Non-relational and Modern Data Stores

Prof. Daniel S. Kaster

Departamento de Computação
Universidade Estadual de Londrina
dskaster@uel.br

Outline

- The need for scalability
- Data-driven scale-out techniques
- The CAP theorem
- NoSQL solutions
  - Key-value stores (Redis)
  - Columnar and column family stores (Cassandra)
  - Document stores (MongoDB)
- NewSQL products

The need for scalability

- Google and Hadoop (2003-2007)
  - Google File System, Map-reduce, Big Table
- Other big tech companies facing scalability issues
  - Facebook (exhausting the capability of a MySQL-based sharding)
  - Amazon (facing problems to scale e-commerce transactions)

Cloud computing

- Applications shifted from rich desktop applications based on the client-server model to web-based applications
  - Data stores and application servers resided somewhere accessible via the Internet
- Amazon’s Infrastructure as a Service (IaaS): Amazon Web Services (AWS) (2006-2008)
  - Elastic Compute Cloud (EC2)
  - Simple Storage Service (S3)
  - Virtual Private Cloud (VPC)
- Other leading players followed this trend
  - Google Cloud Plataform, Microsoft Azure

Rich web applications

- Starting about 2004, an increasing number of websites were able to offer a far richer interactive experience
- This was enabled by the programming style known as AJAX (Asynchronous JavaScript and XML)
  - JavaScript within the browser communicates directly with a backend by transferring XML messages
- XML was soon superseded by JavaScript Object Notation (JSON)
  - JSON is a self-describing format similar to XML but is more compact and tightly integrated into the JavaScript language
  - JSON became the de facto format for storing—serializing—objects to disk

The nonrelational explosion

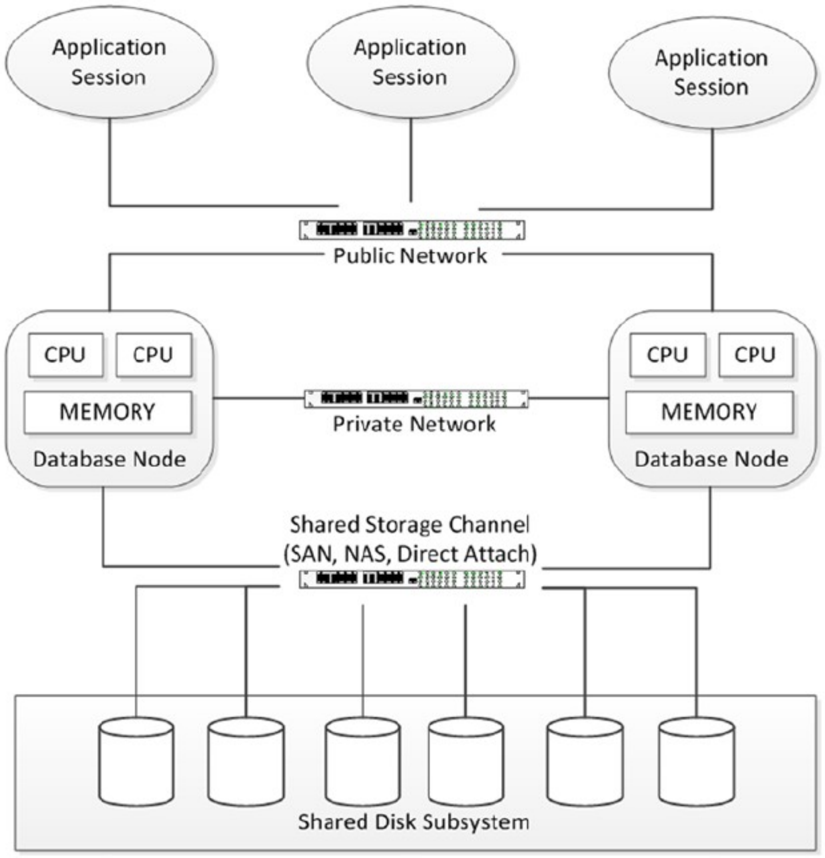
- Cloud, social media, big data, mobile widespread and internet of things have demanded innovative solutions
- Dozens of new database systems emerged in 2008–2009
  - Many of these have fallen into disuse, but some—such as MongoDB, Cassandra, and HBase—have today captured significant market share
- In late 2009, the term NoSQL (Not Only SQL) quickly caught on as shorthand for any database system that broke with the traditional relational database
  - New data models and strategies to scale-out the database system

The “New SQL”

- In 2007, Michael Stonebraker, pioneer of the Ingres and Postgres database systems, led a research team that published the seminal paper
  - “The End of an Architectural Era (It’s Time for a Complete Rewrite)”
    - The hardware assumptions that underlie the consensus relational architecture no longer applied
    - The variety of modern database workloads suggested a single architecture might not be optimal across all workloads (One Size Doesn’t Fit All)
  - H-Store (a pure in-memory distributed database)
  - C-Store (a design for a columnar database)
- By 2011, the term NewSQL became popularized as a means of describing this new breed of databases that, while not representing a complete break with the relational model, enhanced or significantly modified the fundamental principles

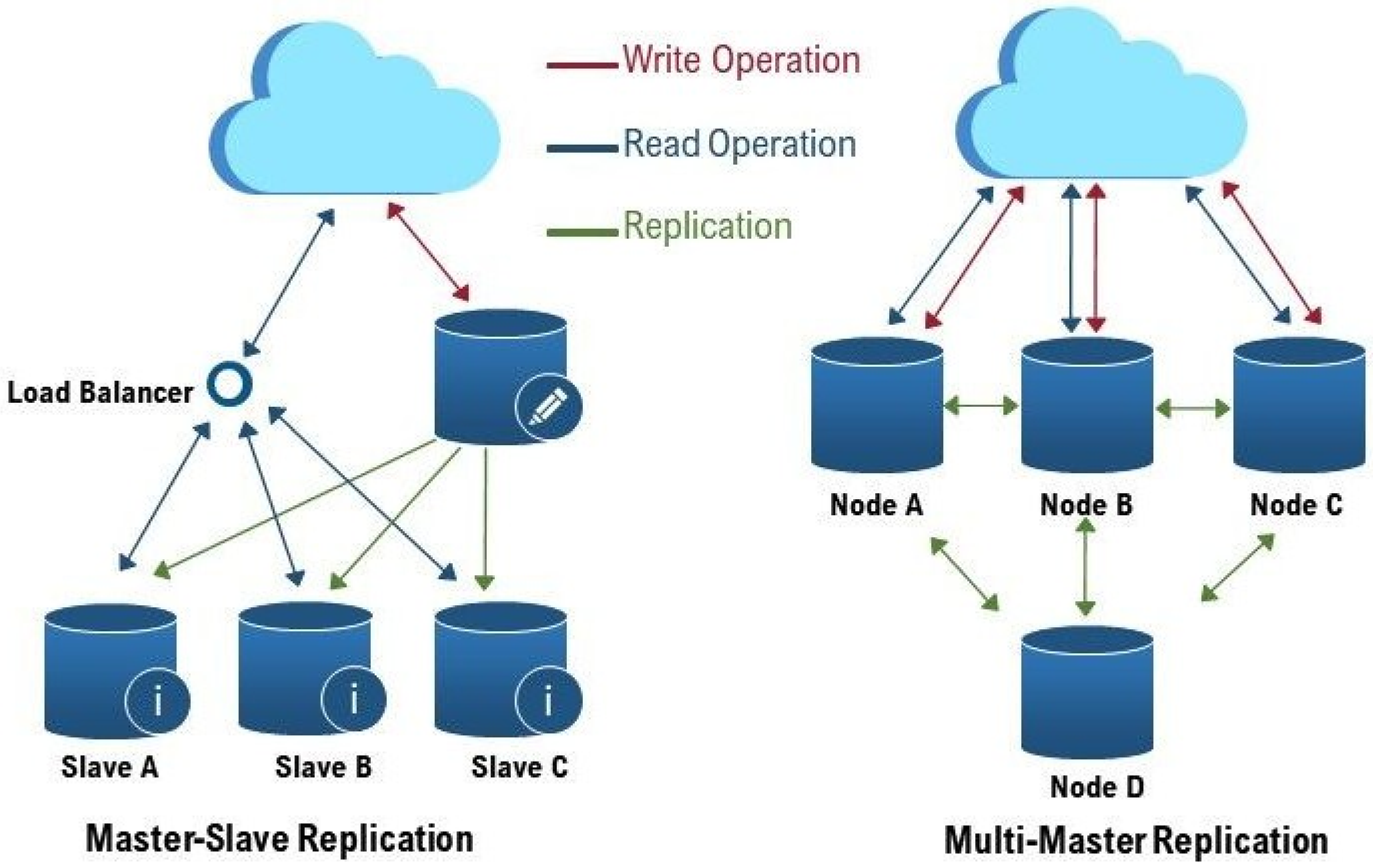
Data-driven application scale-out techniques

- Shared-disk clusters
  - Example: Oracle Real Application Cluster



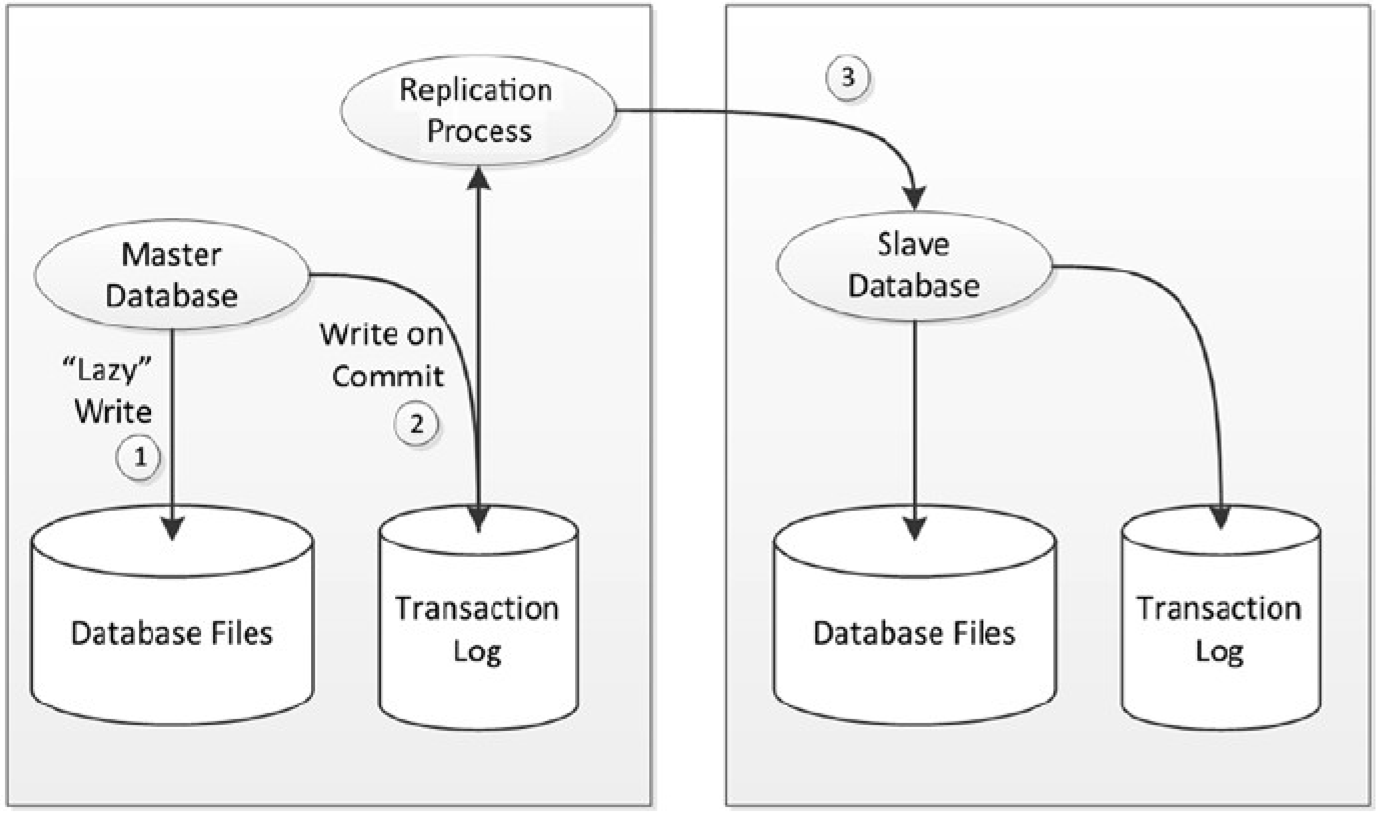
Data-driven application scale-out techniques(2)

- Replication



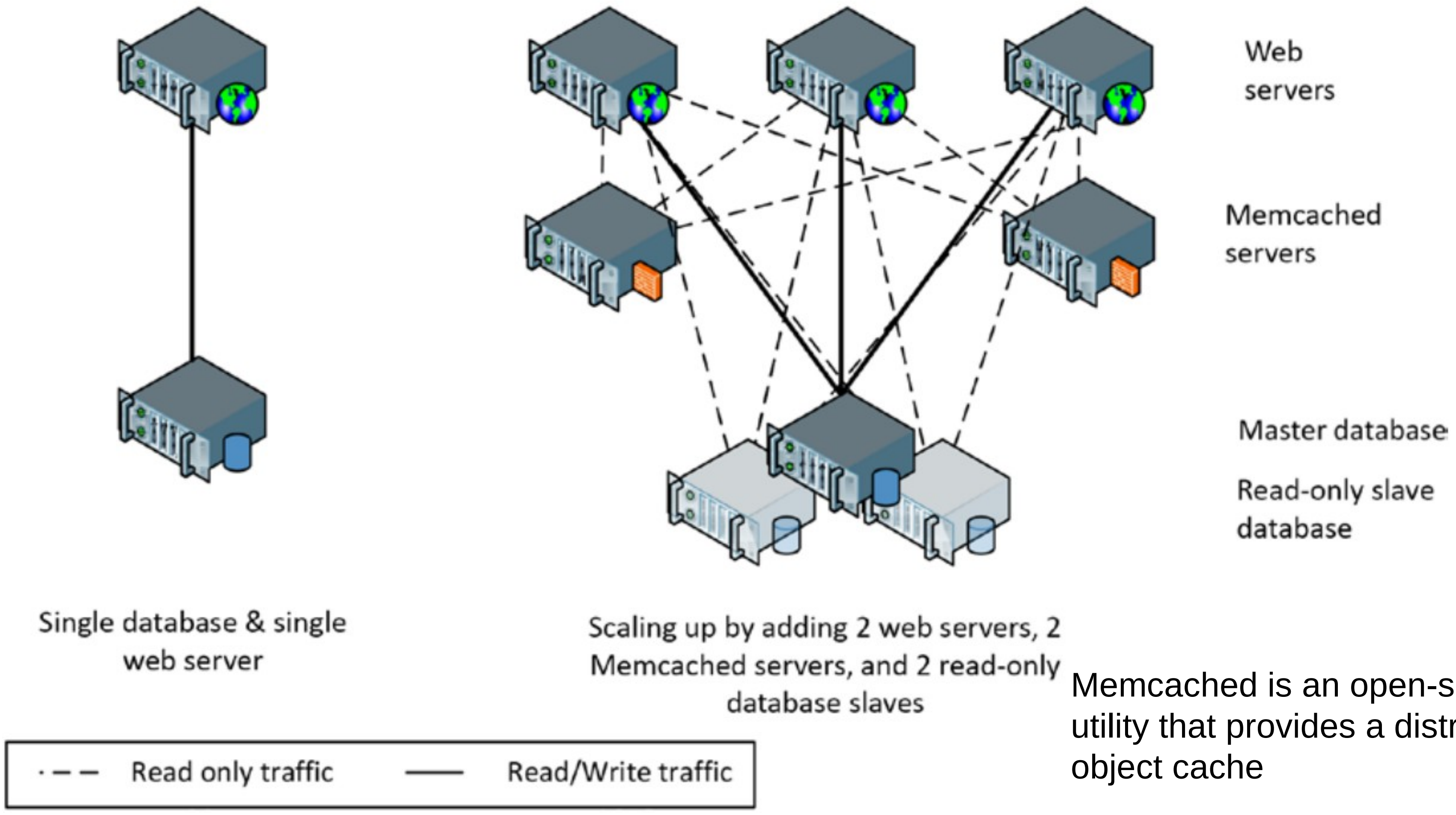
Data-driven application scale-out techniques(3)

- Log-based replications is a common pattern in DBMSs



Data-driven application scale-out techniques(4)

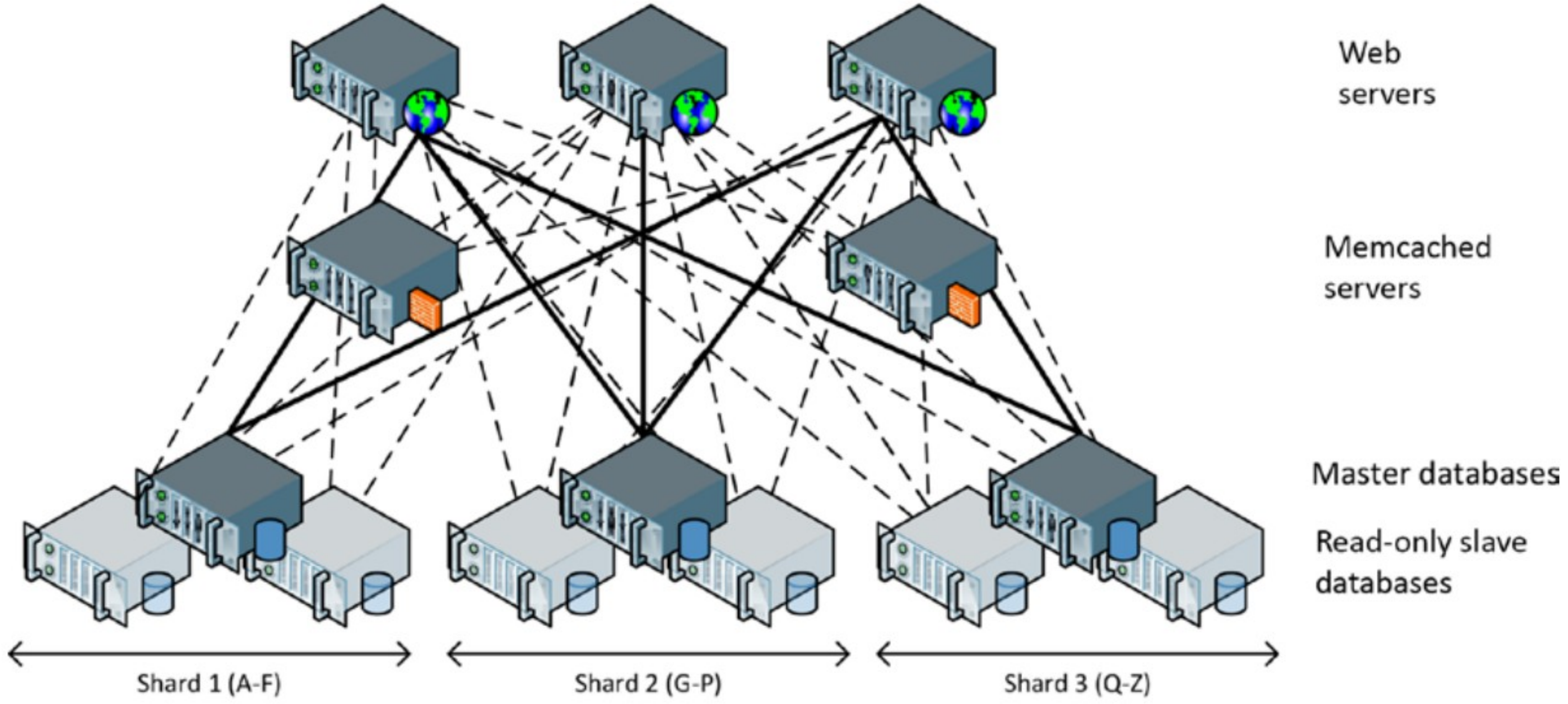
- Database and application server replication


Web servers
Memcached servers
Master database
Read-only slave database
Single database & single web server
Scaling up by adding 2 web servers, 2 Memcached servers, and 2 read-only database slaves
- - - Read only traffic
—— Read/Write traffic

Memcached is an open-source utility that provides a distributed object cache

Data-driven application scale-out techniques(5)

- Sharding



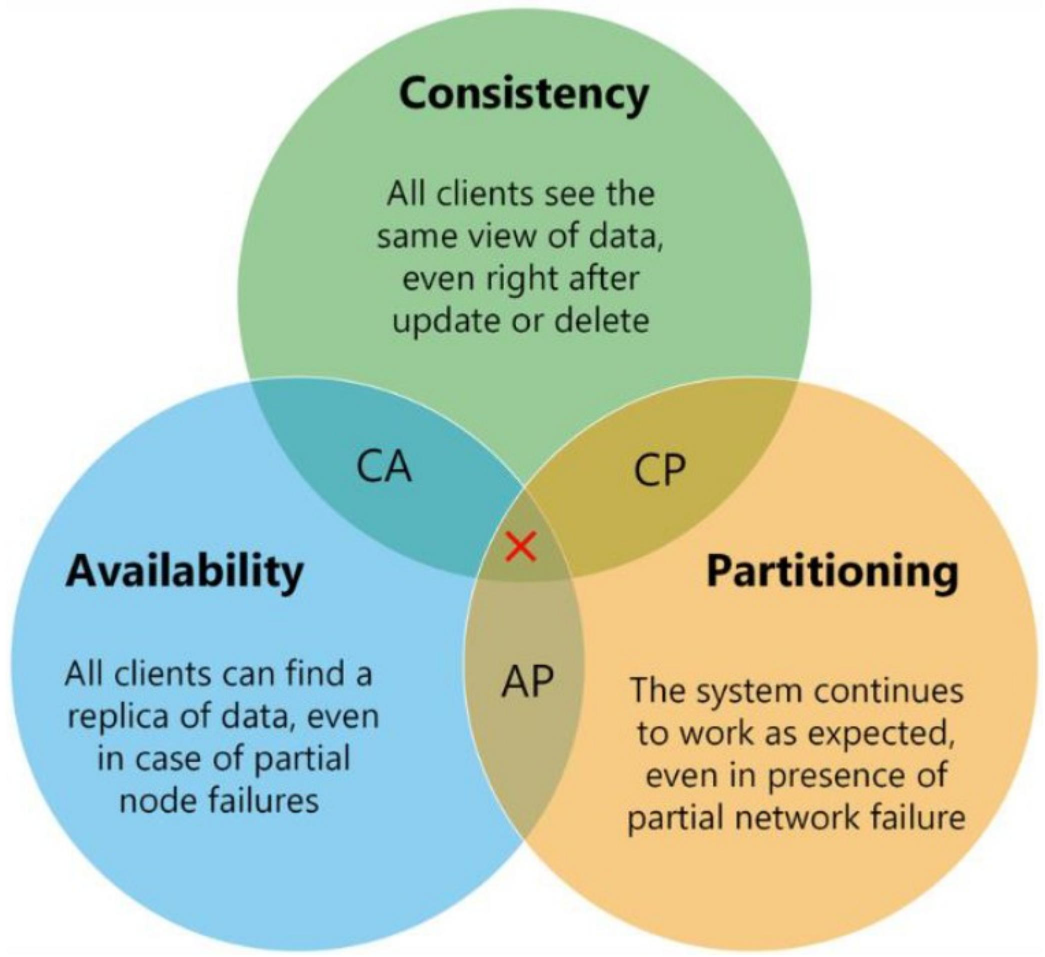
Scaling with 3 web servers, 2 Memcached servers, and 3 database shards with 2 read-only database slaves per shard

--read-only traffic --read/write traffic

Drawbacks of sharding non-sharded DBMSs

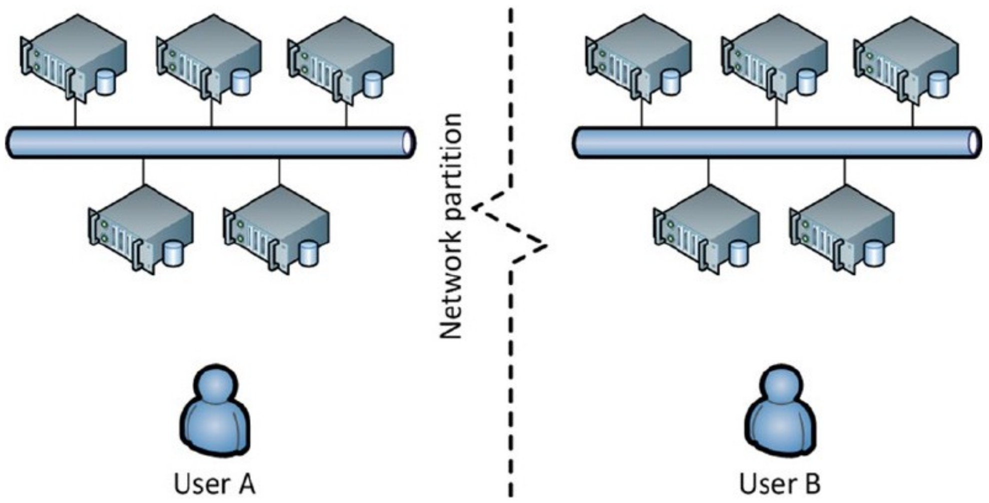
- Application complexity
  - It’s up to the application code to route SQL requests to the correct shard (dynamic routing is needed as shards evolve)
- Fragmented query execution
  - In a sharded database, it is not possible to issue a SQL statement that operates across shards
    - Joins across shards or aggregate GROUP BY operations require programming to access different shards
- Loss of transactional integrity
  - ACID transactions against multiple shards are available
- Operational complexity
  - Load balancing across shards becomes problematic as adding new shards requires a complex rebalancing of data
  - Entails operational effort and administrator skill

The CAP Theorem


Consistency
All clients see the same view of data, even right after update or delete
Availability
All clients can find a replica of data, even in case of partial node failures
Partitioning
The system continues to work as expected, even in presence of partial network failure
CA
CP
AP
×

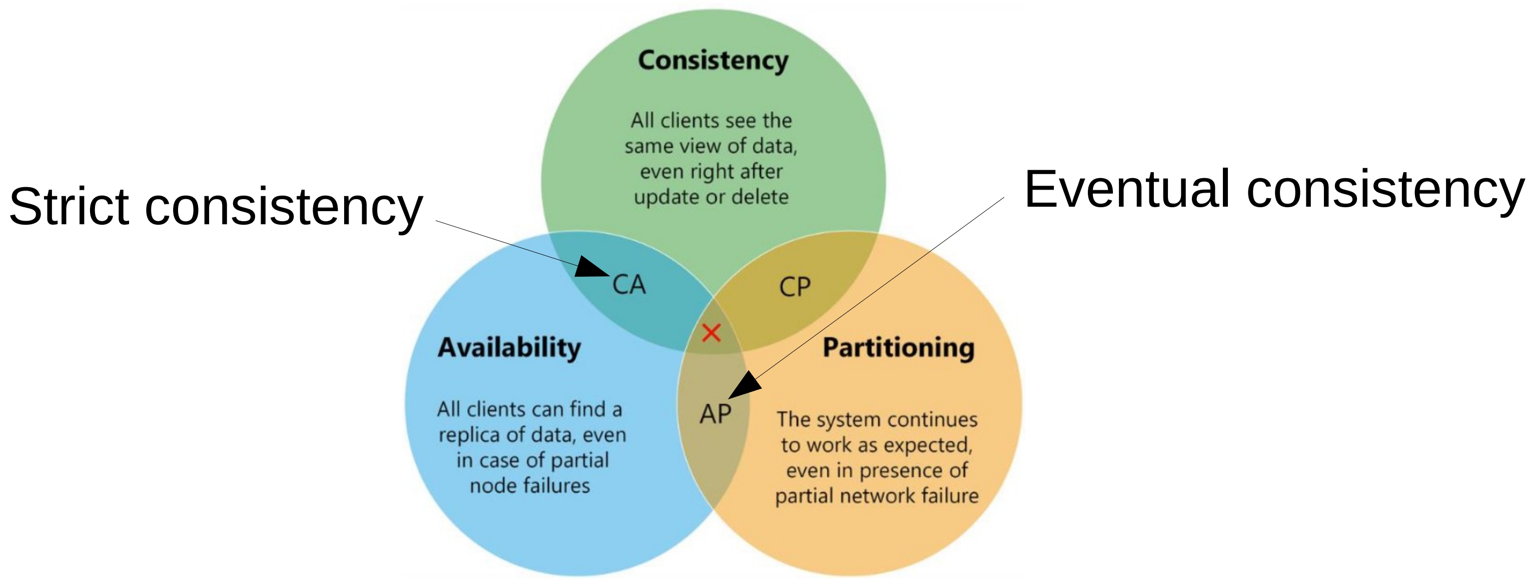
The “split brain” scenario

- In the event of a network partition, the system has two choices
  - Show each user a different view of the data, or
  - Shut down one of the partitions

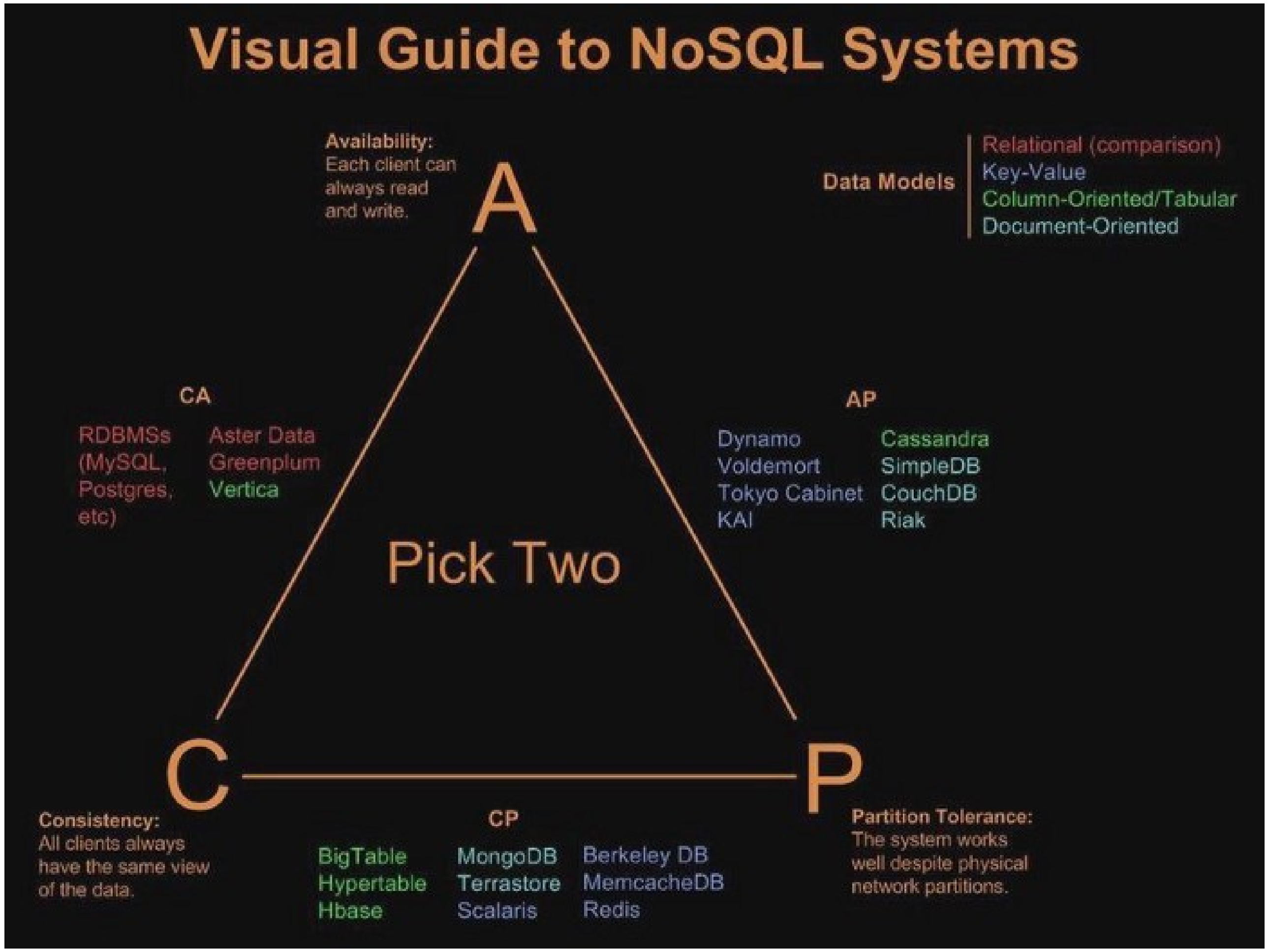


Eventual consistency

- When outage is critical, the data store should relax consistency—within limits—if necessary in order to ensure availability
  - Favor availability over consistency, but in a predictable, controllable, and manageable way


Strict consistency
Eventual consistency
Consistency
All clients see the same view of data, even right after update or delete
CA
CP
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

An approximate CAP landscape



CRITICISM: https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html

Data models

- (Object-)Relational
  - Row-based or columnar storage
- Key-value
- Column families
- Documents
- Graphs
  - Not included in this material

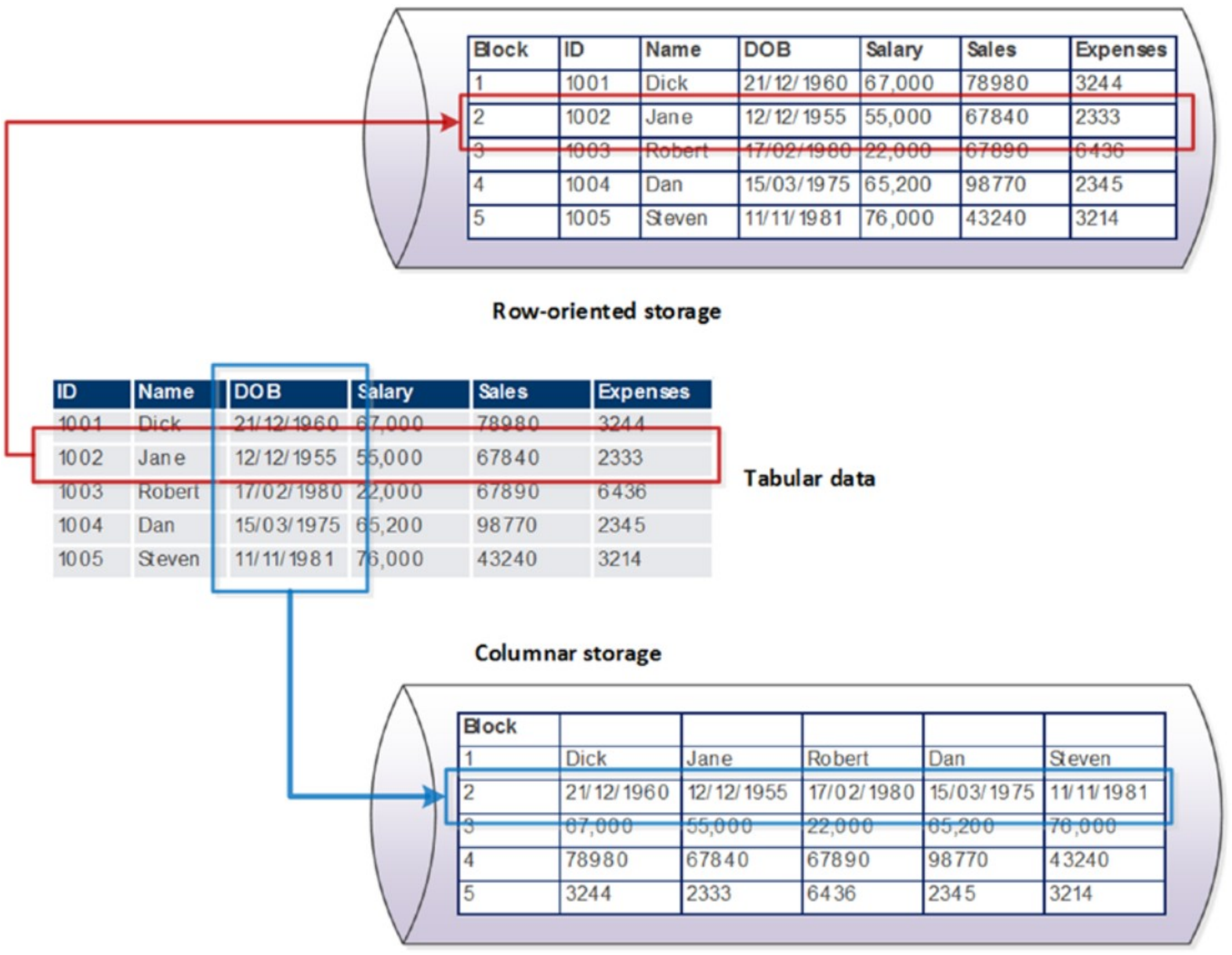
Columnar databases

Columnar databases

- There are many databases following the columnar storage
  - Sybase IQ
  - Vertica
  - MonetDB
  - Amazon Redshift
  - Snowflake
  - Apache Kudu
  - MariaDB ColumnStore
  - ...

Columnar storage

- Columnar storage applies to various data models, including the relational data model
- Data are stored in chunks grouping attribute values

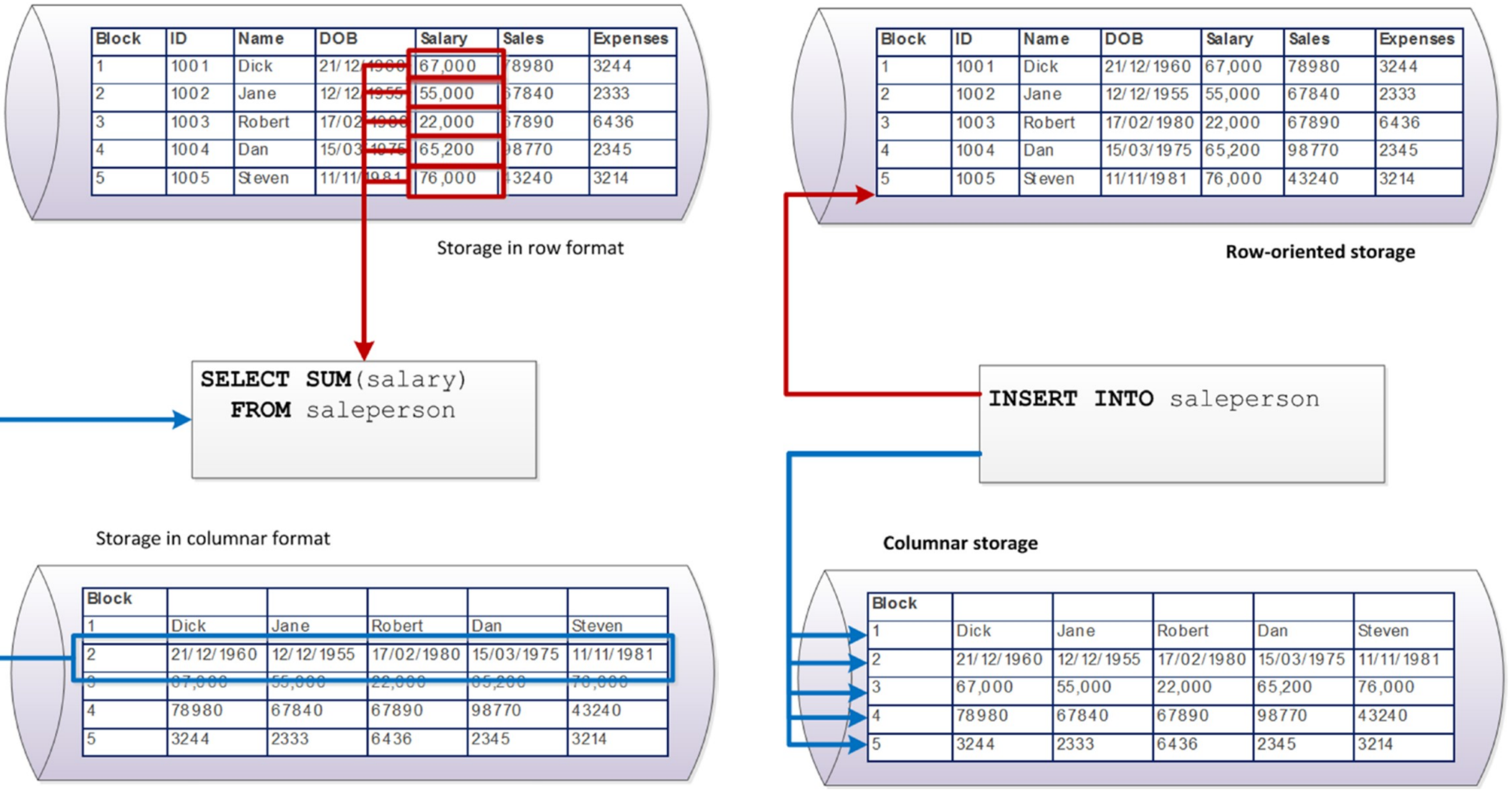


Block	ID	Name	DOB	Salary	Sales	Expenses
1	1001	Dick	21/12/1960	67,000	78980	3244
2	1002	Jane	12/12/1955	55,000	67840	2333
3	1003	Robert	17/02/1980	22,000	67890	6436
4	1004	Dan	15/03/1975	65,200	98770	2345
5	1005	Steven	11/11/1981	76,000	43240	3214

ID	Name	DOB	Salary	Sales	Expenses
1001	Dick	21/12/1960	67,000	78980	3244
1002	Jane	12/12/1955	55,000	67840	2333
1003	Robert	17/02/1980	22,000	67890	6436
1004	Dan	15/03/1975	65,200	98770	2345
1005	Steven	11/11/1981	76,000	43240	3214

Block					
1	Dick	Jane	Robert	Dan	Steven
2	21/12/1960	12/12/1955	17/02/1980	15/03/1975	11/11/1981
3	67,000	55,000	22,000	65,200	76,000
4	78980	67840	67890	98770	43240
5	3244	2333	6436	2345	3214

Columnar storage: pros and cons



Block	ID	Name	DOB	Salary	Sales	Expenses
1	1001	Dick	21/12/1960	67,000	78980	3244
2	1002	Jane	12/12/1955	55,000	67840	2333
3	1003	Robert	17/02/1980	22,000	67890	6436
4	1004	Dan	15/03/1975	65,200	98770	2345
5	1005	Steven	11/11/1981	76,000	43240	3214

Storage in row format

```
SELECT SUM(salary)
  FROM saleperson
```

Storage in columnar format

Block					
1	Dick	Jane	Robert	Dan	Steven
2	21/12/1960	12/12/1955	17/02/1980	15/03/1975	11/11/1981
3	67,000	55,000	22,000	65,200	76,000
4	78980	67840	67890	98770	43240
5	3244	2333	6436	2345	3214

Block	ID	Name	DOB	Salary	Sales	Expenses
1	1001	Dick	21/12/1960	67,000	78980	3244
2	1002	Jane	12/12/1955	55,000	67840	2333
3	1003	Robert	17/02/1980	22,000	67890	6436
4	1004	Dan	15/03/1975	65,200	98770	2345
5	1005	Steven	11/11/1981	76,000	43240	3214

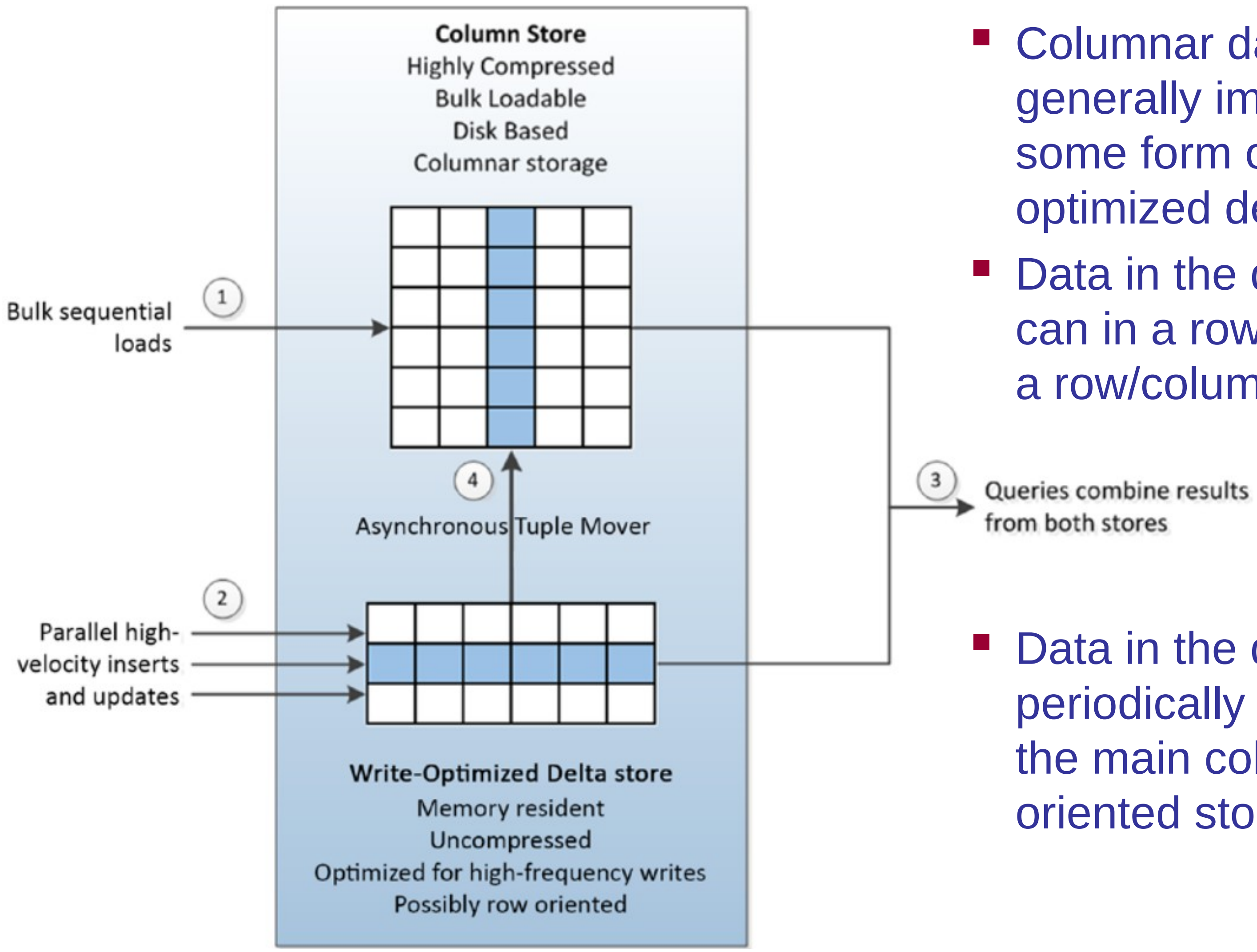
Row-oriented storage

```
INSERT INTO saleperson
```

Columnar storage

Block					
1	Dick	Jane	Robert	Dan	Steven
2	21/12/1960	12/12/1955	17/02/1980	15/03/1975	11/11/1981
3	67,000	55,000	22,000	65,200	76,000
4	78980	67840	67890	98770	43240
5	3244	2333	6436	2345	3214

Write-optimized delta storage



- Columnar databases generally implement some form of write-optimized delta store
- Data in the delta store can in a row, columnar or a row/column hybrid
- Data in the delta store is periodically merged with the main columnar-oriented store

Key-value databases (Redis)

Key-value stores

- Unlike the relational model, there is no formal definition for how data is represented in a key-value store
- Typically a key-value store will accept any binary value within specific size limits as a key and is capable of storing any binary data as a value
  - In this respect, we might say that the key-value store is data-type agnostic
  - However, most key-value stores provide additional support for certain types of data

Redis

- Redis (Remote Dictionary Server) is an open source (BSD licensed) in-memory key-value data structure store
- Redis was originally envisaged as a simple in-memory system capable of sustaining very high transaction rates on underpowered systems, such as virtual machine images on commodity hardware
- In Redis, objects consist mainly of strings and various types of collections of strings (lists, sorted lists, hash maps, etc.)
- Primary key lookups are supported
  - It is also possible to programmaticaly create secondary indexes using sorted sets or lists

Main features

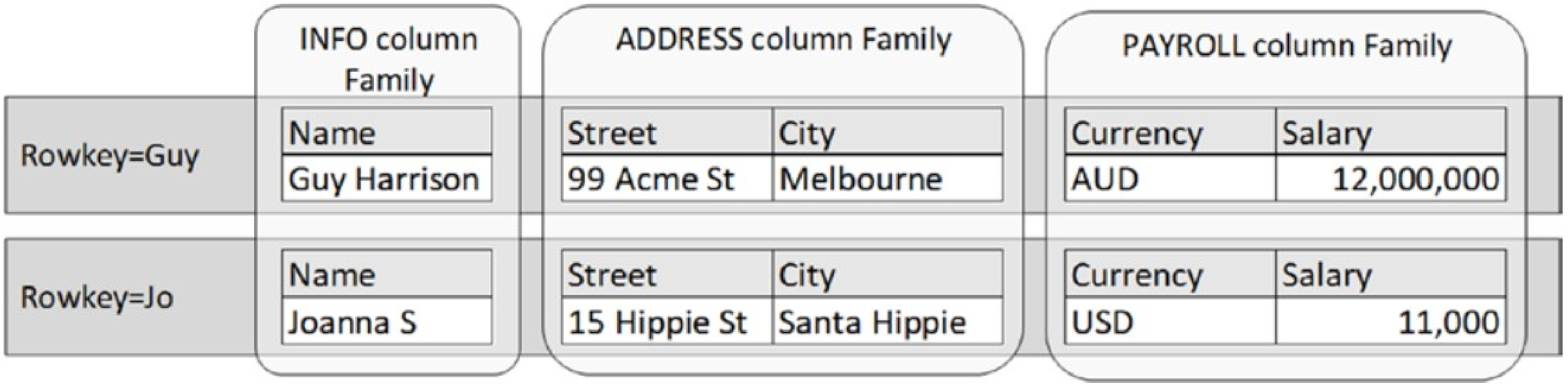
- Built-in replication
- Transactions
  - You can run atomic operations on data structures, like
    - Appending to a string
    - Incrementing the value in a hash
    - Pushing an element to a list
    - Computing set intersection, union and difference
    - Getting the member with highest ranking in a sorted set
- Different levels of on-disk persistence
  - periodically dumping the dataset to disk; or
  - appending each command to a disk-based log
- High availability via Redis Sentinel
- Automatic partitioning with Redis Cluster

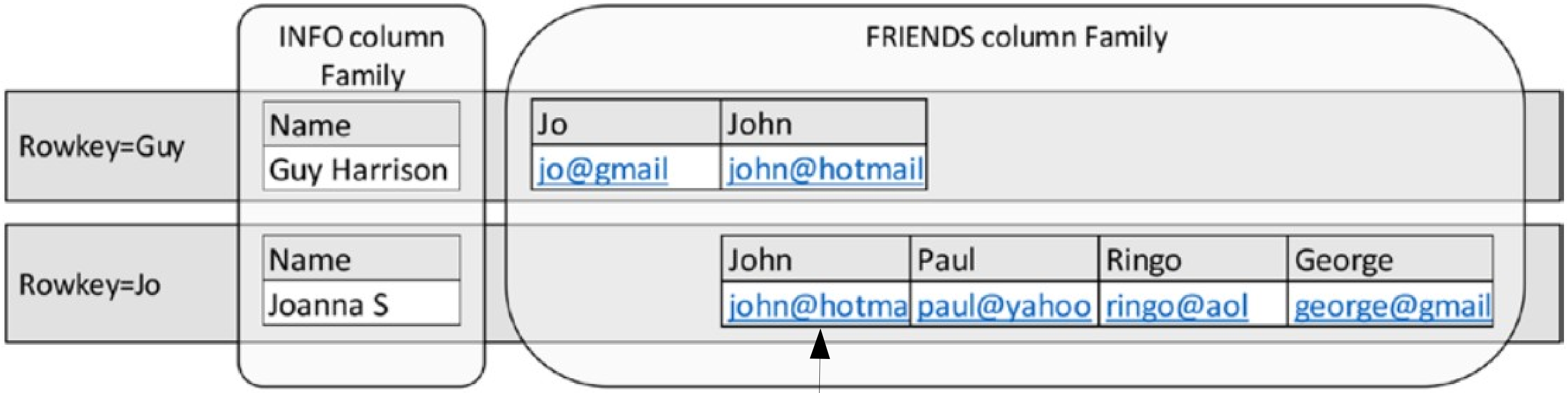
Column family databases (Cassandra)

Column family structure

- Column families were introduced in Google BigTable
- Column families can be used to group related columns for convenience
  - To optimize disk I/O by co-locating columns that are frequently accessed together on disk, or
  - To create a multidimensional structure that can be used for more complex data

Simple and wide column families





This column is shared between the two rows

Cassandra

- Apache Cassandra is a distributed NoSQL database that delivers always-on availability, fast read-write performance, and linear scalability
- Cassandra’s development started at Facebook, was released as an open-source project in 2008, and became a top-level project for the Apache Foundation
- It is primarily based on Amazon’s Dynamo and Google BigTable
  - Cassandra does not have master nodes, it allocates a coordinator for each operation
    - No single point of failure
    - Use of the gossip peer-to-peer protocol

Key characteristics

- Consistent hashing
  - Consistent hashing is a scheme that uses the hash value of the primary key to determine the nodes in the cluster responsible for that key and that allows nodes to be added or removed from the cluster with minimal rebalancing overhead
- Tunable consistency
  - The application can specify trade-offs between consistency, read performance, and write performance
    - It’s possible to specify strong consistency, eventual consistency, or weak consistency
- Data versioning
  - Since write operations will never be blocked, it is possible that there will be multiple versions of an object in the system

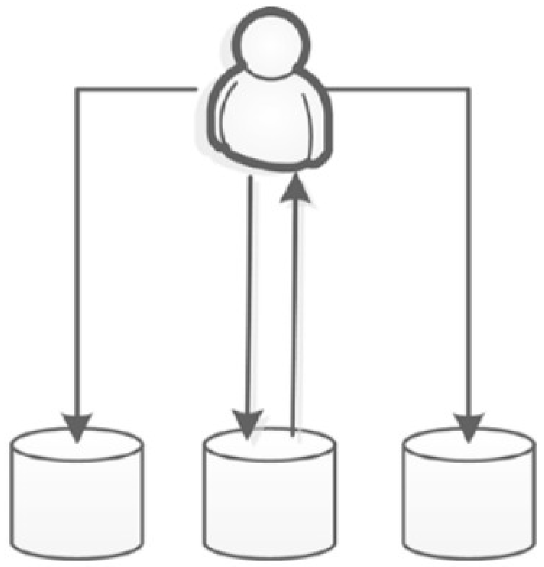
Tunable consistency

- The NWR notation describes how the system will trade off consistency, read performance, and write performance
  - N is the number of copies of each data item that the database will maintain
  - W is the number of copies of the data item that must be written before the write can complete
  - R is the number of copies that the application will access when reading the data

Tunable consistency configurations

- When W = N, the system will always write every copy before returning control to the application
  - This is what ACID databases do when implementing synchronous replication
- The most common configuration is N > W > 1
  - More than one write must complete, but not all nodes need to be updated immediately
- Another common setting is W + R > N
  - This ensures that the latest value will always be included in a read operation, even if it is mixed in with “older” values
  - This is sometimes referred to as quorum assembly

Tunable consistency configurations(2)



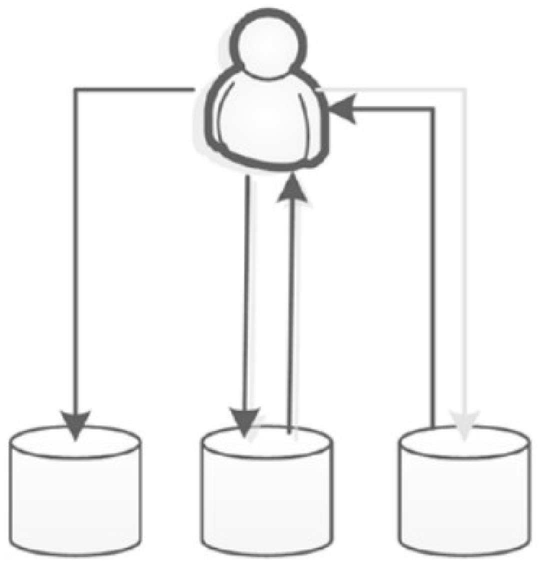
N=3 W=3 R=1

Slow writes, fast reads, consistent

There will be 3 copies of the data.

A write request only returns when all 3 have written to disk.

A read request only needs to read one version.



N=3 W=2 R=2

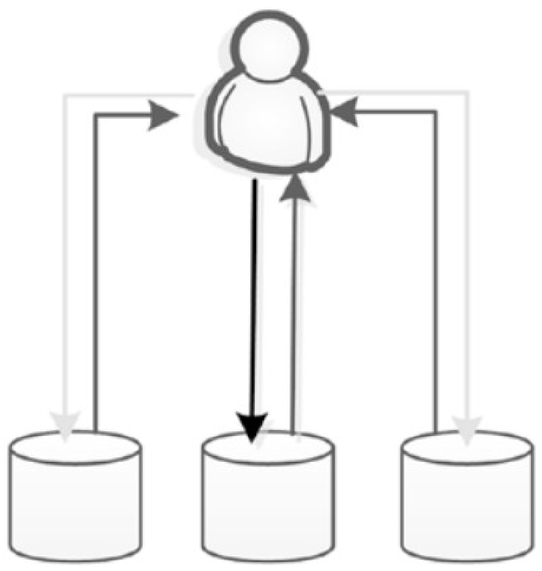
Faster writes, still consistent (quorum assembly)

There will be 3 copies of the data.

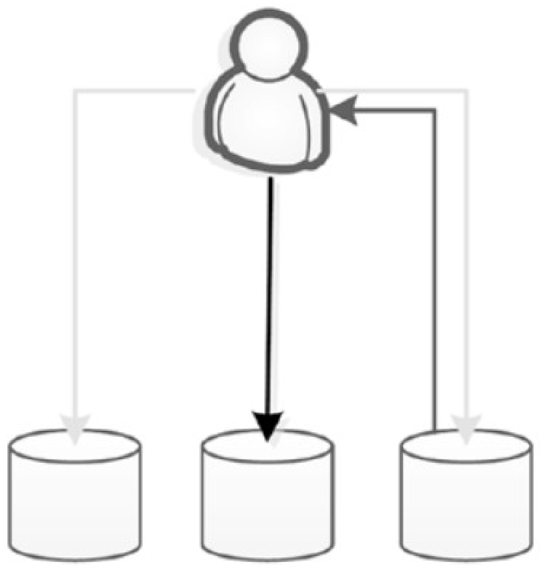
A write request returns when 2 copies are written – the other can happen later.

A read request reads 2 copies make sure it has the latest version.

Tunable consistency configurations(3)



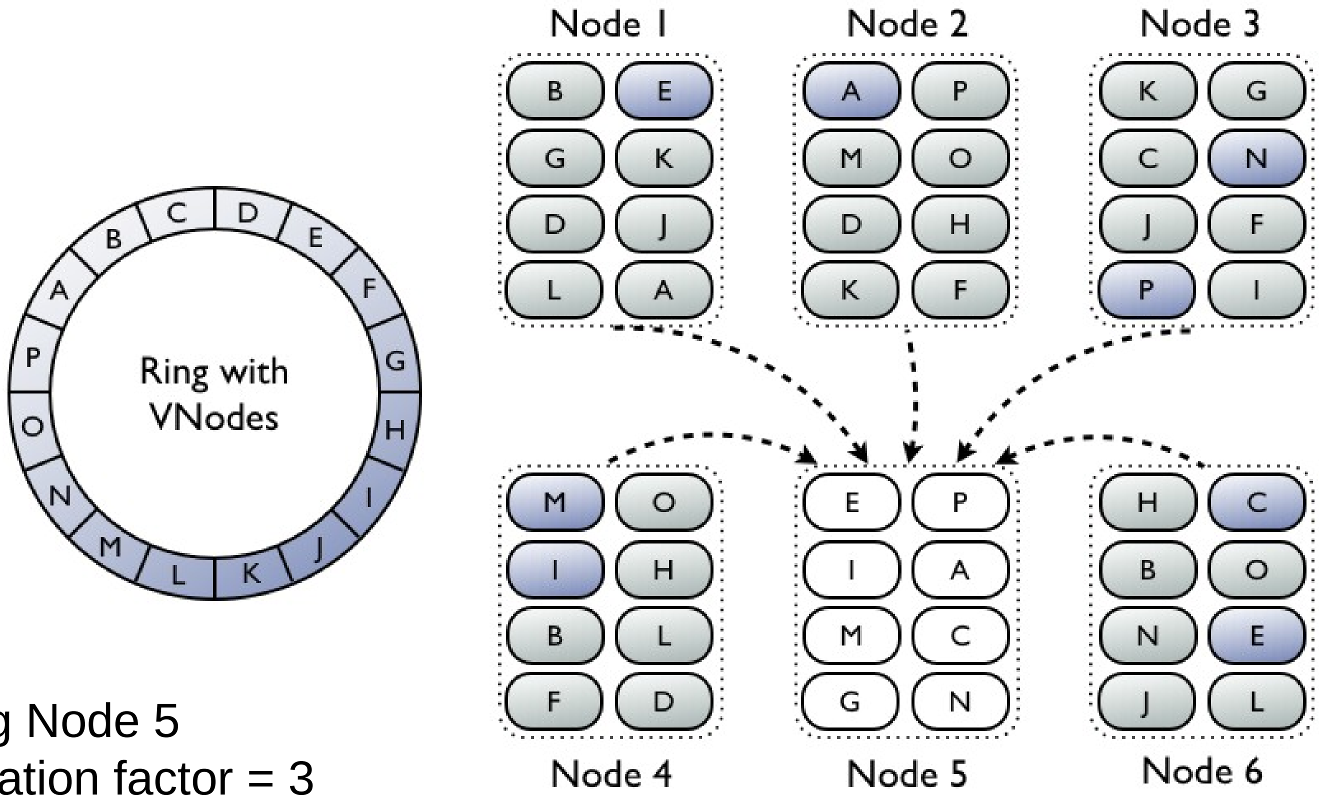
N=3 W=1 R=N
Fastest write, slow but consistent reads
There will be 3 copies of the data.
A write request returns once the first copy is written – the other 2 can happen later.
A read request reads all copies to make sure it gets the latest version.
Data might be lost if a node fails before the second write.



N=3 W=1 R=1
Fast, but not consistent
There will be 3 copies of the data.
A write request returns once the first copy is written – the other 2 can happen later.
A read request reads a single version only: it might not get the latest copy.
Data might be lost if a node fails before the second write.

Virtual nodes

- Cassandra employs virtual nodes to distribute and rebalance data among physical nodes



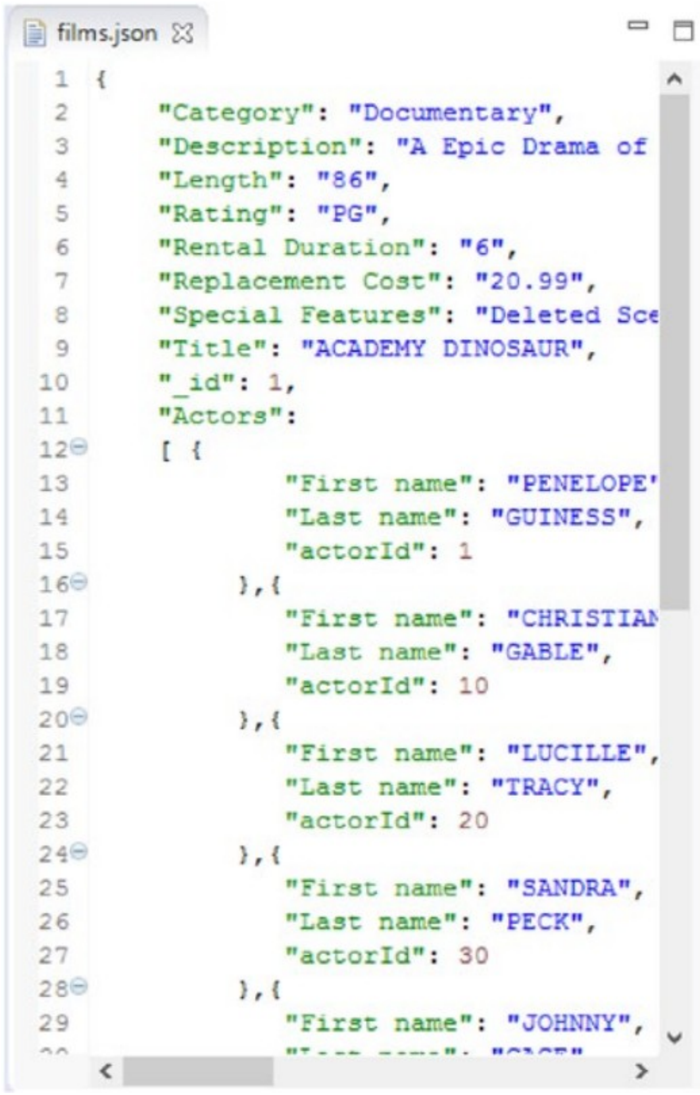
Adding Node 5
Replication factor = 3

Document databases (MongoDB)

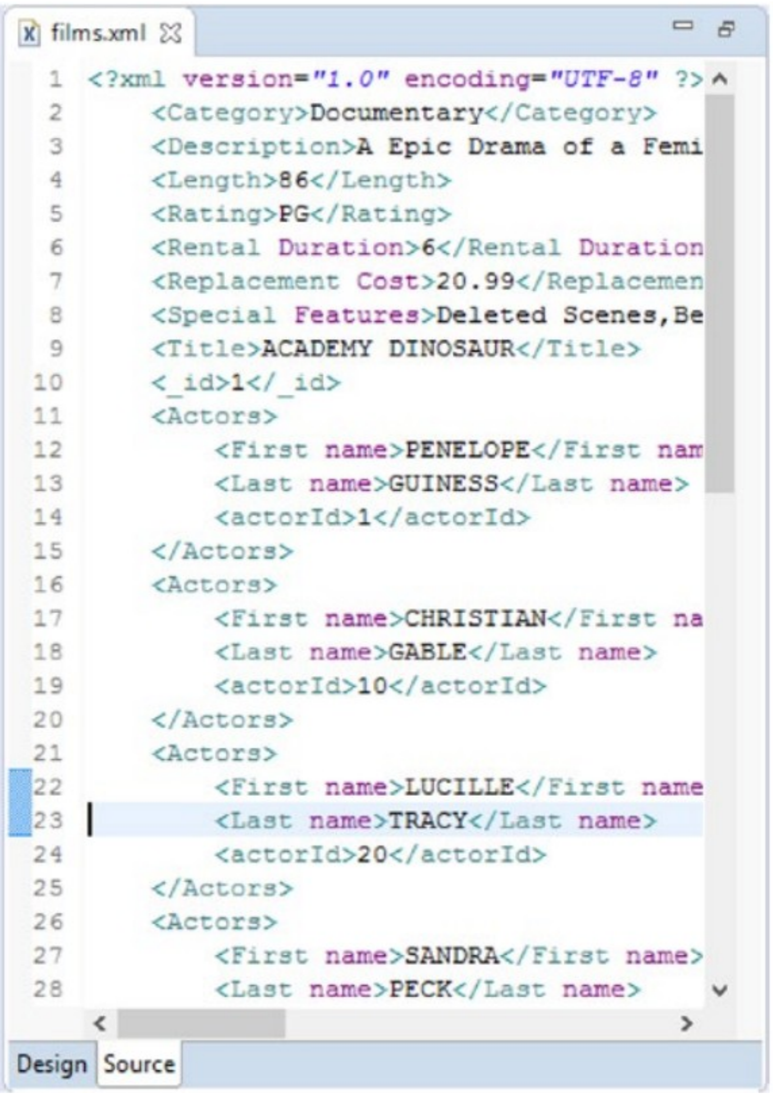
Document databases

- A document database is a nonrelational database that stores data as structured documents
- Allows some form of data description without enforcing a schema
  - A medium between the rigid schema of the relational database and the completely schema-less key-value stores
- Most common data types
  - XML and JSON

JSON vs. XML

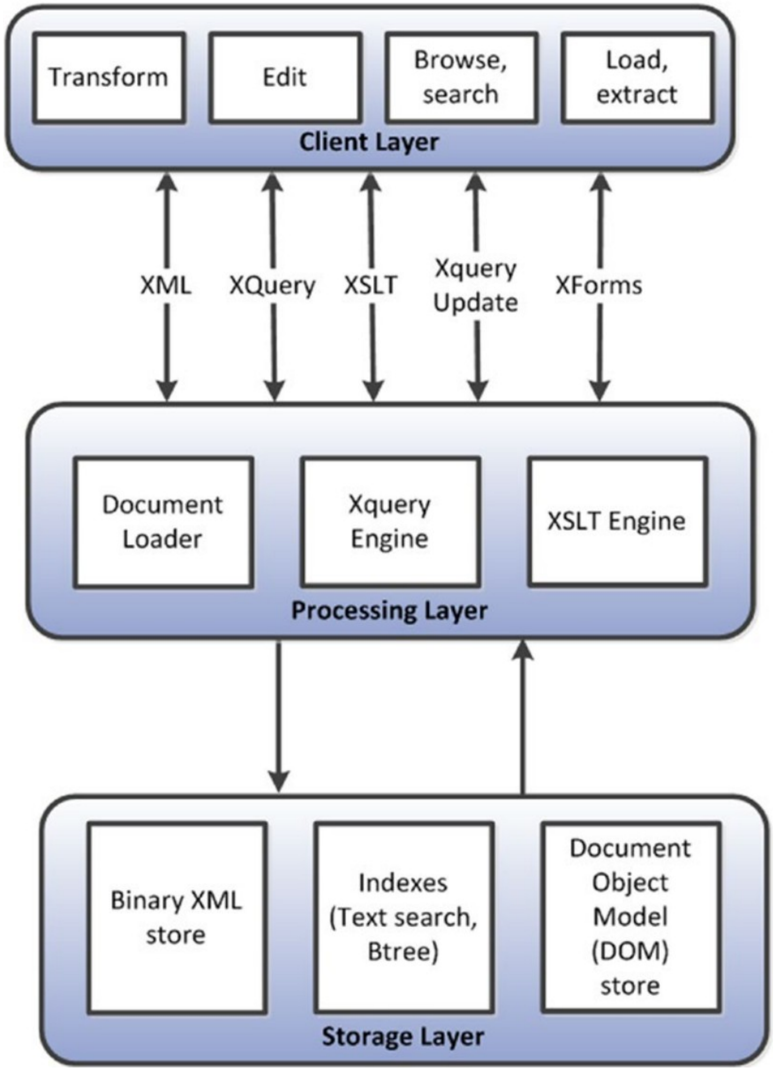


```
{
      "Category": "Documentary",
      "Description": "A Epic Drama of
      "Length": "86",
      "Rating": "PG",
      "Rental Duration": "6",
      "Replacement Cost": "20.99",
      "Special Features": "Deleted Sce
      "Title": "ACADEMY DINOSAUR",
      "_id": 1,
      "Actors":
      [ {
              "First name": "PENELOPE'
              "Last name": "GUINESS",
              "actorId": 1
          },{
              "First name": "CHRISTIAN
              "Last name": "GABLE",
              "actorId": 10
          },{
              "First name": "LUCILLE",
              "Last name": "TRACY",
              "actorId": 20
          },{
              "First name": "SANDRA",
              "Last name": "PECK",
              "actorId": 30
          },{
              "First name": "JOHNNY",
```



```
<?xml version="1.0" encoding="UTF-8" ?>
    <Category>Documentary</Category>
    <Description>A Epic Drama of a Femi
    <Length>86</Length>
    <Rating>PG</Rating>
    <Rental Duration>6</Rental Duration
    <Replacement Cost>20.99</Replacemen
    <Special Features>Deleted Scenes,Be
    <Title>ACADEMY DINOSAUR</Title>
    <_id>1</_id>
    <Actors>
        <First name>PENELOPE</First nam
        <Last name>GUINESS</Last name>
        <actorId>1</actorId>
    </Actors>
    <Actors>
        <First name>CHRISTIAN</First na
        <Last name>GABLE</Last name>
        <actorId>10</actorId>
    </Actors>
    <Actors>
        <First name>LUCILLE</First name
        <Last name>TRACY</Last name>
        <actorId>20</actorId>
    </Actors>
    <Actors>
        <First name>SANDRA</First name>
        <Last name>PECK</Last name>
```

XML databases



JSON databases: MongoDB

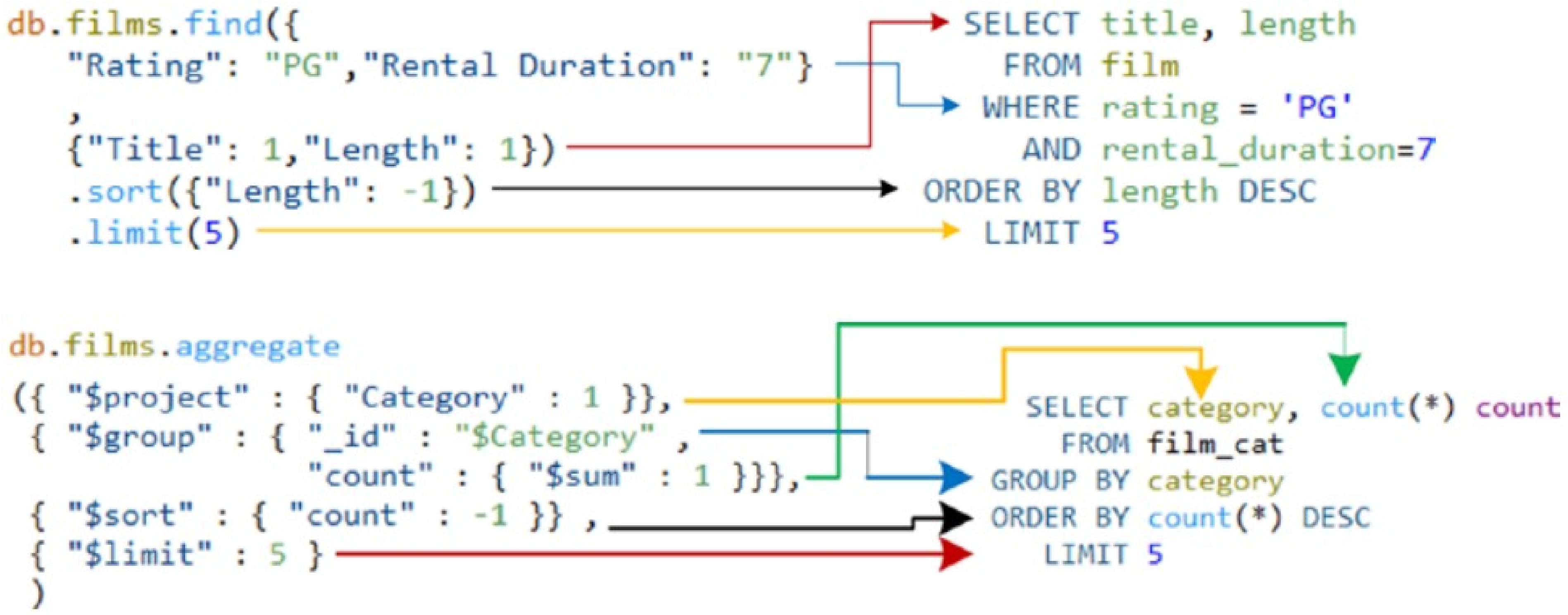
- MongoDB is a JSON-oriented document database
  - It uses a binary encoded variant of JSON called BSON
  - The BSON format supports lower parse overhead than JSON, as well as richer support for data types such as dates and binary data
- MongoDB provides a JavaScript-based query capability that is reasonably easy to learn
- MongoDB established a strong lead in the NoSQL database space by providing a developer-friendly ecosystem and architecture
  - MongoDB achieved a position withing the modern web-development community similary to the LAMP (Linux-Apache-MySQL-PHP) stack applications in the early 2000s

Query language

- MongoDB provides a rich query language implemented in JavaScript, and exposed through the MongoDB shell
- Collection objects within MongoDB include methods that allows fairly complex queries to be constructed
  - find(): for general queries
  - aggregate(): to aggregate accross documents

Query language(2)

- Comparison with SQL



```
db.films.find({
    "Rating": "PG","Rental Duration": "7"}
   ,
   {"Title": 1,"Length": 1})
   .sort({"Length": -1})
   .limit(5)
```

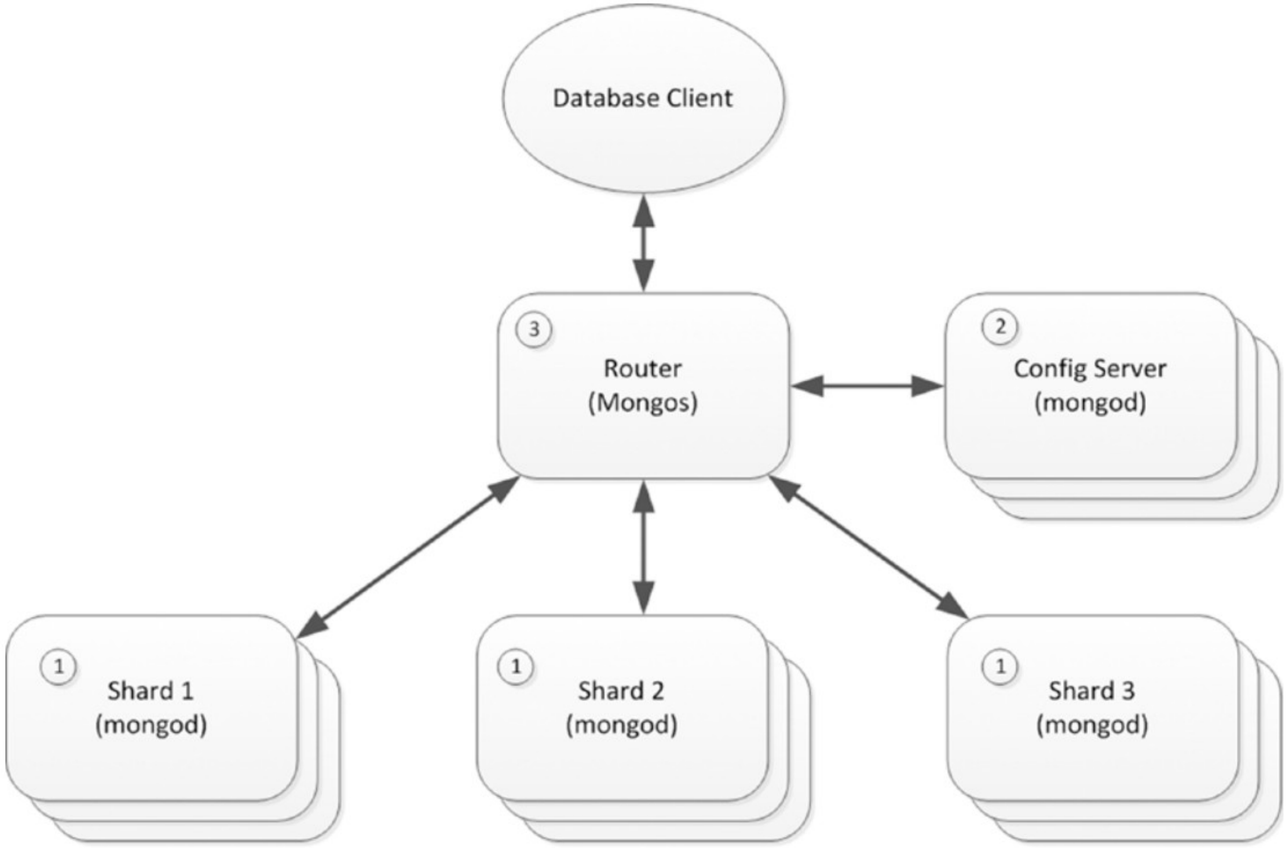
```
SELECT title, length
  FROM film
 WHERE rating = 'PG'
   AND rental_duration=7
ORDER BY length DESC
 LIMIT 5
```

```
db.films.aggregate
({ "$project" : { "Category" : 1 }},
 { "$group" : { "_id" : "$Category" ,
               "count" : { "$sum" : 1 }}},
 { "$sort" : { "count" : -1 }} ,
 { "$limit" : 5 }
 )
```

```
 SELECT category, count(*) count
   FROM film_cat
GROUP BY category
ORDER BY count(*) DESC
   LIMIT 5
```

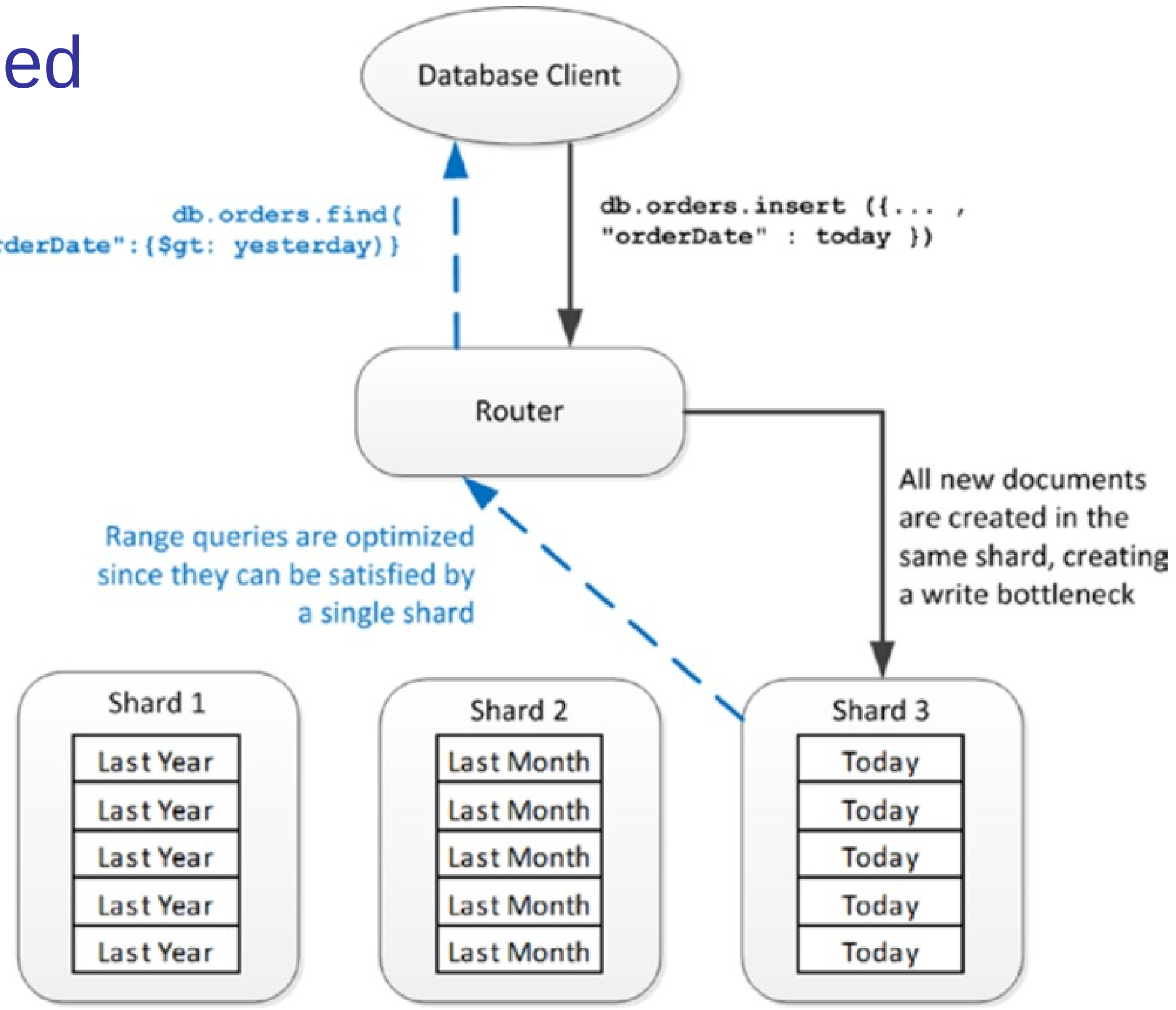
MongoDB architecture

- MongoDB supports sharding to provide scale-out capabilities and replication for high availability



Sharding mechanisms

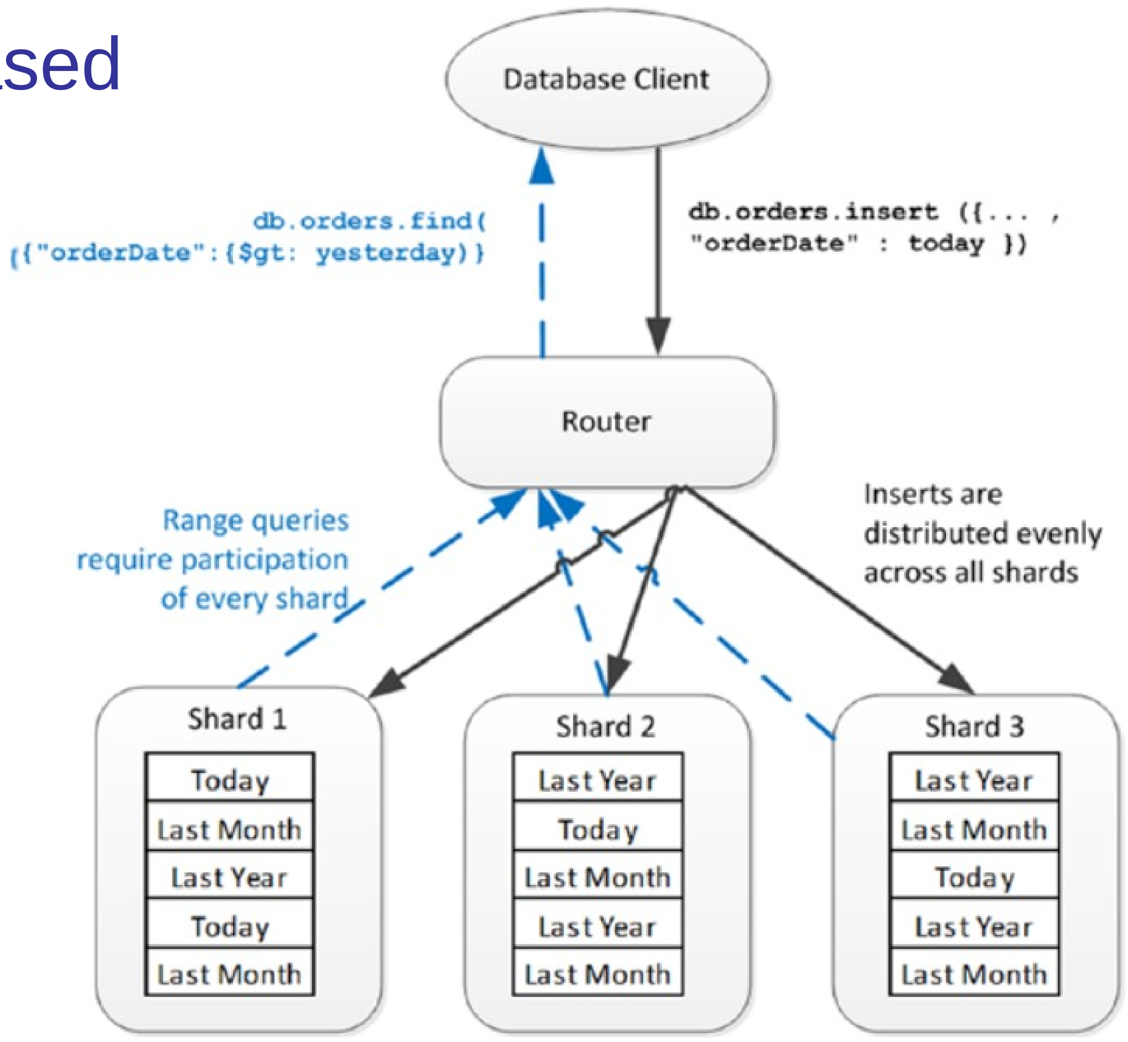
- Range-based



Database Client
db.orders.find({"orderDate":{$gt: yesterday)}
db.orders.insert ({... , "orderDate" : today })
Router
All new documents are created in the same shard, creating a write bottleneck
Range queries are optimized since they can be satisfied by a single shard
Shard 1
Last Year
Last Year
Last Year
Last Year
Last Year
Shard 2
Last Month
Last Month
Last Month
Last Month
Last Month
Shard 3
Today
Today
Today
Today
Today

Sharding mechanisms(2)

- Hash-based


Database Client
db.orders.find({"orderDate":{$gt: yesterday}}
db.orders.insert ({... , "orderDate" : today })
Router
Range queries require participation of every shard
Inserts are distributed evenly across all shards
Shard 1
Today
Last Month
Last Year
Today
Last Month
Shard 2
Last Year
Today
Last Month
Last Year
Last Month
Shard 3
Last Year
Last Month
Today
Last Year
Last Month

Cluster balancing

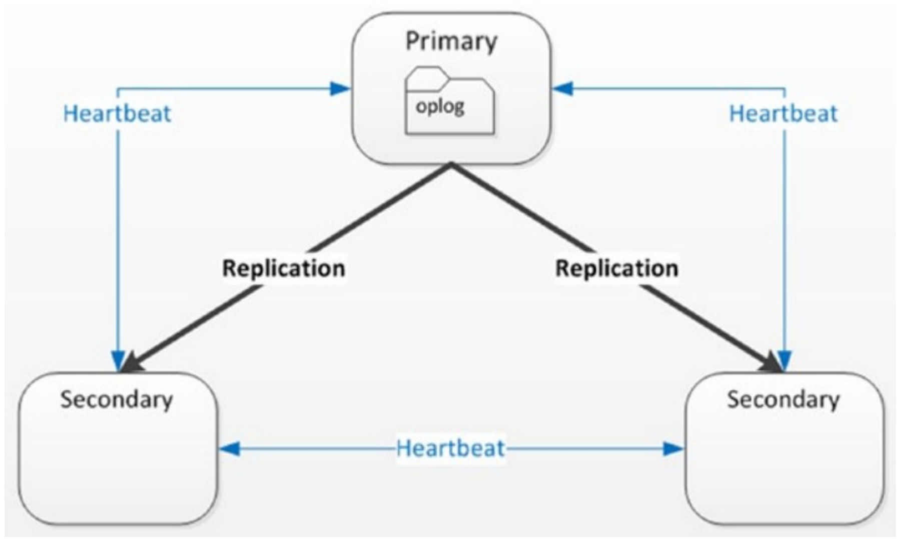
- MongoDB will periodically assess the balance of shards across the cluster and perform rebalance operations, if needed
- The unit of rebalance is the shard chunk
  - Shard chunks—typically 64MB in size—contain contiguous values of shard keys
  - The chunks split if they grow too large
- The cluster balancing process move chunks from one shard to another

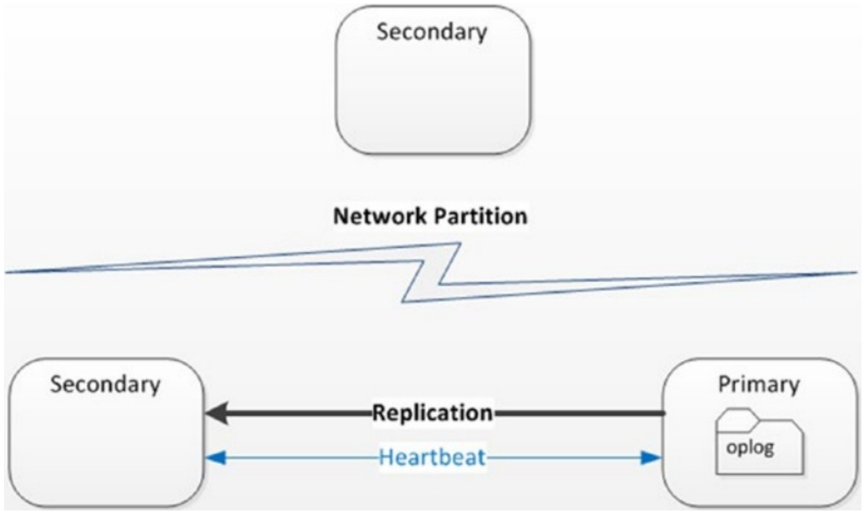
Replication

- In MongoDB, data can be replicated across machines by the means of replica sets
  - A replica set consists of a primary node together with two or more secondary nodes
  - The primary node accepts all write requests, which are propagated asynchronously to the secondary nodes
- The primary node is determined by an election involving all available nodes
  - To be eligible to become primary, a node must be able to contact more than half of the replica set
  - This ensures that if a network partitions a replica set in two, only one of the partitions will attempt to establish a primary

Failover

- Members within a replica set communicate frequently via heartbeat messages
  - The primary stores information about document changes in a oplog and continuously attempt to apply these changes to secondary instances
- If a primary finds it is unable to receive heartbeat messages from more than half of the secondaries, then it will renounce its primary status and a new election will be called

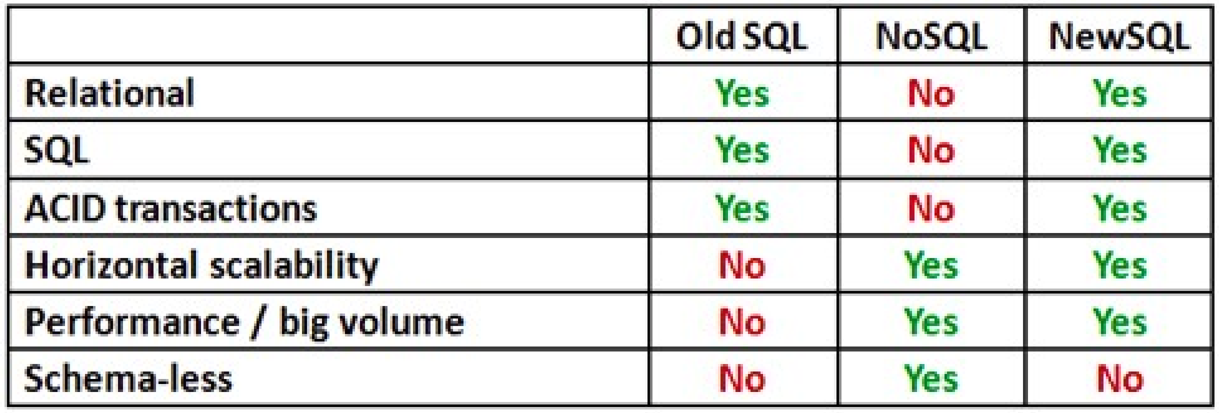




New SQL products

New SQL

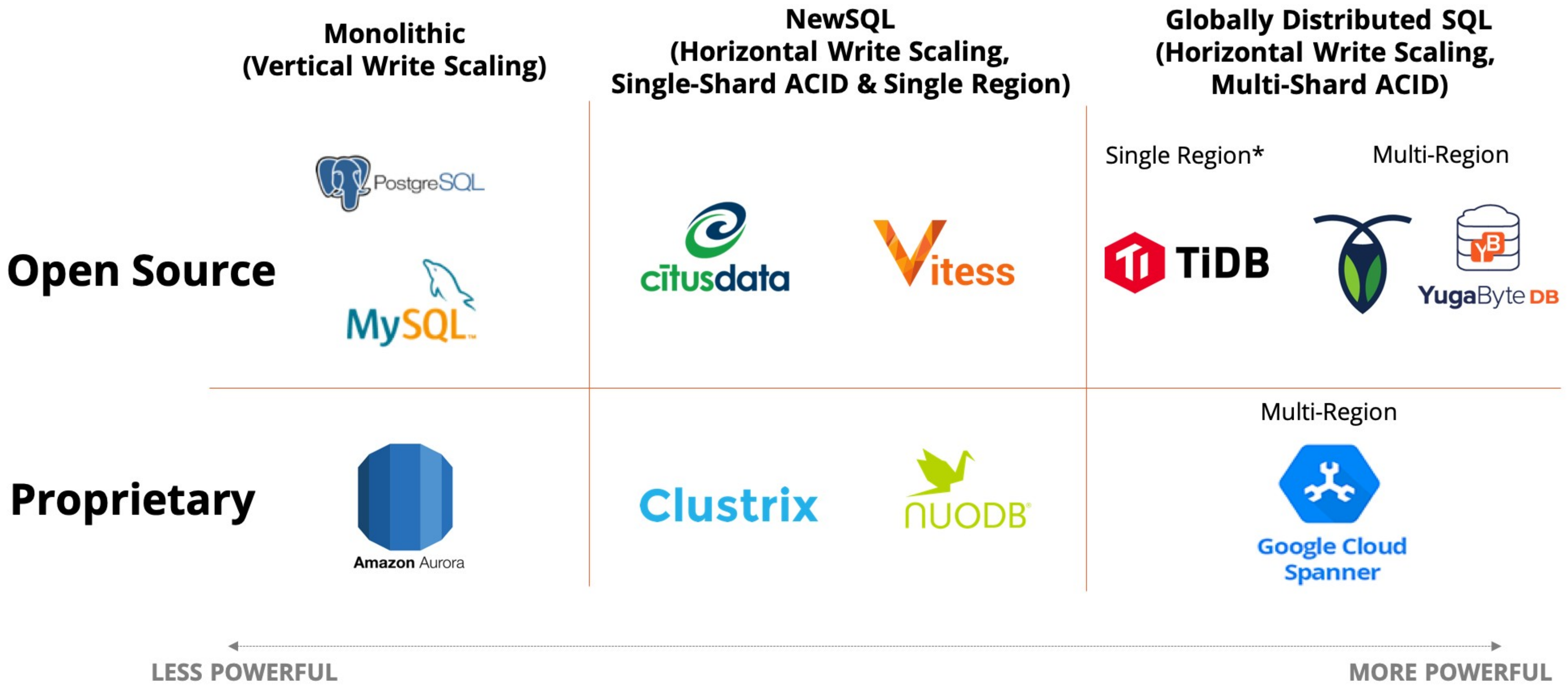
- The primary goal is adding horizontal write scaling to relational databases



	Old SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL	Yes	No	Yes
ACID transactions	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Performance / big volume	No	Yes	Yes
Schema-less	No	Yes	No

Rough comparison between SQL, NoSQL and NewSQL

Distributed DMBS market



- NewSQL: ClustrixDB, NuoDB, CockroachDB, Pivotal GemFire XD, Altibase, MemSQL, VoltDB, Apache Trafodion, ...