Lecture-10
Big Data Systems (SEZG522/CCZG522)

BITS Pilani
Pilani Campus

Slides: Courtesy:.Prof. Anindya

innovate achieve lead

BITS Pilani

Pilani Campus

Second Semester

2024-25

Lecture-10 Contents

NoSQL Databases

- Introduction
- Types of NoSQL Databases
- SQL vs NoSQL
- MongoDB
- Graph Computing

Why NoSQL (1)

RDBMS meant for OLTP systems / Systems of Record

- Strict consistency and durability guarantees (ACID) over multiple data items involved in a transaction
- But they have scale and cost issues with large volumes of data, distributed geo-scale applications, very high transaction volumes

Typical web scale systems do not need strict consistency and durability for every use case

- Social networking
- Real-time applications
- Log analysis
- Browsing retail catalogs
- Reviews and blogs
- …

Why NoSQL (2)

RDBMS ensure uniform structure and modeling of relationships between entities

A class of emerging applications need granular and extreme connectivity information modelled between individual semi-structured data items. This information needs to be also queried at scale without large expensive joins.

- Connectivity between users in a social media application: How many friends do you have between 2 hops ?
- Connectivity between companies in terms of domain, technology, people skills, hiring : Useful for skills acquisition, M&A etc.
- Connectivity between IT network devices: Useful for troubleshooting incidents

What is NoSQL ?

Coined by Carlo Strozzi in 1998

- ✓ Lightweight, open source database without standard SQL interface

Reintroduced by Johan Oskarsson in 2009

- ✓ Non-relational databases

Characteristics

- ✓ Not Only SQL
- ✓ Non-relational
- ✓ Schema-less
- ✓ Loosen consistency to address scalability and availability requirements in large scale applications
- ✓ Open source movement born out of web-scale applications
- ✓ Distributed for scale
- ✓ Cluster Friendly

Data model

- Supports rich variety of data : structured, semi-structured and unstructured
- No fixed schema, i.e. each record could have different attributes
- Non-relational - no join operations are typically supported
- Transaction semantics for multiple data items are typically not supported
- Relaxed consistency semantics - no support for ACID as in RDBMS
- In some cases can model data as graphs and queries as graph traversals

Classification of NoSQL DBs

Key – value
- ✓Maintains a big hash table of keys and values
- ✓Example : Dynamo, Redis, Riak etc

Document
- ✓Maintains data in collections of documents
- ✓Example : MongoDB, CouchDB etc

Column
- ✓Each storage block has data from only one column
- ✓Example : Cassandra, HBase

Graph
- ✓Network databases
- ✓Graph stores data in nodes
- ✓Example : Neo4j, HyperGraphDB, Apache Tinkerpop

Characteristics

Scale out architecture instead of monolithic architecture of relational databases

- Cluster scale - distribution across 100+ nodes across DCs
- Performance scale - 100K+ DB reads and writes per sec
- Data scale - 1B+ docs in DB

House large amount of structured, semi-structured and unstructured data

Dynamic schemas

✓ allows insertion of data without pre-defined schema

Auto sharding

✓ automatically spreads data across the number of servers
✓ applications are not aware about it
✓ helps in data balancing and failure from recovery

Replication

✓ Good support for replication of data which offers high availability, fault tolerance

Pros and Cons

- Cost effective for large data sets
- Easy to implement
- Easy to distribute esp across DCs
- Easier to scale up/down
- Relaxes data consistency when required
- No pre-defined schema
- Easier to model semi-structured data or connectivity data
- Easy to support data replication

- Joins between data sets / tables
- Group by operations
- ACID properties for transactions
- SQL interface
- Lack of standardisation in this space
  - Makes it difficult to port from SQL and across NoSQL stores
- Less skills compared to SQL
- Lesser BI tools compared to mature SQL BI space

SQL vs NoSQL

SQL	NoSQL
Relational database	Non relational, distributed databases
Pre-defined schema	Schema less
Table based databases	Multiple options: Key-Value, Document, Column, Graph
Vertically scalable	Horizontally scalable
Supports ACID properties	Supports CAP theorem
Supports complex querying	Relatively simpler querying
Excellent support from vendors	Relies heavily on community support

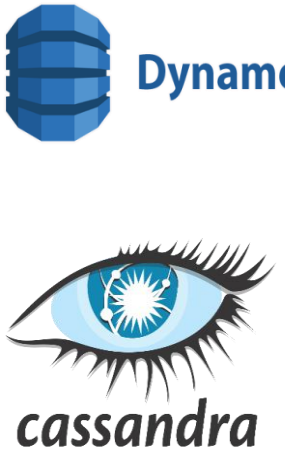
Vendors

Amazon

Facebook

Google

Oracle



Classification: Document-based

Store data in form of documents using well known formats like JSON

Documents accessible via their id, but can be accessed through other index as well

Maintains data in collections of documents

Example,

- MongoDB, CouchDB, CouchBase

Book document :

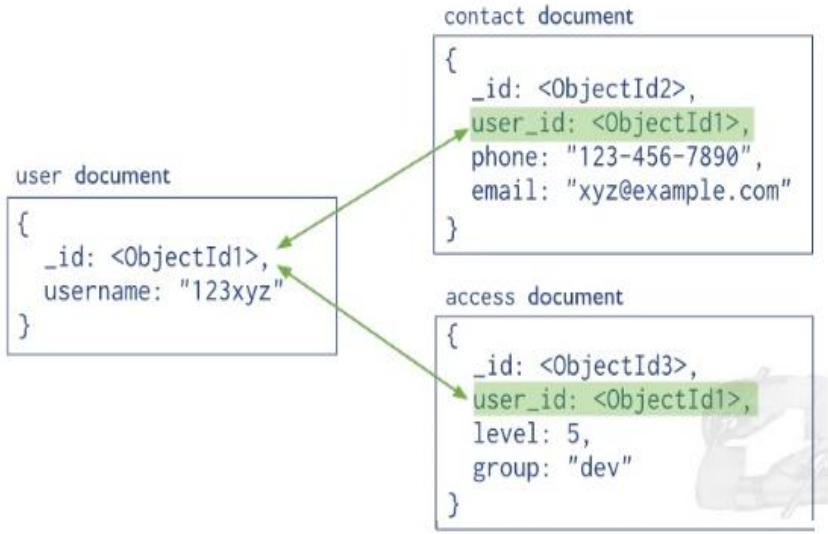
{

“Book Title” : “Database Fundamenta

“Publisher” : “My Publisher”,

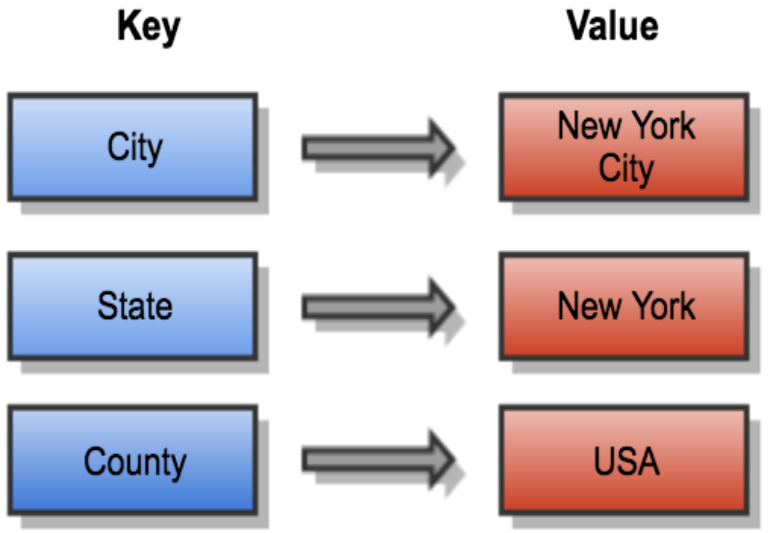
“Year of Publication” : “2020”

}



Classification: Key-Value store

- Simple data model based on fast access by the key to the value associated with the key
- Value can be a record or object or document or even complex data structure
- Maintains a big hash table of keys
- For example,
  - ✓ Dynamo, Redis, Riak

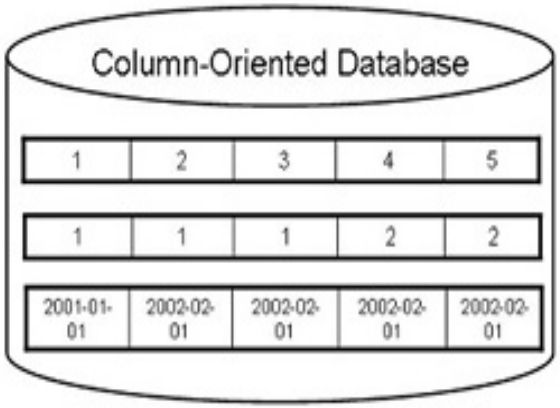


2014HW112220	{ Santosh,Sharma,Pilani}
2018HW123123	{Eshwar,Pillai,Hyd}

Classification: Column-based

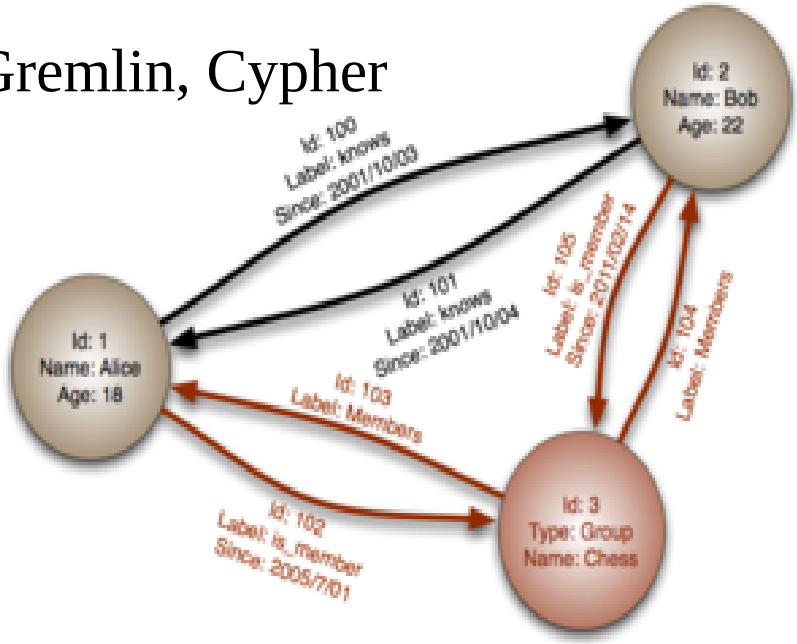
- Partition a table by column into column families
- A part of vertical partitioning where each column family is stored in its own files
- Allows versioning of data values
- Each storage block has data from only one column
- Example,
  - ✓ Cassandra, Hbase

Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura



Classification: Graph based

- Data is represented as graphs and related nodes can be found by traversing the edges using the path expression
- aka network database
- Graph query languages, e.g. Gremlin, Cypher
- Example
  - ✓ Neo4J
  - ✓ HyperGraphDB
  - ✓ Apache TinkerPop



MongoDB

Database is a set of collections

A collection is like a table in RDBMS

A collection stores documents

- BSON or Binary JSON with hierarchical key-value pairs
- Similar to rows in a table

MongoDB

Data is partitioned in shards

- For horizontal scaling
- Reduces amount of data each shard handles as the cluster grows
- Reduces number of operations on each shard

Data is replicated

- Writes to primary in oplog. “write-preference” setting used to tweak write consistency.
- Secondaries use oplog to get local copies updated
- Clients usually read from primary but “read-preference” setting can tweak read consistency

Data updates happen in place and not versioned / timestamped

MongoDB: Indexing

Can create index on any field of a collection or a sub-document fields

e.g. document in a collection

```
{
    "address": {
        "city": “New Delhi",
        "state": "Delhi",
        "pincode": "110001"
    },
    "tags": [
        "football",
        "cricket",
        "badminton"
    ],
    "name": "Ravi"
}
```

indexing a field in ascending order and find

```
> db.users.createIndex({“tags":1})
> db.users.find({tags:"cricket"}).pretty()
```

indexing a sub-document field in ascending order and find

```
> db.users.createIndex({"address.city":1,"address.state":1,"address.pincode":1})
```

MongoDB: Joins

Mongo 3.2+ it is possible to join data from 2 collections using aggregate

Collection books (isbn, title, author) and books_selling_data(isbn, copies_sold)

```
db.books.aggregate([{ $lookup: {
          from: "books_selling_data",
          localField: "isbn",
          foreignField: "isbn",
          as: "copies_sold"
        }
 }])
```

Sample joined document:

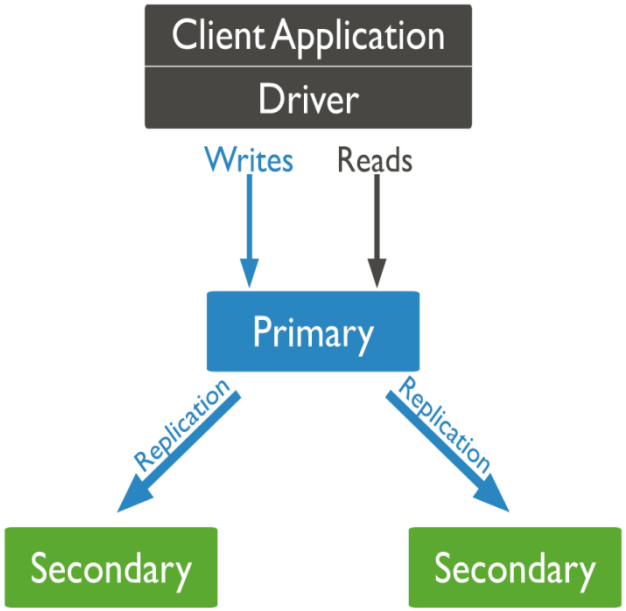
```
{
   "isbn": "978-3-16-148410-0",
   "title": "Some cool book",
   "author": "John Doe",
   "copies_sold": [
       {
           "isbn": "978-3-16-148410-0",
           "copies_sold": 12500
       }
    ]
}
```

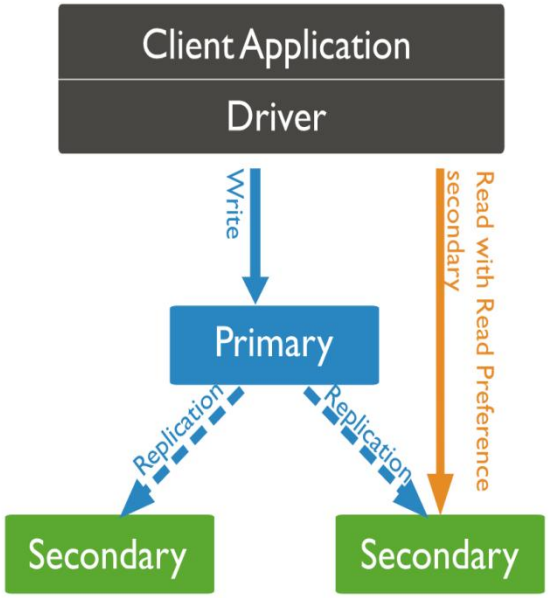
MongoDB

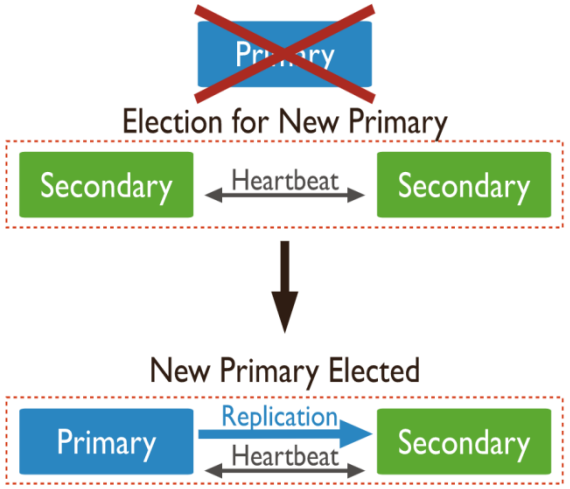
Document oriented DB

Various read and write choices for flexible consistency tradeoff with scale / performance and durability

Automatic primary re-election on primary failure and/or network partition

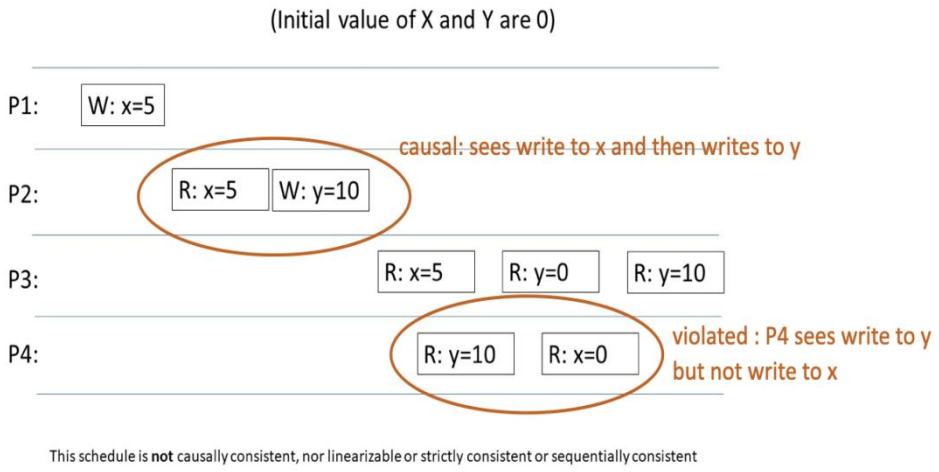






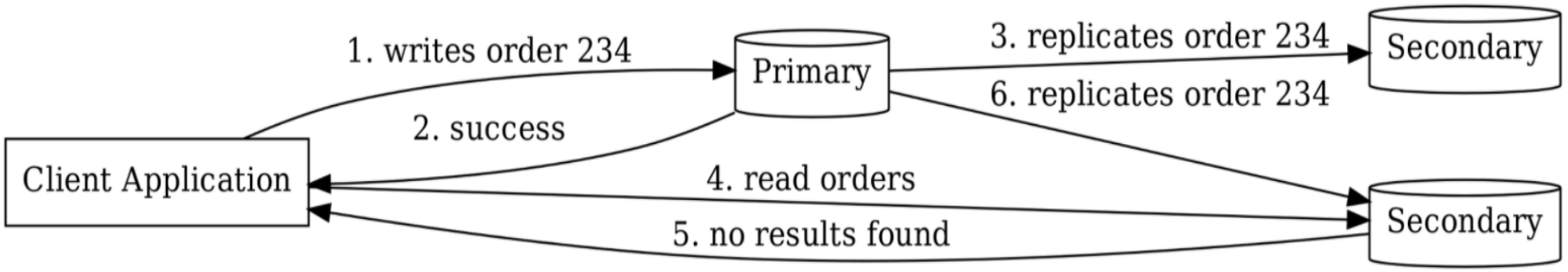
What is Causal Consistency (recap)

Read your writes	Read operations reflect the results of write operations that precede them.
Monotonic reads	Read operations do not return results that correspond to an earlier state of the data than a preceding read operation.
Monotonic writes	Write operations that must precede other writes are executed before those other writes.
Writes follow reads	Write operations that must occur after read operations are executed after those read operations.

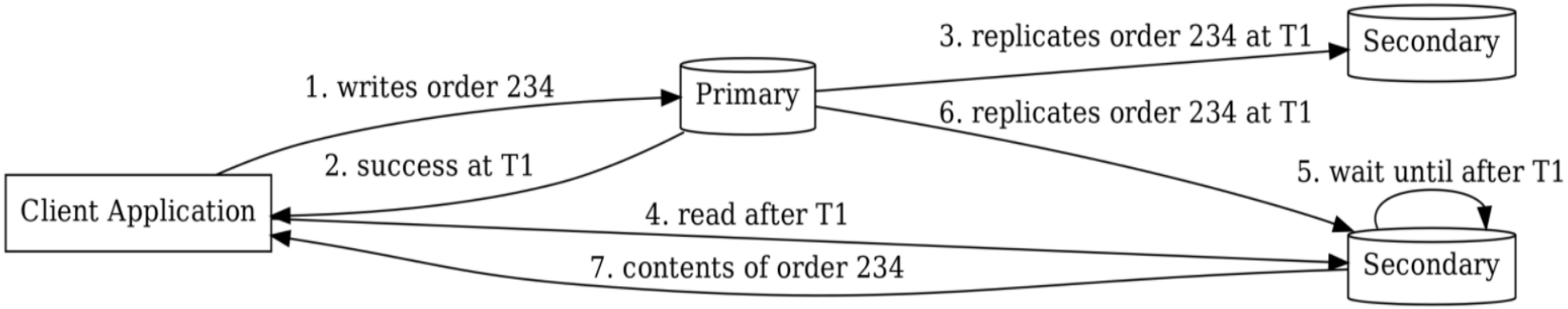


Example in MongoDB

Case 1 : No causal consistency



- Case 2: Causal consistency by making read to secondary wait



https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

MongoDB “read concerns”

local :

- Client reads primary replica
- Client reads from secondary in causally consistent sessions

available:

- Read on secondary but causal consistency not required

majority :

- If client wants to read what majority of nodes have. Best option for fault tolerance and durability.

linearizable :

- If client wants to read what has been written to majority of nodes before the read started.
- Has to be read on primary
- Only single document can be read

https://docs.mongodb.com/v3.4/core/read-preference-mechanics/

MongoDB “write concerns”

how many replicas should ack

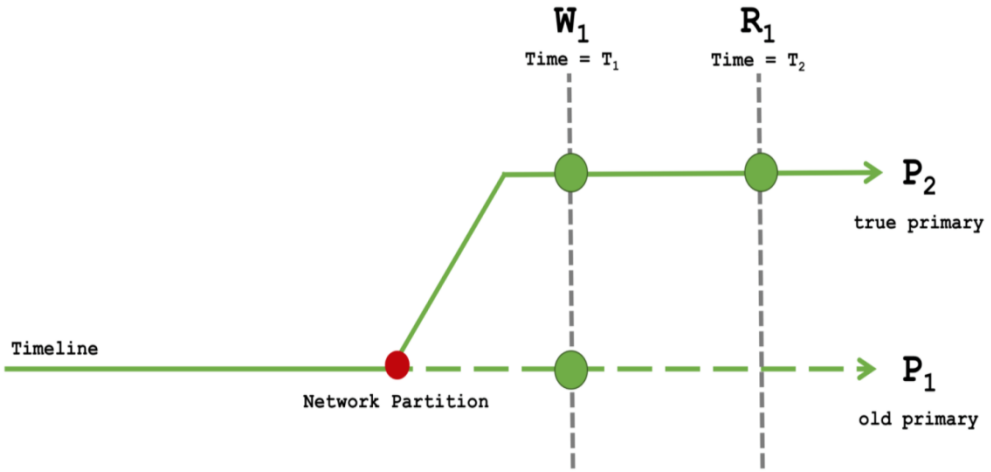
- 1 - primary only
- 0 - none
- n - how many including primary
- majority - a majority of nodes (preferred for durability)

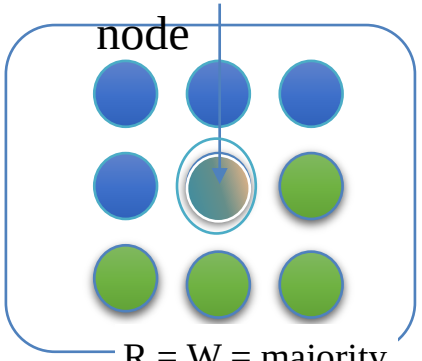
journaling - If True then nodes need to write to disk journal before ack else ack after writing to memory (less durable)

timeout for write operation

https://docs.mongodb.com/manual/reference/write-concern/

Consistency scenarios - causally consistent and durable





read=majority, write=majority

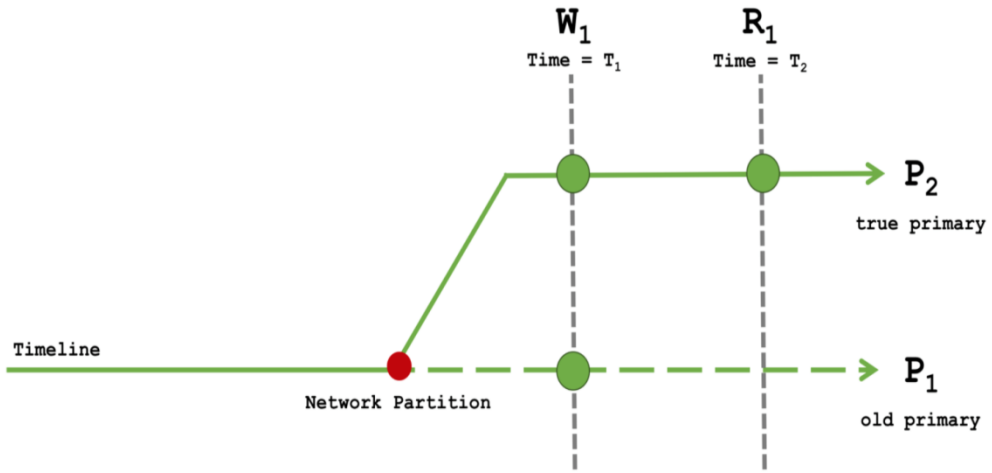
W1 and R1 for P1 will fail and will succeed in P2

So causally consistent, durable even with network partition sacrificing performance

Example: Used in critical transaction oriented applications, e.g. stock trading

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

Consistency scenarios - causally consistent but not durable



read=majority, write=1

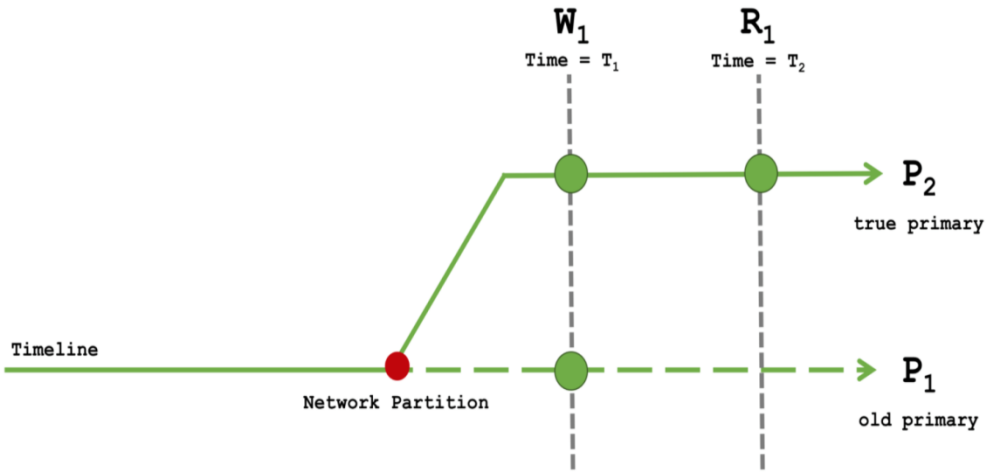
W1 may succeed on P1 and P2. R1 will succeed only on P2. W1 on P1 may roll back.

So causally consistent but not durable with network partition. Fast writes, slower reads.

Example: Twitter - a post may disappear but if on refresh you see it then it should be durable, else repost.

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

Consistency scenarios - eventual consistency with durable writes



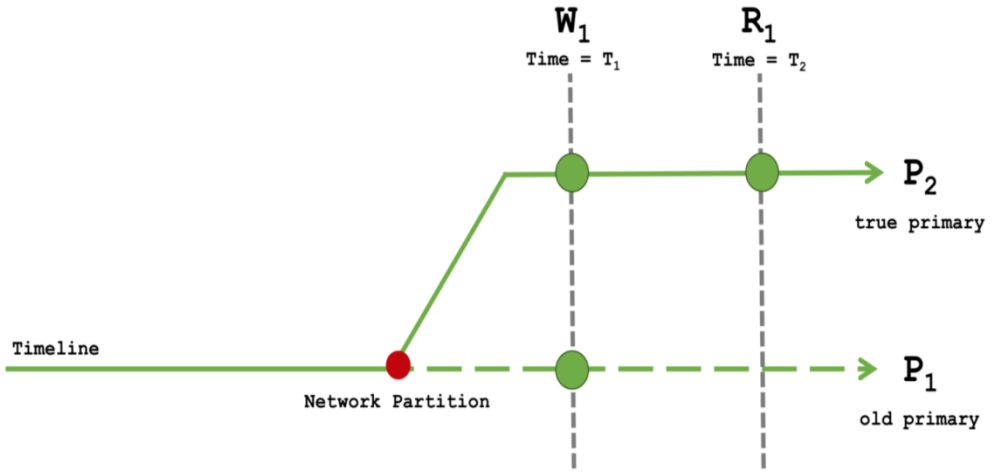
read=local, write=majority

W1 will succeed only for P2 and will not be accepted on P1 after failure. Reads may not succeed to see the last write on P1. Slow durable writes and fast non-causal reads.

Example: Review site where write should be durable if committed but reads don't need causal guarantee as long as it appears some time (eventual consistency).

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

Consistency scenarios - eventual consistency but no durability



read=local, write=1

Same as previous scenario and not writes are also not durable and may be rolled back.

Example: Real-time sensor data feed that needs fast writes to keep up with the rate and reads should get as much recent real-time data as possible. Data may be dropped on failures.

https://engineering.mongodb.com/post/ryp0ohr2w9pvv0fks88kq6qkz9k9p3

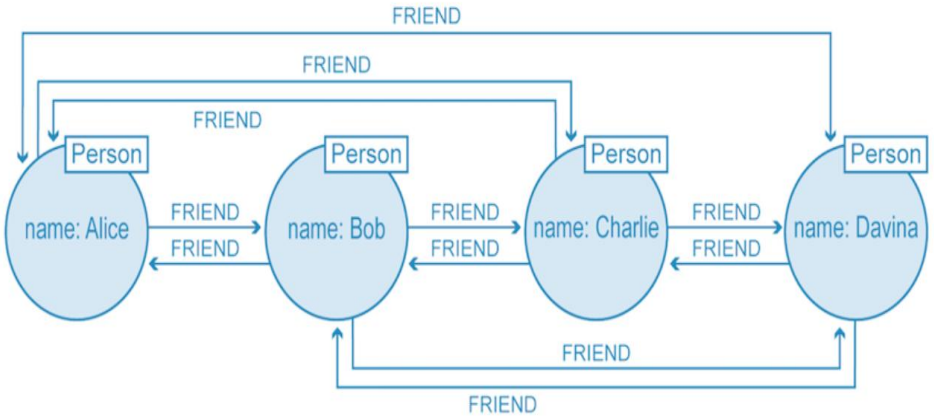
Graph computing

Property graphs

- Data is represented as vertices and edges with properties
- Properties are key value pairs
- Edges are relationships between vertices

When to use a graph DB ?

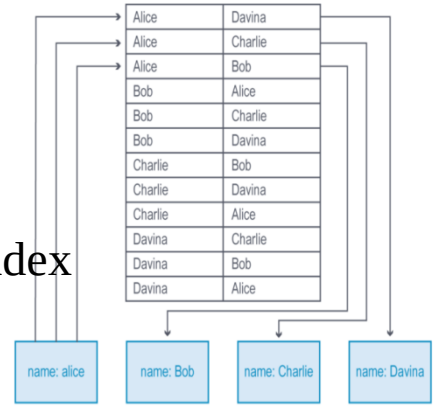
- A relationship-heavy data set with large set of data items
- Queries are like graph traversals but need to keep query performance almost constant as database grows
- A variety of queries may be asked from the data and static indices on data will not work



Native vs Non-Native Graph storage

- Non-native graph computing platforms can use external DBs for data storage
  - e.g. TinkerPop is an in-memory DB + computing framework that can store in ElasticSearch, Cassandra etc.
- Native platform support built-in storage
  - e.g. Neo4j
- Native approach is much faster because adjacent nodes and edges are stored closer for faster traversal
  - In a non-native approach, extensive indexing has to be used
- Native approach scales as nodes get added

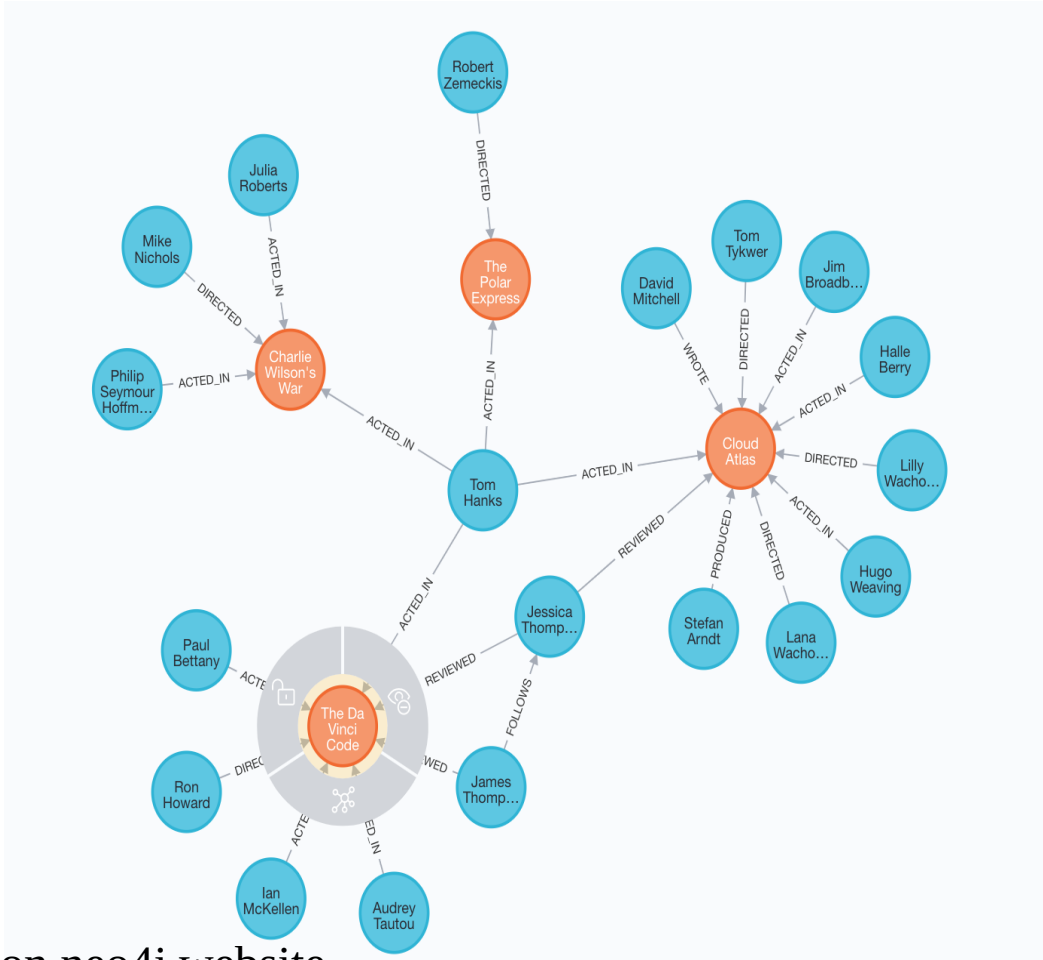
One-hop index



https://neo4j.com/blog/native-vs-non-native-graph-technology/

Neo4j / Cypher

- Cypher is a Declarative language for graph query
- Example: match (:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie) where m.released > 2000 RETURN m limit 5



Launch a free sandbox with dataset on neo4j website

Neo4j / Cypher: More queries

Find movies that Tom Hanks acted in and directed by Ron Howard released after 2000

- Match (:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie),(:Person {name: 'Ron Howard'})-[:DIRECTED]->(m) where m.released > 2000 RETURN m limit 5

Who were the other actors in the movie where Tom Hanks acted in and directed by Ron Howard released after 2000

- Match (:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(m:Movie),(:Person {name: 'Ron Howard'})-[:DIRECTED]->(m), (p:Person)-[:ACTED_IN]->(m) where m.released > 2000 RETURN p limit 5

THANK YOU