Introduction to OpenResty XRay

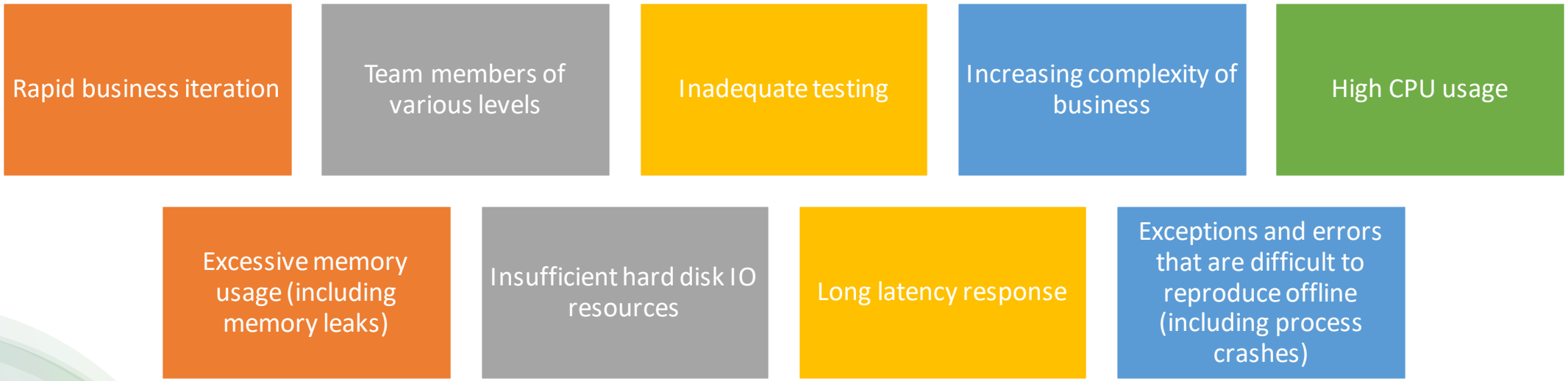
Deep monitoring, analyses and diagnoses for your online applications

Try OpenResty XRay for free

https://openresty.com/en/xray/

info@openresty.com

Challenges in the World of Software



Challenges in the Age of K8s/Docker Containers

Lots of containers, lots of applications, lots of distributions, lots of technology stacks

Minimized containers lack the most basic debugging tools

Minimized set of container permissions

Automatically discard and restart containers when something goes wrong. Software bugs are easily swept under the carpet

Virtualized containers, Microservices - further increase the software complexity

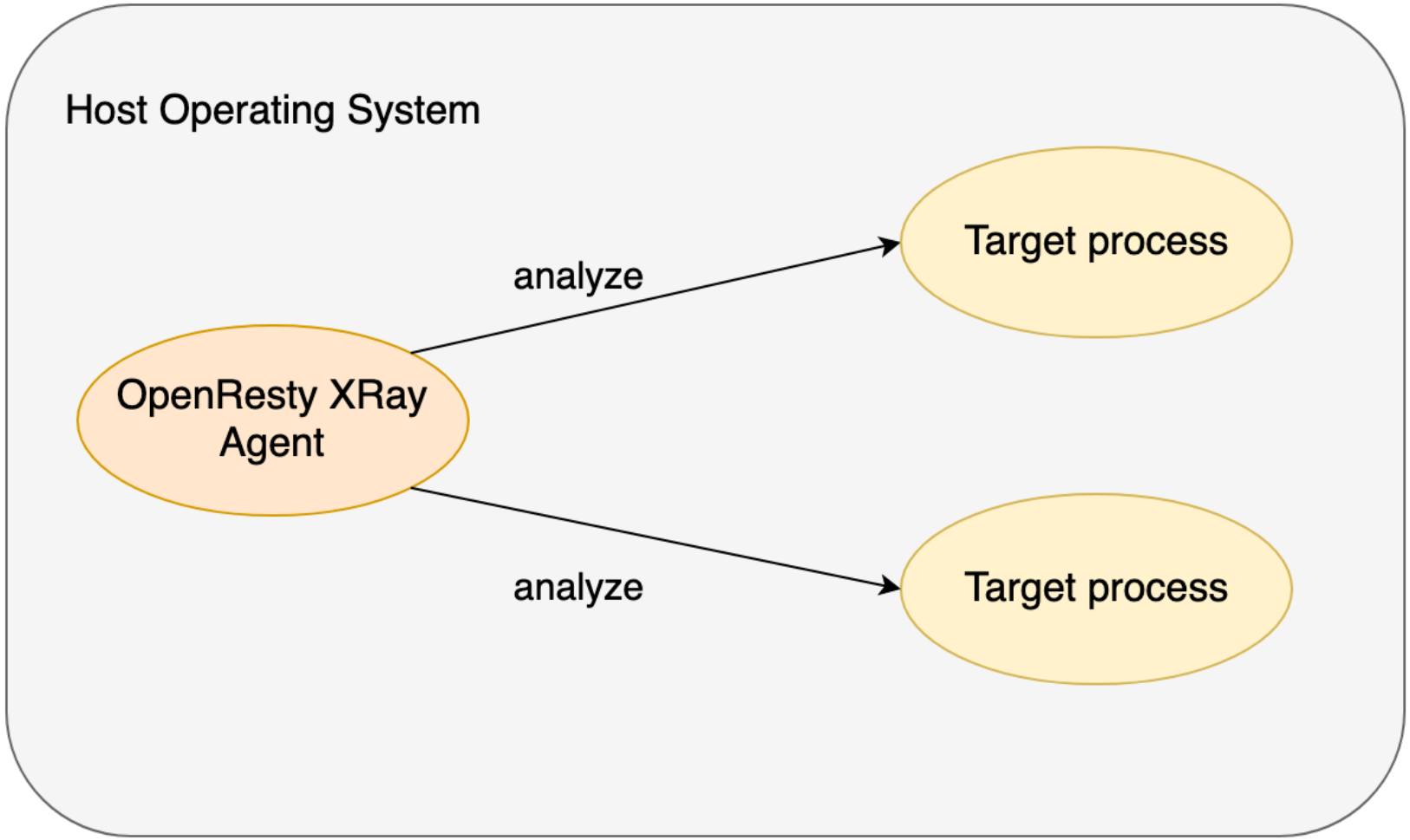
Disadvantages of Traditional Methods

- Invasive - need to modify applications
- Slow response
- Need the storage and analysis of big data
- Superficial indicators only
- Observations without causes
- Lack of in-depth full technology stack analysis and diagnosis
- Complex data collection and processing process, high overhead and error-prone

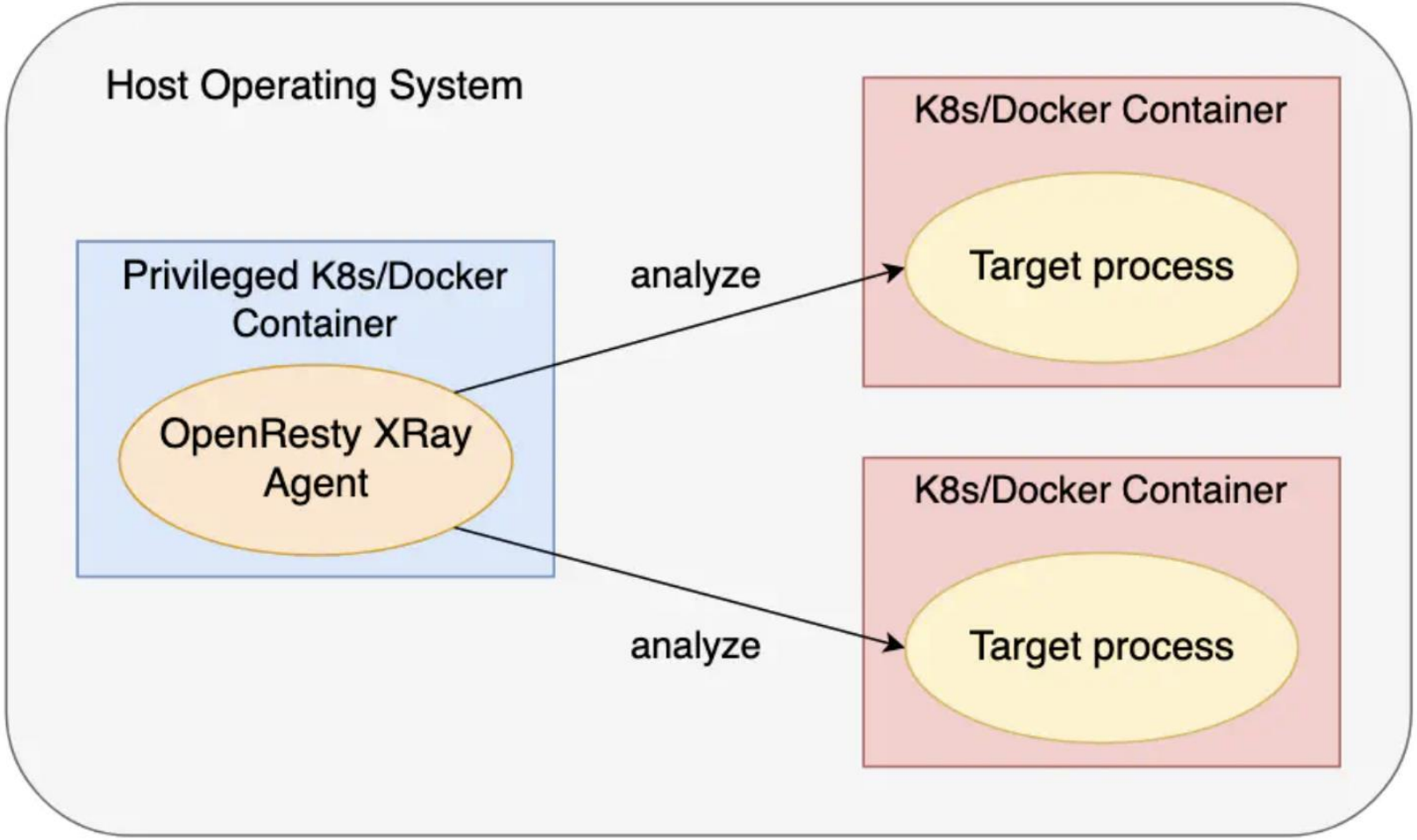
OpenResty XRay

- OpenResty XRay is a dynamic tracing product
- Enables real-time analysis of various cloud and server applications
- Treat running processes and containers as read-only databases and extract the necessary information to resolve performance issues, exceptions, errors, and security vulnerabilities
- With a knowledge base, inference engine, and hundreds of advanced analyzers
- Can diagnose and narrow down the root cause of deep problems without changing or affecting the target application

OpenResty XRay Analyzes Non-container Application Processes Directly



OpenResty XRay Penetrates Containers and Analyzes Applications

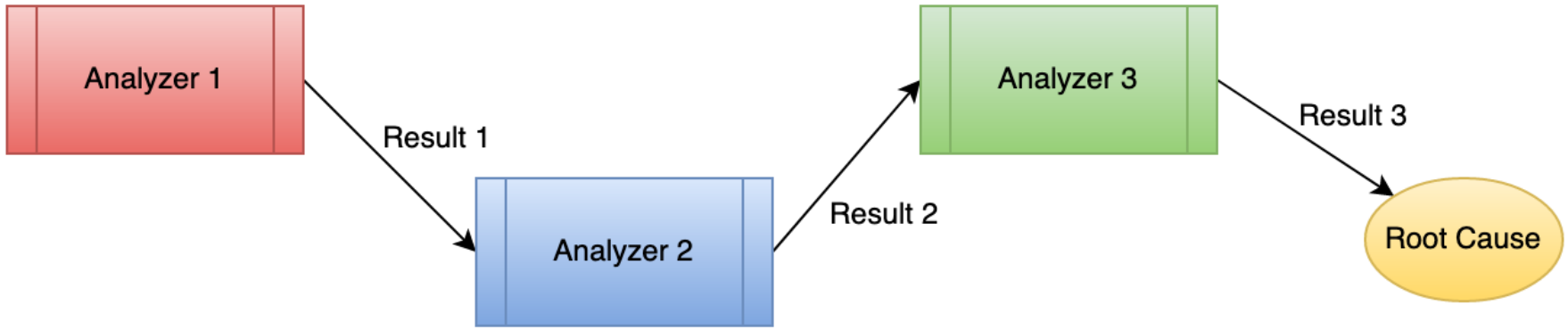


100% Non-Invasive

- No need to modify your application
- No need to add new plug-ins, modules, or patches to your application
- No need to inject any code into your applications
- No need to restart your application processes
- No need to use special startup or compilation options in your application
- No need to rebuild your existing application containers or application packages

OpenResty XRay Fully Automatic Sampling, Unattended Usage Mode

- Time sampling
- Event-driven sampling (CPU changes, memory changes, IO changes, exceptions and errors)
- Reasoning chains driven

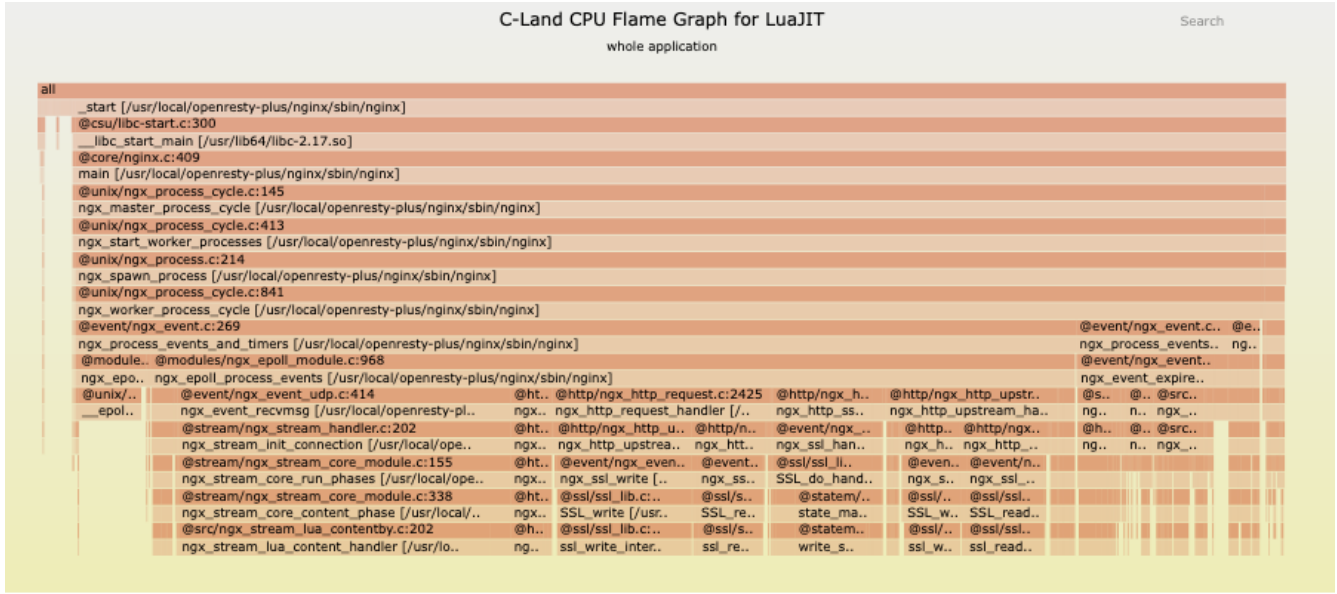


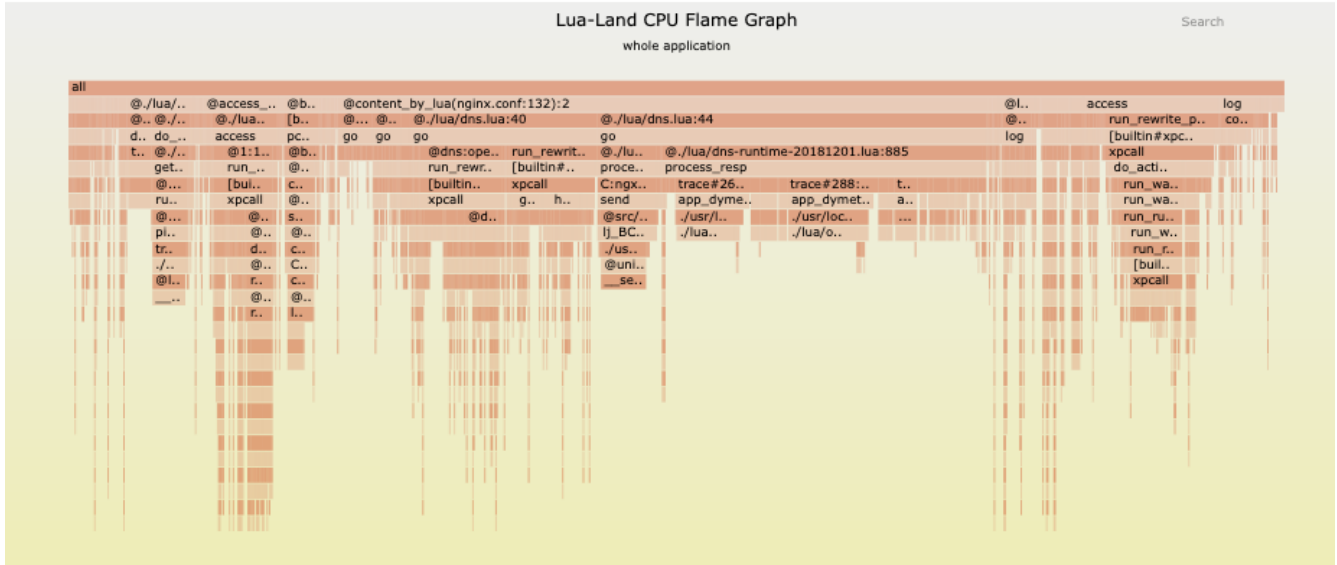
Extremely Low Performance Overhead

- Performance overhead is strictly 0 when not sampling
- Performance overhead is often not noticeable when sampling

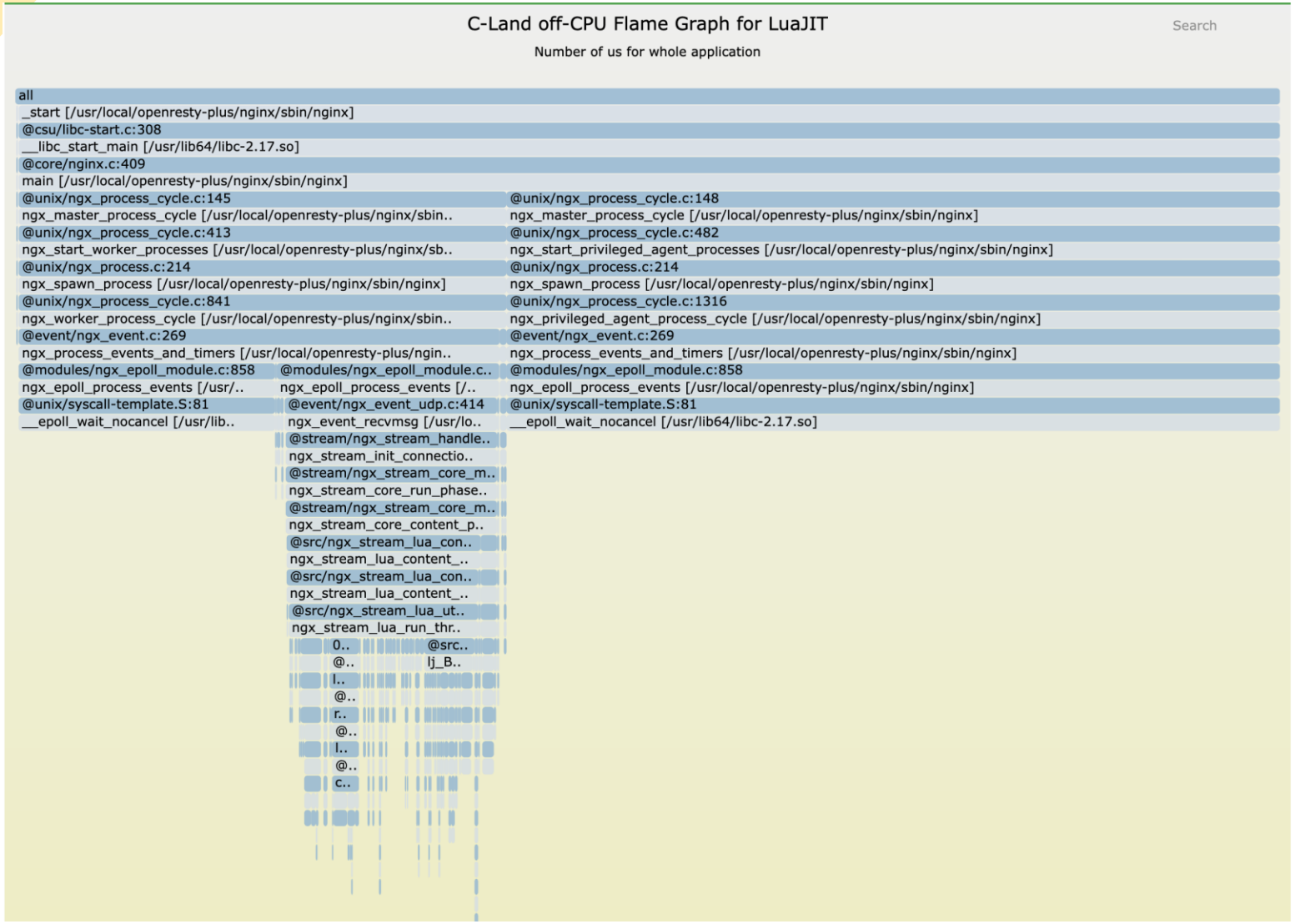
OpenResty XRay CPU Performance Analysis

- High CPU usage can reduce the system stability and quality of service, and even make services unavailable
- How CPU time is distributed over different code paths in different scenarios (flame graphs, automatic flame graph interpreter)
- Cover code paths of different software levels: business programming language level (Lua/Python/PHP/Perl/Go/etc.), system programming language level (C/C++/Rust), OS kernel level (network protocol stack/process scheduler/memory management/system calls)
- Examples of common CPU bottlenecks: duplicate computations (lack of cache), SSL handshake related, garbage collection (GC) overhead, dynamic memory allocation overhead, serialization and deserialization, unexpectedly frequent system calls, infinite loops, wrong regular expression matching, (third-party) software libraries with inefficient implementation, spinlock contention


C-Land CPU Flame Graph for LuaJIT
Search
whole application
all
_start [/usr/local/openresty-plus/nginx/sbin/nginx]
@csu/libc-start.c:300
__libc_start_main [/usr/lib64/libc-2.17.so]
@core/nginx.c:409
main [/usr/local/openresty-plus/nginx/sbin/nginx]
@unix/ngx_process_cycle.c:145
ngx_master_process_cycle [/usr/local/openresty-plus/nginx/sbin/nginx]
@unix/ngx_process_cycle.c:413
ngx_start_worker_processes [/usr/local/openresty-plus/nginx/sbin/nginx]
@unix/ngx_process.c:214
ngx_spawn_process [/usr/local/openresty-plus/nginx/sbin/nginx]
@unix/ngx_process_cycle.c:841
ngx_worker_process_cycle [/usr/local/openresty-plus/nginx/sbin/nginx]
@event/ngx_event.c:269
ngx_process_events_and_timers [/usr/local/openresty-plus/nginx/sbin/nginx]
@module..
@modules/ngx_epoll_module.c:968
ngx_epo..
ngx_epoll_process_events [/usr/local/openresty-plus/nginx/sbin/nginx]
@unix/..
__epol..
@event/ngx_event_udp.c:414
ngx_event_recvmsg [/usr/local/openresty-pl..
@stream/ngx_stream_handler.c:202
ngx_stream_init_connection [/usr/local/ope..
@stream/ngx_stream_core_module.c:155
ngx_stream_core_run_phases [/usr/local/ope..
@stream/ngx_stream_core_module.c:338
ngx_stream_core_content_phase [/usr/local/..
@src/ngx_stream_lua_contentby.c:202
ngx_stream_lua_content_handler [/usr/lo..
@ht..
ngx..
@http/ngx_http_request.c:2425
ngx_http_request_handler [/..
@http/ngx_http_u..
ngx_http_upstrea..
@event/ngx_even..
ngx_ssl_write [..
@ssl/ssl_lib.c:..
SSL_write [/usr..
@ssl/ssl_lib.c:..
ssl_write_inter..
@http/ngx_h..
ngx_htt..
@event/ngx_..
@ssl/ssl_li..
ngx_ss..
@ssl/s..
SSL_re..
@ssl/s..
ssl_re..
@http/ngx_http_upstr..
ngx_http_ss..
@http/n..
ngx_htt..
ngx_ssl_han..
@event/ngx_..
@ssl/ssl_han..
SSL_do_hand..
@statem/..
state_ma..
@statem..
write_s..
@http/ngx_http_upstream_ha..
@http..
ngx_h..
@even..
ngx_s..
@ssl/..
SSL_w..
@ssl/..
ssl_w..
@s..
@..
@src..
ng..
n..
ngx_..
@h..
@..
@src..
ng..
n..
ngx_..
@http/ngx..
ngx_http_..
@event/n..
ngx_ssl_..
@ssl/ssl..
@ssl/ssl..
SSL_read..
@ssl/ssl..
ssl_read..
@event/ngx_event.c..
@e..
ng..
ngx_process_events..
@event/ngx_event..
ngx_event_expire..

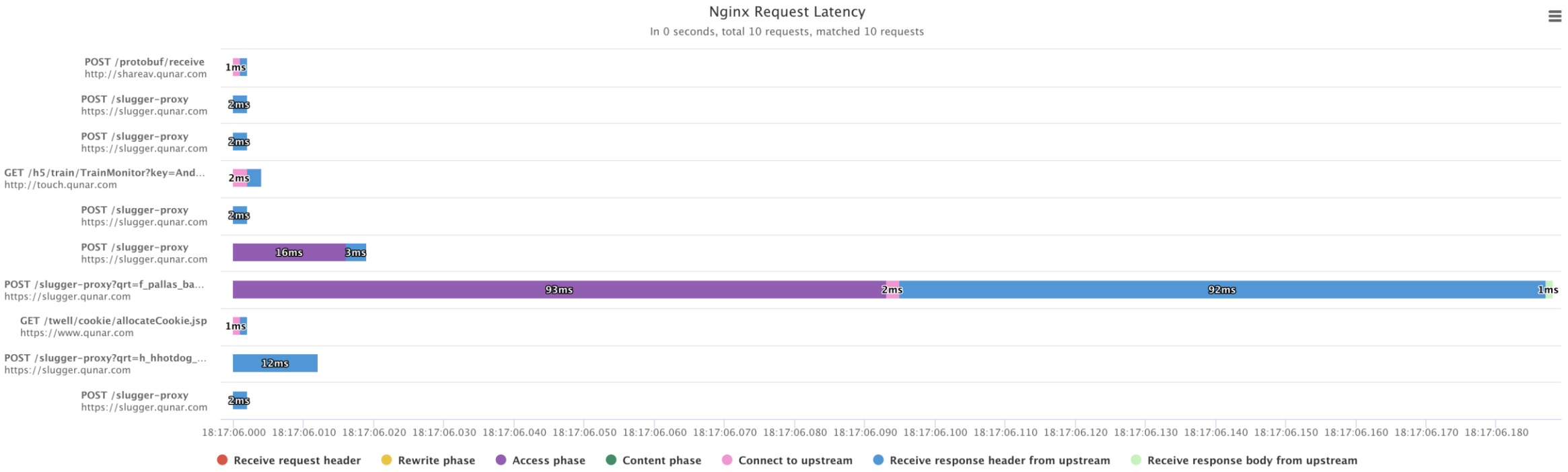

Lua-Land CPU Flame Graph
Search
whole application
all
@./lua/..
@access_..
@b..
@content_by_lua(nginx.conf:132):2
@l..
access
log
@..
@./..
@./lua..
[b..
@...
@..
@./lua/dns.lua:40
@./lua/dns.lua:44
run_rewrite_p..
co..
d..
do_..
access
pc..
go
go
go
go
log
[builtin#xpc..
t..
@./..
@1:1..
@b..
@dns:ope..
run_rewrit..
@./lu..
@./lua/dns-runtime-20181201.lua:885
xpcall
get..
run_..
@..
run_rewr..
[builtin#..
proce..
process_resp
do_acti..
@...
[bui..
c..
[builtin..
xpcall
C:ngx..
trace#26..
trace#288:..
t..
run_wa..
ru..
xpcall
@..
xpcall
g..
h..
send
app_dyme..
app_dymet..
a..
run_wa..
@...
@..
s..
@d..
@src/..
./usr/l..
./usr/loc..
...
run_ru..
pi..
@..
@..
lj_BC..
./lua..
./lua/o..
run_w..
tr..
d..
c..
./us..
run_r..
./..
@..
C..
@uni..
[buil..
@l..
r..
c..
__se..
xpcall
__..
@..
@..
r..
l..

OpenResty XRay CPU Blocking (off-CPU) Analysis



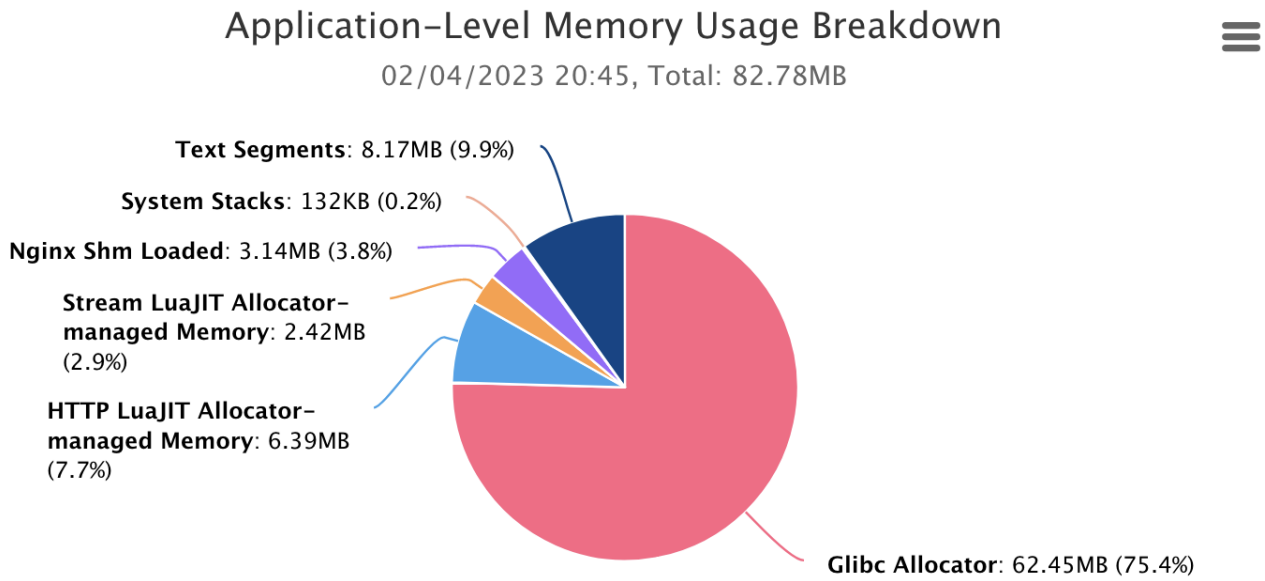
OpenResty XRay Analysis of Request Latency

- Breakdown the latency to different operation and processing phases of applications
- Precise packet capture, only capture network packets on problematic connections (including high latency, timeouts, connection errors, when upper layer applications report errors, etc.)
- Latency statistics of asynchronous non-blocking IO (e.g. the distribution of Lua concurrent yield time over Lua code paths)



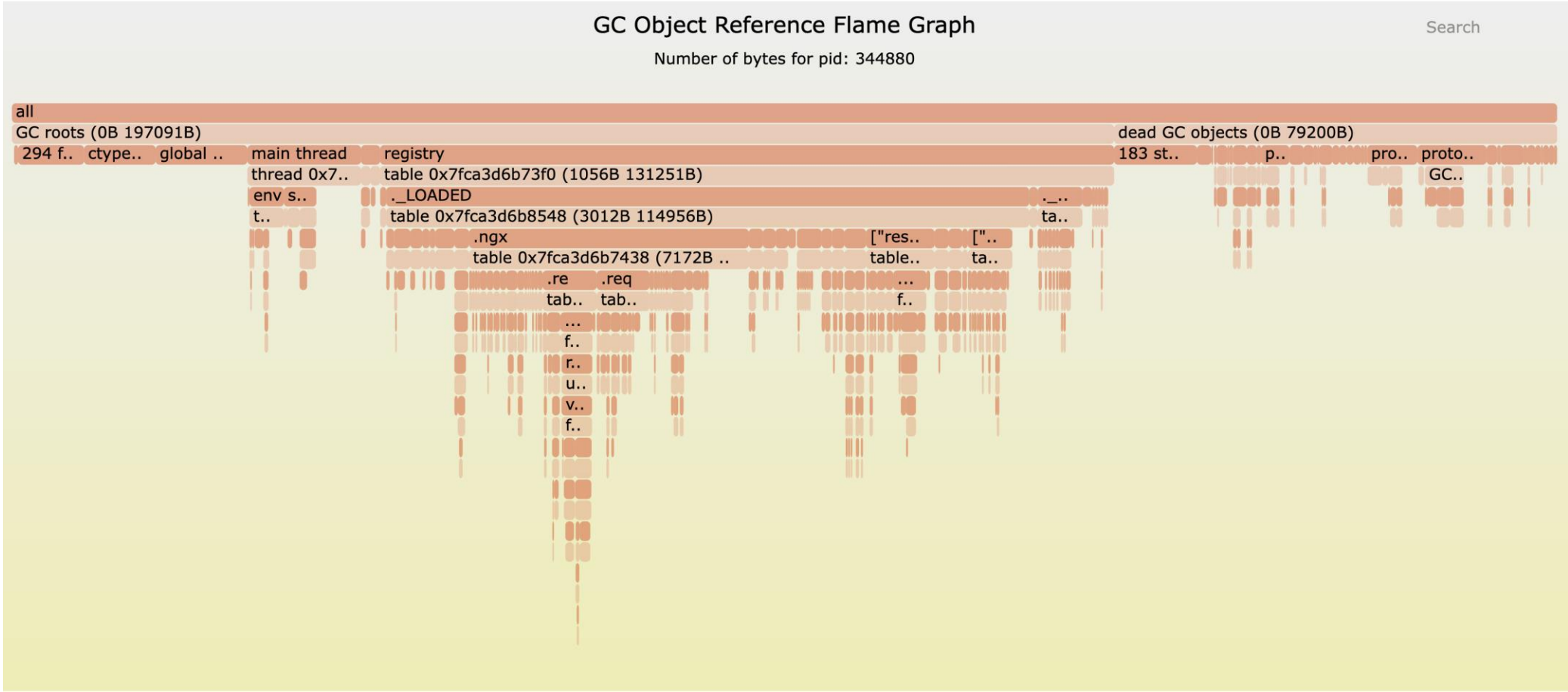
OpenResty XRay Memory Usage Analysis

- Memory usage of C memory allocators such as Glibc/Jemalloc (including Glibc memory fragmentation)
- How memory is distributed quantitatively over all GC objects (e.g. Lua objects, Python objects, PHP objects, etc.), by reference relationships between GC objects.
- Memory leak, memory fragmentation, or delayed release?



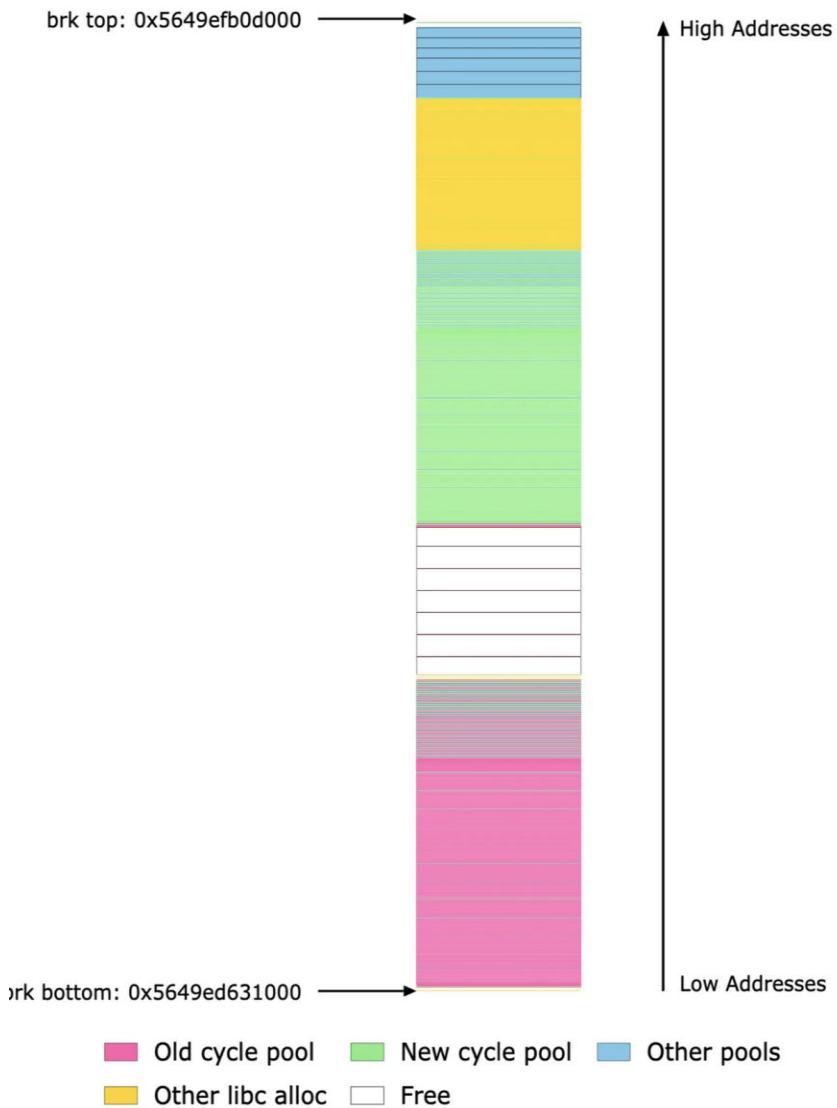
GC Object Reference Relationship Flame Graph

Quantitative visualization of how memory is distributed over all object reference paths



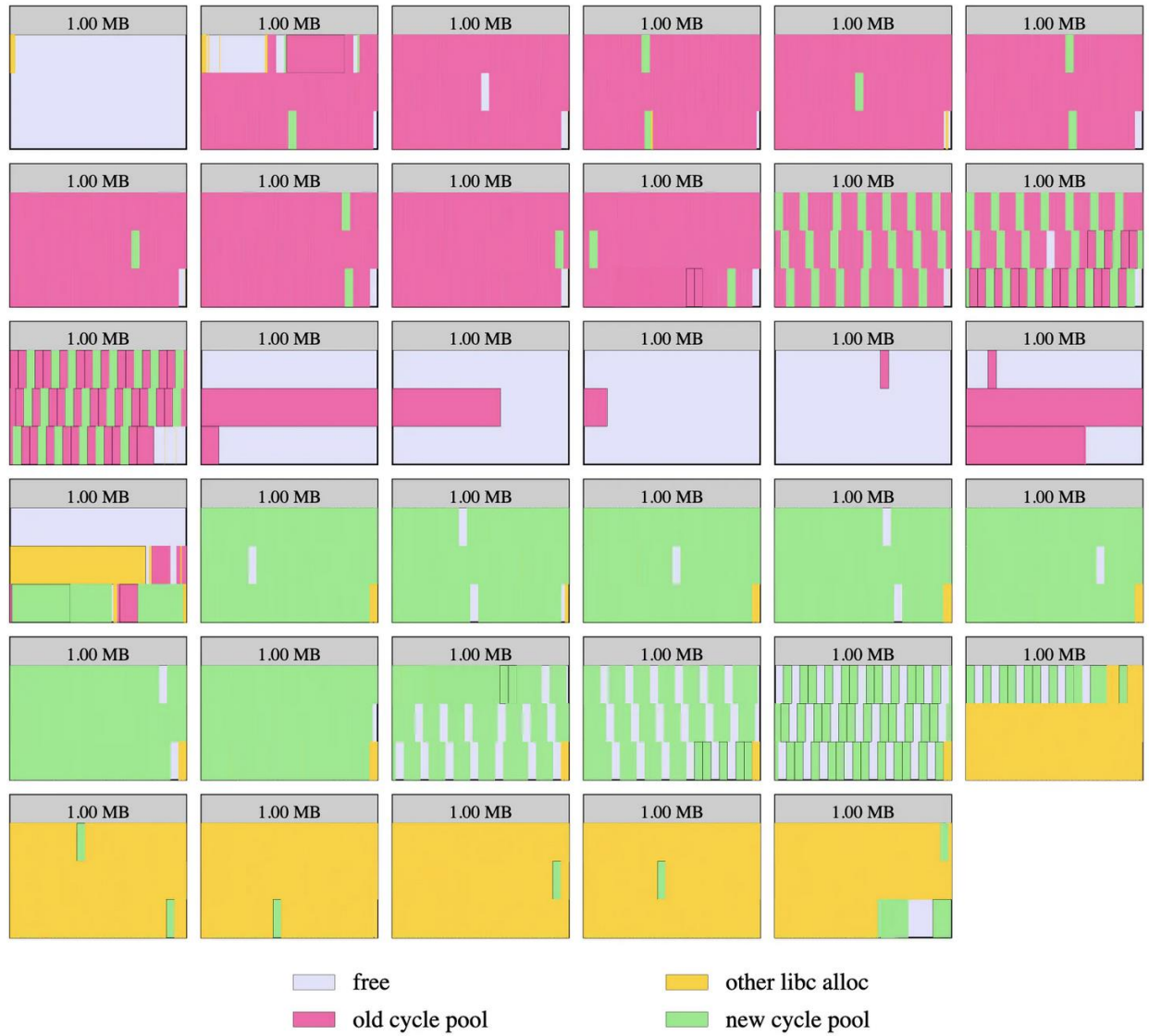
Nginx memory pool allocatons via the brk syscall

When the old and new nginx cycles coexist

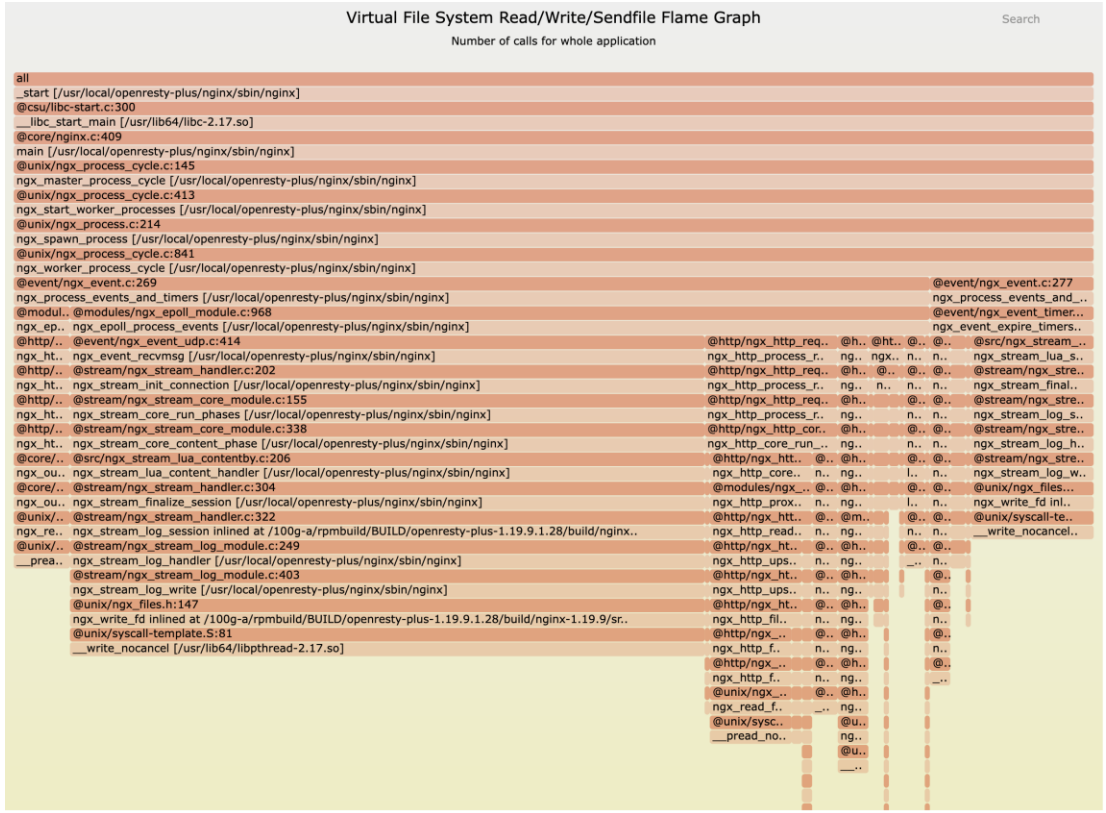


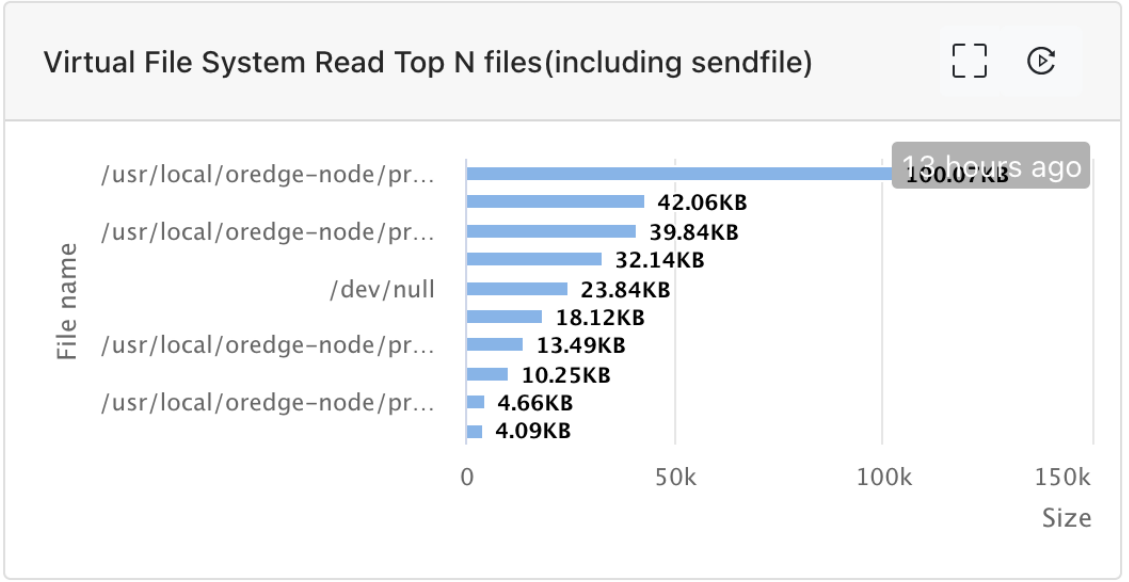
Nginx memory pool allocatons via the mmap syscall

For total 35.00 MB in 35 memory mappings with 49,464 chunks



OpenResty XRay File IO Performance Analysis



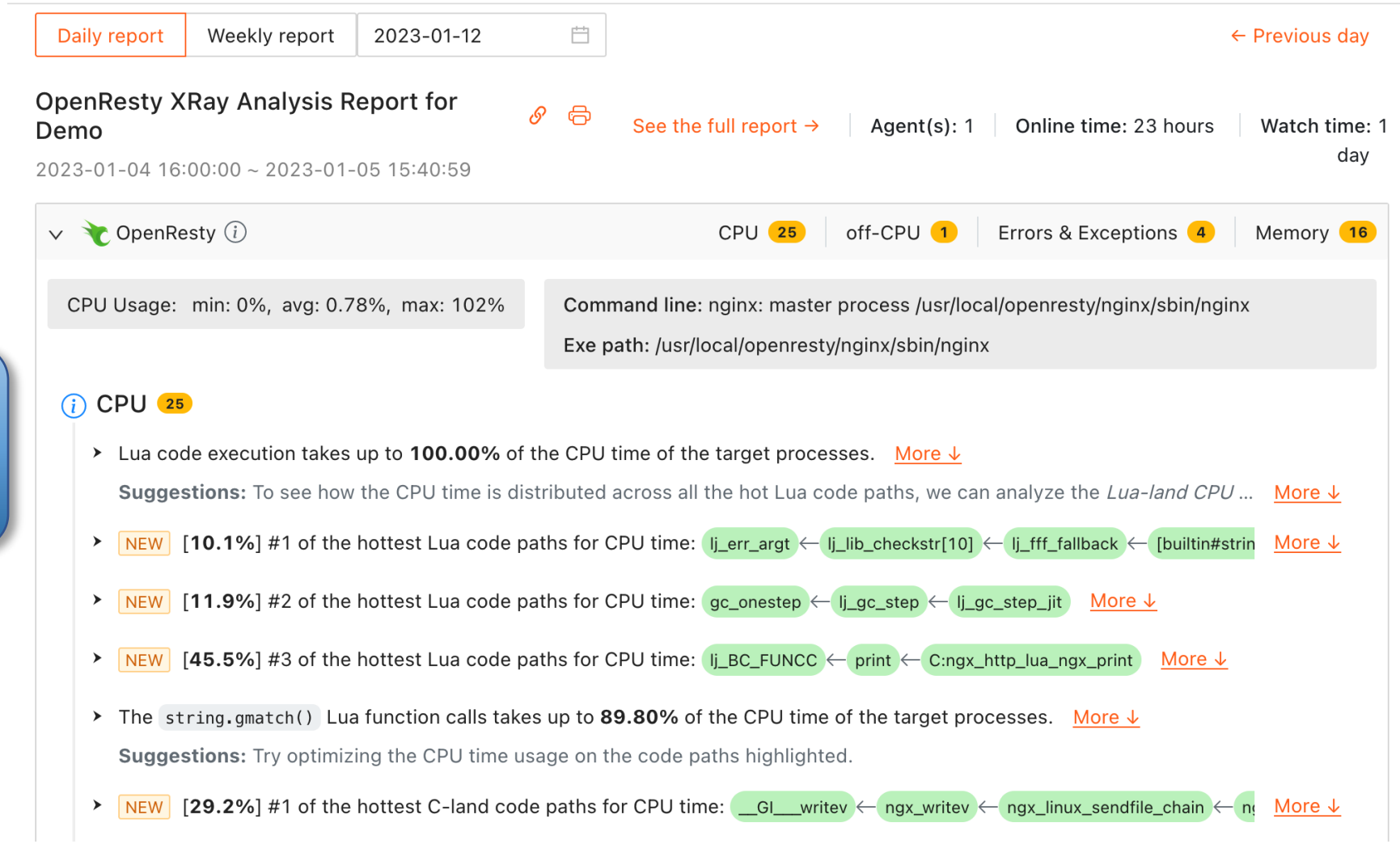


Online intelligent network packet capture

Only catch packets on abnormal network connections

OpenResty XRay Automated Analysis Diagnostic Reports

Read blog posts about automated analysis and diagnostic reports



Automatic Memory Problem Diagnostic Reports

Memory 15

- NEW Glibc allocator takes up to **72.68 MB** in a target process. More ↓
- NEW Glibc arena takes up to **72.68 MB** in a target process. More ↓
- NEW In-use total memory in glibc arena takes up to **67.06 MB** in a target process. More ↓
- NEW Reserved free memory in glibc arena takes up to **5.67 MB** in a target process. More ↓
- NEW Free memory reserved by the LuaJIT allocator takes up to **12.19 MB** in a target process. More ↓
- NEW In-use total memory by the LuaJIT allocator takes up to **4.26 MB** in a target process. More ↓
- NEW Lua GC size of all types takes up to **1.45 MB** in a target process. More ↓
- NEW [**10.7%**] #1 of the hottest reference paths for LuaJIT GC object: table 0x7f950c0a0828 (584B 524.40KB) ← [light userdat More ↓
- NEW [**12.4%**] #2 of the hottest reference paths for LuaJIT GC object: trace 0x7f950a93a8e8 (2.36KB 161.13KB) ← next side ← More ↓

Automatic Latency Analysis and Diagnosis

Latency 5

↑ 22.2% [**100%**] #1 of the hottest Lua code paths for Newly Created CoSocket: C:ngx_http_lua_socket_tcp_connect ← connect ← C:ngx_http_lua_socket_tcp_connect ← check_peer ← spawn_checker ← check_peers ← pcall ← [builtin#pcall]

See Job 4418885005 for more details.

Collapse ↑

[**100%**] #2 of the hottest Lua code paths for Newly Created CoSocket: C:ngx_http_lua_socket_tcp_connect ← connect ← C:ngx_http_lua_socket_tcp_connect ← _request ← request_admin ← pcall ← [builtin#pcall] More ↓

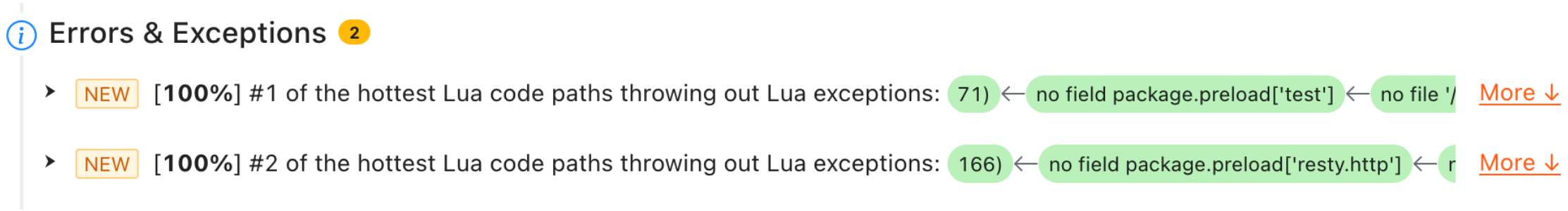
↑ 105.67 ms [**106.97 ms**] #1 of the hottest Lua code paths for Request Yield Latency: lua_yield ← lj_BC_FUNCC ← ngx_sleep ← C:ngx_http_lua_ngx_sleep ← limit_req_rate ← helper_1 ← xpcall ← [builtin#xpcall] ← run_rewrite_phase ← access ← access_by_lua(nginx.conf:586) More ↓

↑ 4.00 ms [**16.00 ms**] #2 of the hottest Lua code paths for Request Yield Latency: lua_yield ← ngx_stream_lua_socket_tcp_receive ← lj_BC_FUNCC ← receive ← C:ngx_stream_lua_socket_tcp_receive ← go ← content_by_lua(nginx.conf:140) More ↓

~~[**1.58 ms**] #2 of the hottest Lua code paths for Request Yield Latency: lua_yield ← lj_BC_FUNCC ← ngx_sleep ← C:ngx_stream_lua_ngx_sleep ← limit_req ← process_req ← go ← content_by_lua(nginx.conf:132)~~ - More ↓

off-CPU Automatic Diagnostic Report

Automatic Diagnostic Reports on Errors and Exceptions



Automatic Analysis of Security Issues

- Automatic checking and reporting of connections without TLS encryption
- Dynamically scanning of TLS connections without certificate source verification
- Check the usage of a non-secure version of the SSL protocol
- Scan remote shell command execution events and code contexts

Core Dump Process Remains Analysis (Process Crashes)

All | Core Dump Analysis

Application Type * OpenResty

Analyzer * openresty-core-dump-analysis (OpenResty Core Dump Analysis)

Core File * /tmp/core.3859164

Executable File Path * /usr/local/openresty/nginx/sbin/nginx

▶ Advanced Settings

Core File Meta Data

File Name: /tmp/core.3859164

Executable File Path: /usr/local/openresty/nginx/sbin/nginx

Size: 14.72MB

Analyze

Extract Deep Information from Core Dump Files

Analysis 7173425

(gdb) lbt
C:ngx_md5_body
trace#1:access.lua:4
check_token
@/usr/local/openresty/lualib/access.lua:3
auth
@/usr/local/openresty/lualib/access.lua:21
@access_by_lua(nginx.conf:51):2

(gdb) full_lbt
C:ngx_md5_body
trace#1:access.lua:4
check_token
@/usr/local/openresty/lualib/access.lua:3
auth
@/usr/local/openresty/lualib/access.lua:21
headers = "access"
token = "hello"
@access_by_lua(nginx.conf:51):2

Compiler Output | Analyzer Output | Graphs

(gdb) ngx_process_info
parent: 3859163
process: worker 0

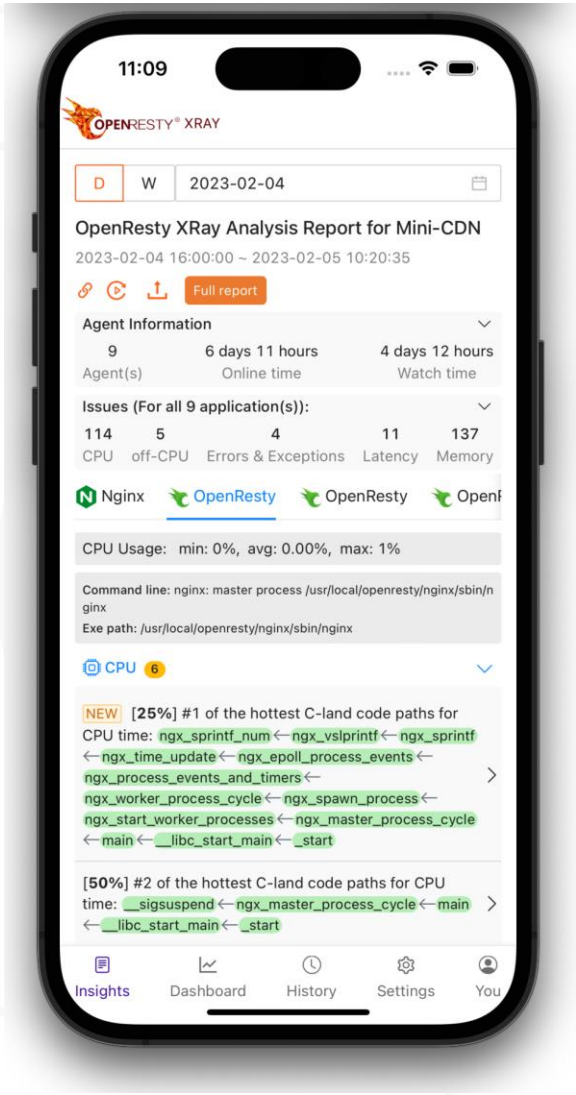
(gdb) cur_http_req
current phase: access
schema: http, req_size: 52, resp_size: 0GET / HTTP/1.1
Host: localhost:80
TOKEN: hello

(gdb) ubt
0x42e958 ngx_md5_body [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/src/core/ngx_md5.c:199]
0x42f1be ngx_md5_final [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/src/core/ngx_md5.c:91]
0x4ea997 ngx_http_lua_ffi_md5 [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/../ngx_lua-0.10.21/s
7f763447ffd3: []
0x4f86a1 ngx_http_lua_run_thread [/usr/src/debug/openresty-1.21.4.1/build/nginx-1.21.4/../ngx_lua-0.10.2

OpenResty XRay Mobile Apps

Watch your applications from any where, any time

- Android (Google Play)
- iOS (Apple Store)



OpenResty XRay is Not just for OpenResty Applications

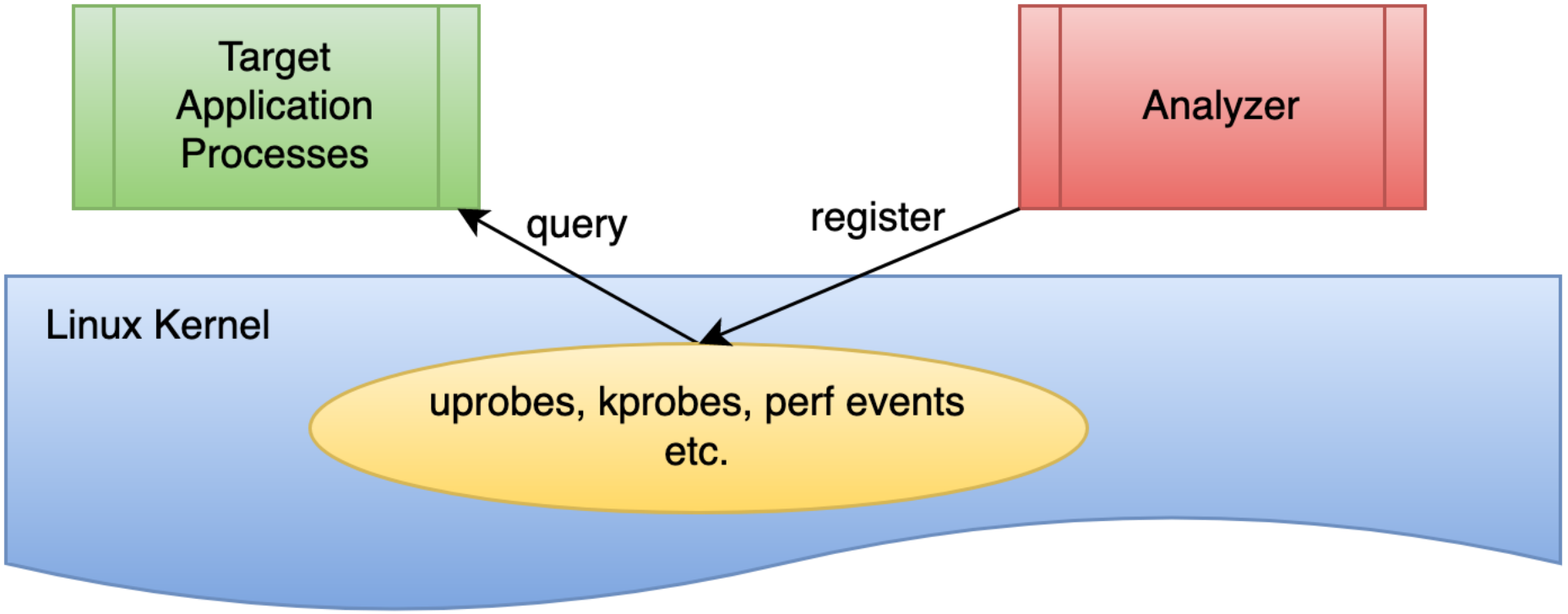
- Nginx, LuaJIT, OpenResty, Python, PHP, Go, Erlang, Perl, Envoy, Ruby, Redis, Rust, Kong
- Preliminary Support:
  PostgreSQL
- Coming soon:
  NodeJS, Java

Supports Most Mainstream Linux Distributions and Container Deployment

Ubuntu

Debian

OpenResty XRay is based On the Advanced Dynamic Tracing Technology



Advantages of Dynamic Tracing

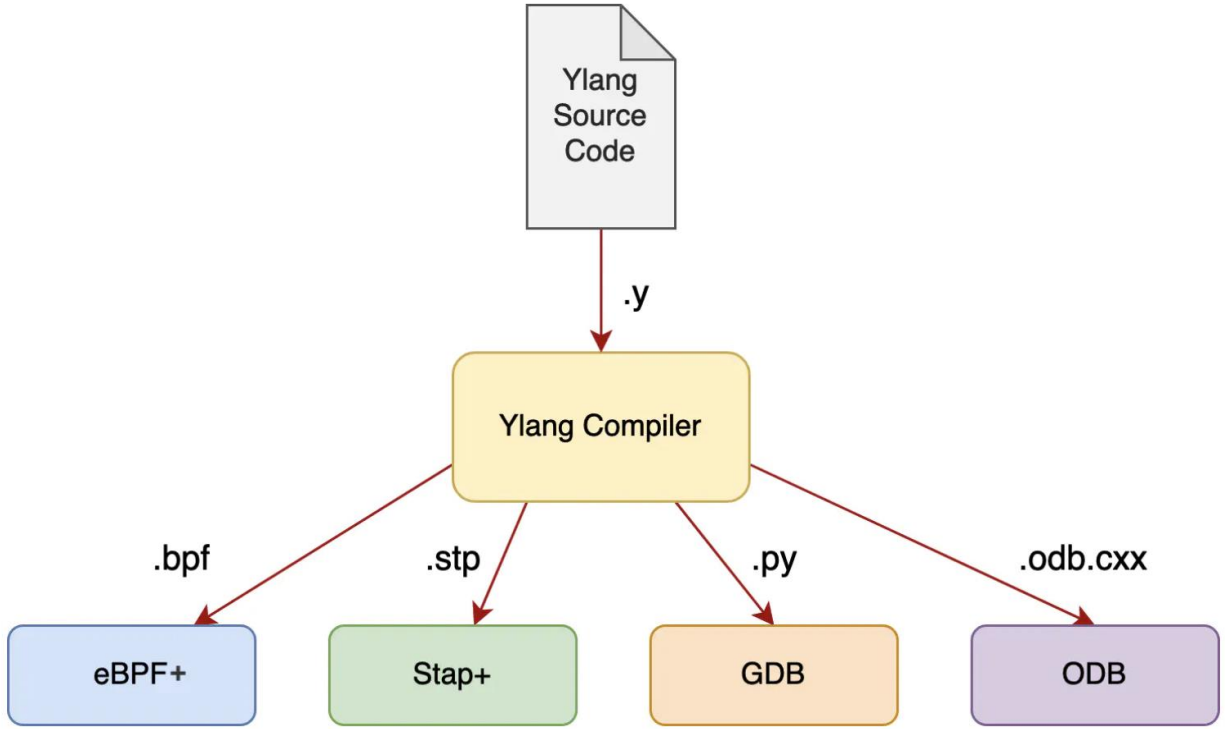
- Non-invasive, no need to modify application modifications
- Hot-plugging, usually do not need the cooperation of applications (many open-source dynamic tracing tools still require the cooperation of applications)
- Overhead is normally low and aggregation can be done at the data source
- Online real-time debug capability in a postmortem manner
- Full technology stack analysis from all angles
- On-demand sampling
- Strictly 0 loss when not sampling

OpenResty XRay New generation of Dynamic Tracing Technology

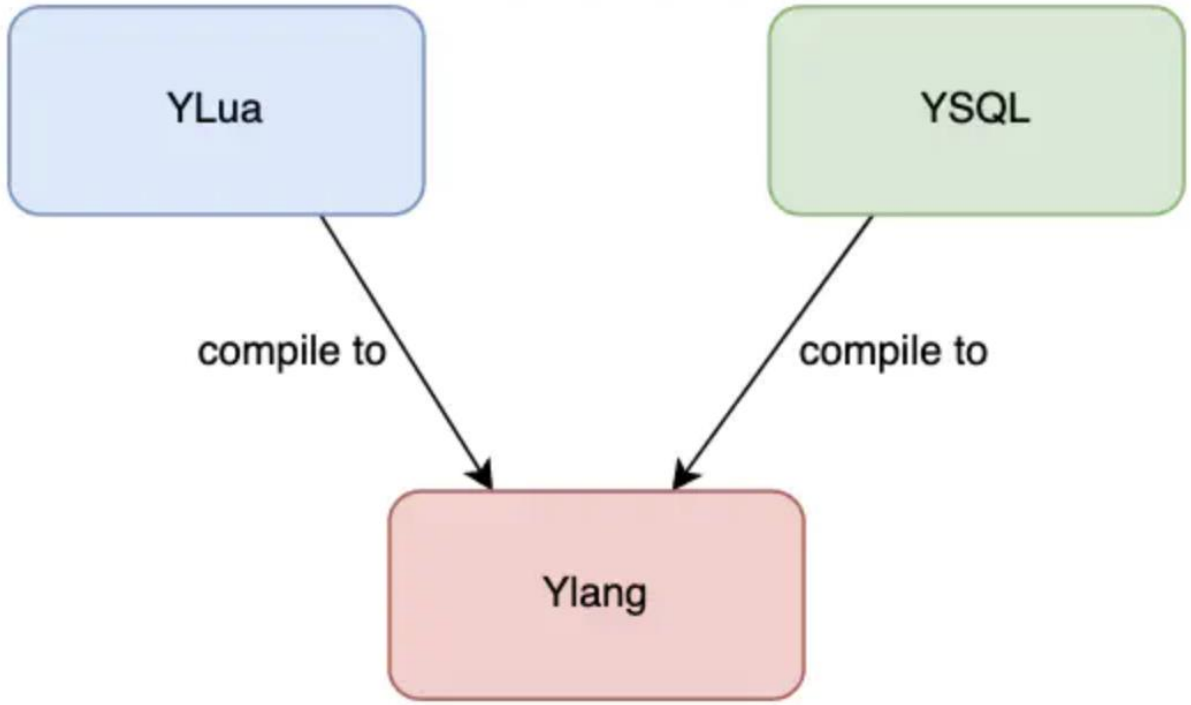
- Y-language (Ylang) compiler (supports most of the syntax of GNU C and standard C)
- Ylua Language
- YSQL Language
- Stap+ has significantly improved SystemTap
- eBPF+ significantly improves eBPF (while LLVM+ also significantly improves open source LLVM)
- ODB is an ultra-lightweight version of GDB
- The Ylang compiler can also generate highly optimized Python extension codes of GDB
- Stringent performance loss control aimed at online production environments

“Write once, run everywhere”

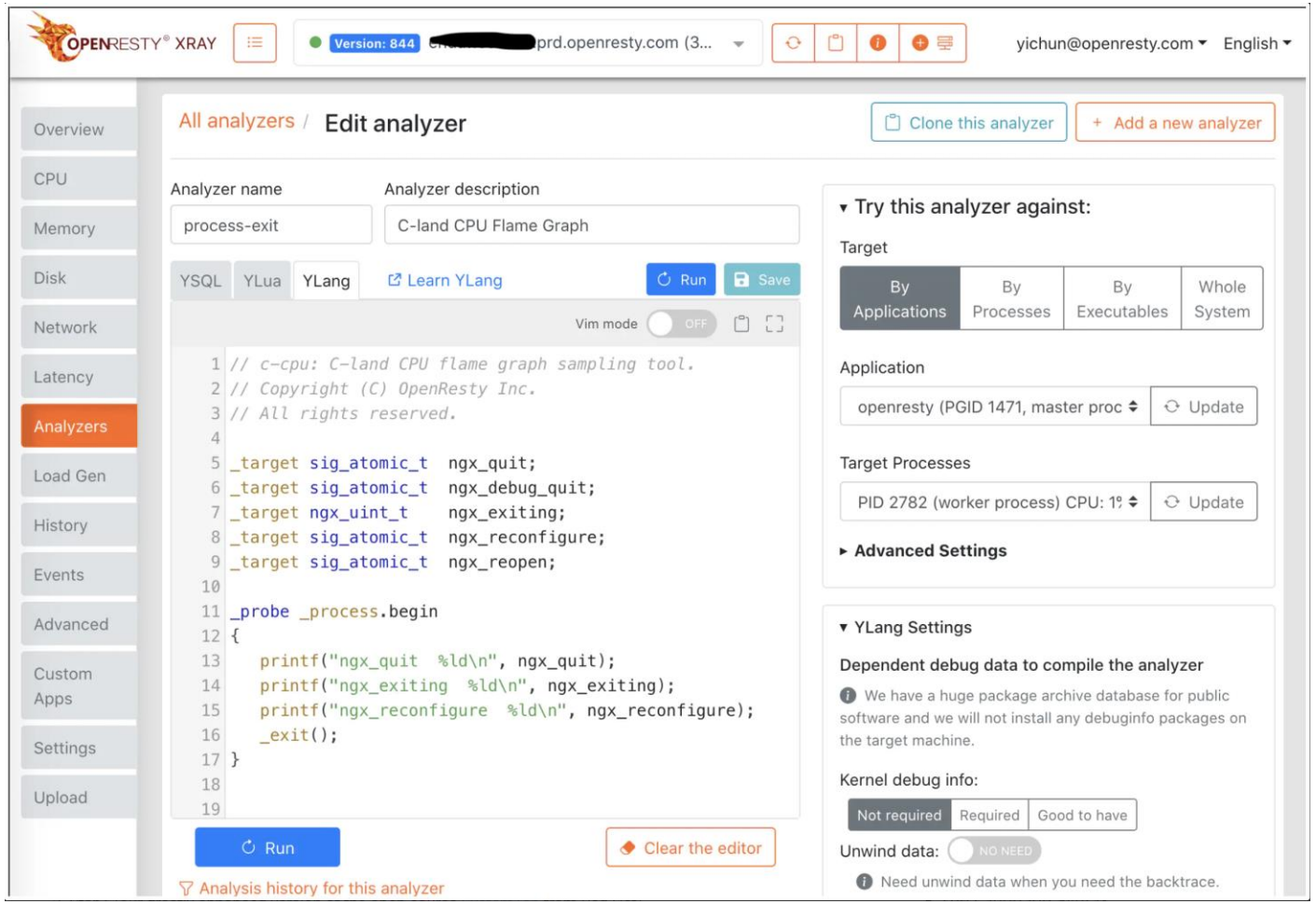
From Y-language code to various runtime codes



More Abstract Languages Which Are Based on Y Language



Write and Debug Analysis Tools Written in Languages like Ylang/YLua/YSQL On the OpenResty Xray Console's Web UI



OpenResty XRay Hundreds of Standard Analyzers

- OpenResty XRay Standard Analyzers
  - c-alloc-fgraph
  - c-count-alloc-free
  - c-memory
  - c-memory-leak-fgraph
  - c-off-cpu
  - c-on-cpu
  - collect-luajit-ffnames
  - cpu-hogs
  - epoll-loop-blocking-distr
  - epoll-sched-latency-distr
  - epoll-wait-ret-distr
  - epoll-wait-timers
  - epoll-wait-timers-fgraph
  - file-system-fgraph
  - func-latency-distr
  - glibc-chunks
  - jemalloc-bins
  - kernel-on-cpu
  - lj-add-timer-lua-fgraph
  - lj-alloc-stats
  - lj-c-memory-leak-fgraph
  - lj-c-off-cpu
  - lj-c-on-cpu
  - lj-config
  - lj-dump-loaded-mods
  - lj-err-mem
  - lj-excep-lua-fgraph
  - lj-free-stats
  - lj-gc-step-calls
  - lj-lua-exception
  - lj-gco-ref
  - lj-gco-stat
  - lj-lua-err-msg
  - lj-lua-new-timer-errors
  - lj-lua-newcdata
  - lj-lua-newfunc
  - lj-lua-newgco
  - ngx-add-timer-event-fgraph
  - lj-trace-stats
  - ngx-add-timer-event-timer-distr
  - lj-vm-states
  - mmap-leaks
  - musl-libc-chunks
  - ngx-access-log-buffer-size
  - ngx-config
  - ngx-config-servers
  - ngx-cpu-hottest-hosts
  - ngx-cpu-hottest-uris
  - ngx-downstream-keepalive-stats
  - ngx-dump-req
  - ngx-dump-timers
  - ngx-epoll-wait-timers
  - ngx-err-log-lvl-distr

Debug symbols

- OpenResty XRay has a central package database indexing hundred TB of debug symbols for public packages. This database is still growing rapidly
- The target machine does not need to install or store debug symbols, as long as they have been indexed by the OpenResty XRay Central Package Database
- For applications in which debug symbols cannot be found or were not generated at compile time, OpenResty XRay will be able to automatically rebuild debug symbols (prototype of working machine learning algorithms already exist)

Trusted By Lots of Enterprise Customers

Learn More

OpenResty XRay Frequently Asked Questions

Visit OpenResty Xray official blog

Visit OpenResty XRay official website

Try OpenResty XRay for free