Snabb

fast software networking,
made simple

Max Rottenkolber <max@mr.gy>

whoami

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

Open source hacker, working on Snabb since 2014

Consulting on networking in userspace, protocols, software optimization, etc...

Snabb?

Software toolkit for high-performance networking applications (in userspace/kernel-bypass mode)

Runs on generic x86_64 Linux servers

Simplicity above all (simple programs are fast, fast programs can be simple)

Agenda

1. Why Snabb? Why networking in userspace?
2. How does Snabb work?
3. How is Snabb used?
4. How do you use snabb? (Tutorial)

Why?

Want to deploy an RFC/feature that no vendor is selling?

Used to mean “tough luck!”

Today, you have open source and commodity servers.

Build it in software! (Usual benefits apply: quicker, cheaper iteration cycles, ...)

Software? Userspace?

Userspace data planes

Bypass kernel (handle data path in userspace)

Snabb (also: DPDK/VPP)

“Kernel, give me PCI, CPU cores, DMA hugepages.
Thanks, bye!”

Fast data path?

Tell kernel to forget about the NIC

Map device registers to memory (poke to initialize)

Allocate descriptors in DMA memory for I/O with the device

Enter polling/busy loop to read/write on the device (receive/transmit descriptor ring buffers)

Fast applications?

Application logic sits *in* the busy loop (no context switches)

Has access to full packets, can modify them in-place, hands off mutated packets to transmit queue.

It's "just" software: can be written in any language
(C, Lua, Rust, Assembly, ...)

How fast?

Limited by PCI bandwidth,
CPU frequency * core count (parallelization possible)

~10-50 Gbps/core

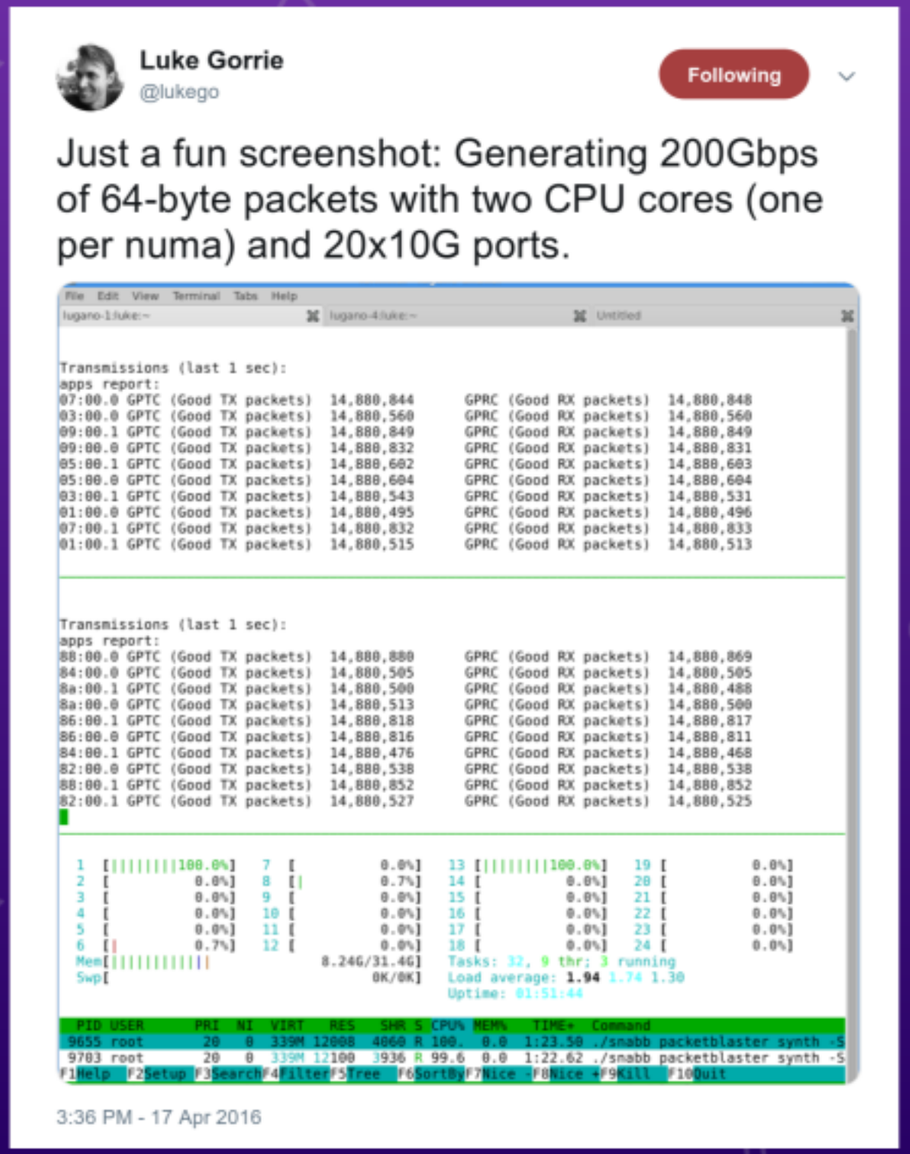
~1-5 Mpps/core

(depends on workload/application)

Examples?

This selection is an incomplete, deliberate sampling.

packetblaster



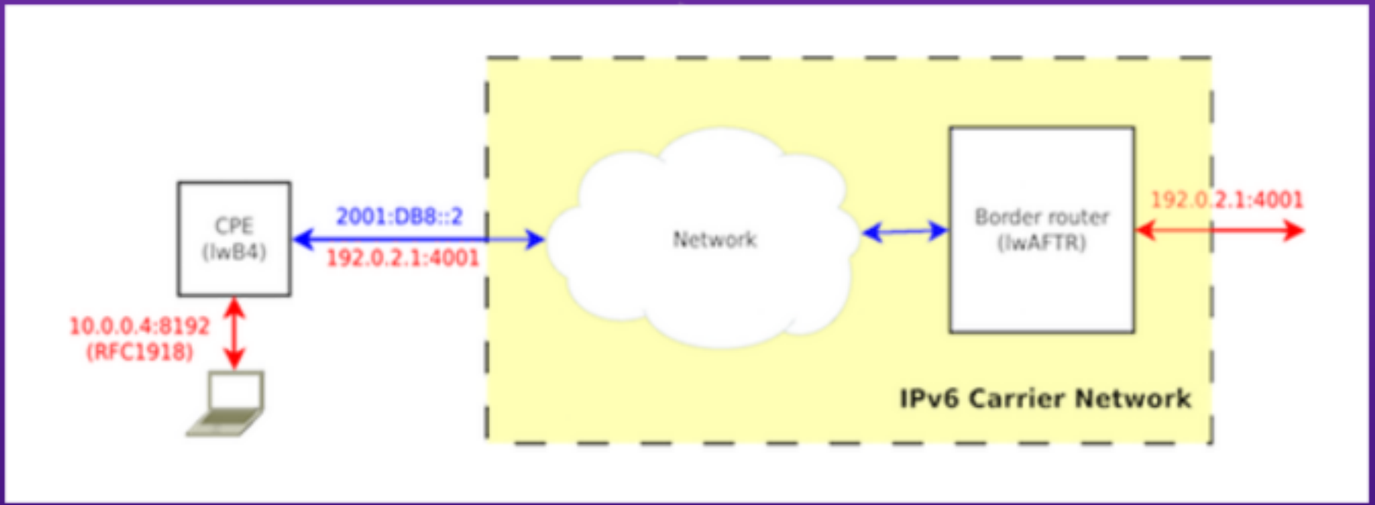
Low-overhead load generator

```
$ snabb packetblaster \
        replay foo.pcap 82:00.1
```

The trick:
pre-fill NIC transmit descriptors traffic
reset queue cursor in a loop, no I/O
required.

lwaftr

Lightweight 4-over-6 AFTR
(border router tunnel endpoint)
Developed by Igalia



Processes all IPv4 traffic for a country-wide ISP network

See K. Zorbadelos (OTE) at RIPE76:
https://ripe76.ripe.net/archives/video/30/

Traffic analysis at $CDN

(Imagine a picture of a server rack filled with one hundred 10g NIC ports)

Network engineers query 1 Tbps of live traffic...

...by writing ad-hoc Snabb scripts (in Lua)

Vita 

High-throughput IPsec/ESP gateway
(prevent wire-tapping on links you do not control)

Encrypts 10-40G L3 paths at line rate

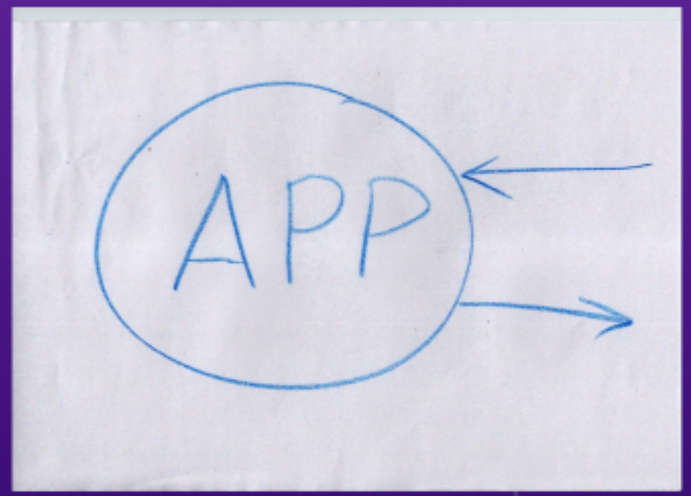
See “High-Performance Traffic Encryption...” @ RIPE78
(https://ripe78.ripe.net/archives/video/65/)

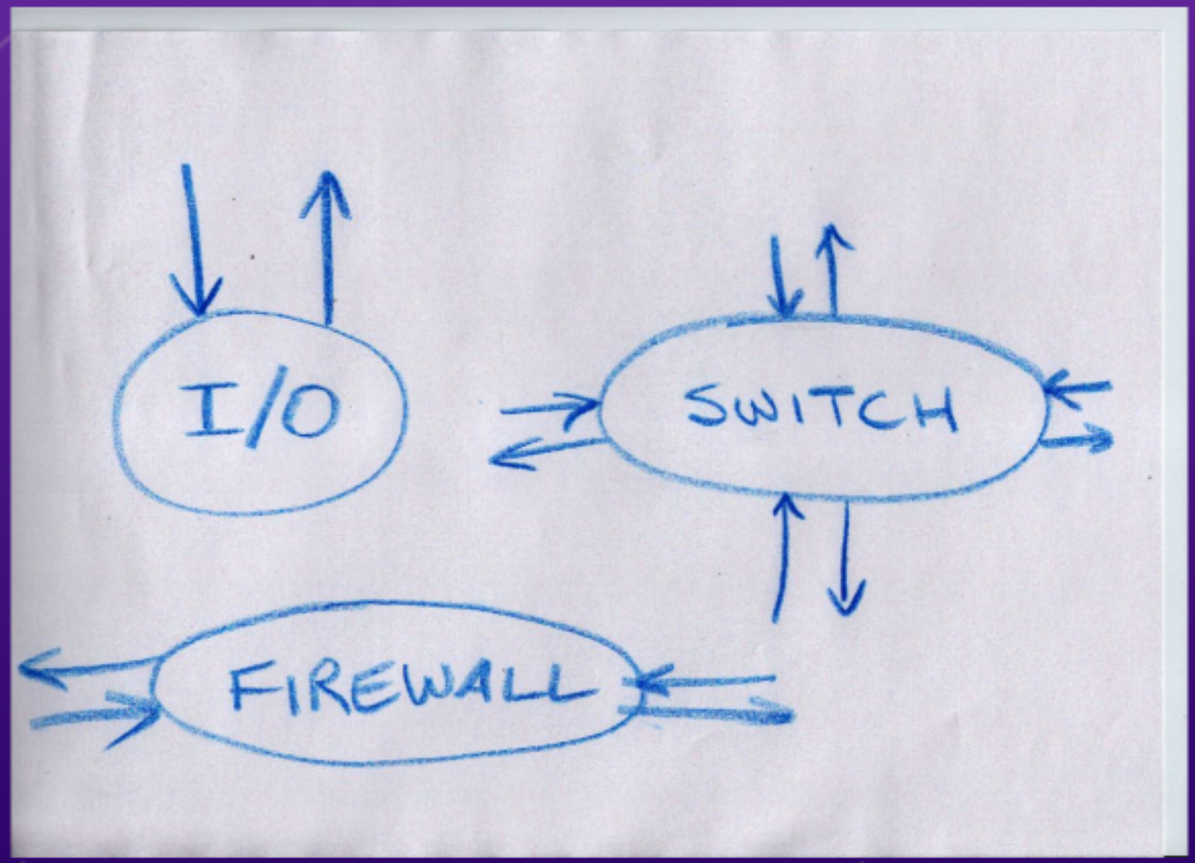
Open Source?

Reminder: this is all open source software, and it runs on off-the-shelf commodity server hardware!

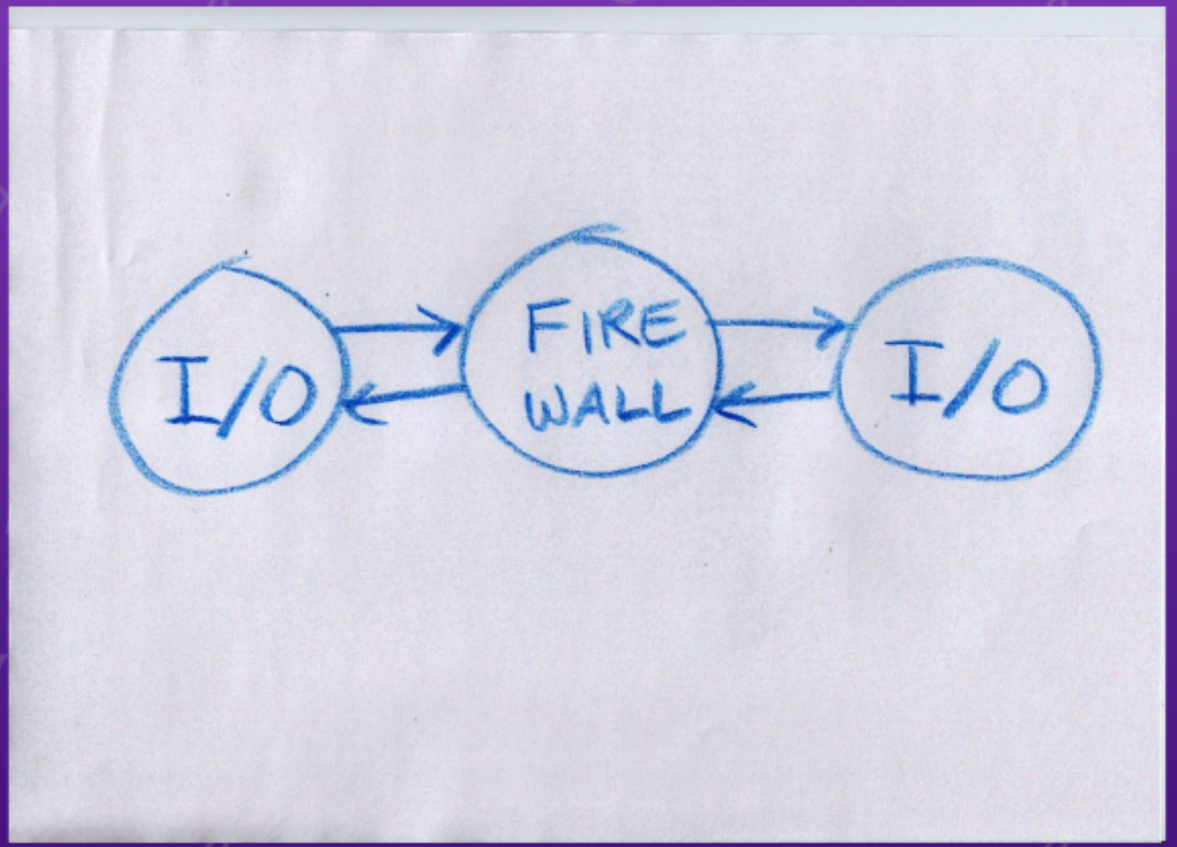
Snabb in a Nutshell

(https://github.com/lukego/blog/issues/10)

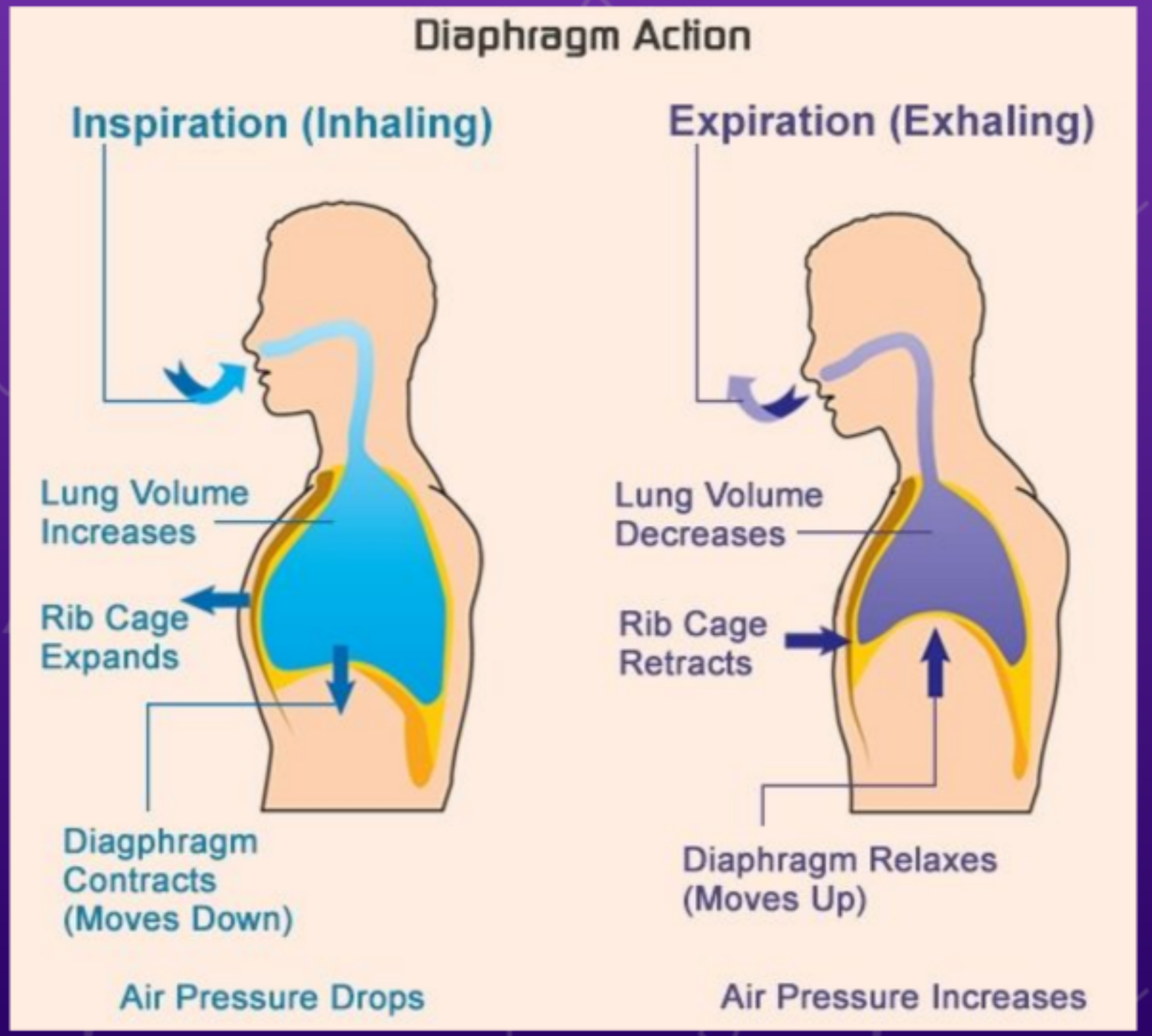




```
local c = config.new()

config.app(c, "IO", Intel82599,
           {pci="07:01.1"})

config.app(c, "FW", PcapFilter,
           "tcp or udp or icmp")

config.link(c, "IO.tx -> FW.input")
config.link(c, "FW.output ->
IO.rx")

engine.configure(c)
engine.main()
```



Metaphor: inhale/exhale



Apps: callbacks

```
App:new(arg)  -- initialize instance

App:push()    -- push packets through the network

App:pull()    -- pull packets into the network

(...plus some more)
```

Apps: initialization

```
Firewall = {}

function Firewall:new (rules)
   local o = {filter=compile(rules)}
   return setmetatable(o, {__index=Firewall})
end
```

Apps: data flow

```
function Firewall:push ()
   local input = self.input.input        -- Input link.
   local output = self.output.output     -- Output link.

   while not link.empty(input) do        -- Can receive?
      local p = link.receive(input)      -- Get next packet.

      if self.filter(p) then             -- Packet matches rule?
         link.transmit(output, p)        -- Yes? Forward.
      else
         packet.free(p)                  -- No? Drop.
      end
   end
end
```

Apps: data flow 2

```
-- Pull in packets from the network and queue them on our 'tx' link.

function Intel82599:pull ()
   local l = self.output.tx
   ...
   for i = 1, engine.pull_npackets do
      if not self.dev:can_receive() then break end
      transmit(l, self.dev:receive())
   end
   ...
end
```

Wait, what? Lua?

Snabb itself is written in Lua

The easiest way to write Snabb applications is by using Lua.

Yes, it goes super fast. Like C.

Note: the packet/link ABI is stable and nothing prevents you from writing apps in other languages.

A fast JIT compiler

We use the blazing fast RaptorJIT compiler (a fork of LuaJIT)

What's a link?

```
struct link {
   struct packet *packets[LINK_RING_SIZE];
   int read;  // next element to be read
   int write; // next element to be written
   ... // stats counters omitted
};
```

What's in a packet?

```
struct packet {
   uint16_t length;
   unsigned char data[PACKET_PAYLOAD_SIZE];
};
```

Poking a packet

(it's raw bits)

```
IP_OFS = 14
IP_PROTO_OFS = 9

function is_tcp(p)
    local ip = p.data+IP_OFS
    return ip[IP_PROTO_OFS] == 0x06
end
```

Poking a packet

(you can use structs, too)

```
eth = ffi.typeof[[
   struct {
     uint8_t dst[6], src[6];
     uint16_t type;
   } __attribute__((packed))
]]

eth_ptr = ffi.typeof("$ *",
eth)
```

```
ip6 = ffi.typeof[[
   struct {
     uint32_t v_tc_fl;
     uint16_t payload_length;
     uint8_t next_header, hop_limit;
     uint8_t src_ip[16], dst_ip[16];
   } __attribute__((packed))
]]

ip6_ptr = ffi.typeof("$ *", ip6)
```

Poking a packet

(with struct casting)

```
function is_tcp6 (p)
   local eth = ffi.cast(eth_ptr, p.data)
   local ip = ffi.cast(ip6_ptr, p.data+ffi.sizeof(eth))
   return ntohs(eth.type) == 0x86dd
      and ip.next_header == 0x06
end
```

Poking a packet

(using the protocol library)

```
local eth = require("lib.protocol.ethernet"):new{}
local ip6 = require("lib.protocol.ipv6"):new{}

function is_tcp6 (p)
   eth:new_from_mem(p.data)
   ip6:new_from_mem(p.data+eth:sizeof())
   return (eth:type() == 0x86dd)
      and (ipv6:next_header() == 0x06)
end
```

We got libs

...for many protocols, LPM, checksums, PMU, token bucket, fast hash tables, logging, YANG, and many more!

Might be that what you need is already included.

We got apps

Basic topology apps (Split, Join, ...), L2 bridge, NIC drivers, IPFIX, IPsec, ARP, ND, ICMP, IP (de)fragmentation, packet filtering, Pcap source/sink, rate limiting, RAW socket, TAP sockets, Virtio, ...

Re-use encouraged!

Getting Started

```
$ git clone https://github.com/snabbco/snabb

$ cd snabb && make -j && cd src

$ cp -r program/example_replay program/myhack
<start hacking>

$ make -j

$ sudo ./snabb myhack
```

Questions?

Engage with us!
https://github.com/snabbco/snabb

Docs!
https://snabbco.github.io

Bonus slide

So you need to go faster than C?

Generate machine code using DynASM!

```
local function auth12_equal(Dst)
    | mov rax, [arg1]
    | mov edx, [arg1 + 8]
    | xor rax, [arg2]
    | xor edx, [arg2 + 8]
    | or rax, rdx
    | ret
end
```