Elements of DeFi

https://web3.princeton.edu/elements-of-defi/

Professor Pramod Viswanath

Princeton University

Lecture 3: Smart Contracts and Pricing

Last time

- Blockchains to track transfer of tokens, maintain a ledger

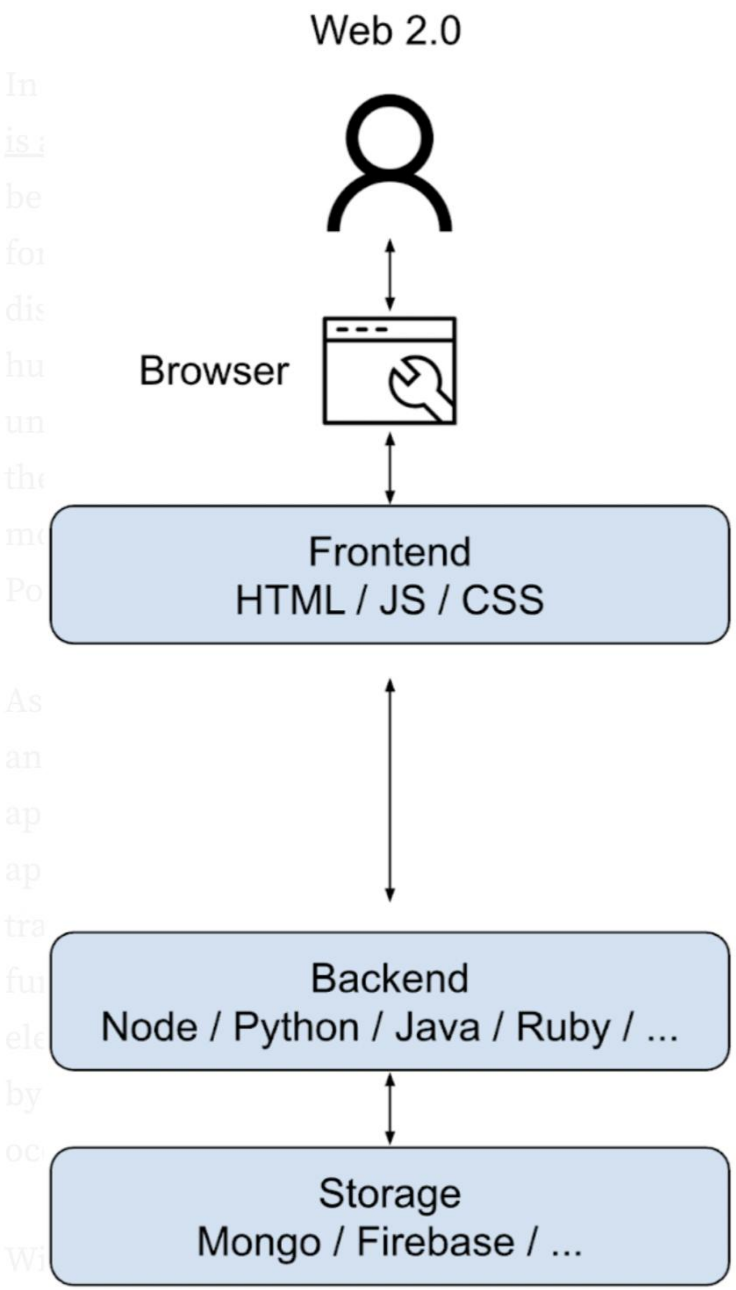
This time

- Blockchains to run programs, store information, transfer tokens,
- Tool used : “smart contracts”
  - computer programs that use the ledger entries as variables and memory
- A “transaction” is now generalized into a “function call”
  - smart contract
- Ledger update is managed via a virtual machine
  - Generalize the simple ledger to a “state machine”, a run-time environment for smart contract execution
  - Ethereum Virtual Machine
- Auctions for selling transaction slots

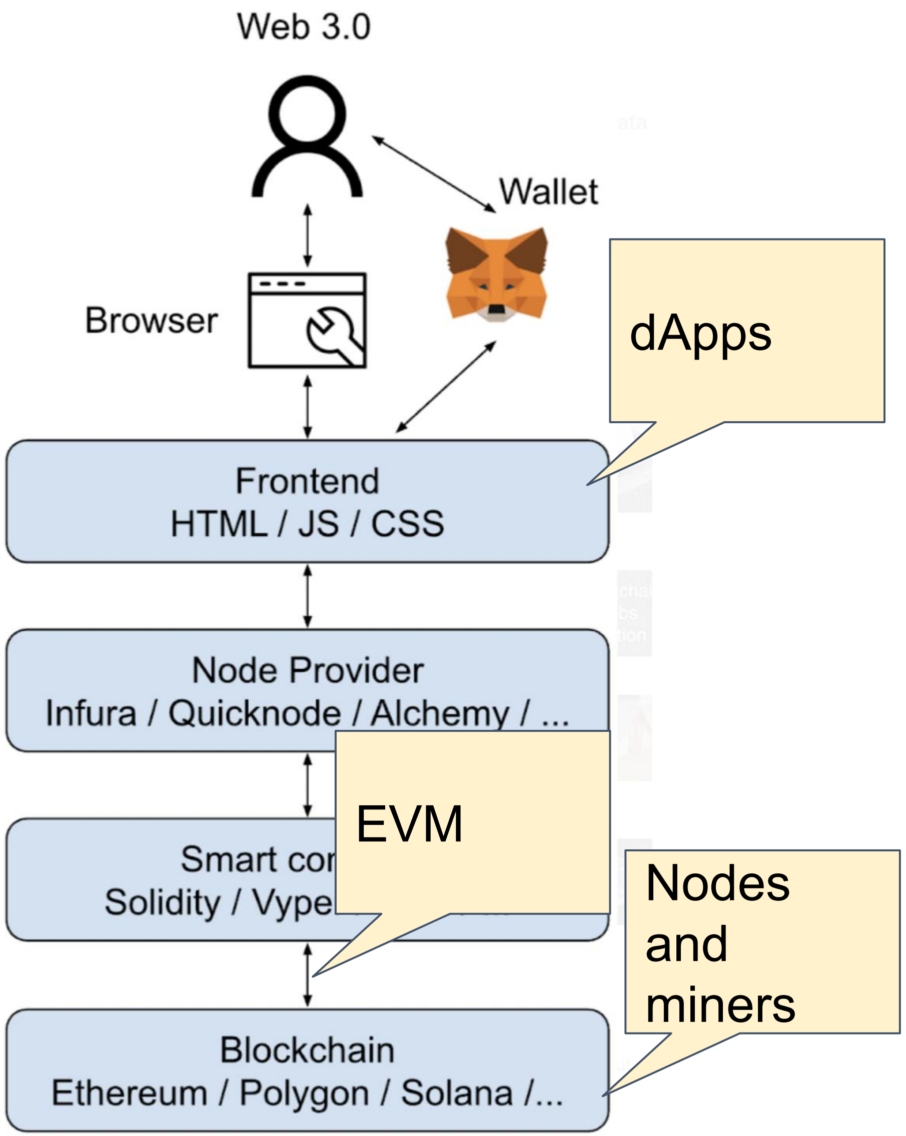
Outline

- What are smart contracts?
- How is smart contract programming different?
  - web 3.0 development
  - dApp programming
- EVM
  - opcodes, gas fees
- Solidity
- Pricing of transactions
- Today’s lab :
  - Fungible tokens: ERC20

Web 2.0 : a system view

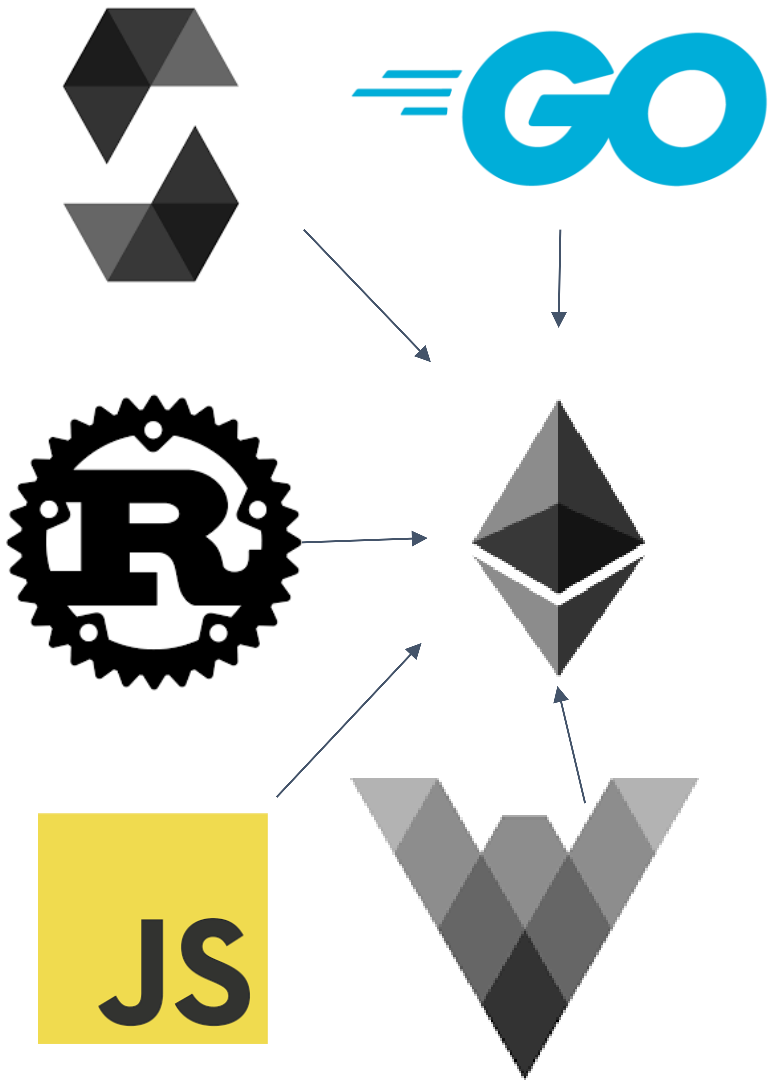


Web 3.0 : a system view



What is a Smart Contract?

- First proposed in 1990s,"digital form of promises"
- Not necessarily related to a contract
- Lifecycle:
  - Deployed by a transaction
  - Establish initial states, immutable once deployed
  - Store states and execute computations
- Languages: Solidity, Rust, JavaScript...



Where is the program stored?

1. Writing a program - using high-level language such as Solidity, Vyper, …
2. Program is compiled - into low-level language that the blockchain state machine can understand
3. Program is deployed - binary is stored in the state of the blockchain

What kind of a programming language should be used?

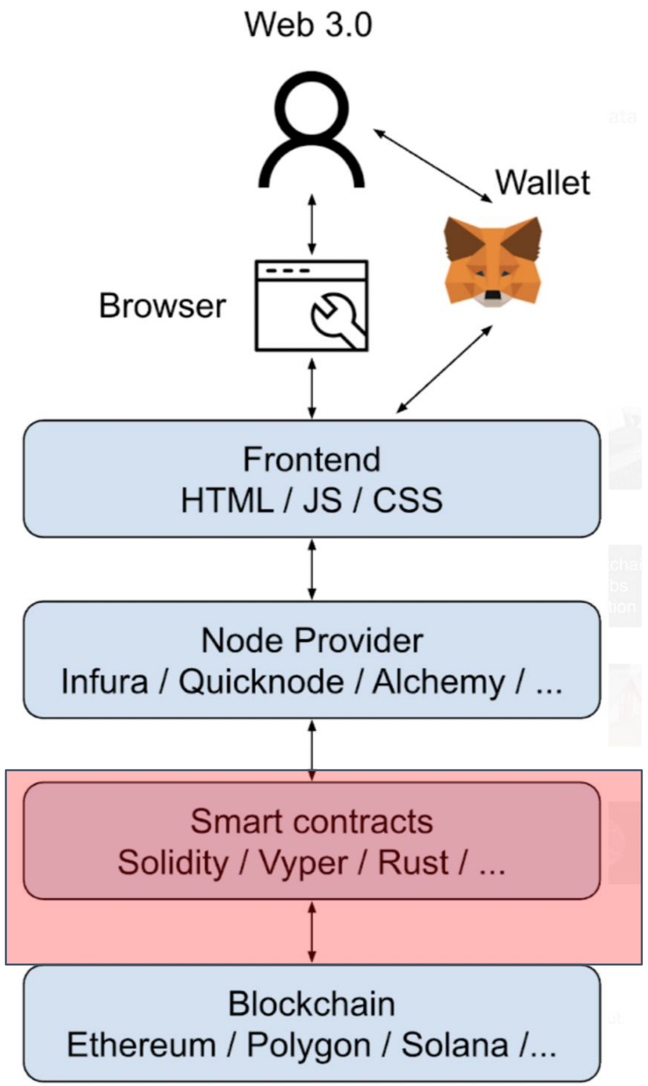
- Should be similar to existing languages in syntax
- Should have additional functionality to access blockchain state
- Every execution should yield the same result
    - so execution output can be verified by everyone else
    - rand() not allowed
    - function calls to outside (e.g., the internet) are not allowed
- Cost of execution (gas fee) should be readily calculated

Design goals as a developer

- Correctness –
    - especially important because of Byzantine actors - program forced to go into a “rare” corner case
    - stakes are high! - cannot change code once deployed
    - tokens handled have value - incentive for bad actors

- Efficiency –
    - one pays fees for each execution of the smart contract and this adds up on each execution

This lecture

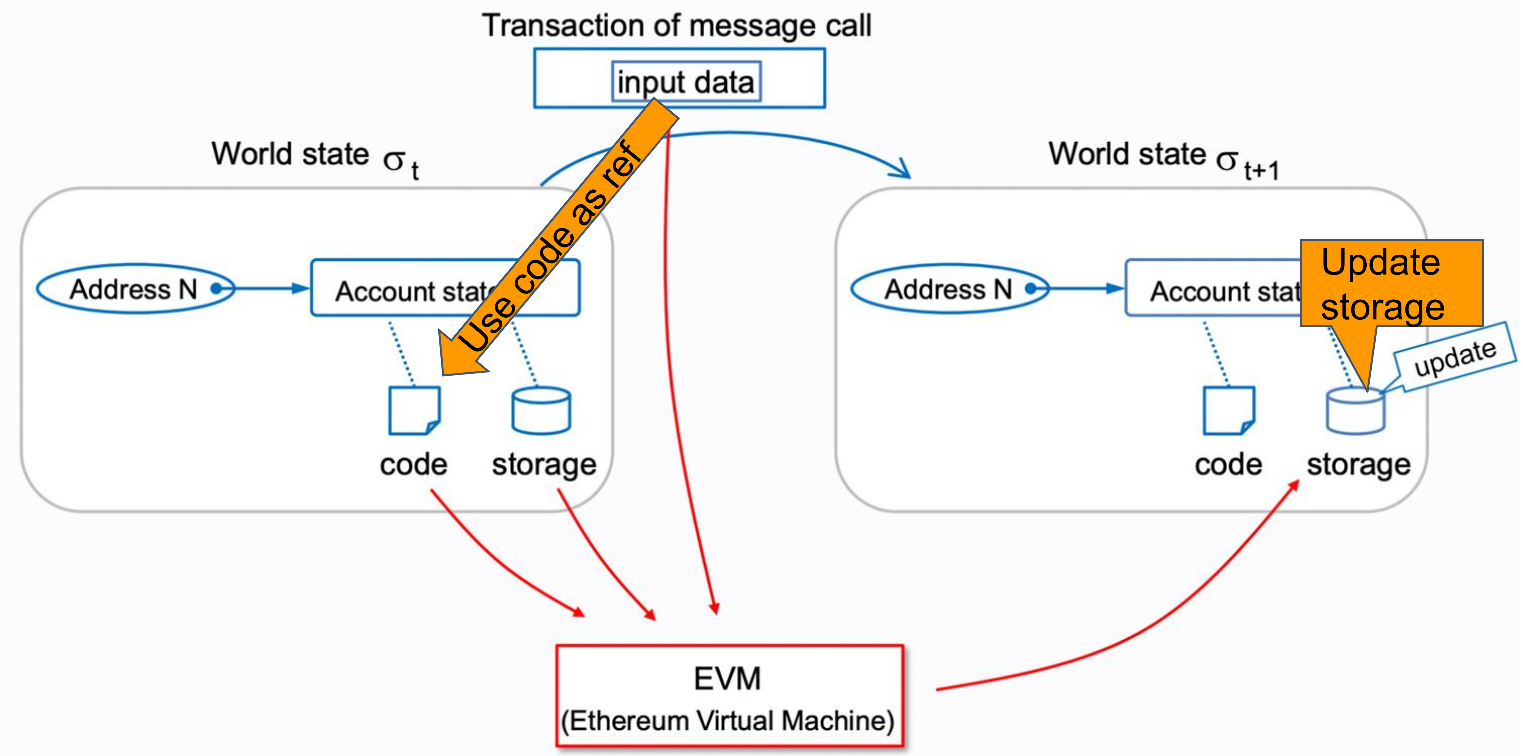


EVM

- The Ethereum Virtual Machine is the **distributed** execution environment (“state machine”) running on the Ethereum blockchain
- Each block on Ethereum changes the state of EVM
- Every Ethereum user sees the same canonical EVM state at any given block

EVM: state updates

- State of EVM changed via transactions :



EVM code is executed on Ethereum Virtual Machine (EVM).

EVM: transactions, opcodes

- Two types of transactions :
  - Resulting in function calls
  - Resulting in contract creation
- Every transaction is decomposed into a sequence of OPCODEs
  - e.g. ADD, SUB, JUMP, LOAD, …
  - fixed number (256) of opcodes
- Every OPCODE consumes a fixed amount of gas
  - total gas of a transaction is the sum of gas of constituent opcodes
  - gas to eth is a variable

EVM: gas

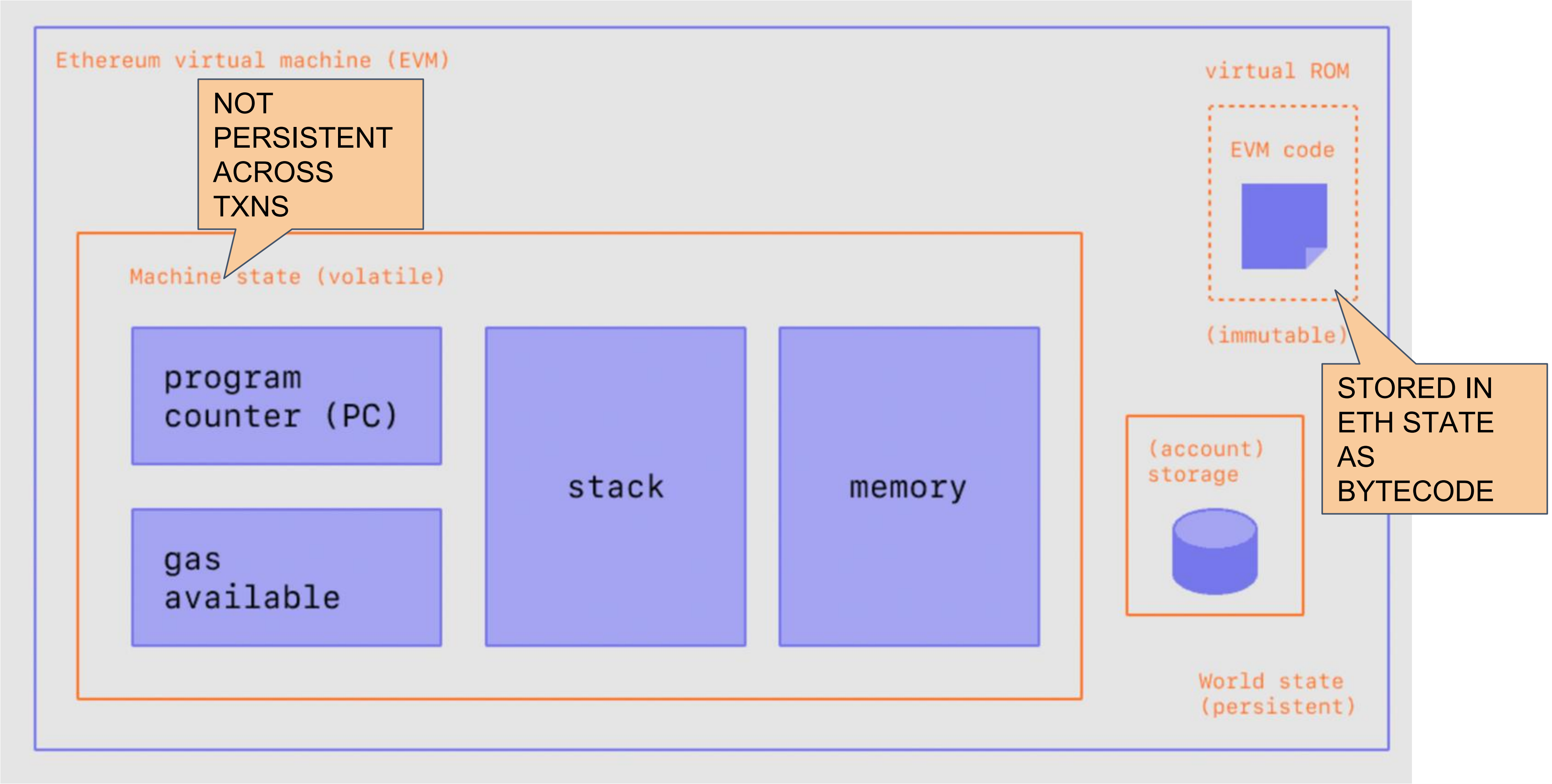
Table 1. EVM opcodes and gas cost

Opcode	Name	Description	Extra info	Gas
`0x00`	STOP	Halts execution	-	0
`0x01`	ADD	Addition operation	-	3
`0x02`	MUL	Multiplication operation	-	5
`0x03`	SUB	Subtraction operation	-	3
`0x04`	DIV	Integer division operation	-	5
`0x05`	SDIV	Signed integer division operation (truncated)	-	5
`0x06`	MOD	Modulo remainder operation	-	5
`0x07`	SMOD	Signed modulo remainder operation	-	5
`0x08`	ADDMOD	Modulo addition operation	-	8
		Modulo multiplication		

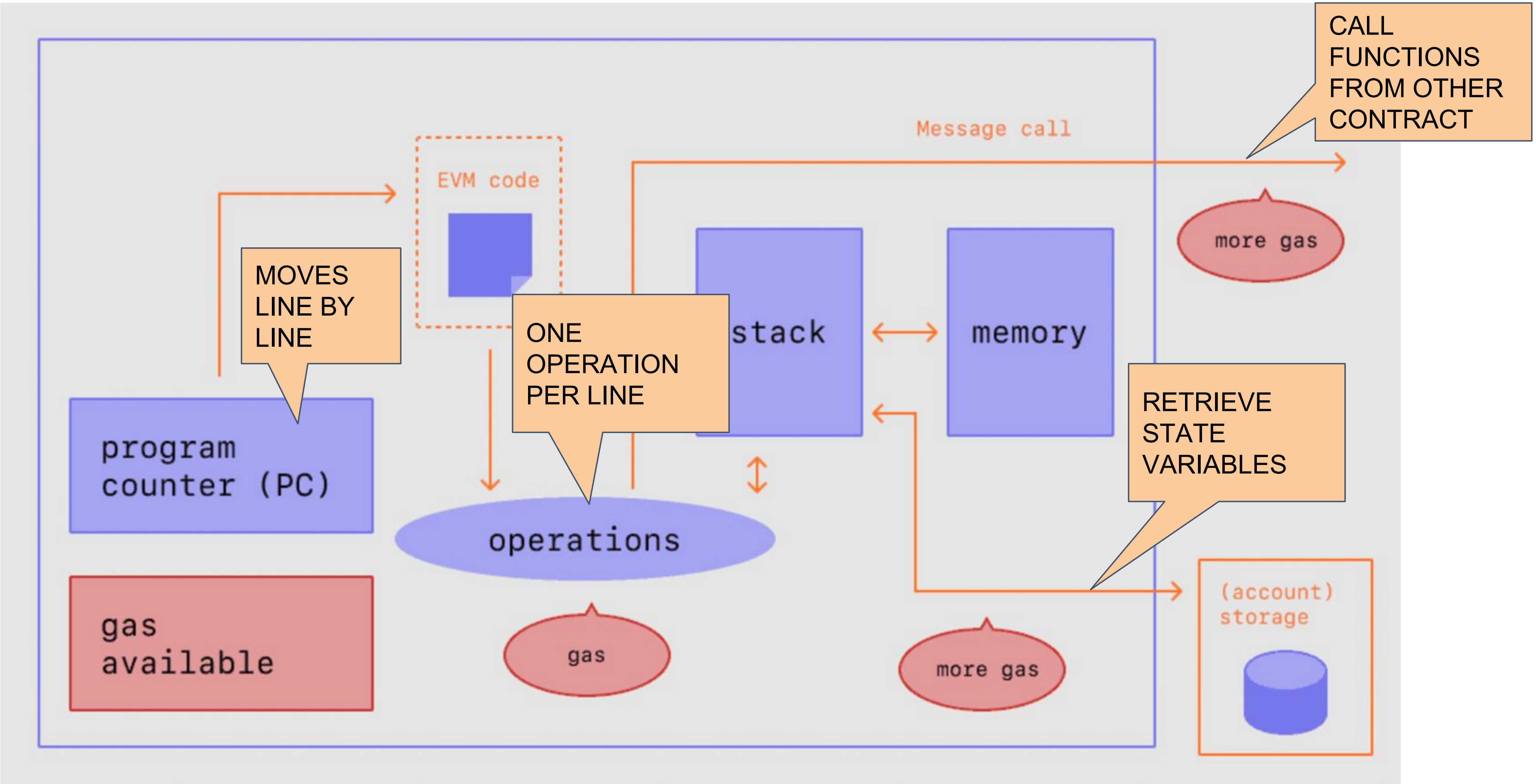
EVM: Data Store

- Storage
  - Written in the blockchain, stored permanently
  - Expensive, some gas is refunded when storage is deleted
- Memory
  - A byte array with slot sizes of 32 bytes
  - Stored during function execution
  - Cheap, but the costs per operation scales quadratically
  - Does not persist across txns
- Stack
  - Only 16 stack variables are accessible
  - Cheapest, manipulated by inline assembly

Tracing a transaction through the data store



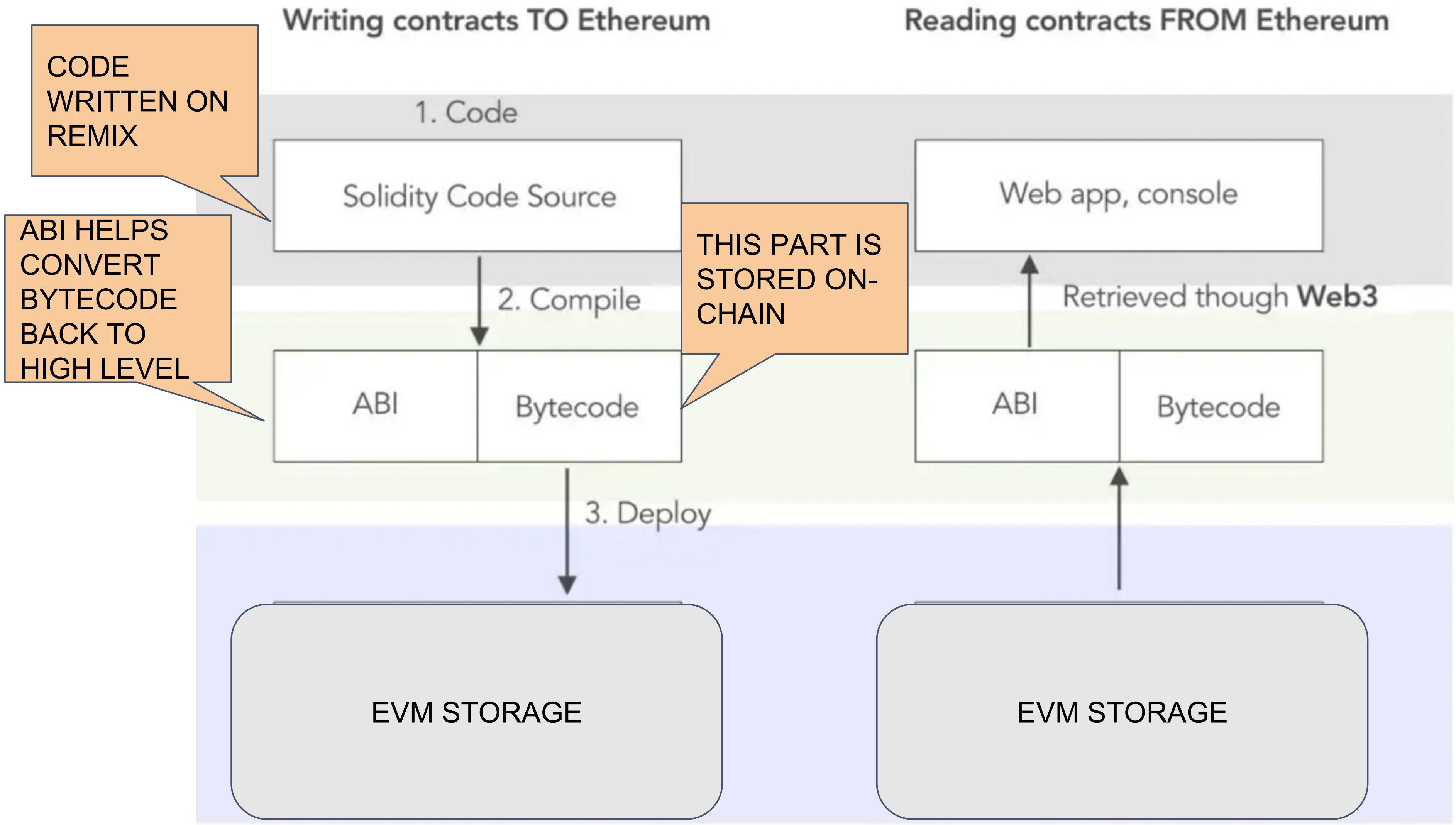
Tracing a transaction through the data store



Solidity

- Object-oriented, statically-typed
- Designed for Ethereum
  - Also used by Binance Smart Chain, Avalanche, XinFin…
- Turing-complete
- Popular IDE – remix (we use this in our lab, assignments)

Solidity : Interaction with EVM



Solidity ⟷ Bytecode ⟷ Opcode

```
60 00          =   PUSH1 0x00
35             =   CALLDATALOAD
60 e0          =   PUSH1 0xe0
1c             =   SHR
80             =   DUP1
63 2e64cec1    =   PUSH4 0x2e64cec1
14             =   EQ
61 003b        =   PUSH2 0x003b
57             =   JUMPI
80             =   DUP1
63 6057361d    =   PUSH4 0x6057361d
14             =   EQ
61 0059        =   PUSH2 0x0059
57             =   JUMPI
```

```
        return number;
    }
}
```

CONTRACT STORED AS BYTECODE

```
61
080fd5b      610075565b60405161
360048    0381019061006e91906100
00081905550505 65b60008135905061
312156100b3576100b26100fe565b5b
          0d3816100f4565b82525050
          15050565b60008190509190
11757600080fd5b5056fea264697066
2de3b991f93d230604b1b8daaef6476
```

INTERPRETED BY EVM

AT EXECUTION, BYTECODE CONVERTED TO OPCODES FOR THE EVM

Problem

- EVM Compute + Storage + Memory are scarce resources

  Therefore, gas is a scarce resource

- How should txn submitters bid for gas?
- Some kind of an auction? How should the auction be settled?
- We look at this question from the perspective of miner incentives

Design 1 : a first-price auction

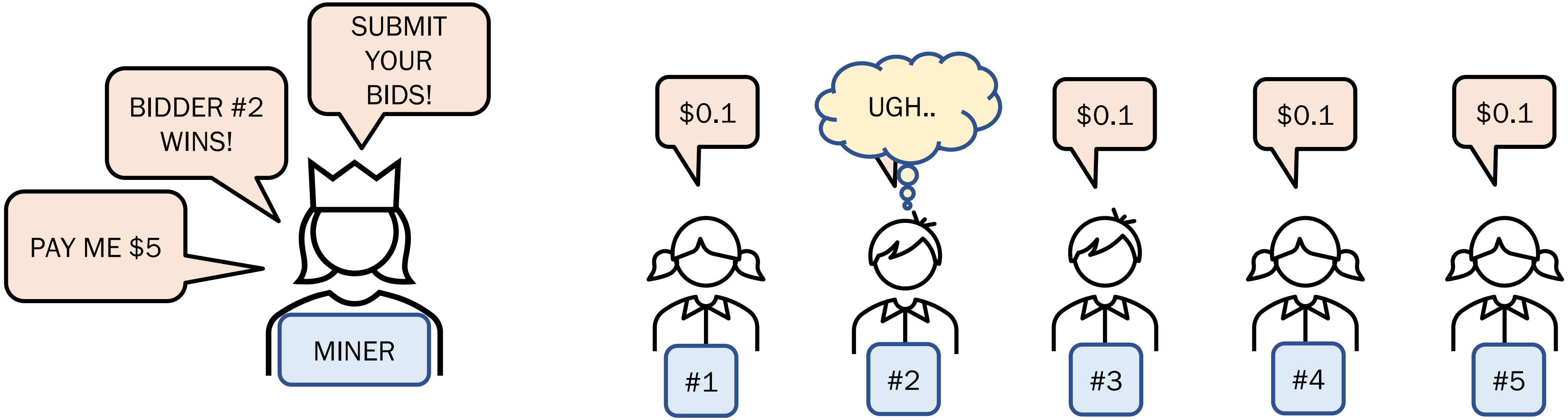
- (Implemented in Bitcoin and Ethereum basic)

1. Every txn submits a bid
2. Miner includes the N highest bids
3. Everyone pays fees equal to their bids
4. All of the money goes to the miner

- Called a “first-price auction”

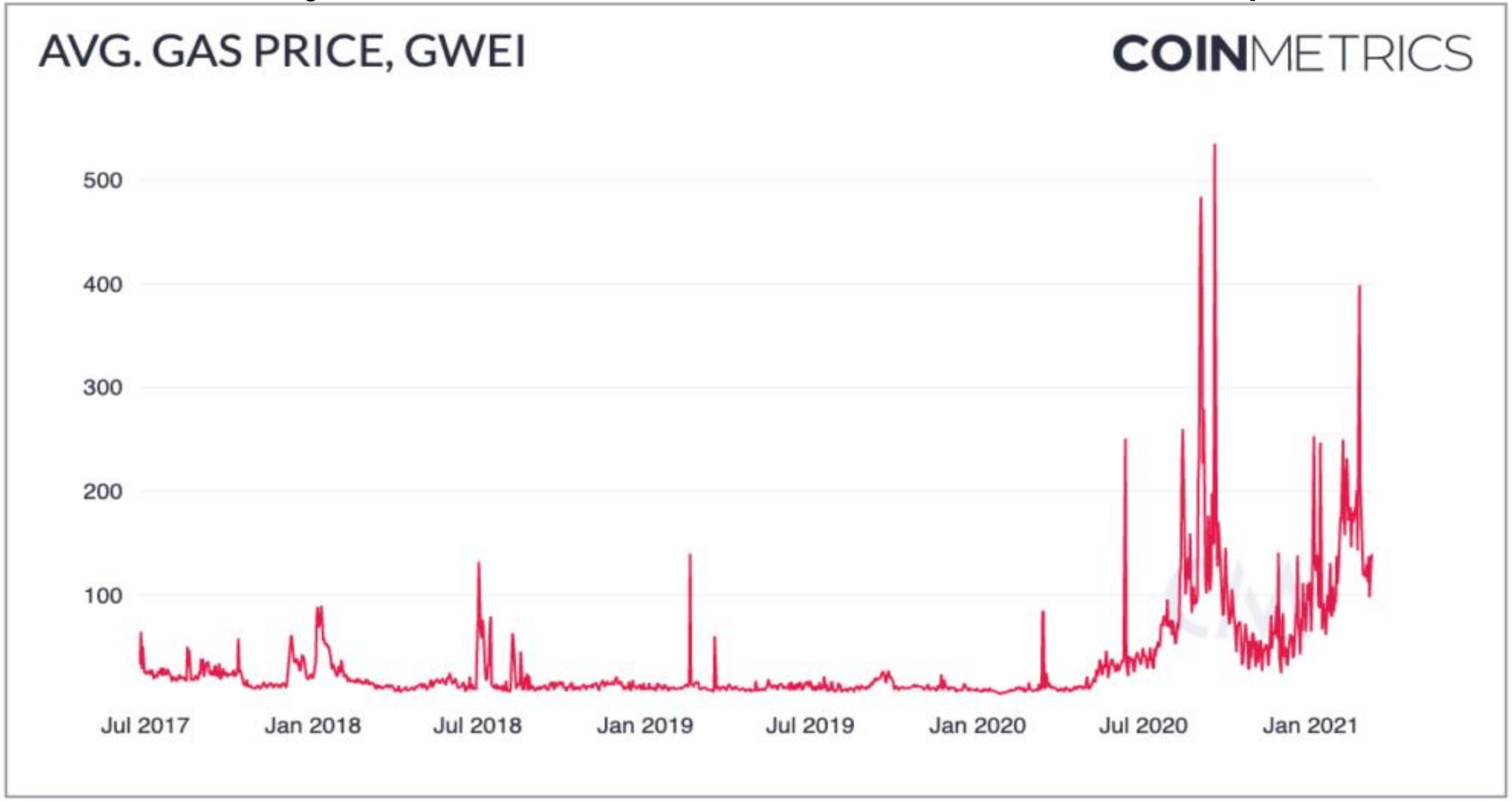
First-price auctions – what's wrong?

- Lack of Incentive compatibility
  - Not clear how much to bid
  - Overpayment of fees – user ends up thinking of what other people might bid
  - Need to estimate prices – complex and still results in overpayment



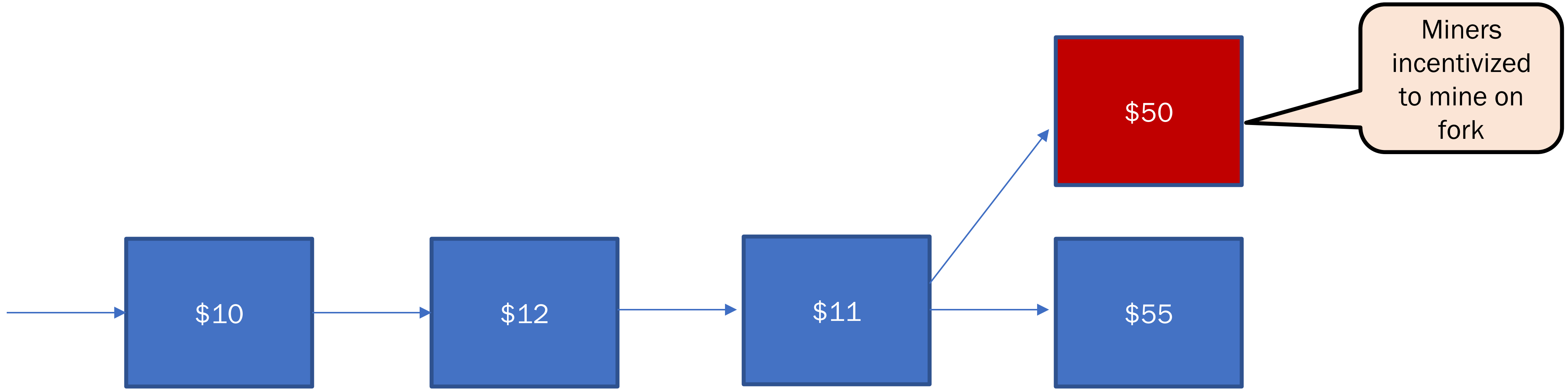
First-price auctions – what's wrong?

- Bidding up of prices – does not match actual txn cost
  - Txn costs skyrocket to unreasonable amounts in practice



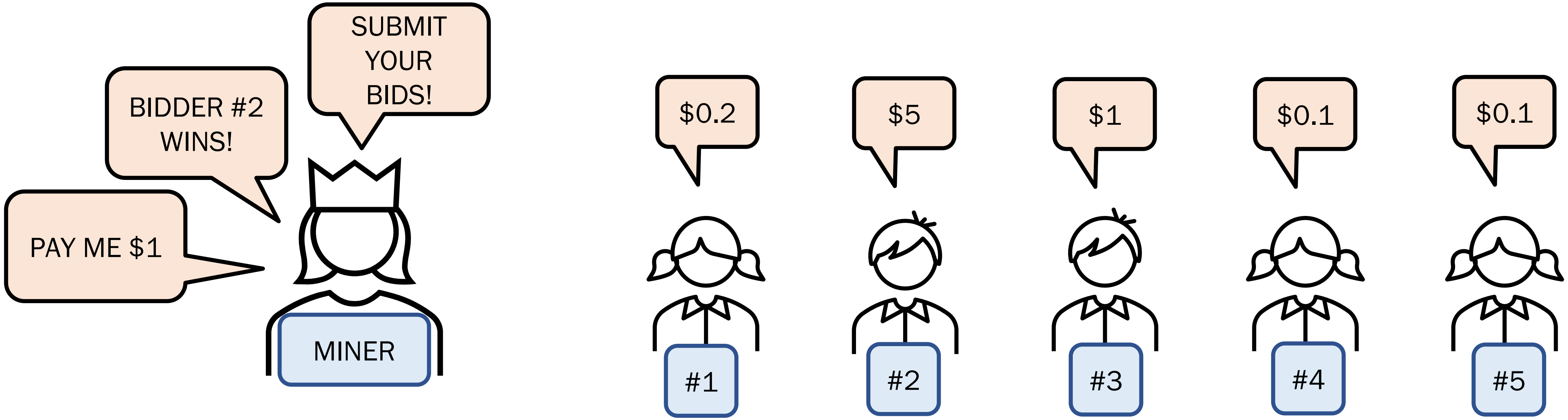
First-price auctions – what's wrong?

- Blockchain instability
  - Even after block has been mined, other miners attempt to undercut
  - Dominant mining strategy : deviate from protocol
  - This makes a "51% attack" achievable at lesser hash power



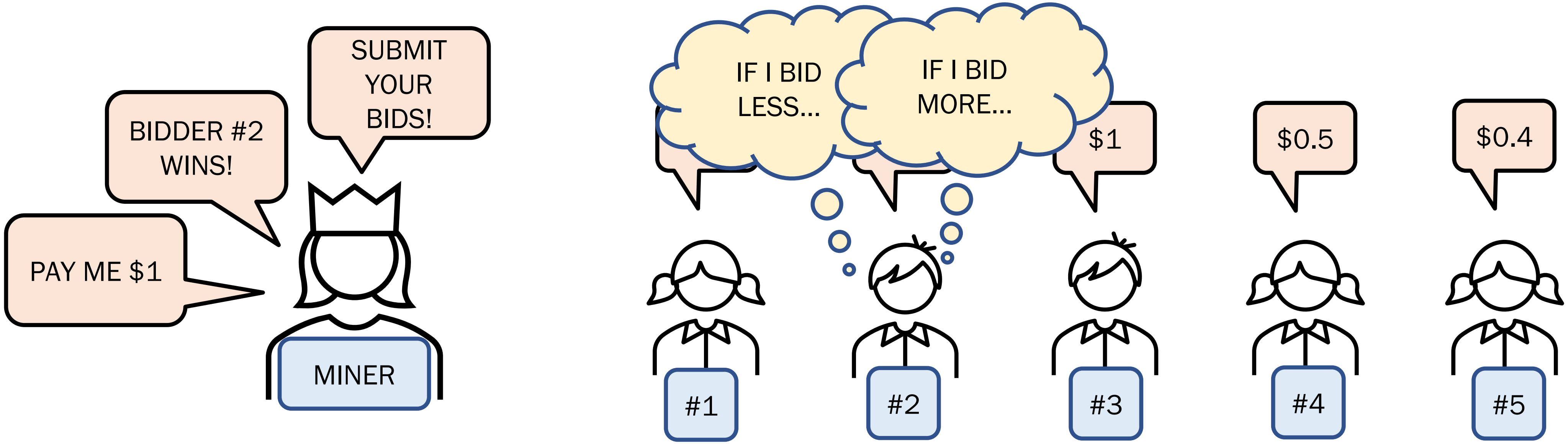
Design 2 : a second-price auction

1. Every txn submits a bid
2. Miner includes the N highest bids
3. Everyone pay fees equal to N+1-highest bid
4. All of the money goes to the miner



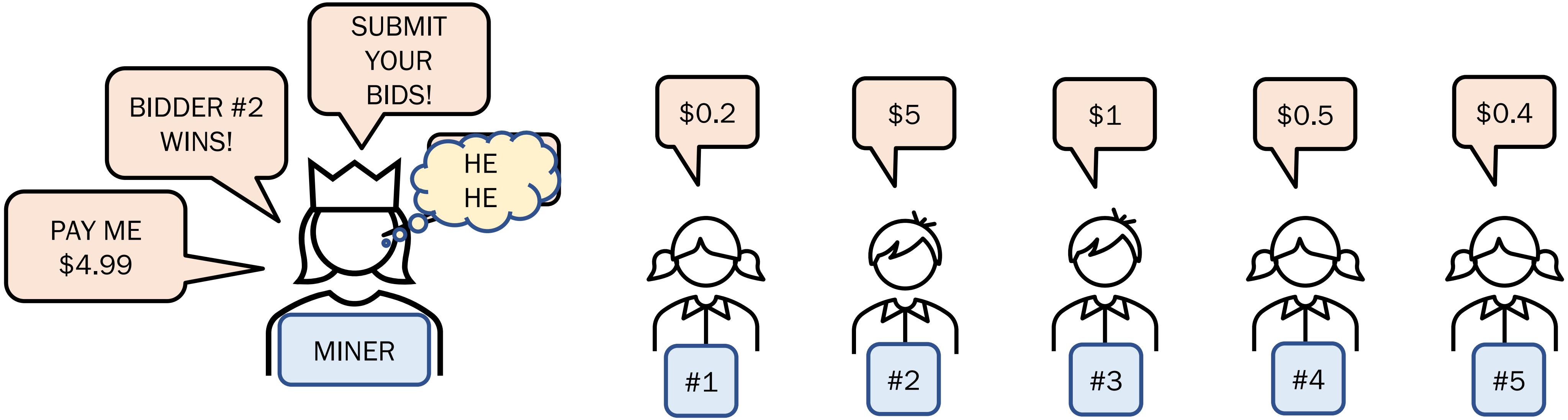
Second-price auction – what's right?

- Incentive compatibility
  - Bidding is easy now
  - Simply bid how much value you have in mind for the txn



Second-price auction – what's wrong?

- Miner can increase fee charged - by inserting txns
  - Miner can insert their own txns
  - Miner bids just below Nth bid OR
  - Miner introduces N/2 txns below N/2th bid …



Second-price auction – what's wrong?

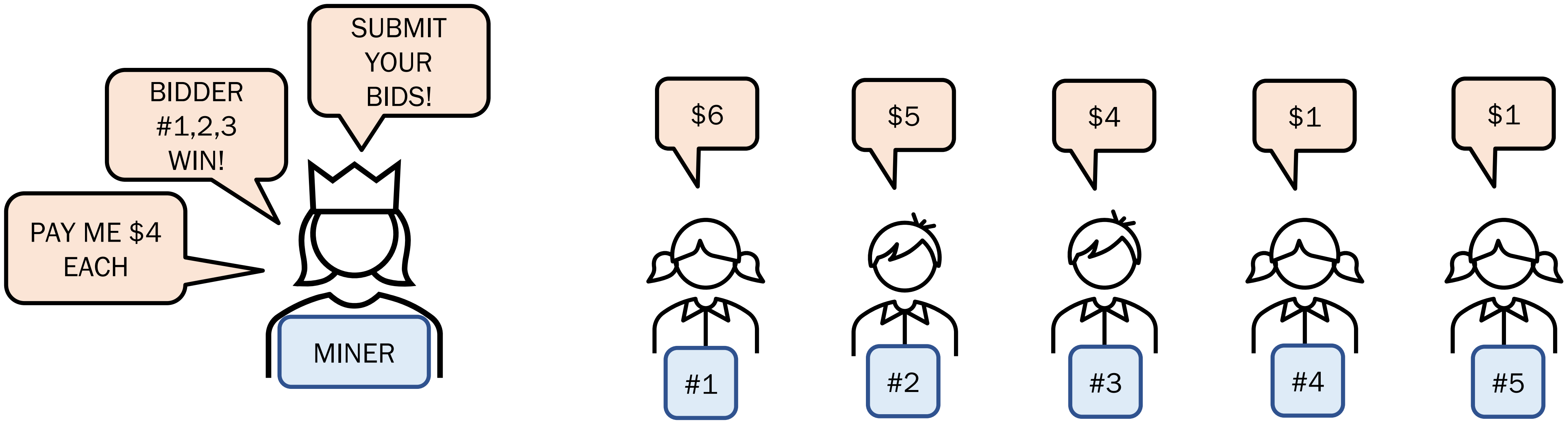
- Miner played by the rules – and earned extra money
  - Such behavior makes user experience worse and markets inefficient
  - Profit obtained is our first encounter with "MEV"
  - MEV – Miner Extractable Value

Design 3 : Monopolistic Auction

1. Every txn submits a bid
2. Calculate N* = argmax (Nth highest bid) x N
3. Miner includes the top N* highest bids
4. Everyone pays fees equal to N* th-highest bid
5. All of the money goes to the miner

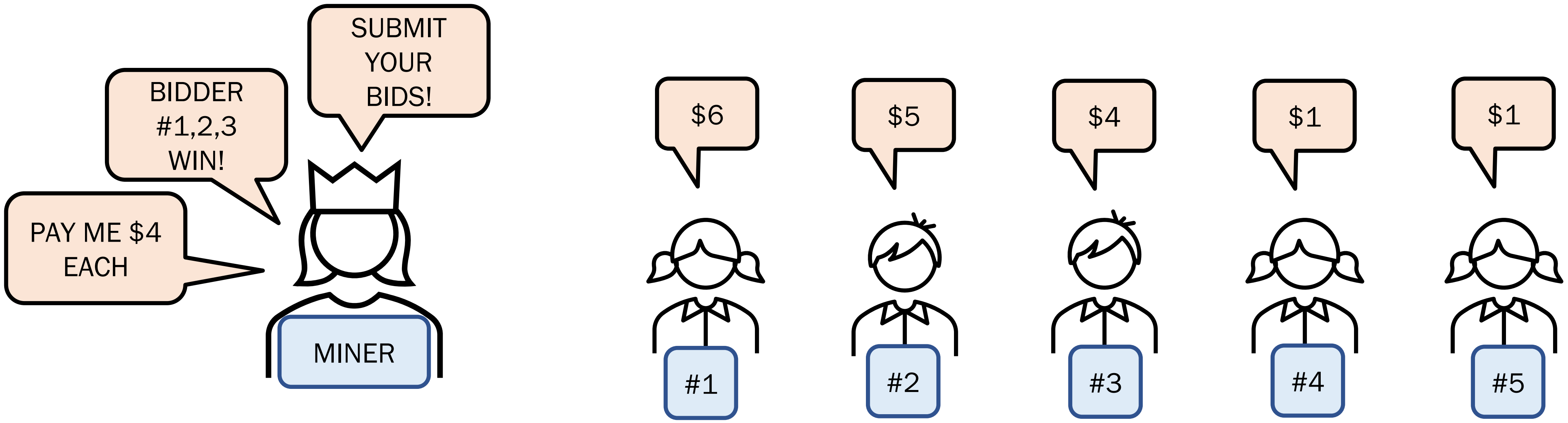
Monopolistic Auction – what's right?

- Incentive compatibility remains
  - Bidding : no incentive to bid higher or lower
  - Caveat : N^th person incentivized to bid slightly lower, but not a big difference irl



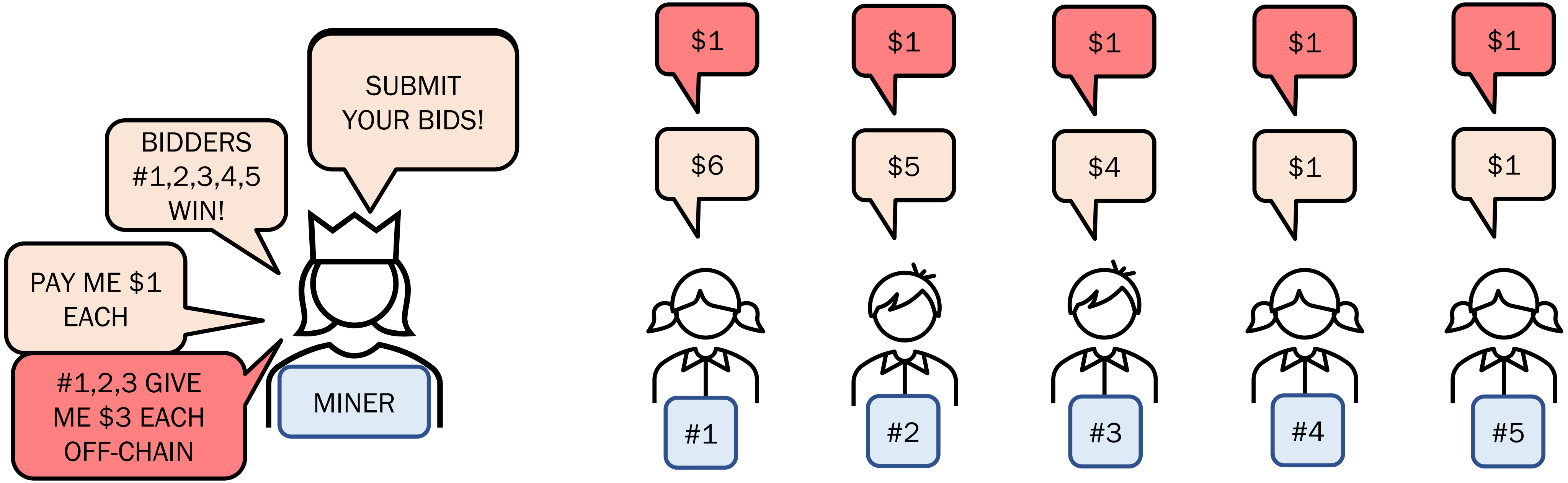
Monopolistic Auction – what's right?

- Miner conducts auction honestly
  - Miner gains no profit by inserting its own transactions
  - Proof is non-trivial, but intuition is – if miner tries to drive up price, then N^ decreases -> revenue collected stays the same



Monopolistic Auction – what's wrong?

- Off-chain collusion
  - Miner gains profit by eliciting bids off-chain first
  - Miner can make an offer that is beneficial to everyone!
  - How? Miner gets $14 instead of $12

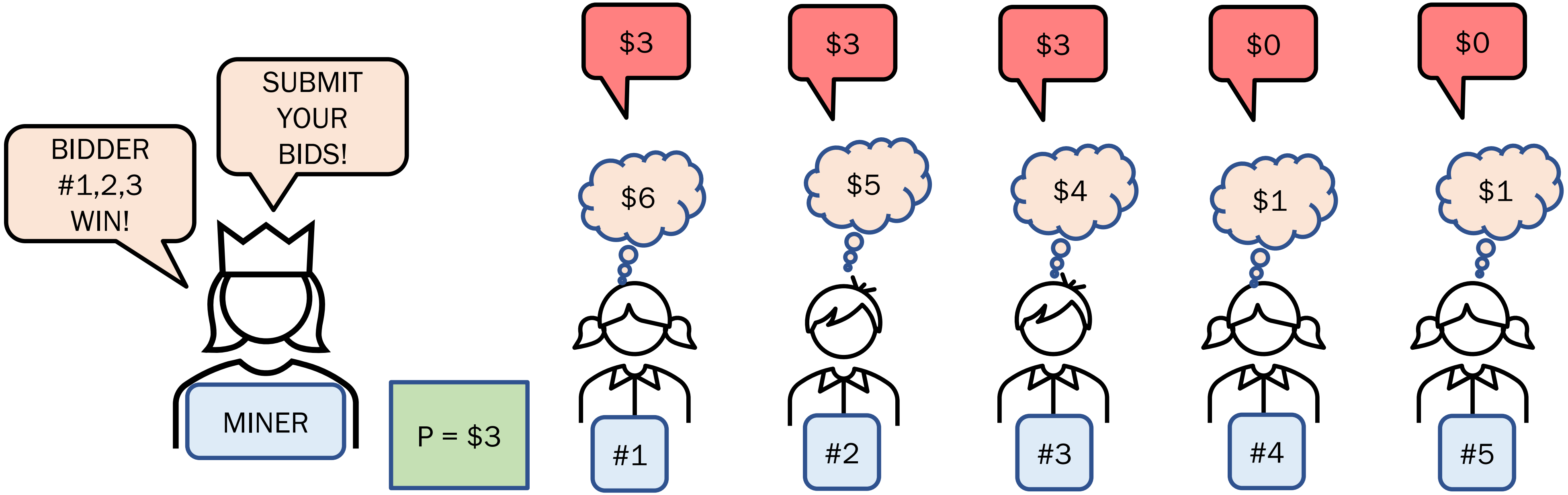


Design 4 : EIP 1559

1. Price p fixed by the protocol “base fee” + can include an optional tip
2. Miner picks at most N txns
3. Amount p from each user is **burnt**
4. All tips collected + a fixed block reward go to miner

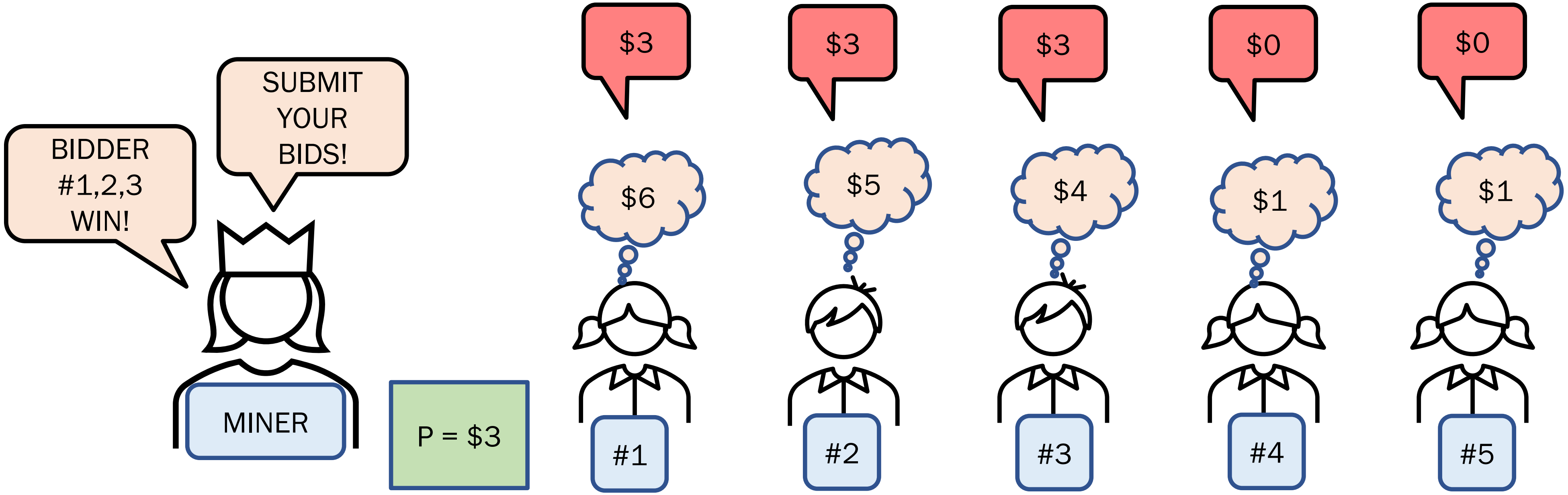
EIP-1559 – what's right?

- Incentive compatibility remains
  - If p is chosen so that < N users have utility > p, then it is incentive compatible
  - Bid only if utility > p, tip = 0



EIP-1559 – what's right?

- Miners are no longer incentivized to cheat or collude
  - If p is chosen so that < N users have utility > p, then everyone charged a fixed price and miner gets a fixed block reward
  - Colluding off-chain would not be beneficial for users



EIP-1559 – what’s wrong?

- What happens when the value of p is off? i.e. > N users have utility > p
  - Get back the first price auction
  - This is still resistant to miner cheating or collusion
  - But no longer incentive compatible

- Protocol needs to choose base fee p carefully

EIP-1559 – updating base fee

1. Define two constants :
   1. *maximum block size* = N
   2. *target block size* = N/2
2. Update p according to previous block size
   1. Increase p when previous block size > target
   2. Decrease p when previous block size < target
3. Uses the following rule to do that :

$$p_{new} = p_{old}\left(1 + \frac{1}{8}\frac{B_{prev} - B_{target}}{B_{target}}\right)$$

EIP-1559 – updating base fee

- Current protocol update rule still a bit ad-hoc
- Open questions :
  - Best way to update the base fee p?
  - Better mechanisms to collect fees?

EIP-1559 – what's wrong?

- Usually the tip revenue is minor – but all the rest of the revenue gets burnt
- Miner not incentivized to process transactions
- Miner goes off-chain to "extort" – announces that will only include txn if paid some amount xyz

- Turns out – no auction exists that is incentive compatible, credible, collusion-proof, and extortion-proof
- Open problem – how do you effectively trade-off between these four properties?

EIP-1559 – what's wrong?

- EVM consumes distinct kinds of resources : Compute, Storage, Memory
- All resources are quantified in the same terms : gas fees
- Problem?

- Suppose block has many txns that are CPU-intensive
- Competition for CPU drives up gas fees
- Drives up costs of doing other non-CPU txns as well – txns consuming normal amounts of bandwidth or memory or storage

Multi-dimensional pricing

- Keep updating a *vector* of prices **p**
- Number of dimensions = Number of distinct resources
- Use vector form of update equation :

$$p_{new} = p_{old}\left(1 + \frac{1}{8}\frac{B_{prev} - B_{target}}{B_{target}}\right)$$

- Here B_prev and B_target are also fixed vectors set by protocol and evolved slowly
- Turns out, the update equation above is same as doing gradient descent to maximize blockchain user welfare – is essentially optimal way to set prices

Summary

- Smart contract introduction
- EVM + Solidity + Gas
- Pricing of transactions on the EVM

Next Lecture

Meet our first element - Exchanges