Prerequisites:

Erlang

Rebar

Git

```
git clone git://github.com/eproxus/meck.git
```

Erlang Solutions Ltd.

Meck Tutorial

Learning Mocks with Meck

What Is Mocking?

- Dummy objects
- Fake objects
- Stubs
- Mocks

http://martinfowler.com/articles/mocksArentStubs.html

What Is Mocking?

- Dummy objects
- Fake objects
- Stubs
- Mocks

} meck

http://martinfowler.com/articles/mocksArentStubs.html

Where Is meck Used?

- ESL - Many internal projects
- Basho - Riak Enterprise
- Hibari - The Hibari key value store
- Mochi Media - In-game Advertising
- 100+ projects on GitHub

Alternative Ways To Mock

```
test() ->
    % Setup mock
    F = "test/mock.erl",
    {ok, M, B} =
        compile:file(F, [binary]),
    code:load_binary(M, F, B),

    % Run tests
    ?assert(system_under_test:run()),

    code:purge(mock).
```

Other Mocking Libraries

- erlymock - https://github.com/sheyll/erlymock

  Unmaintained?
- emock - https://github.com/noss/emock
- effigy - https://github.com/cliffmoon/effigy

Create a Module

```
1> meck:new(test).
ok

2> test:module_info().
[{exports,[{module_info,0},
           {module_info,1}]},
 {imports,[]},
 {attributes,[{vsn,[276740...]}]},
 {compile,[{options,[]},
           {version,"4.7.4"},
           {time,{2011,5,30,11,48,5}},
           {source,"."}]}]
```

Add Functions

```
3> meck:expect(test, foo,
                  fun() -> bar end).
ok

4> test:foo().
bar

5> meck:expect(test, baz, 0, qux).
ok

6> test:baz().
qux
```

Returning Changing Results

```
7> meck:sequence(test, read, 0,
                 ["foo", "bar", eof]).
ok

8> test:read().
"foo"
9> test:read().
"bar"
10> test:read().
eof
11> test:read().
eof
```

Returning Changing Results

```
12> meck:loop(test, count, 0,
                  [1, 2, 3, 2]).
ok

13> [test:count()
     || _ <- lists:seq(1, 10)].
[1,2,3,2,1,2,3,2,1,2]
```

Delete Functions

```
14> meck:delete(test, foo, 0).
ok

15> test:foo().
** exception error: undefined function
test:foo/0

16> test:module_info().
[{exports,[{read,0},{count,0},{baz,0},
           {module_info,0},
           {module_info,1}]},
 ...]
```

Yay, exercises!

(page 1)

Delete Functions

```
14> meck:delete(test, foo, 0).
ok

15> test:foo().
** exception error: undefined function
test:foo/0

16> test:module_info().
[{exports,[{read,0},{count,0},{baz,0},
           {module_info,0},
           {module_info,1}]},
 ...]
```

Delete Functions

```
14> meck:delete(test, foo, 0).
ok

15> test:foo().
** exception error: undefined function
test:foo/0

16> test:module_info().
[{exports,[{read,0},{count,0},{baz,0},
           {module_info,0},
           {module_info,1}]},
 ...]
```

Delete Functions

```
14> meck:delete(test, foo, 0).
ok

15> test:foo().
** exception error: u     d function
test:foo/0

16> test:module_i
[{exports,[{read                az,0},
           {mod
           {module_info,1}]},
 ...]
```

Image by megapiksel

Linking

```
17>
=ERROR REPORT==== 30-May-2011::15:10:55 ===
** Generic server test_meck terminating
** Last message in was
     {'EXIT',<0.63.0>,
      {undef,[{test,foo,[]},...]}}

** When Server state == {state,test,...}

** Reason for termination ==
** {'module could not be loaded',
     [{test,foo,[]},...]}
```

Linking

```
18> meck:new(test, [no_link]).
ok

19> exit(crash).
** exception exit: crash

20> test:module_info().
[{exports,[{module_info,0},
           {module_info,1}]},
 ...]
```

Validating a Module

```
21> meck:validate(test).
true
22> Foo = fun(A) when is_integer(A) ->
                  bar
              end.
#Fun<erl_eval.6.80247286>
23> meck:expect(test, foo, Foo).
ok
24> test:foo(1).
bar
25> meck:validate(test).
true
```

Validating a Module

```
26> test:foo(a).
** exception error: no function clause
     matching
erl_eval:'-inside-an-interpreted-
fun-'(a)
27> meck:validate(test).
false
```

Exceptions

```
28> meck:validate(test).
true
29> meck:expect(test, foo,
          fun() -> exit(crash) end).
ok
30> test:foo().
** exception exit: crash
     in function  meck:exec/4
     in call from test:foo/0
        called as test:foo()
31> meck:validate(test).
false
```

Exceptions

```
32> meck:expect(test, foo,
          fun() ->
              meck:exception(exit, crash)
          end).
ok
33> test:foo().
** exception exit: crash
       in function meck:exception/2
       in call from test:foo/0
          called as test:foo()
34> meck:validate(test).
true
```

Exceptions

```
35> try test:foo()
    catch _:_ -> erlang:get_stacktrace()
    end.
[{meck,exception,2},
 {meck,exec,4},
 {test,foo,[]},
 {erl_eval,do_apply,5},
 {erl_eval,try_clauses,8},
 {shell,exprs,7},
 {shell,eval_exprs,7},
 {shell,eval_loop,3}]
```

Yay, exercises!

(page 2)

Original Modules

```
36> l(original).
{module,original}
37> original:a().
a
38> original:b().
b
6> meck:new(original, [no_link]).
ok
7> original:a().
** exception error: undefined function
     original:a/0
```

Original Modules

```
10> meck:new(original, [no_link,
                        passthrough]).
ok
11> meck:expect(original, b, 0, z).
ok
12> original:a().
a
13> original:b().
z
```

Original Modules

```
14> original:double(1).
2
15> meck:expect(original, double,
15>         fun(1) -> foo;
15>             (A) -> meck:passthrough([A])
15>         end).
ok
16> original:double(1).
foo
17> original:double(2).
4
```

History

```
18> meck:expect(original, a,
                fun(1) -> foo end).
ok
19> original:a(2).
** exception error: function clause...
20> meck:history(original).
[{<0.84.0>,{original,a,[]},a}, % {MFA, R}
 {<0.84.0>,{original,b,[]},z},
 {<0.84.0>,{original,c,[1]},foo},
 % {MFA, C, R, ST}
 {<0.84.0>,{original,a,[2]},
  error,function_clause,
  [{original,a,[2]},...]}]
```

History

```
21> meck:called(original, '_', '_').
true
22> meck:called(original, a, ['_']).
true
22> meck:called(original, a, [2]).
true
23> meck:called(original, x, ["nope"]).
false
```

Multiple Modules

```
29> Mods = [x, y, z].
[x,y,z]
30> meck:new(Mods).
ok
31> meck:expect(Mods, foo, 0, bar).
ok
32> [M:foo() || M <- Mods].
[bar,bar,bar]
33> meck:validate(Mods).
true
34> meck:unload(Mods).
ok
```

Yay, exercises!

(page 3)

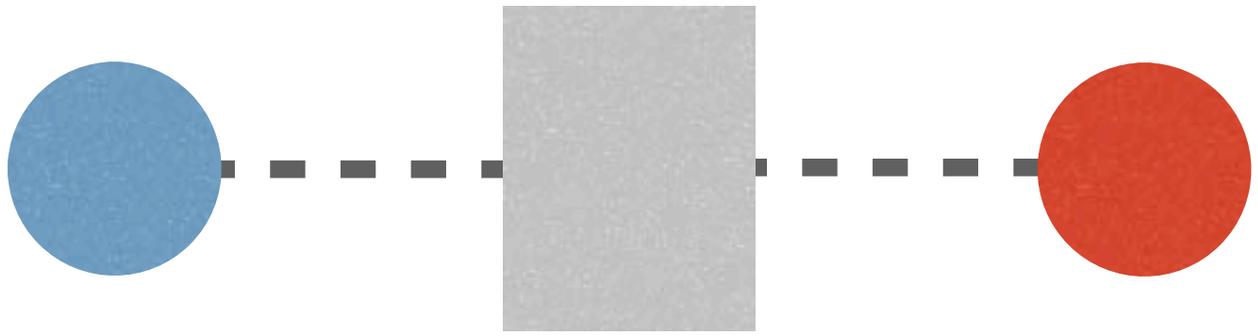
Using meck With EUnit

```
some_test() ->
      ok = meck:new(t),
      ?assertEqual(result, m:f()),
      ?assert(meck:called(t, f, [1, a])),
      ?assert(meck:validate(t)),
      ok = meck:unload(t).
```

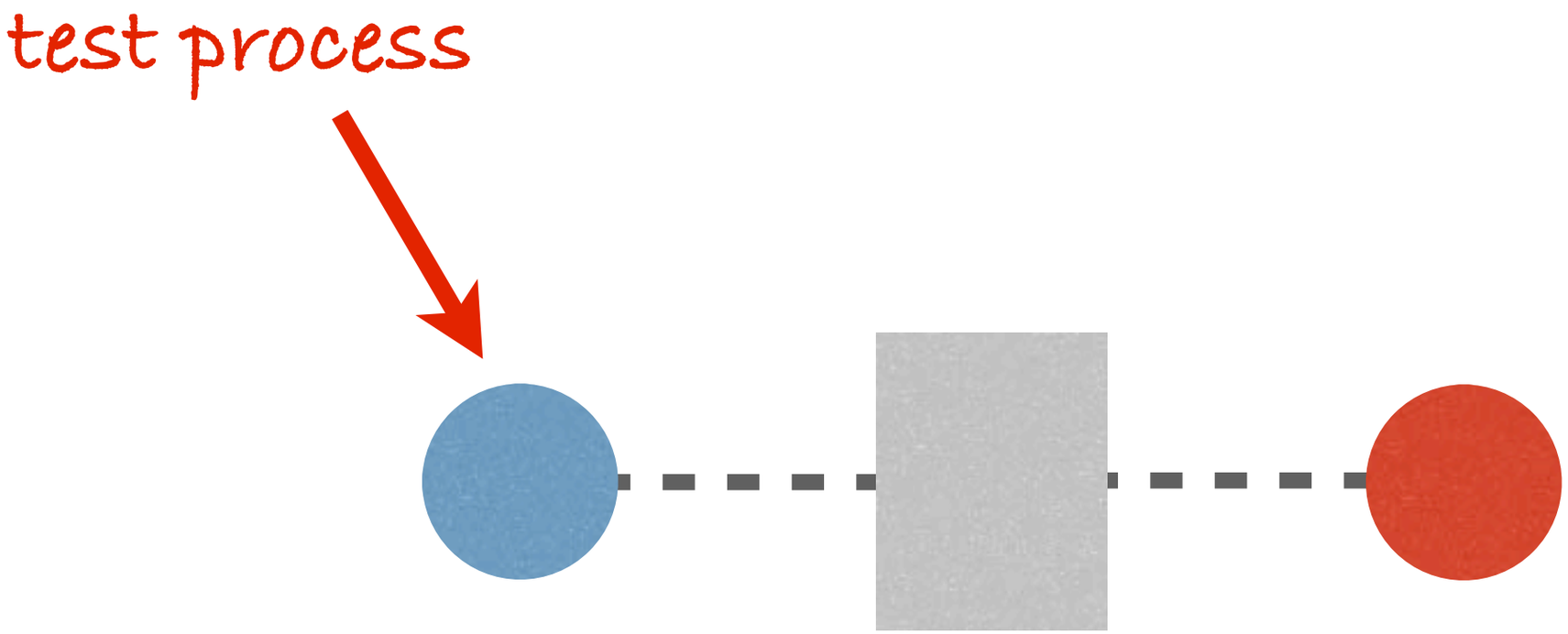
Using meck With EUnit

```
some_test_() ->
  {foreach, fun setup/0, fun teardown/1,
     [fun test_a/0, fun test_b/0,
      fun test_c/0]}.

setup() ->
     Mods = [x, y, z],
     meck:new(Mods),
     meck:expect(x, foo, 0, bar), % ...
     Mods.

teardown(Mods) -> meck:unload(Mods).
```

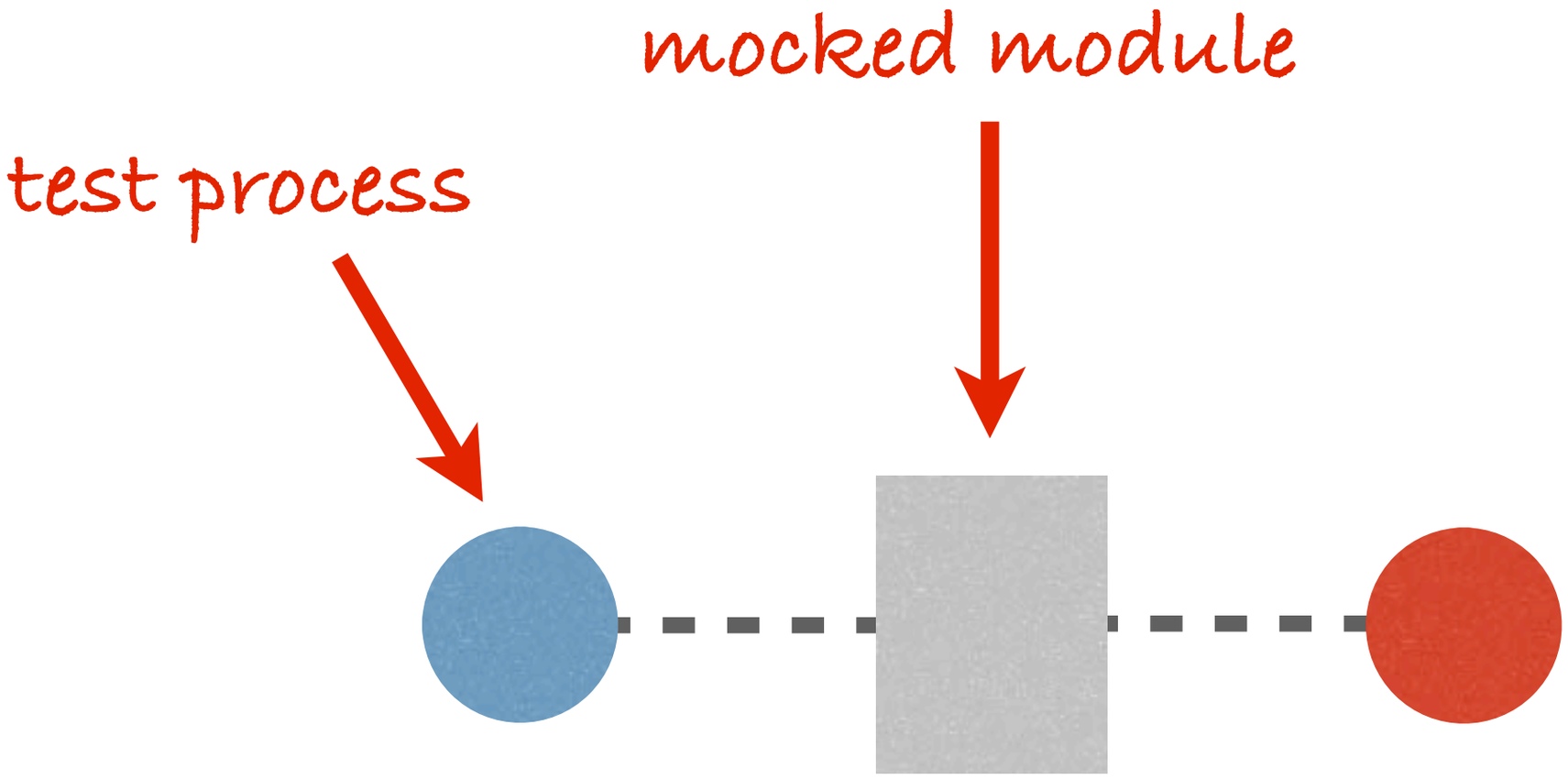
Behind the Scenes



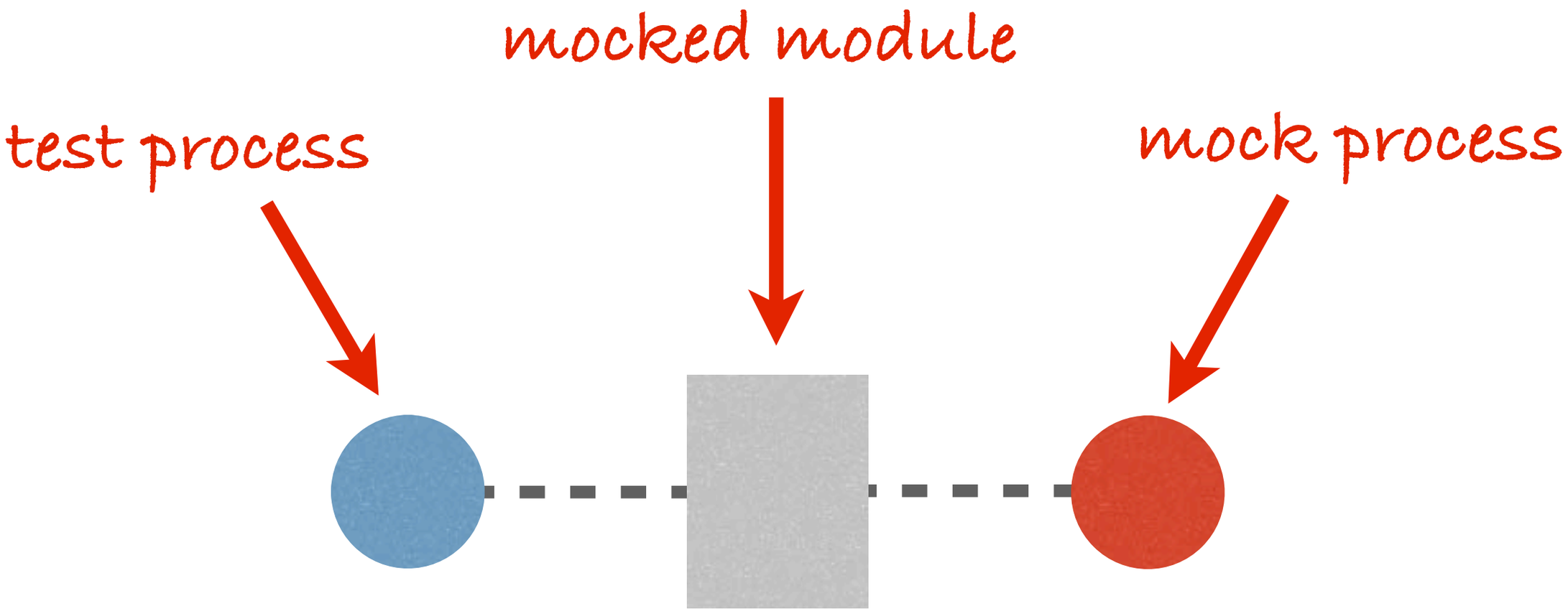
Behind the Scenes



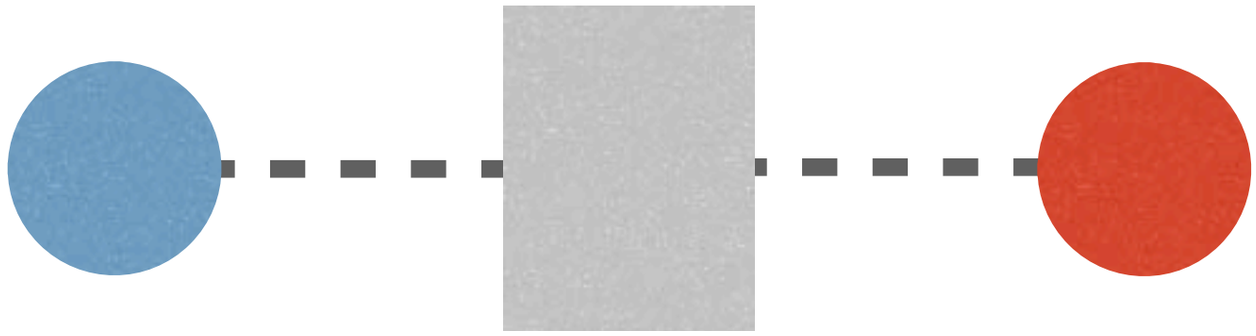
Behind the Scenes



Behind the Scenes

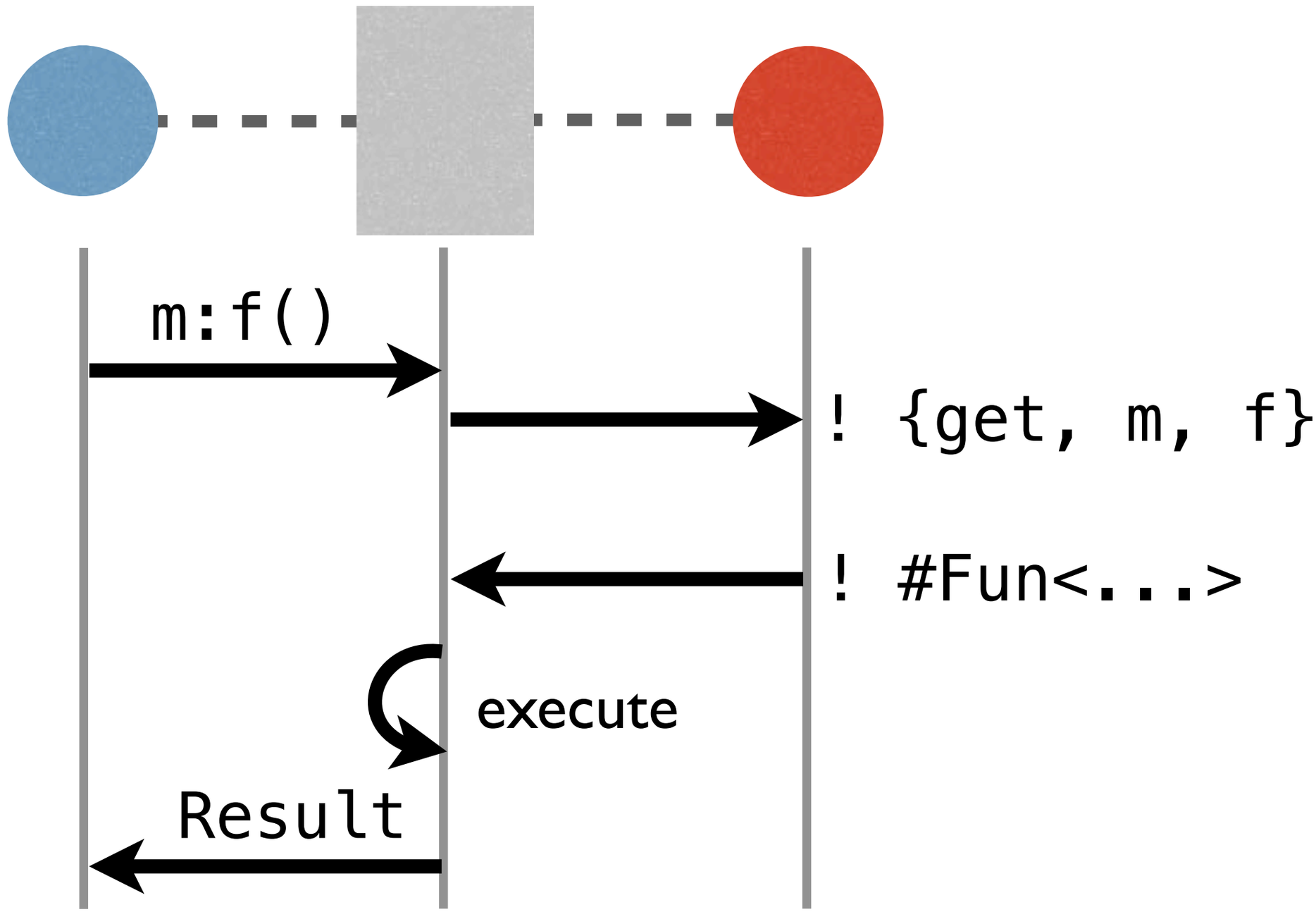


Behind the Scenes



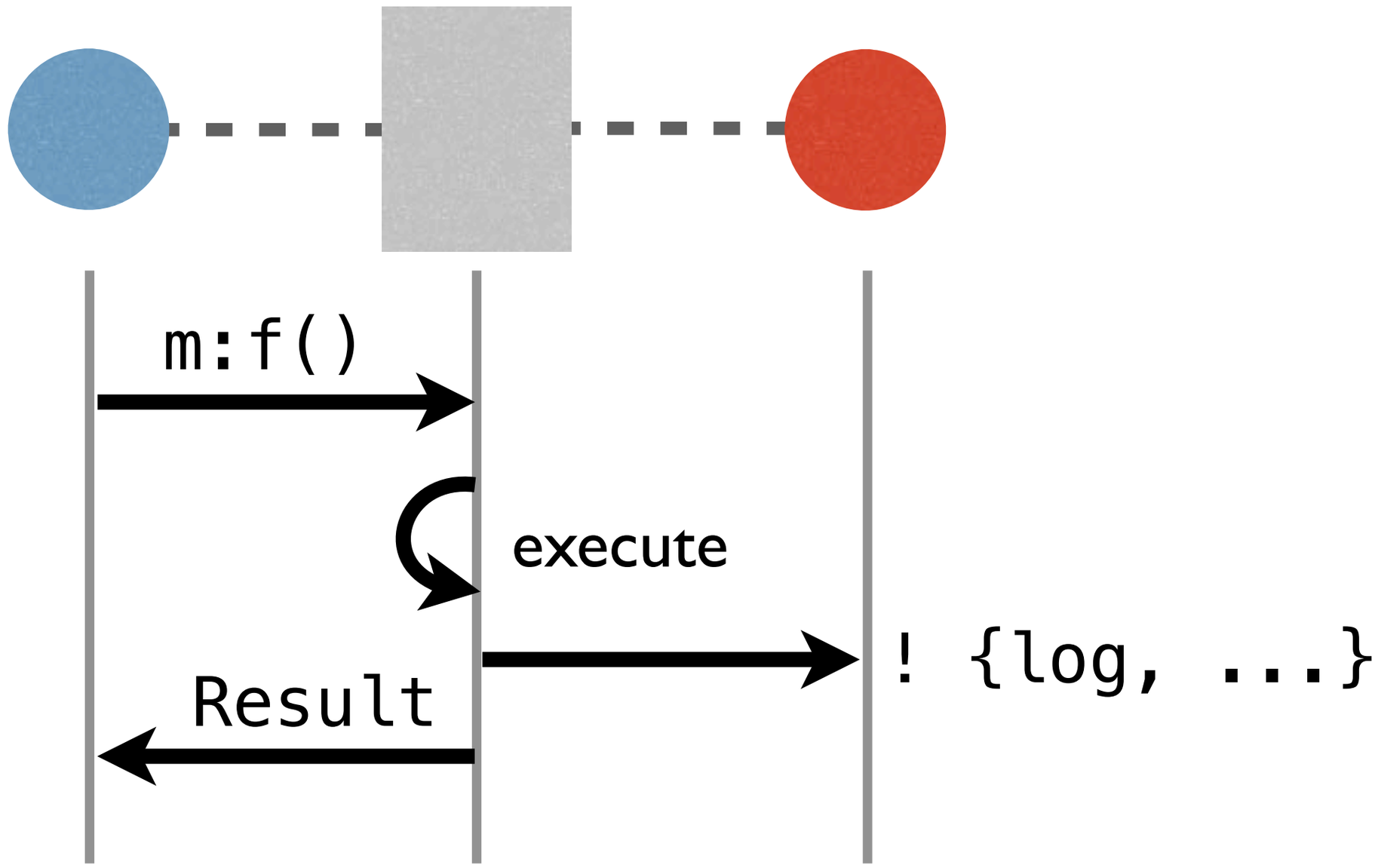
Behind the Scenes

```
meck:expect(m, f, fun() -> Result end)
```



Behind the Scenes

```
meck:expect(m, f, 0, Result)
```



Get It, Use It, Improve It!

github.com/eproxus/meck

@eproxus

eproxus@gmail.com

Erlang Solutions in Stockholm is hiring!
anders.waldner@erlang-solutions.com