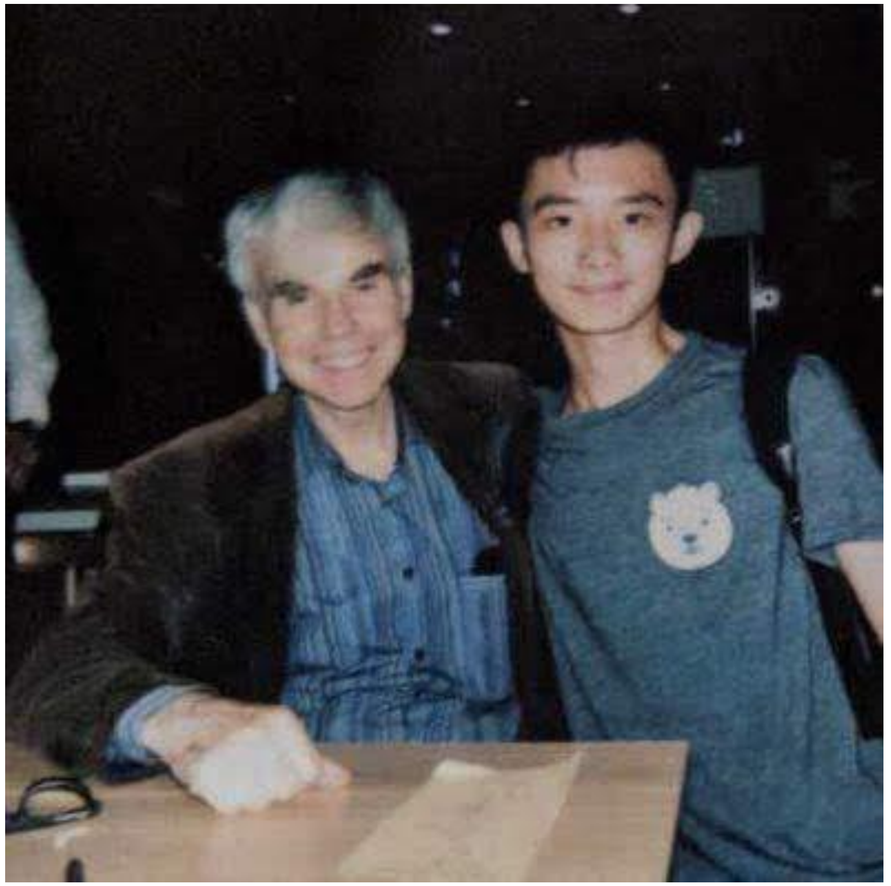
Blockchain Development: A High Level Overview

Minxing Chen
Simon Guo

THE TEAM

WHO IS HE?

- **Minxing Chen**
  - Third year CS^2 Major (Computer Science and Cognitive Science)
  - Worked and promoted a social impact project - help homeless people in Bay Area building digital identities.
  - Worked for building Interledger Protocol.
  - Led Blockchain for Developer edX project.



THE TEAM

WHO IS SHE?

- **Janice Ng**
  - Junior studying Computer Science & Cognitive Science
  - TA for edX's Blockchain Fundamentals course
  - Former Head TA for CS 198-078 Blockchain Fundamentals
  - Technical Ambassador & Developer at JP Morgan, Quorum
  - Interested in the intersection of the healthcare industry and blockchain

THE TEAM

WHO IS HE?

- **Simon Guo**
  - First-year EECS major from Canada (where Ethereum started)
  - Former TA for CS 198-078 Blockchain Fundamentals, Textbook Project Team
  - 2 winning IoT Blockchain hackathon projects at Hack the North (1st) at UWaterloo and TechCrunch Shenzhen (3rd)

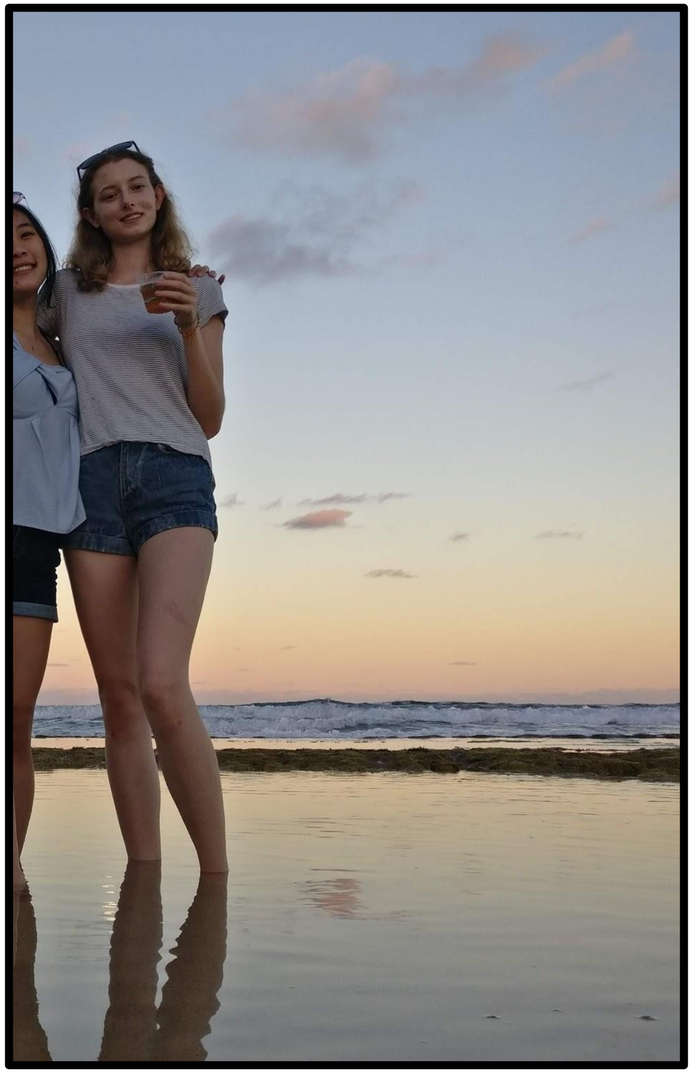
THE TEAM
WHO IS SHE?

- **Haena Lee**
  - Second-Year (Intended) Computer Science and Art Practice Major
  - Former TA for CS 198-078 Blockchain Fundamentals
  - Textbook Project Team
  - Course Administrator for Blockchain Fundamentals DeCal
  - Dark Data Project at ICSI

THE TEAM

WHO IS SHE?

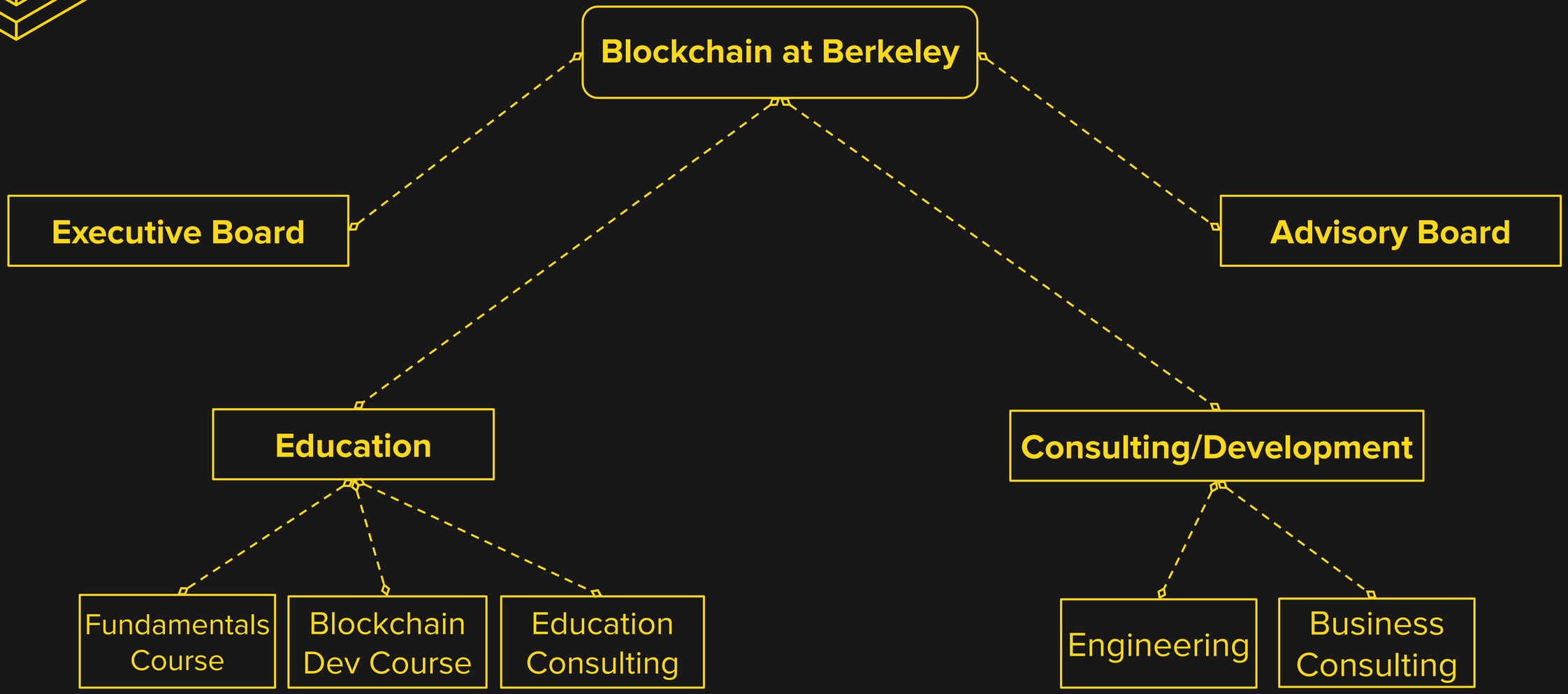
- **Grace Kull**
  - Second year Computer Science major and intended Data Science and Mathematics double minor.
  - Was on the B@B Harmonio project (demo later in this lecture) last semester that focused on streamlining buyer-seller transactions in developing countries.
  - Interested in the intersection of blockchain with politics, especially voting.



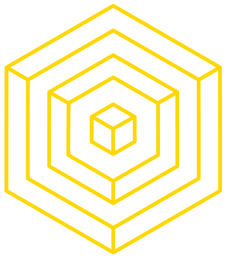
THE TEAM
WHO IS SHE?

- **Erika Badalyan**
  - Second year Behavioral Economics major through ISF
  - Former TA for CS 198-078 Blockchain Fundamentals
  - Avidly interested in blockchain-based mobile voting technology

WHO ARE WE?



BLOCKCHAIN AT BERKELEY



LECTURE OVERVIEW

1. WHY BLOCKCHAIN DEVELOPMENT
2. COURSE DETAILS
3. BLOCKCHAIN & ETHEREUM OVERVIEW
4. SOLIDITY DEVELOPMENT & DEMO
5. HW1 - BUILD A BLOCKCHAIN IN PYTHOn

BLOCKCHAIN
AT BERKELEY

1 WHY DEVELOP BLOCKCHAIN

VISIONS

WHY DEVELOP?

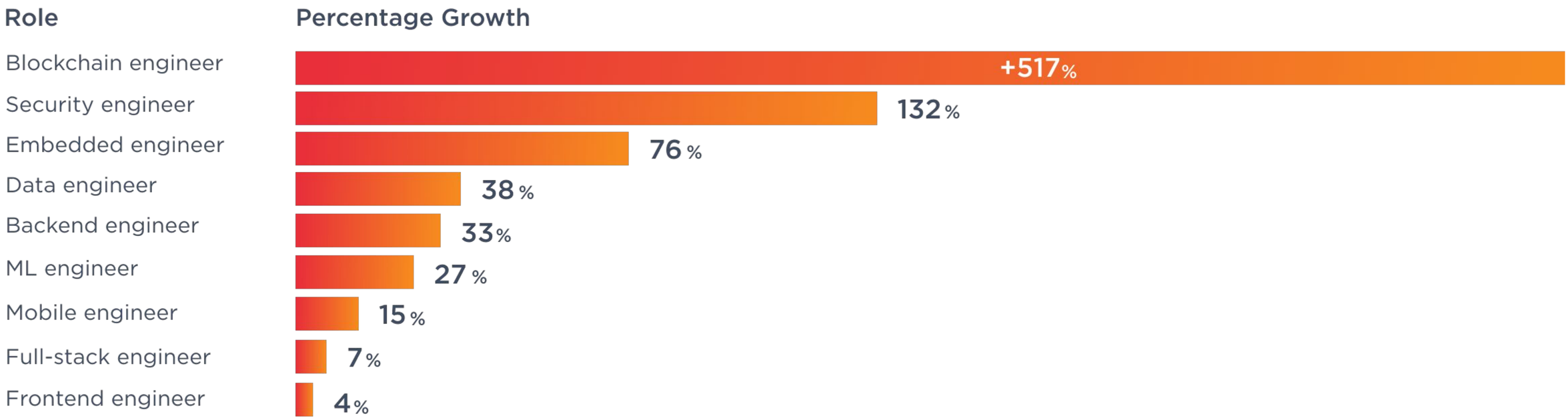
- **It’s early, but this is sort of like the web revolution of the 90’s**
  - Even some parallels to the dot-com bubble
  - Arguable that we are in a cryptocurrency bubble
- **To the majority of people, blockchain is a buzzword**
  - But it’s an important one - you’ll see why after taking this course
- **Even if blockchain is not the future, you will still learn valuable ideas by working in this space**
  - It was ICOs. Now it’s DeFi, but the applications of the future are intriguing

AUTHOR: AKASH KHOSLA
UPDATED: SIMON GUO

DEMAND FOR BLOCKCHAIN DEVS

There are now 14 job openings for every one blockchain developer.

Demand Growth for Engineering Roles



All growth data has been normalized to account for Hired's overall marketplace growth.

Souce: TechCrunch and Hired

AUTHOR: SIMON GUO

OTHER REASONS

WHY BLOCKCHAIN

- **Very open-community**
  - Accessible to anyone, most of the 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.
- **High demand, low supply of developers**
  - That’s why Blockchain at Berkeley is successful
- **Easy to get internships/jobs and make big bank if you know stuff**

NO LONGER MARGINALIZED

Big players are moving into the field.

libra

facebook

Microsoft Azure
Blockchain as a Service

HYPERLEDGER

THE LINUX FOUNDATION

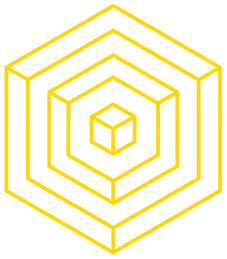
Amazon Managed Blockchain

IBM Blockchain

J.P.Morgan

Quorum

a16zcrypto



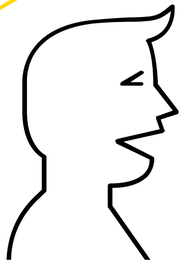
WHAT YOU CAN DO?

You can impact industries that you previously you could not enter

- **For example, DeFi - Decentralized Finance**
  - An industry previously hard to break into without enough capital and governance power
  - Use cases built by distributed ledger technology
    - Equalized financial access, financial inclusions
    - Stable currencies (stablecoin), e.g. Maker Dao, Facebook Libra, Celo
    - Peer-2-Peer transaction and dispute settlement platforms
    - Secure tamper-proof digital record keeping system

WHAT CAN YOU DO WITH IT?

DEMO: B@B HARMONIO PROJECT



Buyer

Buyer can challenge seller if didn't receive product

1 Transaction without mediator: Direct Transaction from Buyer to Seller

2 Transact with mediator + fee

Buyer's Insurance via escrow smart contract

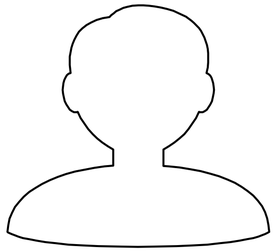
Seller

Seller can challenge buyer's claim by uploading proof of delivery of product

Stability
Using Celo Dollar as stable currency for transaction

Accessible Affordable
No existing banking relationship required
Circumvent fees of traditional payment systems

Mediator
Determine the winner of the challenges.
Selected from a pool of trusted individuals



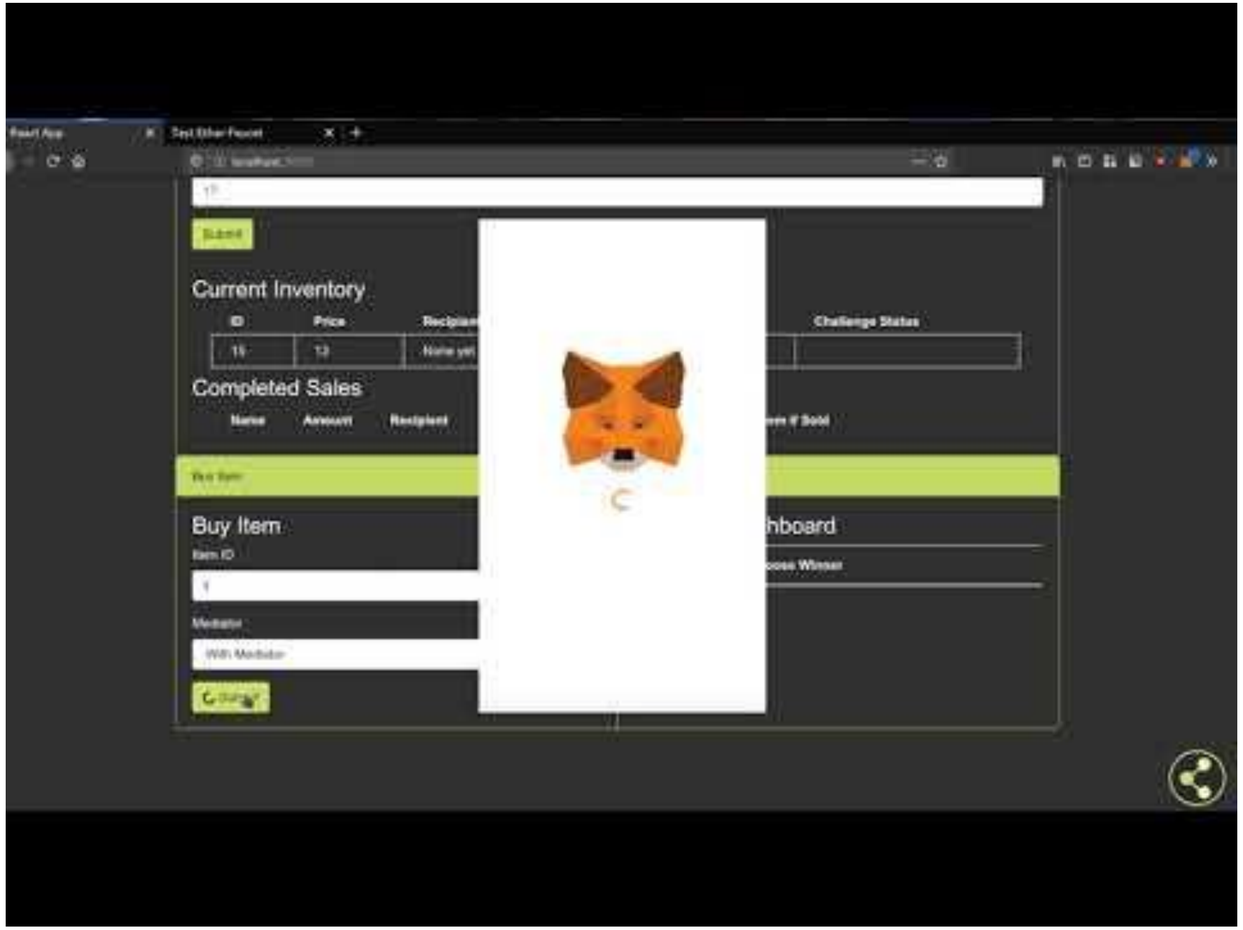
Frictionless
Direct P2P transfer between buyers and sellers

Bookkeeping
Digital record keeping for small businesses

DEMO: B@B HARMONIO PROJECT

BLOCKCHAIN AT BERKELEY x celo

https://youtu.be/g-Od7wGq2HE



AUTHOR: B@B CONSULTING DEPT.

2 COURSE DETAILS

THE GOAL

This course aims to teach students **the fundamentals of blockchains development**, **the Solidity programming language**, as well as **industry-relevant tools** such as Metamask, Infura, Truffle, and Ganache so that students will be equipped with industry-relevant experience in an accessible, collaborative environment.

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.**

LOGISTICS

Units: 1

Day and Times: Mondays 6:30PM – 8:30PM

Location: 240 Mulford Hall

Course Site:
https://blockchain.berkeley.edu/spring-2020-developers-decal/

Each class will be divided into two parts. The first hour of each class will be dedicated to a lecture covering the topics planned for that week. We will then switch gears in the second hour of class to focus on more hands-on work with that week's topics.

CONTACT

Course Administrator:

Minxing Chen

Facilitators:

Simon Zirui Guo, Janice Ng, Haena Lee, Grace Kull, Erika Badalyan

Contact:

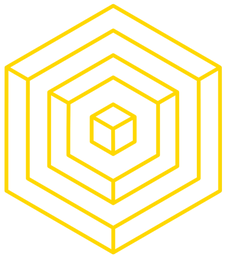
dev-decal@blockchain.berkeley.edu

SYLLABUS
WEEKLY SCHEDULE

- Week 1: Introduction to Blockchain Concepts & Development
- Week 2: Solidity and Developer Tools
- Week 3: Web3.js, Interacting with Ethereum
- Week 4: Token Development and Wallets
- Week 5: Smart Contract Security
- Week 6: Scalability, Interoperability, Zk-SNARKs

SYLLABUS
WEEKLY SCHEDULE

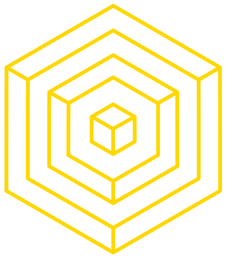
- Week 7-12: Special Topics and Guest Lectures in
  - Security and Attacks
  - Infra and Network Analysis
  - Internet of Blockchains
  - Emerging Protocols (Libra, Cosmos, etc.)
  - On-chain and off-chain Communication
  - Industry Experience Sharing



PREREQUISITES

This course requires backgrounds in Computer Science and Programming

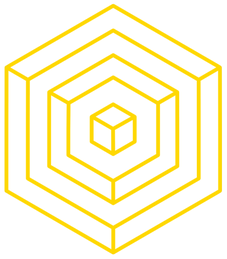
This course requires an understanding of fundamental Computer Science concepts as well as some programming abilities. You should have taken **CS61A** Structure and Interpretation of Computer Programs at least and it is recommended that you have also taken **CS61B** Data Structures, or have the equivalent skills. Bulk of the course will be focused on programming languages like **Solidity** (similar to JavaScript), **JavaScript**, and **Python**.



SUPPORT

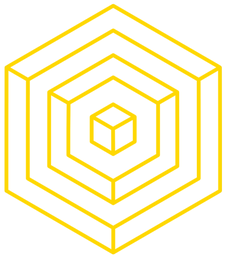
Ask for help!

- 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

Grading is **P/NP** for this course. In order to pass the course, students will need to achieve an overall grade of **above 70%**. Grading will be based on **Homework (50%)** and **Attendance (50%)**.



HOMEWORK (50%):

In blockchain development, you can only learn by doing.

- There will be homework dispersed throughout the course and will be weighted equally, each weighs 5%.
- Each lecture will be structured in a way that homework can be completed within the class. First half of each class covers the concepts, and the second half focuses on hands-on understanding through doing the homework.
- We will check-off in-person each week's homework at the end of the class, or at the latest, during the subsequent lecture. All homework check-off must be done **in-person**.
- Homework will be posted weekly on the course GitHub repository. https://github.com/BerkeleyBlockchain/Dev-DeCal-Spring-2020

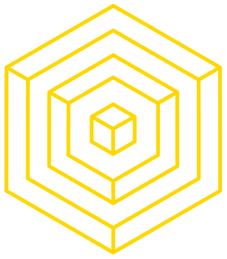
ATTENDANCE (50%):

- We will take attendance every class.
- We grant you 2 unexcused lecture absences without grade penalty.
- Having 3 unexcused absences will result in automatic failure of the course (10% deducted from overall score per absence).
- If you are expecting an academic conflict such as a midterm, please let your facilitator know at least 24 hours in advance, and we will work out an alternative assignment with you. Other emergencies can be excused if instructors are informed far enough in advance.
- We will do our best to be as understanding and accommodating as possible.

EXPECTATIONS
ADMINISTRIVIA

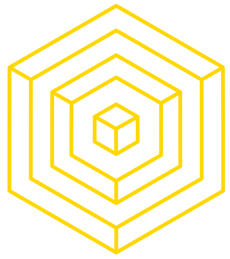
Expect from us:

- 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

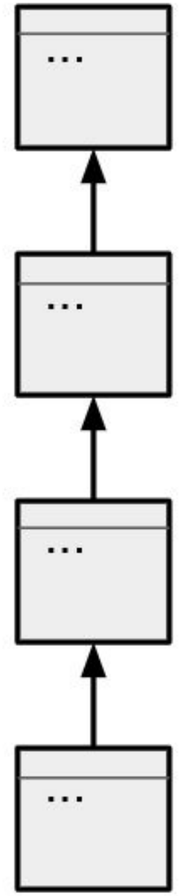
- **Cryptocurrency:** completely digital, formless currency that is tied together using computer science, and cryptography, and economics.

- **Blockchain:** data structure behind cryptocurrency. Method of storing data amongst multiple parties that ensures data integrity without requiring trust.

source:
https://yle.fi/uutiset/osasto/news/finance_ministry_crackdown_on_cryptocurrency_trade/10040789

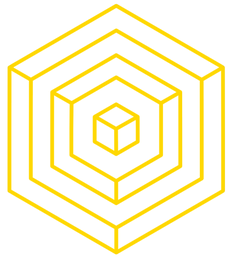
DEFINITIONS

QUICK REVIEW



Key Attributes of Blockchain:

- **Immutable:** secured with cryptographic hash functions
  - Data committed to the blockchain cannot be changed
- **Distributed ledger:** shared database where everyone in the ledger holds a copy
- **Decentralized:** no central authority, no central point of failure
- **Consensus:** Proof of Stake, Proof of Work, etc.



3.2 ETHEREUM OVERVIEW

WHAT IS ETHEREUM?

HIGH LEVEL OVERVIEW

- Ethereum is a **decentralized blockchain** platform designed to run **smart contracts**
  - Like a distributed computer to execute code
  - **Distributed state machine - transactions change global state**
    - transactions == state transaction function
- Ethereum has a native asset called ether

AUTHOR: TIM HUANG

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

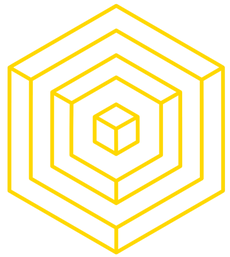
AUGUR

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**

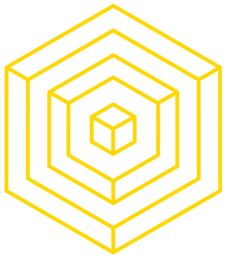
AUTHOR: TIM HUANG

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

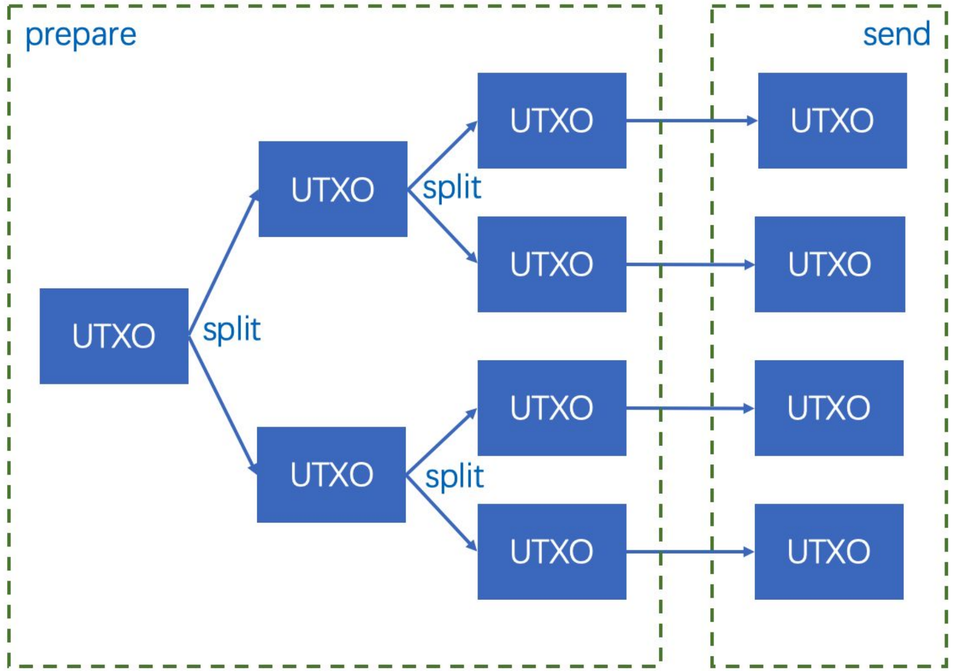
What is stored in account state?

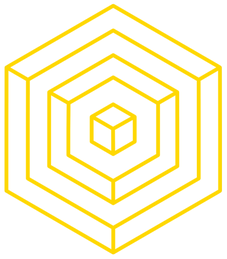
- **nonce**: the number of transactions that are sent from that account, number of contracts created if contract account
- **balance**: The number of Wei owned by this address. 10^18 Wei = 1 Ether.
- **storageRoot**: a hash of the identifier to the storage of the contract.
- **codeHash**: the hash of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

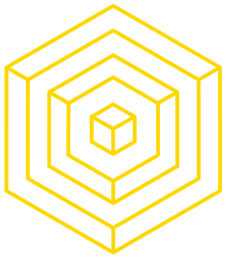
GAS AND PAYMENT

GAS EXPLAINED

What is gas?

- Fee incurred for every computation that occurs as a result of a transaction
- Paid by sender of transaction
- Ensures network isn’t misused

AUTHOR: TIM HUANG

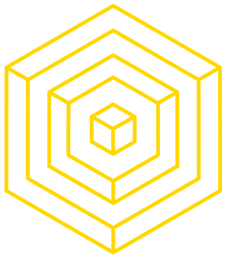


GAS AND PAYMENT

WHY DO WE NEED FEES?

- **Parallel, Distributed Computation**
  - Any transaction on the EVM requires large amounts of computation
  - Smart contracts best used for simple tasks

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



GAS AND PAYMENT

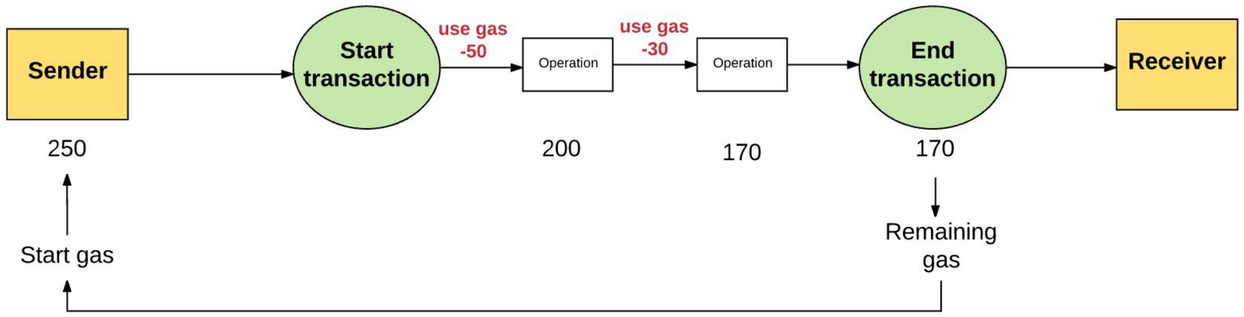
GAS LIMIT AND GAS PRICE

- **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
- **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

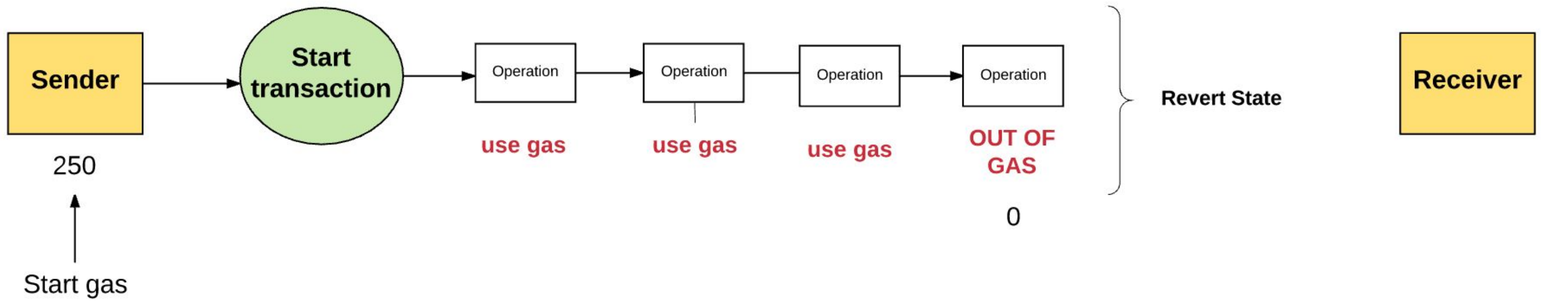


GAS AND PAYMENT

WHAT IF NOT ENOUGH IS SENT?

Not enough gas to execute the transaction?

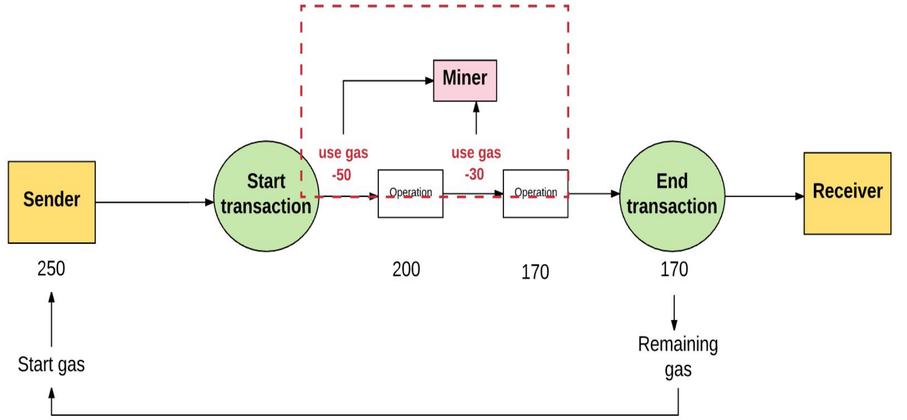
- Transaction runs out of gas and therefore is considered invalid
- State changes are reversed, failing transaction recorded
- Since computation was already expended by the network, none of the gas is refunded



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

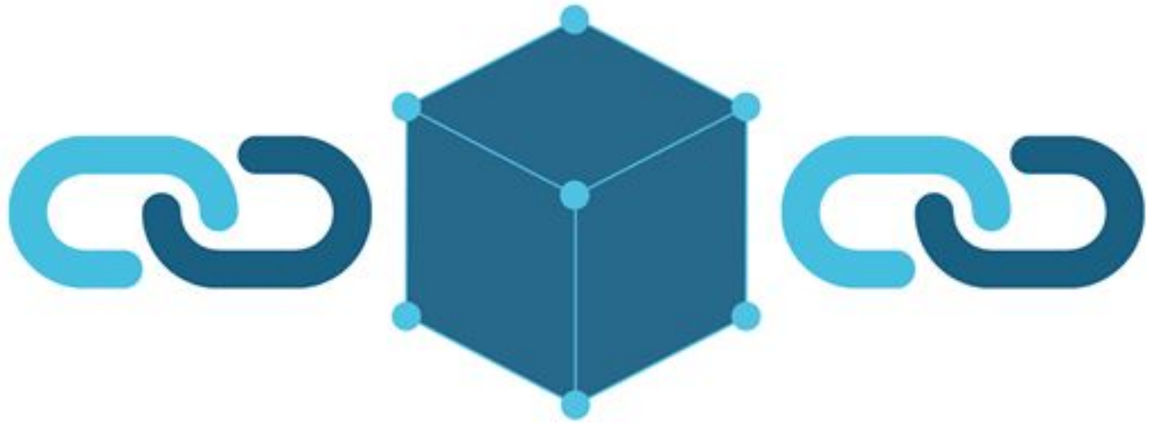


AUTHOR: TIM HUANG

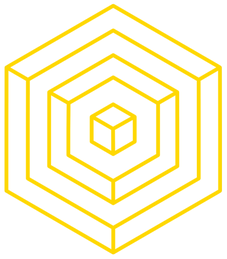
GAS AND PAYMENT

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

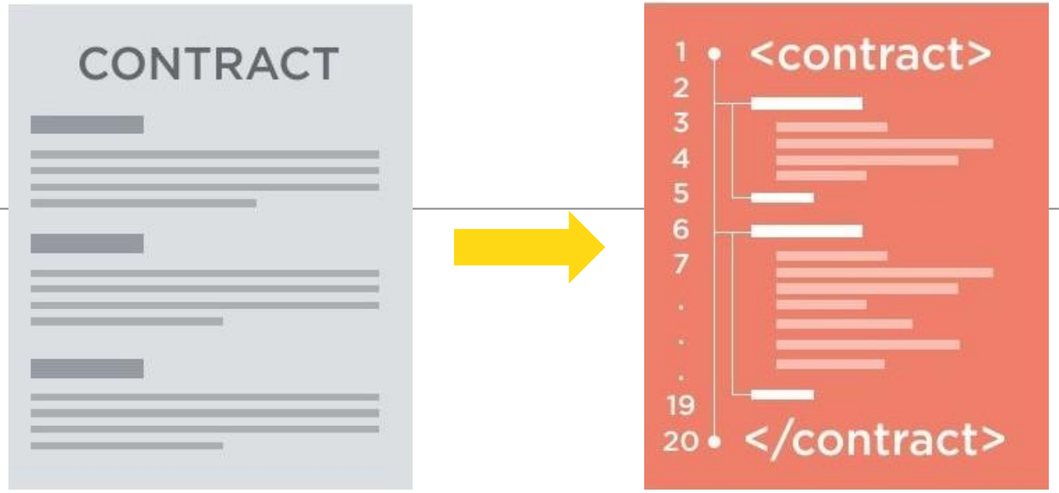
(noun) /ˈkäntrakt/

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/

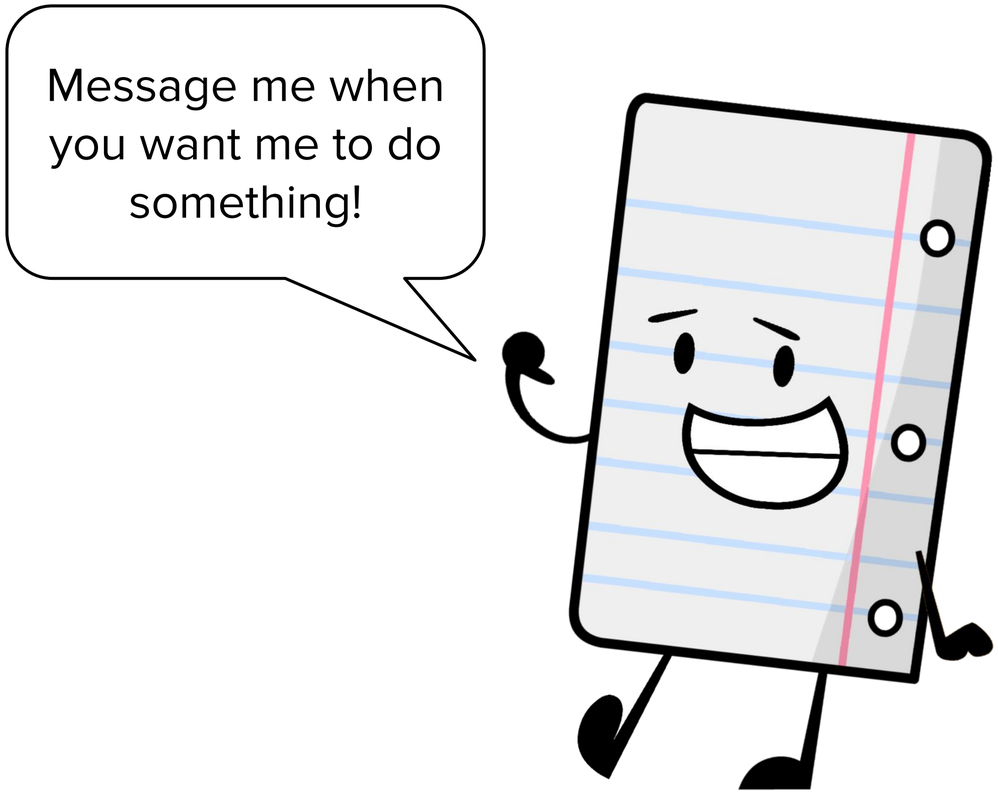
1. code that **facilitates**, **verifies**, or **enforces** the negotiation or execution of a digital contract.

AUTHOR: PHILIP HAYES

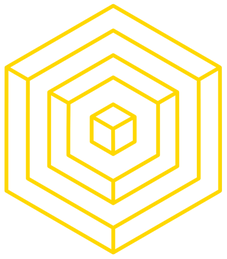
ETHEREUM SMART CONTRACTS
IN CONTEXT

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

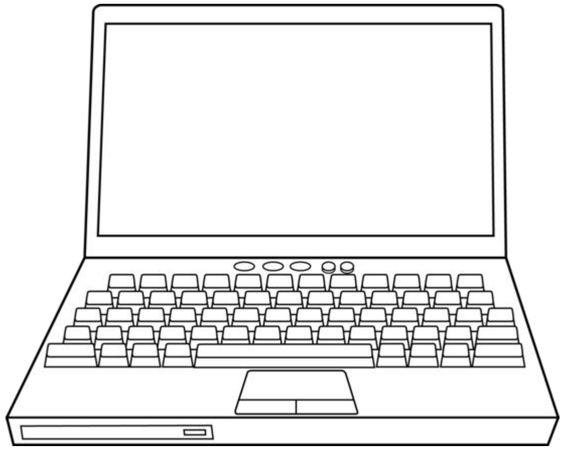
SAMPLE BETTING CONTRACT

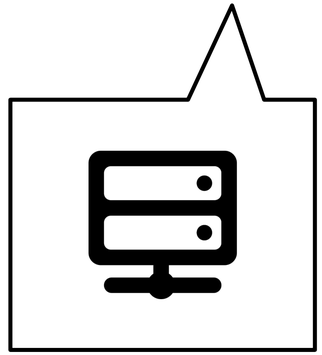
```
contract Betting {
     address public owner;
     address public gamblerA, gamblerB, oracle;
     uint[] outcomes;
     struct Bet {
         uint outcome;                          /* Defines a Bet */
         uint amount;
         bool initialized;
     }

     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

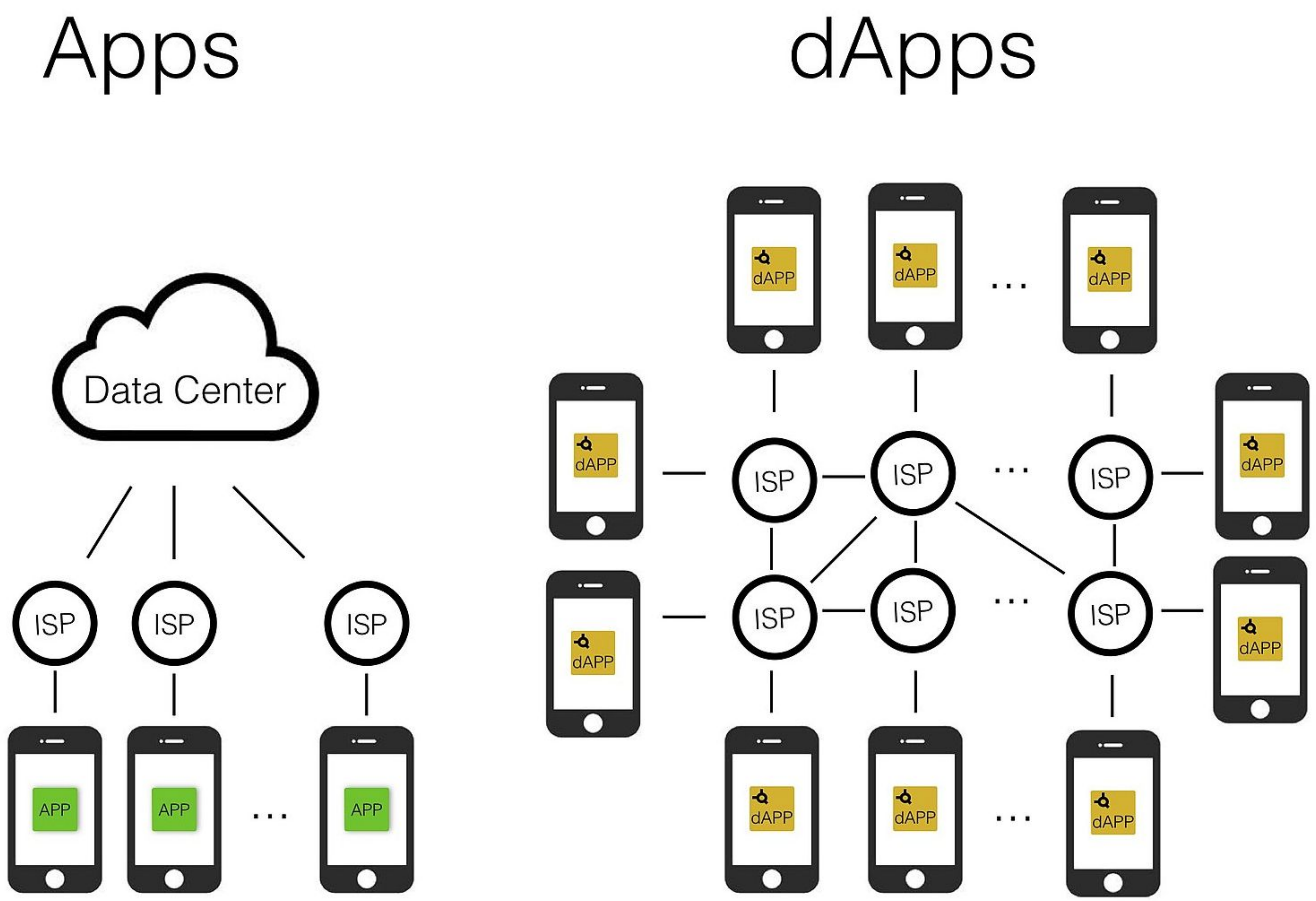




AUTHOR: PHILIP HAYES

DAPPS
HIGH-LEVEL OVERVIEW

- Implemented via smart contracts
- Peer-to-Peer
  - No Central Server
- Immutable
- Open source code = transparency
  - Less phishy activities
- Similar to traditional applications but decentralized

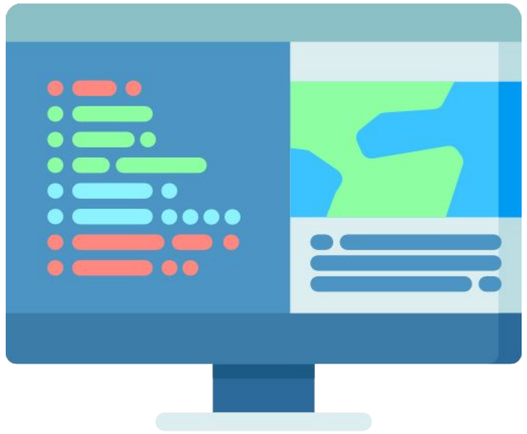


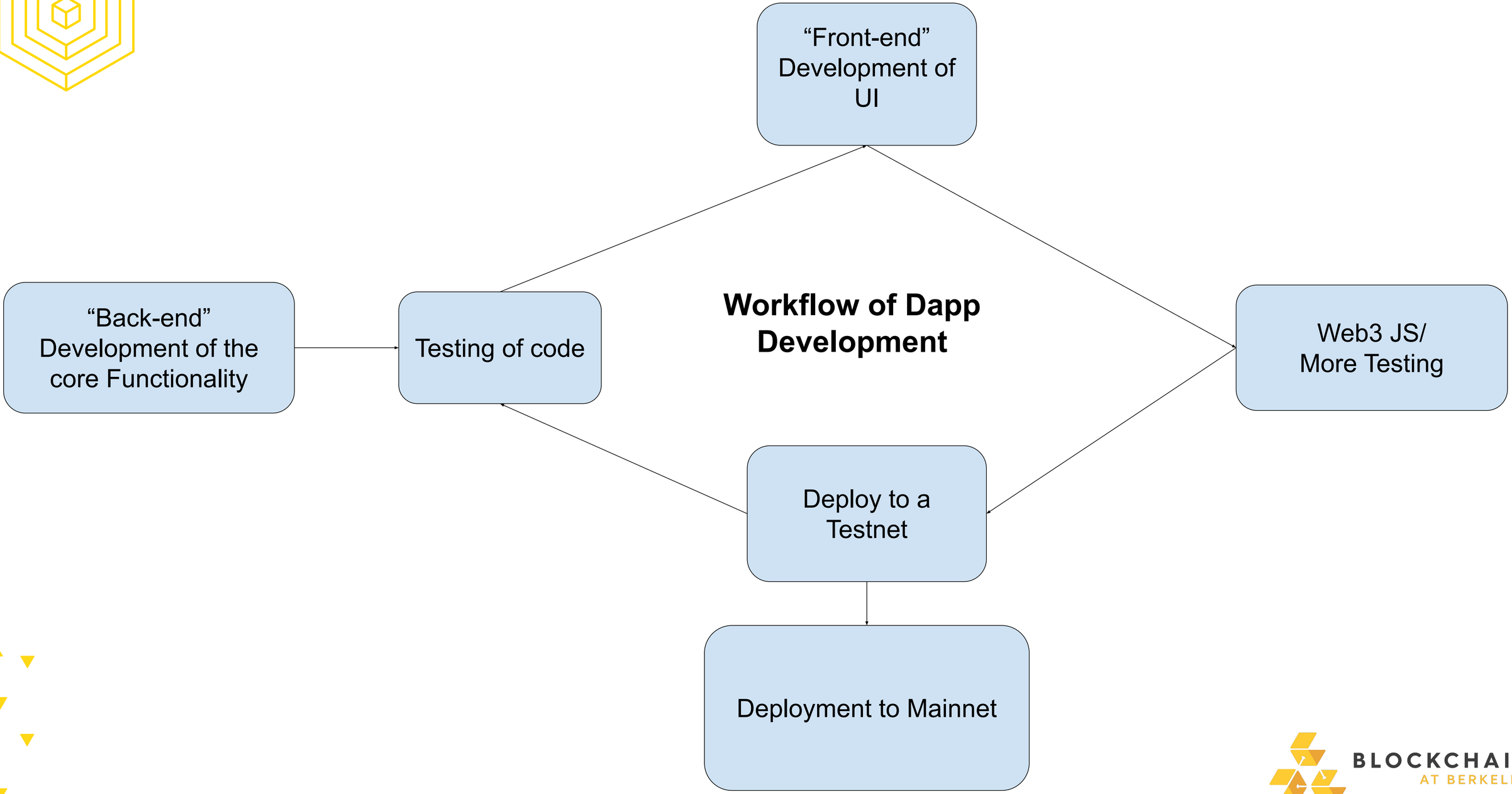
Source: https://towardsdatascience.com/what-is-a-dapp-a455ac5f7def

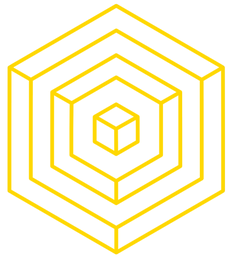
WORKFLOW

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
- More Information on the current state of Dapp Development: https://thecontrol.co/a-brief-overview-of-dapp-development-b8ac1648322c

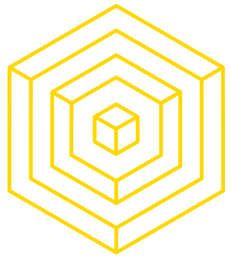



“Front-end”
Development of
UI
“Back-end”
Development of the
core Functionality
Testing of code
Workflow of Dapp
Development
Web3 JS/
More Testing
Deploy to a
Testnet
Deployment to Mainnet



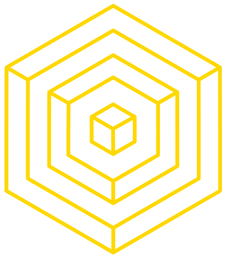
Attendance

https://tinyurl.com/sp20-dev-decal-1



5 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:

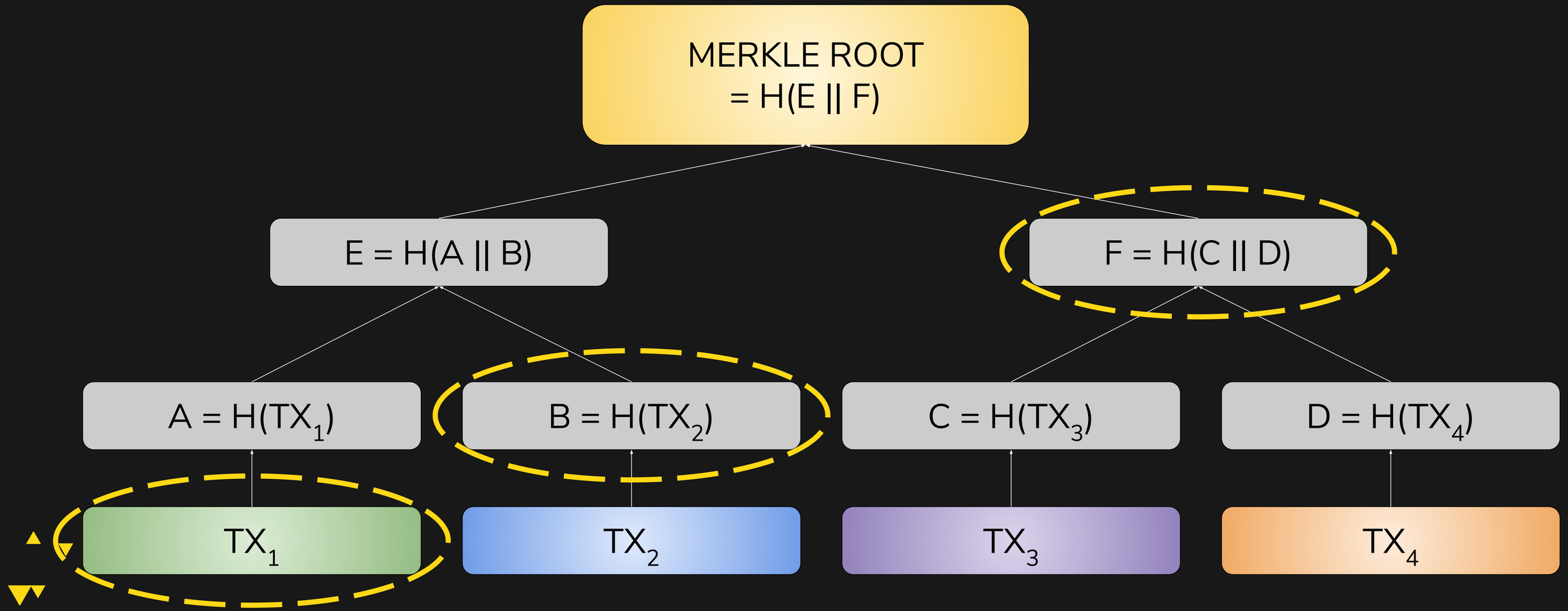
A hash function with three special properties:

- Preimage resistance
- Second preimage resistance
- Collision resistance

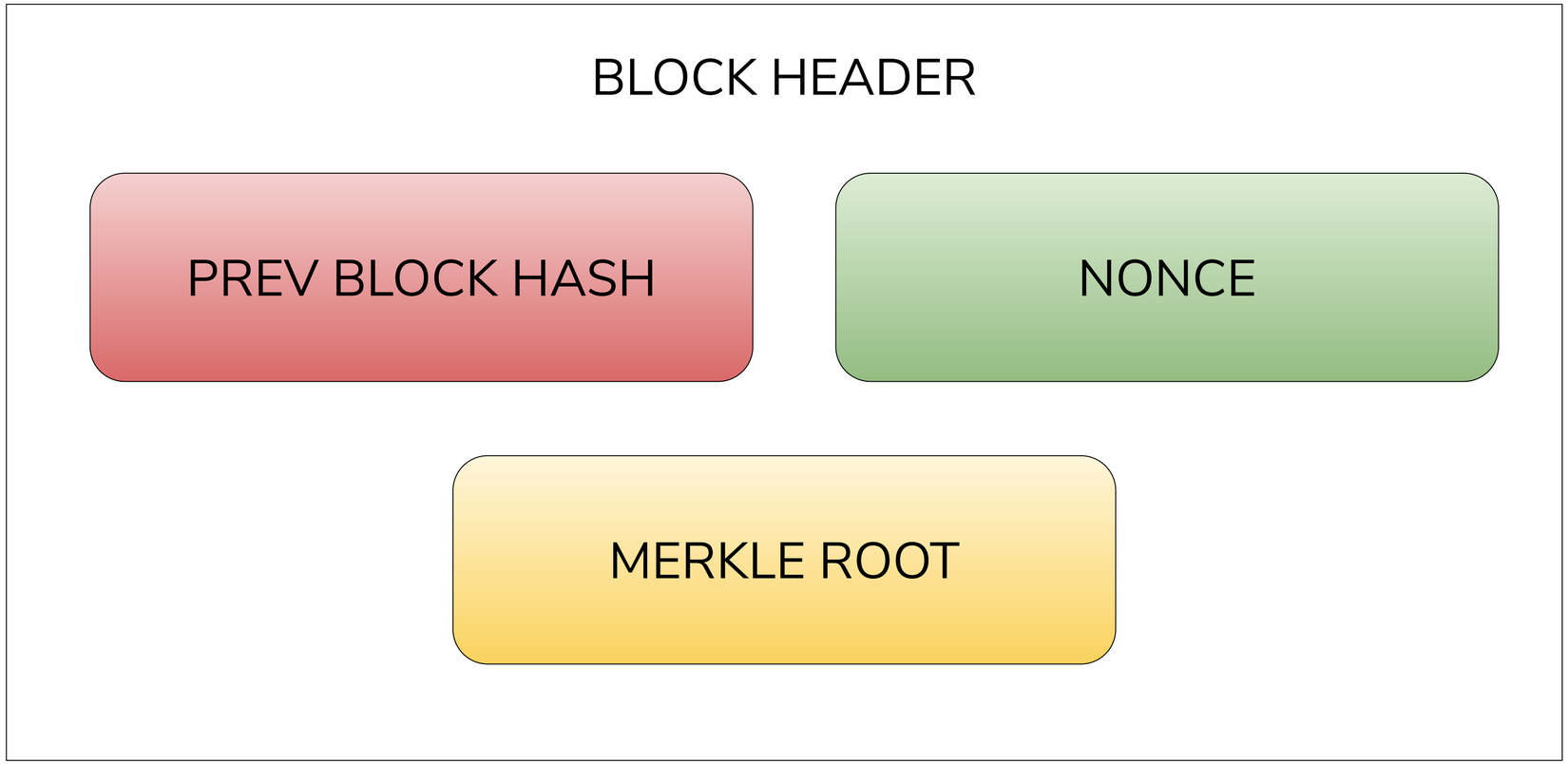
```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
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...
```

A TAMPER-EVIDENT DATABASE

MERKLE BRANCH & PROOF OF INCLUSION

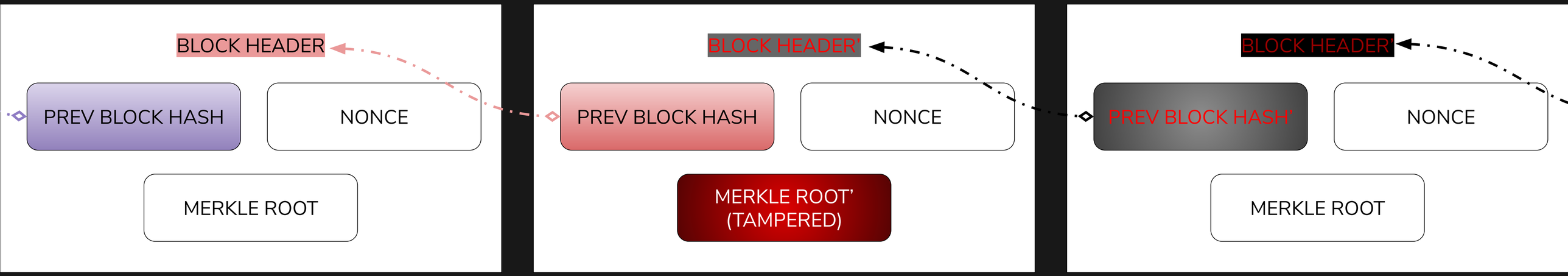


COMPONENTS OF A BLOCK



A TAMPER-EVIDENT DATABASE

PROTECTING THE CHAIN



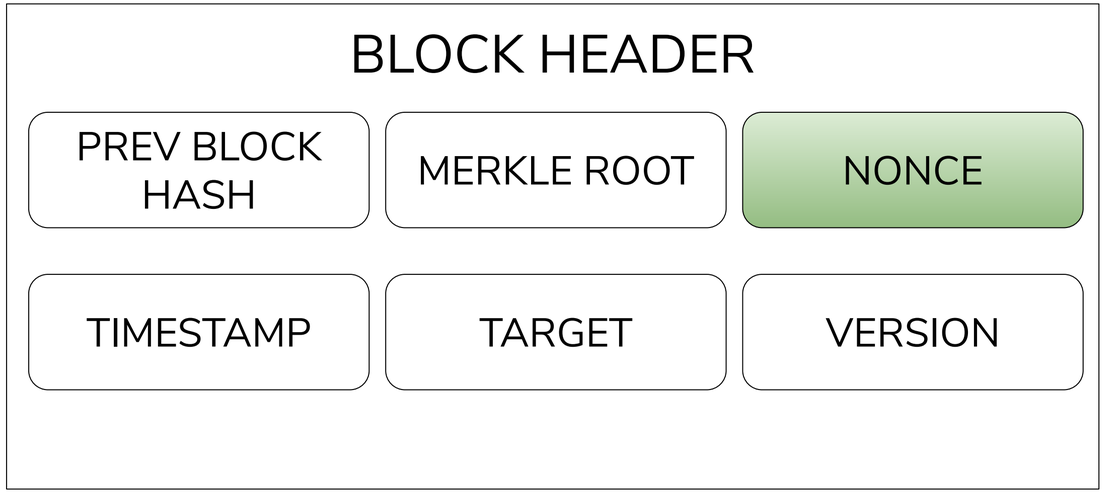
SHA256(SHA256(x))

```
prevBlockHash = 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)`

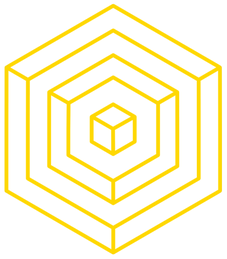
`H("Hello, world!4250")`

`0x0000c3af42fc31103f1fdc0151fa747ff87349a4714df7cc52ea464e12dcd4e9`

<

`0x0000ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff`

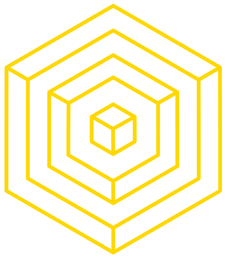
Solved!



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

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