8 Must-Know Strategies to Build Scalable Systems

Don't ignore scalability

SAURABH DASHORA
DEC 10, 2024

What do a $500 billion e-commerce platform, a global ride-hailing service, and the most popular streaming service on the planet have in common?

It’s the ability to build scalable systems.

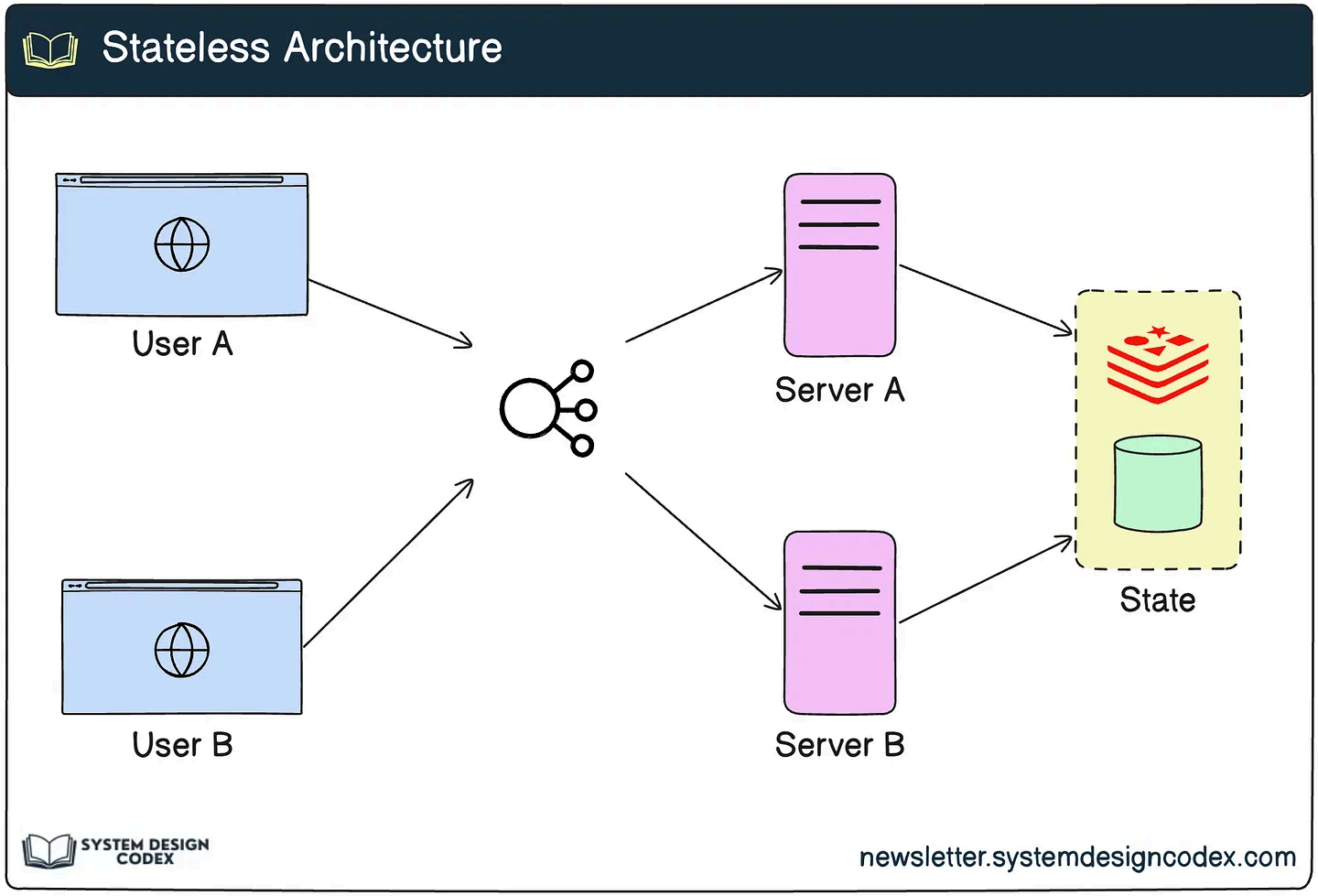
> **This ability to scale ensures your system can handle increased loads without sacrificing performance or user experience. Sure, not every system needs the scale of Amazon, Uber, or Netflix. But scaling strategies can still help.**

Here are eight must-know strategies to build scalable systems:

1 - Stateless Services

A stateless service is one that doesn't retain information about client sessions between requests. Each request contains all the information necessary for the server to process it.

Stateless architecture makes scaling much easier because it allows servers to be interchangeable and reduces the complexity of managing the state.



You can play around with the diagram on Eraser.io

Why It Matters:

- **Ease of Scaling**: Stateless services can easily be duplicated across multiple servers.

- **Fault Tolerance**: If a server fails, requests can be redirected to another server without losing session data.

Implementation Tips:

- Use tokens like JSON Web Tokens (JWT) to store session data on the client side instead of the server.
- For operations requiring state (e.g., shopping cart sessions), consider externalizing state management to a database or caching layer like Redis.

2 - Horizontal Scaling

Horizontal scaling, or "scale-out," involves adding more servers to share the load. Unlike vertical scaling, which involves upgrading hardware, horizontal scaling is more cost-effective and provides better fault tolerance.

Why It Matters:

- **Redundancy**: Multiple servers reduce the impact of single points of failure.
- **Scalability**: You can handle larger workloads by simply adding more servers.

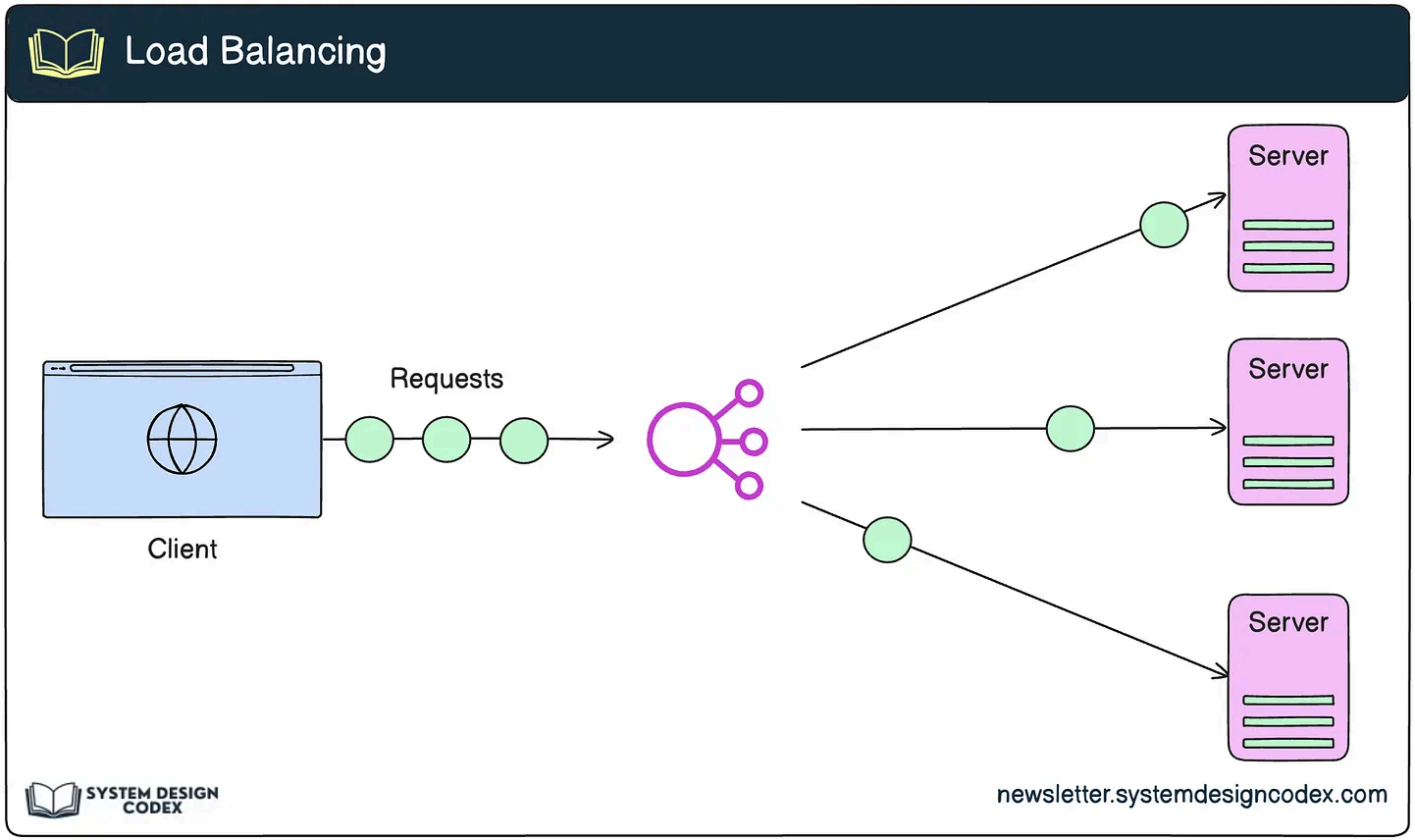
Implementation Tips:

- Ensure your system supports distributed workloads. Tools like Kubernetes can help manage containerized applications across multiple nodes.

- Use stateless services to simplify horizontal scaling, as each server can independently handle requests.

3 - Load Balancing

Load balancing involves distributing incoming requests evenly across multiple servers. A load balancer acts as a middleman, ensuring that no single server is overwhelmed.



You can play around with the diagram on Eraser.io

Why It Matters:

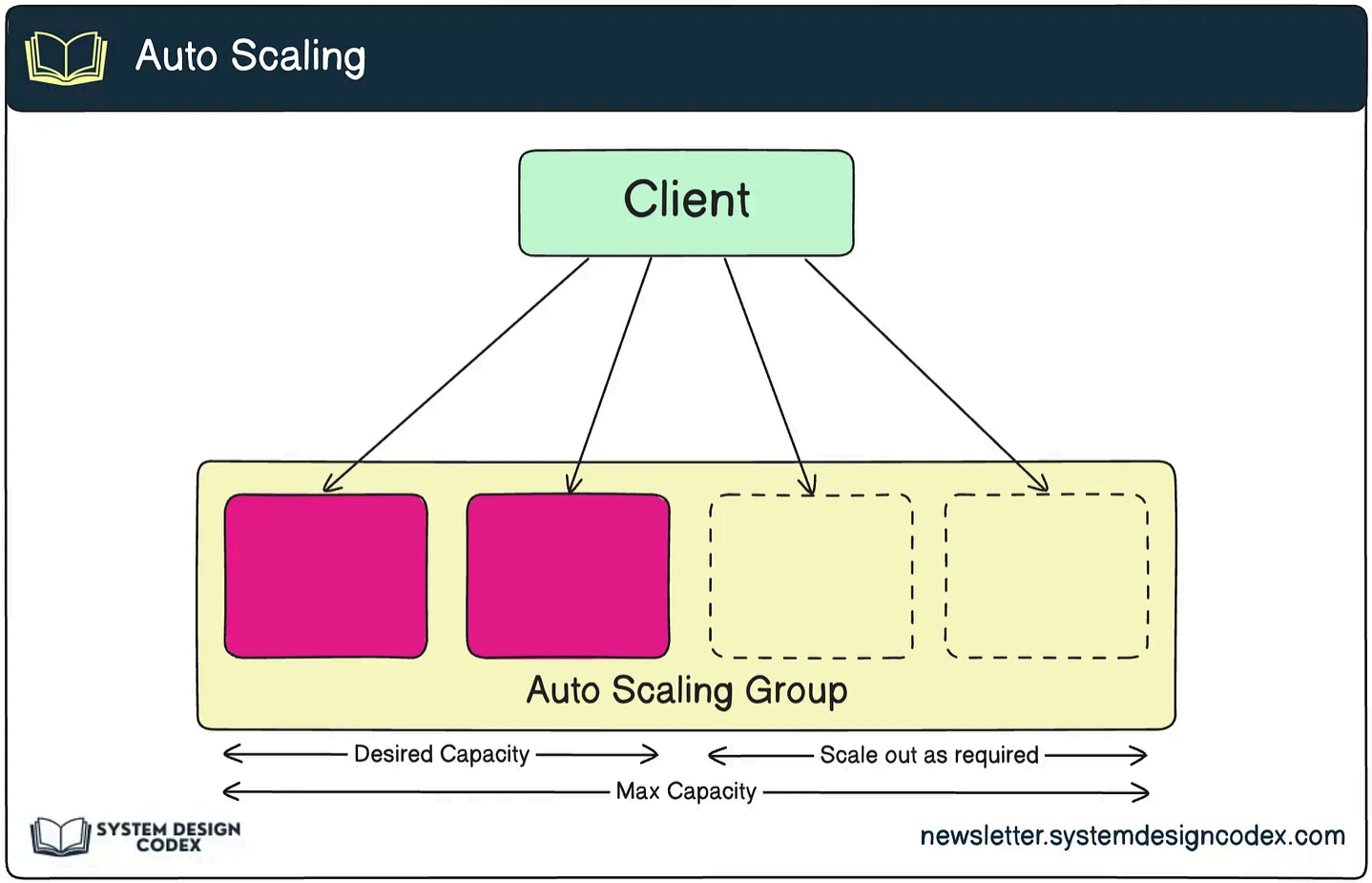
- **Performance**: Prevents overload by spreading the traffic evenly.
- **High Availability**: Automatically redirects traffic to healthy servers in case of failure.

Implementation Tips:

- Use hardware or software load balancers like NGINX, HAProxy, or AWS Elastic Load Balancer.
- Implement health checks to ensure that the load balancer only sends requests to functioning servers.
- Configure sticky sessions cautiously, as they can introduce statefulness and reduce flexibility.

4 - Auto Scaling

Auto-scaling dynamically adjusts the number of servers or resources based on real-time traffic. It ensures you're not over-provisioning during off-peak hours or under-provisioning during traffic spikes.



You can play around with the diagram on Eraser.io

Why It Matters:

- **Cost Efficiency**: Automatically scales down resources when demand decreases, saving costs.

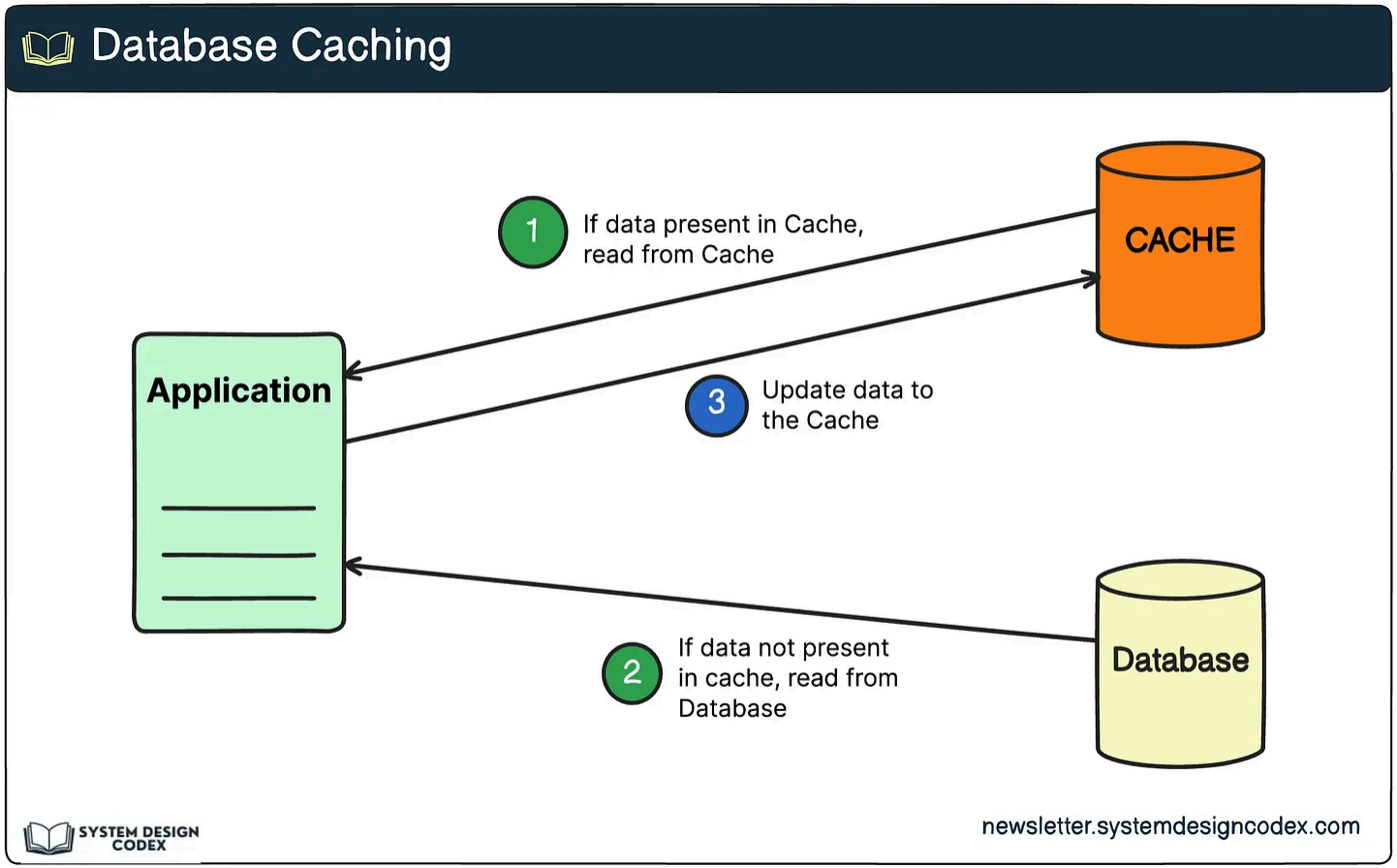
- **Traffic Management**: Handles unexpected surges in traffic without manual intervention.

Implementation Tips:

- Use cloud providers like AWS, Azure, or Google Cloud, which offer built-in auto-scaling tools.
- Set clear thresholds for scaling up and down, based on CPU usage, memory, or request latency.

5 - Caching

Caching involves storing frequently accessed data closer to the user or in-memory systems to reduce the load on the database. Proper caching can significantly improve system performance and scalability.



You can play around with the diagram on Eraser.io

Why It Matters:

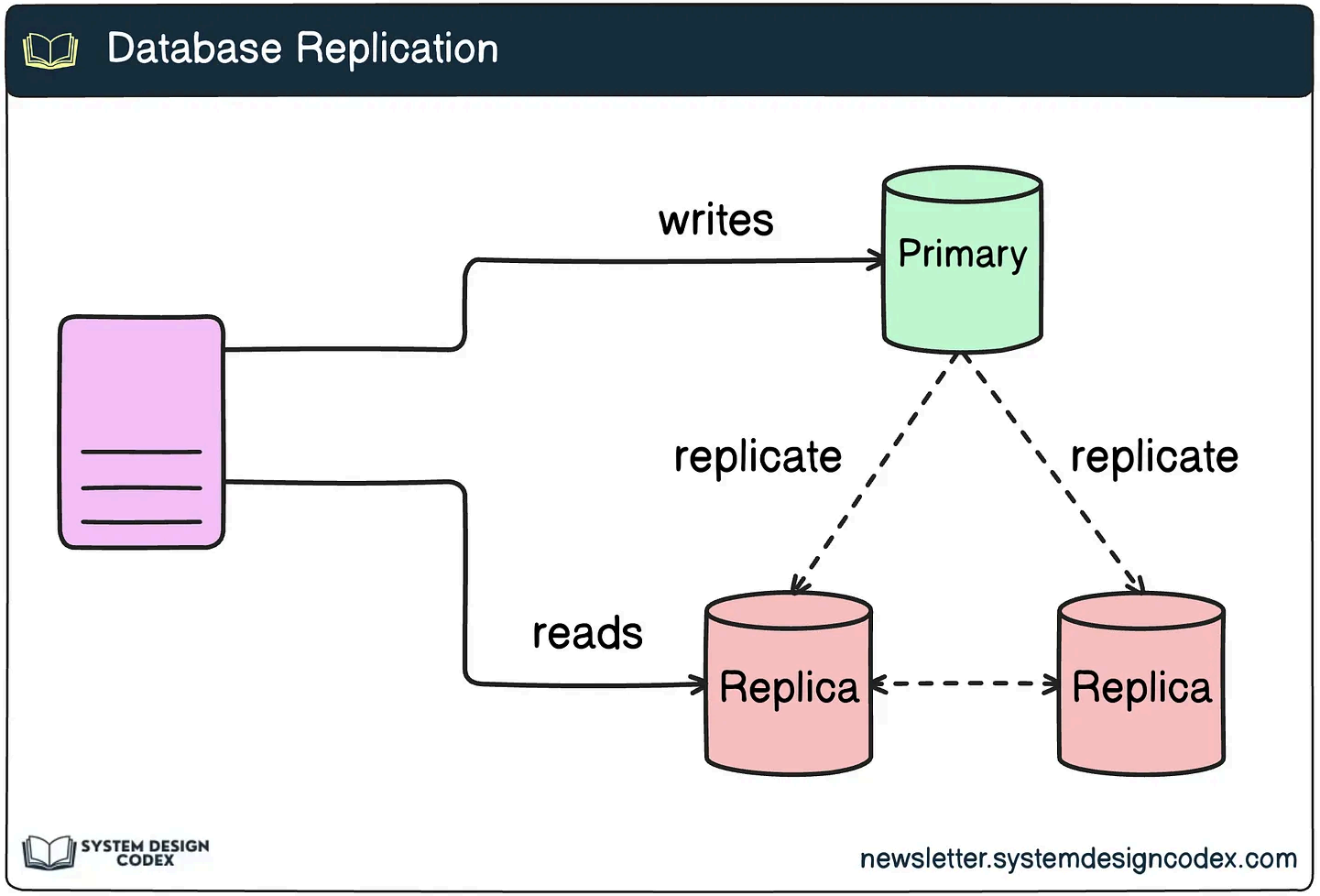
- **Reduced Database Load**: By serving repetitive queries from the cache, you reduce the workload on your database.
- **Improved Speed**: Cached data can be retrieved much faster than fetching it from the database.

Implementation Tips:

- Use caching tools like Redis, Memcached, or Varnish.
- Implement different caching layers:
  - **Database Caching**: Cache query results to avoid recalculating them.
  - **Application Caching**: Store data in memory for quick access.
  - **Content Delivery Networks** (**CDNs**): Cache static assets like images and scripts close to the user.
- Set appropriate expiration times to ensure cached data remains up-to-date.

6 - Database Replication

Database replication involves creating multiple copies of your database across different nodes. These replicas can handle read requests, improving both performance and redundancy.



You can play around with the diagram on Eraser.io

Why It Matters:

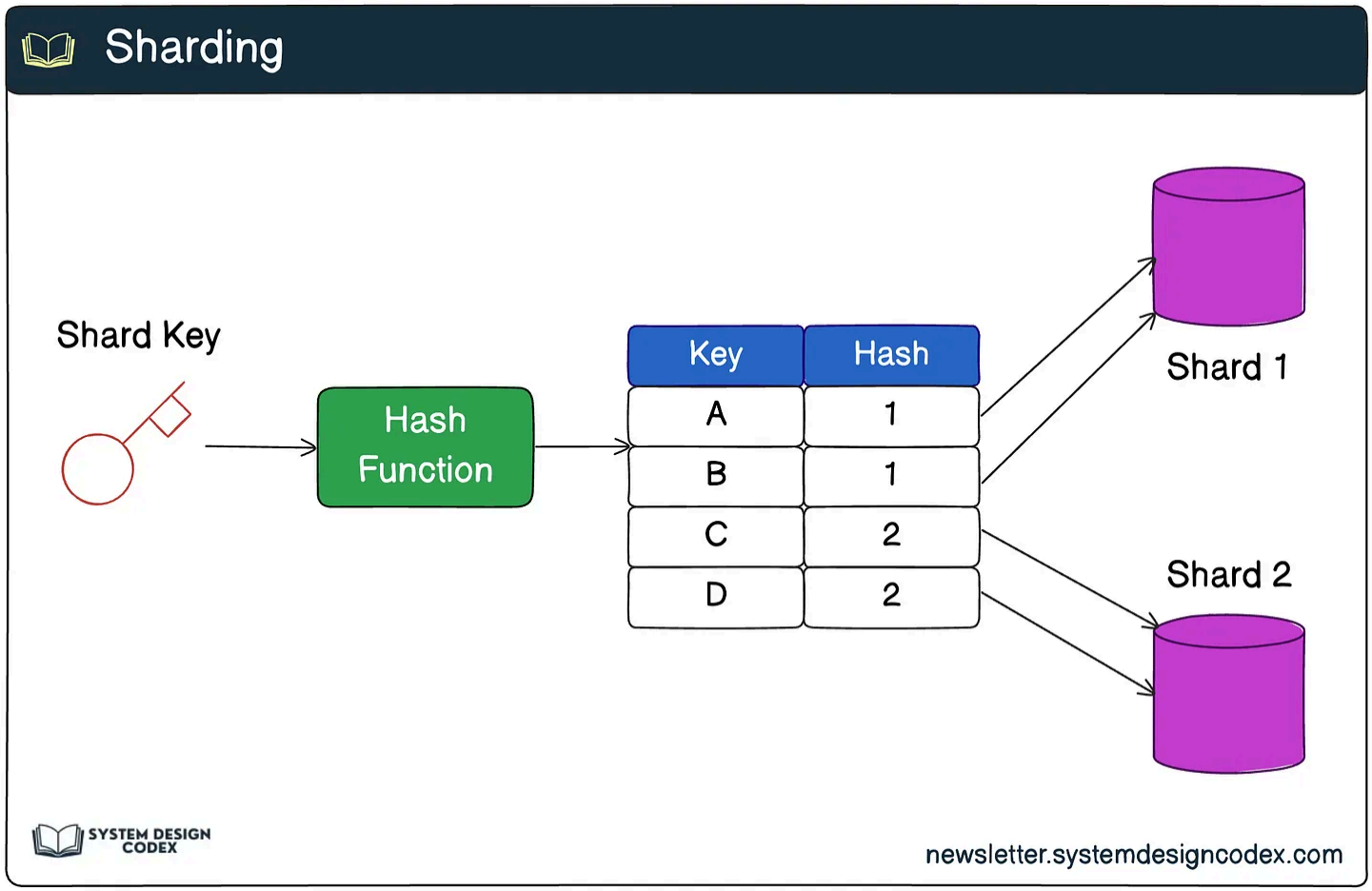
- **Scalable Reads**: Multiple replicas can handle read-heavy workloads without affecting the primary database.
- **Fault Tolerance**: Provides backup nodes in case the primary database fails.

Implementation Tips:

- Use database systems like PostgreSQL, MySQL, or MongoDB, which support replication.
- Use asynchronous replication for better performance, but be aware of eventual consistency issues.
- Design your application to distinguish between read and write queries, sending reads to replicas and writes to the primary node.

7 - Database Sharding

Sharding is the process of dividing your database into smaller, more manageable pieces called shards. Each shard contains a subset of the data and operates independently.



You can play around with the diagram on Eraser.io

Why It Matters:

- **Scalable Writes**: By distributing data, you reduce contention and improve write performance.

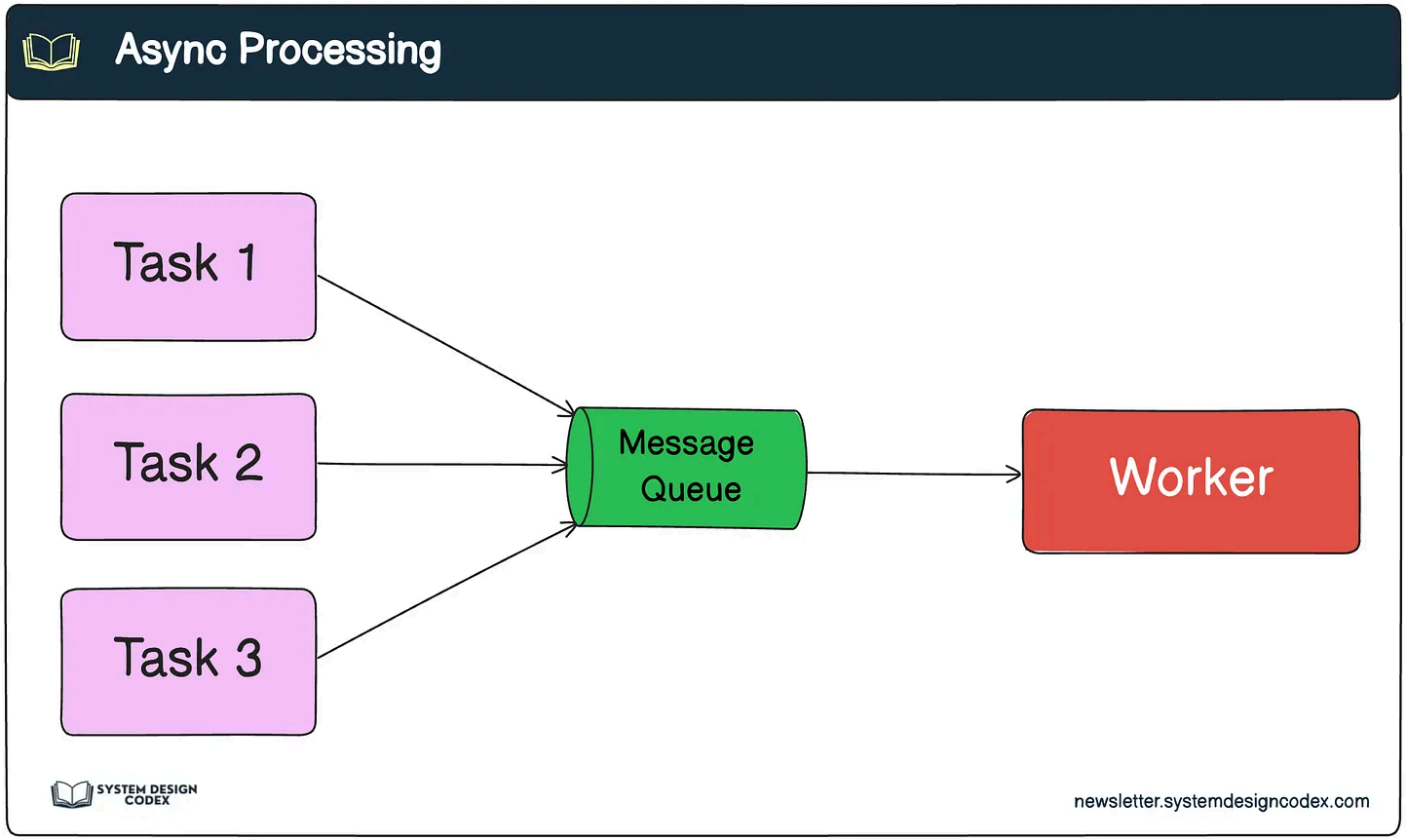
- **High Availability**: Shards can be distributed across different regions for better fault tolerance.

Implementation Tips:

- Use consistent hashing or range-based sharding to distribute data efficiently.
- Plan your sharding strategy carefully, as it can be challenging to re-shard data later.
- Monitor shard performance and ensure balanced distribution to avoid hotspots.

8 - Async Processing

Asynchronous processing moves resource-intensive tasks, such as sending emails or generating reports, to background workers. This allows the system to handle new requests without waiting for the completion of these tasks.



You can play around with the diagram on Eraser.io

Why It Matters:

- **Improved Responsiveness**: Users don't have to wait for tasks to complete before receiving a response.
- **Scalability**: Background workers can scale independently to handle task queues.

Implementation Tips:

- Use message queues like RabbitMQ, Kafka, or AWS SQS to manage task queues.
- Implement retry mechanisms to handle failures in background tasks.
- Prioritize idempotency for tasks to ensure they can be retried safely without duplication.

So - which other strategy will you add to the list?

Shoutout

Here are some interesting articles I've read recently:

- Focus on Building Resilient Interactions, Not Just Resilient Services by Raul Junco
- What is the real cost of not having tests (at all) by Akos
- Writing as a software engineer by Franco Fernando
- The Reliability Factor: Building Trust in Your Professional Journey by Riccardo Causo

---

That's it for today! 🌼

Enjoyed this issue of the newsletter?

Share with your friends and colleagues.

See you later with another edition — Saurabh

79 Likes · 7 Restacks

Discussion about this post

Comments Restacks

Write a comment...

Akos Komuves Dec 19

Liked by Saurabh Dashora

Thanks for the shoutout, Saurabh! There should be a 9th strategy: identifying when you need scaling and writing faster backend code so you don't have to do it super early. Why? I've just encountered an API endpoint that returns some data. The data is gathered with a for loop containing a SELECT query. 😅

LIKE (1) REPLY SHARE

1 reply by Saurabh Dashora

Petar Ivanov Dec 15

Liked by Saurabh Dashora

These are 8 ways to make your application more robust and improve the user-experience.

Great article, Saurabh! 🙌

LIKE (1) REPLY SHARE

1 reply by Saurabh Dashora

12 more comments...