Couchbase Server

ANJU MUNOTH

NoSQL Database

- **NoSQL database** technology is a database type that stores information in JSON documents instead of columns and rows used by relational databases.
- NoSQL databases are built to be flexible, scalable, and capable of rapidly responding to the data management demands of modern businesse

Who are using no-sql

- Examples of Global 2000 enterprises that are deploying NoSQL for mission-critical applications that have been featured in recent news reports:
- **Tesco**, Europe's No.1 retailer, deploys NoSQL for e-commerce, product catalog, and other applications
- **Ryanair**, the world's busiest airline, uses NoSQL to power its mobile app serving over 3 million users
- **Marriott** deploys NoSQL for its reservation system that books $38 billion annually
- **Gannett**, the No.1 U.S. newspaper publisher, uses NoSQL for its proprietary content management system, Presto
- **GE** deploys NoSQL for its Predix platform to help manage the Industrial Internet

Trends creating new technical challenges that NoSQL addresses

Digital Economy trends	Requirements
1. More customers are going online	• Scaling to support thousands, if not millions, of users • Meeting UX requirements with consistent high performance • Maintaining availability 24 hours a day, 7 days a week
2. The internet is connecting everything	• Supporting many different things with different data structures • Supporting hardware/software updates, generating different data • Supporting continuous streams of real-time data

Trends creating new technical challenges that NoSQL addresses

Digital Economy trends	Requirements
3. Big data is getting bigger	• Storing customer generated semi-structured/unstructured data • Storing different types of data from different sources, together • Storing data generated by thousands/millions of customers/things
4. Applications are moving to the cloud	• Scaling on demand to support more customers, store more data • Operating applications on a global scale – customers worldwide • Minimizing infrastructure costs, achieving a faster time to market

Trends creating new technical challenges that NoSQL addresses

Digital Economy trends	Requirements
5. The world has gone mobile	• Creating “offline first” apps – network connection not required • Synchronizing mobile data with remote databases in the cloud • Supporting multiple mobile platforms with a single backend

Why relational databases fall short

- Relational databases were born in the era of mainframes and business applications – long before the internet, the cloud, big data, mobile, and today's massively interactive enterprise.
- First commercial implementation was released by Oracle in 1979.
- RDBMs were engineered to run on a single server – the bigger, the better.
- Only way to increase the capacity of these databases was to upgrade the servers – processors, memory, and storage – to scale up.
- NoSQL databases emerged as a result of the exponential growth of the internet and the rise of web applications.
- The need to **develop with agility** and to **operate at any scale**.

Develop with agility

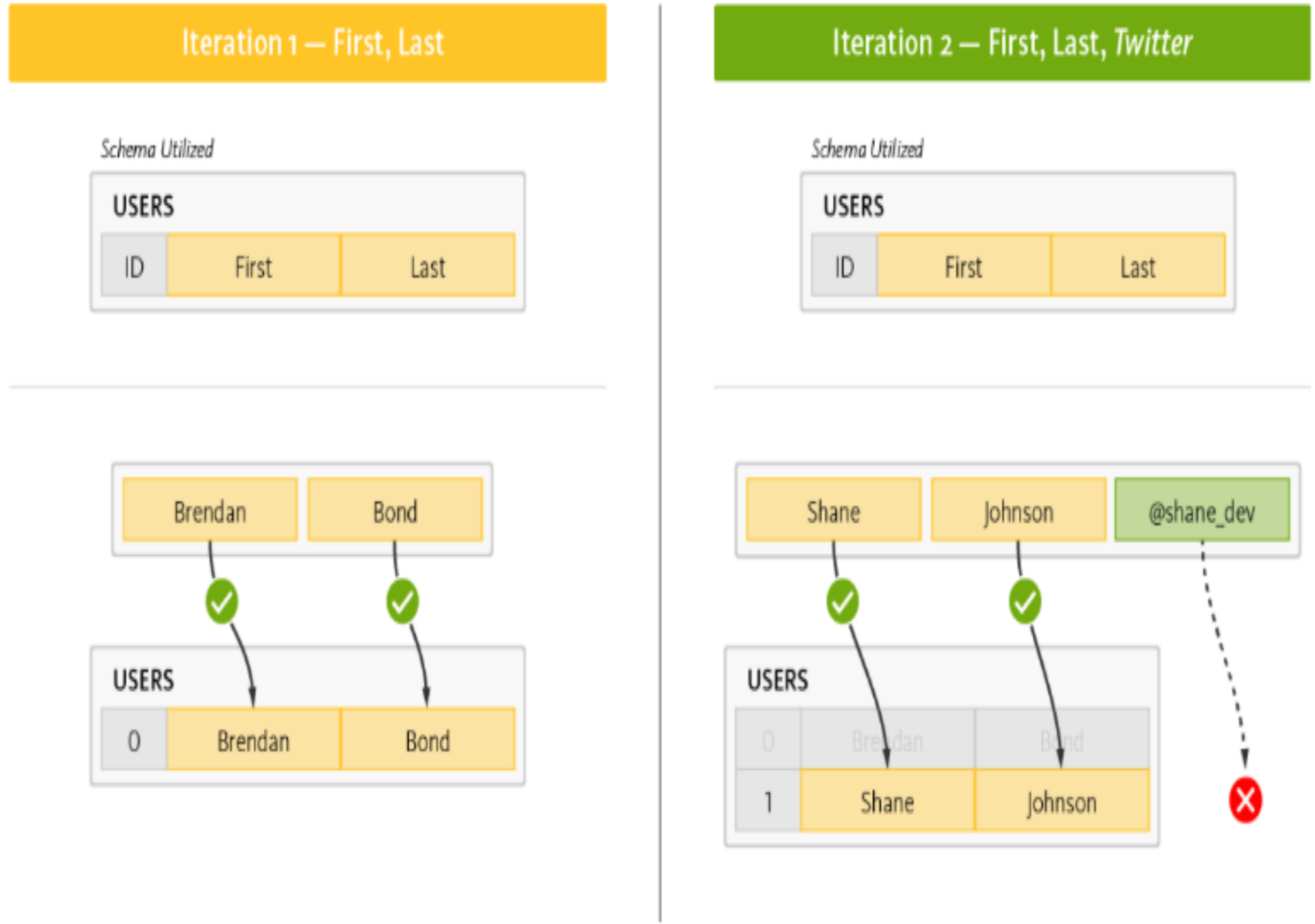
- To remain competitive in today's experience-focused digital economy, enterprises must innovate – and they have to do it faster than ever before.
- And because this innovation centers on the development of modern web, mobile, and IoT applications, developers have to deliver applications and services faster than ever before.
- Speed and agility are both critical because these applications evolve far more rapidly than legacy applications like ERP.
- Relational databases are a major roadblock because they don't support agile development very well due to their fixed data model.

Scoping for changing requirements

- A core principle of agile development is adapting to evolving application requirements: when the requirements change, the data model also changes.
- Is a problem for relational databases because the data model is fixed and defined by a static schema.
- So in order to change the data model, developers have to modify the schema, or worse, request a “schema change” from the database administrators.
- This slows down or stops development, not only because it is a manual, time-consuming process, but it also impacts other applications and services.

Figure 1

RDBMS – An explicit schema prevents the addition of new attributes on demand.



Develop with agility

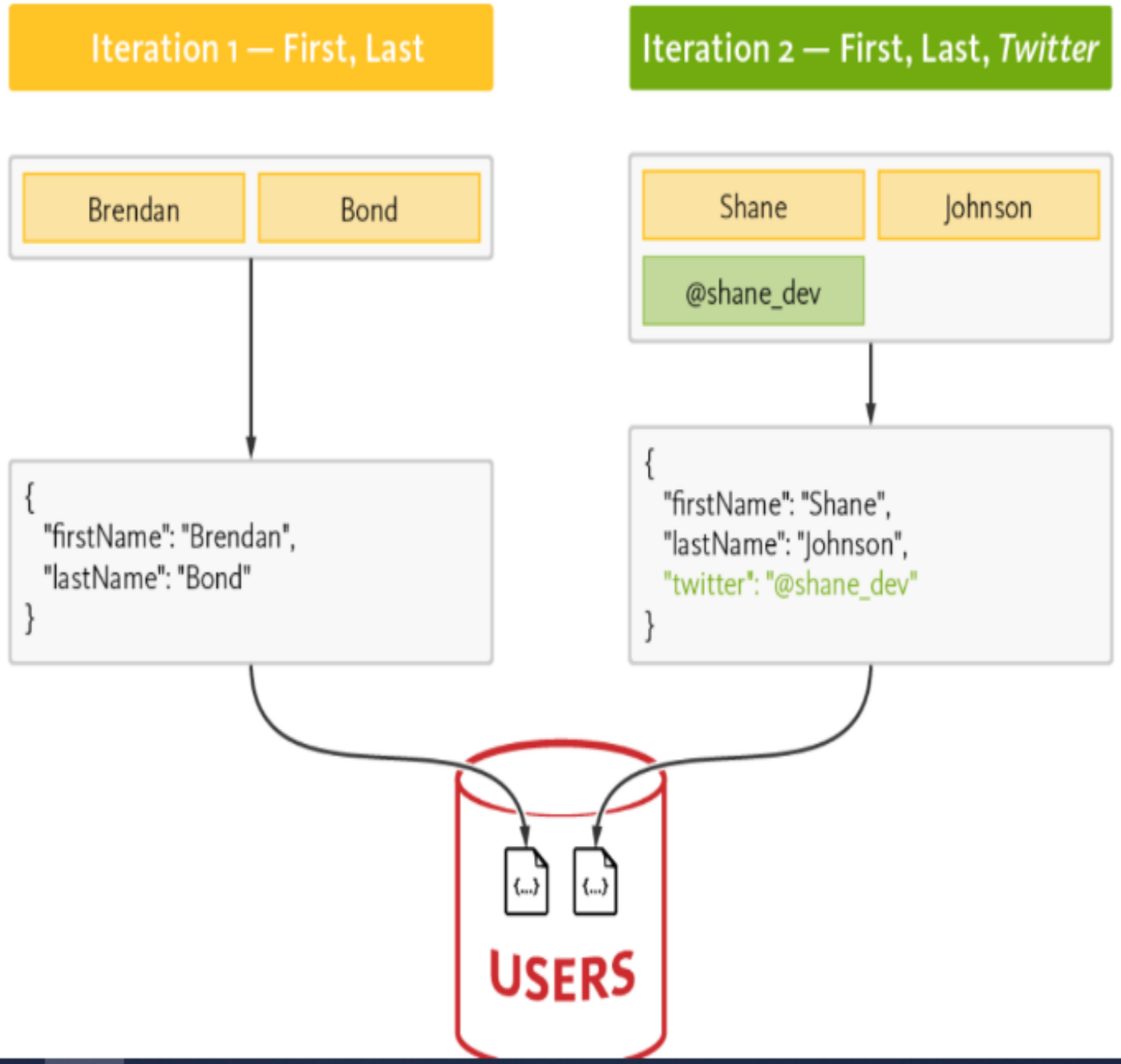
- To remain competitive in today's experience-focused digital economy, enterprises must innovate – and they have to do it faster than ever before.
- And because this innovation centers on the development of modern web, mobile, and IoT applications, developers have to deliver applications and services faster than ever before.
- Speed and agility are both critical because these applications evolve far more rapidly than legacy applications like ERP.
- Relational databases are a major roadblock because they don't support agile development very well due to their fixed data model.

Flexibility for Faster Development

- By comparison, a NoSQL document database fully supports agile development, because it is schema-less and does not statically define how the data must be modeled.
- Instead, it defers to the applications and services, and thus to the developers as to how data should be modeled.
- With NoSQL, the data model is defined by the application model. Applications and services model data as objects.

Figure 2

JSON – The data model evolves as new attributes are added on demand.



Simplicity for easier development

- Applications and services model data as objects (e.g., employee), multi-valued data as collections (e.g., roles), and related data as nested objects or collections (e.g., manager).
- However, relational databases model data as tables of rows and columns – related data as rows within different tables, and multi-valued data as rows within the same table.
- The problem with relational databases is that data is read and written by disassembling, or “shredding,” and reassembling objects. This is the object-relational “impedance mismatch.”
- The workaround is object-relational mapping frameworks, which are inefficient at best, and problematic at worst.
- As an example, consider an application for managing resumes. It interacts with resumes as an object, the user object. It contains an array for skills and a collection for positions.
- However, writing a resume to a relational database requires the application to “shred” the user object.

Figure 3

RDBMS – Applications "shred" objects into rows of data stored in multiple tables.

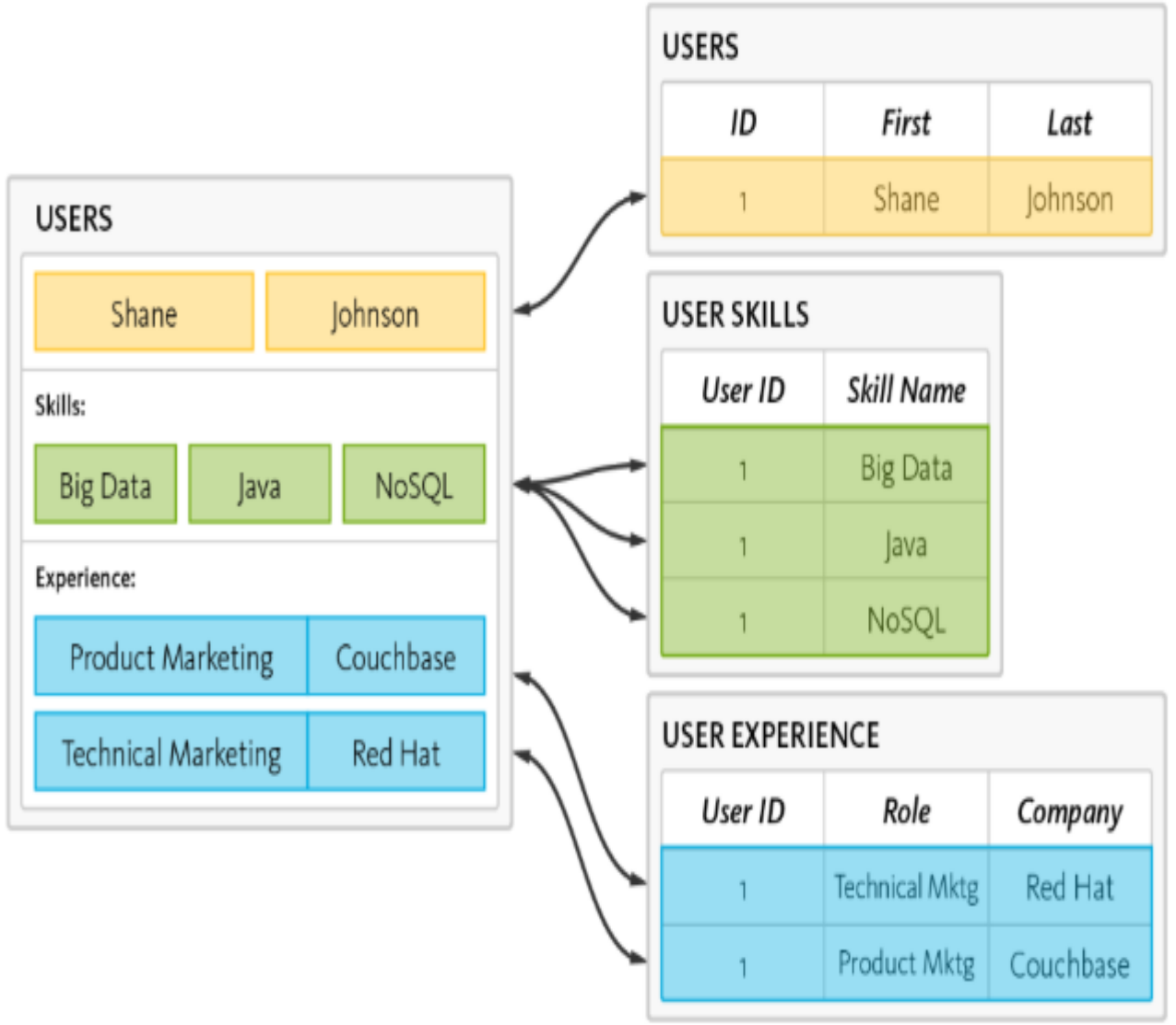


Figure 4

RDBMS – Queries return duplicate data, applications have to filter it out.

Shane	Johnson	Big Data	Product Marketing	Couchbase
Shane	Johnson	Big Data	Technical Marketing	Red Hat
Shane	Johnson	Java	Product Marketing	Couchbase
Shane	Johnson	Java	Technical Marketing	Red Hat
Shane	Johnson	NoSQL	Product Marketing	Couchbase
Shane	Johnson	NoSQL	Technical Marketing	Red Hat

Figure 5

JSON - Applications can store objects with nested data as single documents.

NoSql Database

- Document-oriented NoSQL database reads and writes data formatted in JSON – which is the de facto standard for consuming and producing data for web, mobile, and IoT applications.

- Not only eliminates the object-relational impedance mismatch, it also eliminates the overhead of ORM frameworks and simplifies application development because objects are read and written without “shredding” them (i.e., a single object can be read or written as a single document),

Querying and SQL

- Couchbase Server 4.0 introduced N1QL
- A powerful query language that extends SQL to JSON, enabling developers to leverage both the power of SQL and the flexibility of JSON.
- Supports standard SELECT / FROM / WHERE statements
- Also supports aggregation (GROUP BY), sorting (SORT BY), joins (LEFT OUTER / INNER), as well as querying nested arrays and collections.
- Query performance can be improved with composite, partial, covering indexes, and more.

SQL

```
SELECT breweries.name AS brewery,
  count(*) AS cnt
FROM beers
INNER JOIN breweries
ON beer.brewery_id = breweries.id
WHERE beers.type = "beer" AND
  breweries.type = "brewery" AND
  beers.style = "American-Style Imperial Stout"
GROUP BY breweries.name
HAVING count(*) > 2
ORDER BY cnt DESC;
```

N1QL

```
SELECT breweries.name AS brewery,
  count(*) AS cnt
FROM `beer-sample` beers
INNER JOIN `beer-sample` breweries
ON KEYS beers.brewery_id
WHERE beers.type = "beer" AND
  breweries.type = "brewery" AND
  beers.style = "American-Style Imperial Stout"
GROUP BY breweries.name
HAVING count(*) > 2
ORDER BY cnt DESC;
```

Operate at any scale

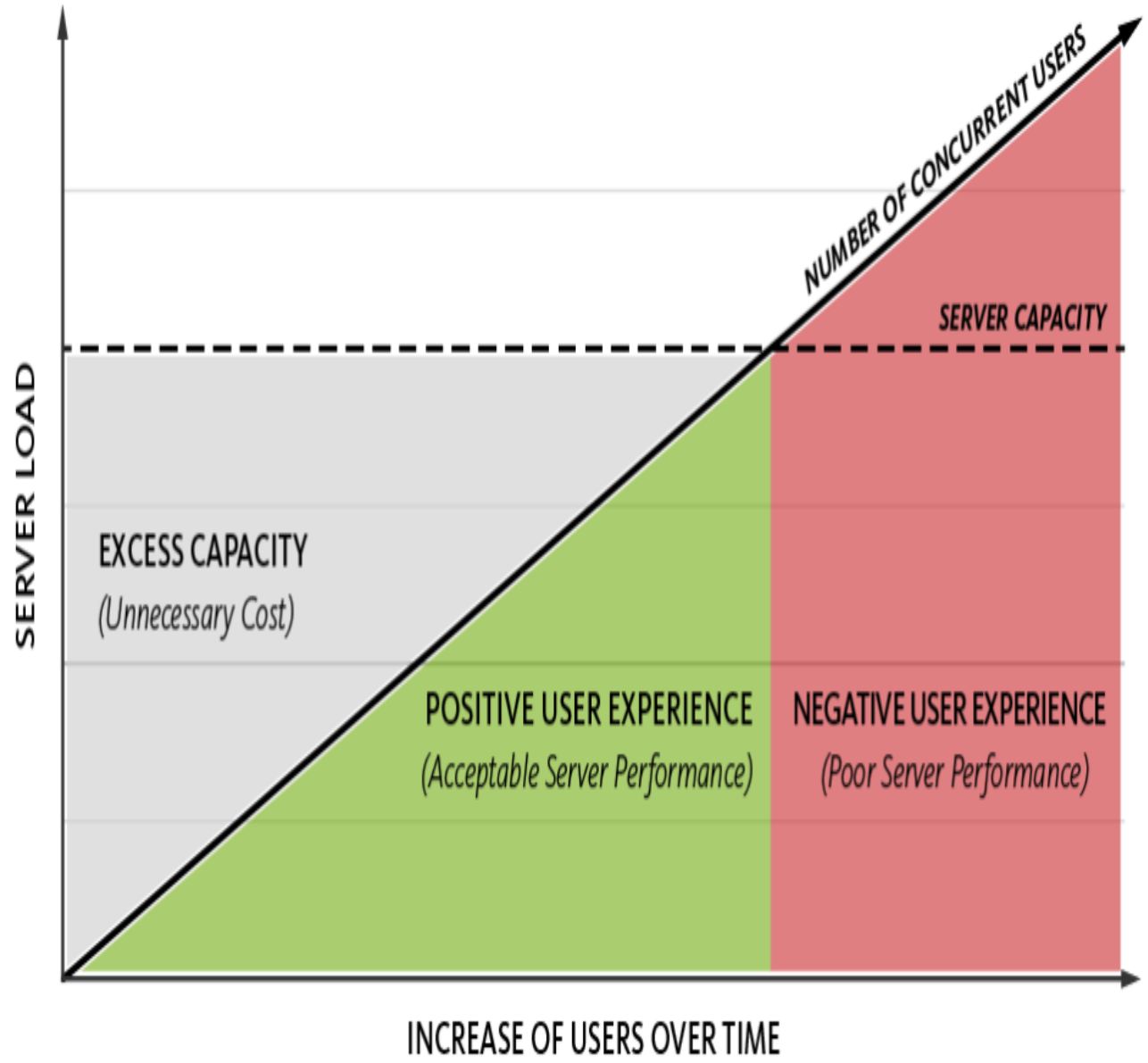
- Databases that support web, mobile, and IoT applications must be able to operate at any scale.
- While it's possible to scale a relational database like Oracle (using, for example, Oracle RAC), doing so is typically complex, expensive, and not fully reliable.
- With Oracle, for example, scaling out using RAC technology requires numerous components and creates a single point of failure that jeopardizes availability.
- By comparison, a NoSQL distributed database – designed with a scale-out architecture and no single point of failure – provides compelling operational advantages.

Elasticity for performance at scale

- Applications and services have to support an ever increasing number of users and data – hundreds to thousands to millions of users, and gigabytes to terabytes of operational data.
- At the same time, they have to scale to maintain performance, and they have to do it efficiently.
- The database has to be able to scale reads, writes, and storage.
- Is a problem for relational databases that are limited to scaling up (i.e., adding more processors, memory, and storage to a single physical server).
- Ability to scale efficiently, and on demand, is a challenge.
- Also becomes increasingly expensive because enterprises have to purchase bigger and bigger servers to accommodate more users and more data.
- Can result in downtime if the database has to be taken offline to perform hardware upgrades.

Figure 6

RDBMS – The server is too big or too small, leading to unnecessary costs or poor performance.

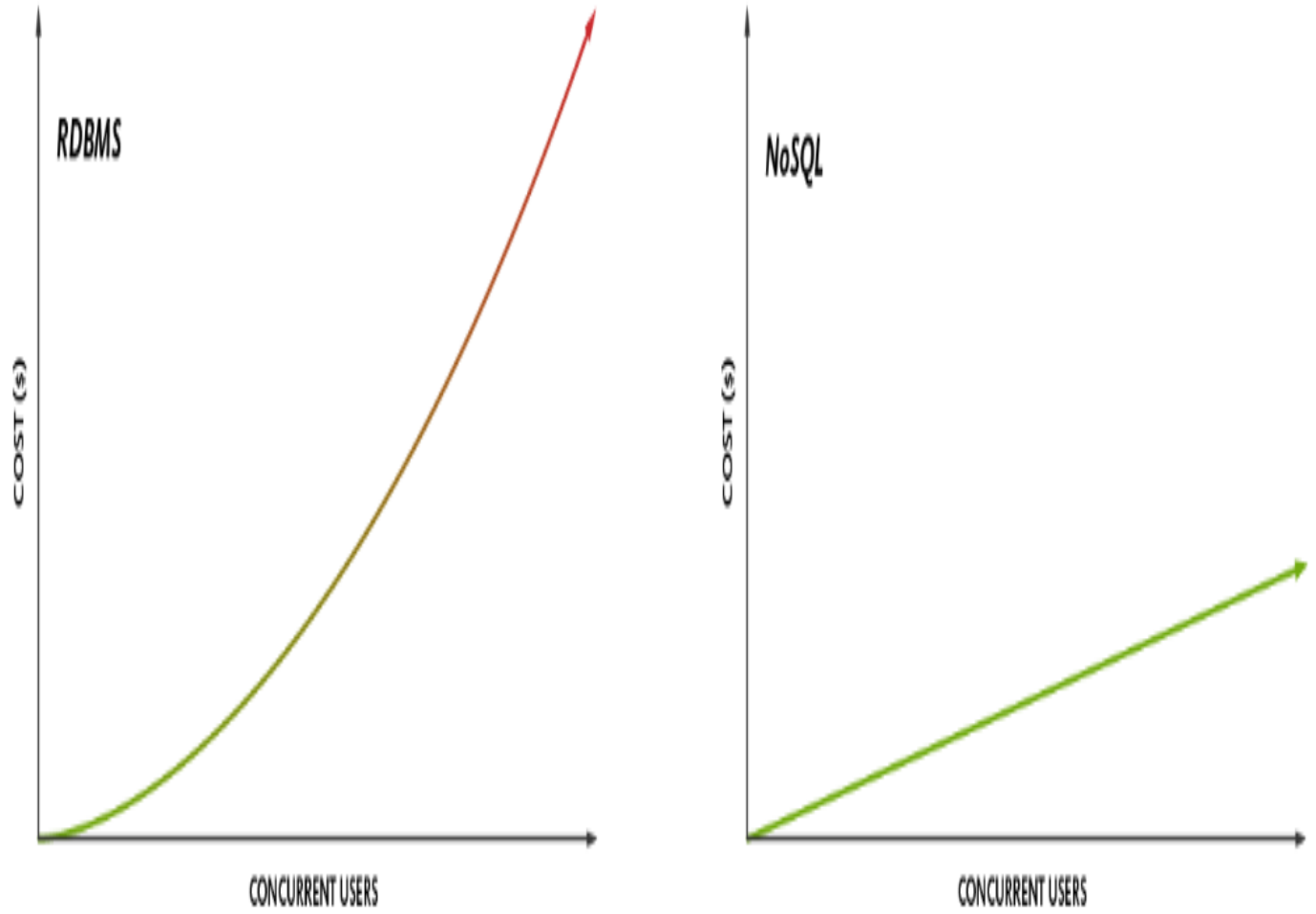


NoSQL and Elasticity

- A distributed NoSQL database, leverages commodity hardware to scale out – i.e., add more resources simply by adding more servers.
- Ability to scale out enables enterprises to scale more efficiently by
  - (a) deploying no more hardware than is required to meet the current load;
  - (b) leveraging less expensive hardware and/or cloud infrastructure;
  - (c) scaling on demand and without downtime.
- In addition to being able to scale effective and efficiently, distributed NoSQL databases are easy to install, configure, and scale.
- Engineered to distribute reads, writes, and storage, and they were engineered to operate at any scale – including the management and monitoring of clusters small and large.

Figure 7

NoSQL - Add commodity servers on demand so the hardware resources match the application load.

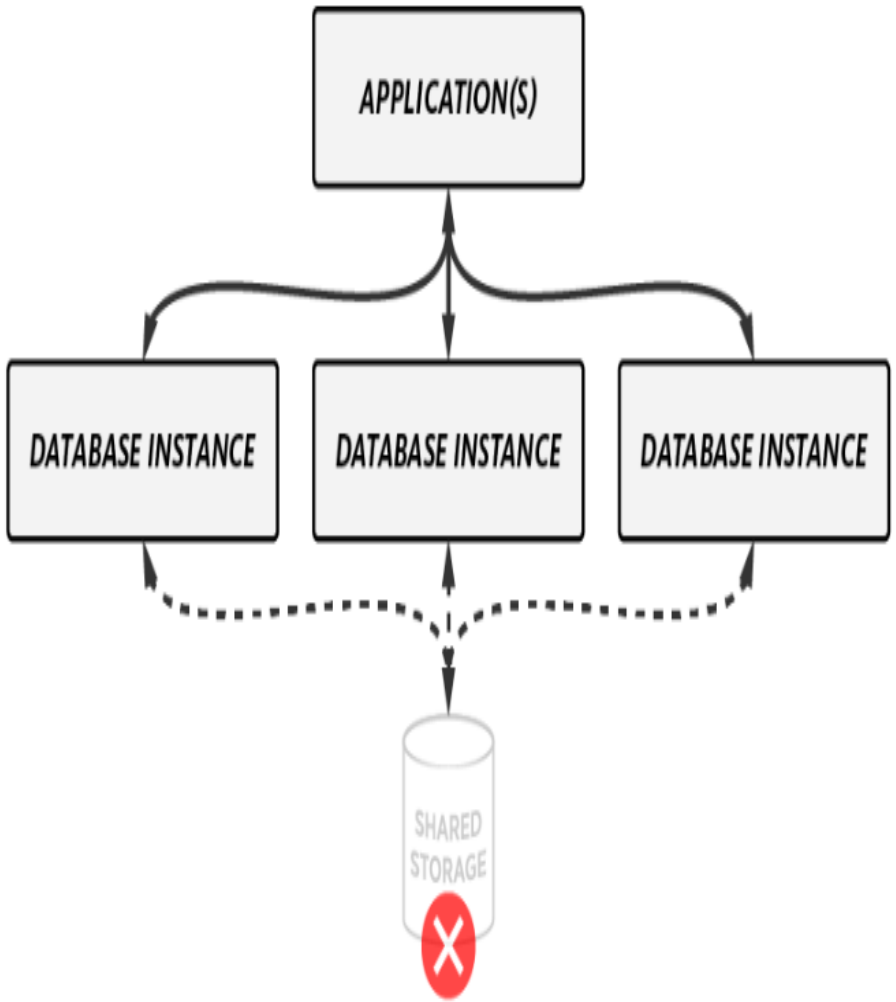


Availability for Always-on, Global Deployment

- As more and more customer engagements take place online via web and mobile apps, availability becomes a major, if not primary, concern.
- Mission-critical applications have to be available 24 hours a day, 7 days a week – no exceptions.
- Delivering 24x7 availability is a challenge for relational databases that are deployed to a single physical server or that rely on clustering with shared storage.
- If deployed as a single server and it fails, or as a cluster and the shared storage fails, the database becomes unavailable.

Figure 8

RDBMS - The failure of a server or storage device brings down the entire database.

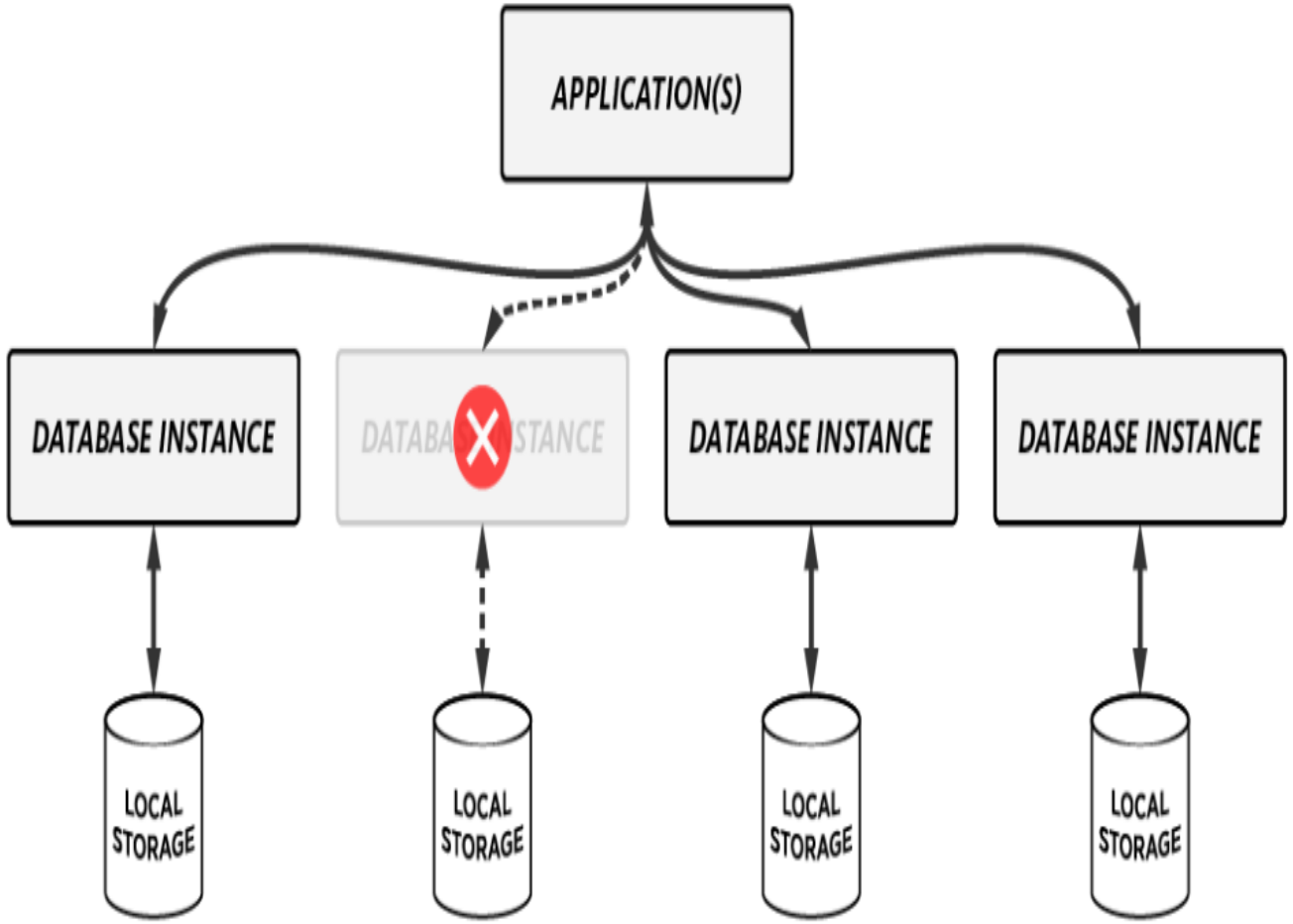


Availability for Always-on, Global Deployment

- distributed, NoSQL database partitions and distributes data to multiple database instances with no shared resources.
- Data can be replicated to one or more instances for high availability (intercluster replication).
- While relational databases like Oracle require separate software for replication (e.g., Oracle Active Data Guard), NoSQL databases do not – it's built in and it's automatic.
- Automatic failover ensures that if a node fails, the database can continue to perform reads and writes by sending the requests to a different node.

Figure 9

NoSQL – If an instance fails, the application can send requests to a different one.

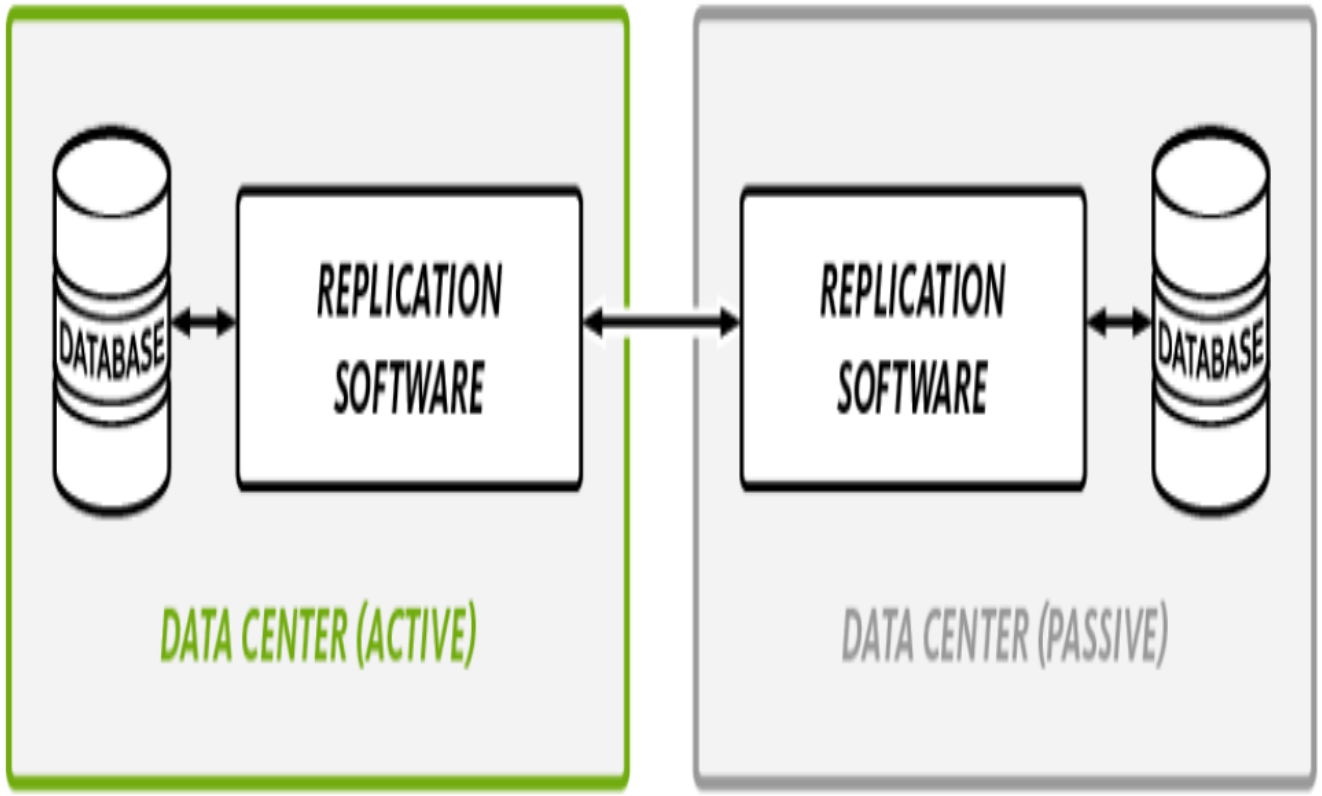


Replication

- As customer engagements move online, the need to be available in multiple countries and/or regions becomes critical.
- While deploying a database to multiple datacenters increases availability and helps with disaster recovery, it also has the benefit of increasing performance, because all reads and writes can be executed on the nearest datacenter, thereby reducing latency.
- Ensuring global availability is difficult for relational databases in cases where the requirement of separate add-ons increases complexity
- Oracle requires Oracle GoldenGate to move data between databases – or when replication between multiple datacenters can only be used for failover because only one datacenter is active at a time.
- Also, when replicating between datacenters, applications built on relational databases can experience performance degradation or find that the datacenters are severely out of sync.

Figure 10

RDBMS - Requires separate software to replicate data to other datacenters.

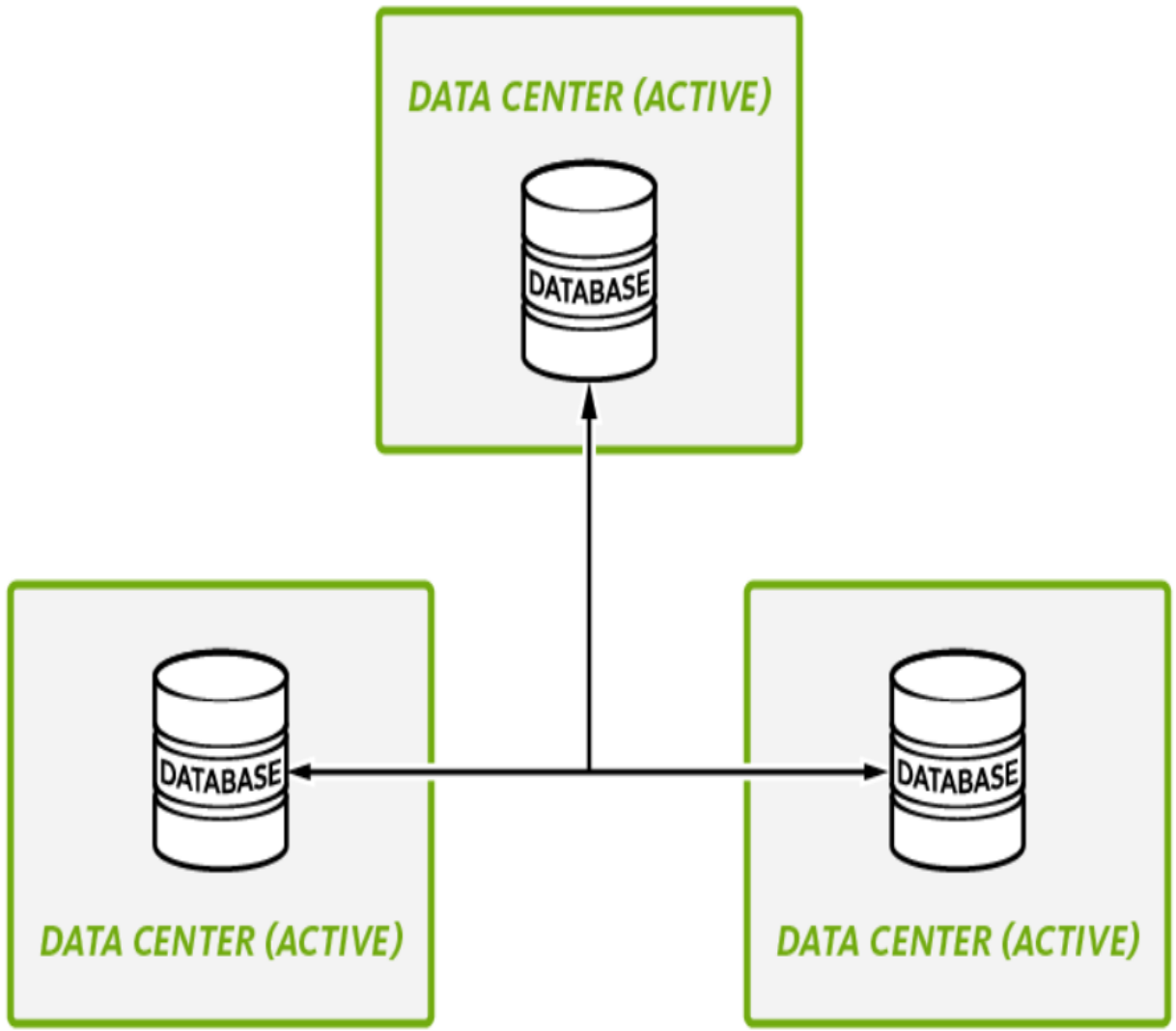


Replication in No-SQL

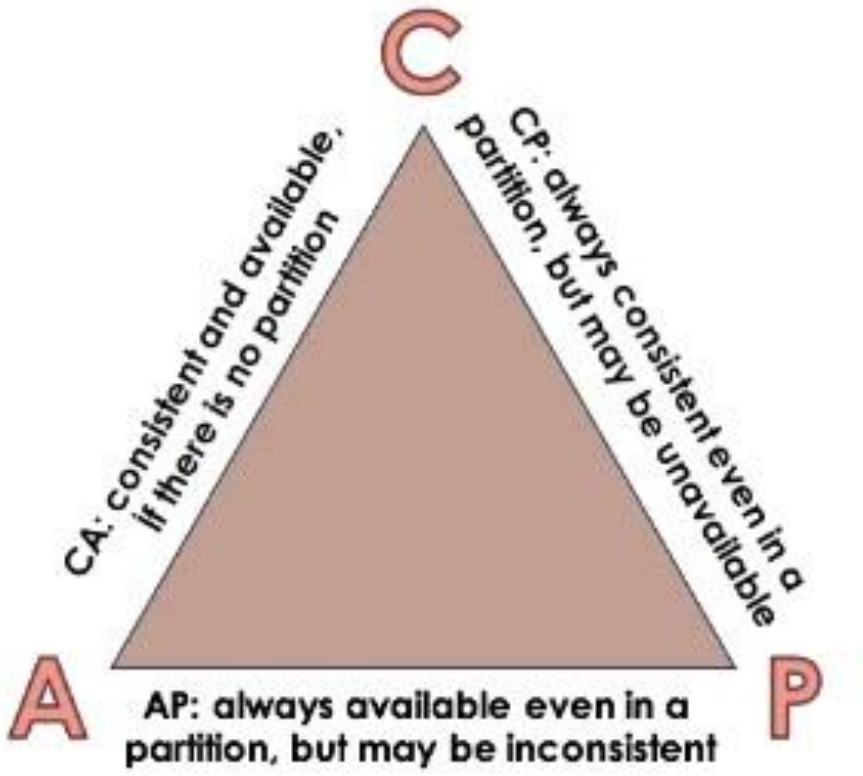
- A distributed, NoSQL database includes built-in replication between datacenters – no separate software is required.
- Some include both unidirectional and bidirectional replication enabling full active-active deployments to multiple datacenters, which allow the database to be deployed in multiple countries and/or regions and to provide local data access to local applications and their users.
- Improves performance
- Also enables immediate failover via hardware routers – applications don't have to wait for the database to discover the failure and perform its own failover

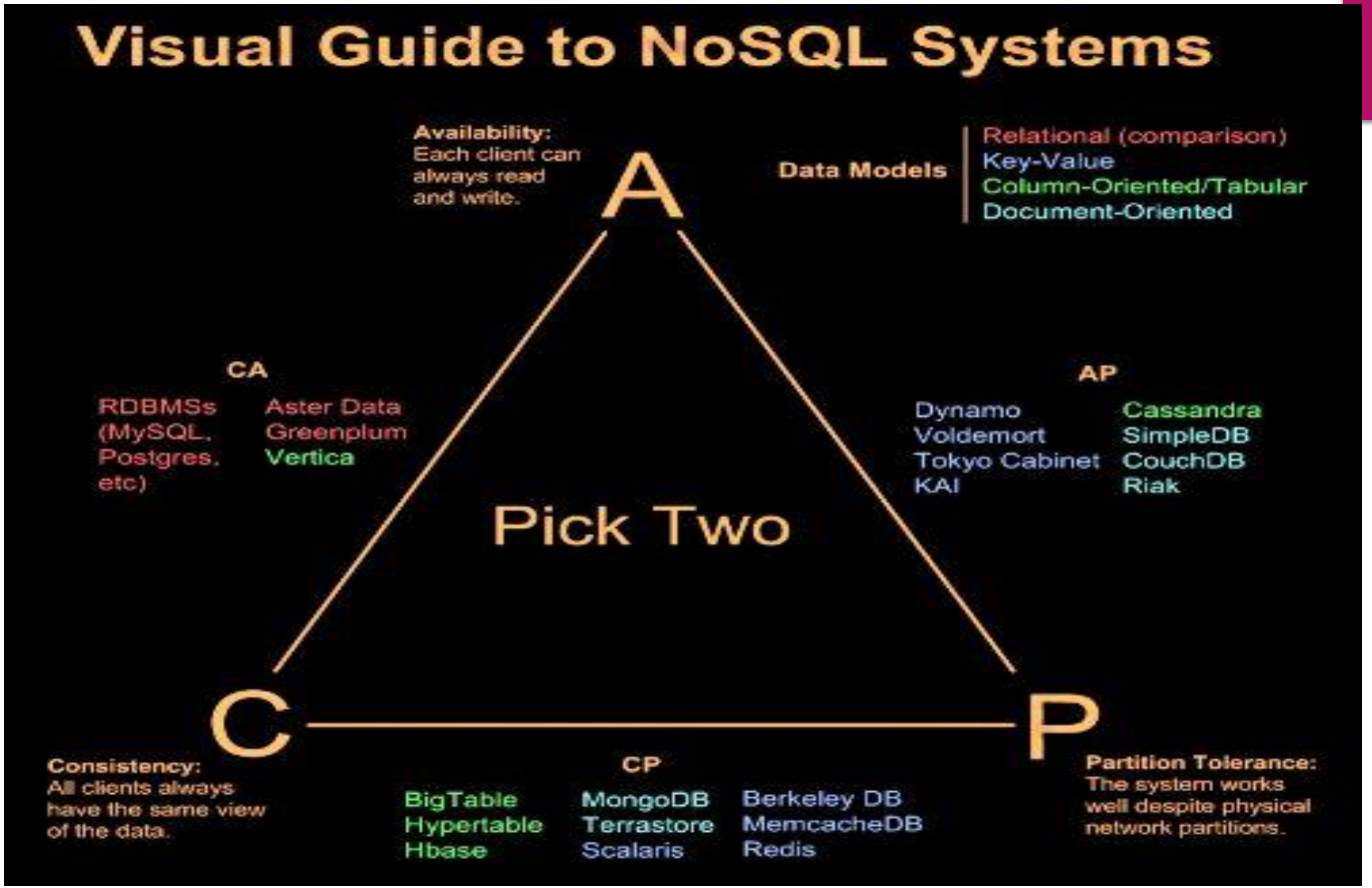
Figure 11

NoSQL – Replication between datacenters is fully built in, and can be bidirectional.

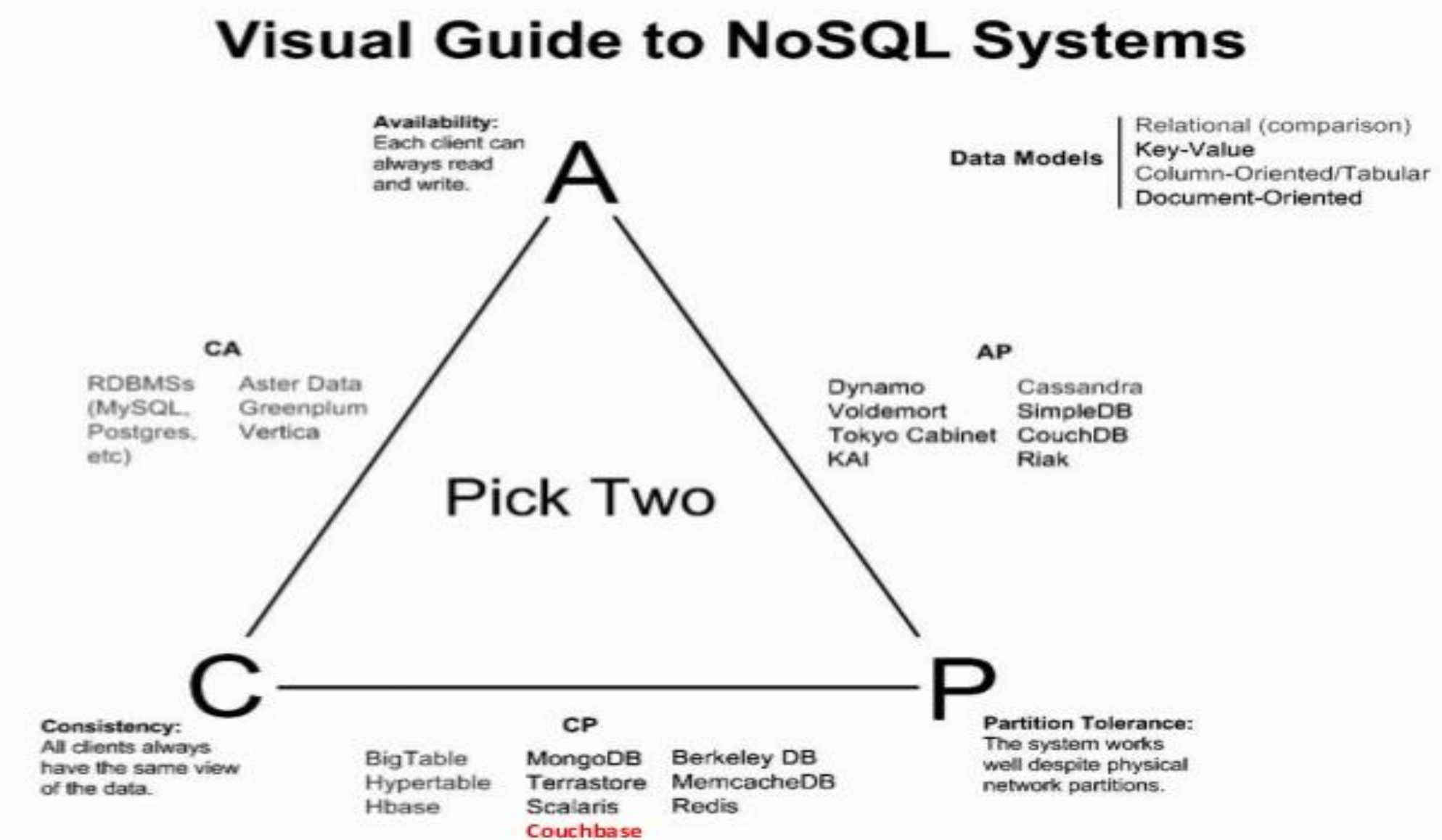


CAP THEOREM


C
CA: consistent and available, if there is no partition
CP: always consistent even in a partition, but may be unavailable
A
P
AP: always available even in a partition, but may be inconsistent


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

Visual Guide to NoSQL Systems



Systems by CAP Classification

AP	CP	AC
Cassandra	Hadoop and EcoSystem	RDBMS
Riak	Spark	Vertica
Dynamo	Mongo	
CouchDB	Couchbase	

CAP Theorem and Couchbase Server

- Single cluster deployment: Couchbase Server is mainly referred to as a CP system with options to replax C in favor of A (for example auto-failover).
- Multi cluster deployment with XDCR: Couchbase Server provides you AP.
  - Can write to one of the clusters and we'll detect and resolve the conflict giving you eventual consistency between multiple clusters (strongly recommend you should understand how we detect and resolve these conflicts to be sure the effect is what you expect).
- **Couchbase Server can be a more CP or a more AP system depending on the deployment topology.**

Couchbase Server

- Open source, distributed, JSON document database.
- Exposes a scale-out, key-value store with managed cache for sub-millisecond data operations, purpose-built indexers for efficient queries and a powerful query engine for executing SQL-like queries.
- For mobile and Internet of Things environments Couchbase also runs natively on-device and manages synchronization to the server.
- Strongly consistent within a local cluster. Couchbase Server also supports cross data center replication with eventual consistency across clusters.

Couchbase

- Couchbase Lite is an embedded mobile database that works offline and synchronizes with Couchbase Sync Gateway when online. Sync Gateway synchronizes with Couchbase Server as well as with multiple Couchbase Lite instances.
- Couchbase Server can be deployed on premises, in the cloud, on Kubernetes, or in hybrid configurations.
- Comes in both open source and enterprise versions.
- Couchbase Server query language, N1QL, is a SQL superset designed for JSON document databases, with extensions for analytics.
- Couchbase also supports key-value data access and full-text search.

History of Couchbase

- Couchbase, the company behind the database, grew from the merger of Membase (maker of an in-memory cached clustered key-value database) and CouchOne (developers of the Apache CouchDB document database) in 2011.
- The new company started with the key-value layer
- Added the JSON document layer in 2012
- Added a mobile database in 2014
- Added SQL-like queries in 2015
- Added Full-text search in 2017
- Added analytics in 2018.

Features

- Sub-millisecond data operations are provided by powerful services for querying and indexing, and by a feature-rich, document-oriented query-language, *N1QL*.
- Multiple instances of Couchbase Server can be combined into a single *cluster*.
- A *Cluster Manager* program coordinates all node-activities, and provides a simple, cluster-wide interface to all clients.
- Cluster administration is supported by a graphical, web-based administration console; as well as by REST and command-line interfaces.
- Individual nodes can be added, removed, and replaced as appropriate, with no down-time required for the cluster as a whole.

DATA

Data can be retained either in memory only, or in both memory and storage, as judged appropriate by the administrator.

Data can be replicated across the nodes of the cluster, to ensure that node-loss (or even rack-loss) does not entail data-loss.

Data items can also be selectively replicated across data centers; for the purpose either of backup only, or of simultaneous, multi-geo application-access.

DATA

- Writes are committed to memory, then persisted to disk and indexed asynchronously without blocking reads or writes.
- Most-used data and indexes are transparently maintained in memory for fast reads.
- Heavy use of memory is good for latency and throughput, although it increases Couchbase’s RAM requirements.

Multi Dimension Scaling

- Couchbase Server can scale each of its services independently, to make them more efficient.
- Query service can benefit from more CPU resources, the index service can use SSDs, and the data service can use more RAM.
- Couchbase calls this multi-dimensional scaling (MDS), and it is one of Couchbase Server's distinguishing features
- Asynchronous operations help Couchbase Server to avoid blocking writes, reads, or queries.
- Developer can balance durability and consistency against latency when needed

Couchbase and JSON

- JSON data model supports both basic and complex data types: numbers, strings, nested objects, and arrays.
- Can create documents that are normalized or denormalized.
- Does not require or even support schemas.
- By contrast, MongoDB doesn't require schemas, but can support and enforce them if the developer chooses.
- Can access Couchbase Server documents through four mechanisms:
  - **key-value, SQL-based queries, full-text search, and JavaScript eventing.**
- If your JSON documents have subdocuments or arrays, can access them directly using path expressions without needing to transfer and parse the whole document.
- Eventing model can trigger on data changes (OnUpdate) or timers.
- Can access Couchbase Server documents through synchronization with Couchbase Mobile.

Couchbase and buckets

- Couchbase Server is organized into buckets, vBuckets, nodes, and clusters.
- Buckets hold JSON documents.
- vBuckets are essentially shards that are automatically distributed across nodes.
- Nodes are physical or virtual machines that host single instances of Couchbase Server.

Clustering in Couchbase

- Clusters are groups of nodes.
- Synchronous replication occurs between the nodes in a cluster.
- Couchbase Server does synchronous replication and has strong consistency within a cluster.
- Does asynchronous, active-active replication across clusters, data centers, and availability zones, to avoid incurring high write latencies.
- XDCR(Cross Data Centre Replication) allows Couchbase to be a globally distributed database, at the cost of allowing eventual (rather than strong) consistency between clusters.

Couchbase query tools

- Can query Couchbase Server using a key to retrieve the associated value, which can be a JSON document or a Blob.
- Can also query it with the SQL-like N1QL language or with a full-text search.
- Both N1QL and full-text queries go faster if the bucket has indexes to support the query.

N1QL

- N1QL, pronounced “nickel,” looks very much like standard SQL, with extensions for JSON.
- Some of the N1QL extensions are USE KEYS, NEST, UNNEST, and MISSING.
- USE KEYS and USE HASH are query hints for JOINs.
- NEST and UNNEST pack and unpack arrays.
- MISSING is a JSON-specific alternative to NULL; IS NOT MISSING means that a specific value is present or NULL in a document.
- The keyword for values that are NOT MISSING and NOT NULL is KNOWN. N1QL queries can use paths, which also apply to full-text searches.

Full-text search

- Couchbase supports external full-text search engines, such as Solr, but it also has its own Go-based, full-text search engine, Bleve.
- Bleve is included in Couchbase Mobile as well as Couchbase Server, and it supports most of the search syntaxes.

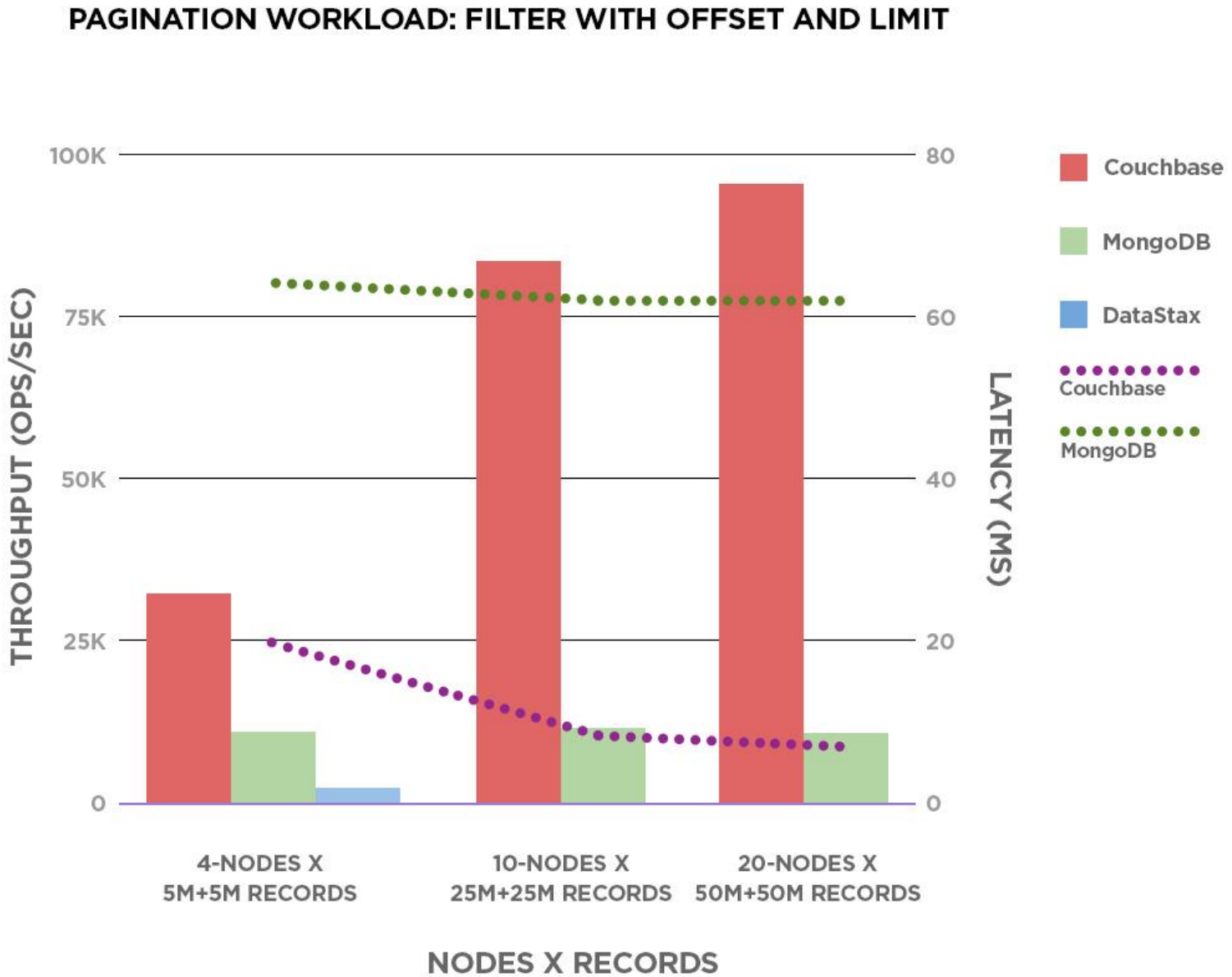
Couchbase SDKs

- All of the main Couchbase services are exposed for programming through the SDK. SDKs are available for C/C++, .Net (C#, F#, and Visual Basic .Net), Go, Java, Node.js, PHP, Python, and Scala.
- In addition to the SDKs, Couchbase offers tight integration with several frameworks: Spring Data, .NET LINQ, and Couchbase's own Ottoman Node.js ODM.

Couchbase Mobile

- Couchbase Mobile has two parts: Couchbase Lite, which runs on a mobile device, and Couchbase Sync Gateway, which runs on a server node.
- Couchbase Lite runs on iOS, Android, .Net, and Xamarin, and supports the Swift, Objective-C, Java, Kotlin, and C++ languages.

Couchbase benchmarks



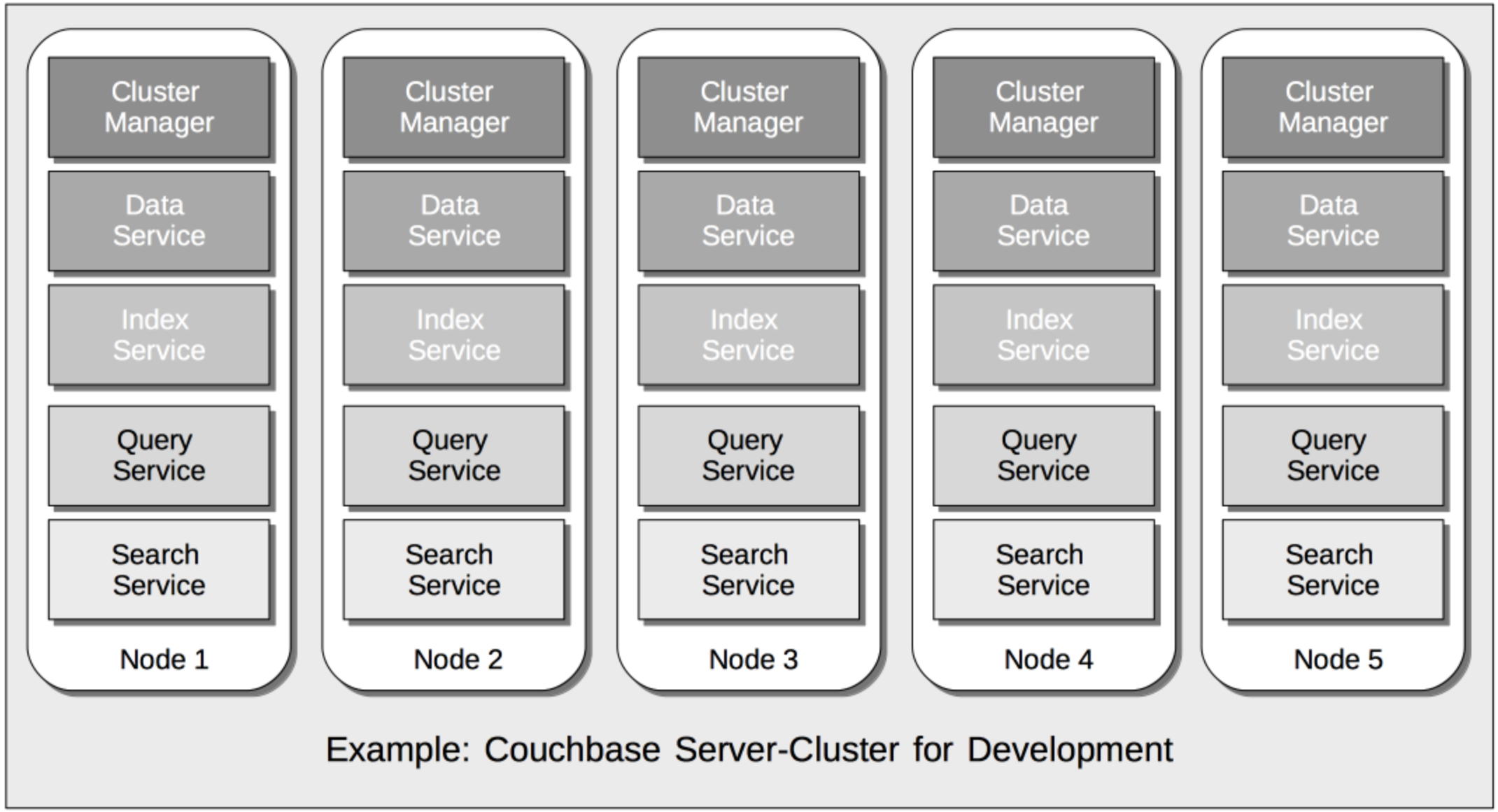
Couchbase provided higher throughput and lower latency, which improved as the data and server were scaled.

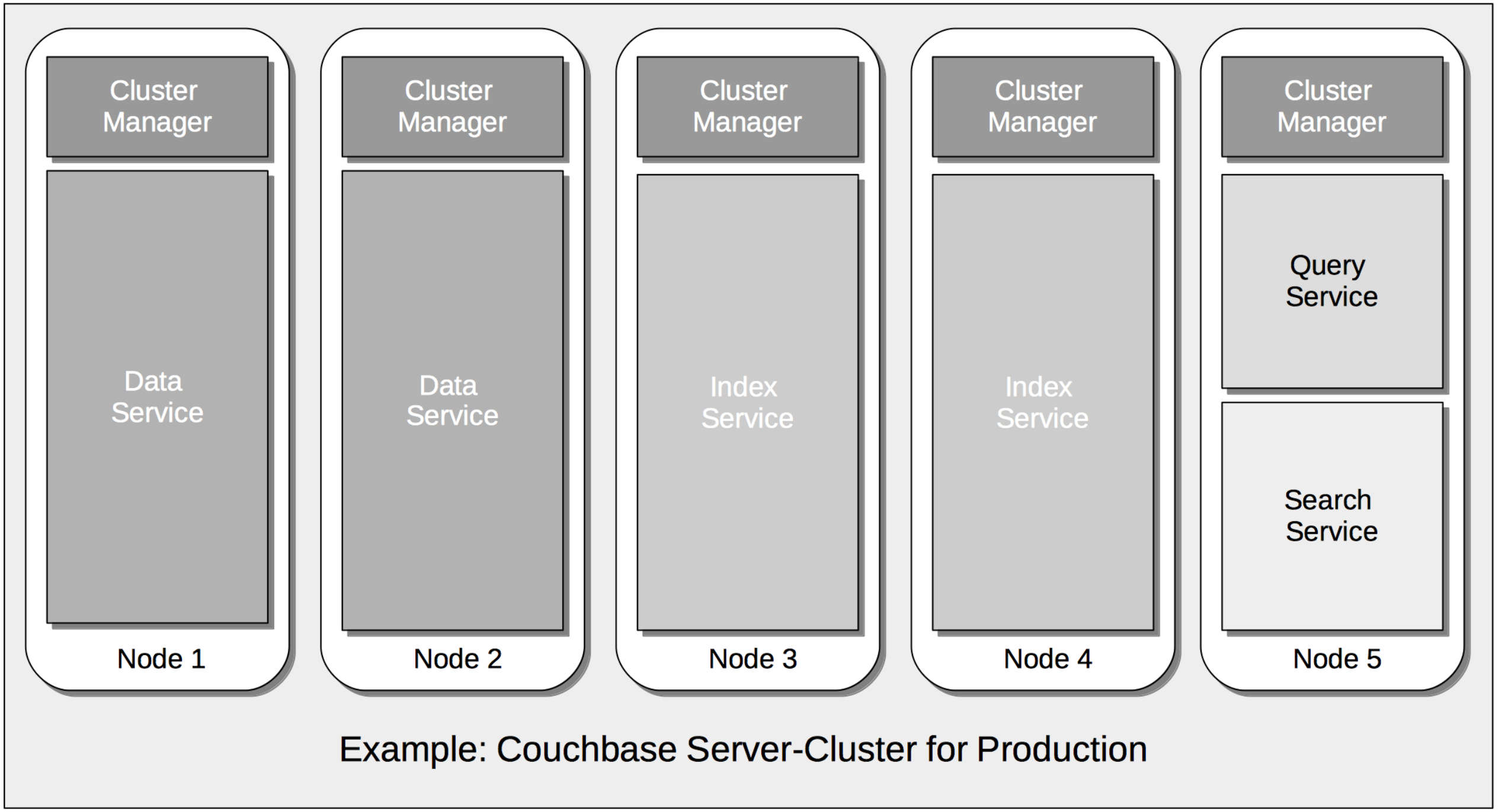
Cost

- Couchbase Server Community Edition: Free.
- Couchbase Server Enterprise Edition: Annual subscriptions are priced by node and available at different price points depending on a node's needed cores and RAM.
- Development and test nodes are free.
- Enterprise Edition cloud deployments are available by the hour, with typical software pricing of $0.662/node/hour on AWS for Couchbase Server and $1.641/node/hour for the Mobile Sync Gateway, with a standard template using four server nodes and two sync nodes initially, with autoscaling.
- Pricing is roughly comparable on Microsoft Azure and Google Cloud Platform.
- Can also bring your own license and pay only for the cloud resources.

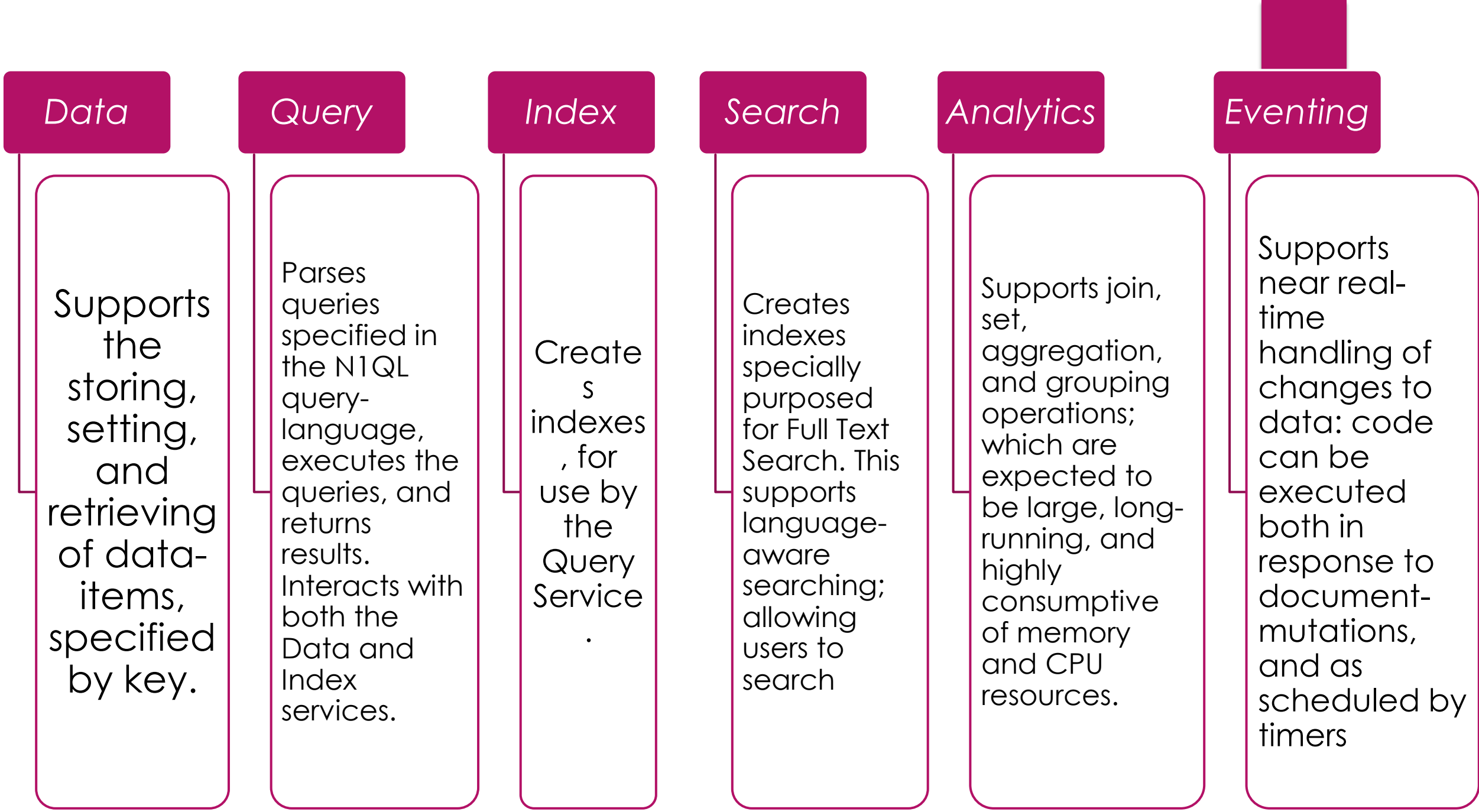
Services

- Couchbase Server provides multiple *Services*.
- Can be deployed, maintained, and provisioned independently of one another, so as to allow *Multi-Dimensional Scaling*.
- For example, a development environment running the Couchbase Data, Index, Query, and Search Services might permit an instance of each on every node of a five-node cluster


Cluster Manager
Data Service
Index Service
Query Service
Search Service
Node 1
Cluster Manager
Data Service
Index Service
Query Service
Search Service
Node 2
Cluster Manager
Data Service
Index Service
Query Service
Search Service
Node 3
Cluster Manager
Data Service
Index Service
Query Service
Search Service
Node 4
Cluster Manager
Data Service
Index Service
Query Service
Search Service
Node 5
Example: Couchbase Server-Cluster for Development



Example: Couchbase Server-Cluster for Production


Data
Supports the storing, setting, and retrieving of data-items, specified by key.
Query
Parses queries specified in the N1QL query-language, executes the queries, and returns results. Interacts with both the Data and Index services.
Index
Creates indexes, for use by the Query Service.
Search
Creates indexes specially purposed for Full Text Search. This supports language-aware searching; allowing users to search
Analytics
Supports join, set, aggregation, and grouping operations; which are expected to be large, long-running, and highly consumptive of memory and CPU resources.
Eventing
Supports near real-time handling of changes to data: code can be executed both in response to document-mutations, and as scheduled by timers

Pros

- Scales both vertically and horizontally
- Can scale different services independently for maximum performance
- Mobile database can synch with server when connected and run independently when not connected
- Document model (JSON) is flexible and doesn't need a schema
- N1QL is SQL-like and easy to learn
- Strongly consistent within a cluster

Cons

- There is no way to apply a schema
- Some of the best features are available only in Enterprise Edition
- Eventually consistent across clusters

Keys

- Each value (binary or JSON) is identified by a unique key, defined by the user or application when the item is saved.
- The key is immutable: once the item is saved, the key cannot be changed.
- Couchbase also refers to an item's key as its id.

Each key:

- Must be a UTF-8 string with no spaces. Special characters, such as (, %, /, ", and _, are acceptable.
- May be no longer than 250 bytes.
- Must be unique within its bucket.

Values

- Maximum size of a value is 20 MiB.

Value can be either:

- Binary: Any form of binary is acceptable. Note that a binary value cannot be parsed, indexed, or queried: it can only be retrieved by key.
- JSON: A JSON value, referred to as a document, can be parsed, indexed, and queried. 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, such as an embedded document or an array

Metadata

- *Metadata* is automatically generated and stored for each item saved in Couchbase Server
- Couchbase Server generates metadata in association with the document, when the document is saved.

Metadata and data


```
{ "airportname": "Brienne Le Chateau",
 "city": "Brienne-le Chateau",
 "country": "France",
 "faa": null,
 "geo":
 {
     "alt": 381,
     "lat": 48.429764,
      "lon": 4.482222 },
"icao": "LFFN",
"id": 1306,
"type": "airport",
 "tz": "Europe/Paris" }
```



```
{ "meta":
{
"id": "airport_1306",
 "rev": "1-
1526338d1bbb00000000000002000000",
"expiration": 0,
 "flags": 33554432,
 "type": "json" },
"xattrs": {}
}
```


Meta Data

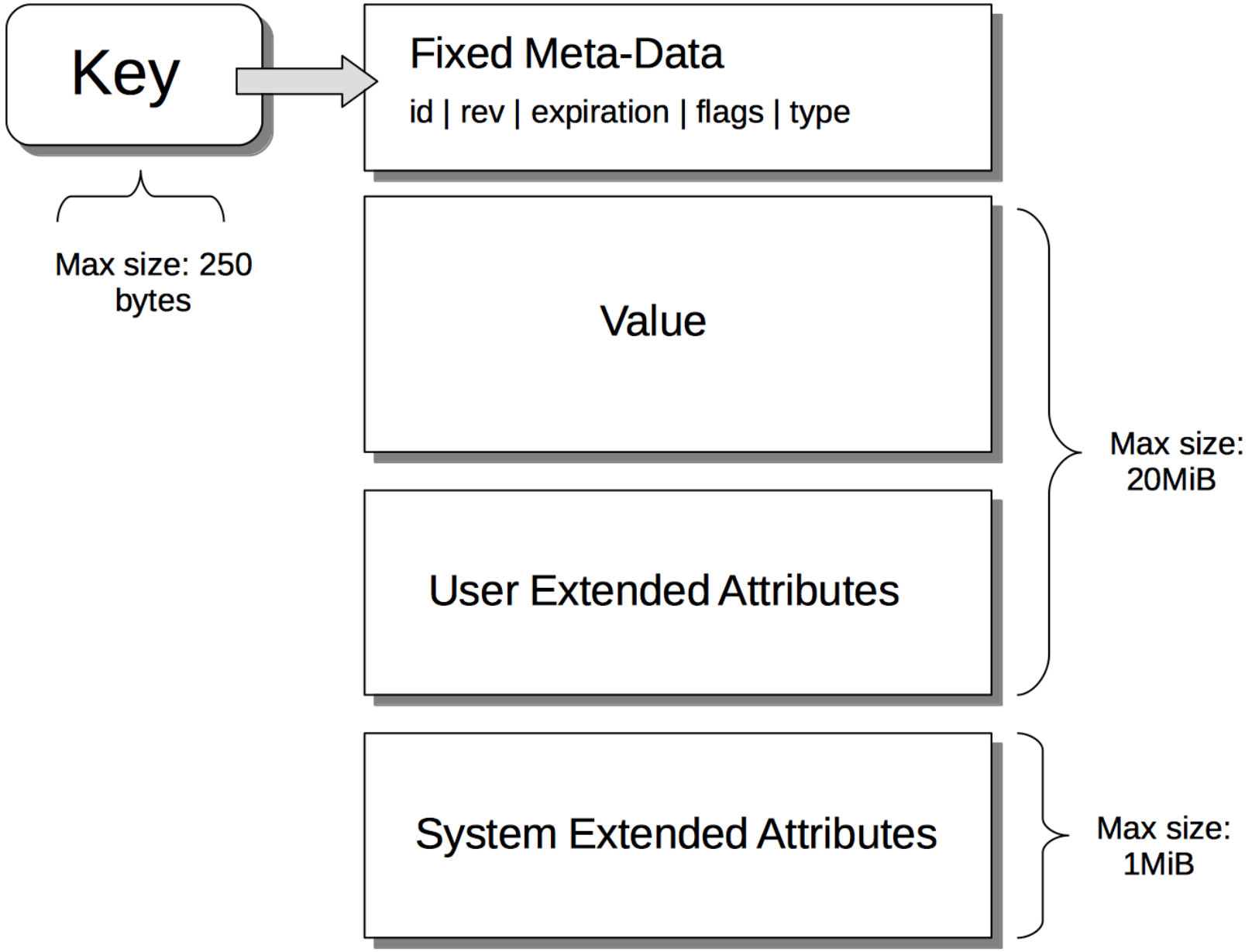
Attributes within metadada

- meta: The attribute whose value is the standard metadata for the saved document, airport-1306.
- id: The key of the saved document, which is airport_1306.
- rev: The revision or sequence number.
  - Value is for internal server-use only:
  - Used in the resolution of conflicts that occur when replicated documents are updated concurrently on different servers, representing the number of times the document has been mutated.
- expiration: The expiration-time (or Time-To-Live) of the document.
  - If non-zero, this determines the point at which the document is removed from the system.
  - The value can be set either per-document, by means of Couchbase SDK APIs (which is referred to as TTL); or per bucket, by means of Couchbase Web Console, the Couchbase CLI, or the Couchbase REST API (which is referred to as Bucket TTL).

Attributes within metadada

- flags: Couchbase SDK-specific values that may be used to identify the type of data saved, or to specify formatting.
- type: The type of the saved value, which in this case is json.
- xattrs: Extended Attributes, which constitute a special kind of metadata, some of which is system-internal, some of which can optionally be written and read by user-applications

Size limits


Key
Fixed Meta-Data
id | rev | expiration | flags | type
Max size: 250 bytes
Value
User Extended Attributes
Max size: 20MiB
System Extended Attributes
Max size: 1MiB

Relational database

```
EMPLOYEE
Name | SSN | Wage
-------------------------------------
Jamie | 234 | 123
Steve | 123 | 456

SCHEMA:
Name -> String of width 100
SSN -> Number of width 9
Wage -> Number of width 10

EMPLOYERS:
-------------------------------------
 Name_Key   | Company    | Start | End
 Jamie      | Yahoo      | 2005  | 2006
 Jamie      | Oracle     | 2006  | 2012
 Jamie      | Couchbase  | 2012  | NULL
```

Couchbase documents

```
(HRData keyspace)
{
     'Name': 'Jamie'
     'SSN': 234
     'Wage': 123
     'History':
      [
        ['Yahoo', 2005, 2006],
        ['Oracle', 2006, 2012],
      ]
},

{
     'Name': Steve
     'SSN':  123,
     'Wage': 456,
}
```

Data Projection

```
(HRData keyspace)
{
     'Name': 'Jamie'
     'SSN': 234
     'Wage': 123
     'History':
      [
       ['Yahoo', 2005, 2006],
       ['Oracle', 2006, 2012],
     ]
},

{
     'Name': Steve
     'SSN':  123,
     'Wage': 456,
}
```

Query:

```
SELECT Name, History, {'FullTime': true} AS 'Status'
     FROM HRData
```

Result:

```
{
'Name': 'Jamie',
'History':
        [
        ['Yahoo', 2005, 2006],
        ['Oracle', 2006, 2012],
        ['Couchbase', 2012, null]
        ],
'Status': { 'FullTime': true }
}
{
'Name': 'Steve',
'Status': { 'FullTime': true }
}
```

Extended Attributes - XATTRs

- Couchbase Server permits the definition of extended attributes.
- Allow developers to define application-specific metadata visible only to those applications that request it or attempt to modify it.
- May be metadata specific to a programming framework that should be hidden by default from other frameworks, or possibly from other versions of the same framework.
- Extended attributes can be formed only as JSON, but can be applied to both JSON and binary items.
- Creator-application can subsequently access and modify the extended attributes it has created within a document.
- However, no other application has knowledge of the extended attributes within a document other than their creator; unless such knowledge is shared explicitly between applications by some mechanism external to Couchbase Server.

Implications for Sizing

- Within Couchbase Server, the maximum size for each document is 20 megabytes
- Extended attributes count against this size-limit
- Consequently, the size of a document may reflect the presence of data inaccessible to some applications

Virtual Extended Attributes

- A virtual extended attribute is server-defined, and exposes additional information about a document.
- Virtual extended attributes are sometimes referred to as VXATTRs.
- Couchbase Server provides the virtual extended attribute, $document, which, when specified as a search-path, returns metadata on the document.

Durability

- Durability ensures the greatest likelihood of data-writes surviving unexpected anomalies, such as node-outages.
- Clients writing to Couchbase Server can optionally specify durability requirements; which instruct Couchbase Server to update the specified document on multiple nodes in memory and/or disk locations across the cluster; before considering the write to be committed.
- The greater the number of memory and/or disk locations specified in the requirements, the greater the level of durability achieved.
- Once the write has been committed as specified by the requirements, Couchbase Server notifies the client of success.
- If commitment was not possible, Couchbase Server notifies the client of failure; and the data retains its former value throughout the cluster.

Durability

- Couchbase Server supports durability for single-document writes only. This form of write is referred to as a durable or synchronous write.
- Couchbase Server supports durability for up to two replicas: it does not support durability for three replicas.
- Durability is used by Transactions, which support the atomicity of mutations across multiple documents

Regular and Durable writes

Regular write

- Is *asynchronous*, and is not supported by durability.
- Such a write may be appropriate when writing data whose loss would produce only a minor inconvenience.
- For example, a single instance of sensor data, out of the tens of thousands of instances that are being continuously received.

Durable write

- Is *synchronous*, and is supported by durability.
- Such a write may be appropriate when saving data whose loss could have a considerable, negative impact.
- For example, data corresponding to a financial transaction.

Majority

- Client-specified durability requirements use the concept of majority, to indicate the number of configured Data Service nodes to which commitment is required, based on the number of replicas defined for the bucket.

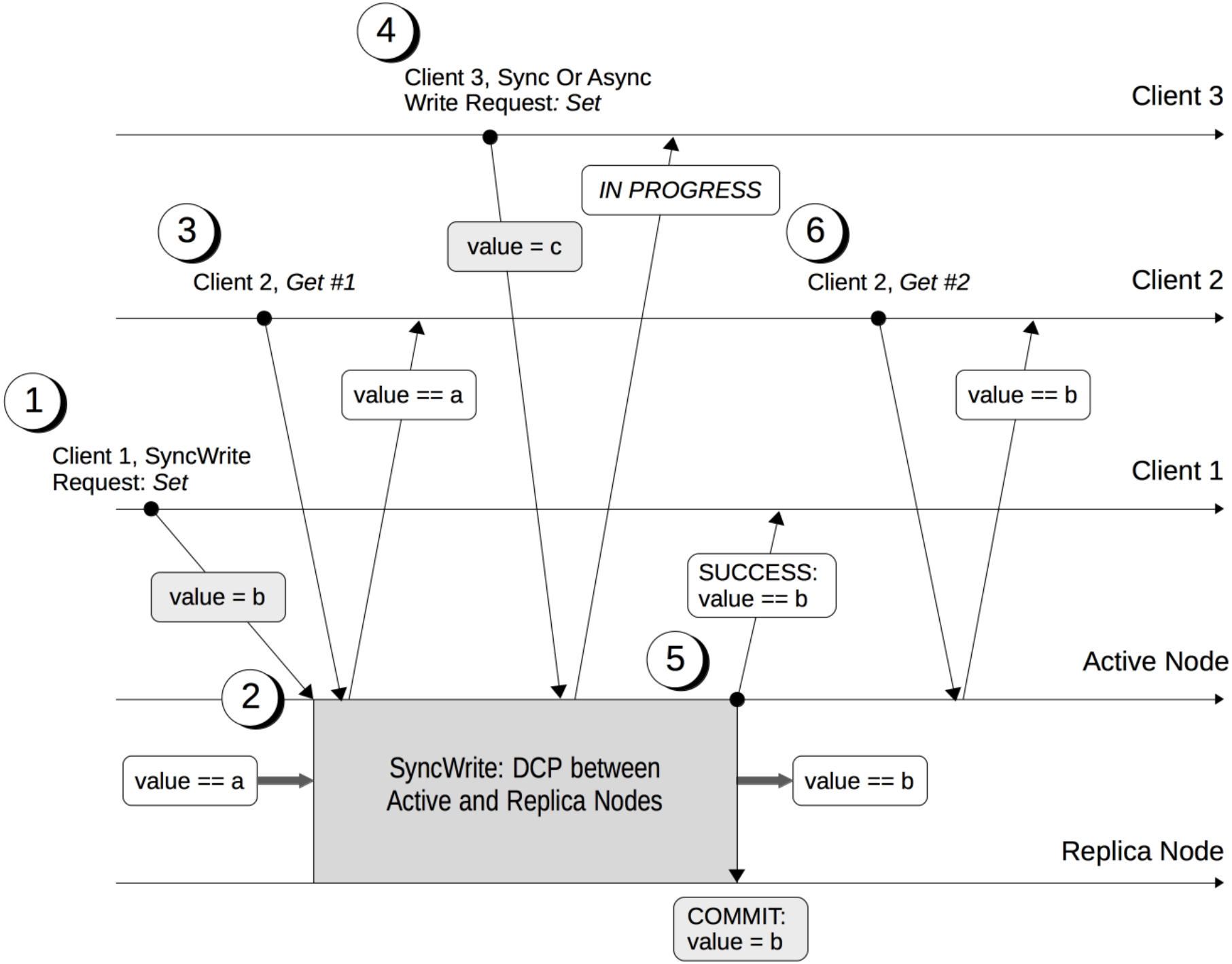
Number of Replicas	Number of Nodes Required for Majority
0	1
1	2
2	2
3	*Not supported*

Durability Requirements

Level :The level of durability required. The possible values are:

- majority. The mutation must be replicated to (that is, held in the memory allocated to the bucket on) a majority of the Data Service nodes.
- majorityAndPersistActive. The mutation must be replicated to a majority of the Data Service nodes. Additionally, it must be persisted (that is, written and synchronised to disk) on the node hosting the active vBucket for the data.
- persistToMajority. The mutation must be persisted to a majority of the Data Service nodes. Accordingly, it will be written to disk on those nodes.

Timeout: The number of milliseconds within which the durability requirements must be met. The SDK specifies a default of 2500


4
Client 3, Sync Or Async
Write Request: Set
Client 3
IN PROGRESS
value = c
3
6
Client 2, Get #1
Client 2, Get #2
Client 2
value == a
value == b
1
Client 1, SyncWrite
Request: Set
Client 1
value = b
SUCCESS:
value == b
5
Active Node
2
value == a
SyncWrite: DCP between
Active and Replica Nodes
value == b
Replica Node
COMMIT:
value = b

Lifecycle of a durable write

- Client 1 specifies durability requirements for a durable write, to change a key's existing value from a to b.
- The Active Node receives the request, and the durable write process is initiated. Couchbase Server attempts to meet the client's specified durability requirements.
- During the durable write process, Client 2 performs a read on the value undergoing the durable write. Couchbase Server returns the value, a, that preceded the durable-write request.
- During the durable-write process, Client 3 attempts either a durable write or a regular write on the value that is already undergoing a durable write. Couchbase Server returns a SYNC_WRITE_IN_PROGRESS message, to indicate that the new write cannot occur.
- At the point the mutation has met the specified durability requirements, the Active Node commits the durable write, and sends a status response of SUCCESS to Client 1.
- After the durable-write process, Client 2 performs a second read on the value. Couchbase Server returns the value, b, committed by the durable write. Indeed, from this point, all clients see the value b.

Regular Writes

- If a client writes data to Couchbase Server without specifying durability requirements, this is considered a regular (that is asynchronous) write.
- No durability requirement is imposed. Couchbase Server acknowledges success to the client as soon as the data is in the memory of the node hosting the active vBucket:
- Couchbase Server does not confirm that the write has been propagated to any replica.
- A regular write therefore provides no guarantee of durability.

Failure Scenarios

A durable write fails in the following situations:

- Server timeout exceeded
  - The active node aborts the durable write, instructs all replica nodes also to abort the pending write, and informs the client that the durable write has had an ambiguous result.
- Replica node fails while SyncWrite is pending
  - (that is, before the active node can identify whether the node hosted a replica).
  - If enough alternative replica nodes can be identified, the durable write can proceed.
  - Otherwise, the active node waits until a server-side timeout has expired; then aborts the durable write, and duly informs the client that the durable write has had an ambiguous result.

Failure Scenarios

A durable write fails in the following situations:

- Active node fails while SyncWrite is pending.
  - This disconnects the client, which must assume that the result of the durable write has proved ambiguous.
  - If the active node is failed over, a replica is promoted from a replica node: depending on how advanced the durable write was at the time of active-node failure, the durable write may proceed.
- Write while SyncWrite is pending.
  - A client that attempts a durable or an asynchronous write on a key whose value is currently undergoing a durable write receives a SYNC_WRITE_IN_PROGRESS message, to indicate that the new write cannot currently proceed. The client may retry.

Handling Ambiguous Results

- Couchbase Server informs the client of an ambiguous result whenever Couchbase Server cannot confirm that an intended commit was successful. T
- Situation may be caused by node-failure, network-failure, or timeout.
- If a client receives notification of an ambiguous result, and the attempted durable write is idempotent, the durable write can be re-attempted.
- If the attempted durable write is not idempotent, the options are:
  - Verify the current state of the saved data; and re-attempt the durable write if appropriate.
  - Return an error to the user.

Transactions

- Transactions guarantee that multiple documents can be updated atomically.
- Distributed ACID Transactions are operations that ensure that when multiple documents need to be modified such that only the successful modification of all justifies the modification of any, either all the modifications do occur successfully; or none of them occurs.
- This Atomicity supports insert, update, and delete operations, across any number of documents.
- Transactions make use of the Durability provided by synchronous writes.

Transactions

- During execution, for each document, a transaction maintains a modified copy of the document in the Extended Attributes area of the document's meta data.
- Consequently, the changes that occur during the transaction prior to commitment — these constituting uncommitted (or dirty) data — are non-readable by any operation other than this transaction itself.
- Only following commit of the documents modified by the transaction — when the modified data-copy is removed from the meta data area, and is written over the main body of the document's data — do the corresponding changes become readable by other transactions and operations.
- This isolation-level is known as Read Committed.
- Note that this use of a document's extended attributes area results in an approximate doubling of its pre-transaction size, while the transaction is underway. In consequence, transactions can only be used on documents whose maximum size is 10 MB (which is half of the maximum-permitted document-size, 20 MB).

Transactions

- As well as making use of individual documents' extended attributes, transactions also create additional documents in each bucket that they access.

Include :

- Active Transaction Records
  - (up to 1024 of which can exist per bucket, and whose names are prefixed with _txn:atr-),
- Transaction Client Records
  - (one of which exists per bucket, duly named _txn:client-record).

- These documents are automatically maintained by Couchbase Server, and should not be modified by any application.

Transactions

- Only nodes that contain data to be updated are affected by a transaction.
- Multiple transactions can read the same document at the same time. If two transactions simultaneously attempt to write to the same document, one is allowed to proceed, while the other is obliged to retry.
- Use of transactions requires *Network Time Protocol* (NTP) to be used to synchronize time across all cluster-nodes.

Limitations on transactions

Only documents whose initial size is 10 MB or less can be included in a transaction.

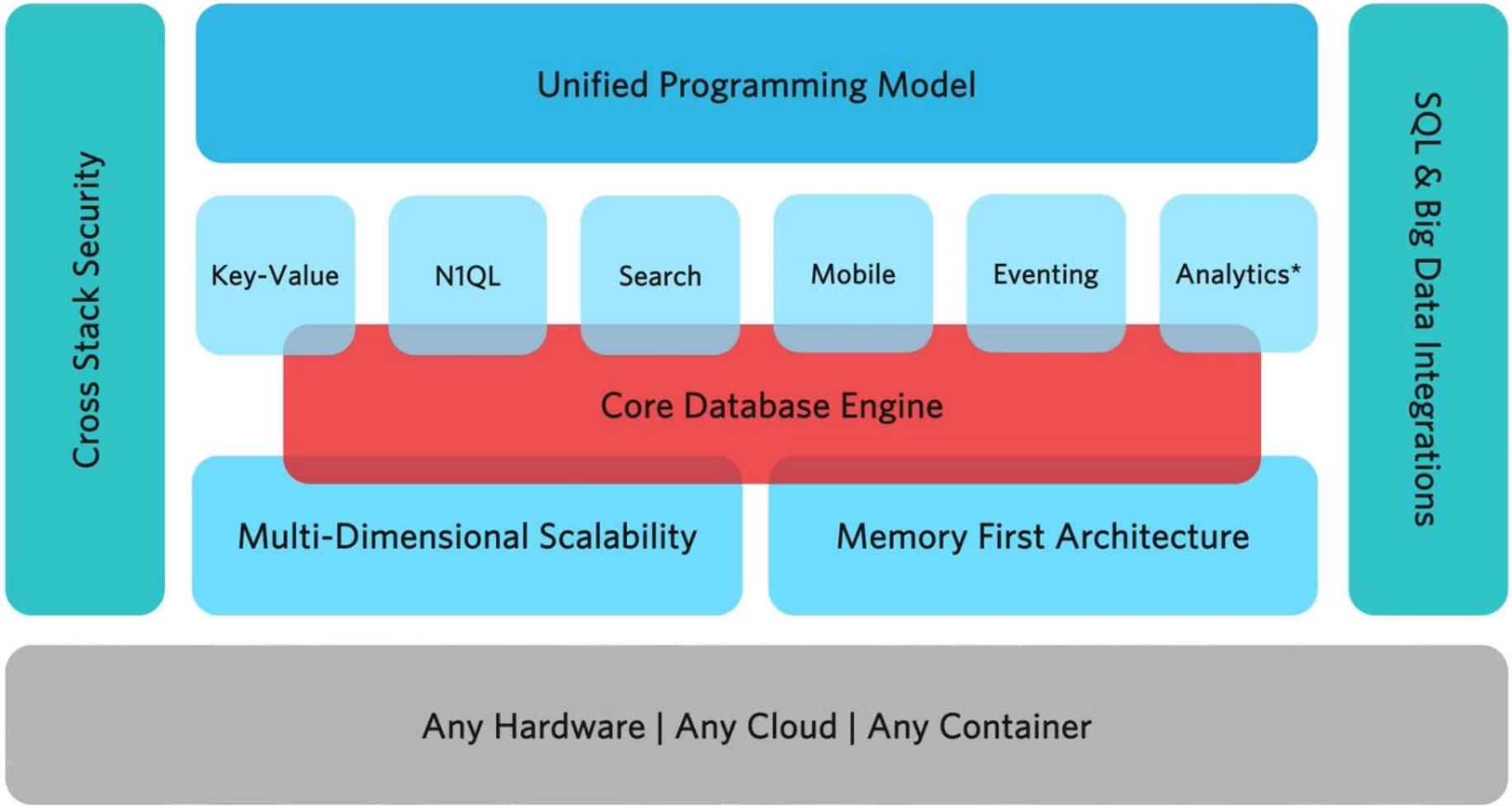
Non-transactional updates should not be made to any document involved in a transaction while the transaction is itself in progress: this prevents the non-transactional update from being overwritten.

The number of writes required by a transactional update is greater than the number required for a non-transactional update: a transactional update requires writes in order to stage and commit data, and also to maintain transaction records. Consequently, transactional updates may be less performant than non-transactional updates.

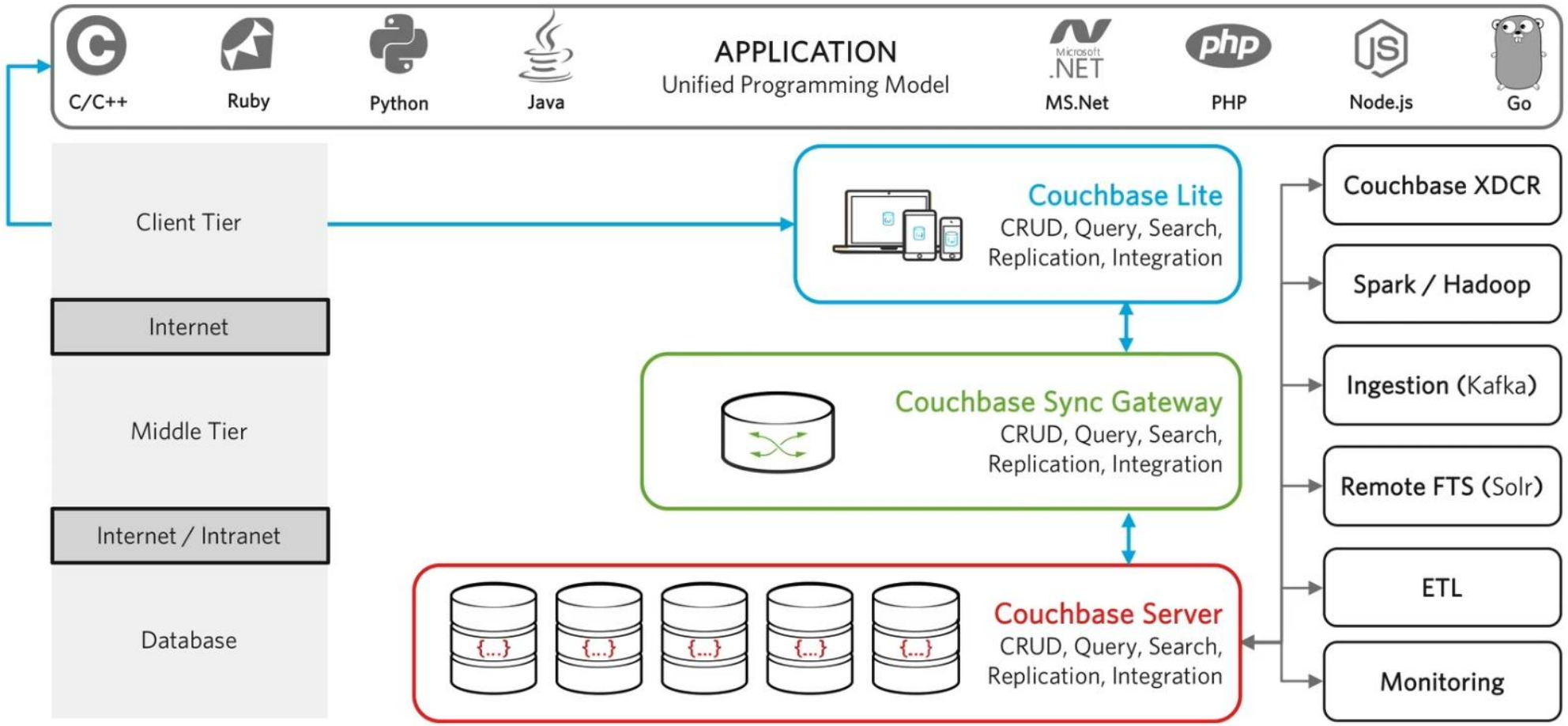
Note that data within a single document is always updated atomically: therefore, whenever multiple key-value pairs consistently require atomic update, their co-location within a single document may best ensure high performance.

Cross Data Center Replication (XDCR) supports eventual consistency: however, it does not support atomicity — nor does XDCR Conflict Resolution. Consequently, transactionally modified documents should only be replicated across clusters if no transactions involving the same documents can occur on those clusters simultaneously.

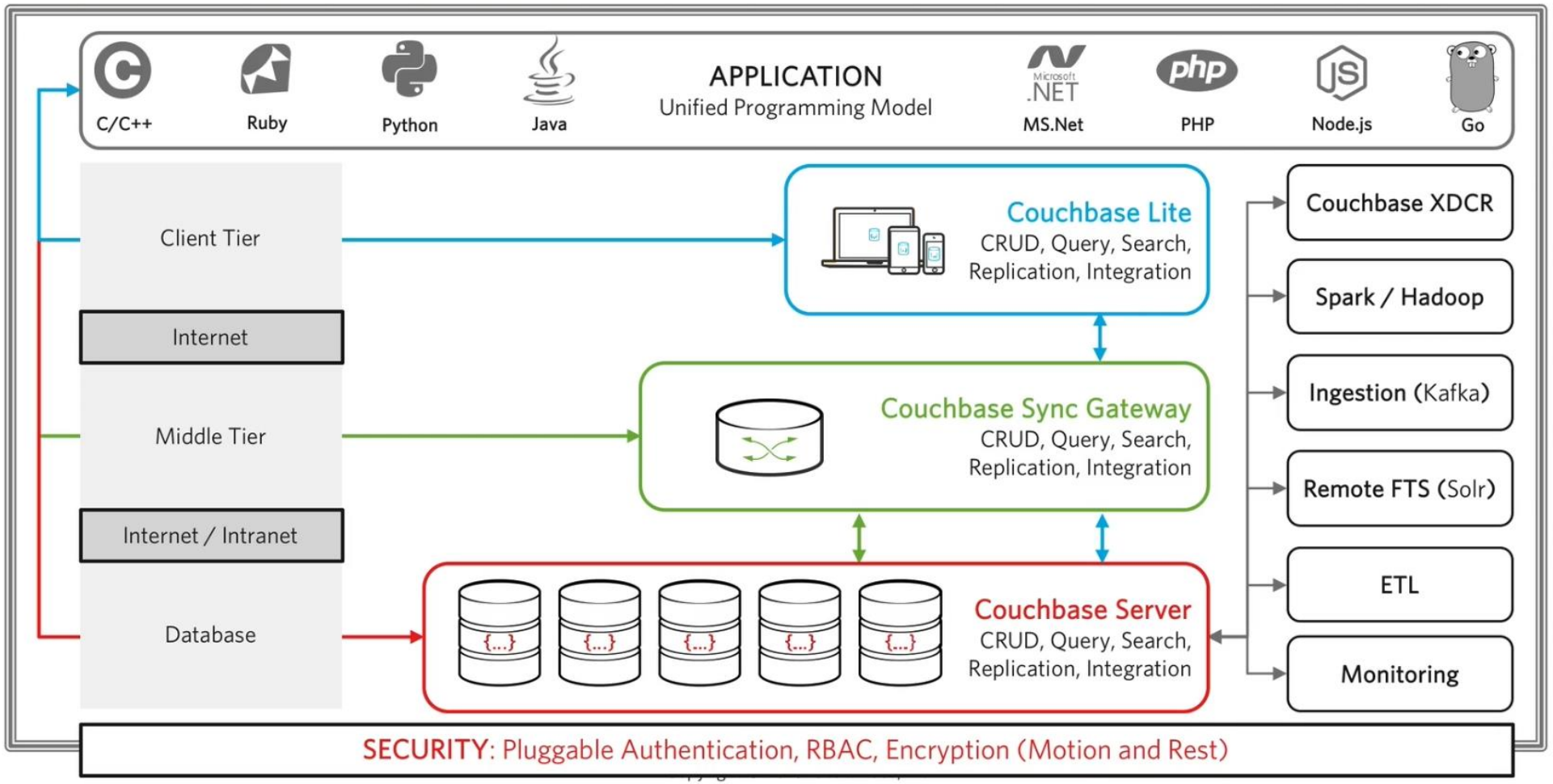
What comprises the Couchbase Data Platform?



How can you access the Couchbase stack?



How can you access the Couchbase stack?



How does Couchbase store data?

Key-Value Pairs

```
2014-06-23-10:15am : 75F

2014-06-23-11:30am : 77F

2014-06-23-02:00pm : 82F
```

Documents

```
user::0001 :
{
     "first_name"  : "Rhonda",
     "last_name"   : "Red",
     "admin"       : true,
     "postal_code" : 97203
}
```

Key ("Document ID")

Any string up to 25 bytes
(~250 UTF-8 characters)

Value

Any value up to 20mb
(~5m to ~20m UTF-8 chars, vary by byte length)

How does Couchbase store data?

Key-Value Pairs

```
2014-06-23-10:15am : 75F

2014-06-23-11:30am : 77F

2014-06-23-02:00pm : 82F
```

Documents

```
user::0001 :
{
     "first_name"  : "Rhonda",
     "last_name"   : "Red",
     "admin"       : true,
     "postal_code" : 97203
}
```

Key ("Document ID")

Any string up to 25 bytes
(~250 UTF-8 characters)

Value

Any value up to 20mb
(~5m to ~20m UTF-8 chars, vary by byte length)

What is Couchbase's high level architecture?

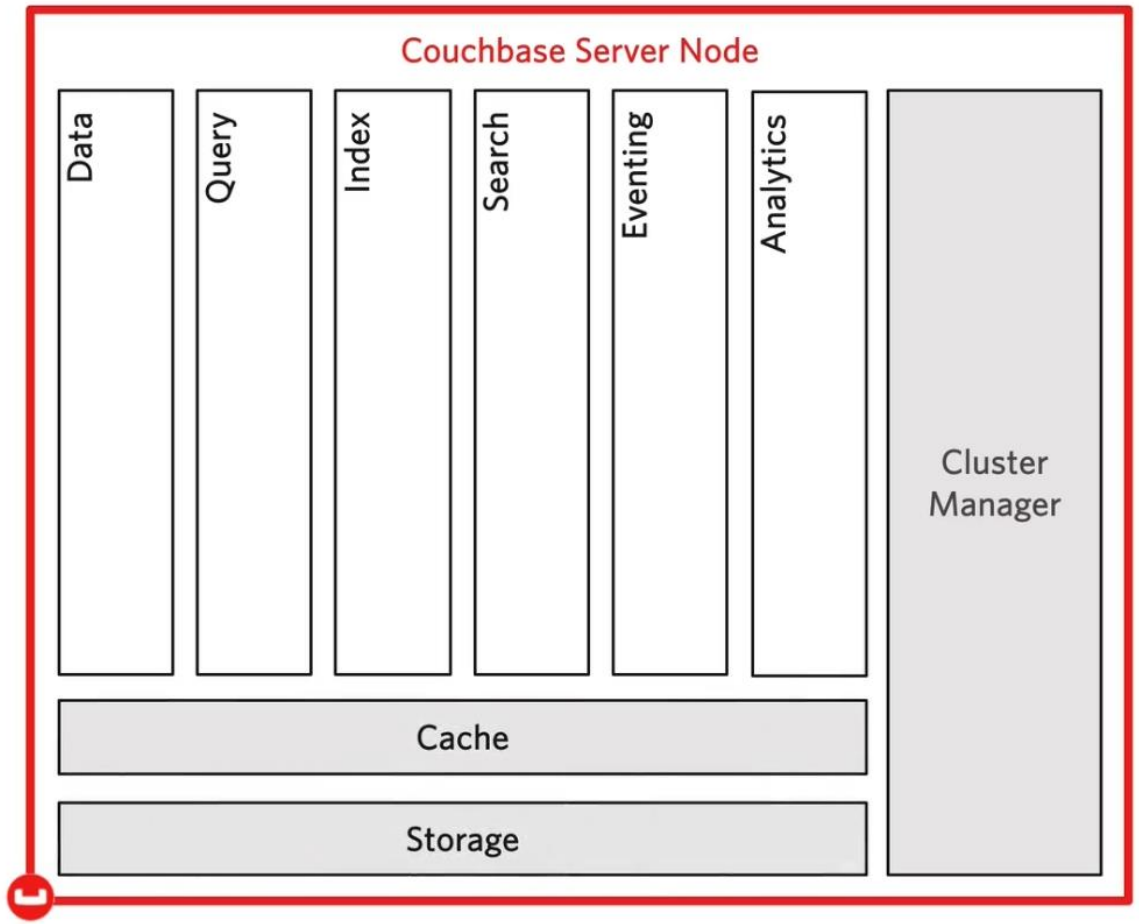
Couchbase nodes are identical

Cluster Manager

Six *independently scalable* services

- ✓ Data
- ✓ Query
- ✓ Index
- ✓ Search
- ✓ Eventing
- ✓ Analytics*

Cache and storage as needed



What is Couchbase's high level architecture?

Data Service
- ✓ Buckets, Key-Value APIs
- ✓ Data replication, MapReduce views

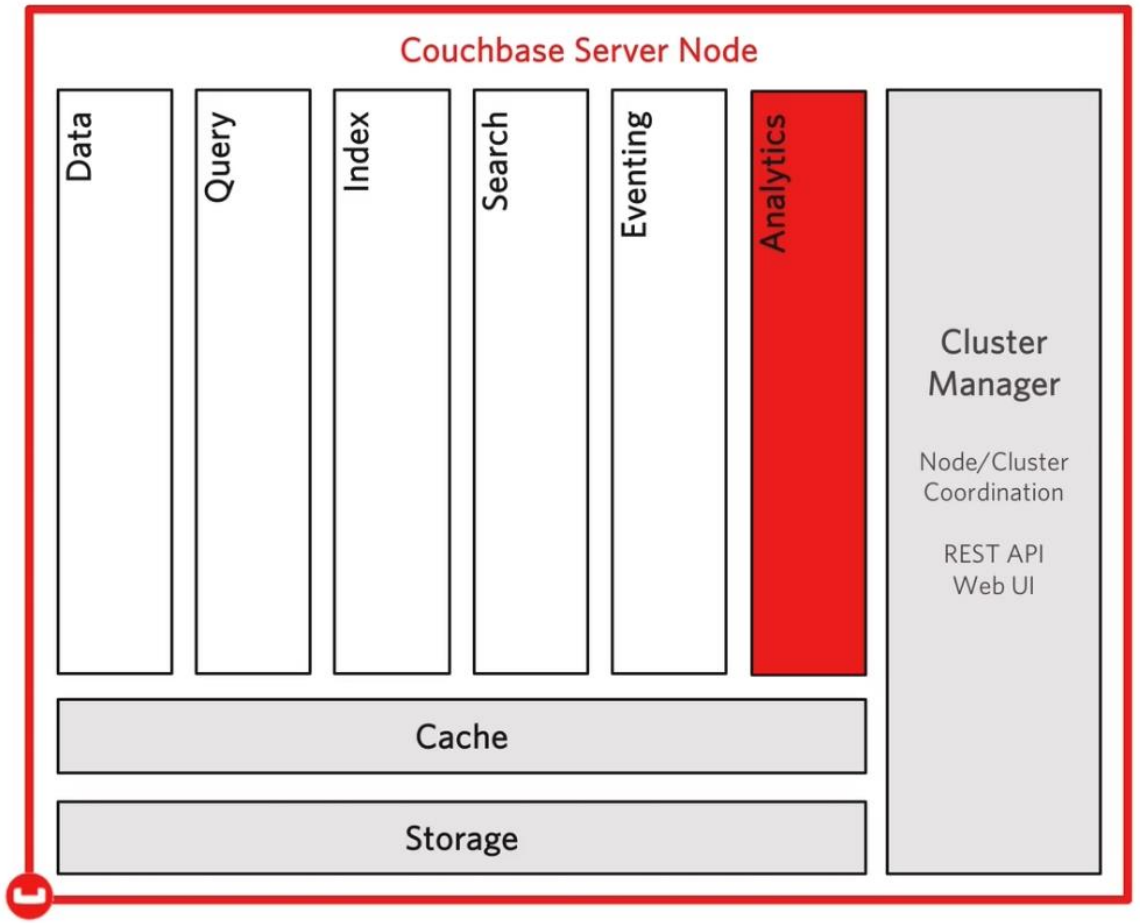
Query Service
- ✓ N1QL (SQL for JSON)
- ✓ Query planning and optimization

Index Service
- ✓ Global secondary indexes
- ✓ In-memory lock free maintenance

Search Service
- ✓ Full Text search, faceting, scoring, etc.
- ✓ Index replication

Eventing Service
- ✓ Business rules, real-time processing
- ✓ Middleware independent

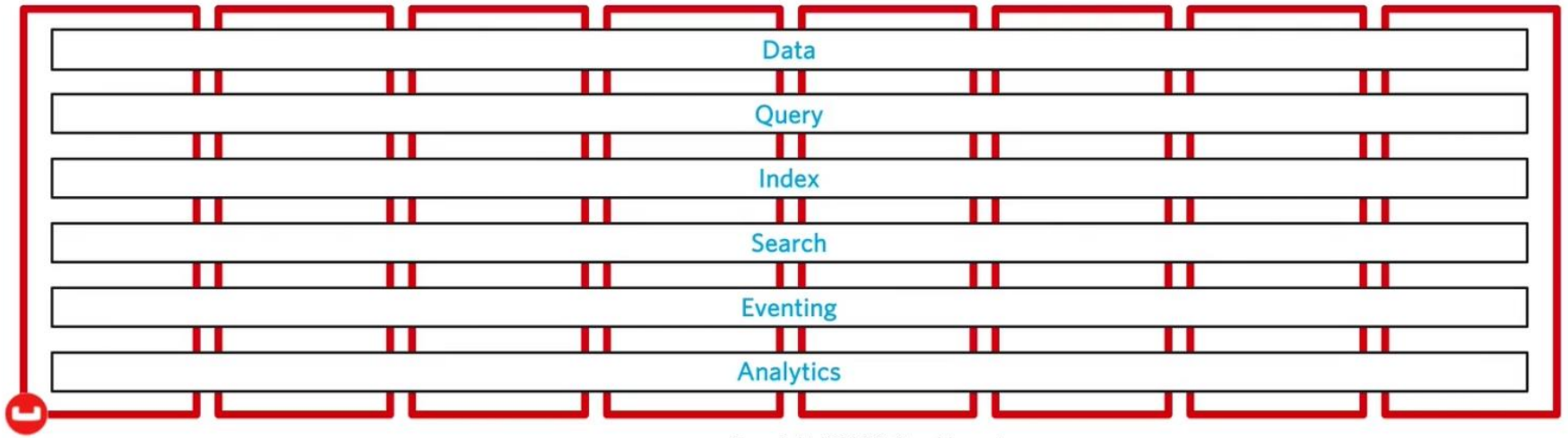
Analytics Service (5.x preview)
- ✓ Batched retrospective queries
- ✓ Workload distributed & isolated



How does Couchbase scale?

✓ Stripe across all cluster nodes (default)

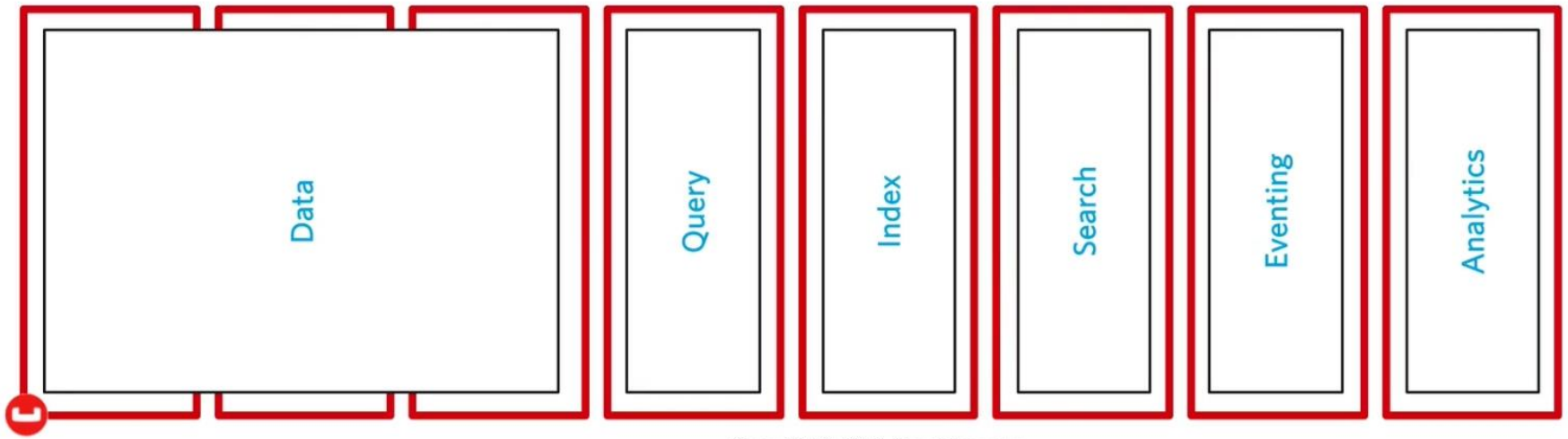
Single node type with all services



How does Couchbase scale?

Single node type with all services

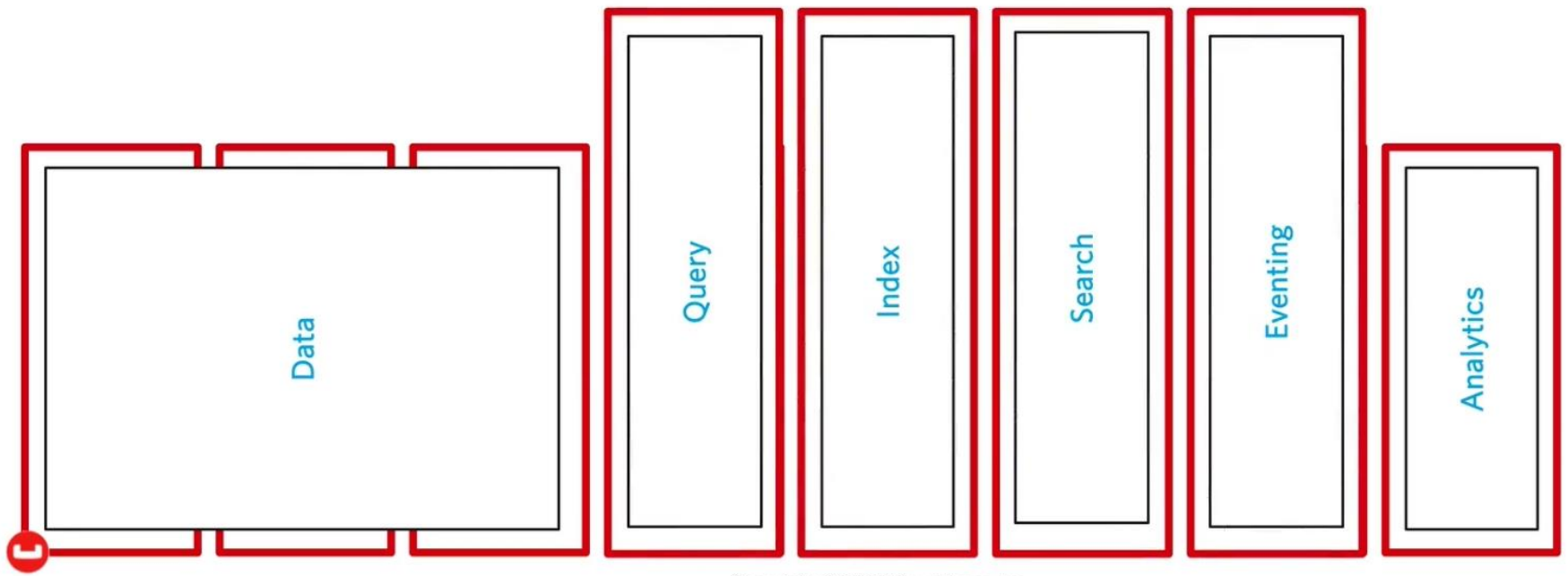
- ✓ Stripe across all cluster nodes (default)
- ✓ Isolate services to optimize workload by hardware



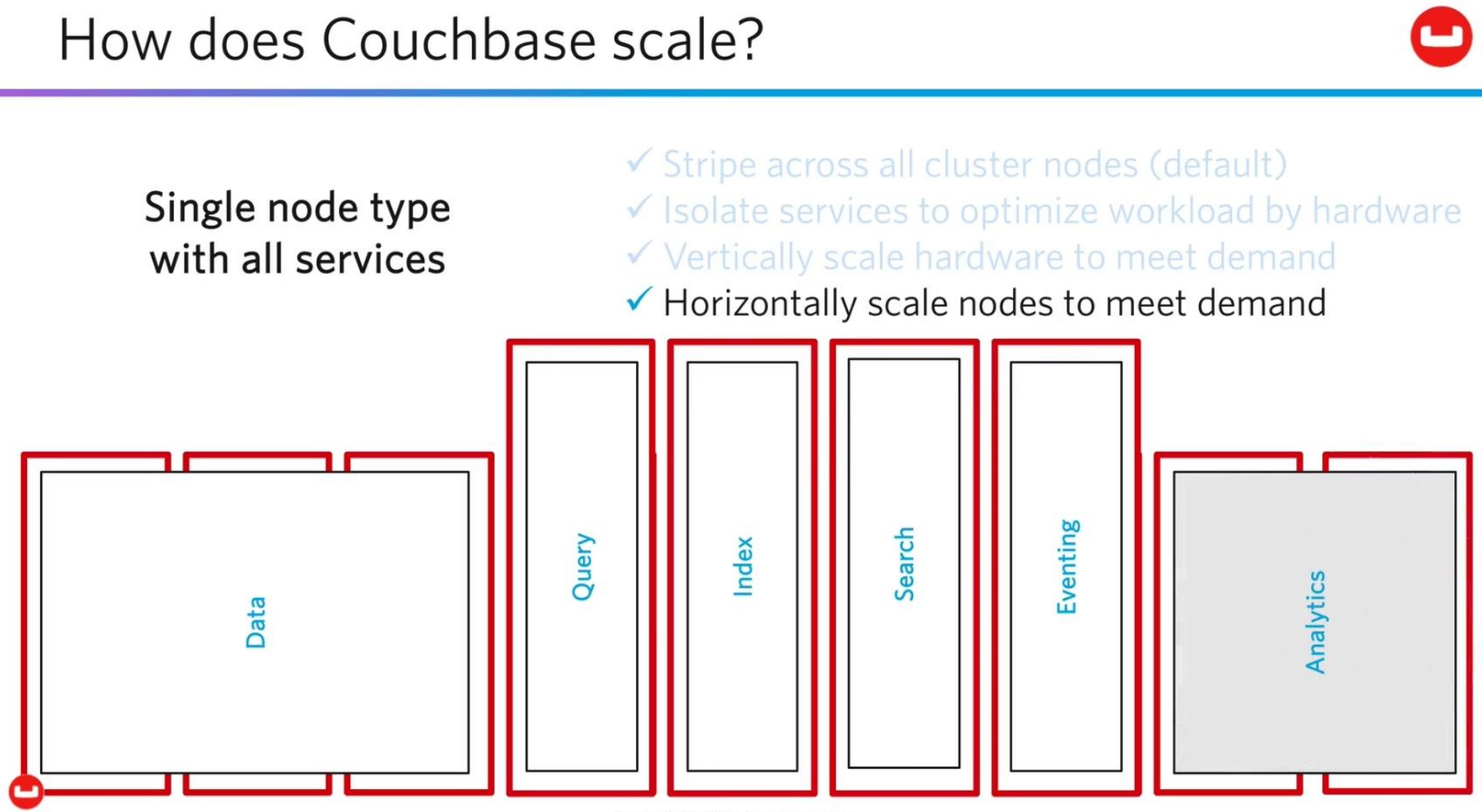
How does Couchbase scale?

Single node type with all services

- ✓ Stripe across all cluster nodes (default)
- ✓ Isolate services to optimize workload by hardware
- ✓ Vertically scale hardware to meet demand



How does Couchbase scale?



How does Couchbase expose data?

Key-Value API Operations

- ✓ get, insert, upsert, replace, remove
- ✓ sub-documents, data structures

Aggregate Views

- ✓ count, sum, evaluate aggregate data
- ✓ distributed map-reduce functions

N1QL Queries

- ✓ select, insert, update, upsert, delete
- ✓ where, join, having, group by, etc.

Full Text Search

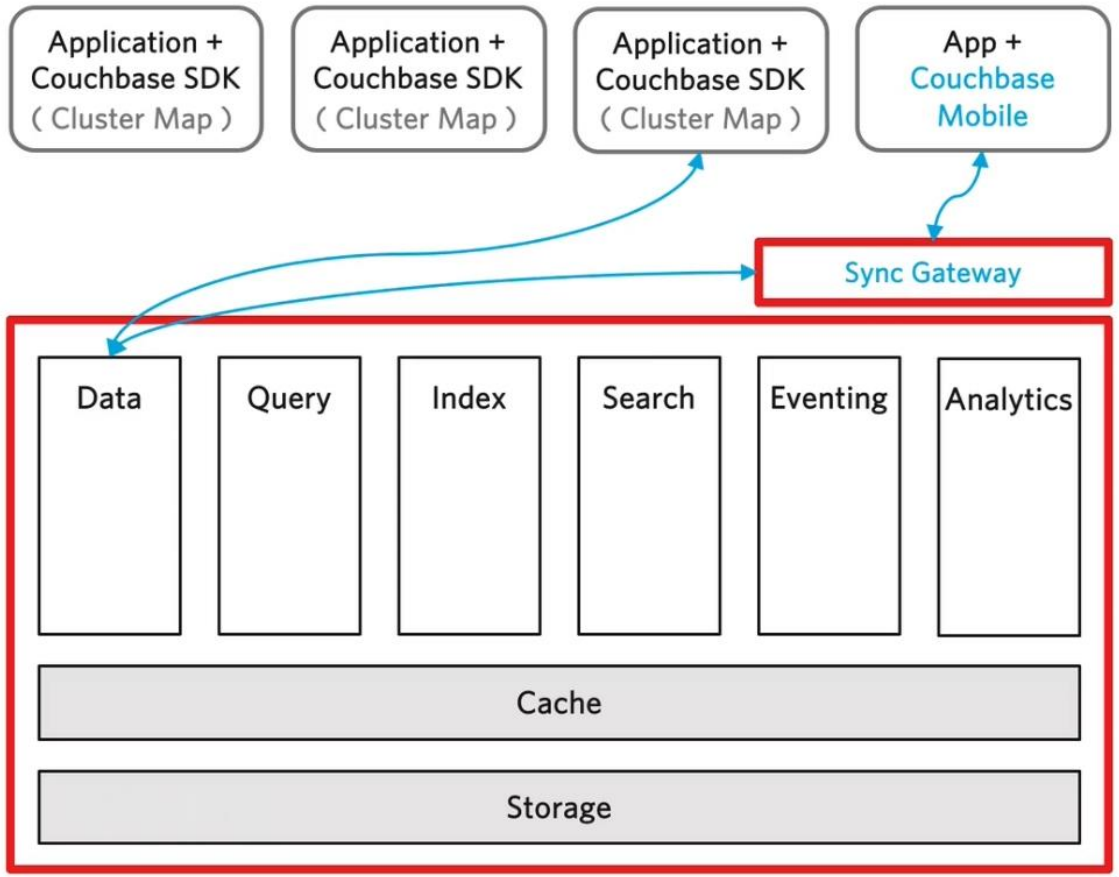
- ✓ natural language inverted index query
- ✓ tokenization, stems, facets, relevancy

Eventing

- ✓ functions triggered by data events
- ✓ Diagnosably respond to DCP actions

Analytics (5.x preview)

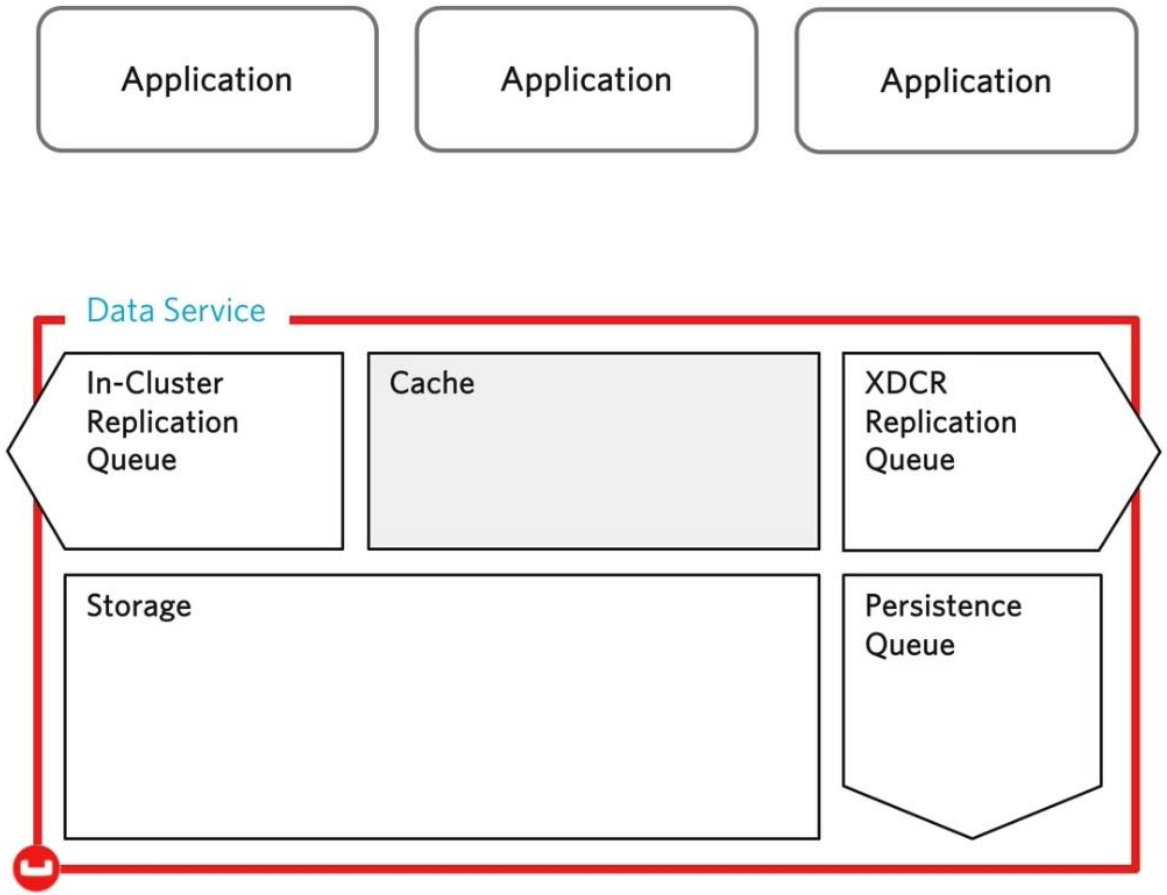
- ✓ continuous online querying
- ✓ workload distributed and isolated



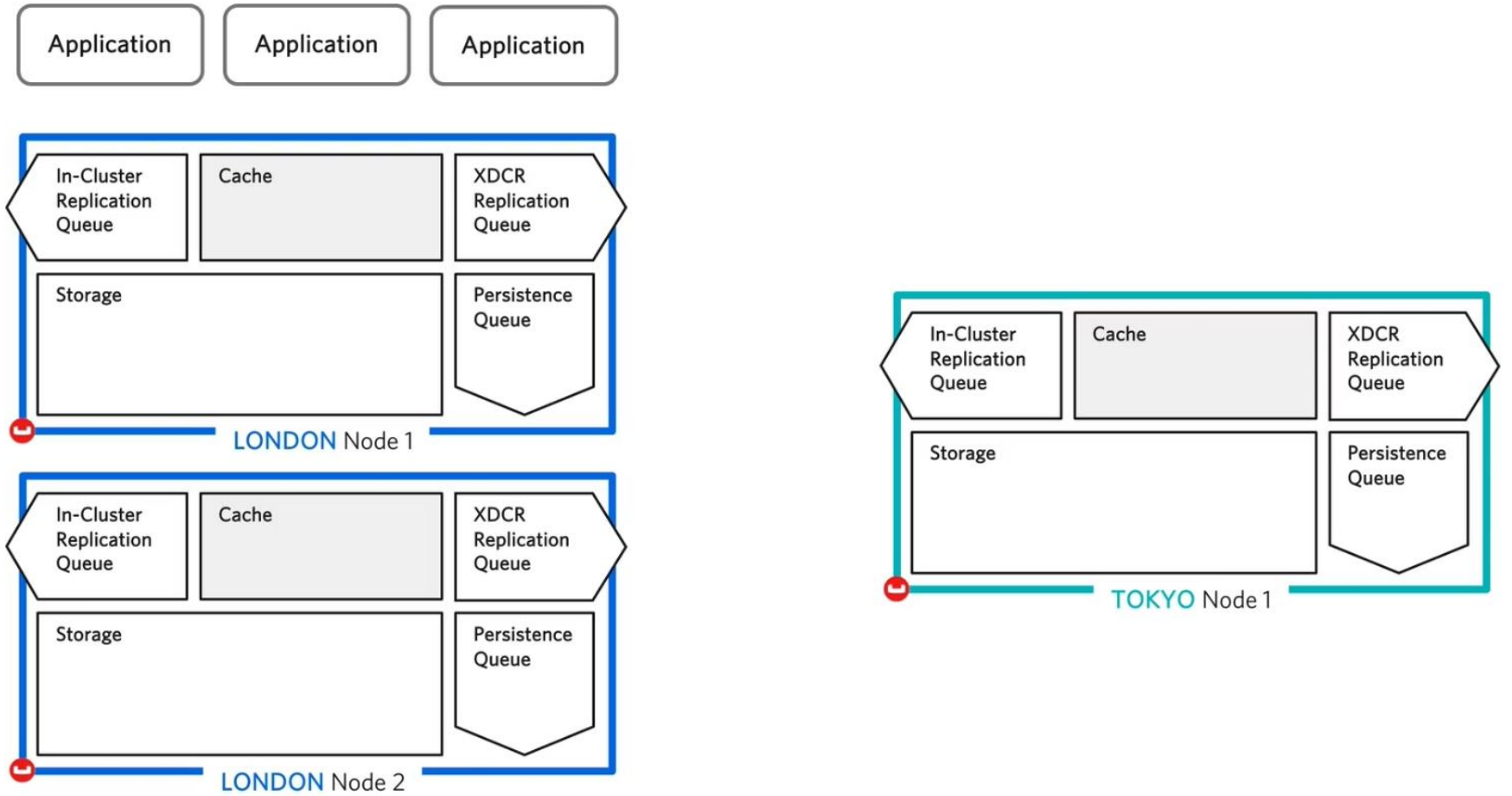
What are the main Data Service components?

Five key components

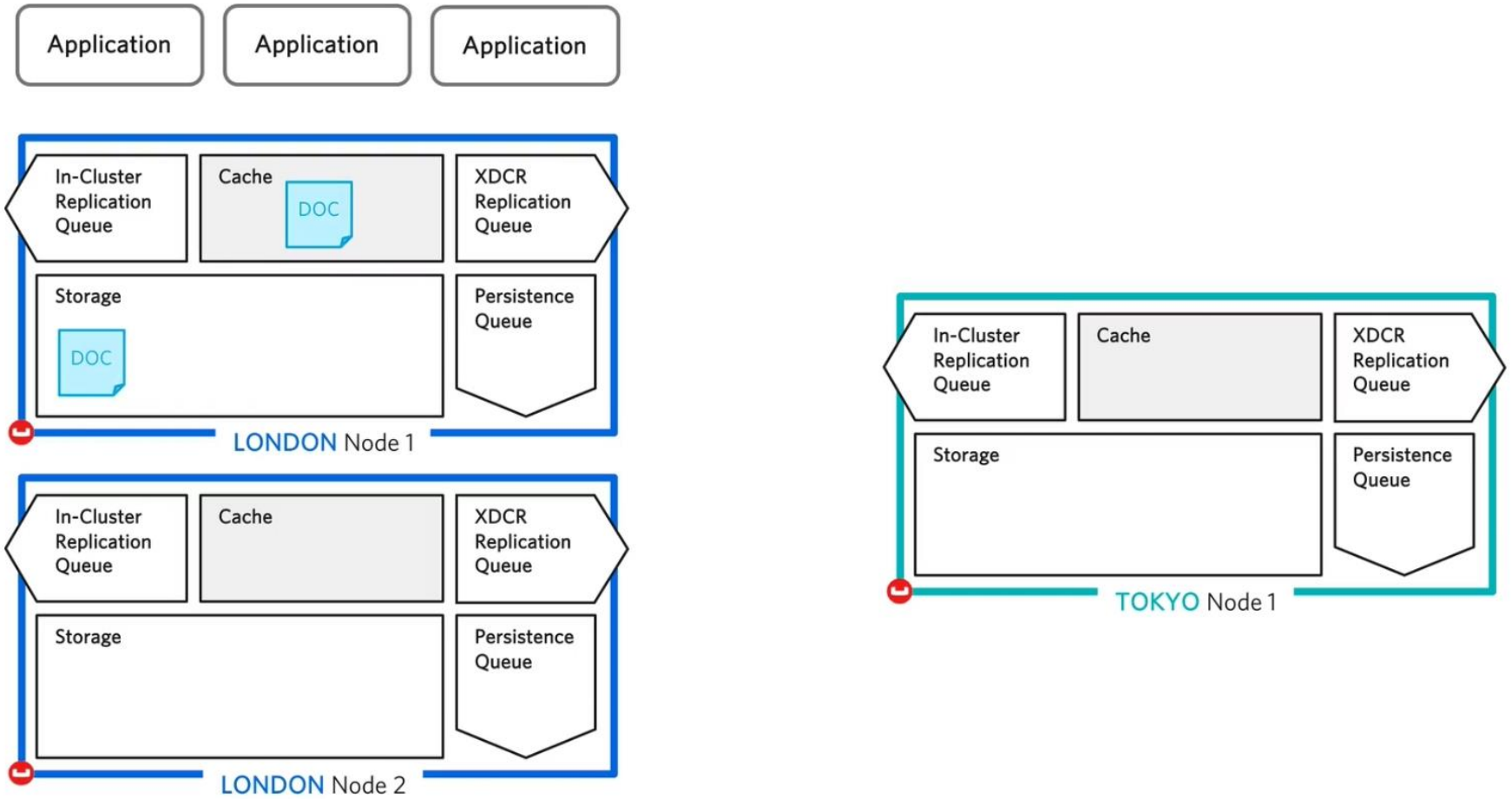
- ✓ Managed Cache
- ✓ Storage
- ✓ Persistence Queue
- ✓ In-Cluster Replication Queue
- ✓ XDCR Replication Queue



How do documents move through the Data Service?

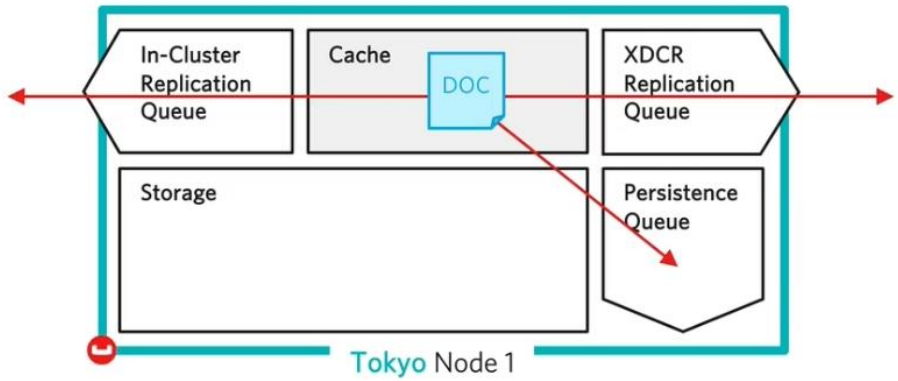


How do documents move through the Data Service?



What is the role of the Database Change Protocol?

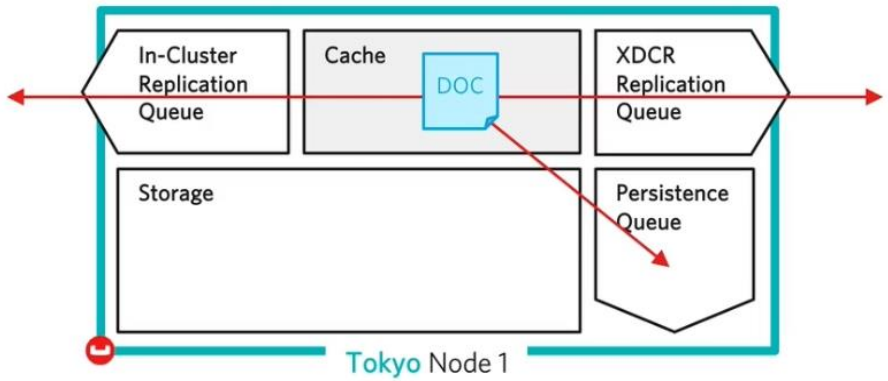
All replication relies on the Database Change Protocol (DCP)



What is the role of the Database Change Protocol?

All replication relies on the Database Change Protocol (DCP)

- ✓ No-loss recovery if interrupted
- ✓ RAM to RAM streaming
- ✓ Multi-threaded, parallel processing



How does Couchbase expose data?

Key-Value API Operations

- ✓ get, insert, upsert, replace, remove
- ✓ sub-documents, data structures

Aggregate Views

- ✓ count, sum, evaluate aggregate data
- ✓ distributed map-reduce functions

N1QL Queries

- ✓ select, insert, update, upsert, delete
- ✓ where, join, having, group by, etc.

Full Text Search

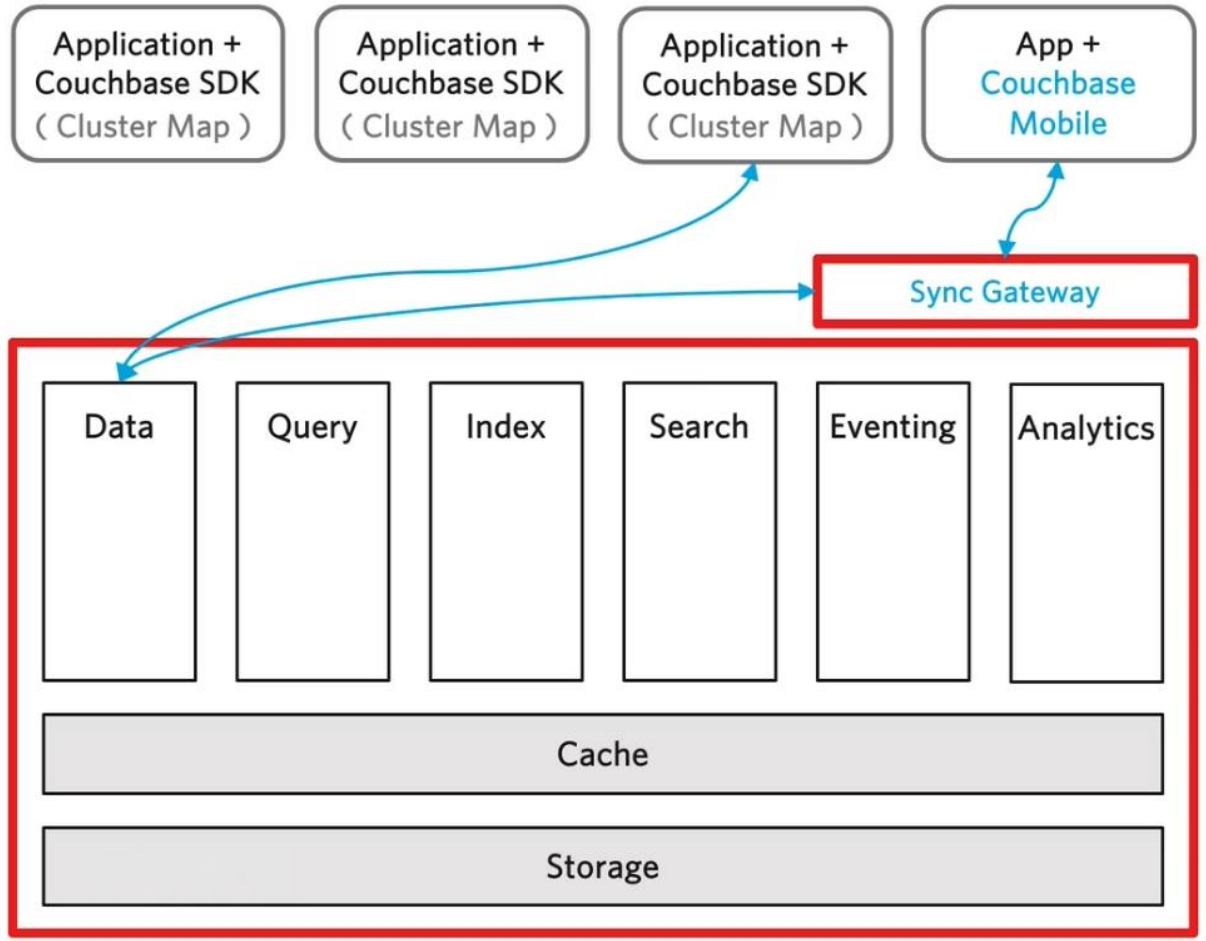
- ✓ natural language inverted index query
- ✓ tokenization, stems, facets, relevancy

Eventing

- ✓ functions triggered by data events
- ✓ Diagnosably respond to DCP actions

Analytics (5.x preview)

- ✓ continuous online querying
- ✓ workload distributed and isolated



How does Couchbase expose data?

Key-Value API Operations
- ✓ get, insert, upsert, replace, remove
- ✓ sub-documents, data structures

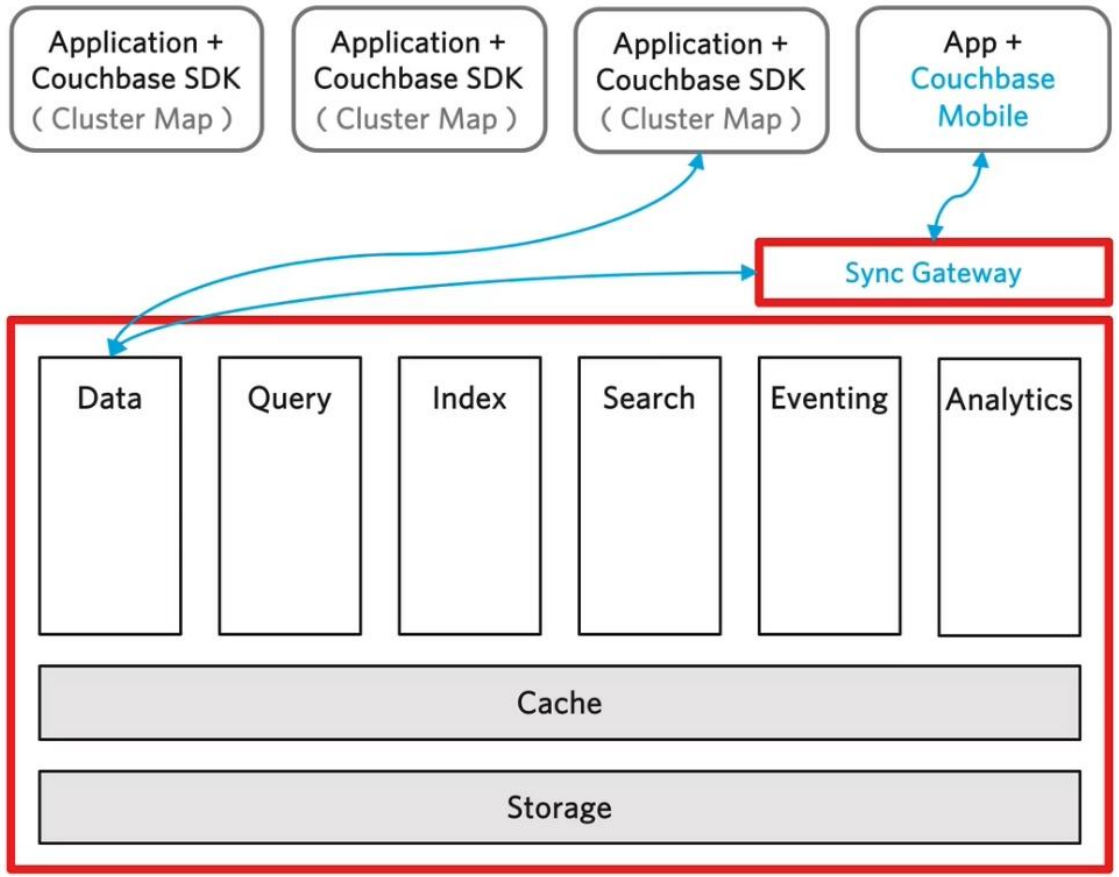
Aggregate Views
- ✓ count, sum, evaluate aggregate data
- ✓ distributed map-reduce functions

N1QL Queries
- ✓ select, insert, update, upsert, delete
- ✓ where, join, having, group by, etc.

Full Text Search
- ✓ natural language inverted index query
- ✓ tokenization, stems, facets, relevancy

Eventing
- ✓ functions triggered by data events
- ✓ Diagnosably respond to DCP actions

Analytics (5.x preview)
- ✓ continuous online querying
- ✓ workload distributed and isolated



Counchbase shell

- cbq -u Administrator -p anjushan -engine http://127.0.0.1:8091/

- ```
  SELECT country FROM `travel-sample` WHERE name = "Excel
  Airways";
  ```

- ```
  SELECT callsign FROM `travel-sample` LIMIT 5;
  ```

- ```
  SELECT name FROM `travel-sample` WHERE callsign = "MILE-
  AIR";
  ```
  ```

- Query returns a maximum of one airport document, and lists all of the fields that it contains
- `SELECT * FROM `travel-sample` WHERE type="airport" LIMIT 1;`

- Query returns the names of (at a maximum) ten hotels that accept pets, in the city of Medway:

- cbq> SELECT name FROM `travel-sample` WHERE type="hotel" AND city="Medway" and pets_ok=true LIMIT 10;

- Query returns the name and phone fields for up to 10 documents for hotels in Manchester, where directions are not missing, and orders the results by name:

- cbq> SELECT name,phone FROM `travel-sample` WHERE type="hotel" AND city="Manchester" and directions IS NOT MISSING ORDER BY name LIMIT 10;