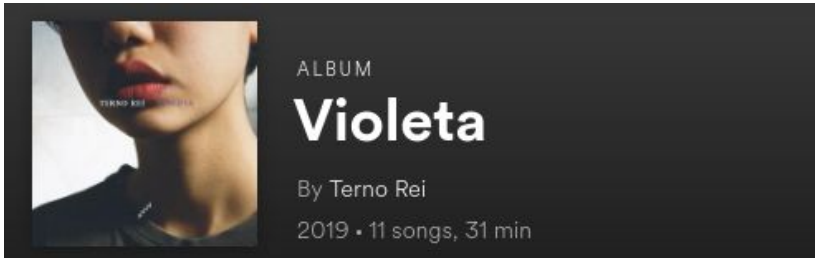


Live de Python #79

Testes unitários na prática

Ajude a Live de Python

apoia.se/livedepython

picPay: @livedepython

Obrigado!

apoia.se/livedepython

```
~/g/apoiadores ❯❯❯ python apoiadores.py
Vicente Marcal        Maria Boladona       Pedro Alves          Eliabe Silva
João Lugão            Sérgio Passos        Magno Malkut         David Reis
Dayham Soares         Fabiano Teichmann    Vergil Valverde      Edimar Fardim
Regis Santos          Wander Silva         Jonatas Oliveira     Fernando Furtado
Rennan Almeida        Renato Santos        Fábio Serrão         Jonatas Simões
Jucélio Silva         Wellington Camargo   Júlia Kastrup        Johnny Tardin
Paulo Tadei           Elias Soares         Jean Vetorello       Rodrigo Vaccari
Fabiano Silos         Tiago Cordeiro       Willian Gl           Andre Machado
William Oliveira      Gleison Oliveira     Bruno Guizi          Pablo Henrique
Nilo Alexandre Pereira
```

Python Pro

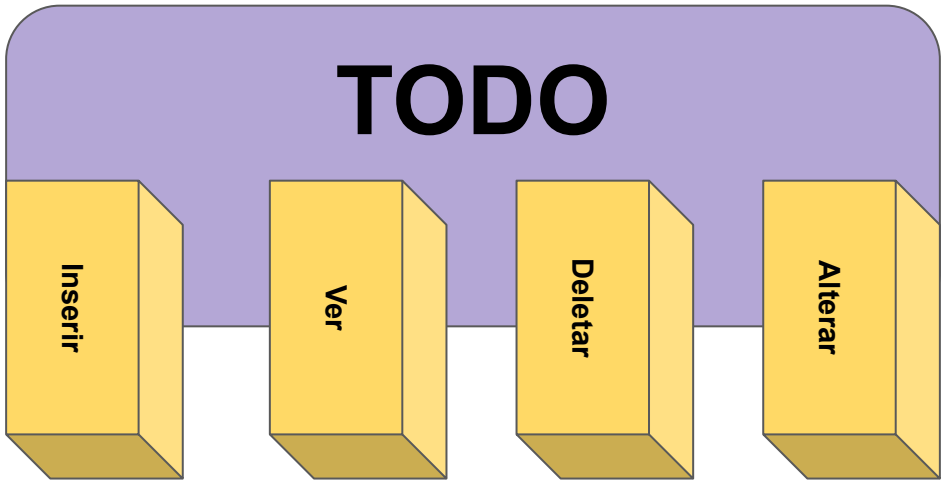
Python para Profissionais

Roteiro de hoje

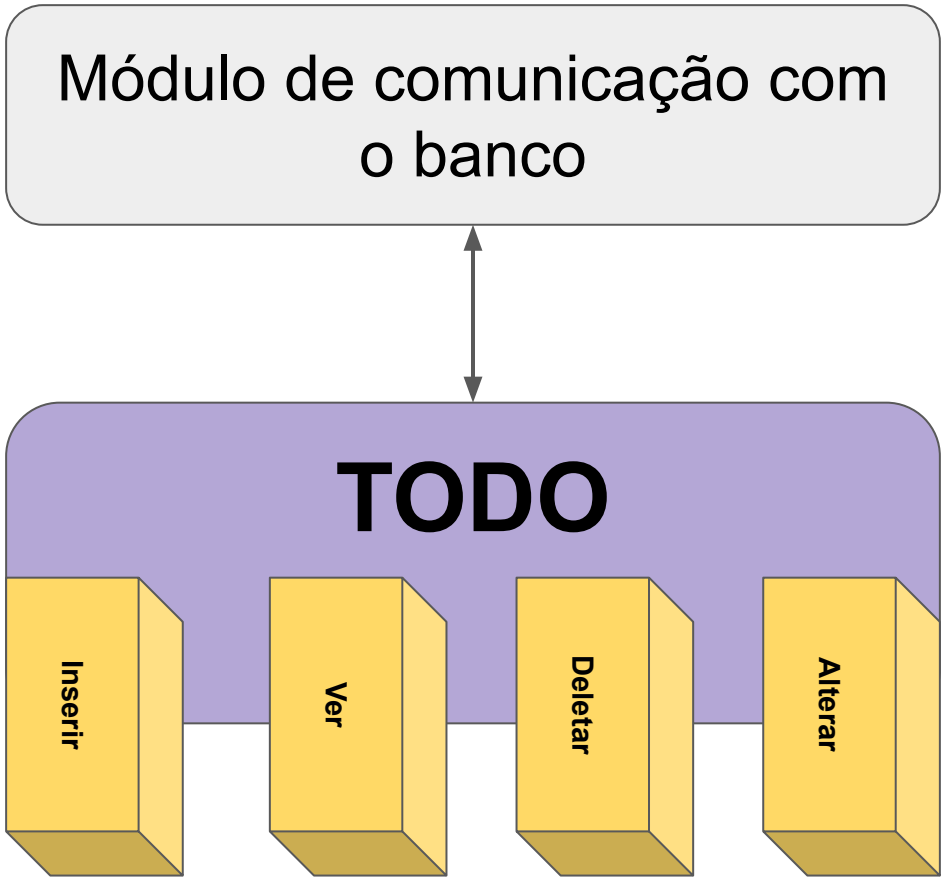
- Fazer um TODO list
- É sério
- é só isso
- ...

Como funciona uma todo list?

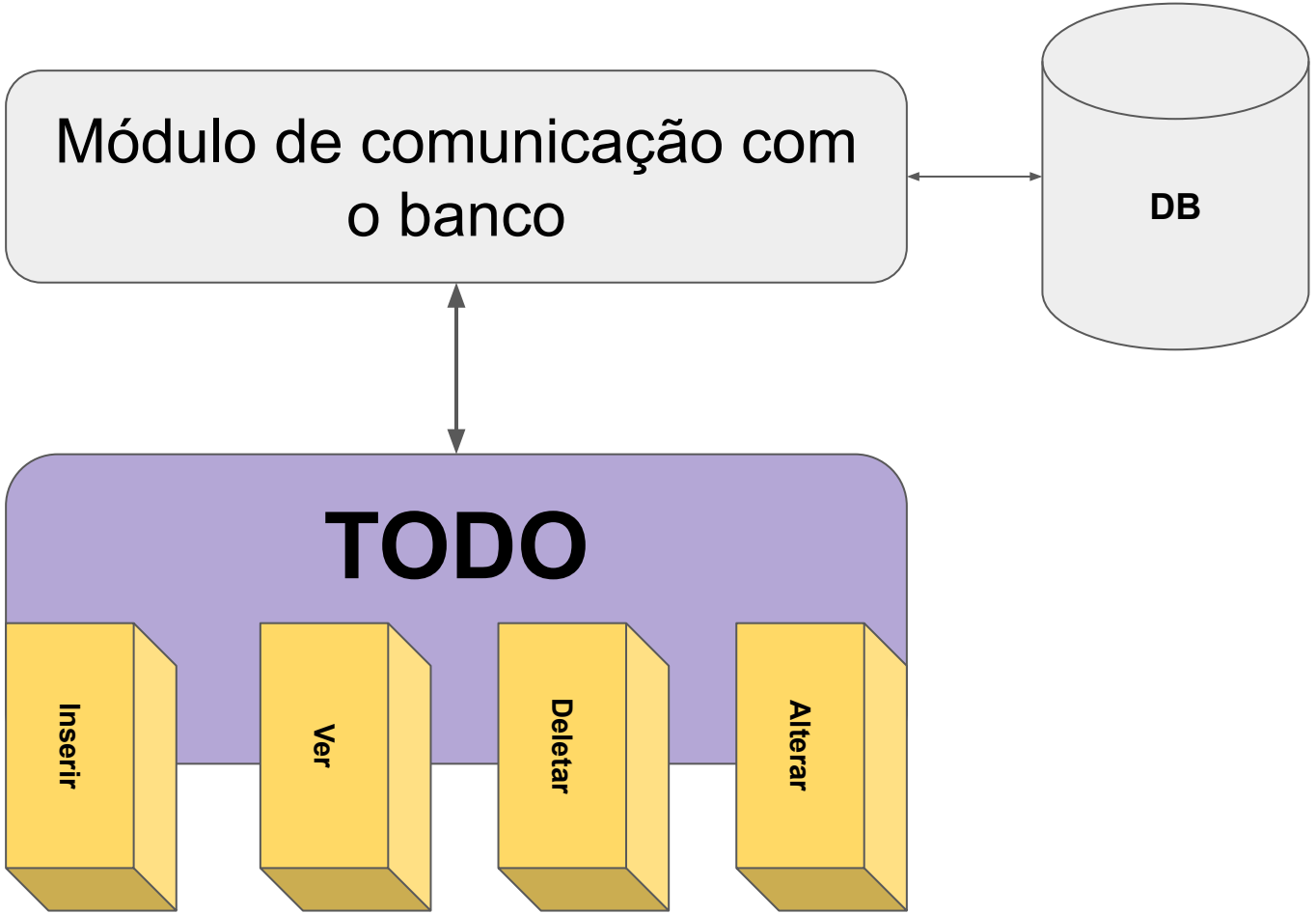
TODO list



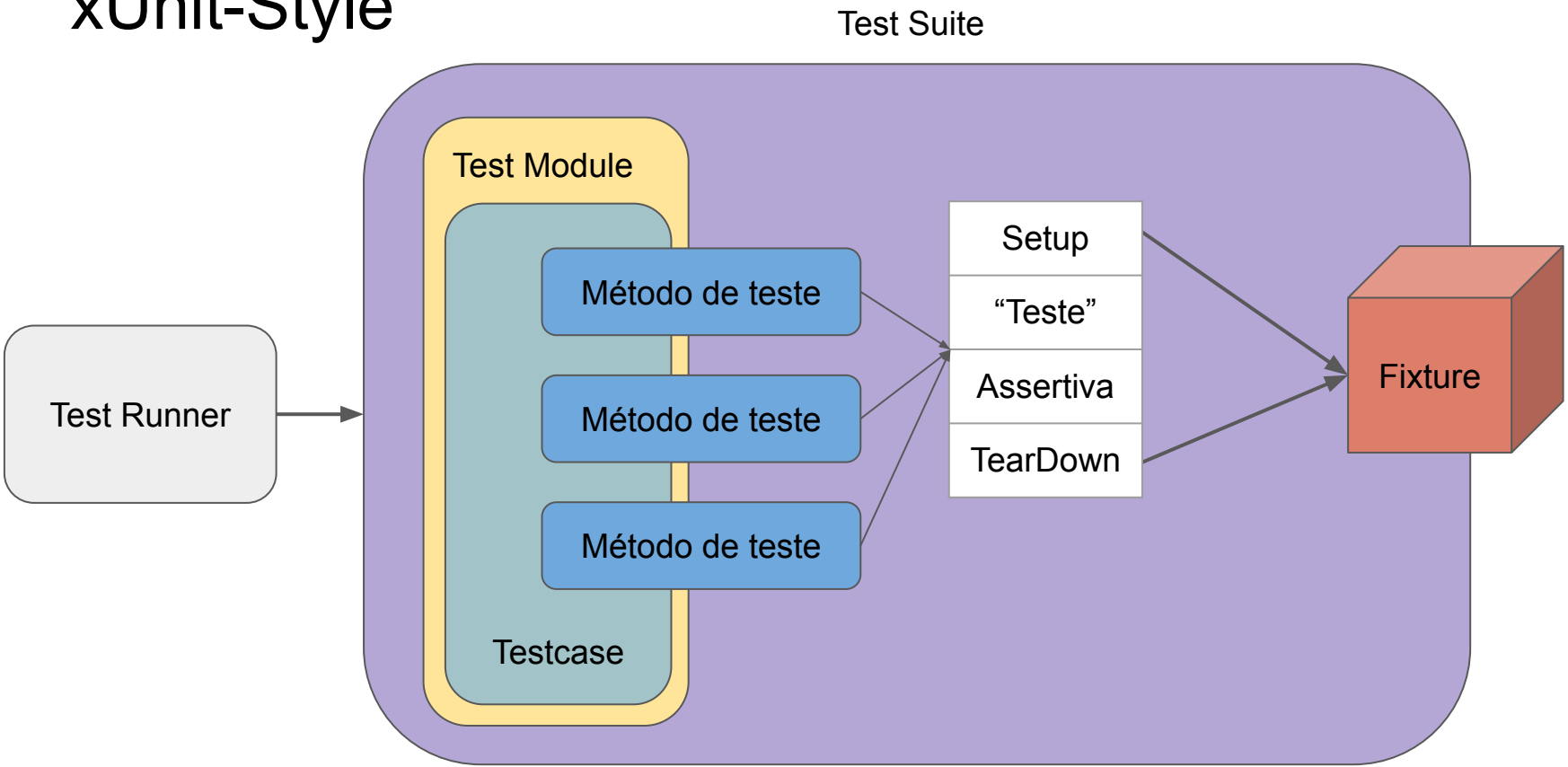
TODO list


Módulo de comunicação com o banco
TODO
Inserir
Ver
Deletar
Alterar

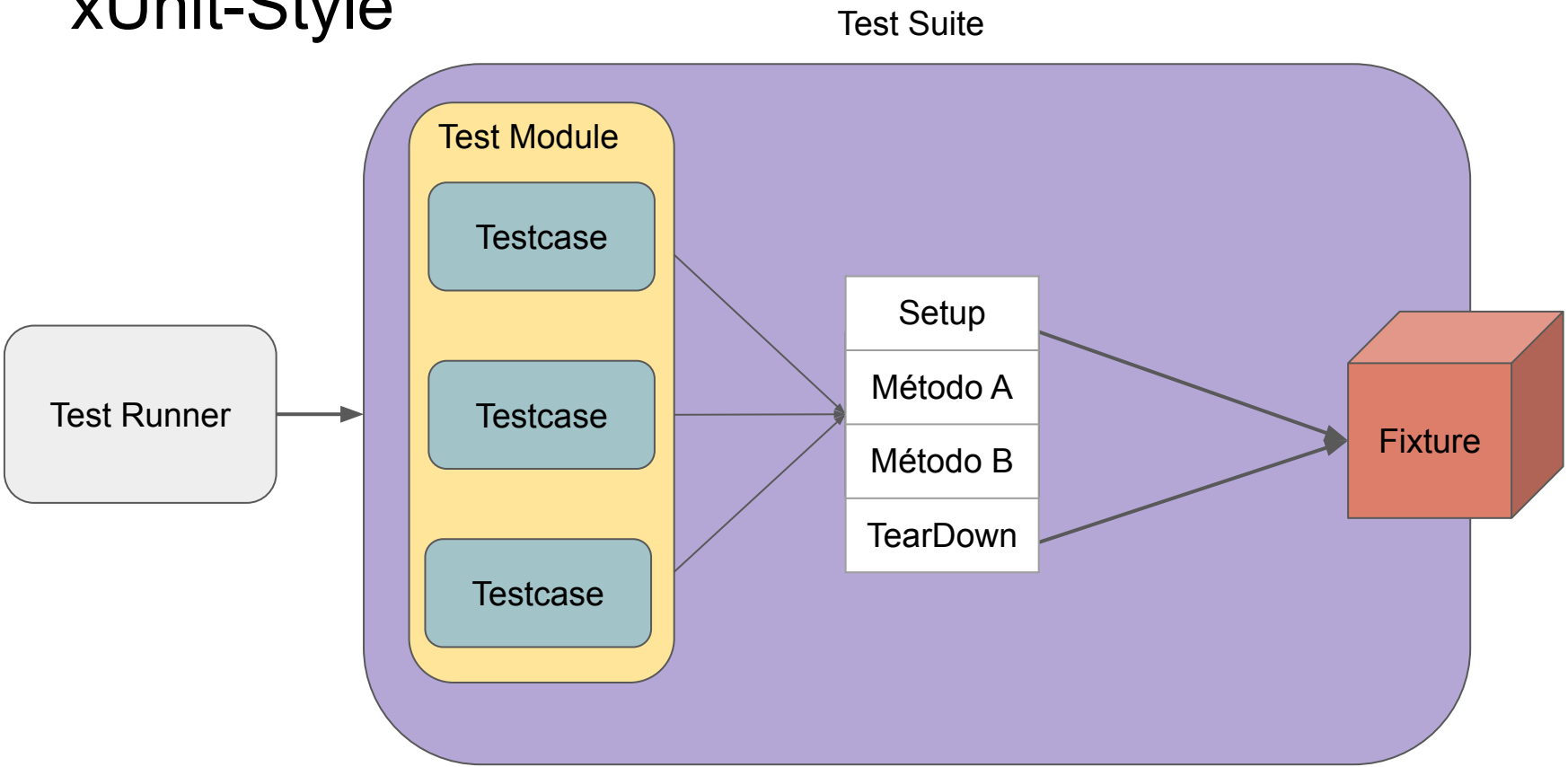
TODO list


Módulo de comunicação com o banco
DB
TODO
Inserir
Ver
Deletar
Alterar

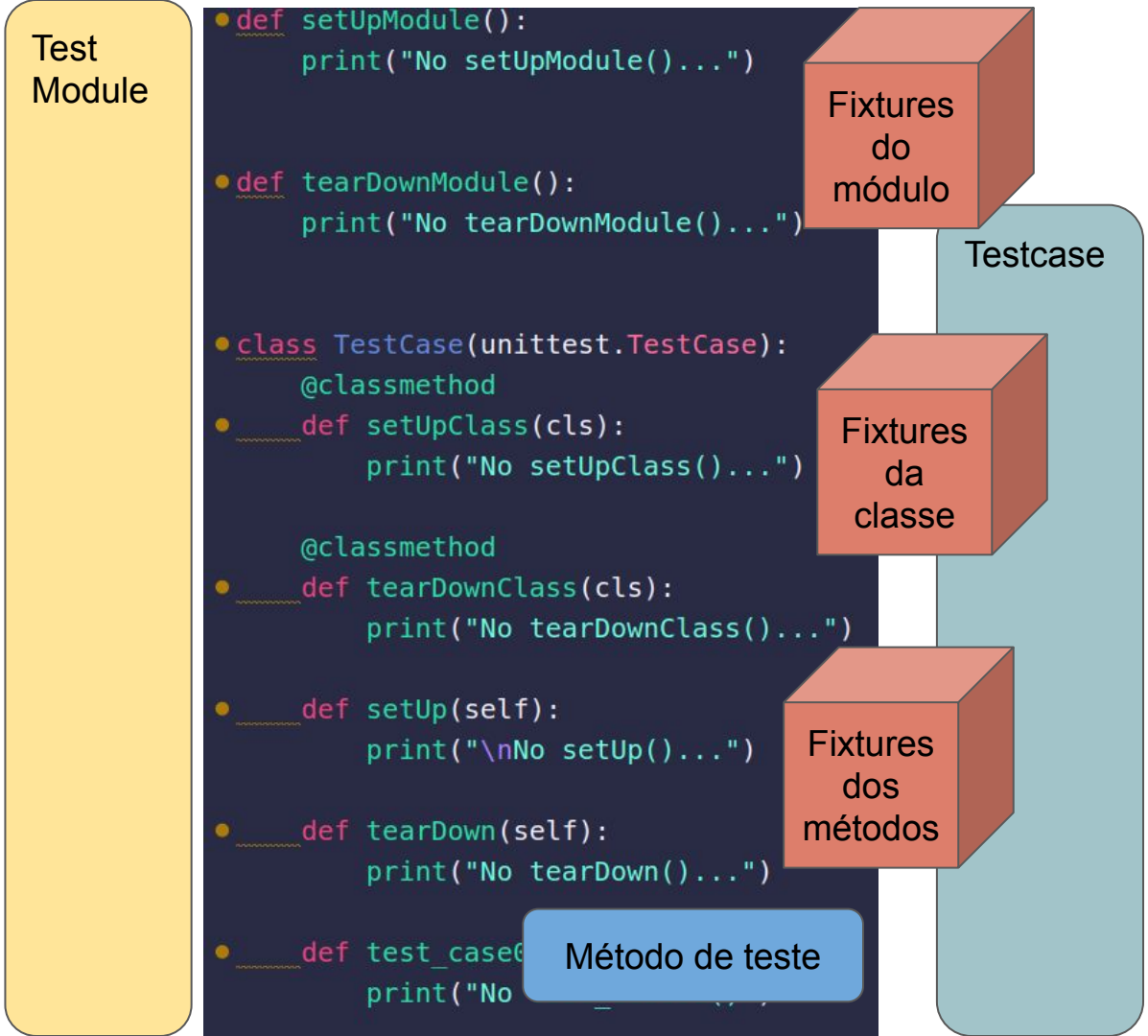
xUnit-Style


Test Suite
Test Module
Testcase
Método de teste
Método de teste
Método de teste
Setup
“Teste”
Assertiva
TearDown
Fixture
Test Runner

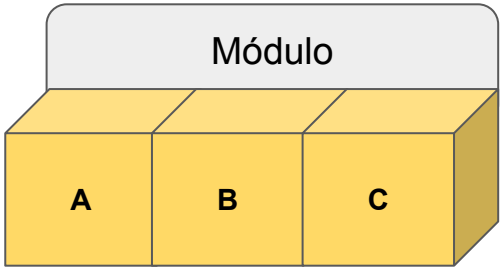
xUnit-Style


Test Suite
Test Module
Testcase
Testcase
Testcase
Test Runner
Setup
Método A
Método B
TearDown
Fixture

xUnit-Style


Test Module
def setUpModule():
print("No setUpModule()...")
def tearDownModule():
print("No tearDownModule()...")
Fixtures do módulo
Testcase
class TestCase(unittest.TestCase):
@classmethod
def setUpClass(cls):
print("No setUpClass()...")
Fixtures da classe
@classmethod
def tearDownClass(cls):
print("No tearDownClass()...")
def setUp(self):
print("\nNo setUp()...")
Fixtures dos métodos
def tearDown(self):
print("No tearDown()...")
def test_case
print("No
Método de teste

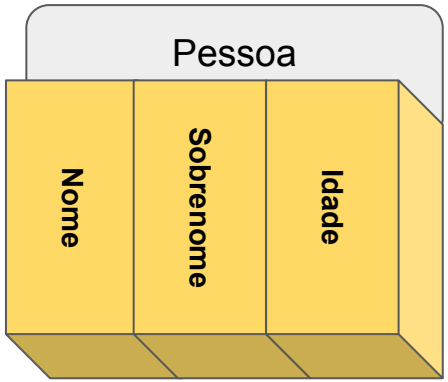
Dummy



Vamos supor que para que o módulo ser testado ele necessariamente precisa receber algo. Ou algo precisa existir para que ele funcione.

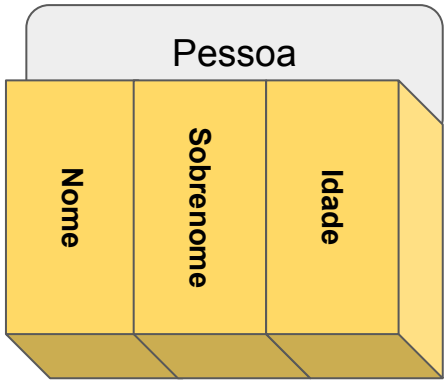
A ideia principal do Dummy não é que ele seja um objeto Nulo, puro e simplesmente, é que ele seja irrelevante para o SUT

Dummy




```python
class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade


Pessoa('Eduardo', 'Mendes')
```

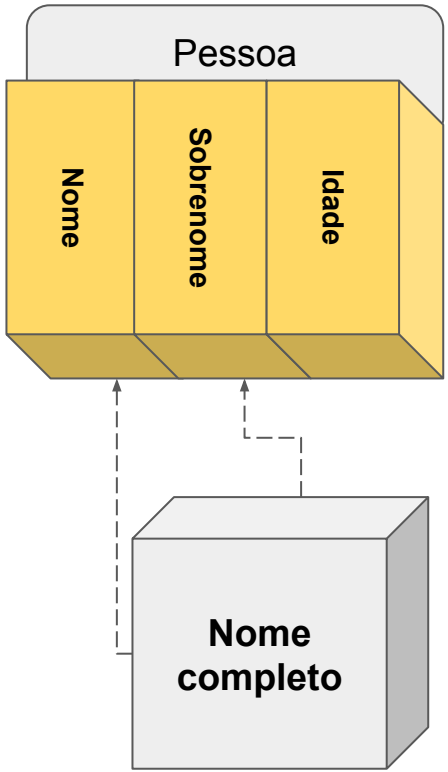
Dummy




```python
class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade


Pessoa('Eduardo', 'Mendes')
```

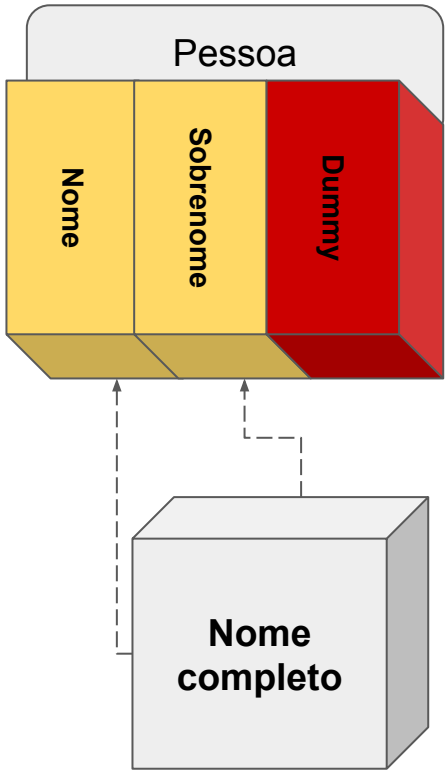
```
Traceback (most recent call last):
  File "dummy_exemplo.py", line 8, in <module>
    Pessoa('Eduardo', 'Mendes')
TypeError: __init__() missing 1 required positional argument: 'idade'
```

Dummy



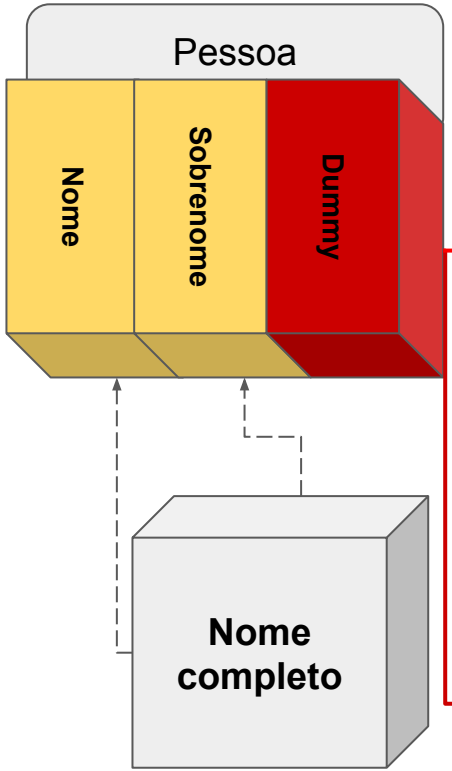

```python
class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade

    @property
    def nome_completo(self):
        return f'{self.nome} {self.sobrenome}'


Pessoa('Eduardo', 'Mendes', ?????)
```

Dummy



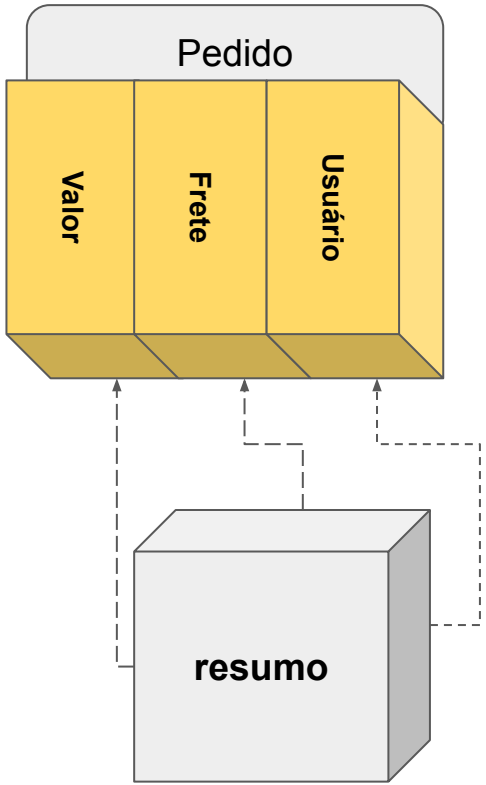

```python
from typing import NewType, Any

Dummy = NewType('Dummy', Any)

class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade

    @property
    def nome_completo(self):
        return f'{self.nome} {self.sobrenome}'


Pessoa('Eduardo', 'Mendes', Dummy(25))
```

Dummy




```python
from typing import NewType, Any

Dummy = NewType('Dummy', Any)

class Pessoa:
    def __init__(self, nome, sobrenome, idade):
        self.nome = nome
        self.sobrenome = sobrenome
        self.idade = idade

    @property
    def nome_completo(self):
        return f'{self.nome} {self.sobrenome}'


Pessoa('Eduardo', 'Mendes', Dummy(25))
```

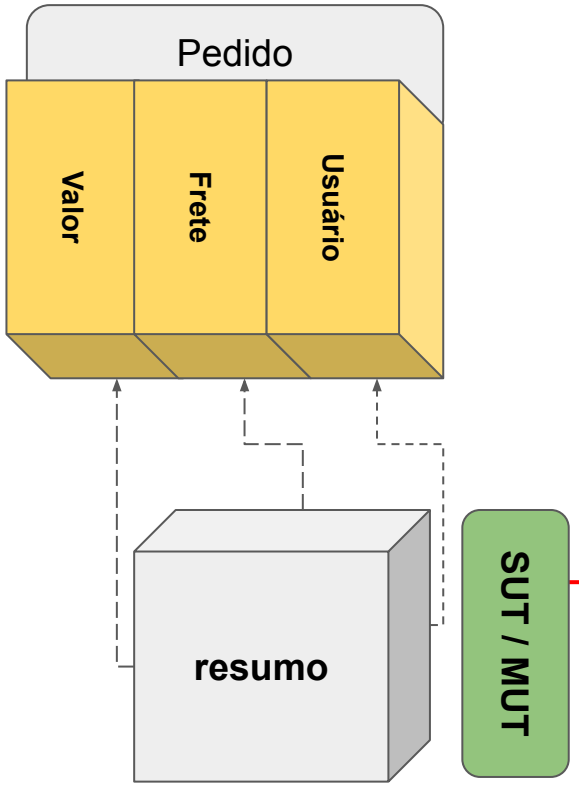
Fake



O fake é usado quando temos um módulo acoplado (DOC) e esse módulo é realmente chamado de alguma maneira.

Então o Fake tem que se comportar como o objeto, com a mesma API, na chamada do SUT.

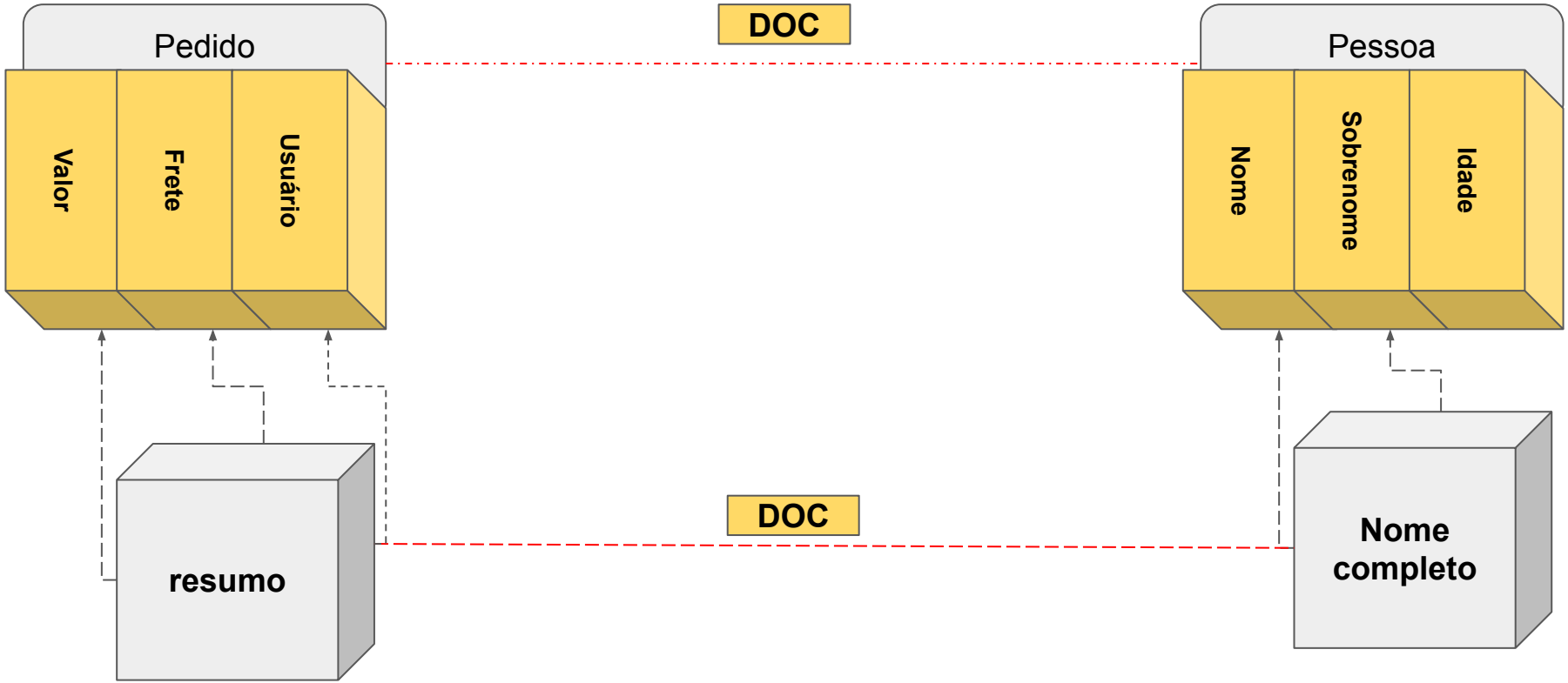
Fake

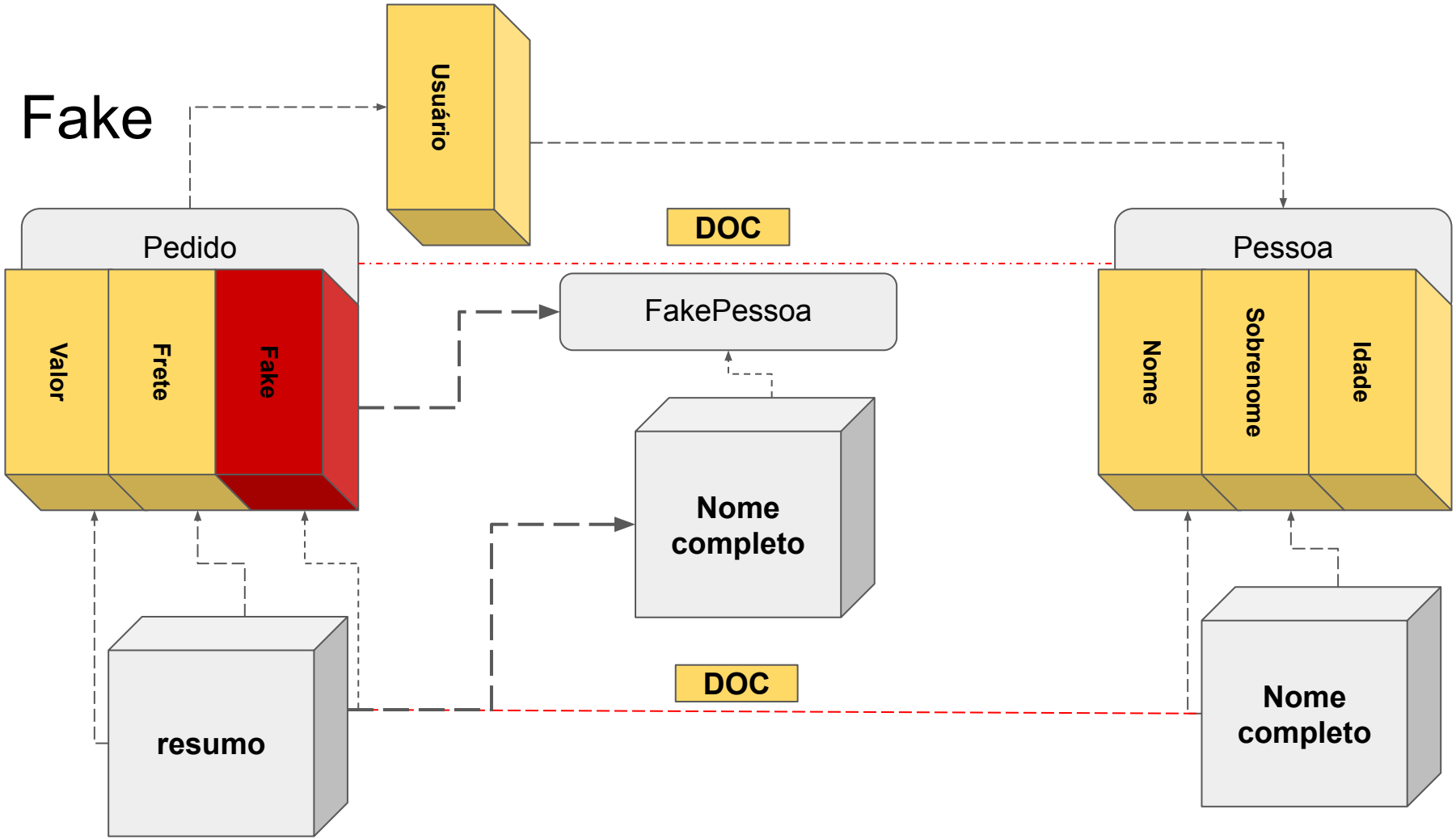



```python
class Pedido:
    def __init__(self, valor, frete, usuario):
        self.valor = valor
        self.frete = frete
        self.usuario = usuario

    @property
    def resumo(self):
        """Informações gerais sobre o pedido."""
        return f'''
        Pedido por: {self.usuario.nome_completo}
        Valor: {self.valor}
        Frete: {self.frete}
        '''
```

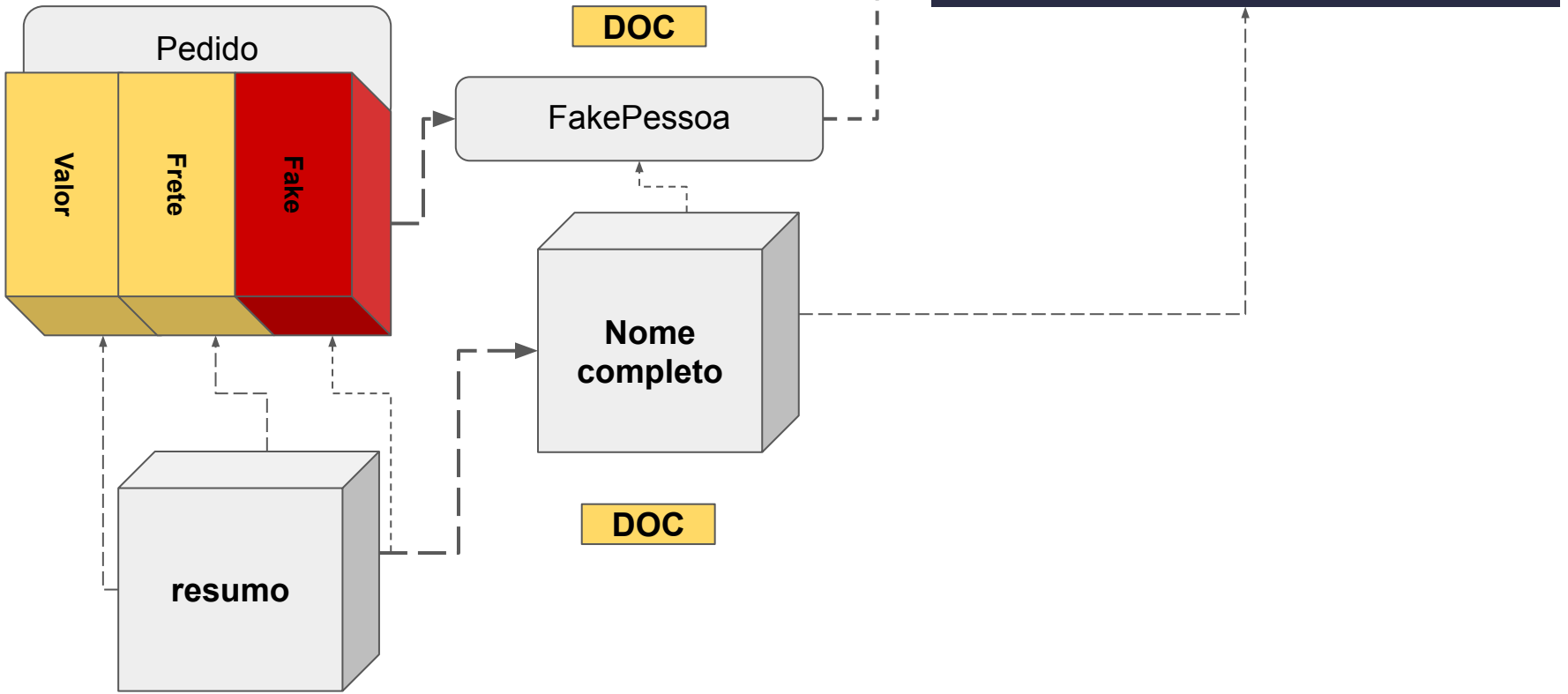
DOC

Fake


Pedido
Valor
Frete
Usuário
DOC
Pessoa
Nome
Sobrenome
Idade
resumo
DOC
Nome completo

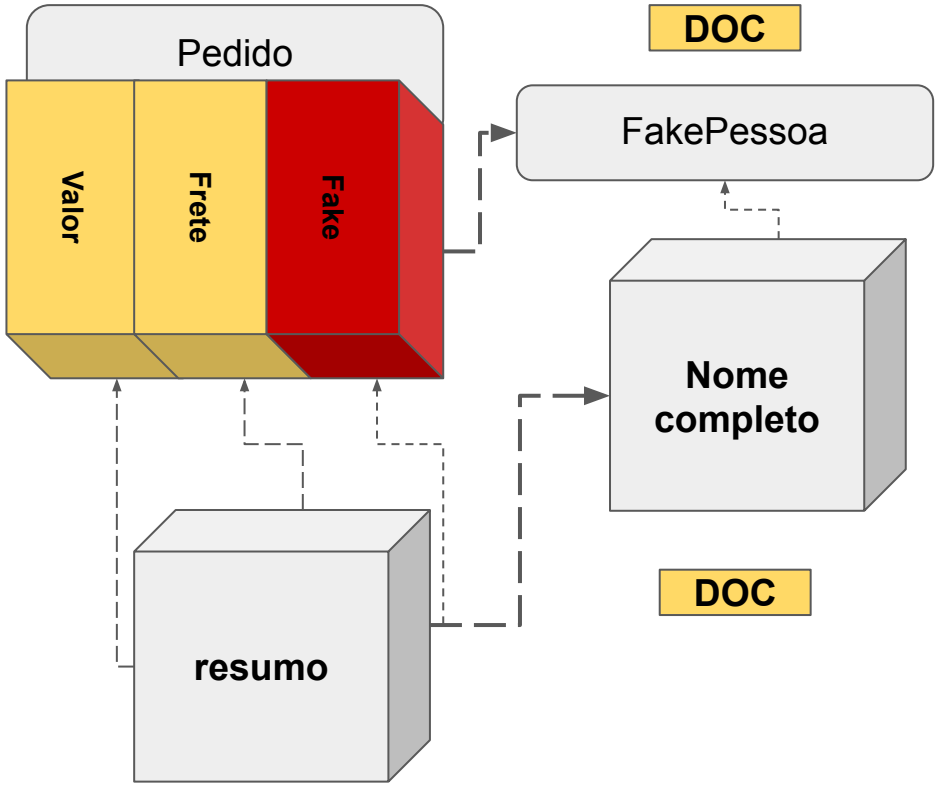

Fake
Usuário
Pedido
Valor
Frete
Fake
DOC
Pessoa
Nome
Sobrenome
Idade
FakePessoa
Nome completo
resumo
DOC
Nome completo

Fake



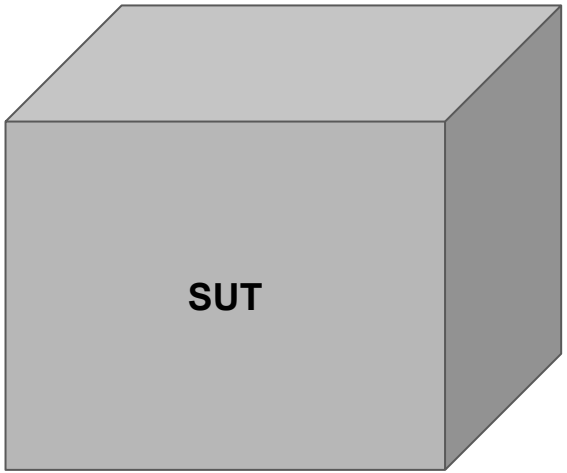

```python
class FakePessoa:
    @property
    def nome_completo(self):
        return 'Eduardo Mendes'
```

Fake




```python
class Pedido:
    def __init__(self, valor, frete, usuario):
        self.valor = valor
        self.frete = frete
        self.usuario = usuario

    @property
    def resumo(self):
        """Informações gerais sobre o pedido."""
        return f'''
        Pedido por: {self.usuario.nome_completo}
        Valor: {self.valor}
        Frete: {self.frete}
        '''


class FakePessoa:
    @property
    def nome_completo(self):
        return 'Eduardo Mendes'


Pedido(100.00, 13.00, FakePessoa()).resumo
```

Spy



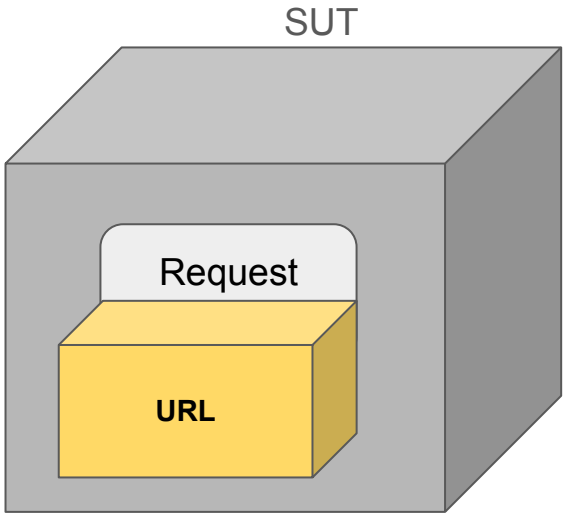
Spies são a KGB dos dublês. A ideia principal deles é saber quando o DOC foi chamado, com que valores ele foi chamado, quantas vezes foi chamado...

Você entendeu...

“Procedural Behavior Verification” é o termo técnico

A função deles é exercitar os inputs indiretos e validar se o DOC foi de fato executado

Spy



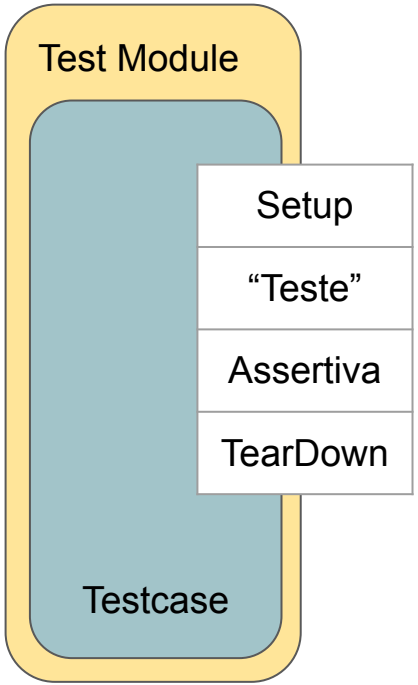
Spies são a KGB dos dublês. A ideia principal deles é saber quando o DOC foi chamado, com que valores ele foi chamado, quantas vezes foi chamado...

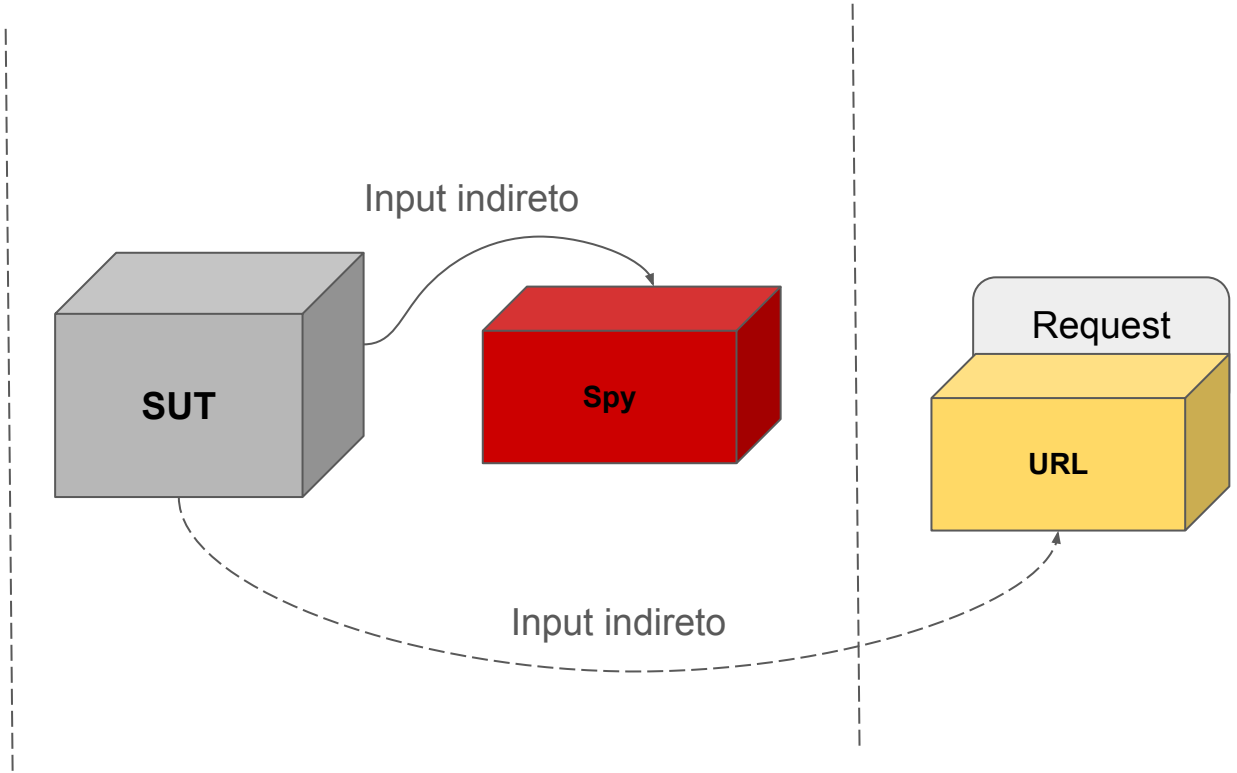
Você entendeu...

“Procedural Behavior Verification” é o termo técnico

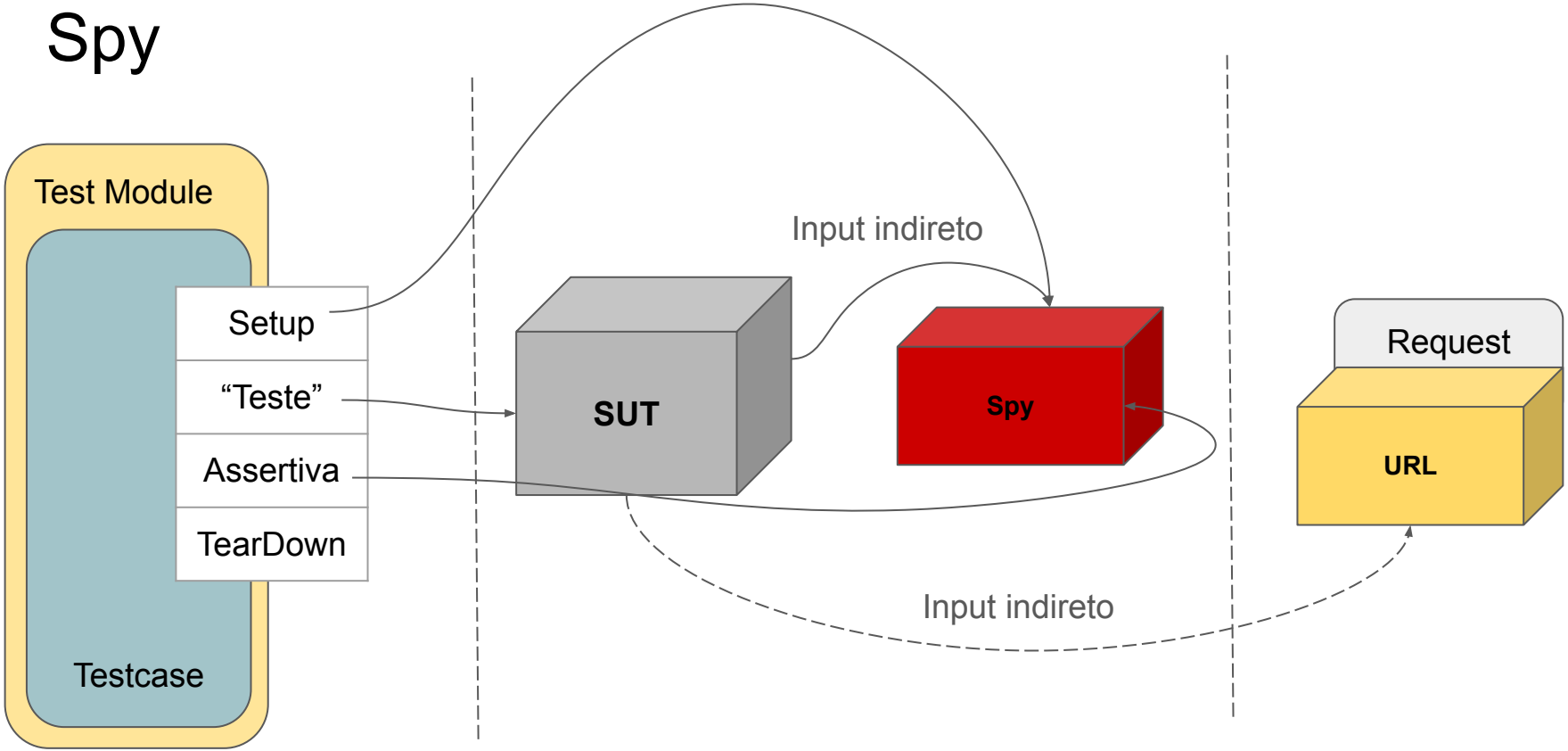
A função deles é exercitar os inputs indiretos e validar se o DOC foi de fato executado

Spy


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase


Input indireto
SUT
Spy
Request
URL
Input indireto

Spy


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Input indireto
Spy
Request
URL
Input indireto

Spy

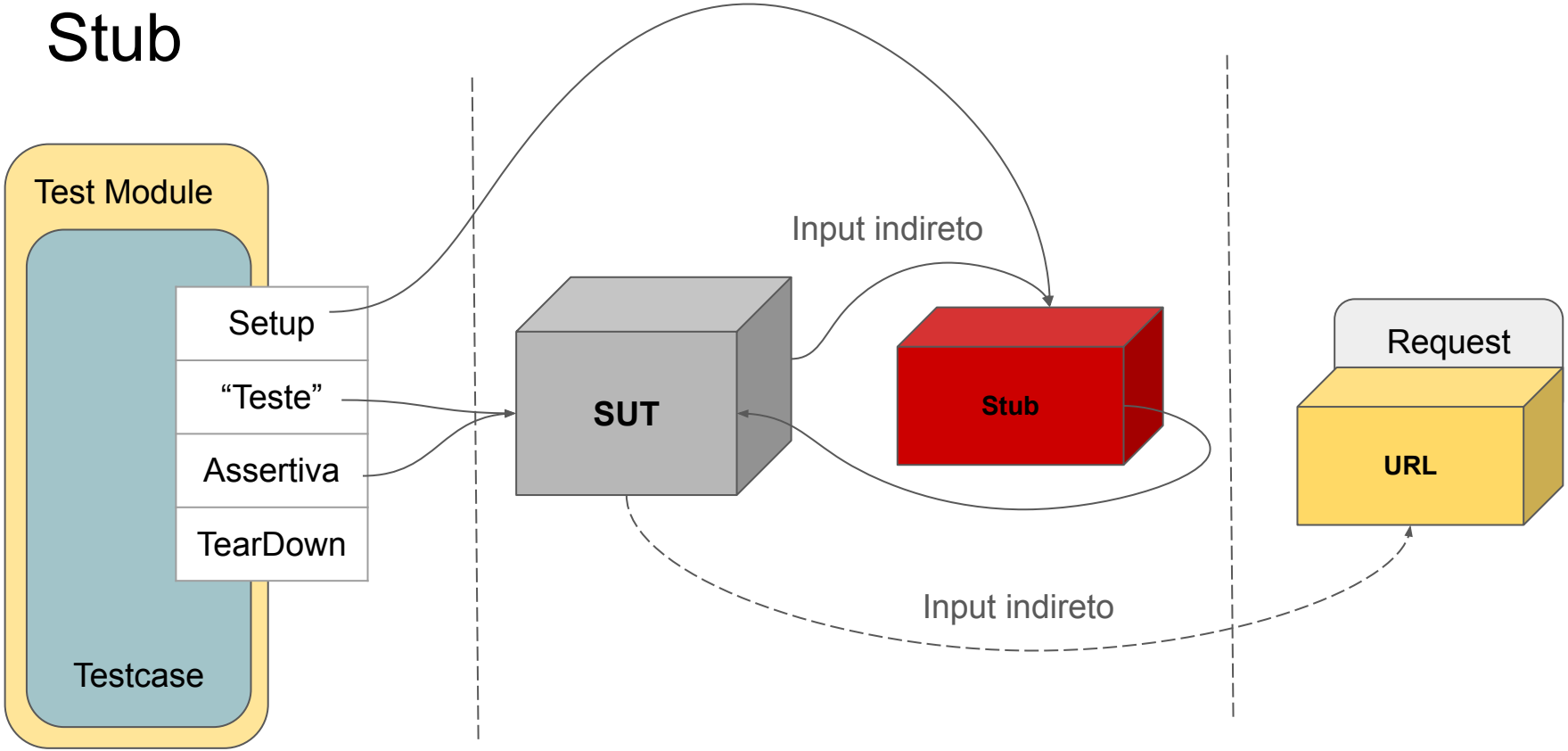
```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

```python
class TestPageContent(TestCase):
    def test_page_content_deve_ser_chamada_com_http(self):

        with mock.patch('acomplamento_exemplo.get') as spy:
            page_content('bababa', False)

        spy.assert_called_with('http://bababa', None)
```

Stub

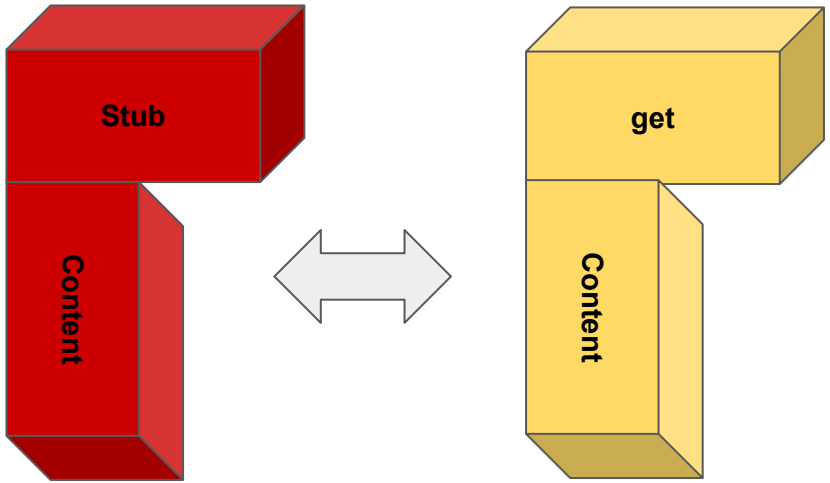
Stub é um objeto que armazena dados predefinidos e os utiliza para atender chamadas durante os testes. Ele é usado quando não podemos ou não queremos envolver objetos que respondam com dados reais ou que tenham efeitos colaterais indesejáveis.

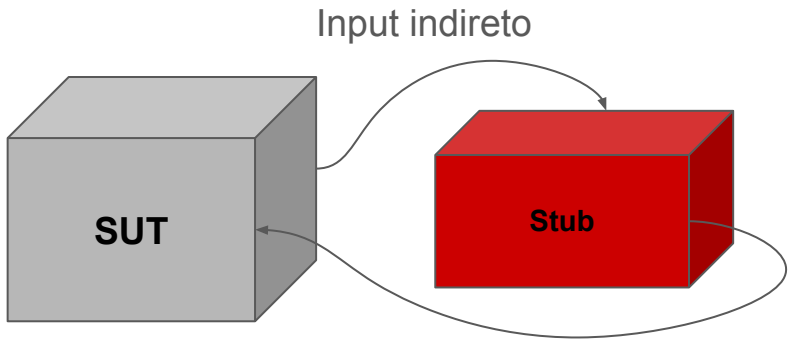
Stub


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Input indireto
Stub
Request
URL
Input indireto

Stub

```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

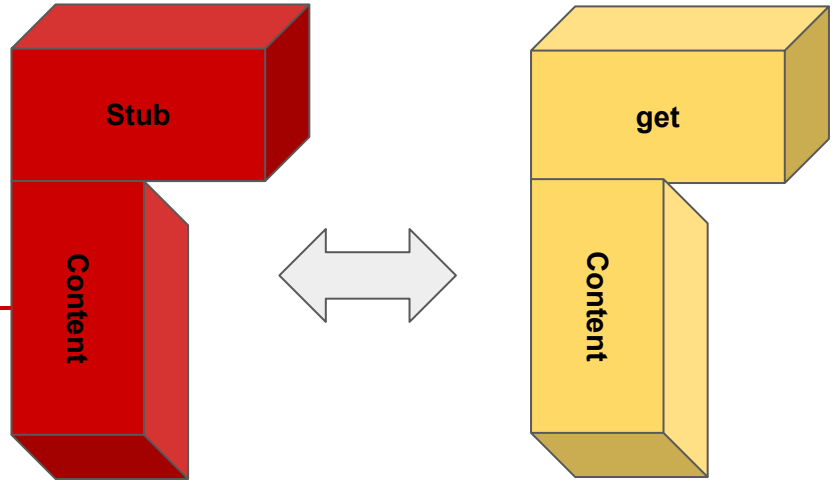




Stub

```python
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

```python
class StubGet:
    @property
    def content(self):
        return b'<html> ... </html>'
```



Stub

```
class StubGet:
    @property
    def content(self):
        return b'<html> ... </html>'
```

```
from requests import get


def page_content(url: str, ssl: bool = False, *, params=None) -> str:
    prefix = 'https' if ssl else 'http'
    return get(f'{prefix}://{url}', params).content.decode()
```

```
class TestPageContent(TestCase):
    def test_page_content_deve_ser_chamada_com_http(self):
        esperado = '<html> ... </html>'

        with mock.patch('acomplamento_exemplo.get', return_value=StubGet()):
            result = page_content('bababa', False)

        self.assertEqual(esperado, result)
```

Mock

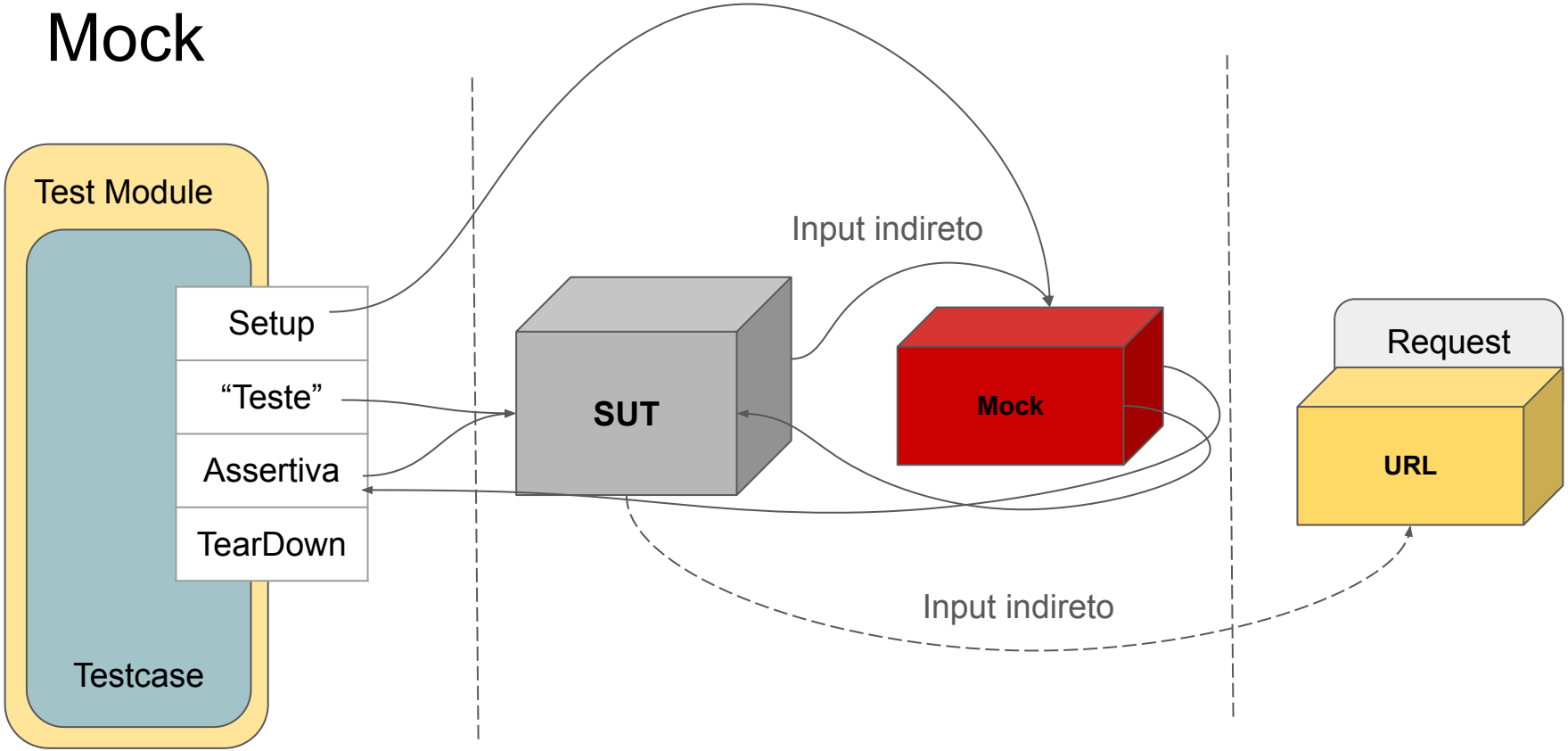
Os mocks são uma loucura que só. Sério, eles são de mais. Ninguém pode dizer o contrário.

Vamos esquecer da teoria por um pequeno minuto.

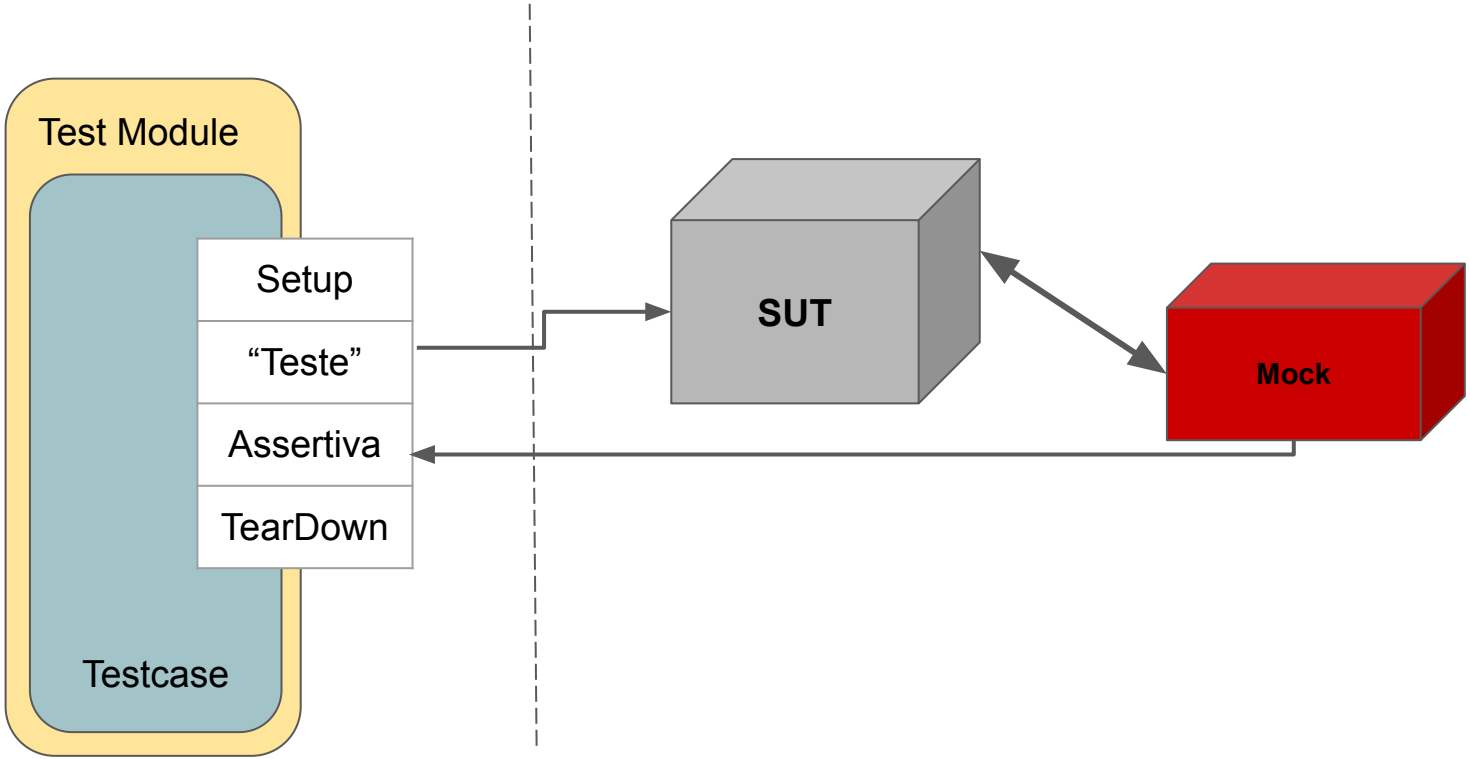
Mocks controlam o fluxo, se colocam no lugar de outras coisas, e ainda pode checar seus resultados.

Ele é um Stub e um Spy ao mesmo tempo.

Mock


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Input indireto
Mock
Request
URL
Input indireto

Mock


Test Module
Setup
“Teste”
Assertiva
TearDown
Testcase
SUT
Mock

Mock

```python
class FakeGet:
    @property
    def content(self):
        return b'<html> ... </html>'


class TestPageContent(TestCase):
    def test_page_content_deve_ser_chamada_com_http(self):
        esperado = '<html> ... </html>'

        with patch(
            'acomplamento_exemplo.get', return_value=FakeGet()
        ) as mocked:
            result = page_content('bababa', False)

        self.assertEqual(esperado, result)
        mocked.assert_called_with('http://bababa', None)
```