Why (and why not) Lua?

Roberto Ierusalimschy

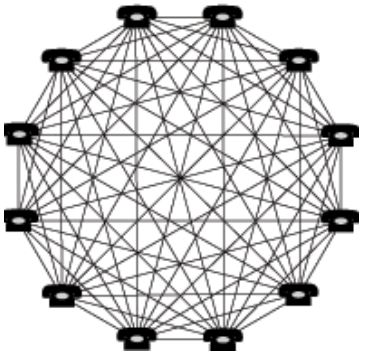
PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Why do we need multiple languages?

Wouldn't the world be better if everybody used the same programming language?

After all, programming languages have strong network effects!

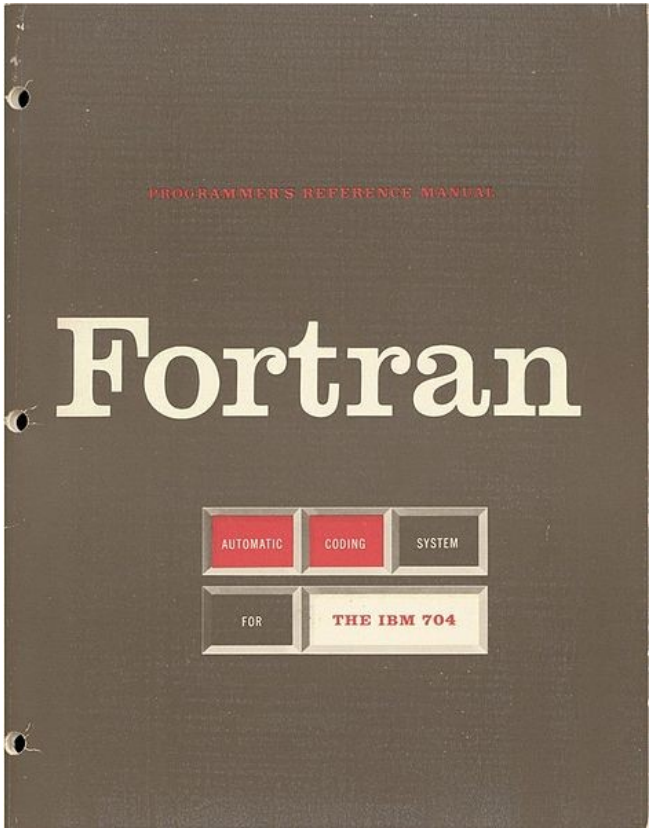
Network Effect



- I should learn the most popular language.
  - to find a job
- I should use the most popular language.
  - to find programmers
- I should use the most popular language.
  - to find libraries, documentation, etc.
- I should use the language I am already using.
  - to not have to learn a new language

Fact: if we all settled for arguments like “it would be good to use the same language as X” or “language X is better known than Y”, we would all be using Fortran today.

Why are we not?



Very Simplified Story

- FORTRAN (FORmula TRANslator) was specialized for scientific computation.
- COBOL (COmmon Business-Oriented Language) was created, specialized for commercial-oriented sofware.
- There were also Algol and Lisp...

But why do we need multiple languages?

PL/I

- A truly general-purpose programming language
  - *“FORTRAN VI is not intended to be compatible with any known FORTRAN IV. It includes the functional capabilities of FORTRAN IV as well as those capabilities normally associated with 'commercial' and 'algorithmic' languages.”*
- Championed by IBM
  - this was 1964
- Why was PL/I not a big success?
  - compare its usage with Fortran and Cobol

A programming language is not a pile of features.

The “Subset Fallacy”

- I can use only the features I like/need.
- Features that I do not use do not affect me.
- The bigger the pile, the better the language.

The “Subset Fallacy”

- Bugs frequently involve parts that you think you are not using
  - so you should know them for debugging
- Other people's code involves parts that you do not use
  - so you should know them for maintenance
  - so you should know them to use libraries

The “Subset Fallacy”

- Compilers and interpreters must support the entire language
  - extra features make them larger, more complex, and slower
  - many features hamper optimizations

The “Subset Fallacy”

- The main benefits offered by a programming language are not only what it allows us to do, but also *what it prevents us from doing*!
  - stack overflows!
  - memory safety

The design of a language involves many trade-offs, and we need explicit goals and priorities to settle these trade-offs. Different languages choose different goals, and therefore settle these trade-offs in different directions. Like any tool, no language is good for everything.

How Does Lua Solve Trade-offs?

- A set of explicit goals that we prioritize:
  - scripting
  - portability
  - small size
  - simplicity

How Does Lua Solve Trade-offs?

- Also, pragmatism!
- Other secondary goals:
  - performance
  - friendliness to non programmers (simplicity)

How does that work in practice?

attained goals

Scripting

- Scripting language x dynamic language
- Program written in two languages
  - a *scripting* language and a *system* language
- System language implements the hard parts
  - algorithms, data structures
  - reasonably stable parts
- Scripting language connects those parts
  - flexible, easy to change

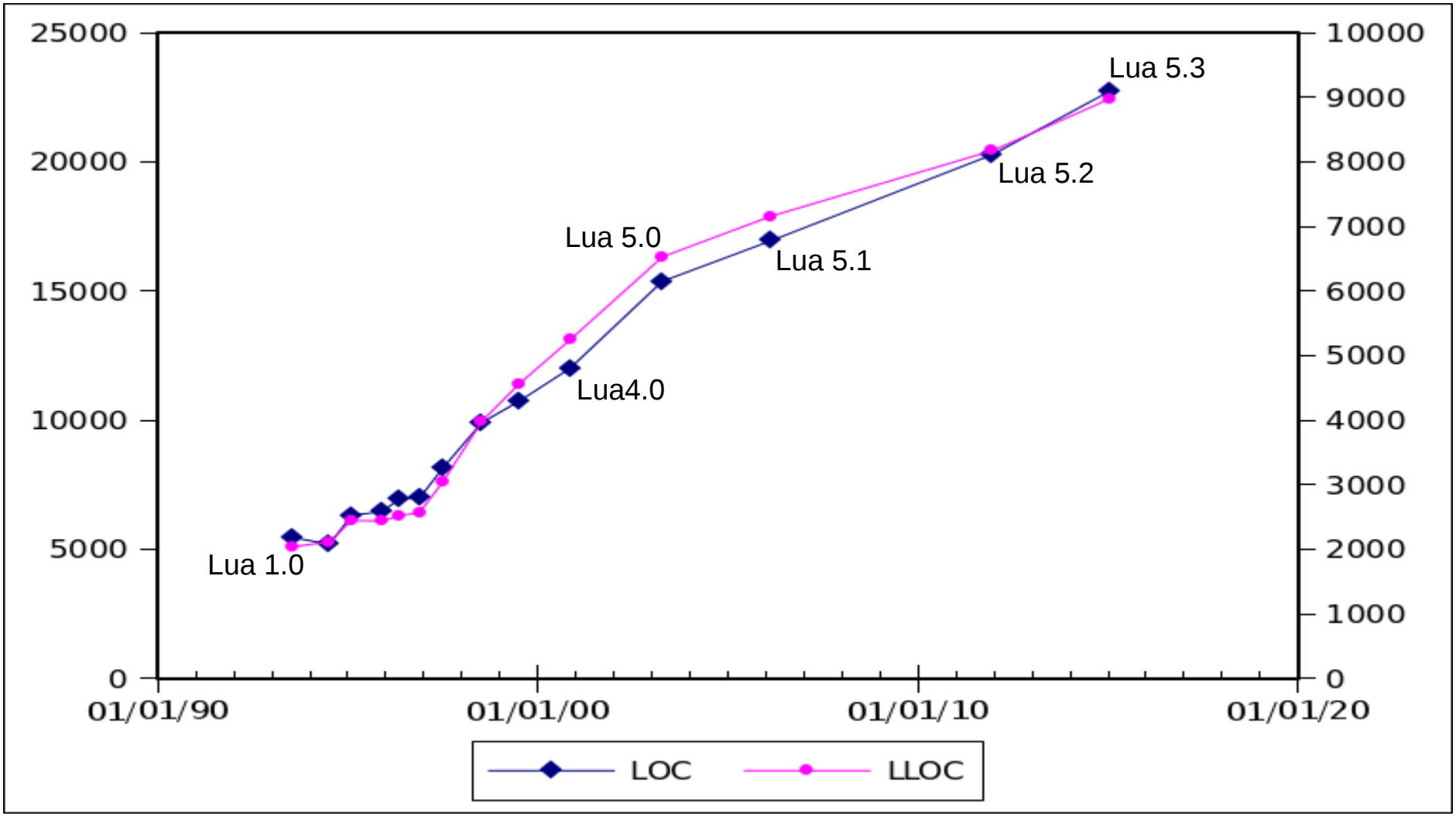
The Two Sides of Scripting

- *Extending: a*n application written in an *extensible language* calls libraries/functions in the system language
- *Embedding*: an application written in the system language calls libraries/functions in an *extension language*
- Lua has been designed to excel in both scenarios!

Portability

- Runs in virtually any platform
  - Posix (Linux, BSD, etc.), OS X, Windows, Android, iOS, Arduino, Raspberry Pi, Symbian, Nintendo DS, PSP, PS3, IBM z/OS, etc.
  - written in ANSI C
- Runs inside OS kernels
  - NetBSD
- Runs directly on the bare metal, without an OS
  - NodeMCU ESP8266

Size


25000
20000
15000
10000
5000
0
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
01/01/90
01/01/00
01/01/10
01/01/20
Lua 1.0
Lua4.0
Lua 5.0
Lua 5.1
Lua 5.2
Lua 5.3
LOC
LLOC

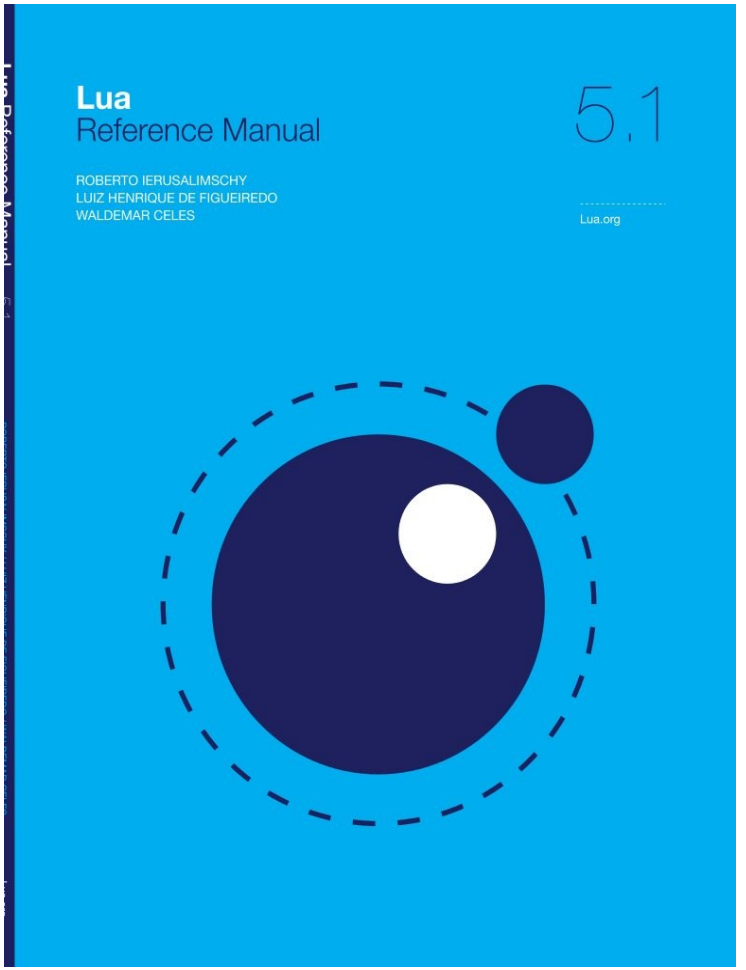
Simplicity

Reference manual with ~100 pages

Documents the language, the C API, and the standard libraries.

(SPINE)

Lua Reference Manual 5.1 ROBERTO IERUSALIMSCHY / LUIZ HENRIQUE DE FIGUEIREDO / WALDEMAR CELES Lua.org



Simplicity

- Few but powerful mechanisms
- Associative arrays (aka *Tables*)
  - implements all data structures (maps, arrays, structures, objects, modules, etc.)
- Closures
- Coroutines

How does that work in practice?

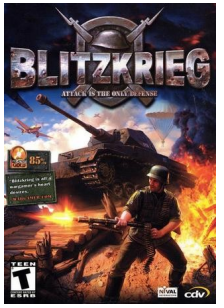
Some case studies


Games for Windows
COMPANY of HEROES
relic
THQ

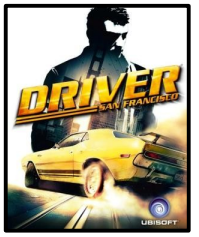

Games for Windows
UNIVERSE AT WAR
SEGA


Games for Windows
DUNGEONS
kalypso

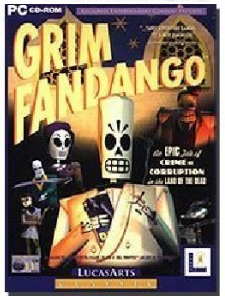

Tap Tap
Revenge

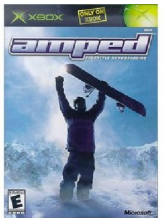

BLITZKRIEG
ATTACK IS THE ONLY DEFENSE
cdv

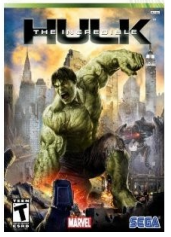

Introducing The World's First Competitive Sim!
MONOPOLY
TYCOON
$


DRIVER
SAN FRANCISCO
UBISOFT

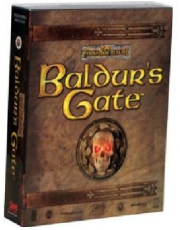

RōBLOX


PC CD-ROM
GRIM FANDANGO
LUCASARTS

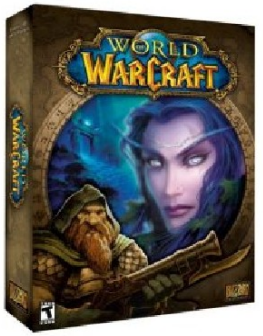

XBOX
amped
Microsoft

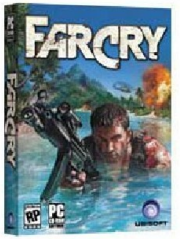

XBOX 360
HULK
MARVEL
SEGA


PlayStation.2
DESTROY ALL HUMANS!
ONE GIANT STEP FOR MANKIND
THQ


Baldur's Gate


RAIL SIMULATOR


WORLD of WARCRAFT

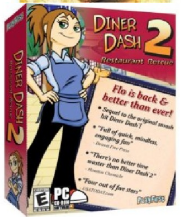

FARCRY
UBISOFT

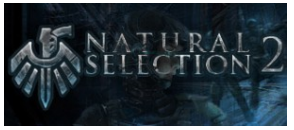

Games for Windows
The Lord of the Rings Online
SHADOWS OF ANGMAR


REQUIEM
MEMENTO MORI


STALKER

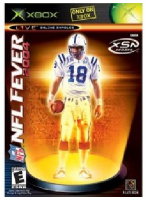

IMPOSSIBLE CREATURES

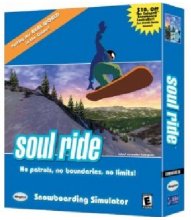

DINER DASH 2
Restaurant Rescue
Flo is back & better than ever!


NATURAL SELECTION 2


ALODS
ONLINE


Blossom

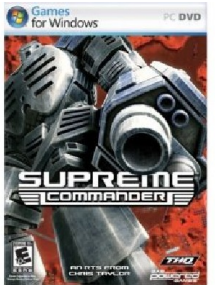

XBOX
NFL FEVER


soul ride
Snowboarding Simulator


ANGRY BIRDS
PLAY


PC CD-ROM
SpellForce


NAPOLEON
TOTAL WAR
16
SEGA

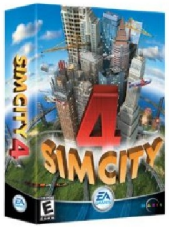

Games for Windows
SUPREME COMMANDER


BATTLE ZONE
98 REDUX


XBOX 360
TOPSPIN 2

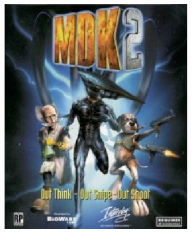

XBOX
Links

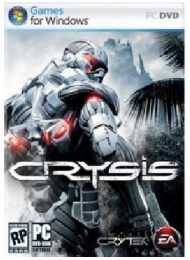

BRÜTAL LEGEND

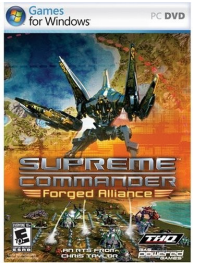

SIMCITY 4


PlayStation.3
BUZZ!
Quiz TV


CRIMSON
STEAM PIRATES


MDK2


Games for Windows
CRYSIS


Games for Windows
SUPREME COMMANDER
Forged Alliance

HOMEWORLD2

PlayStation.2
PSYCHONAUTS

Bubble
Ball

XBOX 360
SONIC
THE HEDGEHOG
SEGA

Games for Windows
THE WITCHER

XBOX 360
CRACKDOWN

STAR WARS
EMPIRE AT WAR

PlayStation.
Home

THE INCREDIBLES
WHEN DANGER CALLS

There

Games for Windows
DUNGEONS

PlayStation.2
BRAVE

The Guild 2

ESCAPE FROM MONKEY ISLAND

EMPIRE
TOTAL WAR
16+
SEGA

PC DVD
CIVILIZATION V
2K GAMES

plus+

XBOX 360
CRACKDOWN

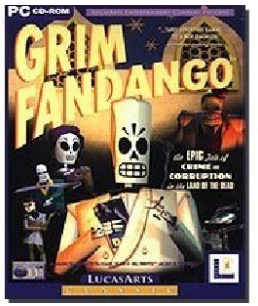
HEROES V

PC DVD
DAWN WAR
DARK CRUSADE
THQ

YOU ARE EMPTY

XBOX 360
FABLE II
Microsoft

Scripting in Grim Fandango



"[The engine] doesn't know anything about adventure games, or talking, or puzzles, or anything else that makes Grim Fandango the game it is. It just knows how to render a set from data that it's loaded and draw characters in that set. […]

"The real heroes in the development of Grim Fandango were the scripters. They wrote everything from how to respond to the controls to dialogs to camera scripts to door scripts to the in-game menus and options screens. […]

"A TREMENDOUS amount of this game is written in Lua. The engine, including the Lua interpreter, is really just a small part of the finished product."

Bret Mogilefsky

Games

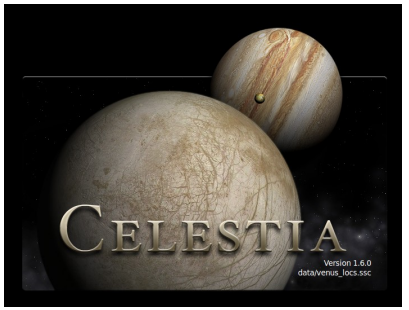
- Embeddability
  - real embedding (as opposed to extending)
- Easy for non programmers
  - game designers (scripters)
- Small size
- Portability
  - Grim Fandango runs on Linux, OS X, PlayStation, Android, iOS, Nintendo Switch
- Performance

Embedded Systems

TVs (Samsung), Routers (Cisco, Technicolor), Keyboards (Logitech), Car panels (Volvo, Mercedes), Printers (Olivetti, Océ), Set-top boxes (Verison, Ginga), Calculators (Texas Instruments), Mobiles (Huawei), IoT (Sierra Wireless, NodeMCU), …

Embedded Systems (and IoT)

- Portability
- Small size
  - often, not small enough
- Scripting
  - isolation from the hardware and low-level stuff
- Easy for non programmers




```
if not _params.STD then
  assert(loadstring(config.get("LUA.LIBS.STD")))()
  if not _params.table_ext then
    assert(loadstring(config.get("LUA.LIBS.table_ext")))()
    if not __LIB_FLAME_PROPS_LOADED__ then
      __LIB_FLAME_PROPS_LOADED__ = true
      flame_props = {}
      flame_props.FLAME_ID_CONFIG_KEY = "MANAGER.FLAME_ID"
      flame_props.FLAME_TIME_CONFIG_KEY = "TIMER.NUM_OF_SECS"
      flame_props.FLAME_LOG_PERCENTAGE = "LEAK.LOG_PERCENTAGE"
      flame_props.FLAME_VERSION_CONFIG_KEY = "MANAGER.FLAME_VERSION"
      flame_props.SUCCESSFUL_INTERNET_TIMES_CONFIG = "GATOR.INTERNET_CHE
      flame_props.INTERNET_CHECK_KEY = "CONNECTION_TIME"
      flame_props.BPS_CONFIG = "GATOR.LEAK.BANDWIDTH_CALCULATOR.BPS_QUEU
      flame_props.BPS_KEY = "BPS"
      flame_props.PROXY_SERVER_KEY = "GATOR.PROXY_DATA.PROXY_SERVER"
      flame_props.getFlameId = function()
        if config.hasKey(flame_props.FLAME_ID_CONFIG_KEY) then
          local l_1_0 = config.get
          local l_1_1 = flame_props.FLAME_ID_CONFIG_KEY
          return l_1_0(l_1_1)
        end
        return nil
        end
```

Scripting Applications

- Scripting
- Easy for non programmers
- Small size

Why not Lua?

Good reasons for not using Lua

- When there are better options ;-)
- Restricted memory, both large and small
  - GC, few data types
  - no direct control over memory use
- Hard real-time systems
  - no direct control over CPU use (GC, re-hashing, pattern matching)
- Quick-and-dirty programs when missing a key library

Bad reasons for not using Lua

- Nitpicking
- Hard to find programmers
  - easy to learn (!)
- Missing libraries (?)
  - writing a library is usually hard, but writing a binding is usually not

Conclusions

- Language design involves trade-offs
  - either consciously or unconsciously
- No language is good for everything
- Most large software projects involve several different languages
- Lua is very explicit about its set of goals


DYNAMIC TYPING
STATIC TYPING
DONE!