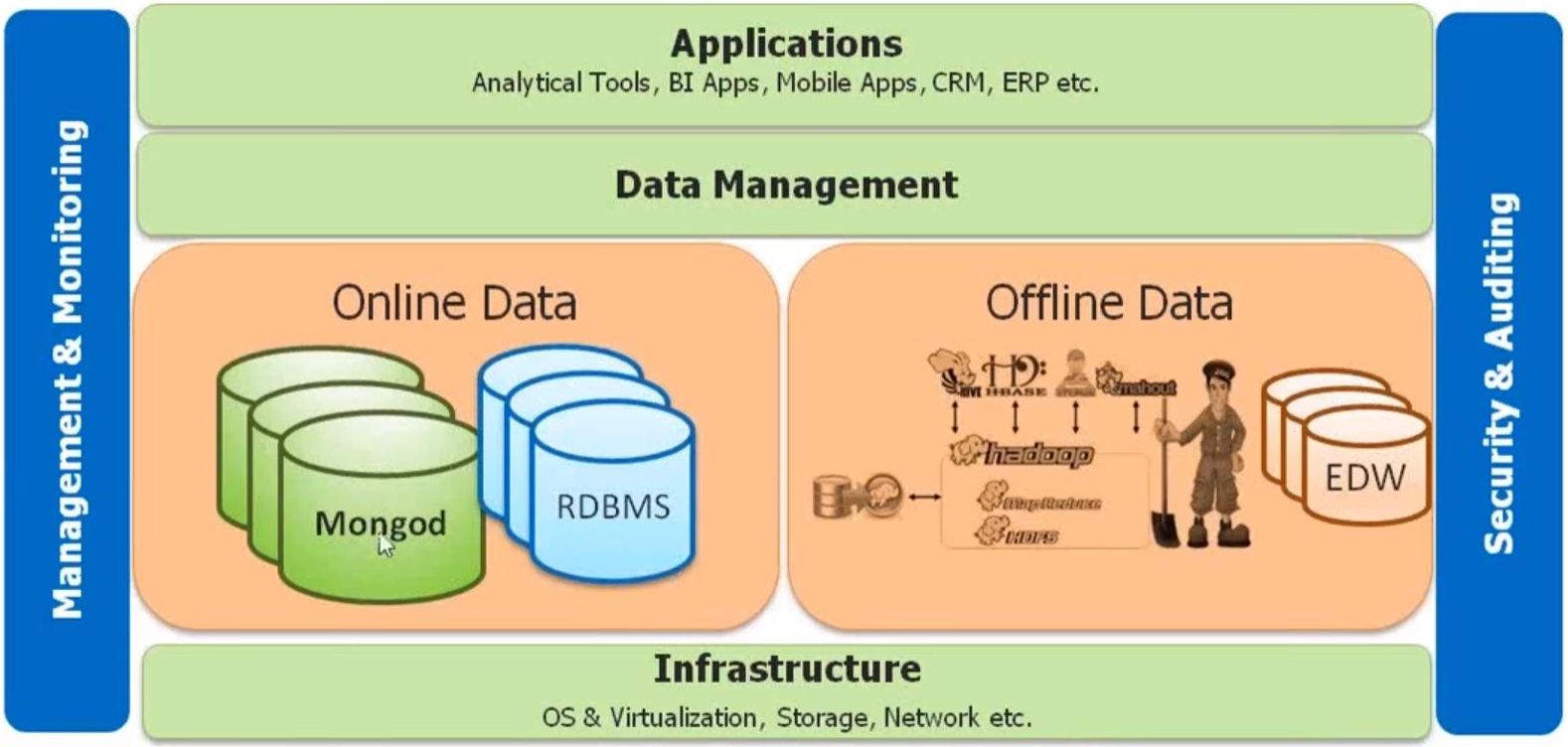
BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

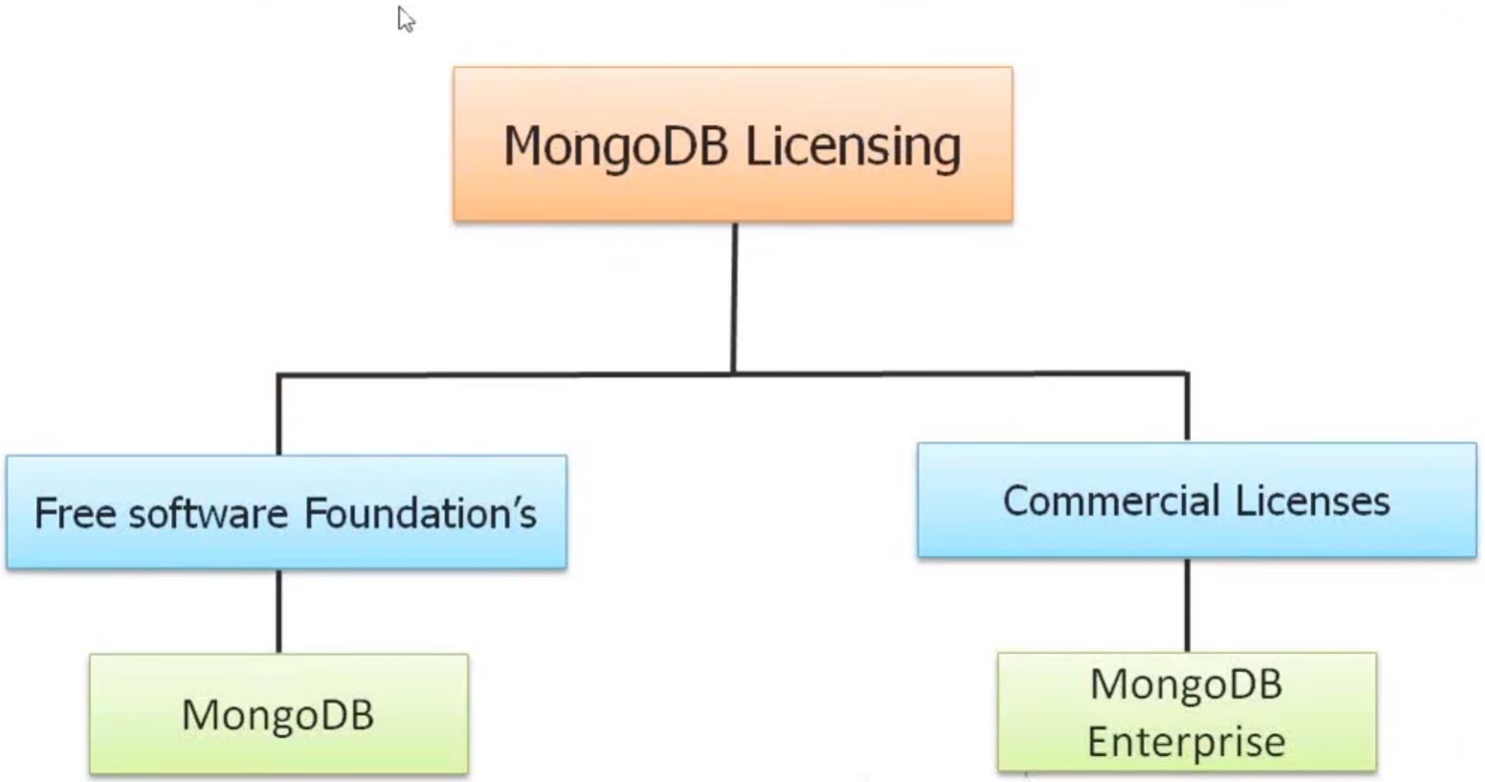
Berkeley DB
MemcacheDB
Redis

Partition Tolerance: The system works well despite physical network partitions.

MongoDB High Level Architecture



MongoDB Licensing



MongoDB Enterprise

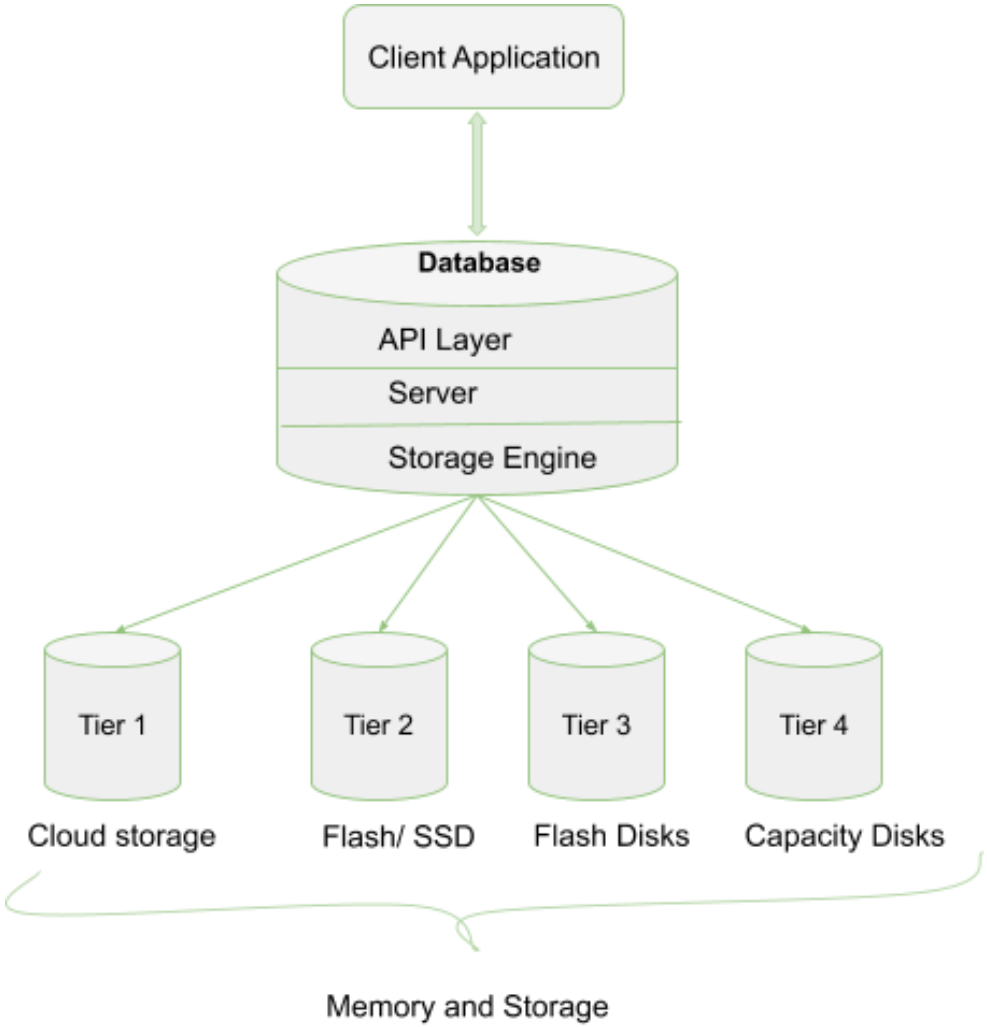
- ✓ MongoDB Enterprise is the commercial edition of MongoDB that provides enterprise-grade capabilities.
- ✓ MongoDB Enterprise includes advanced security features, management tools, software integrations and certifications.
- ✓ These value-added capabilities are not included in the open-source edition of MongoDB.

MongoDB Enterprise

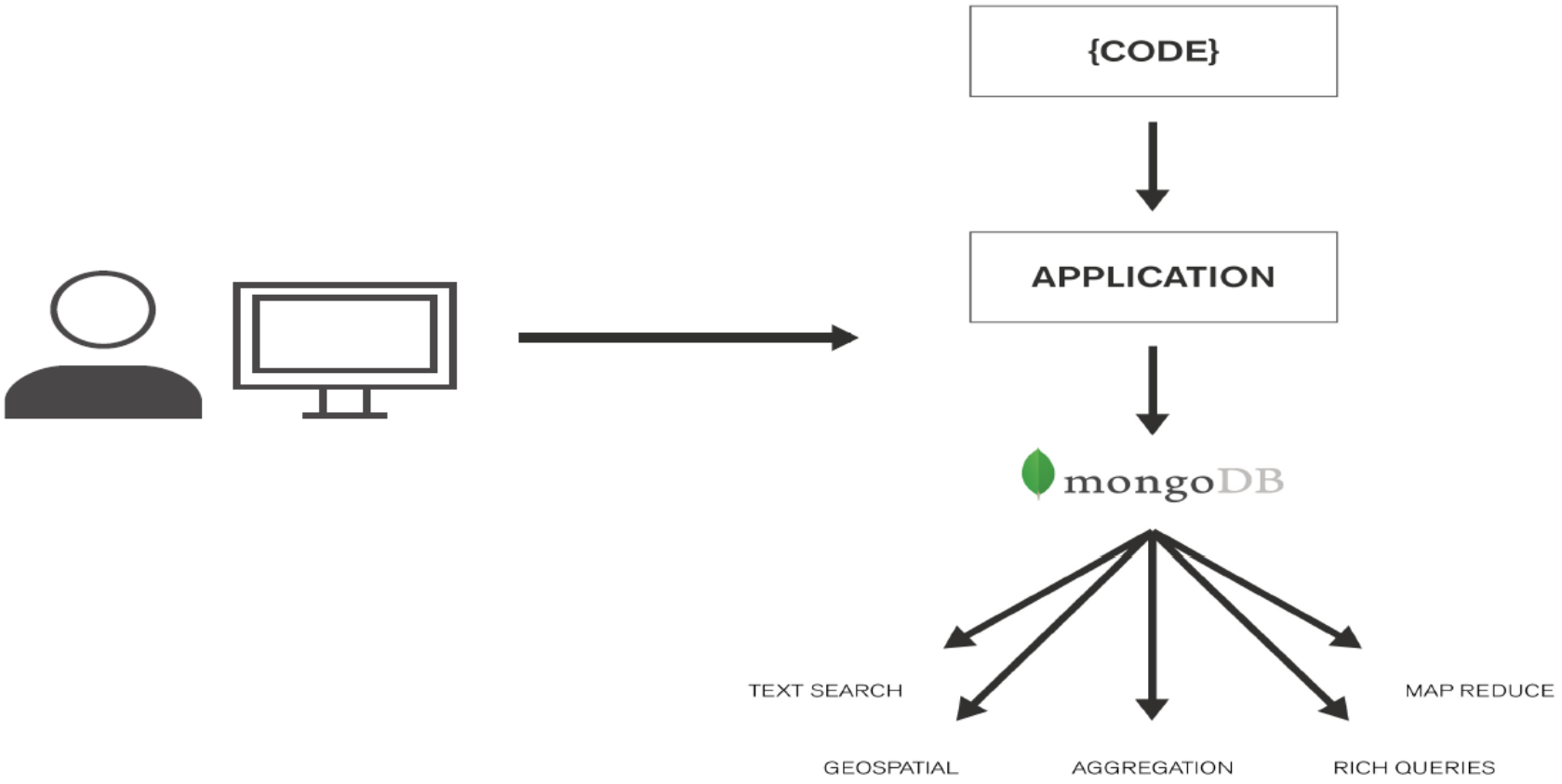
MongoDB Enterprise

Features	MongoDB	MongoDB Enterprise
JSON Data Model with Dynamic Schemas	•	•
Auto-Sharding for Horizontal Scalability	•	•
Built-In Replication and High Availability	•	•
Full, Flexible Index Support	•	•
Rich Document Queries	•	•
Fast In-Place Updates	•	•
Aggregation Framework and MapReduce	•	•
Large Media Storage with GridFS	•	•
Text Search	•	•
Cloud, On-Premise and Hybrid Deployments	•	•
Role-Based Privileges	•	•
Advanced Security with Kerberos		•
On-Prem Management		•
SNMP Support		•
OS Certifications		•
Private On-Demand Training		•

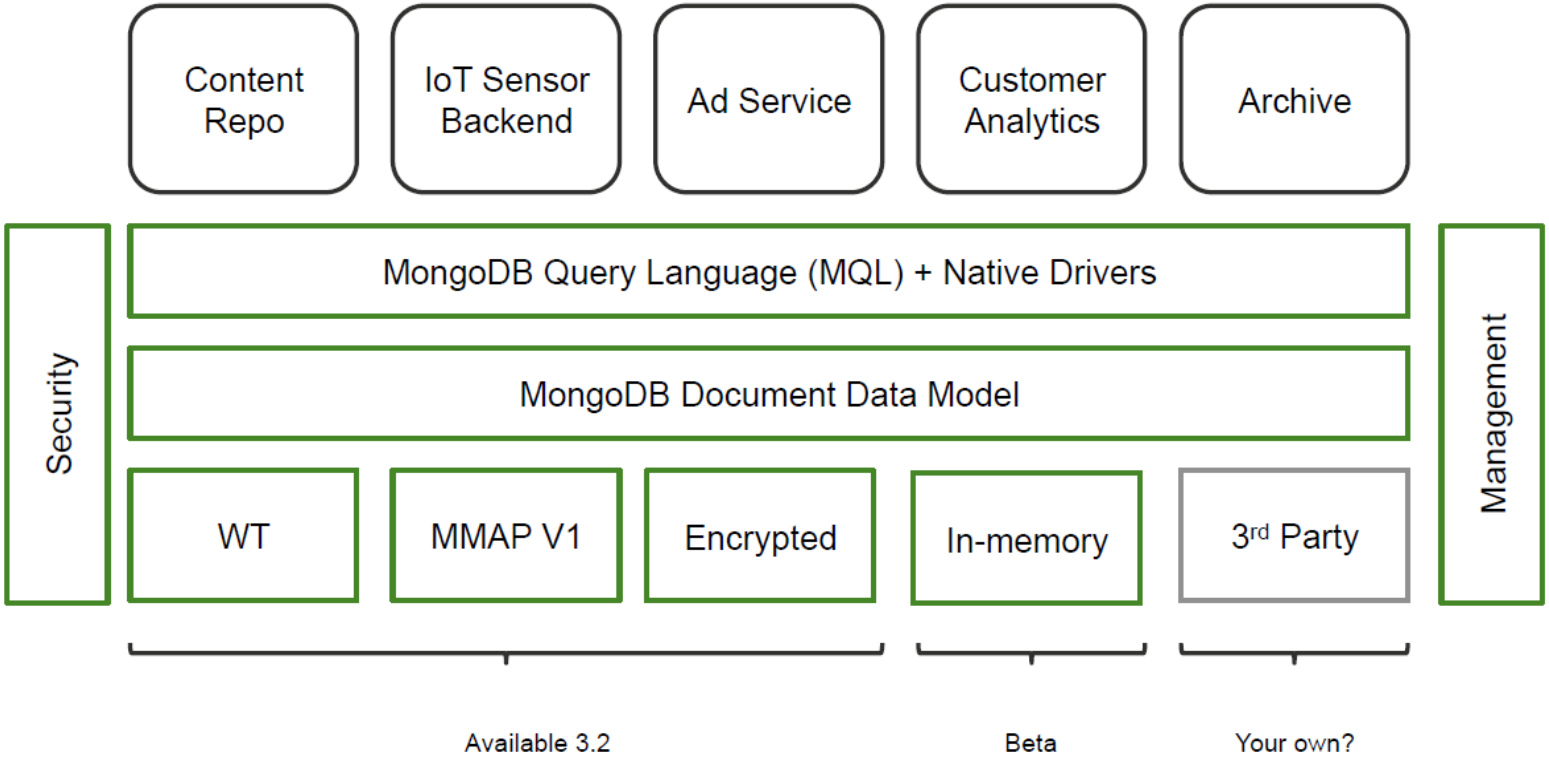
Storage Engine



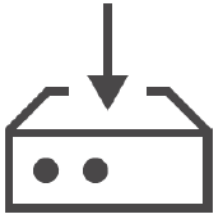
MongoDB Fully Featured



MongoDB Architecture

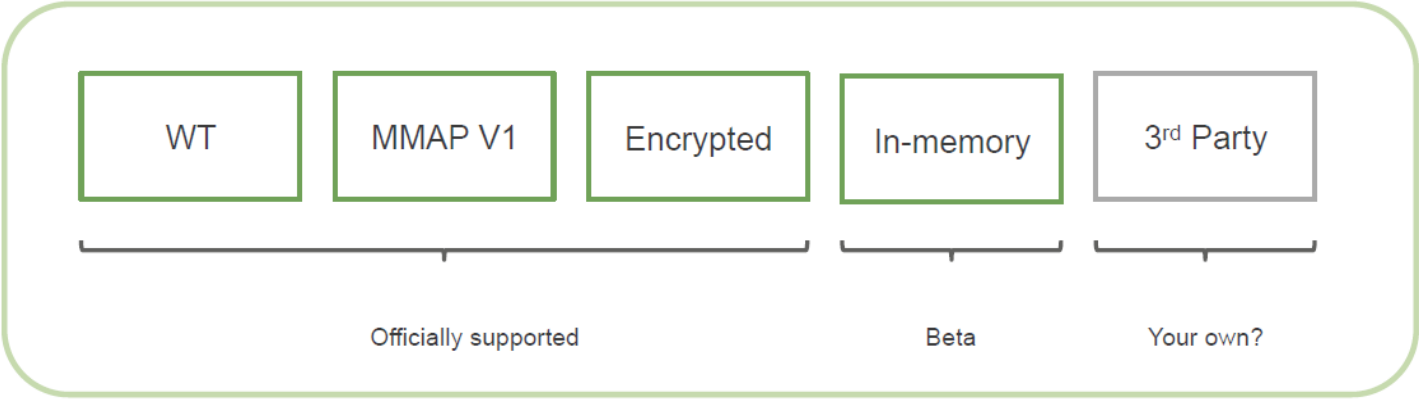


MongoDB Architecture



Storage Layer

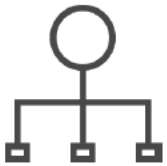
- Different workloads require different storage strategies
- Exposed by a Storage Engine API
- Provides more flexibility to your deployments



MongoDB Architecture

BSON

Collections



Indexes

Databases

MongoDB Document Data Model

Data Model

MongoDB Architecture


Java


Ruby

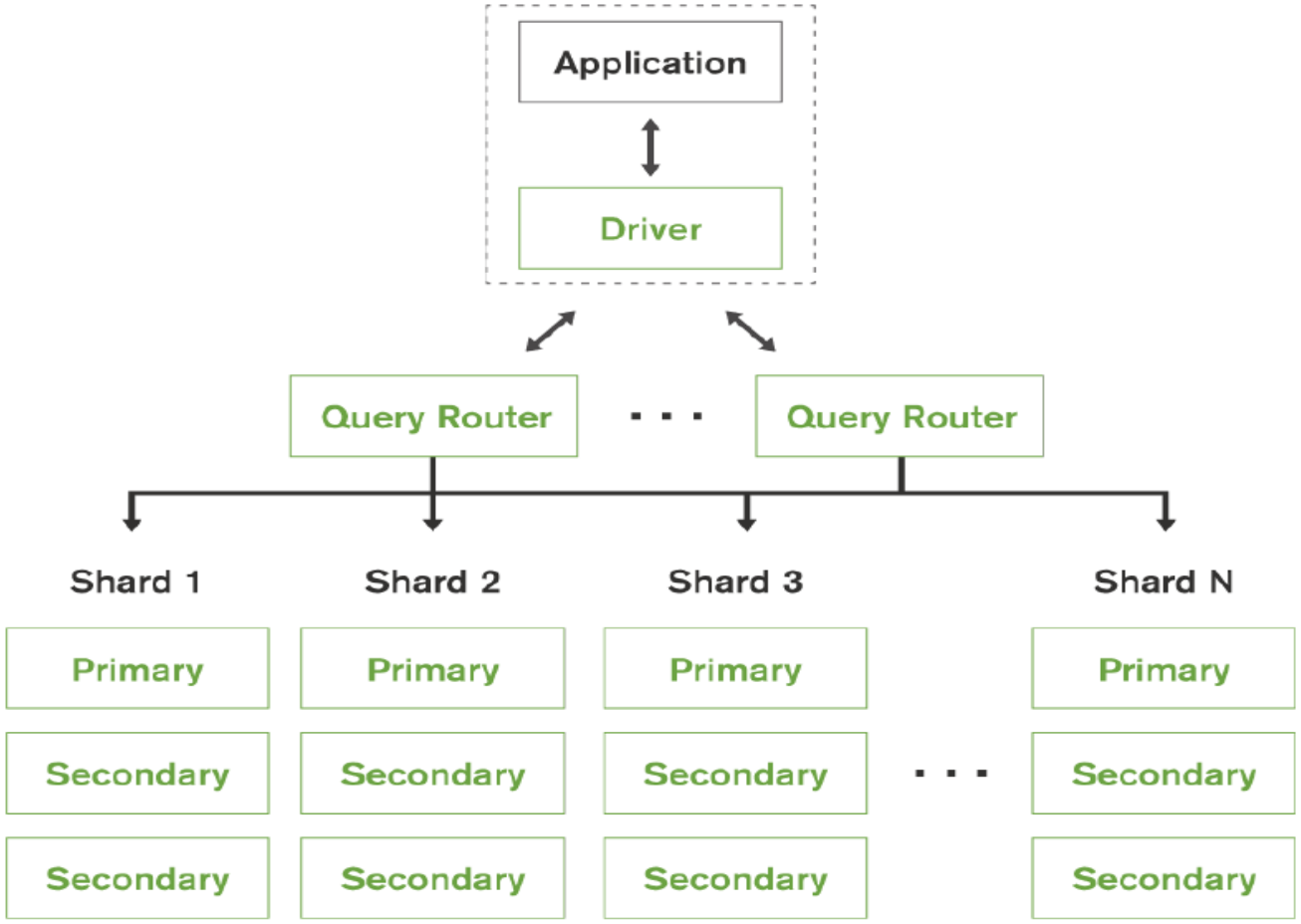

Python


Perl

MongoDB Query Language (MQL) + Native Drivers

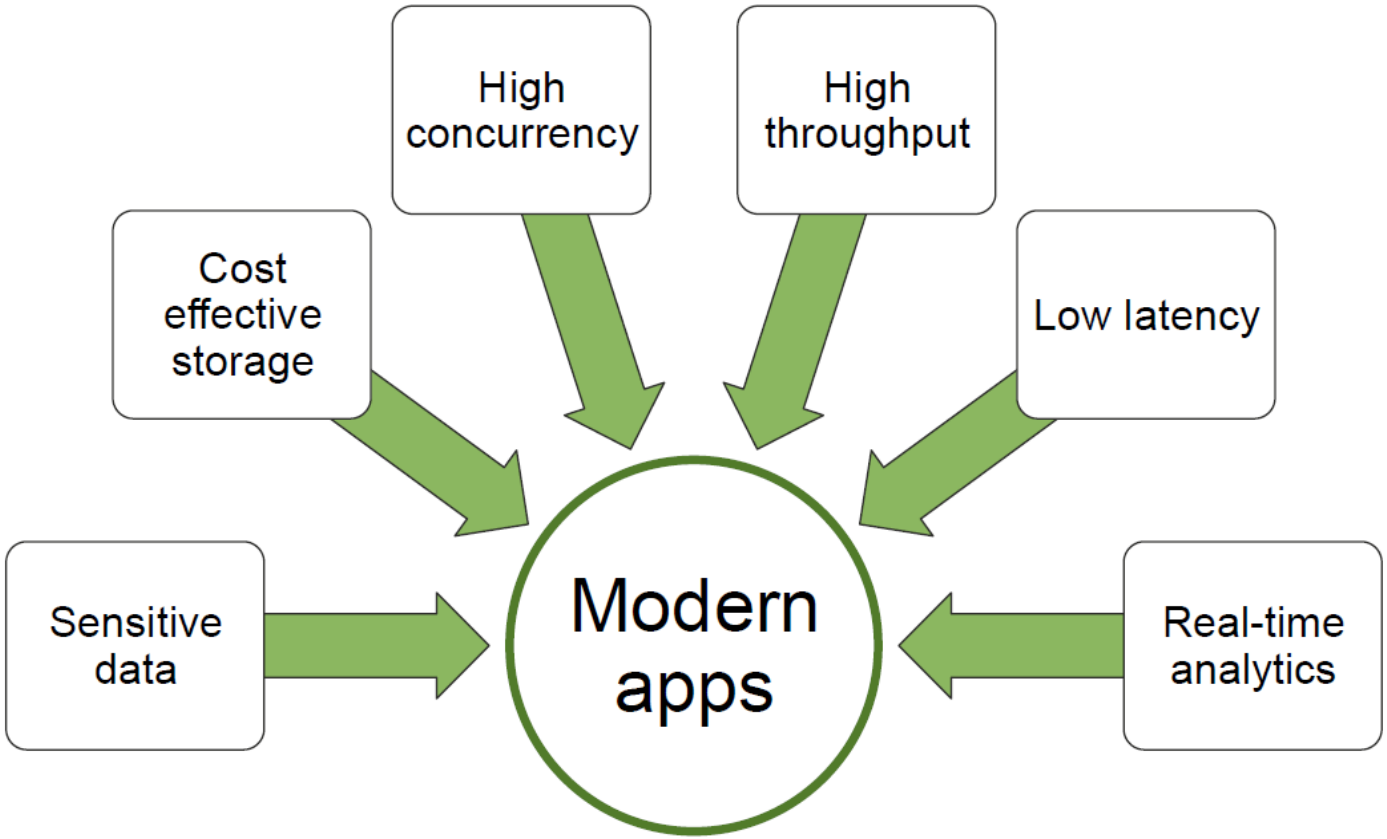

```
db.coll.insert({'name': 'Norberto'})
db.coll.update({'name': 'Norberto'}, {'$set':{'role': 'Developer Advocate'})

db.coll.find({'role': /^Developer/})
db.coll.find({}).skip(10).limit(20)

db.coll.aggregate({'$group': { '_id': '$role', "howmany": {"$sum":1} }})

db.serverStatus()
```

Distributed Database

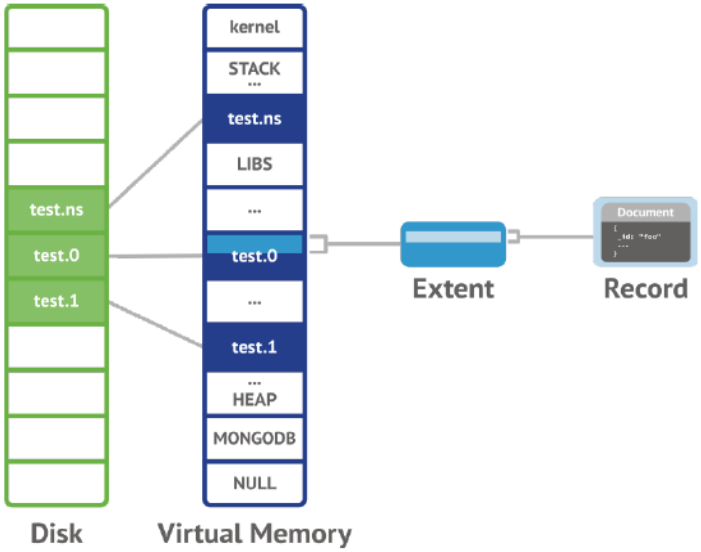


Storage Engines

Varying Access & Storage Requirements



Good Old MMAPv1



MMAPv1 is our traditional storage engine that allows a great deal of performance for read heavy applications

- Improved concurrency control
- Great performance on read-heavy workloads
- Data & Indexes memory mapped into virtual address space
- Data access is paged into RAM
- OS evicts using LRU
- More frequently used pages stay in RAM

Storage Engine API

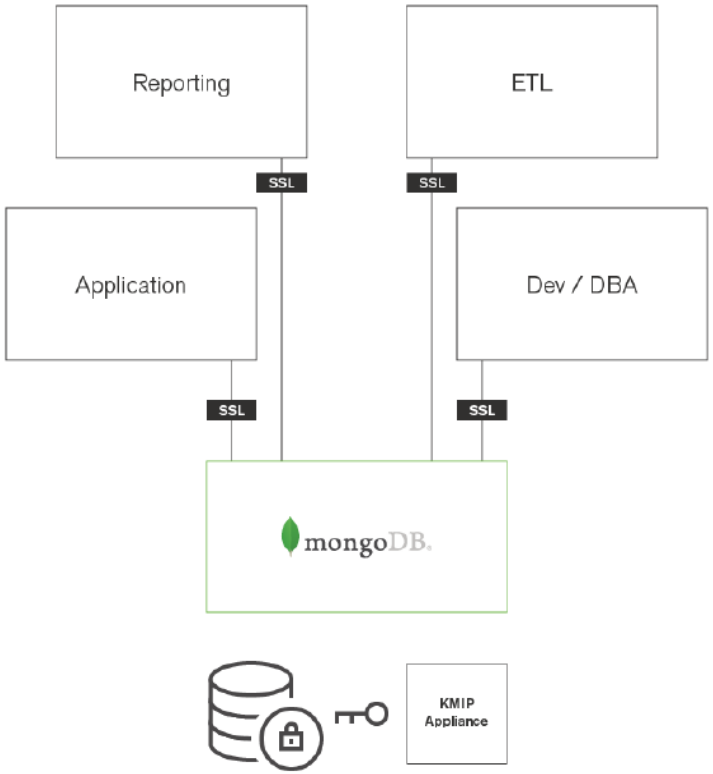
- Allows to "plug-in" different storage engines
  - Different use cases require different performance characteristics
  - mmapv1 is not ideal for all workloads
  - More flexibility
    - Can mix storage engines on same replica set/sharded cluster
- Opportunity to integrate further (HDFS, native encrypted, hardware optimized …)

WiredTiger is the New Default

WiredTiger – widely deployed with 3.0 – is now the default storage engine for MongoDB.

- Best general purpose storage engine
- 7-10x better write throughput
- Up to 80% compression

Encrypted Storage Engine



Encrypted storage engine for end-to-end encryption of sensitive data in regulated industries

- Reduces the management and performance overhead of external encryption mechanisms
- AES-256 Encryption, FIPS 140-2 option available
- Key management: Local key management via keyfile or integration with 3rd party key management appliance via KMIP
- Based on WiredTiger storage engine
- Requires MongoDB Enterprise Advanced

In-Memory Storage Engine (Beta)

Handle ultra-high throughput with low latency and high availability

- Delivers the extreme throughput and predictable latency required by the most demanding apps in Adtech, finance, and more.
- Achieve data durability with replica set members running disk-backed storage engine
- Available for beta testing and is expected for GA in early 2016

Data Modeling

RDBMS	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Join	Embedding & Linking

Document Data Model

Relational

PERSON

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

NO RELATION

CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB

```
{
  first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location:
[45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Documents are Rich Data Structures

```
{
  first_name: 'Paul',
  surname: 'Miller',
  cell: 447557505611,
  city: 'London',
  location: [45.123,47.232],
  Profession: ['banking', 'finance',
'trader'],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Fields

String

Number

Geo-Coordinates

Typed field values

Fields can contain arrays

Fields can contain an array of sub-documents

mongoDB

Document Model Benefits

```
{
    _id : ObjectId("4c4ba5e5e8aabf3"),
    employee_name: "Dunham, Justin",
    department : "Marketing",
    title : "Product Manager, Web",
    report_up: "Neray, Graham",
    pay_band: "C",
    benefits : [
            {  type :  "Health",
               plan : "PPO Plus" },
            {  type :   "Dental",
               plan : "Standard" }
                  ]
}
```

Agility and flexibility

Data model supports business change

Rapidly iterate to meet new requirements

Intuitive, natural data representation

Eliminates ORM layer

Developers are more productive

Reduces the need for joins, disk seeks

Programming is more simple

Performance delivered at scale

Dynamic Schemas

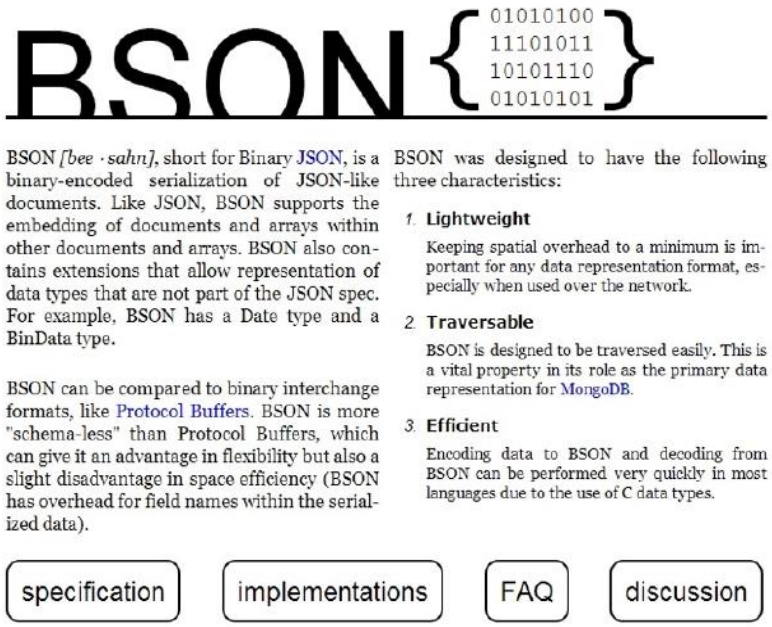
```
{
    policyNum: 123,
    type: auto,
    customerId: abc,
    payment: 899,

    deductible: 500,
    make: Taurus,
    model: Ford,
    VIN: 123ABC456,
}
```

```
{
    policyNum: 456,
    type: life,
    customerId: efg,
    payment: 240,

    policyValue: 125000,
    start: jan, 1995
    end: jan, 2015
    }
```

```
{
  policyNum: 789,
  type: home,
  customerId: hij,
   payment: 650,

  deductible: 1000,
  floodCoverage: No,
  street: “10 Maple Lane”,
   city: “Springfield”,
   state: “Maryland”
}
```

BSON

- Binary JSON serialization format
- JSON with types
- The language MongoDB speaks



BSON { 01010100 11101011 10101110 01010101 }

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON can be compared to binary interchange formats, like Protocol Buffers. BSON is more "schema-less" than Protocol Buffers, which can give it an advantage in flexibility but also a slight disadvantage in space efficiency (BSON has overhead for field names within the serialized data).

BSON was designed to have the following three characteristics:

1. **Lightweight**

   Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.

2. **Traversable**

   BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.

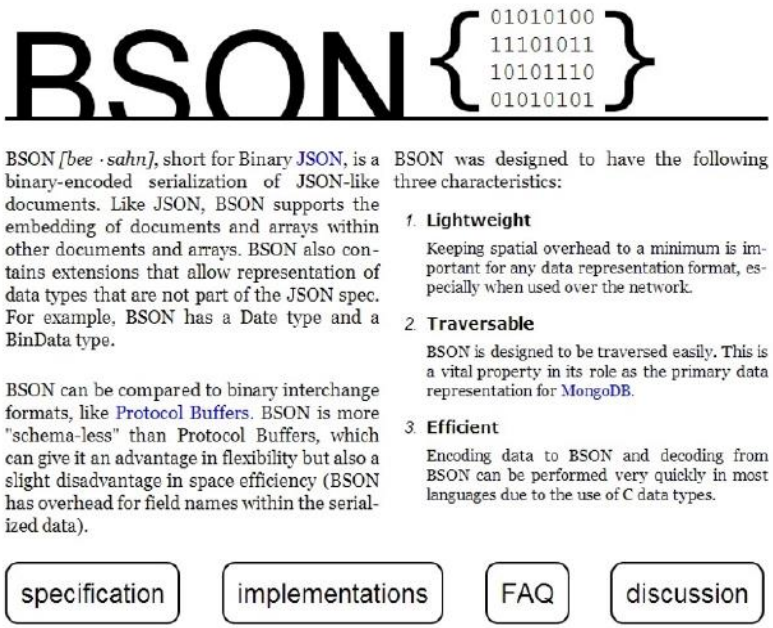
3. **Efficient**

   Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

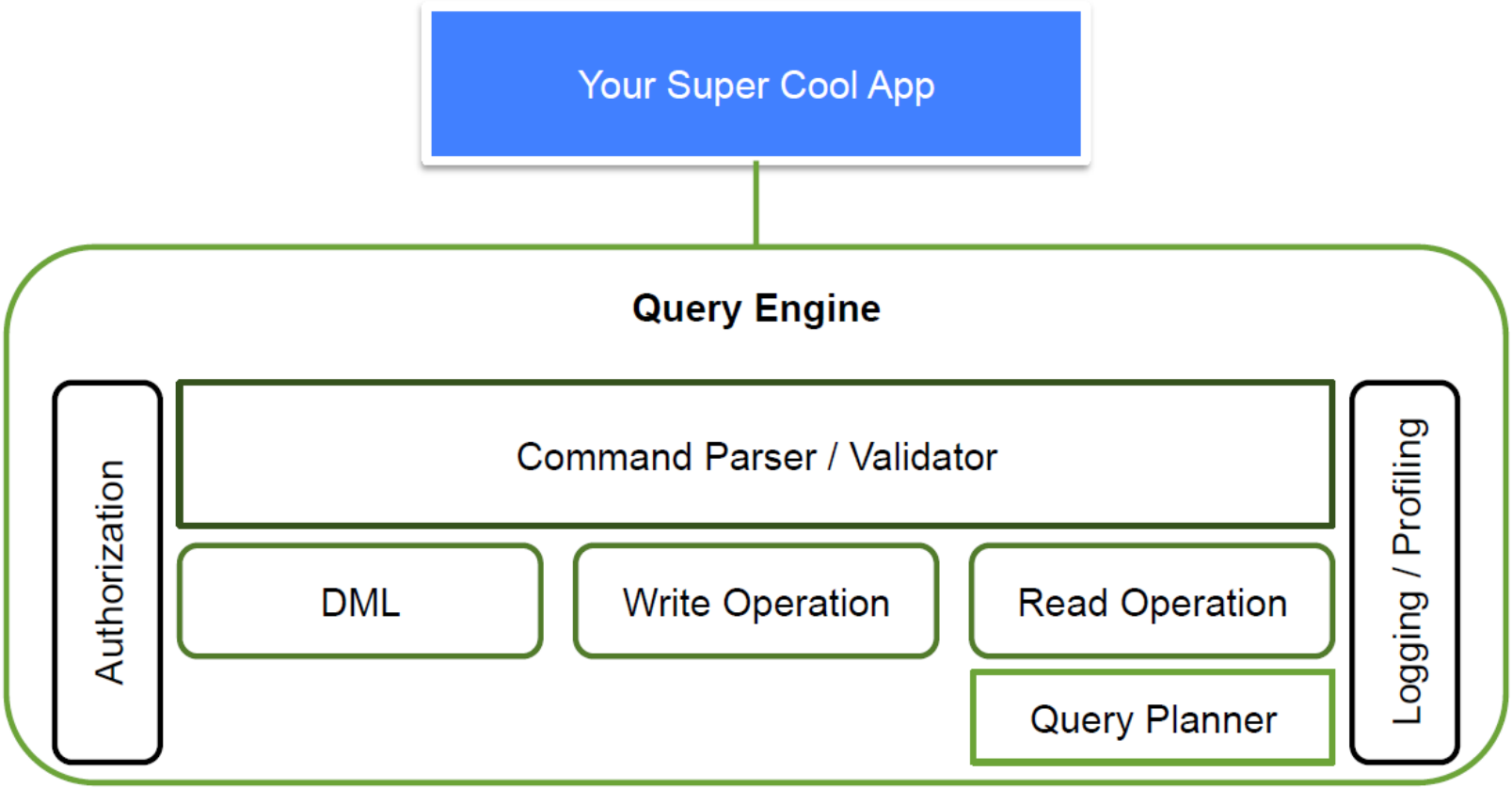
specification | implementations | FAQ | discussion

BSON Data Types

- String
- Document
- Array
- Binary
- ObjectId
- Boolean
- Date
- Timestamp
- Double (64 bit)
- Long (64 bit)
- Integer (32 bit)
- Min Key
- Max Key
- Javascript
- Javascript with Scope
- Null value



Query Engine



Column Based Database Cassandra

- Cassandra was initially developed at Facebook by two Indians Avinash Lakshman (one of the authors of Amazon's Dynamo) and Prashant Malik.
- It was developed to power the Facebook inbox search feature.
- The following points specify the most important happenings in Cassandra history:
  - It was developed for Facebook inbox search feature.
  - It was open sourced by Facebook in July 2008.
  - It was accepted by Apache Incubator in March 2009.
  - Cassandra is a top level project of Apache since February 2010.
  - The latest version of Apache Cassandra is 3.2.1.

Column Based Database Cassandra

Version	Original Release Date	Latest Version	Release Date	Status
0.6	2010-04-12	0.6.13	2011-04-18	No longer supported.
0.7	2011-01-10	0.7.10	2011-10-31	No longer supported.
0.8	2011-06-03	0.8.10	2012-02-13	No longer supported.
1.0	2011-10-18	1.0.12	2012-10-04	No longer supported.
1.1	2012-04-24	1.1.12	2013-05-27	No longer supported.
1.2	2013-01-02	1.2.19	2014-09-18	No longer supported.
2.0	2013-09-03	2.0.17	2015-09-21	No longer supported.
2.1	2014-09-16	2.1.17	2017-02-21	Still supported.
2.2	2015-07-20	2.2.9	2017-02-21	Still supported.
3.0	2015-11-09	3.0.11	2017-02-21	Still supported.
3.10	2017-02-03	3.10	2017-02-03	Latest release.
3.11	2017-02-22	3.11	2017-02-11	Github 3.11 branch.

Cassandra Architecture

- Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure.
- It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.
- In Cassandra, each node is independent and at the same time interconnected to other nodes.
- All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

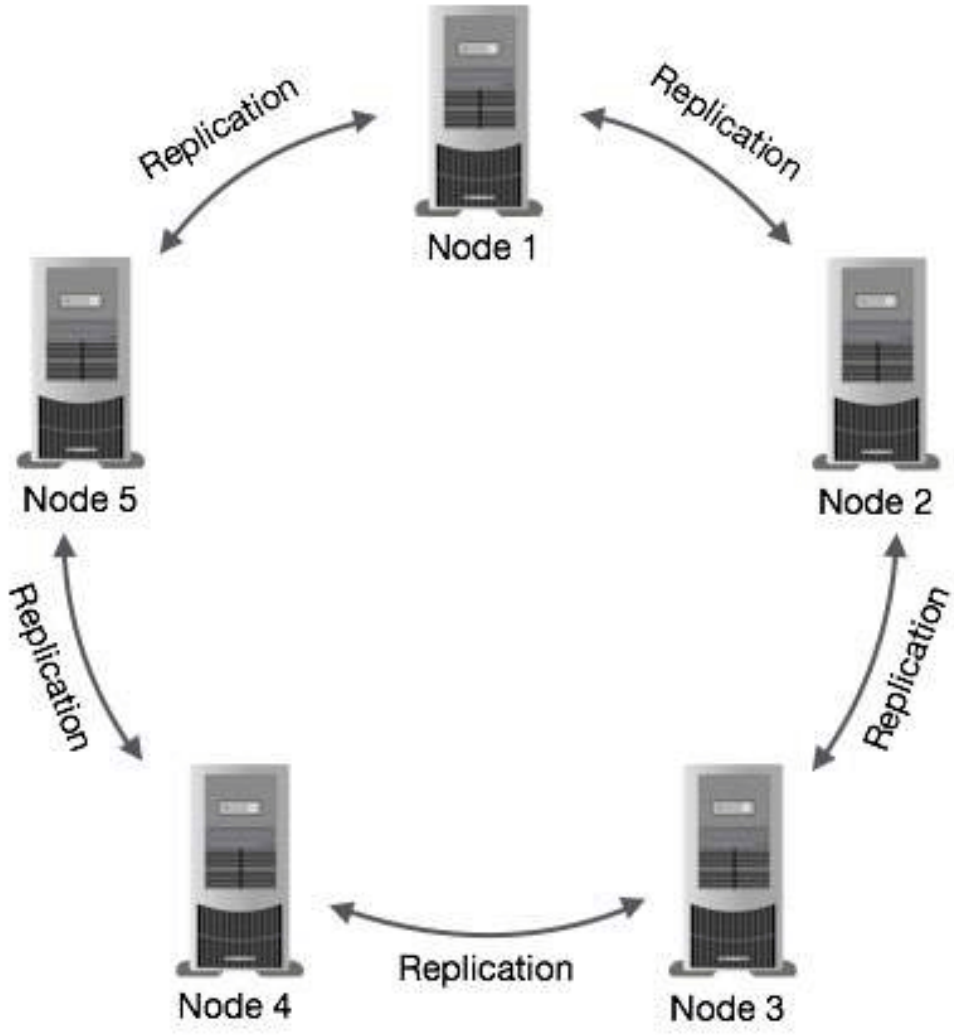
Cassandra Architecture

- Cassandra was designed to handle big data workloads across multiple nodes without a single point of failure.
- It has a peer-to-peer distributed system across its nodes, and data is distributed among all the nodes in a cluster.
- In Cassandra, each node is independent and at the same time interconnected to other nodes.
- All the nodes in a cluster play the same role.
- Every node in a cluster can accept read and write requests, regardless of where the data is actually located in the cluster.
- In the case of failure of one node, Read/Write requests can be served from other nodes in the network.

Data Replication in Cassandra

- In Cassandra, nodes in a cluster act as replicas for a given piece of data.
- If some of the nodes are responded with an out-of-date value, Cassandra will return the most recent value to the client.
- After returning the most recent value, Cassandra performs a read repair in the background to update the stale values.

Data Replication in Cassandra



Components of Cassandra

- The main components of Cassandra are:
  - Node: A Cassandra node is a place where data is stored.
  - Data center: Data center is a collection of related nodes.
  - Cluster: A cluster is a component which contains one or more data centers.
  - Commit log: In Cassandra, the commit log is a crash-recovery mechanism. Every write operation is written to the commit log.
  - Mem-table: A mem-table is a memory-resident data structure. After commit log, the data will be written to the mem-table. Sometimes, for a single-column family, there will be multiple mem-tables.

Components of Cassandra

- SSTable: It is a disk file to which the data is flushed from the mem-table when its contents reach a threshold value.
- Bloom filter: These are nothing but quick, nondeterministic, algorithms for testing whether an element is a member of a set. It is a special kind of cache. Bloom filters are accessed after every query.

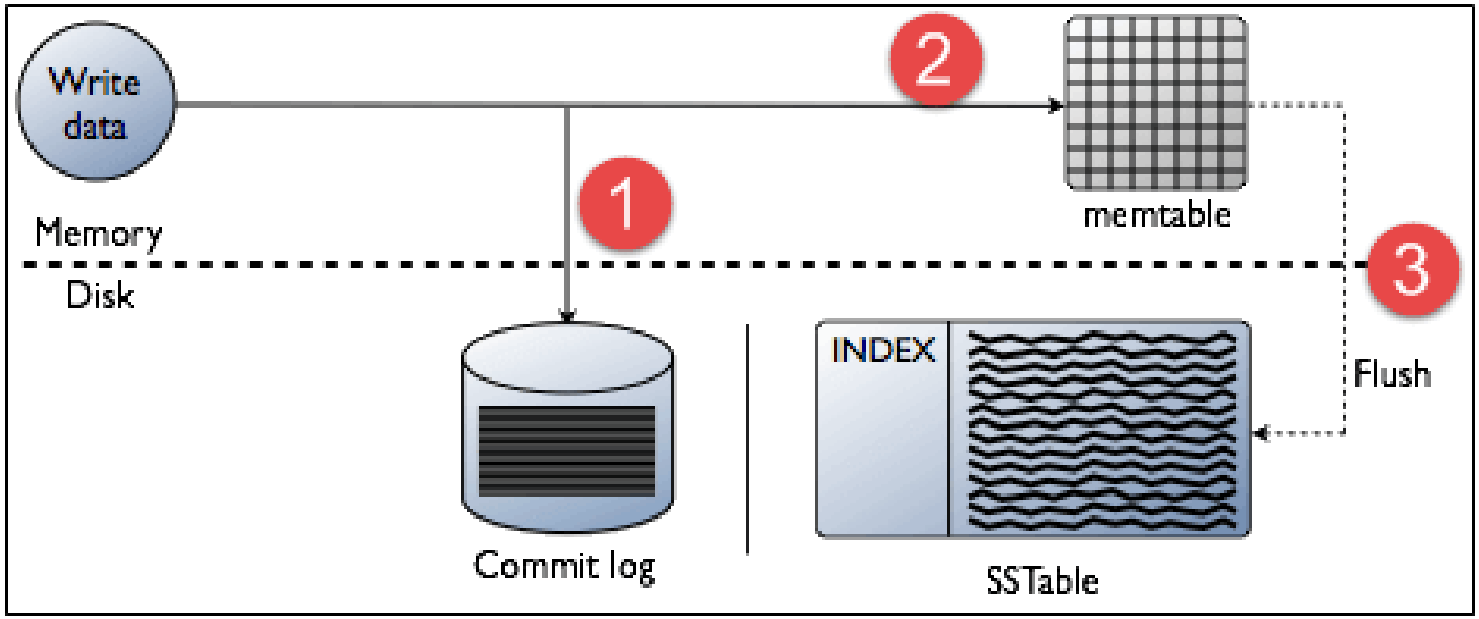
Cassandra Query Language

- Cassandra Query Language (CQL) is used to access Cassandra through its nodes.
- CQL treats the database (Keyspace) as a container of tables.
- Programmers use cqlsh: a prompt to work with CQL or separate application language drivers.
- The client can approach any of the nodes for their read-write operations.
- That node (coordinator) plays a proxy between the client and the nodes holding the data.

Write Operations

- Every write activity of nodes is captured by the commit logs written in the nodes.
- Later the data will be captured and stored in the mem-table.
- Whenever the mem-table is full, data will be written into the SStable data file.
- All writes are automatically partitioned and replicated throughout the cluster.
- Cassandra periodically consolidates the SSTables, discarding unnecessary data.

Write Operations



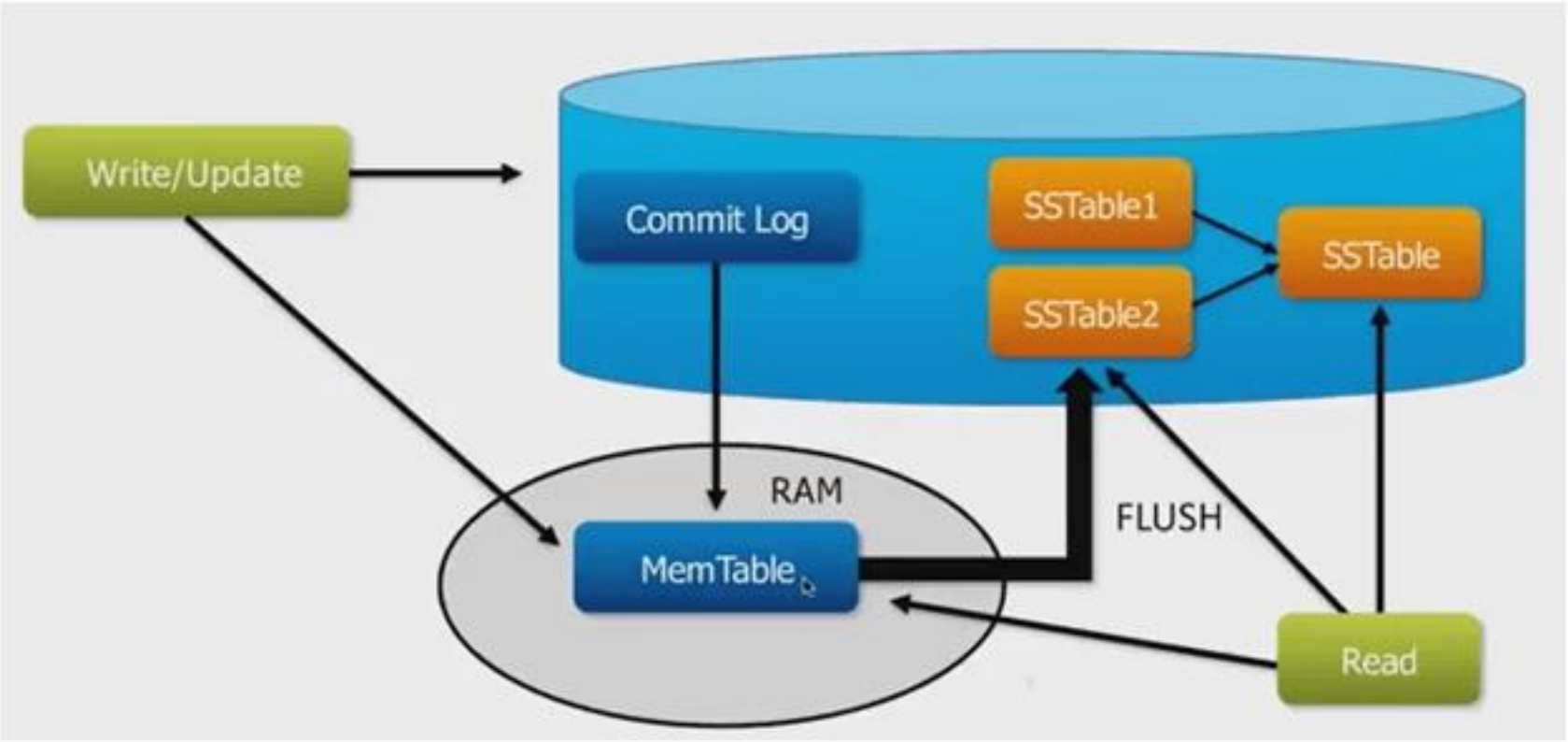
Read Operations

- In Read operations, Cassandra gets values from the mem-table and checks the bloom filter to find the appropriate SSTable which contains the required data.
- There are three types of read request that is sent to replicas by coordinators.
  - Direct request
  - Digest request
  - Read repair request

Read Operations

- The coordinator sends direct request to one of the replicas.
- After that, the coordinator sends the digest request to the number of replicas specified by the consistency level and checks if the returned data is an updated data.
- After that, the coordinator sends digest request to all the remaining replicas.
- If any node gives out of date value, a background read repair request will update that data. This process is called read repair mechanism.

Read Operations



Use Cases/ Applications of Cassandra

- Messaging
  - Cassandra is a great database which can handle a big amount of data. So it is preferred for the companies that provide Mobile phones and messaging services. These companies have a huge amount of data, so Cassandra is best for them.
- Handle high speed Applications
  - Cassandra can handle the high speed data so it is a great database for the applications where data is coming at very high speed from different devices or sensors.

Use Cases/ Applications of Cassandra

- Product Catalogs and retail apps
  - Cassandra is used by many retailers for durable shopping cart protection and fast product catalog input and output.
- Social Media Analytics and recommendation engine
  - Cassandra is a great database for many online companies and social media providers for analysis and recommendation to their customers.

Cassandra Data Types

CQL Type	Constants	Description
ascii	Strings	US-ascii character string
bigint	Integers	64-bit signed long
blob	blobs	Arbitrary bytes in hexadecimal
boolean	Booleans	True or False
counter	Integers	Distributed counter values 64 bit
decimal	Integers, Floats	Variable precision decimal
double	Integers, Floats	64-bit floating point
float	Integers, Floats	32-bit floating point
frozen	Tuples, collections, user defined types	stores cassandra types
inet	Strings	IP address in ipv4 or ipv6 format
int	Integers	32 bit signed integer

Cassandra Data Types

list		Collection of elements
map		JSON style collection of elements
set		Collection of elements
text	strings	UTF-8 encoded strings
timestamp	Integers, Strings	ID generated with date plus time
timeuuid	uuids	Type 1 uuid
tuple		A group of 2,3 fields
uuid	uuids	Standard uuid
varchar	strings	UTF-8 encoded string
varint	Integers	Arbitrary precision integer

Cassandra Data Model

- Cluster
  - Cassandra database is distributed over several machines that are operated together.
  - The outermost container is known as the Cluster which contains different nodes.
  - Every node contains a replica, and in case of a failure, the replica takes charge.
  - Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Cassandra Data Model

- Keyspace
- Keyspace is the outermost container for data in Cassandra. Following are the basic attributes of Keyspace in Cassandra:
- Replication factor: It specifies the number of machine in the cluster that will receive copies of the same data.
- Replica placement Strategy: It is a strategy which species how to place replicas in the ring. There are three types of strategies such as:
  - Simple strategy (rack-aware strategy)
  - old network topology strategy (rack-aware strategy)
  - network topology strategy (datacenter-shared strategy).

Cassandra Data Model

- Column families: column families are placed under keyspace.
- A keyspace is a container for a list of one or more column families while a column family is a container of a collection of rows.
- Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.
- In Cassandra, a well data model is very important because a bad data model can degrade performance, especially when you try to implement the RDBMS concepts on Cassandra.

Cassandra data Model Rules

- Cassandra doesn't support JOINS, GROUP BY, OR clause, aggregation etc. So you have to store data in a way that it should be retrieved whenever you want.
- Cassandra is optimized for high write performances so you should maximize your writes for better read performance and data availability.
- There is a tradeoff between data write and data read. So, optimize you data read performance by maximizing the number of data writes.
- Maximize data duplication because Cassandra is a distributed database and data duplication provides instant availability without a single point of failure.

Data Modeling Goals

- Spread Data Evenly Around the Cluster: To spread equal amount of data on each node of Cassandra cluster, you have to choose integers as a primary key. Data is spread to different nodes based on partition keys that are the first part of the primary key.
- Minimize number of partitions read while querying data: Partition is used to bind a group of records with the same partition key. When the read query is issued, it collects data from different nodes from different partitions.
- In the case of many partitions, all these partitions need to be visited for collecting the query data. It does not mean that partitions should not be created. If your data is very large, you can't keep that huge amount of data on the single partition. The single partition will be slowed down. So you must have a balanced number of partitions

Cassandra vs Hbase

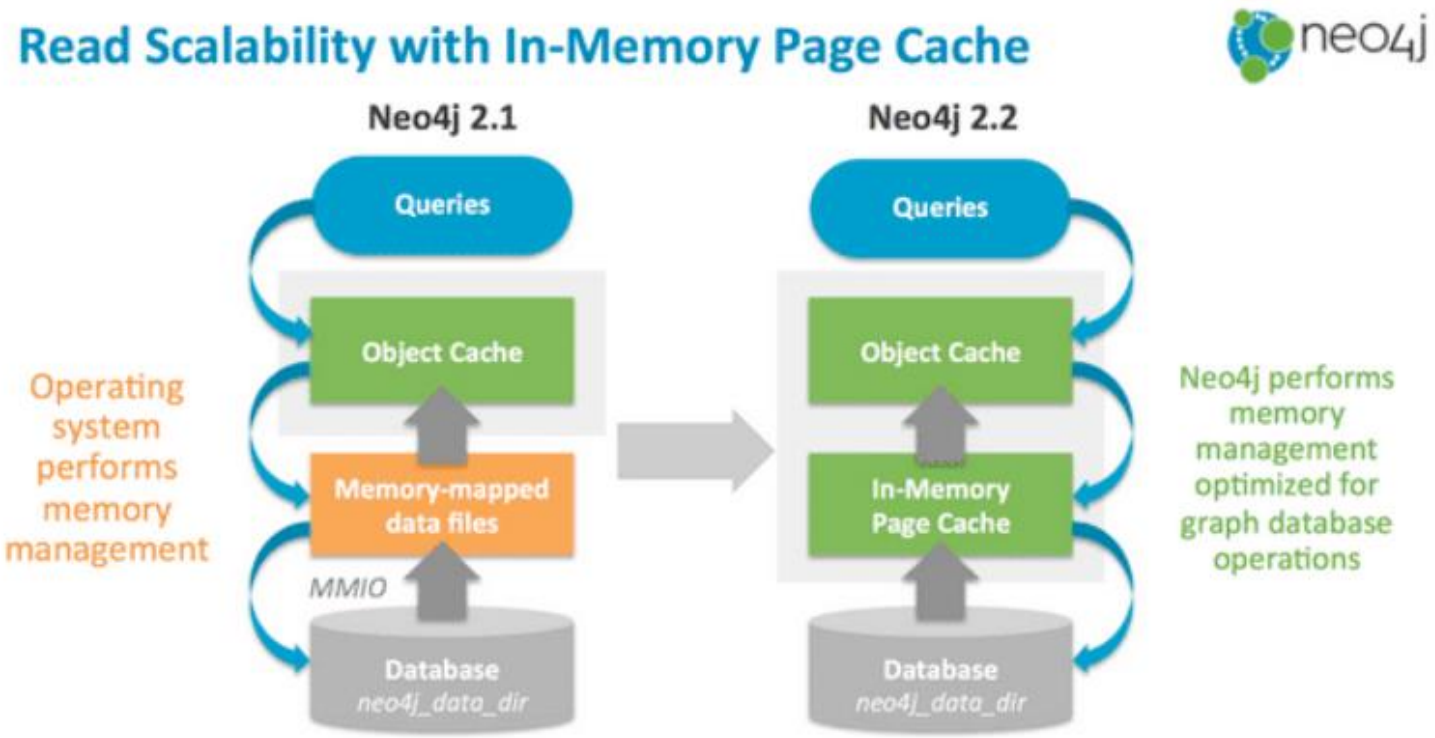
HBase	Cassandra
HBase is based on Bigtable (Google)	Cassandra is based on DynamoDB (Amazon). It was initially developed at Facebook by former Amazon engineers. This is one reason why Cassandra supports multi data center.
HBase uses the Hadoop infrastructure (Zookeeper, NameNode, HDFS). Organizations that deploy Hadoop must have the knowledge of Hadoop and HBase	Cassandra started and evolved separate from Hadoop and its infrastructure and operational knowledge requirements are different than Hadoop. However, for analytics, many Cassandra deployments use Cassandra + Storm (which uses zookeeper), and/or Cassandra + Hadoop.
The HBase-Hadoop infrastructure has several "moving parts" consisting of Zookeeper, Name Node, HBase master, and data nodes, Zookeeper is clustered and naturally fault tolerant. Name Node needs to be clustered to be fault tolerant.	Cassandra uses a single node-type. All nodes are equal and perform all functions. Any node can act as a coordinator, ensuring no Spof. Adding storm or Hadoop, of course, adds complexity to the infrastructure.
HBase is well suited for doing range based scans.	Cassandra does not support range based row-scans which may be limiting in certain use-cases.

Cassandra vs Hbase

HBase provides for asynchronous replication of an HBase cluster across a wan.	Cassandra random partitioning provides for row-replication of a single row across a wan.
HBase only supports ordered partitioning.	Cassandra officially supports ordered partitioning, but no production user of Cassandra uses ordered partitioning due to the "hot spots" it creates and the operational difficulties such hot-spots cause.
Due to ordered partitioning, HBase will easily scale horizontally while still supporting Rowkey range scans.	If data is stored in columns in Cassandra to support range scans, the practical limitation of a row size in Cassandra is 10's of megabytes.
HBase supports atomic compare and set. HBase supports transaction within a row.	Cassandra does not support atomic compare and set.
HBase does not support read load balancing against a single row. A single row is served by exactly one region server at a time.	Cassandra will support read load balancing against a single row.
Bloom filters can be used in HBase as another form of indexing.	Cassandra uses bloom filters for key lookup.
Triggers are supported by the coprocessor capability in HBase.	Cassandra does not support co-processor-like functionality.

Neo4j Graphbased Databases

Neo4j Graphbased Databases



Neo4j Graphbased Databases

Intuitive Data Model
Specifically designed to store and manage interconnected data

Native Graph Processing
Specifically designed to store and manage in-terconnected data

ACID
Fully transactional database to ensure data integrity

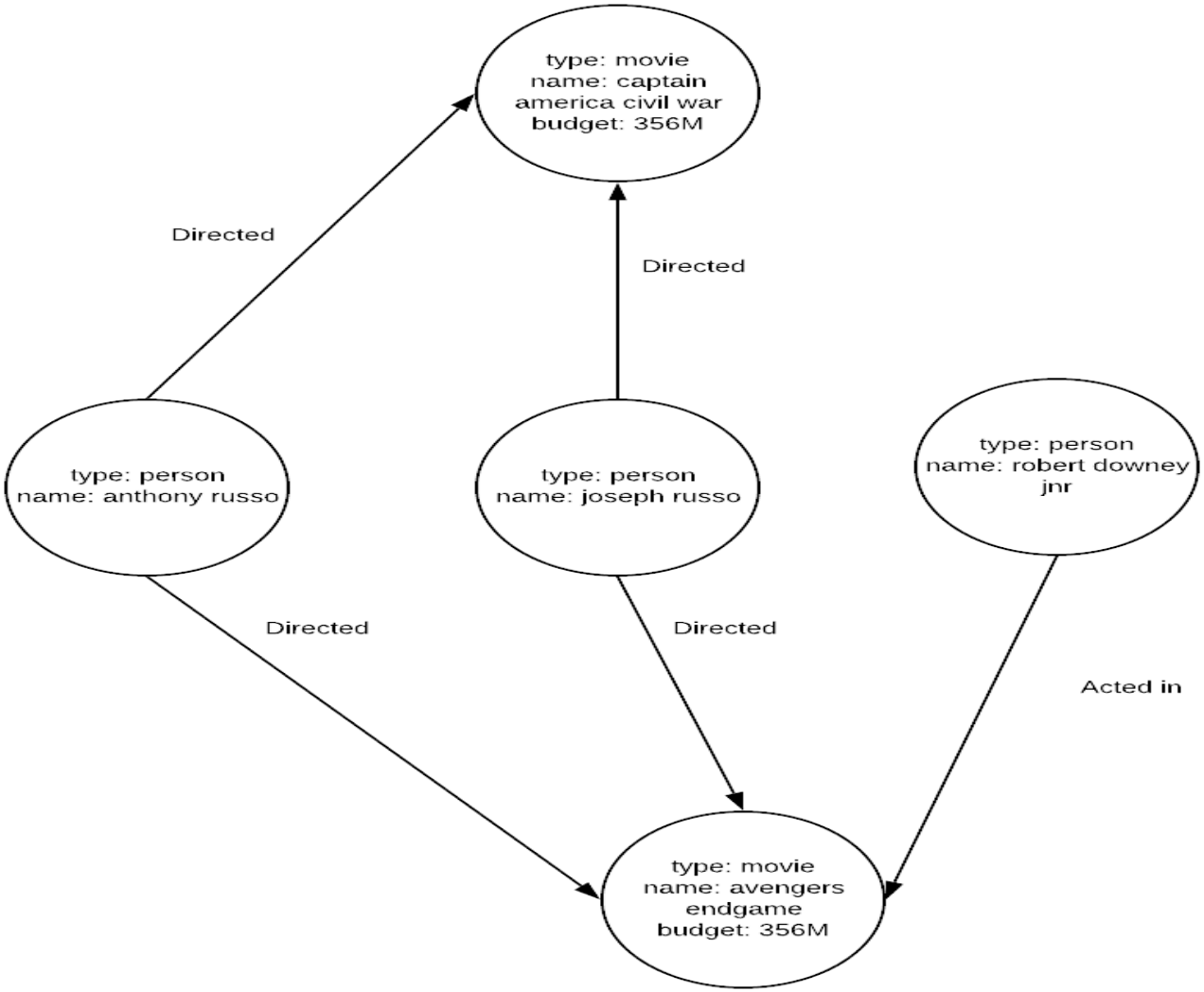
Expressive Cypher Language
The Property Graph Model captures data as it naturally occurs

Speed
Constant-time local graph traversals, regardless of data size

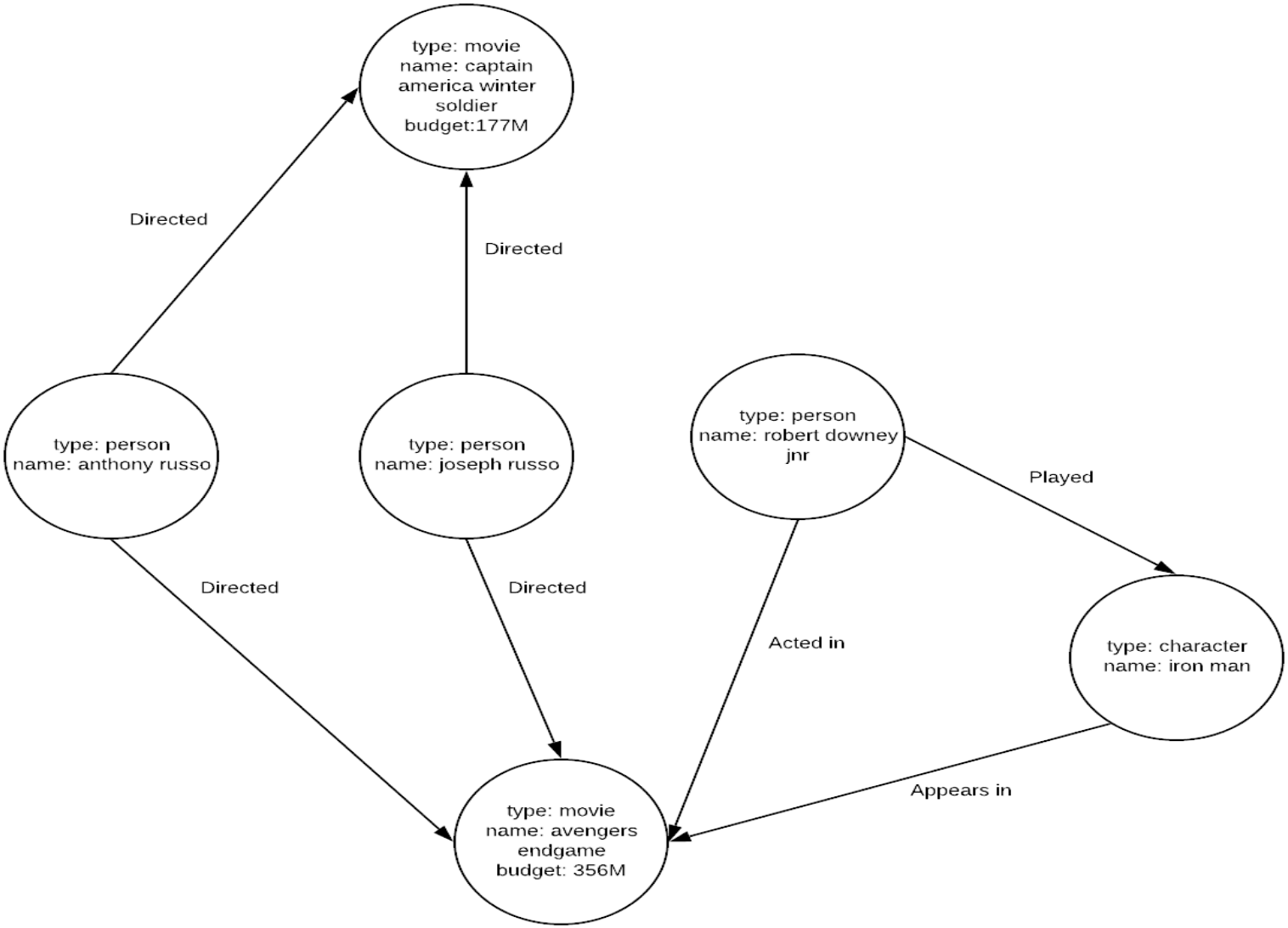
Scale
Highly performant read and write scalability without sacrificing data integrity

neo4j

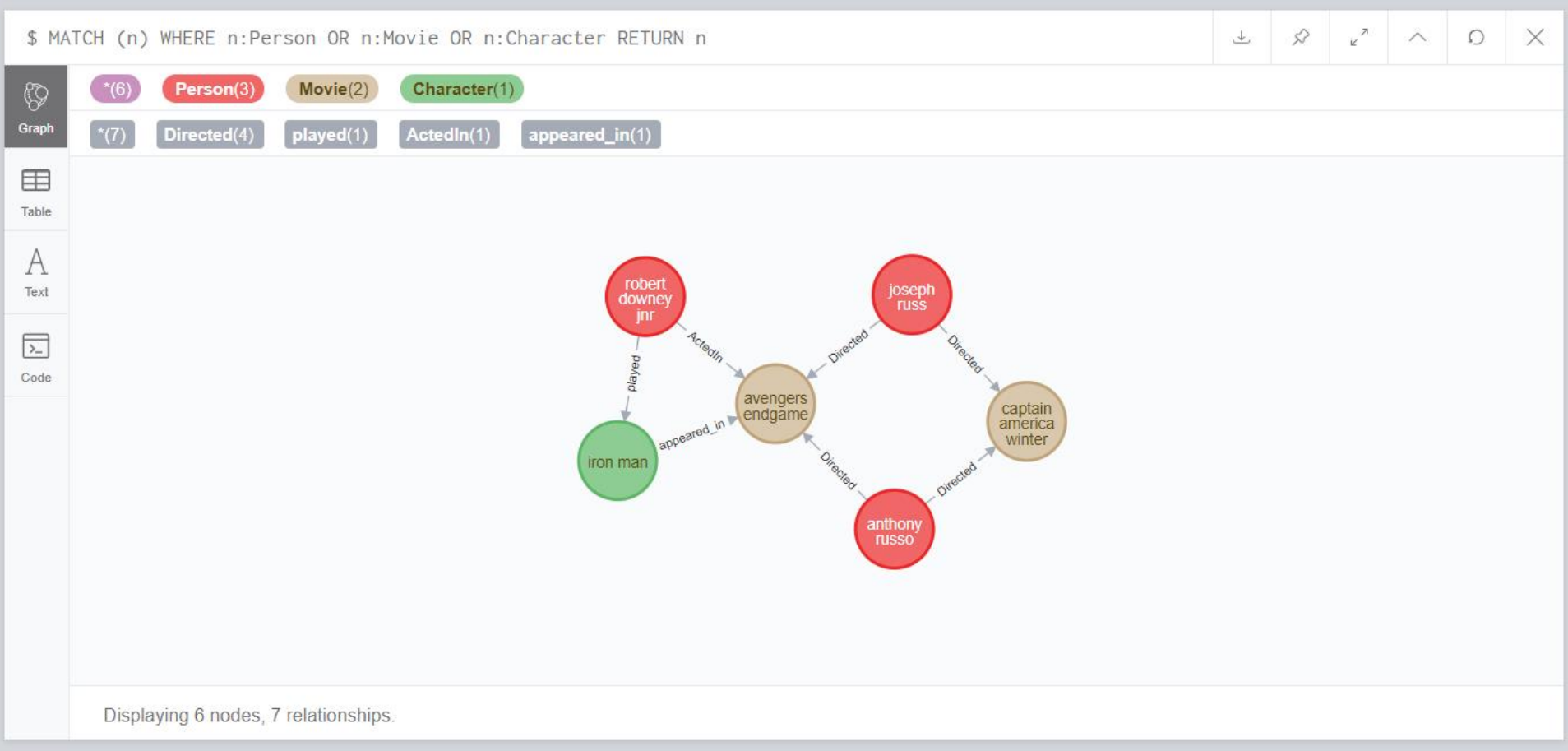
Neo4j Graphbased Databases



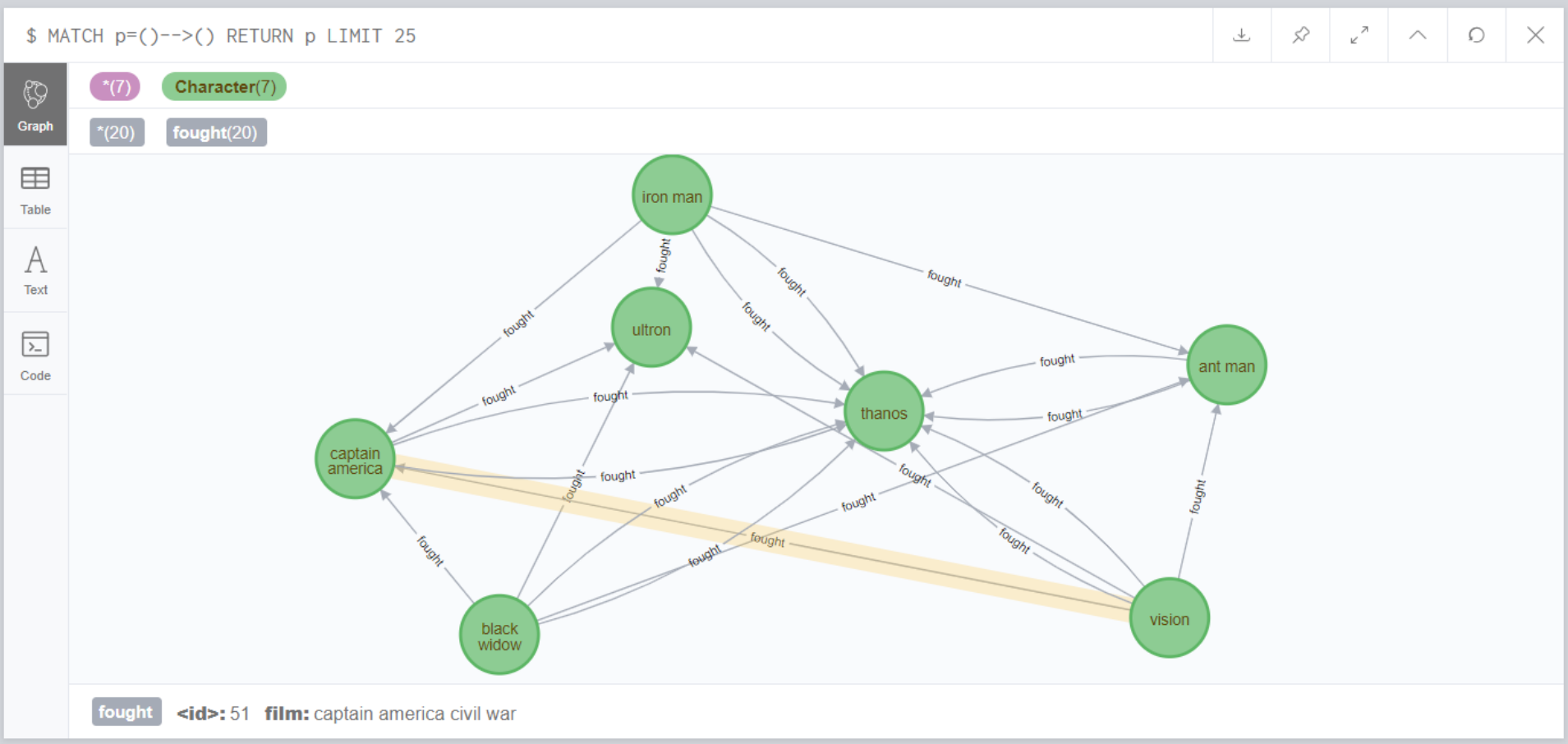
Neo4j Graphbased Databases



Neo4j Graphbased Databases



Neo4j Graphbased Databases



Neo4j Graphbased Databases

- cd neo4j bin folder
- neo4j-admin set-initial-password xxxxxx
- ./cypher-shell -a bolt://localhost:7687 -u neo4j -p xxxxx

Neo4j Graphbased Databases

- create (n);
-
- 0 rows available after 13 ms, consumed after another 0 ms
- Added 1 nodes

Neo4j Graphbased Databases

- create(x) return x ;

- +----+
- | x |
- +----+
- | () |
- +----+

Neo4j Graphbased Databases

- create (o),(p);
- 0 rows available after 13 ms, consumed after another 0 ms
- Added 2 nodes

Neo4j Graphbased Databases

- create (s:Student);
- 0 rows available after 16 ms, consumed after another 0 ms
- Added 1 nodes, Added 1 labels
- create (s:Student:Biology) return s;
- CREATE (x:Employees { name: 'Walker', title: 'Tech Writer' })
- CREATE (y:Employees { name: 'Stephen', title: 'Manager' })
- return x,y;

Neo4j Graphbased Databases

- MATCH (x:Employees),(y:Employees)
- WHERE y.name = "Stephen" AND x.name = "Walker"
- CREATE (x)-[r:BOSS]->(y)
- RETURN type(r);
- MATCH (x:Employees),(y:Employees)
- WHERE y.name = "Stephen" AND x.name = "Walker"
- CREATE (y)-[r:EMPLOYEEOF]->(x)
- RETURN type(r);

Module Summary

- What is a NoSQL database?
- NoSQL vs. SQL
- Types and examples
- When to use NoSQL
- Microservices, polyglot persistence and NoSQL
- NoSQL and Cloud