Lua
For Science!

...you monster

Leo Antunes

HPAC

January 2014

RWTH AACHEN UNIVERSITY

What is Lua?

- Interpreted language
- "Teenage": developed in the early 90s in Brasil (yay!)
- Reference implementation interpreted; also JITed, LLVM
- How many Python users out there?

The Language

- Dynamically typed
- Imperative, but multi-paradigm(ish):
  - Functional: currying, higher-order functions (however: side-effects!)
  - Object-oriented: some syntactic sugar, but mostly bare-bones (everything can *be made* into an object)
- Implemented in pure ANSI C (extremely portable)
- Types:
  - Usual: numbers, strings, bools, functions and nil

The Language

- Dynamically typed
- Imperative, but multi-paradigm(ish):
  - Functional: currying, higher-order functions
    (however: side-effects!)
  - Object-oriented: some syntactic sugar, but mostly bare-bones
    (everything can *be made* into an object)
- Implemented in pure ANSI C (extremely portable)
- Types:
  - Usual: numbers, strings, bools, functions and nil
  - Higher-level: tables, tables and - did I mention - *tables?*

The Language

- Dynamically typed
- Imperative, but multi-paradigm(ish):
  - Functional: currying, higher-order functions (however: side-effects!)
  - Object-oriented: some syntactic sugar, but mostly bare-bones (everything can *be made* into an object)
- Implemented in pure ANSI C (extremely portable)
- Types:
  - Usual: numbers, strings, bools, functions and nil
  - Higher-level: tables, tables and - did I mention - *tables?*
  - Object orientation through tables: metatables (cf. Python)
- Nothing *very* new, so we'll focus on API

A Bit of Syntax (not our focus, but...)

```
-- some examples
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end
some_table = {
    name="whatever",
    func=factorial
    }
some_table.func(3) --> 6
```

- No () for single argument (allows currying like Haskel's)

A Bit of Syntax (not our focus, but...)

```
-- some examples
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end
some_table = {
    name="whatever",
    func=factorial
    }
some_table.func(3) --> 6
```

- No () for single argument (allows currying like Haskel's)
- Can `return` multiple results (no packing/unpacking)

A Bit of Syntax (not our focus, but...)

```
-- some examples
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end
some_table = {
    name="whatever",
    func=factorial
    }
some_table.func(3) --> 6
```

- No () for single argument (allows currying like Haskel's)
- Can `return` multiple results (no packing/unpacking)
- Table indices start at 1 (convention; no actual arrays)

A Bit of Syntax (not our focus, but...)

```
-- some examples
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end
some_table = {
    name="whatever",
    func=factorial
    }
some_table.func(3) --> 6
```

- No () for single argument (allows currying like Haskel's)
- Can `return` multiple results (no packing/unpacking)
- Table indices start at 1 (convention; no actual arrays)
- Scoping: globals per default (cf. Python; **real** closures)

A Bit of Syntax (not our focus, but...)

```
-- some examples
function factorial(n)
  local x = 1
  for i = 2,n do
    x = x * i
  end
  return x
end
some_table = {
    name="whatever",
    func=factorial
    }
some_table.func(3) --> 6
```

- No () for single argument
  (allows currying like Haskel's)
- Can `return` multiple results
  (no packing/unpacking)
- Table indices start at 1
  (convention; no actual arrays)
- Scoping: globals per default
  (cf. Python; **real** closures)
- No named arguments
  (but can pass table)

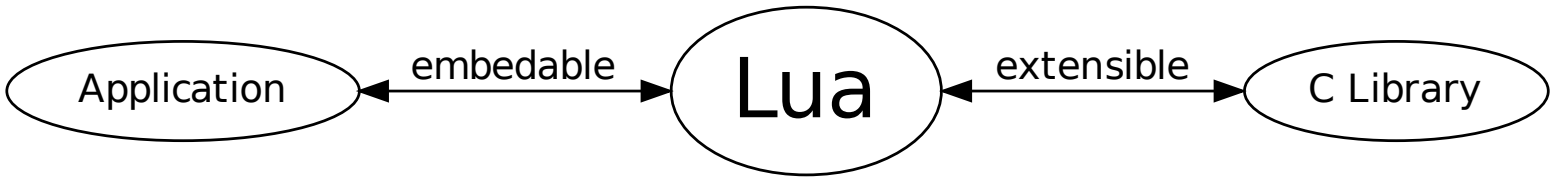
RWTH AACHEN UNIVERSITY

Use-cases

That's all nice and good, but **what is it for?**

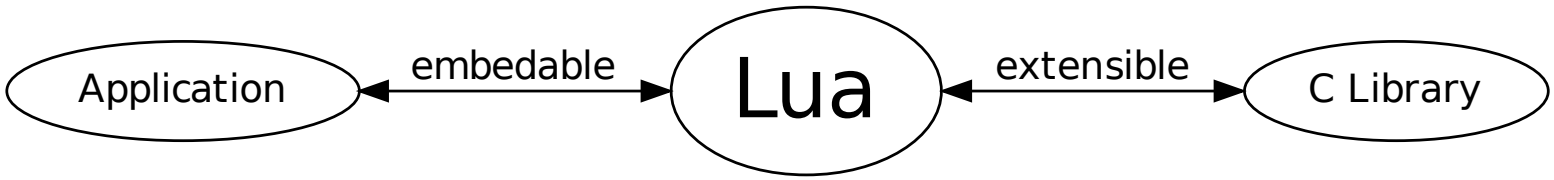
Use-cases

That's all nice and good, but **what is it for?**



Use-cases

That's all nice and good, but **what is it for?**



- Same stack-API on both cases (interpreter is thin-wrapper for lib)
- Focus: simplicity and performance over features (remember tables?)
- What seemed like limitations are now elegance
- Can wrap lib in C module or use Alien/FFI (cf. ctypes)

General Pros/Cons

...or: when should I use Lua instead of *X*?

- Not-so-rapid prototyping:
  - ✗ Boilerplate for OO
  - ✗ Standard library is paltry
  - ✗ LuaRocks still a bit lacking
  - ✓ **But:** performance!
- Embedding:
  - ✓ Very easy with C (cf. Python), theoretically easy with all
  - ✓ Handle configuration, customization, abstraction (DSLs)
- As Glue Language:
  - ✓ Also very easy to extend (remember: same API)
  - ✓ "Selective" Rapid Prototyping

What about science?

- Lua Libraries:
  - ✓ Numerical: NumLua (FFTW, BLAS, LAPACK)
  - ✓ Plotting: PLPlot, GNUPlot
  - ✗ MPI: *very* basic wrapper
  - ✗ Many missing, e.g.: Linear Programing

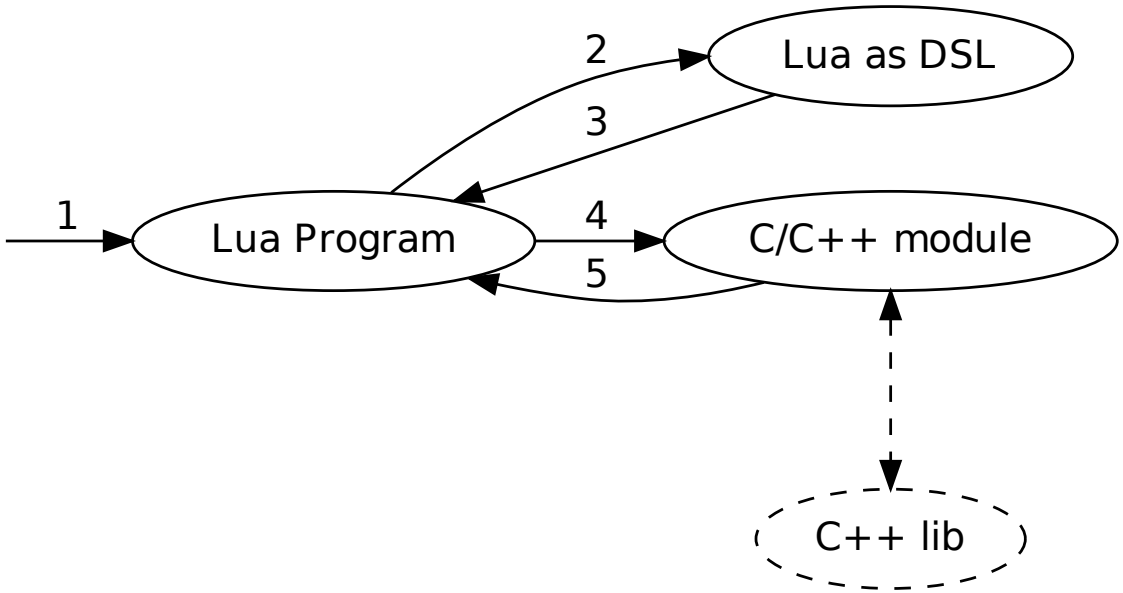
What about science?

- Lua Libraries:
  - ✓ Numerical: NumLua (FFTW, BLAS, LAPACK)
  - ✓ Plotting: PLPlot, GNUPlot
  - ✗ MPI: *very* basic wrapper
  - ✗ Many missing, e.g.: Linear Programing
- Parallelization
  - ✗ No threads from Lua's side (ANSI C; no actual GIL)
  - ✓ But of course: C code can multithread
  - ✓ Elegant and easy coroutines
  - ✓ Easy to implement message-passing multi-threading in C
- Misc:
  - ✗ Only one number type: double; So: no boolean ops
  - ✓ debug.debug(): interactive breakpoint for Lua code

- Up til now: tough sell

- Up til now: tough sell
- Where it shines: performance (later) and simplicity for embedding/extending

Subject allocator



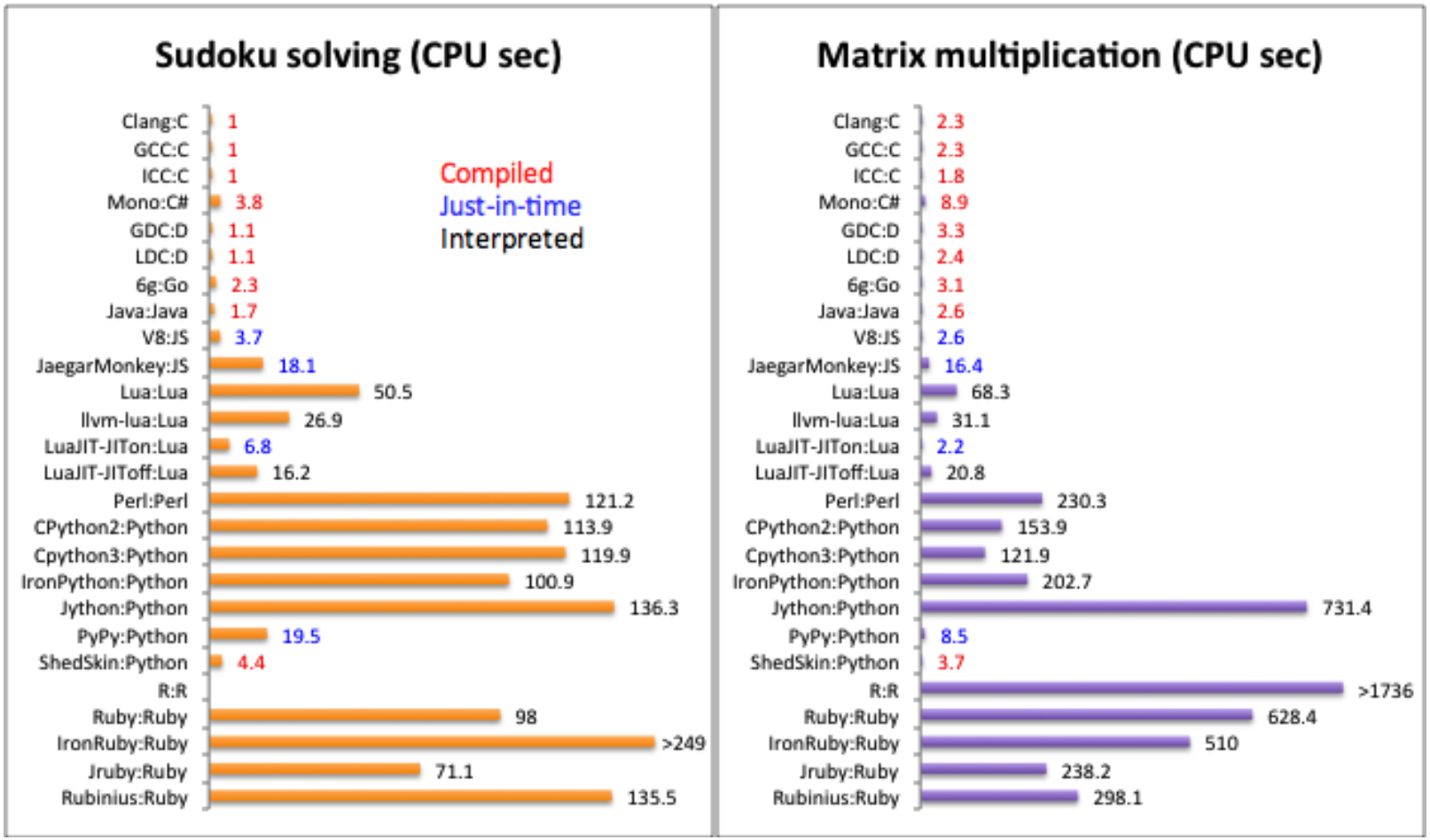


Figure : from http://attractivechaos.github.io/plb (independent)

RWTH AACHEN UNIVERSITY

Conclusion

- Lua is cool!
- Will I use it again? Probably.
- Should you use it? Definitely! For science? Maybe...

Conclusion

- Lua is cool!
- Will I use it again? Probably.
- Should you use it? Definitely! For science? Maybe...
- Is it a silver bullet? No. But I don't think that exists anyway

Thanks!
Questions?