redhat.

Lua's coroutines: the secret sauce in Nmap's Scripting Engine

Lua Workshop 2017

Patrick Donnelly
Software Engineer
2017 October 16th

`whoami`

Patrick Donnelly

- Started as a WoW Addon lurker on lua-l: batrick@batbytes.com
- Got excited finding a few bugs in Lua which led to...
- Got involved with the Nmap project through Google Summer of Code as a student: 2008 and 2009. Mentored students from 2010-2014.
- Outside of Lua: went to graduate school at University of Notre Dame for a PhD and now work on the Ceph file system at Red Hat.

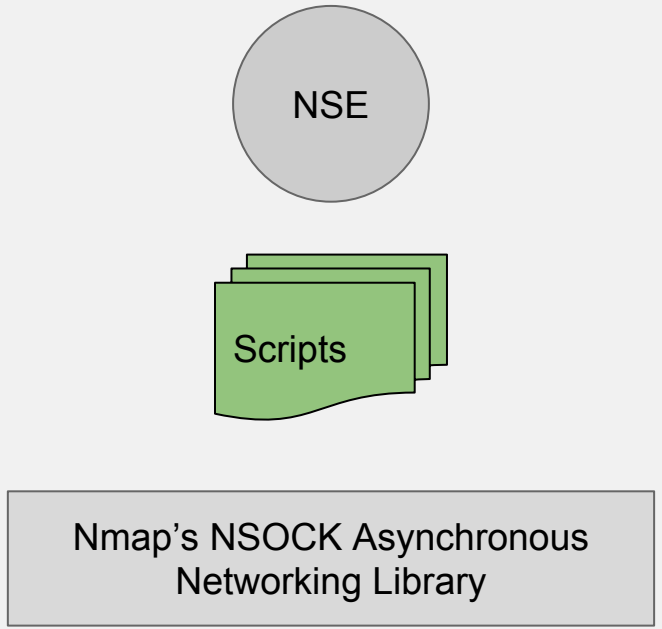
What's Nmap?

- Massively parallel network reconnaissance tool to find:
  - online hosts
  - open ports
  - OS running on hosts
  - network layout/security
  - everything there is to know about the server (NSE!)

```
$ nmap -p 80 --script 'safe and default' www.google.com

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-14 17:44 PDT
Nmap scan report for www.google.com (216.58.216.4)
Host is up (0.025s latency).
Other addresses for www.google.com (not scanned): 2607:f8b0:4007:804::2004
rDNS record for 216.58.216.4: lax02s21-in-f4.1e100.net

PORT   STATE SERVICE
80/tcp open  http
| http-robots.txt: 214 disallowed entries (15 shown)
| /search /sdch /groups /index.html? /? /?hl=*&
|_/?hl=*&*&gws_rd=ssl /imgres /u/ /preferences /setprefs /default /m? /m/ /wml?
|_http-title: Google

Nmap done: 1 IP address (1 host up) scanned in 1.33 seconds
```

Nmap Scripting Engine

- Parallel network script execution framework
- Scripts execute concurrently performing advanced and specific network reconnaissance against the host.
- NSE Includes hundreds of scripts and libraries
- Started as a GSOC project by Diman Todorov in 2008



Example Script

```
local http, stdnse  = require "http", require "stdnse"

description, author = "get the web page title", "Patrick"

categories = {"default", "safe"}

function portrule(host, port)  return port.number == 80 end

function action(host, port)

      local resp = http.get( host, port, stdnse.get_script_args(SCRIPT_NAME..".url") or "/" )

      local title = string.match(resp.body, "<[Tt][Ii][Tt][Ll][Ee][^>]*>([^<]*)</[Tt][Ii][Tt][Ll][Ee]>")

      return title

end
```

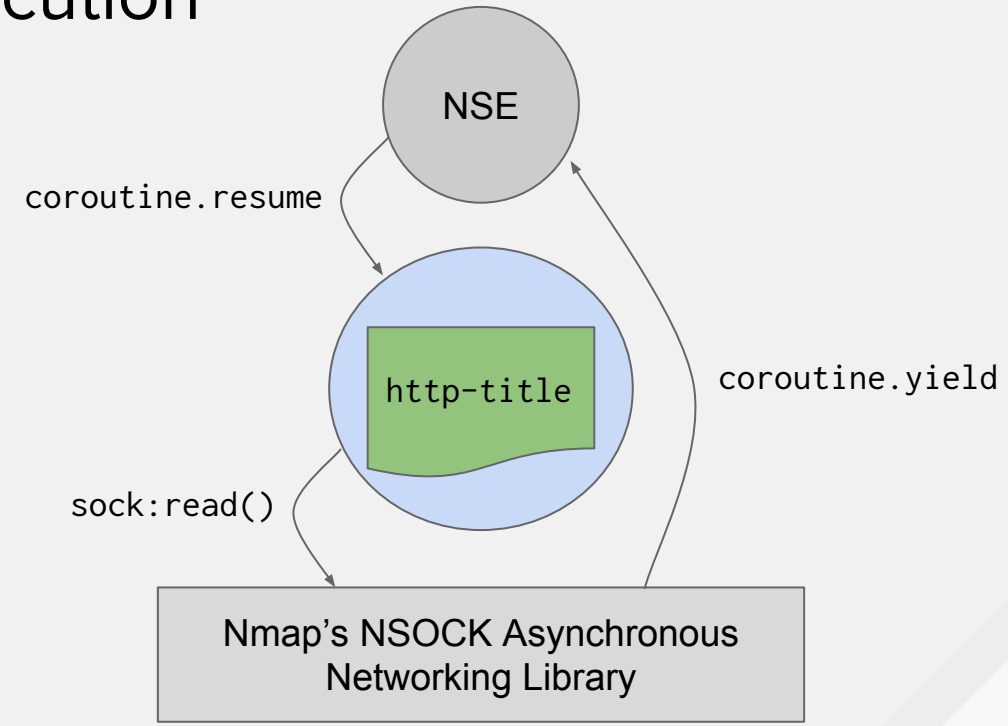
Should the script run?

Script's main function

```
$ nmap -p 80 --script $(pwd)/http-title.nse www.google.com

Starting Nmap 7.60 ( https://nmap.org ) at 2017-10-08 21:18 PDT

[...]

Nmap scan report for www.google.com (216.58.217.196)

Host is up (0.022s latency).

[...]

PORT   STATE SERVICE

80/tcp open  http

|_http-title: Google


Nmap done: 1 IP address (1 host up) scanned in 0.46 seconds
```

Concurrent Script Execution

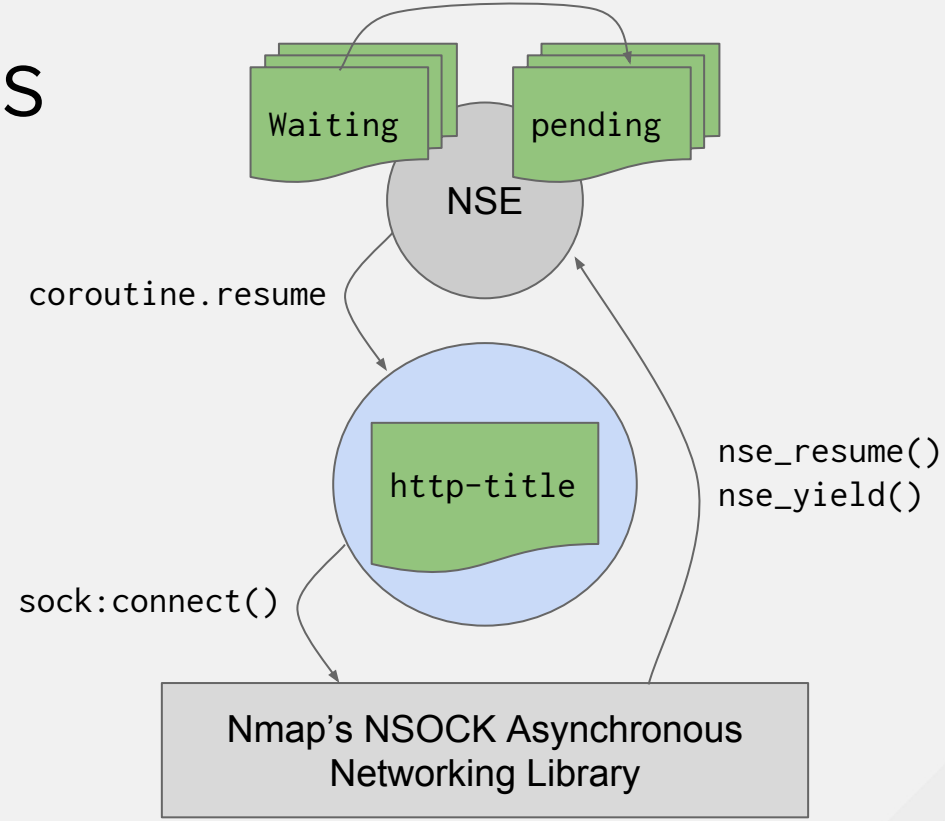
- Each script is instantiated in a coroutine which tests (rule function) the host and then gathers information (action)
- Script threads are run concurrently, network I/O causes scripts to yield



Challenge: Build an OS

But can it run emacs?

- **NSE maintains tables of pending/waiting scripts.**
- **Also, has a generator which produces the next scripts to run (to limit the number of active scripts)**
- **Defines mechanisms for libraries to yield and restart scripts (used by nsock to resume a script once data is available)**
- **Most of it is in nse_main.lua (1500LOC)**
- **NSOCK limits script parallelism (limits # of active sockets / see also `--max-parallelism`)**



Challenge: Mutual Exclusion

Mutual exclusion for coroutines? ... Why

- Scripts sometimes need to limit concurrency doing network operations
- First instance of the problem: whois script which gets WHOIS data from IANA servers. Doing WHOIS lookups for hundreds of target hosts results in getting banned. Who knew?

```
$ nmap -p 80 --script whois-domain.nse www.google.com

[...]

80/tcp open  http

Host script results:

| whois-domain:

| Domain name record found at whois.verisign-grs.com

|      Domain Name: GOOGLE.COM\x0D

|      Registry Domain ID: 2138514_DOMAIN_COM-VRSN\x0D

|      Registrar WHOIS Server: whois.markmonitor.com\x0D

|      Registrar URL: http://www.markmonitor.com\x0D

|      Updated Date: 2011-07-20T16:55:31Z\x0D

|      Creation Date: 1997-09-15T04:00:00Z\x0D

[...]
```

NSE Mutexes

```
mutex = stdnse.mutex(obj)

-- e.g. stdnse.mutex “my-script.x-critical-section”

mutex “lock”

mutex “done” -- yes, I feel bad about that argument name

mutex “trylock” -- joke, no one ever uses this
```

Where are Mutexes used?

- whois-*nse to serialize lookups
- http cached GETs, to avoid concurrent GETs of the same page
- Preventing concurrent SSL cert lookup + caching
- HTTP “slowloris” attack; only one attack at a time

Challenge: Multitasking Scripts

Oh no, we're actually building a kernel...

- By default, a script is limited to doing one nsock operation at a time which prevents parallel network operations.
- Use case: http-spider library that does parallel GET requests against a target www server.

```
local httpspider = require “httpspider”
action = function(host, port)
  local maxpages = stdnse.get_script_args(SCRIPT_NAME .. ".maxpagecount") or 1
  local tries = stdnse.get_script_args(SCRIPT_NAME .. ".tries") or 5

  local dump = {}
  local crawler = httpspider.Crawler:new( host, port, nil, { scriptname = SCRIPT_NAME, maxpagecount =
tonumber(maxpages) } )
  crawler:set_timeout(10000)
  -- launch the crawler
  while(true) do
       local start = stdnse.clock_ms()
       local status, r = crawler:crawl()
       if ( not(status) ) then break end
       local chrono = stdnse.clock_ms() - start
       dump[chrono] = tostring(r.url)
  end
  -- More processing...
```

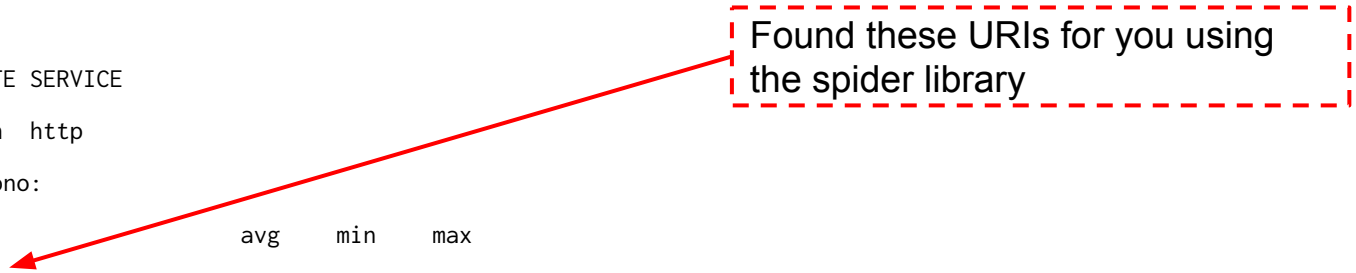
Create the spider

Fetch a URI / crawl the website

Script: http-chrono

```
PORT   STATE SERVICE
80/tcp open  http
|_http-chrono: Request times for /; avg: 2.98ms; min: 2.63ms; max: 3.62ms


PORT   STATE SERVICE
80/tcp open  http
| http-chrono:
| page                      avg     min     max
| /admin/                   1.91ms   1.65ms   2.05ms
| /manager/status           2.14ms   2.03ms   2.24ms
| /manager/html             2.26ms   2.09ms   2.53ms
|_/examples/servlets/       2.43ms   1.97ms   3.62ms
```

Found these URIs for you using the spider library



Example parallel HTTP GET function:

```
local http = require “http”
function pget(host, port, urls)
  local threads, responses = {}, {}
  local function do_get(i)
    responses[i] = http.get(host, port, urls[i])
  end
  for i = 1, #urls do
    threads[#threads+1] = stdnse.new_thread(do_get, i)
  end
  return responses
end
```

Launch concurrent thread

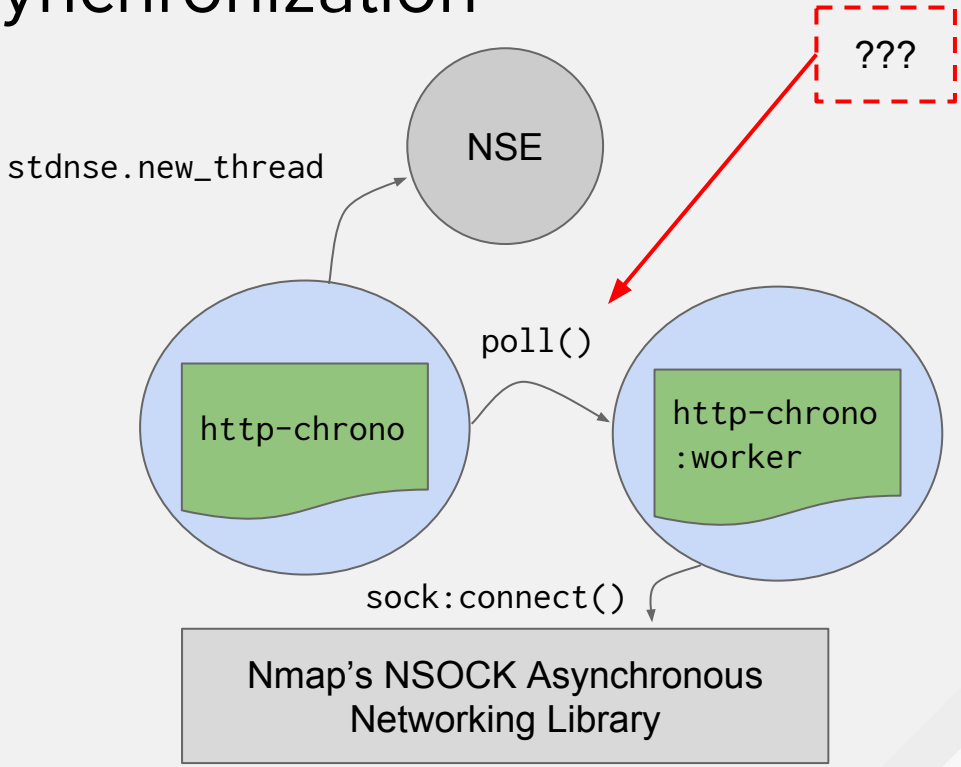
No memory sync needed

Does it work? NO

Challenge: Thread Synchronization

Is this presentation over?

- How do we get scripts to coordinate with each other?
- Well, let's borrow from another synchronization primitive... condition variables.



NSE Condition Variables

```
condvar = stdnse.condvar(obj)

-- e.g. local thread_pool = {}; stdnse.condvar(thread_pool)

condvar “wait” -- wait to be woken up

condvar “signal” -- wake up a sleeper

condvar “broadcast” -- wake up everyone
```

Fixed parallel HTTP GET function:

```
local http = require “http”
function pget(host, port, urls)
  local threads, responses = {}, {}
  local condvar = stdnse.condvar(threads)
  local function do_get(i)
    responses[i] = http.get(host, port, urls[i])
    condvar “signal”
  end
  for i = 1, #urls do threads[#threads+1] = stdnse.new_thread(do_get, i) end
  repeat
     condvar “wait”
     for i, thread in ipairs(threads) do
       if coroutine.status(thread) == “dead” or responses[i] then threads[i] = nil end
     end
  until next(threads) == nil
  return responses
end
```

Wake up the main thread

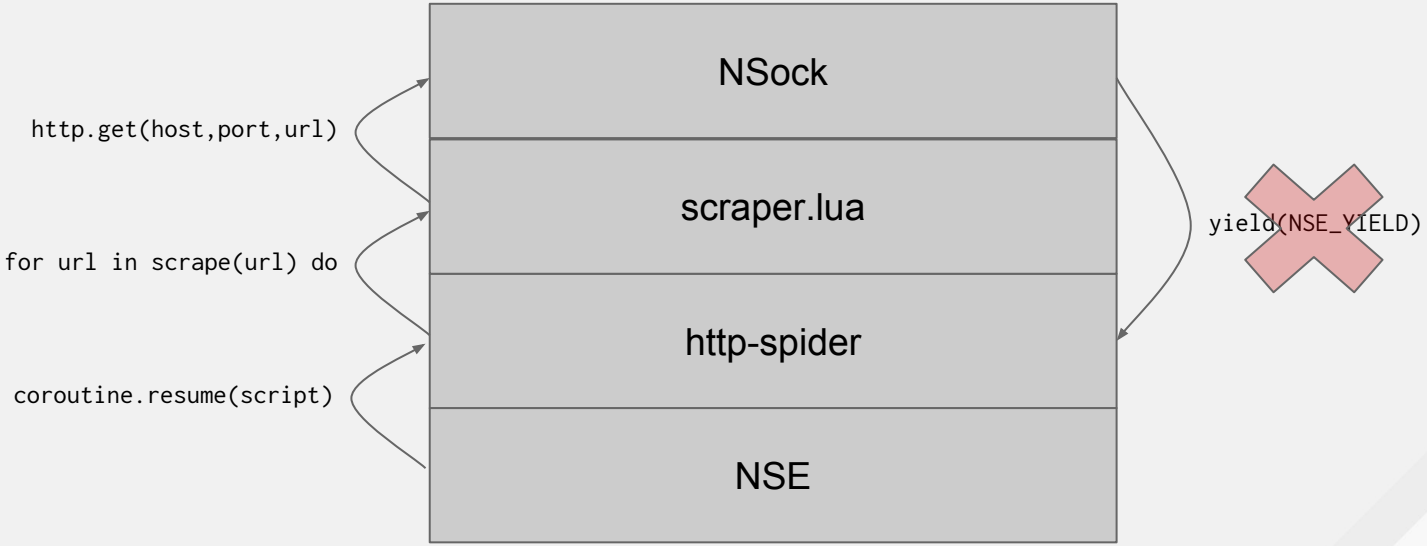
Loop until threads are done

Does it work? YES, but:

- Should use worker thread pool
- Needs error checking

Challenge: Coroutine Stacks (or “nested”)

- Yields across multiple threads requires changes to Lua’s coroutine library.



NSE coroutine yield tags

Defined by NSE. New coroutine.resume called by scripts/libraries

```
int nse_yield (lua_State *L, lua_KContext ctx, lua_KFunction k)
{
  lua_getfield(L, LUA_REGISTRYINDEX, NSE_YIELD);
  lua_pushthread(L);
  lua_call(L, 1, 1); /* returns NSE_YIELD_VALUE */
  return lua_yieldk(L, 1, ctx, k); /* yield with NSE_YIELD_VALUE */
}
```

Called by nsock library

```
Local NSE_YIELD_VALUE = {}
local function handle (co, status, ...)
      if status and NSE_YIELD_VALUE == ... then -- NSE has yielded
the thread
      return handle(co, resume(co, yield(NSE_YIELD_VALUE)));
      else
      return status, ...;
      end
 end
 function coroutine.resume (co, ...)
      return handle(co, resume(co, ...));
```

See also recent discussion on lua-l:
http://lua-users.org/lists/lua-l/2015-09/msg00316.html

NSE Base Thread

nse_main.cc:

```
void nse_base (lua_State *L)
{
  lua_getfield(L, LUA_REGISTRYINDEX, NSE_BASE);
  lua_call(L, 0, 1); /* returns base thread */
}
```

nse_main.lua:

```
  _R[BASE] = function ()
       return current and current.co;
  end
```

nse_nsock.cc:

```
static int socket_lock (lua_State *L, int idx)
{
  unsigned p = o.max_parallelism == 0 ? MAX_PARALLELISM :
o.max_parallelism;
  int top = lua_gettop(L);
  nse_base(L);
  lua_rawget(L, THREAD_SOCKETS);
```

Challenge: Not re-inventing network libraries

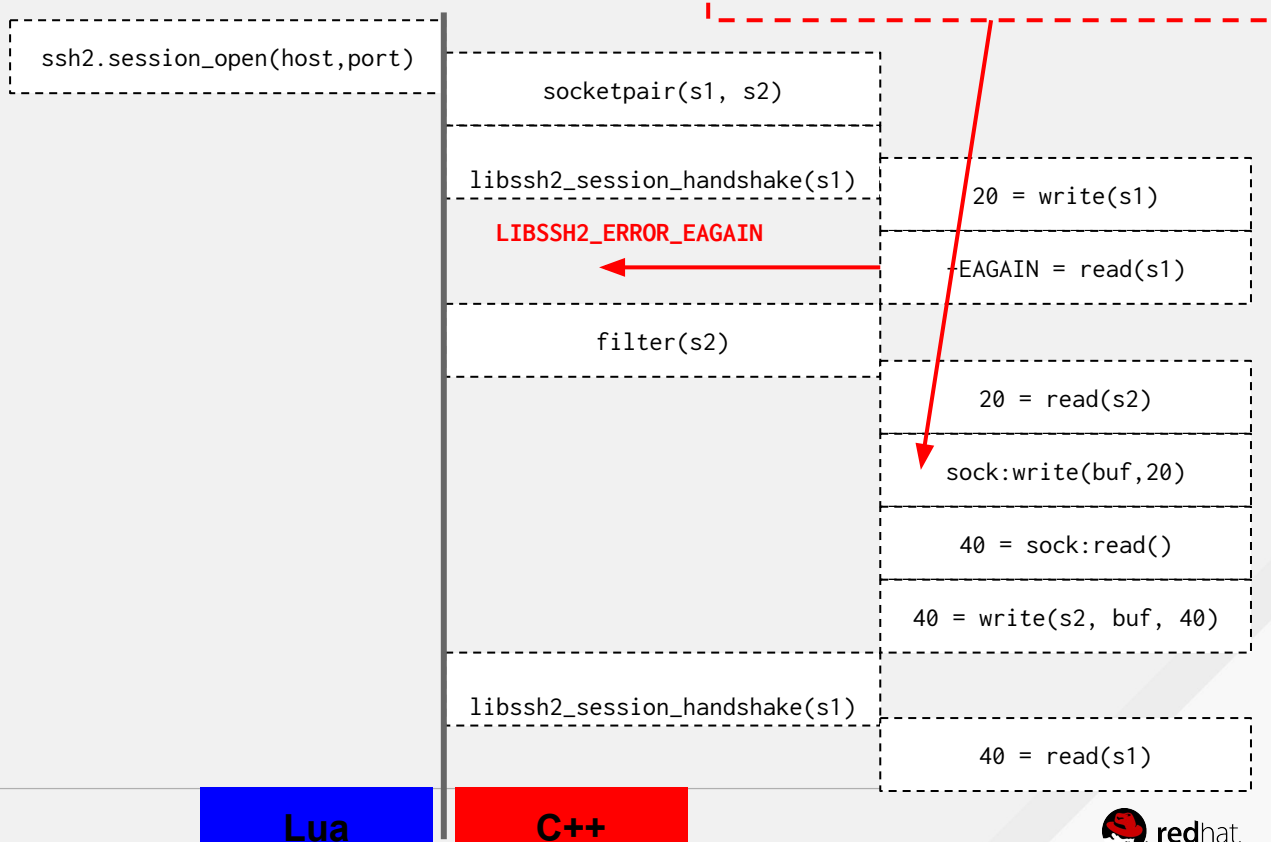
But re-implementing in Lua code is so fun!

- **Idea: we'd like to link to libssh2 to run scripts against ssh servers**
- **Problem: how do we get libssh2 to play nice with other scripts by using our networking framework?**
- **Naive solution/surrender: just accept ssh sessions block the process**

```
#include <libssh2.h>

int libssh2_session_handshake(
        LIBSSH2_SESSION *session,
        libssh2_socket_t socket)
```

Solution: Give SSH a UNIX socket

- Each ssh session allocates a socketpair
- SSH gets one end of the socket to talk to non-blocking
- Benefit: we now get network I/O parallelism when using libssh2.



redhat.

Conclusion: coroutines are awesome!

Patrick Donnelly

pdonnell@redhat.com

https://nmap.org/