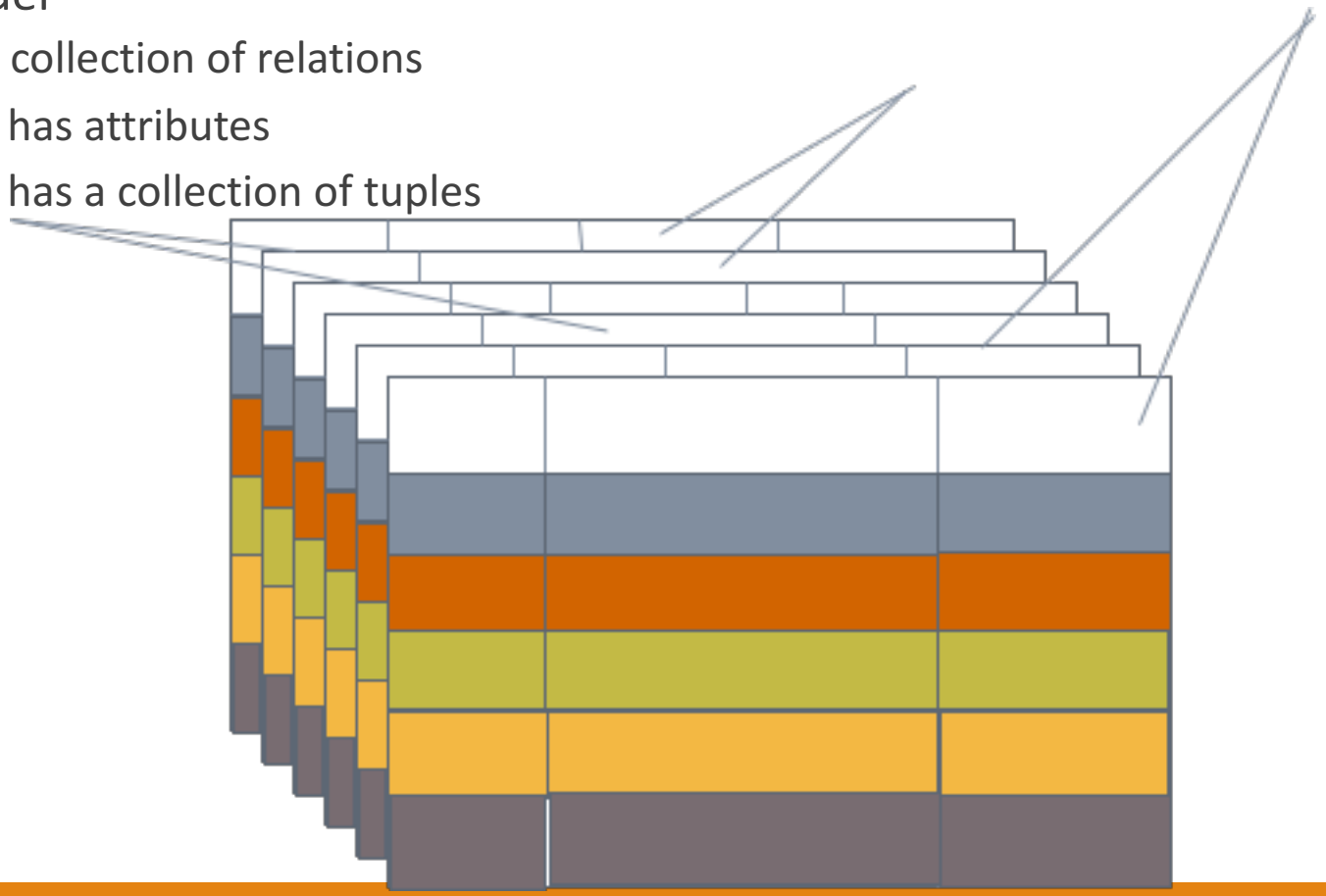
Relational Database

Relational Model

- Database is a collection of relations
- Each relation has attributes
- Each relation has a collection of tuples



Pros : Relational Database

Concurrency Control

- ACID (Atomic, Consistent, Isolated and Durable) Transaction Management
  - Atomicity :
    - An operation either succeeds or fails entirely.
    - Many rows spanning many tables are updated as a single operation.
  - Consistency:
    - Any given transaction must change affected data only in allowed ways.
  - Isolation:
    - Defines how/when the changes made by one operation become visible to other.
    - Concurrent operations are isolated from each other so that they can't see a partial update.
  - Durablility:
    - Once a transaction has been committed, it will remain permanently.

https://en.wikipedia.org/wiki/ACID

Pros : Relational Database

Standard Model

- Different vendors' query languages are similar.
- Transaction operations work in the similar way.

Relational Databases

Join

- Purpose
  - Relate rows from two different tables based on constraints
- Input
  - Two tables
- Output
  - Merged rows from two tables satisfying given constraints

NoSQL

Why NoSQL?

- Impedence Mismatch
  - Relation model ≠ In-memory data structure (object)
  - For application Development Productivity,
    - Better mapping with in-memory data structures for the application.
- Large volumes of data (2000s)
  - Scaling up vs. Scaling out?
    - Run large data on clusters of many smaller and cheaper machines.
    - Cheaper and reliable.
- Example of non-relational database.
  - Google BigTable and Amaon Dynamo.

SQL and NoSQL

SQL

Pros
Persistent data storage.
Concurrency.
Standard Model.

Cons
Impedence mismatch.
Hard to scale.
Fixed schema.

Example
MySQL, PostgreSQL,
SQL Server, Oracle

VS

NoSQL

Pros
Mostly open-source.
Schemaless.
Good for non-relational data.
Scalable.
Runs well on distributed systems.

Cons
Installation, toolsets still maturing.

Example
Redis, MongoDB, Cassandra,
OrientDB

Aggregate-oriented NoSQL Databases

Aggregate: Collection of related objects treated as a unit.

- For analyzing data, you might want to place some data together as an aggregate.
- On a cluster, an aggregate is stored together on a node.
- Aggregate-oriented databases use aggregates indexed by key for data lookup.
- A single aggregate is a unit of atomic updates.
- Aggregate-oriented databases don't have ACID transactions that span multiple aggregates.
- RDBMS is aggregate-ignorant.


```
{
  "_links":{
     "self":{
        "href":"/member/109087/cart"
     },
     "http://example.com/rels/payment":{
        "href":"/payment"
     }
  },
  "_embedded":{
     "http://example.com/rels/cart-item":[
        {
           "_links":{
              "self":{
                 "href":"/member/109087/cart/14418796"
              }
           },
           "id":"14418796",
           "quantity":1,
           "expire_time":"2009-09-11T08:00:00-07:00",
           "_embedded":{
              "http://example.com/rels/sku":[
                 {
                    "_links":{
                       "self":{
                          "href":"/skus/654654"
                       },
                       "http://example.com/images/cart/item":"http://example.com/product/6895/thumbnail.jp
                    },
                    "color":"Black",
                    "size":"S",
                    "returnable":false,
                    "price":49
                 }
              ]
           }
        }
     ]
  }
}
```


https://gist.github.com/hjr3/2289546

Aggregate-oriented NoSQL Databases

Pros

- Provides clearer semantics to consider by focusing on the aggregate unit used by applications.
- Better design choice for running on a cluster.

Cons

- Drawing boundaries of an aggregate is not easy.
- When a goal of data management/analysis is not clear, aggregate models might not be the best choice.
- Doesn’t support ACID transactions.

Aggregate-oriented NoSQL Databases

Types

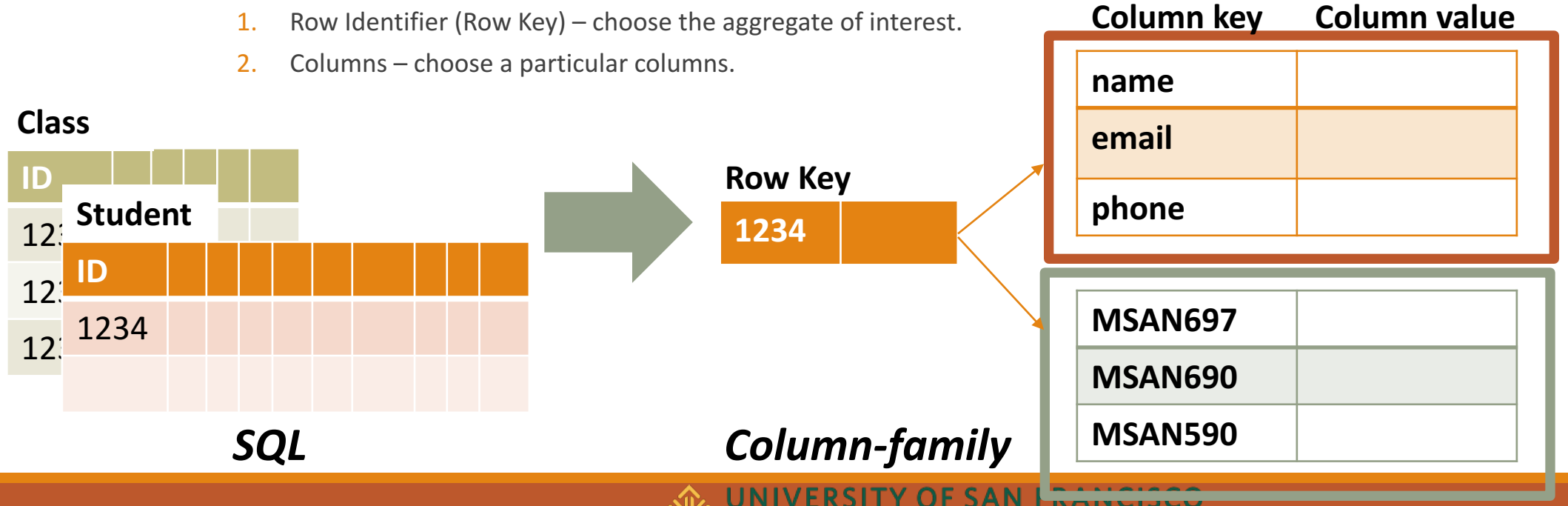
- Key-value and Document Database
  - Each aggregate has a key (ID).
  - Key-value database
    - We can store whatever we want in aggregates.
    - Key lookup for the entire aggregate.
  - Document database
    - It has allowable structures and types.
    - Access by key and also by the fields in the aggregates. (can retrieve fields in the value.)

➔Key-value and Document DB are similar, and their distinction is often blurry.
But with Document DB, you can submit a query based on the internal structure of the document.

Aggregate-oriented NoSQL Databases

Types

- Column Family
  - Optimize for cases when write is rare, but columns are read together in many rows together .
  - Organizes columns into column families (Unit of access).
  - Two level map structure.
    1. Row Identifier (Row Key) – choose the aggregate of interest.
    2. Columns – choose a particular columns.



Relationship-oriented NoSQL Database

Needs of relationship-oriented DB

- Relational database with complex schema
  - Hard to understand, query, generalize and integrate data.
- Aggregate-oriented NoSQL
  - Atomicity is only supported within a single aggregate.

Relationship-oriented NoSQL Database

Graph Database

- Nodes (Object) and edges (Relationships) representation.
- For data with complex relationships.
  - Focuses on graph traverse (more than insert.).
  - Cf) RDBMS : Many joins can cause poor performance.
- Running on a single server rather than distributed across clusters.

NoSQL Database Examples

Key-value

- Redis, Riak, Berkeley DB, etc.

Document

- MongoDB, CouchDB, OrientDB, RavenDB, etc.

Column Family

- Cassandra, Hbase, Amazon SimpleDB, etc.

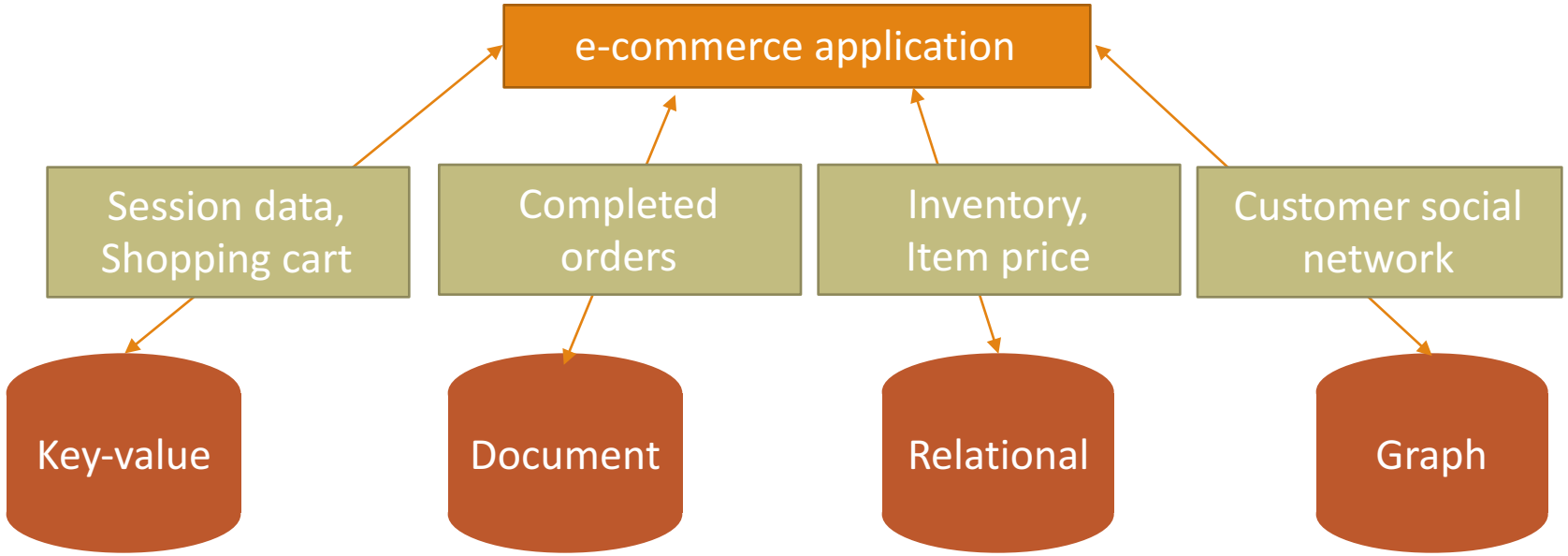
Graph

- OrientDB, Neo4J, FlockDB, etc.

Choice of DBMS

Polygot Persistence

- Using multiple data storage technologies, chosen based on the way data is being used by individual applications.
- NoSQL data stores do not replace relational databases.



SQL Interview Questions

Important RDBMS concepts to remember

- Create
- Insert
- Select
- Update
- Join (Inner, Outer, Left, Right, etc.)*
- Union All/Union/ Union Distinct
- Minus
- Intersect
- Difference WHERE vs. Having
- Normalization (1NF, 2NF and 3NF)*
- Transaction(Concurrency Control) – ACID*
- Indexing* - R tree, Hash
- Truncate vs Delete
- Execution Plan
- Rank, Dense_rank, Rownum

Week 2: NoSQL Overview

DIANE WOODBRIDGE, PH.D

NoSQL Interview Questions

What is NoSQL?

Relational Database vs. NoSQL

Impedence mismatch

Polygot persistence

Aggregate-oriented database

Key-value database

Document database

Column family database

Graph database

Replication vs sharding

CAP Theorem

Eventual Consistency

Distribution Models

Aggregate

- Collection of related objects treated as a unit.
- Natural unit for distribution.

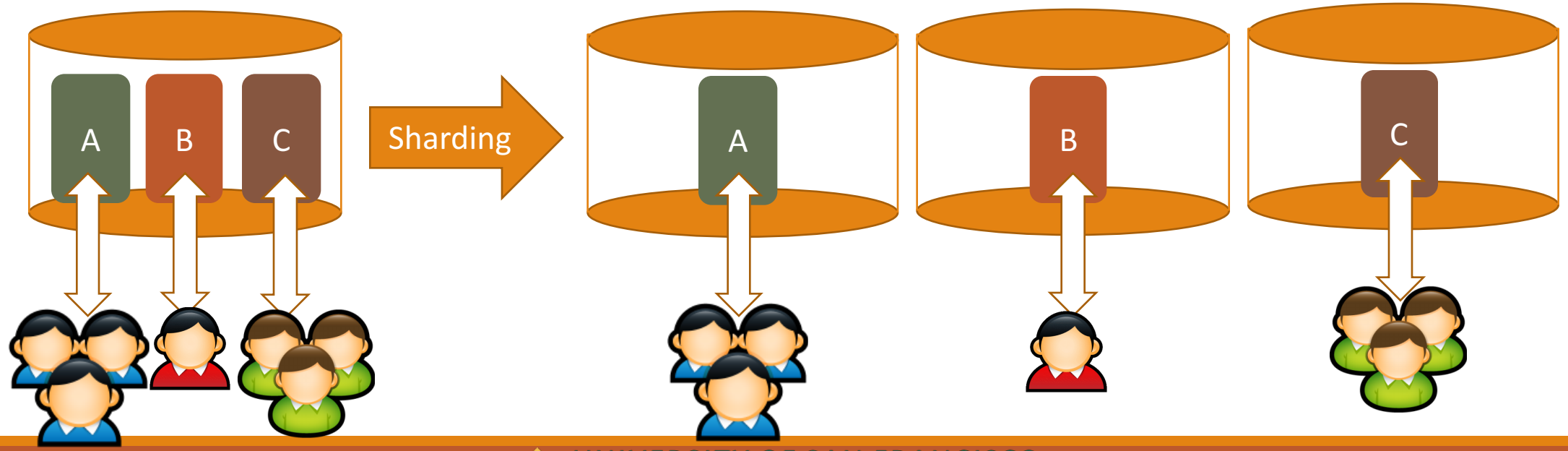
Two ways for data distribution

- Sharding : Place different data on different nodes.
- Replication : Copy the same data over multiple nodes.
  - Master-Slave Replication
  - Peer-to-Peer Replication

Distribution Models

Sharding

- Placing different parts of data into different servers, and each of them does its own reads and writes.
- Things to consider
  - Locate the data commonly accessed together on the same node (Aggregate or Data accessed sequentially together.).
  - Physical location.
  - Keep the load even.



Distribution Models

Sharding

- Options : auto-sharding in the DB vs. writing in your application code.
- Pros : Improves read and writes.
- Cons : Low resilience.

Distribution Models

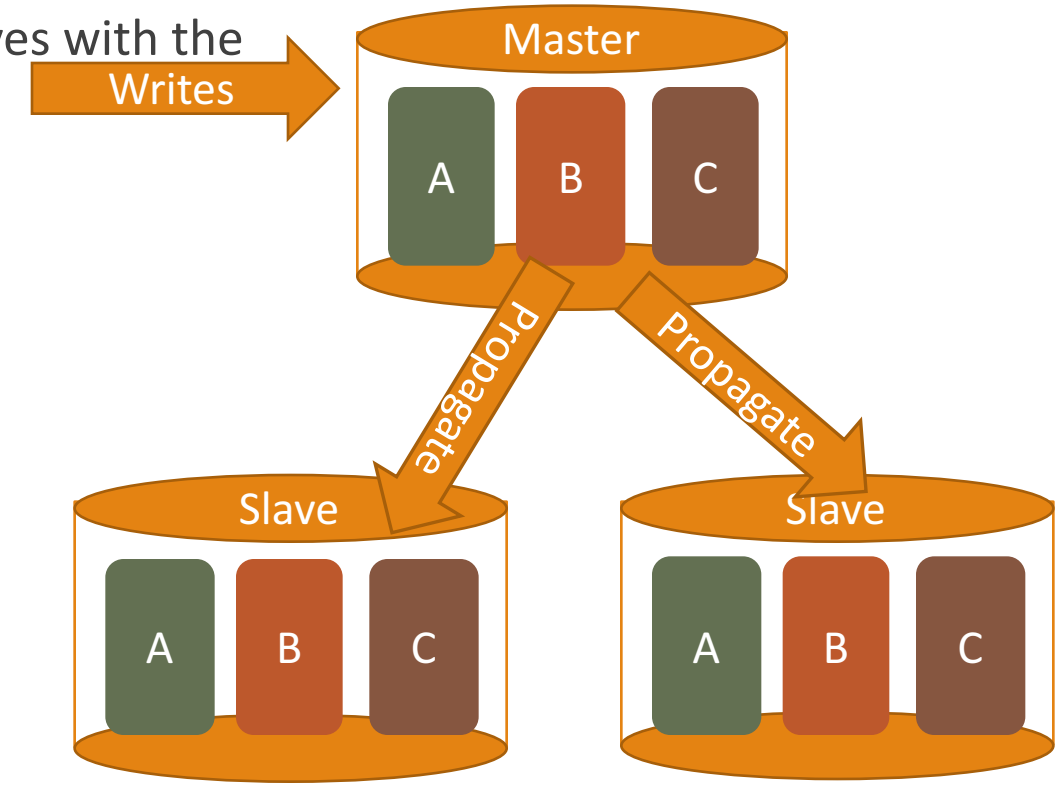
Replications: Copy the same data over multiple nodes.

- Master-slave replication
- Peer-to-peer replication

Distribution Models

Replications

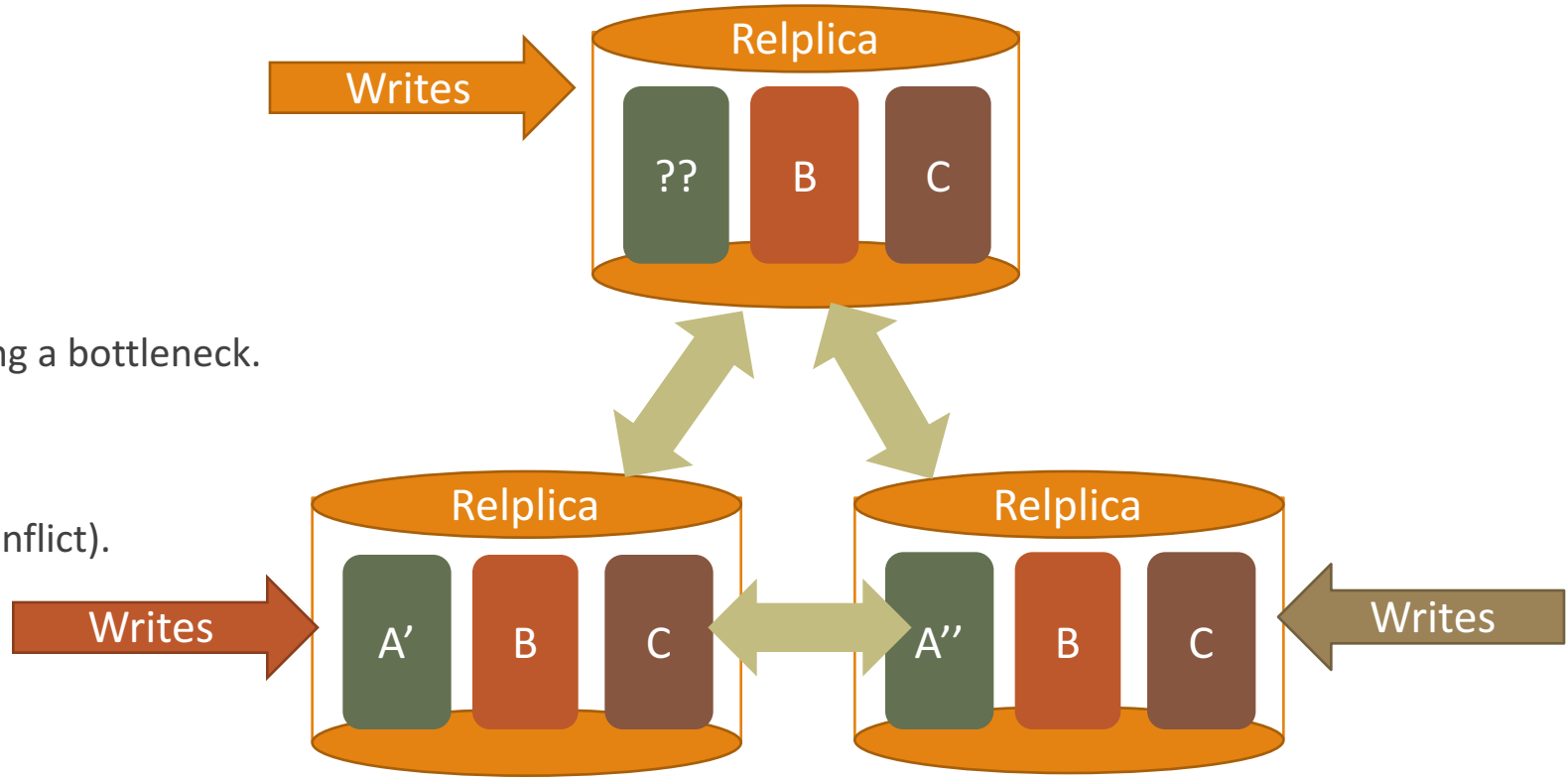
- Master-slave replication : Synchronize slaves with the master.
  - Structure
    - Master(primary)
      - Authorative source for the data.
      - Responsible for processing updates.
      - Manually or automatically assigned.
    - Slaves (secondaries)
  - Pros
    - Good scalability with intensive read.
    - Read resilience.
  - Cons
    - Poor with intensive writes.
    - Inconsistency.



Distribution Models

Replications

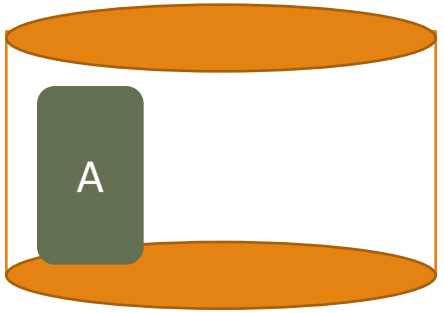
- Peer-to-peer replication
  - All replicas have equal weight.
  - All replicas accept reads/writes.
  - Pros
    - Higher availability.
      - No worries about one node being a bottleneck.
    - Good performance.
  - Cons
    - Inconsistent write. (Write-write conflict).



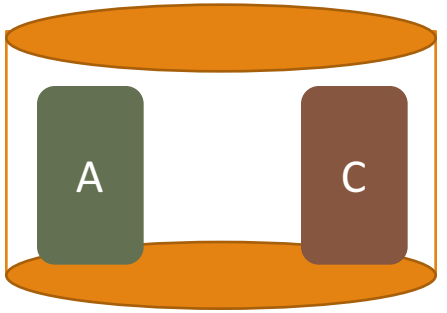
Distribution Models

Sharding + Replications

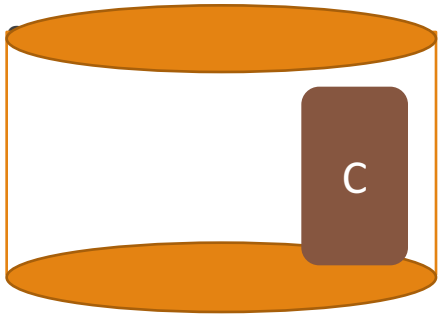
- Master-slave replication and sharding
  - Multiple masters.
  - Each data only has one master.
  - A node can be a master for some data and slave for others.
- Peer-to-peer replication and sharding
  - Common for column-family databases.
  - Many nodes in a cluster with data shared



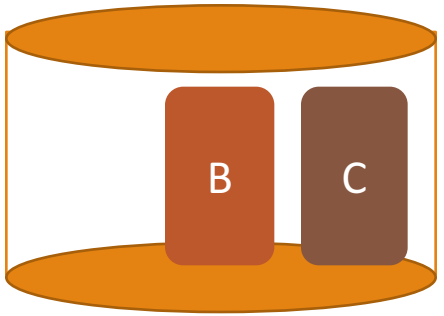
Master for one shard



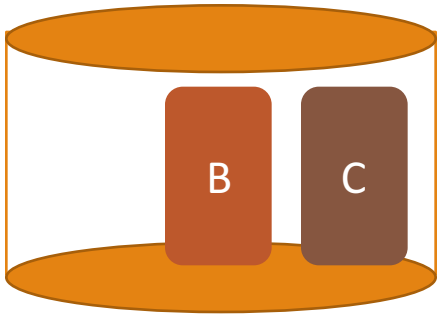
Master for one shard
Slave for one shard



Slave for one shard



Master for one shard
Slave for one shard



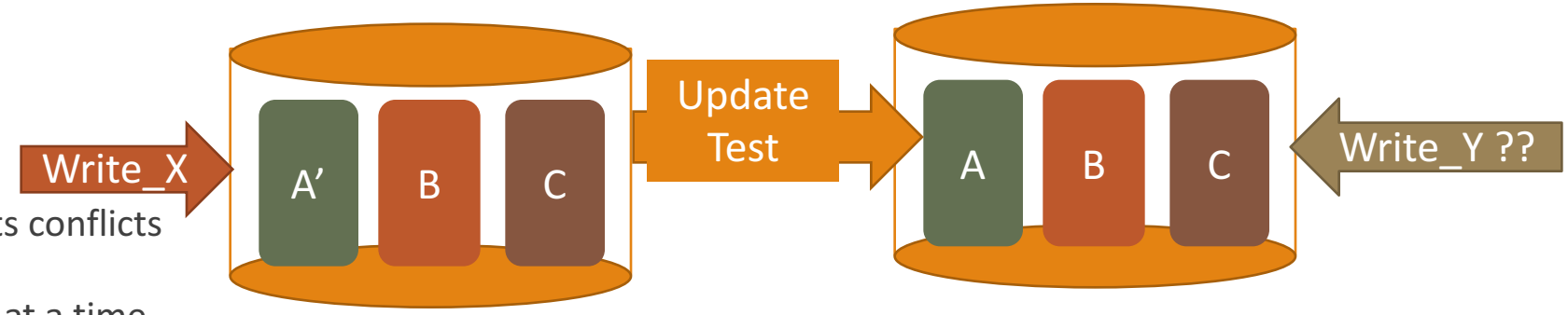
Slave for two shards

Consistency

Update Consistency

- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
    2. One will see the updates, before deciding to apply its update.
  - Optimistic approach – Lets conflicts happen, but takes care of it. (Responsive)
    1. Conditional updates – Checks whether there is an update right before updating data.
    2. Version control – Save both updates and record that they are in conflict.

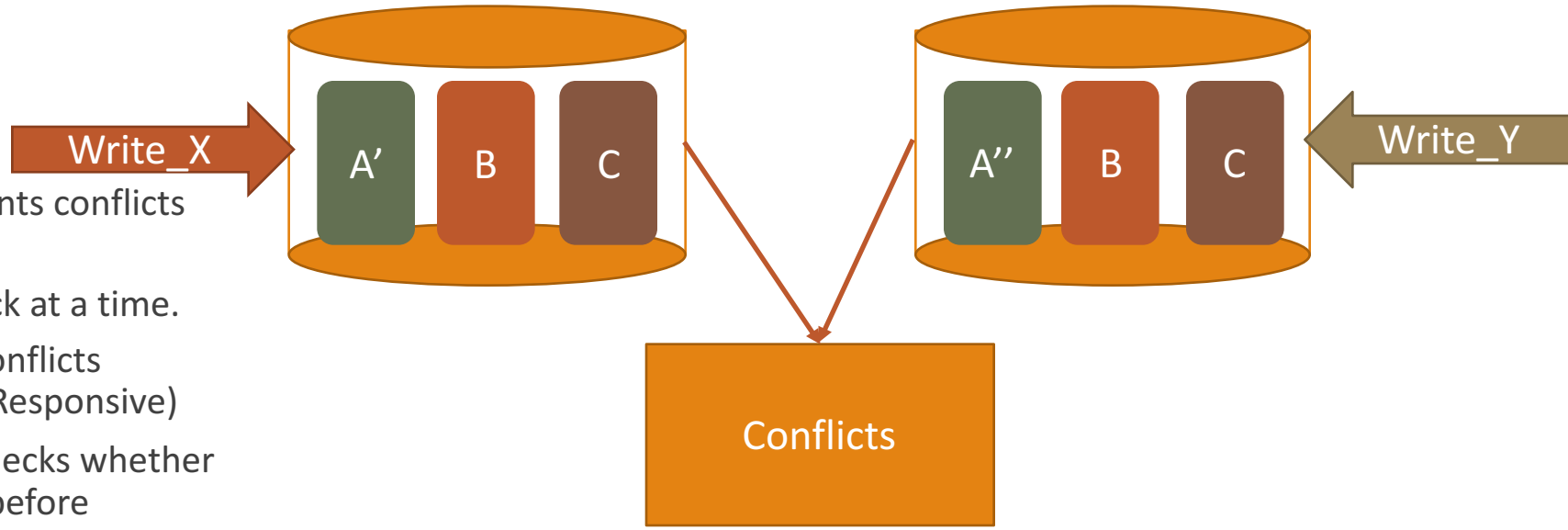
Requires sequential consistency.



Consistency

Update Consistency

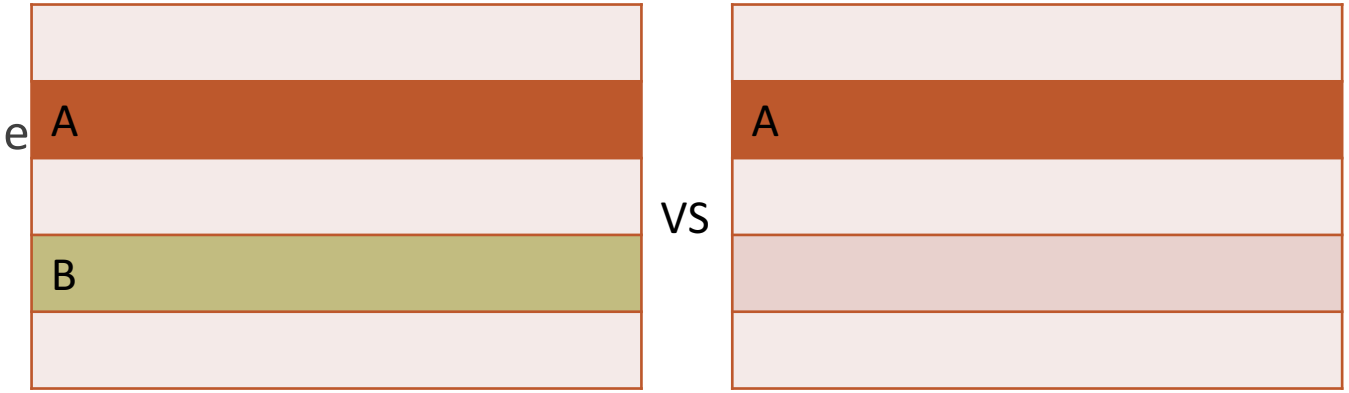
- Write-write conflict
- Consistency control
  - Pessimistic approach – Prevents conflicts (Safety)
    1. Use write locks – One lock at a time.
  - Optimistic approach – Lets conflicts happen, but take care of it. (Responsive)
    1. Conditional updates – Checks whether there is an update right before updating data.
    2. Version control – Save both updates and record that they are in conflict.



Consistency

Read Consistency

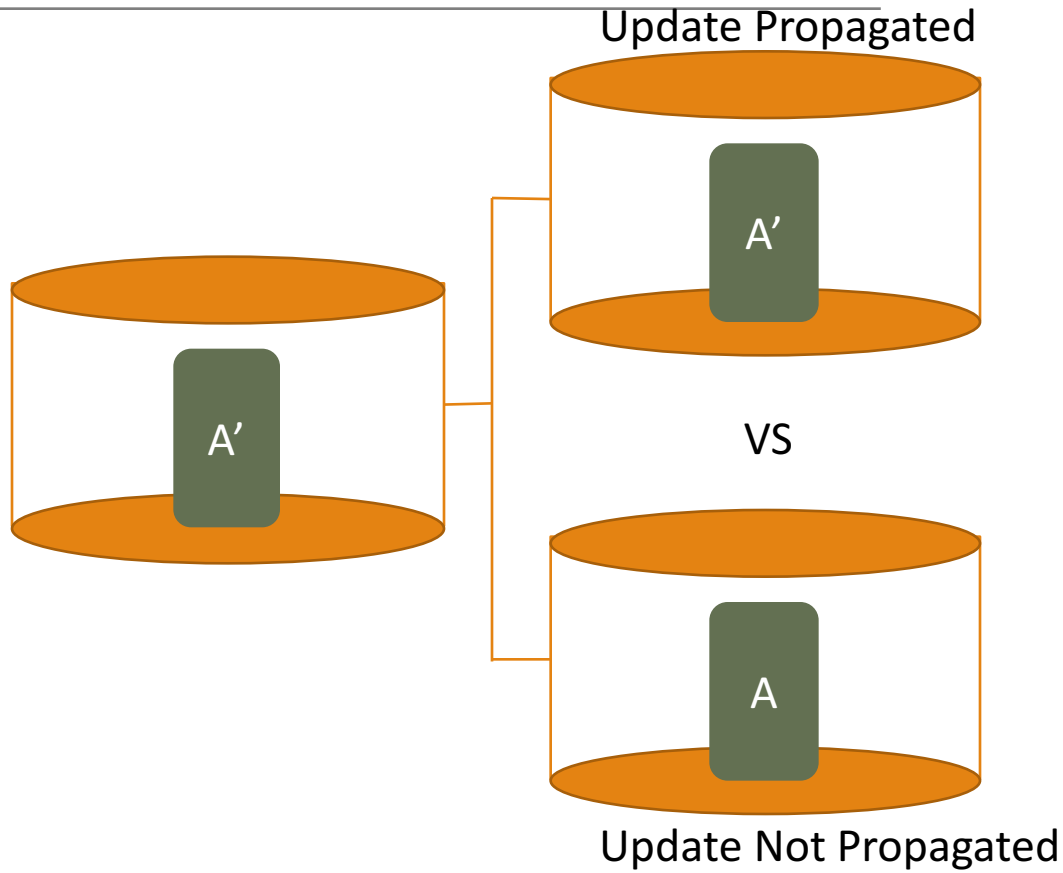
- Read-write conflict (inconsistent read)
  - Logical Consistency
    - Different data items make sense together.
  - RDBMS – Transaction.
- Inconsistency window – the length of time that inconsistency exists.



Consistency

Read Consistency

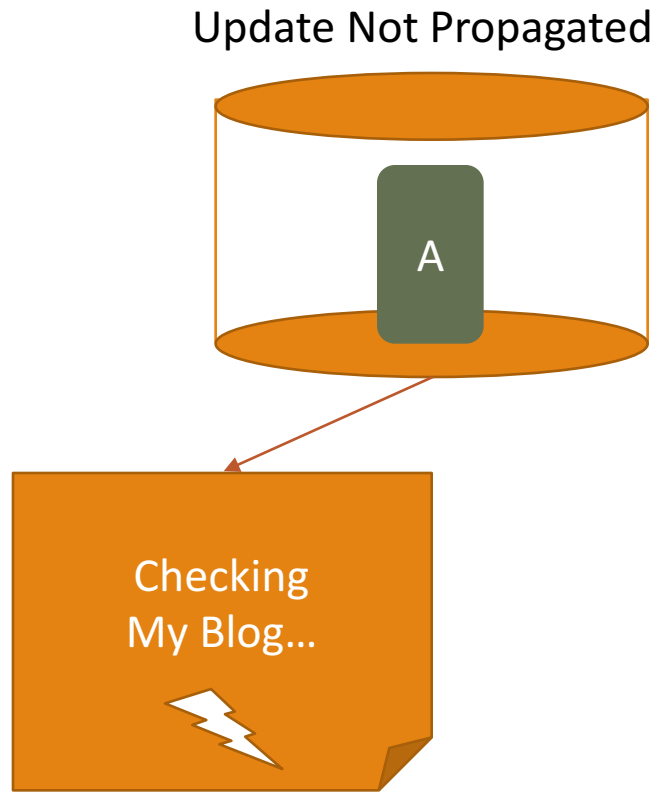
- Replication Consistency
  - The same data has the different value when reading from different replicas.
    - Eventually consistent : The updates will propagate fully eventually.



Consistency

Read Consistency

- Session Consistency (Read-your-writes Consistency)
  - Sticky Session (Session affinity) : A session is tied to one node and keep session consistency on the node.
  - Version Stamp : Make sure that every interaction with a node returns data with the latest version stamp seen by the session.



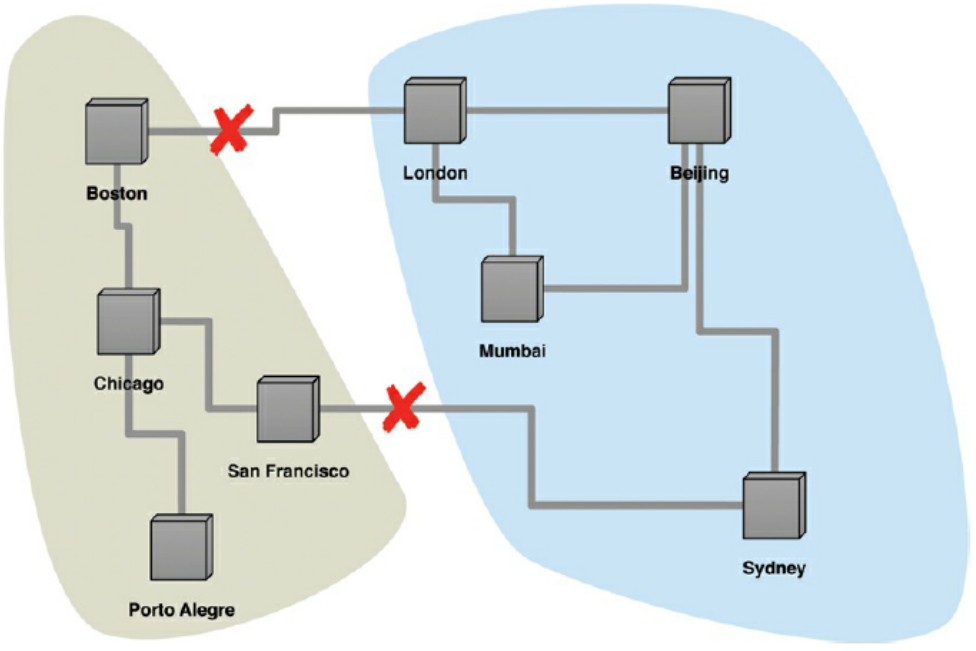
Consistency

Relaxing Consistency/Availability

- CAP Theorem
  - Consistency
    - All nodes have most recent data via eventual consistency.
  - Availability
    - Every request received by a non-failing node must return a response.
  - Partition Tolerance
    - Clusters can survive from communication breakages in the cluster.

➔ You can only get two.

ACID addresses an individual node's data consistency.
CAP addresses cluster-wide data consistency .

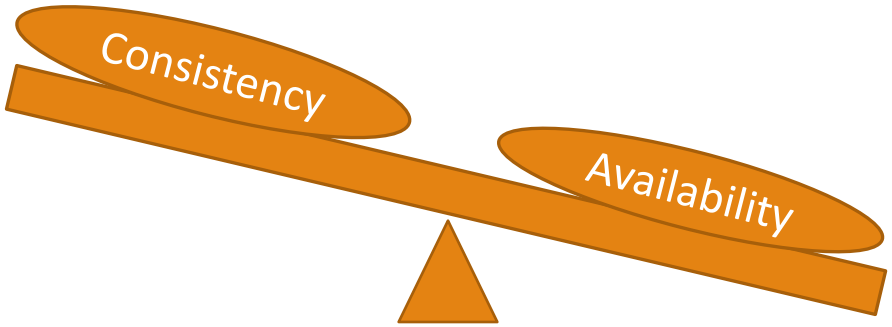


http://blingtechs.blogspot.com/2016/02/cap-theorem.html

Consistency

Relaxing Consistency/Availability

- CAP Theorem and Distributed Database
  - Requirement - Partition-Tolerance *
  - Availability or Consistency??
    - Availability – Shopping
    - Consistency – Stock Market



Consistency

Quorum

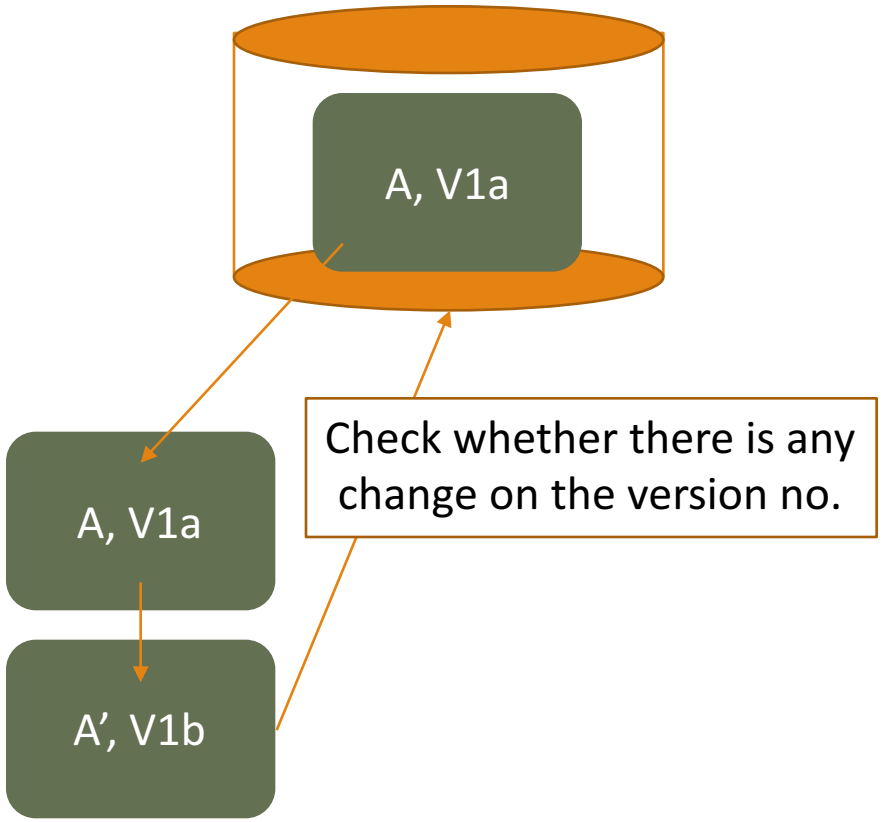
- Write Quorum : How many nodes should acknowledge your writes?
  - W > N/2 (W: nodes sending confirmation in write, N – nodes involved in replication (replication factor))
- Read Quorum : How many nodes you need to contact to make sure you have the most up-to-date value?
  - Depending on W.
  - For strong read consistency, R+W > N (R: nodes to be contacted for read consistency)

Version Stamps

Version Stamps : A field that changes every time when the data value is changed.

- Check updates won't be based on stale data.
- Example
  - Counter – increment when you update the data. Need a single master.
  - GUID – A large random number (Unique). Can't tell the recentness.
  - Resource content hash - Deterministic. Can't tell the recentness.
  - Timestamp - Clocks have to be kept in sync. Can't take care of too granular updates.

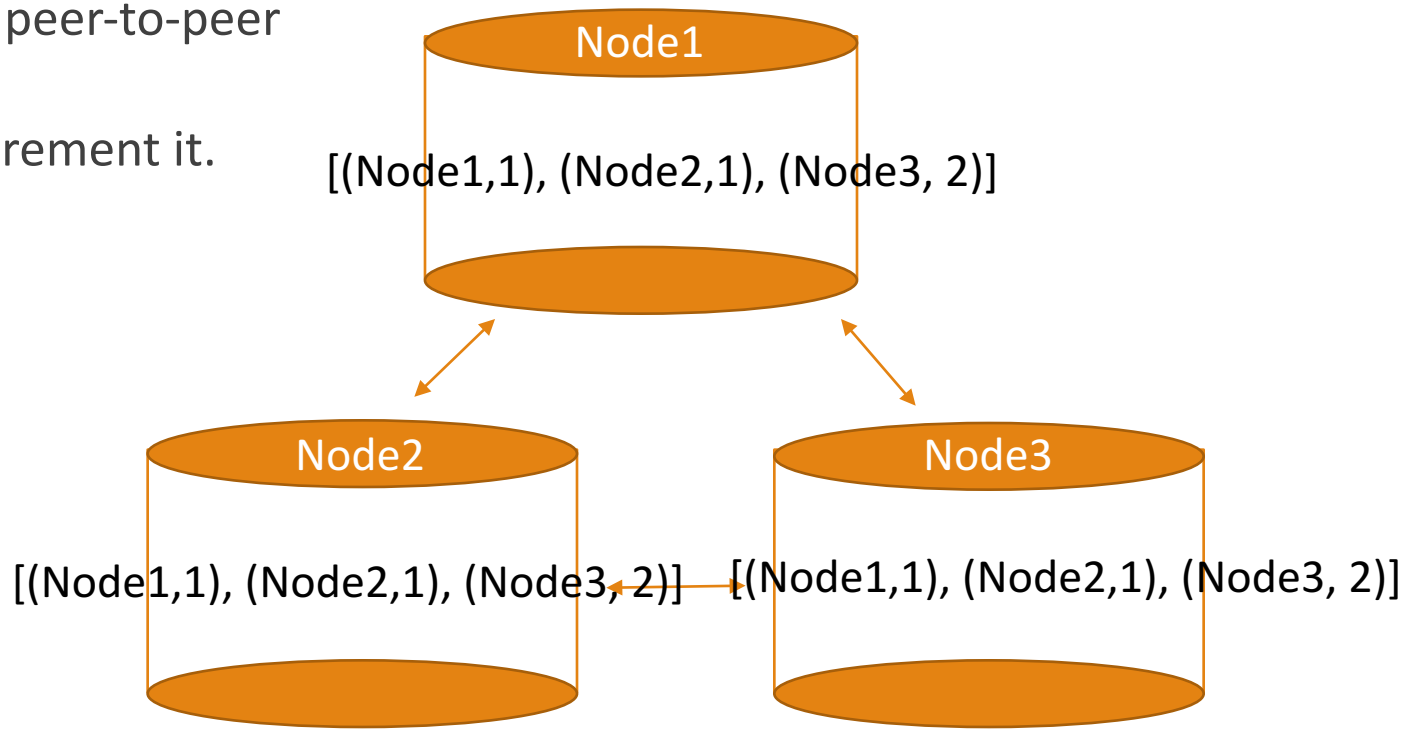
➔ Use more than one.



Version Stamps

Vector Stamp

- Good for managing version stamps on peer-to-peer NoSQL.
- Each node has its own counter and increment it.
- Vector of (NodeID, Counter)
  - Ex. [(Node1, 1), (Node2, 2) , (Node3, 6)]



Week 3: NoSQL (Cont.) and Document Databases

DIANE WOODBRIDGE, PH.D

Map-Reduce Model

Map-Reduce: Allow computations to be parallelized over a cluster.

- Basic Map-Reduce
  - The map-reduce framework plans map tasks to be run on the correct nodes and move data for the reduce function.
  - Map : Return key-value pairs from a single aggregate.
  - Reduce : Return one key-value pair from multiple key-value pairs.

Walmart Order

OrderID: 1
Customer : Diane
Timestamp : 2016-08-15 05:04:32 PST
Items:
{ProductName : San Francisco Giants Hat M,
{Qty: 2, UnitPrice : 10, Price : 20}},
{ProductName : San Francisco Giants Hat S,
{ Qty: 3, UnitPrice : 8, Price : 24}}
Shipping: Corte Madera

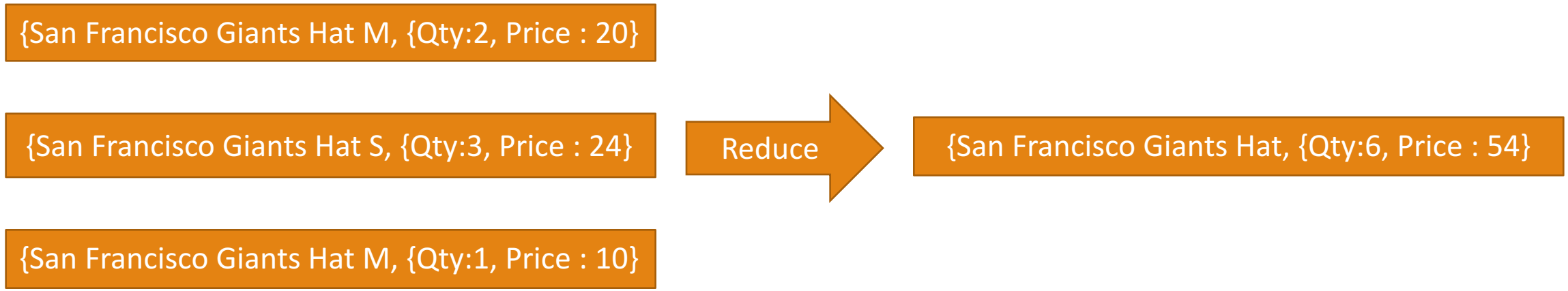
{San Francisco Giants Hat M, {Qty:2, Price : 20}

{San Francisco Giants Hat S, {Qty:3, Price : 24}

Map-Reduce Model

Map-Reduce: Allow computations to be parallelized over a cluster.

- Basic Map-Reduce
  - The map-reduce framework plans map tasks to be run on the correct nodes and move data for the reduce function.
  - Map : Return key-value pairs from a single aggregate.
  - Reduce : Return one key-value pair from multiple key-value pairs.



MongoDB Interview Questions

MongoDB's type

MongoDB's characteristics

Alternative databases

Supported programming languages

Index

Aggregation Operations(aggregation pipeline)

Sharding

Replication

GridFS

ObjectId

Consistency

Document Databases

Document Databases: For storing and retrieving documents with self-contained schema including XML, JSON, etc.

- Store documents in the value part of the key-value store.

```
<?xml version="1.0" standalone="yes"?>
<BankAccount>
      <Number>1234</Number>
      <Type>Checking</Type>
      <OpenDate>11/04/1974</OpenDate>
      <Balance>25382.20</Balance>
      <AccountHolder>
            <LastName>Singh</LastName>
            <FirstName>Darshan</FirstName>
      </AccountHolder>
</BankAccount>
```

```
{
    "empid": "SJ011MS",
    "personal": {
        "name":"Smith Jones",
        "gender": "Male",
        "age": 28,
        "address":
        {
            "streetaddress": "7 24th Street",
            "city": "New York",
            "state": "NY",
            "postalcode": "10038"
        }
    },
    "profile": {
        "designation": "Deputy General",
        "department": "Finance"
    }
}
```

www.kodingmadesimple.com

Document Databases

- Schema of the data can differ across documents, but these documents can still belong to the same collection.
  - Some attributes do not exist in another document.
  - New attributes can be created without defining them in the existing documents.

```
{
_id : 1234
firsname : Diane,
lastname : Woodbridge,
address :
                {state: CA, city: Corte Madera},
Interest : Books
}
```

```
{
_id :2345
firsname : Diane,
lastname :  Woodbridge,
order :
        {“Chodorow, Kristina. MongoDB:
        the definitive guide. " O'Reilly
        Media, Inc.", 2013.”}
type :  e-copy
}
```

MongoDB 

Why MongoDB?

- Easy of use
  - No defined schema : Key/values are not fixed types/sizes.
- Easy scaling
  - MongoDB takes care of
    - Balancing data.
    - Loading data across a cluster.
    - Redistributing documents automatically.
    - Routing user request to the correct machines.

MongoDB

Why MongoDB?

- Many features
  - Creating, reading, updating, and deleting (CRUD) data.
  - Indexing : Supports secondary indexes, allowing fast queries.
  - Aggregation pipeline : Allow you to build complex aggregations from simple pieces.
  - Special collection types : Time-to-live collections(session), fixed-size collections.
  - File storage : Stores large files and file meta data (GridFS).
- Supported drivers
  - C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go and Erlang

https://docs.mongodb.com/ecosystem/drivers/

MongoDB

Commonly Supported Data Types

- Null- `null`
- Boolean – `true`, `false`
- Number – 64 bit float (default), `NumberInt()` (4-byte), `NumberLong()` (8 byte)
- String – UTF-8 characters
- Date – Stored as millisecond since the epoch (No time zone). `new Date()`
- Regular expression – Use JavaScript's regular expression syntax.
- Array – `[Element1, Element2,…]`
- Embedded document – Entire documents can be embedded as value.
- Object ID – 12-Byte ID for documents. "`_id`"
- Binary data – String of arbitrary bytes.
- Code – JavaScript code.

MongoDB

Commonly Supported Data Types

- Object ID – 12-byte ID for documents and a default for "`_id`".
  - Generated by the driver on the client.
  - Every document in MongoDB must have an "`_id`" key.
    - Its value can be any type, but needs to be unique for a collection.
    - Default is `ObjectId`.

MongoDB

CRUD Overview

- Update
  - Atomic.
  - `update(Arg1, Arg2, Arg3, Arg4, Arg5)`
    - Two required arguments.
      - Arg1: Criteria to find
      - Arg2: new document/change criteria.
    - Three optional arguments.
      - Arg3 : Upsert (default = false)
        - When no document meets the Arg1 criteria, it creates a new document.
      - Arg4 : multiple-update (default = false).
        - Update multiple documents that meet the Arg1 criteria.
      - Arg5 : writeConcern. Error handling.

https://docs.mongodb.com/v3.2/reference/method/db.collection.update/

MongoDB

CRUD Overview

- Update
  - Update certain portions of a document.
    - Use atomic update modifier.
      - Field modifier
      - Array modifier
        - Operators
        - Modifiers

MongoDB

Field modifier

Name	Description
`$inc`	Increments the value of the field by the specified amount.
`$mul`	Multiplies the value of the field by the specified amount.
`$rename`	Renames a field.
`$setOnInsert`	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
`$set`	Sets the value of a field in a document.
`$unset`	Removes the specified field from a document.
`$min`	Only updates the field if the specified value is less than the existing field value.
`$max`	Only updates the field if the specified value is greater than the existing field value.
`$currentDate`	Sets the value of a field to current date, either as a Date or a Timestamp.

https://docs.mongodb.com/manual/reference/operator/update/

MongoDB

CRUD Overview

- Update
  - Update certain portions of a document.
    - Use atomic update modifier.
      - Array modifier
        - Operators : $, $addToSet, $pop, $pullAll, $pull, $pushAll, $push
        - Modifiers : $each, $slice, $sort, $position

MongoDB

Array modifier

- Operators

Operators

Name	Description
`$`	Acts as a placeholder to update the first element that matches the query condition in an update.
`$addToSet`	Adds elements to an array only if they do not already exist in the set.
`$pop`	Removes the first or last item of an array.
`$pullAll`	Removes all matching values from an array.
`$pull`	Removes all array elements that match a specified query.
`$pushAll`	*Deprecated.* Adds several items to an array.
`$push`	Adds an item to an array.

https://docs.mongodb.com/manual/reference/operator/update/

MongoDB

Array modifier

- Modifier

Modifiers

Name	Description
`$each`	Modifies the `$push` and `$addToSet` operators to append multiple items for array updates.
`$slice`	Modifies the `$push` operator to limit the size of updated arrays.
`$sort`	Modifies the `$push` operator to reorder documents stored in an array.
`$position`	Modifies the `$push` operator to specify the position in the array to add elements.

https://docs.mongodb.com/manual/reference/operator/update/

MongoDB

CRUD Overview

- Delete
  - Permanently deletes documents from the database.
    - `remove(criteria)`
    - `drop()` : Delete entire collection efficiently.

```
> db.mydb.remove({"name" : "Yannet Interian"});
WriteResult({ "nRemoved" : 1 })
> db.mydb.find();
{ "_id" : ObjectId("581a1946ff611b997cb2bde7"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101
, "noKids" : 2 }
```

MongoDB

Use external database.

- `mongo hostname:port/dbname`
- ```
  conn = new Mongo("hostname:port")
  db = conn.getDB("dbname")
  ```
  ```

Week 4: Document Databases

DIANE WOODBRIDGE, PH.D

MongoDB- Querying

find (), findOne() Operators

- find() : Returns subset of document in a collection.
  - db.collection_name.find({search_criteria}, {key_to_return : 1/0})
    - Key_to_return : 1/true- include, 0/false- exclude

```
> db.friend.find({"name":"Diane MK Woodbridge"})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard
ts" : [ { "name" : "Abigail", "content" : "Please cook mac and cheese" }, { "name" : "Bora", "content" : "Need more
 your class." }, { "name" : "Yannet", "content" : "How's your class going?" } ] }
```

```
> db.friend.find({"name":"Diane MK Woodbridge"},{"comments":1})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "comments" : [ { "name" : "Abigail", "content" : "Please co
d", "content" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "H
```

MongoDB- Querying

Type-specific queries

- Query Arrays
  - `$slice`
    - Return a subset of elements for an array key.
    - Helpful when you know the index of the element.
    - db.collection_name.findOne(criteria, {array_name:{”$slice”: N}})
      - Number N
        - Positive N – first N elements
        - Negative N – last N elements
        - [index, # of element]

```
> db.food.find()
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "apple", "banana", "peach", "watermelon" ] }
```

MongoDB- Querying

Cursors

- MongoDB returns result from find() using a cursor.
- Using cursors you can control the output of a query including skip, sort, limit, etc.

Name	Description
`cursor.addOption()`	Adds special wire protocol flags that modify the behavior of the query.'
`cursor.batchSize()`	Controls the number of documents MongoDB will return to the client in a single network message.
`cursor.close()`	Close a cursor and free associated server resources.
`cursor.comment()`	Attaches a comment to the query to allow for traceability in the logs and the system.profile collection.
`cursor.count()`	Modifies the cursor to return the number of documents in the result set rather than the documents themselves.
`cursor.explain()`	Reports on the query execution plan for a cursor.

https://docs.mongodb.com/v3.2/reference/method/js-cursor/

MongoDB- Indexing

- Indexing
  - B-Tree is default.
  - Optimize query performance.
    - getIndexes()
      - Check which indexes exist on a collection.
      - Default : _id (Cannot be deleted)
    - createIndex({“field” : direction})
      - Direction : 1 (Ascending), -1 (Descending)
    - dropIndex({“field” : direction})

```
db.friend.find({"name":"Yannet Interian"}).explain("executionStats")
```

Try it.

https://docs.mongodb.com/v3.0/tutorial/modify-an-index/

https://docs.mongodb.com/v3.2/tutorial/analyze-query-plan/

MongoDB- Indexing

- Indexing
  - B-Tree is default.
  - Optimize query performance.
    - getIndexes()
      - Check which indexes exist on a collection.
      - Default : _id (Cannot be deleted)
    - createIndex({“field” : direction})
      - Direction : 1 (Ascending), -1 (Descending)
    - dropIndex({“field” : direction})

Try it.

```
> db.friend.createIndex({"name":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
db.friend.find({"name":"Yannet Interian"}).explain("executionStats")
```

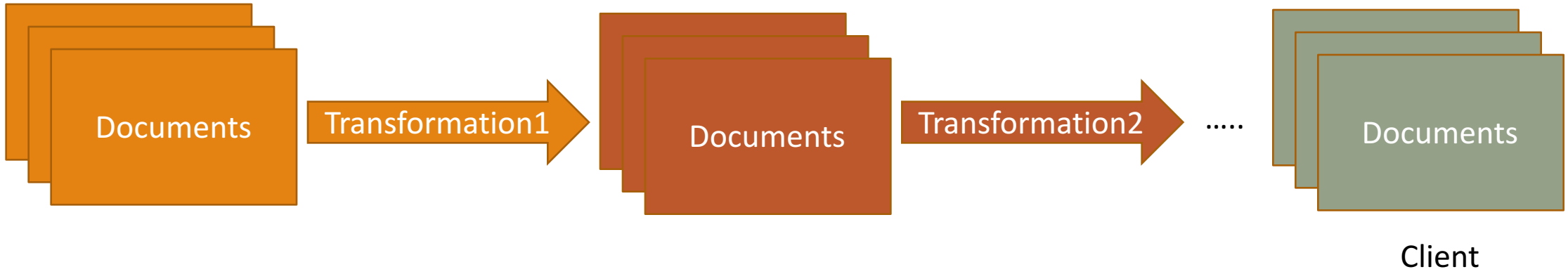
https://docs.mongodb.com/v3.0/tutorial/modify-an-index/


```
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 3,
                "totalKeysExamined" : 1,
                "totalDocsExamined" : 1,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 1,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 2,
                        "advanced" : 1,
                        "needTime" : 0,
                        "needYield" : 0,
                        "saveState" : 0,
                        "restoreState" : 0,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "docsExamined" : 1,
                        "alreadyHasObj" : 0,
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "nReturned" : 1,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 2,
                                "advanced" : 1,
                                "needTime" : 0,
                                "needYield" : 0,
                                "saveState" : 0,
                                "restoreState" : 0,
                                "isEOF" : 1,
                                "invalidates" : 0,
                                "keyPattern" : {
                                        "name" : 1
                                },
                                "indexName" : "name_1",
                                "isMultiKey" : false,
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 1,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "name" : [
                                                "[\"Yannet Interian\", \"Yannet Interian\"]
                                        ]
                                },
                                "keysExamined" : 1,
                                "dupsTested" : 0,
                                "dupsDropped" : 0,
```

MongoDB

Aggregation Pipeline

- Returns an array of result document to the client. (Not write to collections.)
  - db.collecion_name.aggregate({Transform_operator_1 : criteria_1},{Transform_operator_2 : criteria_2}, …)
  - $match, $project, $lookup, $group, $unwind, $out, etc.



Week 5: Document Databases and Column-family Stores

DIANE WOODBRIDGE, PH.D

MongoDB

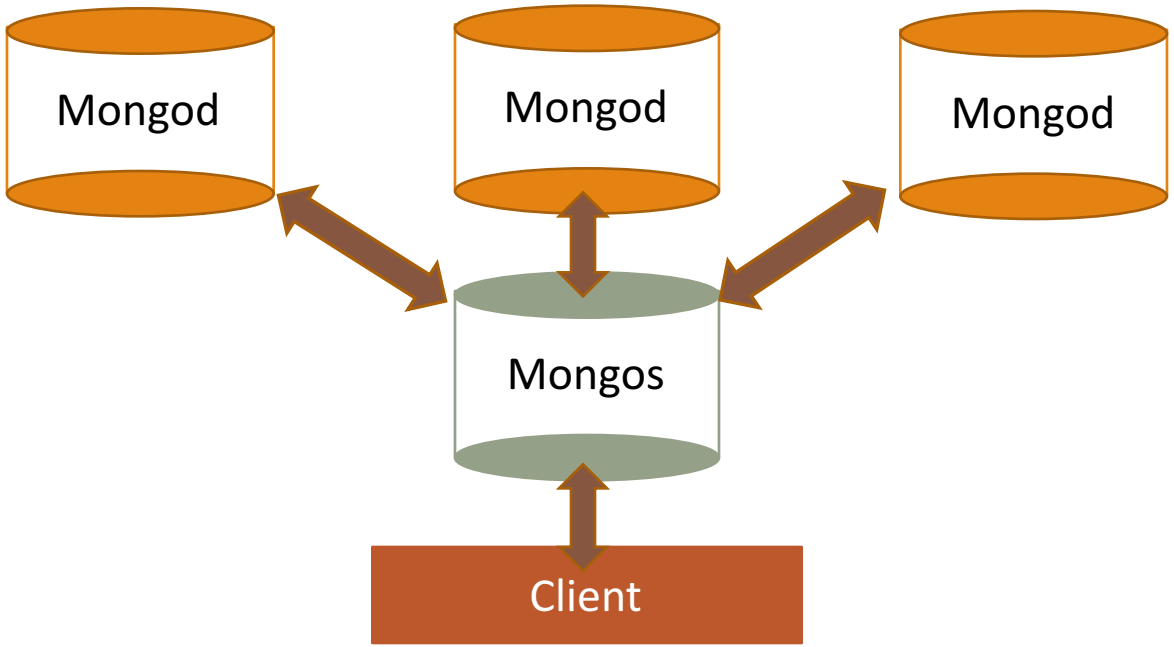
Replication

- Replica sets should always have an odd number of members.
  This ensures that election using quorum will proceed smoothly.
- Create 3 folders.
  - rs_1, rs_2, rs_3
  - start all 3 mongod processes with
    - --dbpath definite_path_of_each_folder
    - --port port_number
    - --replSet replica_set_name

MongoDB

Sharding

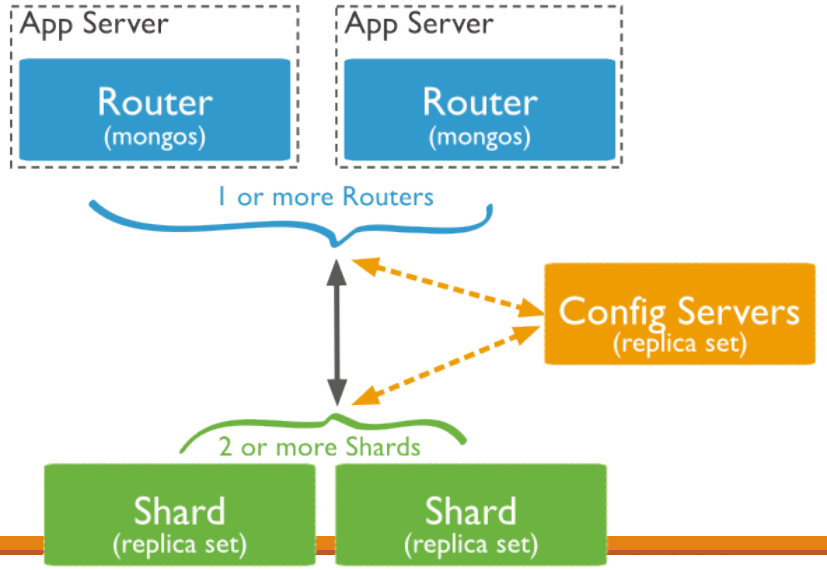
- MongoDB supports autosharding.
  - Create a cluster of many machines (shards).
  - Mongos : Routing process to manage storing different data on different servers and query against the right server.



MongoDB

Sharding

- Config servers store the metadata for a sharded cluster.
  - The metadata stores
    - state and organization for all data and components within the sharded cluster.
    - the list of data chunks on every shard and the ranges that define the chunks.
  - In MongoDB 3.2+, you can deploy config servers as a replica set.



https://docs.mongodb.com/v3.2/tutorial/convert-replica-set-to-replicated-shard-cluster/

MongoDB

Features

- Consistency
  - Use replica sets and wait for the writes to be replicated to all/some slaves.
  - Read consistency
    - Secondaries will refuse read requests by default to prevent from reading from stale data.
    - Setting up setSlaveOk() allows read operations to run on secondaries .
  - Write consistency
    - For a replica set, the default writeConcern() requests acknowledgement only from the primary.
    - Overriding the write concern allows to confirm write operations on a specified number of the replica set members.
      - db.friend.insert({ name: ”Jonathan"}, { writeConcern: {w: “majority”} })

Make sure to propagate to the primary and at least one secondary.

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

Cassandra Interview Questions

Cassandra data model

Tunable consistency (Quorum)

Difference between RDBMS and Cassandra

CAP Theorem

Cassandra query language

cqlsh

Compaction

Super column

Column family

Memtable

SSTable

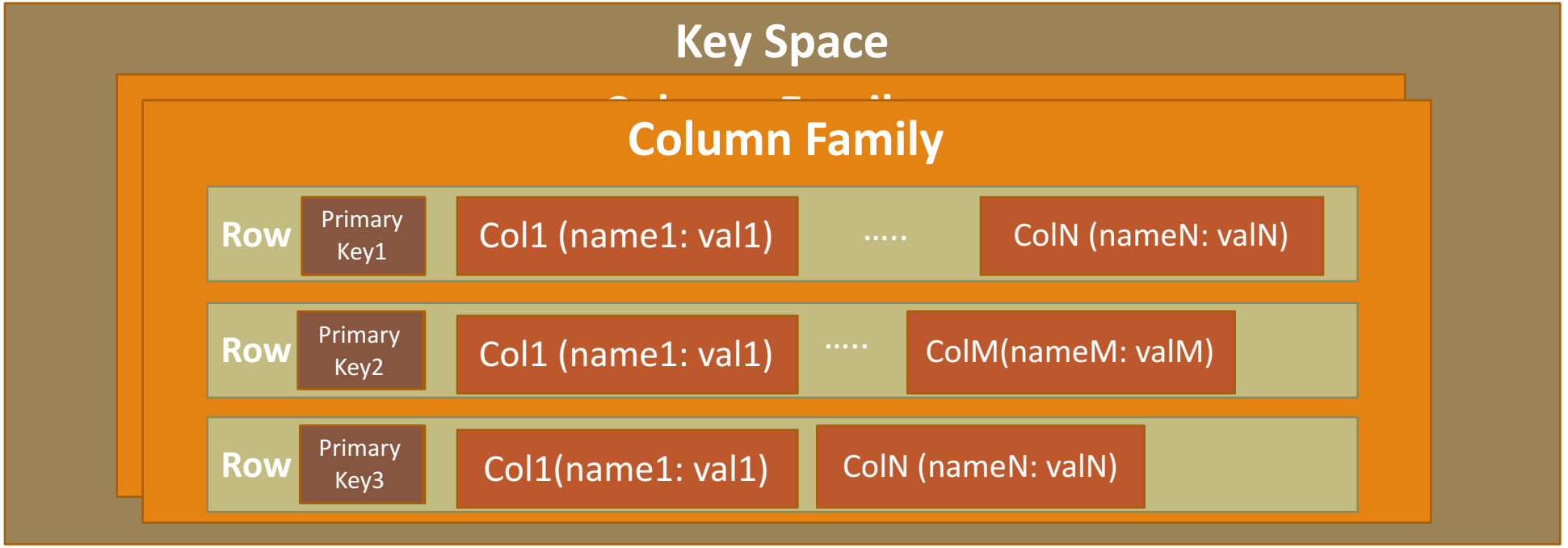
Keyspace

Column-Family Stores

Column-family Stores: also called as wide column stores.

Store data in column families where rows that have many columns associated with a primary key (row key).

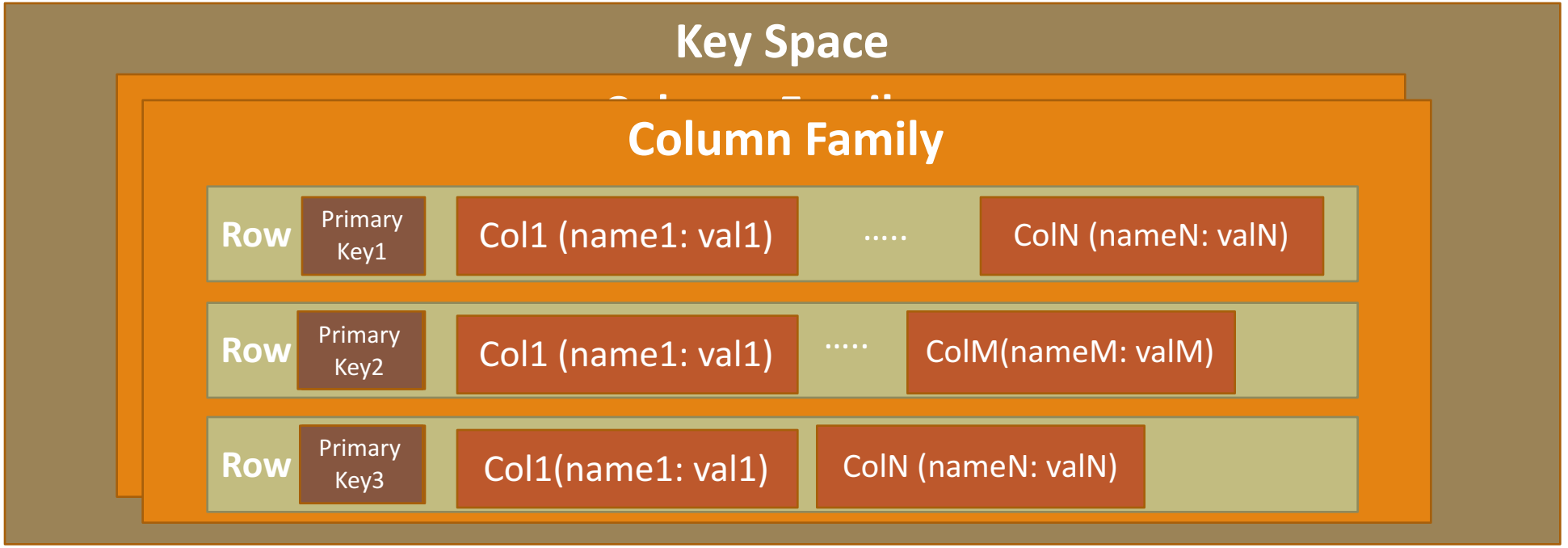
Column families are groups of related data that is often accessed together.



Column-Family Stores

Various rows do not have to have the same columns.

Columns can be added to any row at any time without having to add it to other rows.



Cassandra

Why Cassandra?

- Open source
- Distributed and decentralized
- Highly scalable
- High availability and fault tolerant
- Tunable consistent
- High performance (Writes *)

Cassandra

Terminology

- Keyspace
- Column family

SQL(RDBMS)	Cassandra
	Cluster
Database	Keyspace
Table	Column Family/Table
Row	Row
Column	Column (can be different per row)

http://cassandra.apache.org/doc/latest/

Cassandra

Structure

- Column : Basic/atomic unit of storage.
  - Name(Key)-value pair.
  - Stored with a timestamp.
    - Used to expire data, resolve conflicts, deal with stale data, etc.
  - Super column : a map of columns.
    - Pros : Good to keep related data together.
    - Cons: When some of the columns are not needed most of the time, the columns are still fetched and deserialized.
- Row : collection of columns attached to a key.
  - Data is organized around rows.
  - Doesn't need to have all the same columns. (Flexible schema)
  - Wide row (Partition)
- Column family : collection of similar rows.

http://cassandra.apache.org/doc/latest/

Cassandra

Column Data Type

CQL Type	Description
ascii	US-ASCII character string
bigint	64-bit signed long
blob	Arbitrary bytes (no validation), expressed as hexadecimal
boolean	true or false
counter	Distributed counter value (64-bit long)
decimal	Variable-precision decimal
double	64-bit IEEE-754 floating point
float	32-bit IEEE-754 floating point
inet	IP address string in IPv4 or IPv6 format*
int	32-bit signed integer
list	A collection of one or more ordered elements
map	A JSON-style array of literals: { literal : literal, literal : literal ... }
set	A collection of one or more elements
text	UTF-8 encoded string
timestamp	Date plus time, encoded as 8 bytes since epoch
uuid	A UUID in standard UUID format
timeuuid	Type 1 UUID only (CQL 3)
varchar	UTF-8 encoded string
varint	Arbitrary-precision integer

https://docs.datastax.com/en/cql/3.0/cql/cql_reference/cql_data_types_c.html

Cassandra

Column Data Type

- uuid : type 4 universal unique identifier. Fixed length random number (128 bit value) .
- timeuuid : type1 universal unique identifier. Combinatino of MAC address, system time and sequence numbers.
- Set : unordered collection. {}
- List : ordered collection. []
- Map : collection of key-value pairs.
- User defined type (UDT). : considered as a set.

```
CREATE TYPE name { name type, name type, …};
```

```
cqlsh:mydb> CREATE TYPE address (street text, city text, state text, zip int);
```

https://docs.datastax.com/en/cql/3.0/cql/cql_reference/cql_data_types_c.html

Cassandra

Create a table.

- CREATE TABLE table_name (name type, name type, PRIMARY KEY(name));

```
cqlsh:mydb> CREATE TABLE friend(name text, PRIMARY KEY (name));
```

```
cqlsh:mydb> DESCRIBE TABLE friend;

CREATE TABLE mydb.friend (
    name text PRIMARY KEY
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Data access requires primary key!
Primary key cannot be modified.

Casandra

Selecting the data which is not a primary key.

- Query is not allowed.
- Assign a secondary index on a column that is not part of the primary key.
  - Each node maintains its own copy of a secondary index based on the data.
  - Secondary indexes are not recommended.
  - ➔ Denormalized tables or materialized views are preferred.

Queries involving a secondary index typically involve more nodes, making them more expensive.

```
cqlsh:mydb> select * from friend WHERE emails CONTAINS 'dwoodbridge@usfca.edu';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may
u want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
cqlsh:mydb> select * from friend WHERE emails CONTAINS 'dwoodbridge@usfca.edu' ALLOW FILTERING;

 name                | address                                                                                | emails
      | kids                          | phone_numbers
---------------------+----------------------------------------------------------------------------------------+-----------------------------------------
-------+-------------------------------+-----------------------------------
 Diane MK Woodbridge | {street: '101 Howard St', city: 'San Francisco', state: 'CA', zip: null} | {'dwoodbridge@gmail.com', 'dwoodbridge@usfca
5c38ae | {'Abigail': 6, 'Jackson': 2} | ['123-456-7890', '111-222-3333']

(1 rows)
```

Cassandra

RDBMS VS. Cassandra

- No Joins : Have to work on the client side or create denormailzed second table/materilalized view (preferred).
  - Materialized view :

```
CREATE MATERIALIZED VIEW view_name
AS SELECT  column_names FROM table_name WHERE constraints
PRIMARY KEY (primary_keys_in_table_name_and_additional);
```

- No referential integrity across tables : no primary-foreign key relationships.
- Query-first design : Model the queries first and let the data be organized around them.
  - Logical design and then physical design (data type, etc.)

Materialized Views

http://www.datastax.com/dev/blog/new-in-cassandra-3-0-materialized-views

https://docs.datastax.com/en/cql/3.3/cql/cql_using/useCreateMV.html

Week 7: Column-family Stores

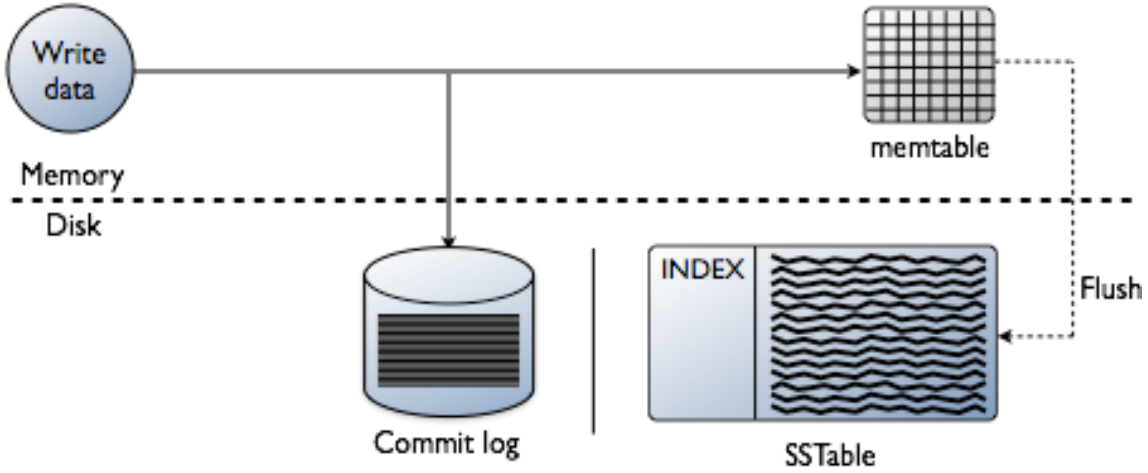
DIANE WOODBRIDGE, PH.D

Cassandra

Writes

VERY FAST! (WHY??)

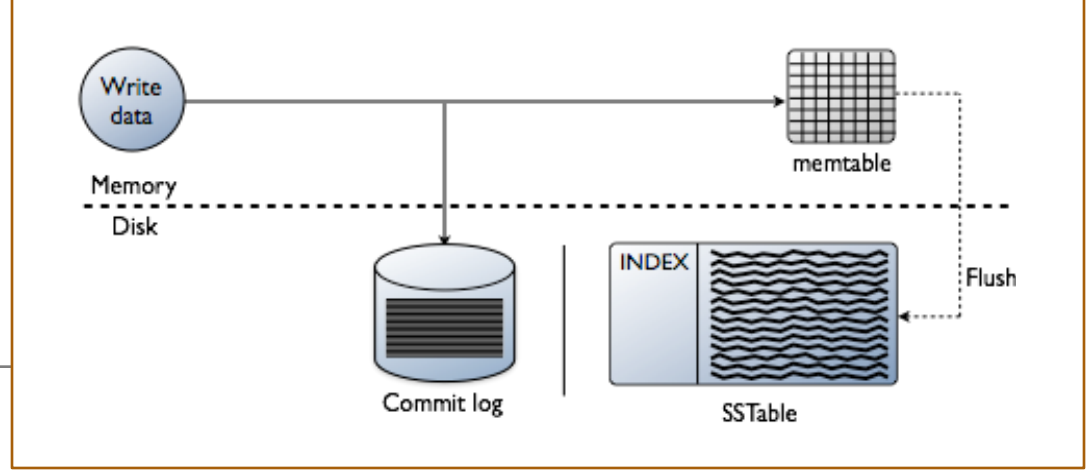
- When a write is received by Cassandra, the data is first recorded in a **commit log**, then written to an in-memory structure called **memtable**. A write operation is considered successful once it is written to the commit log and the memtable. Writes are batched in memory and periodically written out to structures known as **SSTable**.



1. Logging data in the commit log
2. Writing data to the memtable
3. Flushing data from the memtable
4. Storing data on disk in SSTables
5. Compaction

https://docs.datastax.com/en/cassandra/2.2/cassandra/dml/dmlHowDataWritten.html

Cassandra

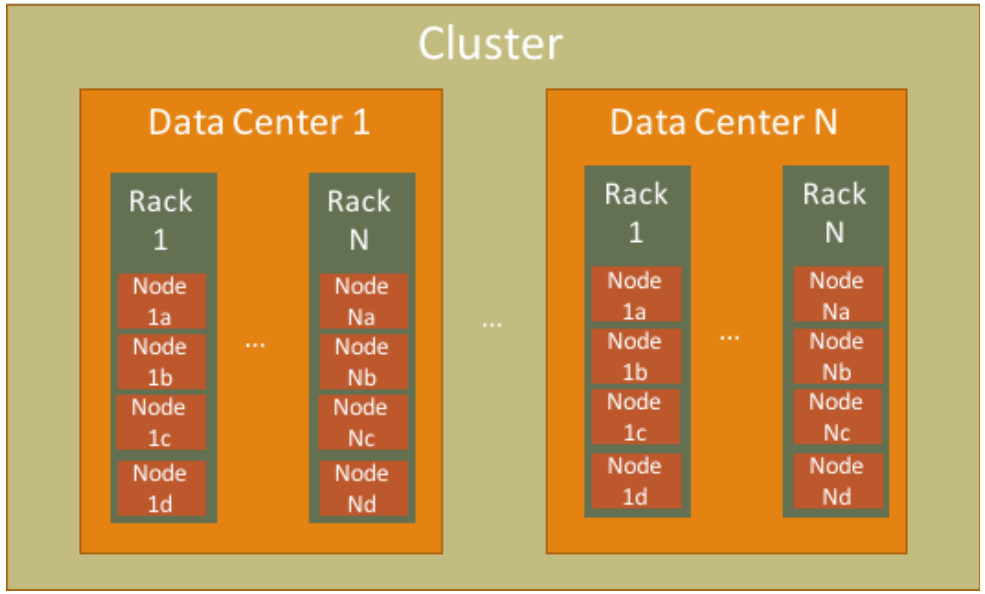


Writes

1. **Logging data in the commit log.** : It is a crash-recovery mechanism. If a write operation does not make it to the in-memory store (memtable), it will still be possible to recover the data.
2. **Writing data to the memtable.** : memtable is a memory-resident data structure which stores data rows that can be looked up by key.
3. **Flushing data from the memtable.** : When he number of objects stored in the memtable reaches a threshold, the contents are flushed to disk called SSTable .
4. **Storing data on disk in SSTable.** : SSTables are immutable, not written to again after the memtable is flushed.
5. **Periodic compaction.** : Reorganize SSTables for better future read performance.

https://docs.datastax.com/en/cassandra/2.2/cassandra/dml/dmlHowDataWritten.html

Cluster Topology in Cassandra



Node (vnode): A node is the storage layer within a server.

Rack : A logical set of nodes in close proximicity .

Data Center : A logical set of racks.

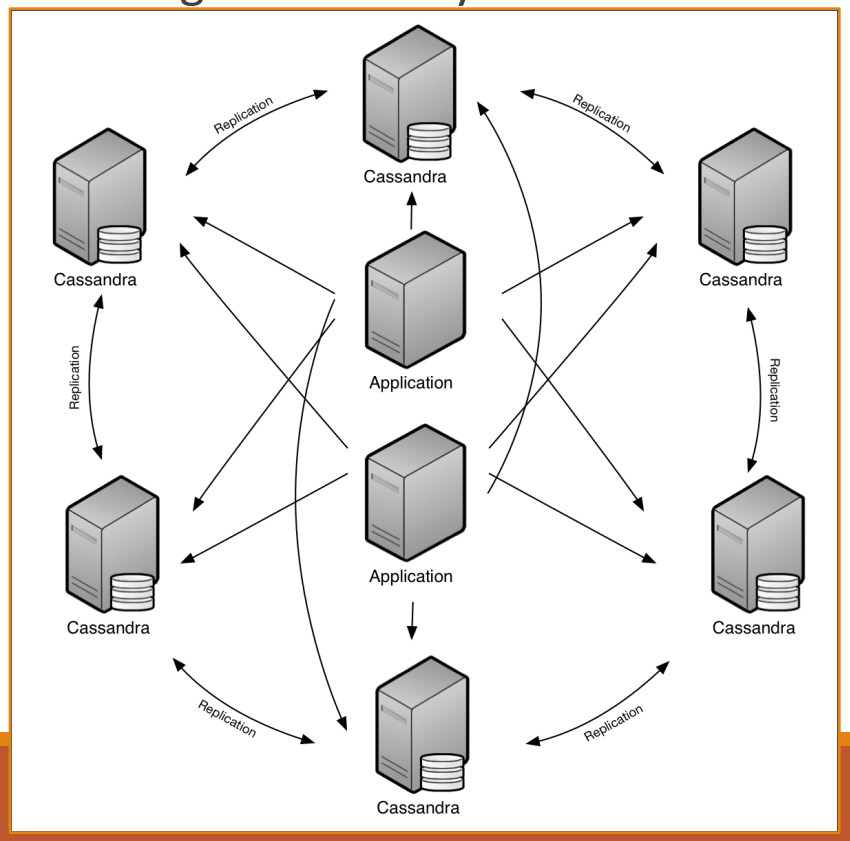
Cluster : A set of data centers.

Snitches : Gather information about the topology so that Cassandra can efficiently route requests.

Cassandra

Replication

- All replicas are equally important; there is no primary or master replica.
- Uses peer-to-peer replication. ➔ High availability.



Cassandra

Replication Strategies (2)

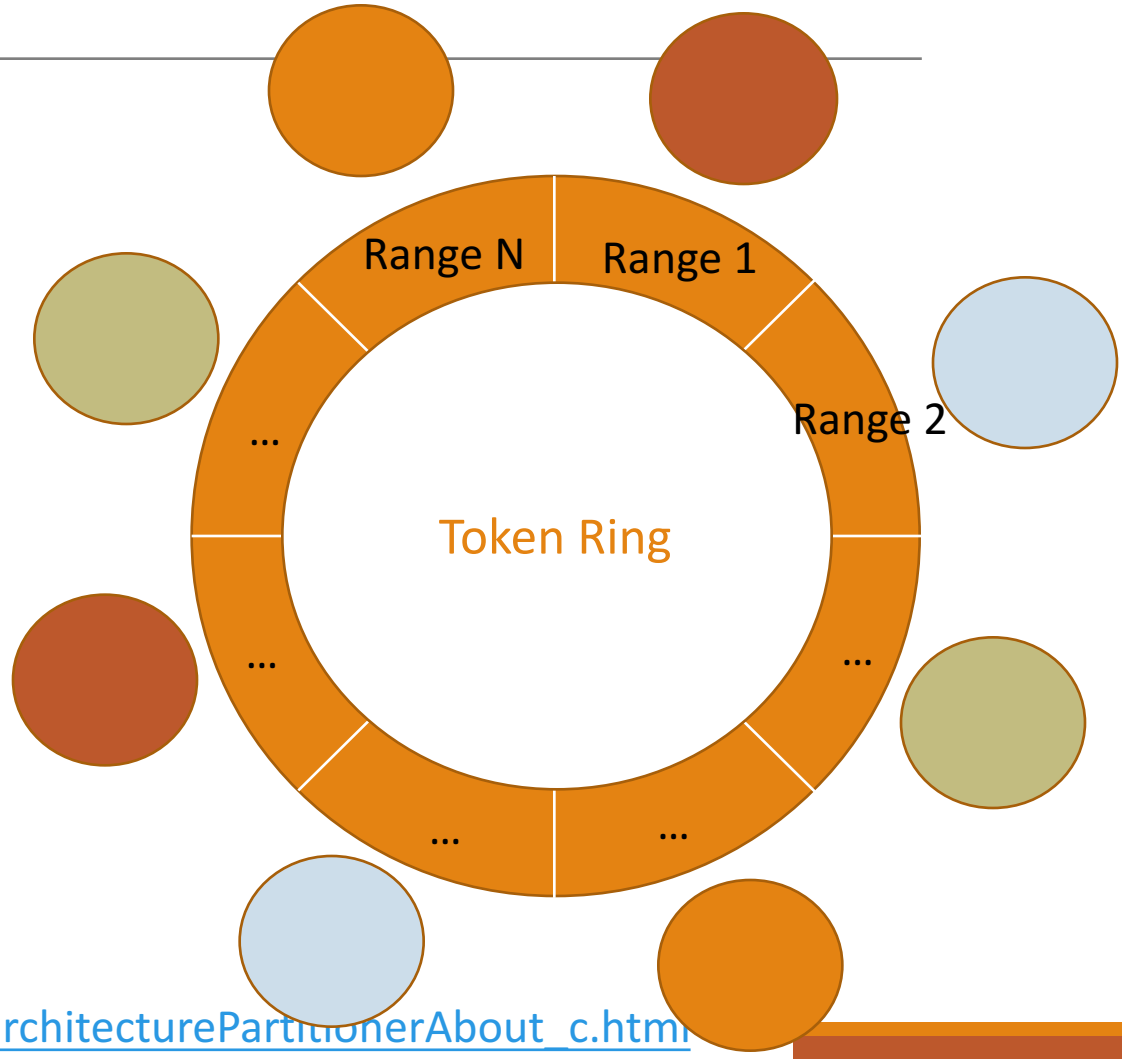
- SimpleStrategy
  - Use only for a single data center.
  - Not aware of their placement on a data center rack
- NetworkTopologyStrategy
  - Use NetworkTopologyStrategy when you have (or plan to have) your cluster deployed across multiple data centers. This strategy specify how many replicas you want in each data center. NetworkTopologyStrategy attempts to place replicas on distinct racks because nodes in the same rack (or similar physical grouping) often fail at the same time due to power, cooling, or network issues.
  - Much easier to expand to multiple data centers when required by future expansion.

https://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architectureDataDistributeReplication_c.html

Cassandra

Partitioning (Sharding)

- Token Ring
  - Each node is assigned to a position in a ring with a range of the token values.
    - Token : 64-bit integer ID used to identify each partition.
  - Try to arrange consecutive token ranges to be spread across nodes in different racks.
- Partitioner
  - Determines how data is distributed across the nodes in the cluster by computing the partition key token.
  - By default, it uses hash function. (Murmur3Partitioner)



https://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architecturePartitionerAbout_c.html
https://docs.datastax.com/en/datastax_enterprise/4.5/datastax_enterprise/deploy/deployCalcTokens.html

Cassandra

Tunable Consistency

- Setting `Consistency` will ensure the majority of the nodes to respond to read/write.
  - Read: The column with the newest timestamp is returned back to the client.
  - Write: The new update will be propagate to the majority of the nodes.
  - Available consistency levels : ALL, EACH_QUORUM, QUORUM, LOCAL_QUORUM, ONE, TWO, THREE, LOCAL_ONE, ANY, SERIAL, LOCAL_SERIAL.

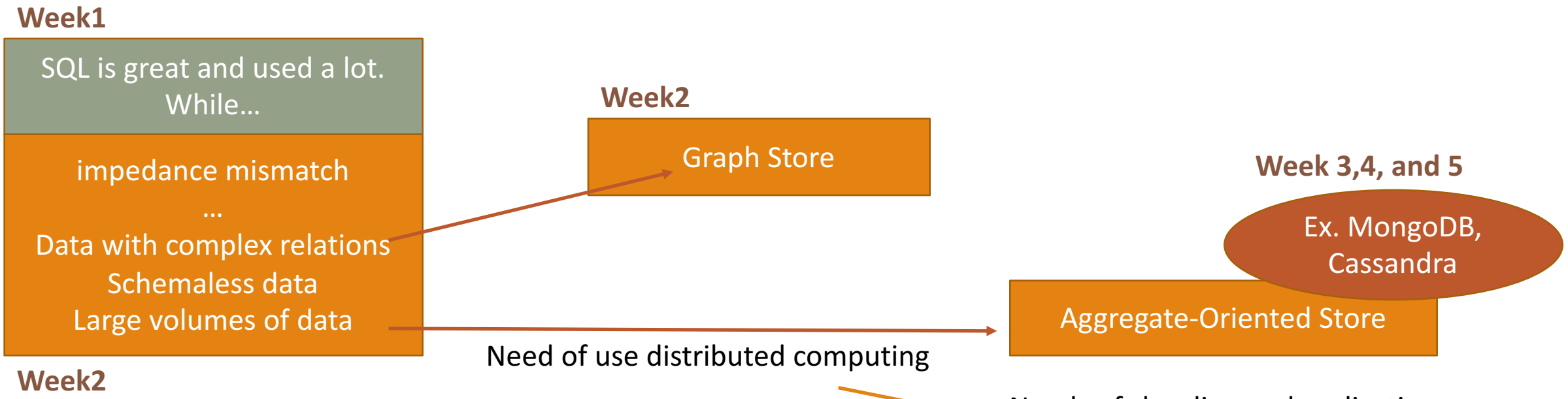
```
cqlsh> consistency;
Current consistency level is ONE.
cqlsh> consistency QUORUM;
Consistency level set to QUORUM.
```

ONE is default.

https://docs.datastax.com/en/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html

Summary

Remember the big picture!



Summary

NoSQL

Pros (In general)
Mostly open-source.
Schemaless.
Good for non-relational data.
Scalable.
Runs well on distributed systems.

Cons
Installation, toolsets still maturing.

Example
Redis, MongoDB, Cassandra, OrientDB

Two main reasons to consider NoSQL.

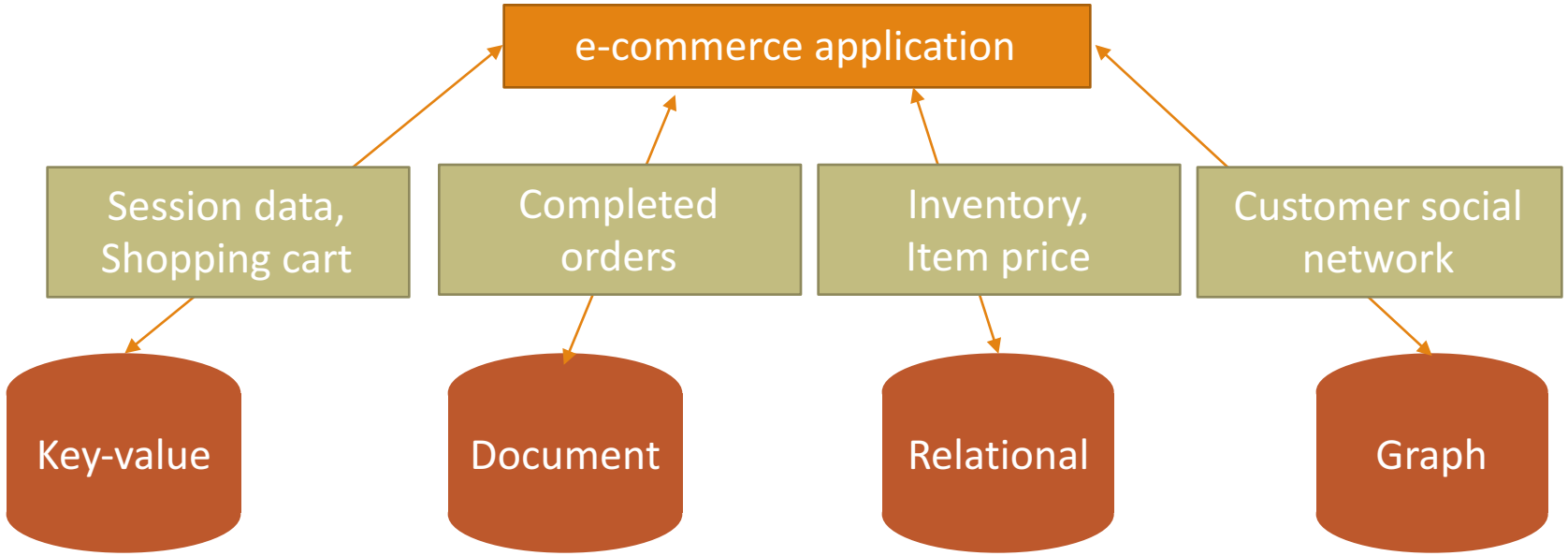
- To improve programmer productivity that better matches an application's needs.
- To improve data access performance via some combination of handling larger data volumes, reducing latency and improving throughput.

Test your expectations about programmer productivity and performance before committing to using a NoSQL technology.

Summary

Polygot Persistence

- Using multiple data storage technologies, chosen based on the way data is being used by individual applications.
- NoSQL data stores do not replace relational databases.



Relational Database Interview Questions

- Basic Operations
  - Create
  - Insert
  - Select
  - Update
  - Join (Inner, Outer, Left, Right, etc.)*
  - Union All/Union/ Union Distinct
  - Minus
  - Intersect
- Normalization (1NF, 2NF and 3NF)*
- Transaction(Concurrency Control) – ACID*
- Indexing* - B tree, Hash
- Truncate vs Delete
- Difference Where vs. Having

NoSQL Interview Questions

What is NoSQL?

Relational Database vs. NoSQL

Impedence mismatch

Polygot persistence

Aggregate-oriented database

Key-value database

Document database

Column family database

Graph database

Replication vs sharding

CAP Theorem

Eventual Consistency

Map-Reduce

MongoDB Interview Questions

MongoDB's type

MongoDB's characteristics

Alternative databases

Supported programming languages

Index

Aggregation Operations(aggregation pipeline)

Sharding

Replication

GridFS

ObjectId

Consistency

Cassandra Interview Questions

Cassandra data model

Tunable consistency (Quorum)

Difference between RDBMS and Cassandra

CAP Theorem

Cassandra query language

cqlsh

Compaction

Super column

Column family

Memtable

SSTable

Keyspace

Tombstone