# Achieving Consensus in a Cluster – Part 1

This lesson covers achieving consensus in a cluster.

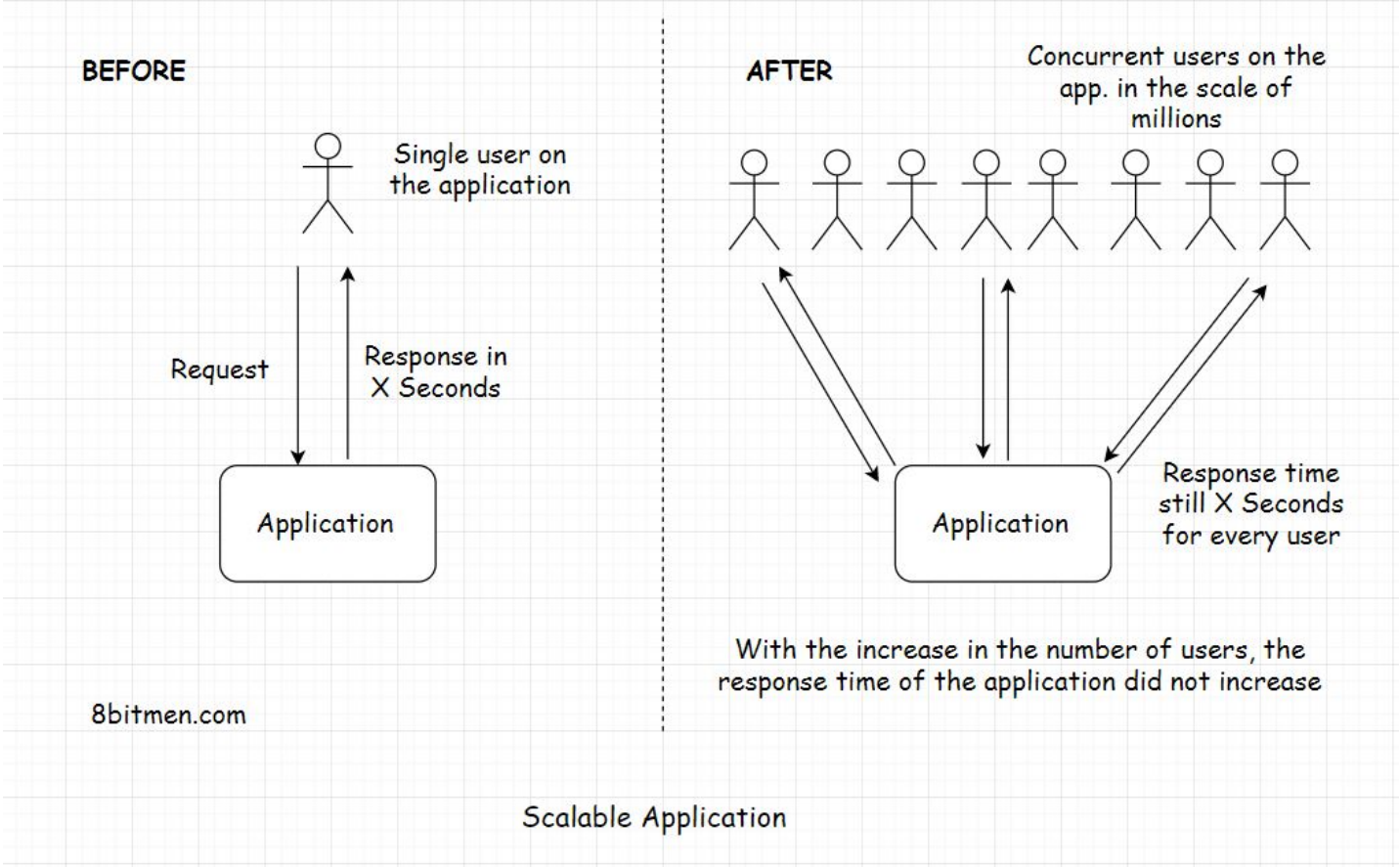
**We'll cover the following**



## Overview

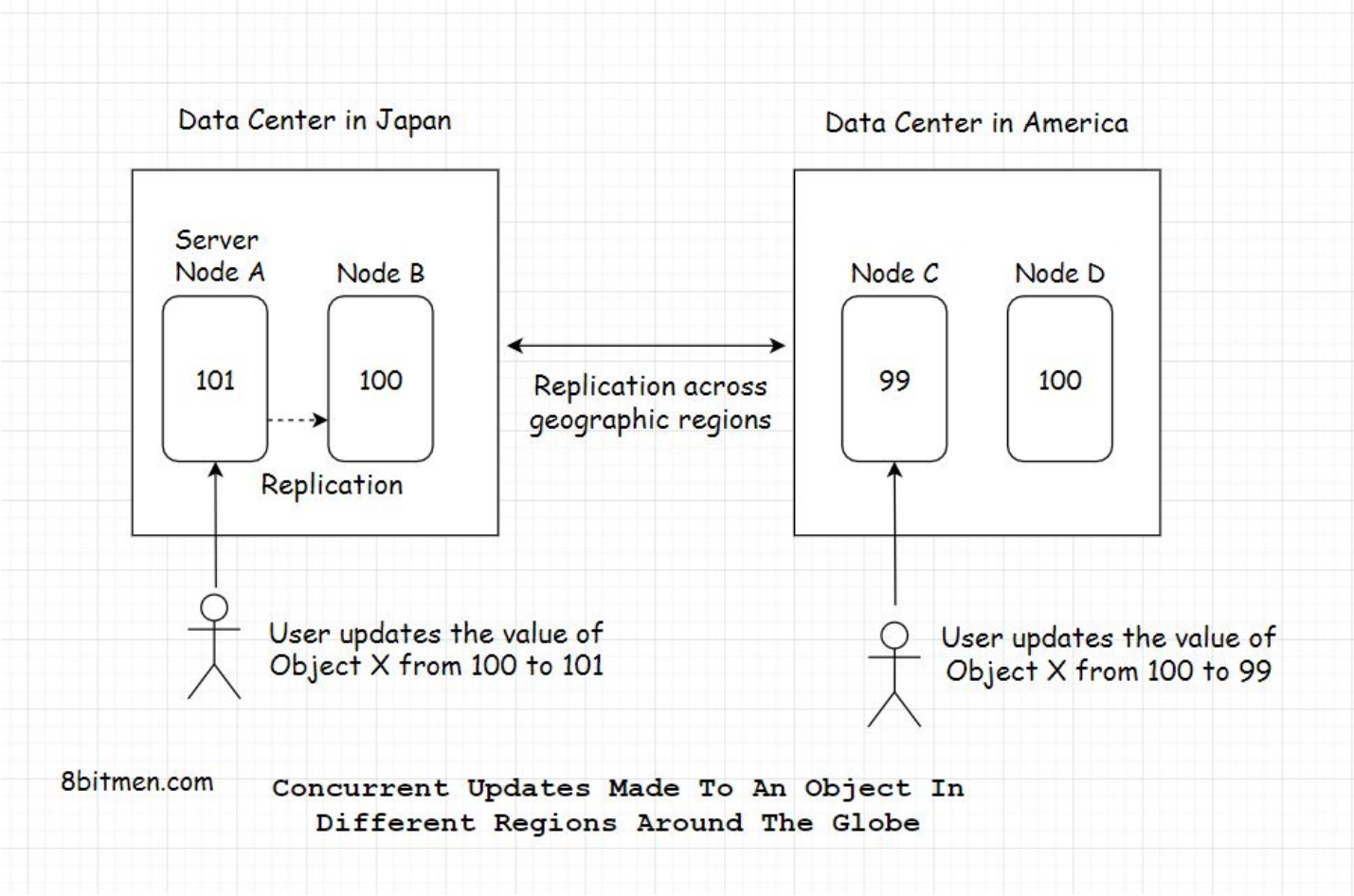
In the previous lesson, I discussed how nodes work together in a cluster with a simple two-node cluster example. In this lesson, you will learn how multiple nodes running in a cluster in replication mode reach a consensus on the value of an object.

The whole point of having multiple instances in a cluster is to make the system *scalable, highly available*, and *fault-tolerant*. When a service is scalable, a large number of concurrent users can perform *read-write* operations on the service without experiencing any sort of latency as the traffic builds up.




## Consensus #

Now, when a high number of concurrent users interact with the system hosted by thousands of nodes deployed across the globe, all the nodes get updated concurrently in real-time. That means at any point in time different nodes may have different values for a certain object. Also, among those nodes, a few might even fail.




When a user requests the value of a certain object from the system, the system is expected to give a single, consistent value of that object despite the several different values that the thousands of nodes running in the cluster hold for that particular object.

*How does the system achieve this?*

*By reaching a consensus among all the nodes in the cluster on the value of that object.*

To return a single data value to the user, the system has to make all the nodes agree on a common value. All the nodes have to reach a consensus. This mechanism of reaching a consensus in a cluster has several applications such as facilitating database transactions in distributed systems, atomic broadcasts, state machine replication, and so on.

Let's get a quick look into reaching a consensus in a distributed system from a database transaction standpoint.

## Consensus from a database transaction standpoint #

When the application is expected to receive a large amount of traffic, it has to scale horizontally on the fly to avert any kind of latency that arises due to the heavy traffic load.

*NoSQL* databases became popular solely due to their ability to auto-scale dynamically; something that was not so trivial to achieve with the traditional relational databases.

## Data consistency models #

When the databases run in a distributed environment, there are two data consistency models that apply to the transactions. When I say transactions here, it doesn't necessarily mean a financial transaction. A transaction can be the persistence of any kind of event. It may be an increment in the Like-counter on a post made by a user on a social network or the writes performed by players in an online multiplayer game.

The two data consistency models involved in this are the *eventual consistency* and the *strong consistency*. *Eventual consistency* means that when thousands of nodes deployed across the globe hold different values of an object the system, after a short span of time, will reach a consensus and have one single value for the object across all the nodes running globally. The value of that object will eventually be consistent.

On the contrary, *strong consistency* means that at any point in time all the nodes deployed across the globe should have the same consistent value of an object. The strong consistency data model is key when implementing financial services.

Cloud providers generally keep the nodes that ensure consistent data in the same availability zone.

There is also a theorem called the *CAP Theorem* that is key when writing services that would run using a distributed system such as a *NoSQL* database. CAP stands for *consistency, availability*, and *partition tolerance*. As per the CAP Theorem, when the nodes in our system fail, we have two choices: we can keep our service available, and the users can continue to perform the write operations on it. Or, we can disable the writes in our system until the nodes bounce back online. The latter choice ensures a strong consistency of data in our system.

> I have discussed the following topics in detail in my Web Application and Software Architecture 101 course:
>
> - *Eventual consistency*
> - *Strong consistency*
> - *CAP theorem*
> - *How NoSQL databases work?*
> - *Why can't traditional relational databases scale horizontally on the fly? Which database to pick for your use case?*

Reaching a consensus among the nodes is vital for the reliability of distributed systems and for the decentralized tech like blockchain and the cryptocurrencies built on it.

*Let's continue this discussion in the next lesson.*