



# RAPTORJIT

a fast, dynamic systems programming language

Max Rottenkolber  
max@mr.gy  
@eugeneia\_

# Hello, World!

Hi, I'm Max <max@mr.gy>.

Open source systems hacker

Currently dabbling in high-performance networking applications

# Lua

Simple, minimalistic, high-level language,  
Schemeish semantics, Pascalesque syntax

First class functions, multiple return values, prototype OOP

Central data structure: table (sparse array/hash map hybrid)

Canonical implementation: PUC Lua (simple, embeddable  
interpreter)

# Lua

```
local function hello (name)
    return "Hello, "..name.."!"
end

function greet (name, greeting)
    greeting = greeting or hello
    print(greeting(name or "World"))
end
```



# LuaJIT

Implements a dialect of Lua (5.1½ + goodies)

Strong JIT compiler, efficient implementation (performance competitive with C)

Can express close-to-the-metal programs (native C data access)

Good language for systems programming?

# LuaJIT

```
local p = ffi.new [[struct {  
    uint16_t length;  
    char data[10000];  
}]]  
  
local msg = "Hello, World"  
  
p.length = math.min(#msg, ffi.sizeof(p.data))  
ffi.copy(p.data, msg, p.length)
```



# RaptorJIT

Fork of LuaJIT. Goal: to be a really good systems language

Simplify implementation, improve maintainability

Improve JIT for heavy duty server applications (eliminate performance pitfalls, unexpected JIT behavior, provide more reliable performance)

Add features (zero-overhead profiler, introspection tools, + many more to come?)



# Simplify and Maintain

---

 **Big bang: Remove all the features that I can live without** ●

#5 by lukego was merged on Mar 12, 2017

---

Removed support for all architectures except x86\_64, Windows, 32-bit heap, ...

Got rid of a TONNE of #ifdefs

Cut code to maintain by ~50%



# Simplify and Maintain

LuaJIT interpreter used to be handwritten assembly for each supported architecture.

 **WIP: Max's C VM branch** ✓

#254 opened on Jun 5, 2019 by eugeneia


We almost completed rewriting the interpreter in C (easier to port, easier to change!)

Rationale: we spent 99% of time in compiled code, no use for an overly optimized interpreter

# Simplify and Maintain

lj\_str.c: Remove special-case string interning fast-path  
#150

 Merged

lukego merged 1 commit into `raptorjit:master` from `lukego:reoptimize-string-intern`  on Jan 15, 2018



RAPTORJIT

# Simplify and Maintain

“Fast-path” bad because:

- tricky custom memcmp routine that needs to be maintained
- slower than "slow-path" (stock memcmp on modern Linux/x86)
- confusing performance behavior: totally unrelated memory allocation could bias an important buffer towards the “fast-path” and impact overall performance

# Simplify and Maintain

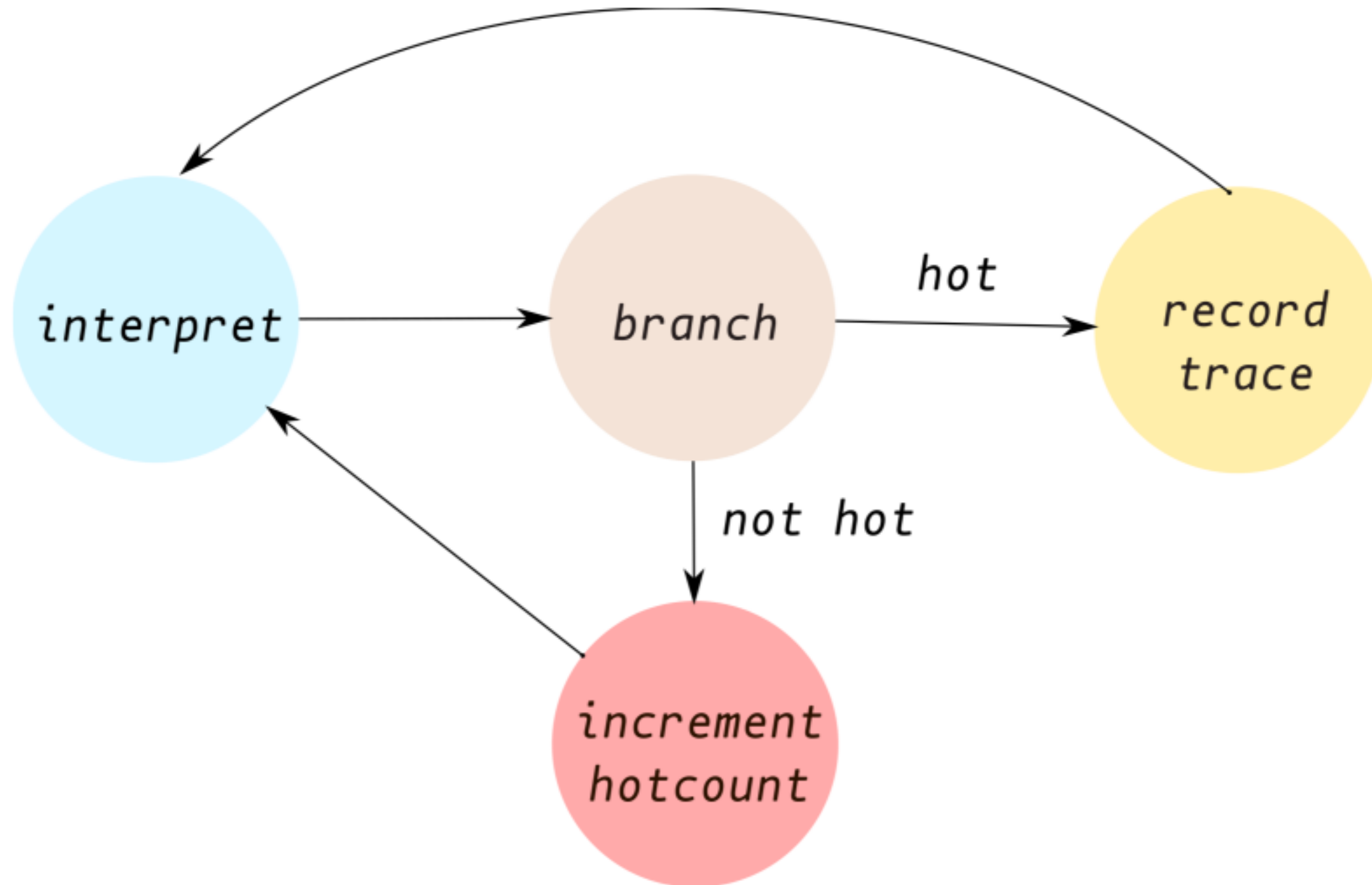
*“The fast-path code was written in 2010 and a lot has happened since then [...]. I think the optimization had simply bit-rotted.”*

# Hacking the JIT

LuaJIT acts as a “best-effort for all use cases” drop-in replacement for PUC Lua. (Fast JIT, when it fails it drops into fast interpreter, huge to solid gains in any case.)

Can we do better for a narrowed use-case?

# Hacking the JIT



# Hacking the JIT

Don't treat the compiler as a black box gifted from a geni(e/ous)!

Study and understand the JIT

Formulate design goals & implement them

---

! IAF: Where can a trace end? Why? IAF

#103 opened on Sep 9, 2017 by lukego

---

! IAF: What is the difference between a root trace and a side trace? IAF help wanted  
question

#99 opened on Sep 5, 2017 by lukego

---

! Goal: Avoid "high-impact medium-generality" optimizations goal

#148 opened on Dec 22, 2017 by lukego



# Hacking the JIT


LuaJIT aggressively blacklists codepaths that it fails to compile (Good for short running programs, bad for server applications.)

RaptorJIT spends more effort to find traces and provide stable, predictable performance needed for heavy duty server apps.



# Hacking the JIT

JIT heuristic updates for stable performance  
[experimental] #101

 Merged

lukego merged 4 commits into `raptorjit:master` from `lukego:long-running-stable`  on Dec 11, 2017

# Hacking the JIT

LuaJIT doesn't consider the time domain when selecting traces (Causes new traces to be compiled long after initial warmup for code that really isn't hot at all!

Maybe RaptorJIT should only compile code that is actually executed frequently?

 **[draft] lj\_trace.c: clear all hotcounts every second ✓**

#260 opened on Oct 9, 2019 by eugeneia

# New Features

Replaced LuaJIT profiler with low-overhead, “always on” trace profiler and Auditlog (flight recorder).

Created an interactive tool (Studio) to help understand trace and profile data.

Inspector on a ByteString ('with import <studio>;')

a R JITProcess

Trace List Trace Map VM Profile Events Items Raw Meta

Profiler datasets (VMProfile)

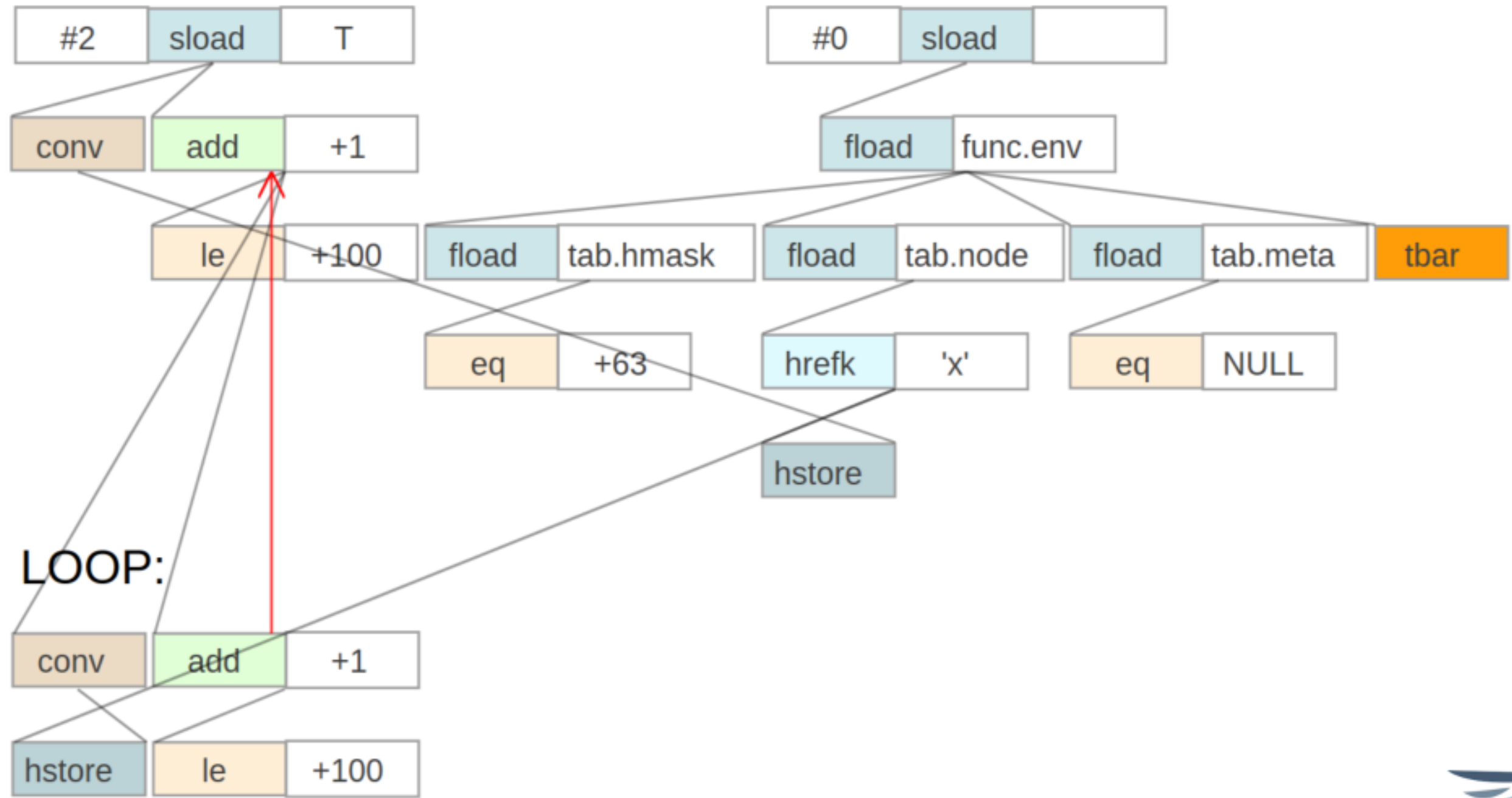
Profile	Samples	Mcode	VM	GC
apps.intel_mp.intel_mp	5099	99.9%	0.1%	0.0%
apps.ipv4.arp	910	99.9%	0.1%	0.0%
apps.ipv4.echo	567	100.0%	0.0%	0.0%
apps.ipv4.fragment	620	99.8%	0.2%	0.0%
apps.ipv4.reassemble	37207	12.5%	71.8%	15.7%
apps.ipv6.echo	640	100.0%	0.0%	0.0%
apps.ipv6.fragment	1470	100.0%	0.0%	0.0%
apps.ipv6.reassemble	599	97.7%	2.3%	0.0%
apps.lwaftr.lwaftr	11678	97.7%	0.1%	2.2%
apps.lwaftr.ndp	1029	97.2%	2.3%	0.5%
engine	1858	79.2%	20.6%	0.2%
program	0	-	-	-

Source code locations of root traces that are hot in the selected profile

Location	Samples	Mcode	VM	GC	#Root	#Side
(app)breathe:14 (core/app.lua:590)	32523	0.0%	82.1%	17.9%	9	25
(link)receive:1 (core/link.lua:48)	1575	99.6%	0.0%	0.4%	1	0
(lib)htons:1 (core/lib.lua:379)	1108	100.0%	0.0%	0.0%	1	0
(counter)add:1 (core/counter.lua:91)	1091	99.0%	0.3%	0.7%	4	7
(reassemble)ipv4_packet_has_valid_length:1 (apps/ipv4/870)		100.0%	0.0%	0.0%	1	0
(app)with_restart:1 (core/app.lua:128)	18	77.8%	22.2%	0.0%	5	4
(alarms)Alarm:check:1 (lib/yang/alarms.lua:691)	9	100.0%	0.0%	0.0%	1	0
(alarms)Encoder:uint32:1 (lib/ptree/alarms.lua:77)	8	0.0%	75.0%	25.0%	1	1
(app)now:1 (core/app.lua:121)	4	0.0%	100.0%	0.0%	2	4
(link)nreadable:1 (core/link.lua:86)	1	0.0%	100.0%	0.0%	1	1

Root traces starting at selected location (and their side-traces as children)

Trace	Samples	Link	Mcode	VM	GC	Start line	Stop line
98	32521	interp	0.0%	82.1%	17.9%	(app)breathe:15 (core/app.lua:591)	-
▼ 91	2	loop	0.0%	100.0%	0.0%	(app)breathe:15 (core/app.lua:591)	-
97	0	->56	-	-	-	(app)breathe:15 (core/app.lua:591)	-
143	0	->116-	-	-	-	(reassemble)Reassembler:push:14 (apps/ipv6/reassembl-	-
174	0	->116-	-	-	-	(link)nreadable:3 (core/link.lua:88)	-
▼ 87	0	loop	-	-	-	(app)breathe:15 (core/app.lua:591)	-
▼ 96	0	->63	-	-	-	(app)breathe:15 (core/app.lua:591)	-



**LOOP:**

# New Features

Lot's of experimentation (open to evolving the language)

## Add `jit.tracebarrier()` primitive #116

**Merged** lukego merged 2 commits into `raptorjit:master` from `lukego:jit-tracebarrier` on Nov 7, 2017

## Add `jit.unlikely()` primitive #143

**Closed** lukego wants to merge 1 commit into `raptorjit:master` from `lukego:trace-unlikely`

## Add `jit.seal(tab)` primitive (early version) #151

**Closed** lukego wants to merge 1 commit into `raptorjit:master` from `lukego:sealed`

# New Features

🚨 **Problem: Expensive heap-allocated boxes for 64-bit values (integers and pointers)**

**bug**

#91 opened on Aug 21, 2017 by lukego

🔗 **[WIP] Development branch for 96-bit VM ●**

#199 opened on Nov 25, 2018 by lukego

---

64-bit values don't fit into the VM's 64-bit tagged words.

If they did, that would simplify a lot of things!

# Future Goals

---

📌 [literature] Incremental Dynamic Code Generation with Trace Trees literature

#219 opened on Jan 16, 2019 by lukego

---

A weakness of LuaJIT are loops with unbiased branches

In this paper the authors claim to solve that problem.  
We'd love to solve it for RaptorJIT!



# Future Goals

---

## ! Goal: Safe FFI memory access

#156 opened on Feb 1, 2018 by lukego

All (FFI) type information is available at runtime, and the JIT is really good at hoisting/eliminating guards/checks.

We want to try to provide memory safety for operations on C data by default!

# Get Involved!?

<https://github.com/RaptorJIT/RaptorJIT>



RAPTORJIT