Back To Basics
Debugging and Testing

GREG LAW & MIKE SHAH

Cppcon
The C++ Conference

2024
September 15 - 20

Writing software is really hard.

If it’s not tested, it doesn’t work.

Or at least it will soon stop working.

WHAT'S WRONG WITH THIS PICTURE

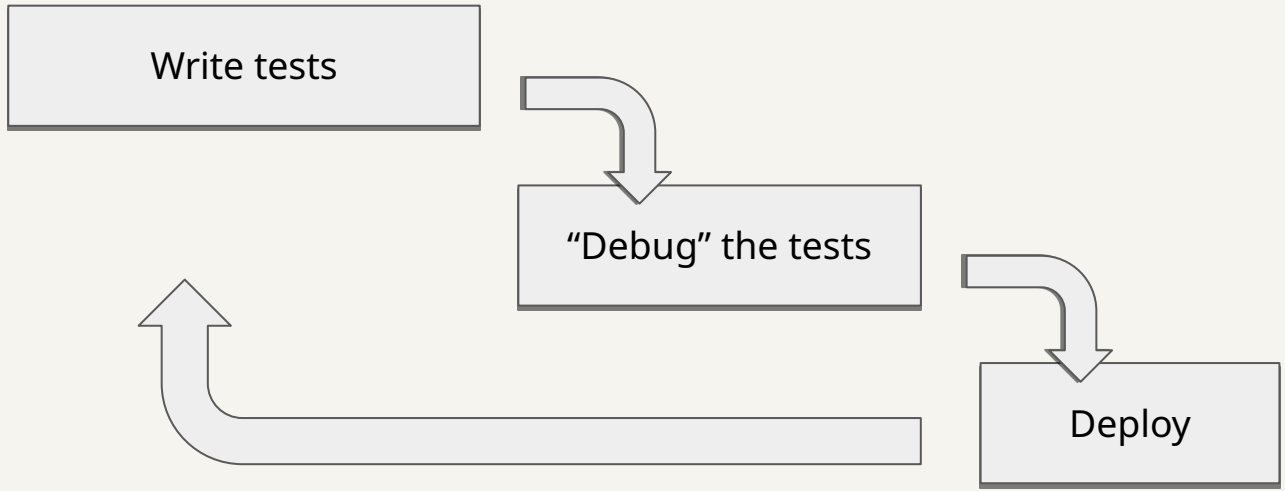
Write the code

ests

Debug

Deploy

Test-Driven Development




I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION
quickmeme.com

undo

undo

Software development done right is basically testing and debugging.

GOOD TESTS ARE...

- Independent
- Repeatable
- Fast

undo

DIFFERENT TESTS – ALL ARE NEEDED

- Unit tests.
- Integration tests.
- System tests.
- Acceptance tests.

undo

DIFFERENT TESTS AT DIFFERENT TIMES

- Iteratively / interactively.
- Pre-merge (aka Barrier Tests).
- Post-merge (aka Continuous Integration).

undo

GIVEN ... WHEN ... THEN ...

Given a server with maximum accepted connections.

When a new connection attempt is made.

Then error code ETOOMANY is returned.

UNIT TESTS

Test a module in isolation.

‘Mock’ interfaces to other modules / system.

Bugs are generally shallow / easy-to-fix.

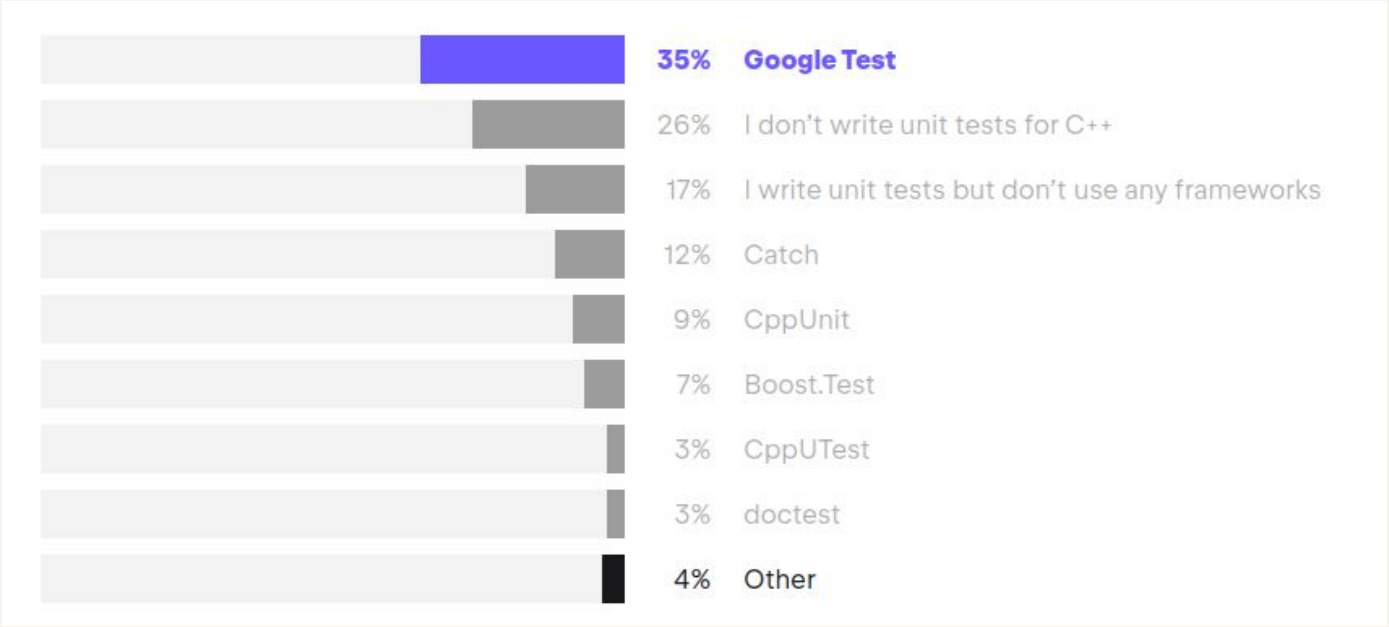
Tests run fast – tight feedback loop.

But…

Assumptions!

UNIT TESTING FRAMEWORKS

- Google Gest
- Catch2
- Boost.Test
- CppUTest
- doctest



GOOGLE TEST (GTest)

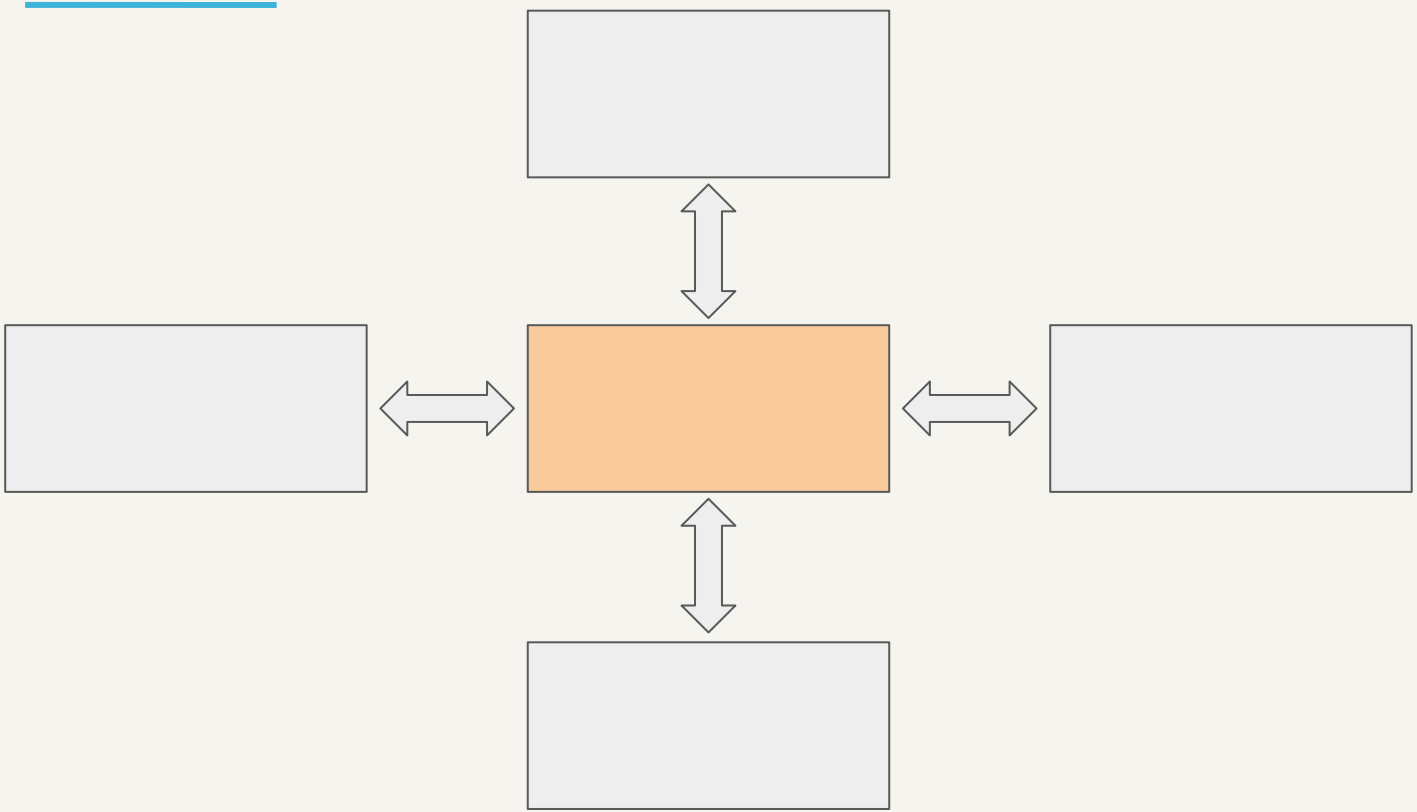
```
// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
  EXPECT_EQ(Factorial(0), 1);
}

// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(Factorial(1), 1);
  EXPECT_EQ(Factorial(2), 2);
  EXPECT_EQ(Factorial(3), 6);
  EXPECT_EQ(Factorial(8), 40320);
}
```

Catch2

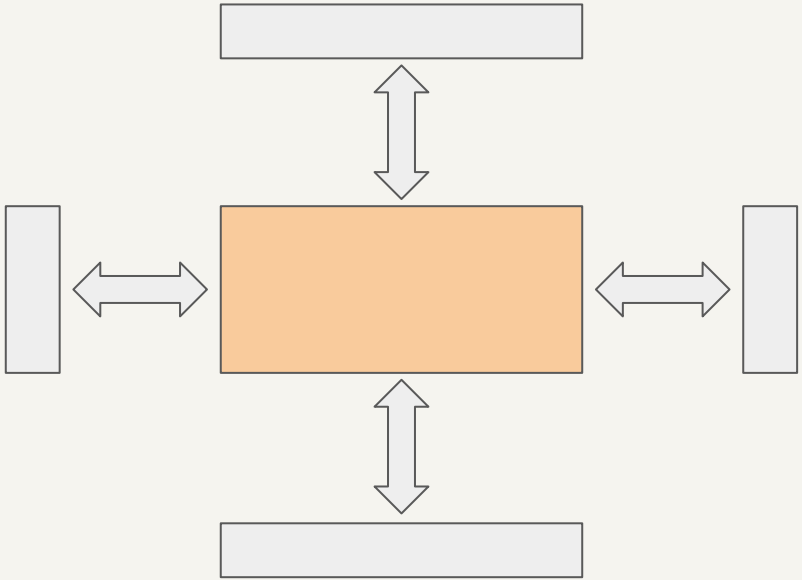
```
#include <catch2/catch_test_macros.hpp>


TEST_CASE( "Factorials are computed", "[factorial]" ) {
    REQUIRE( Factorial(1) == 1 );
    REQUIRE( Factorial(2) == 2 );
    REQUIRE( Factorial(3) == 6 );
    REQUIRE( Factorial(10) == 3628800 );
}
```

undo

MOCKING / DEPENDENCY INJECTION



MOCKING / DEPENDENCY INJECTION



undo

MOCKING / DEPENDENCY INJECTION

```
#include <gmock/gmock.h>

class MockFS : public FS {

 public:

  MOCK_METHOD(int, read, (int fd, void*buf, size_t len), (override));

  MOCK_METHOD(int, write, (int fd, const void *buf, size_t len), (override));

};
```

undo

MOCKING / DEPENDENCY INJECTION

```
using ::testing::Return;

...

EXPECT_CALL(filesystem, read())
    .Times(4)
    .WillOnce(Return(100))
    .WillOnce(Return(-1))
    .WillRepeatedly(Return(200));
```

undo

LOTS AND LOTS OF ASSERTIONS

TEST OR PANIC

If it’s not tested, it doesn’t work.

So decide: write a test case, or panic.

undo

DO NOT SLEEP!

HOW TESTABLE IS YOUR CODE?

- How many configure options do you have?

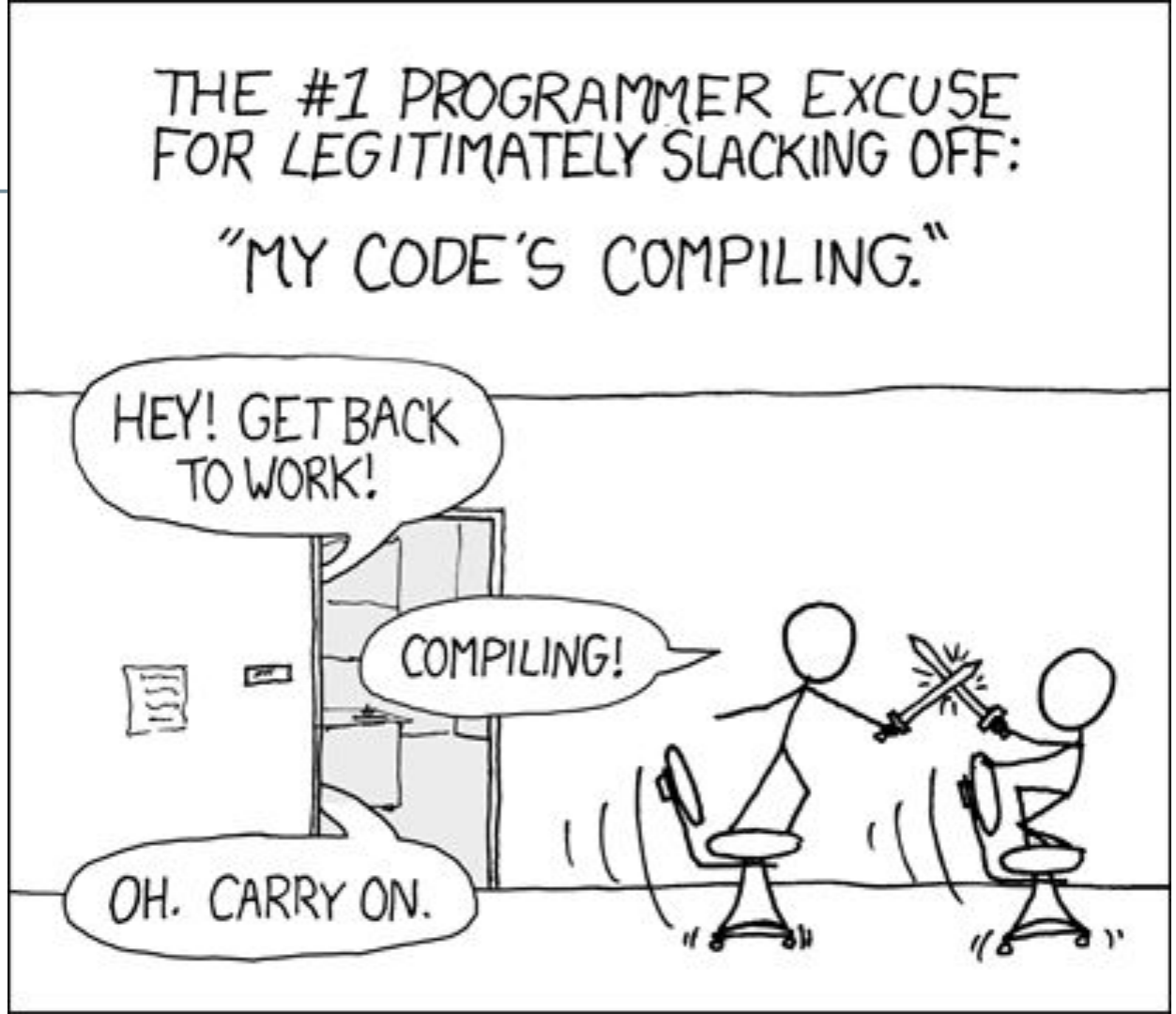
- How deterministic is it?

Most programmers spend most of their time debugging.

Everyone knows that debugging is twice as hard as writing a program in the first place.

So if you're as clever as you can be when you write it, how will you ever debug it?

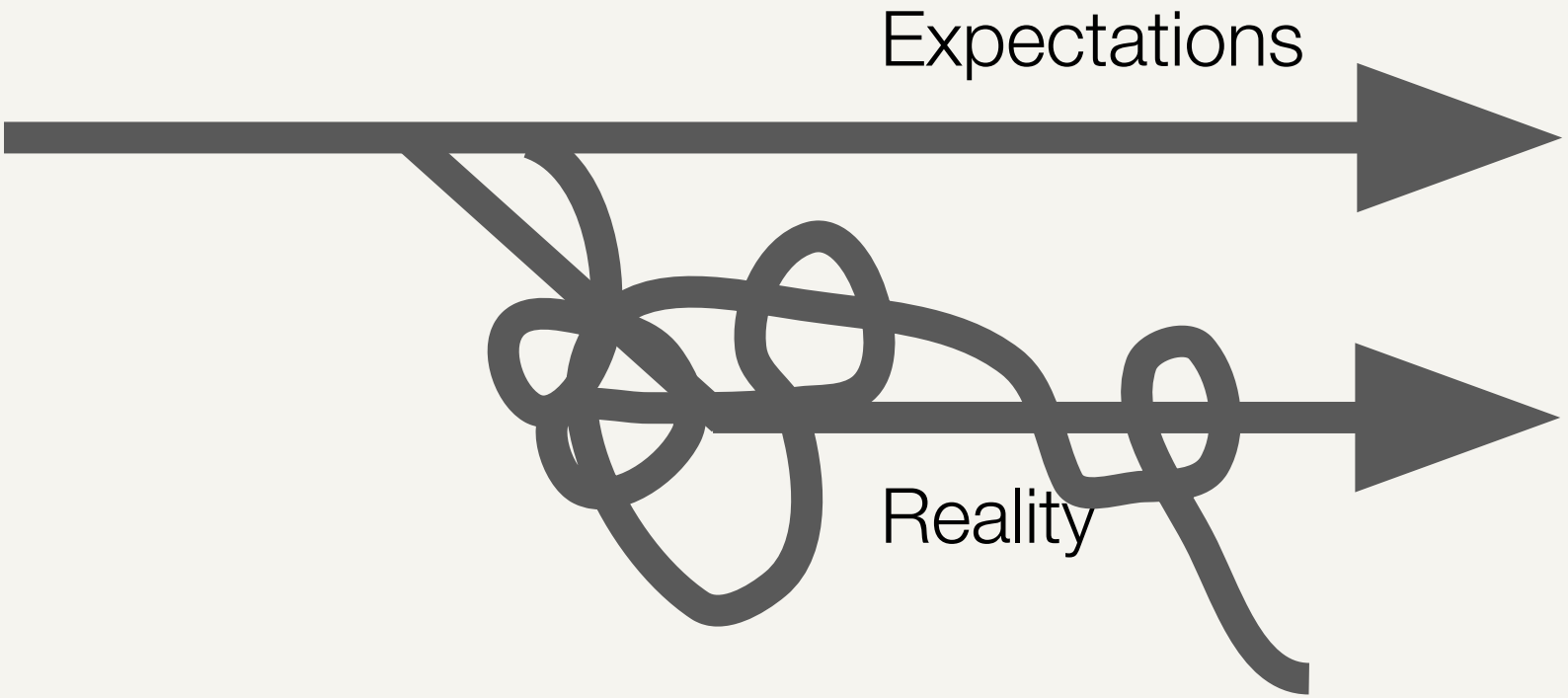
Brian Kernighan


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

What is debugging?

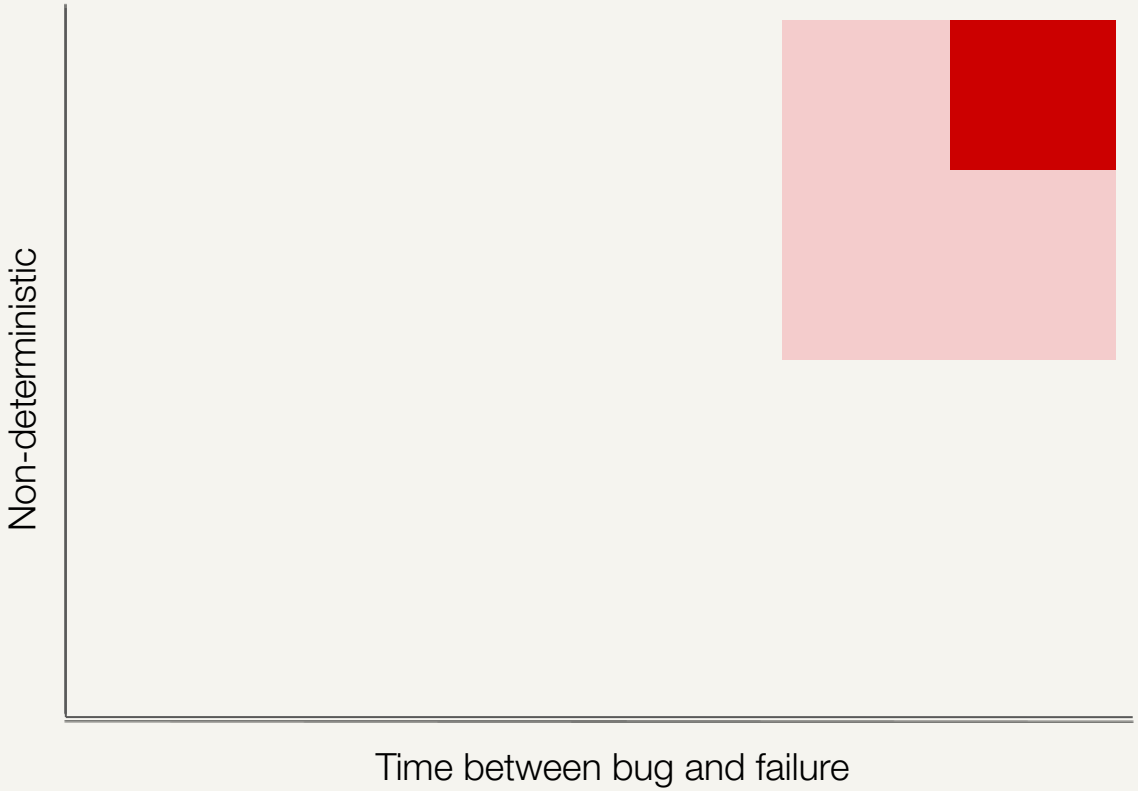
undo

What is debugging?



undo

What makes bugs hard to fix?



‘Heisenbugs’

MANY DIFFERENT KINDS OF BUG

- Logic bugs
- Pointer errors
- Error handling
- Race conditions
- Interface assumptions

- Platform incompatabilities
- Backwards compatibility
- Algorithimic errors
- IO errors

ADVICE

The ‘impossible happened’

An assumption is something that you don’t realise you have made.

WHEN YOU SPELL SMOKE, ACT!

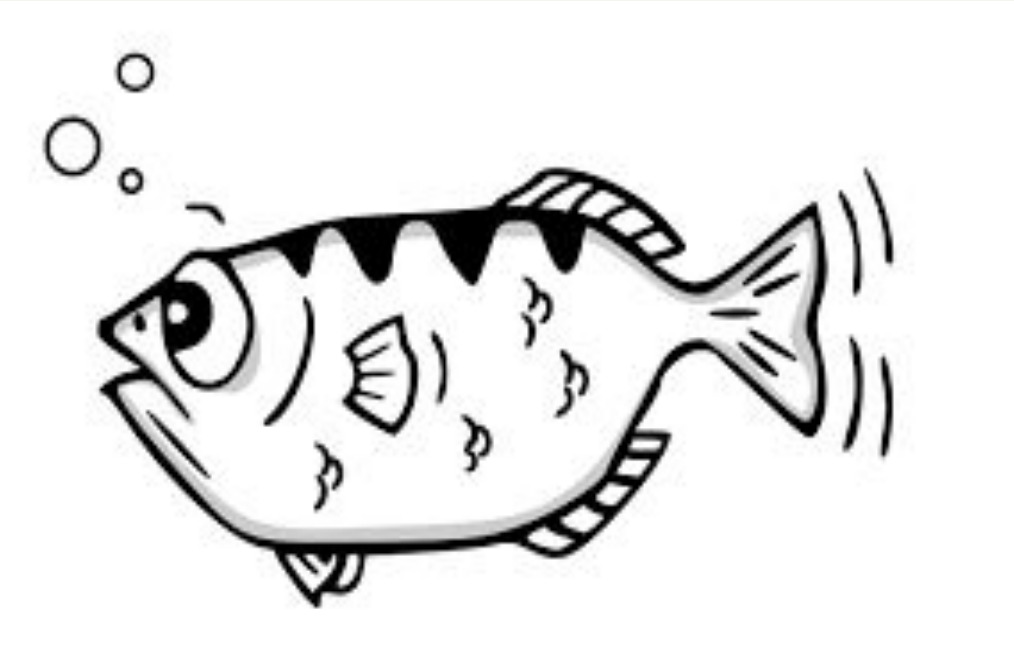
Keep going until you fully understand the root cause

Allow yourself to go down tangents

KNOW THE TOOLS... AND USE THEM!

1. GDB
2. LLDB
3. Valgrind
4. Sanitizers
5. strace & ltrace
6. libc++ debug mode
7. time travel

GDB



GNU Debugger

- TUI mode
- Python integration
- corefiles
- Attach
- Remote
- Pretty printers
- GDB dashboard
- Dynamic printf
- Lots of frontends
  - VS Code, CLion, Emacs, DDD, vimspector, ...

LLDB

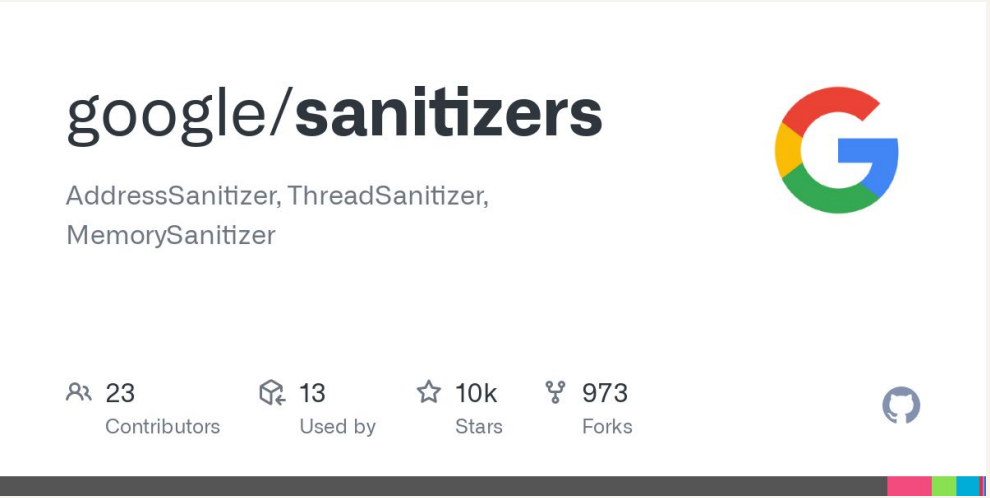
LLVM Debugger

- Like GDB
  - (Except worse and better)
- GUI mode
- Python integration
- Attach
- Remote
- Other frontends
  - X-Code

VALGRIND

- Suite of tools
  - memcheck
  - helgrind & drd
  - cachegrind
  - massif
- No need to recompile
- Slow

AddressSanitizer



- Suite of tools:
  - AddressSanitizer (asan)
  - ThreadSanitizer (tsan)
  - MemorySanitizer (msan)
- Essentially a compiler feature:
  - Much faster runtime
  - Knows more stuff

SO MANY SANITIZERS...

```
address
returns-nonnull-attribute
alignment
bool
bounds
bounds-strict
builtin
enum
vla-bound
```

```
float-cast-overflow
unreachable
float-divide-by-zero
hwaddress
integer-divide-by-zero
kernel-address
kernel-hwaddress
leak
signed-integer-overflow
shift-exponent
thread
```

```
nonnull-attribute
vptr
null
object-size
pointer-compare
pointer-overflow
pointer-subtract
return
shift
shift-base
undefined
```

SO MANY SANITIZERS...

address

returns-nonnull-attribute

alignment

bool

bounds

bounds-strict

builtin

enum

vla-bound

float-cast-overflow

unreachable

float-divide-by-zero

hwaddress

integer-divide-by-zero

kernel-address

kernel-hwaddress

leak

signed-integer-overflow

shift-exponent

thread

nonnull-attribute

vptr

null

object-size

pointer-compare

pointer-overflow

pointer-subtract

return

shift

shift-base

undefined

SO MANY SANITIZERS...

address

returns-nonnull-attribute

alignment

bool

bounds

bounds-strict

builtin

enum

vla-bound

float-cast-overflow

unreachable

float-divide-by-zero

hwaddress

integer-divide-by-zero

kernel-address

kernel-hwaddress

leak

signed-integer-overflow

shift-exponent

thread

nonnull-attribute

vptr

null

object-size

pointer-compare

pointer-overflow

pointer-subtract

return

shift

shift-base

undefined

libc++ debug mode

`gcc:` `GLIBCXX_DEBUG`

`clang:` `_LIBCPP_DEBUG`

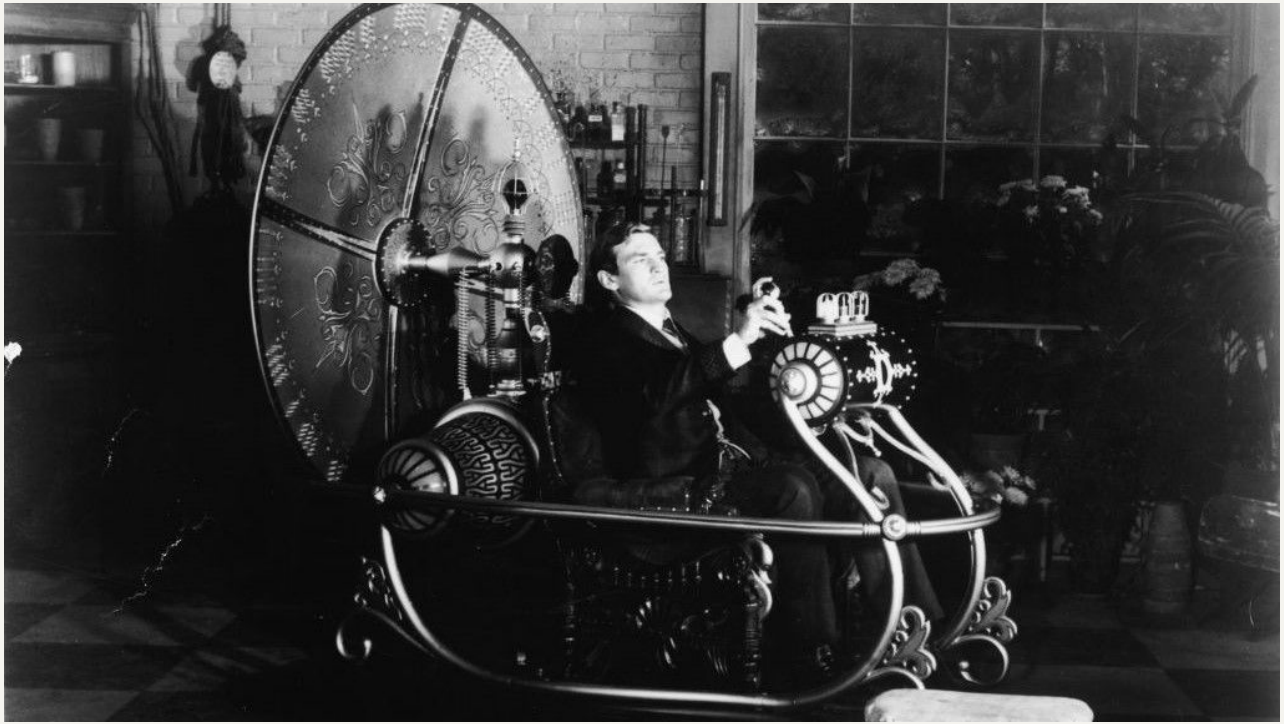
0: Enables most assertions.

1: Enables “iterator debugging”

Container	Header	Debug container	Debug header
`std::bitset`	`bitset`	`__gnu_debug::bitset`	`<debug/bitset>`
`std::deque`	`deque`	`__gnu_debug::deque`	`<debug/deque>`
`std::list`	`list`	`__gnu_debug::list`	`<debug/list>`
`std::map`	`map`	`__gnu_debug::map`	`<debug/map>`
`std::multimap`	`map`	`__gnu_debug::multimap`	`<debug/map>`
`std::multiset`	`set`	`__gnu_debug::multiset`	`<debug/set>`
`std::set`	`set`	`__gnu_debug::set`	`<debug/set>`
`std::string`	`string`	`__gnu_debug::string`	`<debug/string>`
`std::wstring`	`string`	`__gnu_debug::wstring`	`<debug/string>`
`std::basic_string`	`string`	`__gnu_debug::basic_string`	`<debug/string>`
`std::vector`	`vector`	`__gnu_debug::vector`	`<debug/vector>`

STRACE & LTRACE

TIME TRAVEL



TIME TRAVEL

TIME TRAVEL