

# Clockwork Finance: Automated Analysis of Economic Security in Smart Contracts

To Appear in IEEE S&P'23



**Kushal  
Babel**



Philip  
Daian



Mahimna  
Kelkar



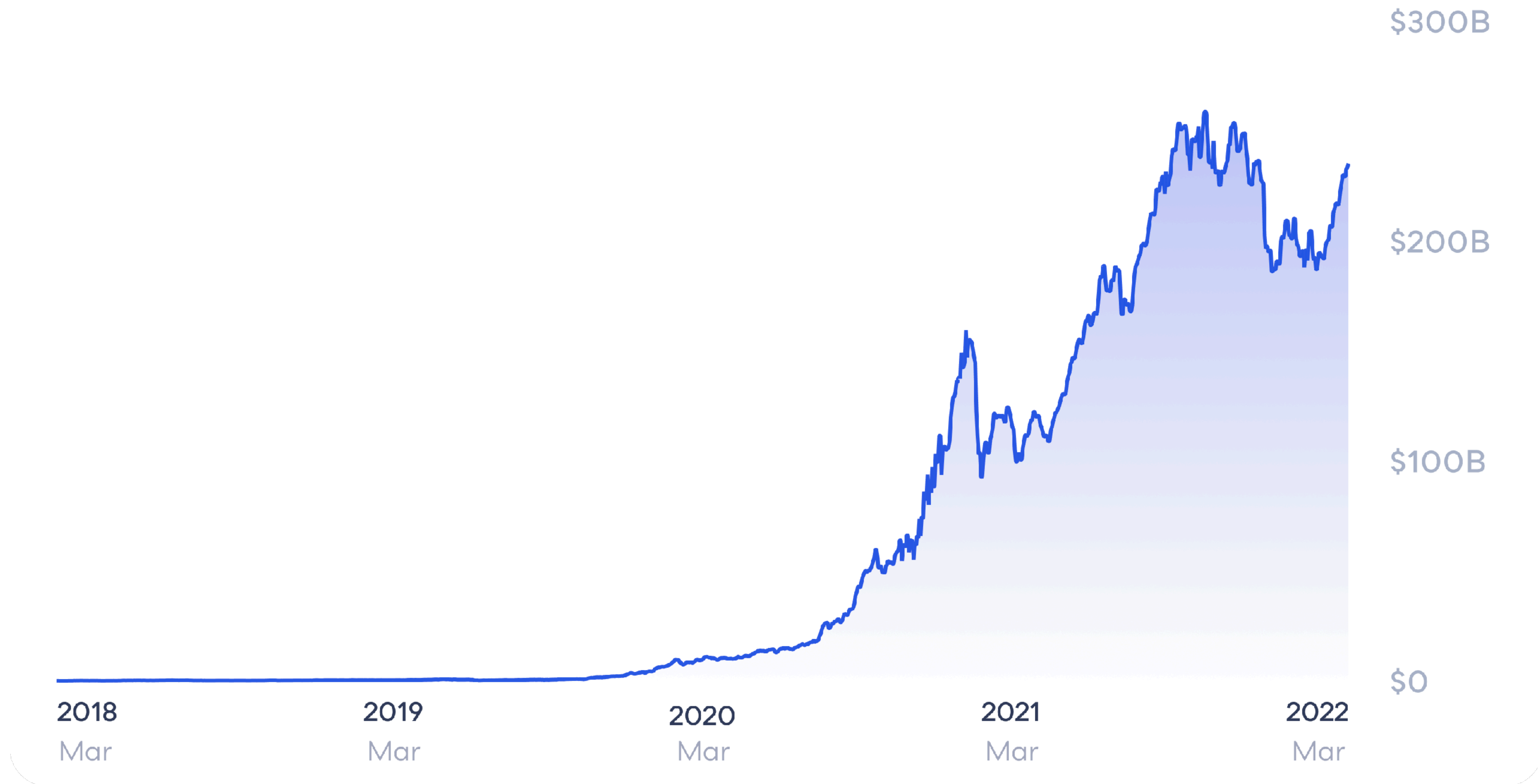
Ari  
Juels

(First three authors contributed equally)

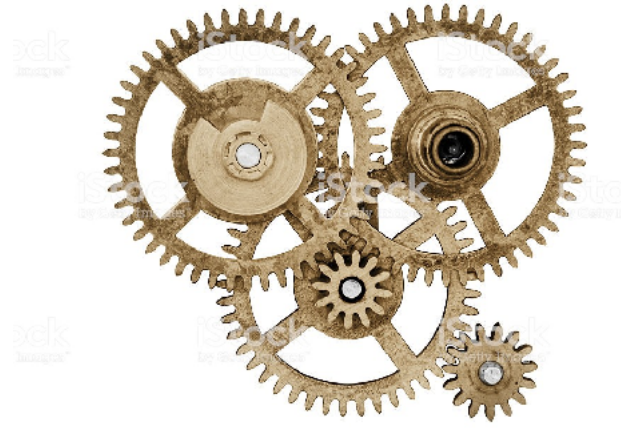
# DeFi

DeFi total TVL

revix

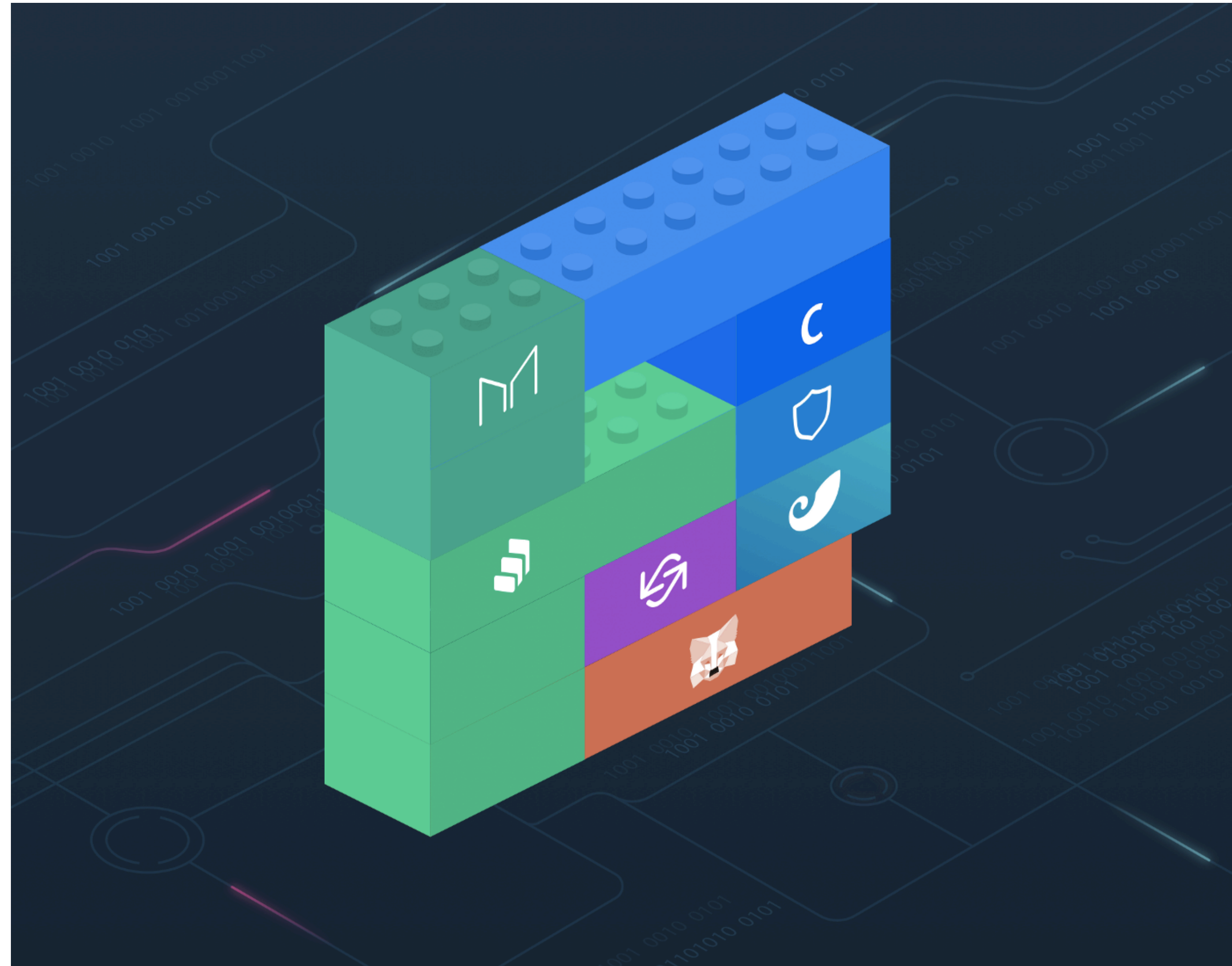


# Smart Contracts, Like Clockwork!



- 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

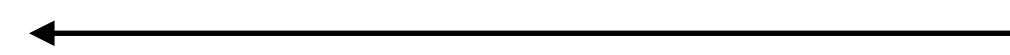


ETH Price

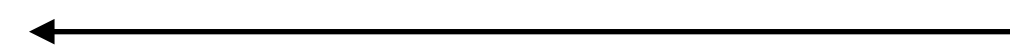


Sandwich

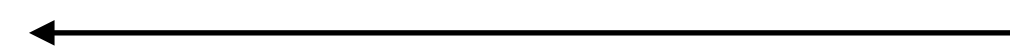
Swap X USD into ETH



Swap 1,000 USD into ETH



Swap Y ETH into USD





# Unintended Behaviour



ETH Price



Sandwich

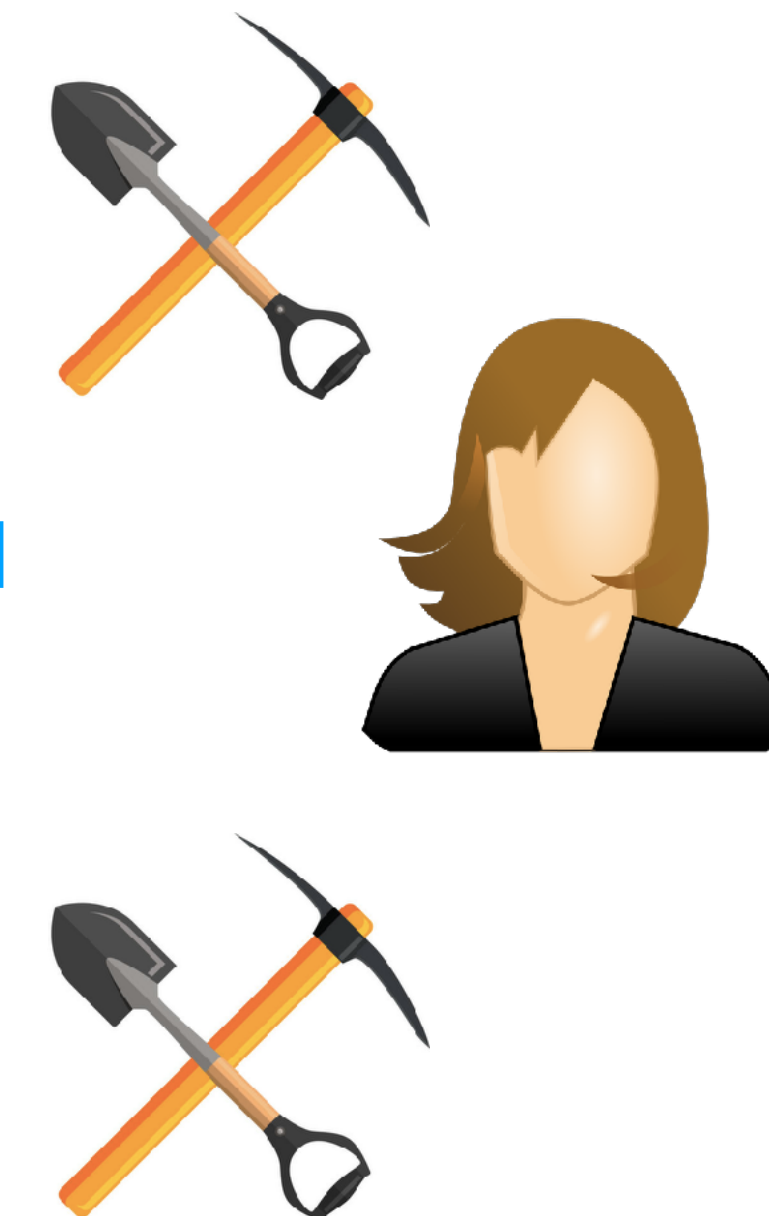
Swap X USD into ETH



Swap 1,000 USD into ETH

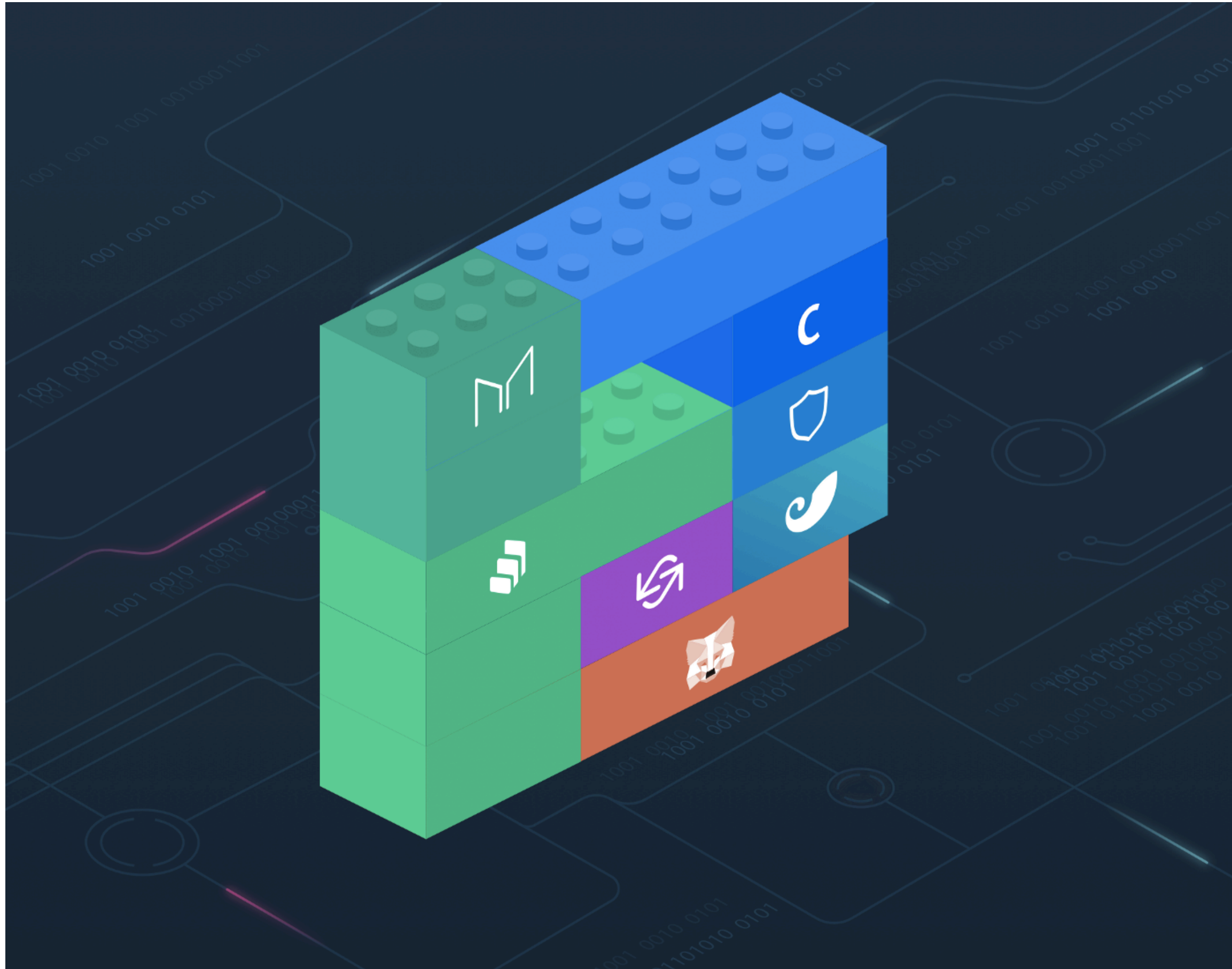


Swap Y ETH into USD



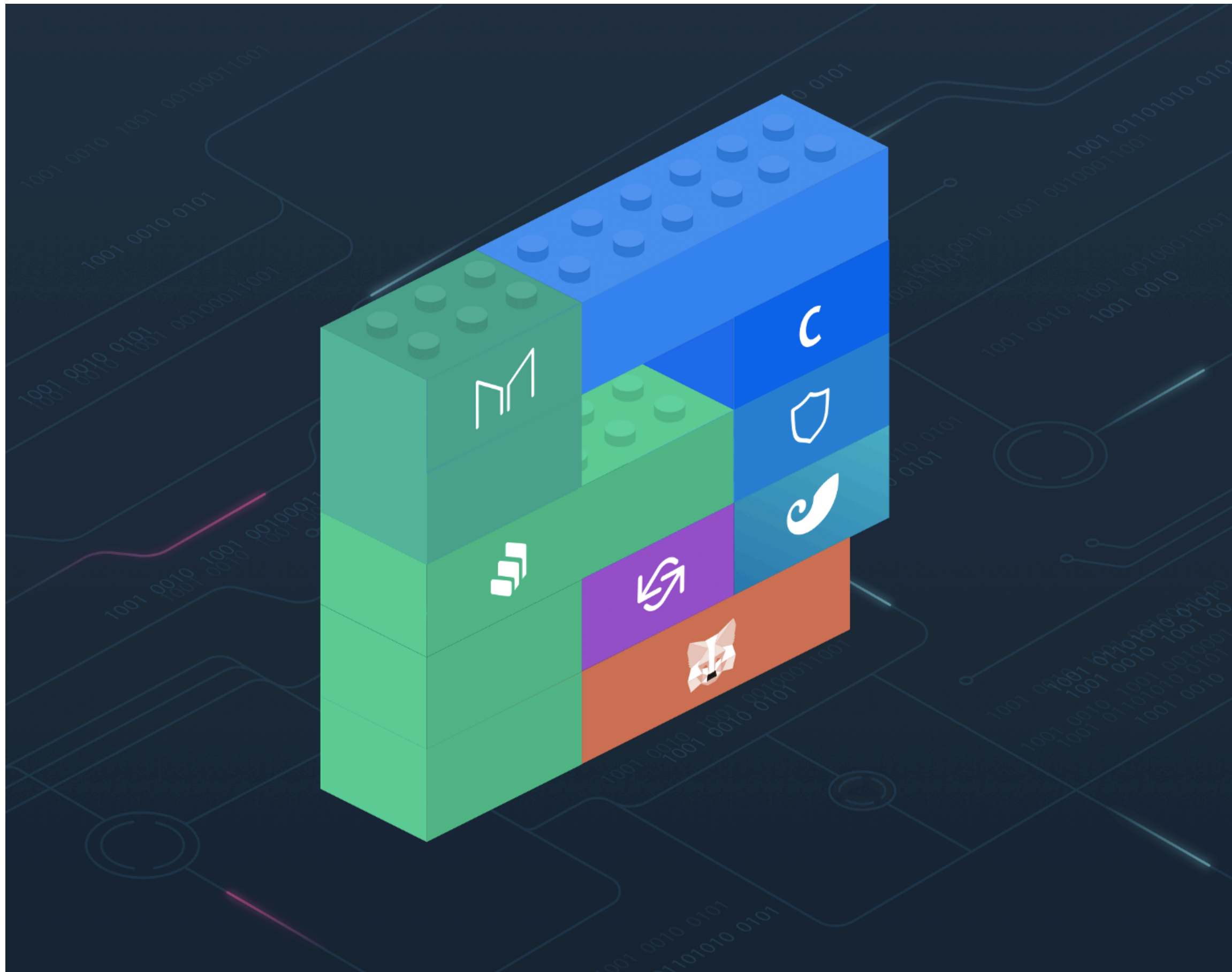
**MEV = Miner Extractable Value (or Maximal Extractable Value) - Ability to extract value by reordering, inserting or censoring transactions**

# Contract Composition



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

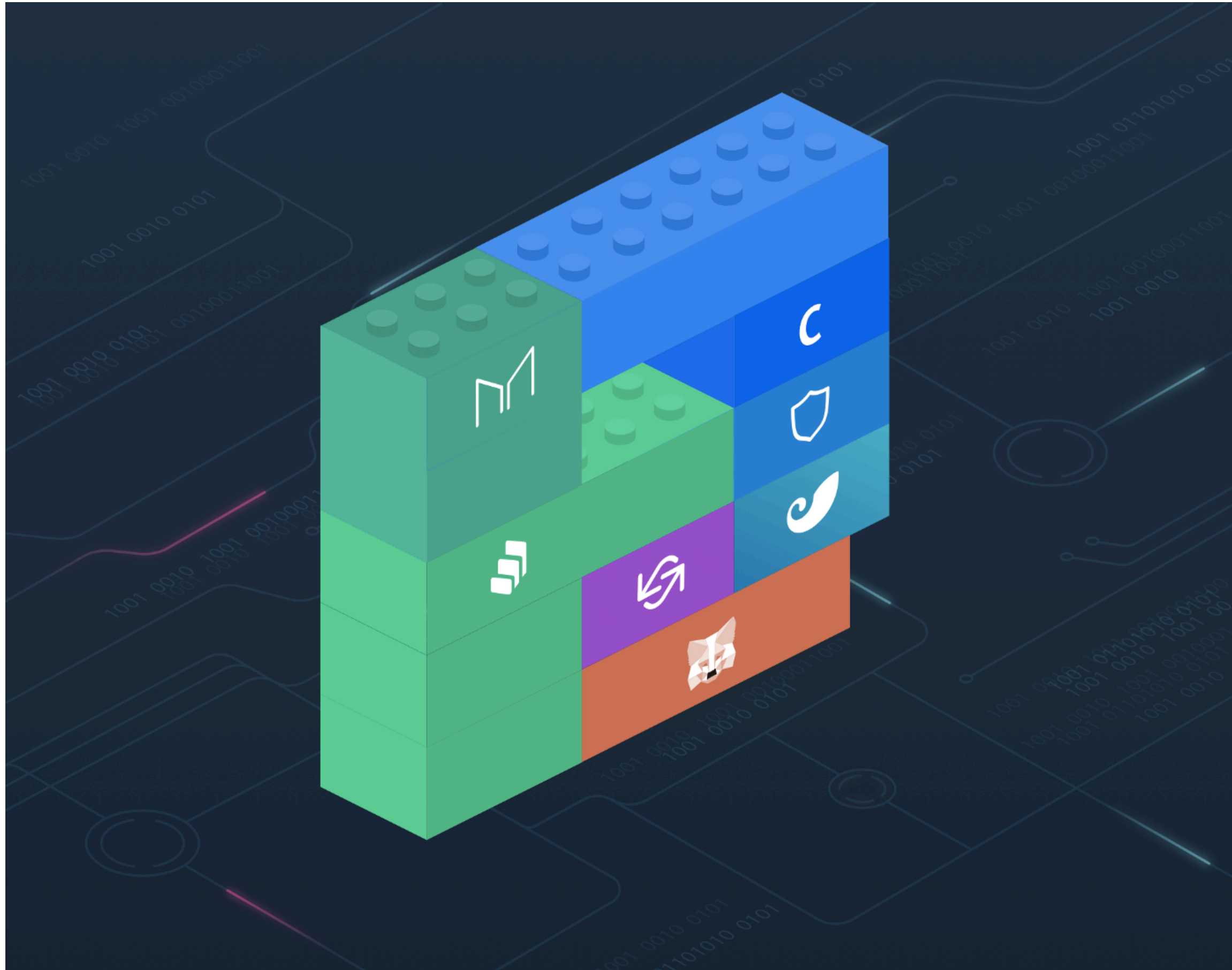
# Contract Composition



- Flashloans + DEX

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

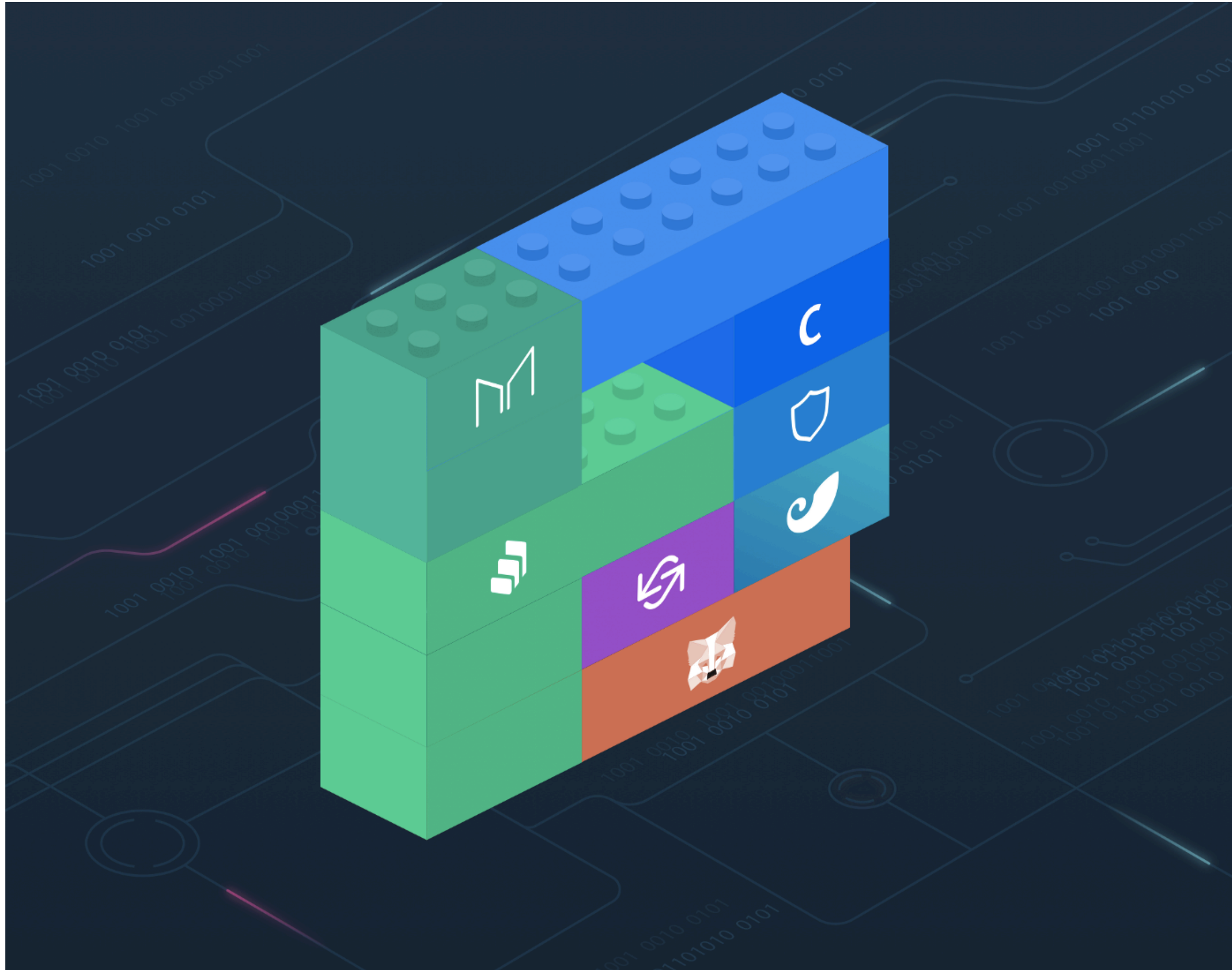
# Contract Composition



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

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

# Contract Composition



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

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

# Contract Composition



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

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

# Unintended Behaviour

09 May 2022 00:53 GMT-7 · 2 min read



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

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



JESSE COGLAN

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





# 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

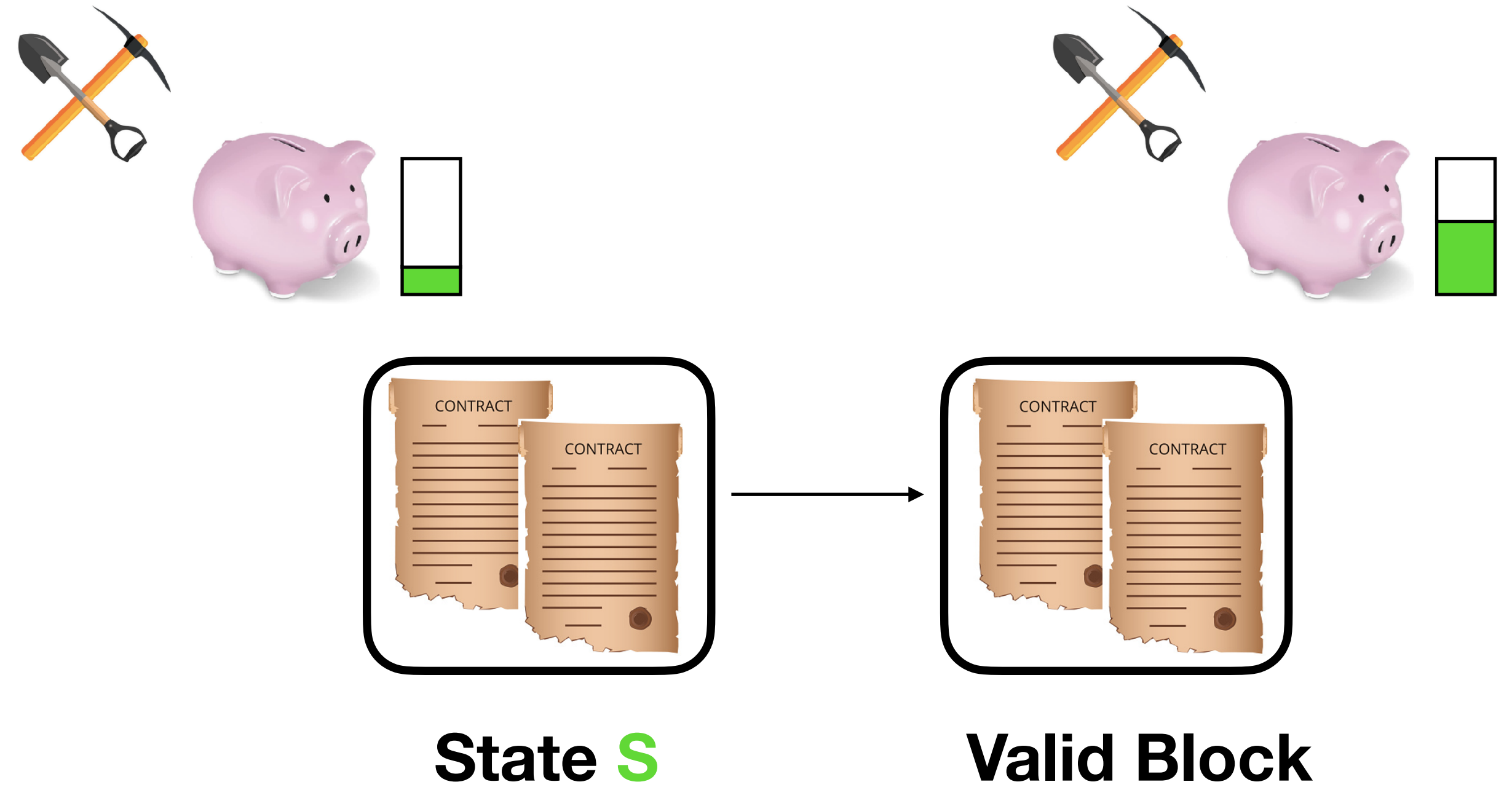
# Outline

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

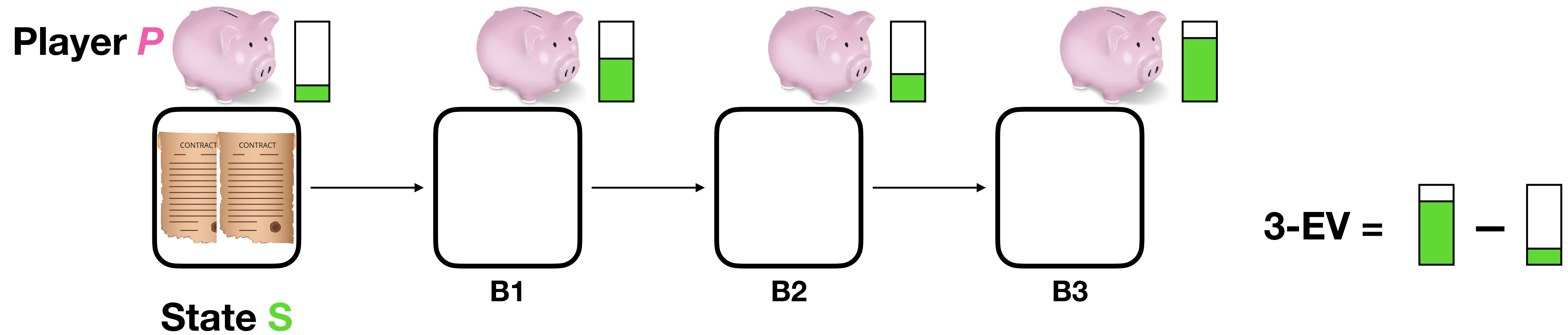
# Outline

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

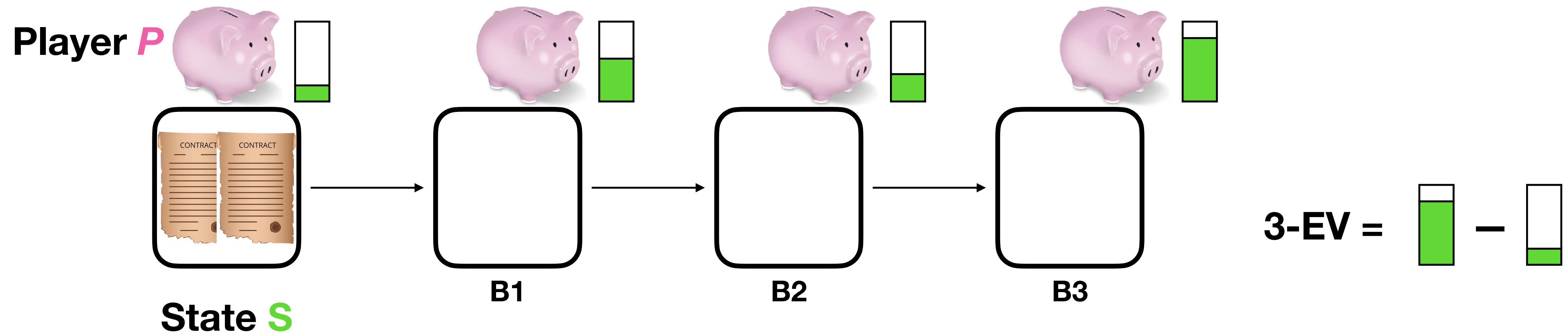
# Miner Extractable Value (MEV)



# Extractable Value (EV)



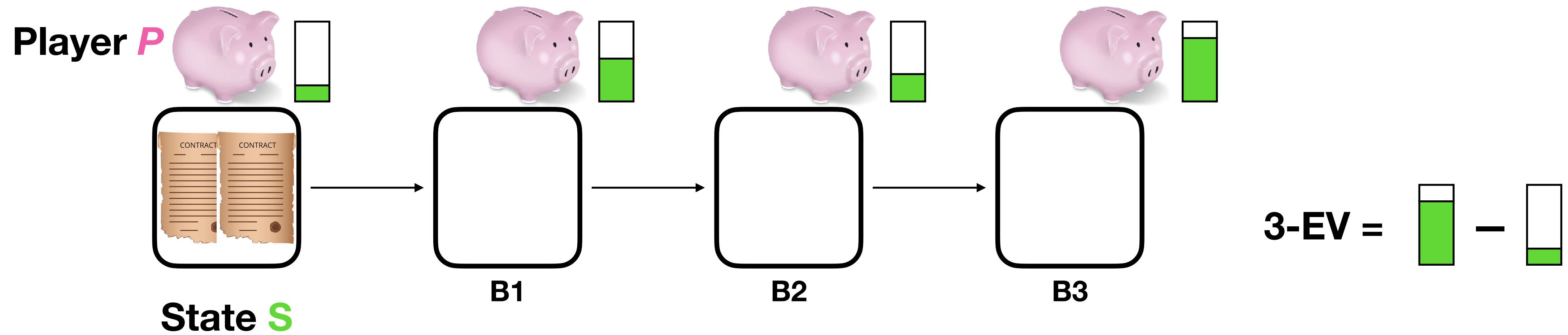
# Extractable Value (EV)



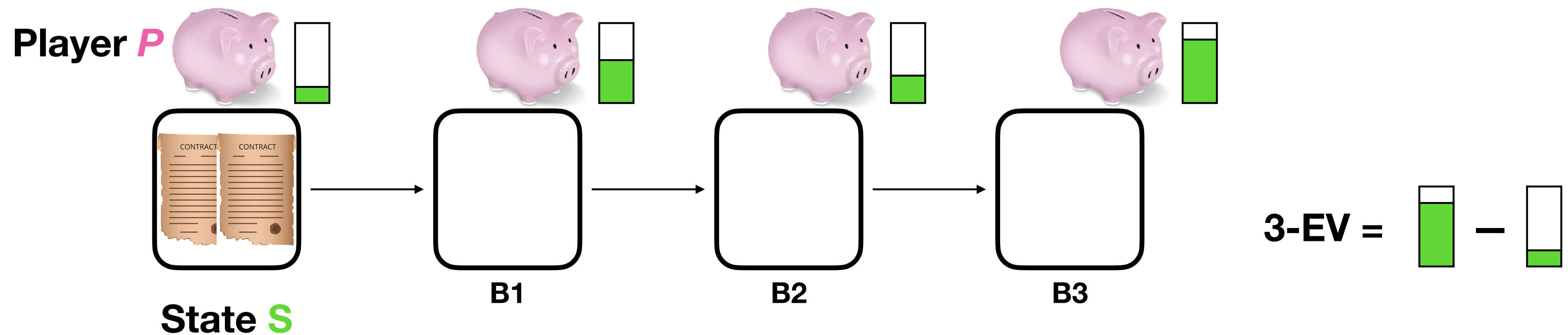
$$EV(P, \mathcal{B}, s) = \max_{(B_1, \dots, B_k) \in \mathcal{B}} \left\{ \sum_{a \in A_P} \left. \begin{array}{l} \text{balance}_k(a)[0] \\ -\text{balance}_0(a)[0] \end{array} \right\}$$



# Extractable Value (EV)



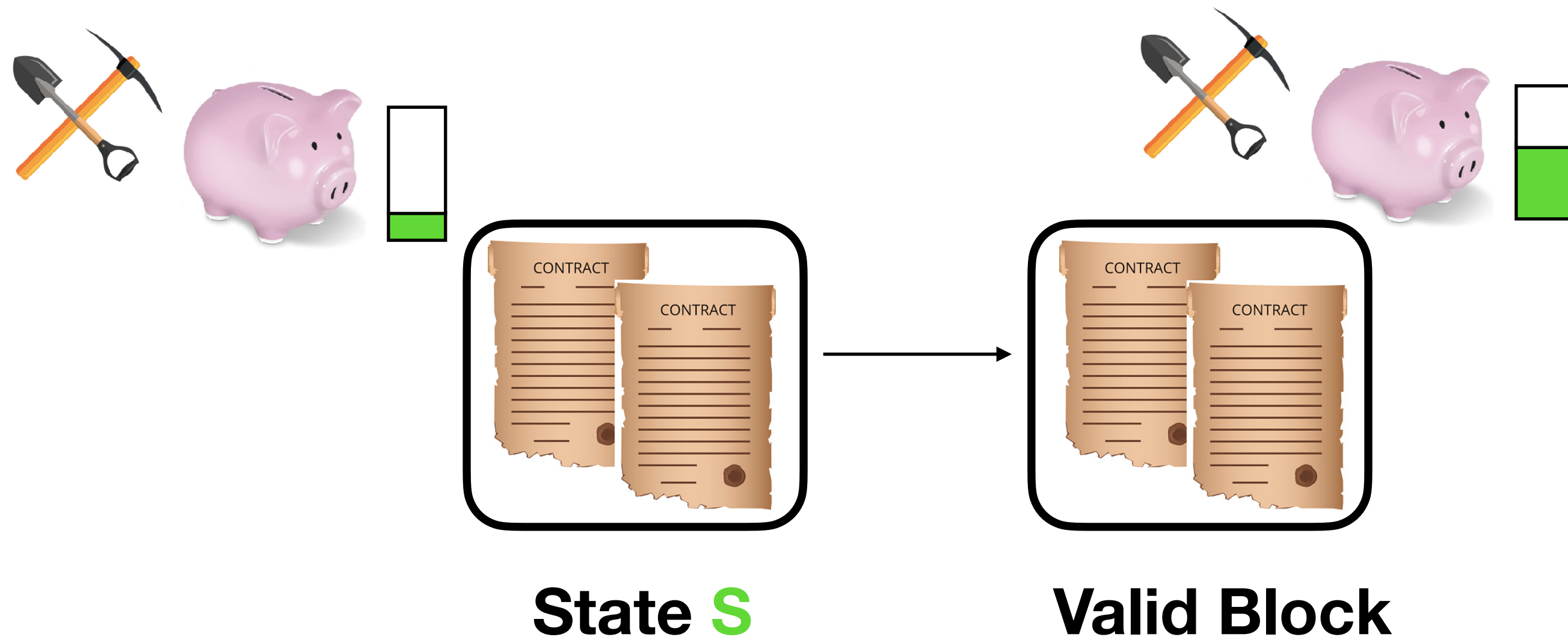
# Extractable Value (EV)



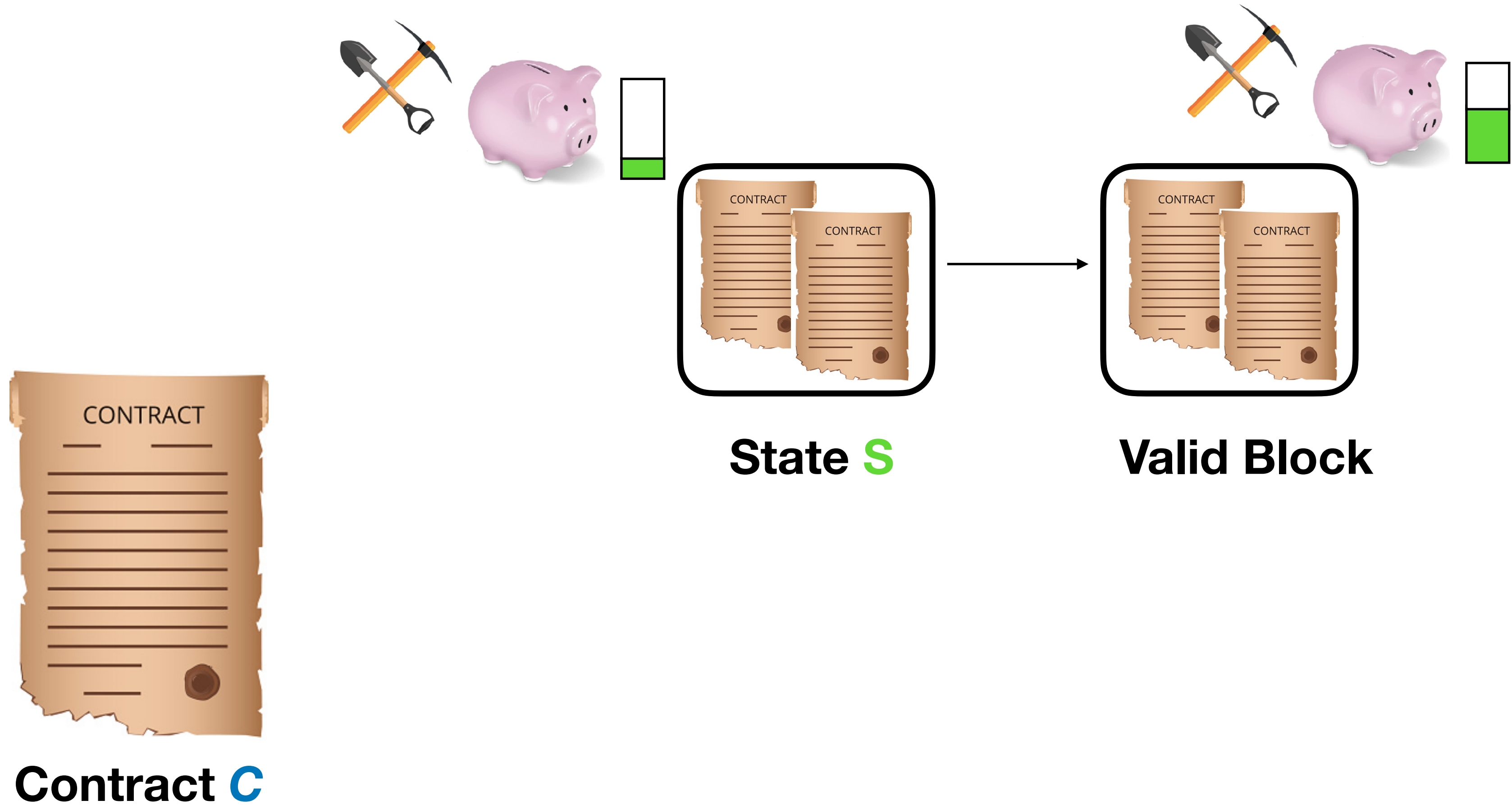
**Use MEV as the measure of economic security**

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

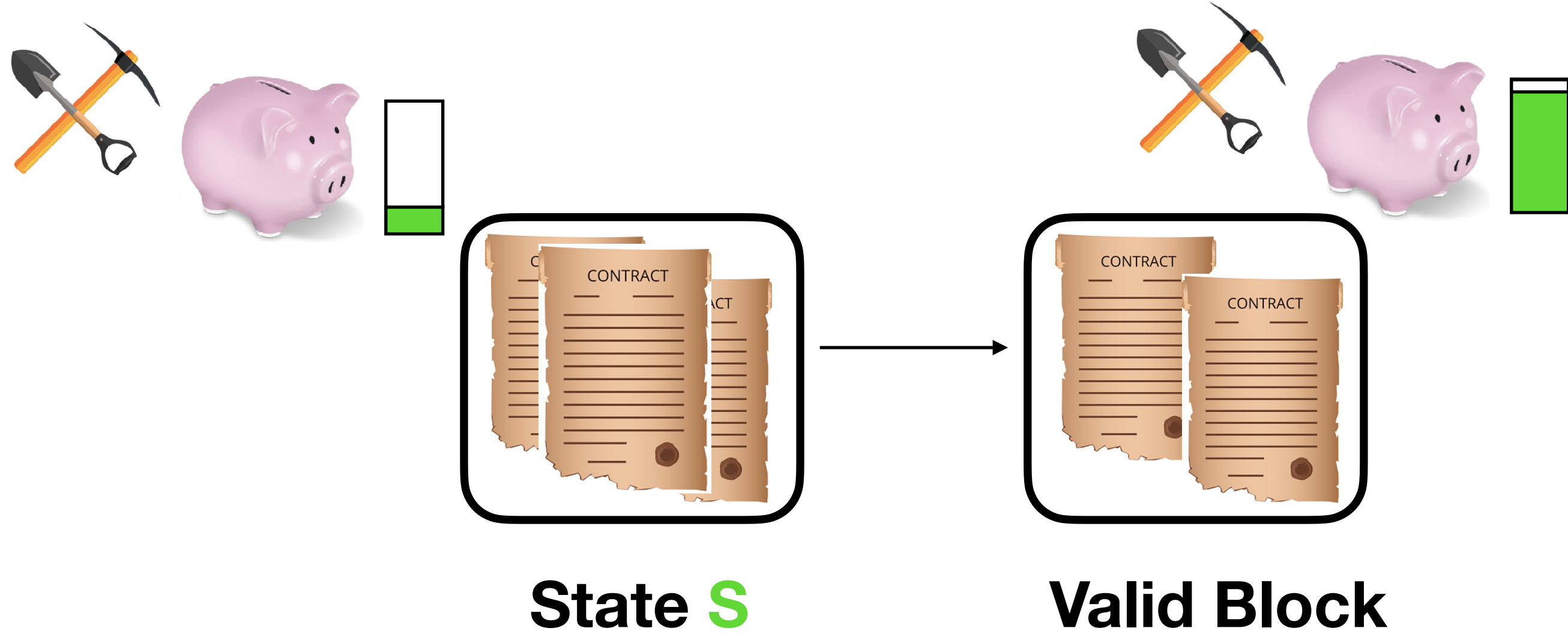
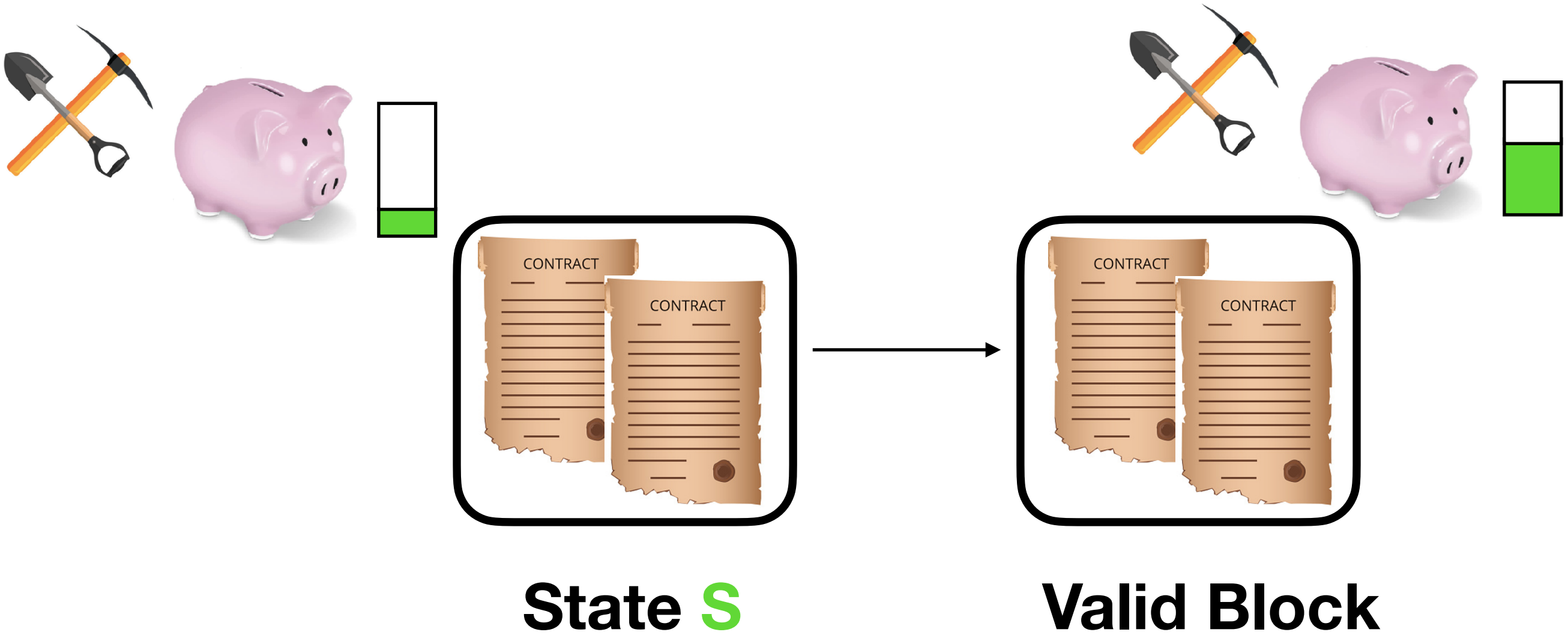
# Secure Composition



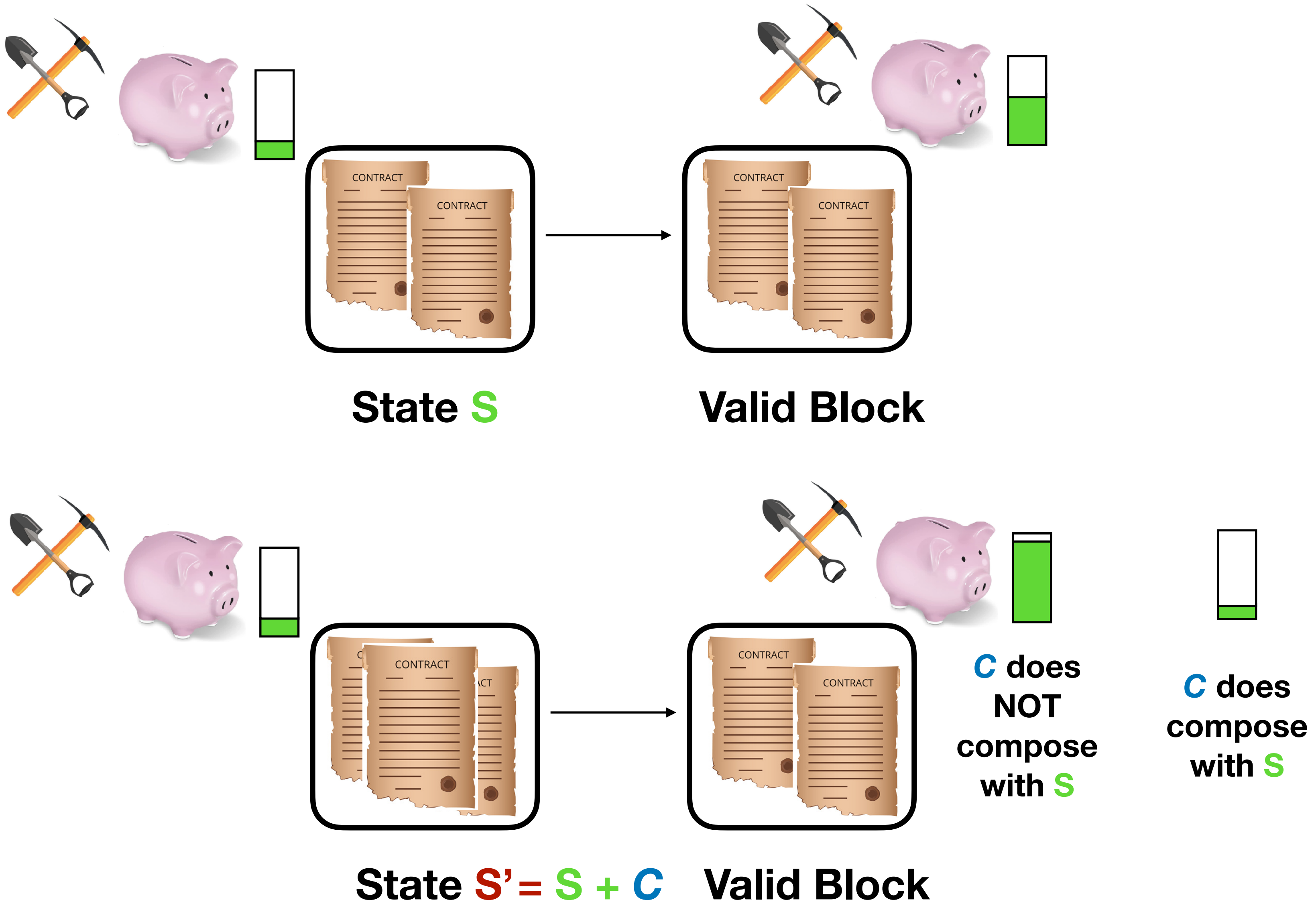
# Secure Composition



# Secure Composition



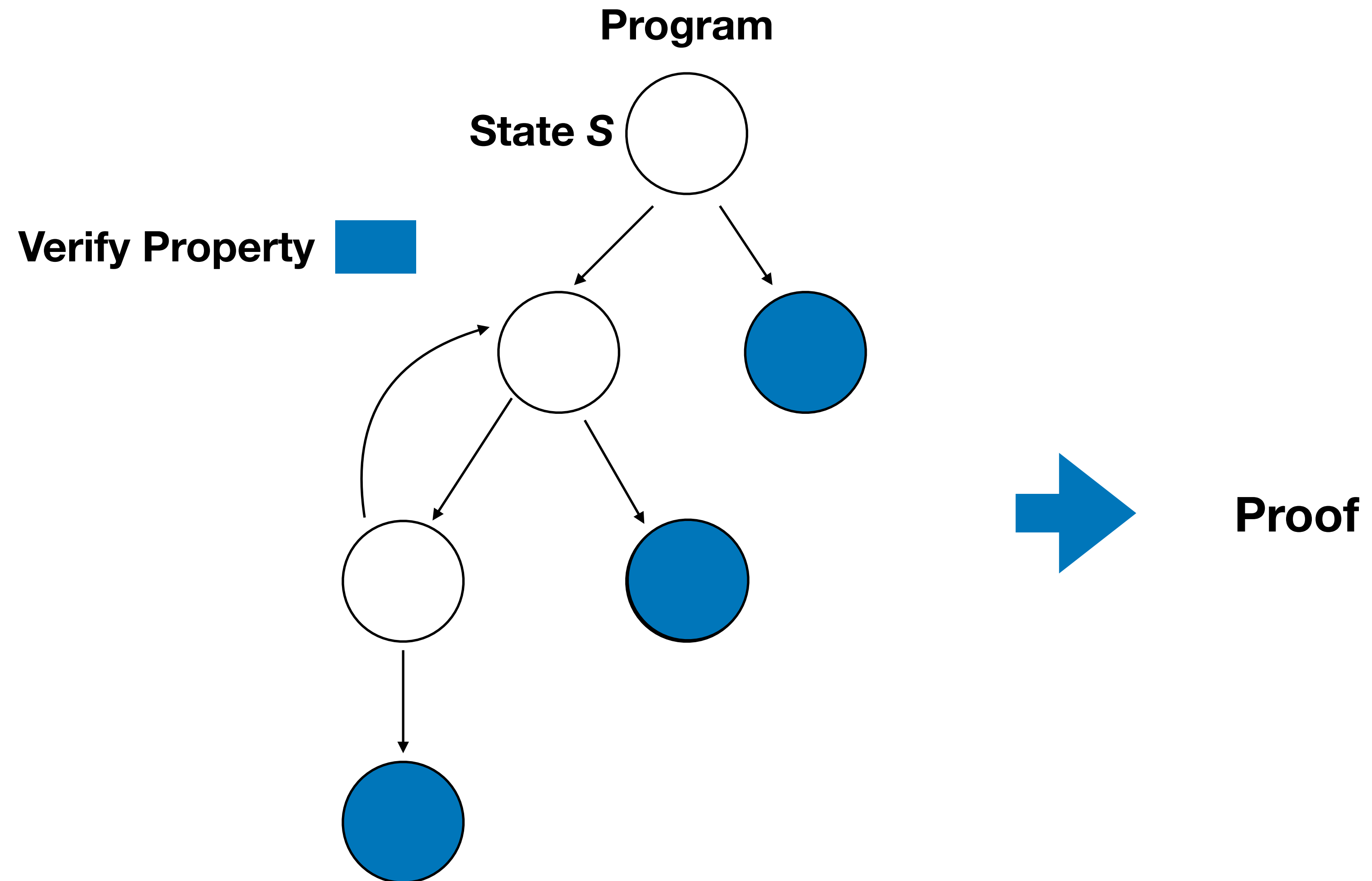
# Secure Composition



# Outline

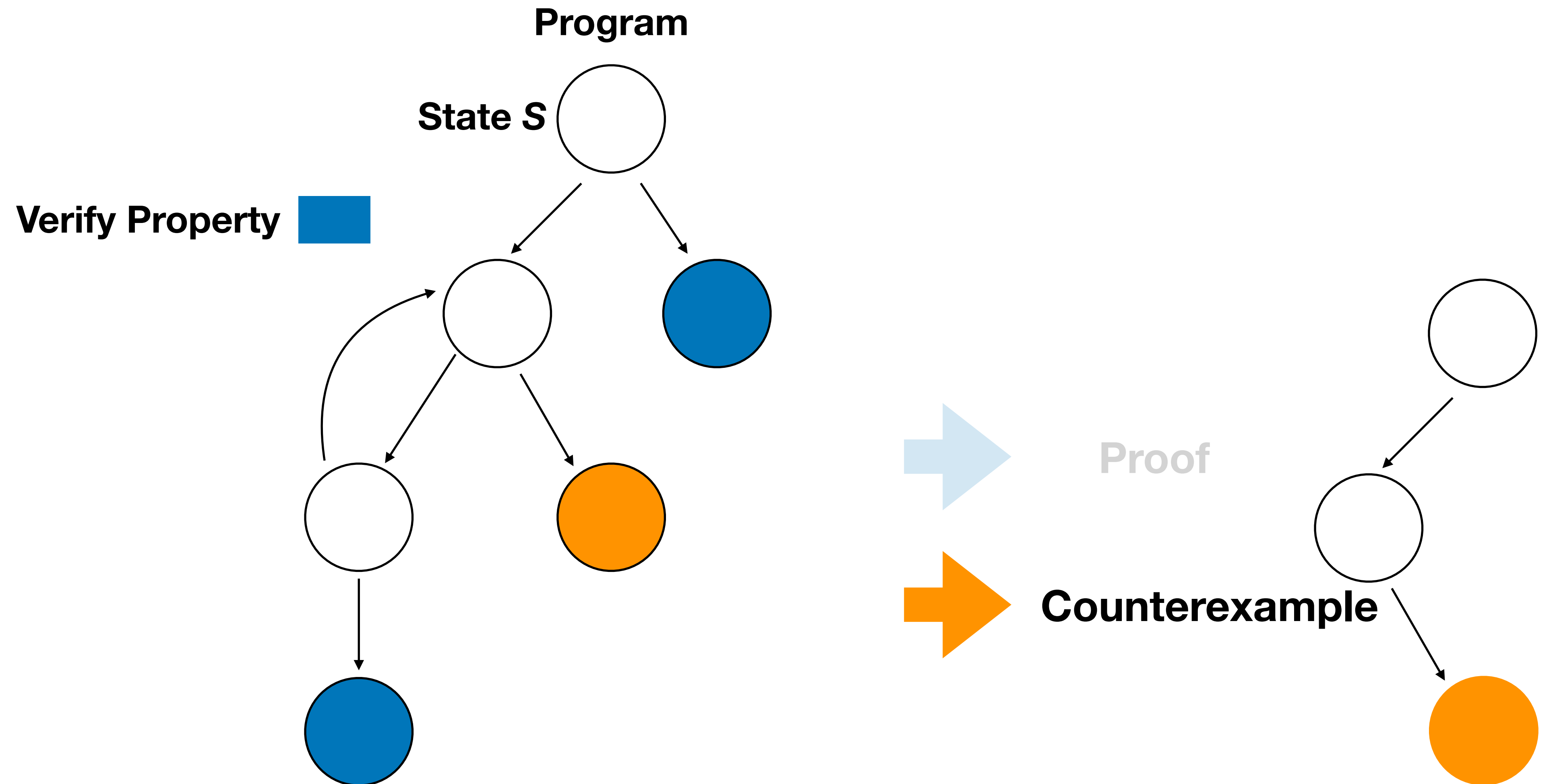
- Closer look at an example
- Definitional tools
  - Defining EV
  - Defining Secure Composition
- Practical Instantiation into Clockwork Finance Framework (CFF)
  - Design
  - Use for proofs
  - Use for finding attacks

# Formal Verification





# Formal Verification



# Clockwork Finance Framework (CFF)

# Clockwork Finance Framework (CFF)



(Symbolic) Transactions = tx1, tx2, tx3

Verify Property:  $MEV < \delta$

# Clockwork Finance Framework (CFF)



(Symbolic) Transactions = tx1, tx2, tx3

Swap **X** Eth for **Y** USD  
**X**  $\geq$  0, **Y**  $\geq$  0

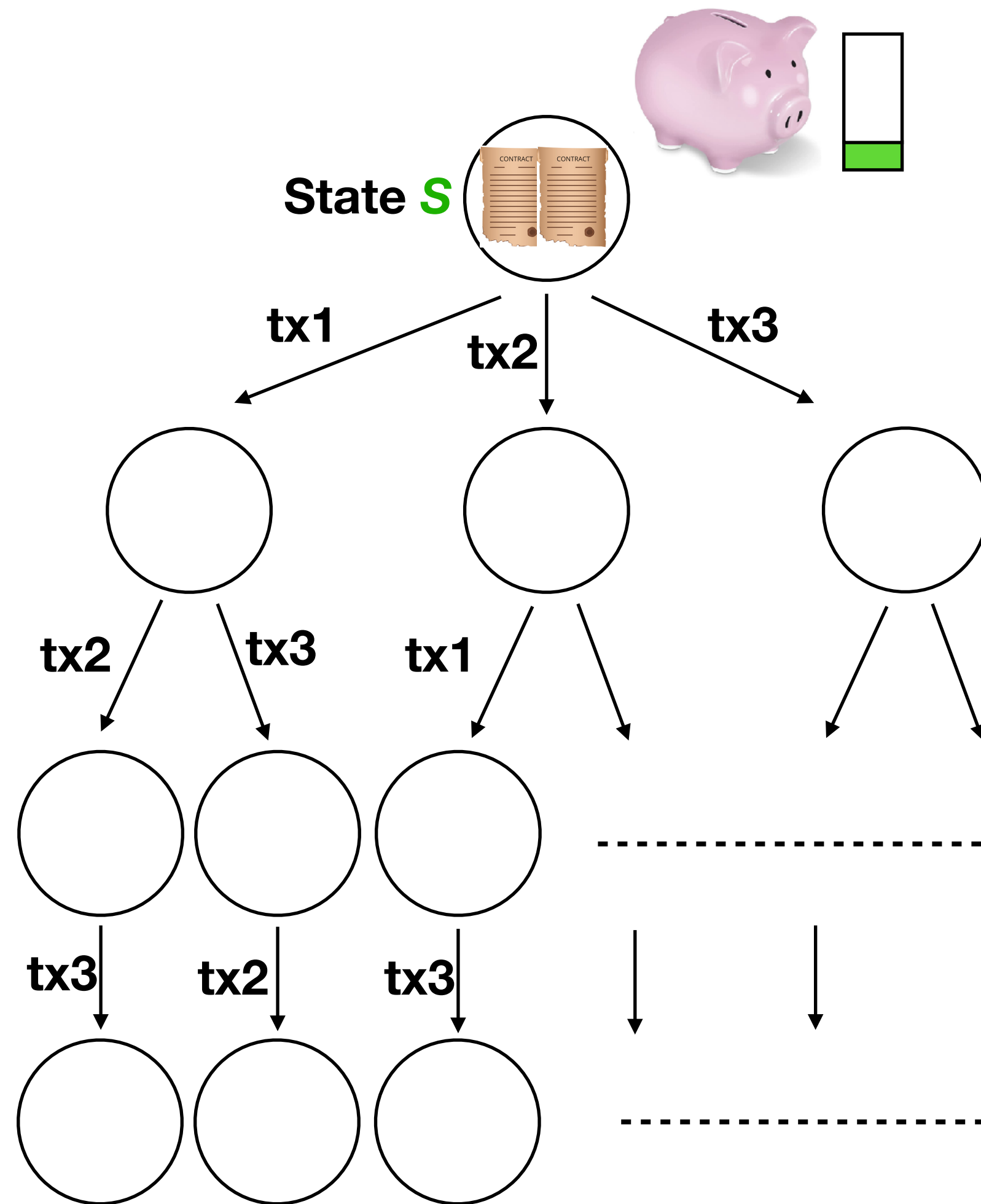
Verify Property:  $MEV < \delta$

# Clockwork Finance Framework (CFF)

(Symbolic) Transactions = tx1, tx2, tx3

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

Verify Property:  $MEV < \delta$

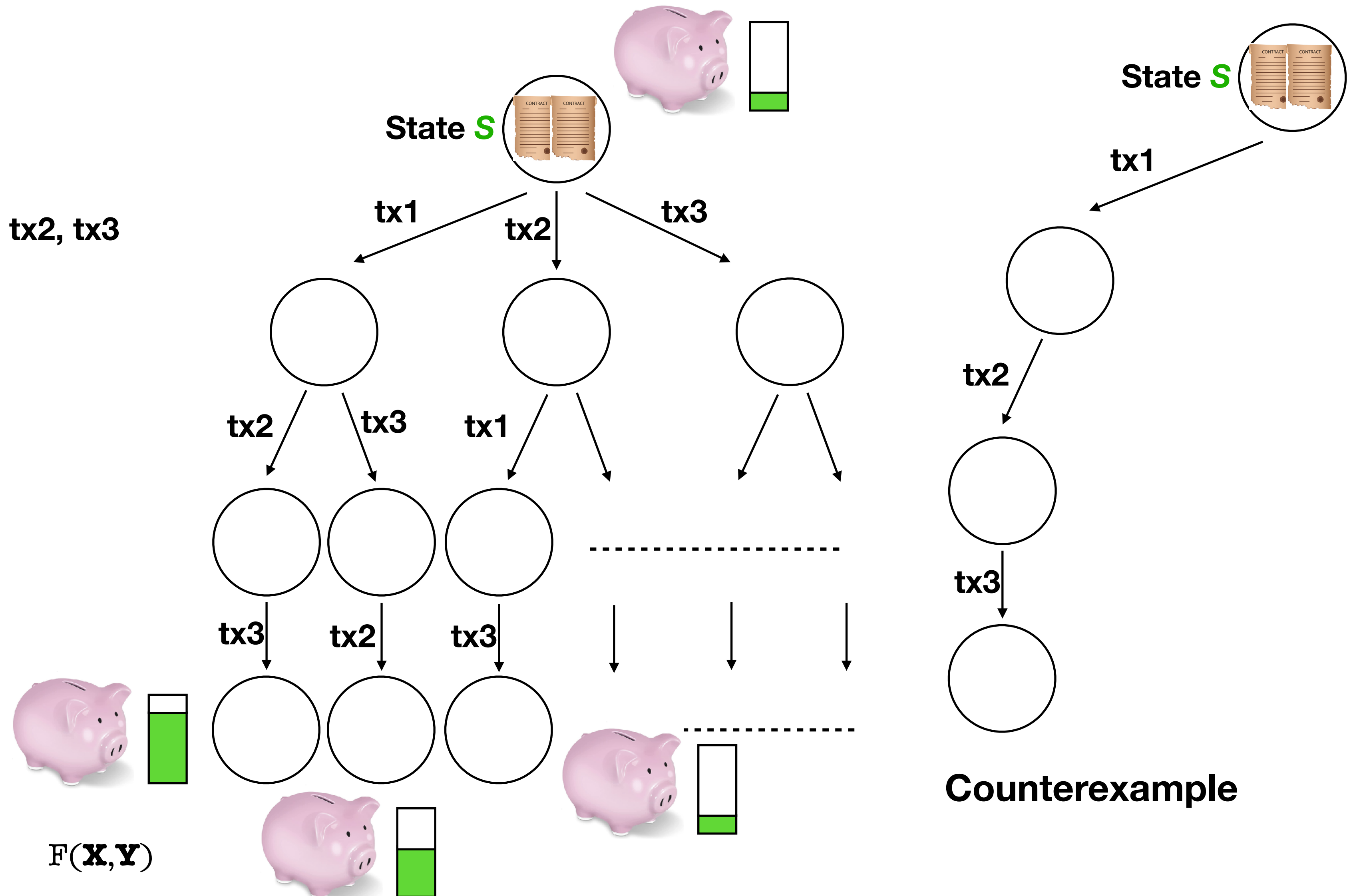


# Clockwork Finance Framework (CFF)

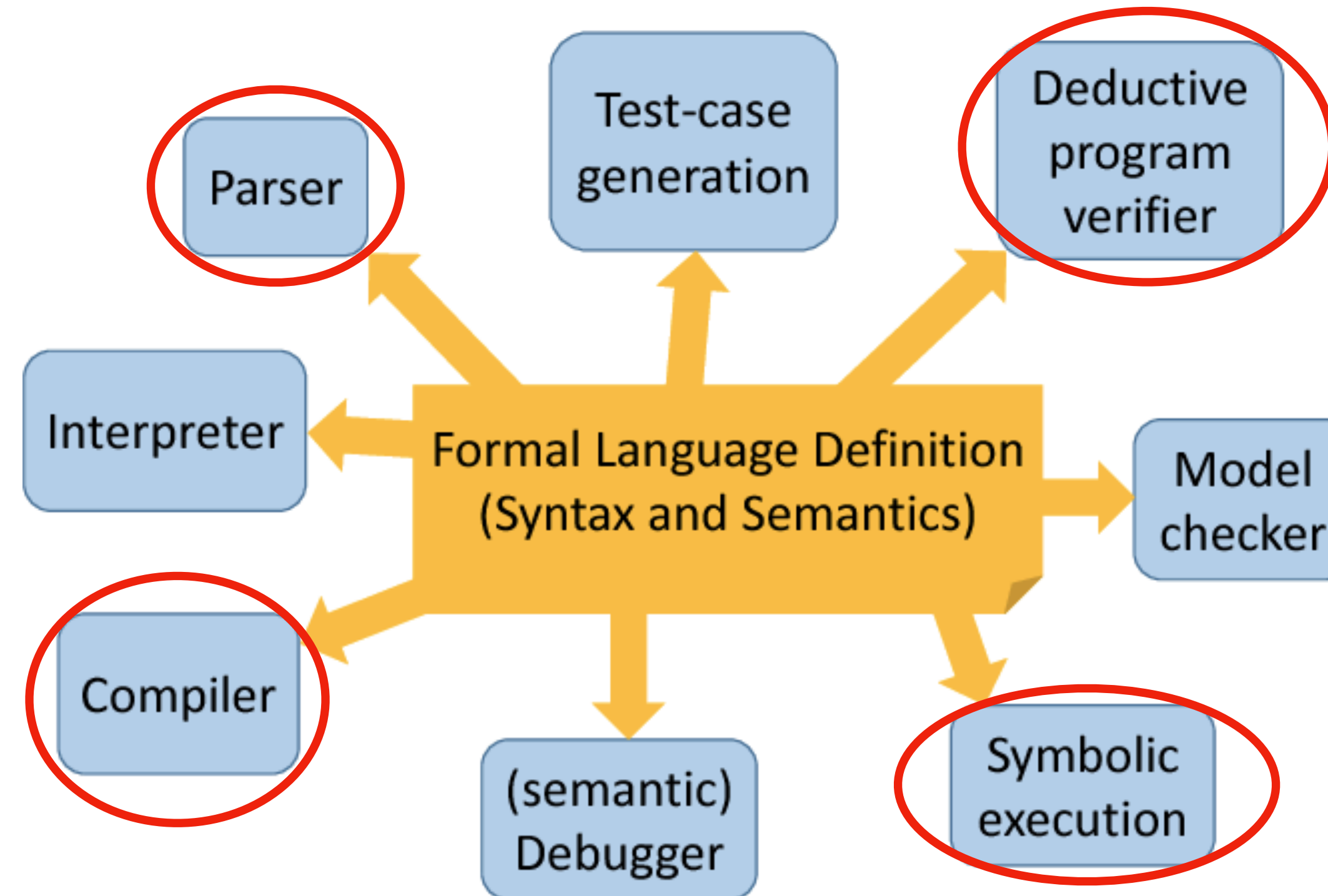
(Symbolic) Transactions = tx1, tx2, tx3

Swap  $X$  Eth for  $Y$  USD  
 $X \geq 0, Y \geq 0$

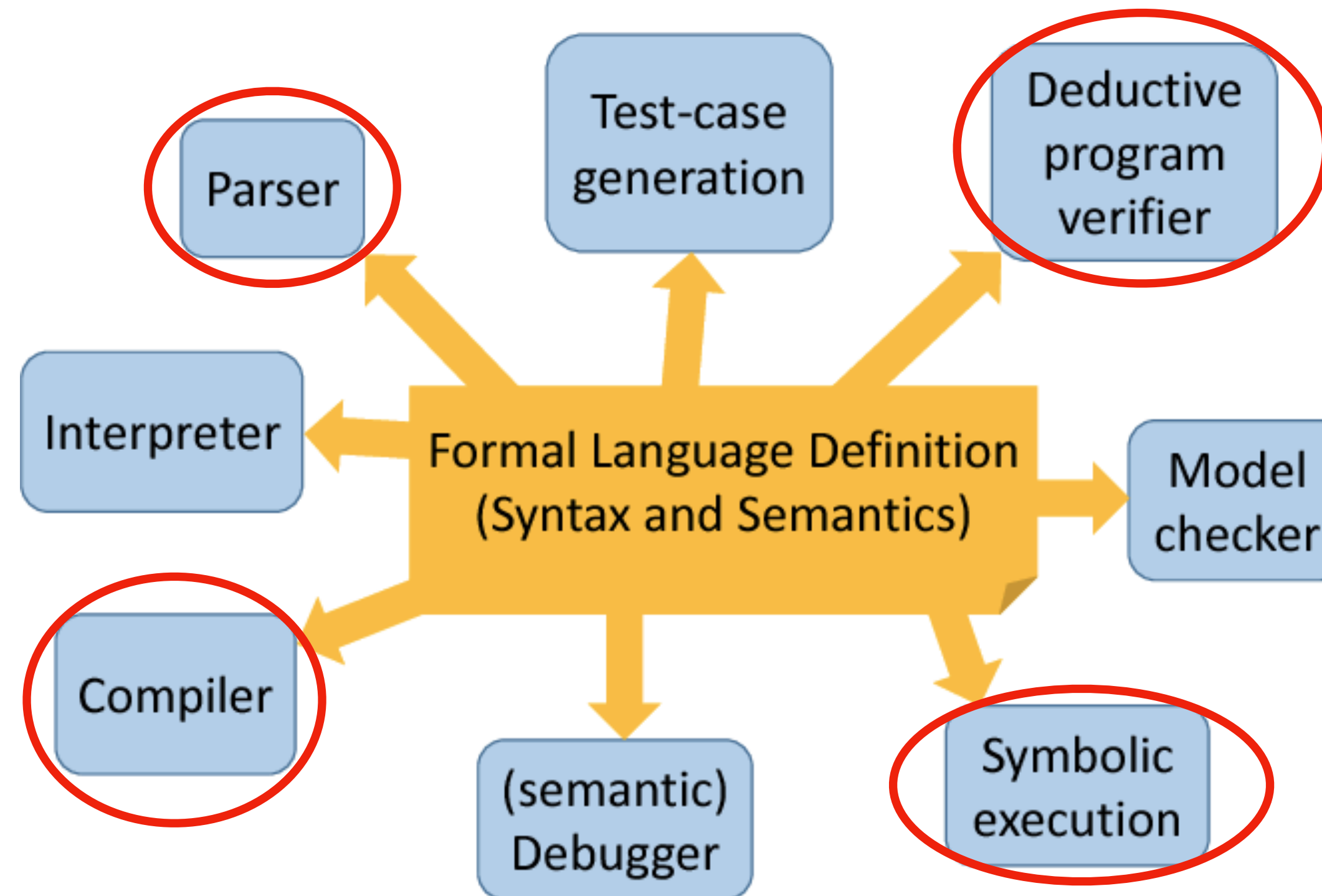
Verify Property:  $MEV < \delta$



# K Framework



# K Framework

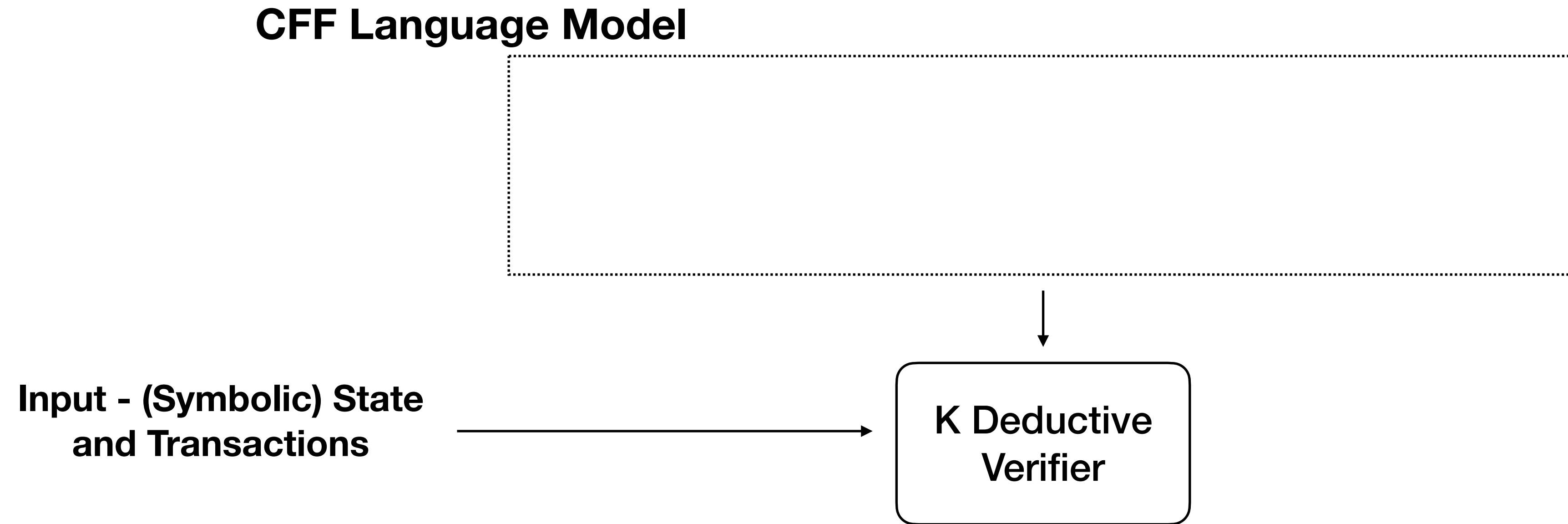


- Human Readable Formal Specification
- KEVM - Formal Ethereum Semantics in K

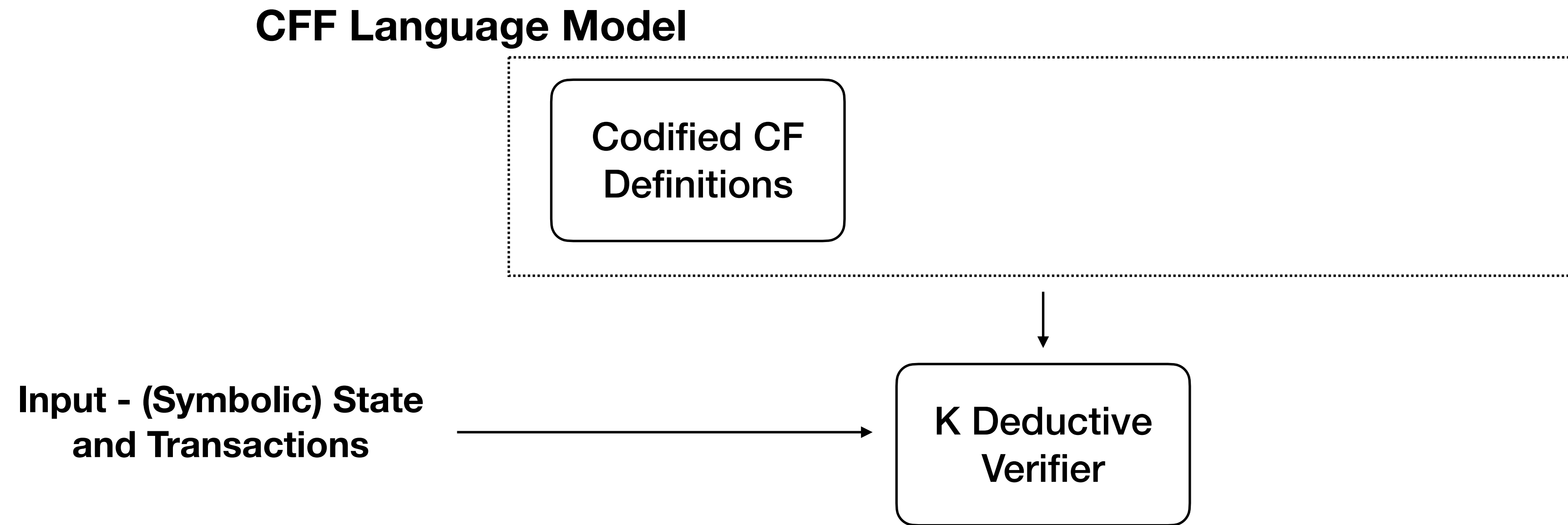


# CFF Design

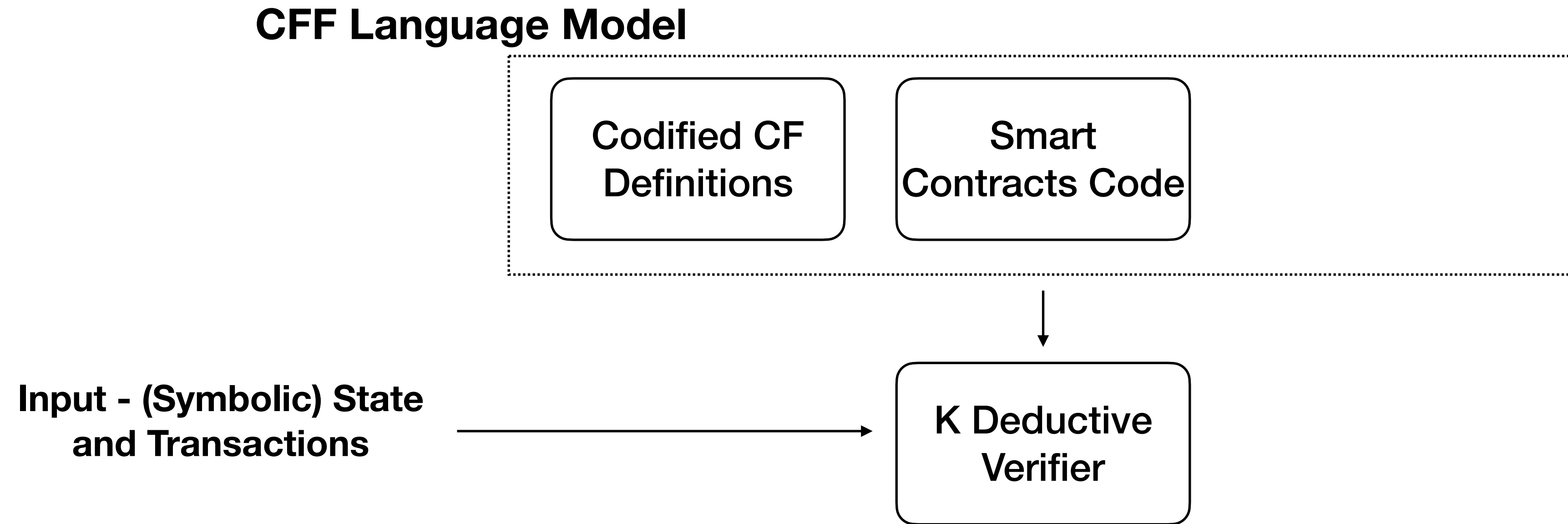
# CFF Design



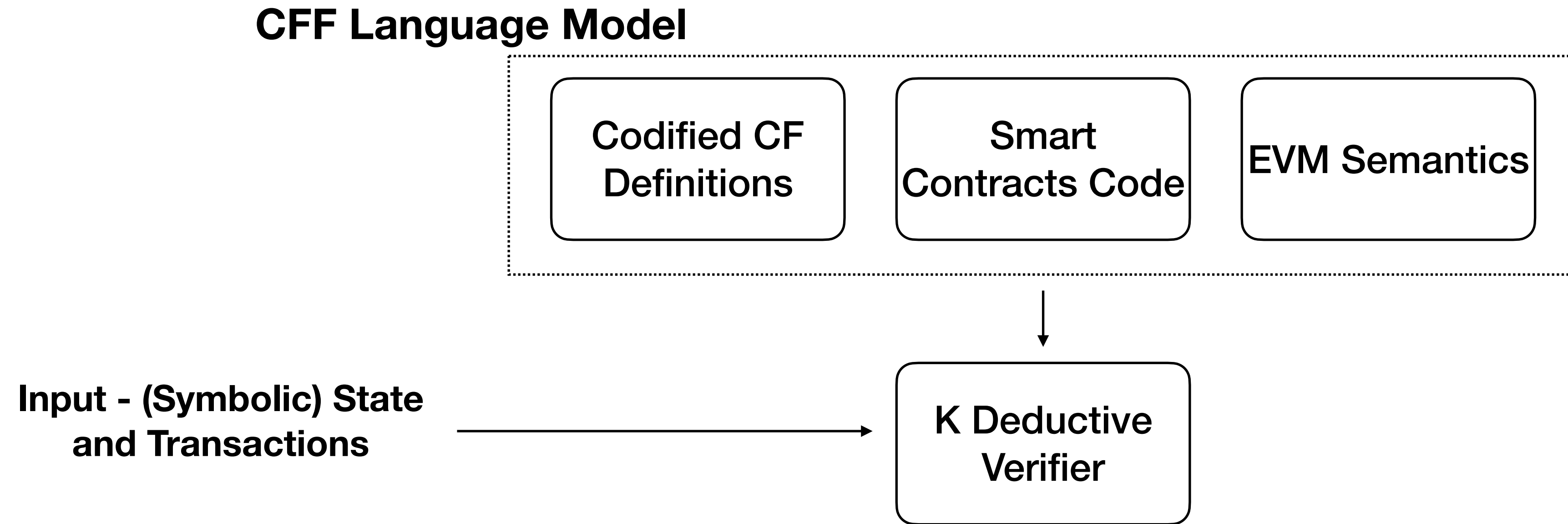
# CFF Design



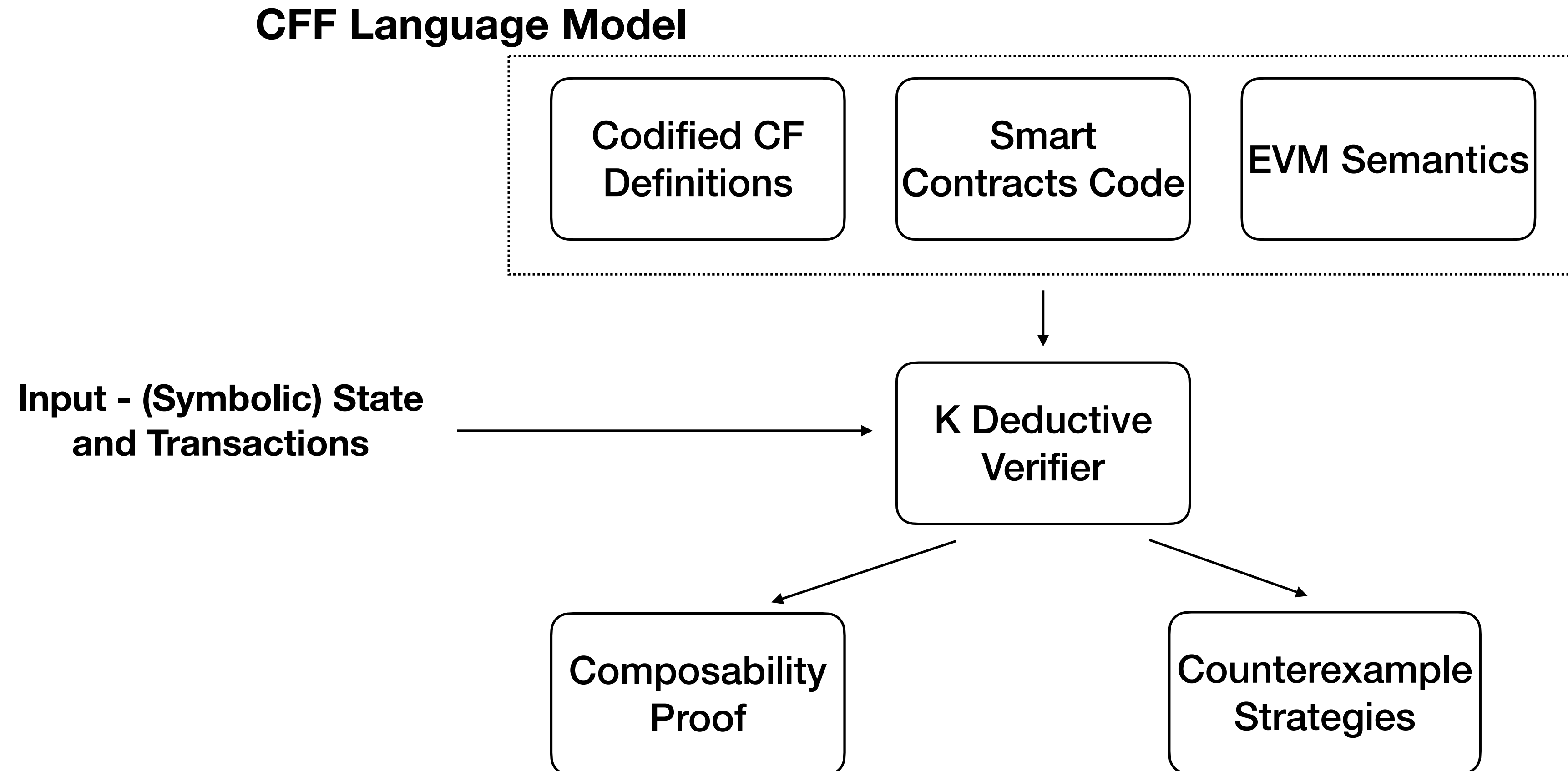
# CFF Design



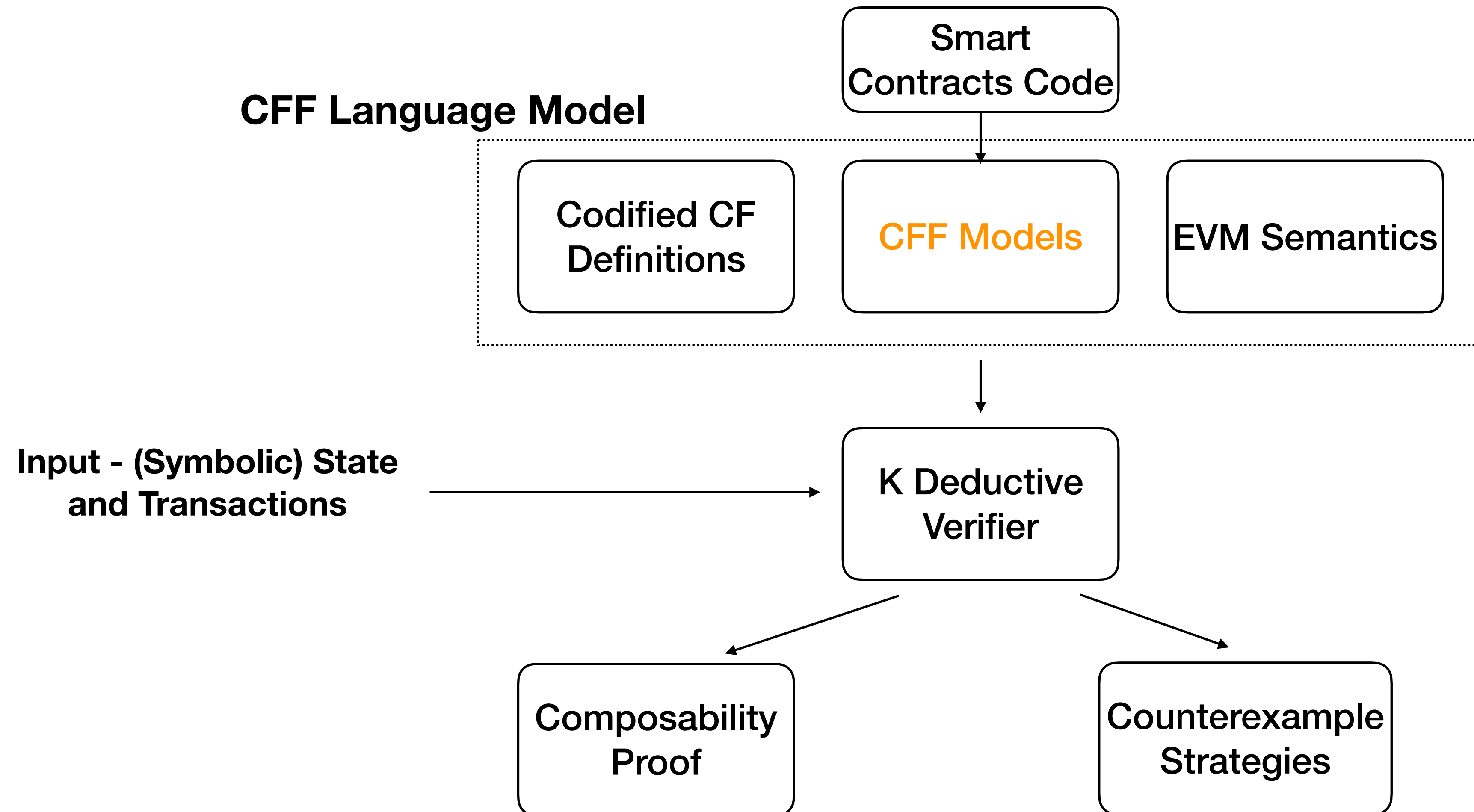
# CFF Design



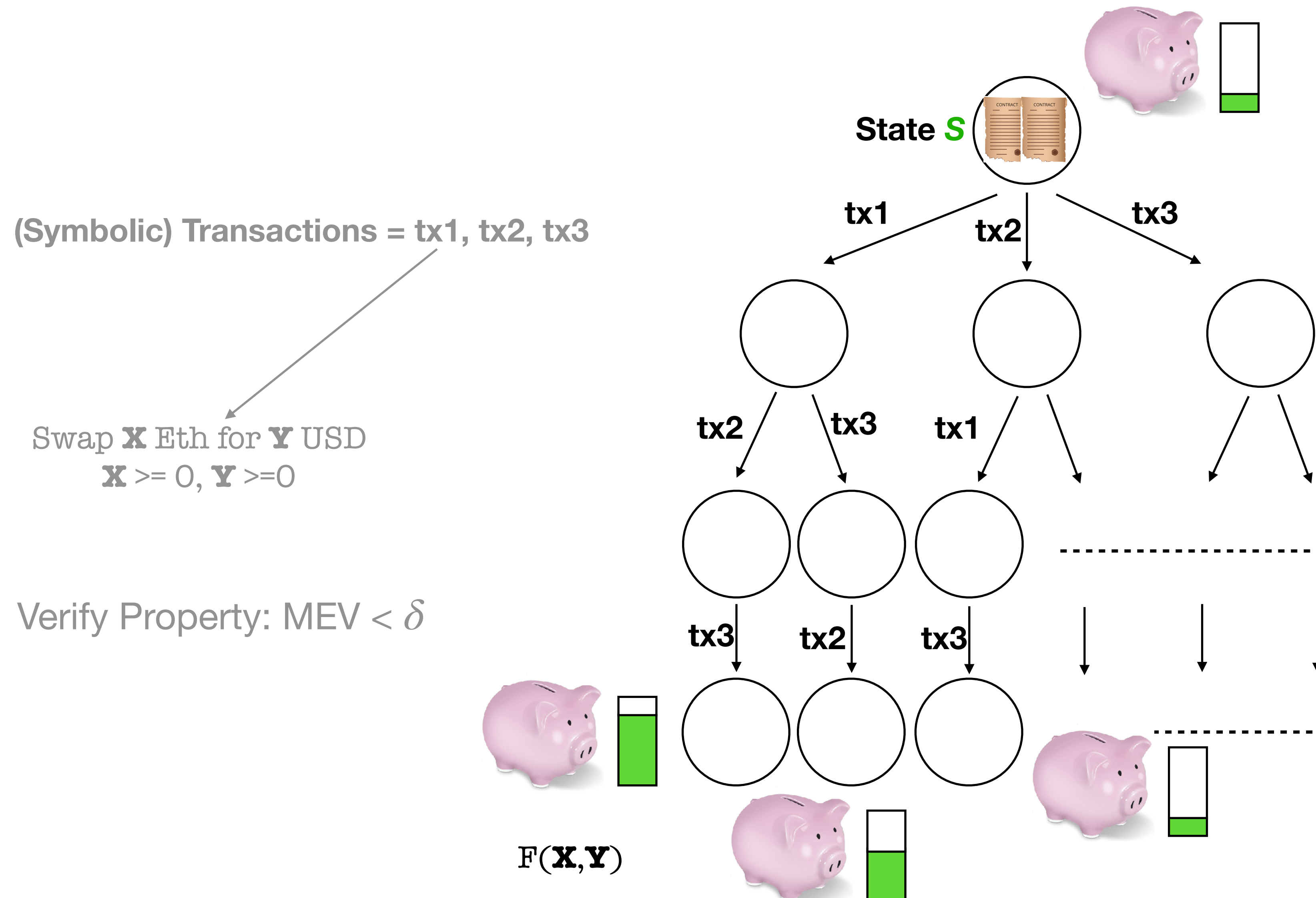
# CFF Design



# CFF Design

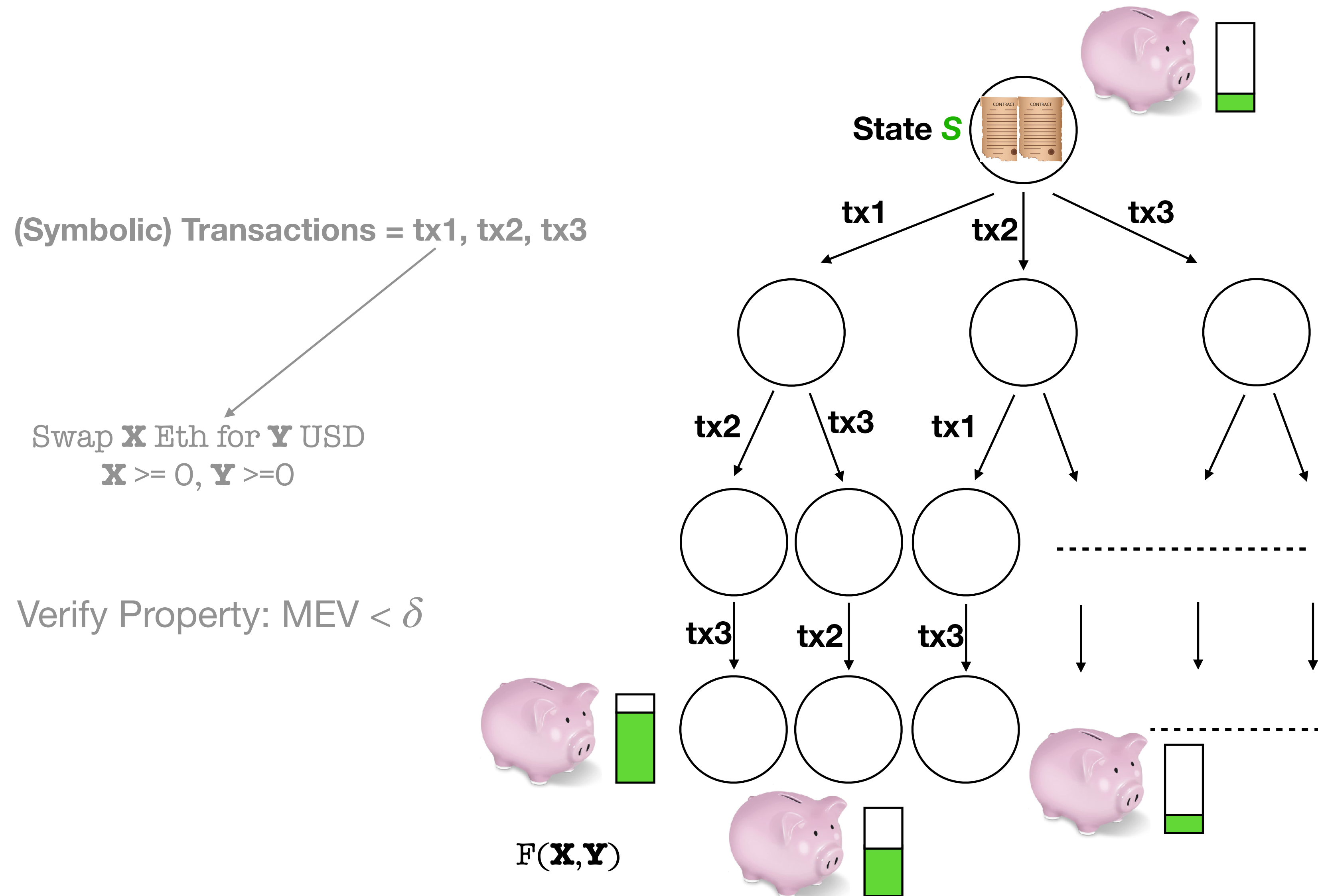


# CFF Models



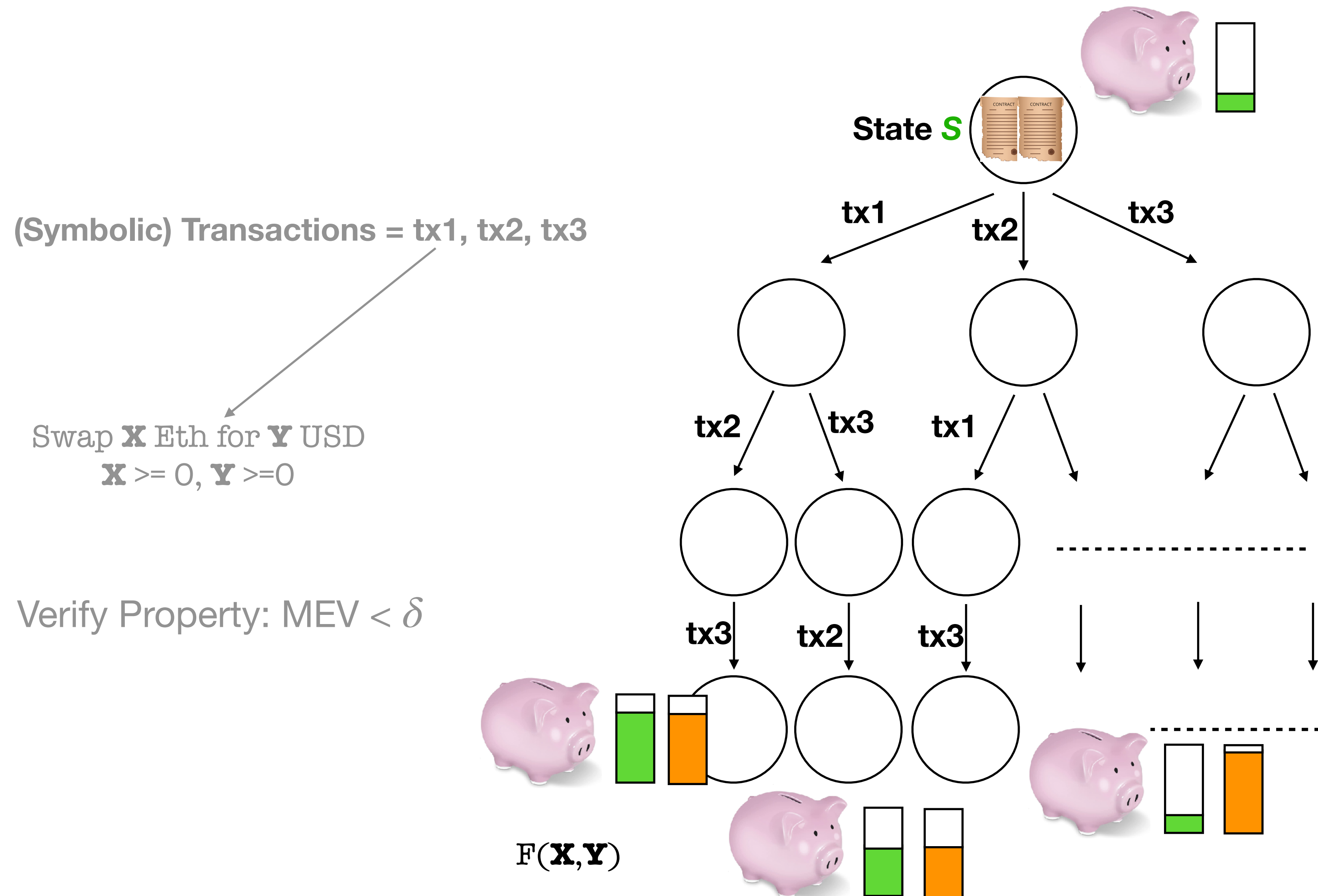


# CFF Models



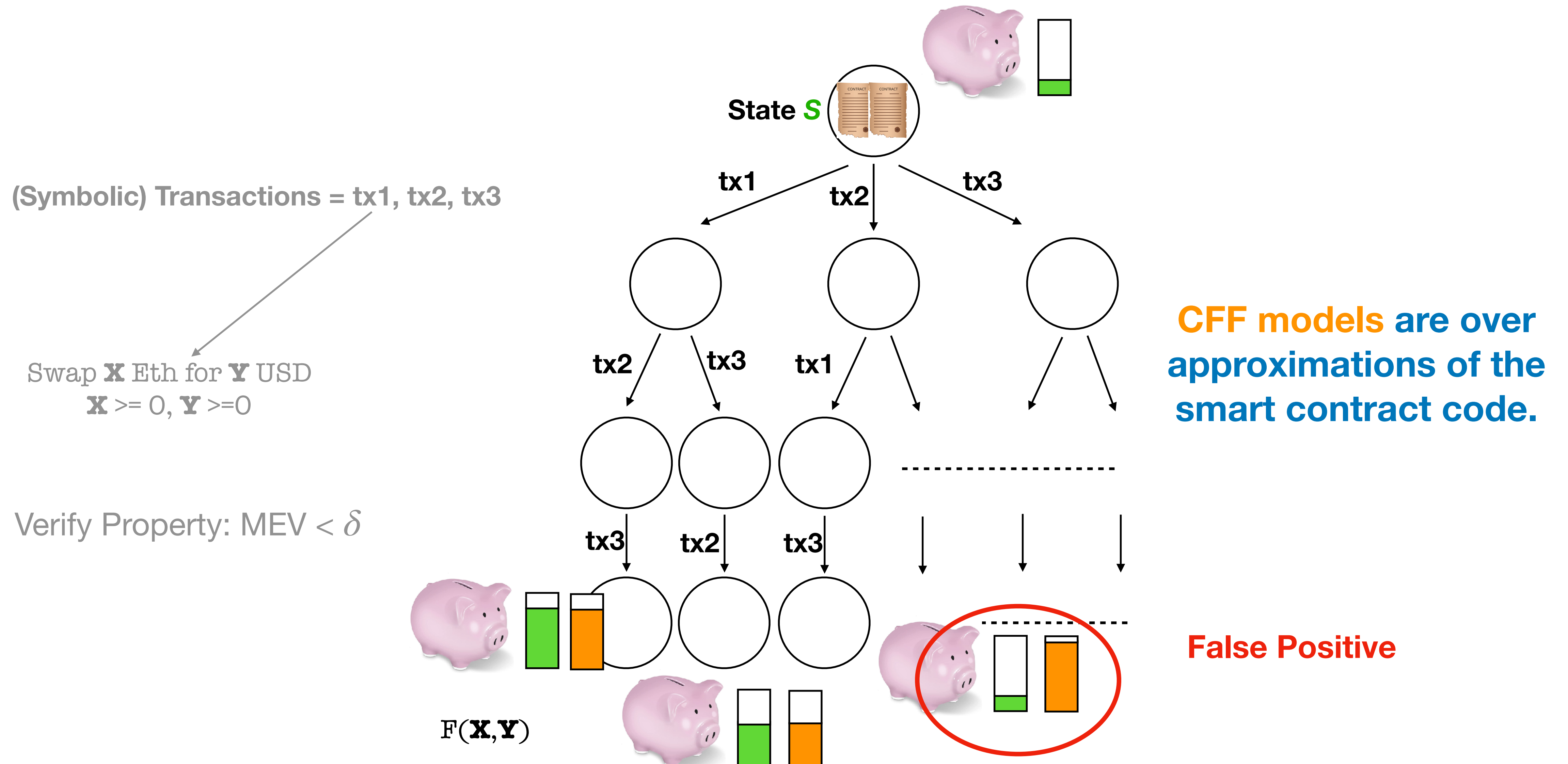
**CFF models are over approximations of the smart contract code.**

# CFF Models

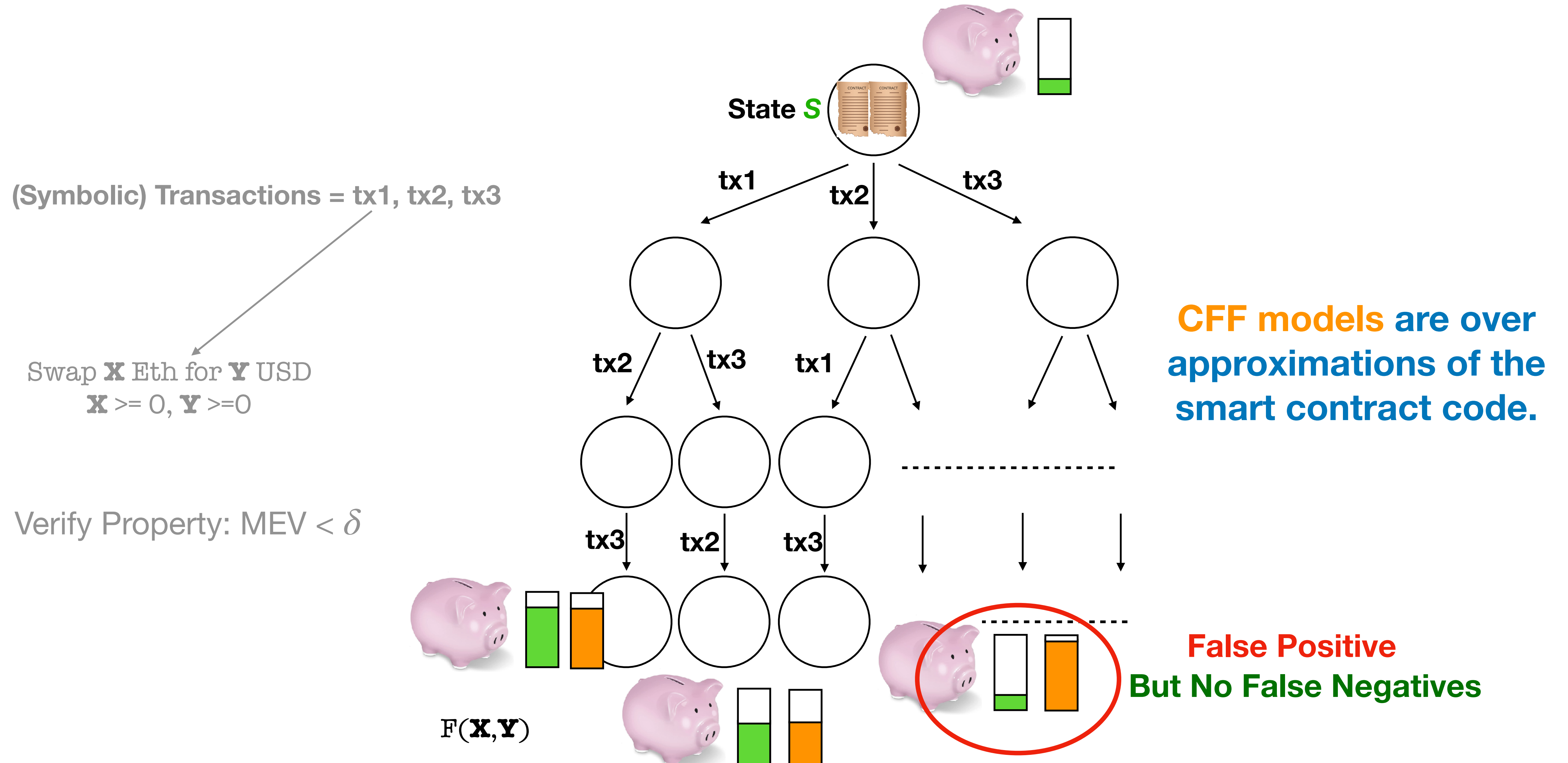


**CFF models are over approximations of the smart contract code.**

# CFF Models



# CFF Models



# 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_unitless_number(token_reserve))
    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

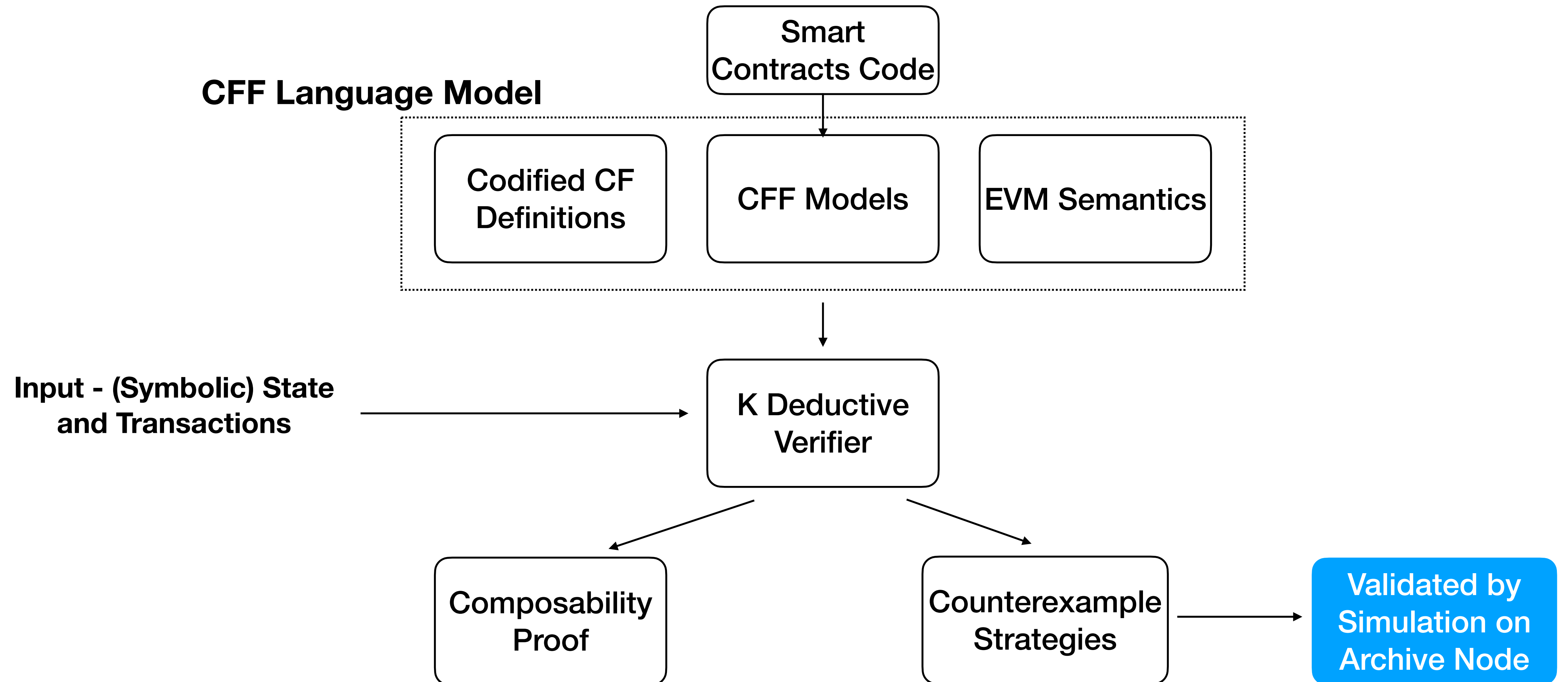
```
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp,
assert deadline >= block.timestamp and (eth_sold > 0 and min_tokens > 0)
    token_reserve: uint256 = self.token.balanceOf(self)
    tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unitless_number(token_reserve))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





# More Scaling Optimisations

## 1. General Optimisations

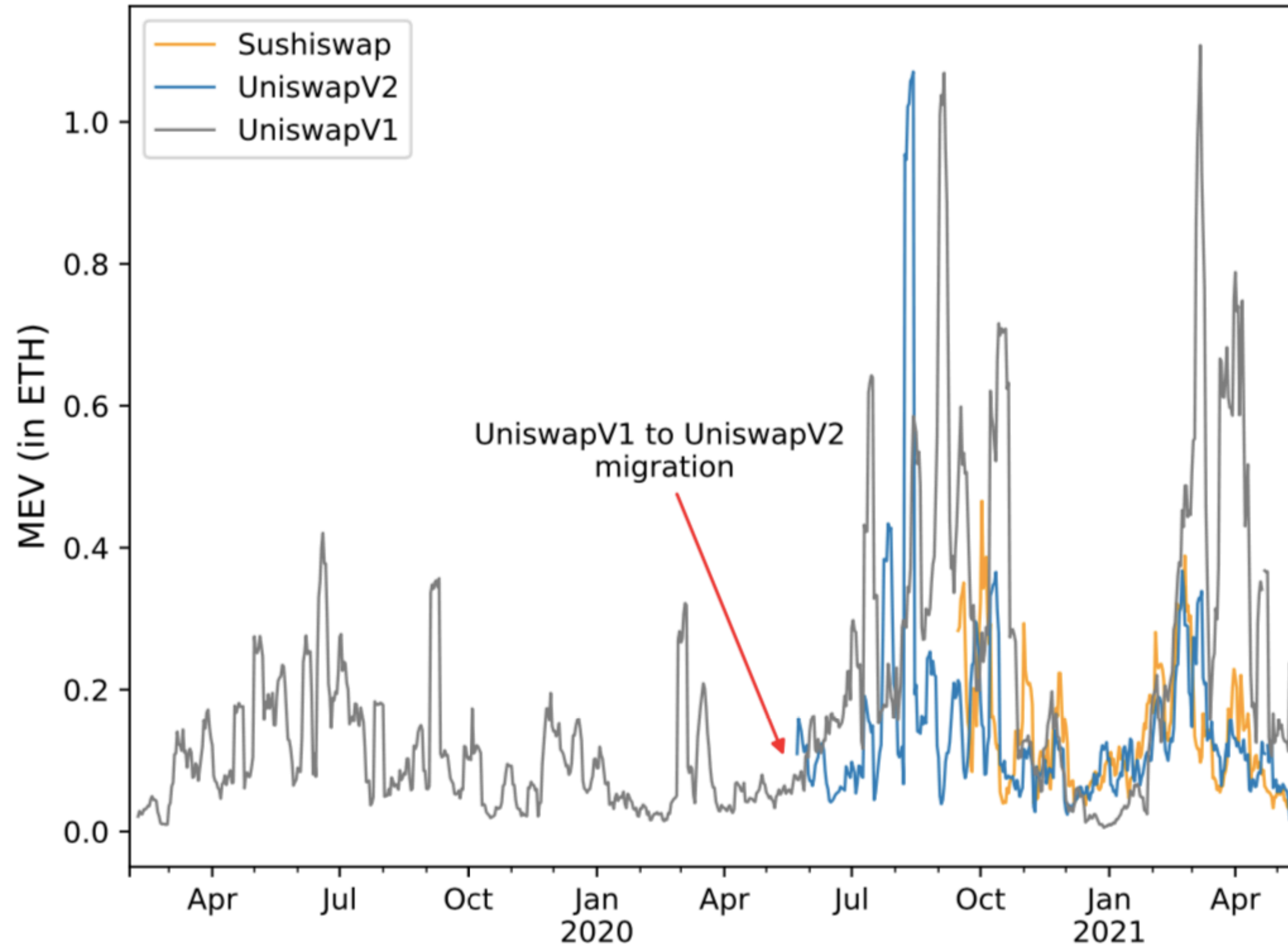
1. Transactions for a sender need to be serialised using “nonces”. Many invalid orderings are equivalent
2. Reorderings across different non interacting contracts are equivalent
3. Randomised reorderings lead to better convergence in practice.

## 2. Contract Specific Optimisations

1. Uniswap-like AMMs are path independent

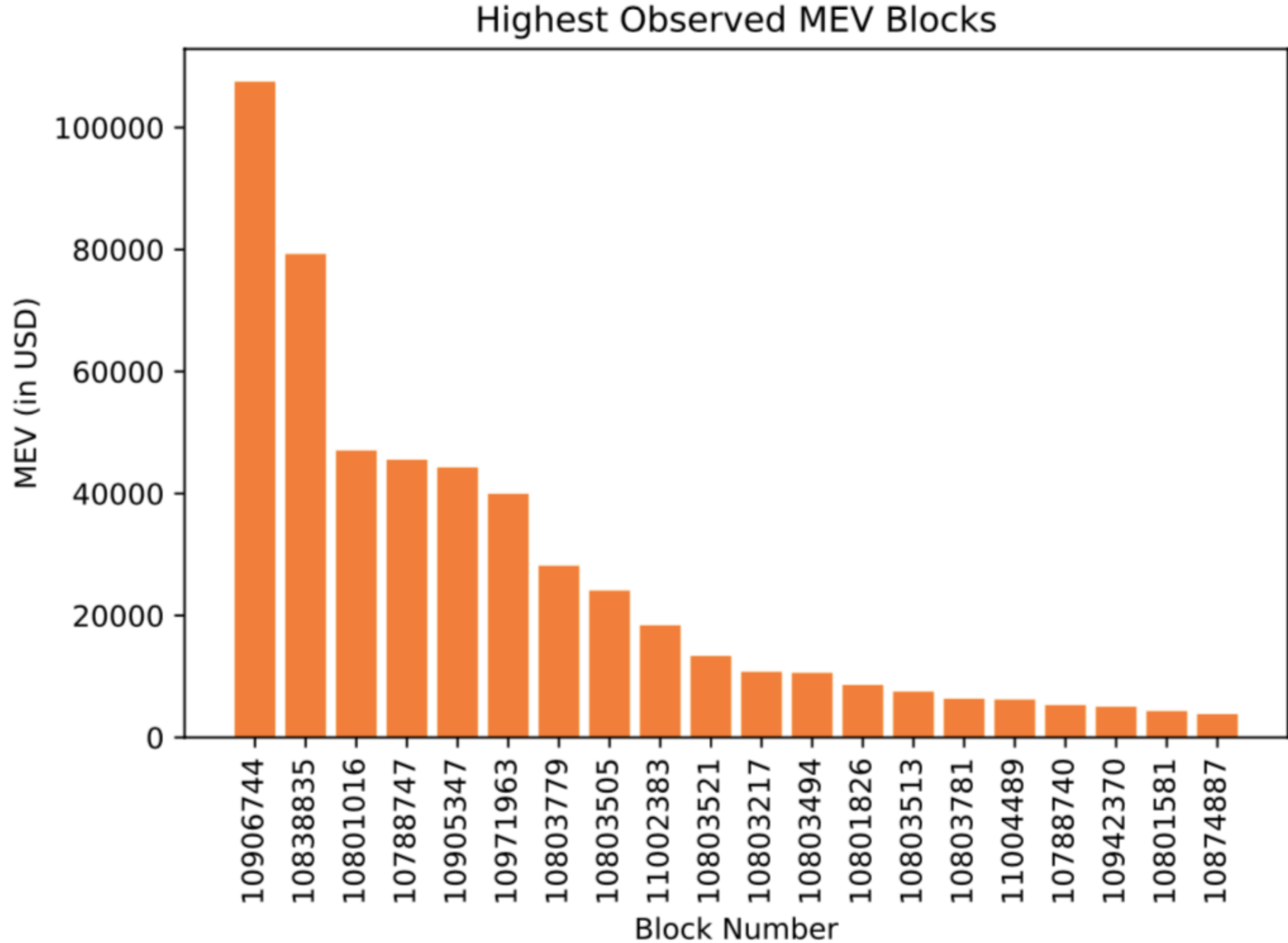
# CFF Evaluation - AMM

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.

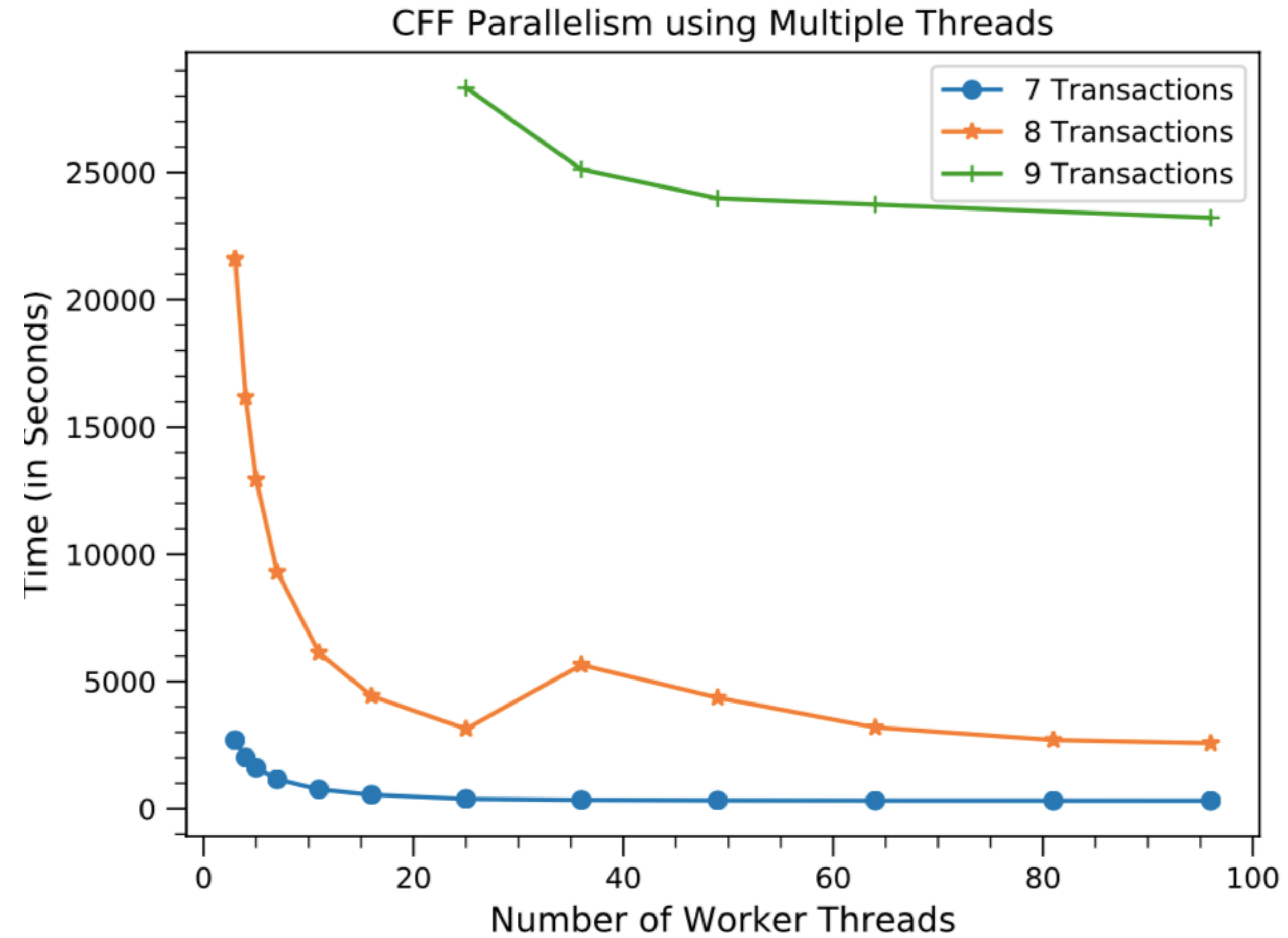
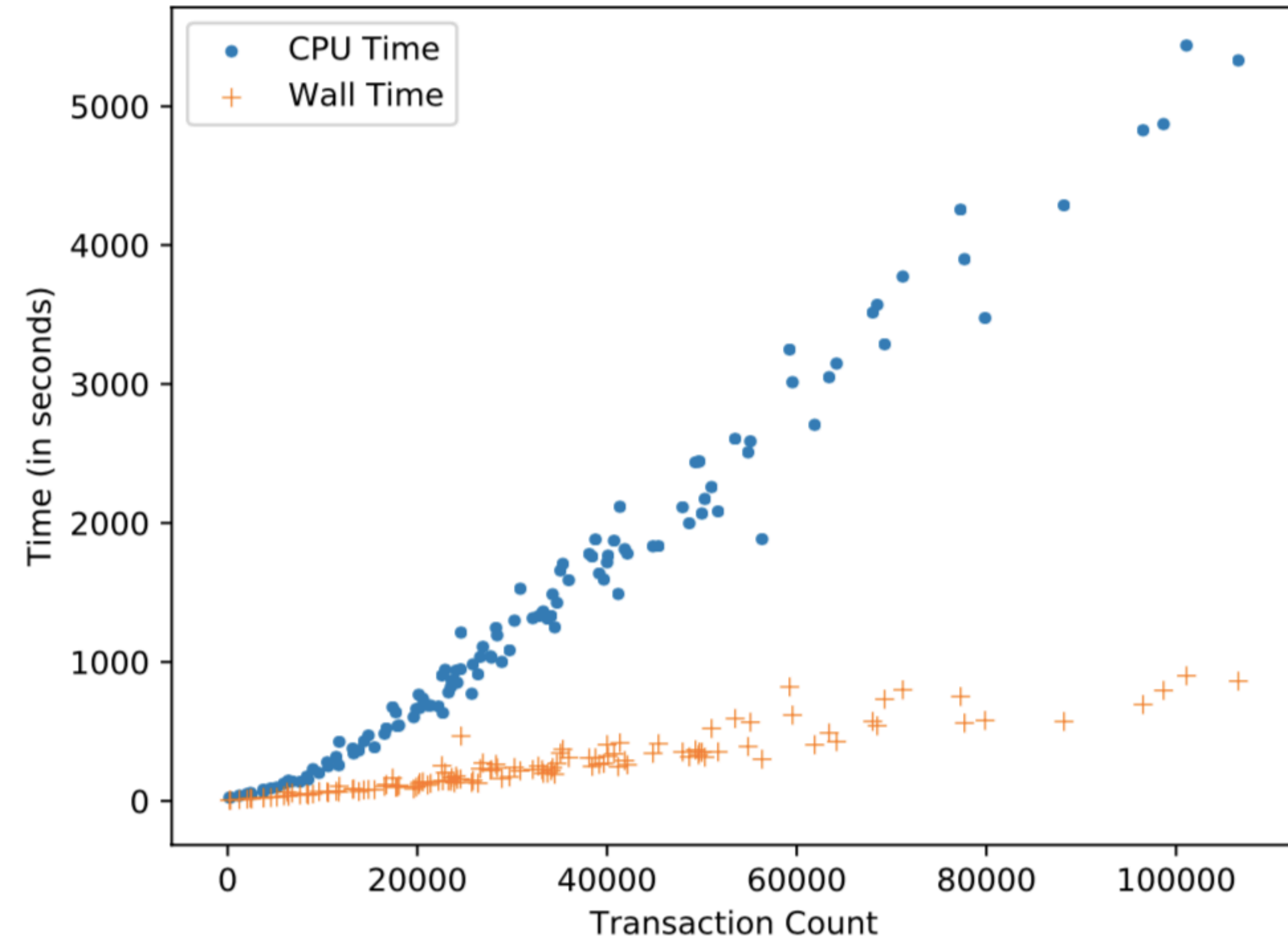
**Paper** : <https://cs.cornell.edu/~babel/cff.pdf>

**Github** : <https://github.com/defi-formal/cff>

**Contact** : [babel@cs.cornell.edu](mailto:babel@cs.cornell.edu)

# Appendix

# Execution and proving times



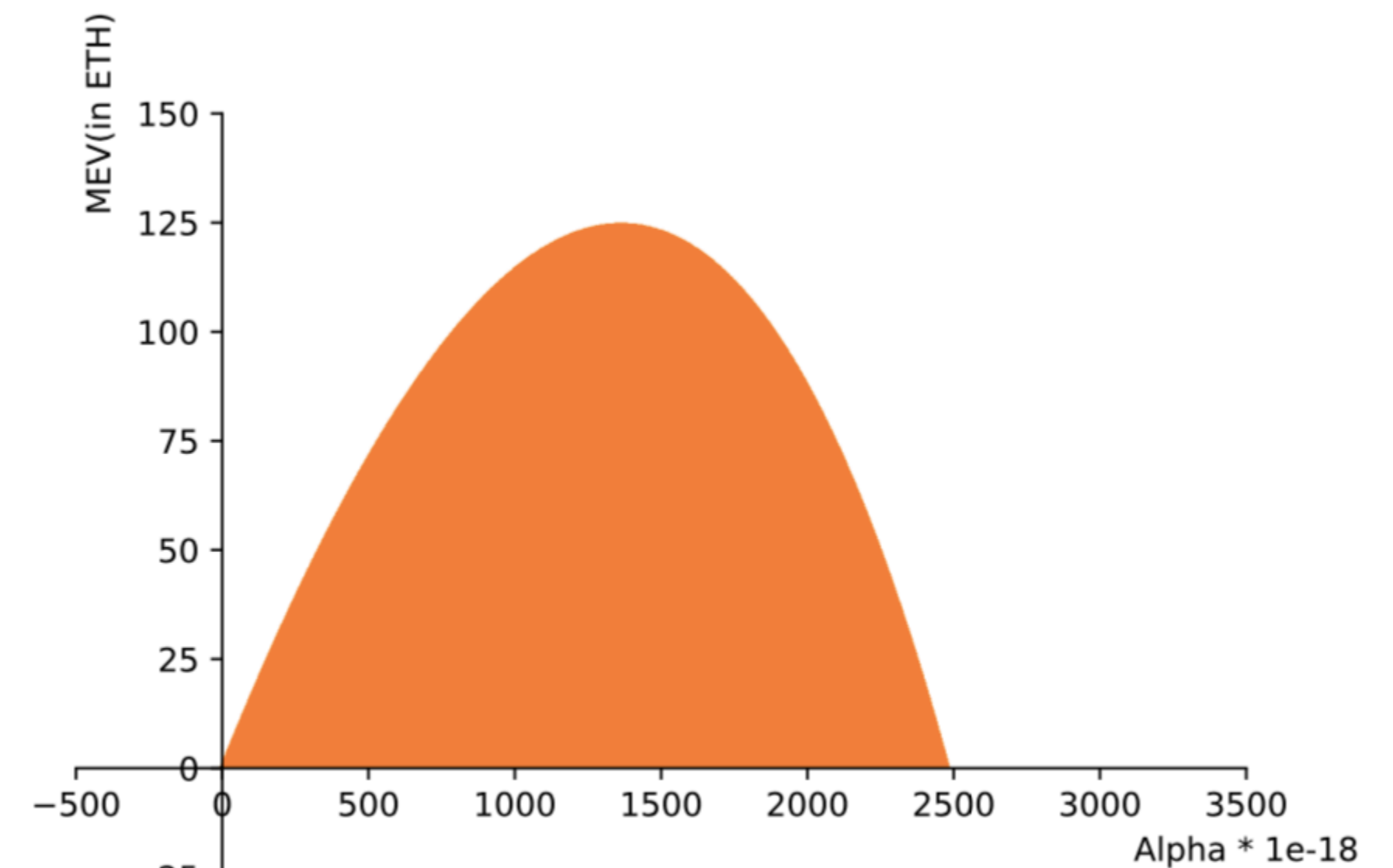
# Directions for Future Work

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



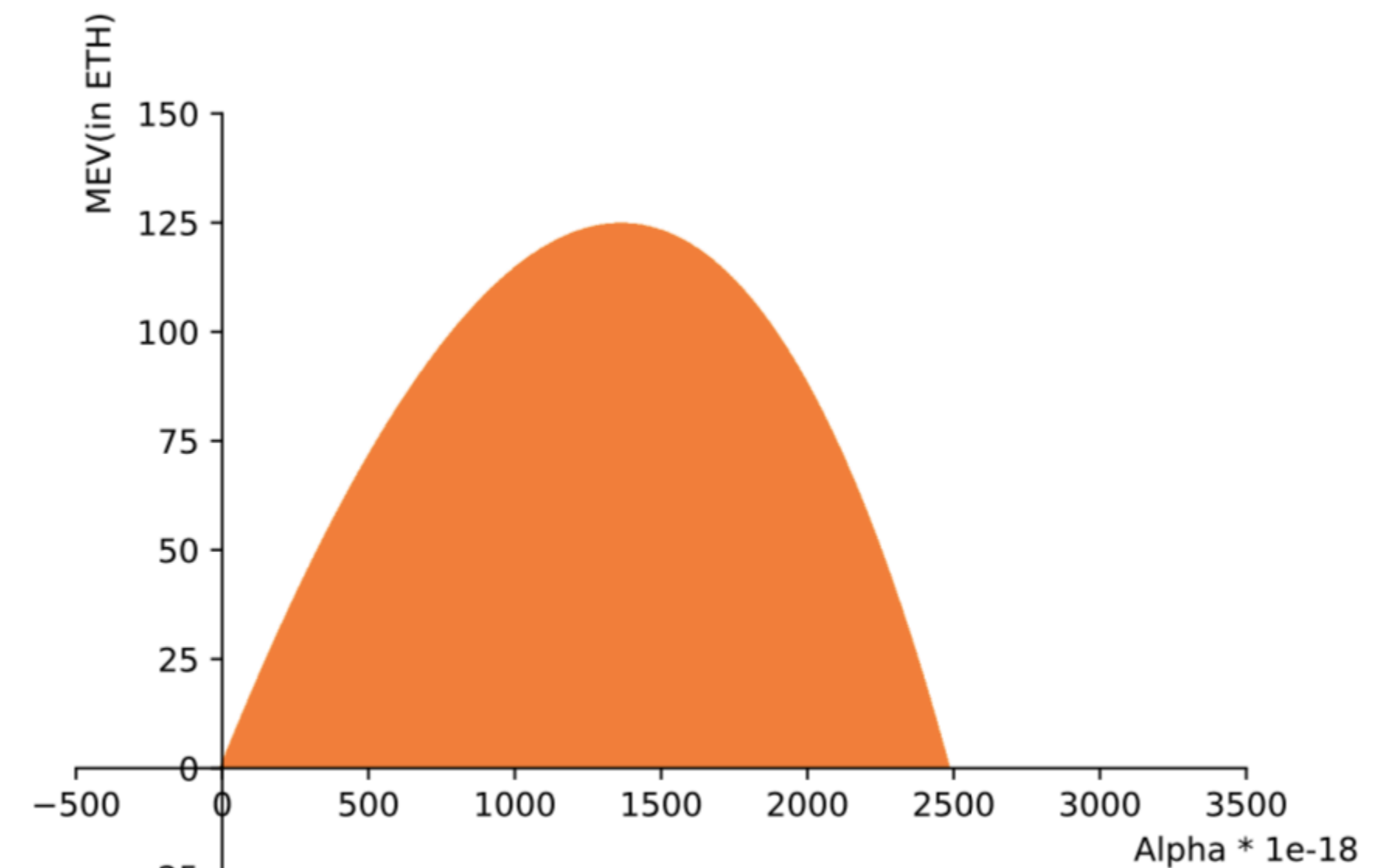
# Under the Hood- Sushiswap + Uniswap

```
1 claim <k>
2 On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
  ↳ 13000000000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
3 On Sushiswap Miner swaps for ETH by providing Alpha: Int COMP and 0 ETH with change 0 fee 0 ;
4 On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;
5
6 => .K
7 </k>
8 <S> (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 49835502094518088853633
  ↳ (UniswapV2 in COMP) |-> 5945498629669852264883 (UniswapV2 in ETH) |-> 2615599823603823616442 =>
  ↳ ?S:Map </S>
9 <B> .List => ?_ </B>
10 requires (Alpha >Int 0) andBool (Alpha <Int 1000000000000000000000000) //10**22
11 ensures ({?S[Miner in COMP]}:>Int <=Int 0 ) andBool ({?S[Miner in ETH]}:>Int <=Int 0 )
```



# Under the Hood- Sushiswap + Uniswap

```
1 claim <k>
2 On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
  ↳ 13000000000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
3 On Sushiswap Miner swaps for ETH by providing Alpha: Int COMP and 0 ETH with change 0 fee 0 ;
4 On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;
5
6 => .K
7 </k>
8 <S> (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 49835502094518088853633
  ↳ (UniswapV2 in COMP) |-> 5945498629669852264883 (UniswapV2 in ETH) |-> 2615599823603823616442 =>
  ↳ ?S:Map </S>
9 <B> .List => ?_ </B>
10 requires (Alpha >Int 0) andBool (Alpha <Int 1000000000000000000000000) //10**22
11 ensures ({?S[Miner in COMP]}:>Int <=Int 0 ) andBool ({?S[Miner in ETH]}:>Int <=Int 0 )
```



# Under the Hood- Sushiswap + Uniswap

```
1 claim <k>
2 On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
  ↳ 13000000000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
3 On Sushiswap Miner swaps for ETH by providing Alpha: Int COMP and 0 ETH with change 0 fee 0 ;
4 On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;
5
6 => .K
7 </k>
8 <S> (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 49835502094518088853633
  ↳ (UniswapV2 in COMP) |-> 5945498629669852264883 (UniswapV2 in ETH) |-> 2615599823603823616442 =>
  ↳ ?S:Map </S>
9 <B> .List => ?_ </B>
10 requires (Alpha >Int 0) andBool (Alpha <Int 1000000000000000000000000) //10**22
11 ensures ({?S[Miner in COMP]}:>Int <=Int 0 ) andBool ({?S[Miner in ETH]}:>Int <=Int 0 )
```

