ETHEREUM SMART CONTRACT DEVELOPMENT

Solidity – Remix – Hardhat
Ethers.js – OpenZeppelin – Pinata

Course Overview 1/2

➢ The basics: The Ethereum Network, Blockchain, Ethereum accounts, transactions, ETH, wei & gas...

➢ Metamask: Setup, configuration...

➢ Remix: compiling, deploying and debugging smart contracts

➢ Hello World: Your first smart contract

➢ Hardhat: Setting up a local development environment to compile, test and deploy SC's

➢ Solidity: Types (struct, mappings...), functions (receive, fallback...), modifiers, events, error handling, inheritance...
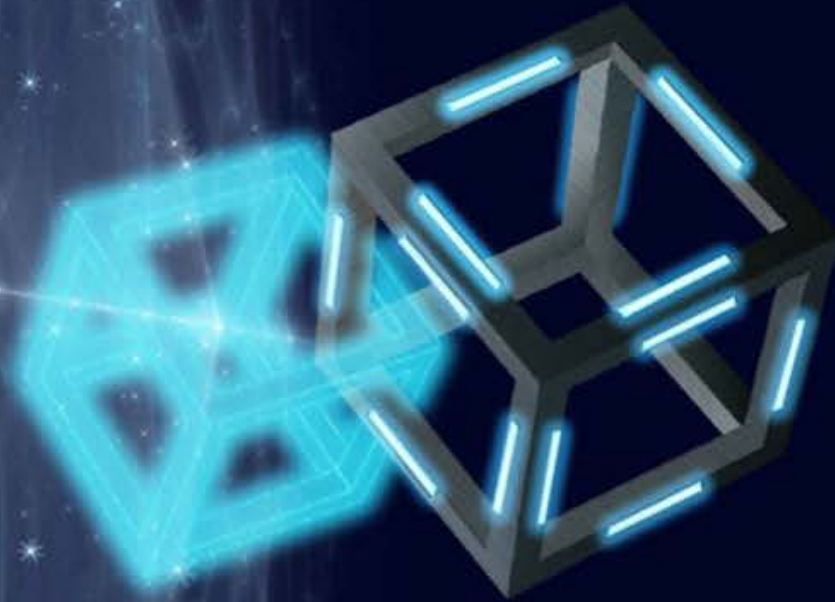
➢ Voting smart contract

Course Overview 2/2

➢ Creating a web front-end using React and Ethers.js: Provider, Signer, Contract, Utils

➢ The Metamask RPC API

➢ Creating an ERC20 token using OpenZeppelin

➢ Testing smart contracts in Hardhat using Mocha and Chai

➢ Creating ERC721 NFT's with OpenZeppelin, IPFS and Pinata

➢ Protecting your code against a reentrancy attack

➢ Creating a Decentralized Autonomous Organization (DAO) => Governance token, treasury, timelock and governor contract

# Slides, documents, source code…

Download from : https://github.com/rspadinger/ETH-Course

Introduction To Ethereum Smart Contract Development

**The Basics:** Network, Blockchain, Accounts...

The Ethereum Network

➢ Started in 2013 by Vitalik Buterin – first production release in 2015

➢ Ethereum: Open-source, public, decentralized, permissionless, immutable blockchain

➢ Allows deployment and execution of Turing complete smart contract code (mostly Solidity)

➢ P2P network – 12000+ nodes running Ethereum clients (Geth, Erigon...)

➢ Node is a gateway for applications and users to access and modify data on the blockchain

➢ Nodes verify transaction data and send it to other connected nodes => until all nodes have the same information

➢ 3 types of nodes: Full nodes, archive nodes and light nodes
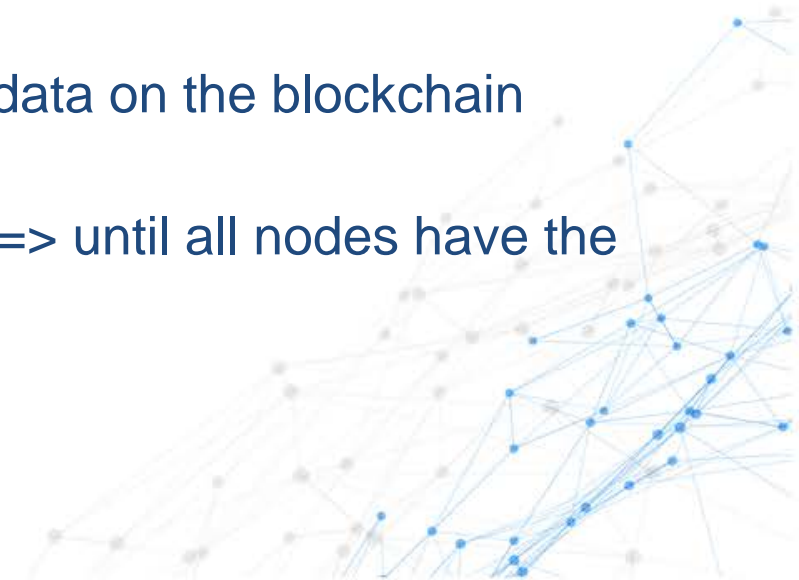
The Ethereum Network – Node Types

Full Nodes:

➢ Store most recent state of the blockchain (last 128 blocks)

➢ Can query blockchain data

➢ Can process transactions(send ETH, execute SC functions) and validate new blocks

➢ SC's can be deployed directly to a full node

➢ Requires ~ 2 TB of disk space
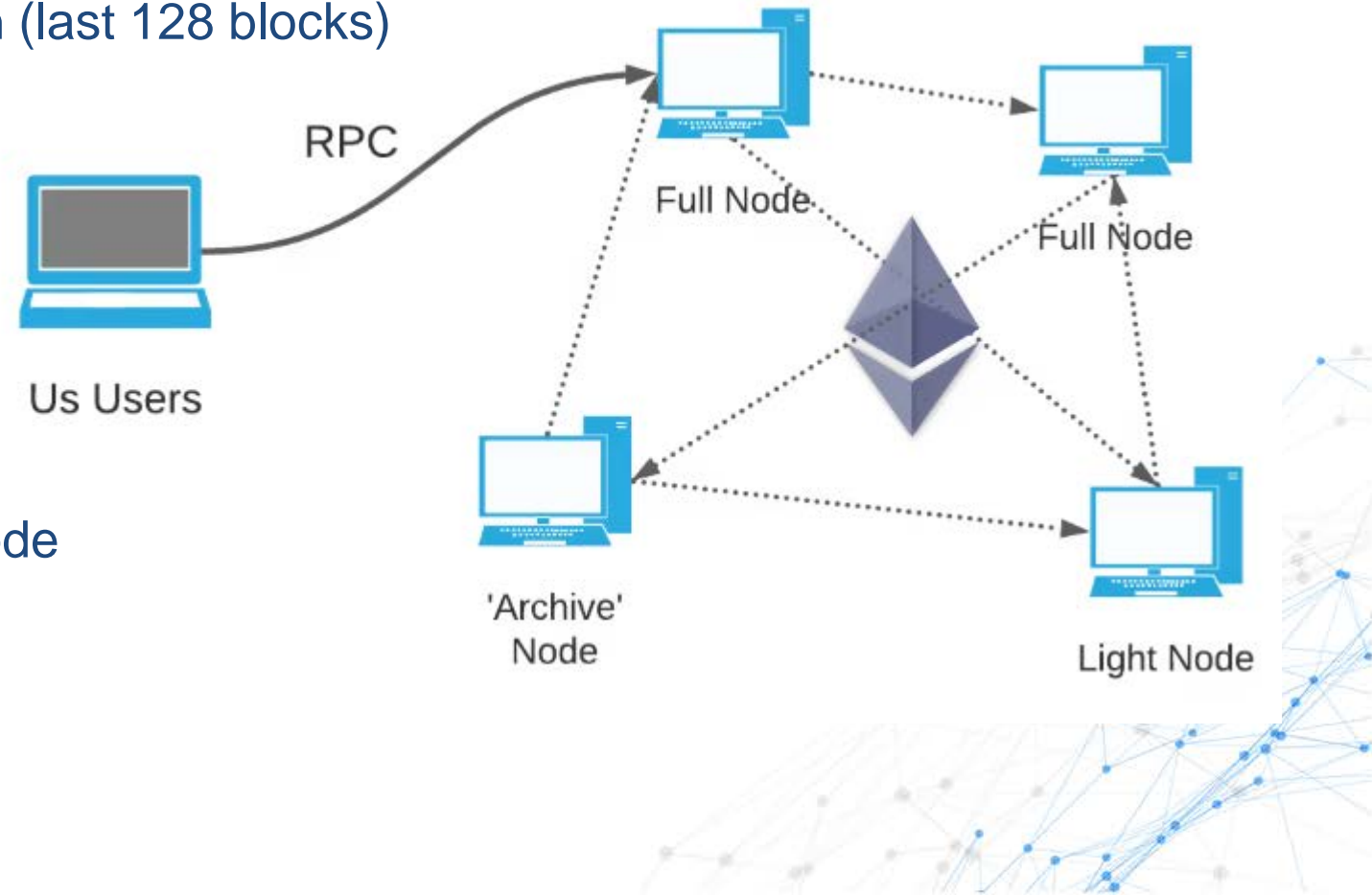
The Ethereum Network – Node Types

Archive Nodes:

➢ Store complete historical data of the blockchain (required by blockchain explorer)

➢ Requires ~ 20 TB of disk space

Light Nodes:

➢ Stores only block headers (direct access only to minimal data – blockNumber, timestamp...)

➢ Often holds data about a single account (wallet app)

➢ Requires full node to get additional data or send transactions

➢ Do not participate in block validation

➢ Requires ~ 400 MB of disk space

The Ethereum Network – Connection

Connecting to the Ethereum Network:

➢ Using a plugin like Metamask – for consumers

➢ Using a blockchain explorer (https://etherscan.io)

➢ Using a library like ethers.js or web3.js – for developers

➢ Using a third party provider like Alchemy, Infura...

➢ Running your own node by installing a client like Geth, Prysm...

The Ethereum Network – Types of Networks

Main Network

Real blockchain

Persistent

Costs money

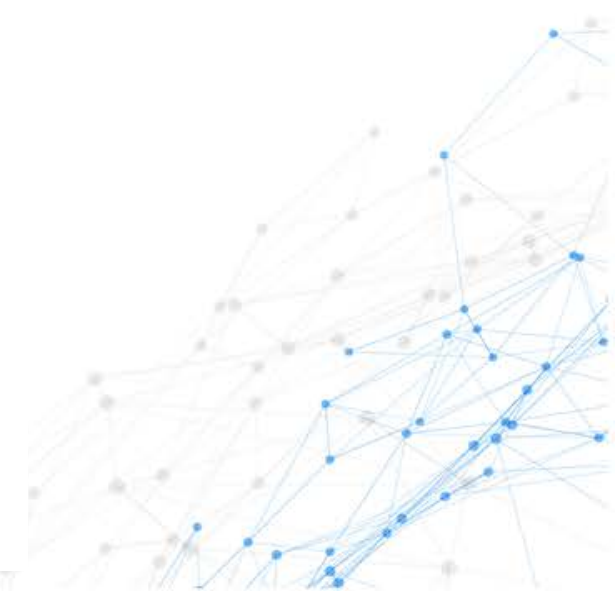**Test Network**

Sepolia...

Real blockchain

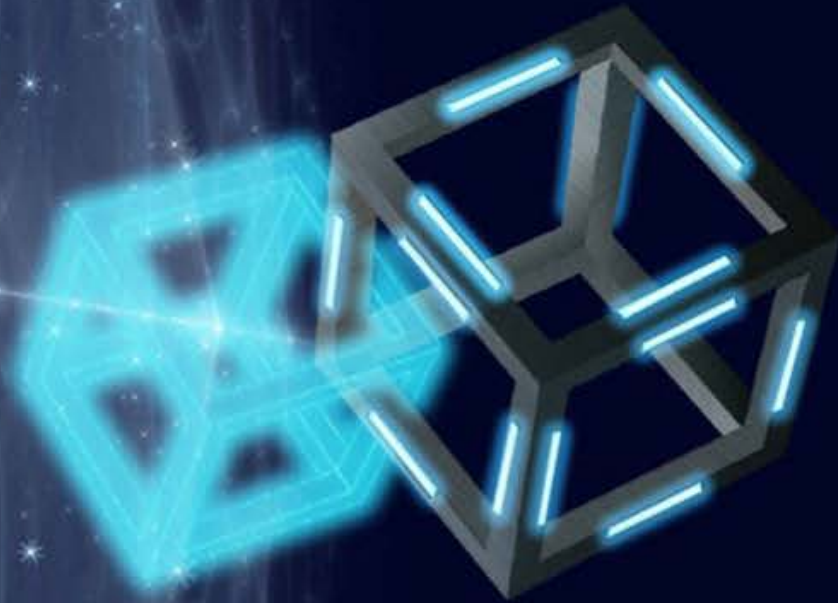Persistent (can be deleted)

Costs no money

Developer Network

Ganache, Hardhat

Test blockchain

Very fast

Non-persistent

Costs no money

Introduction To Ethereum Smart Contract Development

Ether, Wei & Gas

The Ethereum Network – ETH

ETH, wei & gas:

➢ Ether or ETH is the currency that is used on the Ethereum network

➢ 1 ETH can be divided into 10 ^18 wei (wei is the smallest unit)

➢ Executing a transaction (sending ETH, executing a state changing smart contract function) costs a fee. The unit of that fee is called gas

➢ The amount of gas to be paid depends on the complexity of the transaction

➢ The price per unit of gas depends on the state of the network

The Ethereum Network – Gas Price/Limit

Every operation (addition, subtraction...) consumes a certain amount of gas units and the price per unit of gas depends on the state of the network.

Gas limit: Maximum amount of gas you are willing to pay for your transaction. If your transaction takes less gas, then the extra gas that you have provided will be refunded back to your wallet. Otherwise, your transaction will fail.

Gas price (in Gwei): The gas price is the price per unit of gas you are willing to pay for executing your transaction. If you set a higher gas price, your transaction may be processed quicker.

EVM Opcodes: https://ethereum.org/en/developers/docs/evm/opcodes/

OPCODES

Stack	Name	Gas	Initial Stack
00	STOP	0	
01	ADD	3	a, b
02	MUL	5	a, b
03	SUB	3	a, b
04	DIV	5	a, b
05	SDIV	5	a, b

The Ethereum Network – Calculation of Txn Fees
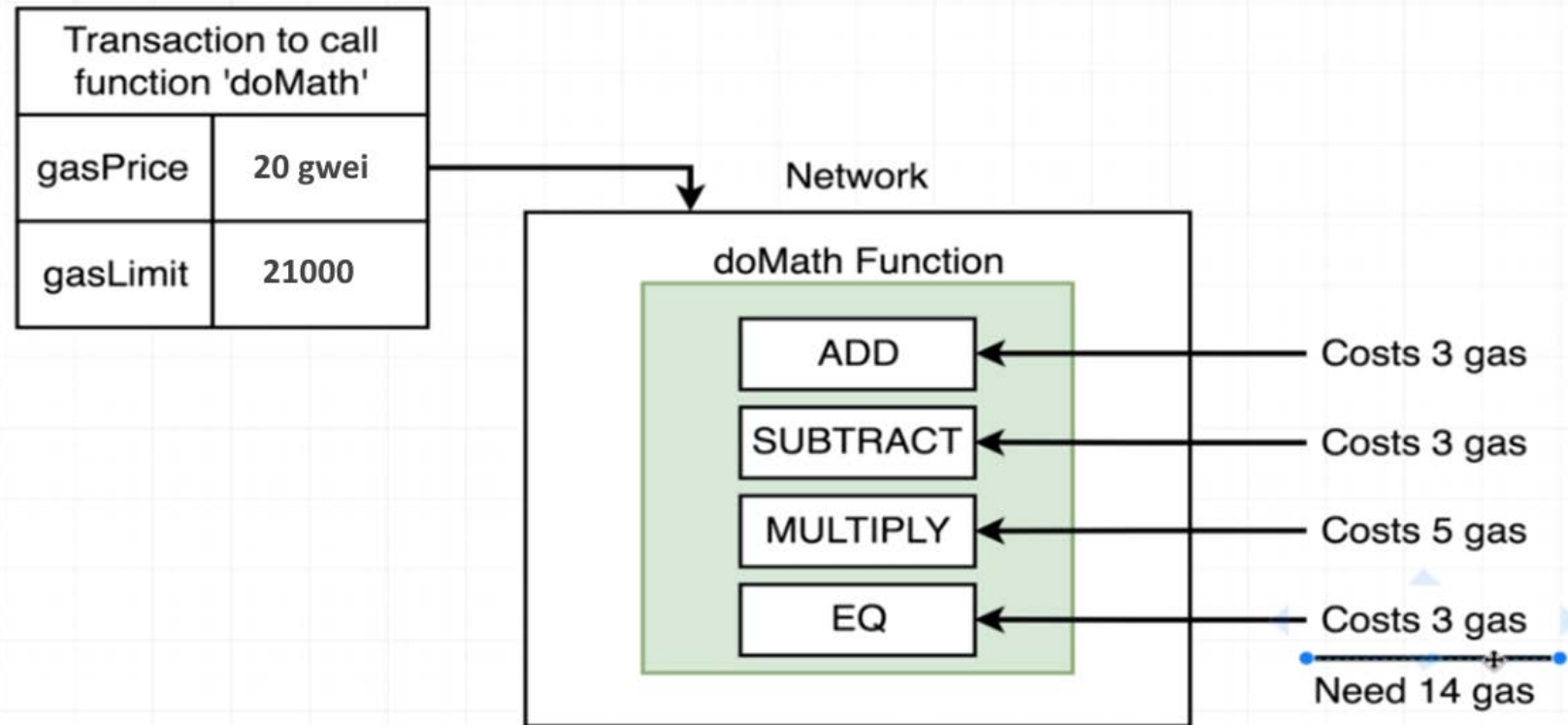
Total Cost: 20 gwei/gas x 14 gas = 28 gwei

The Ethereum Network – Example Txn Fees

⑦ From:	0xBaF6dC2E647aeb6F510f9e318856A1BCd66C5e19 (MEV Builder: 0xBaF...e19) 📋
⑦ To:	0x2e08451B79a01Cda253811E45719CeB42640c20d 📋

⑦ Value:	♦ 0.022236801683902839 ETH ($36.42)
⑦ Transaction Fee:	0.00040547801574 ETH ($0.66)
⑦ Gas Price:	19.30847694 Gwei (0.0000000193084 7694 ETH)

⑦ Gas Limit & Usage by Txn:	21,000 | 21,000 (100%)
⑦ Gas Fees:	Base: 19.30847694 Gwei | Max: 19.30847694 Gwei | Max Priority: 0 ETH
⑦ Burnt & Txn Savings Fees:	🔥 Burnt: 0.00040547801574 ETH ($0.66) 🐢 Txn Savings: 0 ETH ($0.00)

https://etherscan.io/tx/0xa85520109eb43c31a0ea3df7b453ae5b859d22e02605db8a082433cf7a394503
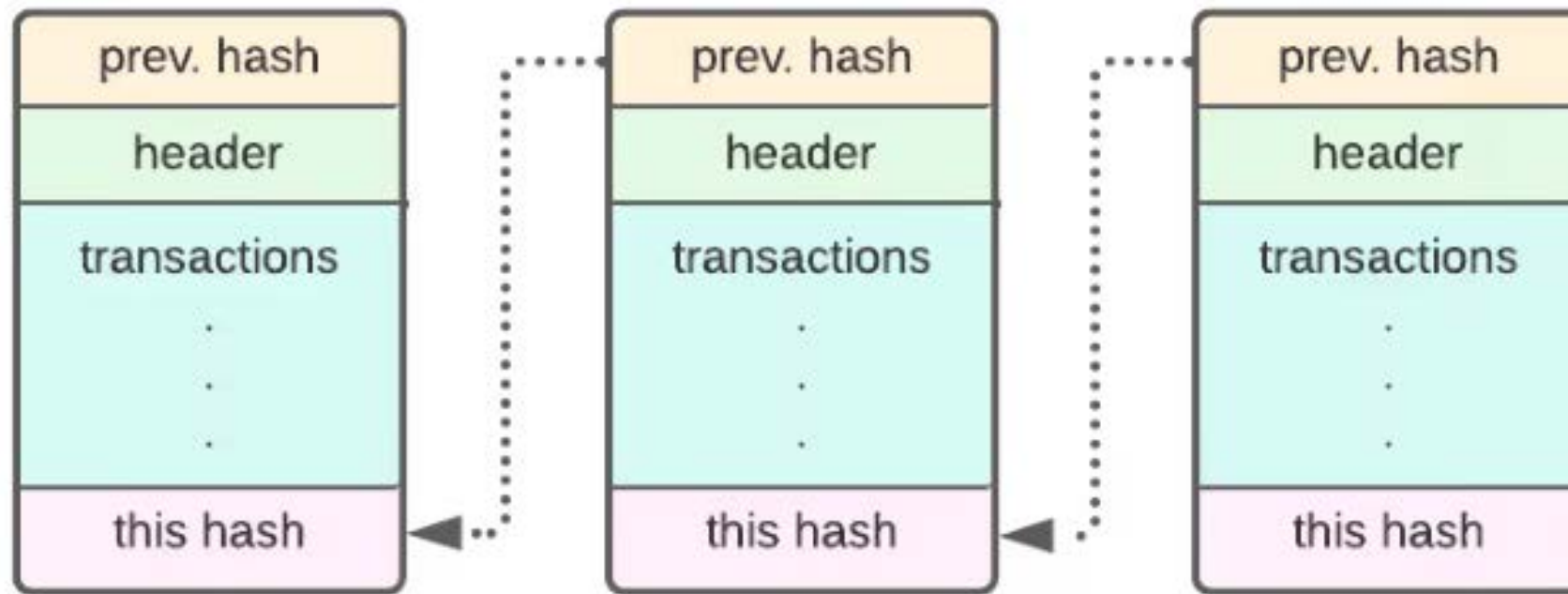
Blockchain Basics

➤ A chain of connected blocks (hash of the previous block)

➤ Every 10-20 seconds a new block is generated (PoW in past, now PoS)

➤ Block header: Timestamp, block number, hash of the previous block, total gas used...

➤ Each block has a limit of 30 million units of gas

Blockchain Basics - Hashing

➢ Digital fingerprint of a specific input (a phrase, PDF, MP3...)

➢ Generates output (digest) of a specific length

➢ The same input always provides the exact same output

➢ Two different input values will never generate the same hash

➢ A slight modification generates a completely different output

➢ A hashing function is a one-way function

➢ Quick to compute

➢ Solidity uses Keccak-256 => generates a 32 byte (64 character) output

➢ Keccak-256 online tool: https://emn178.github.io/online-tools/keccak_256.html

Example:

Hello => 06b3dfaec148fb1bb2b066f10ec285e7c9bf402ab32aa78a5d38e34566810cd2
Hello1 => f78fd070e76f73c4e7282620a7ab5ba58dc4f724d5dfd45d97ec4e01f817ce9d
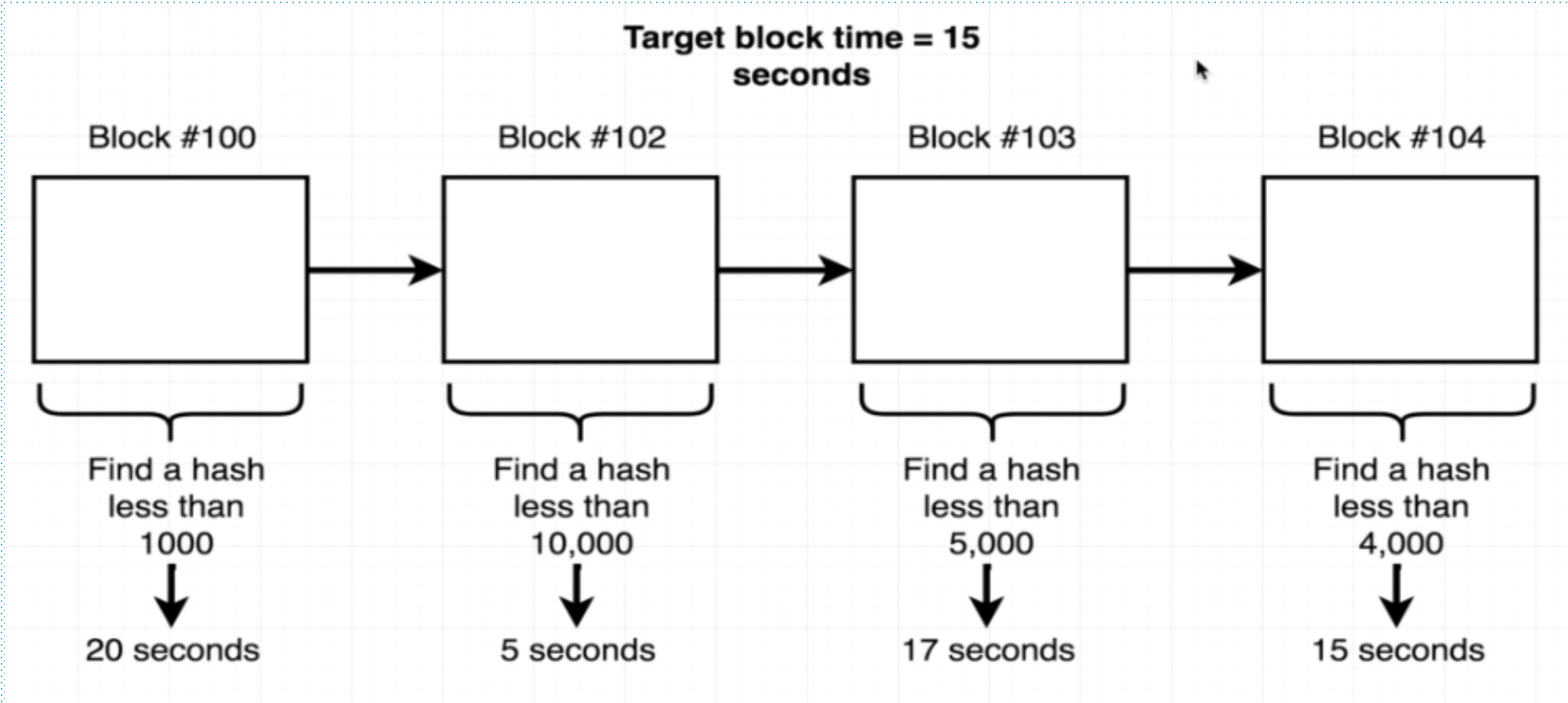
Blockchain Basics – Ethereum Consensus

The previous (PoW) Ethereum consensus:

Data	+	Nonce	=	Output Hash	Output hash as a base 10 number	Is this less than 1000?
'Hi There'		0		a23042b2e	178917215	no
'Hi There'		1		cbc1491	29589283	no
'Hi There'		2		0ca24258	94869869	no
'Hi There'		3		d9eed91	13938166	no
'Hi There'		4		1488baec	419386918	no
'Hi There'		5		0077bbb	100	yes

Blockchain Basics – Ethereum Difficulty

Adjusting the PoW difficulty level:

# Blockchain Basics – Blockchain Demo

Simplified blockchain model: https://andersbrownworth.com/blockchain/block

ETH 2.0 – Beacon Chain

➤ Launched in December 2020 => introduces PoS & prepares the blockchain for sharding

➤ Merge with original PoW chain in September 2022 => replacing PoW with PoS

➤ PoS validators have taken over from PoW miners => rewards for creating & validating blocks
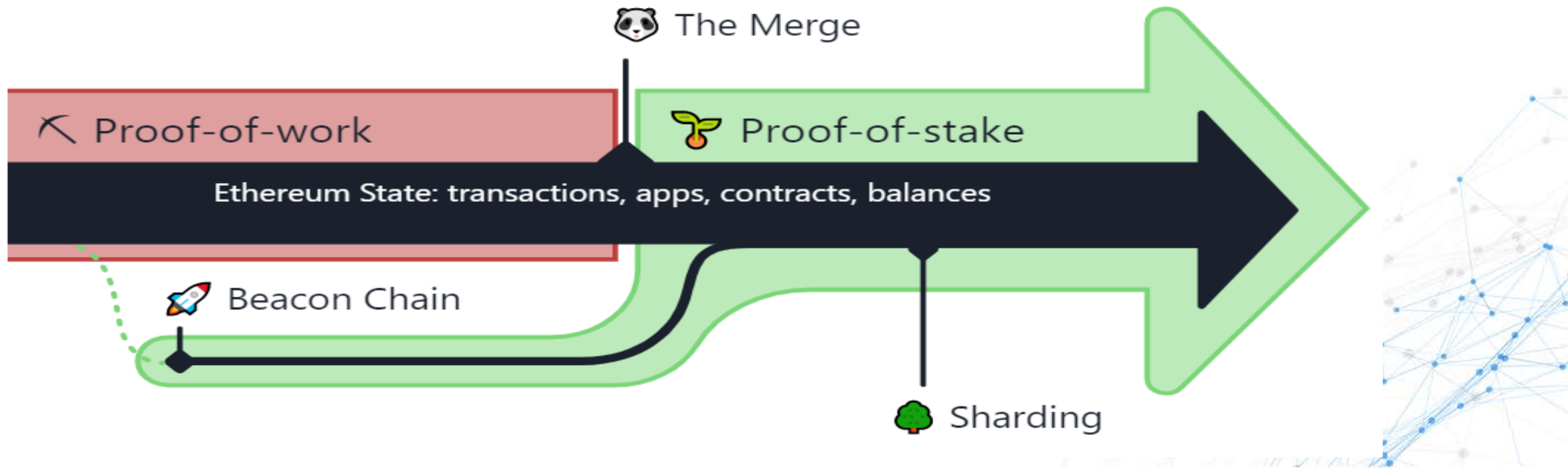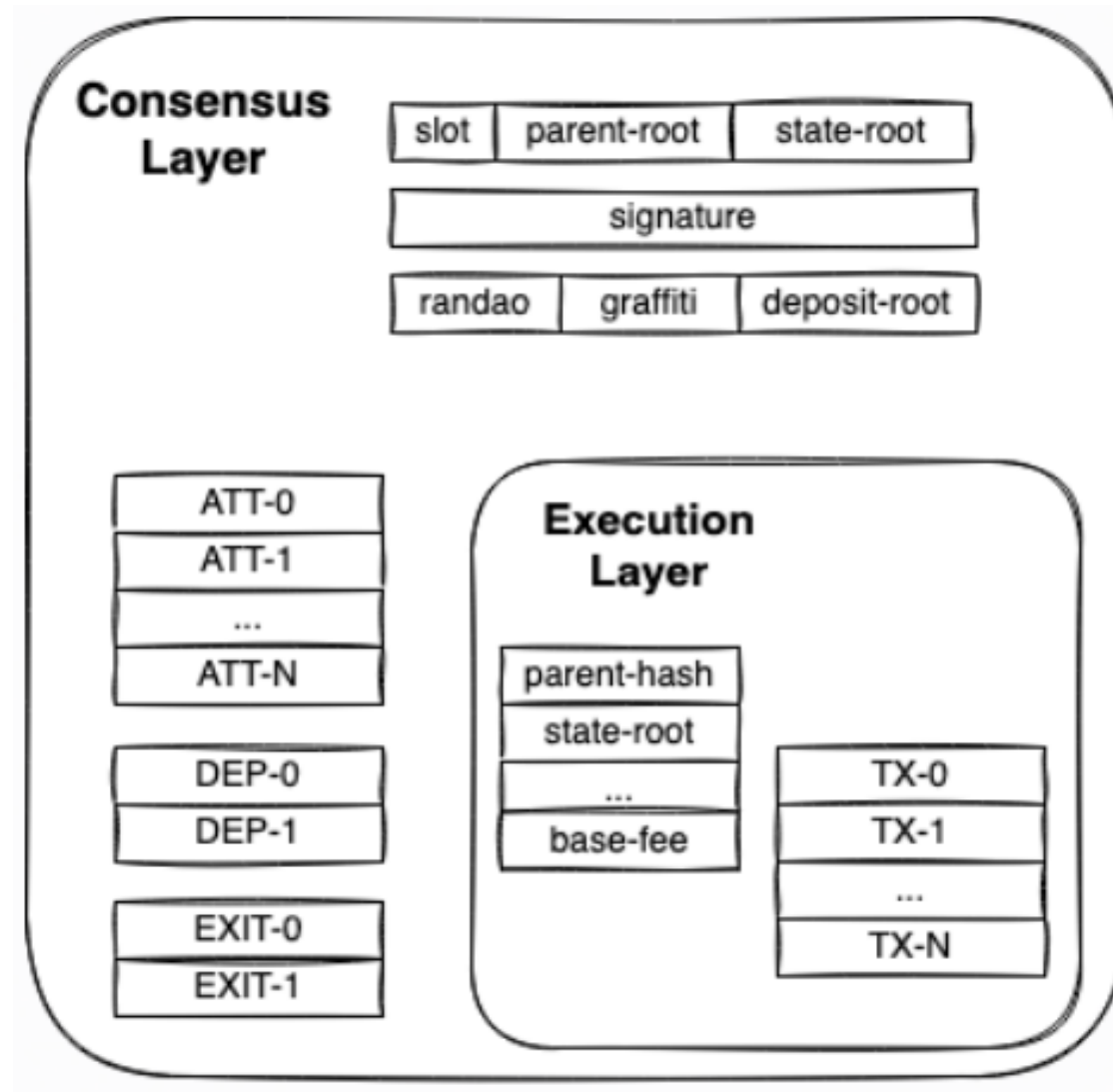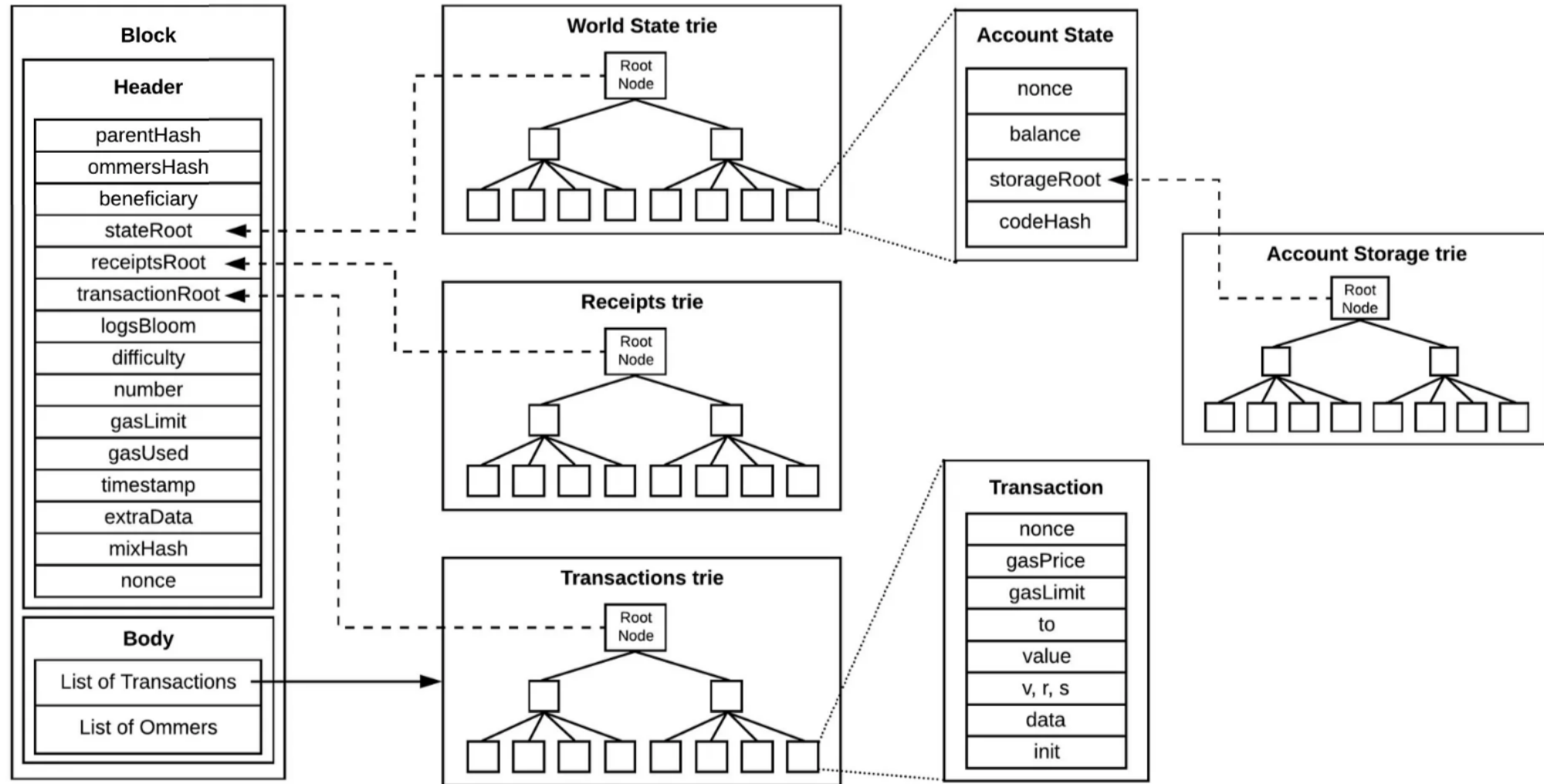
ETH 2.0 – PoS

➢ Participants in PoS consensus are called validators => need to deposit 32 ETH

➢ Time on the Beacon chain is divided into epochs and slots. Each slot is 12 seconds long (previously ~13 s). An epoch is a period of 32 slots (~6.4 minutes)

➢ Validators need to validate and generate new blocks => earn staking rewards

➢ Validators get penalized if their nodes go offline - must remain online at least 55% of the time to remain profitable

➢ Validators get slashed (3-100% of their stake) if they misbehave – voting for invalid blocks...

➢ Change in block headers: PoW related fields like difficulty, nonce… are no longer used => set to 0

ETH 2.0 – Post-Merge Chain
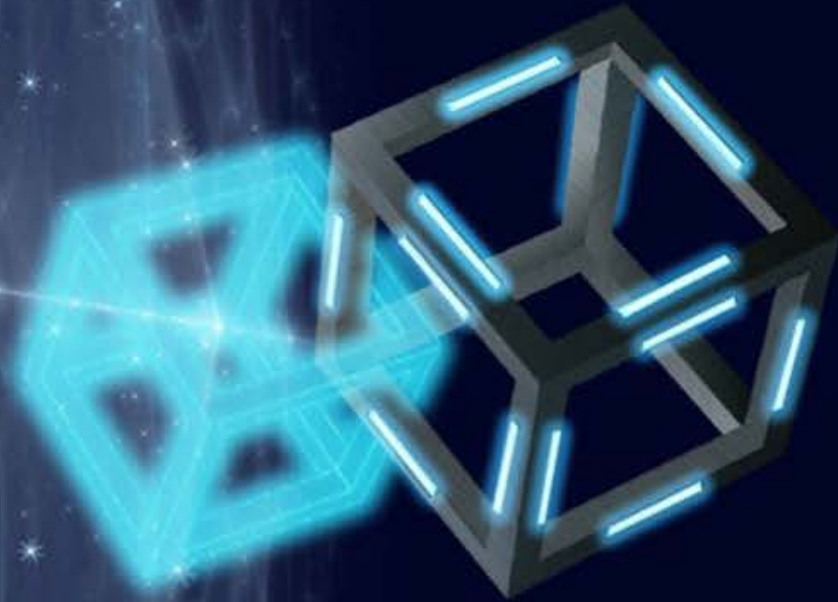
Ethereum Block Fields

A detailed description of all block fields: https://ethereum.org/en/developers/docs/blocks/

ETH 2.0 – Post-Merge Chain

➢ The current Eth1 and Beacon clients become the execution and consensus layers of Ethereum

➢ Node operators need to install both clients (consensus and execution)

➢ The consensus layer communicates with the execution engine and asks it to either produce or validate ExecutionPayloads (post-merge equivalent of Eth1 blocks)

➢ The execution engine handles block creation and validation

➢ Once produced / validated, the node shares them with other nodes on the p2p network

➢ The consensus layer shares attestations, slashings... and the execution layer shares transactions, sync state... each on their independent p2p network

Introduction To Ethereum Smart Contract Development

Ethereum Accounts

Blockchain Basics - Accounts

➢ 2 types of accounts: externally owned accounts (EOA) and contract accounts

➢ Both can hold balance in ETH

➢ Account addresses are 20 bytes (40 characters): 0x3fF34b4000308E581DC5b3CF009ad42b04479bAC

Externally Owned Accounts:

➢ Private key (held by wallet) => public key => account address

➢ Private key cannot be generated from account address

➢ Private key is required to sign transactions

➢ Anyone who has access to your private key can access your funds

➢ Private key can be generated from a 12 – 24 word key phrase (mnemonic) => store on paper

➢ Account address is public, can be shared with everyone, can be used on all Ethereum networks

Blockchain Basics – EOA's
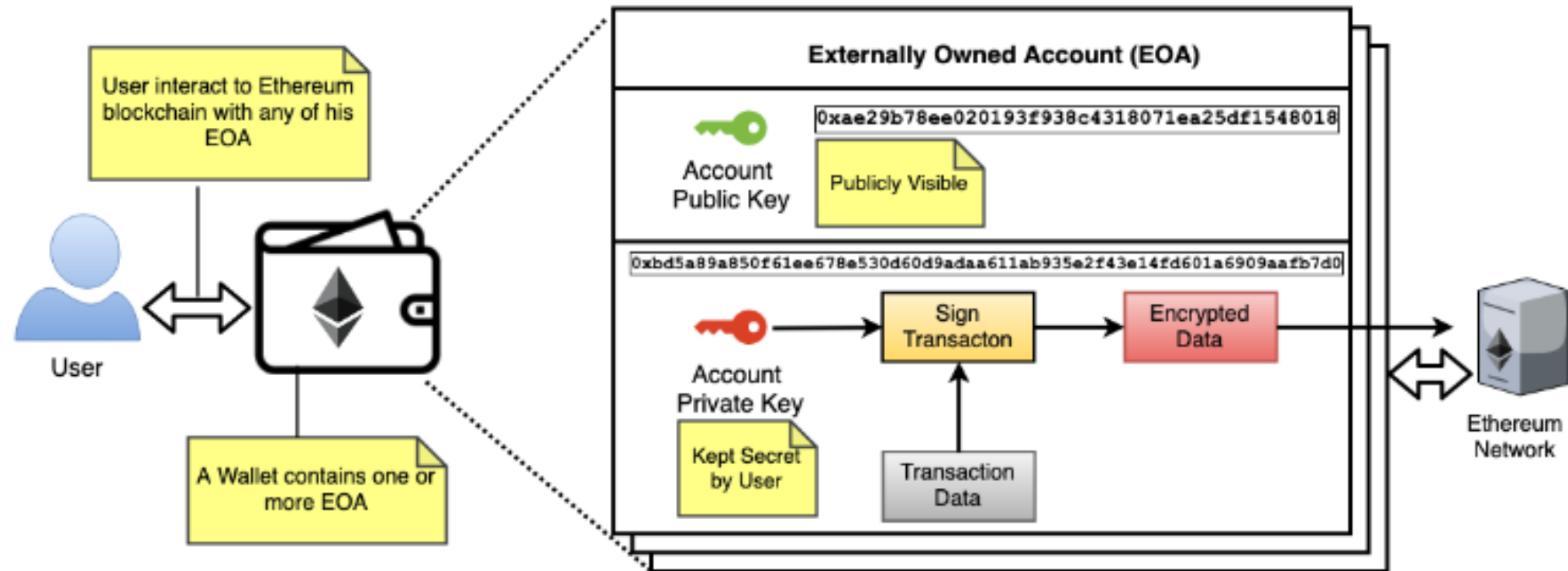
Blockchain Basics – Contract Accounts

Contract Accounts:

➢ When we deploy a smart contract, a similar 20 byte contract address is created

➢ A contract account is created by sending a transaction from an EOA without a value for the "to" field and the byte code in the "data" field

➢ The contract address is derived from the address of the creating account

➢ The byte code of the smart contract is stored in the block with the transaction

➢ Contract addresses have no private key

➢ A contract cannot initiate a transactions on its own => can only be done by an EOA

Blockchain Basics – Contract Accounts
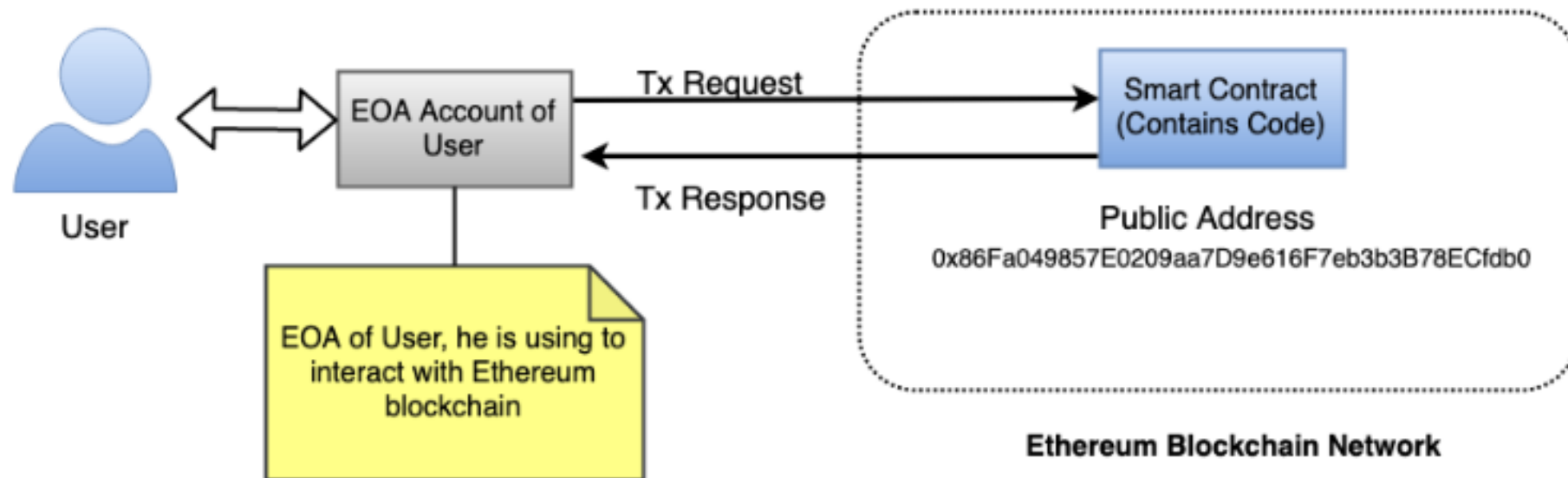
Introduction To Ethereum Smart Contract Development

Transactions & Signatures

Blockchain Basics - Transactions

➢ A transaction is required whenever we want to modify something on the blockchain => send ETH or change the state of a smart contract

➢ A transaction can only be initiated from an EOA

➢ Execution of a transaction: Provide data for various txn fields => sign the txn => send txn to Ethereum network => signature of the txn is validated => valid txn is included in a new block

Transaction Fields:

➢ *From:* Address of the EOA that initiated the transaction

➢ *To:* Address of another EOA (send ETH) or a contract (call a SC method or send ETH)

➢ *Value:* Amount of ETH to be sent to another EOA or a contract - can be empty

Blockchain Basics - Transactions

Transaction Fields:

➢ ***Gas limit:*** Maximum amount of gas you are willing to spend for the transaction

➢ ***Max fee and max priority fee:*** The fee you are willing to pay for the transaction (in gwei)

➢ ***Nonce:*** Counter that increases by 1, each time you initiate a transaction

➢ ***Data:*** Data (hex) you want to transfer with a transaction. Required to call a contract method => holds the hex encoded method and its arguments. Data field is empty on a simple ETH transfer

A transaction hash is generated for every valid transaction. This can be used to track the status of the transaction and to get additional details from https://etherscan.io

Blockchain Basics – Sample Transaction

Example

```
web3.eth.signTransaction({
    from: "0xEB014f8c8B418Db6b45774c326A0E64C78914dC0",
    gasPrice: "20000000000",
    gas: "21000",
    to: '0x3535353535353535353535353535353535353535',
    value: "1000000000000000000",
    data: ""
}).then(console.log);
> {
    raw: '0xf86c808504a817c800825208943535353535353535353535353535353535353535880de0b6b3a76400008025a04...
    tx: {
        nonce: '0x0',
        gasPrice: '0x4a817c800',
        gas: '0x5208',
        to: '0x3535353535353535353535353535353535353535',
        value: '0xde0b6b3a7640000',
        input: '0x',
        v: '0x25',
        r: '0x4f4c17305743700648bc4f6cd3038ec6f6af0df73e31757007b7f59df7bee88d',
        s: '0x7e1941b264348e80c78c4027afc65a87b0a5e43e86742b8ca0823584c6788fd0',
        hash: '0xda3be87732110de6c1354c83770aae630ede9ac308d9f7b399ecfba23d923384'
    }
}
```
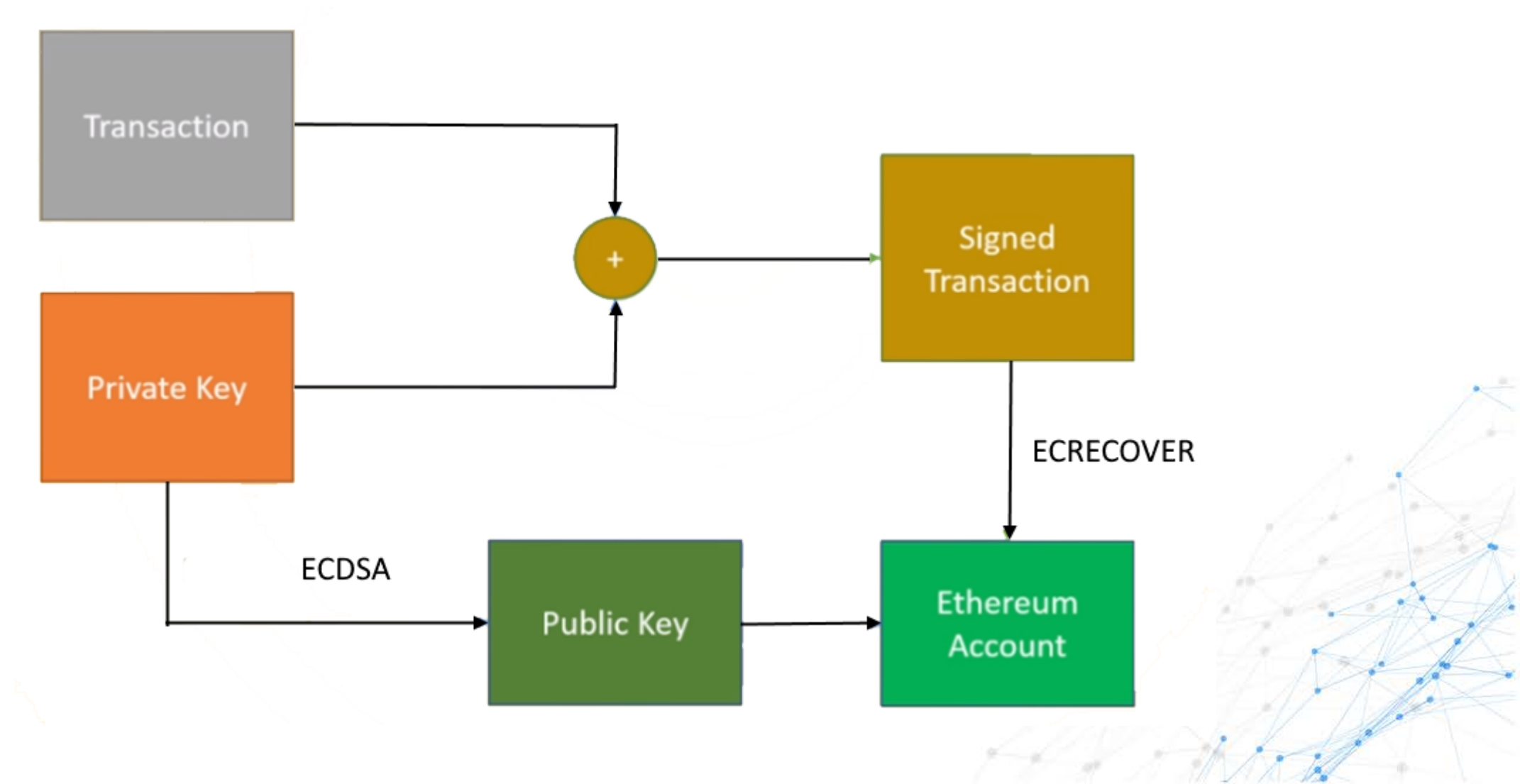
The v, r and s fields are cryptographic data that are generated from the senders' private key and they can be used to re-generate the address of the senders account and therefore verify the validity of the transaction.

https://web3js.readthedocs.io/en/v1.10.0/web3-eth.html

Blockchain Basics –Transaction Verification

Blockchain Basics – Etherscan Transaction

⑦ Transaction Hash:	0xa85520109eb43c31a0ea3df7b453ae5b859d22e02605db8a082433cf7a394503
⑦ Status:	✅ Success
⑦ Block:	⧗ 18062033 [2 Block Confirmations]
⑦ Timestamp:	🕐 21 secs ago (Sep-04-2023 08:50:47 AM +UTC)
⚡ Transaction Action:	▸ Transfer 0.022236801683902839 Ether To 0x2e0845...2640c20d
⑦ Sponsored:	
⑦ From:	0xBaF6dC2E647aeb6F510f9e318856A1BCd66C5e19 (MEV Builder: 0xBaF...e19)
⑦ To:	0x2e08451B79a01Cda253811E45719CeB42640c20d
⑦ Value:	♦ 0.022236801683902839 ETH ($36.42)
⑦ Transaction Fee:	0.00040547801574 ETH ($0.66)
⑦ Gas Price:	19.30847694 Gwei (0.0000000193847694 ETH)

https://etherscan.io/tx/0xa85520109eb43c31a0ea3df7b453ae5b859d22e02605db8a082433cf7a394503

Introduction To Ethereum Smart Contract Development

Smart Contracts / Solidity

Blockchain Basics – Smart Contracts/Solidity

➢ Software program running on the blockchain

➢ Can be viewed as a state machine

➢ A signed transaction needs to be transmitted to a node and validated by the network
   to change the state of a smart contract

➢ Most smart contracts are written in Solidity

➢ Solidity is a strongly typed, Turing complete language that has some similarities with JavaScript

➢ Solidity compiler generates the smart contract byte code and the Application Binary Interface (ABI)

➢ Byte code is deployed to the blockchain and stored in the contract account (visible on Etherscan)

➢ ABI contains the specification of all public contract functions including the types of
   the function arguments and the return values

➢ The ABI is required to access contract functions from a client application

Blockchain Basics – Smart Contracts/Solidity

| Transfers | Holders | Info | DEX Trades | Contract ✓ | Analytics | Comments |

Code Read Contract Write Contract

✅ **Contract Source Code Verified** (Exact Match)

| Contract Name: | **TetherToken** | Optimization Enabled: |

| Compiler Version | **v0.4.18+commit.9cf6e910** | Other Settings: |

📄 **Contract Source Code** (Solidity) **Audit Report**

```
76 ▾ /**
77    * @title ERC20Basic
78    * @dev Simpler version of ERC20 interface
79    * @dev see https://github.com/ethereum/EIPs/issues/20
80    */
81 ▾ contract ERC20Basic {
82       uint public _totalSupply;
83       function totalSupply() public constant returns (uint);
84       function balanceOf(address who) public constant returns (uint);
85       function transfer(address to, uint value) public;
86       event Transfer(address indexed from, address indexed to, uint value);
87    }
88
```

https://etherscan.io/token/0xdac17f958d2ee523a2206206994597c13d831ec7#code

Blockchain Basics – SC Functions

Functions that return information:

➢ No transaction is required, a simple message call is sufficient

➢ No data on the blockchain is modified

➢ The function returns some data

➢ The function runs instantly

➢ Running the function does not cost anything

Function that modify state:

➢ A transaction is required to call the function

➢ Data on the blockchain is modified

➢ No data is returned - just a transaction hash

➢ The transaction first needs to be added to a new block before the update becomes visible on the blockchain

➢ A transaction fee needs to be paid

Setting up Metamask

Setting up Metamask

➢ Install Metamask plugin: https://metamask.io/download/

➢ Create a wallet and make a note of the secret key phrase

➢ Change the name of the default account

➢ Add an additional account

➢ Advanced settings: Switch on advanced gas control, show test networks and customize nonce

➢ Add a local network for Hardhat or Ganache

➢ Additional network settings: https://chainlist.org/

➢ Export wallet private key

➢ Add additional ERC20 tokens to the wallet: https://etherscan.io/tokens

➢ Get Sepolia ETH from a faucet: https://faucetlink.to/sepolia ; https://sepolia-faucet.pk910.de/

Setting up Metamask

Add a new network:
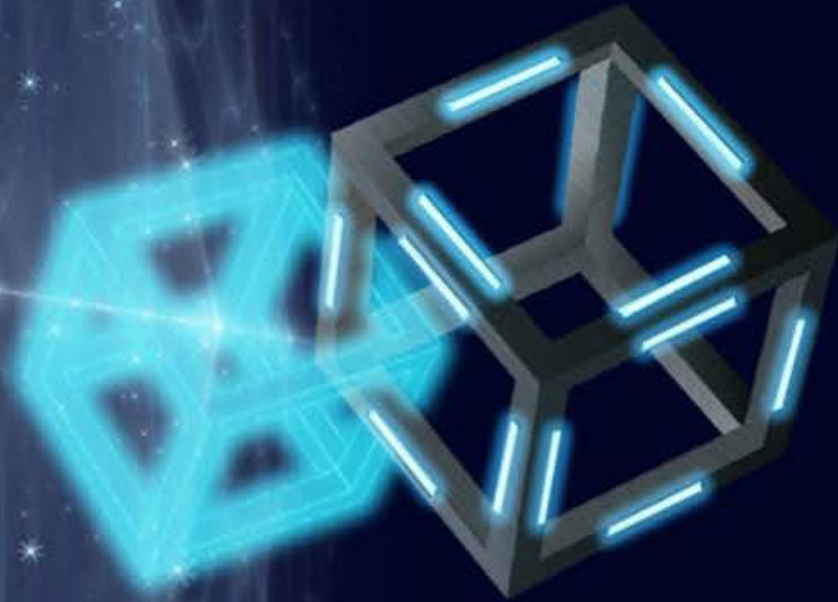

My Accounts => Settings => Networks

Export the account private key:
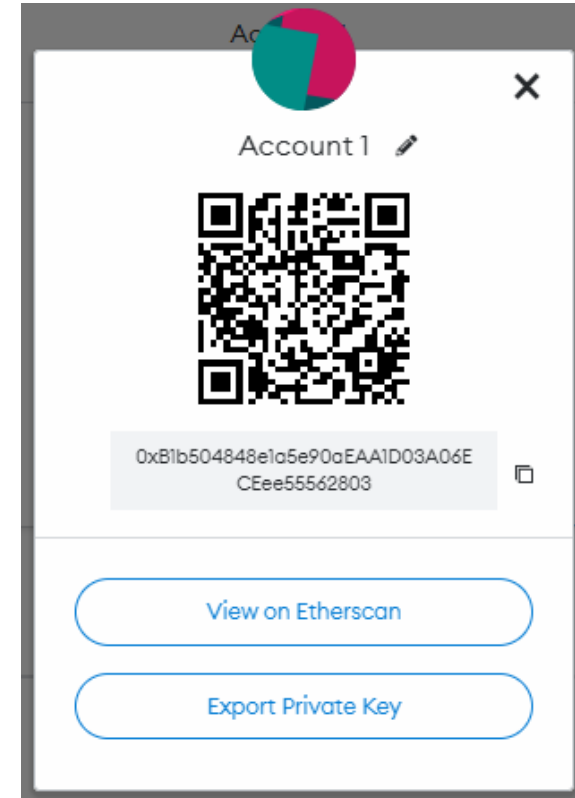
Account Options => Account Details

Writing, Compiling, Deploying and Debugging Smart Contracts on Remix

Remix

Remix IDE: https://remix.ethereum.org/

Main Pages:

➤ File Explorer: Several example projects, import/export workspace,
    ABI & bytecode for smart contracts in artifacts folder

➤ Compiler: Select the desired compiler version and check "Auto compile",
    copy ABI and bytecode

➤ Deploy & Run Transactions - various environments: Remix VM, Injected Provider, Dev (Hardhat,
    Ganache)

Hello World Smart Contracts on Remix

Hello World Smart Contract on Remix

➢ Create a basic smart contract that can read/write a string message and emit an event whenever the message is updated

➢ Deploy the contract using the **Remix VM**

➢ Test the contract functions

➢ Use the debugger: breakpoints, state variables, bytecode execution,
step details: gas cost, block number...

➢ Deploy the contract to the Sepolia test network using "*injected provider – Metamask*"

➢ Check deployed bytecode in Metamask

➢ Check deployment transaction on Etherscan

➢ Execute the "updateMessge" function on Remix and check the transaction on Etherscan

Remix – Deployment & Execution

Target network, deployment account, contract account, owner rights...:
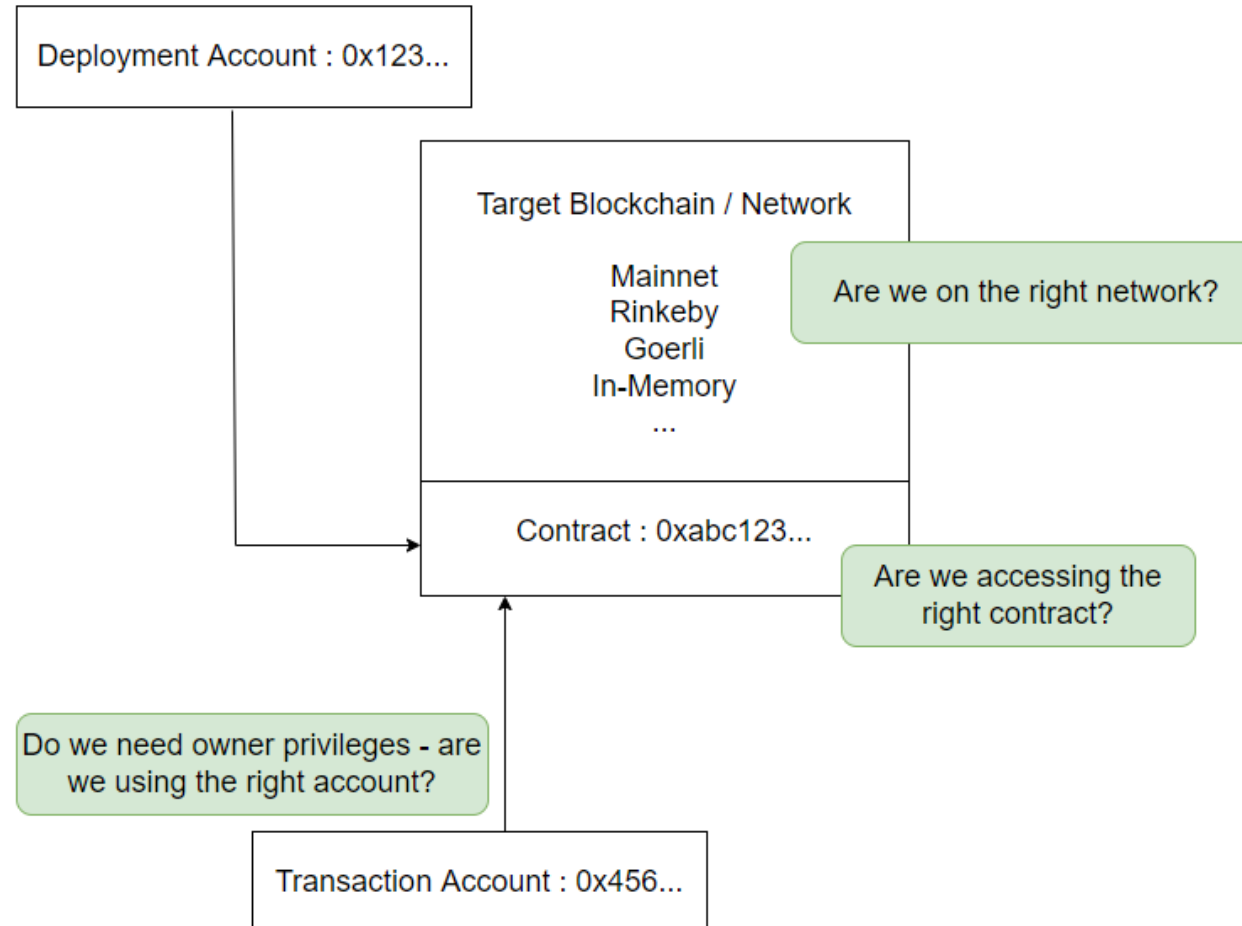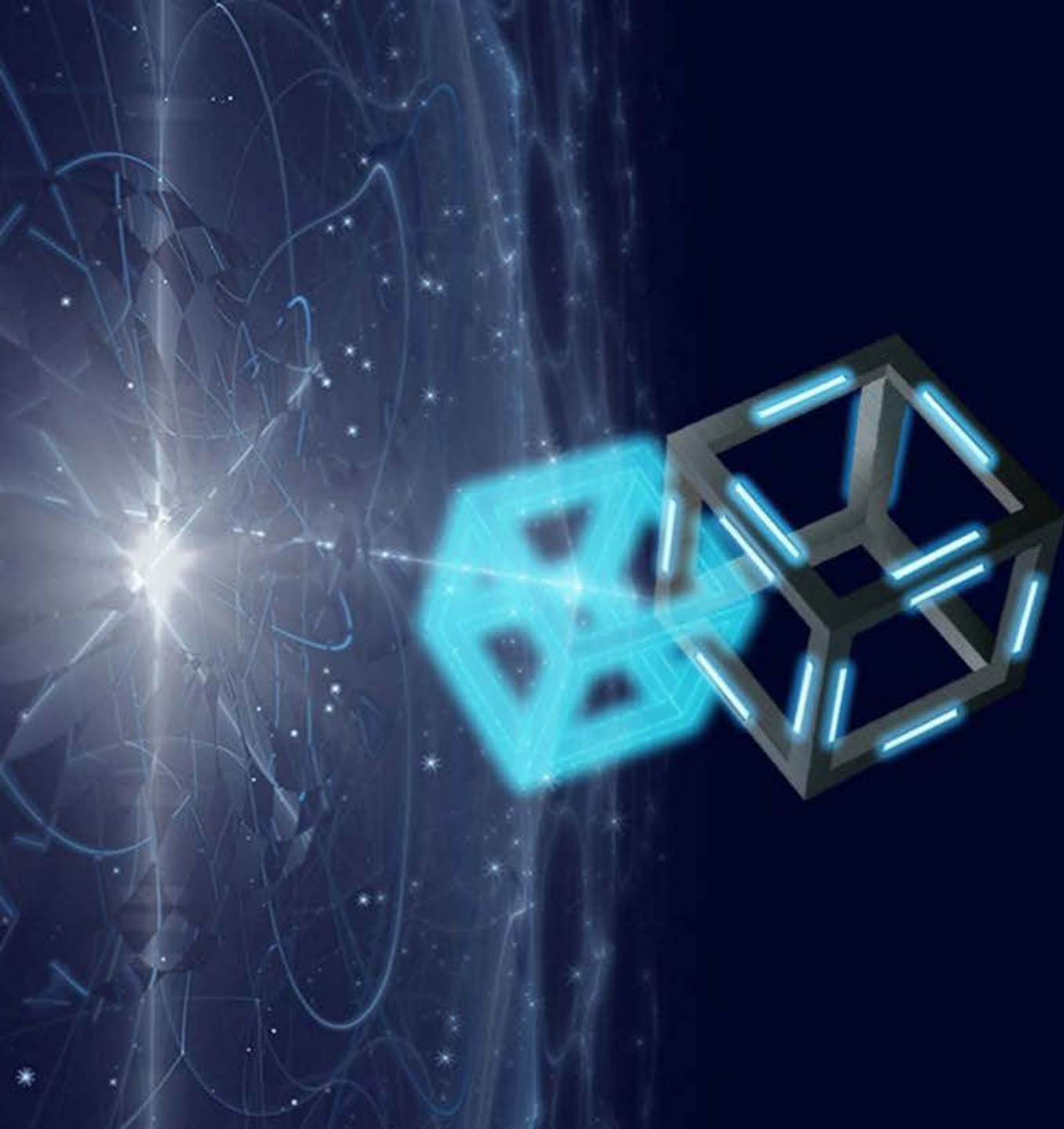
Remix - Exercise

Tasks:

➢ Add additional functions to retrieve and update and integer value (type: uint)

➢ Deploy the new contract to the Remix VM

➢ Test the new functions

➢ Use the debugger to step through the code – check variables, executed opcodes, step details...

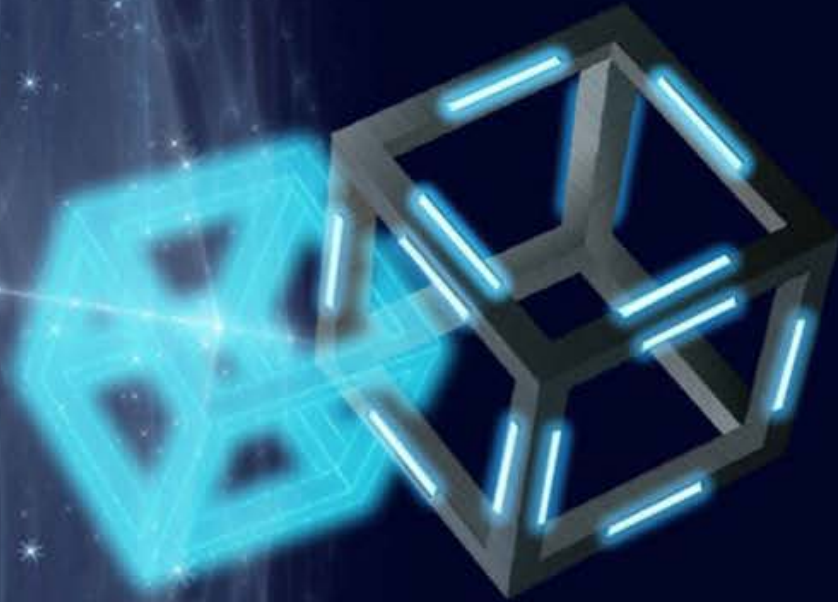# Setting Up a Local Development Environment

Local Development Environment

Install the following Applications and plugins:

➢ Node.js (includes NPM) - LTS version: https://nodejs.org/en/

➢ Visual Studio Code: https://code.visualstudio.com/download

➢ VS Code Plugins:

  ➢ Solidity + Hardhat

  ➢ Live Server

  ➢ Prettier

Compiling, Testing and Deploying Smart Contracts with Hardhat

What is Hardhat?

➢ Development environment to compile, deploy, test, and debug Ethereum smart contracts

➢ Hardhat comes built-in with the Hardhat Network

➢ Hardhat Runner - the CLI to interact with Hardhat. For example: npx hardhat compile

➢ To see all available tasks and options, run: npx hardhat (compile, node, run, test... --network)

Creating a Hardhat project:

➢ Create a package.json file in the project folder: *npm init*

➢ Install Hardhat locally: *npm i hardhat -D*

➢ Create a hardhat project: *npx hardhat init* => select: "Create a JavaScript project"

➢ Install the following plugin: npm i dotenv --- *[npm i -D @nomicfoundation/hardhat-toolbox]*

Configuring Hardhat

➢ Sample configuration file: https://hardhat.org/config

➢ Configuration file in the root of the project folder (hardhat.congig.js) is executed before each task

➢ Paths for sources, artifacts, cache and tests can be configured

➢ Additional options for network configuration: from, gas, gasPrice, chainId, timeout

```javascript
module.exports = {
    solidity: {
        version: "0.8.9",
        settings: {
            optimizer: {
                enabled: true,
                runs: 200,
            },
        },
    },
    defaultNetwork: "localhost",
    networks: {
        rinkeby: {
            url: REACT_APP_ALCHEMY_API_URL_RINKEBY,
            accounts: [
                `0x${REACT_APP_PRIVATE_KEY}`,
            ],
        },
    },
};
```

Alchemy & Etherscan API Keys

Alchemy:

➢ Create a free Alchemy account at: https://auth.alchemy.com/signup

➢ Create an App on Alchemy and select **Ethereum** for the chain and **Sepolia** for the network

➢ On the App, click on "**API Key**" and copy/paste the value under "**HTTPS**" into the .env file

Etherscan:

➢ Create a free Etherscan account at: https://etherscan.io/register

➢ On the left menu bar, click on "**API keys**" and create a new API key

➢ Copy/paste the API key into the .env file

Compiling & Deploying SC's with Hardhat

Compiling all contracts:

npx hardhat compile

Bytecode and ABI is generated
in the artifacts folder

Deploying a contract:

npx hardhat run scripts/deploy.js

Or:

npx hardhat run scripts/deploy.js -- network sepolia

```javascript
async function main() {
    const helloWorld = await ethers.deployContract("HelloWorld",
        ["Initial message"]);

    await helloWorld.waitForDeployment()

    console.log("HelloWorld deployed to:", helloWorld.target)
}

main().catch((error) => {
    console.error(error)
    process.exitCode = 1;
})
```

The Hardhat Network

Running an in-memory instance:

➢ *npx hardhat run scripts/deploy.js* **--network hardhat**

Running a standalone instance (can be accessed by external clients – Metamask):

➢ Open a command window: *npx hardhat node*

➢ This starts a local node at: http://localhost:8545 with chainId: 31337

➢ Second command window: *npx hardhat run scripts/deploy.js* **--network localhost**

➢ Configure the local network in Metamask

Debugging with Hardhat

Debugging => print logging messages to the console!

```solidity
pragma solidity ^0.6.0;

import "hardhat/console.sol";

contract Token {
  //...
}
```

```solidity
function transfer(address to, uint256 amount) external {
    console.log("Sender balance is %s tokens", balances[msg.sender]);
    console.log("Trying to send %s tokens to %s", amount, to);

    require(balances[msg.sender] >= amount, "Not enough tokens");

    balances[msg.sender] -= amount;
    balances[to] += amount;
}
```
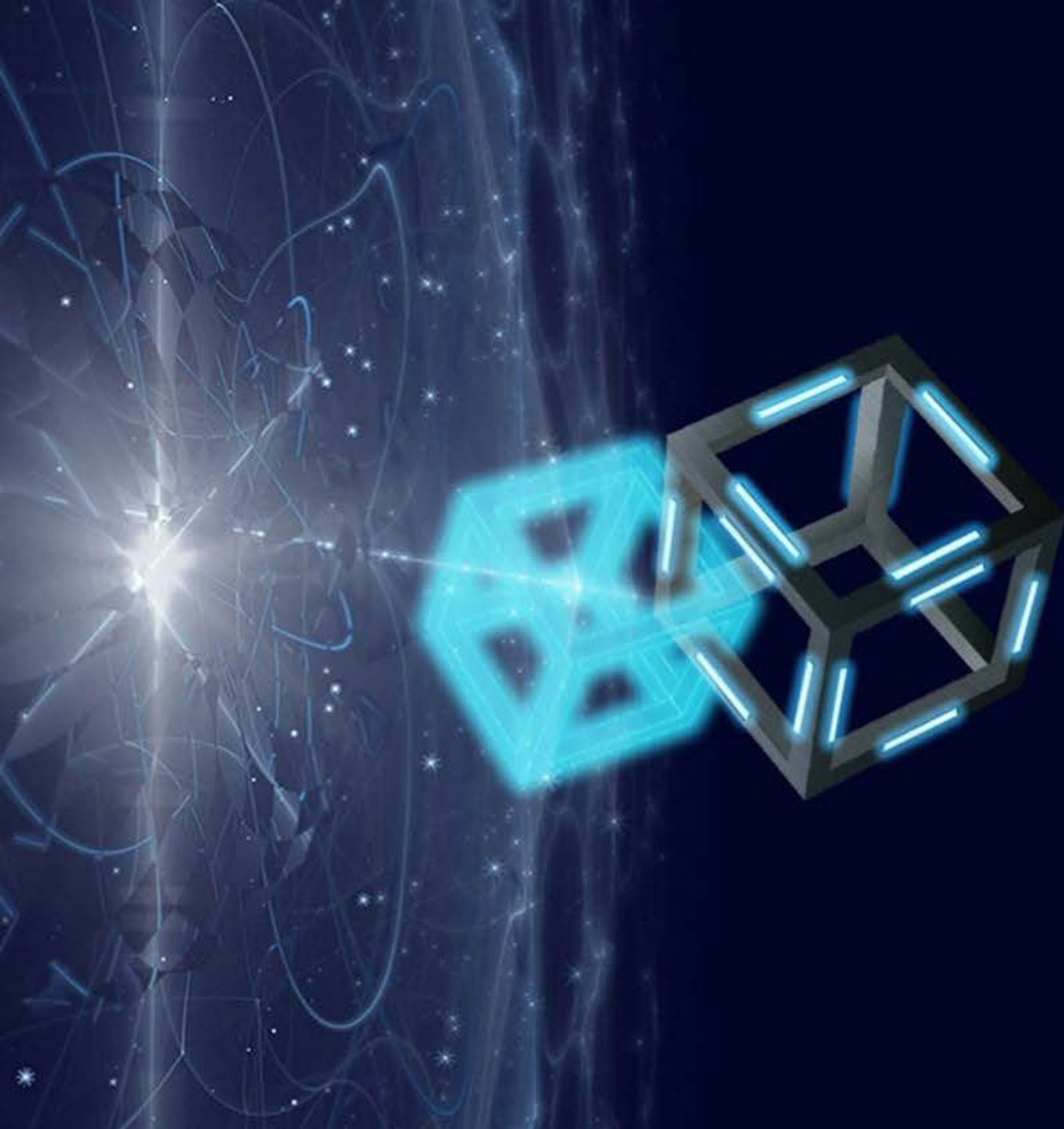
Obtaining Test Ether

➤ Sepolia is a test network and the fees need to be paid in Sepolia Ether

➤ Can be obtained for free on so-called faucets

Sepolia faucets:

➤ https://sepolia-faucet.pk910.de/

➤ https://sepoliafaucet.com/

➤ https://faucetlink.to/sepolia

Hello World Project on Hardhat

Project Requirements

- ➢ Create a basic Hardhat project with all required npm packages

- ➢ Add the Sepolia testnet to **hardhat.config.js**

- ➢ Add the Hello World smart contract we previously created in Remix

- ➢ Deploy the smart contract to a local node and to Sepolia

- ➢ Write a simple script that interacts with the smart contract (read/write message)

- ➢ Add the smart contract code to **sepolia.etherscan.io**
  npx hardhat verify –network sepolia CONTR_ADDR "constructor argument"

Exercise

➢ Update HelloWorld.sol: Add code to read/write a number

➢ Emit an event whenever the number changes

➢ Deploy the smart contract to a local node

➢ Update scripts/interact.js: Call the read/write number functions

➢ Deploy the smart contract to the Sepolia testnet

➢ Run scripts/interact.js on the smart contract deployed on Sepolia

➢ Check the transactions on https://sepolia.etherscan.io

➢ Add the new smart contract to https://sepolia.etherscan.io