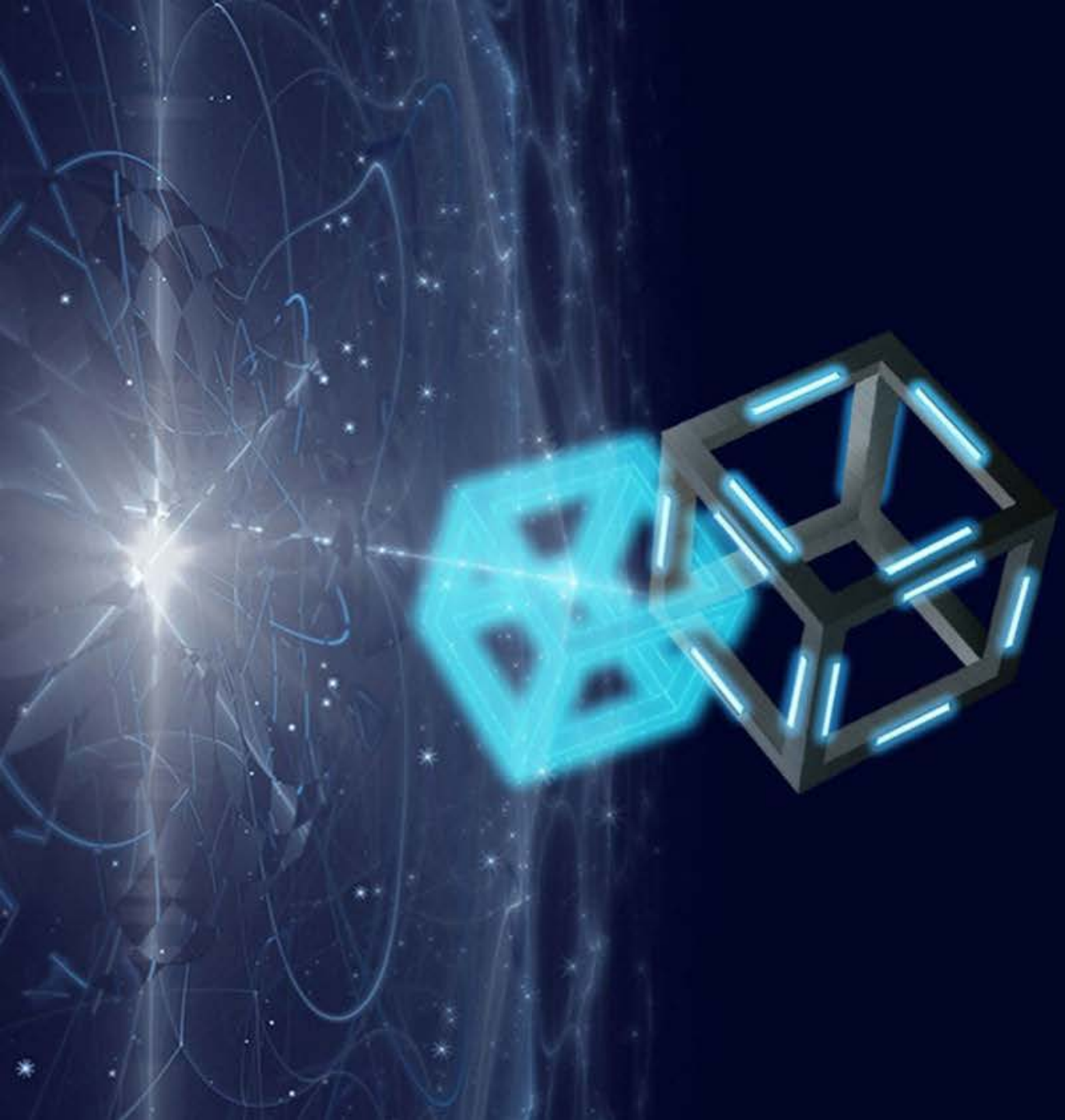

Développer et déployer des
contrats intelligents sur Ethereum
Fuzzing les Contrats
Intelligents Avec Echidna



Installation d'Echidna

Installation d'Echidna

- **Prérequis :**
  - Installer Python 3.8+
    - Télécharger Python : https://www.python.org/downloads/
    - Installation : https://www.datacamp.com/blog/how-to-install-python
  - Installer Solc-Select (permet de passer rapidement entre les versions du compilateur Solidity)
    - Lien Github officiel : https://github.com/crytic/solc-select
    - Installation : ***pip3 install solc-select***
    - **Utilisation de Solc-Select :**
      - Vérifier la version actuelle de solc : ***solc --version***
      - Installer une version spécifique de solc : ***solc-select install 0.8.20***
      - Utiliser une version spécifique : ***solc-select use 0.8.20***
  - Installer Slither : ***pip3 install slither-analyzer***

Installation d'Echidna

- macOS & Linux (avecHomebrew) : ***brew install echidna***

- Windows (installation standalone) => copier echidna.exe sur C:\Windows : https://github.com/crytic/echidna/releases

Aperçu d'Echidna

Aperçu d'Echidna

- Échidna est un fuzzer basé sur les propriétés qui tente de violer des invariants définis par l'utilisateur
- Le fuzzing implique la génération d'entrées plus ou moins aléatoires pour trouver des bogues dans un programme
- Pour chaque invariant, il génère des séquences aléatoires d'appels au contrat et vérifie si l'invariant est respecté. S'il peut trouver un moyen de falsifier l'invariant, il affiche la séquence d'appels qui le fait
- Échidna propose deux modes de test importants : le test de propriété (par défaut) et le test d'assertion

Types d'invariants

Invariants au niveau de la fonction :

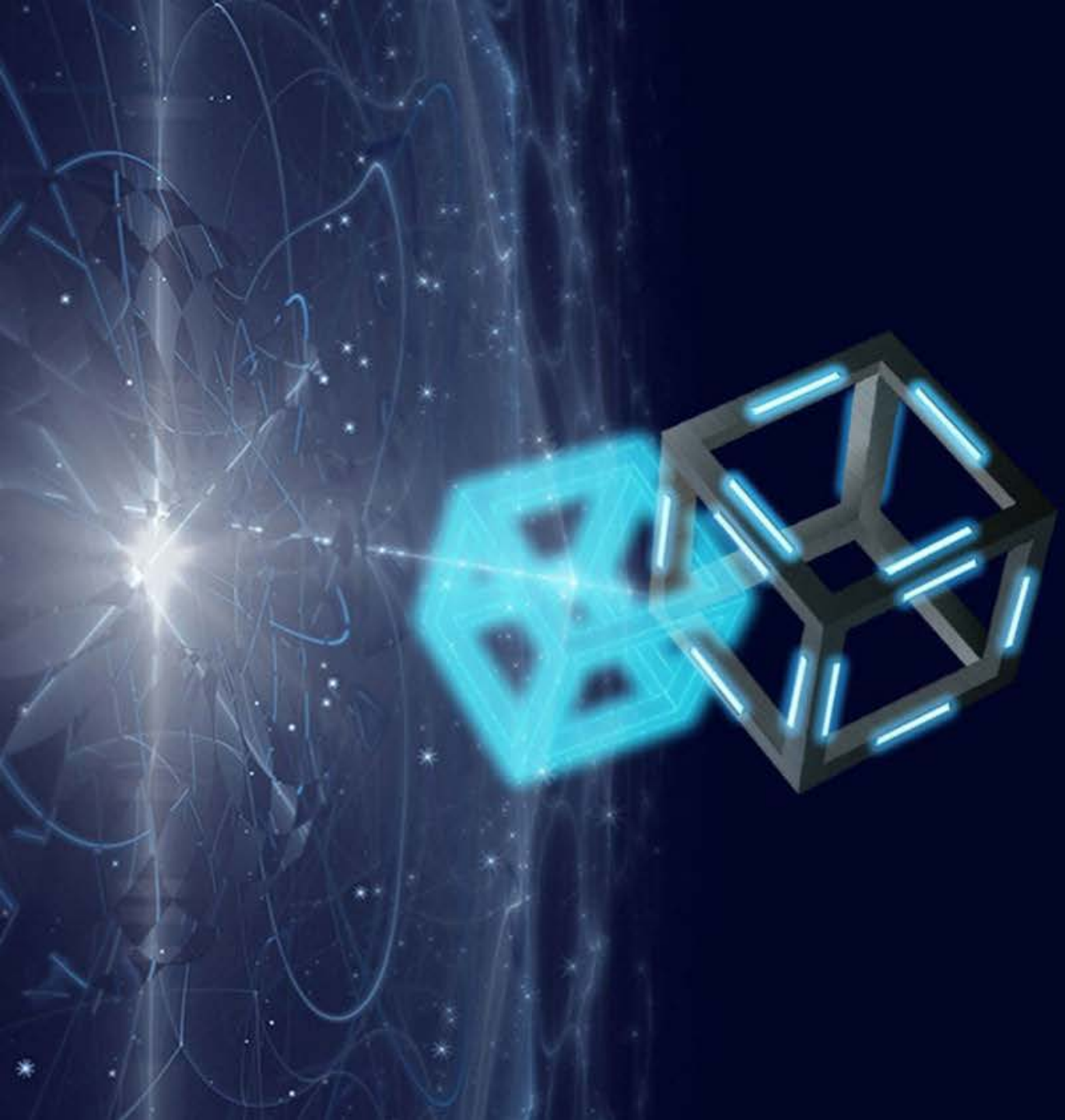
- Ne dépendent pas beaucoup du système - principalement stateless
- Peuvent être testés de manière isolée
- Hériter du contrat cible, créer des fonction(s) de test qui appellent les fonction(s) cibles, utiliser "assert" pour vérifier la propriété

```
contract TestMath is Math{
    function test_commutative(uint a, uint b) public {
        assert(add(a, b) == add(b, a));
    }
}
```

Types d'invariants

Invariants au niveau du système :

- Dépend du déploiement d'une grande partie ou de l'ensemble du système
- Les invariants sont généralement stateful
- Nécessite un déploiement / une initialisation : déployer/initialiser tout dans le constructeur
- Pour les grands systèmes, un framework comme ***Etheno*** peut être utilisé : https://github.com/crytic/etheno



Echidna
Modes de test

Mode de test de propriété

- Par défaut, Echidna utilise le mode de test "propriété", qui signale les échecs à l'aide de fonctions spéciales appelées propriétés
- Les fonctions de test sont nommées avec le préfixe : ***echidna***_
- Les fonctions de test ne prennent aucun argument et retournent toujours une valeur booléenne
- Tout effet secondaire (modification d'une variable de stockage) sera annulé à la fin de l'exécution de la propriété
- Les propriétés réussissent si elles renvoient true et échouent si elles renvoient false ou si elles sont annulées
- Alternativement, les propriétés qui commencent par "***echidna_revert***_" échoueront si elles renvoient une valeur quelconque (true ou false) et réussiront si elles sont annulées

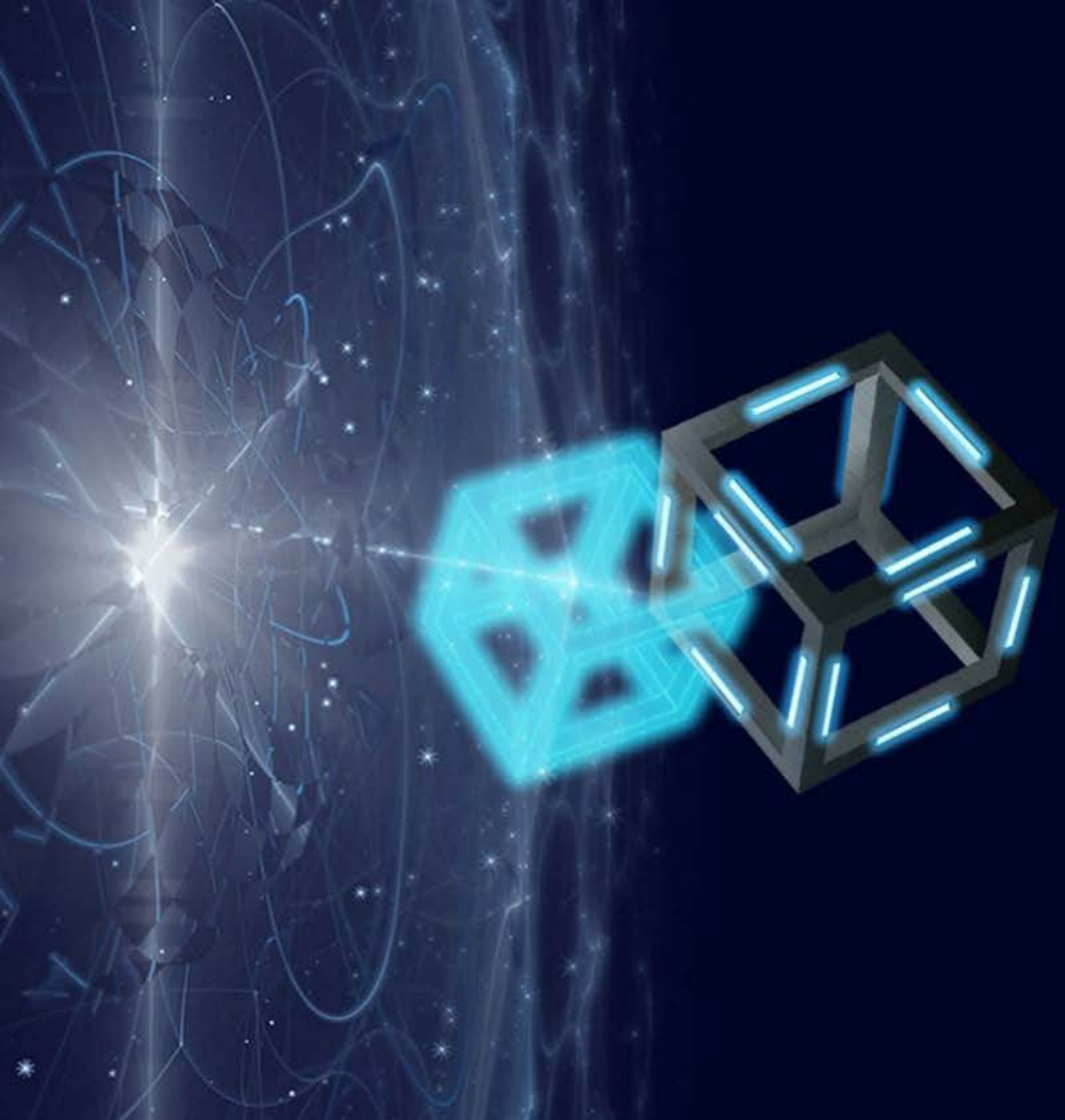
Exemple de propriété (invariant) : le solde ne doit jamais descendre en dessous de 20

```
function echidna_check_balance() public returns (bool) {
        return(balance >= 20);
}
```

Mode de test par assertion

- Ce mode est bien adapté pour un code plus complexe nécessitant la vérification ou la modification de variables d'état
- Les fonctions de test ne nécessitent pas de nom particulier
- Les arguments sont autorisé dans la fonction de test
- Les effets secondaires sont conservés s'ils ne sont pas annulés
- Les déclarations suivantes doivent être ajoutées au fichier config.yaml : ***testMode: assertion***

- Les affirmations suivantes déclenchent un échec :
  - assert(false);
  - emit AssertionFailed(...);



Exemple : Mode de test de propriété

Exemple : mode de test de propriété

```
contract Token {
    mapping(address => uint256) public balances;

    function airdrop() public {
        balances[msg.sender] = 1000;
    }

    function consume() public {
        require(balances[msg.sender] > 0);
        balances[msg.sender] -= 1;
    }

    function backdoor() public {
        balances[msg.sender] += 1;
    }
}
```

- **Propriété du contrat requise :**

  Tout le monde peut détenir un maximum de 1000 jetons

- **Echidna va :**
  - Générer automatiquement des transactions arbitraires pour tester la propriété
  - Signaler toute transaction qui conduit à ce qu'une propriété retourne false ou lance une erreur

Exemple : mode de test de propriété

- La propriété suivante vérifie que l'appelant ne peut avoir plus de 1000 jetons :

```
contract TestToken is Token {
    constructor() public {}

    function echidna_balance_under_1000() public view returns (bool) {
        return balances[msg.sender] <= 1000;
    }
}
```

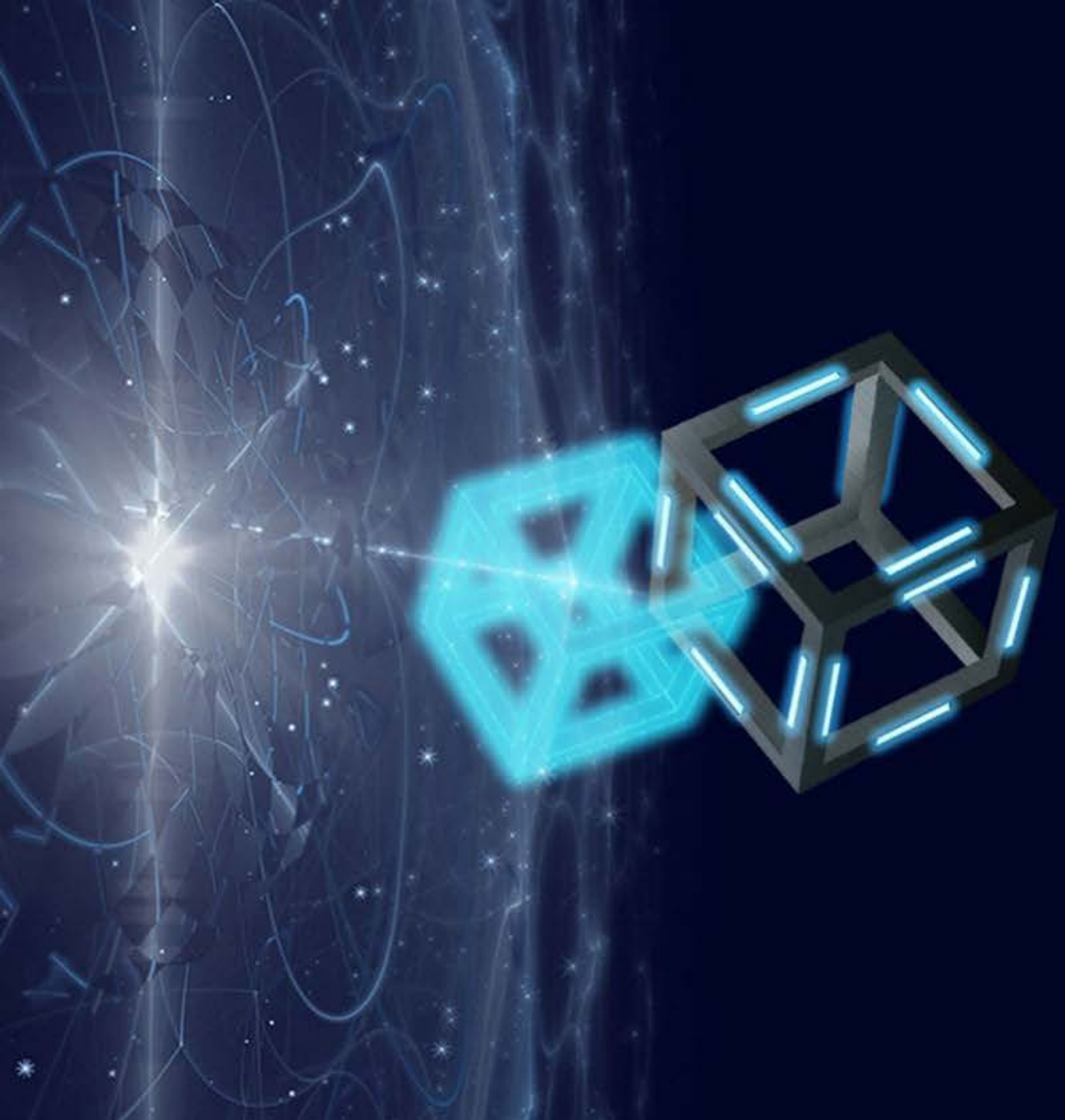
- Si un contrat nécessite une initialisation spécifique, elle doit être effectuée dans le constructeur
- Il existe quelques adresses spécifiques dans Echidna :
  - 0x30000 appelle le constructeur
  - 0x10000, 0x20000 et 0x30000 appellent aléatoirement d'autres fonctions.

Exécution d'Echidna

- echidna contract.sol
- echidna contract.sol --contract myContract
- echidna contract.sol --contract myContract --config config.yaml

```
echidna testtoken.sol --contract TestToken
...

echidna_balance_under_1000: failed!💥
  Call sequence, shrinking (1205/5000):
    airdrop()
    backdoor()
```



Optimisation, Configuration, Coverage...

Optimisation des performances du fuzzer

- À chaque fois qu'un "require" échoue, une computation est gaspillée, ce qui devrait être évité pour maximiser chaque séquence de transaction

- Limiter les entrées avec des préconditions solides ou une manipulation arithmétique :
  - Précondition : ***require(usdc.balanceOf(msg.sender) > 0))***
  - Manipulation arithmétique : ***if (abs(x) == abs(y)) { y = x + 1 };***
  - Arithmétique modulaire : ***uint256 x = inputValue % UPPER_BOUND;*** => limité à [0, UPPER_BOUND - 1]

Structuration des tests

Vérifications des preconditions :

- Obstacles à l'entrée pour le fuzzer :
- Ne pas tester cette propriété à moins que ces préconditions ne soient remplies => par exemple : ***require(usdc.balanceOf(msg.sender) > 0))***

Action :

- Exécutez la fonction/scénario à tester => par exemple : ***usdc.transfer(to, amount)***

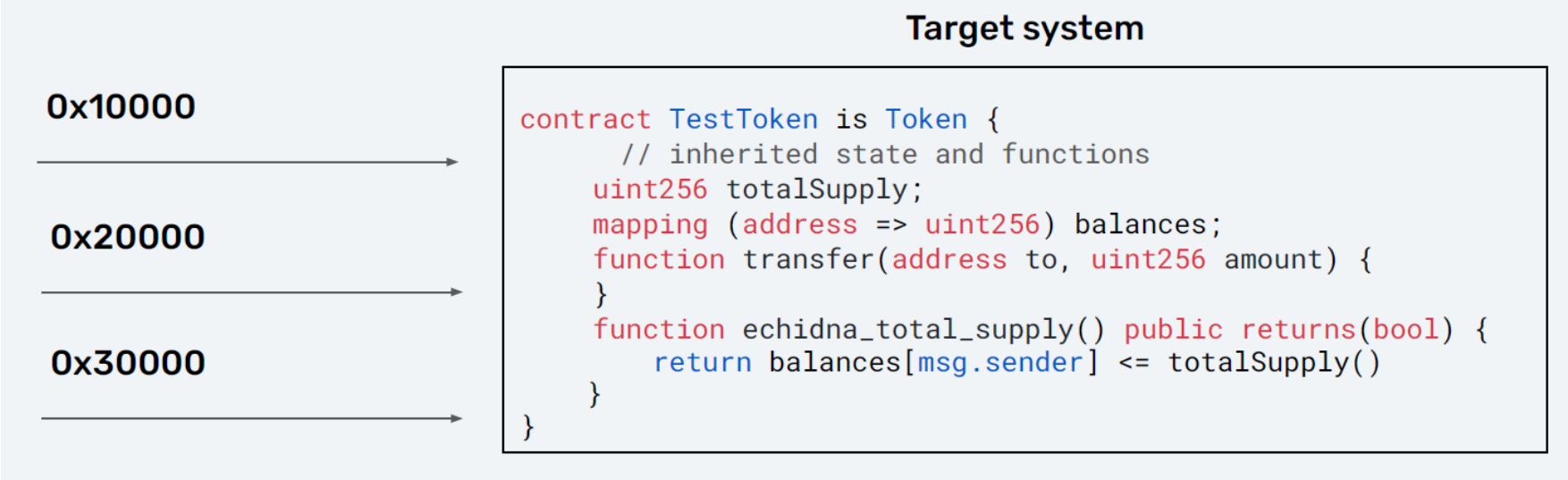
Vérifications des post-conditions :

- Ce sont les "vérités" que nous testons
- Il faut tester à la fois le scénario idéal et les scénarios moins favorables (try/catch)
- Exemple : ***assert(usdc.balanceOf(msg.sender) < 1000);***

Tests internes versus tests externes

Tests internes :

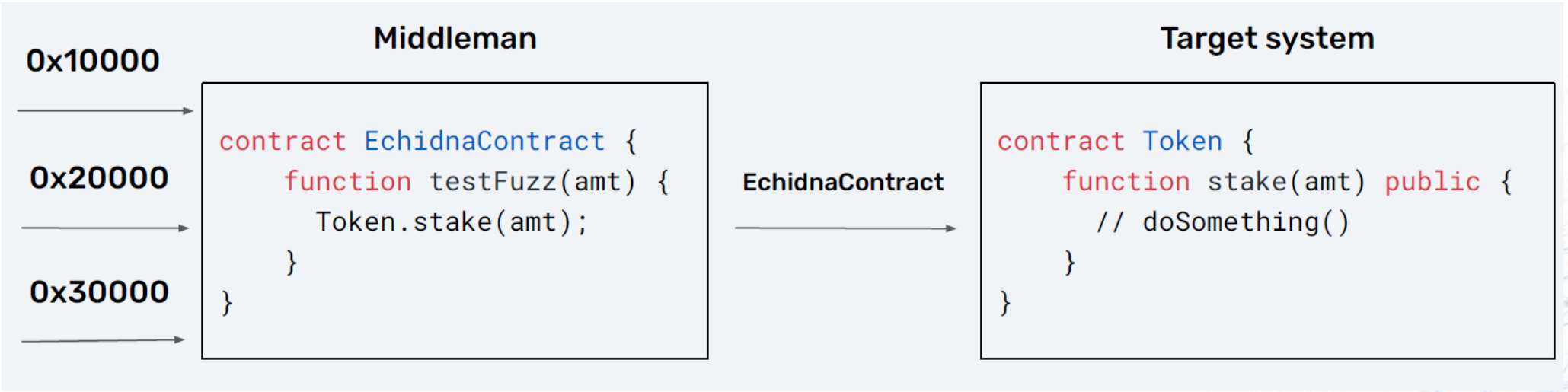
- Utilisation de l'héritage pour tester le contrat cible
- Facile à mettre en place
- Obtenir l'état et toutes les fonctions publiques/externes des contrats hérités
- msg.sender est préservé
- Non adapté aux systèmes complexes



Tests internes versus tests externes

Tests externes:

- Utilisation d'appels externes aux contrats cibles
- Plus difficile à configurer
- msg.sender n'est pas préservé
- Utilisé pour les systèmes complexes



Les tests stateless et les tests stateful

- Les deux modes de test peuvent être utilisés, soit en mode stateful (par défaut), soit en mode stateless (en utilisant ***seqLen: 1*** dans config.yaml)

- En mode stateful, Echidna conserve l'état entre chaque appel de fonction et tente de violer les invariants. En mode stateless, Echidna ignore les changements d'état pendant le fuzzing

Adresses spécifiques dans Echidna :

- 0x30000 appelle le constructeur
- 0x10000, 0x20000, and 0x30000 appellent aléatoirement d'autres fonctions

Configuration d'Echidna

- Fichier YAML par défaut : https://github.com/crytic/echidna/blob/master/tests/solidity/basic/default.yaml
- Options de configuration : https://github.com/crytic/echidna/wiki/Config

- Exécution d'Echidna : ***echidna contract.sol --contract myContract --config config.yaml***

Exclusion des fonctions :

filterBlacklist: true
filterFunctions: ["myContract.function1(uint256,uint256)", " myContract.function1()"]

Inclusion des fonctions :

filterBlacklist: false
filterFunctions: ["myContract.function3()", " myContract.function4()"]

Estimation de la consommation de gaz :

estimateGas: true

Echidna Coverage

- "Coverage" est le suivi du code touché par le fuzzer. Echidna peut enregistrer les informations de couverture dans un répertoire spécifié avec l'option ***corpusDir*** dans le fichier config.yaml
- Créer un répertoire corpus dans la racine du projet - par exemple : corpus/
- Ajoutez au fichier config.yaml : ***corpusDir: "corpus"***
- Exécutez Echidna avec le fichier config.yaml : ***echidna myContract.sol --config config.yaml***
- Vérifiez le ***fichier corpus/covered.*.t***xt - exemple :

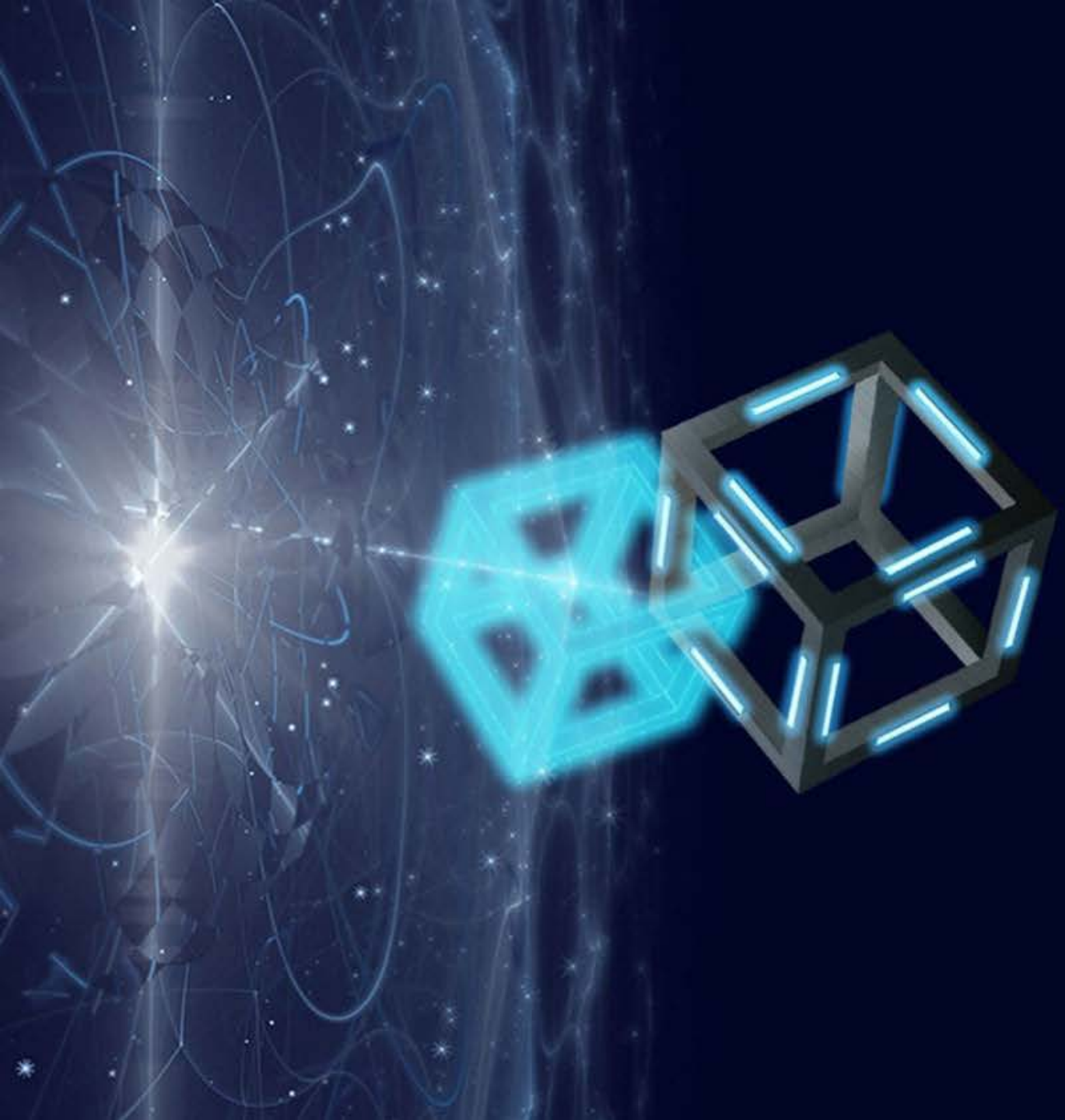
```
*r    function set0(int val) public returns (bool){
*       if (val % 100 == 0)
*         flag0 = false;
```

* : Si une exécution s'est terminée par un STOP - la ligne a été exécutée sans erreur

r : si une exécution s'est terminée par un REVERT

o : si une exécution s'est terminée par une erreur de gas - commun avec les boucles

e : si une exécution s'est terminée par une autre erreur (division par zéro, échec d'assertion, etc.)



Exemple : Mode de test par assertion

Exemple: mode de test par assertion - UniswapV2 pool liquidity

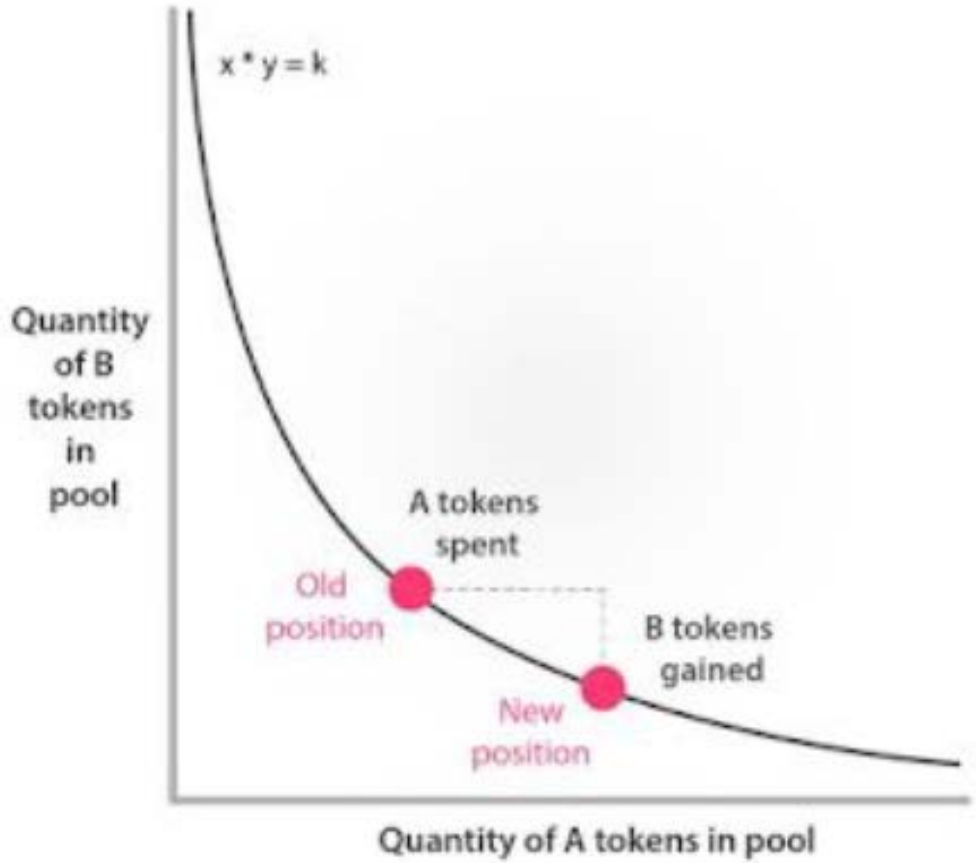
Qu'est-ce qu'un Market Maker Automatisé (AMM) ?

- Échange sans carnet d'ordres
- Le prix est basé sur la formule de liquidité du pool => x * y = k
- Le prix est calculé comme un ratio entre deux actifs
- k (invariant du pool) est constant

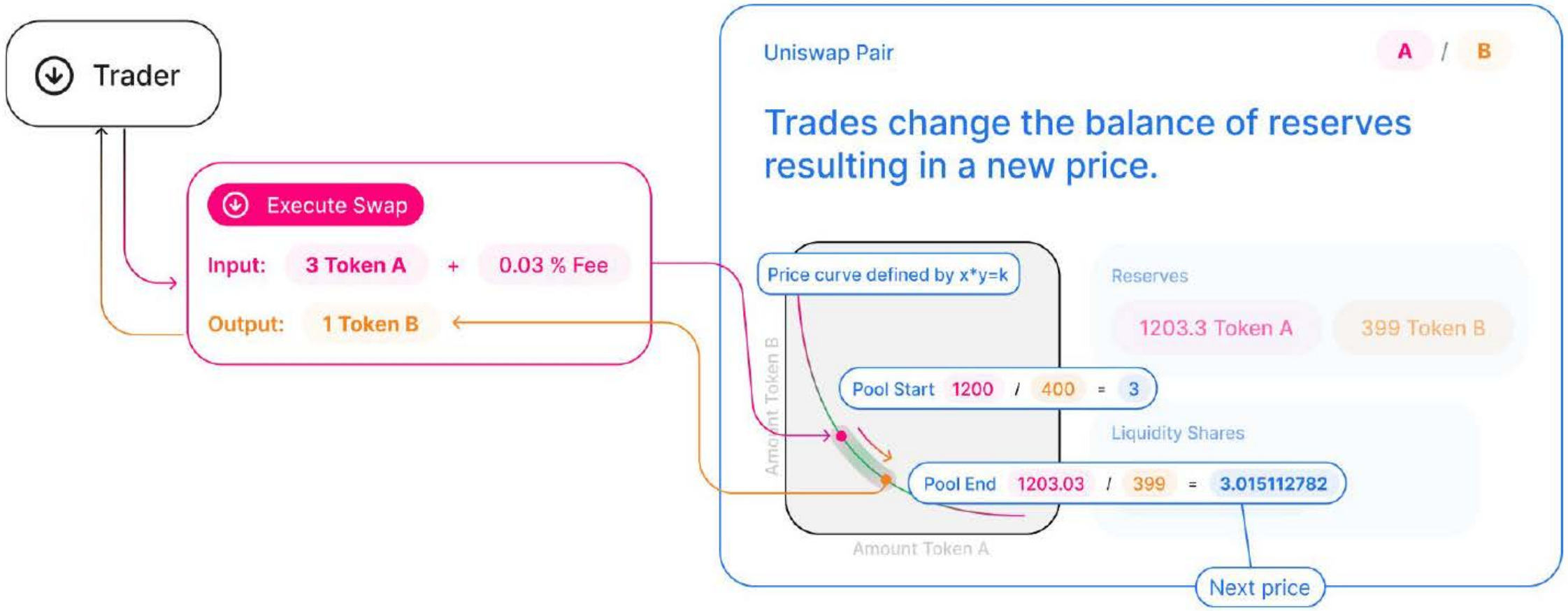
Exemple: mode de test par assertion - UniswapV2 pool liquidity

Échangez une quantité x de jetonA contre du jetonB - combien de jetonB obtenons-nous en retour ?

- Δx : La quantité de jetonA que nous échangeons contre du jetonB (Δy)
- x * y = k
- (x + Δx) * (y - Δy) = k
  => le pool gagne Δx et perd Δy
- Δy = y - (k / (x + Δx))



Exemple: mode de test par assertion - UniswapV2 pool liquidity



Exemple: mode de test par assertion - UniswapV2 pool liquidity

UniswapV2 Core comprend deux contrats importants :

- Factory : crée des contrats de paire uniques pour chaque pool
- Pair : représente le pool de liquidité, suit les soldes des jetons et contient la logique de swap de base

Invariants :

- Invariant 1 : lorsque nous ajoutons de la liquidité au pool, notre pile de jetons LP doit augmenter
- Invariant 2 : lorsque nous ajoutons de la liquidité (x et y) au pool, k = x * y doit augmenter