FLP Impossibility

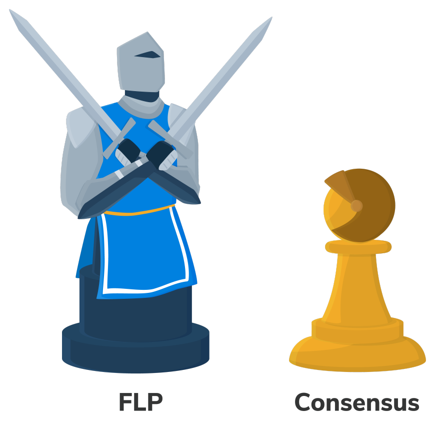
Learn if consensus is possible in an asynchronous model when nodes can crash.

We'll cover the following

- Understanding the FLP impossibility theorem
  - Understanding the proof
- Application of FLP to real-world systems
- Summary

Due to the Two Generals’ Problem, we have established that we can't guarantee consensus if our network can drop messages, even under the friendly settings of the synchronous computational model. Due to FL82, we also saw that consensus is possible under a synchronous model when the network is non-faulty, but nodes can show crash-stop failures. The next question is: Is consensus possible in an asynchronous setting where a network is assumed to be non-faulty but nodes can exhibit crash stop failures? Unfortunately, it’s not possible due to a result called FLP impossibility. We’ll understand what the FLP impossibility is in the rest of this lesson.

The FLP impossibility is a crucial result in distributed systems. It addresses whether a deterministic consensus algorithm can satisfy agreement, validity, termination, and fault tolerance in an asynchronous distributed system. However, the FLP theorem suggests that it is impossible to implement consensus algorithms that tolerate even a single node fault.



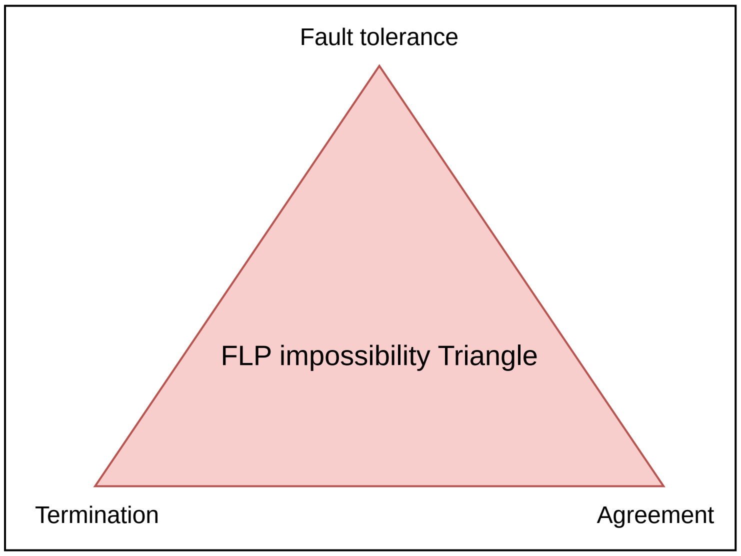
This result can be counterintuitive and confusing, considering the widespread use of consensus algorithms such as Paxos, Raft, and Zookeeper in large distributed systems where node failures are common.

Understanding the FLP impossibility theorem

Before translating the FLP theorem into practical applications, it is important to understand the assumptions and context in which it operates. The FLP impossibility theorem is based on the asynchronous network model, where message delivery time between nodes is finite but unbounded. It is hard to differentiate a dead node from a slow processing node. The theorem assumes non-Byzantine failures and that the message channels do not lose messages.

According to the **FLP theorem**, simultaneously achieving all three properties of termination, agreement, and fault tolerance is impossible under the asynchronous network model. These three properties are defined as follows:

1. **Termination (liveness):** All nodes (that have not failed) eventually decide.
2. **Agreement (safety):** All nodes (even nodes that have failed after deciding) that decide should decide on the same value. If all nodes have the same initial input, that value should be the only possible decision value. (We subsumed the validity condition in the agreement condition for the sake of this discussion.)
3. **Fault Tolerance:** All nodes require a protocol to be effective in case of node failures.



FLP triangle

The FLP theorem does not imply that distributed consensus is generally impossible or that we cannot have any of the three properties. Instead, it implies that we cannot have all three properties simultaneously under the asynchronous network model at all times.

Understanding the proof

The FLP proof uses a clever "trick" to show that even in a fault-tolerant system, there are runs that do not terminate. The proof tricks the system into treating a process as if it has failed, and then lets it resume execution. This creates a delay that prevents the system from progressing, despite no actual failures.

Point to ponder

Question

What does "impossibility" mean in distributed computing?

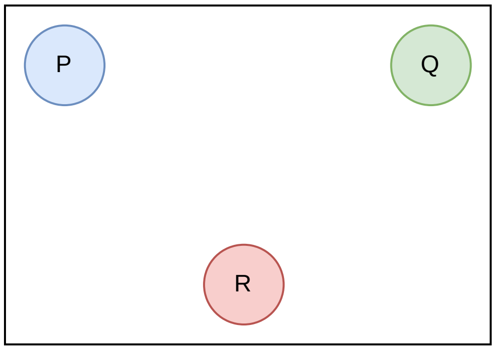
Hide Answer

In distributed computing, an algorithm is considered totally correct if it computes the right thing and always terminates. The FLP impossibility result shows that there is no totally correct algorithm for achieving consensus in a completely asynchronous system at all times. While runs that do not terminate are extremely unlikely, they are still possible, which means consensus cannot always be achieved.

Imagine a group of people trying to agree on where to have lunch. P is the group leader and is crucial in deciding the restaurant. However, the group is fault-tolerant, meaning that if P is unavailable, they can still decide on a restaurant and move on.

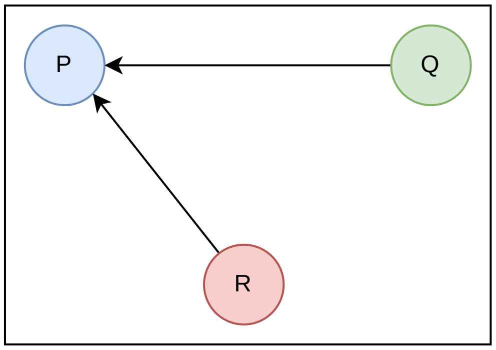
Let's say the group uses a fault-tolerant consensus protocol to decide on the restaurant, meaning it should still function even if P is unavailable. However, the protocol is "tricked" into treating P as if it had failed by delaying its messages. The protocol performs the following steps:

1. Everyone sends their preferred restaurant to P for aggregation.
2. P aggregates the preferences and sends the result back to all others.
3. Q and R sense that P has “failed” before the group can indicate their acceptance or rejection.
4. All others start a new round of aggregation without P.
5. P resumes execution and rejoins the group, which restarts the protocol from the beginning.



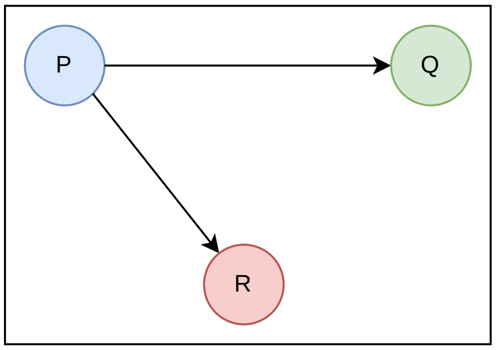
Three friends that need to decide on a restaurant

1 of 6



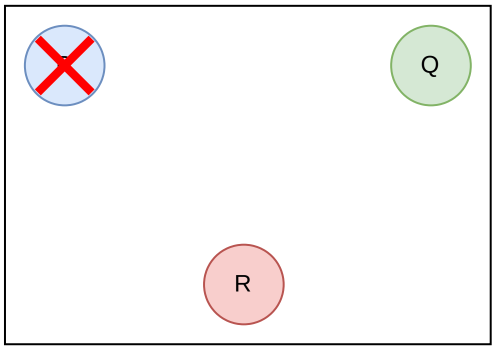
Q and R send their suggestion to P

2 of 6



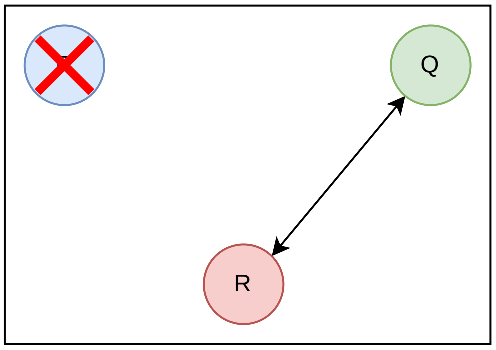
P aggregates the preferences and sends the result back to Q and R

3 of 6



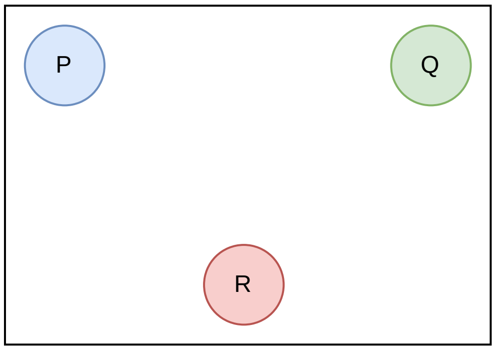
Q and R sense as if P has “failed” before the group can indicate their acceptance or rejection

4 of 6



Q and R assume P has failed and start a new round

5 of 6



P resumes execution and rejoins the group, which restarts the protocol from the beginning

6 of 6

> **Note:** If you want to see more details of the theorem, please click **this↴** (The FLP paper was awarded the Dijkstra Prize in 2001 and is one of the most influential works in the field of distributed consensus. We highly encourage our learners to go through the details of the theorem above.)

In summary, the FLP result shows that having a completely asynchronous consensus protocol that can tolerate even a single unannounced process death is impossible.

Application of FLP to real-world systems

In real-world systems, it is possible to make sacrifices or relax constraints to develop a consensus algorithm that meets business requirements. For example, a partially synchronous network model can be used where failures can be detected through timeout mechanisms. This means that a predetermined timeout can bound message delays. Therefore, for most real-world scenarios, we might escape the assumptions of asynchrony, where the system works like a synchronous system for quite a while before drifting to an asynchronous state.

However, consensus algorithms often sacrifice one of the three properties of the impossibility triangle to devise a compromise. For example, Paxos and Raft sacrifice termination under certain conditions. Raft uses randomized election timeouts to decrease unavailability periods and to reduce the likelihood of the protocol not terminating and not making progress. Randomized algorithms are commonly used in many consensus algorithms.

> **Note:** It is foreshadowing for the upcoming chapters, but this important point deserves to be mentioned here. All the practical consensus algorithms we study in this section (two-phase commit, Paxos, Raft) will sacrifice termination to make an acceptable compromise out of the FLP impossibility.

Summary

The FLP impossibility theorem is a fundamental concept in distributed systems. Although it suggests that achieving all three properties of termination, agreement, and fault tolerance is impossible, in practice, we can relax the constraints made by the theorem or sacrifice one of the properties that meet our business needs. Consensus algorithms often classify their required conditions into safety (nothing bad ever happens) and liveness (something good eventually happens). They often sacrifice liveness when for example, the systems drift into an asynchronous-like situation. Such algorithms never compromise safety conditions. Such a consensus algorithm works fine for most cases, but occasionally it can stop making progress. It will be a recurring concept in the upcoming chapter on two-phase commit, Paxos and Raft.

← Back
Consensus Prerequisit...

Next →
The Byzantine Genera...

Mark as Completed