Back To Basics
Debugging

GREG LAW

Cppcon
The C++ Conference

2023

October 01 - 06

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

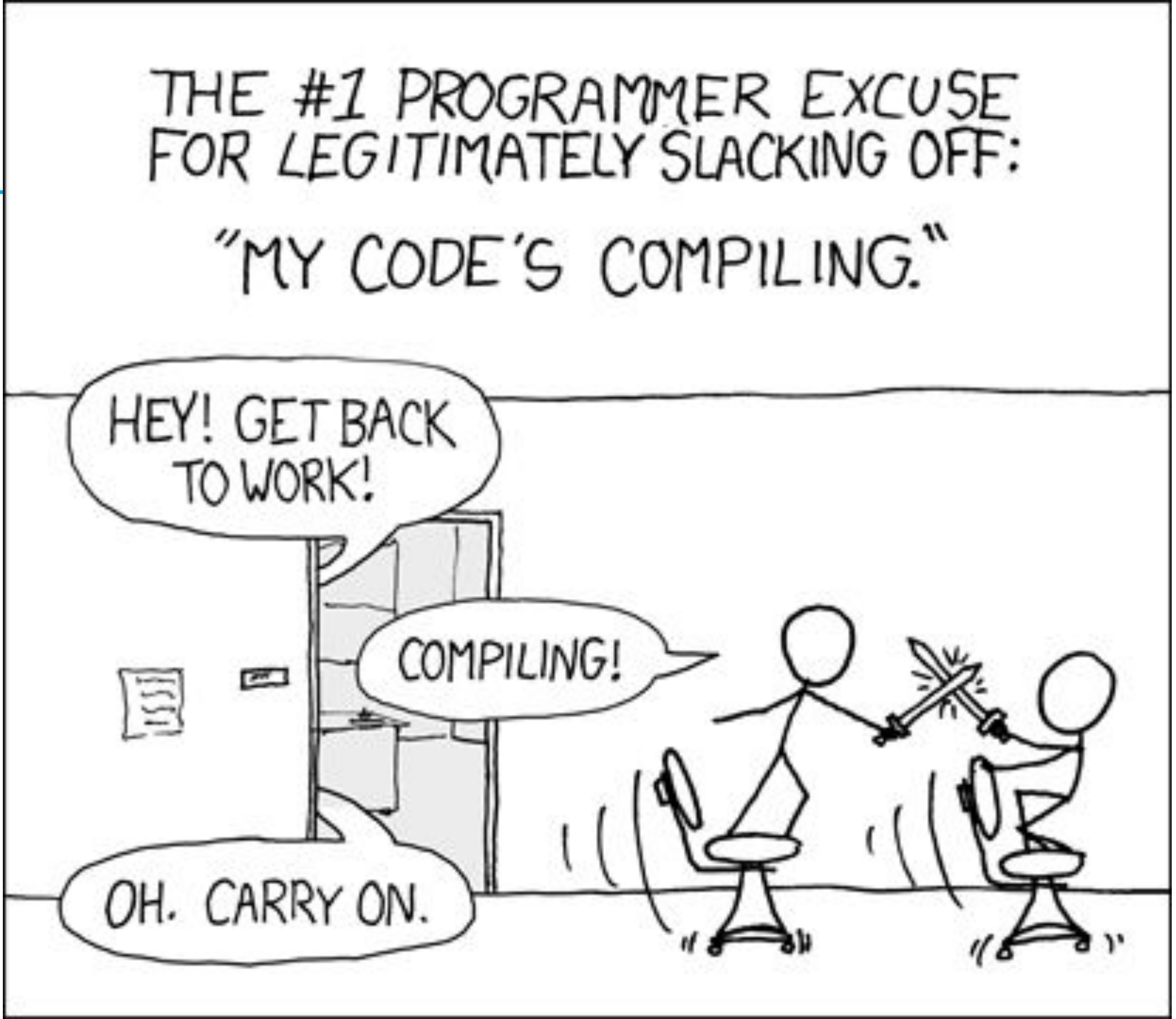
How do we debug?

Use dynamic checkers (e.g. valgrind, ASAN)

Use a debugger (e.g. IntelliJ, GDB)

Dynamic logging / probing (e.g. LightRun)

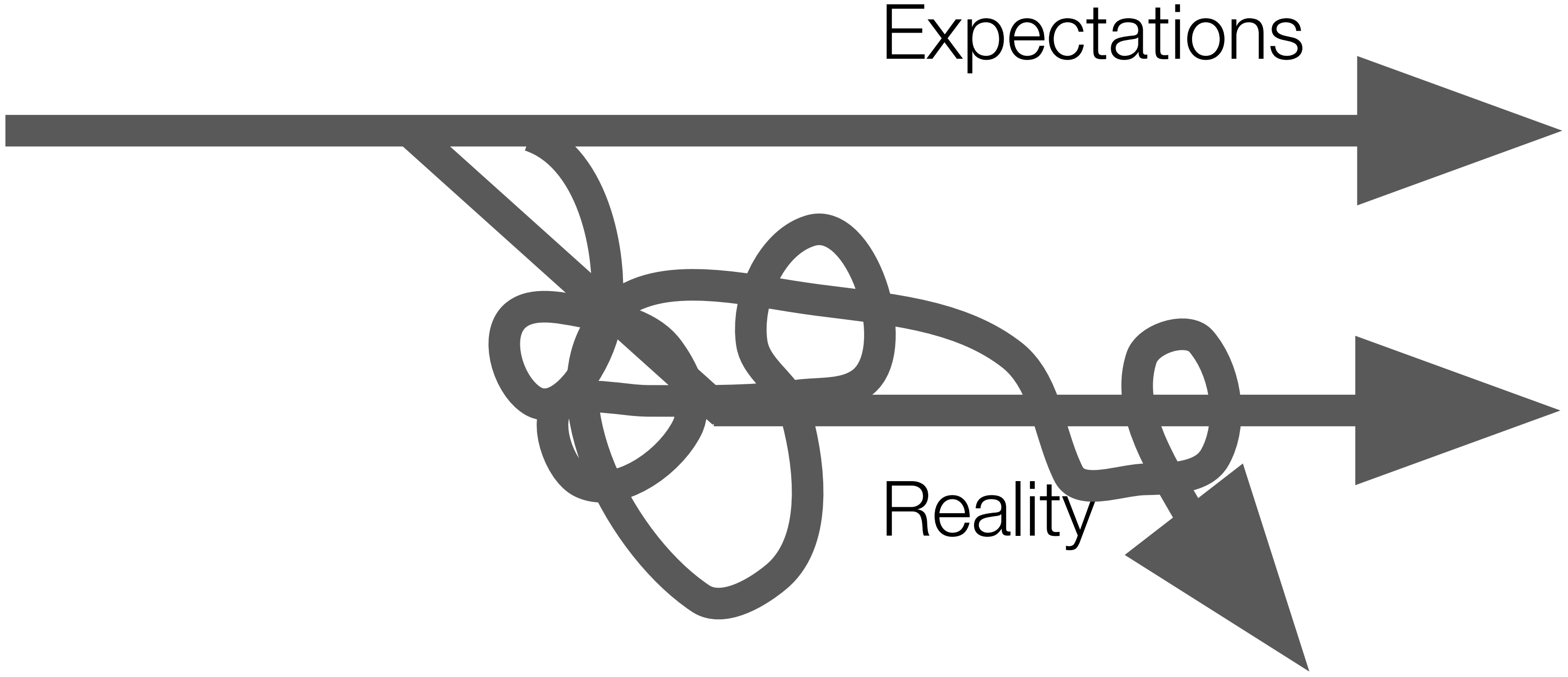
`printf()`


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

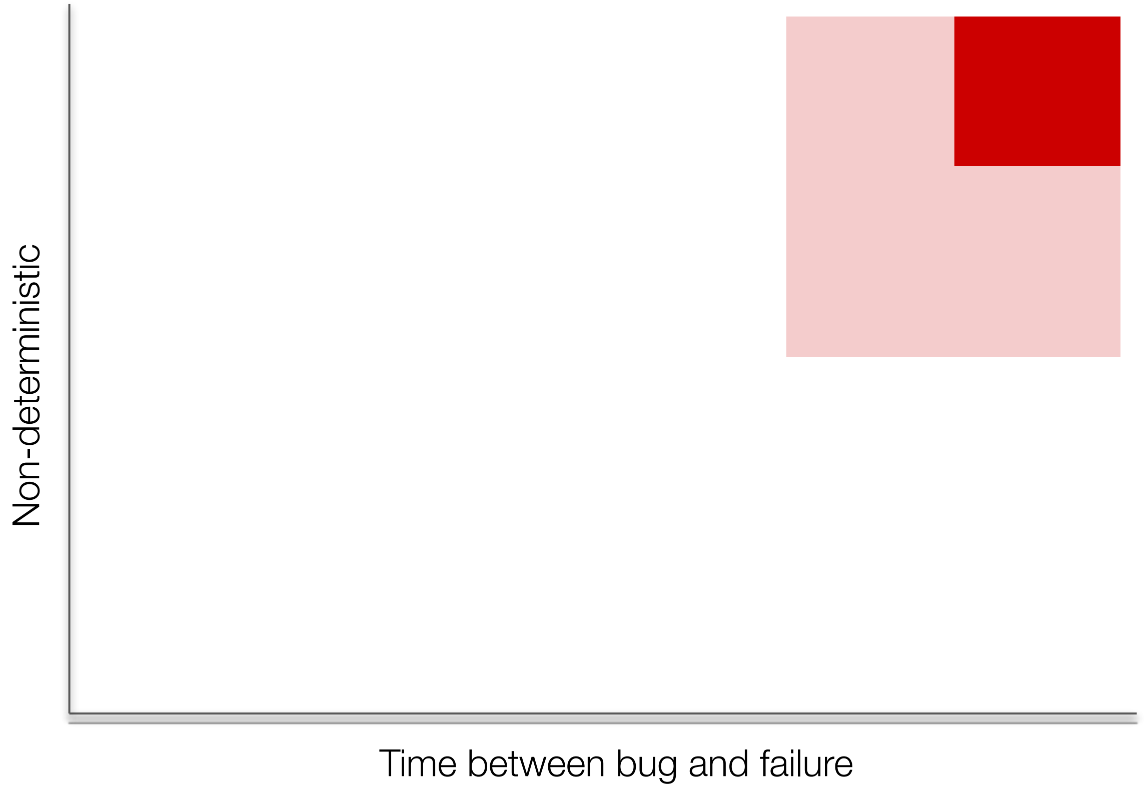
Advice vs tools

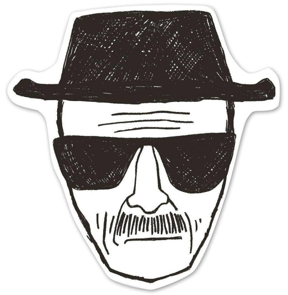
1. General advice.
2. Fantastic tools and where to find them.

Part 0: What is debugging?


Expectations
Reality

What makes bugs hard to fix?





‘Heisenbugs’

Different kinds of bug

- Logic bugs
- Pointer errors
- Error handling
- Race conditions
- Interface assumptions
- Unitialised variables
- Conversions
- Undefinited behaviour
- Architecture differences

undo™

Part 1: Advice

The ‘impossible happened’

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

When you smell smoke, act

Keep going until you fully understand the root cause

Allow yourself to go down tangents

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.

The final piece of advice

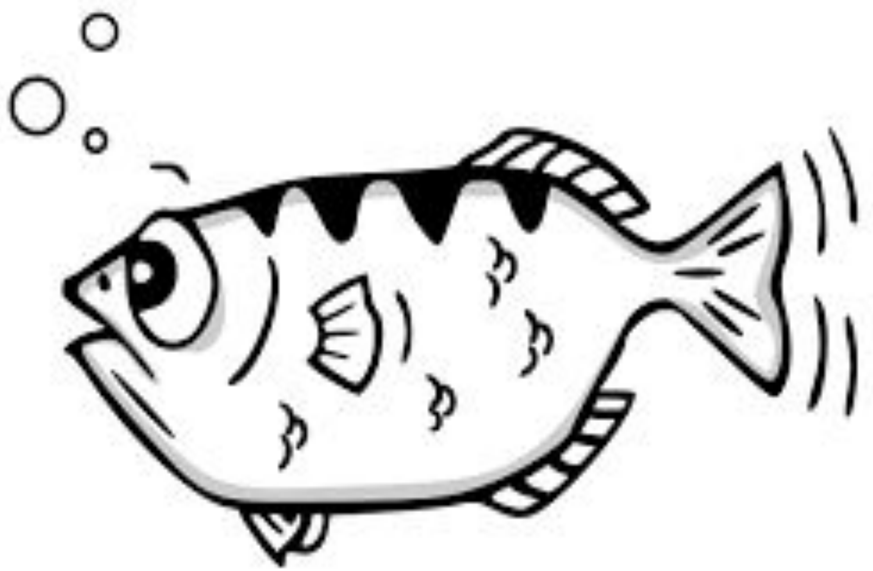
Use the tools!

Part 2

So many tools

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

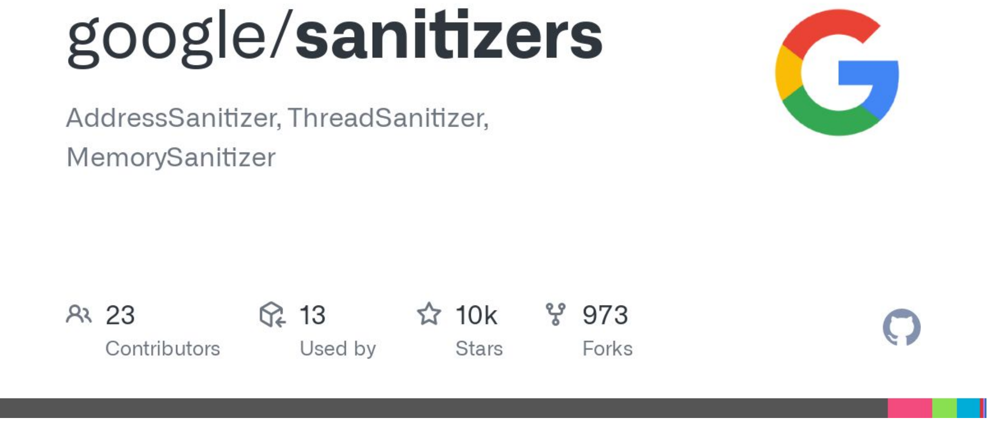
- Like GDB
  - (Except worse and better)
- GUI mode
- Python integration
- Attach
- Remote
- Other frontends
  - X-Code

undo

Valgrind

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

AddressSanitizer



google/**sanitizers**

AddressSanitizer, ThreadSanitizer, MemorySanitizer

23 Contributors · 13 Used by · 10k Stars · 973 Forks

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

undo

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`	`float-cast-overflow`	`nonnull-attribute`
`returns-nonnull-attribute`	`unreachable`	`vptr`
`alignment`	`float-divide-by-zero`	`null`
`bool`	`hwaddress`	`object-size`
`bounds`	`integer-divide-by-zero`	`pointer-compare`
`bounds-strict`	`kernel-address`	`pointer-overflow`
`builtin`	`kernel-hwaddress`	`pointer-subtract`
`enum`	`leak`	`return`
`vla-bound`	`signed-integer-overflow`	`shift`
	`shift-exponent`	`shift-base`
	`thread`	**`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