Beginning Ethereum Smart Contracts Programming

With Examples in Python, Solidity, and JavaScript

—

Wei-Meng Lee

Apress®

Beginning Ethereum Smart Contracts Programming

With Examples in Python, Solidity, and JavaScript

Wei-Meng Lee

Apress®

Beginning Ethereum Smart Contracts Programming: With Examples in Python, Solidity, and JavaScript

Wei-Meng Lee
Ang Mo Kio, Singapore

ISBN-13 (pbk): 978-1-4842-5085-3 ISBN-13 (electronic): 978-1-4842-5086-0
https://doi.org/10.1007/978-1-4842-5086-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Joan Murray
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image Photo by Bryan Garces on Unsplash

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484250853. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

I dedicate this book with love to my dearest wife (Sze Wa)
and daughter (Chloe), who have to endure
my irregular work schedule, for their companionship
when I am trying to meet writing deadlines!

Table of Contents

About the Author

Wei-Meng Lee is the founder of Developer Learning Solutions, a technology company specializing in hands-on training of blockchain and other emerging technologies. He has many years of training expertise, and his courses emphasize a learn-by-doing approach. He is a master at making learning a new programming language or technology less intimidating and fun. He can be found speaking at conferences worldwide such as NDC, and he regularly contributes to online and print publications such as DevX.com, MobiForge.com, and CoDe Magazine. He is active on social media on his blog learn2develop.net, on Facebook at DeveloperLearningSolutions, on Twitter @weimenglee, and on LinkedIn at leeweimeng.

About the Technical Reviewer

Chaim Krause is a lover of computers, electronics, animals, and electronic music. He's tickled pink when he can combine two or more in some project. The vast majority of his knowledge is through self-learning. He jokes with everyone that the only difference between what he does at home and what he does at work is the logon he uses. As a lifelong learner, he is often frustrated with technical errors in documentation that waste valuable time and cause unnecessary stress. One of the reasons he works as the technical editor on books is to help others avoid those same pitfalls.

Acknowledgments

Writing a book is immensely exciting, but along with it comes long hours of hard work and responsibility, straining to get things done accurately and correctly. To make a book possible, a lot of unsung heroes work tirelessly behind the scenes.

For this, I would like to take this opportunity to thank a number of special people who made this book possible. First, I want to thank my acquisitions editor – Joan Murray, for giving me this opportunity. Thanks for attending my session at NDC Minnesota 2018, and for your trust in me!

Next, a huge thanks to Jill Balzano, my associate editor, who was always very patient with me, even though I have missed several of my deadlines for the book. Thanks, Jill, for your guidance. I could not finish the book without your encouragement and help!

Equally important is my technical editor – Chaim Krause. Chaim has been very eager-eyed editing and testing my code and never fails to let me know if things do not work the way I intended. Thanks for catching my errors and making the book a better read, Chaim!

Last, but not least, I want to thank my parents and my wife, Sze Wa, for all the support they have given me. They have selflessly adjusted their schedules to accommodate my busy schedule when I was working on this book. I love you all!

Introduction

Welcome to *Beginning Ethereum Smart Contracts Programming*!

This book is a quick guide to getting started with Ethereum Smart Contracts programming. It first starts off with a discussion on blockchain and the motivations behind it. You will learn what is a blockchain, how blocks in a blockchain are chained together, and how blocks get added to a blockchain. You will also understand how mining works and discover the various types of nodes in a blockchain network.

Once that is out of the way, we will dive into the Ethereum blockchain. You will learn how to use an Ethereum client (Geth) to connect to the Ethereum blockchain and perform transactions such as sending ethers to another account. You will also learn how to create private blockchain networks so that you can test them internally within your own network.

The next part of this book will discuss Smart Contracts programming, a unique feature of the Ethereum blockchain. Readers will be able to get jumpstarted on Smart Contracts programming without needing to wade through tons of documentation. The learn-by-doing approach of this book makes you productive in the shortest amount of time. By the end of this book, you would be able to write smart contracts, test them, deploy them, and create web applications to interact with them.

The last part of this book will touch on tokens, something that has taken the cryptocurrency market by storm. You would be able to create your own tokens and launch your own ICO and would be able to write token contracts that allow buyers to buy tokens using Ethers.

This book is for those who want to get started quickly with Ethereum Smart Contracts programming. Basic programming knowledge and an understanding of Python or JavaScript are recommended.

I hope you will enjoy working on the sample projects as much as I have enjoyed working on them!

CHAPTER 1

Understanding Blockchain

One of the hottest technologies of late is *Blockchain*. But what exactly is a blockchain? And how does it actually work? In this chapter, we will explore the concept of blockchain, how the concept was conceived, and what problems it aimed to solve. By the end of this chapter, the idea and motivation behind blockchain would be crystal clear.

Tip For the clearly impatient – A blockchain is a digital transaction of records that's arranged in chunks of data called blocks. These blocks link with one another through a cryptographic validation known as a hashing function. Linked together, these blocks form an unbroken chain – a blockchain. A blockchain is programmed to record not only financial transactions but virtually everything of value. Another name for blockchain is *distributed ledger*.

Hold on tight, as I'm going to discuss a lot of concepts in this chapter. But if you follow along closely, you'll understand the concepts of blockchain and be on your way to creating some really creative applications on the Ethereum blockchain in the upcoming chapters!

Tip Ethereum is an open-source public blockchain that is similar to the Bitcoin network. Besides offering a cryptocurrency known as Ether (which is similar to Bitcoin), the main difference between Bitcoin and itself is that it offers a programming platform on top of the blockchain, called Smart Contract. This book focuses on the Ethereum blockchain and Smart Contract.

W.-M. Lee, *Beginning Ethereum Smart Contracts Programming*,
https://doi.org/10.1007/978-1-4842-5086-0_1

Motivations Behind Blockchain

Most people have heard of cryptocurrencies, or at least, Bitcoin.

Note The technology behind cryptocurrencies is blockchain.

To understand why we need cryptocurrencies, you have to first start with understanding a fundamental concept – *trust*. Today, any asset of value or transaction is recorded by a third party, such as bank, government, or company. We trust banks won't steal our money, and they are regulated by the government. And even If the banks fail, it is backed by the government. We also trust our credit card companies – sellers trust credit card companies to pay them the money, and buyers trust credit card companies to settle any disputes with the sellers.

Placement of Trusts

All these boil down to one key concept – placement of trust. And that is, we place our trust on a central body. Think about it, in our everyday life, we place our trusts on banks, and we place our trusts on our governments.

Even for simple mundane day-to-day activities, we place our trusts in central bodies. For example, when you go to the library to borrow a book, you trust that the library would maintain a proper record of the books that you have borrowed and returned.

The key theme is that we trust institutions but don't trust each other. We trust our government, banks, even our library, but we just don't trust each other. As an example, consider the following scenario. Imagine you work at a cafe, and someone walks up to you and offers you a US ten-dollar bill for two cups of coffee. And another person who offers to pay you for the two cups of coffee using a handwritten note saying he owes you ten dollars. Which one would you trust? The answer is pretty obvious, isn't it? Naturally you would trust the US ten-dollar bill, as opposed to the handwritten note. This is because you understand that using the ten-dollar bill, you can use it elsewhere to exchange for other goods or services, and that it is backed by the US government. In contract, the handwritten note is not backed by anyone else (except perhaps the person who wrote it), and hence it has literally no value.

Now let's take the discussion a bit further. Again, imagine you are trying to sell something. Someone comes up to you and suggests paying for your goods using the currencies as shown in Figure 1-1.

Figure 1-1. *Currencies from two countries*

Would you accept the currencies as shown in the figure? Here, you have two different currencies – one from Venezuela and one from Zimbabwe. In this case, the first thing you consider is whether these currencies are widely accepted and also your trust in these governments. You might have read from the news about the hyperinflation in these two countries, and that these currencies might not retain its value over time.

And so, would you accept these currencies as payment?

Trust Issues

Earlier on, I mentioned that people trust institutions and don't trust each other. But even established economies can fail, such as in the case of the financial crisis of the United States in 2007–2008. Investment bank Lehman Brothers collapsed in September 2008 because of the subprime mortgage market. So, if banks from established economies can

collapse, how can people in less developed countries trust their banks and governments? Even if the banks are trusted, your deposits may be monitored by the government, and they could arrest you based on your transactions.

As we have seen in the example in the previous section, there are times when people don't trust institutions, especially if the political situation in that country is not stable.

All these discussions bring us to the next key issue – even though people trust institutions, institutions can still fail. And when people lose trust in institutions, people turn to *cryptocurrencies*. In the next section, we will discuss how we can solve the trust issues using *decentralization*, a fundamental concept behind cryptocurrency.

Solving Trust Issues Using Decentralization

Now that you have seen the challenges of trust – who to trust and who not to trust, it is now time to consider a way to solve the trust issues. In particular, blockchain uses decentralization to solve the trust issue.

In order to understand decentralization, let's use a very simple example that is based on our daily lives.

Example of Decentralization

To understand how decentralization solves the trust issue, let's consider a real-life example.

Imagine a situation where you have three persons with DVDs that they want to share with one another (see Figure 1-2).

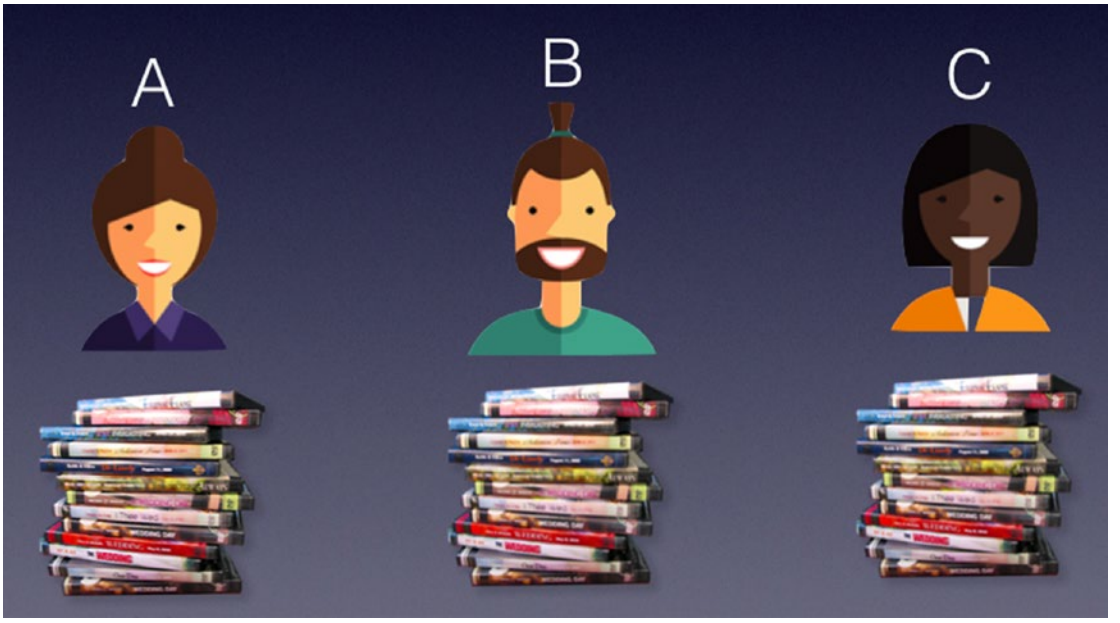

Figure 1-2. *Sharing DVDs among a group of people*

The first thing they need to do is to have someone keep track of the whereabouts of each DVD. Of course, the easiest is for each person to keep track of what they have borrowed and what they have lent, but since people inherently do not trust each other, this approach is not very popular among the three persons.

To solve this issue, they decided to appoint one person, say B, to keep a ledger, to hold a record of the whereabouts of each DVD (see Figure 1-3).

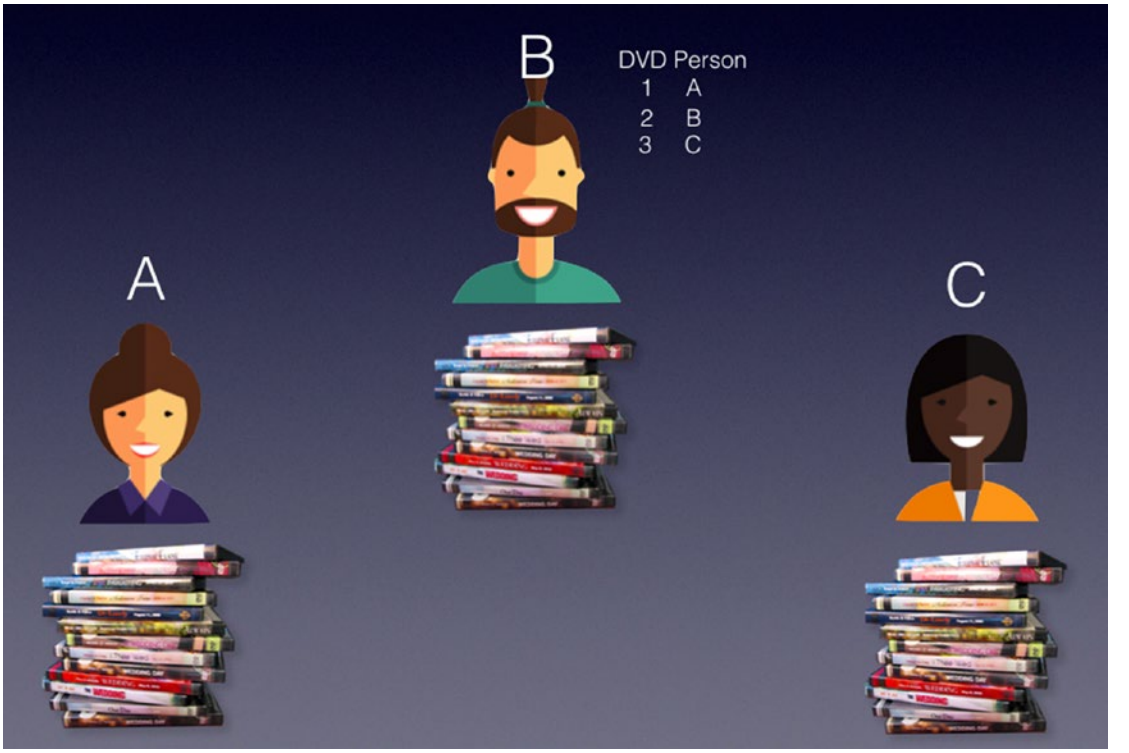

Figure 1-3. *Appointing a particular person to keep the records*

This way, there is a central body to keep track of the whereabouts of each DVD. But wait, isn't this the problem with centralization? What happens if B is not trustworthy? Turns out that B has the habit of stealing DVDs, and he in fact could easily modify the ledger to erase the record of DVDs that he has borrowed. So, there must be a better way.

And then, someone has an idea! Why not let everyone keep a copy of the ledger (see Figure 1-4)? Whenever someone borrows or lent a DVD, the record is broadcast to everyone, and everyone records the transaction.

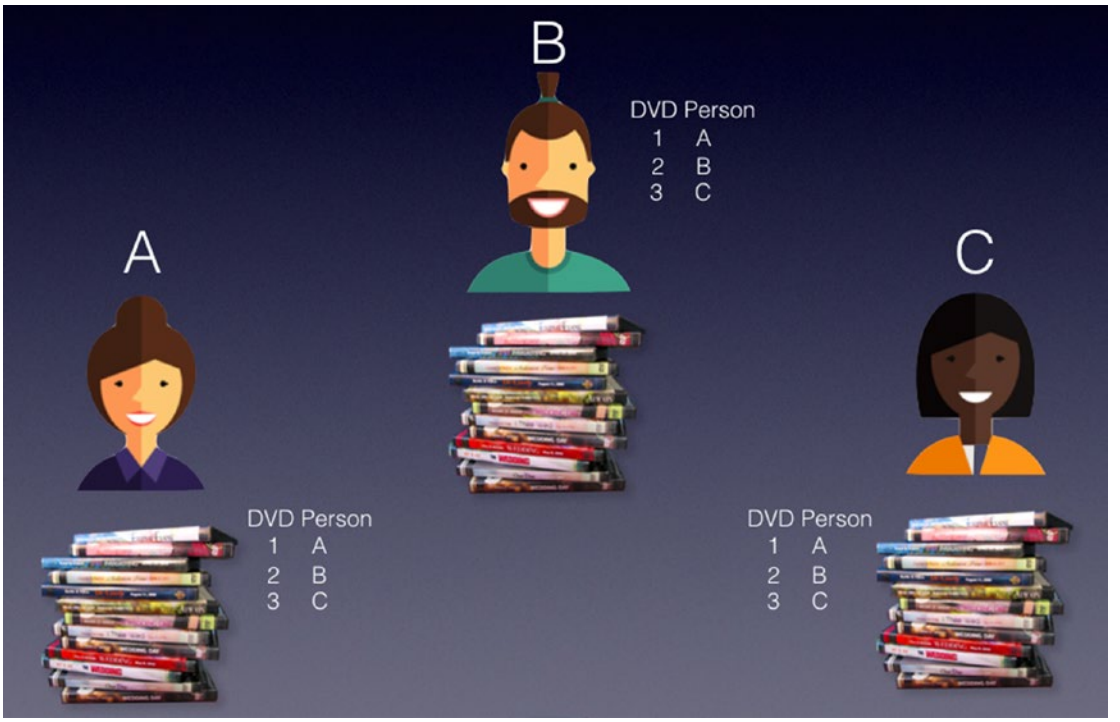

Figure 1-4. *Getting everyone to keep the records*

We say that the record keeping is now decentralized! We now have three persons holding the same ledger. But wait a minute. What if A and C conspire to change the records together so that they can steal the DVDs from B? Since majority wins, as long as there is more than 50% of the people with the same records, the others would have to listen to the majority. And because there are only three persons in this scenario, it is extremely easy to get more than 50% of the people to conspire.

The solution is to have a lot more people to hold the ledger, especially people who are not related to the DVDs sharing business (see Figure 1-5).

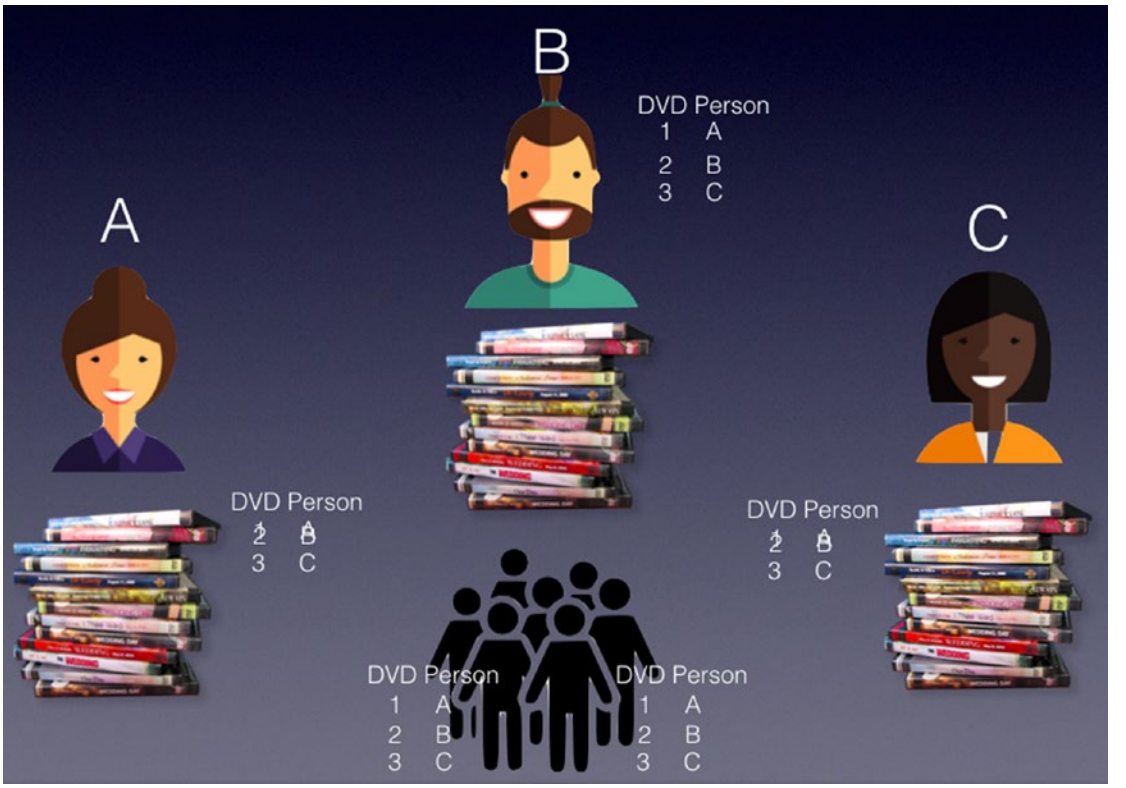

Figure 1-5. *Getting a group of unrelated people to help keep the records*

This way, it makes it more difficult for one party to alter the records on the ledger, and that in order to alter a record, it would need to involve a number of people altering the record all at the same time, which is a time-consuming affair. And this is the key idea behind *distributed ledger*, or commonly known as blockchain.

Blockchain As a Distributed Ledger

Now that we have a better idea of a distributed ledger, we can now associate it with the term – blockchain. Using the DVD rental example, each time a DVD is borrowed or returned, a transaction is created. A number of transactions are then grouped into a block. As more transactions are performed, the blocks are linked together cryptographically, forming what we now call a blockchain (see Figure 1-6).

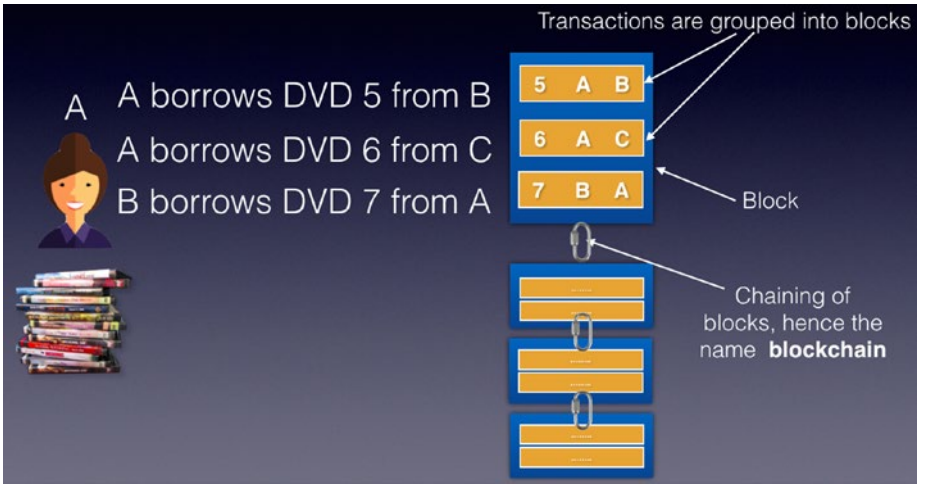

Figure 1-6. *Transactions form a block, and then blocks are then chained*

Based on what we have discussed, we can now summarize a few important points:

- Centralized databases and institutions work when there is trust in the system of law, governments, regulatory bodies, and people.
- A decentralized database built on the blockchain removes the need for the trust in a central body.
- A blockchain can be used for anything of value, not just currencies.

How Blockchain Works

At a very high level, a blockchain consists of a number of blocks. Each block contains a list of transactions, as well as a timestamp (see Figure 1-7).

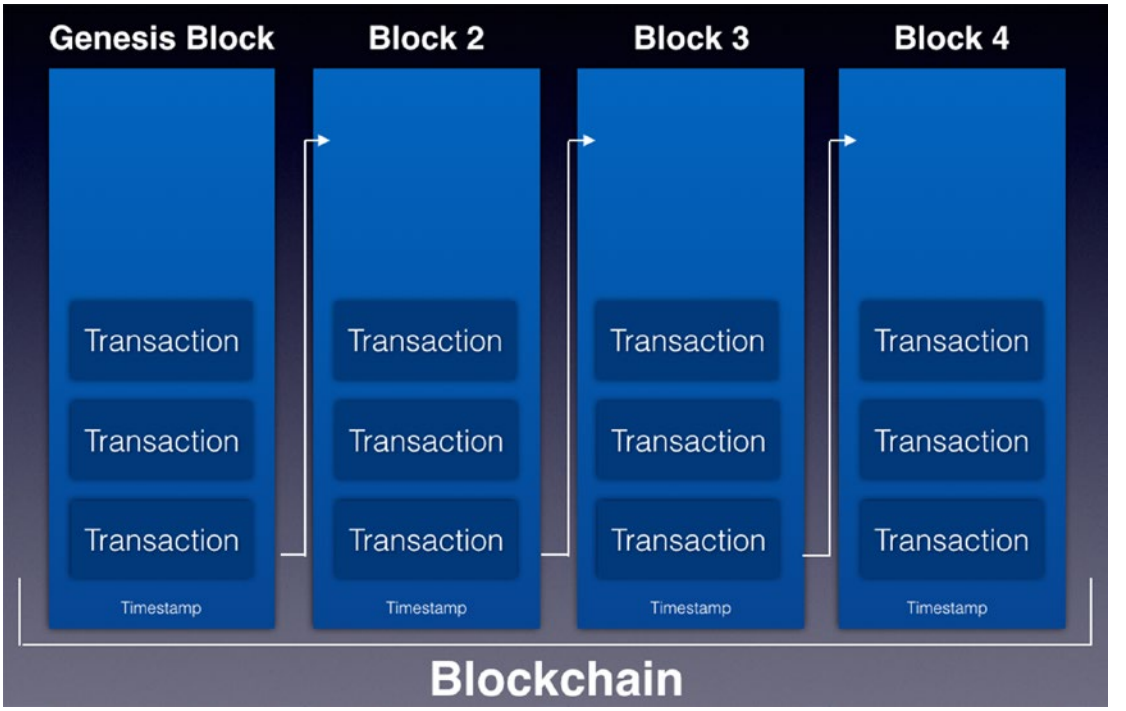

Figure 1-7. *Every blockchain has a beginning block known as the genesis block*

The blocks are connected to each other cryptographically, the details in which we will discuss in the sections ahead. The first block in a blockchain is known as the *genesis* block.

Note Every blockchain has a genesis block.

So, the next important questions is – how do you chain the blocks together?

Chaining the Blocks

Before we discuss how blocks in a blockchain are chained together, we have to discuss a key concept in blockchain – hashing. A hash function is a function that maps data of arbitrary size to data of fixed size. By altering a single character in the original string, the resultant hash value is totally different from the previous one. Most importantly, observe that a single change in the original message results in a completely different hash, making it difficult to know that the two original messages are similar.

A hash function has the following characteristics:

- It is deterministic – the same message always results in the same hash.
- It is a one-way process – when you hash a string, it is computationally hard to reverse a hash to its original message.
- It is collision resistant – it is hard to find two different input messages that hash to the same hash.

We are now ready to discuss how blocks in a blockchain are chained together. To chain the blocks together, the content of each block is hashed and then stored in the next block (see Figure 1-8). That way, if any transactions in a block is altered, that is going to invalidate the hash of the current block, which is stored in the next block, which in turn is going to invalidate the hash of the next block, and so on.

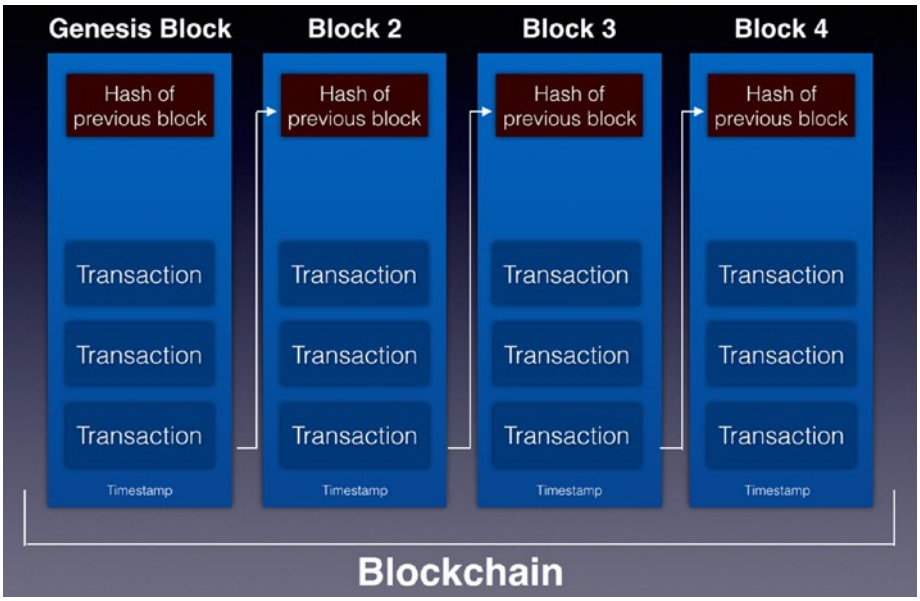

Figure 1-8. *Chaining the blocks with hashes*

Observe that when hashing the content of a block, the hash of the previous block is hashed together with the transactions. However, do take note that this is a simplification of what is in a block. Later on, we will dive into the details of a block and see exactly how transactions are represented in a block.

Storing the hash of the previous block in the current block assures the integrity of the transactions in the previous block. Any modifications to the transaction(s) within a block causes the hash in the next block to be invalidated, and it also affects the subsequent blocks in the blockchain. If a hacker wants to modify a transaction, not only must he modify the transaction in a block but all other subsequent blocks in the blockchain. In addition, he needs to synchronize the changes to all other computers on the network, which is a computationally expensive task to do. Hence, data stored in the blockchain is immutable, for they are hard to change once the block they are in is added to the blockchain.

Up to this point, you have a high-level overview of what constitutes a blockchain and how the blocks are chained together. In the next section, you will understand the next important topic in blockchain – mining.

Mining

Whenever you talk about blockchain or cryptocurrencies, there is always one term that comes up – mining. In this section, you will learn what is mining, and what goes on behind the scene.

Mining is the process of adding blocks to a blockchain. In a blockchain network, such as the Bitcoin or Ethereum network, there are different types of computers known as nodes. Computers on a blockchain that add blocks to the blockchain are known as *miner nodes (*or *mining nodes*, or more simply *miners)*.

We will talk about the different types of nodes later on in this course, but for now, we want to talk about a particular type of node, known as the miner node. The role of the miner node is to add blocks to the blockchain.

But how are blocks added?

Broadcasting Transactions

When a transaction is performed, the transaction is broadcasted to the network (see Figure 1-9).

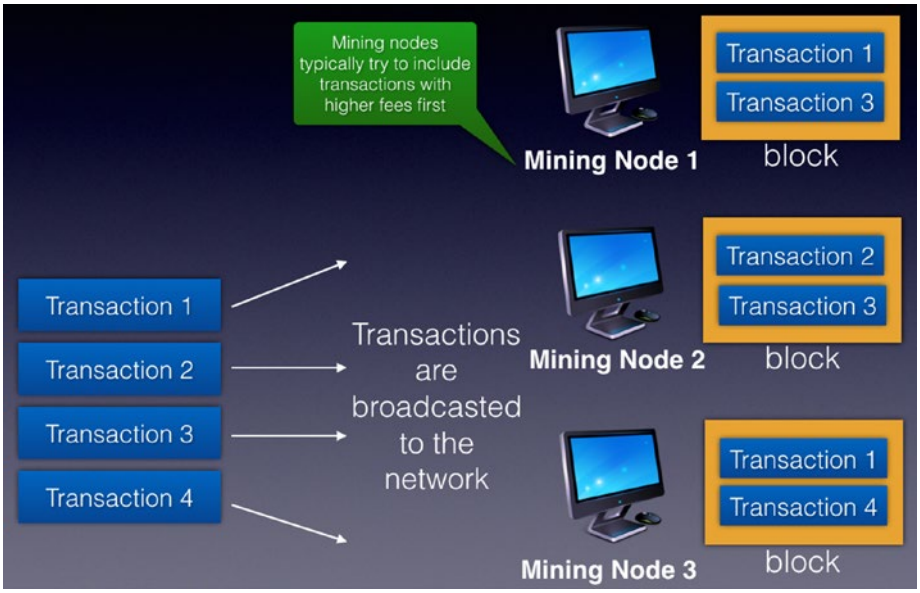

Figure 1-9. *Transactions are broadcasted to mining nodes, which then assemble them into blocks to be mined*

Each mining node may receive them at different times. As a node receives transactions, it will try to include them in a block. Observe that each node is free to include whatever transactions they want in a block. In practice, which transactions get included in a block depends on a number of factors, such as transaction fees, transaction size, order of arrival, and so on.

At this point, transactions that are included in a block but which are not yet added to the blockchain are known as *unconfirmed transactions*. Once a block is filled with transactions, a node will attempt to add the block to the blockchain.

Now here comes the problem – with so many miners out there, who gets to add the block to the blockchain first?

The Mining Process

In order to slow down the rate of adding blocks to the blockchain, the *blockchain consensus protocol* dictates a network difficulty target (see Figure 1-10).

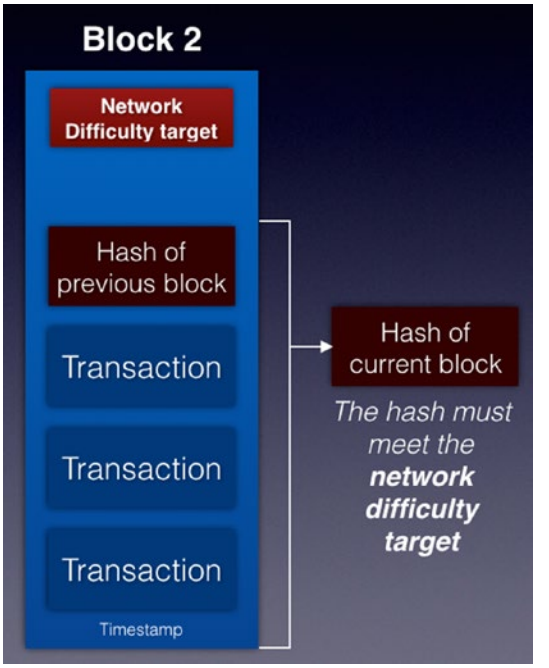

Figure 1-10. *Hashing the block to meet the network difficulty target*

In order to successfully add a block to the blockchain, a miner would hash the content of a block and check that the hash meets the criteria set by the difficulty target. For example, the resultant hash must start with five zeros and so on.

As more miners join the network, the difficultly level increases, for example, the hash must now start with six zeros and so on. This allows the blocks to be added to the blockchain at a consistent rate.

But, wait a minute, the content of a block is fixed, and so no matter how you hash it, the resultant hash is always the same. So how do you ensure that the resultant hash can meet the difficulty target? To do that, miners add a nonce to the block, which stands for *number used once* (see Figure 1-11).

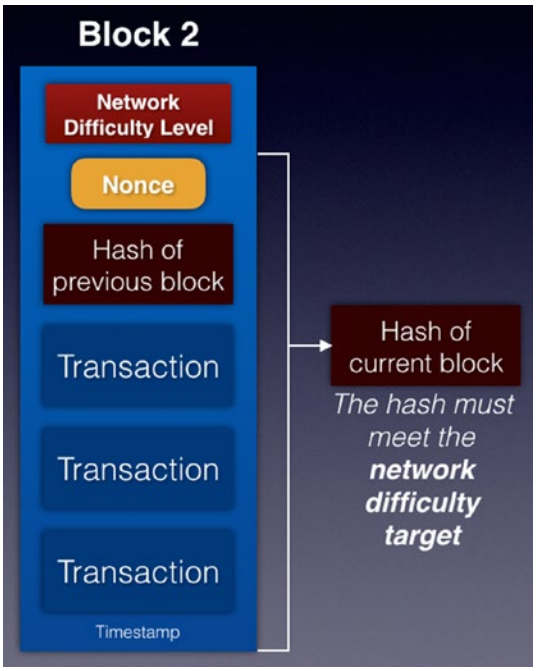

Figure 1-11. *Adding a nonce to change the content of the block in order to meet the network difficulty target*

The first miner who meets the target gets to claim the rewards and adds the block to the blockchain. It will broadcast the block to other nodes so that they can verify the claim and stop working on their current work of mining their own blocks. The miners would drop their current work, and the process of mining a new block starts all over again. The transactions that were not included in the block that was successfully mined will be added to the next block to be mined.

REWARDS FOR MINERS

In the case of Bitcoin, the block reward initially was 50 BTC and will halve every 210,000 blocks. At the time of writing, the block reward is currently at 12.5 BTC, and it will eventually be reduced to 0 after 64 halving events. For Ethereum, the reward for mining a block is currently 2 ETH (Ether).

BLOCKS ADDING RATES

For Bitcoin, the network adjusts the difficulty of the puzzles so that a new block is being mined roughly every 10 minutes. For Ethereum, a block is mined approximately every 14 seconds.

Proof of Work

The process in which blocks are mined and added to the blockchain is known as the **Proof of Work** (**PoW**). It is difficult to produce the proof but very easy to validate. A good example of Proof of Work is cracking a combination lock – it takes a lot of time to find the right combination, but it is easy to verify once the combination is found.

Proof of Work uses tremendous computing resources – GPUs are required, while CPU speed is not important. It also uses a lot of electricity, because miners are doing the same work repeatedly – find the nonce to meet the network difficulty for the block.

A common question is why you need to use a powerful GPU instead of CPU for mining? Well, as a simple comparison, a CPU core can execute 4 32-bit instructions per clock, whereas a GPU like the Radeon HD 5970 can execute 3200 32-bit instructions per clock. In short, the CPU excels at doing complex manipulations to a small set of data, whereas the GPU excels at doing simple manipulations to a large set of data. And since mining is all about performing hashing and finding the nonce, it is a highly repetitive task, something that GPU excels in.

Tip When a miner has successfully mined a block, he earns mining fees as well as transaction fees. That's what keeps miners motivated to invest in mining rigs and keep them running 24/7, thereby incurring substantial electricity bills.

Immutability of Blockchains

In a blockchain, each block is chained to its previous block through the use of a cryptographic hash. A block's identity changes if the parent's identity changes. This in turn causes the current block's children to change, which affects the grandchildren, and so on. A change to a block forces a recalculation of all subsequent blocks, which requires enormous computation power. This makes the blockchain immutable, a key feature of cryptocurrencies like Bitcoin and Ethereum.