Blockchain Application Sharing

CUHK FTEC 5520 – 9Mar, 2023

Background

DIGIGEAR TECHNOLOGY LIMITED

Founded in 2021, DigiGear is a Blockchain company specializing in NFT, digital assets and Web3. Aside from its proprietary product TicketBear, DigiGear strives to provide private and public Blockchain-oriented

(1) Infrastructure building services;
(2) Consultancy serivces

Google Cloud
HashiCorp Terraform
aws
Microsoft Azure
corda
HYPERLEDGER FABRIC

TICKET BEAR

Started off as a passion project, TickerBear is a proprietary product of DigiGear's, TicketBear aims to tackle painpoints of

(1) Traditional ticketing;
(2) Decentralisation of communities on a B2C scale

HKSTP Incubatee (Batch of 2022)

HKSTP 香港科技園

Our company website

- https://digigear.io/

DIGIGEAR
TECHNOLOGY LIMITED

Agenda

- Section 1 - The blockchain landscape
- Section 2 - Public blockchain sharing
- Section 3 - Enterprise blockchain sharing

The blockchain landscape

- Top blockchains
- Solving blockchain scalability
- Cross-chain communication
- Ecosystem development
- Application trends

Public blockchain sharing

- Look at DeFi
  - Swaps & Liquidity Providers
  - GameFi
- Launching NFT collectibles on ETH
  - ERC-721 / ERC-1155
  - User minting
  - Secondary market on NFT marketplaces
- Public Blockchain & Ethereum high level overview
  - Blockchain
  - Ethereum overview
  - World state (UTXOS VS ACCOUNTS)
  - Wallet
  - GAS
  - Smart contract
- DeFi Hacks and defenses

Enterprise Blockchain

- Introduction to enterprise blockchain
- Enterprise applications
- Central bank digital currency (CBDC)

1. The blockchain landscape

Top blockchains

Bitcoin – “digital gold”

Ethereum – most established chain for smart contracts

Solving blockchain scalability

- Problem 1: Low transaction throughput
  - Bitcoin: ~5 tps
  - Ethereum: ~20 tps
  - Throughput = txs per block / block time
- Problem 2: Long confirmation times
  - Bitcoins: 30 mins?
  - Ethereum: 2 min?
  - Confirmation time = confirmation blocks required * block time

Solving blockchain scalability

- Layer-2 solutions
  - Bitcoin Lightning (Bitcoin)
  - Polygon (Ethereum)
- Move to PoS / DPoS
  - **Proof of stake**: Each validator must stake a minimum number of tokens to participate in consensus, earns staking reward
  - **Delegated proof of stake**: User may delegate stake to a chosen validator; validator earns commission on staking reward
    - Eth 2.0 -> DPoS
  - **Slashing**: Staked tokens may be slashed if validator is found to misbehave

Solving blockchain scalability

- High-throughput, low latency blockchains
  - Avalanche (Avalanche consensus protocol)
  - Solana (Proof of history)
  - Binance Smart Chain (Centralized)
- “Layer-0” application-specific blockchains
  - Powered by Cosmos SDK and Inter-blockchain (IBC) protocol

Cross-chain communication

- Inter-blockchain (IBC) protocol
  - Interface for cross-chain communication channels
  - Channel: send, receive, ack, timeout
  - High-level features can be implemented on top of the channel
  - Requires a trusted relayer between the two chains to operate
  - Implemented for Cosmos, ETH

Cross-chain communication

- Token bridges
  - Example: Gravity Bridge (Cosmos), Wormhole Bridge (Solana), Avalanche Bridge (AB)
  - “Sends” tokens from one chain (*origin*) to another (*target*)
  - Token in origin chain is locked in the bridge contract
  - Token in target chain is issued by the bridge contract
  - Token becomes “wrapped”; Wrapped tokens are 1-to-1 backed and can be redeemed through the bridge

Ecosystem development

- Active development of new blockchain ecosystems
  - *Move fast and get hacked!*
- Many blockchains / protocols launching ecosystem funds to grow their ecosystems
- Ecosystem effect

Ecosystem development

- DeFi:
  - Dapp demand -> Token demand -> DeFi market demand
  - Funds raised from IEO/IDO/NFT sale -> Fund management
  - Demand for crypto investment -> Derivative products
- NFT:
  - Art & collectibles
  - Community development
  - Games

Application trends

- Decentralized protocols
  - On-chain technology + Token economy to remove middlemen intermediaries (replaced with middleware protocols)
- Decentralized governance
  - Decentralized autonomous organization (DAO)
  - Autonomous -> Operation of smart contracts
  - Non-autonomous -> Participation in governance by token holders through community engagement, proposals and voting
  - Example: MakerDAO (Algorithmic stablecoin DAI)

Regulatory trends

- Increased scrutiny of
  - Centralized exchanges
  - Stablecoins
- Classification of crypto-assets
  - Cryptocurrency: Security token or not?
- Ongoing concerns on AML & CTF

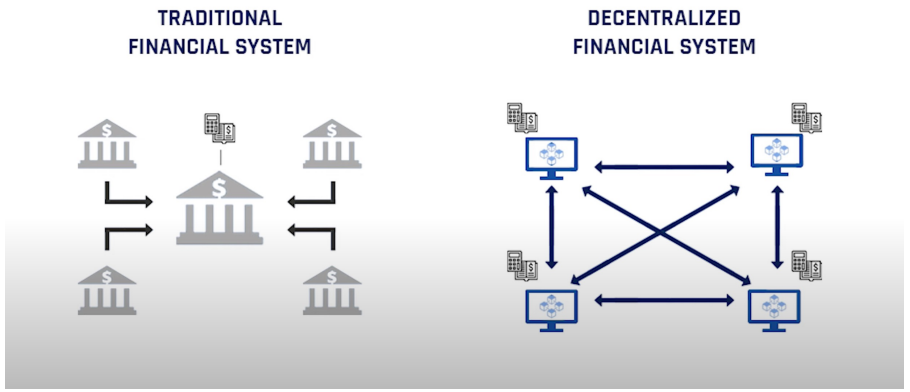
2. Public Blockchain & Ethereum high level overview

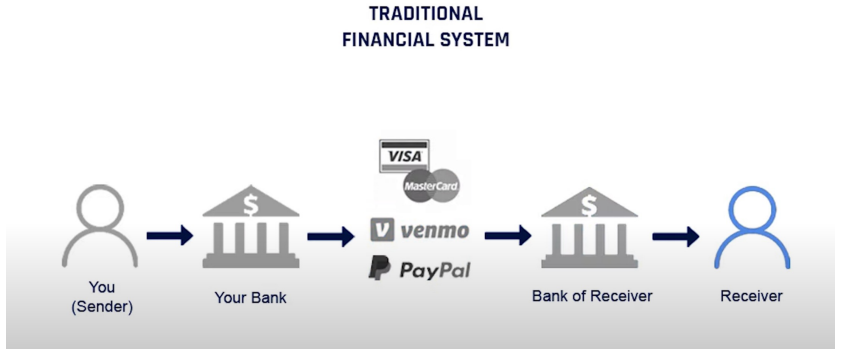
Some slides are borrowed form
Blockchain@Berkeley's Blockchain for Developers DeCal

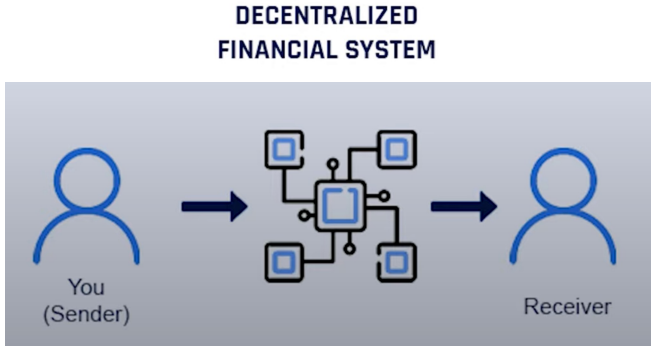
2.1 De-Fi

Some slides are borrowed form Blockchain@Berkeley's Blockchain for Developers DeCal by Daniel Pyrathon

A brief overview of DeFi
Financial services on blockchain



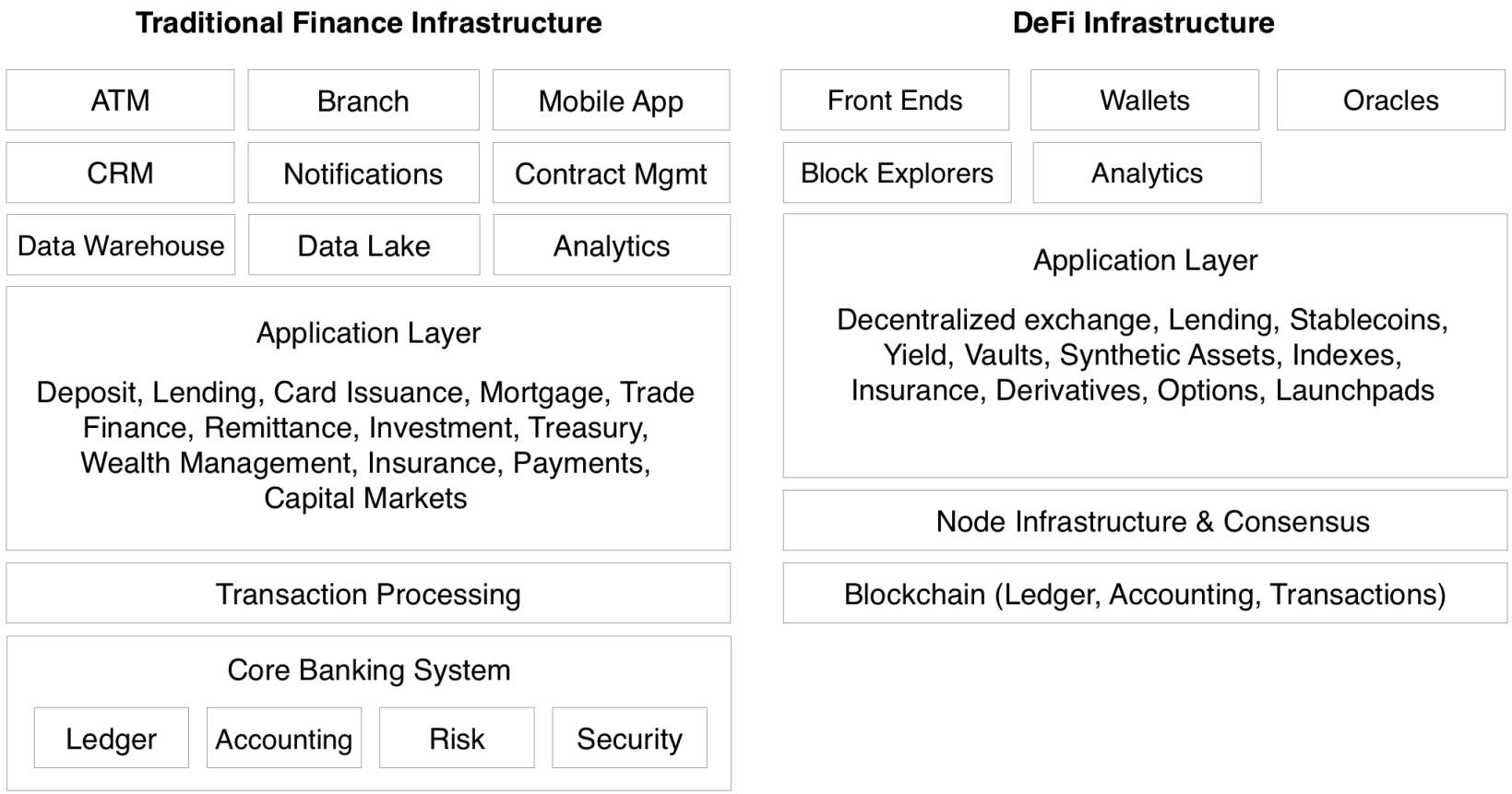




Source: https://www.youtube.com/watch?v=1Nl6bTrc3gY

A brief overview of DeFi

DeFi Infrastructure



Composability of DeFi apps

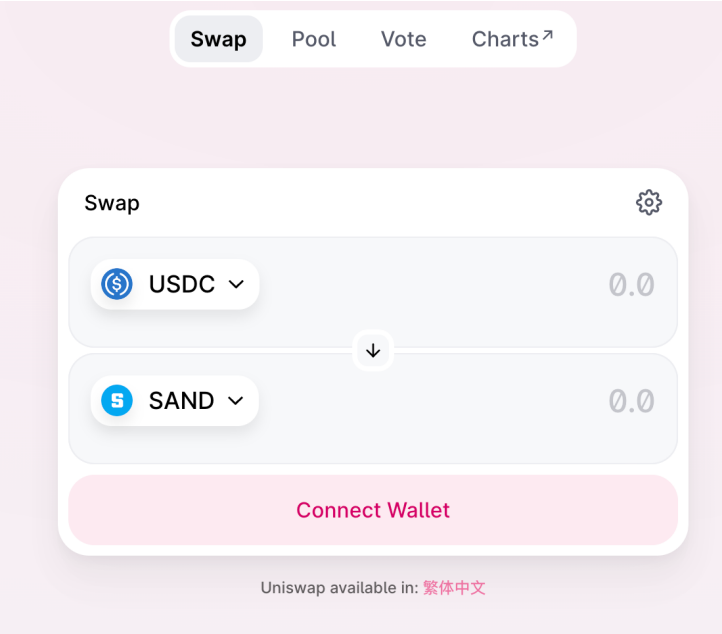
- DeFi applications are more composable relying on the base layer of the Blockchain. Within one single ETH transaction, you can interact atomically with many applications.
  - Build on the Progress of Others
  - Build for interoperability
  - Think of “Money Legos”

Notable DeFi products

- Stable coins / Stable ERC20 Tokens

Notable DeFi products

- Decentralized Exchanges (DEXs)



Uniswap

Oasis

PancakeSwap

kyber network

DIGIGEAR TECHNOLOGY LIMITED

Decentralized Exchange Trading ERC-20 Tokens

Traditional Exchange

- Custodial: Risk of hacks
- Risk of downtime
- Location-based access
- Must pay to list your token

Decentralized Exchanges

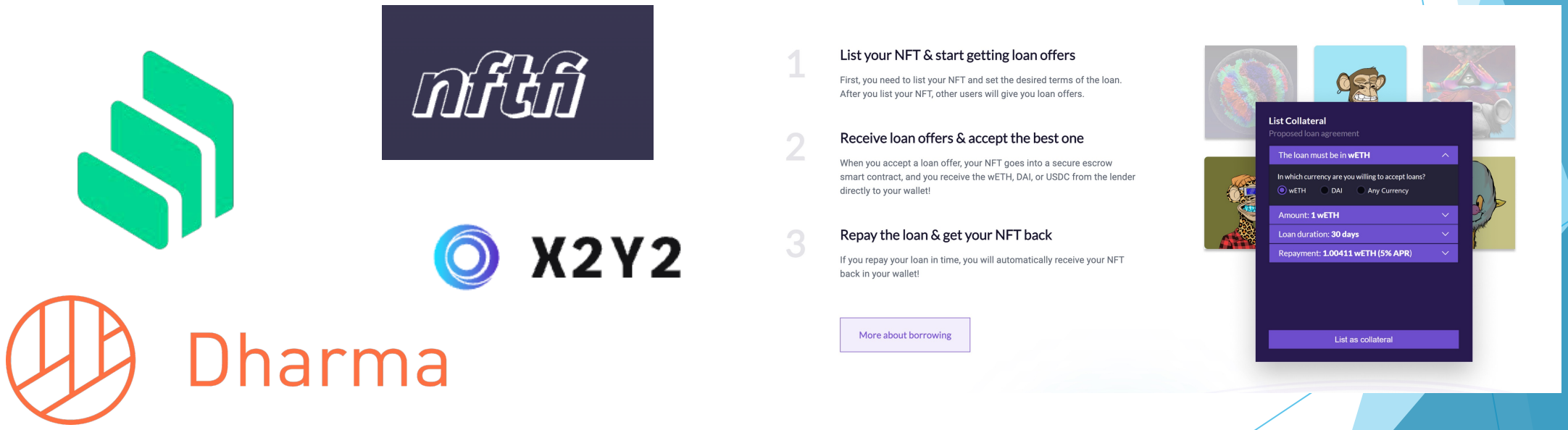
- Transact peer-to-peer through open source smart contracts
- Self-custody
- Open globally
- Composability and interoperability
- All token supported, enabling the discovery of new markets

Users hold custody of their own funds

- User don’t push their funds into a centralized financial service, instead they give a smart contract an “allowance” to pull funds out

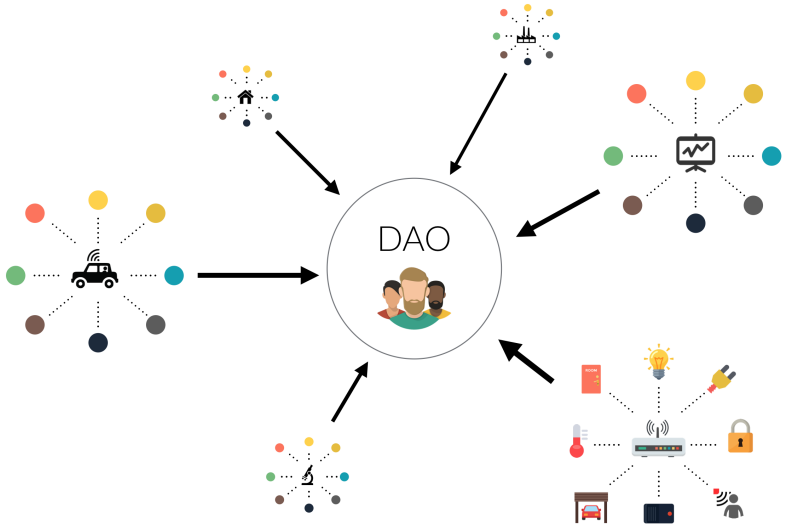
Notable DeFi products

- Landing and borrowing



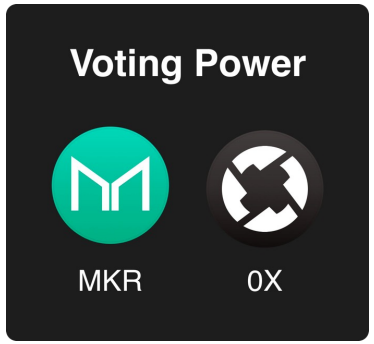
Governance of financial services

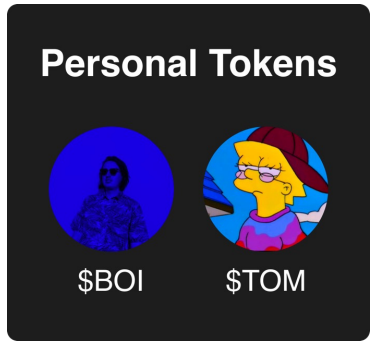
- DeFi apps create voting mechanisms (e.g.DAO) that ensure accountability to users, allowing them to suggest, debate, and drive the direction of the product



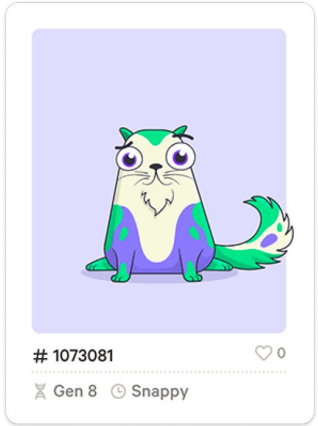
ALL forms of value will be eventually represented as tokens

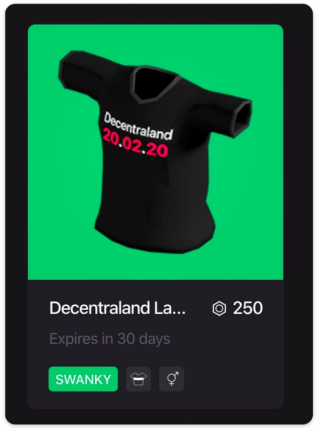
- ERC-20

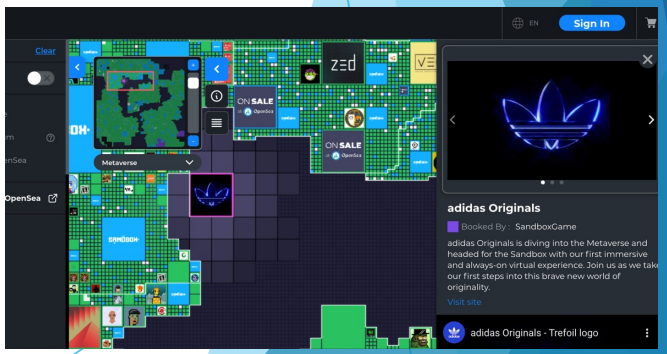




- NFT
  ERC-721

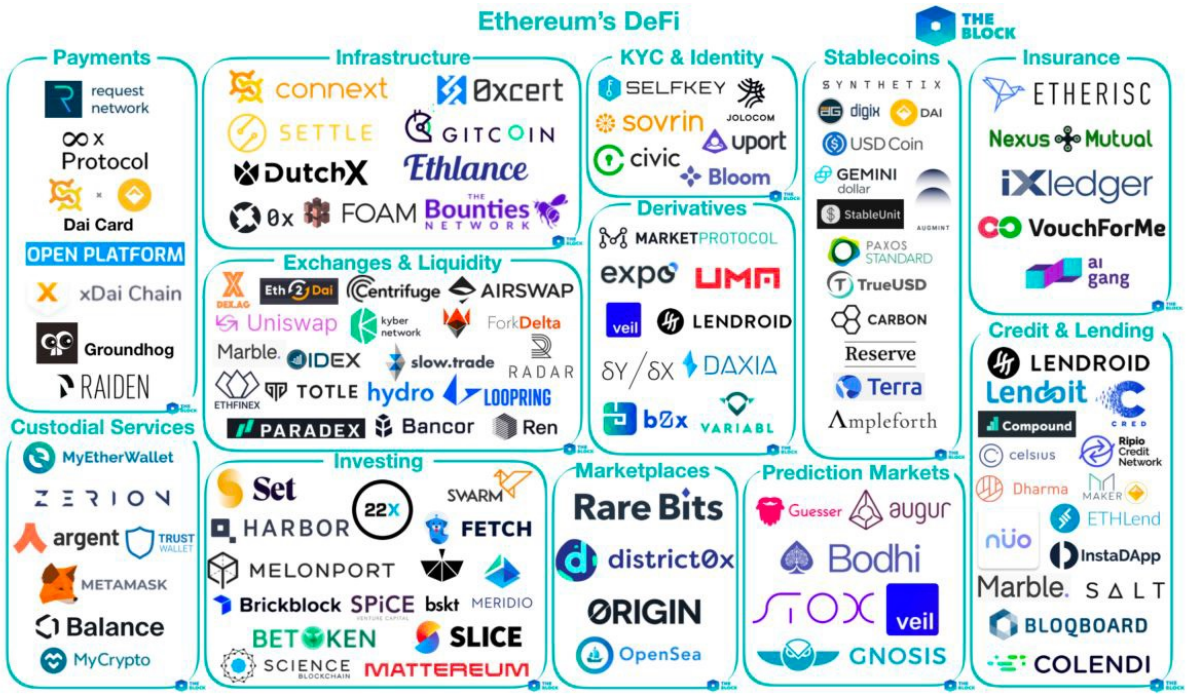






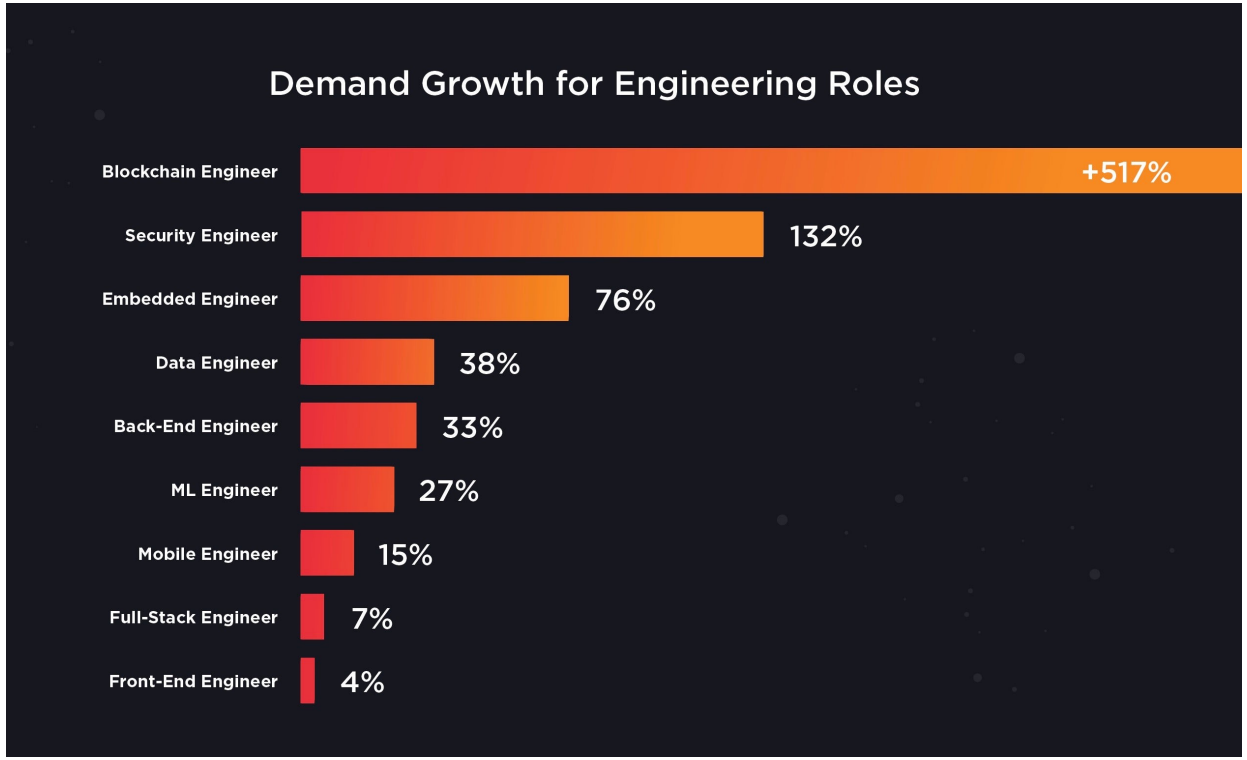
Notable DeFi products

- And more…



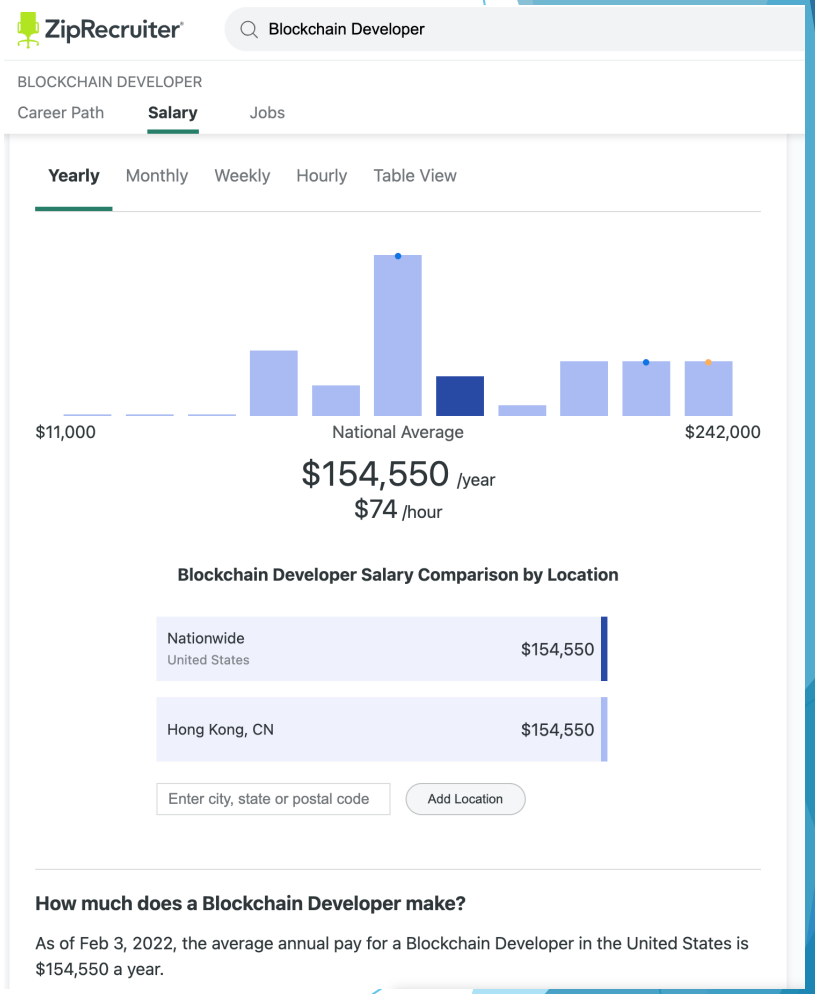
Value locked in DeFi

Source: https://defillama.com/



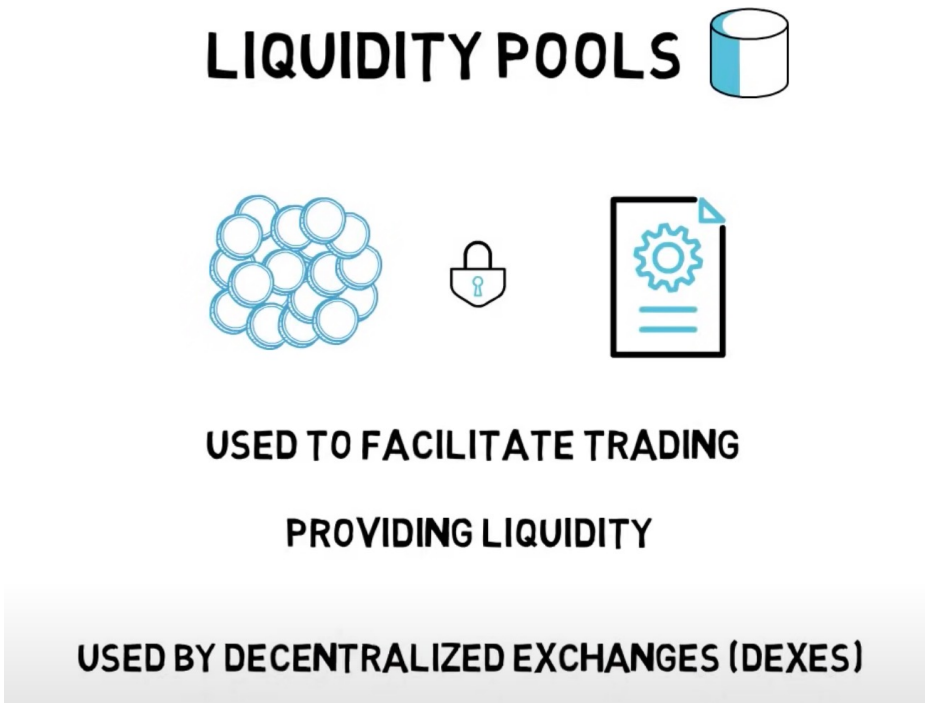
FEB 28, 2019

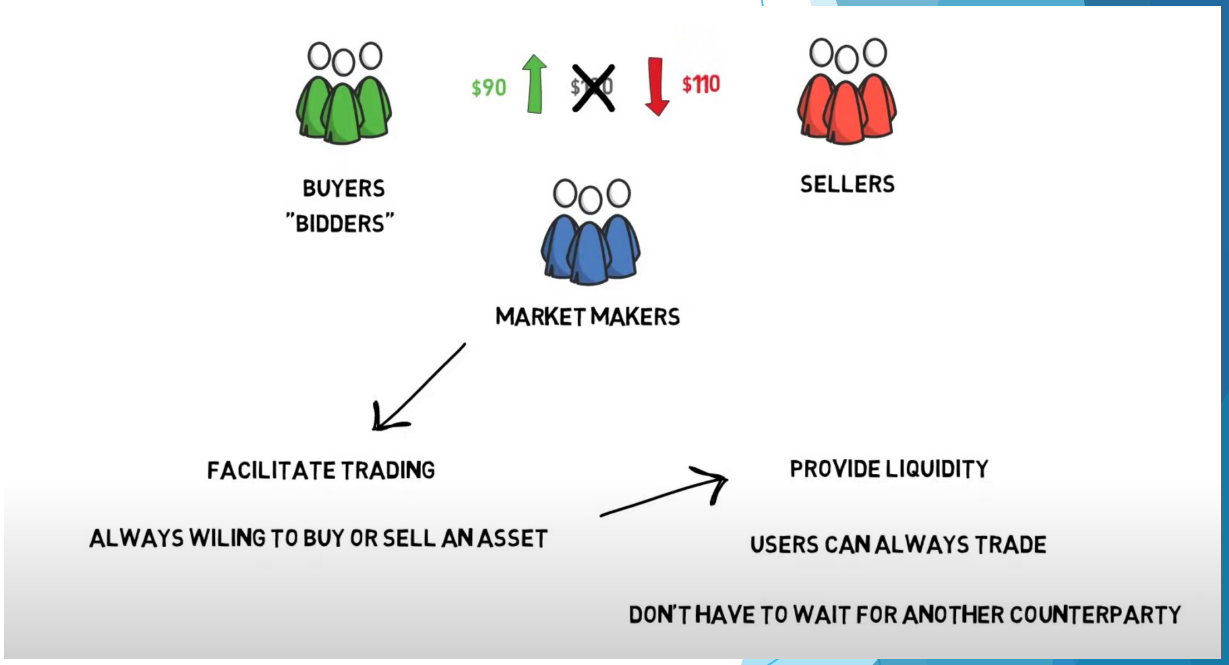
Source: https://www.computerworld.com/article/3345998/demand-for-blockchain-engineers-is-through-the-roof.html



Let's Build a DeFi Application

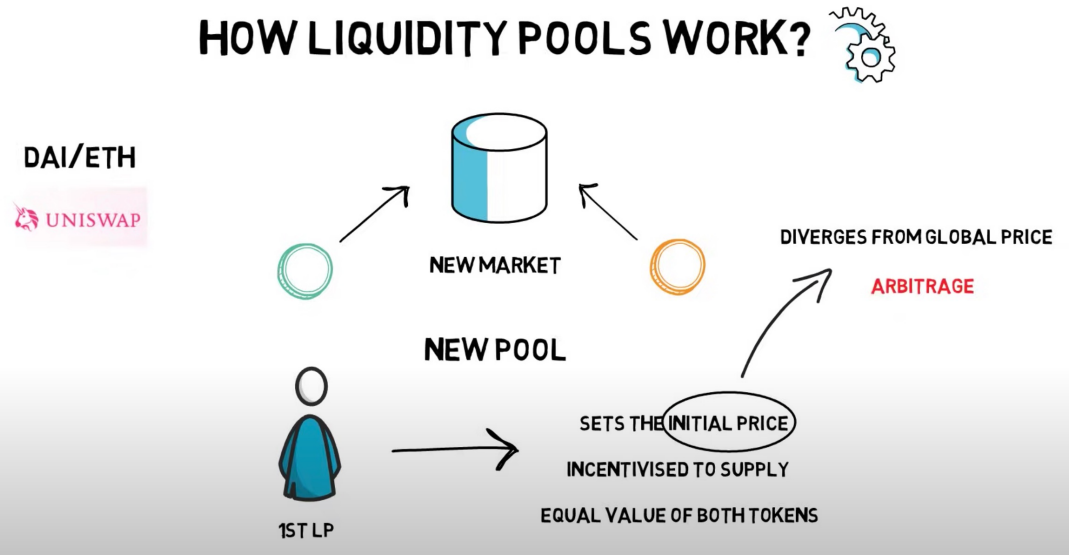
Launching Swaps & Liquidity Providers
What Is Liquidity Pool

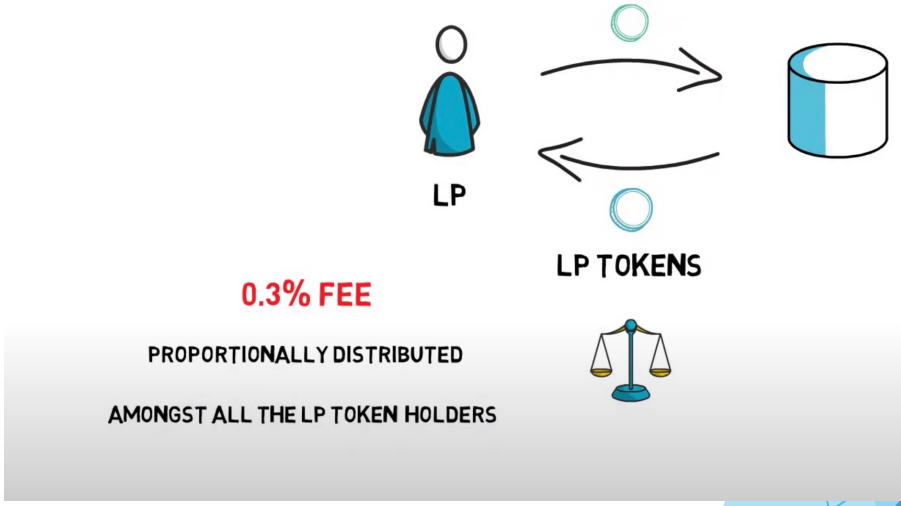




Source: https://www.youtube.com/watch?v=cizLhxSKrAc

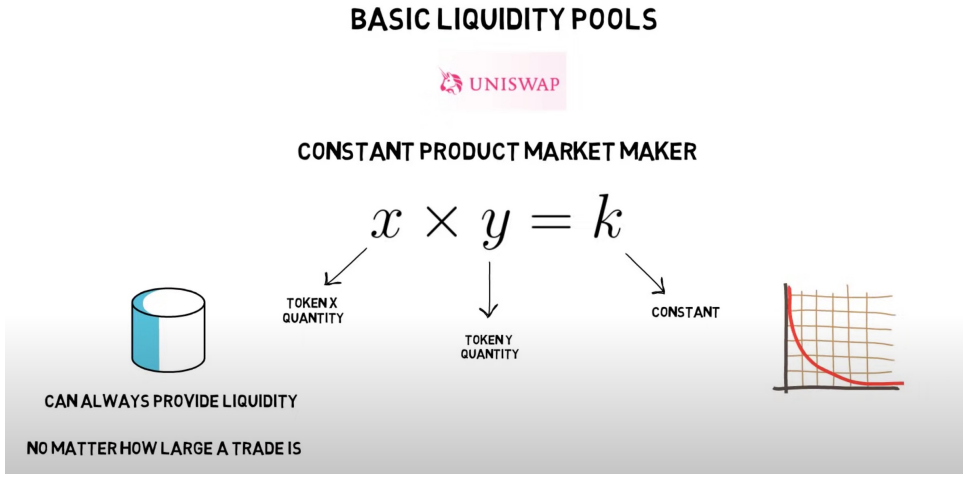
Launching Swaps & Liquidity Providers
How Is Liquidity Pool Works

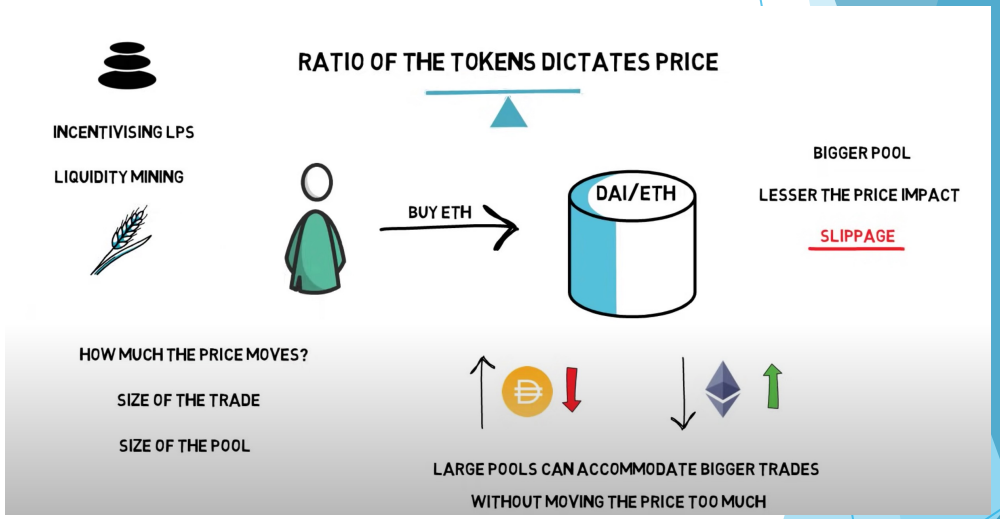




DIGIGEAR TECHNOLOGY LIMITED

Launching Swaps & Liquidity Providers
Liquidity mining





Launching Swaps & Liquidity Providers Liquidity Pool Implementation example with Uniswap

- Issuing your own token
  - Create ERC20 contract
- Create Liquidity Pool
  - Select token pair to swap
  - Create Uniswap v2 contract
  - Call addLiquidity function with the token pair.

Issuing your own token
Create ERC20 contract

```
// contracts/GLDToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract GLDToken is ERC20 {
    constructor(uint256 initialSupply) public ERC20("Gold", "GLD") {
        _mint(msg.sender, initialSupply);
    }
}
```

Source: https://docs.openzeppelin.com/contracts/3.x/erc20

Create Liquidity Pool
Create Uniswap v2 contract

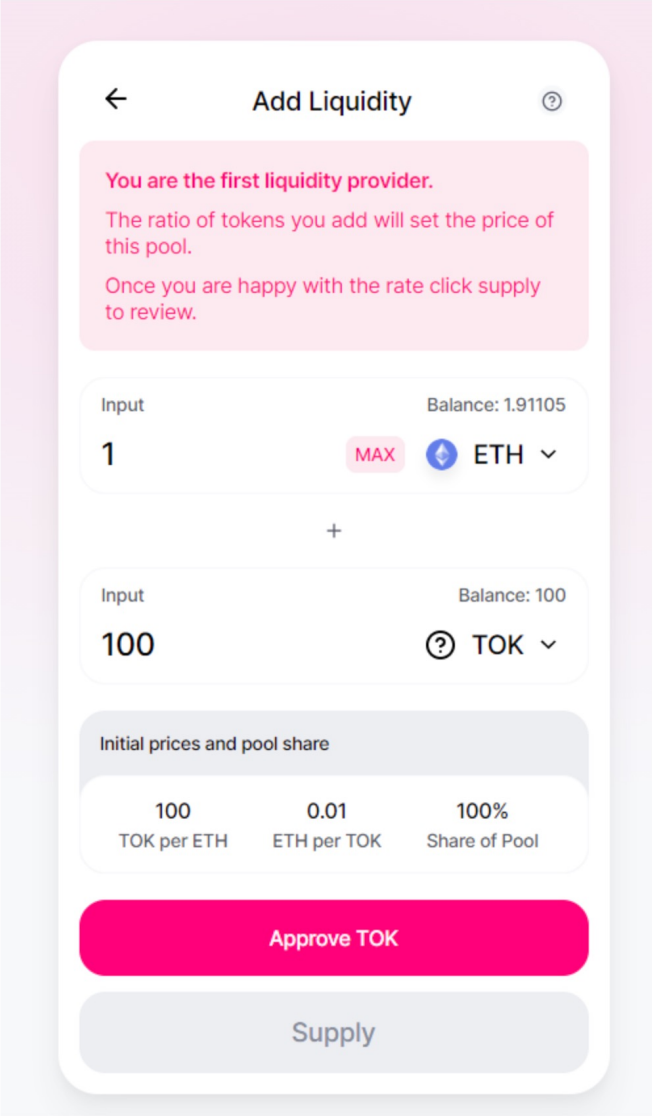
201 lines (176 sloc) | 9.56 KB — Raw | Blame

```solidity
pragma solidity =0.5.16;

import './interfaces/IUniswapV2Pair.sol';
import './UniswapV2ERC20.sol';
import './libraries/Math.sol';
import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/IUniswapV2Factory.sol';
import './interfaces/IUniswapV2Callee.sol';

contract UniswapV2Pair is IUniswapV2Pair, UniswapV2ERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity event

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'UniswapV2: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'UniswapV2: TRANSFER_FAILED');
```

```solidity
    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'UniswapV2: FORBIDDEN'); // sufficient check
        token0 = _token0;
        token1 = _token1;
    }
```

Source: https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol

Create Liquidity Pool
Add Liquidity



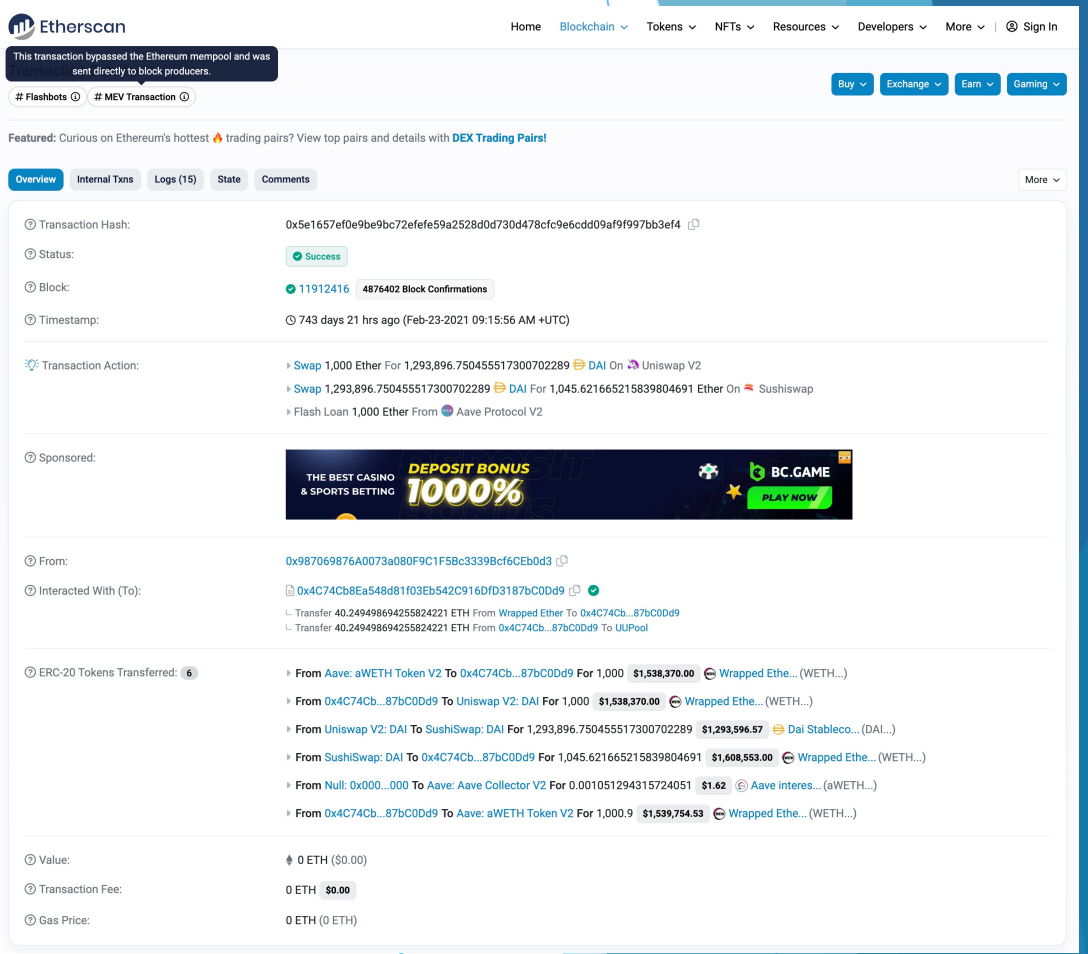
Through the Uniswap UI on https://app.uniswap.org.

Read more

- Implementation
  Read more:
  https://ethereum.org/ca/developers/tutorials/uniswap-v2-annotated-code/#conclusion
- liquidity-math
  Read more:
  https://atiselsts.github.io/pdfs/uniswap-v3-liquidity-math.pdf

Some interesting things related to LP: MEV(Maximal extractable value)

- **DEX arbitrage**
- Here's an example of a profitable arbitrage transaction where a searcher turned 1,000 ETH into 1,045 ETH by taking advantage of different pricing of the ETH/DAI pair on Uniswap vs. Sushiswap.

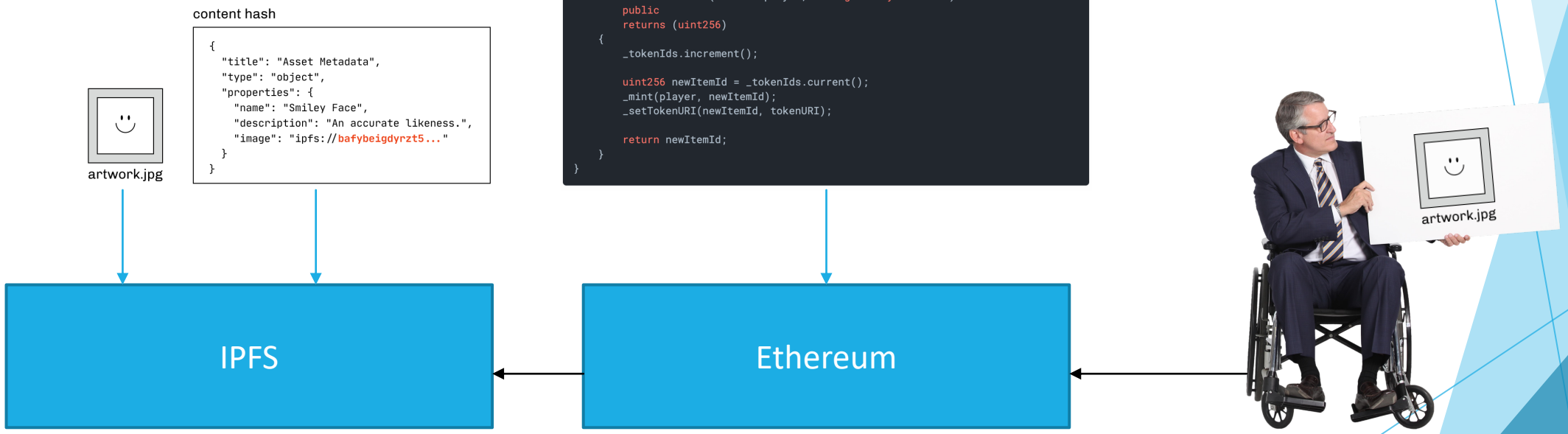


MEV after ETH2.0

- Stake more, more chance to do MEV transaction
  - In a proof-of-stake model, an algorithm selects which validator gets to add the next block to a blockchain-based on how much cryptocurrency the validator has staked.
  - Capitalist always win?

Launching NFT collectibles on ETH Overview

```
// contracts/GameItem.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract GameItem is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("GameItem", "ITM") {}

    function awardItem(address player, string memory tokenURI)
        public
        returns (uint256)
    {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(player, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```



Launching NFT collectibles on ETH ERC-721 / ERC-1155

- Example contract

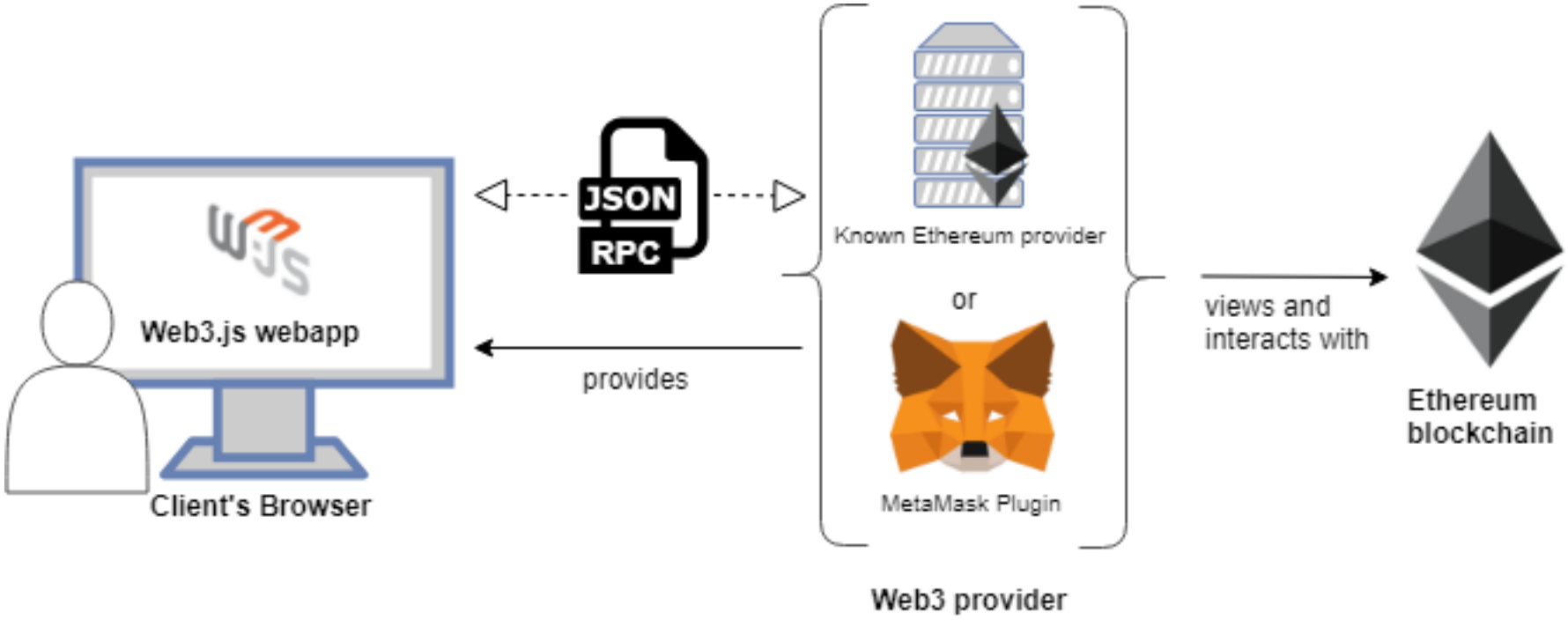
```sol
// contracts/GameItem.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract GameItem is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("GameItem", "ITM") {}

    function awardItem(address player, string memory tokenURI)
        public
        returns (uint256)
    {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(player, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```

```sol
{
    "title": "Asset Metadata",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "description": "Identifies the asset to which this NFT represents"
        },
        "description": {
            "type": "string",
            "description": "Describes the asset to which this NFT represents"
        },
        "image": {
            "type": "string",
            "description": "A URI pointing to a resource with mime type image/* representing the asset to which
        }
    }
}
```

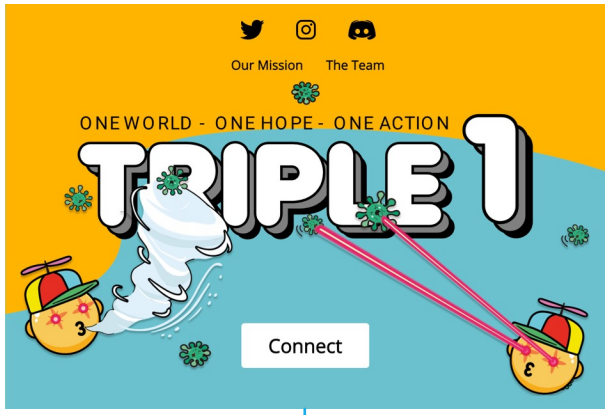
Source: https://docs.openzeppelin.com/contracts/4.x/erc721

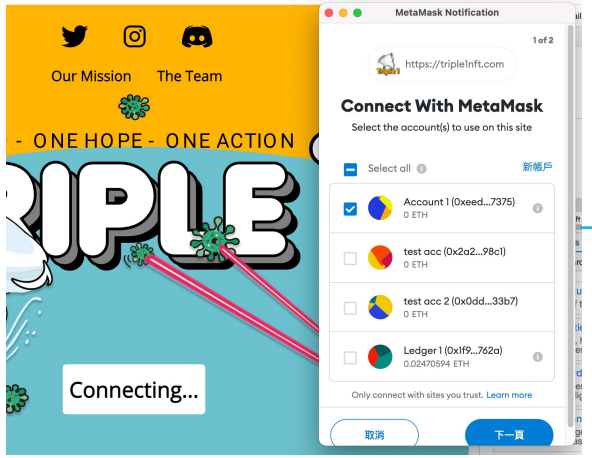
Launching NFT collectibles on ETH
User minting

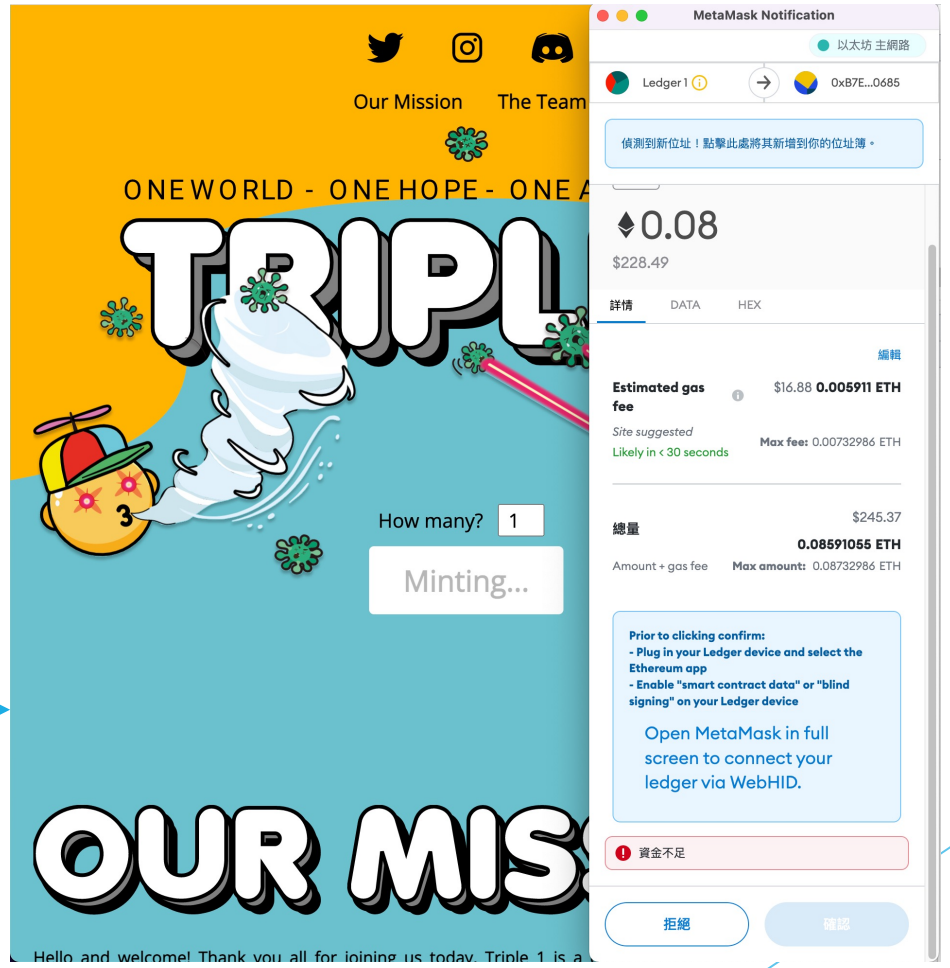
- Web3.js



Launching NFT collectibles on ETH
User minting



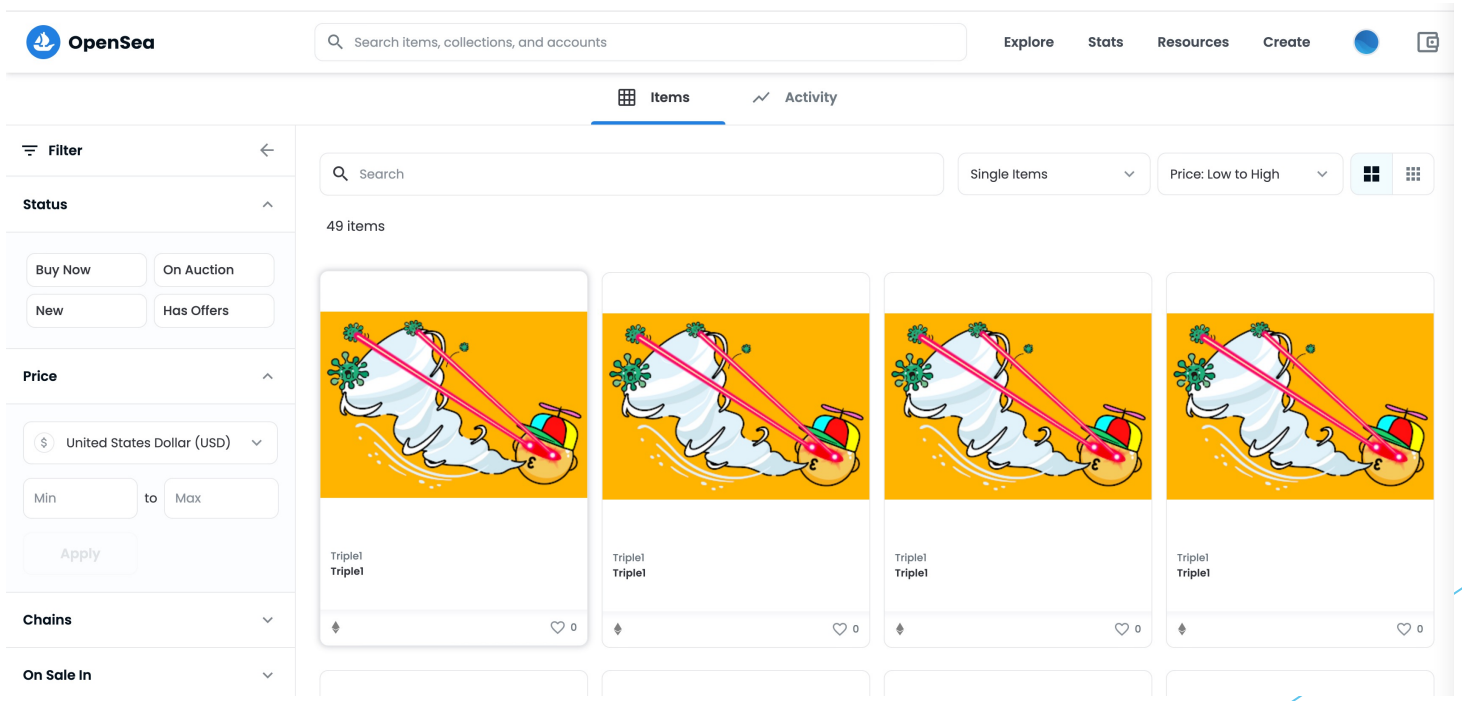




DIGIGEAR TECHNOLOGY LIMITED

Launching NFT collectibles on ETH
Secondary market on NFT marketplaces

- OpenSea



DIGIGEAR TECHNOLOGY LIMITED

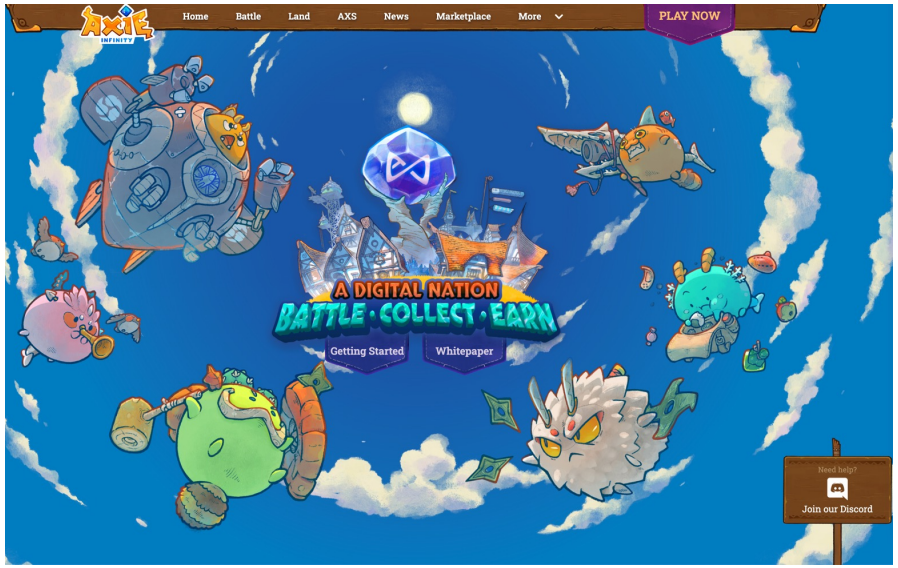
GameFi
What is GameFi

- The rise of GameFi is closely related to decentralized finance (DeFi) and non-fungible tokens (NFT).
  - The former makes NFTs commercial value through financial means; the latter endows game props with unprecedented scarcity and ownership guarantees of digital assets.
  - Then, the game itself is the surface and shell of the entire concept and technology.

- GameFi = DeFi + NFT + Game

GameFi Example

- Axie Infinity
  - Pokemon for crypto
  - Game-Fi, Play to earn card game



All Categories | Games | DeFi | Gambling | Exchanges | Collectibles | Marketplaces | Social | Other | High Risk

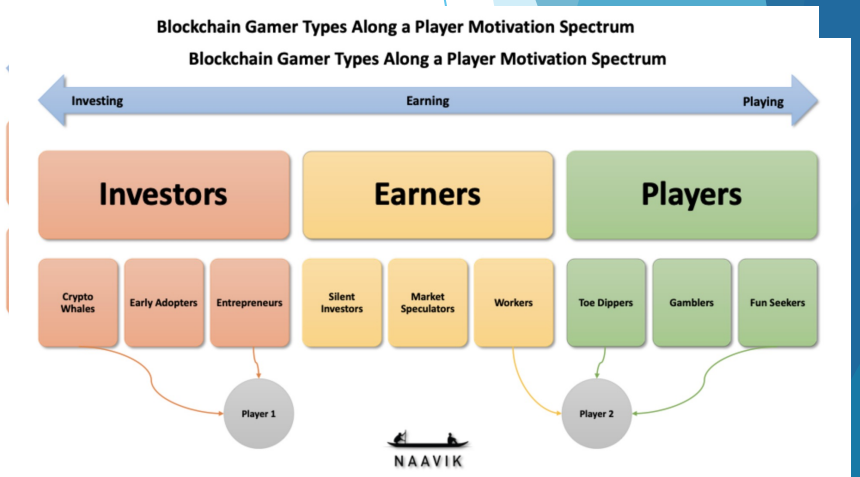
NEW | Filter | 24H | 7D | 30D

		CATEGORY	BALANCE	USERS	VOLUME	ACTIVITY
1	DeFi Kingdoms HARMONY • AVALANCHE	Games	$2.18B	21.55k +6.64%	$12.56M	
2	Axie Infinity RONIN • ETH	Games	$1.34B	30.88k +7.00%	$17.88M	
3	Arc8 by GAMEE Polygon	Games	$442.26M	15.98k +0.60%	$0	
4	Fantasy Space BSC	Games	$327.17M	–	$0	
5	REVV Racing Polygon	Games	$300.9M	2.18k +4.74%	$1.16k	
6	Crazy Defense Heroes ETH • Polygon	Games	$243.87M	48.2k +5.01%	$0	
7	BinaryX BSC	Games	$174.78M	370 -11.69%	$23.85k	
8	Illuvium ETH	Games	$134.74M	51 +10.87%	$34.09k	
9	MOBOX: NFT Farmer BSC	Games	$106.62M	36.31k +344.72%	$236.29k	
10	League of Ancients BSC	Games	$89.19M	1 –	$0	

NOT INVESTMENT ADVICE

GameFi Players

- Investors
  - "Investors" refer to the most active and early investors. These players simply treat the game as an investment.
  - Crypto Whales, Early Adopters, Entrepreneurs.
- Earners
  - The biggest difference between "profit-seekers" and investors lies in the degree and method of investment in the game.
  - Silent Investors, Market Speculators, and Workers.
- Players
  - The real players called "players" are, to a certain extent, the stakeholders who make blockchain games truly become games.
  - Toe Dippers, Gamblers, and Fun Seekers.



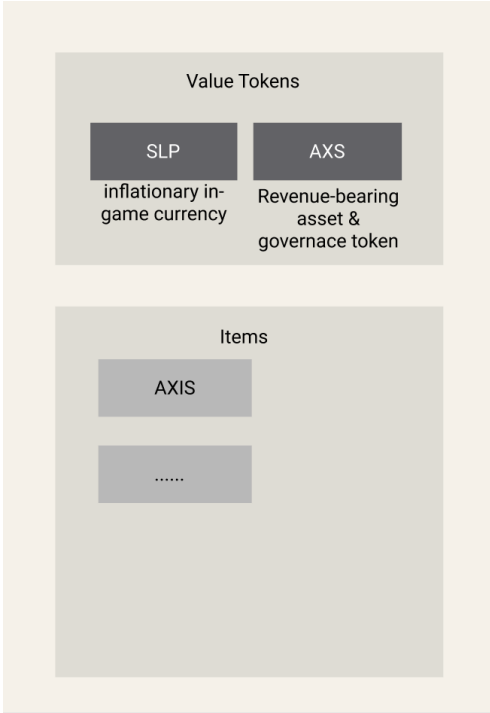
Source: https://naavik.co/themetas/blockchain-players

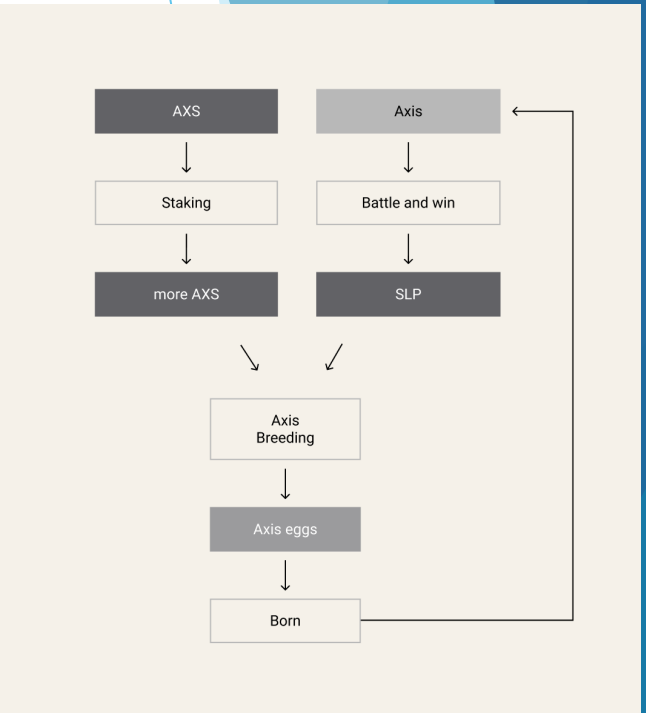
GameFi
Token Economy In Axie Infinity

GameFi
Token Economy In Axie Infinity

- Player must buy 3 or more Axies (pay by AXS) in order to battle
  - a 4.25% marketplace fee is taken from the seller
- Battle to win SLP
- Use SLP + AXS to breed more Axis

Loop…

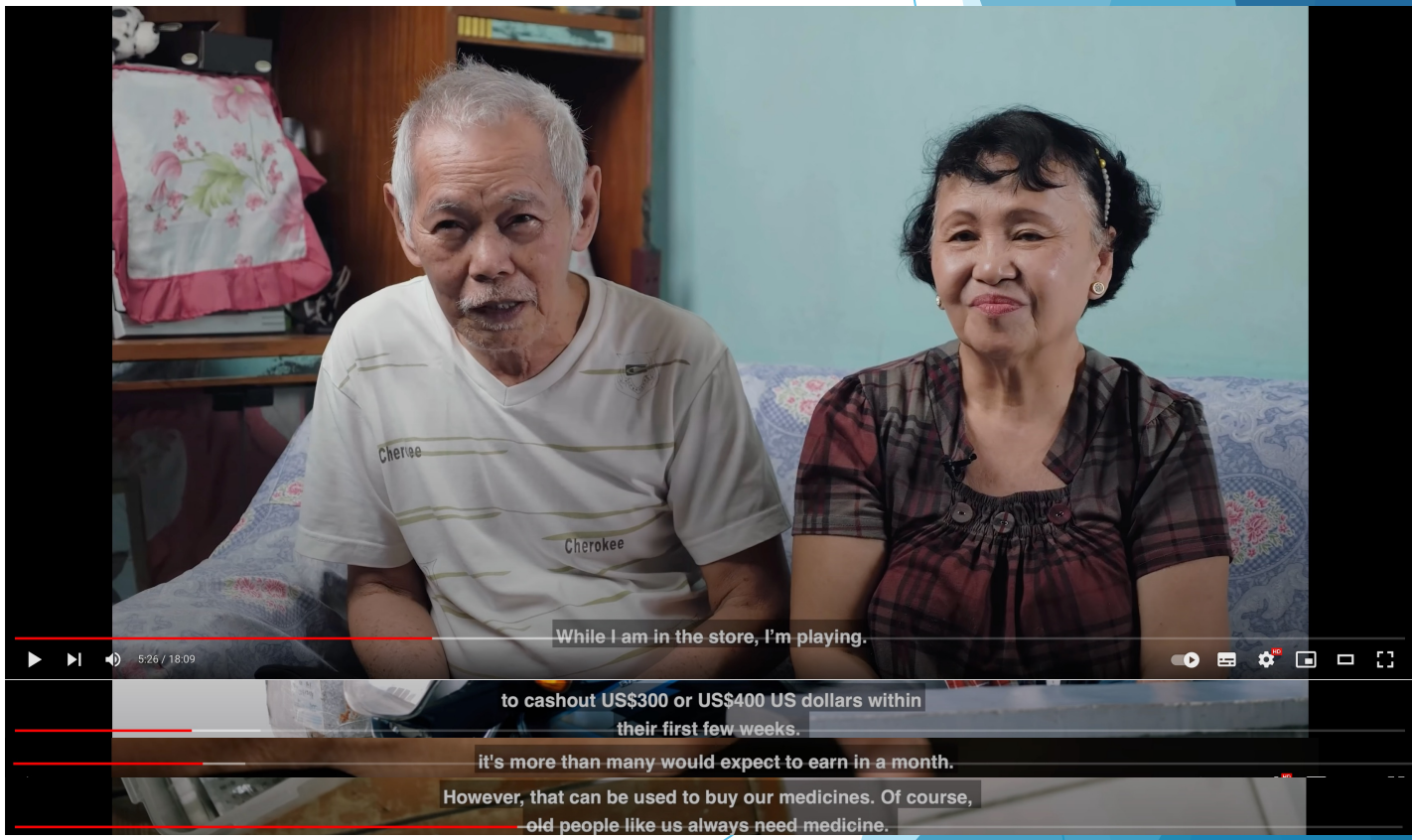




GameFi
Axie Infinity in Philippines

- People in the Philippines are earning cryptocurrency during the pandemic by playing Axie Infinity

- Read more: https://www.cnbc.com/2021/05/14/people-in-philippines-earn-cryptocurrency-playing-nft-video-game-axie-infinity.html



Source: https://www.youtube.com/watch?v=Yo-BrASMHU4

2.2 Blockchain review

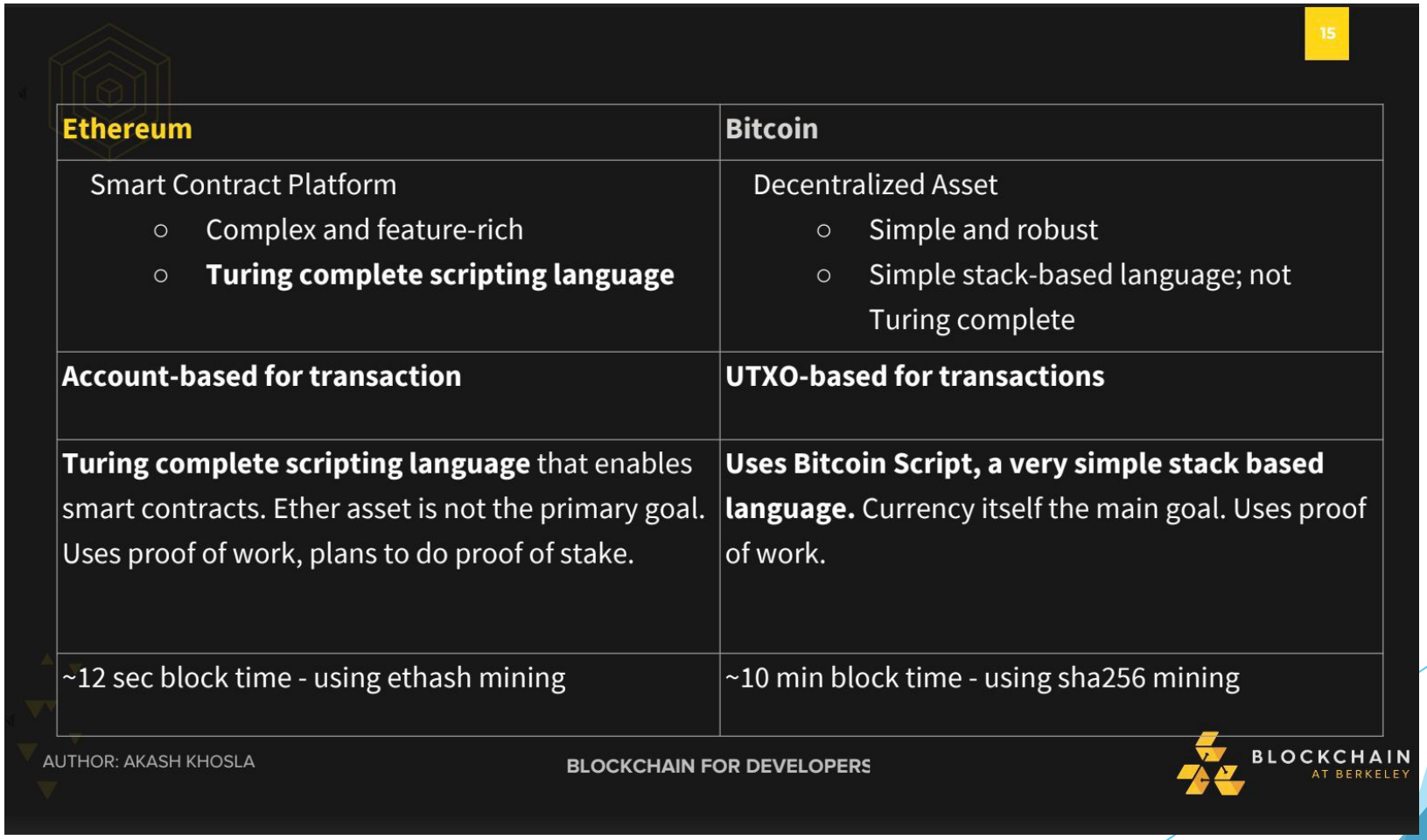
2.2.1 Ethereum Overview

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 transition function
  - Ethereum has a native asset called ether

ethereum

Ethereum Vs Bitcoin General Overview



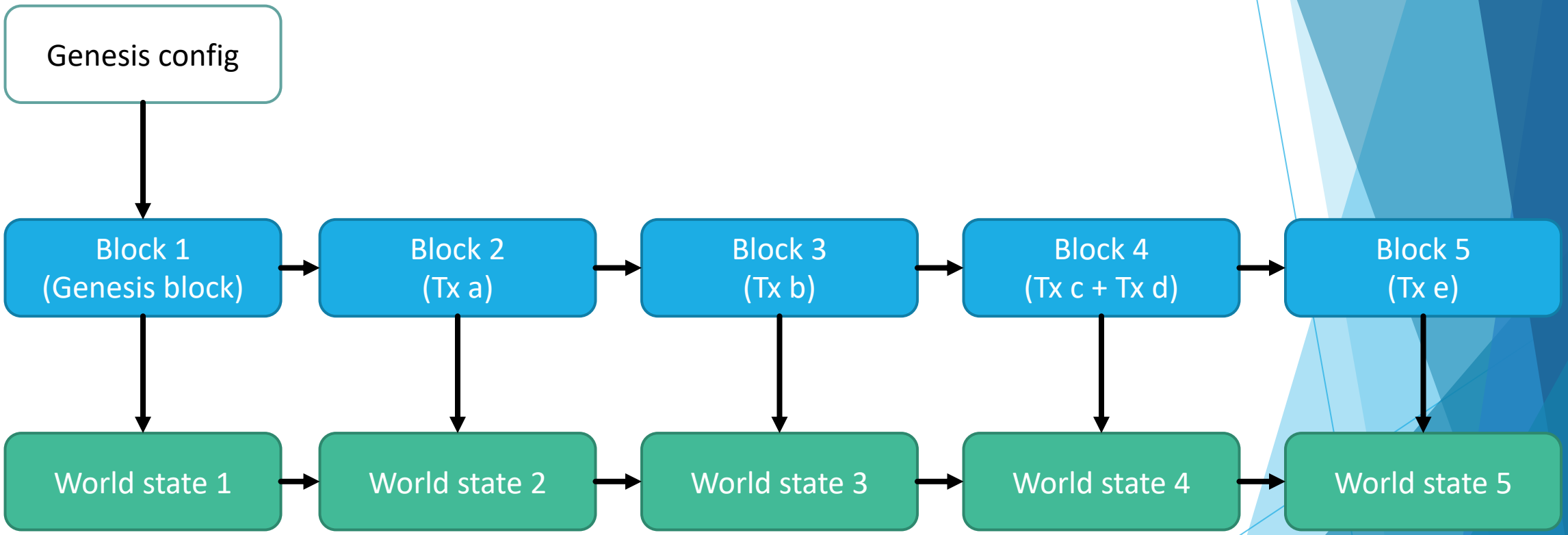
2.2.2 Account

UTXOS VS Accounts

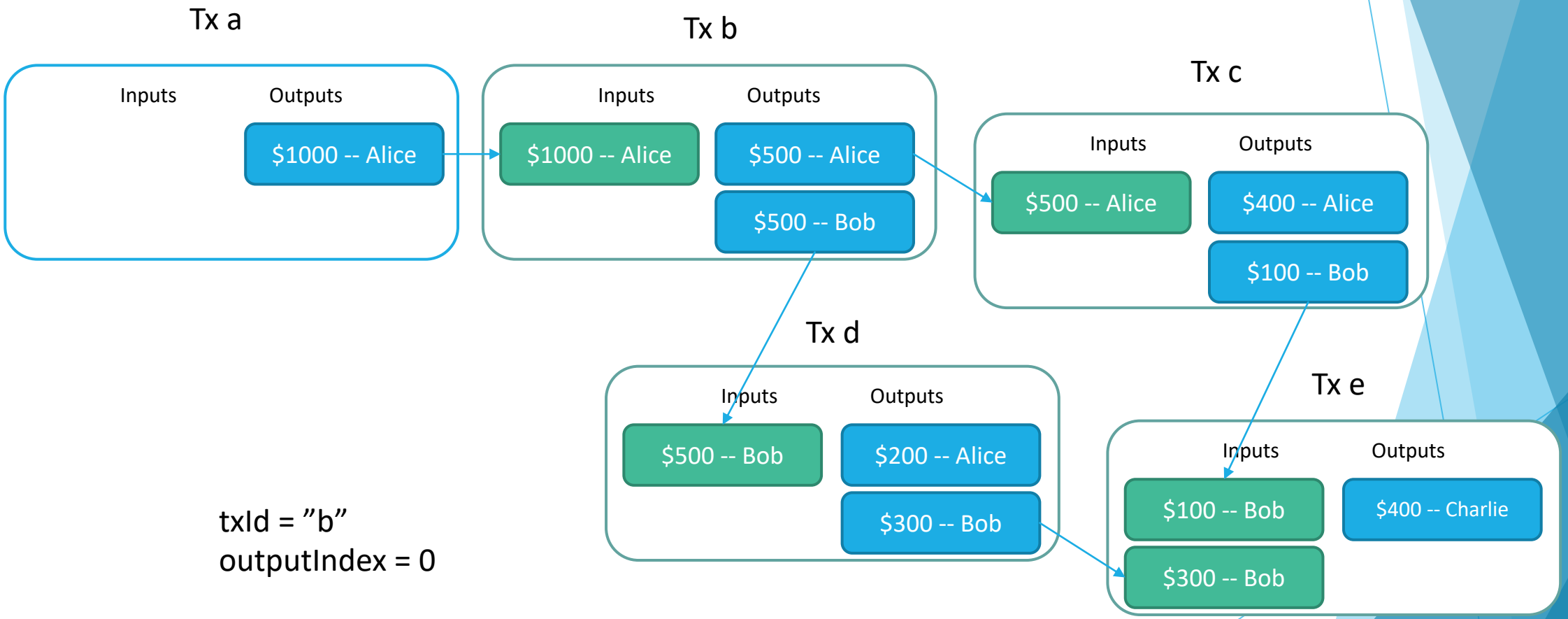
UTXO stands for Unspent Transaction Output

- **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**

Blockchain recap



Why UTXOS?
UTXO model & UTXO Set – Example



txId = ”b”
outputIndex = 0

txId = ”b”
outputIndex = 1

UTXOS VS Accounts

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

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

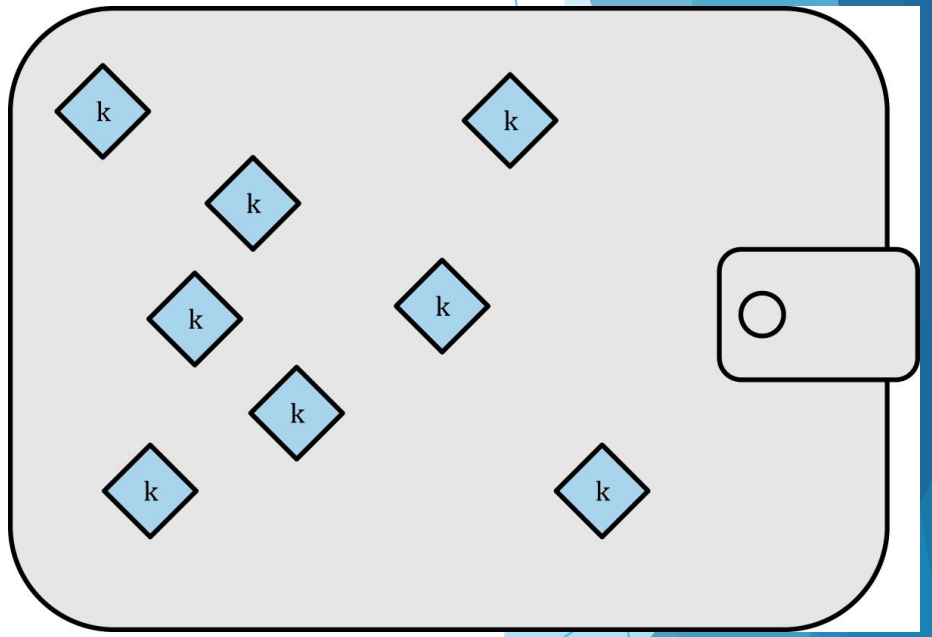
Why Accounts?
Benefits Of The Accounts Model

- 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 querying a UTXO set to compute balance
  - Easier to program smart contracts

2.2.3 Types Of Wallets

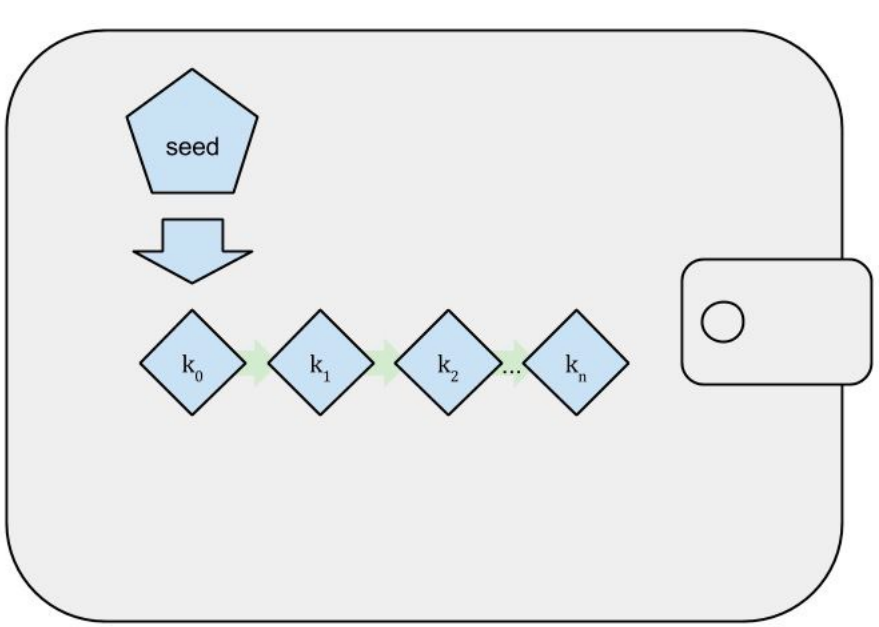
Nondeterministic Wallets

- Nondeterministic wallet:
  - A collection of independently randomly generated keys
- Disadvantages:
  - cumbersome to manage, back up, and import
  - Reduces privacy due to address reuse



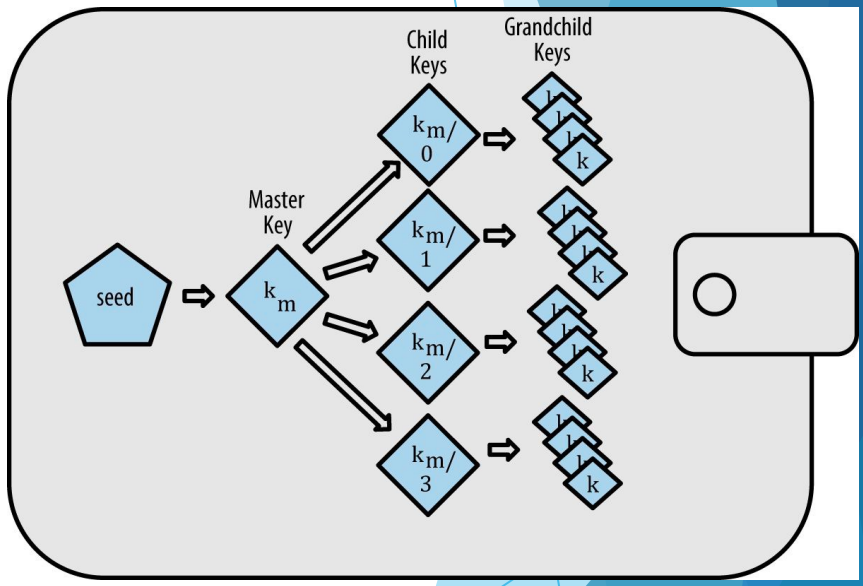
Deterministic Wallets
Seeded

- Deterministic wallet:
  - A deterministic sequence of keys derived from a seed using a hash function
- Advantage:
  - the seed is sufficient to recover all the derived keys -> efficient backup



Deterministic Wallets
HD Wallets

- HD wallet:
  - A collection of keys derived in a tree structure
- Advantage:
  - Additional structural management: giving access to specific branch of keys
  - users can create a sequence of public keys without having access to the corresponding private keys



Multisig Wallets

- Contract Wallets can be setup as Multisig Wallets.
- Simple Wallets and Multisig Wallets are both examples of Contract Wallets.
- A Simple Wallet:
  - only one Account both creates and owns the wallet
- Multisig Wallet:
  - has several owner Accounts one of which will also be the creator Account.
  - M-of-N type wallets

Multisig Wallets
What Does It Need?

- A list of **people** who are allowed to do things
- Rules on **how many** of those people **have to agree** before it happens
- A way to **receive** ether
- A way to **submit** a request
- A way to **agree** to a request (and submit it if you are last)
- A way to **resubmit** the request if it fails

We will come back to this topic later since there was a sad story…

2.2.4 GAS

GAS

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

What Is Gas?

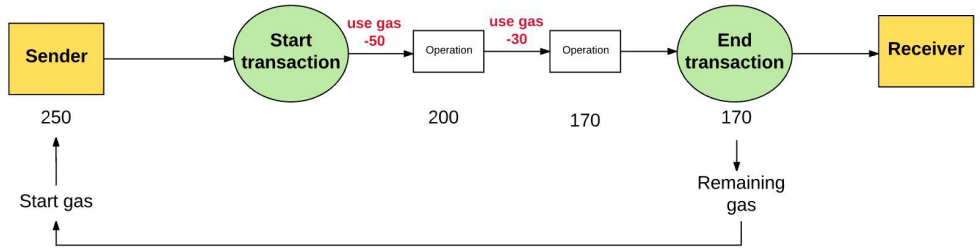
- **Gas**: measures the computational “work” necessary to perform an action
  - ex. one Keccak256 hash: 30 gas + 6 gas per 256 bits of data
  - Detailed in Ethereum Yellowpaper: https://ethereum.github.io/yellowpaper/paper.pdf
- **Gas limit**: maximum amount of gas the user is willing to spend
- **Gas 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 = 10^18 gwei

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

How Transactions Work

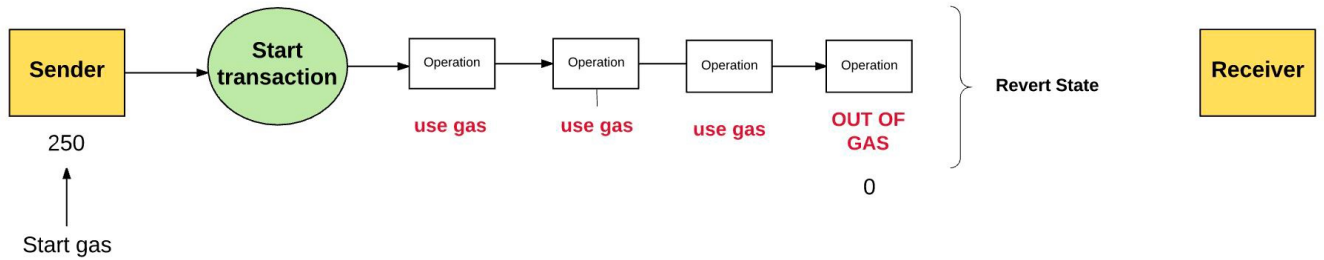
- With every transaction, the sender specifies a gas limit and a gas price
  - gas price * gas limit = max amount sender is willing to pay
- Example:
  - Gas Limit: 50,000, Gas Price: 20 gwei
  - Max transaction fee: 50,000 * 20 gwei = 1,000,000,000,000,000,000 wei = 0.001 Ether
- For the transaction 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



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



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

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

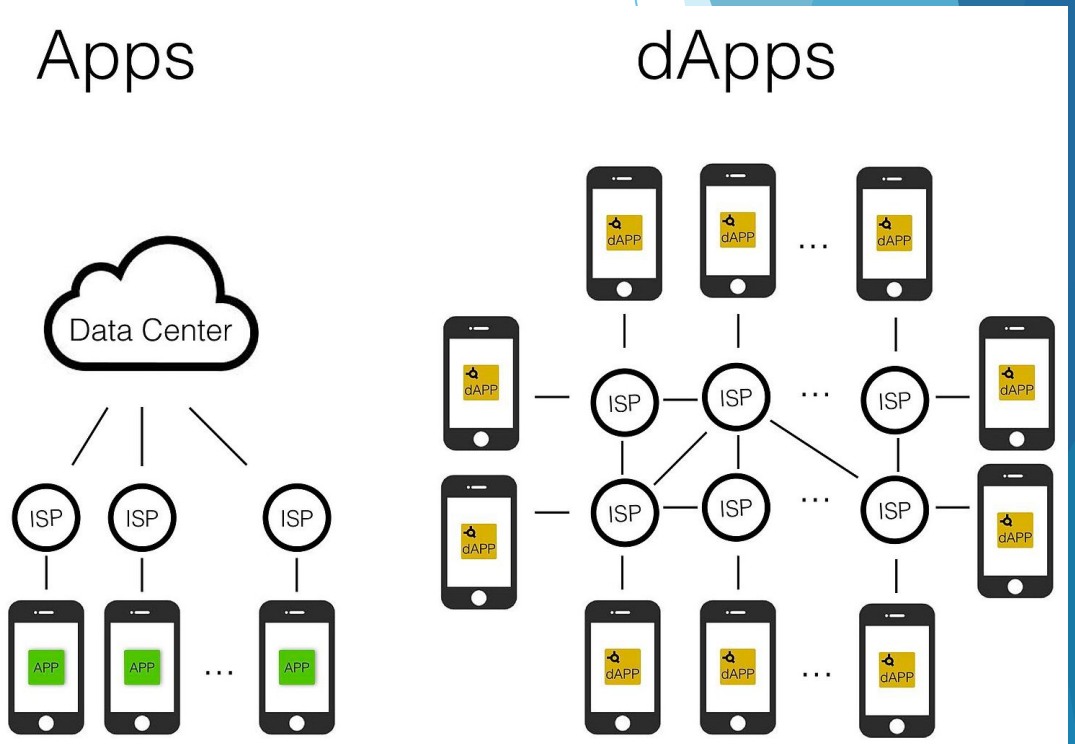
2.2.5 Smart Contracts

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

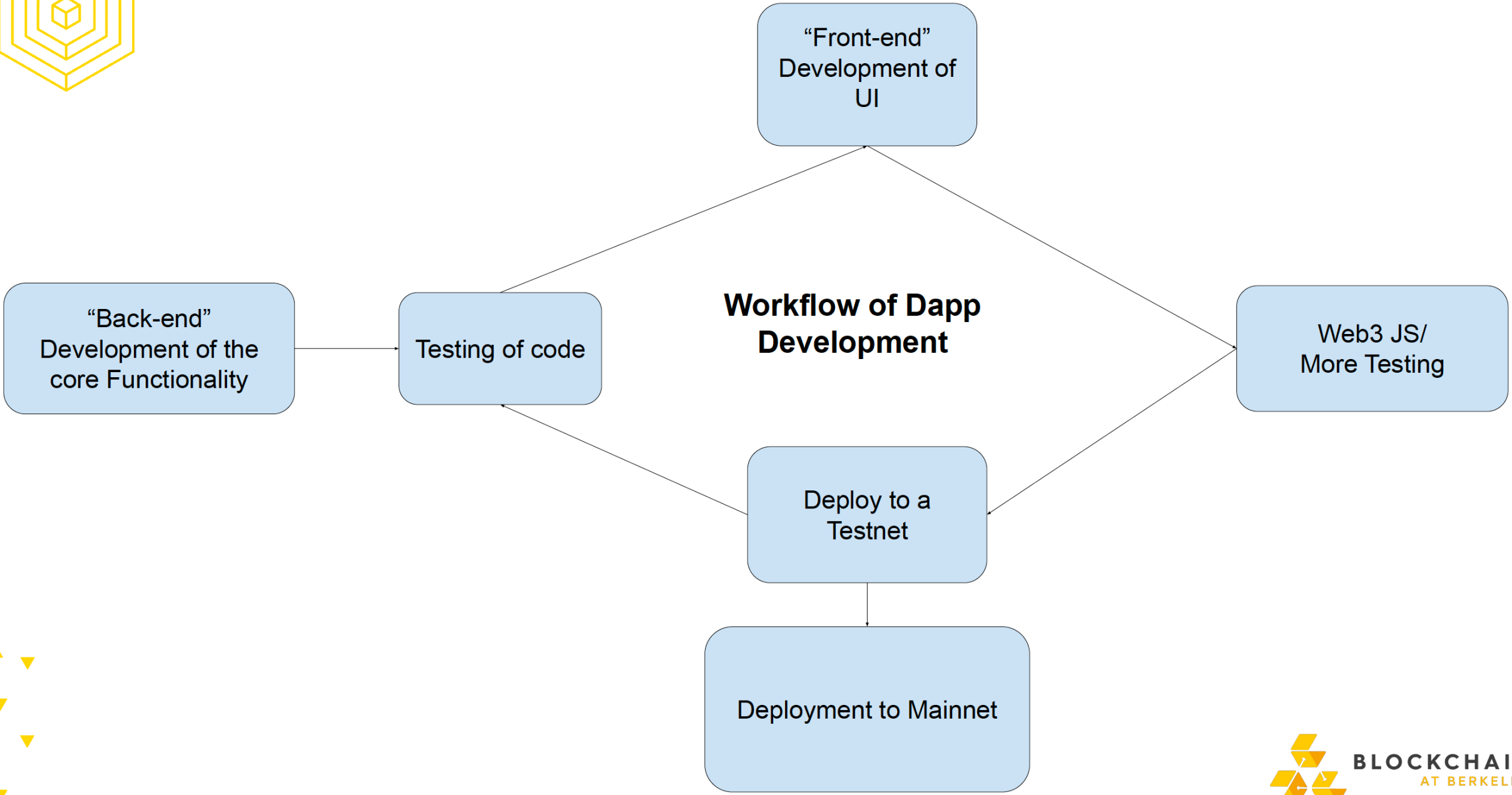
DAPPS
High-level Overview

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



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

2.3 DeFi hacks

Overview
Why Is Public Blockchain Security Hard?

- ANYONE CAN:
  - Call whatever functions of it they want
  - See the bugs or vulnerabilities

- Incentive Structure:
  - Asks users to find and exploit your code to maximize gain

Overview
Why Should You Care?

- Immutable systems make it so that exploits are often extremely difficult to reverse the effects of
  - Without a legal system in place, security is the only thing that prevents network adversaries from destroying or bankrupting your protocol
  - Has a very strong tie to how much people trust your product

History
What is DAO (Decentralized autonomous organization)

Virtual venture capital (VC) fund that is governed by the investors of the DAO

- Main Idea:
  - Token holders can become contractors by submitting proposals for funding of their project by using the DAO funds
  - Proposals are approved by a quorum of 20% of all tokens
  - Transfers Ether automatically to the Smart Contract that represents the proposal
- Raised 12.7 mil. Ether (150 mil USD in June 2016)
- BUT...they got **H A C K E D**.

DAO HACK
What was the HACK?

- **“SPLIT PROCEDURE”, THE CATALYST OF THE HACK.**
  - The creators of DAO implemented the ability of a DAO to be split into 2
  - “Minority” = able to retrieve funds when a proposal they objected gets approved
    - At least 48 days before getting it in an account YOU control
- **THE HACK:**
  - Coder found a loophole in this procedure
  - Once a split function is called, the code was written in a way to **retrieve ether → update balance**
  - Recursively call the “split procedure” and retrieve their funds **multiple times** before getting to the step where the **code checked the balance**
- **RESULT OF THE HACK:**
  - On June 16, 2016, the attacker retrieved **~3.6 million Ether = 150 mil. USD**
  - Now known as, the **“recursive call exploit”**

DAO HACK
Check Permissioning

```
// constructor - just pass on the owner array to the multiowned and  // the
limit to daylimit
function initWallet(address[] _owners, uint _required, uint _daylimit) {
  initDaylimit(_daylimit);
  initMultiowned(_owners, _required);
}
```

- Attacker discovered that the default function routed all calls to the Wallet Library, which had an unpermissioned init Wallet function
  - Allowed anyone to freely change wallet ownership: change owner rights and able to kill it
  - Freeze 573 wallets, with balances of more than 500K Ether (~152 mil USD)

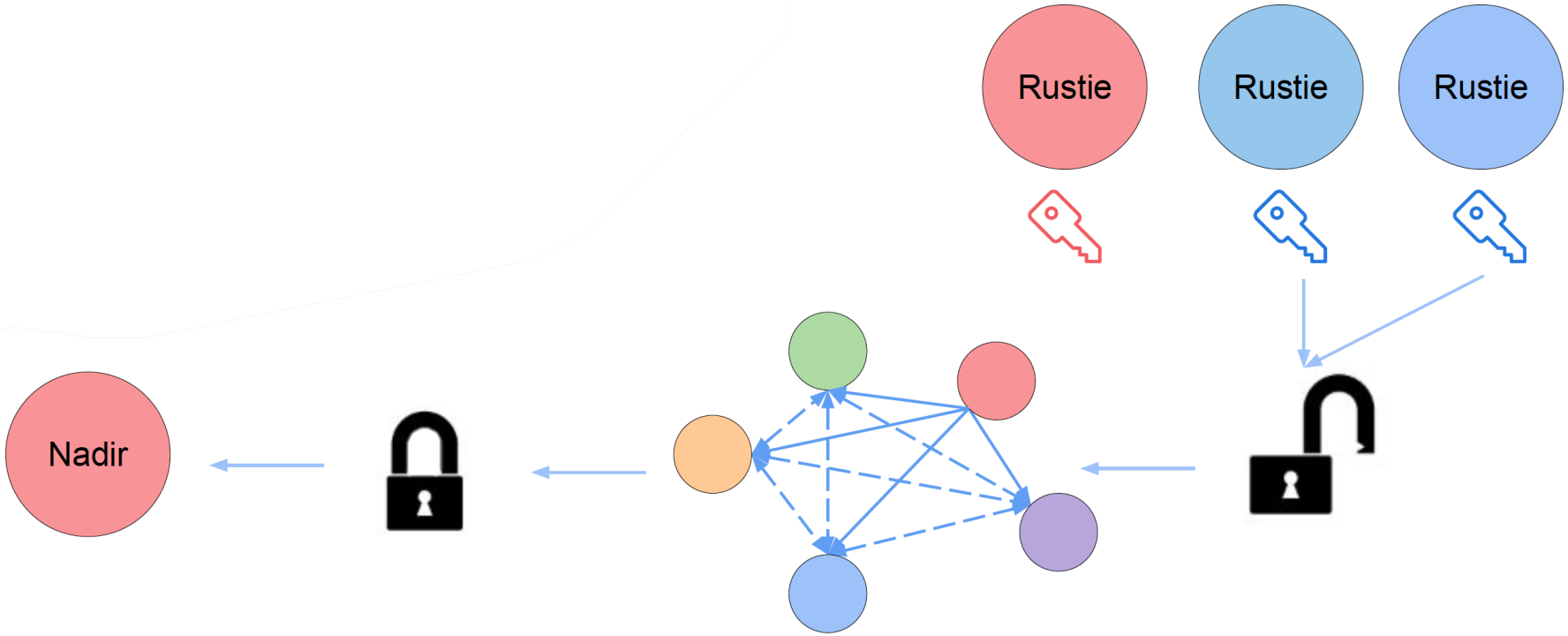
Multisig Wallet HACK
PARITY

- **PARITY** is a company responsible for multiple “core blockchain” solutions, such as:
  - - A multisig wallet
  - - An Ethereum client
  - - Polkadot, for interoperable blockchains

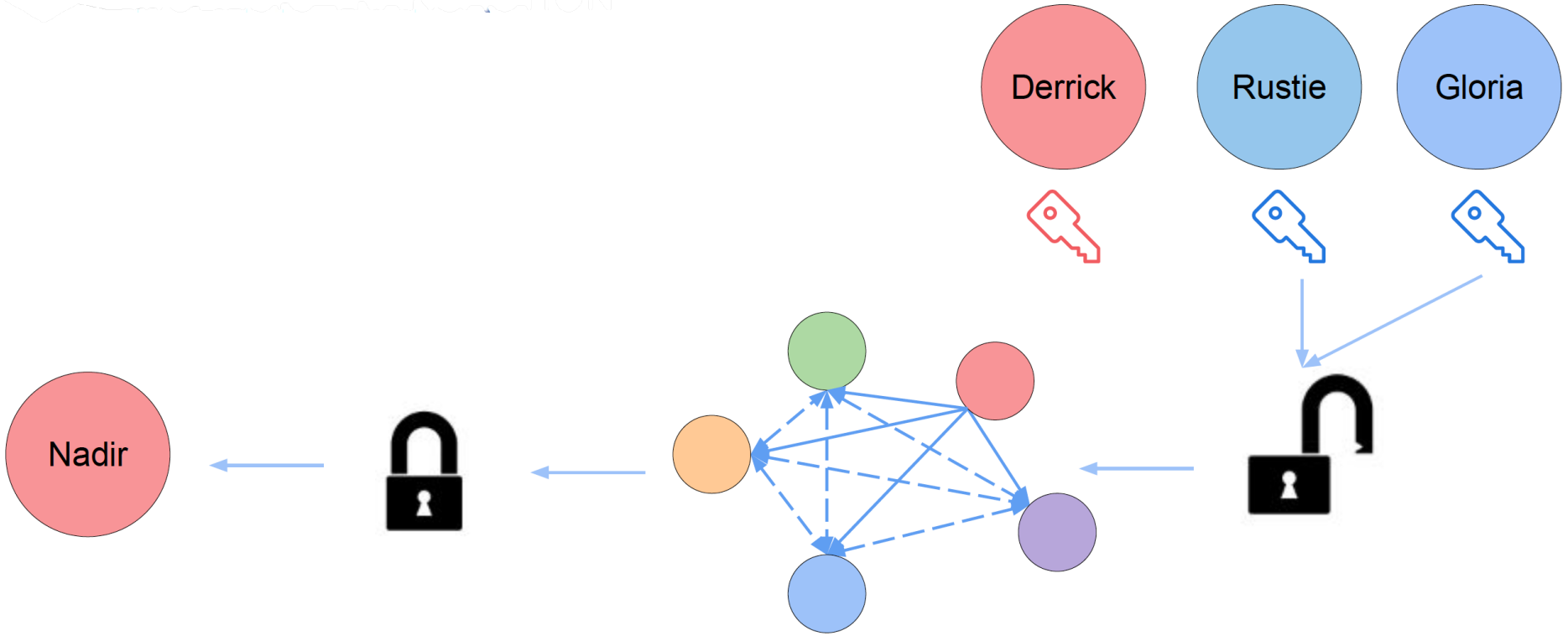
...but what went wrong?

What Is A Multisig Wallet

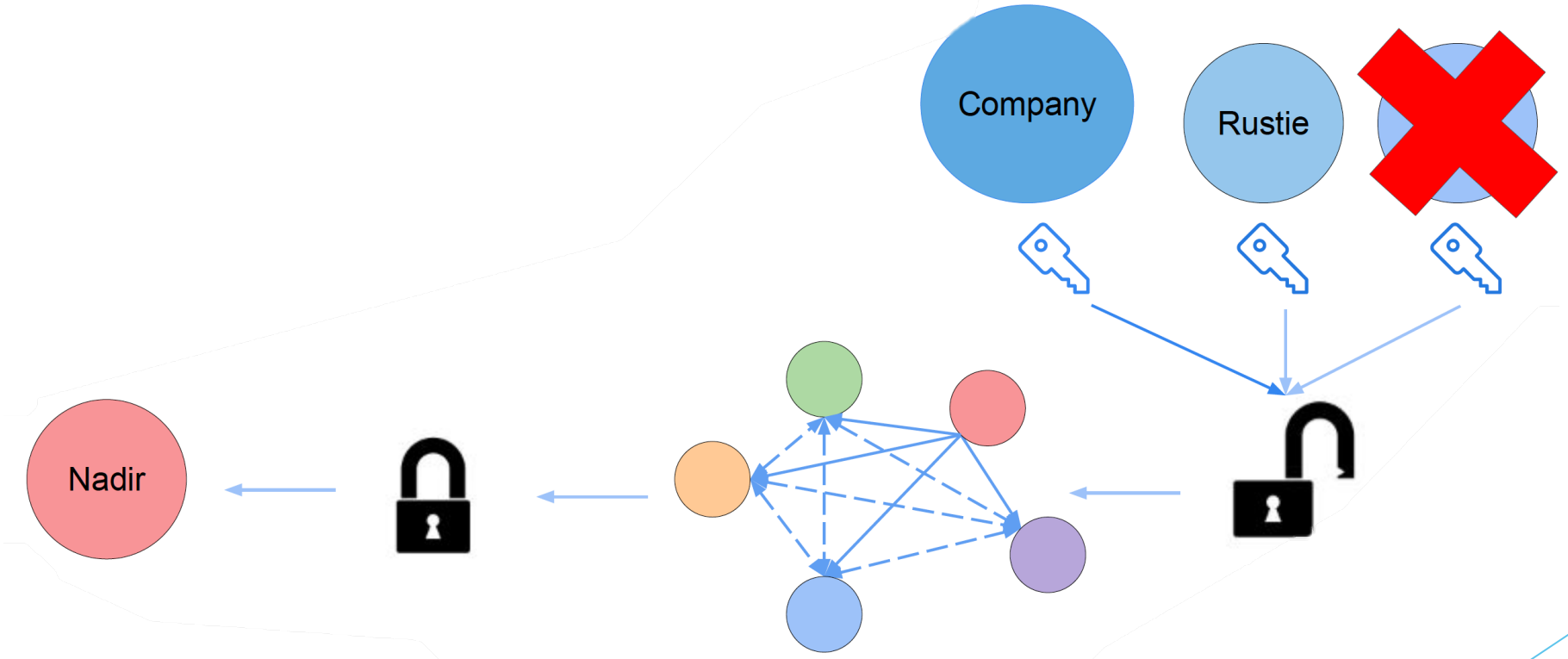
- A multisig wallet exists to allow for a single account to allow for multiple users to produce valid signatures from that account, requiring a minimum threshold.
- For example, a 2-of-3 multisig wallet requires 2 signatures from a total of 3 privileged users.



- **Use case**: You own some ether and want to store it securely but are concerned that just relying on a single private key may not be safe. So you create a Multisig Wallet, a 2-of-3 wallet, but all the owner accounts are under your control, that is, you control all the private keys.



- **Use case**: You set up a Multisig Account jointly owned by say three people. So you have one private key and one for each others

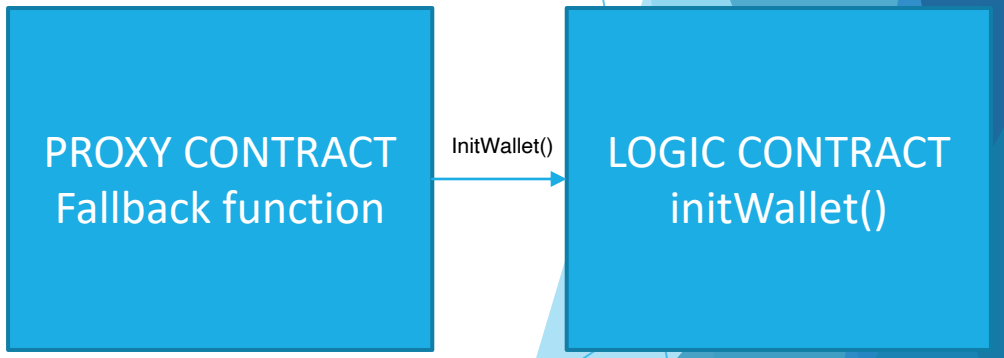


Multisig Wallet Hack #1

```
function() payable {
  // just being sent some cash?
  if (msg.value > 0)
    Deposit(msg.sender, msg.value);
  else if (msg.data.length > 0)
_walletLibrary.delegatecall(msg.data);
}
```

```
// constructor - just pass on the owner array to the multiowned and  /
function initWallet(address[] _owners, uint _required, uint _daylimit)
  initDaylimit(_daylimit);
  initMultiowned(_owners, _required);
}
```

- Contract design:
  - Proxy contract as interface
  - Logic contract to hold the actual logic (library contract)
- An attacker made two calls:
  - One to a fallback function
  - The other to redeem funds
- The fallback function would delegate arbitrary calls to the logic contract
  - Beware “arbitrary”!



The initWallet function could be called more than once... requiring a National Treasure whitehat rescue

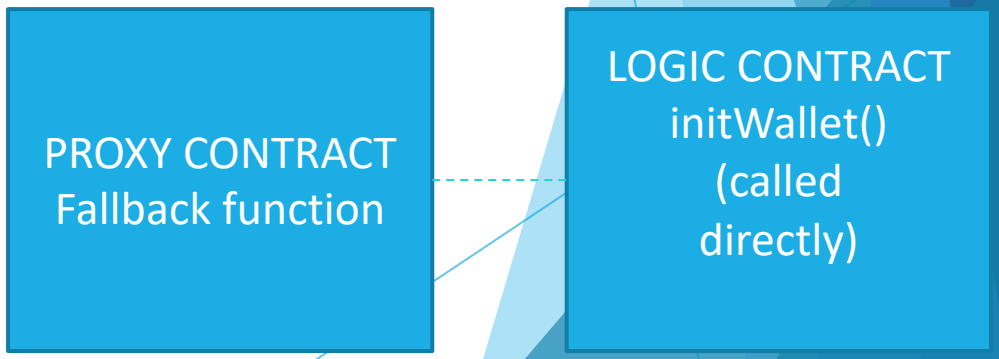
Read more: https://blog.openzeppelin.com/on-the-parity-wallet-multisig-hack-405a8c12e8f7/

Multisig Wallet Hack #2

- The previous fix? Add a modifier to ensure that it has not been initialized
- The consequence? No one initialized the logic contract!
- The logic contract was:
  - 1. Directly called and initialized
  - 2. Self-destructed (“killed”)
- Now all proxy contracts are useless, meaning all funds held are frozen

```
modifier only_uninitialized { if (m_numOwners > 0) throw; _; }
```

```
// gets called when no other function matches
function() payable {
  // just being sent some cash?
  if (msg.value > 0)
    Deposit(msg.sender, msg.value);
  else if (msg.data.length > 0)

_walletLibrary.delegatecall(msg.data);

}
```



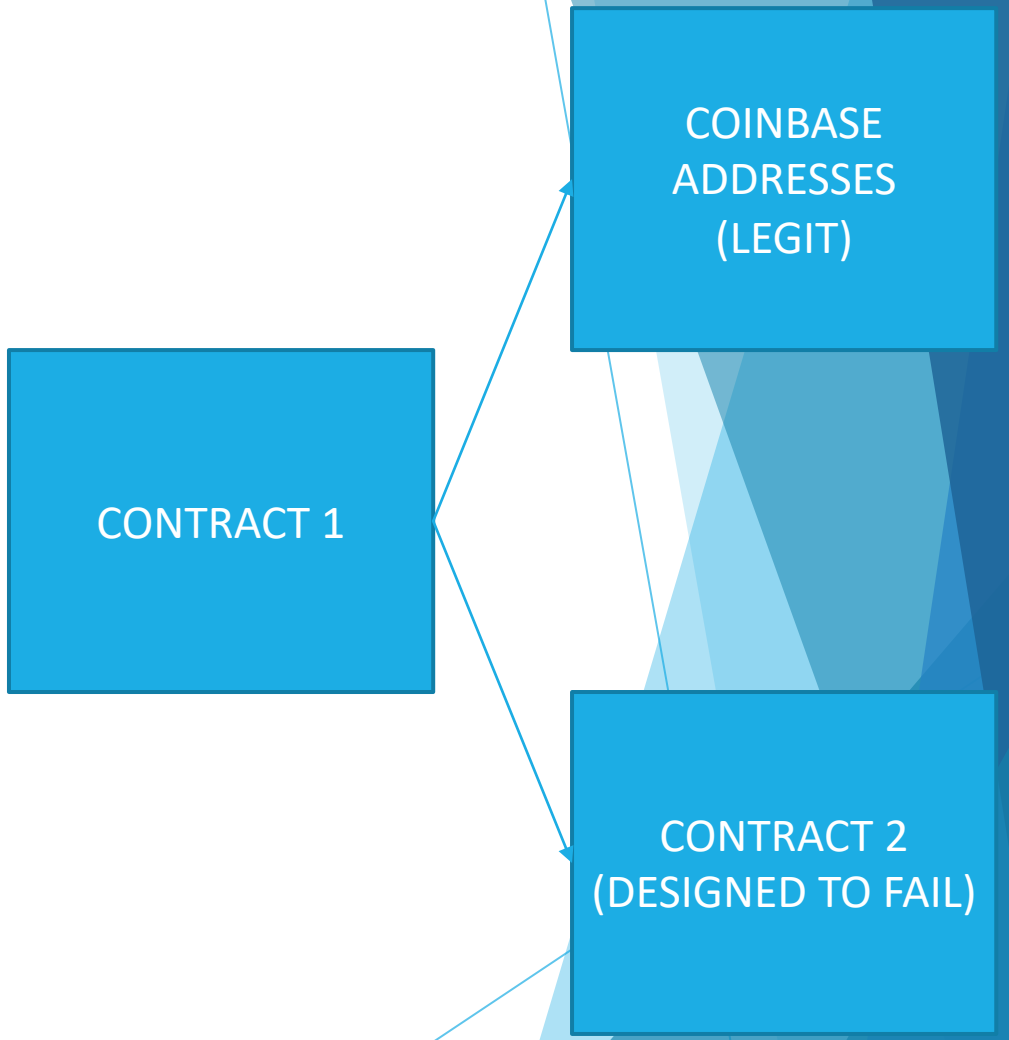
Readmore: https://blog.openzeppelin.com/parity-wallet-hack-reloaded/

Coinbase

Coinbase is an exchange for cryptocurrencies that misconfigured the way it parsed transactions over two years ago.

- The following takes place in a single transaction:
  - 1. Contract 1 sends funds to some number of Coinbase addresses
  - 2. Contract 1 makes a call to another contract, which always reverts.

Coinbase didn't recognize the revert, meaning the on-chain balance stayed the same but Coinbase credited the deposit.



Readmore: https://cointelegraph.com/news/coinbase-bug-allowed-users-to-steal-unlimited-eth-wallet-paid-10k-bounty-for-discovery

2.4 Defenses

What Can I Do?

- Don’t write fancy code
- Code adversarially, the contract should take before giving
- Use audited and tested code
- Write as many unit tests as possible
- Prepare for failure - at any moment, in any contract or method
- Rollout carefully - bug bounties before ICO
- Keep contracts simple - more complexity = more attack vectors
- Keep updating with software and community
- Beware of blockchain properties: public vs. private, .send() vs. .call()

Randomness

- Everything on the Ethereum Virtual Machine (EVM) is deterministic
  - People have tried to use now and block.blockhash for business logic in a contract
  - However, results are predictable and can be manipulated by miners
- “Timestamp dependence”
  - Block numbers and average block time can be used to estimate time,
  - but this does not act as future proof as block times may change

Pragma Statements

- Make sure to lock contract solidity versions
- New versions always come out that may introduce some new vulnerabilities if contract isn't locked to one version

```
// dangerous

pragma solidity ^0.5.0;

// good practice

pragma solidity 0.5.0;
```

External Calls

- Avoid external calls when possible
  - What is an external call?
  - A call from one contract to another untrusted contract or account.
  - delegatecall, callcode, call
- Previous (Expensive) Bugs:
  - The Dao hack
  - The Parity multisignature wallet hack

PULL > PUSH

How do we get data/value to addresses?

- **Push**
  - Contract transfers immediately to addresses after logic
  - **Potentially vulnerable if recipient contract is poisoned**

- **Pull (!!)**
  - “Lazy”, give addresses the ability to get their data or money
  - Like a withdraw function
  - **Use these**

ASSERT AND REQUIRE

Use assert and require properly

- Require(condition) is meant to be used for input validation
  - Should be done on any user input, and throws if condition is false.
- Assert(condition) also throws if condition is false
  - Should be used only for internal errors or to check if your contract has reached an invalid state.
- By following this paradigm it would become possible to use a formal analysis tool, that verifies that the invalid opcode can never be reached, for the absence of errors assuming valid inputs.

Rounding With INT Division

All integer division rounds down to nearest integer

If you need more precision, consider using a multiplier, or store both the numerator and denominator.

```
// bad
uint x = 5 / 2; // Result is 2, all integer division rounds DOWN to the nearest integer

// good
uint multiplier = 10;
uint x = (5 * multiplier) / 2;

uint numerator = 5;
uint denominator = 2;
```

All Data Is Public

- **Remember that on-chain data is public**
  - Many applications require submitted data to be private up until some point in time in order to work
  - If you are building an application where privacy is an issue, take care to avoid requiring users to publish information too early
- **Example: Rock – Paper – Scissors**
  - Require both players to submit a hash of their intended move first
  - Require both players to submit their move
  - If the submitted move does not match the hash throw it out

Always Remember Testing

- As seen from all these possible attacks and vulnerabilities, it is vital to always try and get 100% coverage of testing
- What this means is that all functions and possible interactions within the contract are completely tested for

Tools To Help

- **Solidity** is taking steps to introduce semantic checking of code
- **Securify.ch** is a static analysis tool for Smart Contracts.
- **Hydra** is a “framework for cryptoeconomic contract security, decentralized security bounties”
- **Porosity** is a “decompiler for Blockchain-based Ethereum Smart-Contracts”
- **Manticore** is a dynamic binary analysis tool with EVM support
- **Remix** performs static analysis to your code

OpenZeppelin

Zeppelin is a library for writing secure smart contracts on Ethereum

What kind of contracts are available?

- Ownership: declaring, claiming transferring
- Safe Math: takes care of integer overflows and overdrawing
- Push and Pull: deposit and withdraw methods you can trust
- Standard and Basic Token: ERC20/721/1155 implementable and ready out of the box

Github: https://github.com/OpenZeppelin/openzeppelin-contracts

3. Enterprise Blockchain

3.1 Introduction to enterprise blockchain

Differences with public blockchain

- **Sovereignty**: An enterprise blockchain is generally owned and operated by a consortium
  - Consists of major industry players
- **Consensus**: Consensus is achieved by CFT / BFT algorithms run by a small number of validator nodes. Potential to support > 1000 tps with sub-second transaction confirmation.
- **Regulation**: Enterprise blockchains are operated under regulation by laws and legal agreements such as consortium agreements and rulebooks
- **Privacy**: Privacy is valued highly in order to protect business interests and satisfy regulatory requirements

Technologies

- Blockchain platforms
  - Ethereum / Quorum
  - Hyperledger Fabric
  - Ethereum
  - Corda
  - … Many more proprietary solutions

Applications

- Banking and finance
  - Financial instruments (cash, bond, fx)
  - Payment with immediate settlement
- Trade and logistics
  - Supply chain provenance
  - Trade finance
- Healthcare
  - Medical record sharing
- Certification and audit trail
  - Academic certificates

Key value propositions

- Removal of data silos and operational burdens
  - Enable interoperability between proprietary systems, through **standardized protocols on blockchain**
  - Remove unnecessary manual checking
- Audit trail
  - Track items throughout the lifecycle across multiple owners
  - Identify fraudsters
- Creating better financial products
  - More automated, less risky
  - The epitome of smart contract

Challenges

- Large upfront investment
- Large adoption cost
- Requires industry collaboration among players who are potentially competitors

3.2 Enterprise applications

Trade finance: Contour

- Trade finance
  - Trade – Import and export of goods
  - Trade finance
    - Importers and exporters
    - Banks / financiers
- Focus on letter of credit
  - Banks as intermediaries for document checking and invoice payment

Trade finance: Contour

- Traditional LOC:
  - Release of payment requires mailing physical documents, manual document checking and back-and-forth communication
  - End-to-end process takes 10 days on average
- Contour digital LOC:
  - Release of payment can be mostly automated
  - End-to-end process can be completed within 24 hours

Security token offering

- Traditional: IPO – Initial public offering
  - Asset can be traded on public exchanges
  - Publicly listed companies must be transparent
- Crypto: ICO – Initial coin offering
  - Ownership tracked on blockchain, traded on crypto exchanges or DEXs
  - Unregulated: Utility tokens, not tied to stocks / bonds
- New: STO – Security token offering
  - Ownership tracked on blockchain, traded on security token exchanges
  - Regulated: Security tokens, legal framework still in development

https://www.tannerdewitt.com/security-token-offerings/
https://www2.deloitte.com/content/dam/Deloitte/cn/Documents/audit/deloitte-cn-audit-security-token-offering-en-201009.pdf

Security token offering

- Benefits:
  - Improves liquidity for traditionally illiquid assets
    - e.g. fractionalization of collectibles, real estates
  - Easier and lower admin cost to raise funds
  - Protection from fraud thanks to regulations
- Challenges:
  - Risky to navigate the regulatory landscape

Payment

- Blockchain-based permissioned settlement system
- Consumer payment network targeting underbanked countries

Certification & Provenance

- Upload certificates and provenance records on blockchain
- Secured by digital signatures
- Easy to verify the genuineness of the records

3.3 Central bank digital currency (CBDC)

Motivation

- Cryptoassets have volatile prices
- Stablecoins present issuer risks
- A digital currency issued by a central bank, i.e., CBDC, would be compatible with the financial system and present smaller risks
- A financial ecosystem can be developed based on CBDC
  - Traditional finance but Better with Blockchain

Research direction

- CBDC will generally be an enterprise blockchain
  - Central bank has control over the network infrastructure
- **Wholesale CBDC**: CBDC that circulates between banks and relevant financial institutions
  - Aims to solve problems like inter-bank clearing, immediate settlement, management of treasury securities, international payment, foreign currency exchange, etc.
  - Researched in past 4 years
- **Retail CBDC**: CBDC that is available to the general public
  - Actively researched in recent 2 years

Challenges

- **Sovereignty**: Each central bank wants to retain control of their infrastructure, i.e., blockchain
- **Scalability and reliability**: Transaction throughput, fault tolerance, contract security, disaster recovery, business continuity
- **Privacy vs. regulatory oversight**: Striking a balance between user privacy and compliance requirements (e.g., KYC, AML)
- **Vested interest**: CBDC presents a vastly different business landscape; existing financial institutions may be left out of equation
- **Competition**: Among central banks and from the cryptocurrency world

wCBDC

- **Inter-bank clearing**: Banks settle the obligations among themselves
  - Liquidity saving mechanism
- **Bond repo**: Central bank provides short-term liquidity for banks through loans collateralized by treasury bonds

rCBDC

- **Direct model**:
  - Customer uses rCBDC directly, like how cryptocurrency works
- **Two-tier model**:
  - Central bank – Bank – Customer
  - Customer uses rCBDC through their banking services
  - Problem: Is bank custodial or non-custodial?