OSTTRA innovation

Session 1 - Week 1

Web3 Labs

Week 1

Goal at end of week:

- Read up on relevant topics, material and links as provided
- Install and run Geth locally

What are blockchains?

A blockchain is a chain of activity, with each activity typically described as a ‘transaction’. These transactions are grouped into ‘blocks’, with each block referencing the previous block. This creates a chain of changes, which together represents the current ‘world state’.

Blockchains typically consist of a ‘network of nodes’, where these nodes through a ‘consensus algorithm’ together come to agreement about what this world state should be. In this network we usually find one node proposing a new block with a number of transactions in it, and the rest of the network will verify the correctness of said block. If valid, it will be included in the chain.

Which node that gets to propose the next block depends on the consensus algorithm, and can be random or preselected based on this. Invalid blocks are at minimum ignored, but the offending node might be punished by the other nodes in the network, depending on the network setup and consensus algorithm.

The smallest possible blockchain network consists of just one node, but usually they are bigger. The aim of having a bigger network is to avoid any one node being able to take over or corrupt the network with invalid blocks.

Some key terms to understand

Transaction: Describes a change to the ‘world state’

Blocks: A grouping of ‘transactions’ proposed by a ‘node’

World state: The sum of all ‘transactions’ if we apply each ‘block’ in sequence

Node: Software that knows how to process ‘transactions’ and ‘blocks’ and connect to other nodes

Network of nodes: Two or more nodes that know how to connect with each other over a network

Consensus algorithm: The system that allows the nodes in the network to cooperate

Recommended reading material + slides:

https://www.codecademy.com/resources/blog/what-is-blockchain/

https://ethereum.org/en/learn/

Blockchain tech

Diving into some details

Blockchains are part of Web3

Many technologies are part of the Web3 umbrella, blockchains being one of them.

Blockchains are key to this because:

- They remove the need for a gatekeeper (through decentralisation)
- They allow me to efficiently find, and securely interact with, you (enabling peer-to-peer)
- They function as trustless intermediaries (and arbitrator)
- They enforce digital rules (with smart contracts)

Blockchains are networks of nodes

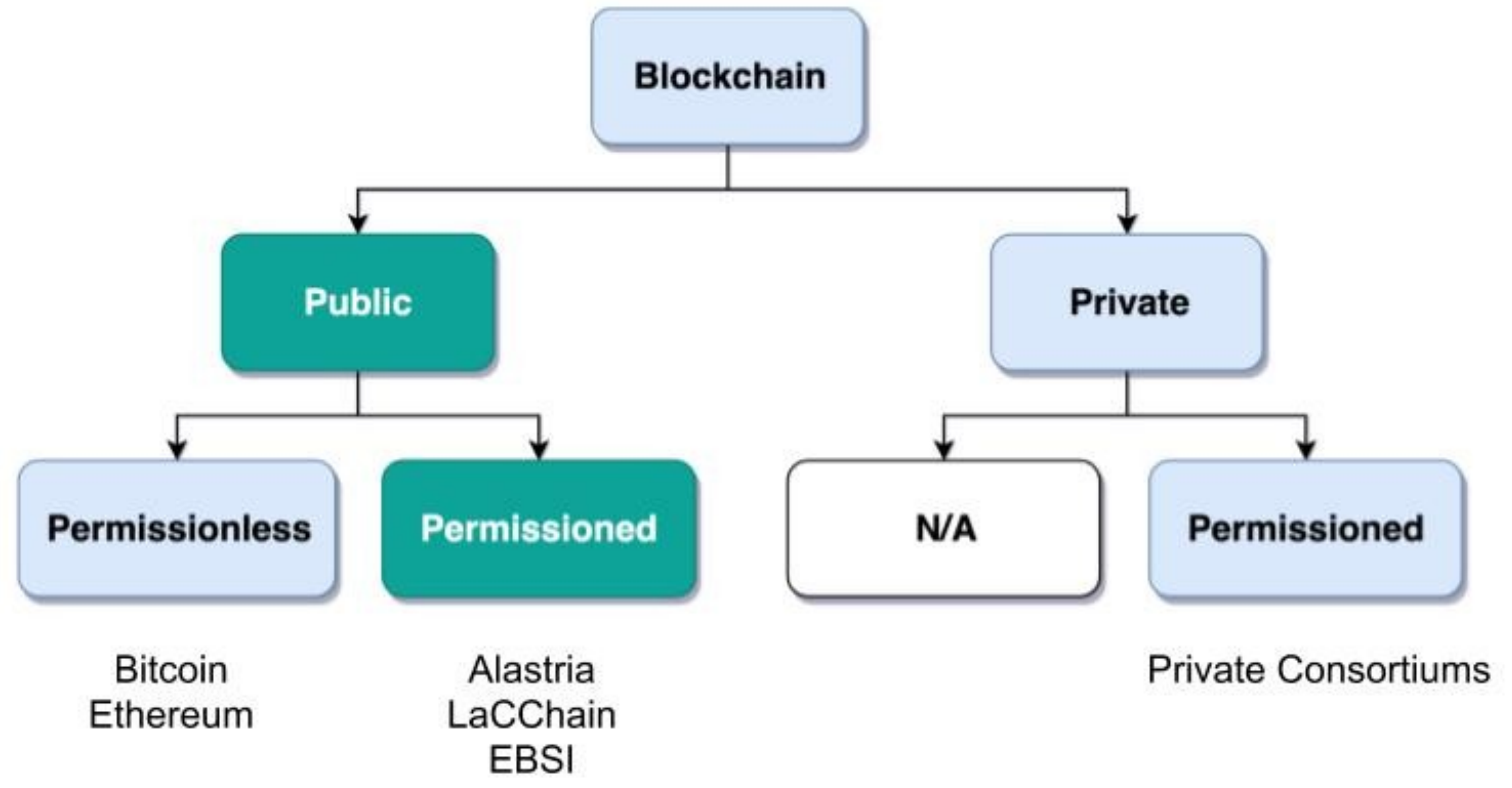
The nodes (software/hardware), through a shared protocol, agree on what's is allowed to take place on a shared ledger.

Who is allowed to join the network?

Who is allowed to execute transactions on the network?

Depending on how you answer these questions, you get different types of blockchain networks, known as:

- Public permissionless
- Public permissioned
- Private permissioned



https://www.linkedin.com/pulse/public-permissioned-blockchains-common-pool-resources-jesus-ruiz/

Blockchains are not a database replacement

Blockchains have a lot of limitations (when views with the wrong lens), and represent a new type of systems.

It takes time to understand this, and people often try to fit other labels in their attempt to understand how to use them.

It’s also a quickly evolving area, with new technology being developed to address current limitations and new use cases.

These will over time address:

- Scalability, i.e., how many transactions per second, or how much bandwidth can they support?
- Privacy, i.e., how public will my transactions be? Or how private will my identifier be?
- Storage, i.e., how can a blockchain manage an every increasing world state and related history?

Solving these are not easy, and many take shortcuts when trying to solve for them. This can have negative implications for the decentralisation or security requirements of a public permissionless blockchain.

The blockchain trilemma

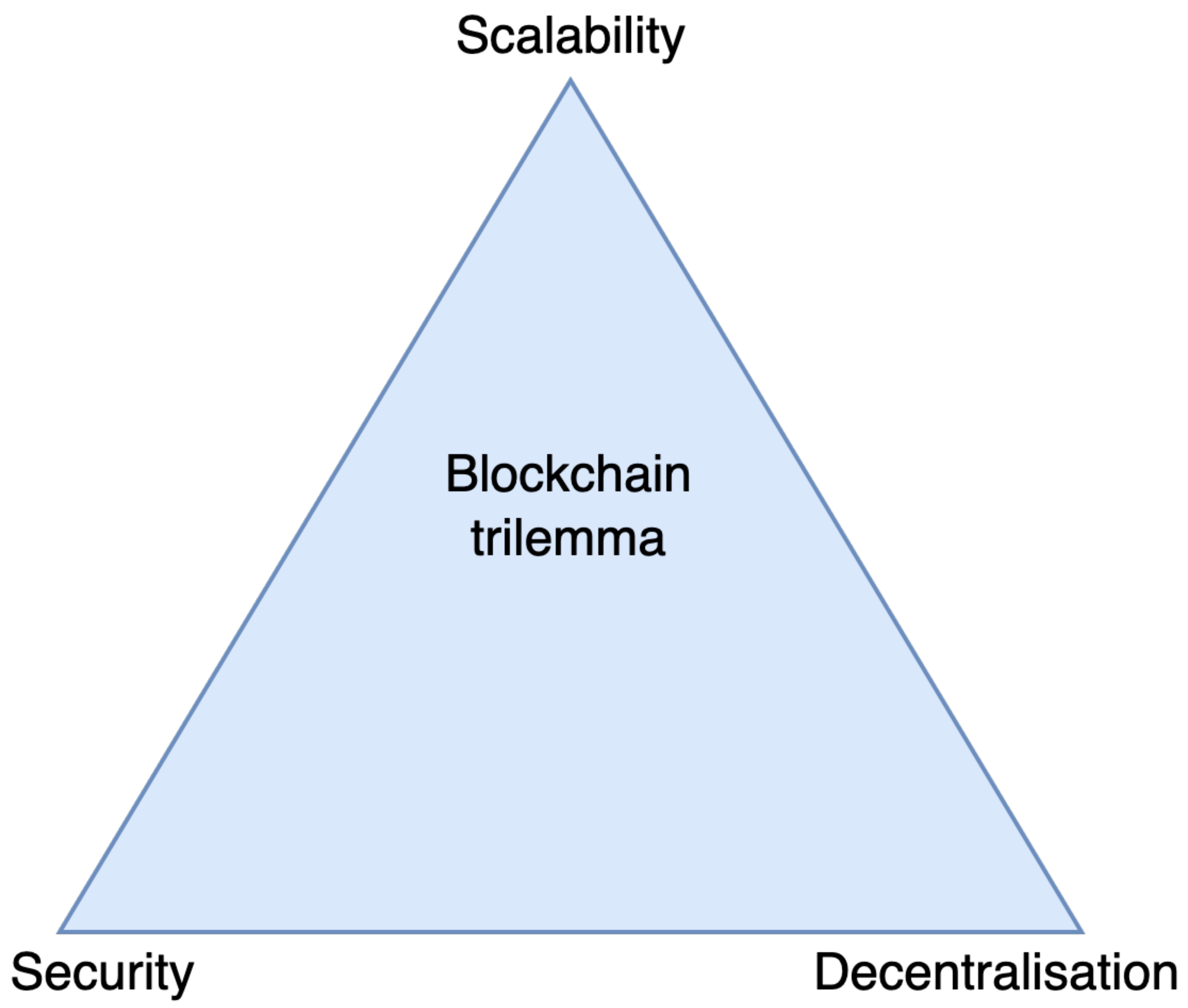
Ideally you want all 3:

- Scalability (transactions per second)
- Security (immutability, tamperproof)
- Decentralisation (no gatekeeper)

But often you have to compromise on one to get the other two.

Bitcoin and Ethereum prioritise decentralisation and security over scalability.

Current efforts are working on increasing scalability while maintaining or improving current levels of security and decentralisation.



How we get security — The consensus layer

Blockchain security is about the ability to remain immutable / tamperproof.

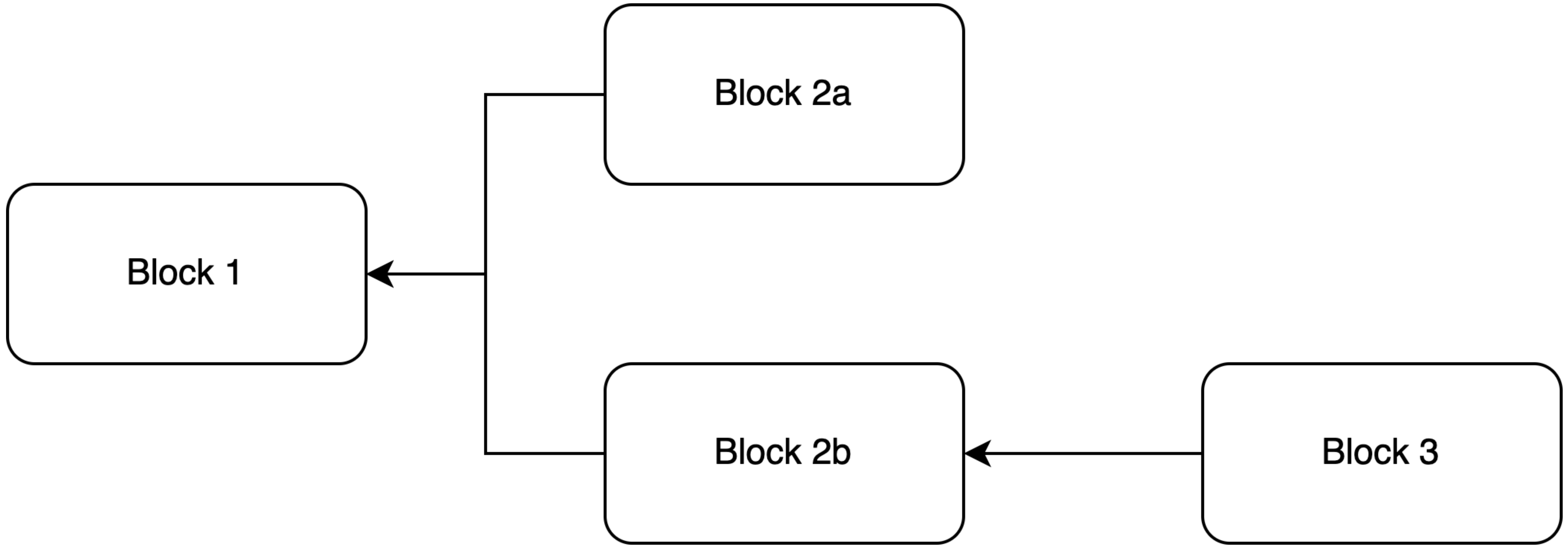
It means I can have certainty that my transaction / activity isn't going to be reverted by an attacker.

In general we have two consensus types to ensure this type of security:

- Proof of Work (PoW)
- Proof of Stake (PoS)

We also have others, like Proof of Authority (PoA), but these only apply for private/permissioned blockchains, where you trust all the node operators through other mechanisms.

How we get security — PoW



How we get security — PoW

One node propose a block, and upload this to all the nodes it is connected to. A block contains the transactions, selected from all available unprocessed transactions that this node is aware of.

The nodes that received the block verify the block (only new and valid transactions, etc), making sure it follows the protocol rules, and upload it to other nodes that they again are connected to. Through this process, valid blocks are propagated ‘virally’ to the whole network. Each block contain a reference to the previous block, creating a chain..

The above is the basics of any blockchain node network and how blocks are included. But how do we make this work in practise?

If all the nodes in a big network could propose a block, the network would spam itself. It would not be able to agree on a single valid block because there would be too many valid proposals. Proof of Work is a decentralised method of slowing everything down. A proposed block is only valid if it included some form of proof that the node did some work on it.

Through this process, no single node can control what blocks are included, because PoW randomises what node can propose a block, and slows it down to a rate the network can manage. And for each valid block, it becomes increasingly difficult to challenge its validity, because the amount of work needed is added per block.

How we get security — PoS

PoW is easy to implement, it’s a rather simple algorithm. But it’s very wasteful, by design, causing computers and miners to spend a lot of energy finding (mining) ‘the secret value’ needed to propose the next valid block.

Proof of Stake removes the mining need, by going through a complex process of electing the next validator. At each block, through a predefined process, a random validator is selected to propose the next valid block. This removes the need for using a lot of energy.

If the validator is online, and proposes a valid next block, the other validators include this block in the chain, and the validator is rewarded.

If the validator is offline, or otherwise misbehave by trying to propose an invalid block, the other validators punish the incorrect behaviour by removing some of the validator collateral. The invalid block is not included by the other validators.

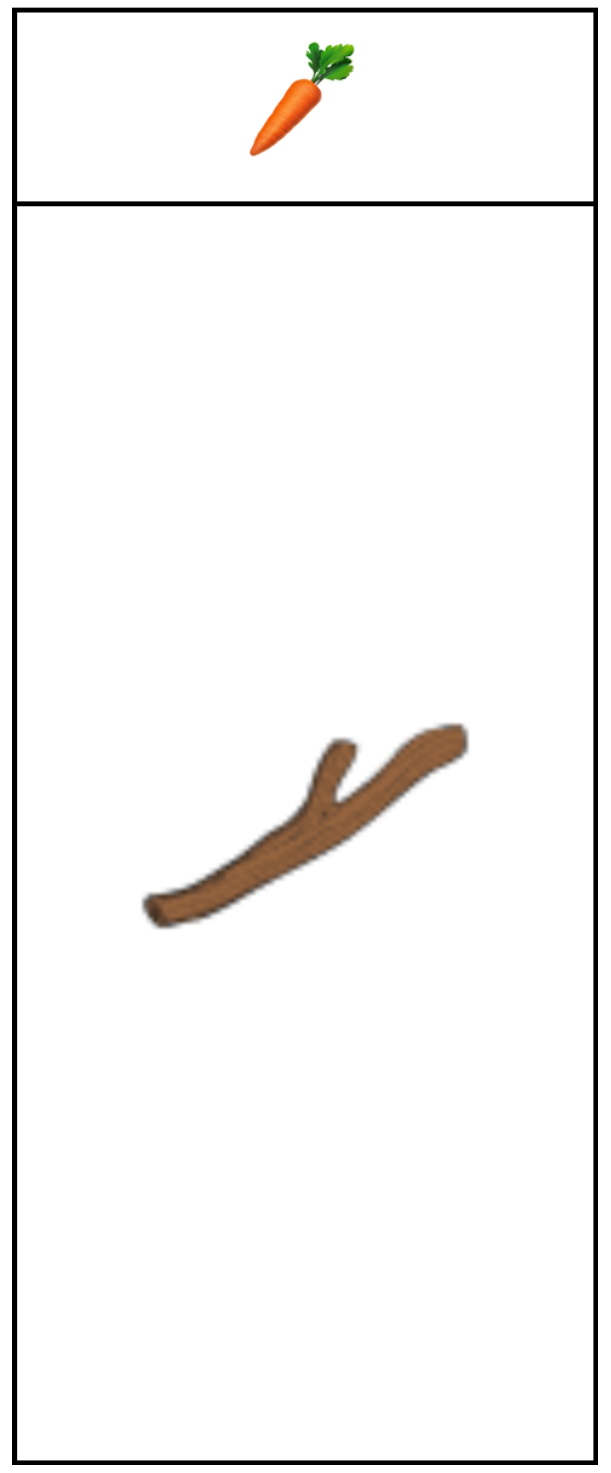
PoS overall better than PoW

Even with added complexity and some criticism around the capital lock up of PoS, it is overall beneficial to the security of a blockchain because it removes some economies of scale, hence encouraging broader participation.

PoW only provide the carrot to those that run miners, while PoS also provide a stick against those that try to misbehave by taking away collateral.

We can expect Ethereum to move from PoW to PoS in 2022.

Note that PoS does not in itself provide much scalability benefits, but can be used to build other scalability solutions afterwards.



How we get scalability — The execution layer

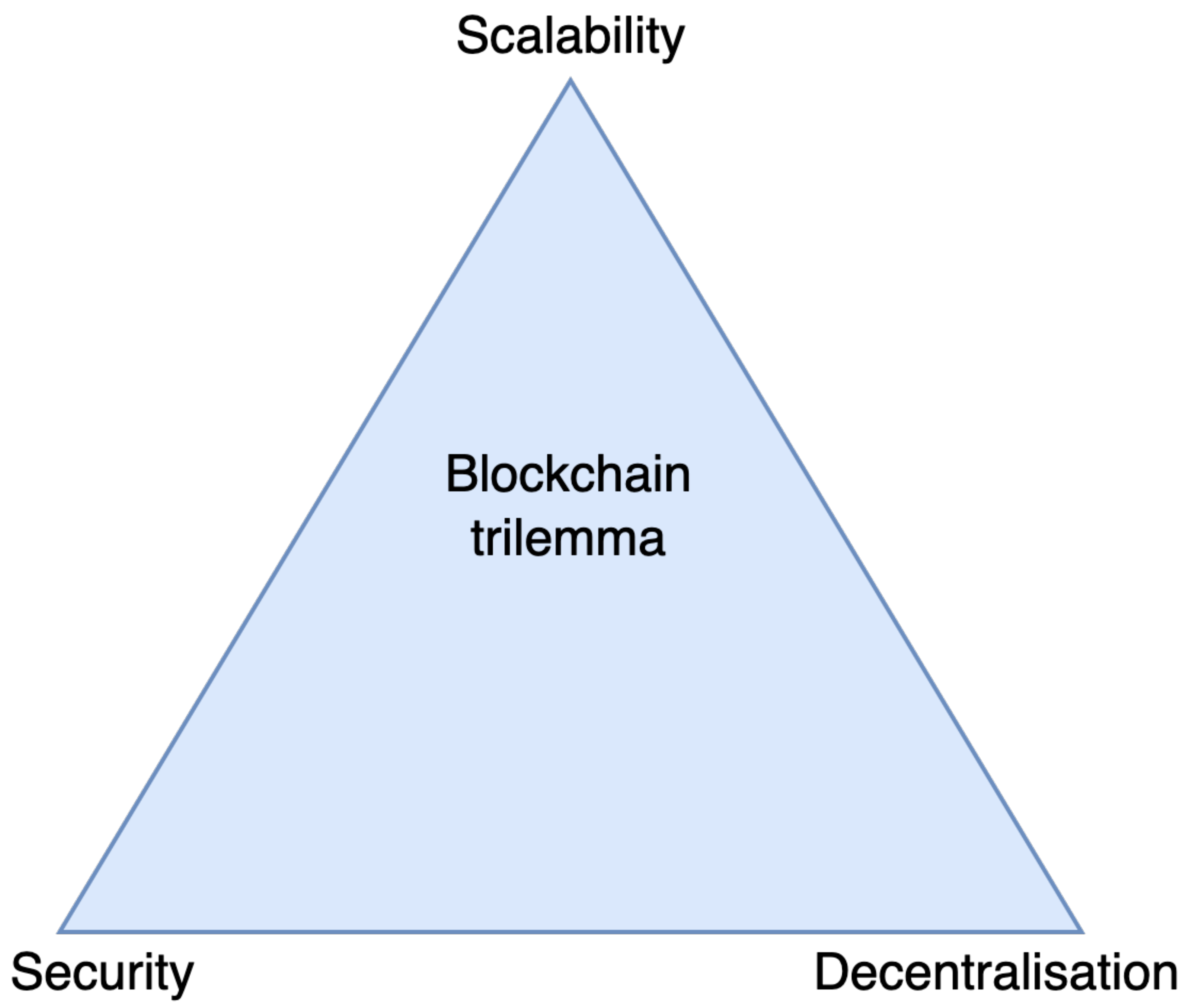
Scalability is about increasing the number of transactions per second. This can also be called bandwidth.

Why is scalability difficult in a blockchain?

The number transactions you can process depends on how many transactions fit into a block. If you make a block too big, to fit more transactions in it, you risk needing too much time to upload the block to all the other nodes.

Reducing the number of nodes could allow for bigger blocks, but then you are reducing security and/or decentralisation.

Again we are faced with the blockchain trilemma.



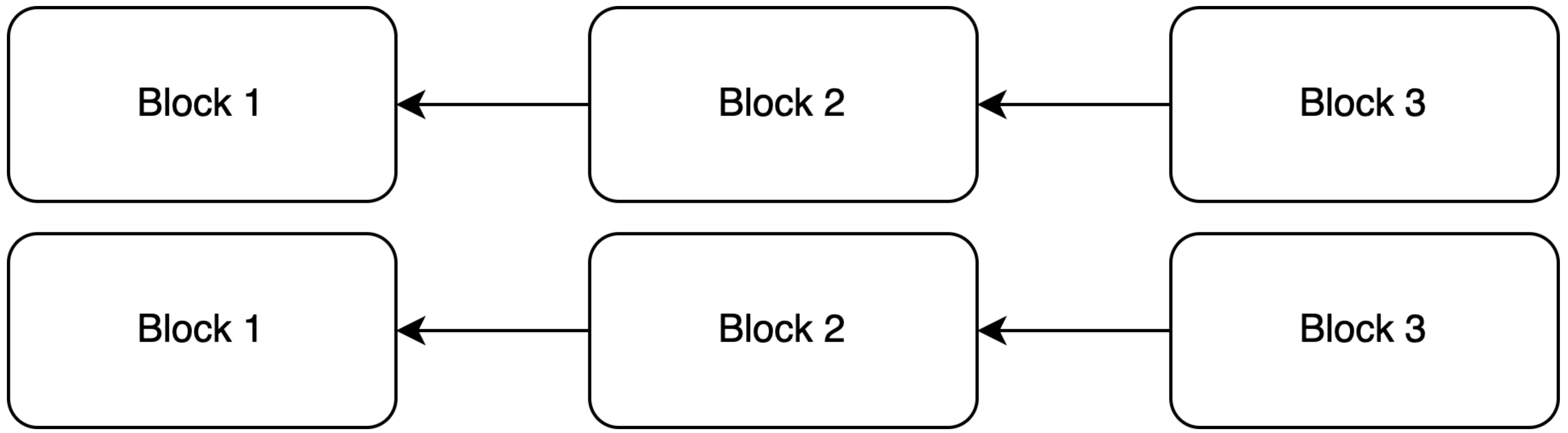
Scaling through multiple blockchains

If one blockchain can do 10 transactions per second, why not have two, so we can do 20 transactions per second?

This is currently the go-to solution in order to provide more bandwidth, but it comes with some drawbacks. Imagine what would happen if we needed 1000 TPS total bandwidth, and each blockchain could only support 10 TPS. We'd then need 100 blockchains.

Across these 100 independent blockchains, we'd need to spread our resources. This risks us spreading the security and decentralisation efforts too thin, making it too easy to attack.

While it might be the current solution, it's not a viable long term solution for this reason.



Scaling through sharding

Sharding is much like having multiple blockchains, but they all share one consensus layer.

This then allows us to concentrate all the security and decentralisation efforts on one blockchain, but still enable many more transactions per second in this blockchain.

Ethereum will introduce sharding, and it will likely happen in 2023/2024, with additional features added to this over time.

Scaling through layer 2

In the blockchain world, we often speak of layers. Typically, when people talk about layer 1, they talk about the main Ethereum chain. There’s also a layer 0, which is the governance layer, i.e., the humans working together to define the protocol and their policies/governance around this.

Layer 2 is a scaling solution that periodically upload changes to layer 1. Imagine you and me trade back and forth 100 Ether. If I first send you 100, then you send me back 50, before I send you 25, the final ledger view on this is that I hold 25 Ether and you hold 75 Ether.

In simplified terms, what a layer 2 does is to periodically upload this change to the layer 1 blockchain. Because it doesn’t need to upload everything, while still being able to upload enough to prove who owns what, it is able to compress the amount of data we need to upload. Because it's compressed, we can get more transaction in per data upload, hence improving the bandwidth. Also because of that, the individual transaction costs go down on the layer 2 solution.

Because data is uploaded back to the layer 1 blockchain, if the operator of the layer 2 solution shut down tomorrow, someone could pick up from where they left off. This is why we say that a layer 2 gives us layer 1 security.

Other reasons for a layer 2

As Ethereum introduces sharding, there will over time potentially be less need for layer 2 solutions. Layer 2s still add scaling capabilities, even after sharding, which is always welcome, but it's a less pressing issue.

There are however other reasons for using a layer 2 besides just scaling, and this is why we believe layer 2 solutions will remain interesting.

Layer 2s change the trilemma dynamics, by allowing for scaling while still maintaining security (through layer 1 and validators) and decentralisation (through data availability on layer 1 and multiple operators). Because of this they are also well positioned to enable technology which is difficult to implement directly on a layer 1.

One example of this is privacy. While it might be difficult or impossible to have sufficient privacy on Ethereum layer 1 directly, while still maintaining all the features it offers, a layer 2 can offer this instead. We can then move tokens to a layer 2, where the internal movement and ownership of these tokens are not publicly available.

We can say that the layer 2 encrypts the details, while still maintaining the security and decentralisation values of the layer 1.

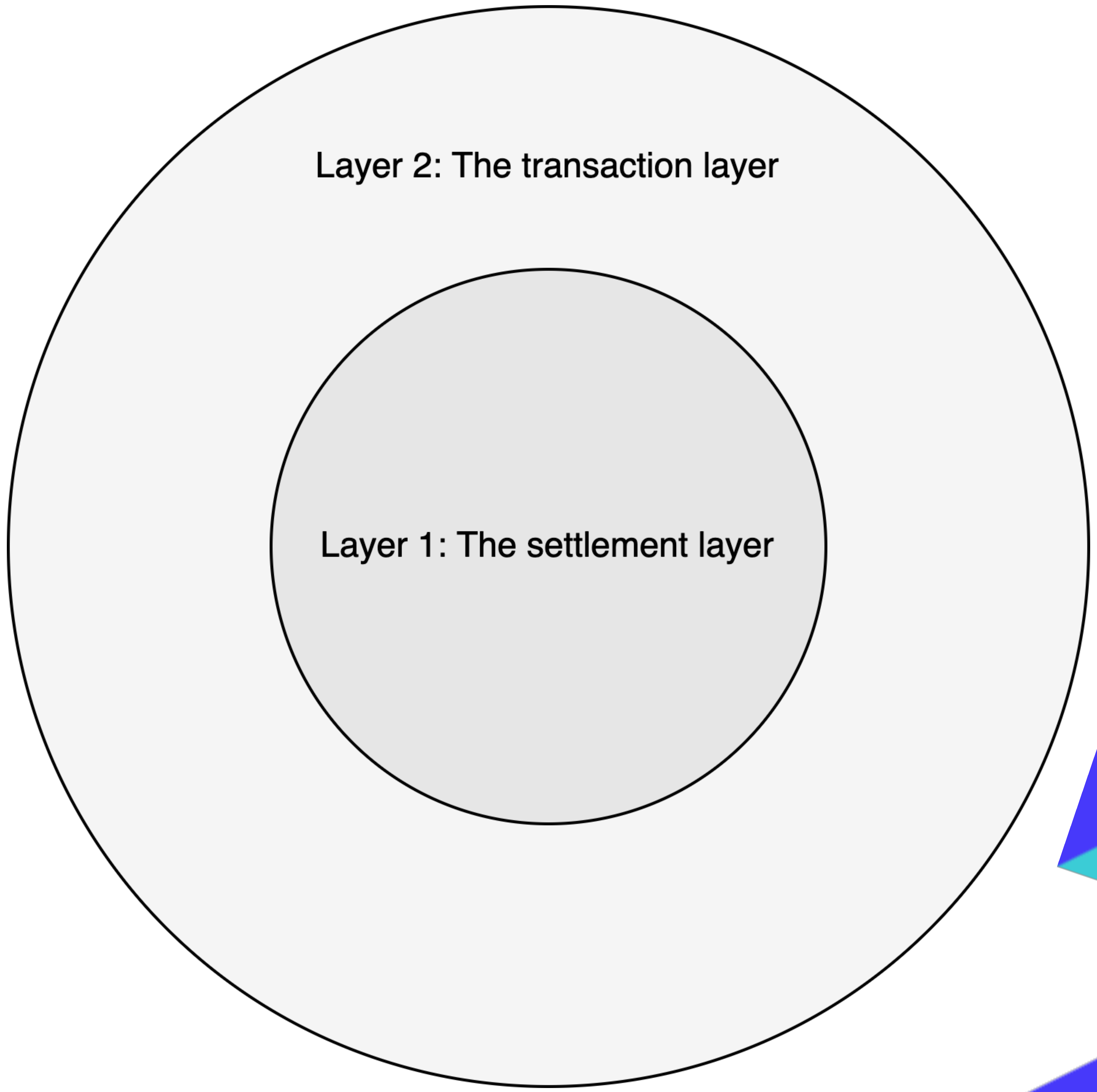
Multiple layers

Ethereum layer 1 is the settlement layer, while layer 2s represent the transaction layer.

A layer 2 operator and solution compete among other layer 2s to add additional features and functionality.

Because they all link the data back to the layer 1 network, you gain the security and decentralisation of that inner layer.

Then you can pick and choose which layer 2 caters to your needs based on technology, performance, cost, liquidity and so on.



Layer 2 implementations

Layer 2 technology is currently at the edge of what's technically possible within zero knowledge proofs and related research. It's a fast changing landscape with a lot of resources backing the various efforts.

Below we summarise some of the more well known or interesting efforts within layer 2 technology as of right now.

Layer 2s for scalability:

- Arbitrum and Optimism (technology: optimistic rollups)
- zkSync, Polygon Miden, StarkWare, Scroll and others! (technology: zk-based rollups)

Layer 2s for privacy:

- Polygon Nightfall and Aztec

Summary of blockchain developments

Based on previous slides we expect to see:

- Ethereum moving from PoW to PoS in 2022 (known as The Merge), preparing it for sharding
- Ethereum getting sharding in 2023/2024, dramatically increasing bandwidth and lowering transaction fees
- We expect many different layer 2 solutions to introduce interesting new technology as they compete among themselves
- Because most layer 2s will be linked with Ethereum as their layer 1, this brings further consolidation
- We expect to see some other separate blockchains for niche use cases or special interest groups

Homework

Install Geth and get it running

Geth

As stated earlier, blockchain networks consists of blockchain nodes. Different blockchains have different implementation, and some, like for Ethereum, have many largely compatible implementations too.

Your homework this week is to install Geth, the most well known Ethereum node implementation.

Installing and running a node is the first step towards joining a network, but we are not here concerned about joining any network. Instead, we want to simply run a single node locally, in order to simulate the most realistic use of a blockchain network. Therefore, we only want this node to run locally in developer mode.

Bonus homework: If you find this task very straight forward, research online how you can join an Ethereum test network with your own node.

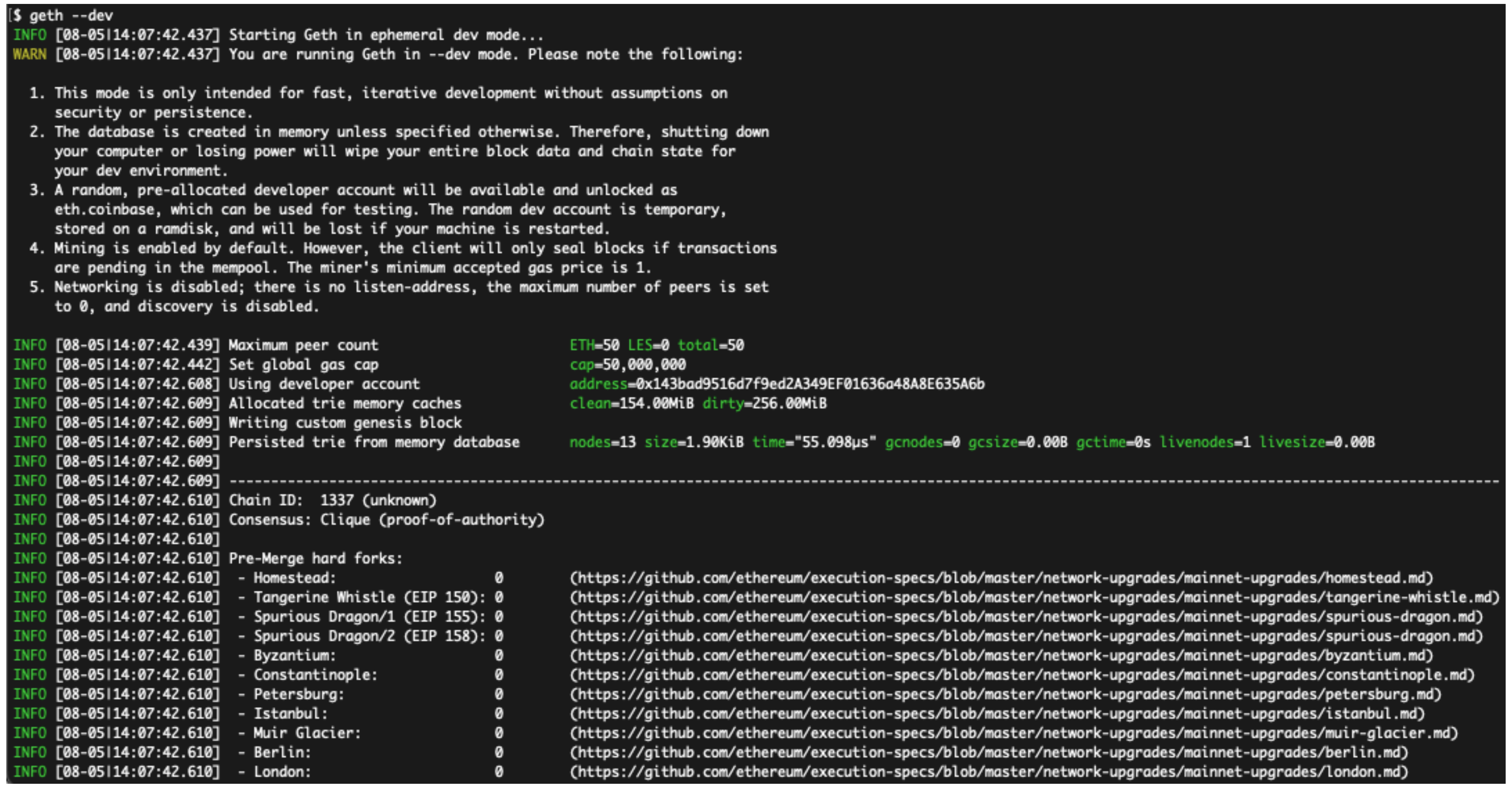
Geth

Step 1: Follow installation instructions relevant for your setup:

https://geth.ethereum.org/docs/install-and-build/installing-geth

Step 2: After installation, you should be able to run the command below to start geth in local dev mode, with an expected output similar to what's shown in the terminal screenshot.

```
geth --dev
```



```
[$ geth --dev
INFO [08-05|14:07:42.437] Starting Geth in ephemeral dev mode...
WARN [08-05|14:07:42.437] You are running Geth in --dev mode. Please note the following:

  1. This mode is only intended for fast, iterative development without assumptions on
     security or persistence.
  2. The database is created in memory unless specified otherwise. Therefore, shutting down
     your computer or losing power will wipe your entire block data and chain state for
     your dev environment.
  3. A random, pre-allocated developer account will be available and unlocked as
     eth.coinbase, which can be used for testing. The random dev account is temporary,
     stored on a ramdisk, and will be lost if your machine is restarted.
  4. Mining is enabled by default. However, the client will only seal blocks if transactions
     are pending in the mempool. The miner's minimum accepted gas price is 1.
  5. Networking is disabled; there is no listen-address, the maximum number of peers is set
     to 0, and discovery is disabled.

INFO [08-05|14:07:42.439] Maximum peer count                       ETH=50 LES=0 total=50
INFO [08-05|14:07:42.442] Set global gas cap                       cap=50,000,000
INFO [08-05|14:07:42.608] Using developer account                  address=0x143bad9516d7f9ed2A349EF01636a48A8E635A6b
INFO [08-05|14:07:42.609] Allocated trie memory caches             clean=154.00MiB dirty=256.00MiB
INFO [08-05|14:07:42.609] Writing custom genesis block
INFO [08-05|14:07:42.609] Persisted trie from memory database      nodes=13 size=1.90KiB time="55.098µs" gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [08-05|14:07:42.609]
INFO [08-05|14:07:42.609] ---------------------------------------------------------------------------------------------------------------------------------------------------------
INFO [08-05|14:07:42.610] Chain ID:  1337 (unknown)
INFO [08-05|14:07:42.610] Consensus: Clique (proof-of-authority)
INFO [08-05|14:07:42.610]
INFO [08-05|14:07:42.610] Pre-Merge hard forks:
INFO [08-05|14:07:42.610]  - Homestead:                   0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/homestead.md)
INFO [08-05|14:07:42.610]  - Tangerine Whistle (EIP 150): 0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/tangerine-whistle.md)
INFO [08-05|14:07:42.610]  - Spurious Dragon/1 (EIP 155): 0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [08-05|14:07:42.610]  - Spurious Dragon/2 (EIP 158): 0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [08-05|14:07:42.610]  - Byzantium:                   0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/byzantium.md)
INFO [08-05|14:07:42.610]  - Constantinople:              0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/constantinople.md)
INFO [08-05|14:07:42.610]  - Petersburg:                  0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/petersburg.md)
INFO [08-05|14:07:42.610]  - Istanbul:                    0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/istanbul.md)
INFO [08-05|14:07:42.610]  - Muir Glacier:                0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/muir-glacier.md)
INFO [08-05|14:07:42.610]  - Berlin:                      0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/berlin.md)
INFO [08-05|14:07:42.610]  - London:                      0        (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/london.md)
```

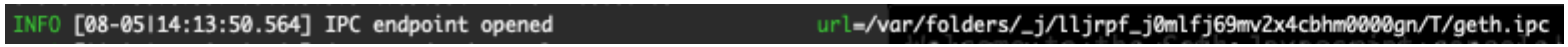
Connecting to Geth

In week 2 we'll start looking at smart contracts, and those will then be deployed to your local Geth node.

Until then, you are encouraged to play around with the Geth JavaScript console. In a different terminal, while having geth running in the first terminal from previous slide, you should be able to connect with the 'geth attach <IPC path>' command as shown on:

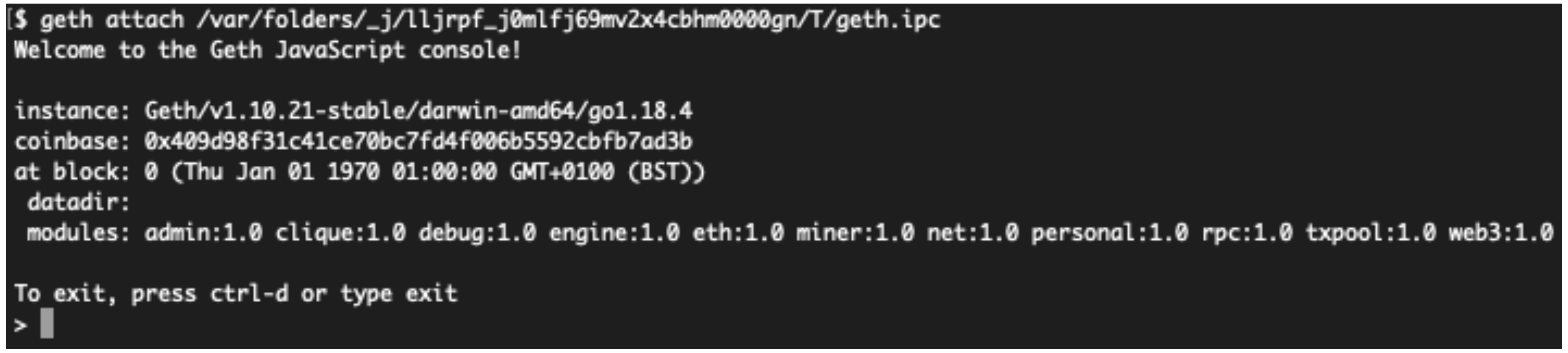
https://geth.ethereum.org/docs/interface/javascript-console

The IPC path is part of the output you get when starting 'geth --dev', looking something like the below:



```
INFO [08-05|14:13:50.564] IPC endpoint opened                      url=/var/folders/_j/lljrpf_j0mlfj69mv2x4cbhm0000gn/T/geth.ipc
```

You url will be different, but here I'd run the below to connect:



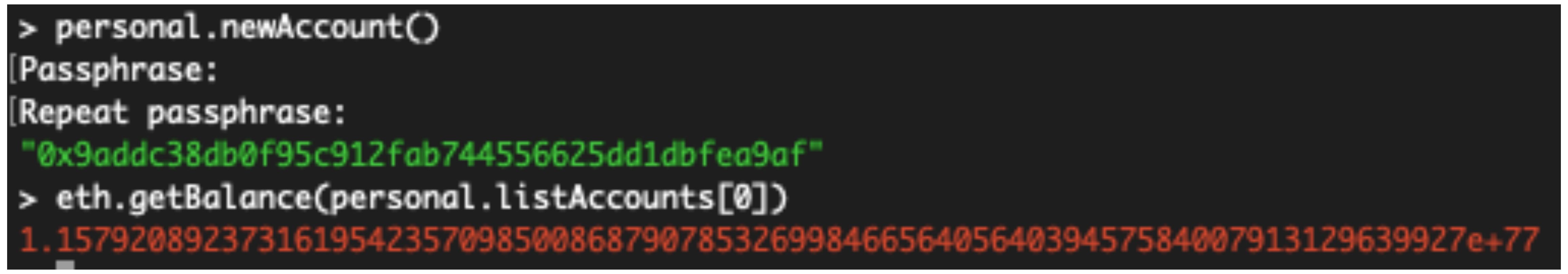
```
$ geth attach /var/folders/_j/lljrpf_j0mlfj69mv2x4cbhm0000gn/T/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.21-stable/darwin-amd64/go1.18.4
coinbase: 0x409d98f31c41ce70bc7fd4f006b5592cbfb7ad3b
at block: 0 (Thu Jan 01 1970 01:00:00 GMT+0100 (BST))
 datadir:
 modules: admin:1.0 clique:1.0 debug:1.0 engine:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> 
```

Web3 Labs

Creating an account

Because you're on your local node in dev mode, you can create new account prefunded with local test Ether, which will be useful when deploying contracts and making transactions in week 2.

As outlined on https://geth.ethereum.org/docs/interface/javascript-console you can do this as shown:



```
> personal.newAccount()
[Passphrase:
[Repeat passphrase:
"0x9addc38db0f95c912fab744556625dd1dbfea9af"
> eth.getBalance(personal.listAccounts[0])
1.15792089237316195423570985008687907853269984665640564039457584007913129639927e+77
```

Two commands are shown, 'personal.newAccount()' requiring you to pick a passphrase, and 'eth.getBalance(personal.listAccount[0])', both of which are actually JavaScript code executed within the geth client.