

# Sell Your Smart Contract

## Elevator Pitch: eBay for Smart Contracts

### Problem

Smart contracts on public blockchains are enabling individuals and organizations to sell on-chain products and services. Many, if not most, of these organizations are not publicly traded, meaning they do not have an ERC-20 token or on-chain treasury, that accrues revenue. This means there is now a niche, but rapidly growing market for on-chain private transactions.

Examples include:

- **Creator Royalty Agreements and Bonds:** NFTs enable creators to sell their art, music, and literary works directly to consumers. These creators may wish to sell all or a portion of the royalty streams from their works for upfront cash payments (See [Royalty Agreements](#) and [Bowie Bond](#)).
- **DAO Mergers and Acquisitions:** DAOs are merging with and acquiring both publicly-traded DAOs (i.e. DAOs with a token) and privately-owned DAOs (i.e. [Moloch DAOs](#)) (See [Rari Capital and Fei Protocol merger](#)). DAOs are also incorporated off-chain via [DAO LLCs](#) and other structures. It is not difficult to imagine a future where DAOs acquire off-chain companies and off-chain companies acquire DAOs!
- **On-Chain SaaS Products Offered by Companies:** Companies are launching on-chain services and products. For instance, DeFi Pulse launched the DeFi Pulse Index, a decentralized index ETF tracking the growth of decentralized finance, from which it earns [streaming fees](#) that accrue to a contract owned by DeFi Pulse. Other companies and DAOs including [Bankless](#), [MetaPortal](#), and [D4 DATA](#) have launched decentralized index funds and ETFs with the same on-chain SaaS business model.
- **DAO Lending:** [DebtDAO](#) is building a decentralized, permissionless credit union to buy, sell, underwrite and manage debt for the DeFi economy. DebtDAO's novel *Spigot contract* will enable loans which are automatically repaid by borrowers in a trustless manner. If the development of traditional finance provides us with any indication of the future of on-chain lending, we would expect that Spigot loans will be bought and sold after loan origination by DAOs, banks, hedge funds, and many other parties.
- **Programmable Cash Flows:** [Superfluid](#) is a protocol for programmable cash flows enabling streaming payments for subscriptions, salaries, rewards, and any imaginable composable stream of value with real-time, continuous settlement. Many of the

applications in the [Superfluid X HackMoney Ideas List](#) involve selling all, or a portion of, a Superfluid stream to users or investors.

**Unfortunately, there is no existing solution that enables trustless private market transactions on public blockchains.** How do individuals and businesses exchange smart contracts that own revenue streams for fixed or lump sum payments?

## Solution

In traditional off-chain markets, trusted 3rd parties like investment banks and title agencies (i.e. real estate transactions) facilitate the exchange of privately held property and businesses via [escrow agreements](#). Usually, investment banks and law firms execute the escrow process to facilitate mergers and acquisitions of privately held companies, typically taking 1% to 3% of the transaction's value in fees.

We can build an escrow smart contract that facilitates the *trustless* exchange of privately owned smart contracts for fixed payments, regardless of the asset type or value (1 ETH or 10,000 ETH), and return the value captured by trusted 3rd parties in traditional markets back to buyers and sellers.

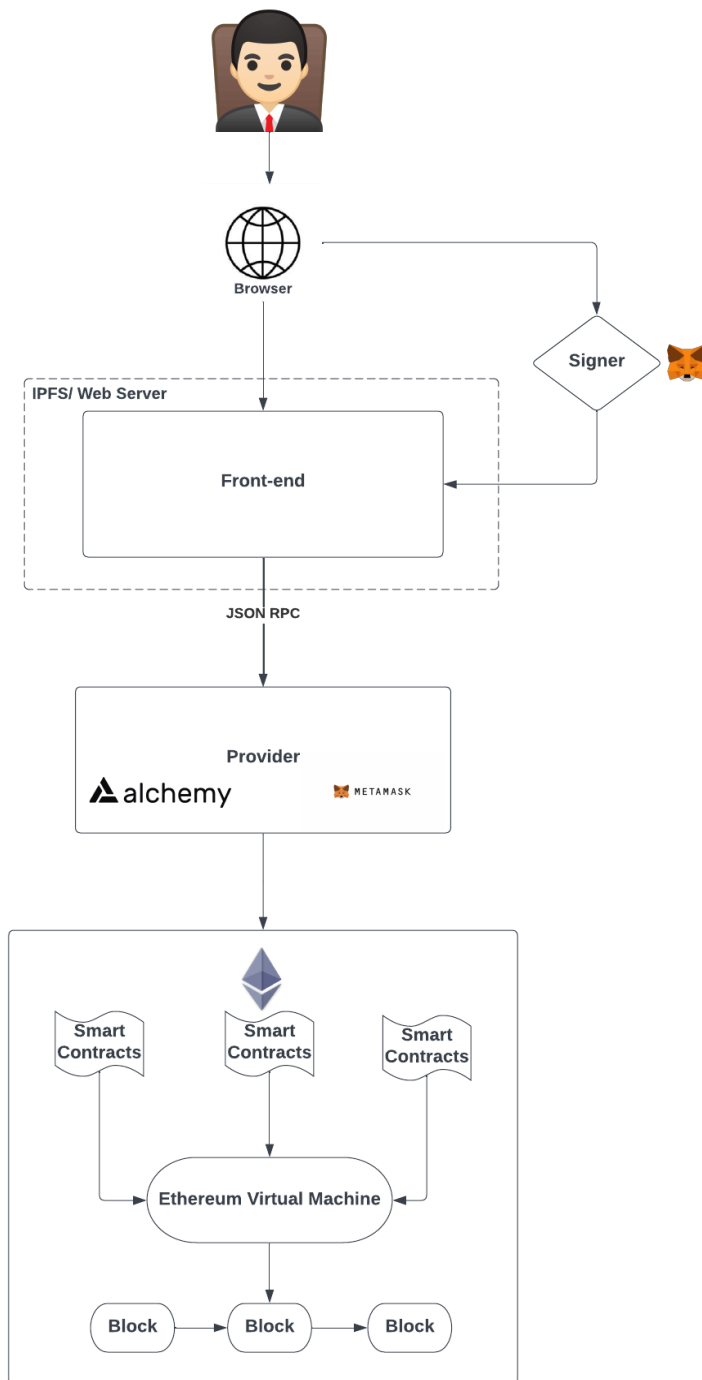
## Core Features & Functionality

- **Support First Use Case:**
  - Support the sale and transfer of [TokenSets](#), an Ethereum-based standard for decentralized ETFs:
  - TokenSets owners charge [streaming fees](#) analogous to [management fees](#) charged by traditional index funds and ETFs.
- **Necessary Pieces:**
  - Escrow Contract
  - Smart Contract w/ Revenue Stream
  - Live UI view of contract receiving eth and account ownership of token
  - Cancel button (if contract hasn't been signed)
  - Interested Purchaser/Acquirer
- 2-of-2 multisig / transaction between the Buyer and Seller.
- Buyer and Seller can cancel transaction at any-time (reverting to original state)
- Escrow contract initiator (Buyer or Seller) specifies *contract address* (i.e. smart contract being acquired) and *payment*.

## Future Possibilities

- ERC-721
- Fee model
- Transaction has time limit (respond within 30 days or canceled)
- M-of-n multisig
- Exit fee for breaking the contract.
- Study Mergers & Acquisitions.
- DaaS - escrow of large datasets.
- Escrow account doesn't accrue streaming revenue while it has ownership.

## Technical Architecture

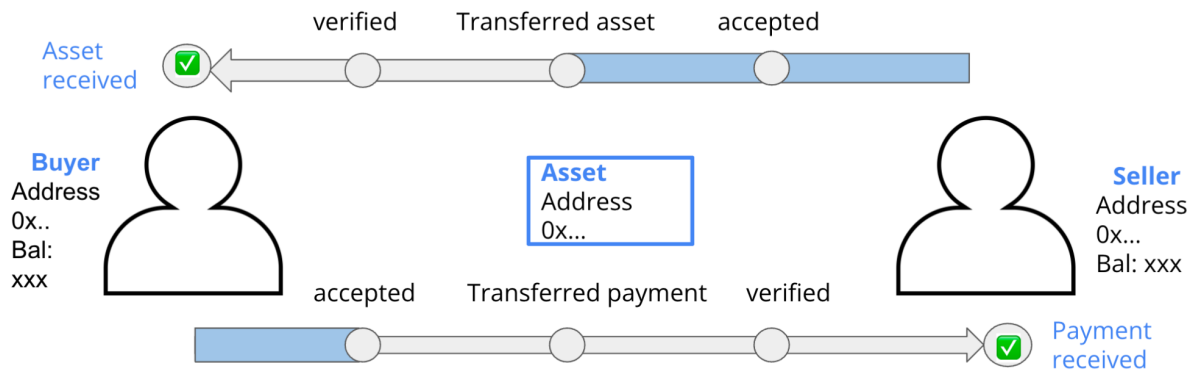


Front End



# Sysc

Sell your smart contract



# Sysc

Sell your smart contract

## Selling?

Address of asset

Your wallet address

Asking price

Go



Sysc

Sell your smart contract

Buying?

Address of asset to  
buy

Your wallet address

Offering price

2.3 ETH

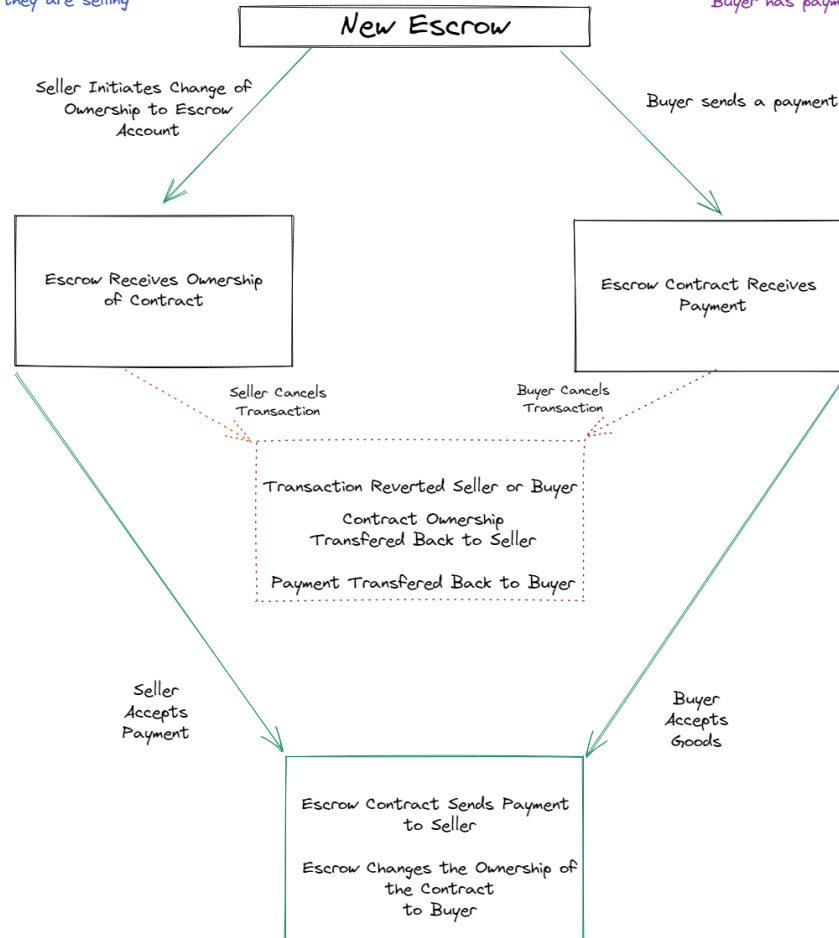
Buy

## Additional Notes

### Architecture Flow

Seller has contract they are selling

Buyer has payment (ETH) to buy contract



#### ○ Mockup

- Card1, card2, card3
  - Card1: streaming revenue
  - Card2: Account1 balances
  - Card3: Account2 balances
  - Buyer landing page (login, approval, etc)
  - Seller landing page (login approval, etc)
  - Front-end is interacting with Etherscan:
  - Need a TokenSets index on Ropsten / test network.
- 
- Smart Contracts
  - Backend

## Pseudocode for each entity:

### Escrow:

- Mapping to track contractHeld => asking price
- Mapping to track contractWanted => ethSent
- uint to track buyer addr (msg.sender)
- uint to track seller addr (msg.sender)
- function goodToSettle();
  - Make sure the contract exchanged for eth is the same one the buyer wants
  - Checks local mappings to see if askingPrice[contractAddress] <= ethHeld
    - If yes, fire call to seller to transfer eth
    - Change address of contract to buyer
    - Emit event to notify FE of latest status
- function cancelTransfer();
  - Can be called to cancel contract by seller or buyer only if not already settled
- function currentStatus();
  - Checks status of eth received, contract held
  - Emits event to FE on call of function

### Seller:

- Create an interface with EscrowContract
- function moveContractToEscrow(address contractAddress, uint askingPrice, address escrowAddr);
  - Check that contractAddress being sold, is indeed in the seller's wallet
  - Update contractHeld mapping in escrow to reflect with contractAddress => askingPrice
  - Change the address of the contract provided to the escrowAddr
  - Call function goodToSettle in Escrow contract to check if buyer and seller can settle
  - Emit event to notify FE of latest status

### Buyer:

- Create an interface with EscrowContract
- function payForContract(address contractAddress, uint ethToSend, address escrowAddr);
  - Update mapping in escrow contract contractAddress wanted => buyer address
  - Use to.call to send eth to escrow
  - Call function goodToSettle in Escrow contract to check if buyer and seller can settle
  - Emit event to notify FE of latest status

### Front End:

- Create event listeners for Seller, Buyer, Escrow
- Modify status bar to upon event transmission



- Toggle form shown based on whether user is buyer/seller

---

## Notes from Krishna:

- user journey for cancellation is low priority
- function should not be reentrant
  - moving contract ownership
- transaction needs to be 1 transaction
- do not use IPFS for the frontend
- want to focus on solidity, ethereum, metamask, etc.
- does ERC20 template have *getOwner* and *setOwner*?
- modifier to check if method exists to get owner → check if contract is eligible for escrow
- modifier *isEscrowContract* (does not need full tokenset UI) (cast address into the interface) (catch the error and return “contract not eligible for escrow”)
- Could create an adapter function to mapper (or do simple if/else) to escrow other contracts that are escrowable
- [OpenZeppelin Ownable Contract](#) Module: *isOwner* and *transferOwnership*
- Has someone created an ERC standard for escrowable contracts? You can show an interface in the demo for future DappCamps can use.
- Given an address, how do you know it's an ERC-20? If escrowable/ownership,
- Make it so other teams can check if a contract is Escrowable or not
- When a contract is transferred, does it change ownership of assets as well?

## Superfluid Notes

### Tokenize a Superfluid Stream Using APWine 🔥

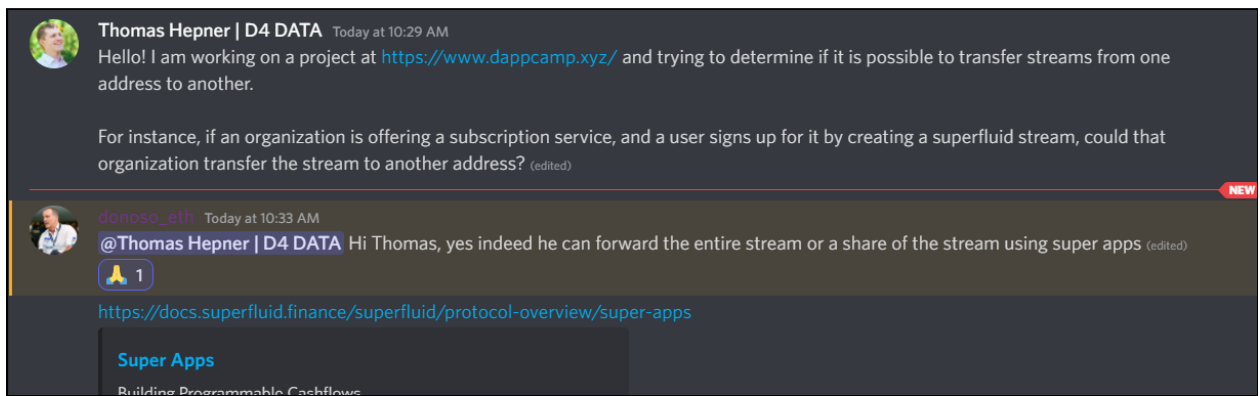
-build a contract that generates tokens which represent access to a future cash flow (i.e. a money stream), then sell those tokens using APWine to get access to that future cash flow *now*. This is a good use case for DeFi yields, tokenized subscriptions, and salaries

## Revenue Based Financing

Use something like the Tradeable Cashflow contract to allow subscription based businesses to raise funding based on their existing customer base

## API Payments Marketplace

-allow web3 devs to sell access to their APIs via streams in a marketplace



The screenshot shows a Discord chat interface. At the top, a user named 'Thomas Hepner | D4 DATA' (with a profile picture of a man) posts a message at 10:29 AM: 'Hello! I am working on a project at <https://www.dappcamp.xyz/> and trying to determine if it is possible to transfer streams from one address to another.' Below this, a second message from the same user asks: 'For instance, if an organization is offering a subscription service, and a user signs up for it by creating a superfluid stream, could that organization transfer the stream to another address? (edited)'. A red 'NEW' tag is visible to the right of this message. Then, a user named 'donoso\_eth' (with a profile picture of a man) responds at 10:33 AM: '@Thomas Hepner | D4 DATA Hi Thomas, yes indeed he can forward the entire stream or a share of the stream using super apps (edited)'. Below the response is a reaction of a '👍' emoji with a count of '1'. At the bottom, there is a link to 'https://docs.superfluid.finance/superfluid/protocol-overview/super-apps' and a section titled 'Super Apps' with the subtitle 'Building Programmable Cashflows'.

[Superfluid - Tradeable Cash Flow Tutorial](#)

## OpenZeppelin Notes

- [Escrow](#)
- [Roles](#)

## App idea #1

What do you want to build?	DAO-to-DAO Escrow Tool for Trustlessly Executing Mergers & Acquisitions
What problem does this solve?	Trustlessly transfers digital products and services (i.e. smart contracts, decentralized websites) offered by one DAO to another removing need for complicated legal agreements.

What are the core features and functionalities of the app?	<ul style="list-style-type: none"> <li>- Interacts with Gnosis Safe</li> <li>- One party puts up ETH or ERC20 tokens</li> <li>- One party puts up a transaction transferring control of a smart contract to the other.</li> <li>- Transaction executes when both conditions are met.</li> </ul>
--	---

## Additional Notes:

- Name Ideas: “On-Chain M&A” ; “M&A Marketplace”
- Works for anything with payment streams or royalties
- Case Study: Tribe acquisition of Rari Capital
- [Escrow Ideas](#) for DAppCamp Cohorts
- [Escrow My Ether](#): very different than this idea
- [SmartLink](#): On Tezos?

## Whitepaper:

- Starting from the end - how do you want to present the idea?
- Do you want to just show the contract? Think about the user journey?
- Process:
  - Whitepaper
  - Smart Contract
  - Demo
- Some teams that liked the contract enough to deploy to mainnet
- Past cohorts have raised money from VCs
- Ask Krishna about scope of project
- Questions about implementation → ask Krishna or post in Discord
- Difference between contracts that ownership can be transferred and which cannot
- DATA Contract:
  - <https://etherscan.io/token/0x33d63ba1e57e54779f7ddaeaa7109349344cf5f1#writeContract>
- setManager, manager
- Transfer contract / money to escrow, then set conditions
- Escrow does not execute until conditions are met
- Everything you want to implement should go in whitepaper
  - List core features and functionality
  - Brief description of overall technical architecture
- **Finalize whitepaper by Monday**
-

# Notes

Potential demo flow:

Potential use cases/users: