EVM

An (advanced) introduction

Miguel Palhas - @naps62

About me

- **naps62**
- **subvisual.com**, a venture studio, since 2012 *we're hiring!*
- helped found and build Utrust (2017-2020)
- smart contract developer / auditor since 2019
- building **ethui.dev**, a wallet for Ethereum developers

EVM / Solidity intro

Hello, World

```
contract Counter {
  uint256 public value;

  constructor(uint256 _v) {
    value = _v
  }

  function add(uint256 _x) external returns (uint256) {
    value += _x;
    return value;
  }
}
```

Transaction encoding

Take the function header	`function add(uint256 x) external returns (uint256)`
Convert to canonical form	`add(uint256)`
keccak256 hash it	`0x1003e2d21e48445eba32f76cea1db2f704e754da30edaf86...`
First 4 bytes are the signature	`0x1003e2d2`

Append RLP-encoded arguments	`0x1003e2d2000000000000000000000000000`
	`0000000000000000000000000000000000001`

Basic ETH transfer

```
{
  /*
    origin address is not part of the payload, but is implicitly included in the
signature
    "from": "0x0000000000000000000000000000000000000001",
  */
  "to": "0x0000000000000000000000000000000000000002",
  "value": "0x100",
  "gas": "0x100000",
  "gasPrice": "0x100",
  "data": "",
  "nonce": "0x0", // Strictly sequential, per sender
  "chainId": "0x1" // Ethereum mainnet
}
```

Contract deployment

```
{
  "to": "", // no destination implies contract creation
  "data": "0x6080604052...", // init code for the contract
  "value": "0x100",
  "gas": "0x100000",
  "gasPrice": "0x100",
  "nonce": "0x1",
  "chainId": "0x1"
}
```

Contract interaction

```
{
  "to": "0x0000000000000000000000000000000000002", // contract address
  "data": "0x6080604052...", // calldata, as seen earlier
  "value": "0x100",
  "gas": "0x100000",
  "gasPrice": "0x100",
  "nonce": "0x2",
  "chainId": "0x1"
}
```

Transactional function calls

All transactions are atomic, and will either:

- succeed (updating the EVM's state), or
- revert (no state changes, except for gas spent by the caller)

There is no no parallelism, txs are strictly ordered within a block.

Language details

Types

```
contract Types {
  uint256 x = 1;
  uint8 = 2;
  address alice = 0x1f9090aaE28b8a3dCeaDf281B0F12828e676c326;
  Counter counter = Counter(alice);

  Counter[] counters;
  mapping(uint256 => Counter) countersByIndex;

  struct Name {
    address addr;
    string name;
  }
  Name[] names;
}
```

Memory Types

```
contract Types2 {
  // constants
  uint256 public constant MAX_UINT256 = type(uint256).max;

  // contract storage, part of EVM's state
  uint256 private x;

  function increment() public {
    // in-memory stack, discarded after function execution
    uint256 tmp = x;

    if (tmp == MAX_UINT256) {
      revert("Overflow");
    }

    tmp += 1;
  }
```

Functions

```
contract Functions {
   // callable only as transaction calldata
  function add(uint256 _x) external;

  // callable both from a transaction and from within the contract
  function mul(uint256 _x) public;

  // only callable from within the contract
  function sub(uint256 _x) internal;
}
```

Cross-contract calls

```
interface Counter {
  function increment() external;
}

contract Caller {
  Counter counter;

  function incrementCounter() external {
    counter.increment();
  }
}
```

Developer concepts

EIPs

The formal process of proposing new features or standards for EVM contracts. Some examples:

EIP-20 - Token Standard

```
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    // ...
}
```

ERC20 Sample Implementation

```
contract ERC20 is IERC20 {
    uint256 public totalSupply;
    mapping (address => uint256) private _balances;

    // ...
}
```

ERC20 Sample Implementation

```
contract ERC20 is IERC20 {
    // ...

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }
    // ...
}
```

EIP-721 - Non-Fungible Token Standard

```
interface IERC721 {
    function balanceOf(address owner) external view returns (uint256 balance);
    function ownerOf(uint256 tokenId) external view returns (address owner);
    function safeTransferFrom(address from, address to, uint256 tokenId) external;
    function transferFrom(address from, address to, uint256 tokenId) external;
```

Permissions

```
contract PermissionedToken is ERC20 {
  mapping(address => bool) public whitelisted;

  function transfer(address to, uint256 value) public {
    require(whitelisted[msg.sender], "Not whitelisted");
    super.transfer(to, value);
  }
}
```

EIP-155

Simple replay attack protection

Enforcing inclusion of chainId in transaction payload. prevents signed transactions from being replayed on other chains.

Commit-reveal schemes

```
contract CommitReveal {
  bytes32 public commitment;
  uint256 public value;

  function commit(bytes32 hashedValue) public {
    commitment = hashedValue;
  }

  function reveal(uint256 _value, bytes32 salt) public {
    assertEq(keccak256(abi.encode(_value, salt)), commitment);
    require(_value == commitment, "Not the commitment");
    value = _value;
  }
}
```

Building & Testing

Foundry

https://book.getfoundry.sh/

- `anvil`: a test node
- `cast`: CLI tool for interacting with EVM chains or data
- `forge`: build & test smart contracts
- `chisel`: Solidity REPL

Forge

Build contracts

```
$ forge build
```

Format code

```
$ forge fmt
```

Run a deploy script

```
$ forge script script/Counter.s.sol:CounterScript --rpc-url $RPC_URL --private-key
$PRIVATE_KEY --broadcast
```

Forge testing

Watch file changes and run tests

```
forge test --watch
```

See stack traces for failed tests

```
forge test -vvv
```

Run a single test

```
forge test --match-test test_SetNumber
```

Run a single test file

```
forge test --match-path test/Counter.t.sol
```

Anvil / Cast

Spawning a local test node from a mainnet fork:

```
$ anvil --fork-url $MAINNET_FORK_URL --fork-block-number 21070000
```

JSON-RPC call to a node:

```
$ cast rpc eth_blockNumber
```

Contract read call:

```
$ cast call $CONTRACT_ADDRESS "balanceOf(address)" $alice
```

Give yourself some free ETH

```
cast rpc anvil_setBalance $ADDRESS 0x1000000000000000000
```

Testing

```
import "forge-std/Test.sol";

contract CounterTest is Test {
  Counter counter;

  function setUp() public {
    counter = new Counter();
  }

  function testIncrement() public {
    counter.increment();
    assertEq(counter.number(), 1);
  }
}
```

Cheatcodes - block number

```solidity
import "forge-std/Test.sol";

contract BlockNumberTest is Test {
  function testBlockNumber() public {
    assertEq(block.number, 1);
    vm.roll(100);
    assertEq(block.number, 100);
  }

  function testBlockTimestamp() public {
    assertEq(block.timestamp, 1);
    vm.roll(100);
    assertEq(block.timestamp, 100);
  }
}
```

Cheatcodes - deal

```
import "forge-std/Test.sol";

contract DealTest is Test {
  address constant alice = address(0x1111);

  function testDeal() public {
    deal(alice, 100);
    assertEq(alice.balance, 100);
  }
}
```

Cheatcodes - prank

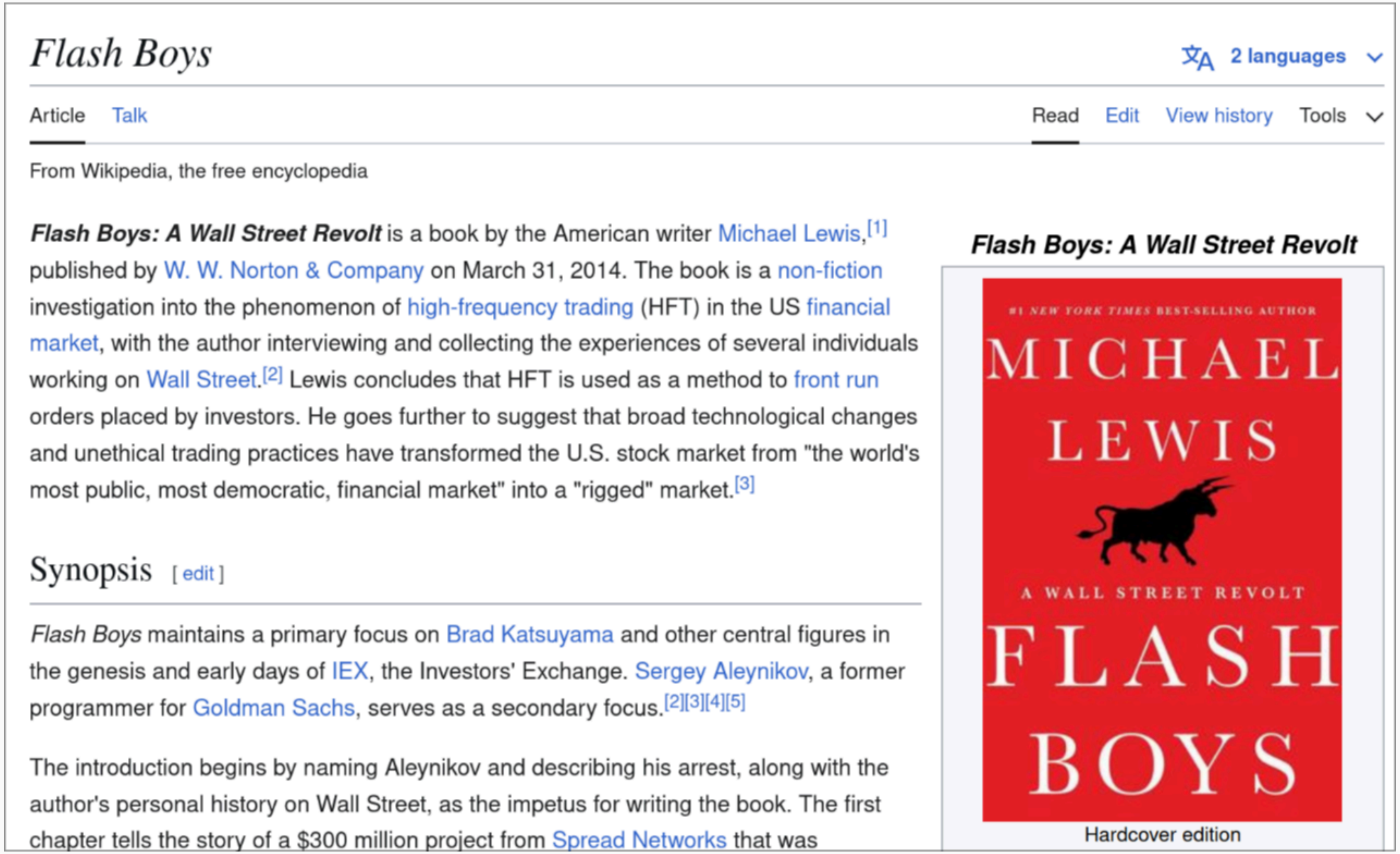
```solidity
import "forge-std/Test.sol";

contract PrankTest is Test {
  address constant alice = address(0x1111);

  function testPrank() public {
    vm.expectRevert(bytes("Not the owner"));
    contract.doSomething();

    vm.startPrank(alice);
    contract.doSomething();
    vm.stopPrank();
  }
}
```

Fork Testing

```solidity
import "forge-std/Test.sol";

contract ForkTest is Test {
  address constant realWallet = address(0x123);

  function testFork() public {
    vm.createSelectFork("mainnet");
    vm.rollFork(100);

    assertEq(realWallet.balance, 1 ether);
  }
}
```

MEV (and other topics)

Flash boys (Wall street)



Flash Boys

2 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

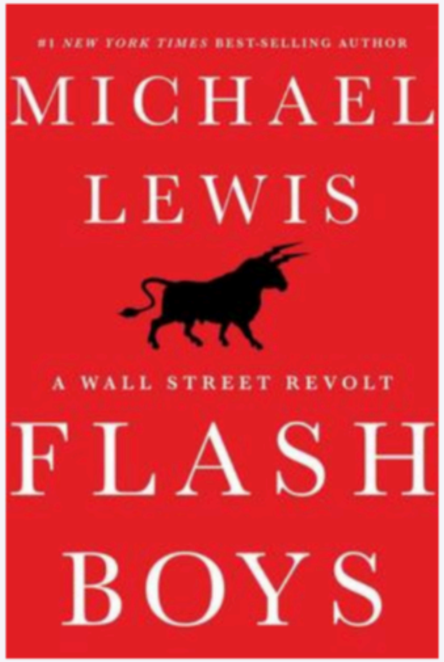
Flash Boys: A Wall Street Revolt is a book by the American writer Michael Lewis,[1] published by W. W. Norton & Company on March 31, 2014. The book is a non-fiction investigation into the phenomenon of high-frequency trading (HFT) in the US financial market, with the author interviewing and collecting the experiences of several individuals working on Wall Street.[2] Lewis concludes that HFT is used as a method to front run orders placed by investors. He goes further to suggest that broad technological changes and unethical trading practices have transformed the U.S. stock market from "the world's most public, most democratic, financial market" into a "rigged" market.[3]

Synopsis [edit]

Flash Boys maintains a primary focus on Brad Katsuyama and other central figures in the genesis and early days of IEX, the Investors' Exchange. Sergey Aleynikov, a former programmer for Goldman Sachs, serves as a secondary focus.[2][3][4][5]

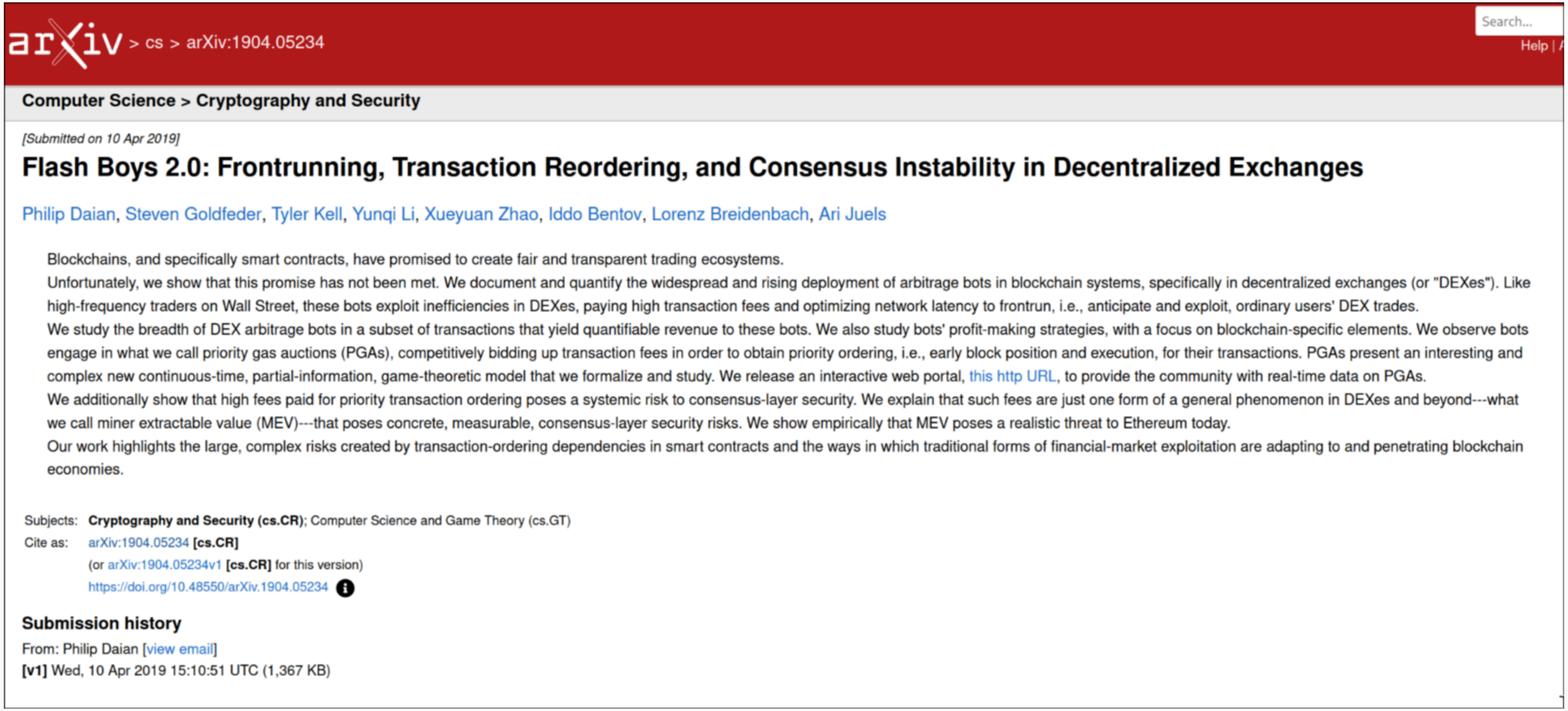
The introduction begins by naming Aleynikov and describing his arrest, along with the author's personal history on Wall Street, as the impetus for writing the book. The first chapter tells the story of a $300 million project from Spread Networks that was

Flash Boys: A Wall Street Revolt



Hardcover edition

Flash boys 2.0



arXiv > cs > arXiv:1904.05234

Search...

Help |

Computer Science > Cryptography and Security

[Submitted on 10 Apr 2019]

Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges

Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, Ari Juels

Blockchains, and specifically smart contracts, have promised to create fair and transparent trading ecosystems.
Unfortunately, we show that this promise has not been met. We document and quantify the widespread and rising deployment of arbitrage bots in blockchain systems, specifically in decentralized exchanges (or "DEXes"). Like high-frequency traders on Wall Street, these bots exploit inefficiencies in DEXes, paying high transaction fees and optimizing network latency to frontrun, i.e., anticipate and exploit, ordinary users' DEX trades.
We study the breadth of DEX arbitrage bots in a subset of transactions that yield quantifiable revenue to these bots. We also study bots' profit-making strategies, with a focus on blockchain-specific elements. We observe bots engage in what we call priority gas auctions (PGAs), competitively bidding up transaction fees in order to obtain priority ordering, i.e., early block position and execution, for their transactions. PGAs present an interesting and complex new continuous-time, partial-information, game-theoretic model that we formalize and study. We release an interactive web portal, this http URL, to provide the community with real-time data on PGAs.
We additionally show that high fees paid for priority transaction ordering poses a systemic risk to consensus-layer security. We explain that such fees are just one form of a general phenomenon in DEXes and beyond---what we call miner extractable value (MEV)---that poses concrete, measurable, consensus-layer security risks. We show empirically that MEV poses a realistic threat to Ethereum today.
Our work highlights the large, complex risks created by transaction-ordering dependencies in smart contracts and the ways in which traditional forms of financial-market exploitation are adapting to and penetrating blockchain economies.

Subjects: **Cryptography and Security (cs.CR)**; Computer Science and Game Theory (cs.GT)
Cite as: arXiv:1904.05234 **[cs.CR]**
(or arXiv:1904.05234v1 **[cs.CR]** for this version)
https://doi.org/10.48550/arXiv.1904.05234

Submission history

From: Philip Daian [view email]
[v1] Wed, 10 Apr 2019 15:10:51 UTC (1,367 KB)

Frontrun attacks

- Alice wants to buy an NFT on sale for 1 ETH

Frontrun attacks

- Alice wants to buy an NFT on sale for 1 ETH
- Alice submits a transaction to the public mempool, offering 60 gwei as gas price

Frontrun attacks

- Alice wants to buy an NFT on sale for 1 ETH
- Alice submits a transaction to the public mempool, offering 60 gwei as gas price
- Bob is scanning the mempool, and detects this attempt

Frontrun attacks

- Alice wants to buy an NFT on sale for 1 ETH
- Alice submits a transaction to the public mempool, offering 60 gwei as gas price
- Bob is scanning the mempool, and detects this attempt
- Bob submits an equivalent transaction, but offering 61 gwei as gas price

Frontrun attacks

- Alice wants to buy an NFT on sale for 1 ETH
- Alice submits a transaction to the public mempool, offering 60 gwei as gas price
- Bob is scanning the mempool, and detects this attempt
- Bob submits an equivalent transaction, but offering 61 gwei as gas price
- Bob’s transaction is preferred, ends up included first

Frontrun attacks

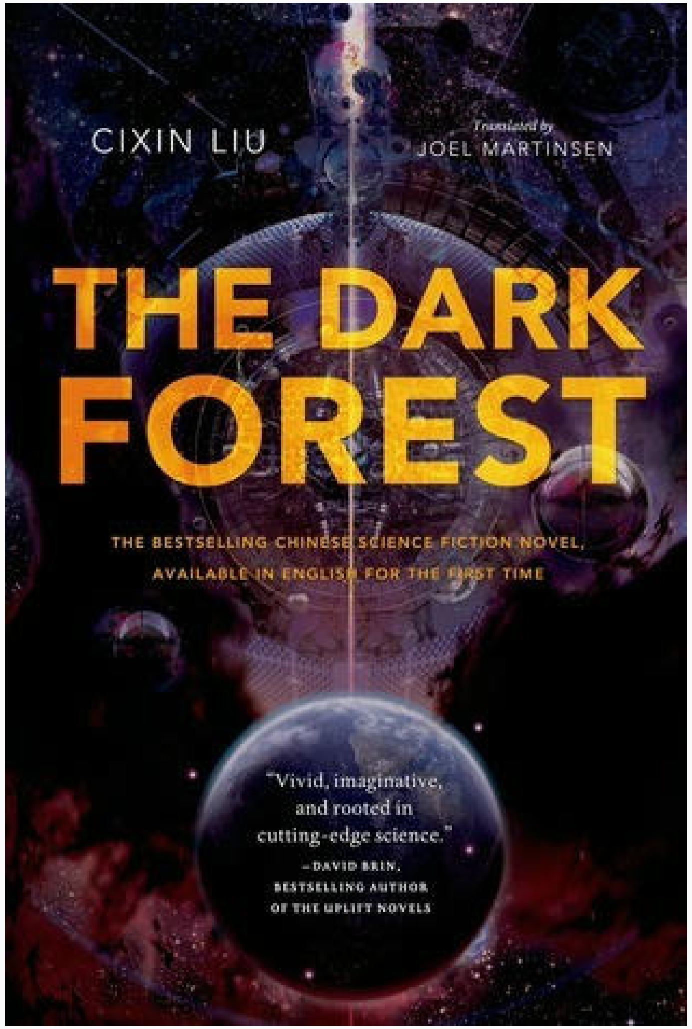
- Alice wants to buy an NFT on sale for 1 ETH
- Alice submits a transaction to the public mempool, offering 60 gwei as gas price
- Bob is scanning the mempool, and detects this attempt
- Bob submits an equivalent transaction, but offering 61 gwei as gas price
- Bob's transaction is preferred, ends up included first
- Alice's transaction is still included, but reverts (sale is no longer available)

Sandwich attacks

- ETH is currently worth $100
- Alice wants to sell 1 ETH at current market price (oversimplification, ask me later)
- Alice submits a tx on Uniswap
- Bob is running a validator node, and detects the attempt
- Bob sees an arbitrage opportunity:
  - Bob sells 10 ETH for $1000 , driving price down to $99
  - Alice tx gets included, ends up receiving only $99
  - Bob buys back 10 ETH for $990

Bob profits $10, alice receives $1 less than expected

The Dark Forest


CIXIN LIU
Translated by
JOEL MARTINSEN
THE DARK FOREST
THE BESTSELLING CHINESE SCIENCE FICTION NOVEL, AVAILABLE IN ENGLISH FOR THE FIRST TIME
"Vivid, imaginative, and rooted in cutting-edge science."
—DAVID BRIN, BESTSELLING AUTHOR OF THE UPLIFT NOVELS

Ethereum is a Dark Forest

Arbitrage bots typically look for specific types of transactions in the mempool [...] and try to frontrun them according to a predetermined algorithm. Generalized frontrunners look for any transaction that they could profitably frontrun by copying it and replacing addresses with their own.

[...]

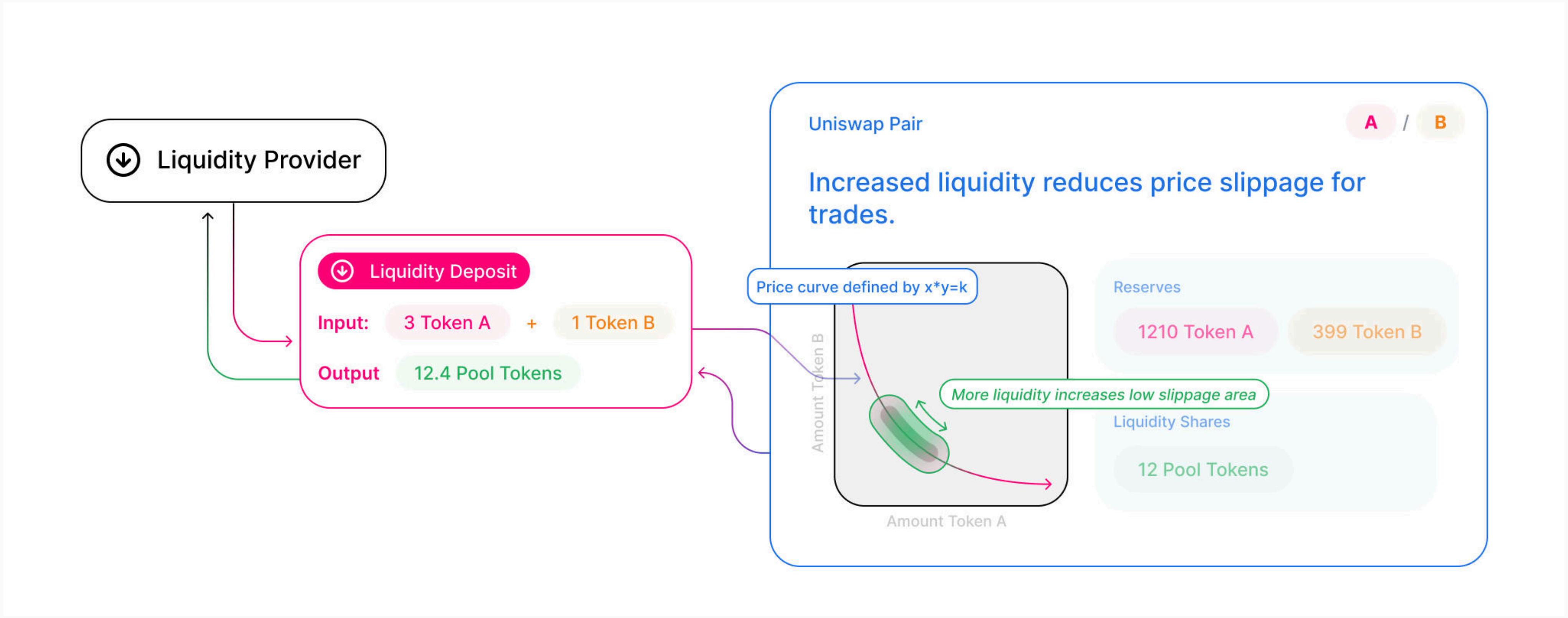
To try to extract the money without alerting the bots, I would need to obfuscate the transaction so that the call to the Uniswap pair couldn't be detected

— **https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest**

Other References (for the curious)

Uniswap V2

$$xy = k$$



Flashbots

Private mempool for MEV and white-hat arbitrage

Vyper

A python-based EVM language. Alternative to Solidity

(other more esoteric languages: YUL, Huff, Fe)

Thank you!