# Blockchain Economy issues: Maximal Extractable Value (MEV)

Semia Guesmi

08-06-2024

PRIN NiRvAna: Noninterference and Reversibility Analysis in Private Blockchains

# Our aim

- The DeFi ecosystem involves complex interactions and dependencies between protocols. These protocols have inherent vulnerabilities that malicious actors can exploit to economically harm the compound service.

- We focus solely on the contract, verifying its security, and identifying any potentially risky instructions within it.

- We identify MEV opportunities within the smart contract using a formal modeling framework.

# MEV: Maximal Extractable Value

- MEV represents the maximum potential gain that users, including miners and validators, can achieve through interactions with a smart contract and its associated dependencies in a malicious way.


# MEV: A Serious Risk to Blockchain Security

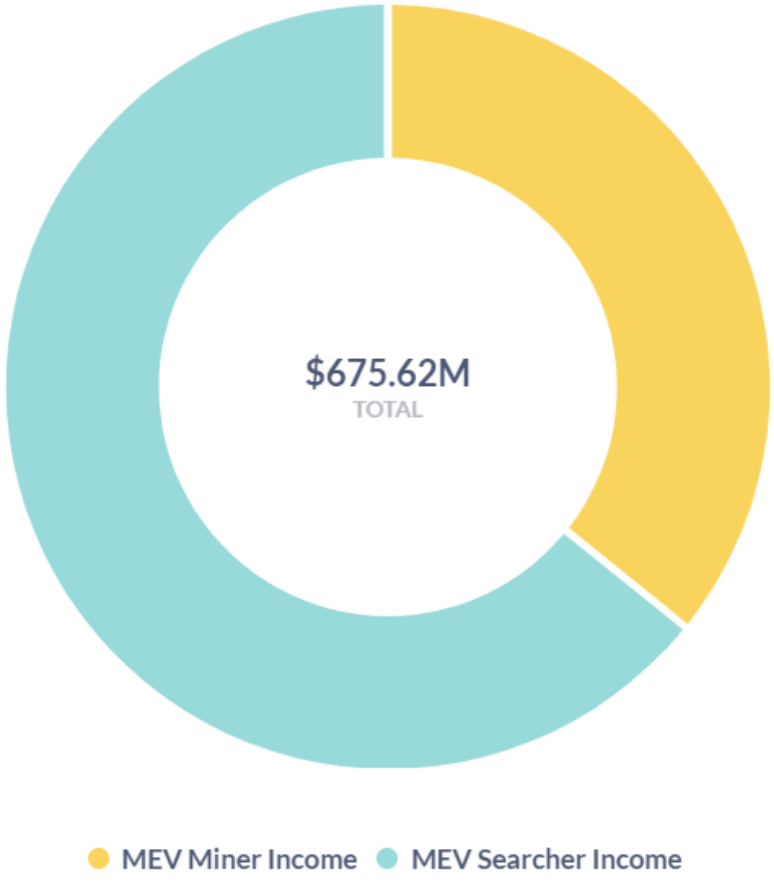
- By the start of 2021, the cumulative value of MEV extracted on Ethereum reached $78m, which then shot up to $554m by the end of the year. By the end of 2022, MEV extracted on Ethereum stands at over $686m.

Ref: https://chain.link/

# MEV Actors

**Block producer (Miners / Validators)**

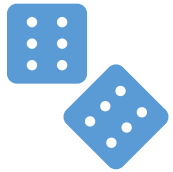
- Transaction manipulation




Extracted MEV Split by Role

**Searchers**

- Run complex algorithms
- Generalized frontrunners (bots)



# MEV vulnerabilities

**Arbitrage**:  Miners can exploit price differences across different decentralized exchanges (DEXs) by inserting their transactions ahead of others.

**Front-Running**: Miners observe pending transactions in the mempool and insert their transactions before high-value trades to profit from price movements.

**Back-running:** Miners observe pending transactions in the mempool and insert their transactions after high-value trades to profit from price movements.

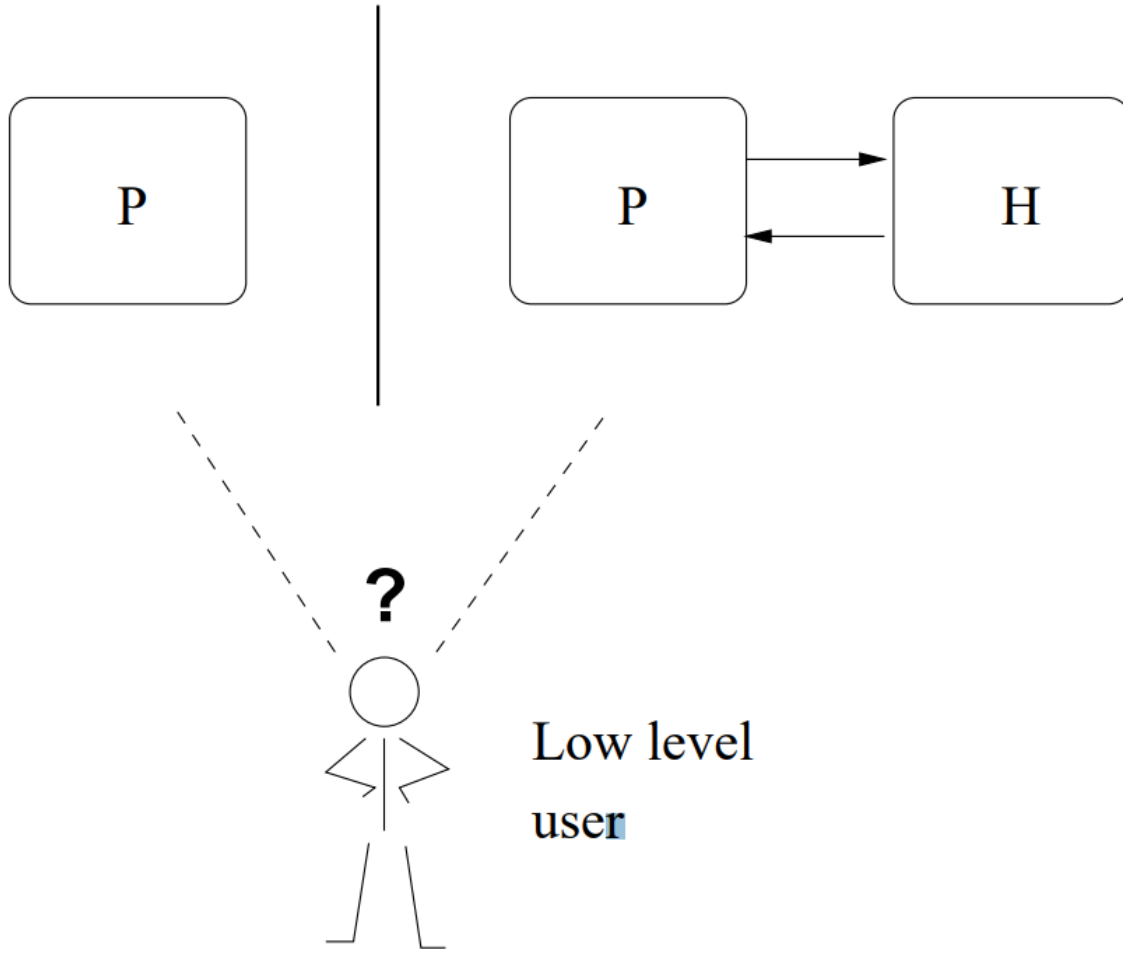
**Sandwich Attacks**: Involves placing one transaction before and one after a victim's transaction to manipulate market prices and extract value.

**Liquidation**: Miners can capitalize on liquidations in DeFi lending protocols by ensuring their liquidation transactions are processed first.

**Smart Contract Design**: Certain DeFi protocols may unintentionally enable MEV opportunities due to their logic and transaction flows.

# Noninterference

- Noninterference aims to capture unwanted information flows in multi-level systems.
- The notion of confidentiality: High and low levels.
- A flow of information from high to low could represent the public disclosure of private data.



# State of the art

**Adversary perspective:** Secure if the global MEV does not significantly increase.

**Global MEV** $MEV(S) = \max\left\{ gain_{Adv}\ (S, \underline{X}) \;\middle|\; \underline{X} \in K(Adv)^* \right\}$

$\Delta$ interacts safely with S $\leftrightarrow$ $MEV\ (S|\Delta) \leq (1 + \varepsilon)\ MEV(S)$

$(\varepsilon - composability, see\ Clockwork\ Finance\ bt\ Babel, Daian, Kelkar, Juels)$

**Contract perspective:** Secure if being in a composition does not cause loss.

**Local MEV** $MEV(S, \Delta) = \max\left\{ loss_{\Delta}\ (S, \underline{X}) \;\middle|\; \underline{X} \in K(Adv)^* \right\}$

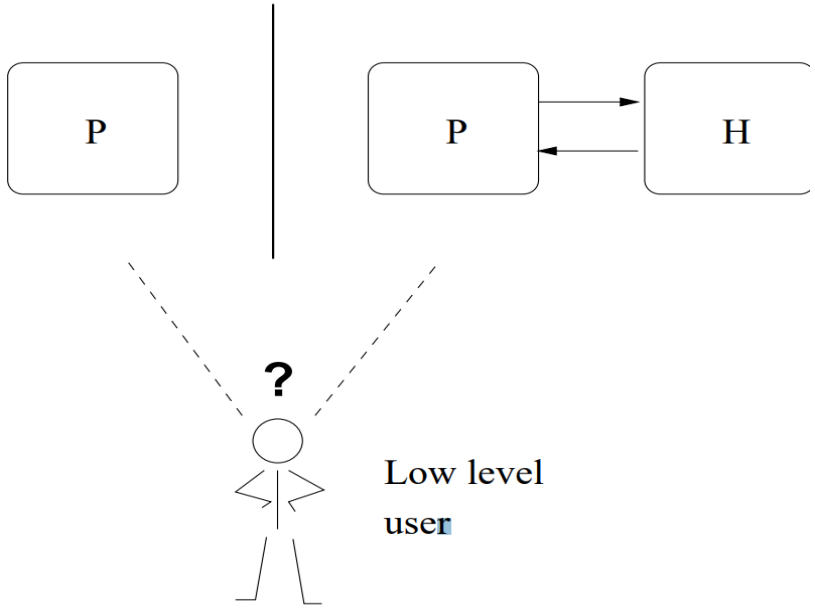
S does not interfere with the new contract $\Delta$ if: $MEV\ (S \mid \Delta\ , \Delta) = MEV_{\Delta}\ (S \mid \Delta\ , \Delta)$

$(DeFi\ composability\ as\ MEV\ non - interference\ Bartoletti, Marchesin, Roberto\ )$

# Generalized Unwinding Condition $\mathcal{W}(\doteq, \mathcal{R}, \div)$

**Contract perspective in Computational framework:**

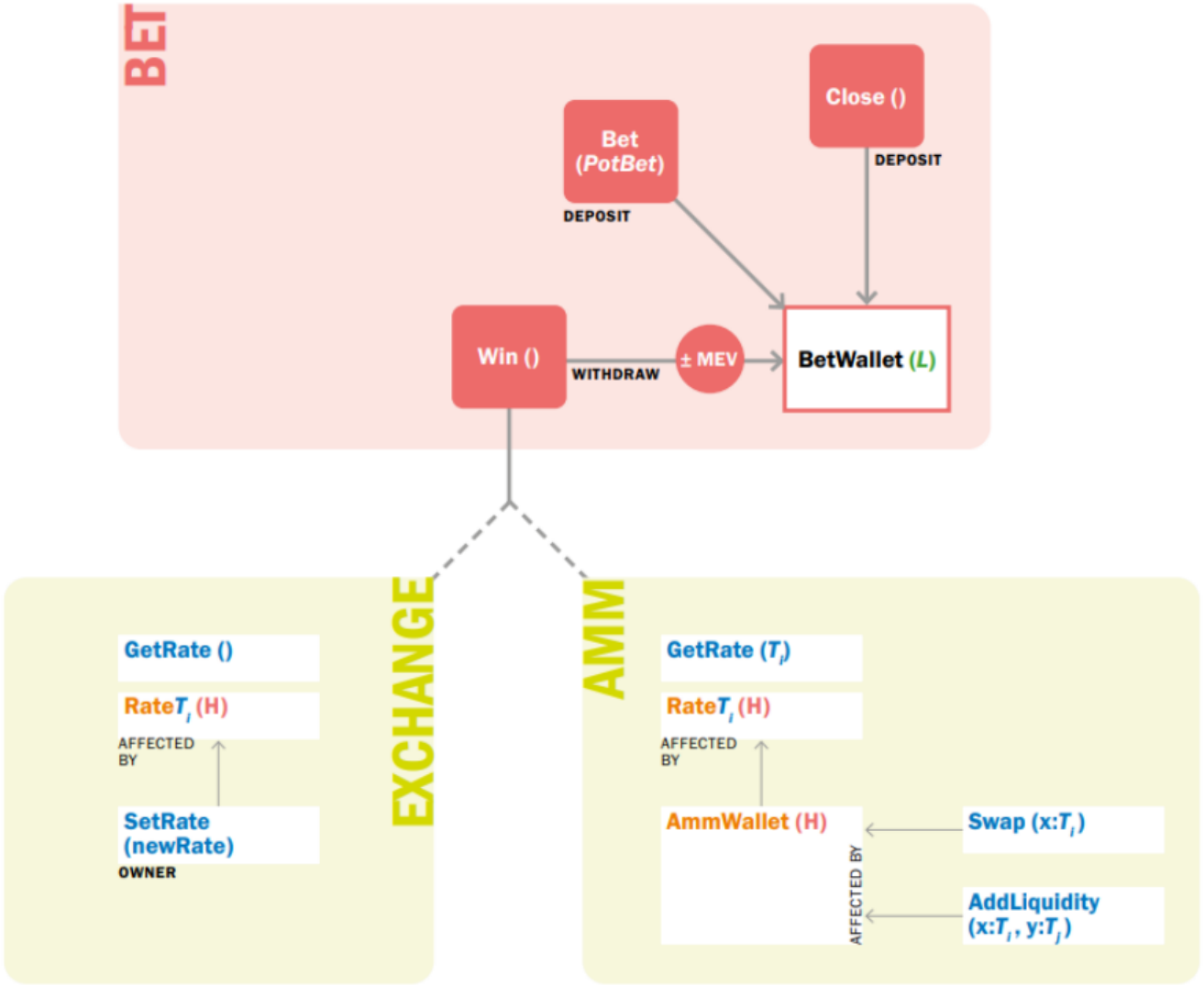
- ✓ formalizing noninterference through unwinding conditions to analyze MEV.
- ✓ Guarantees that any reachable state resulting from high-level interactions still maintains indistinguishability with respect to low-level observations.




$$\begin{array}{ccccc} \langle F, \psi \rangle & \xrightarrow{\text{high}} & \langle G, \varphi \rangle & \rightsquigarrow & \langle \text{end}, \varphi' \rangle \\ \updownarrow & & & & \updownarrow \\ \pi =_l \psi & & & & \varphi' =_l \rho' \\ \updownarrow & & & & \updownarrow \\ \langle F, \pi \rangle & \rightsquigarrow & \cdots & \rightsquigarrow & \langle \text{end}, \rho' \rangle \end{array}$$

A pictorial representation of the unwinding condition

$Bet\_Contract \equiv \mathbf{co}\ Constructor \mid Bet \mid Win \mid Close\ \mathbf{oc}$ (**imperative language**)




# Example: The Bet Contract

$$S = \mathtt{M}[310\colon \mathtt{ETH}] \mid \mathtt{AMM}[600\colon \mathtt{ETH}, 600\colon \mathtt{T}] \mid \mathtt{block.num} = n - k \mid \cdots$$
$$\Delta = \mathtt{Bet}[10\colon \mathtt{ETH}, \mathtt{tok} = \mathtt{T}, \mathtt{rate} = 3, \mathtt{owner} = \mathtt{A}, \mathtt{deadline} = n]$$

The adversary M is rich enough, she can fire the following sequence:

$$S \mid \Delta \xrightarrow{\mathtt{M:Bet.bet(?\,10:ETH)}} \mathtt{M}[300\colon \mathtt{ETH}] \mid \mathtt{AMM}[600\colon \mathtt{ETH}, 600\colon \mathtt{T}] \mid \mathtt{Bet}[20\colon \mathtt{ETH}, \cdots]$$
$$\xrightarrow{\mathtt{M:AMM.swap(?\,300:ETH,0)}} \mathtt{M}[200\colon \mathtt{T}] \mid \mathtt{AMM}[900\colon \mathtt{ETH}, 400\colon \mathtt{T}] \mid \mathtt{Bet}[20\colon \mathtt{ETH}, \cdots]$$
$$\xrightarrow{\mathtt{M:Bet.win()}} \mathtt{M}[20\colon \mathtt{ETH}, 200\colon \mathtt{T}] \mid \mathtt{AMM}[900\colon \mathtt{ETH}, 400\colon \mathtt{T}] \mid \mathtt{Bet}[0\colon \mathtt{ETH}, \cdots]$$
$$\xrightarrow{\mathtt{M:AMM.swap(?\,200:T,0)}} \mathtt{M}[320\colon \mathtt{ETH}] \mid \mathtt{AMM}[600\colon \mathtt{ETH}, 600\colon \mathtt{T}] \mid \mathtt{Bet}[0\colon \mathtt{ETH}, \cdots]$$

$$Bet \,\&\, Amm \;\equiv\; \boldsymbol{co}\; Bet\_Contract \mid Amm\_Contract\; \boldsymbol{oc}$$

$$\langle Bet \,\&\, Amm, \sigma \rangle$$
$$\downarrow$$
$$\ldots \;\%\; execution\; of\; Swap \quad (\mathrm{H})$$
$$\downarrow$$
$$\langle Bet\&Amm', \sigma\, [AmmWalletEther/900, AmmWalletT2/400] \rangle$$
$$\downarrow$$
$$\ldots \;\%\; execution\; of\; GateRate$$
$$\downarrow$$
$$\langle Bet, \sigma\, [AmmWalletEther/900, AmmWalletT2/400, AmmRateEther/2.25] \rangle$$
$$\downarrow$$
$$\ldots \;\%\; execution\; of\; Win$$
$$\downarrow$$
$$\langle end, \sigma\, [AmmWalletEther/900, AmmWalletT2/400, AmmRateEther/2.25, \boldsymbol{BetWallet/0}] \rangle$$

$$\langle Bet \ \& \ Amm, \sigma \rangle$$
$$\downarrow$$
$$\ldots \ \% \ execution \ of \ Win$$
$$\downarrow$$
$$\langle Amm, \sigma \ [SenderWin/`NULL'\rangle$$
$$\downarrow$$
$$\ldots \ \% \ execution \ of \ GateRate$$
$$\downarrow$$
$$\langle end, \sigma \ [SenderWin/`NULL', AmmWalletEther/600, AmmWalletT2/600, \mathbf{BetWallet/10}]\rangle$$

- ✓ Identify the precise instructions and variables within the code that could potentially lead to information flows.
- ✓ Identify the specific dependencies of the contract that require deeper analysis.

# Arbitrage Example

```
contract LPArbitrage_{c0,c1,lp} {
  constructor() {
    (t0,t1)=lp.getTokens();
    require lp.getToken()==t0 && c1.getTokens()==(t0,t1);
  }
  arbitrage(x) {
    lp.borrow(x);          // borrow x:t0
    c0.swap(?x:t0,0);      // sell t0, buy t1
    c1.swap(?#t1:t1,0);    // sell t1, buy t0
    lp.repay(?x:t0);       // repay x:t0
    require #t0>0;         // gain is positive
    sender!#t0:t0          // transfer gain to sender
  }
}
```

A contract to arbitrage with a Lending Pool.

$$\boldsymbol{\sigma}\,[W_{ARB}T0/0\ ,W_{ARB}T1/0\,]$$

$$\boldsymbol{Arbitrage}\ \equiv Borrow\,;Swap;Swap;Repay\,;if\ (W_{ARB}T0>0)\ \{Transfer\}\,;$$

## ❑ Convenient exchange

$\langle \boldsymbol{Arbitrage},\boldsymbol{\sigma}\,\rangle$

↓

… % *execution of* $LP.Borrow(10:T0)$

↓

$\langle\ \boldsymbol{Arbitrage'},\boldsymbol{\sigma}[W_{ARB}T0\ /\mathbf{10}\ ,W_{ARB}T1/\mathbf{0}\,]\,\rangle$

↓

… % *execution of* $AMM1.Swap(10:T0)$

↓

$\langle\boldsymbol{Arbitrage''},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\mathbf{0}\ ,W_{ARB}T1/\mathbf{12}\,]\,\rangle$

↓

… % *execution of* $AMM2.Swap(12:T1)$

↓

$\langle\boldsymbol{Arbitrage'''},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\mathbf{13}\ ,W_{ARB}T1/\mathbf{0}]\,\rangle$

↓

… % *execution of* $LP.Repay(10:T0)$

↓

$\langle\boldsymbol{Arbitrage'''},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\ \boldsymbol{13-10}\ ,W_{ARB}T1/\mathbf{0}]\,\rangle$

↓

… % *execution of* $Sender.Transfer(\boldsymbol{W_{ARB}T0}:T0)$

↓

$\langle\boldsymbol{End},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\ \boldsymbol{0}\ ,W_{ARB}T1/\mathbf{0}]\,\rangle$

↑ **Gain**

## ❑ Inconvenient exchange

$\langle \boldsymbol{Arbitrage},\boldsymbol{\sigma}\,\rangle$

↓

… % *execution of* $LP.Borrow(10:T0)$

↓

$\langle\ \boldsymbol{Arbitrage'},\boldsymbol{\sigma}[W_{ARB}T0\ /\mathbf{10}\ ,W_{ARB}T1/\mathbf{0}\,]\,\rangle$

↓

… % *execution of* $AMM1.Swap(10:T0)$

↓

$\langle\boldsymbol{Arbitrage''},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\mathbf{0}\ ,W_{ARB}T1/\mathbf{12}\,]\,\rangle$

↓

… % *execution of* $AMM2.Swap(12:T1)$

↓

$\langle\boldsymbol{Arbitrage'''},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\mathbf{9}\ ,W_{ARB}T1/\mathbf{0}]\,\rangle$

↓

… % *execution of* $LP.Repay(10:T0)$

↓

$\langle\boldsymbol{Arbitrage''''},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\ 9-10\ ,W_{ARB}T1/0]\,\rangle$

↓

… % *execution of* $Sender.Transfer(W_{ARB}T0:T0)$

↓

$\langle\boldsymbol{End},\boldsymbol{\sigma}\ [W_{ARB}T0\ /\boldsymbol{9-10},W_{ARB}T1/\mathbf{0}]\,\rangle$

# Await operator as a guarantee

```
  arbitrage(x) {
    lp.borrow(x);
    c0.swap(?x:t0,0);
    c1.swap(?#t1:t1,0);
    lp.repay(?x:t0);
    require #t0>0;
    sender!#t0:t0
  }
}
```

$$\boldsymbol{Arbitrage} \equiv \boldsymbol{await}(\boldsymbol{C0}.\boldsymbol{getRate}(\boldsymbol{T0}) * \boldsymbol{C1}.\boldsymbol{getRate}(\boldsymbol{T0}) > \boldsymbol{1})\{$$
$$Borrow;$$
$$Swap;$$
$$Swap;$$
$$Repay;$$
$$if\ (W_{ARB}T0 > 0)\ \{Transfer\};$$
$$\}$$

# Downgrading Mechanism

- Differentiate a secure scenario from the potentially risky one.

- This mechanism enables explicit allowance of delimited flows, providing an approach to managing information flow within the system.

- The "$\boldsymbol{downgrade}()$" function **declassifies** high-level variables, thereby reducing its sensitivity to a lower-level variable. Consequently, rendering it not considered as dangerous in the unwinding test.

$$P \equiv H := 0;\ D := \boldsymbol{downgrade}(H);\ if(D > 0)\{L := D\}\ else\ \{skip\}$$

# Example: Implementing the Downgrading Mechanism in the Win Program

```
1:  Program WIN
2:      while (Deadline > BlockNum) do
3:          await ( SenderWin = Player ) do
4:              skip
5:          if (Player ≠ ExchangeOwner) then
6:              CurrentRate:=downgrade(ExchangeRate);
7:          else
8:              CurrentRate:=0;
9:
10:         if (BetRate < CurrentRate) then
11:             PlayerWalletEther := PlayerWalletEther + BetWallet;
12:             BetWallet := 0;
13:         else
14:             Player := 'NULL'
```

Bet Contract: price oracle (**Exchange**)

*Win* Program Demonstrates Non-interference within $\mathcal{W}(\doteq, \mathcal{R}, \div)$ Framework

# Unwinding conditions for security in imperative languages

$$\frac{}{\langle \mathtt{skip}, \sigma\rangle \xrightarrow{\mathrm{low}} \langle \mathrm{end}, \sigma\rangle}$$

$$\frac{\langle a, \sigma\rangle \to n}{\langle X := a, \sigma\rangle \xrightarrow{\epsilon} \langle \mathrm{end}, \sigma[X/n]\rangle}\ a \in \epsilon$$

$$\frac{\langle P_0, \sigma\rangle \xrightarrow{\epsilon} \langle P_0', \sigma'\rangle}{\langle P_0; P_1, \sigma\rangle \xrightarrow{\epsilon} \langle P_0'; P_1, \sigma'\rangle}\ P_0' \not\equiv \mathrm{end}$$

$$\frac{\langle P_0, \sigma\rangle \xrightarrow{\epsilon} \langle \mathrm{end}, \sigma'\rangle}{\langle P_0; P_1, \sigma\rangle \xrightarrow{\epsilon} \langle P_1, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \mathrm{true}}{\langle \mathtt{if}(b)\ \{P_0\}\ \mathtt{else}\ \{P_1\}, \sigma\rangle \xrightarrow{\epsilon} \langle P_0, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \mathrm{false}}{\langle \mathtt{if}(b)\ \{P_0\}\ \mathtt{else}\ \{P_1\}, \sigma\rangle \xrightarrow{\epsilon} \langle P_1, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \mathrm{true}}{\langle \mathtt{while}(b)\ \{P\}, \sigma\rangle \xrightarrow{\epsilon} \langle P; \mathtt{while}(b)\ \{P\}, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \mathrm{false}}{\langle \mathtt{while}(b)\ \{P\}, \sigma\rangle \xrightarrow{\epsilon} \langle \mathrm{end}, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle b, \sigma\rangle \to \mathrm{true} \quad \langle S, \sigma\rangle \overset{\epsilon_2}{\rightsquigarrow} \langle \mathrm{end}, \sigma'\rangle}{\langle \mathtt{await}(b)\ \{S\}, \sigma\rangle \xrightarrow{\epsilon_1 \cup \epsilon_2} \langle \mathrm{end}, \sigma'\rangle}\ b \in \epsilon_1$$

$$\frac{\langle b, \sigma\rangle \to \mathrm{false}}{\langle \mathtt{await}(b)\ \{S\}, \sigma\rangle \xrightarrow{\epsilon} \langle \mathtt{await}(b)\ \{S\}, \sigma\rangle}\ b \in \epsilon$$

$$\frac{\langle P_i, \sigma\rangle \xrightarrow{\epsilon} \langle P_i', \sigma'\rangle}{\langle \mathtt{co}\ P_1 | \ldots | P_i | \ldots | P_n\ \mathtt{oc}, \sigma\rangle \xrightarrow{\epsilon} \langle \mathtt{co}\ P_1 | \ldots | P_i' | \ldots | P_n\ \mathtt{oc}, \sigma'\rangle}$$

$$\frac{}{\langle \mathtt{co}\ \mathrm{end} | \ldots | \mathrm{end} | \ldots | \mathrm{end}\ \mathtt{oc}, \sigma\rangle \xrightarrow{\mathrm{low}} \langle \mathrm{end}, \sigma\rangle}$$

(*Compositional Information Flow Security for Concurrent Programs Annalisa, Carla, Sabina,* )

# Future Work

- In depth investigating the relationships between Unwinding Conditions and MEV, and to implement this methodology.
- Applying this method to analyze other case studies involving MEV attacks.
- Define this framework on fragments of languages for smart contracts, such as solidity.

# THANKS FOR THE ATTENTION.