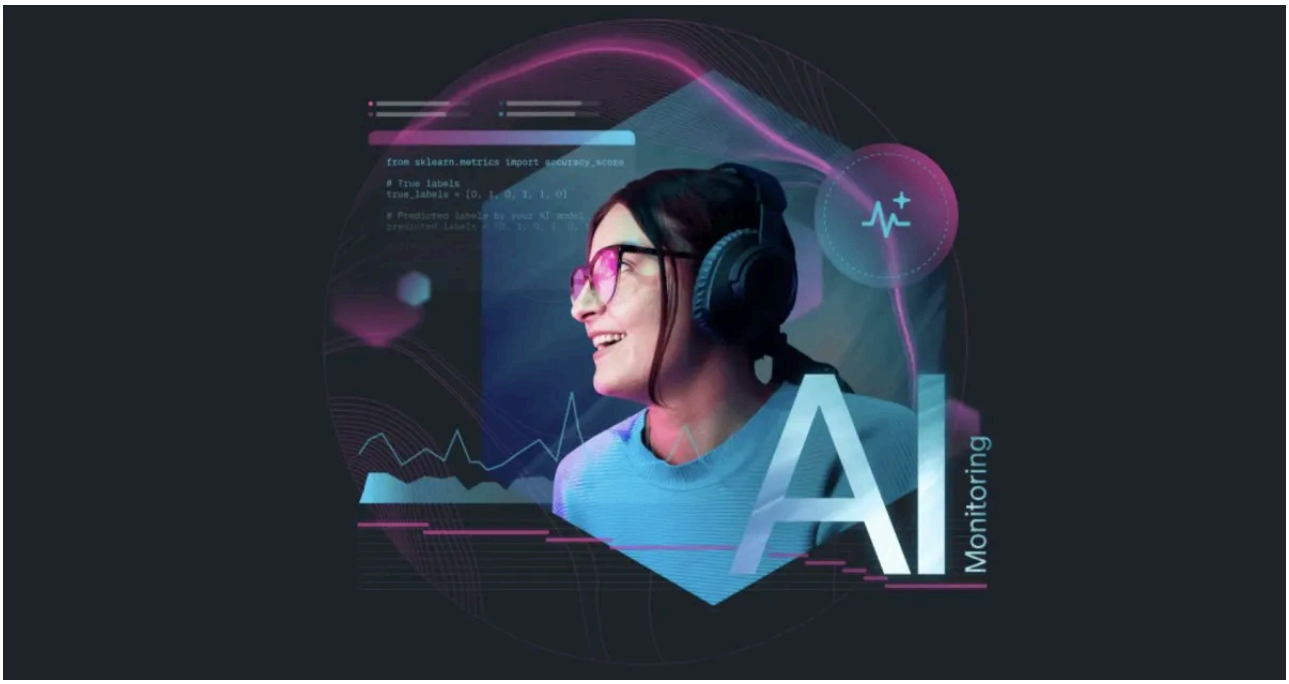
EP110: Top 5 Strategies to Reduce Latency

BYTEBYTEGO
MAY 04, 2024

This week's system design refresher:

- Top 5 Strategies to Reduce Latency
- Load Balancer Realistic Use Cases You May Not Know
- Top 4 data sharding algorithms explained
- Top 8 C++ Use Cases
- Apache Kafka in 100 Seconds
- SPONSOR US

New Relic AI monitoring, the industry's first APM for AI, now generally available (Sponsored)



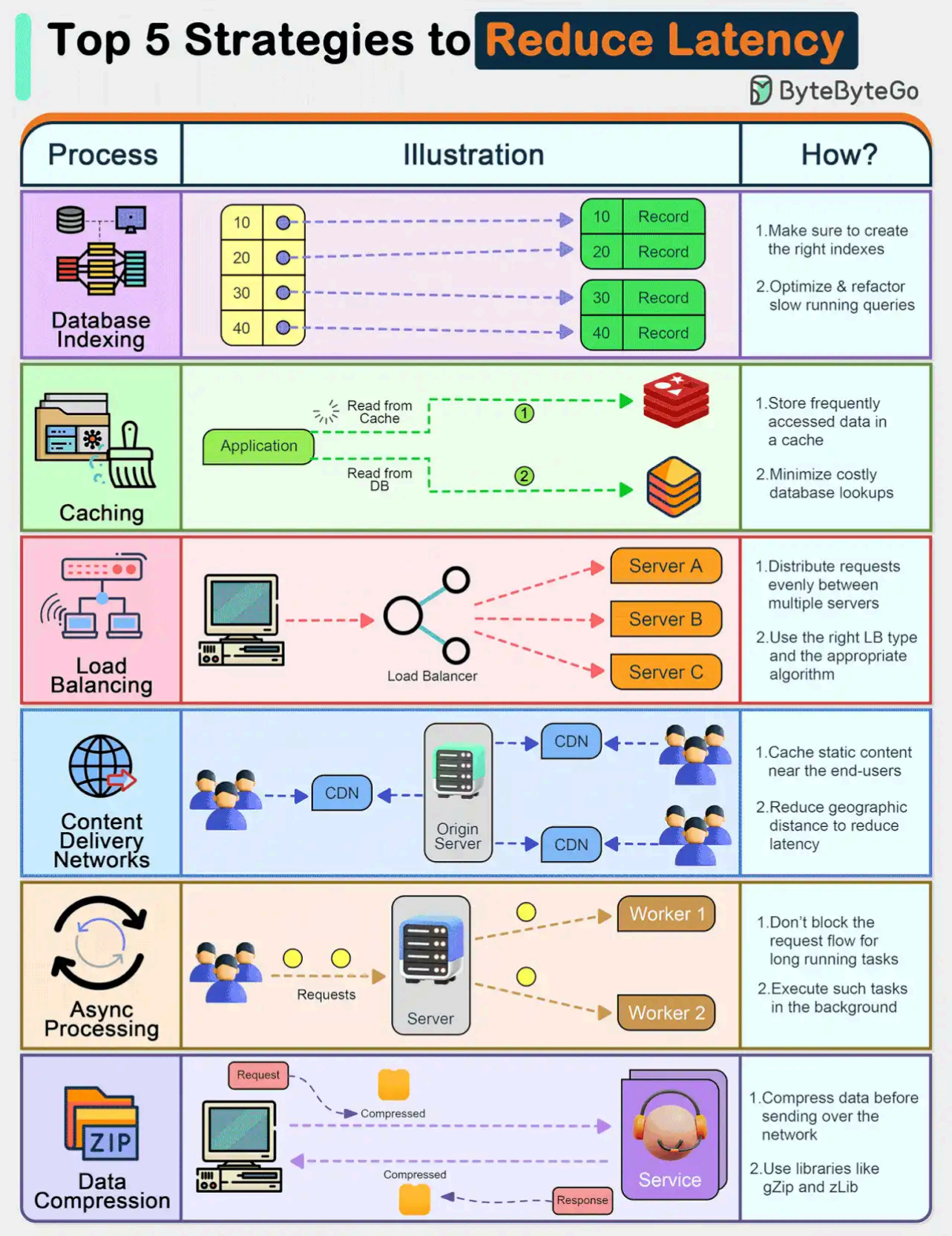
New Relic AI monitoring provides unprecedented visibility and insights to engineers and developers who are modernizing their tech stacks. With New Relic AI monitoring, engineering teams can monitor, alert, debug, and root-cause AI-powered applications.

Top 5 Strategies to Reduce Latency

10 years ago, Amazon found that every 100ms of latency cost them 1% in sales.

That's a staggering $5.7 billion in today's terms.

For high-scale user-facing systems, high latency is a big loss of revenue.



Here are the top strategies to reduce latency:

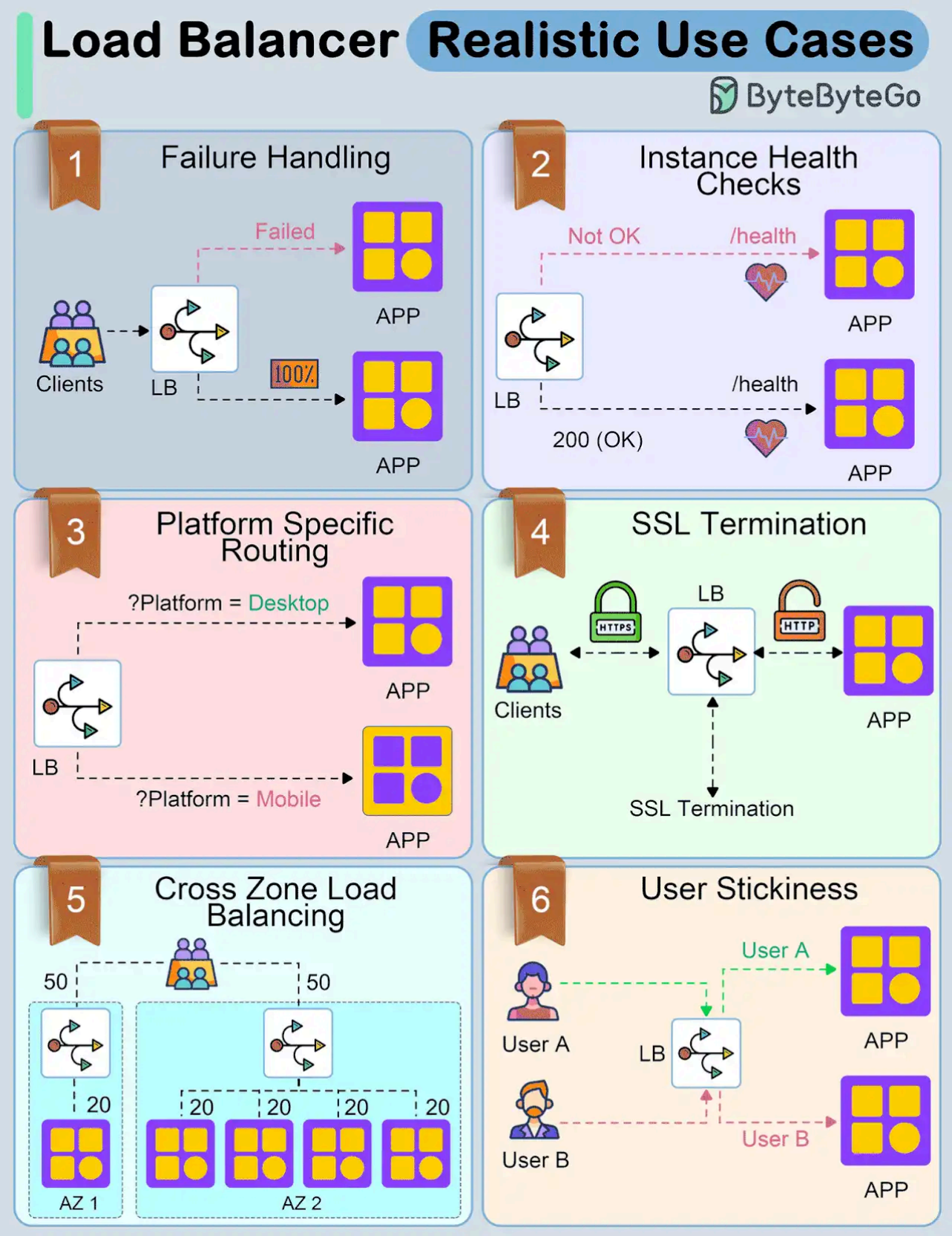
1. Database Indexing
2. Caching

3. Load Balancing
4. Content Delivery Network
5. Async Processing
6. Data Compression

Over to you: What other strategies to reduce latency have you seen?

Load Balancer Realistic Use Cases You May Not Know

Load balancers are inherently dynamic and adaptable, designed to efficiently address multiple purposes and use cases in network traffic and server workload management.



Let's explore some of the use cases:

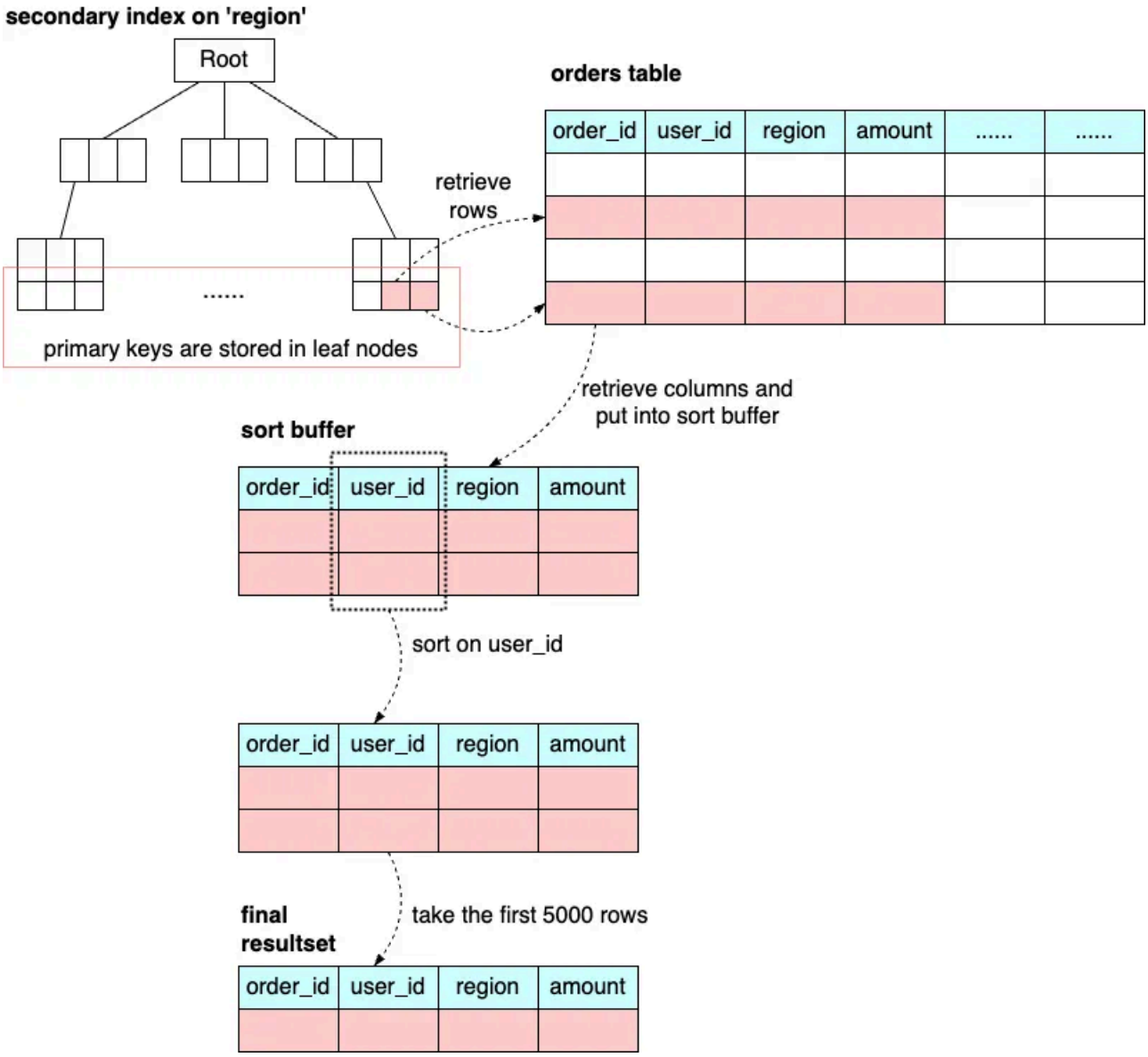
1. Failure Handling:
   Automatically redirects traffic away from malfunctioning elements to maintain continuous service and reduce service interruptions.

2. Instance Health Checks:
   Continuously evaluates the functionality of instances, directing incoming requests exclusively to those that are fully operational and efficient.
3. Platform Specific Routing:
   Routes requests from different device types (like mobiles, desktops) to specialized backend systems, providing customized responses based on platform.
4. SSL Termination:
   Handles the encryption and decryption of SSL traffic, reducing the processing burden on backend infrastructure.
5. Cross Zone Load Balancing:
   Distributes incoming traffic across various geographic or network zones, increasing the system's resilience and capacity for handling large volumes of requests.
6. User Stickiness:
   Maintains user session integrity and tailored user interactions by consistently directing requests from specific users to designated backend servers.

Over to you:
Which of these use cases would you consider adding to your network to enhance system reliability and why?

Latest articles

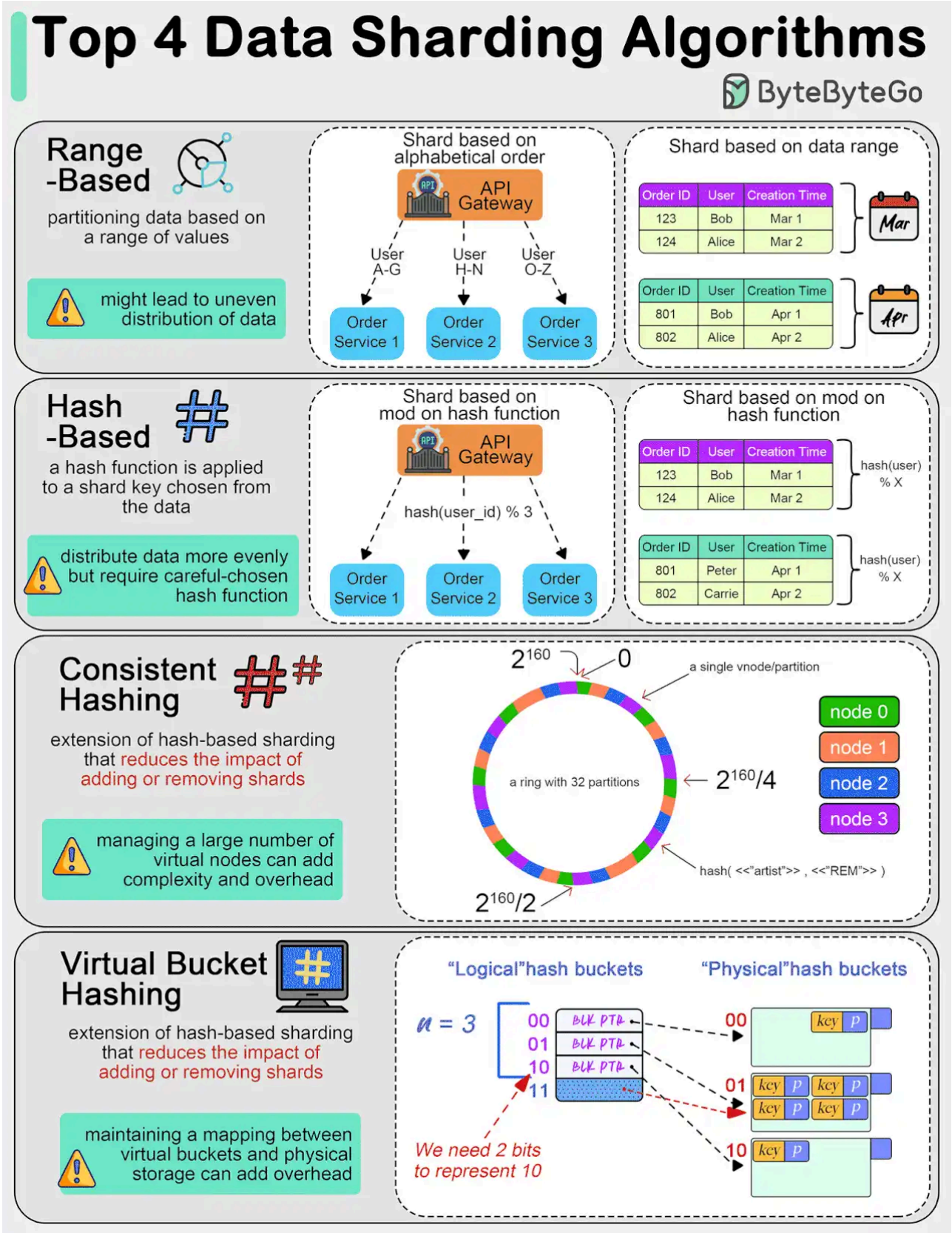
If you’re not a paid subscriber, here’s what you missed.



1. Unlocking the Power of SQL Queries for Improved Performance
2. What Happens When a SQL is Executed?
3. A Crash Course in API Versioning Strategies
4. Embracing Chaos to Improve System Resilience: Chaos Engineering
5. A Crash Course in CI/CD

To receive all the full articles and support ByteByteGo, consider subscribing:

Top 4 Data Sharding Algorithms Explained

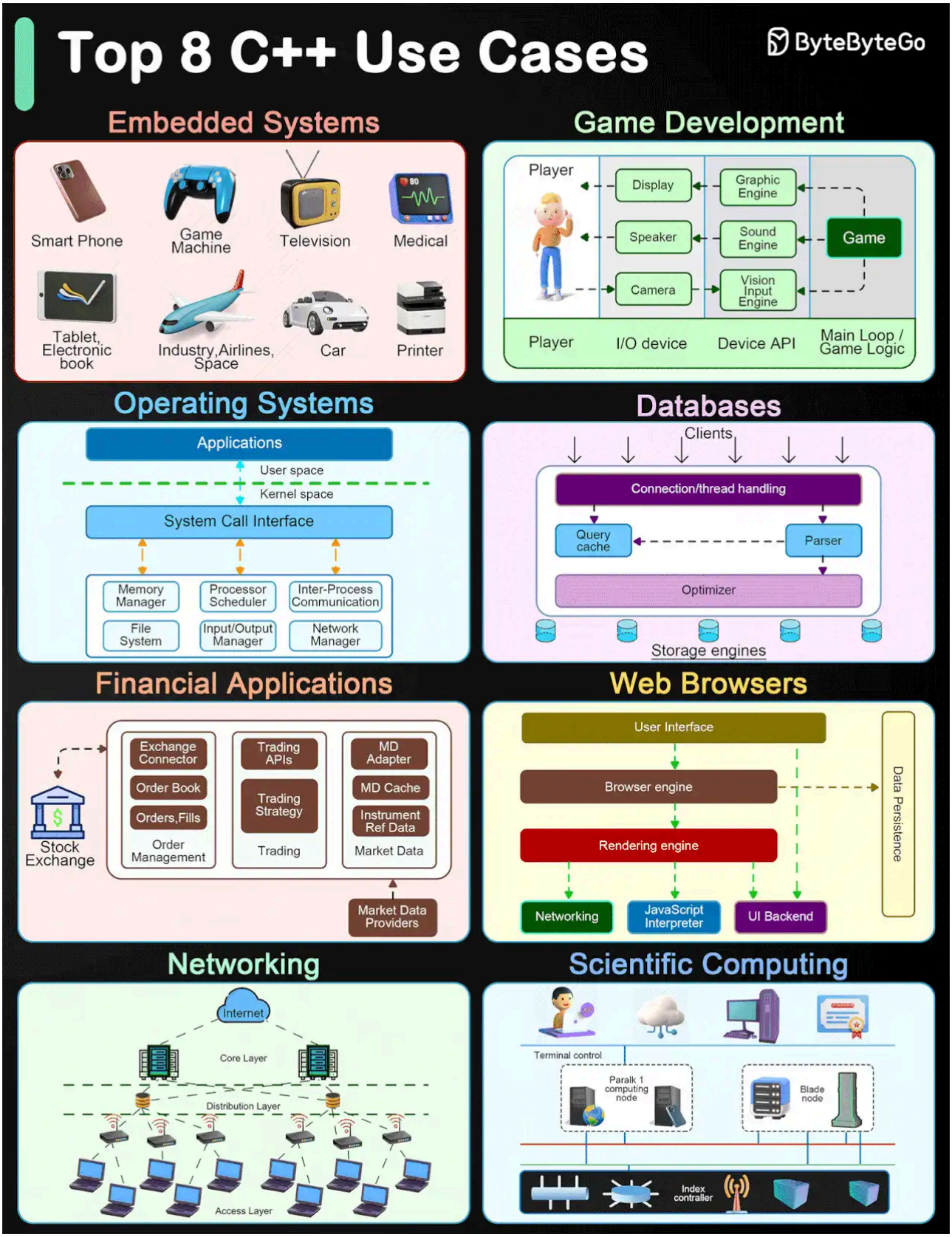


We are dealing with massive amounts of data. Often we need to split data into smaller, more manageable pieces, or “shards”. Here are some of the top data sharding algorithms commonly used:

- Range-Based Sharding
  This involves partitioning data based on a range of values. For example, customer data can be sharded based on alphabetical order of last names, or transaction data can be sharded based on date ranges.
- Hash-Based Sharding
  In this method, a hash function is applied to a shard key chosen from the data (like a customer ID or transaction ID).
  This tends to distribute data more evenly across shards compared to range-based sharding. However, we need to choose a proper hash function to avoid hash collision.
- Consistent Hashing
  This is an extension of hash-based sharding that reduces the impact of adding or removing shards. It distributes data more evenly and minimizes the amount of data that needs to be relocated when shards are added or removed.
- Virtual Bucket Sharding
  Data is mapped into virtual buckets, and these buckets are then mapped to physical shards. This two-level mapping allows for more flexible shard management and rebalancing without significant data movement.

Top 8 C++ Use Cases

C++ is a highly versatile programming language that is suitable for a wide range of applications.



- Embedded Systems
  The language's efficiency and fine control over hardware resources make it excellent for embedded systems development.

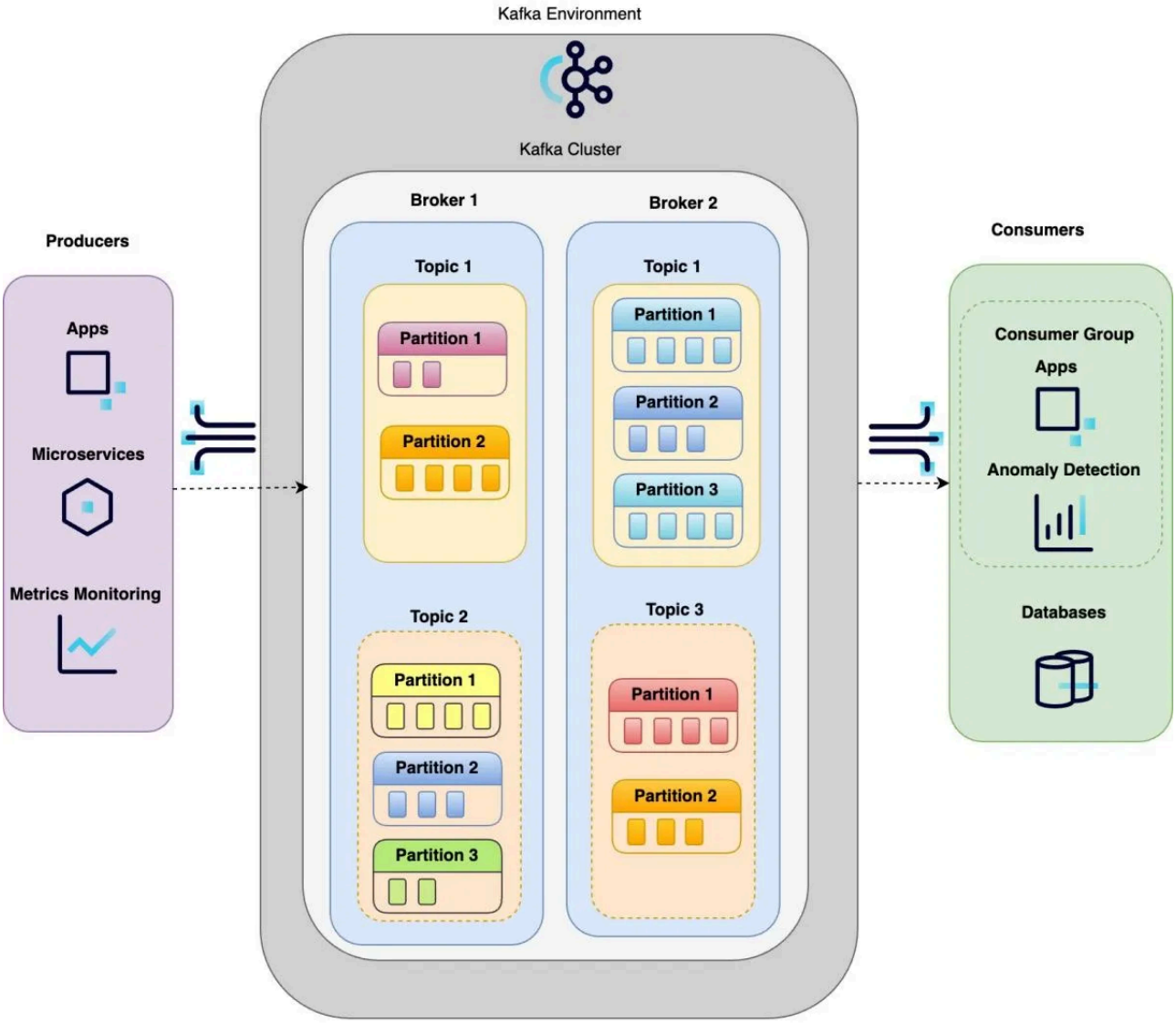
- Game Development
  C++ is a staple in the game development industry due to its performance and efficiency.
- Operating Systems
  C++ provides extensive control over system resources and memory, making it ideal for developing operating systems and low-level system utilities.
- Databases
  Many high-performance database systems are implemented in C++ to manage memory efficiently and ensure fast execution of queries.
- Financial Applications
- Web Browsers
  C++ is used in the development of web browsers and their components, such as rendering engines.
- Networking
  C++ is often used for developing network devices and simulation tools.
- Scientific Computing
  C++ finds extensive use in scientific computing and engineering applications that require high performance and precise control over computational resources.

Over to you - What did we miss?

Apache Kafka in 100 Seconds

This post is written by guest author Sanaz Zakeri, who is a Senior Software Engineer @Uber.

Apache Kafka is a distributed event streaming platform used for building real-time data processing pipelines and streaming applications. It is highly scalable, fault-tolerant, reliable, and can handle large volumes of data.



In order to understand Kafka, we need to define two terms:

- Events: a log of state of something at a specific point in time
- Event streams: continuous and unbounded series of events

Kafka can be used as a Messaging in a publish-subscribe model, where producers write event streams, and consumers read the events. This publish-subscribe model enables decoupling of event stream producers and consumers. Also, Kafka can be used as a log aggregation platform, ingesting and storing logs from multiple sources in a durable and fault-tolerant way.

- **Kafka Components:**
  Kafka cluster has multiple key components to provide the distributed infrastructure and reliably capture, store, order and provide event streams to client applications.

- **Brokers:**
  At the heart of the Kafka cluster lies the brokers which are physical servers that handle event streams. After events are published by producers, the broker makes the events available to consumers. Brokers bring scalability to Kafka as Kafka clusters can span multiple brokers across a variety of infrastructure setup to handle large volumes of events. They also bring fault tolerance since events can be stored and replicated across multiple brokers.
- **Topics:**
  Topic is the subject name of where the events are published by producers. Topics can have zero or more consumers listening to them and processing the events.
- **Partition:**
  In a topic, data is organized into partitions which store ordered streams of events. Each event within a partition is assigned a unique sequential identifier called offset that represents its position in the partition. Events are appended continually to the partition. A Topics can have one or more partitions. Having more than one partition in a topic enables parallelism as more consumers can read from the topic.
  Partitions belonging to a topic can be distributed across separate brokers in the cluster, which brings high data availability and scalability. If one broker fails, the partitions on the remaining brokers can continue to serve data, ensuring fault tolerance.
- **Producers:**
  Producers are client applications that write events to Kafka topics as a stream of events.
- **Consumers:**
  Consumers are the client applications that subscribe to topics and process or store the events coming to the specific topic. Consumers read events in the order they were received within each partition.
  Applications which require real time processing of data will have multiple consumers in a consumer group which can read from partitions on the subscribed topic.
- **Consumer Groups:**
  Consumer group is used to organize consumers that are reading a stream of events from one or more topics. Consumer groups enable parallel processing of

events and each consumer in the consumer group can read from one partition to enable load balancing on the client application. This functionality not only brings the parallel processing but also brings fault tolerance since if a consumer fails in a consumer group, Partition can be reassigned to another group member.

SPONSOR US

Get your product in front of more than 500,000 tech professionals.

Our newsletter puts your products and services directly in front of an audience that matters - hundreds of thousands of engineering leaders and senior engineers - who have influence over significant tech decisions and big purchases.

Space Fills Up Fast - Reserve Today

Ad spots typically sell out about 4 weeks in advance. To ensure your ad reaches this influential audience, reserve your space now by emailing hi@bytebytego.com.

149 Likes · 12 Restacks

3 Comments

Write a comment...

Robin Bowes May 4

Before optimising anything you should instrument your app/system and understand the bottlenecks.

LIKE (6) REPLY SHARE

Aman May 6

Loved the refresher on these topics, beautifully organized and written

LIKE (2) REPLY SHARE

1 more comment...