Couchbase Architecture

ANJU MUNOTH

Couchbase

- Open source, distributed, document-oriented, NoSQL database.
- Supports multiple data access patterns on top of a flexible JSON data model.
- Consolidates multiple layers into a single platform that would otherwise require separate solutions to work together.
- Provides the performance of a caching layer, the flexibility of a source of truth, and the reliability of a system of record, eliminating the need to manage data models and consistency between multiple systems, learn different languages and APIs, and manage independent technologies.

Features

- Flexible JSON data model
- Easy scalability
- Consistent high performance
- Mobile synchronization
- Always-on 24x365 characteristics
- Advanced security

Features

Develop with agility	Perform at any scale	Manage with ease
Flexible JSON data model supports continuous delivery. Make schema changes without downtime • Leverage common SQL query patterns for joins, aggregations, and more • Extract value using a broad set of key capabilities (mobile sync, full-text search, real-time analytics, etc.) • ACID transaction support • No hassle scale-out	• Memory- and networkcentric architecture, with an integrated cache delivering high throughput and submillisecond latency • Always-on, fault tolerant design • Consistent performance at any scale • Isolated and independent scaling of workloads, with no downtime or code changes	Global deployment with low write latency using active-active cross datacenter replication • Infrastructure agnostic support across physical, virtual, cloud and containerized environments • Microservices architecture with built-in auto-sharding, replication, and failover • Full-stack security with end-to-end encryption and rolebased access control

Develop with agility

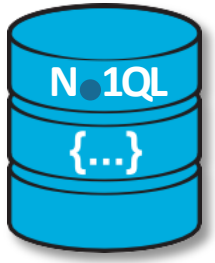
- Flexible JSON data model supports continuous delivery. Make schema changes without downtime
- Leverage common SQL query patterns for joins, aggregations, and more
- Extract value using a broad set of key capabilities (mobile sync, full-text search, real-time analytics, etc.)
- ACID transaction support
- No hassle scale-out

Perform at any scale

- Memory- and networkcentric architecture, with an integrated cache delivering high throughput and submillisecond latency
- Always-on, fault tolerant design
- Consistent performance at any scale
- Isolated and independent scaling of workloads, with no downtime or code changes

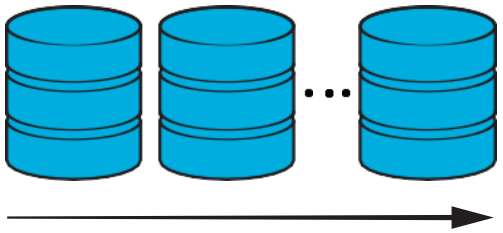
Key Capabilities

Combines the flexibility of JSON, the power of SQL and the scale of NoSQL



Develop with Agility

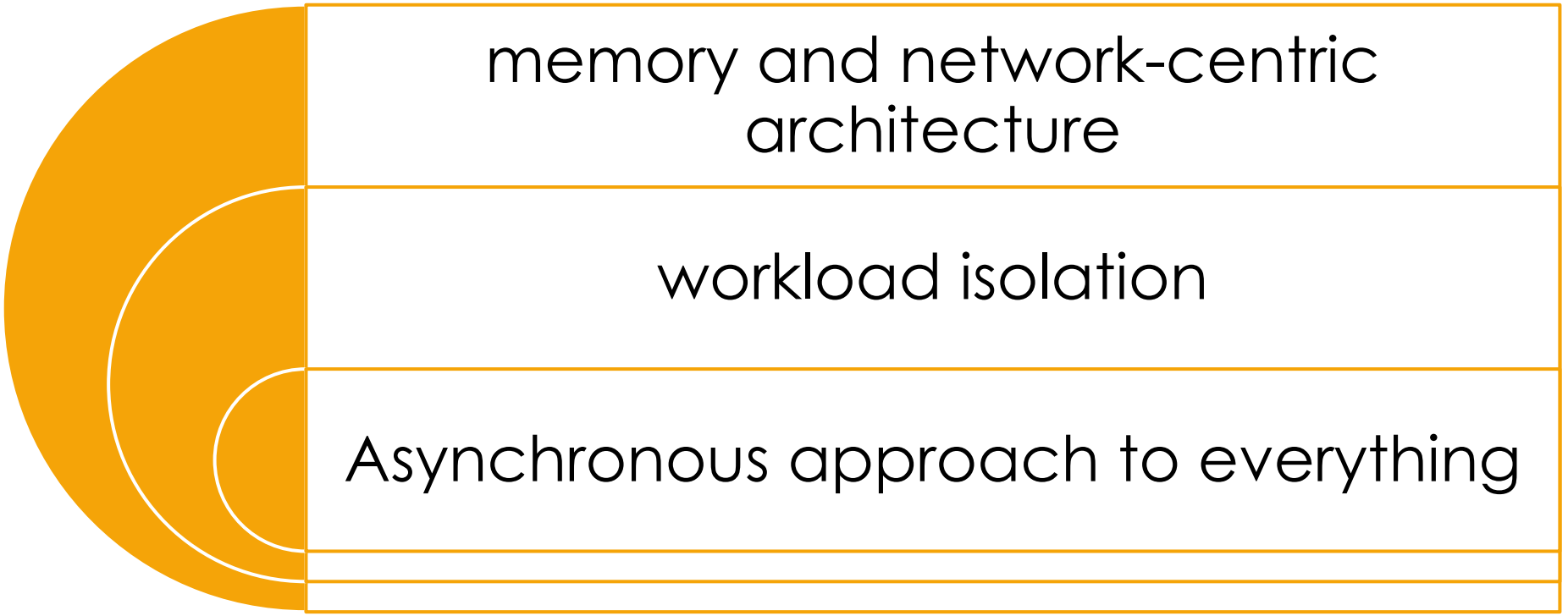
- Multiple data models
- N1QL - SQL-Like query language
- Multiple indexes
- Languages, ODBC /JDBC drivers and frameworks you already know



Operate at Any Scale

- Push-button scalability
- Consistent high-performance
- Always on 24x7 with HA - DR
- Easy Administration with Web UI, Rest API and CLI

Core design principles



Memory and network-centric architecture

The most used data and indexes are transparently maintained in memory for fast reads.

Writes are performed in memory and replicated or persisted synchronously or asynchronously.

Internal Database Change Protocol (DCP) streams data mutations from memory to memory at network speed to support replication, indexing and mobile synchronization.

Workload isolation

All databases perform different tasks in support of an application.

These include persisting, indexing, querying, and searching data.

Each of these workloads has slightly different performance and resource requirements.

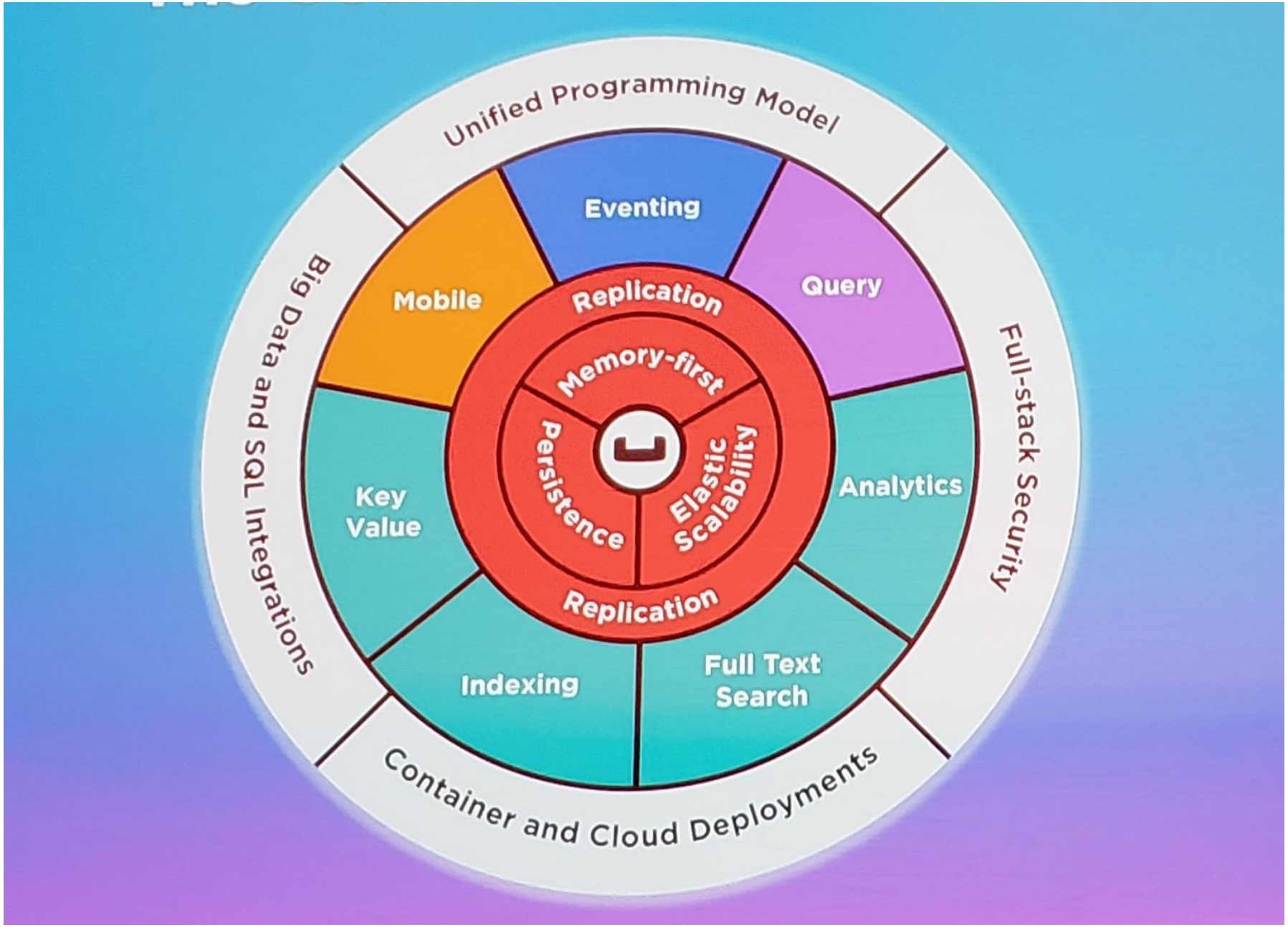
Multi-Dimensional Scaling (MDS) isolates these workloads from one another at both a process and a node level.

MDS allows these workloads to be scaled independently from one another and their resources to be optimized as necessary.

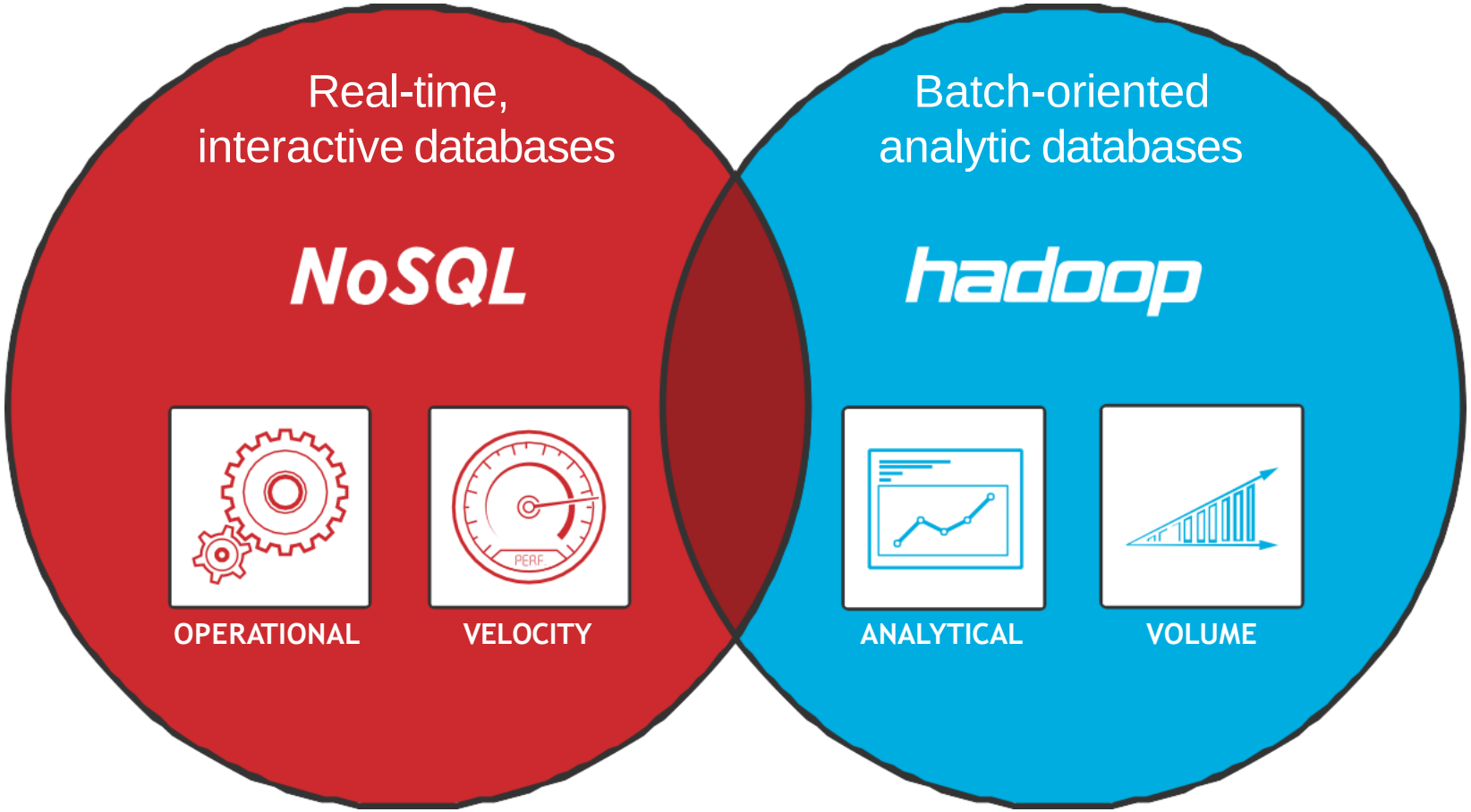
Couchbase manages the topology, process management, statistics gathering, high availability, and data movement between these services transparently

Asynchronous approach to everything

- Traditional databases increase latency and block application operations while running synchronous operations, for example, persisting data to disk or maintaining indexes.
- Couchbase allows write operations to happen at memory and network speeds while asynchronously processing replication, persistence and index management.
- Spikes in write operations don't block read or query operations, while background processes will persist data as fast as possible without slowing down the rest of the system.
- Durability and consistency options are available allowing the developer to decide when and where to increase latency in exchange for durability and consistency.


Unified Programming Model
Big Data and SQL Integrations
Full-stack Security
Container and Cloud Deployments
Eventing
Mobile
Query
Key Value
Analytics
Indexing
Full Text Search
Replication
Memory-first
Persistence
Elastic Scalability
Replication

Big Data = Operational + Analytic (NoSQL + Hadoop)



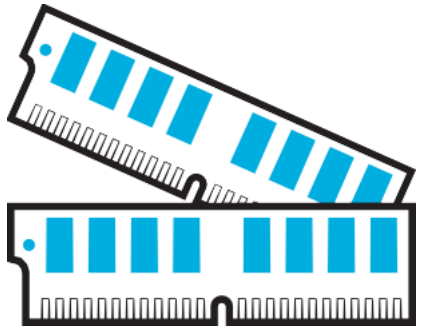
- Online
- Web/Mobile/IoT apps
- Millions of customers/ consumers

- Offline, batch-oriented
- Analytics apps
- Hundreds of business analysts

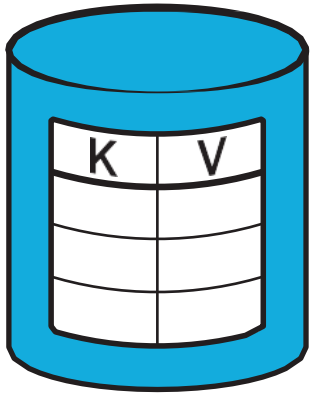
©2015 Couchbase Inc.

Couchbase provides a complete Data Management solution

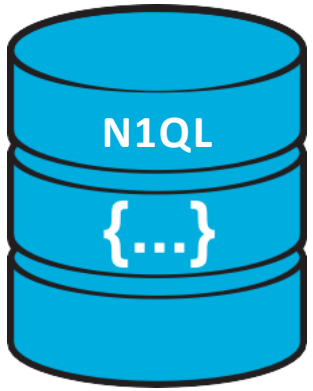
General purpose capabilities support a broad range of apps and use cases



Highly available cache

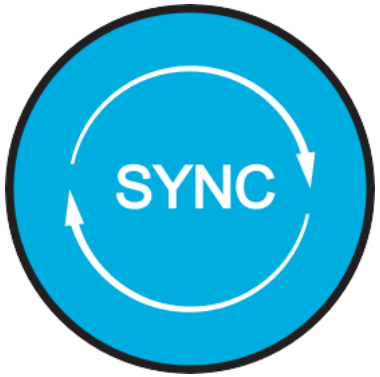


Key-value store

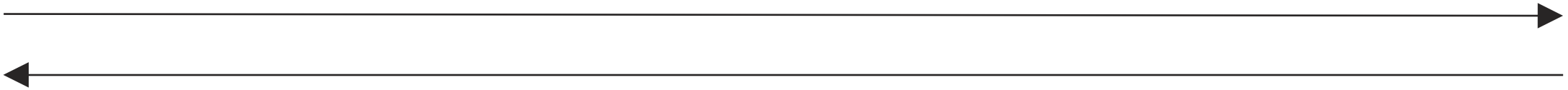


Document database

Embedded database



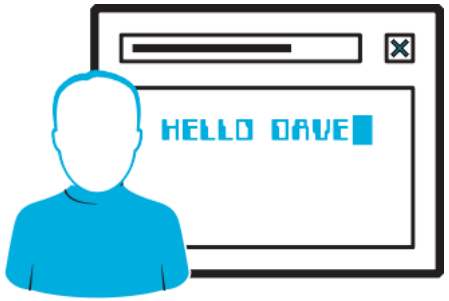
Sync management



Enterprises use Couchbase to enable key objectives

Profile Management

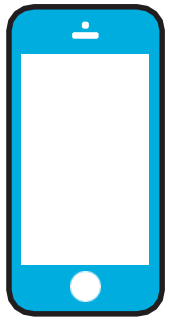
Personalization



360 Degree Customer View

Internet of Things

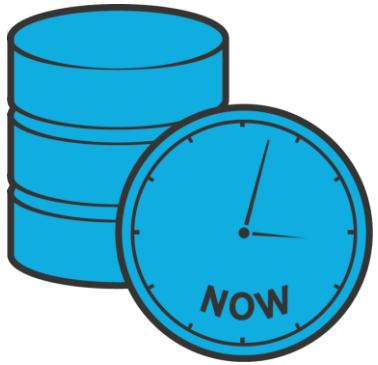
Mobile Applications



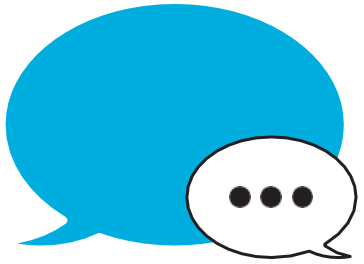
Content Management

Catalog

Real Time Big Data



Digital Communication



Fraud Detection

Develop with Agility

Storing and retrieving documents

Clients

Documents ← User/application data

↓ Read from /Written to

Servers

 Data Buckets

↓ Which live on

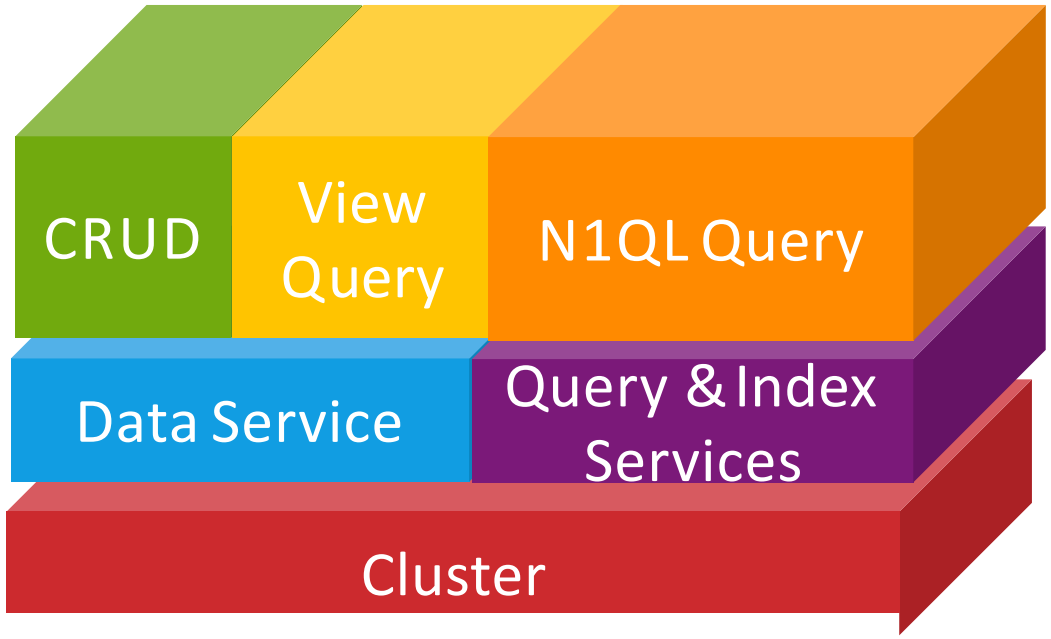
 Server Nodes ← Based on hash partitioning

↓ That form a

Couchbase Cluster ← Dynamically scalable

Accessing Data in Couchbase

- Multiple Access Paths



Functional

Hold on to cluster information such as topology.

API

Reference Cluster Management

openBucket()

info() disconnect()

Functional

Give the application developer a concurrent API for basic (k-v) or document management

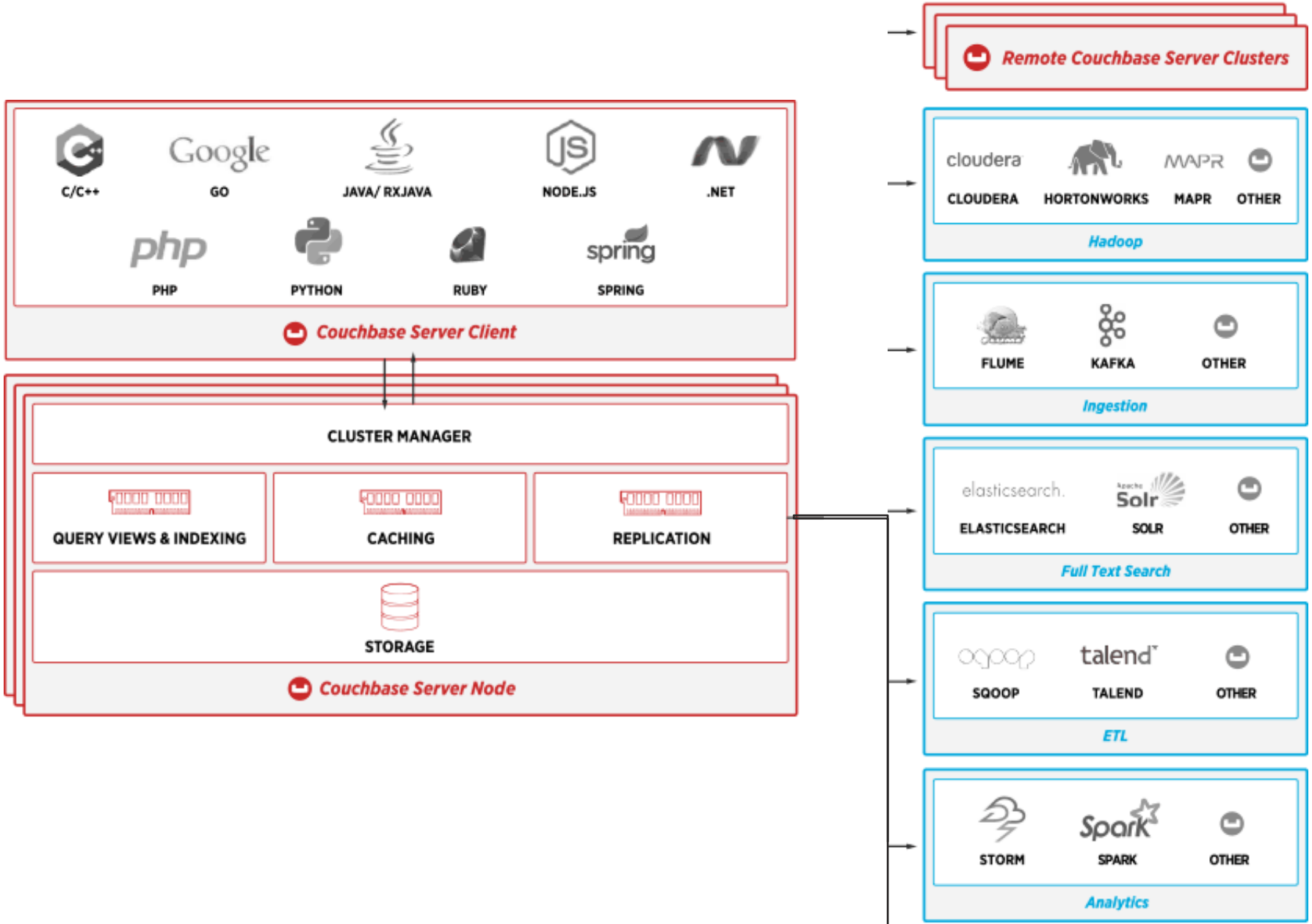
API

get() insert()

upsert()

remove()

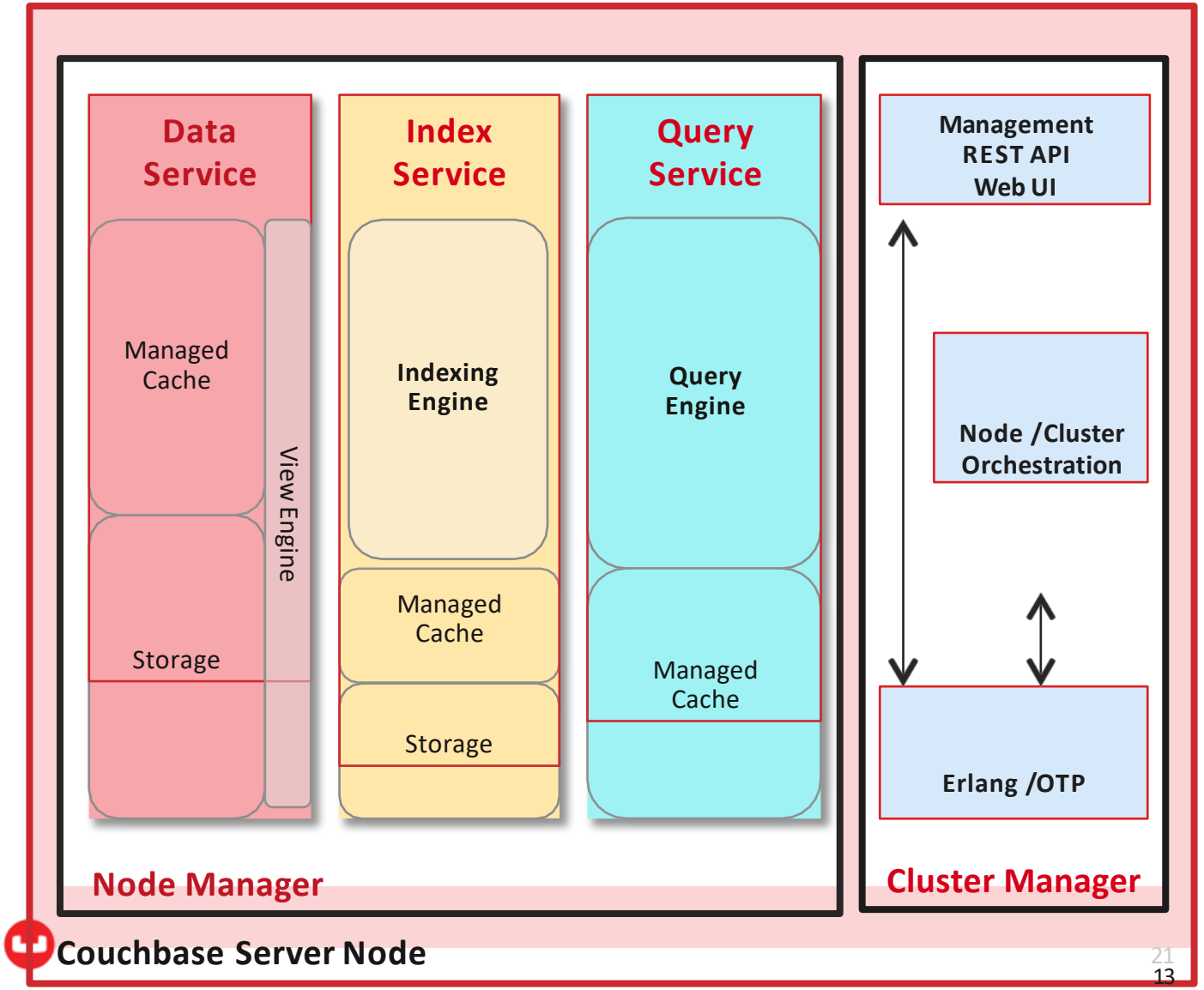
Couchbase SDKs and Connectors



Operate at Any Scale

Couchbase Architecture – Single Node

- **Data Service** – builds and maintains Distributed secondary indexes (MapReduce Views)
- **Indexing Engine** – builds and maintains Global Secondary Indexes
- **Query Engine** – plans, coordinates, and executes queries against either Global or Distributed indexes
- **Cluster Manager** – configuration, heartbeat, statistics, RESTful Management interface



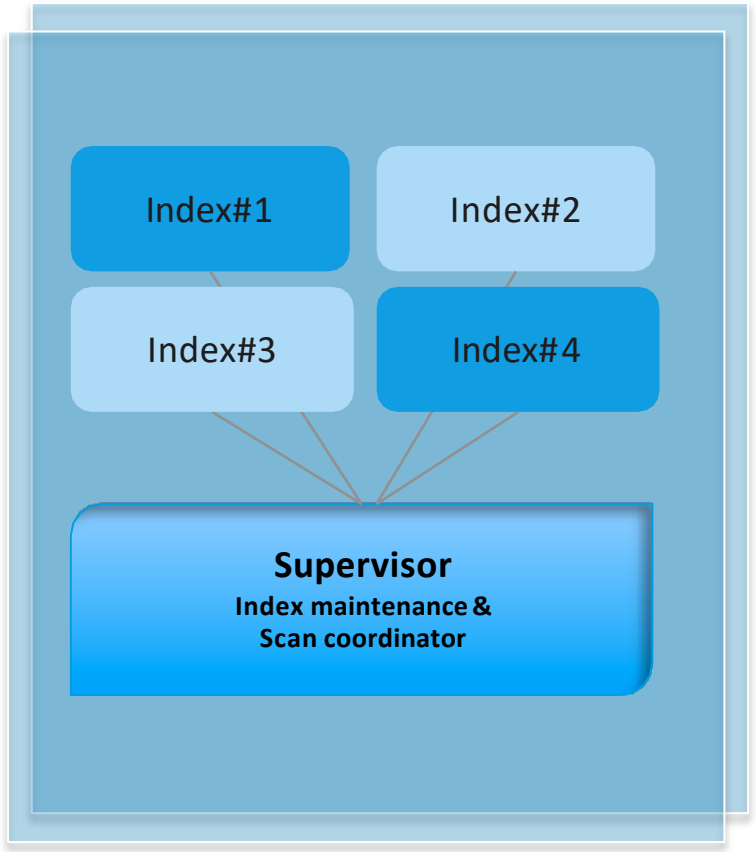
Couchbase Views

- Local Index
  - Distributed indexing and scatter gather querying
- Incremental Map-Reduce
  - Distributed simple real-time analytics
  - Only considers changes due to updated data

©2014 Couchbase, Inc.

Index Service

Couchbase Global Indexing Service



Indexing Service

Global Secondary Index Service

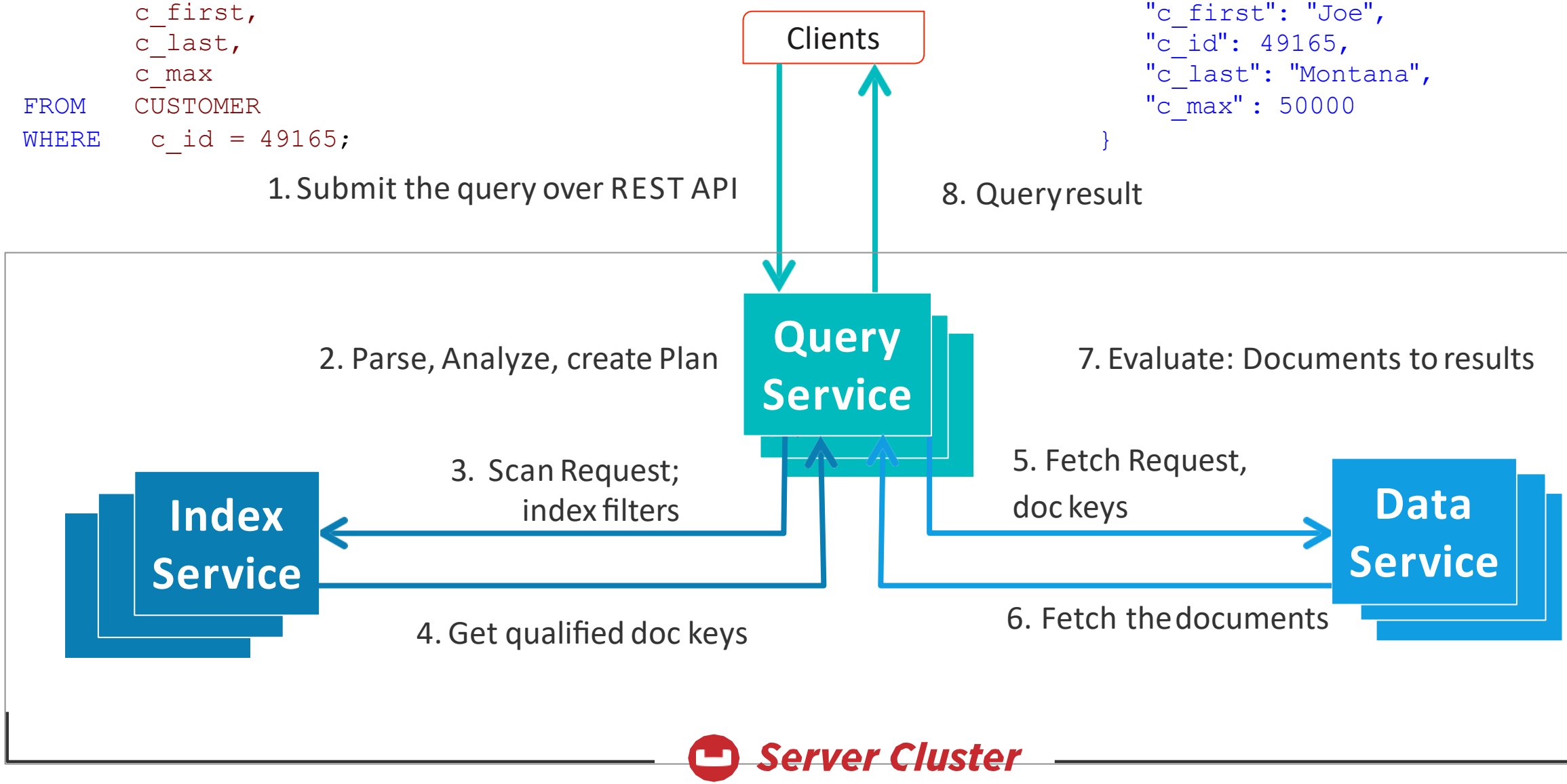
- New to 4.0
- Indexes partitioned independently from data
- Each index receives only its own mutations
- Managed Caching layer
- **ForestDB storage engine**
  - B+ Trie optimized for very large data volumes
  - Optimized for SSD's

Query Service

Query Execution Flow

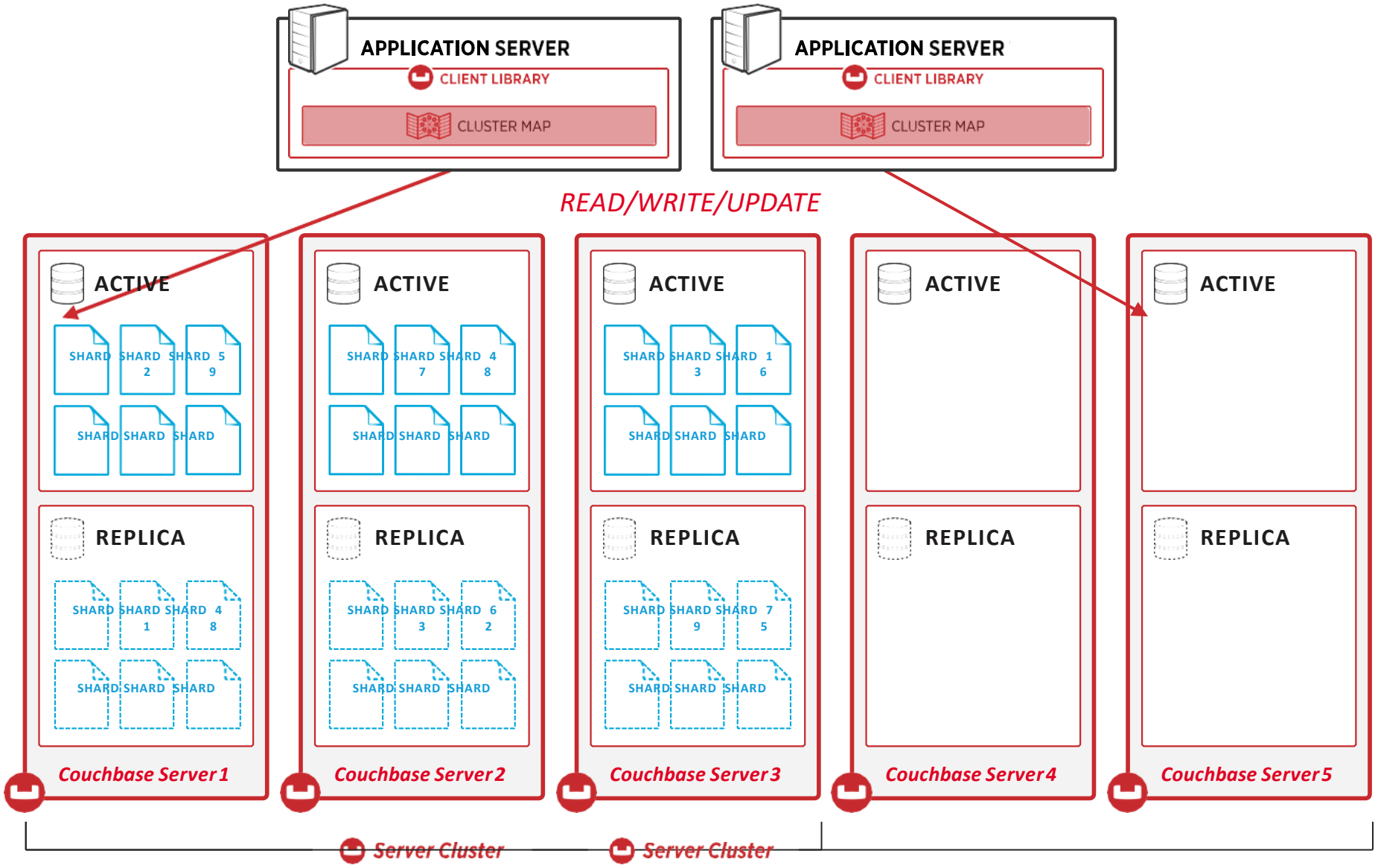
```
SELECT c_id,
       c_first,
       c_last,
       c_max
FROM   CUSTOMER
WHERE   c_id = 49165;
```

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



Couchbase Clustering Architecture

Data Services – Sharding and Replication



Application has single logical connection to cluster (client object)

- Multiple nodes added or removed at once
- One-click operation
- Incremental movement of active and replica vbuckets and data
- Client library updated via cluster map
- Fully online operation, no downtime or loss of performance

Multi-Dimensional Scaling (MDS)

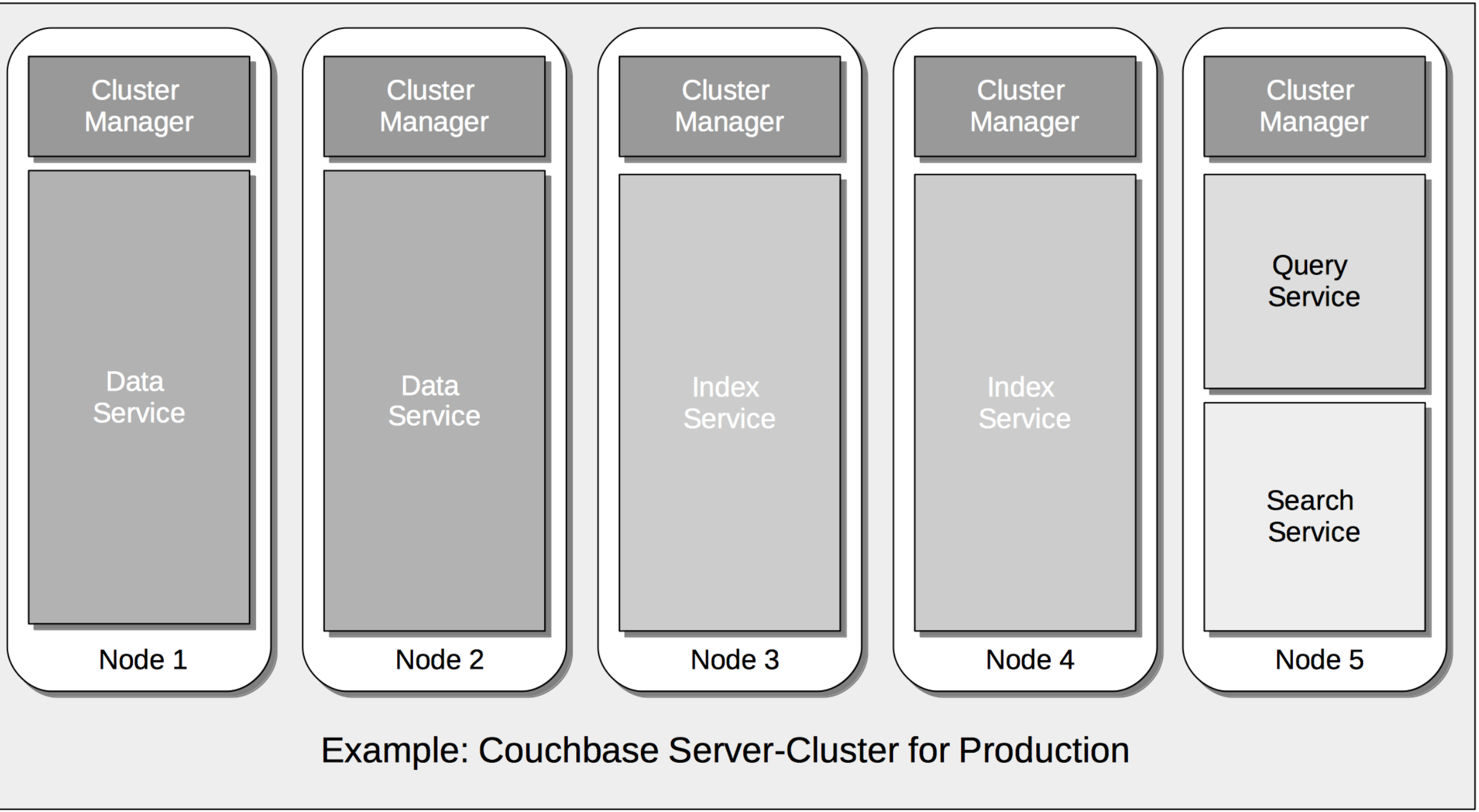
- Couchbase MDS features improve performance and throughput for mission-critical systems by enabling independent scaling of data, query, and indexing workloads.
- Scale-out and scale-up are the two scalability models typical for databases – Couchbase takes advantage of both.
- Unique ways to combine and mix these models in a single cluster to maximize throughput and latencies.
- With MDS, admins can achieve both the existing homogeneous scalability model and the newer independent scalability model.

Homogeneous scaling model

- Application workloads are distributed equally across a cluster made up of the homogeneous set of nodes.
- Each node that does the core processing takes a similar slice of the work and has the same hardware resources.
- This model is available through MDS and is simple to implement but has a couple drawbacks.
- Components processing core data operations, index maintenance, or executing queries all compete with each other for resources.
- It is impossible to fine-tune each component because each of them has different demands on hardware resources.
- This is a common problem with other NoSQL databases.
- 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.

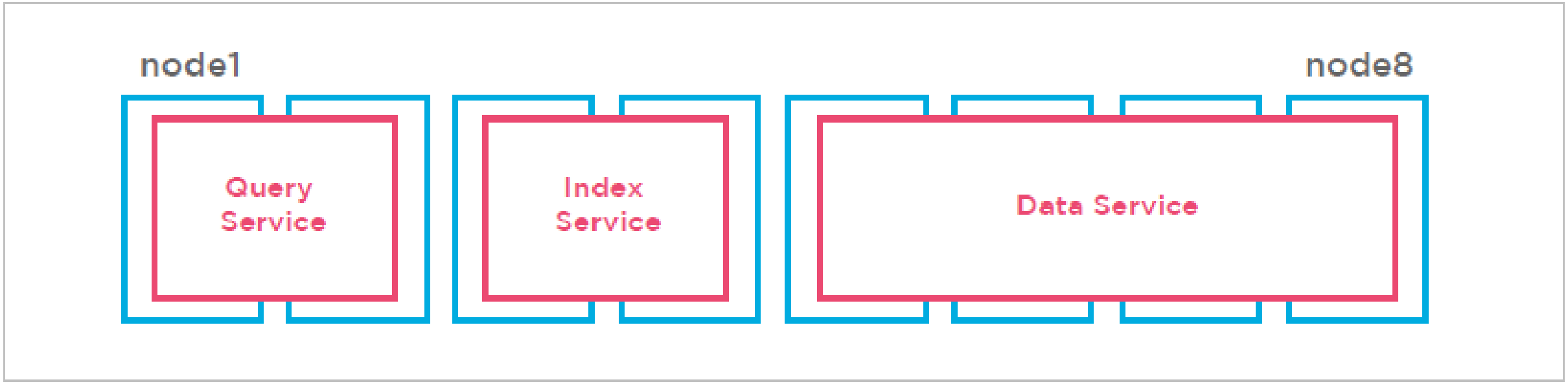
Independent scaling model

- MDS is designed to minimize interference between services.
- When you separate the competing workloads into independent services and isolate them from each other, interference among them is minimized.
- In this topology, each service is deployed to an independent zone within the cluster.
- Each service zone within a cluster (data, query, and index services) can now scale independently so that the best computational capacity is provided for each of them.

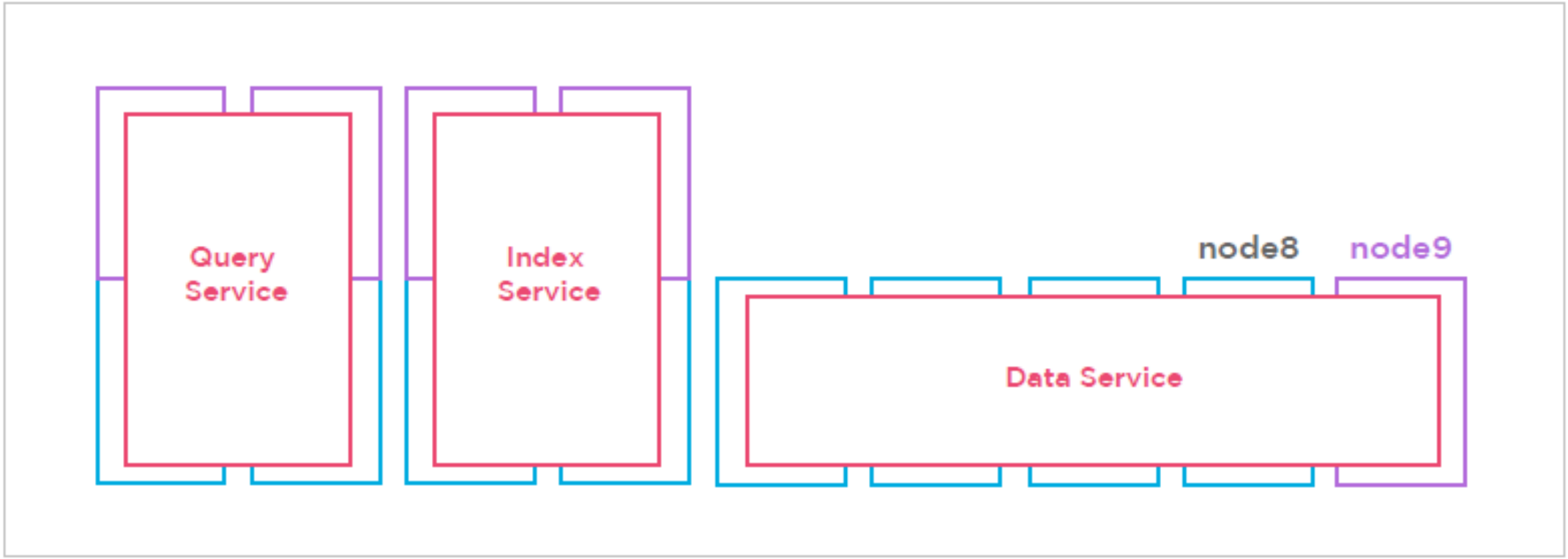


Example: Couchbase Server-Cluster for Production

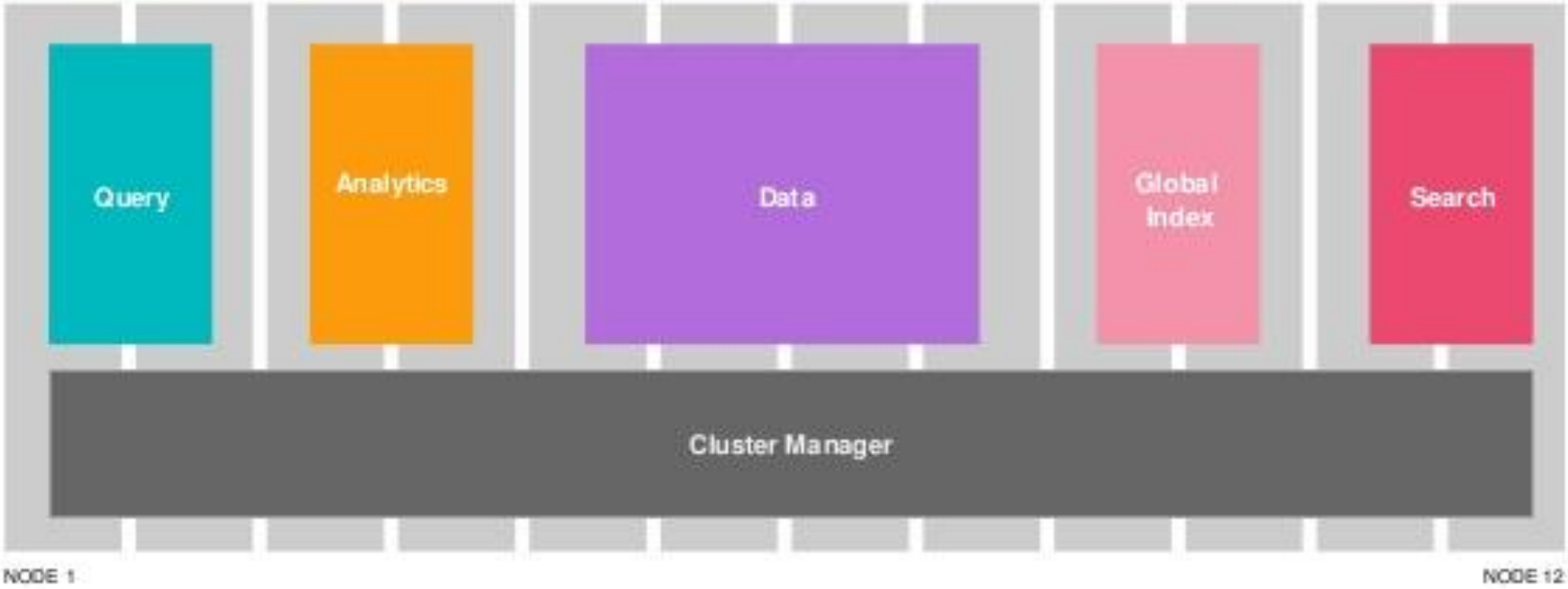
Independent scaling model



Independent scaling model

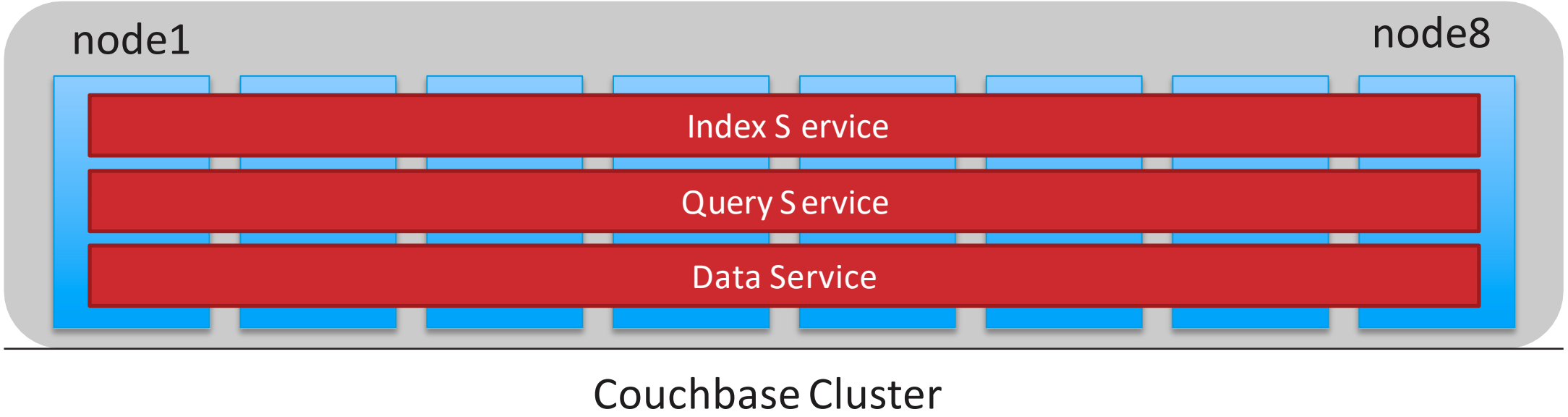


Multidimensional Scaling



What is Multi-Dimensional Scaling?

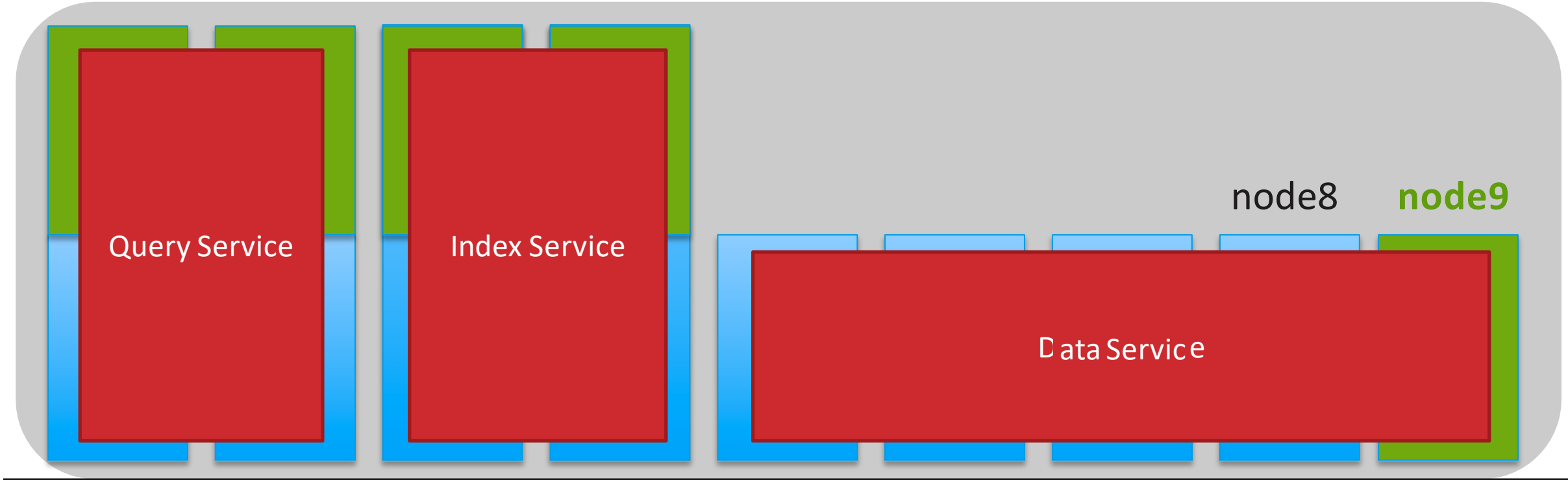
MDS is the architecture that enables independent scaling of data, query and indexing workloads while being managed as one cluster



Modern Architecture

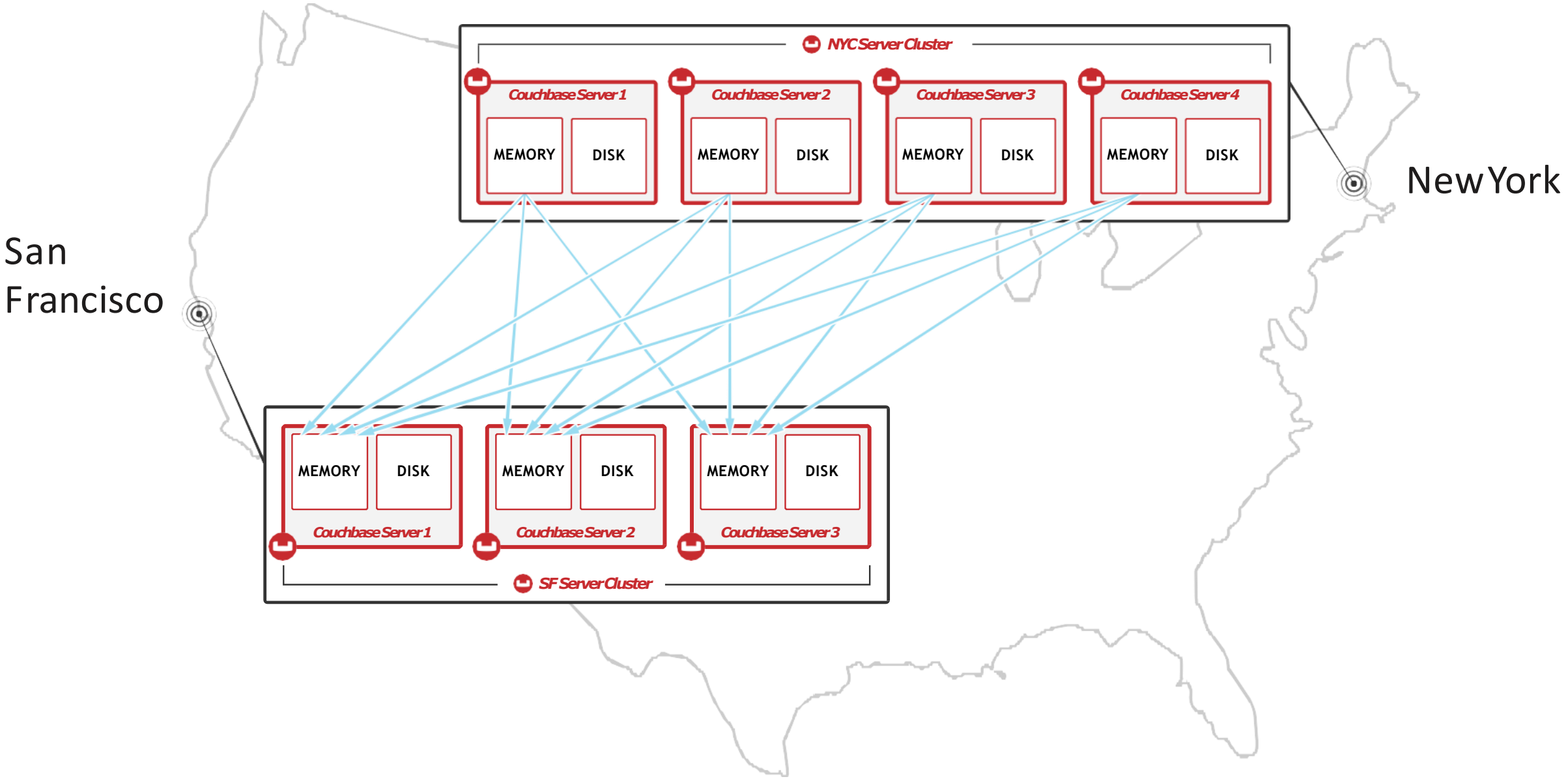
- **Independent Scalability for Best Computational Capacity per Service**

Heavier indexing (index more fields) : scale up index service nodes
More RAM for query processing: scale up query service nodes



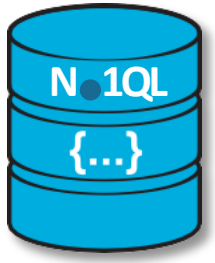
Cross Data Center Replication

Market leading memory-to-memory replication



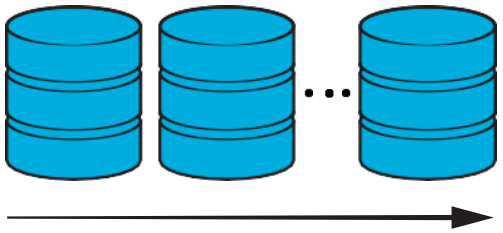
In summary

The best of both worlds



Develop with Agility

- Multiple data models
- N1QL - SQL-Like query language
- Multiple indexes
- Languages, ODBC /JDBC drivers and frameworks you already know



Operate at Any Scale

- Push-button scalability
- Consistent high-performance
- Always on 24x7 with HA - DR
- Easy Administration with Web UI, Rest API and CLI

Caching and Persistence

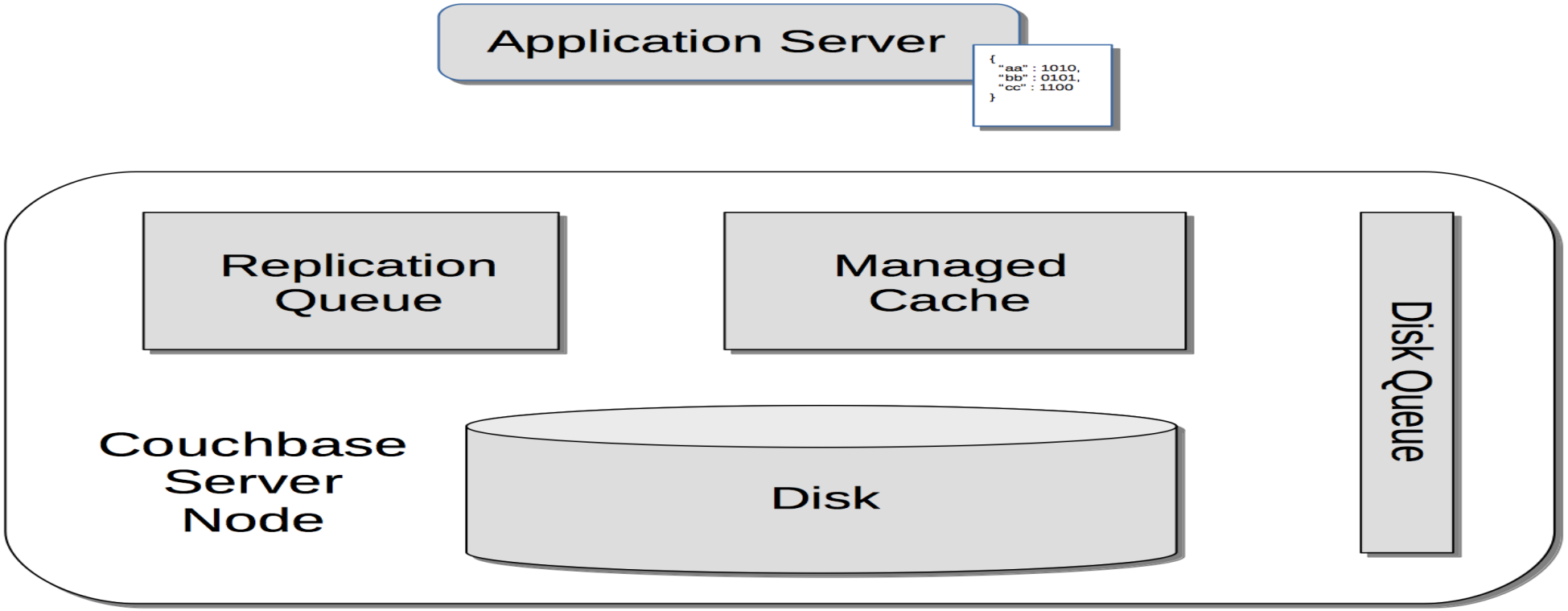
- Provides a fully integrated caching layer, which provides high-speed data-access.
- Automatically manages the caching layer, ensuring that sufficient memory is available in relation to occupied disk-space, in order to maintain optimal performance.
- Data associated with Couchbase (as opposed to Ephemeral) buckets is also maintained persistently, on disk.
- Such items, when they come into the caching layer, are placed on the disk queue, to be written to disk; and at the same time, if appropriate, are placed on a replication queue, so that one or more replica buckets can be created or updated.
- Memory quotas, established by the Full Administrator when the server is configured, are automatically managed by Couchbase Server with reference to watermarks, which specify how much free memory should be consistently maintained within the caching layer.
- Infrequently used items are written to disk; and are then removed from memory in order to free up space.
- Ejection, is managed asynchronously, while the server continues to service active requests. Items ejected from Couchbase buckets can subsequently be re-acquired from disk, as necessary.
- A working-set of most frequently used data is tracked and maintained, and the items kept in memory to ensure high performance. The priority of disk I/O is configurable per bucket.

Saving New Items

- When a new item, such as a document, is saved by an application, it is saved in memory only, if the item is contained by an Ephemeral bucket; and is saved both in memory and on disk, if the item is contained by a Couchbase bucket.
- If the bucket (either Ephemeral or Couchbase) has been replicated one or more times, a copy of the new item is saved in each replica.

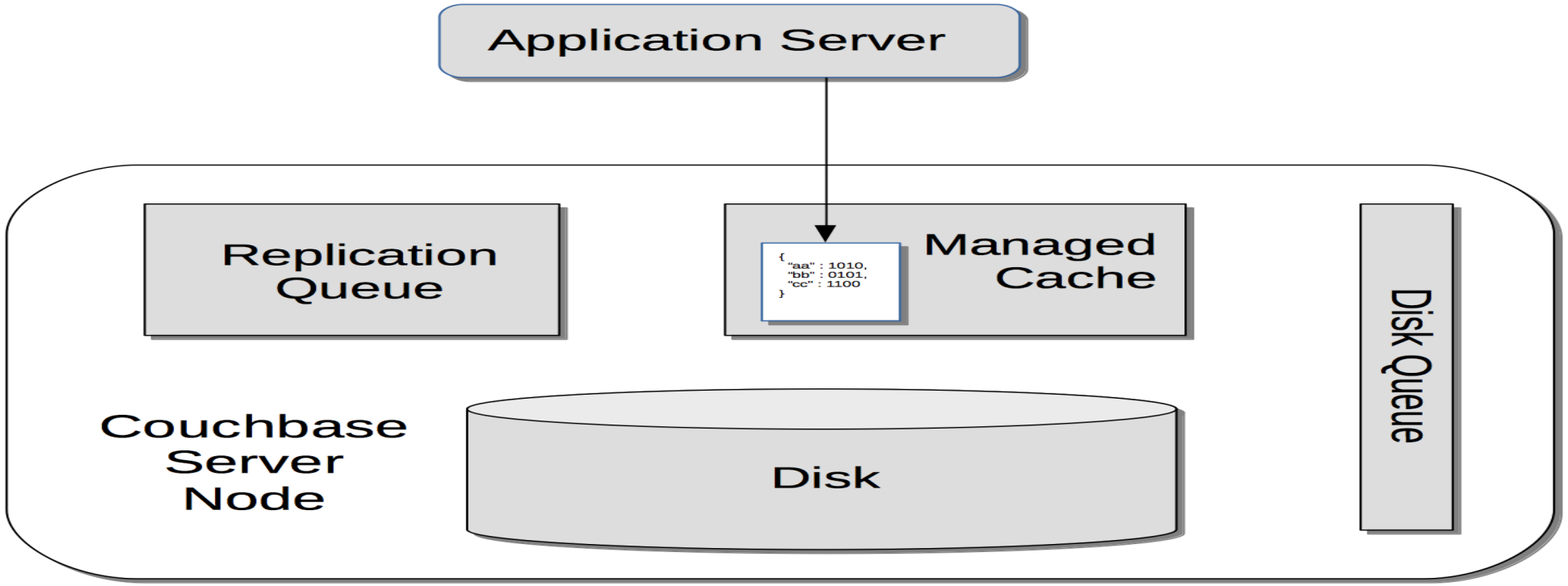
Saving New Items

- Application creates a new document.



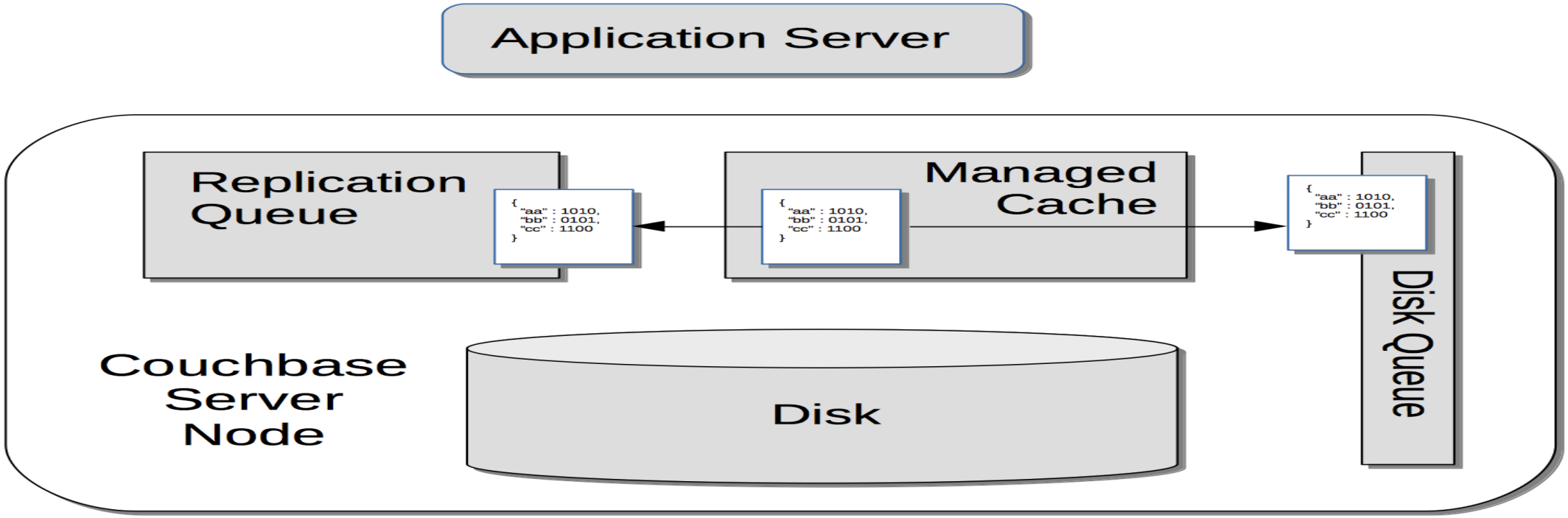
Saving New Items

- Application saves the document on Couchbase Server. The document is received in memory.



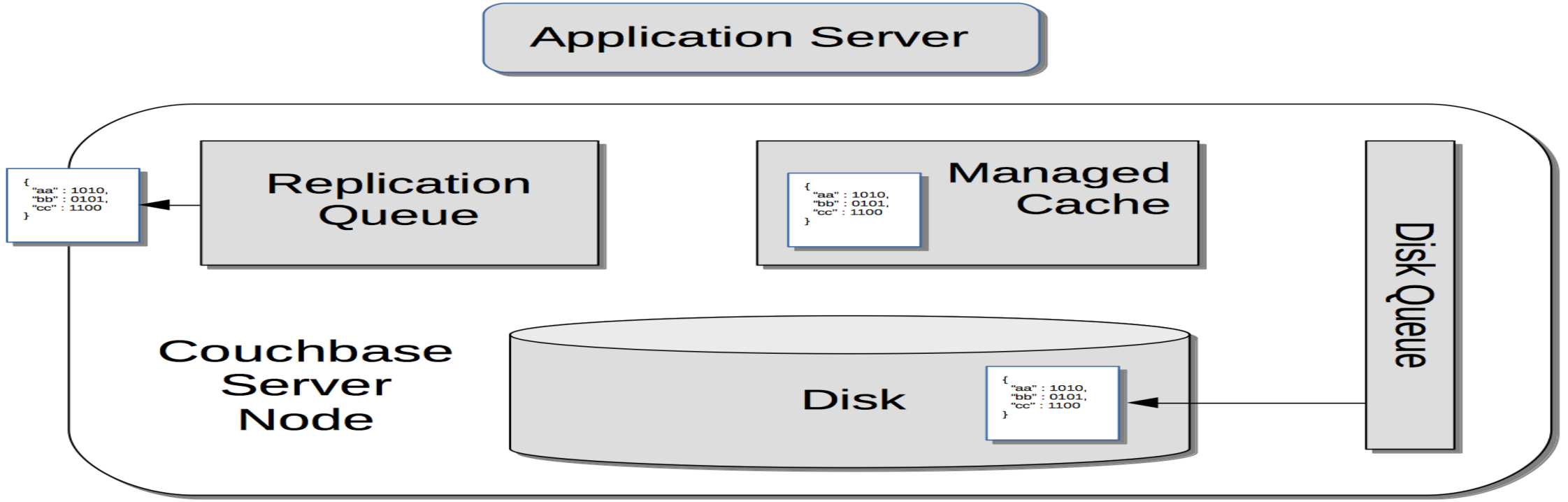
Saving New Items

- Places a compressed copy of the document onto the Disk Queue so that the document can be written to the disk; and a further copy onto the Replication Queue, so that the document can be written to the replica bucket. (Copy on the Replication Queue may or may not be compressed, depending on the compression mode configured for the bucket.)



Saving New Items

- Once written, the new document resides both in the memory and on the disk of the node. It will also reside in the memory and on the disk of whichever other nodes maintain the replicas of its buckets.

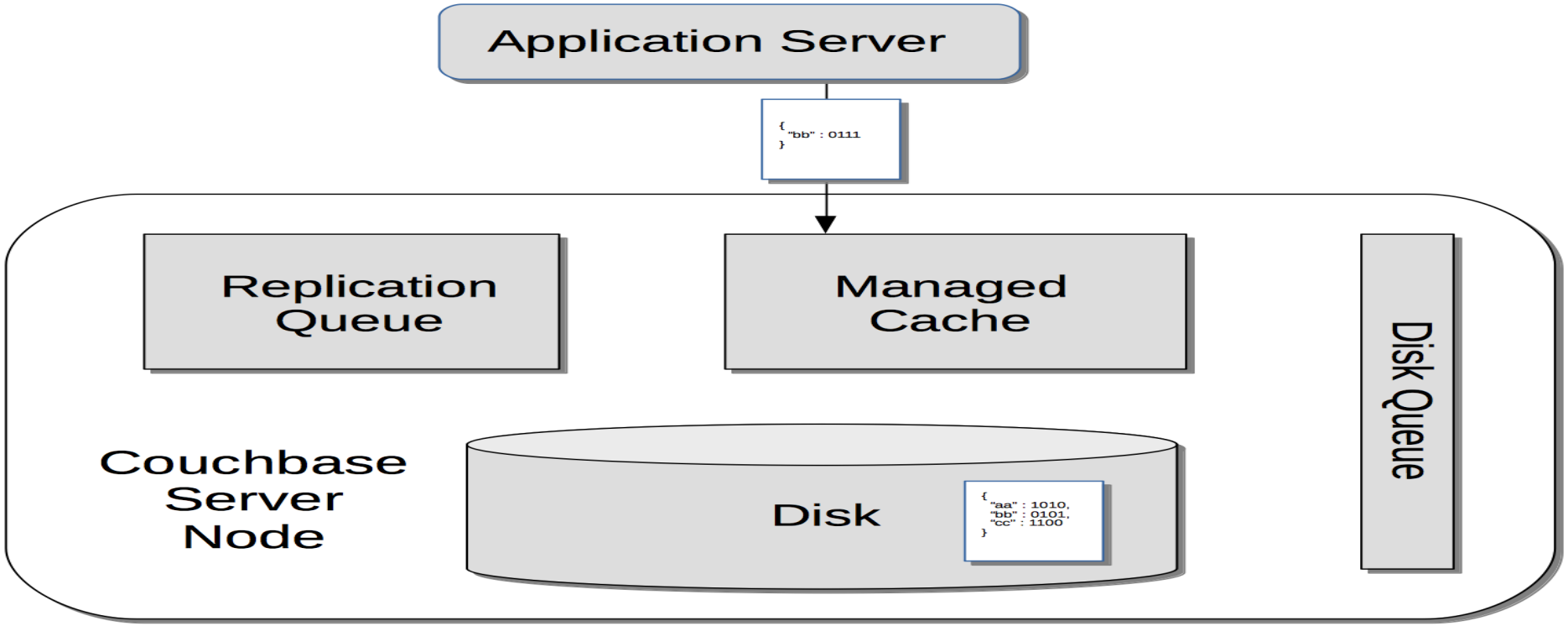


Updating Items

- Items that reside in the memory of Couchbase Server can be updated.
- If the item belongs to a Couchbase bucket, the item is also updated on disk.
- If the item is not currently in memory, but resides on disk, Couchbase Server retrieves the item, so that it can be updated.

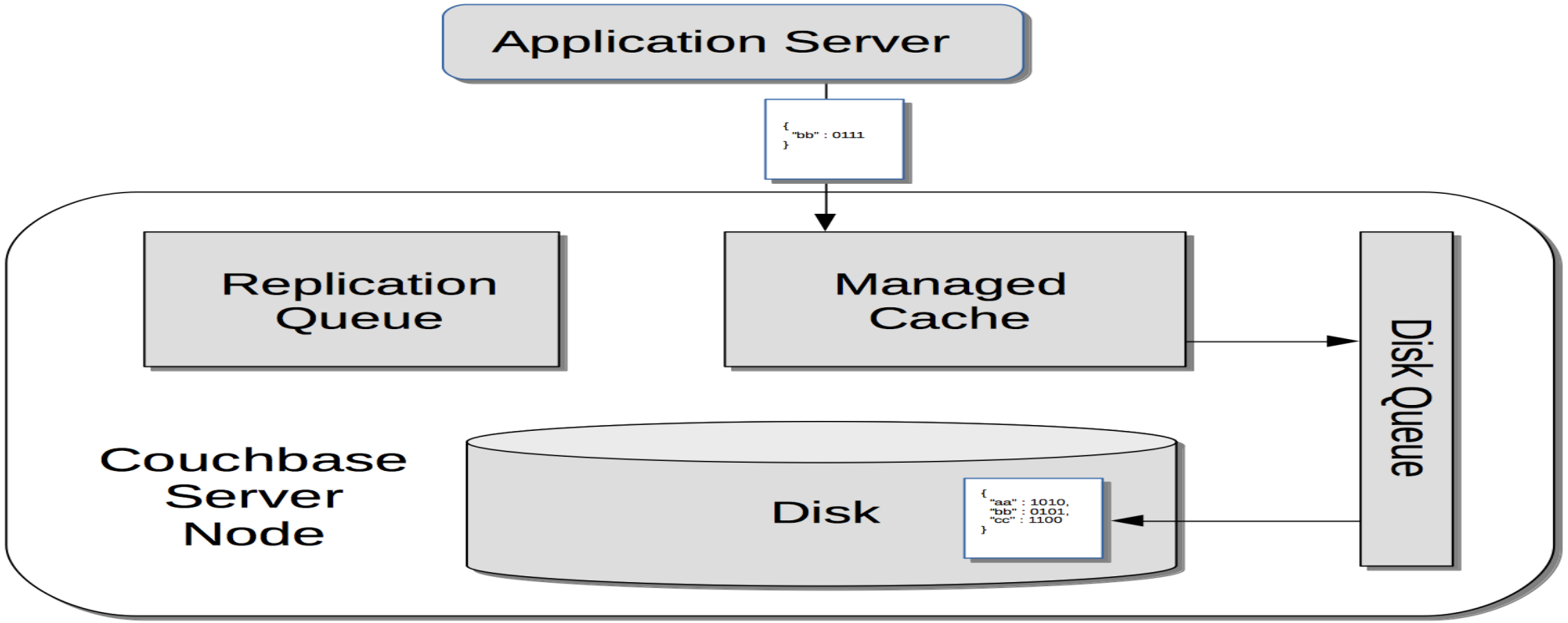
Updating Items

- Application provides an update to an existing document.



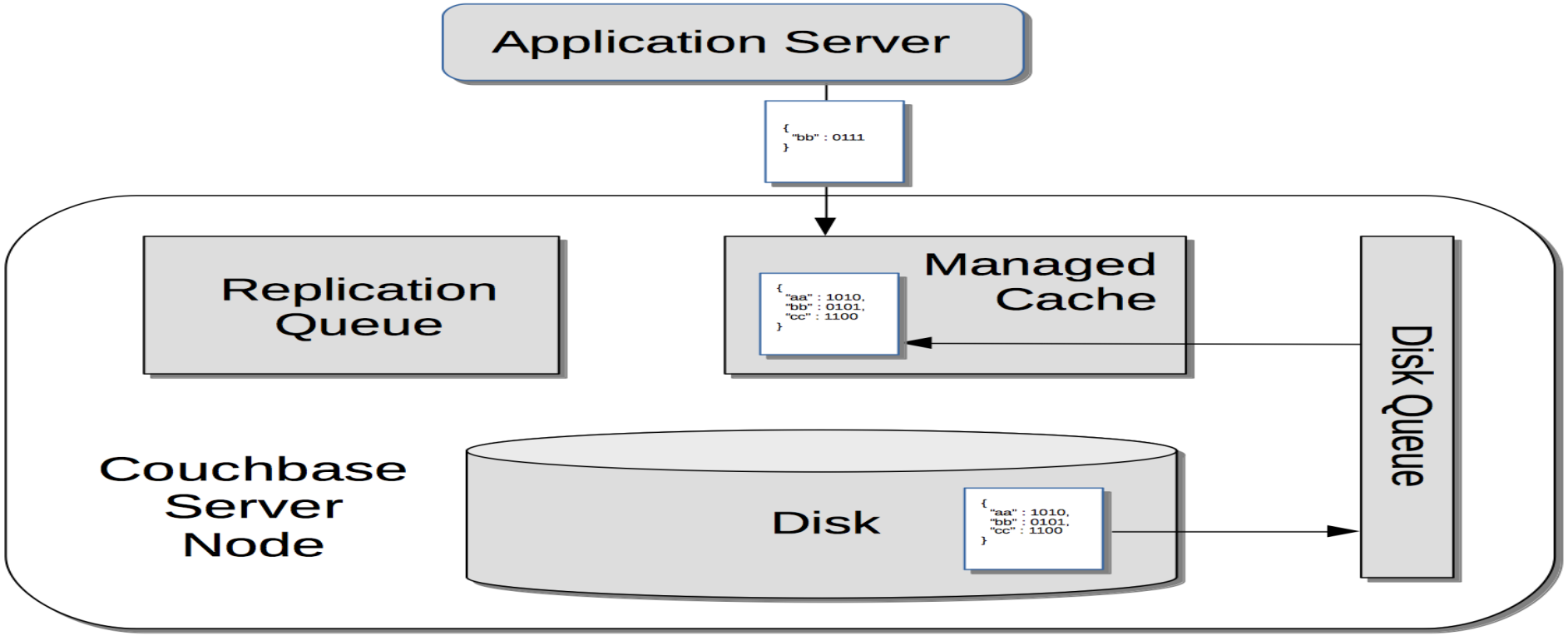
Updating Items

- **Since the document is not currently in memory, Couchbase Server seeks it on disk, where is resides in compressed form.**



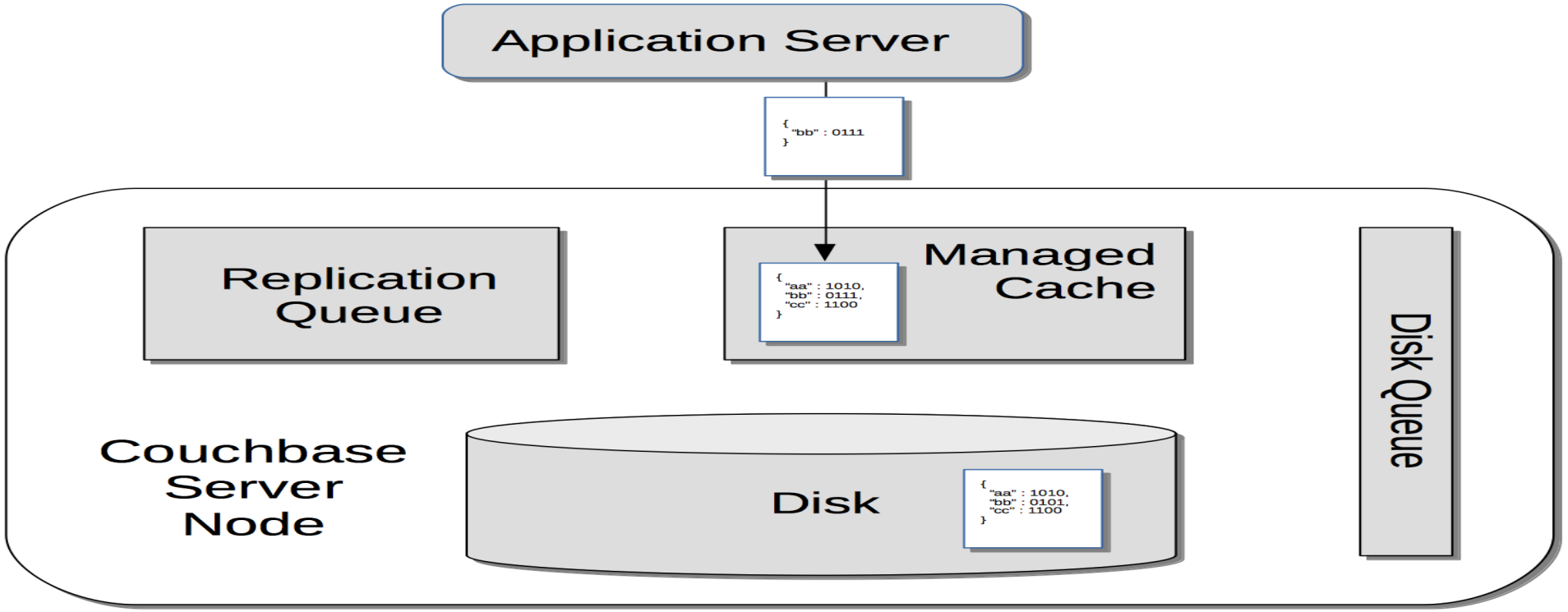
Updating Items

- compressed document is retrieved, brought into memory, and decompressed.



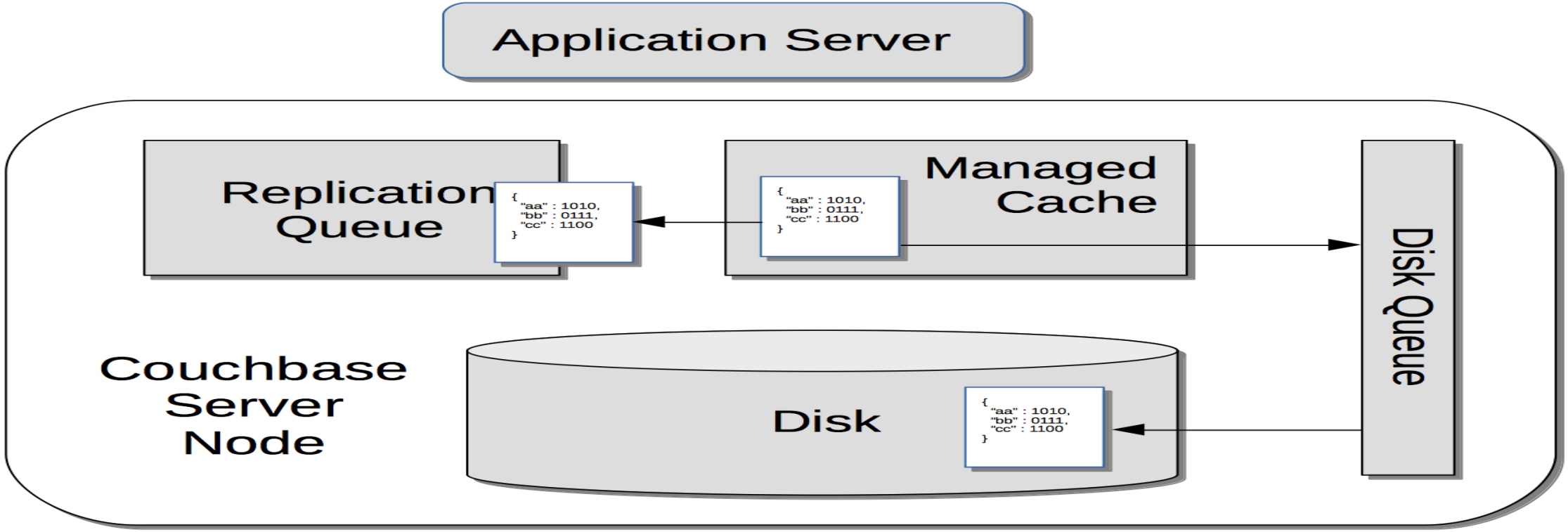
Updating Items

- application's updates are applied to the uncompressed document.



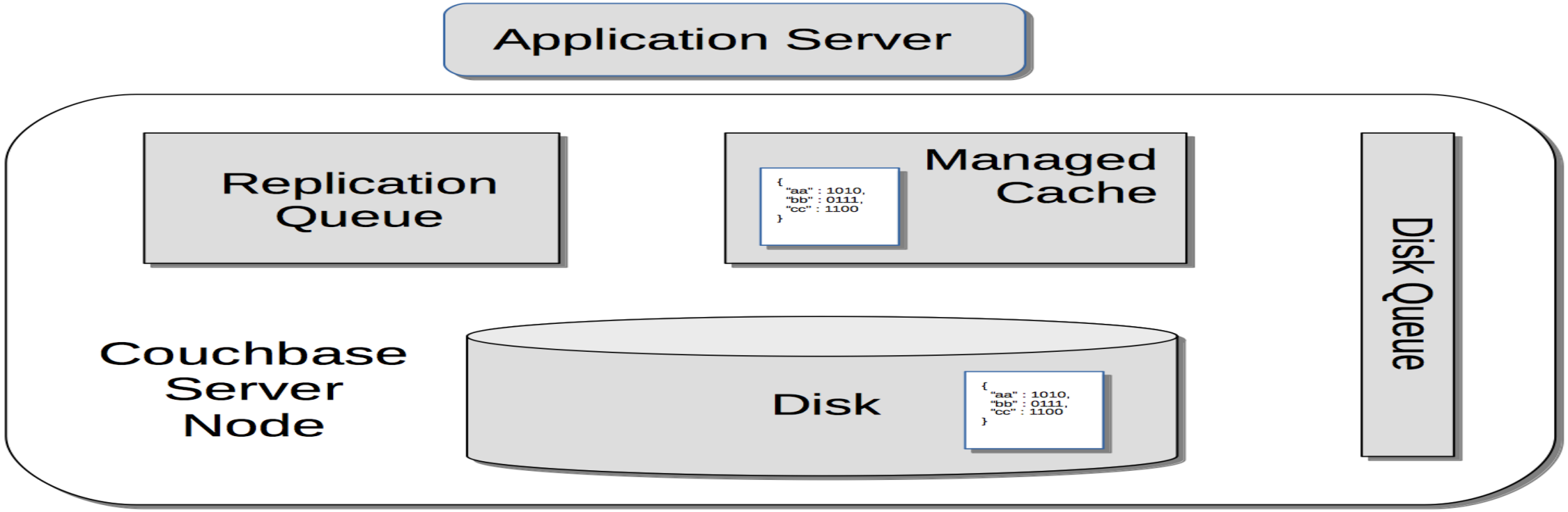
Updating Items

- updated document is placed (in either compressed or uncompressed form, as appropriate) on the replication queue, so that replicas can be updated. The updated document is also re-compressed, and written to disk locally.



Updating Items

- updated document is now retained locally on disk and in memory. The document remains in memory unless it is ejected at some point, after which it continues to reside on disk.



JSON data model

- Supports basic and complex data types: numbers, strings, nested objects, and arrays.
- JSON provides rapid serialization and deserialization, is native to JavaScript, and is the most common REST API data format.
- Consequently, JSON is extremely convenient for web application programming.
- Couchbase stores data as individual documents, comprised of a key and a value.
- When the value is JSON-formatted, Couchbase provides rich access capabilities; when not, the document is stored as a binary BLOB and has more limited access characteristics.

Keys, values, and sub-documents

- Keys and values are fundamental parts of JSON documents and have some limits that are important to understand.

Keys

- Each value is identified by a unique key, or ID, defined by the user or application when the item is originally created.
- Key is immutable: once the item is saved, the key cannot be changed.
- Each key must be a UTF-8 string with no spaces.
- Special characters, such as (, %, /, “, and _, are acceptable, but the key may be no longer than 250 bytes and must be unique within its bucket.

Keys, values, and sub-documents

Values

- The maximum size of a value is 20 MB.
- Each document consists of one or more attributes, each of which has its own value. An attribute's value can be a basic type, such as a number, string, or boolean; or a complex type, such as an embedded document or an array.
- JSON documents can be parsed, indexed, and queried by other Couchbase services.
- A value can also be any form of binary, though it won't be parsed, indexed, or queried.

Sub-documents

- A sub-document is an inner component of a JSON document.
- The sub-document API uses a path syntax to specify attributes and array positions to read/write.
- This makes it unnecessary to transfer entire documents over the network when only partial modifications are required.

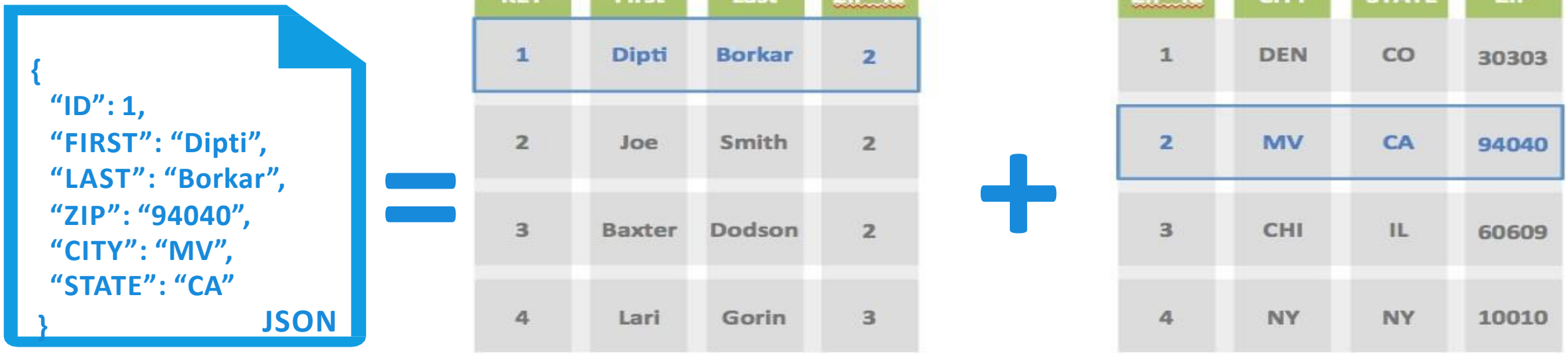
Keys, values, and sub-documents

- Document key: *user200*

```
{ "age": 21, "fav_drinks": { "soda": [ "fizzy", "lemon" ] } "addresses": [
{ "street": "pine" }, { "street": "maple" } ] }
```

- **Path example Result**
- age – a numeric value 21
- fav_drinks.soda – an array of strings --fizzy, lemon
- fav_drinks.soda[0] – first string in array-- fizzy
- addresses[1].street – string value in second part of array -- maple

What does a JSON document look like?



All data in a single document

Buckets

- Buckets hold JSON documents – they are a core concept for document organization in Couchbase. Applications connect to a specific bucket that pertains to their application scope. Memory quotas are managed on a per-bucket and per-service basis. Security roles are applied to users with various bucket-level constraints.
- In addition to standard Couchbase buckets, there are two specialized bucket types useful for different use cases.
- Ephemeral buckets do not persist data but allow highly consistent in-memory performance, without disk-based fluctuations. This delivers faster node rebalances and restarts.
- Memcached buckets also do not persist data. It is a legacy bucket type designed to be used alongside other database platforms specifically for in-memory distributed caching. Memcached buckets lack most of the core benefits of Couchbase buckets, including compression.

COUCHBASE SERVICES

- Couchbase implements the data access methods through a set of dedicated services, with data at its center.
- Each service has their own resource quotas and, where applicable, related indexing and inter-node communication capabilities.
- Provides several very flexible methods to scale services when needed – not just scaling up to larger machines or scaling out to more nodes.
- Couchbase provides both options, as well as the ability to scale specific services differently than one another.
- Multi-dimensional scaling is the foundation of these workload isolation and scaling capabilities.
- Small-scale environments can share the same workloads across one or more nodes, while higher scale and performance can be achieved with dedicated nodes to handle specific workloads – the ultimate in scale-out capability.
- Cluster can be scaled in or out and its service topology changed on demand with zero interruption or change to the application.

Data service and KV engine

- Data service is the foundation for storing data in Couchbase Server must run on at least one node of every cluster –
- Responsible for caching, persisting and serving data to applications and other services within the cluster.
- Principal component of the data service architecture is the key-value management system known simply as KV engine.
- KV engine is composed of a multi-threaded, append-only storage layer on disk with a tightly integrated managed object cache.
- The cache provides consistent low latency for individual document read and write operations and streams documents to other services via DCP.
- Each node running the data service has its own KV engine process and is responsible for persisting and caching a portion of the overall dataset (both active and replica).

Managed object cache

- Managed object cache of each node hashes the document into an in-memory hash table based upon the document ID (key).
- Hash table stores the key, the value and some metadata associated with each document.
- Since the hash table is in memory and lookups are fast, it offers a quick way of detecting whether the document exists in memory or not.
- The cache is both read-through and write-through: if a document being read is not in the cache, it is fetched from disk and write operations are written to disk after being first stored in memory.
- Disk fetch requests are batched to the underlying storage engine, and corresponding entries in the hash table are filled.
- After the disk fetch is complete, pending client connections are notified of the asynchronous I/O completion so that they can complete the read operation.

Document expiration

- Documents may also be set to expire using a time to live (TTL) setting.
- By default, all documents have a TTL of 0, meaning the document will be kept indefinitely.
- When you add, set, or replace a document, you can specify a custom TTL, at which time the document becomes unavailable and is marked for deletion (tombstone) to be cleaned up later.

Memory management

- To keep memory usage of the cache under control, Couchbase employs a background task called the item pager.
- Pager runs periodically (to clean up expired documents) as well as being triggered based on high and low watermarks of memory usage.
- This high water mark is based off of the memory quota for a given bucket and can be changed at runtime.
- When the high watermark is reached, the item pager scans the hash table, ejecting eligible (persisted) items that are not recently used (NRU).
- It repeats this process until memory usage falls below the low watermark.

Compression

- End-to-end data document compression is available across all features of the database using the open source Snappy library.
- Client capabilities, and the compression mode configured for each bucket, determine how compression will run.
- Data can optionally be compressed by the client (SDK) prior to writing into a bucket, within memory of the bucket and on disk.
- It is also compressed between nodes of the cluster and to remote clusters.

Compression modes

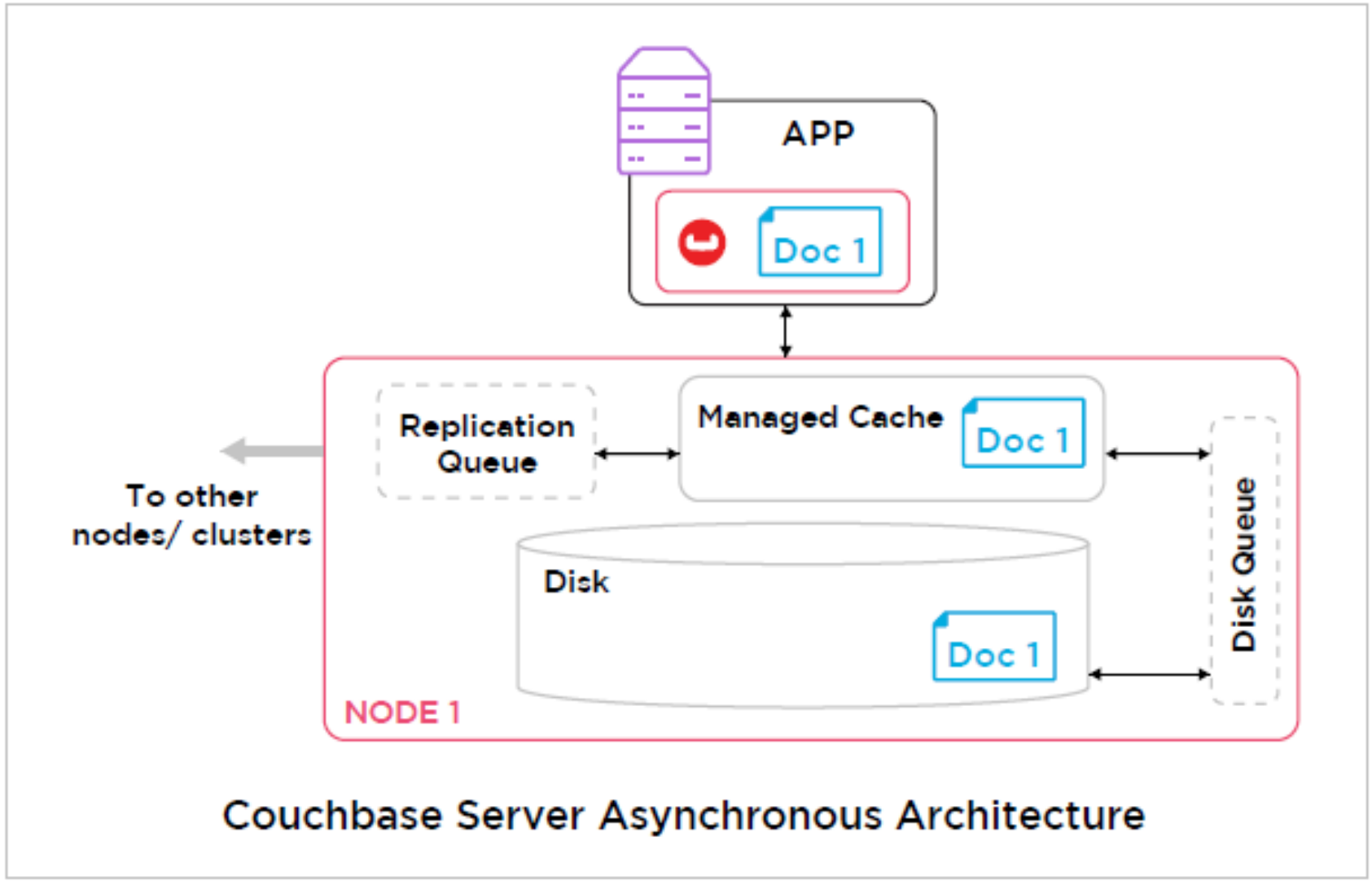
- Data is always compressed on disk, but each client and bucket can control whether it is also compressed on the wire and/or in memory.
- The SDKs communicate whether they will be sending or requesting compressed documents, and the compression mode of the bucket determines what happens within the database. The modes are as follows:
- **Off** – Documents are actively de compressed before storing in memory. Clients receive the documents uncompressed.
- **Passive** – Documents are stored in memory both compressed and uncompressed in memory, depending on how the client has sent them. Clients receive compressed documents if they are able and uncompressed if they are not.
- **Active** – Documents are actively compressed on the server, regardless of how they were received. Clients receive compressed data whenever it is supported by the client, even if it originally sent uncompressed data.

Compaction

- Couchbase writes all data that you append, update and delete as files on disk.
- Can eventually lead to gaps in the data file, particularly when you delete data.
- Can reclaim the empty gaps in all data files by performing a process called compaction.
- For both data files and index files, perform frequent compaction of the files on disk to help reclaim disk space and reduce disk fragmentation.
- Auto-compaction is enabled by default for all buckets, but parameters can be adjusted for the entire cluster or for a specific bucket in a cluster.

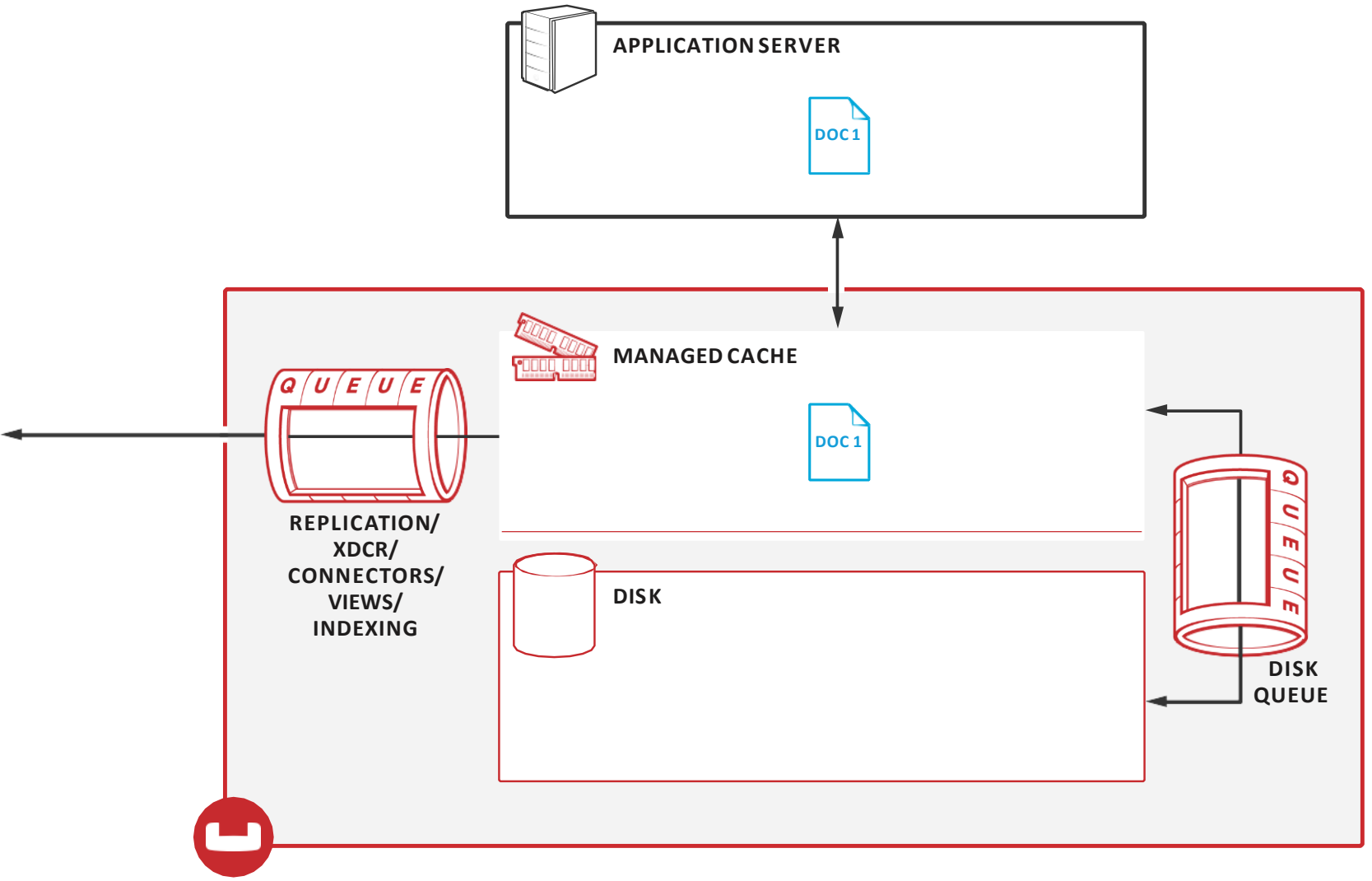
Mutations

- In Couchbase Server, mutations happen at a document level.
- Clients retrieve the entire document from the server, modify certain fields, and write the document updates back to Couchbase.



Couchbase Server Asynchronous Architecture

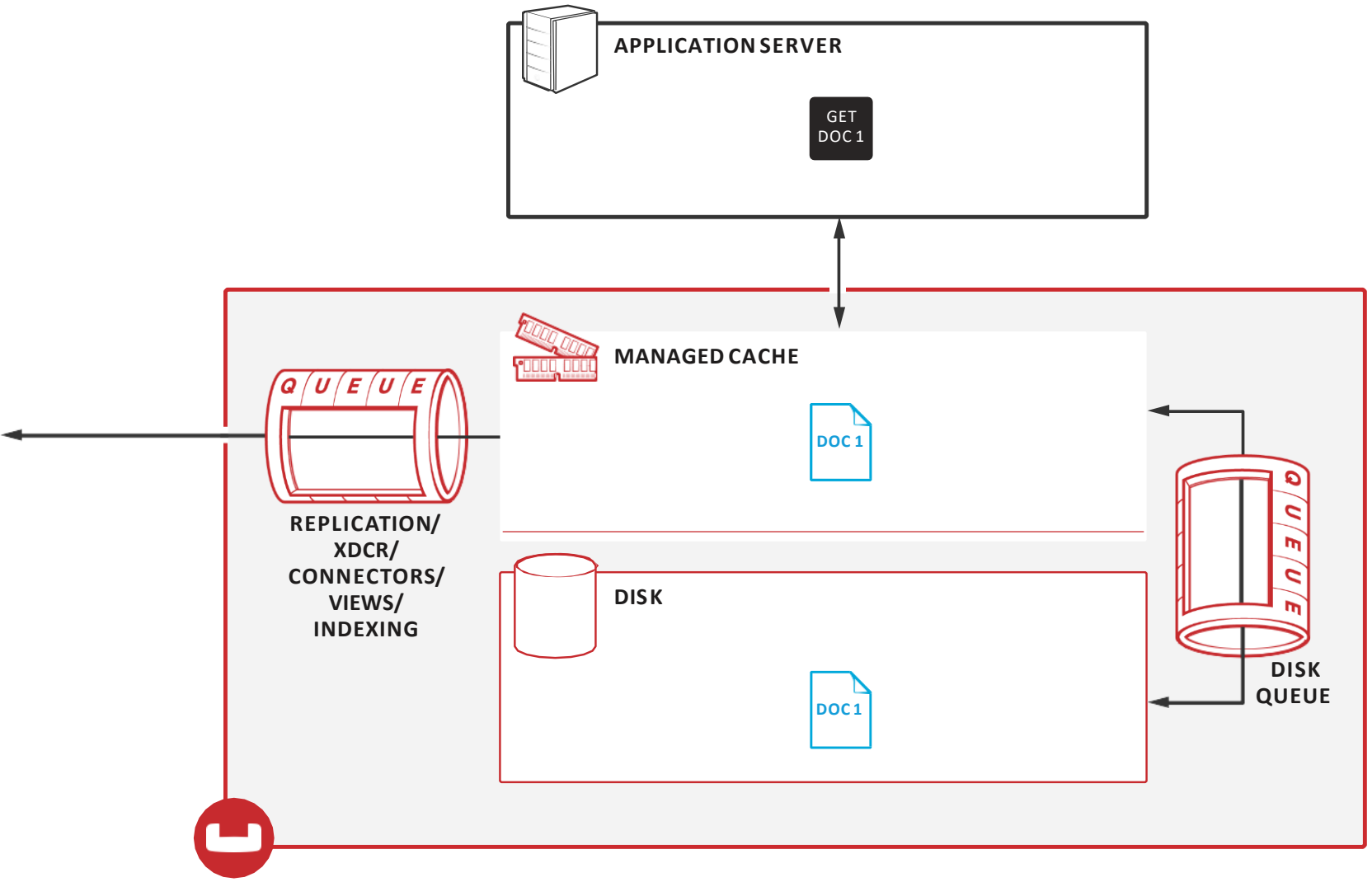
Data Service: Write Operation



Single-node type means easier administration and scaling

- Writes are async by default
- Application gets acknowledgement when successfully in RAM and can trade-off waiting for replication or persistence per-write
- Replication to 1, 2 or 3 other nodes
- Replication is RAM-based so extremely fast
- Off-node replication is primary level of HA
- Disk written to as fast as possible – no waiting

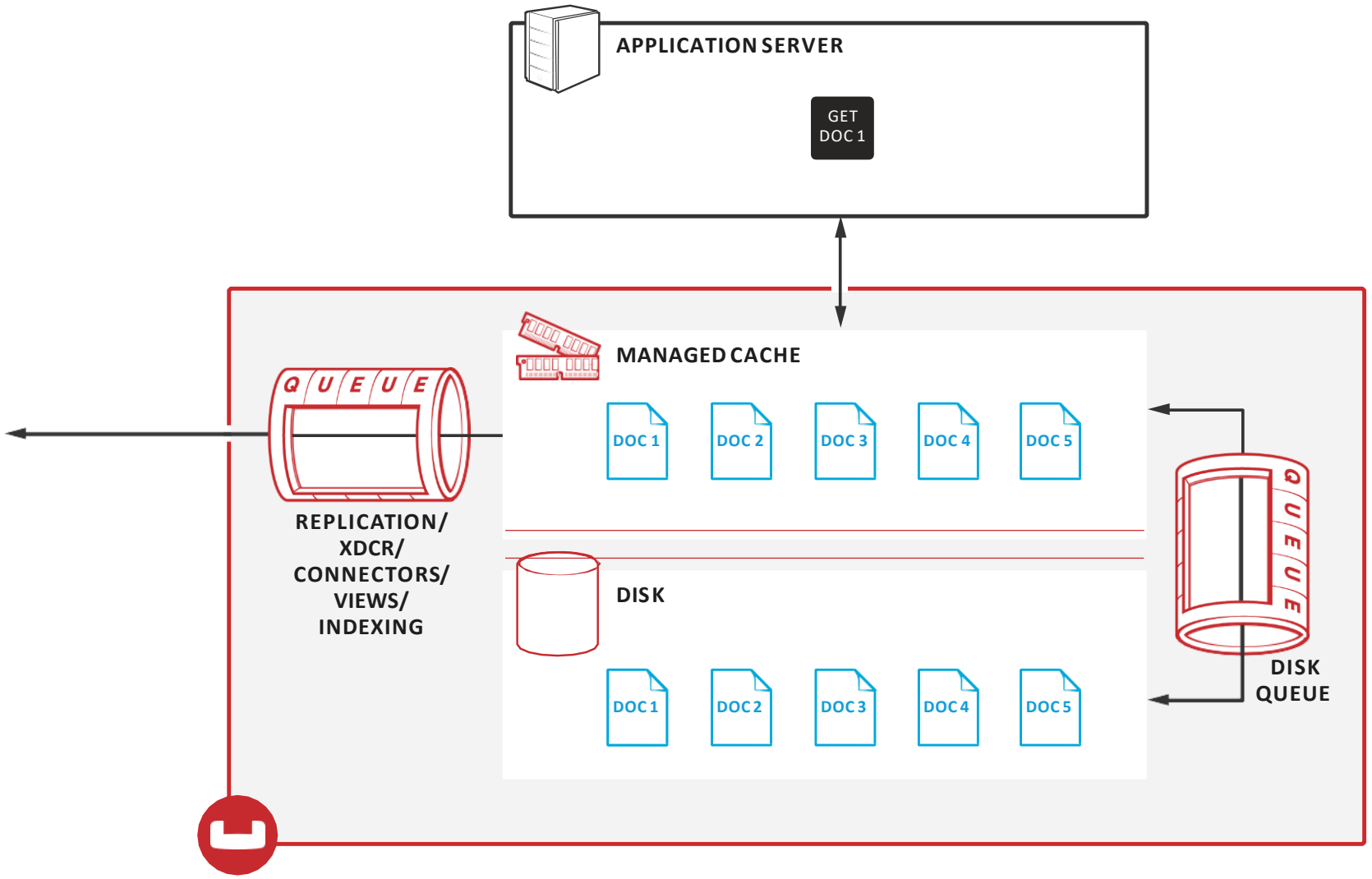
Data Service: Read Operation



Single-node type means easier administration and scaling

- Reads out of cache are extremely fast
- No other process/system to communicate with
- Data connection is a TCP-binary protocol

Data Service: Cache Miss



Single-node type means easier administration and scaling

- Layer consolidation means 1 single interface for App to talk to and get its data back as fast as possible
- Separation of cache and disk allows for fastest access out of RAM while pulling data from disk in parallel

Write Request

- When Couchbase receives a request to write a document, the following occurs:
- Every server in a Couchbase cluster has its own managed object cache. The client writes a document into the cache, and the server sends the client a confirmation. By default, the client does not have to wait for the server to persist and replicate the document as it happens asynchronously.
- The document is added into the intra-cluster replication queue to be replicated to other servers within the cluster.
- The document is added into the disk-write queue to be asynchronously persisted to disk. The document is persisted to disk after the disk-write queue is flushed.
- After the document is persisted to disk, it’s replicated to other clusters using XDCR and eventually indexed.

Key-value data access

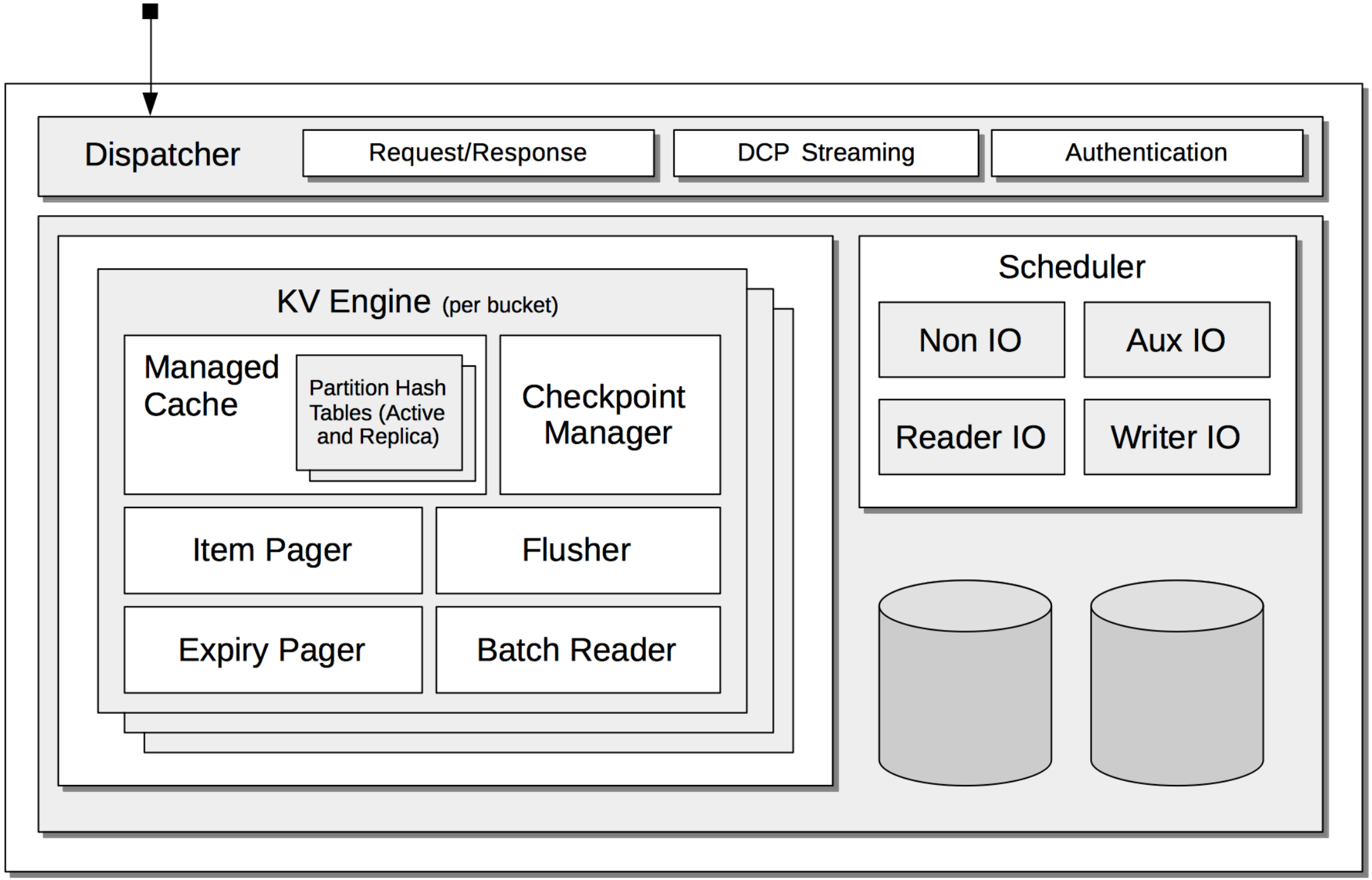
- While Couchbase is document database, at its heart is a distributed key-value (KV) store.
- A KV store is an extremely simple, schema-less approach to data management that,
- Stores a unique ID (key) together with a piece of arbitrary information (value); it may be thought of as a hash map or dictionary.
- The KV store itself can accept any data, whether it be a binary blob or a JSON document, and Couchbase features such as N1QL make use of the KV store's ability to process JSON documents.
- Due to their simplicity, KV operations execute with extremely low latency, often sub-millisecond.
- The KV store is accessed using simple CRUD (Create, Read, Update, Delete) APIs, and provide the simplest interface when accessing documents using their IDs.

Key-value data access

- The KV store contains the authoritative, most up-to-date state for each item.
- Query, and other services, provide eventually consistent indexes, but querying the KV store directly will always access the latest version of data.
- Applications use the KV store when speed, consistency, and simplified access patterns are preferred over flexible query options.
- All KV operations are atomic, which means that Read and Update are individual operations.
- In order to avoid conflicts that might arise with multiple concurrent updates to the same document, applications may make use of Compare-And-Swap (CAS), which is a per-document checksum that Couchbase modifies each time a document is changed.

Understanding the Data Service

- Most fundamental of all Couchbase services, providing access to data in memory and on disk
- Memory allocation for the Data Service is configurable.
- Data Service must run on at least one node of every cluster.


Dispatcher
Request/Response
DCP Streaming
Authentication
KV Engine (per bucket)
Managed Cache
Partition Hash Tables (Active and Replica)
Checkpoint Manager
Item Pager
Flusher
Expiry Pager
Batch Reader
Scheduler
Non IO
Aux IO
Reader IO
Writer IO

Understanding the Data Service

- Dispatcher: Manages networking, by handling each Request for data, and providing the Response.
  - Also streams data to other nodes within the cluster, and to other clusters, by means of the DCP protocol;
  - Handles Authentication.

Understanding the Data Service

- KV Engine: A collection of facilities that is provided for each bucket on the cluster.

These are:

- Managed Cache: Memory allocated for the bucket, according to an established quota.
  - Contains Partition Hash Tables, whereby the location of bucket-items, in memory and on disk, on different nodes across the cluster, is recorded.
  - When written, items enter the cache, and subsequently are placed onto a replication queue, so as to be replicated to one or more other nodes; and (in the case of items for Couchbase buckets) onto a disk queue, so as to be written to disk.
- Checkpoint Manager: Keeps track of item-changes, using data structures named checkpoints.
  - Changes already made to items in memory, but not yet placed on the replication and disk queues, are recorded.

Understanding the Data Service

- Item Pager: Ejects from memory items that have not recently been used, in order to free up space, as required.
- Flusher: Deletes every item in the bucket.
- Expiry Pager: Scans for items that have expired, and erases them from memory and disk; after which, a tombstone remains for a default period of 3 days.
  - The expiry pager runs every 60 minutes by default: for information on changing the interval, see cbepctl set flush_param.
- Batch Reader: Enhances performance by combining changes made to multiple items into batches, which are placed on the disk queue, to be written to disk.

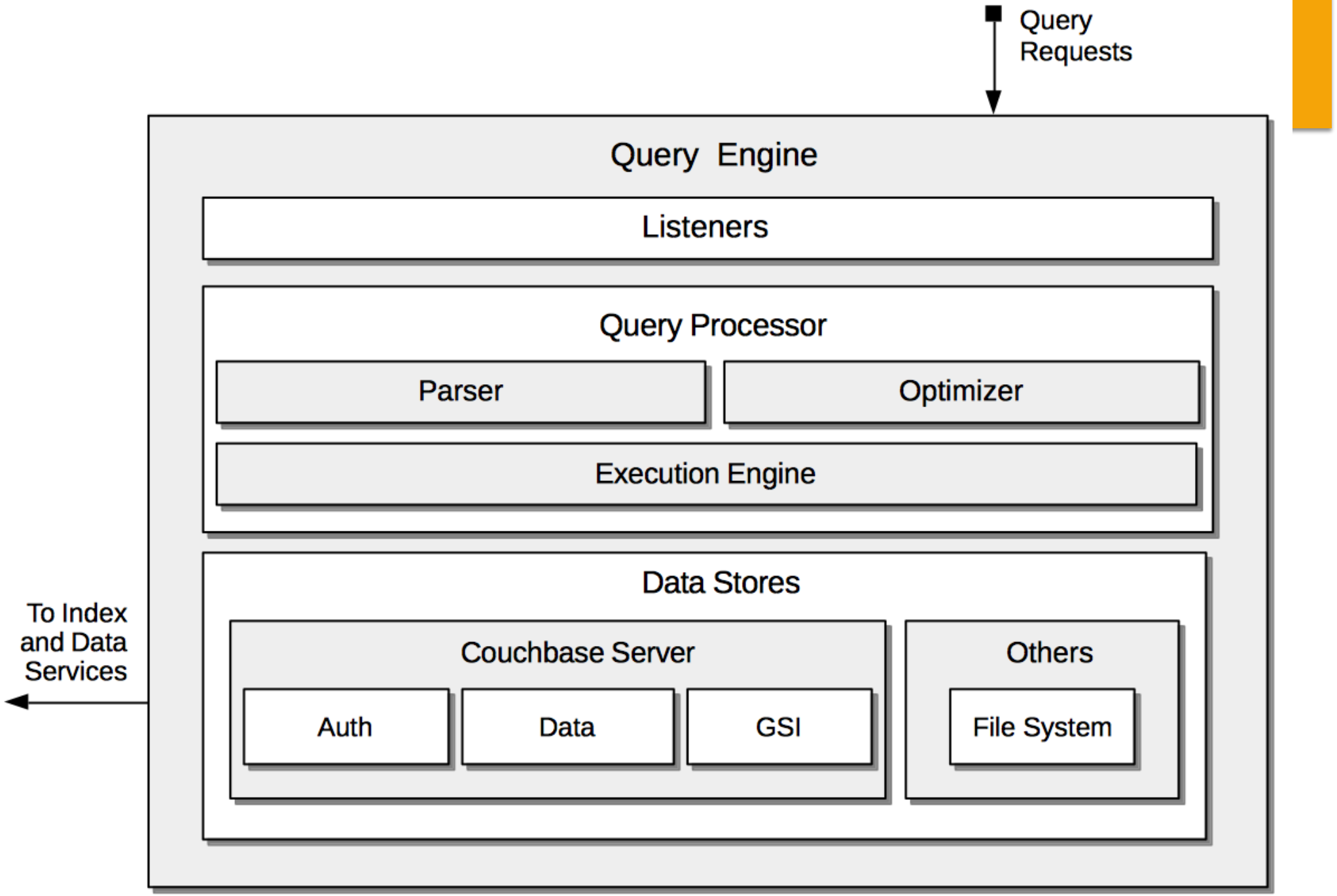
Understanding the Data Service

Scheduler: A pool of threads, mainly purposes for handling I/O. The threads are divided into four kinds, which run independently of and without effect on one another:

- Non IO: Tasks private to the scheduler that do not require disk-access; including connection-notification, checkpoint removal, and hash-table resizing.
- Aux IO: Fetch, scan, and backfill tasks.
- Reader IO: Threads that read information from disk.
- Writer IO: Threads that write information to disk.

Query service

- Engine for processing N1QL queries and follows the same scalability paradigm that all the services use which allows, allowing the user to scale query workloads independently of other services as needed.
- Non-1st normal form query language (N1QL, pronounced “nickel”) is the first NoSQL query language to leverage the flexibility of JSON with the expressive power of SQL.
- N1QL is an implementation of the SQL++ standard.
- N1QL enables clients to access data from Couchbase using SQL-like language constructs, as N1QL’s design was based on SQL.
- Includes a familiar data definition language (DDL), data manipulation language (DML), and query language statements, but can operate in the face of NoSQL database features such as key-value storage, multi-valued attributes, and nested objects.

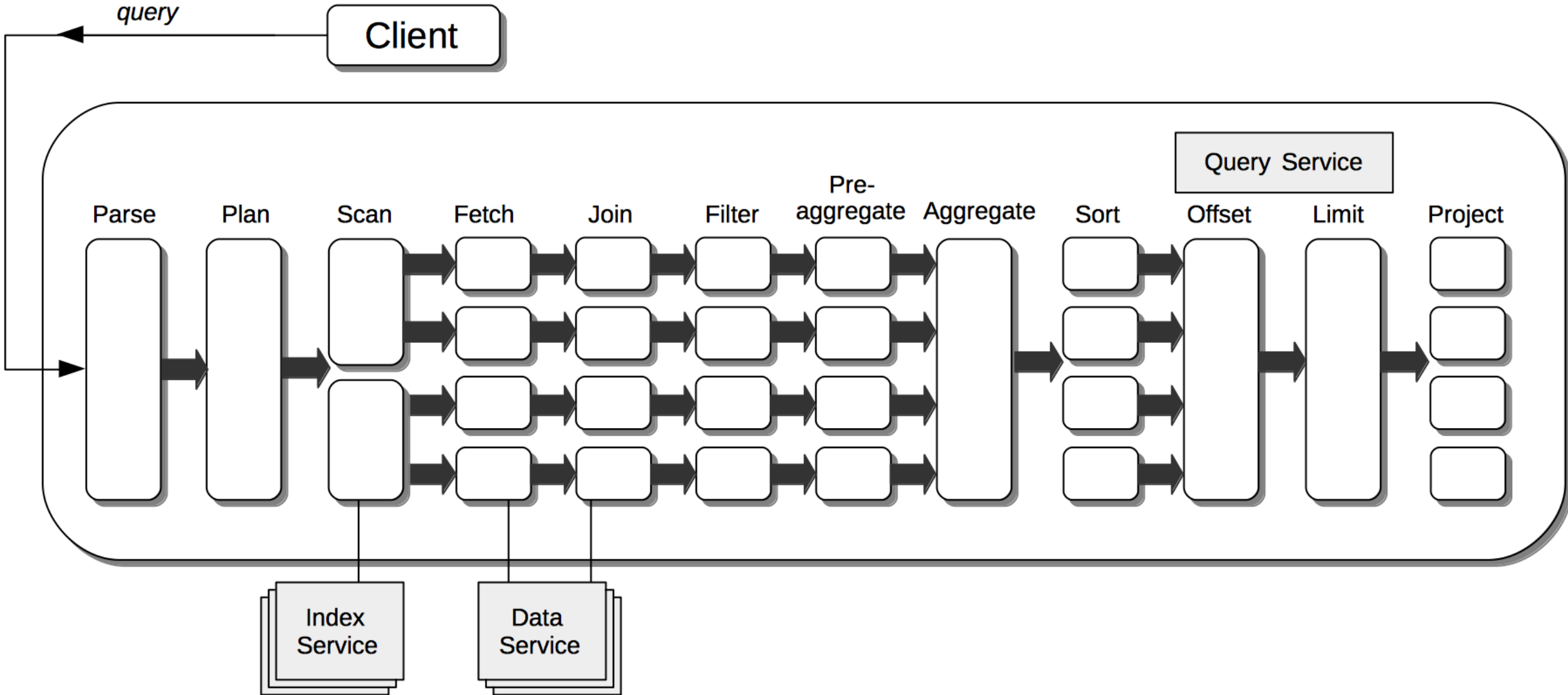

Query
Requests
Query Engine
Listeners
Query Processor
Parser
Optimizer
Execution Engine
Data Stores
Couchbase Server
Auth
Data
GSI
Others
File System
To Index
and Data
Services

Query service

Principal components are:

- Listeners: Concurrent query requests are received on ports 8093 and 18093
- Query Processor: Responsible for applying the Parser to incoming queries, in order to determine whether each is a valid statement.
  - Also employs the Optimizer, which evaluates available execution paths, so determining the path of lowest latency; generates a query-execution plan that uses the lowest-latency path; and assembles the plan into a series of operators.
  - Execution Engine receives the operators, and executes them — in parallel where possible.
- Data Stores: Provides access to various data-sources.
  - The Couchbase Server store is used to access the data and indexes on Couchbase Server, and to handle authentication.
  - Other data stores are also included, such as the store for the local filesystem

Query Execution



Query service

- The client's N1QL query is shown entering the Query Service at the left-hand side.
- The Query Processor performs its Parse routine, to validate the submitted statement, then creates the execution Plan.
- Scan operations are then performed on the relevant index, by accessing the Index Service.
- Next, Fetch operations are performed by accessing the Data Service, and the data duly returned is used in Join operations.
- The Query Service continues by performing additional processing, which includes Filter, Aggregate, and Sort operations.
- Note the degree of parallelism with which operations are frequently performed, represented by the vertically aligned groups of right-pointing arrows

N1QL Queries

- N1QL provides a rich set of features that let users retrieve, manipulate, transform, and create JSON document data.
- Key features include a powerful SELECT statement that extends the functionality of the SQL SELECT statement to work with JSON documents.
- Of particular importance are the USE KEYS, NEST, and UNNEST sub-clauses of the FROM clause in N1QL as well as the MISSING boolean option in the WHERE clause.

N1QL Queries

N1QL supports standard SELECT, FROM, WHERE, GROUP BY clauses as well as JOIN capabilities

```
SELECT c.name, o.order_date
FROM customers AS c
LEFT OUTER JOIN orders AS o
  ON c.custid = o.custid
WHERE c.custid = "C41";
```

```
{
 "results": [
    {
      "name": "R. Duvall",
      "order_date": "2017-04-29"
    },
    {
      "name": "R. Duvall",
      "order_date": "2017-09-02"
    }
  ]
}
```

N1QL Queries

Use UNNEST to extract individual items from a nested JSON array

```
SELECT o.orderno,
       i.itemno AS item_number,
       i.qty AS quantity
FROM orders AS o
UNNEST o.items AS i
WHERE i.qty > 100
```


```
{
  "results": [
    {
      "orderno": 1002,
      "item_number": 680,
      "quantity": 150
    },
    {
      "orderno": 1005,
      "item_number": 347,
      "quantity": 120
    },
    {
      "orderno": 1006,
      "item_number": 460,
      "quantity": 120
    }
  ]
}"
```

N1QL Queries

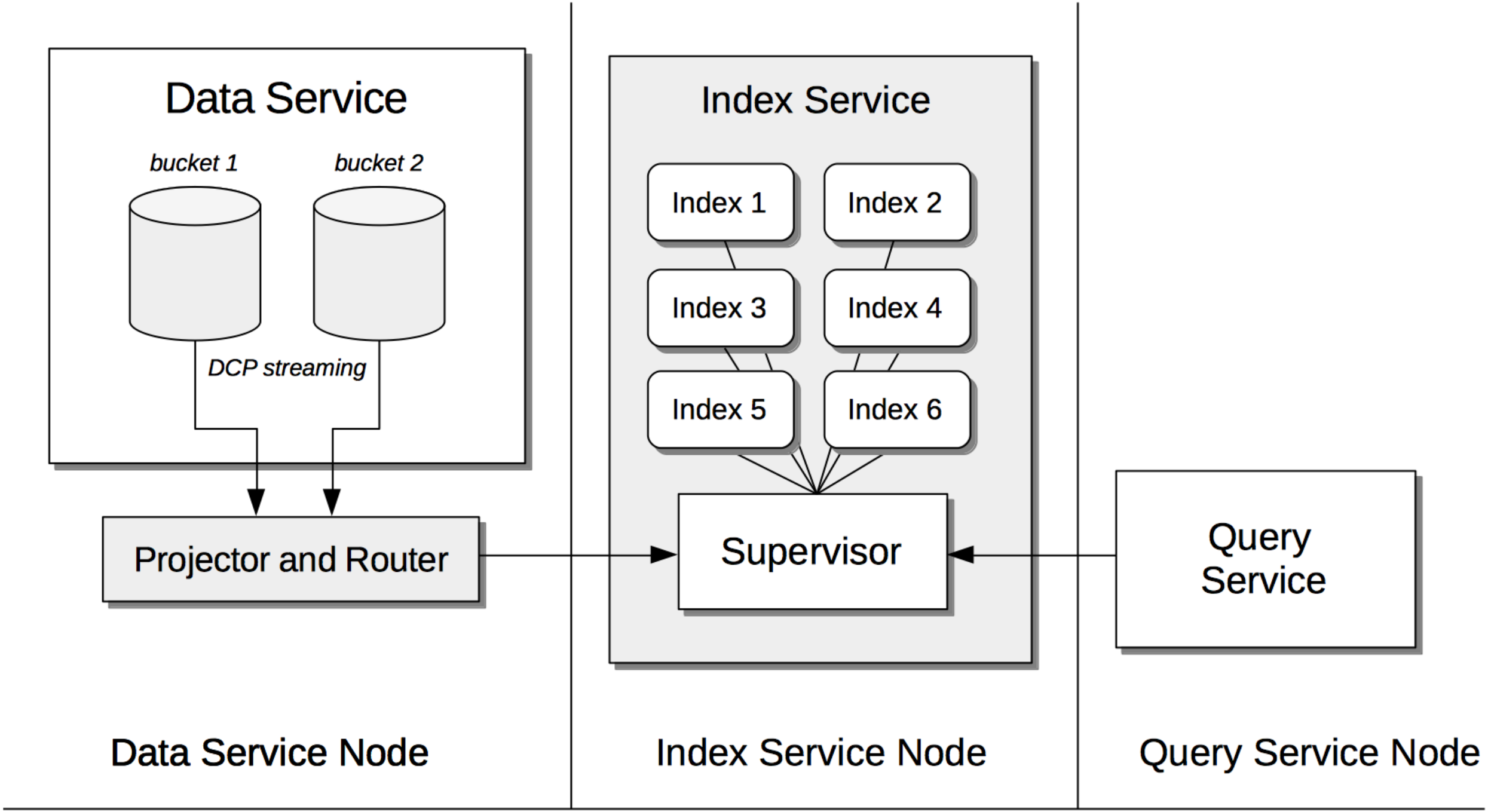
Use MISSING boolean keyword in WHERE clause to adapt queries when a schema has changed or lacks specific keys

```
SELECT o.orderno, SUM(o.cost)
AS cost
FROM orders AS o
WHERE o.cost IS NOT MISSING
GROUP BY o.orderno;
```


```
{
  "results": [
    {
      "orderno": 1002,
      "cost": 220,
    },
    {
      "orderno": 1005,
      "cost": 623,
    }
  ]
}
```

Index service for N1QL

- Secondary indexing is an important part of making queries run efficiently and Couchbase provides a robust set of index types and management options.
- Responsible for all of the maintenance and management tasks of indexes, known as Global Secondary Indexes (GSI).
- Index service monitors document mutations to keep indexes up to date, using the database change protocol stream (DCP) from the data service.
- Distinct from the query service, allowing their workloads to be isolated from one another where needed.


Data Service
bucket 1
bucket 2
DCP streaming
Projector and Router
Index Service
Index 1
Index 2
Index 3
Index 4
Index 5
Index 6
Supervisor
Query Service
Data Service Node
Index Service Node
Query Service Node

Index service

Data Service: Uses the DCP protocol to stream data-mutations to the Projector and Router process, which runs as part of the Index Service, on each Data Service node.

Projector and Router: Provides data to the Index Service, according to the index-definitions provided by the Index Service Supervisor.

When the Projector and Router starts running on the Data Service-node, the Data Service streams to the Projector and Router copies of all mutations that occur to bucket-items. Prior to the creation of any indexes, the Projector and Router takes no action.

Index service

When an index is first created, the Index Service Supervisor contacts the Projector and Router; and passes to it the corresponding index-definitions.

The Projector and Router duly contacts the Data Service, and extracts data from the fields specified by the index-definitions. It then sends the data to the Supervisor, so that the index can be populated.

Subsequently, the Projector and Router continuously examines the stream of mutations provided by the Data Service.

When this includes a mutation to an indexed field, the mutated data is passed by the Projector and Router to the Supervisor, and the index thereby updated.

Index service

Supervisor: The main program of the Index Service; which passes index-definitions to the Projector and Router, creates and stores indexes, and handles mutations sent from the Projector and Router.

Query Service: Passes to the Supervisor clients' create-requests and queries, and handles the responses

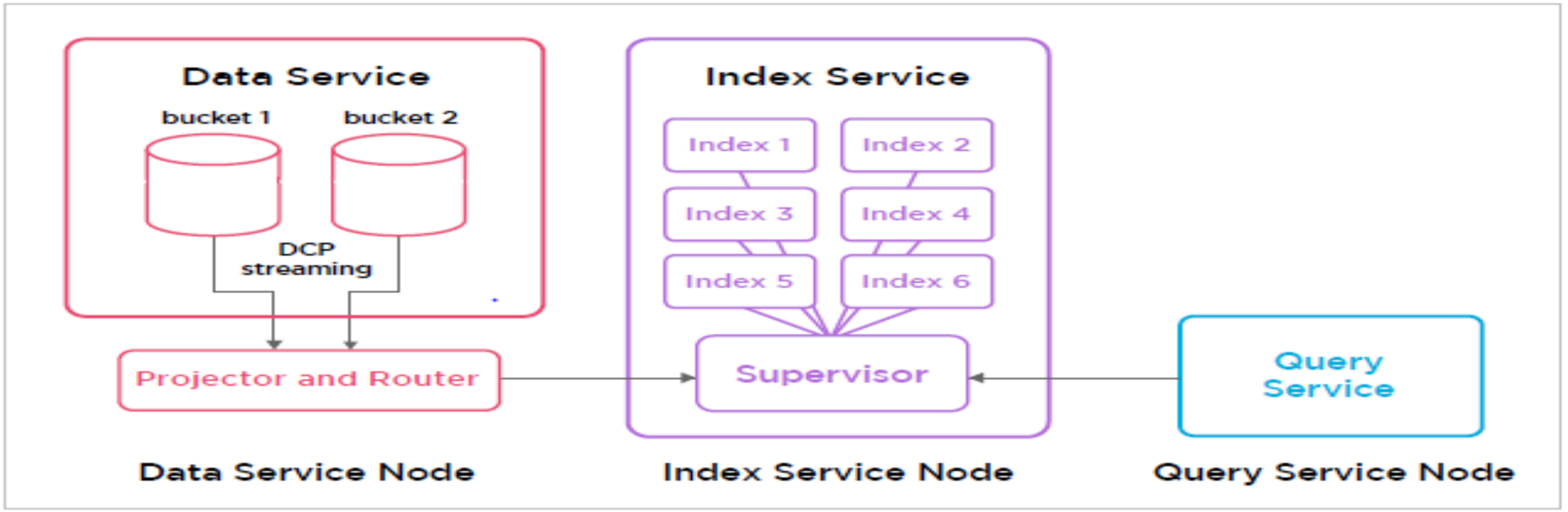
Index service for N1QL

Following are some of the types of indexes supported by the index service:

- **Primary** – indexes whole bucket on document key
- **Secondary** – indexes a scalar, object, or array using a key-value
- **Composite/Covered** – multiple fields stored in an index
- **Functional** – secondary index that allows functional expressions instead of a simple key-value
- **Array** – an index of array elements ranging from plain scalar values to complex arrays or JSON objects nested deeper in the array
- **Adaptive** – secondary array index for all or some fields of a document without having to define them ahead of time

Query optimization

- Query service uses a query optimizer to take advantage of indexes that are available.
- Index nodes can handle much of the data aggregation pipeline as well, so that less data is sent back to the query node for processing.



Search service

- Search service is an engine for performing Full-Text Searches (FTS) on the JSON data stored within a bucket.
- FTS lets you create, manage, and query inverted indexes for searching of free-form text within a document.
- Service provides analyzers that perform several types of operations including multi-language tokenization, stemming, and relevance scoring.
- Search nodes incorporate both an indexer and query processor, much like the query and index services, except these don’t run on separate nodes – both workloads run on each search node.

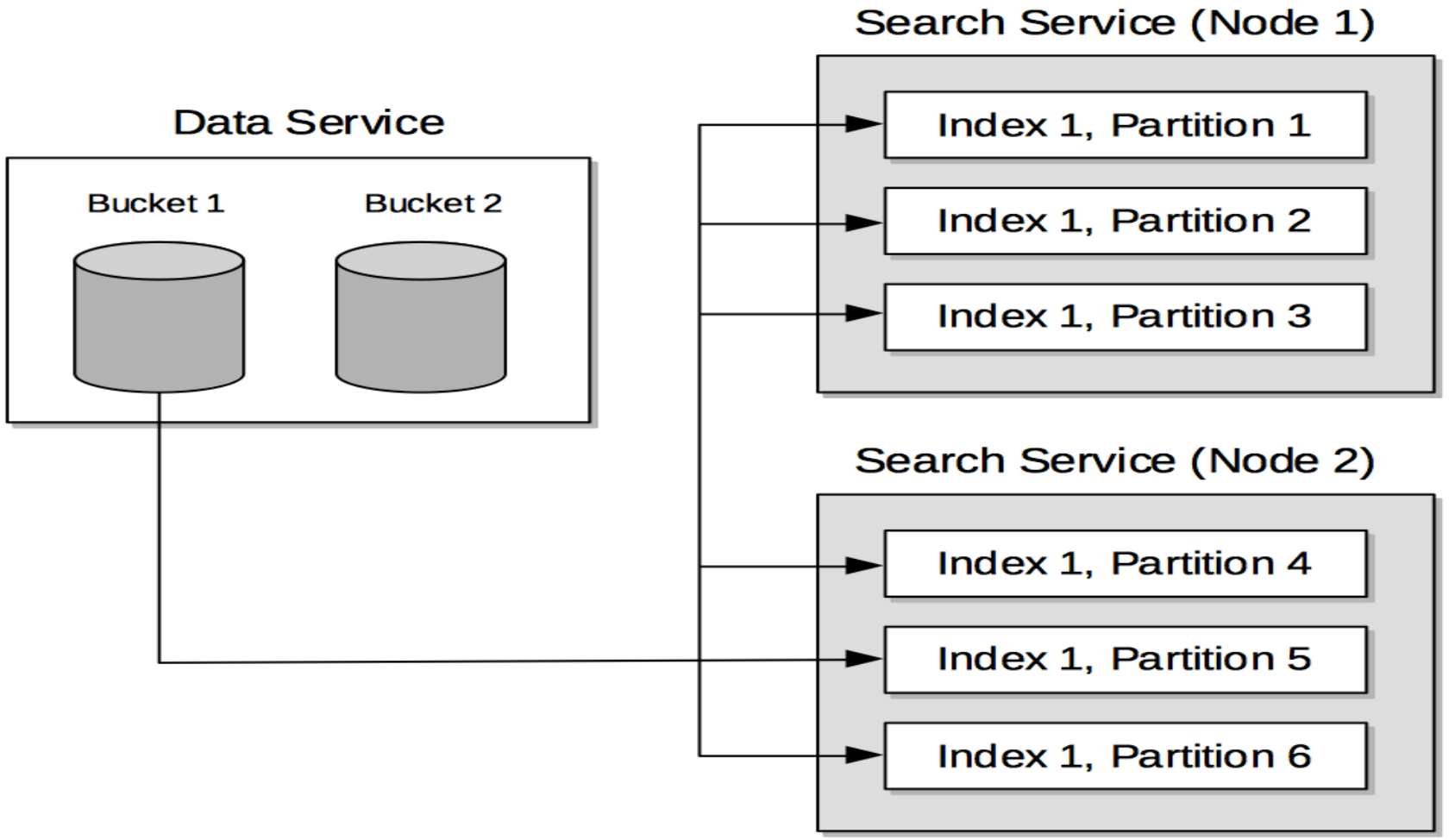
Search service

- Search Service provides extensive capabilities for natural-language querying. These include:
- Language-aware searching; allowing users to search for, say, the word beauties, and additionally obtain results for beauty and beautiful.
- Scoring of results, according to relevancy; allowing users to obtain result-sets that only contain documents awarded the highest scores.
- This keeps result-sets manageably small, even when the total number of documents returned is extremely large.
- Fast indexes, which support a wide range of possible text-searches.
- Indexes that the Search Service creates and uses are entirely separate from and different to those of the Index Service

Search Service Architecture

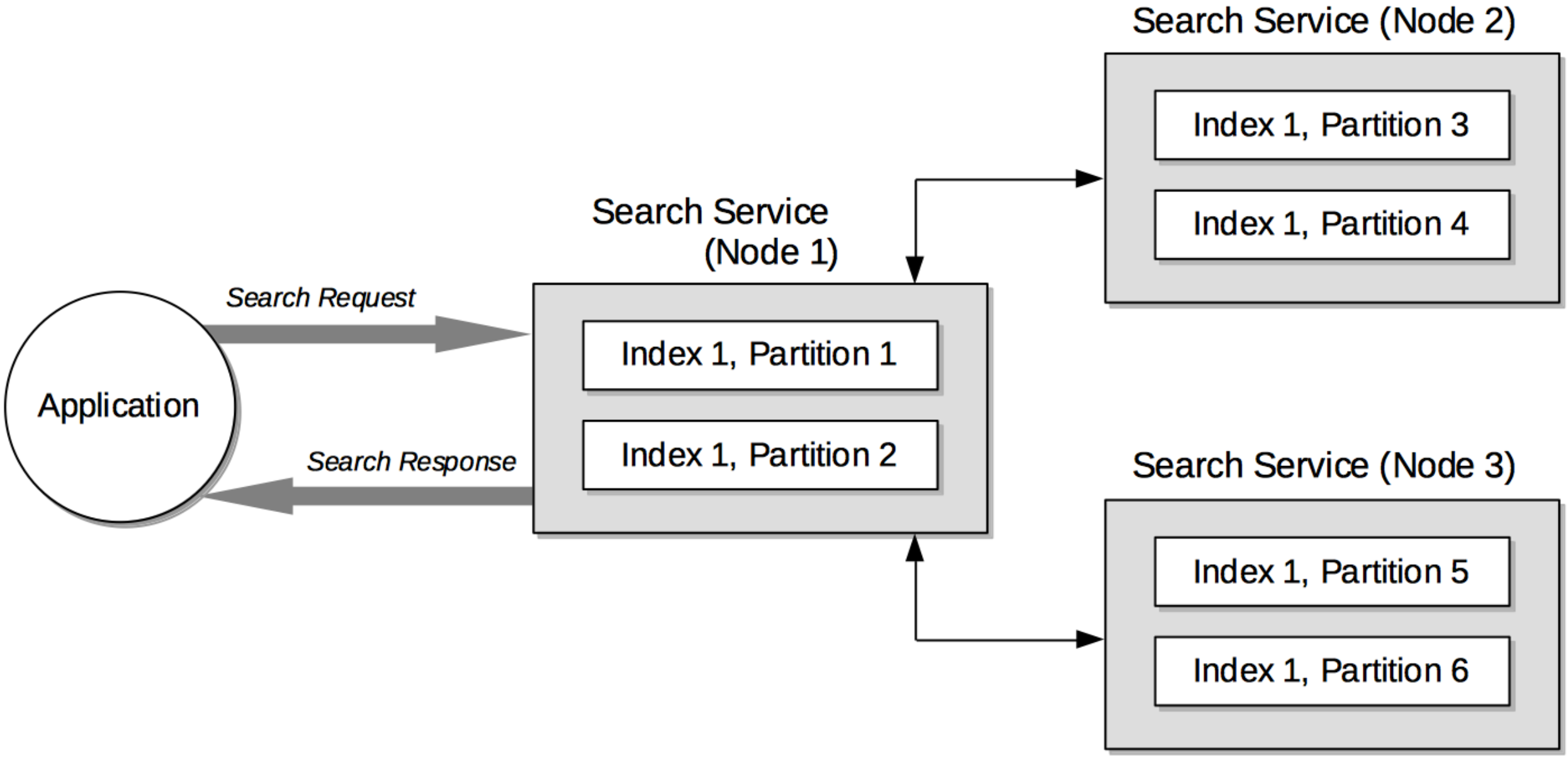
- Depends on the Data Service
- Data Service uses the DCP protocol to stream data-mutations as batches; from the producer to multiple consumers (vBuckets), instantiated across the search nodes.
- When a search index is created by means of the Search Service, search-index data-handling for the vBuckets is divided equally among the established search-index partitions.
- For example, if the number of vBuckets is 120, and the number of search-index partitions chosen is 6, each search-index partition holds data for 20 vbuckets.

Search Service Architecture



Search Service Architecture

- All available Search Service nodes in the cluster are individually searchable.
- When one particular Search Service node is chosen for a search request, it assumes the role of coordinator; and is thereby responsible for applying the search request to the other Search Service nodes, and for gathering and returning results.


Search Service (Node 2)
Index 1, Partition 3
Index 1, Partition 4
Search Service
(Node 1)
Search Request
Application
Index 1, Partition 1
Index 1, Partition 2
Search Response
Search Service (Node 3)
Index 1, Partition 5
Index 1, Partition 6

Search Service Architecture

- The application makes a search request to a specific Search Service node (here, Node 1). This node assumes the role of coordinator.
- The coordinator scatters the search request to all other search-index partitions (here, Node 2 and Node 3) in the cluster.
- Once all the returned data is gathered, the coordinator applies filters as appropriate, and returns the final results to the user.

Search service

- As with the other services, data nodes use the DCP stream to send mutations to the FTS indexer process for index updating whenever data changes.
- Index creation is highly configurable through a JSON index definition file, Couchbase SDK, or through a graphical web interface as part of the administration console.
- Documents can be indexed differently depending on a document type attribute, a document ID, or the value of a designated attribute.
- Each index definition can be assigned its own set of analyzers and specific analyzers can be applied to indexes for a subset of fields.
- Indexes are tied to a specific bucket, but it is possible to create virtual index aliases that combine indexes from multiple buckets into a single seamless index.
- These aliases also allow application developers to build new indexes and quickly change over to new ones without having to take an index offline.

Search service

- Searching and indexing use the same set of analyzers for finding matching data.
- All data, when indexed, flows through the analyzer steps as defined by the index.
- Then search requests are received and passed through the same steps – for example, tokenization, removing stop words, and stemming terms.
- These analyzed search requests are then looked up by the indexer in the index and matches are returned. The results include the source request, list of document IDs, and relevance scoring.
- Other indexing and search-time options provide fine-grained control over indexing more or less information depending on the use case.
- For example, text snippets may also be stored in the index and included in the search response so that retrieving full documents from the data service is not required.
- Couchbase developed Bleve, the open source Go-based search project, for the FTS capabilities, including language support, scoring, etc.

Eventing service

- Eventing service supports custom server-side functions (written in JavaScript) that are automatically triggered using an Event-Condition-Action model.
- These functions receive data from the DCP stream for a particular bucket and execute code when triggered by data mutations.
- Similar to other services, the eventing service scales linearly and independently.
- Code processes the source data and commits it as a new or updated document in another bucket.
- The core of eventing functions is a Google V8 execution container.
- Functions inherit support for most of the standard ECMAScript constructs that are available through V8.
- Code for functions is written in a web-based JavaScript code editor and features an extensive in-browser debugging environment.

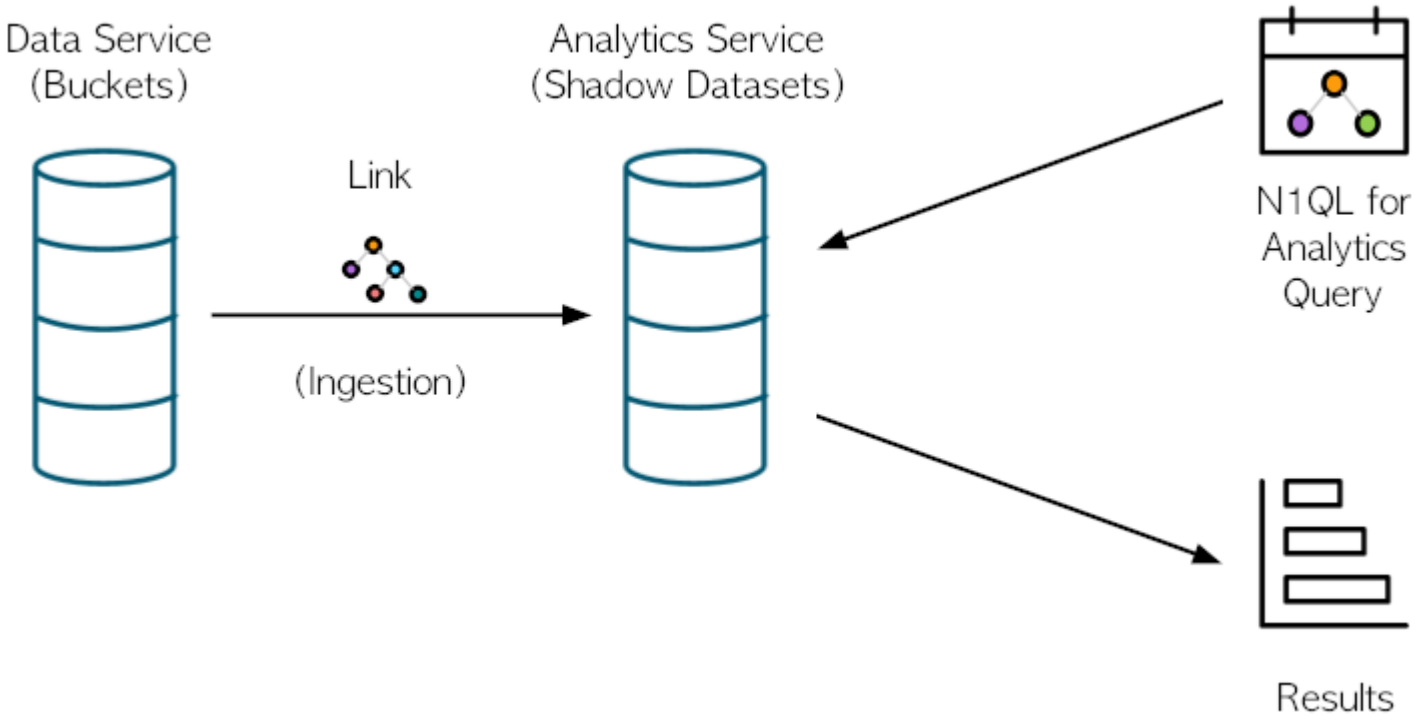
Analytics

- Provides an ad hoc querying capability without the need for indexes, bringing a hybrid operational and analytical processing (HOAP) model for real-time and operational analytics on the JSON data within Couchbase.
- Uses the same N1QL language as the query service.
- Designed to efficiently run complex queries over a large number of documents, including query features such as ad hoc join, set, aggregation, and grouping operations.
- In a typical operational or analytical database, any of these kinds of queries may result in inefficiencies: long running queries, I/O constraints, high memory consumption, and/or excessive network latency due to data fetching and cross-node coordination.

Analytics

- Because the service supports efficient parallel query processing and bulk data handling, and runs on separate nodes, it is often preferable for expensive queries, even if the queries are predetermined and could be supported by an operational index.
- With traditional database technology it is important to segregate operational and analytic workloads.
- This is usually done by batch exporting of data from operational databases into an analytic database that does further processing.
- Couchbase provides both the operational database as well as a scalable analytics database – all in one NoSQL platform.

Analytics Service



Analytics Service

- Enables to create shadow copies of the data to be analyzed.
- When the shadowed Analytics data is linked to the operational data, changes in the operational data are reflected in your Analytics data in real time.
- Can then query the Analytics data without slowing down the operational Data or Query services.
- Can add more Analytics nodes to reduce Analytics query times.

Advantages of analytics

- Analytics approach has significant advantages compared to the commonly employed alternatives:
- Common data model: Couchbase Analytics natively supports the same rich, flexible-schema document data model used for your operational data - you don't have to force your data into a flat, predefined, relational model to analyze it.
- Workload isolation: Operational query latency and throughput are protected from slow-downs due to your analytical query workload - but without the complexity of operating a separate analytical database.
- High data freshness: Couchbase Analytics uses DCP, a fast memory-to-memory protocol that Couchbase Server nodes use to synchronize data among themselves - so Analytics runs on data that's extremely current, without ETL (extract, transform, load) or other hassles and delays.
- In Couchbase Server 6.6 and later, can also create remote links to analyze data on remote Couchbase clusters, and also external links, to analyze data from external sources such as Amazon S3

When to Use Analytics

- Use the Query service for operational queries — for example, the front-end queries behind every page display or navigation.
- Use the Analytics service when you don't know every aspect of the query in advance — for example, if the data access patterns change frequently, or you want to avoid creating an index for each data access pattern, or you want to run ad hoc queries for data exploration or visualization.
- Use the Full Text Search service when you want to take advantage of natural-language querying.

Node-level architecture

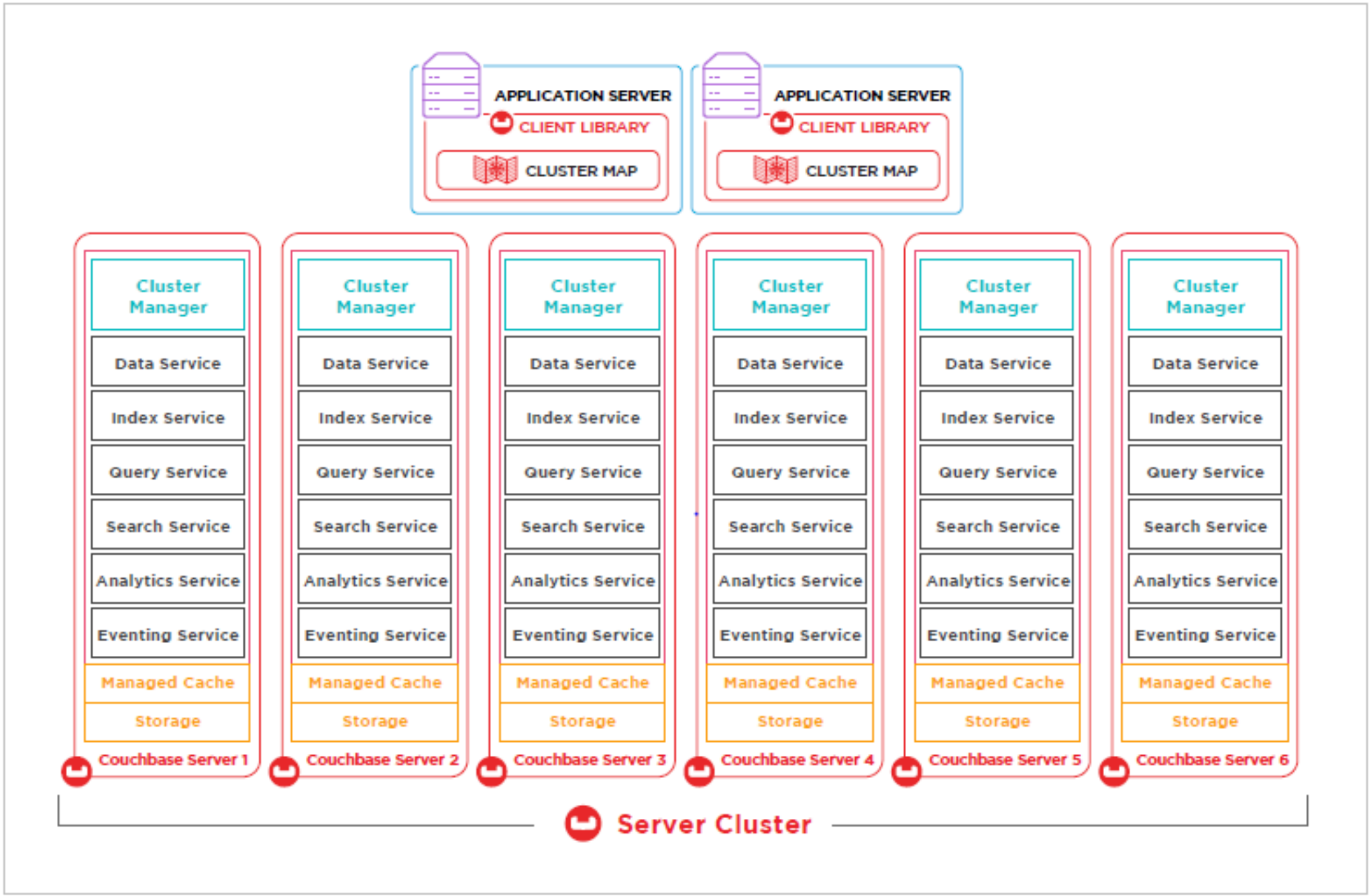
- A Couchbase cluster consists of a group of interchangeable, largely self-sufficient nodes that operate in a peer-to-peer topology.
- There is just one Couchbase node type, though the services running on that node can be managed as required
- Having a single-node type greatly simplifies the installation, configuration, management, and troubleshooting of a Couchbase cluster, both in terms of what you must do as a human operator and what the automatic management needs to do.
- There is no concept of master nodes, slave nodes, config nodes, name nodes, or head nodes.

Node-level architecture

- Components of a Couchbase node include the cluster manager and, optionally, the data, query, index, analytics, search, and eventing services.
- There is also the underlying managed cache and storage components.
- By dividing up potentially conflicting workloads in this way, a Couchbase node can achieve maximum throughput and resource utilization and minimum latency.
- Nodes can be added or removed easily through a rebalance process, which redistributes the data evenly across all nodes.
- Rebalance process is done online and requires no application downtime, and can be initiated at the click of a button or one command on the command line.

Cluster architecture

- A cluster consists of one or more instances of Couchbase Server, each running on an independent node. Data and services are shared across the cluster.
- Various services that Couchbase provides are also fed data through the DCP stream, which is internally used for sending new/changed data to these services as well as providing the basis for keeping data in sync between nodes in the cluster.


APPLICATION SERVER
CLIENT LIBRARY
CLUSTER MAP
APPLICATION SERVER
CLIENT LIBRARY
CLUSTER MAP
Cluster Manager
Data Service
Index Service
Query Service
Search Service
Analytics Service
Eventing Service
Managed Cache
Storage
Couchbase Server 1
Couchbase Server 2
Couchbase Server 3
Couchbase Server 4
Couchbase Server 5
Couchbase Server 6
Server Cluster

Cluster/node configuration

- When Couchbase is being configured on a node, it can be specified either as its own, new cluster, or as a participant in an existing cluster.
- Thus, once a cluster exists, successive nodes can be added to it.
- When a cluster has multiple nodes, the Couchbase cluster manager runs on each node: this manages communication between nodes, and ensures that all nodes are healthy.
- Services can be configured to run on all or some nodes in a cluster, and can be added/ removed as warranted by established performance needs
- For example, given a cluster of five nodes, a small dataset might require the data service on only one of the nodes; a large dataset might require four or five.
- Alternatively, a heavy query workload might require the query service to run on multiple nodes, rather than just one.
- This ability to scale services individually promotes optimal hardware resource utilization.

Cluster manager

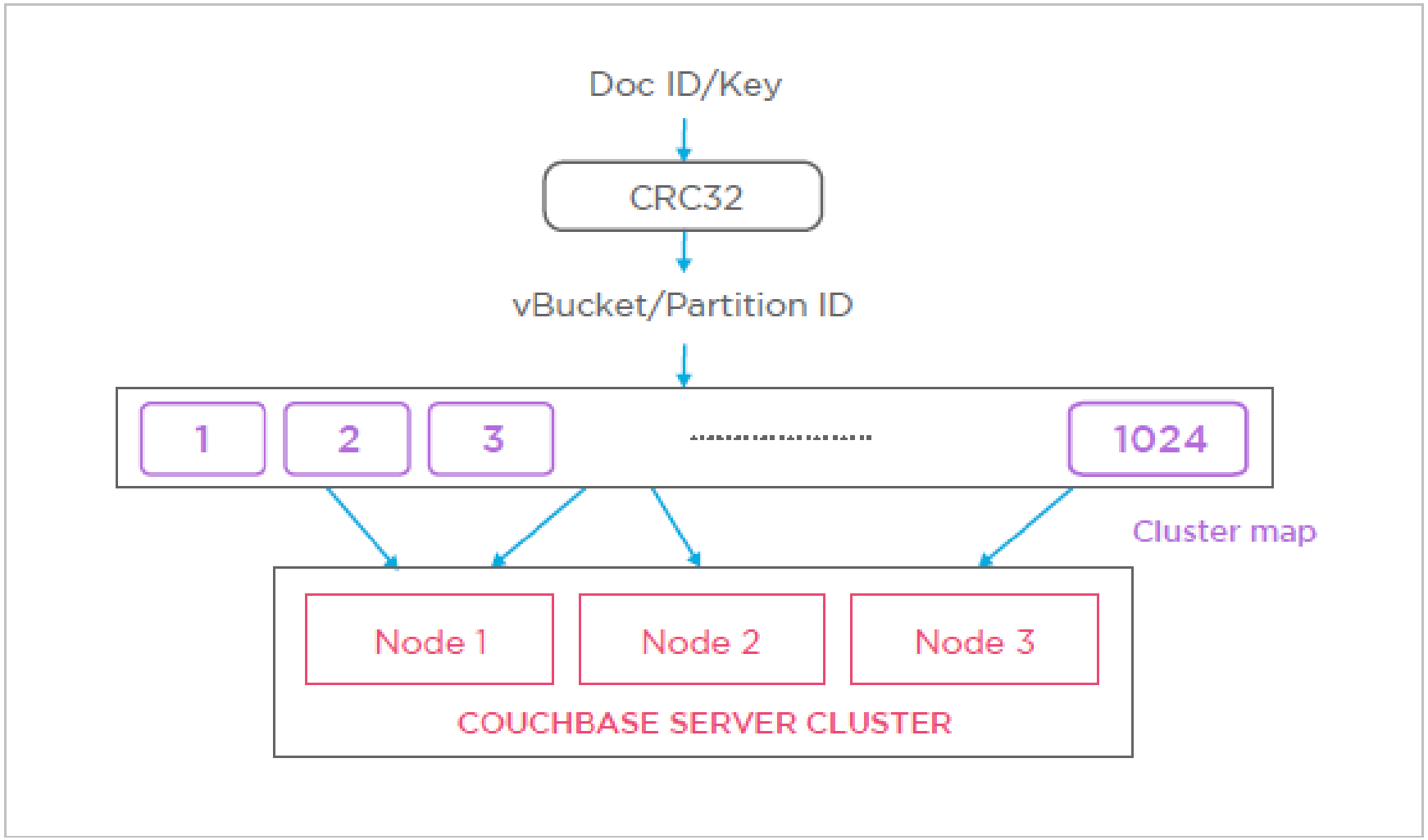
- Cluster manager supervises server configuration and interaction between servers within a Couchbase cluster.
- Critical component that manages replication and rebalancing operations in Couchbase.
- Although the cluster manager executes locally on each cluster node, it elects a clusterwide orchestrator node to oversee cluster conditions and carry out appropriate cluster management functions.
- If a machine in the cluster crashes or becomes unavailable, the cluster orchestrator notifies all other machines in the cluster, and promotes to active status all the replica partitions associated with the server that's down.
- Cluster map is updated on all the cluster nodes and the clients.
- This process of activating the replicas is known as failover.
- Can configure failover to be automatic or manual.
- Additionally, you can trigger failover through external monitoring scripts via the REST API.

Cluster manager

- If the orchestrator node crashes, existing nodes will detect that it is no longer available and will elect a new orchestrator immediately so that the cluster continues to operate without disruption.
- In addition to the cluster orchestrator, there are three primary cluster manager components on each Couchbase node:
- **The heartbeat watchdog** – periodically communicates with the cluster orchestrator using the heartbeat protocol, providing regular health updates for the server. If the orchestrator crashes, existing cluster server nodes will detect the failed orchestrator and elect a new orchestrator.
- **The process monitor** – monitors local data manager activities, restarts failed processes as required, and contributes status information to the heartbeat process.
- **The configuration manager** – receives, processes, and monitors a node's local configuration. It controls the cluster map and active replication streams. When the cluster starts, the configuration manager pulls configuration of other cluster nodes and updates its local copy.

Client connectivity

- To talk to all the services of a cluster, applications use the Couchbase SDK.
- Support is available for a variety of languages including Java, .NET, PHP, Python, Go, Node.js, and C/C++.
- These clients are continually aware of the cluster topology through cluster map updates from the cluster manager.
- They automatically send requests from applications to the appropriate nodes for KV access, query, etc.
- When creating documents, clients apply a hash function (CRC32) to every document that needs to be stored in Couchbase, and the document is sent to the server where it should reside.
- Because a common hash function is used, it is always possible for a client to determine on which node the source document can be found.


Doc ID/Key
CRC32
vBucket/Partition ID
1
2
3
1024
Cluster map
Node 1
Node 2
Node 3
COUCHBASE SERVER CLUSTER

Topology-aware client

- After a client first connects to the cluster, it requests the cluster map from the Couchbase cluster and maintains an open connection with the server for streaming updates.
- The cluster map is shared with all the servers in a Couchbase cluster and with the Couchbase clients. Data flows from a client to the server using the following steps:
- An application interacts with an application, resulting in the need to update or retrieve a document in Couchbase Server.
- The application server contacts Couchbase Server via the smart client SDKs.
- The client SDK takes the document that needs to be updated and hashes its document ID to a partition ID. With the partition ID and the cluster map, the client can figure out on which server and on which partition this document belongs. The client can then update the document on this server.
- When a document arrives in a cluster, Couchbase Server replicates the document, caches it in memory and asynchronously stores it on disk.

Data transport via Database Change Protocol (DCP)

- DCP is the protocol used to stream bucket-level mutations.
- Used for high-speed replication of data as it mutates – to maintain replica vBuckets, global secondary indexes, full-text search, analytics, eventing, XDCR, and backups. Connectors to external services, such as Elasticsearch, Spark, or Kafka are also fed from the DCP stream.
- DCP is a memory-based replication protocol that is *ordering, resumable,* and *consistent*.
- DCP stream changes are made in memory to items by means of a *replication queue*.

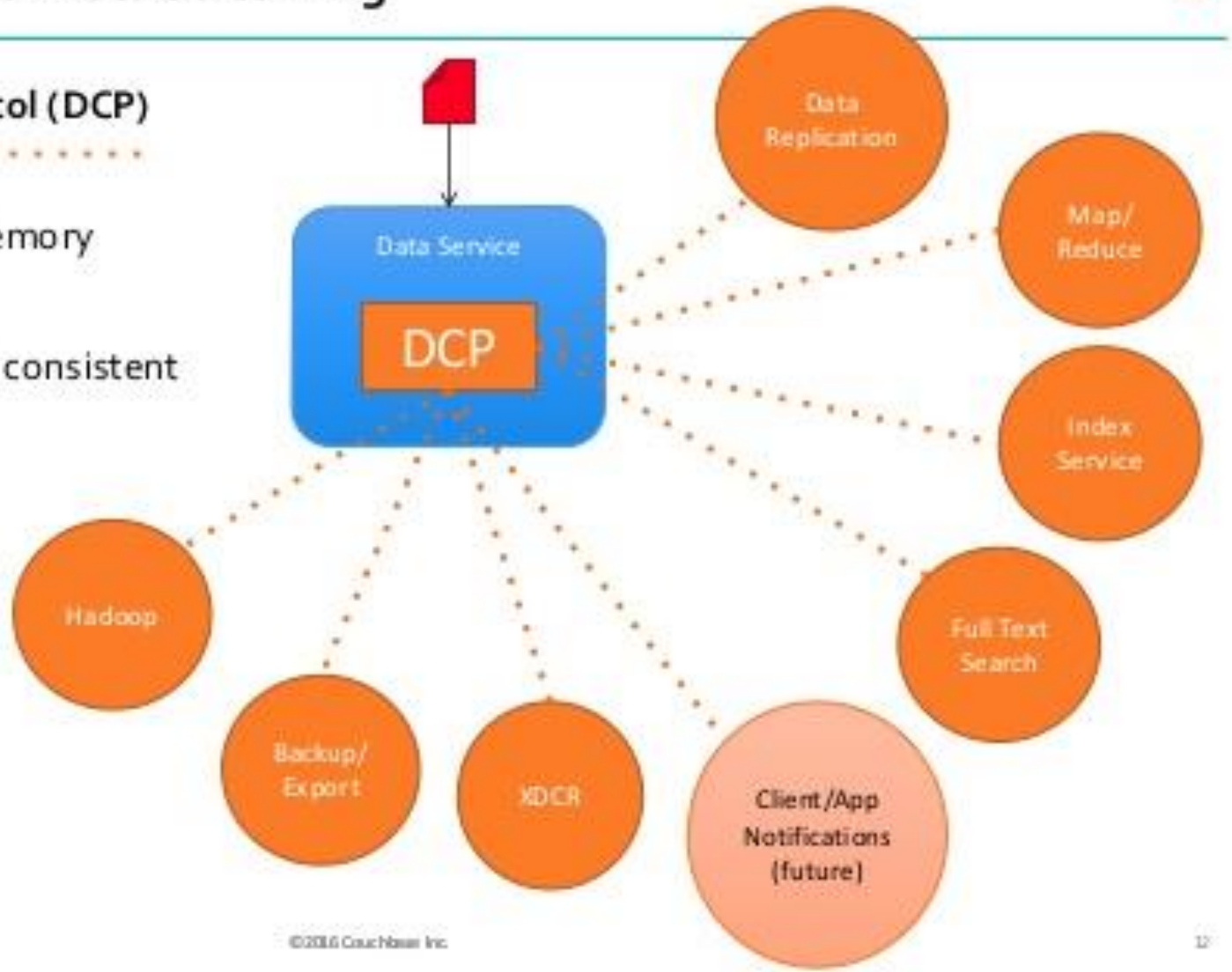
Data transport via Database Change Protocol (DCP)

- An external application client sends the operation requests (read, write, update, delete, query) to access or update data on the cluster.
- These clients can then receive or send data to DCP processes running on the cluster.
- External data connectors, for example, often sit and wait for DCP to start sending the stream of their data when mutations start to occur.
- Whereas an internal DCP client, used by the cluster itself, streams data between nodes to support replication, indexing, cross datacenter replication, incremental backup, and mobile synchronization.
- Sequence numbers are used to track each mutation in a given vBucket, providing a means to access data in an ordered manner or to resume from a given point in time.

Exceptional Data Streaming

- **Database Change Protocol (DCP)**

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

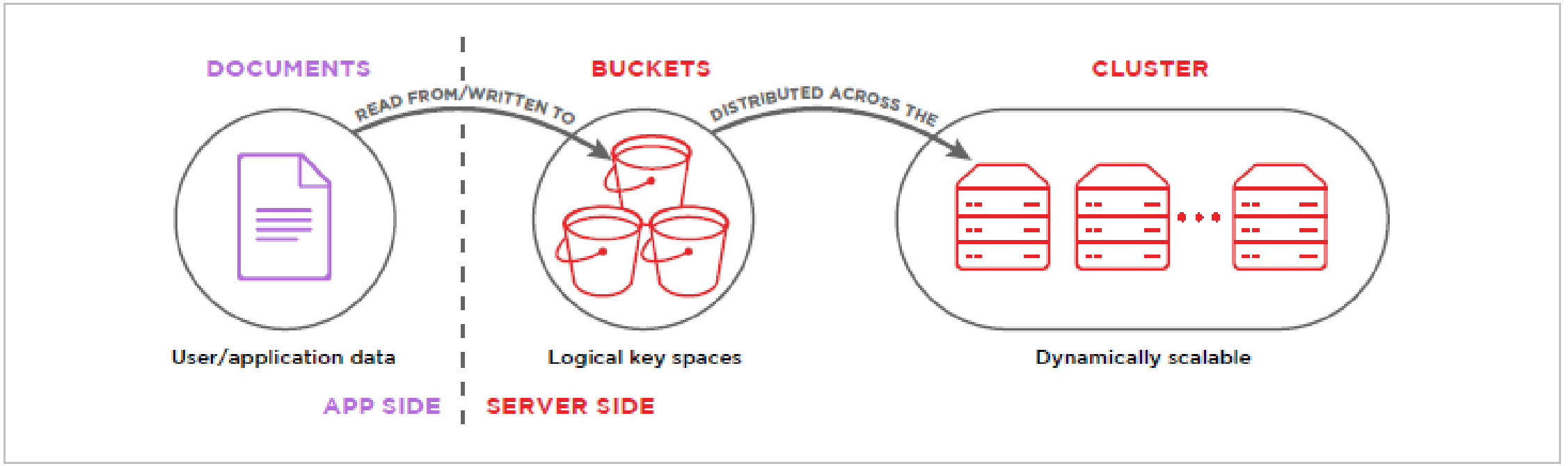


©2016 Couchbase Inc.

Data distribution

- Couchbase partitions data into vBuckets (synonymous to shards or partitions) to automatically distribute data across nodes, a process sometimes known as auto-sharding.
- vBuckets help enable data replication, failover, and dynamic cluster reconfiguration.
- Unlike data buckets, users and applications do not manipulate vBuckets directly.
- Couchbase automatically divides each bucket into 1024 active vBuckets and 1024 replica vBuckets per replica, and then distributes them evenly across the nodes running the data service within a cluster.
- vBuckets do not have a fixed physical location on nodes; therefore, there is a mapping of vBuckets to nodes known as the cluster map.
- Through the Couchbase SDK, the application automatically and transparently distributes the data and workload across these vBuckets.

Data distribution



Rebalancing the cluster

- When the number of servers in the cluster changes due to scaling out or node failures, data partitions must be redistributed.
- ensures that data is evenly distributed across the cluster, and that application access to the data is load balanced evenly across all the servers.
- All Couchbase services are rebalance-aware and follow their own set of internal processes for rebalance as needed.
- Rebalancing is triggered using an explicit action from the admin web UI or through a REST call.

Rebalancing the cluster

- When initiated, the rebalance orchestrator calculates a new cluster map based on the current pending set of servers to be added and removed from the cluster.
- It streams the cluster map to all the servers in the cluster.
- During rebalance, the cluster moves data via partition migration directly between two server nodes in the cluster.
- As the cluster moves each partition from one location to another, an atomic and consistent switchover takes place between the two nodes, and the cluster updates each connected client library with a current cluster map.

Rebalancing the cluster

- Throughout migration and redistribution of partitions among servers, any given partition on a server will be in one of three states:
- **Active** – the server hosting the partition is servicing all requests for this partition.
- **Replica** – the server hosting the partition cannot handle client requests, but can receive replication commands. Rebalance marks destination partitions as replica until they are ready to be switched to active.
- **Dead** – the server is not in any way responsible for this partition.

- The node health monitor receives heartbeat updates from individual nodes in the cluster, updating configuration and raising alerts as required.
- The partition state and replication manager is responsible for establishing and monitoring the current network of replication streams.

Rebalance

- Rebalance redistributes data and indexes among available nodes.
- When one or more nodes have been brought into a cluster (either by adding or joining), or have been taken out of a cluster (either through Removal or Failover), rebalance redistributes data and indexes among available nodes.
- The cluster map is correspondingly updated and distributed to clients.
- The process occurs while the cluster continues to service requests for data.

Rebalance Stages

- Each rebalance proceeds in sequential stages.
- Each stage corresponds to a Couchbase Service, deployed on the cluster.
- If all services have been deployed, there are six stages in all — one each for the Data, Query, Index, Search, Eventing, and Analytics services.
- When all stages have been completed, the rebalance process itself is complete.

Rebalance and the Data Service

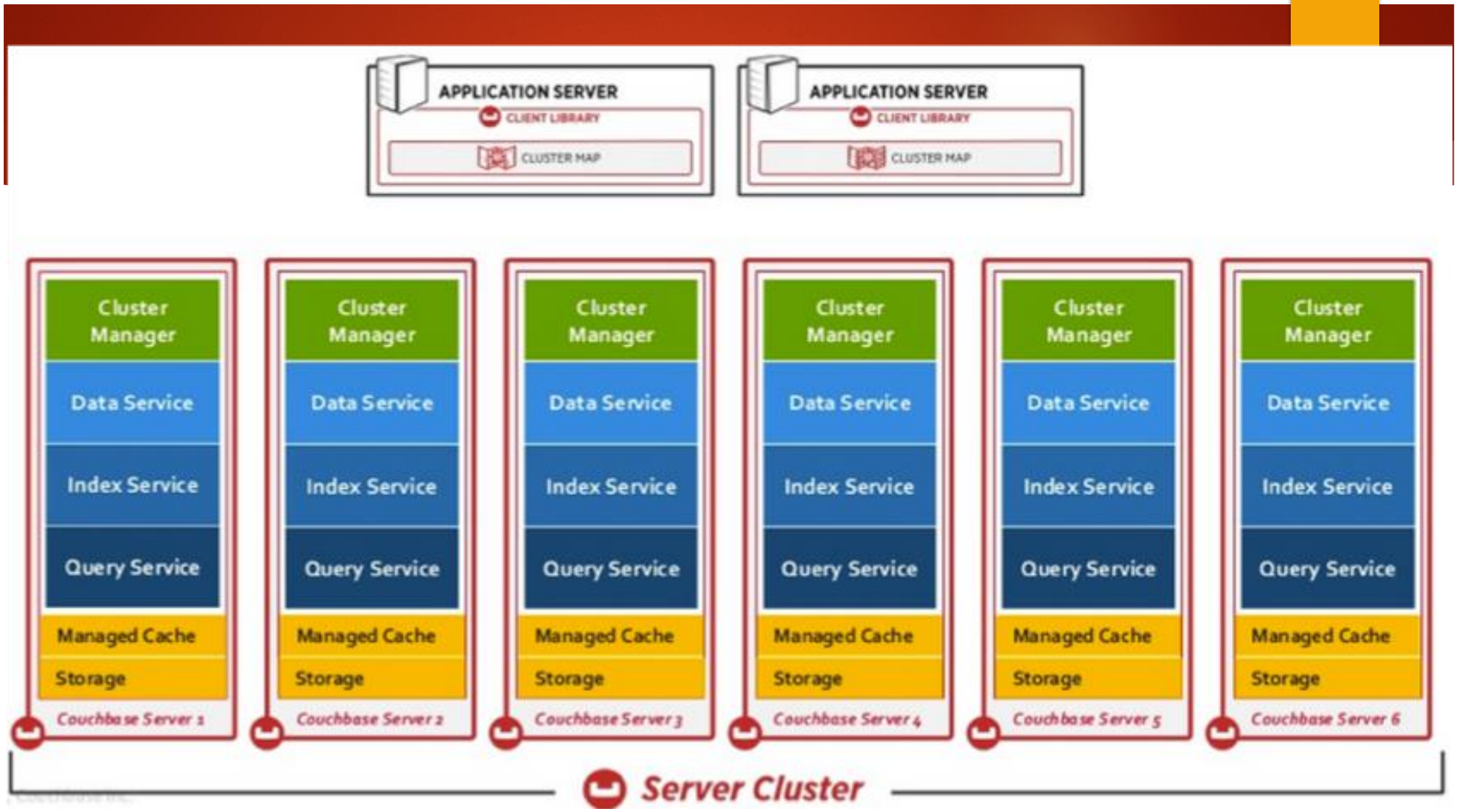
- On rebalance, vBuckets are redistributed evenly among currently available Data Service nodes.
- After rebalance, operations are directed to active vBuckets in their updated locations.
- Rebalance does not interrupt applications' data-access.
- vBucket data-transfer occurs sequentially: therefore, if rebalance stops for any reason, it can be restarted from the point at which it was stopped.
- Note the special case provided by Swap Rebalance, where the number of nodes coming into the cluster is equal to the number of nodes leaving the cluster, ensuring that data is only moved between these nodes.
- If nodes have been removed such that the desired number of replicas can no longer be supported, rebalance provides as many replicas as possible.
- For example, if four Data Service nodes previously supported one bucket with three replicas, and the Data Service node-count is reduced to three, rebalance provides two replicas only.
  - If and when the missing Data Service node is restored or replaced, rebalance will provide three replicas again.

Accessing Rebalance Reports

- Couchbase Server creates a report on every rebalance that occurs.
- Contains a JSON document, which can be inspected in any browser or editor.
- Provides summaries of the concluded rebalance activity, as well as details for each of the vBuckets affected: in consequence, the report may be of considerable length.

On conclusion of a rebalance, its report can be accessed in any of the following ways:

- By means of Couchbase Web Console
- By means of the REST API
- By accessing the directory /opt/couchbase/var/lib/couchbase/logs/reblance on any of the cluster nodes.
- A rebalance report is maintained here for (up to) the last five rebalances performed. Each report is provided as a *.json file, whose name indicates the time at which the report was run — for example, rebalance_report_2020-03-17T11:10:17Z.json.


APPLICATION SERVER
CLIENT LIBRARY
CLUSTER MAP
APPLICATION SERVER
CLIENT LIBRARY
CLUSTER MAP
Cluster Manager
Data Service
Index Service
Query Service
Managed Cache
Storage
Couchbase Server 1
Couchbase Server 2
Couchbase Server 3
Couchbase Server 4
Couchbase Server 5
Couchbase Server 6
Server Cluster

Manage with ease

- Global deployment with low write latency using active-active cross datacenter replication
- Infrastructure agnostic support across physical, virtual, cloud and containerized environments
- Microservices architecture with built-in auto-sharding, replication, and failover
- Full-stack security with end-to-end encryption and rolebased access control