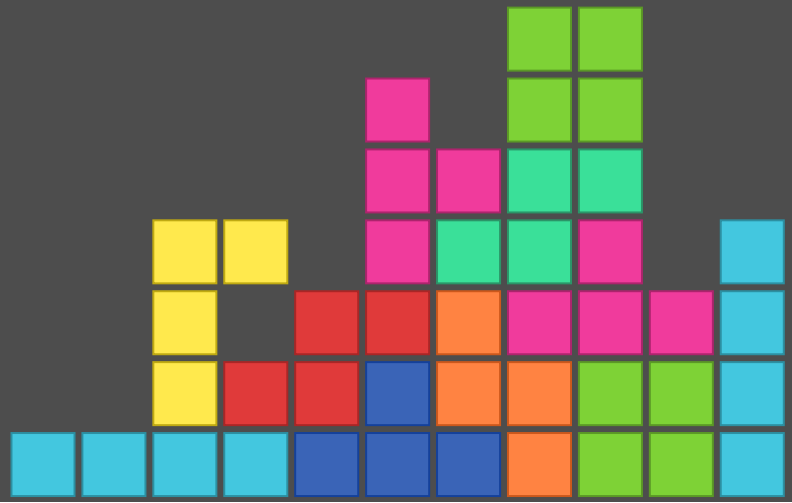
Unit testing and mocking with cmocka

SambaXP 2018

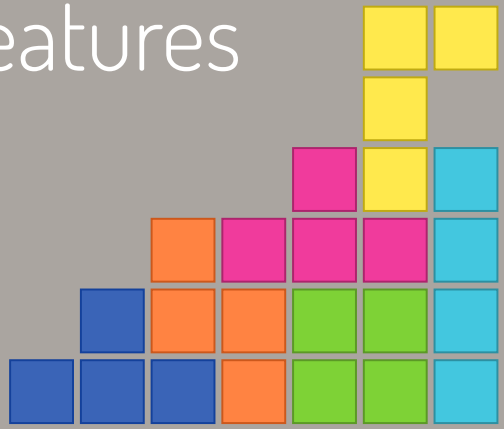
Andreas Schneider
Red Hat Samba Maintainer



About me

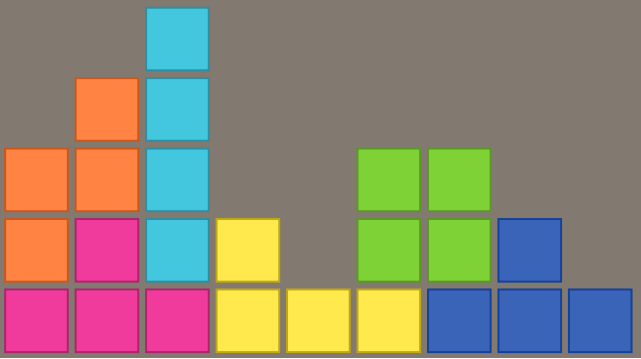
Free and Open Source Software Developer:

- cmocka - a unit testing framework for C
- Samba - The domain controller and file server
- libssh - The SSH Library
- cwrap - Client/Server testing made easy
- LineageOS - Android with Privacy Features



1

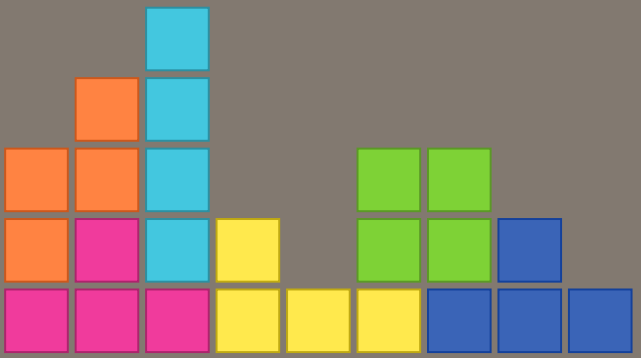
What is this talk about?



What is this talk about?

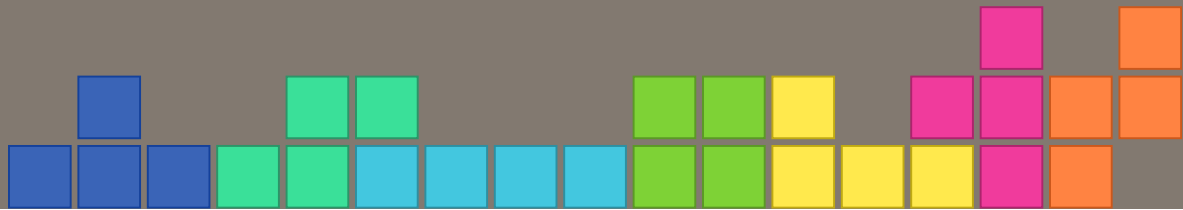
The talk will answer the following questions:

- What is cmocka?
- What features does cmocka provide?
- What is mocking?
- How to write a mocking test?



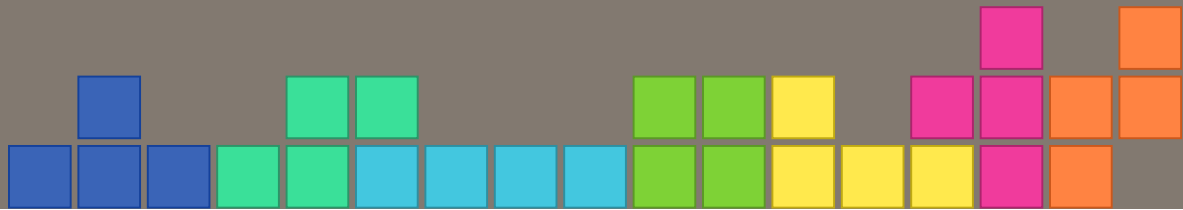
2

What is cmocka?



cmocka ...

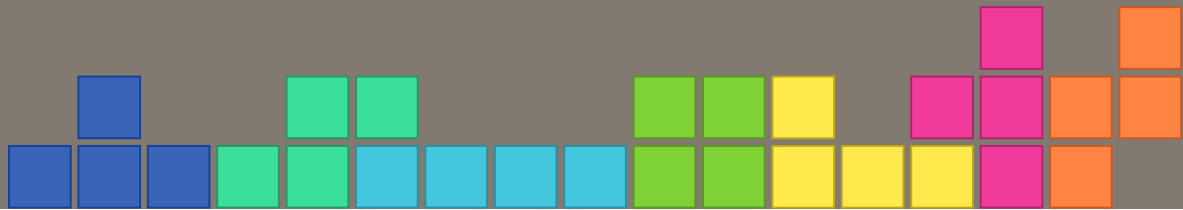
- is an elegant unit testing framework for C
- offers support for mock objects.
- it only requires the standard C library



cmocka ...

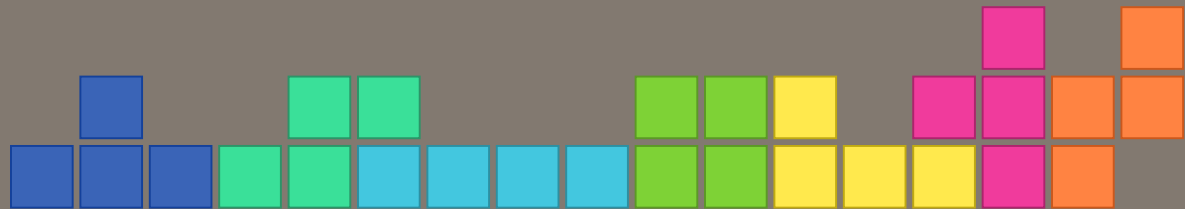
works on a range of computing platforms (including embedded) and works with different compilers.

Linux/BSD/Windows - GCC/Clang/MSVC



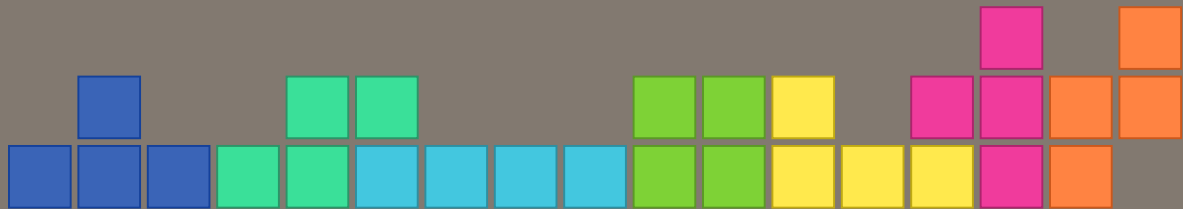
Mission Statement

The goal of this project is to provide a powerful testing framework for C, on different platforms and operating systems, which only requires the standard C library.



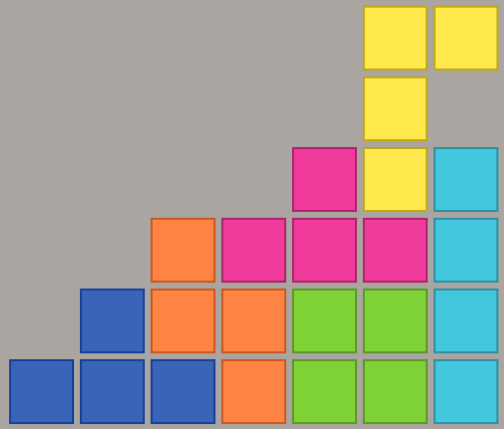
Website

https://cmocka.org/



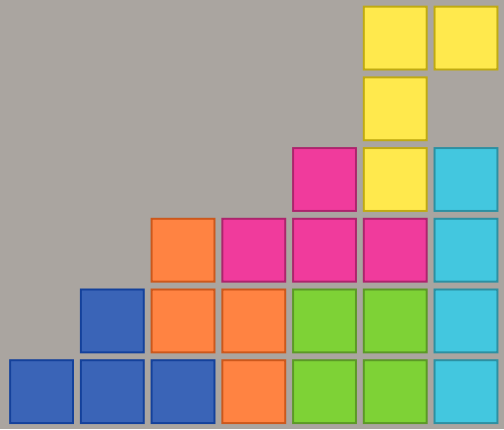
3

Features of cmocka



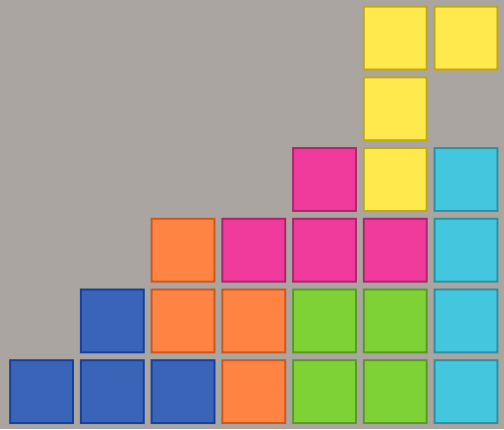
Test Fixtures and groups

Test fixtures are setup and teardown functions that can be shared across multiple test cases to provide common functions that prepare the test environment and destroy it afterwards. This is also supported for groups.



Exception handling

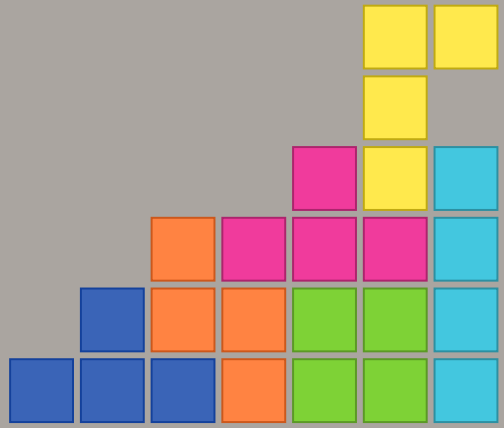
- cmocka is able to recover the test state if there are exceptions like a segfault.
- Handling for `SIGSEGV`, `SIGILL`, etc.
- An attached debugger will stop when the segfault occurs



Exception handling

cmocka doesn't use `fork()` for exception handling in test cases!

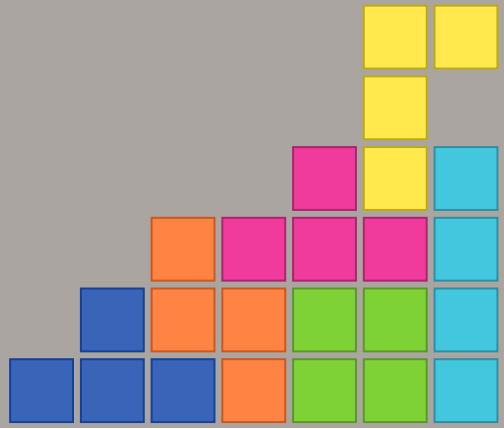
- `fork()` is not available on all platforms
- `fork()` is implemented diffrently on some OSes (Linux vs. MacOSX)



Output formats

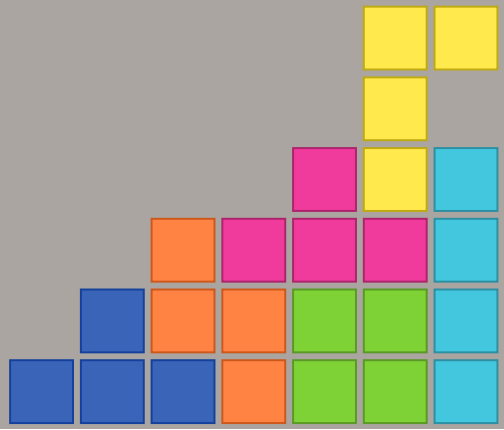
cmocka has it's own console output format, but supports additional message formats like:

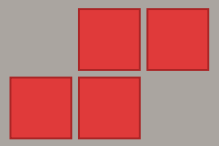
- Test Anything Protocol
- Subunit (used by Samba)
- xUnit XML (parsed by Jenkins)



API Documentation

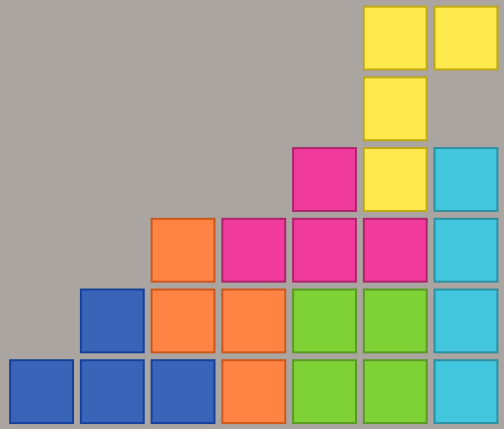
https://api.cmocka.org/





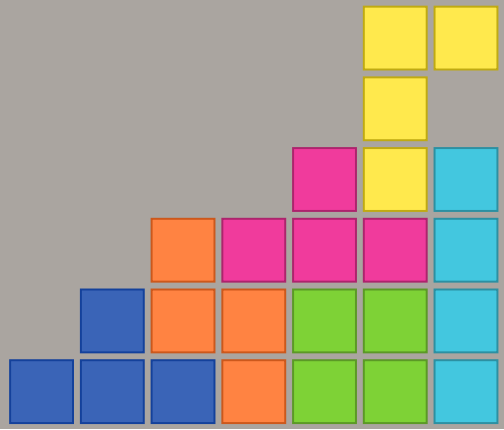
A cmocka test

```
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
```



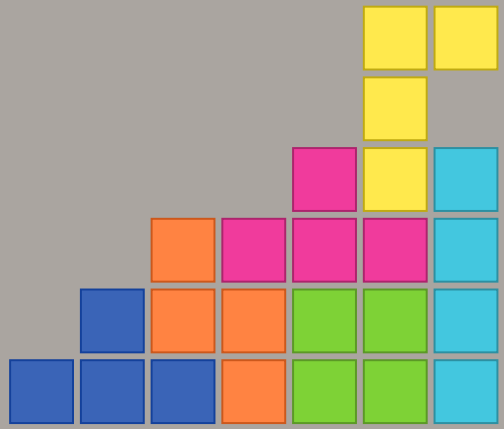
A cmocka test

```
/* A test case that does nothing and succeeds. */
static void null_test_success(void **state) {
    (void) state; /* unused */
}
```



A cmocka test

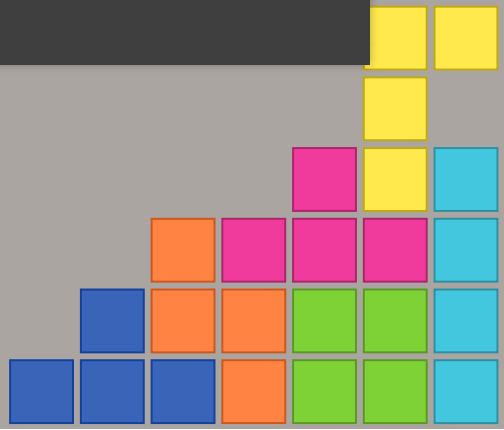
```
int main(void) {
    const struct CMUnitTest tests[] = {
        cmocka_unit_test(null_test_success),
    };

    return cmocka_run_group_tests(tests, NULL, NULL);
}
```



Assert functions

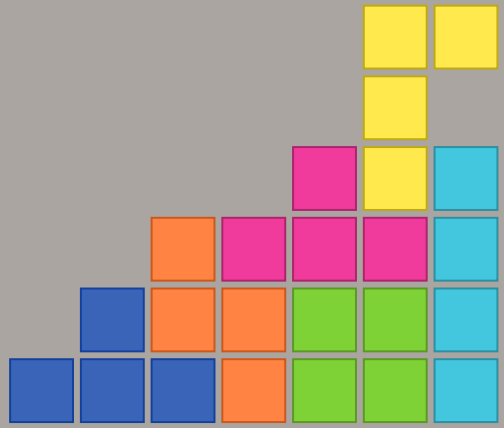
What you will mostly use are assert functions.
Examples:

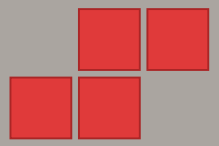
```
assert_true(x)
assert_false(x)

assert_non_null(x)
assert_null(x)

assert_int_equal(a, b)
assert_int_not_equal(a, b)

...
```



Assert functions

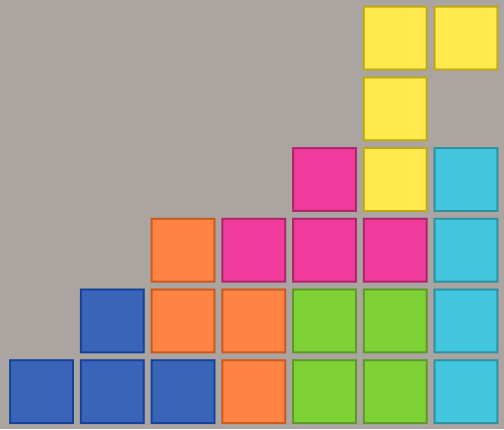
There are functions for comparing strings, memory, pointers and sets. You can also check ranges and return codes.





A cmocka test with assert function

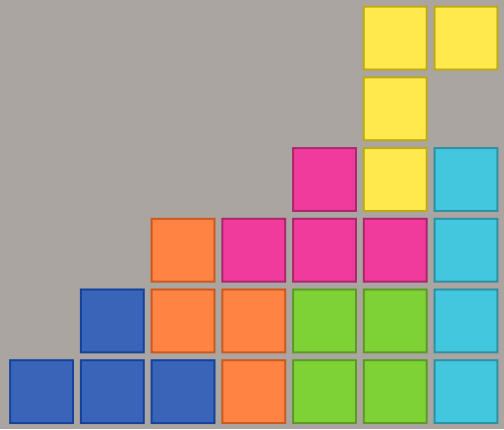
```
/* A test case that compare intetergers and fails. */
static void integer_failure(void **state) {
    int i = 4;

    assert_int_equal(i, 5);
}
```



Extending assert functions

You can also easily extend cmocka writing special assert functions for your project.

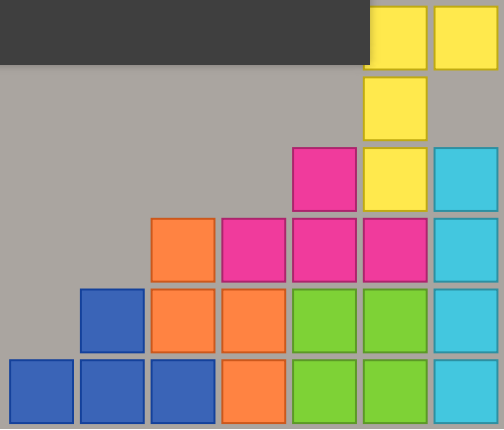
Example: socket_wrapper tests offer:
`assert_sockaddr_equal(ss, a)` and
`assert_sockaddr_port_equal(ss, a, prt)`



assert()

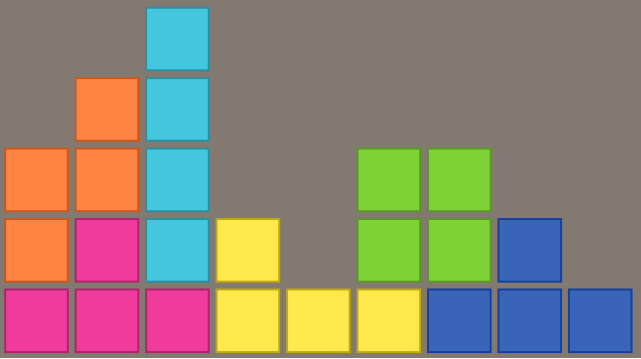
If you test code which use **assert()**, you could redefine **assert()** and test it!

```
#define assert mock_assert
void showmessage(const char *message) {
  assert(message);
}

int main(void) {
  expect_assert_failure(show_message(NULL));
  printf("succeeded\n");
  return 0;
}
```



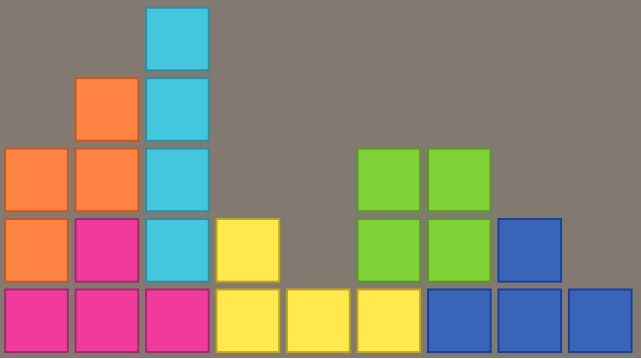
4

Mocking in unit tests



What is mocking?

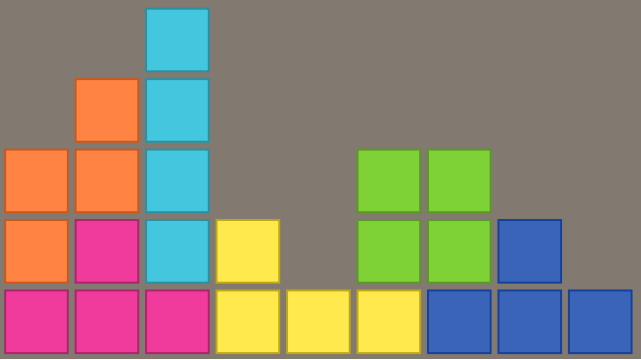
Mocking is a way to create instrumented objects that simulate the behavior of real objects.



What is mocking?

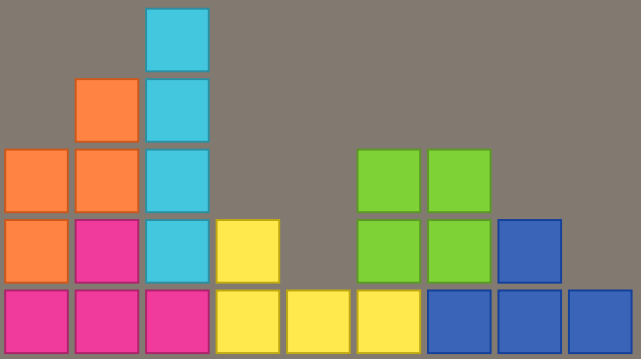
to mock = to imitate something

Mocking in unit testing is a way to isolate behaviour of complex algorithms. This is useful if some functions are impractical to incorporate into the unit test.

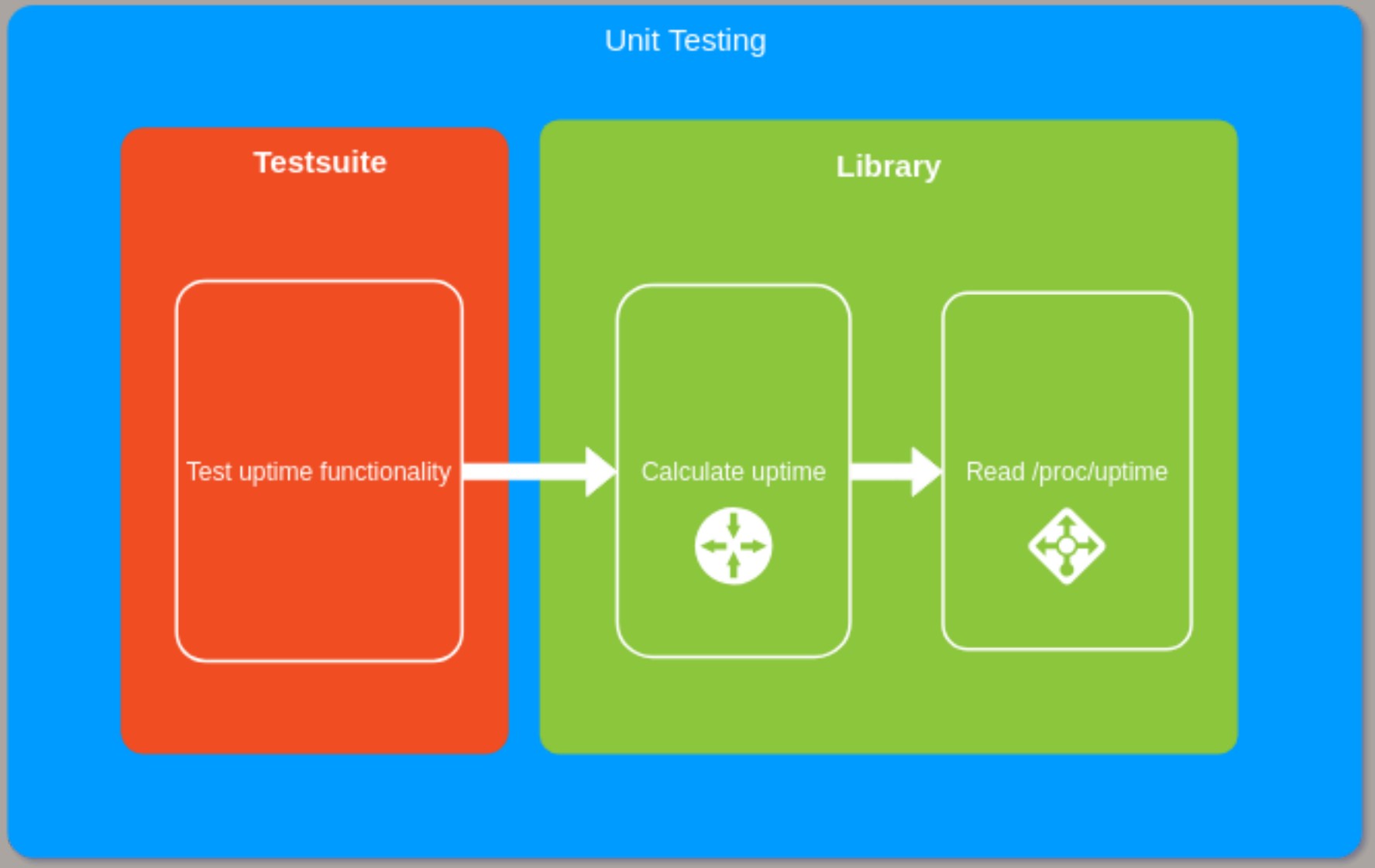


Lets test 'uptime'

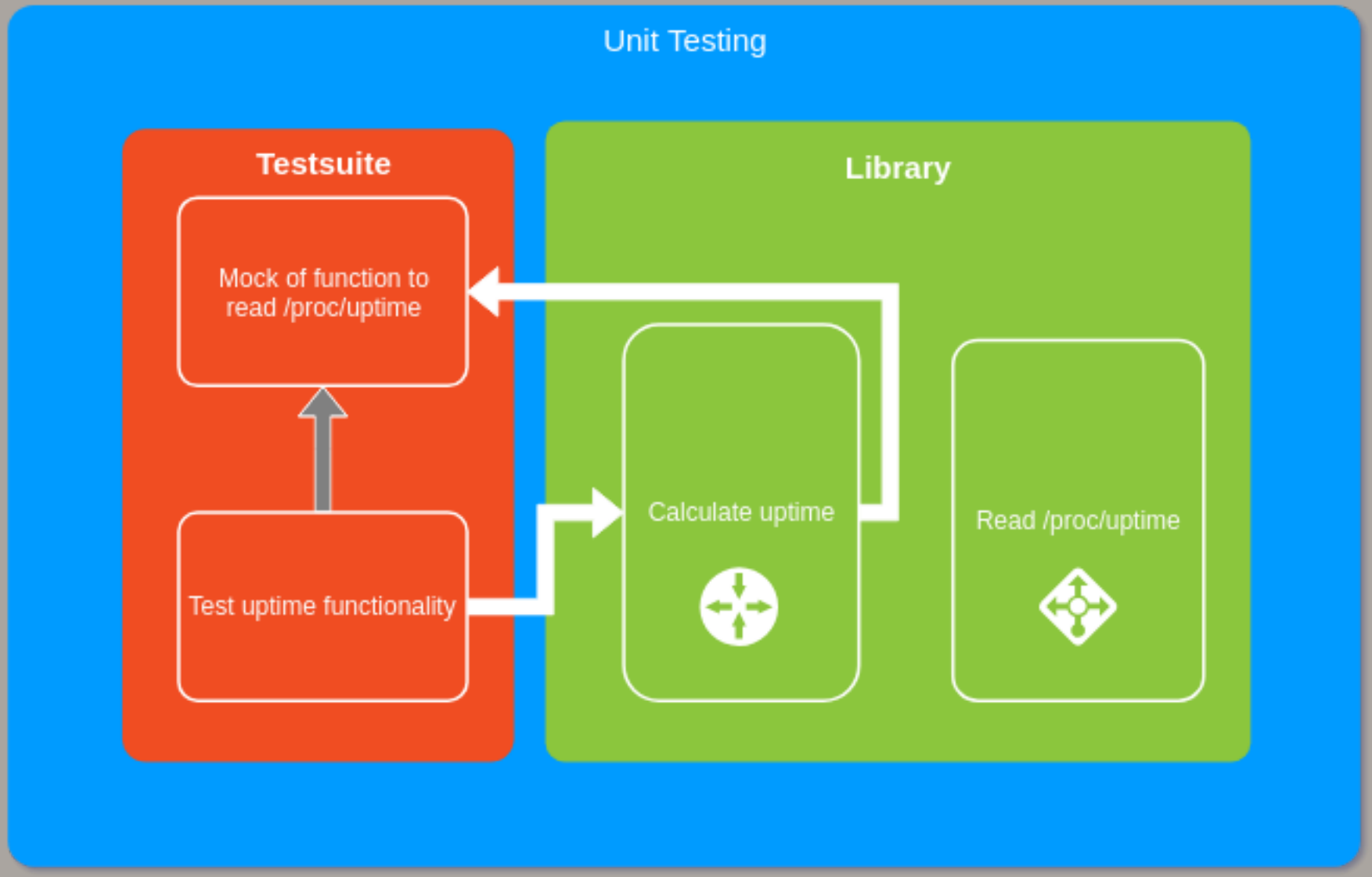
```
./example/uptime/uptime
up 3 days, 24 minutes
```



Standard unit test



Mocking test

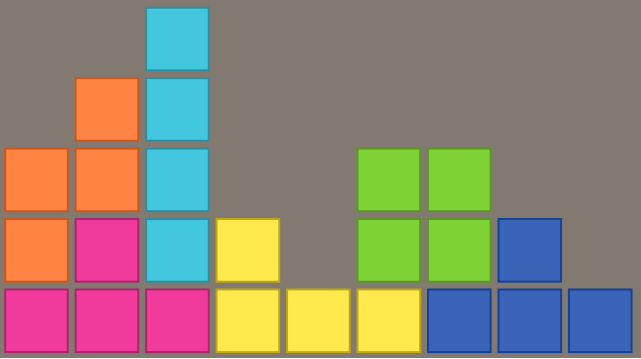


GNU linker magic

Use a wrapper function for a symbol.

```
ld --wrap=<symbol>
```

Supported by `ld.bfd`, `ld.gold` and `llvm-ld`



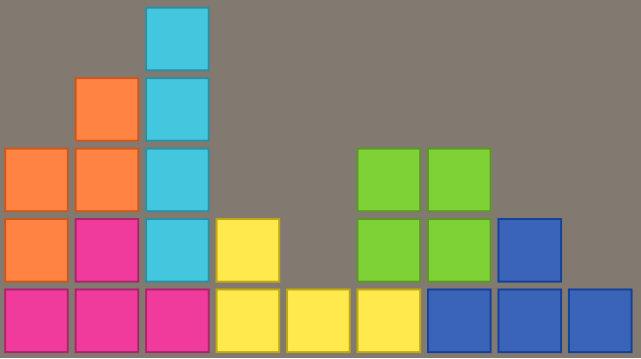
Linker function wrapping

If the function prototype is:

```
int uptime(double *uptime_secs, double *idle_secs)
```

We implement in the the mock function called:

```
int __wrap_uptime(double *uptime_secs, double *idle_se
{
    ...
}
```

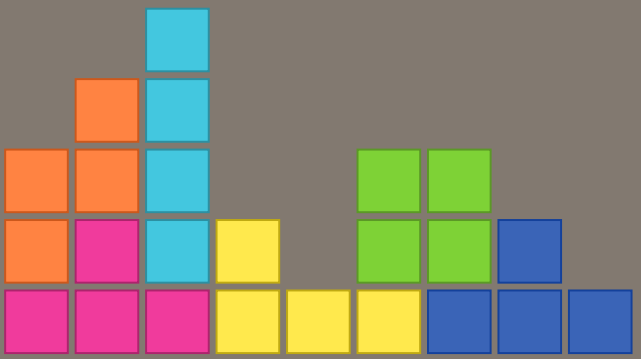


Linker function wrapping

The symbol to the origial `uptime()` function will be renamed to: `__real_uptime`

The `uptime` symbol is rebound to `__wrap_uptime`

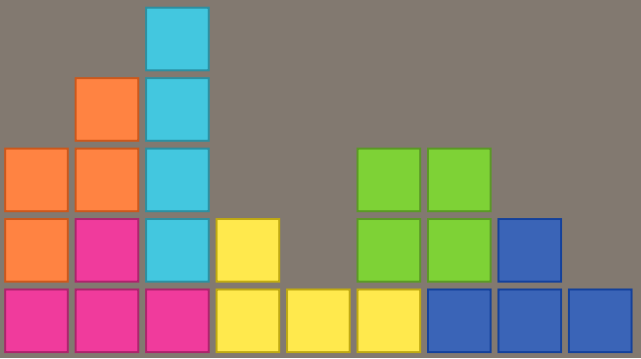
This way we can still call the original function if needed!



Smybol binding order!

Symbols are searched and bound by the linker in the follow order:

1. The executable itself
2. Preloaded libraries
3. Libraries in linking order

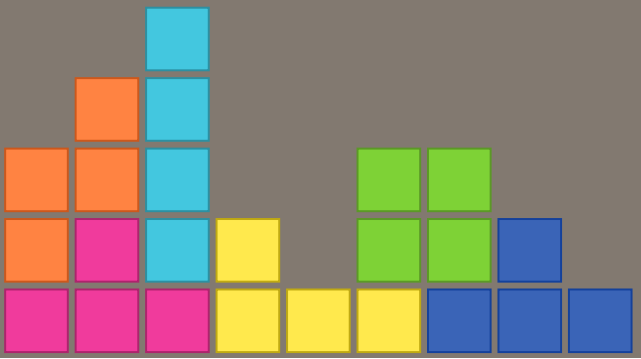


Debug symbol binding

With GNU ld.so ..

```
LD_DEBUG=symbols ./examples/uptime/uptime
```

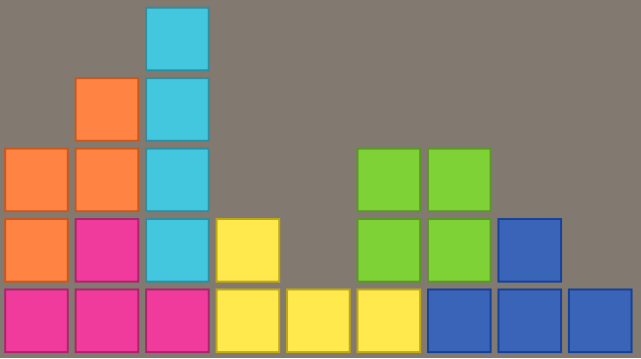
See '`man ld.so`'



Parameter checking

```
int __wrap_mock(char *name) {
    check_expected(name);
}

void test_foo(void **state) {
    expect_string(__wrap_mock, name, "wurst");

    foo("wurstbrot");
}
```

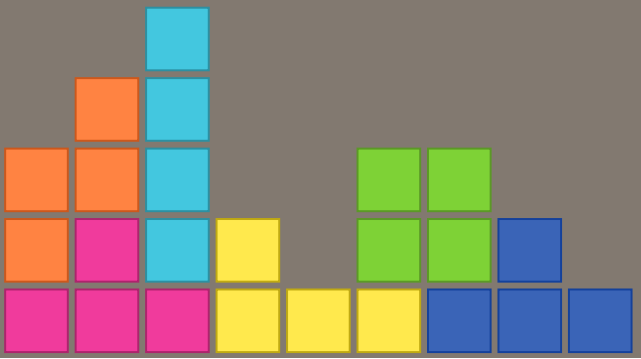
https://api.cmocka.org/ -> Checking Parameters



Mocks

```
int __wrap_mock(char *name) {
    return mock_type(int);
}

void test_foo(void **state) {
    int rc;

    will_return(__wrap_mock, 0);

    rc = foo("wurstbrot");
    assert_return_code(rc, errno);
}
```

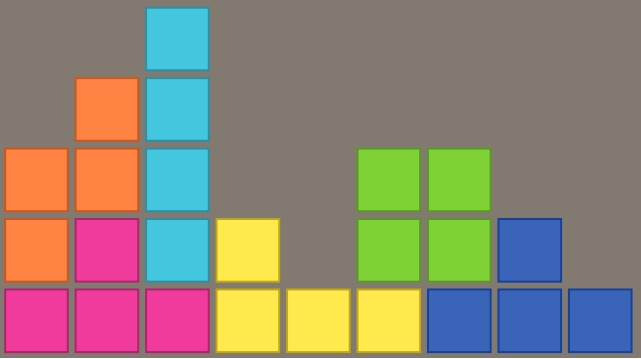
https://api.cmocka.org/ -> Mock Objects



Call ordering

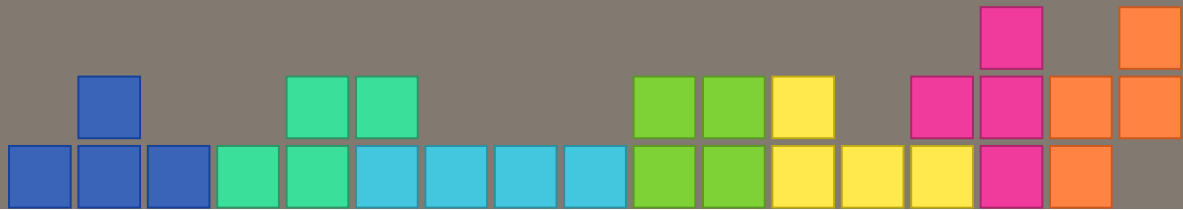
Allows you to check that mock functions are called in the right order!

https://api.cmocka.org/ -> Call Ordering



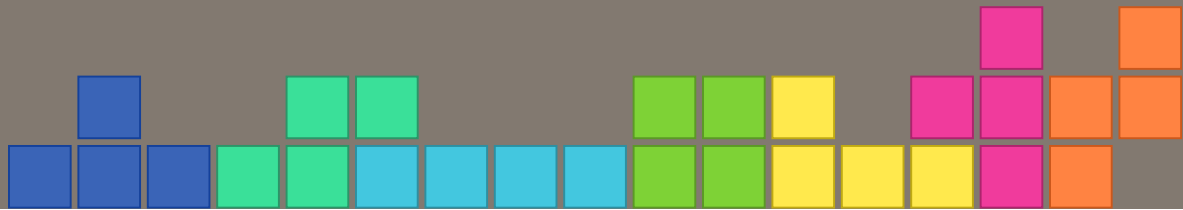
5

How to write a mocking test?



6

How to write a cmocka test in Samba?



GAME OVER