To Mock or Not?

Curse and Blessing of Test Doubles

Taxonomy (Gerard Meszaros, Martin Fowler)

- **Dummy** objects are passed around but never actually used.
- **Fake** objects actually have working (simplified) implementations.
- **Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
- **Spies** are stubs that also record some information based on call.
- **Mocks** are objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

Mock User Code 1 (Python stdlib unit.mock)

```
>>> from unittest.mock import MagicMock
>>> thing = ProductionClass()
>>> thing.method = MagicMock(return_value=3)
>>> thing.method(3, 4, 5, key='value')
3
>>> thing.method.assert_called_with(3, 4, 5, key='value')
```

But beware of **private** implementation details ^^^^^^^^^^^^^^^^^^ !

Mock User Code 2 (Test)

Calling **`ProductionClass().method`** triggers **`something`** method:

```
>>> class ProductionClass:
...     def method(self):
...         self.something(1, 2, 3)
...     def something(self, a, b, c):
...         pass
...
>>> real = ProductionClass()
>>> real.something = MagicMock()
>>> real.method()
>>> real.something.assert_called_once()
```

Mock User Code 3 (Context Manager)

```
>>> from unittest.mock import patch
>>> with patch.object(ProductionClass, 'method', return_value=None) as mock_method:
...     thing = ProductionClass()
...     thing.method(1, 2, 3)
...
>>> mock_method.assert_called_once_with(1, 2, 3)
```

Language capabilities of 🐍 offer easy patching - also via decorators!

Anything goes ... ☁7

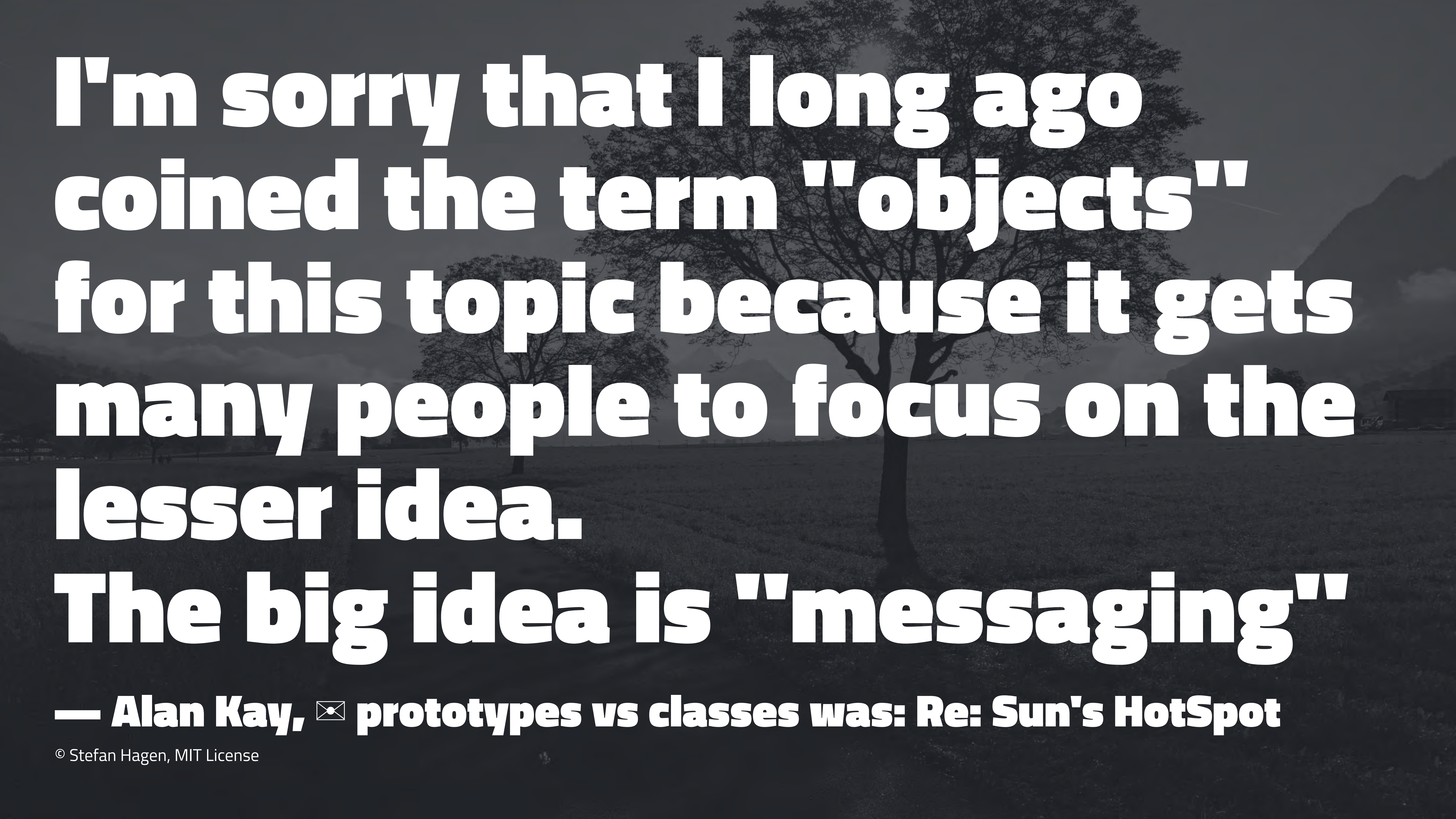
Mock User Code 3 (Patch Dict)

`patch()` **(temporarily) changes the object a name points to with another one**

```
>>> foo = {'key': 'value'}
>>> original = foo.copy()
>>> with patch.dict(foo, {'newkey': 'newvalue'}, clear=True):
...     assert foo == {'newkey': 'newvalue'}
...
>>> assert foo == original
```

Mock System Code (Python stdlib unit.mock)

```
>>> m = mock_open()
>>> with patch('__main__.open', m):
...     with open('foo', 'w') as h:
...         h.write('some stuff')
...
>>> m.mock_calls
[call('foo', 'w'),
 call().__enter__(),
 call().write('some stuff'),
 call().__exit__(None, None, None)]
>>> m.assert_called_once_with('foo', 'w')
>>> handle = m()
>>> handle.write.assert_called_once_with('some stuff')
```



I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea. The big idea is "messaging"

— Alan Kay, ✉ prototypes vs classes was: Re: Sun's HotSpot

Now for something completely different:

... C++

C++ Production Class - Kind of ... (Google Mock)

Easy way to go virtual and hook into the vtable (👴's guess):

```cpp
class ProductionInterface {
  ...
  virtual ~ProductionInterface() {}
  virtual void Method() = 0;
  virtual void Progress(int distance) = 0;
  virtual void GoTo(int x, int y) = 0;
  virtual int GetX() const = 0;
};
```

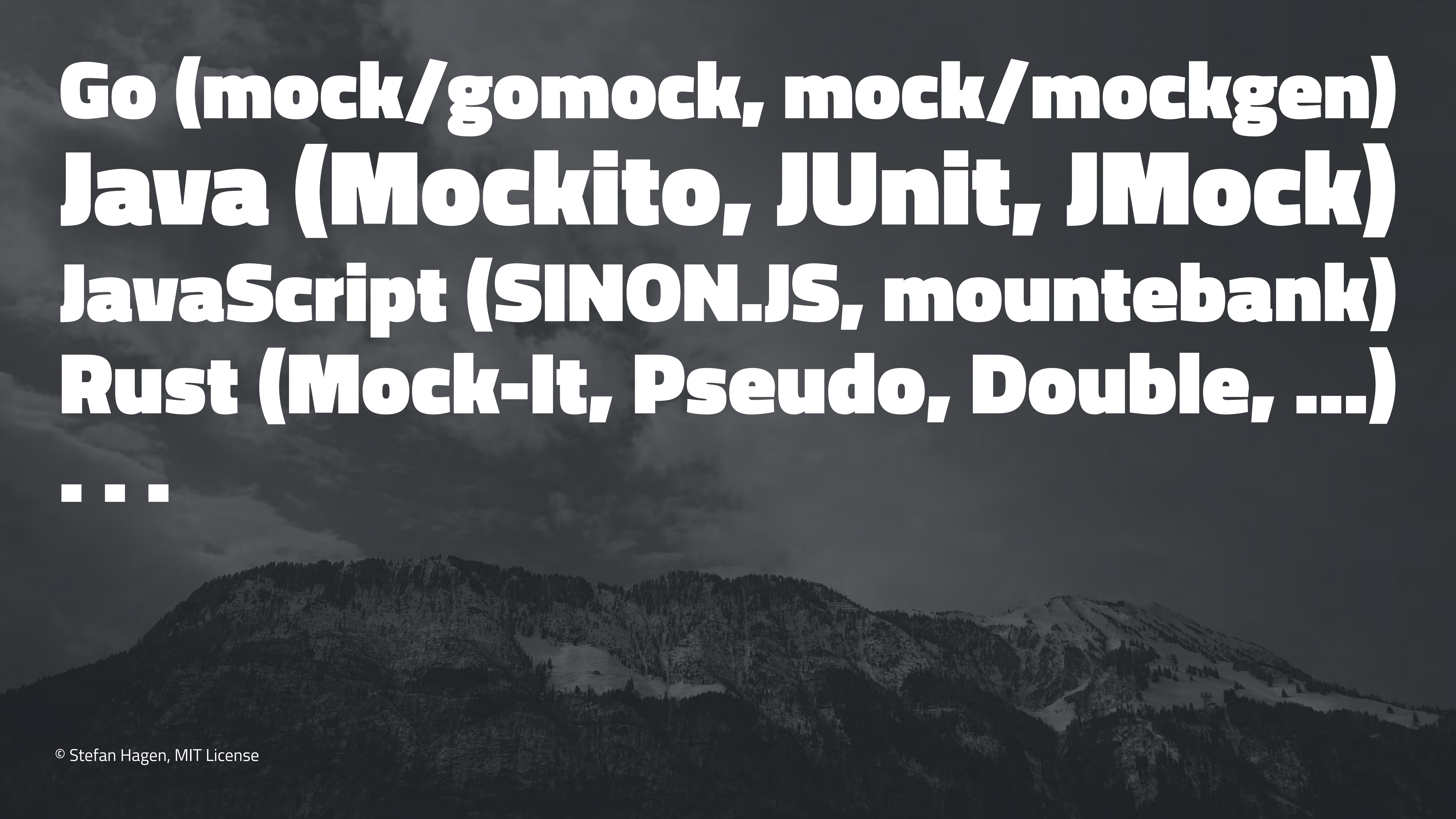
mocking non-virtual methods using templates is much more involved
— googlemock/docs/ForDummies

C++ Mock Class

```cpp
#include "gmock/gmock.h"  // Brings in Google Mock.
class MockProductionInterface : public ProductionInterface {
 public:
  ...
  MOCK_METHOD0(Method, void());
  MOCK_METHOD1(Progress, void(int distance));
  MOCK_METHOD2(GoTo, void(int x, int y));
  MOCK_CONST_METHOD0(GetX, int());
};
```

C++ Test With Mock

```
#include "path/to/mock-productioninterface.h"
#include "gmock/gmock.h"
#include "gtest/gtest.h"
using ::testing::AtLeast;
TEST(MethodTest, CanUseSpecialMethod) {
  MockProductionInterface blueprint;
  EXPECT_CALL(blueprint, Method()).Times(AtLeast(1));
  ProductionClass thing(&blueprint);
  EXPECT_TRUE(thing.SpecialMethod(0, 0, 10));
}
int main(int argc, char** argv) {
  ::testing::InitGoogleMock(&argc, argv);
  return RUN_ALL_TESTS();
}
```



Go (mock/gomock, mock/mockgen)

Java (Mockito, JUnit, JMock)

JavaScript (SINON.JS, mountebank)

Rust (Mock-It, Pseudo, Double, ...)

...

The best way to predict the future is to invent it
— Alan Kay

Mock Guidelines

- Use with care - always 💣
- Low-level ease of use depends on language capablities 🐍
- Do not test private implementation details (order of arguments, ...)
- High-level: Anything goes 😄 but:
    - Closely monitor benefit / cost ratio
    - Shape expectations

👋👴👉 . . . To be continued