Flash Boys 2.1.0

An Evolution of MEV in Decentralized State Machines

by

Brendan McCaffrey
B.A.S. Computer Science † B.S. Finance

School of Engineering & Applied Sciences
Wharton School of Business
The University of Pennsylvania

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Computer Science

December 18, 2021

Keywords: Miner Extractable Value, Maximal Extractable Value, Blockchain, Distributed State Machines, Arbitrage.

In accordance with the requirements of the degree of Bachelor of Computer Science in the School of Engineering & Applied Sciences
Wharton School of Business, I present the following thesis entitled,

Flash Boys 2.1.0
An Evolution of MEV in Decentralized State Machines

This work was performed under the supervision of Professor Tal Robin. I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at The University of Pennsylvania or any other institution.

Brendan McCaffrey

Abstract

The rise of distributed state machines have led to an increased use of decentralized markets. As the adoption of smart contract platforms continues to grow, the implications of on-chain arbitrage dynamics are increasingly important.

In this paper, I explore the concept of Maximal Extractable Value, and investigate the evolution of the space since the term was coined by Daian et al. in 2019.

I identify three categories of MEV: Single-Domain Atomic MEV, Single-Domain Non-Atomic MEV (also known as long-tail MEV), and Cross-Domain MEV (which is inherently non-atomic). I show that Single-Domain Non-Atomic MEV strategies have yielded over $10 million USD of extracted value on the Ethereum network in just the last six months. This paper also explores the future potential for cross-domain arbitrage in a multi-chain economic ecosystem with a high level evaluation of the key logic.

My work aims to highlight the importance of MEV as a concept within distributed systems, and also includes a brief discussion of the ethical implications of the subject.

Contents

Chapter 1

Introduction

The Blockchain

A blockchain is a decentralized digital ledger that exists across a distributed network of computer systems. Although many credit Bitcoin with the invention of the blockchain in 2008, the timeline begins in the 20^{th} century (Chohan, 2017).

- 1975: The Two Generals Problem is proven unsolvable.
- 1982: Byzantine Generals Problem (Byzantine Fault Tolerance, BFT) receives several algorithmic solutions.
- 1991: Stuart Haber and W Scott Stornetta describe, for the first time, a cryptographically-secured chain of blocks.
- 1998: Computer scientist Nick Szabo begins his work on 'Bit Gold', a decentralized digital currency.
- 2000: Stefan Konst publishes his theory of cryptographic secured chains, plus ideas for implementation.
- 2004: Hal Finney delivers the first reusable Proof of Work system.
- 2008: Developer(s) under the psuedonym Satoshi Nakamoto release the whitepaper, *Bitcoin: A Peer-to-Peer Electronic Cash System* (Nakamoto, 2008).
- 2009: Hal Finney downloads the Bitcoin software on it's release date, and receives the first Bitcoin transaction - 10 Bitcoins, from Satoshi Nakamoto ("Here's The Problem with the New Theory That A Japanese Math Professor Is The Inventor of Bitcoin", 2015).

The Bitcoin whitepaper was revolutionary for several reasons. Firstly, in the wake of a global recession caused by fiat monetary policy, it is an anonymously-created revolution against fiat banking standards. Although Satoshi Nakamoto didn't *invent* blockchain technology in 2008,

he/she/they introduced the world to a new paradigm: money controlled by math, not people.

Bitcoin implements Proof-of-Work as a byzantine fault tolerance (BFT) protocol, ensuring consensus under the assumption that an attacker has less than 50% of hashing power.

definition. *Byzantine Fault Tolerance (BFT)* is a mechanism that enables a decentralized, trustless network to function in the presence of malfunctioning or malicious actors.
definition. *Proof-of-Work (PoW)* is a form of cryptographic proof in which one party proves to others that a certain amount of a specific computational effort has been expended.

The Bitcoin network is a PoW ledger that contains a full history of transactions independently verified by mining node operators. The introduction of Bitcoin is considered the first successful implementation of a cryptographically-secured, distributed ledger, and it quickly invoked a period of steady growth in the idea of a decentralized systems.

- 2010: Laszlo Hanyecz pays 10,000 bitcoin for two Papa John's pizzas, considered the worst trade ever.
- 2011: Litecoin is launched, often recognized as the most prominent Bitcoin fork.
- 2013: Dogecoin is launched as a "joke" payments system.
- 2013: The Ethereum whitepaper is released by Vitalik Buterin.
- 2015: The Ethereum mainnet is launched.

The Distributed State Machine

Six years after the launch of Bitcoin, Vitalik Buterin introduced what would become the next major innovation in distributed, decentralized technology – Smart Contracts.

definition. A *smart contract* is a self-executing computer program stored on a blockchain that runs when predetermined conditions are met.

Ethereum's genesis block effectively commenced the era of decentralized Turing-complete systems, replacing the *distributed ledger* with the *distributed state machine* (Buterin et al., 2013).

The ability to execute program logic on the blockchain presented a paradigm shift in blockchain technology. Unlike Bitcoin, which had a sole capability

of storing and transacting Bitcoins, Ethereum was capable of performing arbitrary code executions on what is, essentially, a decentralized computer.

Internet Money and Decentralized Finance

The following years were an essential period of development for decentralized technology, primarily focused in decentralizing the financial sector (DeFi). In 2017, new projects being built on Ethereum avoided the traditional process of fundraising from institutional capital, instead offering their own tokens to the public in exchange for ETH - this commenced the Initial Coin Offering (ICO) era. The coins introduced in the ICO era were ERC-20 tokens. ERC-20 is a technical standard for fungible tokens that live on the Ethereum Virtual Machine (EVM). Below are some of the most notable DeFi projects from the ICO era:

- Aave – a lending and borrowing protocol
- Synthetix (previously known as Havven) – a liquidity protocol for derivatives
- REN (previously Republic Protocol) – a protocol for providing access to inter-blockchain liquidity
- Kyber Network – an on-chain liquidity protocol
- 0x – an open protocol that enables the peer-to-peer exchange of assets
- Bancor – an on-chain liquidity protocol

After the ICO mania was over and the bear market kicked in, DeFi seemed, from the outside, to experience a relatively quiet period - but this was far from the truth.

Behind the scenes, major DeFi protocols were being developed on Ethereum. Additionally, talented individuals were leaving their high-profile positions in the traditional private sector to develop superior smart-contract platforms, focusing on speed, security, interoperability, and, of course, decentralization.

Non-Fungible Tokens

In 2018, an Ethereum Improvement Proposal, EIP-721, introduced a technical standard, ERC-721, for Non-Fungible Tokens (NFTs). The difference between fungible and non-fungible tokens can be illustrated as follows.

Imagine you are in a group of 5 people, $P_1, ..., P_5$ where each person owns one item from each of the following two sets.

- **Set 1** contains 5 1-ounce gold bars.

- **Set 2** contains 5 art pieces from the famous artist Pablo Picasso.

The first set is fungible - P_1 can offer \$1 to P_2 to exchange gold bars, and expect him to accept. The gold bars are identical, so P_2 ultimately gets a benefit of \$1.

With set 2, however, P_1 cannot expect P_2 to accept \$1 to exchange art pieces, as there is a significant difference between Picasso's works. The art pieces are *non-fungible*, as no two items in Set 2 are exactly the same.

The introduction of the ERC-721 standard allowed developers to create non-fungible tokens on the blockchain, which have since become an integral part of the decentralized world.

The current state of decentralized technology is an ecosystem of various blockchains and smart contract platforms - with Bitcoin as the prodominant store of value, and Ethereum as the primary data layer. Protocols ranging from decentralized exchanges, lending and borrowing protocols, and gaming applications are being developed and used on a collection smart contract networks.

What is Miner/Maximal Extractable Value?

MEV refers to the financial value that can be extracted by miners (or bot operators) through the inclusion, censorship, and reordering of transactions within a block.

For a simplified illustration of how MEV manifests, consider the following scenarios (*MEV Miner*, n.d.):

You are a blockchain miner, and you have been selected to construct and propose the next block.

Scenario 1: You can fit 3 transactions in your block, and there are five transaction (tx) requests, all transfers.

1. Tx 1: offers \$5.00 in fees for a transfer.
2. Tx 2: offers \$4.20 in fees for a transfer.
3. Tx 3: offers \$1.15 in fees for a transfer.
4. Tx 4: offers \$6.50 in fees for a transfer.
5. Tx 5: offers \$5.00 in fees for a transfer.

If you are rational, you would include Tx 4, Tx 1, and Tx 5, as this combination maximizes your revenue for the block: \$16.50.

Scenario 2: You can fit 3 transactions in your block, and there are five transaction (tx) requests.

1. Tx 1: offers $17.00 in fees for a transfer.
2. Tx 2: offers $6.70 in fees for a transfer.
3. Tx 3: offers $13.10 in fees for an arbitrage yielding $900 profit.
4. Tx 4: offers $4.50 in fees for an arbitrage yielding $20 profit.
5. Tx 5: offers $12.00 in fees for an arbitrage yielding $18 profit.

The simple action to take is to simply include Tx 3, Tx 5, and Tx 1, as this combination seems to maximize your revenue for the block: $42.10. However, as a miner you have an additional possibility: replace the "from" and "to" addresses of arbitrage transactions 3 and 4 from the respective arbitrageurs' addresses to your own address, essentially executing the arbitrage for yourself rather than on behalf of the arbitrageurs. This action truly maximizes your revenue for the block: $937.00.

Notably, the MEV actor does not necessarily have to be a miner. In this scenario (2), an arbitrageur with a low-latency view of incoming transactions can also extract value. For example, replicate Tx 3 (replacing "from" and "to" addresses to your own) with a higher gas fee for the miner, $13.20. If the miner is not running arbitrage software, or is content with the proportion of arbitrage profit paid in fees, they will include your transaction over Tx 3 in the block.

As you can see, *miner extractable value* represents an upper-bound with actors P being miners, the most privileged actors in a PoW network. The transition to *maximal* generalizes to non-PoW chains (and even non-blockchain domains), and allows for the participation of third-party arbitrageurs (Obadia et al., 2021).

Formal Definitions

Now that we have a general sense of MEV, let us iterate the formal definitions of *reachable states*, *extractable value*, and *single-domain maximal extractable value*, provided by (Obadia et al., 2021).

They define a set of actions A_P which represent all valid actions available to player P. They say a sequence of actions $a_1...a_n \in A_P$ if and only if $\forall i, a_i \in A$ (where A is the global set of all actions for all players in the protocol), and $\forall i, j, a_i \neq a_j$ (each action is allowed and unique).

For each sequence of valid actions, there exists a state s' that will be the

state of the domain after each action is applied sequentially to the current state s.

definition. *Reachable States*
All s's together represent the *reachable states* of the domain under the influence of player P (S'_P), given the current state s and set of actions A_P. They define:

$$a \underset{A}{\rightarrow} S'_P$$

such that $S'_P = \{s' | s \underset{a_1 ... a_n \in A_P}{\longrightarrow} s'\}$

definition. *Extractable Value*
The *extractable value* (ev_i) in domain i, after executing a sequence of actions $a_1, ..., a_n$ on an initial state s, is defined as:

$$ev_i(P, s, a_1 ... a_n) = b_i(s', P) - b_i(s, P)$$

where $s \rightarrow_{a_1 ... a_n} s'$ and $b_i(s, P)$ is the balance of a player P at a state s.

definition. *Single-Domain Maximal Extractable Value*
Maximal Extractable Value (MEV) is simply the maximum of the above definition over all valid sequences of actions for player P. We have:

$$mev_i^j(P, s) = \max_{a_i ... a_n \in A_j} \{ev_i(P, s, a_1 ... a_n)\}$$

The superscript j binds the action set available to the player (in which domains the player can act) to those that exist in domain j. mev_i^j can be interpreted as the MEV that can be extracted from a balance change in domain i, given actions in domain j.

Single-Domain Maximal Extractable Value is simply the case wherein $i = j$:

$$mev_i^i(P, s) = \max_{a_i ... a_n \in A_i} \{ev_i(P, s, a_1 ... a_n)\}$$

Goals and Scope

At first, the concept of MEV may seem to be a complex negative externality unique to blockchain systems. However, I propose it is simply a new manifestation of market arbitrage, an idea dating back at least as far as the imperial period (Poitras, 2021). Although the concept of value-extraction is not unique to the blockchain, I claim the possibility of democratizing this extracted value *is* unique to decentralized systems.

The goal of this paper is to introduce and investigate the concept of MEV in blockchain systems, highlighting the evolution of the space since the publication of the first academic paper on the subject (Daian et al., 2019). It will include an exploration of MEV defined by each of the following categories:

1. Atomic, Single-Domain Maximal Extractable Value
2. Non-Atomic, Single Domain Maximal Extractable Value
3. Cross-Domain Maximal Extractable Value (inherently non-atomic)

Chapter 2

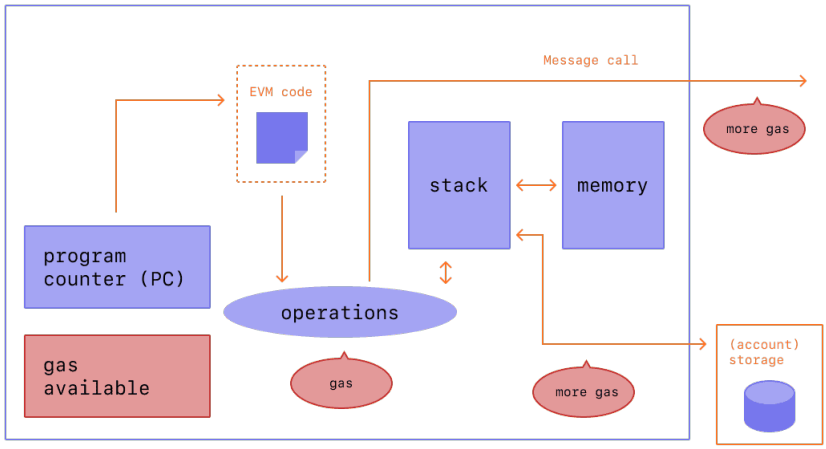
Ethereum Virtual Machine

A high-level understanding of the architecture of a decentralized state machine is a prerequisite to fully understanding MEV.

As Ethereum set the precedent for such technology, and remains the dominant smart contract platform, we will explore how the Ethereum Virtual Machine (EVM) works, and identify several key terms.

The EVM executes as a stack machine. During execution, the EVM maintains a transient memory (as a word-addressed byte array), which does not persist between transactions. Contracts, however, do contain a Merkle Patricia storage trie (as a word-addressable word array), associated with the account in question and part of the global state. Compiled smart contract bytecode executes as a number of EVM opcodes, which perform standard stack operations like XOR, AND, ADD, SUB, etc.

Below is an illustration of the EVM (Hildenbrandt et al., 2018).



Key Terms

- Transaction: An action initiated by an externally-owned account (a human, not a smart contract).
- Mempool: Short for memory pool, it's a list of all transaction requests Ethereum miners' heard about that have not yet been committed to the blockchain in a block.
- Gas: The fuel that allows Ethereum to operate, in the same way that a car needs gasoline to run. The unit for gas is Gwei, which is equivalent to 1×10^{-9} ETH.
- Transaction Gas Limit: The maximum amount of gas units you are willing to consume on a transaction. Transactions with more complex code executions require a higher gas limit.
- Block Gas Limit: The upper bound on block size, in terms of gas units.

Mining

MEV operators rely on a great understanding of "the rules of the game", the block mining procedure. This is because many instances of MEV simply boil down to precisely positioning your transaction relative to surrounding transactions.

Mining is the process of creating a block of sequential transactions to be appended to the blockchain. Ethereum currently uses PoW as a consensus mechanism (similar to Bitcoin), but is expected to transition to Proof-of-Stake in early 2022 (this transition isn't significant for the purposes of this paper). On Ethereum, a new block is mined roughly every 12-14 seconds (*Ethereum Documentation*, n.d.).

The flow of submitting a transaction to be included on the ledger is as follows:

1. A user writes and signs a transaction request with the private key of some account, and broadcasts the transaction request to the network.
2. Upon hearing the transaction request, each node in the network adds the request to their local mempool.
3. At some point, a mining node aggregates several dozen or hundred transaction requests into a potential block, in a way that maximizes the transaction fees they earn while staying under the block gas limit. The mining node then:
   (a) Verifies the validity of each transaction request, and then executes the code of the request, sequentially altering the state of their local copy of the EVM. The miner awards the transaction fee for each such transaction request to their own account.

   (b) Begins the process of producing the Proof-of-Work "certificate of legitimacy" for the potential block, once all transaction requests in the block have been verified and executed on the local EVM copy.

4. Upon completing a block which includes our specific transaction request, the miner broadcasts the completed block to the network, which includes the certificate and a checksum of the claimed new EVM state.

5. Other nodes hear about the broadcasted block, verify the certificate, execute all transactions on the block themselves, and verify the checksum of their new EVM state.

6. Each node removes all transactions in the new block from their local mempool of unfulfilled transaction requests.

Development

The overwhelming majority of Ethereum-deployed code is written in the dominant smart contract language, Solidity. Solidity is a statically-typed, object-oriented programming language inspired by JavaScript, C++, Eiffel, and Python. It was initially proposed by Nathan Wood in 2014, and later developed by the Ethereum project's Solidity team, led by Christian Reitwiessner. Solidity files have a .sol extension.

It is important to note that smart contract languages like Solidity cannot be executed on the EVM directly. Instead, they are compiled to low-level machine language (Opcodes) before execution.

Clients

"Node" refers to a running piece of client software. A client is an implementation of Ethereum that verifies all transactions in each block, keeping the network secure and the data accurate. There are currently several client implementations, Go-Ethereum (Geth) being the most popular. The client is an essential piece of the EVM architecture, and an essential piece of MEV, serving as "customs" to and from the Ethereum chain. For latency purposes, all of the top MEV competitors either run their own client, or co-locate with a node.

For the majority of Ethereum's life, the client software only accepted transactions adhering to the strict specifications as described above. If someone wanted their transaction to be included, they could express their willingness through the transaction fee. To understand how MEV developed in these conditions, consider the following:

Imagine a "pure profit" arbitrage opportunity of value p, arises with a price misalignment between two on-chain exchanges. User A, an experienced arbitrageur, sends transaction A to perform the arbitrage. A second later, User B, also experienced in MEV, sees both 1) the same opportunity and 2) transaction A. User B, eager to secure profit p, sets transaction B's gas price slightly higher than transaction A's. Game theory ensues, and the arbitrageurs take part in what's known as *priority gas auctions* (PGAs), bidding gas fees up to the value of the opportunity, p.

In this dynamic, the "winner" of the gas auction executes the arbitrage, but yields a profit far below the value of the arbitrage opportunity. Likewise, the "loser" of the auction ends up paying gas fees without yielding any arbitrage profits.

The rise of PGAs presented negative externalities to the Ethereum network. In addition to increased congestion on both the network (P2P message propagation) and the chain layer (block space limited by gas limit), consensus stability was at risk.

Introducing MEV-Geth

On November 23, 2020, researcher Alex Obadia published an article addressing the state of MEV. In it, Obadia introduced a company called Flashbots and its flasgship product, a fork of the Go-Ethereum client called MEV-Geth (Obadia, 2020).

> *...we find ourselves at a critical junction between alternative futures for Ethereum. A series of events in the past 6 months have lead usage of the network to reach a tipping point...*
>
> *Flashbots is a research and development organization formed to mitigate the negative externalities and existential risks posed by MEV to smart-contract blockchains. We propose a permissionless, transparent, and fair ecosystem for MEV extraction to preserve the ideals of Ethereum...*
>
> *MEV-Geth is our initial effort to Democratize Extraction. It is an upgrade to the go-ethereum client to enable a sealed-bid block space auction mechanism for communicating transaction order preference.*

The introduction of MEV-Geth was dramatic development in the evolution of MEV. This client software introduced a private relay - a channel to communicate with the miner while "sidestepping" the public mempool. If the block producer was using the MEV-Geth client, arbitrageurs could

now send a transaction with fee \$0, and *directly bribe* the miner to include the transaction *as desired*.

Imagine the same "pure profit" opportunity as before...

User A, an experienced arbitrageur, sends arbitrage transaction A to a private endpoint. Transaction A includes a fee of 0, but user A offers bribe B_a to the miner if his arbitrage succeeds. User B also sees the opportunity, but does not see transaction A in the local mempool (because it was submitted privately to the miner). User B also submits transaction B to a private endpoint, with fee 0, and bribe B_b to the miner if his arbitrage succeeds. The miner includes transaction A if $B_a > B_b$, and transaction B if $B_a < B_b$ (latency / time of submission for A and B also plays a factor, but less significant here).

The use of MEV-Geth eliminates priority gas auctions (PGAs) (Daian et al., 2019), and replaces the dynamic with a first-price, sealed-bid auction. This is significant as the loser of the arbitrage auctions, under MEV-Geth, does not lose the capital devoted to his/her bid. Essentially, the arbitrage is much closer to "risk-free".

Chapter 3

Atomic, Single-Domain MEV

The overwhelming majority of focus in the MEV space is in the category that can be described as atomic, single-domain MEV. For obvious reasons, instances of MEV in this category almost always utilize/involve DeFi platforms.

definition. *Atomic* in the context of MEV implies the entire value-extracting trade was performed in a single block.
definition. A *Domain* is a self-contained system with a globally shared state. This state is mutated by various players through actions (often referred to as "transactions"), that execute in a shared execution environment's semantics (Obadia et al., 2021).
definition. *Decentralized Finance (DeFi)* is a blockchain-based form of finance that does not rely on central financial intermediaries such as brokerages, exchanges, or banks to offer traditional financial instruments, and instead utilizes smart contracts.

Unlike a user in the traditional financial systems, DeFi users have complete custody of their assets. From 2017 to 2021, the total value locked (TVL) in DeFi exploded from $1.62MM to a high of $157B – a period of growth from which few similarities can be drawn.

Atomicity in arbitrage transactions is desirable for several reasons. For one, the gains from the arbitrage are realized almost instantly, producing unmatched time-measured returns. Secondly, and perhaps most notably, is that atomic arbitrages can be executed virtually risk free. To understand how, consider the following piece of code.

```
contract MEVBot {
    // Constructor, gobal variables, ect. omitted

    // Function assumes starting token is ETH
    function doArb(address[] pools, uint minerBribe) payable {
        preBalance = address(this).balance;
        // do arbitrage here
        require(address(this).balance > preBalance);
        block.coinbase.transfer(minerBribe);
    }
}
```

As you can see, the function `doArb()` includes a `require()` statement, which will cause the entire transaction to revert if the contained boolean is false. The arbitraguer can simulate the transaction on the current EVM state prior to submitting the transaction. But even if the simulation yields success and the actual execution reverts, the attempted arbitrage trades are never executed on-chain.

The DEX

The most fundamental DeFi product is a decentralized exchange (DEX). A decentralized exchanged is an exchange platform wherein none of {capital deposits, order broadcasting, order matching, exchange fo tokens} are dependent on a centralized party/authority.

Today's premier DEXes utilize an automated market maker (AMM). The most prolific AMM implementation utilizes the simple yet innovative *constant product formula* (*On Path Independence*, 2017), credit to Martin Köppelmann.

$$x \times y = k$$

In this equation, x and y represent the reserves of two tokens A and B. The constant product formula says that in order to purchase some amount of token A, one must pay a proportional amount of token B to maintain the constant product k (before fees). If we let p represent the price of token B denominated in token A, we get:

$$x = \sqrt{k \times p}$$

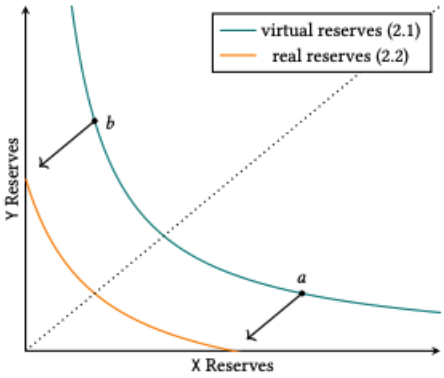
$$y = \sqrt{k/p}$$

Anyone may add liquidity into a DEX pool by depositing equal quantities of each asset at the currently-trading ratio, and in return, they get rewards in proportion of the fees generated multiplied by their weight in the liquidity pool.

It's important to note that k may change, with liquidity deposits and withdrawals, but never with swaps.

It's worth noting that there have been further innovations with respect to AMM technology. Most notably, Uniswap V3's introduction of *concentrated liquidity*. This AMM implementation essentially maps real reserves to a larger virtual reserves. For this implementation, the pool tracks two values: *liquidity (L)* and *sqrtPrice* ($\sqrt{P}$) (Adams, Zinsmeister, Salem, Keefer, & Robinson, 2021).

$$L = \frac{\Delta y}{\Delta \sqrt{P}}$$



The main benefit of concentrated liquidity is twofold. First, liquidity providers are offered greater efficiency of funds, as a liquidity position now includes price bounds, declared by the depositor, at which his capital is effective. Once the trading price of the asset pair moves beyond this price range, the liquidity position becomes dormant, earning no fees. Second, concentrated liquidity offers the DEX user more efficient trade execution (lower slippage), as the slippage is calculated by from the virtual reserves rather than the real reserves.

The majority of trade volume is still routed through DEXes implementing the original constant product formula, although Uniswap V3 is one of the top on-chain exchanges, with $> 5\%$ market share.

Lending Protocols

In addition to the DEX, a common DeFi platform are decentralized lending and borrowing protocols. Platforms like Maker, Aave, and Compound allow users to deposit collateral (ETH, USDC, etc.) and borrow a desired token up to a specified loan-to-value (LTV) ratio.

For example, Aave's requried LTV on a DAI-collateralized loan is 75%,

and the liquidation threshold is 80%. This means a user can borrow up to 75% of their posted DAI value in the desired token, and are at risk of being liquidated if their LTV reaches 80%.

On the other side of the relationship, lenders can earn yield by allowing lending platforms to use their capital for loans.

Flash Loans

A *flash loan* is a type of under-collateralized atomic loan wherein a user can borrow a large amount of capital (sometimes 7 or 8 figures) for an extremely small fee (most often 0.09%). The reason why the fees can be so small is because the lender faces no counter-party risk. A flash loan must be repayed in the same block which it was lended - a failure to repay the flash loan causes a revert, wherein the initial loan is never executed (this is the same concept that makes atomic arbitrages risk-free).

A flash loan essentially provides an arbitrageur with unbounded capital to perform arbitrages or exploits. This DeFi innovation is highly controversial, as it has been an integral tool for many attacks, including a $\$130mm$ exploit on Cream.finance (2, 2021). On the other hand, however, it lowers the barriers to entry for arbitrageurs who can't afford to put up the necessary capital.

One of the most recent innovations in flash loans is the *flash mint*. Rather than borrowing the capital for a single-block duration, the user can *mint* tokens. Free flash-minting (zero fees) is currently available for the token WETH10.

An Overview of Atomic, Single-Domain MEV Manifestations

Below is a general overview of the replicable strategies that have developed to date. We will analyze each strategy with an example.

- The Sandwich: Arbitrageur "sandwiches" the victim's transaction with both a front-run and a back-run.
  - The Front-Run: Arbitrageur places his/her transaction directly before the victim's transaction.
  - The Back-Run: Arbitrageur places his/her transaction directly after the observed transaction.
- The Liquidity Provision Sandwich: Arbitrageur "sandwiches" an observed transaction with the entrance and exit of a liquidity position in the relevant asset pair.
- Cyclical Price Arbitrage: Arbitrageur swaps mispriced assets across

DEXes, ending the transaction with a larger amount of the starting asset.

- Liquidations: Arbitrageur liquidates an under-collateralized debt position for a reward.
- Attacking the Arbitrageur: Salmonella and beyond.

What Makes an MEV Bot Competitive?

Before we dive into each of the above strategies, it's important to note that an understanding of a strategy is not sufficient for being successful in the MEV space. As with any other trading/arbitrage strategy, MEV capture is extremely competitive. Here are a few points of differentiation by which "good" MEV bots are separated from the "bad".

- **Latency**

  As MEV is often regarded as the on-chain variant of high frequency trading, it's clear that latency is a primary factor in the success of an MEV operation. There are primarily two sources of latency in an MEV operation: connection to the network, and local code run-time.

  Minimizing latency in data propagation to and from the network is imperative for obvious reasons: the first person to receive data of a new block will be the first person able to begin the search for arbitrage opportunity in the next block. To decrease latency to and from the network, MEV bot operators integrate with a node (miner or validator, depending on the network's consensus mechanism). If you can't run a node (due to capital requirements, etc.) the next best thing is to co-locate with one. Proximity to the network can be responsible for latency edges north of 500 ms.

  As nearly all of the search process happens locally, decreasing the run-time of your code is also significant from a competitiveness perspective. Each strategy has a unique set of restraints by which logic run-time can be optimized, these will be discussed in the respective sections.

- **Gas Golfing**

  Gas golfing refers to the practice of minimizing the gas usage needed for the arbitrage logic to be executed on-chain - this is a major factor in determining the success of an MEV strategy. To understand how, consider the following: MEV bots A and B each find a price arbitrage between Uniswap and Sushiswap, where $ETH/USDC$ is quoted at 4100 and 4050, respectively. The arbitrage can be executed (at a

certain quantity) for a maximum of 0.2 ETH profit. Bot A submits an arbitrage execution that uses $450,000$ gas, and offers the miner 400 Gwei: a total cost of .18 ETH. Bot B, however, has an execution script that only uses $200,000$ gas units. With this advantage, bot B can win the auction with a miner bribe of 410 Gwei, while maintaining a total cost of .082 ETH - significantly lower than A's cost. Bot B essentially has a higher margin to bribe the miner for the same arbitrage opportunities as his/her competitors!

Most gas optimizations come from a deep understanding of contract interactions. At a high level, the goal is to interact with the EVM (or respective chain) *as little as possible*.

Another gas-optimization strategy is to minimize the storage required for your address by having a large prefix of 0's. This is why many arbitrageurs mine vanity addresses like `0x0000000000000008bAf6Fe3676B23Af7CDBdBC98`.

Most successful MEV bots are careful to write gas-efficient code in their execution contracts. If your arbitrage uses less gas than the competitors', you have a larger margin to bribe the miner (Gwei per unit gas) at which your transaction is still profitable.

- **Search Space**

  The definition of your search space is one of the most important factors determining the success of an MEV strategy. For one, it can supplement overall latency, as more focused search space means less time searching.

  Additionally, however, having a niche in the overall search space essentially means less competition. For example, in price arbitrage, quickly configuring your script to scan DEXes for the mispricing of just a few new, high volume meme-tokens, can give a temporary (but valuable) competitive edge. Likewise, in liquidations, omitting Aave and Compound (highly competitive), for example, saves time while also providing the benefit of being "a big fish in a small pond".

The Sandwich Attack

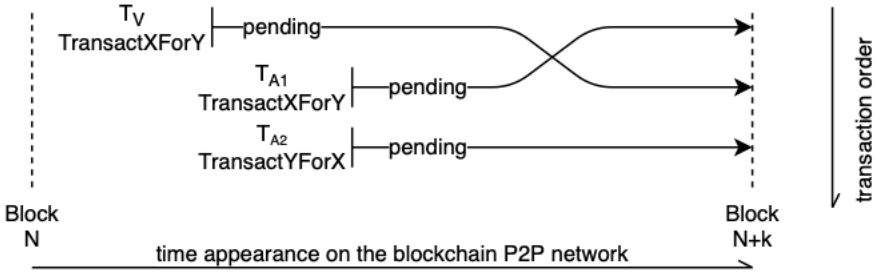
The following is a breakdown of how a sandwich attack occurs (Zhou, Qin, Torres, Le, & Gervais, 2021).

1. Detects a victim's transaction, which is, WLOG, a request to swap asset X for Y.

2. Front run the victim with some action, taking long position in the same direction as (1) (or taking short position in opposite direction). For example, swap asset X for Y.

3. Let the victim's transaction execute - this creates an increase in the price of Y denominated in X. Because of (2), the victim's order is executed at a worse price than it would otherwise.

4. Back run the victim with the action opposite to (2), closing the position for a risk-free arbitrage. For example, swap asset Y for X.



Using the constant product formula, it's clear how the arbitrageur generates profit at the victim's expense.

Let the quantities of transaction (2) be a swap of a amount of asset x for b amount of asset y. According to the constant product formula, we know that the reserves after this trade execution will be

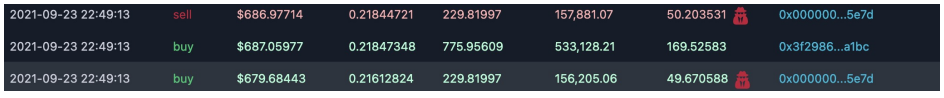
$$(x + a) \times (y - b) = k$$

Thus, there is an increase in the price of y, equal to

$$\Delta \frac{x}{y} = \frac{x + a}{y - b} - \frac{x}{y}$$

This difference roughly equates to the per-unit profit made by the arbitrageur (minus fees). I say "roughly" because the arbitrageur's transaction (3) is also exposed to slippage, depending on size. Determining this slippage for exact calculation of arbitrageur profit is unnecessary for the explanation of this concept, but can be calculated using the same constant product formula, reserve quantities, and trade quantities.

Below is a screenshot of an example of a sandwich attack - the sequence of execution is bottom → top.



As you can see, the arbitrageur makes off with 0.53 ETH profit.

The Liquidity Provision Sandwich

The Liquidity Provision Sandwich, also known as the Just-In-Time Liquidity, closely resembles the Sandwich attack. However, rather than front-running and back-running the observed transaction with trades, the arbitrageur "sandwiches" the observed transaction with an LP position to earn fees on the trade.

It's important to note here that the LP Sandwich does not actually harm the user behind the "victim" transaction. In fact, the "victim"'s transaction actually executes at a *better* price than if the arbitrageur were to not be involved. This is because there is a greater amount of liquidity at the moment the observed transaction is executed, and thus, lower slippage. Of course, this extra value isn't coming out of thin air. The losing side of this type of arbitrage are the *passive liquidity providers* - those who stake liquidity in the same asset pool without an MEV bot to actively manage it.

This strategy is only profitable on AMMs that implement the concentrated liquidity model. Because the arbitrageur is intercepting a particular trade, the price of execution can be pre-determined. This allows the arbitrageur to set the price bounds on the liquidity position such that he/she represents close to 100% of the liquidity in the range of the "victim" trade's execution price.

Additionally, the LP sandwich is often only profitable on extremely large trade sizes, particularly on Ethereum. This is because the arbitrageur's profit margin is lessened by the transaction fees required to enter and exit an LP position, which can be a couple hundred dollars on the Ethereum network, depending on network congestion.

Currently, this MEV strategy is dominated by a single address:
`0xa57bd00134b2850b2a1c55860c9e9ea100fdd6cf`.
Since the launch of Uniswap V3 in May 2021, it's estimated that `0xa57` generated approximately $\$1,123,000$ in profit (Analytics, 2021)

For an example, we can take a look at `0xa57`'s most profitable arbitrage. Ironically, the "victim" transaction was a complex price arbitrage that utilized a flash loan.

1. Transaction Hash: `0x20b7e1068e3fe1ea05e9fb0186aa2ac3154931d5bcf76d98f99fa1253f14e0af`
   Bot `0xa57` Adds liquidity to UniswapV3's USDC/WETH pool.

12,105,953 USDC and 246.44 WETH.

2. Transaction Hash: `0x5a652390cf9d683475052382a3d09ecfc27904 62665b1ebbb8756bdfc5c8d1bc`
   User `0x5aa` (the "victim") performs a complex series of swaps, including a swap of 281.57 WETH for 599,340 USDC through the Uniswap V3 USDC/WETH pool.

3. Transaction Hash: `0x277a2e83babc335be12c193811f818a9050d7 40fc6fe15a79e970d8d711fb586`
   Bot `0xa57` Removes liquidity from UniswapV3's USDC/WETH pool. 11,644,026 USDC and 401.01 WETH.

The approximate marked-to-market profit from this transaction (to `0xa57`) is $\$8,518$ USD. Funny side note, `0x5aa` also profited here.

Cyclical Price Arbitrage

This category of MEV is perhaps the most simple, conceptually: If two exchanges have different quote prices for the same asset pair, buy on the "cheaper" exchange and sell on the "expensive" exchange.

Mispricing between two DEXes can happen for a variety of reasons: a large swap was executed on one exchange, causing its new price to vary from others; far more liquidity on one exchange, and they receive the same (directional) volume, etc. DEX aggregators, which route trades to various exchanges for the best execution price, help solve this - but don't completely eliminate periodic arbitrage opportunities.

In the following example, MEV Bot at address `0x00000000003b3cc22af3a e1eac0440bcee416b40` found and executed an arbitrage between exchanges Sushiswap and Uniswap V2 for a marked-to-market profit of $\$2,707,921.16$ USD. The following two steps were executed in the same transaction.

1. Swap 32.85 ETH for 274.02 DG (decentraland.games token) on Uniswap V2

2. Swap 274.02 DG for 625.19 ETH on Sushiswap.

The transaction occurred on December 3, 2021, at transaction hash `0xa8b ae8393eb549d9328224be2732baf97494bd2ee4b0eb61b32ad08207cf0a0b`

Liquidations

Most lending/borrwoing protocols open source the ability to perform liquidation calls in order to increase the security of their debt ecosystem. Arbitrageurs can scan the blockchain for debt positions, and take action if, for a particular debt position, the ratio of borrowed asset / collateral

values exceeds the required LTV. Advanced arbitrageurs may use the back-run technique (back-running the oracle update) to ensure their liquidation call is the first in queue once an opportunity arises.

In a liquidation, an arbitrageur (liquidator), calls the respective platform's liquidation call, and submits a payment to instantly pay off the lenders, and free up the borrower's collateral. The platform then collects a portion of the borrower's collateral, as a penalty, and gives a chunk to the liquidator for their service. The liquidator is essentially awarded for maintaining the health of the lending/borrowing ecosystem, at the under-collateralized borrower's expense. This dynamic is very similar to margin calls in traditional finance.

Attacking the Arbitrageur: Salmonella and Beyond

In the Spring of 2021, as arbitrage bots became significantly more popular, many developers/traders became ecstatic with their "risk-free" alpha. This led to the development of overgeneralized strategies by sandwich bots, which exposed some crucial exploit possibilities.

A developer by the name of Nathan Worsley took notice, and decided to create a decoy asset pool on Uniswap, for fake token called the Salmonella contract. The key portion of the Salmonella contract was the poisonous transfer function.

```
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
  require(sender != address(0), "ERC20: transfer from the zero address");
  require(recipient != address(0), "ERC20: transfer to the zero address");
  uint256 senderBalance = _balances[sender];
  require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
  if (sender == ownerA || sender == ownerB) {
    _balances[sender] = senderBalance - amount;
    _balances[recipient] += amount;
  } else {
    _balances[sender] = senderBalance - amount;
    uint256 trapAmount = (amount * 10) / 100;
    _balances[recipient] += trapAmount;
  }
  emit Transfer(sender, recipient, amount);
}
```

As you can see, the function returns only 10% of the expected amount to any address that is not one of the two owner addresses (set in the constructor). This clever trick yielded over 130 ETH in less than 24 hours (Worsley, 2021).

News of the Salmonella contract spread like wild fire in the crypto-network of Twitter, and sandwich bot operators quickly updated their bot code with two lines of defense. First, simulate the sandwich on the current EVM state before submitting the transaction. Second, submit the miner bribe (which incentivizes the block producer to include the transaction in the block) only if the transaction yields profit.

This, however, proved to not be enough, as clever developers produced

a more complication poisonous token that overcomes these two measures.

1. To trick the simulation, the ERC20 transfer function checked to see if the block was mined by Flashbots' miners (a miner using MEV-Geth) before deciding to transfer back the full amount or the trap amount. In a local simulation, the sandwicher gets the expected profit, but in production, the sandwich bot gets trapped.

2. To ensure the miner still gets the incentive to include the transaction, the poisonous token also includes a direct payment to the miner in the "trap" case.

This is demonstrated in the psuedocode below (@bertcmiller, 2021).

```
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
  require(sender != address(0), "ERC20: transfer from the zero address");
  require(recipient != address(0), "ERC20: transfer to the zero address");
  uint256 senderBalance = _balances[sender];
  require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");

  if (block.coinbase.address == SOME_MINER_ADDRESS || sender != owner){
    _balances[sender] = senderBalance - amount;
    uint256 trapAmount = (amount * 10) / 100;
    _balances[recipient] += trapAmount;
  } else {
    _balances[sender] = senderBalance - amount;
    _balances[recipient] += amount;
  }

  emit Transfer(sender, recipient, amount);
}
```

Notice, the `require(address(this).balance > preBalance)` from the beginning of this chapter would have saved the sandwich bots hundreds of ETH, as it would have reverted the transaction upon on-chain execution.

This category of atomic MEV ultimately recognizes the various creative ways to gain profit in single-block transactions. The ethical boundaries here are blurry: it's safe to assume the methods in this category will continue to develop until regulation in the space matures.

Chapter 4

Non-Atomic, Single-Domain MEV

Although the term MEV is mostly associated with atomic transactions, there are many lesser-known strategies that have an execution duration > 1 block. The downside of this category is that, unlike the atomic transactions, you can't revert your position entry if the exit is not profitable. The upside, however, is a dramatically increased potential upside. One example of this is back-running token sales, or "token sniping".

Token Sniping

We will explore this strategy through an analysis of the IRON v2 launch of the $ICE token.

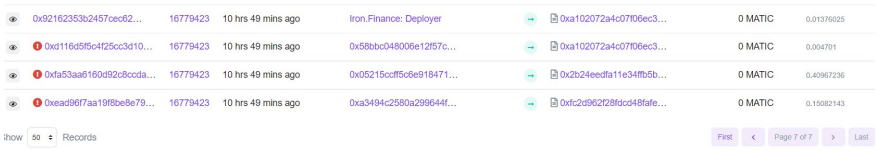
For some context, Iron Finance is a company that offers a suite of decentralize finance platforms and tools. In June of 2021, a vulnerability in three of the the company's smart contracts allowed for an exploit that broke their stablecoin's (a token with value pegged to the USD) peg, and generated panic which ensued a race to sell the TITAN token. The price of TITAN collapsed from a high of $62 to nearly zero in just a few hours - a market value loss of over $2 billion.

In the wake of the tragedy, Iron Finance capitalized on their spotlight to launch a v2 token with greater security promises: ICE token. Despite one's confidence in the ICE token, MEV arbitrageurs were certain that early entry would be well positioned for large profits in the immediate and temporary pump in value upon launch - so they prepared.

The ICE token was being launched on Polygon - a fast, low-cost side chain to Ethereum (EVM compatible). Besides being unable to utilize private transactions offered by MEV-Geth, the arbitrage dynamics are similar identical to the Ethereum ecosystem.

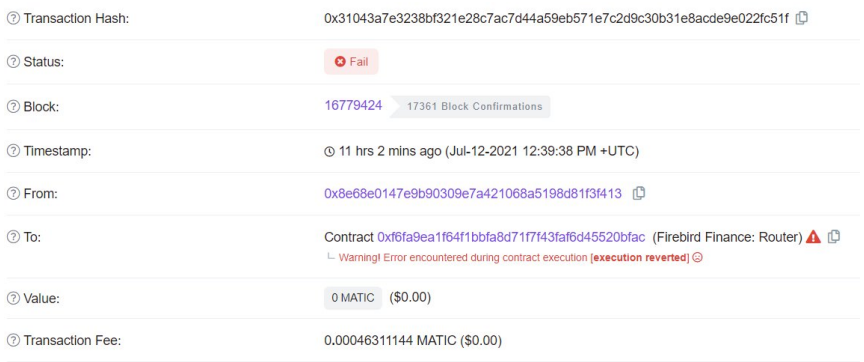
On June 12, 2021, liquidity in the form of ICE/USDC was added, initiating the sale of ICE token at a price of $0.01. The following is an overview of the MEV activity that ensued in the first minutes of the launch (@YannickCrypto, 2021).

1. **12:39:36 PM UTC** || Upon observing the liquidity transaction in the mempool, three bots submit their transactions to purchase ICE. However, they made a crucial mistake: by setting high gas prices in hopes to be the first purchases, they set the gas price *higher* than the gas price in the `addLiquidity()` transaction. This caused them to front-run the liquidity transaction. As a result, there was no liquidity available at execution, causing their purchase attempts to revert.

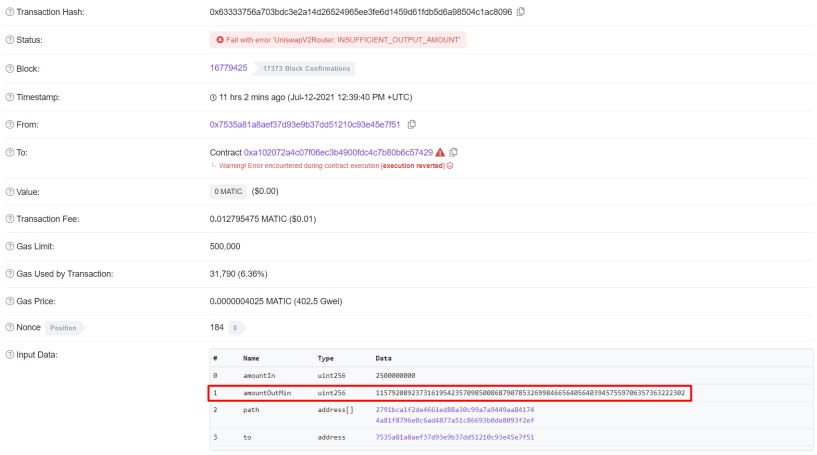


2. **12:39:36 PM UTC** || The first successful back-run is executed, purchasing 8312 $ICE for $100 USDC. As a side note, this person decided to sell their ICE position just minutes later for $1300. If he held for another hour, the sale could have been worth approximately $250,000.

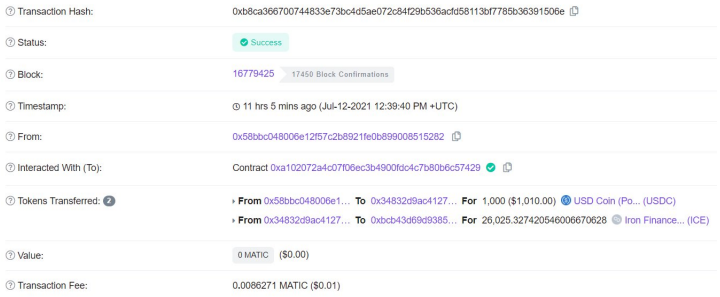
3. **12:39:38 PM UTC** || Another back-run attempts to swap .1 WETH for the $ICE token, but fails because the liquidity was added for USDC (not WETH).



4. **12:39:40 PM UTC** || Another attempt fails as the bot made an error in the `minOutput` parameter, asking for infinity $ICE



5. **12:39:40 PM UTC** || Finally, we have our winner. `0x58bb` purchased 26,025 $ICE for just 1,000 USDC. He went on to sell this 2 hours later for $537,119 USDC, securing a profit of $536,119 (53700%).



This example not only displays the opportunity to make tremendous returns by holding risk for a non-zero duration, but also shows the difficulty of implementing and executing the strategy to perfection.

The arbitrageur must have his/her infrastrucure configured to 1) scan for interactions with the correct DEX(es) 2) recognize the correct token address, and 3) analyze the `addLiquidity()` function to correctly parse out the resulting liquidity pool address and address of the $ICE-paired token.

This strategy pivots around the fact that a highly anticipated asset is being released. Under the assumption that the asset will appreciate immediately after launch, the ability to utilize MEV infrastructure to place your transaction before other purchase attempts is extremely valuable.

It may already be clear that this fact is applicable to other crypto-economies. In 2021, it became evident that this strategy is highly applicable to the NFT market.

Non-Fungible Tokens

As mentioned in the introduction, non-fungible tokens officially entered the Ethereum ecosystem in 2017 with the introduction of the ERC721 standard. However, the first NFT dates back to May, 2014 (Cascone, 2021), when Kevin McCoy minted a non-fungible blockchain marker with an explicit link to a unique work of art, via on-chain metadata on Namecoin, a Bitcoin fork.

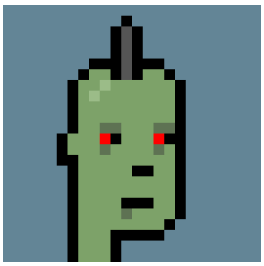
The first NFT project on Ethereum was Etheria, which was launched and demonstrated at the first Ethereum developer conference, DEVCON 1, just three months after the launch of Ethereum. Most of Etheria's 457 purchasable and tradable hexagonal tiles went unsold for more than 5 years until March 13, 2021, when renewed interest in NFTs sparked a buying frenzy. Within 24 hours, all tiles of the current version and a prior version, each hardcoded to 1 ETH ($0.43 cents at the time of launch), were sold for a total of $1.4 million (Matney, 2021).

Today, one of the most coveted NFT collections is Cryptopunks, a series of 10,000 pixelated images with randomly generated attributes. Cryptopunks were released in June 2017, by an American development studio Larva Labs. Originally, the punks were available for free to anyone with an Ethereum wallet.

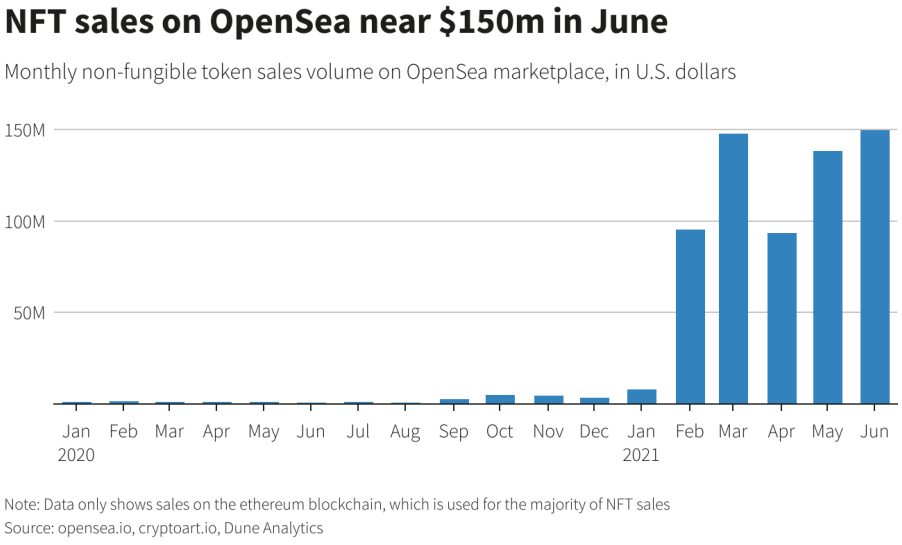
For several years, the NFT space gained little traction - Punks were traded for years in the range of just a couple hundred dollars. From 2017 to 2020, NFT collectors and blockchain developers began to invest time and effort under the belief that the innovation of digital ownership would soon resonate with a larger community of consumers... and they were right.

In the Spring of 2021, the NFT space began a period of unprecedented growth. The theses of early NFT enthusiasts came to fruition, allowing Cryptopunk owners to executed some of the greatest trades in recent history.

For example, one Ethereum user purchased Punk #2338 for 3.5 ETH, a market value of $443.00 USD, in November of 2018. On August 6, 2021, he sold his Punk for 1500 ETH, a market value of $4,379,924.93 USD. This trade yielded nearly 1,000,000% return in less than three years.



As this unprecedented appreciation of NFTs spanned beyond just the Cryptopunks project, perhaps the greatest bet was the founding of NFT-platform Opensea by Alex Atallah and Devin Finzer in 2017. The company's first seed round came from Y Combinator in January of 2018 at $120,000, when "blue-chip" NFT projects were still being traded below $1,000. This early conviction allowed Opensea to effectively claim a monopoly over the secondary NFT market on Ethereum. The magnitude of the NFT-boom of 2021 can be best visualized by the sales volume of the Opensea platform.



Non-Atomic, Single-Domain MEV: NFTs

In the NFT rage of 2021, MEV bot operators began to take notice. Hyped projects would consistently sell out in minutes, and secondary sales on Opensea could reach 100x the original mint price in under an hour. As new NFT mints took place on-chain, it became clear that MEV infrastructure could be utilized to gain an advantage in primary NFT sales.

MEV activity in NFT sales created several problems in the space. First, technical users could gain an early-mover advantage in finding and analyzing the metadata for a project, which corresponds to an unfair informational asymmetry that would allow for strategic trading on secondary markets. Second, bot activity in mints dramatically increase network congestion, increasing gas prices beyond the point an average user can participate.

To understand how MEV can manifest in the NFT space, let us dissect a sequence of events following the July 26^{th} release of the Vogu NFT Project (@0xmev, 2021).

- **July 26, 6:40:37 PM UTC** || The Vogu: VGT Token is created by the owner at address thevogu.eth
- **9:01:24 PM UTC** || The contract owner (likely accidentally) calls `toggleActive()`, allowing public mints
- **9:02:18 PM UTC** || A non-MEV user successfully executes the first public purchase (the Vogu team privately minted the first 77 items prior to sale)
- **9:02:34 PM UTC** || The Vogu team (realizing their mistake) again calls `toggleActive()` to prohibit further purchases until the scheduled launch time
- **9:02:34 PM to 11:55:26 PM UTC** || Several calls to `mintVogu()` revert, as the sale is no longer active
- **11:55:26 PM UTC** || The contract owner calls `toggleActive()` for the last time, officially commencing the public sale
- **11:55:26 PM UTC** || An MEV bot (in a public transaction) at address `0x00000005d...088` backruns the `toggleActive()` function call with a call to `mintVogu()`, securing the first 10 NFTs of the official sale in the same block as the sale is commenced
- A few people's mint attempts revert on error `Out of Gas`
- **11:56:05 PM UTC** || The first private mint transaction is executed by `0x6708...b19f`, using Flashbots' private endpoint. `0x6708` bribed the miner 1 ETH ($\$2,228$ at the time), for an effective gas cost of $\approx .059$ ETH per NFT.
- Over the next few blocks, `0x6708` executed two more private mints, spending a total of 3.93 ETH for 51 Vogu NFTs.

- **11:58:55 PM UTC** || The release is sold out.

Once the sale is officially over, `0x6708` quickly begins to flip his NFTs on Opensea, at an average price of 0.4 ETH. In less than 2 hours, `0x6708` completes the operation for a secured profit of ≈ 16 ETH, or $\approx \$35,600$.

This type of activity was extremely common throughout the Summer of 2021. For several months this activity was done "in the dark" - it wasn't until July when crypto Twitter profiles like @mevintern and @0xmev began to publicly share knowledge of these events.

Once a larger portion of people realized the inefficiencies in NFT sales, several strategies were adopted by the teams of upcoming projects (Hasu, 2021).

One such strategy is referred to as the "stealth launch". The idea is that if no one is aware of the project prior to launch, MEV developers wouldn't be capable of preparing the infrastructure necessary for an advantage prior to the sale.

Some projects also began to hard-code a mint-transaction limit into the NFT sale contract, prohibiting someone like `0x6708` from securing a mass portion of the sale.

An additional measure that prevented the traditional on-chain bot contract from participating in mints was a piece of code that ensured the mint transaction was called by an externally-owned-account (a real user, not a smart contract).

Despite these strategies, it remained extremely difficult for even well-funded projects to circumnavigate the issue of sales bots. One example of this is the NFT project by Time Magazine.

Despite valiant attempts to maintain the integrity of the launch, TIME Magazine's release of TIMEPieces also fell victim to MEV. While the TIMEPieces contract restricted participants to minting a maximum of ten NFTs per address, bot operator `0x35` planned ahead, split funds across five wallets, and sniped 50 NFTs in a single Flashbots bundle. Even after paying a 20 ETH miner tip (bringing their average mint cost to 0.5 ETH per NFT), this bot operator stood to profit nearly 120 ETH when the NFTs began to hit the secondary market (Hasu, 2021).

A Replication of an NFT MEV Operation

Now that we have an understanding of the NFT market, and its relation to MEV, let us attempt to replicate the strategy of what is potentially one of the most profitable NFT MEV operations to date.

On September 8^{th}, 2021, a project called Cryptoadz, despite attempting a stealth launch and contract-restriction, became the target of potentially one of most successful MEV operations in the NFT space.

First, let us take a look at the Cryptoadz contract, deployed on Ethereum on September 8, 2021. Anyone with knowledge of the project was able to directly read the Solidity source code of the Toadz contract, as its bytecode was verified by the owner on Etherscan.io. The contract can be found at address `0x1CB1A5e65610AEFF2551A50f76a87a7d3fB649C6`.

Between the contract declaration on line 42 of Toadz.sol and the constructor on line 53, you can see some valuable information about the project.

```
contract Toadz is ERC721, PaymentSplitter, Ownable {
    uint256 public constant maxTokens = 6969;
    uint256 public constant maxMintsPerTx = 10;
    uint256 public tokenPrice = 69000000000000000; //0.069 ether
    uint256 public startingBlock =999999999;
    string private _contractURI;
    string public provenance;
    uint256 public nextTokenId=1;
    bool public devMintLocked = false;
    bool private initialized = false;

    constructor() public ERC721("Cryptoadz", "TOADZ") { }
```

Mimicking the MEV operator, we are able to learn several things from reading this contract. The total amount of NFTs in the project is 6969, the

maximum purchase per person was limited to 10, and the price per NFT was 0.069 ETH. Additionally, there is a `startingBlock` variable which likely corresponds to the start of the sale. As it was initially set to an arbitrarily large number, we can expect to see a function that allows the contract owner to change this value ahead of the launch.

Further down the contract, we see the two relevant functions, `setStartingBlock(uint256 _startingBlock)` (as we expected) and `mint(uint256 quantity)`, the purchase function.

Let's first take a look at the `setStartingBlock()` function.

```
function setStartingBlock(uint256 _startingBlock) public onlyOwner {
    startingBlock=_startingBlock;
}
```

We notice that this function is just as we expected. It has a sole function of replacing the value `startingBlock`, from line 46, with a uint256 value provided by the contract owner in the function call. Since this will tell us when the sale will become active, we know we want to keep an eye on any activity related to this function.

We will decompile the Solidity contract (using an online Solidity decompiler) into bytecode, so we can retrieve the four-byte string corresponding to this method header: we get `0xec17b20e`. We will use this four-byte string, in combination with the Toadz.sol contract address, as a filter for incoming transactions to the Ethereum mempool, to find out which Ethereum block this sale will become active.

Next we take a look at the `mint()` function.

```
function mint(uint256 quantity) external payable {
    require(block.number >= startingBlock,
        "Sale hasn't started yet!");
    require(quantity <= maxMintsPerTx,
        "There is a limit on minting too many at a time!");
    require(nextTokenId -1 + quantity <= maxTokens ,
        "Minting this many would exceed supply!");
    require(msg.value >= tokenPrice * quantity,
        "Not enough ether sent!");
    require(msg.sender == tx.origin,
        "No contracts!");
    for (uint256 i = 0; i < quantity; i++) {
        _safeMint(msg.sender, nextTokenId++);
    }
}
```

The `require()` statements declare all requirements for an incoming `mint()` call to be successful - so we take note of each.

First, the sale is active starting (and including) the block matching `startingBlock`, so any attempts to purchase beforehand will fail. Second, we cannot request to mint more than 10 NFTs per transaction, confirming our intuition from earlier. Third, we cannot request to mint an amount that would exceed the total supply (if there are 8 items left in the sale, and we attempt to purchase 9, we will receive none). Next, we must pay for our transaction - $0.069 \times$ `quantity` ETH.

Finally, Toadz.sol prevents mint calls from contracts (similar to TIME-Pieces), with the requirement `msg.sender == tx.origin`. Thus, we must send our mints from an EOA address, not through an on-chain contract relay.

To prepare for our operation, we also find the four-byte filter corresponding to the method header `mint(): 0xa0712d68`.

The first thing we want to do is begin scanning for a `setStartingBlock()` function call. This can be done at any time, and the `_startingBlock` value can be set to the block immediately after the function call.

In order to do this, we need to establish a connection with the Ethereum network. An experienced arbitrageur such as the one that performed this operation likely runs his/her own Ethereum node. However, one can also use paid services like QuickNode or BloXroute, or free (slower, not optimal for this type of activity) services offered in introductory plans on Alchemy and Infura, for example.

For the purposes of our example, we will be using BloXroute. The first thing we want to do is establish a connection, and subscribe to receive new transactions. The logic may look something like this.

```
func (ds *BxDatasource) ListenNewTransactions() error {
        log.Println("BxDatasource.ListenNewTransactions().. ")

        // Configure authentication
        tlsConfig := &tls.Config{
                Certificates:       []tls.Certificate{ds.cert},
                InsecureSkipVerify: true,
        }
        dialer := websocket.DefaultDialer
        dialer.TLSClientConfig = tlsConfig

        // Dial websocket
        wsSubscriber, _, err := dialer.Dial(ds.Addr, nil)
        if err != nil {
                log.Println("failed to dial with authorization:", err)
                return err
        }
        ds.TransactionConnection = wsSubscriber

        // Create and submit subscription
        subRequst := `{"id": 1, "method": "subscribe", "params": ["pendingTxs", {"include":
            ["tx_hash", "raw_tx", "tx_contents.input", "tx_contents.to", "tx_contents.from",
```

```
            "tx_contents.value", "tx_contents.gas", "tx_contents.gas_price",
            "tx_contents.max_priority_fee_per_gas", "tx_contents.max_fee_per_gas"]}]}`
        err = ds.TransactionConnection.WriteMessage(websocket.TextMessage, []byte(subRequst))
        if err != nil {
                log.Println("Failed to subscribe to new txs", err)
                return err
        }

        go ds.looperNewTxs()
        return nil
}
```

Once we created our connection to the network and began streaming transactions, `ds.looperNewTxs()` simply loops on incoming messages, marshals the transaction data into the appropriate structs, and forwards the data to our server.

Here, we filter incoming transactions by our desired fields, and forward matching transactions to all client channels. For Cryptoadz, we would have:

```
newSub.filter["to"] == "0x1CB1A5e65610AEFF2551A50f76a87a7d3fB649C6"
newSub.filter["fourByte"] == "ec17b20e"
```

```
done := false
for !done {
  select {
  case newTransactionEvent := <-newSub.subscriptionChan:
    newTx := newTransactionEvent.Transaction.Transaction

    // Filter by "from", "to", and "fourByte" fields
    if newSub.filter["to"] != "" && newSub.filter["to"] !=
        newTx.TxContents.To {
      log.Println("Filtered out tx - incorrect *to*")
      continue
    }
    if newSub.filter["from"] != "" && newSub.filter["from"] !=
        newTx.TxContents.From {
      log.Println("Filtered out tx - incorrect *from*")
      continue
    }
    if newSub.filter["fourByte"] != "" && newSub.filter["fourByte"]
        != newTx.TxContents.Input[2:10] {
      log.Println("Filtered out tx - incorrect *fourByte*")
      continue
    }

    if err := stream.Send(tx); err != nil {
      done = true
      break
    }
  }
}
```

We now have our mempool infrastructure up and running. The next thing we want to do is set up our mint strategy, to be invoked at the `startingBlock`. For the purposes of speed and server/client compatibility, we will set up our execution infrastructure in Golang as well.

To interact with the Cryptoadz contract in Golang, we need to compile the Solidity contract, and generate a Golang ABI. We will use the Solc package for compilation and Abigen for the ABI.

```
$ solc --bin ./contracts/Toadz.sol -o ./build --overwrite --metadata-hash none --revert-strings
    strip --optimize --optimize-runs 1000
$ solc --abi ./contracts/Toadz.sol -o ./build --overwrite --metadata-hash none --revert-strings
    strip --optimize --optimize-runs 1000
$ abigen --abi=./build/Toadz.abi --bin=./build/Toadz.bin --pkg=toadz
    --out=../client/abis/toadz.go
```

The generated binding has the structs and functions we need to generate our transactions. Once our infrastructure sees the `setStartingBlock()` transaction, it will invoke the action sequence to execute just before the specified block. The action sequence will be

1. Create and sign the buy transactions using multiple wallets
2. Simulate the generated transactions on the EVM
3. Send the transactions in bundles to Flashbots' private relay

Below we show the code to execute steps 1 and 2. For each of our wallets, we create our transaction using the generated `ToadzTransactor` struct, simulate the transaction, and append it to our list to be relayed to the network.

Let `OpData` be a struct containing relevant data (wallet keystores, preset purchase limits, etc.) and let "..." denote omission of parameters for simplicity.

```
allTxs := []*types.Transaction{}
for i := 0; i < len(opData.Keys); i++ {
  // Make sure account has enough money
  if balances[opData.Keys[i].Account.Address].Cmp(totalCost) < 0 {
    log.Println("Not enough eth:", opData.Keys[i].Account.Address)
    continue
  }

  // Create and simulate transactions
  var outTx *types.Transaction
  err = execute.SimulateTx(opData.Keys[i], ..., func(auth *bind.TransactOpts) error {
    monitorContract, _ := monitor.NewToadz(opData.Contract, client)
    tx, err := monitorContract.ToadzTransactor.MintTokens
        (auth, opData.UnitCount)
    if err != nil {
      log.Println("buy failed", err)
      return err
    }
    outTx = tx
    return nil
  })
  if outTx != nil {
    allTxs = append(allTxs, outTx)
  }
}
return allTxs, err
```

Now that we have our list of valid transactions, all we have to do is send them to the network! This can be done by calling `SendBundle()` to an open-source script `flashbots.go` (ctrngk, 2021). Now we are done - we are ready to *mev* an NFT sale.

To find out how we would have done, had we preapred for the Cryptoadz launch, let's take a look at the details of the September 8^{th} operation.

By analyzing Flashbots bundles submitted to the Cryptoadz contract in the first minutes of launch, we can find the wallets involved, from which we can retrieve the rest of the information.

On September 8^{th}, the arbitrageur used the above technique with 10 wallets to purchase a total of 150 Toadz: 5 wallets purchased 10 Toadz each and the other 5 wallets purchased 20 each (2 purchases of 10). The average cost, including Gas fees, to the arbitrageur was 0.15 ETH. Over the following days, the Toadz were sold at an average of 1.38 ETH, including one sale for 69 ETH (Toadz 318).

In less than a week, the arbitrageurs made ≈ 185 ETH in profit - a current USD value of over $\$800,000$.

The rise of NFT markets yielded a new application of MEV dynamics in 2021. Over the course of 6 months (July-December 2021), it's safe to estimate MEV activity in NFT markets were responsible for over $10 million USD in extracted value. This is a lower bound - the real numbers are likely much higher than this estimate.

Chapter 5

Cross-Domain MEV

Since the launch of Ethereum, the first decentralized smart contract platform, in 2015, the ecosystem of such platforms has evolved tremendously. Over the last five years, dozens of teams formed to create additional smart contract platforms. This includes alternative Layer 1 chains to Ethereum, as well as Layer 2 rollups to provide additional scalability and functionality on top of the Ethereum base layer. Although Ethereum maintains dominance in the ecosystem by market cap, the significant growth of alternative chains suggest the future of blockchain technology is one of many domains.

definition. *Layer 2 Rollups* is a collective term for solutions designed to help scale an Ethereum application by handling transactions off the Ethereum Mainnet (layer 1) while taking advantage of the robust decentralized security model of Mainnet (*Ethereum Documentation*, n.d.).

Below is a brief timeline highlighting the conception and launch of a few notable networks since the Ethereum genesis block.

2016

- Cosmos ($ATOM) whitepaper is published by Tendermint co-founders Ethan Buchman and Jae Kwon
- Polkadot ($DOT) is founded by Gavin Wood, former Co-Founder and CTO of Ethereum

2017

- Algorand ($ALGO) is founded by cryptography pioneer, Turing award winner and MIT professor, Silvio Micali
- Solana ($SOL) Foundation is founded, as Anatoly Yakovenko publishes a whitepaper describing a new BFT-mechanism, Proof of History

2018

- Terra ($LUNA) is founded by Daniel Shin and Do Kwon
- Avalanche ($AVAX) is anonymously introduced on IPFS
- Arbitrum is founded by Ed Felton, a computer science and public affairs professor at Princeton.

2019

- Algorand mainnet is launched
- Cosmos mainnet is launched
- Avalanche mainnet is launched
- Terra mainnet is launched
- Optimsm (L2 Rollup) is founded

2020

- Solana mainnet beta is launched
- Polkadot mainnet is launched
- Avalanche mainnet is launched
- Binance Smart Chain (BSC) is anounced and launched by Binance

2021

- Optimism mainnet is launched
- Arbitrum mainnet is launched
- Starknet (L2 Rollup) is announced and later launched

This brief overview of the development of the blockchain ecosystem is necessary to understand the current and future state of MEV. MEV is not unique to Ethereum, nor is it restrained to a single domain. Nearly all of the strategies previously explored are being deployed on the chains listed above, generating revenue north of eight figures. And as developments in bridge and Layer 0 technology improve chain interoperability, the possibility of cross-domain value extraction is becoming increasingly evident.

definition. A *bridge*, in the context of blockchains, is a connection that allows the transfer of tokens and/or arbitrary data from one chain to another.

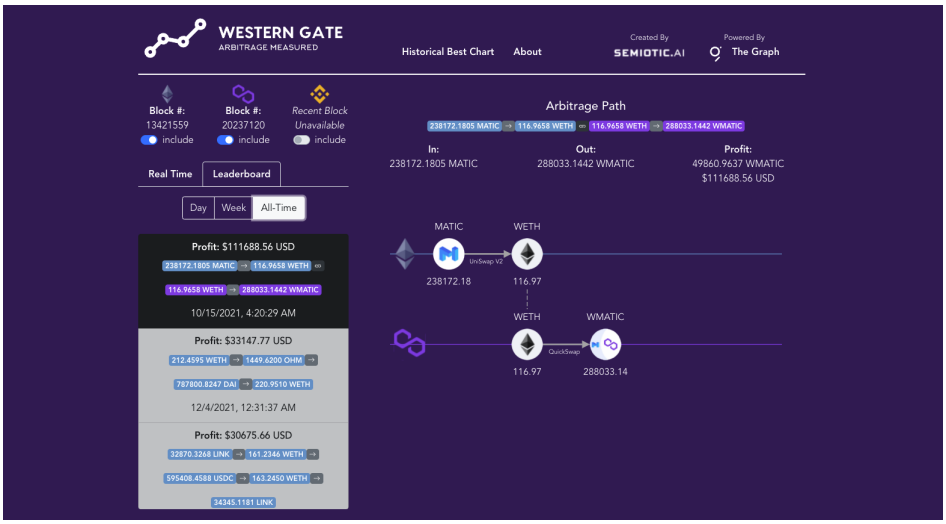
The Current State of Cross-Domain MEV

It remains today that the overwhelming majority of MEV is single-domain.

Although several MEV operators have distinct operations deployed on various chains, for the most part, each executes as a single-domain system.

Due to the inherent ambiguity of cross-domain MEV, it's difficult to measure or estimate current activity that would adhere to this classification. What is clear, however, is the fact that investigation into this sector has begun.

A cross-domain MEV tracker built by Semiotic.AI monitors and displays cross-domain arbitrage opportunities by analyzing exchange rates across DEXes on Ethereum (L1), Binance Smart Chain, and Polygon. Below is a screenshot of Semiotic's dashboard, deployed at westerngate.xyz (?, ?).



As seen above, on October 15, 2021, there was an opportunity to exchange 238172.18 MATIC token on the Ethereum chain for 288033.14 WMATIC (Polygon's wrapped MATIC token, equal in value to MATIC) for a mark-to-market profit of $\$111,688.56$ USD. The existence of these opportunities implies there exists liquidity pools with identical asset pairs, but with varying levels of market depth, volume, and general activity.

A Formal Definition of Cross-Domain Maximal Extractable Value

Before we investigate cross-domain MEV, let us establish the formal definition evolved, by (Obadia et al., 2021), from the single-domain definition. Recall the definition of *single-domain maximal extractable value*:

$$mev_i^i(P, s) = \max_{a_i...a_n \in A_i} \{ev_i(P, s, a_1...a_n)\}$$

with *extractable value*

$$ev_i(P, s, a_1...a_n) = b_i(s', P) - b_i(s, P)$$

This can be generalized to an n-domain definition, with a player P that has access to an action space representing abilities across multiple domains $A = A_1 \cup A_2 \cup ... \cup A_n$, and that wants to maximize balances across domains $B = B_1 \cup B_2 \cup ... \cup B_n$ with respective pricing functions of each asset priced in domain B_1 as $p_{B_1 \to B_1}, ..., p_{B_n \to B_1}$.

The generalized formula requires a pricing function $P_{j \to i}$ that computes the balance in domain j in units of the native asset of domain i by multiplying it by this pricing function. This notion of equivalent conversion is required to meaningfully add the balances in two domains.

We will also assume that $p_{i \to j} = \frac{1}{p_{j \to i}}$. As noted by the authors of the referenced definition, this property is guaranteed if assets share a single global orderbook, but may not hold for all decentralized assets (Obadia et al., 2021).

$$mev_B^A(P, s) = \max_{a_i ... a_n \in A} \{ \sum_{b \in B} p_{b \to B_1}(ev_i(P, s, a_1 ... a_n)) \}$$

Ultimately, cross-domain MEV is the maximum of the sum of final balances across all considered domains, when some mix of actions across all those domains are executed together.

A Cross-Chain Arbitrage Strategy

Now that we have our definition, we will continue our investigation with hypothetical implementation approaches to a 2-domain MEV system, clarifying how it can be expanded to n domains.

As nearly all MEV-related research is confined to Ethereum/EVM networks, I am motivated to include a non-EVM layer 1 chain. Thus, the cross-domain arbitrage example will be between Ethereum and Solana.

A quick note on Solana.

With a native token currently sitting as the #5 ranked cryptocurrency by market cap, Solana is a high performance layer 1 blockchain that combines Proof-of-History and Proof-of-Stake for a high-frequency BFT mechanism.

definition. *Proof-of-History* is a sequence of computation that can provide a way to cryptographically verify passage of time between two events (Yakovenko, 2018).

definition. *Proof-of-Stake* is a class of consensus mechanisms for blockchains that work by selecting validators in proportion to their quantity of holdings in the associated cryptocurrency.

Because Solana is not natively (N.A., n.d.) EVM-compatible, Geth, MEV-Geth, and the Solidity code featured in Chapters 3 and 4 do not apply. In fact, the concept of single-domain MEV on Solana is in its infancy, and is out of the the scope of this paper. I will note however, that early work shows some form of it may exists as a result of modifying the Solana Validator source code, similar in concept to the modification of Geth to MEV-Geth (Labs, 2021).

One of the most obvious potential sources of arbitrage between Ethereum and Solana is the difference in network speed. Ethereum handles approximately 20 transactions per second, with new blocks generated every $12-14$ seconds on average. Solana, on the other hand, can currently handle $65,000$ transactions per second with block time 400ms.

We define our set of domains and our set of balances to include Ethereum and Solana. Formally, we have domains $A = A_E \cup A_S$ and balances $B = B_E \cup B_S$. Let ϵ represent the "negligible" quantity, a constant that approximates the sum of transaction fees associated with executing an arbitrage and rebalancing the portfolio. Let eb_i be the i^{th} Ethereum block Let $B_{E,i}$ be the i^{th} Ethereum block.

$$max_{b \in B_S}[\Delta_{eb_i, eb_{i+1}}(p_{b \to B_E})] > \frac{\epsilon}{|B_E|} \implies MEV_B^A(P, s) > \epsilon$$

We say that if the maximum change in price (over the Ethereum block duration) if an asset on Solana is greater than our "negligible" term divided by our single-domain portfolio size, then the cross-domain MEV opportunity for the given duration is non-negligible.

We say that if an MEV opportunity does not exists between Solana and Ethereum in the duration of an Ethereum block production, then we know the maximum relative price change of a Solana asset denominated in the corresponding Ethereum asset in the same time frame is less than "negligible" divided by the size of our single-domain balance.

Applying this logic, the following pseudocode displays the logic triggered on a new Ethereum block.

```
for {
  // Stream solana asset prices
  solPrices <- solPriceStrem
  // Update graph representation
  arbGraph.UpdateSolQuotes(solPrices)
  select {
  case newBlock := <-newEthBlocks:
    // Calculate Ethereum asset prices off-chain from new Dex reserves
```

```
    ethPrices := newReserves(newBlock)
    // Update graph
    arbGraph.UpdateEthQuotes(ethPrices)
    // Find best arbitrage with Ford graph algorithm
    bestArb := arbGraph.BellmanFord()
    // Execute arbitrage
    executor.DoArb(bestArb)
  }
}
```

Although this logic seems simple, there is hidden complexity in the graph and path-finding implementations of `arbGraph` as well as the execution script noted `executor.doArb(bestArb)`.

`arbGraph` is a graph representation of the action space. Nodes represent (asset, domain) pairs which the trader has "whitelisted". Edges are weighted ((asset, domain), (asset, domain)) pairs of nodes representing paths of asset exchange, with weight corresponding to the notional value ratio of $\frac{StartBalance}{EndBalance}$ resulting in its traversal (this is so that shortest path $\rightarrow$ best arbitrage).

There are multiple different approaches with respect to the implementation of edges. The choice of this implementation has consequences on the implementations of `arbGraph.BellmanFord()` and `executor.DoArb(bestArb)`. The investigation below describes my approach, which is simply one of many.

In my approach, we simplify edge complexity by separating the execution of single-domain trades from cross-domain transfers (rebalancing). This will allow us to execute single-domain trades within each domain immediately with our `Executor` system, and queue the cross-domain transfers for execution by a separate system, we we call `DomainManager`. Thus, our two types of edges are

1. Single-domain DEX pairs
2. Cross-domain, identical asset pairs, with constant weight equal to 1

By maintaining a constant weight of one for our synthetic bridge edges, we formally equate identical assets on different domains. Thus, our Bellman Ford algorithm runs just as it would in a single-domain arbitrage infrastructure, finding the shortest cyclical path, which represents the maximum arbitrage path. We compute the length of the entire path by multiplying edge weights, and confirm the result is less than one (meaning end balance $>$ start balance).

We will now describe, at a high level, the logic of `executor.DoArb()`.

WLOG, let our best arbitrage path for a particular Ethereum block consist of a DEX trade on Ethereum, a bridge transfer, and a trade on Solana. We have `bestArb` $= [t_E, t_b, t_S]$. The execution logic would then be as follows.

1. Start a goroutine to execute t_E utilizing our Ethereum infrastructure.
2. Start a goroutine to prepare and execute t_S on Solana.
   (a) Borrow the specified quantity of the asset required for t_S from an on-chain lending platform.
   (b) Execute t_S to secure the desired *endBalance*.
3. Send the data of the synthetic bridge transfer t_b to our `DomainManager`.

Our `DomainManager` can operate in a variety of ways. Perhaps most simply, it can be a custodial account on a centralized exchange such as FTX or Coinbase. In this case, t_b essentially includes a transfer of the result of t_E to the `DomainManager`, and a corresponding transfer of the same asset for loan repayment (step 2.a) on Solana.

A more complex implementation is also possible, where an aggregation of decentralized bridge paths are constantly evaluated. Depending on the risk profile of the arbitrageur, the bridge execution can either immediately executed to lock in the current best rate, or can be held in queue for some duration of time, until the bridge aggregator identifies a positive expected value path.

In summary, the approach to cross-domain arbitrage closely resembles its single-domain counterpart. As this category of MEV evolves, key strategic developments will revolve around the handling of the cross-domain edges within the arbitrage path.

Technological developments in blockchain bridges, "Layer 0" solutions, and general inter-blockchain communication will heavily influence the evolution of cross-domain MEV strategies.

Although cross-domain arbitrage has the ability to contribute to market efficiency across the multi-chain ecosystem, there are also concerns regarded some of the negative developments that can arise.
One concerning possibility is the rise of barriers to entry, enforced by cross-domain infrastructure and low latency achieved only by expensive hardware. Cross-Domain MEV may also become an economy of scale, wherein trading firms with large capital develop an unfair advantage over smaller arbitrageurs with an ability to maintain balances across domains for all "whitelisted" tokens (with a hedging mechanism for delta-neutrality).

Ultimately, this chapter only scratches the surface of possibilities with respect to the cross-domain category of MEV. Besides describing the high-level logic behind a multi-chain arbitrage, the investigation attempts to shine light on the ability to expand MEV strategies to the multi-domain action space.

Chapter 6

Ethical Considerations

The concept of MEV was first introduced by Phil Daian in the 2019 paper, *Flash Boys 2.0*. Over the following months, it quickly ascended to one of the important concepts in cryptoeconomics, with potentially negative implications on both user experience and network-wide consensus stability. As others began to explore the subject, two opposing opinions began to emerge. These opinions, among many others, ultimately revolve around a single question...

Is MEV unavoidable?

The "yes" perspective is embodied by Phil Daian, one of the leading authors of the paper that coined the term *MEV* in the 2019 paper, *Flashbots 2.0*. The perspective shared by Daian, Flashbots, and others is that MEV is a fundamental metric for network security in both decentralized and centralized systems with economic incentives. There will always be users of the network that prefer one version of events to another - and the resulting effort to condition payments to express their preferred outcome will furnish some form of MEV (Daian, 2021).

The major opposition to this viewpoint is led by Cornell professor and researcher Dr. Ari Juels. Juels breaks down his criticisms of the Flashbots perspective in an opinion piece titled *Miners, Front-Running as a Service is Theft*. In the piece, Juels equates front-running as a service (FaaS) to "auction[ing] off the right to mug and burglarize homes" (Ari Juels, 2021).

Juels represents the belief that MEV can be avoided through the meticulous design of fair order sequencing protocols. He states there are "alternative approaches to ordering transactions that provide stronger fairness assurances for users."

One example of an alternative approach is to encrypt message contents until the block order is finalized, at which point the transactions can be

decrypted and executed on-chain (Cachin, Kursawe, Petzold, & Shoup, 2001). Another approach is to randomly sequence transactions within a block, but this brings up alternative problems concerning chain efficiency and insecurity against spamming.

Perhaps the most promising alternative approach is the Fair Sequencing Service (FSS), led by a collaborative effort between Juels and researchers at Chainlink Labs (Juels, 2021). FSS uses a consensus mechanism to decide single-block transaction ordering rather than granting the priveledges to a single entity for the block's duration.

Ultimately, it's important that both approaches continue to make strides. The Flashbots organization has an essential role in the ecosystem today - and until FSS (or other approaches) become widely deployed, its important for value extraction to be deomcratized as much as possible. And as is true with almost every other area of technological development, competition is essential for the best solutions to rise to the top.

Chapter 7

Conclusion

The purpose of this paper was to explore the various manifestations of MEV, and highlight the complexity of some of the strategies that have evolved in the space. As the role of blockchains becomes increasingly important in global markets, it's likely that "illuminating the dark forest" becomes increasingly imperative.

In addition to reducing the informational asymmetry regarding on-chain arbitrage, I hope this paper reveals parallels to traditional economic systems.

MEV is a metaphysical concept. There will always be heirarchies of power, and there will always be variance of preferences.

Ultimately, the design of incentives is infinitely more powerful than a design of regulations. In the long term, future work regarding MEV may have larger consequences on distributed market dynamics than any future regulations imposed by centralized governments.

References

@0xmev. (2021). https://twitter.com/0xmev/status/1420105555910213636?s=20. Twitter. 32

2, C. F. R. (2021). https://rekt.news/cream-rekt-2/. (Accessed: 2021-11-25) 18

Adams, H., Zinsmeister, N., Salem, M., Keefer, R., & Robinson, D. (2021). *Uniswap v3 core* (Tech. Rep.). Tech. rep., Uniswap. 17

Analytics, C. (2021). https://dune.xyz/ChainsightAnalytics/Uniswap-v3-Just-in-Time-(JIT)-Liquidity-MEV. (Accessed: 2021-11-10) 22

Ari Juels, M. K., Ittay Eyal. (2021). Miners, front-running-as-a-service is theft. *Coindesk*. 49

@bertcmiller. (2021). https://twitter.com/bertcmiller/status/1381296097130377216?s=20. Twitter. 25

Buterin, V., et al. (2013). Ethereum white paper. *GitHub repository*, *1*, 22–23. 2

Cachin, C., Kursawe, K., Petzold, F., & Shoup, V. (2001). Secure and efficient asynchronous broadcast protocols. In *Annual international cryptology conference* (pp. 524–541). 50

Cascone, S. (2021). Sotheby's is selling the first nft ever minted—and bidding starts at $100. *Artnet News*. 30

Chohan, U. W. (2017). A history of bitcoin. *Available at SSRN 3047875*. 1

ctrngk. (2021). *flashbots.go*. https://github.com/ctrngk/flashbots-bundle/blob/master/flashbots.go. GitHub. 39

Daian, P. (2021). Mev... wat do? *Phil Does Security*. 49

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., ... Juels, A. (2019). Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*. 6, 13

Ethereum documentation. (n.d.). https://ethereum.org/en/developers/docs/. (Accessed: 2021-11-04) 10, 41

Hasu, A. A. (2021). https://www.paradigm.xyz/2021/10/a-guide-to-designing-effective-nft-launches/. Paradigm. 33

Here's the problem with the new theory that a japanese math professor is the inventor of bitcoin. (2015). *San Francisco Chronicle*. 1

Hildenbrandt, E., Saxena, M., Rodrigues, N., Zhu, X., Daian, P., Guth,

D., ... others (2018). Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 ieee 31st computer security foundations symposium (csf)* (pp. 204–217). 9

Juels, A. (2021). Fair sequencing services: Enabling a provably fair defi ecosystem. *Chainlink*. 50

Labs, J. (2021). Solana validator 101: Transaction processing. *Medium*. 45

Matney, L. (2021). The cult of cryptopunks. *TechCrunch*. 30

Mev miner. (n.d.). https://mevminer.vercel.app/. (Accessed: 2021-11-04) 4

N.A. (n.d.). Neon evm: Ethereum smart contracts scaled by solana v1.1. 45

Nakamoto, S. (2008). Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*. 1

Obadia, A. (2020). Flashbots: Frontrunning the mev crisis. *Medium*. 12

Obadia, A., Salles, A., Sankar, L., Chitra, T., Chellani, V., & Daian, P. (2021). *Unity is strength: A formalization of cross-domain maximal extractable value.* 5, 15, 43, 44

On path independence. (2017). https://vitalik.ca/general/2017/06/22/marketmakers.html. (Accessed: 2021-11-07) 16

Poitras, G. (2021). Origins of arbitrage. *Financial History Review*, *28*(1), 96–123. doi: 10.1017/S0968565021000020 6

Worsley, N. (2021). *Wrecking aandwich traders for fun and profit.* https://github.com/Defi-Cartel/salmonella. GitHub. 24

Yakovenko, A. (2018). Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*. 44

@YannickCrypto. (2021). https://twitter.com/YannickCrypto/status/1414733397054955525?s=20. Twitter. 28

Zhou, L., Qin, K., Torres, C. F., Le, D. V., & Gervais, A. (2021). High-frequency trading on decentralized on-chain exchanges. In *2021 ieee symposium on security and privacy (sp)* (pp. 428–445). 20