

Fundamentals of Blockchain

Digital Assets - Week 1-2 (Lecture)

Rhys Bidder
rhys.m.bidder@kcl.ac.uk

KBS, QCGBF

Autumn 2024

Disclaimer 1: Any opinions expressed, errors or omissions should be regarded as those of the author and not necessarily those of KCL, KBS, QCGBF, QCB, CBI, or BoE.

Disclaimer 2: These notes on cryptography are heavily simplified and leave out many important details. For any real-world security applications these notes should not be relied upon. Instead you should consult appropriate security resources and reliable security professionals.

Disclaimer 3: Cryptoasset transactions are illegal in some jurisdictions. Do not violate these or any other laws. I am not promoting the use of crypto in countries where it is illegal in any form and these slides are not a promotion of crypto or an invitation to participate in crypto-related activities in such countries. They are purely for educational purposes.

Outline

Introduction to blockchain

Blocks

Network structure

Addresses, wallets, keys. . .

Transactions

Consensus

Introduction to blockchain

Introduction to blockchain

- ▶ **Q:** What is a ledger?
 - A database recording current 'account' balances and the history of transactions between 'accounts'
 - E.g. 1: Set of accounts handwritten in a physical book
 - E.g. 2: Set of accounts entered in an Excel spreadsheet

Introduction to blockchain

- ▶ People often use the terms 'distributed ledger technology' and 'blockchain' interchangeably
- ▶ But blockchain is only a *particular type* of DLT
- ▶ Not all examples of DLT are blockchains (see [here](#) or [here](#))
- ▶ One can create a **distributed ledger** by putting an excel spreadsheet in a shared passwordless Google drive
- ▶ Shared ledger that anyone can inspect, store and change
- ▶ **DLT, but not a blockchain**

Introduction to blockchain

Blockchain \Rightarrow *DLT*

DLT \nRightarrow *Blockchain*

Introduction to blockchain

We will be discussing blockchains

Introduction to blockchain

- ▶ Imagine we want to store pieces of data - or '*transactions*' - that arrive as time passes
 - For now, we will not specify the precise form of a transaction
- ▶ A **block** is made up of. . .
 - A header
 - Transactions data
 - Perhaps other objects, depending on the blockchain protocol
- ▶ Header contains several components - the most vital being:
 - Hash of the previous block (the **genesis block** has no previous block in which case hash is set trivially to zero)
 - Merkle root of tree of hashed transactions

Introduction to blockchain

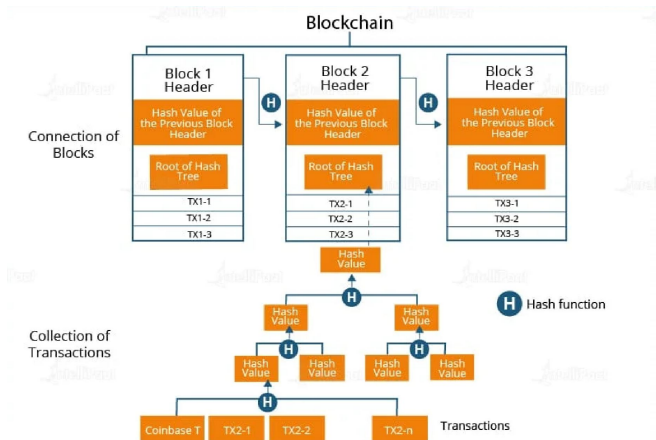
Q: Why is the hash of the previous block so vital?

- ▶ Because it links the block to the previous block
- ▶ Iterate on this logic: **this is what yields a blockchain**
- ▶ All blocks are linked together *via* repeated pairwise inclusion of the previous block's hash in the current block's header

Q: Why is the Merkle root of transactions so vital?

- ▶ Because it allows rapid comparison of transactions data

Introduction to blockchain



Source: [IntelliPaat](#)

Blocks

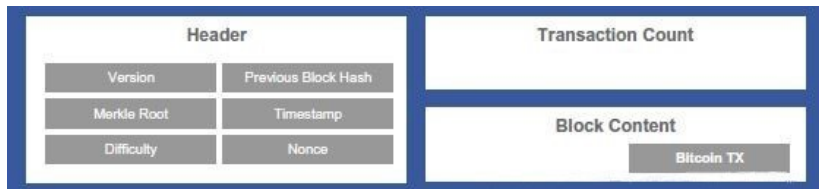
Consider a (simplified) block structure:



Source: [Hellwig et al](#) - *Build Your Own Blockchain* (amended)

Blocks

Block structure in the **Bitcoin** blockchain...



Source: [Hellwig et al](#) - *Build Your Own Blockchain*

- ▶ We will discuss 'difficulty' and the 'nonce' shortly
- ▶ Permitted 'size' of the transactions set varies across protocols

Blocks

Let us consider an actual Bitcoin block:

- ▶ The TX list here is actually the list of transaction IDs (TXIDs) - *hashes* of the transactions data
- ▶ Strictly, the *current block's* hash is not part of current block

```
{
  "hash": "0000000000000000000000000000000000000000000000000000000000000000",
  "confirmations": 37371,
  "size": 138429,
  "height": 277316,
  "version": 2,
  "merkleroot": "c5c808c26e98763a9f548b8b2fc323735f73577effbc55582c51eb4cc7c2a",
  "tx": [
    {
      "d5da064c6417ca25c4308bd158c34b77e1c8eca2a73cda16c737e7424efb2f",
      "b26b45c5b39d75961475771b09918caf0b9a9d97c6f3691956f877c5b3f0e",
      "04905f9876d4c4f6d3b03c7b7ca5b0a51d8961f8f5f265c38f763a0a421fd",
      "32467a8b5d04f51940073055c2f20b0d1195727c961431b0aff8443f9710f81",
      "561c5216946a21fa2b6d12aa1a45a3397f9c0d888359cb05a1f796a73da37bd",
      [...] hundreds of transactions ...
      "78b30b02a1d20949958b07bc718c84d6e0579617e0da4991b9734cef7ab23a",
      "6c87130ec283ab4c2c493b190c20d64b28ff3ca72d16fFa1ce3e96f2069aca9",
      "6f423dbc3636ef193f8898df7621dcade1bb509a963f7b0f9f5f696d81a62",
      "8022a8b2adabc5796a9471f25b02a6aaw02439c579a5c33c40bcaw97a0811596",
      "aaf6a48380d9a4d4c19c2539d571d80af396c532a30a8b51a6767a1af",
      "e67abc6bd5a2ca369821af51b207127f42b92a841e976f9b752157879ba8bd",
      "d38985a6a1bf035037cb7776b2dc86797abb7a9638f5d03df2785d90d5a2ac",
      "45e0a3f681662b099b092c347a0c9767671a8a84b9bcca0c019a60197c134b",
      "c0984456748ced5f170ef2ff96f2758bcecb32c0fcd5db313bca3d3bc98f"
    ]
  },
  "time": 1388185914,
  "mediantime": 1388183675,
  "nonce": 924519712,
  "bits": "1903a30c",
  "difficulty": 1180921195.258026,
  "chainwork": "0000000000000000000000000000000000000000000000000000000000000000",
  "previousblockhash": "000000000000002a7bbd25a417c8374cc55261021eba9ca74442b01284f0569",
  "nextblockhash": "0000000000000000001036c2696d6dd714dd5db39436b33959979678fd431ba7"
}
```

Output from [bitcoin-cli getblock](#). Source: [Antonopoulos - Mastering Bitcoin](#)

Blocks

- ▶ Block is at 'block height' 277316 with hash...
**00000000000000001b6b9a13b095e96db41c4a928b97ef2d
944a9b31b2cc7bdc4**
- ▶ Blocks are being stacked 'on top' of each other (another way of looking at the chain)
- ▶ **Remember for later:** Note the run of zeros at the start of the hash

Blocks

```
{
  "hash": "00000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
  "height": 277316,
  "version": 2,
  "merkleroot": "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
  "tx": [
    "d5ade064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afb2f",
    "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbc",
    "04905ff987ddd4cfe603b03cfb7ca50ee081d89d1f8f5f265c38f763eea4a21fd",
    "32467aab5d04f51940075055c2f20bbd1195727c961431bf0aff8443f9710f81",
    "561c5216944e21fa29dd12aaa1a45e3397f9c0d888359cb05e1f79fe73da37bd",
    [... hundreds of transactions ...]
  ],
  "time": 1388185914,
  "nonce": 924591752,
  "difficulty": 1180923195.258026,
  "previousblockhash": "00000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569"
}
```

Output from `bitcoin-cli getblock`. Source: [Antonopoulos - Mastering Bitcoin](#)

Blocks

Let's find that transaction in the Bitcoin ledger!

- ▶ Many ways to do this - e.g. programmatically, using command line and API
- ▶ Here we will use [Blockchain.com's explorer](#)
- ▶ The block is found [here](#)
- ▶ The first entry in the transaction list is found [here](#) and the second is found [here](#)
- ▶ **Q:** Look at the urls of these links - what are they?
 - For block, the height is used, and for transactions, the TXID

Blocks

Pause. Breathe. Take a step back.

Blocks

Pause. Breathe. Take a step back.

Our discussion \Rightarrow 'just' a database with particular structure:

- ▶ Data arrives
- ▶ It gets bundled up in a block that points to the previous block
- ▶ Repeat (over and over)

What is in the data might be very simple. . .

- ▶ In simplest Bitcoin case, just value transfers

. . . or it may be more complex

- ▶ In Ethereum, the transactions may encode *elaborate* smart contracts and other data (in fact, the Bitcoin transactions involve simple smart contracts too. . .)

But it's still just bits of data

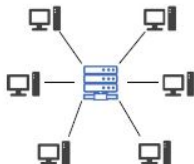
Network structure

- ▶ We have discussed data being bundled into blocks, and added to the blockchain
- ▶ But how? Who does it? Who stores the ledger?
- ▶ I could make a blockchain on my computer just for me
 - That would be pretty stupid (and sad/lonely)
- ▶ Blockchain comes into its own in a particular **network** context

Network structure

- ▶ Many systems we are familiar with operate under a **centralized** network structure
- ▶ Ledger stored on a 'server' run by a centralized authority
- ▶ Other nodes - using 'client' software - access ledger only through the server
- ▶ Changes to ledger only possible by interfacing with, and with the approval of, the server
- ▶ Central authority *could* unilaterally change data, reform system, or limit user access

Network structure



Server model. Source: [Hellwig et al](#) - *Build Your Own Blockchain* (amended)

Network structure

Another centralized system central bankers will be familiar with. . .

- ▶ Settling transactions between commercial banks' reserve accounts, *via* the central bank's balance sheet
- ▶ Very common for transactions between financial institutions (typically banks) to settle in this way
- ▶ CB maintains a single copy of the ledger, reflecting past transactions and current account balances (see [here](#) for further discussion)

Network structure

Consider another framework, with less centralization:

- ▶ Network participants interact through 'transactions' but...
- ▶ Suppose:
 - each participant maintains own ledger of *their* transactions
 - messages with transaction details are sent bilaterally by phone, email, or some system *separate from the ledgers*
 - each participant updates their ledger by converting the message data into their own *idiosyncratic* format
 - conversion possibly subject to error, using techniques/software unknown to other participants
- ▶ In many (even advanced) countries, this is how much financial trading occurs

Network structure

- ▶ It is only after **checking and reconciliation** occurs that trades (implied by the 'messages') are actually **settled**
 - **$T + K$ delays:** T is transaction day, K might be as long as 3
 - Many markets now achieve $T + 2$
 - Some examples (not Blockchain-based) of $T + 0$ but *rare*
 - These processes are expensive (direct and time costs)
- ▶ Settlement may fail or be delayed due to disagreements
 - Adds to the K of $T + K$
 - Expensive reversion of trades/litigation
- ▶ In summary. . .
 - Trade \neq settlement
 - Multiple distinct, proprietary ledgers are at play
 - Various *ad hoc* coordination systems required to agree on accepted common truth

Network structure

A peer to peer network differs from a client-server model in that:

- ▶ **Client-Server:** Clients request data and services from server
- ▶ **P2P:** *All* nodes can request data **and** provide services

Network structure



P2P model. Source: [Hellwig et al](#) - *Build Your Own Blockchain* (amended)

Network structure

In the extreme case, a fully decentralized p2p network:

consist(s) of individual nodes, which make their computational resources directly available to all other members of the network without having any central point of coordination. The nodes in the network are equal concerning their rights and roles in the system. Furthermore, all of them are both suppliers and consumers of resources.

In the context of blockchain this means that *any* node has the same rights and responsibilities in administering, accessing, storing, and updating the ledger

- ▶ This is the Bitcoin/Ethereum setup

Network structure

Blockchains differ in their precise network structure

- ▶ Roles and capabilities of different nodes
- ▶ Extent of decentralization (we will return to this topic)

Important distinction: Is the blockchain **‘public’** or **‘private’**?

- ▶ Often referred to as **‘permissionless’** or **‘permissioned’**

Network structure

Public/permissionless blockchain:

- ▶ **Open participation**
 - Anyone with relevant hardware + client software can join
 - No credentials required
 - Pseudonymous (and in some cases anonymous)
- ▶ Explicitly **decentralized** across (often enormous) number of participants
 - No central authority
 - Though some centralization may be latent/unintentional
- ▶ Designed to operate in **‘trustless’** contexts
 - Mechanism design/incentives ensure robustness to (some) nefarious behavior
 - **Arguably the most spectacular achievement of blockchain (starting with Bitcoin)**

Network structure

Private/permissioned blockchain:

- ▶ **Limits participation**

- Typically through some sort of certificate authority
- Significant credentialing requirement (AML/KYC)

- ▶ More likely to entail (explicit) **centralization** (e.g CA, monitoring/regulator nodes, validating/notary nodes, block-building nodes)

- There may even be a 'natural' central authority
- Nevertheless, still significant decentralization in most cases

- ▶ Significant **'trust'** often can be assumed among participants

- Perhaps familiarity through business network / activities
- Threat of reputation loss/punishment (CA \Rightarrow identifiable)

- ▶ Several companies / consortia now help set these up

- Hyperledger (Fabric)
- R3 (Corda)

Addresses, wallets, keys. . .

Recall our earlier discussions of asymmetric key cryptography

- ▶ Let us assume we have (carefully) obtained a private key (a very large random integer) and its public counterpart

Blockchain addresses are **derived from** the public key

- ▶ Basically involves a load of hashing of the public key
- ⇒ Results in a shorter and more easily usable representation

Addresses, wallets, keys. . .

Knowing the private key \Leftrightarrow ownership of cryptoassets associated with public address

- ▶ **The private key should be kept safe and never revealed**
- ▶ Loosely: 'Not your keys, not your coins'

Transactions will be sent to (public) addresses

- ▶ Only the person with the private key can use assets associated with that address
- ▶ **Why?** They have the private key to sign (and thus authorize) any transactions

Addresses, wallets, keys. . .

A **wallet** stores private keys

- ▶ Typically also provides other functionality, such as signing transactions, analytics, a nice user interface etc.
- ▶ It may hold many private keys and help administer many associated addresses

In some blockchains (e.g. Bitcoin) there isn't a native concept of an 'account' distinct from an address

- ▶ Some wallets may allow an account abstraction by collecting information on transactions to and from addresses associated with a given user
- ▶ **Note:** A user can (and typically will) have several (or even a vast number of) addresses

Addresses, wallets, keys. . .

Many crypto wallets are now available (see [here](#))

- ▶ Wallets offer [security-convenience tradeoffs](#)
 - **Cold wallets**: Disconnected from the internet (keys stored offline - inconvenient but relatively secure)
 - **Hot wallets**: Connected to the internet (keys stored online - convenient but relatively vulnerable to theft/hacks)
- ▶ In the UK, I legally use (without endorsement!)

Cold [Ledger Nano X](#)

Hot [Metamask](#) (app and [browser extension](#))

Hot [Wallet](#) (app from [Coinbase](#))

Terminology warning: Many people (including me) will often talk about '*sending BTC to an address, or wallet*' but transactions simply transfer **ownership claims** among addresses and only **keys** are stored in a wallet (not BTC).

Addresses, wallets, keys. . .

I hold (very) limited amounts of cryptoassets for research and education purposes

- ▶ **For me** there is limited downside to losing my crypto, so (while being careful) I don't adopt a super secure approach
- ▶ If I were to get nervous or hold significant amounts, I would use my Nano X much more and generally take greater care!

Transactions

So far, we have discussed:

- ▶ **Addresses**, between which transactions are sent and the cryptographic techniques underpinning them
- ▶ How **private key** knowledge allows the use (and effectively defines ownership) of cryptoassets, by signing transactions with **digital signatures**
- ▶ How **blocks** bundle up **transactions** and are **linked together into a chain**

But what are transactions?

Transactions

Taking a technical (rather than legal) perspective, a 'transaction' is a *signed message*

- ▶ Messages are constantly being sent and received among the nodes in the P2P network
- ▶ Client software used by nodes execute certain actions on the basis of the messages and their current view of the chain
- ▶ These actions update the 'state' of a blockchain when they are included in blocks (if they are ever included) and consensus is achieved

The format of the message, the actions they induce, and the nature of the 'state' will (obviously) depend on the specific protocol

Concept of 'state'

Digression on concept of a 'state' and its 'evolution'

- ▶ Most economists encounter the concept of the 'state' in first year PhD macro or econometrics!
- ▶ Our models talk about how an economy evolves from time t to time $t + 1$
- ▶ By iterating on this logic, we can talk about how the economy evolves over many periods
- ▶ We talk about how the world 'is' today, how it 'will be' tomorrow, and how it gets there

Concept of 'state'

How can we describe how the world 'is' today?

- ▶ Do we need to list 'everything'?
- ▶ No - we can describe a smaller set of variables *from which everything else that 'is' can be derived?*
- ▶ **That set of variables is the state**

Tell me the state today and I can tell you 'everything' about today

- ▶ Similarly, tell me what the state will be tomorrow and I can tell you 'everything' about tomorrow

A model determines how the state evolves (or transitions) from the value it holds today, to the value it holds tomorrow

Concept of 'state'

There may be various ways of choosing state variables, or representing it

- ▶ Maybe there is some reason to represent it differently
- ▶ Possibly for clarity, computational load, compatibility. . .

▶ Examples

Concept of 'state'

For a blockchain, **transactions are what evolve the state**

- ▶ They are the only source of randomness
- ▶ Blockchain nodes running clients will come to the same answer if they execute with the same knowledge of the state
- ▶ The blockchain protocols and the actions of nodes collectively constitute a 'replicated deterministic state machine'

Concept of 'state'

Essentially **a blockchain and its protocols define a type of computer** that continually calculates updates to the state and records the state's evolution (see [here](#))

- ▶ This interpretation is *heavily* emphasized in Ethereum
- ▶ In Bitcoin, 'transactions' have a similar meaning to our daily use of the word
- ▶ In Ethereum (and other blockchains) a 'transaction' means something far more general

This is why so many people think blockchain is such a transformative computing technology for decentralized and trustless environments

Concept of 'state'

Digression over!

- ▶ Let's get back to discussing what a transaction is in practice

Consult pre-record for **Bitcoin** transactions

- ▶ We now discuss **Ethereum** transactions

Ethereum transactions

Note: Calling this subsection 'Ethereum transactions' is a bit of a trick because it will contain...

- ▶ Discussion of Ethereum's background
- ▶ Accounts, Ether, gas
- ▶ Plus a little bit of (see pre-record) smart contracts

Ethereum transactions

There are many topics we discussed above in the context of Bitcoin that will apply to Ethereum also - so I will try and focus on the particularities of Ethereum

- ▶ We have already (in the example of the account balance view of transactions) begun to transition towards Ethereum
- ▶ Many of the cryptographic tools are - allowing for implementation differences - very similar
- ▶ The abstract concept of a blockchain is also, essentially shared (though see later for discussion of *consensus*)

Ethereum transactions

Ethereum is in many respects an attempt to extend the functionality of Bitcoin

- ▶ While it offers a cryptocurrency (Ether or ETH), its functionality is designed to be far broader than simply payments
- ▶ ETH is used to allow more elaborate computation to be done by the Ethereum system - it is a *utility currency*
- ▶ Far more general and powerful smart contracts than is possible in Bitcoin

Ethereum transactions

Far more general and powerful **smart contracts** than is possible in Bitcoin

- ▶ **What?** Where were the smart contracts in Bitcoin?
- ▶ Remember the PubKeyScript?

Bitcoin has a smart contract programming language: **Script**

- ▶ Script is not 'Turing complete'
- ▶ Inherent theoretical limitations to what it can compute

Ethereum transactions

I was going to write a high-level summary of Ethereum but I can't do any better than the following three paragraphs written in the excellent book - [Mastering Ethereum](#) - by one of its founders, [Gavin Wood](#) (though the person most people associate with it is its other founder, [Vitalik Buterin](#)). . .

Ethereum transactions

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 crypto currency 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.

Ethereum transactions

*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 **crypto currency 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.*

Ethereum transactions

- ▶ Note that the 'computer' he refers to is often termed the 'Ethereum Virtual Machine' or (EVM)
- ▶ The EVM acts on 'bytecode': a bunch of instructions for the computer that are hard for me and you to understand
 - Bytecode is obtained by 'compiling' 'sourcecode' that you or I could write and understand - with some practice... 😊
 - We could write the smart contract sourcecode in a few languages, but the dominant one is Solidity
 - Smart contracts + interface \Rightarrow 'dApp' (decentralized App)
- ▶ Transactions (sending ETH and data) instigate evolution of the system's state
- ▶ Participants in the Ethereum blockchain network must run client software - one implementation of which is `geth`
- ▶ Like Bitcoin, Ethereum is a public/permissionless blockchain

Ethereum transactions

```
PUSH1 0x60 PUSH1 0x40 MSTORE CALLVALUE ISZERO PUSH2 0xF JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST PUSH1 0xE5 DUP1 PUSH2 0x1D PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN
STOP PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH1 0x3F JUMPI
PUSH1 0x0 CALLDATALOAD PUSH29
0x1000000000000000000000000000000000000000000000000000000000000000
SWAP1 DIV PUSH4 0xFFFFFFFF AND DUP1 PUSH4 0x2E1A7D4D EQ PUSH1 0x41 JUMPI
JUMPDEST STOP JUMPDEST CALLVALUE ISZERO PUSH1 0x40 JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST PUSH1 0x5F PUSH1 0x4 DUP1 DUP1 CALLDATALOAD SWAP1 PUSH1 0x20 ADD SWAP1
SWAP2 SWAP1 POP POP PUSH1 0x61 JUMP JUMPDEST STOP JUMPDEST PUSH8
0x1634578508A0000 DUP2 GT ISZERO ISZERO ISZERO PUSH1 0x77 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST CALLER PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND
PUSH2 0xBFC DUP3 SWAP1 DUP2 ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1
0x40 MLOAD DUP1 DUP4 SUB DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO
ISZERO PUSH1 0xB6 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP JUMP STOP LOG1 PUSH6
0x627A7A723050 KECCAK256 PUSH9 0x13D1EA839A4438EF75 GASLIMIT CALLVALUE LOG4 0x5F
PUSH24 0x7541F409787592C988A079407FB28B4AD000290000000000
```

Bytecode for simple 'faucet' example. Source: [Mastering Ethereum](#)

Ethereum transactions

```
1 // Our first contract is a faucet!
2 contract Faucet {
3
4     // Give out ether to anyone who asks
5     function withdraw(uint withdraw_amount) public {
6
7         // Limit withdrawal amount
8         require(withdraw_amount <= 10000000000000000);
9
10        // Send the amount to the address that requested it
11        msg.sender.transfer(withdraw_amount);
12    }
13
14    // Accept any incoming amount
15    function () public payable {}
16
17 }
```

Solidity source code for simple 'faucet' example. Source: [Mastering Ethereum](#)

Ethereum transactions

Native currency of the Ethereum system is Ether (ETH)

- ▶ There are various **sub-denominations**
- ▶ For example: $1ETH = 1 \times 10^{18}wei$

To execute any transaction the sender must pay 'gas'

- ▶ The cost of a unit of gas is denominated in ETH
- ▶ How much gas is required depends on complexity of computation implied by the transaction
 - If triggers a complicated smart contract, then 'more' gas
 - If simply sending ETH from A to B then 'less' gas

One of the common criticisms of Ethereum is that gas prices are volatile and prohibitively high

Ethereum transactions

Why have gas?

- ▶ Gas fees go towards incentivizing network participants (validators) who verify and process TX
 - More on validators later...
- ▶ Gas limits are also key in preventing the Ethereum system being overwhelmed by endless computation
 - Turing completeness \Rightarrow 'anything' can be programmed
 - Inadvertently/maliciously, a smart contract might never stop
 - Every TX must set a max amount of gas it can 'spend'
 - Once that runs out, the computation will cease

Ethereum transactions

As in Bitcoin, Ethereum has **accounts** controlled by private key

- ▶ Referred to as '**Externally Owned Accounts**' (EOA)
- ▶ Not operated by smart contract code

But it also has a second type of account, controlled by smart contract code assigned to it at its creation

- ▶ These are referred to as '**Contract Accounts**' (CA)
- ▶ Not *directly* controlled by an agent with a private key

CA may *respond* if a user sends a transaction to it, or to another CA that in turn sends a transaction to it

- ▶ Only EOA can **start** this (perhaps complex sequence of) calls

Ethereum transactions

Here is where the concept of a transaction becomes very broad

- ▶ In Bitcoin, value was sent between addresses, with a bit of programmatic (Script) information
- ▶ **In Ethereum, TXs can contain ETH and data**
- ▶ When the CA receives the ETH and the data, its associated smart contract will operate on both
 - ETH is deposited in the 'contract balance'
 - Data will specify a function (there may be many) in the contract and inputs for it

On running, the CA may send TXs to other EOA or CA to...

- ▶ Trigger (or make) other smart contracts
- ▶ Make other transfers of value and data

Ethereum transactions

Another key distinction between Ethereum and Bitcoin is that contracts can store info between TXs

- ▶ In Bitcoin, a contract would simply receive UTXO, consume it, and output another UTXO
 - Nothing is saved between calls, so it executes the same way for a given input
- ▶ Ethereum associates 'storage' with contracts, allowing them to be 'stateful'
 - The value of the (contract) state may determine its response to a given input
- ▶ Vital for storing token balances, among other things

Ethereum transactions

Transactions have the following components:

- ▶ **Nonce** ('number used once')
- ▶ Gas, Maximum Fee per Gas, Maximum Priority Fee per Gas
- ▶ Digital signature of sender
- ▶ Recipient address
- ▶ Value of ETH to be sent
- ▶ Data to be sent
- ▶ ChainID

Ethereum transactions

The 'nonce' is important because of the **account** rather than **UTXO** approach of Ethereum:

- ▶ Ethereum's state is a list of address-**account balance** pairs
- ⇒ Risk that a transaction could be 'replayed'
 - Could keep sending identical transaction until account drained
 - In UTXO, first transaction consumes a particular UTXO
 - UTXO no longer exists for a 'replay' transaction
- ▶ Nonce tracks # transactions from an address
 - Incremented with every transaction from the address
- ▶ So it will never be the same in two transactions
 - Trying the 'same' transaction again will be revealed

Sidenote: ChainID also plays a similar role in preventing replays across different chains following forks

Ethereum transactions

Maximum Fee per Gas and Maximum Priority Fee per Gas

- ▶ **Max fee:** Limits what sender is prepared to pay for gas
- ▶ **Max priority fee:** Controls how much TX fee sender will pay to validator to include in block

Consensus

So far, we understand that:

- ▶ A blockchain is a database that:
 - enables transactions
 - organizes and records transactions data
- ▶ The resulting ledger comprises a chain of linked blocks
 - Transactions (transfers and SC calls) bundled into blocks
 - 'State' of the ledger evolves via transactions
- ▶ Communication is based on a distributed network
 - But degree of centralization may differ according to protocol
- ▶ Blockchains rely *heavily* on cryptographic techniques
 - Asymmetric key systems for wallets, signing
 - Hashing for predictable and compact data formats

But there is a big gap in our coverage so far...

Consensus

- ▶ Blockchain participants must (eventually) agree on what the 'true' ledger is (the history of valid transactions)
 - What transactions should be added to blocks (validation)
 - What sequence of blocks is the accepted 'state' of the system
- ▶ How this comes about is referred to as the '**consensus mechanism**' of the blockchain
 - 'Consensus' connotes a **distributed** form of agreement
 - Public blockchains (and many private) do not rely on a single centralized trusted party to impose agreement

Consensus

If the consensus mechanism does not ensure sensible outcomes then everything else we've discussed is pointless

- ▶ Why should we expect many pseudonymous participants, unknown to each other and untrusted by each other, to reach agreement on a desirable outcome?
- ▶ What if incompetent, malfunctioning, or nefarious participants are present?
- ▶ Think about the earlier 'shared public Excel spreadsheet' example (that would be a **disaster**)

Consensus

- ▶ All nodes must (eventually) accept the current 'state' of the blockchain as a single, shared truth
- ▶ This allows deterministic execution/approval of transactions
 - All nodes (eventually) treat the same transaction in same way
- ▶ The protocol may achieve this 'consensus' in various ways:
 - **Bitcoin:** Proof of Work (PoW) and Longest Chain Rule
 - **Ethereum:** Was PoW, [now](#) Proof of Stake (PoS)
 - **Permissioned systems:** Often use Proof of Authority (PoA)
 - Other protocols exist - see [here](#) for example

Consensus

Let us consider what is known as a 'Sybil attack':

- ▶ A single entity creates (very) many accounts/addresses that *ostensibly* are different users, but in fact **are all controlled by that entity**
 - Recall, public blockchains are typically pseudonymous
 - It's essentially costless to create an account
 - Thus, they are especially vulnerable to this sort of attack
- ▶ Makes it difficult to reach legitimate consensus - certainly with a naive 'voting' protocol
 - Suppose one player controls more than half of all addresses
 - ⇒ enough influence to manipulate the blockchain record
 - Replace previously 'confirmed' blocks, manipulate which transactions get added, censor other participants. . .

Proof of work

We first consider PoW

- ▶ Still the most 'famous' consensus mechanism and part of the broader 'Nakamoto' consensus protocol used by Bitcoin

Basic idea:

- ▶ Provide incentives such that proper behavior is individually rational and bad behavior is individually irrational
- ▶ Requires proposal of a valid block to be 'costly'
- ▶ On the flip side, if a valid block is proposed by a node, there should be 'rewards'
- ▶ Recall 'incentive compatibility' from university economics (?)

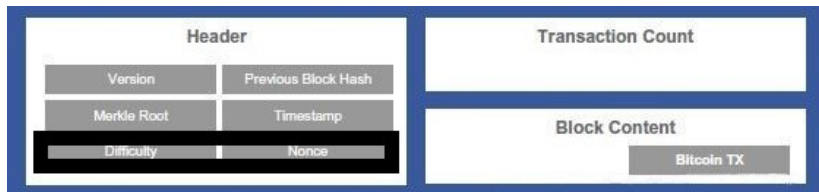
'Work': Makes block proposals costly

'Proof': Impossible to propose valid block without work

Proof of work

To understand this we reconsider the contents of a generic block in the chain. . .

Proof of work



Source: [Hellwig et al](#) - *Build Your Own Blockchain*

Recall we deferred discussion of 'difficulty' and the 'nonce'

- ▶ They are just numbers - it's how they are used that is clever

This is a *Bitcoin* block but similar logic applies to other PoW setups (see others [here](#))

- ▶ What constitutes 'work' could differ
- ▶ *Something* costly that can be verifiably completed is necessary

Proof of work

Let us summarize the process of proposing a new block:

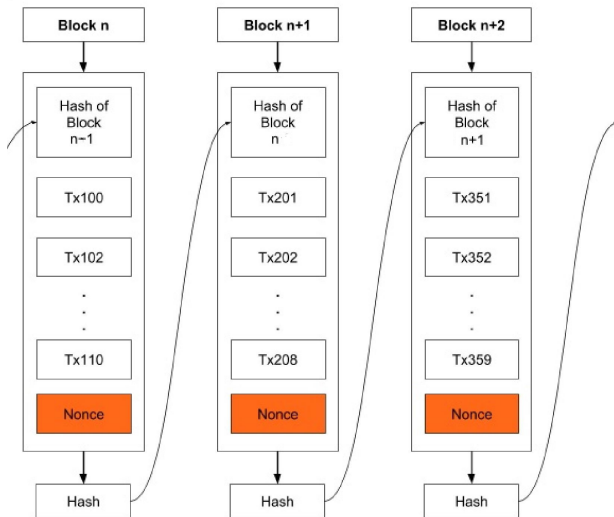
1. Download 'the' up-to-date chain from network peer(s)
 - Different chains may be regarded as 'the' up to date chain by different nodes (P2P network latency)
 - Eventually this is resolved (will discuss)
2. Check validity of all the blocks/transactions in the chain
 - Hashes, Merkle trees, Merkle roots etc. \Rightarrow quick process
3. Choose a set of *pending* transactions from the 'mempool(s)'
 - A *mempool* is like a waiting area of validated transactions not yet included in a block in the chain
 - Nodes come to be aware of such transactions because of P2P communication across the network

Proof of work

Once the node (the '**block miner**') has selected transactions it wishes to add to the proposed block, the miner...

4. Assembles all the elements of a block (*except* the nonce)
5. Guesses a value for the **nonce** to complete the candidate block
6. Hashes the *candidate* block's header (twice) using SHA-256
7. The hash value will have a certain number of leading zeros
 - If number is 'large enough' then block is valid and the miner broadcasts it to the network
 - If not, then the node tries another nonce (repeat process...)

Blockchain



Source: [TutorialsPoint](#)

Proof of work

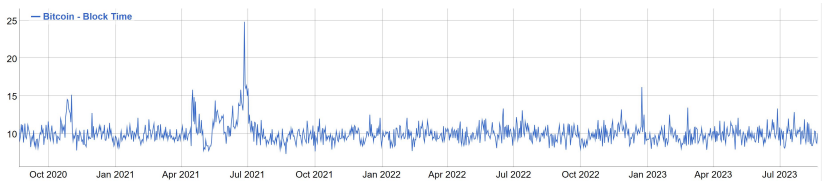
If number is 'large enough' then block is valid and the miner broadcasts it to the network

- ▶ **Q:** What is 'large enough'?
- ▶ **A:** This is determined by the 'difficulty'

In the Bitcoin protocol, the difficulty is automatically adjusted every two weeks

- ▶ **Goal:** Keep the time taken for a new block to be added at approximately 10 minutes
- ▶ The more hashing power is devoted to mining, the greater the difficulty must be (see [recent discussions](#))

Proof of work



Source: bitinfocharts.com

Proof of work

Recall, because of the nature of hash functions, it's impossible to know in advance what nonce will work

- ▶ Thus - at the cost of enormous computational power, electricity, hardware installation - the only approach is brute force guessing
 - In mid-2022 Bitcoin mining was consuming 150 terawatt-hours of electricity (more than Argentina)
- ▶ Miner keeps trying until finds a suitable nonce **or** (often) some other miner succeeds first!
 - Risky business \Rightarrow risk-averse miners often pool together to share risk and rewards

Proof of work

Once a miner finds a valid block, it broadcasts the new block to peers in the network

- ▶ Peers that receive it check it and add it to the chain they currently possess
- ▶ **Note that it is hard/slow to find the nonce - but very quick to check it**
- ▶ They remove any of the transactions included in the new block from their mempools

Proof of work

Three questions:

- ▶ Why do miners bother?
 - They aren't charities!
- ▶ What happens if two or more miners find legitimate blocks almost simultaneously?
 - Leads to a 'fork' \Rightarrow network isn't fully synchronized across nodes (for a time)
- ▶ What prevents (or discourages) bad behavior?
 - Builds on answers to the previous two questions

Why do miners bother?

Why do miners bother?

- ▶ Mining is competitive and costly
- ▶ Soon after Bitcoin's emergence, mining nodes began to invest in specialized hardware to enhance their hash rate
- ▶ Higher hash rate \Rightarrow higher chance of finding legit nonce first

Clearly there must be some incentive to do this:

- ▶ **Block rewards** sent to miners in a 'coinbase transaction'
- ▶ **Transaction fees** assigned to successful miner by the initiators of transactions
- ▶ These rewards are paid in BTC
 - **Common for platforms to use 'native currency' to incentivise**

Why do miners bother?

Block rewards

- ▶ This is how newly issued bitcoins enter the system (the 'monetary policy') of Bitcoin
- ▶ Only a maximum of 21bn bitcoin will ever be issued and the block reward **halves** approx. every 4 years

Transaction fees

- ▶ Optional, but miners less likely to select transactions for inclusion without it
- ▶ Up to some point, a higher fee will likely get a transaction **added to a block more quickly**
- ▶ Even if it's added to the 'next' block, that will typically take 10 minutes

These rewards incentivise good behavior

What happens in the case of a fork?

It could be that two or more miners almost simultaneously find legitimate nonces and broadcast their valid block

- ▶ Given the difficulty, there is unlikely to be many that do so (one reason why Bitcoin targets 10 min. blocks)
- ▶ But it regularly happens and is referred to as a 'fork' (excellent discussion [here](#))

Some nodes will see and accept one block, and others will see and accept another

- ▶ Reflects P2P nature of communication and network latency
- ▶ Some miners attempt add to one chain, and some to another
- ▶ Two (or more) chains emerge from a common block
- ▶ Which one grows faster is somewhat random - **but one will become 'longer'** (entailing more hashing 'work')

What happens in the case of a fork?

Knowledge of the chains spreads through the P2P network. . .

- ▶ The '**Longest Chain Rule**' (LCR) is respected by nodes
- ▶ When they see the longer chain, they discard the alternative
- ▶ They switch to the longest chain (called a '**reorganization**') and eventually (typically) the entire network will do so
 - Unless shorter chain's miners add to chain quickly enough
 - Requires massive amount of computational effort (+ luck)
 - Especially difficult if longer chain is 'several' blocks ahead
 - Each block added 'on top' of longer chain is a '**confirmation**'

Strictly, it is PoW + LCR that constitutes the Bitcoin consensus mechanism

What happens in the case of a fork?

- ▶ Transactions included in discarded chain (and not in the longest) return to mempool(s)
 - Can be enormously disruptive
 - **Good practice:** Wait for a 'reasonable' number of confirmations before treating as 'final'
 - What constitutes 'reasonable' depends on cost/benefit of speed vs reorg risk and is problem specific
- ▶ There is always some (eventually vanishingly small) probability or a transaction being reverted
 - **Bitcoin transactions are never completely final!**
- ▶ Block rewards (coinbase and transaction fees) to miners of 'orphaned' blocks are withdrawn
 - **Note:** While Ethereum used PoW, there was some residual reward kept by miners on 'ommer' or 'uncle' chains

What prevents bad behavior?

Some mechanical checks:

- ▶ Client software on nodes can rapidly check the formal correctness of transactions and blocks
- ▶ Quickly discard invalid transactions/blocks
- ▶ Merkle root \Rightarrow rapid detection of any tampering with previously accepted blocks

Incentives and network structure:

- ▶ Waste of computational power and loss of block rewards if fail
- ▶ If platform has massive hashing power distributed across many uncoordinated players, it is wildly expensive to manipulate the blockchain
- ▶ *Coordinated* nefarious behavior makes attacks more plausible (but still thought highly unlikely in mature systems)

What prevents bad behavior?

Type 1 'double spend' - attempt a transaction that uses currency that has already been spent

- ▶ Assume the agent is 'small' (not much hashing power)

Response: Time stamps and checking if UTXO is already used

What prevents bad behavior?

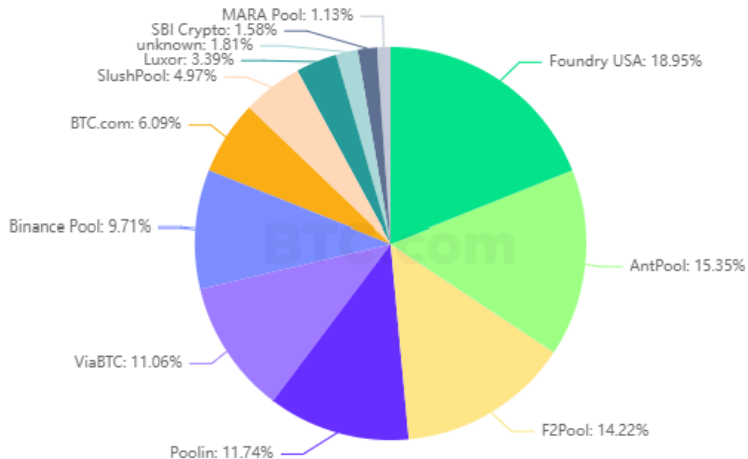
Type 2 'double spend' - manipulate blockchain with 51% attack

- ▶ Assume the agent is 'large' (or coordinated group of agents)
- ▶ *Somewhat* more plausible they control enough hash power
- ▶ Create fork where new chain omits original payment
- ▶ Release chain once long enough to be accepted by network
and after goods purchased are delivered
- ▶ Attacker still has currency to 'spend' a second time

Response: PoW makes attack very expensive and difficult

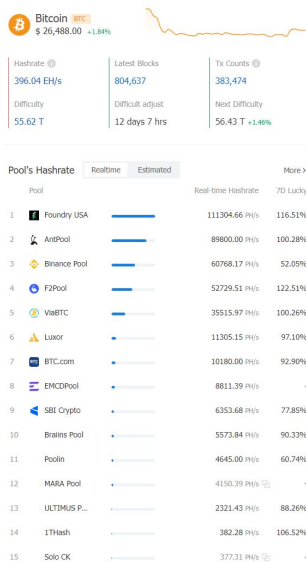
- ▶ Must be extremely costly to acquire 51% hashing power
- ▶ Vital the network collectively has massive hashing power relative to any nefarious agent
- ▶ Hashing power shouldn't be too concentrated

What prevents bad behavior?



Source: [Coindesk](#) - March 26, 2022

What prevents bad behavior?



Source: [BTC.com explorer](#) - August 24, 2023

What prevents bad behavior?

How problematic is the concentration?

- ▶ Depends - well known mining pools have a lot of 'franchise value' riding on the continued health of PoW blockchains
- ▶ The problem with 51% attacks is not so much the spend, but the enormous reputational loss for the platform
- ▶ Many of these pools would lose a lot of future revenue (and possibly capital loss on any currency they hold) if they were to abuse the system
- ▶ More of a concern for non-economic (e.g. state) actors / terrorists (rich and tooled up terrorists)
- ▶ Note the virtuous circle of confidence, high BTC valuation, adoption and **security**

What prevents bad behavior?

Side note 1: People sometimes claim that 'code is law' - the idea that the blockchain and smart contracts does, or will do, away with traditional legal structures

- ▶ But clearly if a double spend were achieved, that would be breaking the law
- ▶ Need for appropriate legal/regulatory framework (will discuss)

Side note 2: People sometimes (often) claim blockchains are 'immutable'

- ▶ In the case of Bitcoin, there is a clear sense in which it is not (orphaning of many blocks under a 51% attack)

What prevents bad behavior?

Side note 3: We referred to agents' economic interest as inducing good behavior - and the possible loss of currency value in the case of bad behavior

- ▶ The issue of having a 'stake' in the success of the blockchain platform is formalized in another common consensus mechanism, **Proof of Stake**

Side note 4: Website [here](#) *claims* to estimate cost of 1-hour long attack on PoW chains

- ▶ Greater risk for smaller blockchains

Proof of stake

- ▶ Since 'the merge' Ethereum has been a Proof of Stake blockchain (see [here](#) for other PoS chains)
- ▶ We will focus on Ethereum's PoS protocol, but the thrust of the logic is quite general

Proof of stake

- ▶ Instead of winning the right to add the next block by solving a hard problem, a 'validator' node is chosen randomly
- ▶ The more Ether is 'staked' by a validator, the higher the probability of being chosen
 - Stake by locking ETH into the system ('skin in the game')
 - 32 ETH needed per validator (\approx \$53k as of August 24th 2023)
- ▶ By staking, the agent helps enhance the validation process and secure the blockchain, in exchange for rewards
 - There are also penalties/punishment if they neglect responsibility or are malicious
- ▶ See [here](#) (and the appendix) for more detail
 - Also, see [here](#) and [here](#) for a great low-tech overviews

PoS vs PoW

- ▶ PoW uses **vastly** more energy - with environmental consequences (though much may be from renewable sources)
 - One of the most commonly cited benefits of PoS
 - Regulators definitely **paying attention**
 - Ethereum's energy consumption ↓ 99.99% after 'the merge'
 - ETH simply needs to be locked up - no work done
- ▶ PoW has simpler governance
 - One dimensional and based on simple incentives - may make more accessible
 - But PoS offers simplicity of delegated staking and has limited hardware demands
- ▶ PoS regarded as having faster validation of transactions
 - No need to wait for mining and blocks created on a more regular schedule - though there are (much) faster mechanisms
 - May make other scalability improvements easier to implement
 - PoS gives quicker and less ambiguous finality than PoW

PoS vs PoW

- ▶ PoS arguably has a tendency to centralize staked ETH
 - Rewards may make the rich richer (and thus more influential)
 - Some people also point to (excessive?) influence of key voices in Ethereum ([Vitalik Buterin](#), for example)
- ▶ Early implementations of PoS locked up staked assets
 - Perhaps reduces liquidity
 - [Liquid staking](#) now available (tokens issued against stake)
 - But there are centralization risks from pools
- ▶ Raging debate over security
 - Neither protocol has ever been successfully attacked (though the [DAO hack](#) in 2016 was very disruptive to Ethereum)
 - Relates to how centralization tendencies stack up (of ETH holdings or of computational power)
 - PoW has a long track record of success under Bitcoin

Proof of authority

- ▶ Common among permissioned/private blockchains
- ▶ **Why?** Depends heavily on reputation and can assume a somewhat higher level of trust
 - Strict joining criteria
 - Valued relationships underpin threat of punishment or removal
 - Dominant or regulatory players may already be in place
- ▶ Get unambiguous finality very quickly with low energy use
 - But quite centralized

Proof of authority

PoA can be implemented in various ways (e.g. [Clique](#) or [Aura](#))

- ▶ A group of N validators is chosen
 - Process can survive a certain number of dishonest validators
 - In Clique that is $N/2 - 1$
- ▶ Authority to choose blocks rotates through the validators
 - Proposed block from 'leader' may be voted on by other validators
 - Votes may also occur to add/remove validators (reputation)
- ▶ Amount of messaging/rounds of communication among the validators will affect the speed with which finality is reached
 - **But massively faster than PoW/PoS**

Proof of authority

- ▶ PoA is an example of a Byzantine Fault Tolerant protocol

▶ Generals

- ▶ Other examples are Practical BFT and Istanbul BFT
- ▶ Such protocols are robust to different fractions of validators being dishonest or faulty
- ▶ Other protocols, such as Raft, may only be robust to nodes that crash, rather than being malevolent
- ▶ Some comparisons may be found [here](#), [here](#) and [here](#)

We will revisit these when we discuss pros and cons of Blockchains - and permissioned blockchains in finance and CBDC applications

Escape slides

Concept of 'state'

Consider a stationary ($|\rho| < 1$) Autoregressive Process of order 1:

$$y_t = \rho y_{t-1} + \epsilon_t$$

$$y_0 = \bar{y}$$

$$\epsilon_t \sim \mathcal{N}(\mu, \sigma^2)$$

What is the state at time t in this model?

- ▶ y_t
- ▶ y_{t-1} and ϵ_t
- ▶ $y_0, \epsilon_1, \epsilon_2, \dots, \epsilon_t$

All these are representations of the state. Now consider a more intuitive example. . .

Concept of 'state'

If you are a *banker*, and all you care about is a prospective borrower's net worth, the state might be:

- ▶ Just that one number
- ▶ The number last period, together with net income this period
- ▶ The number two periods ago, together with net income in the last two periods
- ▶ The value of her assets today, together with the value of her value of her debts
- ▶ The above numbers, but for all the divisions of her company separately
- ▶ Net worth of all the divisions of her company in the first period they were operating, and all the in-goings and out-goings since

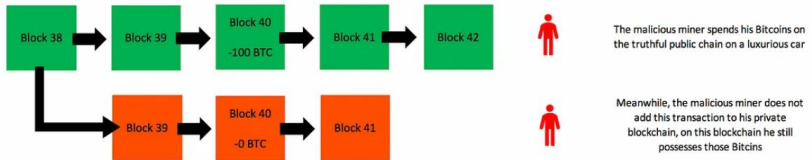
Type 2 double spend / 51% attack

Order of events:

1. Buy a good/service from another agent (a 'supplier')
2. Payment transaction gets added to a block and block is added to chain
3. Supplier provides good or service
4. Attacker quietly (without broadcasting - yet) works very hard to build an alternative chain
5. Alternative chain doesn't include the original transaction
6. Once chain is long enough, attacker broadcasts the fork
7. Reorgs take place and the alternative chain becomes the new accepted blockchain
8. Attacker has good/service and can still spend the same currency

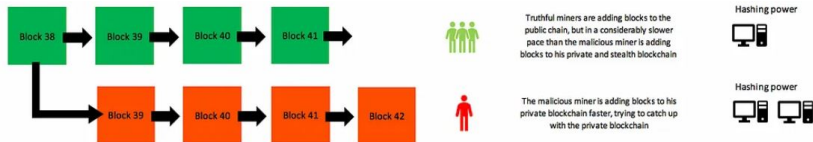
There is an *excellent* discussion [here](#) and we use some of its diagrams

Type 2 double spend / 51% attack



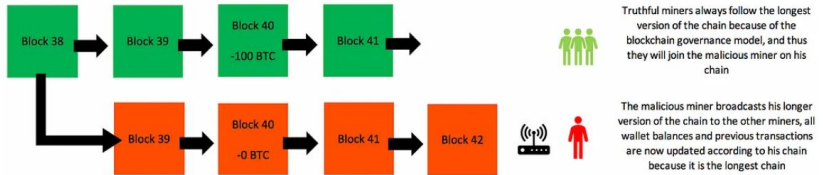
Source: [Medium](#) - May 5, 2018

Type 2 double spend / 51% attack



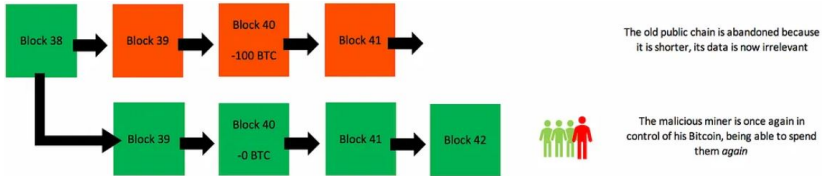
Source: [Medium](#) - May 5, 2018

Type 2 double spend / 51% attack



Source: [Medium](#) - May 5, 2018

Type 2 double spend / 51% attack



Source: [Medium](#) - May 5, 2018

► Back

Carrot and stick



Inactivity leak

There are also special penalties for inactivity in the case of difficulties of reaching consensus

- ▶ If no finalizations for 4 consecutive epochs (haven't reached 2/3 super majority of staked ETH), an 'inactivity leak' is implemented
- ▶ The inactive (at least 1/3 of validators) have their stake reduced gradually, *via* burning - this is the 'leak' of ETH (from the offending validators)
- ▶ Eventually they will represent $< 1/3$ of staked ETH and finalization can be re-established
- ▶ Happened for the first time in 2023 ([here](#) and [here](#))
- ▶ Emergency incentive system to ensure robustness of system - big incentive for validators to come (back) online

Proof of stake

Users continually create and sign transactions, before sending them to the network

- ▶ Nodes check transactions for validity, broadcast P2P (if valid), and add to mempool(s)

The process below occurs in every 'slot' (12 second intervals), with 32 slots (6.4 mins) constituting an 'epoch'

1. If chosen, a validator may propose and broadcast a block
2. The block is then checked by other validators
3. When a validator 'attests' positively, it adds the block to its view of chain

Proof of stake

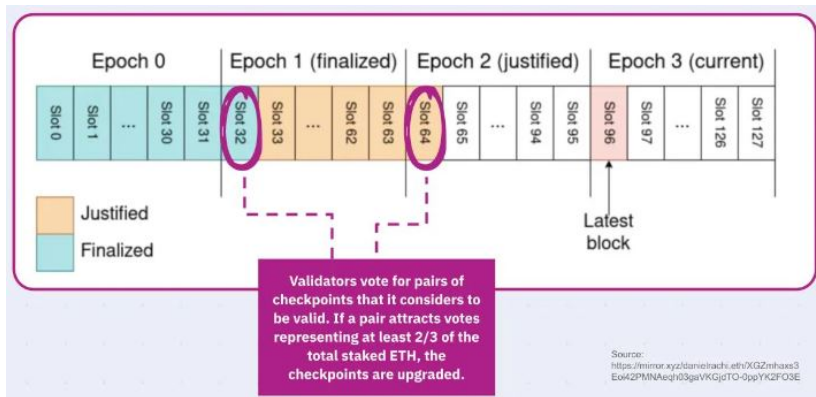
The first block in each *epoch* is called a 'checkpoint'

- ▶ If 2/3 of validators - **weighted by staked ETH** - vote in favour of a pair of consecutive checkpoints, the earlier of the pair is regarded as '**finalized**' and the later as '**justified**'

After 2 epochs there is 'no chance' of block reversion

- ▶ **Caveat:** Reversion *possible*, but would require expenditure of 1/3 of total stake of ETH (**vast** amount of money)
- ▶ More rapid finality than PoW would offer (recall reorgs. . .)
- ▶ And timing is regular (finality requires about 12-13 minutes)
- ▶ Great discussion of slots/finality [here](#)

Proof of stake



Source: [Blocknative](#) - August 3, 2023

Proof of stake

WARNING: The precise checkpoint, slot, epoch structure is not necessary for a PoS more generally. It is, however, key to the Ethereum implementation, and thus extremely influential

Proof of stake

In unlikely case of forks, Ethereum validators use 'heaviest chain'

- ▶ Based on number of validators, weighted by ETH staked

More complicated than PoW - and clearly more aspects to be 'tweaked'

Proof of stake

Validators in Ethereum PoS partly incentivised by rewards paid every epoch. . .

- ▶ Execution

- Tips/Priority fees
- Maximal Extractable Value (MEV)

- ▶ Issuance

- Proposal rewards
- Attestation rewards
- Sync committee rewards (won't discuss)
- Reporting violation of slashing rules (won't discuss)

The rewards are adjusted according to 'effective balance' and stakes of all validators (won't discuss)

Proof of stake

Validators in Ethereum PoS partly incentivised by rewards paid every epoch. . .

- ▶ Execution

- Tips/Priority fees (paid by network users to prioritize transactions for inclusion in block)
- [MEV](#) (complicated bargaining with searchers/builders who optimize the transactions included in block and then have the validator propose it - see also [here](#))

- ▶ Issuance

- Proposal rewards (1/7 of all rewards associated with block)
- Attestation rewards (must be correct and timely)

These rewards provide the return to staking - though how much you *receive* will depend on how you stake

Proof of stake

- ▶ Solo 'home' staking
 - If you have tech ability to run node and 32 ETH for validator
 - Most preferred as decentralizes system
 - No intermediary needs to be trusted
- ▶ Staking as a service
 - If you have 32 ETH but want someone else to handle admin
 - Possible concerns with centralization of validation
 - Trust [SaaS providers](#) they will validate appropriately
- ▶ Pooled staking
 - Can contribute fraction of 32 ETH to a pooling service
 - Partly helps decentralization, but large pools account for [large fractions](#) of staking amount
 - Must trust [pool admin](#) (e.g. [Lido](#))
- ▶ Centralized exchanges (CEX)
 - Can do staking for you using the ETH entrusted with them
 - Simple - but massive centralization and incentive for attacks

Proof of stake

We've discussed the carrot, what about the **stick**?

- ▶ **Penalties** (mainly penalizes laziness/incompetence)
 - Failure to vote - or doing it slowly
 - In some cases, literally penalties - in others (e.g. failing to propose when it's your turn) do not receive reward
 - Gradually grinds down the stake - once ≤ 16 ETH, validator status removed
- ▶ **Slashing** (mainly penalizes attempt to harm Ethereum)
 - For *more severe offences*, such as 'equivocating' (proposing or signing two blocks in a slot) or double voting
 - Max of 1/32 of staked ETH or 1 ETH is '**burned**' **daily**
 - Process continues for 36 days until validator is removed
 - *Additional* loss of ETH after 16 days, depending on the ETH staked by all validators are being slashed (worse penalties in case of a 'coordinated' attack)

Must reliably keep your system active **and** that you play by rules

Byzantine Generals Problem

A decentralized system is Byzantine Fault Tolerant if it can function despite the failure - or malice - of some of its nodes (well discussed [here](#) and [here](#))

- ▶ Its strange name derives from a famous thought experiment. . .

Consider a number of generals who are attacking a fortress. The generals must decide as a group whether to attack or retreat; some may prefer to attack, while others prefer to retreat. The important thing is that all generals agree on a common decision, for a halfhearted attack by a few generals would become a rout, and would be worse than either a coordinated attack or a coordinated retreat.

The problem is complicated by the presence of treacherous generals who may not only cast a vote for a suboptimal strategy, they may do so selectively. The problem is complicated further by the generals being physically separated and having to send their votes via messengers who may fail to deliver votes or may forge false votes.

Smart contract example: Piggybank

```
pragma solidity ^0.5.1;

contract PiggyBank
{
    address payable public owner;

    constructor() public
    {
        owner = msg.sender;
    }

    function() payable external {}

    function spend() public
    {
        require(msg.sender == owner);
        require(address(this).balance >= 1 ether);
        owner.transfer(address(this).balance);
    }
}
```

Smart contract example: Piggybank

```
pragma solidity ^0.5.1;
```

The first line controls which versions of Solidity should be assumed in compiling the source code

- ▶ Necessary since Solidity is a fairly new language that is rapidly changing

We then define the name of our contract, 'Piggybank'

```
contract PiggyBank{}
```

Smart contract example: Piggybank

```
constructor() public {}
```

This is the constructor function of the smart contract

- ▶ Will only be executed when the piggybank is deployed

```
owner = msg.sender
```

In the constructor, we set the variable owner to be the address of the sender of this message (the address that deployed the contract)

- ▶ We will use this variable to check for conditions later

Smart contract example: Piggybank

```
function() payable external {}
```

This is a default fallback function

- ▶ Does not need a name because owner can simply send ETH to the SC without a data component to the payload
- ▶ The 'payable' keyword \Rightarrow function can be used to accept funds to be deposited in the smart contract's account

Smart contract example: Piggybank

```
function spend() public {}
```

This defines the function named 'spend'

- ▶ Used to retrieve funds from the SC's account

```
require(msg.sender == owner);
```

The message sender must be the same address as the one that deployed the SC (set by the constructor function)

Smart contract example: Piggybank

Funds in the smart contract must be more than 1 ether.

```
address(this).balance >= 1 ether);
```

Also we have

```
owner.transfer(address(this).balance);
```

If the above two conditions are met, the smart contracts sends all of its funds to the owner address

- ▶ Empties the Piggybank
- ▶ Owner gets all the funds he saved back - to spend on whatever...