Towards smaller, safer, bespoke OSes with Unikernels

KC Sivaramakrishnan

IIT MADRAS

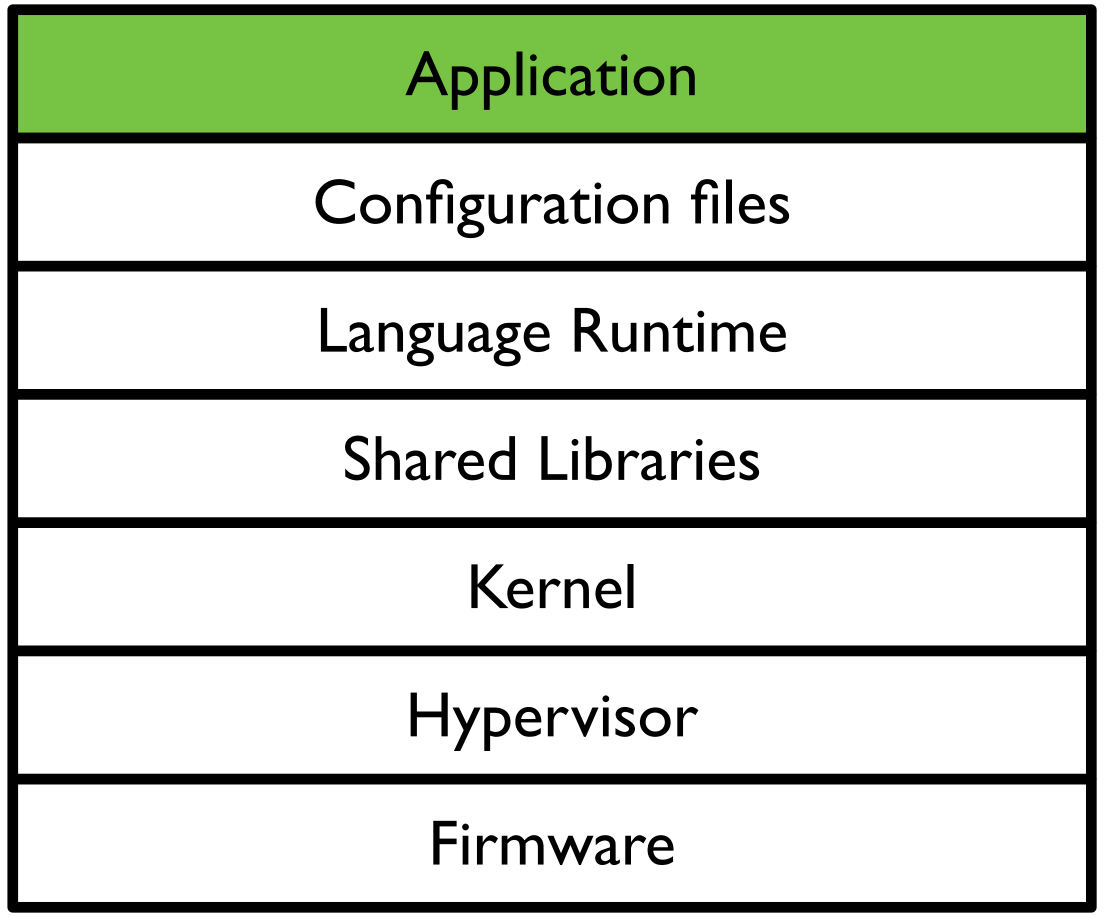
Tarides

In this talk…

You are here!

Why do we need an operating system?

- The main goal of an OS is to support running applications
  - **Stability:** most applications are not yet written when the system is deployed
  - **Scalability:** do not rewrite everything for every new hardware device
- OS does this by providing an abstraction over hardware
  - **Drivers** for different hardware devices
  - **Resource management:** files, users, CPU, memory, network
- Application code is ***a small %*** of the runtime environment



Application

Configuration files

Language Runtime

Shared Libraries

Kernel

Hypervisor

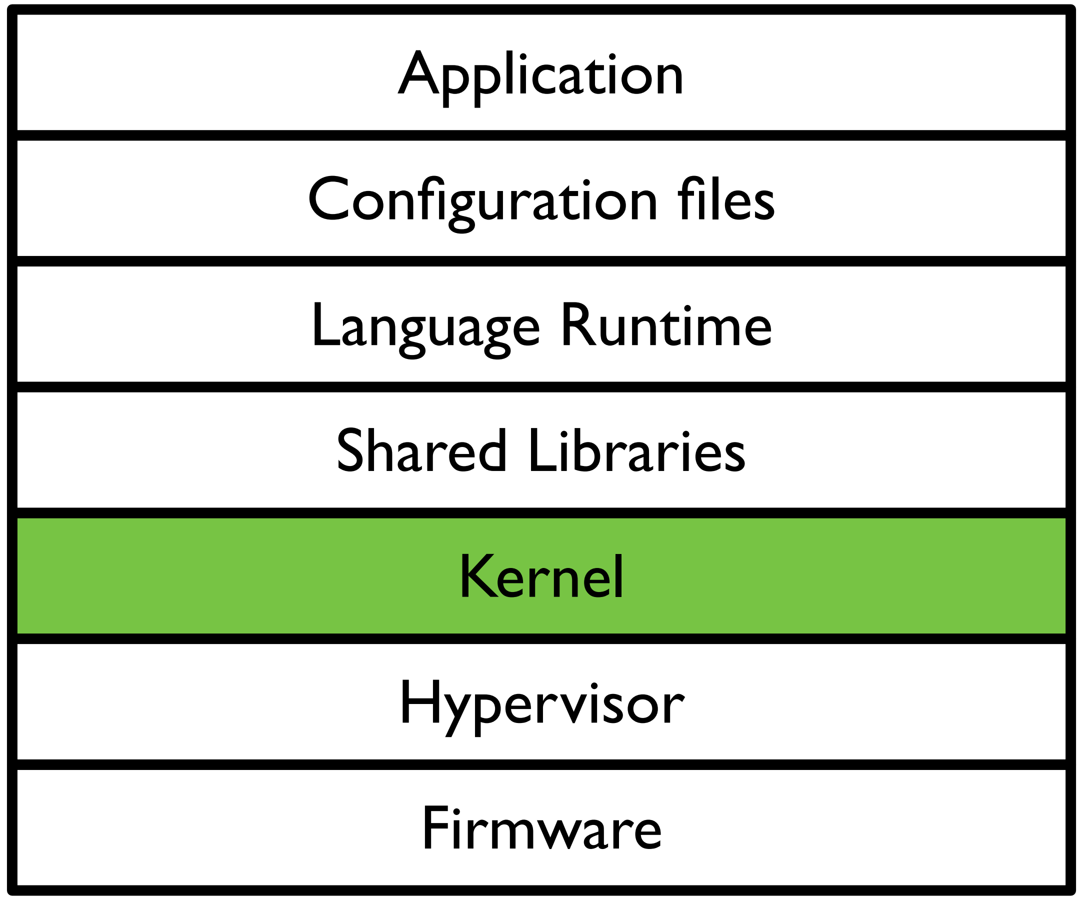
Firmware

Kernel: A Core OS component

"*True, Linux is monolithic, and I agree that microkernels are nicer... As has been noted (not only by me), the Linux kernel is a minuscule part of a complete system:*

Full sources for Linux currently run to about 200kB compressed. And all of that source is portable, except for this tiny kernel that you can (provably: I did it) re-write totally from scratch in less than a year without having /any/ prior knowledge."

– Linus Torvalds, 1992



Application

Configuration files

Language Runtime

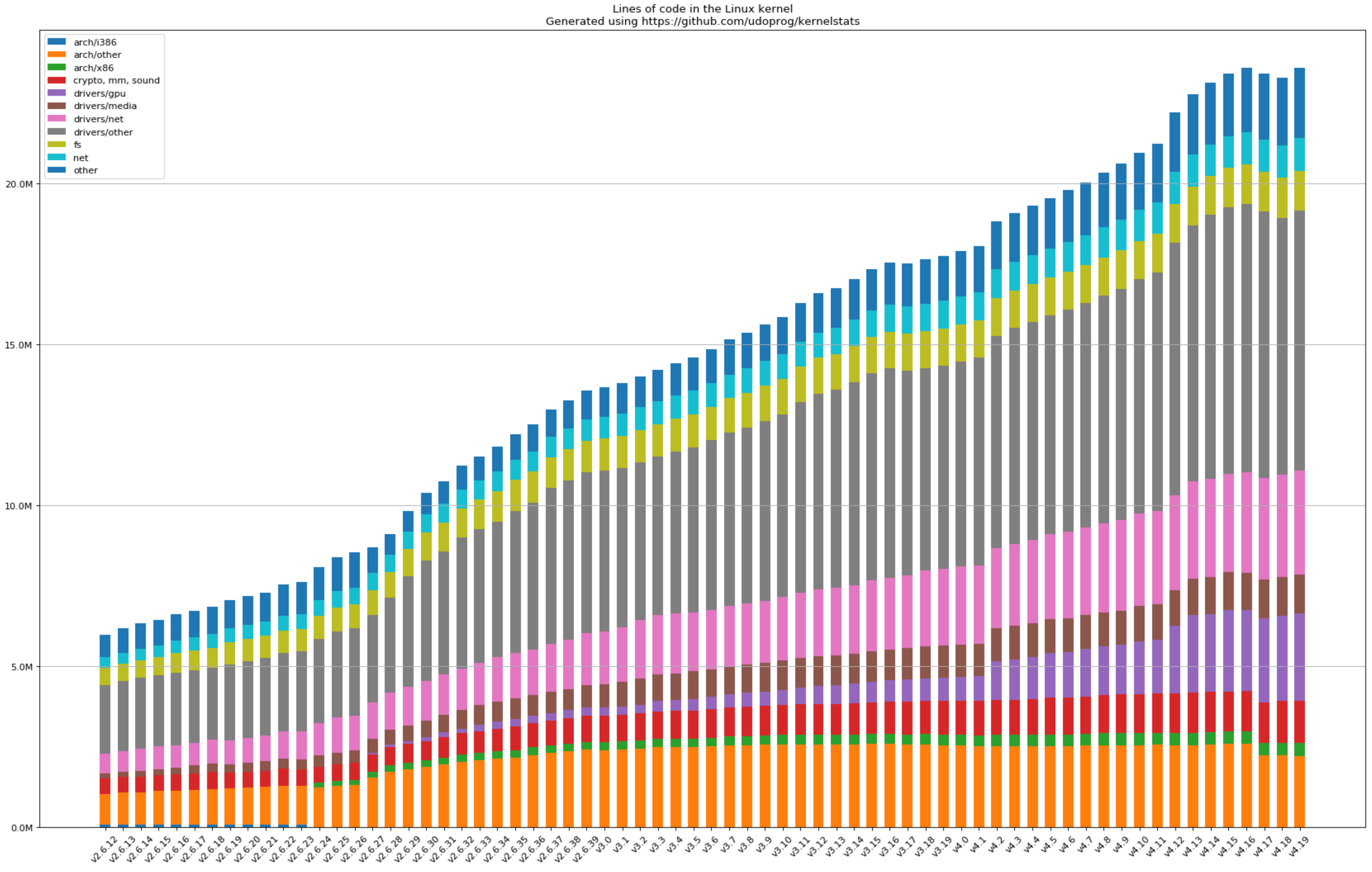
Shared Libraries

Kernel

Hypervisor

Firmware

Linux Kernel



Lines of code in the Linux kernel
Generated using https://github.com/udoprog/kernelstats

Drivers!

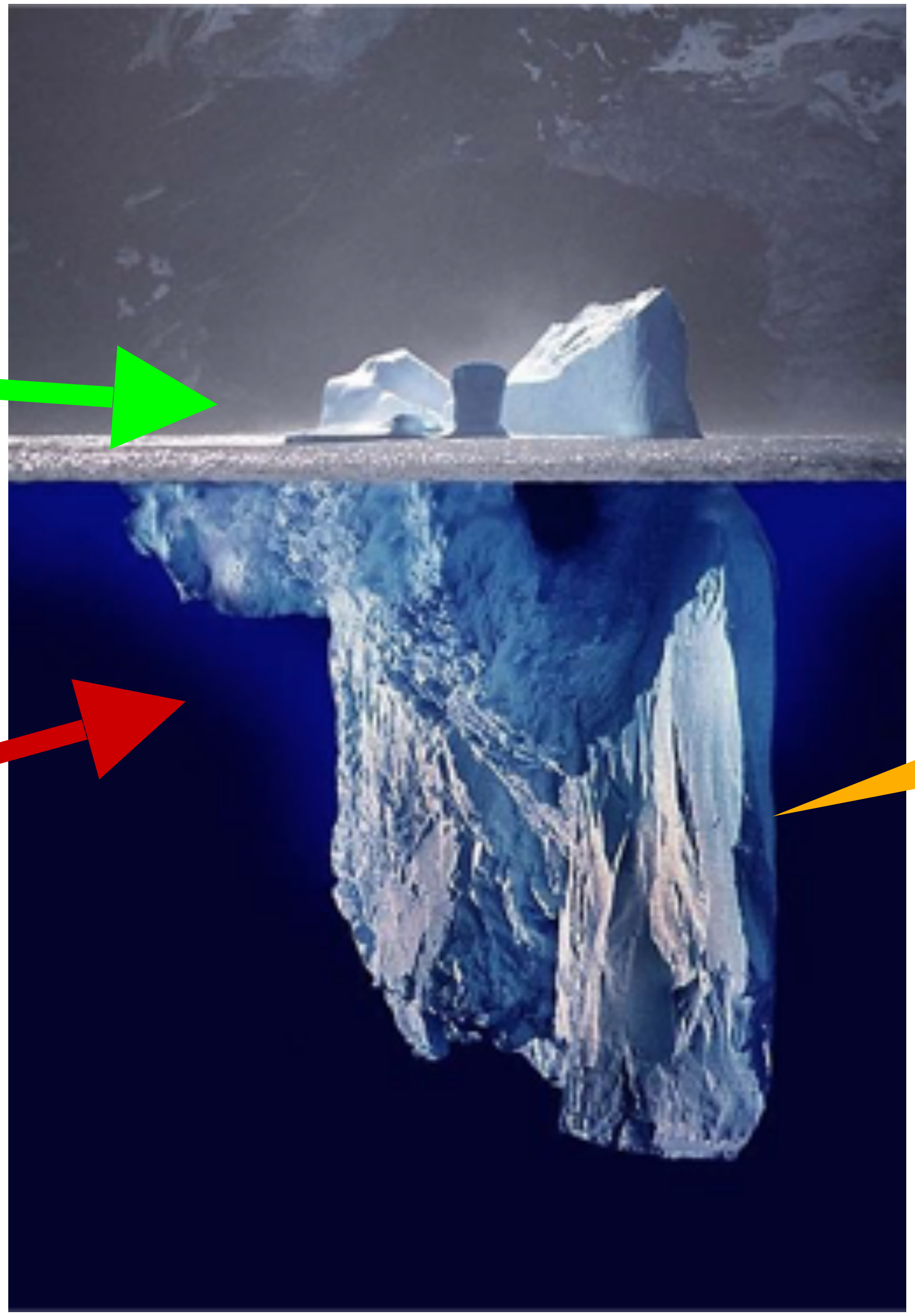
Linux 5.11 has 30.14 million lines of code, 60% drivers

Windows has 50 million lines of code

Monolithic OS Icebergs

Code you want to run

Code your operating system insists you need!



Huge TCB ⟹ Security concern

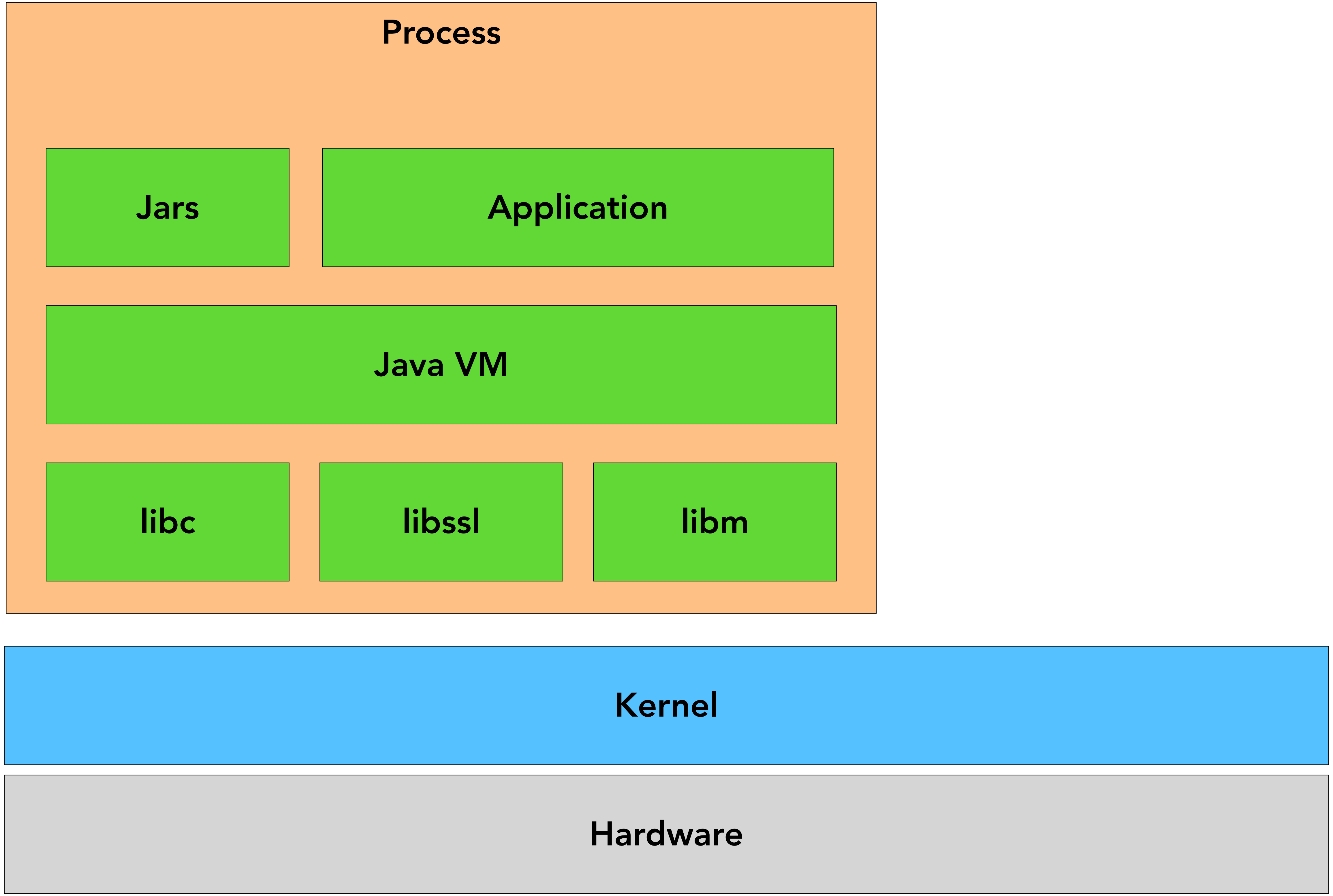
How do we reduce the OS complexity?

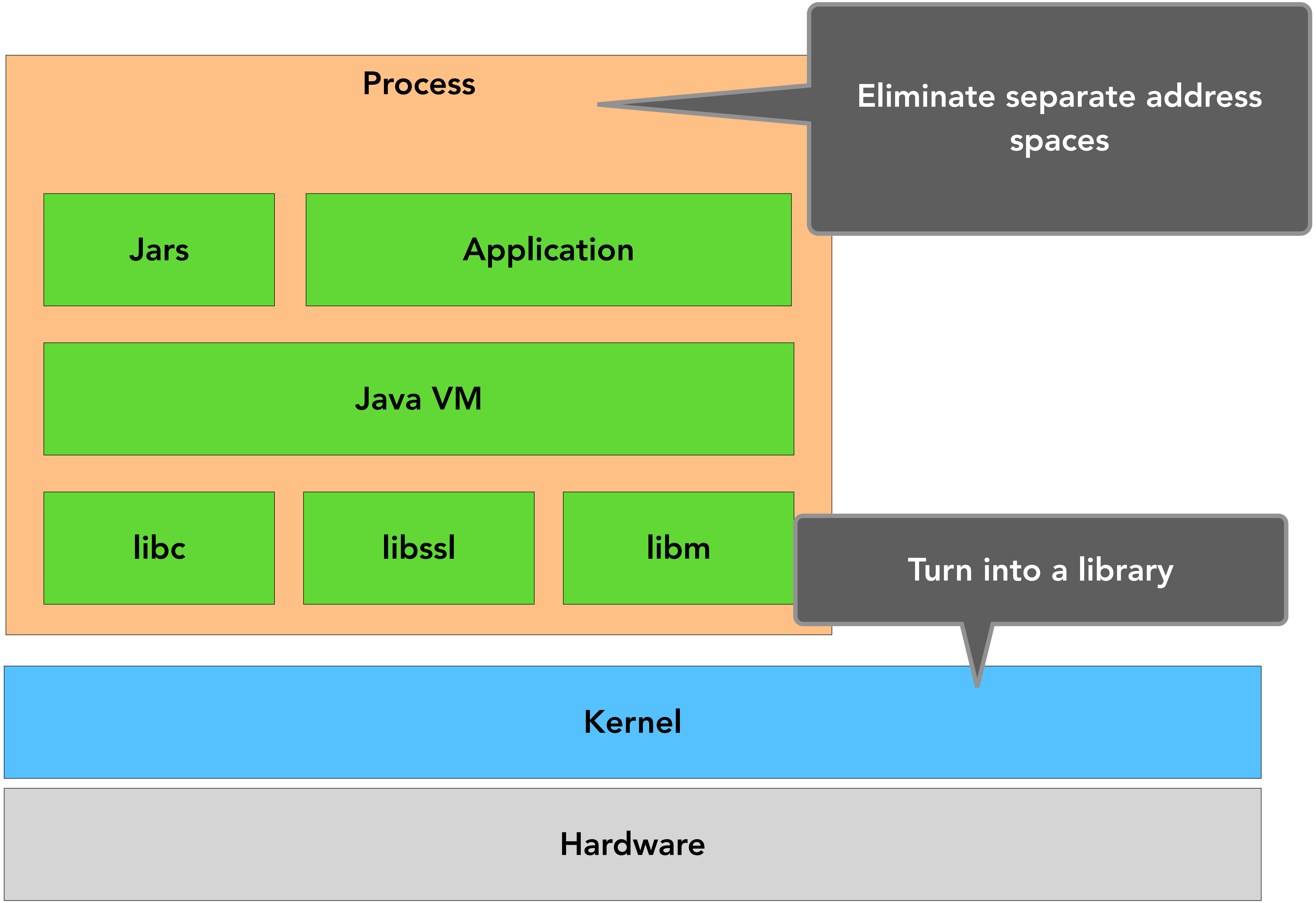
Here’s our take…

Ingredient 1: Library OS

Library operating systems

- Kernel functionality is broken up from its ***monolith*** into many ***individual libraries.***
  - There is no ambient kernel; just ***function calls*** are left.
- Device drivers, schedulers, networking, and storage stacks are ***directly linked*** to the application
  - Eliminate the need for an intermediary kernel layer.
  - Applications **select libraries they need** with a small boot layer and jump straight into the code.
- Hardware is driven directly from the application, usually in a single address space.


Process
Jars
Application
Java VM
libc
libssl
libm
Kernel
Hardware



Process

Eliminate separate address spaces

Jars

Application

Java VM

libc

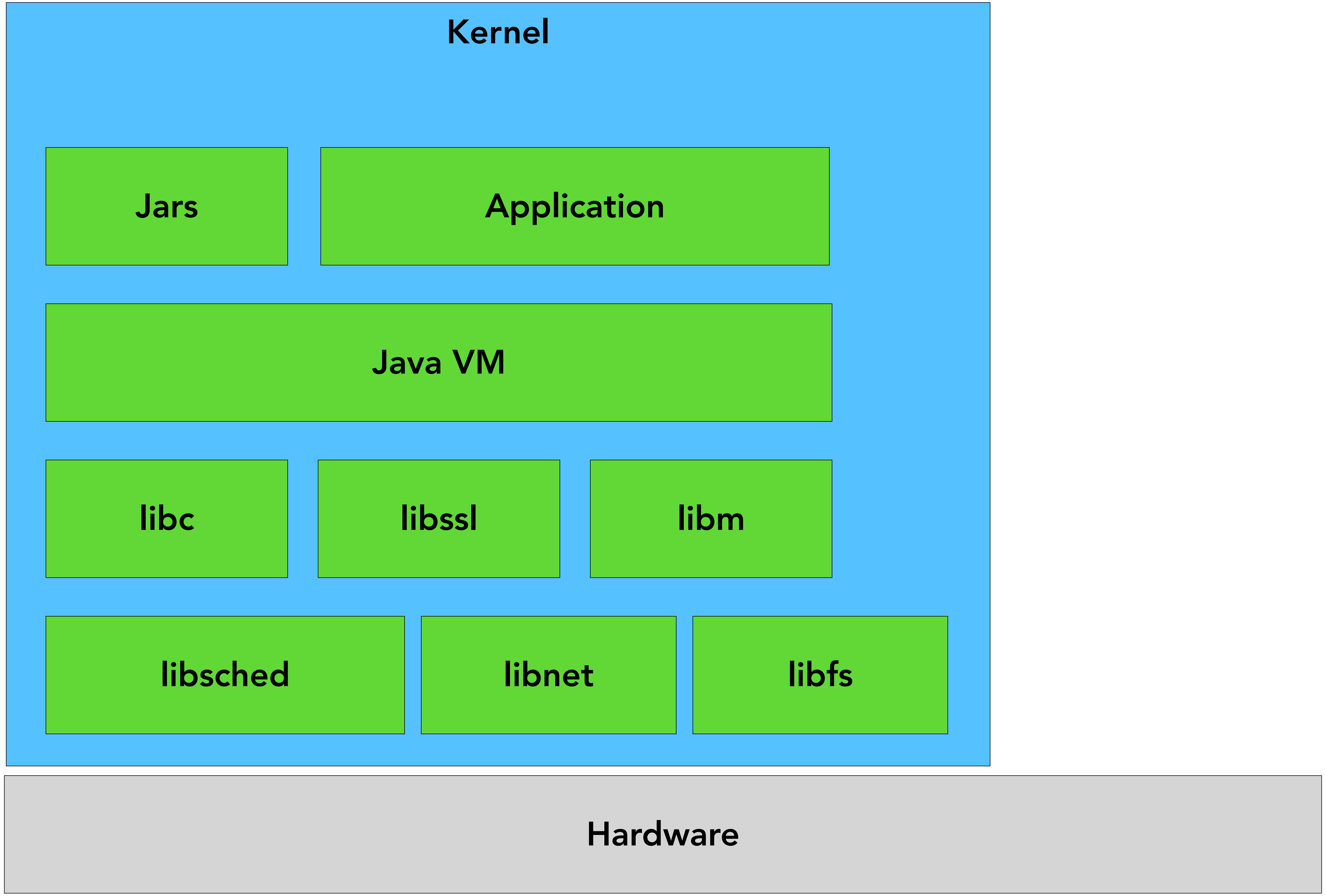
libssl

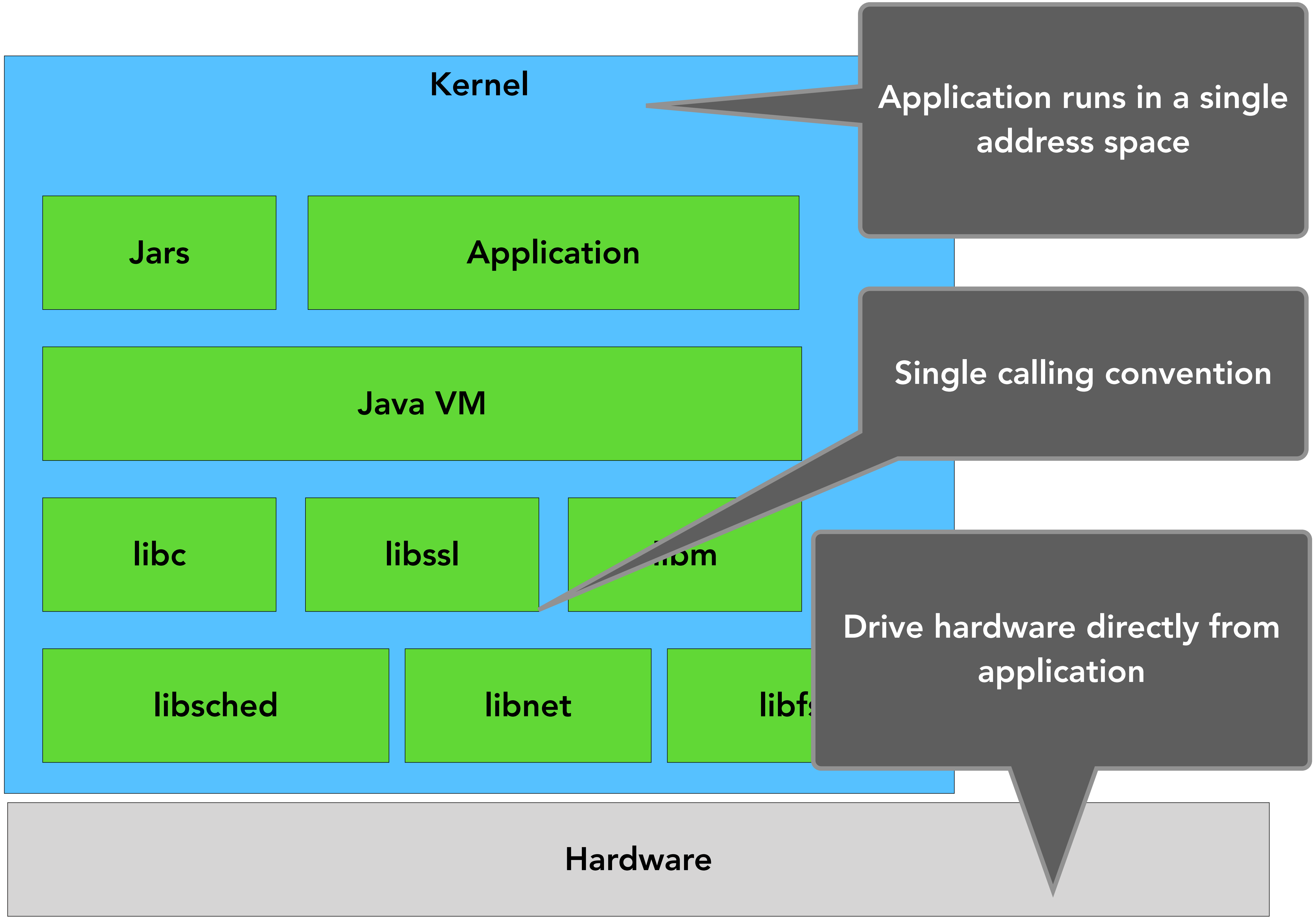
libm

Turn into a library

Kernel

Hardware


Kernel
Jars
Application
Java VM
libc
libssl
libm
libsched
libnet
libfs
Hardware



Application runs in a single address space

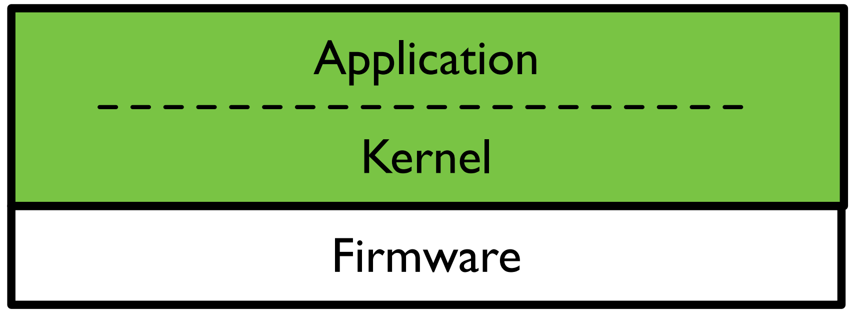
Single calling convention

Drive hardware directly from application

Library operating systems: History

- In the 90s, we had
  - **Nemesis:** Cambridge/Glasgow
  - Exokernel: MIT
- Neither succeeded outside of academia due to the device drivers needing to be updated regularly to stay relevant.
- Became popular in niche areas (network appliances or high-frequency trading).

Library operating systems: Pros & Cons



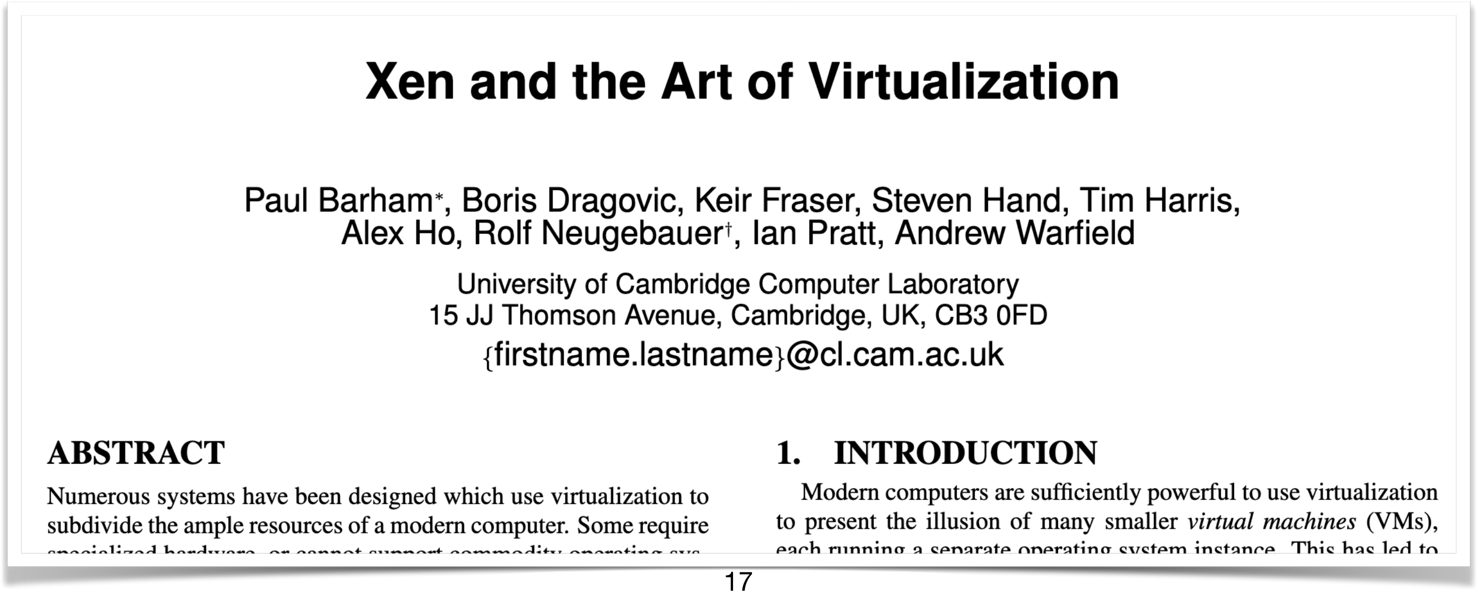
Pros: application-level control of hardware, small attack surface, high-performance.

Cons: There is no kernel protection internally, and device drivers all need to be rewritten from a normal kernel.

Ingredient 2: Virtualisation

Virtualisation

- In the 2000s, hardware vendors added extensions that allow the creation of virtual versions of physical resources, such as servers, networks, and storage devices.
- It enables multiple virtual machines (VMs), with their own operating systems, to **run in isolation, side-by-side, on the same physical hardware**.
- Hypervisor (aka VMM) — creates and runs virtual machines



Xen and the Art of Virtualization

Paul Barham[*], Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris,
Alex Ho, Rolf Neugebauer[†], Ian Pratt, Andrew Warfield
University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

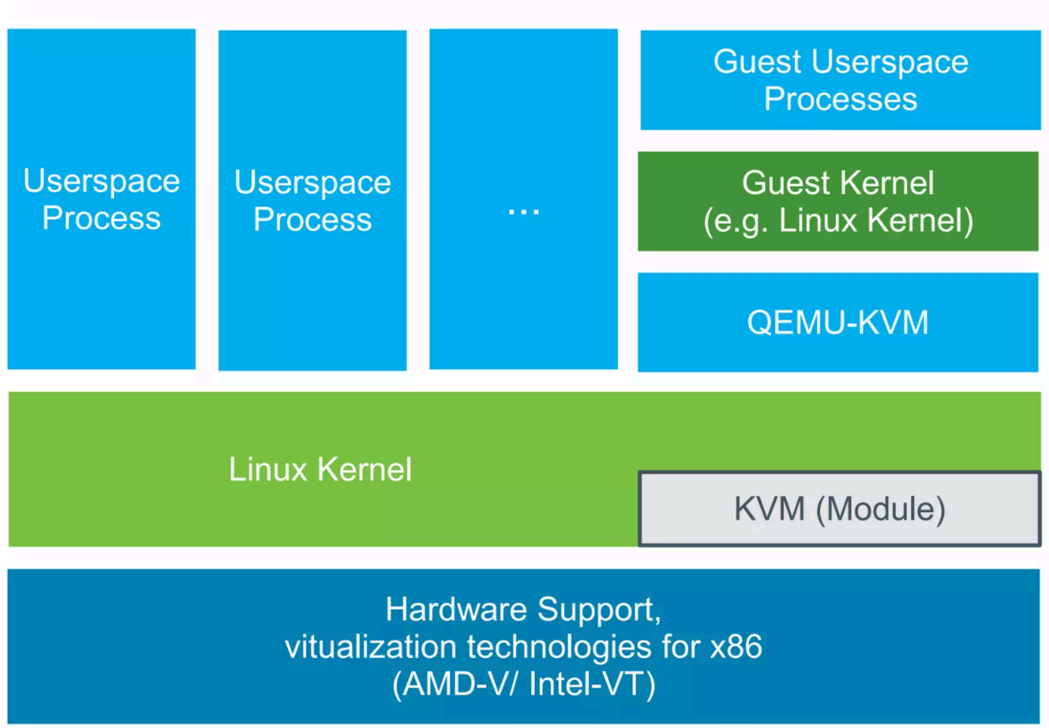
ABSTRACT

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require specialized hardware, or cannot support commodity operating sys-

1. INTRODUCTION

Modern computers are sufficiently powerful to use virtualization to present the illusion of many smaller *virtual machines* (VMs), each running a separate operating system instance. This has led to

Linux KVM


Userspace Process
Userspace Process
...
Guest Userspace Processes
Guest Kernel (e.g. Linux Kernel)
QEMU-KVM
Linux Kernel
KVM (Module)
Hardware Support, vitualization technologies for x86 (AMD-V/ Intel-VT)

- Turns Linux into a Type 1 VMM
- QEMU emulates CPUs and missing hardware
- **VirtIO** — virtualisation of networks and disk device drivers
    - *Can take advantage of Linux Kernel's vast driver support!*

Library operating systems

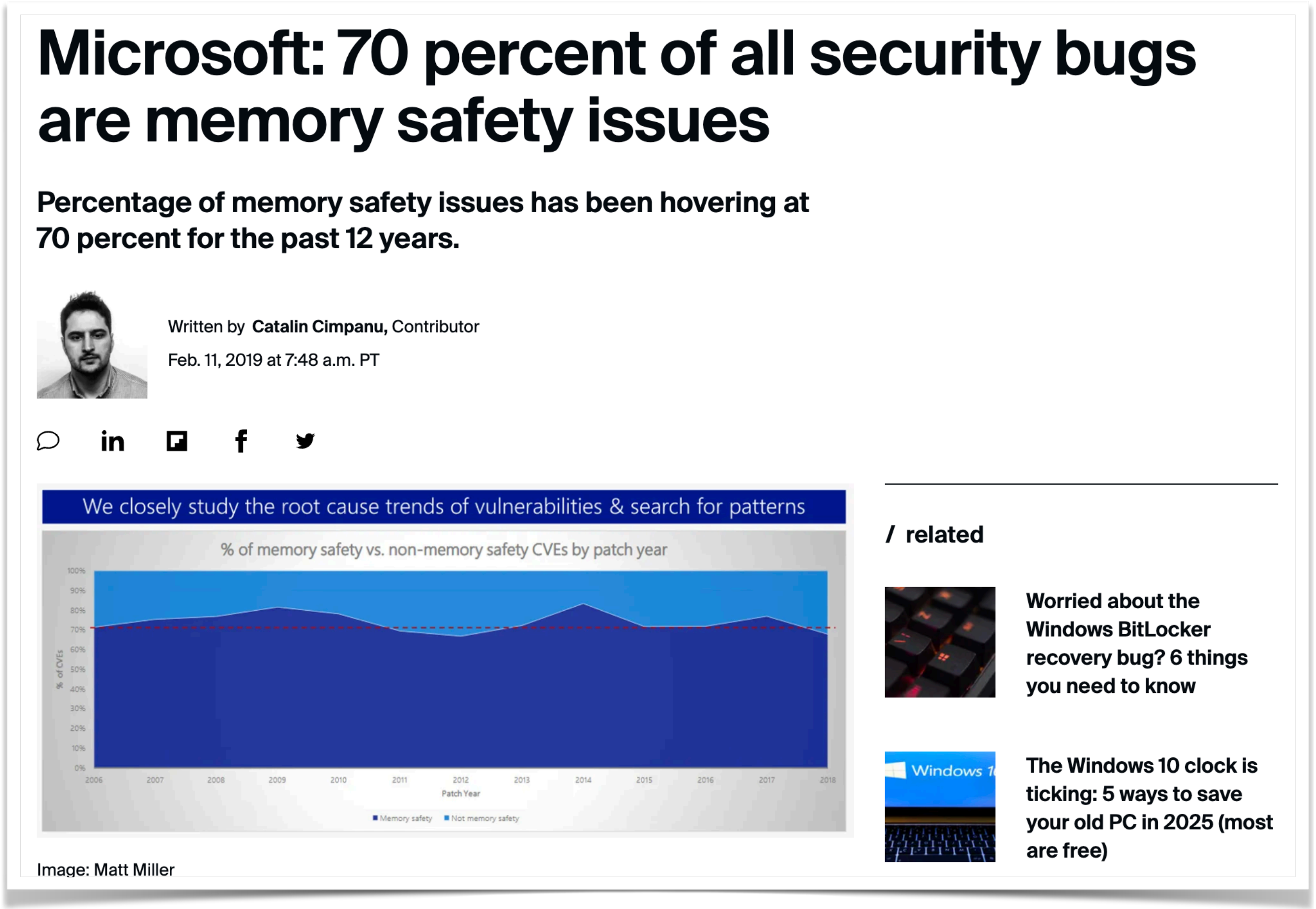
Cons: There is no kernel protection internally, and ~~device drivers all need to be rewritten from a normal kernel.~~

Ingredient 3: OCaml

Memory safety

Library operating systems

Cons: There is no kernel protection internally, and ~~device drivers all need to be rewritten from a normal kernel.~~



Microsoft: 70 percent of all security bugs are memory safety issues

Percentage of memory safety issues has been hovering at 70 percent for the past 12 years.

Written by **Catalin Cimpanu,** Contributor

Feb. 11, 2019 at 7:48 a.m. PT

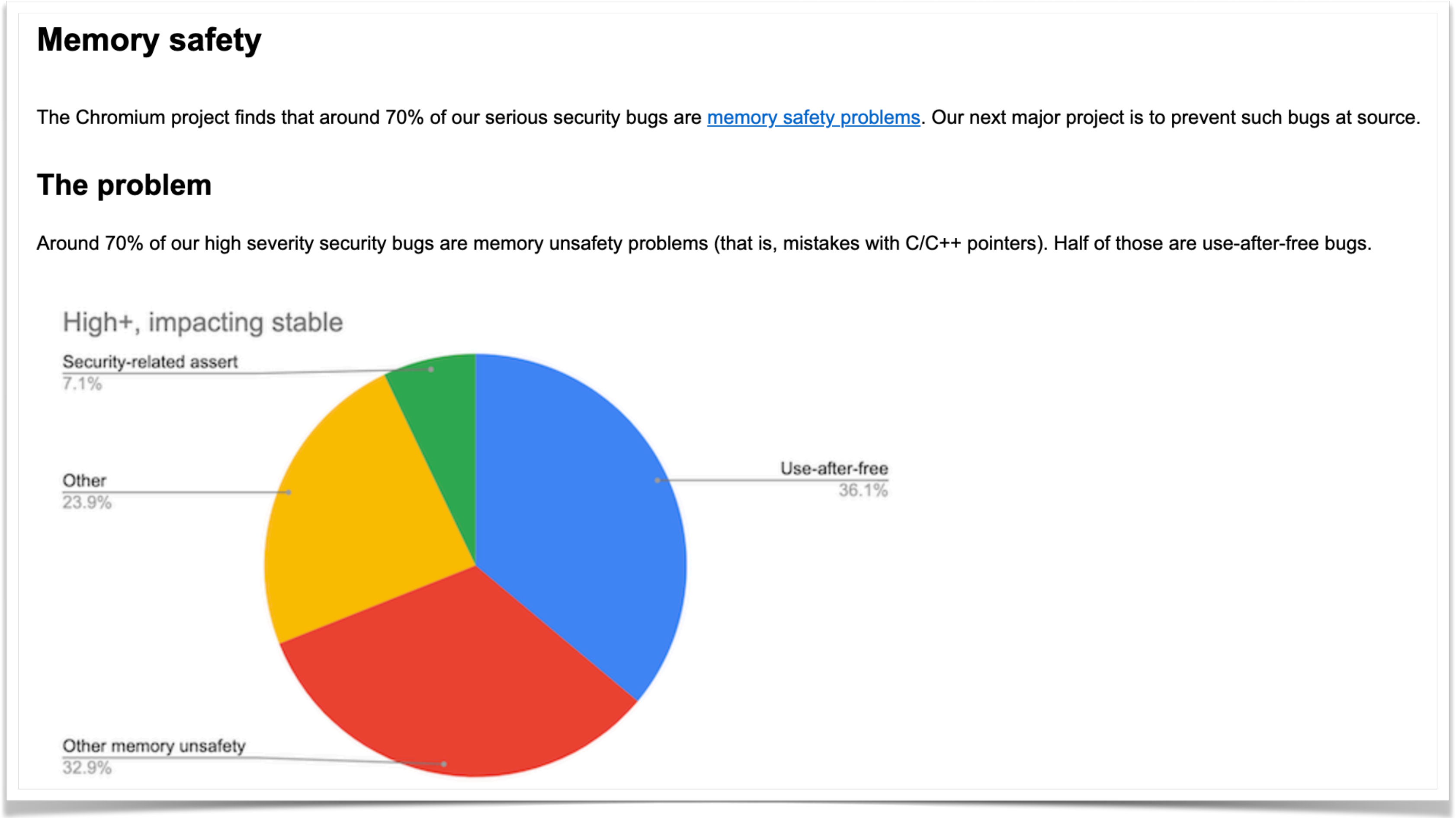
We closely study the root cause trends of vulnerabilities & search for patterns
% of memory safety vs. non-memory safety CVEs by patch year
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
% of CVEs
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
Patch Year
Memory safety
Not memory safety

Image: Matt Miller

/ related

Worried about the Windows BitLocker recovery bug? 6 things you need to know

The Windows 10 clock is ticking: 5 ways to save your old PC in 2025 (most are free)



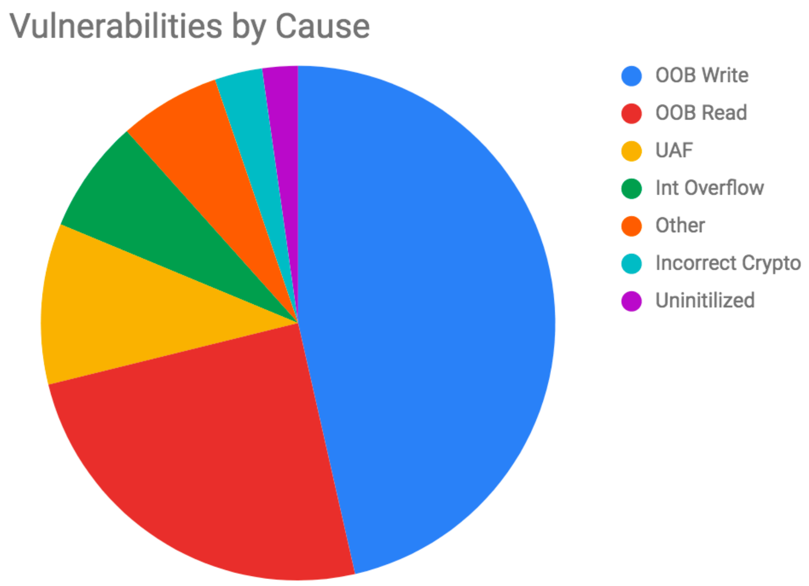
Memory safety

The Chromium project finds that around 70% of our serious security bugs are memory safety problems. Our next major project is to prevent such bugs at source.

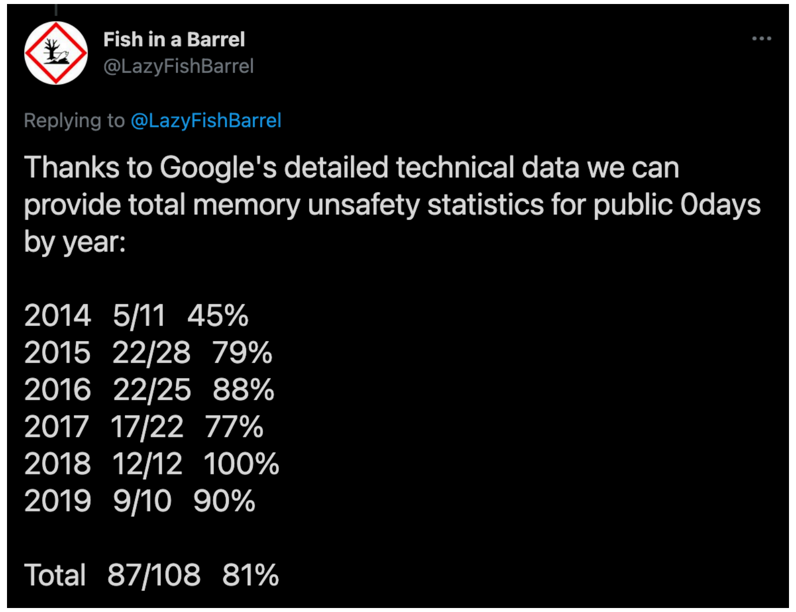
The problem

Around 70% of our high severity security bugs are memory unsafety problems (that is, mistakes with C/C++ pointers). Half of those are use-after-free bugs.

Memory safety


Vulnerabilities by Cause

90% of Android vulnerabilities are memory safety issues



Fish in a Barrel
@LazyFishBarrel

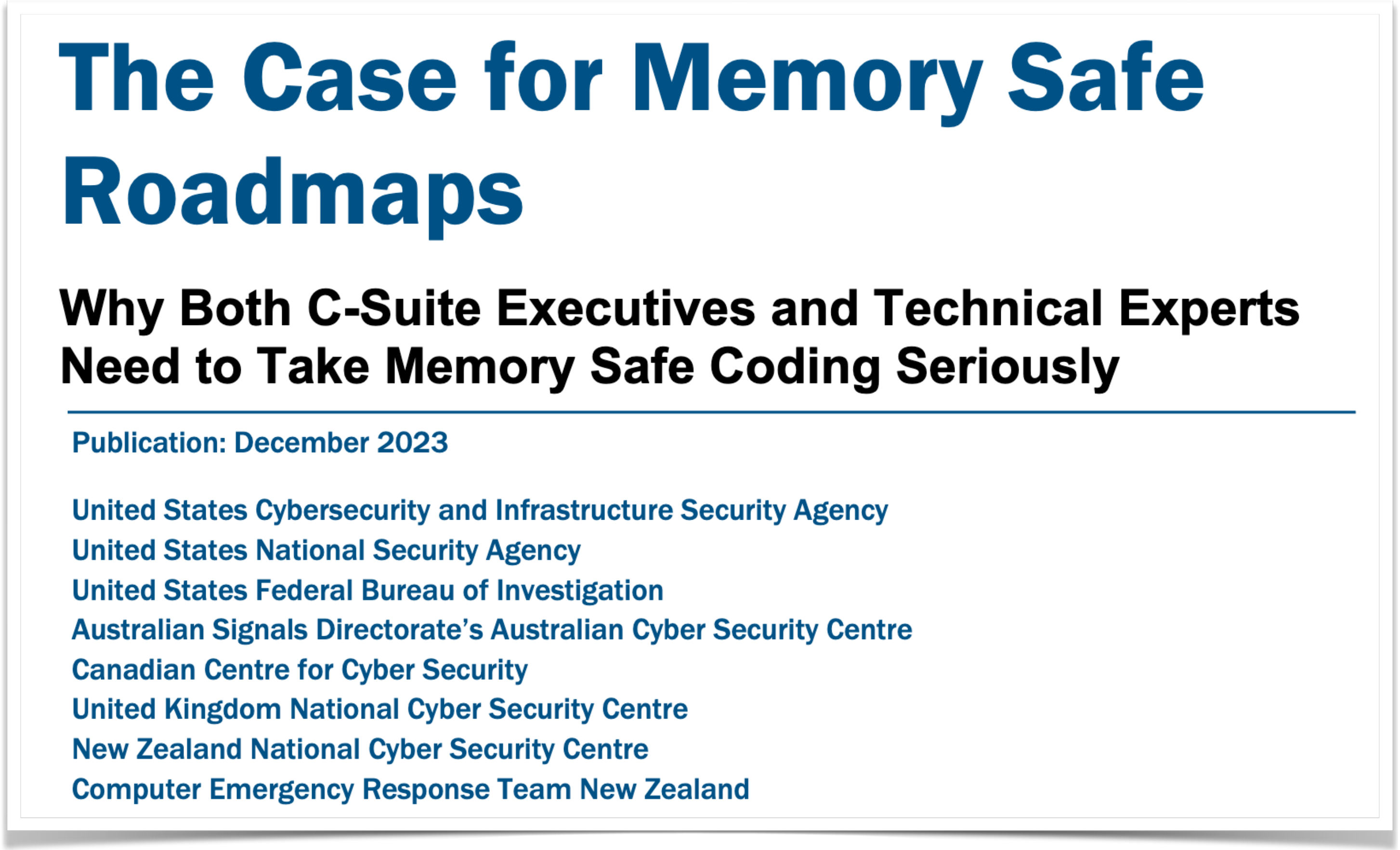
Replying to @LazyFishBarrel

Thanks to Google's detailed technical data we can provide total memory unsafety statistics for public 0days by year:

2014	5/11	45%
2015	22/28	79%
2016	22/25	88%
2017	17/22	77%
2018	12/12	100%
2019	9/10	90%
Total	87/108	81%

80% of the exploited vulnerabilities of known 0-days were memory safety issues

Memory safety

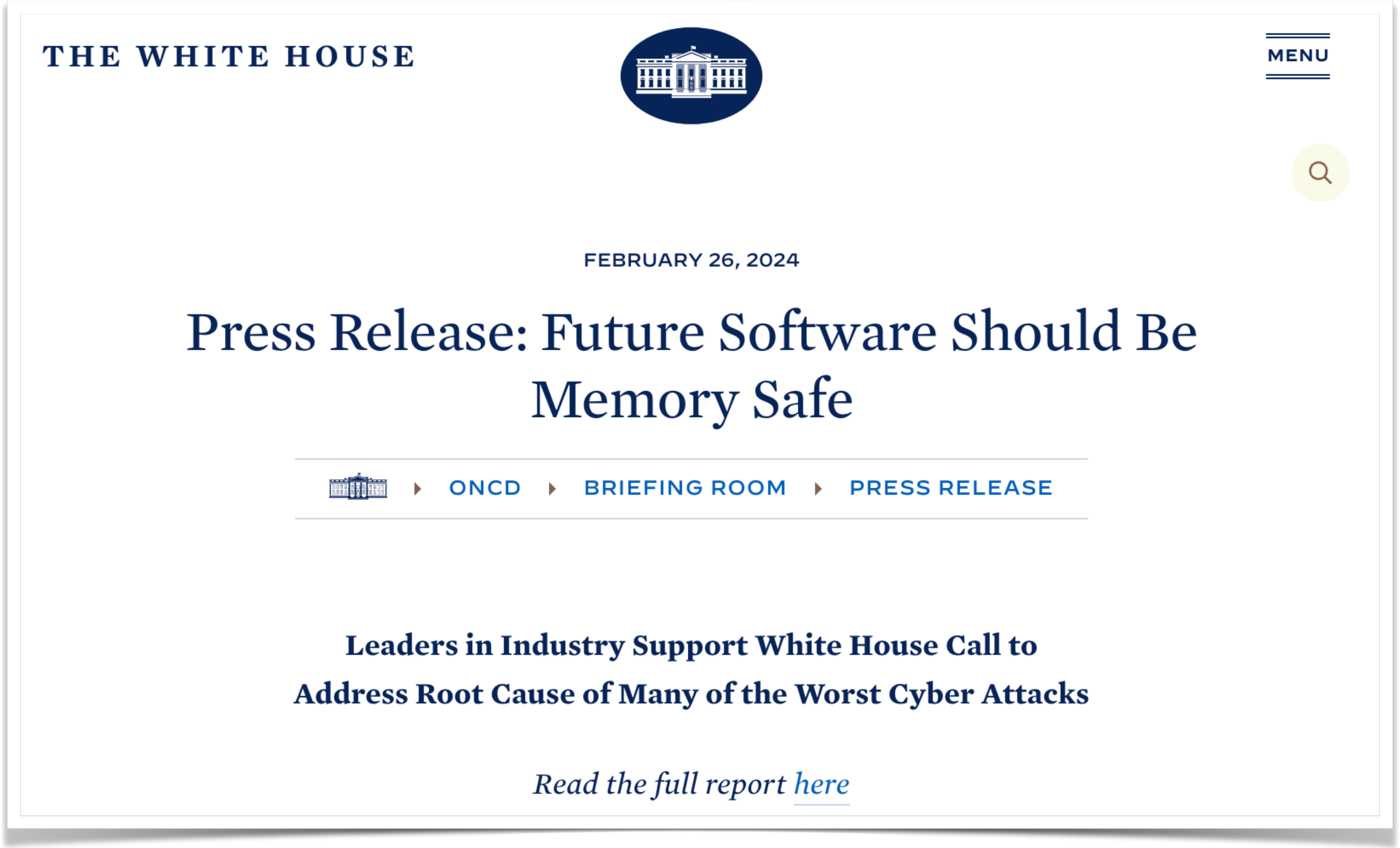


The Case for Memory Safe Roadmaps

Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously

Publication: December 2023

United States Cybersecurity and Infrastructure Security Agency
United States National Security Agency
United States Federal Bureau of Investigation
Australian Signals Directorate's Australian Cyber Security Centre
Canadian Centre for Cyber Security
United Kingdom National Cyber Security Centre
New Zealand National Cyber Security Centre
Computer Emergency Response Team New Zealand



FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

Memory safety and Programming Languages

- Unsafe languages
  - C, C++, Assembly, Objective-C
- Safe languages
  - With the help of a garbage collector (GC) — JavaScript, Python, Java, Go, OCaml, …
  - Without a GC — Rust
- Unsafe parts of safe languages
  - Unsafe Rust, unsafe package in Go, Obj in OCaml

Library operating systems

Cons: ~~There is no kernel protection internally, and device drivers all need to be rewritten from a normal kernel.~~

OCaml

industrial-strength, pragmatic, functional programming language

Industry

FACEBOOK Tarides Bloomberg

docker ahrefs SimCorp

Microsoft Tezos Jane Street

Projects

hack flow

COMPCERT

Higher-order functions

Hindley-Milner Type Inference

Powerful module system

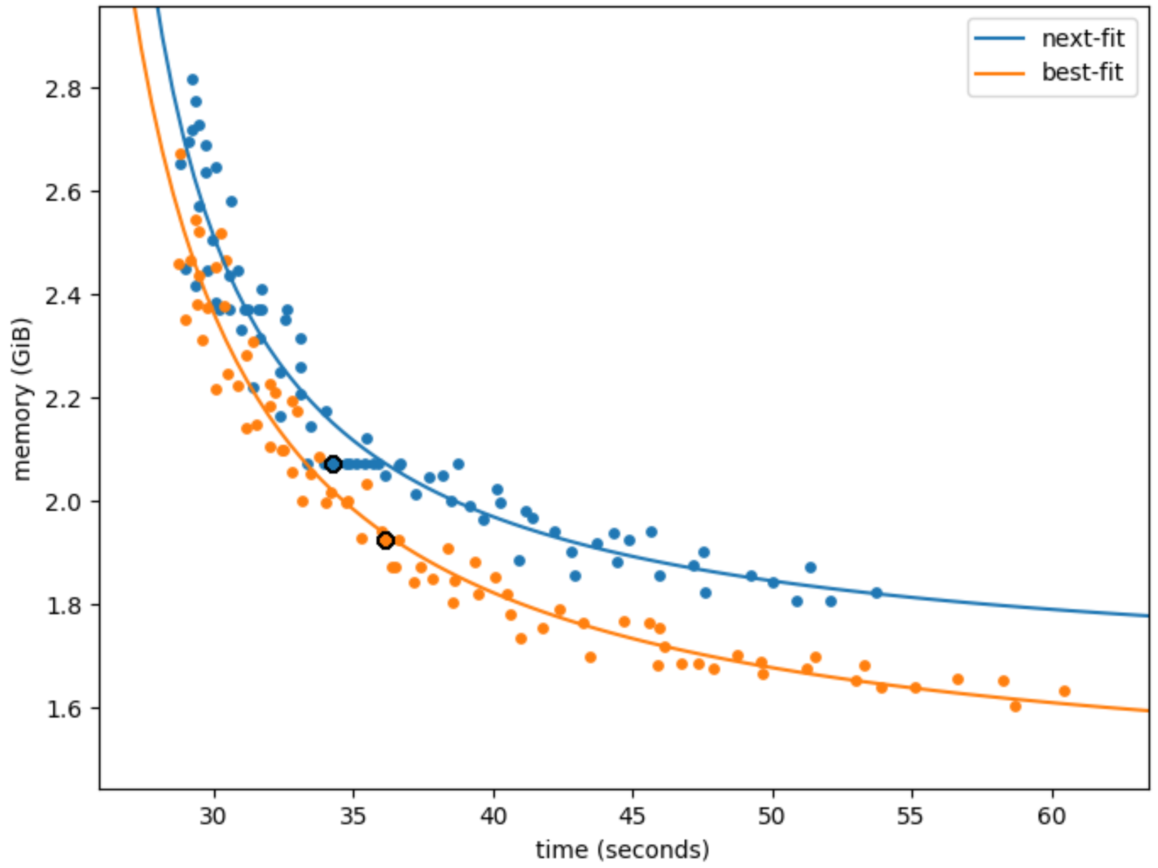
20% of Wall Street trade goes through OCaml

Functional core with imperative and object-oriented features

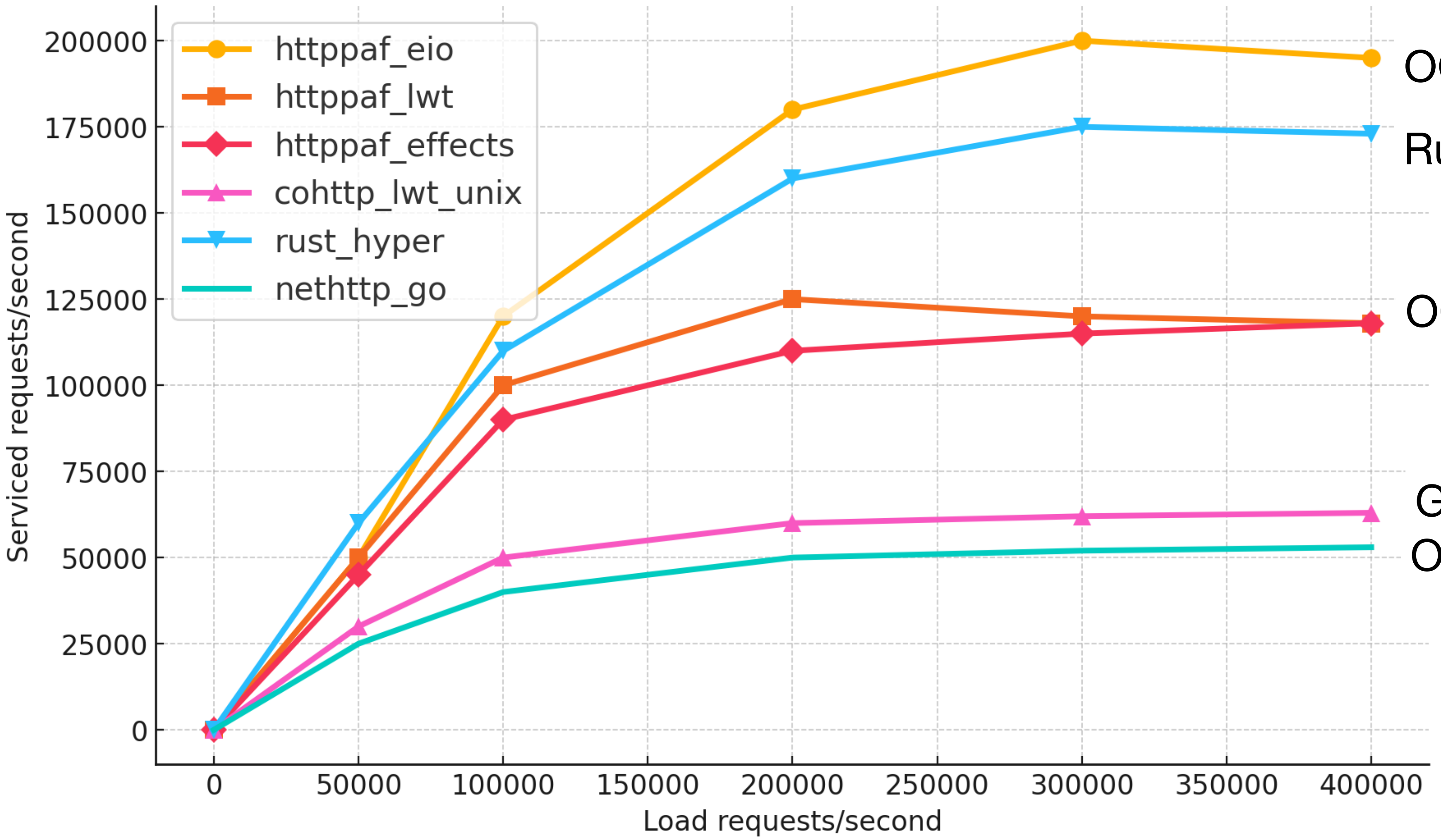
Native (x86, Arm, Power, RISC-V), JavaScript, WebAssembly

OCaml Performance

- GC is tuned for low-latency
  - If your application can tolerate 1 ms latency, then OCaml is a good fit
  - 95% of code that we write fit this model
- GC is a tradeoff between space and time
- OCaml is typically 1.5x to 2x slower than C for algorithmic workloads
  - Python will be 10x to 100x slower than C
- Fast FFI to C for speed



OCaml Performance — Web Server



OCaml eio

Rust Hyper

OCaml (Http/af + Lwt)

Go NetHttp

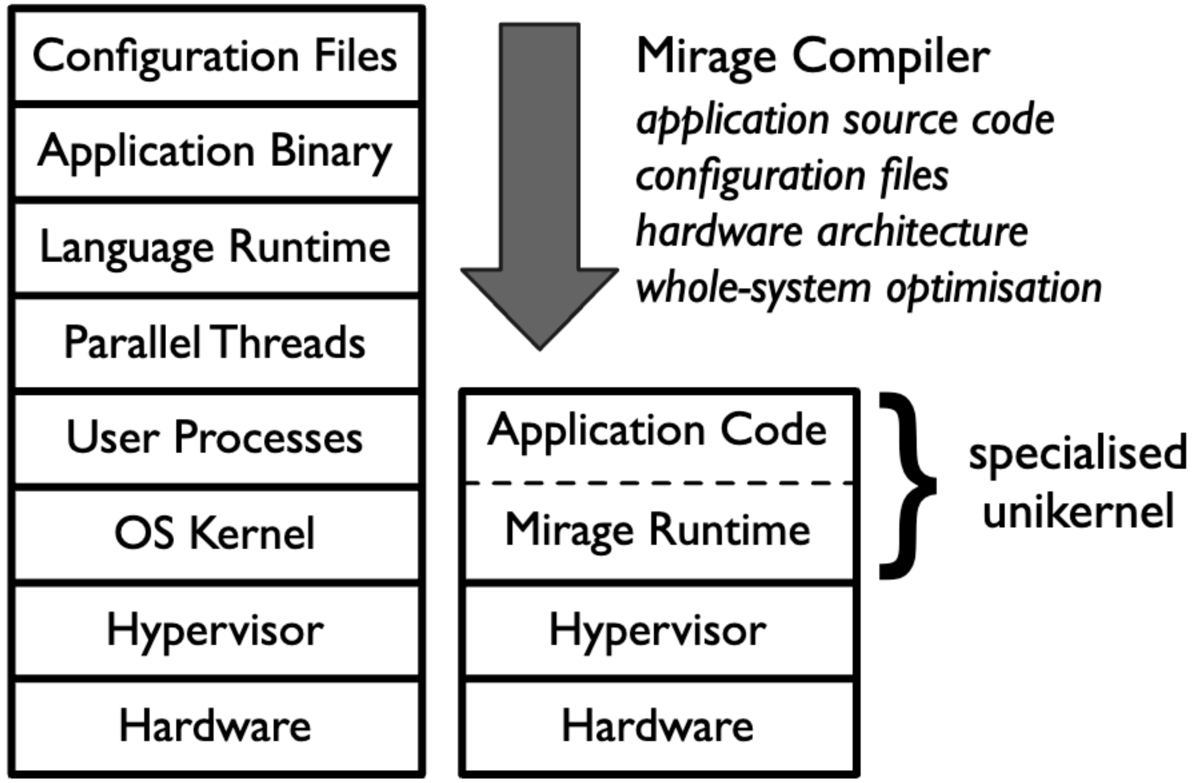
OCaml (cohttp + Lwt)

https://github.com/ocaml-multicore/eio

MirageOS = Library OS + Virtualisation + OCaml

MirageOS Unikernels

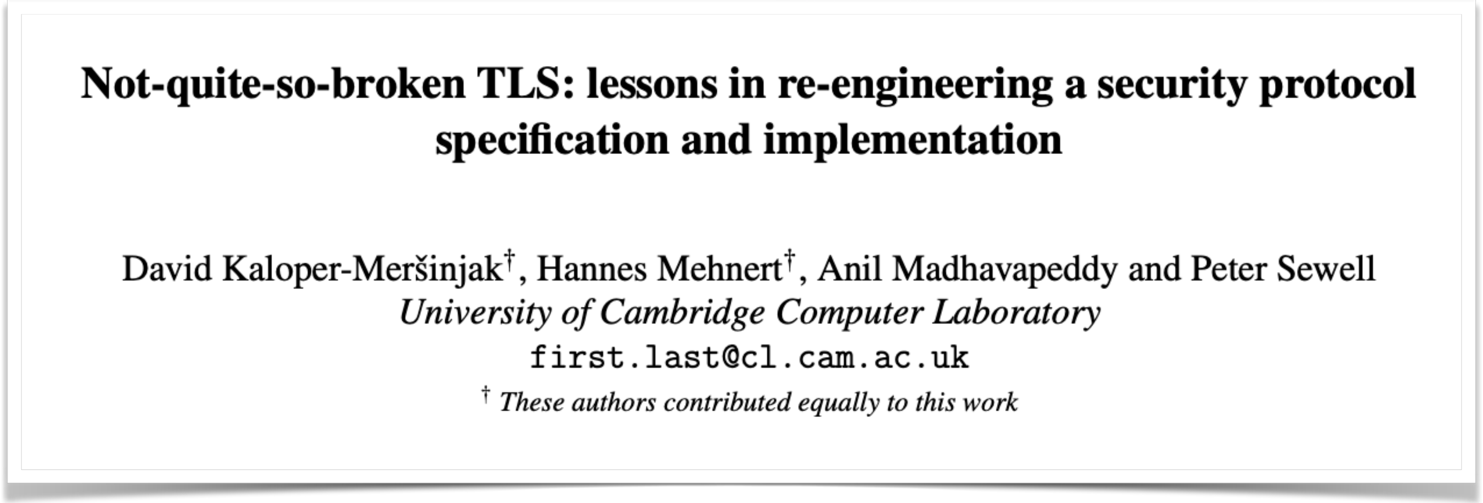
- MirageOS is a **library OS** and a **compiler** that can build specialised images containing only the runtime environment needed by the application
  - Cut the complexity by designing the layers as independent type-safe libraries.
- The MirageOS compiler transforms an application manifest into a **specialised image.**
  - Rely on the OCaml compiler for modular static analysis, dead-code elimination, etc.
- Rely on the OCaml runtime as the sole trusted runtime environment (and selected C bindings)



Available Libraries

Network:
Ethernet, IP, UDP, TCP, HTTP 1.0/1.1/2.0, ALPN, DNS, ARP, DHCP, SMTP, IRC, cap-n-proto, emails
Storage:
block device, Ramdisk, Qcow, B-trees, VHD, Zlib, Gzip, Lzo, Git, Tar, FAT32
Data-structures:
LRU, Rabin's fingerprint, bloom filters, adaptative radix trees, discrete interval encoding trees
Security:
x.509, ASN1, TLS, SSH
Crypto:
hashes, checksums
Ciphers (AES, 3DES, RC4, ChaCha20/Poly1305)
AEAD primitives (AES-GCM, AES-CCM)
Public keys (RSA, DSA, DH)
Fortuna

- Reimplemented in OCaml
- TLS: "rigorous engineering"
  - same pure code to generate test oracles, verify oracle against real-world TLS traces and the real implementation
  - Use Fiat (Coq extraction) for crypto primitives.



Not-quite-so-broken TLS: lessons in re-engineering a security protocol specification and implementation

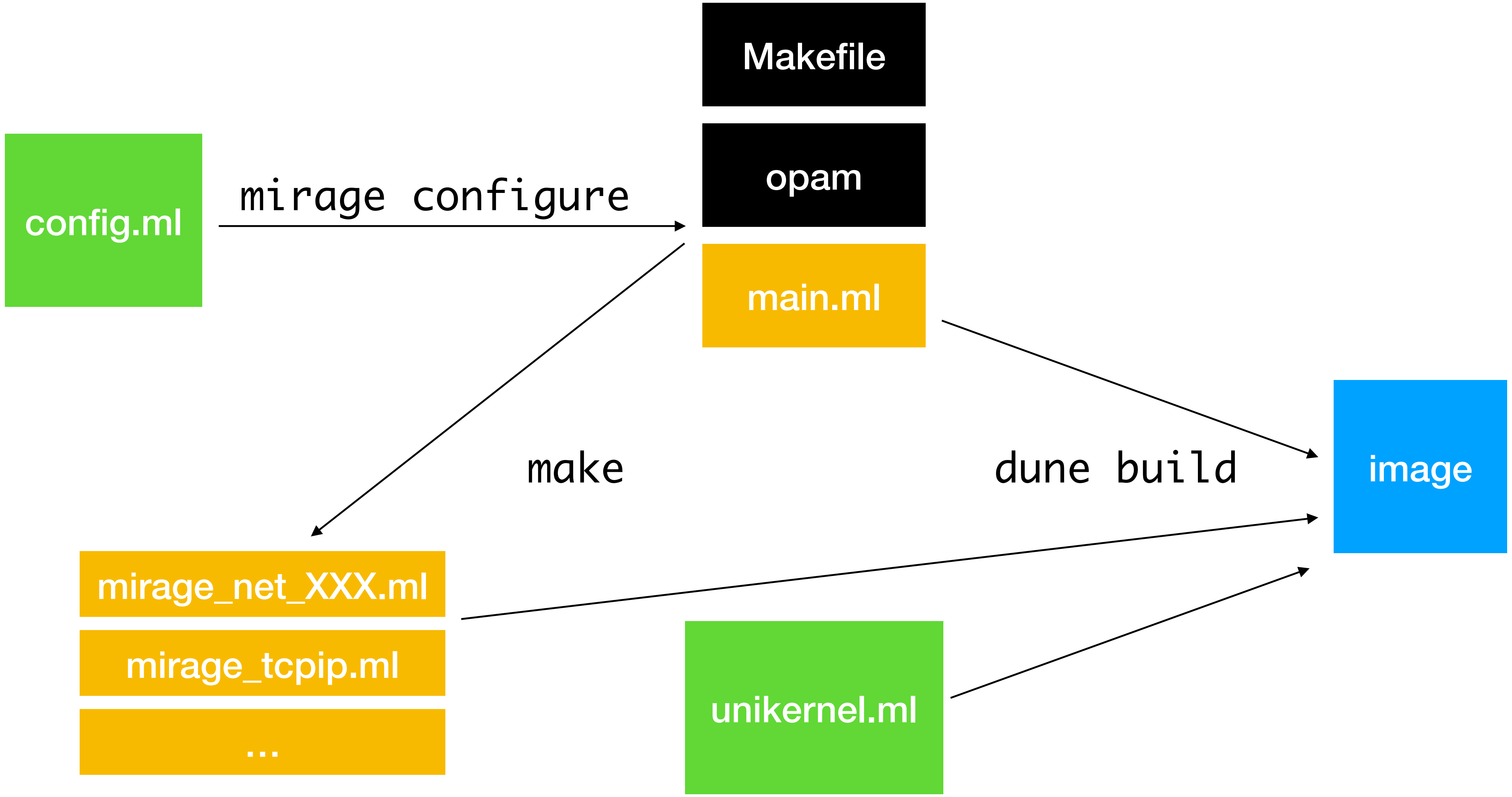
David Kaloper-Meršinjak[†], Hannes Mehnert[†], Anil Madhavapeddy and Peter Sewell
University of Cambridge Computer Laboratory
first.last@cl.cam.ac.uk
[†] *These authors contributed equally to this work*

What is a MirageOS Unikernel?

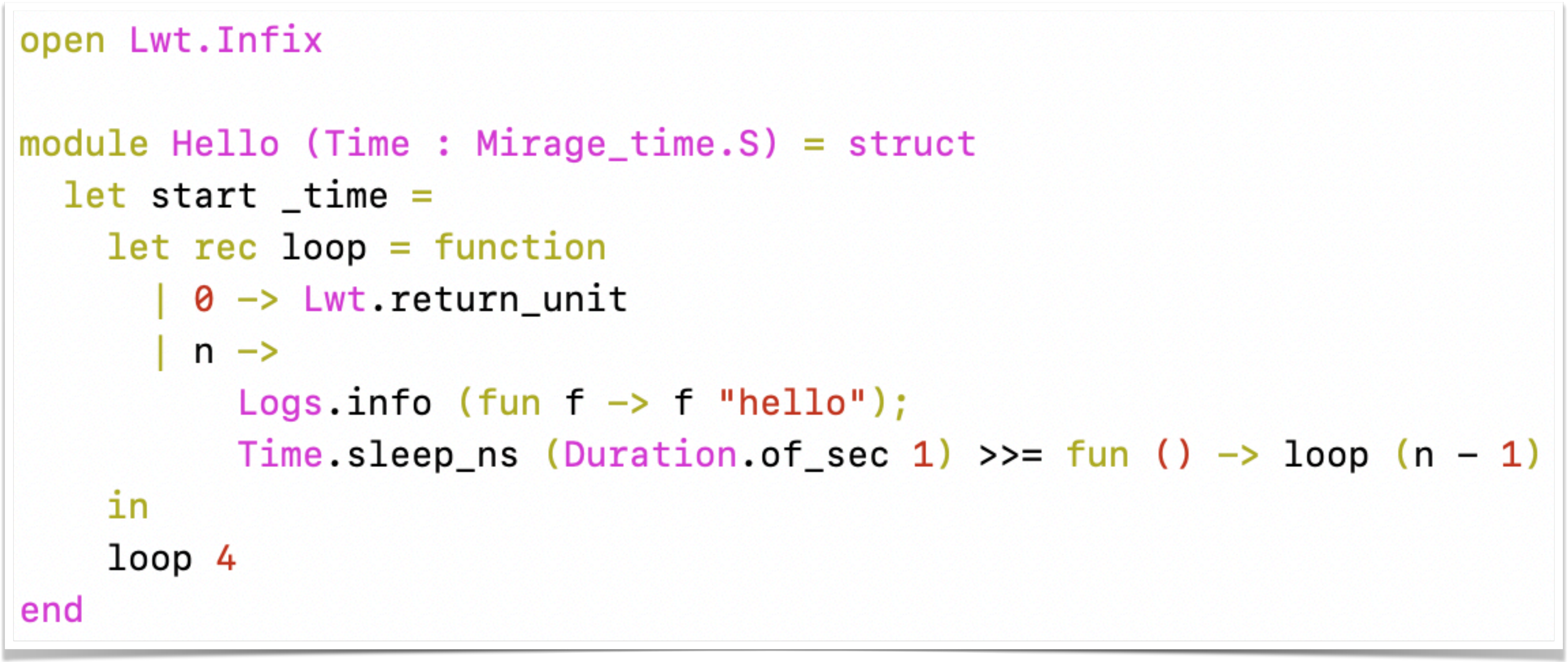
- A statically compiled ELF binary
- Executed as a virtual machine
  - Solo5 is the host system process (“tender”)
    - Provides the platform-specific details for MirageOS applications to interact with the underlying hardware or virtualisation frameworks
  - Supports — KVM, Xen, virtio, muen, Linux Seccomp
- Can also be executed as a Unix process
  - Useful for debugging and development

MirageOS Compiler

multi-stage pipeline



Hello Unikernel — unikernel.ml

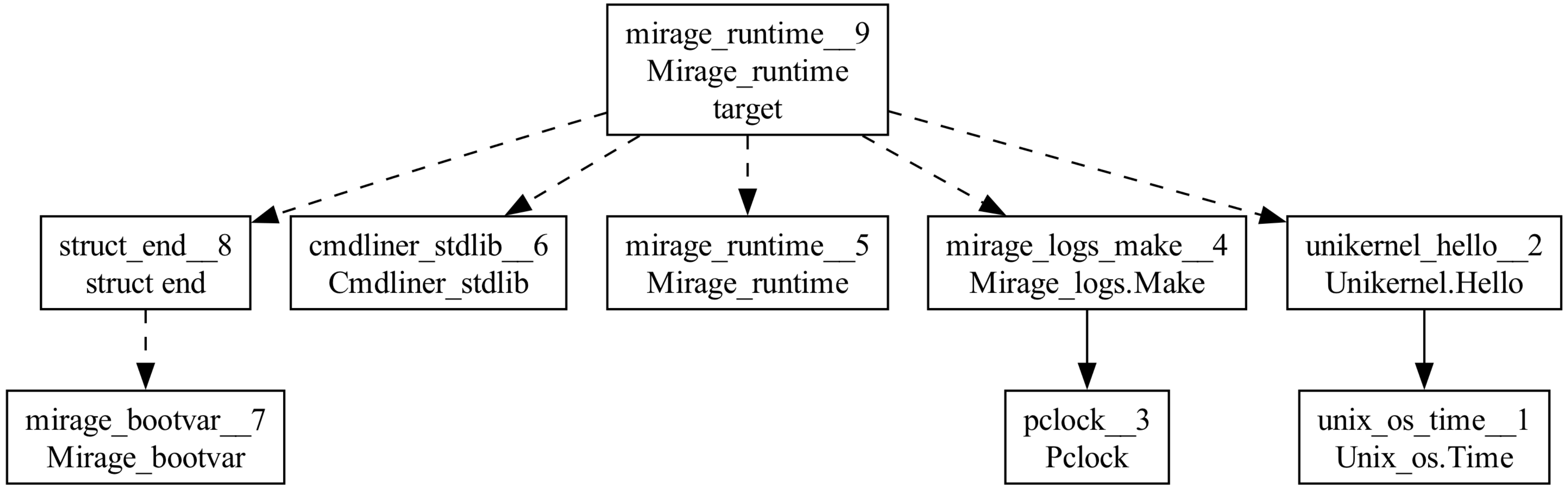


```ocaml
open Lwt.Infix

module Hello (Time : Mirage_time.S) = struct
  let start _time =
    let rec loop = function
      | 0 -> Lwt.return_unit
      | n ->
          Logs.info (fun f -> f "hello");
          Time.sleep_ns (Duration.of_sec 1) >>= fun () -> loop (n - 1)
    in
    loop 4
end
```

Hello Unikernel — Unix backend

```
$ mirage configure -t unix
$ make
$ ./dist/hello
2024-11-25T17:04:16+05:30: [INFO] [application] hello
2024-11-25T17:04:17+05:30: [INFO] [application] hello
2024-11-25T17:04:18+05:30: [INFO] [application] hello
2024-11-25T17:04:19+05:30: [INFO] [application] hello
```



mirage_runtime__9
Mirage_runtime
target

struct_end__8
struct end

cmdliner_stdlib__6
Cmdliner_stdlib

mirage_runtime__5
Mirage_runtime

mirage_logs_make__4
Mirage_logs.Make

unikernel_hello__2
Unikernel.Hello

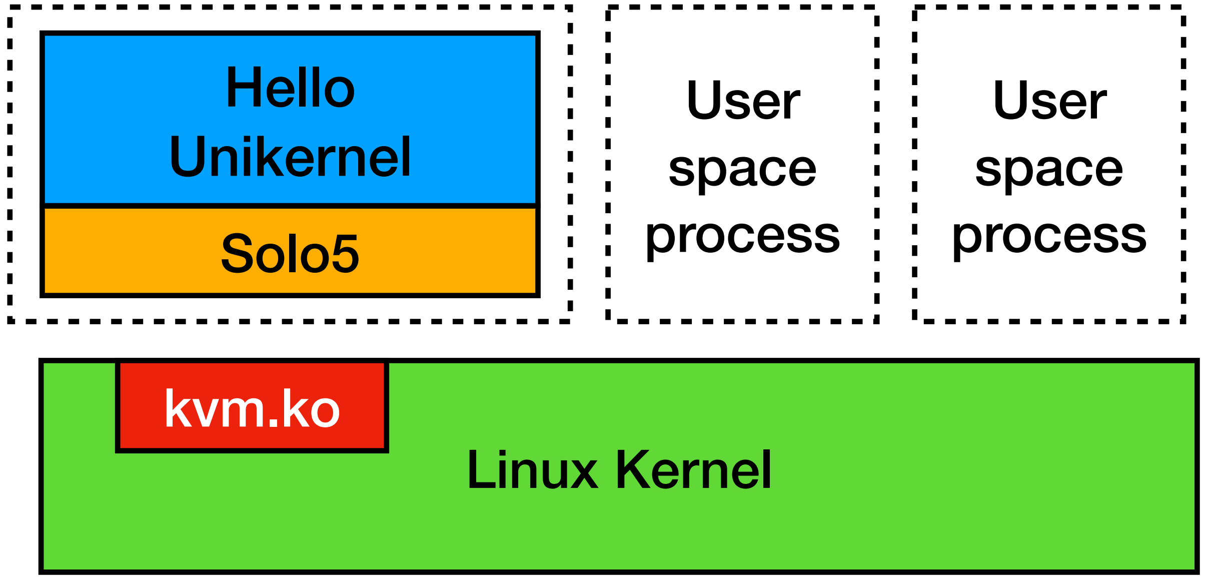
mirage_bootvar__7
Mirage_bootvar

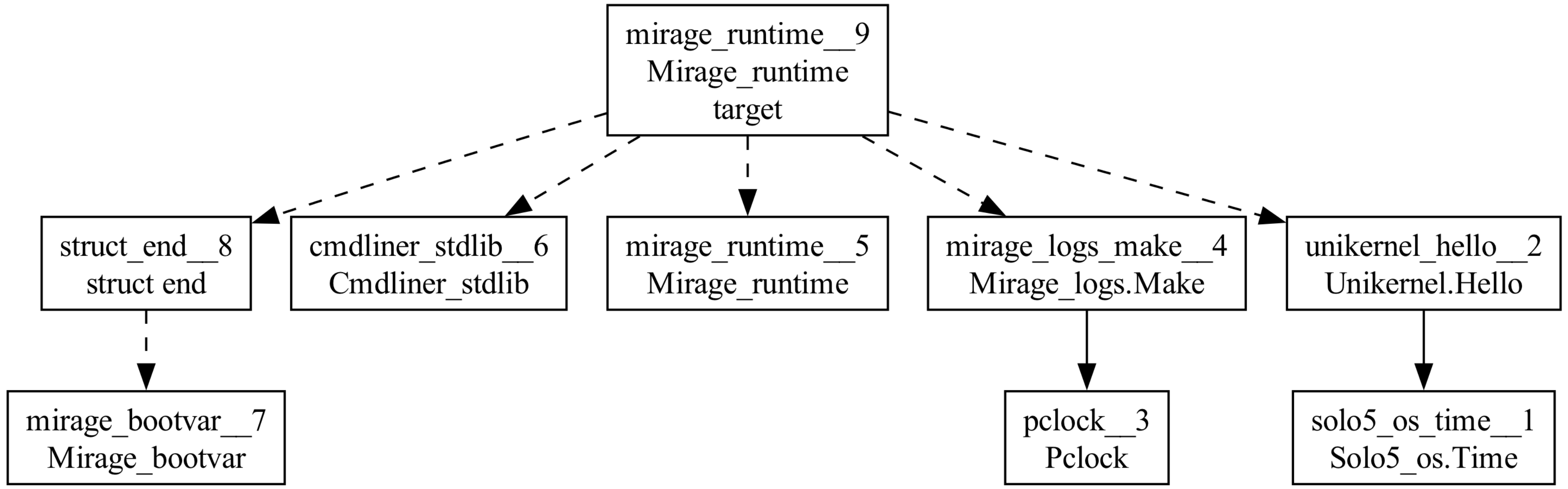
pclock__3
Pclock

unix_os_time__1
Unix_os.Time

Hello Unikernel — solo5-hvt on kvm

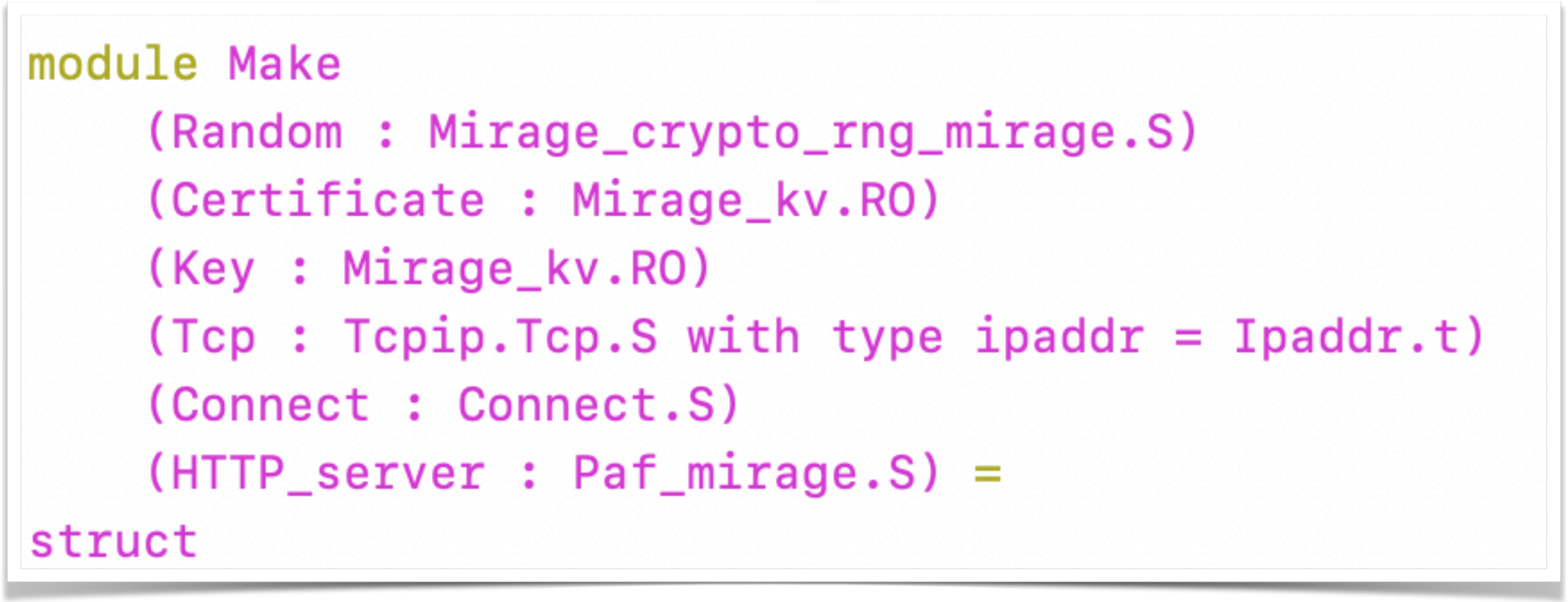
```
$ mirage configure -t hvt
$ make
$ solo5-hvt -- dist/hello.hvt
            |      ___|
  __|  _ \  |  _ \ __ \
\__ \ (   | | (   |  ) |
____/\___/ _|\___/____/
Solo5: Bindings version v0.9.0
Solo5: Memory map: 512 MB addressable:
Solo5:   reserved @ (0x0 - 0xfffff)
Solo5:       text @ (0x100000 - 0x1c4fff)
Solo5:     rodata @ (0x1c5000 - 0x1f5fff)
Solo5:       data @ (0x1f6000 - 0x289fff)
Solo5:       heap >= 0x28a000 < stack < 0x20000000
2024-11-25T11:47:10-00:00: [INFO] [application] hello
2024-11-25T11:47:11-00:00: [INFO] [application] hello
2024-11-25T11:47:12-00:00: [INFO] [application] hello
2024-11-25T11:47:13-00:00: [INFO] [application] hello
Solo5: solo5_exit(0) called
```



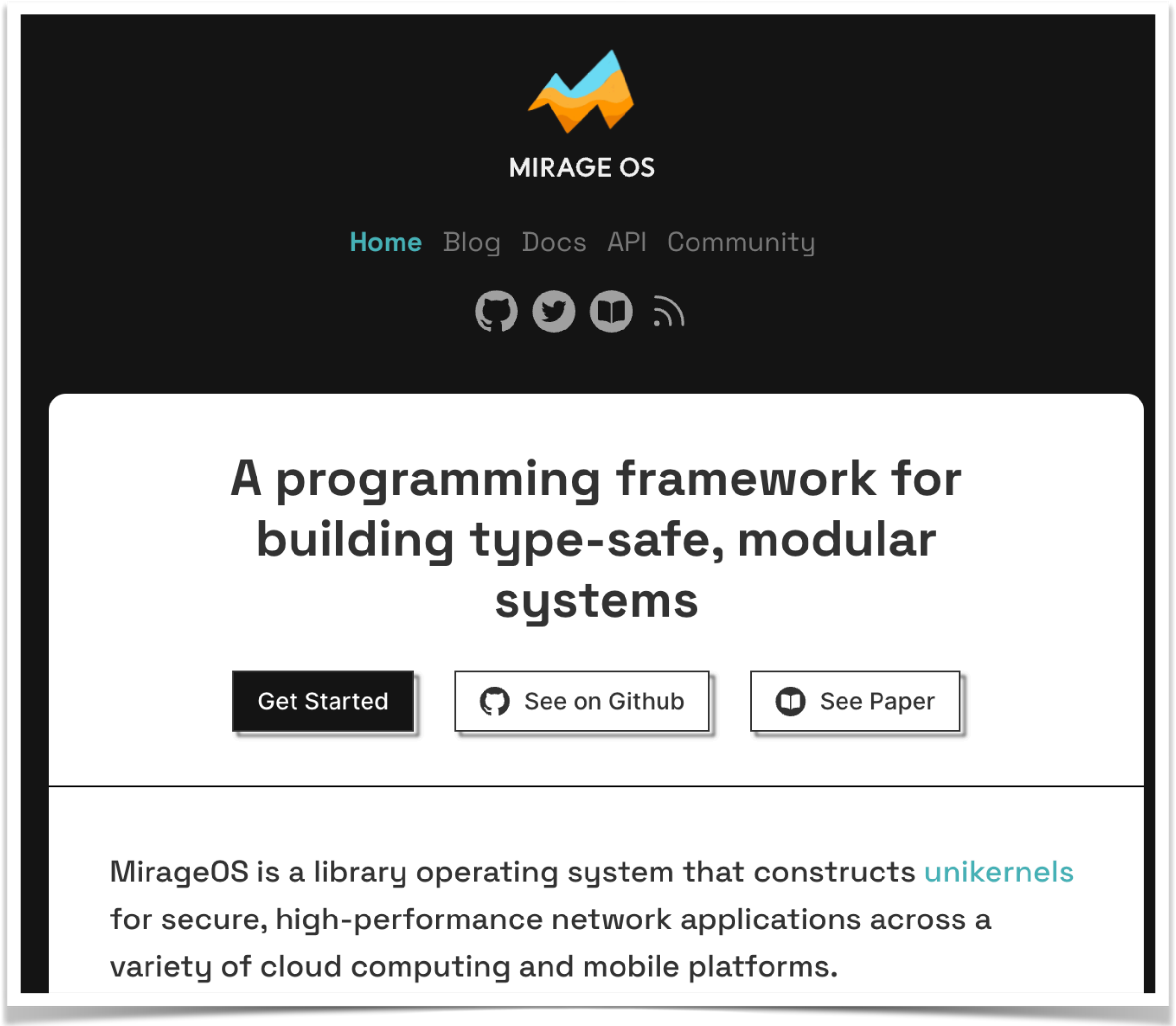


mirage.io website

- A full-fledged https server
- Uses TLS encryption



```
module Make
    (Random : Mirage_crypto_rng_mirage.S)
    (Certificate : Mirage_kv.RO)
    (Key : Mirage_kv.RO)
    (Tcp : Tcpip.Tcp.S with type ipaddr = Ipaddr.t)
    (Connect : Connect.S)
    (HTTP_server : Paf_mirage.S) =
struct
```

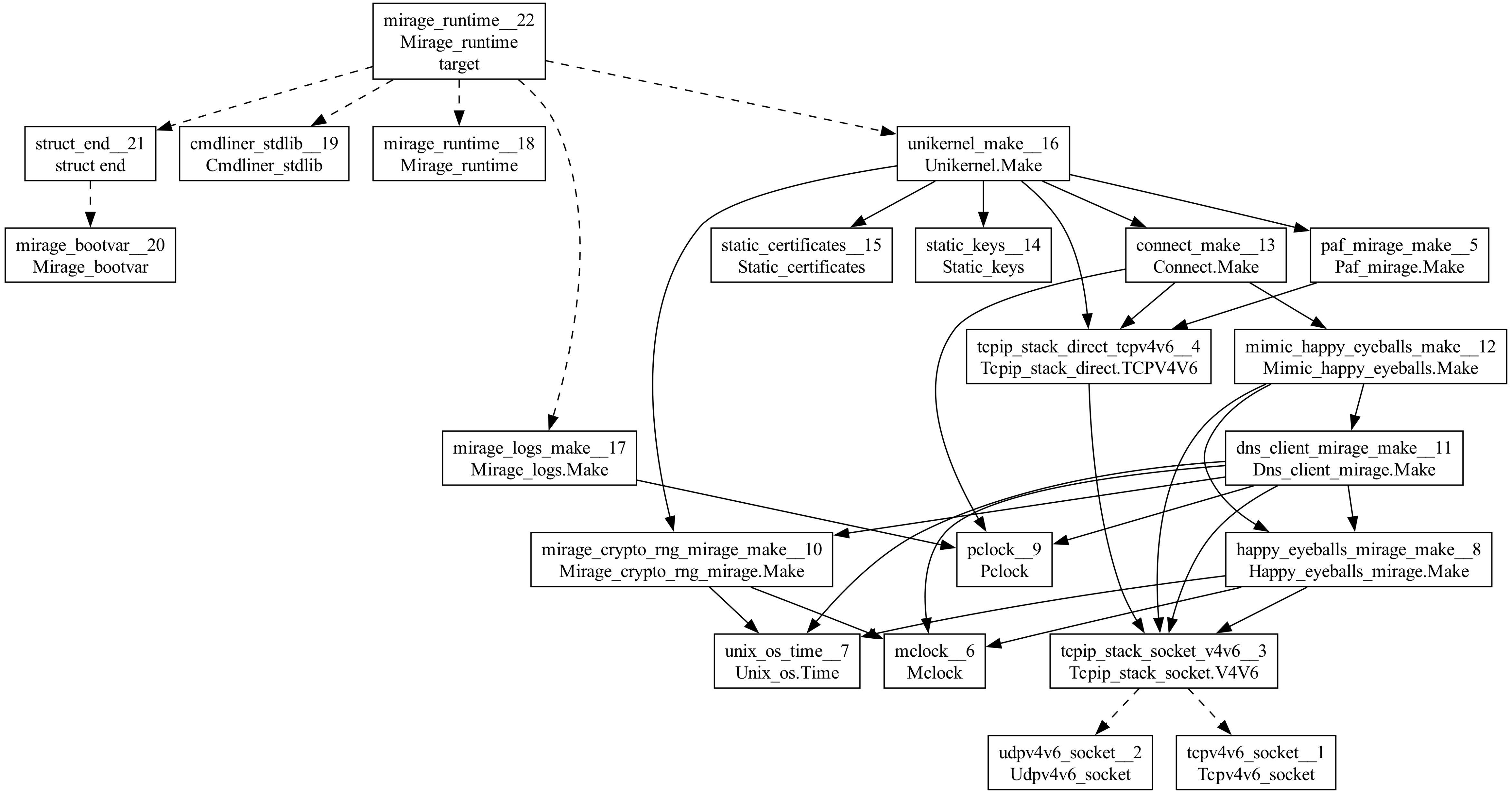


A programming framework for building type-safe, modular systems

MirageOS is a library operating system that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.

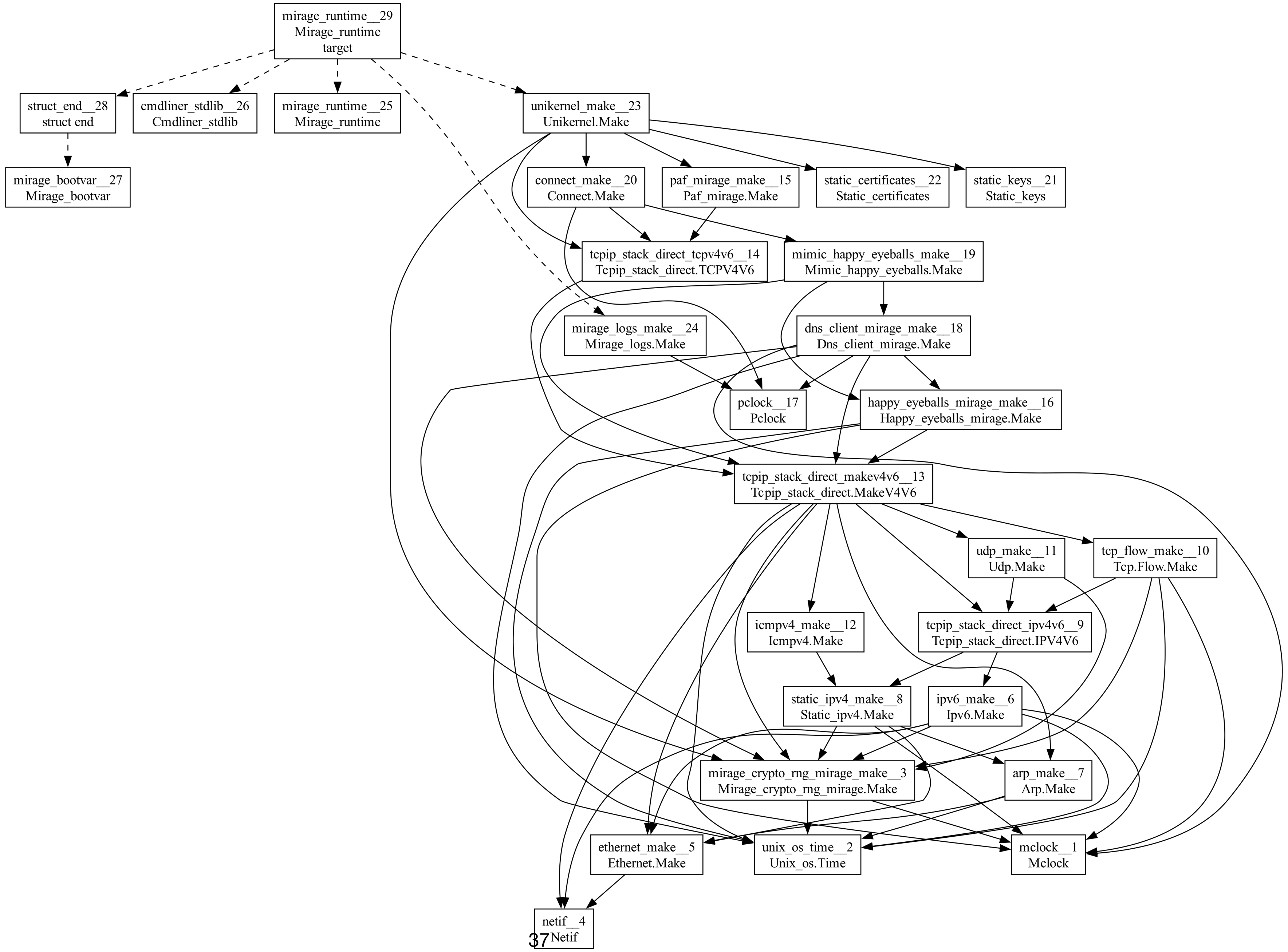
mirage.io website

```
$ mirage configure -t unix --net=host
```



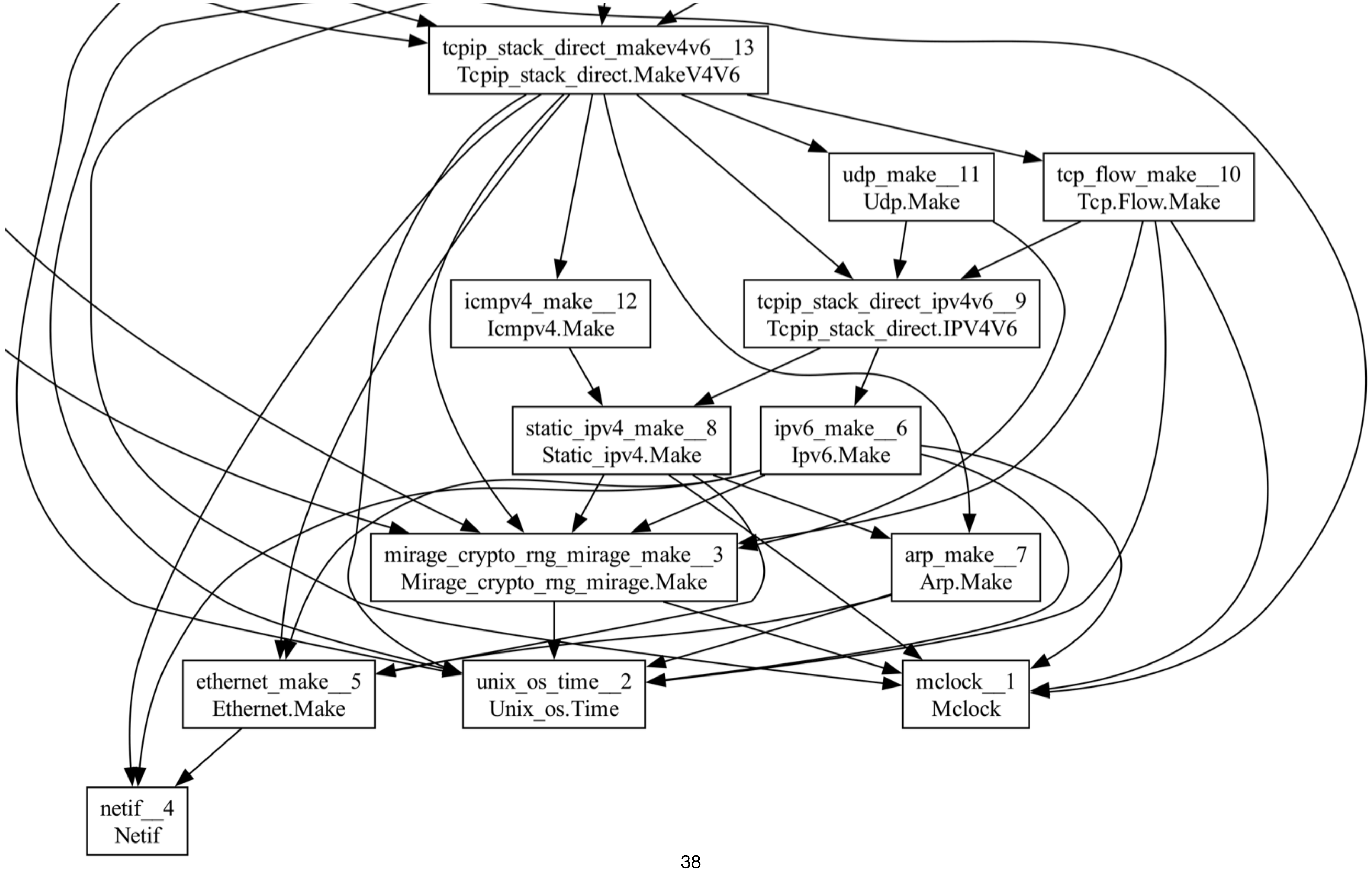
mirage.io website

```
$ mirage configure -t unix --net=direct
```



mirage.io website

```
$ mirage configure -t unix --net=direct
```



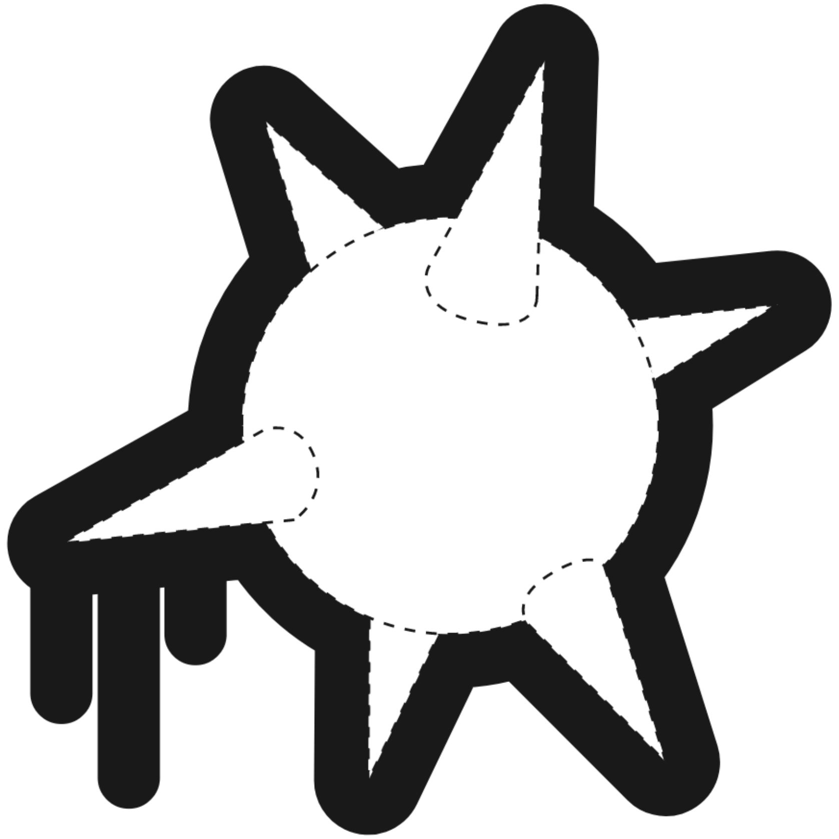
MirageOS Compiler

- Remove dead code and inline code across traditionally opaque layer
  - Resulting images usually have a size of **a few MiB**.
  - Our HTTPS web server which runs mirage.io is **only 10 MiB!**
- Configuration can be partially evaluated at compile-time
  - Extreme specialisation enables a **boot time of a few ms**.
- If something (e.g. networking) is not used, it will not be available at runtime
  - Minimal runtime environments use **a few MiB of RAM**.
- The kernel and user space share the same address space
  - Many runtime checks are removed, so **static safety** is critical.

MirageOS Usecases

Bitcoin Piñata

- https://hannes.robur.coop/Posts/Pinata
- 1.1 MB Unikernel, which ran from 2015 to 2018
- Hold the key to 10 bitcoins (peak worth $165k)
  - Now worth ~$1M
- A successful authenticated TLS session reveals the private Bitcoin key
- 500,000 accesses to the Piñata website, more than 150,000 attempts at connecting to the Piñata bounty
- The bitcoins were safe!



Nitrokey NetHSM

- NitroKey is developing NetHSM, a new HSM solution to manage cryptographic keys securely.
- Aim for high-performance, low-power, customizability and high-security
    - Open-source ⟹ auditable by anyone
- They chose to use **MirageOS** running on the **Muen** micro-kernel

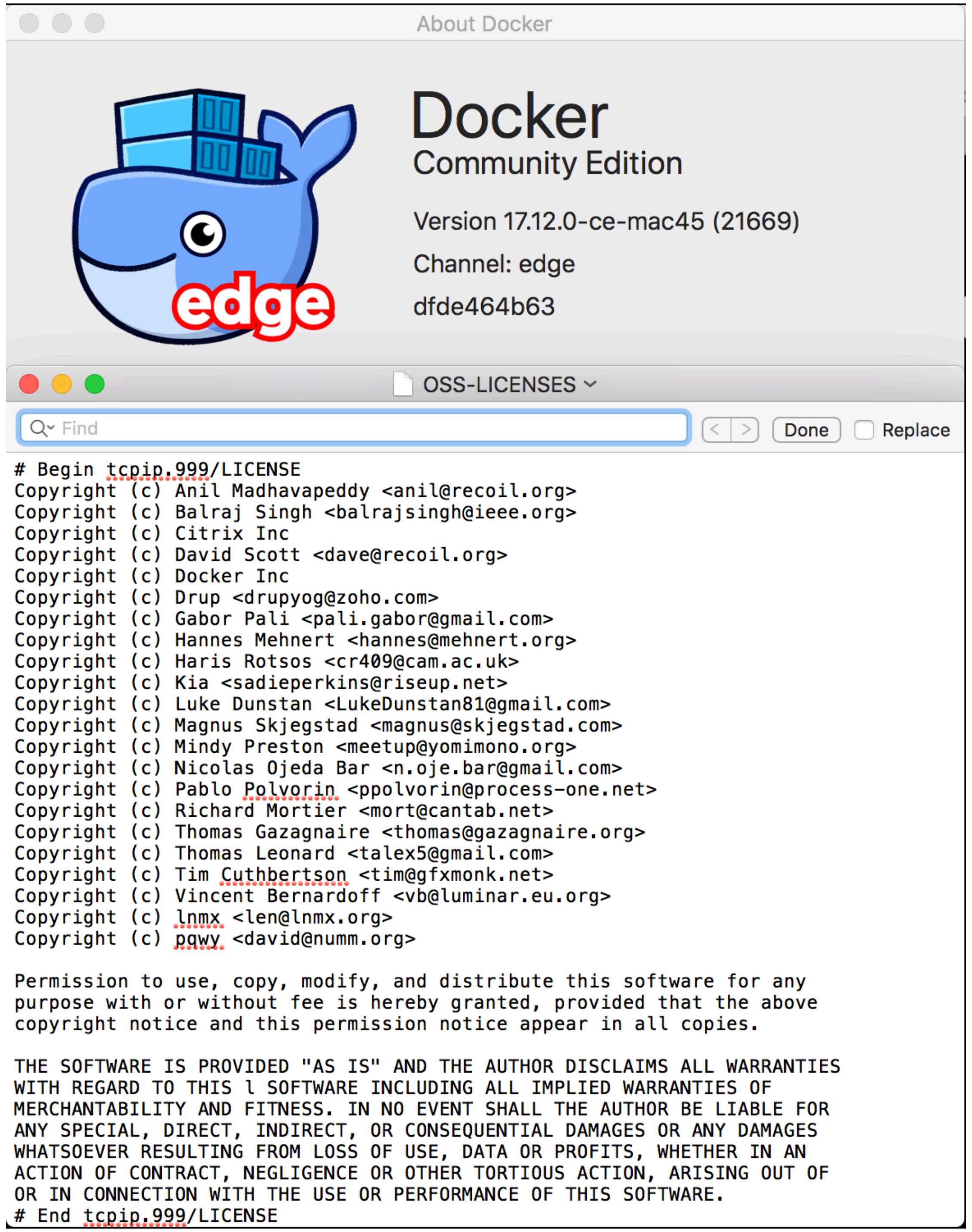
NetHSM - The Trustworthy, Open Hardware Security Module That Just Works

https://www.nitrokey.com/products/nethsm

Docker for Mac

MirageOS libraries used by millions of users

- Normally Docker use Linux namespaces and other Linux features
- On macOS
  - Docker daemon runs in a light Linux VM (using hypervisor.framework)
  - Docker client is a Mac application
- MirageOS libraries are used to translate semantics differences between platforms:
  - **volumes:** FUSE format + fsevent/inotify
  - **network:** Linux ethernet packets to MacOS network syscalls



Docker
Community Edition

Version 17.12.0-ce-mac45 (21669)

Channel: edge

dfde464b63

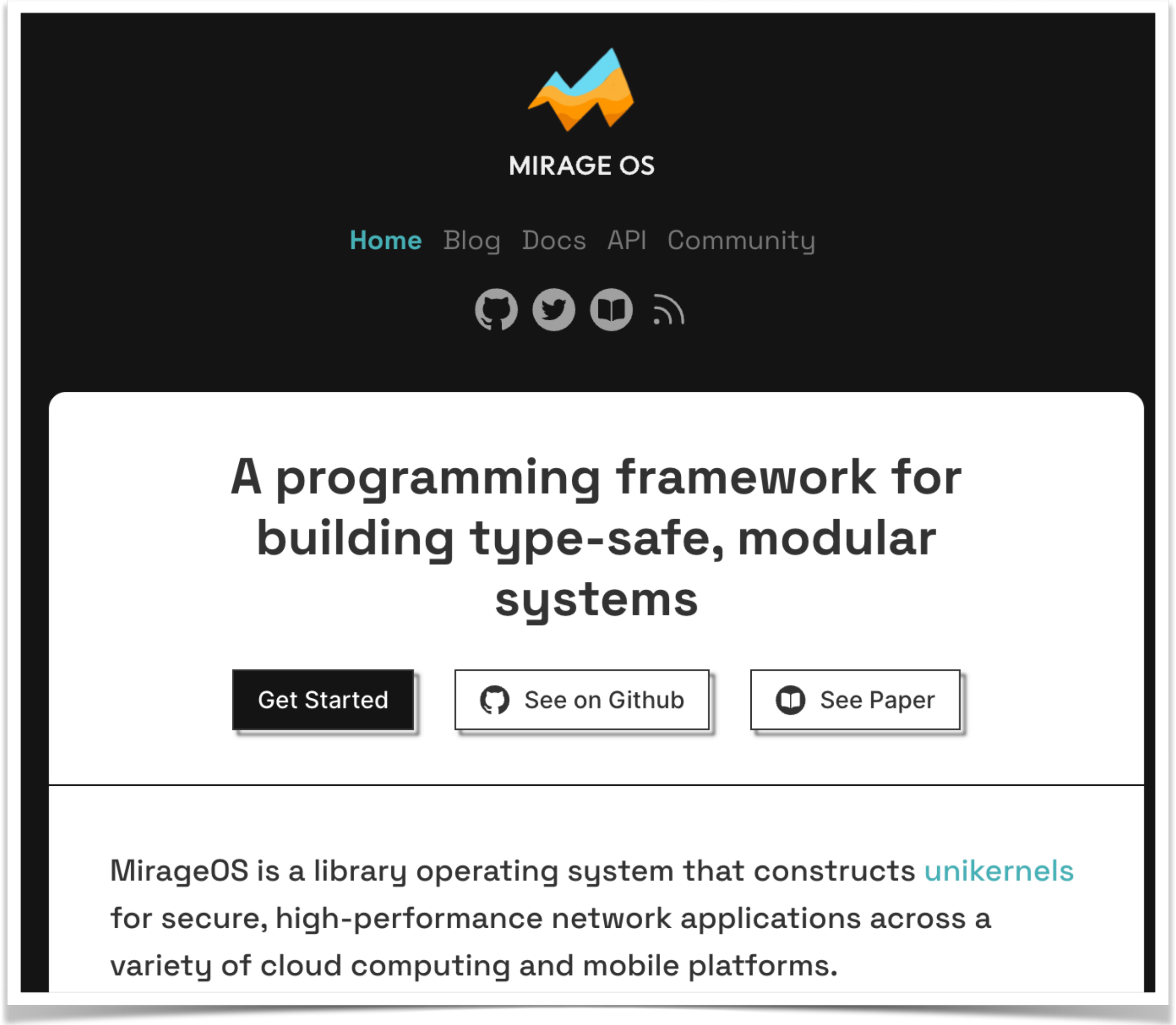
```
# Begin tcpip.999/LICENSE
Copyright (c) Anil Madhavapeddy <anil@recoil.org>
Copyright (c) Balraj Singh <balrajsingh@ieee.org>
Copyright (c) Citrix Inc
Copyright (c) David Scott <dave@recoil.org>
Copyright (c) Docker Inc
Copyright (c) Drup <drupyog@zoho.com>
Copyright (c) Gabor Pali <pali.gabor@gmail.com>
Copyright (c) Hannes Mehnert <hannes@mehnert.org>
Copyright (c) Haris Rotsos <cr409@cam.ac.uk>
Copyright (c) Kia <sadieperkins@riseup.net>
Copyright (c) Luke Dunstan <LukeDunstan81@gmail.com>
Copyright (c) Magnus Skjegstad <magnus@skjegstad.com>
Copyright (c) Mindy Preston <meetup@yomimono.org>
Copyright (c) Nicolas Ojeda Bar <n.oje.bar@gmail.com>
Copyright (c) Pablo Polvorin <ppolvorin@process-one.net>
Copyright (c) Richard Mortier <mort@cantab.net>
Copyright (c) Thomas Gazagnaire <thomas@gazagnaire.org>
Copyright (c) Thomas Leonard <talex5@gmail.com>
Copyright (c) Tim Cuthbertson <tim@gfxmonk.net>
Copyright (c) Vincent Bernardoff <vb@luminar.eu.org>
Copyright (c) lnmx <len@lnmx.org>
Copyright (c) pqwy <david@numm.org>

Permission to use, copy, modify, and distribute this software for any
purpose with or without fee is hereby granted, provided that the above
copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
WITH REGARD TO THIS l SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
# End tcpip.999/LICENSE
```

MirageOS

You were here!



A programming framework for building type-safe, modular systems

MirageOS is a library operating system that constructs unikernels for secure, high-performance network applications across a variety of cloud computing and mobile platforms.

https://mirage.io