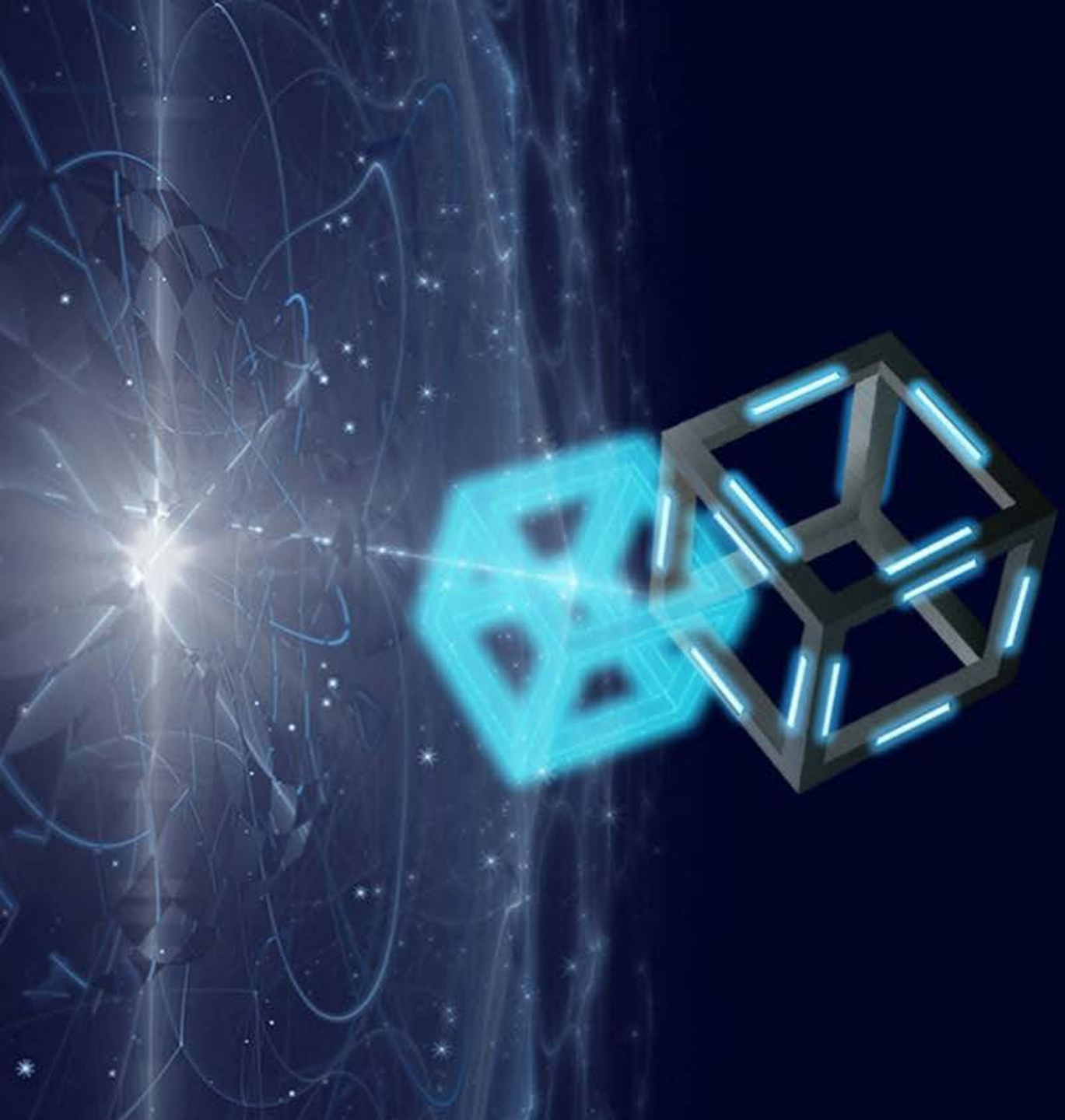

Développer et déployer des
contrats intelligents sur Ethereum
Tester les Contrats Intelligents
Avec Foundry



Installation de Foundry

Installation de Foundry

Installation de Git (requis pour Foundry):

- Linux Fedora: $ sudo dnf install git-all
- Linux Ubuntu: $ sudo apt install git-all

- MacOS: $ git --version => demandera de l'installer Git

- Windows: https://git-scm.com/download/win

Installation de Foundry avec Foundryup (Foundry toolchain installer)

- Pour Windows, installer GitBash ou WSL - PowerShell et CMD ne sont pas pris en charge !

- Terminal / GitBash: ***curl -L https://foundry.paradigm.xyz | bash*** => installation de foundryup

- Terminal / GitBash: ***foundryup*** => mise à jour de foundry, forge, cast, anvil and chisel

Commandes de base de Foundry

Commandes de base de Foundry 1/2

- Foundry Book – documentation officielle : https://book.getfoundry.sh
- Créer un nouveau projet : ***forge init projectName***

  Installation automatique de la bibliothèque standard Forge => bibliothèque de test préférée pour les projets Foundry
- Build du projet : ***forge build***
- **Exécuter le(s) test(s) :**
  - ***forge test***
  - ***forge test --match-contract ContractName --match-test testName --gas-report -vvvv***
- Couverture du code : ***forge coverage***

Commandes de base de Foundry 2/2

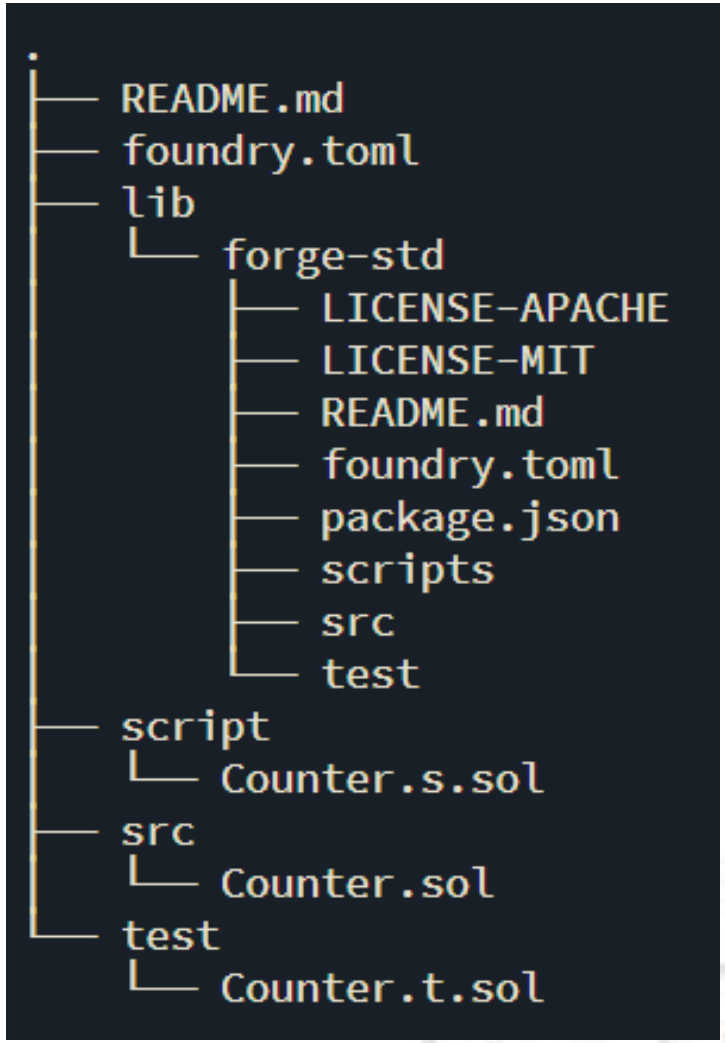
- Installer toutes les dépendances pour un projet existant (git clone...) : ***forge install***
- Installer des dépendances spécifiques :

  forge install OpenZeppelin/openzeppelin-contracts
  forge install transmissions11/solmate@v7

- Mettre à jour une dépendance : ***forge update lib/solmate***
- Supprimer une dépendance : ***forge remove lib/solmate***

Disposition du projet

- Fichier de configuration : ***foundry.toml***
- Le répertoire par défaut pour les contrats : ***src***
- Le répertoire par défaut pour les tests : ***test*** => tout contrat avec une fonction commençant par "test" est considéré comme un test
- Les dépendances sont stockées dans le répertoire : ***lib***



Déploiement & vérification des contrats

```
forge create --rpc-url <your_rpc_url> \
  --constructor-args "ForgeUSD" "FUSD" 18 1000000000000000000000 \
  --private-key <your_private_key> \
  --etherscan-api-key <your_etherscan_api_key> \
  --verify \
  src/MyToken.sol:MyToken
```

Exemple:

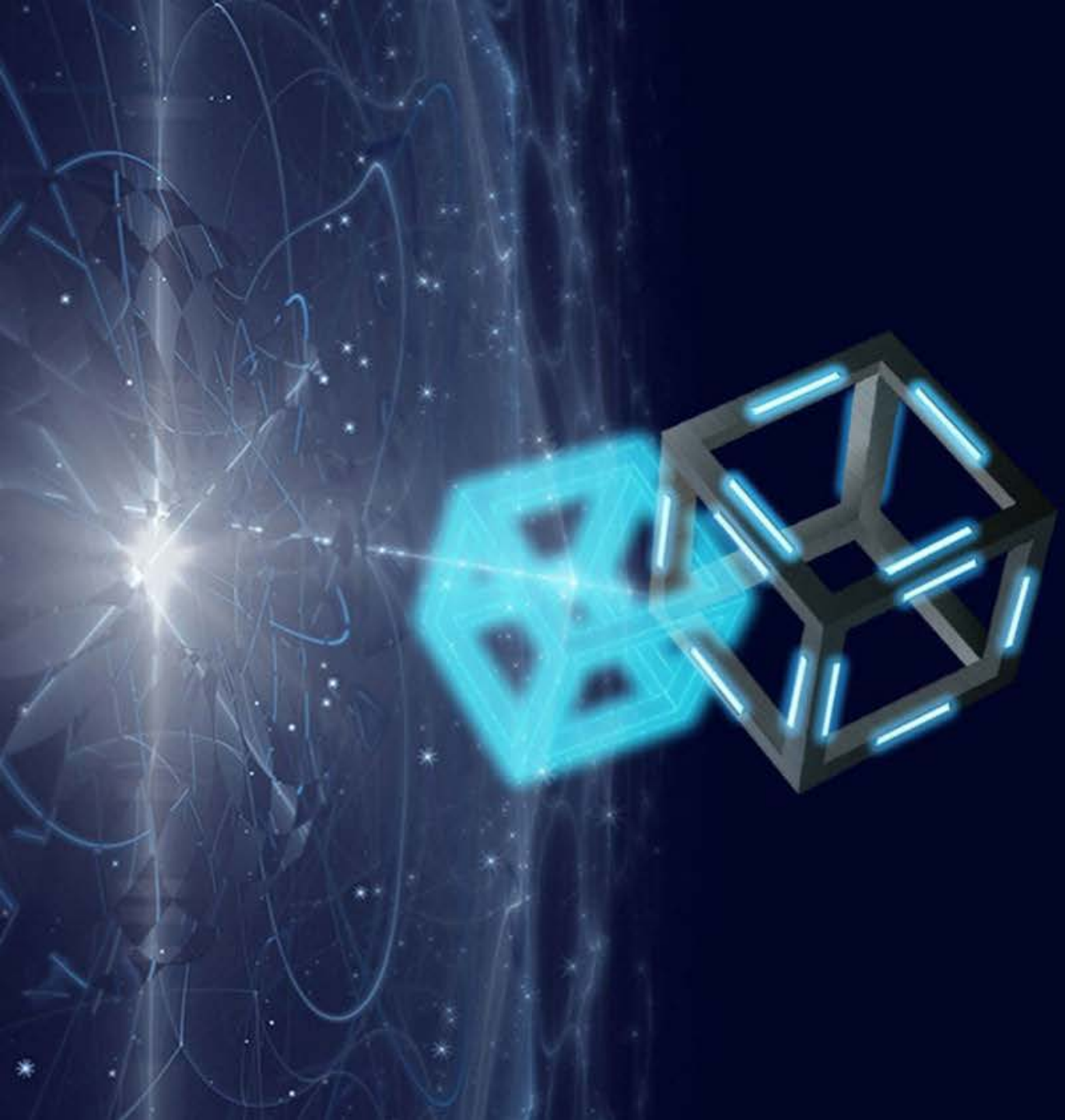
forge create --rpc-url sepolia --private-key $PK_SEPOLIA --constructor-args "My NFT" "MNFT" "baseUri" --etherscan-api-key sepolia --verify src/2_StandardUnitTests.sol:NFT

- Configurations pour les RPC et Etherscan (par exemple: Sepolia) sont dans ***config.tom***l
- Configuration des variables d'environnement (e.g.: PK_SEPOLIA) : dans ***.bash_profile*** (C:/User/USERNAME/) => export PK_SEPOLIA="0x77..."

Configuration de Foundry - foundry.toml

```
[profile.default]
src = "src"
out = "out"
libs = ["lib"]
solc_version = "0.8.20"
optimizer = true
optimizer_runs = 200

remappings = [
    "openzeppelin-contracts/=lib/openzeppelin-contracts/"
]

[rpc_endpoints]
sepolia = "${ALCHEMY_SEPOLIA_API_URL}"

[etherscan]
sepolia = { key = "${ETHERSCAN_SEPOLIA_API_KEY}", url = "https://api-sepolia.etherscan.io/api" }

[fuzz]
runs = 256
depth = 15
fail_on_revert = false

```

Options supplémentaires : https://book.getfoundry.sh/reference/config/



Tester avec Foundry

Tester avec Foundry

- Les tests sont écrits en Solidity. S'il y a une "revert", le test échoue, sinon, il réussit
- Les fonctions préfixées par test_ sont exécutées comme des cas de test
- Les fonctions de test doivent avoir une visibilité externe ou publique
- La méthode préférée pour écrire des tests : utiliser le contrat de test de la bibliothèque standard Forge => ***import "forge-std/Test.sol";***
- ***setup()*** : fonction facultative invoquée avant l'exécution de chaque cas de test => souvent utilisée pour déployer d'autres contrat(s) qui doivent être testés

Tester avec Foundry

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.20;

contract Counter {
    uint256 public number;

    function setNumber(uint256 newNumber) public {
        number = newNumber;
    }

    function increment() public {
        number++;
    }
}
```

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.20;

import {Test, console} from "forge-std/Test.sol";
import {Counter} from "../src/Counter.sol";

contract CounterTest is Test {
    Counter public counter;

    function setUp() public {
        counter = new Counter();
        counter.setNumber(0);
    }

    function test_Increment() public {
        counter.increment();
        assertEq(counter.number(), 1);
    }
}
```

Foundry Asserts

- ***assertEq*** *: vérifier l'égalité*
- ***assertLt*** *: vérifier inférieur à*
- ***assertLe*** *: vérifier inférieur ou égal à*
- ***assertGt*** *: vérifier supérieur à*
- ***assertGe*** *: vérifier supérieur ou égal à*
- ***assertTrue*** *: vérifier que c'est vrai*
- *Les deux premiers arguments de l'assertion sont les arguments de comparaison. Un message d'erreur peut être fourni en tant que troisième argument* :

  assertEq(someNumber, 55, "expect someNumber to equal to 55");

Foundry Cheatcodes – comptes & adresses

Changer le msg.sender pour une adresse spécifique :

- ***vm.prank(someAddress);***
  myContract.someFonction(); // msg.sender est someAddress

- vm.prank fonctionne uniquement pour la transaction qui se produit immédiatement après. Si plusieurs transactions doivent utiliser la même adresse, utiliser vm.startPrank et vm.stopPrank :

  vm.startPrank(someAddress);
  myContract.function1();
  myContract.function2();
  vm.stopPrank();

Les adresses :

- address owner = address(1234);
- address owner = 0x0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045;
- address alice = ***makeAddr("alice");***

Foundry Cheatcodes – Reverts, Errors & Events

- **Tester les Reverts:**

  vm.expectRevert("incorrect amount");
  someContract.depositExactly1Ether{value: 1 ether + 1 wei}();

- **Tester les erreurs personnalisées :**

  error SomeError(uint256); // l'erreur spécifique doit être déclarée dans le contrat de test

  vm.expectRevert(abi.encodeWithSelector(SomeError.selector, 5));
  customErrorContract.functionReverstsWithSomeError(5);

- **Tester les événements :**

  L'événement doit être émis dans le test pour garantir son fonctionnement dans le contrat intelligent

  vm.expectEmit();
  emit EventName();
  someContract.functionThatEmitsEvent();

Foundry Cheatcodes – Timestamp & Balances

- **Ajuster le block.timestamp**

  vm.warp(1680616584 + 3 days);

- **Ajuster le block.number**

  vm.roll(1000);

- **Définir les soldes des adresses:**

  address alice = makeAddr("alice");
  vm.deal(alice, balanceToGive);

Exemple : Tester un contrat NFT simple

- **Installer les dépendances :**

  forge install transmissions11/solmate Openzeppelin/openzeppelin-contracts

- **Tester le contrat :**

  forge test --match-contract NFTTest --gas-report –vv
  forge test --match-contract NFTTest --match-test test_RevertMintWithoutValue --gas-report -vv

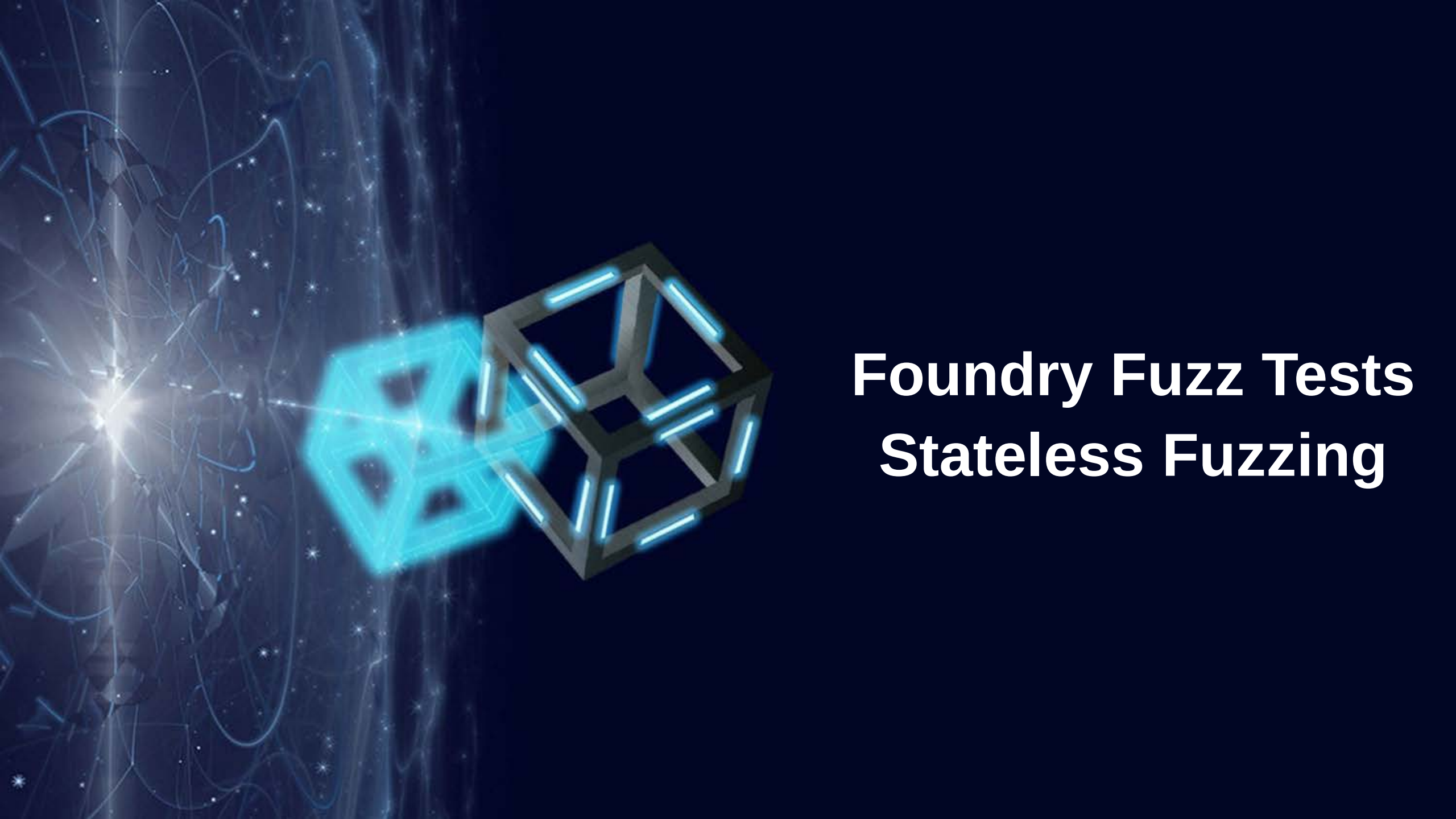
- **Déployer et verifier le contrat:**

  forge create --rpc-url sepolia --private-key $PK_SEPOLIA --constructor-args "My NFT" "MNFT" "baseUri" --etherscan-api-key sepolia --verify src/NFT.sol:NFT

- **Exécuter les fonctions du contrat en utilisant cast**

  cast send --rpc-url=sepolia --private-key=$PK_SEPOLIA "CONTRACT_ADDRESS" "mintTo(address)" "RECEIVER_ADDRESS"

  cast call --rpc-url=sepolia "CONTRACT_ADDRESS" "ownerOf(uint256)" 1



Foundry Fuzz Tests
Stateless Fuzzing

Foundry Fuzz Tests - Stateless Fuzzing

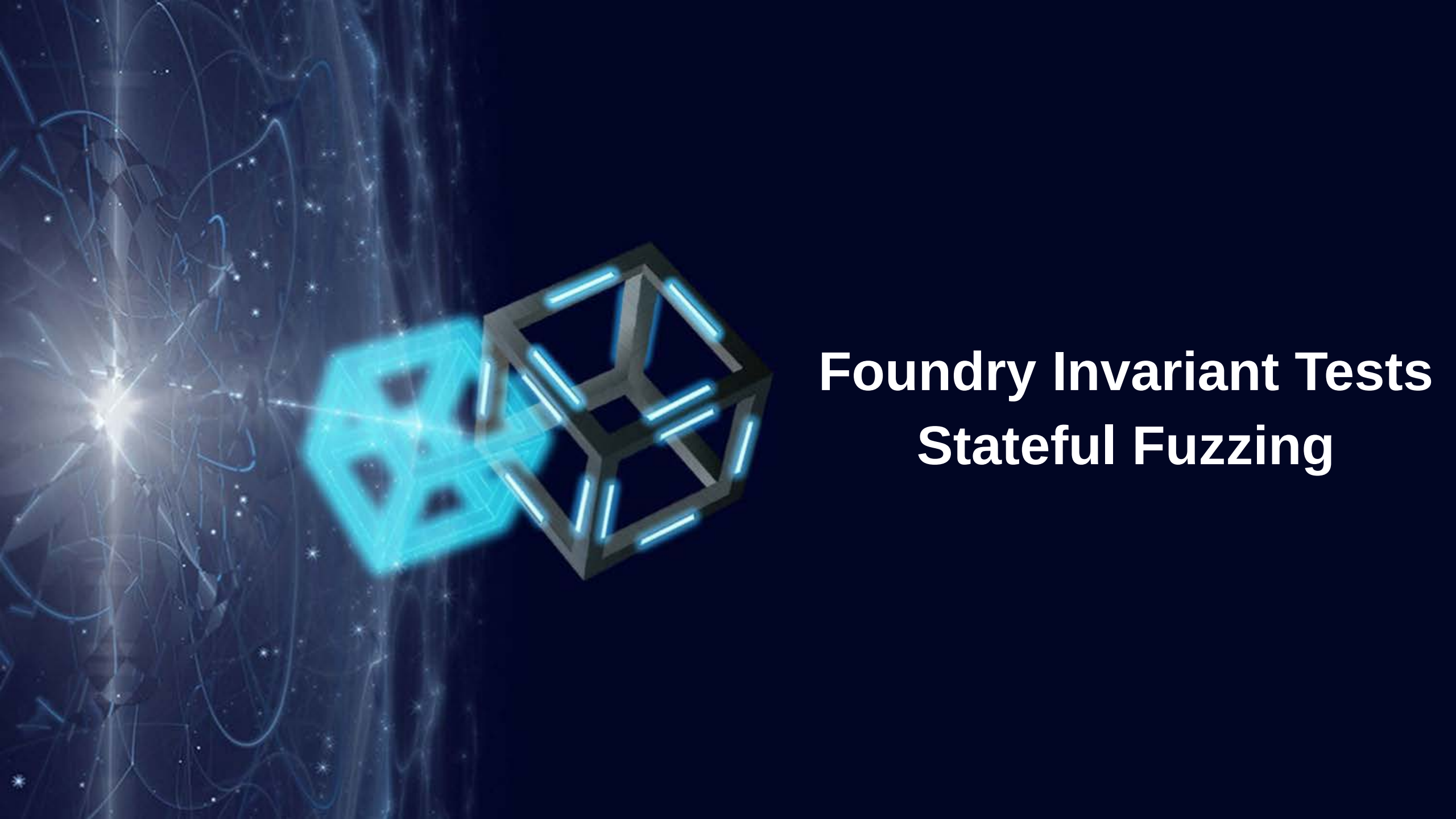
- Stateless fuzzing : l'état des variables sera oublié à chaque exécution
- Foundry exécute tout test qui a au moins un paramètre comme un test de fuzz
- Foundry exécute le test avec différentes valeurs pour les arguments spécifiés
- **Configuration de fuzzing :**
  - Le nombre d'exécutions et d'autres paramètres peuvent être configurés dans la section [fuzz] du fichier foundry.toml =>
  - ***runs*** : Le nombre d'exécutions à effectuer pour chaque cas de test - par défaut : 256
  - ***depth*** : Le nombre d'appels exécutés pour tenter de rompre les invariants en une seule exécution - par défaut : 15
  - ***fail_on_revert*** : Échoue le test de fuzz en cas d'une revert - par défaut : false
  - Paramètres supplémentaires : https://book.getfoundry.sh/reference/config/testing#fuzz

Exemple : Stateless Fuzzing

- Un invariant est une condition qui doit toujours être vraie
- Notre ***invariant*** : un utilisateur ne devrait jamais pouvoir retirer plus d'argent qu'il n'a déposé
- Importer la bibliothèque de tests standard (***forge-std/Test.sol***)
- Hériter du contrat de test de la bibliothèque de tests standard
- Dans la fonction ***setup(),*** déployer le contrat qui doit être testé
- Créer des fonctions de test avec des paramètres d'entrée
- Foundry appellera ces fonctions de test avec des paramètres d'entrée aléatoires
- Si l'invariant n'est pas respecté, le test va échouer

Exemple : Stateless Fuzzing

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity 0.8.20;
3
4  import {Test} from "forge-std/Test.sol";
5  import "../src/3_FuzzingStateless.sol";
6
7  contract SimpleDappTest is Test {
8      SimpleDapp simpleDapp;
9      address public user;
10
11     function setUp() public {
12         simpleDapp = new SimpleDapp();
13         user = address(this);
14     }
15
16     function test_DepositAndWithdraw(uint256 depositAmount, uint256 withdrawAmount) public payable {
17         // Ensure the user has enough Ether to cover the deposit
18         uint256 initialBalance = 100 ether;
19         vm.deal(user, initialBalance);
20         vm.deal(address(simpleDapp), initialBalance);
21
22         if (depositAmount <= initialBalance) {
```



Foundry Invariant Tests
Stateful Fuzzing

Foundry Invariant Tests - Stateful Fuzzing

- **Stateful fuzzing:** L'état de l'exécution précédente est l'état de départ du prochaine exécution de fuzz
- Dans un test de "stateful fuzzing", les fonctions d'un contrat sont appelées de manière aléatoire avec des entrées aléatoires par le fuzzer, cherchant à rompre tout invariant spécifié
- Pour écrire un test de "stateful fuzzing", utiliser le mot-clé "invariant" :

  function invariant_testAlwaysReturnsZero () public { ...

Exemple : Stateful Fuzzing

- ***Notre invariant :*** N'importe quel montant déposé devrait être retiré par la même personne
- ***targetContract() :*** définir le contrat qu'on va tester. En définissant un contrat cible, Foundry commencera automatiquement à exécuter toutes les fonctions du contrat de manière aléatoire avecinir des paramètres aléatoires

  targetContract(adresse(ContratSélectionné));
- ***Exécuter le test :*** forge test --match-contract BankTest --mtinvariant_alwaysWithdrawable
- Pourquoi changeBalance() est-il appelé ?

Handler-based Testing

- Un contrat de type "***handler***" est utilisé pour tester des protocoles ou des contrats plus complexes - nécessaire lorsque l'environnement doit être configuré d'une manière spécifique
- Le "handler" est utilisé pour interagir avec le contrat cible
- Créer un dossier ***handler*** à l'intérieur du dossier de test et un fichier ***handler.sol*** à l'intérieur avec des fonctions d'enveloppe ("***wrapper***") pour toutes les fonctions cibles qui doivent être appelées par le fuzzer
- Créer un fichier de test et déployer le handler dans la fonction ***setup()***
- Il faut définir le contrat handler comme contrat cible en utilisant la fonction ***targetContract()***
- Ajouter des fonctions de test d'invariant au fichier de test => les noms de fonction commencent par : ***invariant***_ et ils vérifient les invariants spécifiques au protocole
- Seules les fonctions définies dans le handler seront appelées de manière aléatoire par le fuzzer
- Si une fonction dans le contrat principal nécessite une certaine condition avant de pouvoir être appelée, on peut facilement la définir dans le handler avant l'appel de fonction

Handler-based Testing

```
4   import "forge-std/Test.sol";
5   import "../../src/5_FuzzingStatefulWithHandler.sol";
6
7   contract Handler is Test {
8       BankWithHandler bank;
9       bool canWithdraw;
10
11      constructor(BankWithHandler _bank) {
12          bank = _bank;
13          vm.deal(address(this), 100 ether);
14      }
15
16      function deposit() external payable {
17          uint256 amount = msg.value;
18          vm.assume(amount > 10); //use assume only fo
19          amount = bound(amount, 1 ether, 100 ether);
20
21          bank.deposit{value: amount}();
22
23          canWithdraw = true;
24      }
25
```

```
4   import {Test} from "forge-std/Test.sol";
5
6   import "../src/5_FuzzingStatefulWithHandler.sol";
7   import "./handlers/Handler.sol";
8
9   contract BankTestWithHandler is Test {
10      BankWithHandler bank;
11      Handler handler;
12
13      function setUp() external {
14          bank = new BankWithHandler{value: 25 ether}();
15          handler = new Handler(bank);
16
17          // set the handler contract as the target for our test
18          targetContract(address(handler));
19      }
20
21      function invariant_bankBalanceAlwaysGreaterThanInitialBalance()
22          assert(address(bank).balance >= bank.initialBankBalance());
23      }
```