LuaJIT

(25')

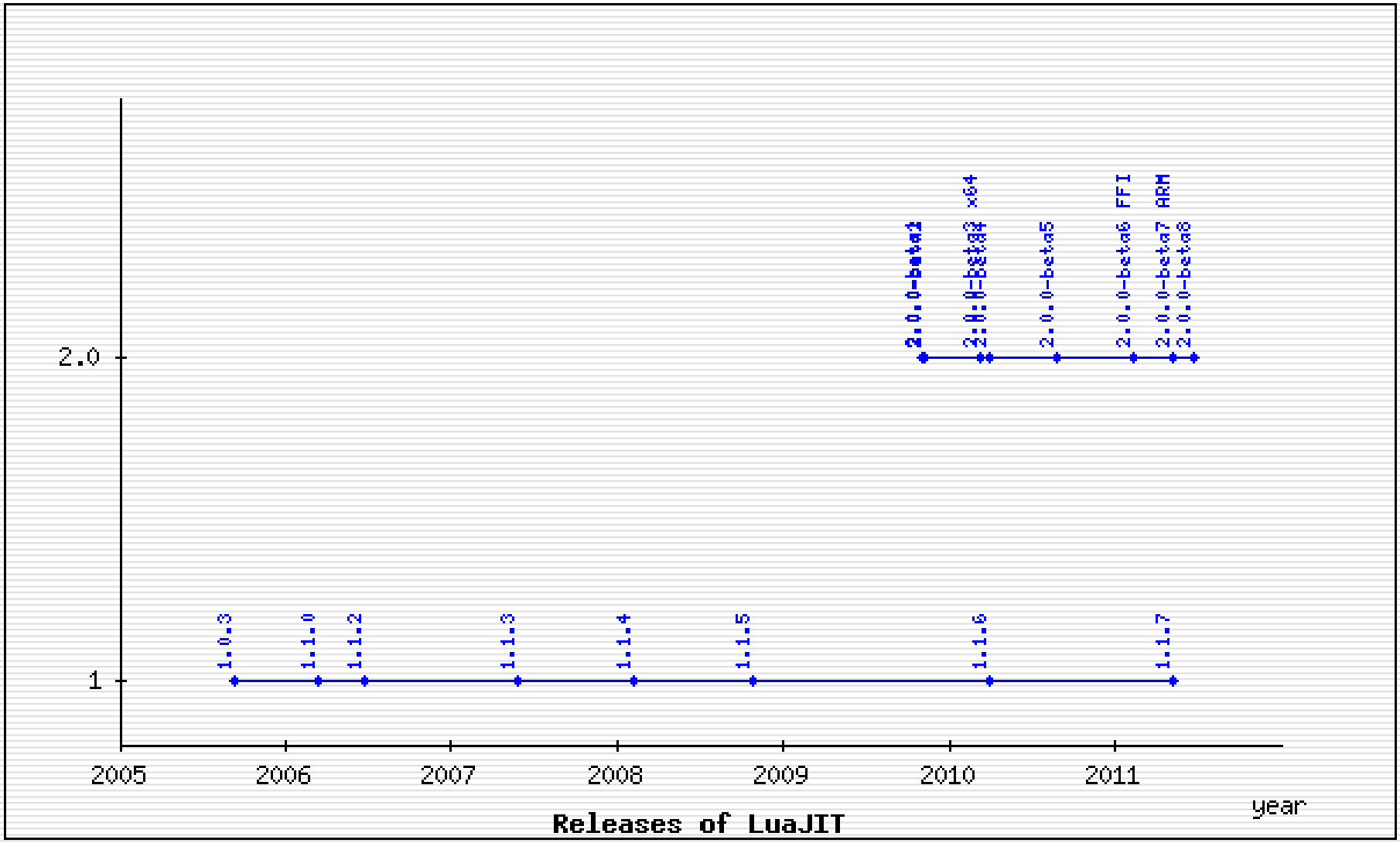
François Perrad

francois.perrad@gadz.org

LuaJIT : Overview

- LuaJIT is a Just-In-Time Compiler for the Lua 5.1 language
- Lead dev : Mike Pall
- started on 2005
- Code size ~ 42 KLoC of C
- Binary ~ 300 Kb
- License : MIT (like Lua)
- public Git repository
- use Lua mailing list

History of releases



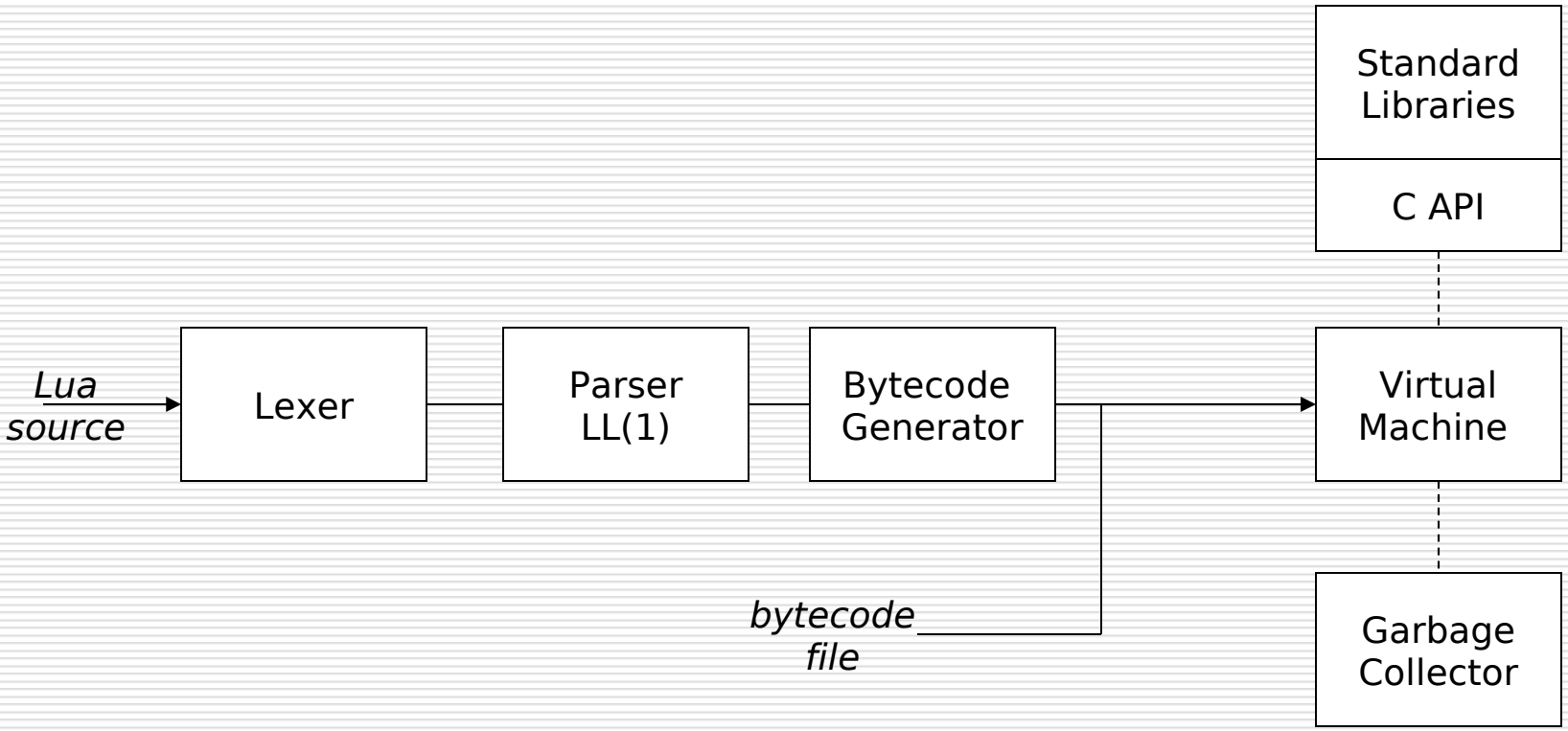
Sponsors for new features

- x64 port
  - Athena Capital Research : 5 k€
  - Google Inc. : 8 k
- PPC port
  - PowerPC/e500v2
    - anon. corporate
  - PowerPC/e300
    - anon. corporate
- ARM port
  - QUALCOMM Inc.
    - ARMv5 architecture with soft FP
    - dual number VM
- Bytecode load/save
  - anon. Corporate

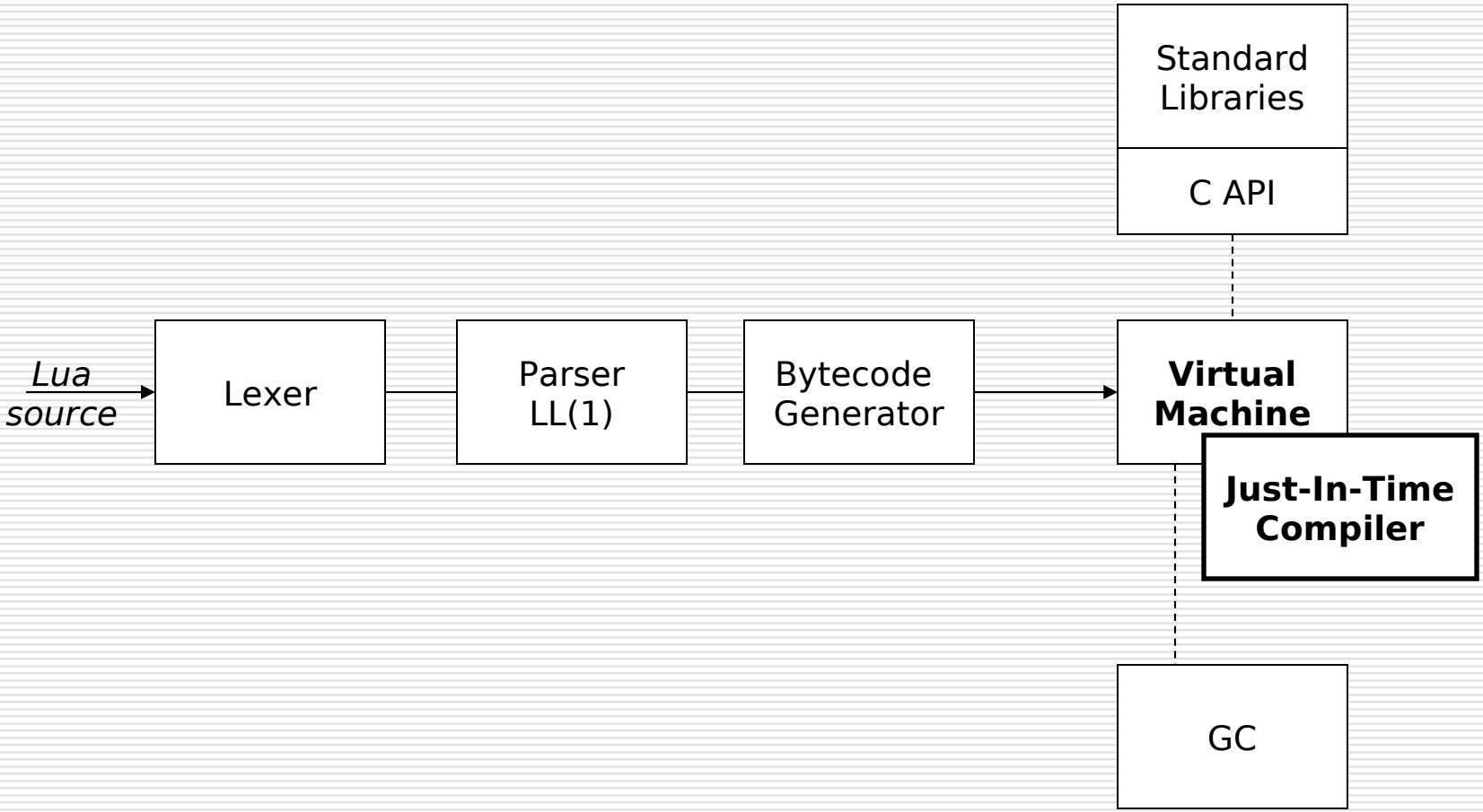
Compatibility with *standard* Lua

- Language Lua 5.1
  - without backward compatibility with 5.0
- C API 5.1
  - API : source compatible
  - ABI : binaire compatible (linker/dynamic loader)
    - ie. LuaSocket build with Lua works with LuaJIT
- Not with bytecode of Lua 5.1
  - which is an detail of implementation, not a part of Lua 5.1 specifications (ref. man.)

Design of *standard* Lua



Design of LuaJIT



DynASM

- a **Dynamic Assembler** for code generation engines
- developed for LuaJIT, but might be useful for other projects
- Search with Google Scholar :
  - Trace Compiler
  - JIT Compiler
  - Dynamic Language Optimizations
  - SSA Form
  - Linear Scan Register Allocation

Factorial

```
$ cat fact.lua
local function factorial (n)
    local a = 1
    for i = 1, n do
        a = a * i
    end
    return a
end

for a = 1, 1000 do  --> trigger the JIT
   print(factorial(7))
end
```

Standard Lua bytecode

```
$ luac –l fact.lua
function <fact.lua:2,8> (9 instructions, 36 bytes at 0x
1 param, 6 slots, 0 upvalues, 6 locals, 1 constant, 0 f
    1   [3]    LOADK      1 -1    ; 1
    2   [4]    LOADK      2 -1    ; 1
    3   [4]    MOVE       3 0
    4   [4]    LOADK      4 -1    ; 1
    5   [4]    FORPREP    2 1     ; to 7
    6   [5]    MUL        1 1 5
    7   [4]    FORLOOP    2 -2    ; to 6
    8   [7]    RETURN     1 2
    9   [8]    RETURN     0 1
```

LuaJIT bytecode

```
$ luajit -b -l fact.lua
-- BYTECODE -- fact.lua:2-8
0001    KSHORT   1   1
0002    KSHORT   2   1
0003    MOV      3   0
0004    KSHORT   4   1
0005    FORI     2 => 0008
0006 => MULVV    1   1   5
0007    FORL     2 => 0006
0008 => RET1     1   2
```

LuaJIT assembler

```
$ luajit –jdump fact.lua
---- TRACE 1 mcode 115
     … <cut> …
->LOOP:
b771ffe0  xorps xmm6, xmm6
b771ffe3  cvtsi2sd xmm6, edi
b771ffe7  mulsd xmm7, xmm6
b771ffeb  add edi, +0x01
b771ffee  cmp edi, eax
b771fff0  jle 0xb771ffe0 ->LOOP
b771fff2  jmp 0xb7718014 ->3
---- TRACE 1 stop -> loop
```

Performances

- One of the fastest
- No longer on shootout
  - The Computer Language Benchmark Game
  - now, only the *standard* Lua
- See this blog : The speed, size and dependability of programming languages

FFI library

- Foreign Function Interface
- allows calling functions in DLLs or shared libraries
- **parses plain C declarations**

Hello World with FFI

```
local ffi = require "ffi"

ffi.cdef [[
int printf(const char *fmt, ...);
]]

ffi.C.printf("Hello %s!", "world")
```

Another Sample

```
local ffi = require "ffi"
ffi.cdef [[
unsigned long compressBound(unsigned long sourceLen);
int compress2(uint8_t *dest, unsigned long *destLen, const
   uint8_t *source, unsigned long sourceLen, int level);
]]
local zlib = ffi.load "z"

local function compress (txt)
  local n = zlib.compressBound(#txt)
  local buf = ffi.new("uint8_t[?]", n)
  local buflen = ffi.new("unsigned long[1]")
  local res = zlib.compress2(buf, buflen, txt, #txt, 9)
  assert(res == 0)
  return ffi.string(buf, buflen[0])
end
```

Bibliography / Webography

- www.lua.org
- www.luajit.org