No Problem.
Lua & Lua C API

- Lua
  - stateful, stack-based
  - well-documented

- Lua C API
  - mostly clear semantics / mappings
Limitations of Lua C API

- Stack-Based
  - Hard to grok sometimes
  - Must clean up or following operations will overflow the stack

- Simple in Lua ≠ Simple in API
  - Incredible amounts of boilerplate
  - *Efficient* stack management is hard
Lua C API can do Simple Things

- `my_table[“a”]`
  - get ‘my_table’ global
  - get field
  - `lua_to{x}` value

- `my_func(2)`
  - push `my_func` global function
  - push argument
  - call, get return(s)
other_func(
    my_table["a"]['b'], my_func(2)
)

Lua C API does not scale
- amount of necessary boilerplate
- developer time
Limitations of C

- No overloading
  - “which one do I need, again?”
  - Hard to specialize general-purpose routines

```c
lua_gettable()
lua_getglobal(const char*)
lua_getfield(const char*)
lua_geti(int) [5.3+]

lua_rawgeti(int)
lua_rawget()
lua_rawgetp(void*)
```
Okay... so we wrap it?

- Type tells us what we need to do
  - Overloading/Dispatching to cover up the base
  - Stuff implementation details into various functions
More MeatPower

- Higher-level, complex operations
  - Calling a function
    - with complex arguments
  - Tables
    - with nested lookup
  - Structured data
    - Mimicking C, C++ structures
Sol2

- Started by Danny Y. “Rapptz”
  - Unmaintained because he has other great ideas
  - Pull requests sitting dead in repository

- Rewritten, developed into Sol2
Disclaimers

- I’m the author of sol2
- I did not author the 12 other benchmarked libraries
  - E-mailed every single library author, however
  - All of them got back to me with proper usage notes

- Great benchmarking technology
  - nonius: https://nonius.io/
  - statistically-significant benchmarking
  - much better than my hand-rolled loops
sol::stack

- The core of the API; usually never seen

```c++
lua_State* L = ...;
sol::stack::get_field<true>(L, "some_key");
int the_value = sol::stack::get<int>(L, -1);
lua_pop(L, 1);

lua_createtable(L, 0, 2);
sol::stack_reference ref(L, -1);
sol::stack::set_field(L, 1, "val1");
sol::stack::set_field(L, 2, "val2", ref.stack_index());
ref.pop();
```

Basics

- Demonstrating some basics
  - Load a config file, mess with it

```lua
config.lua

number = 24
number2 = 24.5
important_string = 'woof woof'
some_table = { value = 48 }
function bark (val)
    print(val .. ' waf waf!')
end
```
Basics - tables

```cpp
sol::state lua;
lua.open_libraries(sol::lib::base);
lua.script_file("config.lua");

int number = lua["number"];  
std::string important_string = lua["important_string"];  
int value = lua["some_table"]["value"];  

sol::optional<int> safe = lua["this_is"]["not_real"];  
int default_value = safe ? safe.value() : 24;  // 24
```

Basics - functions

```cpp
sol::function bark = lua["bark"];  // woof woof waf waf!
bark(lua["important_string"]);  

lua["woof"] = [](std::cout << "Hey there!" << std::endl);  

lua.script("woof()");  // prints "Hey there!"
```

- Very easy to use
  - Painless to set up
  - Can be used without `sol::state;` just `lua_State*`

usertype

- The Big One™ - best part of Sol2
  - member function/variable bindings
  - metamethod
    - automatically generated equality/comparison methods
  - properties (like luabind)!
  - static functions as member functions
    - Take self argument
  - static variables, functions
  - (simple_usertype) runtime extensible

usertype – a live example
Implementation - functions

userdata

metatable of functions

__(new)index: itself
Implementation - variables

userdata →metatable of functions →__(new)index: lua_CFunction
Implementation – variables, speed

userdata → pass-off table

__(new)index: second table

metatable with functions → __(new)index: lua_CFunction
Can’t use the speed method
- userdata not ‘failed lookup’ item
- metatable is the ‘failed lookup’ item
- 2x-4x performance hit for ALL methods/variables

Karel Tuma patched item in his LuaJIT fork

metatable-per-userdata?
“I think it’s better than Selene”

- Shohnwal, March 21, 2016
- Sol2 had better support at the time
  - Failure to communicate, so improved: [http://sol2.rtfd.io](http://sol2.rtfd.io)

https://github.com/ThePhD/sol2/issues/36
Benchmarks

“To be honest with you, Sol2 is the first binding library I have compared against where I have had to disable runtime checks in OOLua”
– Liam Devine, OOLua,
https://github.com/ThePhD/sol2/issues/156#issuecomment-236913783

https://github.com/ThePhD/lua-bench
Lua wants

- \_\_index/\_\_newindex extra argument fix
  - add the original userdata / table that triggers the whole lookup cascade as the last argument
  - keeps backwards compatibility, enable efficient member function lookup
- New GC
  - corsix is on it with LuaJIT!
Thanks To

- Professor Gail E. Kaiser
  - COMS E6156 – Advanced Software Engineering

- Iris Zhang
  - Vetted documentation

- Kevin Brightwell (😈 : Nava2)
  - Took great interest in sol2 before anyone else
  - Vastly improved the CI
    - [https://travis-ci.org/ThePhD/sol2](https://travis-ci.org/ThePhD/sol2)
Thanks To

- Lounge<C++>

- Elias Daler (@EliasDaler), Eevee (@eevee)

- Jason Turner (@lefticus)
  - Encouraged me to present, talk about Sol2
  - Runs CppCast (http://cppcast.com)
Thank You!

- Questions and/or Comments?
  - If you end up using Sol2, tell me about it here: https://github.com/ThePhD/sol2/issues/189

- Thoughts about Future Direction?

- Concerns?
  - .... Lunch?~
Bug Hunting

• “The road to success in Software Development is paved with the tears of your failed tests and the sleepless nights over your Heisenbugs.” - Some Poor Developer
Lua

- Very few actual bugs in the implementation, except...!
- Investigating one now
  - Compile with C++
  - pcall from a C function that throws an exception
    - returns -1 (not a defined error)
    - does not even clean stack?
Clang

- “internal linkage” bugs
- Excessively pedantic
  - “condition is the result of a constant”
    - it’s a template argument, clang, please stop torturing me with all these warnings :<
- apple-clang’s only purpose is to literally introduce new strange, build-breaking, progress-stopping bugs
  - negative value on enum breaks demangler
  - forced us to parse from __PRETTY_FUNCTION__
VC++ (Visual Studio)

- Help
- Me...!
GCC

- Less compiler bugs
  - `auto&&` in lambda declaration
- More actual unsupported features
  - `has_*` vs. `is_*` trait debacle
  - extended `constexpr` not backported to GCC 4.x.x