

Formation technique

Smart contracts et token : développement et
déploiement

Sommaire

1. Ce que l'on va faire
2. Comment le faire

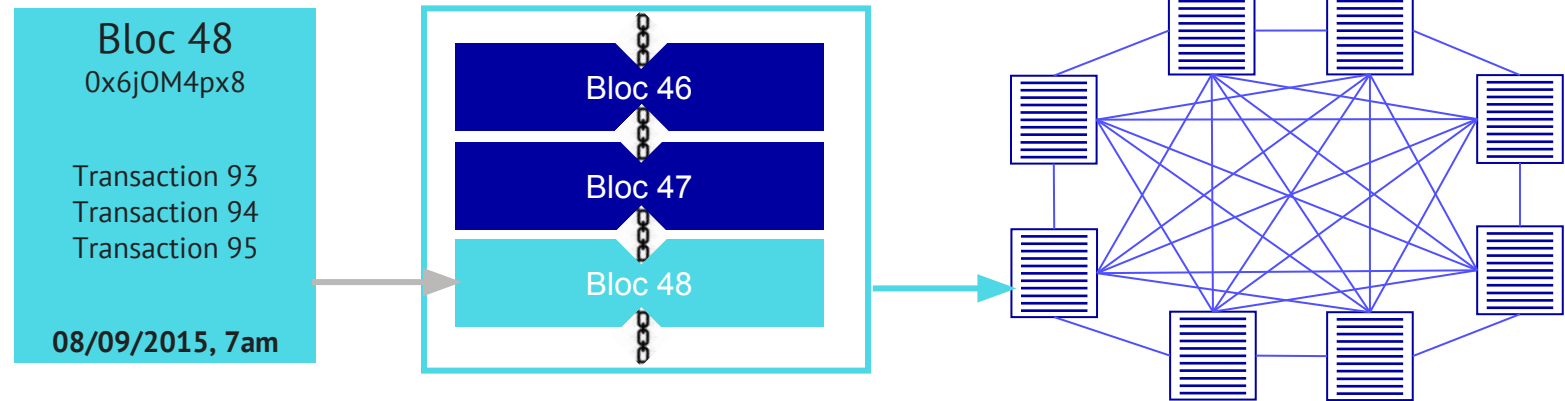
Ce que l'on va faire

Objectif

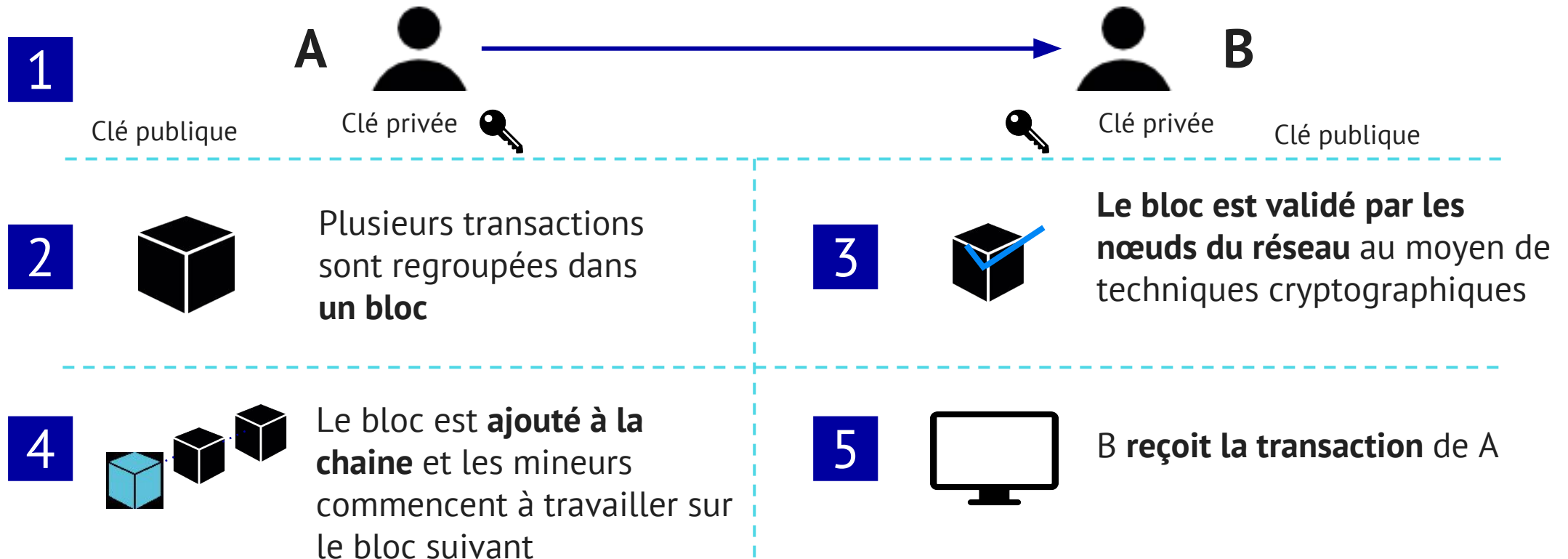
- Se familiariser avec l'environnement de développement et comprendre la blockchain (et un peu plus précisément Ethereum)
- Créer son premier projet Ethereum et son premier smart contract : un token échangeable sur la blockchain
- Le déployer et interagir avec
- Comprendre les mécanismes de distribution et effectuer une ICO

Définition

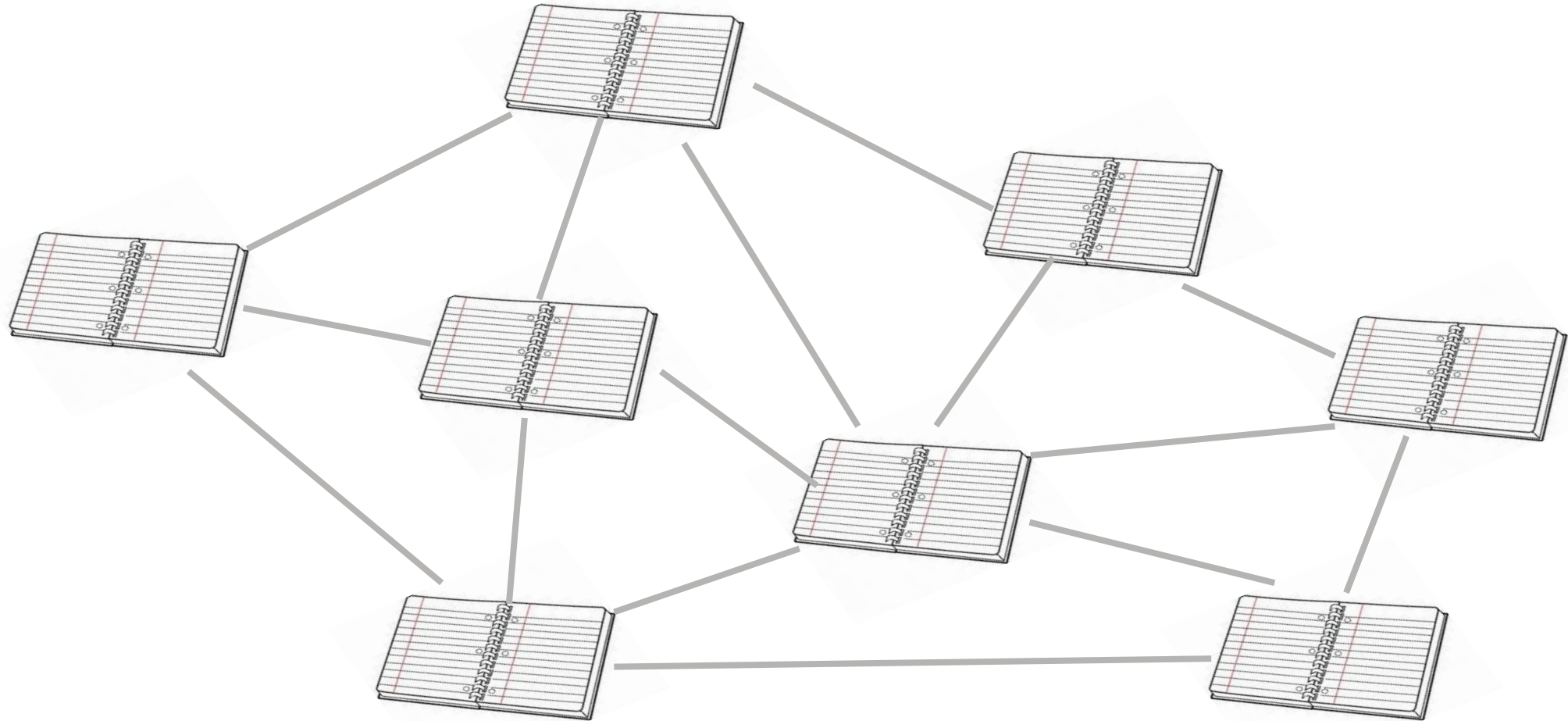
- Une **base de données** structurée en blocs regroupant des faits (exemple : transactions dans une cryptomonnaie)
- **Immuable : augmente constamment** avec l'ajout de nouveaux blocs
- Chaque bloc est **horodaté** lors de sa validation
- Copie des données **distribuée sur l'ensemble des nœuds** (ordinateurs) du réseau
- Fonctionne en **autonomie sans organe central de contrôle**



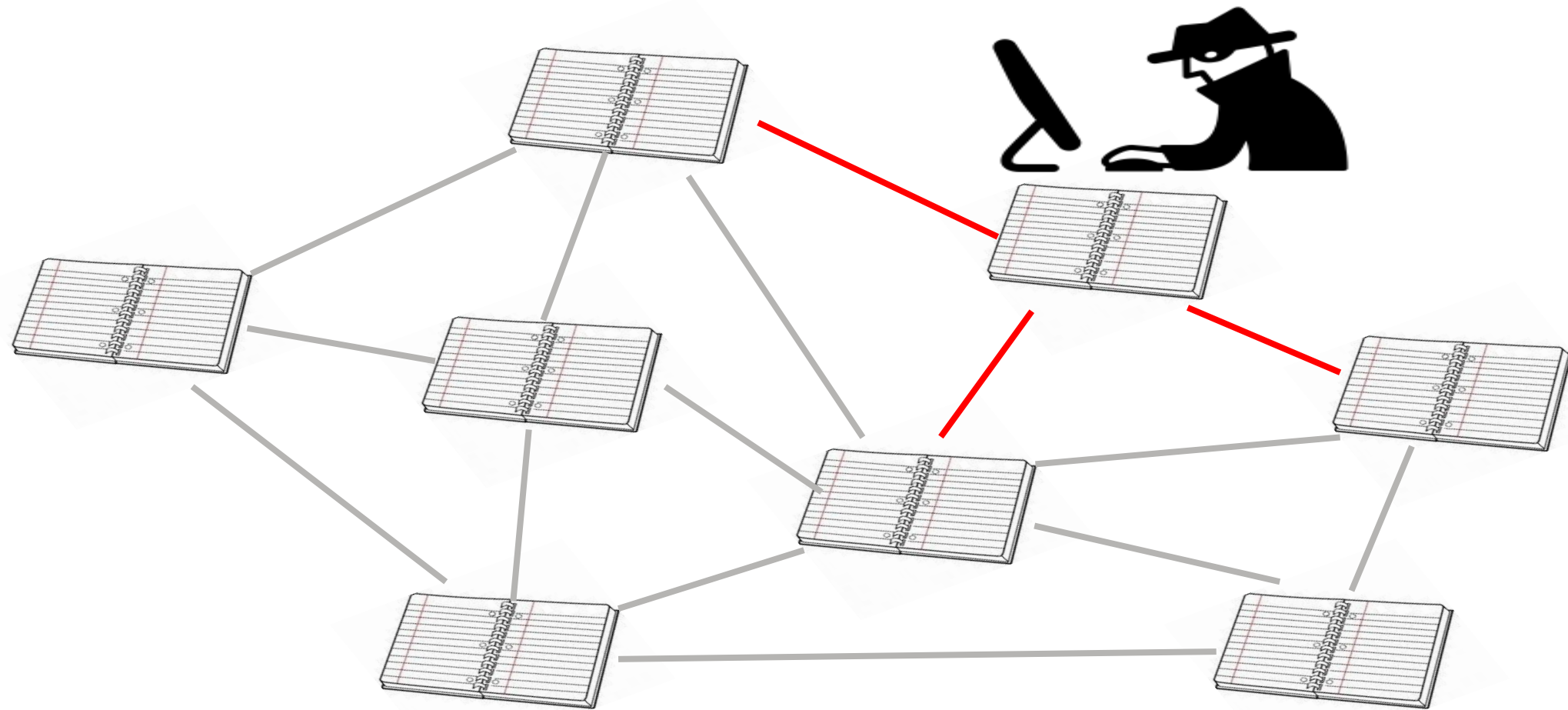
Déroulé d'une transaction



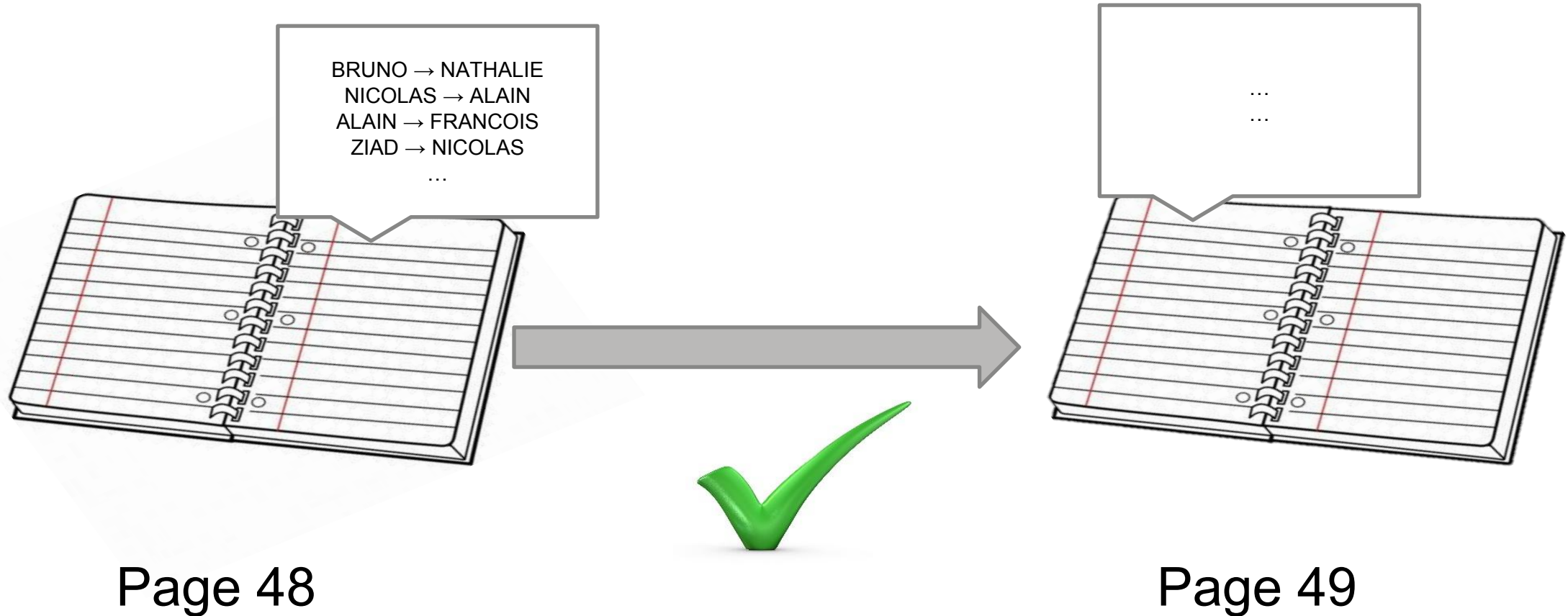
Sous le capot



Sous le capot



Sous le capot



Ethereum

- La blockchain la plus avancée pour coder des smart contracts (donc des tokens)
- 2e blockchain en terme de capitalisation
- Une gouvernance équilibrée et ouverte
- L'environnement le plus prometteur pour développer des dApps (applications décentralisées)



ethereum

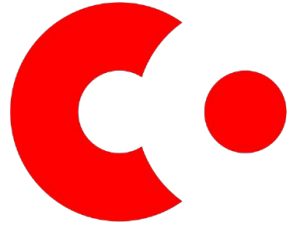
Ethereum et ses concurrents

Hyperledger Fabric



- Poussée par IBM principalement
- Des contrats en langages répandus (Go, JavaScript, Java)
- Axée blockchain privée, légèrement plus centralisée qu'Ethereum (Kafka)
- Pas de tokens

R3 Corda



- DLT (Distributed Ledger Technology) axée sur la confidentialité
- Chaque noeud contient la partie du ledger qui le concerne
- Java/Kotlin
- Applications financières
- Pas de tokens

Blockchain publique vs. blockchain privée

Blockchain publique

- Décentralisée (10 000+ noeuds)
- Monnaie avec une valeur réelle
- Utilisée par le monde entier
- On considère une dApp en production si elle est sur la blockchain publique
- Quasiment inaltérable (The DAO)

Blockchain privée

- Centralisée (environ 10 noeuds)
- Monnaie sans valeur
- Environnement contrôlé (on peut le redémarrer/changer facilement)
- Environnement de test ou destiné à des entreprises (supply chain, logistique)

Mais qu'est-ce qu'un noeud ?

- Une porte d'accès à la blockchain
 - › On est obligés de passer par un noeud pour interagir avec une blockchain : obtenir des infos ou en soumettre
- Un noeud contient une copie (plus ou moins complète) de la blockchain
- Tout le monde peut en posséder un
 - › Plus il y a de noeuds, plus la blockchain est décentralisée
- Les noeuds ont souvent un pouvoir de décision et une incentive à tourner
 - › Vote sur des décisions majeures sur la blockchain
 - › Les noeuds mineurs (validateurs de blocs) sont récompensés

Développer un smart contract sur Ethereum ?

- Quel langage ? ⇒ **Solidity** : un mélange de C et de JavaScript, développé spécialement pour Ethereum
- Quel framework ? ⇒ **Truffle** : le framework de référence (open-source) pour commencer et coder des smart contracts
- Quel réseau ? ⇒ **Testnet Ethereum (local ou online)** : on souhaite pouvoir tester le contrat avec des sandboxes avant de déployer sur la blockchain publique ; **déployer un contrat coûte de l'ether**

Pour commencer avec Truffle


- Installer node/npm : <https://nodejs.org/en/download/>
- Installer Truffle : <http://truffleframework.com/>
- Créer un nouveau répertoire, ouvrir un terminal, lancer “truffle init” dans le nouveau répertoire :

```
[MacBook-Pro-de-Maxime:~ mhg$ mkdir first-contract  
[MacBook-Pro-de-Maxime:~ mhg$ cd first-contract/  
[MacBook-Pro-de-Maxime:first-contract mhg$ truffle init  
Downloading...  
Unpacking...  
Setting up...  
Unbox successful. Sweet!
```

Commands:

```
Compile:      truffle compile  
Migrate:      truffle migrate  
Test contracts: truffle test
```


Pour commencer avec Remix

- Rendez-vous sur <http://remix.ethereum.org/>
- En haut à gauche, cliquer sur  et créer un nouveau contrat : le nom du fichier doit être le même que le nom du contrat.
- Nous sommes prêts à coder !

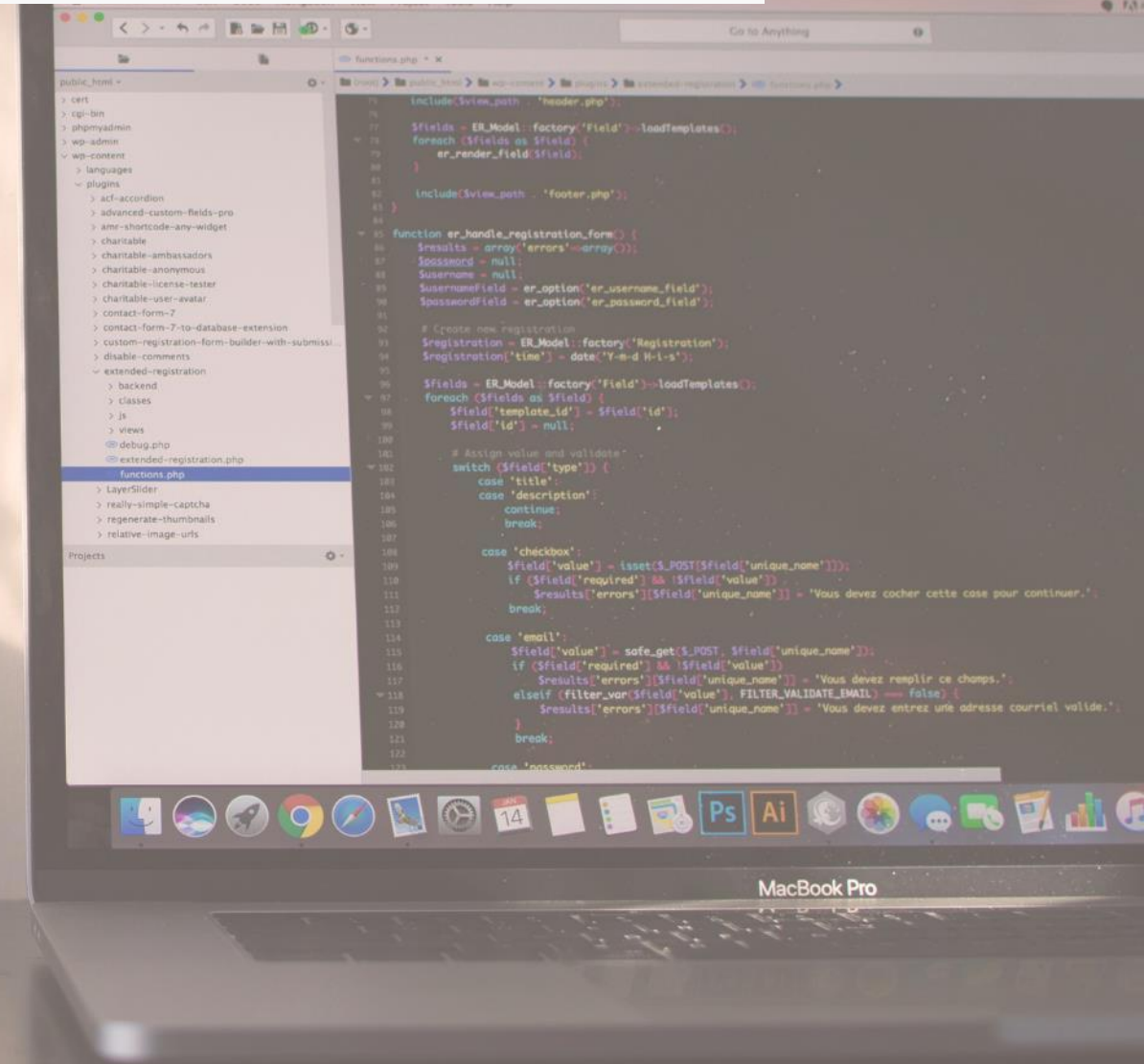
A propos de Solidity

- Langage fortement typé :
 - › **uint** : tous les entiers non-signés
 - › **string** : les chaînes de caractères
 - › **bytes32** : tableau de bytes à taille fixe (ici 32, le max)
 - › **bool** : true ou false
 - › **mapping (clé => valeur)** : structure de stockage de données
 - › **uint[], bytes32[]...** : tableau d'entiers, de bytes32...
 - › Possibilité de créer des struct, comme en C.
- Plus d'infos : <http://solidity.readthedocs.io/en/develop/types.html>

Recoder la roue ?

- Des standards (audités des dizaines de fois) sont disponibles, ils sont utilisés dans la majeure partie des contrats
- On va les utiliser pour créer notre token !
- Librairie Open-Zeppelin disponible ici : <https://github.com/OpenZeppelin/zeppelin-solidity/tree/master/contracts> ; copiez le code du fichier qui vous a été transmis et collez le dans Remix

Création du token



On a tout pour commencer !

- Etudions le fichier MintableToken.sol

<https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/MintableToken.sol>

Oui, mais moi je veux MON token

- Très bien, créons un nouveau contrat dans notre dossier contrats pour créer un token original ! NomToken.sol est un bon nom.
- On veut importer MintableToken
- On veut lui donner trois caractéristiques :
 - › string public constant name
 - › string public constant symbol (l'abbréviation)
 - › uint8 public constant decimals (la norme est 18, jusqu'à quelle décimale on peut diviser notre token)

Le nouveau contrat est prêt ?

- Votre nom de contrat apparaît en vert sur la droite
- Oui, mais, on génère comment des tokens, j'en veux moi ?!
 - › Créer un constructeur et allouer des tokens au créateur
- Comment on fait ?
 - › Un indice, le constructeur s'écrit fonction `NomContrat()` {<code>}
 - › Le créateur du contrat est disponible grâce à la variable `msg.sender`
 - › Quelle fonction génère des tokens ? On cherche dans la lib :)
 - › 1 token = 1000000000000000000, pourquoi ?

La réponse

`mint(msg.sender, amount)`

1 token = 1000000000000000000000000 car il y a 18 décimales, donc on peut recevoir jusqu'à 0.000000000000000000000001 token et pas de flottant dans Solidity pour le moment !

Je veux déployer mon token !

- Deux solutions : en local ou sur un réseau de test
 - › Local : <http://truffleframework.com/ganache/> ou Remix JS VM
 - › Testnet : Metamask + Remix <https://metamask.io/> ;
<https://remix.ethereum.org/>
- Essayons le testnet Rinkeby !
- Faucet : <https://faucet.rinkeby.io/> (eh oui, il faut de l'ether !)
 - › Normalement vous en avez déjà tous !

Exemple : le HG Token

```
contract HgToken is MintableToken {  
  
    using SafeMath for uint256;  
    string public constant name = "Hg Token";  
    string public constant symbol = "HG";  
    uint8 public constant decimals = 18;  
  
    function HgToken() public {  
        mint(msg.sender, 10000000000000000000000000000);  
    }  
  
}
```

Je veux créer plus de tokens !

- Appeler la fonction **mint** du contrat en envoyant une transaction !
 - › Grâce à Remix
 - › Grâce à MyEtherWallet (par exemple)
- On peut décider de manière irrévocable que la création de tokens est terminée en appelant la fonction **finishMinting**

Je veux envoyer des tokens à mes parents

- Appeler la fonction **transfer** du contrat
 - › Grâce à Remix
 - › Grâce à MyEtherWallet
 - › Fonction standard commune à tous les tokens ERC20
- Visualiser les balances grâce au contrat ou à etherscan (exemple : <https://rinkeby.etherscan.io/token/0xd06d39eef0e6c4341f41df2b0b4042cd80c28a17>)

Je veux permettre au monde entier d'acheter mon token

- On va réaliser une ICO (Initial Coin Offering)
- On intègre une fonction d'achat au sein de notre token
- Une fois intégrée, on déploie le code, et on donne l'adresse aux potentiels investisseurs !
- Maintenant, dites nous pourquoi nous devrions acheter votre token... Quel est son prix, ses caractéristiques...

Je veux permettre au monde entier d'acheter mon token

- Notre ICO a été un grand et franc succès !
- Maintenant, on souhaite intégrer notre token à des plateformes d'échange décentralisées
 - › EtherDelta
 - › ForkDelta
 - › Kyber Network, 0x
- Pas besoin de passer par un exchange centralisé vulnérable !

Pour aller plus loin...

- Tester ses contrats en local grâce à Ganache et **truffle test**
 - › Voir les fichiers de tests open-zeppelin
- ICO : comment distribuer son token ?
 - › Dutch auctions
 - › Interactive ICOs

Pour aller plus loin...

- Utiliser MyEtherWallet ou Etherscan pour vérifier ses contrats et interagir avec simplement (ABI)
- Déployer un contrat sur la chaîne principale (ça coûte de l'argent) et vendre son token !
- Création de contrat pour d'autres fins : supply chain, betting etc. Un exemple disponible ici : <https://github.com/MaximeHg/sb52-contracts>



MAXIME HAGENBOURGER

CTO

maxime@blockchainpartner.fr

[+33 6 77 94 84 70](tel:+33677948470)