Couchbase

# Architecture and Administration Basics

## Architecture

# 1 | Buckets and Documents

# What is Couchbase Server?

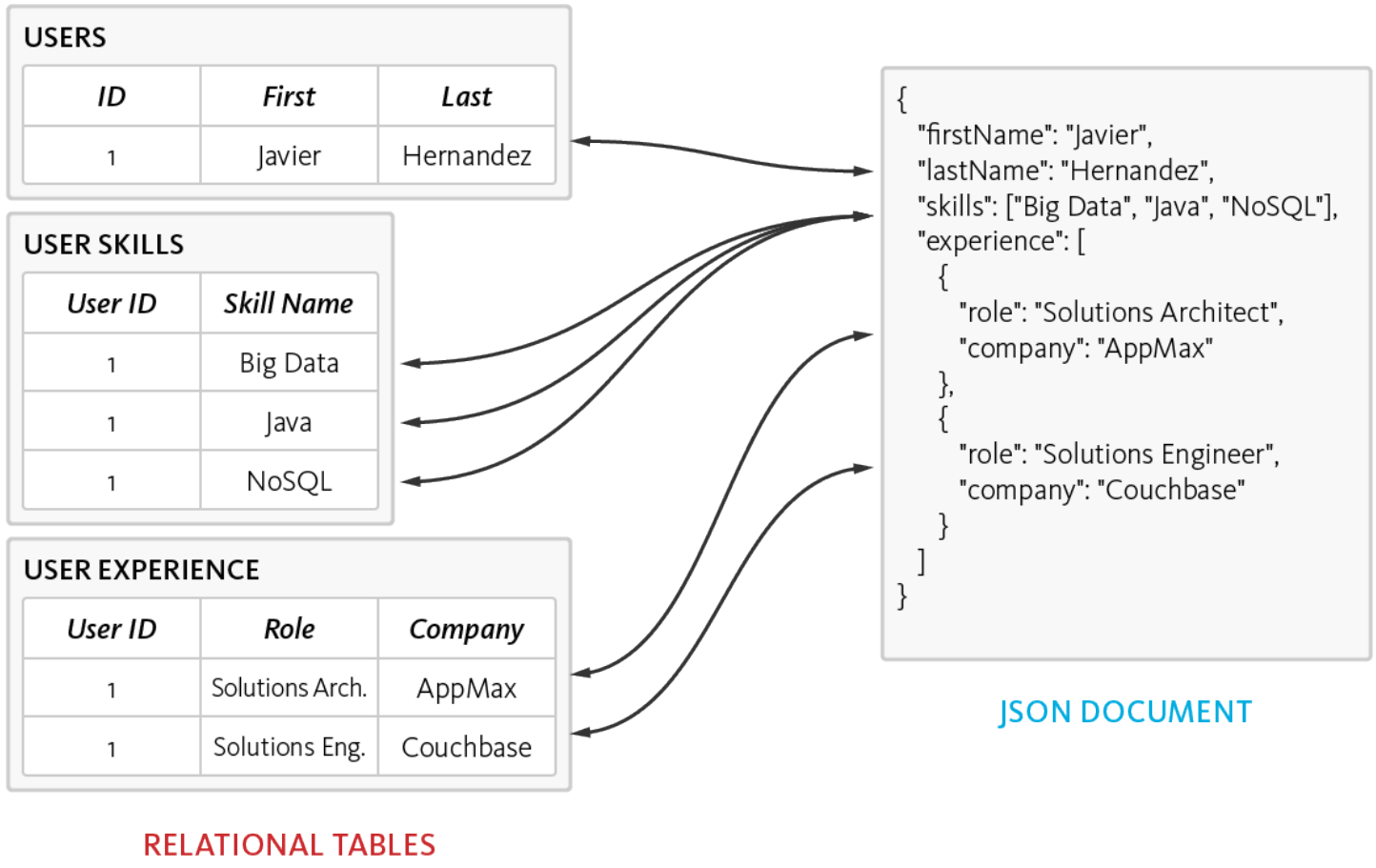
- Couchbase Server
  - Core database has key/value based orientation
  - Has various indexing and querying capabalities
  - Is geared for JSON
  - Runs on a networked cluster of nodes
  - Has caching and persistence layers
- Couchbase Server is best suited for fast-changing data items of relatively small size

# Couchbase as Key-Value Store vs. Document Store

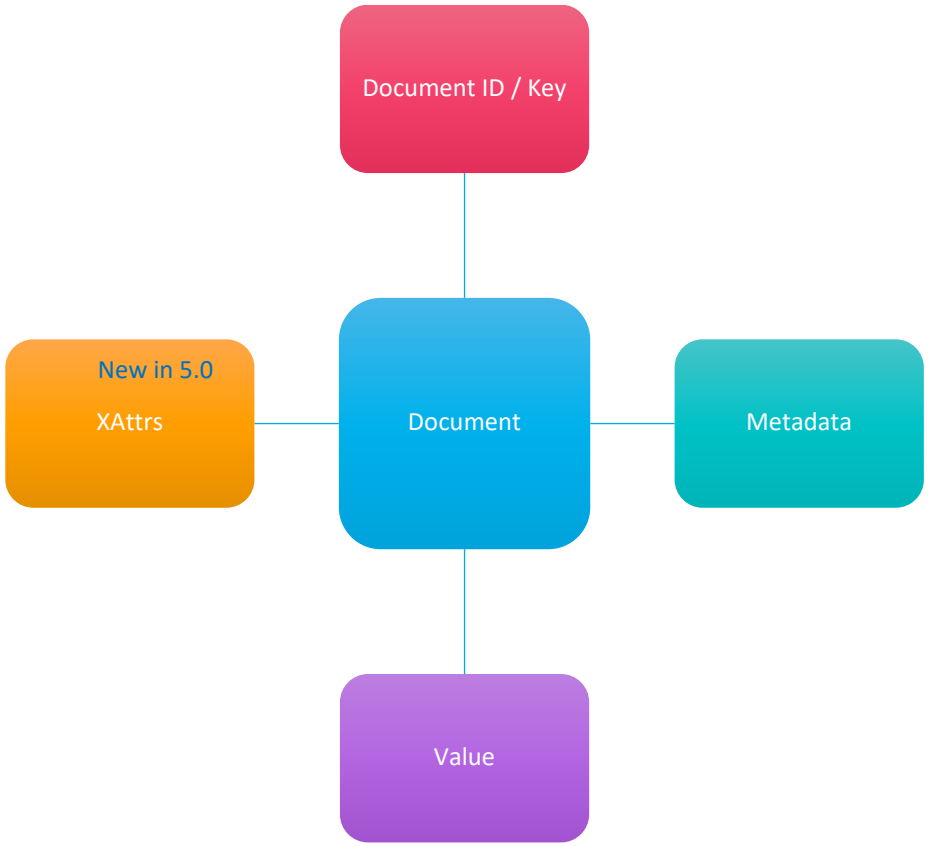
- Couchbase is capable of storing multiple data types.
  - Simple data types such as string, number, datetime, and boolean
  - Arbitrary binary data
  - For most of the simple data types, Couchbase offers a scalable, distributed data store that provides both key-based access as well as minimal operations on the values
- Document databases encapsulate stored data into “documents” that they can operate on
  - A document is simply an object that contains data in some specific format.
  - For example, a JSON document holds data encoded in the JSON format

# The Power of JSON

- A lightweight format to store structured data
- Both human readable and easy to process
- Denormalized single document, as opposed to normalizing data across multiple tables
- Dynamic schema makes it easy to add new values when needed



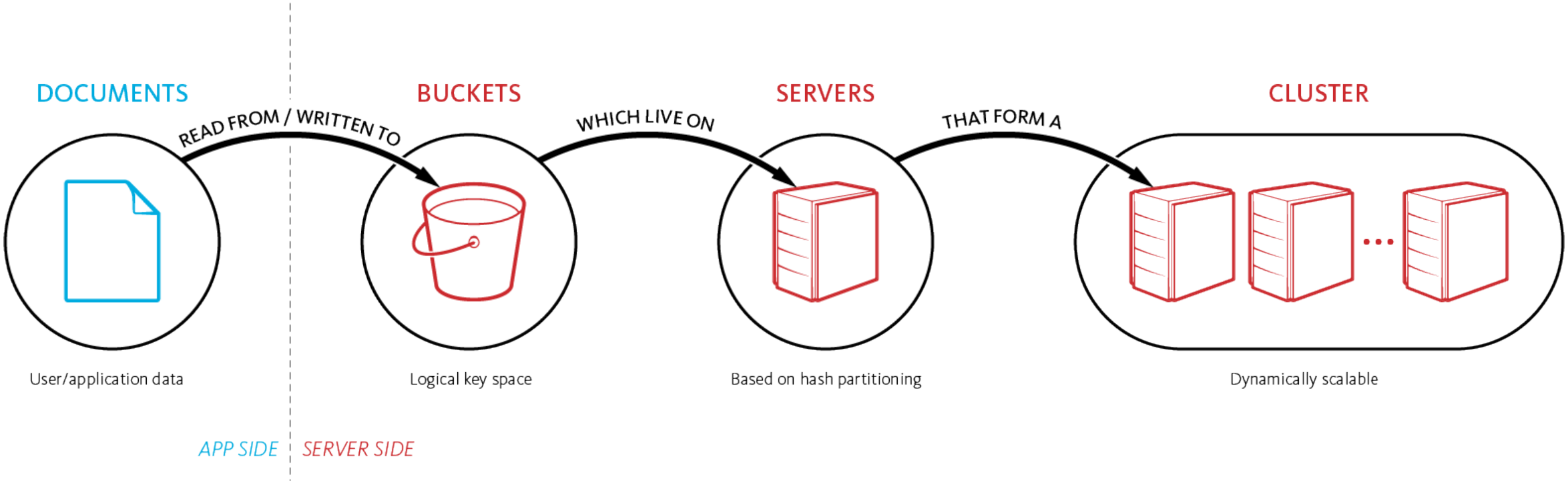
# Document Structure




- **Document ID / Key (Max 250 bytes):**
  - Must be unique / Lookup is extremely fast
  - Similar to primary keys in relational databases
  - Documents are partitioned based on the document ID
- **Value (Max 20 MB)**
  - JSON
  - Binary - integers, strings, booleans
  - Common binary values include serialized objects, compressed XML, compressed text, encrypted values
- **Metadata (Fixed 56 bytes)**
  - CAS Value (unique identifier for concurrency)
  - TTL
  - Flags (optional client library metadata)
  - Revision ID # New in 5.0
- **XAttr (Max 20 MB)**
  - Non-enumerable e**X**tended **Attr**ibutes

# Buckets

- The Couchbase server stores the data as key value pairs – Data lives in a “Bucket” as key/value pairs.
  - Create a named bucket on the cluster (logical structure)
  - Data managed for you as vBuckets
  - Data is spread around the cluster automatically

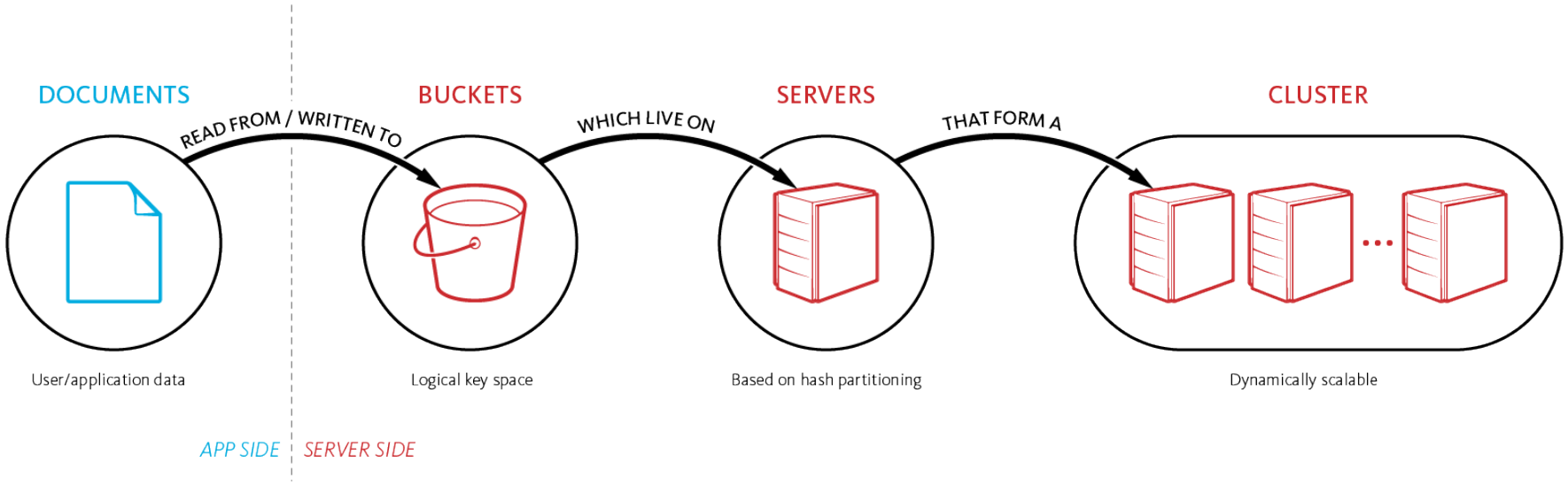


# Map RDBMS to Couchbase

| RDBMS | Couchbase |
| --- | --- |
| Database | Bucket |
| Table | Documents with common shapes |
| Row | JSON document |
| Fixed Schema | Flexible Schema |

# Buckets - When to use more than one ?

- When you need to treat or access the data differently
  - Different High Availability requirements (1,2 or 3 replicas)
  - Different performance / residency needs (how much data to cache)
  - Security / Multi-tenancy
  - Segregating Binary and JSON data – especially with view usage



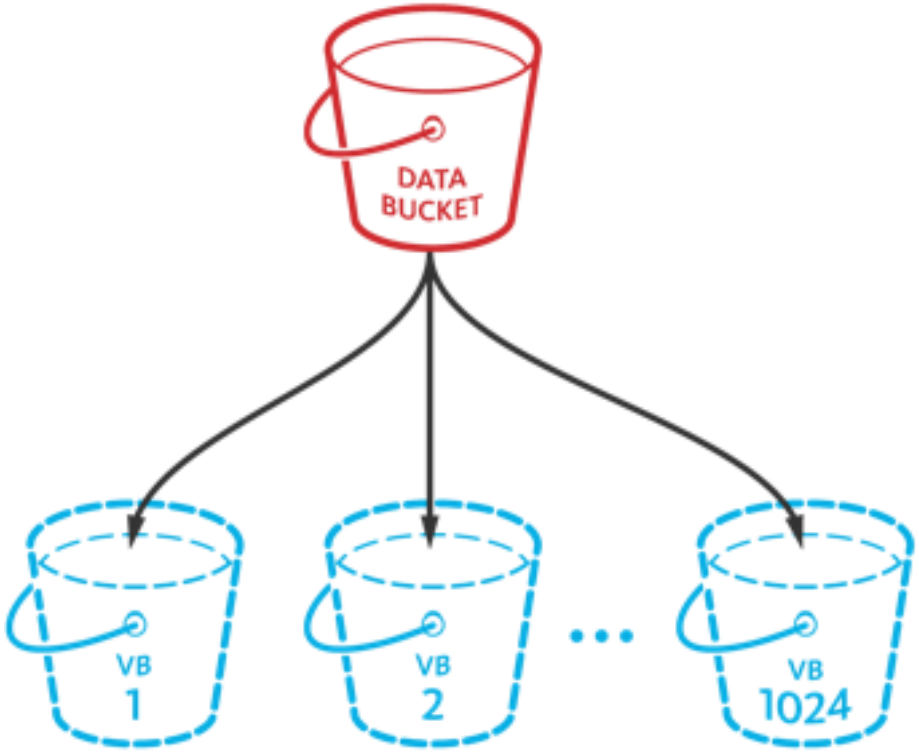
# Bucket Types

| | Memcached | Couchbase | Ephemeral<br>New in 5.0 |
|---|---|---|---|
| **Persistence** | X | | X |
| **Replication** | X | | |
| **Rebalance** | X | | |
| **XDCR** | X | | |
| **N1QL** | X | | |
| **Indexing** | X | | * |
| **Max Object Size** | 1MB | 20MB | 20MB |

* MOI, FTS only

# 2 | Data Sharding

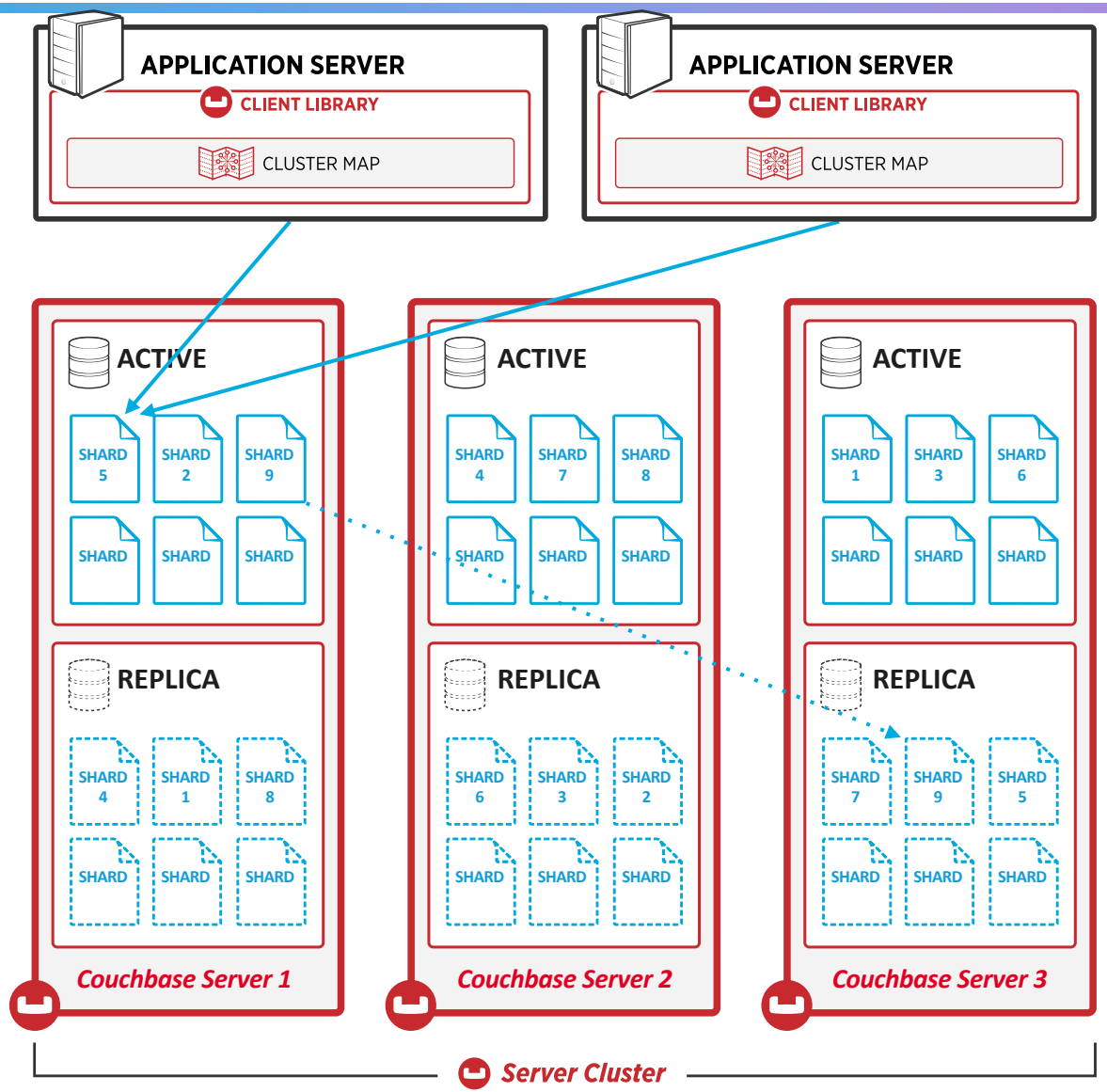
# Auto sharding – Bucket and vBuckets



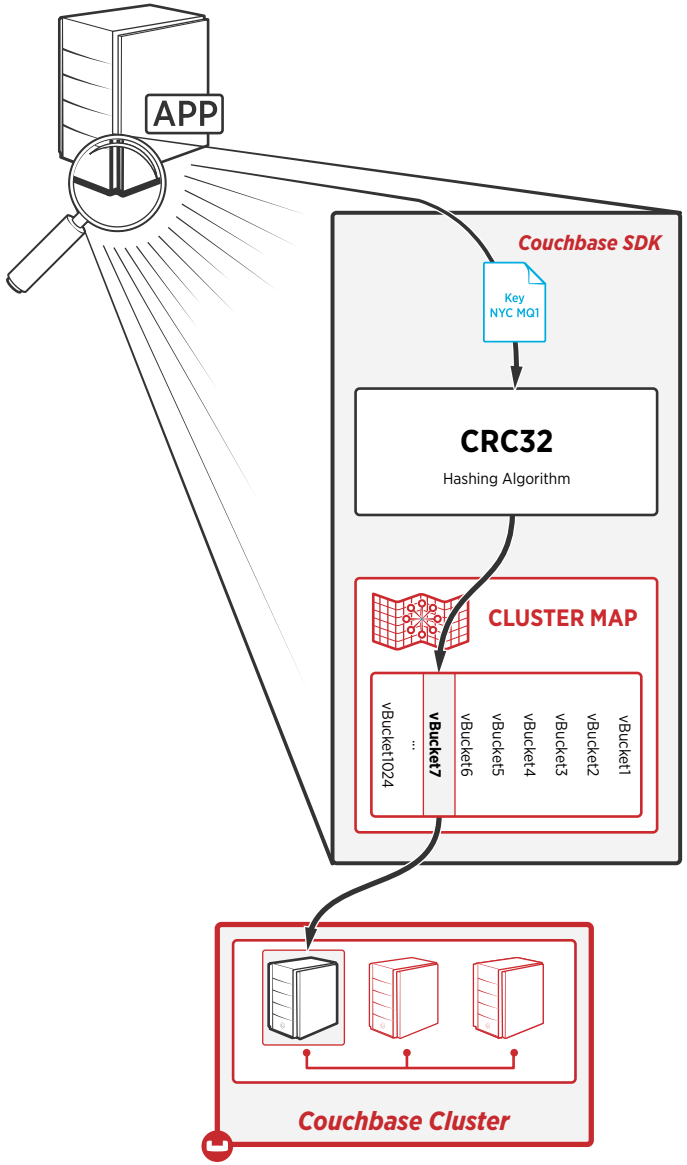

Virtual buckets

- Bucket
  - A bucket is a logical, unique key space
  - Multiple buckets can exist within a single cluster of nodes
- vBuckets
  - Each bucket has active and replica data sets (1, 2 or 3 **extra** copies)
  - Each data set has 1024 Virtual Buckets (vBuckets)
  - Each vBucket contains 1/1024th portion of the data set
  - vBuckets do not have a fixed physical server location
  - Mapping between the vBuckets and physical servers is called the cluster map
  - Document IDs (keys) always get hashed to the same vBucket (consistent hashing)
  - Couchbase SDK's lookup the vBucket -> server mapping

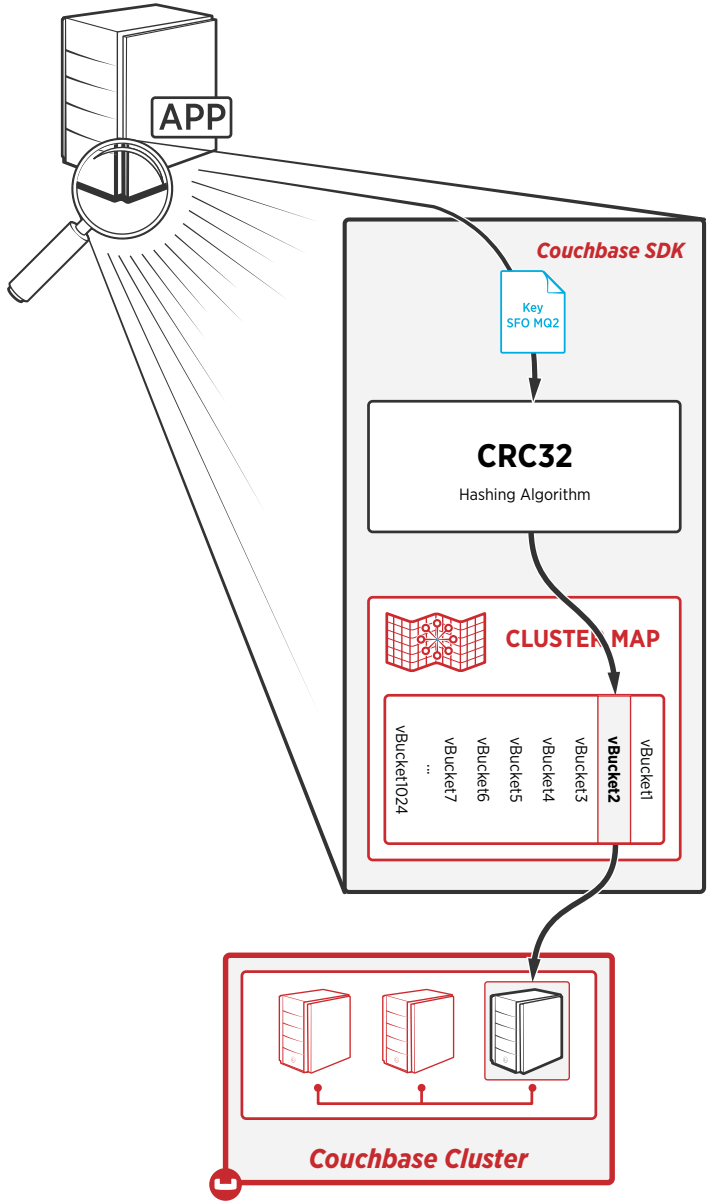
# Sharding Architecture



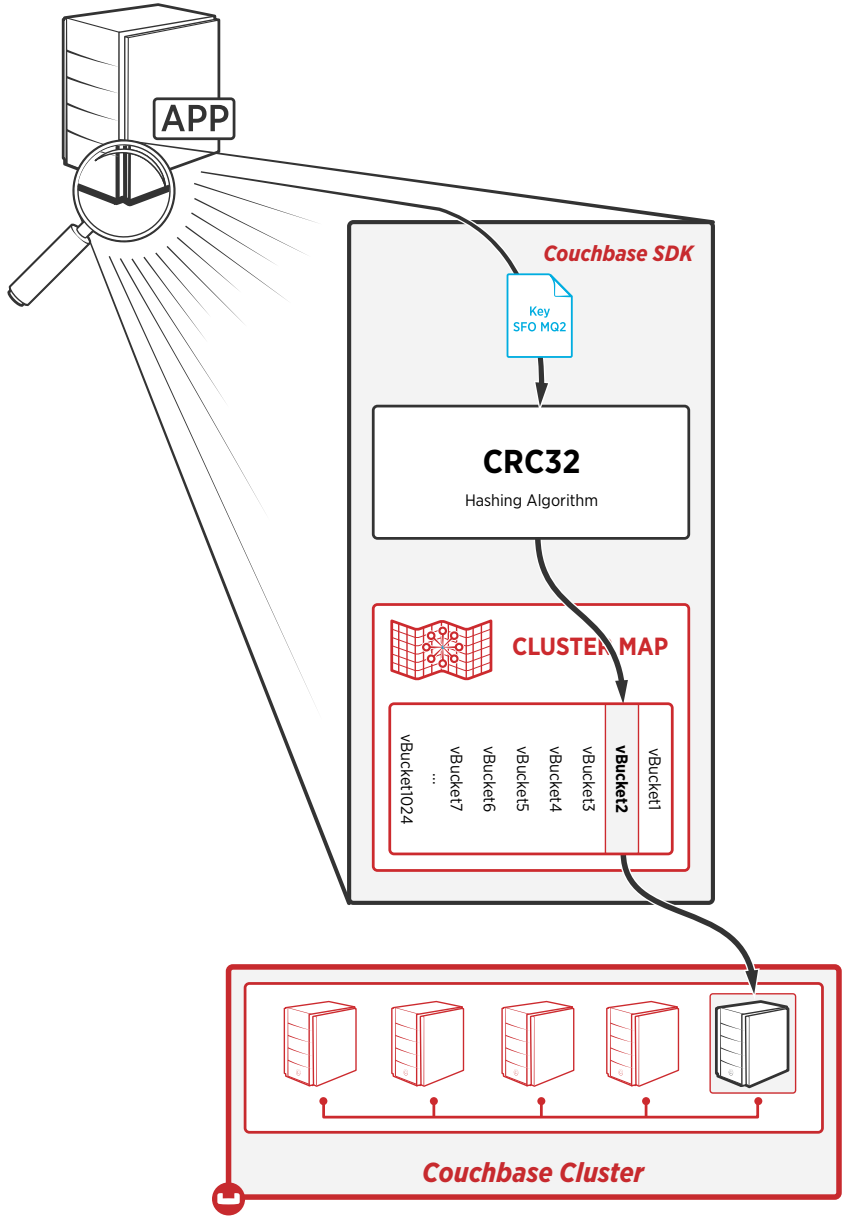
# Cluster Map


APP
Couchbase SDK
Key
NYC MQ1
CRC32
Hashing Algorithm
CLUSTER MAP
vBucket1
vBucket2
vBucket3
vBucket4
vBucket5
vBucket6
vBucket7
...
vBucket1024
Couchbase Cluster

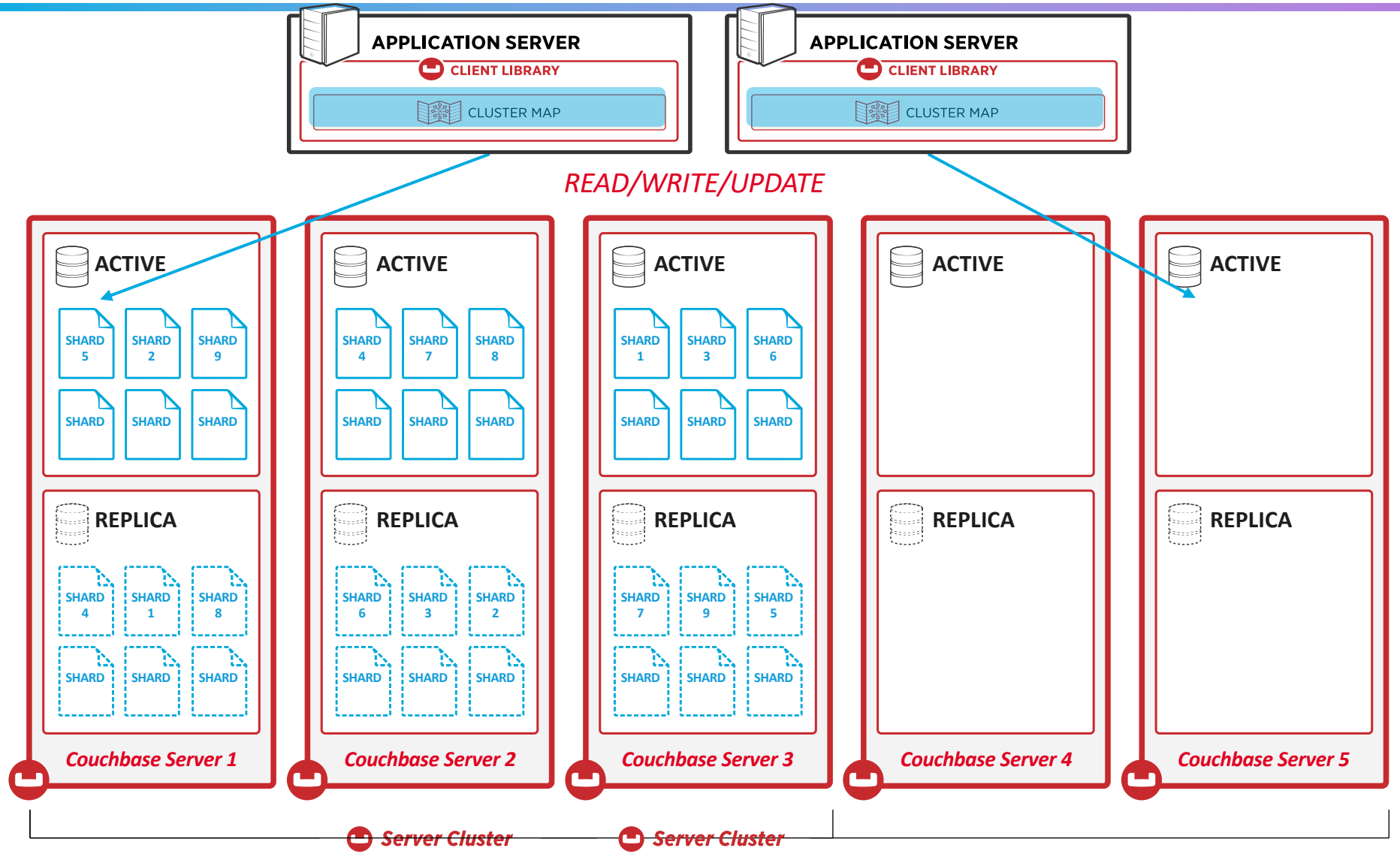
# Cluster Map


APP
Couchbase SDK
Key
SFO MQ2
CRC32
Hashing Algorithm
CLUSTER MAP
vBucket1
vBucket2
vBucket3
vBucket4
vBucket5
vBucket6
vBucket7
...
vBucket1024
Couchbase Cluster

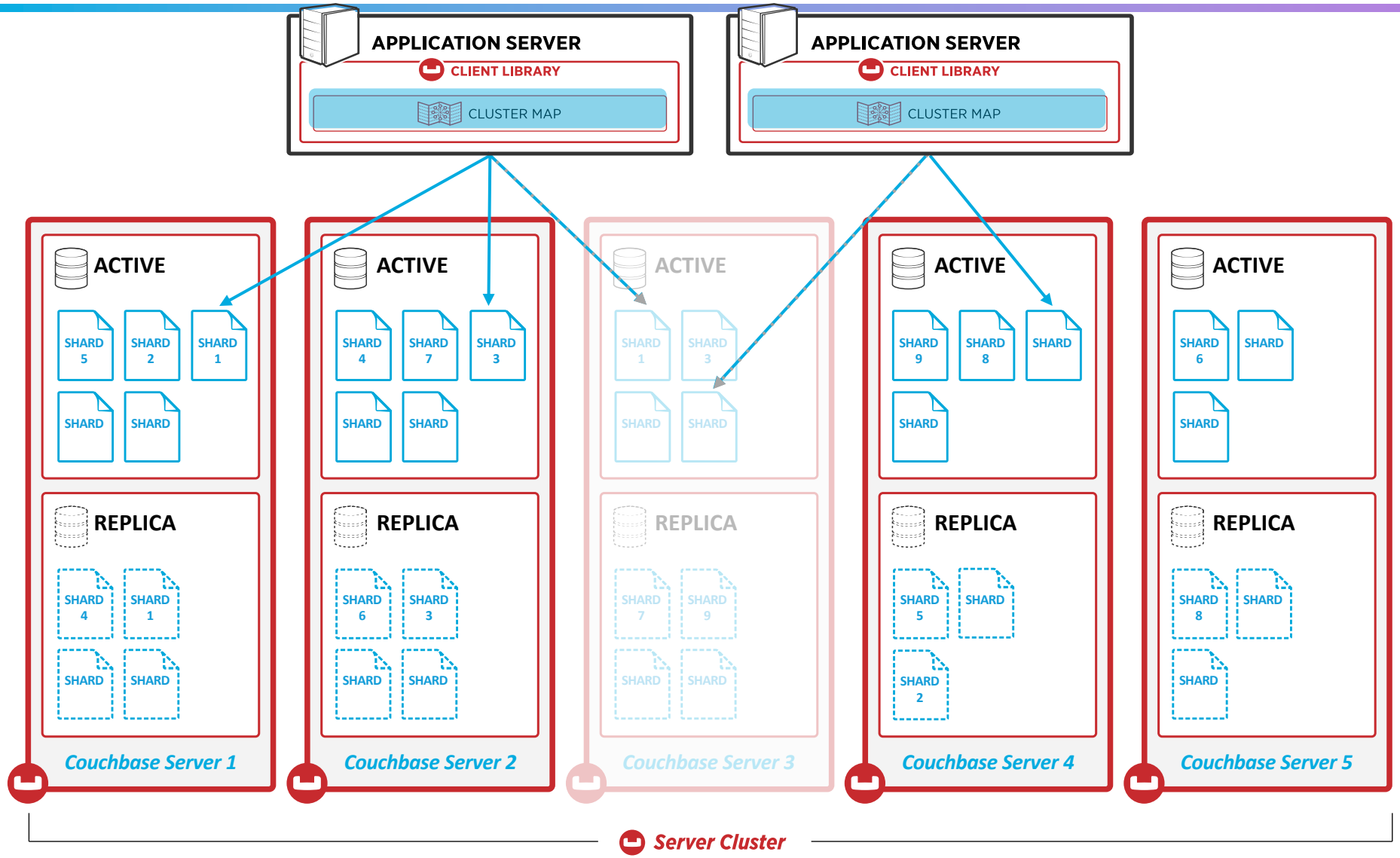
# Cluster Map – Addition of 2 Nodes



# Adding Nodes to Cluster

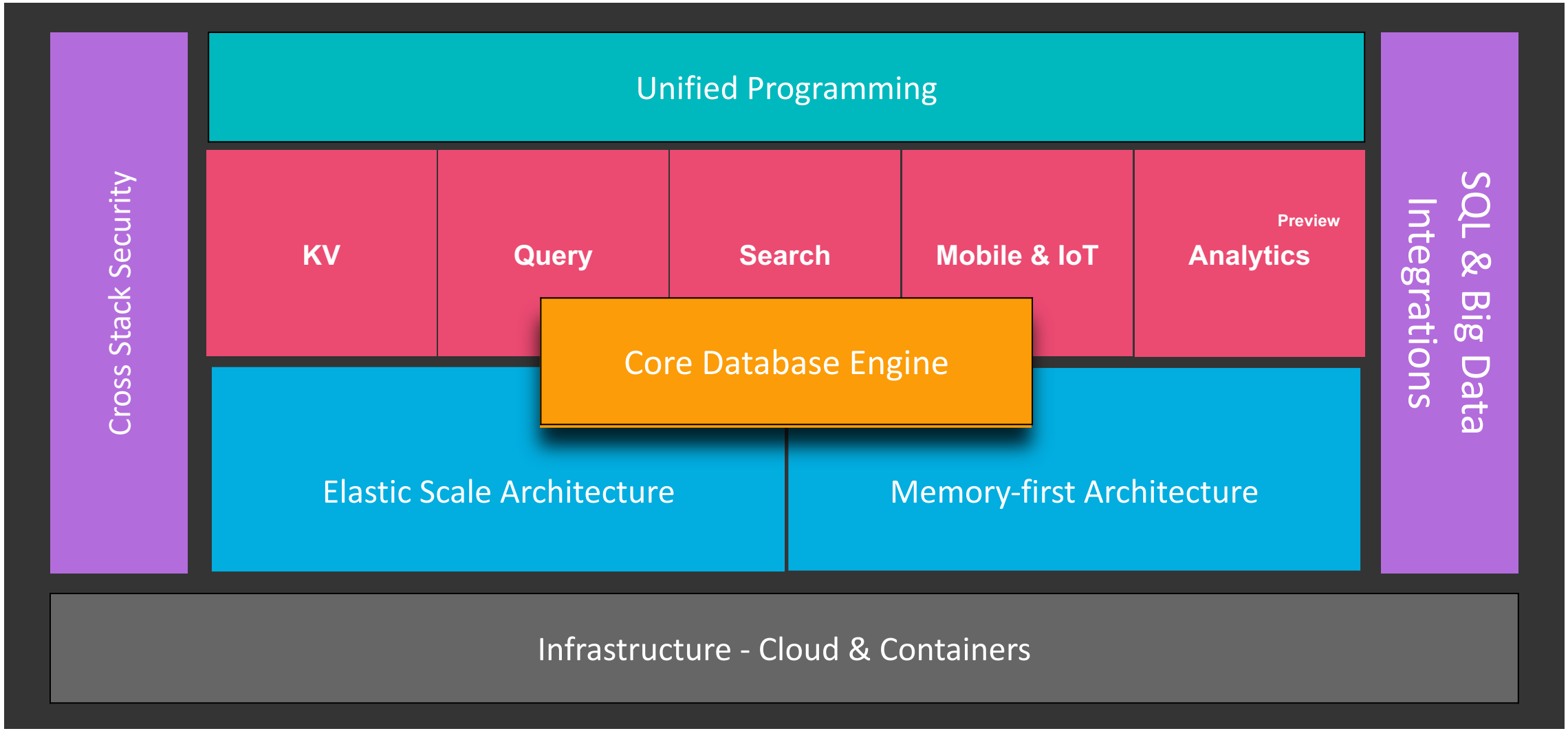


# Node Failover



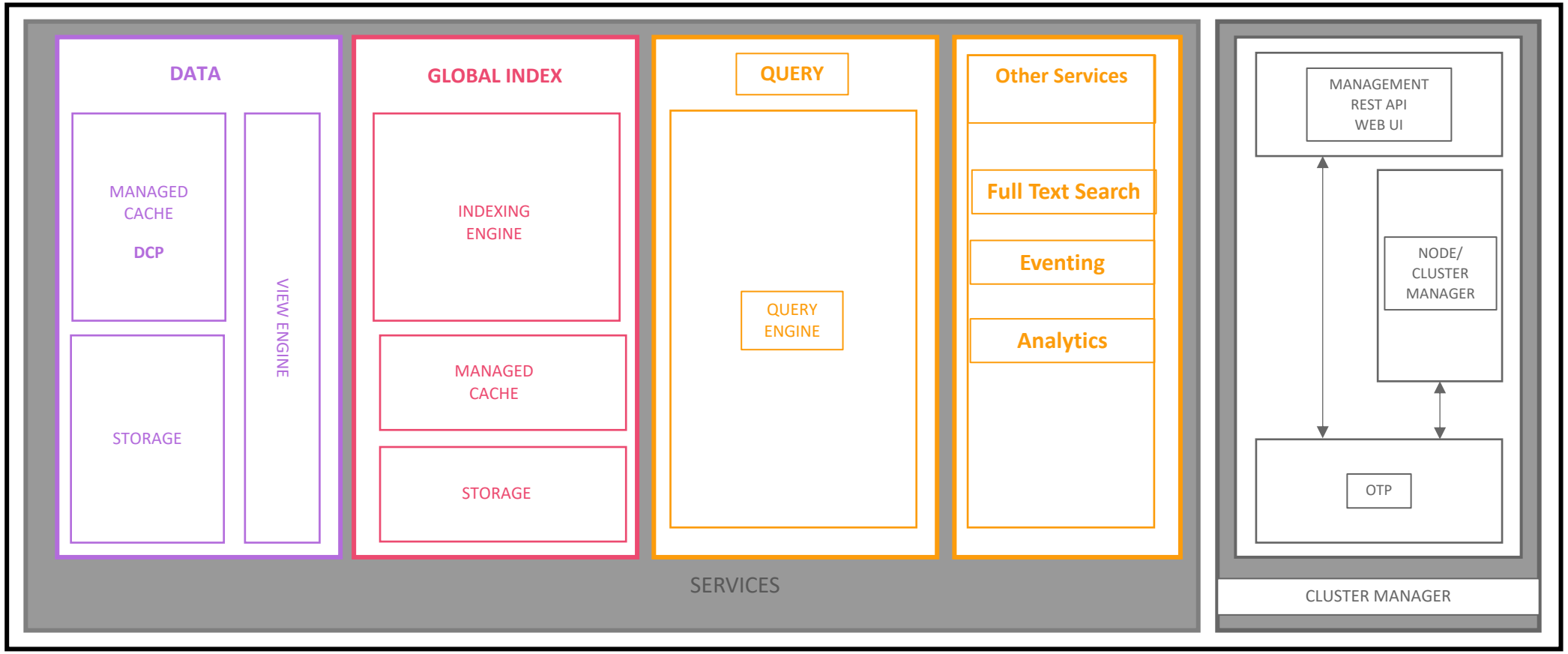
# 3 | Services

# Couchbase Data Platform



# Couchbase Node Architecture, Services

- Every Couchbase node can be configured to provide an arbitrary set of services
- Every node provides REST API, Web UI, participates in cluster management

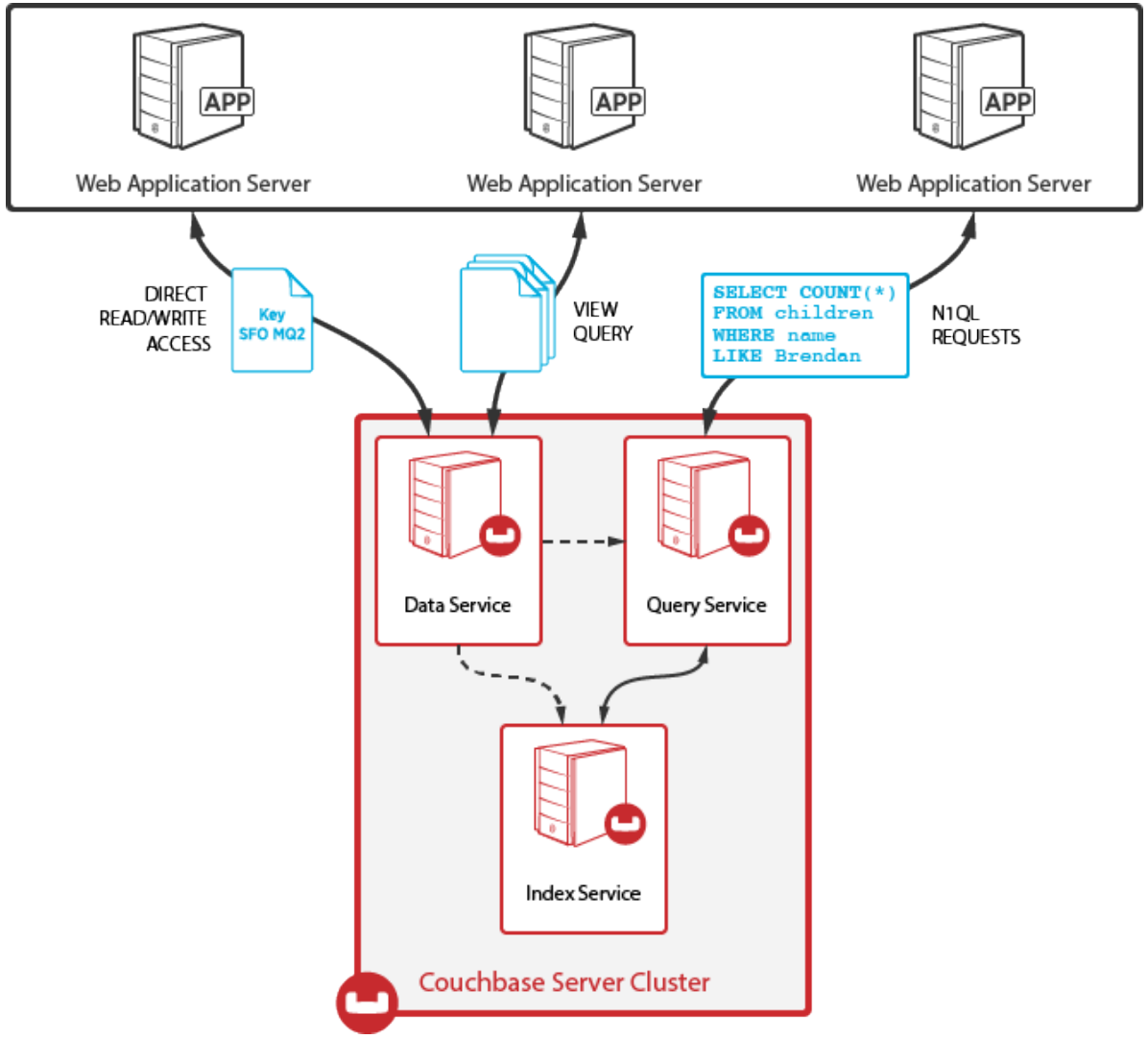



COUCHBASE SERVER NODE

# Couchbase Services

- Data Service:
  - Stores the data and provides key-value access to it
  - Supplies data to all other services
  - Takes care of persistence and replication
  - Indexing by map-reduce views (becoming obsolete)
- Services, providing different data access methods:
  - **Query Service:** Access data over N1QL queries
  - **Index Service:** Index data by various keys, used by query service
  - **FTS Service:** Enable searching documents by keywords
  - **Analytics Service:** Run analytics queries
  - **Eventing Service:** Trigger actions on changes to data
- Cluster Manager, Web UI, REST API

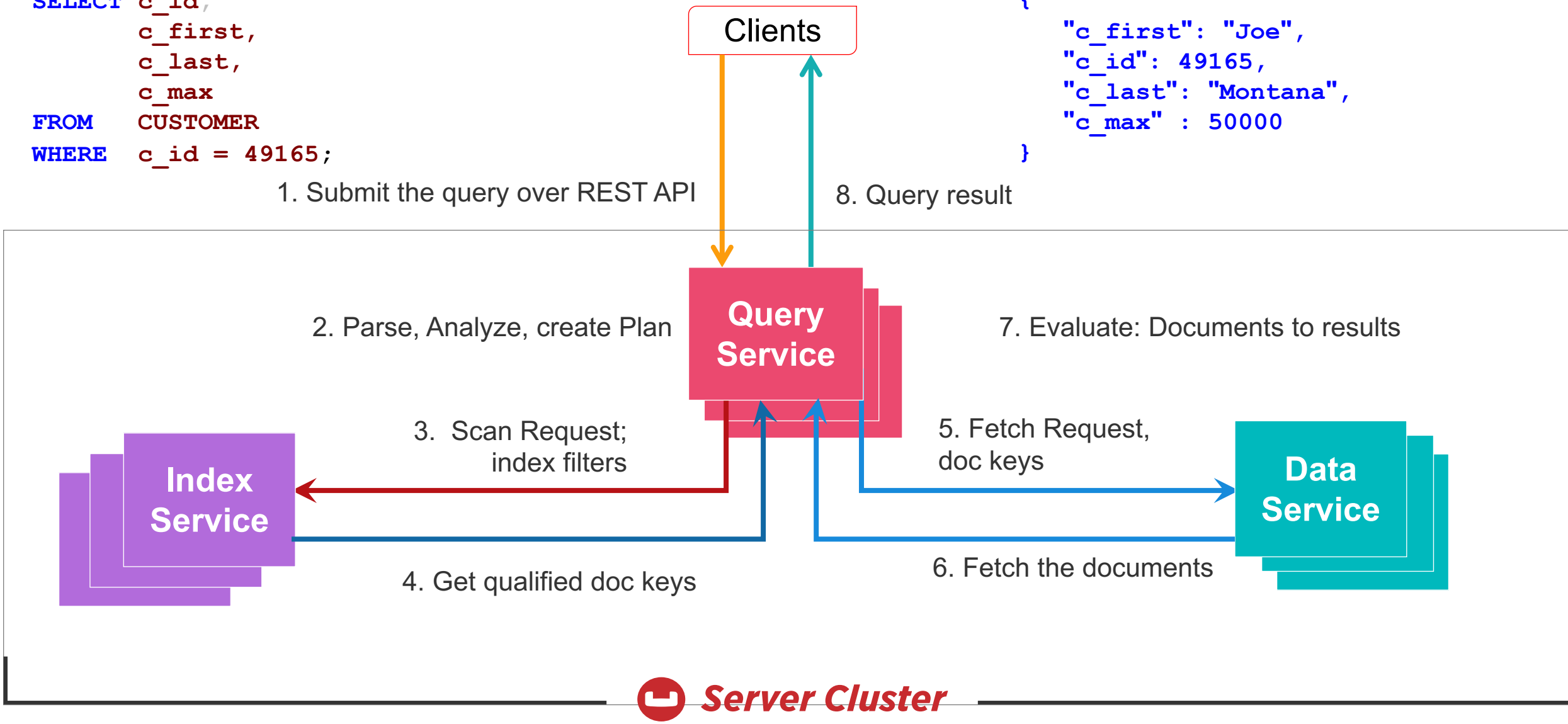
# Application to Database Interaction



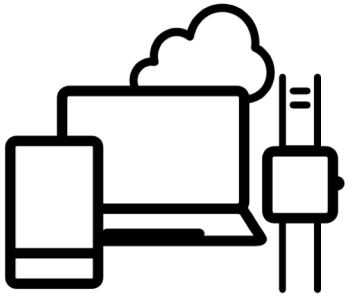
# Example: Interaction of Services for Execution of N1QL Queries

```
SELECT c_id,
       c_first,
       c_last,
       c_max
FROM   CUSTOMER
WHERE  c_id = 49165;
```

```
{
   "c_first": "Joe",
   "c_id": 49165,
   "c_last": "Montana",
   "c_max" : 50000
}
```

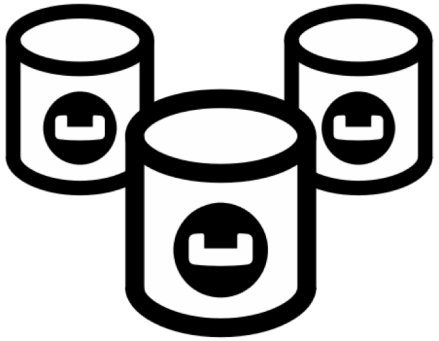


# Database Change Protocol



**Anything**
From
**Anywhere**

**Data Service**

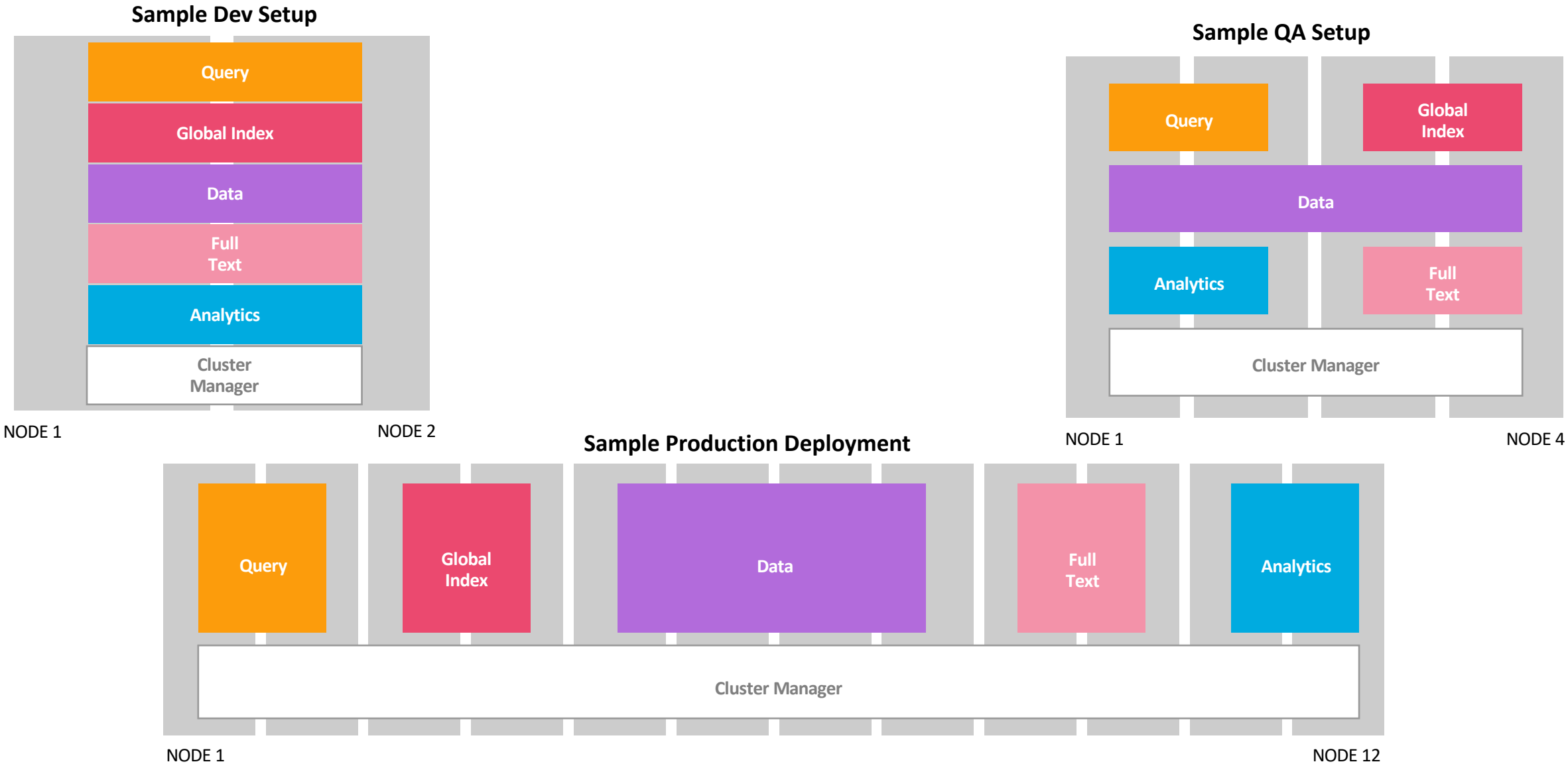


**DCP**

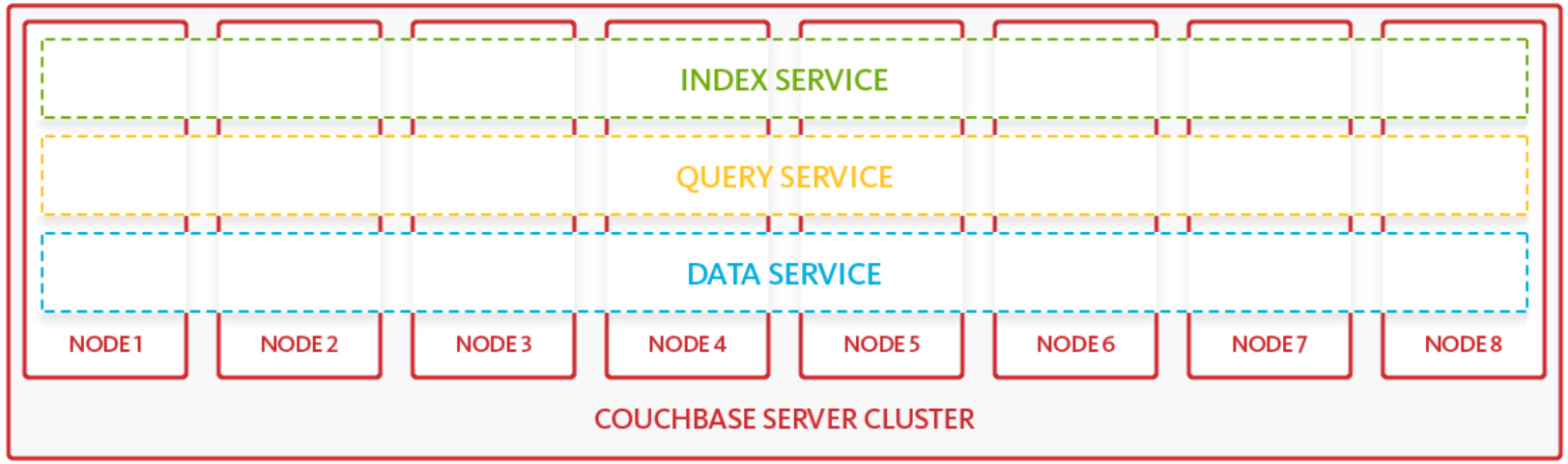
- Data Replicas
- XDCR
- GSI
- Views
- FTS
- Connectors
- Eventing

- High Performance / In-Memory
- De-Duplication
- Ordered, predictable and consistent
- Restartable

# Elastic Scaling Architecture



# Homogeneus Scaling




- Components processing core data operations (inserts, updates, deletes), index maintenance or executing queries compete and interfere with each other
- It is impossible to fine-tune each component because each of them has different demands on hardware resources
- While the core data operations can benefit greatly from scale-out with smaller commodity nodes, many low latency queries do not always benefit from wider fan-out

# Multi-Dimensional Scaling

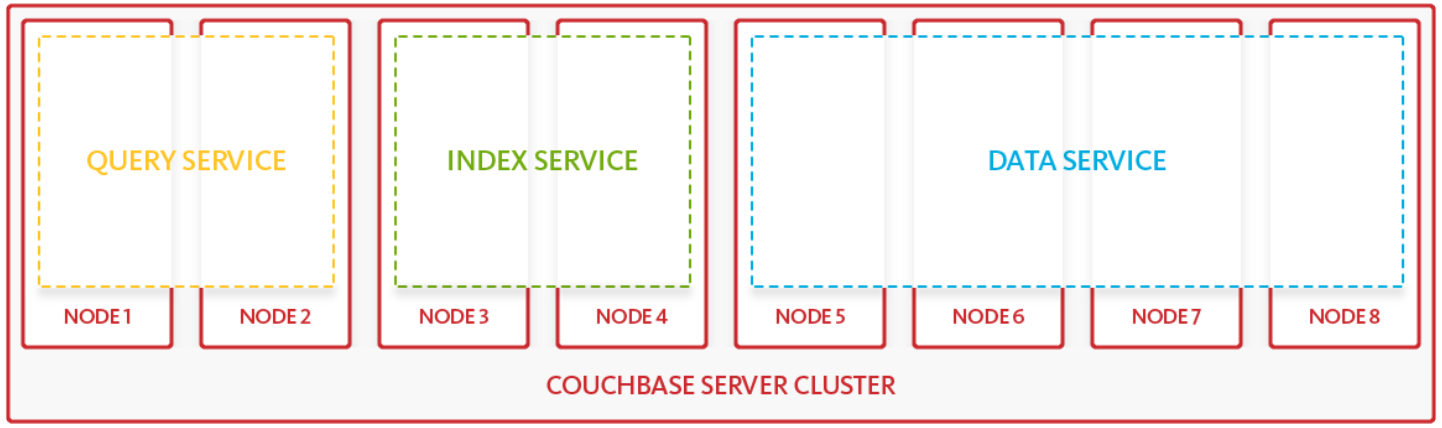
Independent Scalability for Best Computational Capacity — *per Service.*

***Heavier Indexing?***

Scale up or out
Index Service Nodes.

***More RAM for Query Processing?***

Scale up or out
Query Service Nodes.



# Multi-Dimensional Scaling

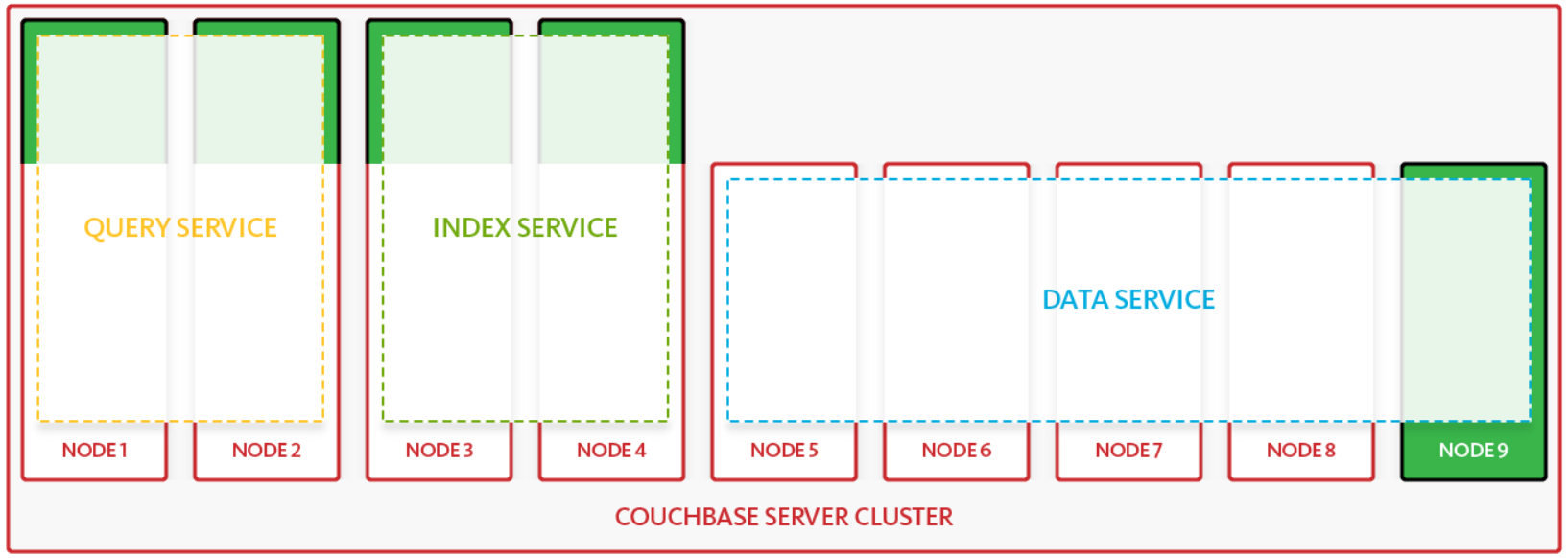
Independent Scalability for Best Computational Capacity — *per Service.*

***Heavier Indexing?***

Scale up or out
Index Service Nodes.

***More RAM for Query Processing?***

Scale up or out
Query Service Nodes.

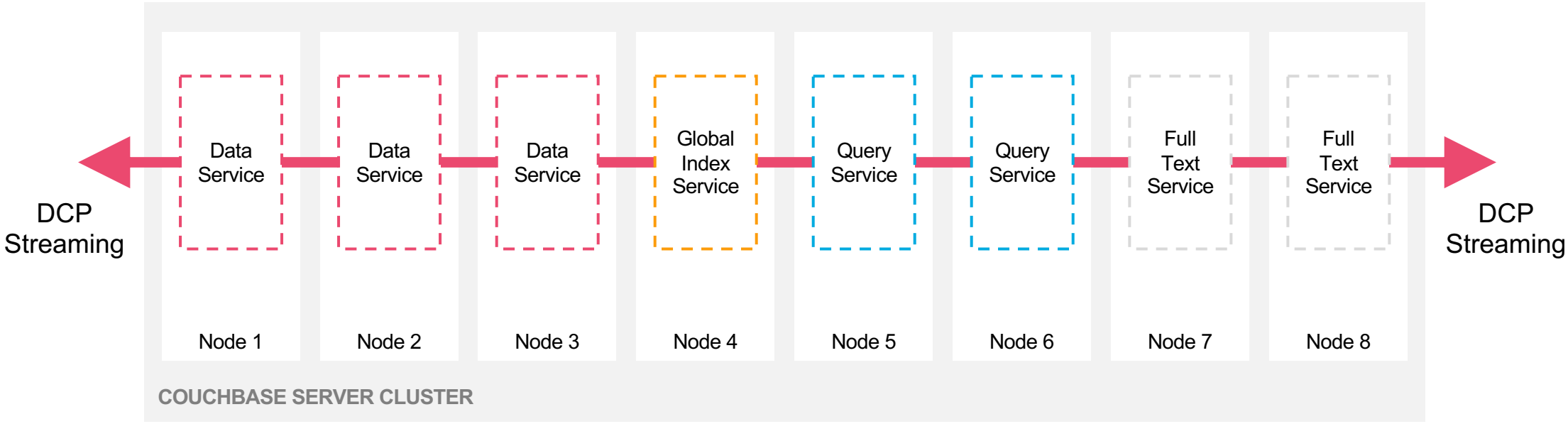



The same applies for other services

# 4 | Database Operations

# #1 Principle: Memory-first architecture

Data movement free from disk bottlenecks




- In-memory streaming of updates to all components
- In-memory cache
- Memory-only data buckets
- Memory-only indexes

**In-memory streaming of updates to all components**
**In-memory (cached) access to data and indexes**
**Memory-only indexes**

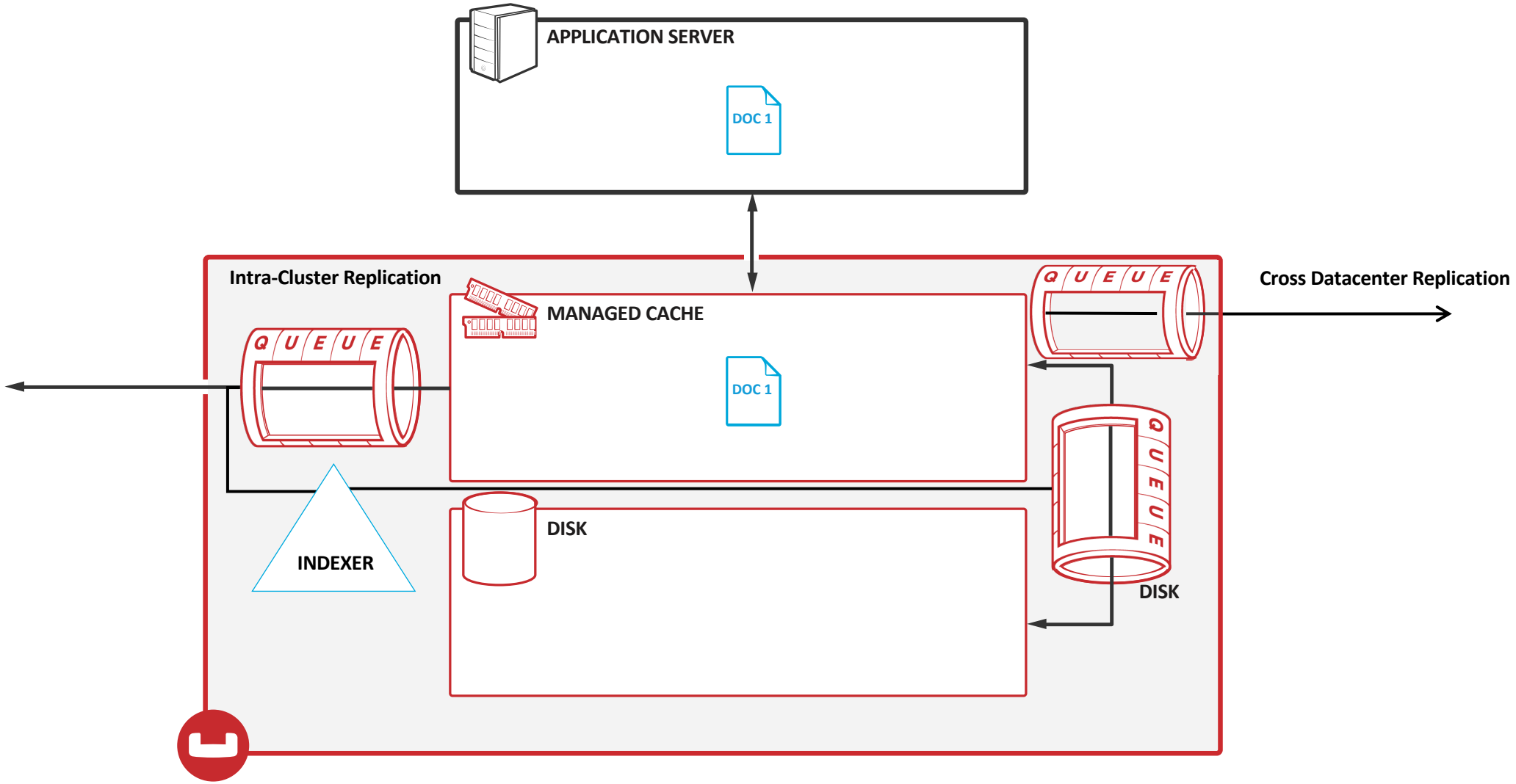
# Usage of RAM

- RAM is used as much as possible
  - Cached documents
  - Metadata and key information
  - Internal process data structures
- By default all document IDs and metadata are stored in RAM at all times
  - Also persisted to disk
- All data is transferred in and out through RAM
  - Client Libraries accesses go through RAM layer
- Data is persisted immediately to disk
  - Queued if necessary
  - Checkpoints used for de-duplication and synchronization off-node
- Replication is RAM-to-RAM

# #2 Principle: Asynchronous approach to everything

- Persistence
- Intra-cluster Replication
- Inter-cluster Replication
- Global secondary Indexing updates
- Full-Text Search update
- Analytics service updates

# Asynchronous Approach to Everything

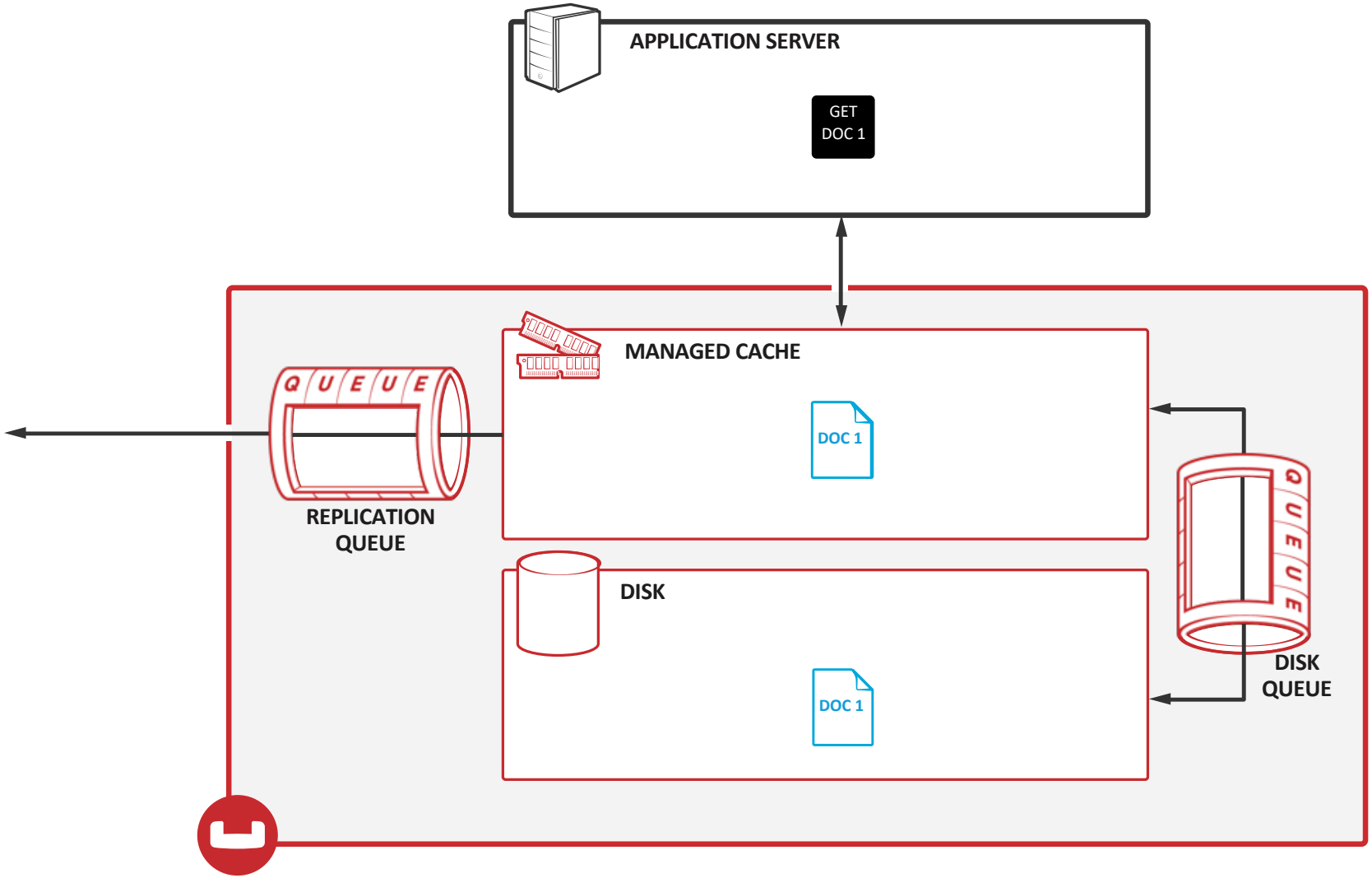


# Cache Management

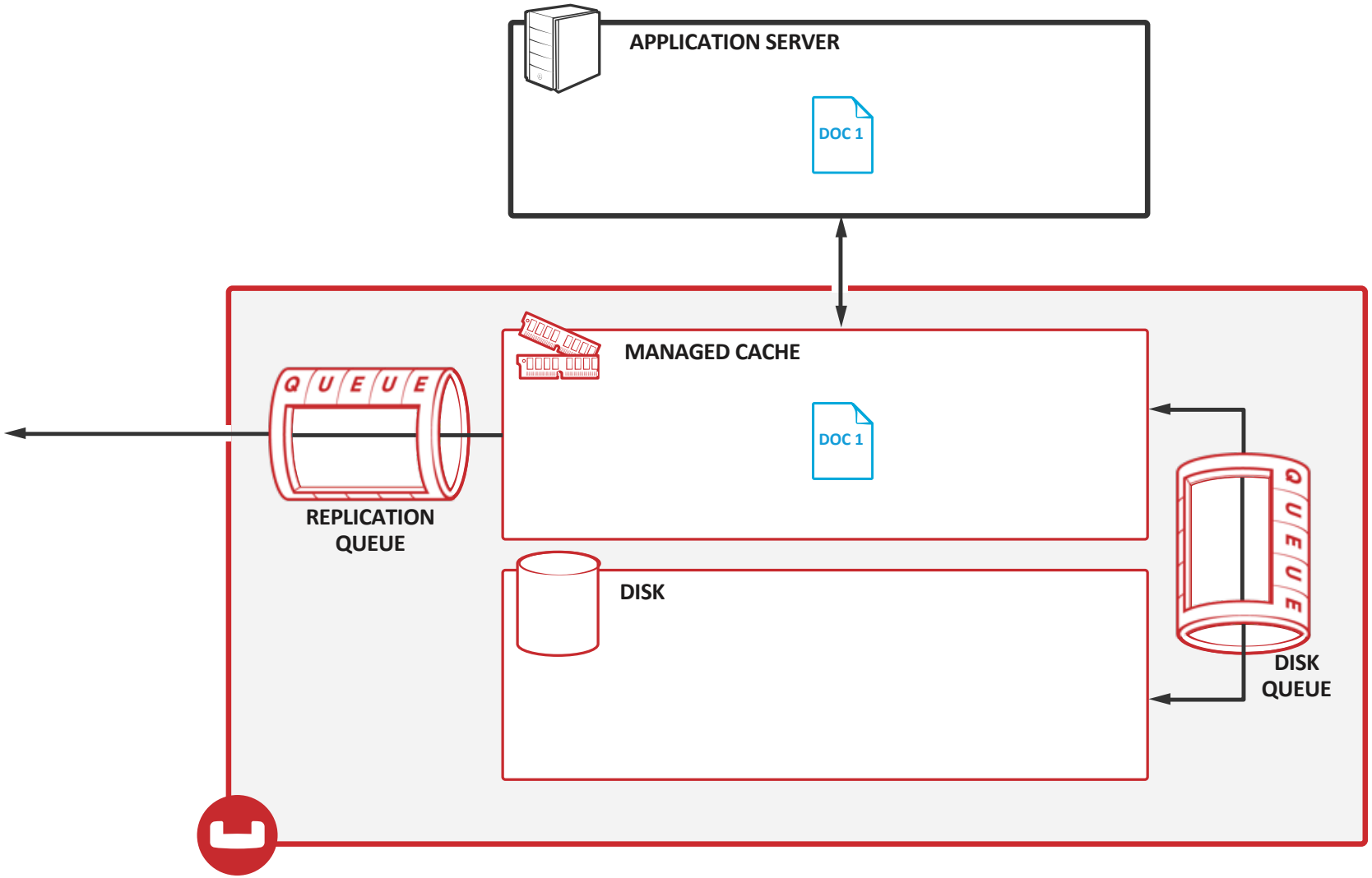
- As RAM fills up, data will be ejected to disk:
  - Only data already persisted can be ejected
  - Ejection happens in background, not related to traffic
  - Only the “values” can be ejected; metadata remains in RAM at all times

- NRU (Not Recently Used) Algorithm used for ejection
  - Client reads or writes update an item NRU attribute
  - Daily process scrapes and resets frequently used items
  - Resulting list of items is used to determine items to be ejected when needed

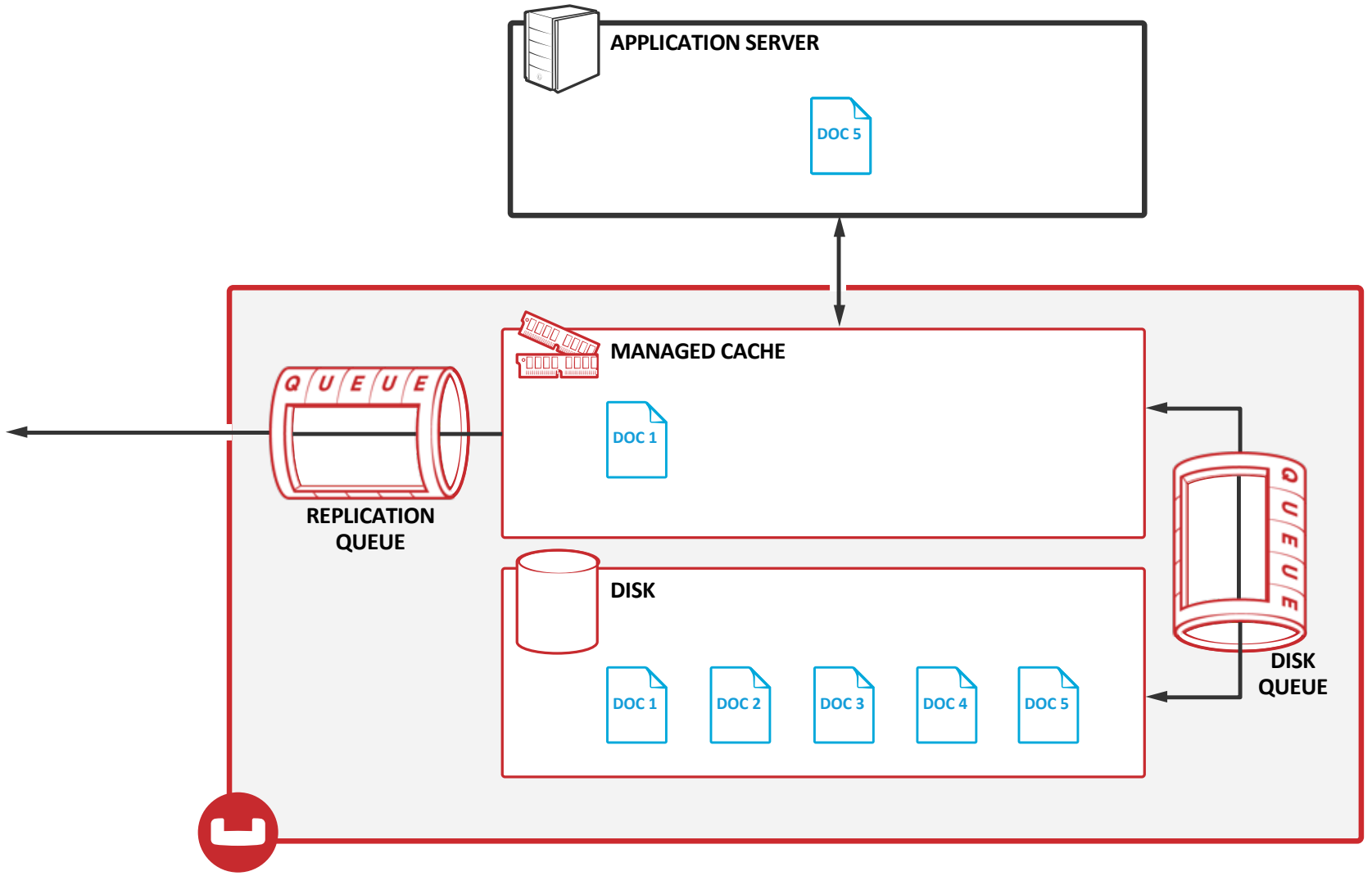
# Couchbase Read Operation



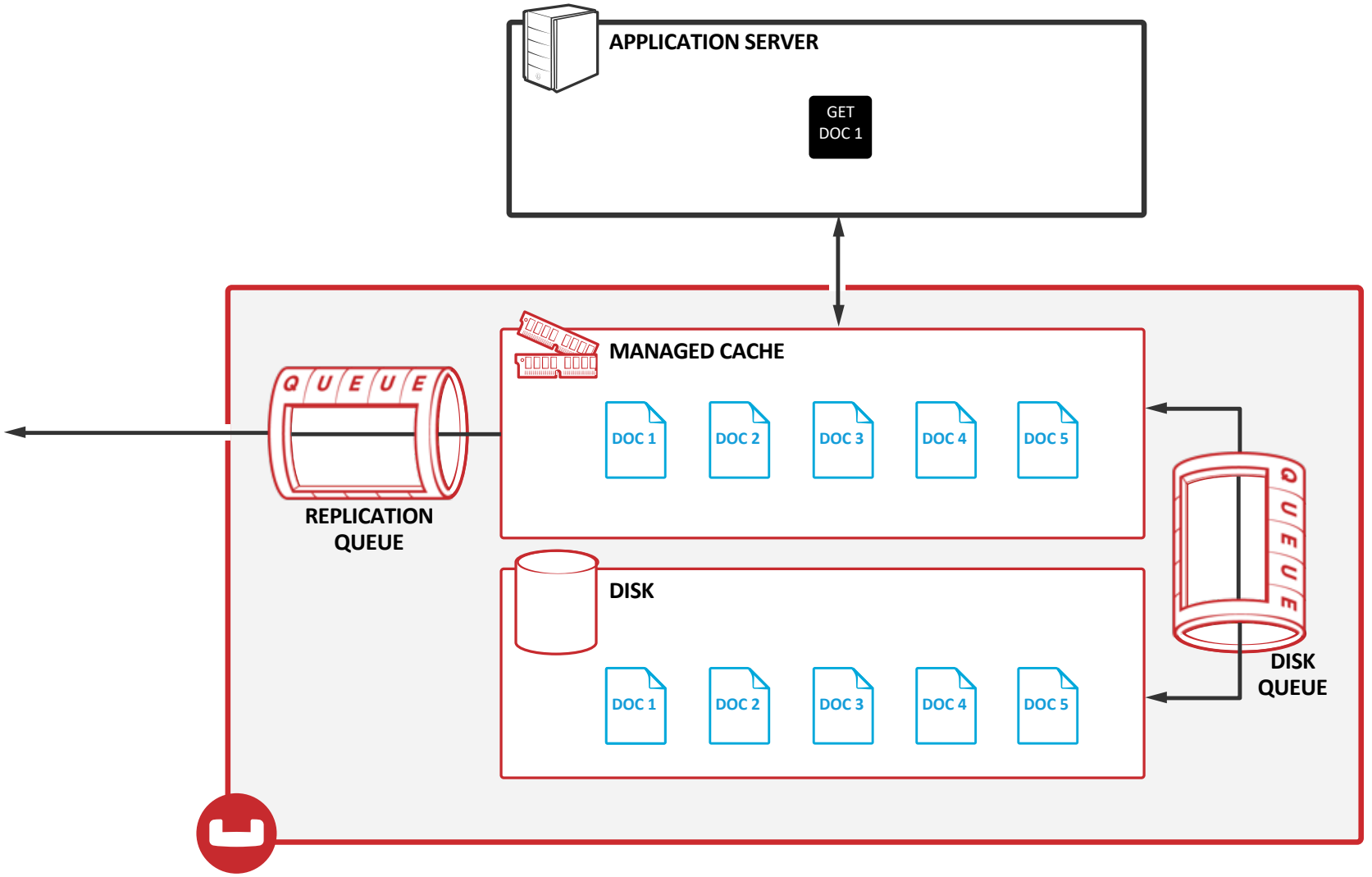
# Write Operation



# Cache Ejection

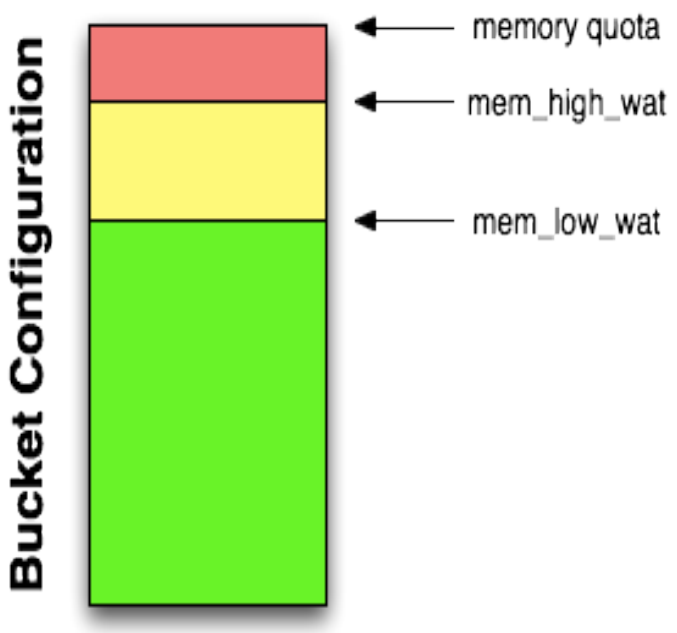


# Cache Miss


APPLICATION SERVER
GET
DOC 1
MANAGED CACHE
DOC 1
DOC 2
DOC 3
DOC 4
DOC 5
QUEUE
REPLICATION
QUEUE
DISK
DOC 1
DOC 2
DOC 3
DOC 4
DOC 5
QUEUE
DISK
QUEUE

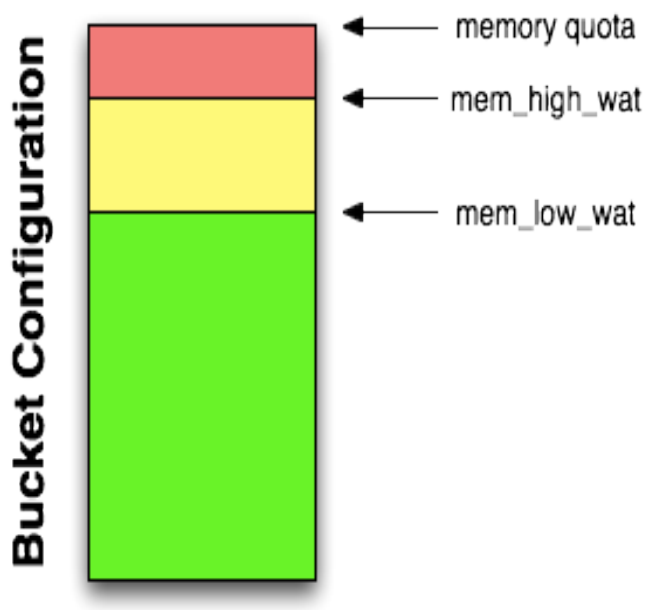
# Cache Management

- User defined quota is max memory usage
- High and Low watermarks are automatically set (%85 and %75 by default):
  - Watermarks can be raised/lowered manually if needed (not recommended)



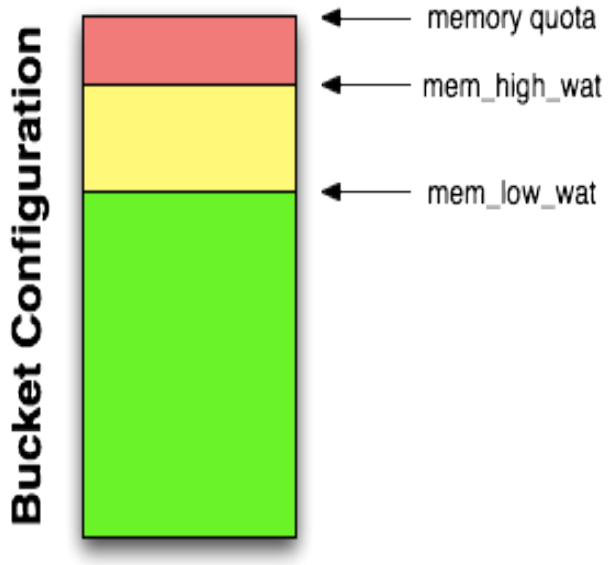
# Cache Management

- When memory usage passes:
  - Low watermark:
    - Nothing
  - High watermark: NRU used first, random afterwards
    - Ejection continues as needed until low watermark reached
  - Overhead is built-in to leave space for:
    - unexpected spikes
    - rebalancing
- Best practice is to keep “working set” (some percentage of overall dataset) below high watermark for best performance



# Out-of-Memory Errors

- If memory usage reaches 90% of quota, clients are told to temporarily back off (“temp_oom” )
  - Typically caused by too many/spike in writes
- If memory cannot be reclaimed, clients are told system is full (“oom”)
  - Typically caused by RAM being filled with metadata



# Working Set Management

- The high water mark and low water mark values can be tuned for performance
  - 85% high water mark default
  - 75% low water mark default
  - Configured as percentage or absolute RAM usage
- Balance of replica to active ejection after NRU can be set
  - 60/40 replica to active by default
- Expiry pager interval can be changed
  - 60 minutes by default

# Storage on Disk

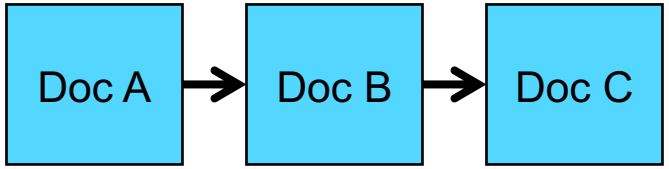
- *ALL* files are append-only b-trees (as of CB6.0):
- All modifications (writes/updates/deletes/index changes) go to end of file
- New b-tree header written to end of file after batch of changes
- Each vbucket (per-bucket) is separate file

# Compaction

- On-disk size increases compared to actual stored data (append-only)
- Compaction defragments data and index information
- Operates on a live bucket (no downtime)
- Both automatic and manual compaction available
- Compaction operates on a single server basis
  - Nodes operate independently
  - Compaction is per-vbucket, whole data set can be compacted incrementally

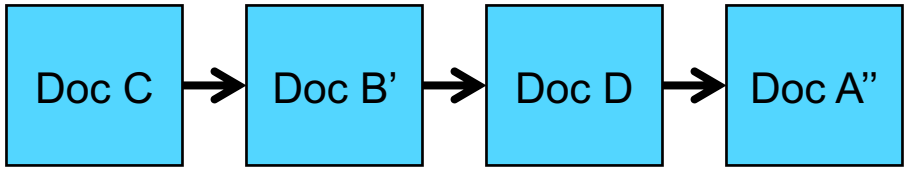
# Compaction

Initial file layout:




Update some data:


After compaction:



# Compaction

- Compaction creates a new file
  - Available disk space is checked (need twice current data size)
- Existing data is written to new file
  - Existing file is used until compaction has completed
  - Data file extension is numeric; new version has uses next increment (i.e. current data in 1.couch.12 is compacted to 1.couch.13)
- Once compaction is complete, old file disabled, new file enabled, old file deleted
- Compaction is both disk and CPU intensive
  - Configuration available to control time of day of automatic compaction

# Thank you

Couchbase