



# □ DissFuzz □

Decentralised fuzzing

# Participants



*Milana Maric*



*Nikita Evsiukov*



*Vladimir Hanin*



*Chau Ying Kot*



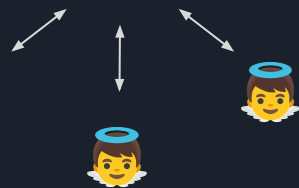
*Qifan Pan*



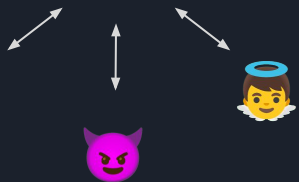
*Derya Cöğendez*

# Fuzzing

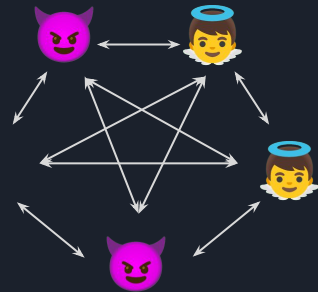
libFuzzer  
AFL



OSSFuzz

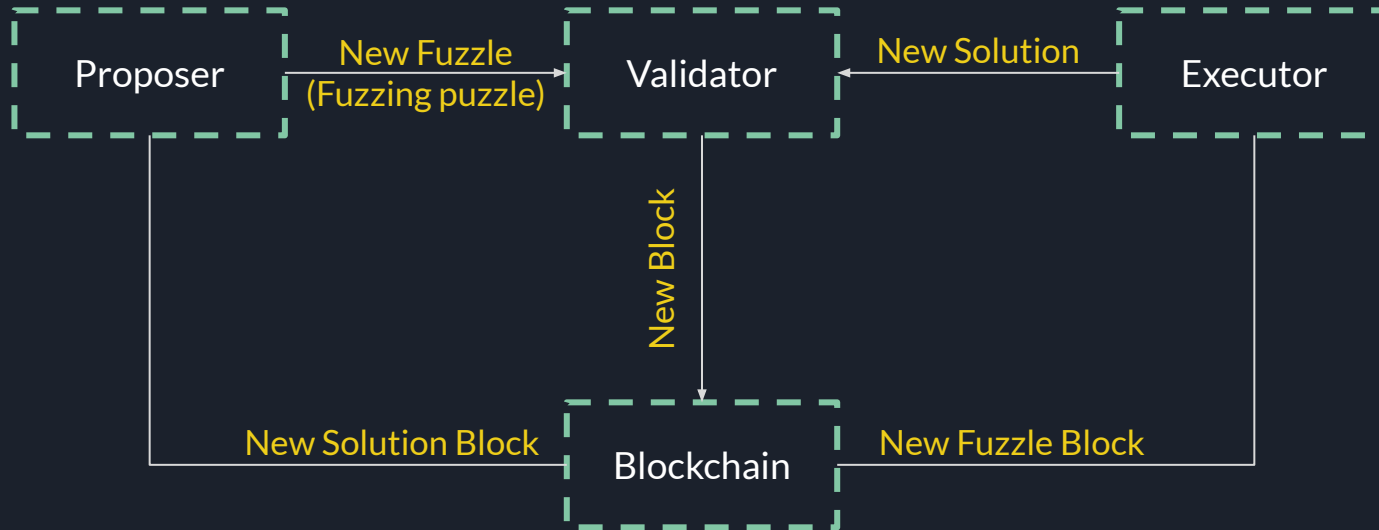


FuzzCoin



DissFuzz

# System overview

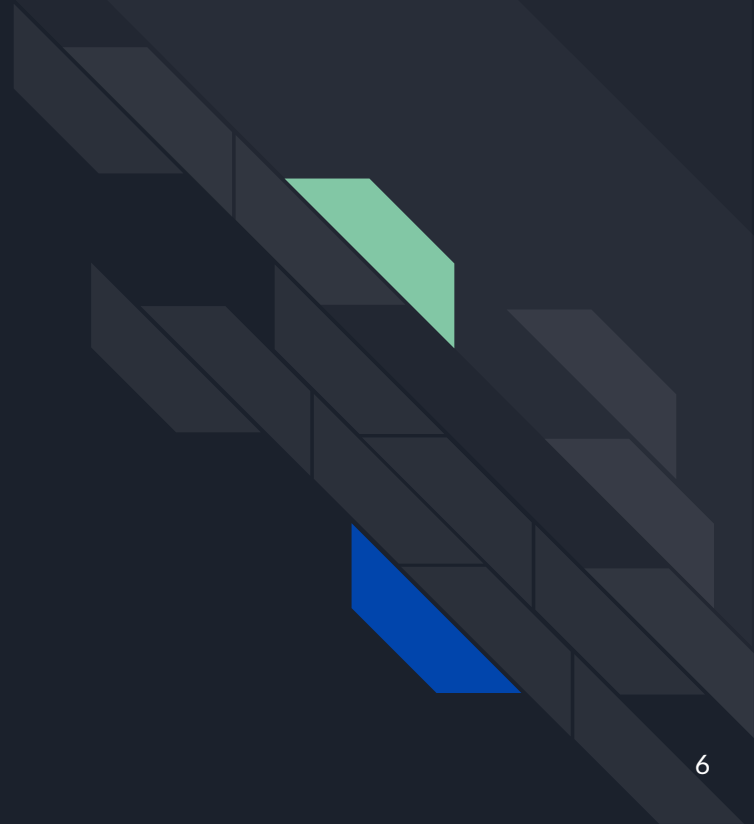




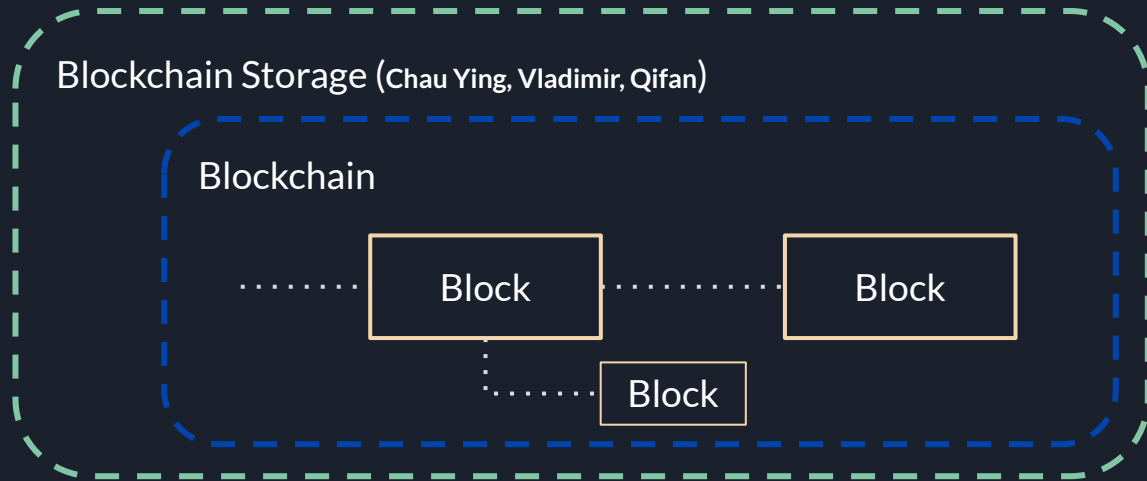
# Threat model

Problem	Solution
Executor submitting solution without doing any work	Proof-of-fuzzing-work
Proposer submitting a fuzzle without solution	Proof-of-stake
Proposer spamming with fuzzles	
Ability to propose more difficult fuzzles	
Stealing solutions from other nodes	Digital signature
Executor report bugs that do not exist	Majority of correct Validators
Forks in the blockchain	

# Features



# Blockchain



# Blockchain

Blockchain Storage (Chau Ying, Vladimir, Qifan)

Blockchain

Block

Block

Block



- Balance Manager (Qifan)
- Fuzzle Tracker (Qifan)
- Mempool (Qifan, Vladimir)







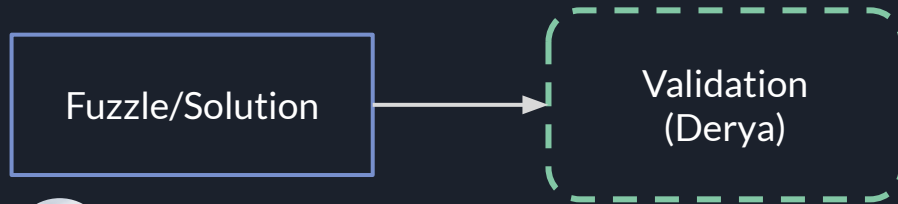
# Block



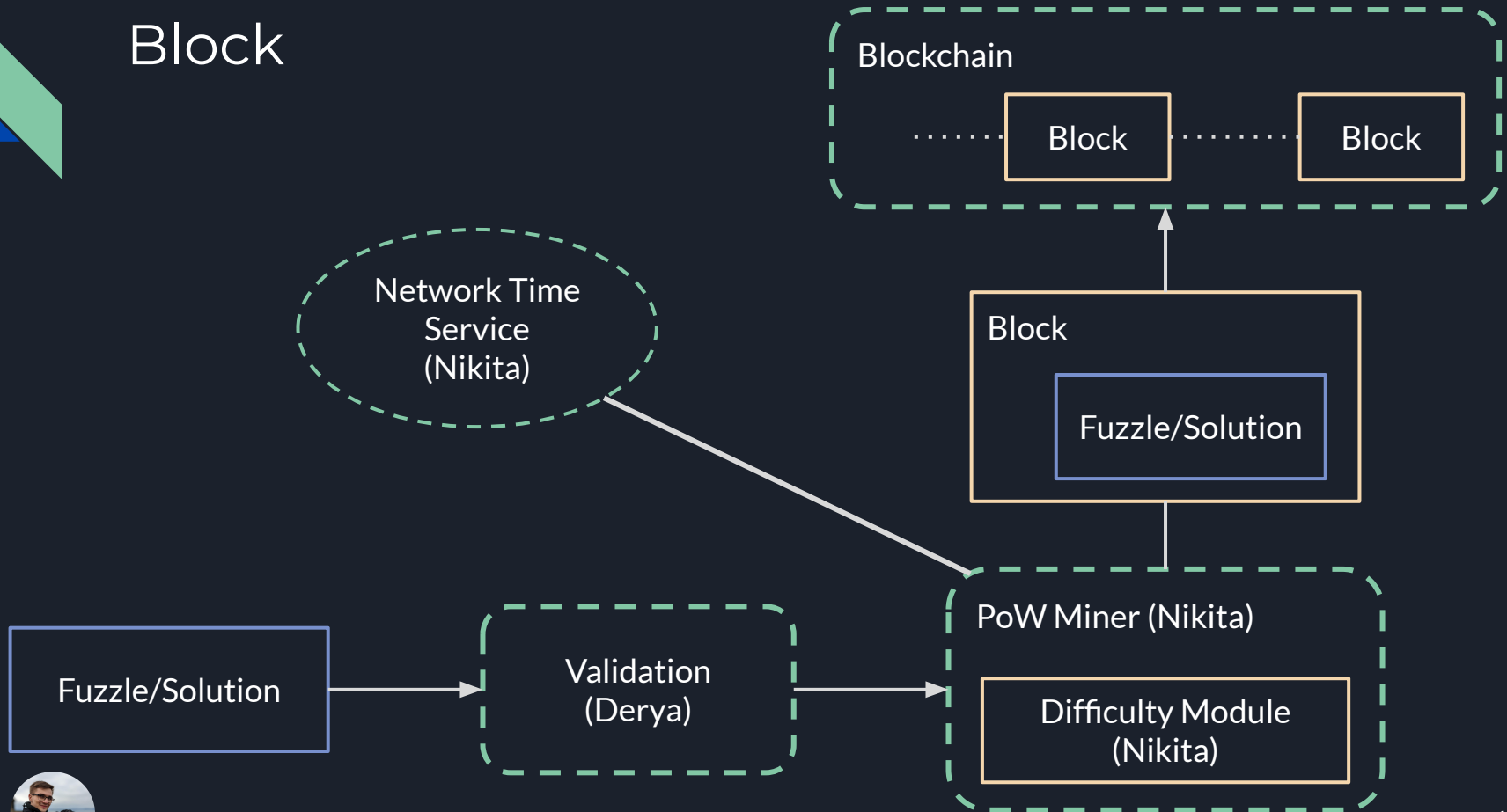
Fuzzle/Solution



# Block



# Block





# Validator

- Validate the content of the block
- Mine the nonce with a PoW schema
- Receive money for his work

=> Maintain the correctness of the blockchain

- Download the program to mitigate congestion



# Proposer

```
#include <stddef.h>
#include <stdint.h>

bool FuzzMe(const uint8_t *Data, size_t DataSize) {
    return DataSize >= 3 && Data[0] == 'F' && Data[1] == 'U' && Data[2] == 'Z' && Data[3] == 'Z'; // :(
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t * Data, size_t Size) {
    FuzzMe(Data, Size);
    return 0;
}
```



# Proposer

```
#include <stddef.h>
#include <stdint.h>

bool FuzzMe(const uint8_t *Data, size_t DataSize) {
    return DataSize >= 3 && Data[0] == 'F' && Data[1] == 'U' && Data[2] == 'Z' && Data[3] == 'Z'; // :(
}

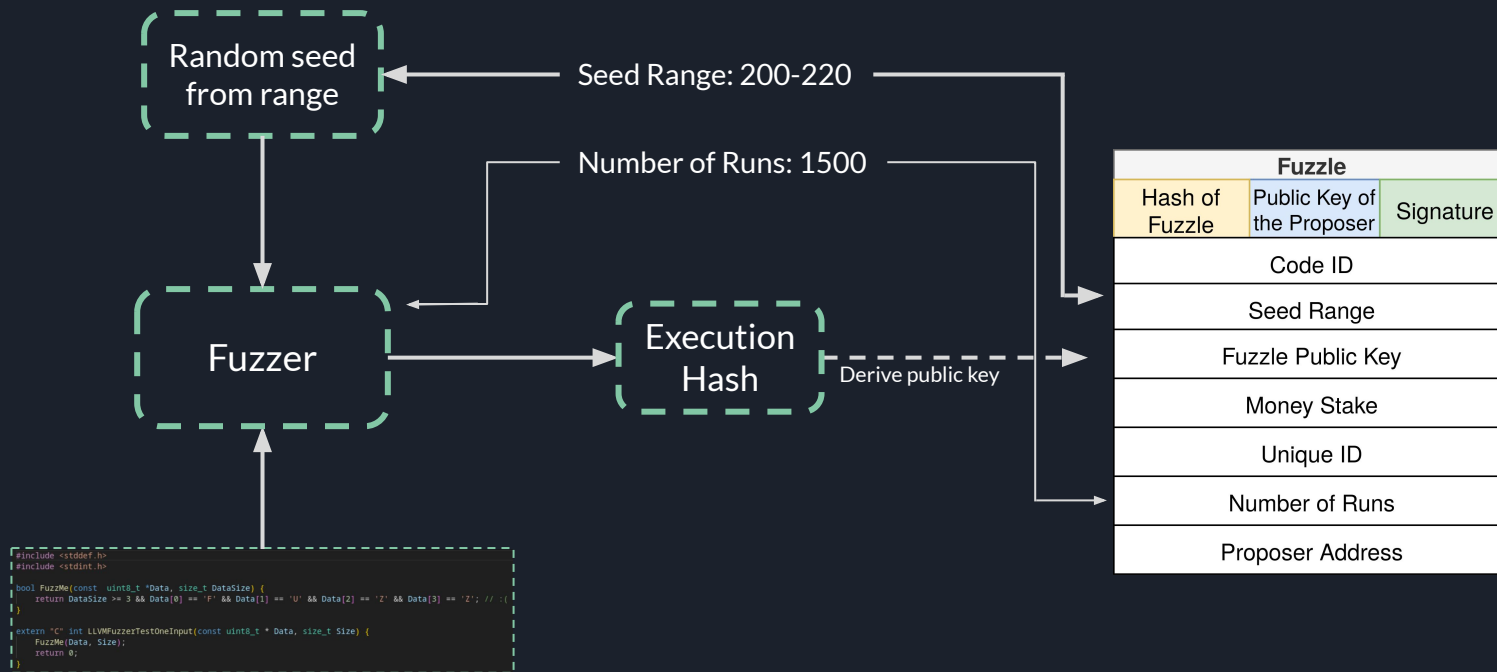
extern "C" int LLVMFuzzerTestOneInput(const uint8_t * Data, size_t Size) {
    FuzzMe(Data, Size);
    return 0;
}
```

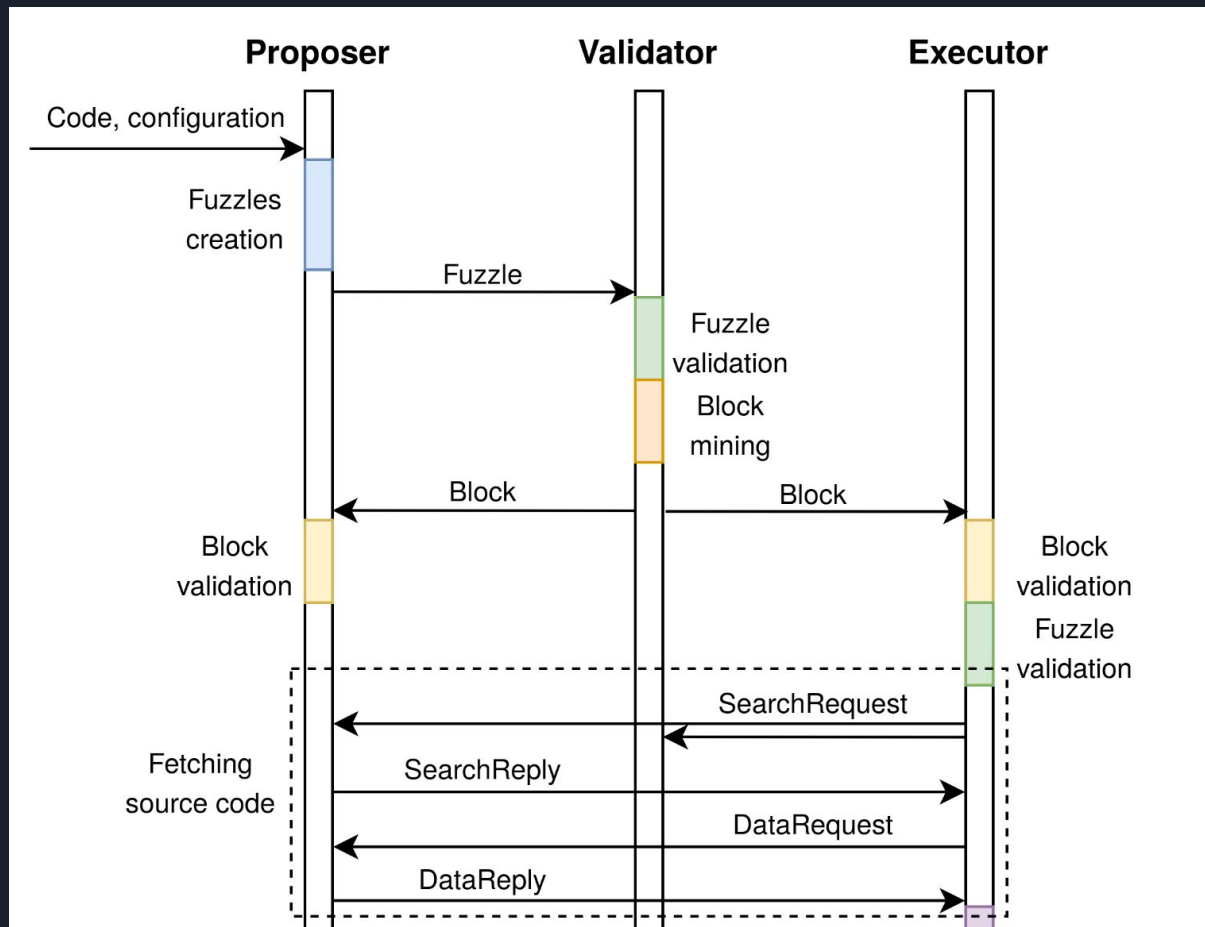
Metahash

Fuzzle		
Hash of Fuzzle	Public Key of the Proposer	Signature
Code ID		
Seed Range		
Fuzzle Public Key		
Money Stake		
Unique ID		
Number of Runs		
Proposer Address		



# Proposer







# Executor

Seed Range: 1-100

Number of Runs: 1000

Fuzzer

Execution Hash

Encrypt (Hash of Fuzzle | Solution Hash)

```
#include <stdint.h>
#include <stdint.h>

bool FuzzMe(const uint8_t *Data, size_t DataSize) {
    return DataSize >= 3 && Data[0] == 'F' && Data[1] == 'U' && Data[2] == 'Z' && Data[3] == 'Z'; // 
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    FuzzMe(Data, Size);
    return 0;
}
```

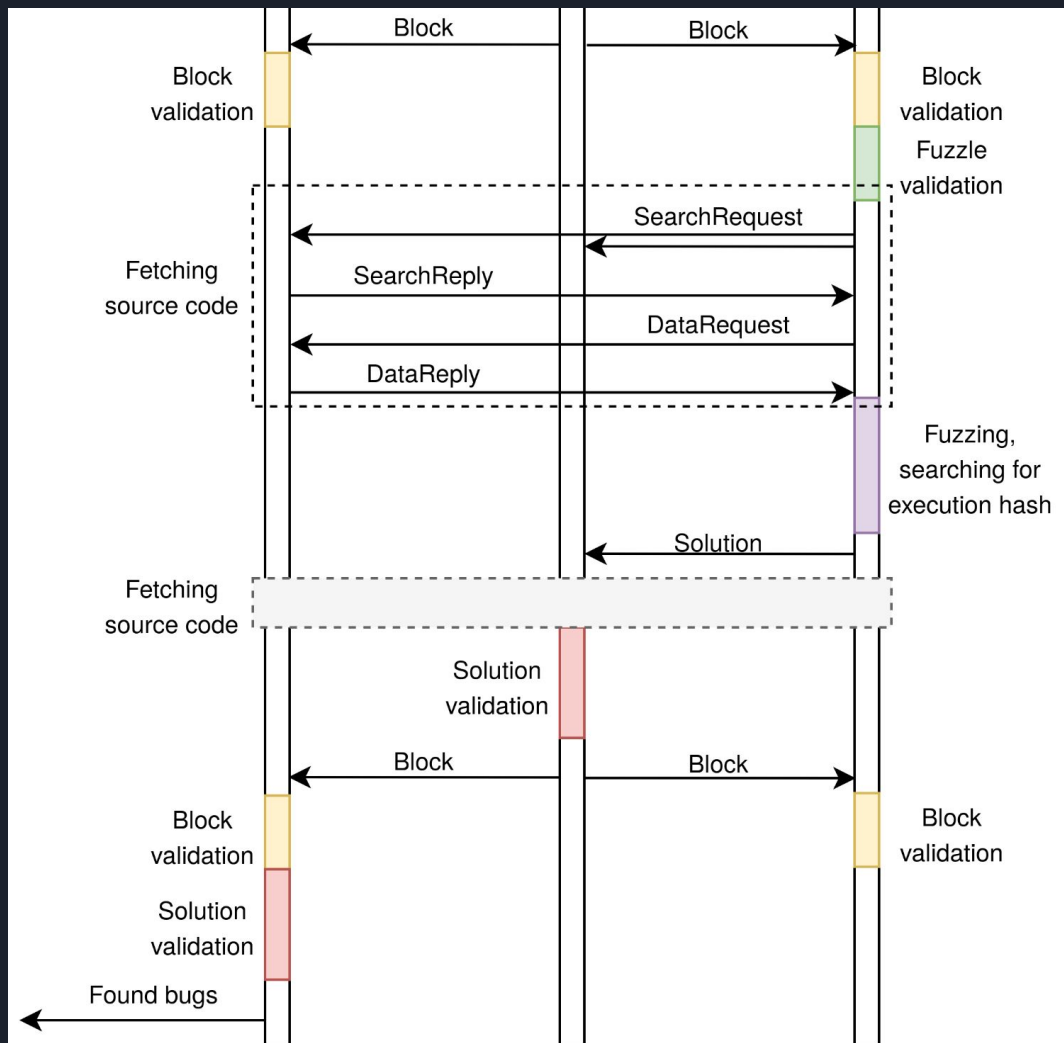
## Fuzzle

Hash of Fuzzle	Public Key of the Proposer	Signature
Code ID		
Seed Range		
Fuzzle Public Key		
Money Stake		
Unique ID		
Number of Runs		
Proposer Address		

## Solution

Hash of Fuzzle	Solution Hash	Public Key of Executor	Signature	Proof
Executor Prize				
Seeds for Additional Bugs Found (Optional)				
Proposer Refund				







# Challenges


## Design

- Money generation: VALIDATOR
- Fuzzle spamming: staking



# Challenges

## Testing: RANDOMNESS

- Mining time 
- Execution time
- Blockchain forks



# Demo

(don't include this slide in the video, this slide is only to show where the demo comes during the video)



# Evaluation

## Testing Infrastructure

**Matrix** testing framework

**19** CI workflows

## End-to-End tests

**7** Blockchain scenarios

**8** Fuzzing scenarios

## Benchmarks

**4** scenarios

**4** hardware setups

# Evaluation

## Testing Infrastructure

**Matrix** testing framework

19 CI workflows



Simplify & speed up writing scenarios



Generate random topology



Automatize configuration & assertions



Simulate faulty and malicious nodes



Save tons of team time!



# Evaluation

## End-to-End tests

7 Blockchain scenarios

8 Fuzzing scenarios



Maintain constant block addition time



Consensus on blockchains



Invalid blocks attack



Forks resolution





# Evaluation

## End-to-End tests

7 Blockchain scenarios

8 Fuzzing scenarios

- ❑ Multiple executors workload
- 👹 Invalid Fuzzle attack
- 👹 Invalid Solution attack
- 👛 Solution stealing attack



# Evaluation

## Benchmarks

4 scenarios

4 hardware setups



System overhead



Validators overhead



Executors speedup



Proposals scalability



# Evaluation

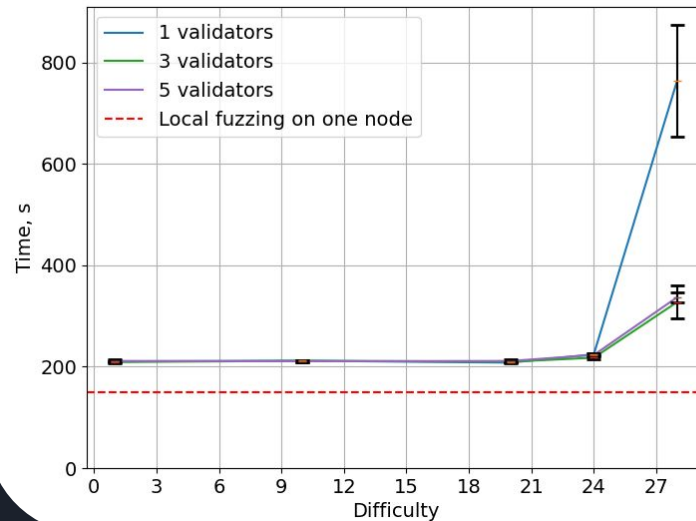
## System overhead

1 Proposer

1 Executor

1-5 Validators

5 Fuzzles



30-40% overhead

# Evaluation

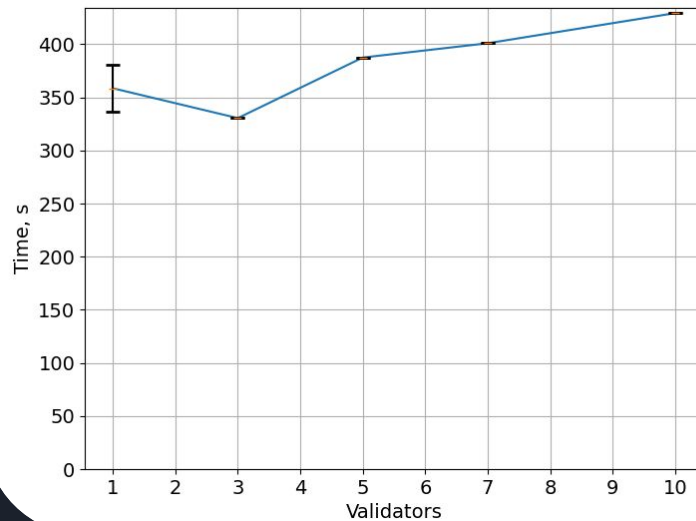
## Validators overhead

1 Proposer

1 Executor

1-10 Validators

5 Fuzzles



Linear relationship

# Evaluation

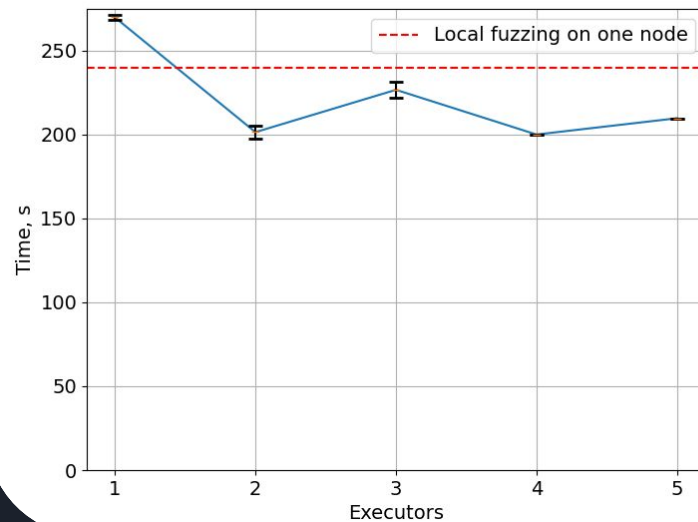
## Executors speedup

1 Proposer

1-5 Executors

5 Validators

8 Fuzzles



17% speedup

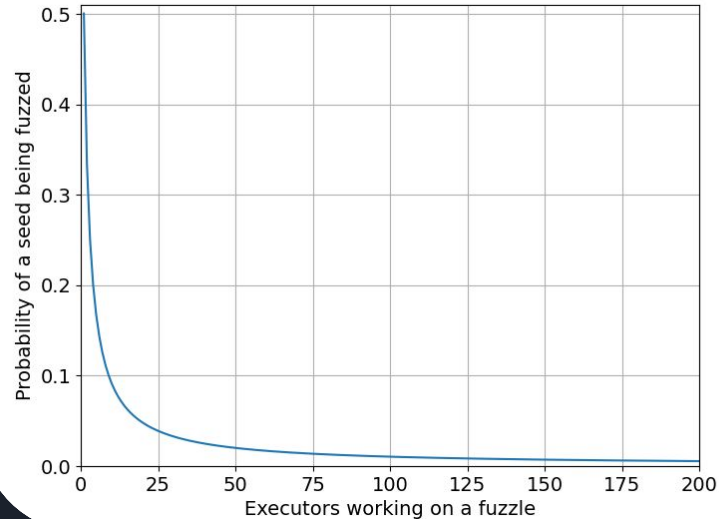
# Evaluation

## Fuzzing Effectiveness

50% for 1 executor

10% for 9 executors

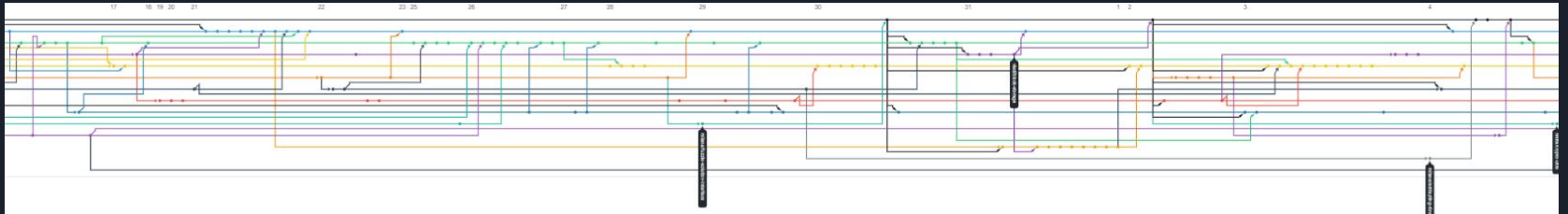
Exponential decrease







Fuzzle range size and overlapping adjust effectiveness

# Statistics of project

- 200 additional tests
- 34,692 lines of code in total (21,173 written for HW3)
- 26,952 additions and 10,148 deletions
- 40 PRs
- 6,552 workflow run results
- 19 workflows
- 12 design docs



# Summary

 <p>Milana Maric</p>	 <p>Nikita Evsiukov</p>	 <p>Vladimir Hanin</p>
<ul style="list-style-type: none"><li>• Peerster Implementation</li><li>• File Sharing</li><li>• Executor</li><li>• End-to-End tests</li></ul>	<ul style="list-style-type: none"><li>• Modified libFuzzer</li><li>• Proposer</li><li>• Integration tests</li><li>• Benchmarks</li></ul>	<ul style="list-style-type: none"><li>• Mempool</li><li>• Blockchain Storage</li><li>• GUI</li></ul>
 <p>Chau Ying Kot</p>	 <p>Qifan Pan</p>	 <p>Derya Cögendez</p>
<ul style="list-style-type: none"><li>• Blockchain Storage</li><li>• Valitador</li><li>• Integration tests</li></ul>	<ul style="list-style-type: none"><li>• Balance Manager</li><li>• Fuzzle Tracker</li><li>• Mempool reorg (design)</li><li>• Fuzzing Statistics</li></ul>	<ul style="list-style-type: none"><li>• Docker</li><li>• Go Clang/Crypto Wrappers</li><li>• Validation Module</li><li>• Linter</li></ul>