#13 Maximal Extractable Value (MEV)

Lecture Notes for CS190N: Blockchain Technologies and Security November 12, 2025

This lecture explores Maximal Extractable Value (MEV), a fundamental economic phenomenon in blockchain systems. We examine how block producers and specialized actors can extract value by strategically ordering transactions, analyze common MEV strategies including front-running and sandwich attacks, and discuss both the harmful and beneficial impacts of MEV on the ecosystem. The lecture concludes with a case study of the Sentinel framework, which repurposes MEV mechanics for defensive purposes.

1 INTRODUCTION: THE BLOCK PRODUCTION LOTTERY

1.1 The Blockchain Mempool: A Public Waiting Room

To understand Maximal Extractable Value (MEV), we must first understand the transaction lifecycle on a blockchain. When a user sends a transaction, whether it's a simple token transfer, a DEX swap, or a loan repayment, it is not immediately added to a block. Instead, it is first broadcast to a public waiting area known as the **mempool** (memory pool). The mempool is a collection of all pending, unconfirmed transactions. Every node in the network has its own version of the mempool, but they all converge on a similar set of pending transactions.

The key characteristic of the mempool is its transparency. All transactions within it are visible to anyone, including their contents and the gas fees users are willing to pay. This transparency is foundational to how a decentralized network operates, but it also creates a unique economic environment that enables MEV.

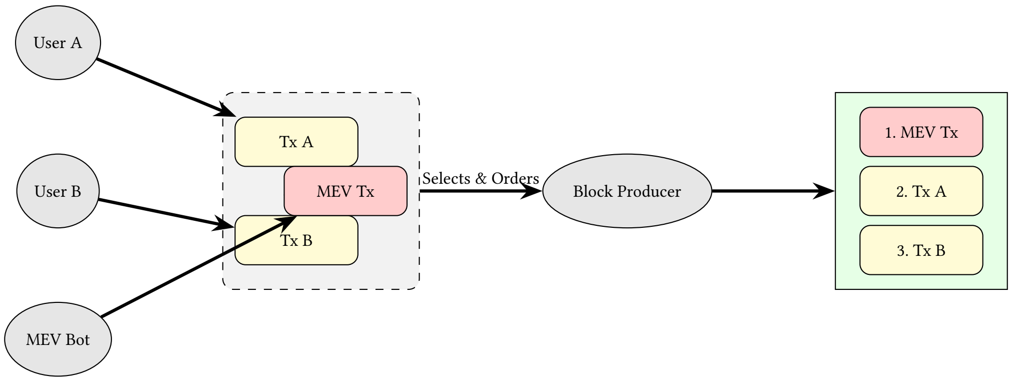


Fig. 1. The transaction lifecycle and the origin of MEV. User transactions enter the public mempool, where a block producer has the authority to select and reorder them to maximize their profit before including them in a new block.

1.2 What is MEV? Defining a New Financial Primitive

Originally, MEV stood for **Miner Extractable Value**, as block producers in Proof-of-Work (PoW) systems like early Ethereum were called "miners." Following the transition to Proof-of-Stake (PoS) and the rise of other privileged block producers (like sequencers on rollups), the term has evolved to **Maximal Extractable Value**.

MEV is defined as the maximum value that can be extracted by a block producer by strategically including, excluding, or changing the order of transactions within a block they create. This value is in addition to the standard block rewards and transaction fees. At its core, MEV is an economic phenomenon that arises from a block producer's ability to coordinate and sequence user transactions in a way that benefits them. It is a fundamental force in how blocks are built on nearly all blockchains.

The primary reason MEV exists is the information asymmetry in the system. Transactions are public and can be front-run, back-run, or reordered before they are confirmed. Block producers and other specialized actors known as "searchers" are in a prime position to exploit this by observing the mempool for profitable opportunities and creating their own transactions to capture that value.

2 CORE MEV STRATEGIES: THE GOOD, THE BAD, AND THE UGLY

MEV strategies are complex and often involve multiple transactions bundled together to be executed atomically. We will examine the most common types, starting with the basic building blocks and progressing to more sophisticated attacks.

2.1 Front-running and Back-running: The Basic Tools

- **Front-running:** This is the most straightforward form of MEV. A front-runner sees a pending, profitable transaction in the mempool, such as a large DEX swap that will significantly move an asset's price. The front-runner then submits their own transaction (e.g., a buy order for the same asset) with a higher gas fee. By paying more, their transaction is prioritized and confirmed *before* the original transaction. This causes the price to move up, and the original transaction then executes at a worse, less favorable price.
- **Back-running:** This involves executing a transaction immediately *after* a target transaction to profit from its price impact. For example, if a large trade causes a price imbalance between two DEXs, a back-runner can execute a transaction immediately afterward to buy the asset on the cheaper DEX and sell it on the pricier one, capturing the arbitrage profit. Back-running is often considered less harmful than front-running because it doesn't directly hurt the original user; it simply captures a profit opportunity that the user's trade created.

2.2 The Sandwich Attack: A Profitable Combination

The **sandwich attack** is a powerful and very common MEV strategy that combines both front-running and back-running to profit at the expense of a user. The attacker "sandwiches" a victim's transaction between two of their own.

Let's consider a concrete example on a DEX with an AMM liquidity pool.

- **The Scenario:** A user, Alice, wants to swap 100,000 USDC for a low-liquidity token, M-Token. She submits her transaction to the mempool.
- **The Attacker's Play:** An MEV bot belonging to an attacker, Bob, spots Alice's large transaction in the mempool. It knows that this large buy order will cause a significant price increase for M-Token. The bot's goal is to profit from this price movement.
- **Step 1: The Front-run (Buy).** Bob's bot immediately creates and submits a buy order for M-Token with a slightly higher gas fee than Alice's transaction. Because of the higher fee, this transaction is confirmed and executed first.
- **Step 2: The Victim's Transaction (The Sandwich).** Alice's original transaction for 100,000 USDC then executes. However, because of the price impact caused by Bob's front-running buy, Alice's trade now fills at a higher price than she expected, meaning she receives fewer M-Tokens.
- **Step 3: The Back-run (Sell).** Right after Alice's transaction is confirmed, Bob's bot submits a sell order for all the M-Token it just acquired, with a slightly lower gas fee than Alice's original transaction but still high enough to be included in the same block. This sell order takes advantage of the even higher price caused by Alice's large trade.

The result is that Alice received fewer tokens for her trade, and Bob has successfully profited from the slippage and price impact created by Alice's transaction, all within a single block. This is a classic example of a "negative MEV" strategy that directly harms users.

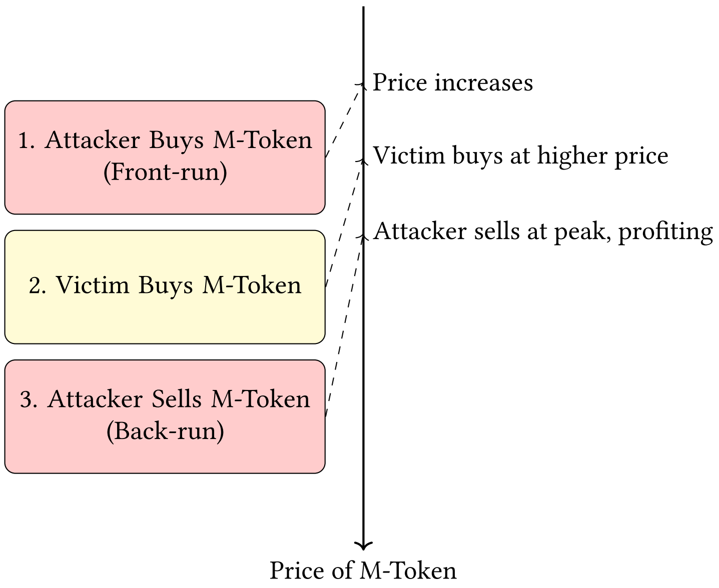

Price increases
1. Attacker Buys M-Token (Front-run)
Victim buys at higher price
Attacker sells at peak, profiting
2. Victim Buys M-Token
3. Attacker Sells M-Token (Back-run)
Price of M-Token

Fig. 2. Anatomy of a sandwich attack. The attacker's transactions (red) are placed before and after the victim's transaction (yellow) within the same block, allowing the attacker to profit from the price impact of the victim's trade.

Let's do the math for a sandwich attack: Assume a Uniswap v2-style CPMM pool with the following initial state:

- x (USDC reserve) = 50,000,000 USDC
- y (M-Token reserve) = 500,000 M-Token
- Initial price: 1 USDC = 0.01 M-Token (or 1 M-Token = 100 USDC)
- The constant product invariant $k = x \times y = 50,000,000 \times 500,000 = 2.5 \times 10^{13}$

Now, let's walk through the steps of an attack on a victim's trade of 1,000,000 USDC for M-Token.

(1) **Attacker's Front-run (Buy):** The attacker, Bob, sees Alice's 1,000,000 USDC transaction. Bob decides to front-run with a 100,000 USDC buy order.
   - Bob's input: $\Delta x_{Bob1} = 100,000$ USDC
   - New USDC reserve: $x_1 = 50,000,000 + 100,000 = 50,100,000$
   - New M-Token reserve: $y_1 = k/x_1 = 2.5 \times 10^{13}/50,100,000 \approx 499,001.996$
   - Bob's output: $\Delta y_{Bob1} = 500,000 - 499,001.996 = 998.004$ M-Token
   - Bob's average price: $100,000/998.004 \approx 100.2$ USDC per M-Token

(2) **Victim's Trade:** Alice's transaction for 1,000,000 USDC now executes. The pool's state is now x_1 and y_1.
   - Alice's input: $\Delta x_{Alice} = 1,000,000$ USDC
   - New USDC reserve: $x_2 = 50,100,000 + 1,000,000 = 51,100,000$
   - New M-Token reserve: $y_2 = k/x_2 = 2.5 \times 10^{13}/51,100,000 \approx 489,236.79$
   - Alice's output: $\Delta y_{Alice} = 499,001.996 - 489,236.79 = 9,765.206$ M-Token

- Alice's average price: $1,000,000/9,765.206 \approx 102.4$ USDC per M-Token, a significantly worse price than she would have received without the front-run

(3) **Attacker's Back-run (Sell):** Bob now sells the 998.004 M-Token he acquired back into the pool. For this transaction, we need to consider M-Token as the input and USDC as the output.
   - Initial state for the sell: M-Token reserve = $y_2 = 489,236.79$, USDC reserve = $x_2 = 51,100,000$
   - Bob's input: $\Delta y_{Bob2} = 998.004$ M-Token
   - Bob's output using the AMM formula:

$$\Delta x_{Bob2} = \frac{\Delta y_{Bob2} \cdot x_2}{y_2 + \Delta y_{Bob2}} = \frac{998.004 \times 51,100,000}{489,236.79 + 998.004} \approx 102,159.21 \text{ USDC} \quad (1)$$

(4) **Calculate Attacker's Profit:**
   - Bob's initial capital (for the front-run): 100,000 USDC
   - Bob's final capital (from the back-run): 102,159.21 USDC
   - Gross Profit: $102,159.21 - 100,000 = 2,159.21$ USDC
   - Assuming a total gas cost of 10 USDC for the two transactions (front-run and back-run), Bob's net profit would be: $2,159.21 - 10 = 2,149.21$ USDC

This calculation illustrates how a sandwich attacker can profit directly from the price impact of a victim's trade.

2.3 Liquidation MEV: A Necessary Evil

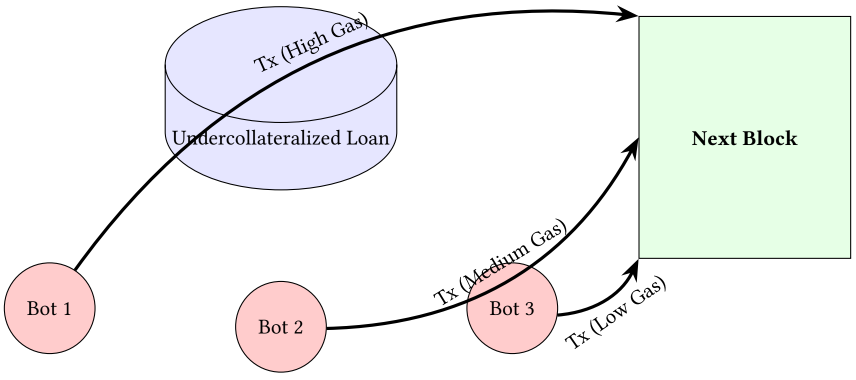
In decentralized lending protocols like Aave, MEV takes the form of **liquidation hunting**. As discussed in the previous lecture, a borrower's position becomes eligible for liquidation if their Health Factor falls below a critical threshold. Liquidators, often sophisticated bots, are incentivized to repay a portion of the debt in exchange for a portion of the borrower's collateral at a discount (the **liquidation bonus**).

This process is a form of MEV because liquidators are in a competitive race to be the first to call the liquidation function. When multiple liquidator bots spot the same opportunity, they engage in a high-stakes gas war, bidding against each other to ensure their transaction is included in the next block. The winner gets the liquidation bonus, while the losers' transactions fail. This MEV is essential for the health of lending protocols as it ensures that under-collateralized loans are closed quickly, protecting the protocol from bad debt.

2.4 Oracle Latency MEV

Blockchain oracles are crucial for bringing off-chain data, like real-world asset prices, to smart contracts. This is a vital but slow process, as oracle updates must be propagated, verified, and included in a block. The delay between a price change in the real world and a smart contract's knowledge of that price is known as **oracle latency**.

Oracle latency MEV is a form of MEV that exploits this delay. An attacker can see a pending oracle update transaction in the mempool. They can then use this information to front-run the update. For example, if an attacker sees an oracle transaction that will drastically increase the price of an asset, they can quickly take out a large loan using that asset as collateral, profiting from the outdated, lower price before the new price is confirmed on-chain. This attack is often a part of larger oracle manipulation exploits, and it highlights a critical vulnerability in the bridge between the on-chain and off-chain worlds.



Liquidator bots race to be the first to liquidate the loan, engaging in a gas war to get their transaction included in the next block.

Fig. 3. The competitive nature of liquidation MEV. Multiple bots detect an opportunity and submit transactions simultaneously, with the winner being the one who pays the highest gas fee to the block producer.

3 THE BROADER IMPACT OF MEV: PROS AND CONS

While MEV is most often discussed in the context of negative user outcomes, its impact on the ecosystem is more nuanced.

- **Benefits (The "Pros"):**
  - **Price Efficiency:** Arbitrage MEV, where bots automatically re-balance prices across different DEXs, helps to keep market prices consistent and efficient.
  - **Protocol Solvency:** Liquidation MEV is a powerful incentive mechanism that ensures lending protocols remain solvent by quickly liquidating bad debt.
  - **Network Incentives:** The revenue from MEV can be a significant part of a block producer's income, incentivizing them to secure the network.
- **Drawbacks (The "Cons"):**
  - **User Harm:** Front-running and sandwich attacks directly harm users by causing increased slippage and financial losses.
  - **Network Congestion:** The fierce competition among MEV bots for profitable opportunities leads to "gas wars," which can artificially inflate gas fees and clog the network for all users.
  - **Centralization Risk:** The high-value nature of MEV can create a powerful incentive for centralization. Only a few sophisticated, well-resourced entities (searchers and block builders) have the technical expertise and capital to consistently extract MEV, giving them an outsized influence on the network.

4 CASE STUDY: ANTI-HACKING WITH MEV (SENTINEL)

While MEV is often viewed as a force for harm, its fundamental mechanism, the ability to reorder and front-run transactions, can be repurposed for defense. The Sentinel framework is a perfect example of this.

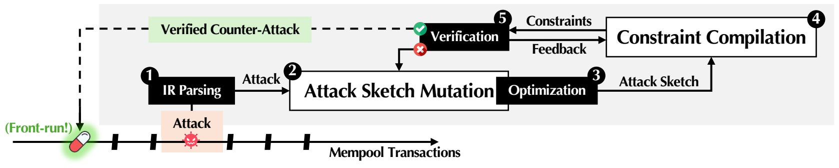


Fig. 4. Framework overview of Sentinel.

4.1 The Challenge of On-chain Exploits

Smart contracts, by their nature, are immutable. Once a vulnerability is deployed, it cannot be easily patched or corrected. When a hacker discovers such a vulnerability, they can execute a malicious transaction to drain funds from a protocol. Once this "exploit transaction" is sent, it sits in the mempool for a few seconds, waiting to be included in the next block. This brief window is the only chance to stop a devastating attack.

4.2 Sentinel: A Real-Time Counter-Attack Engine

The Sentinel framework is designed to be a "last line of defense" against these in-flight exploits. Its core innovation is that it uses MEV's own mechanics to fight back against the attacker.

Let's consider a concrete example:

- **The Scenario:** A hacker, Eve, finds a vulnerability in a lending protocol and crafts a transaction to drain a large sum of funds. This malicious transaction is broadcast to the public mempool.
- **Detection:** The Sentinel system, which is constantly monitoring the mempool, detects Eve's transaction and identifies it as a known exploit pattern.
- **Analysis and Synthesis:** In a matter of milliseconds, Sentinel performs a deep analysis of Eve's transaction. It understands the attacker's objective and the sequence of steps required to execute the exploit. The system then automatically synthesizes a new "counter-attack" smart contract designed to neutralize the exploit and rescue the funds.
- **The Front-run (Defense):** Sentinel then bundles its newly synthesized counter-attack transaction with a very high gas fee. This bundle is submitted to a private transaction relay, which guarantees that it will be included in the next block *before* Eve's original transaction.
- **Neutralization and Recovery:** Because Sentinel's transaction is confirmed first, it is able to execute its defense logic, for example, it might transfer all the vulnerable funds to a secure, community-controlled "rescue vault." By the time Eve's original transaction is confirmed, the funds she was targeting are no longer there, and the exploit is neutralized.

This case study demonstrates the immense value of MEV as a defensive tool. By using the same tools and tactics as attackers, namely, the ability to front-run and reorder transactions, Sentinel can turn a would-be disaster into a successful recovery, illustrating a crucial and highly beneficial use case for a mechanism often considered a purely negative force.