
ACCU 2021
VIRTUAL EVENT
Bloomberg
Engineering
undo
mosaic
CONSULTANTS TO FINANCIAL SERVICES
C++20 + Lua = Flexibility
James Pascoe

C++20 + LUA = FLEXIBILITY

Jim (James) Pascoe
http://www.james-pascoe.com
james@james-pascoe.com

http://jamespascoe.github.io/accu2021
https://github.com/jamespascoe/LuaChat.git

OVERVIEW

A Tutorial on Combining C++20 and Lua 5.4.2
Up-to-date practical advice with code

- Why Combine C++ and Lua?
  - Example: High Speed Transport
- How to Combine C++ and Lua
  - Sol3 and Swig 4.0.2
- Benchmarking and Concurrency
  - Coroutines and performance

WHY COMBINE C++ AND LUA?

Flexibility Post Release

- Behaviour can be modified after code is shipped
  - Cope with future unknowns proactively
- Modifications are fast
  - No compile, package, deploy cycle
- Barrier to entry is much lower for Lua
  - Appeals to FAEs, Architects, Customers


HIGH SPEED
TRANSPORT EXAMPLE

BLU WIRELESS

- IP networking over 5G mmWave (60 GHz) modems
  - 802.11ad MAC + PHY (Hydra) + software
- High-bandwidth, low latency mobile Internet
  - Up to 3.5Gbps wireless links (up to 1km)
- Embedded quad-core ARMv8 NPUs
  - Track-side / train-top mmWave radios

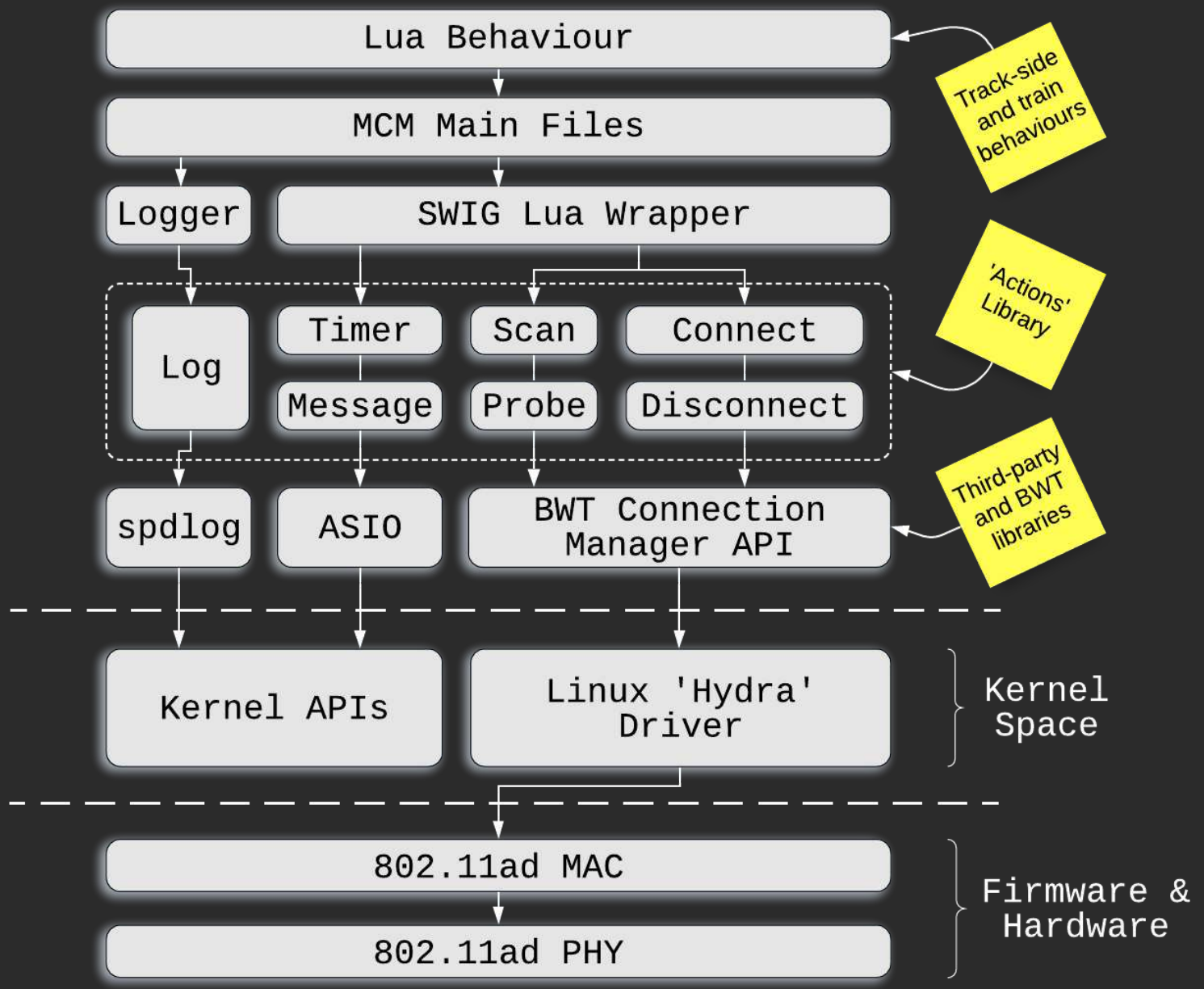

Train Top
In Train
Track Side

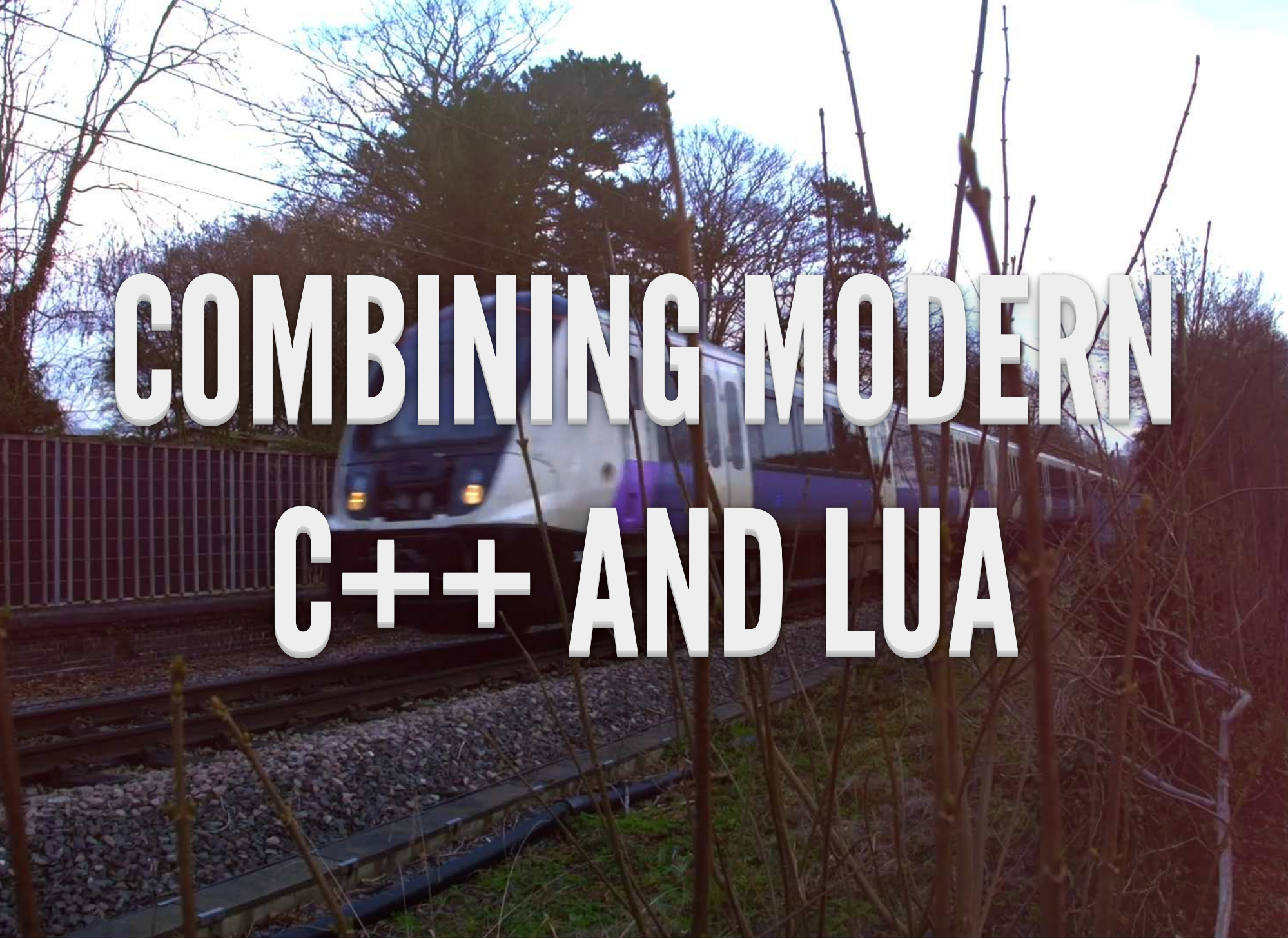
CONNECTION MANAGEMENT

- Mission critical software component
- Decides which radio to connect to and when
- v1.0 fixed behaviour: connect to strongest signal
- Anomalies led to poor performance
- Software updates were costly
- Improvements could not be made fast enough

MOBILE CONNECTION MANAGER

- Complete redesign
- Decoupled architecture (C++17 & Lua 5.4.2)
- Actions (C++): 'Scan', 'Connect', 'Probe' etc.
- Behaviours (Lua): implement 'beam choreography'
- Changes can be made in the field by FAEs
- Consolidated into supported releases


Lua Behaviour
MCM Main Files
Logger
SWIG Lua Wrapper
Log
Timer
Message
Scan
Probe
Connect
Disconnect
spdlog
ASIO
BWT Connection Manager API
Kernel APIs
Linux 'Hydra' Driver
802.11ad MAC
802.11ad PHY
Track-side and train behaviours
'Actions' Library
Third-party and BWT libraries
Kernel Space
Firmware & Hardware


COMBINING MODERN
C++ AND LUA

LUA

- Lightweight embeddable scripting language
- Dynamically typed, runs by interpreting bytecode
- Simple procedural syntax
- Emphasis on meta-mechanisms
- Instant appeal for Architects, FAEs etc.

THE LUA C API

- Lua communicates with C++ through a virtual stack
- Strict stack discipline is not enforced
  - indices ≥ 1 are positions from the bottom
  - negative indices are relative to the top
- Pseudo indices for the Lua Registry and Upvalues
- Compile with LUA_USE_APICHECK to enable checks

LUA C API: LUA

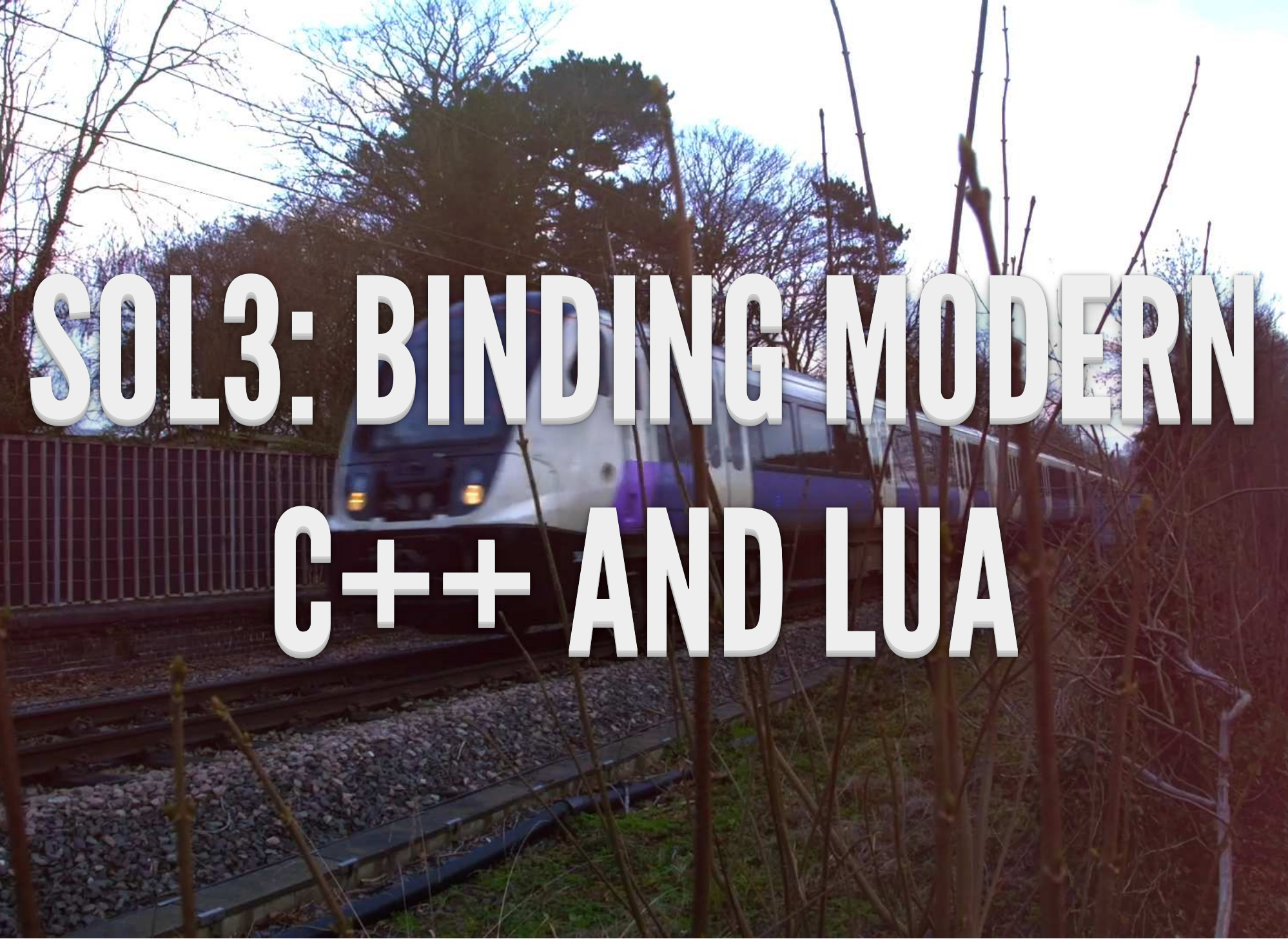
```lua
-- Create a global table 't'
t = { x=1, y=2 }

function f (str, val, int)
  print(
    string.format(
      "Lua: f called with args: %s %d %d", str, val, int
    )
  )

  -- Call a C++ function
  local rc = cppFunc(str, t.y, int)

  return rc
end
```

LUA C API: C++

```cpp
#include <iostream>
#include "lua.hpp"

int cppFunc(lua_State *L) {
  std::cout << "cppFunc called with args:" << std::endl;

  for (int n=1; n <= lua_gettop(L); ++n)
    std::cout << lua_tostring(L, n) << std::endl;

  return 0;
}

int main([[maybe_unused]] int argc, char ** argv)
{
  // Create a new lua state
  lua_State *L = luaL_newstate();

  // Open all libraries
  luaL_openlibs(L);

  // export a C++ function to Lua
  lua_register(L, "cppFunc", cppFunc);
```

LUA C API: BUILD & RUN

```
> brew install lua # sudo apt-get -y install lua5.4
> clang++ -std=c++2a -llua -o lua-cpp lua-cpp.cpp
> ./lua-cpp lua-cpp.lua
Lua: f called with args: how 1 14
cppFunc called with args:
how
2
14
```


SOL3: BINDING MODERN
C++ AND LUA

SOL3

- Sol: Danny Rapptz
  - Last commit: 2015
- Sol2/3 (Sol2 v3): JeanHeyd Meneide (ThePhD)
  - Active since 2013, 100+ contributors
- Modern C++ Binding for Lua
  - Header only, fast
  - Support for Modern C++ types
  - Nice upgrade path

SOL3: STACK MANIPULATION

```cpp
#include <iostream>

#define SOL_ALL_SAFETIES_ON 1
#include <sol/sol.hpp>

int cppFunc(lua_State *L) {
  std::cout << "cppFunc called with args:" << std::endl;

  for (int n=1; n <= lua_gettop(L); ++n)
    std::cout << lua_tostring(L, n) << std::endl;

  return 0;
}

int main([[maybe_unused]] int argc, char ** argv)
{
  // Create a new lua state and open libraries
  sol::state lua;
  lua.open_libraries(sol::lib::base, sol::lib::string);

  // Export a C++ function to Lua
  lua["cppFunc"] = cppFunc;
```

SOL3: IMPROVED EXAMPLE

```cpp
#include <iostream>

#define SOL_ALL_SAFETIES_ON 1
#include <sol/sol.hpp>

int cppFunc_oneline(std::string str, int a, int b) {
  std::cout << "cppFunc_oneline called with args: " <<
    str << " " << a << " " << b << std::endl;

  return 0;
}

int main([[maybe_unused]] int argc, char ** argv[1]);
{
  // Create a new lua state and open libraries
  sol::state lua;
  lua.open_libraries(sol::lib::base, sol::lib::string);

  // Export a C++ function to Lua
  lua["cppFunc"] = cppFunc_oneline;

  // Load and run the Lua file
```

SOL3 EXAMPLE: BUILD & RUN

```
> brew install lua # sudo apt-get -y install lua5.4
> git clone https://github.com/ThePhD/sol2.git
> clang++ -std=c++2a -I sol2/include -llua -o lua-sol3 lua-sol3.cpp
> ./lua-sol3 lua-cpp.lua
Lua: f called with args: how 1 14
cppFunc called with args:
how
2
14
```

```
> clang++ -std=c++2a -I sol2/include -llua -o lua-sol3 lua-sol3-ol.cpp
> ./lua-sol3 lua-cpp.lua
Lua: f called with args: how 1 14
cppFunc_oneline called with args: how 2 14
```

SOL3: CONTAINER EXAMPLE

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <chrono>
#include <utility>

#define SOL_ALL_SAFETIES_ON 1
#include <sol/sol.hpp>

#include "date.h"

class timestamped_messages {

  using ts_msg = std::pair<std::string, std::string>;
  using ts_msg_vec = std::vector<ts_msg>;
  ts_msg_vec ts_msgs;

public:

  using value_type = ts_msg_vec::value_type;
  using iterator = ts_msg_vec::iterator;
```

SOL3 CONTAINER EXAMPLE: BUILD & RUN

```
> git clone https://github.com/HowardHinnant/date.git
> git clone https://github.com/ThePhD/sol2.git
> clang++ -std=c++2a -I sol2/include/ -I date/include/date -l lua -o
> ./container
Lua: 21:35:10.437971 msg 1
Lua: 21:35:10.438393 msg 2
Lua: 21:35:10.438403 msg 3
C++: 21:35:10.437971 msg 1
C++: 21:35:10.438393 msg 2
C++: 21:35:10.438403 msg 3
```

NEXT STEPS

- What other features does Sol3 support?
  - state_view: non-owning access to a lua_State*
  - optionals, callables, user-types, concurrency
  - lots more - feature matrix available here
- Customisation Traits
  - Containers, reference-counted resources, UDTs
- Further examples
  - Comprehensive selection in examples directory

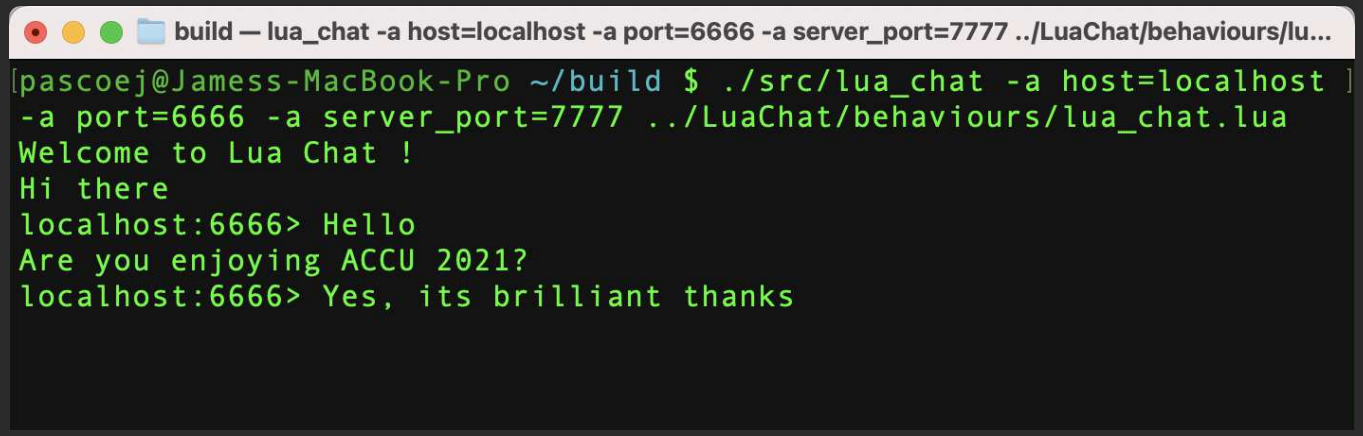

SWIG AND LUACHAT

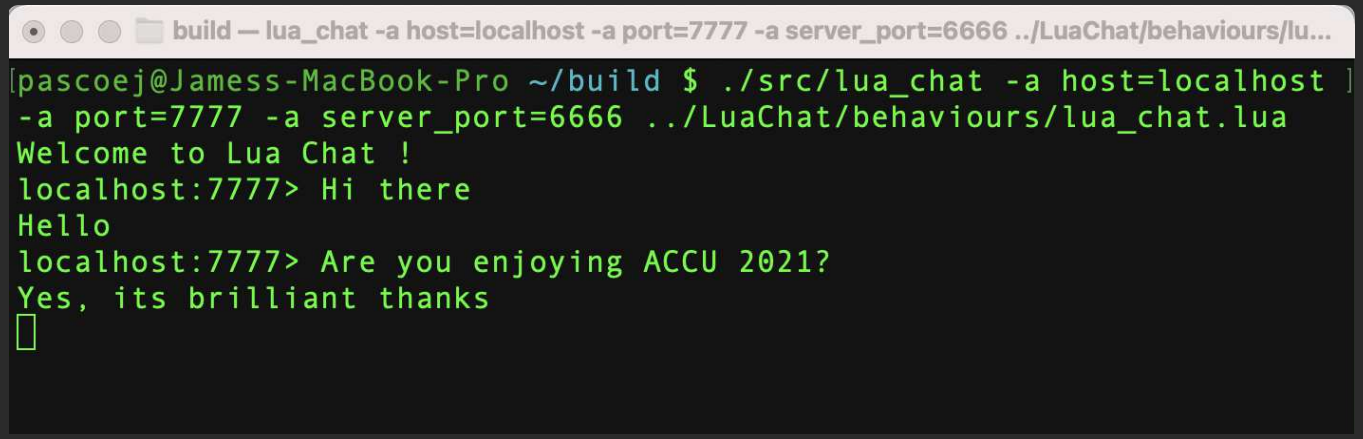
SWIG

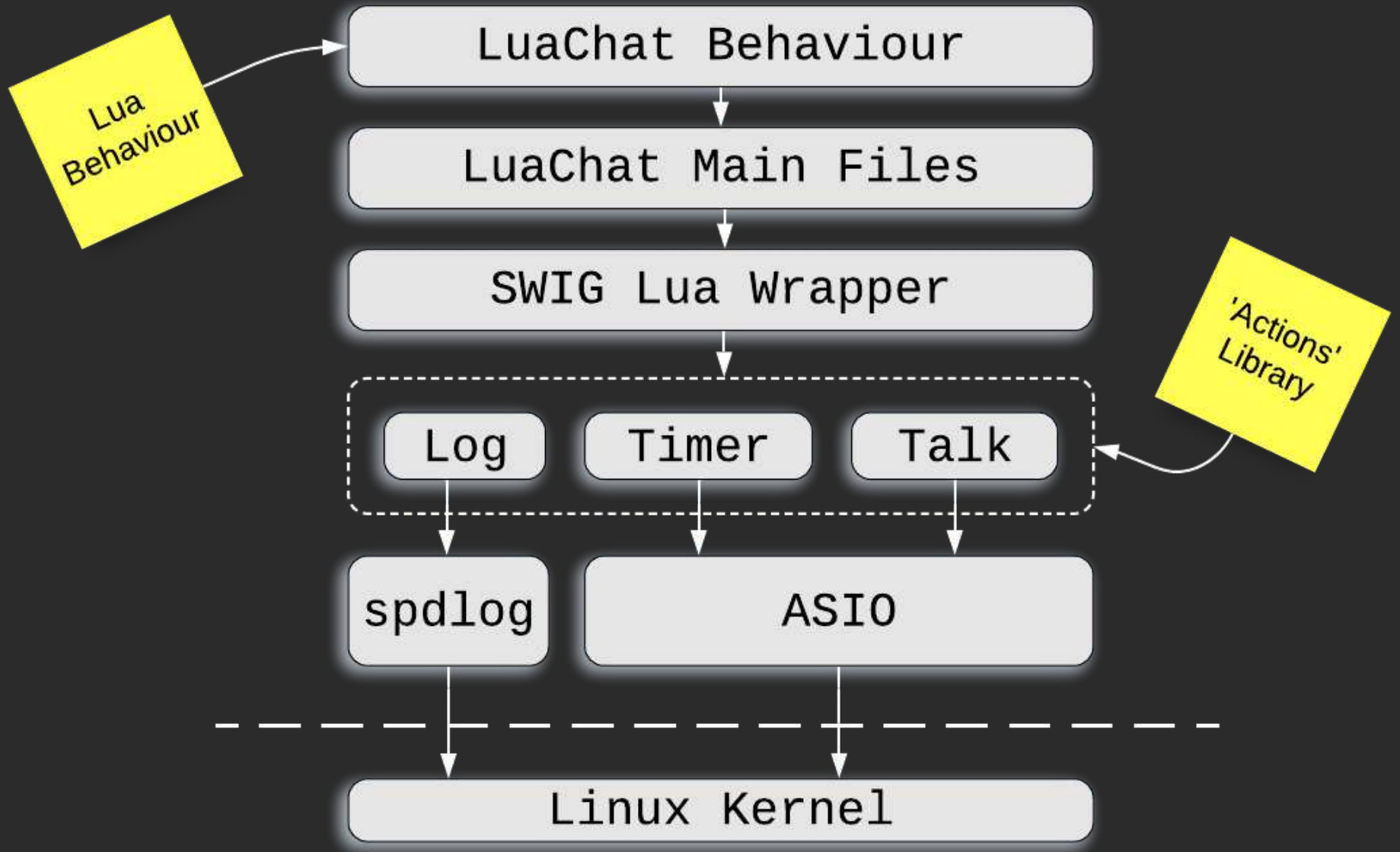
- Simplified Wrapper and Interface Generator
- Produces C++ bindings for many target languages
- Generates Lua stack calls for std C++ types
  - std::string, std::vector, std::map etc.
- C++20 types can be supported with typemaps
- Integrates well with CMake

LUACHAT

- Unix 'talk' program (written in C++17 & Lua 5.3/5.4)
- Available on GitHub (MIT license)
- Asio for asynchronous TCP and timers
- spdlog for logging, cxxopts for command line processing and CMake for build generation


build — lua_chat -a host=localhost -a port=6666 -a server_port=7777 ../LuaChat/behaviours/lu...
[pascoej@Jamess-MacBook-Pro ~/build $ ./src/lua_chat -a host=localhost
-a port=6666 -a server_port=7777 ../LuaChat/behaviours/lua_chat.lua
Welcome to Lua Chat !
Hi there
localhost:6666> Hello
Are you enjoying ACCU 2021?
localhost:6666> Yes, its brilliant thanks


build — lua_chat -a host=localhost -a port=7777 -a server_port=6666 ../LuaChat/behaviours/lu...
[pascoej@Jamess-MacBook-Pro ~/build $ ./src/lua_chat -a host=localhost
-a port=7777 -a server_port=6666 ../LuaChat/behaviours/lua_chat.lua
Welcome to Lua Chat !
localhost:7777> Hi there
Hello
localhost:7777> Are you enjoying ACCU 2021?
Yes, its brilliant thanks


Lua Behaviour
LuaChat Behaviour
LuaChat Main Files
SWIG Lua Wrapper
'Actions' Library
Log
Timer
Talk
spdlog
ASIO
Linux Kernel

BUILD INSTRUCTIONS

Ubuntu 18.04 (Linux Mint 19):

```
1 git clone https://github.com/jamespascoe/LuaChat.git
2 sudo apt-get -y install lua5.3 lua5.3-dev luarocks swig
3 sudo luarocks install luaposix
4 mkdir build; cd build; cmake ../LuaChat; make
5 ./src/lua_chat
```

MacOS (Big Sur):

```
1 git clone https://github.com/jamespascoe/LuaChat.git
2 brew install lua luarocks swig
3 luarocks install luaposix
4 mkdir build; cd build; cmake ../LuaChat; make
5 ./src/lua_chat
```

LUACHAT SWIG CMAKE

```
set(LUA_CHAT_SWIG_SRCS
    lua_chat_actions.i lua_chat_action_log.cpp)

set_source_files_properties(${LUA_CHAT_SWIG_SRCS}
                            PROPERTIES CPLUSPLUS ON)

swig_add_library(actions TYPE USE_BUILD_SHARED_LIBS
                 LANGUAGE lua
                 SOURCES ${LUA_CHAT_SWIG_SRCS})

target_include_directories(actions PRIVATE
                           ${CMAKE_CURRENT_SOURCE_DIR}
                           ${LUA_CHAT_SOURCE_DIR}/src
                           ${LUA_CHAT_SOURCE_DIR}/third_party
                           ${LUA_INCLUDE_DIR})

target_compile_definitions(actions PRIVATE ASIO_STANDALONE)

target_link_libraries(actions PRIVATE std::filesystem)
```

LUACHAT SWIG INPUT

```
 1 %module Actions
 2
 3 %include <std_string.i>
 4
 5 // Definitions required by the SWIG wrapper to compile
 6 %{
 7 #include "lua_chat_log_manager.hpp"
 8 #include "lua_chat_action_log.hpp"
 9 #include "lua_chat_action_talk.hpp"
10 #include "lua_chat_action_timer.hpp"
11 %}
12
13 // Files to be wrapped by SWIG
14 %include "lua_chat_action_log.hpp"
15
16 %define CTOR_ERROR
```

TYPEMAPS

- Maps C++ types onto types in the target language
- We can add support for Modern C++ abstractions
  - E.g. callbacks: Lua functions → std::function
- Acknowledgement: thanks to Petar Terziev for the original version of the following example

LUA CALLBACK: SWIG TYPEMAP

```
%typemap(typecheck) Example::Callback & {
  $1 = lua_isfunction(L, $input);
}

%typemap(in) Example::Callback & (Example::Callback temp) {
  // Create a reference to the Lua callback
  SWIGLUA_REF fn;
  swiglua_ref_set(&fn, L, $input);

  temp = [&fn]() {
    swiglua_ref_get(&fn);

    lua_pcall(fn.L, 0, 0, 0);
  };

  $1 = &temp;
}

// %include source files AFTER typemap declarations
```


ACTIONS

LUACHAT ACTIONS

- Talk: sends messages to a remote LuaChat
  - Based on Asio - must also act as a server
  - Use TCP for fault-tolerant in-order delivery
  - One asynchronous TCP connection per message
- Timer: implements blocking and non-blocking waits
  - Use Asio - required for Lua coroutine dispatcher
- Log: wraps spdlog primitives

TCP CONNECTIONS

```
 1 class tcp_connection
 2 {
 3 public:
 4   using pointer = std::shared_ptr<tcp_connection>
 5
 6   static pointer create(asio::io_context& io_context) {
 7     return pointer(new tcp_connection(io_context));
 8   }
 9
10   asio::ip::tcp::socket& socket() { return m_socket; }
11
12   std::string& data() { return m_data; }
13
14 private:
15   tcp_connection(asio::io_context& io_context)
16     : m_socket(io_context) {}
17
18   asio::ip::tcp::socket m_socket;
19   std::string m_data;
20 };
```

CONNECTION HANDLING

```
1  Talk::Talk(unsigned short port)
2      : m_acceptor(m_io_context,
3                   tcp::endpoint(tcp::v4(), port)) {
4    start_accept();
5
6    m_thread = std::thread([this](){ m_io_context.run(); });
7
8    log_trace("Talk action starting");
9  }
10
11 void Talk::start_accept() {
12   tcp_connection::pointer connection =
13       tcp_connection::create(
14         m_acceptor.get_executor().context()
15       );
16
17   m_acceptor.async_accept(connection->socket(),
18       [this, connection](const asio::error_code& error) {
19         handle_accept(connection, error);
20       }
21   );
22 }
```

ACCEPTING CONNECTIONS

```
1  void Talk::handle_accept(tcp_connection::pointer connection,
2                           asio::error_code const& error) {
3    if (!error) {
4      log_debug("Accepted message connection");
5
6      asio::async_read(
7        connection->socket(),
8        asio::dynamic_buffer(connection->data()),
9        [this, connection](
10         const asio::error_code& error,
11         std::size_t bytes_transferred)
12         {
13           handle_read(error, bytes_transferred, connection);
14         }
15     );
16   } else
17     log_error("Talk accept failed: {}", error.message());
18
19   start_accept();
20 }
```

STORING DATA

```
 1 void Talk::handle_read(asio::error_code const& error,
 2                         std::size_t bytes_transferred,
 3                         tcp_connection::pointer connection) {
 4   // Check error - 'eof' means remote connection closed
 5   if (!error || error == asio::error::eof) {
 6
 7     // Limit the message array
 8     if (m_messages.size() > max_messages)
 9       m_messages.erase(m_messages.begin());
10
11     // Store the message for Lua retrieval
12     m_messages.emplace_back(connection->data());
13
14     log_info("Received message ({} bytes): {}",
15              bytes_transferred,
16              connection->data());
17   } else
18     log_error("Talk read failed: {}", error.message());
19 }
```

LUA RETRIEVAL

```
 1 std::string Talk::GetNextMessage(void) {
 2   if (!IsMessageAvailable())
 3     return "";
 4
 5   std::string ret = m_messages.front();
 6
 7   m_messages.erase(m_messages.begin());
 8
 9   return ret;
10 }
```


BEHAVIOUR

COROUTINES

- Great for event-driven asynchronous systems
- Lua coroutines are stackful
- C++20 coroutines are stackless
- Single threaded so lock-free, no races etc.
- Implement your own dispatcher in Lua

LUACHAT BEHAVIOUR

- Sender coroutine: sends user input to peer
- Receiver coroutine: prints received messages
- Dispatcher coroutine: schedules sender and receiver
- main: processes arguments and creates coroutines

SENDER COROUTINE

```lua
function sender (talk, host, port)

  while true do

    local ret = require 'posix'.rpoll(0, 1000)
    if (ret == 1) then
      local message = io.read()
      if (message ~= "") then

        local ret = talk:Send(
          tostring(host), tostring(port), tostring(message)
        )

        if (ret == Actions.Talk.ErrorType_SUCCESS) then
          Actions.Log.info(
            string.format(
              "Message sent to %s:%s %s", host, port, message
            )
          )
        end
      end
```

RECEIVER COROUTINE

```lua
function receiver (talk, host, port)

  while true do

    -- Yield until a message arrives, at which point, print it
    repeat
      coroutine.yield()
    until talk:IsMessageAvailable()

    local message = talk:GetNextMessage()

    Actions.Log.info(
      string.format(
        "Received from %s:%s %s", host, port, message
      )
    )

    print(host .. ":" .. tostring(port) .. "> " .. message)

  end

end
```

DISPATCHER

```lua
function dispatcher (coroutines)

  local timer = Actions.Timer()

  while true do
    if next(coroutines) == nil then break end -- no coroutines

    for name, co in pairs(coroutines) do
      local status, result = coroutine.resume(co)

      if result then -- coroutine has exited

        if type(result) == "string" then -- runtime error
          Actions.Log.critical(
            "Coroutine '" .. tostring(name) .. "' error " .. result
          )
        else
          Actions.Log.warn(
            "Coroutine '" .. tostring(name) .. "' exited"
          )
        end
```

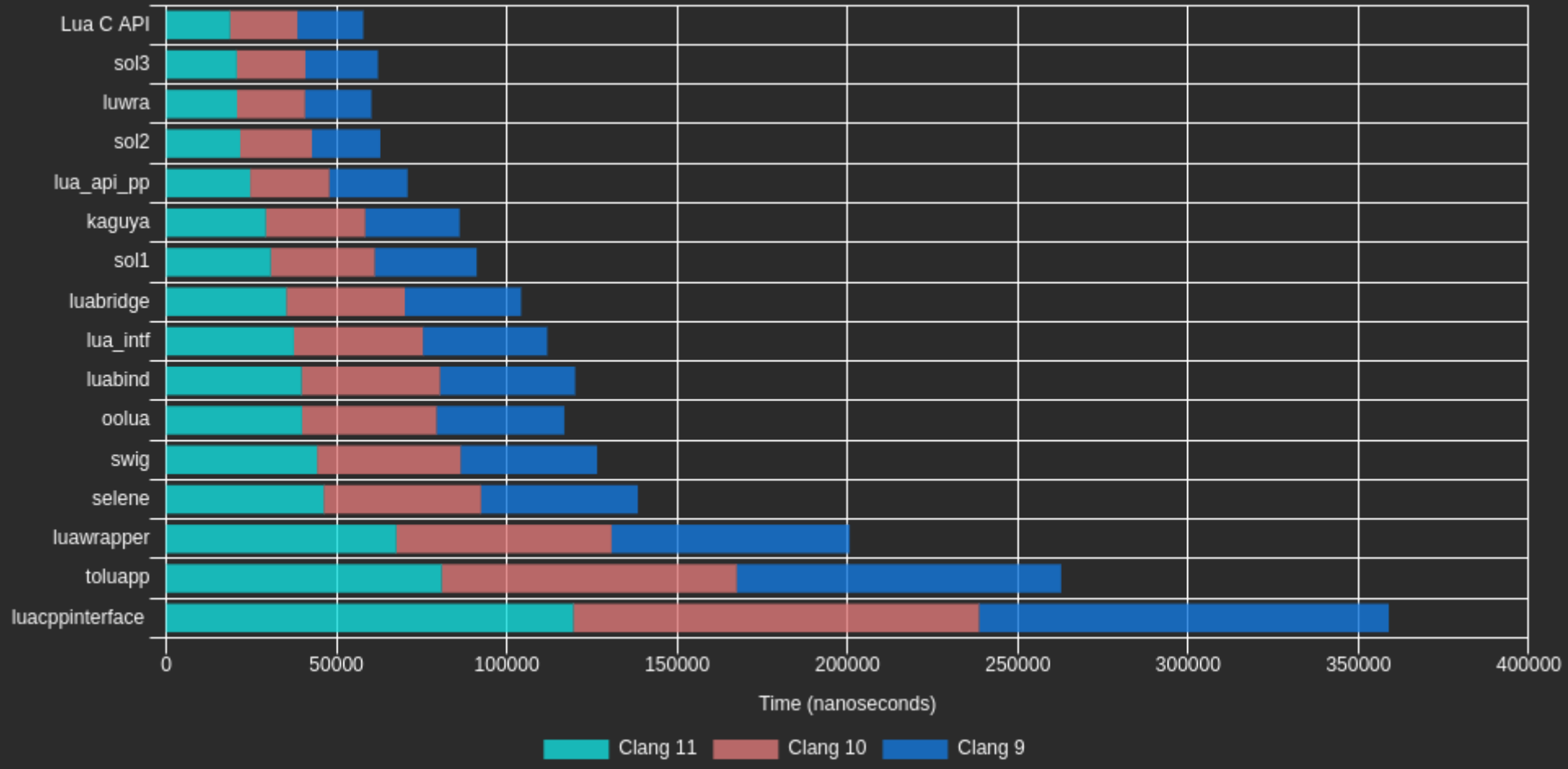

PERFORMANCE

MEASUREMENT

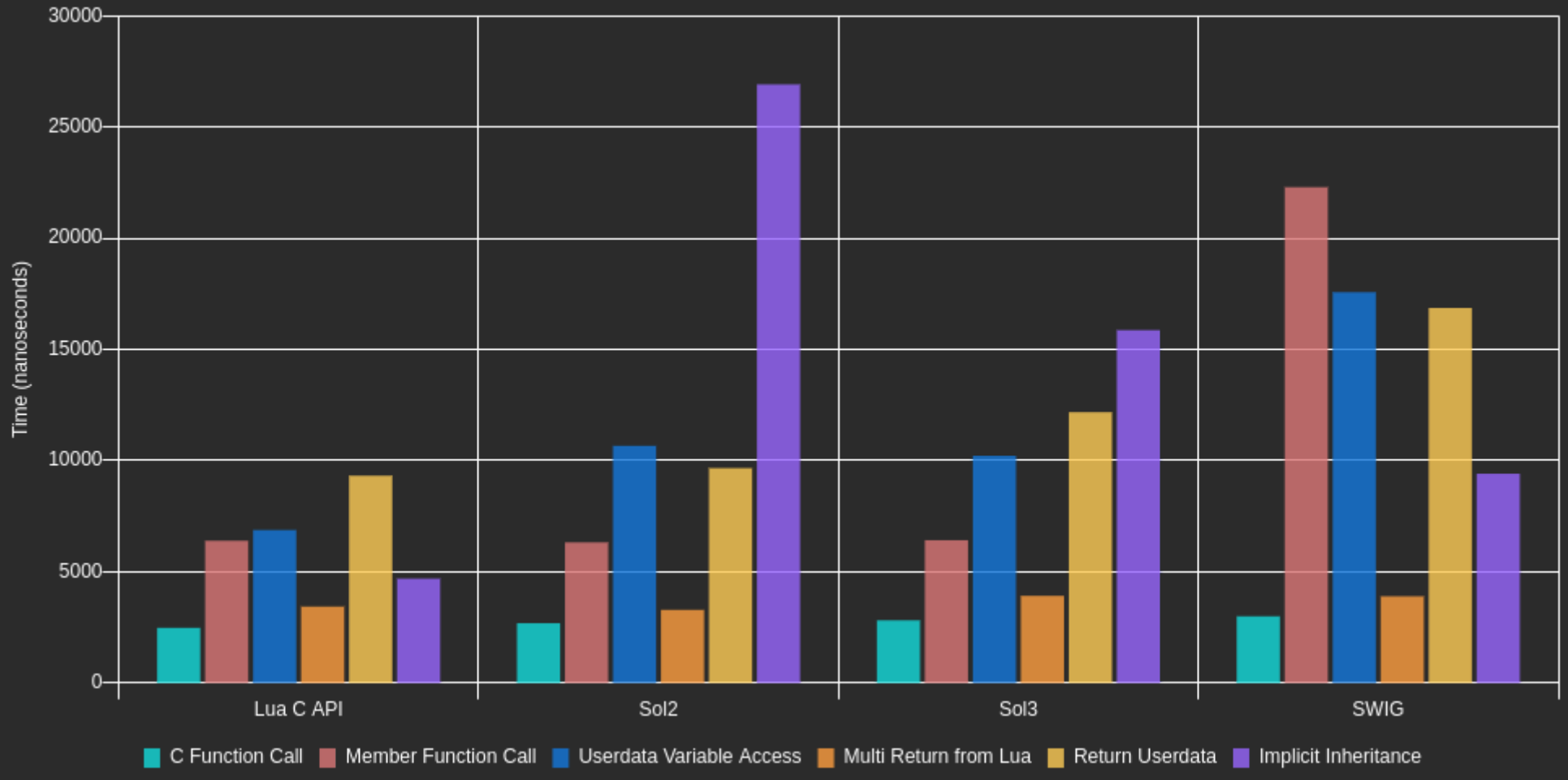
How do we compare performance?

- Benchmark Suites e.g. Lua Bindings Shootout
  - 16 Lua Bindings
  - Sol3 3.2.3 and SWIG 4.0.2 (Lua 5.4.2)
  - x86-64 i5-6200u and Embedded ARMv8
    - 64 bit dual core (4 threads) Skylake
    - 2.3 Ghz, L1 128 KiB, L2 512 KiB, L3 3 MiB
  - Clang 9/10/11

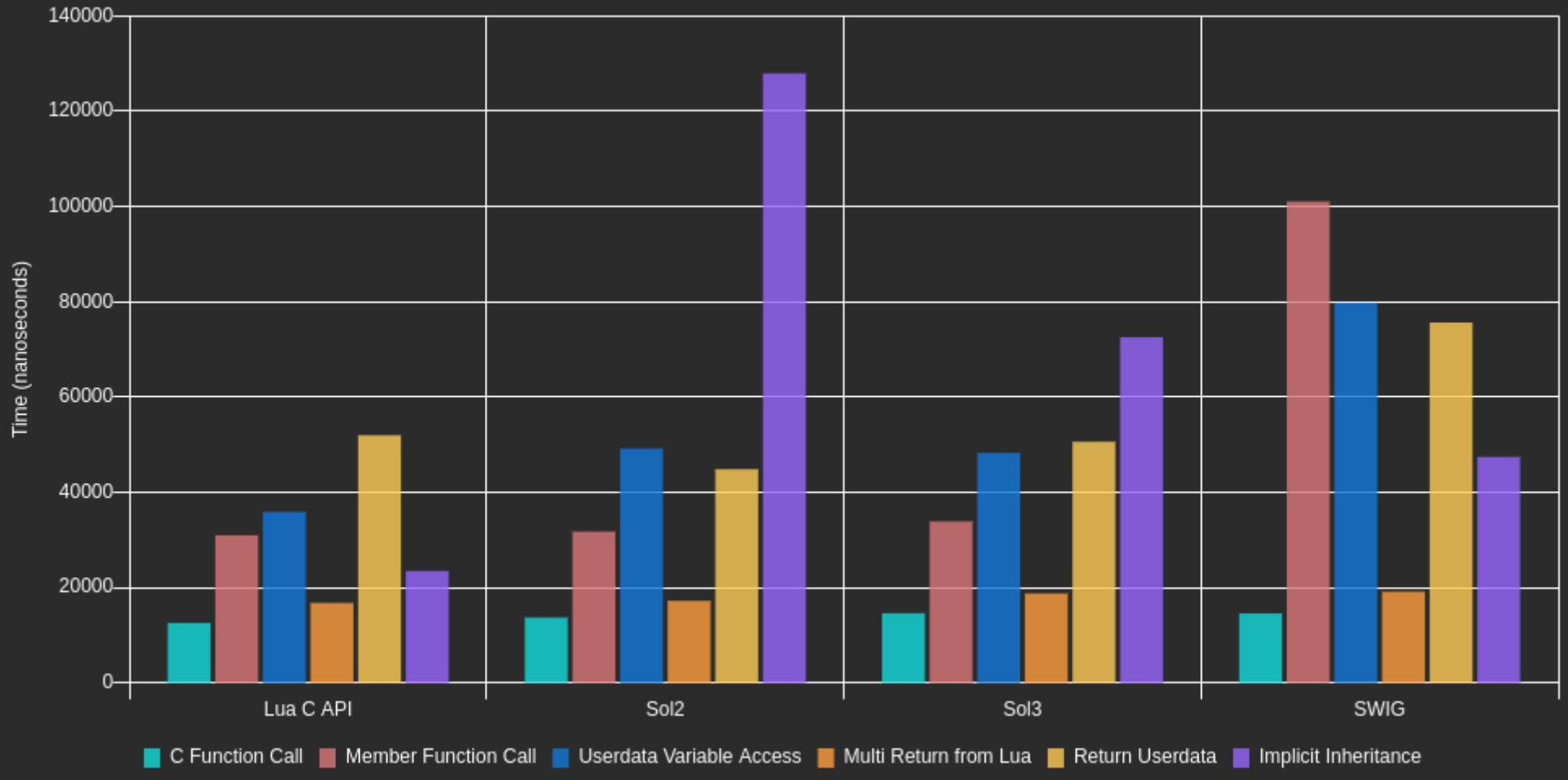
LUA BINDINGS BY COMPILER (X86-64 I5-6200U)



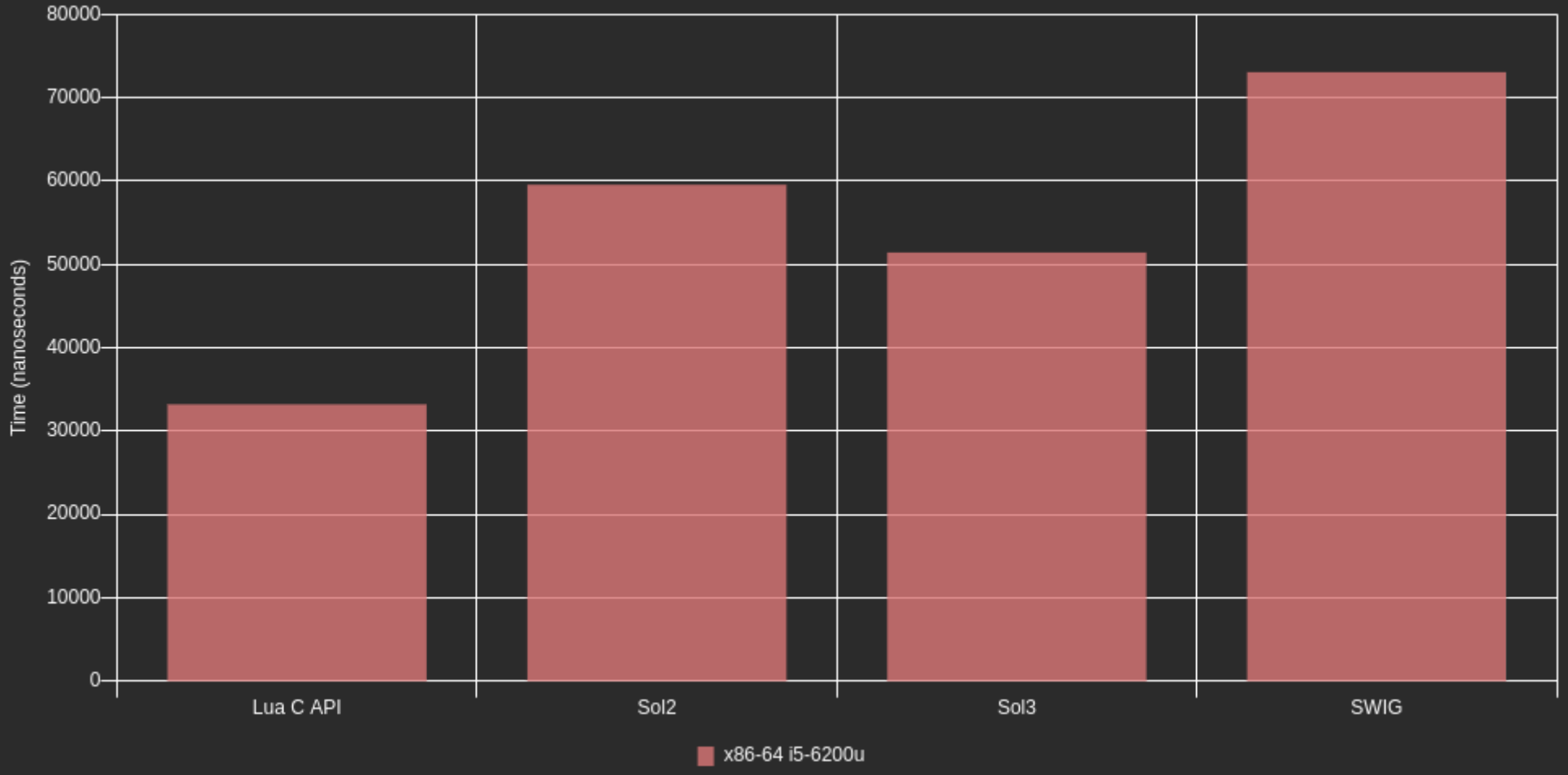
SELECT LUA BINDINGS: X86-64 I5-6200U



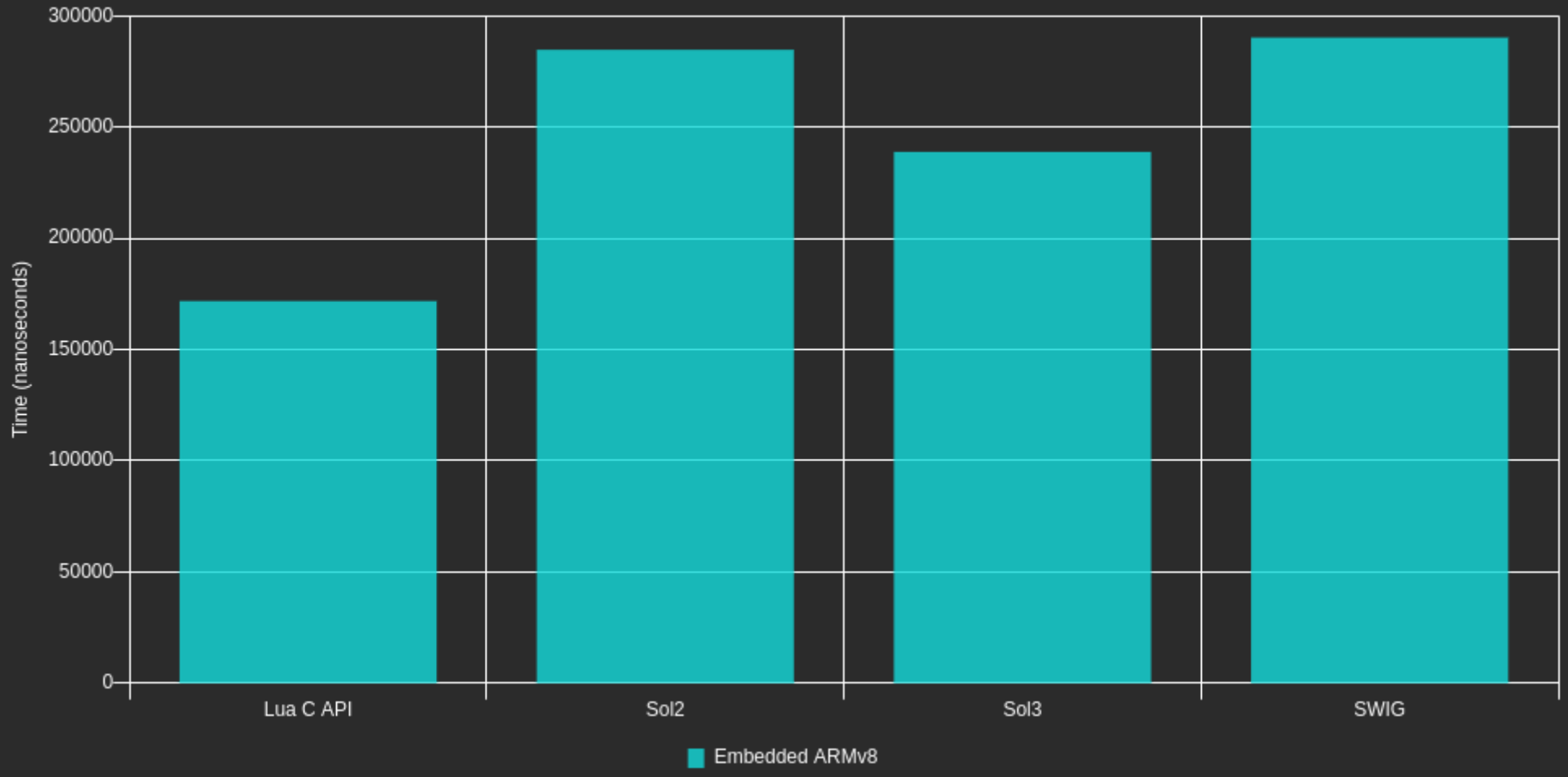
SELECT LUA BINDINGS: EMBEDDED ARMV8



AGGREGATED BINDINGS: X86-64 I5-6200U



AGGREGATED BINDINGS: EMBEDDED ARMV8




CONCLUSION

PERFORMANCE ADVICE

- Sol3 is fast but you can go faster
  - lots of good advice here
- MCM spends a lot of time in the SWIG wrapper
  - prefer lightweight typemaps
- The partition between C++ and Lua is important
  - as is the concurrency design
- How the code interacts with Lua is significant
  - prefer pre-compiled long-lived behaviours

CONCLUSION

- The combination of C++ and Lua is powerful
  - Actions (C++) and Behaviours (Lua)
- Sol3 binds Modern C++ to Lua
  - simple-to-use, fast, ideal for Modern C++
  - by definition is a C++ to Lua binding
- SWIG allows us to map C++ types to/from Lua
  - generates bindings in many languages
  - be mindful of performance
- Lua 5.4.2 is now available
  - Lua Quick Reference (updated for Lua 5.4)

QUESTIONS?

http://www.james-pascoe.com
james@james-pascoe.com

http://jamespascoe.github.io/accu2021
https://github.com/jamespascoe/LuaChat.git