We're making this a reality

#include <os>

From bootloader to REST API with the new C++

Let's put some operating system
inside your ELF binary

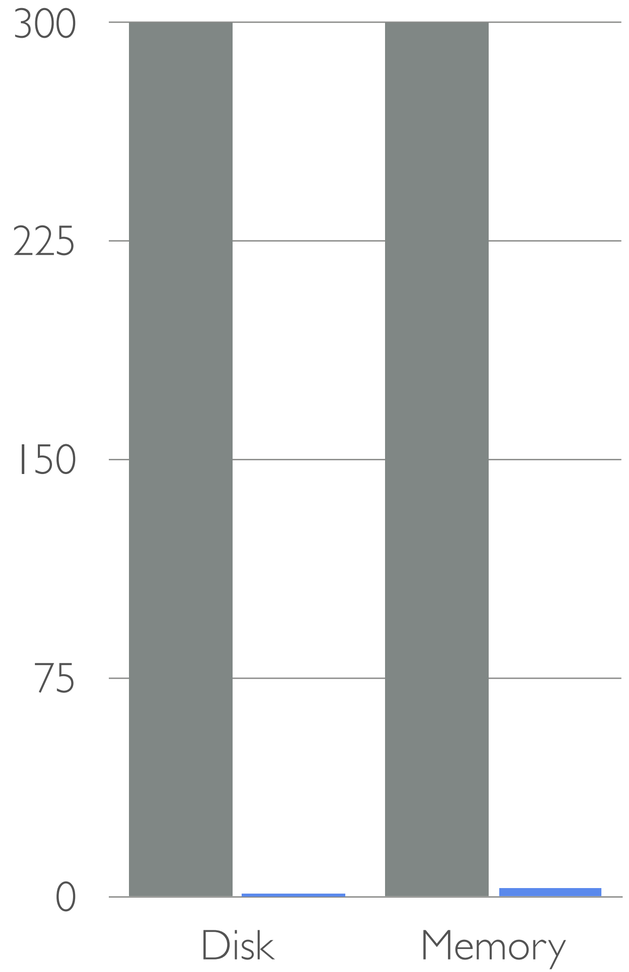
Unikernels 101

- Coined in 2013 by Madhavapeddy, Mortier et. al.[1]
- Single purpose: lots of servers are anyway
- Library operating system: Link in what you need
- Turns a binary into a bootable disk image
- Mainly targeted at virtual machines and IoT

IncludeOS: A minimal, resource efficient implementation from scratch in modern C++ [3].

includeOS v.s. ubuntu

- 300x smaller disk image
- 100x less memory usage by default
- That much reduction in attack surface



v.s.

- IncludeOS is single threaded by default
- Faster, more efficient for single core VM's:
  - 5-20% less CPU cycles spent for the same binary and workload [3]
  - IncludeOS boots in ~300ms. Ubuntu boots in minutes.
- Ubuntu is awesome. As a general purpose OS.

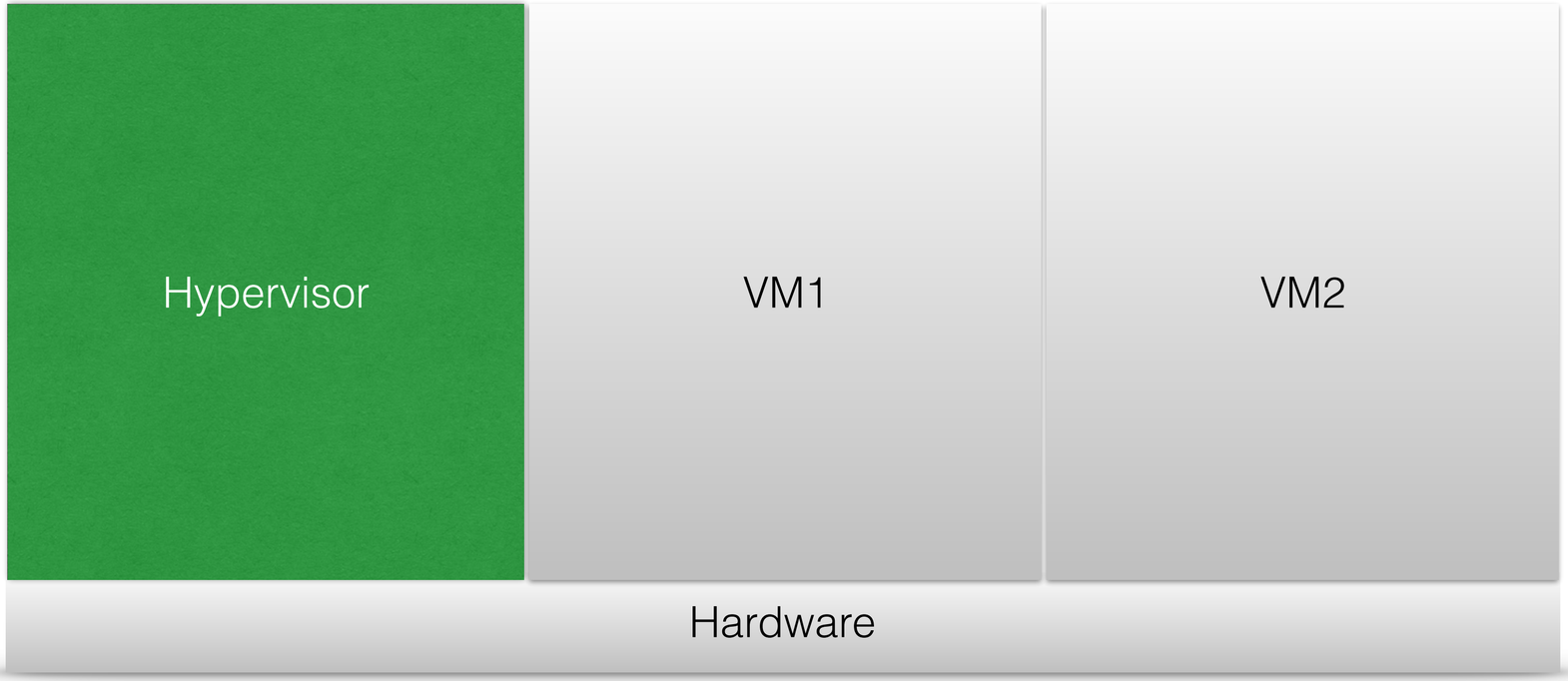
You all know what virtual machines are.
...let's go through it anyway

Popek / Goldberg 1974

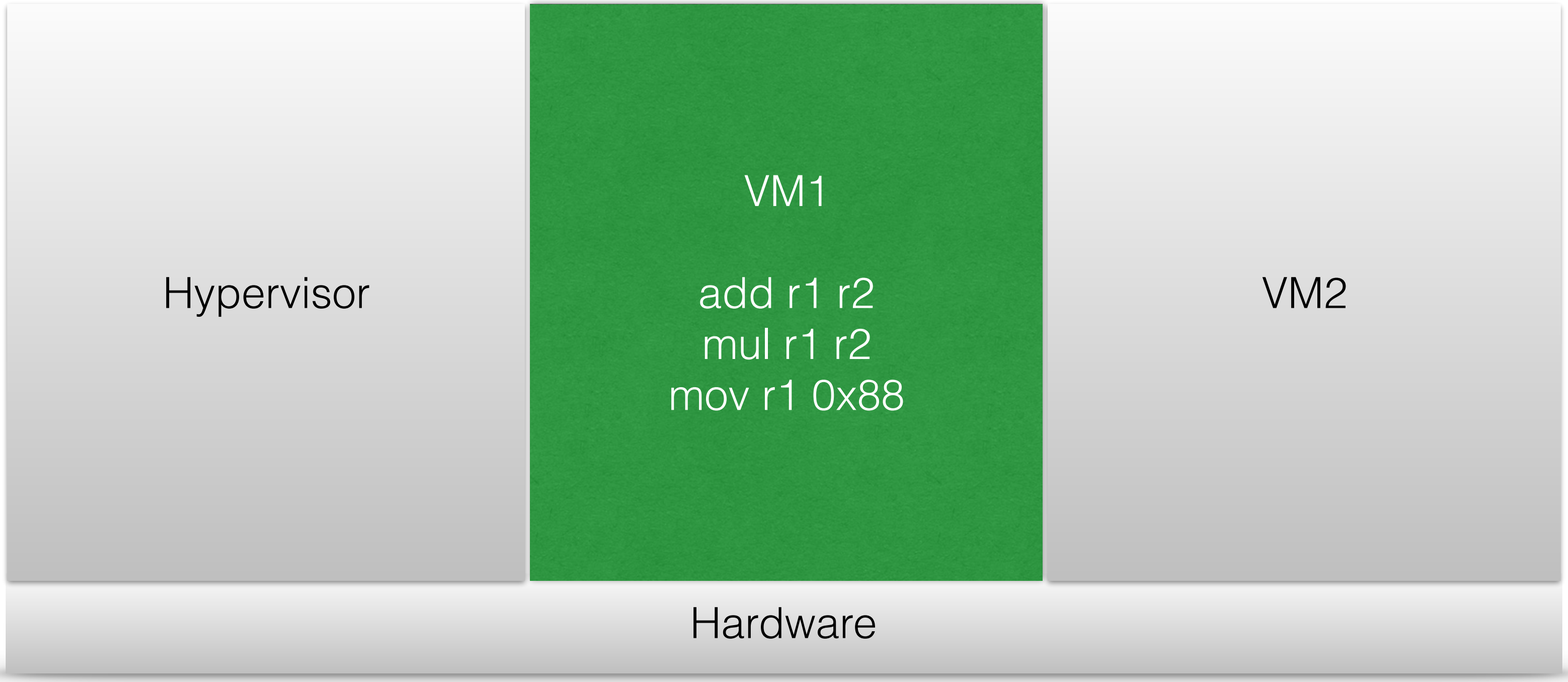
Gerald J. Popek, Robert P. Goldberg CACM [4]

«A virtual machine is taken to be an efficient, isolated duplicate of the real machine»

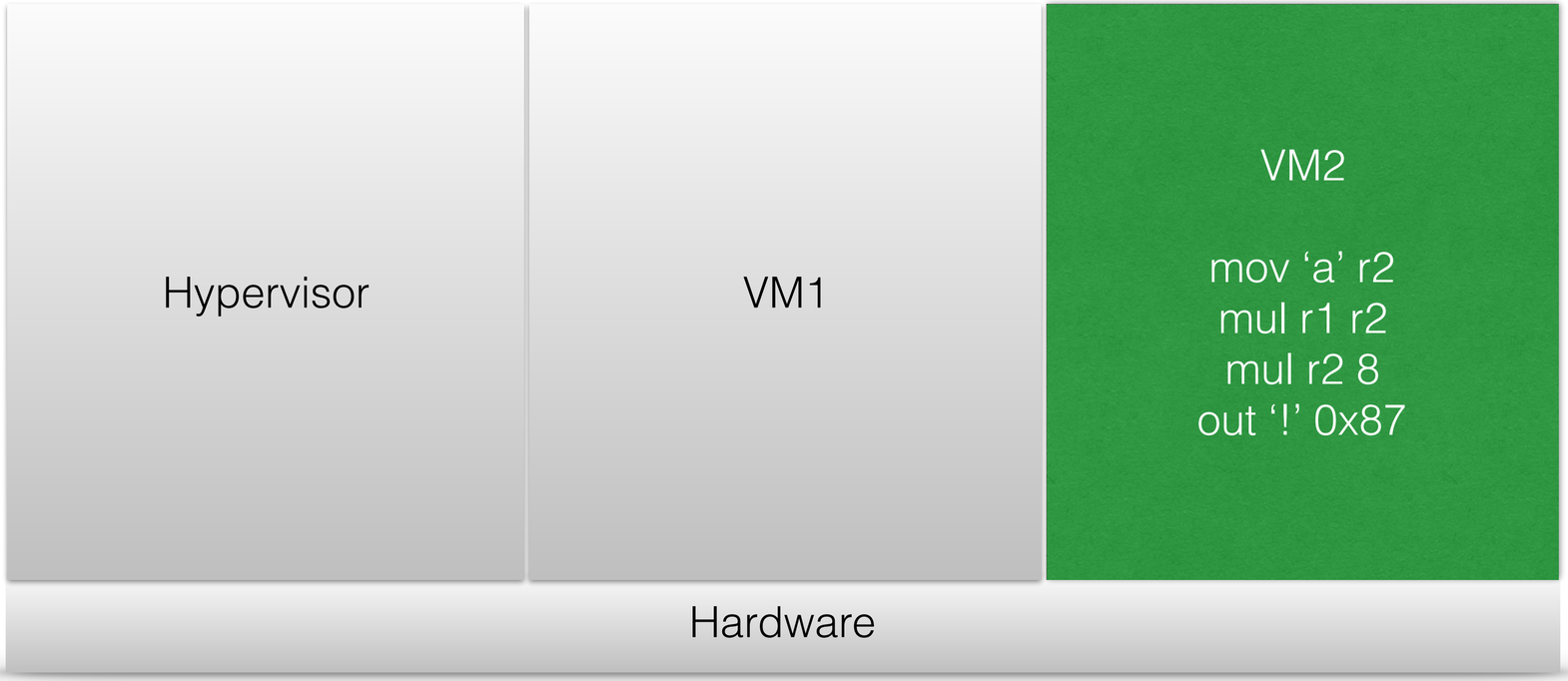
All sensitive instructions have to trap!



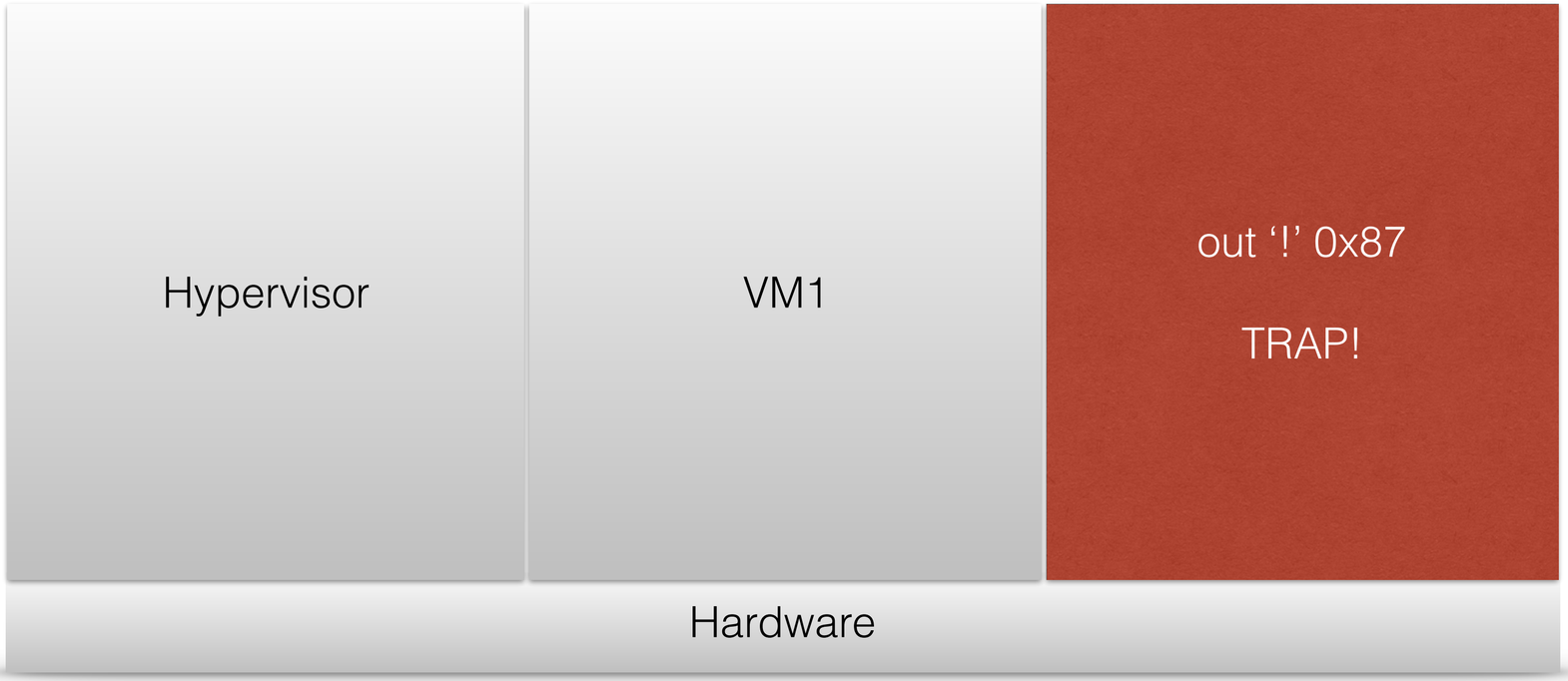
All sensitive instructions have to trap!



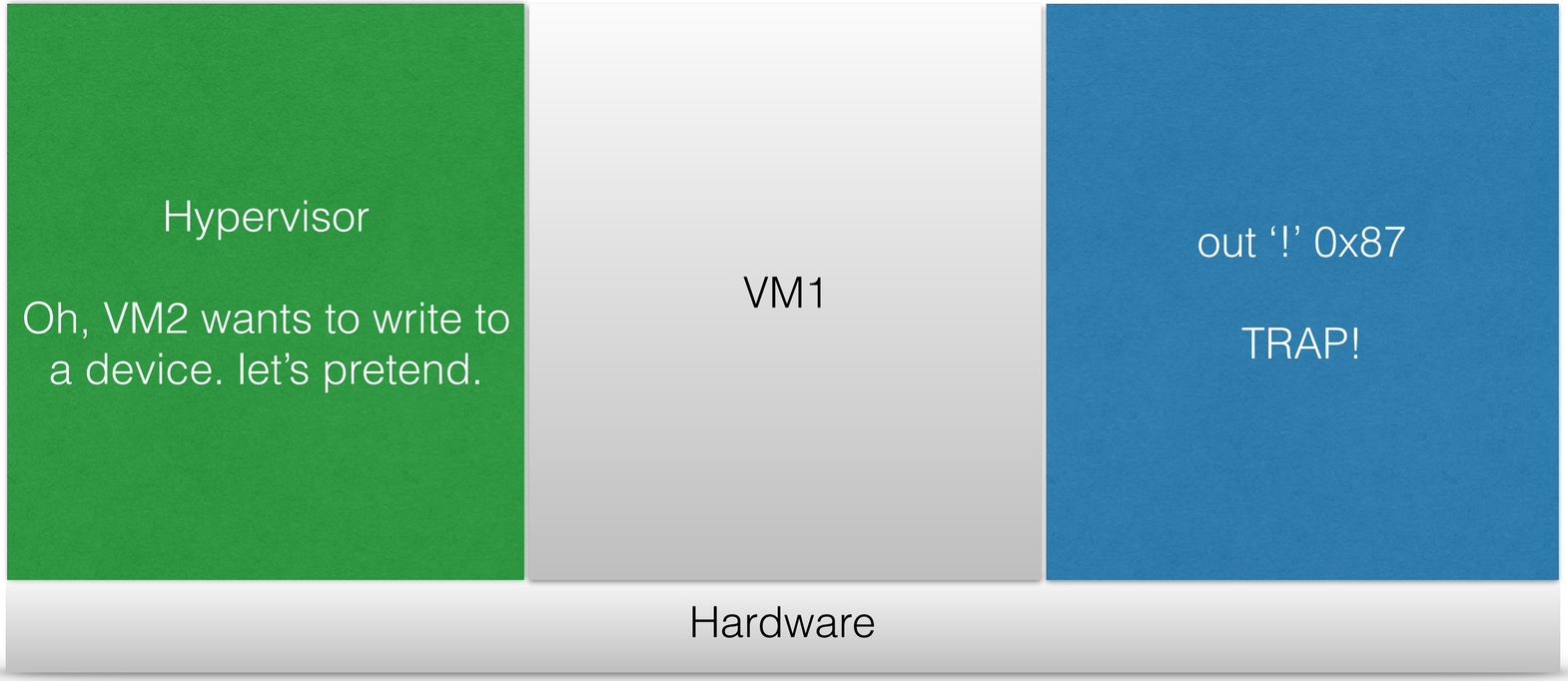
All sensitive instructions have to trap!



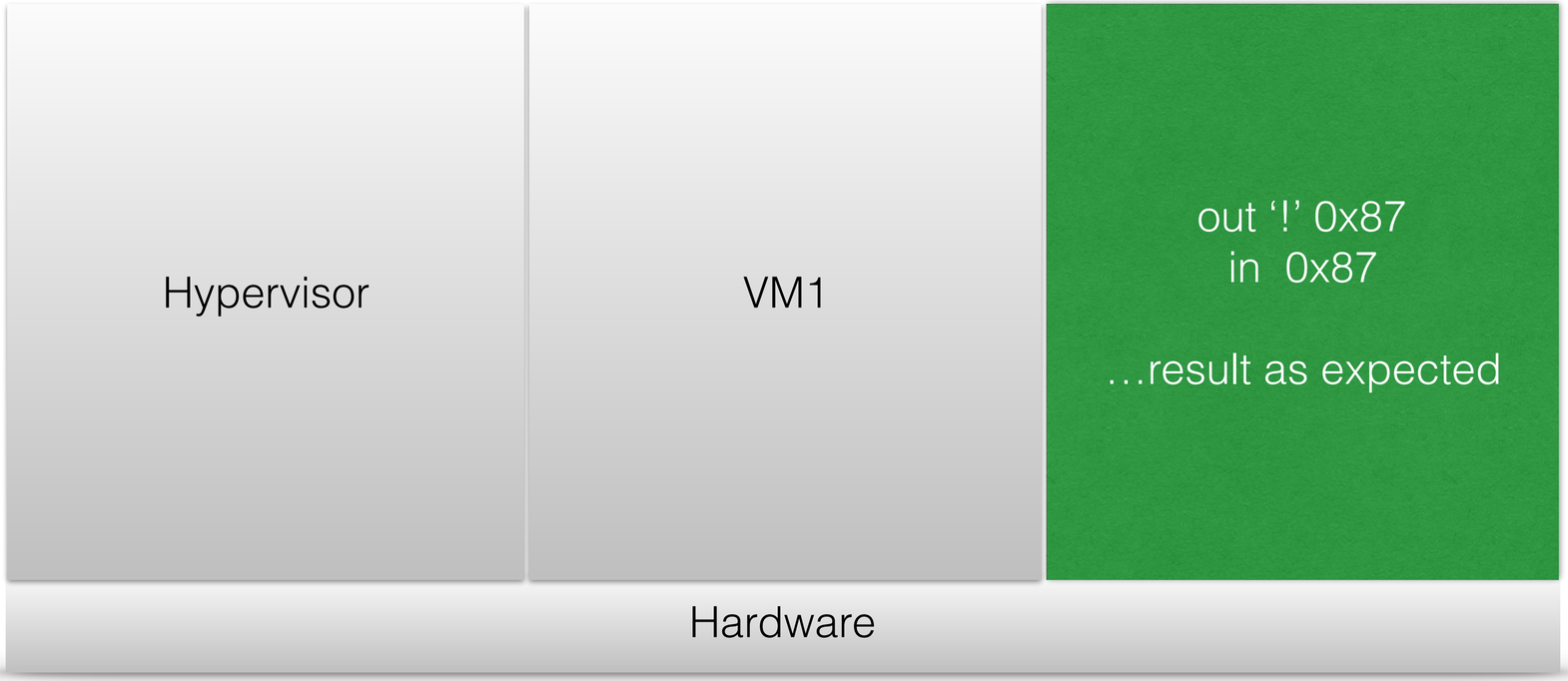
All sensitive instructions have to trap!



All sensitive instructions have to trap!

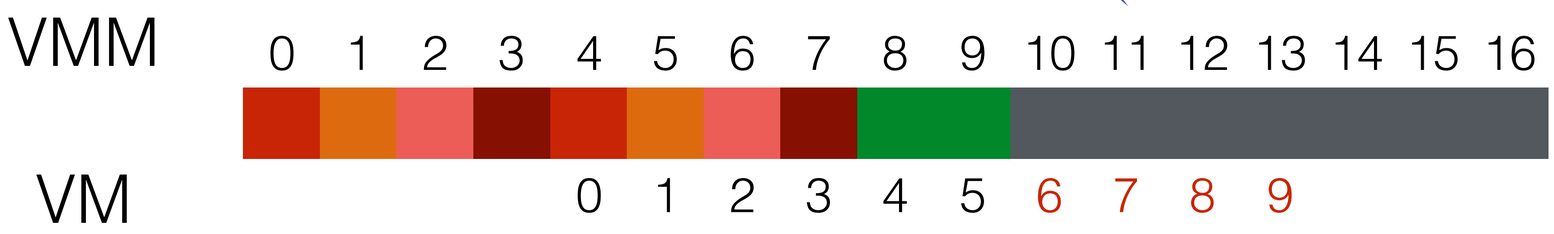


All sensitive instructions have to trap!



Virtual memory

If an address translation results in a physical address out of VM bounds: TRAP!



30 years later

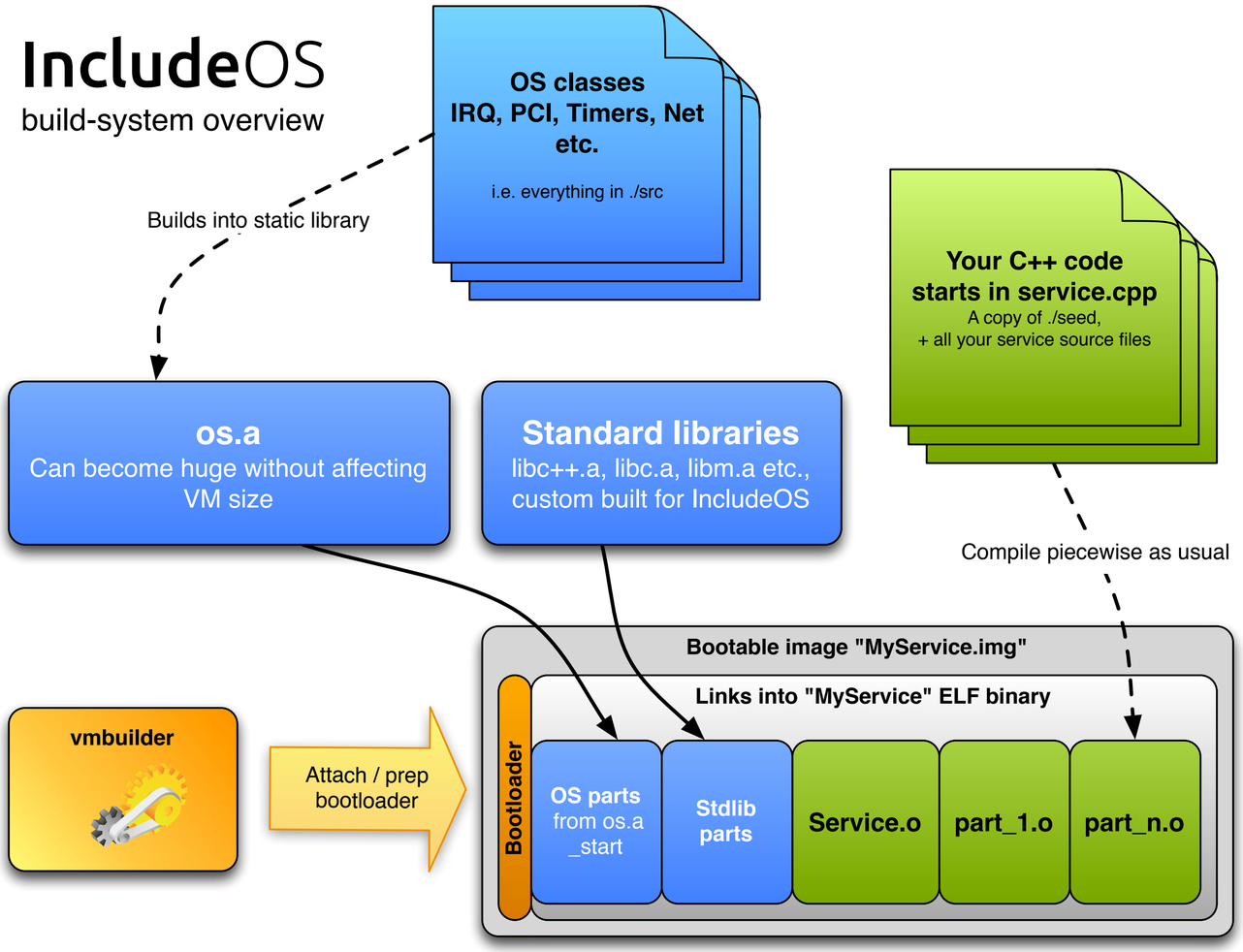
Oh, good idea.

You run directly on hardware.
The VM-trap is just a glorified context switch.

Since we’re 100% self contained
we can run on Linux, Windows, Mac unchanged

It’s like a process.
It just happens to need some operating system.

- os.a: static library containing all OS parts
  - Drivers are now separated
- Linker pulls in what the program needs
- OS library can bloat without affecting image size
- vmbuild: Attach and modify bootloader



Why C++?

(Yes, people really ask that)

C++ is the most evolved

“independent systems language”

- You need to write arbitrary bytes to arbitrary addresses
- You need access to the full instruction set
- You can’t write an OS in a 100% type safe language
- But new C++ gets you the best of both worlds

Hello World!

“kernel” params with multiboot

What about “main”?

```
#include <os>

void Service::start(const std::string&) {
  std::cout << "Hello world - OS included!\n";
}
```

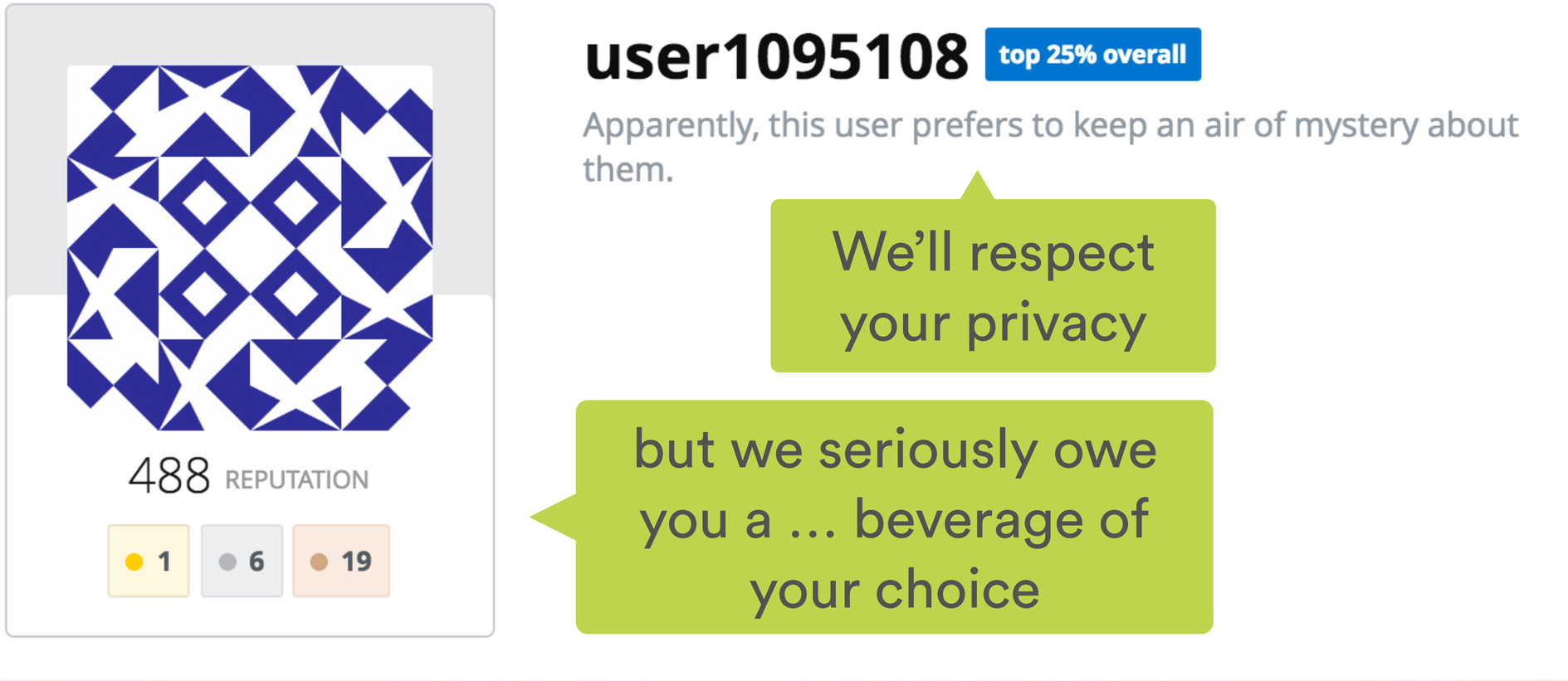
And where does stdout go?

No parent process - where do you return?

Delegates everywhere!

A C++11 version of Don Clugstons
"Fastest possible C++ Delegates" was found
… on Stack Exchange

http://codereview.stackexchange.com/questions/14730/impossibly-fast-delegate-in-c11



http://codereview.stackexchange.com/questions/14730/impossibly-fast-delegate-in-c11

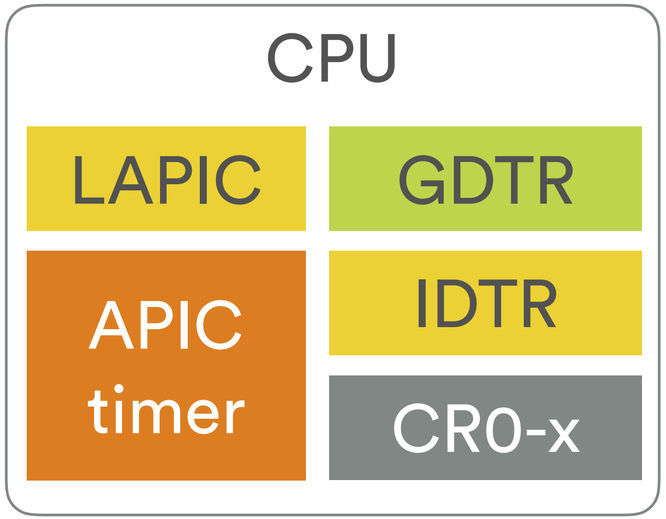
Routing stdout anywhere

```
// VGA
OS::add_stdout(VGA::get_rsprint_handler());

// Custom
OS::add_stdout([] (const std::string& data) {

    // Memory-only (webserver 'logger')
    logger_->log(timestamp() + data);
});
```

In principle: nothing

What's different about an OS for virtual machines?

Hardware overview



PIT Timer

Serial

PIC

PCI Bus

IDE / SCSI

VirtIO Net

VirtIO Block

VirtIO xyz

"Virtual" devices
"Physical" interface

OS doesn't know if they're physical or virtual

How to include only the drivers you need - without having to mention them in code?

Solution: link in self registering drivers

```
// From IncludeOS v0.9, src/drivers/virtionet.cpp
// ...VirtioNet implementation

/** Register VirtioNet's driver factory at the PCI_manager */
struct Autoreg_virtionet {
  Autoreg_virtionet() {
    PCI_manager::register_driver<hw::Nic>(hw::PCI_Device::VEN
                0x1000, &VirtioNet::new_instance);
  }
} autoreg_virtionet;
```

1) Global constructor registers a delegate to driver initialization.

2) OS probes PCI bus and applies any registered driver

Service Makefile:

DRIVERS = virtionet ...

Deployment tools / sysadmins can apply appropriate drivers

Kernel overview

```
// 1. Initialize stack, heap, .bss etc.
// 2. Call global constructors
// 3. Initialize hardware
// 4. Call Service::start()

void OS::event_loop() {

  while (power_) {
    IRQ_manager::get().notify();
  }

  // Cleanup
  Service::stop();
  // ACPI shutdown sequence
  hw::ACPI::shutdown();
}
// From src/kernel/os.cpp
```

Call any delegates subscribing to IRQ's, then "hlt"

Note: All subscribed IRQ's are deferred

Subscribing to interrupts

```
// Actual code from src/drivers/virtionet.cpp

auto del(delegate<void()>
         ::from<VirtioNet,
         &VirtioNet::irq_handler>(this));

IRQ_manager::get().subscribe(10, del);
```

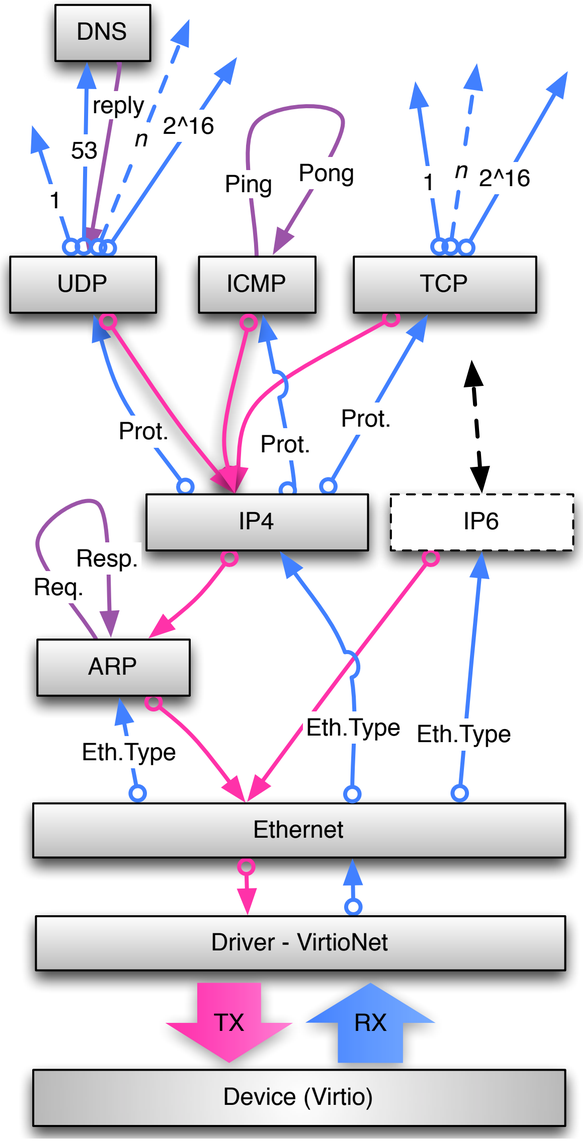
Create delegate to member function of “this”

Subscribe to a given IRQ number called by event loop

For most use cases don’t to worry about IRQ (this is driver code). Abstractions coming up

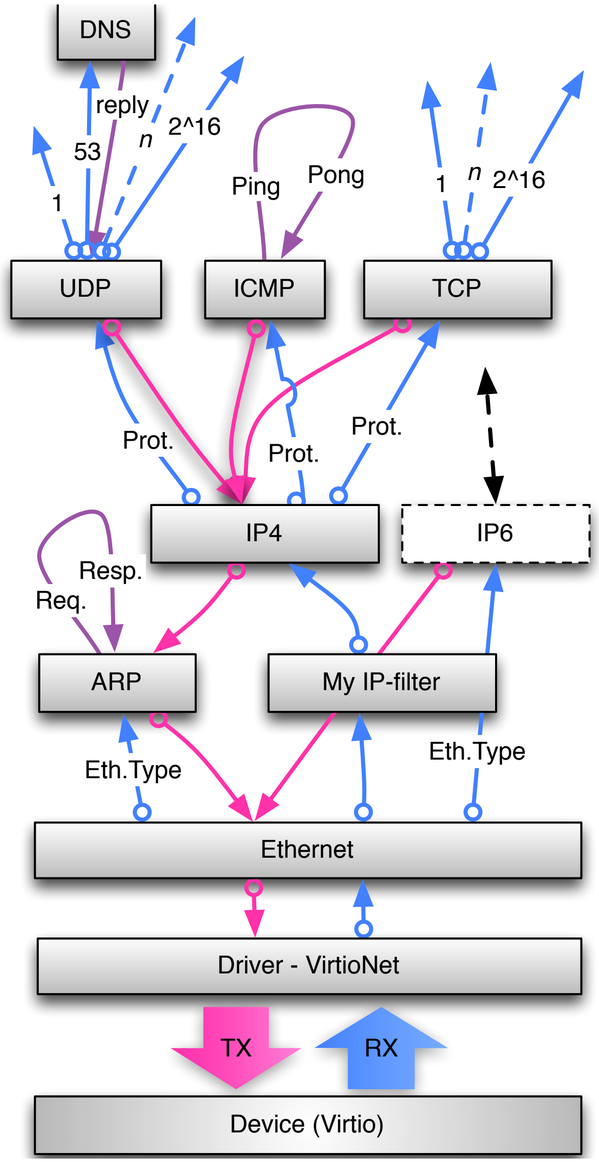
Wiring the IP stack

- Each “box” is a class instance
- All connections are delegates
- The wiring happens in the “Inet” wrapper class
- Rewiring during runtime is easy
  - e.g. for activating and deactivating an IP-filter
  - Can be done from the application



Wiring the IP stack

- Each “box” is a class instance
- All connections are delegates
- The wiring happens in the “Inet” wrapper class
- Rewiring during runtime is easy
  - e.g. for activating and deactivating an IP-filter
  - Can be done from the application



Unikernels:
The default should be minimal

- Single threaded by default: “no overhead principle”
  - In a VM, threading is more expensive [5]
- 100% async: cheapest way to get performance
- No blocking POSIX calls (yet)
  - You’ll get it. It’s your “constitutional right…”

Writing a TCP server

```
#include <os>
#include <net/inet4>

void Service::start(const std::string&) {

  auto& server = net::Inet4::stack().tcp().bind(80);

  server.on_connect([] (auto conn) {
    conn->on_read(1024, [conn] (auto buf, size_t n) {
      conn->write("My first unikernel!\n"s);
    });
  });
}
```

Attach any delegate to TCP events

How do you break in?

- We don’t know
- If you get in, you’re in a hardware protected sandbox
- No shell or other “ad-hoc” code access
- Minimal attack surface: no tools left for the thieves
- Everything inside can be randomized

You made it!

“Leave no room for a lower-level language below C++ (except assembler)”

- Bjarne Stroustrup

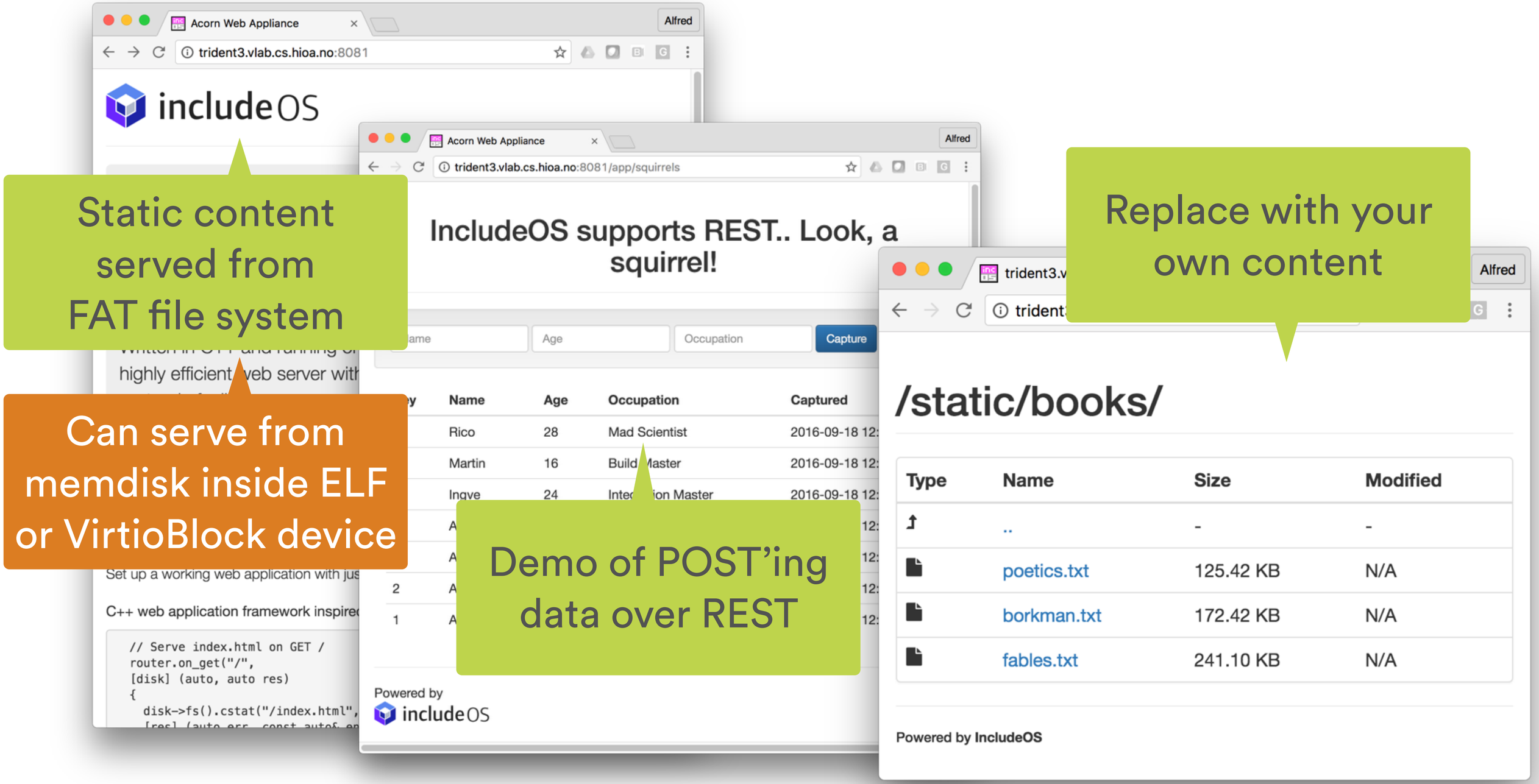
But do we need another language above C++?

The new C++ is awesome for high-level abstractions

Introducing “Acorn”

IncludeOS web server
with C++ framework for REST APIs

https://github.com/includeos/acorn

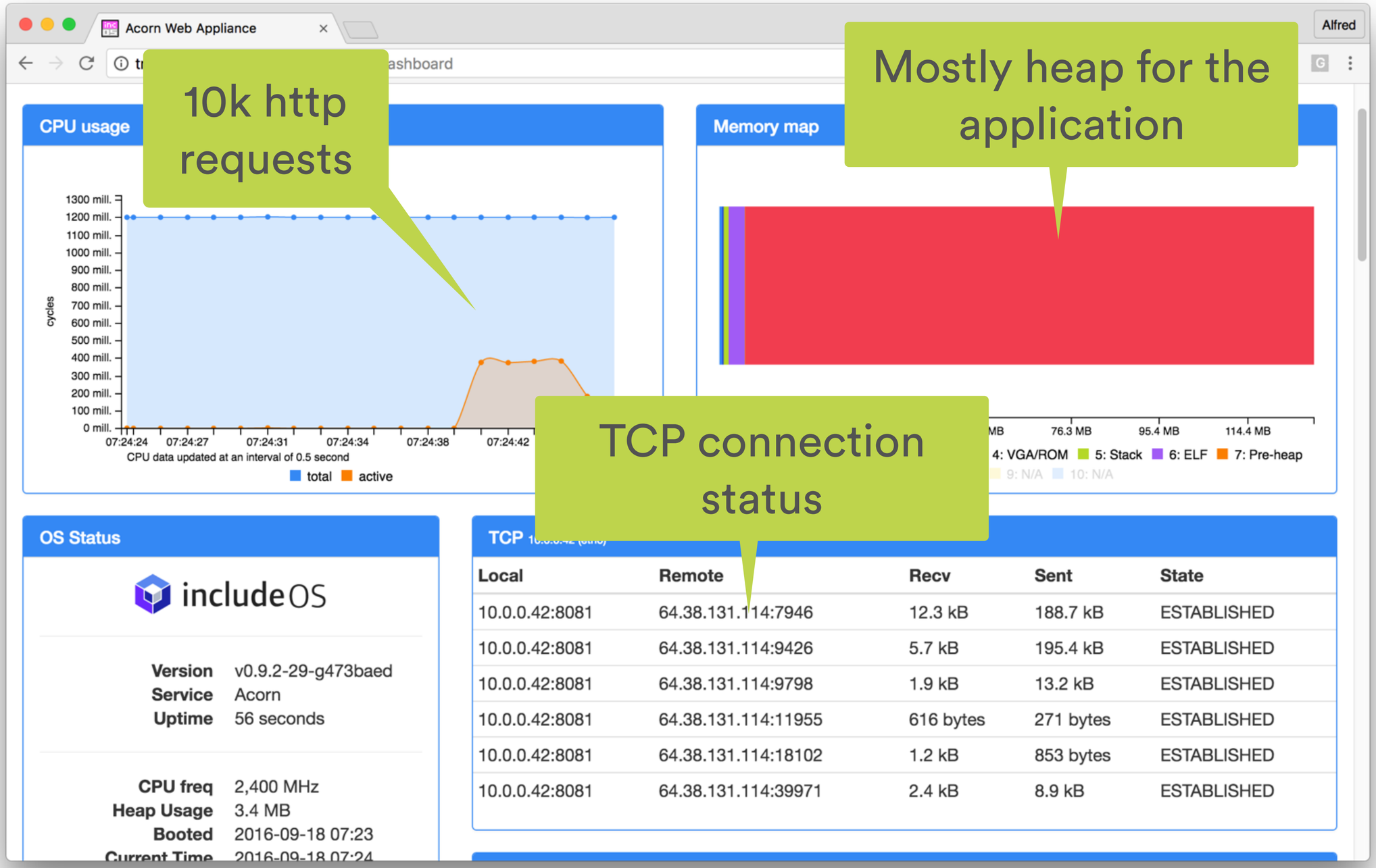

Acorn Web Appliance
Alfred
trident3.vlab.cs.hioa.no:8081
includeOS
Static content served from FAT file system
Can serve from memdisk inside ELF or VirtioBlock device
trident3.vlab.cs.hioa.no:8081/app/squirrels
IncludeOS supports REST.. Look, a squirrel!
Age
Occupation
Capture
Name
Age
Occupation
Captured
Rico
28
Mad Scientist
Martin
16
Demo of POST'ing data over REST
Powered by
includeOS
Replace with your own content
/static/books/
Type
Name
Size
Modified
..
poetics.txt
125.42 KB
N/A
borkman.txt
172.42 KB
N/A
fables.txt
241.10 KB
N/A
Powered by IncludeOS

“C++ is not the language for putting up a simple web page”

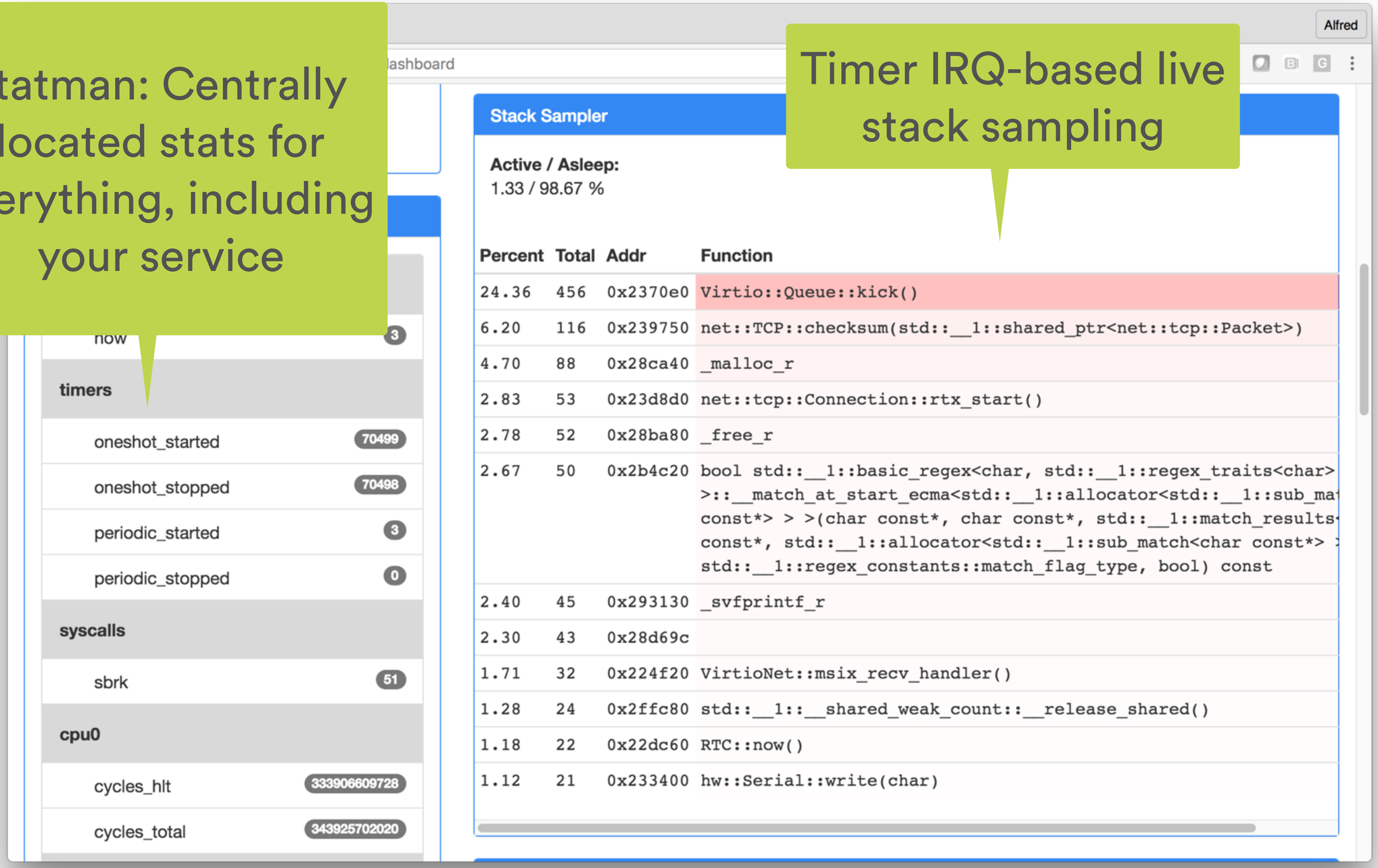
– Bjarne Stroustrup earlier today

How about a fancy web page?

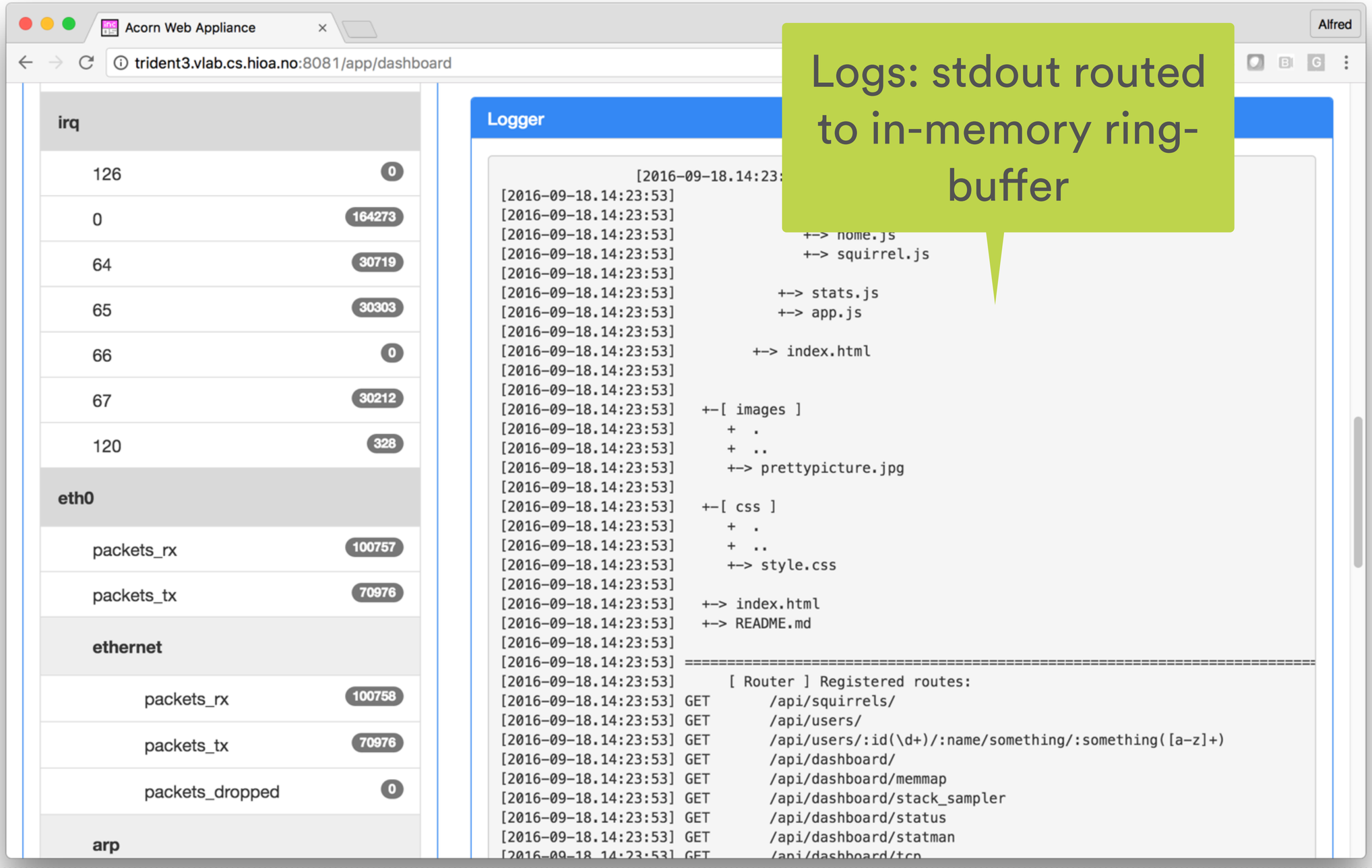
Let’s do a dashboard with live CPU stats and live profiling.
It’s surprisingly easy with C++


Acorn Web Appliance
Alfred
10k http requests
Mostly heap for the application
TCP connection status
CPU usage
1300 mill.
1200 mill.
1100 mill.
1000 mill.
900 mill.
800 mill.
700 mill.
600 mill.
500 mill.
400 mill.
300 mill.
200 mill.
100 mill.
0 mill.
cycles
07:24:24
07:24:27
07:24:31
07:24:34
07:24:38
07:24:42
CPU data updated at an interval of 0.5 second
total
active
Memory map
76.3 MB
95.4 MB
114.4 MB
4: VGA/ROM
5: Stack
6: ELF
7: Pre-heap
9: N/A
10: N/A
OS Status
includeOS
Version v0.9.2-29-g473baed
Service Acorn
Uptime 56 seconds
CPU freq 2,400 MHz
Heap Usage 3.4 MB
Booted 2016-09-18 07:23
Current Time 2016-09-18 07:24
TCP
Local
Remote
Recv
Sent
State
10.0.0.42:8081 64.38.131.114:7946 12.3 kB 188.7 kB ESTABLISHED
10.0.0.42:8081 64.38.131.114:9426 5.7 kB 195.4 kB ESTABLISHED
10.0.0.42:8081 64.38.131.114:9798 1.9 kB 13.2 kB ESTABLISHED
10.0.0.42:8081 64.38.131.114:11955 616 bytes 271 bytes ESTABLISHED
10.0.0.42:8081 64.38.131.114:18102 1.2 kB 853 bytes ESTABLISHED
10.0.0.42:8081 64.38.131.114:39971 2.4 kB 8.9 kB ESTABLISHED

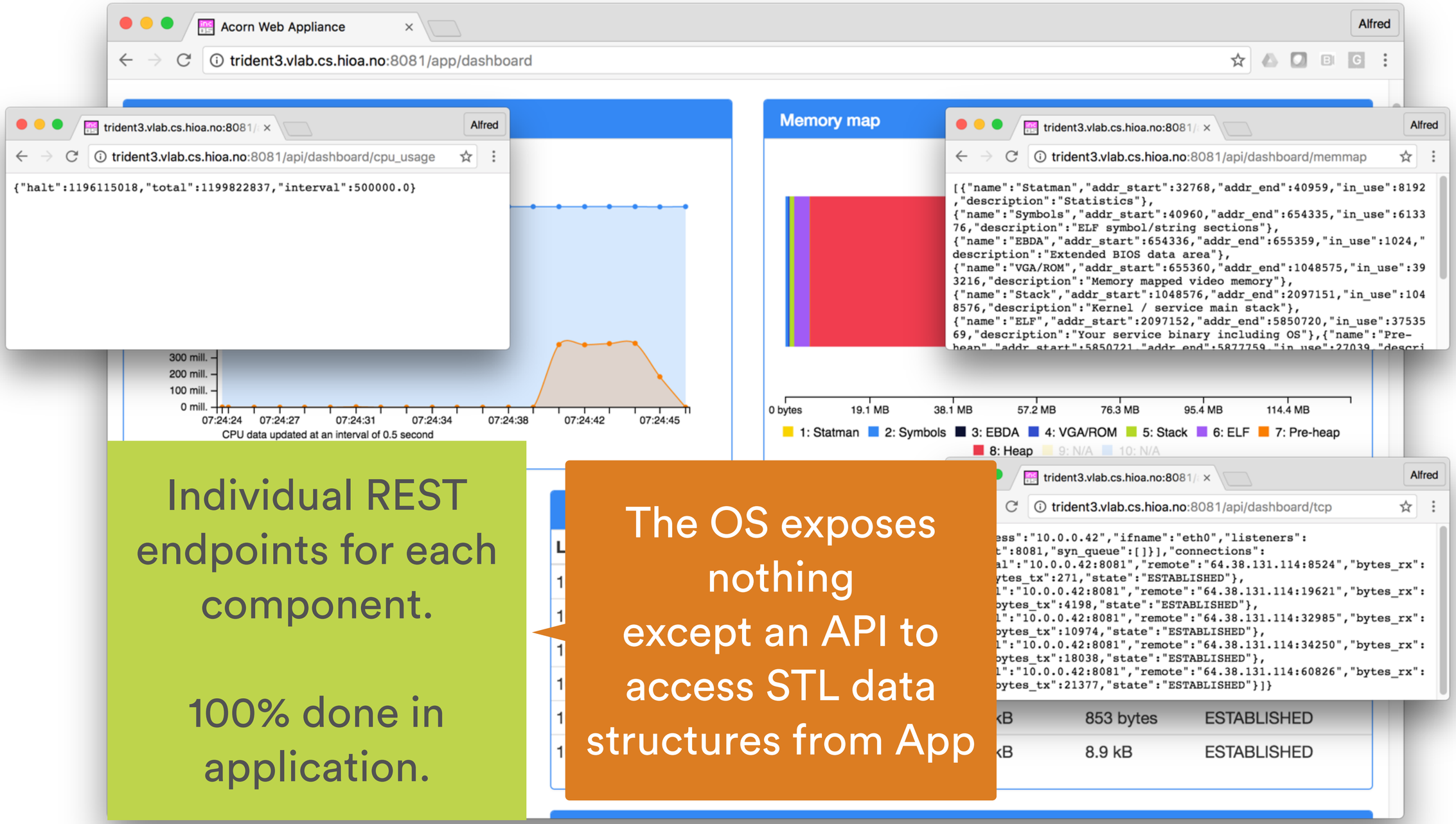
statman: Centrally located stats for everything, including your service



Timer IRQ-based live stack sampling



Logs: stdout routed to in-memory ring-buffer


Acorn Web Appliance
Alfred
trident3.vlab.cs.hioa.no:8081/app/dashboard
trident3.vlab.cs.hioa.no:8081/api/dashboard/cpu_usage
{"halt":1196115018,"total":1199822837,"interval":500000.0}
300 mill.
200 mill.
100 mill.
0 mill.
07:24:24
07:24:27
07:24:31
07:24:34
07:24:38
07:24:42
07:24:45
CPU data updated at an interval of 0.5 second
Memory map
trident3.vlab.cs.hioa.no:8081/api/dashboard/memmap
[{"name":"Statman","addr_start":32768,"addr_end":40959,"in_use":8192
,"description":"Statistics"},
{"name":"Symbols","addr_start":40960,"addr_end":654335,"in_use":6133
76,"description":"ELF symbol/string sections"},
{"name":"EBDA","addr_start":654336,"addr_end":655359,"in_use":1024,"
description":"Extended BIOS data area"},
{"name":"VGA/ROM","addr_start":655360,"addr_end":1048575,"in_use":39
3216,"description":"Memory mapped video memory"},
{"name":"Stack","addr_start":1048576,"addr_end":2097151,"in_use":104
8576,"description":"Kernel / service main stack"},
{"name":"ELF","addr_start":2097152,"addr_end":5850720,"in_use":37535
69,"description":"Your service binary including OS"},{"name":"Pre-
0 bytes
19.1 MB
38.1 MB
57.2 MB
76.3 MB
95.4 MB
114.4 MB
1: Statman
2: Symbols
3: EBDA
4: VGA/ROM
5: Stack
6: ELF
7: Pre-heap
8: Heap
9: N/A
10: N/A
trident3.vlab.cs.hioa.no:8081/api/dashboard/tcp
ess":"10.0.0.42","ifname":"eth0","listeners":
t":8081,"syn_queue":[]}],"connections":
al":"10.0.0.42:8081","remote":"64.38.131.114:8524","bytes_rx":
ytes_tx":271,"state":"ESTABLISHED"},
l":"10.0.0.42:8081","remote":"64.38.131.114:19621","bytes_rx":
ytes_tx":4198,"state":"ESTABLISHED"},
l":"10.0.0.42:8081","remote":"64.38.131.114:32985","bytes_rx":
ytes_tx":10974,"state":"ESTABLISHED"},
l":"10.0.0.42:8081","remote":"64.38.131.114:34250","bytes_rx":
ytes_tx":18038,"state":"ESTABLISHED"},
l":"10.0.0.42:8081","remote":"64.38.131.114:60826","bytes_rx":
ytes_tx":21377,"state":"ESTABLISHED"}]}
kB
853 bytes
ESTABLISHED
kB
8.9 kB
ESTABLISHED
Individual REST endpoints for each component.
100% done in application.
The OS exposes nothing except an API to access STL data structures from App

Routes!

RESTful endpoints for anything in IncludeOS

“Node/Express.js”-style routes

```
Router ro
// Assume

// GET /api/users/:id where 'id' is a numeral
router.on_get("/api/users/:id(\\d+)", [users] (auto req, auto res) {
    try {
      // Try to retrieve param "id"
      auto& params = req->params();
      std::string id = params.get("id");
      // Look for the user inside the bucket
      auto& user = users->pick_up(std::stoi(
    } catch (const std::exception& e) {
      res->send_code(http::Not_Found);
    }
  });
```

http://my_domain/api/users/42
matches regex and captures “id”, “42”

Regex-based mini-language for expressing routes.
ported to C++ from express.js

The “:id” part is made accessible to the app as key to a value.

Default route: serve from FAT fs

```
Router router;

// Serve index.html from root URL
router.on_get("/", [](auto, auto res) {
    // Stat disk for index.html
   disk->fs().cstat("/public/index.html", [res] (auto err, const auto& entry) {
     if(err)
       res->send_code(http::Not_Found);
     else
       // Serve index.html
       res->send_file({disk, entry});
   }
 });
});
```

Middleware

Applying a stack of functions to all requests

“Node/Express.js”-style middleware

```
server.use([](auto req, auto res, auto next) {
    if(has_content_type_json(req)) {
      auto json = std::make_shared<Json_doc>();
      json->doc().Parse(req->get_body().c_str());
      req->set_attribute(json);
    }
    (*next)();
  });

router.on_post("/users", [](auto req, auto res) {
    auto json = req->get_attribute<Json_doc>();
    if(json) {
      auto& doc = json->doc();
      // Register user based on parsed JSON data
    }
  });
```

Check for JSON content type

Parse and attach to request

Call next middleware (or exit to hang up)

Expect parsed JSON

Readymade middleware

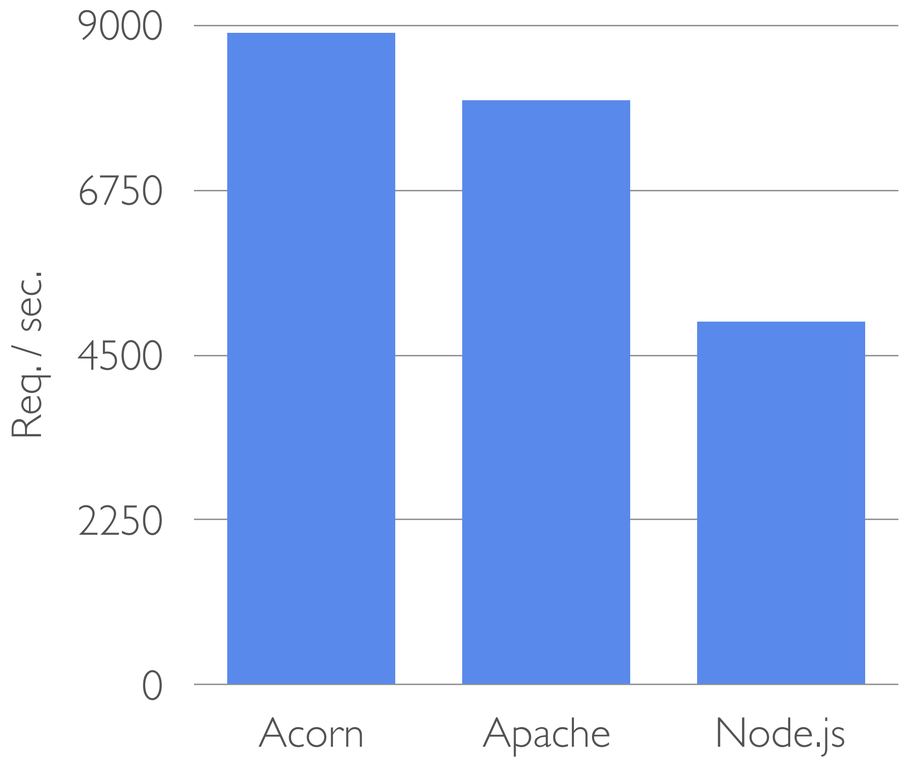
```
// Director: Provide an auto-generated HTML directory listing
auto director = std::make_shared<middleware::Director>(disk, "/public/static");
server.use("/static", director);


// Cookie parser: parse all cookies on incoming requests
auto cookie_parser = std::make_shared<middleware::CookieParser>();
server.use(cookie_parser);

// Butler: Serve static files on root route with disk root on /public
auto opt = {"index.html", "index.htm"};
auto butler = std::make_shared<middleware::Butler>(disk, "/public", opt);
server.use(butler);
```

Performance:
Virtual machines aren’t faster than bare metal

The goal is to have thousands of self-contained, secure micro services

“Acorn” preliminary performance data

- All services run in single core VM
- Apache / Node on Ubuntu 16.10
- “Acorn” on IncludeOS
  - 11% faster than Apache
  - 79% faster than Node.js
- 10k http req. with httperf
  - Average over 66 samples
  - Intel i7 desktop PC w. Ubuntu 16.04



Deploy in the cloud!

Deploy in OpenStack

Literally three commands

```
$ make
$ glance --image-create --file <My_image.img> …
$ nova boot --image …
```

Get an IP,
Fire up a browser!

We'll be adding support for AWS, Google compute, Azure, you name it.

Project status

- 1.0 “Stable API” depends on your feedback!
- Production ready core by years end
- “Acorn”: open source alpha today (!)
- TLS, POSIX and multi-arch are priorities
- Tooling improvements:
  - project unik for deployment / management
  - Package manager (3rd party)
  - CMake builds
  - IncludeOS specific CLI tool

includeOS

unik

Recap

- Virtual machines aren’t heavy - OS’es are
- IncludeOS is faster, more efficient, more secure for single purpose
- C++ is great at both high- and low level.
  - Definitely the best language for the job

includeOS

Please contribute!

Follow @includeos

- Pull Requests
- Ideas for cool use cases
- Expert advice!

alfred@includeos.org

Demo!

Open Content session showing the implementation of an IRC server on IncludeOS on Thursday

Fork:

Chat:

Questions?

References

1. “Unikernels: Library operating systems for the cloud”, Anil Madhavapeddy, Richard Mortier et. al, ASPLOS 2013
2. “Unikernels: Rise of the Virtual Library Operating System”, Anil Madhavapeddy, Dave Scott, ACM Queue, 2013
3. “IncludeOS: A minimal, Resource efficient Unikernel for Cloud Systems, A. Bratterud, A. A. Walla et. al, IEEE CloudCom 2015
4. “Formal requirements for Virtualizable Third Generation Architectures”, G. Popek, R.P. Goldberg, Communications of the ACM Vol. 17, 1974
5. “APLE: Addressing Lock Holder Preemption Problem with High Efficiency”, J. Shan, X. Ding and N. Gehani, IEEE CloudCom 2015