Ethereum and Smart Contracts

Thierry Sans

The Bitcoin Inspiration

What if you could program money and decentralized logic into a blockchain?

Ethereum in a Nutshell

- Uses Elliptic Curve Public Keys (secp256k1) and ECDSA signature algorithm
- Consensus :
  (before) Proof of Work
  (since 2022) Proof of Stake
- Block time : ~12 seconds
- `ETH` are created through staking rewards and transaction fees
- Account-based blockchain **+ programmable smart contracts**

➡ Not just a cryptocurrency
it's a decentralized computing platform to automate trustless transactions
a.k.a decentralized applications (dApp)

Ethereum Accounts & Transactions

Different types of Ethereum **accounts** (all associated with an address):

- Externally Owned Accounts (EOAs)
- Contract Accounts
- Account abstractions (newest - will be covered in another lecture)

Different types of Ethereum **transactions**

- transfer ETH from EOA accounts to Ethereum addresses
- deploy smart contracts
- call methods of a deployed smarts

Smart Contracts

What is a smart contract?
A computer program (EVM bytecode) deployed on the blokchain that defines 1) a set of state variables and 2) methods to read/write these state variables

Can a smart contract hold ETH?
Yes, a smart contract has an address and can hold ETH but there is no private key associated with that address

How to write a smart contract?
Either write an EVM bytecode program directly
or use a high-level language (e.g. *Solidity*) that compiles programs into EVM bytecode

How to deploy a smart contract?
By sending a transaction that will write the EVM bytecode on the Ethereum blockchain

Can you change the code of a smart contract once deployed?
(short answer) no, the code is immutable
However, the contract state can change (by modifying contract state variables) when smart contract methods are called

How to call a method of a deployed smart contract?
Either directly using EOA account (sending a transaction) or from another contract

EVM code

What can the code do?

- Perform Arithmetic, Logical Operations, Bit Operations plus conditionals and loops (Turing complete)
- Store data (through contract state variables)
- Read transaction and block data
- Transfer ETH (held by the contract) to other another address
- Receive ETH (and execute some logic when funds are received)
- Call methods from other deployed contracts
- Emits events (logs that will be written on the blockchain)
- Self-destruct

Execution and Gas Fee

Who executes smart-contracts?
The Ethereum nodes that process transactions

When is the smart contract executed?

- When the transaction is received (unconfirmed mempool), the code is executed (by the node) but the contract's state is not modified (dry-run)
- When the transaction is confirmed (into a block), the code is executed (by the node chosen to confirmed the next block) and the contract's state is modified (i.e written to the blockchain)

➡ Deterministic execution: given the sequences of transaction and the blockchain state, the outcome can be determined

If the code has loops, how do we ensure that the execution will terminate?
In a nutshell, calling a smart contract method costs money (a.k.a gas). Whoever calls a smart contract method must pay some fee that will reward the node (selected to confirm the next block) for executing the smart contract

What happen when a method call fails or does not terminate because it runs out of gas?
The transaction is confirmed as a failed transaction. The contract state is not updated (full reverse) but the gas fee is not returned to the caller but kept by the node.

Gas Fee Calculation

$$\text{Total Fee} = (\text{Base Fee} + \text{Priority Fee}) \times \text{Gas Used}$$

- **Base Fee:** set by the protocol, dynamically adjusted based on network congestion
- **Priority Fee:** the tip paid to miners/validators as an incentive to prioritize the transaction
- **Gas Used:** the amount of gas consumed by the smart contract execution
  Each operation (storage, computation, external calls) consumes gas
  Examples :
    - Writing a new storage variable: 20,000 gas
    - Modifying an existing storage variable: 5,000 gas
    - Simple arithmetic operation: ~3 gas
    - Sending ETH: ~21,000 gas

➡ If the caller supplies more gas than actually needed, the excess of gas is refunded once the transaction processed

In summary

In summary, what data is written onto the Ethereum blockchain?

Transactions, smart contract code, smart contract state variables and events

Why are smart-contracts useful?

Automates agreements without intermediaries by enabling trustless transactions

Examples of dApps

- Payment Automation
- Tokens (Fungible) including Stablecoins, NFTs (Non-Fungible) and RWAs (Real-World Assets)
- Funds and Assets Management
- Decentralized Exchanges and Decentralized Marketplaces
- Lending and Borrowing Platforms
- Insurance and Derivatives
- Governance (DAO - Decentralized Autonomous Organization)
- Supply Chain Management

Benefits and Risks of Smart Contracts

Benefits:

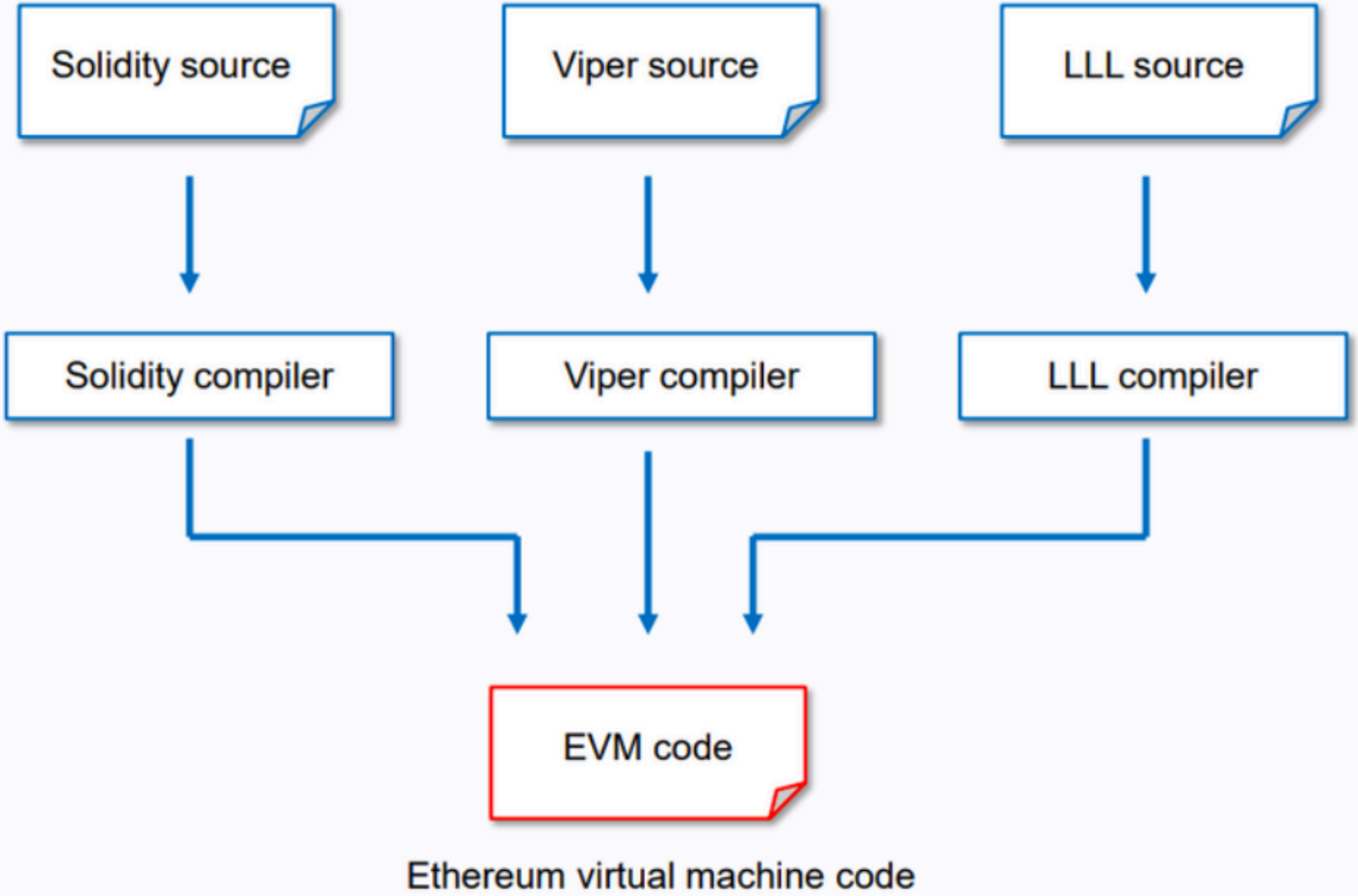
- Trustlessness
- Automation
- Transparency

Risks:

- Security vulnerabilities
- Immutable bugs
- Gas cost considerations

Solidity

EVM code generation



Ethereum virtual machine code

Introduction to Solidity

High-level, contract-oriented language for Ethereum

Evolved alongside Ethereum to meet dApp needs (through EIP)

Similar in syntax to JavaScript/C++

Solidity Language Constructs

Data types

uint, address, bool, string, byte, enum, struct, array, mapping, ...

Structure

state variables, functions, events, modifiers

Code organization

contracts, inheritance, libraries, interfaces

Development Tools for Solidity

Remix IDE for quick prototyping

Frameworks such as Truffle, **Hardhat**, Foundry for development and testing