Smart contract testing

John Hughes

CHALMERS QuviQ

”DAO attack”

June 2016

$60 million stolen

ethereum

CARDANO

Search docs

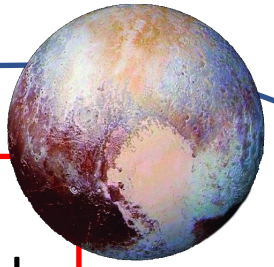
Property-based testing of Plutus contracts

Plutus comes with a library for testing contracts using QuickCheck. Tests generated by this library perform a sequence of calls to contract endpoints, checking that tokens end up in the correct wallets at the end of each test. These sequences can be generated at random, or in a more directed way to check that desirable states always remain reachable. This tutorial introduces the testing library by walking through a simple example: a contract that implements a guessing game.

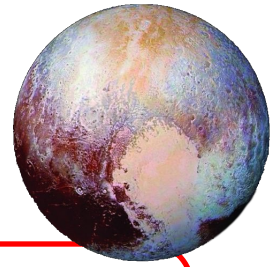
An overview of the guessing game

The source code of the guessing game contract is

$$\sum \text{outputs} = \sum \text{inputs} + \text{tokens forged}$$



Verifier
TX ➔ Bool

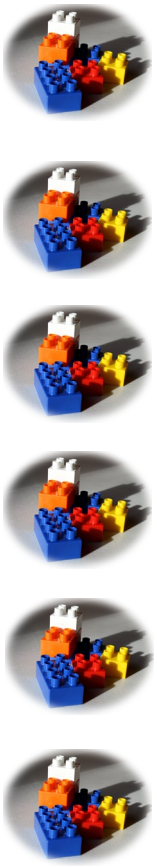


Monetary
policy
TX ➔ Bool

TX TX

Extended UTXO

Contract endpoints



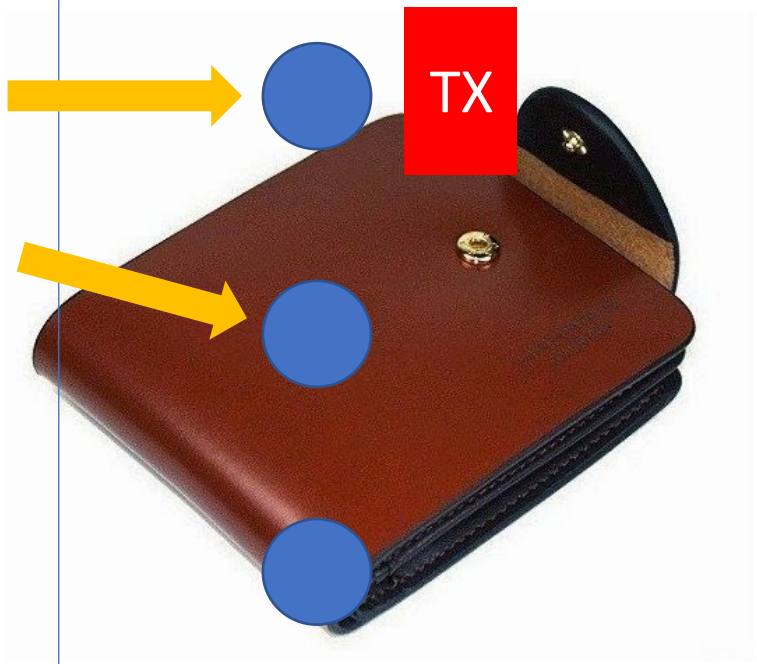
Action

Action

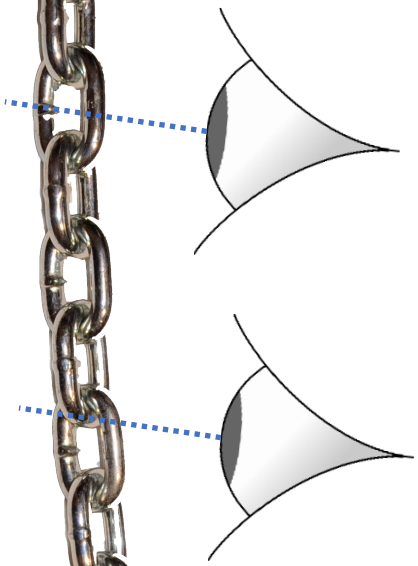
Action

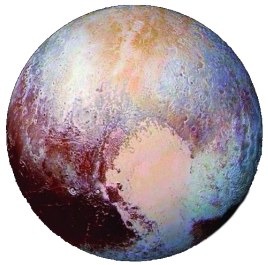
Action

Action

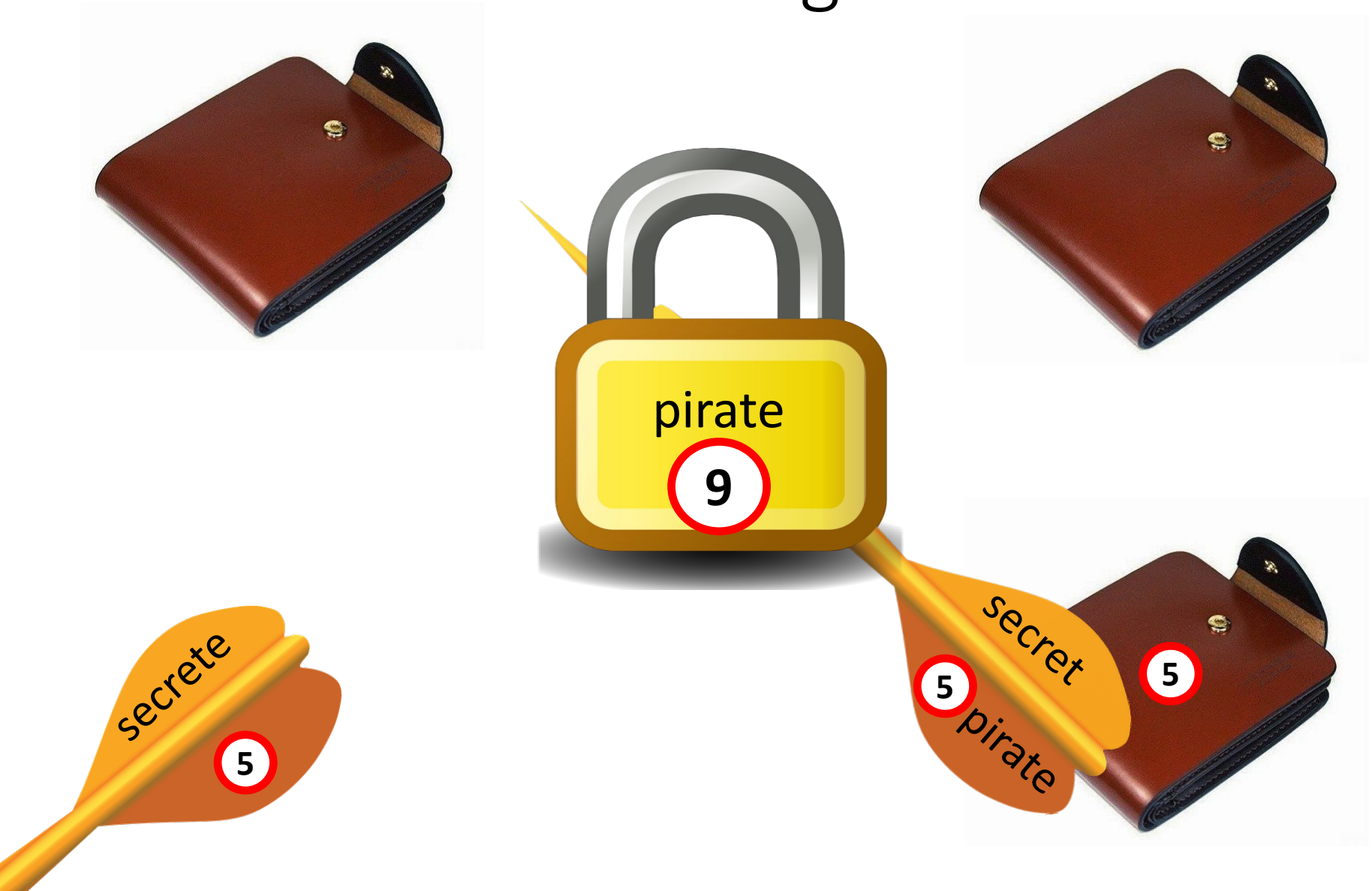


Do the wallets have the right contents?

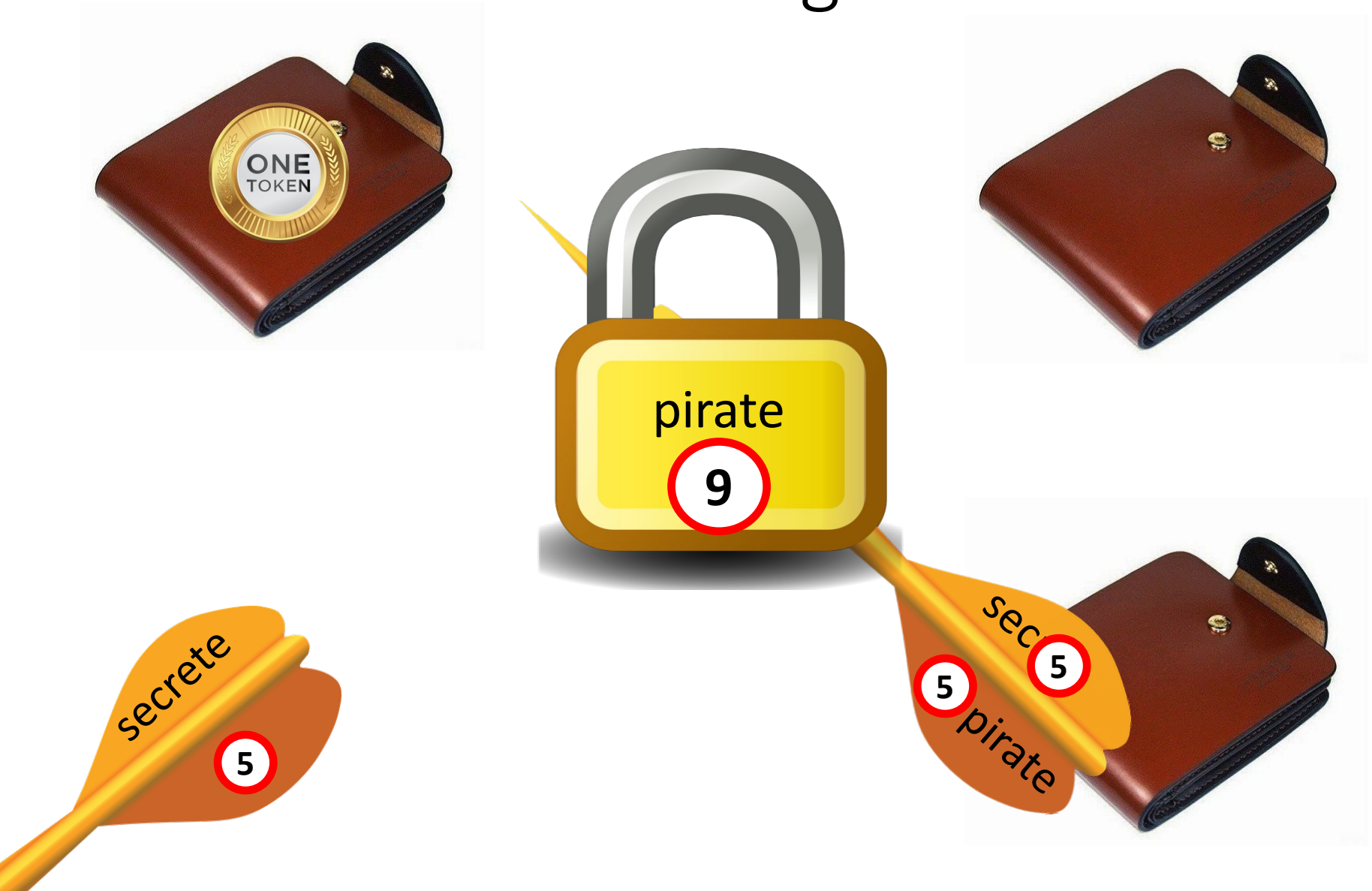




The Guessing Game



The Guessing Game



Modelling Actions

To model a guess:

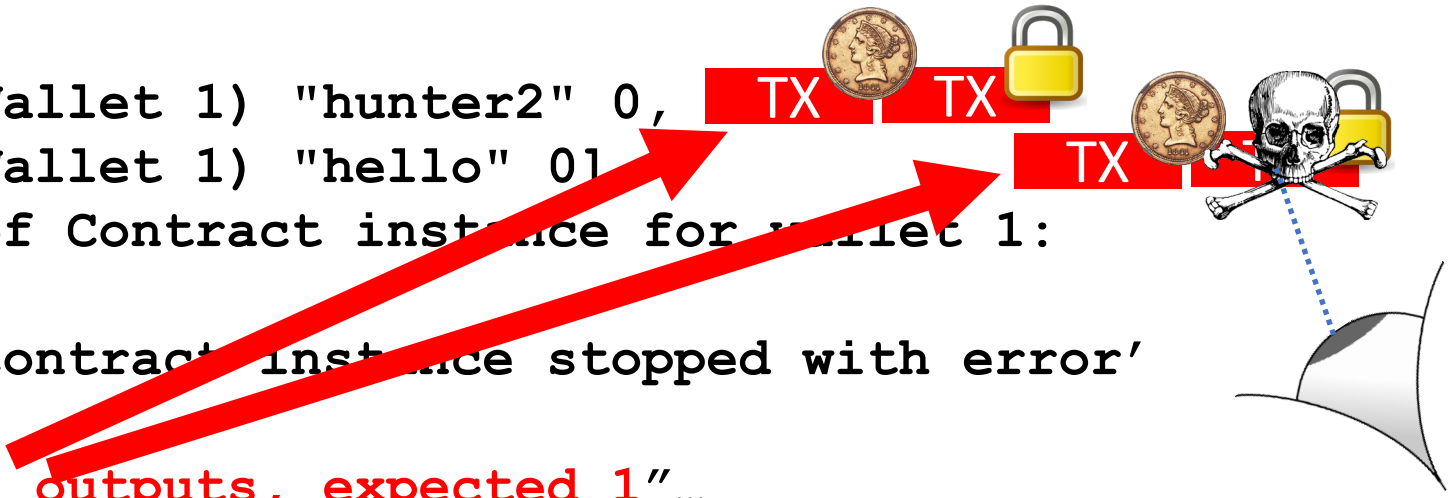
What is the password?
How much is locked?
Where is the game token?

```
nextState (Guess w old new val) = do
    correctGuess <- (old ==)    <$>
      viewContractState currentSecret
    holdsToken   <- (Just w ==) <$>
      viewContractState hasToken
    enoughAda    <- (val <=)    <$>
      viewContractState gameValue
    when (correctGuess && holdsToken && enoughAda) $ do
        deposit w     $ Ada.lovelaceValueOf val
        currentSecret $= new
        gameValue     $~ subtract val
```

What happens when we test?

```
> quickCheck prop_Game
*** Failed! Assertion failed (after 5 tests and 7
shrinks):
Actions
 [Lock (Wallet 1) "hunter2" 0,
  Lock (Wallet 1) "hello" 0]
Outcome of Contract instance for wallet 1:
  False
Failed 'Contract instance stopped with error'
```

…"Found 2 outputs, expected 1"…



Preconditions

Precondition for Lock:

- There is no game token yet

What happens now?

```
> quickCheck $ propGame' Warning
*** Failed! Assertion failed (after 10 tests and 6
shrinks):
Actions
  [Lock (Wallet 2) "hello" 0,
   Guess (Wallet 1) "hello" "secret" 0]
Outcome of Contract instance for wallet 1:
   False
Failed 'Contract instance stopped with error'
```

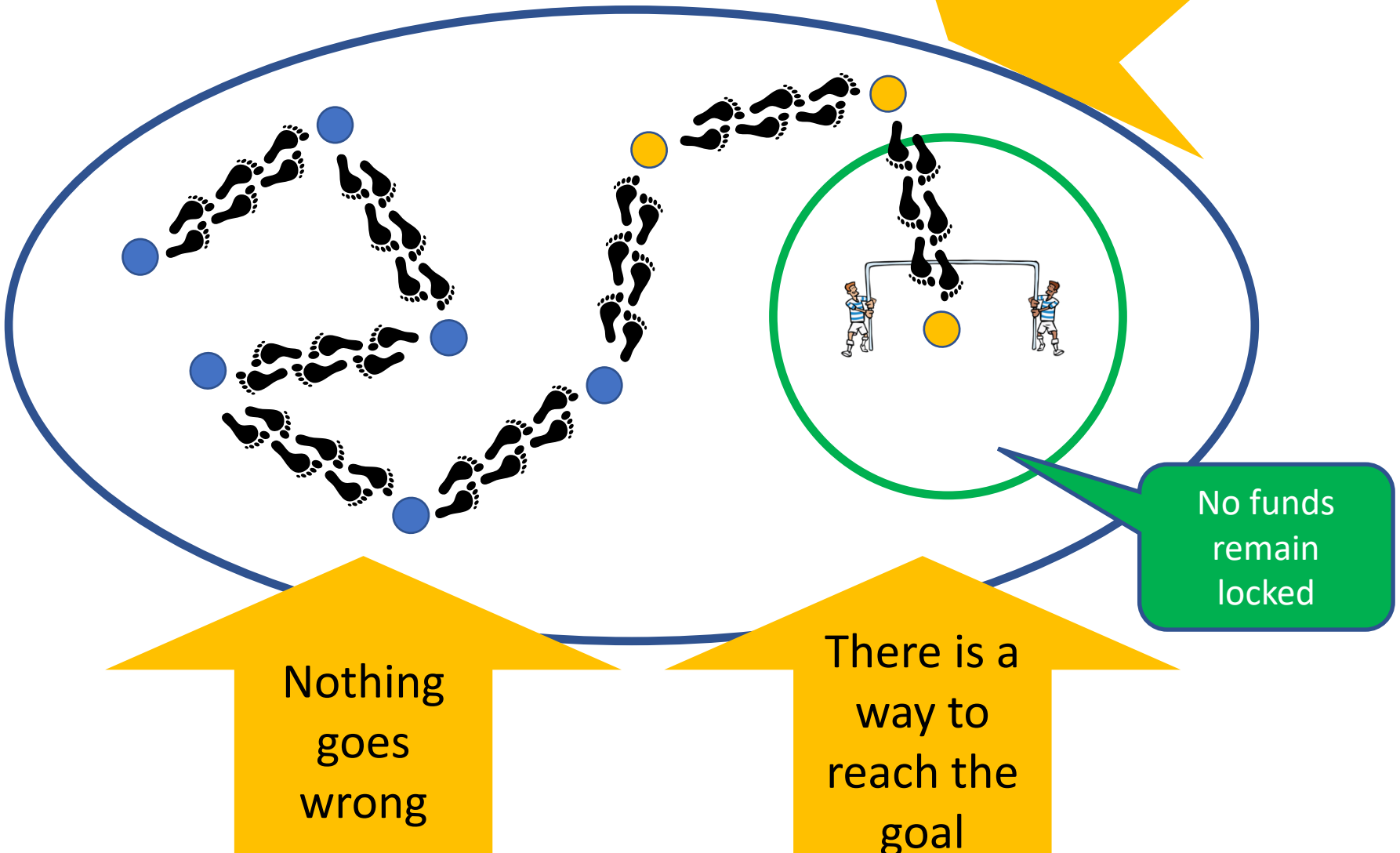
…InsufficientFunds…

Preconditions

Precondition for Guess:

- The wallet holds the game token

This isn’t enough…



Specifying a goal

```
noLockedFunds = do
  anyActions_
  assertModel "Locked funds should be zero" $
    isZero . lockedValue
```

After any action sequence at all…

…no funds are locked in contracts

```
 > quickCheck $ forAllDL noLockedFunds prop_Game
*** Failed! Falsified (after 1 test and 2
shrinks):
BadPrecondition
   [Do $ Lock (Wallet 1) "*******" 1]
   [Assert "Locked funds should be zero"]
   (GameModel {_gameValue = 1, _hasToken = Just (Wallet
1), _currentSecret = "*******"})
```

Specifying a strategy

```
noLockedFunds = do
  anyActions_
  w      <- forAllQ $ elementsQ wallets
  secret <- viewContractState currentSecret
  val    <- viewContractState gameValue
  action $ Guess w "" secret val
  assertModel "Locked funds should be zero" $
    isZero . lockedValue
```

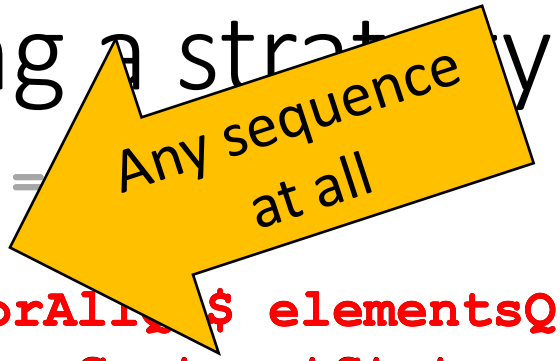
```
> quickCheck $ forAllDL noLockedFunds prop_Game
*** Failed! Falsified (after 1 test and 2 shrinks):
BadPrecondition
  [Witness (Wallet 1 :: Wallet)]
  [Action (Guess (Wallet 1) "" "" 0)]
  (GameModel {_gameValue = 0, _hasToken = Nothing,
_currentSecret = ""})
```

Preconditions

Precondition for Guess:

- The wallet holds the game token

Specifying a strategy




```
noLockedFunds =
  anyActions_
  w      <- forAll $ elementsQ wallets
  secret <- viewContractState currentSecret
  val    <- viewContractState gameValue
  action $ Guess w "" secret val
  assertModel "Locked funds should be zero" $
    isZero . lockedValue
```

```
> quickCheck $ forAllDL noLockedFunds prop_Game
*** Failed! Falsified (after 1 test and 2 shrinks):
BadPrecondition
  [Witness (Wallet 1 :: Wallet)]
  [Action (Guess (Wallet 1) "" "" 0)]
  (GameModel {_gameValue = 0, _hasToken = Nothing,
_currentSecret = ""})
```

Specifying a strategy

```
noLockedFunds = do
    anyActions_
    w      <- forAllQ $ elementsQ wallets
    secret <- viewContractState currentSecret
    val    <- viewContractState gameValue
    when (val > 0) $ do
      action $ Guess w "" secret val
    assertModel "Locked funds should be zero" $
      isZero . lockedValue
```

Of course not…

```
> quickCheck $ forAllDL noLockedFunds prop_Game
*** Failed! Falsified (after 1 test and 1 shrink):
BadPrecondition
   [Do $ Lock (Wallet 1) "*******" 1,
    Witness (Wallet 2 :: Wallet)]
   [Action (Guess (Wallet 2) "" "*******" 1)]
   (GameModel {_gameValue = 1, _hasToken = Just (Wallet 1),
_currentSecret = "*******"})
```

Specifying a strategy

```
noLockedFunds = do
    anyActions_
    w      <- forAllQ $ elementsQ wallets
    secret <- viewContractState currentSecret
    val    <- viewContractState gameValue
    when (val > 0) $ do
      action $ GiveToken w
      action $ Guess w "" secret val
    assertModel "Locked funds should be zero" $
      isZero . lockedValue
```

```
> quickCheck $ forAllDL noLockedFunds prop_Game
  +++ OK, passed 100 tests
```

Other strategies…

```
anyActions
action $ a…
anyActions
action $ b…
```

```
action a <|> action b
```

```
weight 2.5 a <|> action b
```

Limitations

- Testing via endpoints only
- Timing and race conditions
- Information leaks

Something good is always possible