Working with C++ Legacy Code

Dror Helper | www.helpercode.com | @dhelper

About.ME

Consultant & Software Architect

Developing software since 2002
Clean Coder & Test Driven Developer

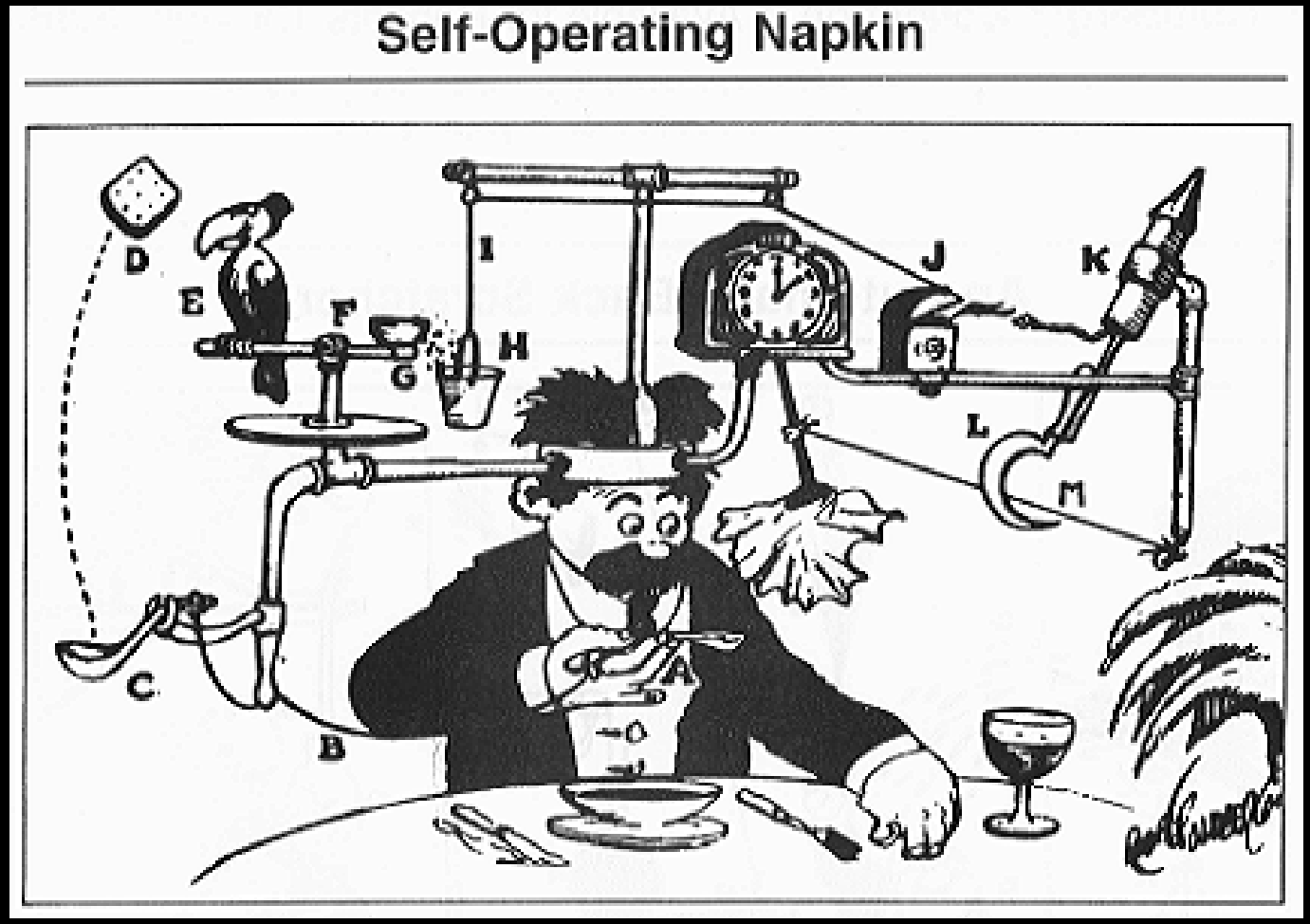
Pluralsight author
https://www.pluralsight.com/authors/dror-helper

B: http://helpercode.com
T: @dhelper

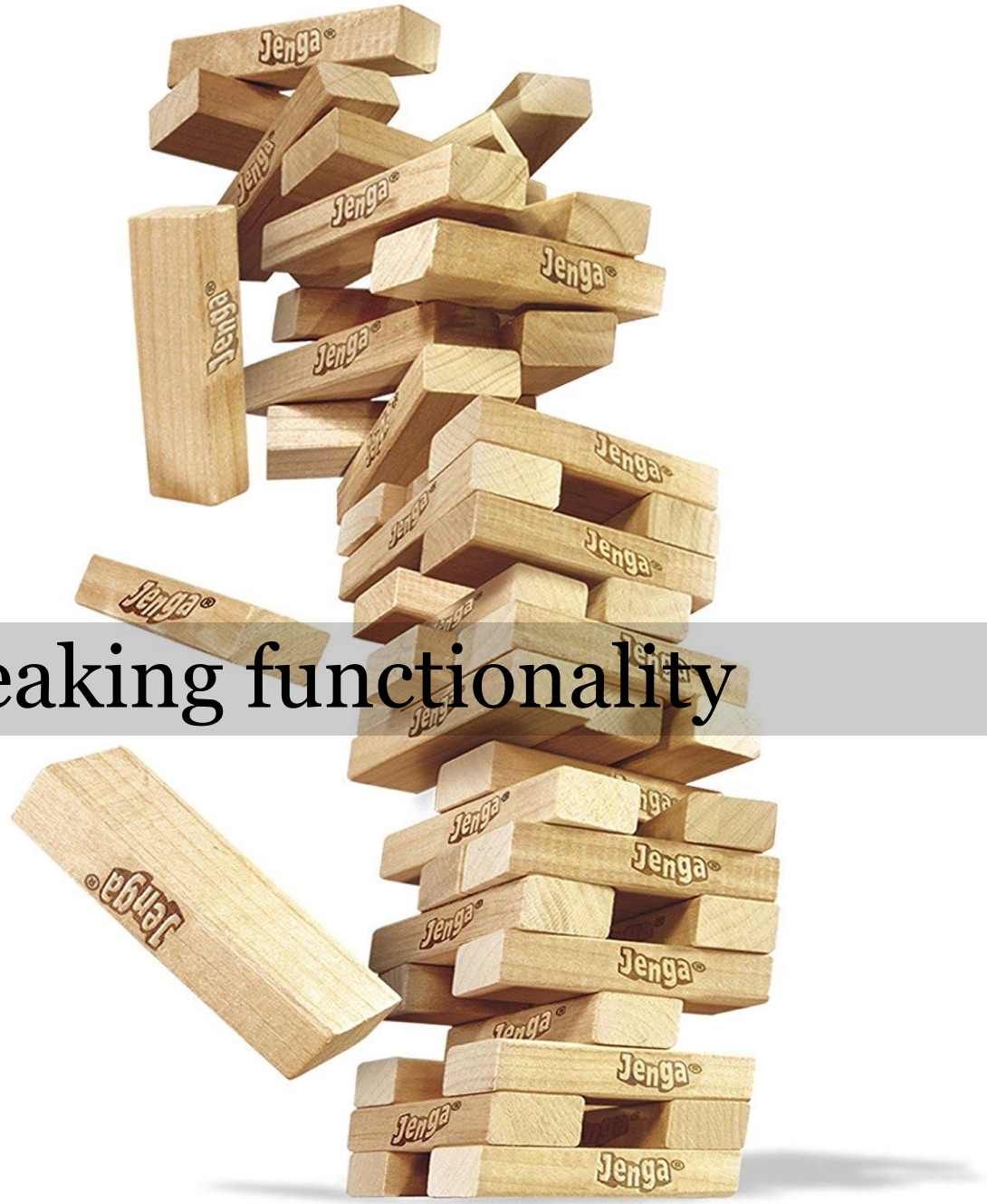
¯_(ツ)_/¯

What is “Legacy Code”?

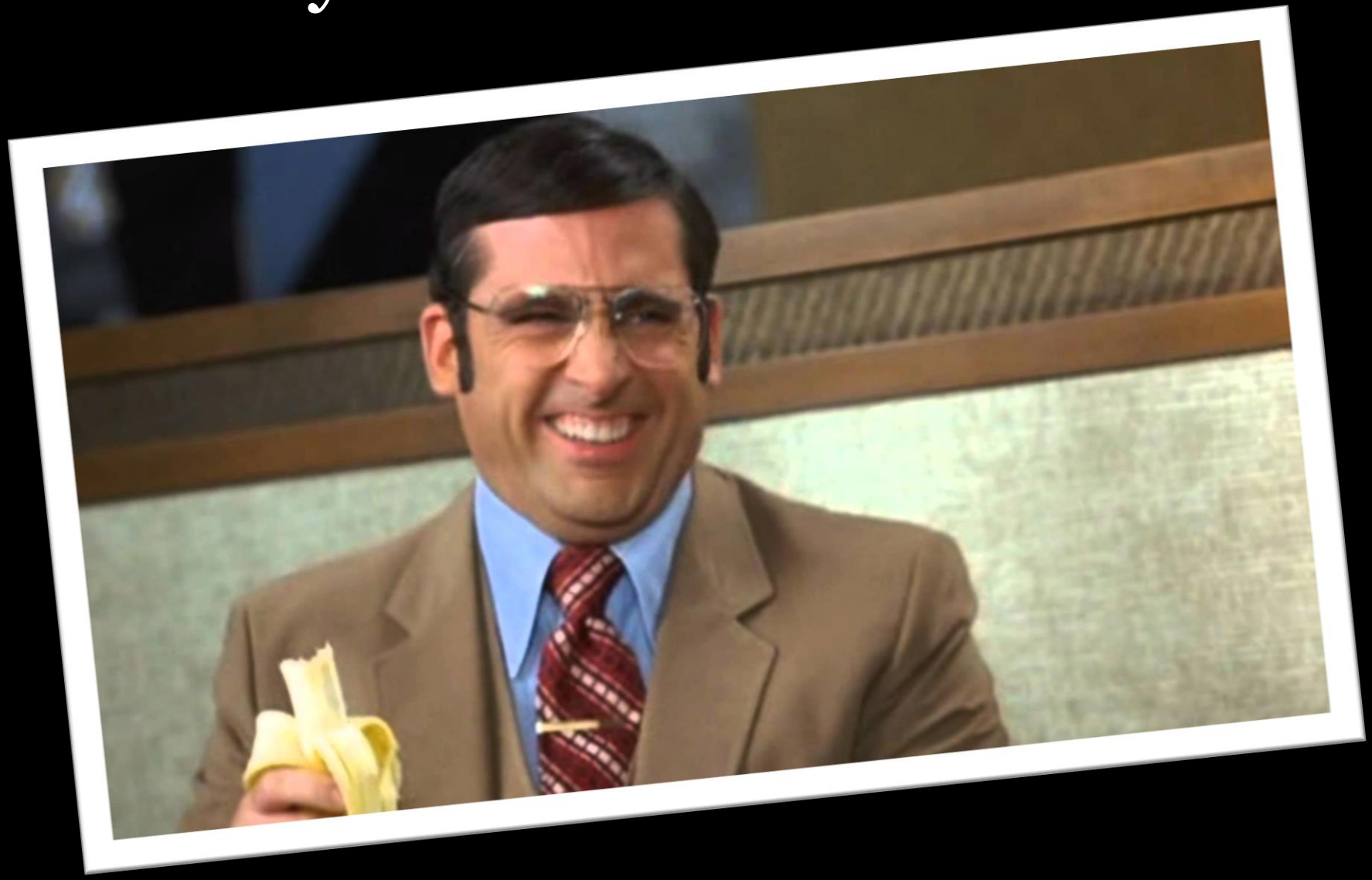
No longer engineered – continuedly patched



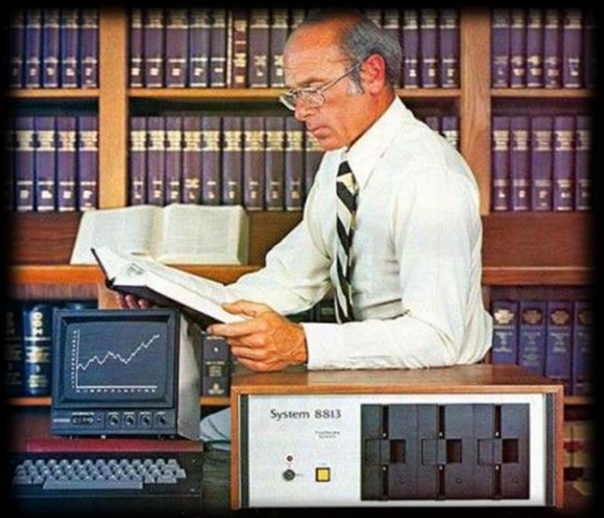
Difficult to change without breaking functionality

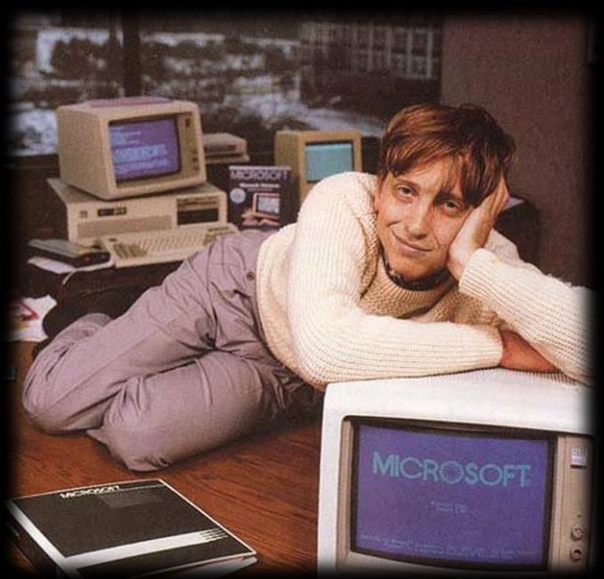


Written by someone else



Legacy code has users!





The good news and bad news

- 👎 There's a lot of legacy C++ Code
- 👎 There's a good chance you're doing it
- 👎 C++ makes it easy to make critical mistakes

- 👍 C++ is a powerful language

Gaining control over legacy code

To stop fearing legacy code:

1. Learn what the code does
2. Make sure it keeps doing it!
3. Iterative improvement via Refactoring

This is a unit test (GTest)

```
TEST(HolyGrailTests, WhoWeAreTest)
{
      Knight knight;

      ASSERT_EQ(“Ni!”, knight.Say());
}
```

A quick reality check

- Code have dependencies
- Refactoring == code change
- Code change == breaking functionality (78.3%)
- Breaking functionality → go home == false

Sensing and Separation

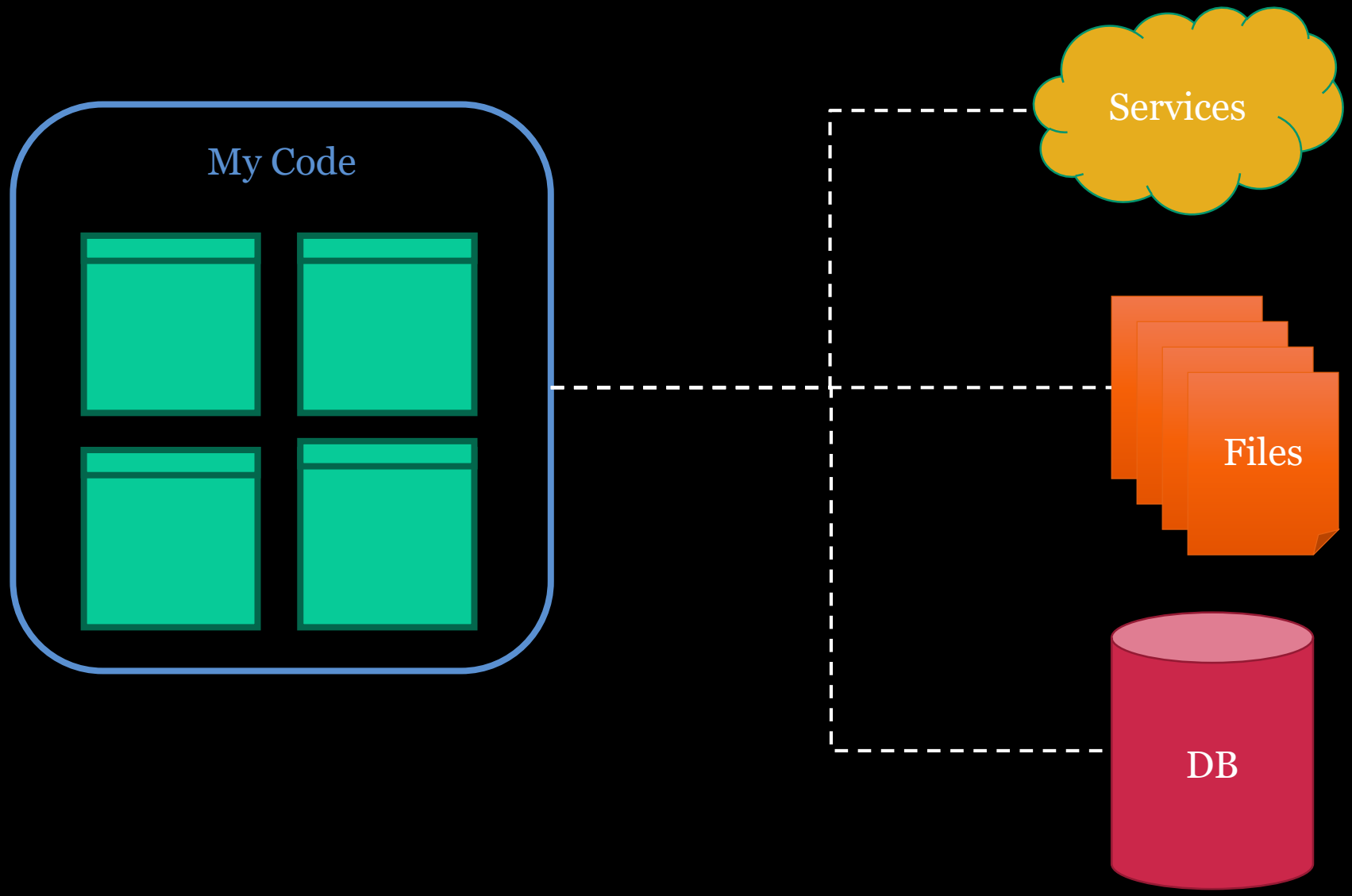
```
public void SendEmailTest()
{
    auto sut = new ...
    User user;
    sut->SendEmailToUser(user);

    // Sensing problem
    ASSERT???
}
```

```
void SendEmailToUser(User user)
{
    EmailClient client;

    // Separation problem
    client.Send(user.Email, ...);

    // Important business logic
    ...
}
```

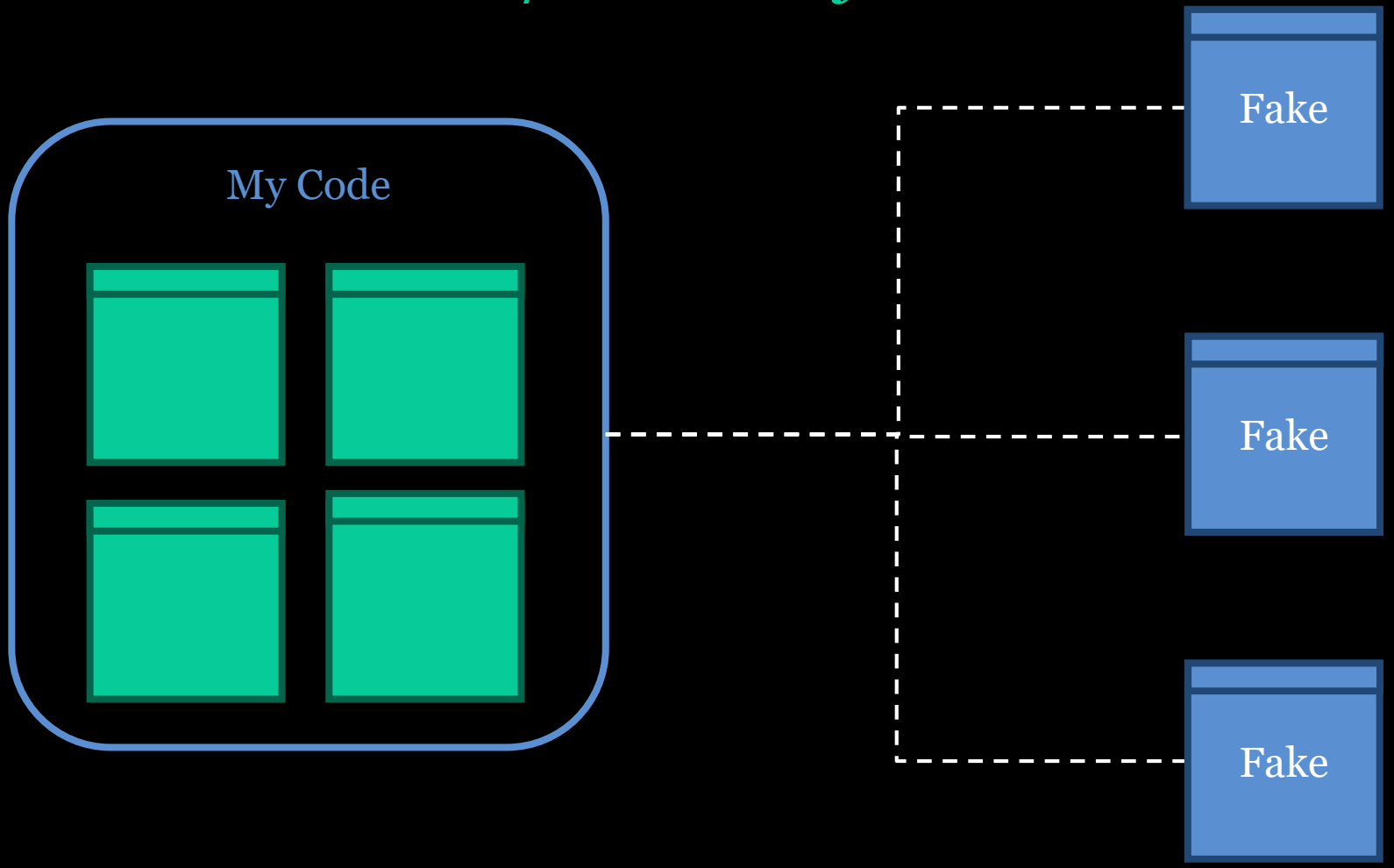
“My focus is to forget the pain of life. Forget the pain, mock the pain, reduce it. And laugh.”

Jim Carrey

The problem - dependencies



The solution – fake/mock objects



Fake objects using Google Mock

```
class FakeRestApiClient : public RestApiClient
{
public:
    MOCK_METHOD2(HttpPost, void(string&, string&));
    MOCK_METHOD1(HttpGet, string(string&));
    MOCK_METHOD2(HttpPut, void(string&, string&));
    MOCK_METHOD1(HttpDelete, void(string&));
};
```

Injecting fake objects into existing code

Non virtual methods

Hard to inherit classes

Static method calls

Singletons

Internally instantiated

Heavy classes

Not all dependency are created equal

Some are harder/impossible to fake

Faking Singletons

```
class MySingleton {

        static std::once_flag onceFlag;
        static MySingleton* instance;

public:
        static MySingleton* GetInstance() {
                std::call_once(onceFlag, [] {
                        instance = new MySingleton();
                });

                return instance;
        }

private:

        MySingleton(){}

}
```

Friend is your friend

```
class FakeSingleton;


class MySingleton {
{
        . . .
private:
        friend FakeSingleton;
};
```

Problem: un-fakeable methods

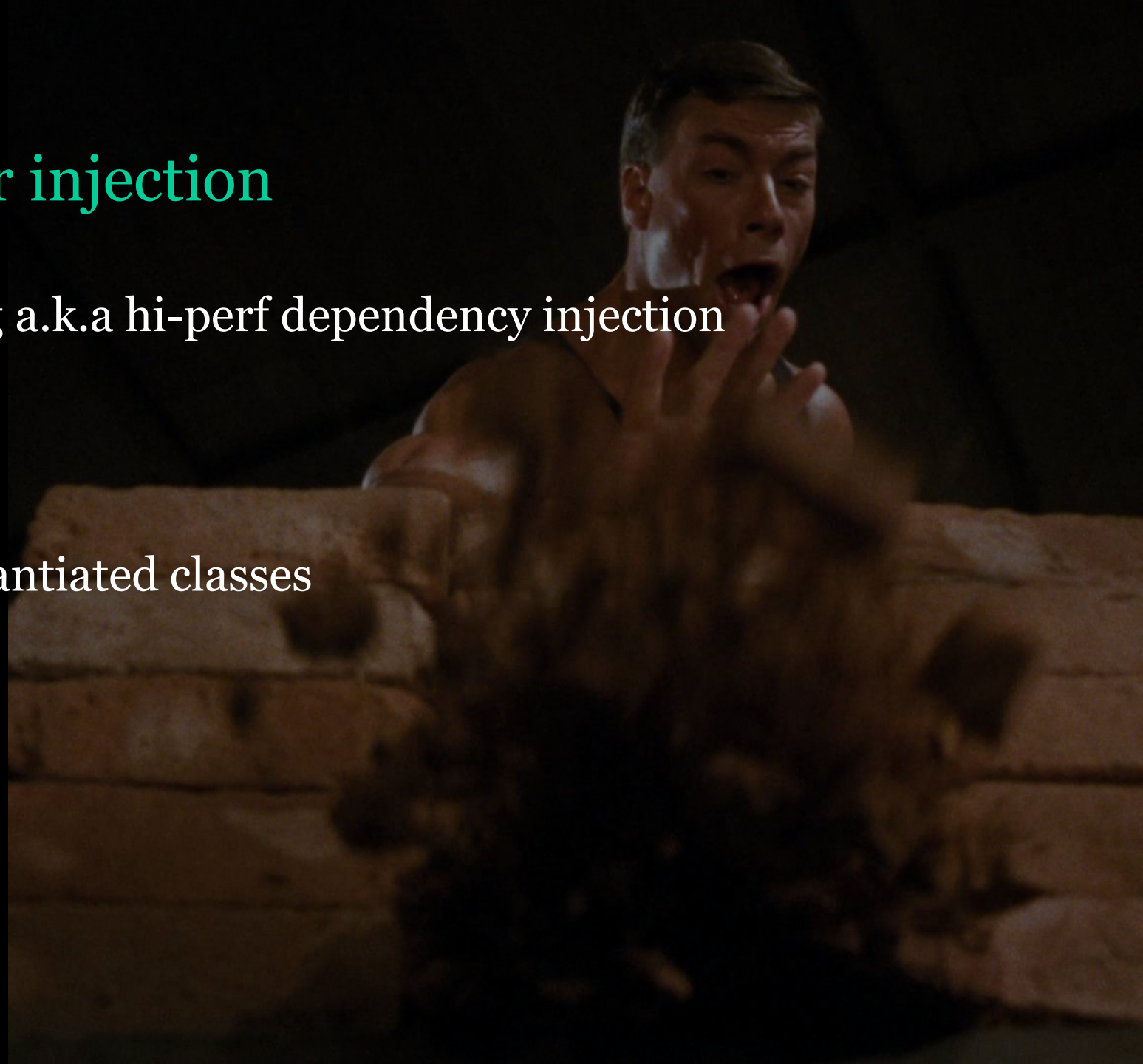
- Static methods
- Hard/impossible to instantiate class
- Non-virtual methods

Cannot be inherited == cannot be injected!

Using templates for injection

Compile time duck typing a.k.a hi-perf dependency injection

- Can fake unfakeables
- Can fake internally instantiated classes



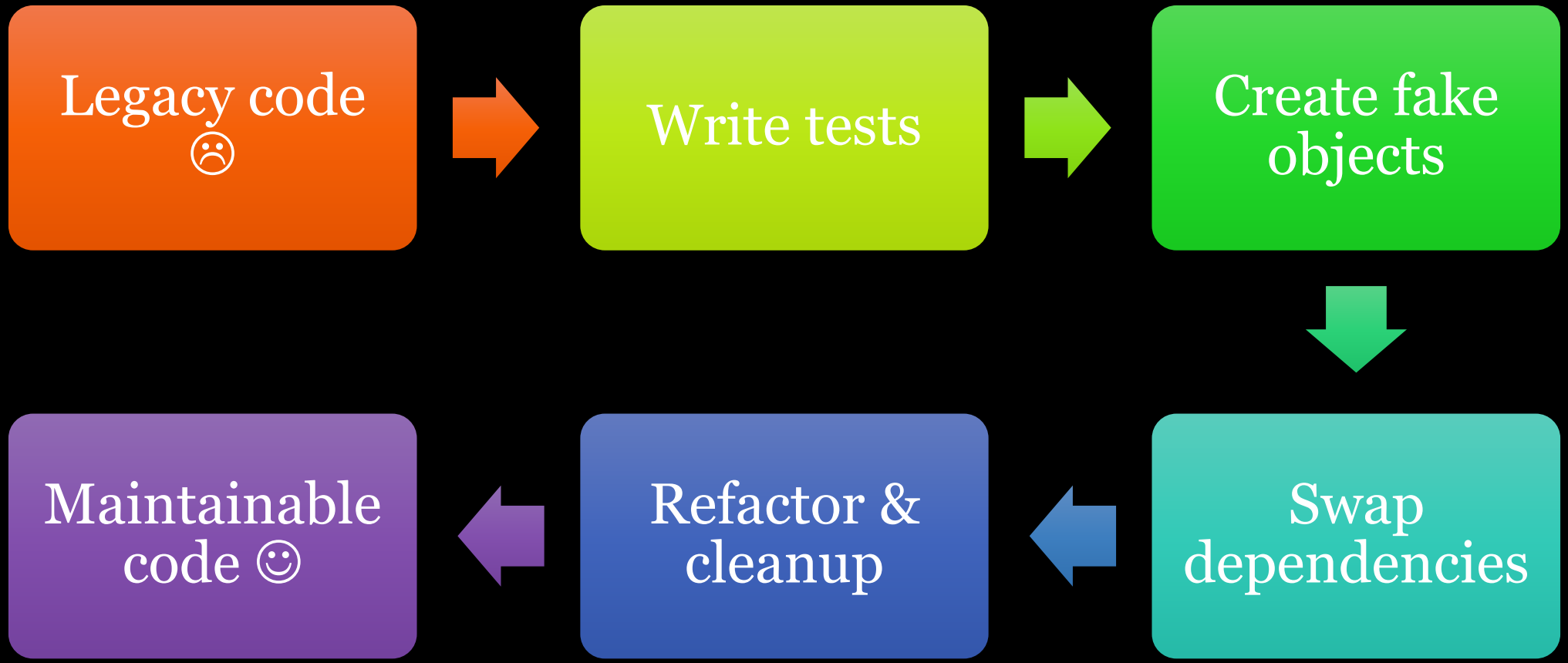
Faking Private and Protected Methods

```
class Foo
{
public:
        virtual bool MyPublicMethod(MyClass* c){...};
protected:
        virtual void MyProtectedMethod(int a, int b){...};
private:
        virtual int MyPrivateMethod(){...}
}
```

Faking Private and Protected Methods

```
class Foo
{
public:
        MOCK_METHOD1(MyPublicMethod, bool(MyClass*));
        MOCK_METHOD2(MyProtectedMethod, void(int, int));
        MOCK_METHOD0(MyPrivateMethod, int());
}
```

Taking control of existing legacy code



Thank you

Dror Helper | @dhelper | http://helpercode.com

Pluralsight courses:

- C++ Unit Testing Fundamentals Using Catch
- Advanced C++ Mocking Using Google Mock