#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 [**? ?**]. As shown in Figure 1, a block producer has the power to observe this public pool and reorder transactions to their own benefit.

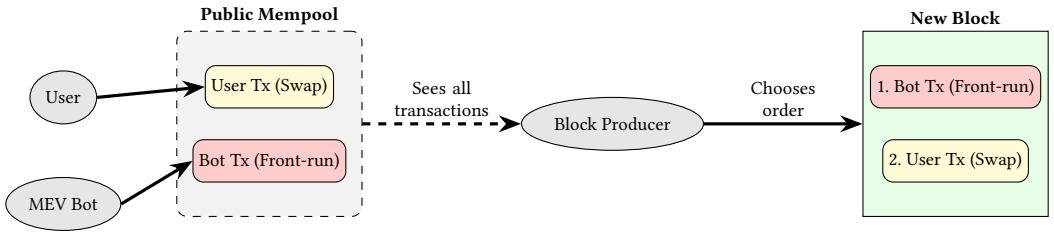

Fig. 1. The MEV transaction lifecycle. Transactions from all actors enter the public mempool. The Block Producer observes this pool and has the authority to reorder transactions (e.g., placing the Bot's transaction first) to maximize profit when creating the new block.

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

process is visualized in Figure 2. This is a classic example of a "bad MEV" strategy that directly harms users.

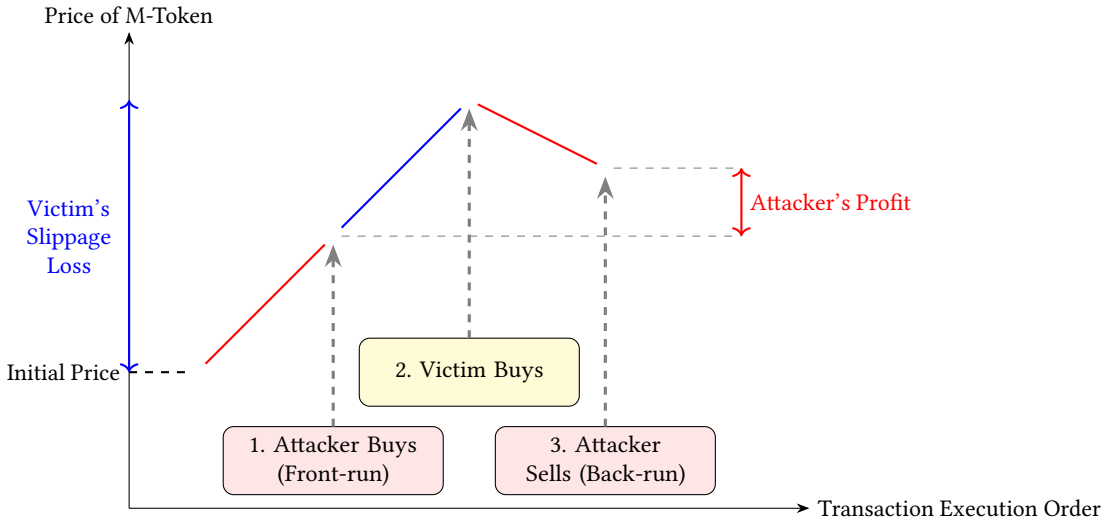

Fig. 2. Anatomy of a sandwich attack, showing the price impact of each transaction. The attacker profits by buying before the victim (at a low price) and selling after the victim (at a high price), directly causing the victim to suffer worse price execution (slippage) [**?** **?**].

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 (see Figure 3). 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.

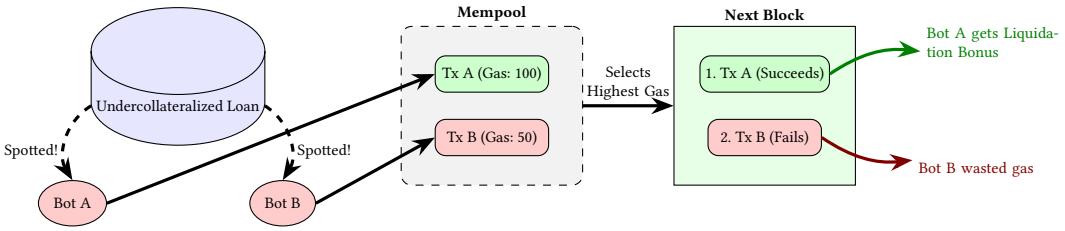

Fig. 3. The competitive nature of liquidation MEV. Multiple bots detect the same opportunity and submit transactions to the mempool. They engage in a gas war (bidding with gas fees) to be first. The block producer selects the highest-paying transaction (Tx A), which succeeds. Subsequent transactions (Tx B) fail, as the loan is already liquidated.

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. Oracles do not update instantly; they typically update on

a time-based schedule (e.g., once every 30 minutes) or after a certain price deviation (e.g., the price changes by > 1%).

This delay between a price change in the real world and the smart contract's *knowledge* of that change is known as **oracle latency**.

Oracle latency MEV is a form of attack that exploits this delay, often with devastating consequences. The 2022 collapse of LUNA and UST provides a catastrophic, real-world example.

2.4.1 Case Study: The LUNA De-peg and Oracle Latency. During the LUNA price crash, the real-world price on centralized exchanges (CEXs) like Binance was falling far faster than the on-chain oracles (like the ones used by the Anchor protocol) were updating. This created a massive, profitable gap between perception and reality.

Here is the step-by-step exploit that attackers used:

- **The Scenario:**
  - **Real-World Price (on Binance):** 1 LUNA = $0.10
  - **On-Chain Oracle Price (Stale):** A lending protocol's oracle has not updated yet. It still reports that 1 LUNA = $1.00
- **The Attack (MEV):** An attacker sees this 90% price discrepancy.
  (1) **Step 1: Buy Cheap Off-Chain.** The attacker goes to Binance and buys 1 million LUNA for $100,000 (1M * $0.10).
  (2) **Step 2: Deposit On-Chain at Stale Price.** The attacker immediately deposits this 1 million LUNA into the on-chain lending protocol.
  (3) **Step 3: Exploit Stale Price.** The protocol's smart contract consults its oracle, which still reads the *stale price* of $1.00. The protocol thus incorrectly values the attacker's collateral at $1,000,000 (1M * $1.00).
  (4) **Step 4: Borrow Against Fake Value.** The protocol allows the attacker to borrow stablecoins (e.g., USDC) against this "valuable" collateral. The attacker borrows the maximum allowed, for example, $500,000 in USDC.
- **The Result:** The attacker walks away with $500,000 in real money (USDC), leaving behind collateral (LUNA) that is only worth $100,000 in the real world. When the oracle *finally* updates to the new, crashed price, the lending protocol is left with a massive $400,000 in bad debt that will never be repaid. The attacker successfully extracted this value directly from the protocol's treasury, enabled entirely by oracle latency.

2.4.2 Front-running the Update. A related, more traditional form of this attack is when an MEV bot sees a pending oracle update transaction (e.g., a transaction from Chainlink that is about to update the price) in the public mempool.

If the bot sees the update will drastically *increase* an asset's price, it will execute a transaction before the update (a front-run) to take out a large loan using that asset as collateral, profiting from the outdated, lower price. Both of these are attacks on the "information delay" that is fundamental to oracles.

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: MEV MITIGATION STRATEGIES

Given the significant drawbacks of malicious MEV, particularly user harm and network congestion, a large field of research and engineering is dedicated to mitigating it. These strategies can be broadly categorized into chain-level and application-level solutions.

4.1 Chain-Level and Protocol-Level Solutions

These solutions aim to change the fundamental rules of the blockchain or the transaction supply chain to reduce harmful MEV.

4.1.1 Private Transaction Relays (e.g., Flashbots Protect RPC). The most widely adopted solution today is the use of private transaction relays, pioneered by organizations like Flashbots.

- **How it works:** Instead of broadcasting a transaction to the public mempool, a user (or their wallet) sends it directly to a private relay (e.g., the Flashbots Protect RPC). This relay forwards the transaction to a specialized block builder who has an agreement with block producers.
- **Benefit:** The transaction is **hidden** from public MEV bots. This provides strong protection against front-running and sandwich attacks, as attackers in the public mempool never see the transaction until it is already included in a block.
- **Limitation:** The user must trust the relay and the builder not to front-run or sandwich the transaction themselves. However, the reputation and economic incentives of major builders are aligned to prevent this.

4.1.2 Proposer-Builder Separation (PBS). This is a more fundamental, long-term protocol change (the "vision" for Ethereum) designed to formalize and democratize the MEV market by fundamentally restructuring the block production supply chain, as illustrated in Figure 4.

- **Why PBS is Needed:** Before PBS, the validator who is chosen to propose a block (the Proposer) is also responsible for *building* that block. To maximize profit, this proposer must run sophisticated, resource-intensive software to find and extract all available MEV. This creates a massive **economy of scale**.
  - Only large, well-resourced staking pools can afford the hardware, engineering talent, and low-latency network infrastructure to be competitive at MEV extraction.
  - A small solo-staker cannot compete. Their blocks will be less profitable because they are missing out on this MEV revenue.

  - This creates a powerful centralizing force: users will preferentially stake with large pools to get higher returns, and solo stakers are economically non-viable, pushing the entire validator set towards centralization.
- **How it works:** PBS breaks this centralizing link by **separating the roles**. The Proposer and Builder roles are split into two distinct, specialized actors.
  (1) **Builders (Specialists):** These are highly-specialized, competitive entities. Their only job is to use their sophisticated algorithms to build the most profitable block possible (packed with MEV). They then submit a bid to the proposer, which is essentially the total value they are willing to pay for the right to have their block included.
  (2) **Proposers (Validators):** The proposer's job becomes much simpler. They no longer need to find MEV. They just listen for bids from all the competing builders and select the single *most profitable block* (i.e., the one with the highest bid). They include that block's header, pay themselves the bid, and know nothing about the block's contents.
- **Benefit:** This separation of concerns directly promotes decentralization and market efficiency.
  - **Prevents Validator Centralization:** A solo staker (Proposer) can now be just as profitable as the largest staking pool. Neither of them builds the block; they both just accept the winning bid from the open market. This **democratizes access to MEV revenue** and removes the economic pressure for validators to centralize.
  - **Maximizes Revenue for Stakers:** It creates a hyper-competitive, open market for *block building*. Builders are forced to be maximally efficient and to bid almost all of their extracted MEV value (minus their operational costs) directly to the proposer. This ensures the maximum value is channeled back to the protocol's stakers, rather than being kept by a few dominant, vertically-integrated validators.

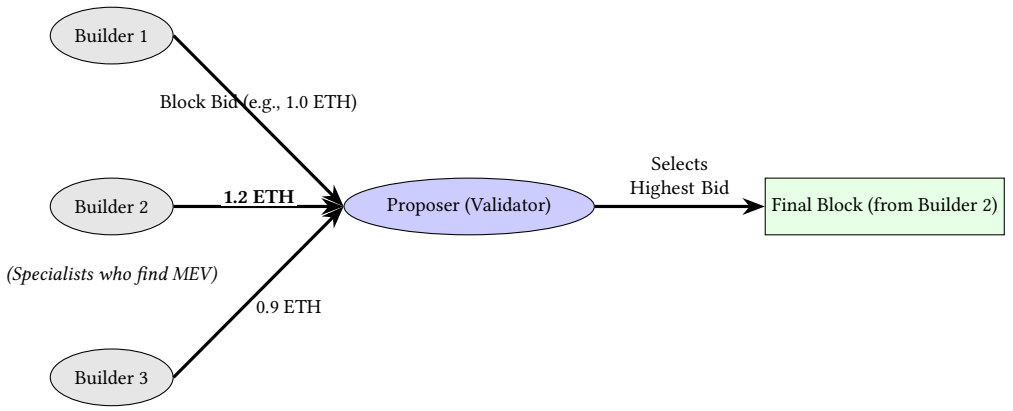

Fig. 4. The Proposer-Builder Separation (PBS) workflow. Specialized Builders compete to build the most profitable block. The Proposer simply selects the block with the highest bid, democratizing access to MEV revenue without needing to run complex MEV-extraction software.

It's critical to understand that PBS **does not stop MEV**. Instead, it starts from a simple truth: **extracting MEV is very difficult**.

Without PBS, this difficulty means only a few highly-skilled actors (the "smart attackers" or specialized pools) can successfully find and capture all this profit. This makes them rich and powerful, forcing everyone else to join them, which leads to centralization.

PBS's solution is to **acknowledge MEV exists** and change the rules. It creates an open auction where all these "smart" actors (Builders) are forced to compete against each other. To win the right to build a block, they must bid their *entire* potential profit to the Proposer.

This system cleverly **redistributes the profits**: the money is moved from the few specialists who find the MEV to *all* the validators who secure the network.

4.2 Application-Level Solutions

These are strategies that decentralized applications (dApps), especially DEXs, can implement to protect their own users from malicious MEV.

4.2.1 Slippage Tolerance. This is the most common, user-facing protection, acting as a defensive tripwire.

- **How it works:** When a user makes a swap, the DEX interface requires them to set a slippage tolerance (e.g., 0.5%). This is a strict instruction in the smart contract: I only agree to this trade if the final price I get is no worse than 0.5% from the price I was quoted. If the price moves against me more than that, **fail my transaction**.
- **Benefit:** This setting directly attacks the profitability of a sandwich attack.
  - A sandwich attack requires three steps: (1) Attacker buys, (2) Victim buys (at a worse price), (3) Attacker sells.
  - With slippage tolerance, if the attacker's Step 1 (the front-run) pushes the price beyond the user's 0.5% limit, the user's Step 2 trade **fails and reverts**.
  - This breaks the sandwich. The attacker is now stuck with the tokens from Step 1 and has no victim to sell to at a high price. The attack fails, and the attacker loses money on gas.
- **Limitation:** This is not a perfect fix. It just sets an upper bound on the *amount that can be sandwiched/stolen*. Attackers are fully aware of this setting and simply design their bots to extract the maximum value possible *without* triggering the slippage limit. If your tolerance is 0.5%, the bot will try to steal 0.49% from you.

4.2.2 Batch Auctions (e.g., CoW Swap). Instead of executing trades instantly and sequentially, some DEXs use a batch auction model. This fundamental change in mechanism is contrasted with a standard AMM in Figure 5.

- **How it works:** This model fundamentally breaks from a standard AMM.
  - In a normal DEX, trades execute one-by-one. If your trade is second, you get the price *after* the first trade has already impacted the pool. This sequential execution is what makes front-running possible.
  - In a batch auction, the protocol collects all user trades from a short time window (e.g., 30 seconds) into a single "batch." A third-party "solver" then looks at all trades in the batch *at once* and calculates a single, uniform clearing price for everyone.
- **Benefit:** This design makes transaction **ordering within the batch irrelevant**. An attacker cannot front-run you by placing their trade "first," because "first" doesn't matter. Everyone in the batch gets the same price regardless of the order their transactions were submitted. This provides powerful protection from front-running and sandwich attacks.

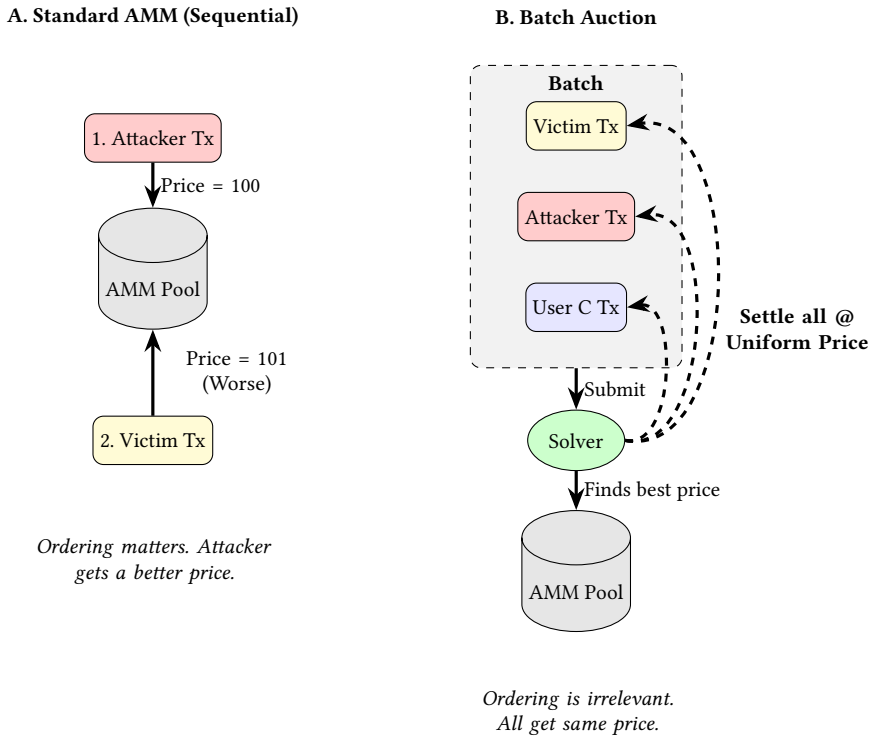

Fig. 5. Comparison of execution models. (A) In a standard AMM, sequential execution allows an attacker to go first and worsen the price for the victim. (B) In a batch auction, all transactions are collected, given to a Solver who finds the best price (e.g., from an AMM), and then settles all at one uniform price, neutralizing front-running.