Blockchain for Developers:

Welcome!

Course Leads:
Sara Magaziotis-Ginori
Olivia Li
Isaac Oh

Aman Shah
Elson Liu

Fujia Wang

BLOCKCHAIN
AT BERKELEY

Sara Magaziotis-Ginori

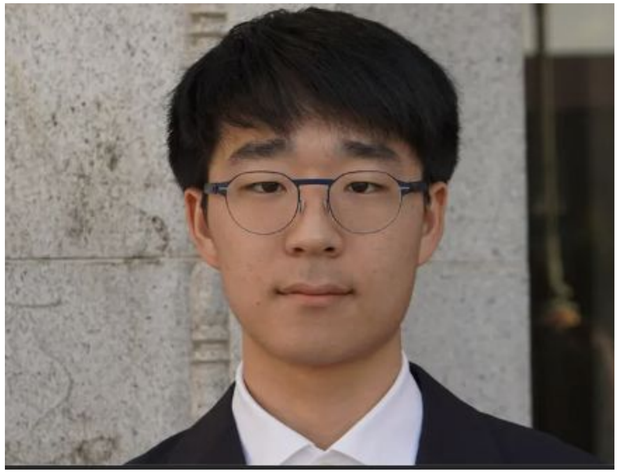
Consulting

smagazi@berkeley.edu

Olivia Li

Consulting

oliviali2028@berkeley.edu



Isaac Oh

Head of Research

isaac_oh@berkeley.edu

Aman Shah

Head of Consulting

amanshah@berkeley.edu

Elson Liu

Head of Education

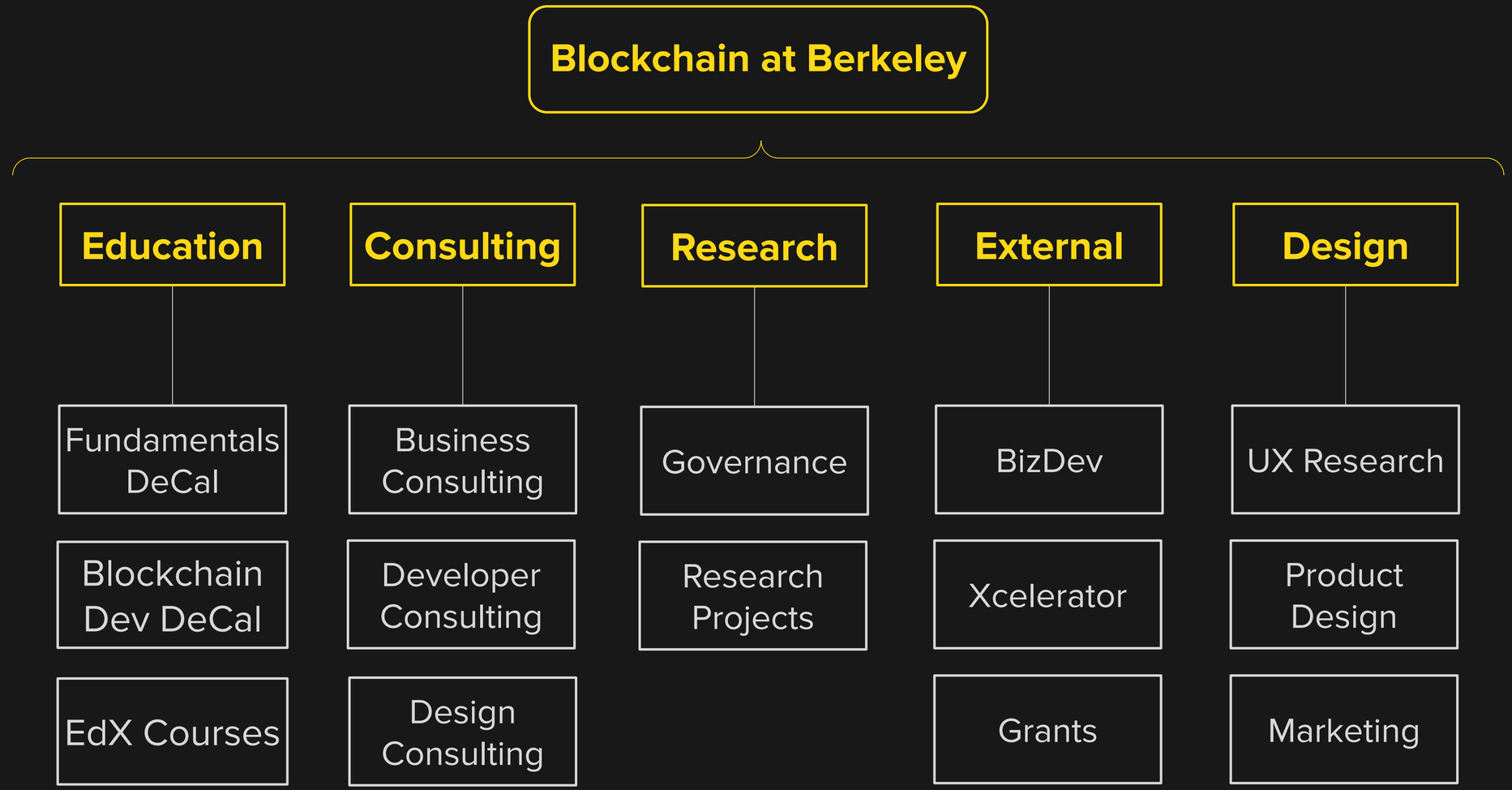
elsonliu28@berkeley.edu

Fujia Wang

Consulting

fujiawang@berkeley.edu

WHO ARE WE



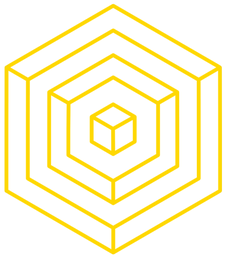
BLOCKCHAIN AT BERKELEY

Course Structure

- Lecture
  - Every **Wednesday** at **5:00 - 6:30 pm PST**
  - In **SOCS 20**
  - Class will usually end early.
    - We'll hold Office Hours with remaining Time
- Homework
  - Weekly homework assignments to practice material learned
  - **For help, email: dev-decal@blockchain.berkeley.edu**
- 3 Units
  - You can audit the course
- Slides, Syllabus, Homework, Projects
  - Uploaded on GitHub: https://github.com/BerkeleyBlockchain/fa25-dev-decal

Weekly Schedule for now...

Week #	Date	Lecture Title / Content
Week 1	9/10	❑ Introduction to Blockchain Development: A High-Level Overview
Week 2	9/17	❑ Solidity and Developer Tools
Week 3	9/24	❑ Advanced Solidity
Week 4	10/1	❑ Ethers.js, Wagmi, and Connecting to the Web
Week 5	10/8	❑ Writing Secure and Efficient Solidity
Week 6	10/15	❑ dApp Integrations: Compiling Contracts, Upgradability, Oracles
Week 7	10/22	❑ Memory Management in Blockchain Development
Week 8	10/29	❑ Rust Basics
Week 9	11/5	❑ Rust 2
Week 10	11/12	❑ Rust 3
Week 11	11/19	❑ Rust 4
Week 12	12/3	❑ Finale / Guest Lecture



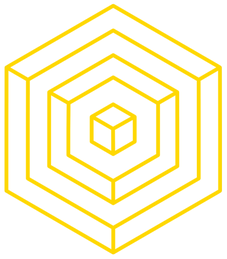
LECTURE OVERVIEW

1. WHY BLOCKCHAIN DEVELOPMENT
2. COURSE DETAILS
3. BLOCKCHAIN & ETHEREUM OVERVIEW
4. HW1 - BUILD A BLOCKCHAIN IN PYTHON
5. THANKS!

1 WHY DEVELOP BLOCKCHAIN

Origins of Blockchain

- **problem with traditional financial systems**
  - centralized insitutions to manage transactions (banks, stock exchanges etc)
  - inefficient, fees, security problems
  - 2008 global financial crisis, currency devaluation Venezuela 2018
- Blockchain a distributed ledger
  - single source of truth, we can see all transaction histories, we all agree
  - **Satoshi Nakamoto 2008 whitepaper**
    - Bitcoin is “peer-to-peer version of electronic cash”
    - don’t have to go thru financial institutions
    - Bitcoin (BTC) is “digital gold” bc limited number of bitcoins



VISIONS

WHY DEVELOP?

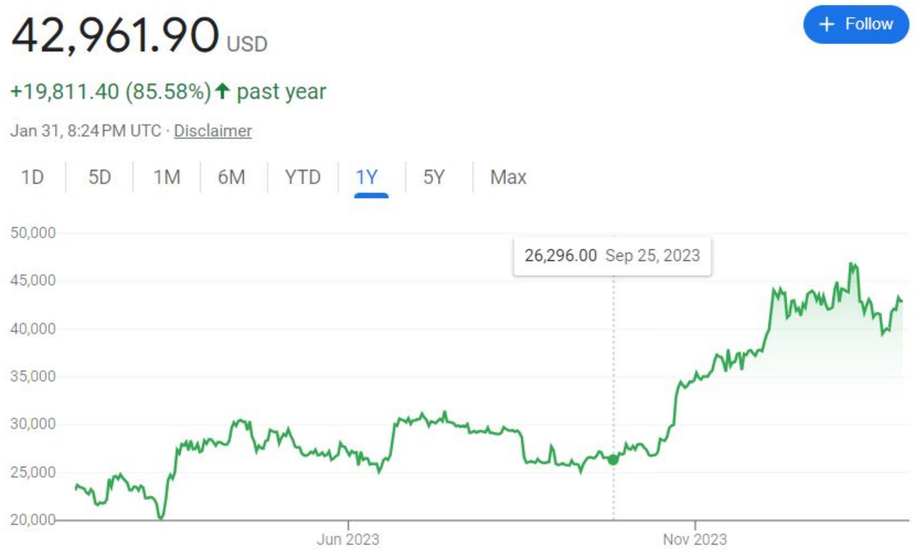
- **Building a new, significant, infrastructure-level technology**
  - Deprecating the outdated, centralized financial architecture of today
  - Parallels the 90’s Internet boom (have you read zero to one?)
- **Working in a field where impact can be made quickly**
  - Blockchain is a young field with several emerging companies
- **Intellectually interesting problems**
  - Getting to explore problems in fields ranging from cryptography (CS70) to algorithms to game theory to economics to trading.

DEMAND FOR BLOCKCHAIN

Government support:

- President Trump wants to make America a leader in digital assets.
- Est strategic Bitcoin Reserve and U.S Digital Asset Stockpile

2023



9/10/2025

AUTHOR: Krishna Mandal

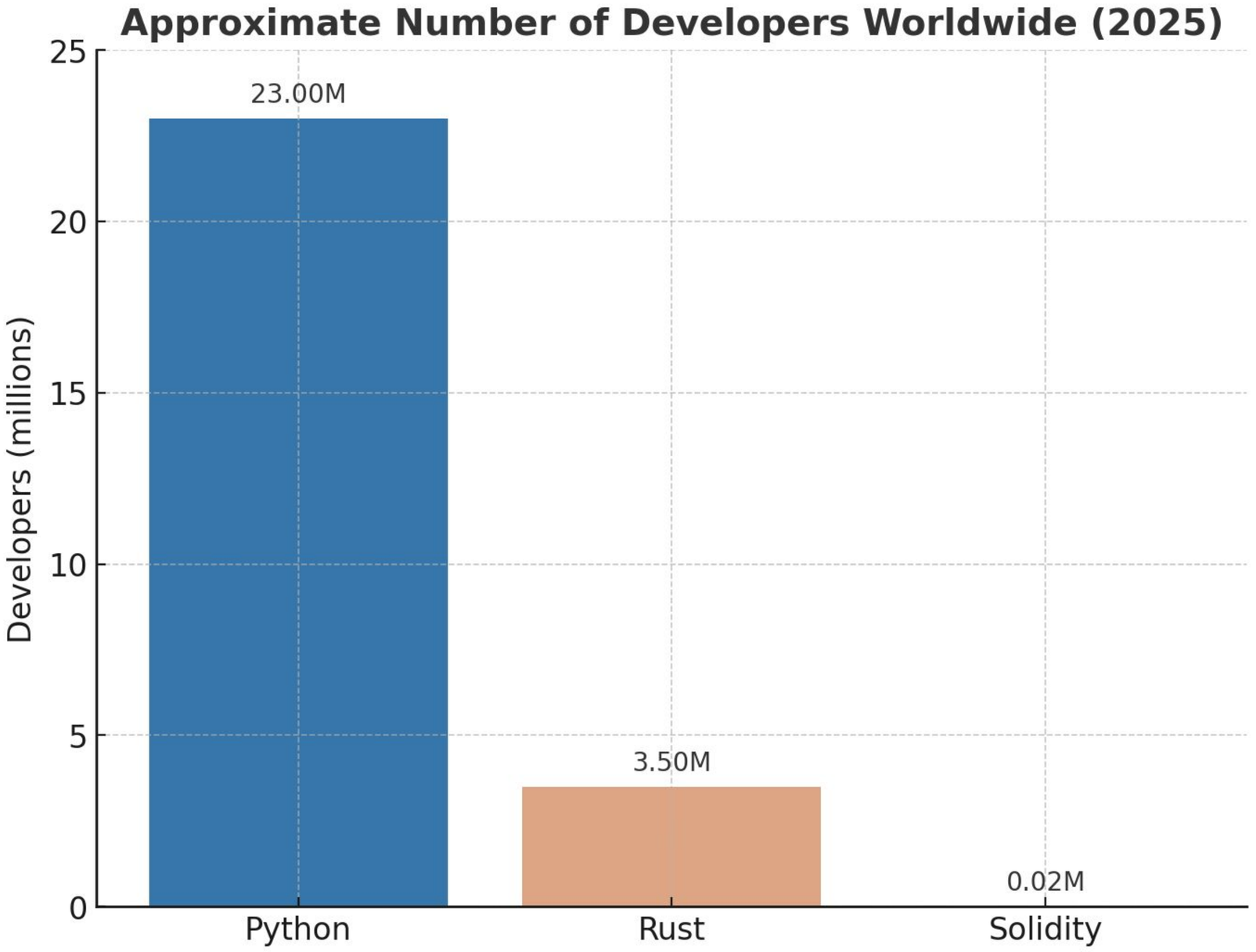
BLOCKCHAIN AT BERKELEY

OTHER REASONS

WHY BLOCKCHAIN

- **Open Community**
  - Accessible to anyone, most code is open source
- **Niche, but also not ridiculously competitive**
  - There’s not a lot of people who know about this stuff, and not everyone is eyeing it right now.
- **Higher demand, lower supply of developers**
  - That’s why Blockchain at Berkeley is successful
- **Easy to get internships/jobs and make big bank if you know stuff**
  - Can be a 16 year old with billion dollar TVL (Total Value Locked) protocol

AUTHOR: AKASH KHOSLA
UPDATED: DANIEL GUSHCHYAN



worldwide in 2025 for Python, Rust, and Solidity.

- **Python:** ~23 million
- **Rust:** ~3.5 million
- **Solidity:** ~20,000 (0.02M)

Big Players

Nasdaq

Goldman Sachs

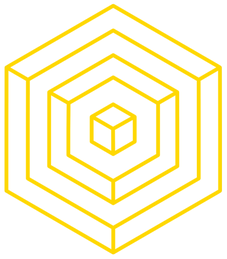
Chainlink

Google Cloud

HSBC

APTOS

space &time

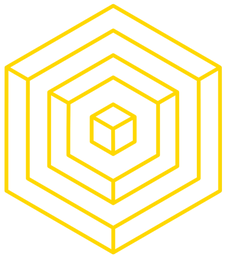


WHAT YOU CAN DO?

You can impact industries that you previously you could not enter

- **For example, DeFi - Decentralized Finance**
  - Financial services industry previously hard to break into without enough capital and power
  - Use cases built by distributed ledger technology
    - Automated Market Makers (AMMs)
    - Stable currencies / stablecoins
    - Trustless derivatives trading
    - Democratized financial and banking access

AUTHOR: SIMON GUO
UPDATED: MOHAMMED ALOBAIDI



WHAT CAN YOU DO WITH IT?

More Examples

- **Experience the bleeding edge of cryptography**
  - Leverage Zero Knowledge Proof technologies such as SNARKs and STARKs to facilitate peer-to-peer **private** transacting and computing
  - Build systems that preserve peoples rights and privacy

WHAT CAN YOU DO WITH IT?

More Examples

- **Creator economy**
  - Blockchain allows creators to cut out middlemen in the distribution of their content to fans
  - It allows fans to invest in their favorite creators, creating an open creator economy
  - De-platforms creators making for censorship resistant consumption layers
  - Allows creators to keep track and manage IP so they are fairly rewarded for their work
  - Globalizes royalty distribution

WHAT CAN YOU DO WITH IT?

More Examples

- **Defi Gaming**
  - lots of potential, need blockchain tech to be even more mainstream
  - Kevin Chou
    - spent 7 years trying to create crypto games
    - Fote: integrate crypto directly into games
    - SuperLayer: building consumer-focused crypto products
    - “good idea, not right timing yet”

KABAM

GEN.G

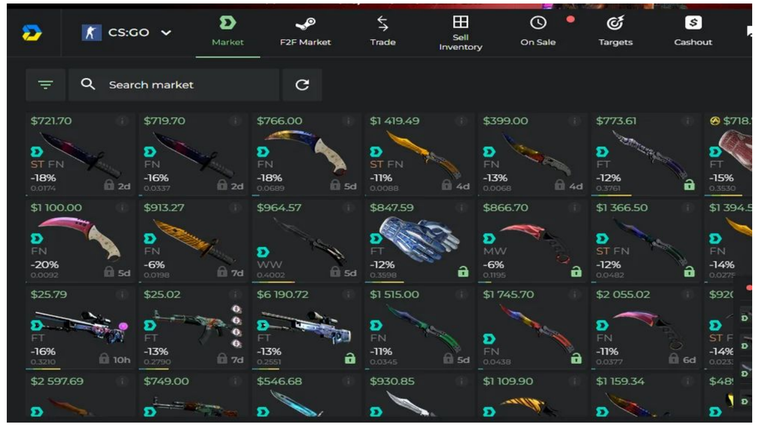
BLOCKCHAIN AT BERKELEY

AUTHOR: Mohammed Alobaidi

WHAT CAN YOU DO WITH IT?

More Examples

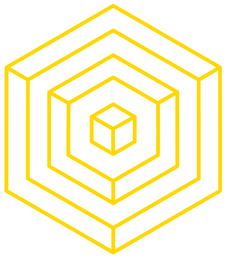
- **NFTs**
  - Fun fact: CSGO's skin marketplace was major precursor of nft marketplaces
    - in game tradable items have real world value
  - NFTs in fashion
    - verify luxury goods, authentication
    - digital fashion, collectible, build community



CSGO marketplace

Louis Vuitton NFT
Vogue 2024

AUTHOR: Mohammed Alobaidi



Summary

What's waiting for you

With blockchain, you can:

- Build global financial apps
- Build software that has a real, worldwide social impact
- Build infrastructural protocols, not just one-off apps
- Disrupt many industries by decentralizing business models

2 COURSE DETAILS

THE GOAL

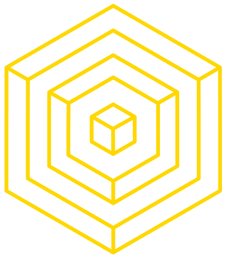
This course aims to teach students **the fundamentals of blockchains development**, the **Ethereum ecosystem** and the **Solidity programming language**. We also introduce **industry-relevant tools** such as Foundry in an accessible, collaborative environment.

This semester, we will also be delving into **Rust** development, and use tools such as **CosmWasm** to explore the frontiers of blockchain innovation.

We hope that through this course, students will become more confident in their ability to **develop and deploy blockchain-based solutions on important industry issues.**

Course GitHub

https://github.com/BerkeleyBlockchain/fa25-dev-decal

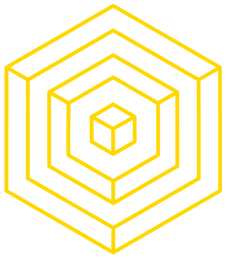


PREREQUISITES

This course requires backgrounds in Computer Science and Programming

This course requires an understanding of fundamental Computer Science concepts as well as some programming abilities. You should have taken **CS61A** Structure and Interpretation of Computer Programs at least and it is recommended that you have also taken **CS61B** Data Structures and **CS61C** Machine Structures, or have the equivalent skills.

Bulk of the course will be focused on programming languages like **Solidity** (similar to JavaScript), **JavaScript**, and **Rust**. Comfort with command line tools such as Git and NPM will help.



SUPPORT

Ask for help!

- As previous experience with programming is required in enrolling for this course, we do expect that students be on some level familiar with developing software already.
- However, it is likely that many of the tools and concepts that we introduce in this course will be unfamiliar and sometimes difficult to work with or implement.
- Please do not hesitate to ask any of the DeCal staff for help!

GRADING

The grading is **P/NP** for the course and will be primarily based on completion and participation. In order to pass the course, students will need to finish all the labs, complete the final project*, and attend all lectures.

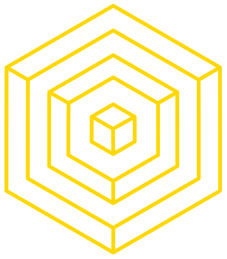
*To be determined

ATTENDANCE:

- This class is synchronous, recordings will (likely) be posted on the course website.

- Live lecture is from 5:00-6:30pm every Wednesday.
  - Part of this time will be lecture, the rest in-person workshopping.

- We will do our best to be as understanding and accommodating for conflicts as possible.



VITAMIN:

- Each lecture will be accompanied by a short quiz that we're calling 'Vitamins'
- These are completion based, meant to help you synthesize the information you learned during lecture
- We'll be partially tracking attendance via the completion of these vitamins

Homework:

In blockchain development, you can only learn by doing.

- Each lecture will (typically) be accompanied by an assignment
  - We will aim to begin working on these assignments **together** in class. If you don't finish in class or have to early, you can submit it from home by **11:59pm the following Tuesday** (night before lecture)

- Homework is due before next week's lecture. We will be grading based on completion.

- Homework will be posted weekly on the course GitHub repository. https://github.com/BerkeleyBlockchain/fa25-dev-decal

Homeworks :

In blockchain development, you can only learn by doing.

- Homeworks check your understanding, and each part of the lab will build on the previous to lead to a complete project.

- Therefore, it’s mandatory to complete these assignments on time- to ensure both your learning and to pass the class.

EXPECTATIONS
ADMINISTRIVIA

Expect from us:

- Review of Blockchain Fundamentals concepts that are essential to understanding the developer side.
- Technical, in-depth talks that can go as far down as the protocol code.
- Rewarding programming assignments.
- A mildly frustrating, but extremely unique experience that you can’t find elsewhere.

POLICIES
ADMINISTRIVIA

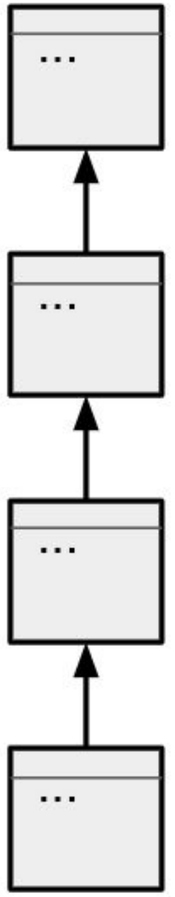
- Plagiarism and academic dishonesty is strictly prohibited and is treated with automatic failure of the course. Working collaboratively and discussing ideas are encouraged.
- Please do not post your solutions on public repository on sites such as GitHub or GitLab. Please make them local or set them to private repository.
- If you have any questions, please feel free to contact any course staff.

3 BLOCKCHAIN & ETHEREUM OVERVIEW

3.1 BLOCKCHAIN REVIEW

DEFINITIONS

QUICK REVIEW



- **Cryptocurrency:** A form of currency that's stored completely digitally, and isn't issued by a central authority. Made secure with cryptography, distributed consensus, and economic incentive alignment.
  - Bitcoin is a cryptocurrency.

- **Blockchain:** The data structure used to enable decentralized cryptocurrencies. Stores information in a way that allows multiple parties to agree upon the data and access it without having to trust one another.
  - Cryptocurrencies are a use case of Blockchain. The data stored is the transfers of money.

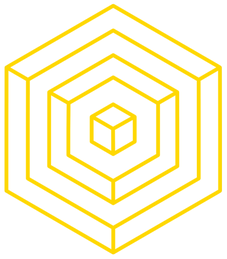
BLOCKCHAIN OVERVIEW

CHARACTERISTICS

- **Decentralization:** Communal consensus, rather than one party's decision, dictates who gets to access or update the blockchain. No user / entity on the blockchain has greater privileges than another.

- **Immutability:** Information committed to the blockchain cannot be removed.

- **Tamper-evidence:** It's immediately obvious if data stored on the blockchain has been tampered with.



BLOCKCHAIN OVERVIEW

CONSENSUS

- **Nakamoto Consensus:** to add to the blockchain, one has to provably spend resources.

- Choice of the resource to expend differentiates the different consensus mechanisms.
  - Proof of Work: computational power, electricity
  - Proof of Stake: the network's token

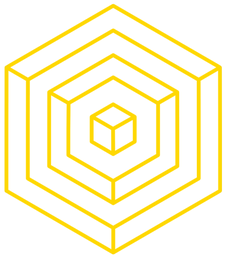
- Expending more resources increases your chance of writing the next block. Bad actors are de-incentivized to act maliciously due to the cost of doing so.

BLOCKCHAIN OVERVIEW

MINING = Proof of Work

- **Proof of Work:**
  - To prove work, miners must create a block that hashes to a **small hash value**.
    - Small hash value = “certain number of zeros in front of it”
  - To change the hash of the block, miners adjust a number called the ‘nonce’ in the block’s header
  - Likelihood of finding a block increases with ability to check more hashes

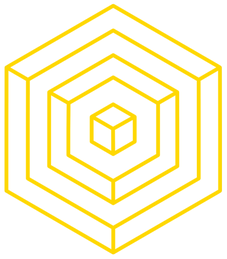
- **Proof of Stake:**
  - Validators deposit a ‘stake’ of coins into a contract on the blockchain. Validators are chosen randomly to create the next block.
  - Malicious behavior will cause the validator to be ‘slashed’- their stake burned.



Bitcoin Lifecycle Overview

THE MECHANICS

- Alice wants to send some of her Bitcoin to Bob.
- Alice creates a transaction that sends Bob some bitcoin.
- Alice submits this transaction to the “mempool”- where all pending transactions are held.
- Hundreds of thousands of miners in the world pick up transactions from the mempool and start mining for the next block



Bitcoin Lifecycle Overview

THE MECHANICS

- One of these miners finds a valid nonce, and propagates the completed block to the rest of the network.

- Hundreds of thousands of nodes on the network verify that all the transactions included in the block are legitimate, and that the block is legitimate

- Once nodes have verified the block, they will expect miners to submit the next block as having been built on top of this most recent one.

Bitcoin Lifecycle Overview

THE MECHANICS

- As soon as the block containing Alice's transaction is confirmed, she knows that her TX has been accepted. But she cannot trust that it is final yet.

- There's still a chance that a different block that was made at the same time or earlier has not been propagated fully yet.

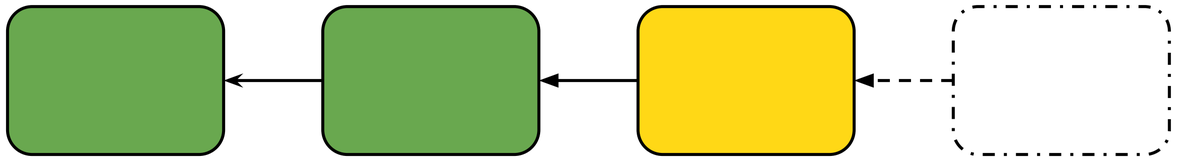
- When a node receives a block that 'forks' from the chain that the node has, the node will accept the longest fork.

AUTHOR: DANIEL GUSHCHYAN

Bitcoin Lifecycle Overview

THE MECHANICS

- When a node receives a block that ‘forks’ from the chain that the node has, the node will accept the longest fork.

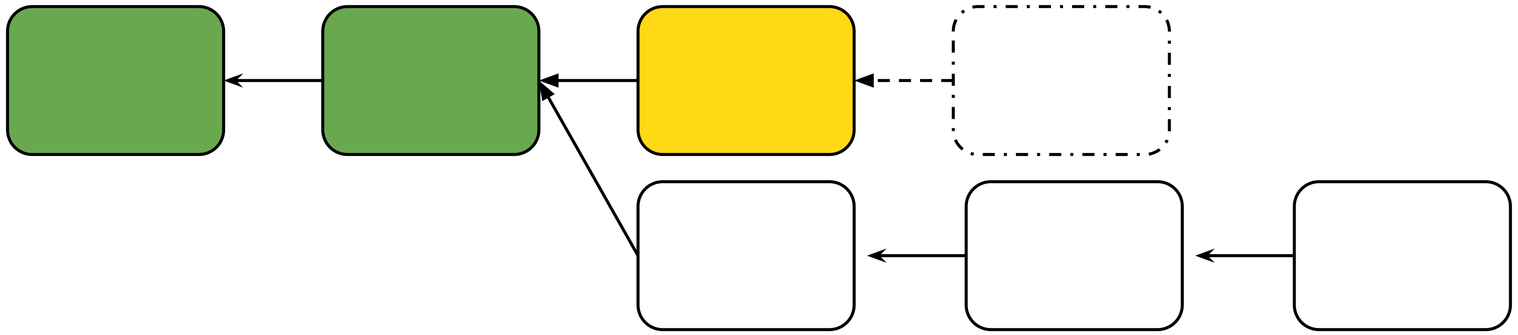


AUTHOR: DANIEL GUSHCHYAN

Bitcoin Lifecycle Overview

THE MECHANICS

- When a node receives a block that ‘forks’ from the chain that the node has, the node will accept the longest fork.

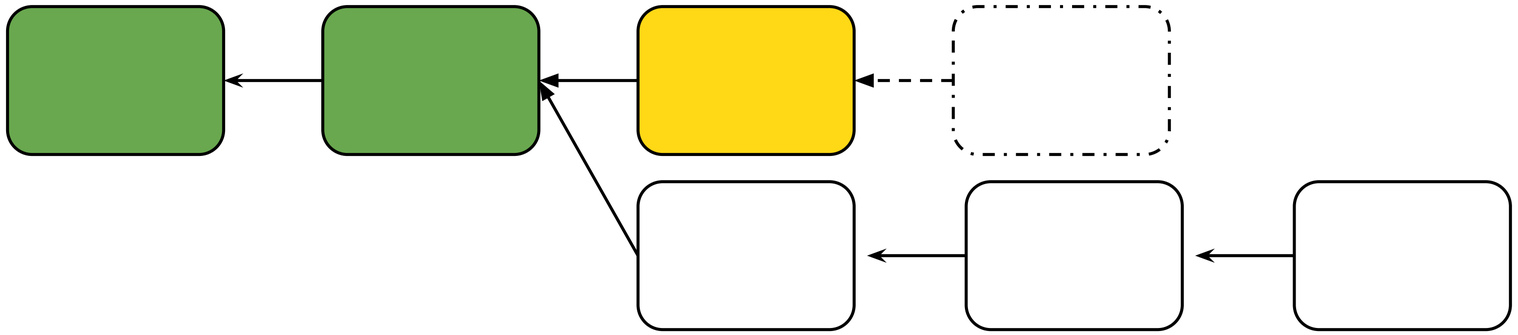


AUTHOR: DANIEL GUSHCHYAN

Bitcoin Lifecycle Overview

THE MECHANICS

- When a node receives a block that ‘forks’ from the chain that the node has, the node will accept the longest fork.



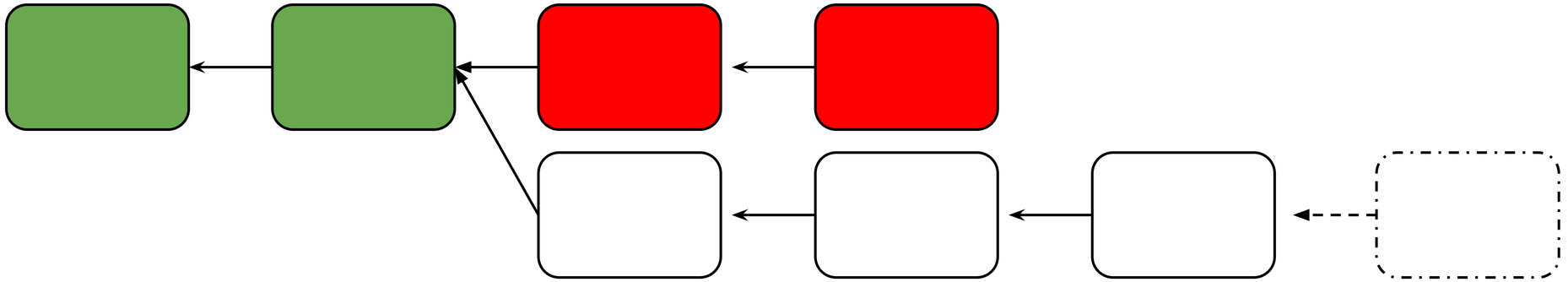
- The chain with the most work is considered the valid chain

AUTHOR: DANIEL GUSHCHYAN

Bitcoin Lifecycle Overview

THE MECHANICS

- When a node receives a block that ‘forks’ from the chain that the node has, the node will accept the longest fork.



- The chain with the most work is considered the valid chain
  - Look up “Ethereum Hard Fork”!

AUTHOR: DANIEL GUSHCHYAN

Bitcoin Lifecycle Overview

THE MECHANICS

- By the time the block containing Alice's transaction is 6 blocks deep into the chain, Alice can be reasonably sure that her transaction is final.
  - By 'final', we're referring to a low probability of a longer chain being mined.

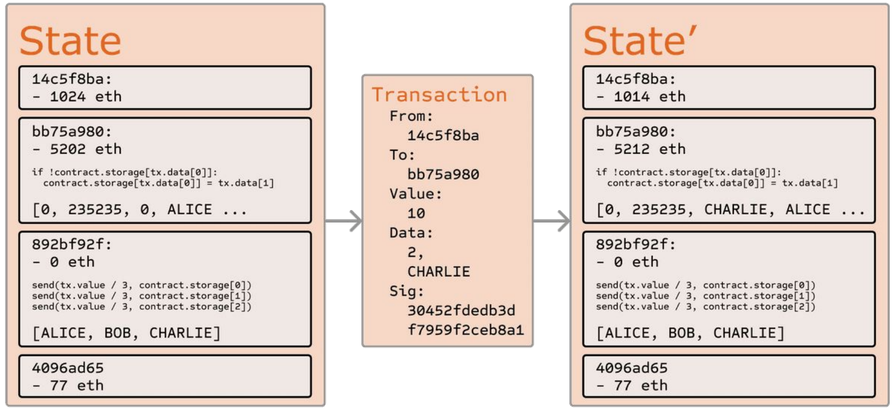
- Everyone now agrees that Bob received the bitcoin.

3.2 ETHEREUM OVERVIEW

WHAT IS ETHEREUM?

HIGH LEVEL OVERVIEW

- Ethereum is a blockchain designed to run code known as **smart contracts**
  - **Distributed state machine** - transactions change a global state
    - transactions == state transaction function
- Think one big, distributed computer in the cloud.



AUTHOR: TIM HUANG
UPDATED: DANIEL GUSHCHYAN

DECENTRALIZED APPLICATIONS

PREDICTION MARKETS AND CATS

- **Augur**: Prediction Markets
  - Market for any event
  - Censorship resistant, automatic trustless payments
- **Cryptokitties:** trade virtual cats
  - Non-Fungible Tokens, or NFTs, are completely unique and cannot be replicated
  - complete ownership of digital assets

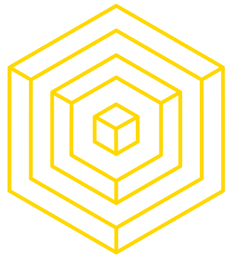
AUGUR

AUTHOR: TIM HUANG

ETHEREUM VS BITCOIN

GENERAL OVERVIEW

Ethereum	Bitcoin
Smart Contract Platform ○ Complex and feature-rich ○ **Turing complete scripting language**	Decentralized Asset ○ Simple and robust ○ Simple stack-based language; not Turing complete
Account-based for transaction	**UTXO-based for transactions**
Turing complete scripting language that enables smart contracts. Ether asset is not the primary goal. Uses proof of work, plans to do proof of stake.	**Uses Bitcoin Script, a very simple stack based language.** Currency itself the main goal. Uses proof of work.
~12 sec block time - using ethash mining	~10 min block time - using sha256 mining



3.3 ACCOUNTS

UTXOS VS ACCOUNTS

UTXOS - PAPER BILLS

UTXO stands for Unspent Transaction Output

- **Transactions** spend outputs from previous transactions and generate new outputs for transactions in the future
- **Balance** is the sum of unspent transaction outputs associated with the addresses owned by the user.
- **Analogy: Paper Bills**

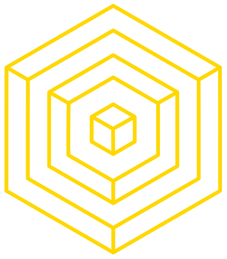
AUTHOR: TIM HUANG

UTXOS VS ACCOUNTS

ACCOUNTS - DEBIT CARD

Accounts automatically track the balance of a user

- An account has **state** and 20 byte/160 bit byte **identifier**,
  - **i.e: 0x91fff4cbd6159a527ca4dcce2e3937431086c662**
- **Two Types of Accounts:**
  - Externally owned: controlled private keys and have no code associated with them.
  - Contract accounts, which are controlled by their code and have code associated with them.



ACCOUNT STATE

FOUR COMPONENTS

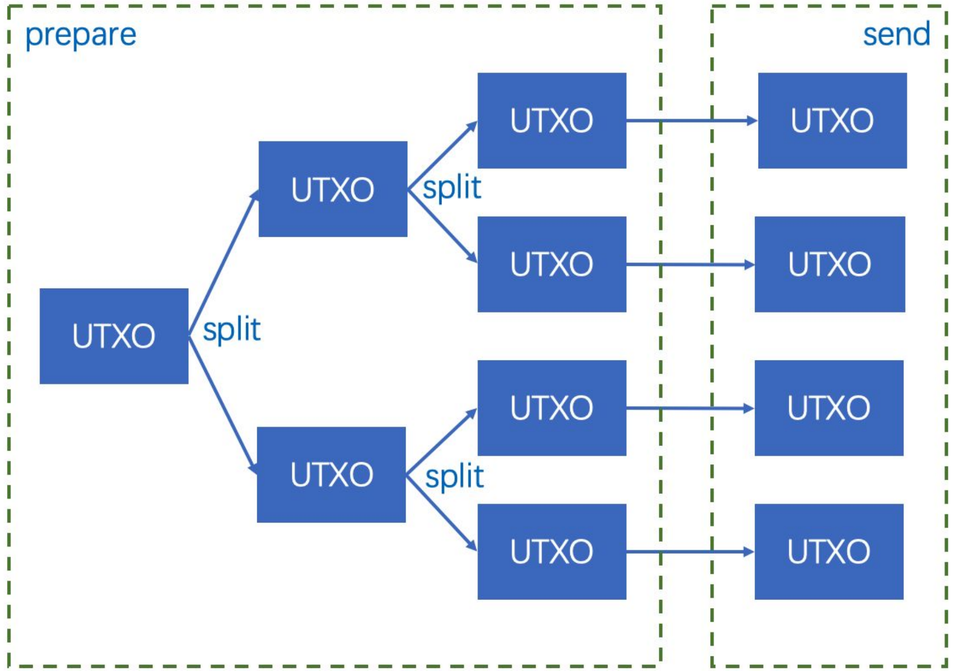
What is stored in account state?

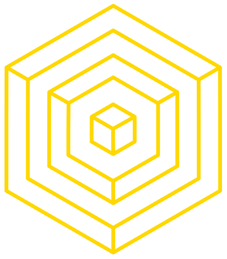
- **nonce**: the number of transactions that are sent from that account, number of contracts created if contract account
- **balance**: The number of Wei owned by this address. 10^18 Wei = 1 Ether.
- **storageRoot**: a hash of the identifier to the storage of the contract.
- **codeHash**: the hash of the code that is present in the contracts

WHY UTXOS?

BENEFITS OF THE UTXO MODEL

- **Greater Privacy**
  - Encourages users to use a new address for every transaction received
  - Harder to link accounts to real-world identities
- **Scalability**
  - Ability to process multiple UTXOs concurrently, enabling parallel transactions

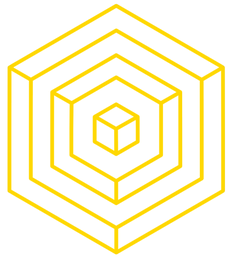




WHY ACCOUNTS?

BENEFITS OF THE ACCOUNTS MODEL

- **Storage**
  - Only need to update each account's balance instead of storing every UTXO
    - ex. 5 UTXOs: (20 + 32 + 8) * 5 = 300 bytes (20 for the address, 32 for the txid and 8 for the value)
    - Using accounts: 20 + 8 + 2 = 30 bytes (20 for the address, 8 for the value, 2 for a nonce
- **Simplicity**
  - Far more intuitive than constantly updating a UTXO set to compute balance
  - Easier to program smart contracts



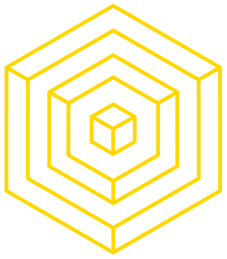
3.4 GAS

GAS AND PAYMENT

GAS EXPLAINED

What is gas?

- Fee incurred for every computation that occurs as a result of a transaction
  - Using a network requires electricity and resources. Gas fees exist to pay for these costs
- Paid by sender of transaction
- Ensures network isn't misused!



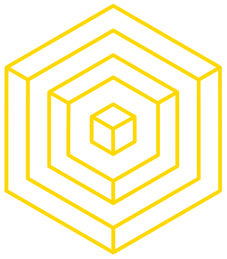
GAS AND PAYMENT

WHY DO WE NEED FEES?

- **Cost of Distributed Computation**
  - Every transaction on the EVM must be redundantly recalculated by every node on the network, which is costly
  - Due to this multiplied cost, smart contracts are best used for simple tasks

- **Unpredictability & Turing Completeness**
  - Loops and susceptibility to halting problem
  - Impossible to determine ahead of time whether a given contract will ever terminate

AUTHOR: TIM HUANG
UPDATED: DANIEL GUSHCHYAN



GAS AND PAYMENT

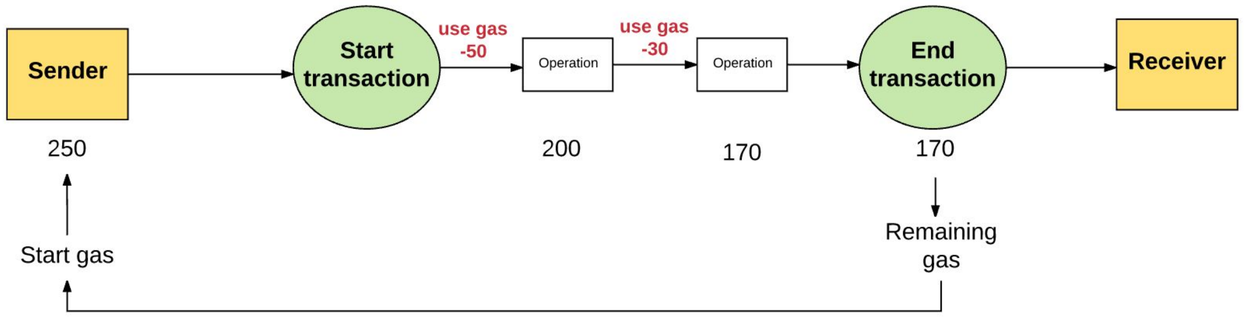
GAS LIMIT AND GAS PRICE

- **Gas:** measures the computational “work” necessary to perform an action
  - ex. one Keccak256 hash: 30 gas + 6 gas per 256 bits of data
  - Detailed in Ethereum Yellowpaper: https://ethereum.github.io/yellowpaper/paper.pdf
- **Gas limit:** maximum amount of gas the user is willing to spend
  - Prevents unexpected, undesired cost
- **Gas price:** the amount of Ether you are willing to spend on every unit of gas
  - Measured in “gwei”: 1 gwei = 10^9 wei, 1 eth = 10^9 gwei

GAS AND PAYMENT

HOW TRANSACTIONS WORK

- With every transaction, the sender specifies a gas limit and a gas price
  - gas price * gas limit = max amount sender is willing to pay
- **Example:**
  - Gas Limit: 50,000, Gas Price: 20 gwei
  - Max transaction fee: 50,000 * 20 gwei = 1,000,000,000,000,000,000 wei = 0.001 Ether
- For the transaction to be executed:
  - (1) the user has enough ether in their account to cover the maximum transaction fee, (2) the gas limit is enough to execute the transaction
- Unused gas returned to sender



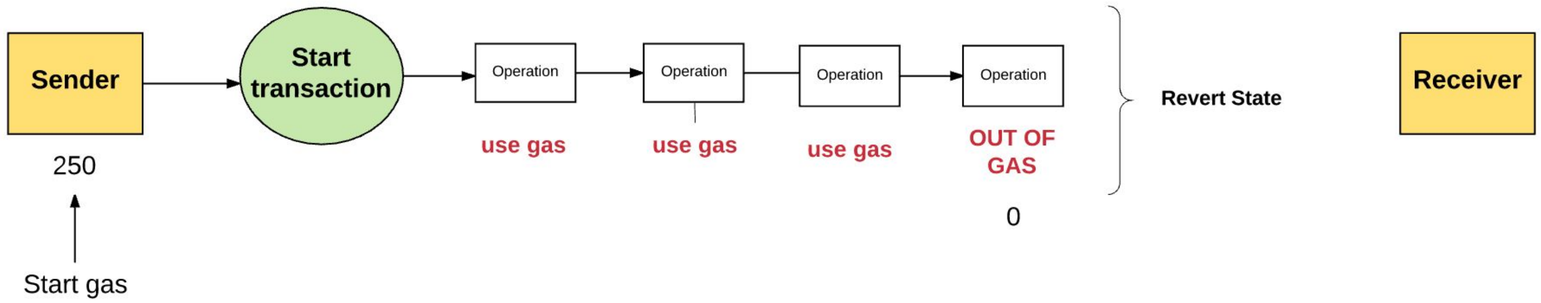
AUTHOR: TIM HUANG

GAS AND PAYMENT

WHAT IF NOT ENOUGH IS SENT?

Not enough gas to execute the transaction?

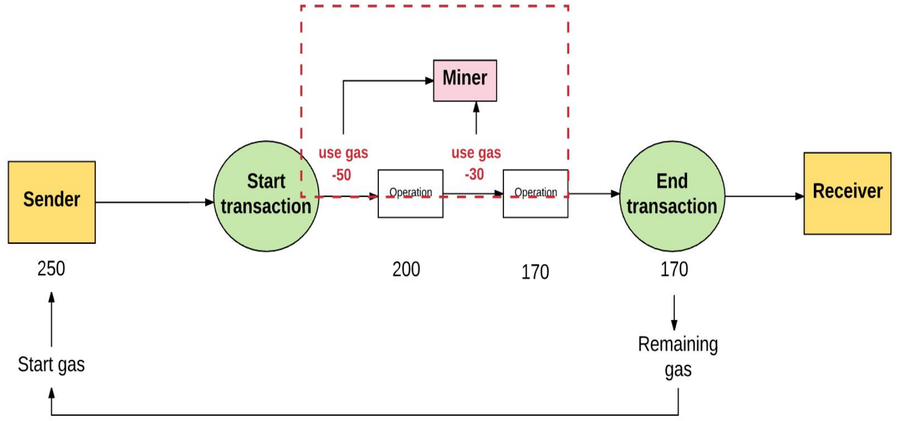
- Transaction runs out of gas and therefore is considered invalid
- State changes are reversed, failing transaction recorded
- Since computation was already expended by the network, none of the gas is refunded



GAS AND PAYMENT

MINER INCENTIVES

- **Where does the fee go?**
  - The miner's address, as they have used computational effort to validate transactions
  - Incentive to contribute to network
- **Gas Price and incentives**
  - Miners can choose which transactions to validate or ignore
  - The greater the gas price the sender sets, the more likely the miner is to select the block

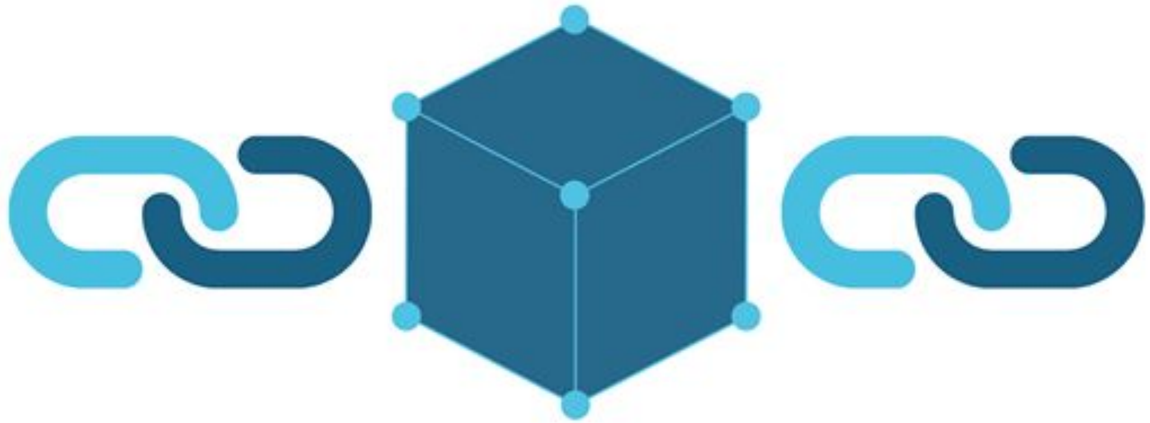


AUTHOR: TIM HUANG

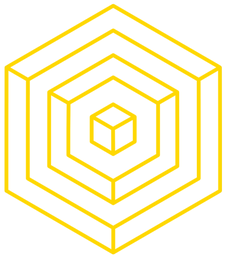
GAS AND PAYMENT

STORAGE FEES

- **Gas** is also used to pay for storage, proportion to the **smallest multiple of 32 bytes** used
- Increased storage increases the size of the Ethereum state on all nodes
  - Incentives to keep data small
- Refund given if a transaction frees up data in storage



3.5 SMART CONTRACTS



SMART CONTRACTS

CONTRACTS

con·tract

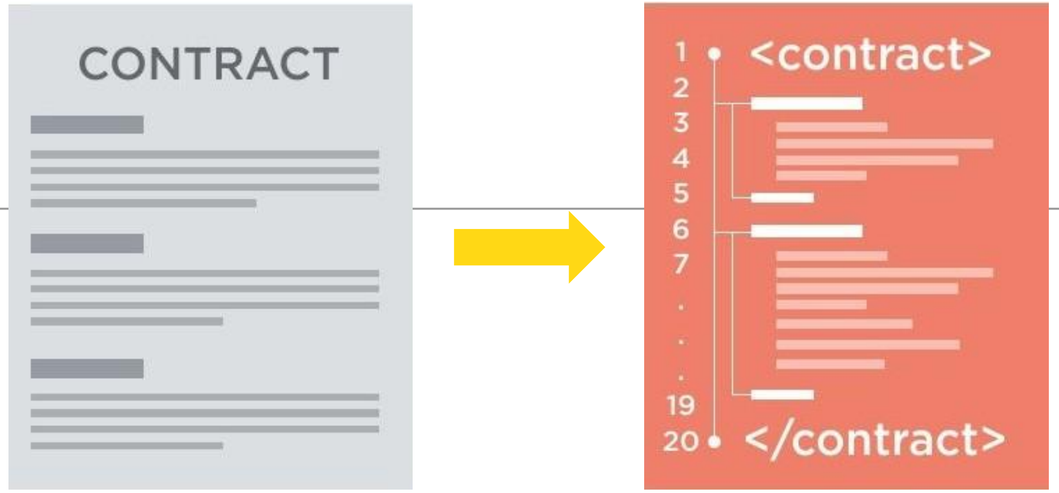
(noun) /ˈkäntrakt/

1. a written or spoken agreement ... that is intended to be enforceable by law.

AUTHOR: PHILIP HAYES

SMART CONTRACTS

SMART CONTRACTS



smart con·tract

(noun) /smärt ˈkäntrakt/

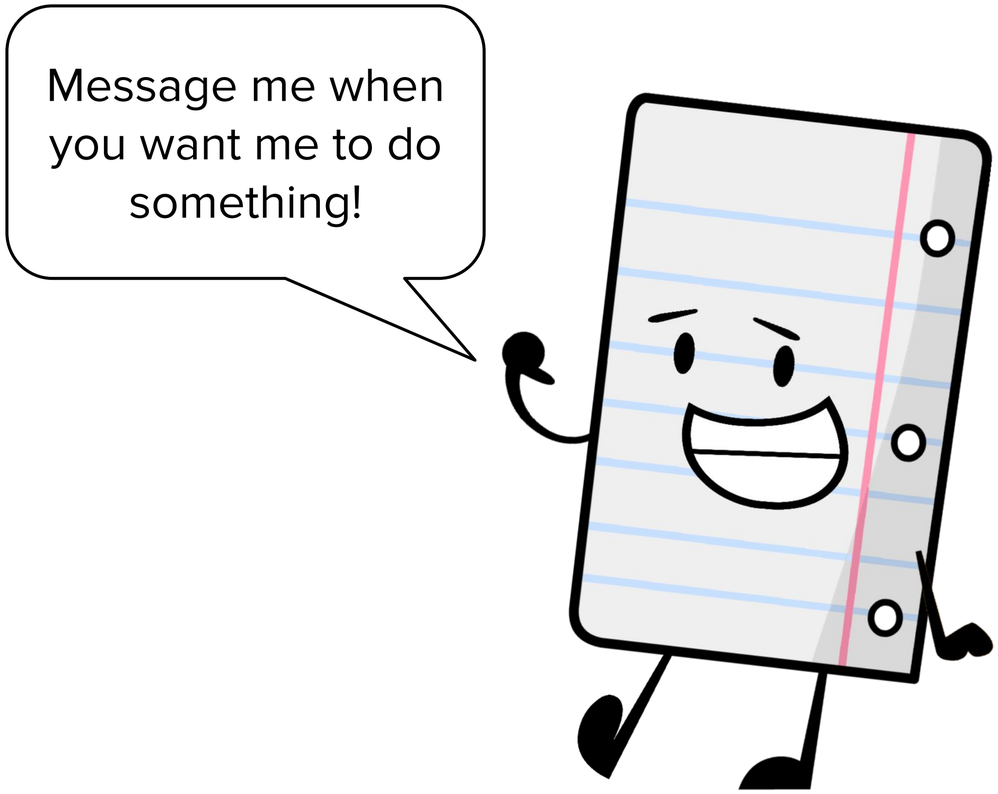
1. code that **facilitates**, **verifies**, or **enforces** the negotiation or execution of a digital contract.

AUTHOR: PHILIP HAYES

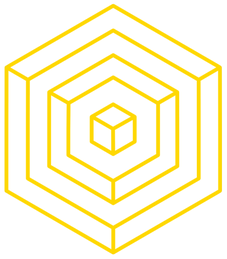
ETHEREUM SMART CONTRACTS
IN CONTEXT

Smart Contracts in Ethereum are like autonomous agents that live inside of the Ethereum network

- React to external world when "poked" by transactions (which call specific functions)
- Have direct control over:
  - **internal ether balance**
  - **internal contract state**



AUTHOR: PHILIP HAYES



ETHEREUM SMART CONTRACTS

PURPOSES

Ethereum Contracts generally serve four purposes:

- **Store and maintain data**
  - Data represents something useful to users or other contracts
  - Ex: a token currency or organization's membership
- **Manage contract or relationship between untrusting users**
  - Ex: financial contracts, escrow, insurance
- **Provide functions to other contracts**
  - Serving as a software library
- **Complex Authentication**
  - Ex: M-of-N multisignature access

ETHEREUM SMART CONTRACTS

IMPORTANT ATTRIBUTES

- **Immutable**
  - No tampering post deployment
- **Public Code**
  - Can request source code and compile
- These two characteristics ensure that absolute trust in the smart contract is possible since:
  - Functionality is universally visible
  - “The Rules” will always be constant

ETHEREUM SMART CONTRACTS

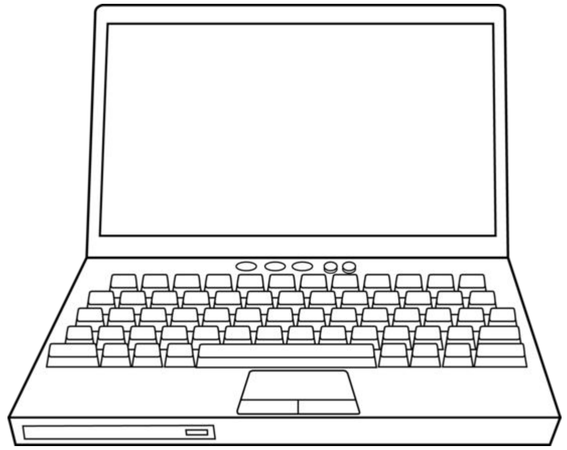
SAMPLE BETTING CONTRACT

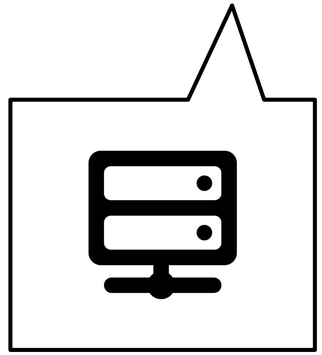
```
contract Betting {
     address public owner;
     address public gamblerA, gamblerB, oracle;
     uint[] outcomes;
     struct Bet {
         uint outcome;                          /* Defines a Bet */
         uint amount;
         bool initialized;
     }

     mapping (address => Bet) bets;             /* Keep track of every gambler's bet */
     mapping (address => uint) winnings;        /* Keep track of every player's winnings */
     ...
     function makeBet(uint _outcome) payable returns (bool) { … }
     function makeDecision(uint _outcome) oracleOnly() { … }
     function withdraw(uint withdrawAmount) returns (uint remainingBal) { … }

}
```

ETHEREUM VIRTUAL MACHINE

HIGH-LEVEL OVERVIEW

- The **EVM (Ethereum Virtual Machine)** is a “mini computer” that runs contract code
- Contract code that actually gets executed on every node is EVM code
  - **EVM code**: low-level, stack based bytecode language (i.e. JVM bytecode)
- Every Ethereum node runs EVM

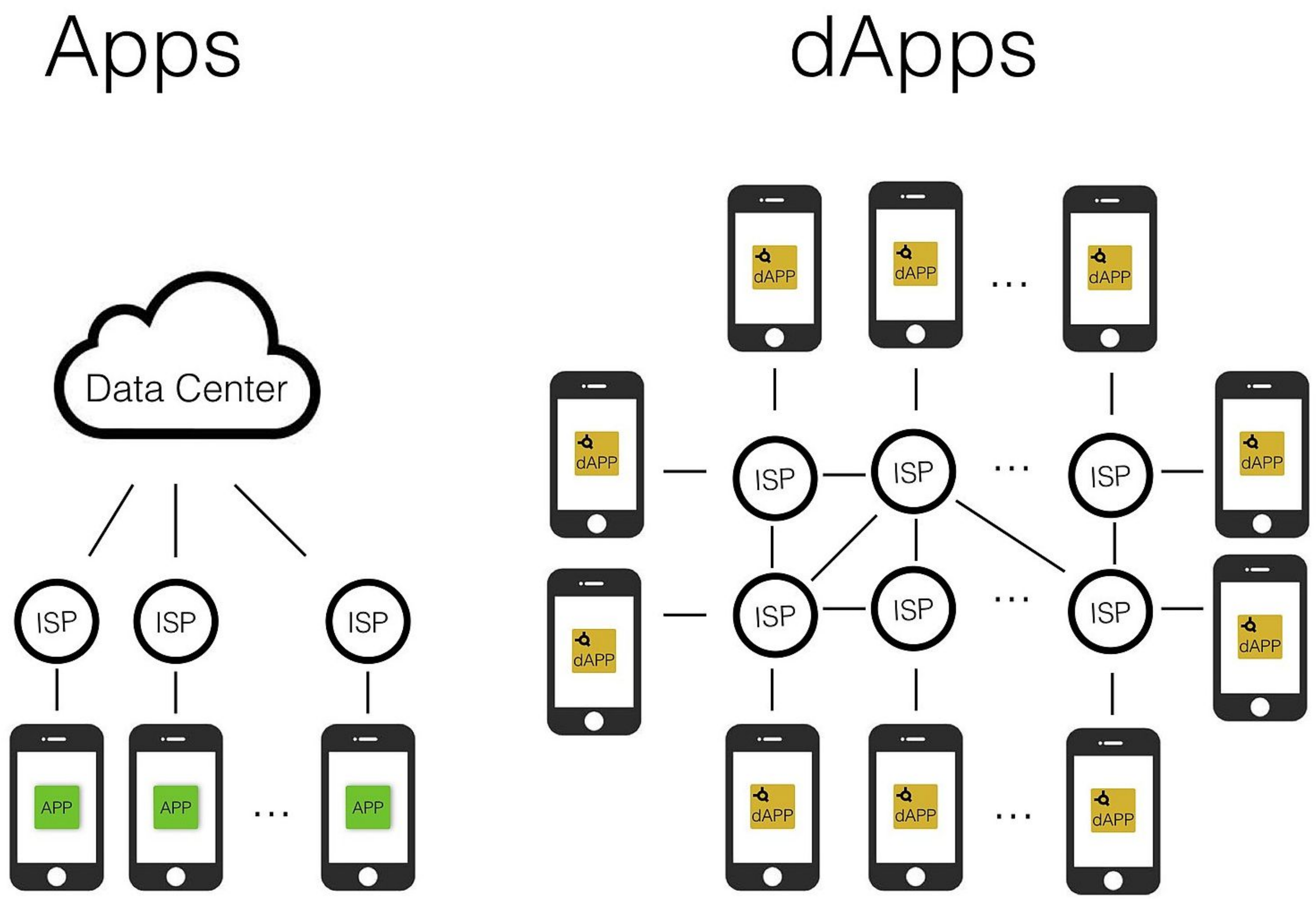




AUTHOR: PHILIP HAYES

DAPPS
HIGH-LEVEL OVERVIEW

- Applications implemented via smart contracts
- Open source code = transparency
- Running on blockchain as opposed to a central server
- May be interfaced with via centralized front-ends
- (Potentially!) Immutable
- Peer-to-peer software

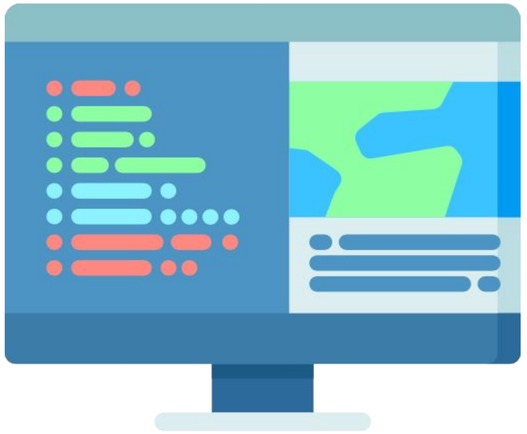


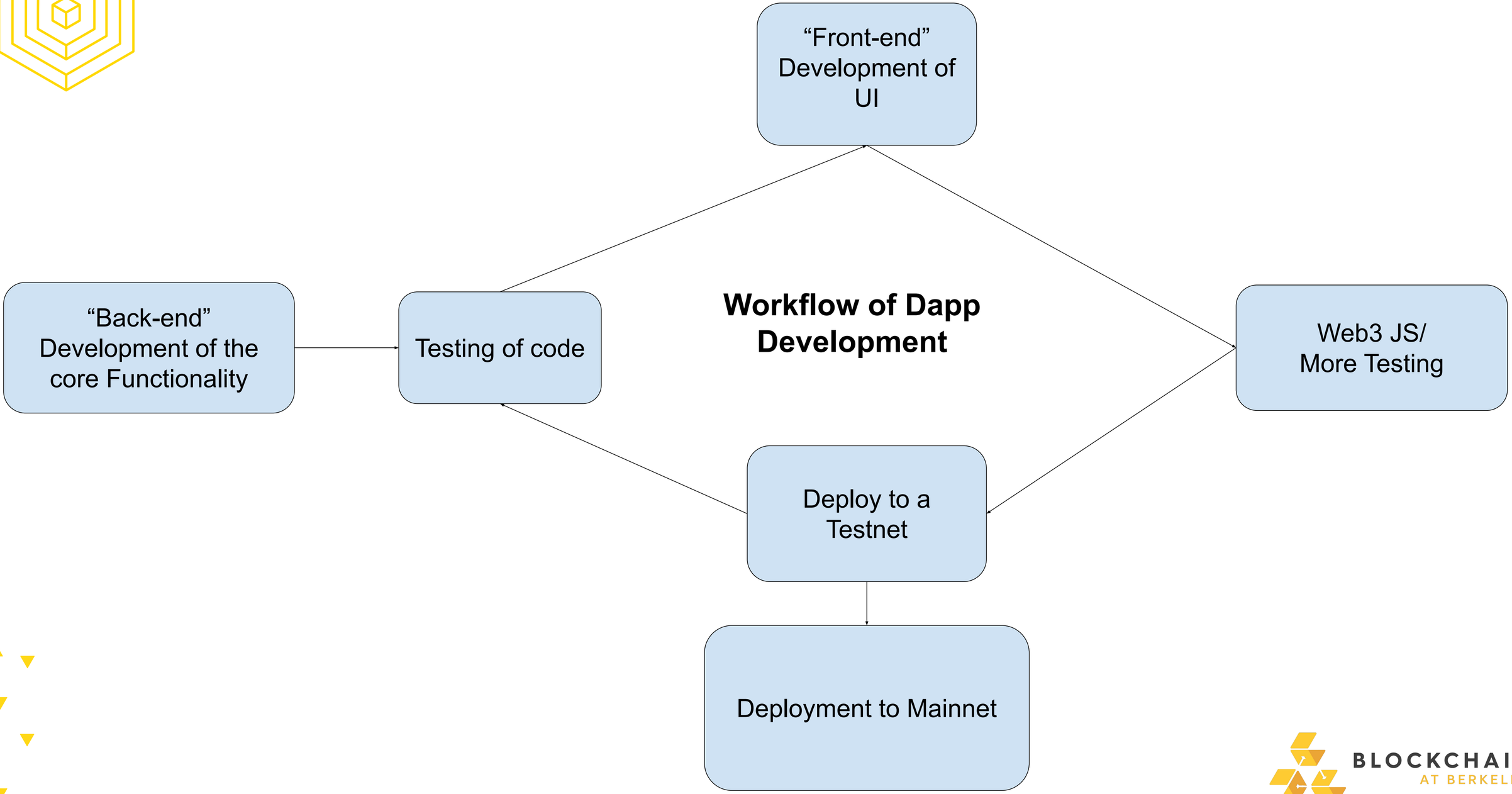
Source: https://towardsdatascience.com/what-is-a-dapp-a455ac5f7def

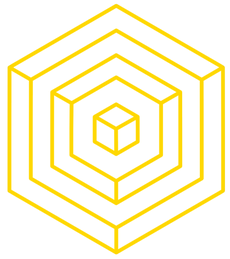
WORKFLOW

DAPP DEVELOPMENT PROCESS

- Get it right the first time!
- TEST-DRIVEN DEVELOPMENT
- Can be compared to Hardware Development as opposed to traditional software app development

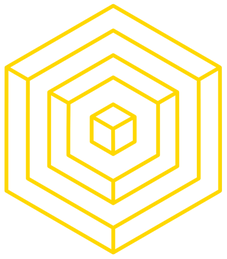



“Front-end”
Development of
UI
“Back-end”
Development of the
core Functionality
Testing of code
Workflow of Dapp
Development
Web3 JS/
More Testing
Deploy to a
Testnet
Deployment to Mainnet



4 HW1: BUILD A BLOCKCHAIN IN PYTHON

BLOCKCHAIN
AT BERKELEY



Purpose of this exercise

Credit to Omkar Shanbhag

This notebook is meant to be a short introduction to Blockchain implementations, aimed at helping us take the topics we learn about in fundamentals, and seeing how they translate to code.

CRYPTOGRAPHIC HASH FUNCTIONS

INTEGRITY OF INFORMATION

How do we ensure trust in communication in a trustless environment?

⇒ With **cryptographic hash functions**

Image source: https://spiritegg.com/wp-content/uploads/2016/03/63180952_fingerprint_types624.jpg

CRYPTOGRAPHIC HASH FUNCTIONS

AVALANCHE EFFECT

Cryptographic hash function:

A hash function with three special properties:

1. Preimage resistance
   - You can't figure out the original message just from its hash.
2. Second preimage resistance
   - You can't find a different message that gives the same hash as a given one.
3. Collision resistance
   - You can't find any two different messages that give the same hash.

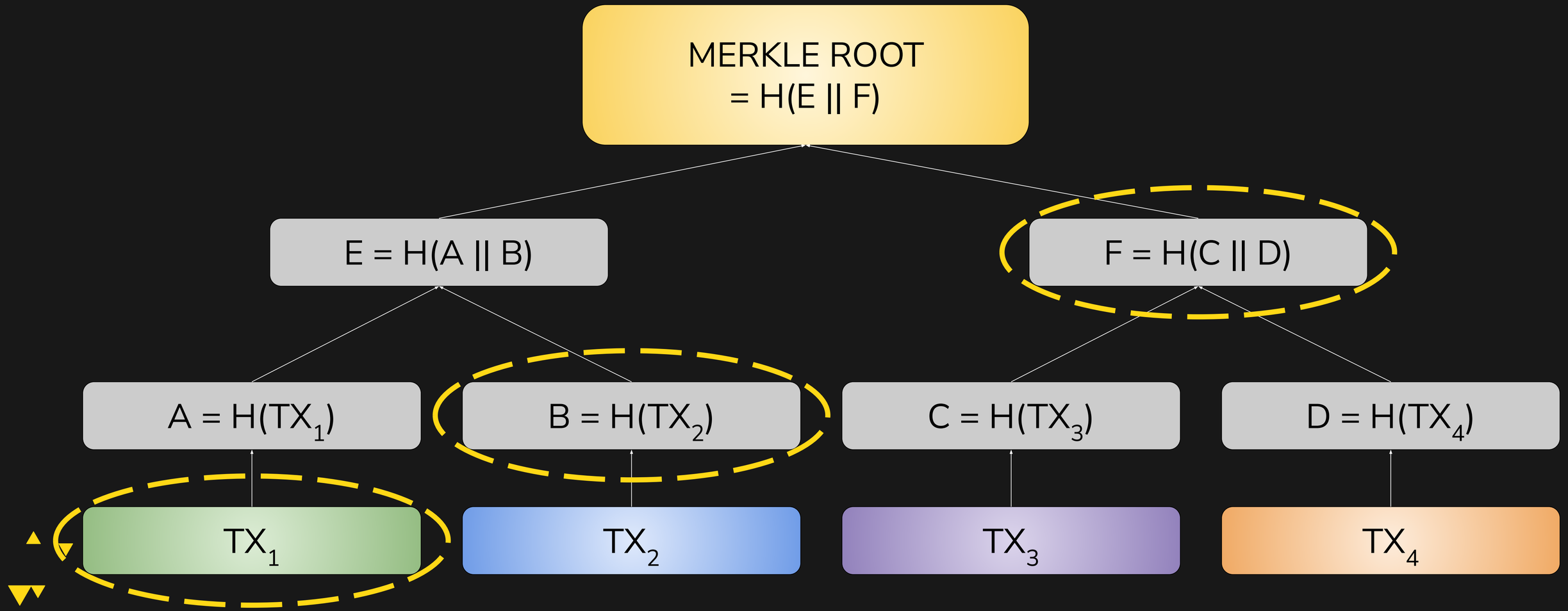
```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

Important Distinction:

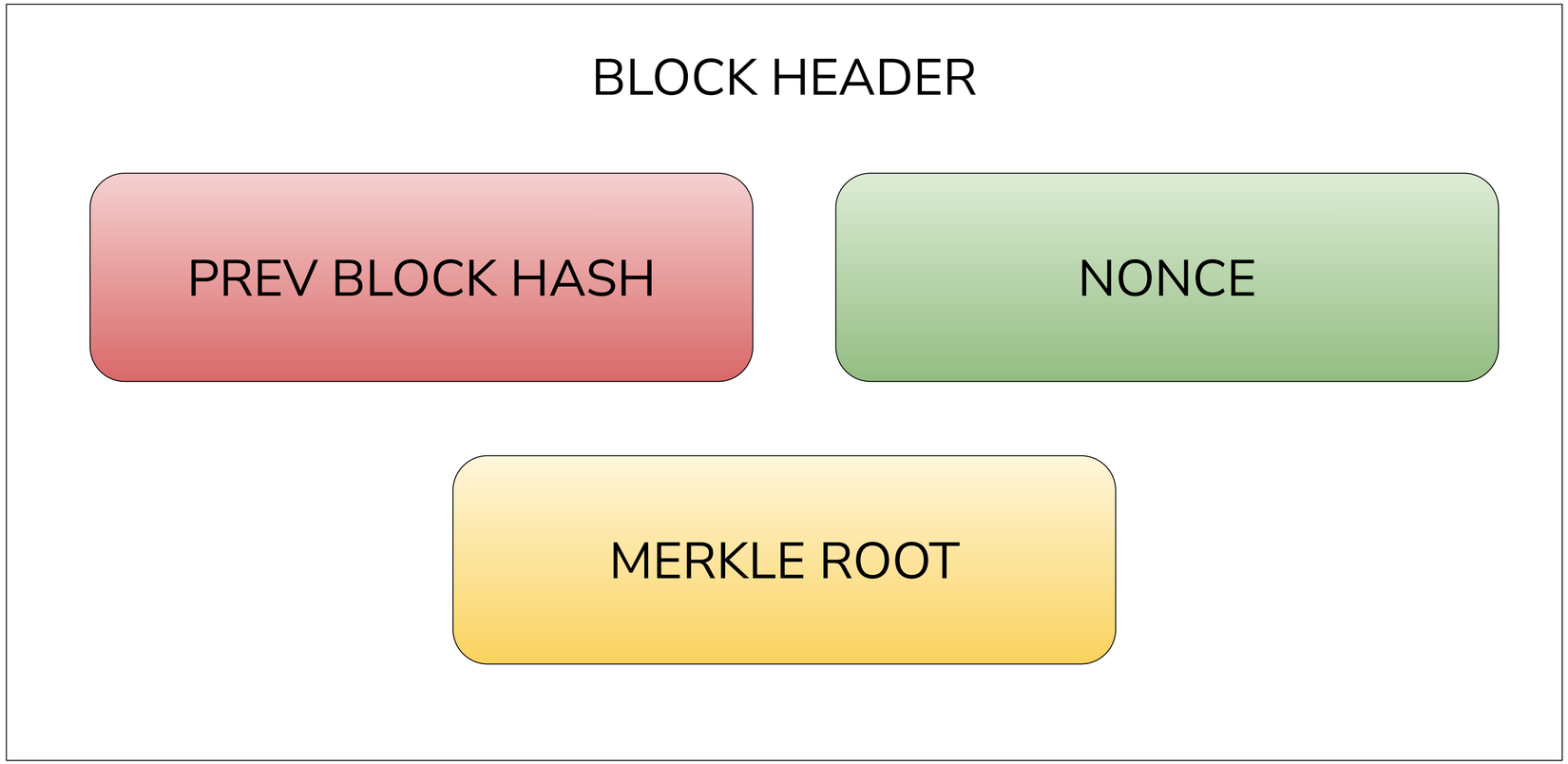
What makes it cryptographic is not the ***absence*** of collisions (impossible), but that ***finding*** those collisions is computationally infeasible.

A TAMPER-EVIDENT DATABASE

MERKLE BRANCH & PROOF OF INCLUSION

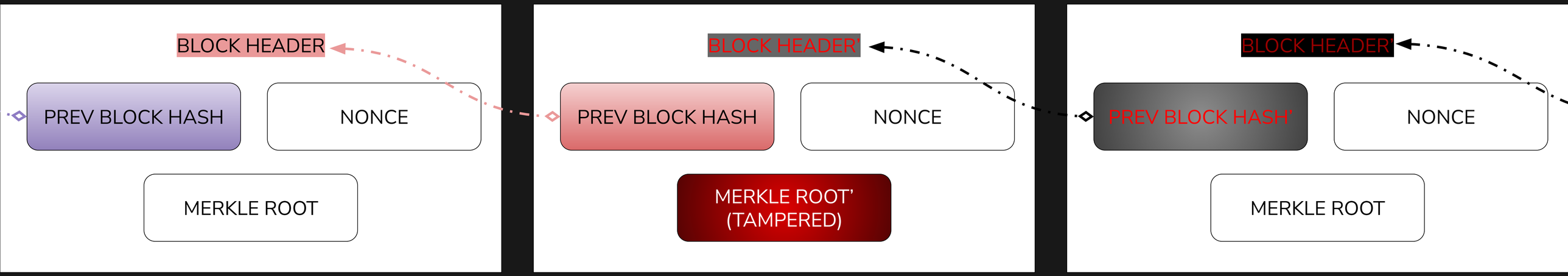


COMPONENTS OF A BLOCK



A TAMPER-EVIDENT DATABASE

PROTECTING THE CHAIN



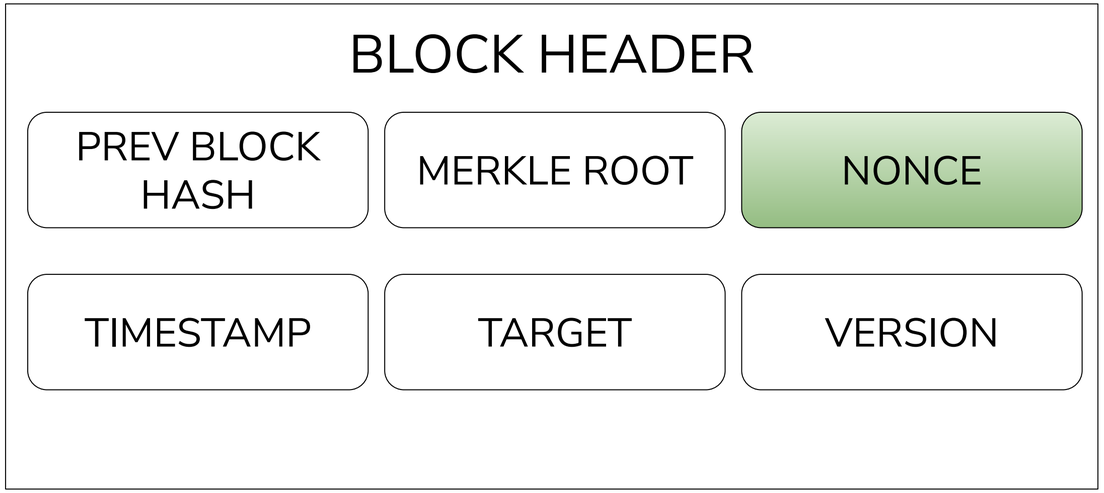
SHA256(SHA256(x))

```
BlockHash = H(prevBlockHash || merkleRoot || nonce)
```

RECIPE FOR MINING

FIND A VALID NONCE

- Find the proof-of-work
- Expend computational power
- Incrementing header nonce first, then coinbase nonce as necessary to change puzzle




```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
      header = makeBlockHeader(transactions, coinbase_nonce);
      for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
            if (SHA256(SHA256(makeBlock(header, header_nonce))) <
      TARGET)
                  break; //block found!
      }
      coinbase_nonce++;
}
```

Figure 5.6 : CPU mining pseudocode.

Image source: Mastering Bitcoin

A TAMPER-EVIDENT DATABASE

PARTIAL PREIMAGE HASH PUZZLE

H(prevBlockHash || merkleRoot || nonce)

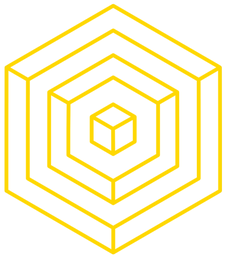
H(“Hello, world!4250”)

0x0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9

<

0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

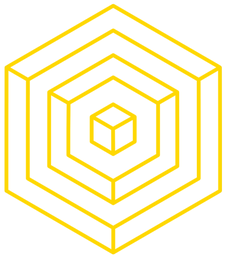
Solved!



ABOUT THIS EXERCISE

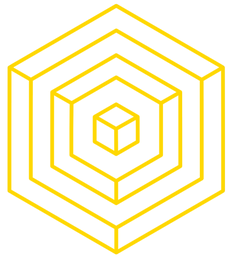
In this notebook we will implement various different aspects of Blockchain technology that we understand including:

- The Blockchain Data Structure using OOP
- A proof of work simulation
- Understanding of the concept of difficulty of finding the next block hash
- A simulation of multiple miners with varying computational powers
- A bit of data analytics to see if what we've implemented makes sense



GETTING STARTED

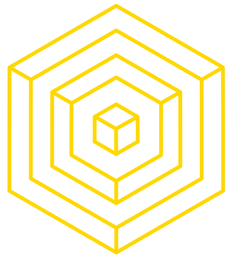
1. Download the repository from course Github using Git clone or just downloading the zip
2. Read and follow the instructions
3. Complete the skeleton code and run through the blocks
4. Fill in the blanks for conceptual questions
5. Show any instructor for check-off once you finished. If you cannot finish it in class, you have until next class for check-off.



Attendance/Vitamin

Code:
Dev

BLOCKCHAIN
AT BERKELEY



Thank You!

BLOCKCHAIN
AT BERKELEY