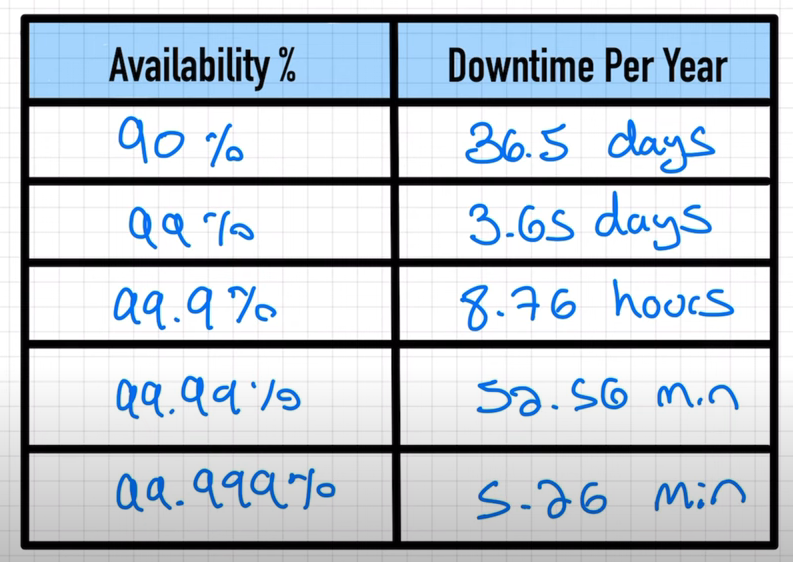
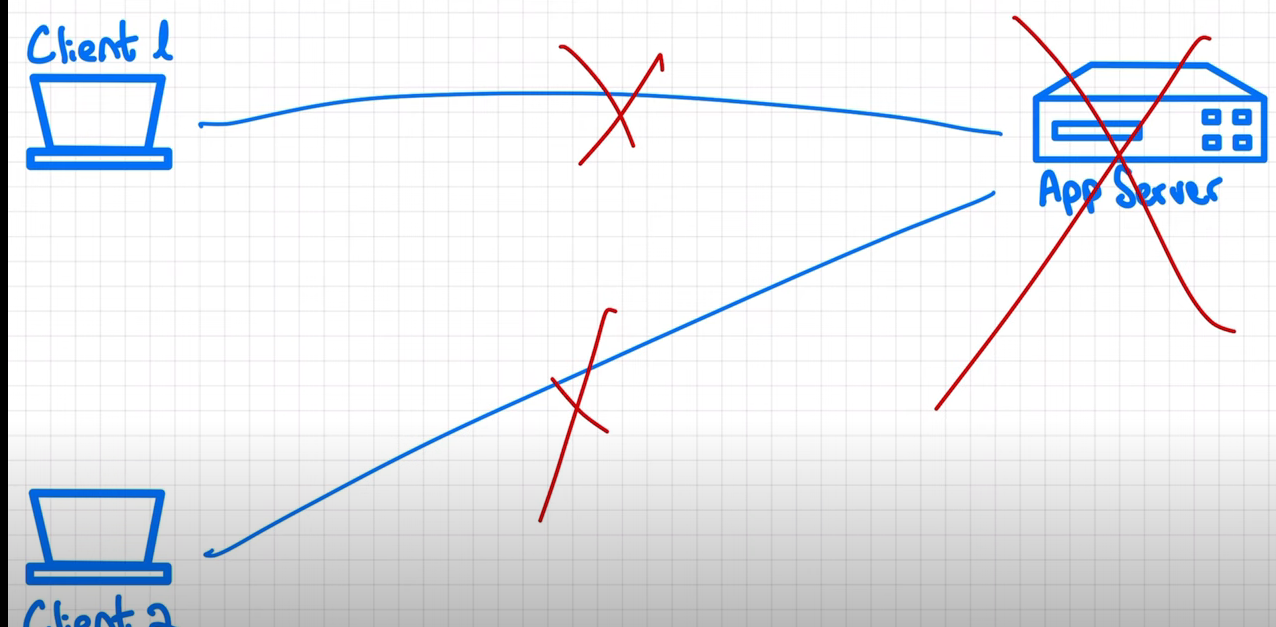
**What are the best practices for designing high-availability systems?**

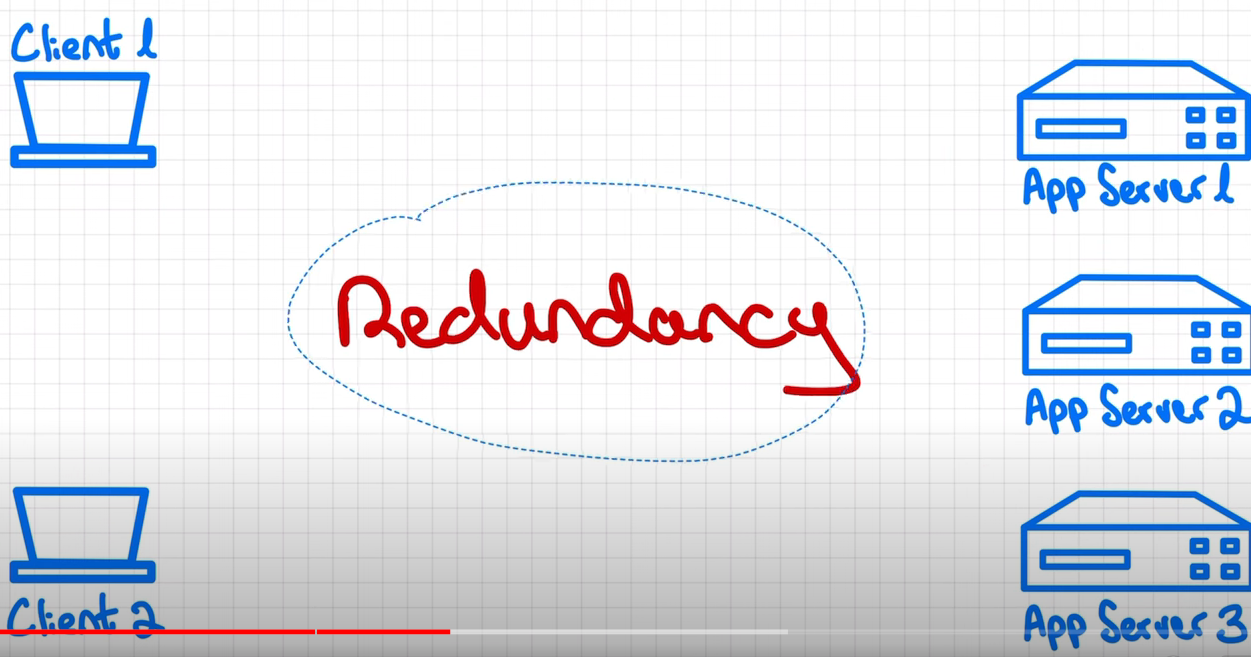
* High-availability systems are designed to minimize downtime and maximize performance, reliability, and user satisfaction. They are essential for critical applications and services that cannot afford to fail or slow down.
* The percentage of time the system is **up** and **performing** its intended function
* The availability is important for a system, any minute a system down can cause huge revenue loss.
* A air traffic system will always require high level of availability than a restaurant reservation system.
* If the system is 90% available the then it will be 36.5 day will be down a year(huge loss).

****

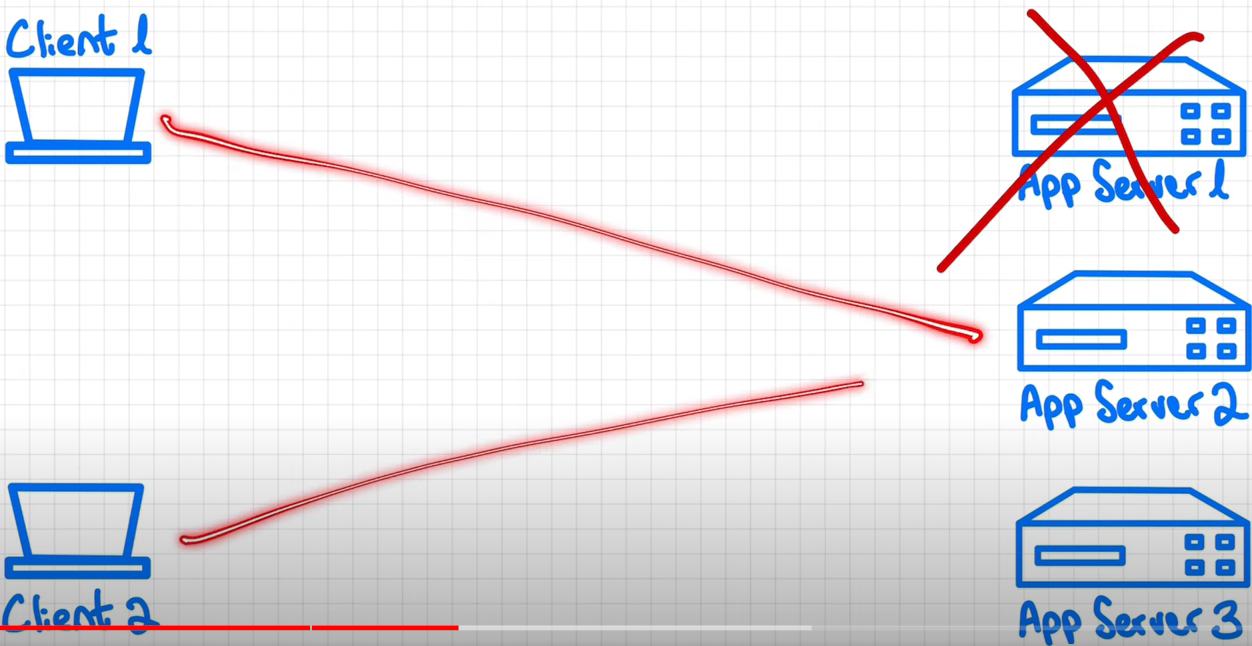
* **Reasons why service would fail for high availability.**
* Hardware failure
* Servers crash
* Power outages
* Natural disasters
* Resources run out (over usage/overloaded and not accept new request)
* Software bugs/defects can cause various ways(memory leaks/ software developer are not good)
* **How can we eliminate high availability failure**
* We can achieve it by eliminating single point of failure.



* If the app server goes down then all the clint will start failing there will be no alternat server to reroute the requests which is the **single point failure**.
* Also as the client increase the app server will become boatel neck and we will need to have power full machine to server more requests per seconds.
* To eliminate the single point of failure we can multiple app servers(not instances) in order to serve more request.



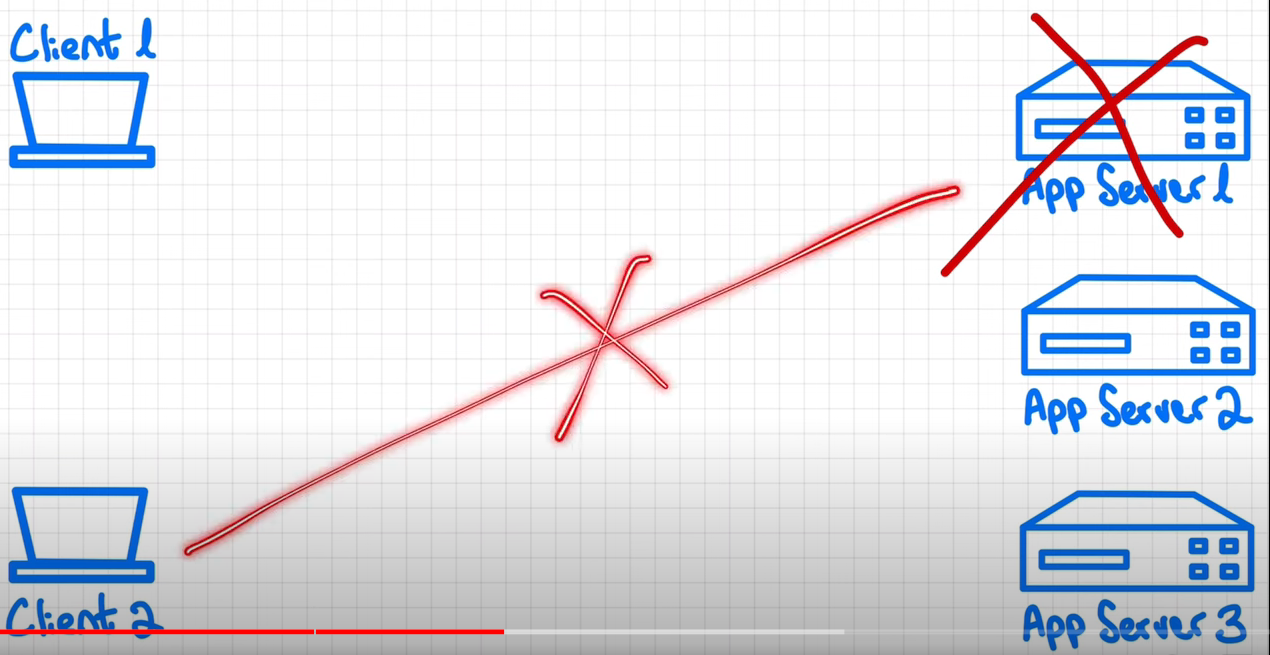
* Redundancy meanc duplicating part of the system if one goes down the other one will pick up the request and system will stay available.



* Also having multiple servers running parallel request also can be survived parallel.



* **But having multiple server running how do the client knows which server needs to be connected and not to connect to the dead server.**



* **some best practices**
* Define your availability goals
* [Choose the right architecture](https://www.linkedin.com/advice/0/what-best-practices-designing-high-availability#choose-the-right-architecture)
* [Implement best coding practices](https://www.linkedin.com/advice/0/what-best-practices-designing-high-availability#implement-best-coding-practices)
* [Test your system thoroughly](https://www.linkedin.com/advice/0/what-best-practices-designing-high-availability#test-your-system-thoroughly)
* [Monitor and optimize your system continuously](https://www.linkedin.com/advice/0/what-best-practices-designing-high-availability#monitor-and-optimize-your-system-continuously)

1. **Define your availability goals**

Before you start designing your system, you need to define your availability goals and metrics. Availability is usually measured by the percentage of time that your system is operational and functional. For example, a 99.9% availability means that your system is down for less than nine hours per year. However, availability also depends on other factors, such as latency, throughput, scalability, and fault tolerance. You need to specify your target values and thresholds for each of these factors and how you will monitor and evaluate them.

1. **Choose the right architecture**

The architecture of your system determines how it is structured, organized, and distributed. It affects how your system handles load balancing, redundancy, replication, backup, recovery, and failover. There are different types of architectures for high-availability systems, such as monolithic, microservices, serverless, or hybrid. Each type has its advantages and disadvantages, depending on your use case, resources, and preferences. You need to choose the right architecture that suits your needs and supports your availability goals.

1. **Implement best coding practices**

The quality of your code also affects the availability of your system. You need to implement best coding practices that ensure your code is readable, maintainable, testable, and secure. You need to follow coding standards, conventions, and guidelines that make your code consistent and easy to understand. You need to use appropriate tools and frameworks that simplify your development process and automate your testing and deployment. You need to apply design patterns and principles that make your code modular, reusable, and extensible. You need to avoid common coding errors and vulnerabilities that can cause bugs, crashes, or breaches.

1. **Test your system thoroughly**

Testing your system is crucial to verify its functionality, performance, and reliability. You need to test your system thoroughly at different levels, such as unit, integration, system, and acceptance. You need to use different testing methods, such as functional, non-functional, load, stress, and security testing. You need to simulate different scenarios, such as normal, peak, and failure conditions. You need to identify and fix any issues or defects that can affect your system's availability.

1. **Monitor and optimize your system continuously**

Monitoring and optimizing your system is an ongoing process that helps you maintain and improve your system's availability. You need to monitor your system continuously using various tools and metrics that collect and analyze data about your system's status, performance, and behavior. You need to optimize your system regularly using various techniques and strategies that enhance your system's efficiency, scalability, and resilience. You need to update and upgrade your system periodically using best practices and standards that keep your system current and compatible.