Decentralized Finance

DeFi Security

Instructors: Dan Boneh, **Arthur Gervais**, Andrew Miller, Christine Parlour, Dawn Song





Stanford University

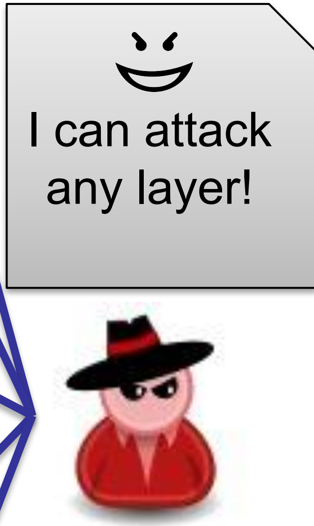
Imperial College London

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Berkeley UNIVERSITY OF CALIFORNIA

DeFi Security Affects Multiple Layer

Layer		
Third party Layer	**UI** Wallet, Website, APIs	**Other** Oracle data feed, Centralized governance
DeFi Protocol + Application Layer	**Asset** Fungible, Non-Fungible	**Atomic Composable DeFi** Exchange, Loan, Mixer, Liquidity incentive
Smart contract Layer	**Data** Block, Transaction, Contract	**Virtual Machine** Contract execution, State transition
Blockchain Layer	**Consensus** Proof-of-Work, Proof-of-Stake	**Incentive Protocol** Block reward, MEV reward, TX fee
Network Layer	**Network Services** DNS, IP, BGP	**Network Protocols** P2P overlay, Peer discovery, Data propagation



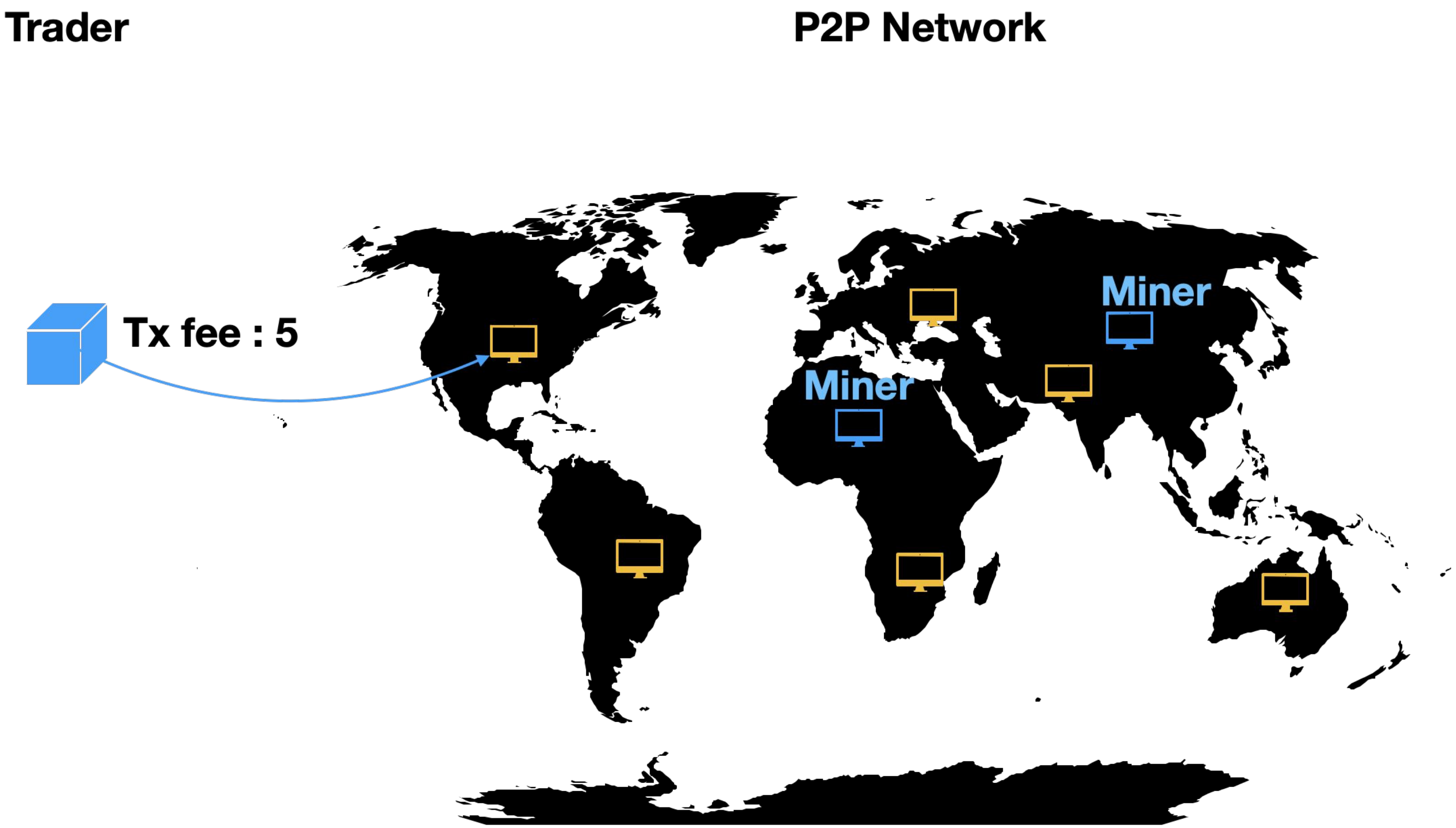
Network Layer Security

https://defi-learning.org

Network Layer

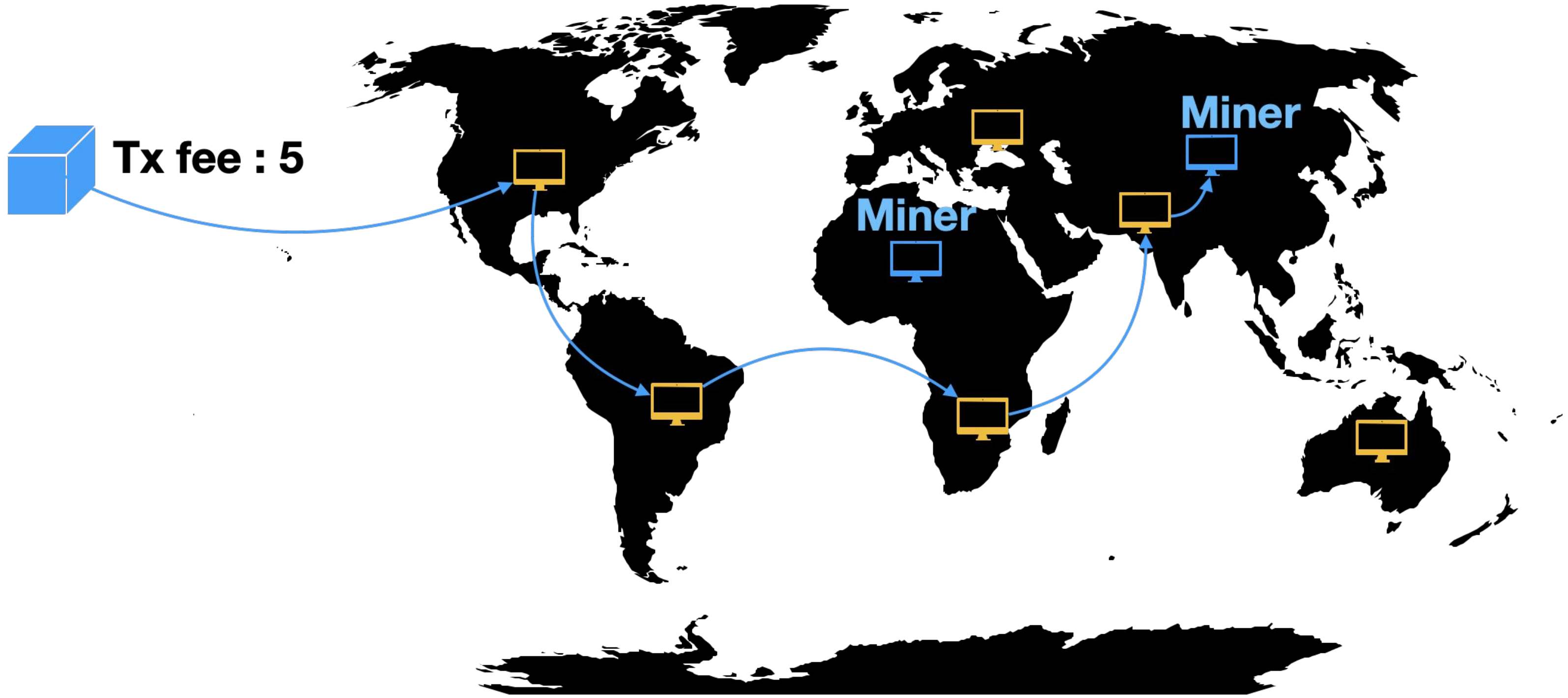
- Why Network Layer?
  - Information dissemination and propagation.
  - Latency matters!
- How many nodes?
  - Bitcoin: about 10’000 reachable full nodes (TCP/8333)
  - Ethereum:
  - Dogecoin:
- What type of nodes exist?
  - Full nodes
  - Light nodes

Exchange Transaction Propagation

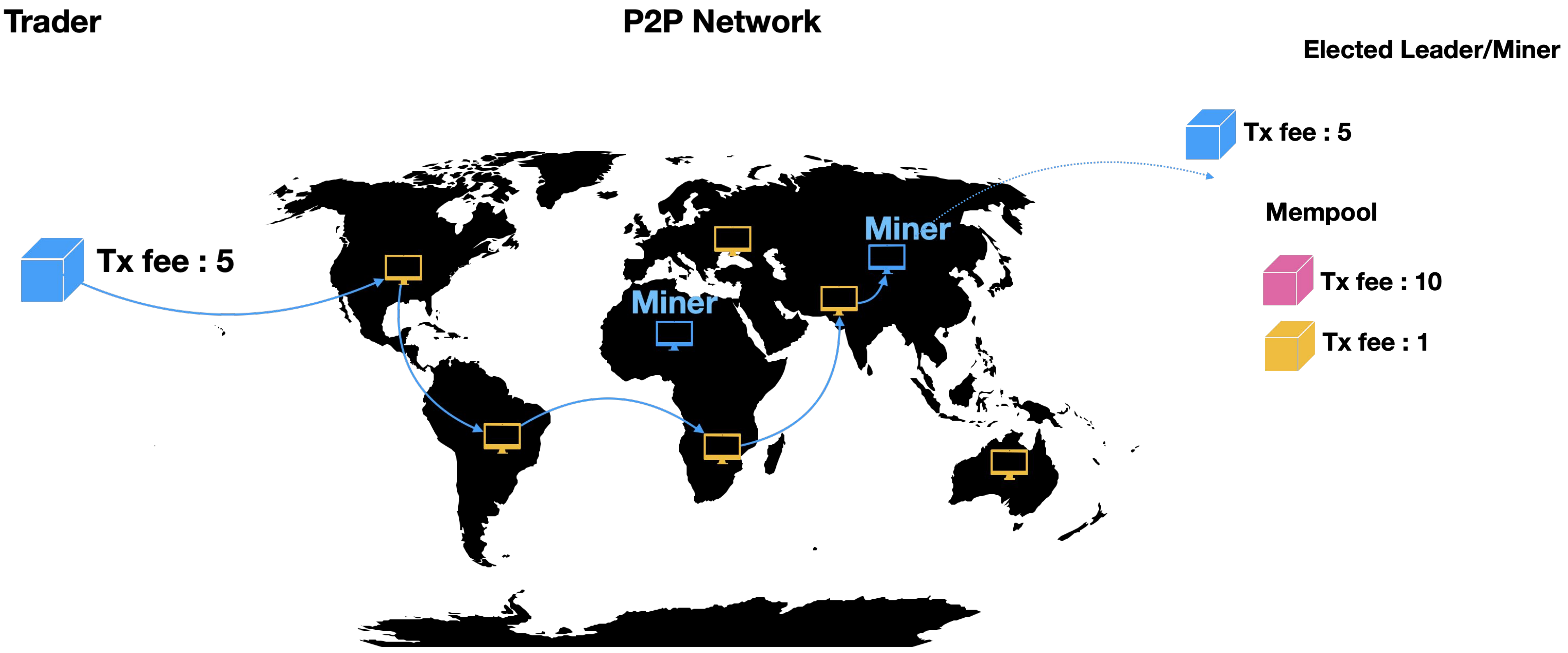


Exchange Transaction Propagation

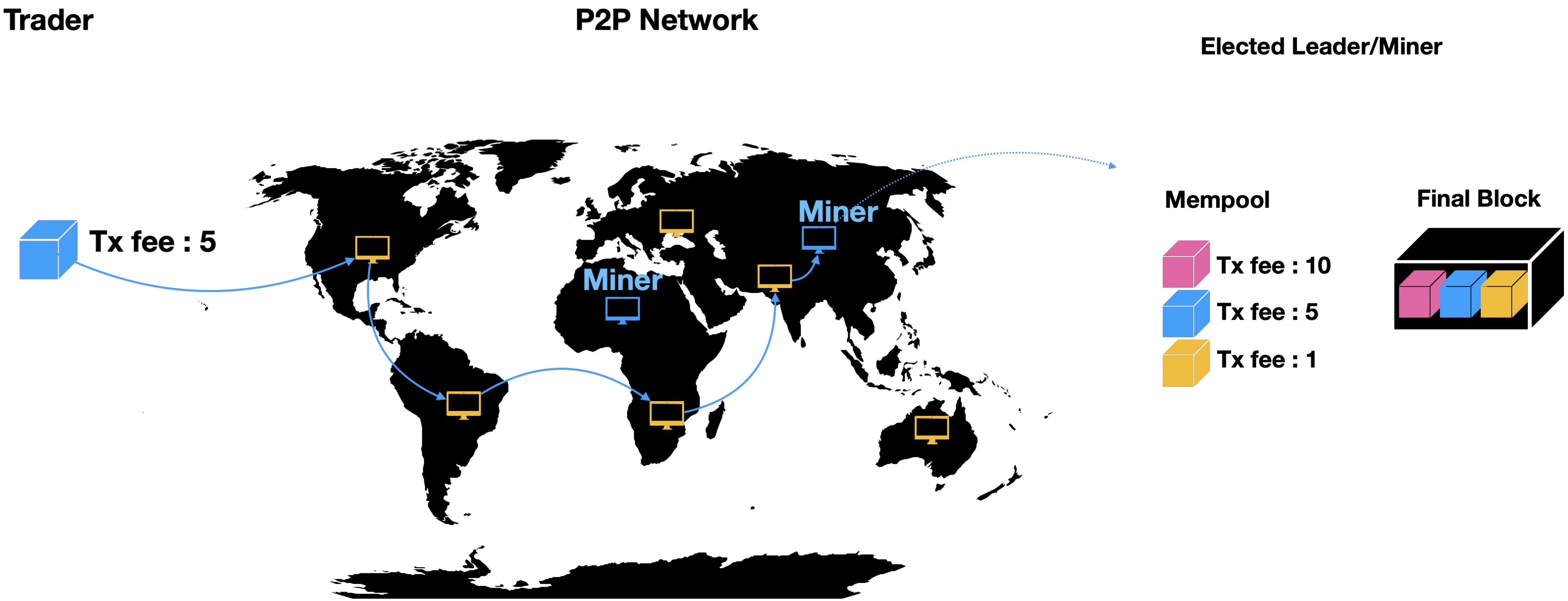
Trader **P2P Network**



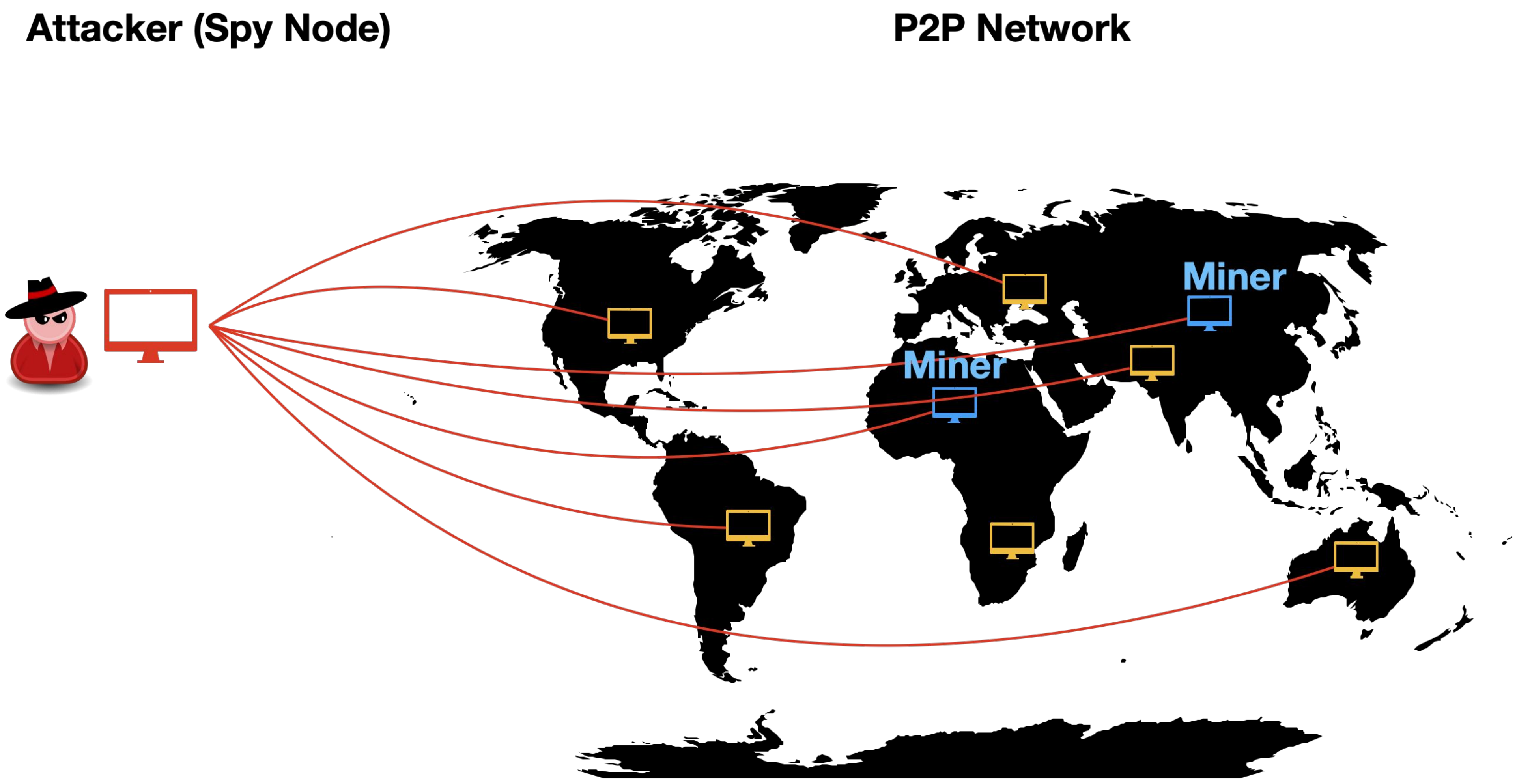
Exchange Transaction Propagation



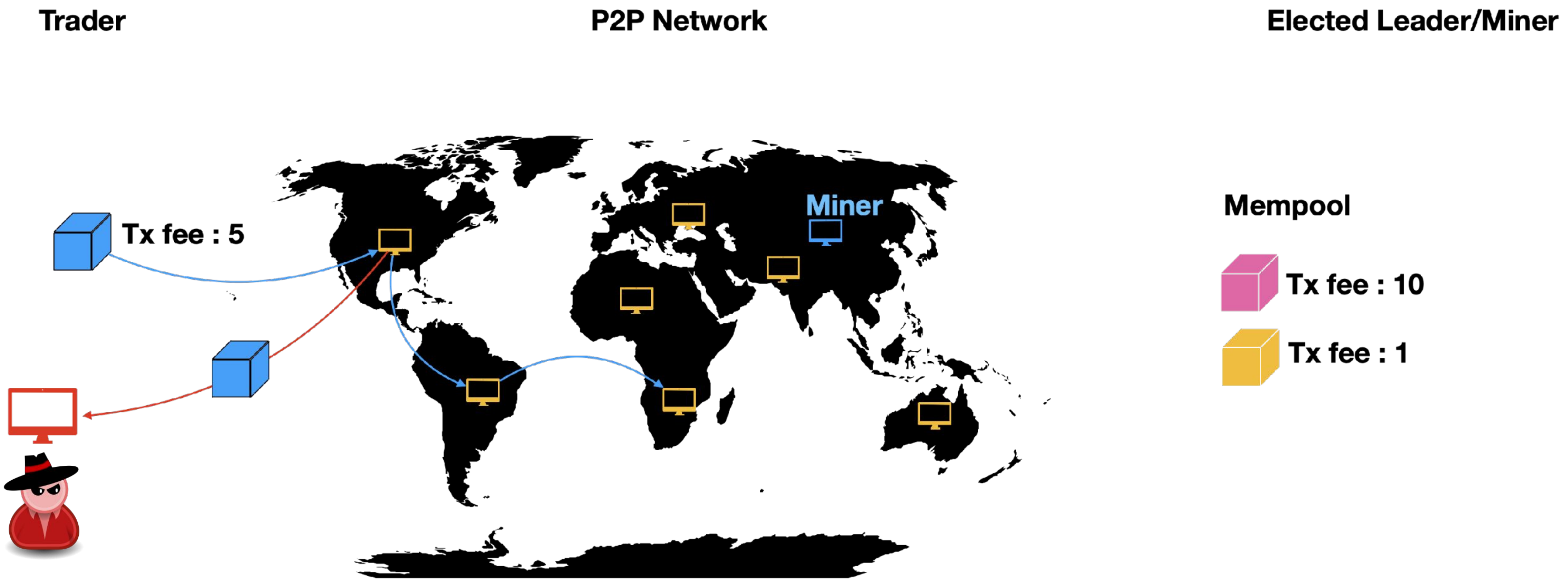
Exchange Transaction Propagation



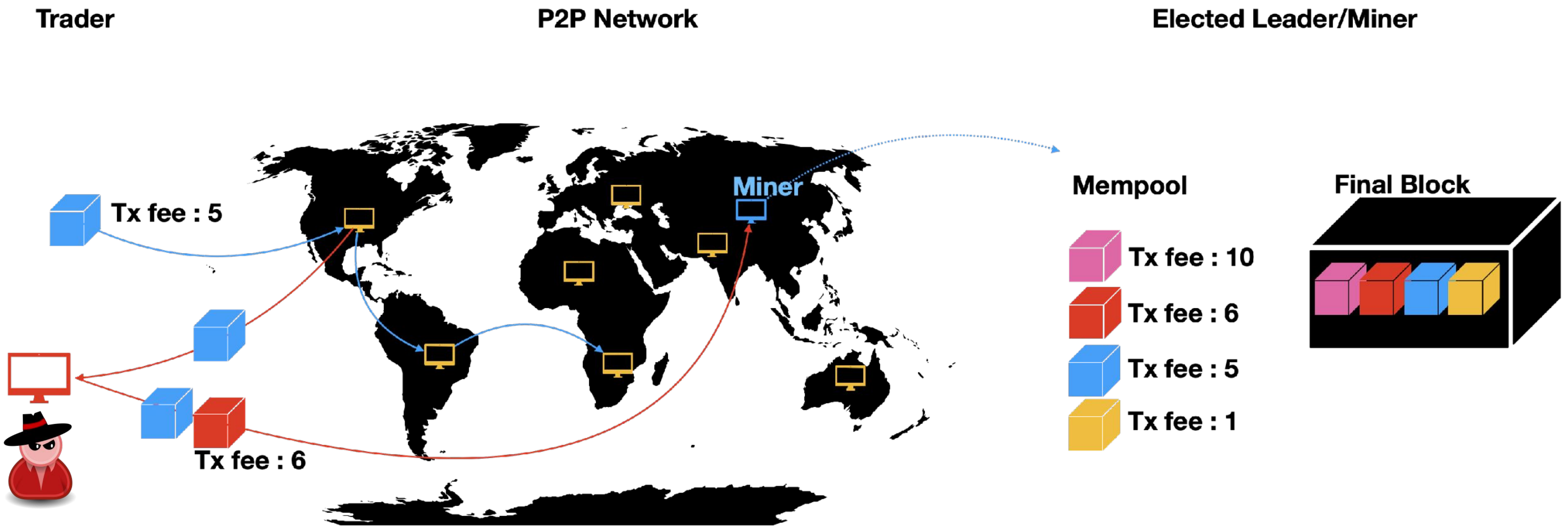
Network Layer – Spy Node



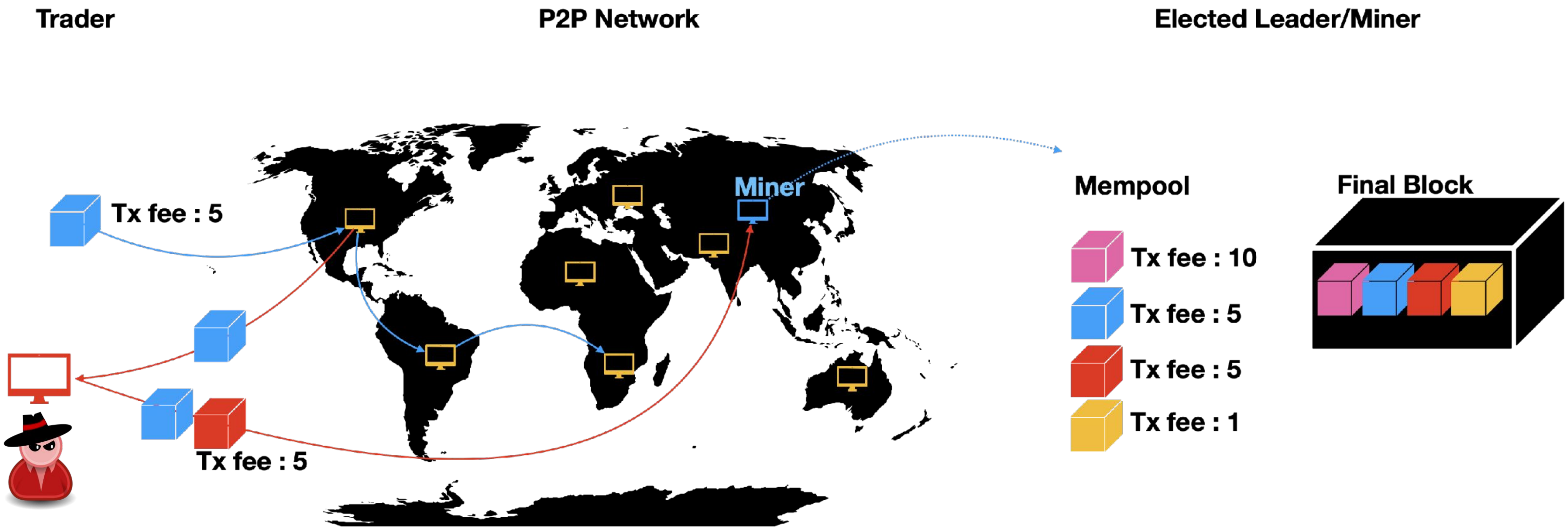
Network Layer – Spy Node



Front-running



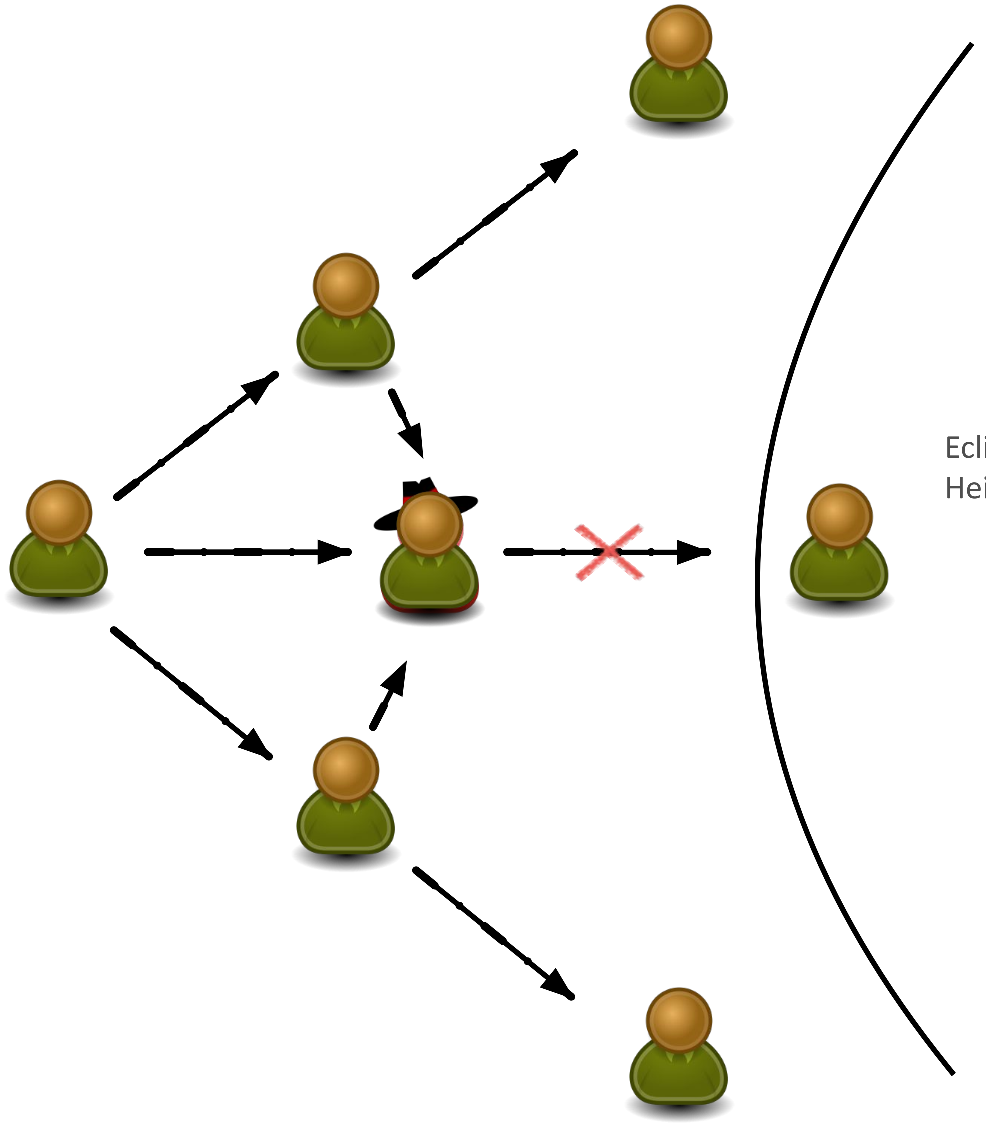
Back-running



Eclipse Attacks

https://defi-learning.org

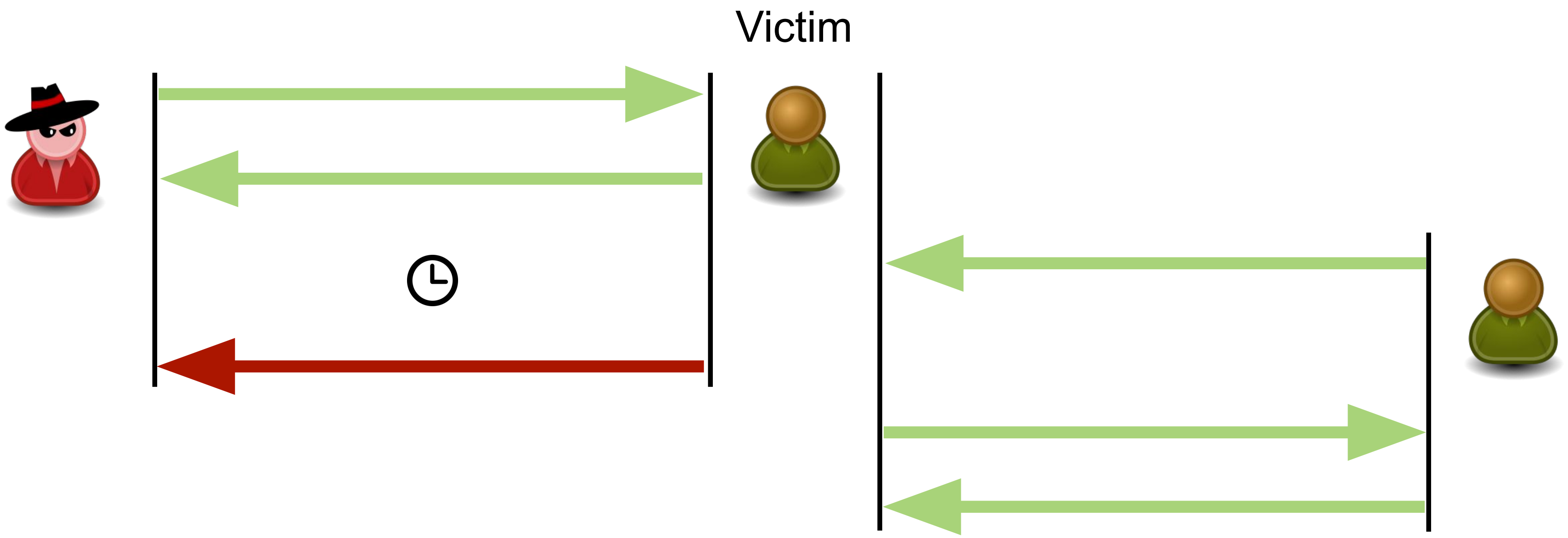
Eclipse Attacks



Eclipse attacks
Heilman et al., Usenix '15

Denial of service
Double spending

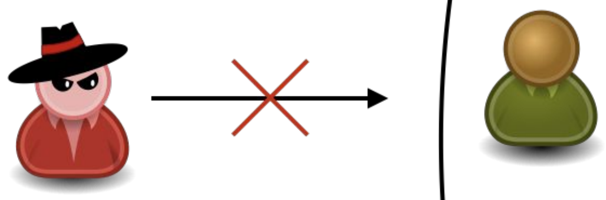
Request timeouts



Block timeout: 20 minutes
Transaction timeout: 2 minutes

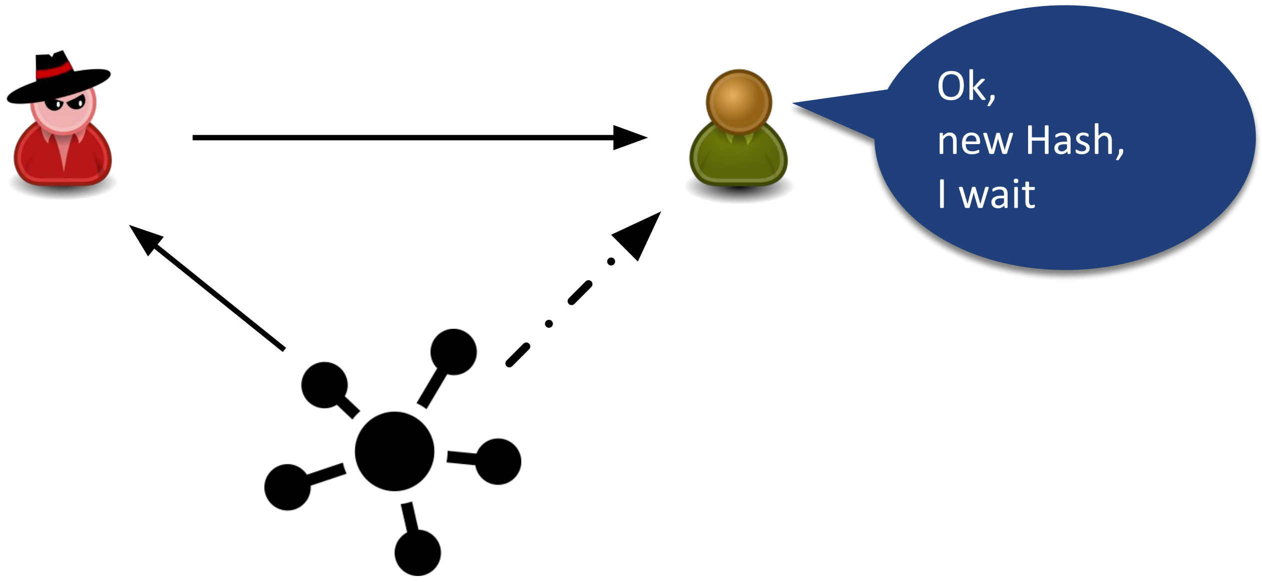
Security Implications

- Adversary
  - Blinds victim from blocks and transaction > 20 min
  - Experimental validation
- Impact
  - Double spend transactions
  - Aggravated selfish mining
  - Network wide Denial of Service
- Mitigations
  - Hardening measures
  - Estimate waiting time for secure transactions



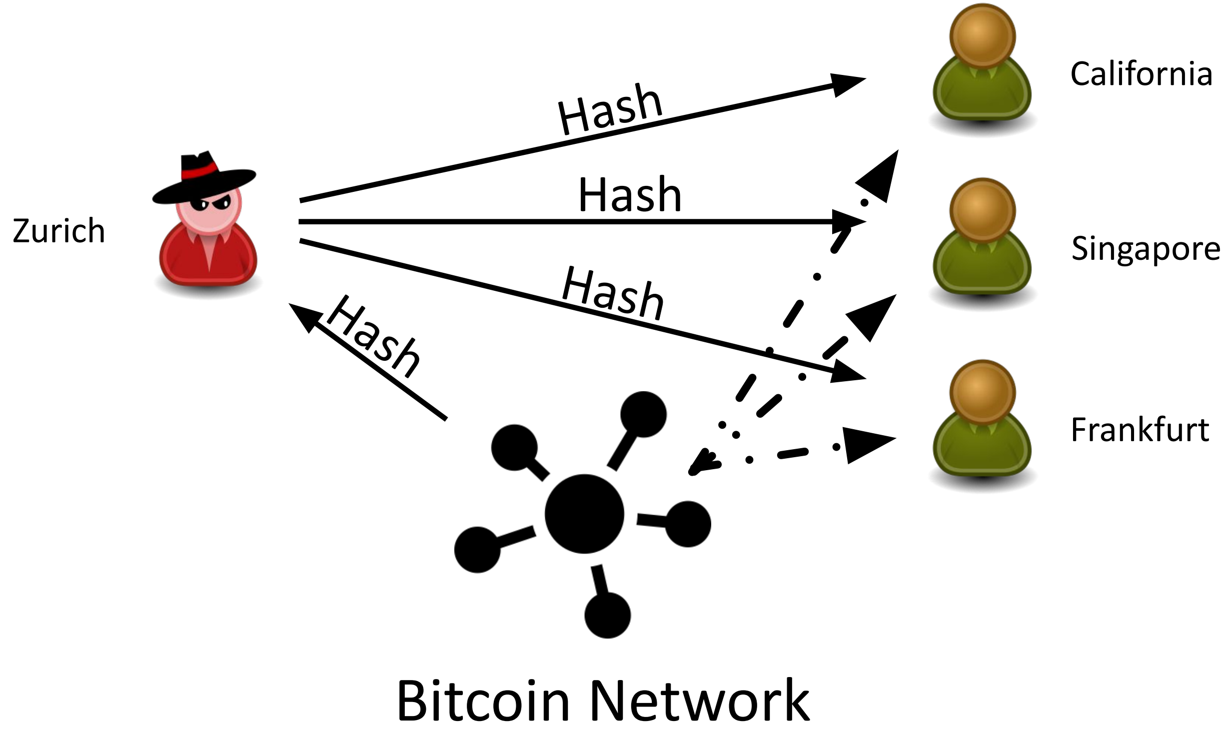
Eclipse Requirements

1. Must be **first** peer to advertise Transaction/Block



2. Victim should wait
   - Block timeout: 20 minutes
   - Transaction timeout: 2 minutes

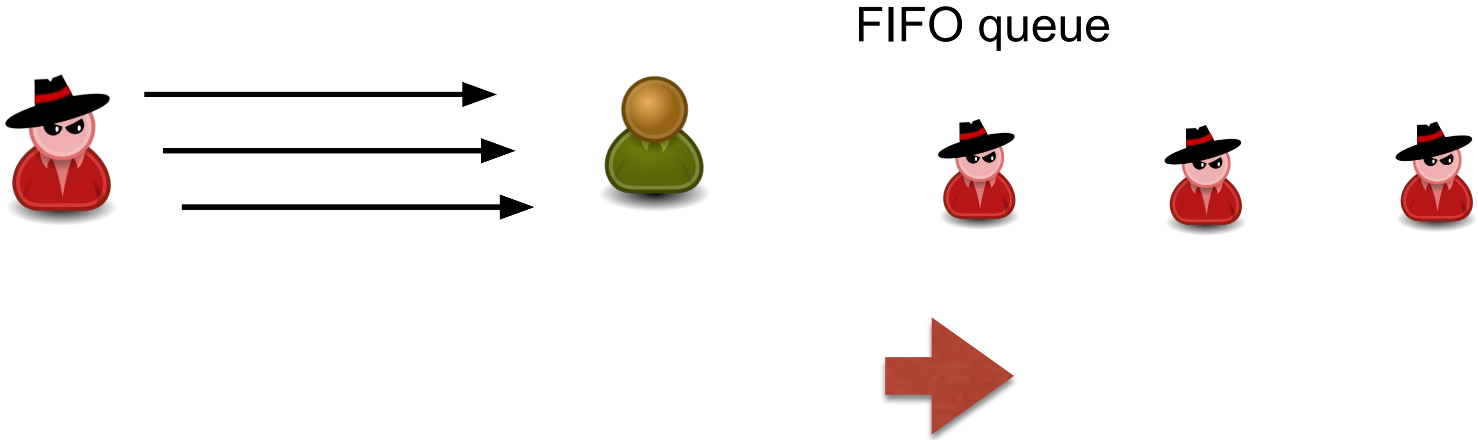
Being First on the Network Layer



Connections of Adversary	**40**	**80**	**200**	**800**
Connections of Victim	40	40	40	40
Average success in being first	0.44± 0.14	0.57± 0.20	0.80± 0.14	**0.89± 0.07**

Network Layer Timeouts

- Transactions
  - After 2 minutes request from other peer (FIFO)



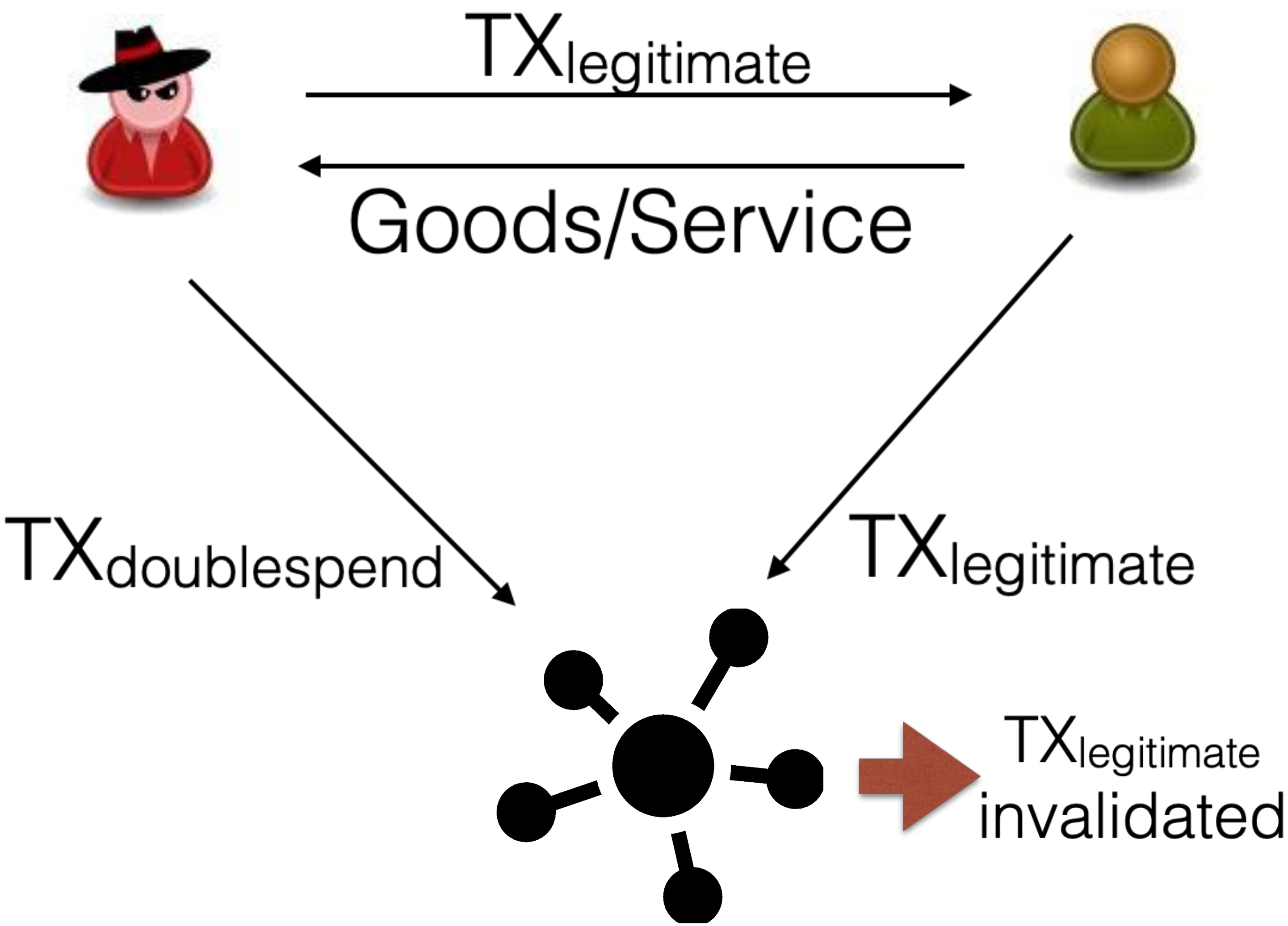
- Blocks (older Bitcoin version)
  - After 20 minutes disconnect and do nothing
  - If received header, disconnect and request block from another peer

Blockchain Layer Security

Why Blockchain Layer?

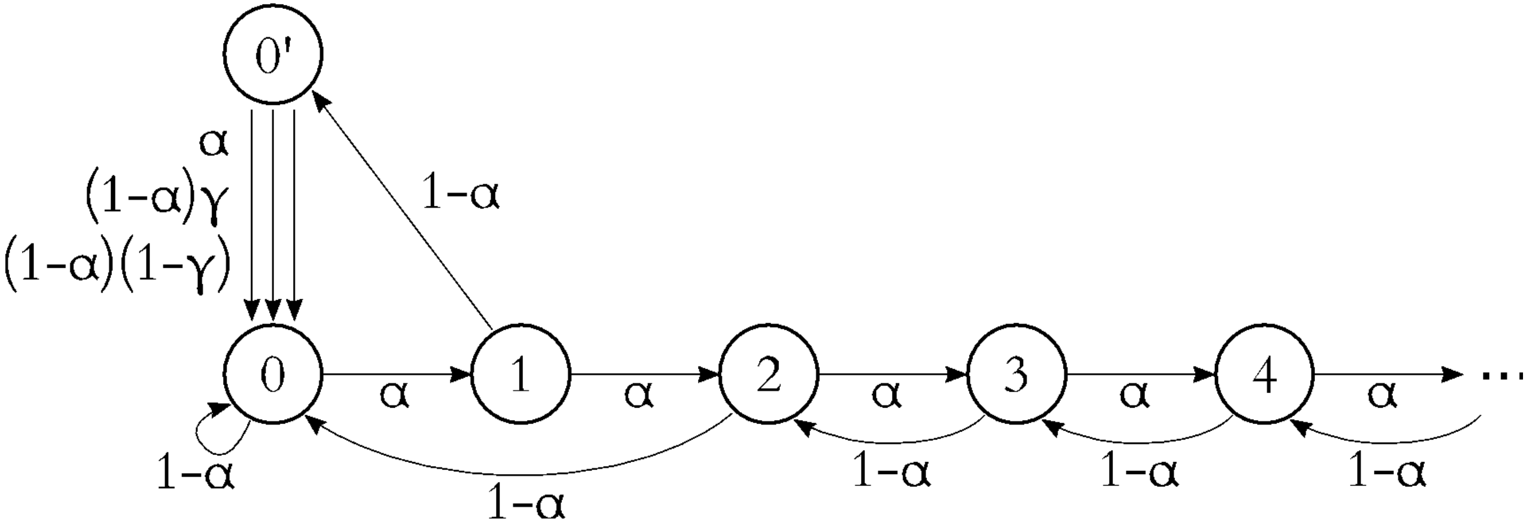
- Double-Spending
- Selfish Mining
- Undercutting
- Bribery

Double-Spending



Increasing Mining Advantage with an Eclipse

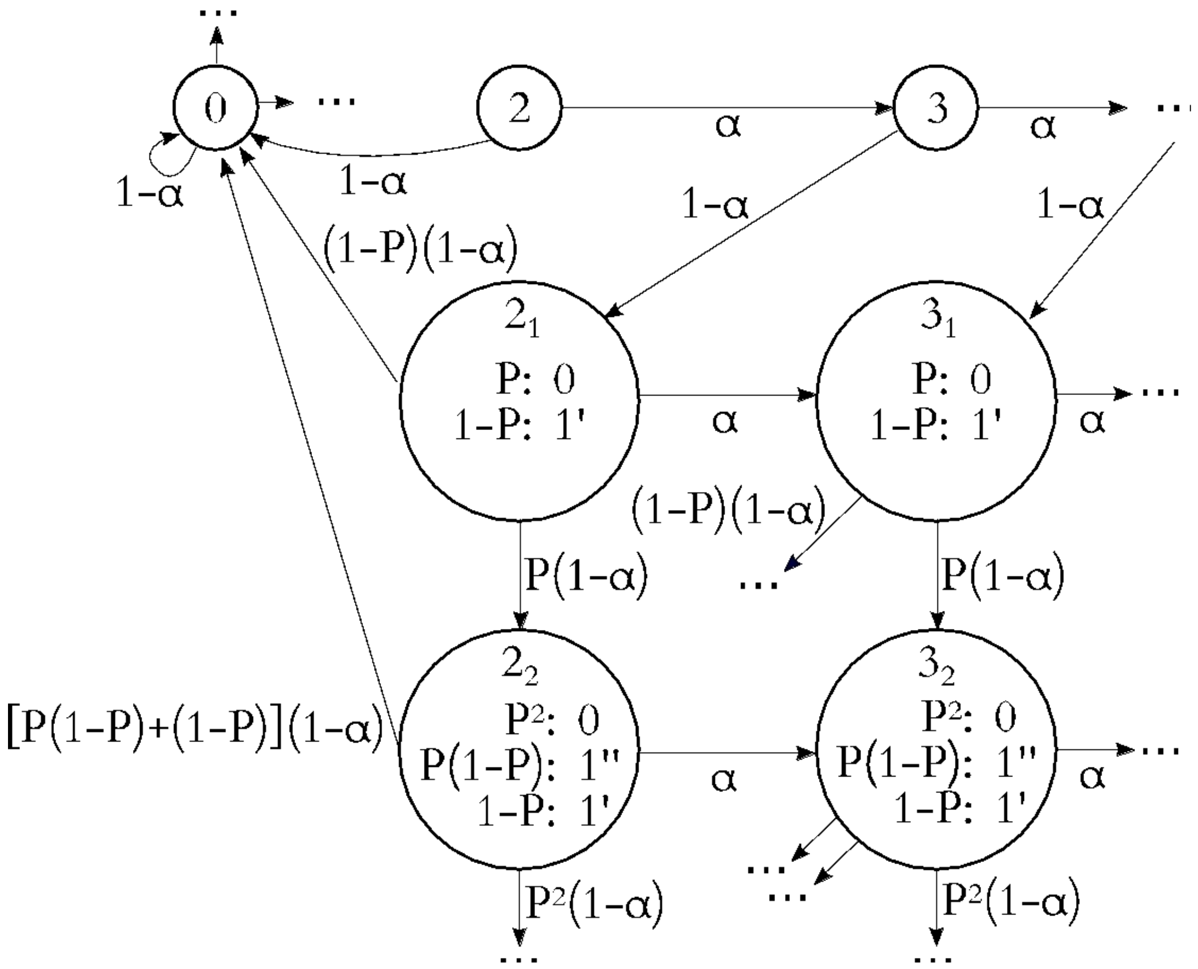
- Idea from Eyal et. al:
  - Instead of publishing, keep a block private
  - Other miners will perform wasteful computations



α : hashing power of adversary

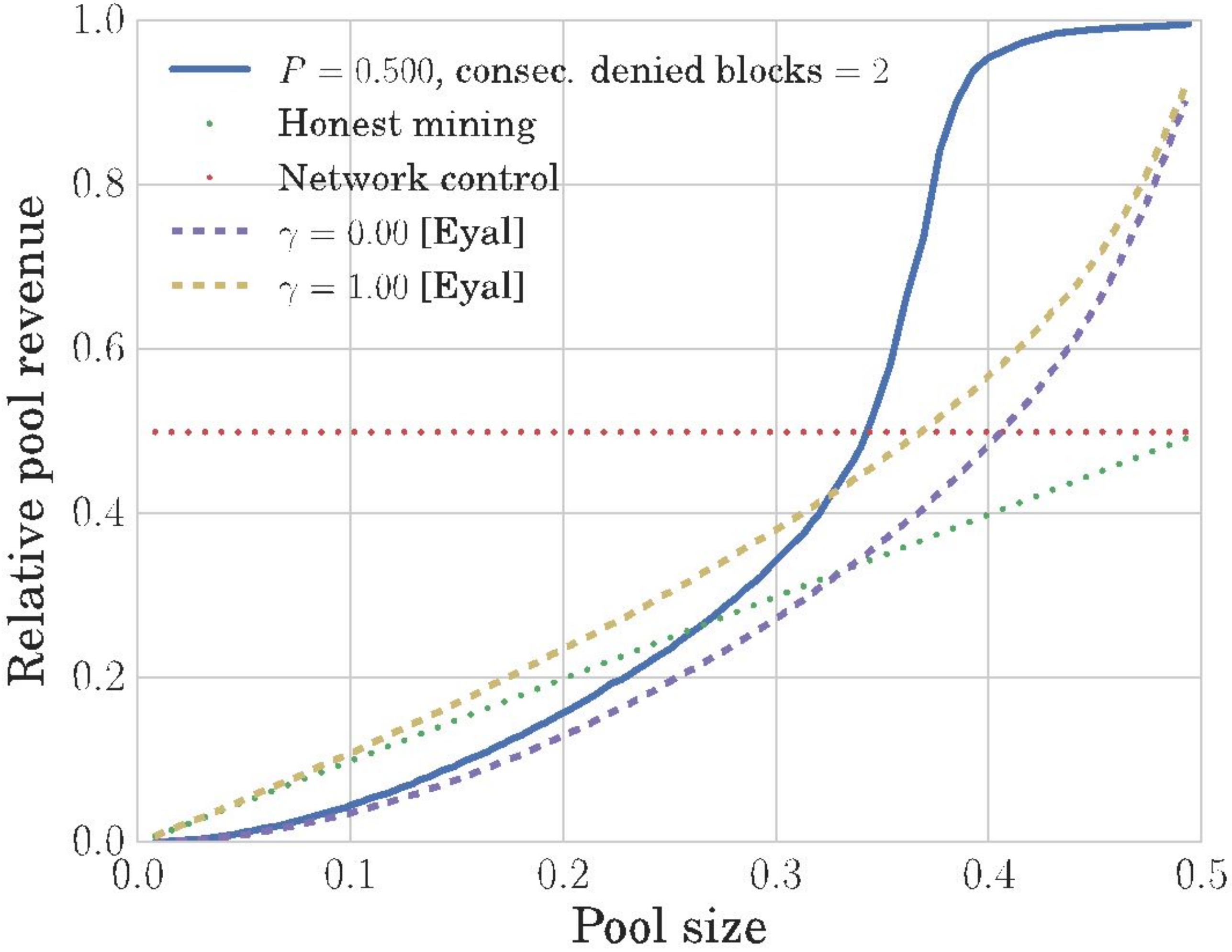
γ : propagation parameter

Increasing Mining Advantage with an Eclipse



P: probability to eclipse a block to a miner

Increasing Mining Advantage with an Eclipse



Smart Contract Layer Security

Smart Contract Layer

```
contract Wallet {
  uint balance = 10;

  function withdraw(){
    if(balance > 0)
msg.sender.call.value(balance)();
    balance = 0;
} }
```

Transfer $$$ to the caller

- Programs that handle money
  - Executed on a blockchain, written in a high-level language, compiled to VM code
- No patching after release
- What can go wrong?

The DAO attack

Funds Stolen From the DAO One Year Ago Would be Worth $1.35bn Today

JP Buntinx June 18, 2017 Crypto, News

33

Etherdice is down for maintenance. We are having troubles with our smart contract and will probably need to invoke the fallback mechanism.

report

King of the Ether Throne

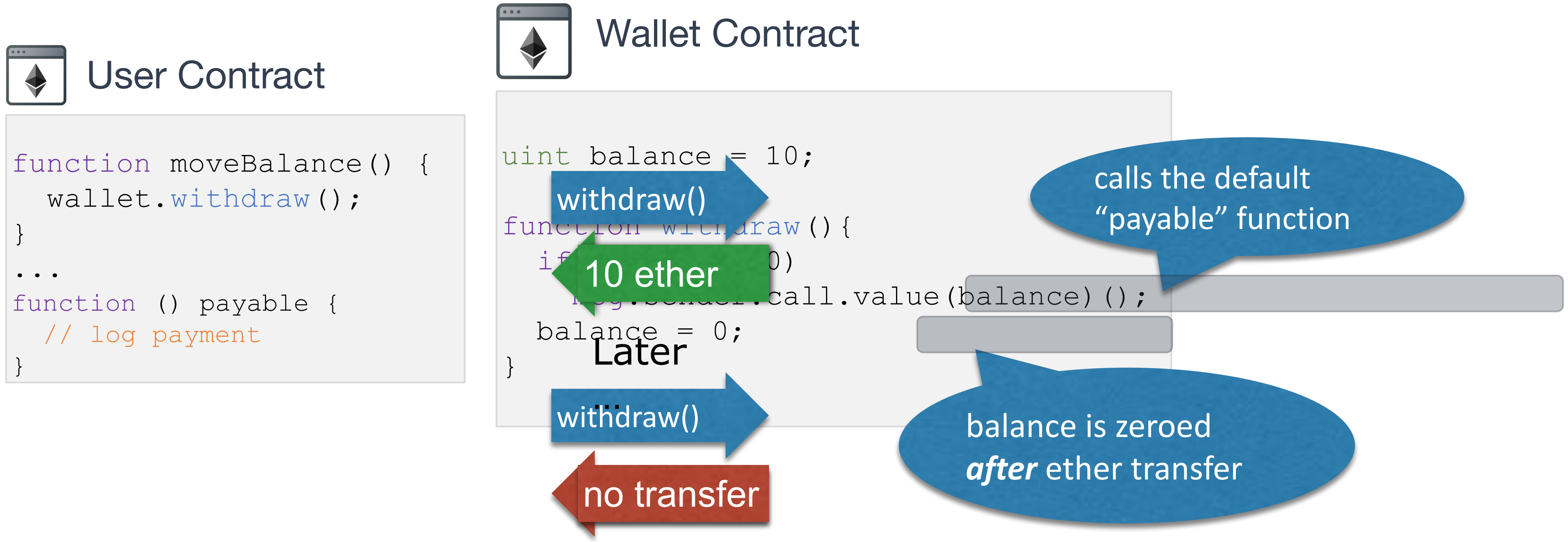
An Ethereum ĐApp (a "contract"), living on the blockchain, that will make you a King or Queen, might grant you riches, and will immortalize your name.

Important Notice

A SERIOUS ISSUE has been identified that can cause monarch compensation payments to not be sent.

DO NOT send payments to the contract previously referenced on this page, or attempt to claim the throne. Refunds will CERTAINLY NOT be made for any payments made after this issue was identified on 2016-02-07.

Security Bug #1: Reentrancy



Can the user contract withdraw more than 10 ether?

Security Bug #1: Reentrancy

User Contract

```
function moveBalance() {
  wallet.withdraw();
}
...
function () payable {
  wallet.withdraw();
}
```

Wallet Contract

```
uint balance = 10;

function withdraw(){
  if(balance > 0)
    msg.sender.call.value(balance)();
  balance = 0;
}
```

Calls withdraw() before balance is set to 0

balance is zeroed *after* ether transfer

An adversary stole 3.6M Ether !

Security Bug #2: Unprivileged write to storage

Wallet Contract

```
address owner = ...;

function initWallet(address _owner) {
  owner = _owner;
}

function withdraw(uint amount) {
  if (msg.sender == owner) {
    owner.send(amount);
  }
}
```

Any user may change the wallet's owner

Only owner can send ether

An attacker used a similar bug to ***steal $32M***

Smart Contract Bug Exercise 1

```
contract Example {

    address public owner;
    string private mySecret;

    constructor {
        owner = msg.sender;
    }

    function setSecret(string _secret) public {
        require(msg.sender == owner);
        mySecret = _secret;
    }

    function getSecret() public returns (string) {
        require(msg.sender == owner);
        return mySecret;
    }
}
```

Any variable is readable on the public Ethereum blockchain. Declaring a variable private only restricts the automatic creation of getter for that variable, but does not hide it.

Hint: who would be able to read mySecret?

Smart Contract Bug Exercise 2

```
contract Vulnerable {

    mapping(address => bool) authorized;
    mapping(address => uint) balances;

    function refund(uint amount) public {
        require(authorized[msg.sender]);
        require(amount <= balances[msg.sender]);

        msg.sender.call.value(amount)("");
        balances[msg.sender] -= amount;
    }
}
```

The code is vulnerable to a **reentrancy attack**. The balance of the *msg.sender* is only updated after a transfer is made. If the *msg.sender* is a contract and has a fallback function that calls into the contract again, the *msg.sender* can deplete the contract of the funds.

Hint: who can be msg.sender?

Smart Contract Bug Exercise 2

```
contract Vulnerable {
    … // vulnerable as the previous example
}


contract Exploit {

    Vulnerable v;

    function register(address contract) public {
        v = Vulnerable(contract);
    }

    function exploit() public {
        // your code here
    }

    // your code here
}
```

Hint: check the previous example

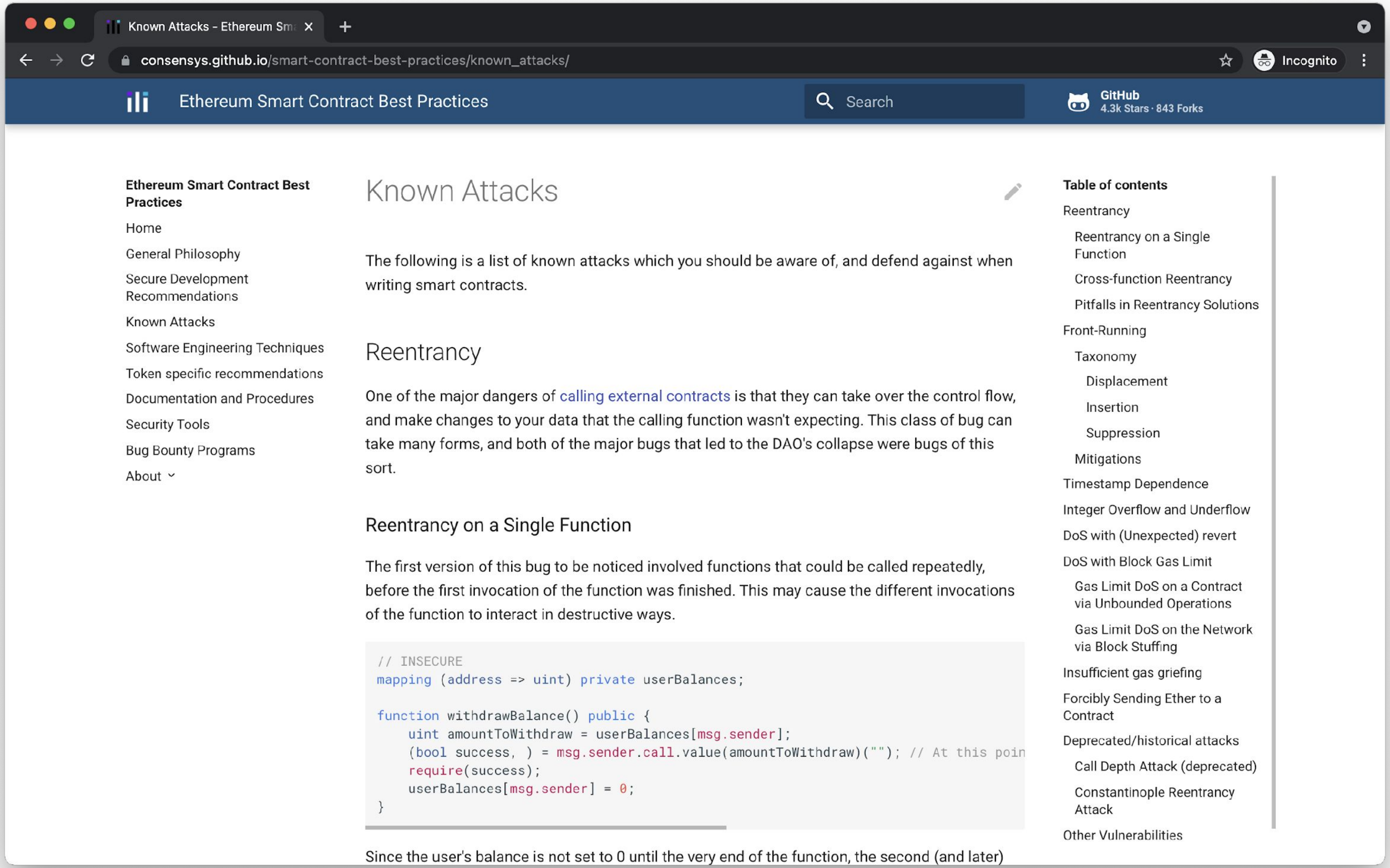
Smart Contract Bug Exercise 2 - Solution

```
contract Vulnerable {
    … // vulnerable as the previous example
}


contract Exploit {

    Vulnerable v;

    function register(address contract) public {
        v = Vulnerable(contract);
    }

    function exploit() public {
        v.refund(1);
    }

    function () public {
        v.refund(1);
    }
}
```

More smart contract security bugs

- Unexpected ether flows
- Insecure coding, such as unprivileged writes *(e.g., Multisig Parity bug)*
- Use of unsafe inputs (e.g., reflection, hashing, …)
- Reentrant method calls *(e.g., DAO bug)*

More smart contract security bugs



Known Attacks - Ethereum Sm

consensys.github.io/smart-contract-best-practices/known_attacks/

Incognito

Ethereum Smart Contract Best Practices

Search

GitHub
4.3k Stars · 843 Forks

Ethereum Smart Contract Best Practices

- Home
- General Philosophy
- Secure Development Recommendations
- Known Attacks
- Software Engineering Techniques
- Token specific recommendations
- Documentation and Procedures
- Security Tools
- Bug Bounty Programs
- About

Known Attacks

The following is a list of known attacks which you should be aware of, and defend against when writing smart contracts.

Reentrancy

One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. This class of bug can take many forms, and both of the major bugs that led to the DAO's collapse were bugs of this sort.

Reentrancy on a Single Function

The first version of this bug to be noticed involved functions that could be called repeatedly, before the first invocation of the function was finished. This may cause the different invocations of the function to interact in destructive ways.

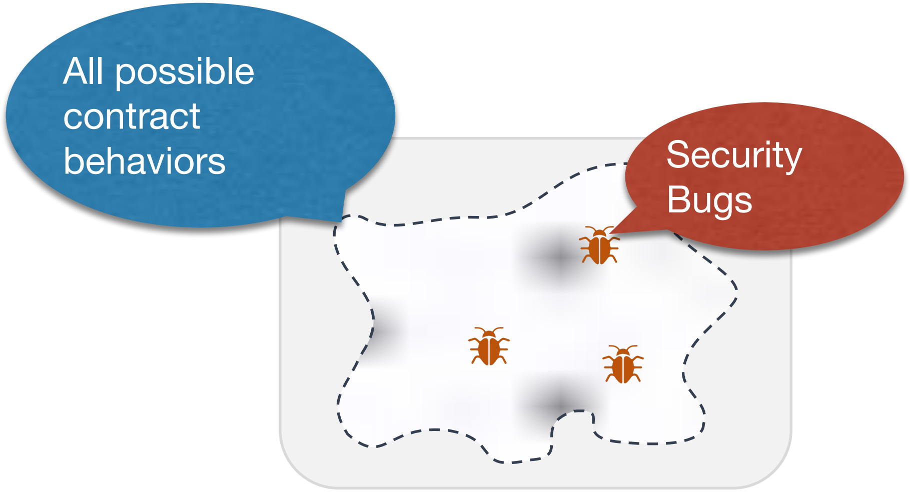
```
// INSECURE
mapping (address => uint) private userBalances;

function withdrawBalance() public {
    uint amountToWithdraw = userBalances[msg.sender];
    (bool success, ) = msg.sender.call.value(amountToWithdraw)(""); // At this poin
    require(success);
    userBalances[msg.sender] = 0;
}
```

Since the user's balance is not set to 0 until the very end of the function, the second (and later)

Table of contents

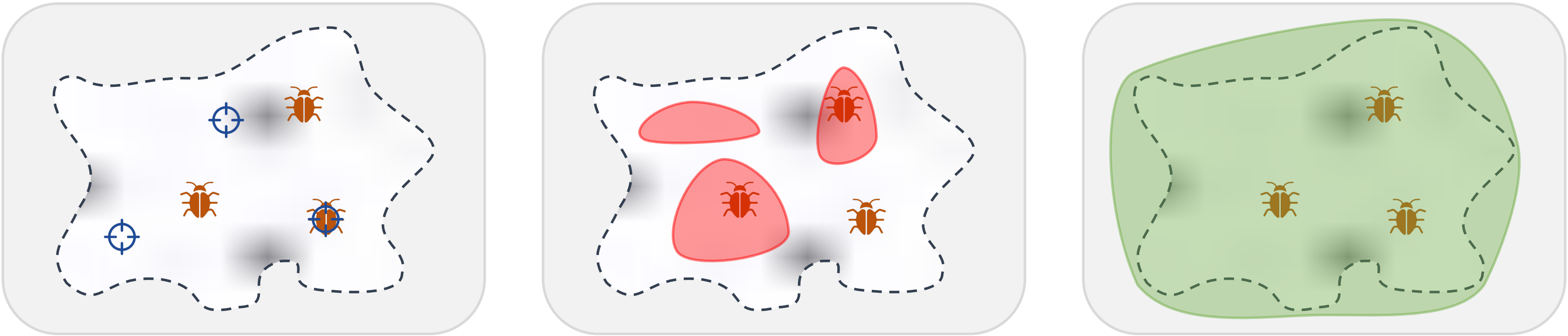
https://consensys.github.io/smart-contract-best-practices/known_attacks/

Automated security analysis



Problem: Cannot enumerate all possible contract behaviors…

Automated security analysis – Existing solutions



Testing	Dynamic analysis Symbolic execution	Static analysis Formal verification
Easy to implement, but very limited guarantees	Better than testing, but can still miss vulnerabilities	Strong guarantees, but many false positives

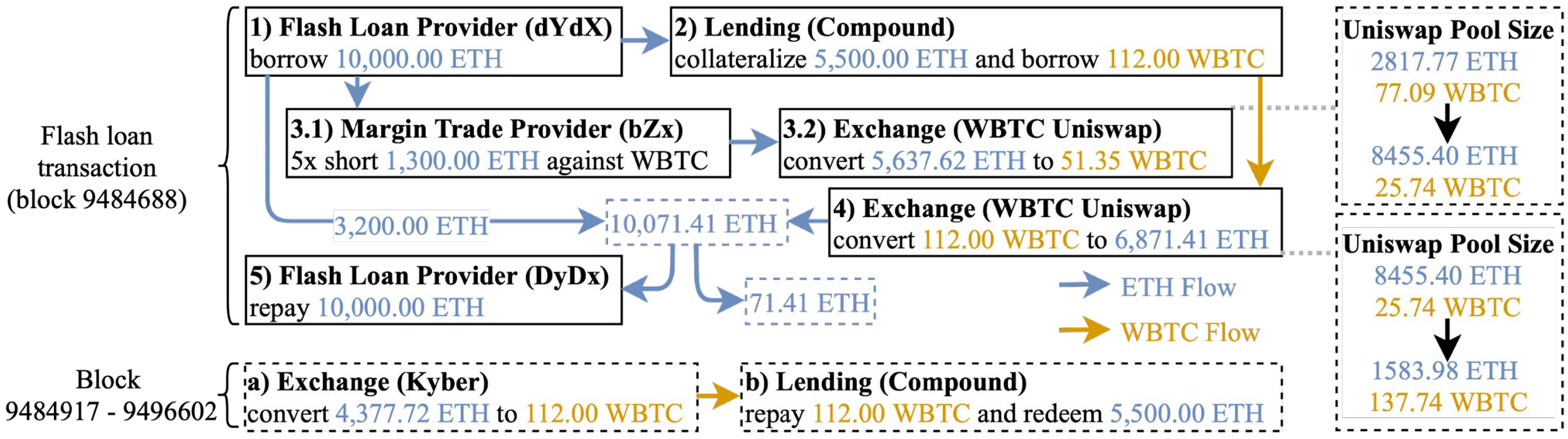
DeFi Flash Loan „Attacks“

https://defi-learning.org

Flash Loan Attacks

+

bZx - Pump and Arbitrage Attack – February 2020



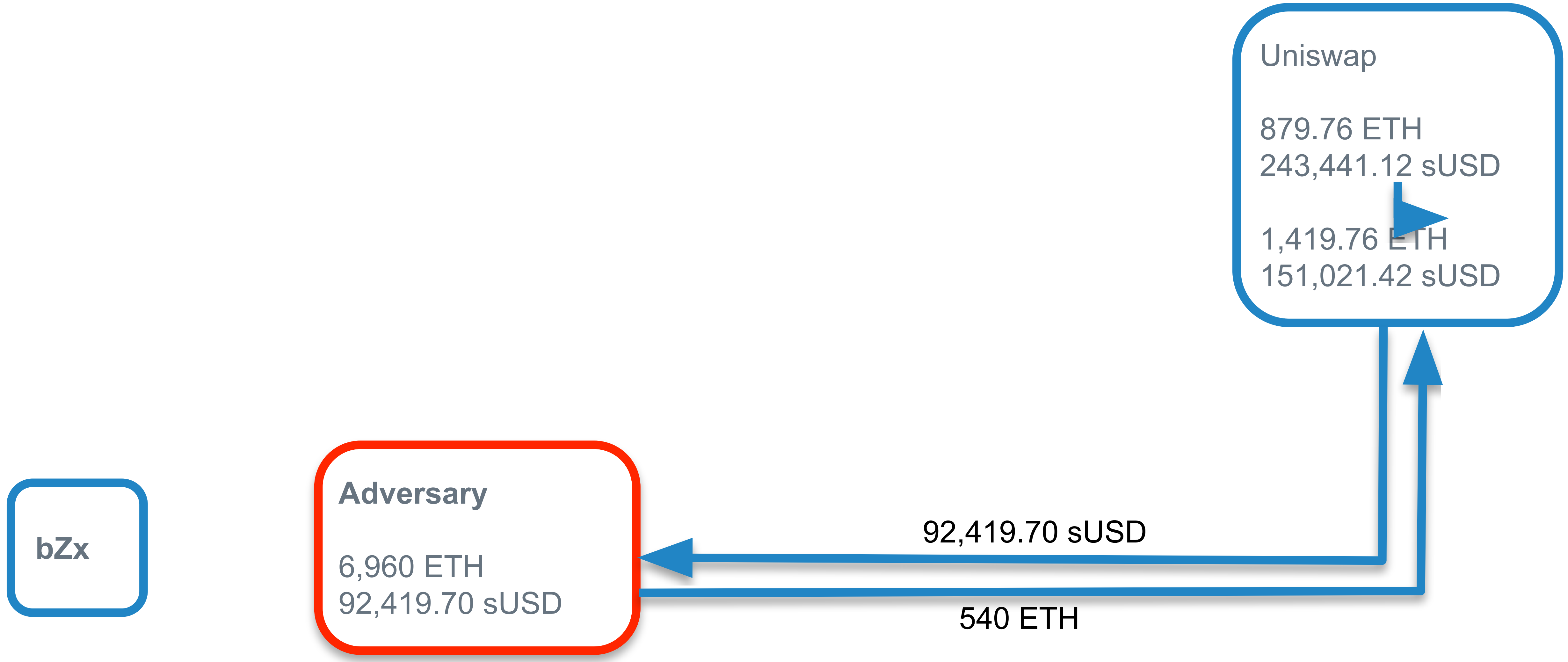
Input: 130 USD gas

Output: 350,000 USD

Optimal: 830,000 USD

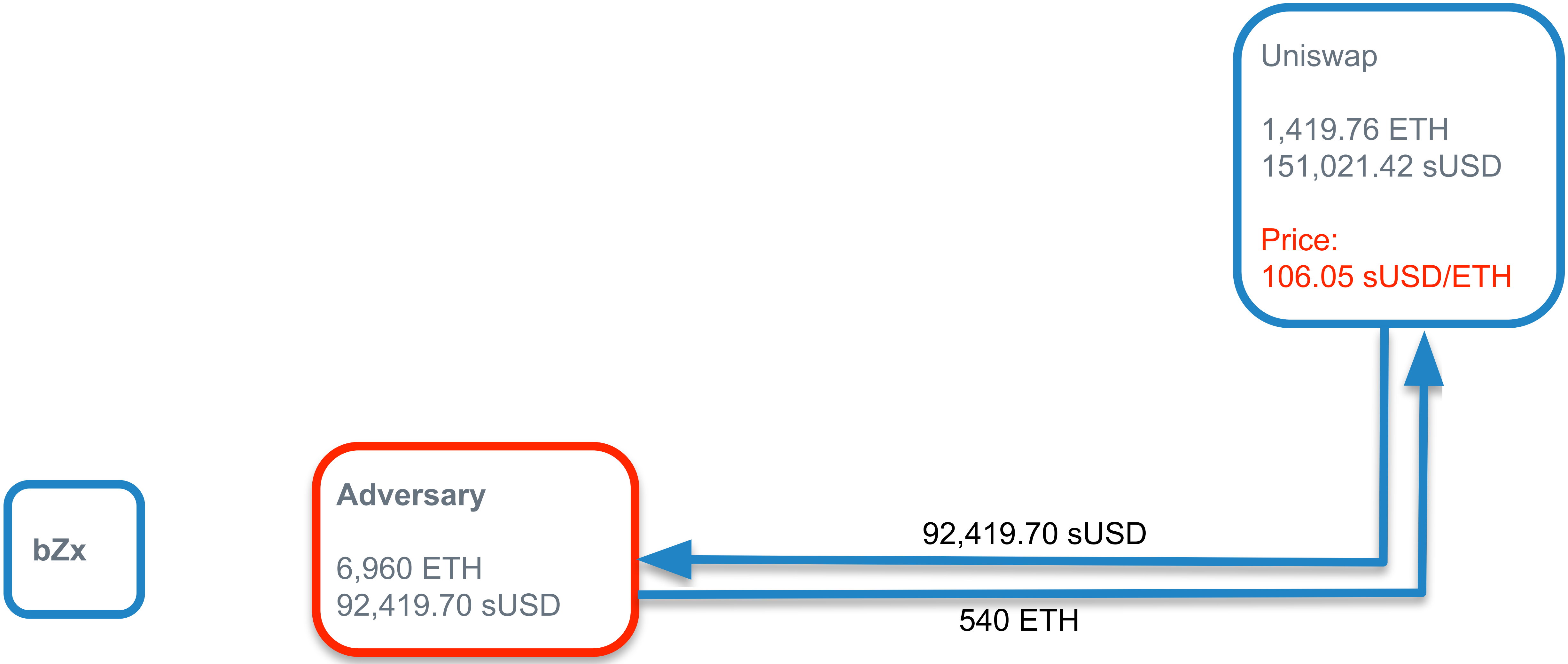
bZx – Oracle manipulation – February 2020

bZx – Oracle manipulation – February 2020



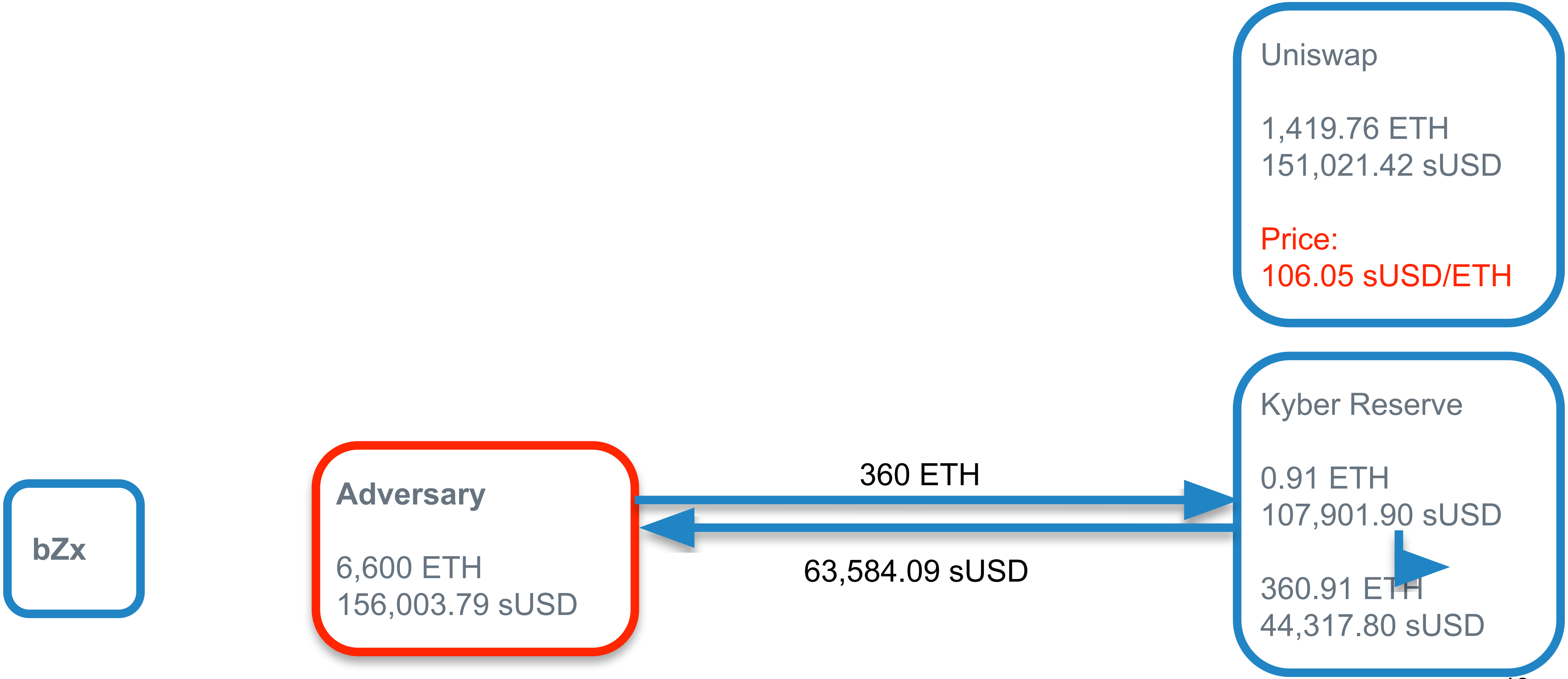
Exchange rate: (step 2) 171.15 sUSD/ETH

bZx – Oracle manipulation – February 2020



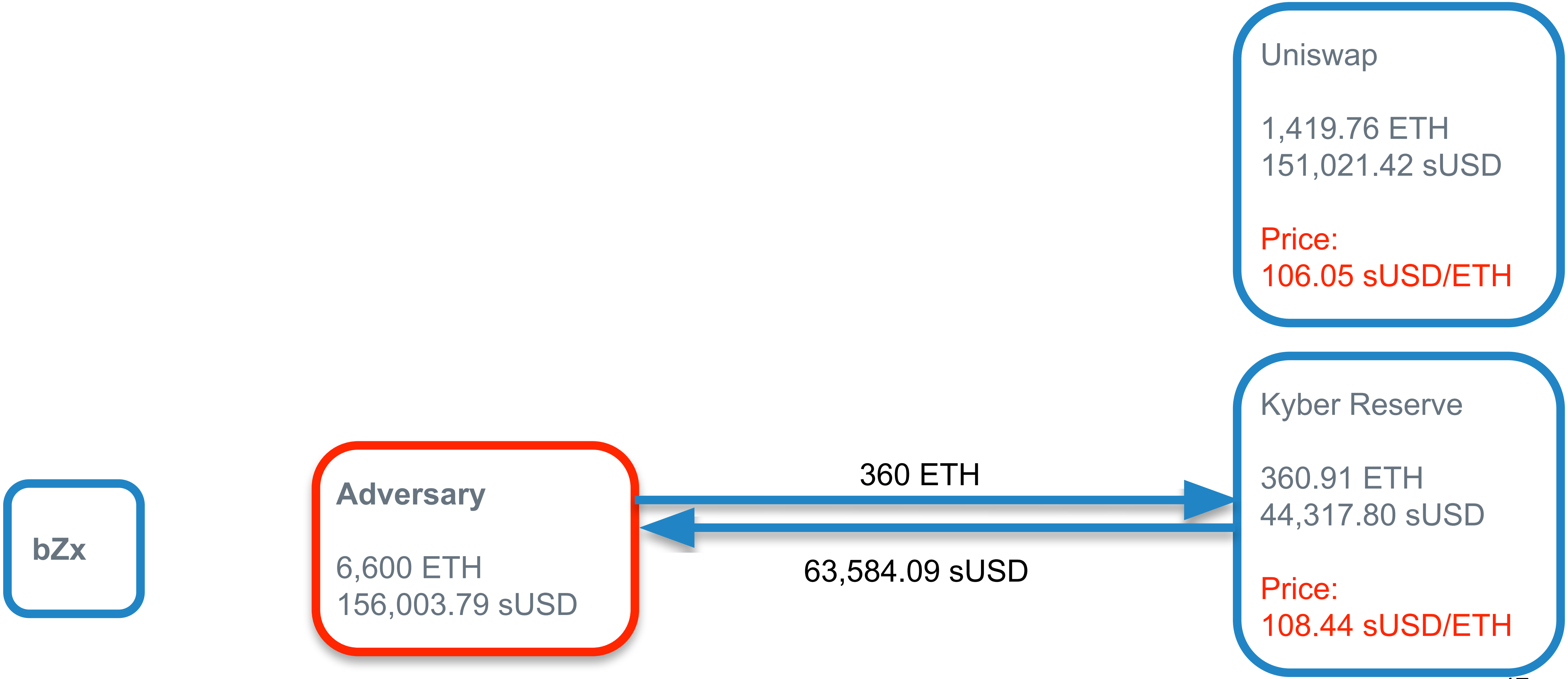
Exchange rate: (step 2) 171.15 sUSD/ETH

bZx – Oracle manipulation – February 2020



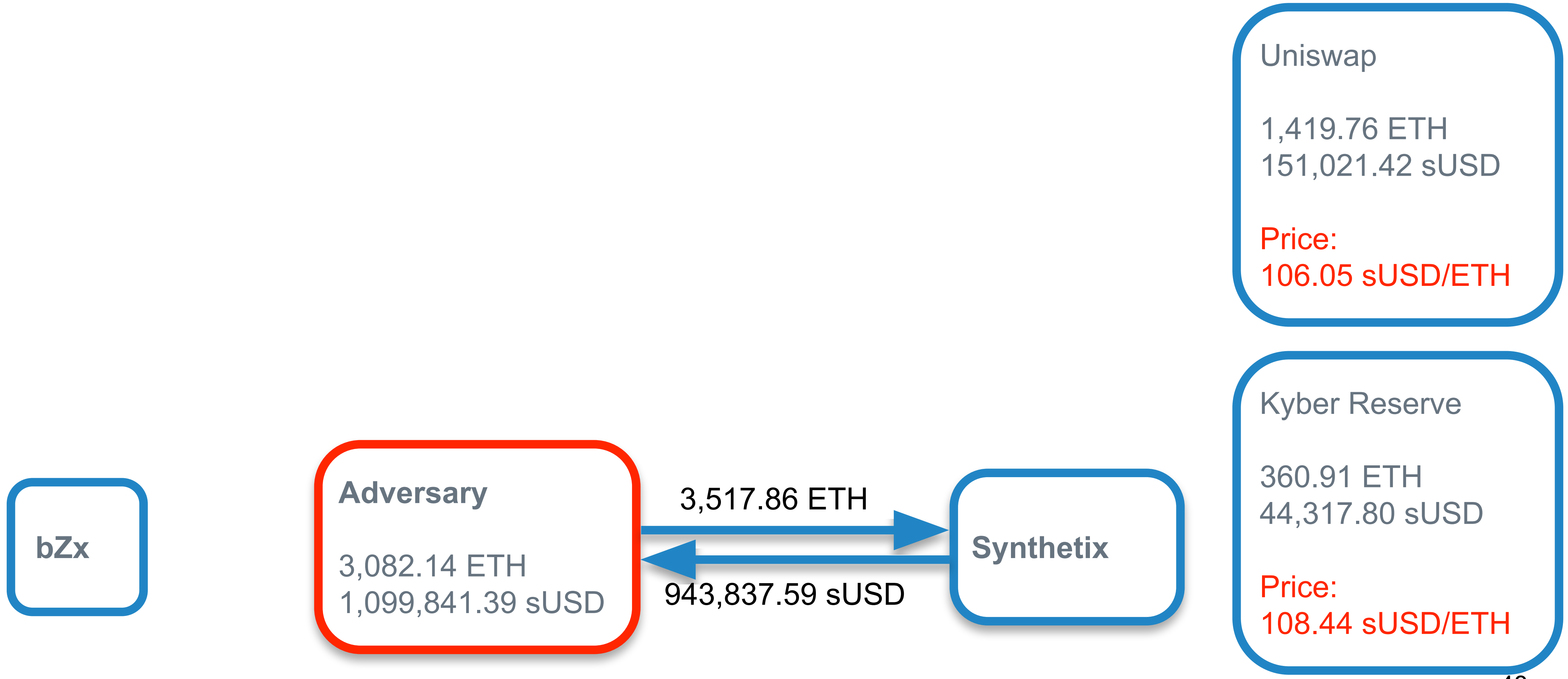
Exchange rate: (step 2) **171.15 sUSD/ETH**; (step 3) **176.62 sUSD/ETH**

bZx – Oracle manipulation – February 2020



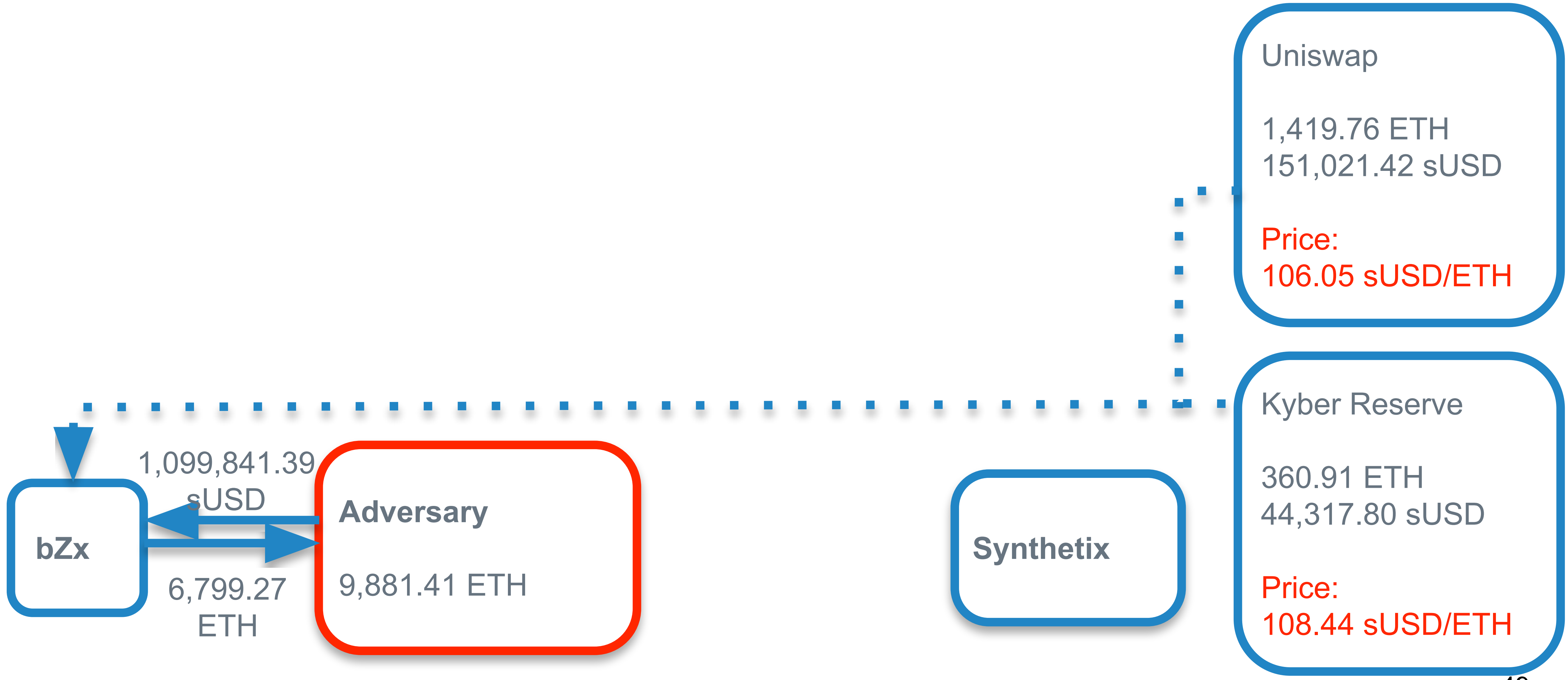
Exchange rate: (step 2) **171.15 sUSD/ETH**; (step 3) **176.62 sUSD/ETH**

bZx – Oracle manipulation – February 2020



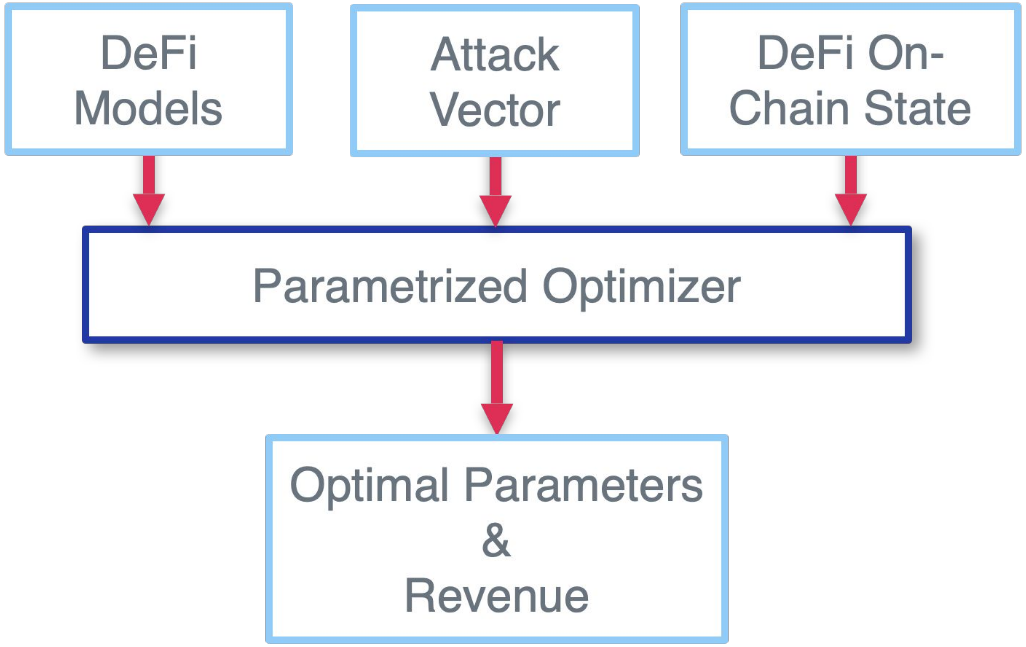
Exchange rate: (step 2) **171.15 sUSD/ETH**; (step 3) **176.62 sUSD/ETH;** (step 4) **268.30 sUSD/ETH**

bZx – Oracle manipulation – February 2020



Constrained Optimization Framework

- Formulate DeFi actions in models
  - Constant product AMM: $\Delta y = y - \frac{xy}{x+\Delta x}$
- Construct a constrained optimization problem based on the attack vector
  - Objective function: outcome profit
- Fetch the on-chain state that the



Optimizing the bZx attack 2

- Borrow X ETH (bZx flash loan)
  - Convert $p1$ ETH to $f1(p1)$ sUSD (Uniswap)
  - Convert $p2$ ETH to $f2(p2)$ sUSD (Kyber)
  - Deposit $p3$ ETH for $f3(p3)$ sUSD (Synthetix)
  - Collateralize z sUSD to borrow $g(z)$ ETH
  - $z=f1(p1)+f2(p2)+f3(p3)$
- Repay X ETH (bZx flash loan)

- Objective: $o=g(f1(p1)+f2(p2)+f3(p3))-X$
  - s.t. $p1+p2+p3<X$

Optimizing the bZx attack 2

- Sequential Least Squares Programming (SLSQP)
  - SciPy
- Ubuntu 18.04.2, 16 CPU cores, 32 GB RAM
- Validation by concrete execution
  - Execution on the real blockchain state

Sandwich Attacks

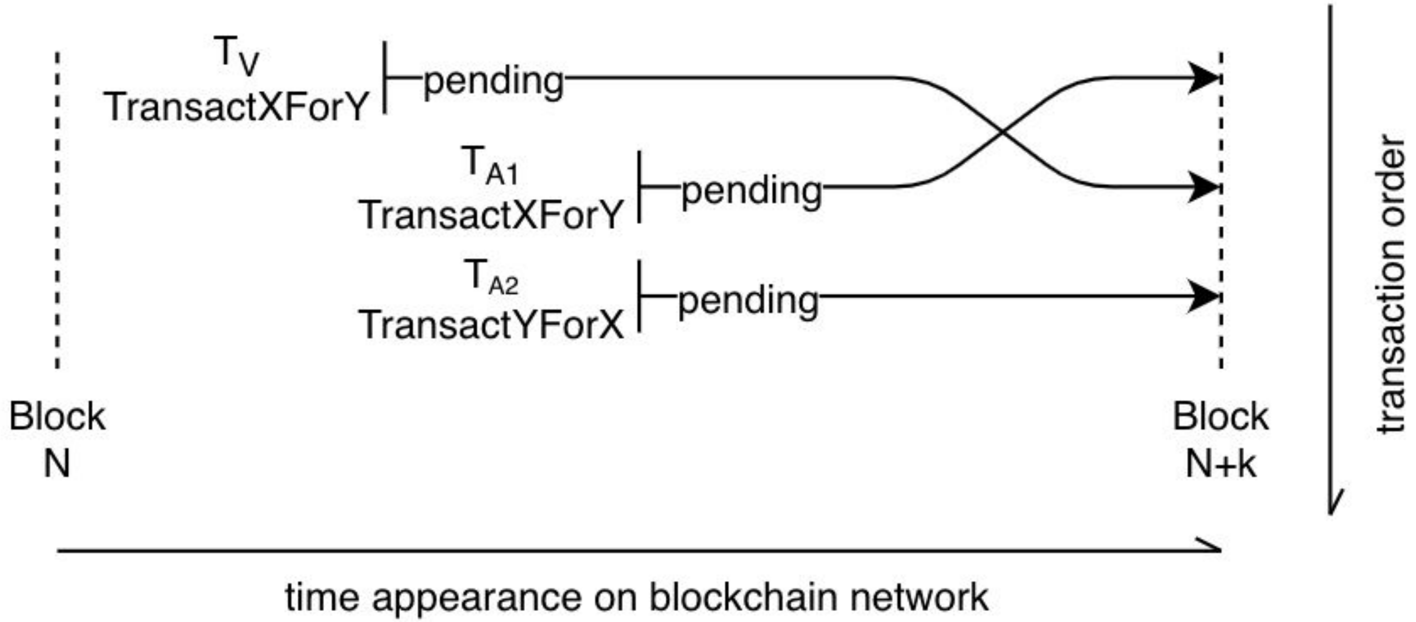
https://defi-learning.org

AMM – Automated Market Maker

$$x \times y = k$$

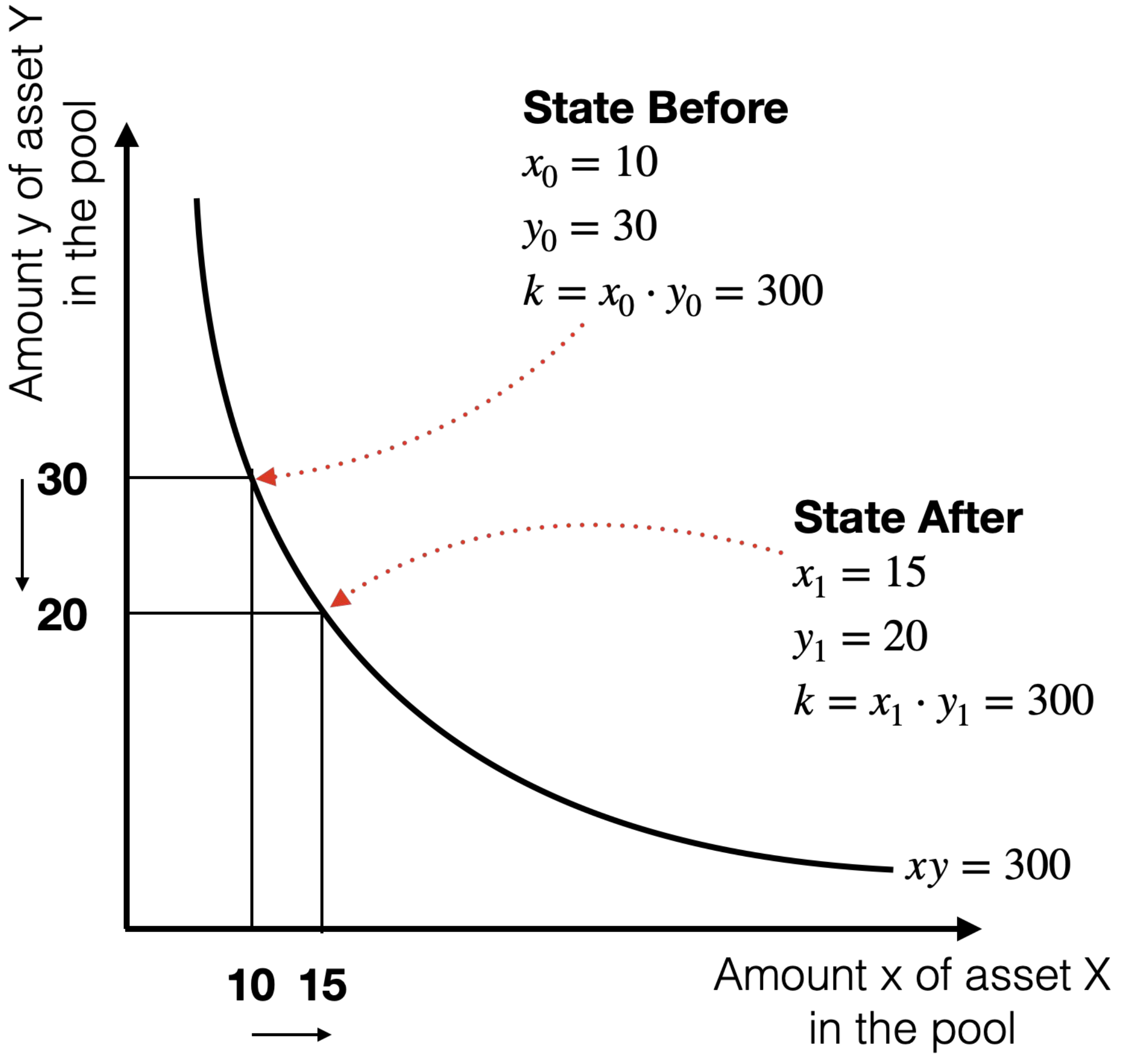
- x → Asset X quantity
- y → Asset Y quantity
- k → constant

Sandwich Attack

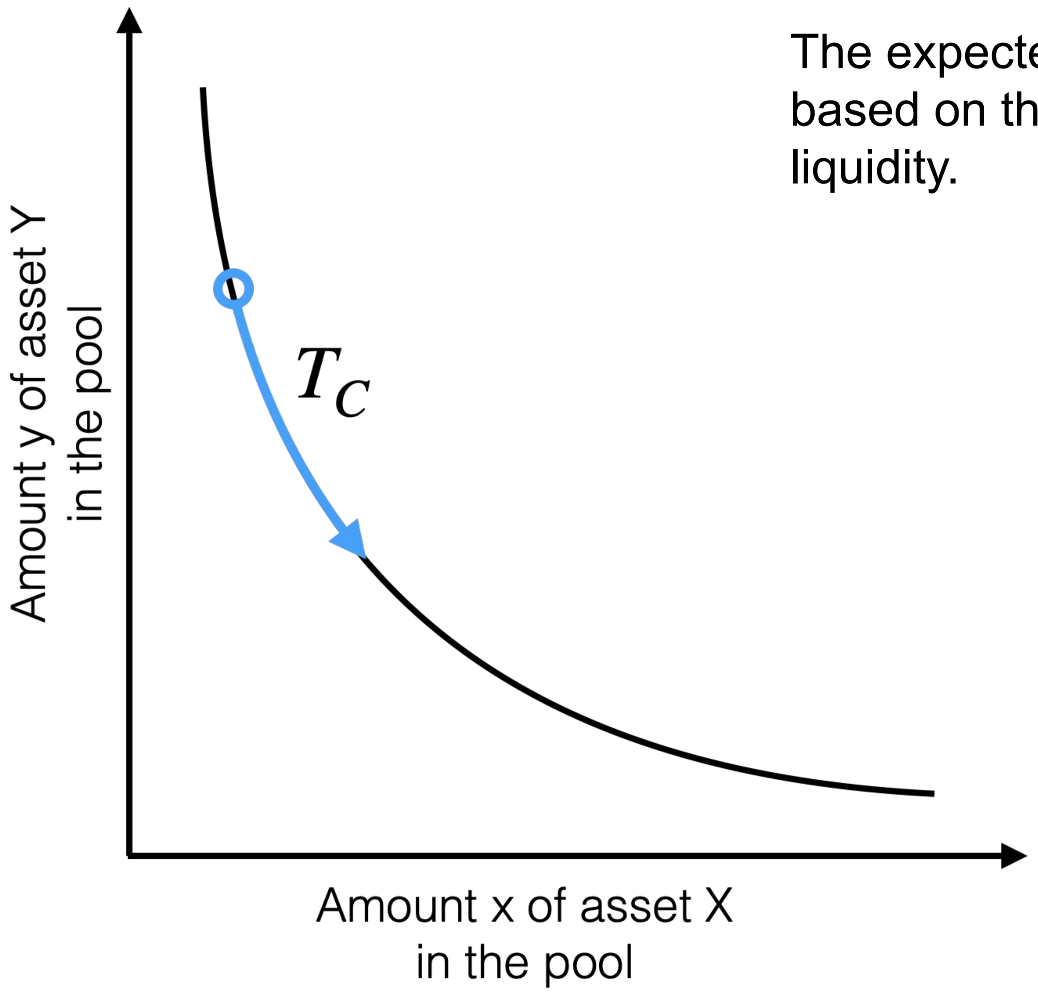


AMM – Constant product formula

AMM – Constant product formula

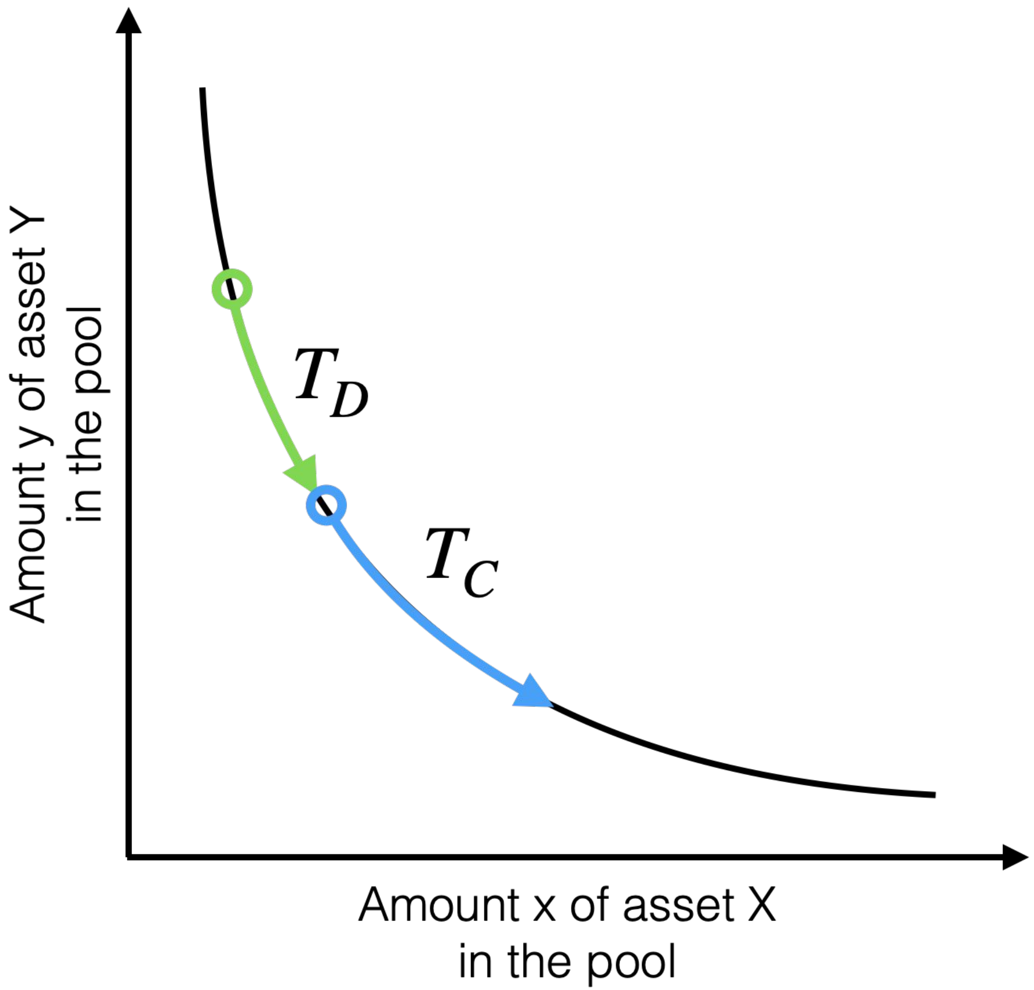


Expected Slippage

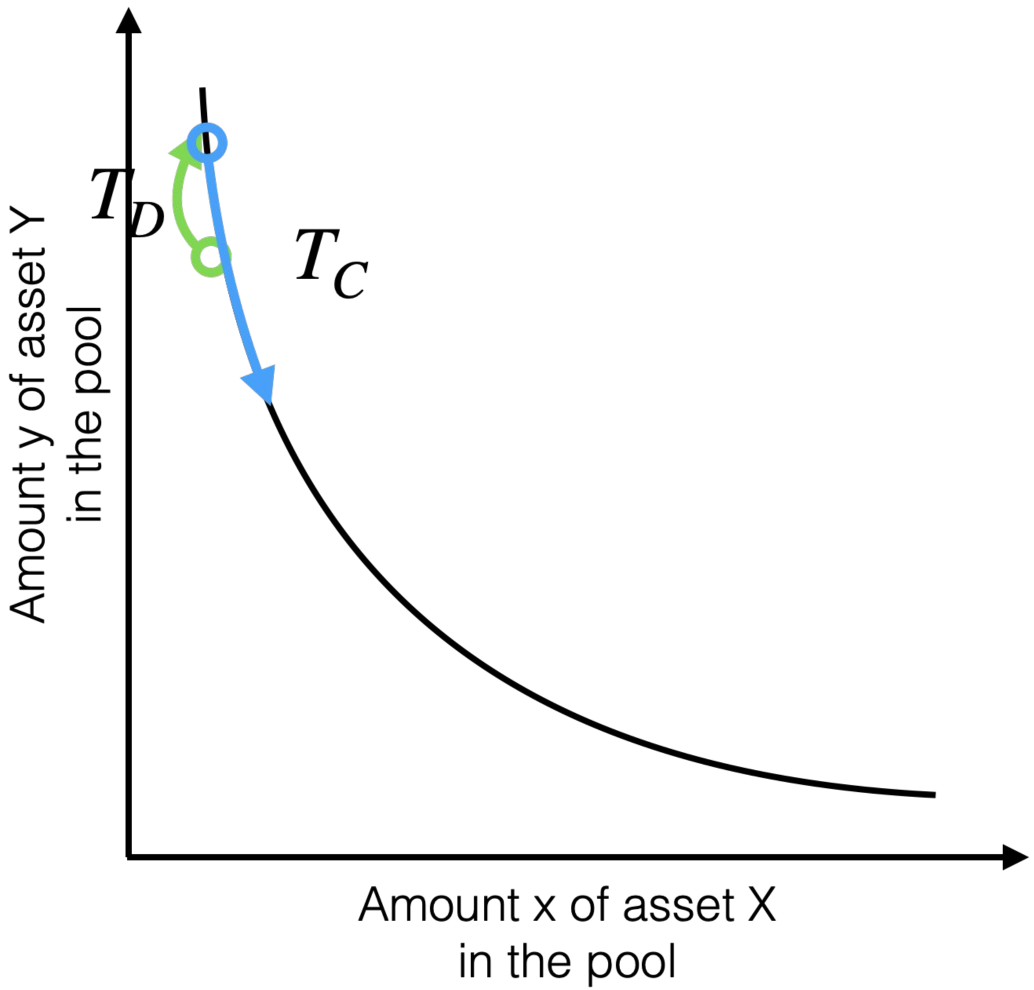


The expected increase or decrease in price based on the trading volume and available liquidity.

Unexpected Slippage -> Worse Execution Price

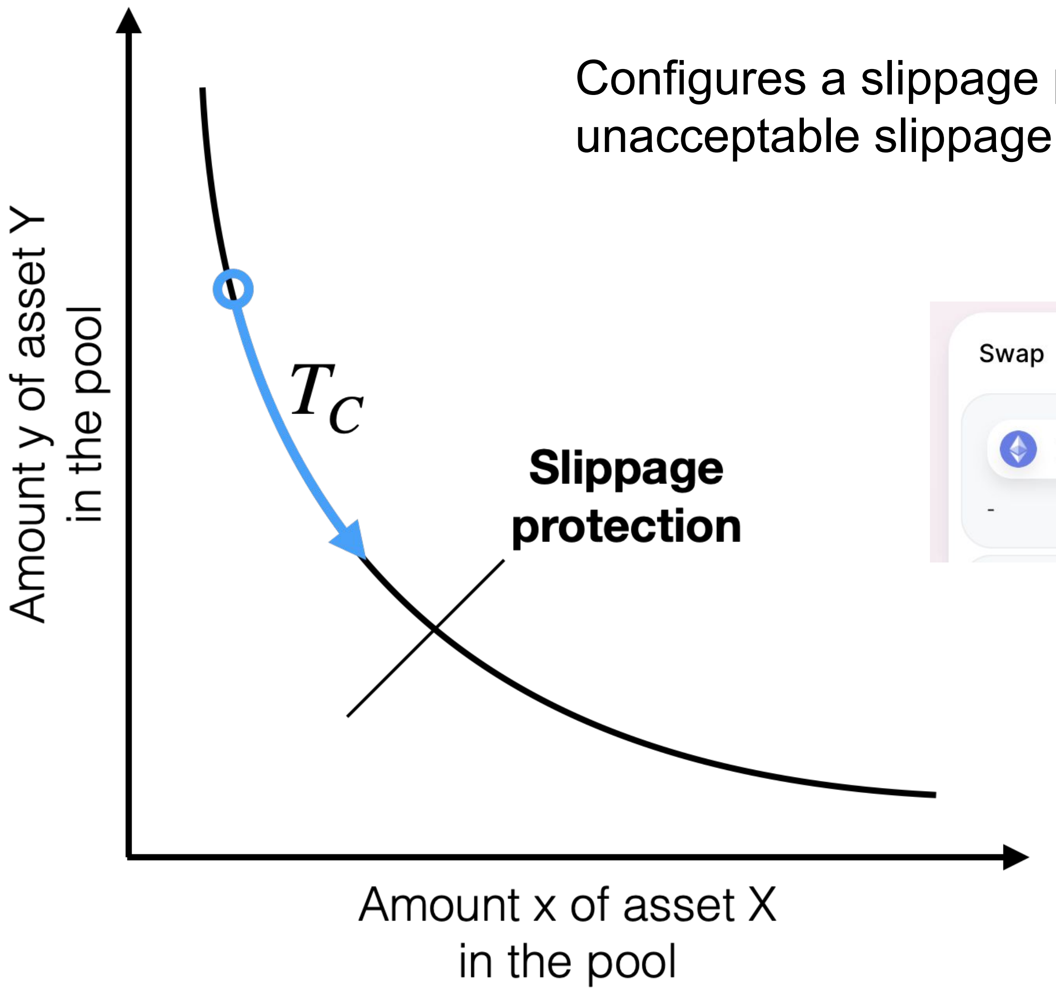


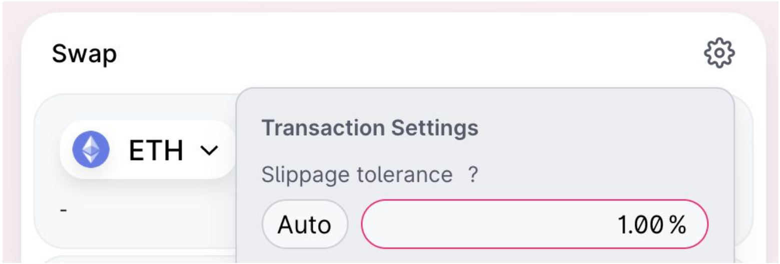
Unexpected Slippage -> Better Execution Price



Slippage Protection

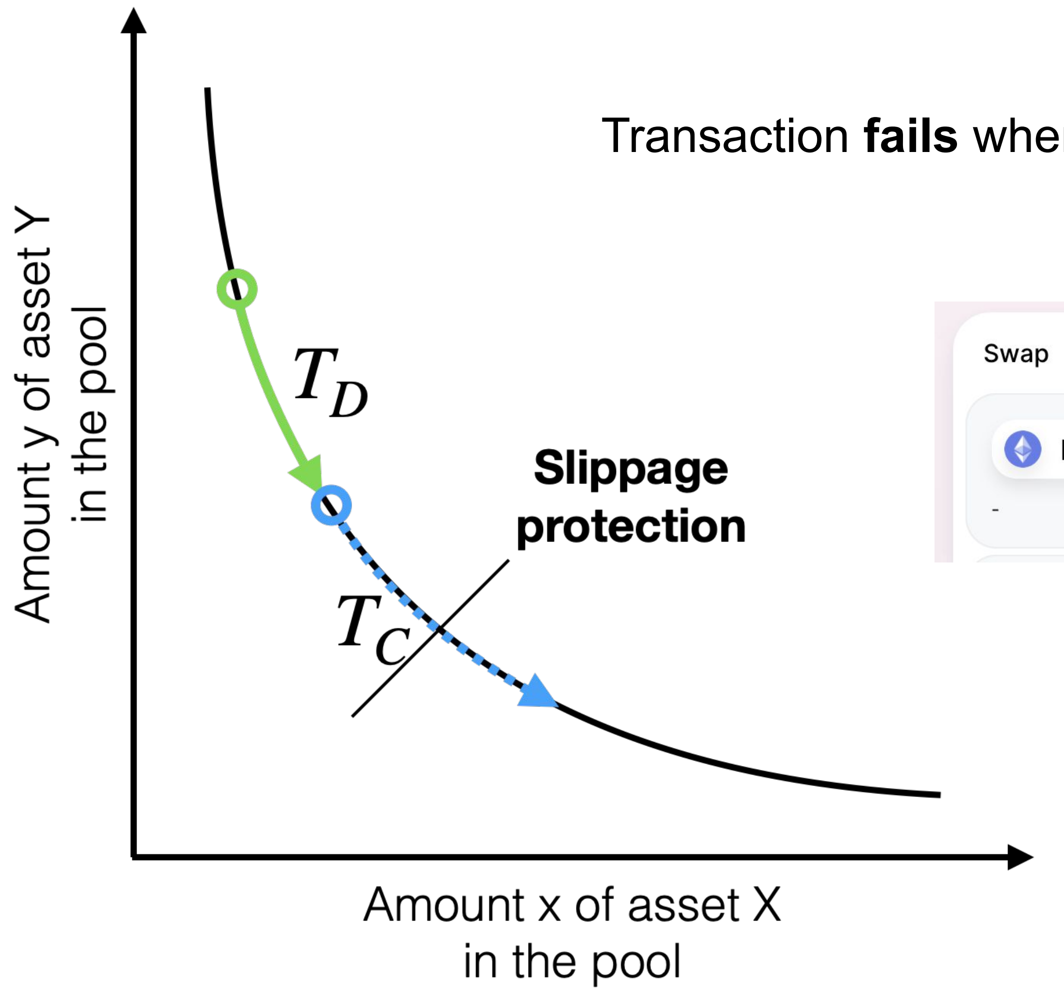
Configures a slippage protection threshold to prevent unacceptable slippage

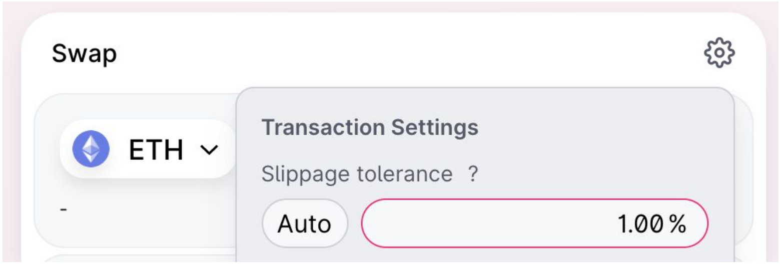




Slippage Protection

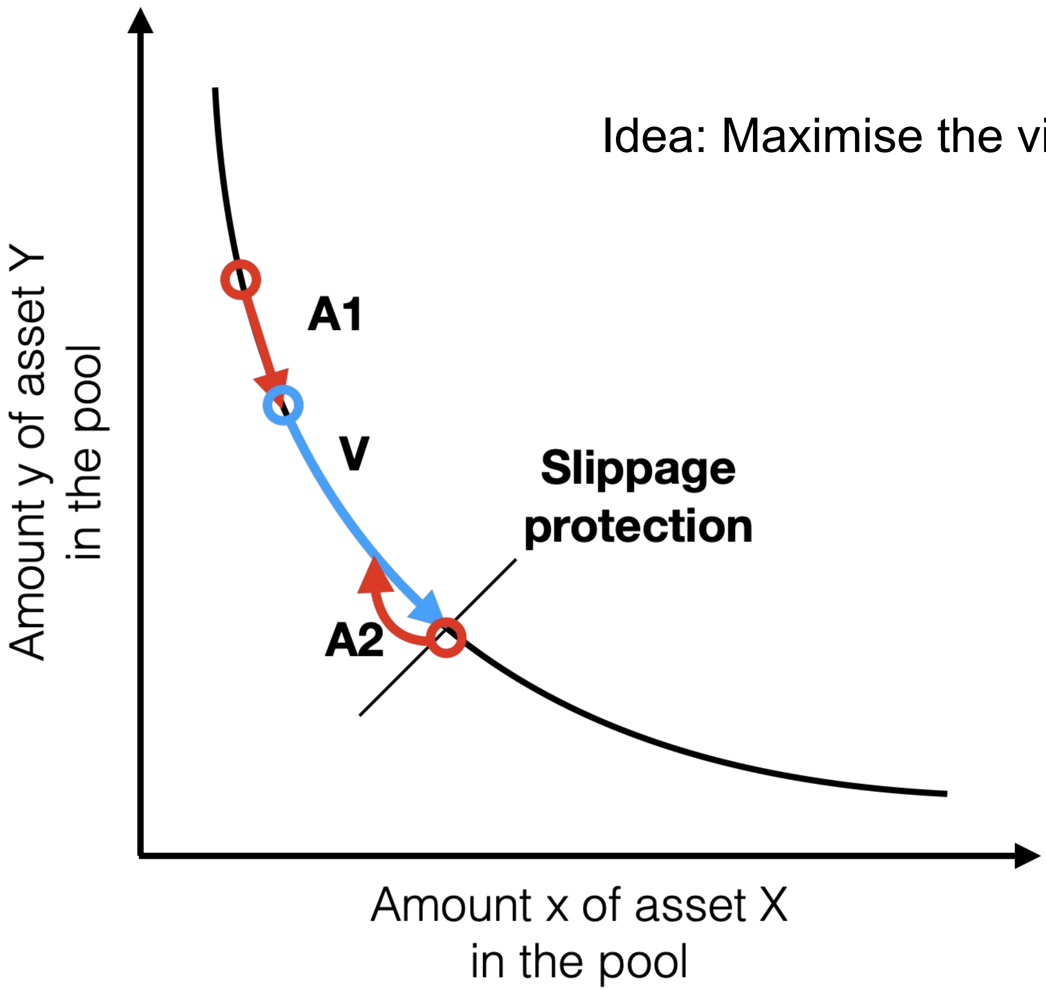
Transaction **fails** when crossing the slippage limit.



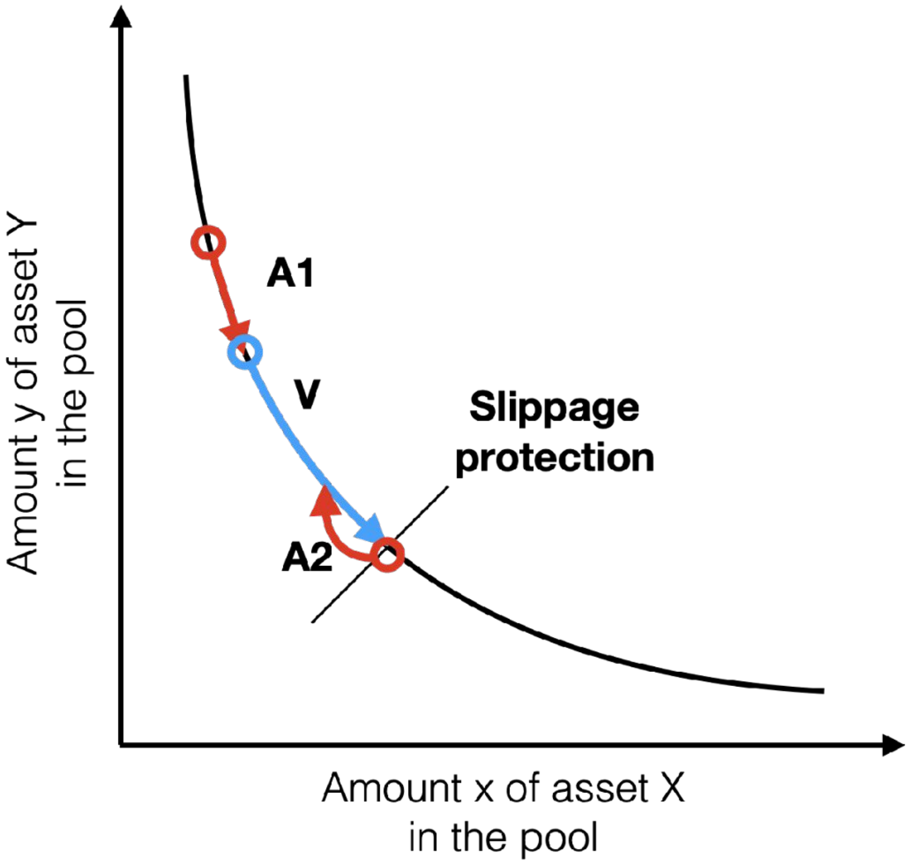


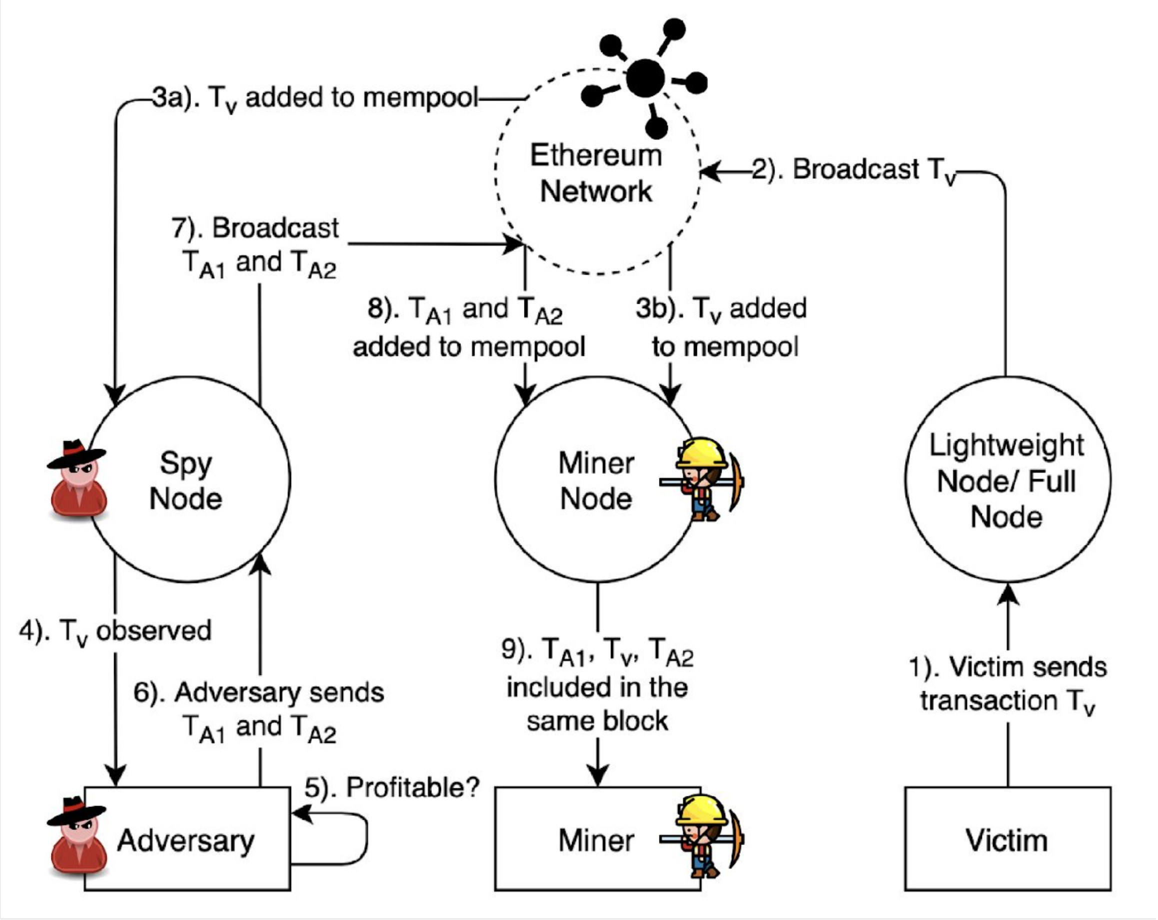
Sandwich Attack Against Taker

Idea: Maximise the victim's slippage

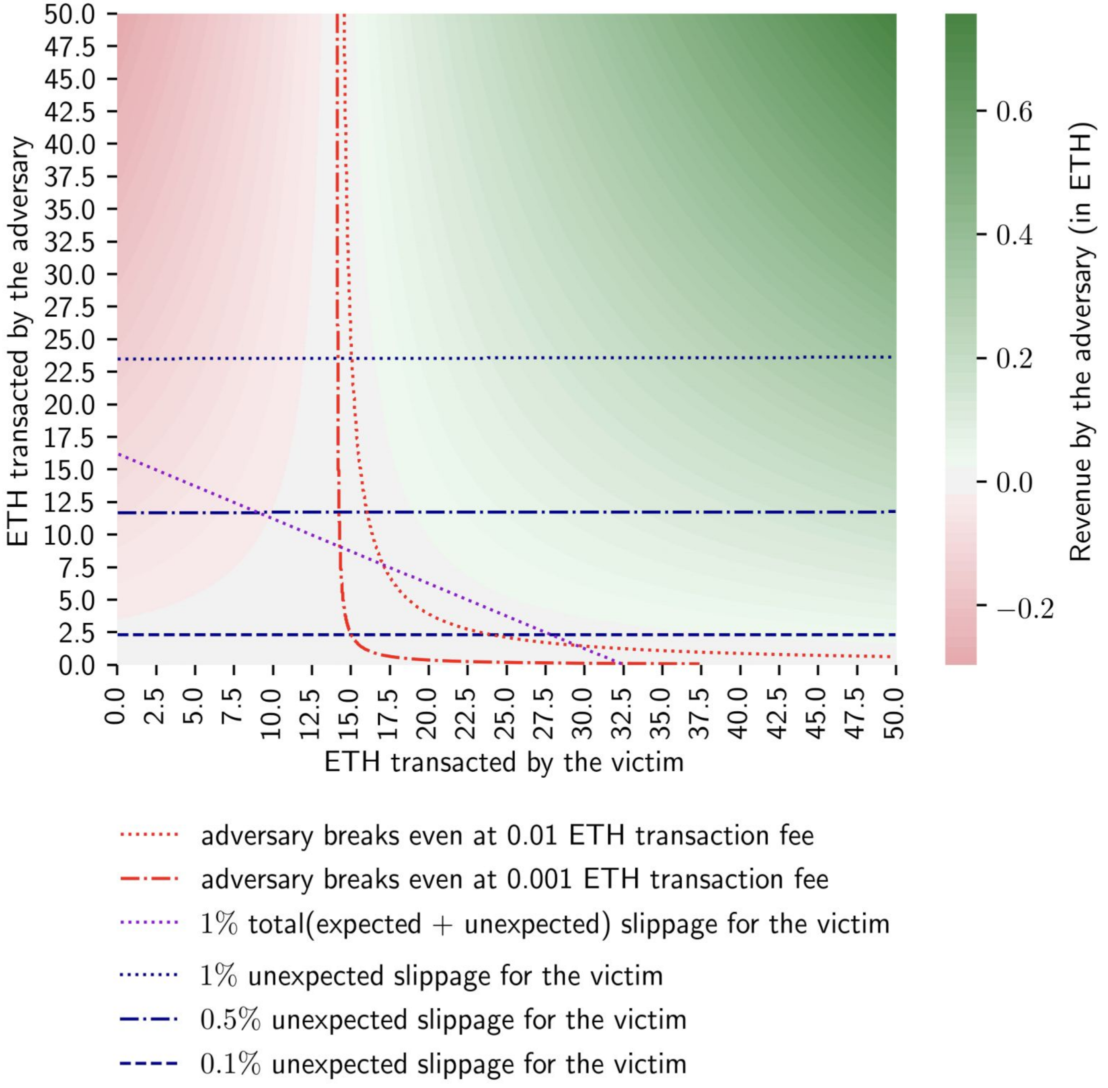


Network layer + DeFi protocol layer



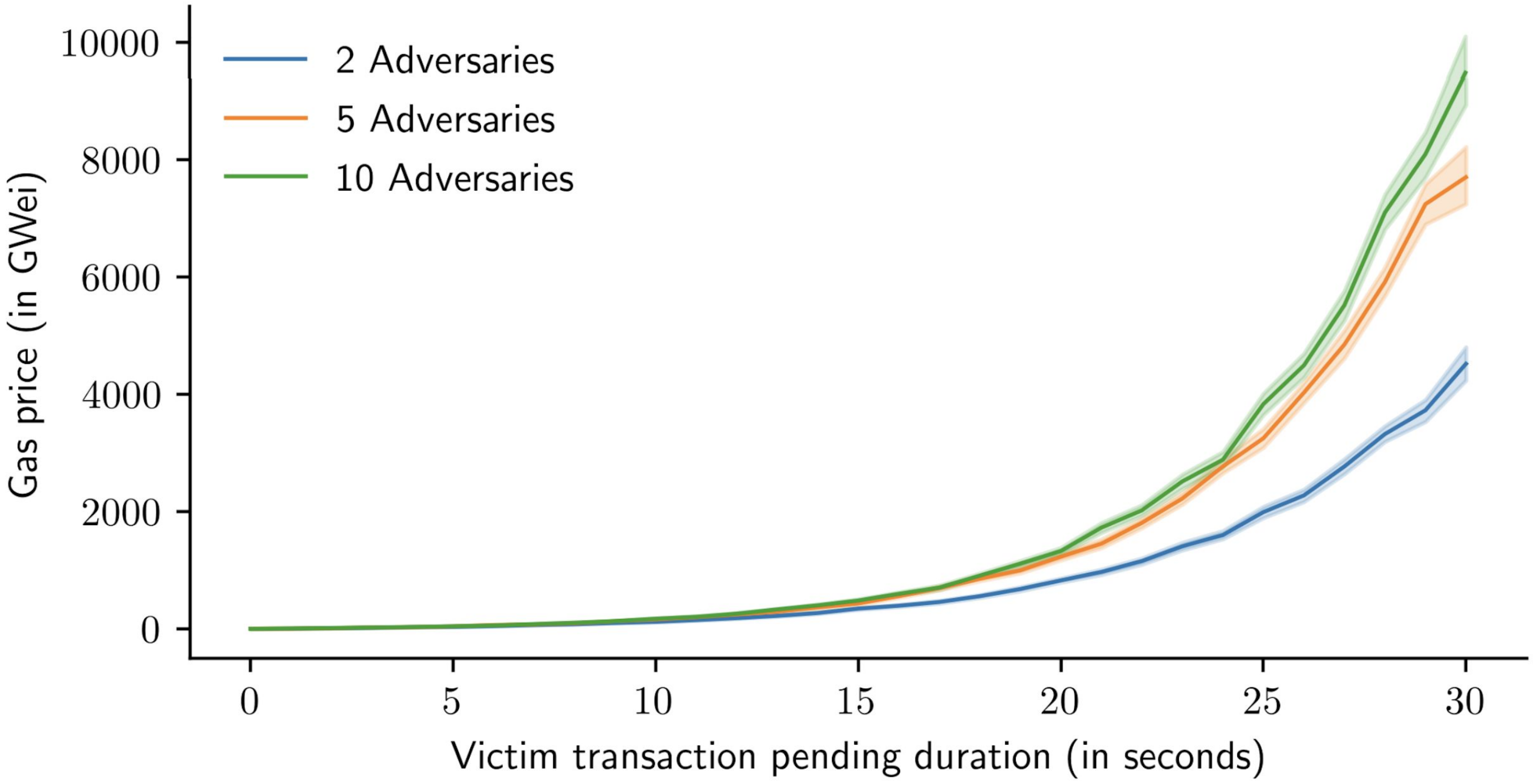


Sandwich attack profitability

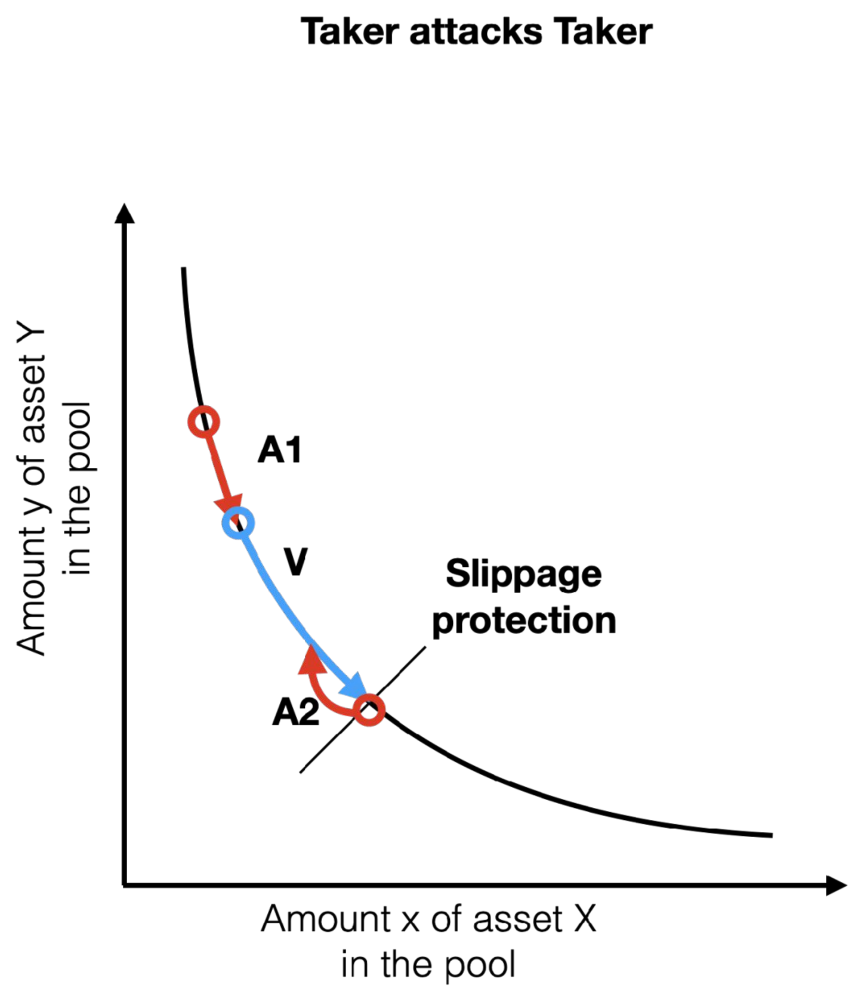


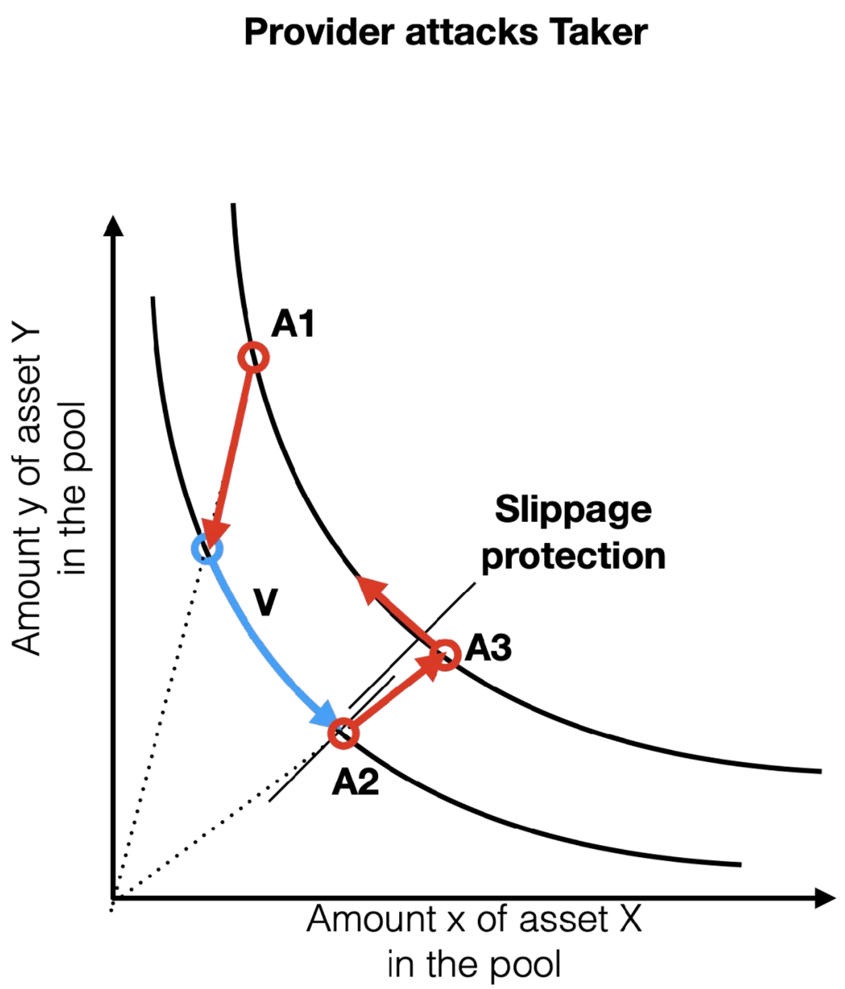
Multiple Adversaries

Break-even of the attacker becomes harder to attain



Advanced Sandwich Attack





Blockchain Extractable Value

What is Blockchain (or Miner) Extractable Value?

Price of collateral drops below health factor

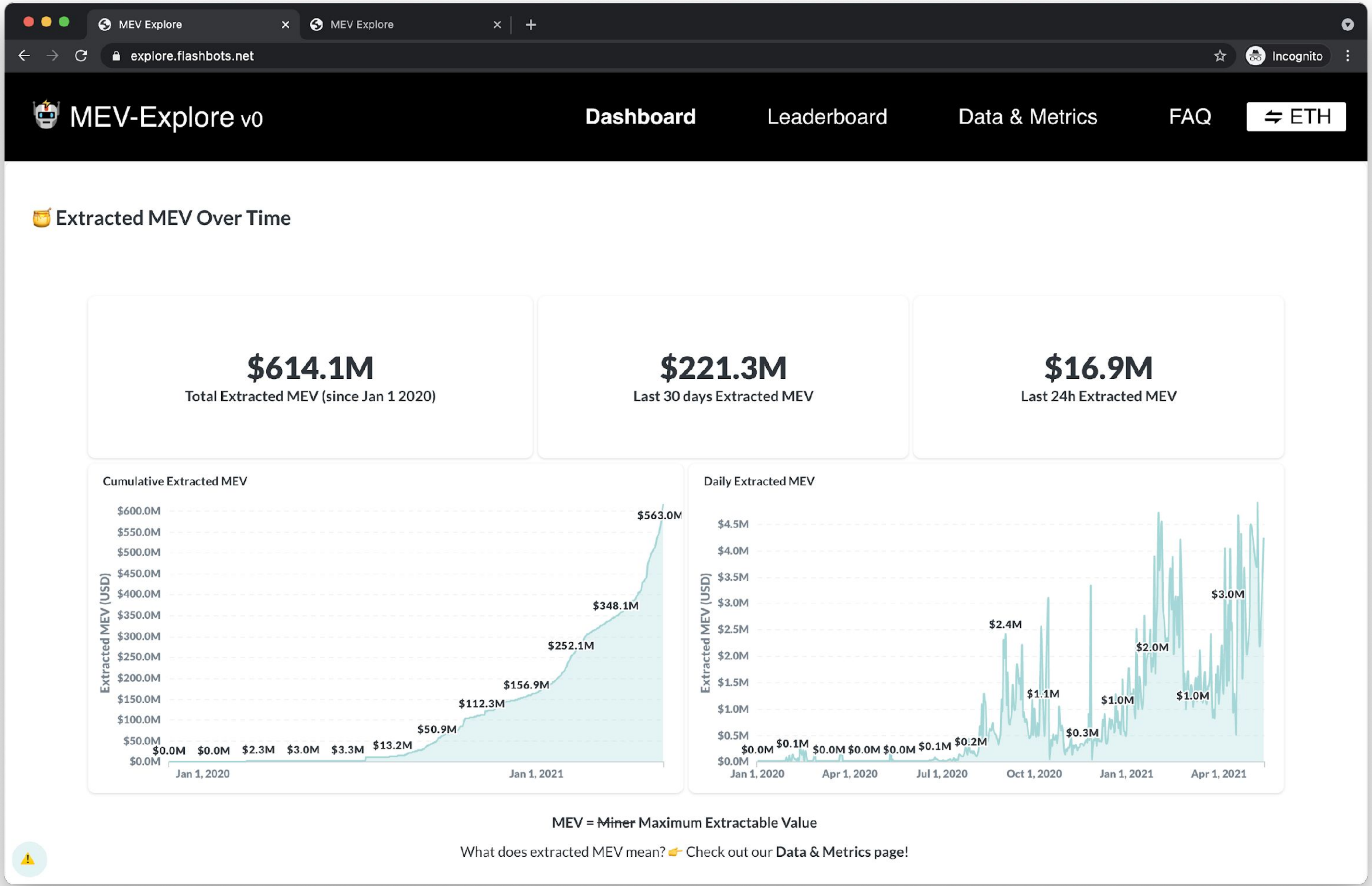
↓

Liquidation!

↓

Who will liquidate?

How much MEV?



MEV Explore
MEV Explore
explore.flashbots.net
Incognito
MEV-Explore v0
Dashboard
Leaderboard
Data & Metrics
FAQ
ETH
Extracted MEV Over Time
$614.1M
Total Extracted MEV (since Jan 1 2020)
$221.3M
Last 30 days Extracted MEV
$16.9M
Last 24h Extracted MEV
Cumulative Extracted MEV
Extracted MEV (USD)
$600.0M
$550.0M
$500.0M
$450.0M
$400.0M
$350.0M
$300.0M
$250.0M
$200.0M
$150.0M
$100.0M
$50.0M
$0.0M
$0.0M $0.0M $2.3M $3.0M $3.3M $13.2M $50.9M $112.3M $156.9M $252.1M $348.1M $563.0M
Jan 1, 2020
Jan 1, 2021
Daily Extracted MEV
$4.5M
$4.0M
$3.5M
$3.0M
$2.5M
$2.0M
$1.5M
$1.0M
$0.5M
$0.0M
$0.0M $0.1M $0.0M $0.0M $0.0M $0.1M $0.2M $2.4M $1.1M $0.3M $1.0M $2.0M $1.0M $3.0M
Jan 1, 2020
Apr 1, 2020
Jul 1, 2020
Oct 1, 2020
Jan 1, 2021
Apr 1, 2021
MEV = Miner Maximum Extractable Value
What does extracted MEV mean? Check out our Data & Metrics page!

How much MEV? – Sandwich Attacks

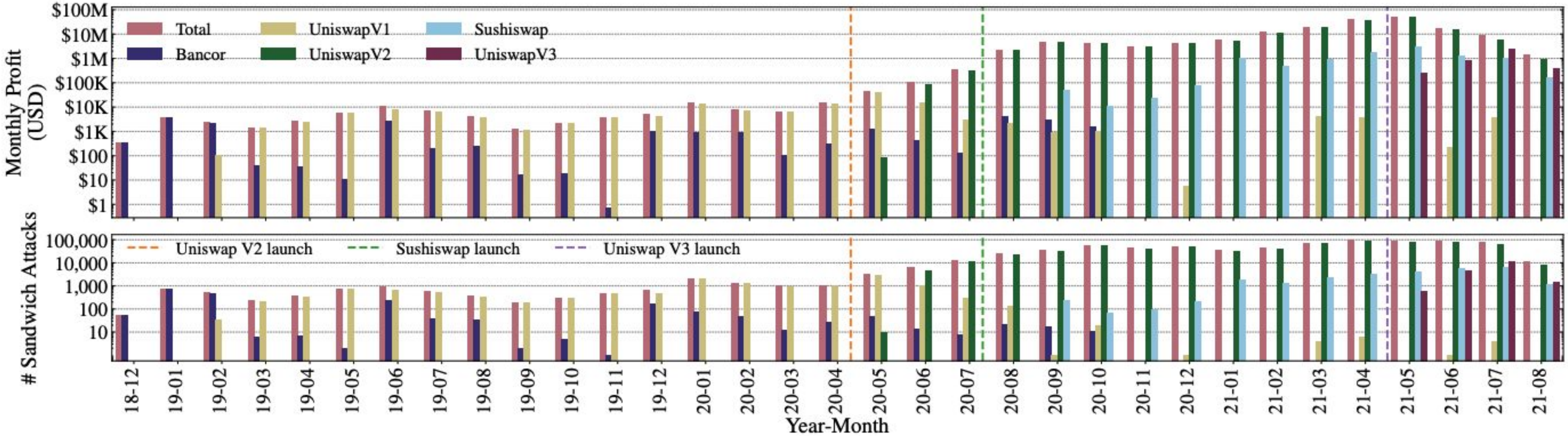
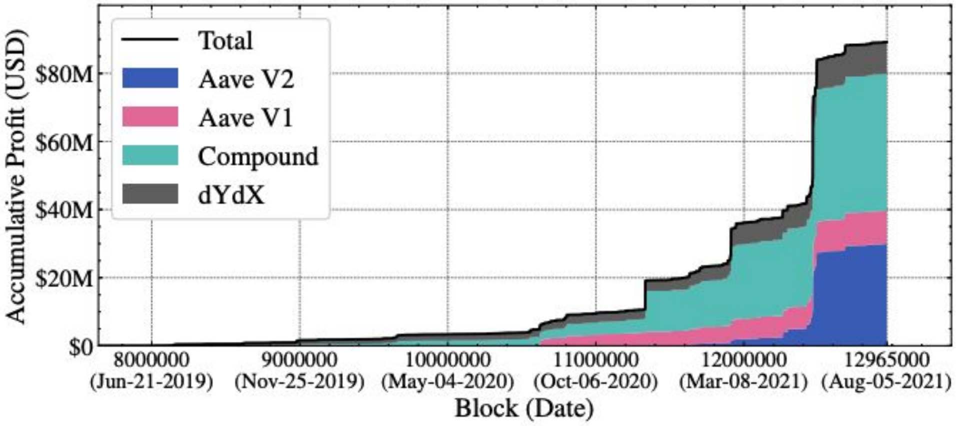
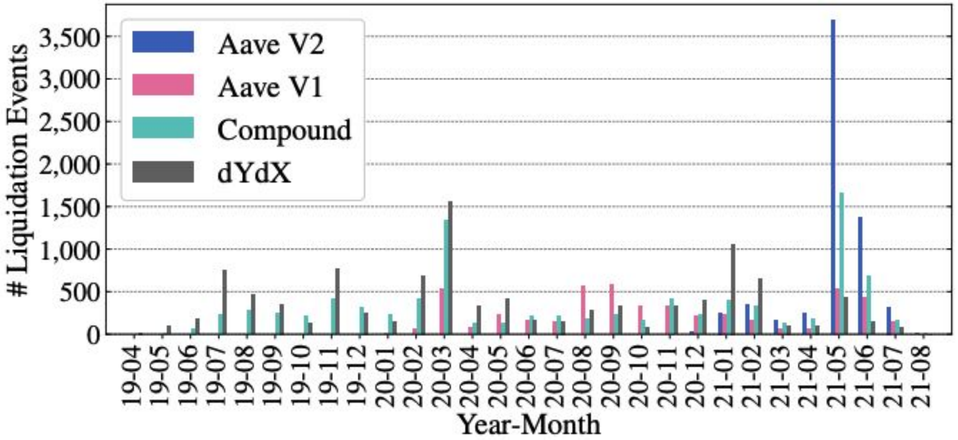


Fig. 3: Sandwich attacks, from block 6803256 (1st of December, 2018) to 12965000 (5th of August, 2021).

How much MEV? – Liquidations



(a) Accumulative profit of fixed spread liquidations.



(b) The monthly number of fixed spread liquidation events.

Fig. 5: The number of liquidations increase in months where the ETH price collapses, e.g., in March, 2020 and May, 2021.

How much MEV? – Arbitrage

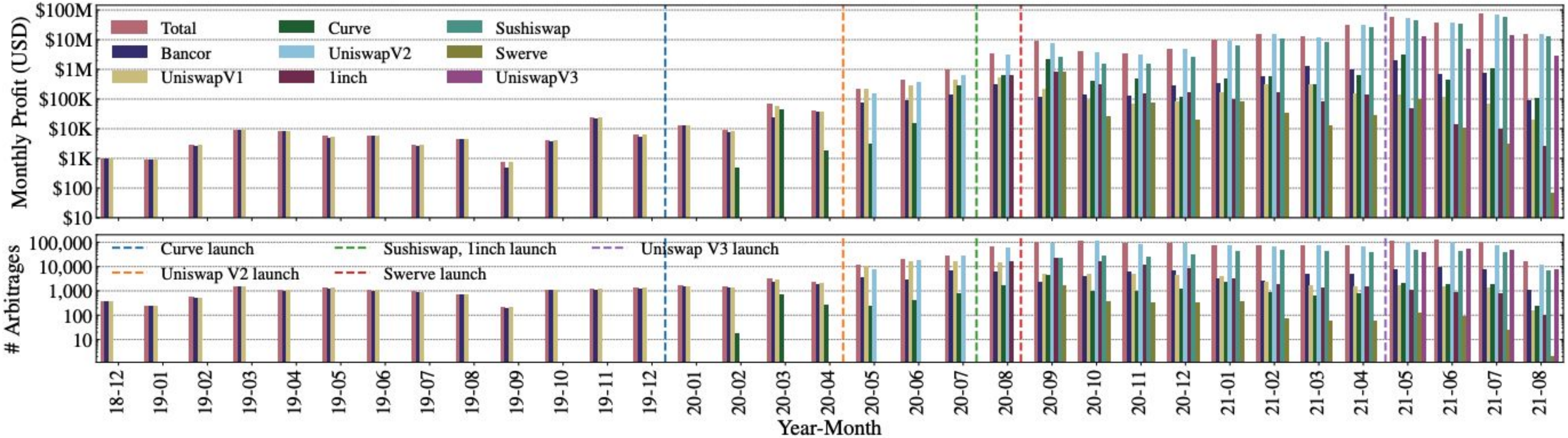


Fig. 7: Monthly arbitrage statistics from block 6803256 (1st of December, 2018) to block 12965000 (5th of August, 2021).

Transaction Replay Attacks

Generalized Front-Running

- “Copy Cat” or “Replay”
  - Observe transaction on the network layer
  - Replace certain data, sign, and broadcast copy

- Potential Profit
  - 35M USD over 32 months
  - 188,365 profitable transactions (0.02%)
  - Real-time algorithm (0.18s ± 0.29)

Generalized Front-Running Algorithm & Results

Algorithm 1: Transaction Replay Algorithm.

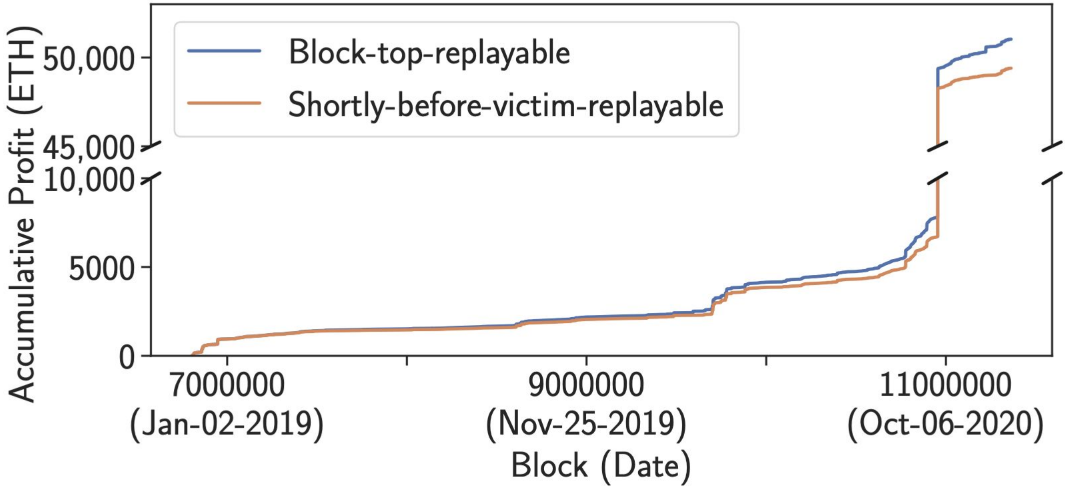
Input: The current highest block B_i; the potential victim transaction T_V; the adversarial account address $\mathcal{A}$.

Function `ConstructReplay`(T_V, $\mathcal{A}$):
- $T.sender \leftarrow \mathcal{A}$
- $T.value \leftarrow T_V.value$
- $T.input \leftarrow$ substituting $T_V.sender$ in $T_V.input$ with $\mathcal{A}$
- **return** T

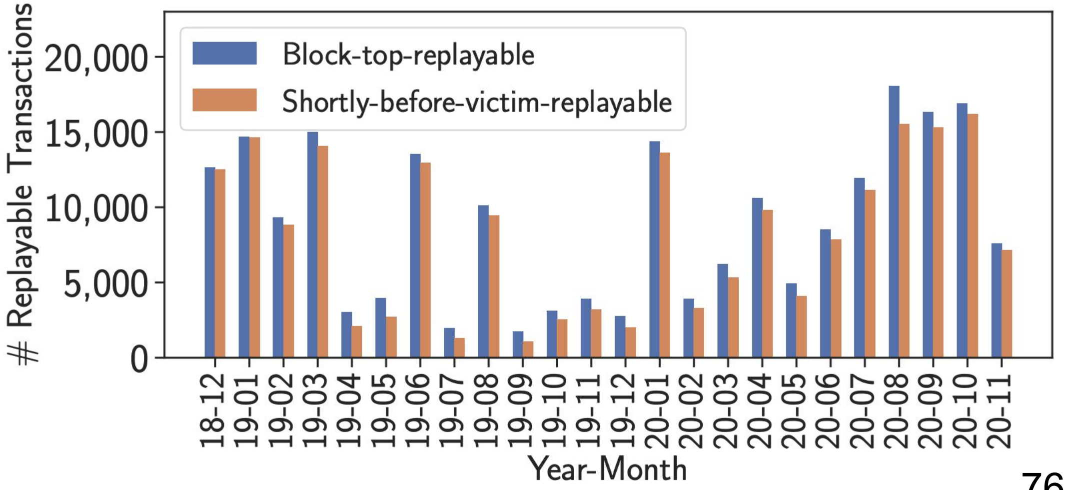
end

Algorithm `TransactionReplay`(T_V, $\mathcal{A}$):
- $T_{replay} \leftarrow$`ConstructReplay`(T_V, $\mathcal{A}$)
- Concretely Execute T_{replay} upon block B_i
- **if** T_{replay} *is profitable* **then**
  - Front-run T_V with T_{replay}
- **end**

end

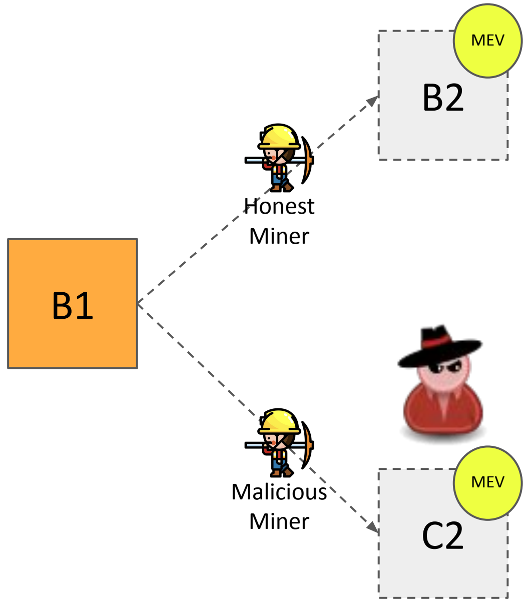


(a) Accumulative profit that can be extracted by replay attacks.

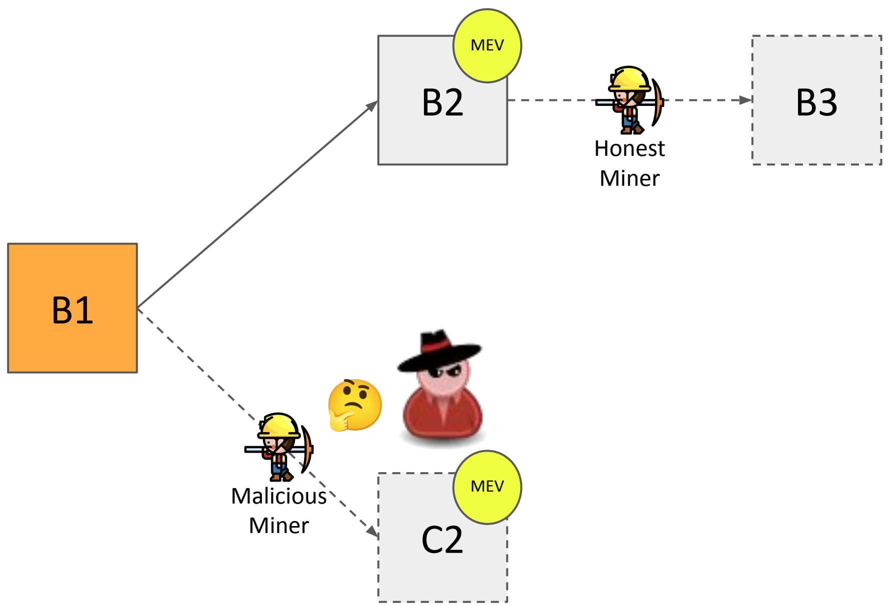


BEV Forking and Chain Reorganisation

The dangers of naively maximizing MEV

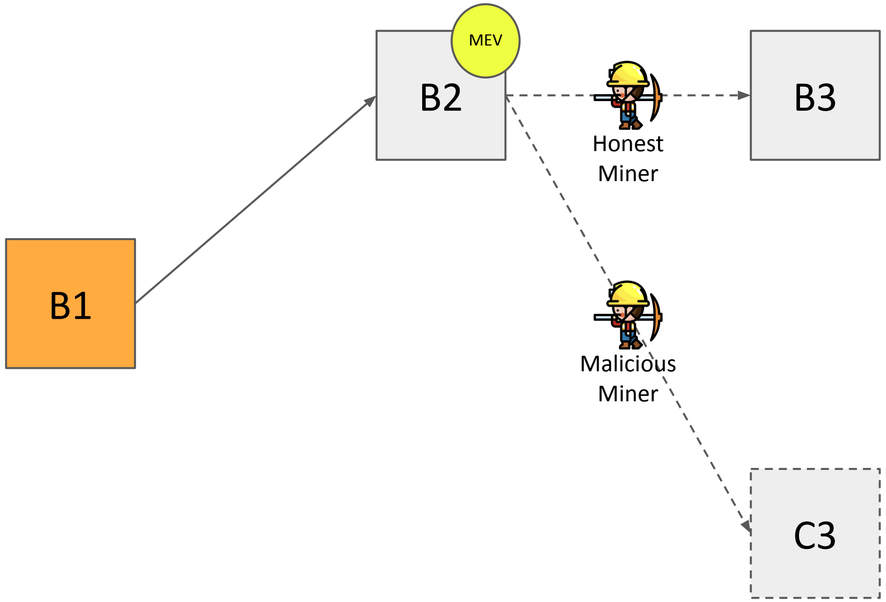


The dangers of naively maximizing MEV



The dangers of naively maximizing MEV

Case 1:

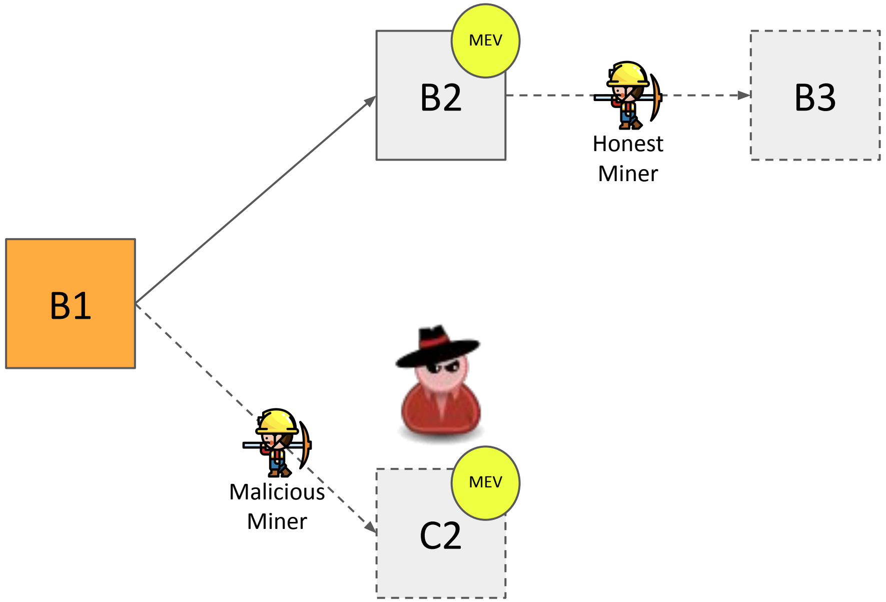


Case 1:

Malicious miner forfeits MEV opportunity

The dangers of naively maximizing MEV

Case 2:



Case 1:

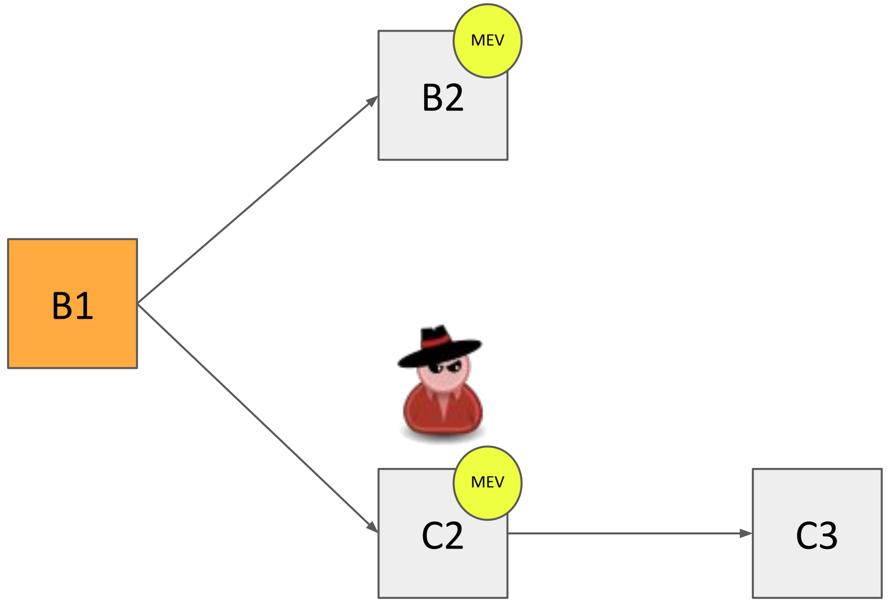
Malicious miner forfeits MEV opportunity

Case 2:

Keeps mining block C2

The dangers of naively maximizing MEV

Case 2:



Case 1:

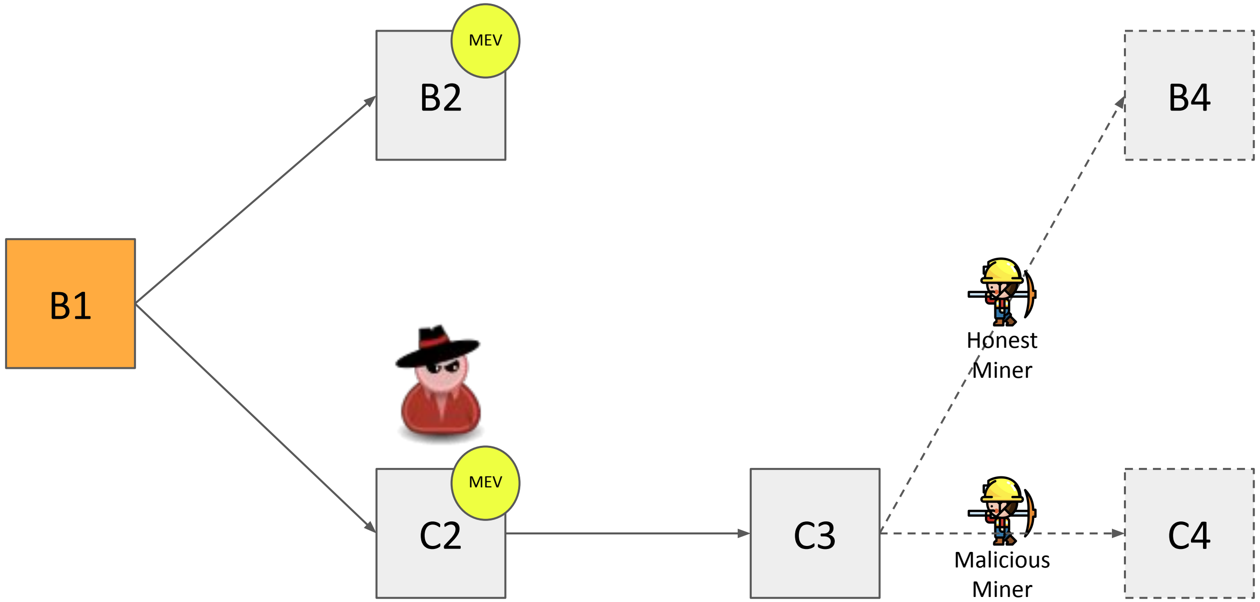
Malicious miner forfeits MEV opportunity

Case 2:

Keeps mining block C2

The dangers of naively maximizing MEV

Case 2:



Case 1:

Malicious miner forfeits MEV opportunity

Case 2:

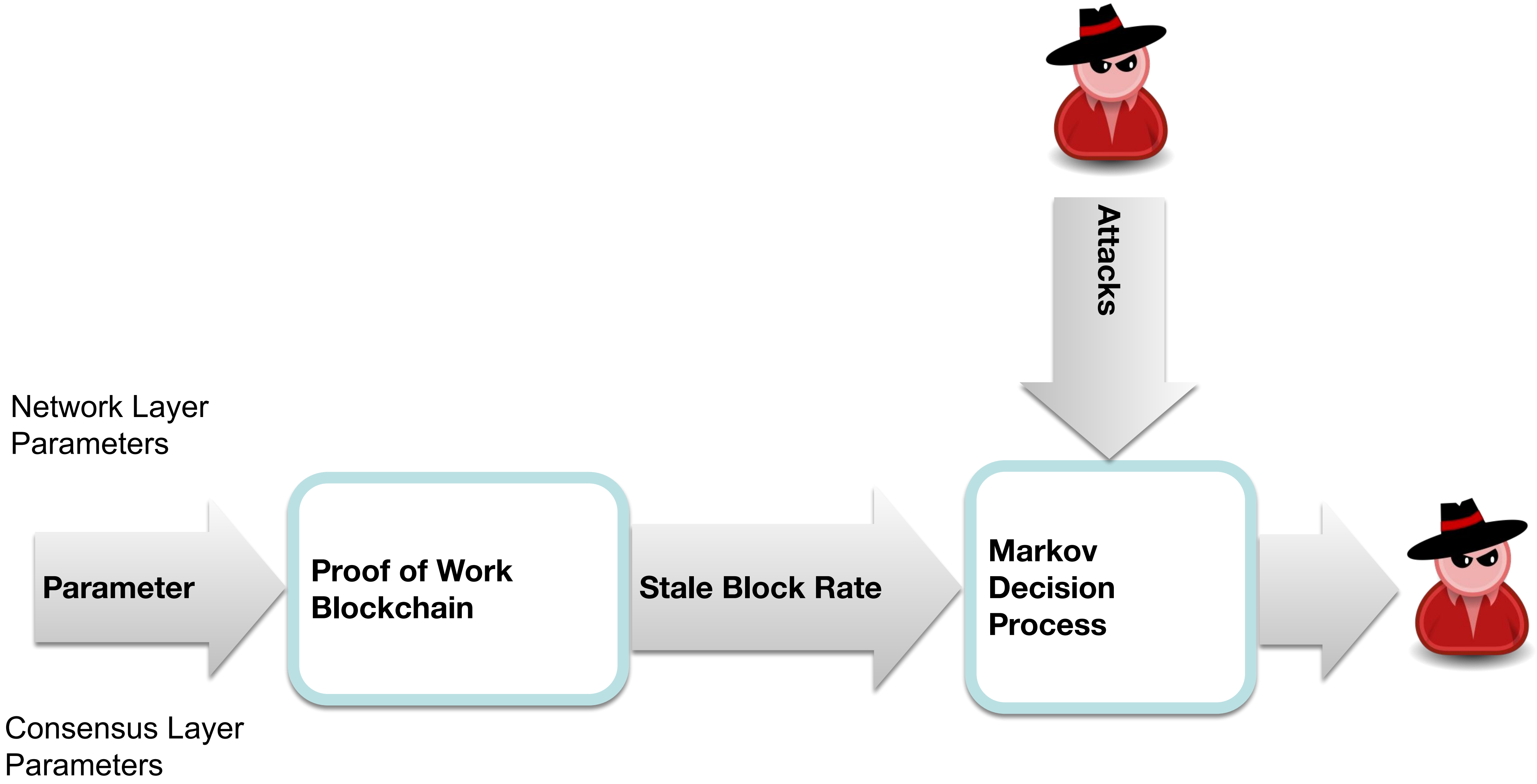
Keeps mining on block C2

→ Waste computational power

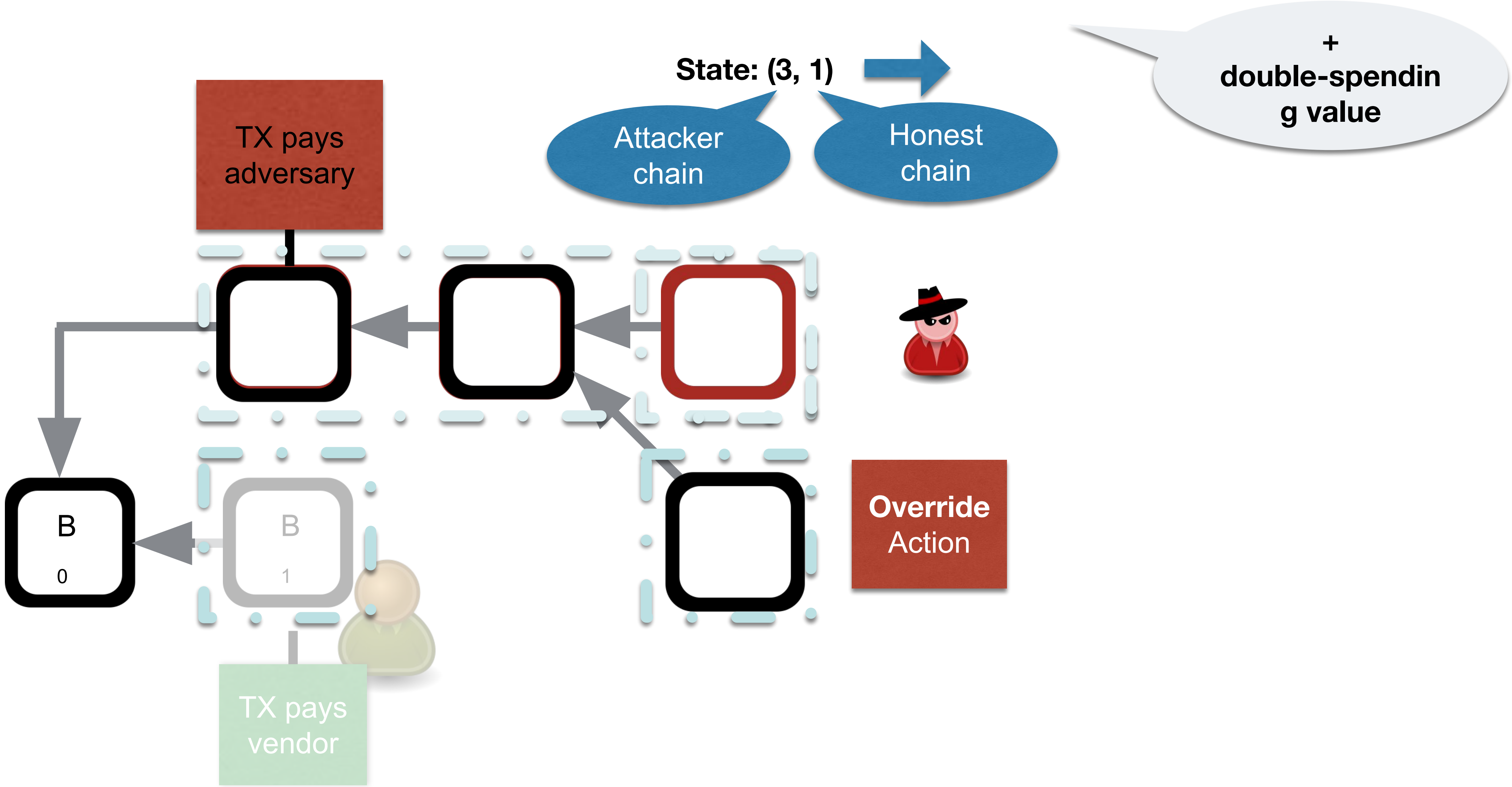
→ Increase stale block rates and risks for:

- **Double spending**
- **Selfish mining**

Markov Decision Process (MDP)



Markov Decision Process (MDP)



Reducing MEV is the key to security (example)

10%
miner

+

MEV, 4x average
block reward

==

"On the just-in-time discovery of profit-generating transactions in defi protocols." peer-reviewed at S&P'21

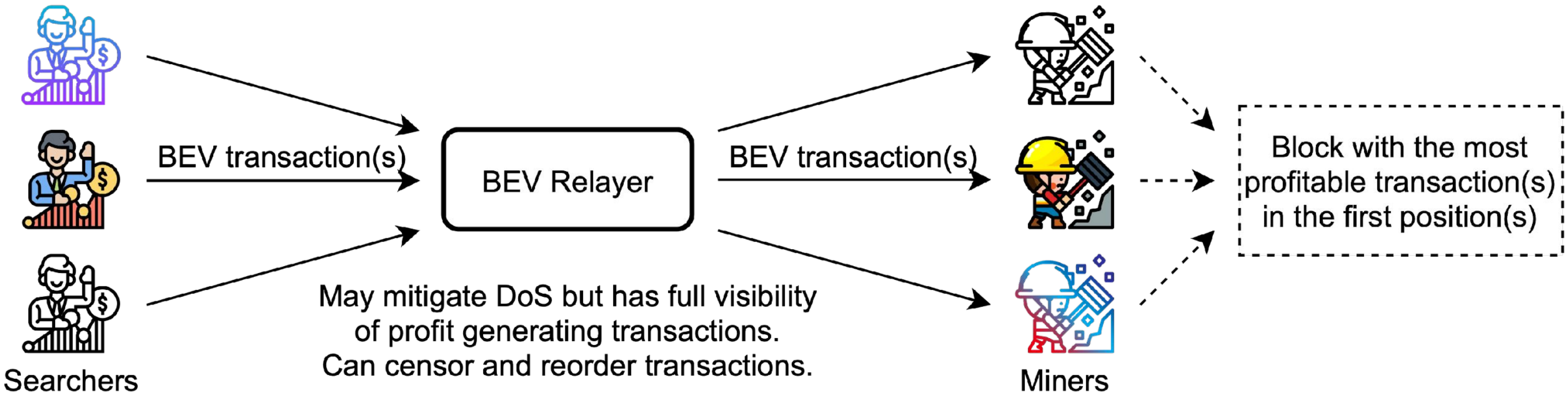
Reducing MEV is the key to security (example)

874x

BEV Relayer &
How to Mitigate BEV?

https://defi-learning.org

BEV Relay Architecture



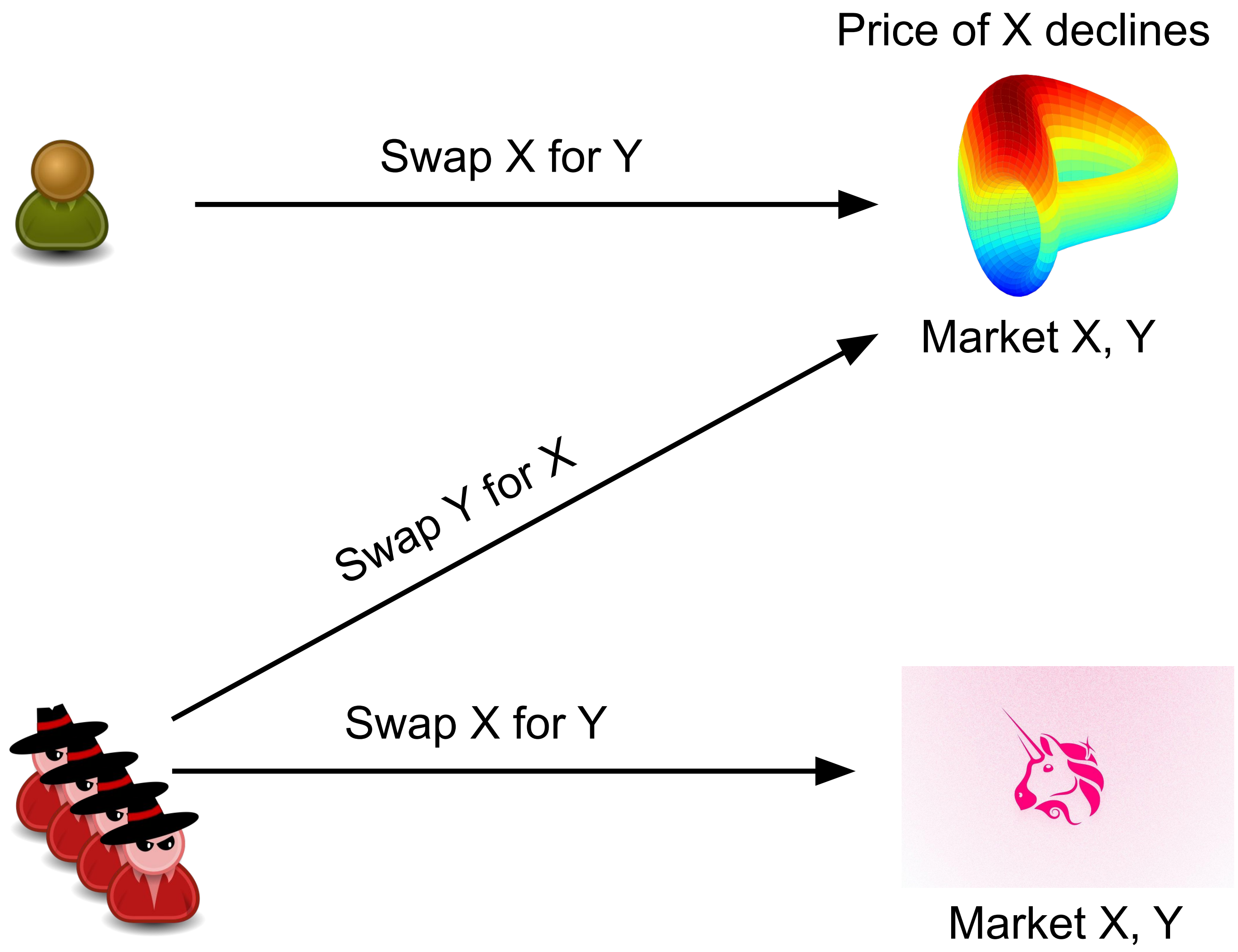
BEV Relayer Concerns

- BEV provably incentivises miners to fork (cf. S&P'21)
- BEV relayer centralise the P2P Network
- The relayer may resell/profit from searcher strategies
- The relay system doesn't necessarily reduce P2P overhead
- A for profit company distributes the geth client to >50% of the miners
- Innocent users are being stolen from systematically

Anti-MEV Solution Space

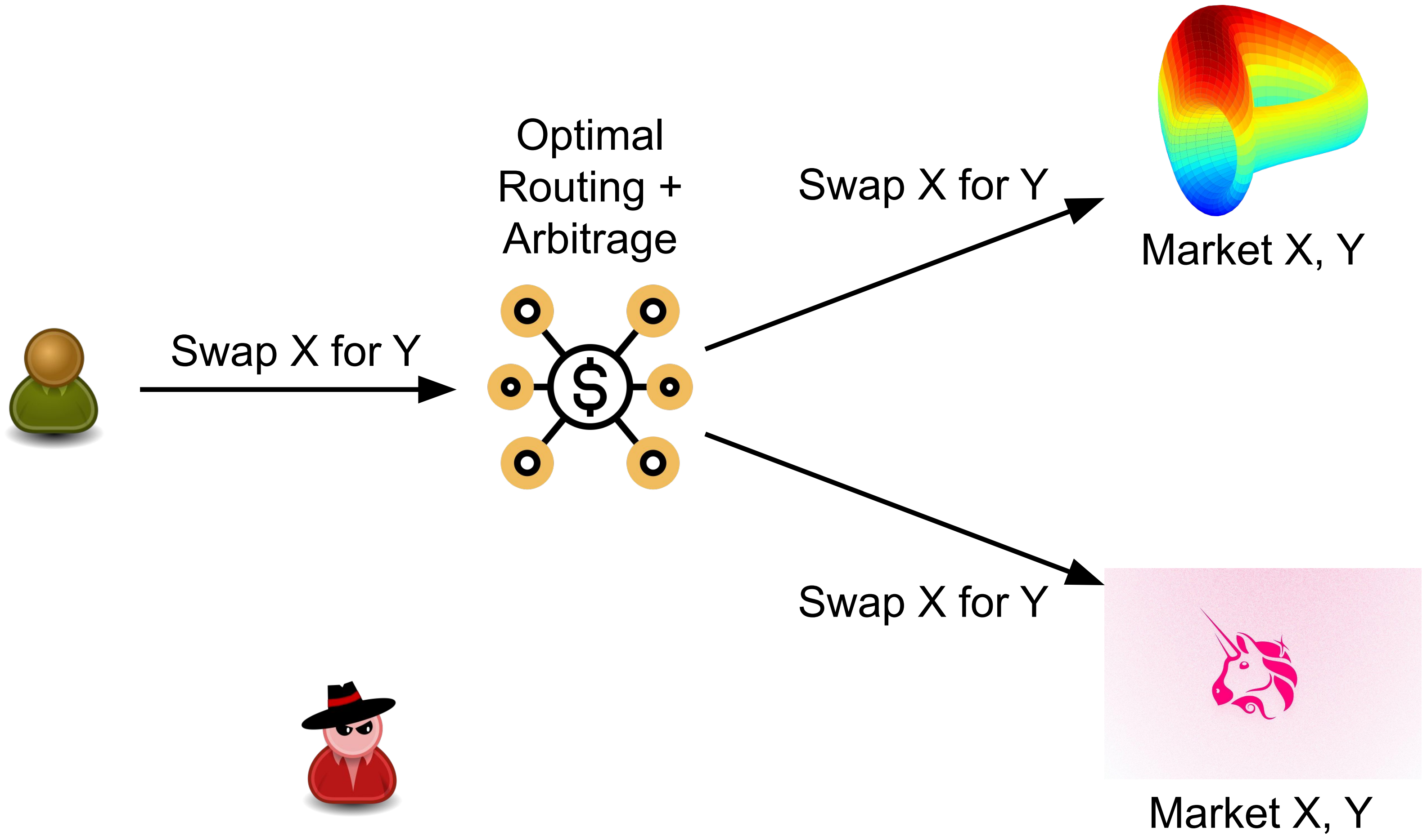
- Fair-Ordering on the Blockchain Layer
  - e.g., Aequitas Protocol Family
- Fixing MEV of existing dApps
  - Merging AMM DEX into one
    - On-chain aggregators such as A2MM (see DEX lecture)
- Designing MEV-Mindful dApps
  - Avoiding MEV by design
  - e.g., a price oracle update immediate performs a liquidation
- Might not fix cross-chain MEV..

Application-Specific MEV Mitigation



- Causes
  - Back-run Flooding
  - Network Congestions
  - Price Gas Auctions
  - Transaction Fee Increase
- The user forgoes an arbitrage opportunity.

Application-Specific MEV Mitigation



- Cons
  - Higher Gas Fees
- Pros
  - Better ex rate
  - Arbitrage profit
  - MEV reduction
  - Healthier chain