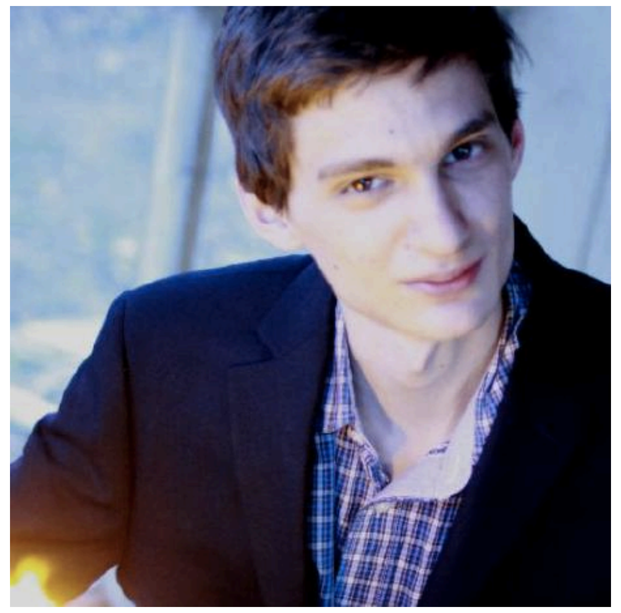
Clockwork Finance: Automated Analysis of Economic Security in Smart Contracts

To Appear in IEEE S&P’23



Kushal Babel

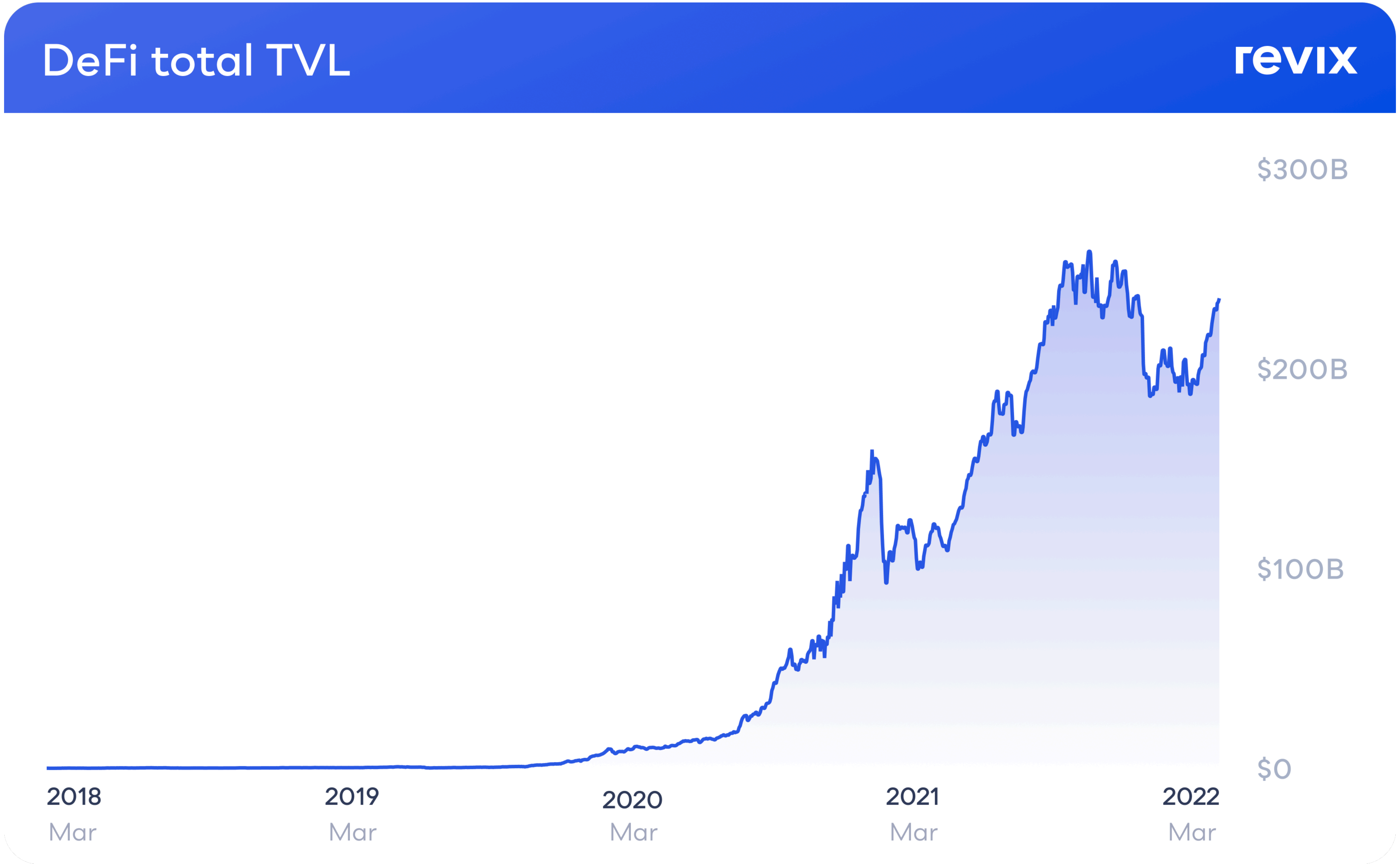
Philip Daian

Mahimna Kelkar

Ari Juels

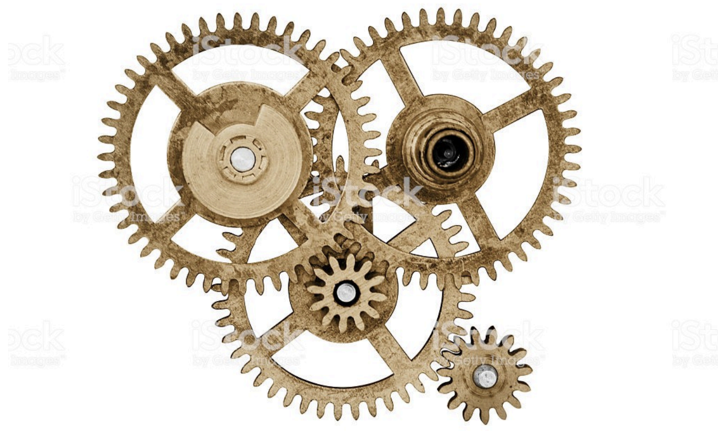
(First three authors contributed equally)

DeFi



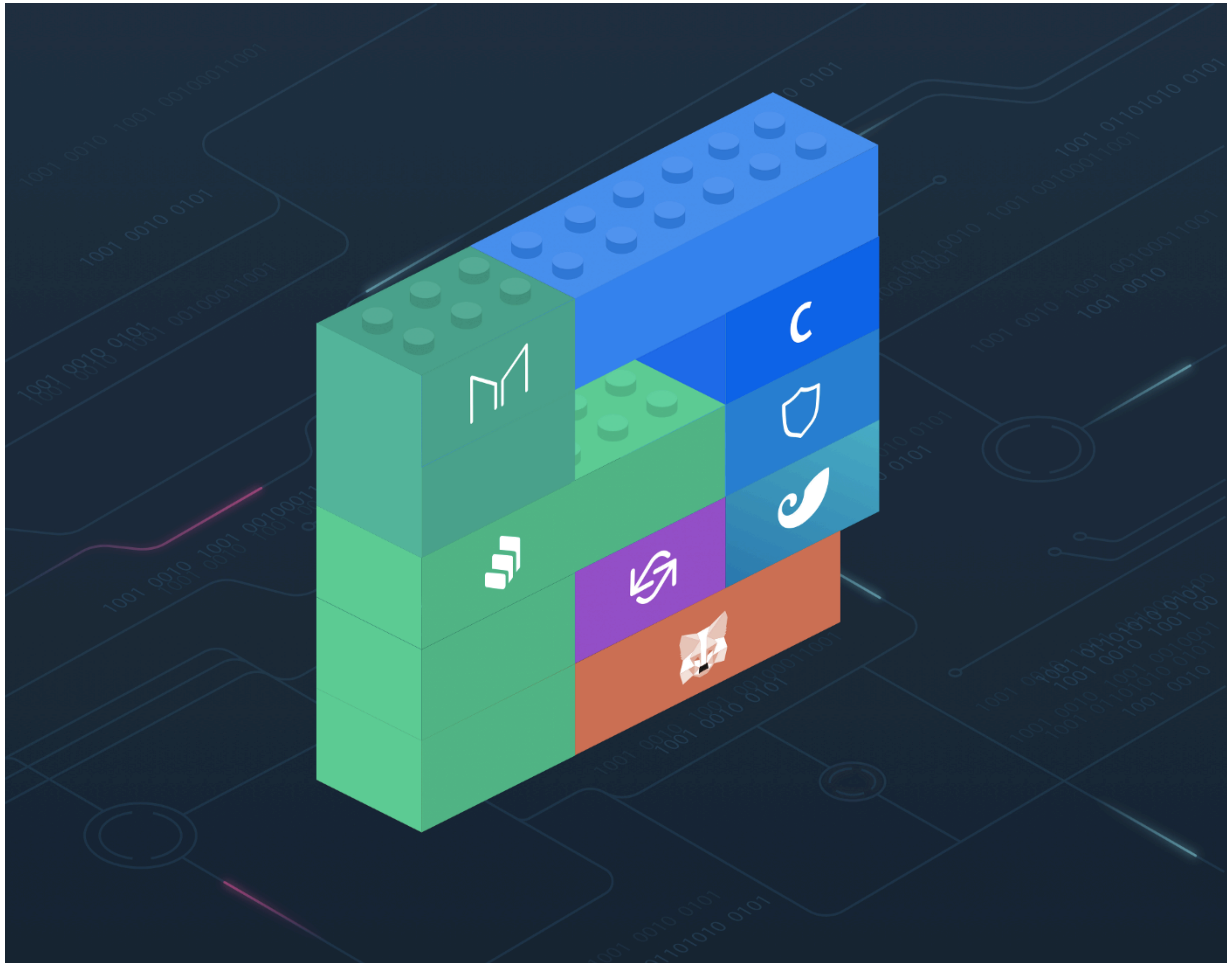
DeFi total TVL

Smart Contracts, Like Clockwork!



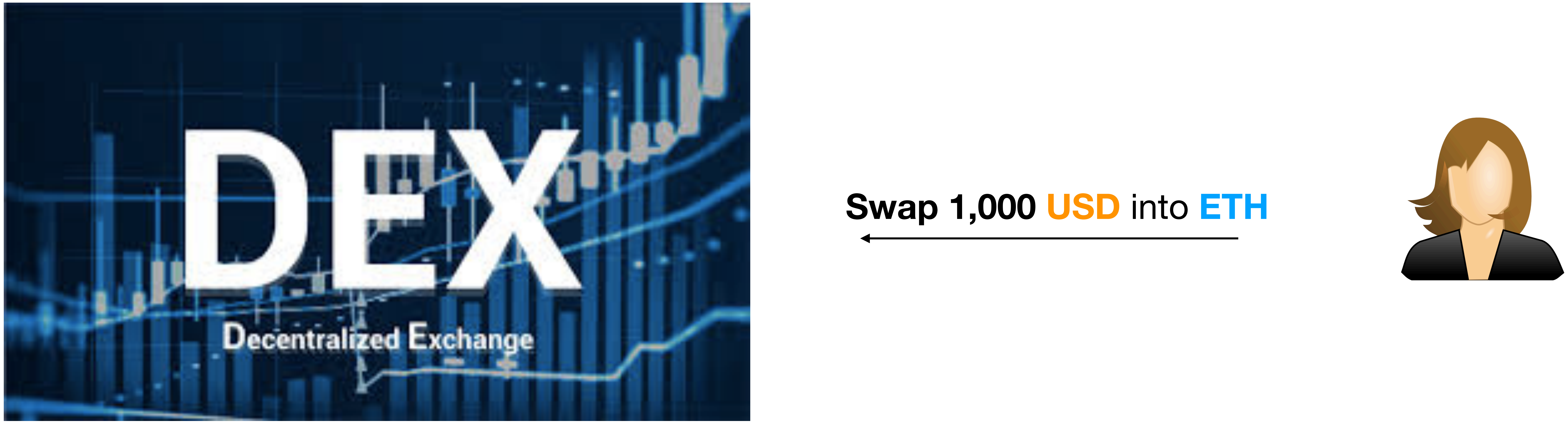
- Smart contracts execute in sequential and atomic transactions
- Execution is deterministic
- Most blockchains have transparent execution
- Therefore: Easy interoperability among smart contracts and novel financial instruments

Money Legos



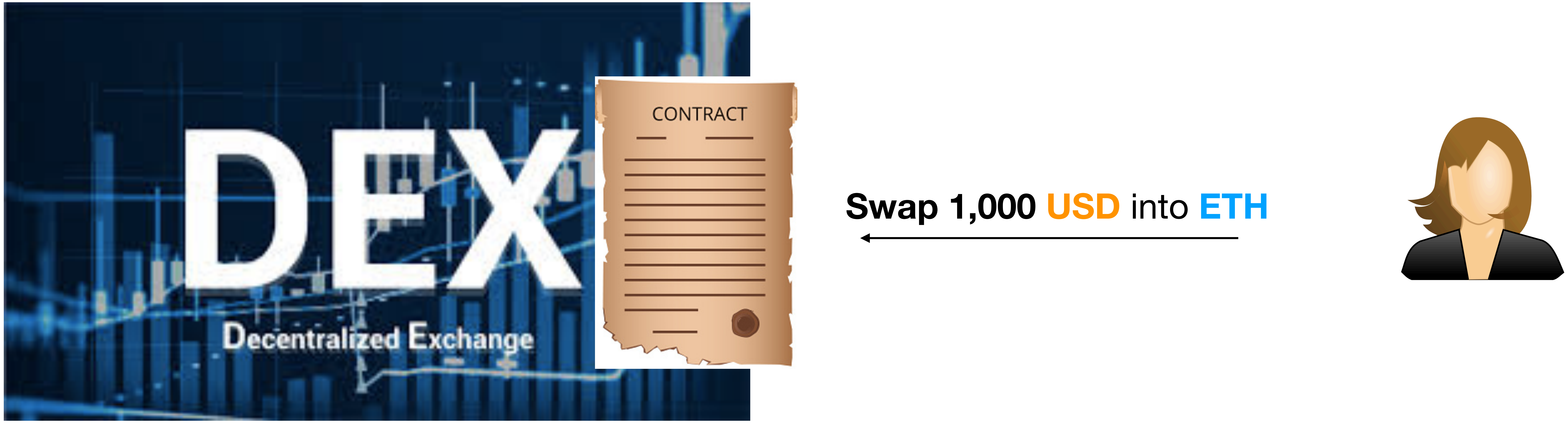
Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764

Unintended Behaviour



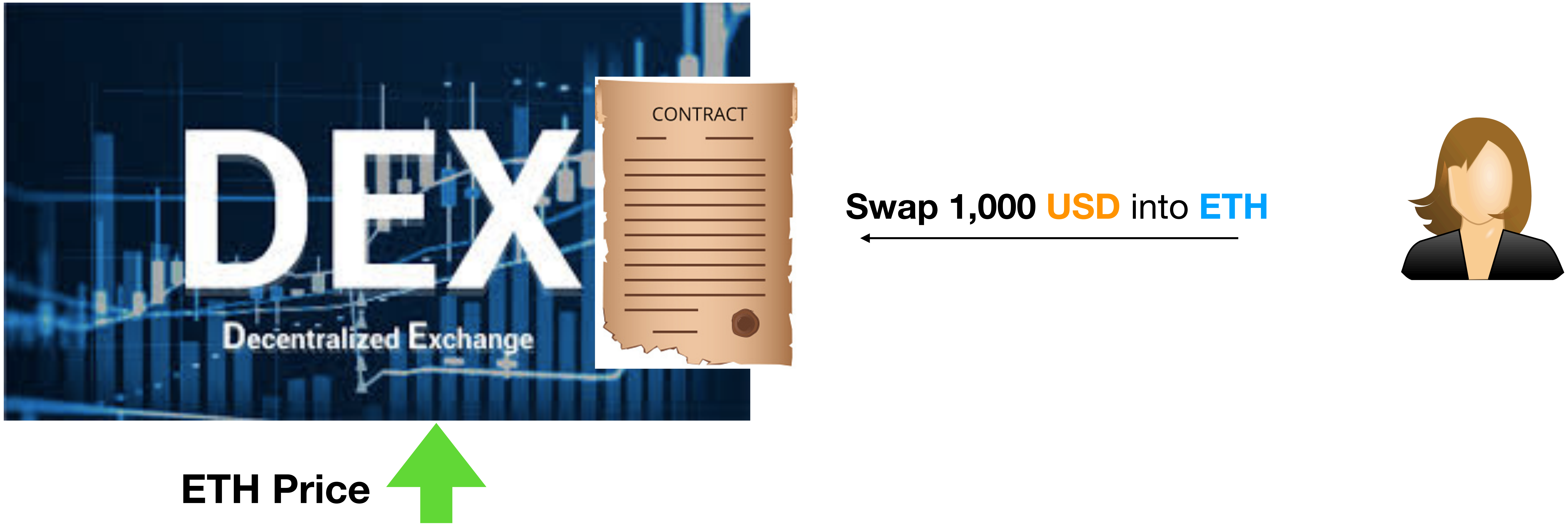
Swap 1,000 USD into ETH

Unintended Behaviour



Swap 1,000 USD into ETH

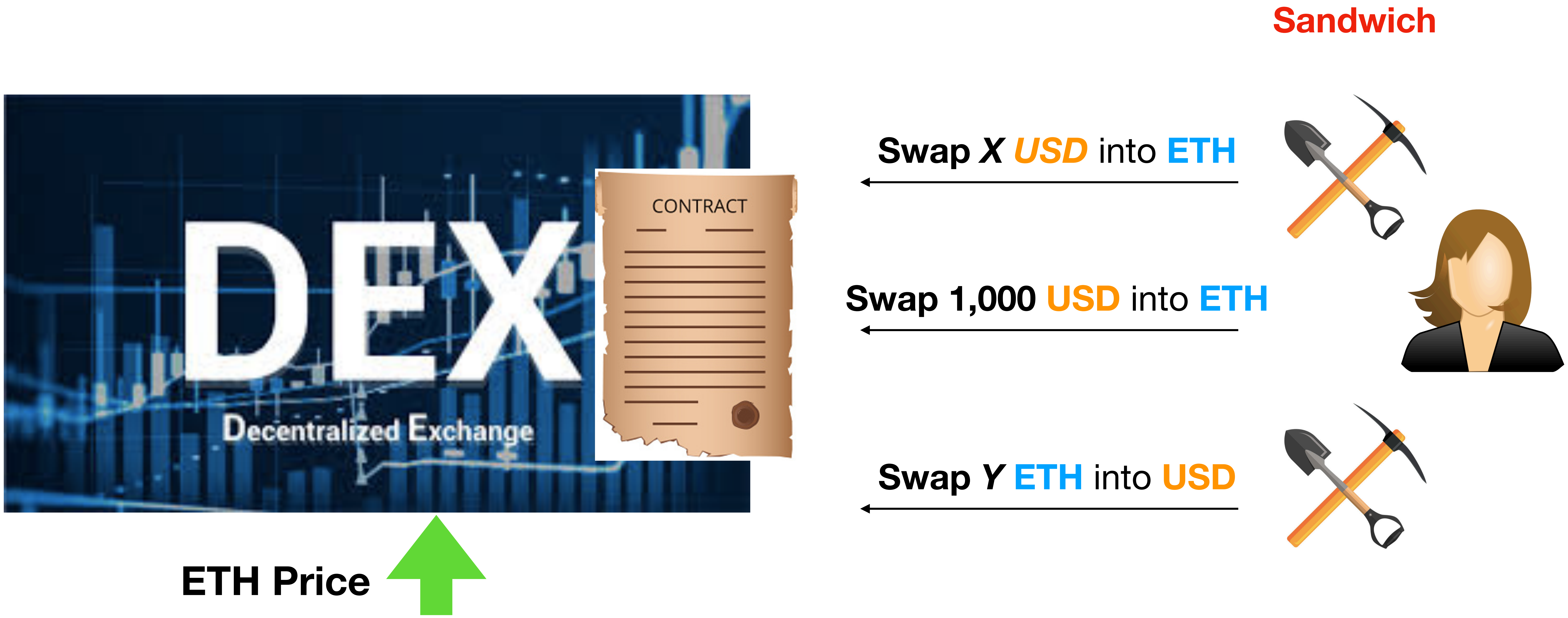
Unintended Behaviour



Swap 1,000 USD into ETH

ETH Price ↑

Unintended Behaviour



Sandwich

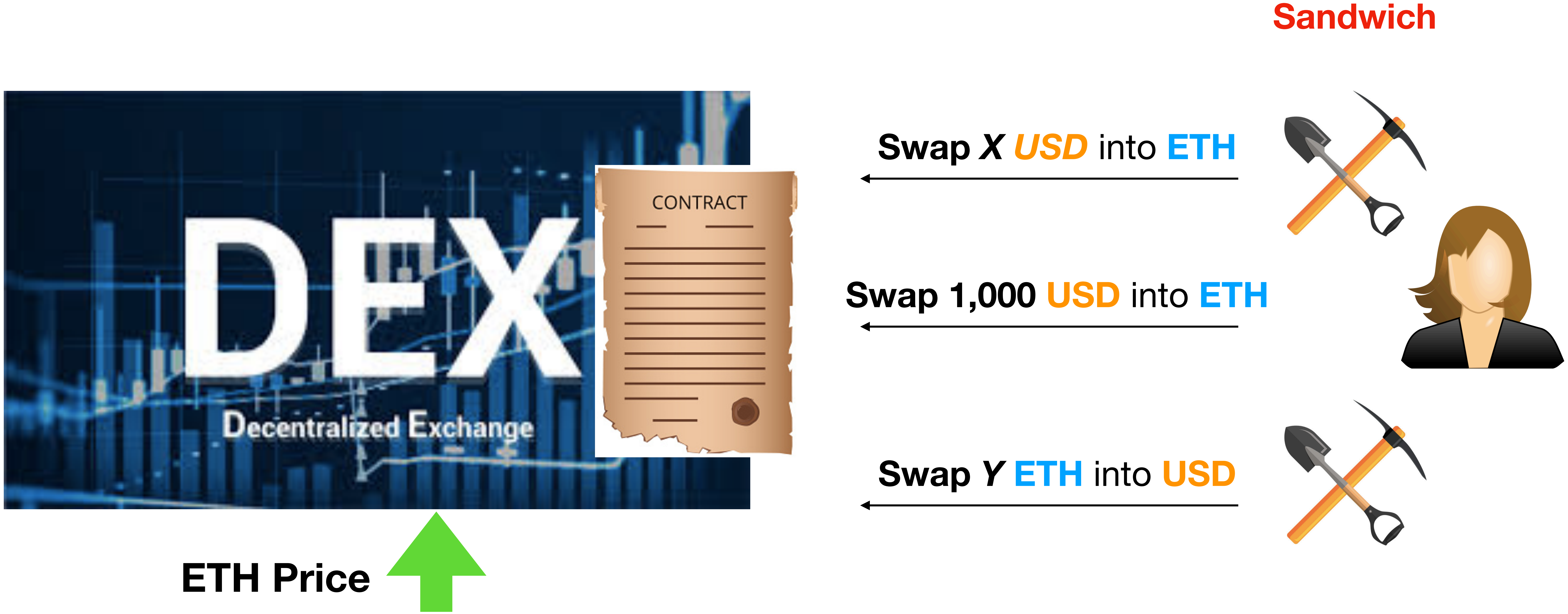
Swap *X* *USD* into **ETH**

Swap 1,000 USD into **ETH**

Swap *Y* ETH into **USD**

ETH Price

Unintended Behaviour



Sandwich

Swap *X* ***USD*** into **ETH**

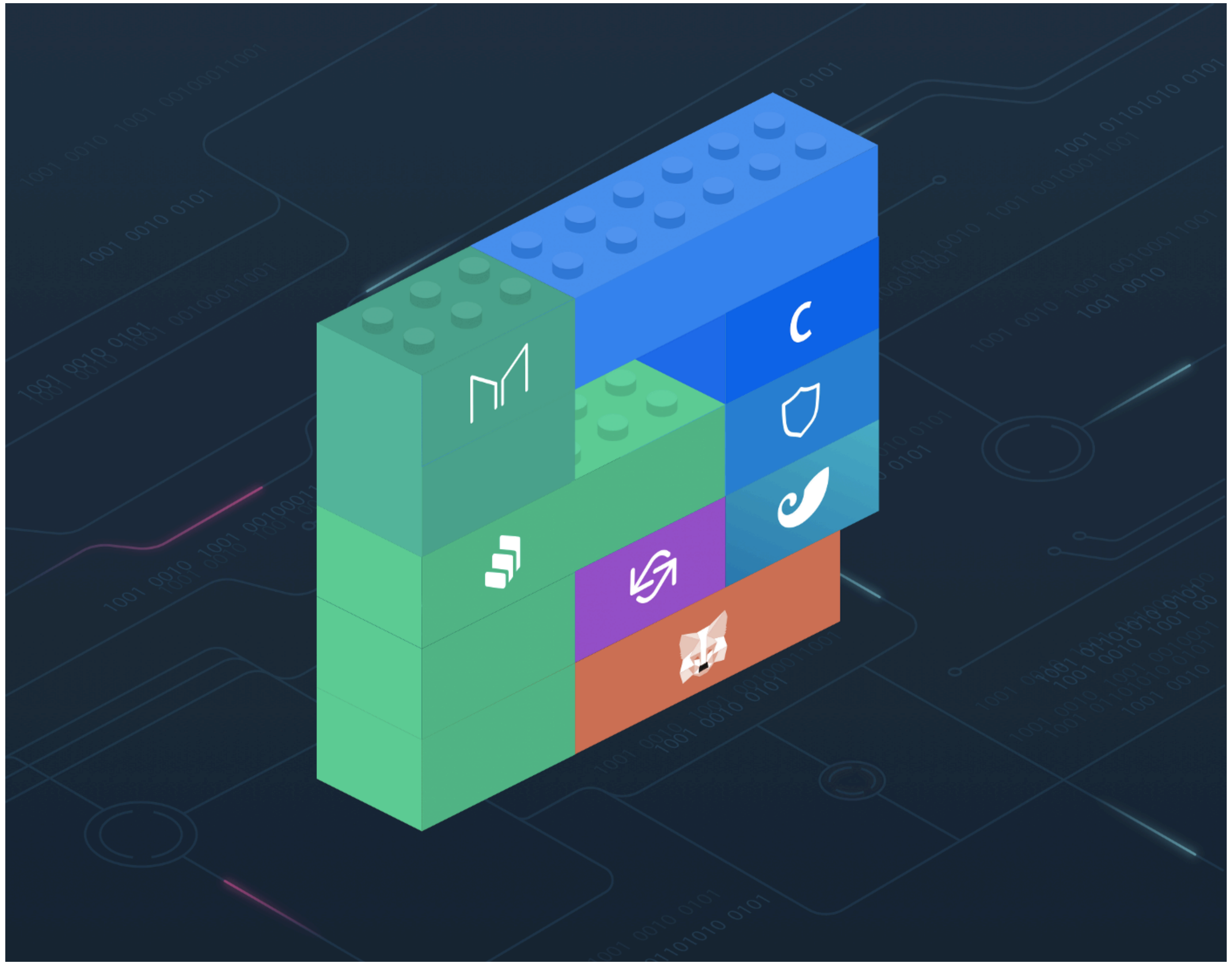
Swap 1,000 USD into **ETH**

Swap *Y* ETH into **USD**

ETH Price

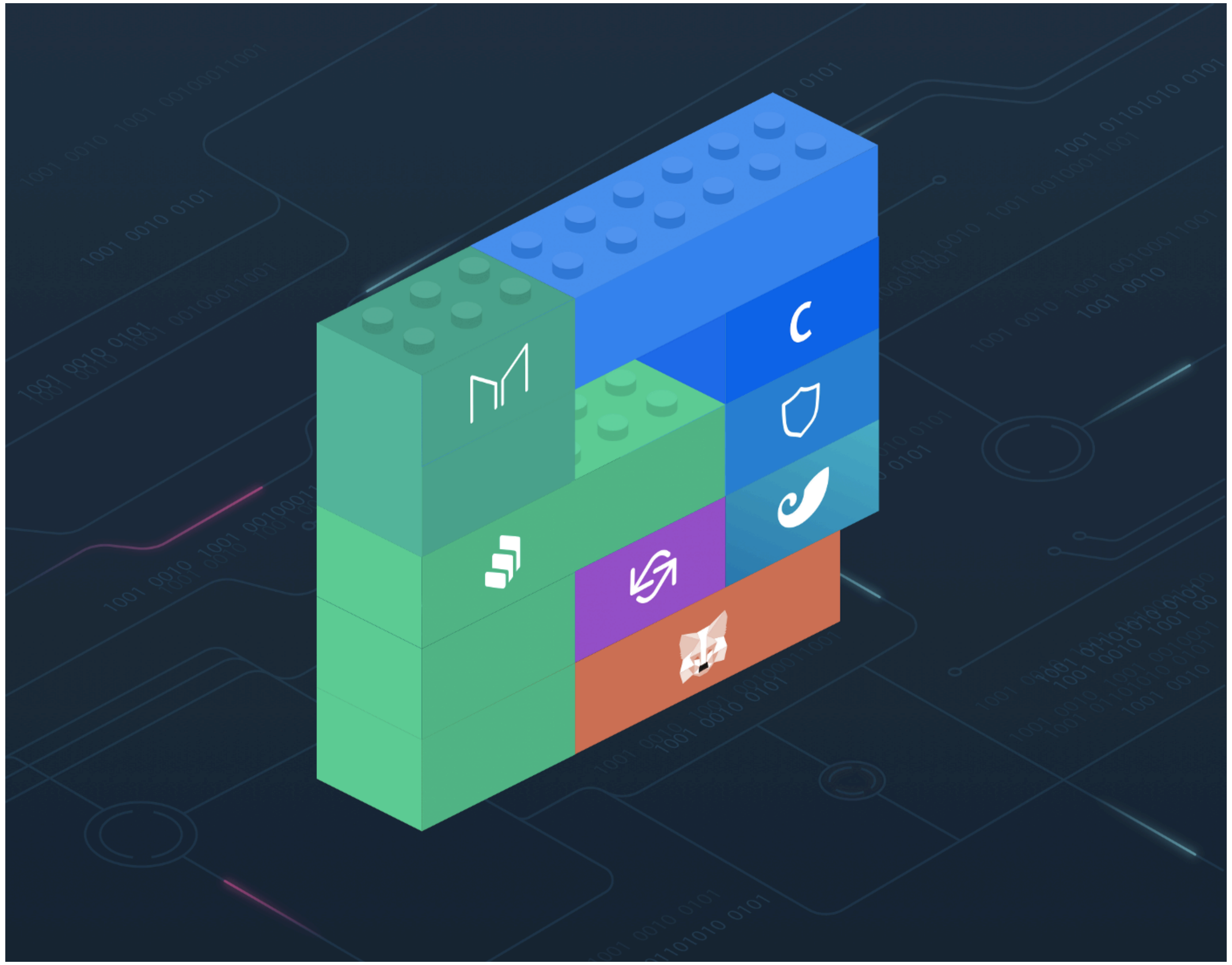
MEV = Miner Extractable Value (or Maximal Extractable Value) - Ability to extract value by reordering, inserting or censoring transactions

Contract Composition



Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764

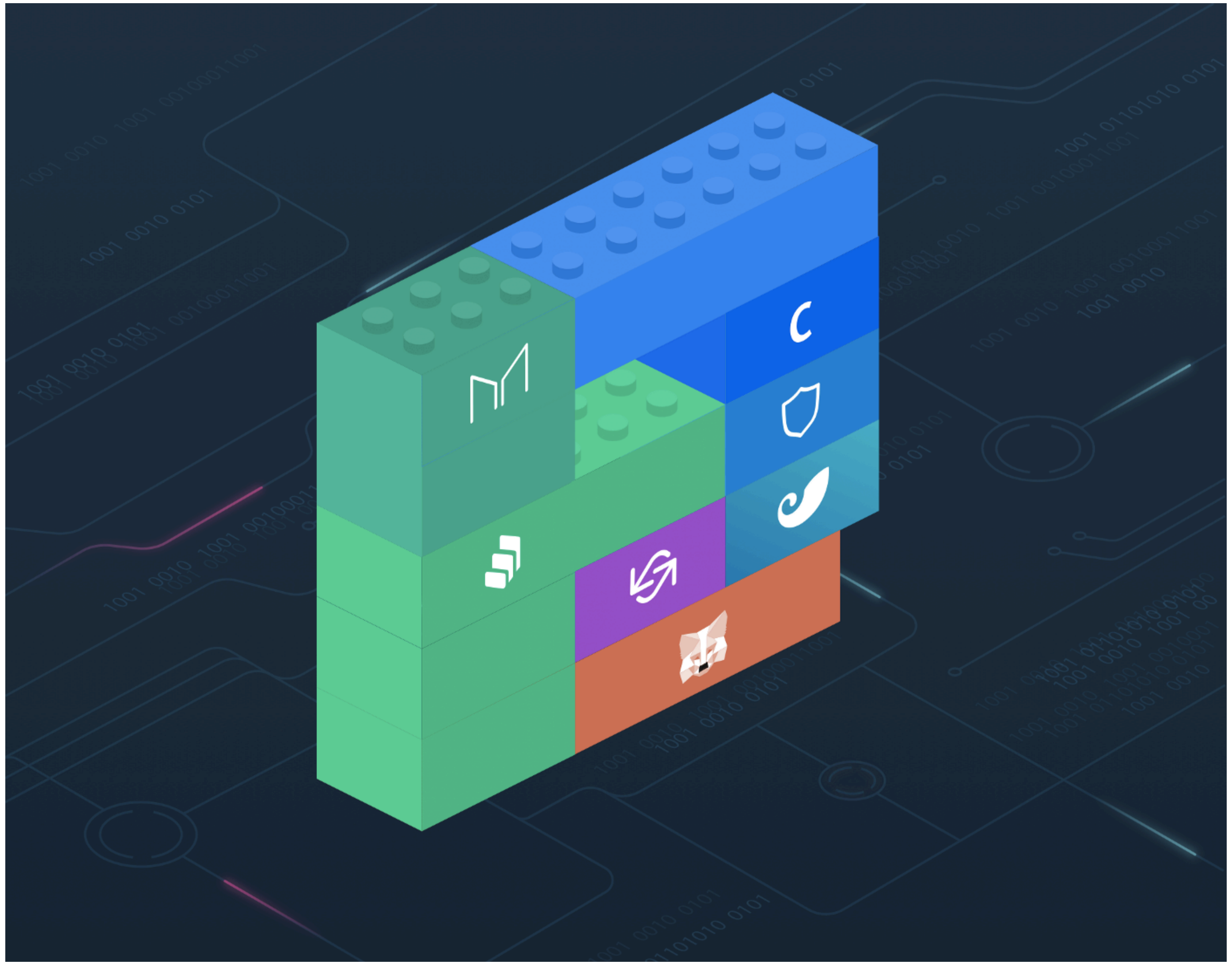
Contract Composition



Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764

- Flashloans + DEX

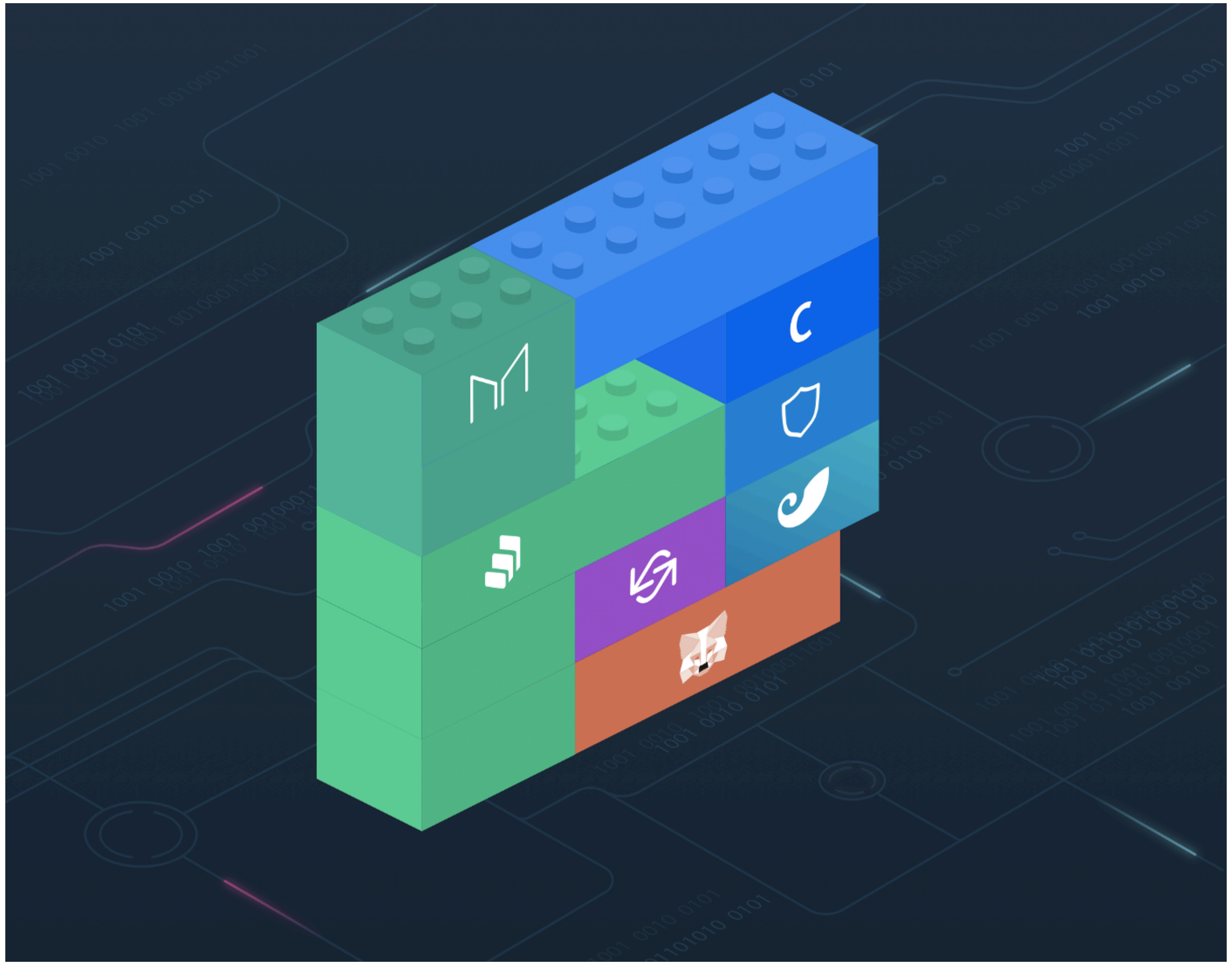
Contract Composition



Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764

- Flashloans + DEX
- Lending contracts using DEX to price the debt

Contract Composition



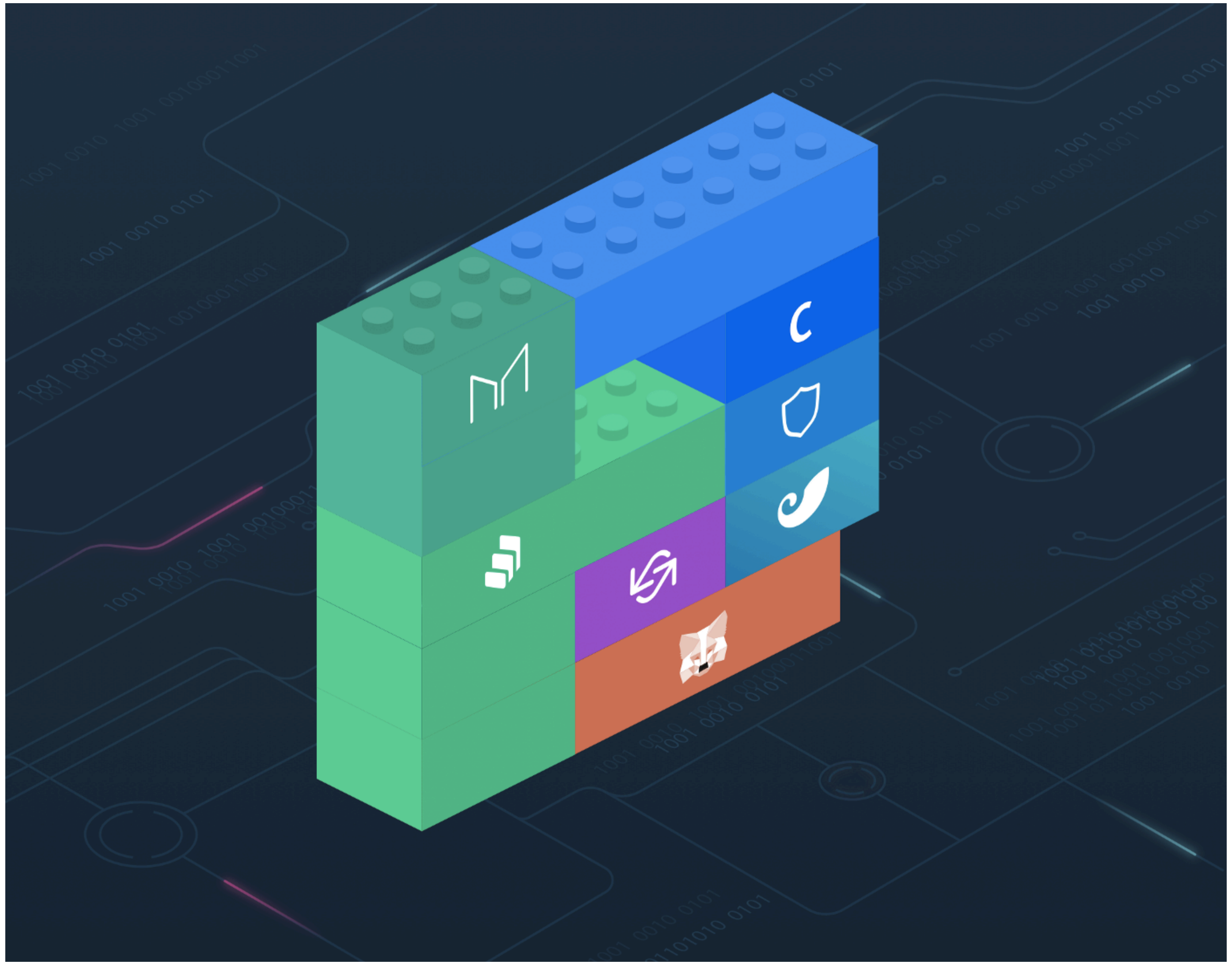
Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764

- Flashloans + DEX

- Lending contracts using DEX to price the debt

- Flashloans + Governance Contract

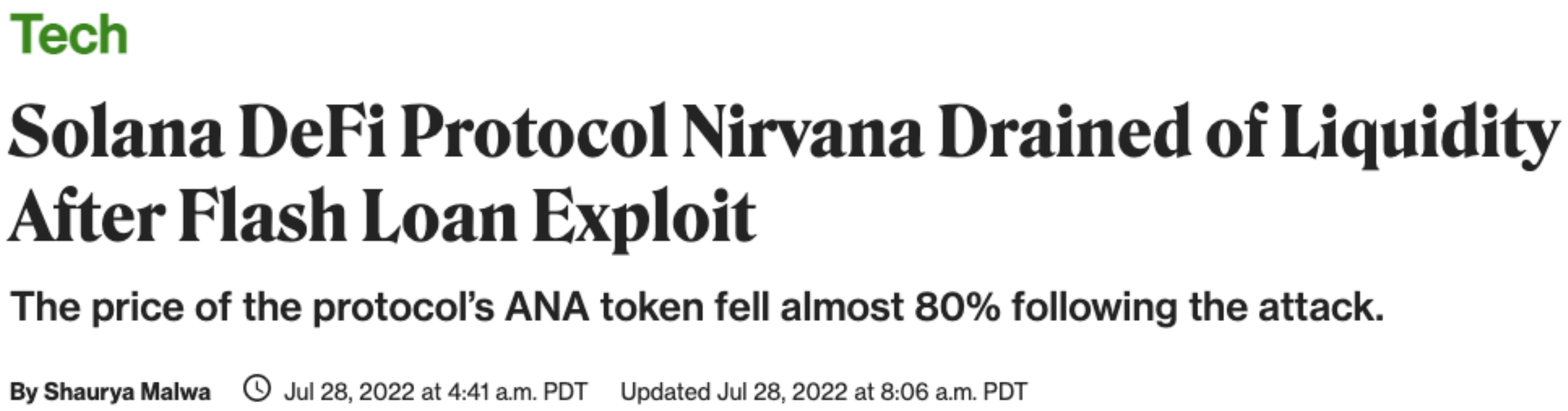
Contract Composition



Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764

- Flashloans + DEX
- Lending contracts using DEX to price the debt
- Flashloans + Governance Contract
- DEX + DEX + DEX …

Unintended Behaviour

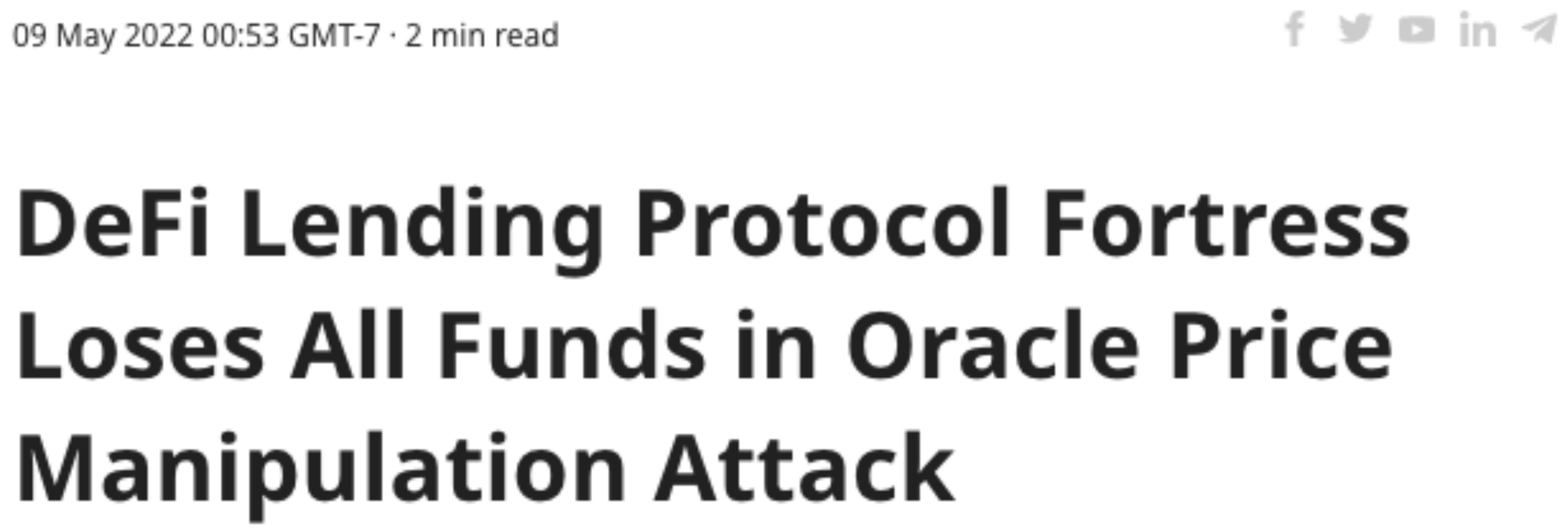


Tech

Solana DeFi Protocol Nirvana Drained of Liquidity After Flash Loan Exploit

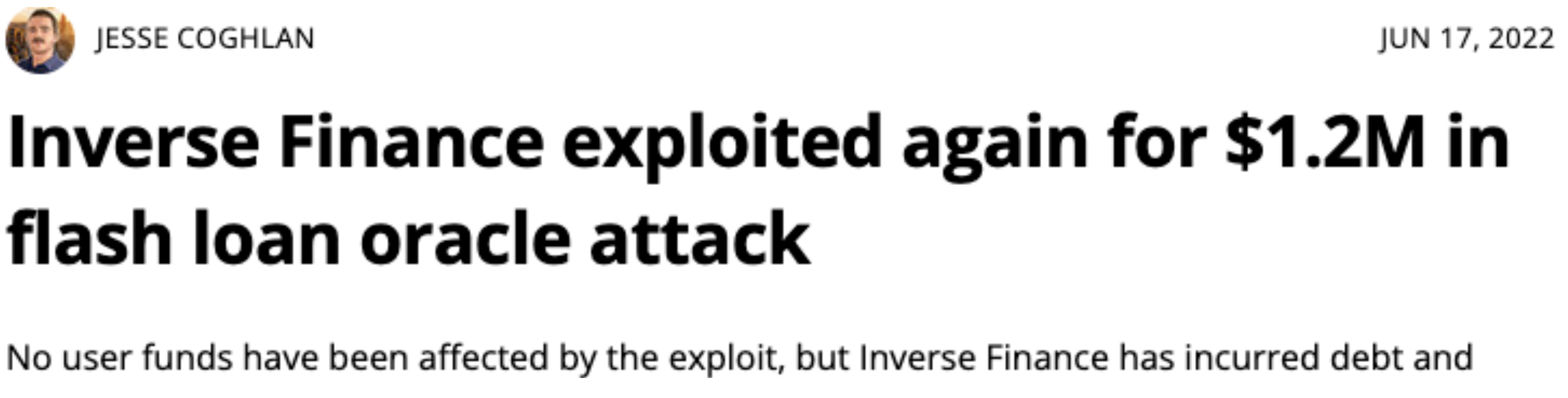
The price of the protocol’s ANA token fell almost 80% following the attack.

By Shaurya Malwa Jul 28, 2022 at 4:41 a.m. PDT Updated Jul 28, 2022 at 8:06 a.m. PDT



09 May 2022 00:53 GMT-7 · 2 min read

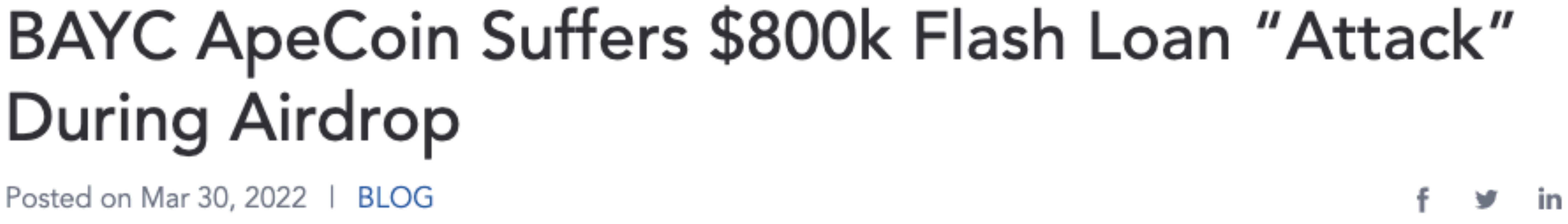
DeFi Lending Protocol Fortress Loses All Funds in Oracle Price Manipulation Attack



JESSE COGHLAN JUN 17, 2022

Inverse Finance exploited again for $1.2M in flash loan oracle attack

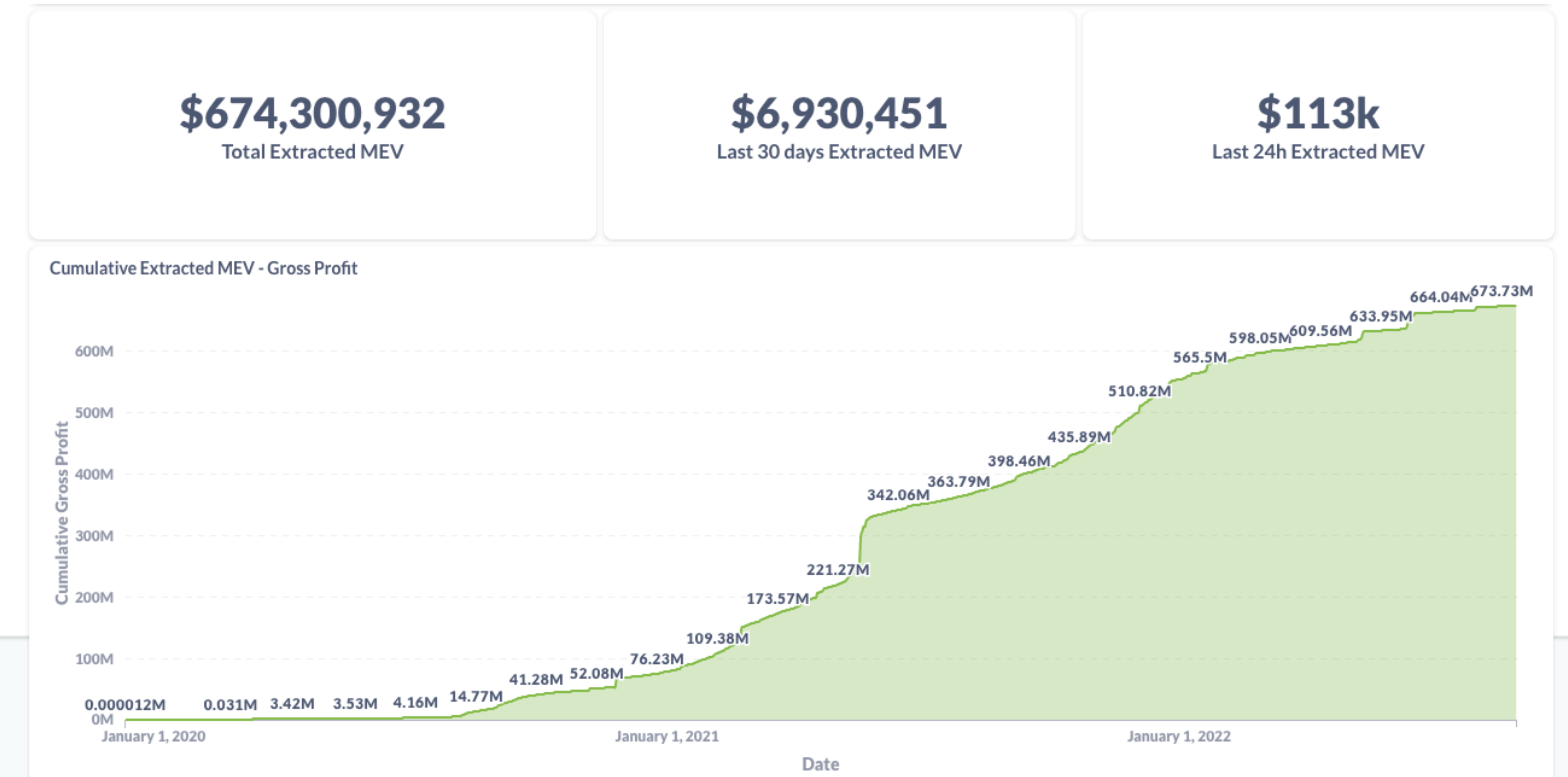
No user funds have been affected by the exploit, but Inverse Finance has incurred debt and offered the attacker a bounty to



BAYC ApeCoin Suffers $800k Flash Loan “Attack” During Airdrop

Posted on Mar 30, 2022 | BLOG

MEV…An Industry



$674,300,932
Total Extracted MEV

$6,930,451
Last 30 days Extracted MEV

$113k
Last 24h Extracted MEV

Cumulative Extracted MEV - Gross Profit

https://explore.flashbots.net

Existing Techniques for Security

- Human Auditing
- Fuzz Testing
- Static Analysis (eg. Slither)
- Formal Verification of functional correctness

Focus on Bug Hunting, Functional Correctness and Secret Leaks

This Work - Clockwork Finance

Directly reason about economic properties of smart contracts (and their interactions) by leveraging existing formal verification techniques

Unlike Traditional Finance, Smart Contracts execution is **deterministic**, **sequential**, **transparent** and **atomic** — allowing for formal verification of the behaviour of DeFi applications

Benefits to the ecosystem

Developers - Prove bounds on the value exposed by their contracts and interaction of their contracts with other contracts

Users - Find bounds on the value extractable from their transactions

Consensus Researchers - Rigorously study the impact of MEV on consensus

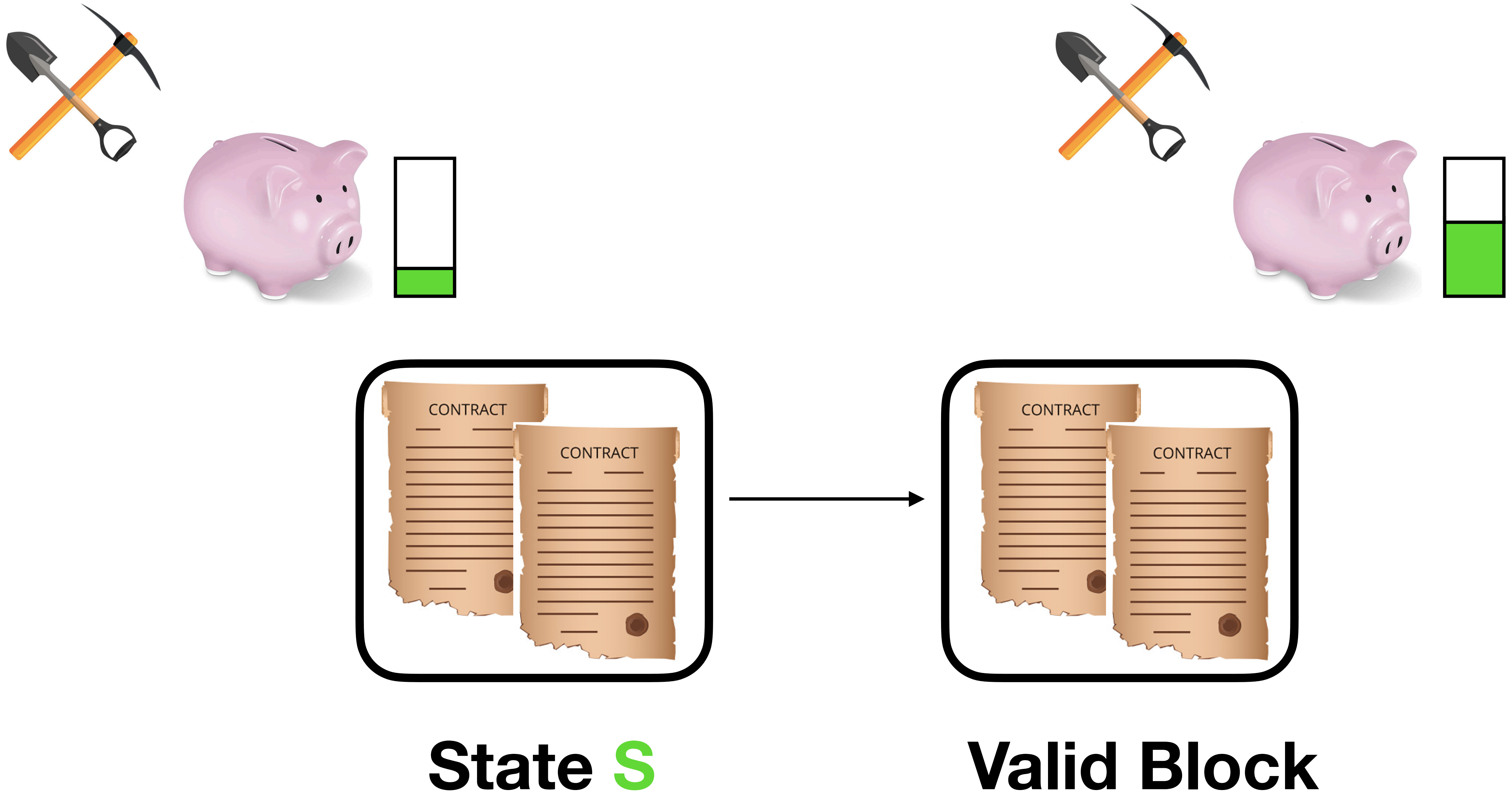
Outline

- Definitional tools
  - Defining (M)EV
  - Defining Secure Composition
- Practical Instantiation into Clockwork Finance Framework (CFF)
  - Design
  - Use for proofs
  - Use for finding attacks

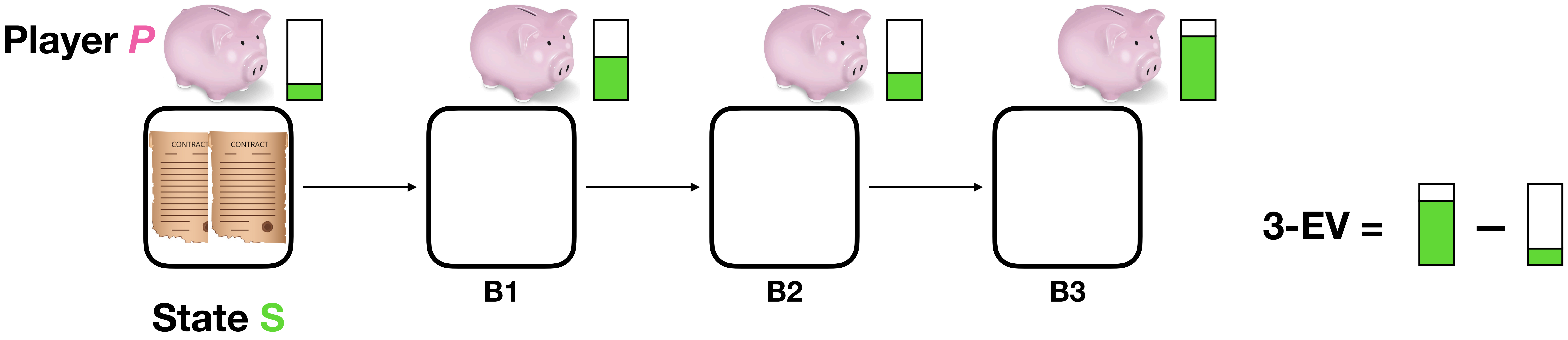
Outline

- Definitional tools
    - Defining (M)EV
    - Defining Secure Composition
- Practical Instantiation into Clockwork Finance Framework (CFF)
    - Design
    - Use for proofs
    - Use for finding attacks

Miner Extractable Value (MEV)



Extractable Value (EV)



Player *P*

State S

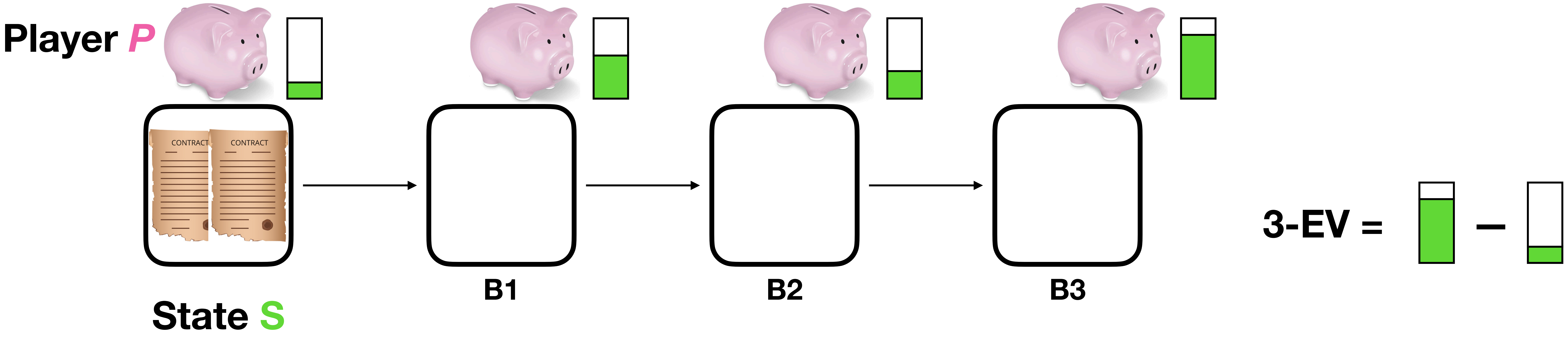
B1

B2

B3

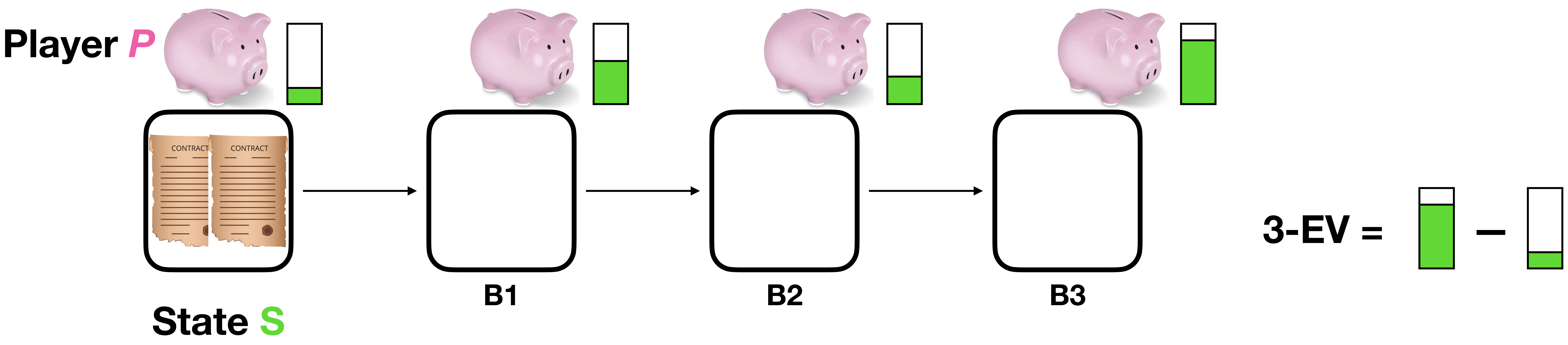
3-EV = 🟩 − 🟩

Extractable Value (EV)

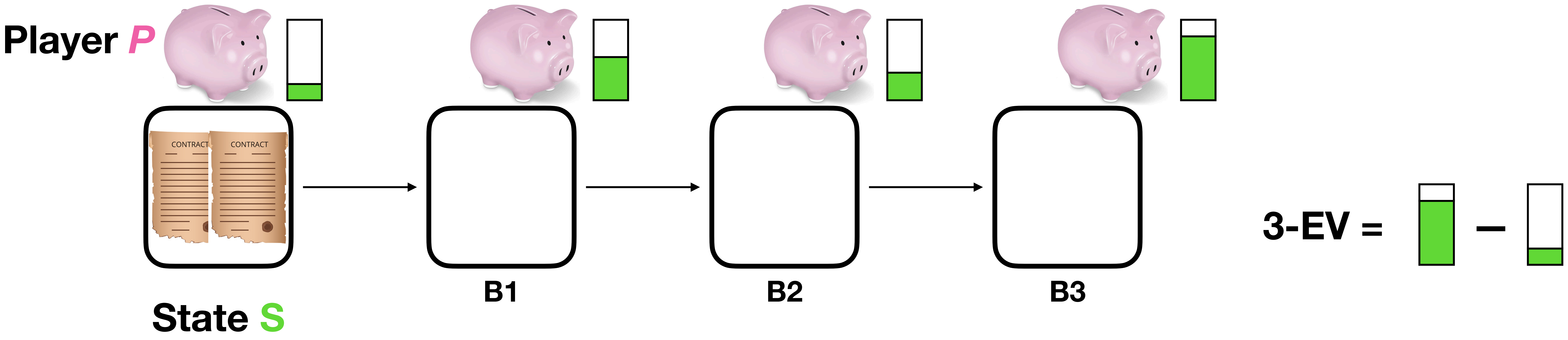


$$\mathrm{EV}(P,\mathcal{B},s) = \max_{(B_1,\ldots,B_k)\in\mathcal{B}} \left\{ \sum_{a\in A_P} \begin{array}{l} \mathsf{balance}_k(a)[0] \\ -\mathsf{balance}_0(a)[0] \end{array} \right\}$$

Extractable Value (EV)



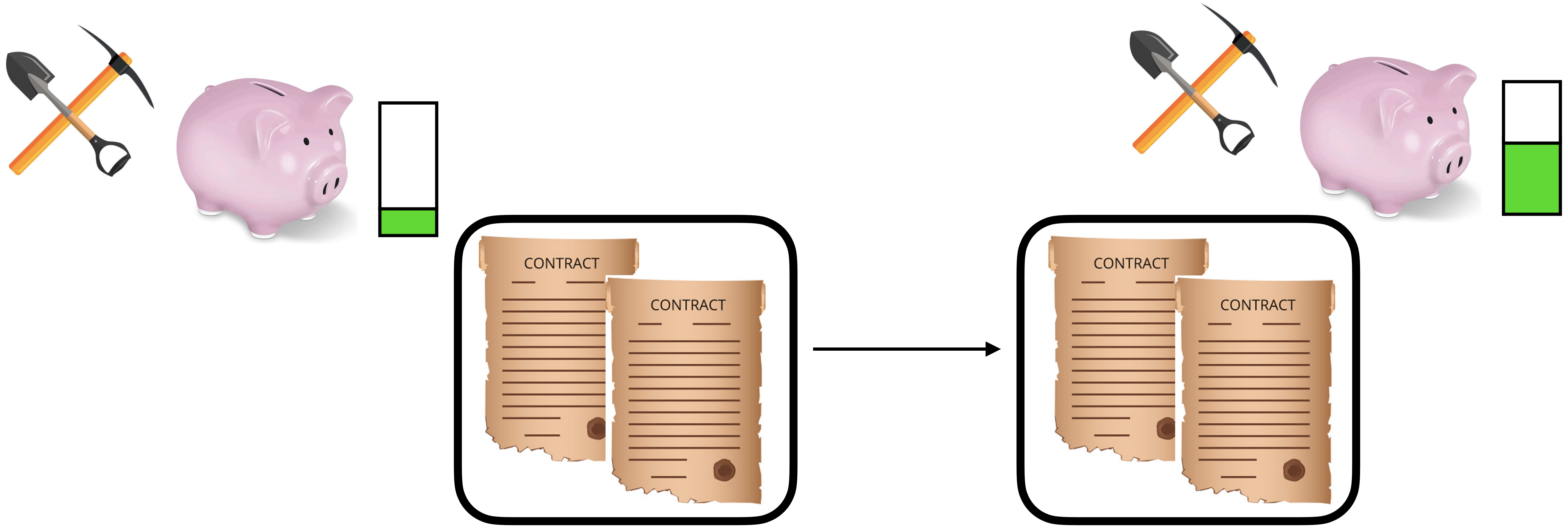
Extractable Value (EV)



Use MEV as the measure of economic security

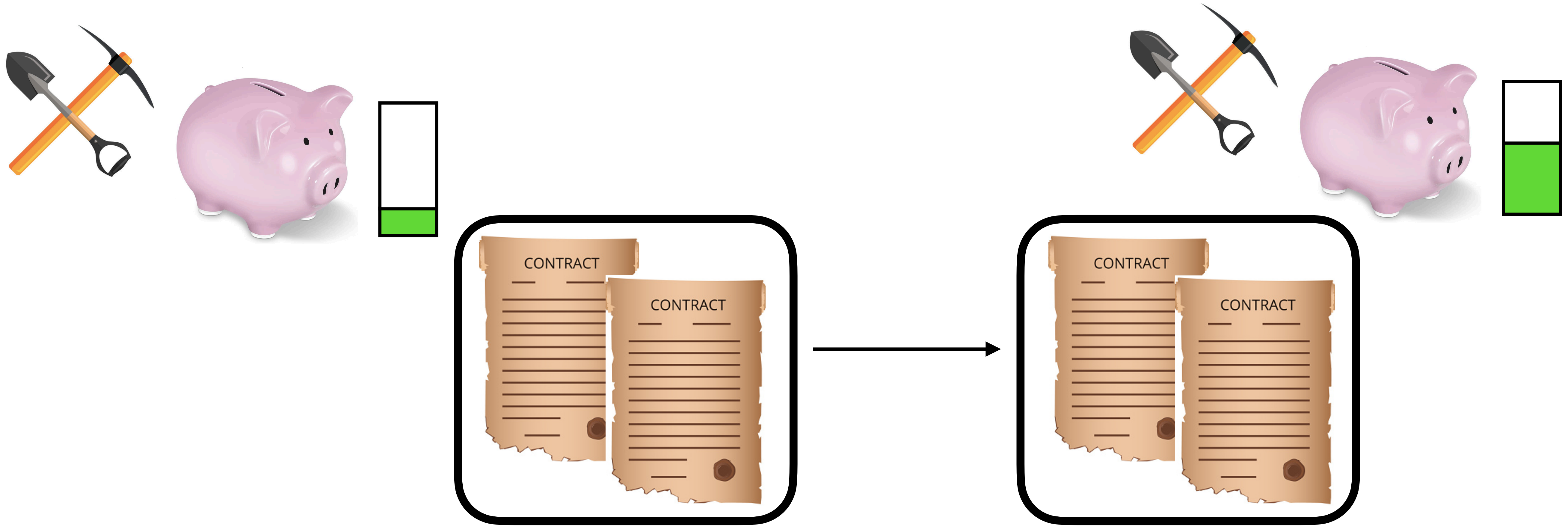
Miner is the most powerful out of all permissionless players - MEV subsumes all other attacks

Secure Composition



State S **Valid Block**

Secure Composition

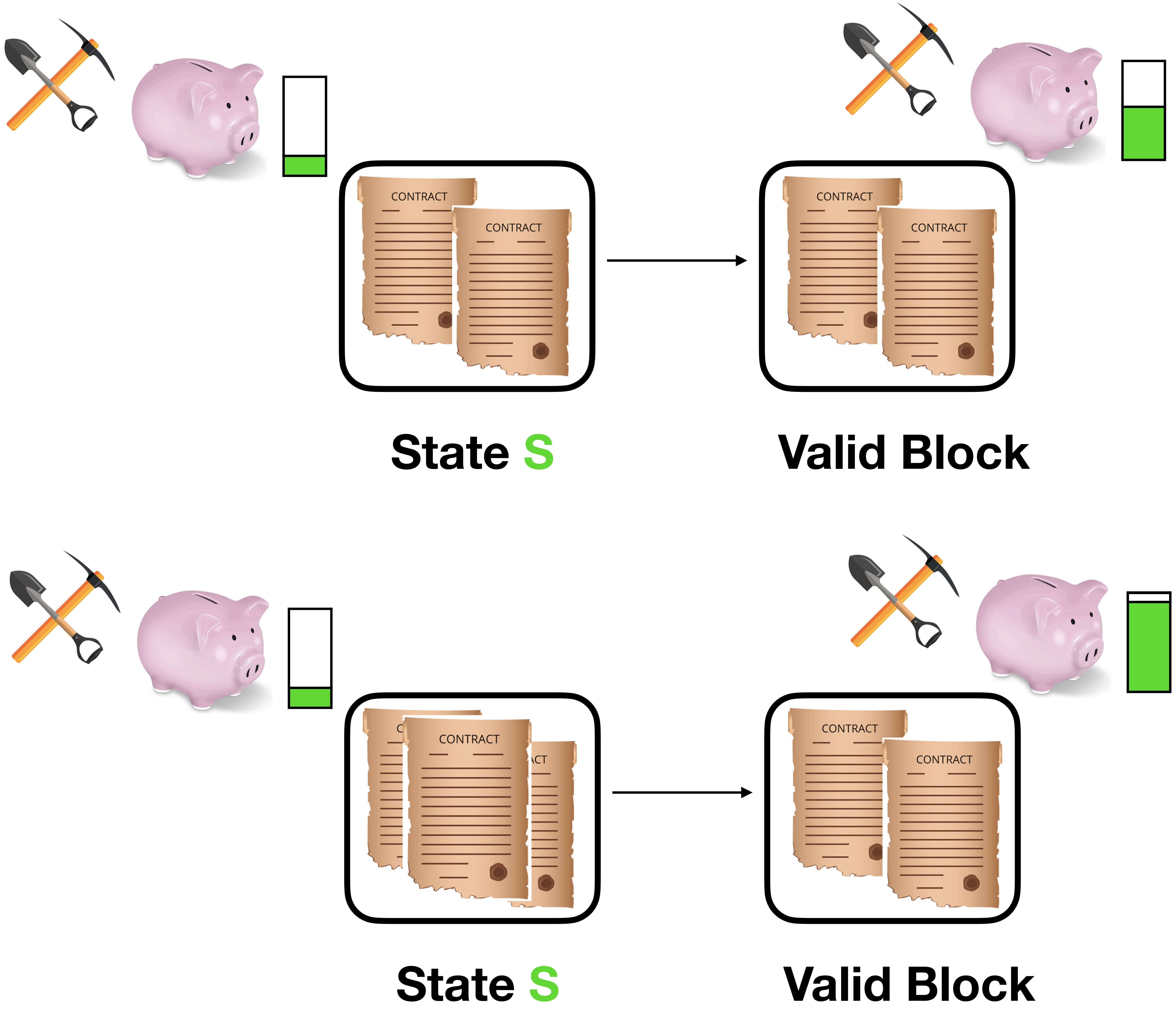


State S

Valid Block

Contract C

Secure Composition



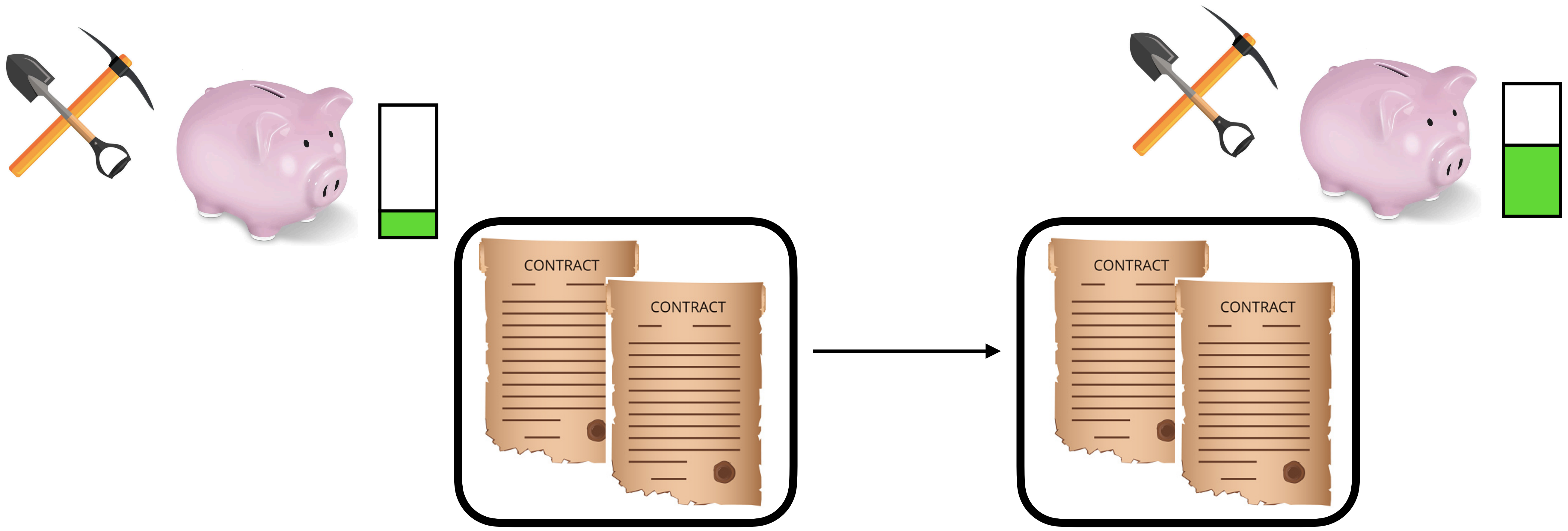
State S

Valid Block

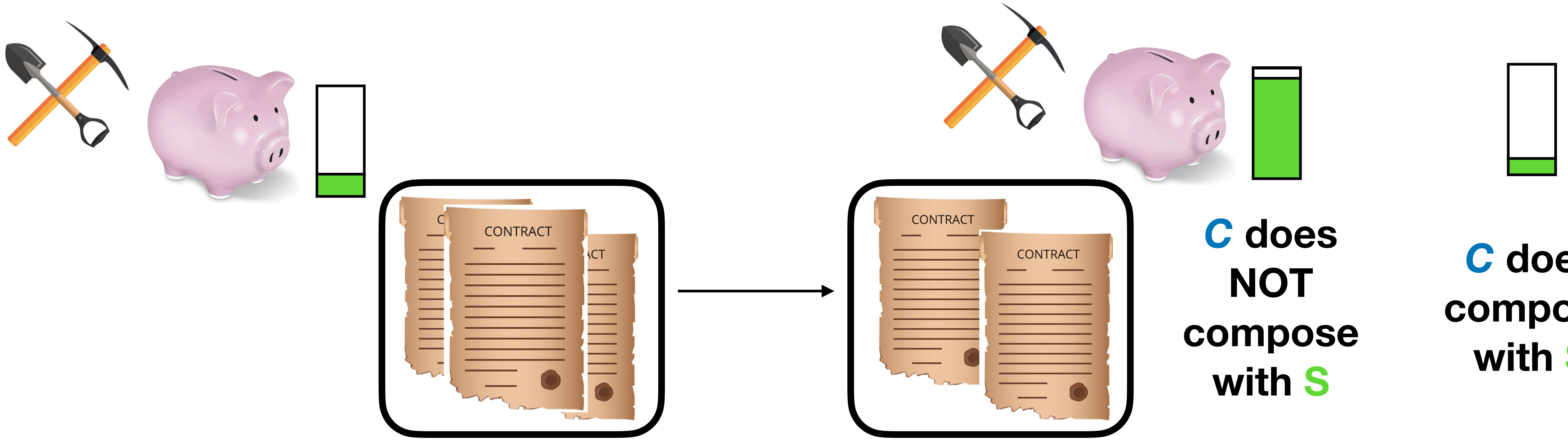
State S

Valid Block

Secure Composition



State S **Valid Block**



C does NOT compose with S

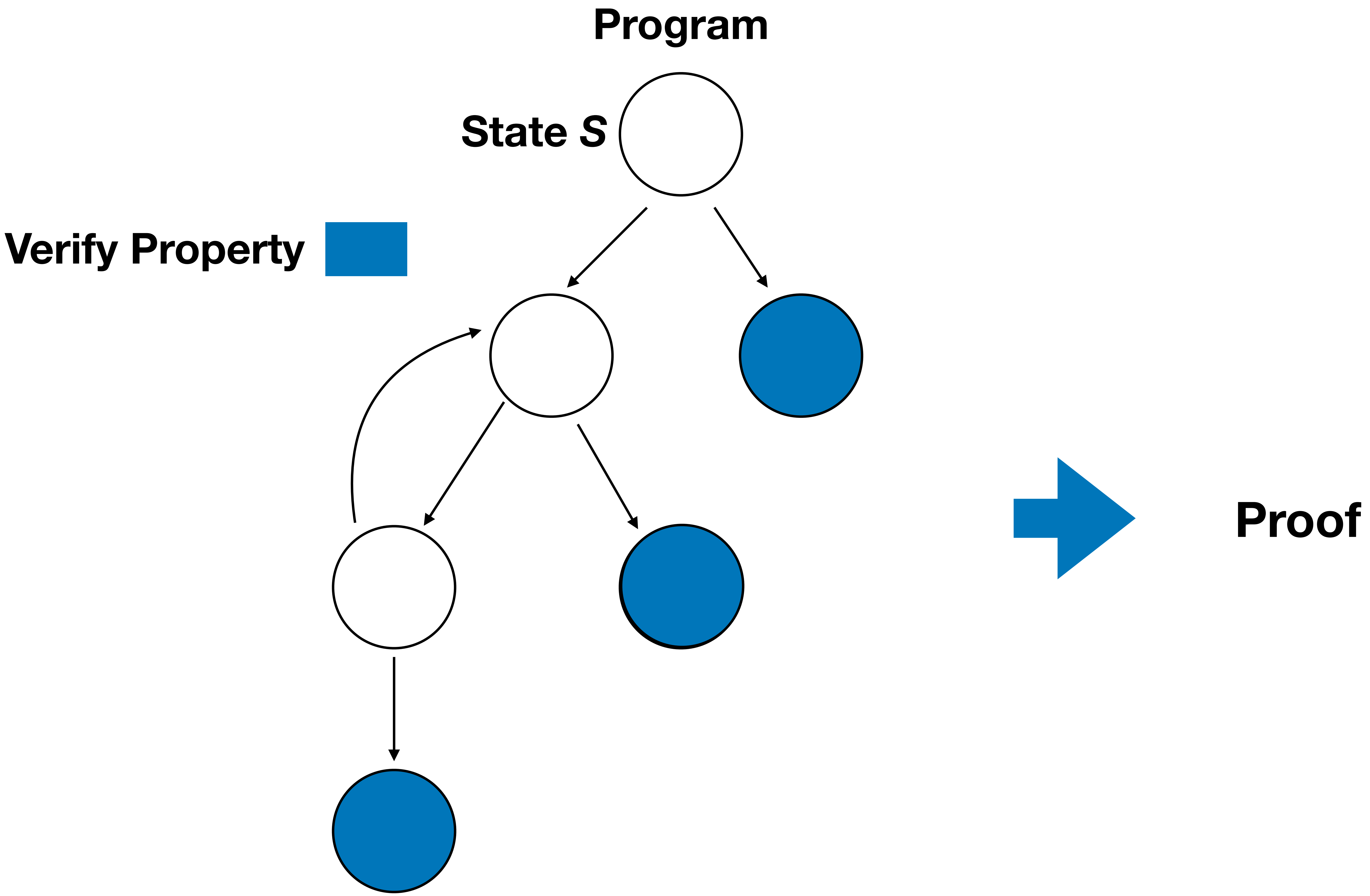
C does compose with S

State S’= S + C **Valid Block**

Outline

- Closer look at an example
- Definitional tools
    - Defining EV
    - Defining Secure Composition
- Practical Instantiation into Clockwork Finance Framework (CFF)
    - Design
    - Use for proofs
    - Use for finding attacks

Formal Verification



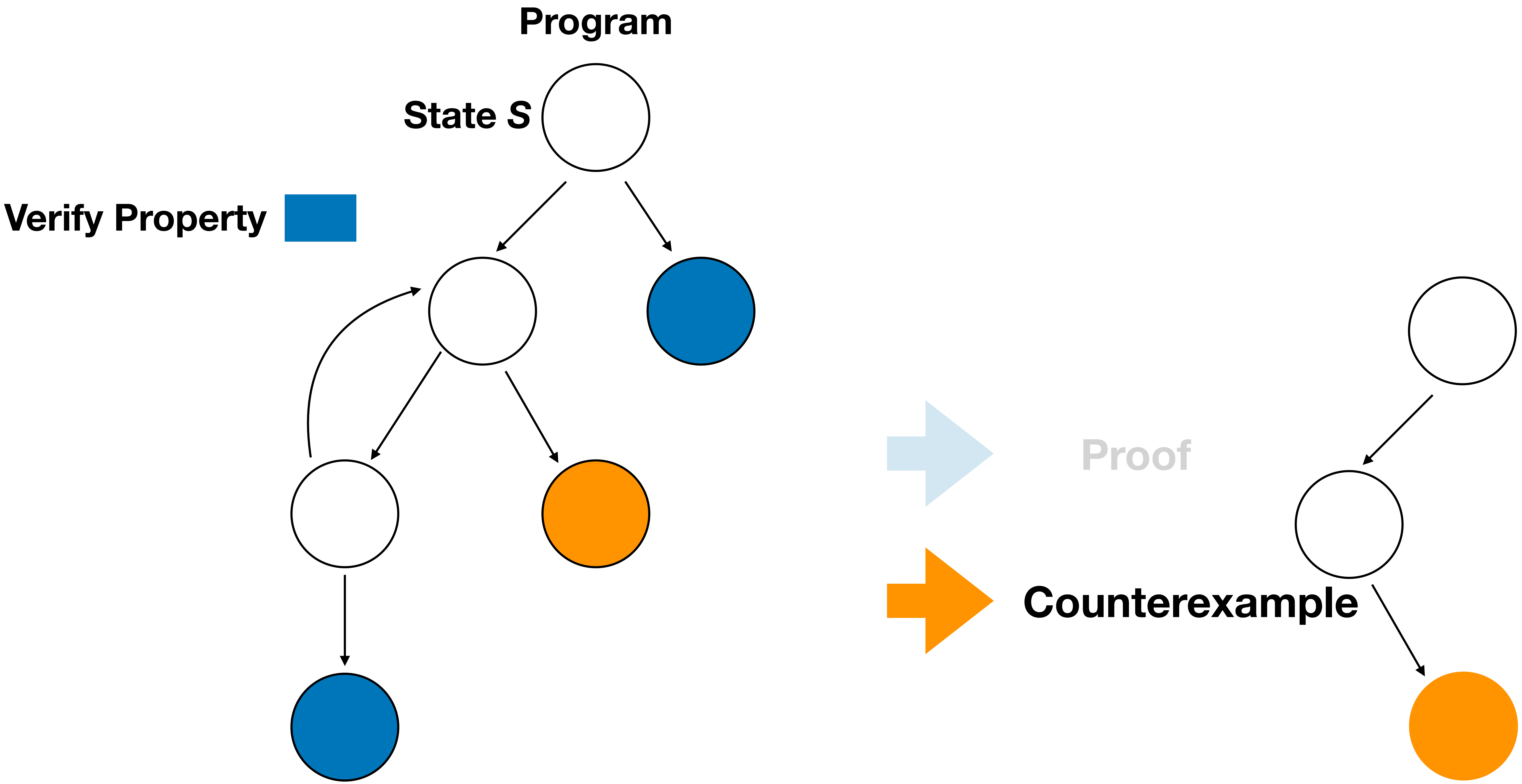
Program

State *S*

Verify Property 🟦

Proof

Formal Verification



Program

State *S*

Verify Property 🟦

Proof

Counterexample

Clockwork Finance Framework (CFF)

Clockwork Finance Framework (CFF)

State *S*

(Symbolic) Transactions = tx1, tx2, tx3

Verify Property: MEV < δ

Clockwork Finance Framework (CFF)

State *S*

(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

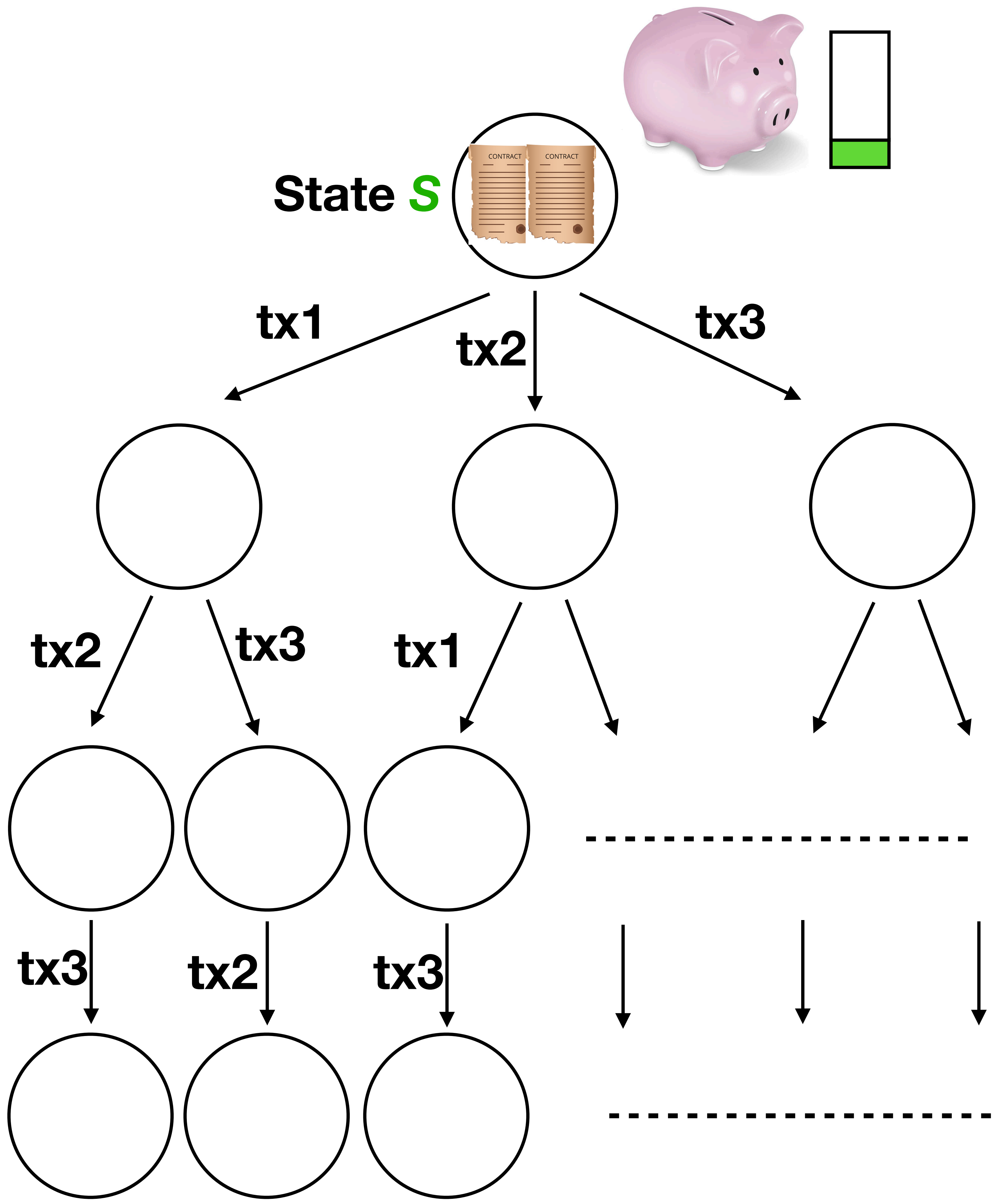
Verify Property: MEV < δ

Clockwork Finance Framework (CFF)

(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

Verify Property: MEV < δ

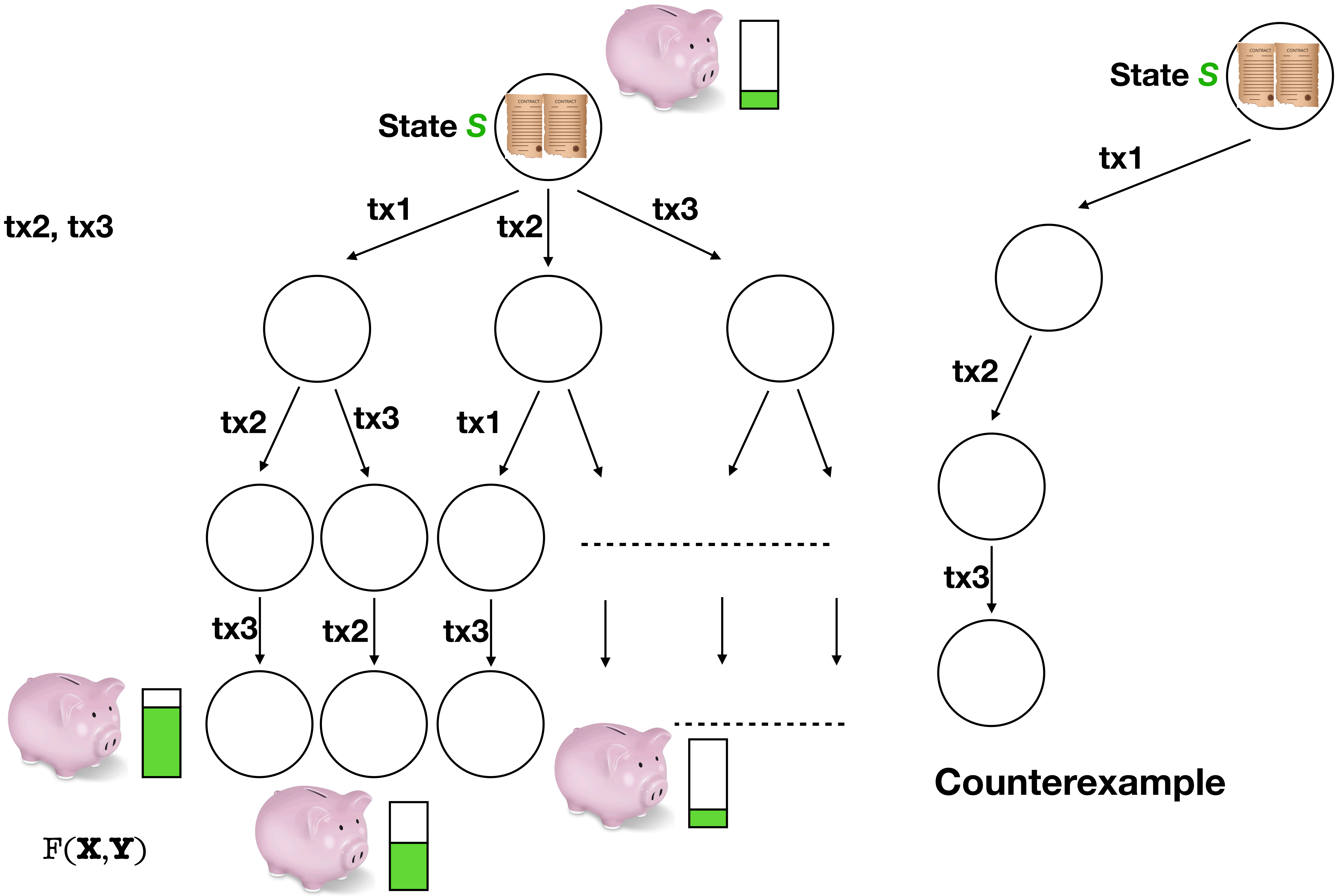


Clockwork Finance Framework (CFF)

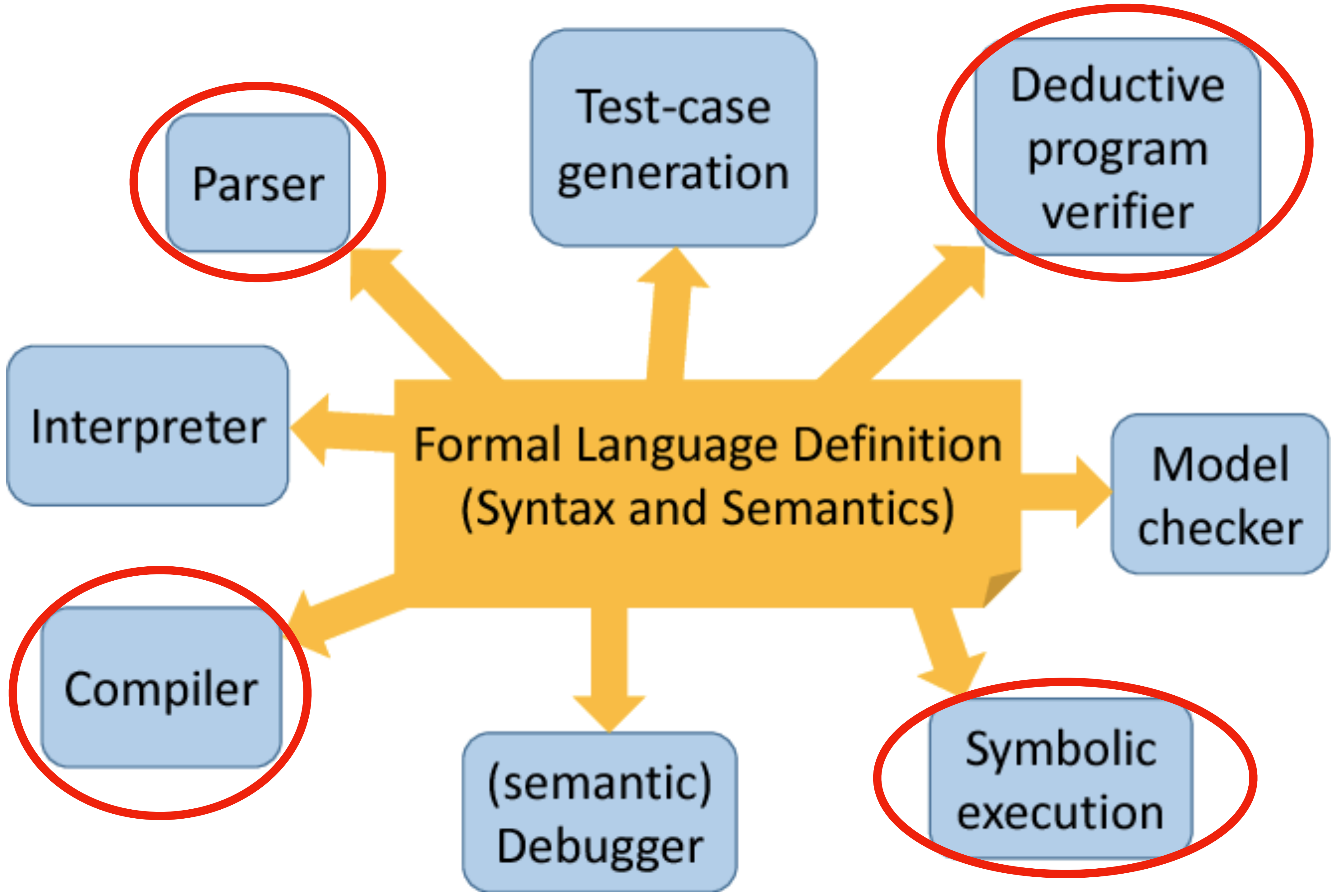
(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

Verify Property: MEV < δ

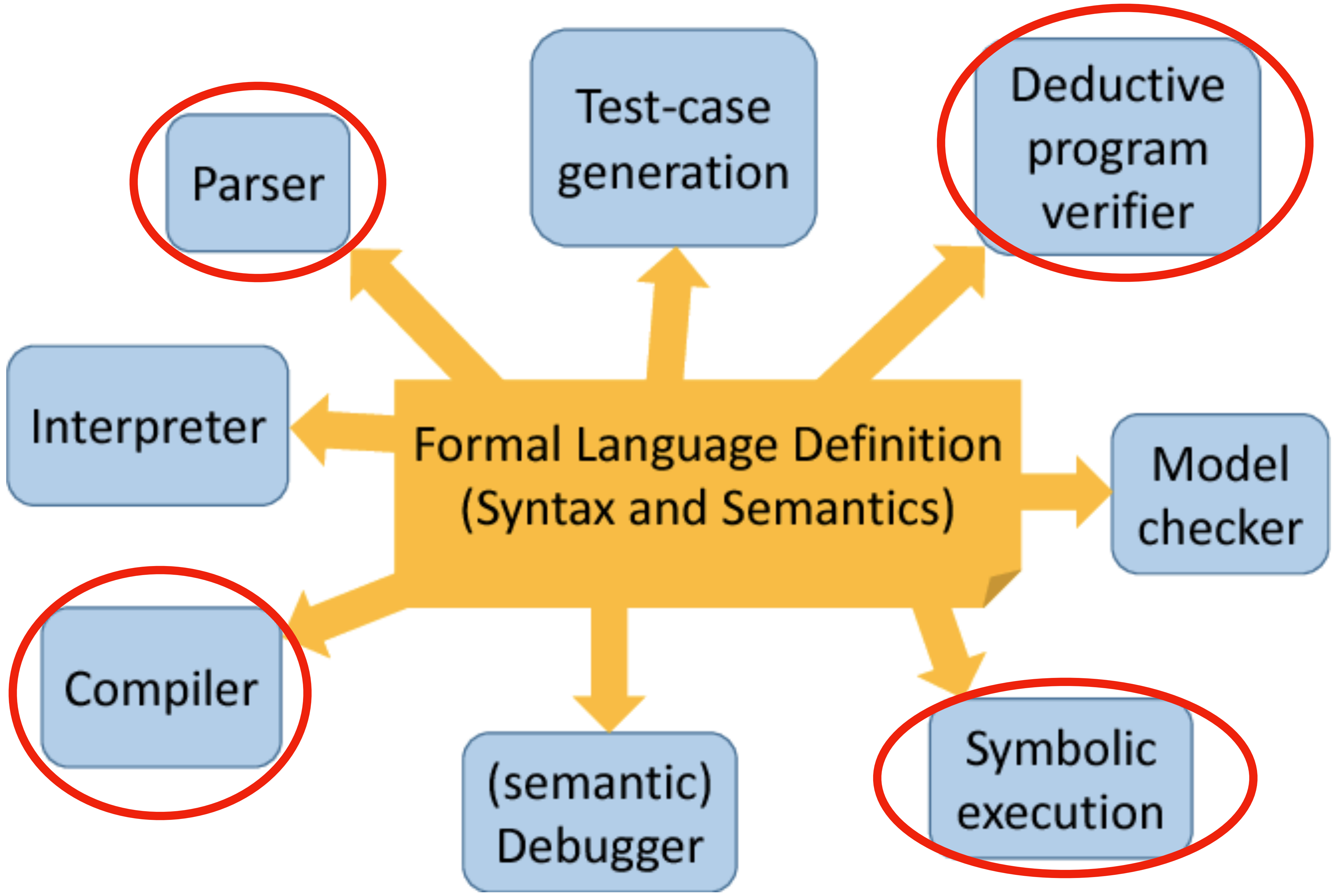


K Framework



Hildenbrandt, E., Saxena, M., Rodrigues, N., Zhu, X., Daian, P., Guth, D., Moore, B., Park, D., Zhang, Y., Stefanescu, A. and Rosu, G., 2018. Kevm: A complete formal semantics of the ethereum virtual machine. In *CSF'18*

K Framework

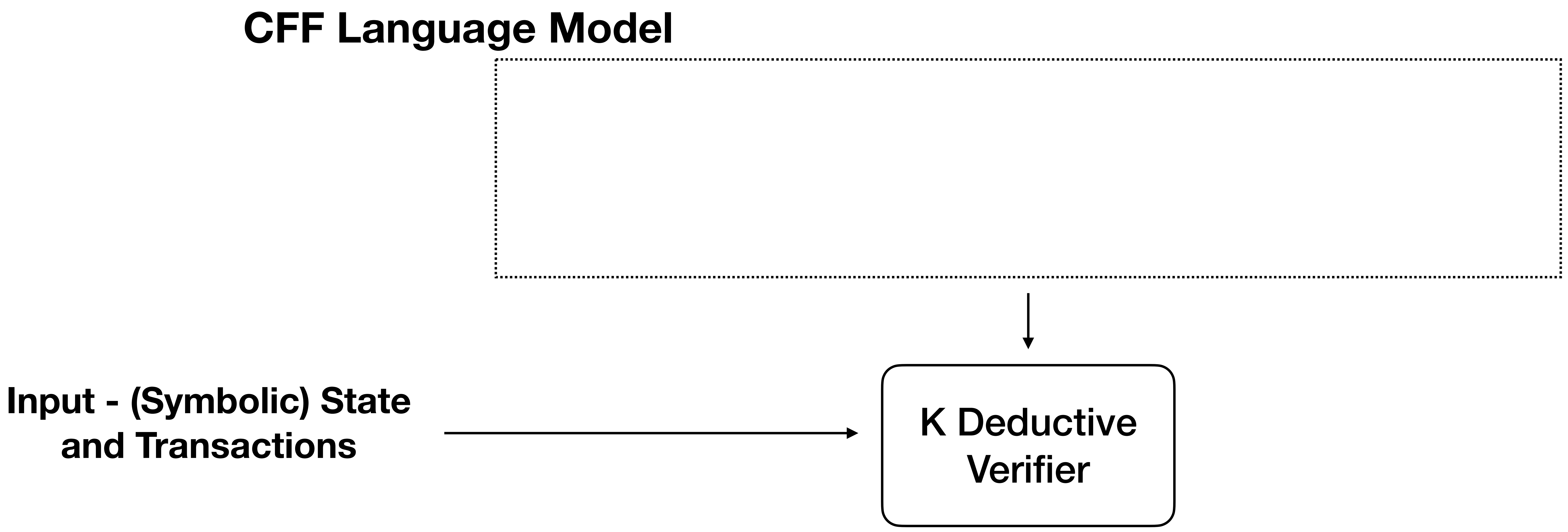


- Human Readable Formal Specification
- KEVM - Formal Ethereum Semantics in K

Hildenbrandt, E., Saxena, M., Rodrigues, N., Zhu, X., Daian, P., Guth, D., Moore, B., Park, D., Zhang, Y., Stefanescu, A. and Rosu, G., 2018. Kevm: A complete formal semantics of the ethereum virtual machine. In *CSF'18*

CFF Design

CFF Design

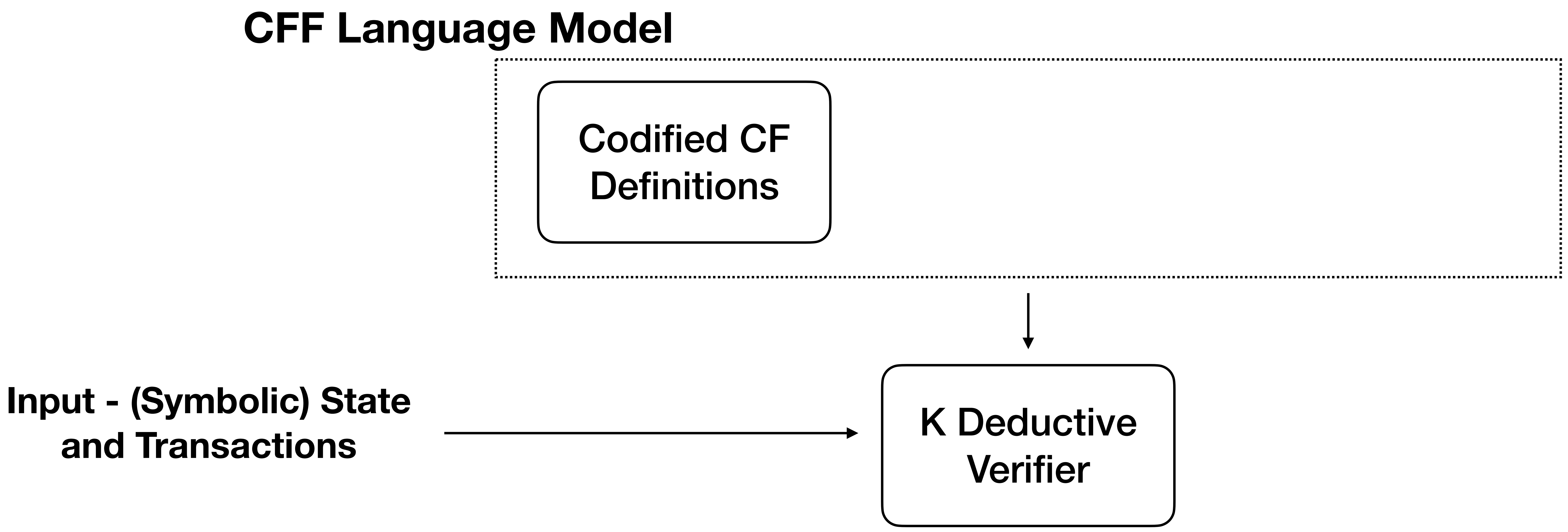


CFF Language Model

Input - (Symbolic) State and Transactions

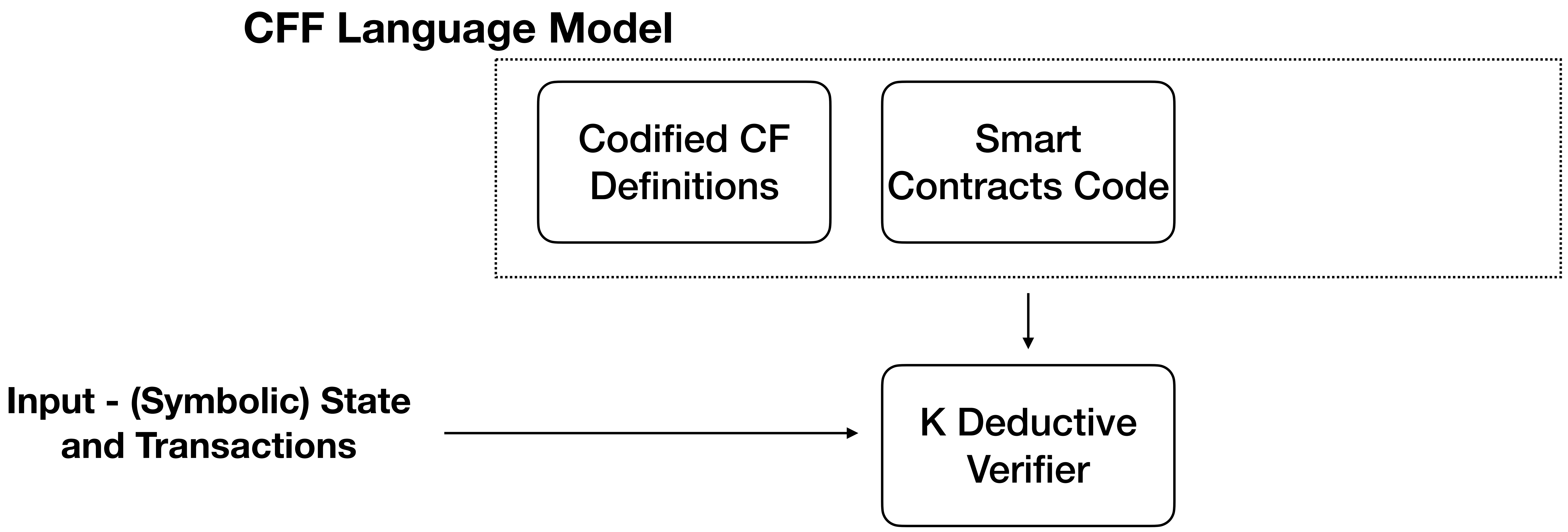
K Deductive Verifier

CFF Design



CFF Language Model
Codified CF Definitions
Input - (Symbolic) State and Transactions
K Deductive Verifier

CFF Design



CFF Language Model

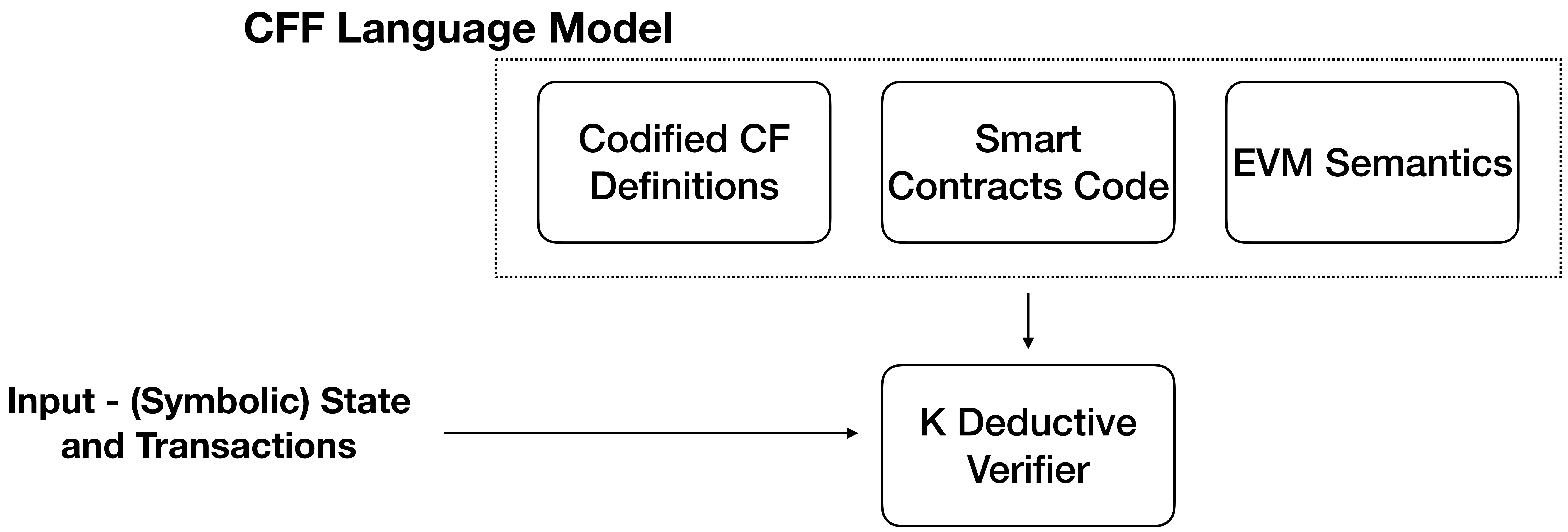
Codified CF Definitions

Smart Contracts Code

Input - (Symbolic) State and Transactions

K Deductive Verifier

CFF Design



CFF Language Model

Codified CF Definitions

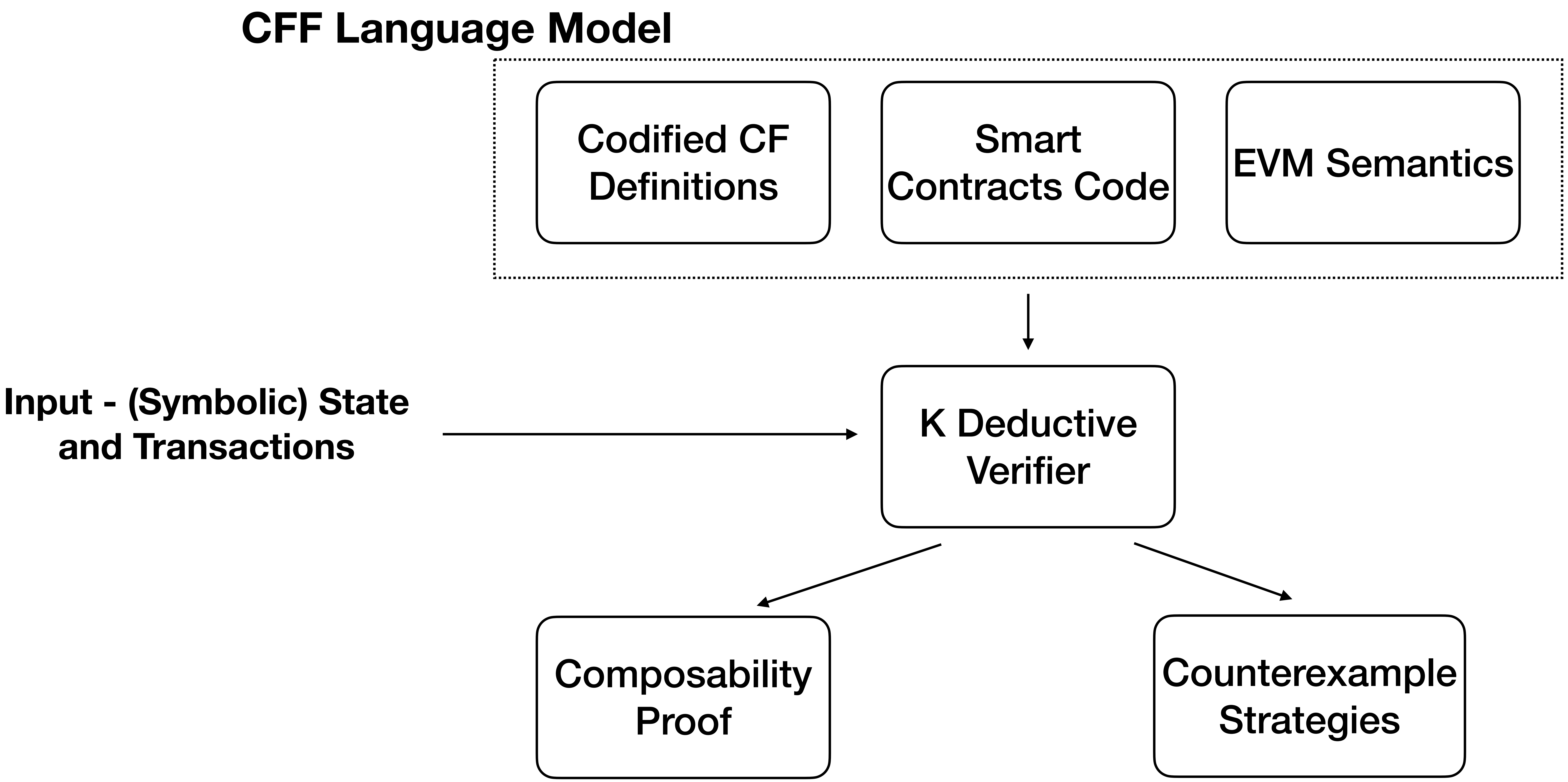
Smart Contracts Code

EVM Semantics

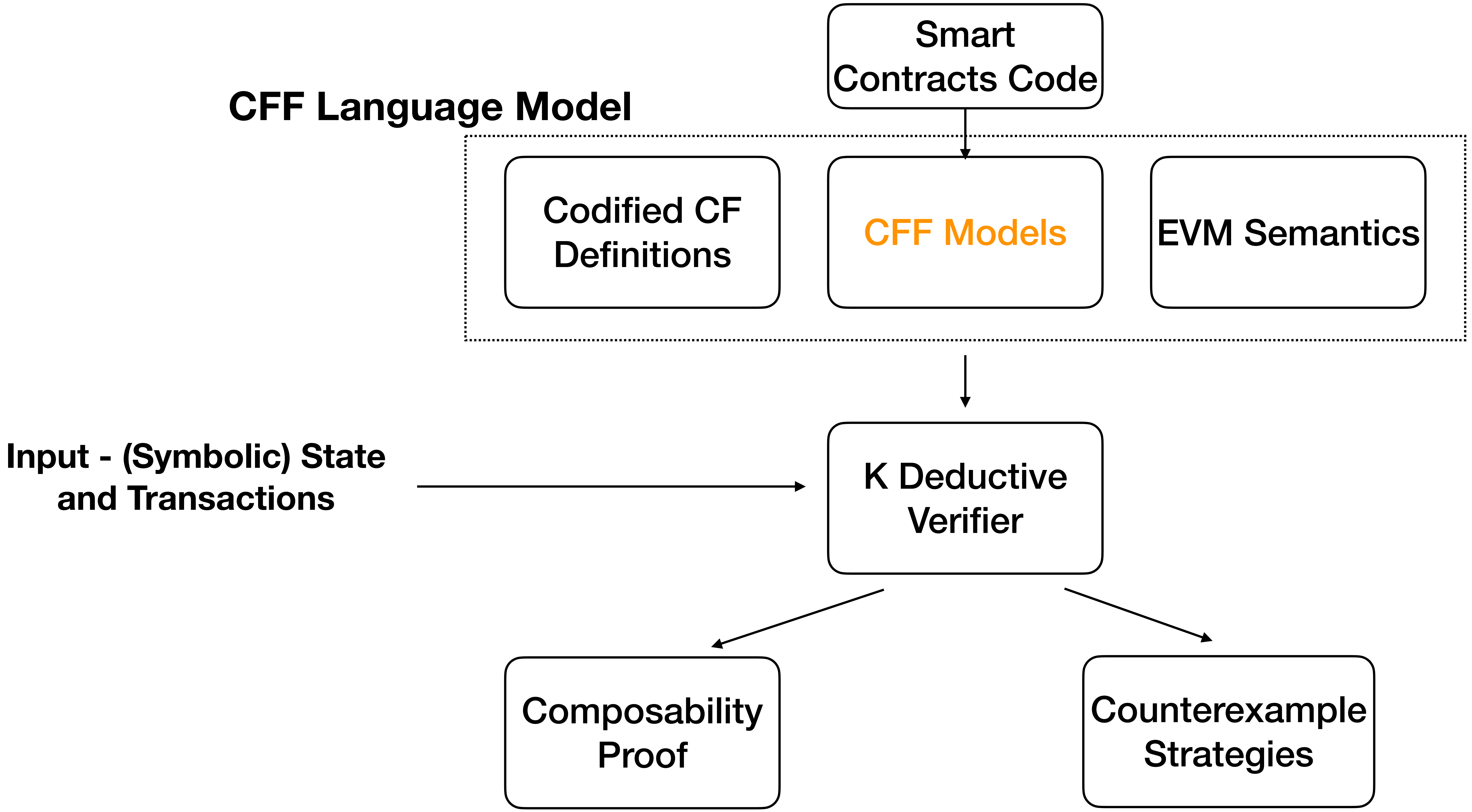
Input - (Symbolic) State and Transactions

K Deductive Verifier

CFF Design


CFF Language Model
Codified CF Definitions
Smart Contracts Code
EVM Semantics
Input - (Symbolic) State and Transactions
K Deductive Verifier
Composability Proof
Counterexample Strategies

CFF Design

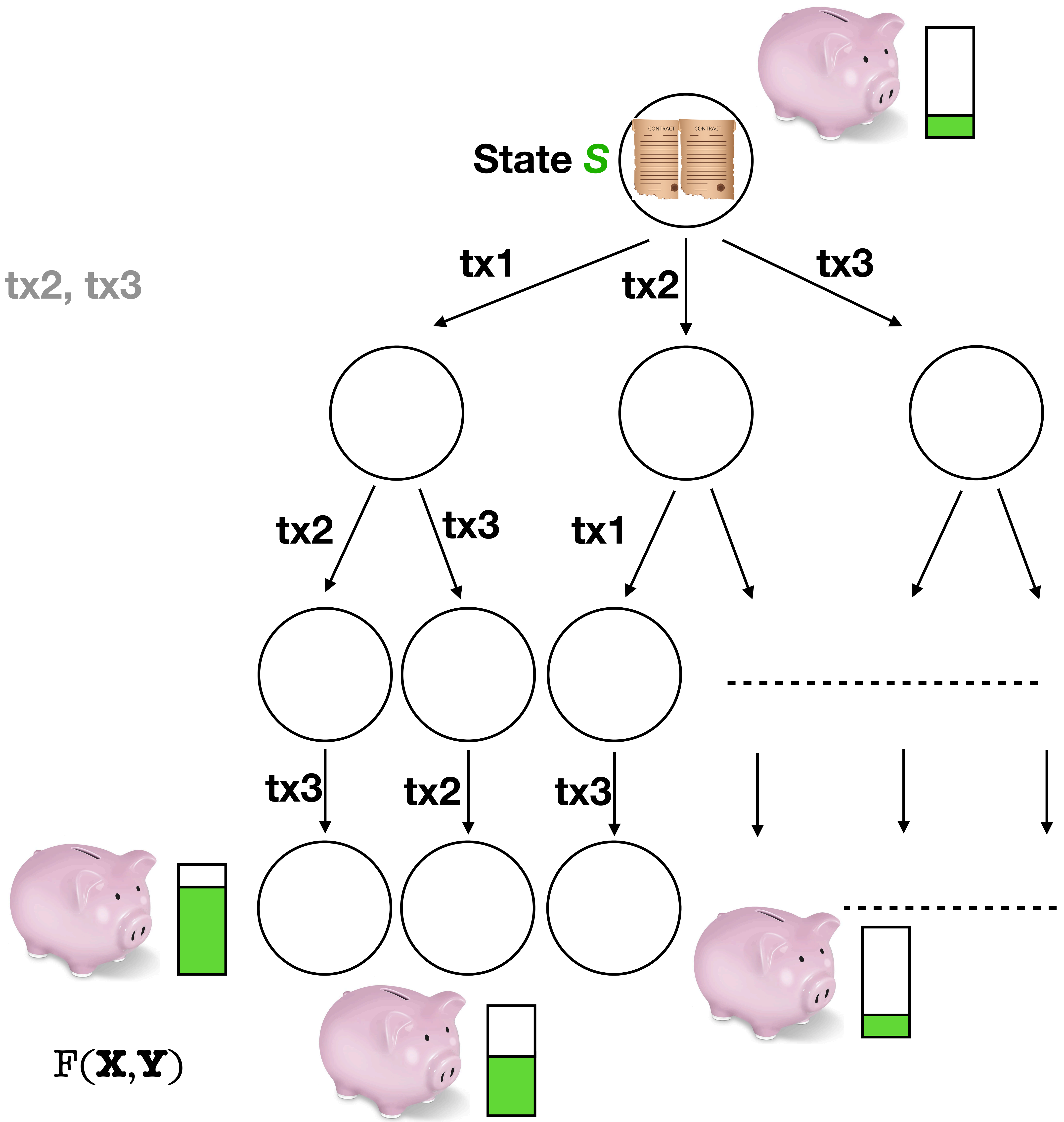

Smart Contracts Code
CFF Language Model
Codified CF Definitions
CFF Models
EVM Semantics
Input - (Symbolic) State and Transactions
K Deductive Verifier
Composability Proof
Counterexample Strategies

CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

Verify Property: MEV < δ

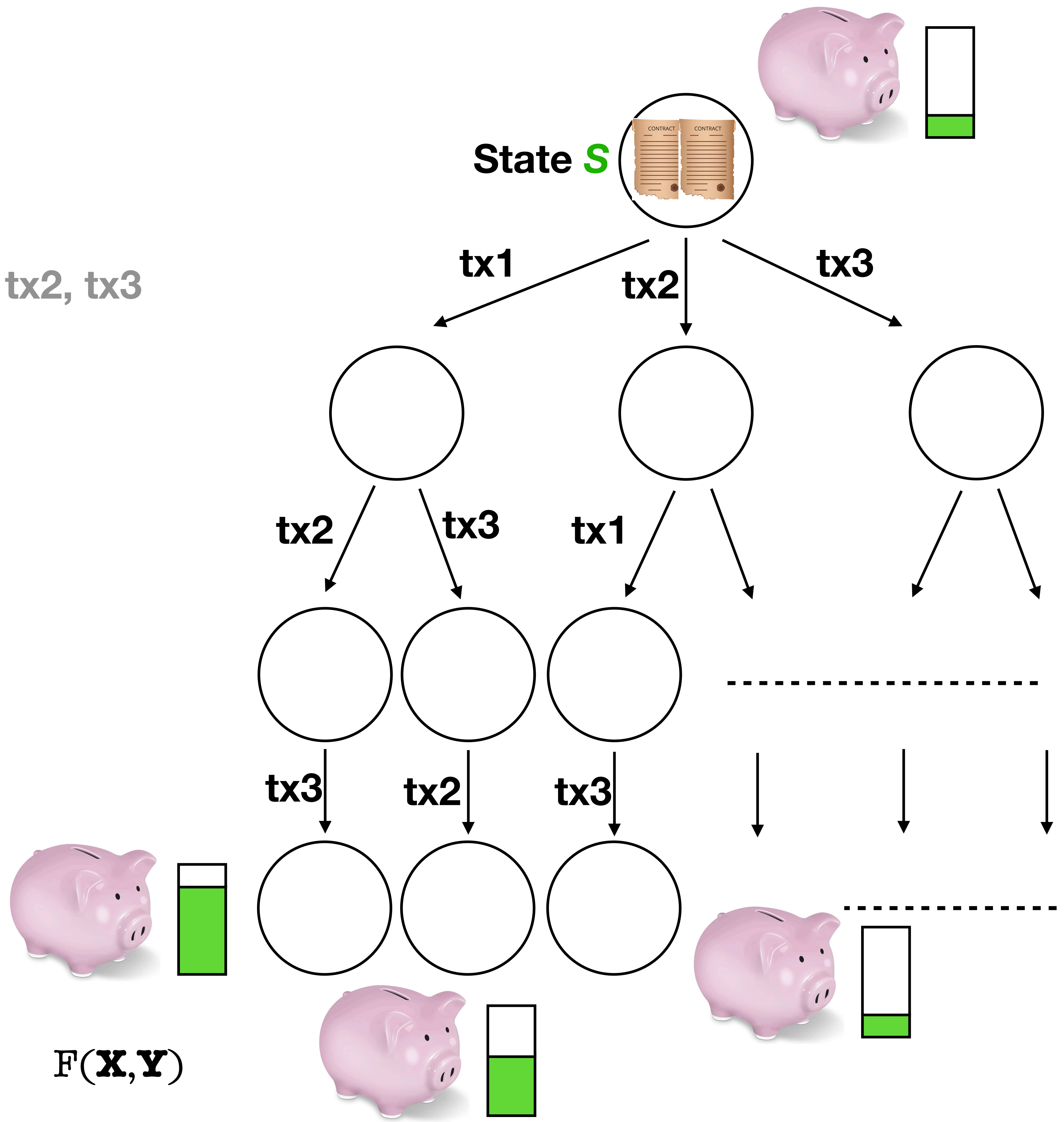


CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

Verify Property: MEV < δ



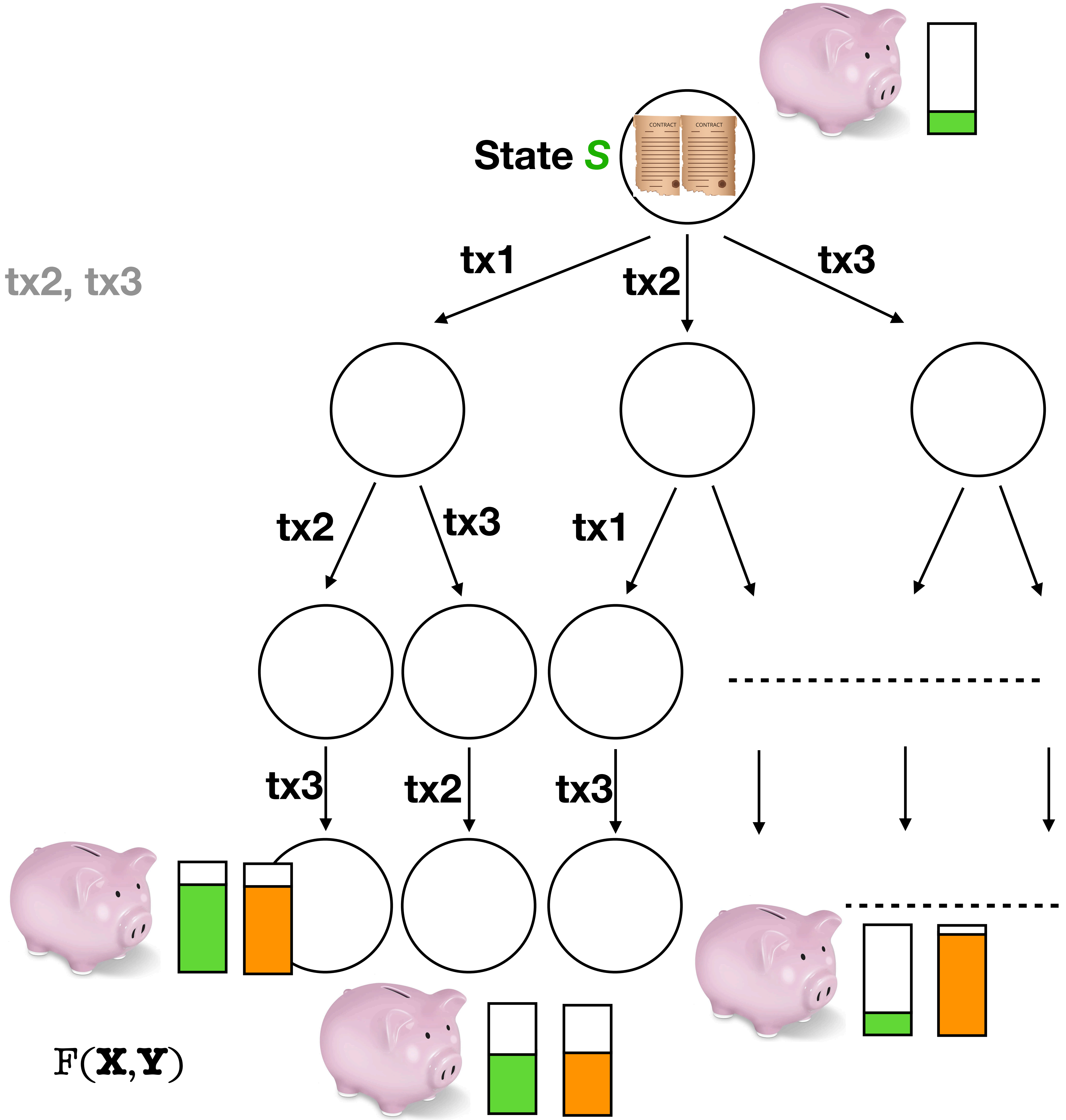
CFF models are over approximations of the smart contract code.

CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap $\mathbf{X}$ Eth for $\mathbf{Y}$ USD
$\mathbf{X} >= 0$, $\mathbf{Y} >= 0$

Verify Property: MEV < δ



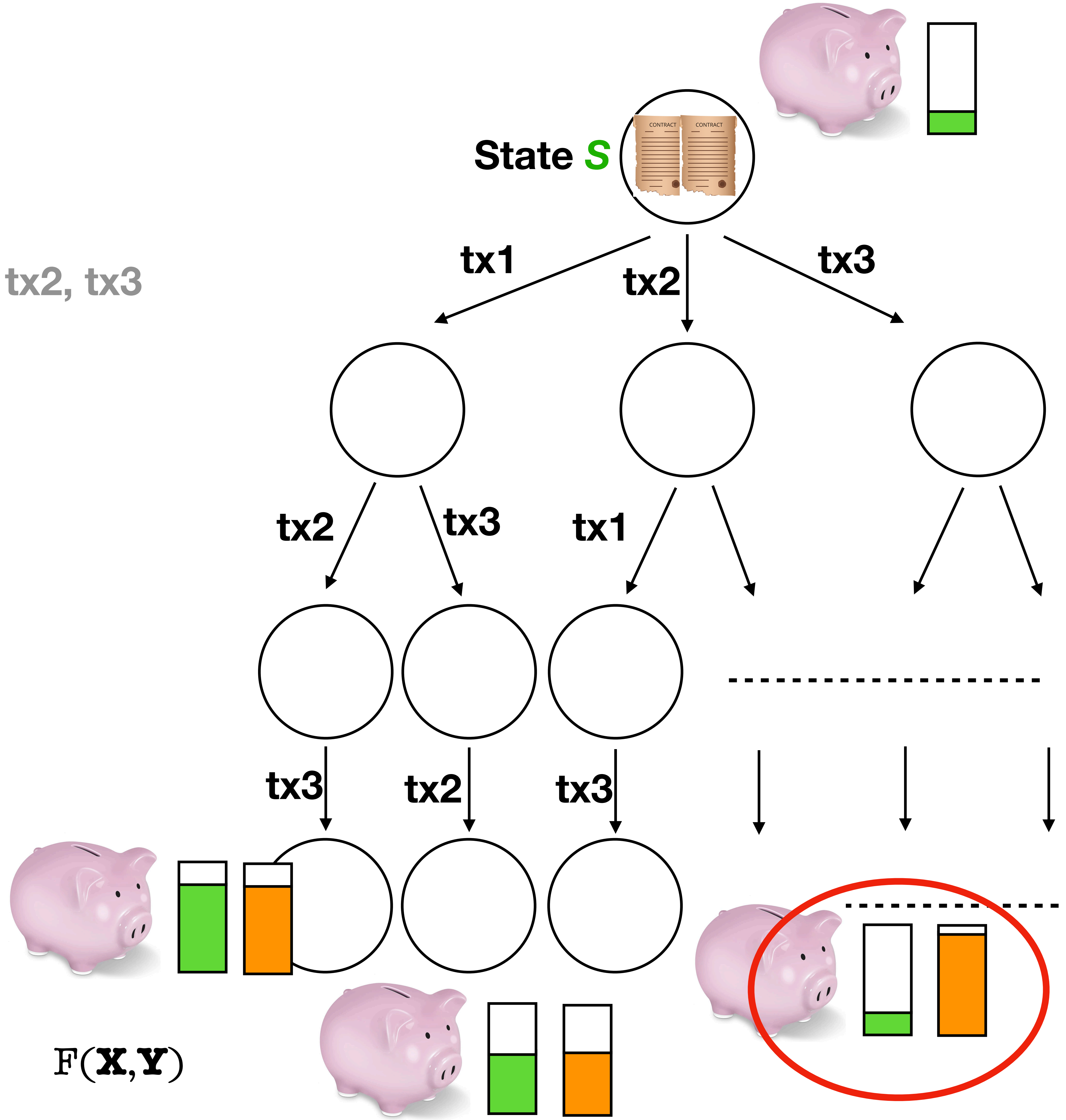
CFF models are over approximations of the smart contract code.

CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

Verify Property: MEV < δ



CFF models are over approximations of the smart contract code.

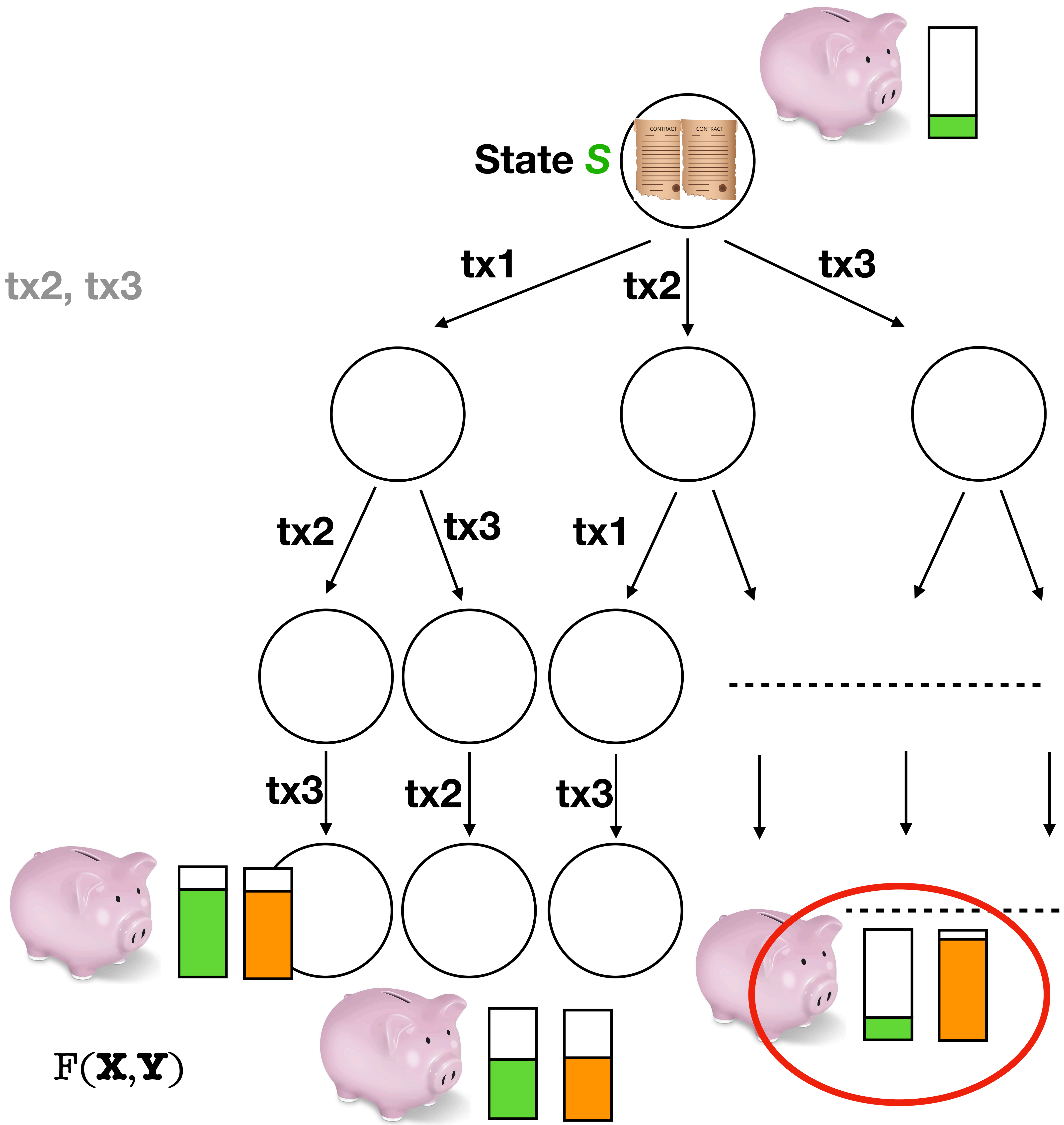
False Positive

CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD
X >= 0, **Y** >=0

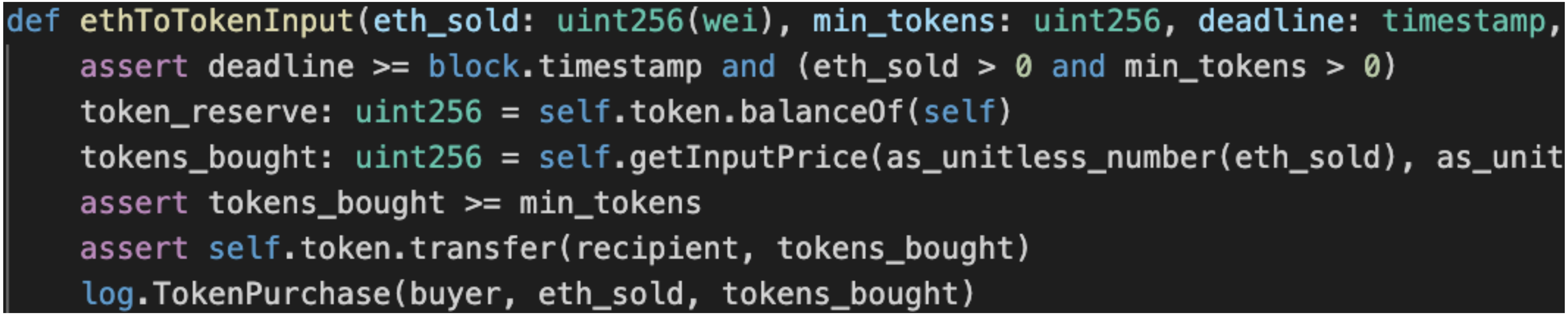
Verify Property: MEV < δ



CFF models are over approximations of the smart contract code.

False Positive
But No False Negatives

CFF Models



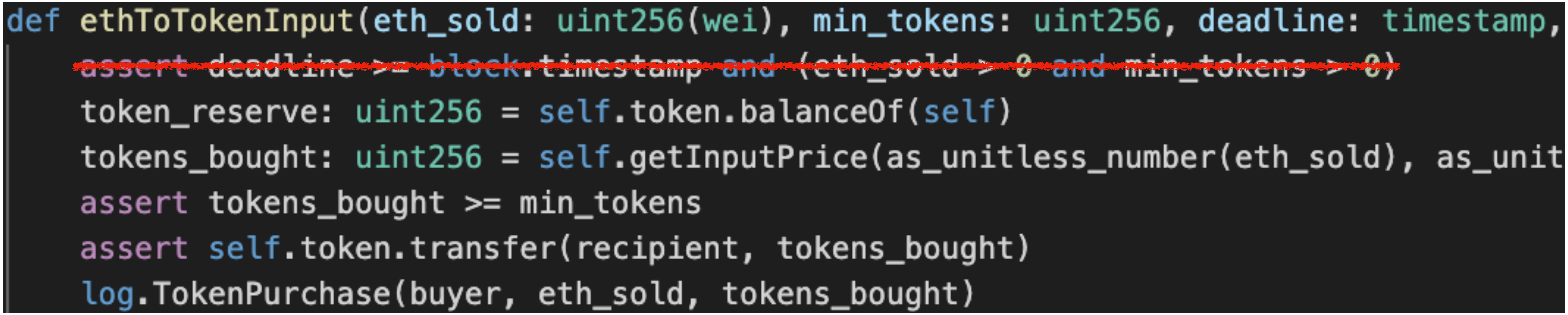
```
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp,
    assert deadline >= block.timestamp and (eth_sold > 0 and min_tokens > 0)
    token_reserve: uint256 = self.token.balanceOf(self)
    tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unit
    assert tokens_bought >= min_tokens
    assert self.token.transfer(recipient, tokens_bought)
    log.TokenPurchase(buyer, eth_sold, tokens_bought)
```

Process : Manual translation by pruning irrelevant code paths.

Becomes easier if the contract has been verified formally for functional correctness

Open sourced CFF models for UniswapV1, UniswapV2, MakerDAO, FlashLoans, Airdrops

CFF Models



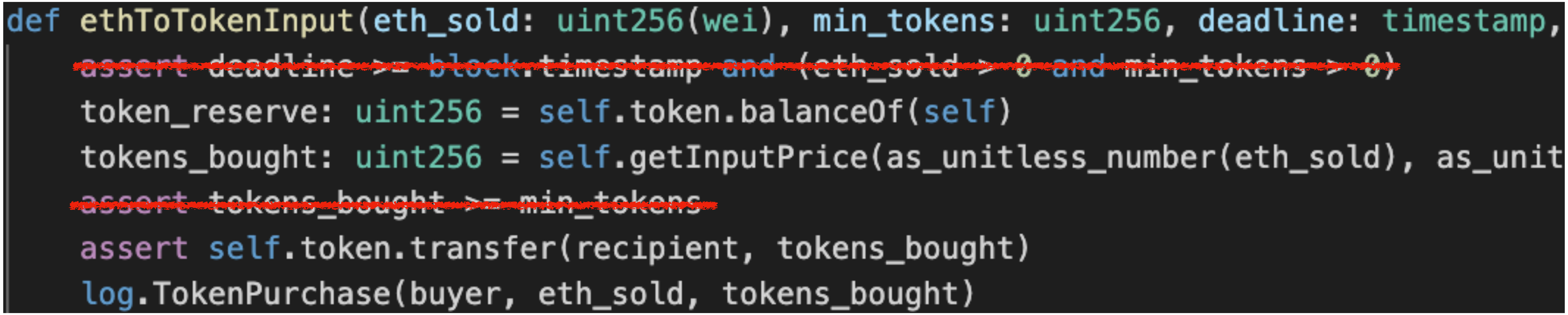
```
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp,
    ~~assert deadline >= block.timestamp and (eth_sold > 0 and min_tokens > 0)~~
    token_reserve: uint256 = self.token.balanceOf(self)
    tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unit
    assert tokens_bought >= min_tokens
    assert self.token.transfer(recipient, tokens_bought)
    log.TokenPurchase(buyer, eth_sold, tokens_bought)
```

Process : Manual translation by pruning irrelevant code paths.

Becomes easier if the contract has been verified formally for functional correctness

Open sourced CFF models for UniswapV1, UniswapV2, MakerDAO, FlashLoans, Airdrops

CFF Models



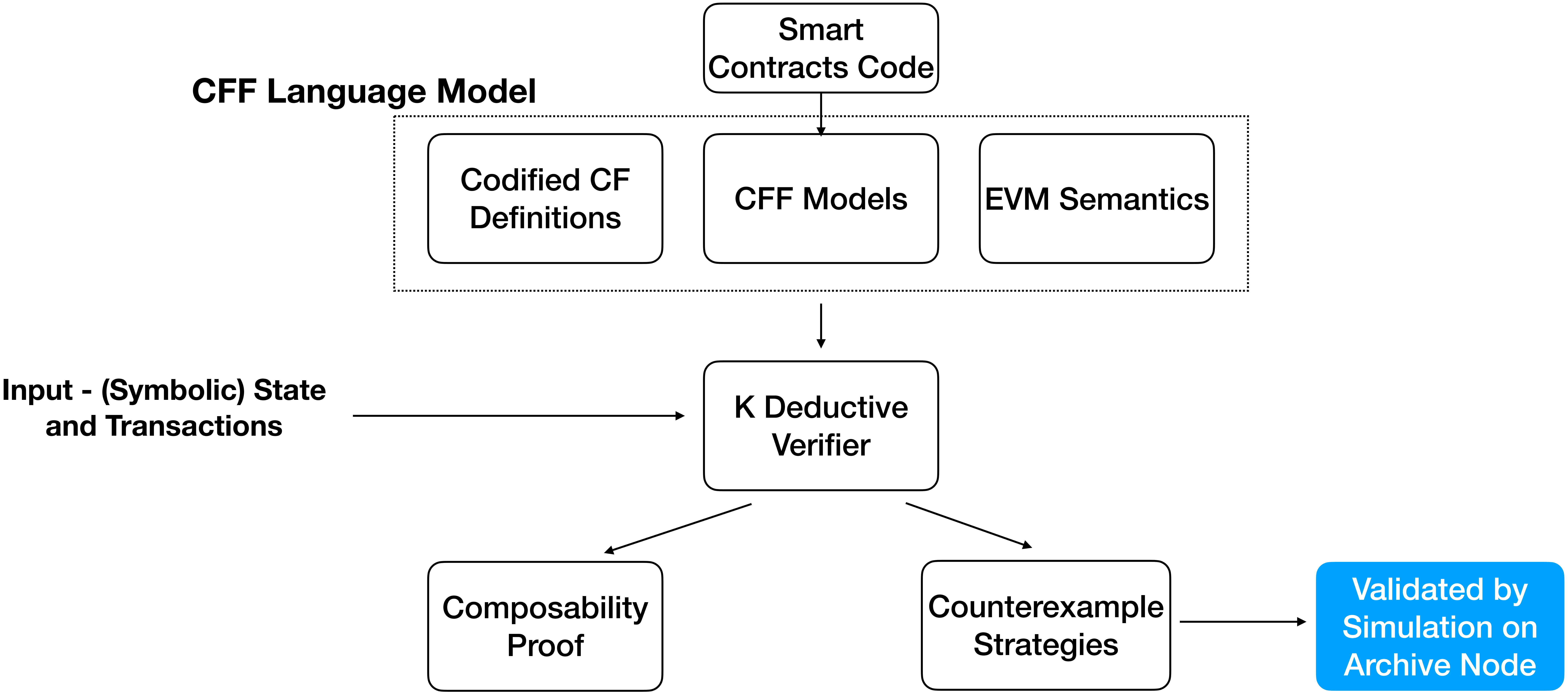
```
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp,
    ~~assert deadline >= block.timestamp and (eth_sold > 0 and min_tokens > 0)~~
    token_reserve: uint256 = self.token.balanceOf(self)
    tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unit
    ~~assert tokens_bought >= min_tokens~~
    assert self.token.transfer(recipient, tokens_bought)
    log.TokenPurchase(buyer, eth_sold, tokens_bought)
```

Process : Manual translation by pruning irrelevant code paths.

Becomes easier if the contract has been verified formally for functional correctness

Open sourced CFF models for UniswapV1, UniswapV2, MakerDAO, FlashLoans, Airdrops

CFF Design



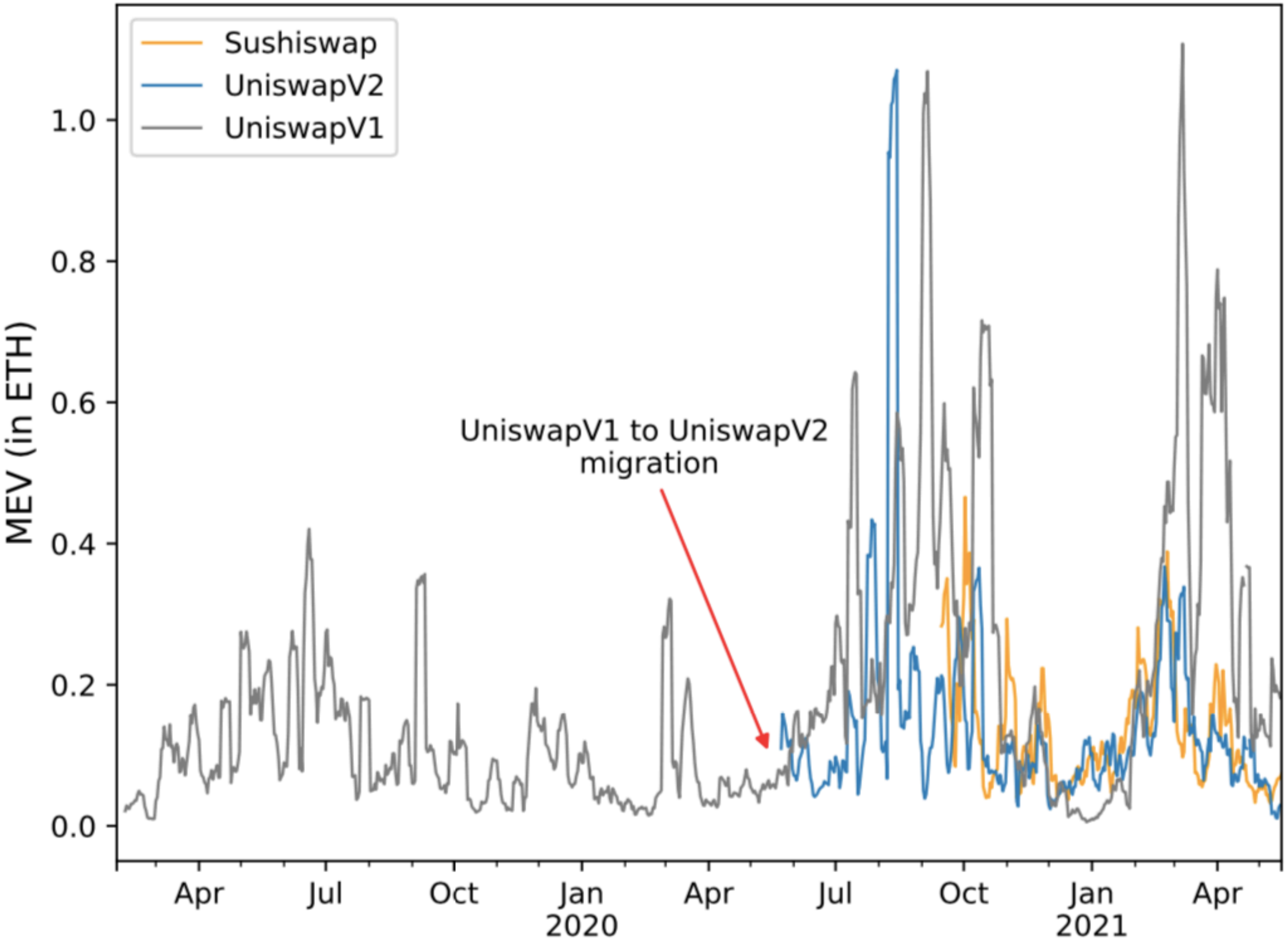
Smart Contracts Code
CFF Language Model
Codified CF Definitions
CFF Models
EVM Semantics
Input - (Symbolic) State and Transactions
K Deductive Verifier
Composability Proof
Counterexample Strategies
Validated by Simulation on Archive Node

More Scaling Optimisations

1. General Optimisations
    1. Transactions for a sender need to be serialised using “nonces”. Many invalid orderings are equivalent
    2. Reorderings across different non interacting contracts are equivalent
    3. Randomised reorderings lead to better convergence in practice.
2. Contract Specific Optimisations
    1. Uniswap-like AMMs are path independent

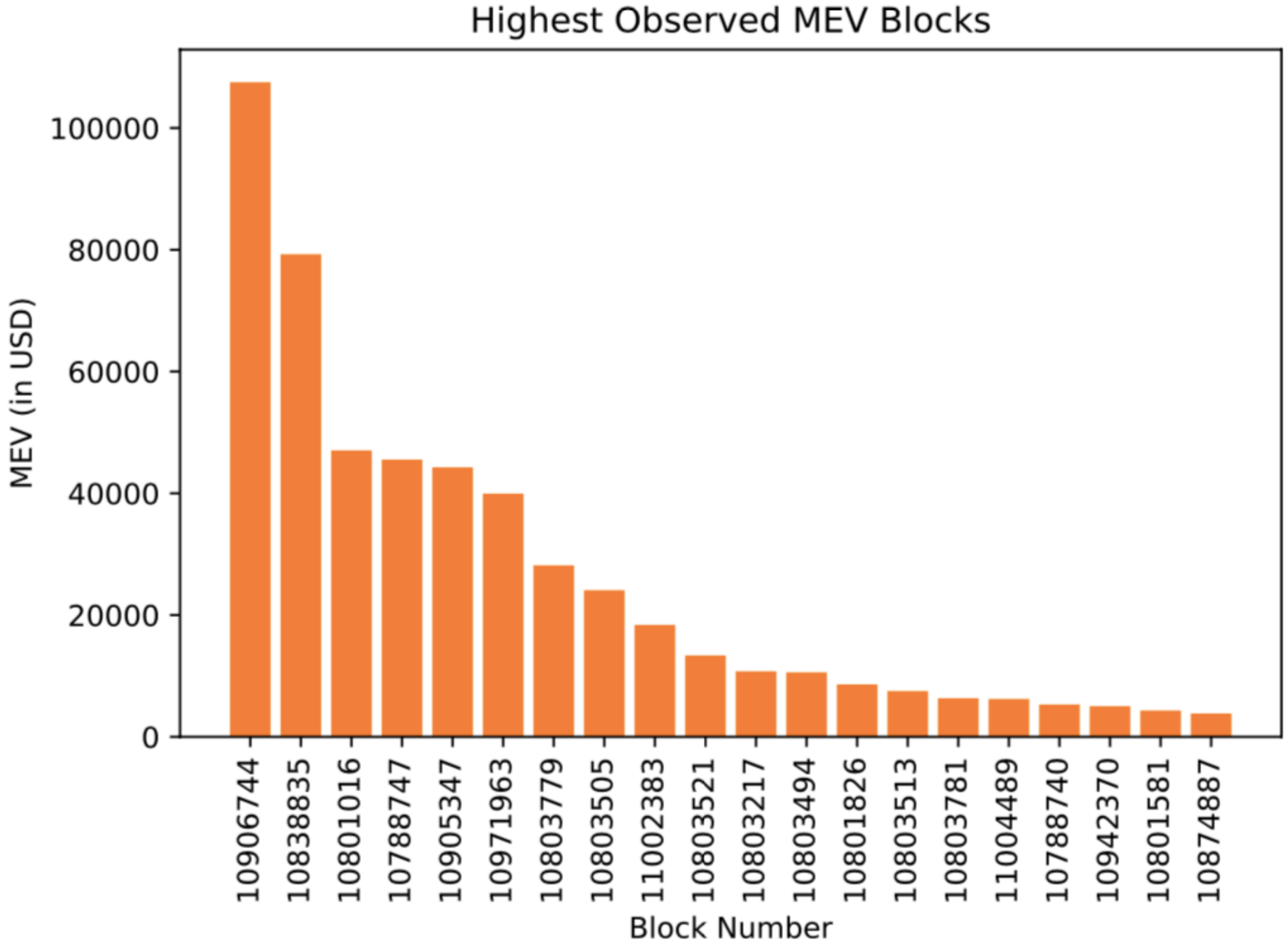
CFF Evaluation - AMM

7-day moving average of MEV per block in a sample of 1000 random blocks in each month



CFF Evaluation - Maker + Uniswap

Uniswap price used as oracle in Maker



CFF Model for Maker abstracts out liquidation auction

CFF Evaluation

Many More in the paper…
Governance, Flashloans, Airdrops

Conclusion

- Initiated the formal study of economic behaviour of smart contracts through the lens of MEV
    - Definitions for MEV and Secure Composition
    - Clockwork Finance Framework (CFF) : **Practical Proof System** based on Formal Verification
- Developers can use CFF to generate proofs of bounds on the MEV exposed by their contracts, and users can use CFF to analyse the MEV extractable from their transactions.

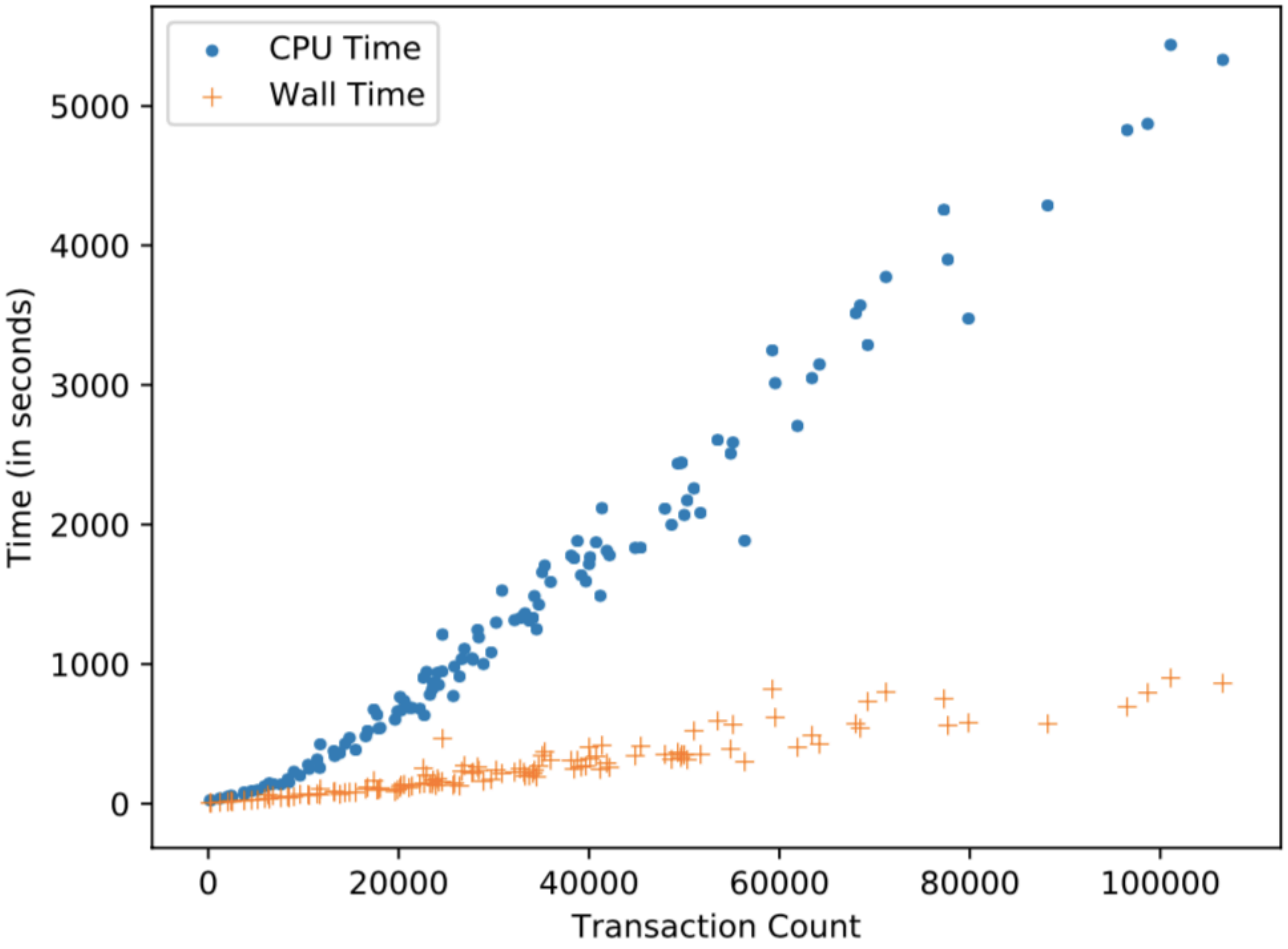
Paper : https://cs.cornell.edu/~babel/cff.pdf

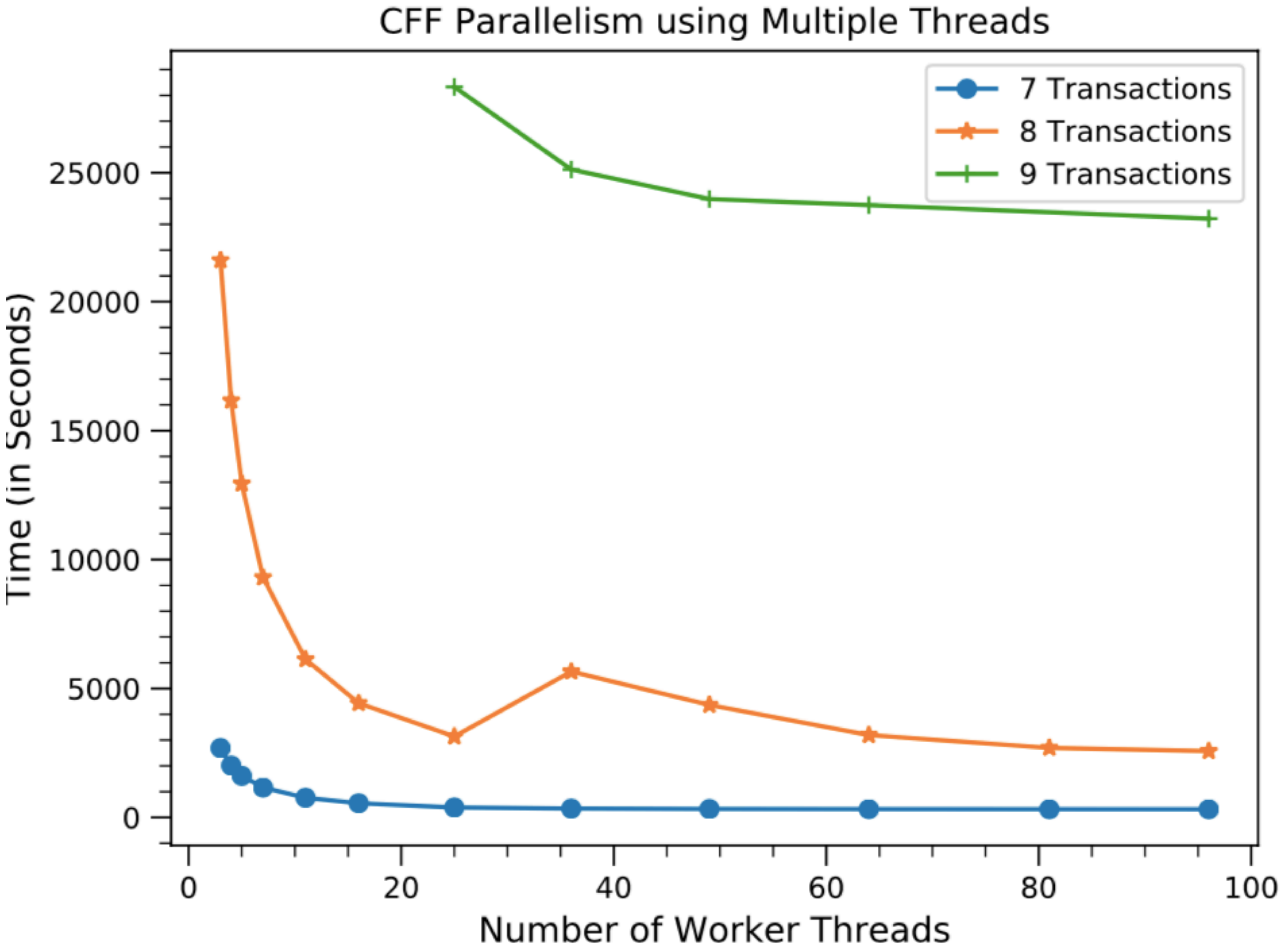
Github : https://github.com/defi-formal/cff

Contact : babel@cs.cornell.edu

Appendix

Execution and proving times

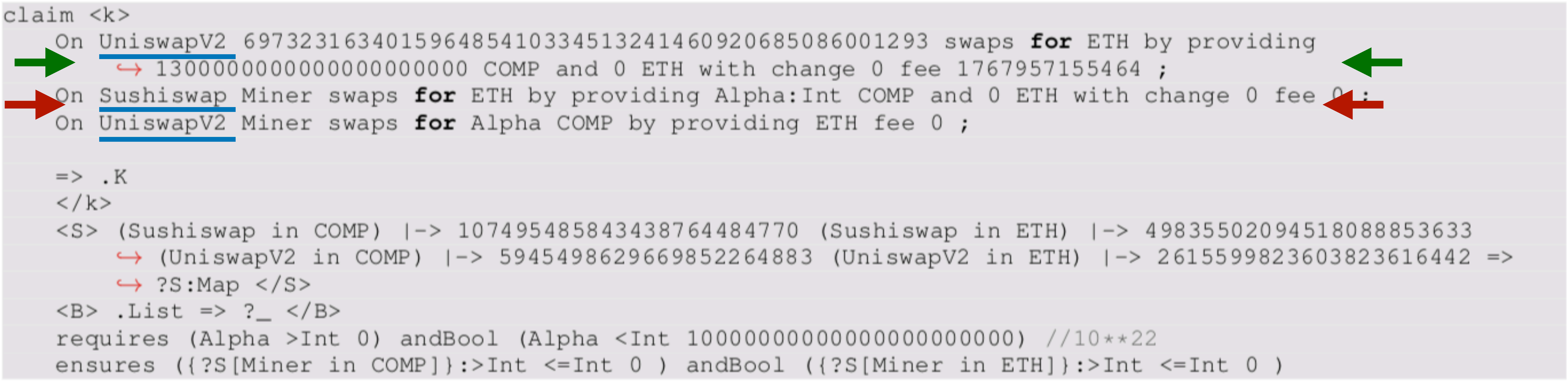




Directions for Future Work

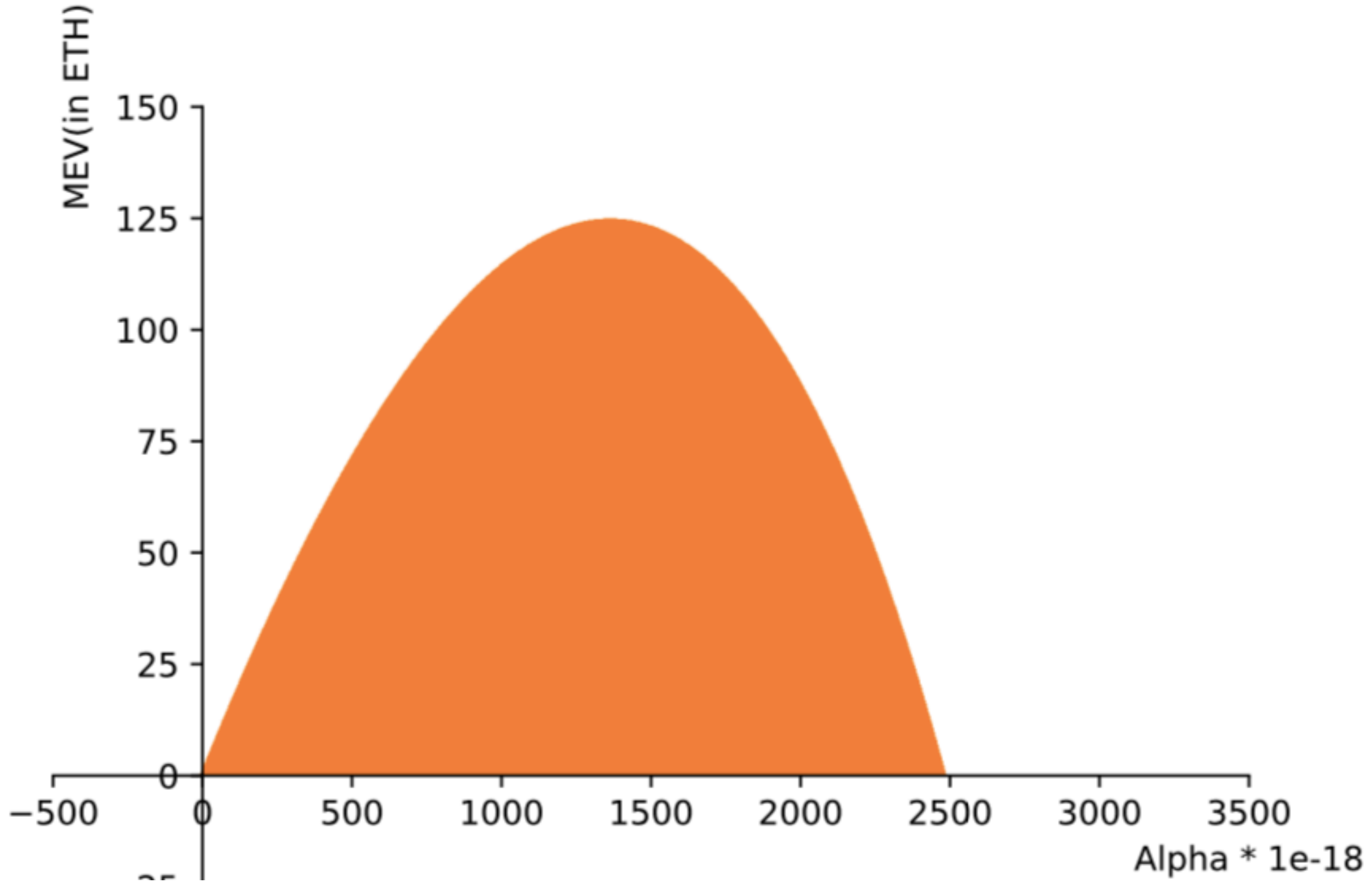
- MEV Definitions for Leaderless Protocols
- Arbitrary Symbolic Transaction Insertions
- Scaling the Backend

Under the Hood- Sushiswap + Uniswap

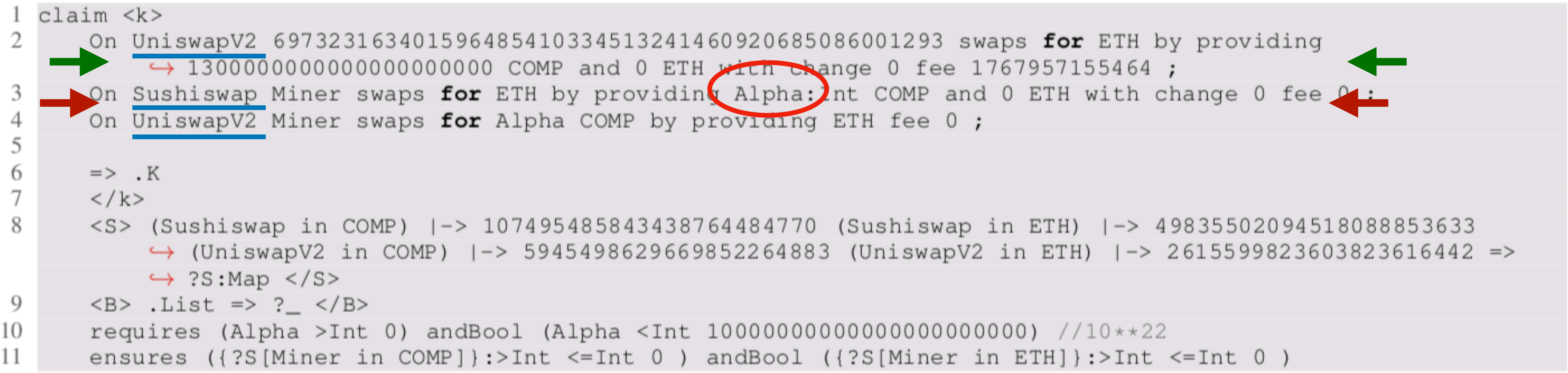


```
claim <k>
    On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
        ↪ 1300000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
    On Sushiswap Miner swaps for ETH by providing Alpha:Int COMP and 0 ETH with change 0 fee 0 ;
    On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;

    => .K
    </k>
    <S> (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 4983550209451808885363 3
        ↪ (UniswapV2 in COMP) |-> 594549862966985226488 3 (UniswapV2 in ETH) |-> 261559982360382361644 2 =>
        ↪ ?S:Map </S>
    <B> .List => ?_ </B>
    requires (Alpha >Int 0) andBool (Alpha <Int 10000000000000000000000) //10**22
    ensures ({?S[Miner in COMP]}:>Int <=Int 0 ) andBool ({?S[Miner in ETH]}:>Int <=Int 0 )
```

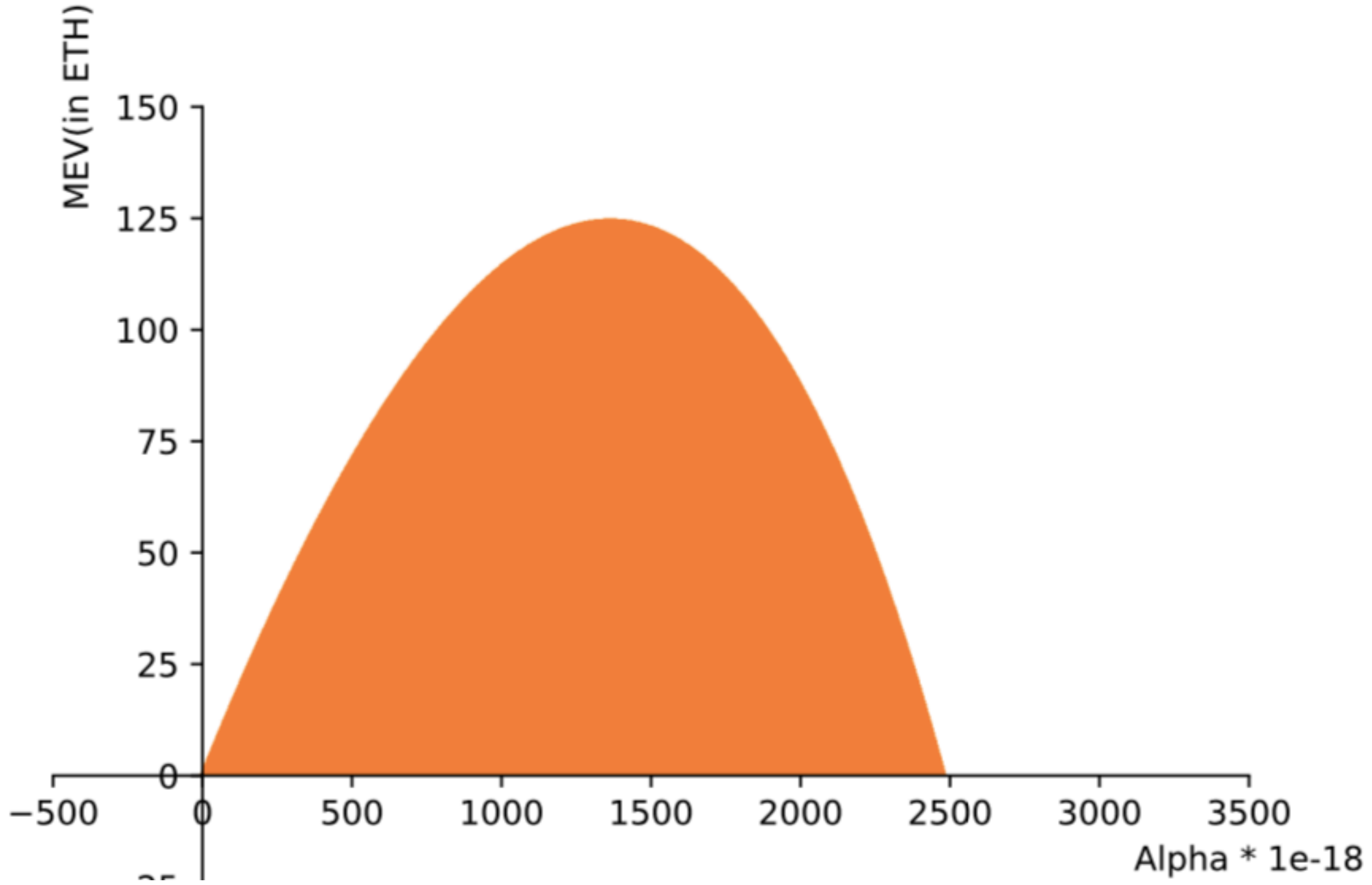


Under the Hood- Sushiswap + Uniswap

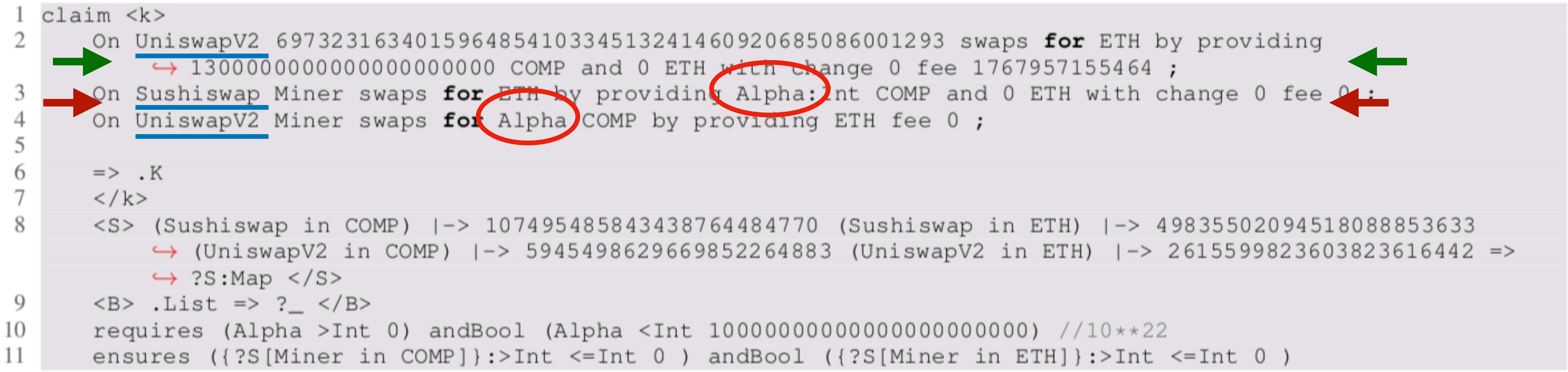


```
claim <k>
    On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
        ↪ 1300000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
    On Sushiswap Miner swaps for ETH by providing Alpha:Int COMP and 0 ETH with change 0 fee 0 ;
    On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;

    => .K
    </k>
    <S> (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 498355020945180888 53633
        ↪ (UniswapV2 in COMP) |-> 59454986296698522648 83 (UniswapV2 in ETH) |-> 261559982360382361 6442 =>
        ↪ ?S:Map </S>
    <B> .List => ?_ </B>
    requires (Alpha >Int 0) andBool (Alpha <Int 10000000000000000000000) //10**22
    ensures ({?S[Miner in COMP]}:>Int <=Int 0 ) andBool ({?S[Miner in ETH]}:>Int <=Int 0 )
```

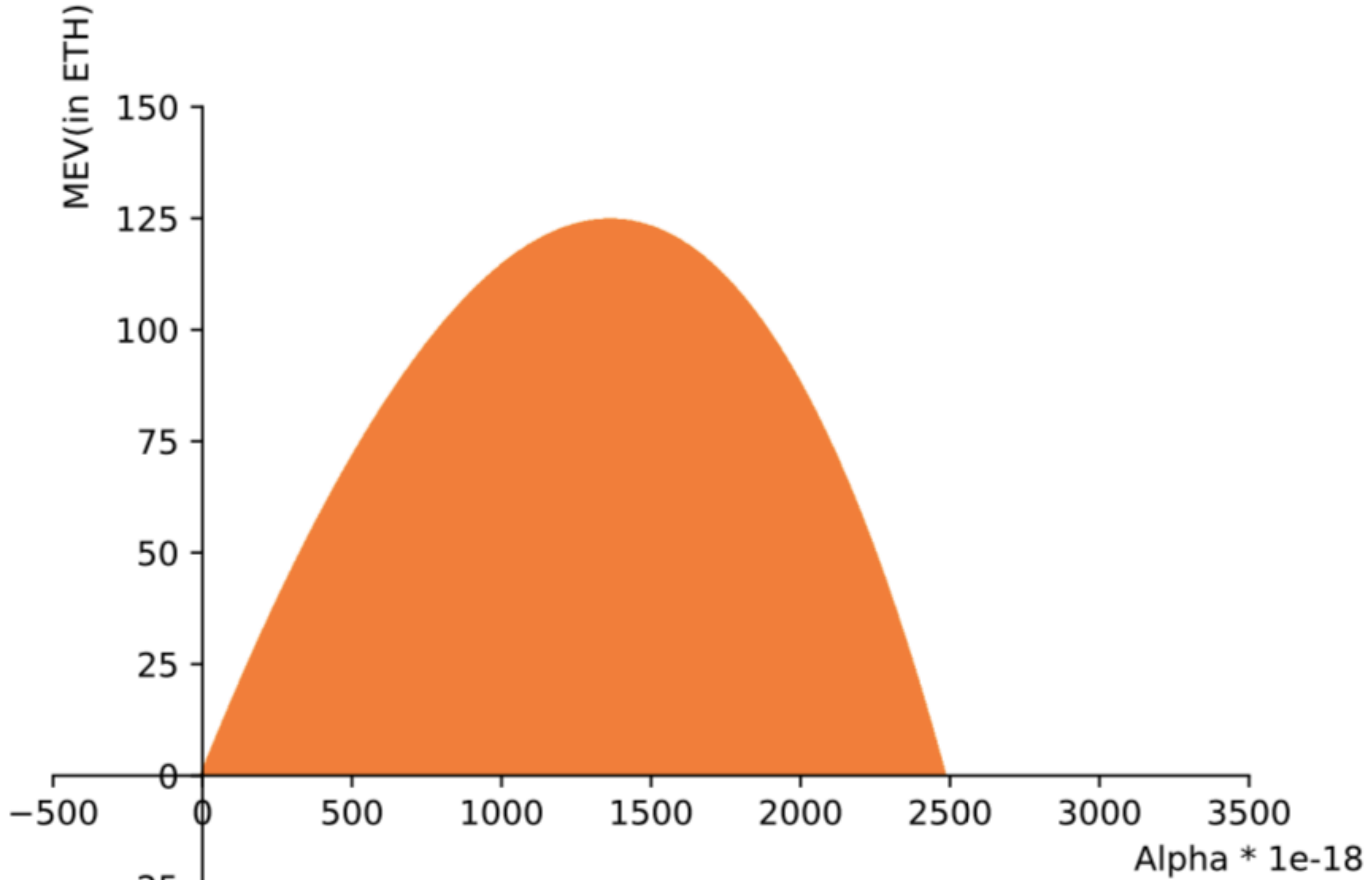


Under the Hood- Sushiswap + Uniswap



```
claim <k>
    On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
       ↪ 1300000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
    On Sushiswap Miner swaps for ETH by providing Alpha:Int COMP and 0 ETH with change 0 fee 0 ;
    On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;

    => .K
    </k>
    <S> (Sushiswap in COMP) |-> 10749548584343876448477O (Sushiswap in ETH) |-> 4983550209451808853633
       ↪ (UniswapV2 in COMP) |-> 5945498629669852264883 (UniswapV2 in ETH) |-> 2615599823603823616442 =>
       ↪ ?S:Map </S>
    <B> .List => ?_ </B>
    requires (Alpha >Int 0) andBool (Alpha <Int 10000000000000000000000) //10**22
    ensures ({?S[Miner in COMP]}:>Int <=Int 0 ) andBool ({?S[Miner in ETH]}:>Int <=Int 0 )
```


MEV(in ETH)
Alpha * 1e-18