Imagine you are a general trying to coordinate an attack together with another friendly general. The only way to win the battle is if both attack at the same time. Both of you have a radio transmitter that can transmit and receive morse code. The transmitter runs of battery, so you cannot keep it on all the time. How do you coordinate the attack?

The answer is: Because any message might be lost, there is no way to coordinate the attack with complete certainty.

You have probably read about that problem before somewhere. The problem might seem staged, but when it comes to computers, it is a real issue. A real issue that has been solved by TCP/IP. In this article, I try to explain the two most popular algorithms for distributed consensus - Paxos and Raft. They solve a similar issue:

You have a cluster of *n* servers. Each of the servers might fail at any time. A constant flow of random messages hits each of the servers. How do you make the cluster agree on what messages are received and in what order?

The short answer is: It is complicated.

The long answer is:

# What are Distributed-State-Machines?

As mentioned above, there are two main algorithms for reaching consensus in a distributed system.

Before we get to explaining the algorithms, let us look into what reaching consensus in a distributed system means. To understand what that means, it is useful to understand what a state machine is. A state machine is a system that stores a state, and given an action goes to another state. What consensus in a distributed system means is that we are able to treat a system of multiple machines as a state machine. That is useful for programmers because they do not have to think about the distributed nature of the system.

# How about giving machines roles?

Ok, so we are trying to create a distributed state machine. First, consider a system with a single machine. We send a message to the machine and it stores them in the same order as they were received. That machine is the leader.

## Leader

The leader is the machine that receives all the messages. If we only have a single machine, and call that machine the leader it might not be a distributed system, but it gets the basic point across.

## Followers

When we have the leader we need more machines in the system. We'll call the machines followers. They also store the state of the system and check whether the leader is alive.

If they suspect the leader is dead they tries to become the leader.

## Candidate

That is a machine that is trying to become the new leader. For it to become the leader, all the other machines have to agree that it is the leader.

# So what does is the lifecycle?

With these roles in place, we have set the play for a lifecycle. Firstly, let us assume that every machine is a follower. The machines then elect a leader.

1. A machine becomes the leader
2. Messages are received by the leader
3. The leader dies
4. Repeat...

You now hopefully understand how the algorithms work.

# Where do Paxos and Raft differ?

In leader election.

Paxos requires the leader to be up to date.

Raft fetches the messages from the other state machines.