



Développer et déployer des contrats intelligents sur Ethereum

Solidity – Remix – Hardhat
Ethers.js – OpenZeppelin – Pinata

Aperçu de la formation 1/2

- Les bases : Réseau Ethereum, blockchain, comptes Ethereum, transactions, ETH, wei & gaz...
- Metamask : Installation, configuration...
- Remix : Compilation, déploiement et débogage des contrats intelligents
- Hello World : Notre premier contrat intelligent
- Hardhat : Mise en place d'un environnement de développement local pour compiler, tester et déployer des contrats intelligents
- Solidity : Types (struct, mappings...), fonctions (receive, fallback...), modificateurs (modifier), événements, traitement des erreurs, héritage...
- Un contrat intelligent plus avancé : Contrat de vote



Aperçu de la formation 2/2

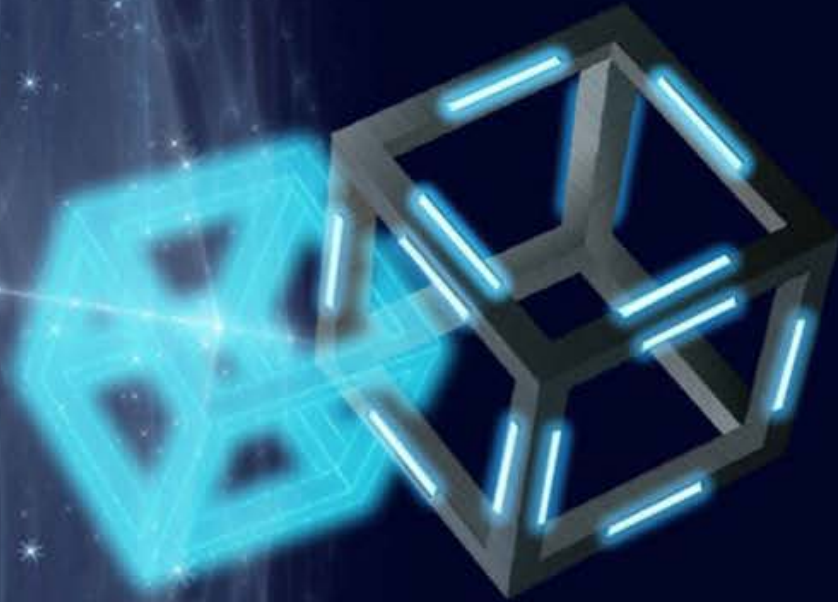
- Création d'une application web en utilisant React et Ethers.js (Provider, Signer, Contract, Utils)
- Le RPC API de Metamask
- Création d'un token ERC20 en utilisant OpenZeppelin
- Tester des contrats intelligents avec Hardhat en utilisant Mocha et Chai
- Création d'un token ERC721 (NFT) avec OpenZeppelin, IPFS et Pinata
- Protection du code contre les attaques de réentrée
- Création d'une organisation décentralisée autonome (DAO) en utilisant des contrats suivants : GovernanceToken, Treasury, Timelock et Governance



Slides, documents, code source...

Télécharger depuis : <https://github.com/rspadinger/ETH-Course>





Introduction au développement des contrats intelligents sur Ethereum

Les bases : Réseau, blockchain, comptes...

Le réseau Ethereum

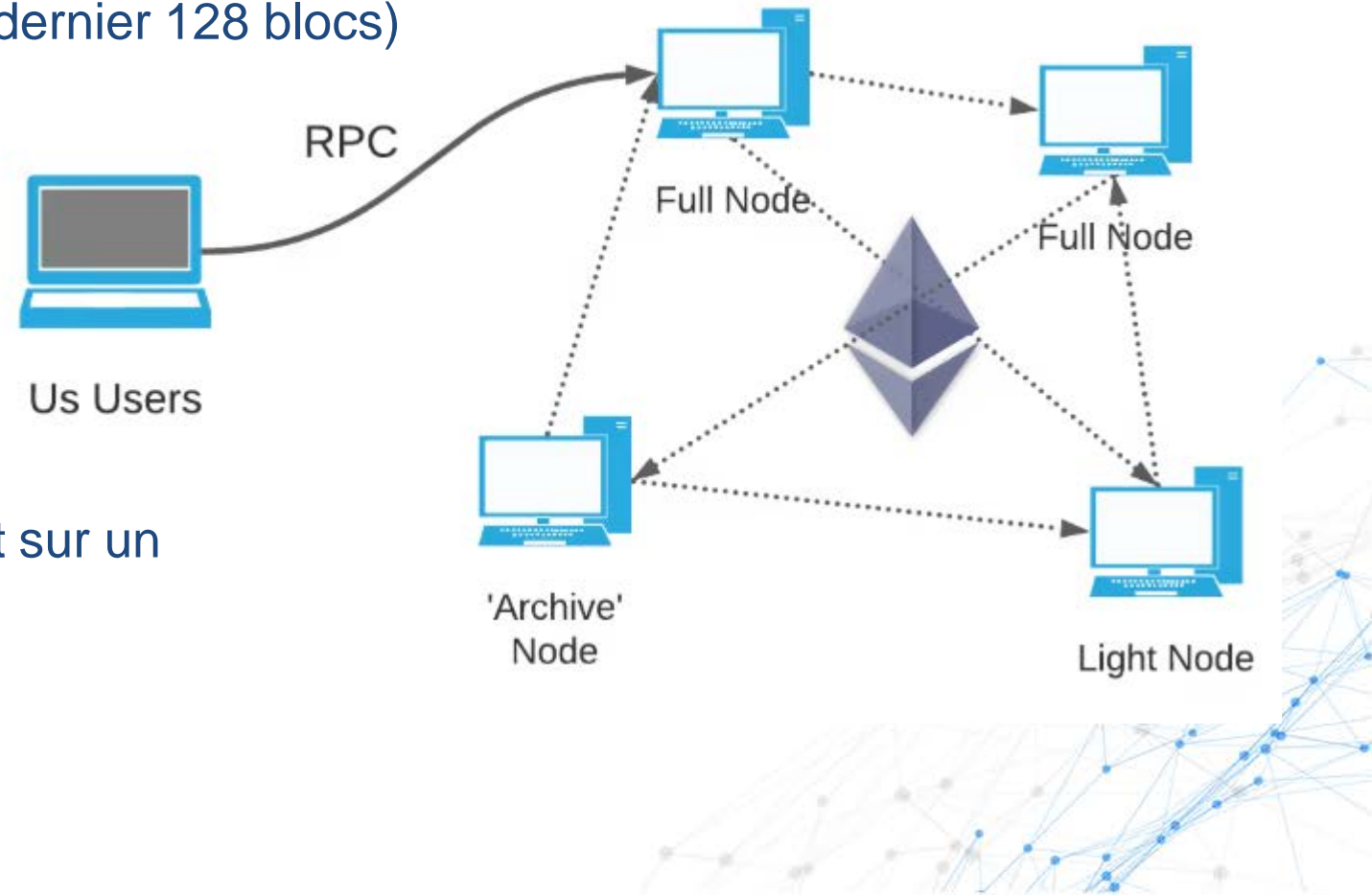
- Créer par Vitalik Buterin (début travail en 2013) – première version production en 2015
- Open-source, public, décentralisé, sans permission, immuable
- Permet le déploiement et l'exécution des contrats intelligents. La plupart des contrats sont développés en Solidity (Turing-complet)
- Réseaux P2P – 2000 nœuds avec des clients Ethereum (Geth, Nethermind, Erigon...)
- Nœud : Permet aux applications et utilisateurs d'accéder & modifier les données sur la blockchain
- Vérification et validation des transactions, envoi des données modifiées aux autres nœuds connectés => jusqu'à tous les nœuds sont à jour
- 3 types de nœuds : Nœud complet, nœud d'archive et nœud léger



Le réseau Ethereum – les types des nœuds

Nœuds complets :

- Stocker le dernier état de la blockchain (dernier 128 blocs)
- Interroger les données sur la blockchain
- Traiter des transactions (envoyer ETH, exécuter des fonctions) et valider des nouveaux blocs
- Un contrat peut être déployé directement sur un nœuds complet
- Nécessite ~ 2 To d'espace disque



Le réseau Ethereum – les types des nœuds

Nœud d'archive :

- Stocker l'histoire complet de la blockchain (requis par un explorateur de blockchain)
- Nécessite ~ 10 To d'espace disque

Nœud léger :

- Stocke uniquement les en-têtes des blocs (accès aux informations très élémentaires – hash du bloc, timestamp...)
- Contient souvent les données d'un seul compte (application portefeuille)
- Nécessite l'aide d'un nœud complet pour obtenir des données supplémentaires ou pour exécuter des transactions
- Ne participe pas à la validation des blocs
- Nécessite ~ 400 Mo d'espace disque



Le réseau Ethereum – connexion

Connexion au réseaux Ethereum :

- Avec un plugin ou application comme Metamask – pour un utilisateur
- Avec un explorateur de blockchain (<https://etherscan.io>)
- En utilisant une librairie comme ethers.js ou web3.js – pour un développeur
- En utilisant d'un fournisseur tiers comme Alchemy, Infura...
- Installation d'un nœud complet avec un logiciel client (Geth, Nethermind...)



Le réseau Ethereum – les types des réseaux

Réseau principal



Blockchain principal
Persistente
Coûte de l'argent

Réseau de test

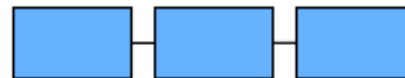
Sepolia...



Blockchain de test
Persistente (peut être supprimé)
Ne coûte pas de l'argent

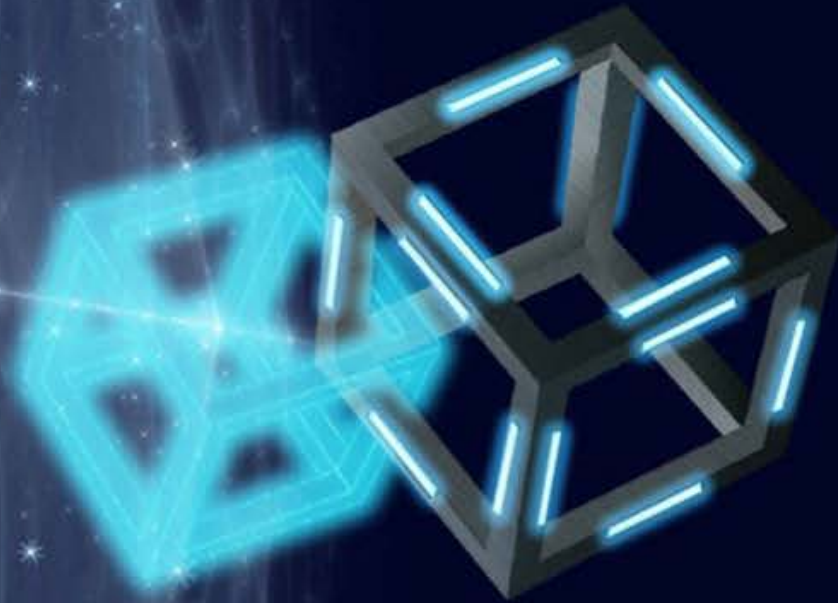
Réseau de développement

Ganache, Hardhat



Blockchain développement
Très rapide
N'est pas persistente
Ne coûte pas de l'argent





Introduction au développement des contrats intelligents sur Ethereum

Ether, wei & gaz

Le réseau Ethereum – ETH

ETH, wei & gaz (gas):

- Ether ou ETH est la monnaie utilisée par la blockchain Ethereum
- 1 ETH peut être divisée en 10^{18} wei (wei est la plus petite sous-unité de l'ETH)
- Pour exécuter une transaction (envoyer de l'ETH, exécution d'une fonction qui modifie l'état de la blockchain) il faut payer des frais. Les frais sont payés en unité de gaz
- La quantité de gaz à payer dépend de la complexité de la transaction à exécuter
- Le prix à payer par unité de gaz dépend de l'état du réseau



Le réseau Ethereum – limite & prix du gaz

Chaque opération (addition, soustraction...) consomme un certain quantité de gaz et le prix par unité de gaz dépend de l'état du réseau.

Limite de gaz : La quantité maximale de gaz que vous êtes prêt à dépenser pour l'exécution de votre transaction. Les unités de gaz qui ne sont pas utilisées seront remboursées. Si la limite spécifiée ne suffit pas, la transaction échouera.

Prix du gaz (en Gwei) : Le prix que vous êtes prêt à payer par unité de gaz pour l'exécution de votre transaction. Si vous spécifiez un prix plus élevé, la transaction pourra être exécuter plus rapidement.

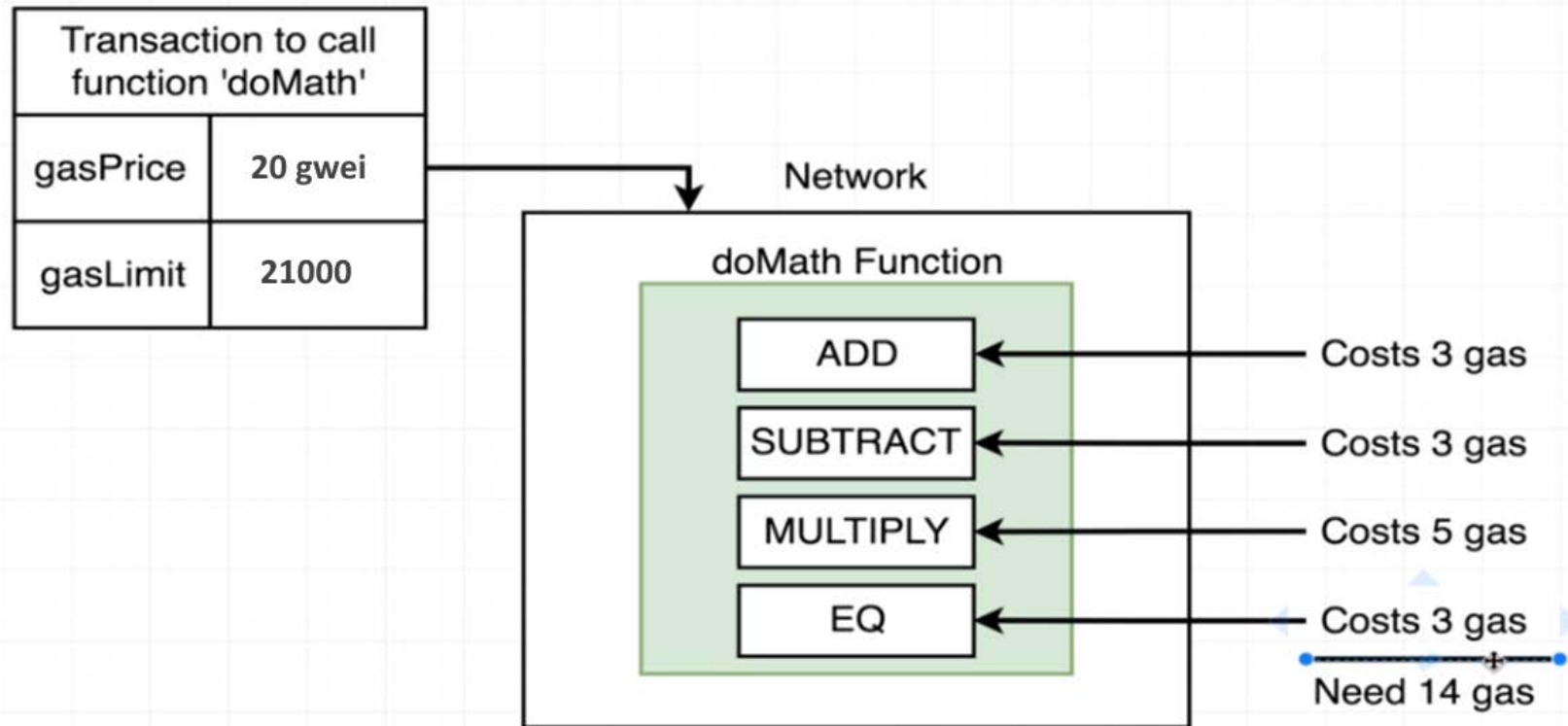
EVM Opcodes: <https://ethereum.org/en/developers/docs/evm/opcodes/>

OPCODES

Stack	Name	Gas	Initial Stack
00	STOP	0	
01	ADD	3	a, b
02	MUL	5	a, b
03	SUB	3	a, b
04	DIV	5	a, b
05	SDIV	5	a, b









Le réseau Ethereum – les frais d'une transaction



Coût total : $20 \text{ gwei/gaz} \times 14 \text{ gaz} = 28 \text{ gwei}$

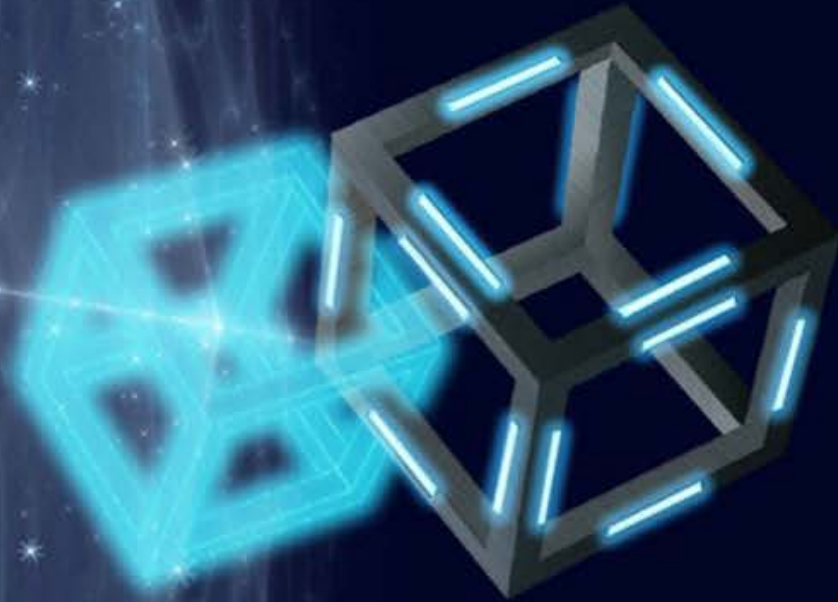


Le réseau Ethereum – frais d'une transaction

Transaction Hash:	0xa85520109eb43c31a0ea3df7b453ae5b859d22e02605db8a082433cf7a394503 
Status:	 Success
Block:	 18062033 2 Block Confirmations
Timestamp:	 21 secs ago (Sep-04-2023 08:50:47 AM +UTC)
Transaction Action:	▶ Transfer 0.022236801683902839 Ether To 0x2e0845...2640c20d
Sponsored:	
From:	0xBaF6dC2E647aeb6F510f9e318856A1BCd66C5e19 (MEV Builder: 0xBaF...e19) 
To:	0x2e08451B79a01Cda253811E45719CeB42640c20d 
Value:	◆ 0.022236801683902839 ETH (\$36.42)
Transaction Fee:	0.00040547801574 ETH (\$0.66)
Gas Price:	19.30847694 Gwei (0.000000001930847694 ETH)

<https://etherscan.io/tx/0xa85520109eb43c31a0ea3df7b453ae5b859d22e02605db8a082433cf7a394503>



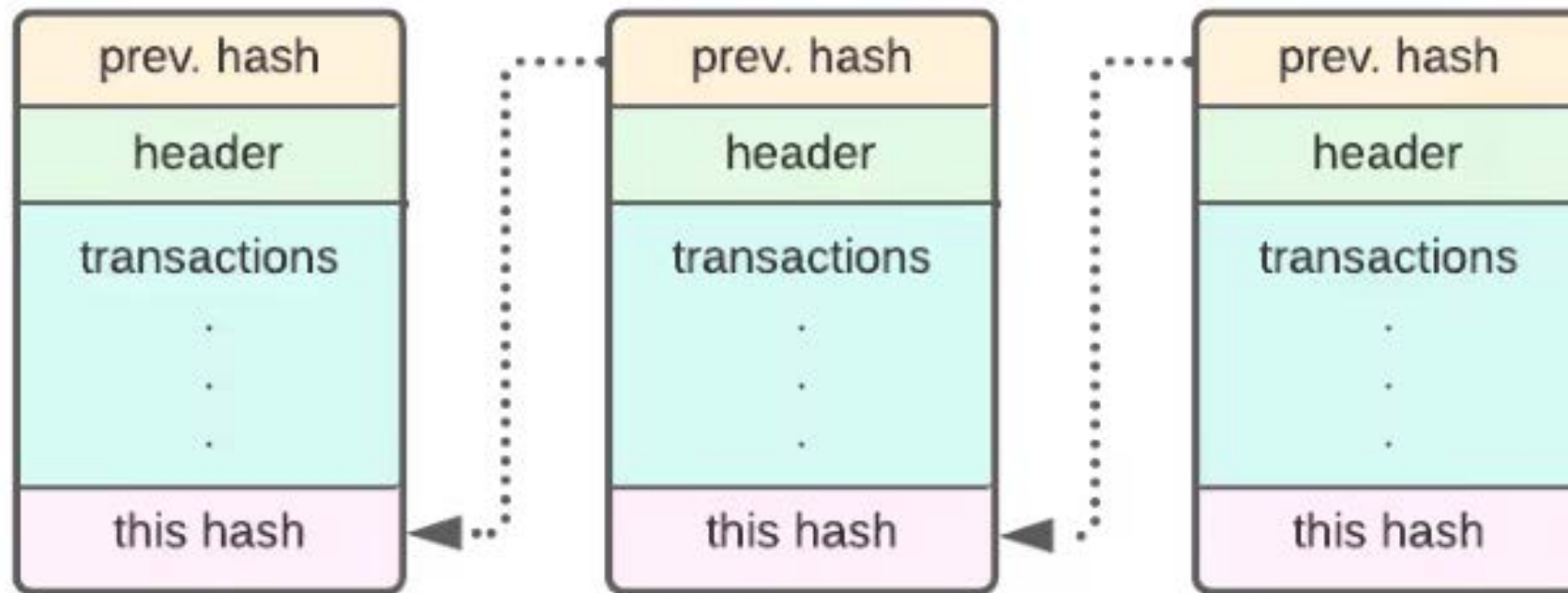


Introduction au développement des contrats intelligents sur Ethereum

Blockchain (l'essential)

Blockchain

- Une chaîne des blocs connectées (hash du bloc précédent)
- Création d'un nouveau bloc tous les 10-20 secondes (PoW dans le passé, PoS maintenant)
- En-tête du bloc : Timestamp, numéro du bloc, hash du bloc précédent, total du gaz utilisé...
- Chaque bloc à une limite de 8 million d'unités de gaz



Blockchain - hachage

- Hash : Empreinte digital d'une phrase, fichier texte, document pdf, mp3...
- L'entrée de la fonction de hachage est de longueur arbitraire mais la sortie (digest) est toujours de longueur fixe (64 caractères)
- La même entrée aura toujours la même valeur de hachage
- La même valeur de hachage ne peut pas être attribuée aux différents textes
- La moindre modification du message aboutit à un résultat totalement différent
- Il ne pas possible de générer le contenu original à partir de la valeur de hachage
- La procédure de calcul de la valeur de hachage est très rapide
- Solidity utilise la fonction keccak256 => créer un hash de 32 octet (64 caractère)
- Outil en ligne Keccak256 : https://emn178.github.io/online-tools/keccak_256.html

Exemple :

Hello => 06b3dfaec148fb1bb2b066f10ec285e7c9bf402ab32aa78a5d38e34566810cd2

Hello1 => f78fd070e76f73c4e7282620a7ab5ba58dc4f724d5dfd45d97ec4e01f817ce9d



Blockchain démo

Modèle d'une blockchain simplifié : <https://andersbrownworth.com/blockchain/block>

Blockchain

Block:	# 1
Nonce:	11316
Data:	
Prev:	0000000000000000000000000000000000000000000000000000000000000000
Hash:	000015783b764259d382017d91a36d206d0600e2cbb3567748f46a33fe9297cf
<button>Mine</button>	

Block:	# 2
Nonce:	35230
Data:	
Prev:	000015783b764259d382017d91a36d206d0600e2cbb3567748f46a33fe9297cf
Hash:	000012fa9b916eb9078f8d98a7864e697ae83ed54f5146bd84452cdafd043c19
<button>Mine</button>	

Block:	# 3
Nonce:	12937
Data:	
Prev:	000012fa9b916eb9078f8d98a7864e697ae83ed54f5146bd84452cdafd043c19
Hash:	0000b9015ce2a08b61
<button>Mine</button>	

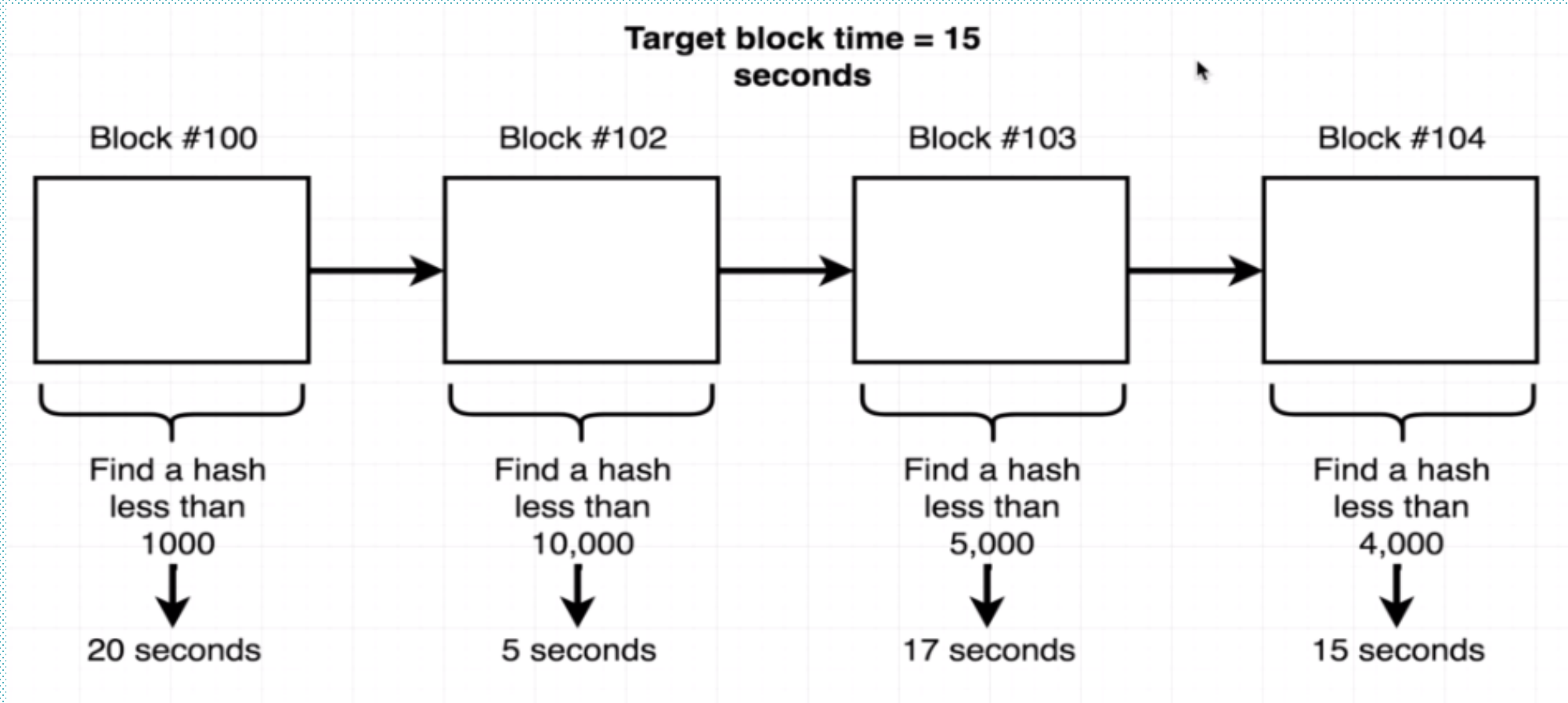
Blockchain – consensus de PoW

L'ancien consensus de preuve de travail :

Data	+	Nonce	=	Output Hash	Output hash as a base 10 number	Is this less than 1000?
'Hi There'		0		a23042b2e	178917215	no
'Hi There'		1		cbc1491	29589283	no
'Hi There'		2		0ca24258	94869869	no
'Hi There'		3		d9eed91	13938166	no
'Hi There'		4		1488baec	419386918	no
'Hi There'		5		0077bbb	100	yes

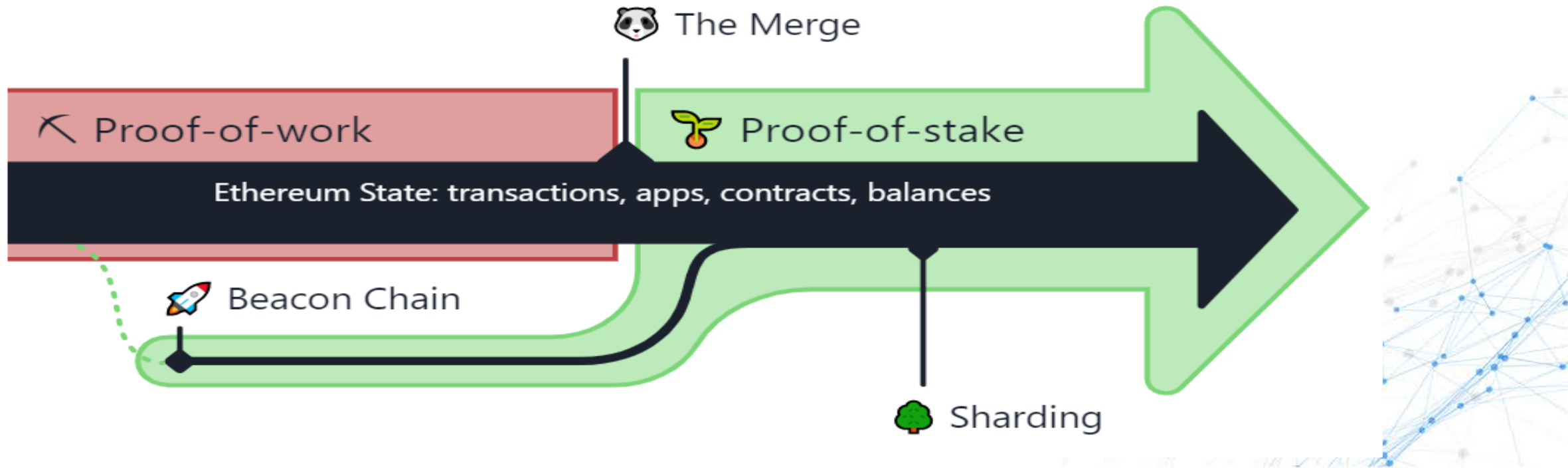
Blockchain – niveau de difficulté

Ajustement du niveau de difficulté (PoW) :



ETH 2.0 – la chaîne Beacon

- Lancé en décembre 2020 => introduit les PoS et prépare la blockchain au "sharding"
- Fusionné avec la chaîne PoW originale en septembre 2022 => remplace PoW par PoS
- Les validateurs PoS ont pris le relais des mineurs PoW => récompenses pour la création et la validation des blocs

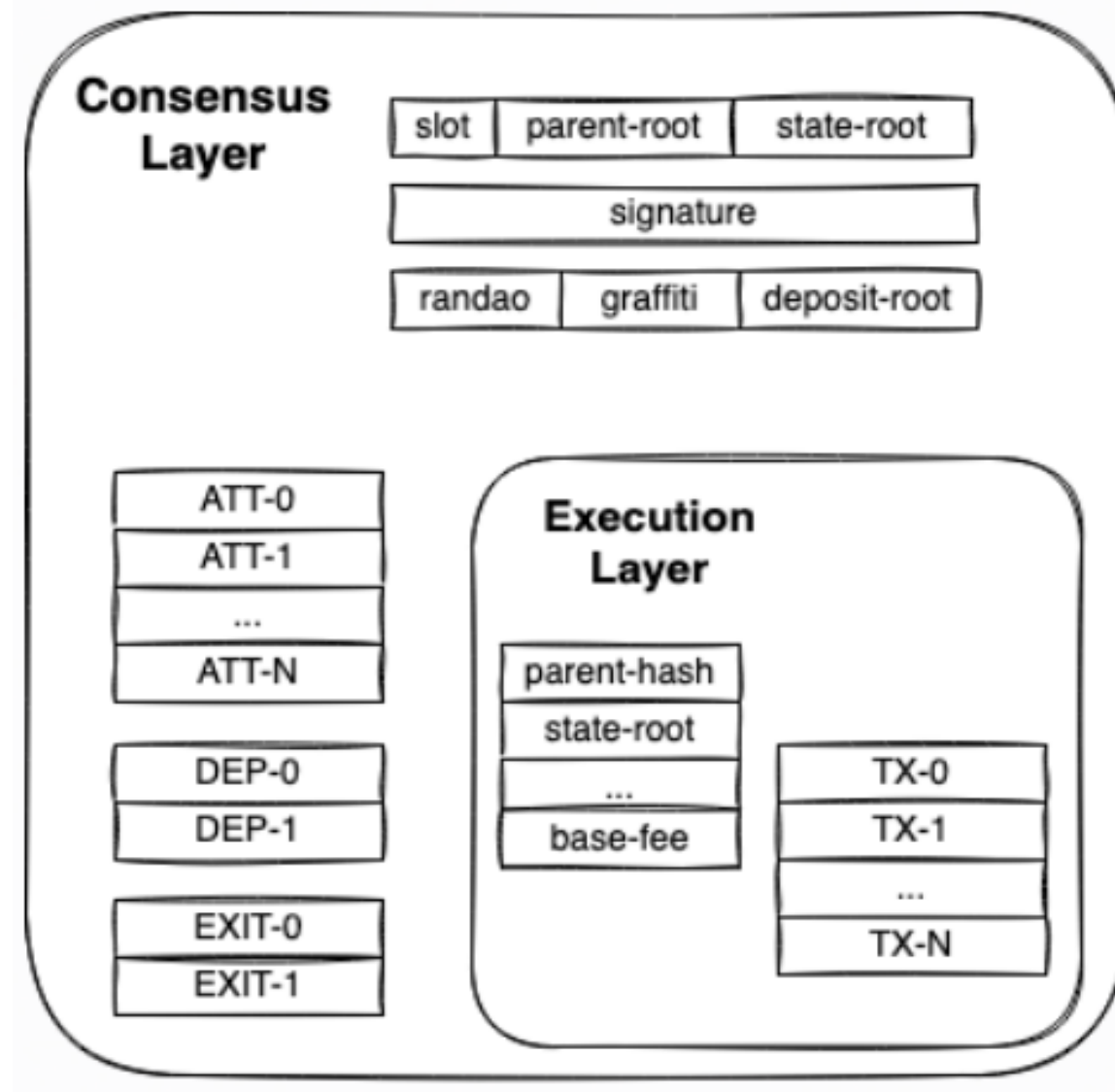


ETH 2.0 – PoS

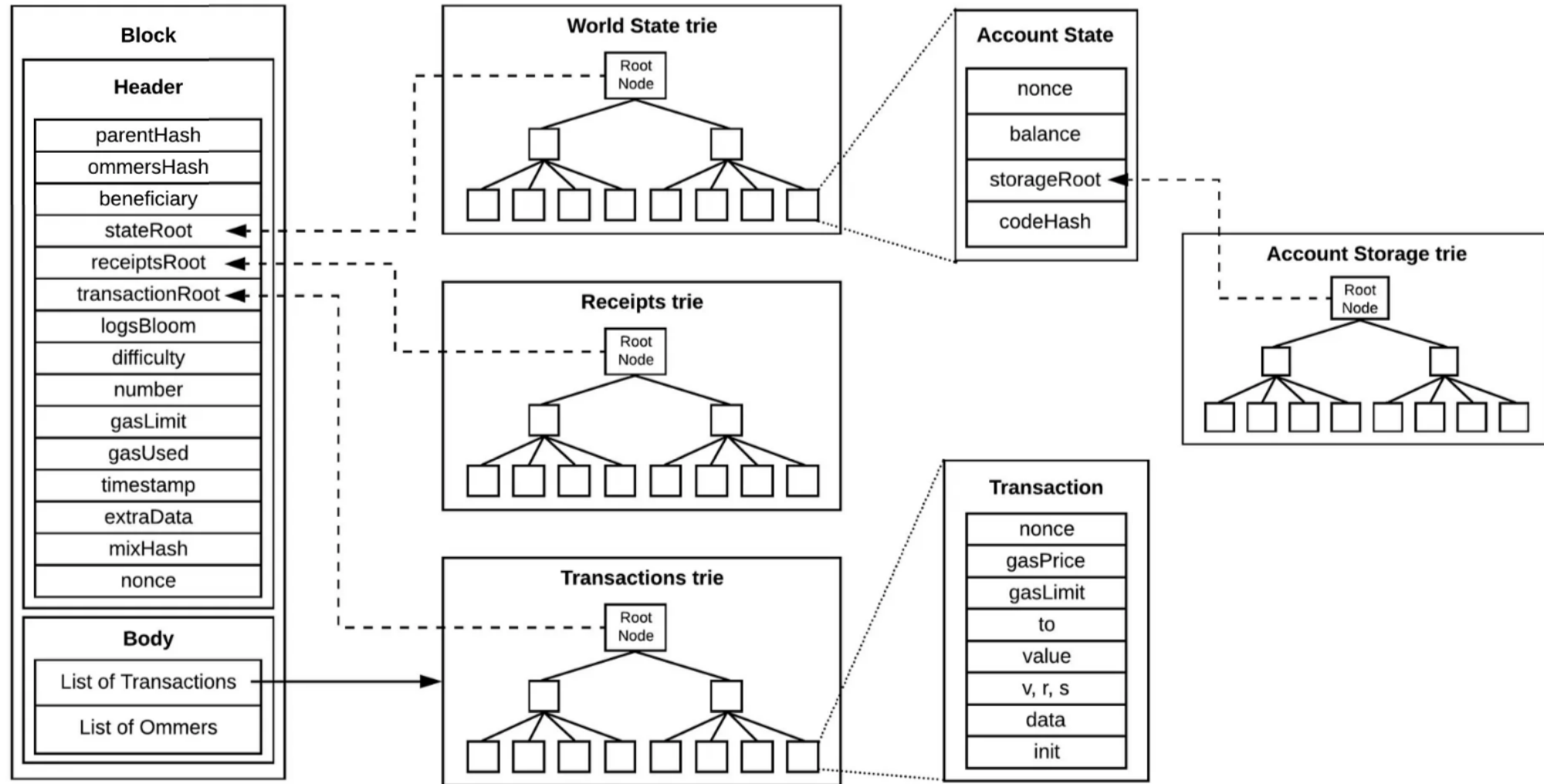
- Les participants au consensus des PoS sont appelés validateurs => dépôt de 32 ETH
- Le temps sur la chaîne Beacon est divisé en "epochs" and "slots". Chaque slot dure 12 secondes (auparavant ~13 s). Une époque est une période de 32 slots (~6,4 minutes)
- Les validateurs doivent valider et générer de nouveaux blocs => obtient une récompense
- Les validateurs sont pénalisés si leurs nœuds sont hors ligne - ils doivent rester en ligne au moins 55 % du temps pour rester rentables
- Les validateurs sont pénalisés (slashed) (3 à 100 % de leur participation) s'ils se comportent mal – en votant pour des blocs invalides...
- Modification des en-têtes de bloc : les champs liés au PoW comme difficulté, nonce... ne sont plus utilisés => mis à 0



ETH 2.0 – après la fusion (The Merge)



Bloc Ethereum



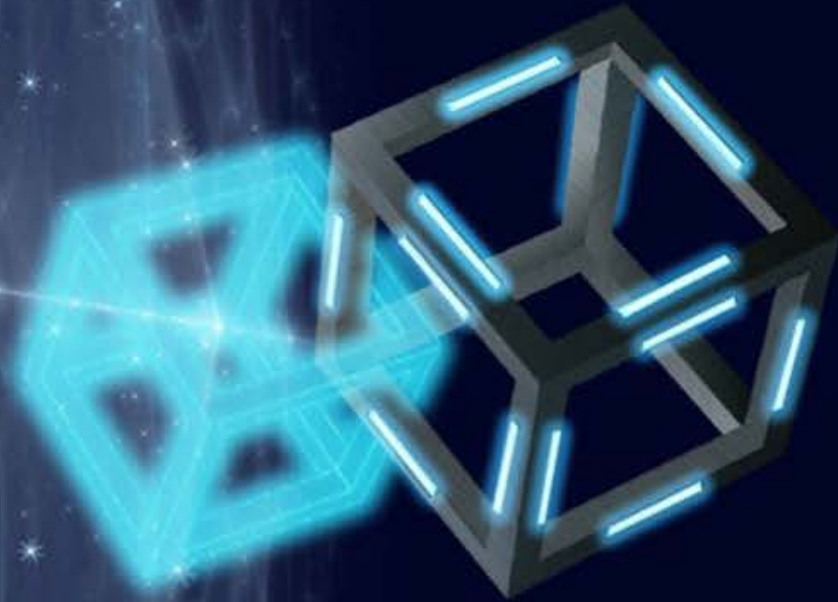
Les champs d'un bloc : <https://ethereum.org/en/developers/docs/blocks/>



ETH 2.0 – après la fusion

- Les clients Eth1 et Beacon deviennent les couches d'exécution et de consensus d'Ethereum
- Les opérateurs de nœuds doivent installer les deux clients (consensus et exécution)
- La couche consensus (Beacon) communiquent avec la couche d'exécution et lui demandent de produire ou de valider des "**ExecutionPayloads**" (équivalent post-fusion de blocs Eth1)
- La couche d'exécution gère la création et la validation des blocs
- Une fois produit/validé, le nœud les partage avec d'autres nœuds du réseau p2p
- La couche consensus transmet des attestations, slashings... la couche d'exécution transmet les transactions, l'état de synchronisation... chacun sur leur réseau p2p indépendant





Introduction au développement des contrats intelligents sur Ethereum

Les comptes sur Ethereum

Blockchain - comptes

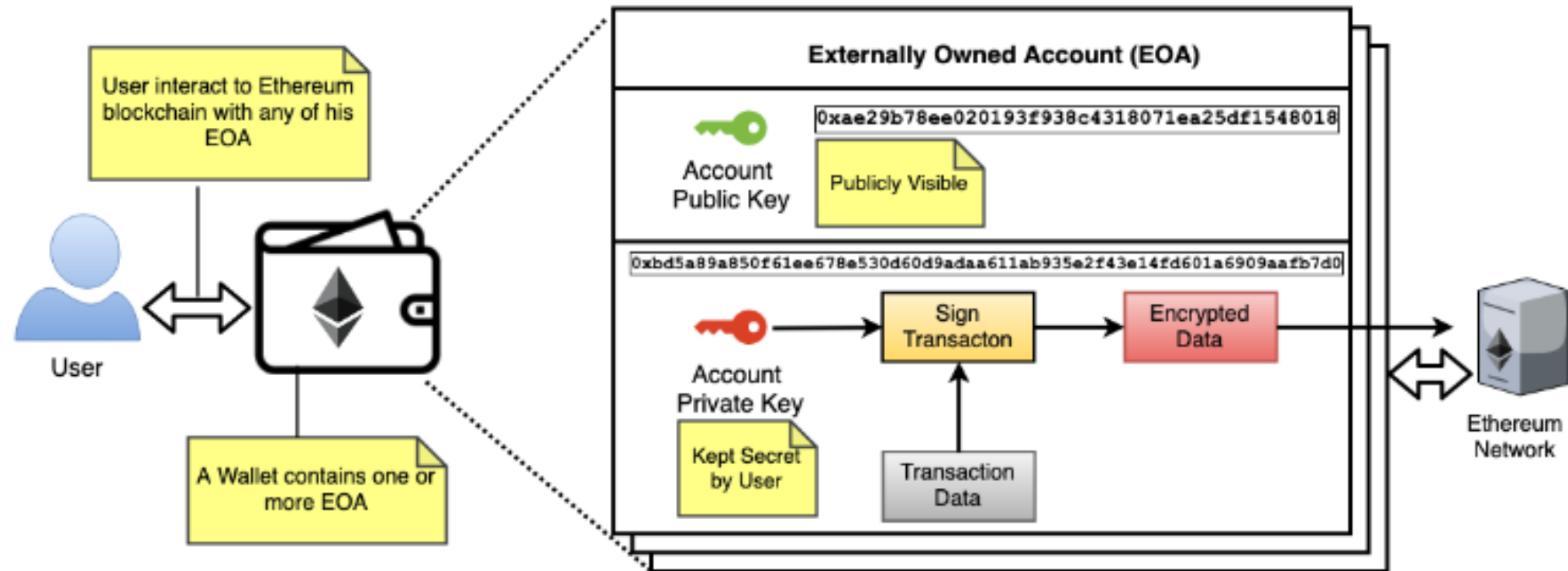
- 2 types des comptes : Comptes externes (EOA) et comptes de contrats
- Les deux types peuvent avoir un solde en ETH
- Adresse d'un compte : 20 octets (40 caractères): 0x3fF34b4000308E581DC5b3CF009ad42b04479bAC

Comptes externes :

- Clé privée (détenu par le portefeuille) => clé public => adresse du compte
- La clé privée ne peut pas être générée à partir de l'adresse du compte (fonction univoque)
- La clé privée est nécessaire pour signer des transactions
- Quelqu'un qui a accès à votre clé privée peut accéder à vos fonds
- On peut récupérer la clé privée avec une phrase secrète (12 ou 24 mots obtenu lors de la configuration de votre portefeuille crypto)
- L'adresse du compte est public, peut être partager avec d'autres personnes, peut être utilisé sur tous les réseaux Ethereum (et EVM compatible)



Blockchain – compte externe



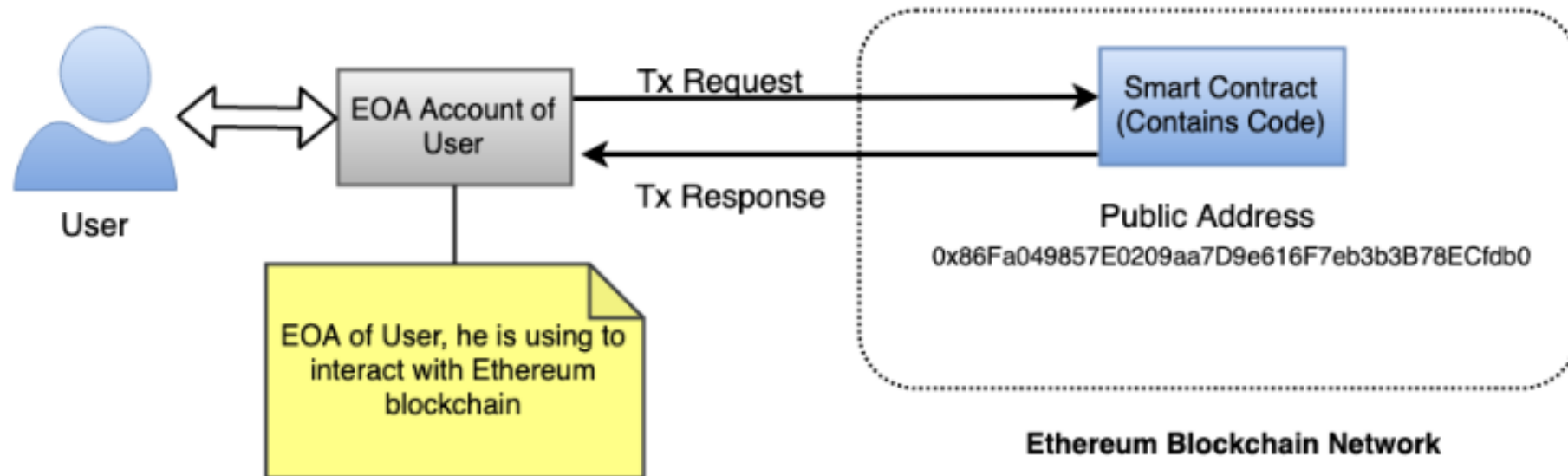
Blockchain – compte de contrat

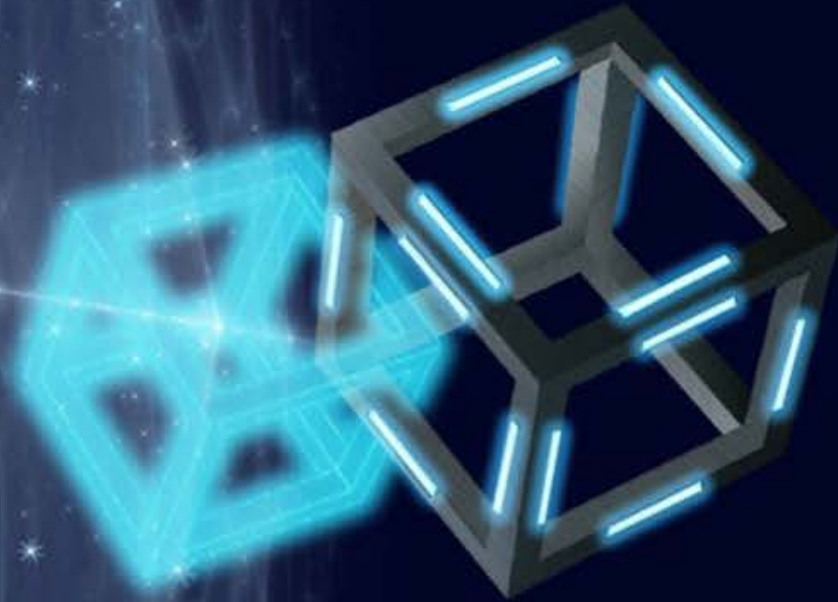
Compte de contrat :

- Création d'un compte de contrat lors du déploiement d'un contrat intelligente
- Pour créer un compte de contrat, il faut exécuter un txn depuis un compte externe => aucune information dans le champ récipient & et le bytecode du contrat dans le champ data
- L'adresse du nouveau contrat est basé sur l'adresse du compte qui créer le contrat
- Le bytecode du contrat est stocké dans le bloc avec la transaction
- Un compte de contrats n'a pas une clé privée
- Un compte de contrat ne peut pas exécuter un transaction en soi => l'exécution d'une transaction doit être déclenché par un compte externe



Blockchain – compte de contrat





Introduction au développement des contrats intelligents sur Ethereum

Transactions & signatures

Blockchain - transactions

- Requis pour changer des données sur la blockchain => envoyer de l'ETH ou modifier l'état d'un contrat
- Seulement un compte externe peut exécuter une transaction
- Pour exécuter une transaction il faut fournir les données nécessaires (to, value, data...) => signer la transaction avec la clé privée => envoyer la transaction sur le réseau Ethereum => la validité de la transaction est évalué => la transaction est inclus dans un nouveau bloc

Les données d'une transaction :

- **From:** Le compte externe qui envoie la transaction (l'expéditeur)
- **To:** L'adresse d'un autre compte externe (pour envoyer de l'ETH) ou d'un contrat (pour envoyer de l'ETH ou appeler une fonction du contrat)
- **Value:** Le montant de l'ETH à envoyer



Blockchain - transactions

Les données d'une transaction :

- **Gas limit** : La quantité maximale de gaz que vous êtes prêt à dépenser la transaction
- **Max fee and max priority fee** : Les frais que vous êtes prêt à payer pour la transaction (en gwei)
- **Nonce** : Compteur qui augmente chaque fois quand on exécute une nouvelle transaction
- **Data** : Les données (en format hexadécimal) qu'on veut envoyer avec la transaction. Nécessaire pour appeler une fonction d'un contrat. Vide pour un simple transfert d'ETH.

Un hash est crée pour chaque transaction envoyé. Le hash peut être utilisé pour vérifié le statut de la transaction et pour récupérer d'autres informations sur la transaction : <https://etherscan.io>



Blockchain – exemple d'une transaction

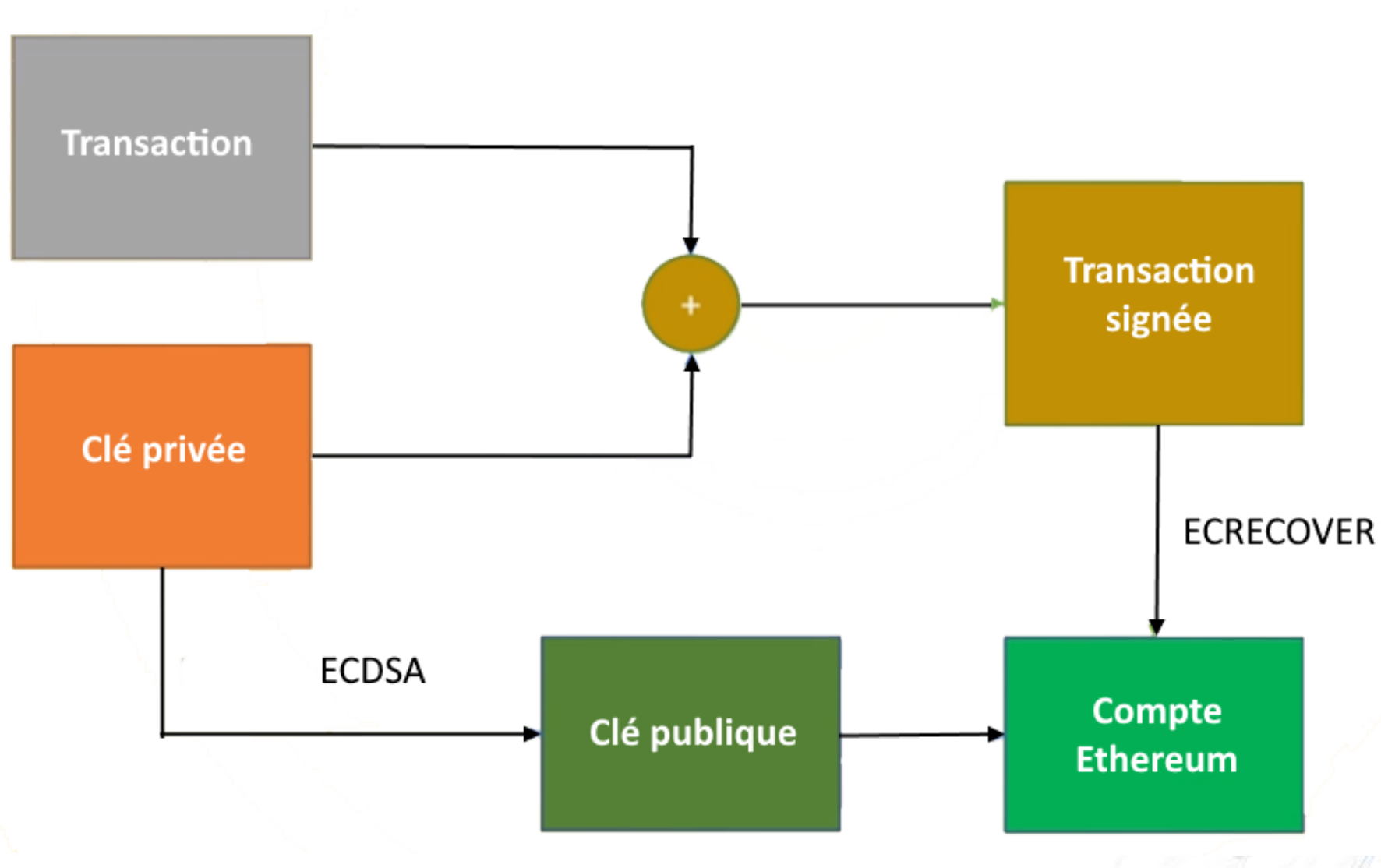
[illegible]

Les champs v, r, et s représente des information cryptographiques qui sont générées par la clé privée. Ces informations peuvent être utilisé pour recréer l'adresse du compte de l'expéditeur et donc vérifier la validité de la transaction.










<https://web3js.readthedocs.io/en/v1.10.0/web3-eth.html#id99>



Blockchain – vérification d'une transaction

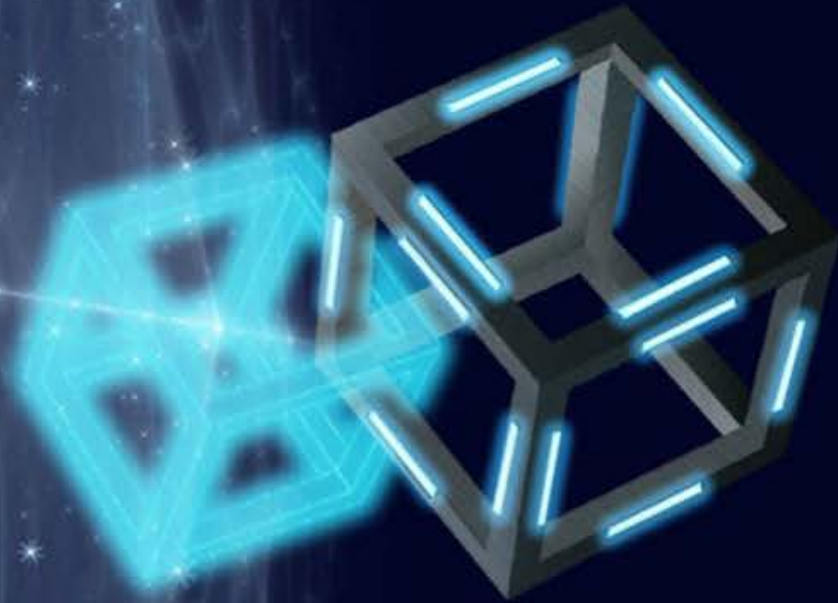


Blockchain – transaction sur Etherscan

Overview	Logs (1)	State	Comments
Transaction Hash:	0x2ae6617a2abb6d8082d859cb3d8c4eef85ce81f91d373bdb63548fb64a0915e2 		
Status:	 Success		
Block:	15016946  1852 Block Confirmations		
Timestamp:	 8 hrs 14 mins ago (Jun-24-2022 06:29:59 AM +UTC)  Confirmed within 30 secs		
From:	0x57ebd61b13b326daf6c46cd069493208a017d72b 		
Interacted With (To):	Contract 0xb8c77482e45f1f44de1745f52c74426c631bdd52 (Binance: BNB Token)  		
Tokens Transferred:	► From 0x57ebd61b13b32... To Binance U... For 0.629784 (\$278.91)  BNB (BNB)		
Value:	0 Ether (\$0.00)		
Transaction Fee:	0.000845753095103994 Ether (\$1.01)		
Gas Price:	0.000000023982110109 Ether (23.982110109 Gwei)		

<https://etherscan.io/tx/0xa85520109eb43c31a0ea3df7b453ae5b859d22e02605db8a082433cf7a394503>





Introduction au développement des contrats intelligents sur Ethereum

Contrats intelligents / Solidity

Blockchain – contrats intelligents & Solidity

- Logiciel tournant sur la blockchain Ethereum
- Peut être considéré comme une machine à états
- Pour changer l'état d'un contrat il faut envoyer une transaction signée au réseau Ethereum
- La plupart des contrats intelligents sont écrits en Solidity
- Solidity est un langage qui est fortement typées, Turing-complet et qui à des similarités avec JavaScript
- Le compilateur Solidity génère le bytecode et le ABI (Application Binary Interface) du contrat
- Le bytecode est déployé sur la blockchain et stocké dans le compte du contrat (Etherscan)
- Le ABI contient la spécification de tous les fonctions publique (avec types & valeurs de retour) et est requis pour accéder aux fonctions du contrat depuis une application client



Blockchain – contrats intelligents & Solidity

Transfers

Holders

Info

DEX Trades

Contract

Analytics

Comments

Code

Read Contract

Write Contract

✔ Contract Source Code Verified (Exact Match)

Contract Name:


TetherToken

Optimization Enabled:

Compiler Version

v0.4.18+commit.9cf6e910

Other Settings:

 Contract Source Code (Solidity)

Audit Report

```
76 ▾ /**
77  * @title ERC20Basic
78  * @dev Simpler version of ERC20 interface
79  * @dev see https://github.com/ethereum/EIPs/issues/20
80  */
81 ▾ contract ERC20Basic {
82     uint public _totalSupply;
83     function totalSupply() public constant returns (uint);
84     function balanceOf(address who) public constant returns (uint);
85     function transfer(address to, uint value) public;
86     event Transfer(address indexed from, address indexed to, uint value);
87 }
88
```

<https://etherscan.io/token/0xdac17f958d2ee523a2206206994597c13d831ec7#code>



Blockchain – types des fonctions

Fonctions qui renvoient des informations :

- Un simple appel de message suffit – aucune transaction requise
- Aucune donnée n'est modifiée sur la blockchain
- La fonction renvoie des données
- La fonction est exécutée immédiatement
- L'exécution de la fonction ne coûte rien

Fonctions qui modifient l'état du contrat :

- Une transaction est requise pour exécuter la fonction
- Modification des données sur la blockchain
- Uniquement le hash de la transaction est renvoyé
- Les modifications seront visibles uniquement après l'intégration de la transaction dans un nouveau bloc
- Il y a des frais à payer pour l'exécution de la transaction





Configuration Metamask

Configuration Metamask

- Télécharger et installer le plugin: <https://metamask.io/download/>
- Création d'un portefeuille – noter la phrase secret
- Changer le nom du compte par défaut
- Ajouter un autre compte
- Paramètres avancées - afficher les réseaux test, customiser la nonce, contrôles de gaz avancés
- Ajouter un réseau local (Hardhat ou Ganache)
- Configuration d'autres réseaux : <https://chainlist.org/>
- Exporter la clé privée du portefeuille
- Ajouter d'autres tokens ERC20 au portefeuille : <https://etherscan.io/tokens>
- Obtenir du ETH Sepolia d'un « faucet »: <https://faucetlink.to/sepolia> ; <https://sepolia-faucet.pk910.de/>



Configuration Metamask

Ajouter un autre réseau :

[Ajouter un réseau](#)

Nom du réseau

Nouvelle URL de RPC

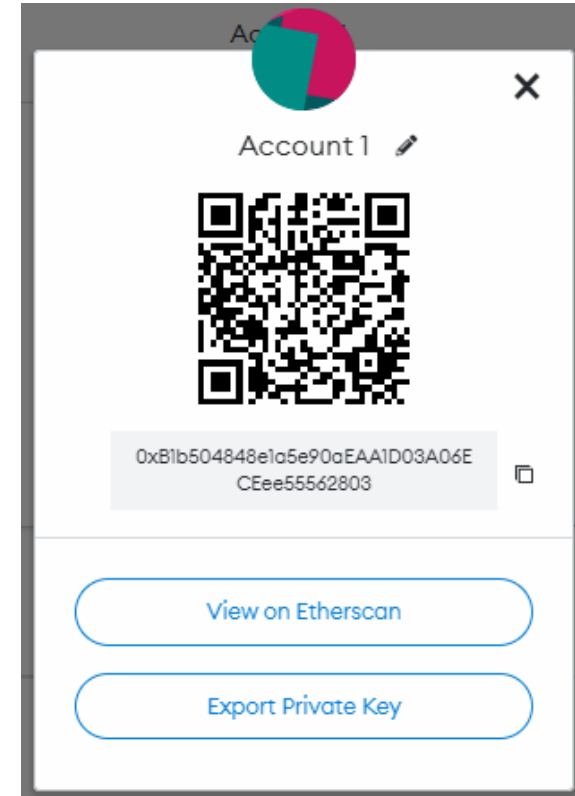
ID de chaîne ⓘ

Symbole de la devise

URL de l'explorateur de blocs (Facultatif)

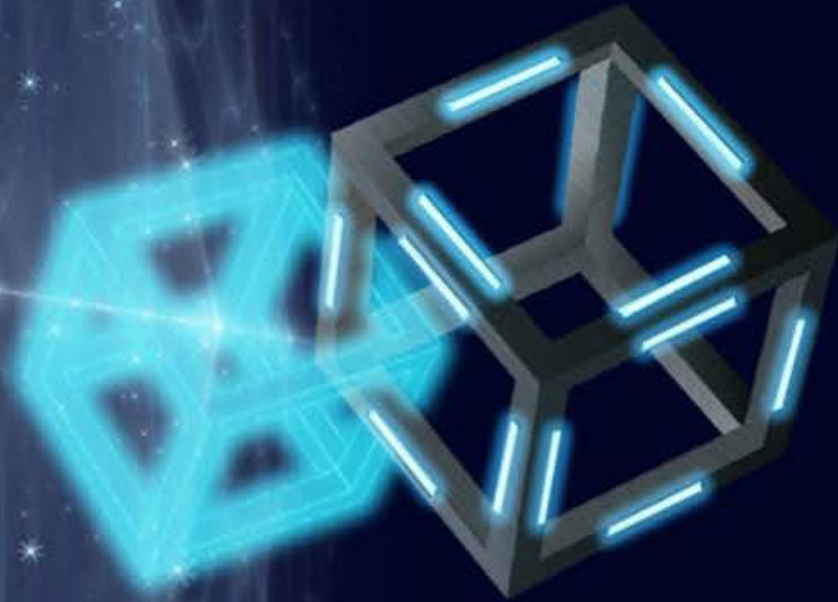
Paramètres avancés => Réseaux

Exporter la clé privée :



Détails du compte => Exporter la clé privée





Compilation, déploiement et débogage des contrats intelligents sur Remix

Remix

Remix IDE: <https://remix.ethereum.org/>

Pages principales :

- Explorateur : Plusieurs exemples des projets, importer/exporter un espace de travail, ABI & bytecode pour les contrats dans le dossier « artifacts »
- Compilateur : Sélectionner la version de compilateur souhaiter et cocher “Auto compile”, copier l’ABI et le bytecode
- Déploiement des contrats & exécution des transactions: Différente environnements => Remix VM, Provider injecté (MetaMask), Dev (Hardhat, Ganache)...





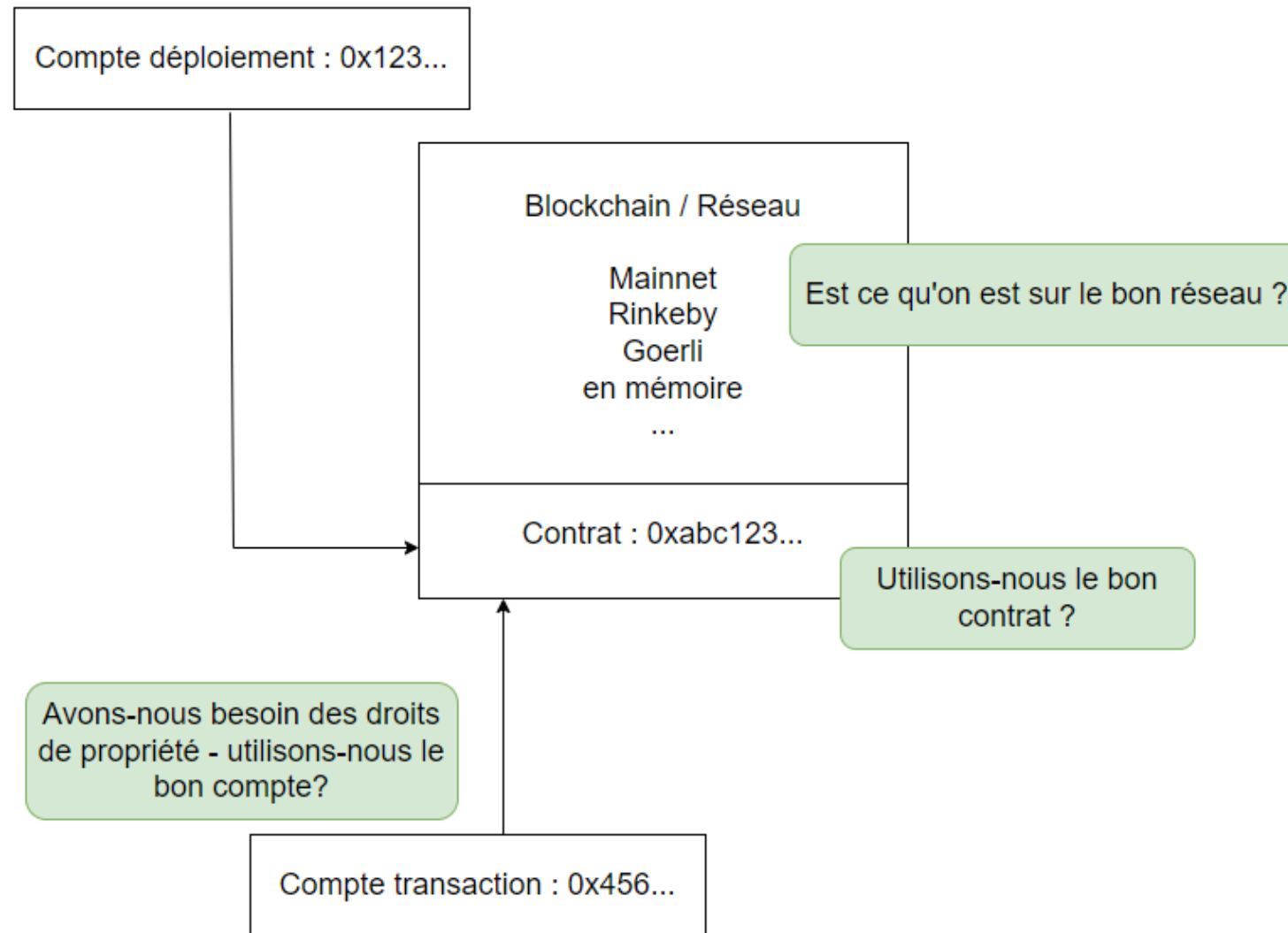
**Un simple contrat
intelligent sur Remix**

Contrat « Hello World » sur Remix

- Créer un simple contrat qui peut lire/écrire un message (string) et émettre un événement quand le message est mis à jour
- Déployer le contrat en utilisant “**Remix VM**”
- Tester les fonctions du contrat
- Déboguer le code : point d’arrêt, variables d’état, exécution du bytecode, détails: numéro du bloc, consommation de gaz...
- Déployer le contrat sur le réseau Sepolia utilisant “**injected provider – Metamask**” sur Remix
- Vérifier le bytecode dans Metamask
- Vérifier la transaction de déploiement sur Etherscan
- Exécuter la fonction “updateMessge” sur Remix et vérifier la transaction sur Etherscan



Remix – déploiement & exécution

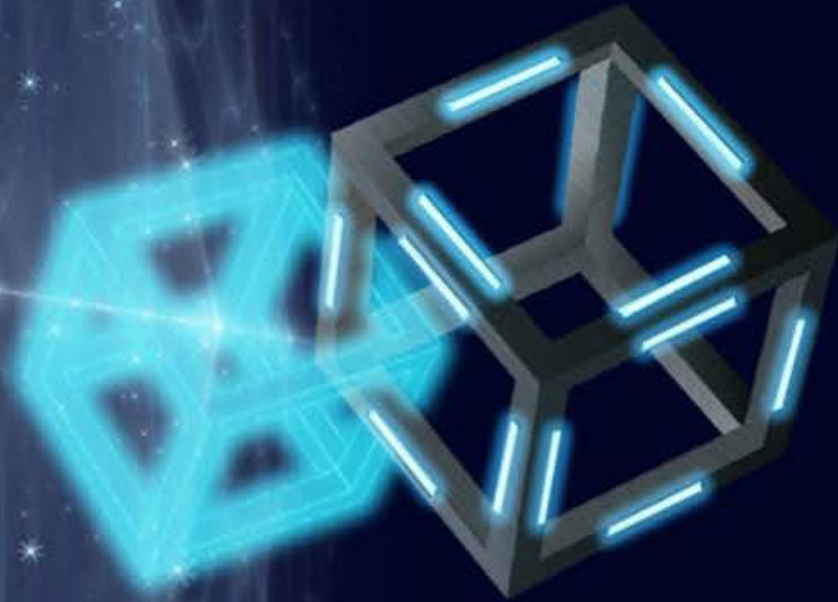


Remix - exercice

Tâches :

- Ajouter des fonctions pour lire/écrire une valeur entière (type : uint)
- Deployer le nouveau contrat en utilisant « Remix VM »
- Tester les nouvelles fonctions
- Utiliser le déboguer : vérifier les variables, les opcodes exécutées, les informations détaillées...





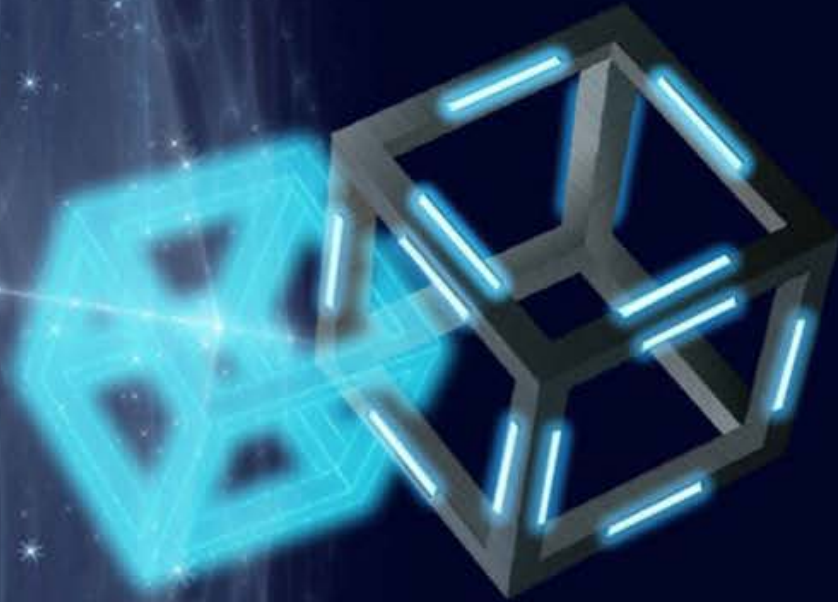
Configuration d'un environnement de développement locale

Environnement de développement local

Installer les applications et plugins suivants :

- Node.js (inclut NPM) - LTS version: <https://nodejs.org>
- Visual Studio Code: <https://code.visualstudio.com/download>
- VS Code plugins:
 - Solidity + Hardhat
 - Live Server
 - Prettier





**Compilation, test et
déploiement des
contrats intelligents
avec Hardhat**

Qu'est-ce que Hardhat ?

- Environnement de développement pour compiler, deployer, tester et déboguer des contrats intelligents sur Ethereum
- Hardhat inclut aussi un réseau de test
- CLI pour interagir avec Hardhat (le « Hardhat Runner »). Par exemple : `npx hardhat compile`
- Pour afficher toutes les commandes disponibles : `npx hardhat (compile, node, run, test, --network)`

Création d'un projet Hardhat :

- Créer un fichier package.json dans le dossier du projet : `npm init`
- Installer Hardhat : `npm i hardhat -D`
- Créer un projet Hardhat : `npx hardhat` => sélectionner : *“Create a basic sample project”*
- Installer le plugin suivant : `npm i -D @nomicfoundation/hardhat-toolbox`



Configuration de Hardhat

- Exemple d'un fichier de configuration : <https://hardhat.org/config>
- Le fichier de configuration (hardhat.config.js) qui se trouve dans la racine du projet est exécuté chaque fois avant le lancement d'une tâche
- Les chemins pour les sources, artefacts, cache et tests peuvent être modifié
- Options supplémentaires pour configurer le réseau : « from, gas, gasPrice, chainId, timeout »

```
module.exports = {  
  solidity: {  
    version: "0.8.9",  
    settings: {  
      optimizer: {  
        enabled: true,  
        runs: 200,  
      },  
    },  
  },  
  defaultNetwork: "localhost",  
  networks: {  
    rinkeby: {  
      url: REACT_APP_ALCHEMY_API_URL_RINKEBY,  
      accounts: [  
        `0x${REACT_APP_PRIVATE_KEY}`,  
      ],  
    },  
  },  
};
```



Clés API pour Alchemy & Etherscan

Alchemy:

- Créez un compte Alchemy : <https://auth.alchemy.com/signup>
- Créez une App sur Alchemy et sélectionnez **Ethereum** pour la chaîne et **Sepolia** pour le réseau
- Cliquez sur "**API Key**" et copiez/collez la valeur sous "**HTTPS**" dans le fichier .env

Etherscan:

- Créez un compte Etherscan : <https://etherscan.io/register>
- Dans la barre de menu de gauche, cliquez sur "**API keys**" et créez une nouvelle clé API
- Copier/coller la clé API dans le fichier .env



Compilation & déploiement des contrats avec Hardhat

Compilation des contrats :

`npx hardhat compile`

Génération du bytecode et ABI
dans le dossier « artifacts »

Déploiement d'un contrat :

`npx hardhat run scripts/deploy.js`

ou:

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

```
async function main() {  
  const helloWorld = await ethers.deployContract("HelloWorld",  
    ["Initial message"]);  
  
  await helloWorld.waitForDeployment()  
  
  console.log("HelloWorld deployed to:", helloWorld.target)  
}  
  
main().catch((error) => {  
  console.error(error)  
  process.exitCode = 1;  
})
```



Le réseau Hardhat

Exécuter « in-process » :

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

Exécuter « stand-alone » => accessible par des clients externes comme Metamask :

- Dans un fenêtre de commande : *`npx hardhat node`*
- Lancement d'un réseau locale : <http://localhost:8545> avec chainId: 31337
- Dans un deuxième fenêtre de commande : *`npx hardhat run scripts/deploy.js --network localhost`*
- Configurer le réseau local dans Metamask



Débogage avec Hardhat

Débogage => écrire des messages dans la console !

```
pragma solidity ^0.6.0;  
  
import "hardhat/console.sol";  
  
contract Token {  
    //...  
}
```

```
function transfer(address to, uint256 amount) external {  
    console.log("Sender balance is %s tokens", balances[msg.sender]);  
    console.log("Trying to send %s tokens to %s", amount, to);  
  
    require(balances[msg.sender] >= amount, "Not enough tokens");  
  
    balances[msg.sender] -= amount;  
    balances[to] += amount;  
}
```



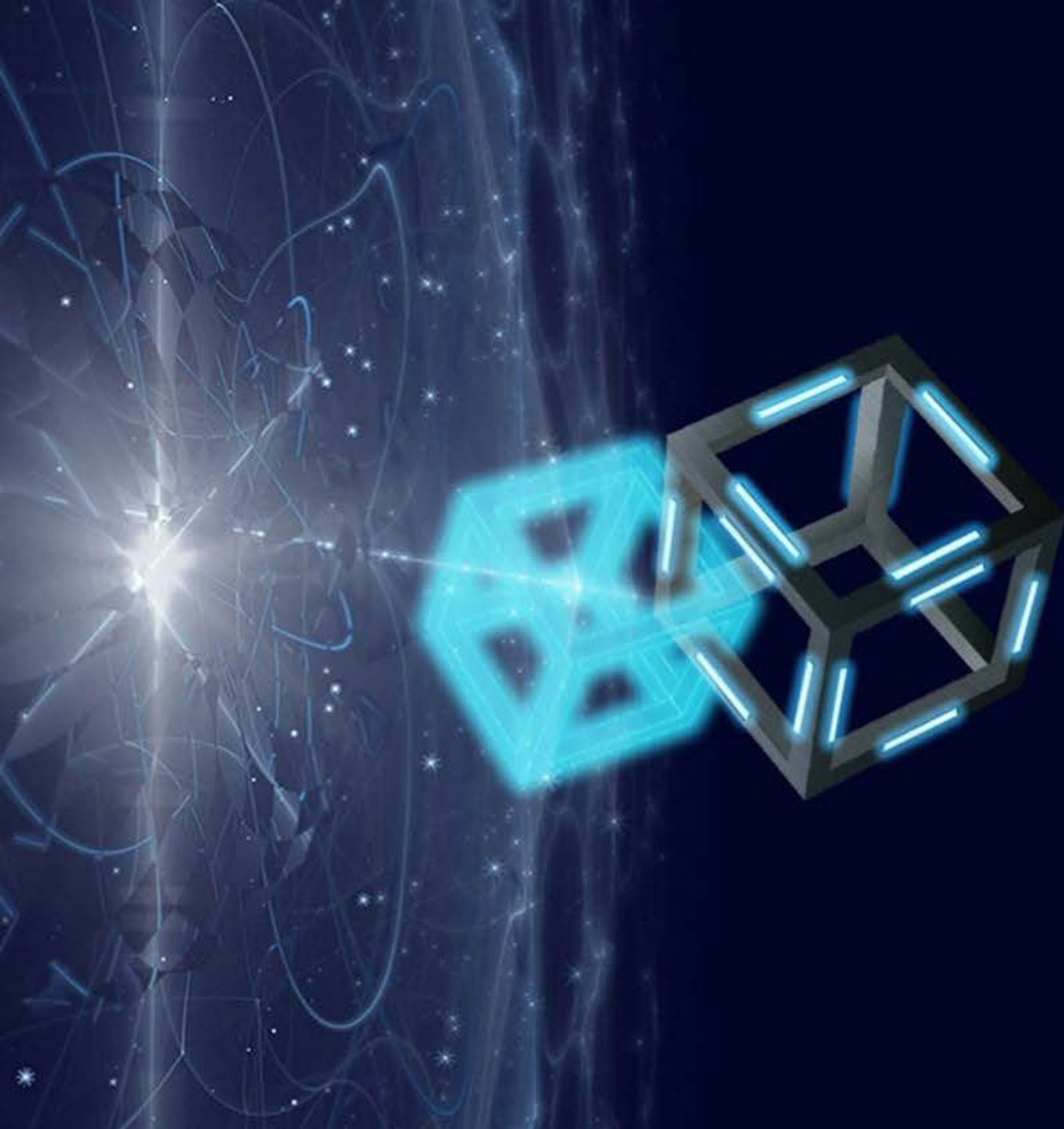
Sepolia Test Ether

- Sepolia est un réseau de test et les frais doivent être payés avec Sepolia Ether
- Peut être obtenu gratuitement sur divers robinets ("faucets")

Robinets pour obtenir Sepolia ETH:

- <https://sepolia-faucet.pk910.de/>
- <https://sepoliafaucet.com/>
- <https://faucetlink.to/sepolia>





Projet « Hello World » avec Hardhat

Besoins de projet

- Créer un projet Hardhat avec toutes les paquets npm requis
- Ajouter le réseau Sepolia (testnet) dans [*hardhat.config.js*](#)
- Ajouter le contrat « Hello World » au projet
- Deployer le contrat sur un réseau local et sur Sepolia
- Ecrire un script qui peut lire et écrire le message du contrat
- Ajouter le code du contrat sur l'explorateur Sepolia : [*sepolia.etherscan.io*](#)
`npx hardhat verify --network sepolia CONTR_ADDR "constructor argument"`



Exercice

- Ajouter une fonction au contrat qui peut lire/écrire un entier (uint)
- Emettre un événement quand l'entier est modifié
- Deployer le contrat sur le réseau local
- Ajouter une fonction au script qui peut lire/écrire l'entier du contrat
- Deployer le contrat sur Sepolia
- Exécuter le script (scripts/interact.js) sur le contrat qui est déployé sur Sepolia
- Vérifier le transactions sur : <https://sepolia.etherscan.io>
- Ajouter le contrat modifié sur : <https://sepolia.etherscan.io>

