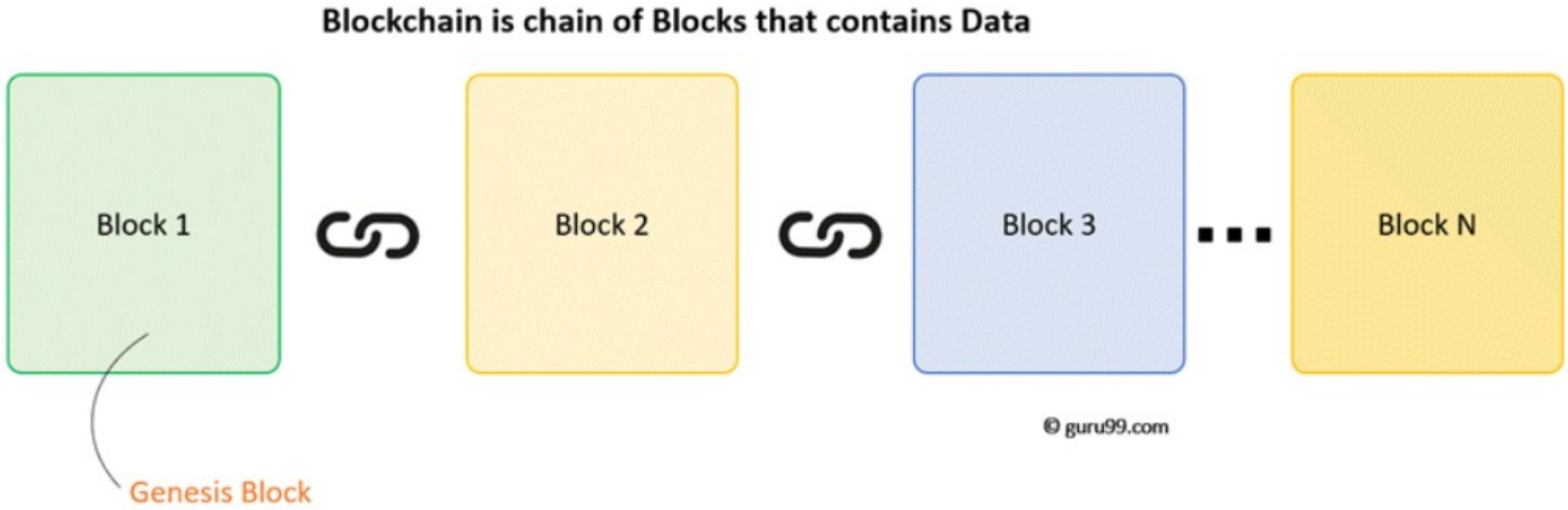
Summary

- A Blockchain is a chain of blocks that contain information
- The blockchain is not Bitcoin, but it is the technology behind Bitcoin
- Every block contains hash.
- Each block has a hash of the previous block
- Blockchain require Proof of Work before a new block is added
- The blockchain database is disturbed amongst multiple peers and is not centralized.
- Block chain technology is Resilience, Decentralize, Time reducing, reliable and its offers unalterable transitions
- The blockchain is Available in three different variants 1) Public 2) Private 3) Consortium
- Higher cost, slower transactions, small ledger, the risk of error are some disadvantage of using this technology

Blockchain Architecture

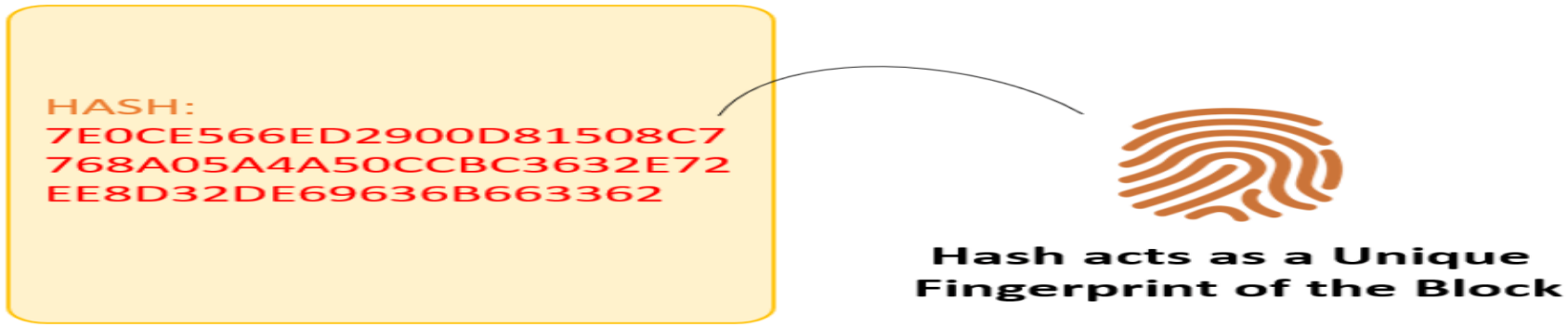
Now in this Blockchain Technology tutorial, let's study the Blockchain architecture by understanding its various components:



Understanding Block- Hash

A block also has a hash. A can be understood as a fingerprint which is unique to each block. It identifies a block and all of its contents, and it's always unique, just like a fingerprint. So once a block is created, any change inside the block will cause the hash to change.

Therefore, the hash is very useful when you want to detect changes to intersections. If the fingerprint of a block changes, it does not remain the same block.



Each Block has

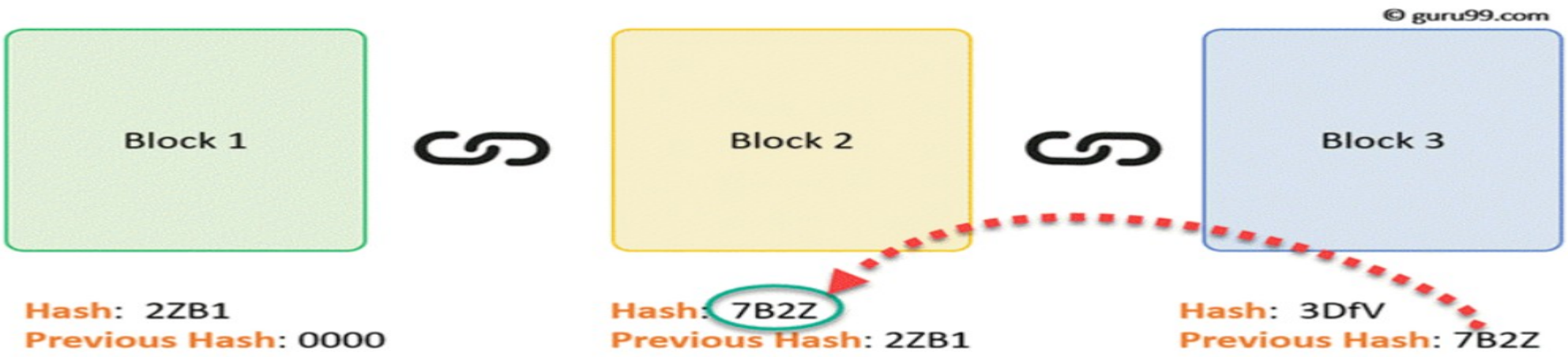
1.Data

2.Hash

3.Hash of the previous block

Consider following example, where we have a chain of 3 blocks. The 1st block has no predecessor. Hence, it does not contain has the previous block. Block 2 contains a hash of block 1. While block 3 contains Hash of block 2.

Therefore, changing a single block can quickly make all following blocks invalid.



Hence, all blocks are containing hashes of previous blocks. This is the technique that makes a blockchain so secure. Let's see how it works -

Assume an attacker is able to change the data present in the Block 2. Correspondingly, the Hash of the Block also changes. But, Block 3 still contains the old Hash of the Block 2. This makes Block 3, and all succeeding blocks invalid as they do not have correct hash the previous block.

Proof of Work

Hashes are an excellent mechanism to prevent tempering but computers these days are high-speed and can calculate hundreds of thousands of hashes per second. In a matter of few minutes, an attacker can tamper with a block, and then recalculate all the hashes of other blocks to make the blockchain valid again.

To avoid the issue, blockchains use the concept of Proof-of-Work. It is a mechanism which slows down the creation of the new blocks.

A proof-of-work is a computational problem that takes certain to effort to solve. But the time required to verify the results of the computational problem is very less compared to the effort it takes to solve the computational problem itself.

In case of Bitcoin, it takes almost 10 minutes to calculate the required proof-of-work to add a new block to the chain. Considering our example, if a hacker would to change data in Block 2, he would need to perform proof of work (which would take 10 minutes) and only then make changes in Block 3 and all the succeeding blocks.

This kind of mechanism makes it quite tough to tamper with the blocks so even if you tamper with even a single block, you will need to recalculate the proof-of-work for all the following blocks. Thus, hashing and proof-of-work mechanism make a blockchain secure.

However, there is one more method which is used by blockchains to secure themselves, and that's by being distributed.

Instead of using a central entity to manage the chain, Blockchains use a distributed peer-peer network, and everyone is allowed to join. When someone enters this network, he will get the full copy of the blockchain. Each computer is called a node

when any user creates a new block. This new block is sent to all the users on the network. Each node needs to verify the block to make sure that it hasn't been altered. After complete checking, each node adds this block to their blockchain.

All these nodes in this network create a **consensus**. They agree about what blocks are valid and which are not. Nodes in the network will reject blocks that are tampered with.

So, to successfully tamper with a blockchain
1.You will need to tamper with all blocks on the chain
2.Redo the proof-of-work for each block
3.Take control of greater than 50% of the peer-to-peer network.

After doing all these, your tampered block become accepted by everyone else. This is next to impossible task. Hence, Blockchains are so secure. Next in this beginners Blockchain development tutorial, we will learn how a Blockchain transaction works?

Blockchain Variants

Public:

In this type of blockchains, ledgers are visible to everyone on the internet. It allows anyone to verify and add a block of transactions to the blockchain. Public networks have incentives for people to join and free for use. Anyone can use a public blockchain network.

Private:

Consortium

The private blockchain is within a single organization. It allows only specific people of the organization to verify and add transaction blocks. However, everyone on the internet is generally allowed to view.

Consortium:

In this Blockchain variant, only a group of organizations can verify and add transactions. Here, the ledger can be open or restricted to select groups. Consortium blockchain is used cross-organizations. It is only controlled by pre-authorized nodes.

Solo

The Solo ordering service consists of a single node. When you use Solo, your network is clearly not decentralized and is not fault tolerant — but that's OK because, again, it is just for development purposes. Solo is designed to provide the ordering service in its simplest possible form so that you can focus on other matters, such as the development of your chaincode and application, without having to worry about the ordering service. However, this is obviously not suitable for production deployments. For production, Fabric 1.4.0 provides the Kafka ordering service.

Kafka

The Kafka ordering service leverages a cluster of Kafka brokers and a Zookeeper ensemble to provide for a crash fault tolerant (CFT) ordering service. It is possible for your ordering service to consist of several ordering nodes that are under the control of different organizations on your network. However, while the result is distributed, it is still not fully decentralized. The difference lies in the point of control because Kafka and Zookeeper are not designed to be run across large networks, but rather in a tight group of hosts. This means that practically speaking you need to have one organization run both the Kafka cluster and the Zookeeper ensemble. Given that, having ordering nodes run by different organizations doesn't give you much in terms of decentralization because they will all go to the same Kafka cluster, which is under the control of a single organization.

Hyperledger Caliper

Hyperledger Caliper provided by the Hyperledger project is a benchmark-ing tool for blockchains, used to measure the performance of specific block-chain implementations. The primary purpose of Caliper is to provide developers with a helping hand in trying to find the right blockchain frame-work, calculate resource consumptions, and cost estimation for setting up the network. The supported metrics are success rate, transaction through-put, transaction latency, and resource consumption (CPU, memory).

- Success rate indicates the number of transactions successfully com-mitted to the ledger. Failures can be caused by multiple factors suchas time-outs, network limitations, peer resources, chaincode, to namea few, and therefore, a failure cause is not easily identifiable.

- Transaction throughput indicates the number of transactions submit-ted to the ledger per second.

- Transaction latency indicates the time a transaction takes to beavailable across the whole network; this metric is calculated pertransaction.

Hyperledger Calipers has structured its architecture into three mainlayers: the adaptation layer, the interface and core layer, and lastly thebenchmark layer. Each layer provides functionalities that allow Caliper to communicate with the ledger, test the performance, and generate a reportbased on the tests

- Adaptation layer: Uses framework-specific adaptors to integrate thee xisting blockchain network into the Caliper framework.

- Interface and Core layer: Used to implement core functionalities that Caliper provides, these consist of:

  –Blockchain operating interfaces: consist of operations to deploy,I nstantiate, install, invoke, and query smart contracts.

  –Resource monitor: consists of the operations for starting ands topping the monitor that fetches the resource consumption statuses such as CPU and memory of the running network.Currently, only two types of monitors are supported, one that monitors the local process, and one that monitors the dockerc ontainers

- Quick High Level Overview:

- Peer can be part of one or more channel

- Every channel has a separate ledgerEvery Channel has one or more chain codes

- Every Chain code has a different endorsement policy

- Chaincode must be part of a channel. As the ledger is part of a channel. One channel can have as many chaincodes as possible.
- Chaincode must be installed in each peer that is part of the channel and instantiated.
- When a Chaincode gets instantiated a policy (endorsing) has to be defined. [consensus: before a transaction can be recorded in the ledger only if a rule is met]

Cryptogen

cryptogen is an utility for generating **Hyperledger Fabric** key material. It is provided as a means of preconfiguring a network for testing purposes. It would normally not be used in the operation of a production network.

In order to generate the crypto-material for the network, we utilize cryptogen, utilized for a preconfigured network in order to quickly setup the necessary key material, and should therefore not be used inproduction. Cryptogen generates the key material through a template file;the file specifies the number of organizations, the number of peers in eachorganization and the number of Orderers.

```
cryptogen generate
```

1. Generating the crypto-material will take the template file asinput and generate a folder with the material
2. Generating the genesis block with the channel policy as the-profile parameter.
3. Generating the channel file with the path to store.

A block header contains:

- Version: The block version number.
- Time: the current timestamp.
- Hash of the previous block.
- Nonce (more on this later).
- Hash of the Merkle Root.
- The current difficulty target. (More on this later).

CouchDB as the State Database