Unikernels @Docker

Thomas Gazagnaire
28 Juin 2017

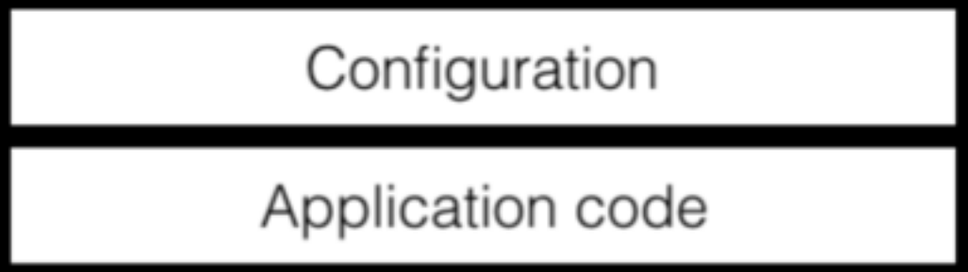
About me

- Core team of MirageOS
- Co-founder of “Unikernel Systems”
- Now work at Docker

Legacy Applications

Legacy Applications

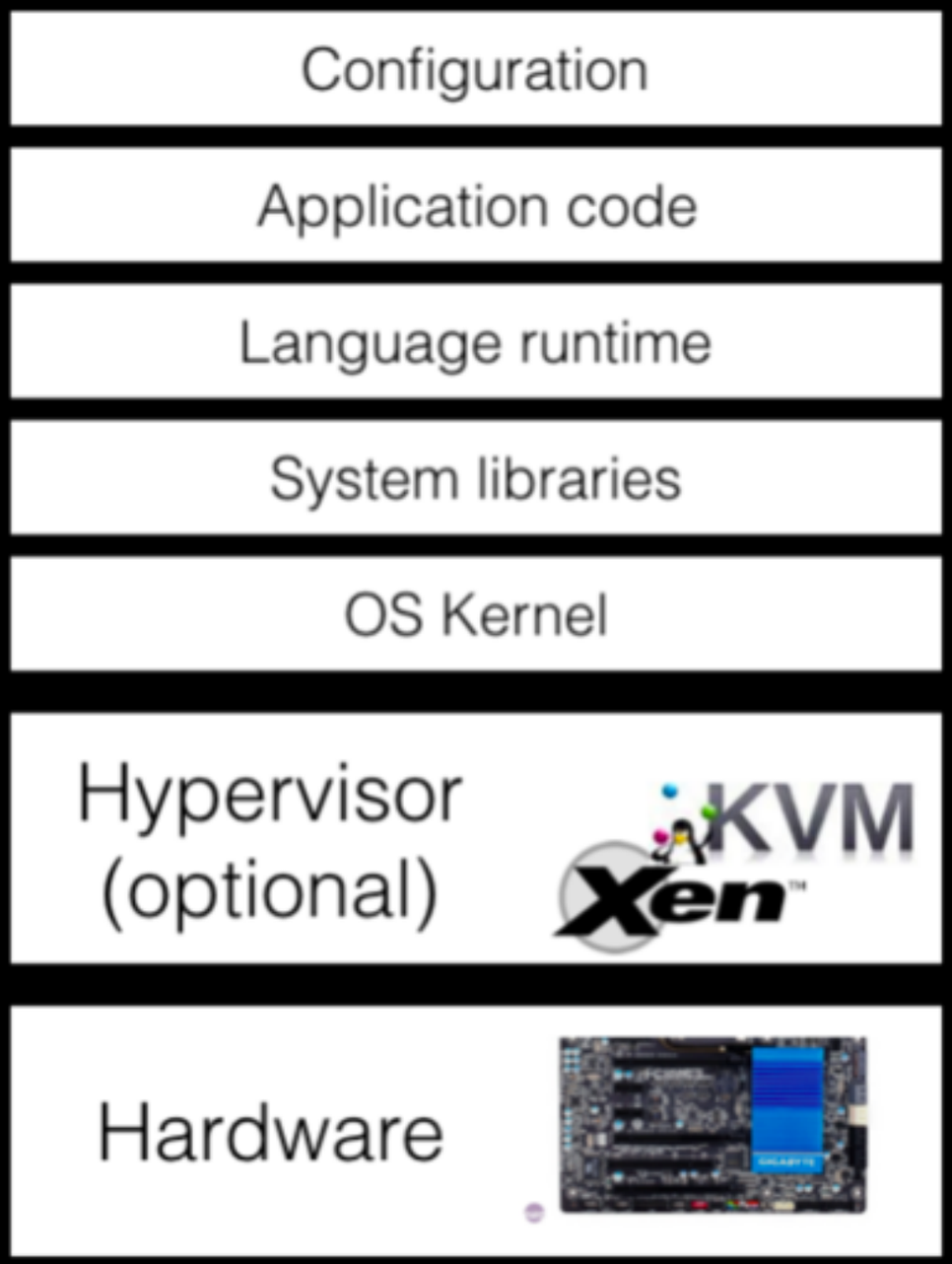
Traditional software stack



- your nice application

Legacy Applications

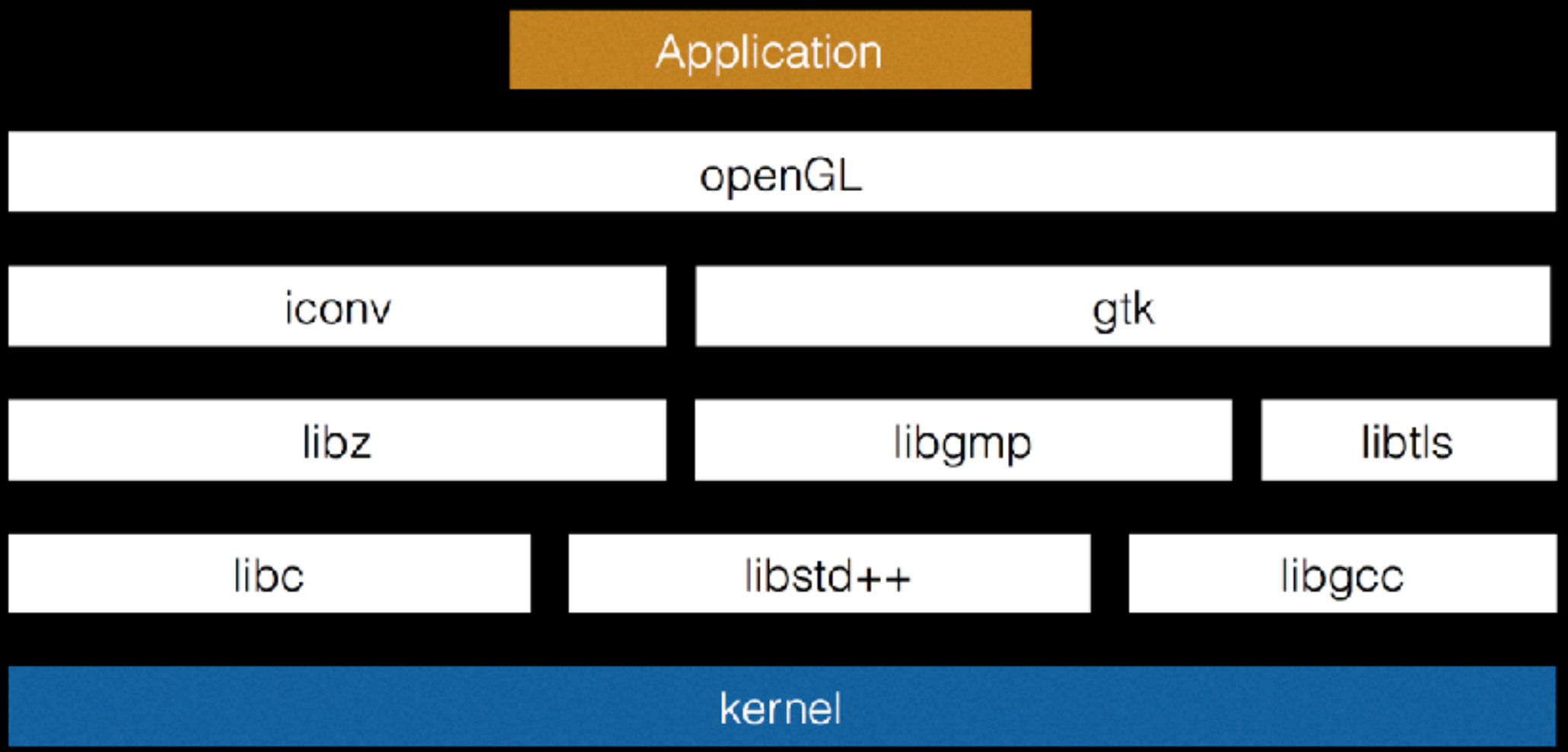
Traditional software stack



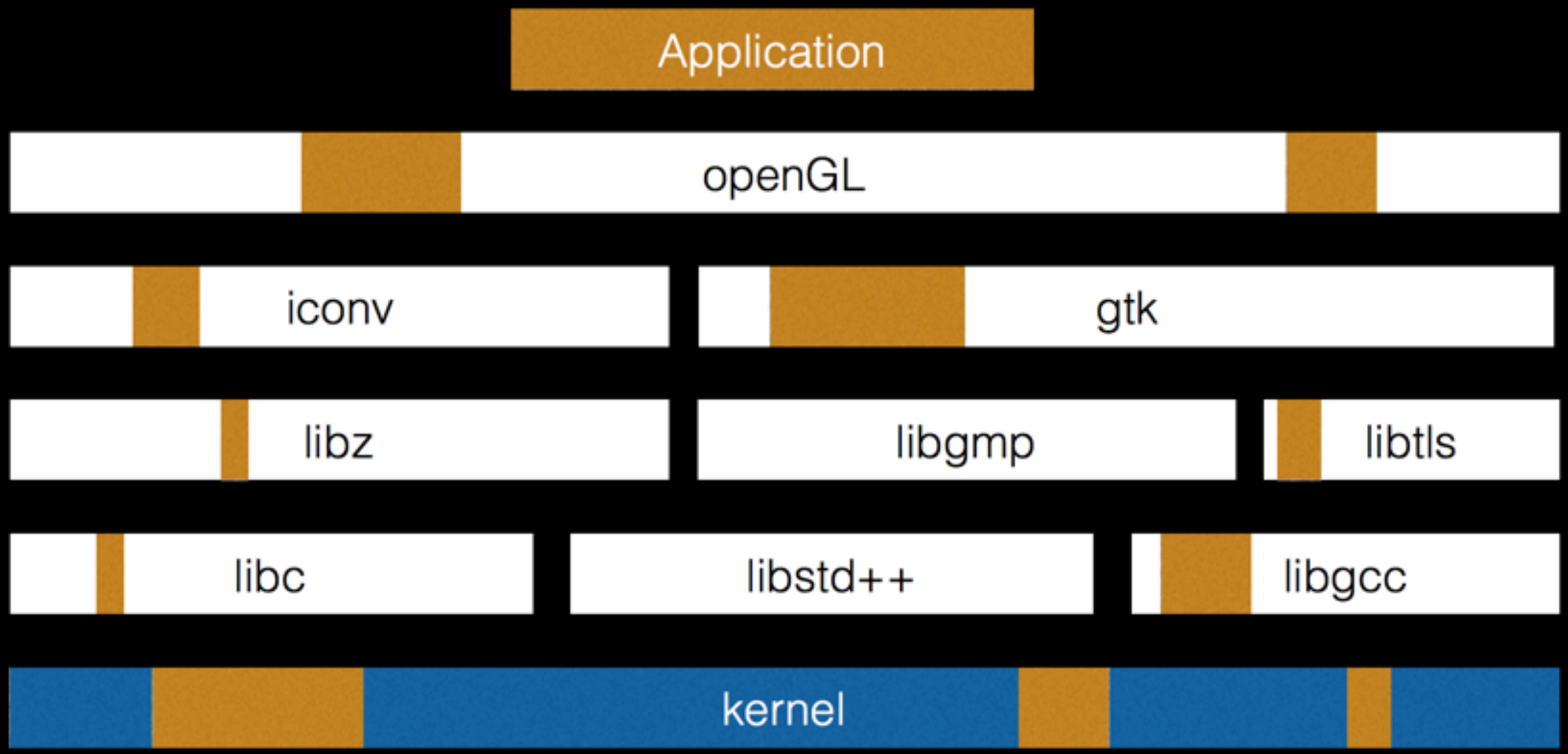
- your nice application

- “Legacy OS” layers:
  - multiple processes
  - multiple purposes
  - multiple users
  - multiple hardware platforms

Legacy Applications



Legacy Applications

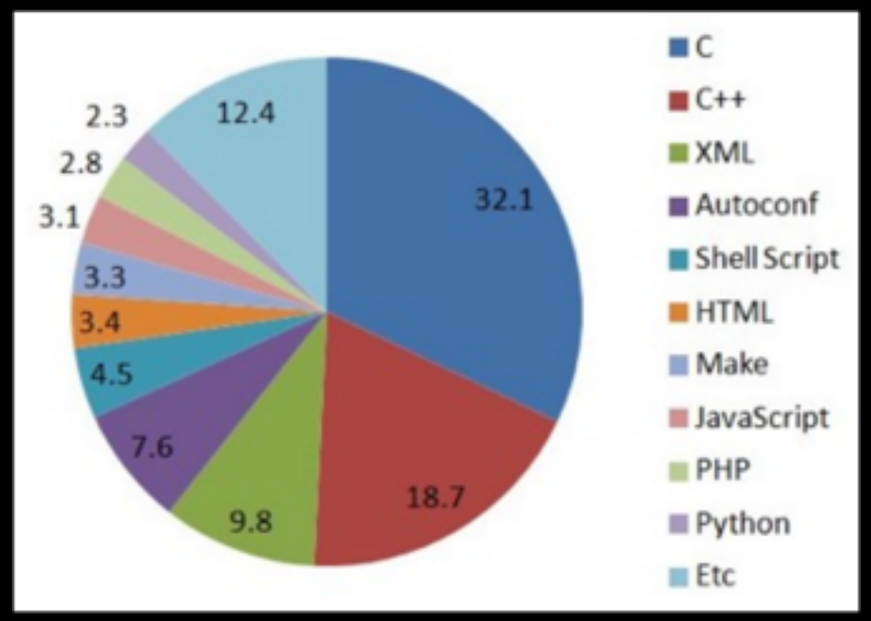


Legacy Applications

(unsafe) code bloat:

- Linux kernel: 25 millions of loc
- Windows kernel: 50 millions of loc
- Debian 5.0: 65 millions of loc
- OSX 10.4: 85 millions of loc

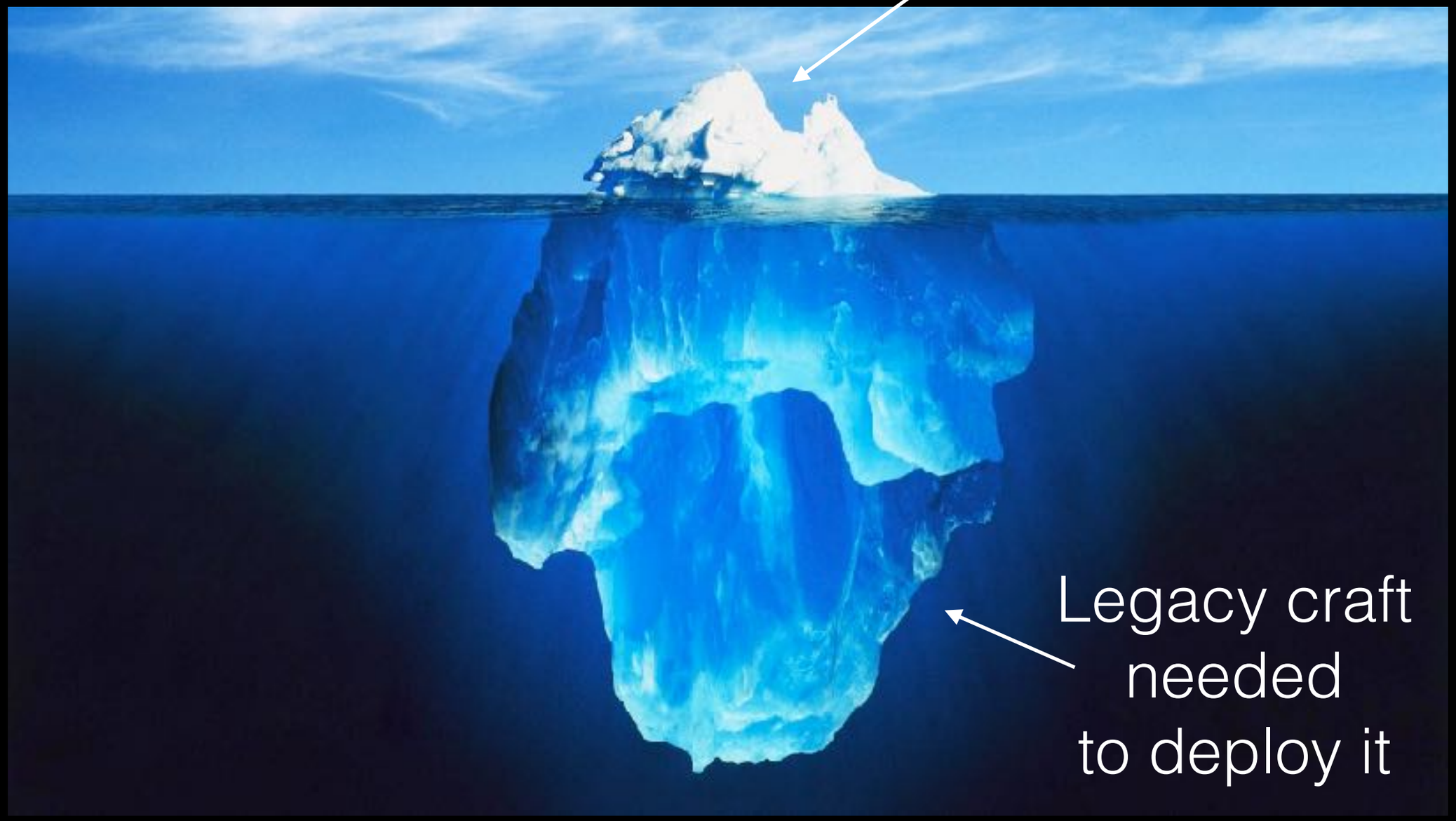
Debian in 2013



OpenSSL

- 500k of C
- used by 2/3 of web servers
- 23 CVE in 2014
- 31 CVE in 2015
- 34 CVE in 1016

Legacy Applications



LinuxKit

https://github.com/linuxkit/linuxkit

Immutable Delivery

"In the cloud, we know exactly what we want a server to be, and if we want to change that we simply terminate it and launch a new server with a new AMI."

Netflix Building with Legos, 2011

linuxkit

Immutable Delivery

“As a system administrator, one of the scariest things I ever encounter is a server that’s been running for ages.

If you absolutely know a system has been created via automation and never changed since the moment of creation, most of the problems disappear.”

Chad Fowler,Trash Your Servers and Burn Your Code, 2013

Built for Docker Editions

first desktop then cloud

immutable delivery was what we needed for reliability

- could not find an existing solution
- iterated since 2015
- found a design that is useful for others
- time to open source and get community input

linuxkit

Requirements

- batteries included, but removable
- fast to build, fast to boot
- build whole system in your CI pipeline
- best-of-breed security technologies by default
- immutable in production
- designed to be managed by external tooling
- container native, cloud native

linuxkit

Design Philosophy

specialise at build time, not run time

existing distributions tend to do things at boot-time which increases the image size and complexity.

- the “unikernel” approach is to highly specialise a deployment based on the application being deployed.

- so we applied this approach to building Linux.

- what if **everything** in the booting image was specified in **one** file and built as easily as “docker build”?

linuxKit

Secure Defaults

which can be replaced

The project provides the base containers to get started, with an emphasis on minimalism and security

- you only need a few containers
- enough to bootstrap distributed applications
- security project incubation

linuxkit

Community of Contributors

48 Contributors/22 External, 3500 commits

yaml file defines boot image

The config file defines the whole system

- kernel
- boot scripts
- config containers
- service containers

Also defines what to output: ISOs, AMIs etc

yaml config file

```
kernel:
  image: "linuxkit/kernel:4.9.x"
  cmdline: "console=ttyS0 console=tty0 page_poison=1"
init:
  - linuxkit/init
  - linuxkit/runc
  - linuxkit/containerd
onboot:
...
services:
...
```

yaml config file

```
services:
  - name: nginx
    image: "nginx:alpine"
    capabilities:
     - CAP_NET_BIND_SERVICE
     - CAP_CHOWN
     - CAP_SETUID
     - CAP_SETGID
     - CAP_DAC_OVERRIDE
  net: host
```

Demo

LinuxKit on macOS

linuxKit

Demo

LinuxKit on GCP

linuxkit

Demo

LinuxKit on packet.net

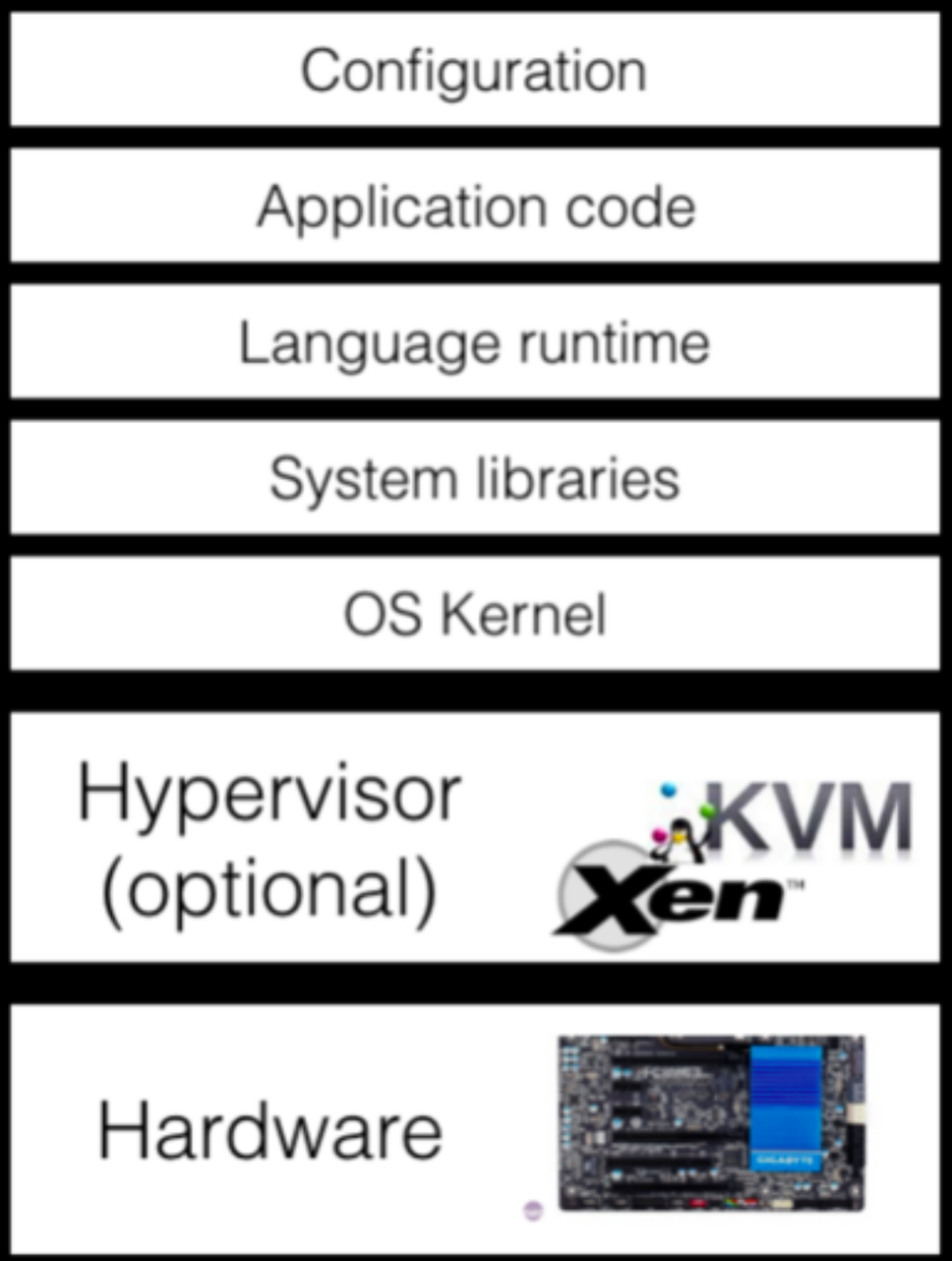
Unikernels

(mainly MirageOS)

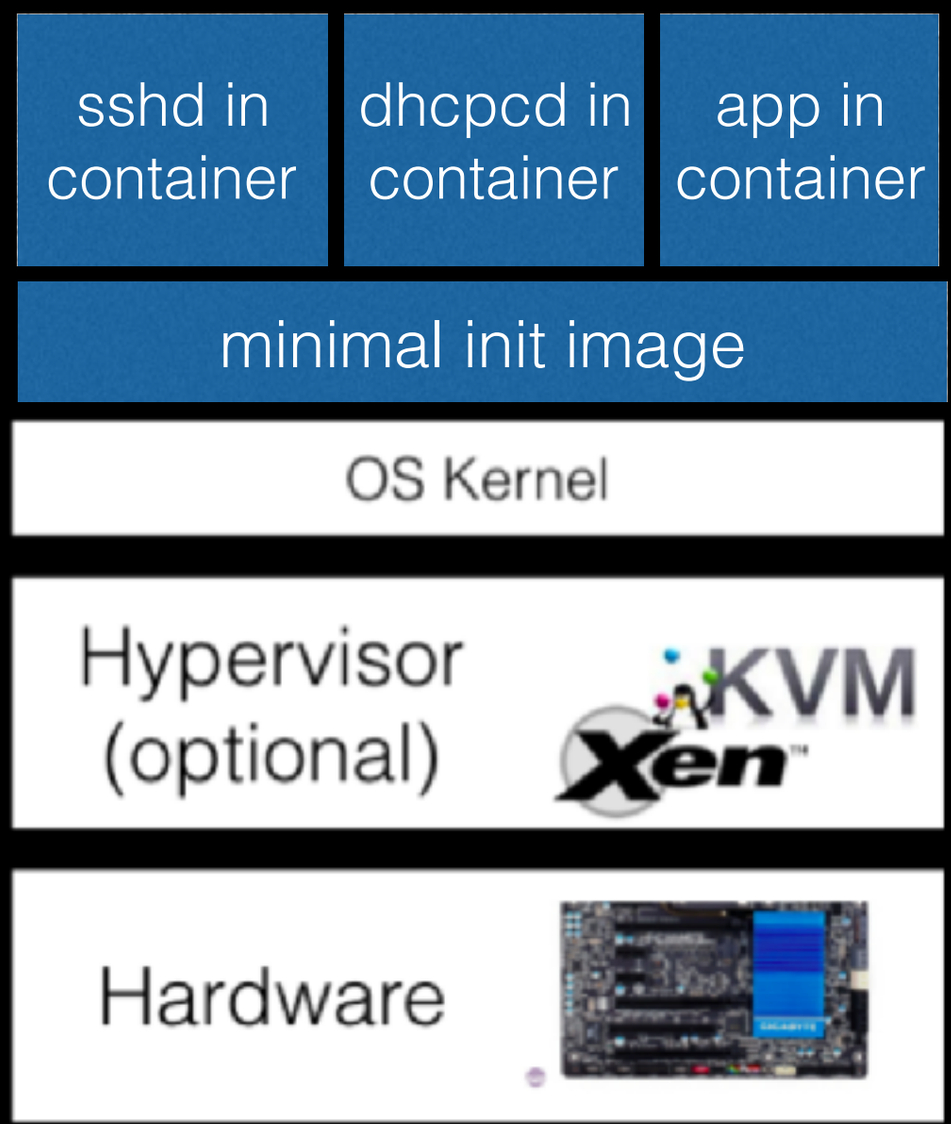
https://github.com/mirage/mirage

Unikernels

Traditional software stack

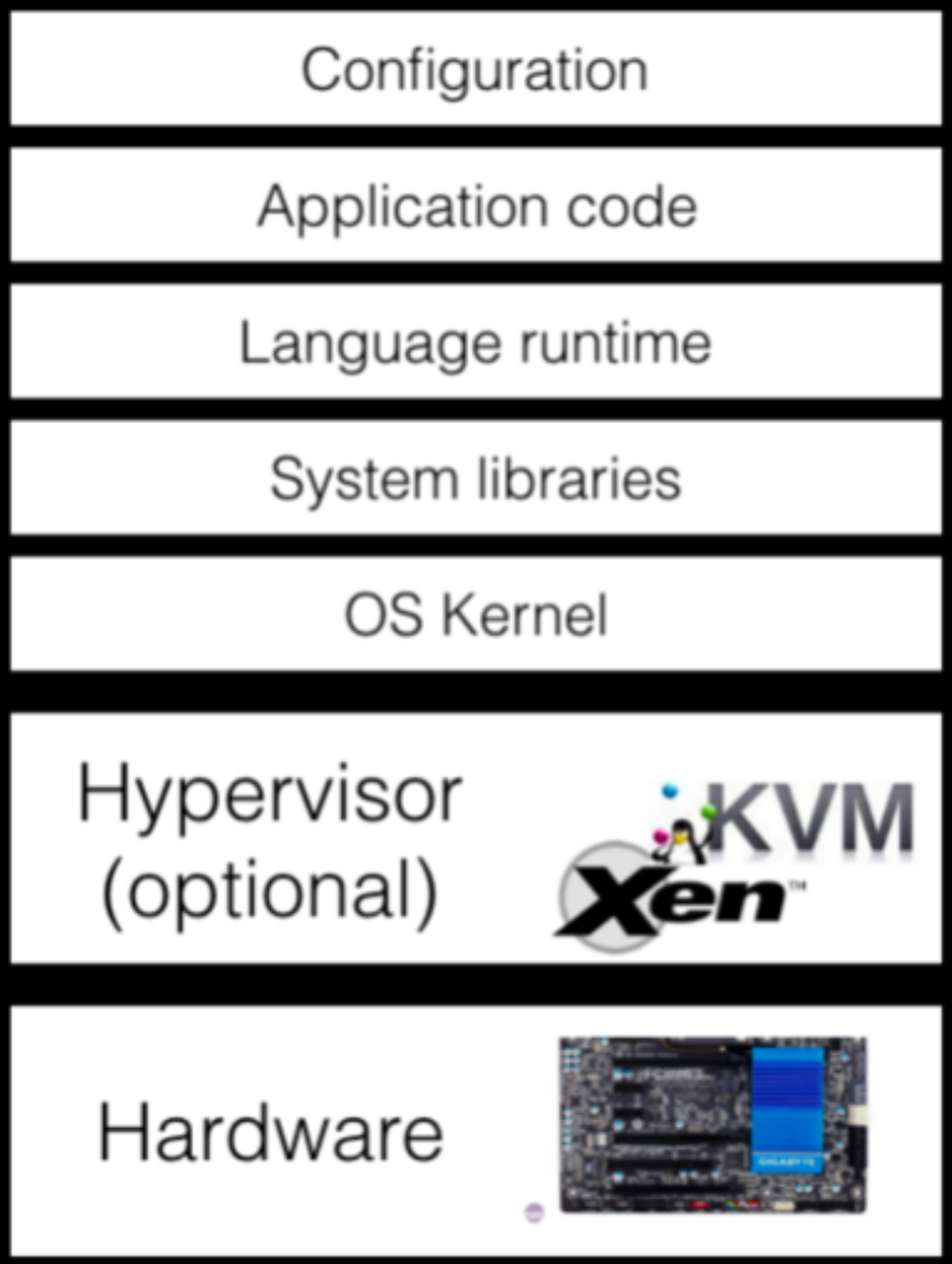


LinuxKit

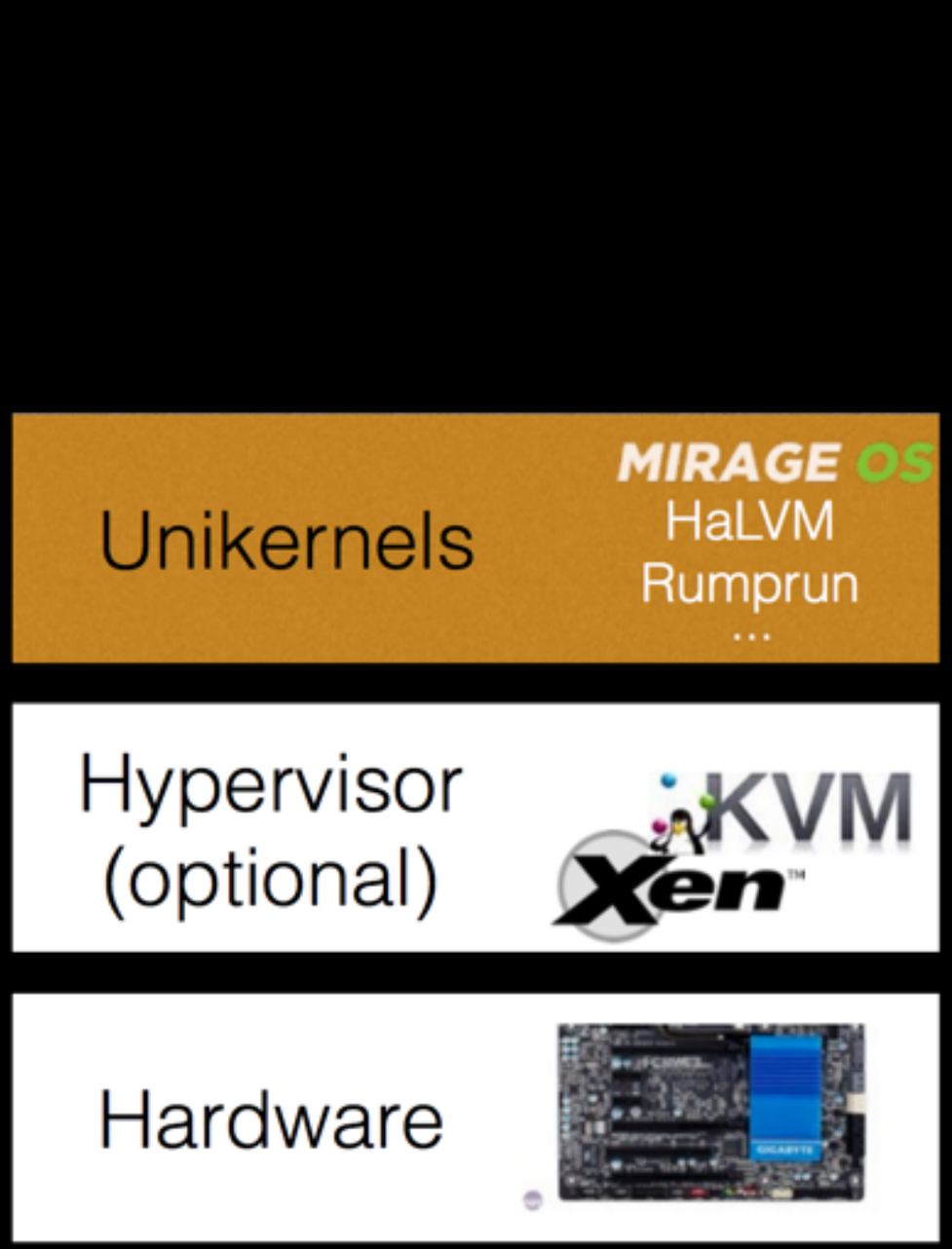


Unikernels

Traditional software stack



Unikernels



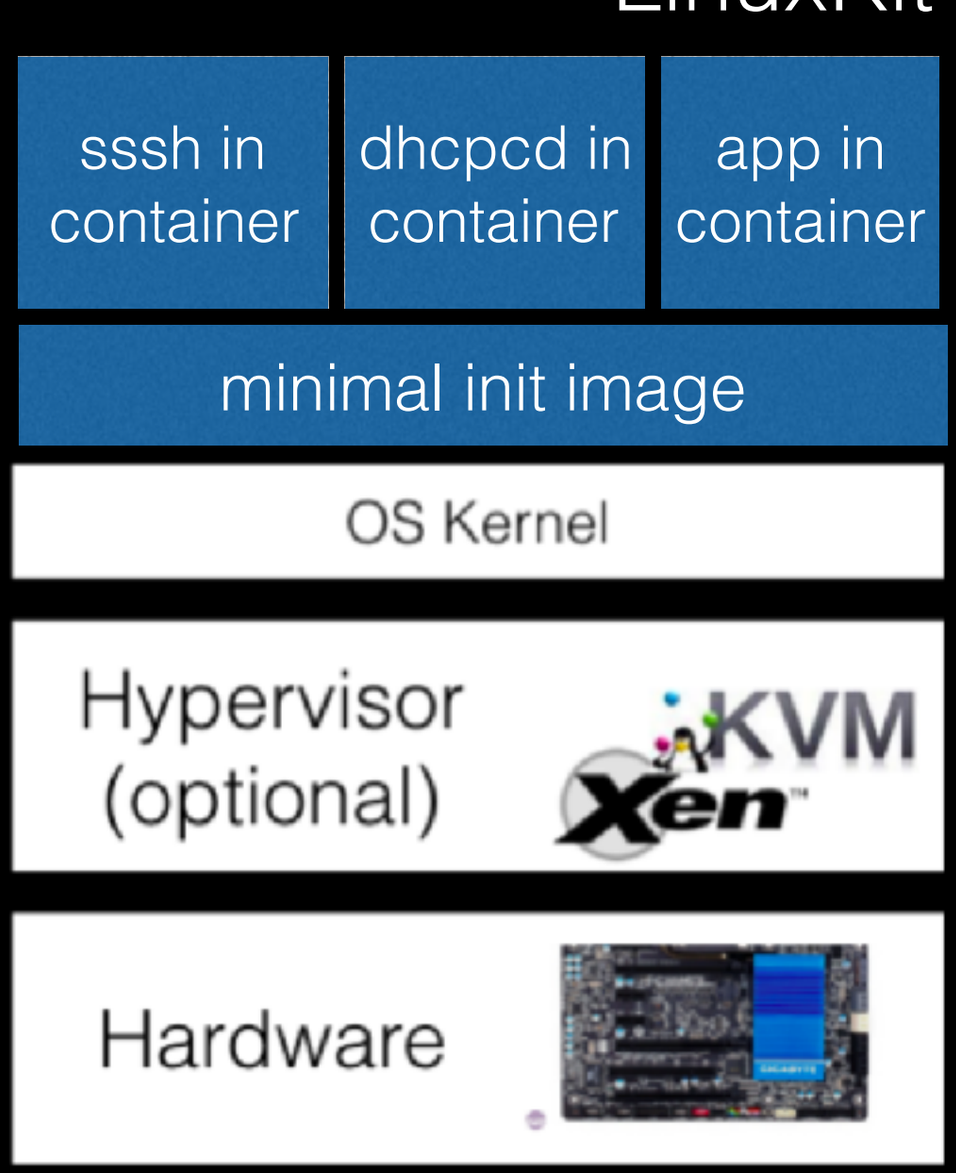
Unikernels

At build time:

- everything (including system services) are packaged into (minimal) **container images**, based on Alpine.
- **Use an external tool (moby)** to pull in only the necessary containers.
- Fix the deployment target: VM, bare-metal

At runtime:

- Strong isolation between processes
- Read-only filesystem for containers.
- RPC
- No package manager in the init image: by default it just contains containerd and runc.



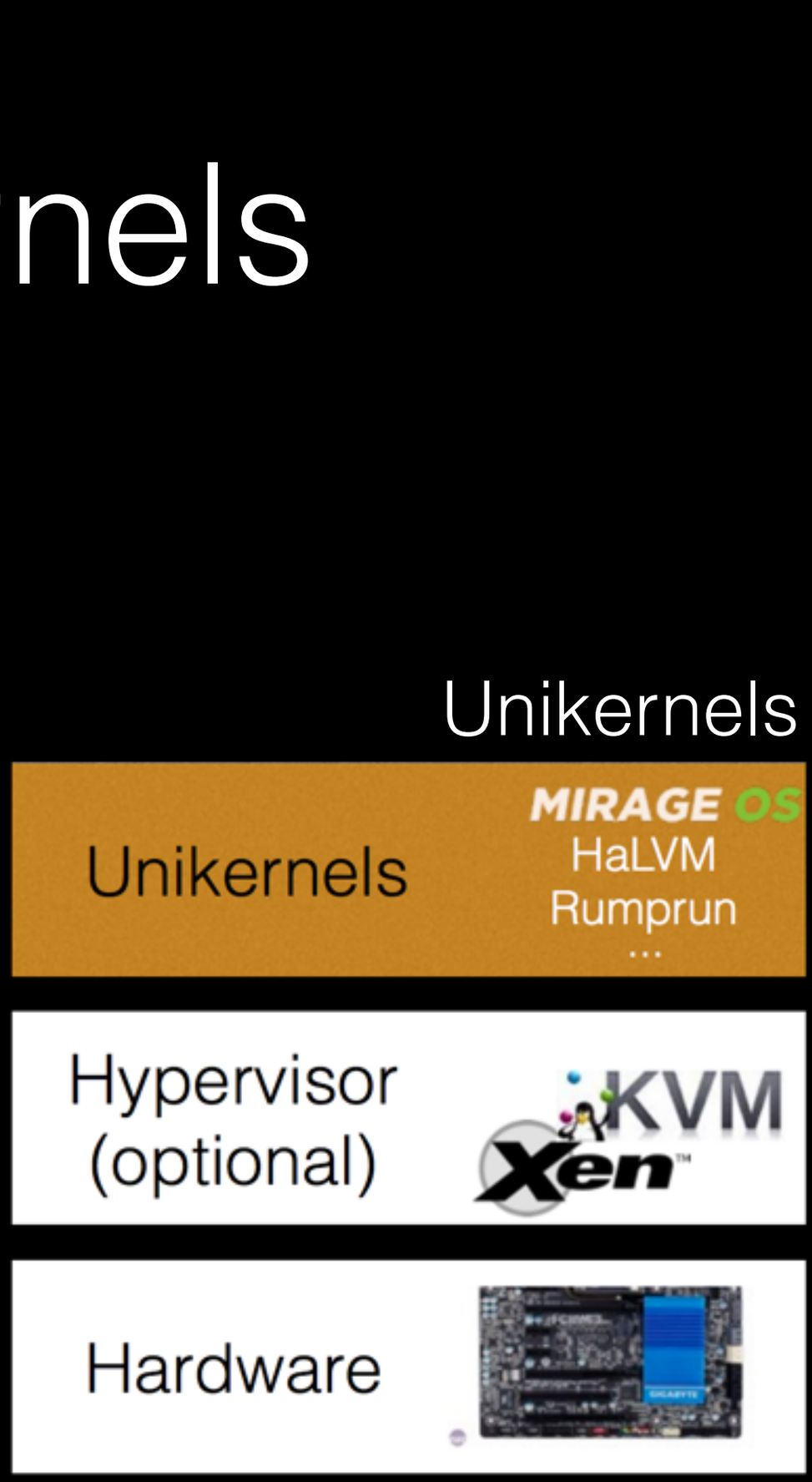
Unikernels

At build time:

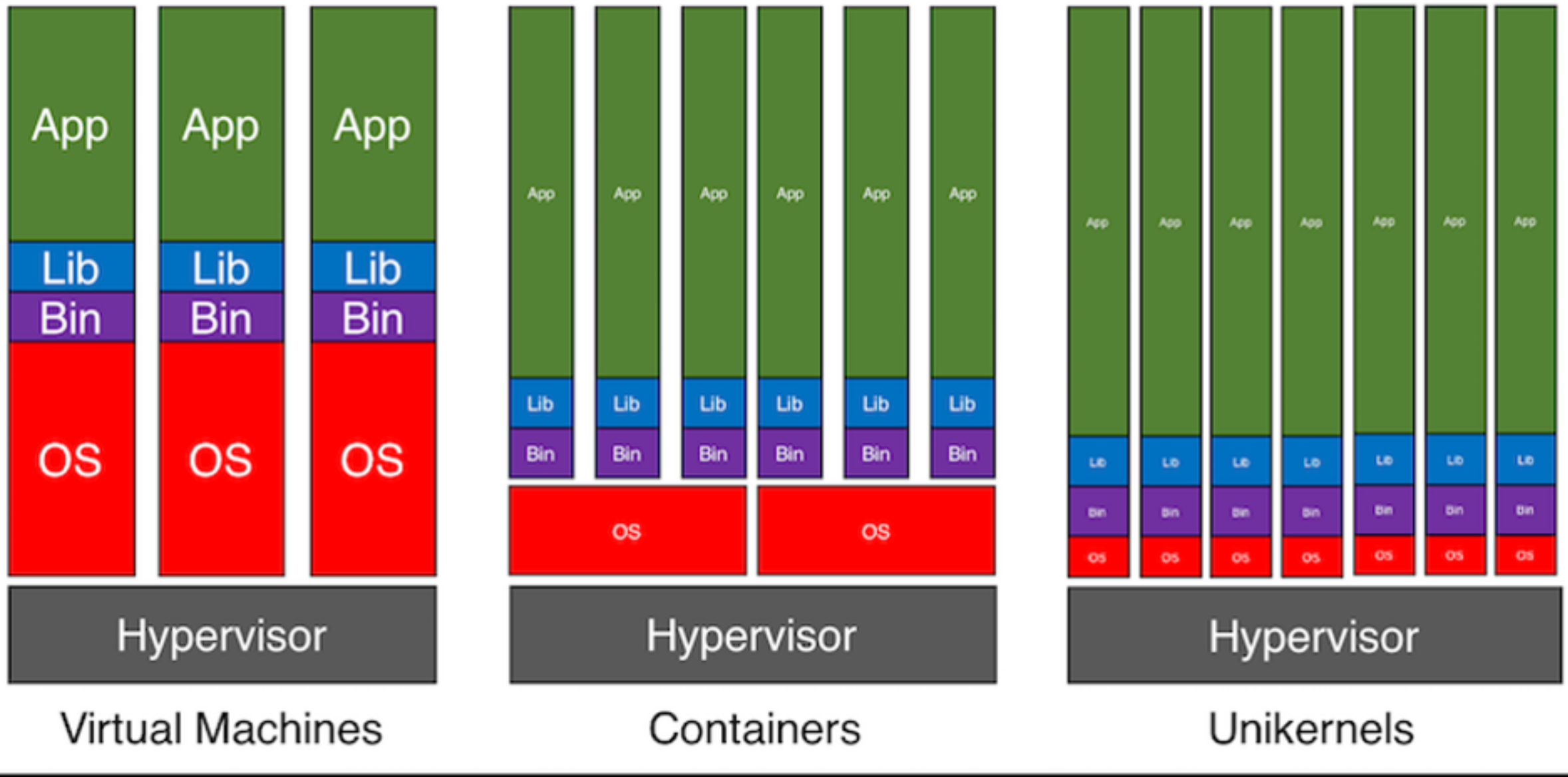
- Everything (including the kernel bits, e.g. the TCP/IP stack) is a **library**.
- Use a package manager and a compiler to **link** only what is needed.
- Fix the deployment target: Unix process, hypervisor (Xen, KVM, **Solo5**), bare metal.

At runtime:

- **single self-contained static image** which runs
  - in a single process
  - in a single address space
  - including the kernel



Unikernels


App
Lib
Bin
OS
Hypervisor
Virtual Machines
Containers
Unikernels

Unikernels

Benefits

- static linking + dead-code elimination:
  - removes all unnecessary code: DNS server is ~100kiB
  - smaller attack surface

- Use a few MiB of RAM: https://mirage.io uses 32 MiB (including the TLS stack)

Demo

MirageOS on macOS

linuxKit

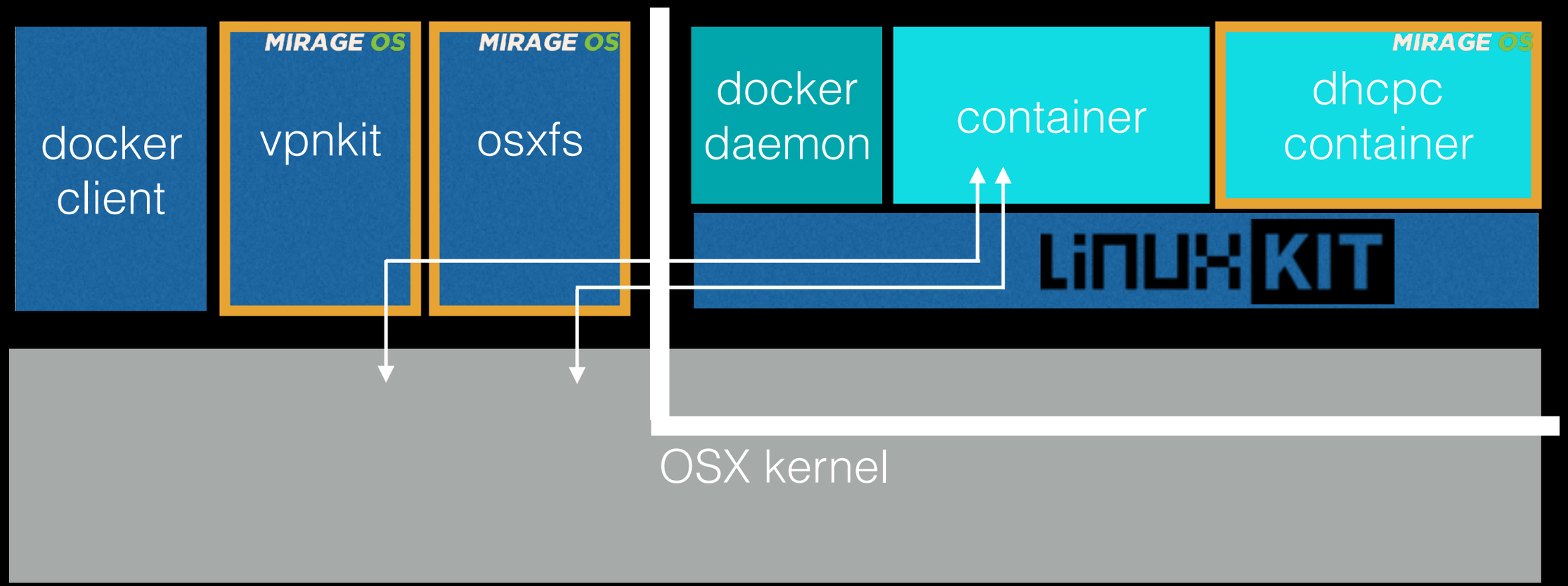
Demo

MirageOS on Xen

linuxKit

Docker for Mac

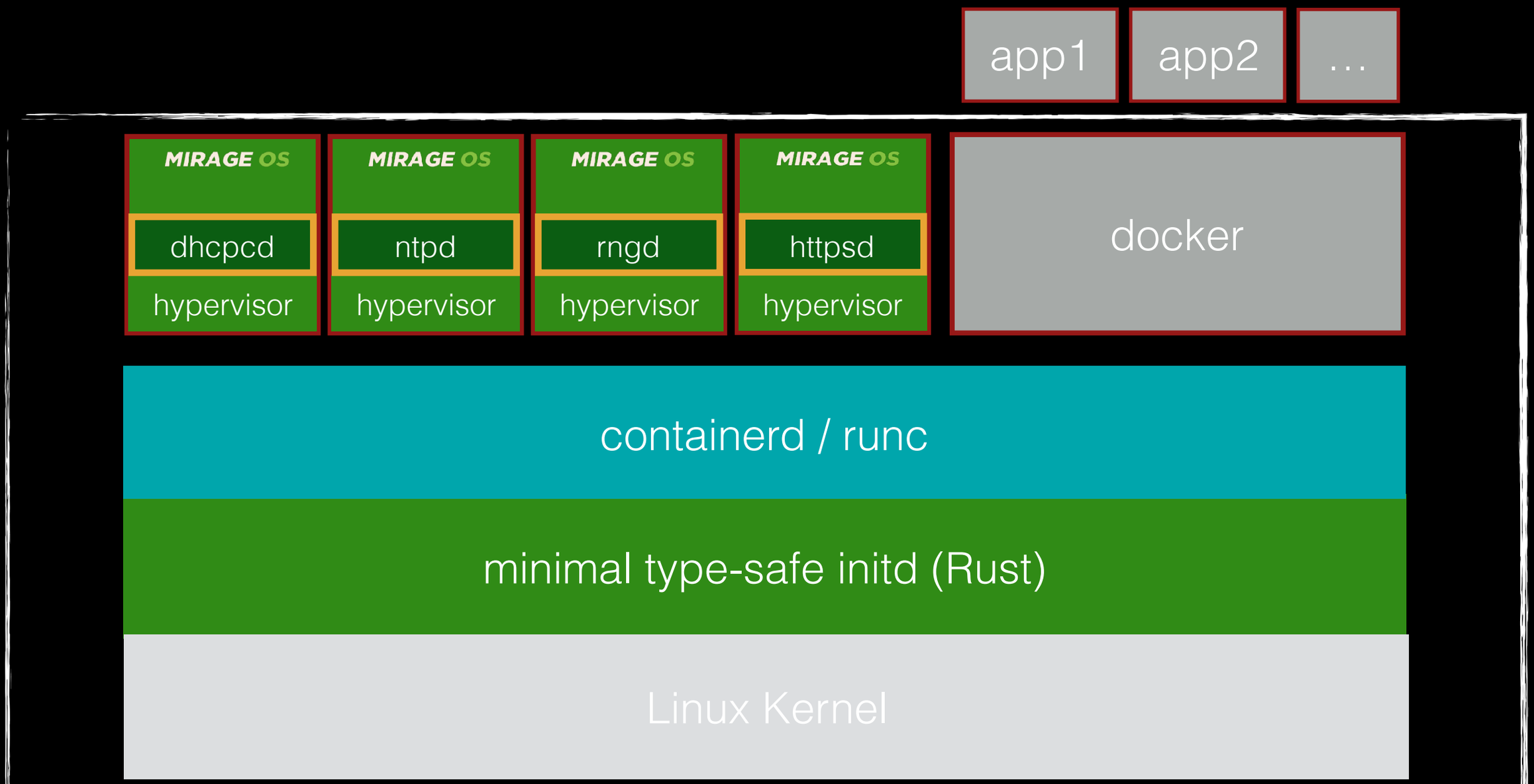
User-space Unikernels



Hardware

MirageSDK

Type-safe System Daemons



linuxkit

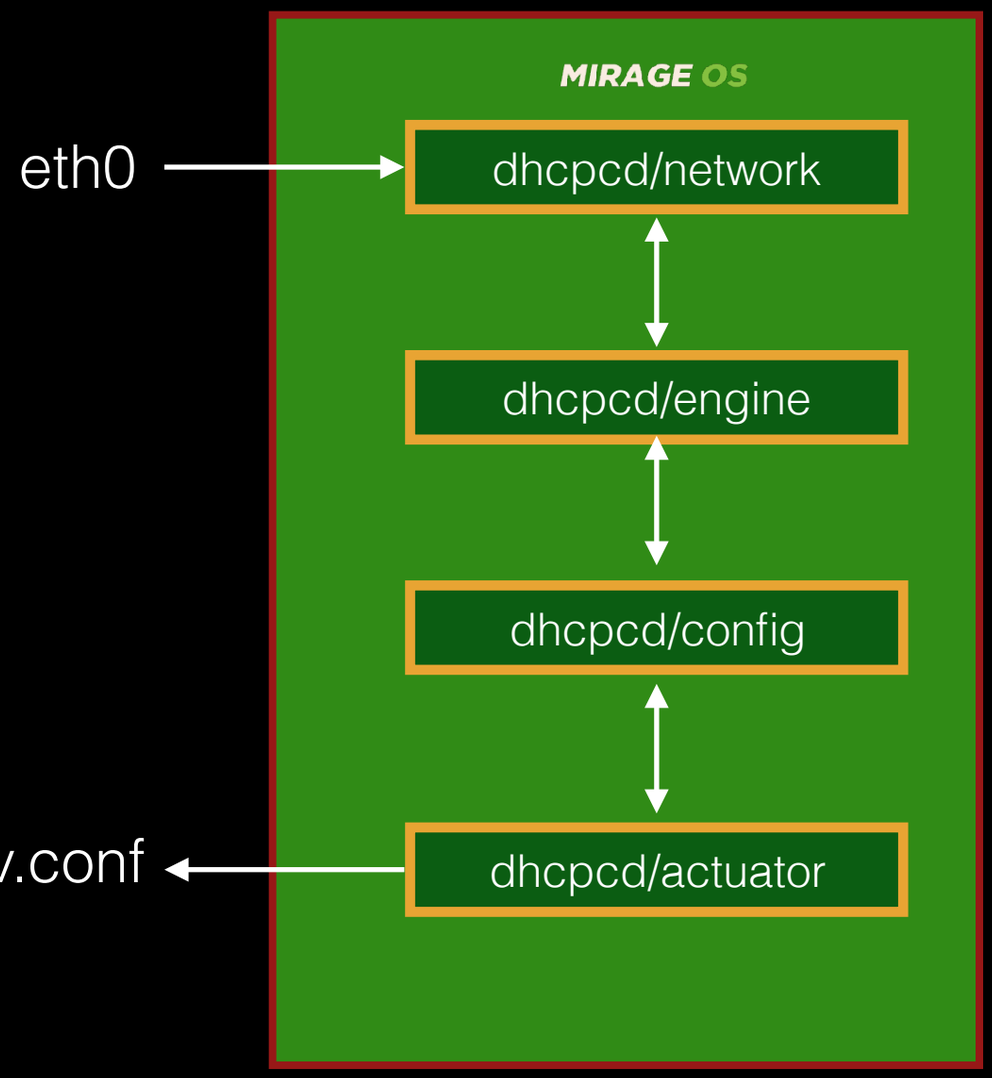
MirageSDK

Type-safe System Daemons

- DHCP is the first daemon that we are prototyping.
- This is a difficult daemon to privilege separate due to the deep (and non portable) system hooks required to handle IP and routing tables (e.g. netlink).
- Implementation fleshes out a lot of architectural questions and makes subsequent protocol implementations such as NTP and HTTPS more straightforward.

MirageSDK

Type-safe System Daemons

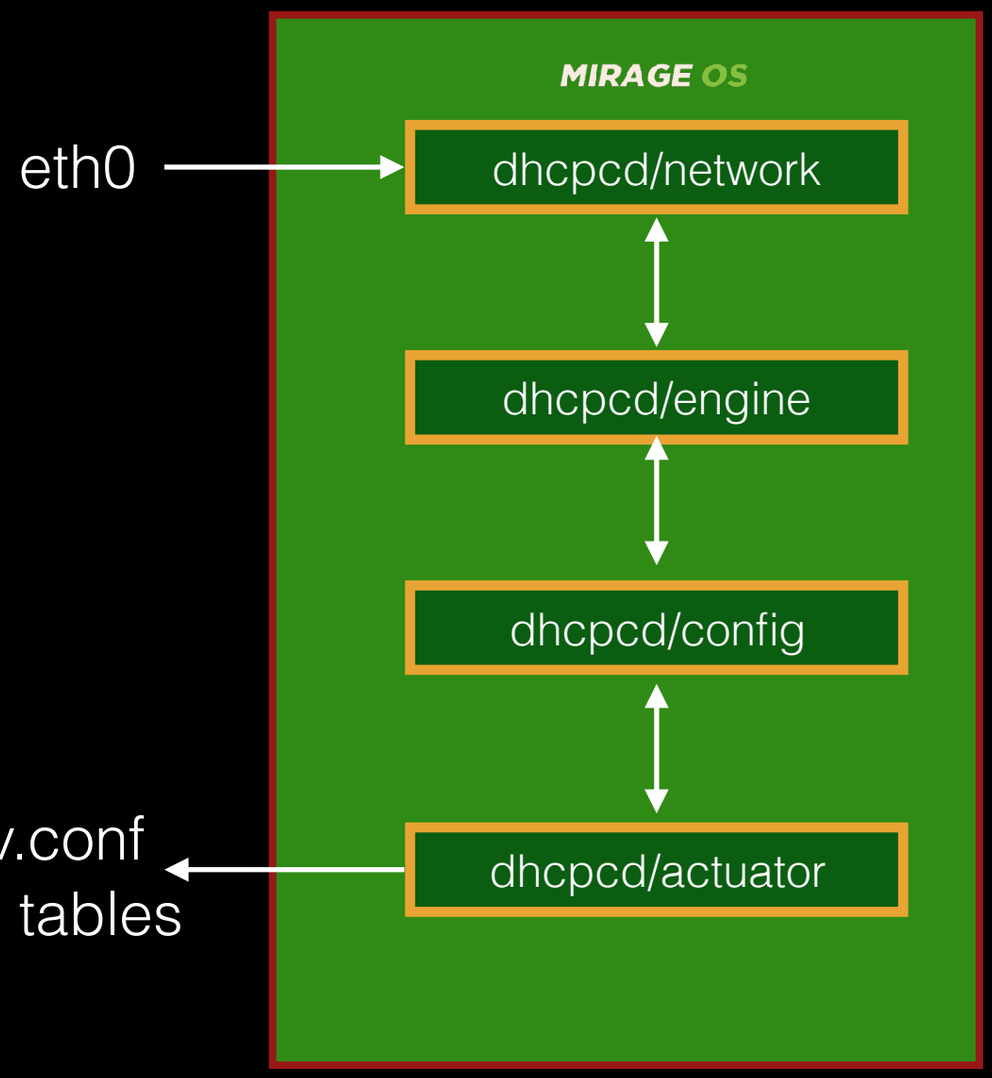


User-space privileged separation:

- every component is a container, with only one process
- components communicate via “secure” channels using RPC (cap’n proto)
- use of seccomp / eBPF to create a sandbox at the syscall granularity

MirageSDK

Type-safe System Daemons



- **network**: can read network interface and forward DHCP traffic.
- **engine**: can see nothing except channels to network and config containers
- **config**: store DHCP configuration
- **actuator**: can manipulate routing tables but cannot see network

MirageSDK

Going Deeper for Security

- Need protection at all levels of the stack for defence in depth:
  - **application level:** static type safety when parsing network traffic (via OCaml, Rust logic) and secure RPC (via capnp)
  - **protocol state machine**: fuzz testing for rapid space exploration (via American Fuzzy Loop aka AFL)
  - **runtime process**: container namespacing and KVM hardware protection if available (via unikernel Solo5)
  - **kernel interface**: eBPF sandboxing for fine-grained access to sys calls
  - **implementation diversity**: the container/rpc approach lets many runtime/language work together without tight coupling
- **What else?** LinuxKit lets us patch kernel and use facility directly into the base daemons, just like BSD distros. SGX, TrustZone, etc

Wrapping Up

- **Docker for Desktop** uses unikernel technology under the hood, ships to millions of users:

  Docker Distributed System Summit: https://www.youtube.com/watch?v=dn4ARS4IDIQ

- **LinuxKit** uses a “unikernel”-like approach to build secure and immutable Linux distributions. It also uses unikernel technologies for improving security of system daemons.

  LinuxKit Security SIG: https://github.com/linuxkit/linuxkit/blob/master/reports/sig-security/2017-06-07.md

Tooling for using and deploying unikernels is improving as a side-effect, and community is growing: includeOS (C++), deferPanic (Go)

Merci!