What Is Ethereum?

Ethereumis often described as "the world computer.” But what does that mean? Let’s start with a computer science–focused description, and then try to decipher that with a more practical analysis of Ethereum’s capabilities and characteristics, while comparing it to Bitcoin and other decentralized information exchange platforms (or "blockchains" for short).

From a computer science perspective, Ethereum is a deterministic but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state.

From a more practical perspective, Ethereum is an open source, globally decentralized computing infrastructure that executes programs called *smart contracts*. It uses a blockchain to synchronize and store the system’s state changes, along with a cryptocurrency called *ether* to meter and constrain execution resource costs.

The Ethereum platform enables developers to build powerful decentralized applications with built-in economic functions. While providing high availability, auditability, transparency, and neutrality, it also reduces or eliminates censorship and reduces certain counterparty risks.

Compared to Bitcoin

Many people will come to Ethereum with some prior experience of cryptocurrencies, specifically Bitcoin. Ethereum shares many common elements with other open blockchains: a peer-to-peer network connecting participants, a Byzantine fault–tolerant consensus algorithm for synchronization of state updates (a proof-of-work blockchain), the use of cryptographic primitives such as digital signatures and hashes, and a digital currency (ether).

Yet in many ways, both the purpose and construction of Ethereum are strikingly different from those of the open blockchains that preceded it, including Bitcoin.

Ethereum’s purpose is not primarily to be a digital currency payment network.While the digital currency ether is both integral to and necessary for the operation of Ethereum, ether is intended as a *utility currency* to pay for use of the Ethereum platform as the world computer.

Unlike Bitcoin, which has a very limited scripting language, Ethereum is designed to be a general-purpose programmable blockchain that runs a *virtual machine* capable of executing code of arbitrary and unbounded complexity. Where Bitcoin’s Script language is, intentionally, constrained to simple true/false evaluation of spending conditions, Ethereum’s language is *Turing complete*, meaning that Ethereum can straightforwardly function as a general-purpose computer.

Components of a Blockchain

The components of an open, public blockchain are (usually):

- A peer-to-peer (P2P) network connecting participants and propagating transactions and blocks of verified transactions, based on a standardized "gossip" protocol
- Messages, in the form of transactions, representing state transitions

- A set of consensus rules, governing what constitutes a transaction and what makes for a valid state transition
- A state machine that processes transactions according to the consensus rules
- A chain of cryptographically secured blocks that acts as a journal of all the verified and accepted state transitions
- A consensus algorithm that decentralizes control over the blockchain, by forcing participants to cooperate in the enforcement of the consensus rules
- A game-theoretically sound incentivization scheme (e.g., proof-of-work costs plus block rewards) to economically secure the state machine in an open environment
- One or more open source software implementations of the above ("clients")

All or most of these components are usually combined in a single software client. For example, in Bitcoin, the reference implementation is developed by the *Bitcoin Core* open source project and implemented as the *bitcoind* client. In Ethereum, rather than a reference implementation there is a *reference specification*, a mathematical description of the system in the Yellow Paper (see Further Reading). There are a number of clients, which are built according to the reference specification.

In the past, we used the term "blockchain" to represent all of the components just listed, as a shorthand reference to the combination of technologies that encompass all of the characteristics described. Today, however, there are a huge variety of blockchains with different properties. We need qualifiers to help us understand the characteristics of the blockchain in question, such as *open, public, global, decentralized, neutral,* and *censorship-resistant*, to identify the important emergent characteristics of a "blockchain" system that these components allow.

Not all blockchains are created equal. When someone tells you that something is a blockchain, you have not received an answer; rather, you need to start asking a lot of questions to clarify what they mean when they use the word "blockchain." Start by asking for a description of the components in the preceding list, then ask whether this "blockchain" exhibits the characteristics of being *open, public*, etc.

The Birth of Ethereum

All great innovations solve real problems, and Ethereum is no exception. Ethereum was conceived at a time when people recognized the power of the Bitcoin model, and were trying to move beyond cryptocurrency applications. But developers faced a conundrum: they either needed to build on top of Bitcoin or start a new blockchain.Building upon Bitcoin meant living within the intentional constraints of the network and trying to find workarounds. The limited set of transaction types, data types, and sizes of data storage seemed to limit the sorts of applications that could run directly on Bitcoin; anything else needed additional off-chain layers, and that immediately negated many of the advantages of using a public blockchain. For projects that needed more freedom and flexibility while staying on-chain, a new blockchain was the only option. But that meant a lot of work: bootstrapping all the infrastructure elements, exhaustive testing, etc.

Toward the end of 2013, Vitalik Buterin, a young programmer and Bitcoin enthusiast, started thinking about further extending the capabilities of Bitcoin and Mastercoin (an overlay protocol that extended Bitcoin to offer rudimentary smart contracts). In October of that year, Vitalik

proposed a more generalized approach to the Mastercoin team, one that allowed flexible and scriptable (but not Turing-complete) contracts to replace the specialized contract language of Mastercoin. While the Mastercoin team were impressed, this proposal was too radical a change to fit into their development roadmap.

In December 2013, Vitalik started sharing a whitepaper that outlined the idea behind Ethereum: a Turing-complete, general-purpose blockchain. A few dozen people saw this early draft and offered feedback, helping Vitalik evolve the proposal.

Both of the authors of this book received an early draft of the whitepaper and commented on it. Andreas M. Antonopoulos was intrigued by the idea and asked Vitalik many questions about the use of a separate blockchain to enforce consensus rules on smart contract execution and the implications of a Turing-complete language. Andreas continued to follow Ethereum's progress with great interest but was in the early stages of writing his book *Mastering Bitcoin*, and did not participate directly in Ethereum until much later.Dr. Gavin Wood, however, was one of the first people to reach out to Vitalik and offer to help with his C++ programming skills. Gavin became Ethereum's cofounder, codesigner, and CTO.

As Vitalik recounts in his "Ethereum Prehistory" post:

> This was the time when the Ethereum protocol was entirely my own creation. From here on, however, new participants started to join the fold. By far the most prominent on the protocol side was Gavin Wood…
>
> Gavin can also be largely credited for the subtle change in vision from viewing Ethereum as a platform for building programmable money, with blockchain-based contracts that can hold digital assets and transfer them according to pre-set rules, to a general-purpose computing platform. This started with subtle changes in emphasis and terminology, and later this influence became stronger with the increasing emphasis on the "Web 3" ensemble, which saw Ethereum as being one piece of a suite of decentralized technologies, the other two being Whisper and Swarm.

Starting in December 2013, Vitalik and Gavin refined and evolved the idea, together building the protocol layer that became Ethereum.

Ethereum's founders were thinking about a blockchain without a specific purpose, that could support a broad variety of applications by being *programmed*. The idea was that by using a general-purpose blockchain like Ethereum, a developer could program their particular application without having to implement the underlying mechanisms of peer-to-peer networks, blockchains, consensus algorithms, etc. The Ethereum platform was designed to abstract these details and provide a deterministic and secure programming environment for decentralized blockchain applications.

Much like Satoshi, Vitalik and Gavin didn't just invent a new technology; they combined new inventions with existing technologies in a novel way and delivered the prototype code to prove their ideas to the world.

The founders worked for years, building and refining the vision. And on July 30, 2015, the first Ethereum block was mined. The world's computer started serving the world.

NOTE

Vitalik Buterin's article "A Prehistory of Ethereum" was published in September 2017 and provides a fascinating first-person view of Ethereum's earliest moments.

You can read it at https://vitalik.ca/general/2017/09/14/prehistory.html.

Ethereum's Four Stages of Development

Ethereum's development was planned over four distinct stages, with major changes occurring at each stage.A stage may include subreleases, known as "hard forks," that change functionality in a way that is not backward compatible.

The four main development stages are codenamed *Frontier*, *Homestead*, *Metropolis*, and *Serenity*. The intermediate hard forks that have occurred to date are codenamed *Ice Age*, *DAO*, *Tangerine Whistle*, *Spurious Dragon*, *Byzantium*, *Constantinople/St. Petersburg*, *Istanbul* and *Muir Glacier*. Both the development stages and the intermediate hard forks are shown on the following timeline, which is "dated" by block number:

Block #0

Frontier—The initial stage of Ethereum, lasting from July 30, 2015, to March 2016.

Block #200,000

Ice Age—A hard fork to introduce an exponential difficulty increase, to motivate a transition to PoS when ready.

Block #1,150,000

Homestead—The second stage of Ethereum, launched in March 2016.

Block #1,192,000

DAO—A hard fork that reimbursed victims of the hacked DAO contract and caused Ethereum and Ethereum Classic to split into two competing systems.

Block #2,463,000

Tangerine Whistle—A hard fork to change the gas calculation for certain I/O-heavy operations and to clear the accumulated state from a denial-of-service (DoS) attack that exploited the low gas cost of those operations.

Block #2,675,000

Spurious Dragon—A hard fork to address more DoS attack vectors, and another state clearing. Also, a replay attack protection mechanism.

Block #4,370,000

Metropolis Byzantium—Metropolis is the third stage of Ethereum. Launched in October 2017, Byzantium is the first part of Metropolis, adding low-level functionalities and adjusting the block reward and difficulty.

Block #7,280,000

Constantinople / St. Petersburg—Constantinople was planned to be the second part of Metropolis with similar improvements. A few hours before its activation, a critical bug was discovered. The hard fork was therefore postponed and renamed St. Petersburg.

Block #9,069,000

Istanbul—An additional hard fork with the same approach, and naming convention, as for the prior two.

Block #9,200,000

Muir Glacier—A hard fork whose sole purpose was to adjust the difficulty again due to the exponential increase introduced by Ice Age.

Two hard forks, Berlin and London, have also been announced, and we are now in the final stage of Ethereum development, codenamed Serenity. Serenity involves a profound reorganization of the infrastructure that will make Ethereum more scalable, more secure, and more sustainable. It is presented as the second version of Ethereum, "Ethereum 2.0".

Ethereum: A General-Purpose Blockchain

The original blockchain, namely Bitcoin's blockchain, tracks the state of units of bitcoin and their ownership.You can think of Bitcoin as a distributed consensus *state machine*, where transactions cause a global *state transition*, altering the ownership of coins. The state transitions are constrained by the rules of consensus, allowing all participants to (eventually) converge on a common (consensus) state of the system, after several blocks are mined.

Ethereum is also a distributed state machine. But instead of tracking only the state of currency ownership,Ethereum tracks the state transitions of a general-purpose data store, i.e., a store that can hold any data expressible as a *key–value tuple*. A key–value data store holds arbitrary values, each referenced by some key; for example, the value "Mastering Ethereum" referenced by the key "Book Title". In some ways, this serves the same purpose as the data storage model of *Random Access Memory* (RAM) used by most general-purpose computers. Ethereum has memory that stores both code and data, and it uses the Ethereum blockchain to track how this memory changes over time. Like a general-purpose stored-program computer, Ethereum can load code into its state machine and *run* that code, storing the resulting state changes in its blockchain. Two of the critical differences from most general-purpose computers are that Ethereum state changes are governed by the rules of consensus and the state is distributed globally. Ethereum answers the question: "What if we could track any arbitrary state and program the state machine to create a world-wide computer operating under consensus?"

Ethereum's Components

In Ethereum, the components of a blockchain system described in Components of a Blockchain are, more specifically:

P2P network

Ethereum runs on the *Ethereum main network*, which is addressable on TCP port 30303, and

runs a protocol called *ĐΞVp2p*.

Consensus rules

Ethereum's consensus rules are defined in the reference specification, the Yellow Paper (see Further Reading).

Transactions

Ethereum transactions are network messages that include (among other things) a sender, recipient, value, and data payload.

State machine

Ethereum state transitions are processed by the *Ethereum Virtual Machine* (EVM), a stack-based virtual machine that executes *bytecode* (machine-language instructions). EVM programs, called "smart contracts," are written in high-level languages (e.g., Solidity) and compiled to bytecode for execution on the EVM.

Data structures

Ethereum's state is stored locally on each node as a *database* (usually Google's LevelDB), which contains the transactions and system state in a serialized hashed data structure called a *Merkle Patricia Tree*.

Consensus algorithm

Ethereum uses Bitcoin's consensus model, Nakamoto Consensus, which uses sequential single-signature blocks, weighted in importance by PoW to determine the longest chain and therefore the current state. However, there are plans to move to a PoS weighted voting system, codenamed *Casper*, in the near future.

Economic security

Ethereum currently uses a PoW algorithm called *Ethash*, but this will eventually be dropped with the move to PoS at some point in the future.

Clients

Ethereum has several interoperable implementations of the client software, the most prominent of which are *Go-Ethereum* (*Geth*) and *Parity*.

Further Reading

The following references provide additional information on the technologies mentioned here:

- The Ethereum Yellow Paper: https://ethereum.github.io/yellowpaper/paper.pdf
- The Beige Paper, a rewrite of the Yellow Paper for a broader audience in less formal language: https://github.com/chronaeon/beigepaper
- ĐΞVp2p network protocol: https://github.com/ethereum/devp2p/blob/master/rlpx.md
- Ethereum Virtual Machine list of resources: https://eth.wiki/en/concepts/evm/ethereum-virtual-machine-(evm)-awesome-list
- LevelDB database (used most often to store the local copy of the blockchain): https://github.com/google/leveldb

- Merkle Patricia trees: https://eth.wiki/en/fundamentals/patricia-tree
- Ethash PoW algorithm: https://eth.wiki/en/concepts/ethash/ethash
- Casper PoS v1 Implementation Guide: http://bit.ly/2DyPr3l
- Go-Ethereum (Geth) client: https://geth.ethereum.org/
- Parity Ethereum client: https://parity.io/

Ethereum and Turing Completeness

As soon as you start reading about Ethereum, you will immediately encounter the term "Turing complete." Ethereum, they say, unlike Bitcoin, is Turing complete. What exactly does that mean?

The term refers to English mathematician Alan Turing, who is considered the father of computer science. In 1936 he created a mathematical model of a computer consisting of a state machine that manipulates symbols by reading and writing them on sequential memory (resembling an infinite-length paper tape). With this construct, Turing went on to provide a mathematical foundation to answer (in the negative) questions about *universal computability*, meaning whether all problems are solvable. He proved that there are classes of problems that are uncomputable.Specifically, he proved that the *halting problem* (whether it is possible, given an arbitrary program and its input, to determine whether the program will eventually stop running) is not solvable.

Alan Turing further defined a system to be *Turing complete* if it can be used to simulate any Turing machine. Such a system is called a *Universal Turing machine* (UTM).

Ethereum's ability to execute a stored program, in a state machine called the Ethereum Virtual Machine, while reading and writing data to memory makes it a Turing-complete system and therefore a UTM. Ethereum can compute any algorithm that can be computed by any Turing machine, given the limitations of finite memory.

Ethereum's groundbreaking innovation is to combine the general-purpose computing architecture of a stored-program computer with a decentralized blockchain, thereby creating a distributed single-state (singleton) world computer. Ethereum programs run "everywhere," yet produce a common state that is secured by the rules of consensus.

Turing Completeness as a "Feature"

Hearing that Ethereum is Turing complete, you might arrive at the conclusion that this is a *feature* that is somehow lacking in a system that is Turing incomplete. Rather, it is the opposite. Turing completeness is very easy to achieve; in fact, the simplest Turing-complete state machine known has 4 states and uses 6 symbols, with a state definition that is only 22 instructions long. Indeed, sometimes systems are found to be "accidentally Turing complete." A fun reference of such systems can be found at http://bit.ly/2Og1VgX.

However, Turing completeness is very dangerous, particularly in open access systems like public blockchains, because of the halting problem we touched on earlier. For example, modern printers are Turing complete and can be given files to print that send them into a frozen state. The fact that Ethereum is Turing complete means that any program of any complexity can be computed by Ethereum. But that flexibility brings some thorny security and resource management problems. An

unresponsive printer can be turned off and turned back on again. That is not possible with a public blockchain.

Implications of Turing Completeness

Turing proved that you cannot predict whether a program will terminate by simulating it on a computer. In simple terms, we cannot predict the path of a program without running it.Turing-complete systems can run in "infinite loops," a term used (in oversimplification) to describe a program that does not terminate. It is trivial to create a program that runs a loop that never ends. But unintended never-ending loops can arise without warning, due to complex interactions between the starting conditions and the code. In Ethereum, this poses a challenge: every participating node (client) must validate every transaction, running any smart contracts it calls. But as Turing proved, Ethereum can't predict if a smart contract will terminate, or how long it will run, without actually running it (possibly running forever). Whether by accident or on purpose, a smart contract can be created such that it runs forever when a node attempts to validate it. This is effectively a DoS attack. And of course, between a program that takes a millisecond to validate and one that runs forever are an infinite range of nasty, resource-hogging, memory-bloating, CPU-overheating programs that simply waste resources. In a world computer, a program that abuses resources gets to abuse the world's resources. How does Ethereum constrain the resources used by a smart contract if it cannot predict resource use in advance?

To answer this challenge, Ethereum introduces a metering mechanism called *gas*. As the EVM executes a smart contract, it carefully accounts for every instruction (computation, data access, etc.). Each instruction has a predetermined cost in units of gas. When a transaction triggers the execution of a smart contract, it must include an amount of gas that sets the upper limit of what can be consumed running the smart contract. The EVM will terminate execution if the amount of gas consumed by computation exceeds the gas available in the transaction. Gas is the mechanism Ethereum uses to allow Turing-complete computation while limiting the resources that any program can consume.

The next question is, 'how does one get gas to pay for computation on the Ethereum world computer?' You won't find gas on any exchanges.It can only be purchased as part of a transaction, and can only be bought with ether. Ether needs to be sent along with a transaction and it needs to be explicitly earmarked for the purchase of gas, along with an acceptable gas price. Just like at the pump, the price of gas is not fixed. Gas is purchased for the transaction, the computation is executed, and any unused gas is refunded back to the sender of the transaction.

From General-Purpose Blockchains to Decentralized Applications (DApps)

Ethereum started as a way to make a general-purpose blockchain that could be programmed for a variety of uses. But very quickly, Ethereum's vision expanded to become a platform for programming DApps. DApps represent a broader perspective than smart contracts. A DApp is, at the very least, a smart contract and a web user interface. More broadly, a DApp is a web application that is built on top of open, decentralized, peer-to-peer infrastructure services.

A DApp is composed of at least:

- Smart contracts on a blockchain
- A web frontend user interface

In addition, many DApps include other decentralized components, such as:

- A decentralized (P2P) storage protocol and platform
- A decentralized (P2P) messaging protocol and platform

TIP	You may see DApps spelled as *ĐApps*. The Đ character is the Latin character called "ETH," alluding to Ethereum. To display this character, use the Unicode codepoint 0xD0, or if necessary the HTML character entity eth (or decimal entity #208).

The Third Age of the Internet

In 2004 the term "Web 2.0" came to prominence, describing an evolution of the web toward user-generated content, responsive interfaces, and interactivity. Web 2.0 is not a technical specification, but rather a term describing the new focus of web applications.

The concept of DApps is meant to take the World Wide Web to its next natural evolutionary stage, introducing decentralization with peer-to-peer protocols into every aspect of a web application. The term used to describe this evolution is *web3*, meaning the third "version" of the web.First proposed by Dr. Gavin Wood, web3 represents a new vision and focus for web applications: from centrally owned and managed applications, to applications built on decentralized protocols.

In later chapters we'll explore the Ethereum web3.js JavaScript library, which bridges JavaScript applications that run in your browser with the Ethereum blockchain. The web3.js library also includes an interface to a P2P storage network called *Swarm* and a P2P messaging service called *Whisper*. With these three components included in a JavaScript library running in your web browser, developers have a full application development suite that allows them to build web3 DApps.

Ethereum's Development Culture

So far we've talked about how Ethereum's goals and technology differ from those of other blockchains that preceded it, like Bitcoin. Ethereum also has a very different development culture.

In Bitcoin, development is guided by conservative principles: all changes are carefully studied to ensure that none of the existing systems are disrupted. For the most part, changes are only implemented if they are backward compatible. Existing clients are allowed to opt-in, but will continue to operate if they decide not to upgrade.

In Ethereum, by comparison, the community's development culture is focused on the future rather than the past. The (not entirely serious) mantra is "move fast and break things." If a change is needed, it is implemented, even if that means invalidating prior assumptions, breaking compatibility, or forcing clients to update. Ethereum's development culture is characterized by rapid innovation, rapid evolution, and a willingness to deploy forward-looking improvements, even if this is at the expense of some backward compatibility.

What this means to you as a developer is that you must remain flexible and be prepared to rebuild your infrastructure as some of the underlying assumptions change. One of the big challenges facing developers in Ethereum is the inherent contradiction between deploying code to an immutable system and a development platform that is still evolving. You can't simply "upgrade" your smart contracts. You must be prepared to deploy new ones, migrate users, apps, and funds, and start over.

Ironically, this also means that the goal of building systems with more autonomy and less centralized control is still not fully realized. Autonomy and decentralization require a bit more stability in the platform than you're likely to get in Ethereum in the next few years. In order to "evolve" the platform, you have to be ready to scrap and restart your smart contracts, which means you have to retain a certain degree of control over them.

But, on the positive side, Ethereum is moving forward very fast. There's little opportunity for "bike-shedding," an expression that means holding up development by arguing over minor details such as how to build the bicycle shed at the back of a nuclear power station. If you start bike-shedding, you might suddenly discover that while you were distracted the rest of the development team changed the plan and ditched bicycles in favor of autonomous hovercraft.

Eventually, the development of the Ethereum platform will slow down and its interfaces will become fixed. But in the meantime, innovation is the driving principle. You'd better keep up, because no one will slow down for you.

Why Learn Ethereum?

Blockchains have a very steep learning curve, as they combine multiple disciplines into one domain: programming, information security, cryptography, economics, distributed systems, peer-to-peer networks, etc. Ethereum makes this learning curve a lot less steep, so you can get started quickly. But just below the surface of a deceptively simple environment lies a lot more. As you learn and start looking deeper, there's always another layer of complexity and wonder.

Ethereum is a great platform for learning about blockchains and it's building a massive community of developers, faster than any other blockchain platform. More than any other, Ethereum is a *developer's blockchain*, built by developers for developers. A developer familiar with JavaScript applications can drop into Ethereum and start producing working code very quickly. For the first few years of Ethereum's life, it was common to see T-shirts announcing that you can create a token in just five lines of code. Of course, this is a double-edged sword. It's easy to write code, but it's very hard to write *good* and *secure* code.

What This Book Will Teach You

This book dives into Ethereum and examines every component. You will start with a simple transaction, dissect how it works, build a simple contract, make it better, and follow its journey through the Ethereum system.

You will learn not only how to use Ethereum—how it works—but also why it is designed the way it is. You will be able to understand how each of the pieces works, and how they fit together and why.