packt

1ST EDITION

Rust for Blockchain Application Development

Learn to build decentralized applications on popular blockchain technologies using Rust

AKHIL SHARMA

Rust for Blockchain Application Development

Learn to build decentralized applications on popular blockchain technologies using Rust

Akhil Sharma

‹packt›

Rust for Blockchain Application Development

Group Product Manager: Kaustubh Manglurkar
Publishing Product Manager: Arindam Majumder
Book Project Manager: Hemangi Lotlikar
Senior Editor: Vandita Grover
Technical Editor: Kavyashree K S
Copy Editor: Safis Editing
Proofreader: Safis Editing
Indexer: Rekha Nair
Production Designer: Shankar Kalbhor
Senior DevRel Marketing Executive: Nivedita Singh

First published: April 2024
Production reference: 1290324

Published by
Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK.

ISBN 978-1-83763-464-4

www.packtpub.com

To the women in my life – my mother, Manisha Sharma, for all her sacrifices; my sister, Neha Sharma, for being a constant source of encouragement; and my wife, Akanksha, for being extremely supportive throughout this journey.

– Akhil Sharma

Contributors

About the author

Akhil Sharma is the founder of Armur AI, a cybersecurity company that is backed by Techstars, Outlier Ventures, and Aptos, and is part of the Google AI startups cloud program.

Akhil teaches advanced engineering topics (Rust, Go, Blockchain, and AI) on his YouTube channel and has mentored more than 200,000 engineers across platforms such as Linkedin Learning, Udemy, and Packt.

Being deeply involved with multiple Rust-based blockchain communities such as Aptos, Solana, and Polkadot inspired Akhil to write this book.

In his free time, Akhil likes to train in jiu jitsu, play the guitar, and surf.

About the reviewers

Denis Cavalli is a lead software engineer with a strong background in embedded systems, software development, and R&D. He graduated in computer engineering from the Universidade Federal do Amazonas in Brazil, and has more than 10 years of experience in software development and team leadership, working for start-ups and big companies.

Since 2021, he has been engaged with the Web3 environment, experimented with Ethereum/Solidity and Solana, worked professionally for Web3 companies using the Helium SDK, designed decentralized solutions targeted for Polkadot/Kusama networks using Substrate, and has had smart contracts deployed on the Arbitrum Nova mainnet.

Ryu Kent is a senior blockchain engineer who has worked in the industry for 7 years. He is particularly active in the DAO space and has launched a number of well-known smart contracts. Prior to moving to Web3, Ryu spent over a decade working in financial services, including HSBC, Barclays Bank, and PriceWaterhouseCoopers, building centralized ledgers.

Table of Contents

2

Rust – Necessary Concepts for Building Blockchains 33

Part 2: Building the Blockchain

3

Building a Custom Blockchain 67

Part 3: Building Apps

6

Using Foundry to Build on Ethereum 167

7

Exploring Solana by Building a dApp 195

8

Exploring NEAR by Building a dApp 243

Part 4: Polkadot and Substrate

9

10

Part 5: The Future of Blockchains

11

Preface

Rust is one of the most widely used languages in blockchain systems and many popular blockchains including Solana, Polkadot, Aptos, and Sui are built with Rust. Rust frameworks such as Foundry are also highly preferred by developers of established chains including Ethereum.

Learning how decentralized apps work on popular Rust chains and also how to build your own blockchains – whether from scratch or using frameworks such as Substrate – is an important skill to have since all big dApps, at some point, end up moving to their own chains, also referred to as *application chains*.

This book is for developers who want to go deep and understand how Rust is used for building dApps and blockchains and add a new dimension to their Rust skills.

Who this book is for

This book is for blockchain and dApp developers, blockchain enthusiasts, and Rust engineers who want to step up their game by adding blockchain to their repertoire of skills.

What this book covers

Chapter 1, *Blockchains with Rust*, outlines the critical blockchain concepts that we will use in the book.

Chapter 2, *Rust – Necessary Concepts for Building Blockchains*, explores the critical Rust concepts that we will be using to build our own blockchain.

Chapter 3, *Building a Custom Blockchain*, lays the foundation and the building blocks for our own custom blockchain that we're building from scratch.

Chapter 4, *Adding More Features to Our Custom Blockchain*, sees up build on our blockchain and add more features to it.

Chapter 5, *Finishing Up Our Custom Blockchain*, brings together all the individual blocks that we have built and combines them into a complete blockchain.

Chapter 6, *Using Foundry to Build on Ethereum*, explores Foundry, a Rust framework that can be used to build and deploy smart contracts on Ethereum.

Chapter 7, *Exploring Solana by Building a dApp,* teaches you how to build a dApp for Solana.

Chapter 8, *Exploring NEAR by Building a dApp*, teaches you how to build a dApp for an upcoming blockchain, NEAR.

Chapter 9, *Exploring Polkadot, Kusama, and Substrate*, explores the basic concepts behind Substrate, which enables developers to build their own chains.

Chapter 10, Hands-On with Substrate, uses our knowledge of Substrate to build a custom blockchain.

Chapter 11, *Future of Rust for Blockchains*, discusses the future of blockchains with Rust.

To get the most out of this book

We're assuming that you know your way around Rust and have knowledge of all its basic concepts.

Software/hardware covered in the book	Operating system requirements
Rust 1.74.0 or higher	Windows, macOS, or Linux
Cargo	Windows, macOS, or Linux

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Rust-for-Blockchain-Application-Development`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "rustup is the toolchain manager that includes the compiler and Cargo's package manager."

A block of code is set as follows:

```
pub struct Block {
    timestamp: i64,
    pre_block_hash: String,
    hash: String,
    transactions: Vec<Transaction>,
    nonce: i64,
    height: usize,
}
```

Any command-line input or output is written as follows:

```
brew install rustup
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Working with strings is straightforward in Rust, so it's important to know the difference between the **String type** and **string literals**."

> **Tips or important notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *Rust for Blockchain Application Development*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://packt.link/free-ebook/9781837634644`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

Part 1: Blockchains and Rust

In this part, we will first get some knowledge about blockchains and some necessary Rust concepts that'll help us in building a fully fledged blockchain.

This part has the following chapters:

- *Chapter 1*, *Blockchains with Rust*
- *Chapter 2*, *Rust - Necessary Concepts for Building Blockchains*

1
Blockchains with Rust

Blockchains have a lot of mystery around them, and only a few engineers have complete clarity of the inner workings and how disruptive they will be to the incumbent way of working for many industries.

With the help of this chapter, we want to tackle the very core concepts of blockchains. Since this is a book about using Rust for blockchains, we want to, at the same time, understand why Rust and blockchains are a match made in heaven. This will also provide us with insight into why some popular blockchains (Solana, Polkadot, and NEAR) have used Rust and why the latest blockchains to enter the market (Aptos and Sui) are also choosing Rust above any other technology that exists on the market today.

The end goal of this chapter is to provide a comprehensive understanding of the critical concepts around blockchains that will enable us to build a blockchain from scratch later in the book.

In this chapter, we're going to cover the following main topics:

- Laying the foundation with the building blocks of blockchains
- Exploring the backbone of blockchains
- Understanding decentralization
- Scaling the blockchain
- Introducing smart contracts
- The future of the adoption of blockchains

Laying the foundation with the building blocks of blockchains

In this section, let's learn the most basic concept of blockchains—what a blockchain is made up of.

A **blockchain** can be imagined as a series of connected blocks, with each **block** containing a finite amount of information.

The following diagram demonstrates this clearly with multiple connected blocks.

Figure 1.1 – Representation of a blockchain

Just like in a traditional database, there are multiple tables in which the data is stored sequentially in the form of **records**, and the blockchain has multiple blocks that store a particular number of **transactions**.

The following diagram demonstrates blocks as a store for multiple transactions:

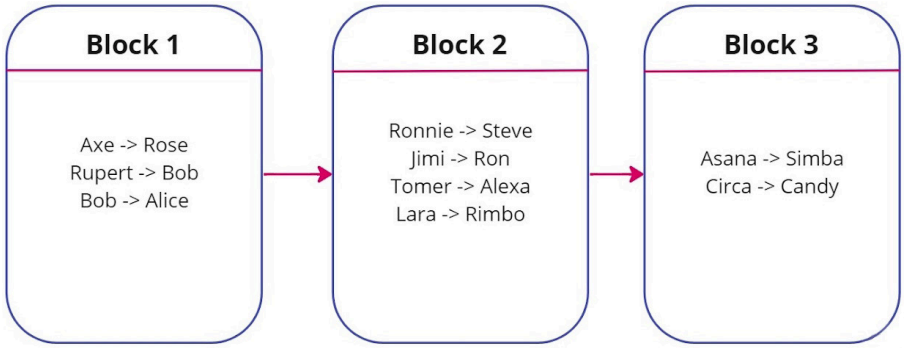

Figure 1.2 – Blocks with transaction data

The question now is, why not just use databases? Why do we even need blockchains? Well, the main difference here is that there is no admin and nobody is in charge. The other significant difference is that most blockchains are engineered to be **permissionless** at the core (even though **permissioned** blockchains exist and have specific use cases at the enterprise level), making them accessible to everyone and not just to people with access.

Another equally substantial difference is that blockchains only have insert operations, whereas databases have **CRUD** operations, making blockchains inherently **immutable**. This also implies that blockchains are not **recursive** in nature; you cannot go back to repeat a task on records while databases are recursive.

Now, this is a complete shift in how we approach data storage with blockchains in comparison to traditional databases. Then there is **decentralization**, which we will learn about shortly and that is what makes blockchains an extremely powerful tool.

Web 3.0, another confusing and mysterious term, can, at a considerably basic level, be defined as the internet of blockchains. Until now, we have had **client-server architecture** applications being connected to each other. That was Web 2.0, but suddenly, with the help of blockchains, we will have a more decentralized internet. Even if most of this does not make sense right now, do not despair, for we have plenty to cover.

In the following subsections, we will learn about things such as hashes, transactions, security, decentralized storage, and computing.

Blocks

The smallest or atomic part of any blockchain is a **block**. We learned in the previous section that blocks contain transactions, but that's not all; they also store some more information. Let's peel through the layers.

Let's look at a visual representation of the inner workings of a block:

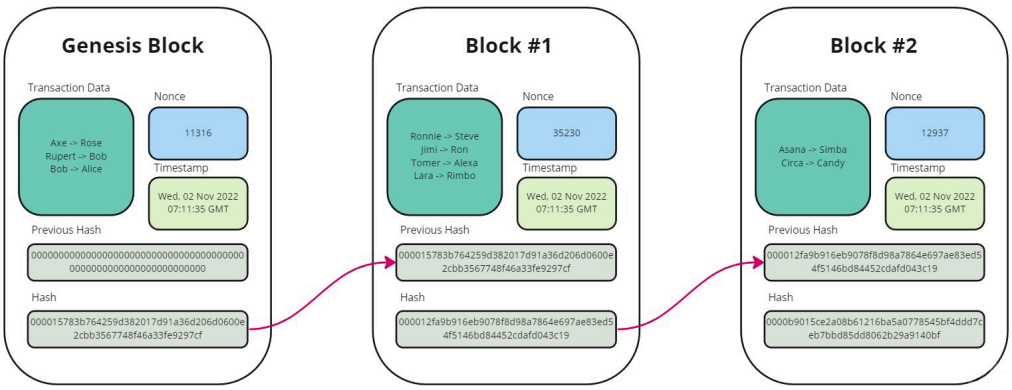

Figure 1.3 – Connected blocks of a blockchain

In the preceding diagram, we notice that the first block is called the **Genesis Block**, which is an industry-standard term for the first block of the chain. Now, apart from transaction data, you also see a **hash**. In the next section, *Hashes*, we will learn how this hash is created and why it is required. For now, let's consider it to be a random number. So, each block has a hash, and you will also notice that the blocks are storing the *previous hash*. This is the same as the hash of the previous block.

The *previous hash* block is critical because it is what *connects* the blocks to each other. There is no other aspect that connects the blocks to make a blockchain; it's simply the fact that a subsequent, sequential block holds the hash of the previous block.

We also notice a field called **nonce**. This stands for **number only used once**. For now, we need to understand that the nonce needs to be consistent with the hash for the block to be valid. If they're not consistent, the following blocks of the blockchain go completely out of sync and this fortifies the *immutability* aspect of blockchains that we will learn about in detail in the *Forking* section. Now, as we go further, we will uncover more layers to this, but we're at a great starting point and have a broad overview.

Hashes

Hashes are a core feature of the blockchain and are what hold the blocks together. We remember from earlier that blocks store hash and previous hash and hashes are simply created by adding up all the data, such as transactions and timestamps, and passing it through some hashing algorithm. One example is the **SHA-256** algorithm.

The following diagram shows a visual representation of data being passed to the SHA-256 algorithm and being converted into a usable hash:

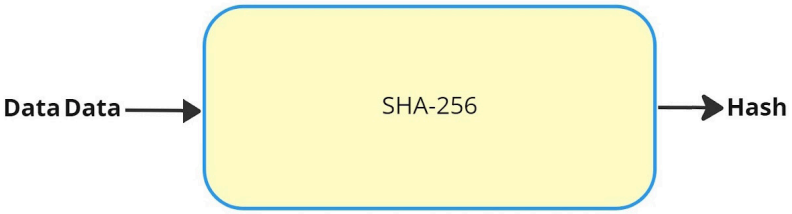

Figure 1.4 – Data to SHA-256 hash

A hash is a unique fixed-length string that can be used to identify or represent a piece of data and a hash algorithm, such as SHA-256, is a function that computes data into a unique hash.

While there are several other SHA algorithms available (such as **SHA-512**), SHA-256 stands as the most prevalent choice within blockchains due to its robust hash security features and the notable fact that it remains unbroken to this day.

There are four important properties of the SHA-256 algorithm:

- One-way: The hash generated from SHA-256 is 256 bits (or 32 bytes) in length and is irreversible; if you want to get the plaintext back (plaintext being the data that we passed through SHA-256), you will not be able to do so.
- Deterministic: Every time you send a particular data through the algorithm, you will get the same predictable result. This means that the hash doesn't change for the same data.
- Avalanche effect: Changing one character of the data, completely changes the hash and makes it unrecognizable.
- For example, the hash for *abcd* is

  `88d4266fd4e6338d13b845fcf289579d209c897823b9217da3e161936f031589`

  but the hash for *abce* is

  `84e73dc50f2be9000ab2a87f8026c1f45e1fec954af502e9904031645b190d4f`.

- The only thing common between them is that they start with `8`. There's nothing else that matches, so you can't possibly predict how the algorithm represents *a*, *b*, or *c*, and you can't work your way backward to either get the plaintext data or predict what the hash representation for some other data will look like.
- Withstand collision: Collision in hashing means the algorithm produces the same hash for two different values. SHA-256 has an extremely low probability of collision, and this is why it's heavily used.

All of these properties of the SHA-256 are the reason why blockchains are the way they are.

Let's understand the effect that these properties have by going over the following few points:

- Irreversibility translates into immutability in blockchains (transaction data, once recorded, can't be changed)
- Determinism translates into a unique, identifiable hash that can identify a user, wallet, transaction, token, or account on the blockchain (all of these have a hash)
- The avalanche effect translates into security, making the system extremely difficult to hack since the information that's encrypted can't be predicted by brute force (running multiple computers to estimate incrementally, starting with a hypothesis)
- Collision tolerance leads to each ID being unique and there being an extremely high mathematical limit to the unique hashes that can be produced, and since we require hashes to represent various types of information on the blockchain, this is an important functionality

In this section, we have seen how the properties of blockchains actually come from the hashing algorithms, and we can safely say that it's the heart and soul of a blockchain.

Transactions

Because of the previously mentioned properties of blockchains, storing financial data is one of the biggest use cases that blockchains are used for, as they have advanced security requirements.

A transaction is showcased through unspent cryptocurrency, or **unspent transaction output (UTXO)**. This refers to unused coins owned by individuals logged on the blockchain for transparency. It's essential to recognize that while UTXO is a key element in certain blockchains such as Bitcoin, it's not a universal feature across all blockchain platforms.

The following diagram helps us visualize all the fields in a transaction:

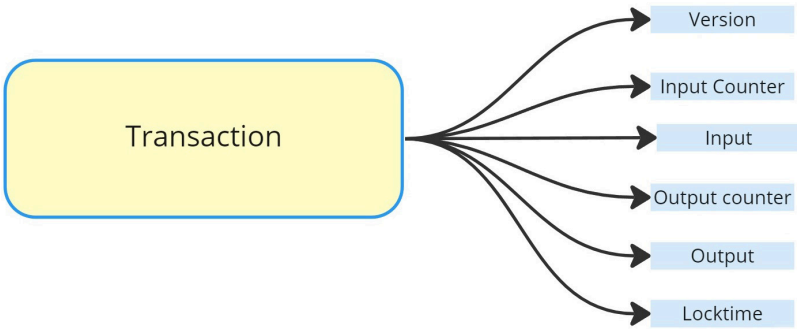

Figure 1.5 – The contents of a blockchain transaction

Let's go through all the fields that form a Bitcoin **transaction**:

- Version: This specifies which rules the transaction follows
- Input counter: This is the number of inputs in the transaction (this is just a count)
- Inputs: This is the actual input data
- Output counter: This is similar to the input counter, but it's for keeping a count of the transactions' output
- Output: This is the actual output data from the transaction
- Blocktime: This is simply a Unix timestamp that records when the transaction happened.

Initially, blockchains were primarily designed to record financial transactions within the realm of cryptocurrencies. However, as they evolved, blockchains demonstrated their versatility by finding applications beyond this initial purpose. Soon, we'll delve into these additional uses.

But for now, it is important to understand that when we mention transactions, it does not strictly mean financial or currency-related transactions. Rather, in modern blockchains, a transaction is *anything that changes the state of the blockchain*, so any program that runs or any information that's stored is simply a transaction.

Security

So, the main selling point for blockchains is that they're extremely secure. Now, let's understand why this is so:

- All the records are secured with cryptography thanks to the SHA-256 algorithm.
- The records and other blockchain data are copied to multiple nodes; we will learn about this in the *Peers, nodes, validators, and collators* section. Even if the data gets deleted in one node, it doesn't mean that it's deleted from the blockchain.

- To participate as a node in the blockchain network, requiring ownership of private keys is essential. Private keys and secret codes known only to you, grant access to control your cryptocurrency holdings, sign transactions, and ensure security. Possessing private keys safeguards your digital assets and enables engagement in network activities.
- Nodes need to come to a consensus on new data to be added to the blockchain. This means bogus data and corrupted data cannot be added to the blockchain, as it could compromise the entire chain.
- Data cannot be edited on the blockchain. This means the information you have stored cannot be tampered with.
- They're decentralized and don't have a single point of failure. The bigger the network or the more decentralized the network, the lower the probability of failure.

We will learn about nodes, decentralization, validation, and consensus later on in this book, and all these points will be clearer.

Storage versus compute

Bitcoin introduced blockchain for the storage of financial transactions, but Ethereum took things a bit further and helped us imagine what it could be like if you could run programs on a blockchain. Hence, the concept of smart contracts was created (we will dig deeper into smart contracts later in this chapter, but you can think of them as code that can run decentralized on the blockchain).

Independent nodes could join a network for the blockchain and pool their processing power in the network.

According to Ethereum, they're building the biggest **supercomputer** in the world. There are two ways to build the biggest supercomputer— build it centralized, where all machines will exist centrally in one location, or build a decentralized version where thousands of machines can be connected over the internet and divide tasks among themselves.

Ethereum enables you to process programs on the blockchain. This means anyone on the internet can build a smart contract and publish it on the blockchain where anyone else across the world can interact with the program.

This is the reason we see so many startups building their products on the Ethereum chain. After Ethereum, blockchains such as Solana, NEAR, and Polkadot have taken this idea much further and brought many new concepts by improving on Ethereum. This book is going to deal with all three of these blockchains.

Exploring the backbone of blockchains

This section is a deep dive into what makes blockchains so special. We will cover topics such as *decentralization*, *forking*, and *mining*, and we will understand how peers interact in a network and how the blocks are validated. Let's dive in.

Decentralization

From a purely technical standpoint, Web 1.0 started with a client-server architecture, usually monoliths. When traffic and data started increasing, the **monolithic** architecture couldn't scale well. Then, with Web 2.0, we had concepts such as **microservices** and **distributed systems**,which helped not only scale systems efficiently but also enhanced **resilience** and **robustness**, reduced **failure** instances, and increased **recoverability**.

The data was still centralized and private and the systems were mostly centralized, meaning they still belonged to a person/company and admins could change anything. The drawbacks were the following:

- A failure at the company's end took the system down
- Admins could edit the data and block users and content from platforms
- Security was still not prioritized, leading to easy data hacks, although this could vary depending on the company's approach to safeguarding information
- All the data generated on the platform belonged to the platform
- Content created and posted on a platform became the property of the platform

Web 3.0 ushers in a new age of decentralization that is made possible with blockchains where the entire blockchain data is copied to all the nodes. But even distributed systems had nodes and node recovery, so the question is, how is this any different?

Well, in the case of distributed systems, the nodes still belonged to the centralized authority or the company that owned the platform, and nodes were essentially their own servers in a private cloud. With decentralized systems, the node can be owned by another entity, person or a company other than the company that developed the blockchain.

In fact, in a blockchain network, having nodes owned by different companies is encouraged and this increases the *decentralization* of the network, meaning there is no real owner or authority that can block content, data, or users out and the data is accessible to all the nodes since all of them can store a copy of the data.

Even if one node goes down, there are others to uphold the blockchain, and this makes the system highly available. Advanced communication protocols among the nodes make sure the data is consistent across all the nodes.

Nodes are usually monetized to stay in the network and to uphold the security of the network (we will read more about this in the next section). Nodes also need to come to a consensus regarding the next block that's to be added to the chain. We will also read more about consensus shortly.

Peers, nodes, validators, and collators

In this section, we will further build upon the knowledge we have gained in the past few sections. A blockchain does not exist in isolation; it is a **peer-to-peer network**, and all full nodes save the complete copy of the blockchain, while some blockchains also permit other types of nodes that maintain state without necessarily possessing a full copy.

In the following diagram, we see this in a visual format:

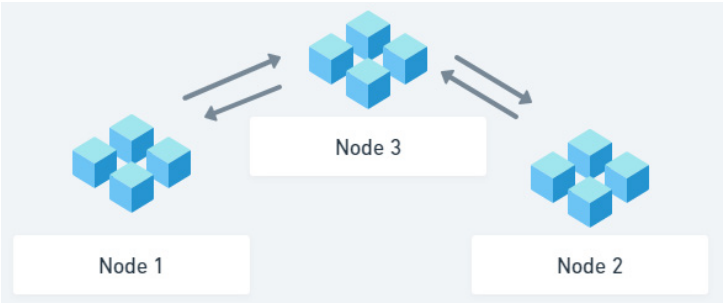

Figure 1.6 – Multi-node networks

So, let's dig a layer deeper. Nodes are listening to **events** taking place in the network. These events are usually related to transactions. It is important to reiterate that a transaction is anything that changes the state of the system.

As we know, a block contains the information of multiple transactions.

The following diagram shows a block with some example transactions:

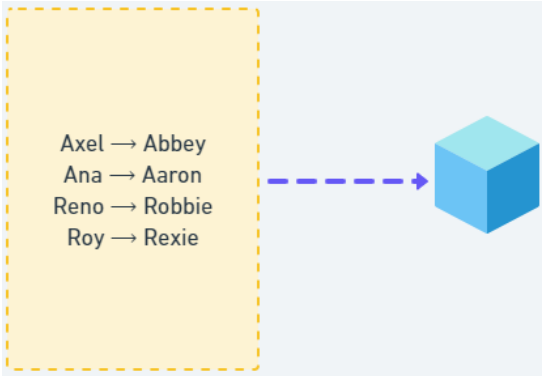

Figure 1.7 – Transactions finalized to a block

Once a new block is added by a node, which is known as mining, this new event is advertised to the entire network. This is visually represented in the following diagram:

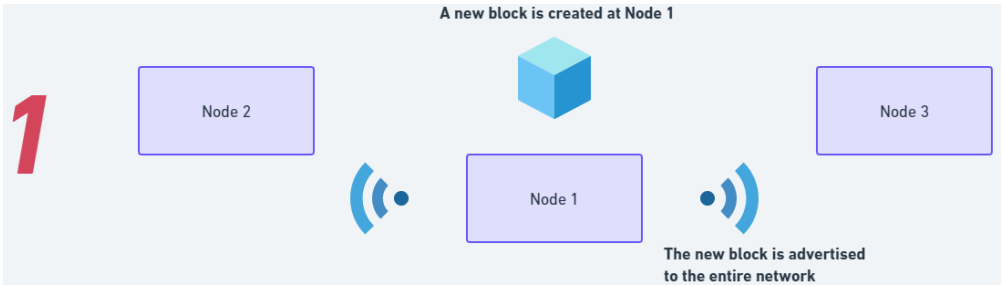

Figure 1.8 – The created block is advertised

Once the new block is advertised, the rest of the nodes act as *validators* that confirm the outputs of the transactions once the block has been validated by the rest of the nodes. The nodes come to a *consensus* that yes, this is the right block that needs to be added to the chain. We can visualize this with the help of the following diagram:

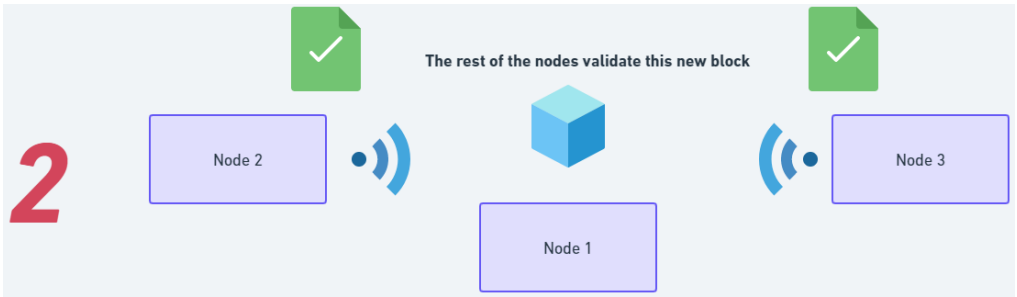

Figure 1.9 – Other nodes validate the block data

The new block is then copied to the rest of the nodes so that all of them are on the same page and added to the independent chains being maintained at each node. This can be seen in the following diagram:

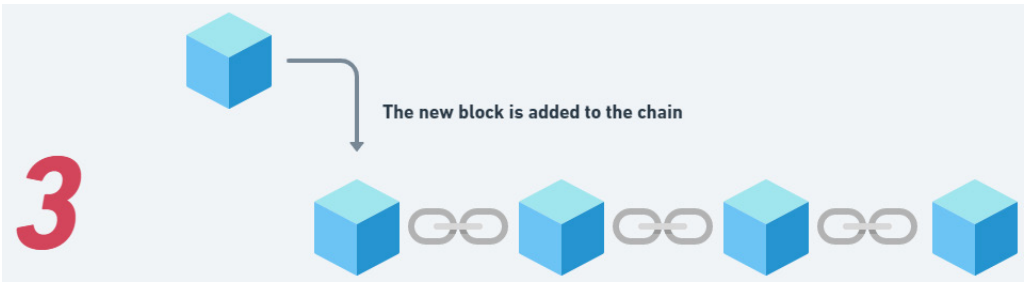

Figure 1.10 – A block gets finalized

Once the blocks are added to the node, and the blockchain at each node is updated on any other node. Another block could be listening to all the new transactions that have happened, and these are then collated onto a block and the entire process then repeats.

The following criteria vary from blockchain to blockchain in terms of the following:

- The number of transactions that the block will store
- The mechanism that nodes use to collate the transactions (time-based or number-based)
- The validation mechanism
- The consensus mechanisms

Contemporary chains improved upon the Bitcoin and Ethereum blockchains by varying and innovating on either all or some of these criteria, but the consensus mechanism is something that is most often innovated upon. This is done to try and save the time required for new nodes to be added and copied by the entire network, which is what really slows down the network.

We learned earlier that the nodes need to be incentivized to stay in the network and keep adding the blocks to the chain. In chains such as Ethereum, this is achieved using **gas fees**, which are simply small fees that users pay to carry forward their transactions. We know that blocks can contain only a few transactions, and if the users want their transactions to get priority, they need to pay gas fees.

The gas fee depends on what other users are willing to pay to get their transactions forwarded; the higher the gas fee, the higher the chance of getting your transaction accepted. Think of gas fees as the rent that the nodes get paid for the users to use the nodes' processors to process and validate their transactions. The words *peers* and *nodes* are used interchangeably, and *validators* and *collators* can also be used interchangeably depending on the blockchain you are on.

Consensus

In the last section, we learned that a node listens to transaction events, collates these transactions, and creates a block. This is called **mining**. After a block is mined, other nodes need to validate it and come to a consensus.

In this section, we want to peel the layers of consensus to understand it deeply. Understanding the mechanics behind some popular consensus mechanisms will help us to learn by running through actual examples, rather than learning in an abstract way. So, let's understand some of these concepts:

- **Proof of work** (**PoW**): Nodes need to solve a particular cryptography problem (we will look at this in detail in the *Mining* section), and the node with the highest processing power is usually able to solve faster than others. This keeps the system decentralized but increases electricity consumption by a huge amount. It's not considered to be very efficient and is even considered bad for the environment, as it increases power wastage since all the nodes are up against each other trying to solve the problem. Examples are Bitcoin, Litecoin, and Dogecoin.

- **Proof of authority (PoA)**: This is a consensus mechanism in blockchain where transactions and blocks are verified by identified validators, typically chosen due to their reputation or authority. Unlike energy-intensive mechanisms such as PoW, PoA offers efficiency by requiring validators to be accountable for their actions. It's commonly used in private or consortium blockchains, ensuring fast transactions and reducing the risk of malicious activities. However, PoA's centralized nature may raise concerns regarding decentralization and censorship resistance compared to other consensus methods.
- **Proof of stake** (**PoS**): Nodes need to buy *stakes* in the network—basically, they buy the cryptocurrency native to the network. Only a few nodes with a majority stake get to participate in the mining activity in some cases. This is highly power efficient, and this is the reason why Ethereum recently switched from PoW to PoS. However, it is considered to be less decentralized, as only the nodes with enough resources get to add the next blocks and it can be seen that some big players have been slowly taking ownership of the majority of the network since Ethereum switched to PoS. The main benefit of PoS is that since nodes have a stake in the system, they are de-incentivized to add unscrupulous blocks to the chain. Since the copy of the chain exists with all the nodes of the entire network, the nodes are running the *software* of the blockchain where the output hashes need to be consistent with the rest of the chain. Hence, when a node tries to add the wrong block, the rest of the nodes do not validate this block, and if such a scenario takes place, these nodes are then penalized where the amount of native cryptocurrency owned by the node that is taken away can differ depending on the seriousness of the violation. Generally, this penalty entails a partial loss of funds rather than a complete forfeiture of all holdings. Some examples are Cardano, Ethereum, and Polkadot.
- **Proof of burn** (**PoB**): Burning is a process where cryptocurrency is sent to a wallet address from which it's irrecoverable. The nodes that can burn the highest amount of cryptocurrency get to add a node. Miners must invest in the blockchain to demonstrate their commitment to the network. Even though PoB is the most criticized consensus model, it can actually be highly effective for some blockchains that want to ensure deflationary tokenomics. Slimcoin is an example of PoB.
- **Proof of capacity**: In this consensus mechanism, the nodes with the highest storage space get to add a node. This means that the nodes that partake in the network can use their hard drive space to compete with each other to win the mining rights. An example is Permacoin.
- **Delegated PoS**: Participants in the network, such as end users buying cryptocurrency, can stake their coins in a pool, and the pool belongs to a particular node that can add blocks to a chain. The more tokens you stake, the bigger your payout. Examples are EOS, BitShares, and TRON.

In this section, we've developed a rich understanding of consensus mechanisms, and this will help us throughout the book, especially while building the blockchain.

Mining

By now, we have a very basic idea of what **mining** is and why it's necessary. In this section, we will dive into the specifics of mining. Mining happens quite differently in different consensus mechanisms. We will look at mining for the two major consensus mechanisms: PoW and PoS. For instance, in PoS, let's consider the example of Ethereum 2.0, where validators are chosen to create new blocks and secure the network based on the amount of cryptocurrency they hold and are willing to "stake" as collateral.

In a PoW blockchain, to add a block to the blockchain, a cryptographic problem needs to be solved. The node that comes up with the solution first gets to win the competition. This means that nodes with the highest computational power usually win and get to add a block.

The blockchain's cryptography challenge adjusts in complexity over time to ensure consistent `block` creation. Nodes predict a specific hash, focusing on a segment that aligns with the existing blockchain, maintaining chain coherence.

Nodes employ a nonce, a unique value, to address the challenge. Incrementing from zero, this value is adjusted until a matching hash is computed, pivotal for generating a valid hash in line with the network's rules.

Solving the cryptographic problem validates transactions and creates new blocks. A successful node broadcasts its solution, swiftly verified by others. The first to find a valid solution is rewarded with newly minted cryptocurrency, incentivizing participation and bolstering network security.

The following diagram shows the different fields that add up to produce a hash:

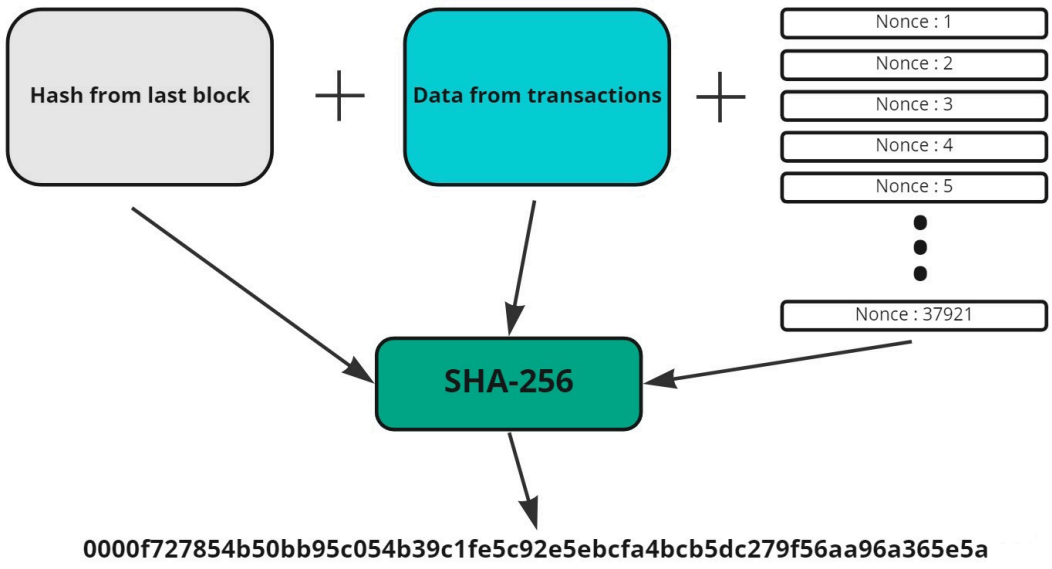

Figure 1.11 – All the data that makes up a hash

Now, this means the following:

- This can only be solved with brute forcing, iterating from zero to a particular number, and cannot be solved *smartly*
- All the nodes in the network compete with each other regardless of whether they ever win, and this means a lot of computational energy gets wasted
- Nodes need to keep upgrading their computational power to win the competition

Now that we understand what mining is and how it works, it's time to learn about forking—an important blockchain concept.

Forking

There is one small detail about blockchains that we have talked about but haven't discussed in detail yet, and that's **immutability**. In one of the earlier sections, we learned how SHA-256's properties translate into immutability for blockchains. This means all transactions that happen on-chain are immutable, and tokens once sent from one account to another cannot be reversed unless an actual transaction is initiated from the second account.

In traditional payment systems, this is not the case. If money is sent to the wrong account by mistake, this can be reversed, but this feature has been manipulated by centralized authorities and therefore immutable transactions are valued highly.

Let's take as an example the **decentralized autonomous organization** (**DAO**) attack in 2016 that led to $50 million being stolen from the Ethereum blockchain due to a code vulnerability. The only way to *reverse* this was to create an entire copy of the chain where this particular transaction didn't take place. This process of creating a different version chain is simply called *forking*. This event divided the blockchain between Ethereum and Ethereum Classic.

The following diagram demonstrates what forking looks like:

Figure 1.12 – Forks in a blockchain

Forking also comes into use when rules for the blockchain need to be modified. Traditional software gets *upgraded* and new updates and patches are applied, whereas the way to upgrade a blockchain is to *fork* (though some blockchains such as Polkadot have invented mechanisms to have forkless upgrades).

Forks typically occur intentionally, but they can also happen unintentionally when multiple miners discover a block simultaneously. The resolution of a fork takes place as additional blocks are appended, causing one chain to become longer than the others. In this process, the network disregards blocks that are not part of the longest chain, labeling them as orphaned blocks.

Forks can be divided into two categories: soft forks and hard forks.

A **soft fork** is simply a software upgrade for the blockchain where changes are made to the existing chain, whereas with a **hard fork**, a new chain is created and both old and new blockchains exist side by side. To summarize, both forks create a split, but a hard fork creates two blockchains.

Permissioned versus permissionless

Blockchains can be permissionless or permissioned depending on the use case. A permissionless blockchain is open to the public with all transactions visible, but they may be encrypted to hide some crucial details and information if required. Anyone can join the network, become a node, or be a validator if the basic criteria are met. Nodes can become a part of the governing committee as well once they can meet additional requirements, and there are no restrictions on who can join the network. You can freely join and participate in consensus without obtaining permission, approval, or authorization.

Most of the commonly known blockchains, such as Ethereum, Solana, and Polkadot, are all permissionless chains and are easily accessible. Their transaction data is publicly available. So, a perfect use case for permissionless chains is hosting user-facing and user interaction-based applications.

Permissioned chains have gatekeepers that define a permission, approval, or authorization mechanism that only a few pre-authorized nodes can operate. So, to be a part of the permissioned blockchain network, you may need a special set of private keys and may also need to match some security requirements. Since the nodes copy the entire data of the chain and are also involved in adding blocks to the chain and being a part of the governing committee for the blockchains, some use cases where data and information need to be kept private can use permissioned chains.

The following diagram shows the difference between a public and a private blockchain network:

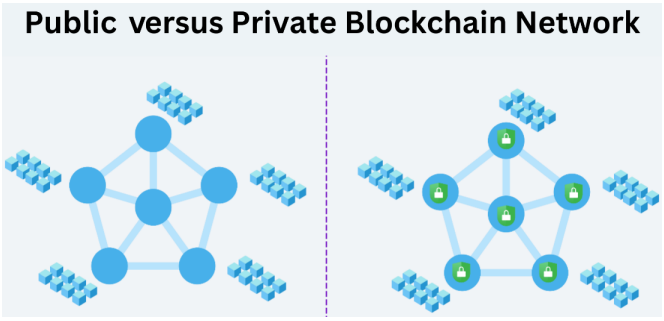

Figure 1.13 – Permissioned versus permissionless chains

Governments, institutions, NGOs, and traditional corporations have found plenty of use cases for permissioned chains, where only a few actors trusted by the centralized authorities are permitted to join the network. Permissioned blockchains also have multiple business-to-business use cases and may be centrally stored on a single cloud provider.

Blockchains help us decentralize computing and resources, and we have been using the word *decentralization* quite often. In the next section, we will understand the concept of decentralization in more depth.

Understanding decentralization

Decentralization is the guiding principle for Web 3.0. It's designed to create a win–win environment for the builders of a platform, the people that build *on* the platform **decentralized applications** (**dApps**), and the people that interact with the platform (users of dApps).

Let's try and understand why decentralization is so important. In 2013, Twitter had a centralized developers platform where developers could use their APIs to build apps on the Twitter platform. A few years later, Twitter stopped the API support and also brought in a few restrictions, and every few months, the API's terms and conditions would change. This affected many app developers who were either banned from the platform due to the restrictions or were unable to stay up to date with the changing terms for API usage.

Similarly, Facebook had an app developer program as well, which many developers built their apps with. However, developers faced similar problems here as well, and this problem is quite common wherever a centralized platform is involved. Play Store and App Store can ban any app from their platform, and Amazon can decide which sellers can sell and Uber can decide which drivers get more rides.

The issue is not just about getting banned from the platform and the policy changes, but it's also about monetization. For example, the Apple App Store can take about 30% of the entire revenue from app developers. To prevent institutions, banks, and governments from curbing the freedom of individuals and communities, decentralization is a popular solution that ensures everyone gets a voice and a few owners of the platform do not end up controlling the entire platform.

It's shared ownership where the ownership of the platform is not held closely by the founding team or the committee; rather, it belongs to the community at large where each user can hold tokens and gets a say in the system. We will read about this further in the *DAOs* section.

A blockchain network implements decentralization in a highly efficient manner, and this is why it's the primary technology for a decentralized use case.

So, now that we have a clearer understanding of decentralization, let's dig into some of the concepts that are closely related with decentralization that make it possible.