GDG SCHOOL

Code The Chain: From Blockchain Fundamentals to Smart Contract Deployment

Presented By: **Lyes Boudjabout**

November 28–29, 2025

GDG Algiers

B.P. 68M, 16270 Oued Smar, Algiers, Algeria.

Agenda

1. Blockchain Fundamentals
2. Types of Blockchains
3. Blockchain Node Types
4. Ethereum Accounts & State
5. Ethereum Transactions
6. Block Structure
7. Consensus Mechanisms
8. Digital Wallets
9. Intro to Smart Contracts
10. EVM Architecture
11. Solidity Basics
12. Write Your First Smart Contract
13. Compilation, ABI, and Bytecode
14. Deploy a Smart Contract
15. Random Numbers
16. Smart Contract Security

Who Am I ?

Lyes Boudjabout

- Smart Contract Security Auditor
- Dev Projects Manager & IT Co-Manager @GDG Algiers
- 3rd Year Student @ESI Algiers

Introduction

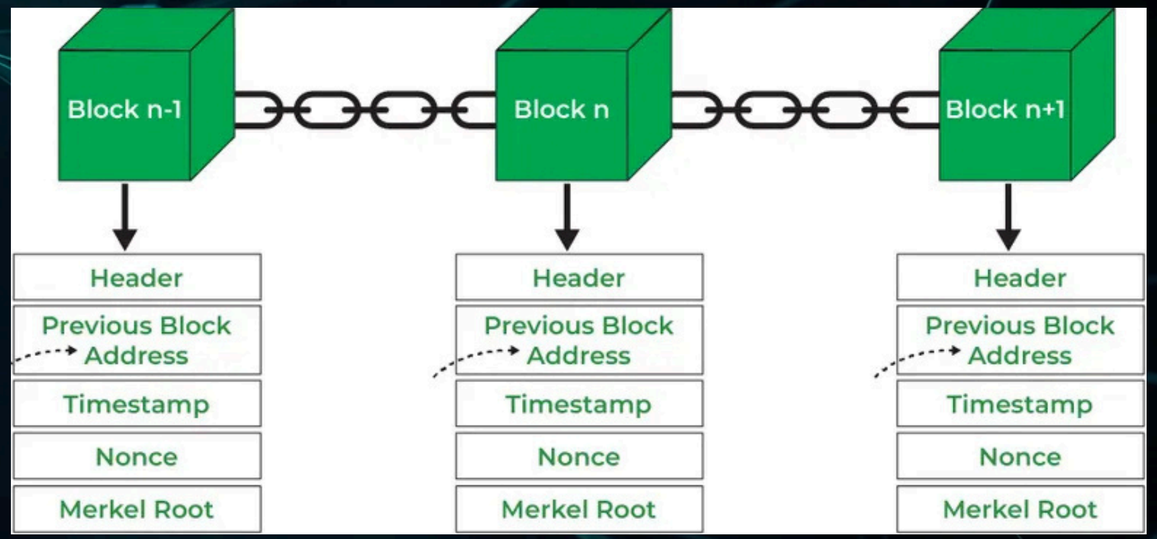
Blockchain isn't Bitcoin !!

- Blockchain is much **more than a cryptocurrency**—Bitcoin is just one example.
- It's a **decentralized**, **secure**, and **transparent** way to **record information**.
- Different blockchains exist with unique capabilities—**Ethereum**, **Solana**, and **more**.
- In this workshop, you'll clearly **see why blockchain is a foundational technology**, **not just Bitcoin**.
- In this workshop, we'll be working on the **Ethereum blockchain**. The concepts we learn are applicable to any other **EVM-compatible chain**.

Blockchain Fundamentals

As a **Data Structure:**

A blockchain is a **decentralized**, **distributed ledger** that records **immutable**, **chronologically ordered**, **cryptographically secured** transactions across a **peer-to-peer network**, ensuring **trust minimization**, **transparency**, and **tamper resistance** without relying on a **central authority**.



Blockchain Fundamentals

As an **Ecosystem:**

A blockchain ecosystem is the interconnected network of **participants**, **protocols**, **tools**, and **applications** that collectively enable the blockchain to function. It typically includes:

- Users & wallets
- Infrastructure & protocols
- Governance mechanisms
- Nodes & validators
- Developers & smart contracts
- Tokens & assets

Types of Blockchains

A blockchain type describes who can access and control the network. It can be public (anyone can join), private (restricted access), or consortium (shared control among organizations), reflecting its decentralization and governance.

Public (Permissionless)

Demonstrates full decentralization and trustless operation.

Private (Permissioned)

Shows controlled, enterprise-grade blockchain applications

Consortium (Hybrid)

Illustrates collaboration between multiple organizations with partial decentralization.

Blockchain Node Types

Ethereum nodes maintain the network and serve different needs. **Full nodes** store and validate the entire blockchain. **Light nodes** store only headers and rely on full nodes for details, making them fast and lightweight. **Archive nodes** keep all historical states, useful for analytics and auditing. These types balance **security, speed, and accessibility.**

Full Node

Light Node

Archival Node

Ethereum Accounts & State

Purpose

- Track all accounts and balances on Ethereum
- Enable deterministic execution of transactions and contracts
- Maintain a single source of truth for the network

Account Types

- **Externally Owned Accounts (EOAs):** user-controlled, nonce, balance
- **Contract Accounts:** program-controlled, storage, bytecode

Ethereum Accounts & State

Ethereum State Components

- **Nonce:** prevents replay attacks
- **Balance:** ETH held by account
- **Storage:** persistent key-value storage for contracts
- **Code:** compiled smart contract (for contract accounts)
- **Demo:** Blockchain live

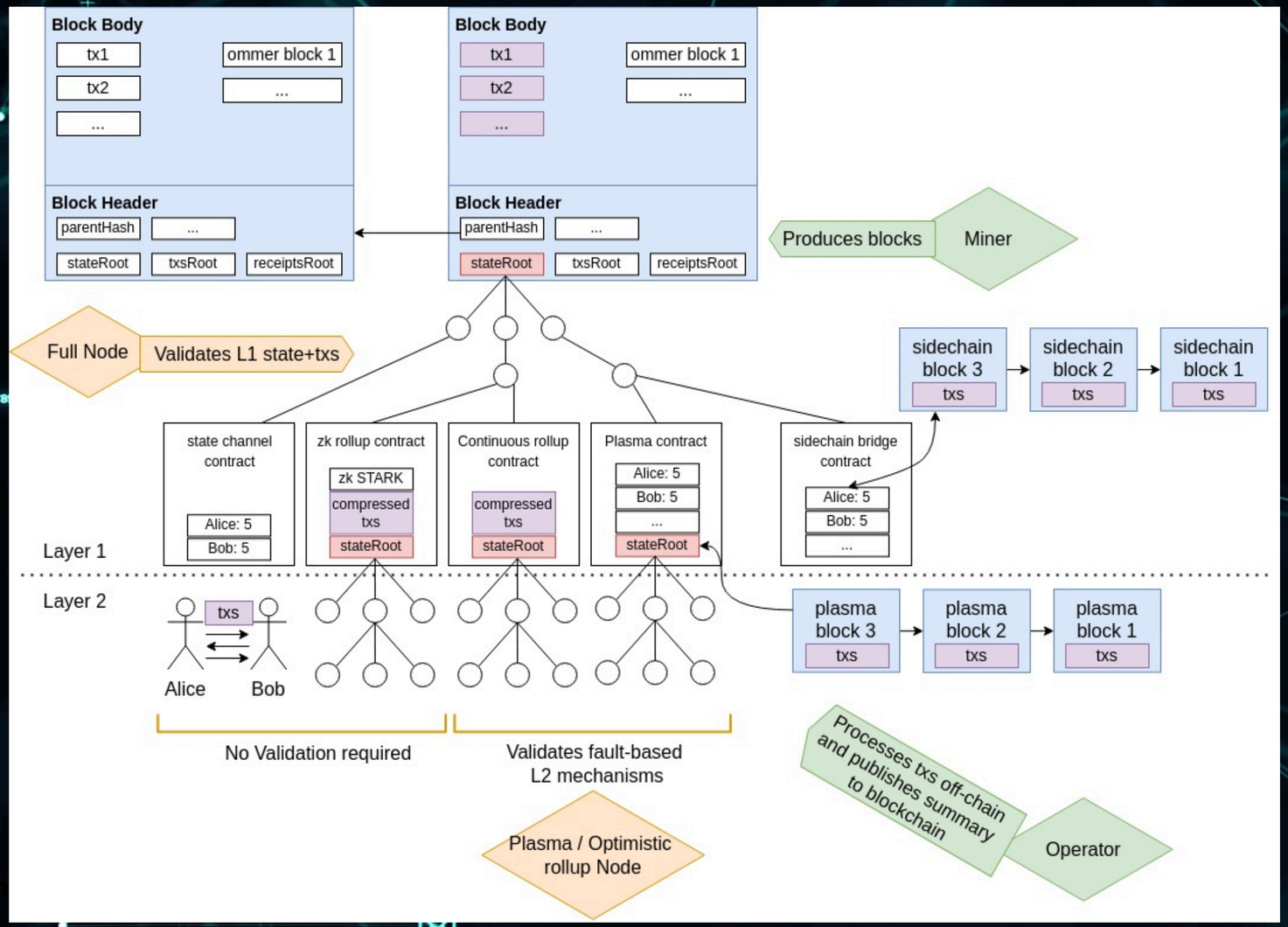
Ethereum Transactions

Demo: Etherscan

Transaction Structure:

- **Sender (From):** Account initiating the transaction
- **Receiver (To):** Account or smart contract receiving value or instructions
- **Value:** Amount of Ether being transferred
- **Gas Limit:** Maximum computational effort allowed for execution
- **Gas Price / Fee:** Payment for transaction processing
- **Data (Optional):** Encoded instructions for smart contract interaction

Block Structure



Minimized Block Structure

Demo: Blockchain Live

- **Index:** Marks the block's position in the chain.
- **Timestamp:** Records when the block was created.
- **Data:** Holds the transactions or information stored in the block.
- **Previous Hash:** Links this block to the one before it to maintain the chain.
- **Hash:** A unique fingerprint of the block, ensuring integrity.
- **Nonce:** A number miners change to satisfy the mining difficulty.

Consensus Mechanisms

- Ensures all distributed nodes maintain a unified, trusted ledger.
- Preserves data integrity and prevents double spending.
- Eliminates reliance on a central authority.
- Enables trustless coordination among participants.
- Protects the network from malicious behavior.
- Keeps all nodes aligned on a single source of truth.

Proof of Work (PoW Mechanism)

Purpose: Enables decentralized agreement by requiring participants to solve computational puzzles.

How It Works:

- Miners compete to find a valid hash.
- First miner to solve the puzzle proposes the next block.
- Network verifies the solution before accepting it.

Key Characteristics:

- **Security:** anchored in high computational cost.
- **Energy-intensive:** requires significant hardware and electricity.
- **Slower finality:** block confirmations reduce certainty.

Demonstration: Blockchain Live

Proof of Stake (PoS Mechanism)

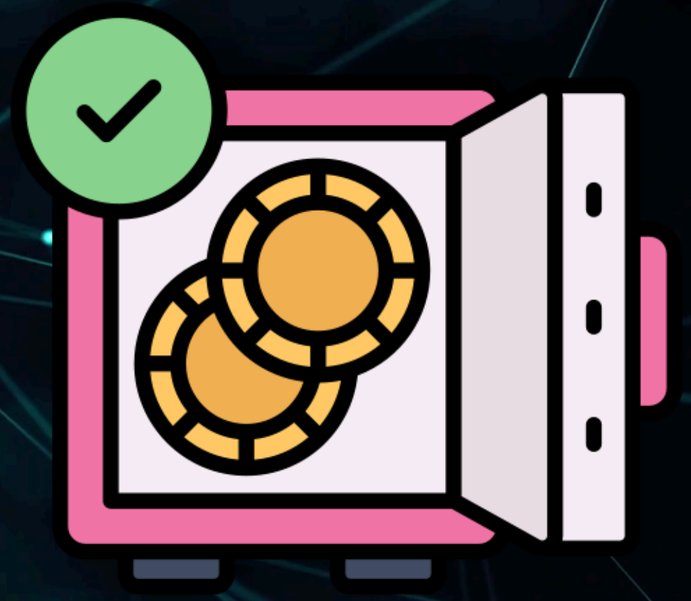
Purpose: Achieves consensus by selecting validators based on the amount of cryptocurrency they lock (stake).

How It Works:

- Validators stake coins to participate.
- Protocol selects validators to propose and attest to blocks.
- Misbehavior results in penalties or slashing of stake.

Key Characteristics:

- **Energy-efficient:** no heavy computation required.
- **Scalable:** faster block times and confirmation.
- **Economic security:** stake at risk discourages attacks.



Other Consensus Mechanisms

- Delegated Proof of Stake (DPoS)
- Proof of Authority (PoA)
- Practical Byzantine Fault Tolerance (PBFT)
- Proof of History (PoH)
- Proof of Capacity / Proof of Space (PoC/PoS)
- Proof of Elapsed Time (PoET)
- Hybrid Consensus

Digital Wallets

Core Components:

- **Private Key:** Cryptographic credential that authorizes transactions.
- **Public Key:** Derived from the private key; used to generate the wallet address.
- **Wallet Address:** Public identifier used to receive assets and interact with contracts.
- **Seed Phrase (Recovery Phrase):** A human-readable representation of the private key, used for full wallet backup and restoration.
- **Primary Functions:**

Sign Transactions | Asset Management | Contract Interaction | Identity & Access Control

How Wallets Connect to the Blockchain

- Wallets **do not run nodes**; instead, they connect through **RPC endpoints** provided by full nodes or node service providers (e.g., Infura, Alchemy, self-hosted nodes).
- The wallet signs the transaction **locally**, then submits it to the **connected node**, which:
  a. Validates the transaction
  b. Propagates it across the P2P network
  c. Includes it in the mempool for block consideration
- Wallets subsequently **query the blockchain state** via the node to update balances and contract data.

Practical Activity: Install MetaMask Extension

Introduction To Smart Contracts

- In 2009, **Bitcoin** proved we don't need **banks** to move money — it gave us **digital currency** that works anywhere, anytime.
- But then came a bigger question: what if we could **program money** itself? What if we could build **applications** — **marketplaces**, **organizations**, even **games** — that run **automatically**, **without servers** or **middlemen** ?
- That's exactly what **Ethereum** introduced: not just digital money, but a **world computer** where anyone can deploy **unstoppable code**.
- That code is called a **Smart Contract**.

What is a Smart Contract ?

A **smart contract** is a **self-executing program** stored on a blockchain that automatically enforces rules and agreements when predefined conditions are met.

Key Points:

- Runs **autonomously**, without intermediaries or servers.
- Ensures **trustless execution**, meaning participants don't need to trust each other.
- Can manage **digital assets**, **execute transactions**, and **interact** with **other contracts**.
- Immutable once deployed: the code **cannot be altered**, ensuring predictable behavior.

How Smart Contracts Work ?

1. Triggered by Transactions: A smart contract only executes when someone calls it or sends Ether.

2. Executes Code on the EVM: The Ethereum Virtual Machine runs the contract's logic deterministically on all nodes.

3. Applies Logic & Conditions: Contract checks predefined rules, updates internal data, or interacts with other contracts.

4. Emits Events & Updates State:

- Changes are recorded in the blockchain state.
- Events notify external apps or wallets about outcomes.

5. Outcome is Transparent and Immutable: Execution is automatic, verifiable, and the same for everyone.

Ethereum Virtual Machine (EVM) Architecture

The EVM is a **safe environment** where smart contracts run. It makes sure all nodes in the network **get the same result** and that contracts **cannot break the network**.

Stack	→ Temporary Desk
Memory	→ Work Table
Storage	→ Record Vault
Opcodes	→ Instruction Manual
Gas	→ Energy Fuel

Solidity Programming Language

- **What it is →** The primary programming language for writing smart contracts on Ethereum.
- **Purpose →** Defines rules & logic that run on the Ethereum Virtual Machine (EVM).
- **Features:**
  a. High-level, contract-oriented language.
  b. Syntax inspired by JavaScript, Python, and C++.
  c. Supports inheritance, libraries, and user-defined types.
- **Why it matters →** Enables developers to build decentralized apps (dApps) that are transparent, trustless, and unstoppable.

Writing Your First Contract

- **contract** defines a contract.
- **string public message** is a state variable.
- **HelloWorld** is the contract name.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HelloWorld {
    string public message = "Hello, Ethereum!";
}

```

Solidity Basics

- State variables
- Constructor
- Modifiers
- Events

- Functions
- Arrays & Structs
- Ether interactions
- Access control & require statements

Let's See These Concepts in a Contract Example

Solidity Data Types

Value Types

- **bool:** Boolean value, true or false
- **uint / uint8...uint256:** Unsigned integers (default 256 bits)
- **int / int8...int256:** Signed integers
- **address:** Ethereum account or contract
- **bytes1...bytes32:** Fixed-size byte arrays
- **enum:** Custom enumerated type

Reference Types

- **string:** UTF-8 encoded text
- **bytes:** Dynamic byte array
- **array:** Fixed-size or dynamic list: uint[] numbers; or uint[5] fixedNumbers;
- **struct:** Custom data structure
- **Mapping:** A key-value data structure; not iterable. Commonly used for storing balances, ownership, or other associations.

Solidity Functions

function getBalance(address account) public view returns (uint256 balance) {}

- **function getBalance:** Function name
- **(address account):** Parameter (input)
- **public: Visibility:** callable from outside the contract
- **view: Mutability:** function only reads blockchain state, does not modify it
- **returns (uint256 balance):** Return type and named return variable
- **{ ... }:** Function body: logic, returns value

Compilation, ABI, and Bytecode

ABI (Application Binary Interface)

- The interface that tells external apps how to interact with the contract.
- Defines functions, inputs, outputs, and events.
- Human-readable: JSON format that wallets and scripts use to call contract functions.
- Demo: VSCode

Bytecode

- The machine-level instructions executed by the EVM.
- Immutable once deployed on Ethereum.
- Not human-readable; only the EVM can understand it.
- Demo: Sepolia Etherscan

Practical Activity: Deploy a Smart Contract and interact with it

Deploying Guide

- Get some sepolia-eth from a faucet: Google Cloud Web3 Faucet
- Write a simple contract inside Remix IDE
- Go to Deploy & Run Transactions section
- Connect your MetaMask Wallet and hit the deploy button

Foundry Framework

- **What it is →** A blazing-fast, Rust-based toolkit for Ethereum smart contract development.
- **Purpose →** Simplifies writing, testing, and deploying Solidity contracts.
- **Why it matters →** Developer-friendly, highly performant, and widely adopted in professional auditing and DeFi projects.
- **Other Ethereum Dev Frameworks:**

1. **Hardhat →** JavaScript-based, plugin ecosystem, debugging & scripting.
2. **Truffle →** Early popular framework, now legacy in many projects.
3. **Brownie →** Python-based framework, integrates well with Vyper.
4. **Embark →** Full-stack dApp framework with deployment automation.

Foundry Framework

Core Components:

- **Forge →** Build, test, fuzz, deploy contracts.
- **Cast →** CLI for interacting with Ethereum (send tx, call functions).
- **Anvil →** Local Ethereum node for testing & simulation.
- **Chisel →** Interactive Solidity REPL for rapid prototyping and debugging.

Download Guide:

- Go to getfoundry.sh
- Run the first two commands
- Run forge -v to confirm installation is successful

Foundry Project Structure

- **src/:** Your Solidity contracts go here
- **lib/:** External dependencies (imported libraries)
- **out/:** Compiler output: ABI + Bytecode
- **script/:** Deployment or helper scripts (Solidity scripts)
- **test/:** Unit and integration tests for your contracts
- **foundry.toml:** Main configuration file (compiler version, optimizer, etc.)
- **remappings.txt:** Dependency remapping for imports

GDG SCHOOL

Practical Activity: Test & Deploy a Smart Contract Using Foundry Tools

How to generate a random number ?

JavaScript (Or any programming language)

```
//Random integer between 0 and N-1
const x = Math.floor(Math.random() * N);

```

Blockchain (Any language)

How to generate a random number ?

JavaScript (Or any programming language)

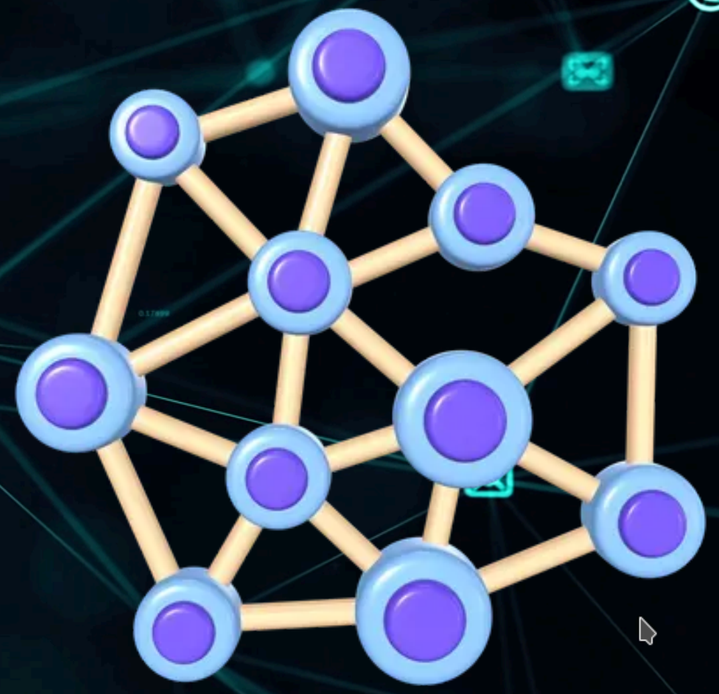
```
//Random integer between 0 and N-1
const x = Math.floor(Math.random() * N);

```

Blockchain (Any language)

Impossible

Why On-Chain Randomness Is Impossible

- **Blockchains Are Deterministic:** Every node must reach the same result for every transaction. Furthermore, determinism ensures consensus, but prevents true randomness.
- **On-Chain "Random" Sources Are Manipulable:** "Random" sources (block.timestamp, blockhash) are manipulable



Randomness Is An Oracle Problem

- Blockchains are **deterministic** and **isolated**.
- Smart contracts cannot access **external data** directly (prices, weather, identities, APIs).
- **Core risks:**

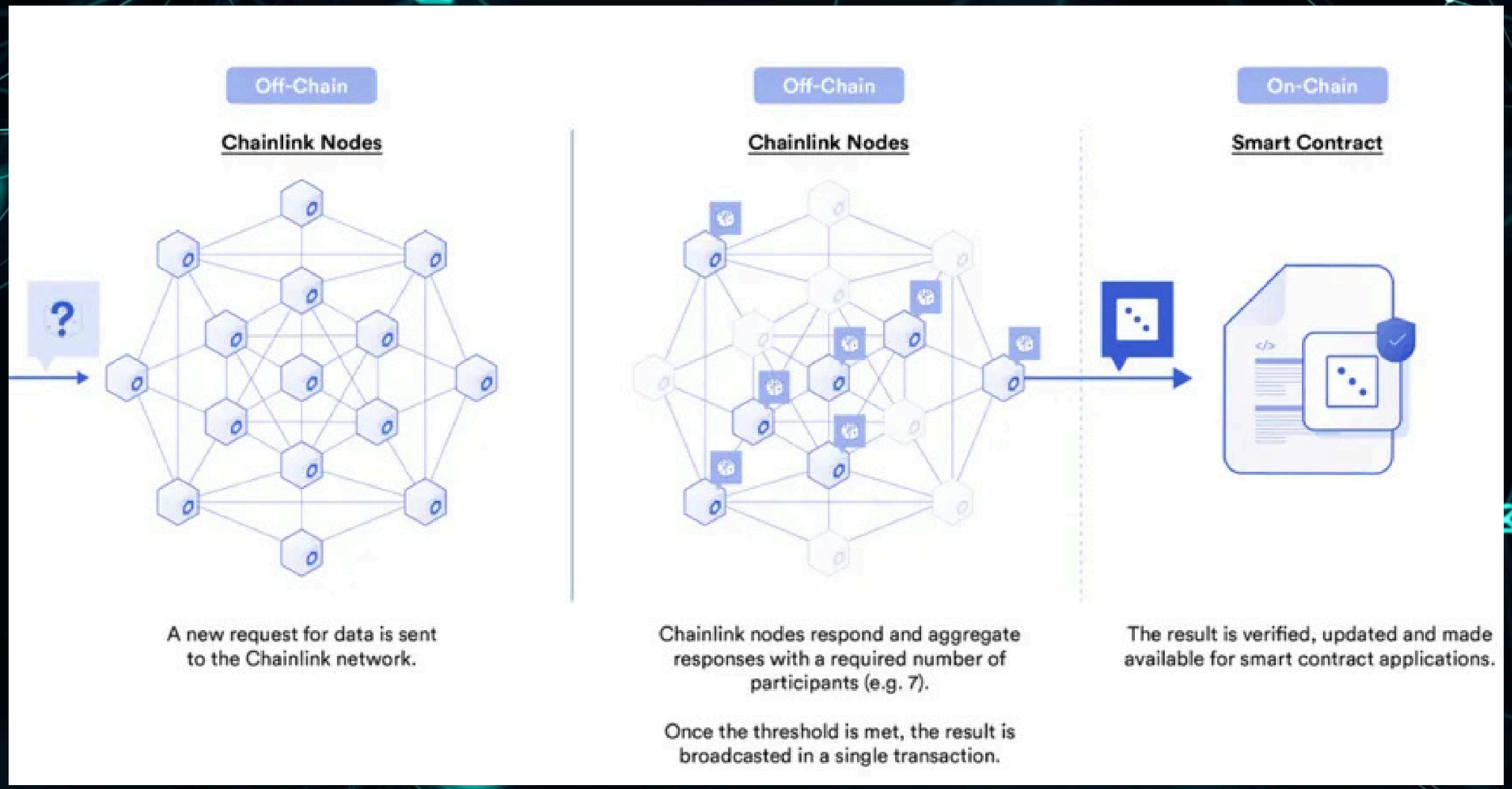
Single Point of Failure | **Data Manipulation** | **Consensus Violation**

- Smart contracts often depend on real-world information: Price feeds (ETH/USD), Sports scores / weather data, Identity verification, IoT sensors

Decentralized Oracles as a Solution

- An oracle is a service or system that provides external, real-world data to a blockchain in a way that smart contracts can use reliably.
- Multiple independent oracle nodes provide the same data.
- Aggregation ensures no single node can manipulate the outcome.
- Oracles can supply proofs (signatures, cryptographic commitments).
- Every blockchain node can verify the data before using it.

Random Number Generation



Why Smart Contract Security

- **Runs Automatically:** The contract executes on its own, so mistakes trigger themselves.
- **Cannot Be Changed:** Once deployed, the code is locked in and bugs stay forever.
- **Always Active:** The contract is live 24/7, constantly exposed to potential attacks.
- **Handles Real Money:** Any issue directly affects user funds or protocol assets.
- **Connected to Many Systems:** A weakness in one contract can impact others that depend on it.

What Is a Smart Contract Audit

Definition:

A **smart contract audit** is a **systematic review** and **assessment** of a smart contract's code to ensure it is **secure**, **functional**, and **free of vulnerabilities** before deployment.

Key Objectives:

- **Security Verification:** Identify vulnerabilities like reentrancy, overflow, and access control issues.
- **Functional Correctness:** Ensure the contract behaves exactly as intended.
- **Best Practices Compliance:** Check adherence to coding standards and design patterns.
- **Gas Efficiency & Optimization:** Highlight inefficient operations and costly patterns.
- **Dependency Assessment:** Review interactions with external contracts, protocols, and oracles.

Practical Activity: Find the vulnerability in the given contract & write the attack contract

Practical Activity:
Complete a CTF Challenge on Openzeppelin Ethernet

OpenZeppelin

Conclusion

- Understand blockchain fundamentals and Ethereum architecture.
- Learn Ethereum accounts, transactions, and EVM execution.
- Write, compile, and deploy Solidity smart contracts.
- Explore randomness and advanced contract features.
- Apply smart contract security principles to build safer applications.

Thank You !
Have a question ?

Github → https://github.com/Lyes-Boudjabout

Linkedin → https://linkedin.com/in/lyes-boudjabout

Email → nl_boudjabout@esi.dz