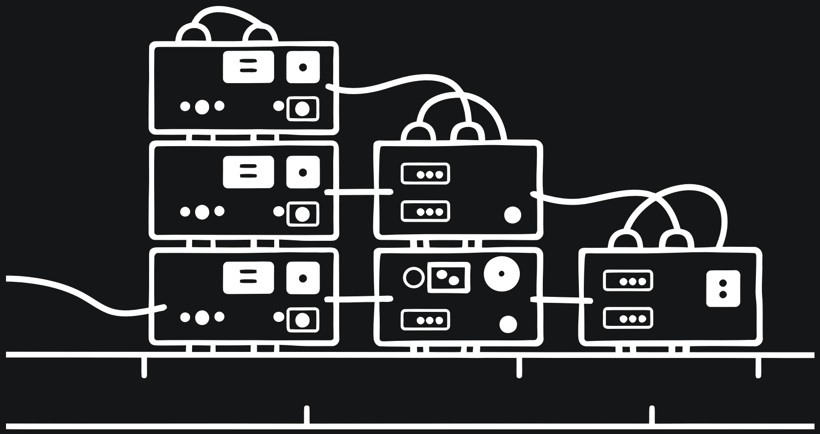
# Beyond Relational: An Introduction to NoSQL Databases

Exploring specialized solutions for modern web-based applications.



## Agenda

This presentation will provide a deep dive into various NoSQL database models, focusing on their architecture, data models, and practical applications. We'll cover:

1. **Introduction to NoSQL**

   Understanding the shift from relational to NoSQL paradigms.

2. **Document Databases**

   Focus on JSON-based DBMSs and their use cases.

3. **Column-Family Databases: BigTable**

   Delving into Google's foundational distributed storage system.

4. **BigTable Data Model & API**

   Structure, operations, and partitioning strategies.

5. **NoSQL Systems Comparison**

   Choosing the right NoSQL solution for your needs.

DATABASE SYSTEMS COURSE

# Understanding NoSQL: Not Only SQL

### Definition

NoSQL means "Not Only SQL," emphasizing a departure from traditional relational models for web-based applications.

### Core Philosophy

Choose the right tool for the right job, adapting to specific application needs rather than a one-size-fits-all approach.

### Key Trade-offs

Sacrifices full SQL support and strict ACID transactions for gains in simplicity, scalability, performance, and programming flexibility.

NoSQL databases offer specialized data models like key-value, document, graph, and columnar, moving beyond SQL as just a query language to explore entirely different data organization paradigms.

## Different Flavors of NoSQL: Diverse Approaches

NoSQL isn't a single technology but a category encompassing several different database types, each optimized for specific data models and use cases.

### Key-Value Stores

The simplest NoSQL databases. Imagine a giant dictionary where each word (key) has a single definition (value). Great for caching and simple data storage.

### Document Databases

Store data in flexible, semi-structured "documents" (like JSON or XML). Perfect for managing content, catalogs, or user profiles.

### Column-Family Stores

Organize data into rows and dynamic columns. Ideal for big data analytics and time-series data, like sensor readings or event logs.

### Graph Databases

Focus on relationships between data points (nodes and edges). Excellent for social networks, recommendation engines, and fraud detection.

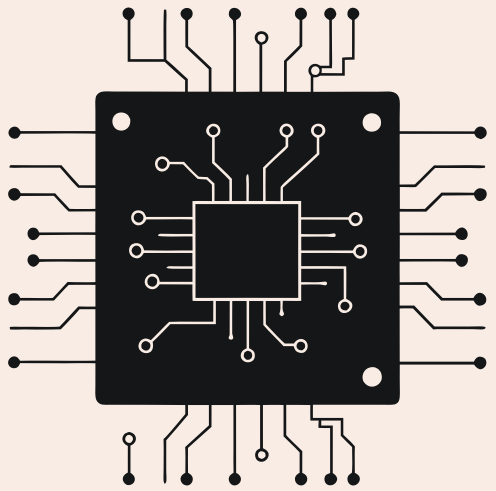
## Why NoSQL? The Paradigm Shift

### Traditional Relational Databases (SQL)

- Schema-rigid: predefined structure
- Vertical scaling: more powerful server
- ACID properties: Atomicity, Consistency, Isolation, Durability
- Best for complex queries and transactional systems

### NoSQL Databases

- Schema-less/flexible: dynamic structure
- Horizontal scaling: distributed across many servers
- BASE properties: Basically Available, Soft state, Eventually consistent
- Best for high-volume, agile, and unstructured data



The move to NoSQL addresses the challenges of big data, cloud computing, and agile development, where flexibility and massive scalability are paramount.

## Key-Value Stores: Simplicity at Scale

### How they work:

Think of a key-value store as a massive, distributed hash map. You store data by associating a unique "key" with a "value." To retrieve the data, you simply provide the key. It's like having a locker number (key) to retrieve your bag (value).

### Real-World Example: Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that supports key-value and document data models. It’s known for its incredibly fast performance at any scale. Many popular apps and websites use it for user sessions, leaderboards, and real-time bidding.

# NoSQL Approaches

## Key-Value Stores

The simplest model, mapping unique keys to values. Ideal for caching and session management. **Example:** Amazon DynamoDB.

## Tabular/Column Stores

Designed for flexible columns and efficient analytics on large datasets. **Example:** Google BigTable, Apache HBase.

## Document Stores

Stores data in flexible, semi-structured documents, often JSON-like. Great for content management. **Example:** MongoDB, CouchDB.

## Graph Databases

Optimized for highly interconnected data, representing nodes and relationships. Perfect for social networks. **Example:** Neo4j, Amazon Neptune.

## Multimodel Databases

Combines features from multiple NoSQL categories into a single system, offering versatility. **Example:** OrientDB, ArangoDB.

# Amazon DynamoDB: Introduction & Data Model

## Overview & Key Features

DynamoDB is a fully managed NoSQL database service by AWS, known for its serverless architecture and automatic scaling. It delivers consistent single-digit millisecond performance globally, with built-in security and backup features.

- Multi-region, multi-active database.
- Supports document and key-value models.
- Used by Netflix, Samsung, and Toyota.

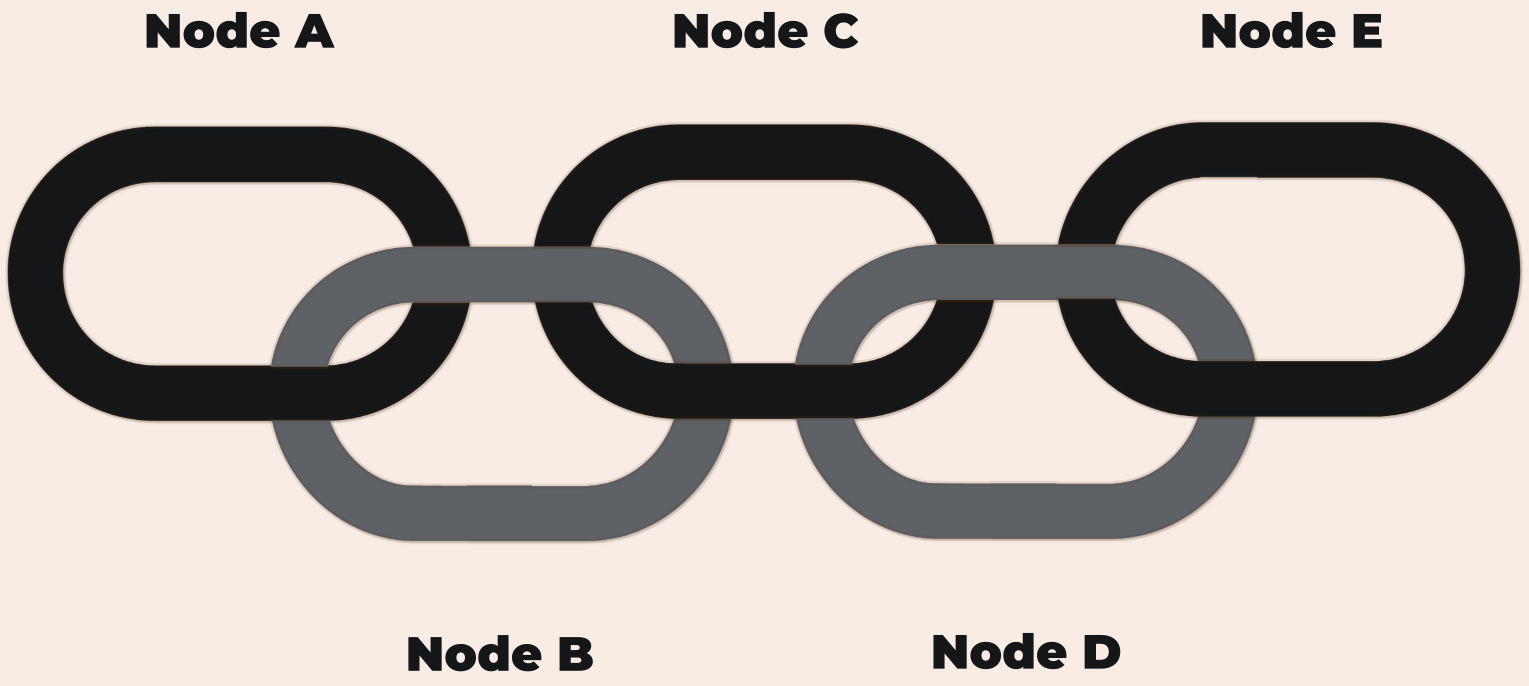
## Data Model Fundamentals

- **TABLE:** A collection of items.
- **ITEM:** A collection of attributes (like a record).
- **ATTRIBUTE:** A name-value pair (like a field).

Primary keys define how data is accessed:

- **PARTITION KEY (Hash Key):** A single attribute for unique identification (e.g., UserID).
- **COMPOSITE KEY (Hash + Range Key):** Two attributes for combined uniqueness (e.g., UserID + GameTitle).

# DynamoDB: Data Partitioning



DynamoDB uses a consistent hashing approach where hash values are arranged in a circular ring. Each node is responsible for a specific range of hash values, ensuring automatic load distribution.

- **Fault Tolerance:** If a node fails, its successor automatically takes over its range, ensuring continuous availability.
- **Seamless Scaling:** New nodes can be added or removed without impacting other nodes, as data rebalancing occurs automatically.
- **Replication:** Data items are typically replicated to a predefined number of successor nodes for high durability and availability.

# Document Data Model

## Characteristics

- **Hierarchical:** Data is stored with nested elements, allowing for complex structures.
- **Semi-structured:** Documents within the same collection can have varying structures, providing flexibility.
- **Rich Data Types:** Supports diverse types like text, numbers, dates, arrays, and objects.

## Two Main Formats

- **XML (eXtensible Markup Language):** A verbose, powerful W3C standard, often used for configurations and web services since 1998.
- **JSON (JavaScript Object Notation):** A simple, lightweight, human-readable format, ideal for web APIs and modern applications, created in 2005.

## JSON Example

```
{
  "name": "Alice",
  "age": 25,
  "courses": ["Math", "CS101"],
  "address": {
    "city": "Boston",
    "zip": "02101"
  }
}
```

# MongoDB: Collections & Query Language

## MongoDB Document Structure

MongoDB stores data in flexible **Documents**, which are grouped into **Collections** (analogous to tables in relational databases). Each document has a unique `_id` field, typically auto-generated.

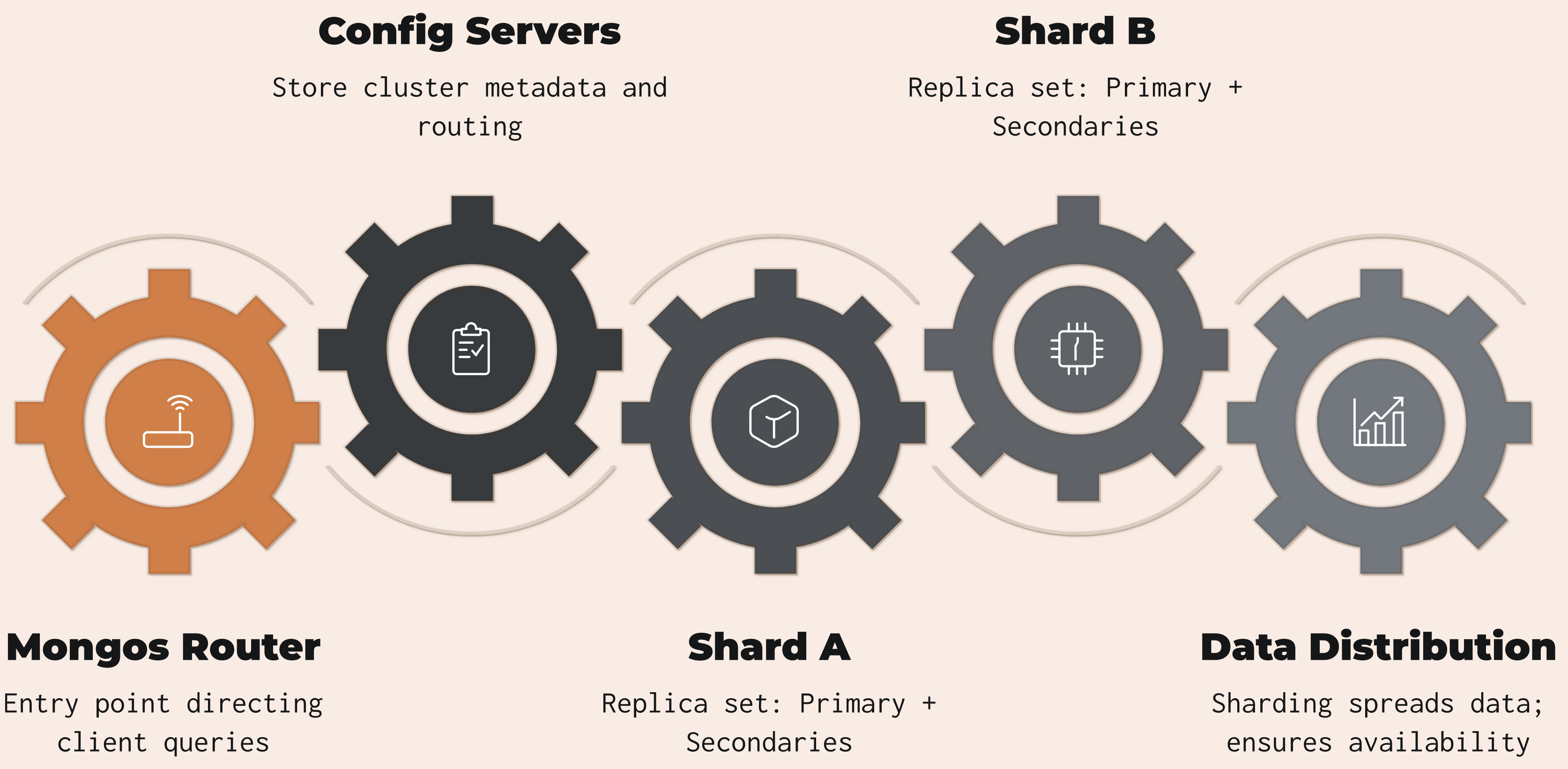
```
{
  "_id": ObjectId("65e6..."),
  "author": "alex",
  "title": "Learning NoSQL",
  "content": "NoSQL databases are fascinating...",
  "tags": ["database", "nosql", "mongodb"],
  "comments": [
    {"who": "jane", "text": "Great post!", "date": "2024-08-31"}
  ],
  "likes": 15
}
```

## Query Operations

MongoDB's query language is intuitive, allowing flexible data manipulation. Queries operate on collections, enabling powerful filtering based on document fields, including nested structures and arrays.

- **INSERT:** `db.posts.insertOne({...})`
- **UPDATE:** `db.posts.updateOne({"author": "alice"}, {"$set": {"age": 30}})`
- **QUERY Examples:**
  - `db.posts.find({"author": "alice"})`
  - `db.posts.find({"comments.who": "jane"})`
  - `db.posts.find({"tags": "database"})`
  - db.posts.find({"likes": {"$gt": 10}})

# MongoDB: Architecture


Config Servers
Store cluster metadata and routing
Shard B
Replica set: Primary + Secondaries
Mongos Router
Entry point directing client queries
Shard A
Replica set: Primary + Secondaries
Data Distribution
Sharding spreads data; ensures availability

MongoDB supports various deployment models, from simple standalone instances to highly scalable sharded clusters for massive datasets. The architecture leverages replica sets for high availability and sharding for horizontal scaling.

- **Mongos:** Acts as a query router and load balancer for sharded clusters.
- **Config Servers:** Store critical metadata about the cluster configuration.
- **Shards:** Hold subsets of the data, distributed across multiple servers.
- **Replica Sets:** Ensure data redundancy and automatic failover within each shard or for standalone deployments.

# Main NoSQL JSON Document DBMSs

Document databases store data in flexible, semi-structured documents, often in JSON or BSON format. This model is ideal for hierarchical data and rapid application development.

| | | | |
|---|---|---|---|
| Apache | CouchDB | JavaScript | Open source, offline-first sync |
| Couchbase Inc. | Couchbase | N1QL | Open source, high-performance distributed caching |
| LinkedIn | Espresso | JSON | Distributed document store (internal) |
| MarkLogic | MarkLogic Server | JSON | Enterprise-grade, multi-model, Hadoop integration |
| MongoDB Inc. | MongoDB | Extended JSON | Most popular document DB, general-purpose |

## Selection Criteria

- **CouchDB:** Offline-first mobile applications
- **MongoDB:** General-purpose document storage & web applications
- **Couchbase:** High-performance caching & real-time analytics
- **MarkLogic:** Enterprise content management & data integration

# BigTable: Introduction to a Column-Family Store

## What is BigTable?

- **Google's distributed storage system** for structured data.
- **Petabyte-scale data** across thousands of commodity servers.
- Built on **Google File System (GFS)**.

## Key Characteristics

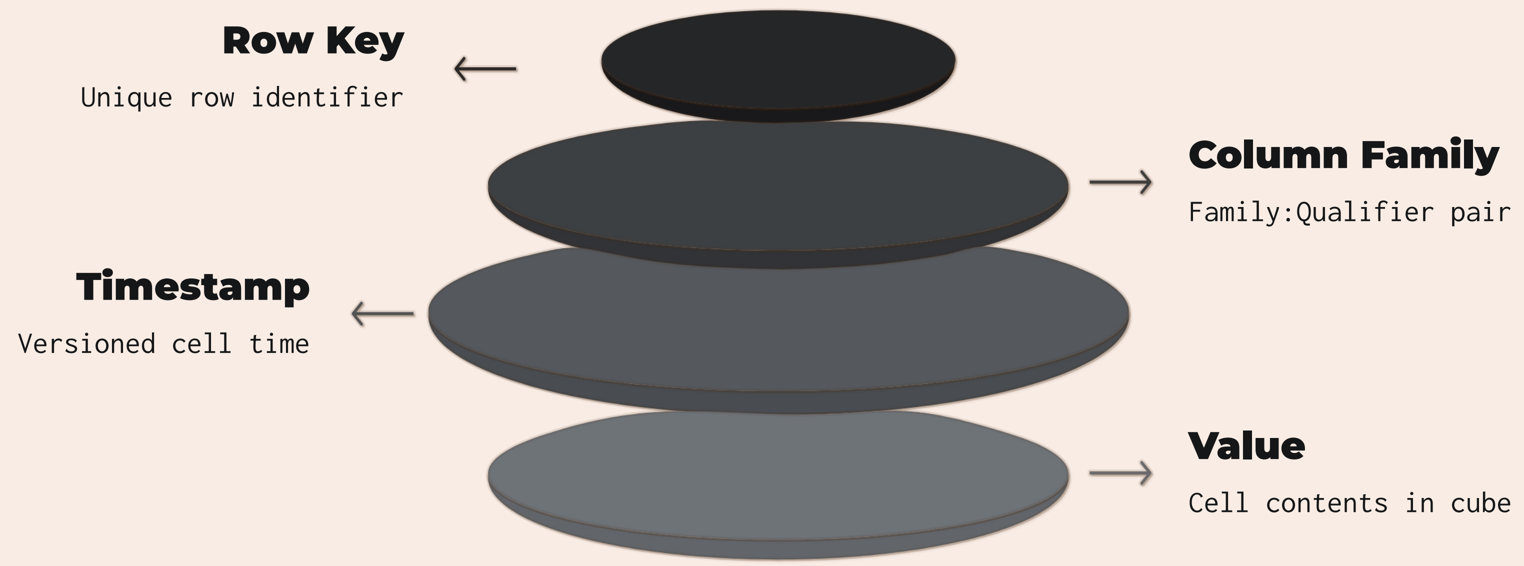
- Fault-tolerant and highly available.
- Combines aspects of **row-store** and **column-store** DBMS.
- Dynamic partitioning for scalability.
- Rows with multi-valued, timestamped attributes.

## Used by Google Applications

## Open Source Implementations

- **Apache HBase:** Runs on Hadoop HDFS.
- **Apache Cassandra:** Originally developed by Facebook.

# BigTable Data Model: (row, column, timestamp) → value



## Row

- Identified by row key (string, up to 64KB).
- Data sorted lexicographically by row key.
- Atomic operations guaranteed within a single row.

## Column Families

- Group of related columns (e.g., 'contents', 'anchor').
- Unit of access control and compression.
- Must be pre-created before data insertion.
- Column key format: `family:qualifier`

## Timestamps

- 64-bit integers (usually time in microseconds).
- Allows multiple versions of the same cell.
- Automatic garbage collection of old versions.

## Example: Web Page Storage

- **Row key:** "`com.google.www`"
- **Column family "contents":** page content
- **Column family "anchor":** anchor text from linking pages
- **Timestamp:** when page was crawled

# BigTable API and Operations

BigTable primarily uses a programmatic API, not a traditional SQL-like language, for data manipulation.

## Basic Operations

**Write Operations:**

- `Set()`: Write value to (row, column, timestamp).
- `DeleteCells()`: Delete specific cells.
- `DeleteRow()`: Delete entire row.

**Read Operations:**

- `Scanner`: Iterate over a subset of data.
- Various filters available:
  - Row range: `[start_row, end_row)`
  - Column family/qualifier filters
  - Timestamp range filters
  - Value filters

## Key Constraints

**Transactions:**

- Single-row transactions only.
- No multi-row transactions.
- Atomic read-modify-write operations for consistency within a row.

**No Complex Operations:**

- No joins between tables.
- No union operations.
- Application must handle complex logic at the client level.

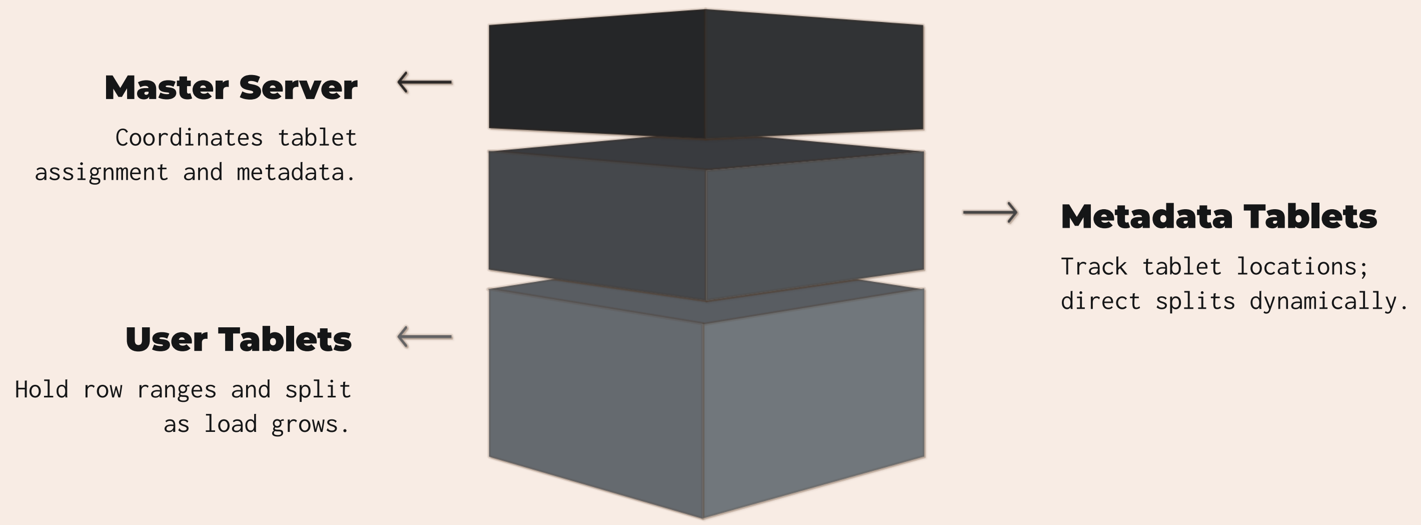
## Code Example (Conceptual Python)

```python
table = bigtable_client.Table('my_table')
row = table.row('row_key_123')
row.set_cell('cf1:col1', 'value_data', timestamp=datetime.now())
row.commit()

scanner = table.scan(row_start='row_key_100', row_end='row_key_200')
for row_key, row_data in scanner:
print(f"Row: {row_key}, Data: {row_data}")
```

# BigTable Dynamic Partitioning: Tablets

BigTable automatically manages data distribution and load balancing through 'tablets', which are contiguous ranges of rows.


Master Server
Coordinates tablet assignment and metadata.
Metadata Tablets
Track tablet locations; direct splits dynamically.
User Tablets
Hold row ranges and split as load grows.

## Tablet Properties

- **Tablets:** Table is divided into tablets (row ranges).
- Initially one tablet per table.
- Automatic splitting as table grows (typical size: 100-200 MB).

## Three-Level Hierarchy

- **1. Master Server:** Stores root tablet location (single point, but lightweight).
- **2. Metadata Tablets:** Store locations of user tablets (each references ~128MB of user tablets).
- **3. User Tablets:** Actual application data, distributed across tablet servers.

## Implementation Optimizations

- Column family compression for storage efficiency.
- Locality groups co-locate related column families for faster reads.
- Aggressive metadata caching by clients to reduce lookup latency.
- Bloom filters for efficient non-existent key lookups.

# Summary: NoSQL Systems Comparison

Choosing the right NoSQL database depends on your specific application requirements, data model, and scalability needs.

## Key-Value (e.g., Redis, DynamoDB)

- Simple key-based access patterns.
- High performance requirements.
- Best for caching, session management, user profiles.

## Document (e.g., MongoDB, CouchDB)

- Complex, nested data structures.
- Rapid application development.
- Ideal for content management, catalogs, user-generated content.

## Column-Family (e.g., BigTable, HBase, Cassandra)

- Large-scale analytics and time-series data.
- Massive scalability needs for sparse data sets.
- Suitable for IoT data, log processing, web analytics.

## Graph (e.g., Neo4j, Amazon Neptune)

- Highly connected data, relationships are key.
- Fraud detection, social networks, recommendation engines.

## Key Takeaways & Next Steps

### Recap

- **NoSQL diversity:** A range of models for different use cases.
- **Scalability:** Designed for horizontal scaling and high availability.
- **Flexibility:** Schema-less designs adapt to evolving data.
- **BigTable as a foundation:** Understanding its architecture reveals distributed system principles.

### Further Exploration

- **Hands-on practice:** Set up a local instance of MongoDB or HBase.
- **Explore APIs:** Work through tutorials for data manipulation.
- **Dive into consistency models:** Understand CAP theorem implications.
- **Case studies:** Research how major tech companies use NoSQL.

# Beyond NoSQL: The Rise of NewSQL

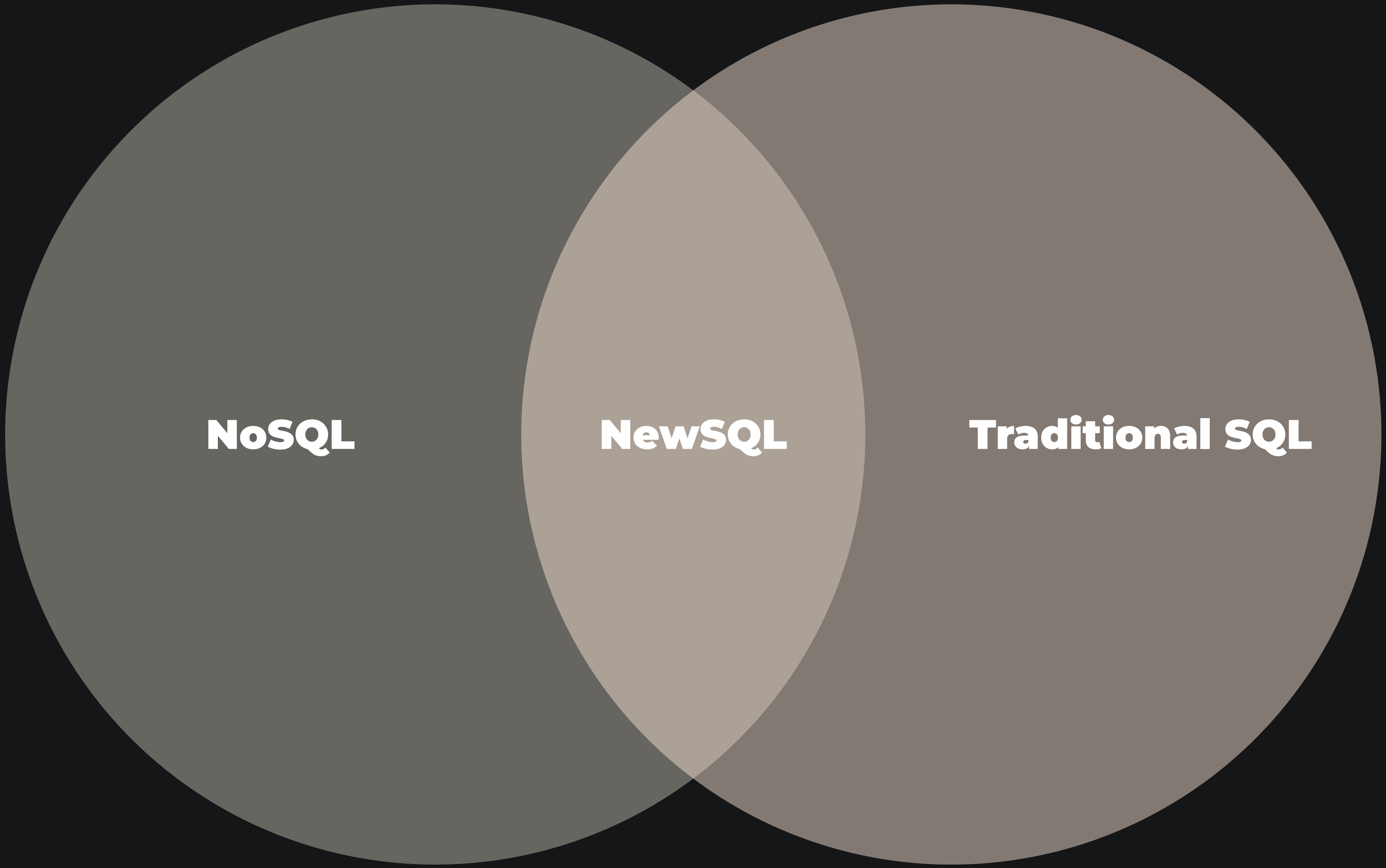
While NoSQL solved problems with scalability and flexibility, some applications still needed the strong consistency and transactional guarantees of traditional SQL. This led to the emergence of "NewSQL."

> ## What is NewSQL?
>
> NewSQL databases are relational database systems that combine the ACID (Atomicity, Consistency, Isolation, Durability) guarantees of traditional SQL databases with the scalability of NoSQL systems. They offer the best of both worlds.

### Example: Google F1

Google F1, used internally by Google for its AdWords platform, is a prime example of a NewSQL database. It provides a globally-distributed, fault-tolerant relational database that scales to petabytes of data while maintaining strong consistency and high availability.

## NewSQL: Best of Both Worlds



NewSQL systems emerge as a hybrid approach, combining the horizontal scalability and performance of NoSQL with the ACID transaction guarantees and familiar SQL interface of traditional relational databases.

## Distributed Architecture & Transaction Processing

Both NoSQL and NewSQL often rely on distributed architectures to achieve scalability.

### Distributed Architecture

This means data is spread across multiple servers or nodes, allowing databases to handle more users and larger amounts of data than a single machine ever could. It's like having many workers collaborate on a single large project.

### Transaction Processing

Transactions ensure that database operations are reliable. In SQL and NewSQL, this often means ACID compliance. NoSQL databases might offer different consistency models (e.g., eventual consistency) for higher availability and performance.

## Main NewSQL Systems: Bridging the Gap

Here are some notable NewSQL database systems aiming to provide both SQL familiarity and distributed scalability:

| | | |
|---|---|---|
| CockroachDB | Geo-distributed, highly scalable, PostgreSQL compatible | High availability, global applications |
| TiDB | MySQL compatible, distributed SQL, HTAP capabilities | Hybrid Transactional/Analytical Processing |
| VoltDB | In-memory, high-speed transactions, real-time data | Telecommunications, financial services |
| Google Spanner / F1 | Globally distributed, strong consistency, high availability | Internal Google services, highly critical applications |

## Which Data Store For What? Choosing the Right Tool

The choice between SQL, NoSQL, or NewSQL depends entirely on your specific needs. There's no one-size-fits-all solution!

| | | |
|---|---|---|
| SQL (Relational) | Strong consistency, complex queries, data integrity | Financial transactions, traditional business apps |
| NoSQL | Scalability, flexibility, handling unstructured data | Big data, real-time web apps, content management |
| NewSQL | ACID transactions, SQL interface, horizontal scalability | Globally distributed services, highly consistent scaled apps |

Understanding these different database paradigms empowers you to make informed decisions for building robust and scalable applications.

## Key Takeaways & Next Steps

### NoSQL Diversity

Understand the specialized data models (key-value, document, column, graph) and their unique use cases.

### Fit-for-Purpose

Always choose the database technology that best fits the specific requirements of your application.

### NewSQL Synthesis

Recognize NewSQL as a solution bridging the gap between NoSQL scalability and traditional SQL consistency.

### Next Steps:

- Explore hands-on labs with DynamoDB, MongoDB, or CockroachDB.
- Research real-world case studies of companies using these database systems.
- Deep dive into distributed consensus algorithms like Paxos and Raft.