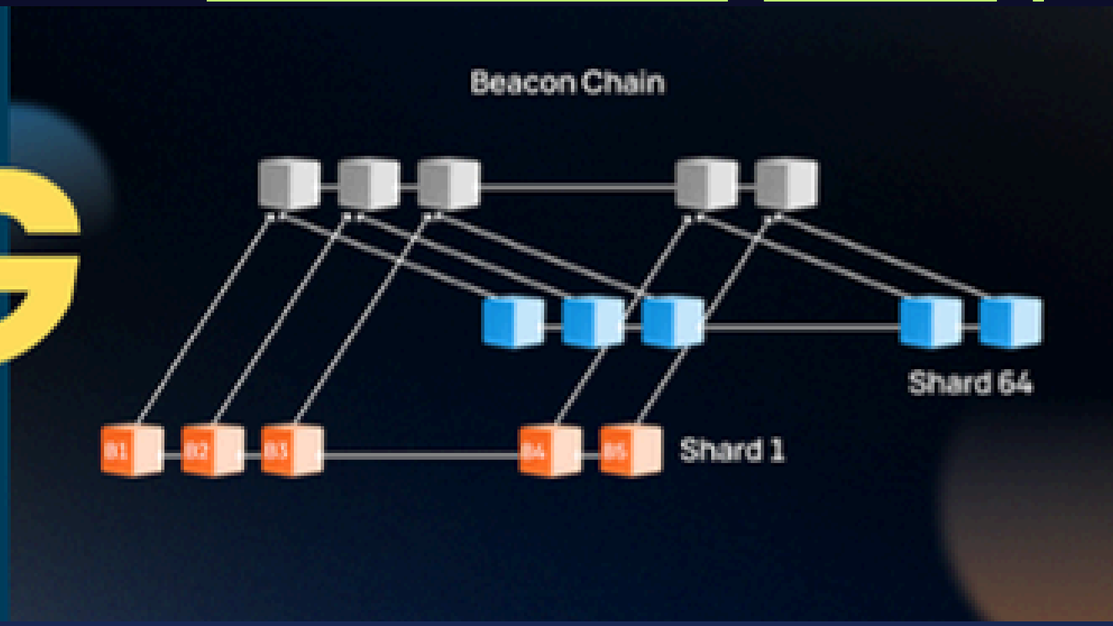
ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE (6-10)

SHARDING
& ETHDO TOOLS



Raja Rizwan Saleem
Lead Blockchain Trainer

Class-4

What is a Dapp?

A dapp is a software application that operates on a decentralized network, typically on a blockchain. Dapps utilizes peer-to-peer networks and smart contracts to function without intermediaries.

Features of dApp

Decentralization:

DApps are developed on distributed blockchain systems, where data and application logic are dispersed among several nodes. By doing away with a central authority or middleman, decentralization increases transparency, lowers the possibility of censorship, and removes single points of failure.

Open Source:

DApps are frequently open source, enabling the community to freely view and audit the code. The fact that anybody may review, contribute to, and suggest improvements to the codebase, encourages transparency, collaboration, and creativity.

Smart Contracts:

Smart contracts, which self-execute and have the conditions of the agreement written directly into the code, are used by DApps. On the blockchain, smart contracts make it possible to execute complicated logic, enforce corporate rules, and automate distrustless transactions.

Features of dApp

Tokenization:

Tokens are frequently used by DApps to symbolize wealth, ownership, or access privileges inside the application. Tokens can be built as custom tokens using smart contracts or as native tokens to the blockchain platform. Within the DApp ecosystem, tokenization provides incentives, rewards, and economic models.

Consensus Mechanisms:

DApps rely on consensus techniques to accomplish network-wide agreement and transaction validation. Proof of work (PoW), delegated proof of stake (DPoS), and other well-known consensus procedures are examples. Security, scalability, and energy efficiency are impacted by the consensus technique utilized.

Immutability and Auditability:

DApps use blockchain's immutability to make sure that once data is recorded, it cannot be changed or tampered with. This permits verifiable and transparent record-keeping and gives a trustworthy audit trail.

Features of dApp

Off-Chain Storage:

While the blockchain stores critical data and smart contracts, some dapps utilize off-chain storage solutions to store larger files, such as media or user-generated content. These off-chain storage systems may include decentralized storage platforms like IPFS (InterPlanetary File System) or traditional centralized cloud storage providers.

Peer-to-Peer Networking:

In some dapp architectures, peer-to-peer (P2P) networking protocols are employed for direct communication between users, bypassing the need for centralized servers. P2P networking can enable decentralized messaging, file sharing, and real-time interactions within the dapp.

1. Define the Problem and Solution

Identify the problem your DApp aims to solve and determine how blockchain technology and decentralization can provide a unique solution. For example, let's consider a supply chain management DApp. The problem could be inefficient tracking and verification of products. The answer would involve creating a transparent and immutable ledger for tracking product movement.

2. Choose the Blockchain Platform

Select a suitable blockchain platform that aligns with your DApp requirements. Some **blockchain platform examples:**

Ethereum: Use the Ethereum blockchain to create smart contracts and leverage its broad community support and ecosystem.

Hyperledger Fabric: Utilize Hyperledger Fabric, an enterprise-grade blockchain platform, for permissioned networks and privacy-focused supply chain solutions.

Polkadot: Explore Polkadot for interoperable and scalable blockchain solutions, enabling communication between multiple blockchains

3. Design and Develop Smart Contracts

Create and construct the smart contracts that are required for your DApp. Depending on the blockchain platform selected, a variety of programming languages can be used to create smart contracts:

Ethereum: Use Solidity, the most popular language for Ethereum smart contracts.

Hyperledger Fabric: Develop chain code (smart contracts) using Go, JavaScript, or other supported languages.

Polkadot: Leverage Substrate, a framework for building blockchain platforms, and its supported languages like Rust or C++.

4. Develop Front-End and Back-End Components

Create the user interface (UI) and user experience (UX) design for your DApp's front end. You can use different tech stacks for front-end and back-end development:

Front-end:

Web-based DApp: Use HTML, CSS, and JavaScript frameworks like React, Angular, or Vue.js.
Mobile DApp: Develop native applications using Swift for iOS or Kotlin for Android, or consider cross-platform frameworks like React Native or Flutter.

Back-end:

Ethereum: Interact with Ethereum nodes and smart contracts using web3.js or ethers.js libraries.
Hyperledger Fabric: Develop server-side components using programming languages like Node.js or Go, and utilize Hyperledger Fabric SDKs.
Polkadot: Leverage the Substrate framework and its supported back-end languages like Rust, JavaScript, or C++.

5. Test and Deploy

Test your DApp thoroughly to make sure it operates as intended and can handle a variety of situations. Test the front-end interactions, smart contracts, and general system behavior. Deploy your smart contracts to the selected blockchain platform after testing is finished. Launch your DApp at this point to make it accessible to users.

Remember that your specific requirements, scalability requirements, team knowledge, and resource availability may affect the technology stack you choose. It's crucial to investigate and assess the top frameworks and tools for your DApp development based on the blockchain platform you've chosen.

Tech Stack for Building a Dapp

Building a dapp, or decentralized application, requires a combination of technologies and tools.

Tech Stack for Building a Dapp

1. Blockchain Platforms

The choice of blockchain platform depends on the specific requirements of the dapp. Ethereum is widely preferred for dapp development due to its robust ecosystem, smart contract support, and large developer community. Other platforms like EOS, TRON, and Binance Smart Chain also offer dapp development capabilities.

2. Smart Contract Development

Smart contracts define the business logic of the dapp and are typically written in platform-specific programming languages. Solidity is the most popular language for Ethereum, while other platforms may have their languages like EOSIO's C++ or TRON's Solidity-compatible language. Development frameworks like Truffle or Hardhat facilitate smart contract compilation, testing, and deployment.

Tech Stack for Building a Dapp

3. User Interface (UI) Development

The dapp's user interface layer can be developed using web technologies, mobile frameworks, or desktop application frameworks. Some commonly used options include:

Web Development: HTML, CSS, and JavaScript frameworks like React, Angular, or Vue.js are popular for web-based dapps. Libraries such as Web3.js or ethers.js enable blockchain communication.

Mobile Development: Cross-platform frameworks like React Native or Flutter facilitate the development of mobile dapps. Mobile-specific libraries or frameworks enable interaction with the blockchain.

D**esktop Development:** Electron, a framework utilizing web technologies, allows the creation of cross-platform desktop apps.

Tech Stack for Building a Dapp

4. Middleware and APIs

Middleware components and APIs enable communication between the dapp's UI and the blockchain layer. Common tools and libraries include:

Web3.js: A JavaScript library for Ethereum interaction and smart contract communication.

ethers.js: Another JavaScript library providing a clean API for Ethereum interaction, including deployment, calling, and event handling of smart contracts.

IPFS: The InterPlanetary File System, a decentralized storage protocol, enables off-chain file storage and retrieval.

Oracles: Chainlink, Band Protocol, or custom-built oracles provide access to external data sources, enabling dapps to interact with real-world information.

Tech Stack for Building a Dapp

5. Development Tools and Testing

Development tools and testing frameworks ensure code quality and security. Common tools include:

Truffle: An Ethereum development framework simplifying smart contract compilation, deployment, and testing.

Ganache: A personal Ethereum blockchain for local development and smart contract testing.

Hardhat: A flexible development environment and testing framework for Ethereum smart contracts.

Solhint/Solium: Linters for Solidity that identify code issues and enforce best practices.

6. Backend Infrastructure

Additional backend infrastructure may be required based on dapp requirements. This can involve traditional server-side technologies or decentralized backend solutions:

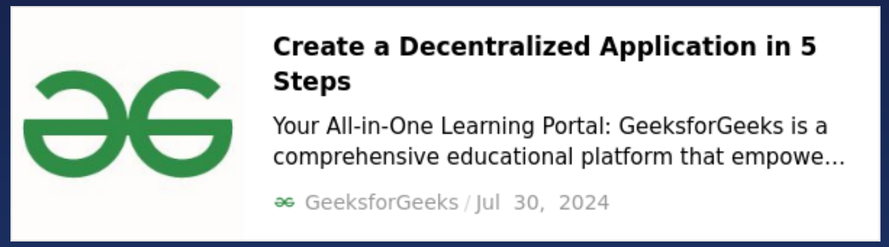
Traditional Backend: Node.js, Python, Ruby, or other backend frameworks can be used if centralized server-side processing or data storage is required.

Decentralized Backend: Decentralized solutions like IPFS for storage or OrbitDB for databases provide distributed data storage and retrieval capabilities

Tech Stack for Building a Dapp

7. Token Standards and Development

If the dapp incorporates its native token or cryptocurrency, adherence to specific token standards is necessary. For example, ERC-20 is commonly used for fungible tokens on Ethereum, while ERC-721 is prevalent for non-fungible tokens (NFTs).



When you’re peering curiously into blockchain...

“I think I want to build a decentralized app but can’t figure out WHY I actually should or HOW to do it.”

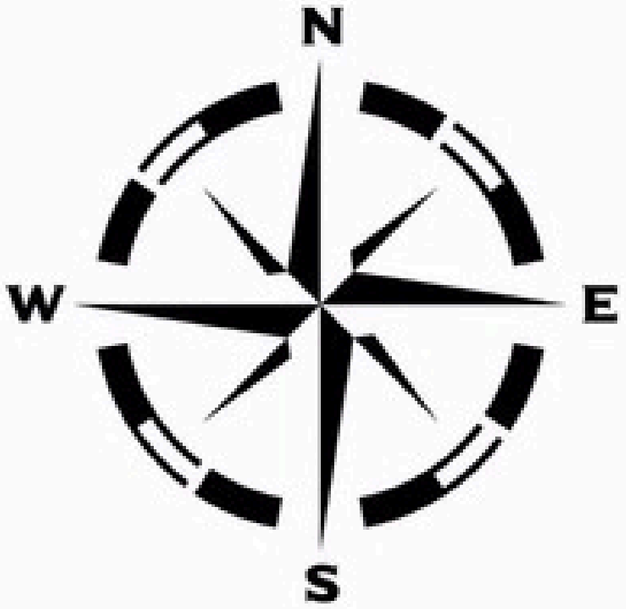
2 problems

1. It's really hard to find information on what the heck a "DApp" actually is and why anyone should really care. Just a bunch of crap about writing your ICO whitepaper.
2. Anything more detailed gets super technical and still doesn't provide concepts.

I'll address both of these today.

Roadmap

1. **About Decentralized Apps**
2. Blockchain Primitives
3. DApp Interactions
4. Case Study: CryptoKitties
5. DApp Technical Architecture
6. Further Considerations



What is a decentralized app (DApp)?

Requirements:

1. Running of the app (and its data) is distributed across lots of people/nodes
2. There is no single (central) point of failure… it's "unkillable"

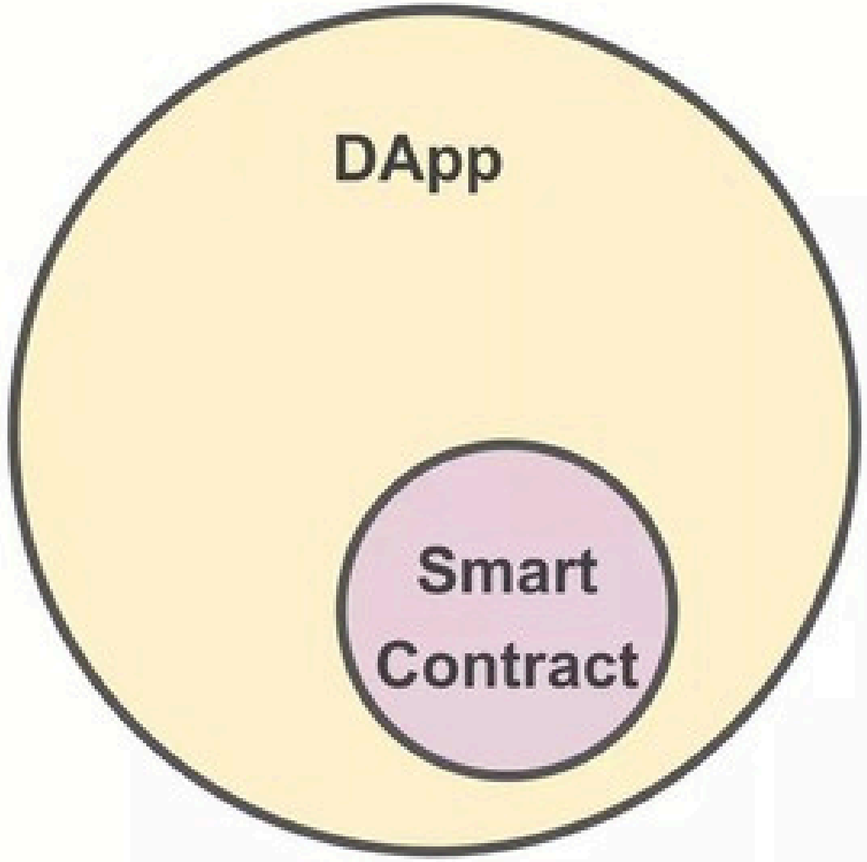
NOT requirements:

1. Having a token (or an ICO).
2. Being open source (though they *should* be for trust purposes)

Terminology: DApps vs smart contracts

The **DApp** is the full application, including its front-end markup, any non-blockchain code on the front end or a separate server AND the smart contract.

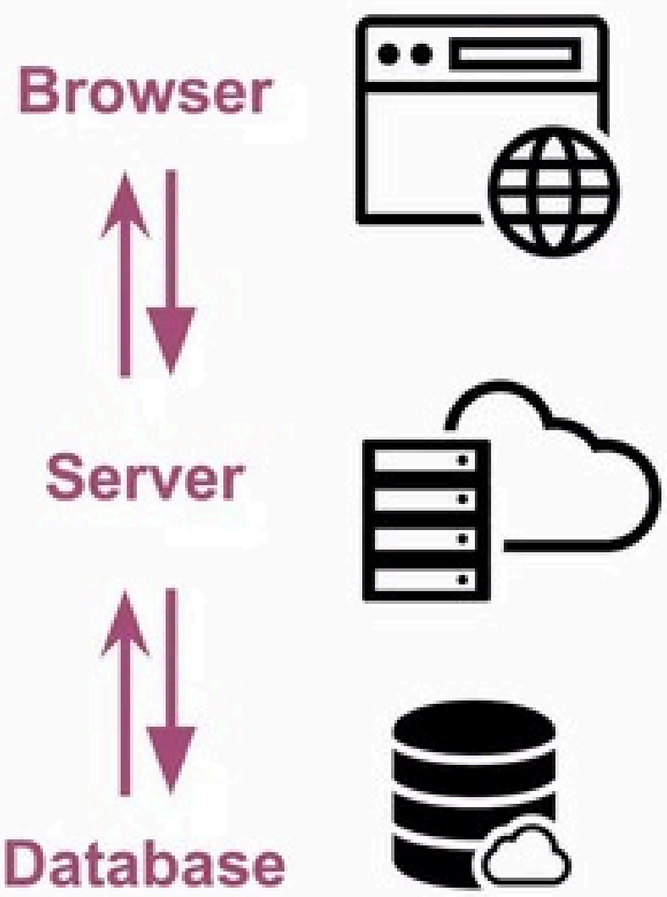
The **Smart Contract** is just the portion of the app that actually works with the blockchain.



Centralized applications of “Web 2.0” (today)

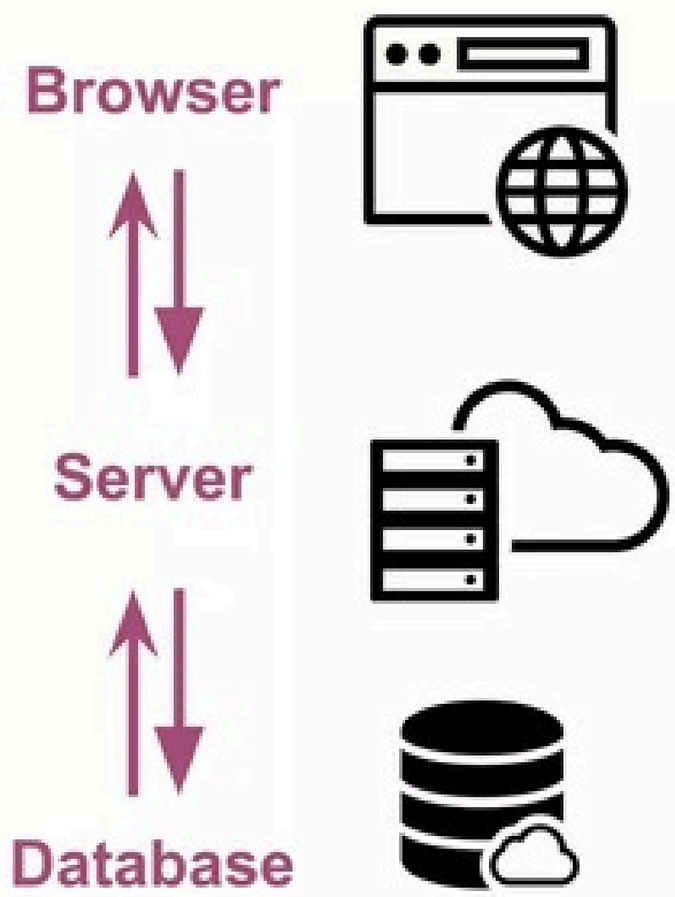
1. The front end (what you see) of the app is served from a server controlled by a particular developer or company. It’s probably cloud-hosted today.
2. Requests from front end are run on ONE server.
3. The data from that server and all your interactions with it flow to the company’s ONE database.

The server and/or database might be distributed across multiple data centers or the cloud but it’s still run once and **controlled by one individual or company.**



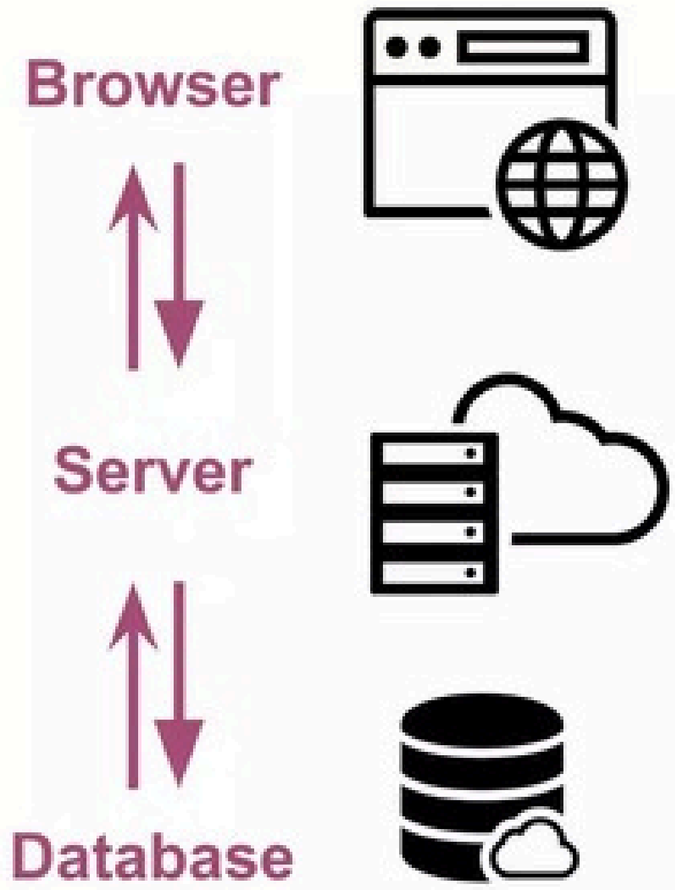
Advantages of centralization

1. **Experience Control:** The product roadmap and user experience can be tightly controlled.
2. **Clear Business Models:** We know exactly how to create business value by monetizing user data.
3. **Cost:** It's very low cost to host apps.
4. **Speed:** It's very fast to run apps.
5. **Iteration:** It's easy to iterate quickly and make changes quickly to stay ahead of technology or the market.
6. **Clear Moats:** User lock-in is well understood.

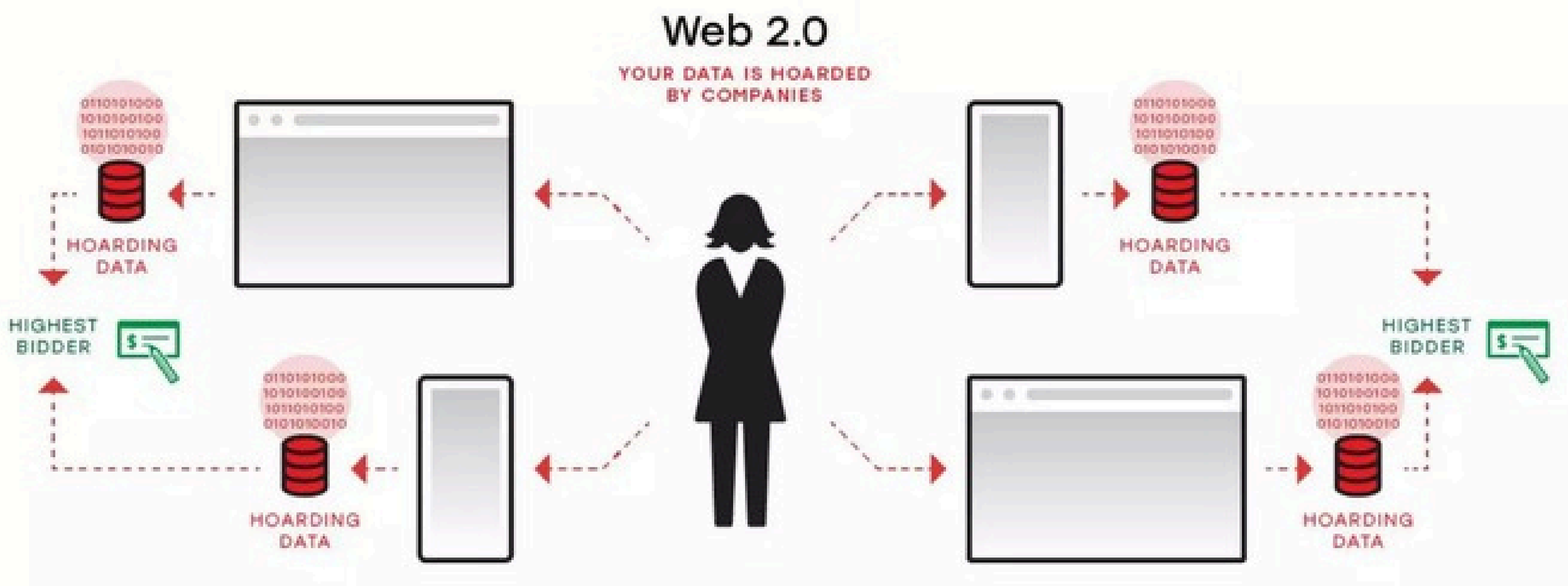


Downsides of centralization

1. **Censorship:**
   a. **Intentional:** eg Facebook kicking you or your app off their platform
   b. **Unintended:** eg Government requiring Facebook to turn over their data on you or ceasing operations
2. **Hacking:** Central data is a constant target
3. **Data Sales:** The company gets purchased or goes bankrupt and your data is now in the hands of the highest bidder.



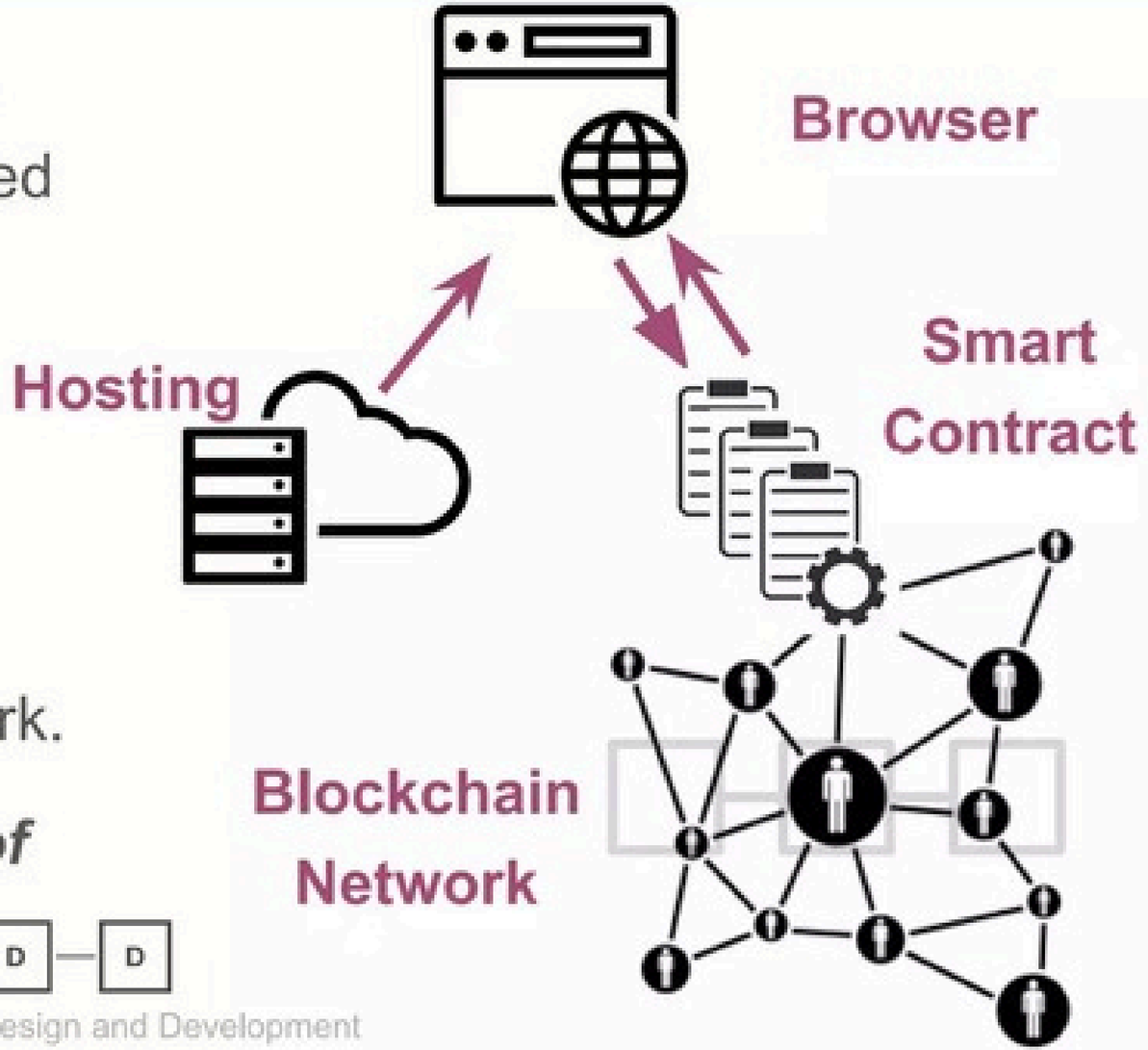
Centralized apps lead to data hoarding and lock-in



Decentralized applications of “Web 3.0”

1. The application still needs the same front-end stuff (HTML/CSS/JS) served from somewhere, eg. a cloud or a static hosting site or P2P network.
2. The front end talks to the **Smart Contract** using its API (via wallet).
3. The Smart Contract runs code and stores data on the blockchain network.

*This data is generally stored **on behalf of the user**.*



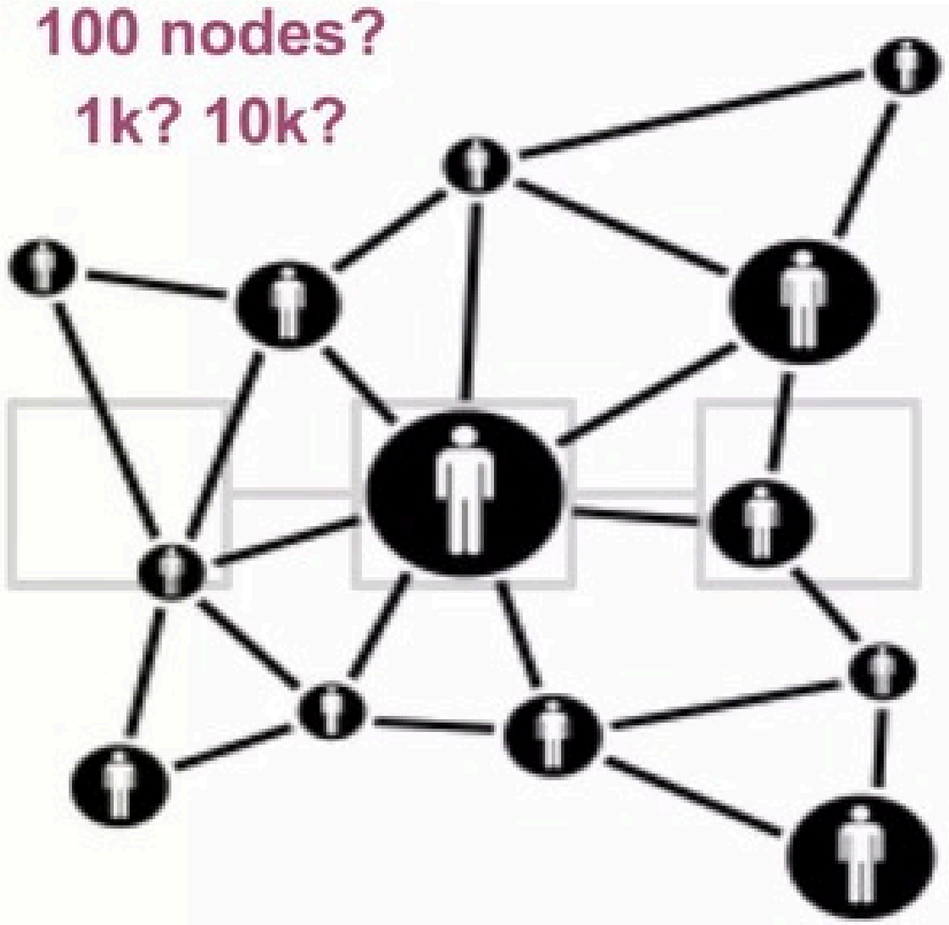
Blockchains are peer-to-peer *networks*

Most diagrams get this wrong!

The key feature is that EVERY node that makes up the network runs the smart contract and stores the blockchain's data.

These nodes, including your app, agree on the result to return to the frontend.

This gives us **decentralization.**



Advantages of decentralization

1. **No Censorship:** Data and code execution are spread out over lots of nodes so there is no single point to censor or fail.
2. **No Platform Risk:** A special case of censorship, no platform (like Facebook) can shut down your app because it became too successful or controversial.
3. **Transparency:** Everyone can see what code will run so it's verifiably fair and accurate.

Problems with decentralizing your app

1. **Cost:** Because computation runs on every node, by definition it is at least as costly as the number of nodes in the network (eg 100x amazon AWS).
2. **Time:** Since multiple nodes have to first run the computation (fast) and then come to a consensus about its result (slow), it's much slower than central servers.
3. **Services:** How do you provide services to users?
4. **Privacy:** Right to be forgotten.

"You can compete on any 2"

...unless you're decentralized.

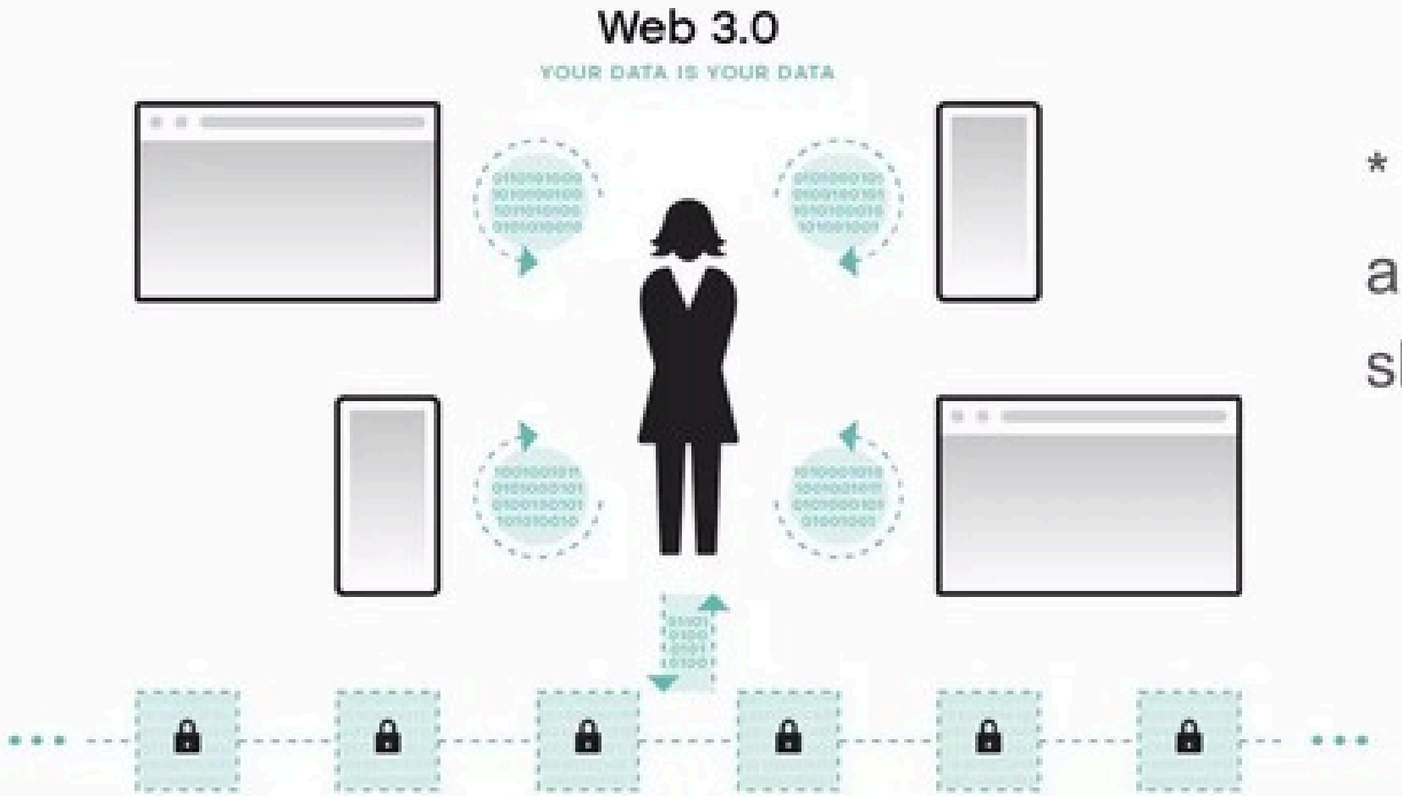
So what apps are worth the tradeoffs?

1. **Money:** Bitcoin's store of value and Ripple's transactions are still really the only killer apps.
2. **Black/gray markets:** There literally isn't any other way to allow people to transact.
3. **Censored systems:** Governments prevent people from using normal means to communicate, collaborate or transact so they have to go p2p.
4. **And…?** We'll look at primitives shortly.

"You can compete on any 2"

...unless you're decentralized.

Decentralized apps give the user more control



* We'll talk about wallets shortly...

Blockchain DApps are getting more sophisticated

1. **Money Transfer:** Bitcoin is the decentralized network and its transfers act like a basic DApp.
2. **Programmable Money:** Adding logic to the transfers and executing basic code is a DApp.
3. **Turing Complete Smart Contracts:** The blockchain runs a virtual computer that can process anything.

Blockchain primitives (aka “what it gives you for free”)

1. **Identity:** Everyone gets an account which gives them a bucket of stuff to hold onto, sort of like a safety deposit box that lives on the chain.
2. **Transactions:** The idea of transferring tangible and intangible goods is fundamental to blockchains.
3. **Cryptography:** Chains only exist because of cryptography and they expose it to anyone’s use.

Why should I build on blockchain?

“So decentralized apps have a sort of abstract protection against censorship but are slower and more expensive than centralized ones... how do they help me as a developer?”

Blockchain IDENTITY primitives

Everything on the chain gets assigned to an account.

1. **Accounts:** Accounts can represent a person, company, app or even a thing (eg refrigerator).
2. **Ownership:** That account has total control of its money, its digital goods and its data (which can represent real-world things like identification)
3. **Login/Reputation:** Every account's transaction history gives it reputation which you can tap into.

In today's web, developers have to custom code these.

Blockchain TRANSACTION primitives

Everything on the chain gets assigned to an account.

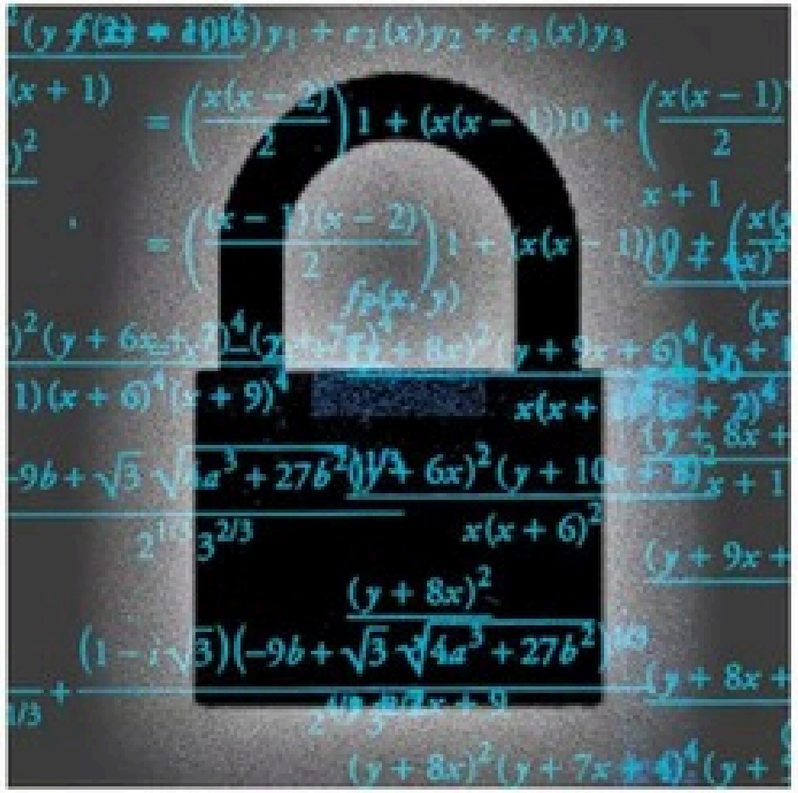
1. **Direct:** Anything can transfer directly between accounts without a central authority (allowing peer-to-peer marketplaces or transfers)
2. **Instant:** Financial and digital-good transactions have near-instant finality.
3. **Micro:** Negligible fees make high frequency and/or small amounts okay.
4. **Conditional:** It's easy to add logic like escrow.

Blockchain CRYPTOGRAPHY primitives

Cryptographic primitives give anyone easy ability to:

1. **Verify Activity:** Did you do a specific thing or have specific data on a specific date? Eg supply chains, ad activity, transfers.
2. **Verify Process Integrity:** Is this “random” algorithm actually fair?

This is because it’s easy for anyone to use their account’s private key to “sign” transactions or data.



Blockchain makes a number of things easy… sort of.

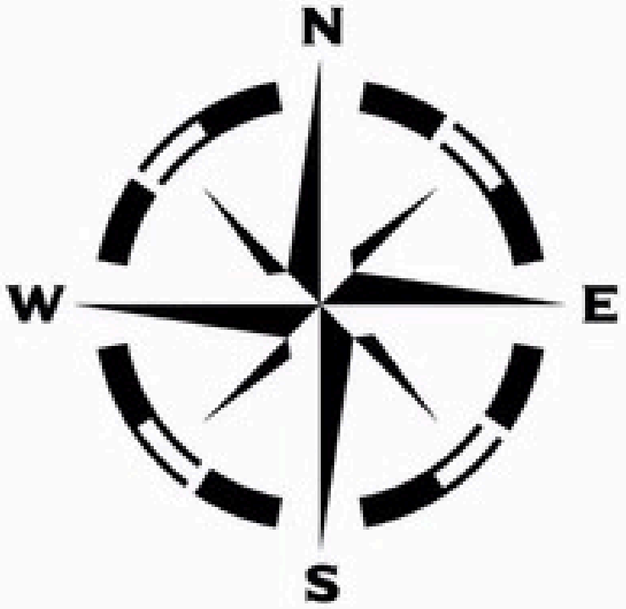
- ✓ Need P2P transactions?
- ✓ Need to handle money transfers?
- ✓ Need simple verifiability of actions?
- ✓ Need to give users digital stuff?

Instant transactions? Escrow?

...and so on.

Roadmap

1. About Decentralized Apps
2. Blockchain Primitives
3. **DApp Interactions**
4. Case Study: CryptoKitties
5. DApp Technical Architecture
6. Further Considerations



Decentralized app interactions require permission

Because users control their data (which includes their money, identity, etc), we have to ask them for permission before accessing/editing it.

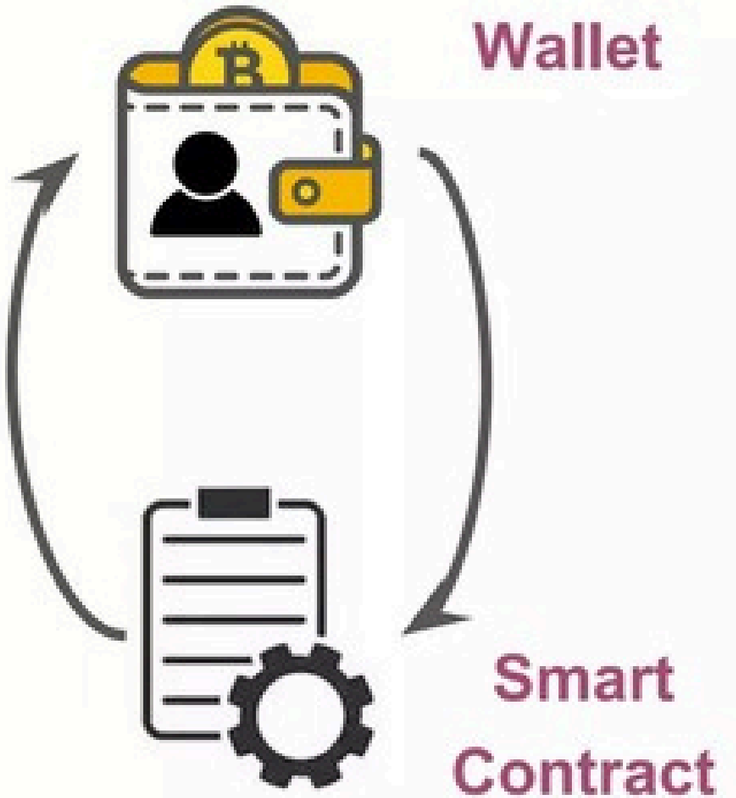
For example, if we run a decentralized marketplace that facilitates peer-to-peer trades of online trading cards, if there is a match we need to ask each party for them to sign the transaction that completes the trade.

The wallet allows this

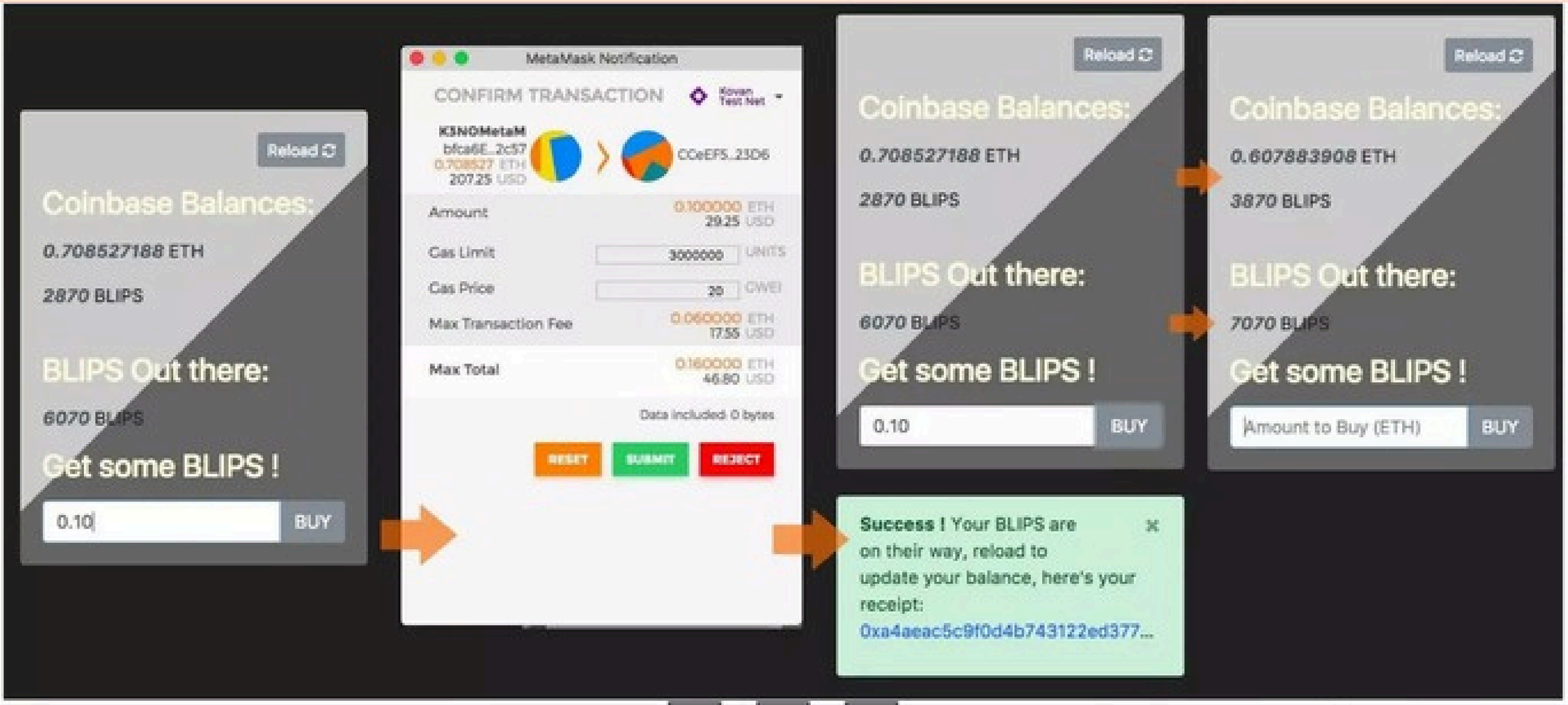
A user's **"Wallet"** contains the cryptographic key which allows them to sign (verify permission for) transactions for a particular account (or accounts).

The wallet typically then **propagates that transaction** to the blockchain network (usually by either running its own node or using a trusted node.

...So every time we have a transaction that needs permission, it must go through a wallet.



How a wallet transaction flow looks



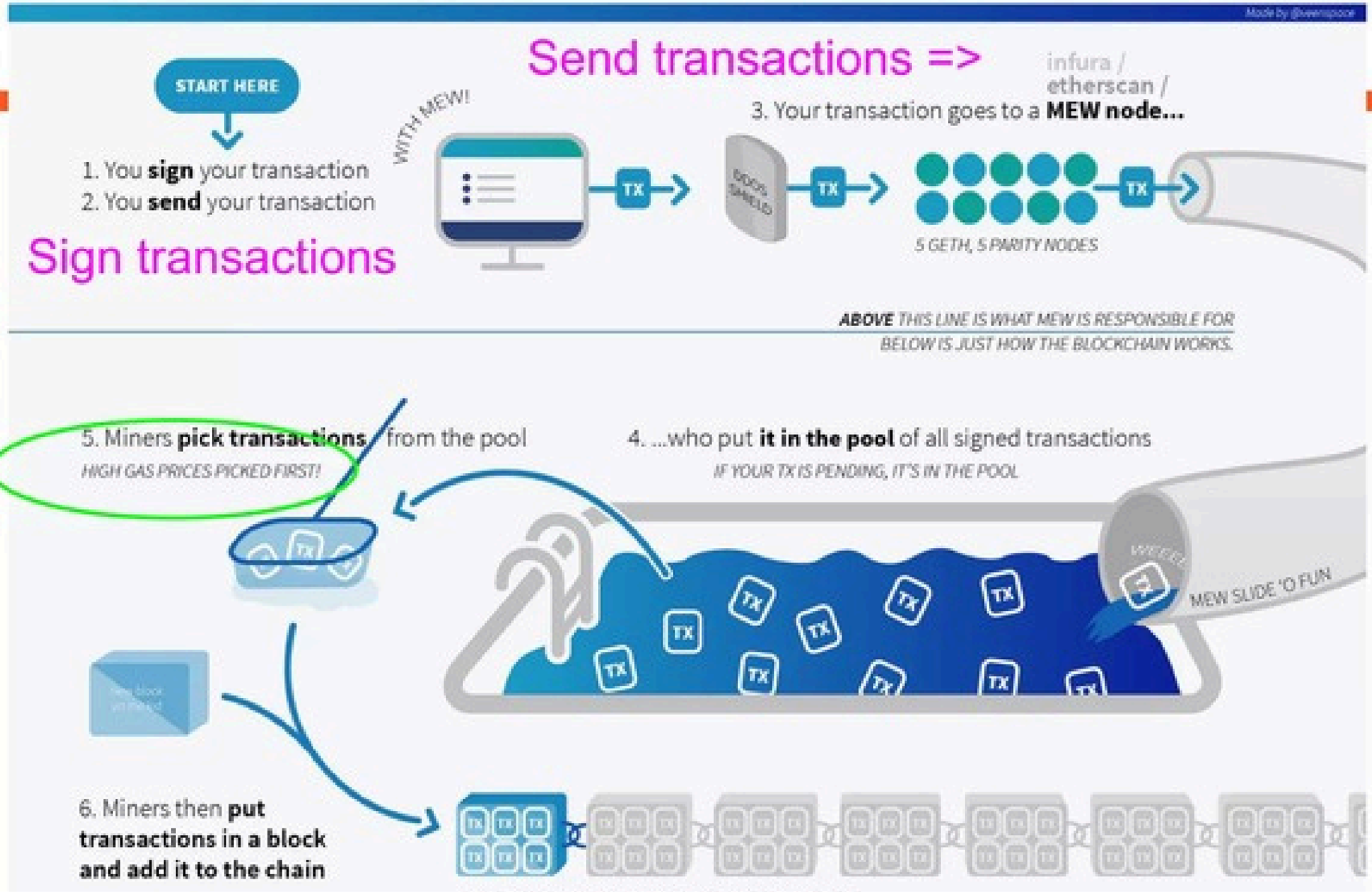
Wallets unlock accounts not just money

Remember -- accounts contain ANYTHING.

The wallet could contain token balances but also crypto collectibles, identity data or anything else that's linked to your account ID.

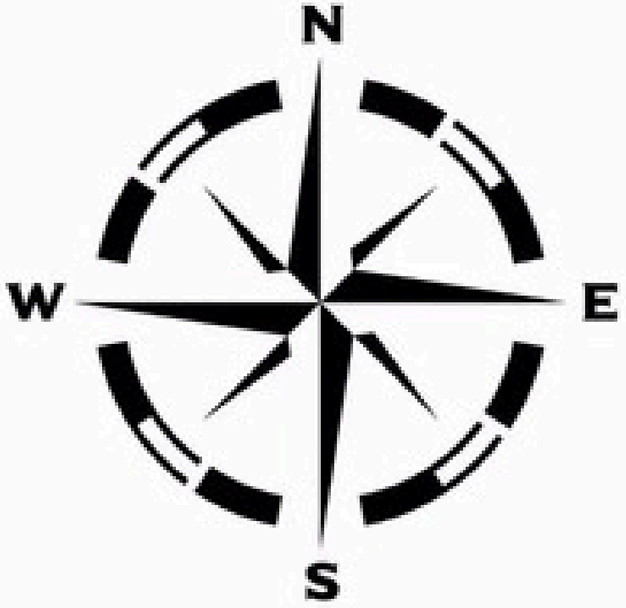
Any time we want to get something unique from the user, we need their permission to access that data from the blockchain on their behalf.

MyEtherWallet Behind-The-Scenes



Roadmap

1. About Decentralized Apps
2. Blockchain Primitives
3. DApp Interactions
4. **Case Study: CryptoKitties**
5. DApp Technical Architecture
6. Further Considerations

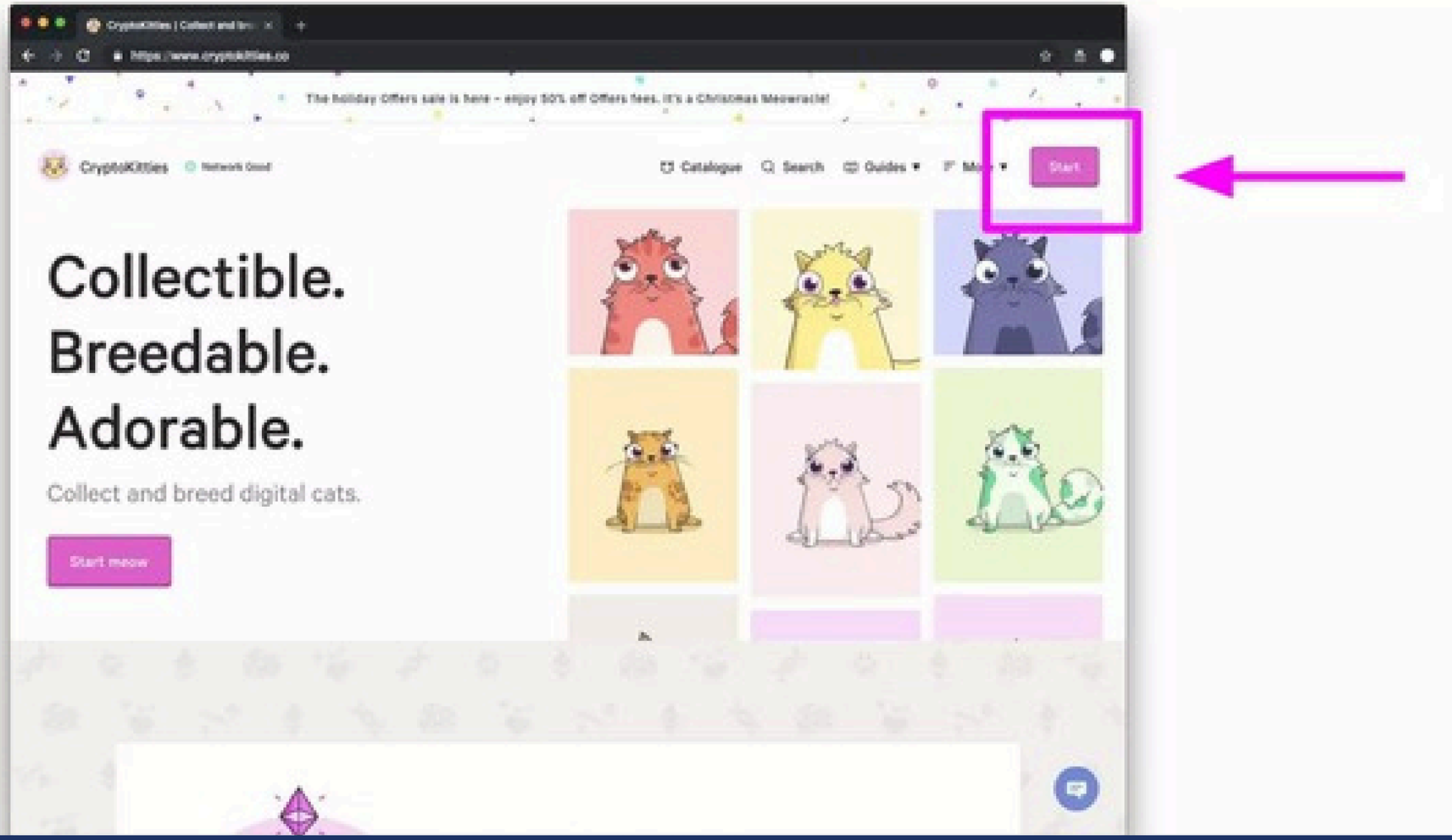


Case study: CryptoKitties UI

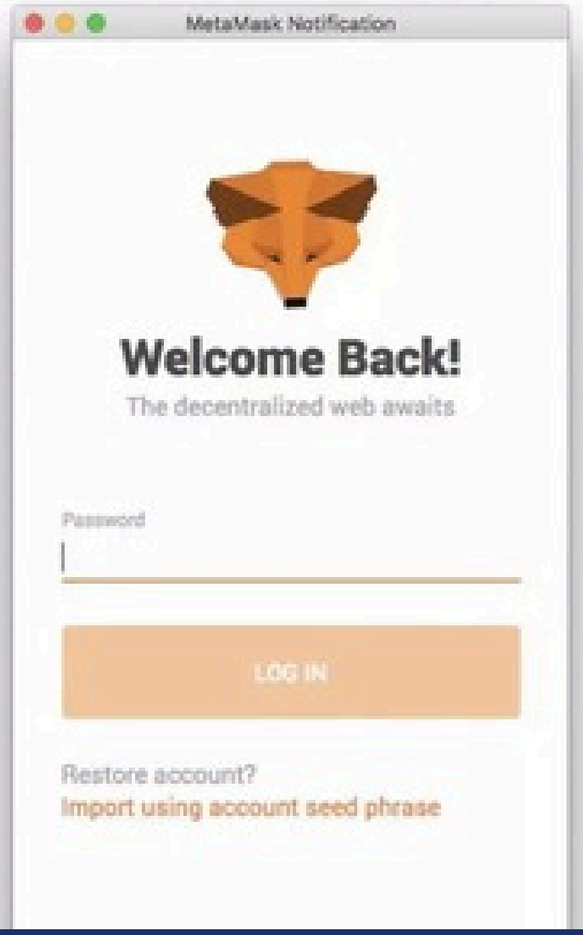
What is CryptoKitties?

CryptoKitties is a game centered around breedable, collectible, and oh-so-adorable creatures we call CryptoKitties! Each cat is one-of-a-kind and 100% owned by you; it cannot be replicated, taken away, or destroyed.

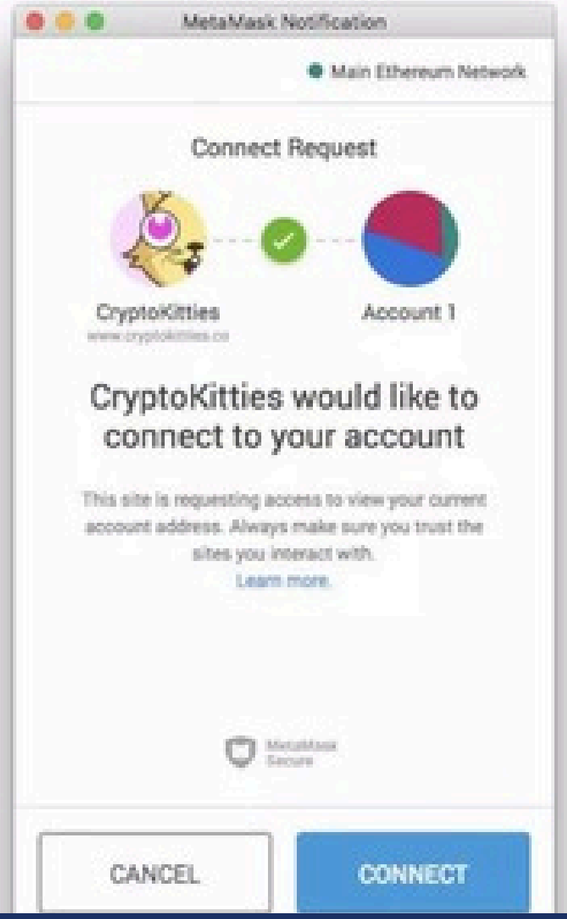
Case study: CryptoKitties UI



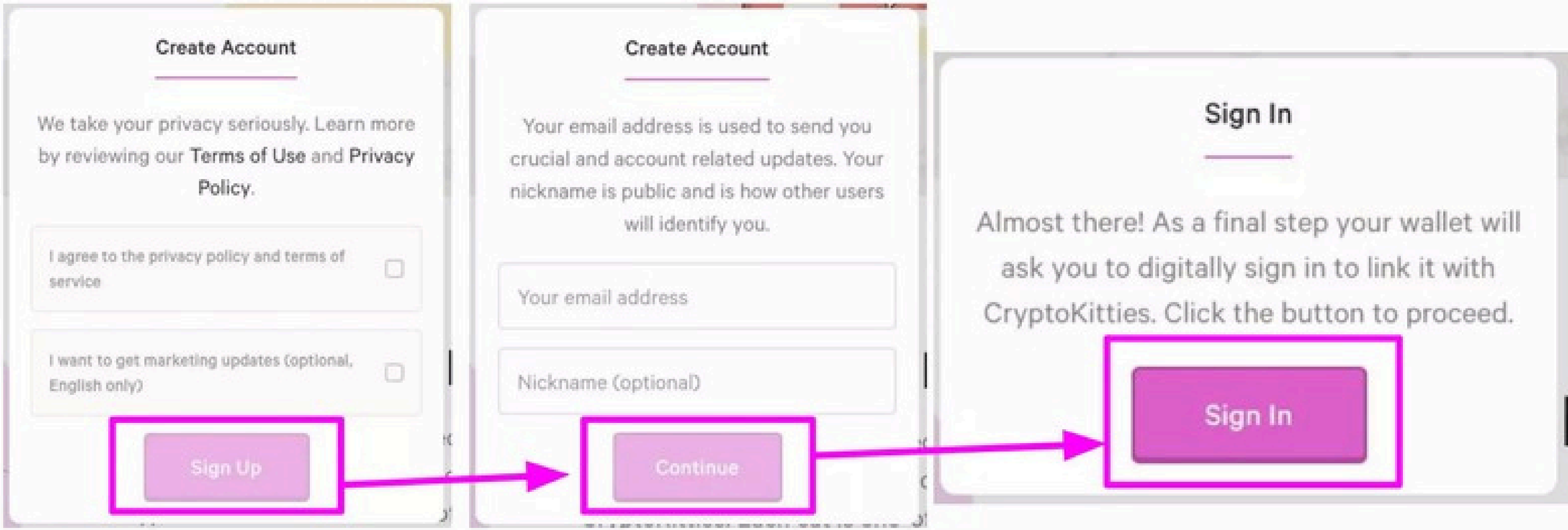
Case study: CryptoKitties UI



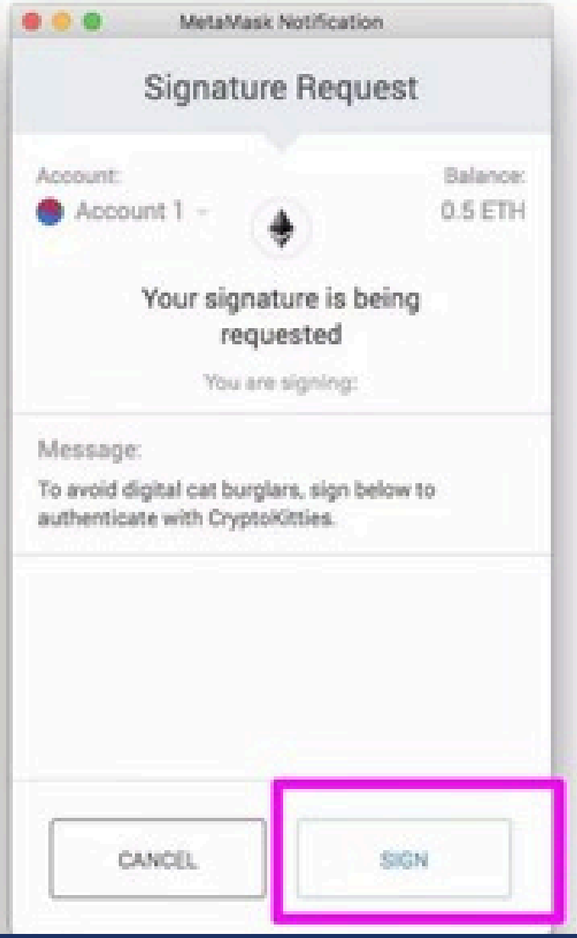
Immediately asks me for my MetaMask just to get started.



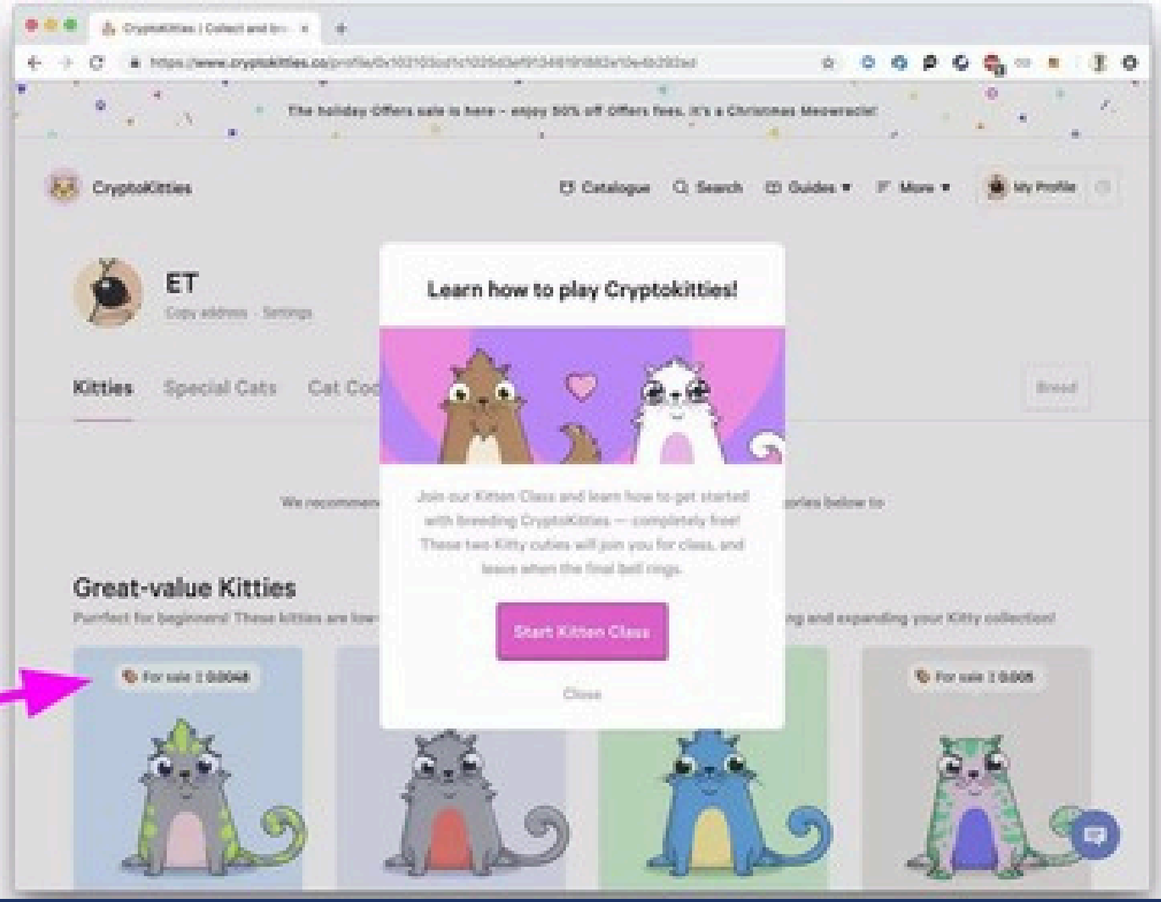
Case study: CryptoKitties UI



Case study: CryptoKitties UI

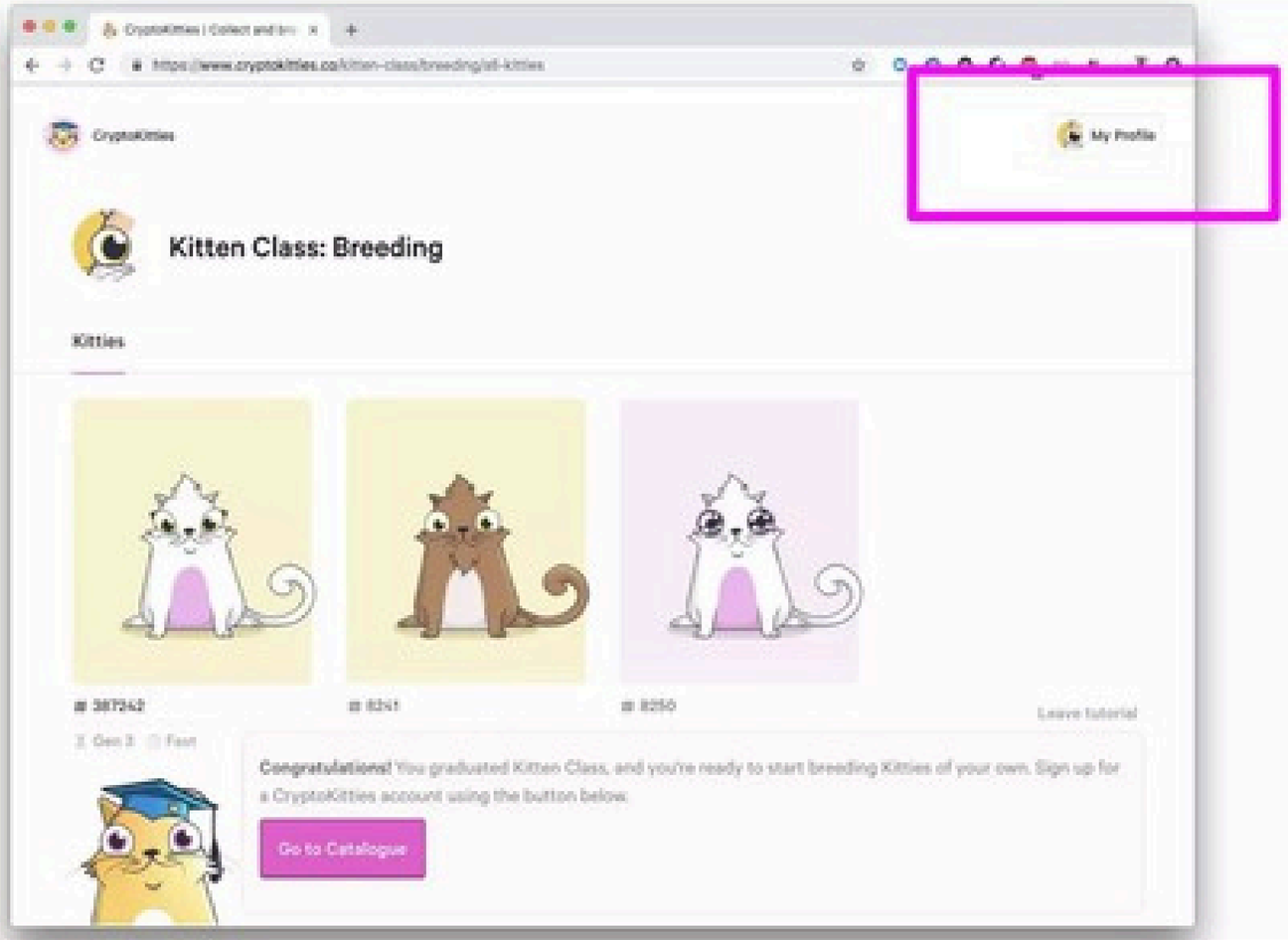


Create the Kitties account and link to my wallet account.

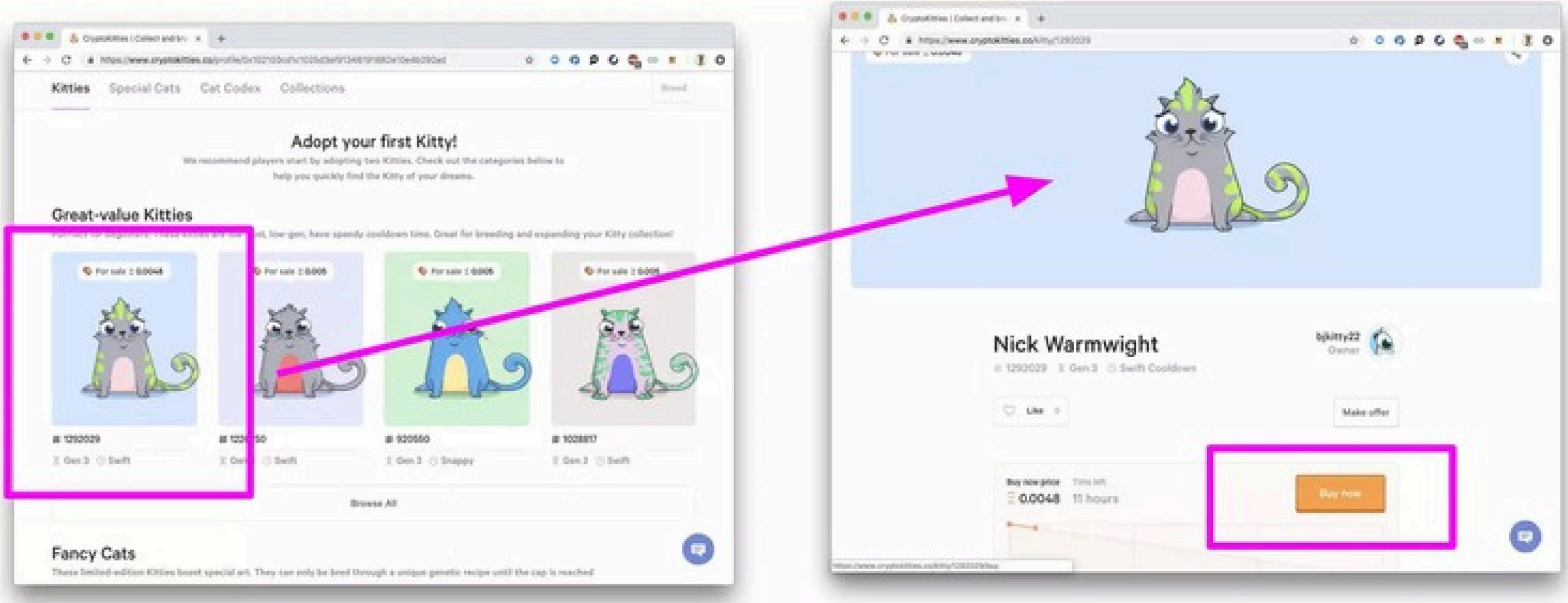


Case study: CryptoKitties UI

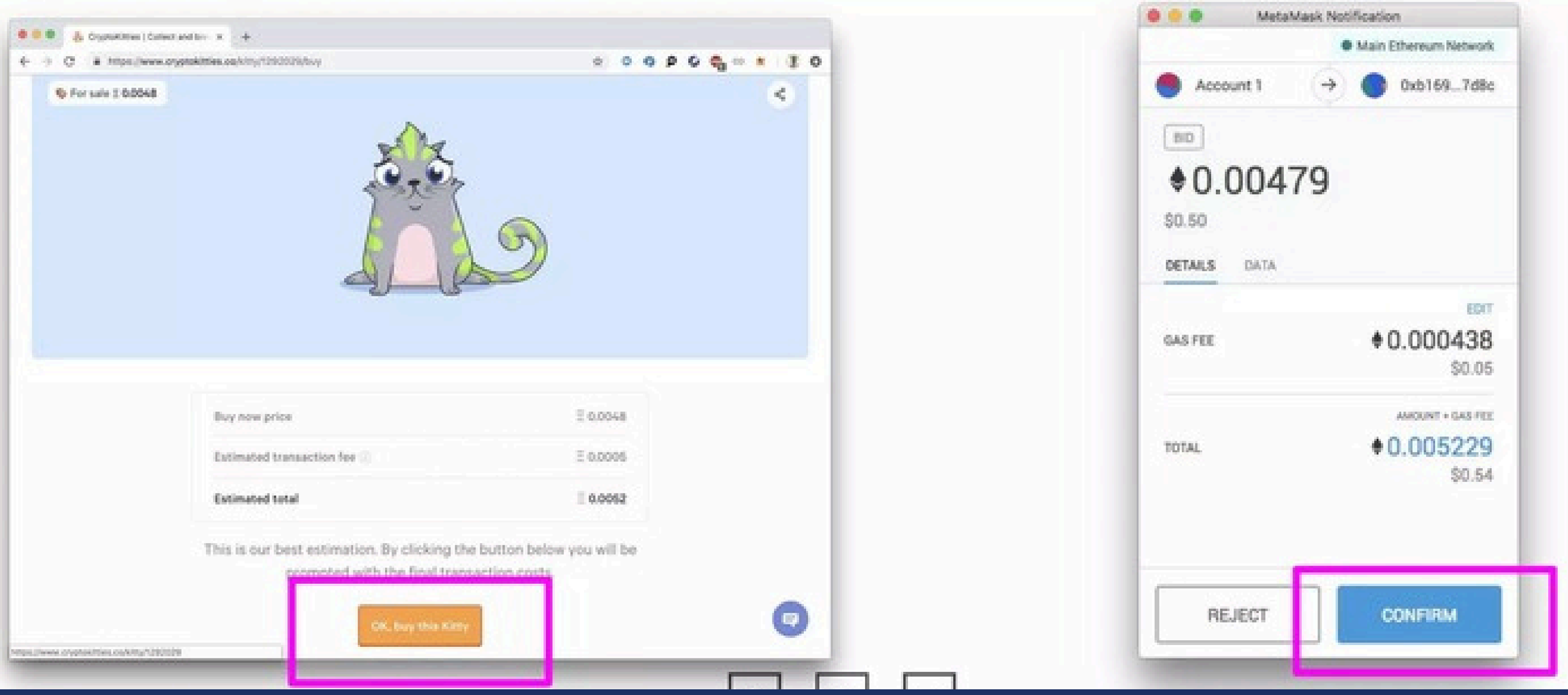
Tutorial. No chain writes required.



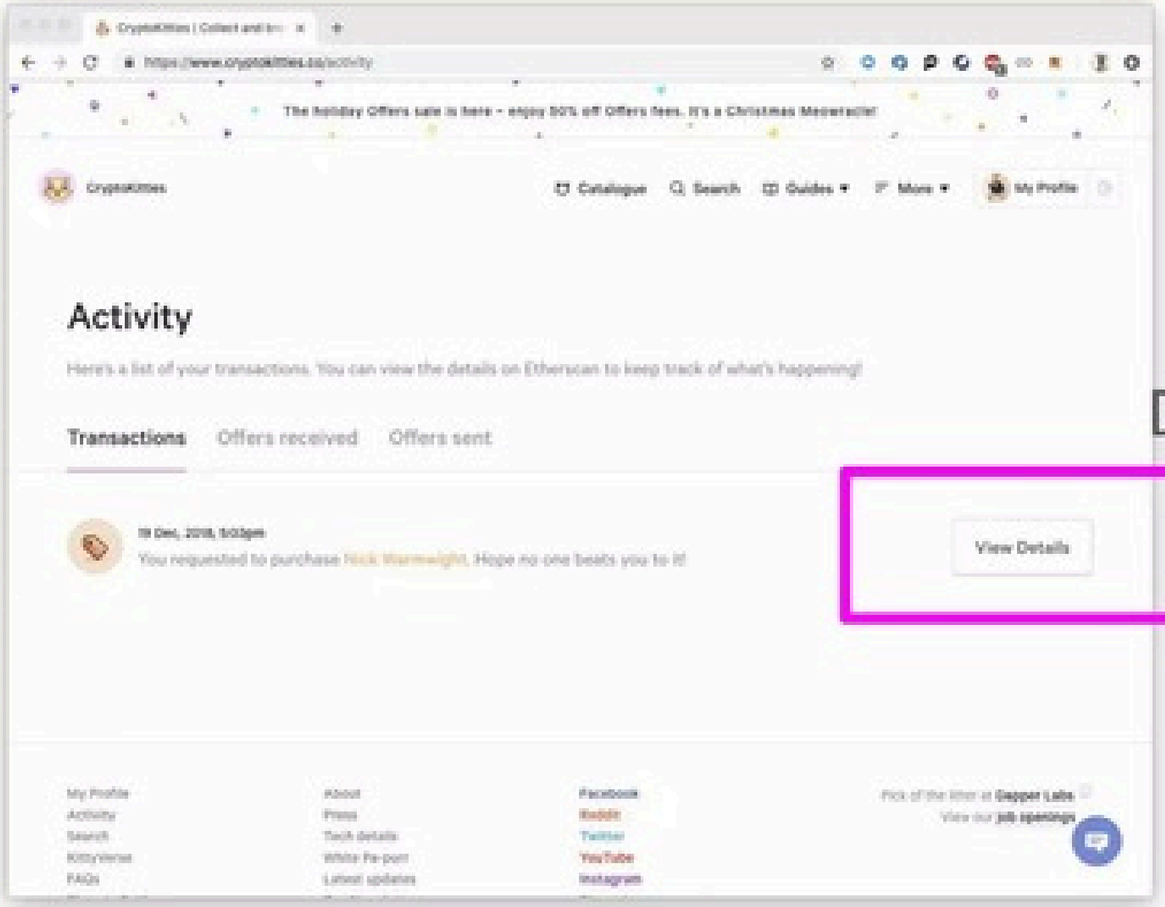
Case study: CryptoKitties UI



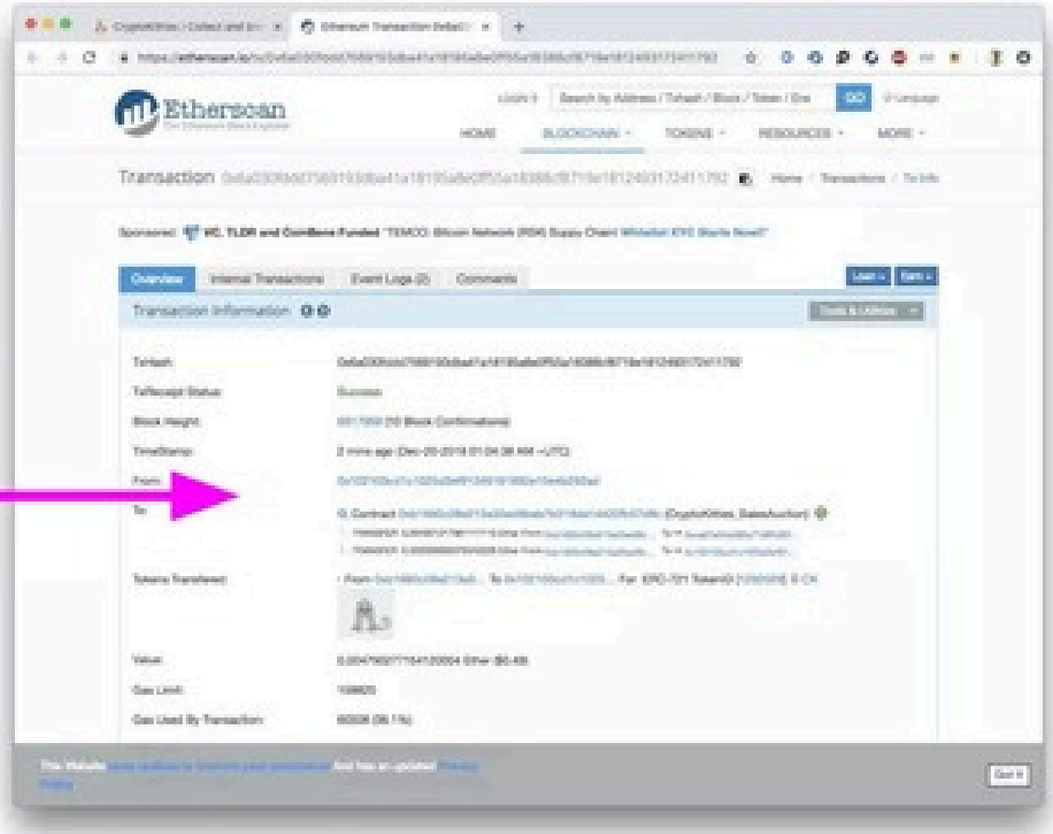
Case study: CryptoKitties UI



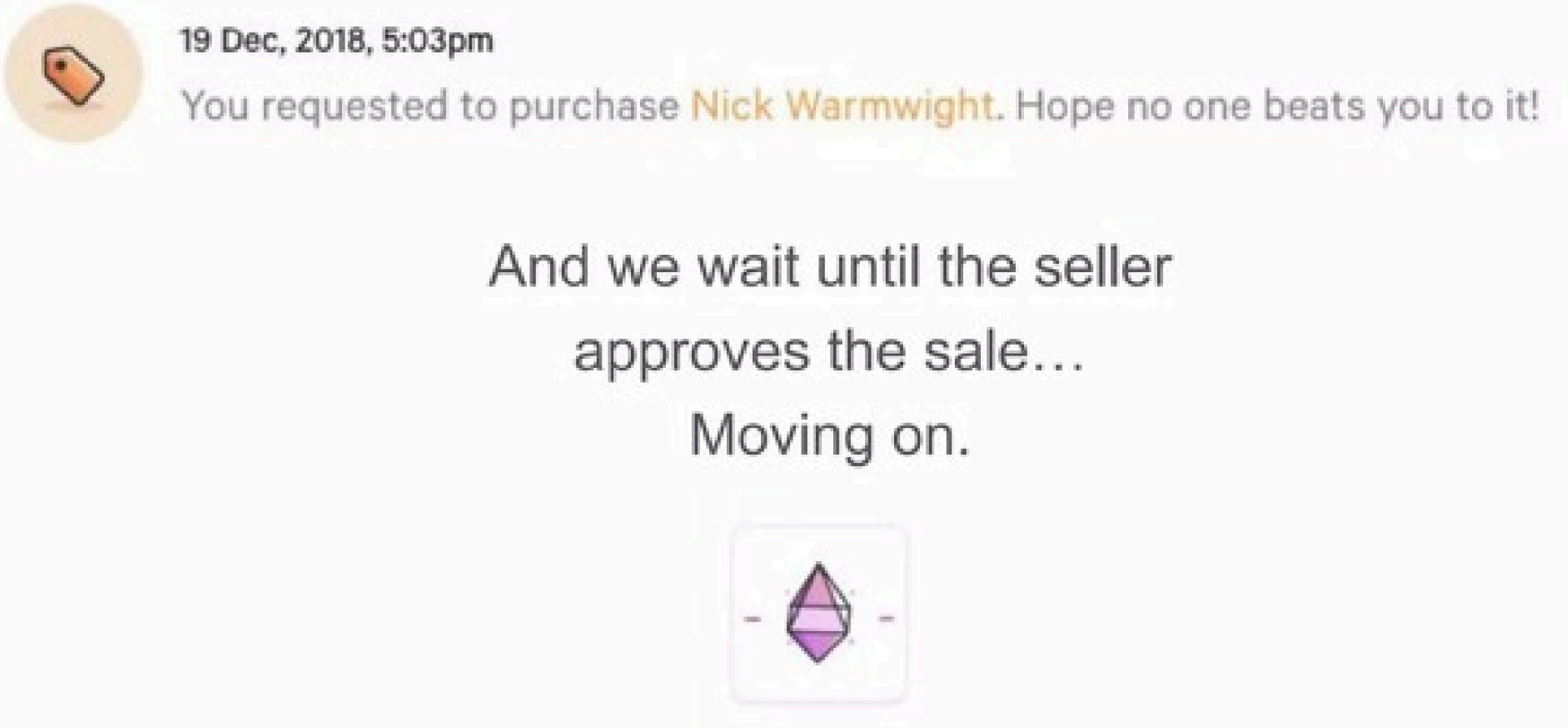
Case study: CryptoKitties UI



Details

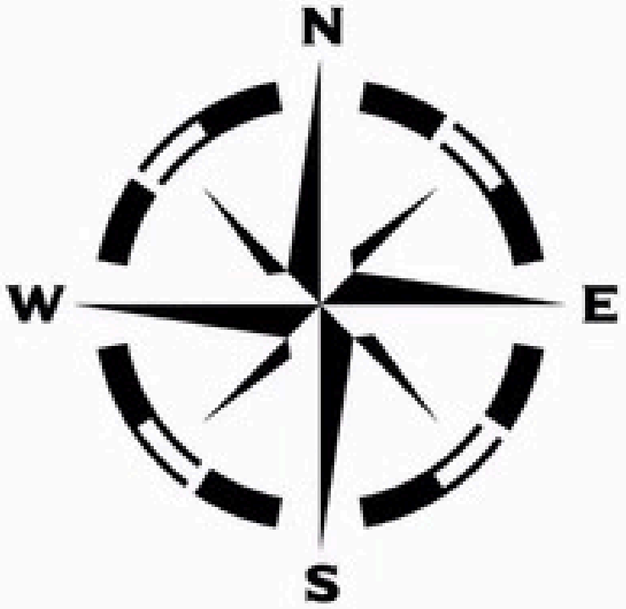


Case study: CryptoKitties UI



Roadmap

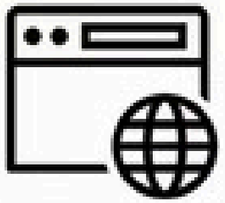
1. About Decentralized Apps
2. Blockchain Primitives
3. DApp Interactions
4. Case Study: CryptoKitties
5. **DApp Technical Architecture**
6. Further Considerations



Building a DApp

1. **Front-End:** Build and deploy a normal HTML / CSS / JS front end. You'll need app-specific JavaScript.
2. **Library:** Add the code that allows your frontend to connect to the wallet and blockchain network.
   a. The user's third-party wallet will likely connect to the network and sent transactions for you.
3. **Smart Contract:** Write the smart contract(s) that perform the app's core functions, including anything that modifies the user's wallet "contents."
4. **Deploy** the smart contract to the blockchain.

Frontend

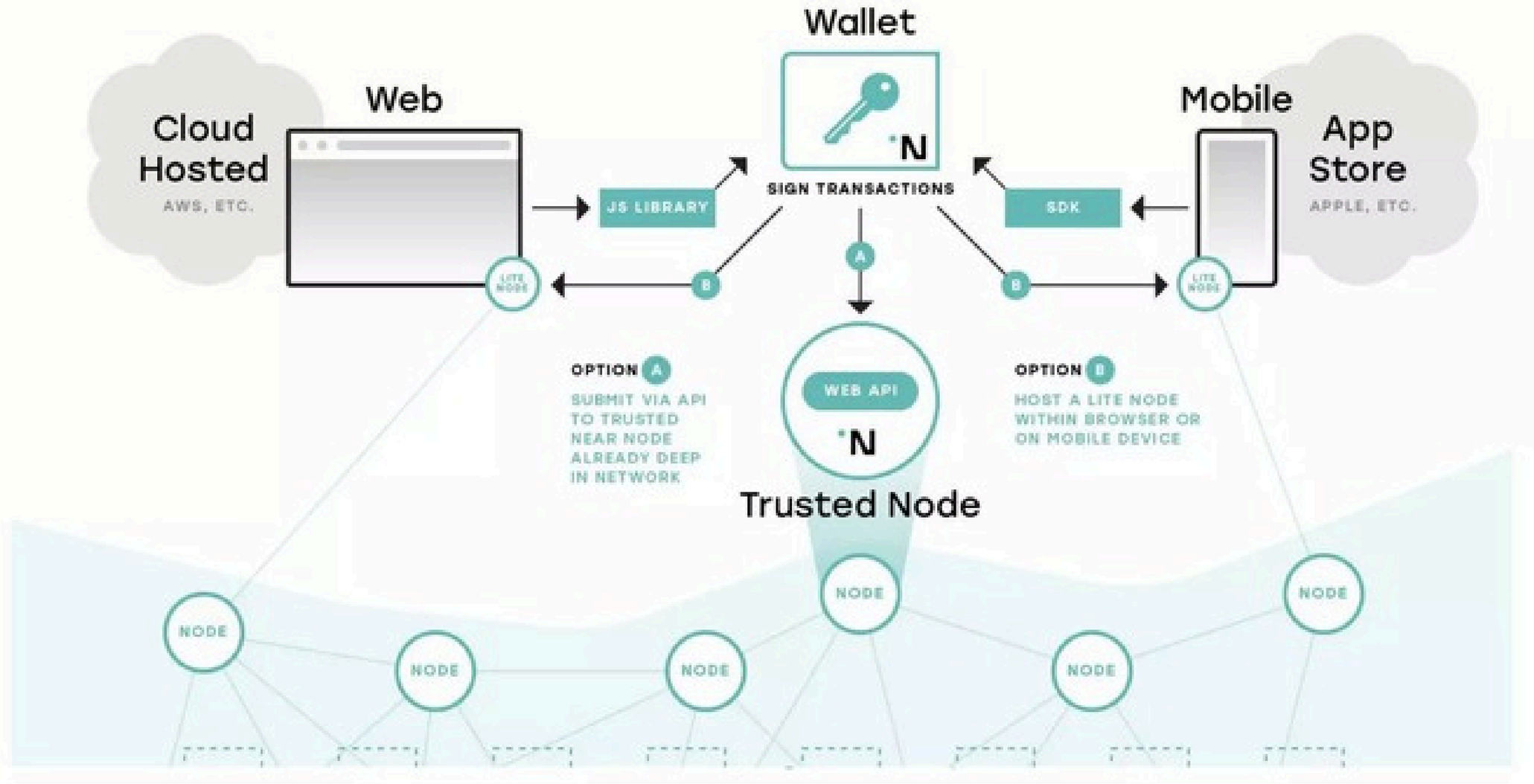


JS Library

Smart Contract

Deployed to Blockchain

High level flows of a web or mobile DApp

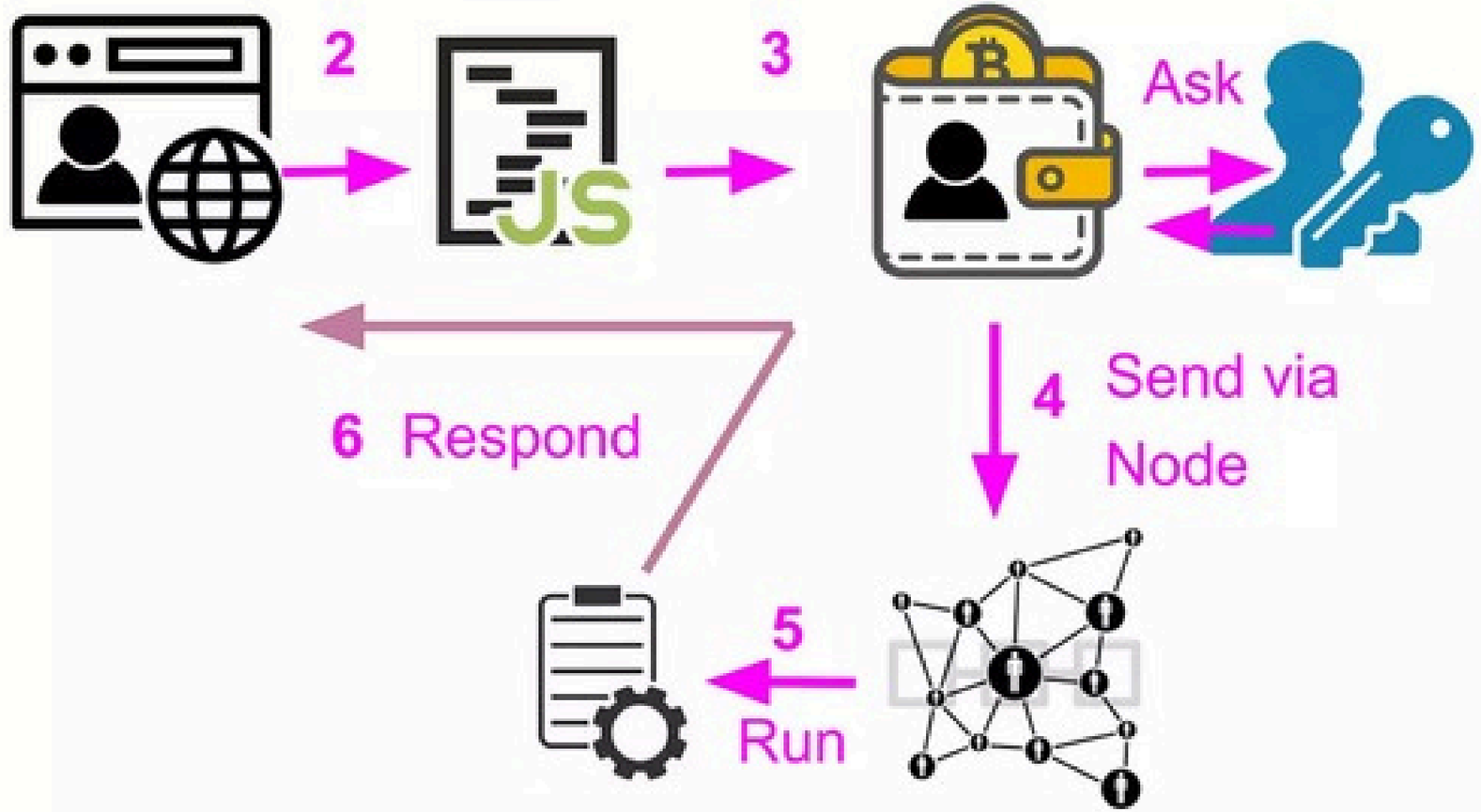


The lifecycle of a user's transaction

1. **Interact:** Your user interacts with the page and says "do this thing".
2. **Wallet Check:** Your frontend's JavaScript code checks for a wallet and sends it the intended transaction ("Tx").
3. **Get Permission:** Wallet asks the user for permission and signs the Tx.
4. **Send Tx:** The wallet's sends the signed Tx to the blockchain network (with a local node, trusted node)
5. **Run Smart Contract:** The network runs the right smart contract code.
6. **Respond:** The network returns the response.

The lifecycle of a user's transaction (diagram)

1. Interact
2. Wallet Check
3. Get Permission
4. Send Tx
5. Run Smart Contract
6. Respond



What you'll pay attention to

As a DApp developer, you'll probably pay most of your attention to:

1. **Smart Contract:** Writing the smart contract in a secure fashion can be very challenging.
2. **JavaScript:** The code on your front end needs to understand what requests can be sent to the smart contract and how to do so. Once you've defined your smart contract interface you can code against it in your frontend (eg React) app.
3. **User Interface:** The user flows can be challenging to get right.

Aside: Smart contracts can talk to other smart contracts

Each smart contract is given an account of its own which is externally indistinguishable from that of a user. It is identified by a long hash.

...except, of course, you can make function calls to this “account” and the contract that lives there will execute them if properly formatted, just like an API.

Case study: Cryptokitties smart contract architecture

Multiple smart contracts segregate individual functions:

KittyCore: Ties it all together via inheritance.

KittyAccessControl: Decides meta things like contract governance.

KittyBase: All the data structures we need to define and track kitties.

KittyOwnership: The ERC721 spec to define non-fungible kitties.

KittyBreeding: Logic for how to breed kitties. GeneScience != OSS

KittyAuctions: How to buy, sell and auction siring rights.

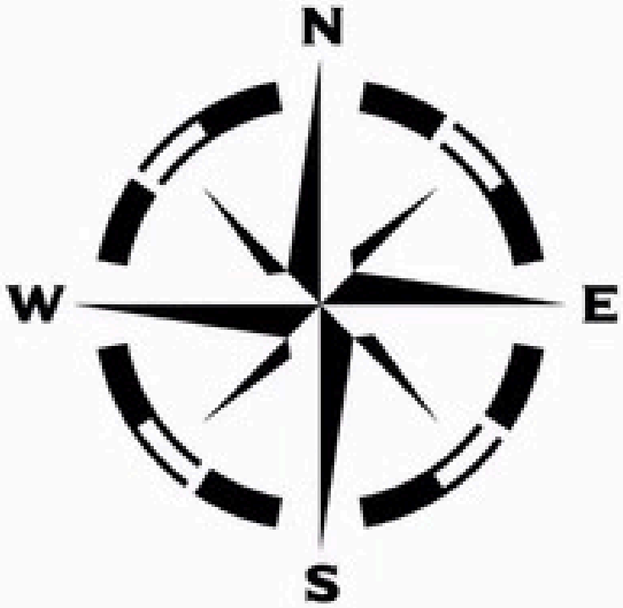
KittyMinting: Creating new generation-0 kitties.

Case study: Cryptokitties architectural notes

1. The display logic for kitties is on their server. Kitties are defined by a genetic code which you own but the UI is theirs.
2. Their website is served from some sort of static host, not the blockchain.
3. They have essentially hard-coded the use of MetaMask, which is a third-party wallet provider for Ethereum.
4. Their code is available at: https://ethfiddle.com/09YbyJRfil

Roadmap

1. About Decentralized Apps
2. Blockchain Primitives
3. DApp Interactions
4. Case Study: CryptoKitties
5. DApp Technical Architecture
6. **Further Considerations**



Let’s talk about gas

In Bitcoin, each block has a maximum size which is determined in bytes to allow fast propagation. That determines how many transactions each block can fit.

In smart contract platforms, a more appropriate measure is how much computation a given transaction actually requires the miner to perform.

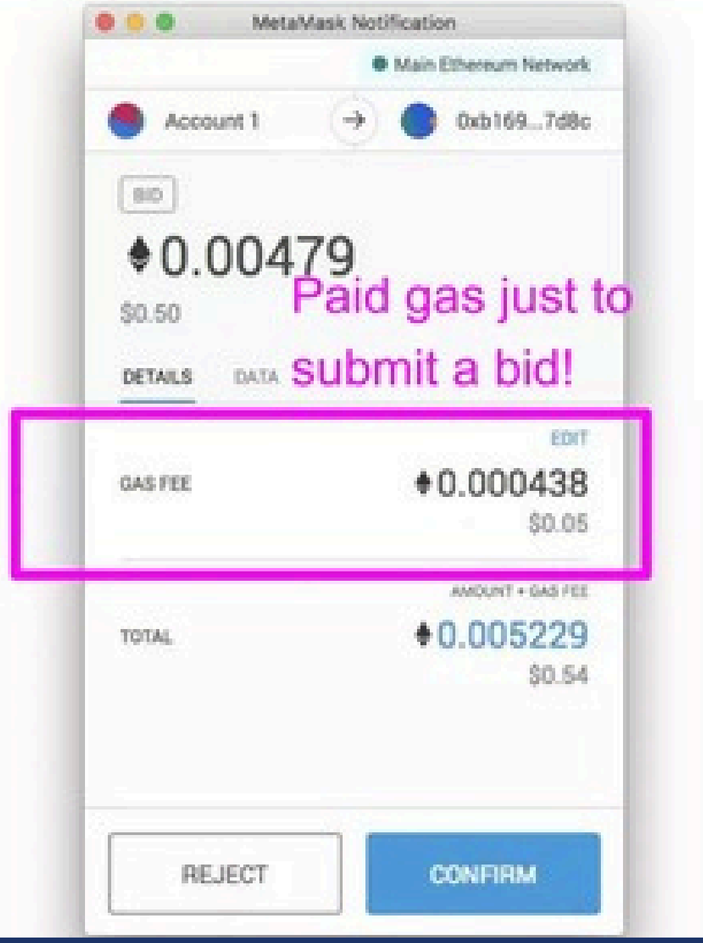
Gas is the accounting measure used to decide how complex a particular transaction is.

Let’s talk about gas

Every operation of your smart contract is added up and any action thus has a total gas cost.

Nodes of the network (aka miners) are trying to fit as many transactions into their blocks as possible and prioritize those which reward them the most per unit of gas.

Thus when your smart contract sends a transaction to the chain, it must (in Ethereum) also specify how much you’re willing to pay (in ETH) per unit of gas.



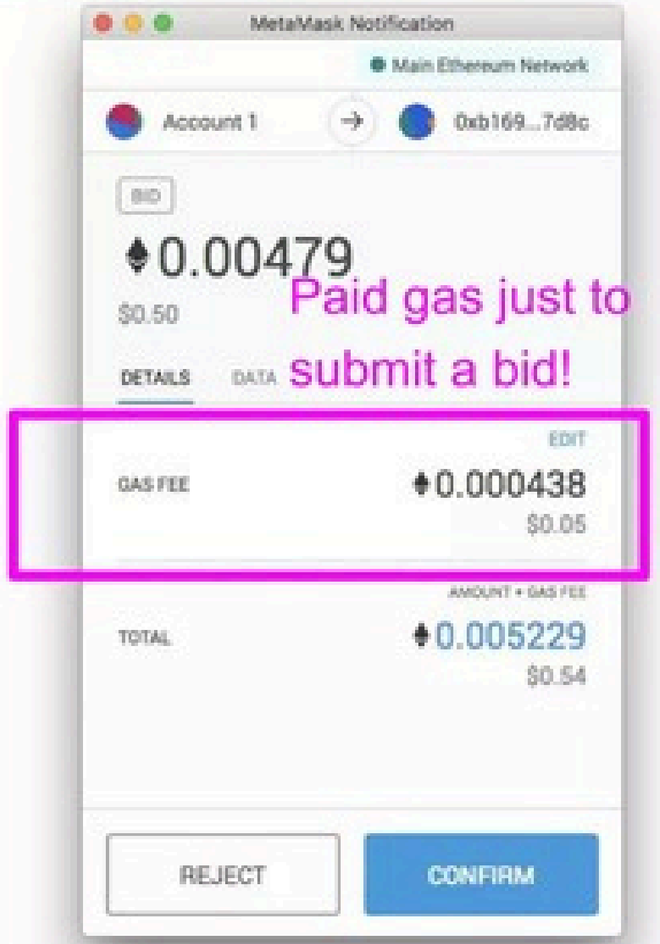
Let’s talk about gas

Who should pay for gas?

It’s a huge pain for users to deal with it.

It can be very costly for developers to foot the bill, especially without protections in place. Code well!

Not all chains use gas the same way but the principle exists -- nodes must be paid for ALL work somehow.



Diving deeper

Data Storage: Storage is very expensive to do on-chain because you're storing in every node! Offload as much as you can. If you need decentralization, check out **IPFS**, a decentralized file storage using peer-to-peer principles similar to BitTorrent.

Naming: Human-readable IDs (instead of long hashes) are a big problem. The project **Namecoin** is one of many projects trying to provide a single decentralized place to register and look up names.

Issues to think about

1. **Open Source:** DApp culture is all about transparency so open-source is the way to go.
2. **Governance:** Who actually decides what happens when you need to upgrade the contract? Who has control over its treasury balances?
3. **Decentralization:** What exactly do you let users control? Eg. CryptoKitties DNA or display?
4. **Oracles:** Where do we get third-party information from for smart contracts to use?

THANK-YOU