#Bitcoin #Bitcoin Developer #Blockchain for Developers

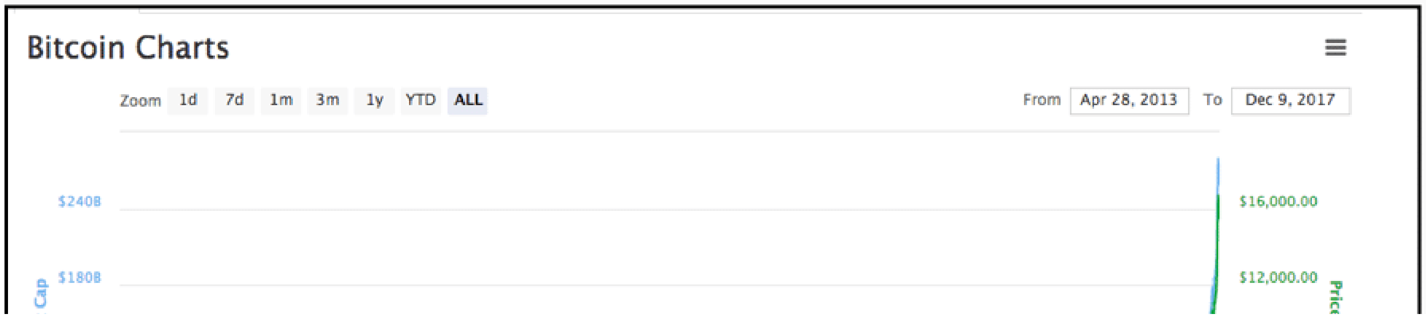
Become A Bitcoin Developer: Basic 101

Ameer Rosic

0:00 / 0:00

In this guide we are going to explore what it takes to be a bitcoin developer.

The fact of the matter is that "bitcoin" and "blockchain technology" in general is the hottest topic on earth right now. The value of 1 BTC has skyrocketed over the last few months (as of writing):



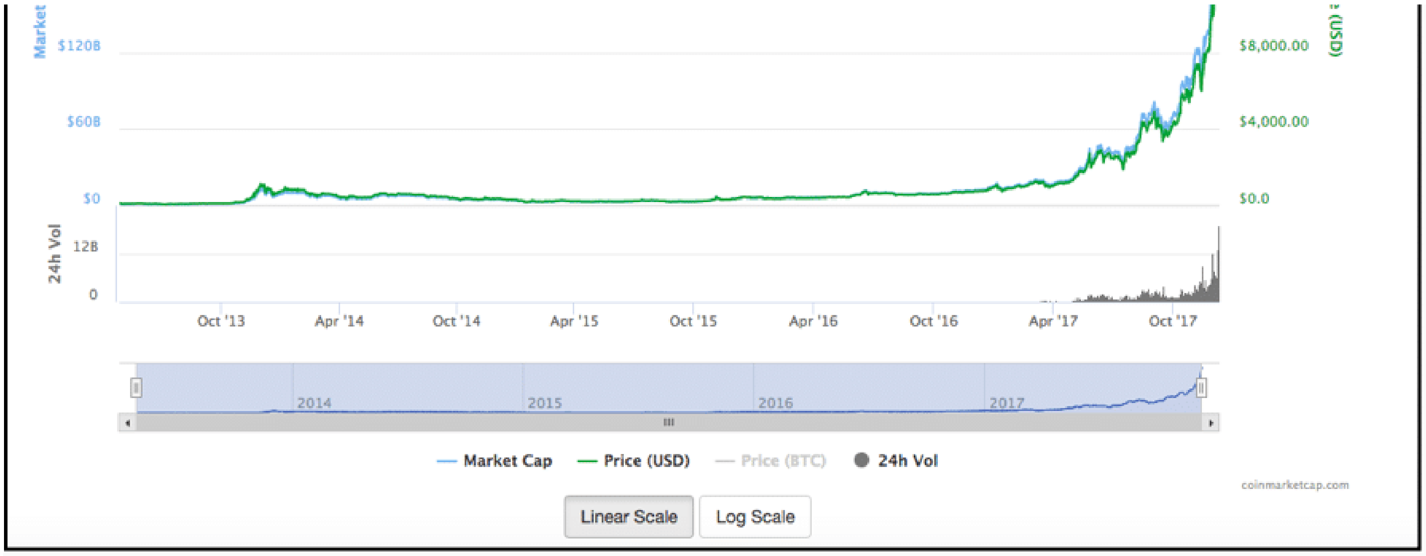
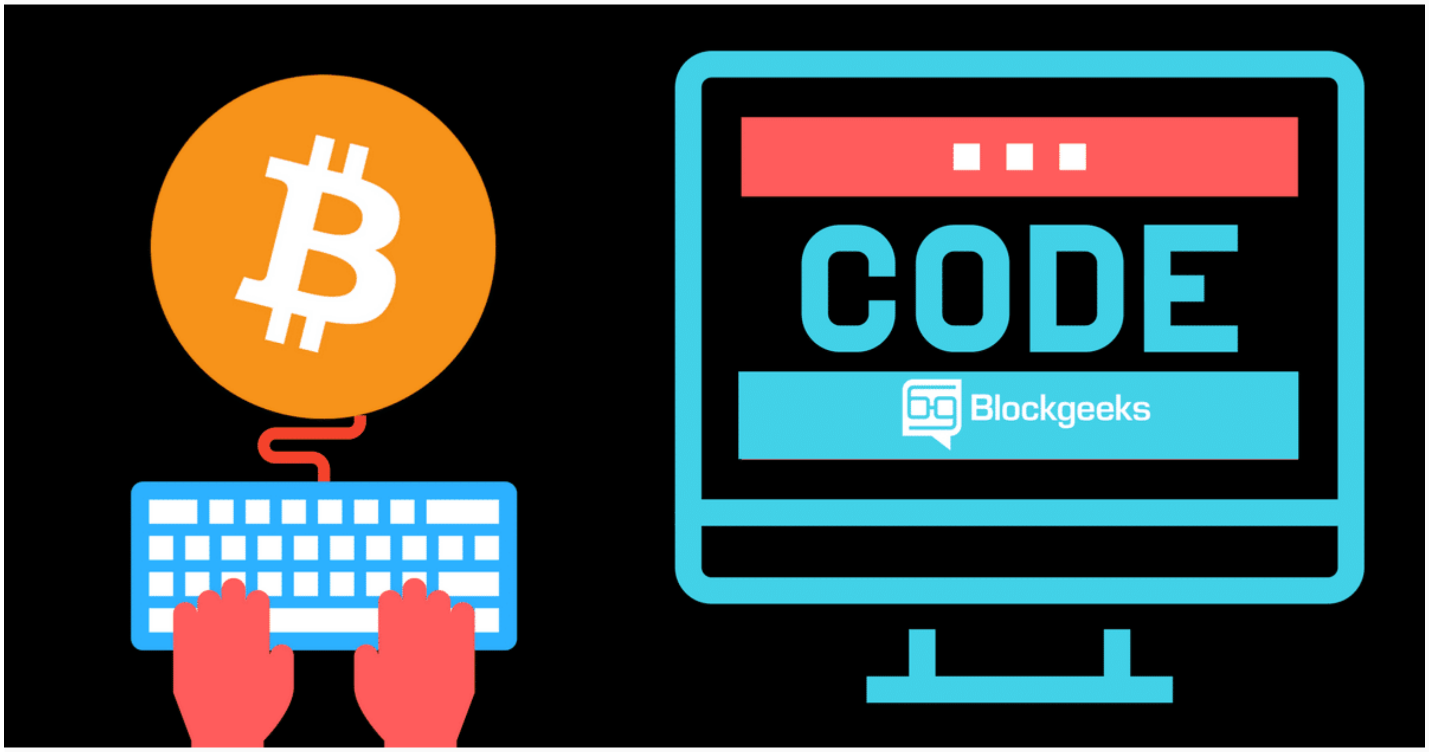


Image Credit: CoinMarketCap

What does it take to become a bitcoin developer?



In this guide we will aim to answer that question

So, what is bitcoin?

Bitcoin is a cryptocurrency that was conceptualized in 2009 by the mysterious Satoshi Nakamoto. It is a decentralized digital currency which works in a peer-to-peer system, utilizing the blockchain technology.If you want to become a bitcoin developer then the first thing that you need to do is to read the Bitcoin Whitepaper by Satoshi Nakamoto.

That whitepaper is essential reading for anyone who wants to get into the blockchain game. Not only did that white paper conceptualize the bitcoin, it also made us see how a byzantine fault tolerant system can function in a decentralized environment.

If you are keen on becoming a Bitcoin Developer, then it is important that you know how the blockchain works.

What is Blockchain?

The blockchain is a chain of blocks where each block contains data of value without any central supervision. It is cryptographically secure and immutable. A blockchain uses two important data structures: Pointers and Linked Lists.

Pointers

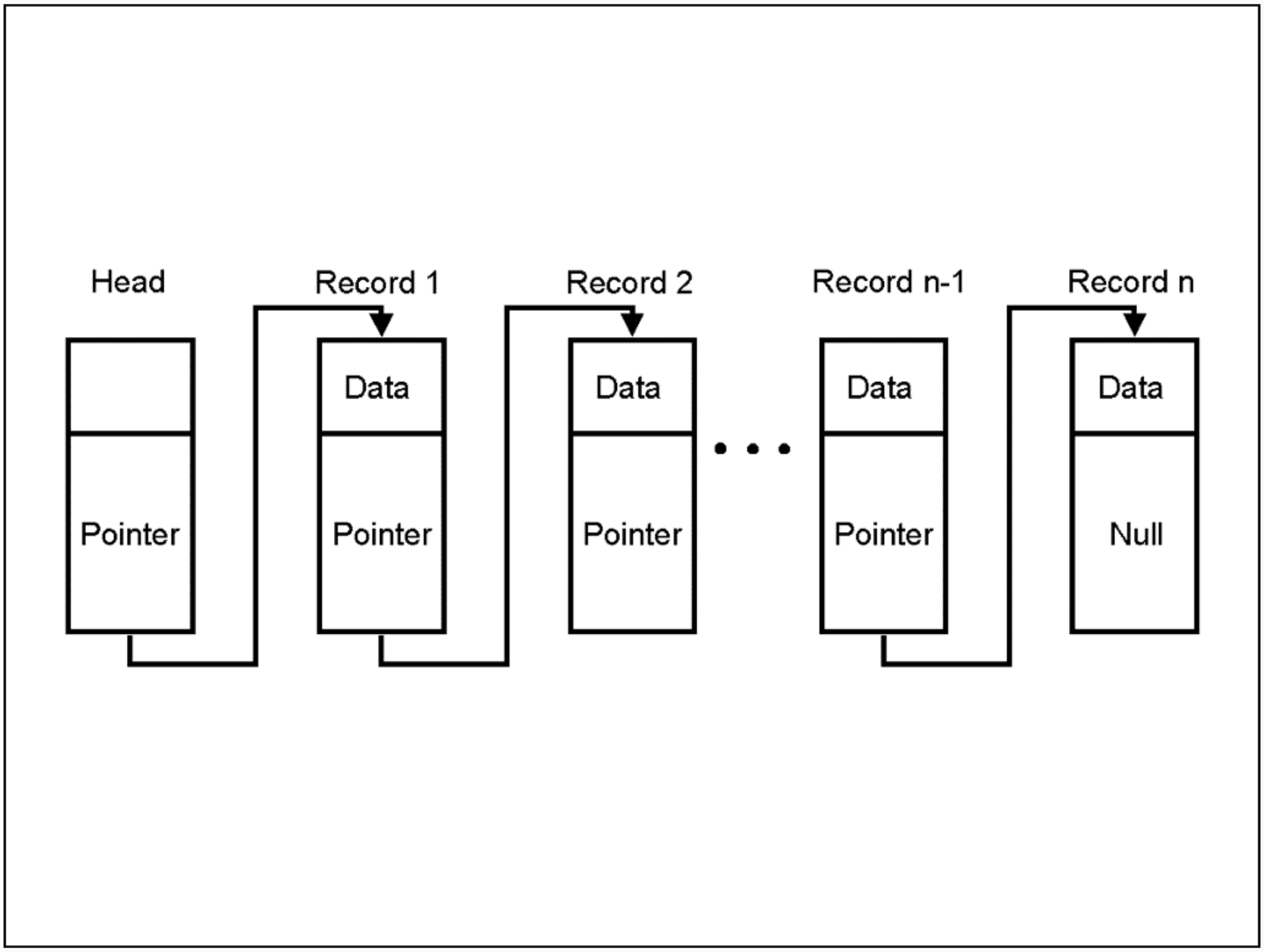
Pointers are variables in programming which stores the address of another variable. Usually normal variables in any programming language stores data.

> ***Eg. int a = 10, means that there is a variable "a" which stores integer values. In this case, it is storing an integer value which is 10. This is a normal variable.***

Pointers, however, instead of storing values will store addresses of other variables. Which is why they are called pointers, because they are literally pointing towards the location of other variables.

Linked Lists

A linked list is one of the most important items in data structures. This is what a linked list looks like:



It is a sequence of blocks, each containing data which is linked to the next block via a pointer. The pointer variable, in this case, contains the address of the next node in it and hence the connection is made. The last node, as you can see, has a null pointer which means that the pointer has no value.

One important thing to note here, the pointer inside each block contains the address of the next block. That is how the pointing is achieved. Now you might be asking what does that mean for the first block in the list? Where does the pointer of the first block stay?

The first block is called the “genesis block” and its pointer lies out in the system itself. It sort of looks like this:

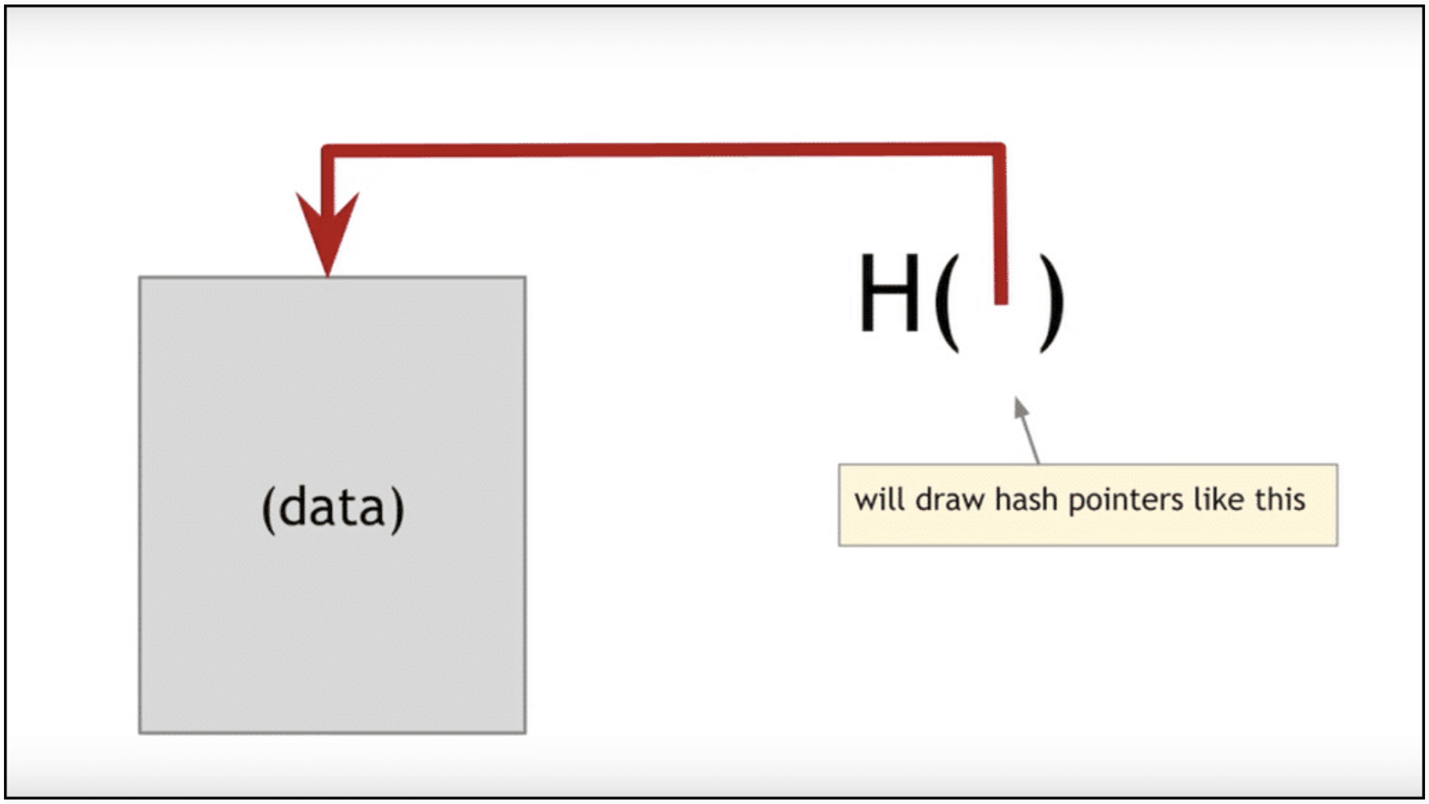
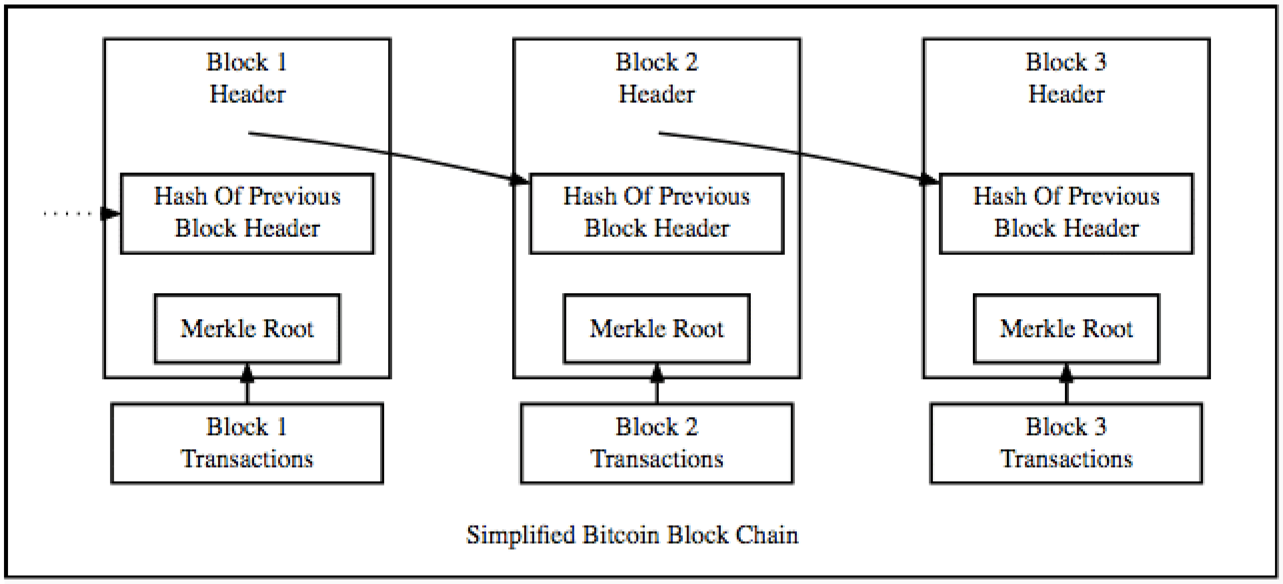


Image courtesy: Coursera

If you are wondering what the “hash pointer” means, it is a pointer which contains the hash of the previous block.

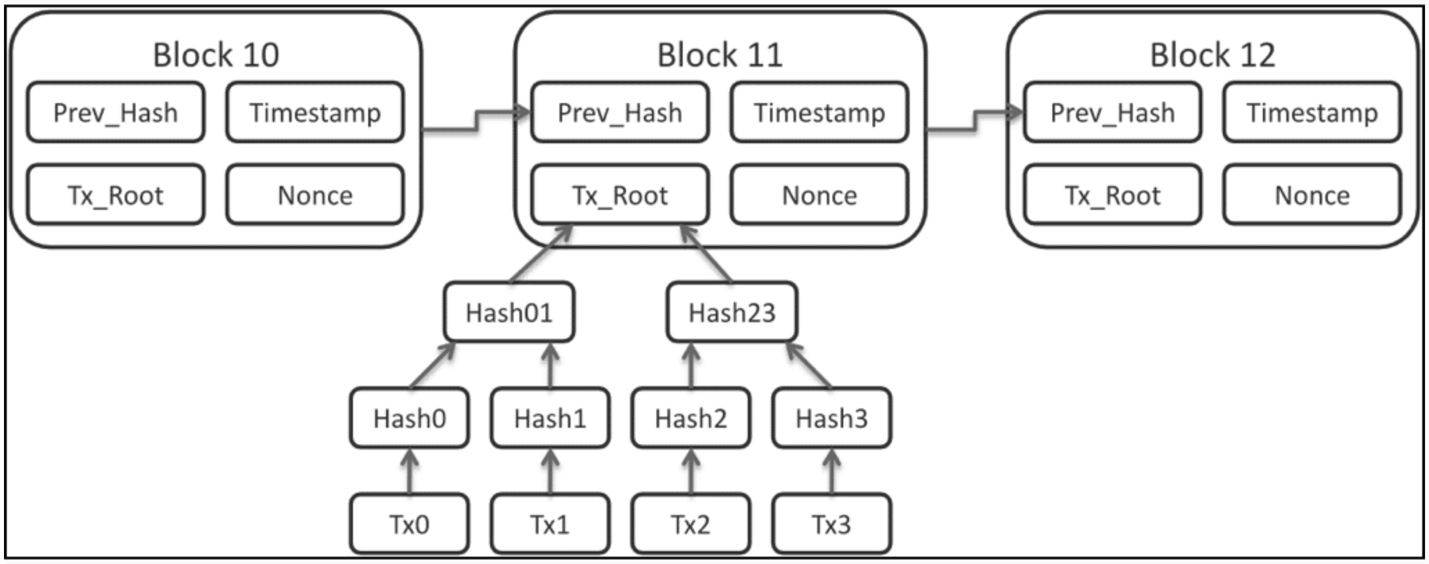
As you may have guessed by now, this is what the structure of the blockchain is based on. A block chain is basically a linked list and looks something like this:



The blockchain is a linked list which contains data and a hash pointer which points to its previous block, hence creating the chain. What is a hash pointer? A hash pointer is similar to a pointer, but instead of just containing the address of the previous block it also contains the hash of the data inside the previous block. This one small tweak, is what makes blockchains so amazingly reliable and trailblazing.

Imagine this for a second, a hacker attacks block 3 and tries to change the data. Because of the properties of hash functions, a slight change in data will change the hash drastically. This means that any slight changes made in block 3, will change the hash which is stored in block 2, now that in turn will change the data and the hash of block 2 which will result in changes in block 1 and so on and so forth. This will completely change the chain, which is impossible. This is exactly how blockchains attain immutability.

So what does a block header look like?



A block header contains:

- Version: The block version number.
- Time: the current timestamp.
- The current difficult target.

- Hash of the previous block.
- Nonce (more on this later).
- Hash of the Merkle Root.

What is mining?

"Mining" is how you produce new bitcoins. Mining works on the "proof-of-work" principle. Proof-of-work, basically means this: Solving a problem must be extremely difficult, but once you solve it, proving that the solution is correct should be simple.

We will see how bitcoin and most cryptocurrencies utilize it later. But for now, we must understand WHY proof-of-work was required in the first place.

One of the many problems that Nakamoto was facing was addressing the Byzantine Generals Problem. Every digital peer-to-peer decentralized currency system failed because they failed to answer the Byzantine Generals Problem. Nakamoto was finally able to answer this using proof-of-work.

So, what is Byzantine Generals Problem?

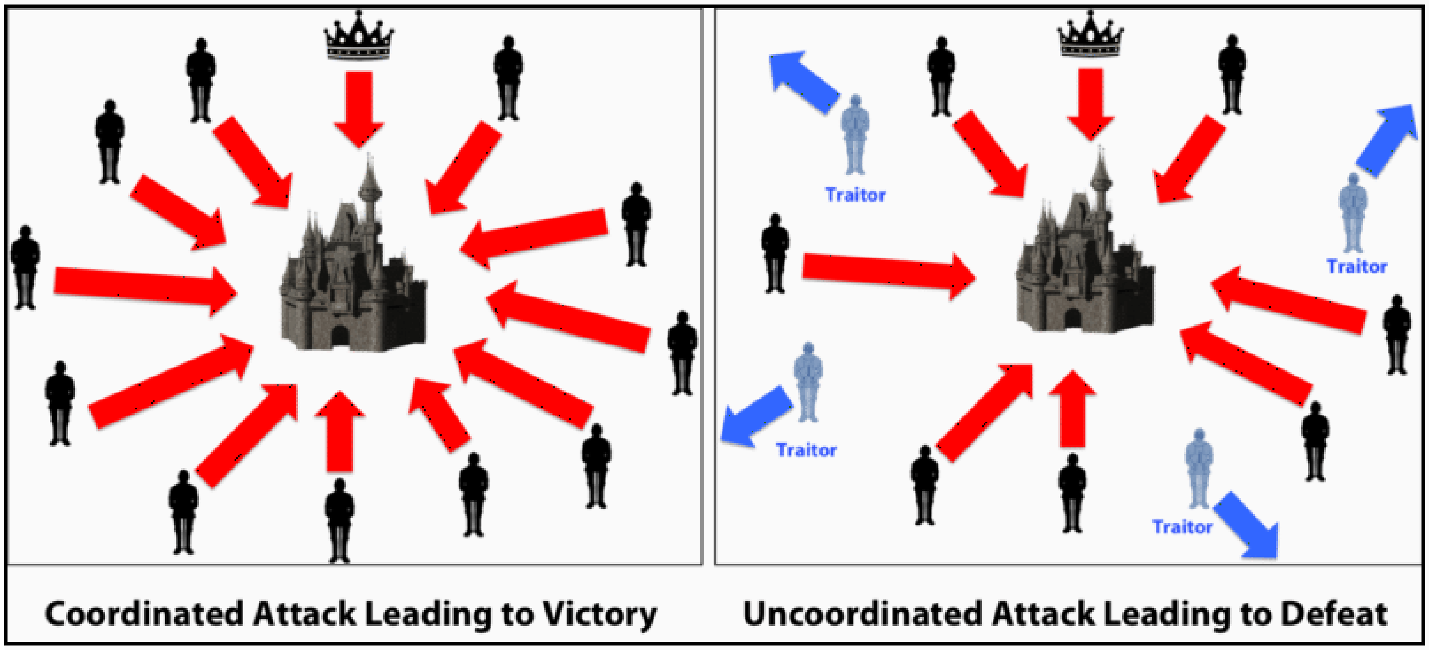

Traitor
Traitor
Traitor
Traitor
Coordinated Attack Leading to Victory
Uncoordinated Attack Leading to Defeat

Image Courtesy: Medium

Ok so imagine that there is a group of byzantine generals and they want to attack a city. They are facing two very distinct problems:

- The generals and their armies are very far apart so centralized authority is impossible, which makes coordinated attack very tough.
- The city has a huge army and the only way that they can win is if they all attack at once.

In order to make successful coordination the armies on the left of the castle send a messenger to the armies on the right of the castle with a message that says "ATTACK WEDNESDAY." However, suppose the armies on the right are not prepared for the attack and say, "NO. ATTACK FRIDAY" and send back the messenger through the city back to the armies on the left.

This is where we face a problem.

A number of things can happen to the poor messenger. He could get captured, compromised, killed and replaced with another messenger by the city. This would lead to the armies getting tampered information which may result in an uncoordinated attack and defeat.

This has clear references to blockchain as well. The chain is a huge network; how can you possibly trust them? If you were sending someone 4 Ether from your wallet, how would you know for sure that someone in the network isn't going to tamper with it and change 4 to 40 Ether?

Satoshi Nakamoto was able to bypass the Byzantine General's problem by inventing the proof of work protocol. This is how it works. Suppose the army on the left want to send a message called "ATTACK MONDAY" to the army on the right, they are going to follow certain steps.

- Firstly, they will append a “nonce” to the original text. The nonce can be any random hexadecimal value.

- After that, they hash the text appended with a nonce and see the result. Suppose, hypothetically speaking, the armies have decided to only share messages which, on hashing, gives a result which starts with 5 zeroes.

- If the hash conditions are satisfied, they will send the messenger with the hash of the message. If not, then they will keep on changing the value of the nonce randomly until they get the desired result. This action is extremely tedious and time consuming and takes a lot of computation power.

- If the messenger does get caught by the city and the message is tampered with, according to hash function properties, the hash itself will get drastically changed. If the generals on the right side, see that the hashed message is not starting with the required amount of 0s then they can simply call off the attack.

However, there is a possible loophole.

No hash function is 100% collision free. Collision resistance means this: Given two different inputs A and B where H(A) and H(B) are their respective hashes, it is infeasible for H(A) to be equal to H(B). What that means is that for the most part, each input will have its own unique hash. However, in practice, no hash function is 100% collision free.

So what if the city gets the message, tampers with it and then accordingly

So what if the city gets the message, tampers with it and then accordingly change the nonce until they get the desired result which has the required number of 0s? This will be extremely time consuming but it is still possible. To counter this, the generals are going to use strength in numbers.

Suppose, instead of just one general on the left sending messages to one general on the right, there are 3 generals on the left who have to send a message to the ones on the right. In order to do that, they can make their own message and then hash the cumulative message and then append a nonce to the resulting hash and hash it again. This time, they want a message which starts with six 0s.

Obviously, this is going to be extremely time consuming, but this time, if the messenger does get caught by the city, the amount of time that they will take to tamper the cumulative message and then find the corresponding nonce for the hash will be infinitely more. It may even take years. So, eg. if instead of one messenger, the generals send multiple messengers, by the time the city is even halfway through the computation process they will get attacked and destroyed.

The generals on the right have it pretty easy. All they have to do is to append the message with the correct nonce that will be given to them, hash them, and see whether the hash matches or not. Hashing a string is very easy to do. That in essence is the process behind proof-of-work.

- The process behind finding the nonce for the appropriate hash target should be extremely difficult and time consuming.
- However, the process of checking the result to see if no malpractice has been committed should be very simple.

So, that is how miners in bitcoin implement proof-of-work to do their mining.

They use their computational power to mine for blocks by solving cryptographic puzzles. One block in bitcoin is mined every 10 mins.

How do transactions work in Bitcoin?

Before we continue, a huge shoutout to Professor Donald J Patterson and his Youtube channel “djp3” for the explanation.

Suppose Alice wants to send a certain number of bitcoins to Bob. How does the transaction system in Bitcoin work? Bitcoin transactions are very different from Fiat wallet transactions. If Alice was to give $2 to Bob, she would physically take 2 dollars from her wallet and give it to Bob. However, things don’t work like that in Bitcoin. You don’t physically own any Bitcoin, what you have is the proof that you have Bitcoins.

There are two more things that you need to know:

- The miners validate your transactions by putting the data inside the mines that they have blocked. In return of giving this service they charge a transaction fee.

- When it comes to FIAT currency, you don’t really keep track of how and where you got that specific note from. Eg. Open your wallet right now and take out all the notes and coins in it. Can you tell where exactly did you get each and every specific note and coin from? Chances are that you don’t. However, in bitcoin, the history of each and every single bitcoin transaction is taken note of.

Ok, so now let’s do a deep dive into how a bitcoin transaction between Alice and Bob takes place. There are two sides to a transaction, the Input and the Output. This entire Transaction will have a name that we will figure out in the end. For now, let’s look at the dynamics.

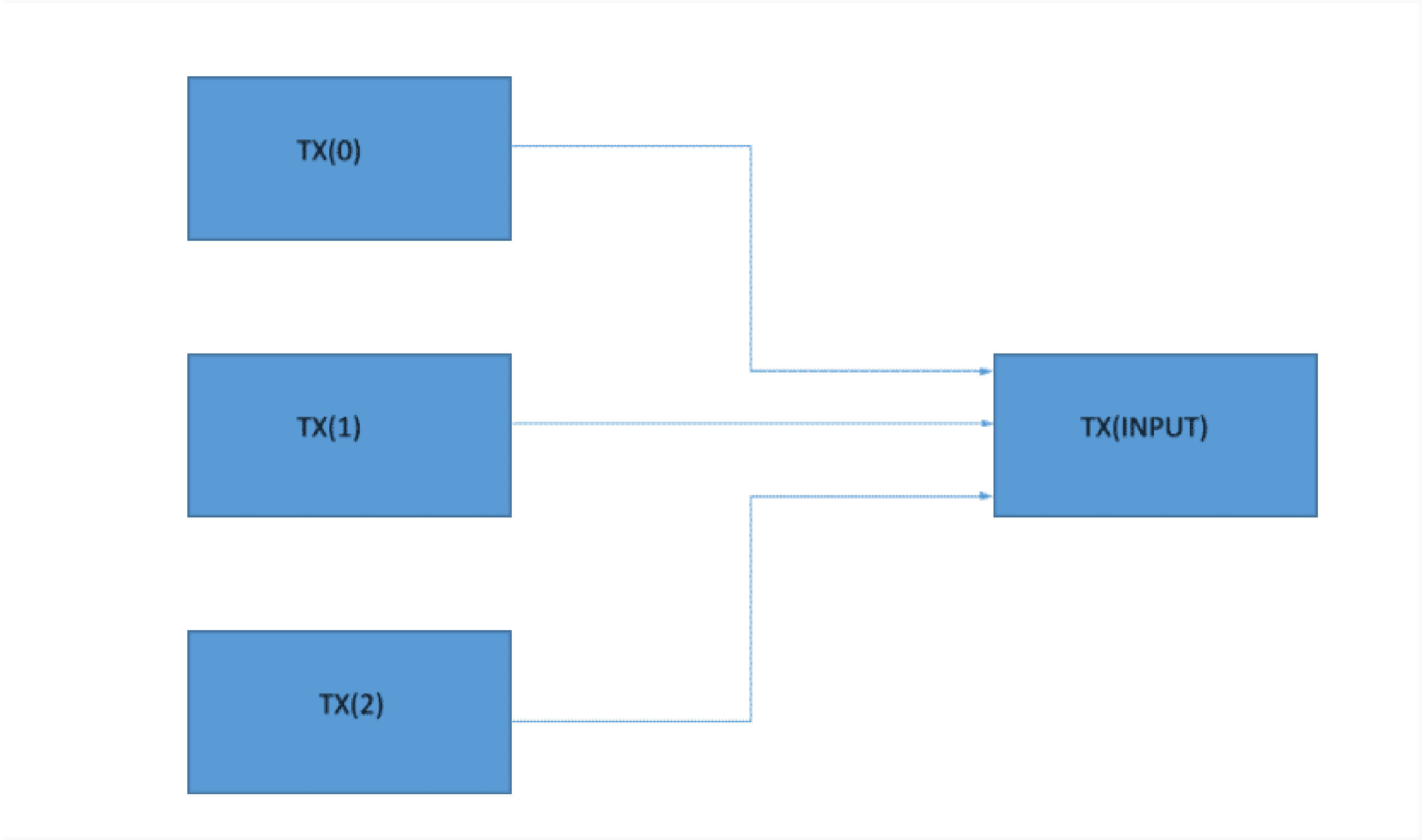
end. For now, let's look at the dynamics.

Transaction Input

In order to make this transaction happen, Alice needs to get bitcoins which she has received from various previous transactions. Remember, like we said before, in bitcoins, each and every coin is accounted for via a transaction history.

So, suppose Alice needs to pull bitcoins from the following transactions which we shall name TX(0), TX(1) and TX(2). These three transactions will be added together and that will give you the input transaction which we shall call TX(Input).

Diagrammatically, it will look like this:

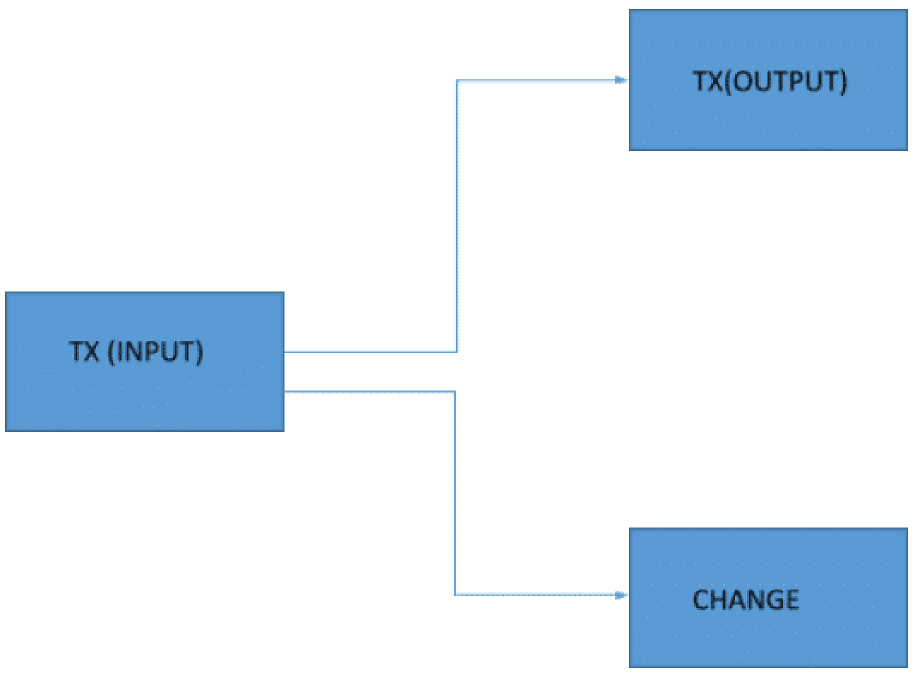


So, that is it from the input side, let's check out what the output side will look like.

Transaction Output

The output basically will have the amount of bitcoins that Bob will posses post transaction and any remaining change that is left over, which is then sent back to Alice. This change then becomes her input value for all future transactions.

A pictorial representation of the output side looks like this:



Now, this is a very simple transaction that has just one output (apart from the CHANGE), there are transactions that are possible with multiple outputs. This is what the basic layout of the transaction looks like. For this entire thing to go through, however, certain conditions must be met.

Conditions of a transaction

- TX(Input) > TX(output). The input transaction has to be always greater than the output transaction. In any transaction the deficit between the input and

the output (output+change) is the transaction fees that miners collect.
So:Transaction fees = TX(Input) – (TX(output) + Change).

- In the input side:TX(0) + TX(1) + TX(2) = TX(Input).If Alice doesn’t have the funds necessary to carry out the transactions then the miners will simply reject the transactions.

- Bob can will have to show that he can provide the proof needed to get the bitcoins. Alice will lock the transactions with Bob’ public address. He will need to produce his private key to unlock the transactions and gain access to his fees.

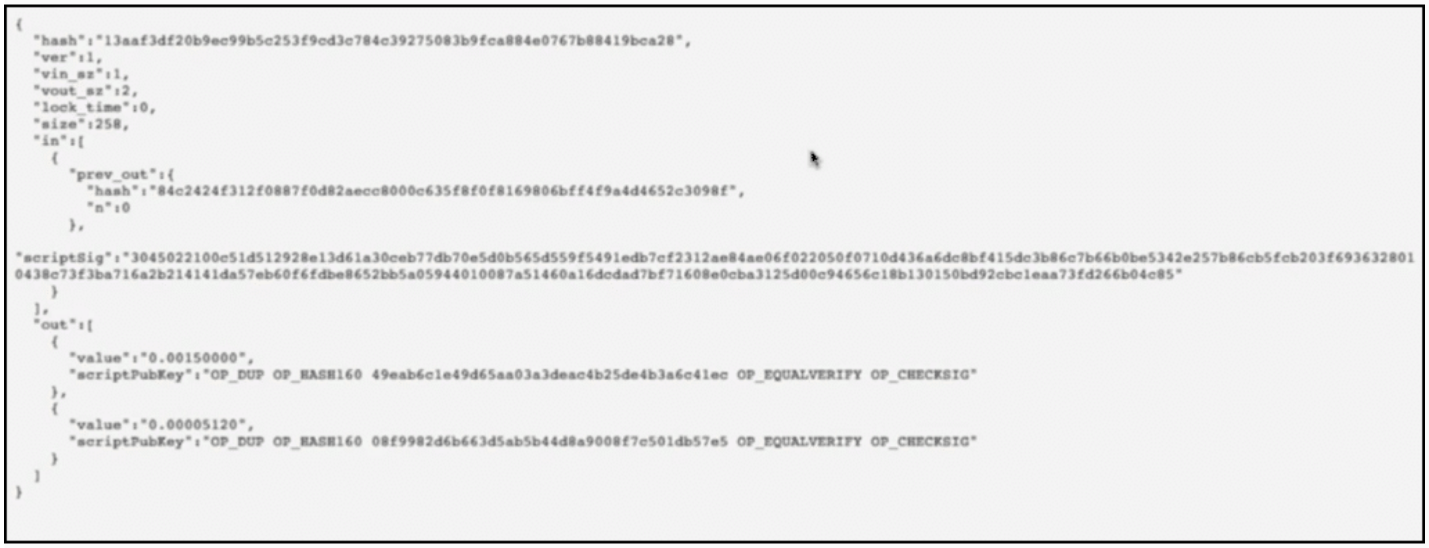
- Alice also needs to verify that she has the required rights to send over the bitcoins in the first place. The way she does that is by signing off the transaction with her digital signature (aka her private key). Anyone can decode this by using her public key and verify that it was indeed Alice who sent over the data. This proof is called “Signature data”. Remember this because this will be very important later on.

So, what is going to be the name of this entire transaction?

The Input (including the signature data) and the output data is added together and hashed using the SHA 256 hashing algorithm. The output hash is the name that is given to this transaction.

The transaction details code

This is what the transaction looks like in the code form aka script form. Suppose Alice wants to send 0.0015 BTC to Bob and in order to do so, she sends inputs which are worth 0.0015770 BTC. This is what the transaction detail looks like:



```
{
  "hash":"13aaf3df20b9ec99b5c253f9cd3c784c39275083b9fca884e0767b88419bca28",
  "ver":1,
  "vin_sz":1,
  "vout_sz":2,
  "lock_time":0,
  "size":258,
  "in":[
    {
      "prev_out":{
        "hash":"84c2424f312f0887f0d82aecc8000c635f8f0f8169806bff4f9a4d4652c3098f",
        "n":0
      },

"scriptSig":"3045022100c51d512928e13d61a30ceb77db70e5d0b565d559f5491edb7cf2312ae84ae06f022050f0710d436a6dc8bf415dc3b86c7b66b0be5342e257b86cb5fcb203f693632801
0438c73f3ba716a2b214141da57eb60f6fdbe8652bb5a05944010087a51460a16dcdad7bf71608e0cba3125d00c94656c18b130150bd92cbc1eaa73fd266b04c85"
    }
  ],
  "out":[
    {
      "value":"0.00150000",
      "scriptPubKey":"OP_DUP OP_HASH160 49eab6c1e49d65aa03a3deac4b25de4b3a6c41ec OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value":"0.00005120",
      "scriptPubKey":"OP_DUP OP_HASH160 08f9982d6b663d5ab5b44d8a9008f7c501db57e5 OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```

Image courtesy: djp3 youtube channel.

The first thing that you see:

Is the name of the Transaction aka the hash of the input and output value.

Vin_sz is the number of input data, since Alice is sending the data using only one of her previous transactions, it is 1.

Vout_sz is 2 because the only outputs are Bob and the change.

See the input data? Alice is only using one input transaction (in the example that we gave above, this will be TX(0)), this is the reason why vin_sz was 1.

- Below the input data is her signature data.
- Underneath all this is the output data:

The first part of the data signifies that Bob is getting 0.0015 BTC.

The second part signifies that 0.00005120 BTC is what Alice is getting back as change.

Now, remember that out input data was 0.0015770 BTC? This is greater than (0.0015 + 0.00005120). The deficit of these two values is the transaction fee that the miners are collecting.

Before we continue though, let’s discuss a special kind of transaction called Coinbase transaction. It is basically the first transaction data that is on the block, and it signifies the mining reward that miners get upon mining the block. As of right now the reward is 12.5 BTC. These transactions have no input data and they only have output data.

One thing you need to keep in mind, a transaction in Bitcoin goes through if and only if the miner, who has mined the block, physically puts in your transaction inside the block. A miner can become a temporary dictator of the block that they have mined. They can charge transaction fees for each and every transaction that they put inside the block.

As the number of transactions increased by leaps and bounds, the rate at which the blocks filled up increased as well. More often than not, people actually had to wait till new blocks were created so that their transactions would go through. This created a backlog of transactions, in fact the only way to get your transactions prioritized was to pay a high enough transaction fee to attract and incentivize the miners to prioritize your transactions.

This introduced the “replace-by-fee” system. Basically, this is how it works. Suppose Alice is sending 5 bitcoins to Bob, but the transaction is not going through because of a backlog. She can’t “delete” the transaction because bitcoins once spent can never come back. However, she can do another transaction of 5 bitcoins with Bob but this time with transaction fees which are high enough to incentivize the miners. As the miners put her transaction in the block, it will also overwrite the previous transaction and make it null and void.

Bitcoin Developer: Programming In Bitcoin

The original Bitcoin Core was coded by Satoshi Nakamoto using C++.

Before we begin, let's checkout some of the challenges that a blockchain developer faces. Creating and maintaining a public blockchain is not easy because of a number of reasons.

(Before we continue, a huge shoutout to David Schwartz for his keynote address regarding C++ use in blockchain software development in CPPCON 2016.)

- **Reason #1: Security**

Blockchains, as David Schwartz puts it, should be fortresses. Firstly, the code is public and open for all to see. Anyone can look over the code and check for bugs and vulnerabilities. However, unlike other open code resources, the downside of finding vulnerabilities on blockchain code is massive. Any programmer can hack in and get away with potentially millions and millions of dollars. Because of these legitimate security concerns, development on blockchain is usually very slow.

- **Reason #2: Resource Management**

It is important to keep pace with the network. You cannot fall too far behind and not keep up with all the network demands. You should be well equipped to handle remote and local queries.

- **Reason #3: Performance**

The blockchain must always perform at its highest possible capabilities, but for that to happen the language chosen must be extremely versatile. The thing is that there are certain tasks in the blockchain which are parallelizable whilst there are some tasks which can't be done in parallel.

A good example of "parallelizable" task is digital signature verification. All that you need for signature verification is the key, transaction and the signature. With just three data you can conduct verifications in a parallelized manner.

However, not all the functions on a blockchain should be done that way. Think of transaction execution itself. Multiple transactions can't be executed in parallel; it needs to be done one at a time to avoid errors like double spends. Some languages are good at parallel operations while some are good in non-parallel operations.

- **Reason #4: Isolation**

What is deterministic behavior?

If A + B = C, then no matter what the circumstances, A+B will always be equal to C. That is called deterministic behavior.

Hash functions are deterministic, meaning A's hash will always be H(A).

So, in blockchain development, all transaction operations must be deterministic. You cannot have a transaction that behaves one way and then behaves another way the next day. Similarly, you cannot have smart contracts that work in two different ways in two different machines.

The only solution to this is isolation. Basically you isolate your smart contracts and transactions from non-deterministic elements.

So, we have discussed the main problems that blockchain developers face. Now let's finally check out some of the languages that the developers can use to let's finally check out some of the languages that the developers can use to code on the blockchain.

Language #1: C++

First and foremost, let's start with the granddaddy of them all, the evergreen C++. C++ was created by Bjarne Stroustrup as an extension of the C language. The Language was designed to have the flexibility and efficiency of the C but with some major differences. The biggest difference between C and C++ is that while C is process-oriented, C++ is object oriented.

What this means is that, in C++, the data and functions are wrapped into one neat little package called "objects" which means that once an object is created, it can easily be called and reused in other programs, which greatly reduces coding time.

Let's look at the simplest C++ program in the world. The "Hello World" program:

```
#include <iostream>

using namespace std;

int main()

{

cout << "Hello, World!";

return 0;

}
using namespace std;
```

```
int main()

{

cout << "Hello, World!";

return 0;

}
```

This code will print: Hello World!

So, why do people still use C++ for coding? Surely there are way more glamorous languages now, why do people still insist on going back to C++? Why is the bitcoin blockchain coded on C++?

Well, as it happens, C++ has certain features that makes it very appealing. (Shout out Peter Wiulle and David Schwartz for the following explanation).

Feature #1: Memory Control

Remember what we said earlier about the challenges of blockchain development? Not only should blockchains be secured fortresses but they should have effective resource management as well. A blockchain is supposed to interact with a lot of untrusted endpoints while still giving quick service to any and all nodes.

This quick and prompt service is critical for the success of a cryptocurrency like bitcoin. Remember, they are all based on the principle of "consensus", all nodes on the network must accept and reject the exact same blocks, or else there could be a fork in the chain.

In order to satisfy all these demands and perform at the highest level, you need tight and complete control over CPU and memory usage. C++ gives that to its users.

Feature #2: Threading

As we have discussed before, one of the main challenges of the blockchain programming is the integration of tasks that parallelize well and the tasks that don't parallelize. Most languages specialize in one, however C++'s threading ability is good enough to handle both parallel and non-parallel tasks. A thread is a set of instructions that can be executed simultaneously. Not only does C++ allow fir superb multithreading facilities with effective inter-thread communication, it also optimizes single-thread performance.

Feature #3: Move Semantics

One of the most interesting aspects of C++ is move semantics. Move semantics provides a way for the contents to be moved between objects rather than be copied outright. Let's checkout the differences between copy semantics and move semantics. (Following data taken from Peter Alexander's answer in "Stackoverflow").

Copy Semantics:

- assert(b == c);
- a = b;

- assert(a == b && b == c);

So what is happening here? The value of b goes into a and b remains unchanged at the end of the whole thing.

Now, consider this.

Move Semantics:

- assert(b = = c);
- move (a,b);
- assert (a = =c);

What is happening here?

Can you see the difference between the two blocks of codes?

When we are using the move semantics, the value of "b" need not be the unchanged. That is the difference between copy semantics and move semantics. The biggest advantage of move semantics is that you can get copies of certain data only when you need them, which greatly decreases redundancy in the code and gives a huge performance boost. So as you can see, this efficient memory management and high performance are both desirable for the blockchain.

Feature #4: Compile Time Polymorphism

What is polymorphism?

Remember when we called C++ an "object oriented programming (OOP) language"? Polymorphism happens to be an OOP property. Using polymorphism, you use a particular feature in more than one ways. In C++ polymorphism can be used in two ways:

- Compile time polymorphism.
- Run time polymorphism.

Over here, we will only be focusing on compile time polymorphism. There are two ways that C++ implements compile time polymorphism:

- Function Overloading.
- Operator Overloading.

Function Overloading:

Function overloading is when you have many functions of the same name but with different parameter intake.

Consider this program:

```
#include <bits/stdc++.h>
using namespace std;

class A

{

void func (int x)  //first instance of the function takes only one integer value

{

cout<<x<<endl;

}

void func (double x) //second instance of the function takes only one double v

{

cout<<x<<endl;

}
}

void func (int x, int y) //third instance of the function takes two integer values

{

cout<<x=y<<endl;

}

}

int main()

{

A obj1 //making one object of the class A

//now we are going to call the functions

obj1.func(2);

obj1.func(2.65);

obj1.func(2,5);

return 0;

}
```

Now when you run this function the output will be:

- 2
- 2.65
- 7

So, as you can see, the same function func() was used in 3 different ways.

Operator Overloading:

In C++ the same operator can have more than one meaning.

Eg. "+" can be used both for mathematical addition and for concatenation

Eg: "+" can be used both for mathematical addition and for concatenation.

Concatenation basically means taking two strings and combining them as one.

> ***So 3+4 = 7.***
>
> ***AND***
>
> ***Block+geeks = Blockgeeks.***

The same operator, did two different functions, this is operator overloading.

The Compile time polymorphism helps a lot in blockchain development. It helps in putting responsibilities separately in various functions and, in turn, boosting the performance of the whole system.

Feature #5: Code Isolation

C++ has namespace features which can be imported from one program to another. Namespace helps in avoiding name collisions. Also, since C++ has classes, it can act as boundaries between various APIs and help in making clear separation.

A class in C++ is a user defined type or data structure declared with keyword class that has data and functions as its members. You can access the functions declared in the class by declaring objects of that particular class.

Feature #6: Maturity

The language is both mature and regularly updated. There are at least 3 solid compilers, as David Schwartz says, and the new features are aimed at solving real issues. Debuggers and analytical tools of all kinds are available for everything from performance profiling to automatic detection of issues of all kinds. This means the language is constantly growing to incorporate newer and better features.

Because of the above features, Satoshi Nakamoto chose C++ to be the base language of the bitcoin source code.

Using Bitcoin Wallets

If you want to become a Bitcoin developer, then you definitely need to to know how bitcoin wallets work.

Without a doubt, the safest way to store any cryptocurrency is using a paper wallet. By following a few pointers below, you can set one up entirely for free.This truly makes you the master of your investment, and if precautions are followed, there's no possibility of your private keys being known by anyone else.

Of course, this means that keeping a record of them is even more important. Losing private keys means you'll forfeit the entire contents of your paper wallet (but then again, that's true for every wallet out there.)

What is a paper wallet?

To keep it very simple, paper wallets are an offline cold storage method of saving cryptocurrency. It includes printing out your public and private keys in a piece of paper which you then store and save in a secure place. The keys are printed in the form of QR codes which you can scan in the future for all your transactions. The reason why it is so safe is because it gives complete control to you, the user. You do not need to worry about the well-being of a piece of hardware, nor do you have to worry about hackers or any piece of malware. You just need to take care of a piece of paper.

Setting up a paper wallet

Paper wallets are formed by using a program to randomly generate a public and private key. The keys will be unique, and the program that generates them is open source. Those with advanced knowledge of coding can check the backend of the program themselves for randomicity in results. What's more, we'll be generating our keys offline. This eradicates the exposure to online threats, and deleting the simple program after use will destroy any trace of them.

Don't worry if it sounds confusing, it's not. You'll need no specific knowledge of coding, or encryption. All you do need is a computer, an internet connection, something to record your keys on.

Anyway, let’s create our paper wallet. Follow these steps:

- Ensure your computer is entirely free from any form of malicious software. A brand-new computer would be ideal, but is often not feasible.
- Visit the page WalletGenerator.net.
- Download the zip file by clicking here:

- Once downloaded open the “index.html” file but before that make sure that your internet is off. This entire process is done to make sure that you wallet is hacker free.

- Now it is time to generate your wallet. Keep hovering over the highlighted text and it will generate more characters. Or if you want, you can manually type in random characters. Just keep doing it until the counter goes to “0”.

- The moment the count goes to zero your wallet will be generated.

- Print the page or make multiple copies of the numbers from it. (Important: Ensure printer is not connected to Wi-Fi at this point).

- Delete saved webpage. You can now safely reconnect to the internet.
- Store your private keys in their long term, private, secure home.

Now that you have your wallet, you can go to one of the exchanges to exchange your fiat currency for bitcoin. Some of the exchanges that you can use are:

- Bitfinex.
- Bitstamp.
- BTC-e.
- Coinbase.

You have to do your research and find out which exchanges work best in your area.

Being in the know

One of the most critical things that you must do in order to be a bitcoin developer is to be in the know.

There are many Developer Communities where you can join and interact with other developers.

Bitcoin Developer Conclusion

So, there you go.

This guide will give you a basic idea of what you need to do and learn in order to become a Bitcoin developer.

You can checkout our courses if you want to kickstart your developer career today.

Bitcoin, and cryptocurrency in general, has infinite possibilities in the future. We may very well be on the cusp of the next great era-defining protocol. Can blockchain technology be the next internet? Only time can tell.

However, what can be said without a doubt is that the sheer scope of bitcoin and other blockchain applications, knows no bounds.

Like what you read? Give us one like or share it to your friends and get +16

297

Have questions?

We have built an incredible

community of blockchain enthusiasts from every corner of the industry. If you have questions, we have answers!

ASK COMMUNITY

Like what you're reading?

Join our community and get access to over 50 free video lessons, workshops, and guides like this! No credit card needed!

GET STARTED

RELATED GUIDES

Stacks: Bringing unprecedented functionality to Bitcoin

Powpeg: The Most Secure, Permissionless and Uncensorable Bitcoin Peg

What is a Bitcoin ETF? [Everything You MUST Know]

Canadian Bitcoin Exchange Comparison [Recently updated List]

Blockchain Domain Name Systems: Web 3.0 Blockchain-Based Domains