# SCALABILITY AND PERFORMANCE

## Scalability:

### What Is Scalability?

Scalability is frequently used as a magic incantation to indicate that something is badly designed or broken. Often you hear in a discussion “but that doesn’t scale” as the magical word to end an argument. This is often an indication that developers are running into situations where the architecture of their system limits their ability to grow their service. If scalability is used in a positive sense it is in general to indicate a desired property as in “our platform needs good scalability”.

As a system grows, the performance starts to **degrade** unless we adapt it to deal with that growth.

Scalability is the property of a system to handle a growing amount of load by **adding resources** to the system.

**A system that can continuously evolve to support a growing amount of work is scalable.**

A system is considered **scalable** if adding more resources (like servers or computing power) leads to a proportional improvement in its performance. Performance improvement usually means handling more tasks or requests, but it can also mean managing larger tasks, like processing bigger datasets.

In distributed systems, we might also add resources for other reasons, such as increasing the system's reliability. For example, adding redundancy (extra copies or backups) is a common way to prevent failures. A system is still considered scalable if adding redundancy doesn’t reduce its performance while keeping the service always available.

## Why is scalability so challenging?

Scalability is hard Because **scalability cannot be an after-thought**.—it has to be built into the design from the start. For a system to scale, adding more resources must lead to better performance. Similarly, adding redundancy (extra backups or fail-safes) should not harm the system's efficiency.

Many algorithms that work well with small datasets or light traffic can become extremely costly when traffic increases, datasets grow, or more nodes are added to a distributed system.

Another challenge is **heterogeneity**, which happens as the system grows. Newer generations of hardware may be faster, store more data, or cost less, and some resources might be located farther away. This diversity means that some parts of the system perform better than others. Algorithms designed for uniform resources might break or fail to take full advantage of the newer, more powerful components.

## Is it possible to achieve good scalability?

Yes, it is absolutely possible, but only if we design and build systems with scalability in mind from the start. This means:

- Identifying the ways in which the system is likely to grow (e.g., more users, larger datasets).
- Planning where redundancy is needed to ensure reliability without sacrificing performance.
- Understanding and managing heterogeneity—how different parts of the system might vary in speed, capacity, or location.

It's also crucial that architects and engineers know the right tools to use for specific situations and are aware of common challenges or pitfalls when scaling systems. Thoughtful planning and design make good scalability achievable.

# Performance

## What do we mean by performance?

Performance refers to how efficiently a system completes its tasks or serves its users. In simple terms, it's about how fast and reliably the system works under a given load. A system with good performance responds quickly, handles many tasks simultaneously, and provides a smooth user experience.

Performance can be measured in several ways, including:

- **Response Time:** How long it takes for the system to complete a task or respond to a request.
- **Throughput:** The number of tasks or requests the system can handle in a given amount of time.
- **Resource Utilization:** How effectively the system uses its resources like CPU, memory, or network.

Performance often depends on:

1. **Workload:** The amount of work the system needs to handle.
2. **Hardware:** The quality and capacity of the underlying resources (servers, storage, etc.).
3. **Design and Optimization:** How well the system is built and whether algorithms, code, and processes are efficient.

## Why is performance important?

A system with poor performance can:

- Frustrate users with slow responses.
- Fail to handle large workloads, leading to crashes or downtime.
- Waste resources, increasing operational costs.

## Performance vs. Scalability

### How Do I know If I have a performance problem?

**- If your system is slow for single user.**

### How Do I know If I have a scalability problem?

**- If your system is fast for a single user but slow under the load.**

While performance measures how well a system works under its current load, scalability is about how well it can handle growth. A system may perform well now but fail to scale if it wasn't designed for larger workloads or more users.

To achieve both good performance and scalability, careful planning, efficient design, and regular monitoring are essential.

# Latency

## What is Latency?

**Latency is a metrics that measure the performance of a computer network.**

Latency in networking refers to the time delay or lag that exists between a request and the response to that request, in simple words it is the time taken by a single data packet to travel from the source computer to the destination computer.

Networks with a longer delay or lag have high latency, while those with fast response times have lower latency.

# Throughput:

## What is Throughout?

Throughput refers to the average volume of data that can actually pass through the network over a specific time. It indicates the number of data packets that arrive at their destinations successfully and the data packet loss.

## Why are throughput and latency important?

You can determine network speed by looking at how quickly a network can transfer data packets to their destinations. This speed is the result of network performance factors like latency and throughput.

Latency determines the delay that a user experiences when they send or receive data from the network. Throughput determines the number of users that can access the network at the same time.

A network with low throughput and high latency struggles to send and process high data volume, which results in congestion and poor application performance. In contrast, a network with high throughput and low latency is responsive and efficient. Users experience improved performance and increased satisfaction.

High-performing networks directly impact revenue generation and operational efficiency. In addition, certain use cases—like real-time streaming, Internet of Things (IoT) data analytics, and high-performance computing—require certain network performance thresholds to operate optimally.

## Key differences: network latency vs. throughput

Although latency and throughput both contribute to a reliable and fast network, they are not the same. These network metrics focus on distinct statistics and are different from each other.

## How to measure

You can measure network latency by measuring ping time. This process is where you transmit a small data packet and receive confirmation that it arrived.

Most operating systems support a ping command which does this from your device. The round-trip-time (RTT) displays in milliseconds and gives you an idea of how long it takes for your network to transfer data.

You can measure throughput either with network testing tools or manually. If you wanted to test throughput manually, you would send a file and divide the file size by the time it takes to arrive. However, latency and bandwidth impact throughput. Because of this, many people use network testing tools, as the tools report throughput alongside other factors like bandwidth and latency.

## Impacting factors: latency vs. throughput

Different factors can impact your latency and throughput metrics.

### 1. Latency

Latency has several factors that contribute to it being high or low.

### 2. Location

One of the most important factors is the location of where data originates and its intended destination. If your servers are in a different geographical region from your device, the data has to travel further, which increases latency. This factor is called propagation..

### 3. Network congestion

Network congestion occurs when there is a high volume of data being transmitted over a network. The increased traffic on the network causes packets to take longer routes to their destination.

### 4. Protocol efficiency

Some networks require additional protocols for security. The extra handshake steps create a delay.

### 5. Network infrastructure

Network devices can become overloaded, which results in dropped packets. As packets are delayed or dropped, devices re transmit them. This adds additional latency.

### 6. Throughput

Throughput speeds are directly impacted by other factors.

### 7. Bandwidth

If your network capacity has reached the maximum bandwidth of your transmission medium, its throughput will never be able to go beyond that limit.

### 8. Processing power

Certain network devices have specialized hardware or software optimizations that improve their processing performance. Some examples are dedicated application-specific integrated circuits or software-based packet processing engines.

These optimizations enable the device to handle higher volumes of traffic and more complex packet processing tasks, which leads to higher throughput.

### 9. Packet loss

Packet loss can occur for a variety of reasons, including network congestion, faulty hardware, or misconfigured network devices. When packets are lost, they must be re-transmitted. This results in delays and reduces the overall throughput of the network.

### 10. Network topology

Network topology refers to the number of network devices, the bandwidth of the network links, and the distance between devices in a network path.

A well-designed network topology provides multiple paths for data transmission, reduces traffic bottlenecks, and increases throughput. Networks with more devices or longer distances require complex network topologies to achieve high throughput.

## Relationship between bandwidth, latency and throughput

Latency and throughput work together to deliver high network connectivity and performance. As both impact the transmission of data packets, they also affect one another.

If a network connection has high latency, it can have lower throughput, as data takes longer to transmit and arrive. Low throughput also makes it seem like a network has high latency, as it takes longer for large quantities of data to arrive.

As they are closely linked, you must monitor both latency and throughput to achieve high network performance.

### Bandwidth and network throughput

Bandwidth represents the total volume of data that you can transfer over a network. Your total bandwidth refers to the theoretical maximum amount of data that you could transfer over a network. You measure it in megabytes per second (MBps). You can think of bandwidth as the theoretical maximum throughput of your network.

Bandwidth is how much data you can transfer, while throughput is the actual amount of data you transmit in any given moment based on real-world network limitations. A high bandwidth does not guarantee speed or a good network performance, but a higher bandwidth leads to higher throughput.

## How can you improve latency and throughput?

To improve latency, you can shorten the propagation between the source and destination. You can improve throughput by increasing the overall network bandwidth.

Next, we give some suggestions to improve latency and throughput together.

### Caching

Caching in networking refers to the process of storing frequently accessed data geographically closer to the user. For example, you can store data in proxy servers or content delivery networks (CDNs).

Your network can deliver data from the cached location much faster than if it had to be retrieved from the original source. And the user receives data much faster, improving latency. Additionally, because the data is retrieved from a cache, it reduces the load on the original source. This allows it to handle more requests at once, improving throughput.

## Transport protocols

By optimizing the transport protocol that you use for specific applications, you can improve network performance.

For instance, TCP and UDP are two common network protocols. TCP establishes a connection and checks that you receive data without any errors. Because of its goal of reducing packet loss, TCP has higher latency and higher throughput. UDP does not check for packet loss or errors, transmitting several duplicate packets instead. So, it gives minimal latency but a higher throughput.

Depending on the application that you are using, TCP or UDP may be the better choice. For example, TCP is useful for transferring data, while UDP is useful for video streaming and gaming.

## Quality of service

You can use a quality of service (QoS) strategy to manage and optimize network performance. QoS allows you to divide network traffic into specific categories. You can assign each category a priority level.

Your QoS configurations prioritize latency-sensitive applications. Some applications and users experience lower latency than others. Your QoS configurations can also prioritize data by type, reducing packet loss and increasing throughput for certain users

### Summary of differences: throughput vs. latency

| | Throughput | Latency |
|---|---|---|
| What does it measure? | Throughput measures the volume of data that passes through a network in a given period. Throughput impacts how much data you can transmit in a period of time. | Latency measures the time delay when sending data. A higher latency causes a network delay. |
| How to measure? | Manually calculate throughput by sending a file or using network testing tools. | Calculate latency by using ping times. |
| Unit of measurement | Megabytes per second (MBps). | Milliseconds (ms). |
| Impacting factors | Bandwidth, network processing power, packet loss, and network topology. | Geographical distances, network congestion, transport protocol, and network infrastructure. |

## High Load Applications Architecture

### Problems

- Single point of failure - SPOF is a part of a system that, if it fails, will stop the entire system from working. SPOFs are undesirable in any system with a goal of high availability or reliability.
- Bottlenecks - Bottleneck occurs when the capacity of an application or a computer system is severely limited by a single component. It has lowest

throughput of all parts of the transaction path. **System works as fast as the slowest part of the system**.

## Main principles

- Reliability
- Scalability
- Efficiency
- Performance

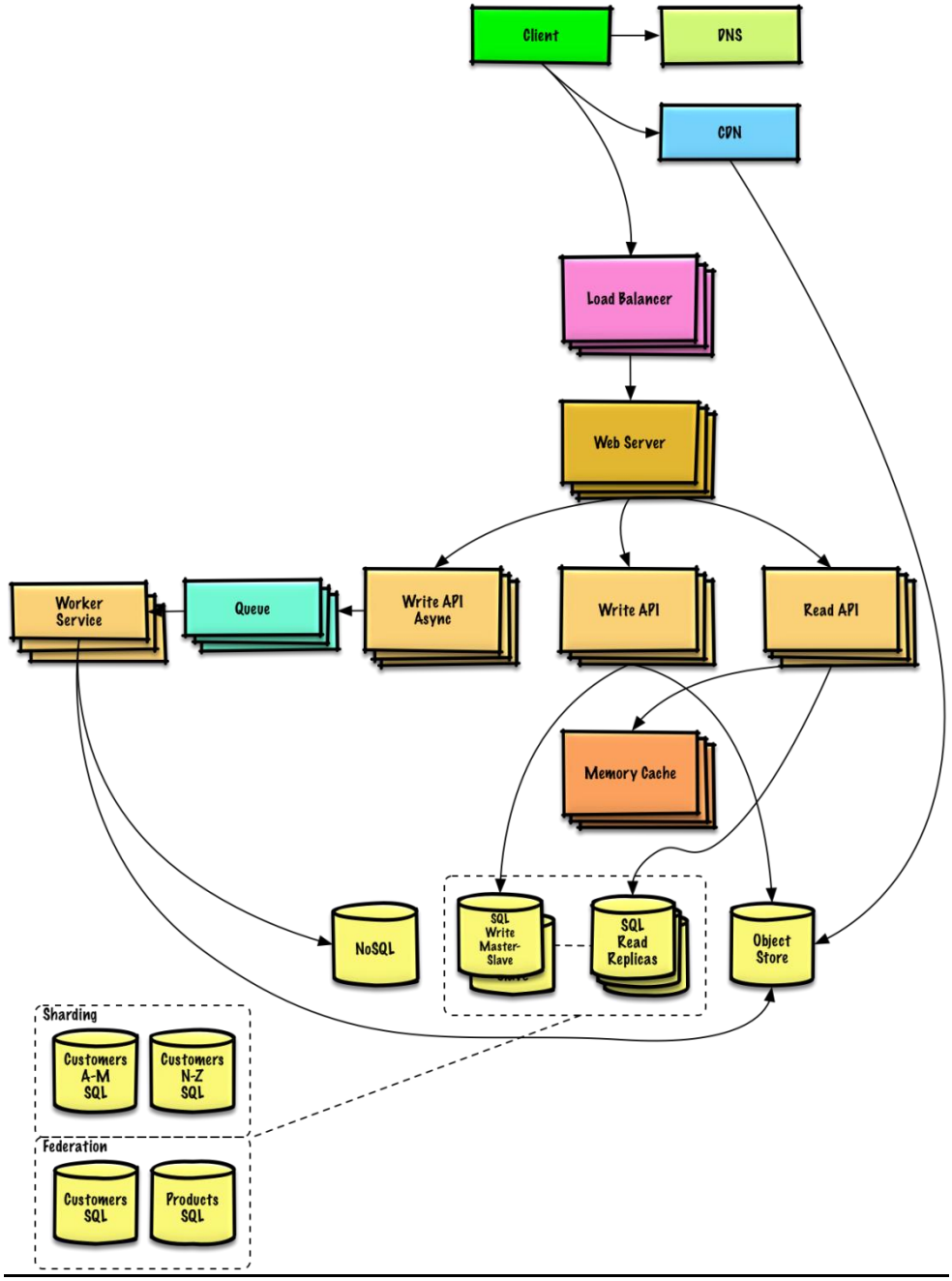
## Simplicity is a key

- It is better to have a bunch of simple components in the system.
- Simple things are easier to understand, analyze, improve, replace.
- Look at SOLID. S there stands exactly for that - Single Responsibility Principle. Each component in the system should serve one and only one goal.

## Proper approach for building High Load Applications

- Always keep in mind what is your main goal.
- Start with simplest solution (**Simple is not weak**).
- Continuously look for bottlenecks and SPOFs - improve first and avoid second.
- Split complex things into simple ones.

# Architecture Patterns

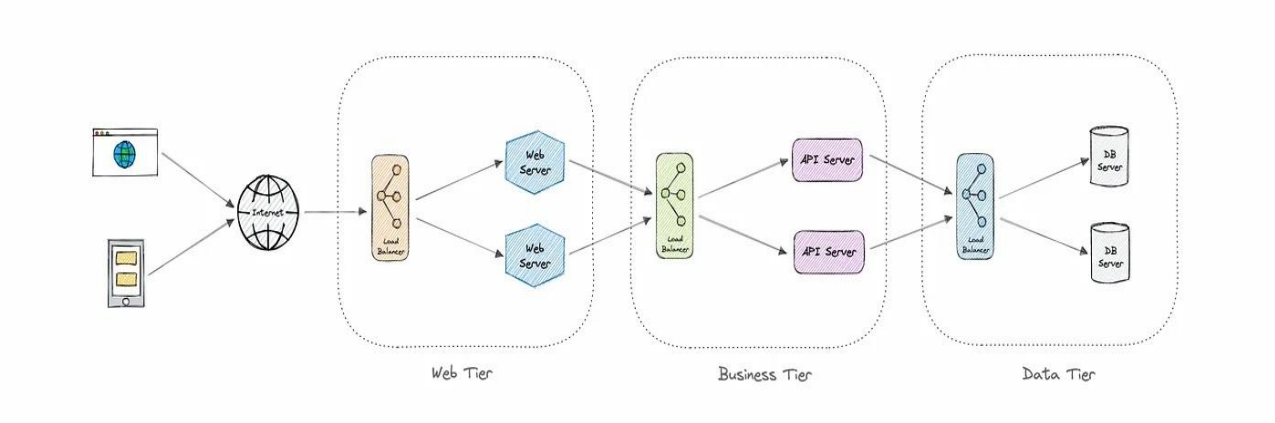



## Load Balancer

- This pattern is used to distribute incoming traffic evenly across multiple servers to prevent overload and improve the system's overall performance. Load balancing patterns can be applied at different levels, including network, transport, application, and database.

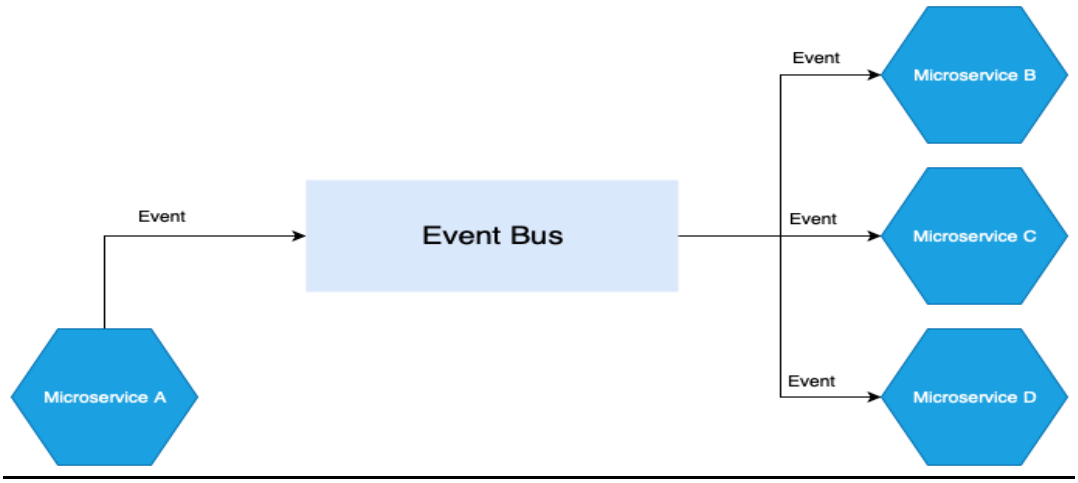
## N Tier

- This pattern involves dividing an application into separate tiers, each responsible for a different aspect of the application, making it easier to scale and manage



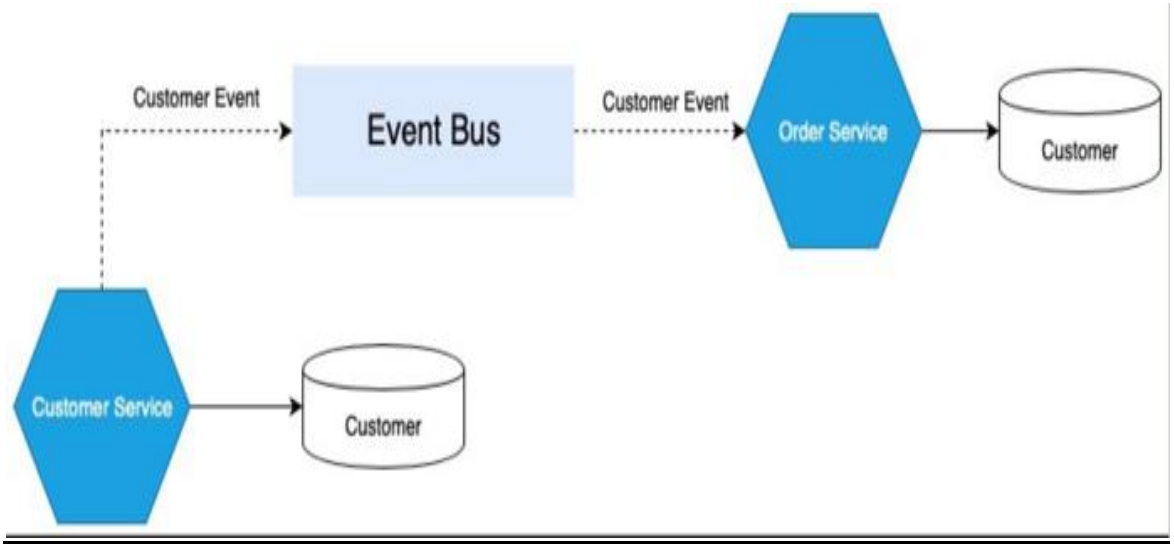

## Event Notification

- Is a pattern in which a system sends an event in order to notify other systems of a change.
- A defining feature of this architecture is that the system doesn't expect a response from the event at all.
- There should be a separation between the logic that sends the event and any logic that get triggered by it.
- This pattern is handy because it makes a lower level of coupling simple.



## Event-carried State Transfer

- The event-carried state transfer pattern is very similar to the event notification pattern.
- It often shows up when you're looking for less communication between systems.
- This is achieved by duplicating data across multiple systems
- Write operations are done on the master, and then the replica can be accessed by other systems.

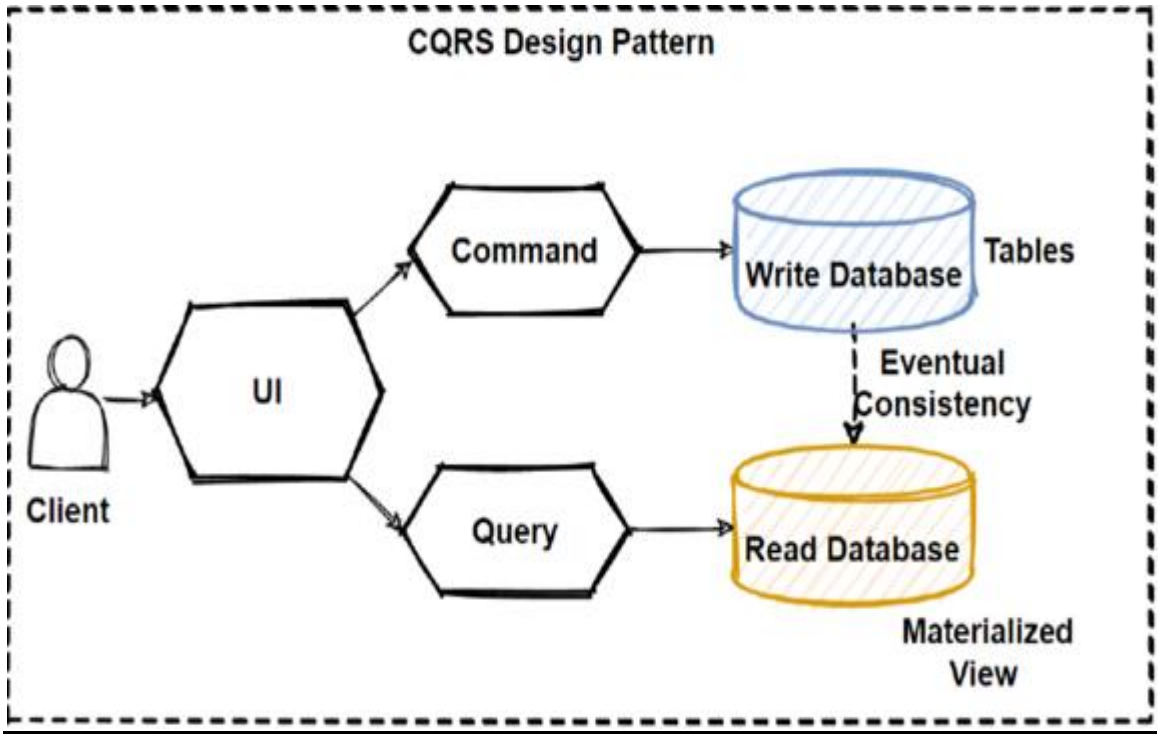



## Event Sourcing

- This pattern is less commonly used but the main idea of event sourcing all changes to a systems state is recorded as an event.
- You should be able to recreate the current state by replaying all previous events.
- This is a fundamentally different way of looking at a system because the event store is the source of truth, not the database.
- A commonly used example for describing this pattern is a version control system like git where each commit is an event.
- Built for audit and replay purposes. Finance systems use this pattern a lot.

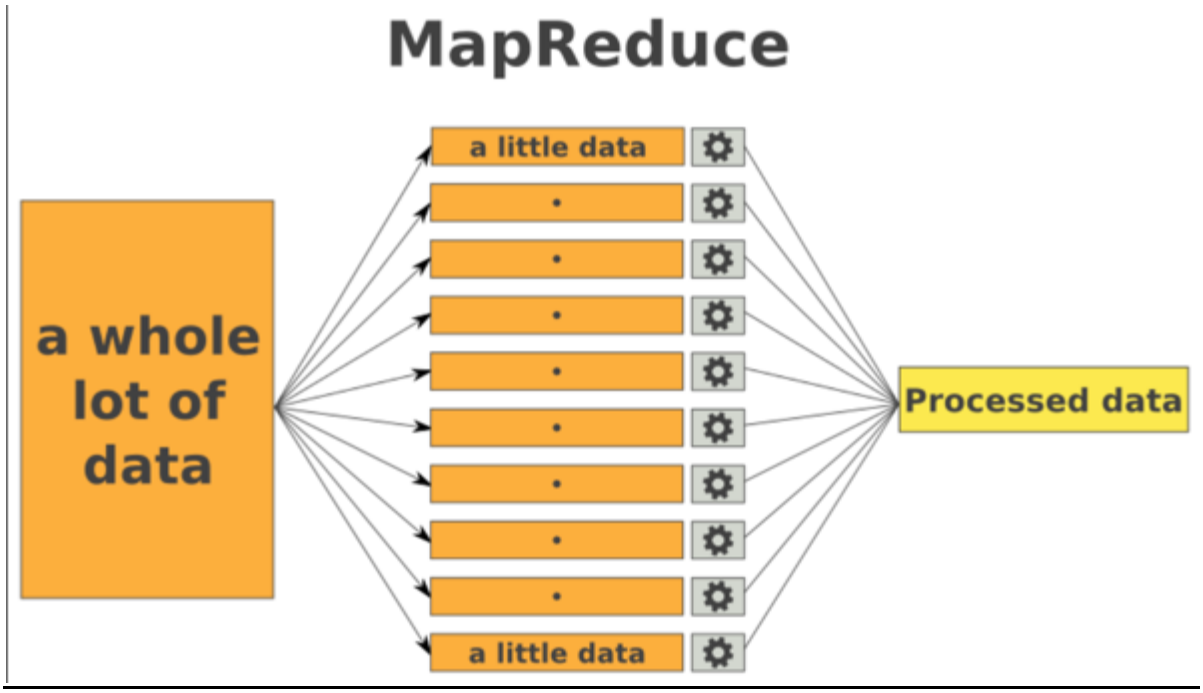
## CQRS

- Command Query Responsibility Segregation Pattern: This pattern separates read and write operations, allowing for the scaling and optimization of each independently.



## Map-Reduce

- This pattern is commonly used in big data processing and involves splitting large data sets into smaller chunks, processing them independently, and then combining the results.




## Microservices

- It is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.
- In microservices architecture, services are fine-grained and the protocols are lightweight.
- Go through the handwritten notes for more details about the Microservices.