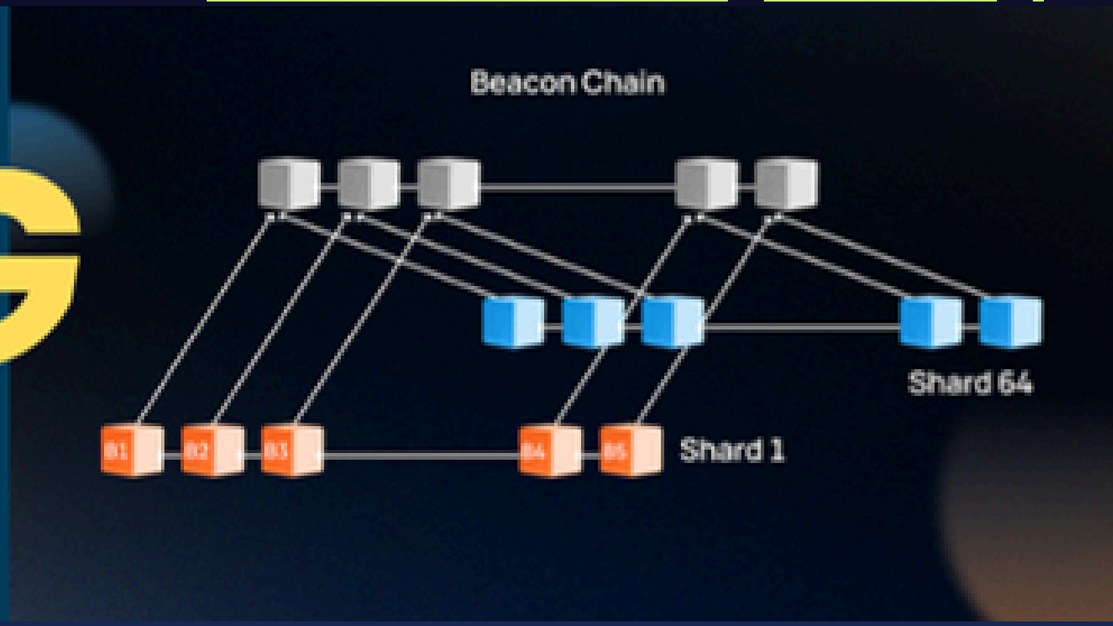
ETHEREUM 2.0 MASTERY PROGRAM

Instructor: Raja Rizwan Saleem

MODULE (6-10)

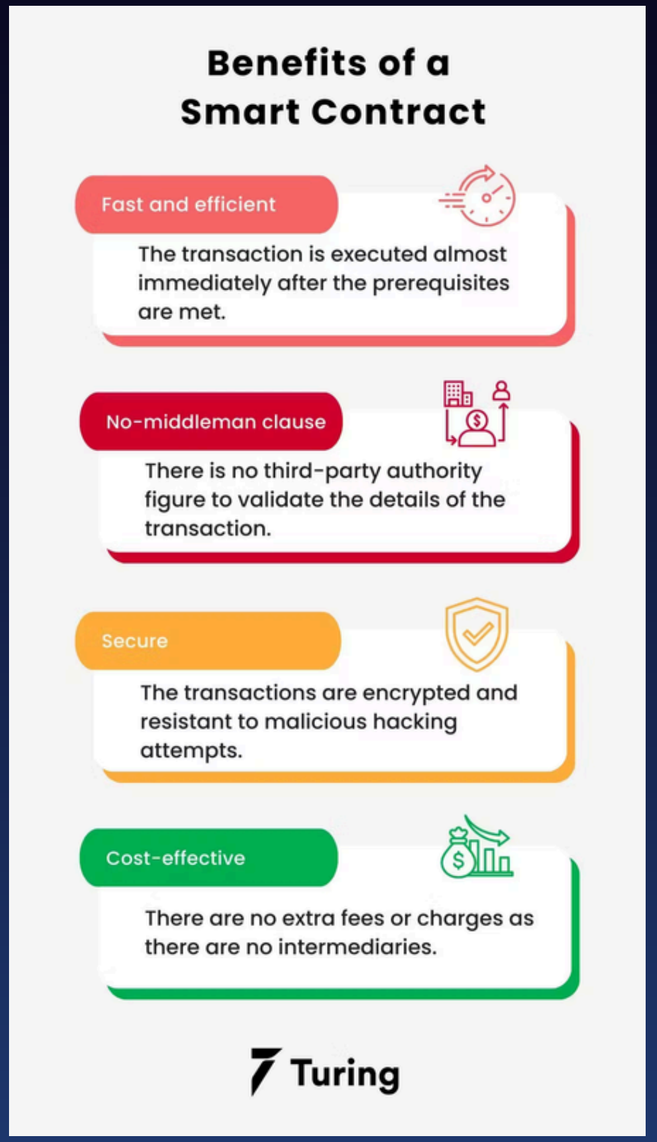
SHARDING
& ETHDO TOOLS



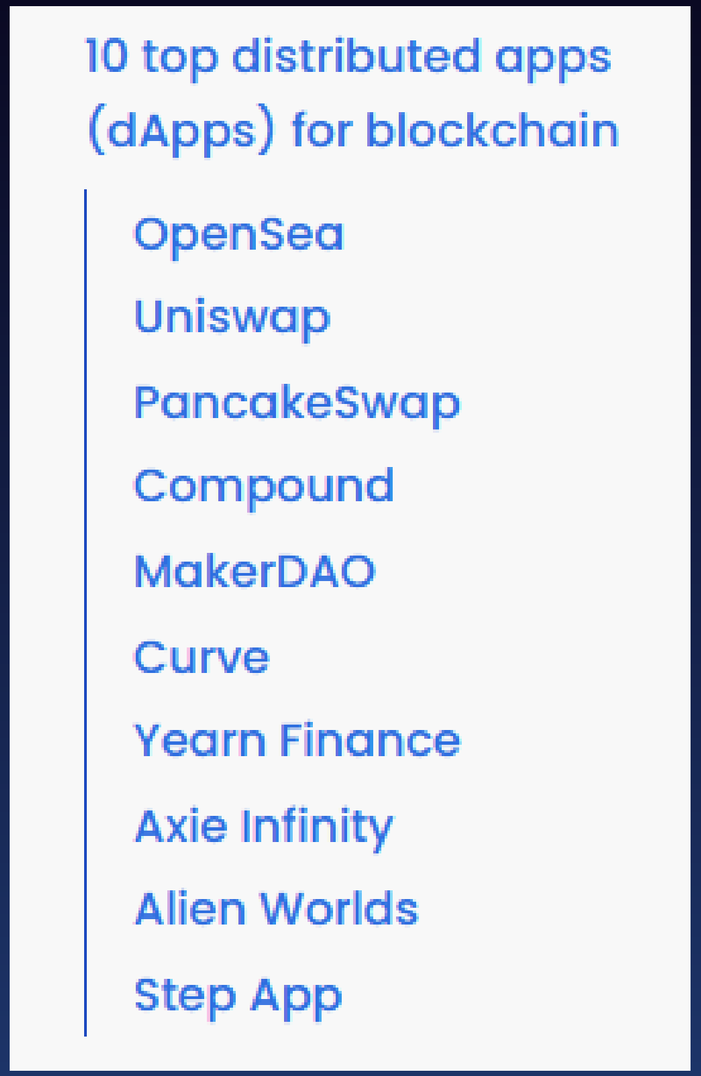
Raja Rizwan Saleem
Lead Blockchain Trainer

Last Session

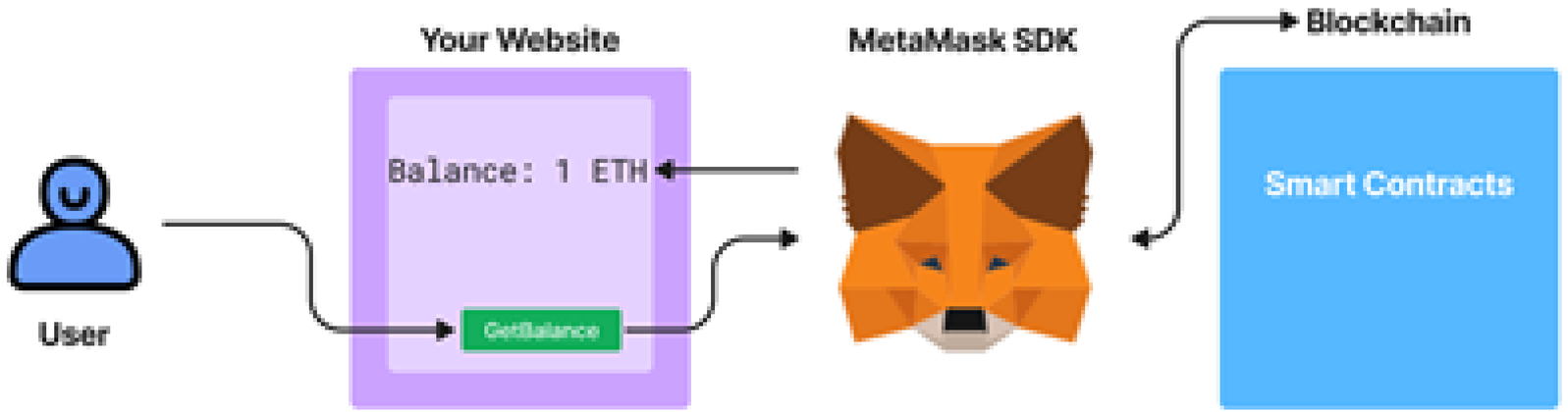
Benefits of a Smart Contract



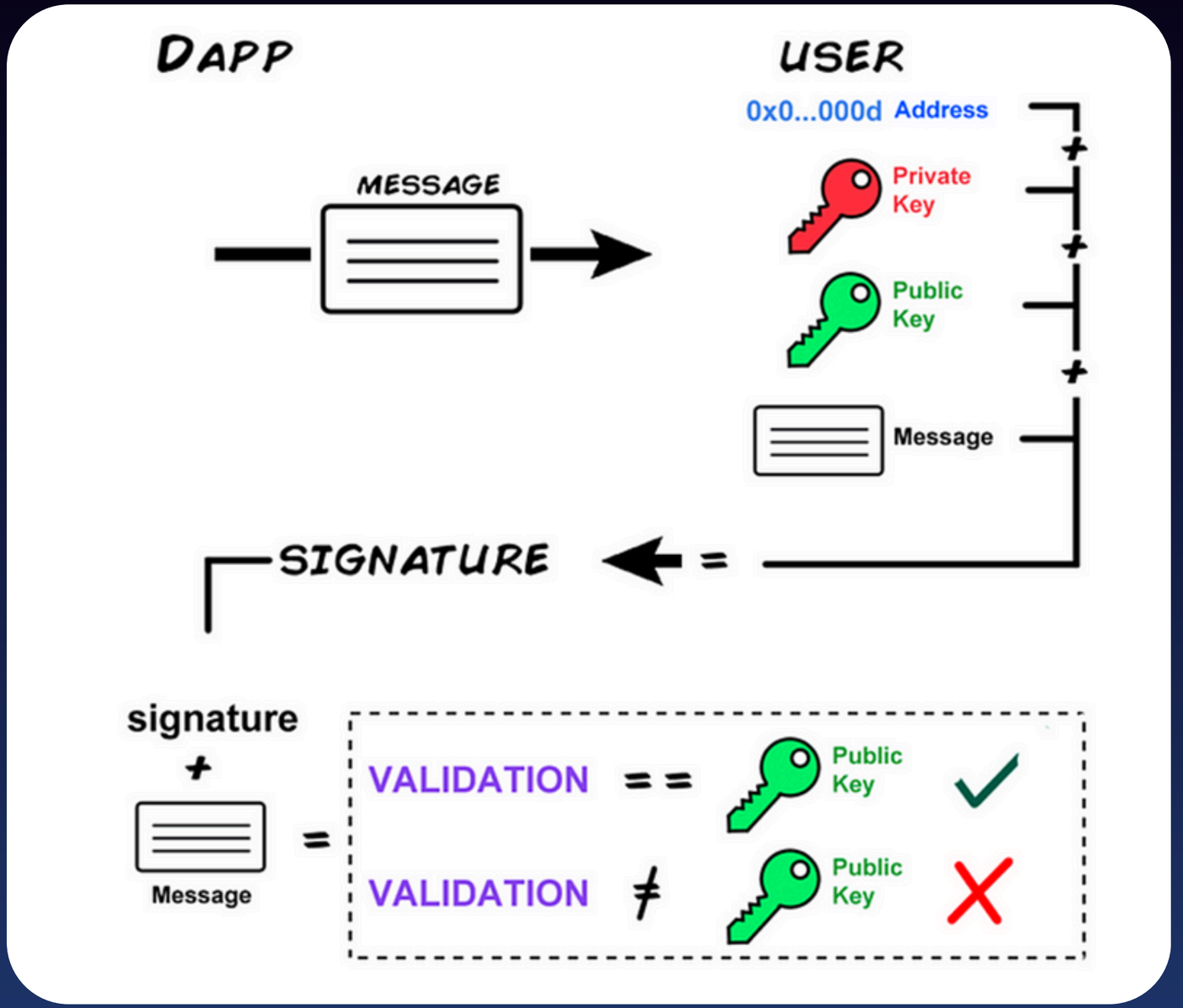
10 Top distributed apps (dApps) for blockchain



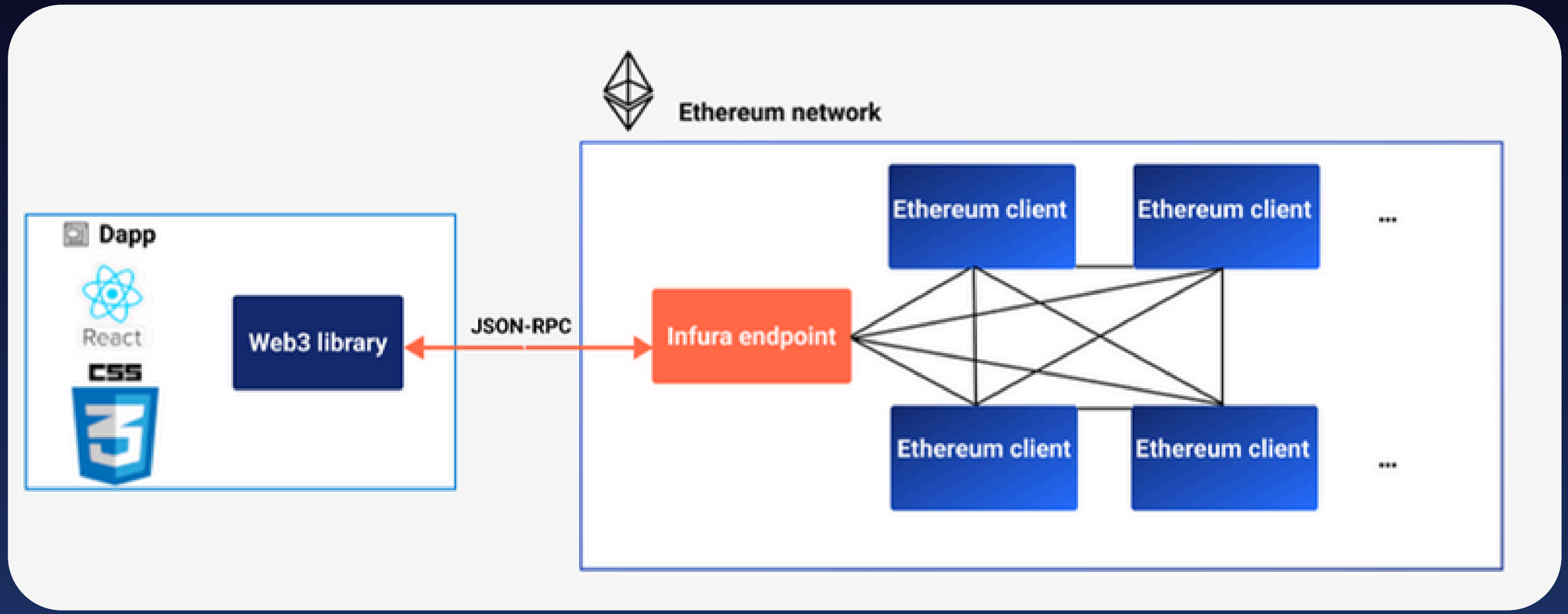
What is a dApp?



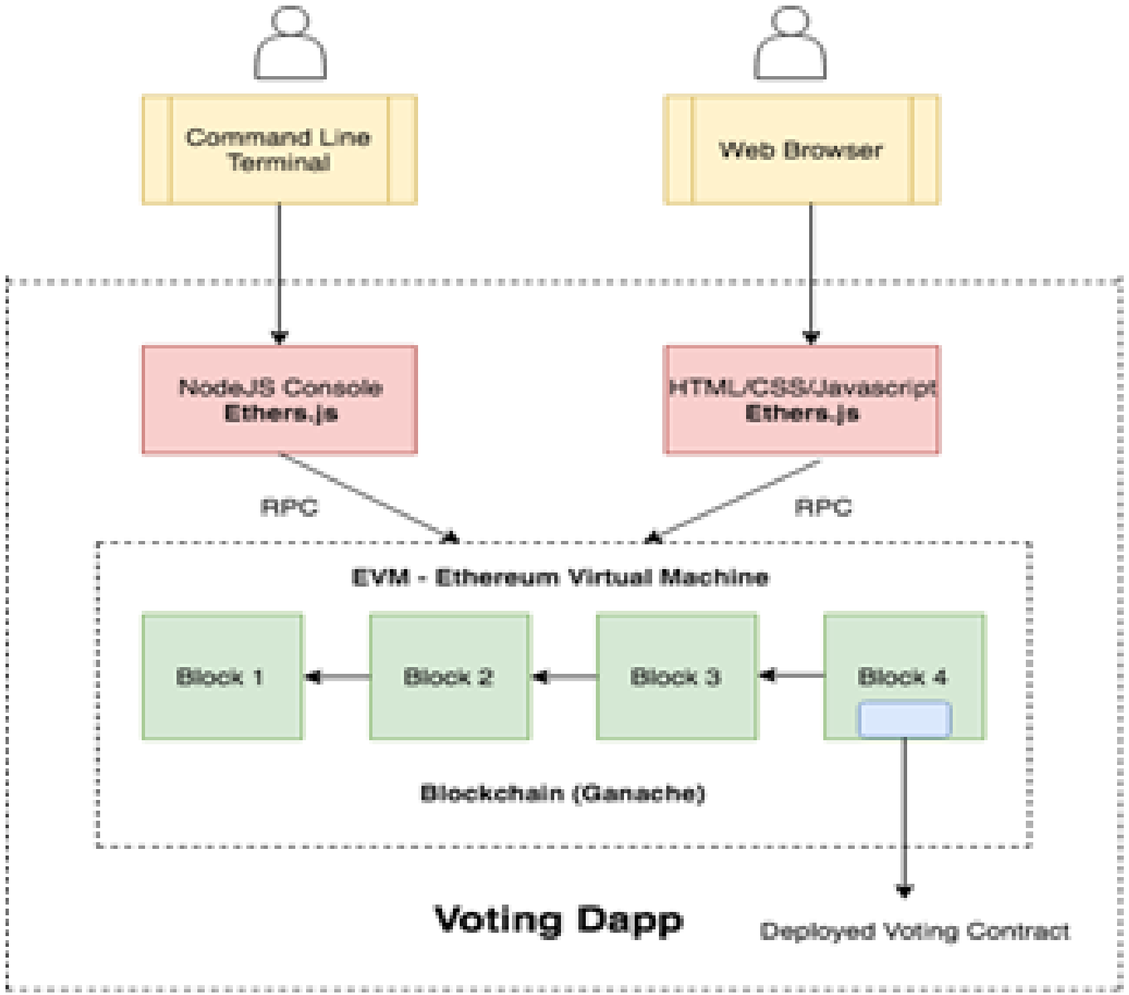
MetaMask SDK will provide a direct connection to the blockchain of your choice


DAPP
USER
0x0...000d Address
MESSAGE
Private Key
Public Key
Message
SIGNATURE
=
signature
+
Message
=
VALIDATION
==
Public Key
VALIDATION
≠
Public Key

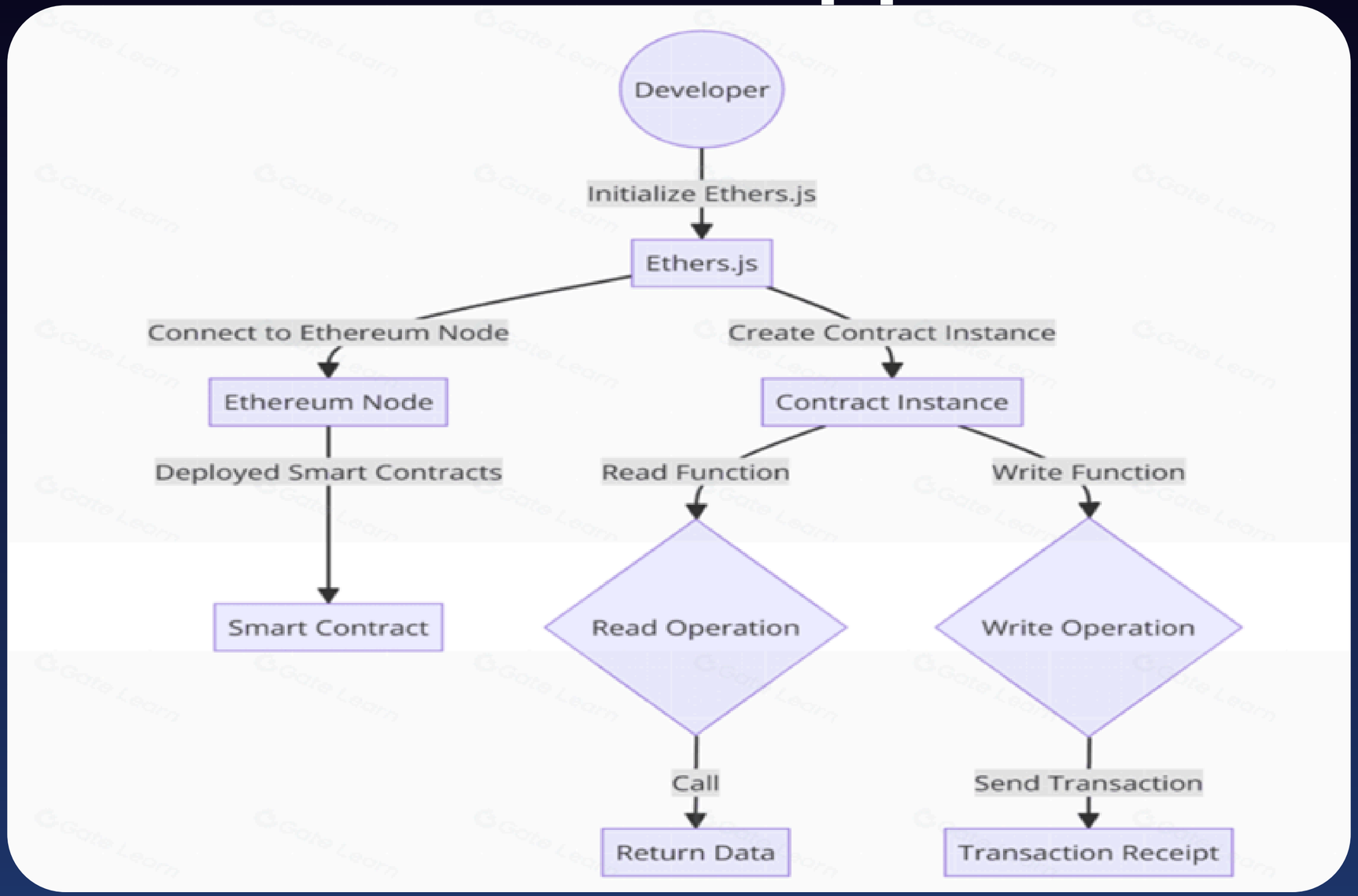
What is dApp


Ethereum network
Dapp
React
CSS
Web3 library
JSON-RPC
Infura endpoint
Ethereum client
Ethereum client
...
Ethereum client
Ethereum client
...

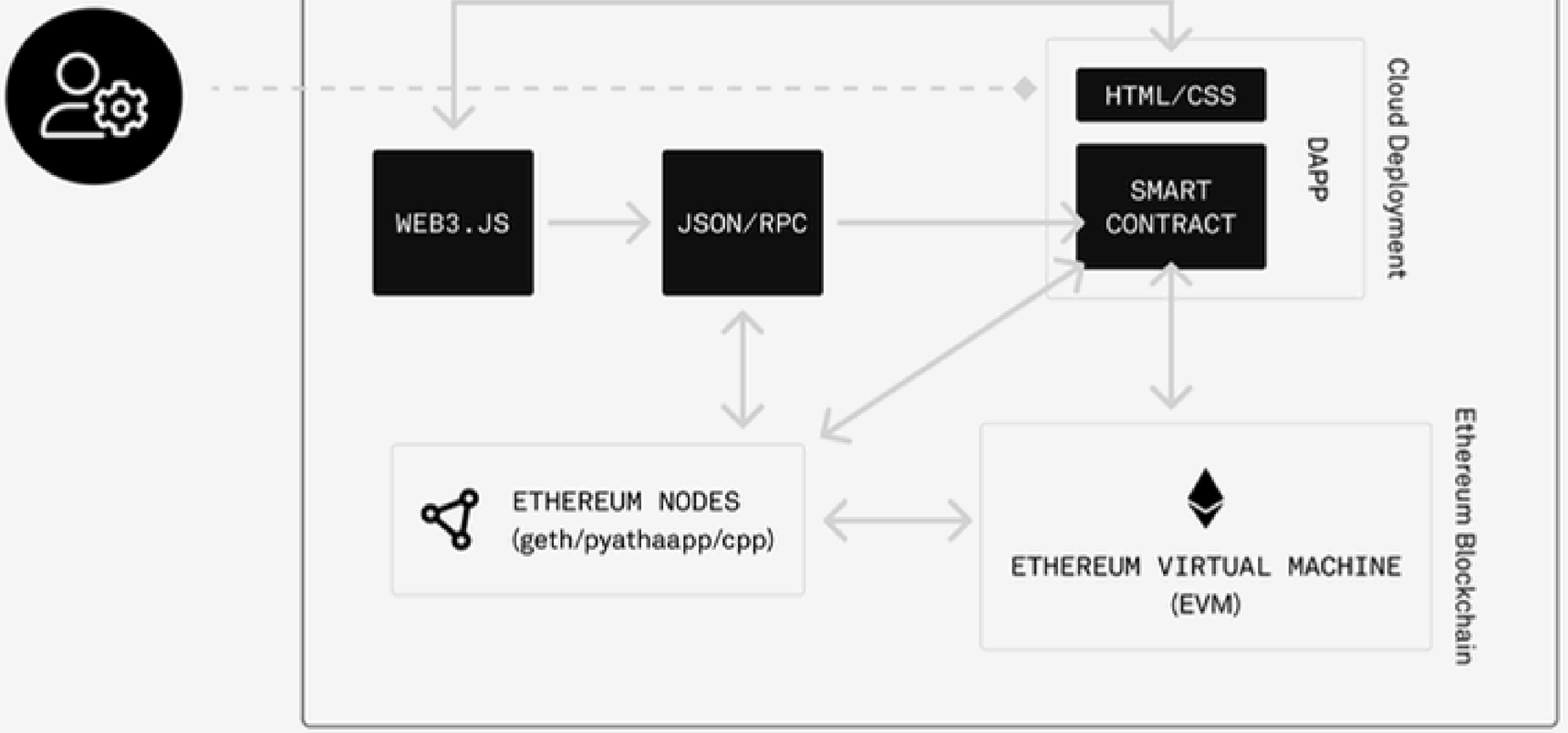
What is dApp


Command Line Terminal
Web Browser
NodeJS Console
Ethers.js
HTML/CSS/Javascript
Ethers.js
RPC
RPC
EVM - Ethereum Virtual Machine
Block 1
Block 2
Block 3
Block 4
Blockchain (Ganache)
Voting Dapp
Deployed Voting Contract

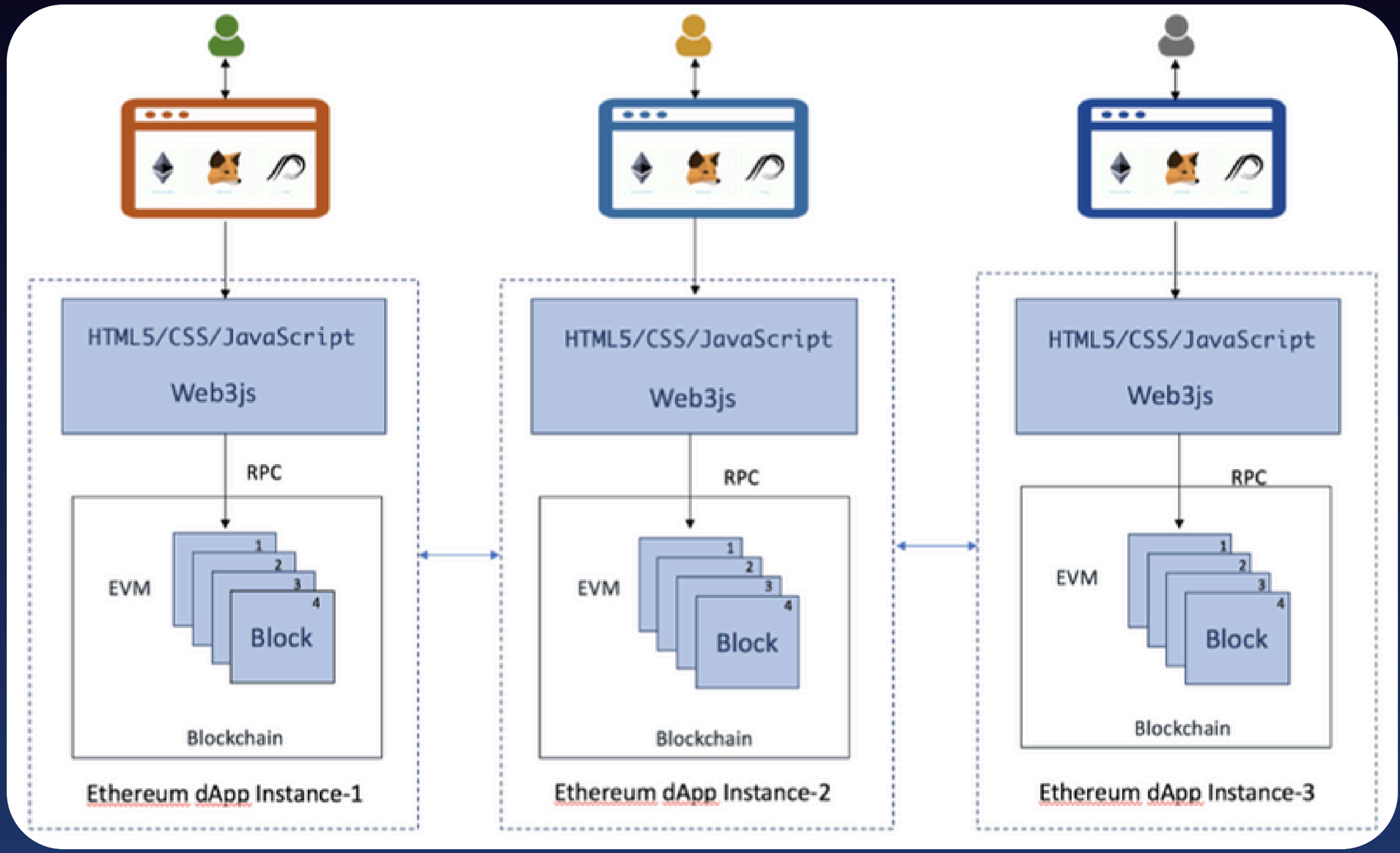
What is dApp


Developer
Initialize Ethers.js
Ethers.js
Connect to Ethereum Node
Create Contract Instance
Ethereum Node
Contract Instance
Deployed Smart Contracts
Read Function
Write Function
Smart Contract
Read Operation
Write Operation
Call
Send Transaction
Return Data
Transaction Receipt

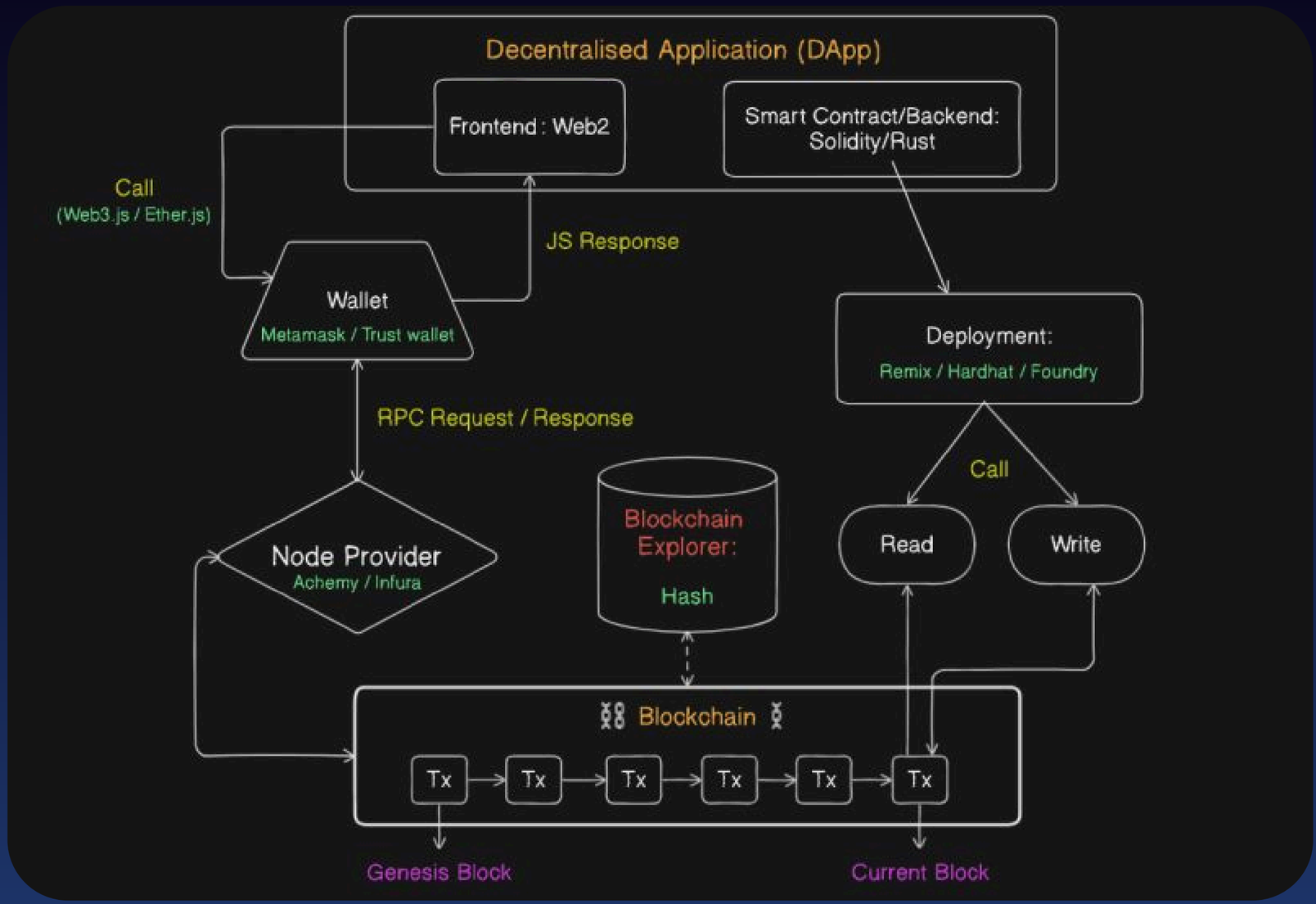
What is dApp


HTML/CSS
SMART CONTRACT
DAPP
Cloud Deployment
WEB3.JS
JSON/RPC
ETHEREUM NODES
(geth/pyathaapp/cpp)
ETHEREUM VIRTUAL MACHINE
(EVM)
Ethereum Blockchain

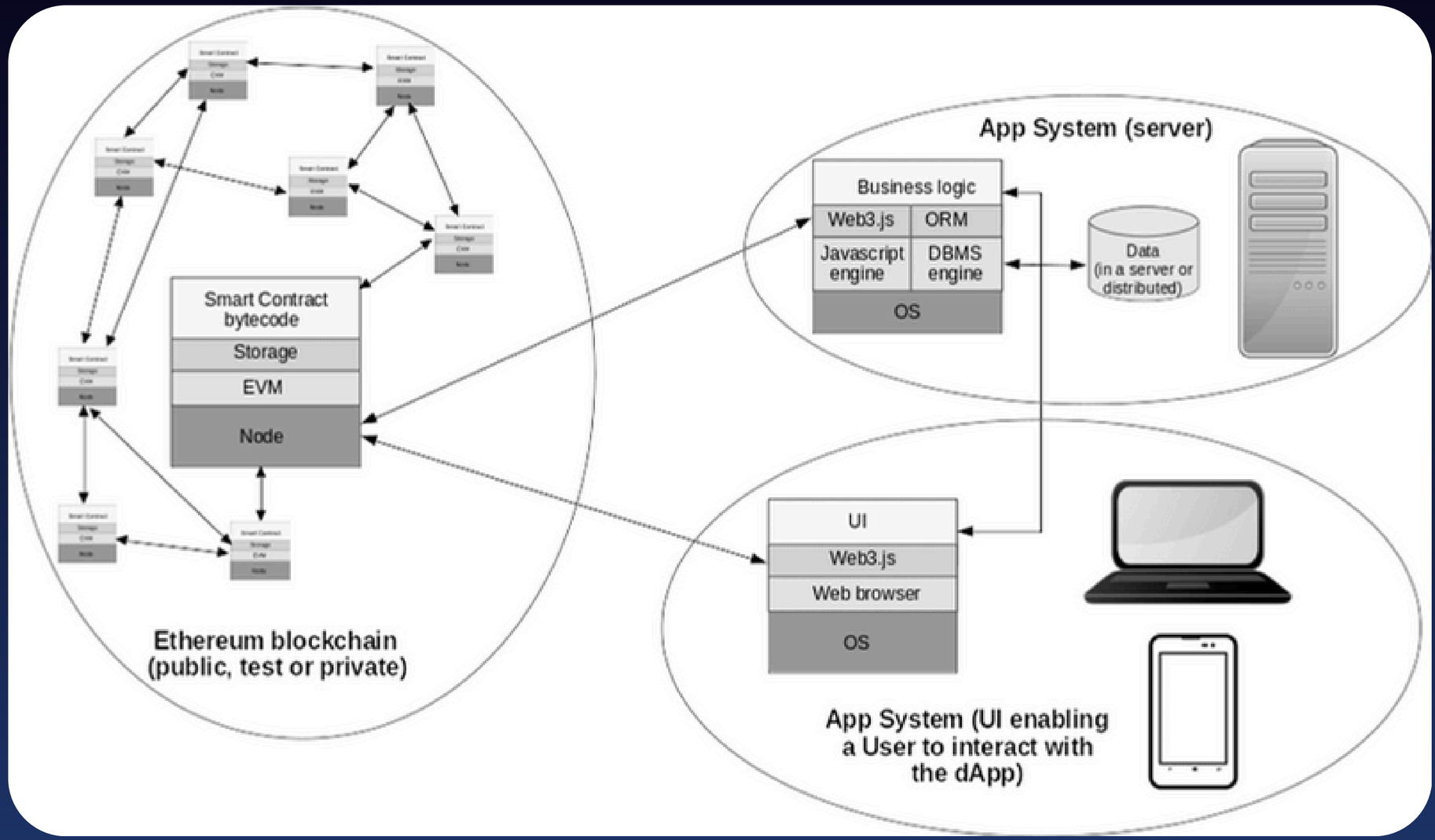
What is dApp


HTML5/CSS/JavaScript
Web3js
RPC
EVM
1
2
3
4
Block
Blockchain
Ethereum dApp Instance-1
HTML5/CSS/JavaScript
Web3js
RPC
EVM
1
2
3
4
Block
Blockchain
Ethereum dApp Instance-2
HTML5/CSS/JavaScript
Web3js
RPC
EVM
1
2
3
4
Block
Blockchain
Ethereum dApp Instance-3

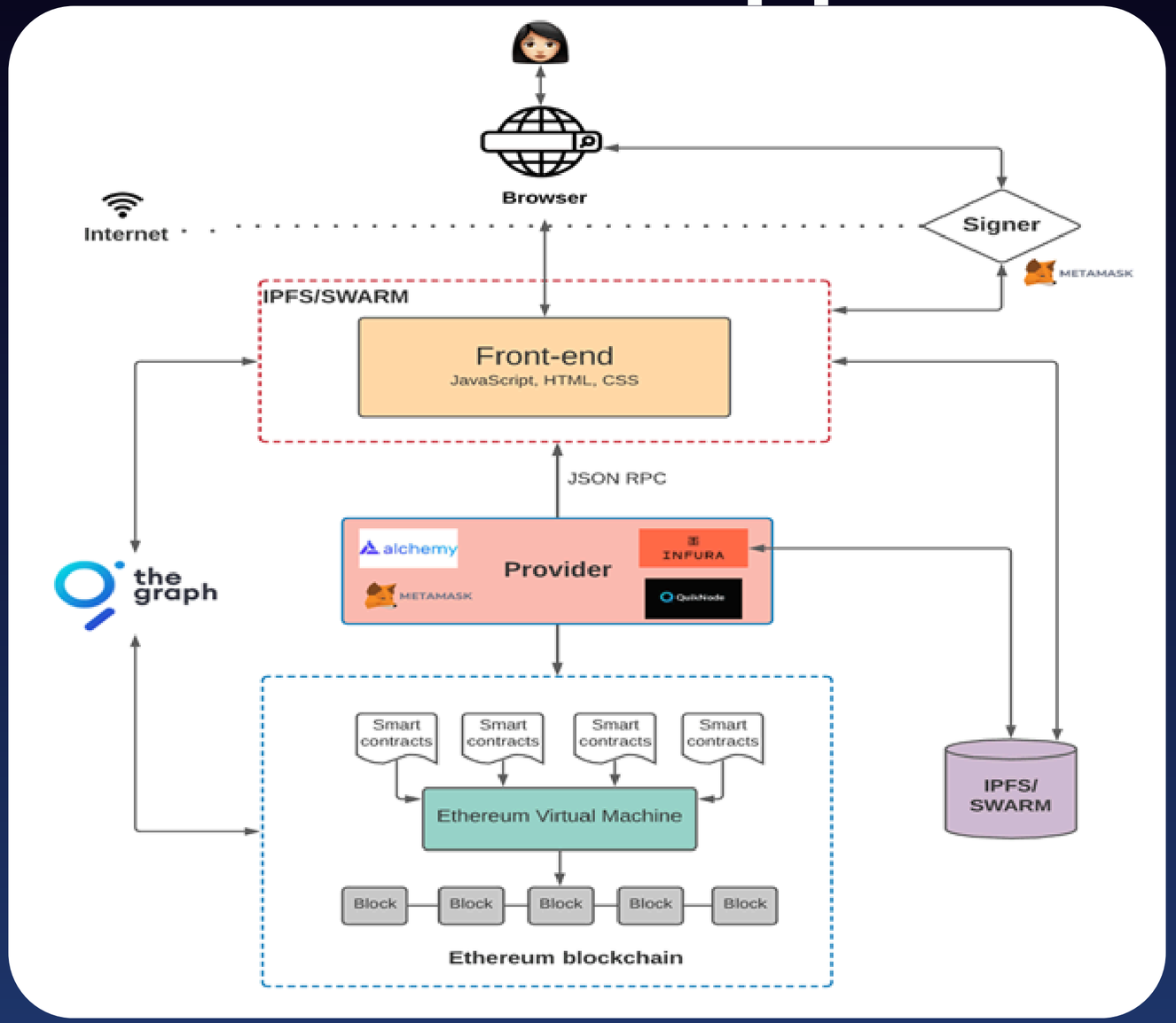
What is dApp


Decentralised Application (DApp)
Frontend : Web2
Smart Contract/Backend: Solidity/Rust
Call
(Web3.js / Ether.js)
JS Response
Wallet
Metamask / Trust wallet
Deployment:
Remix / Hardhat / Foundry
RPC Request / Response
Call
Blockchain Explorer:
Hash
Node Provider
Achemy / Infura
Read
Write
Blockchain
Tx
Tx
Tx
Tx
Tx
Tx
Genesis Block
Current Block

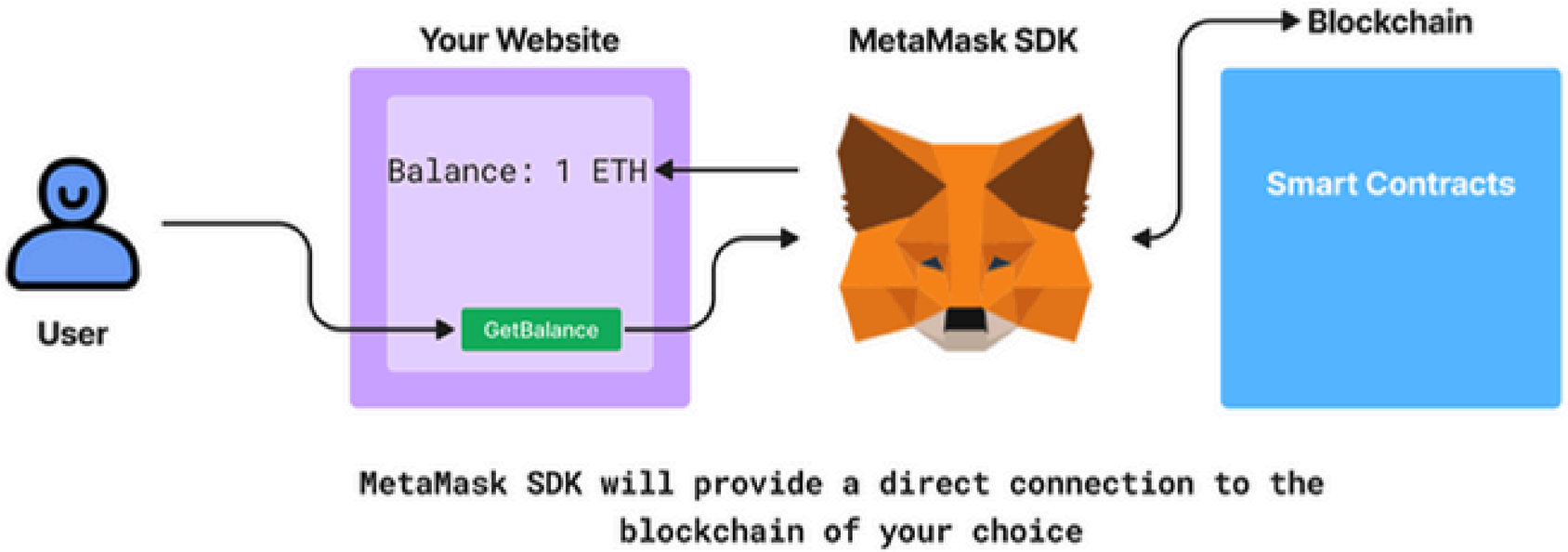
What is dApp


Smart Contract
bytecode
Storage
EVM
Node
Ethereum blockchain
(public, test or private)
App System (server)
Business logic
Web3.js
ORM
Javascript
engine
DBMS
engine
OS
Data
(in a server or
distributed)
UI
Web3.js
Web browser
OS
App System (UI enabling
a User to interact with
the dApp)

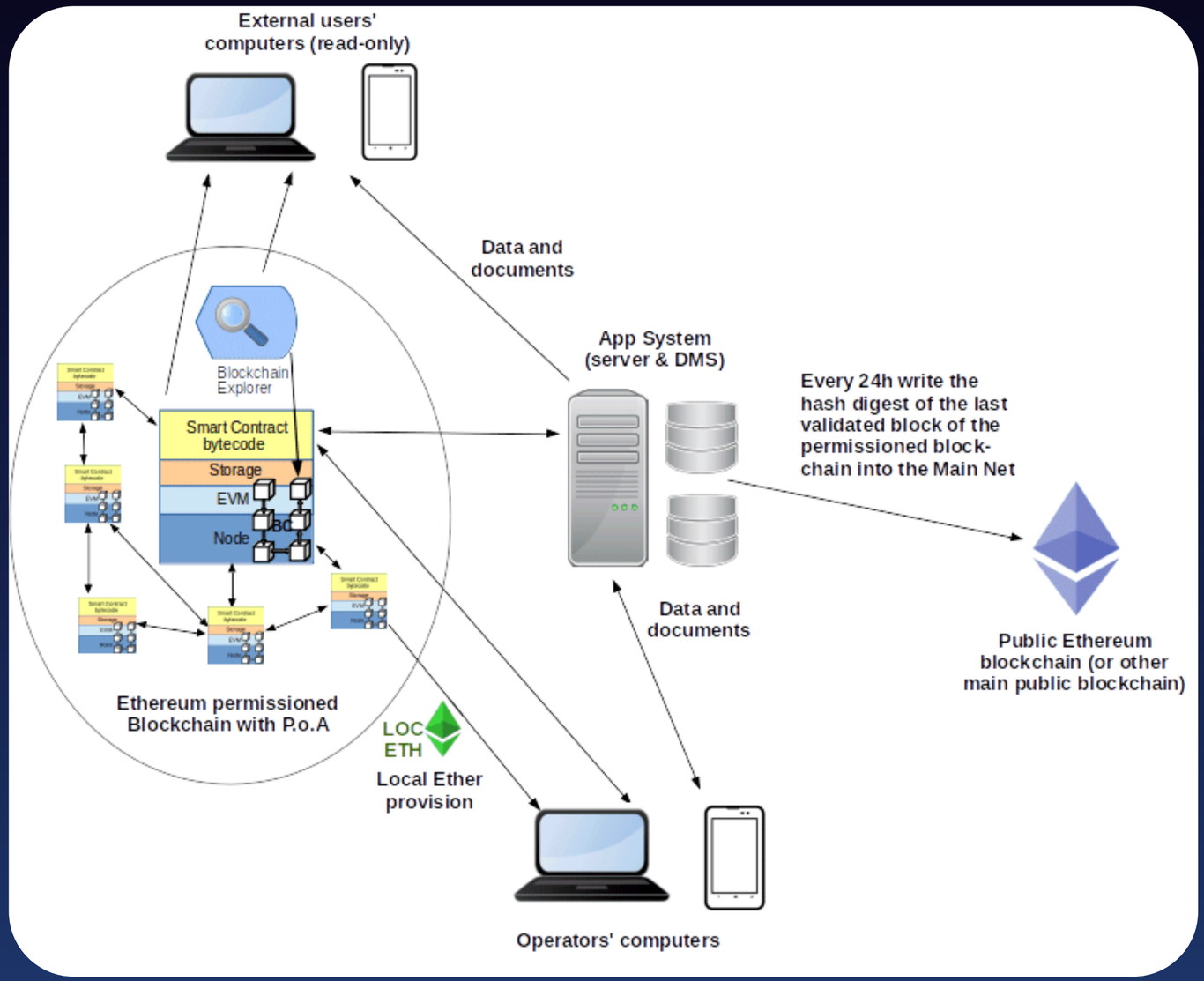
What is dApp


Browser
Internet
Signer
METAMASK
IPFS/SWARM
Front-end
JavaScript, HTML, CSS
JSON RPC
alchemy
METAMASK
Provider
INFURA
QuikNode
the graph
Smart contracts
Smart contracts
Smart contracts
Smart contracts
Ethereum Virtual Machine
Block
Block
Block
Block
Block
Ethereum blockchain
IPFS/ SWARM

What is dApp


Your Website
MetaMask SDK
Blockchain
Balance: 1 ETH
Smart Contracts
User
GetBalance
MetaMask SDK will provide a direct connection to the
blockchain of your choice

What is dApp


External users' computers (read-only)
Data and documents
Blockchain Explorer
App System (server & DMS)
Every 24h write the hash digest of the last validated block of the permissioned block-chain into the Main Net
Smart Contract bytecode
Storage
EVM
Node
BC
Data and documents
Public Ethereum blockchain (or other main public blockchain)
Ethereum permissioned Blockchain with P.o.A
LOC
ETH
Local Ether provision
Operators' computers

🧩 Step-by-Step dApp Development Guide

dApp Development Step by Step

✅ 1. Understand the Requirements and Choose the Use Case

- Identify the problem you want to solve with a decentralized approach.
- Decide why decentralization is necessary (transparency, trustlessness, immutability).
- Common dApp types: finance (DeFi), gaming (GameFi), identity, marketplaces, etc.

dApp Development Step by Step

✅ Essential Components of the dApp Stack

Layer	Description	Common Tools/Stacks
Smart Contract Layer	Backend logic for the token and transfer functions	**Solidity**, **Vyper**
Blockchain Network	Public or test blockchain to deploy and interact with contracts	**Ethereum (Mainnet/Testnet like Sepolia, Goerli)**, **Polygon**, **BNB Smart Chain**
Development Framework	Tools for writing, testing, and deploying smart contracts	**Hardhat**, **Truffle**, **Foundry**
Token Standard	Defines the rules for token creation and transfer	**ERC-20** (fungible tokens), **ERC-721/ERC-1155** (NFTs)
Frontend	UI for users to interact with the dApp	**React.js**, **Next.js**, **Vue.js**, **HTML/CSS/JavaScript**
Wallet Integration	Connects users' crypto wallets to the dApp	**MetaMask**, **WalletConnect**, **Coinbase Wallet SDK**
Blockchain JS Library	Connects frontend to blockchain	**Ethers.js**, **Web3.js**

dApp Development Step by Step

2. Development Framework

Framework	Pros	Cons
Hardhat	Fast, modern, plugin system, integrated with Ethers.js	Slightly steeper learning curve for new devs
Truffle	More beginner-friendly, integrated with Ganache	Slower than Hardhat
Foundry	Rust-like syntax, very fast, testing in Solidity	Newer ecosystem, less documentation

✅ **Recommendation: Hardhat**

> It's widely used, actively maintained, and integrates perfectly with Ethers.js for frontend interaction.

dApp Development Step by Step

3. Blockchain Network

Network	Type	Gas Fees	Speed	Ecosystem Support
Ethereum Sepolia	Testnet	Free	Medium	Best for Ethereum testing
Polygon	Mainnet	Low	Fast	Good for cheaper dApps
Binance Smart Chain	Mainnet	Low	Fast	Popular for low-cost dApps

✅ **Start with: Ethereum Sepolia Testnet**

> Easy to obtain test ETH, widely supported by Hardhat, MetaMask.

dApp Development Step by Step

4. Frontend (UI)

Framework	Pros	Cons
React.js	Component-based, large community	Slightly larger bundle size
Next.js	SEO-friendly, server-side rendering	Bit more setup needed
HTML/CSS/JS	Simple for static sites	Not modular or scalable

✅ **Recommendation: React.js** or **Next.js**

> React is the most commonly used framework in dApp frontends.

dApp Development Step by Step

5. Blockchain JS Library

Library	Pros	Cons
Ethers.js	Lightweight, modern, better TypeScript support	Less support for older contracts
Web3.js	Older, widely used	Heavier, more complex API

✅ **Recommendation: Ethers.js**

> Preferred with Hardhat; easier to integrate with MetaMask and simpler syntax.

dApp Development Step by Step

6. Wallet Integration

- **MetaMask**
  - Easiest and most common for Ethereum users.
  - Connects with Ethers.js easily.
  - Browser extension and mobile versions available.

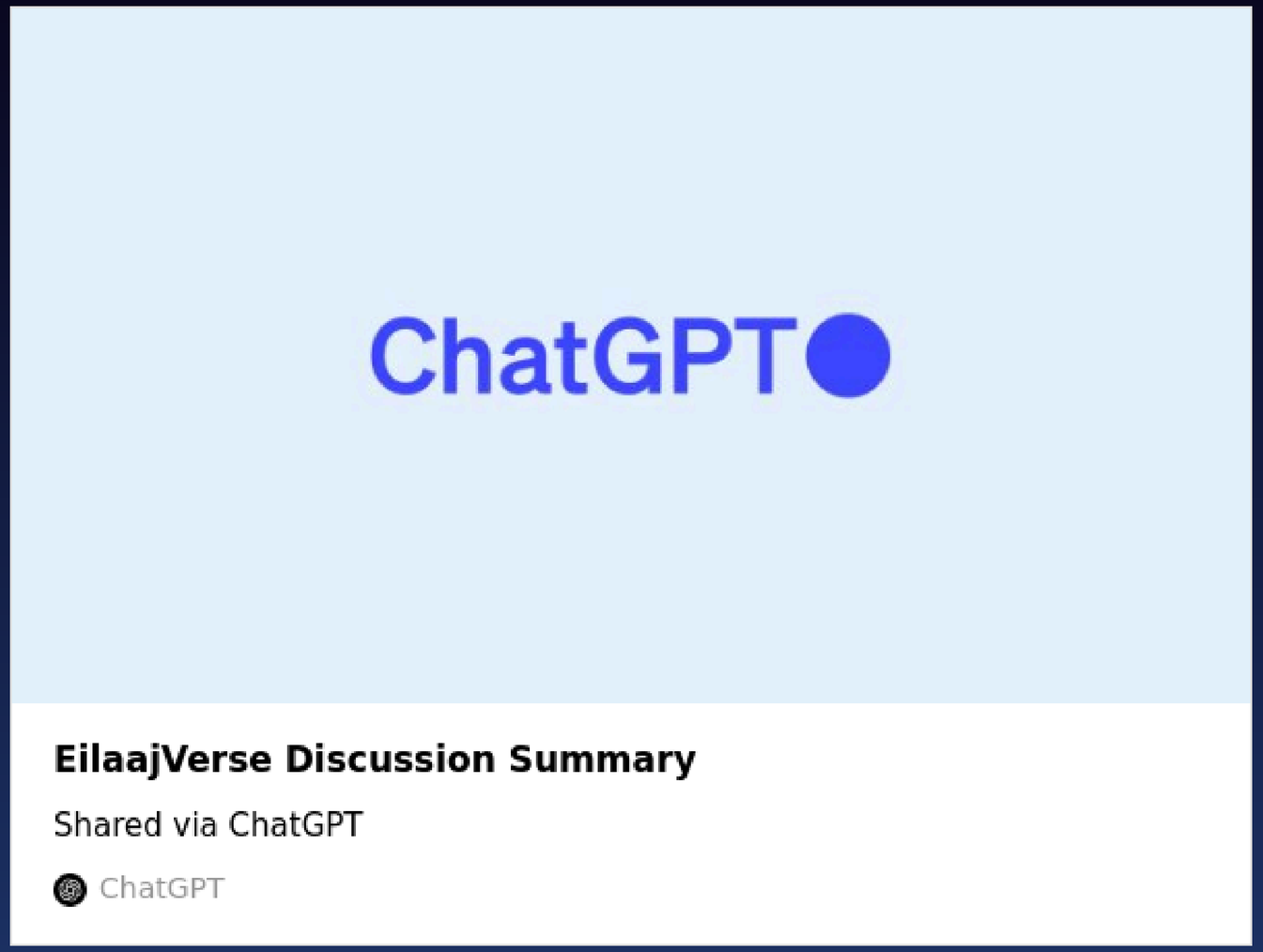
✅ **Recommendation: MetaMask**

dApp Development Step by Step

✅ Example Stack for Token Transfer dApp

Layer	Tool/Stack
Smart Contract	Solidity
Framework	Hardhat
Blockchain	Ethereum Sepolia
Token Standard	ERC-20
Frontend	React.js + TailwindCSS
JS Library	Ethers.js
Wallet	MetaMask

dApp Development Step by Step



dApp Development Step by Step

- **Solidity** (ERC-20 Token Smart Contract)
- **Hardhat** (for development & deployment)
- **MetaMask** (to connect users)
- **Ethers.js** (to interact with smart contract)
- **Frontend** in **HTML** + **CSS** + **JavaScript** (no React)

dApp Development Step by Step

✅ PART 1: Backend – Token Contract with Hardhat

1. Create project and install dependencies

```bash
mkdir token-transfer-dapp && cd token-transfer-dapp
npm init -y
npm install --save-dev hardhat
npx hardhat
# Select: "Create a basic sample project"
```

dApp Development Step by Step

✅ PART 1: Backend – Token Contract with Hardhat

2. Install dependencies

```bash
npm install @openzeppelin/contracts @nomicfoundation/hardhat-toolbox ethers
```

dApp Development Step by Step

✅ PART 1: Backend – Token Contract with Hardhat

3. Create token contract at contracts/Token.sol

solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract Token is ERC20 {
    constructor(uint256 initialSupply) ERC20("DemoToken", "DTK") {
        _mint(msg.sender, initialSupply);
    }
}
```

dApp Development Step by Step

✅ PART 1: Backend – Token Contract with Hardhat

4. Deployment Script scripts/deploy.js

```js
async function main() {
  const Token = await ethers.getContractFactory("Token");
  const token = await Token.deploy(ethers.utils.parseEther("1000000"));
  await token.deployed();
  console.log("Token deployed to:", token.address);
}
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

dApp Development Step by Step

✅ PART 1: Backend – Token Contract with Hardhat

5. Configure hardhat.config.js

```js
require("@nomicfoundation/hardhat-toolbox");

module.exports = {
  solidity: "0.8.18",
  networks: {
    sepolia: {
      url: "https://sepolia.infura.io/v3/YOUR_INFURA_PROJECT_ID",
      accounts: ["YOUR_PRIVATE_KEY"]
    }
  }
};
```

dApp Development Step by Step

✅ PART 1: Backend – Token Contract with Hardhat

6. Deploy the contract

```bash
npx hardhat run scripts/deploy.js --network sepolia
```

Copy the contract address.

dApp Development Step by Step

dApp Development Step by Step

File: index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Token Transfer dApp</title>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <div class="container">
    <h1>Token Transfer dApp</h1>
    <button id="connectWallet">Connect Wallet</button>
    <p id="walletAddress"></p>
    <p id="tokenBalance"></p>

    <input type="text" id="recipient" placeholder="Recipient Address" />
    <input type="text" id="amount" placeholder="Amount to Send" />
    <button id="sendToken">Send Token</button>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/ethers@5.7.2/dist/ethers.umd.min.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

dApp Development Step by Step

File: style.css

```css
body {
  font-family: Arial, sans-serif;
  background: #f2f2f2;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.container {
  background: white;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

input, button {
  display: block;
  margin: 10px 0;
  padding: 10px;
  width: 100%;
}

button {
  background-color: #007bff;
  color: white;
  border: none;
  cursor: pointer;
}
```

dApp Development Step by Step

File: app.js

```
const CONTRACT_ADDRESS = "YOUR_DEPLOYED_CONTRACT_ADDRESS";

const ABI = [
  "function transfer(address to, uint amount) public returns (bool)",
  "function balanceOf(address account) external view returns (uint256)"
];

let signer, contract;

document.getElementById("connectWallet").onclick = async () => {
  if (window.ethereum) {
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    await provider.send("eth_requestAccounts", []);
    signer = provider.getSigner();
    const address = await signer.getAddress();
    document.getElementById("walletAddress").innerText = `Wallet: ${address}`;

    contract = new ethers.Contract(CONTRACT_ADDRESS, ABI, signer);
    const balance = await contract.balanceOf(address);
    document.getElementById("tokenBalance").innerText =
      `Balance: ${ethers.utils.formatEther(balance)} DTK`;
  } else {
    alert("Please install MetaMask!");
  }
};

document.getElementById("sendToken").onclick = async () => {
  const recipient = document.getElementById("recipient").value;
  const amount = document.getElementById("amount").value;
  if (!recipient || !amount) {
    alert("Enter both recipient and amount");
    return;
  }

  const tx = await contract.transfer(recipient, ethers.utils.parseEther(amount));
  await tx.wait();
  alert("Transfer successful!");
};
```

dApp Development Step by Step

✅ How to Use

1. Host `index.html`, `style.css`, `app.js` in a local or online server.
2. Connect MetaMask and switch to Sepolia testnet.
3. Use one wallet to send tokens to another.
4. Change MetaMask account and repeat the transfer.

dApp Development Step by Step

(dApp No. 2)

dApp Development Step by Step

✅ 2. Choose the Right Tech Stack

Layer	Tools
Smart Contract	Solidity (on Ethereum), Vyper
Blockchain Environment	Ethereum (mainnet/testnet), Polygon, BNB Chain
Local Dev Tools	Hardhat, Truffle
Frontend	React.js / Next.js
Wallet Integration	MetaMask, WalletConnect
Backend/Storage	IPFS, Filecoin, The Graph (optional), or off-chain with Node.js
Oracles	Chainlink (if external data is needed)

dApp Development Step by Step

✅ 3. Set Up Your Development Environment

Explanation:

Install necessary tools:

bash Copy Edit

```
npm install --save-dev hardhat
npx hardhat
```

Other essentials:

- Node.js
- MetaMask browser extension
- GitHub repo for version control

dApp Development Step by Step

✅ 4. Write Smart Contracts

Example (ERC20 Token in Solidity):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "MTK") {
        _mint(msg.sender, 1000000 * 10 ** decimals());
    }
}
```

dApp Development Step by Step

✅ 4. Write Smart Contracts

🛠️ Test contracts using Hardhat or Truffle:

bash Copy Edit

```
npx hardhat test
```

dApp Development Step by Step

✅ 5. Deploy Smart Contracts to a Testnet

- Choose testnet (Goerli, Sepolia, Mumbai, etc.)
- Configure Hardhat/Truffle for testnet
- Use MetaMask to manage testnet wallet and get test ETH from a faucet

Deploy with Hardhat:

bash Copy Edit

```
npx hardhat run scripts/deploy.js --network sepolia
```

dApp Development Step by Step

✅ 6. Build a Frontend

Use React.js or Next.js for building your dApp UI.

Common UI Components:

- Connect Wallet Button
- Display user balance
- Buttons to trigger contract functions (stake, transfer, vote, etc.)

Example:

javascript Copy Edit

```javascript
const connectWallet = async () => {
  if (window.ethereum) {
    const accounts = await window.ethereum.request({ method: "eth_requestAccounts" });
    setAccount(accounts[0]);
  }
};
```

dApp Development Step by Step

✅ 7. Integrate Smart Contract with Frontend (Web3 Integration)

Use libraries like ethers.js or web3.js to connect your smart contract to your UI.

Example with ethers.js:

javascript Copy Edit

```javascript
import { ethers } from "ethers";
const provider = new ethers.providers.Web3Provider(window.ethereum);
const signer = provider.getSigner();
const contract = new ethers.Contract(contractAddress, abi, signer);
```

dApp Development Step by Step

✅ 8. Store Data (Optional: IPFS, The Graph, Off-chain DB)

Blockchain storage is costly. For large data (images, files), use:

- IPFS for decentralized file storage.
- The Graph for querying blockchain data.
- Firebase/MongoDB if off-chain logic is acceptable.

dApp Development Step by Step

✅ 9. Test the Entire dApp

- Perform integration testing
- Use testnet to mimic user behavior
- Simulate edge cases, reentrancy attacks, overflow bugs, etc.

Tools:

- Hardhat for unit testing
- Foundry or MythX for security testing

dApp Development Step by Step

✅ 10. Deploy to Mainnet

- Ensure code is optimized and audited
- Deploy contract using Hardhat/Remix
- Point your frontend to the mainnet version

dApp Development Step by Step

✅ 11. Launch and Maintain the dApp

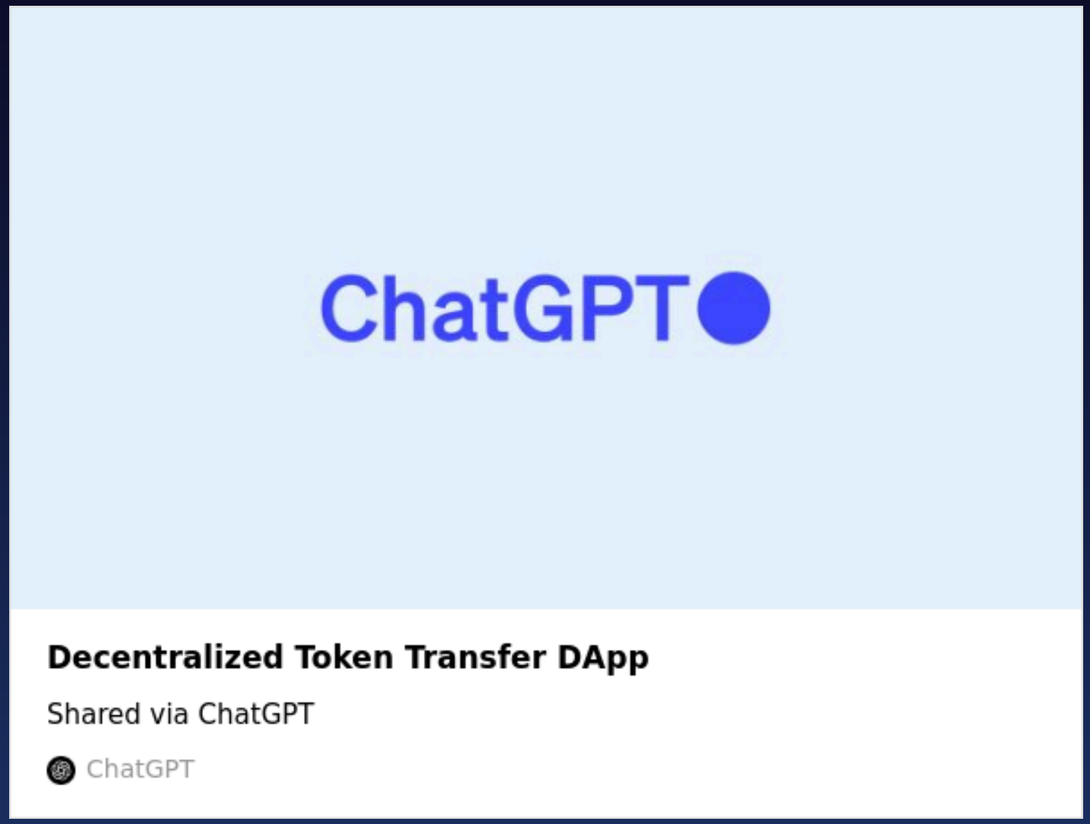
- Share the dApp via IPFS, Fleek, or centralized hosting (like Vercel, Netlify)
- Promote via Web3 communities
- Keep UI updated as needed (Smart contract is immutable!)

dApp Development Step by Step

✅ 12. Bonus: Security Best Practices

- Use OpenZeppelin contracts
- Run audits (manual or tools like Slither)
- Avoid reentrancy, overflow, and access control issues

dApp Development Step by Step



THANK-YOU