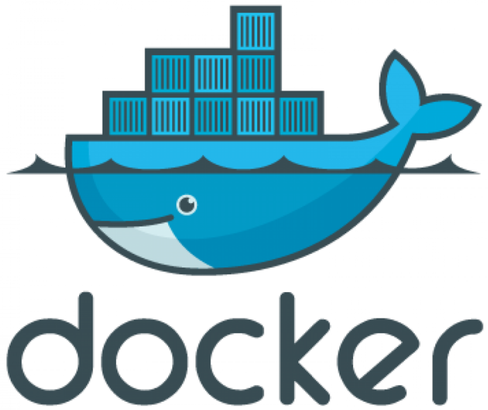
Functional Networking for Millions of Docker Desktops

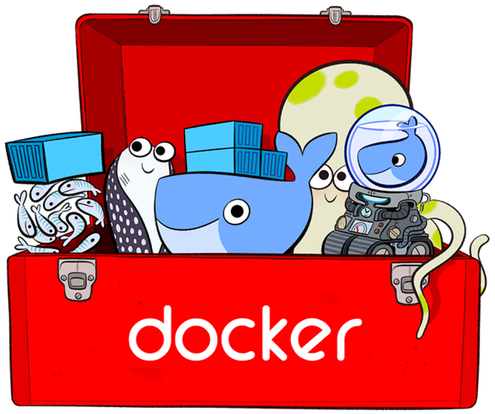
Anil Madhavapeddy (speaker), *David J. Scott,*
Patrick Ferris, Ryan T. Gibb, Thomas Gazagnaire
ICFP 2025, Singapore, Oct 14th 2025

With thanks to all the contributors over the years including Milas Bowman, Emmanuel Briney, Ian Campbell, Mathieu Champlon, Justin Cormack, Frédéric Dalleau, Akim Demaille, Ilya Dmitrichenko, Simon Ferquel, Pierre Gayvallet, Riyaz Faizullabhoy, Christiano Haesbaert, Anca Iordache, Thomas Leonard, Richard Mortier, Terry Moschou, Rolf Neugebauer, Mindy Preston, Michael Roitzsch, Guillaume Rose, Akihiro Suda, Balraj Singh, Magnus Skjegstad, David Sheets, Sebastiaan van Stijn, Tibor Vass, Gaetan de Villele, Ryuichi Watanabe, YAMAMOTO Takashi, Jeremy Yallop, the original Docker project founder Solomon Hykes and the MirageOS and OCaml developer teams.

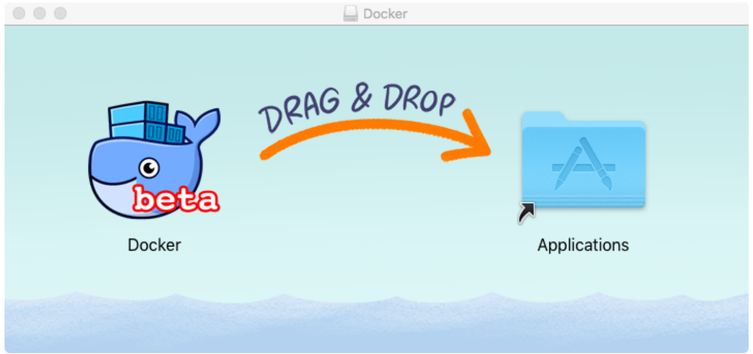


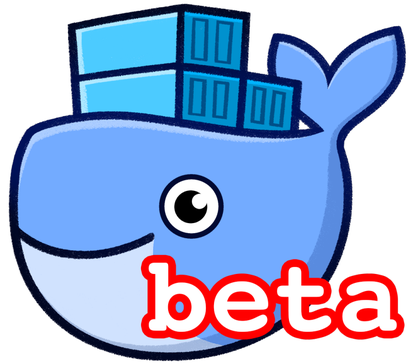
Docker for Desktop

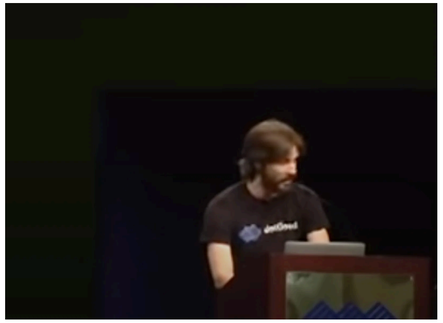
Aiming for a native macOS/Windows experience that works with existing developer workflows.



- Easy drag and drop installation, and autoupdates to get latest Docker.
- Secure, sandboxed virtualisation architecture without elevated privileges.
- Native networking support, with VPN and network sharing compatibility.
- File sharing between container and host: uid mapping, inotify events, etc.
- **Released in 2016, with 100s of millions of users**



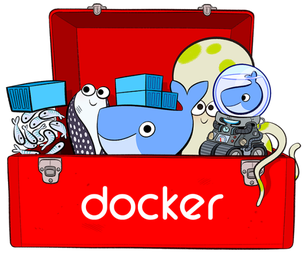


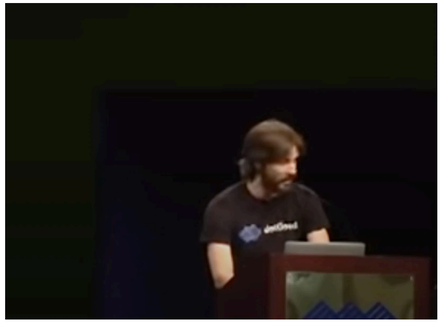


Docker launches at Pycon on Linux and is adopted quickly for cloud and microservices

2013 **2015**

Panic stations as users demand macOS / Windows support for Linux development





moby/**vpnkit**

A toolkit for embedding VPN capabilities in your application

Docker launches at Pycon on Linux and is adopted quickly for cloud and microservices

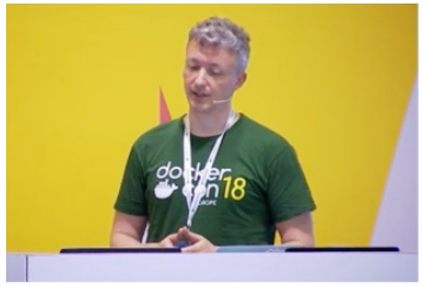
We start hacking on HyperKit unikernels to support macOS & Windows in our new Docker Desktop app

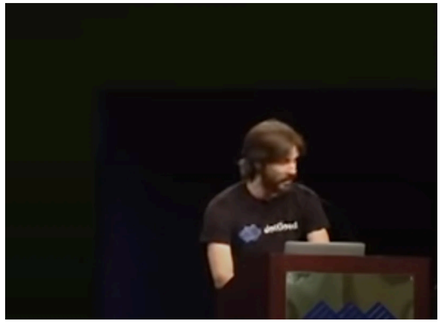
We release the OCaml VPNKit to fix around 99% of the bug reports and growth continues

2013 **2015** **2016**

Panic stations as users demand macOS / Windows support for Linux development

Docker Desktop beta is popular, but a huge influx of bug reports about corporate firewalls





moby/**vpnkit**

A toolkit for embedding VPN capabilities in your application

Docker launches at Pycon on Linux and is adopted quickly for cloud and microservices

We start hacking on HyperKit unikernels to support macOS & Windows in our new Docker Desktop app

We release the OCaml VPNKit to fix around 99% of the bug reports and growth continues

VPNKit is released as open-source using Lwt concurrency and MirageOS's TCP/IP, TLS & HTTP libraries

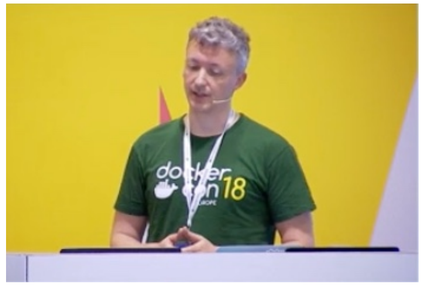
2013 **2015** **2016** **2017** **2026**

Panic stations as users demand macOS / Windows support for Linux development

Docker Desktop beta is popular, but a huge influx of bug reports about corporate firewalls

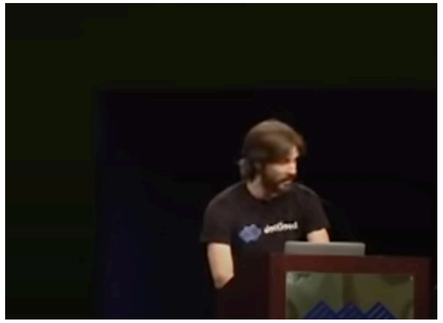
Docker Desktop is now supported by Apple's Hypervisor (and Virtualization) frameworks

A majority of professional software developers report using Docker Desktop in their work!



moby/**hyperkit**

A toolkit for embedding hypervisor capabilities in your application



moby/**vpnkit**

A toolkit for embedding VPN capabilities in your application

OCaml

Docker launches at Pycon on Linux and is adopted quickly for cloud and microservices

We start hacking on HyperKit unikernels to support macOS & Windows in our new Docker Desktop app

We release the OCaml VPNKit to fix around 99% of the bug reports and growth continues

VPNKit is released as open-source using Lwt concurrency and MirageOS's TCP/IP, TLS & HTTP libraries

VPNKit is sped up by using OCaml 5's shiny new effect handlers via our Eio direct-style library

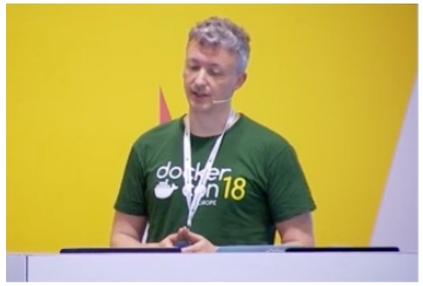
2013 **2015** **2016** **2017** **2026**

Panic stations as users demand macOS / Windows support for Linux development

Docker Desktop beta is popular, but a huge influx of bug reports about corporate firewalls

Docker Desktop is now supported by Apple's Hypervisor (and Virtualization) frameworks

A majority of professional software developers report using Docker Desktop in their work!



moby/**hyperkit**

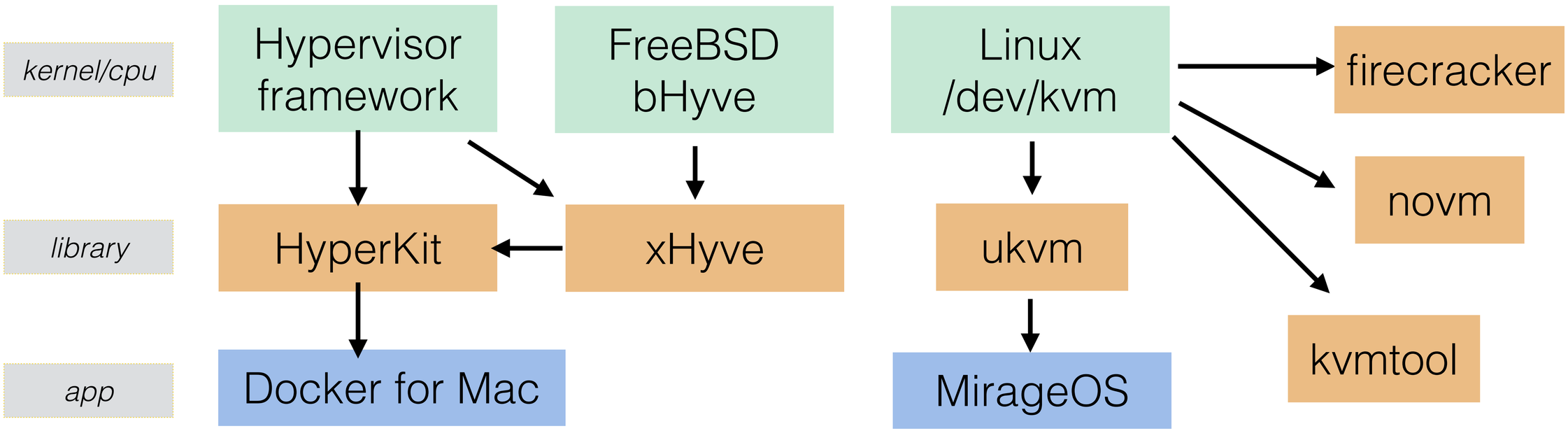
A toolkit for embedding hypervisor capabilities in your application

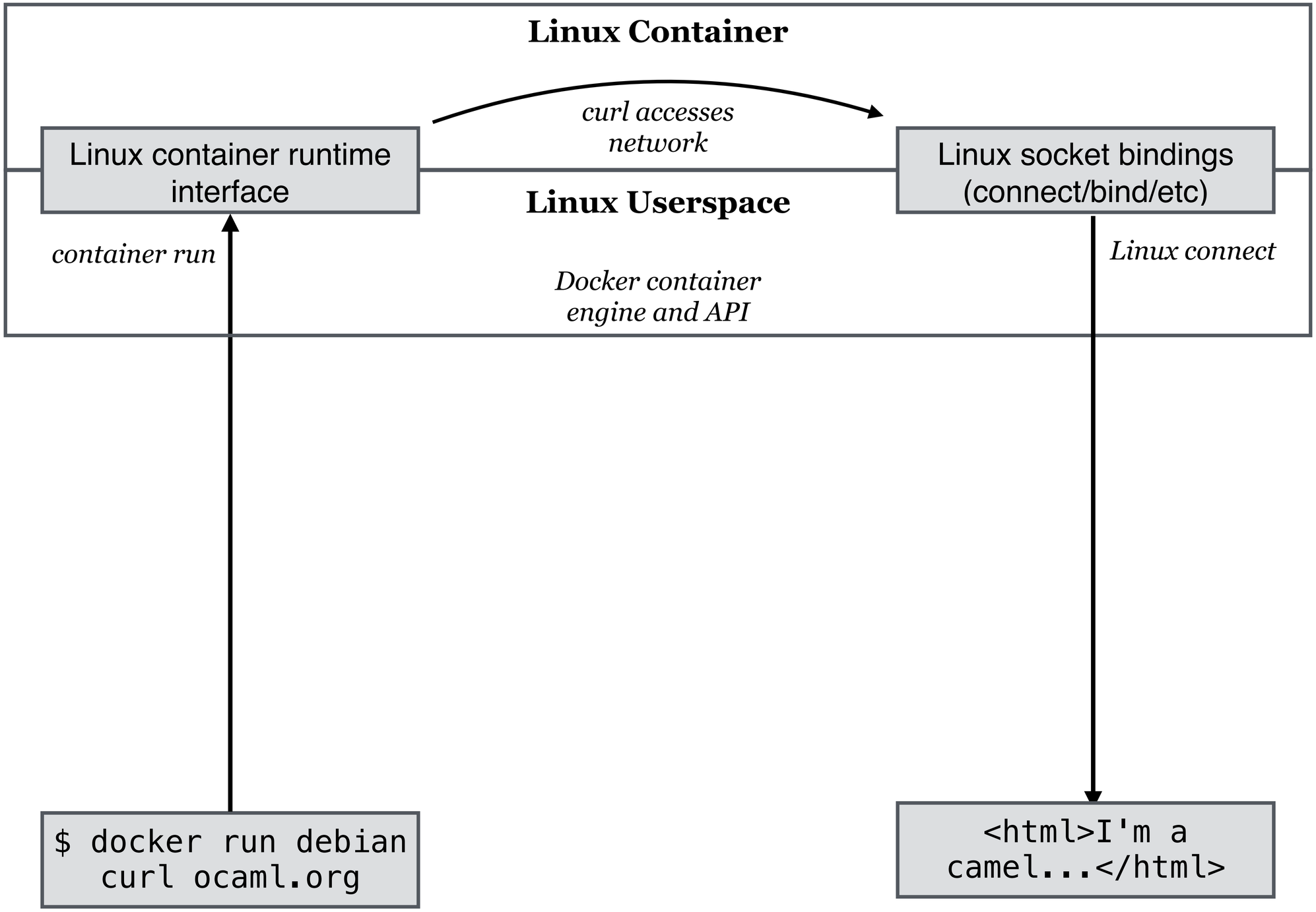
- **The problem:** users need to run Linux containers on macOS and Windows desktops without doing any extra configuration or changing their workflows.

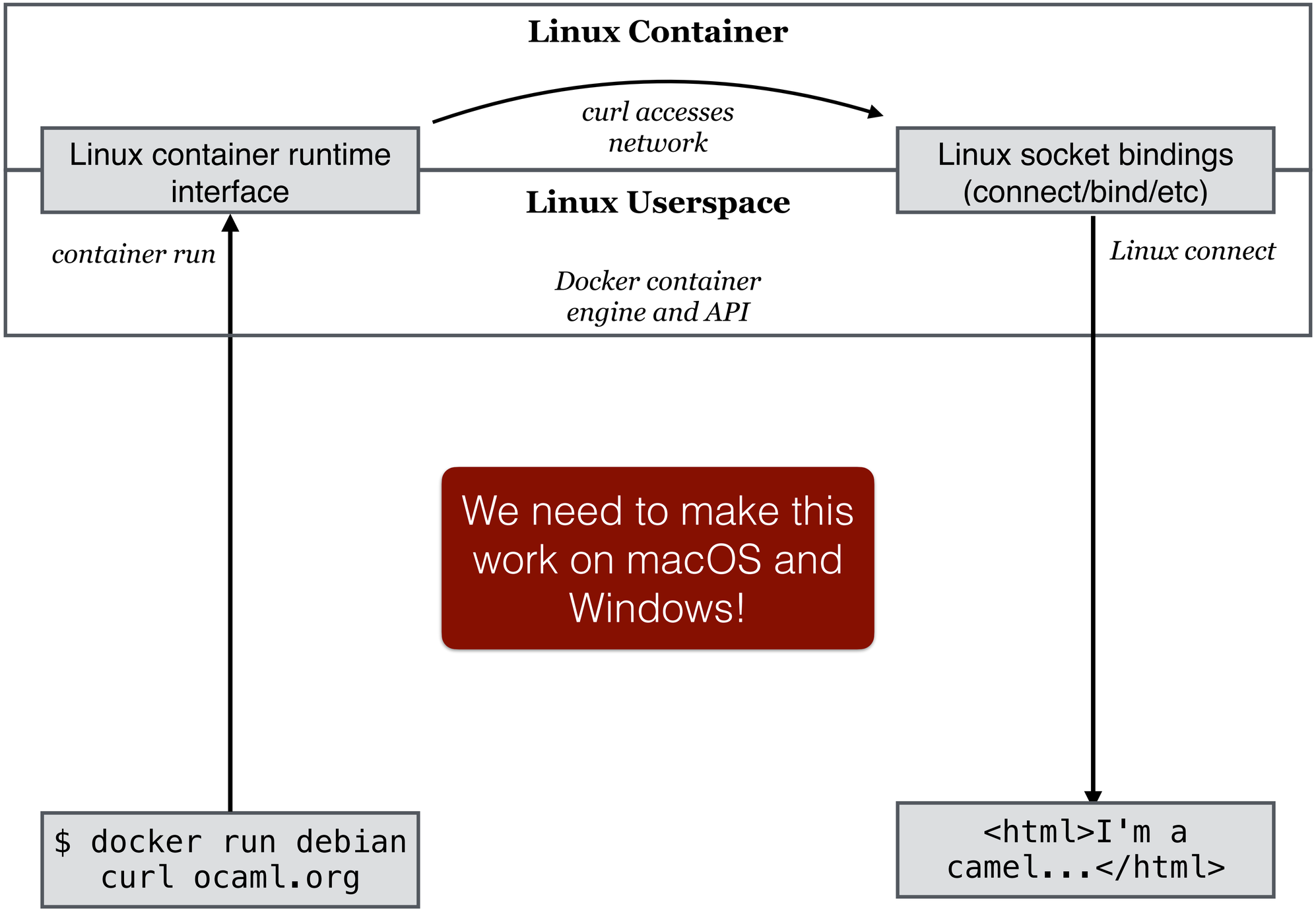
- **Our idea:** link a library Virtual Machine Monitor to a native macOS application, and make Linux itself an implementation detail!

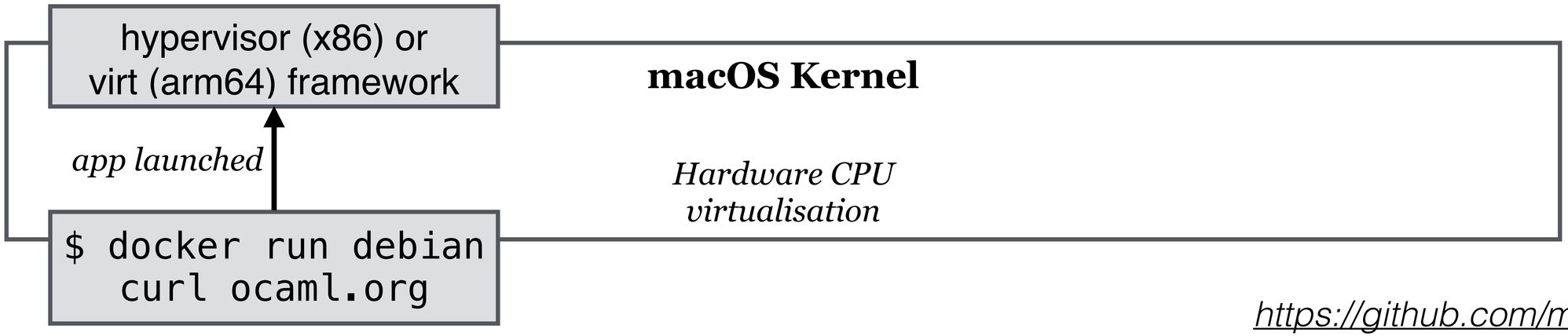
- **The benefits:** users have an "app experience" without the gory details of virtual machine management.

- **The problem:** users need to run Linux containers on macOS and Windows desktops without doing any extra configuration or changing their workflows.
- **Our idea:** link a library Virtual Machine Monitor to a native macOS application, and make Linux itself an implementation detail!
- **The benefits:** users have an "app experience" without the gory details of virtual machine management.




Linux Container
curl accesses network
Linux container runtime interface
Linux socket bindings (connect/bind/etc)
Linux Userspace
container run
Linux connect
Docker container engine and API
$ docker run debian curl ocaml.org
<html>I'm a camel...</html>

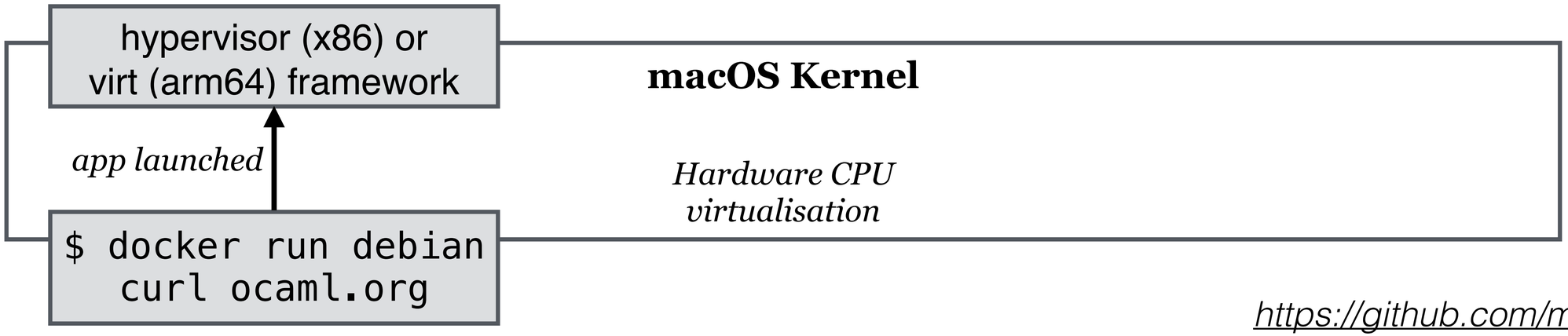

Linux Container
curl accesses network
Linux container runtime interface
Linux socket bindings (connect/bind/etc)
Linux Userspace
container run
Linux connect
Docker container engine and API
We need to make this work on macOS and Windows!
$ docker run debian curl ocaml.org
<html>I'm a camel...</html>



https://github.com/moby/hyperkit

```
static void vm_init(struct vm *vm, bool create) {
  int vcpu;

  if (create) callout_system_init();

  vm->cookie = VM_INIT(vm);
  vm->vioapic = vioapic_init(vm);
  vm->vhpet = vhpet_init(vm);
  vm->vatpic = vatpic_init(vm);
  vm->vatpit = vatpit_init(vm);
  vm->vpmtmr = vpmtmr_init(vm);

  if (create) vm->vrtc = vrtc_init(vm);

  CPU_ZERO(&vm->active_cpus);

  vm->suspend = 0;
  CPU_ZERO(&vm->suspended_cpus);

  for (vcpu = 0; vcpu < VM_MAXCPU; vcpu++) {
    vcpu_init(vm, vcpu, create);
  }
}
```

HyperKit maps VM operations into single-threaded C library function calls in userspace



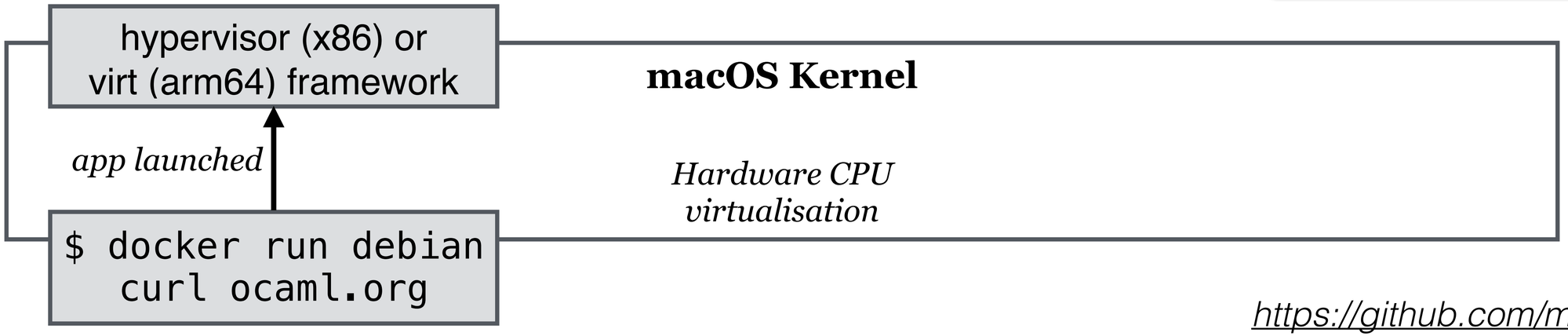
https://github.com/moby/hyperkit

```
static void
ocaml_mirage_block_preadv(const int handle, const struct iovec *iov,
                          int iovcnt, off_t ofs, ssize_t *out, int *err) {
  CAMLparam0();
  CAMLlocal4(ocaml_handle, ocaml_bufs, ocaml_ofs, ocaml_result);
  ocaml_handle = Val_int(handle);
  ocaml_bufs = caml_alloc_tuple((mlsize_t)iovcnt);
  ocaml_ofs = Val_int(ofs);
  for (int i = 0; i < iovcnt; i++ ){
    Store_field(ocaml_bufs, (mlsize_t)i, caml_ba_alloc_dims(CAML_BA_CHAR | CAML_BA_C_LAYOUT
      1, (*(iov+i)).iov_base, (*(iov+i)).iov_len));
  }
  OCAML_NAMED_FUNCTION("mirage_block_preadv")
  ocaml_result = caml_callback3_exn(*fn, ocaml_handle, ocaml_bufs, ocaml_ofs);
  if (Is_exception_result(ocaml_result)) {
    *err = 1;
  } else {
    *err = 0;
    *out = Int_val(ocaml_result);
  }
  CAMLreturn0;
}
```

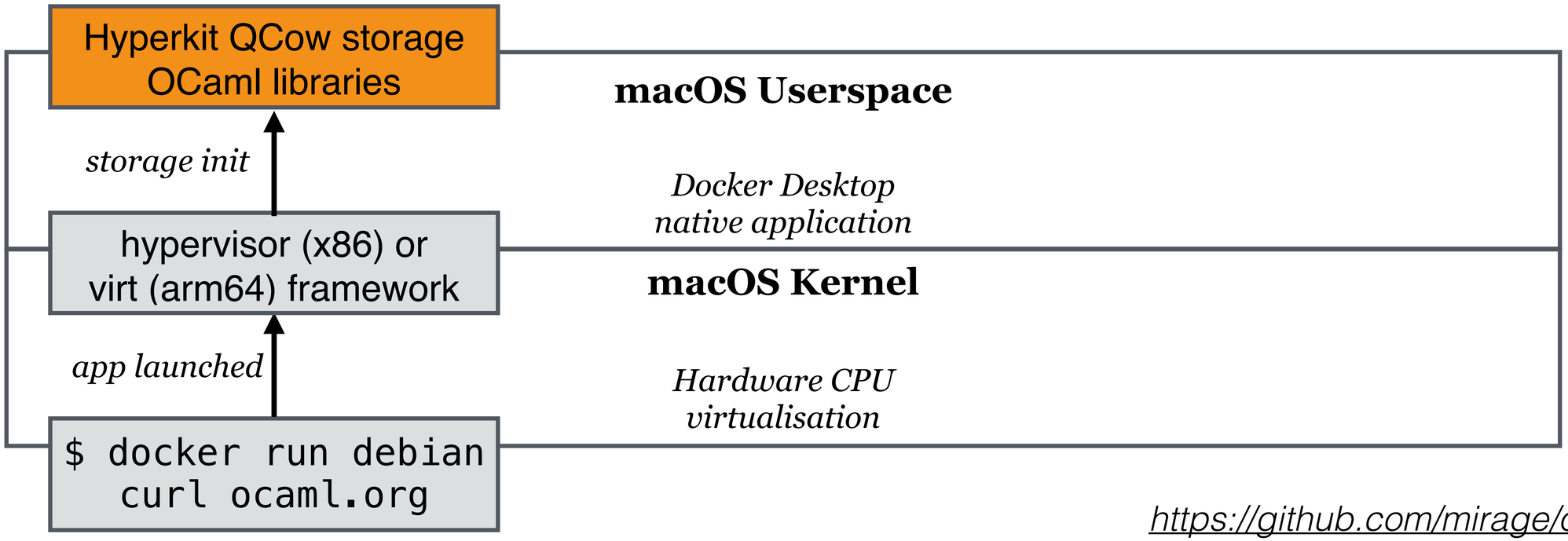
OCaml is also linked as a library and uses its standard callback FFI to call into OCaml code

Standard C library linking conventions + OCaml FFI

Lets us shift to higher level functional programming!



https://github.com/moby/hyperkit



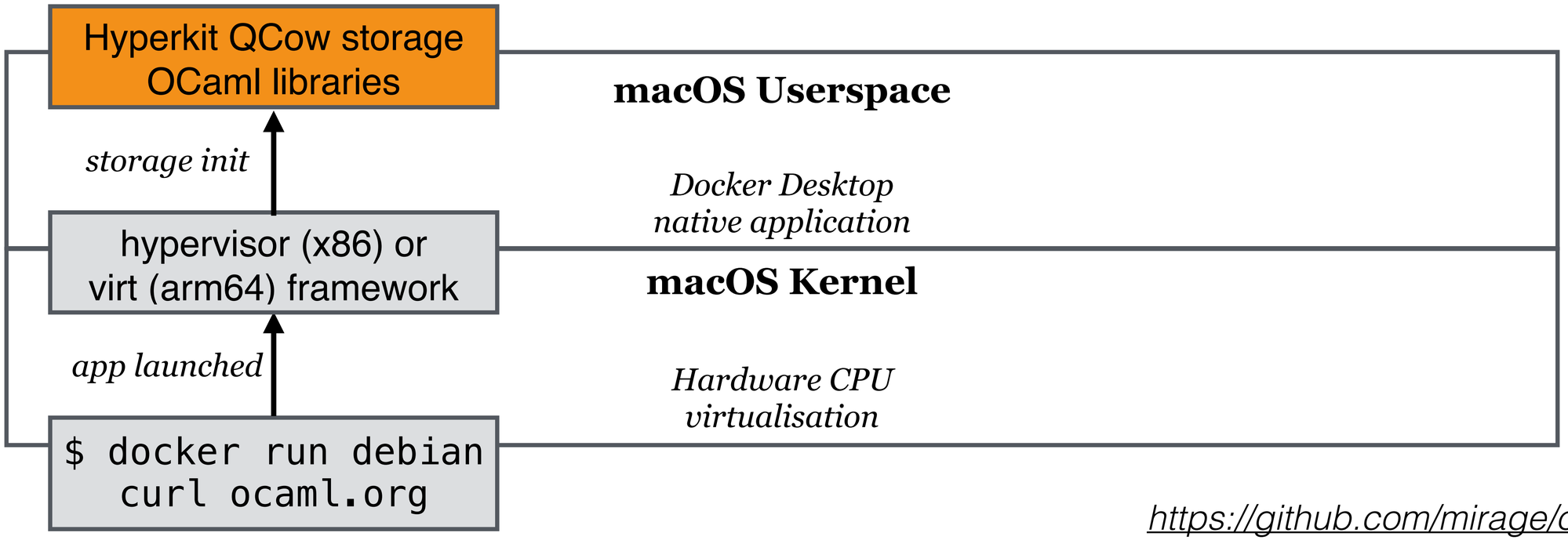
https://github.com/mirage/ocaml-qcow

```
module Request = struct
  type t =
  | Connect of Block.Config.t * Qcow.Config.t
  | Get_info of int
  | Disconnect of int
  | Read of int * int * (Cstruct.t list)
  | Write of int * int * (Cstruct.t list)
  | Delete of int * int64 * int64
  | Flush of int
end
```

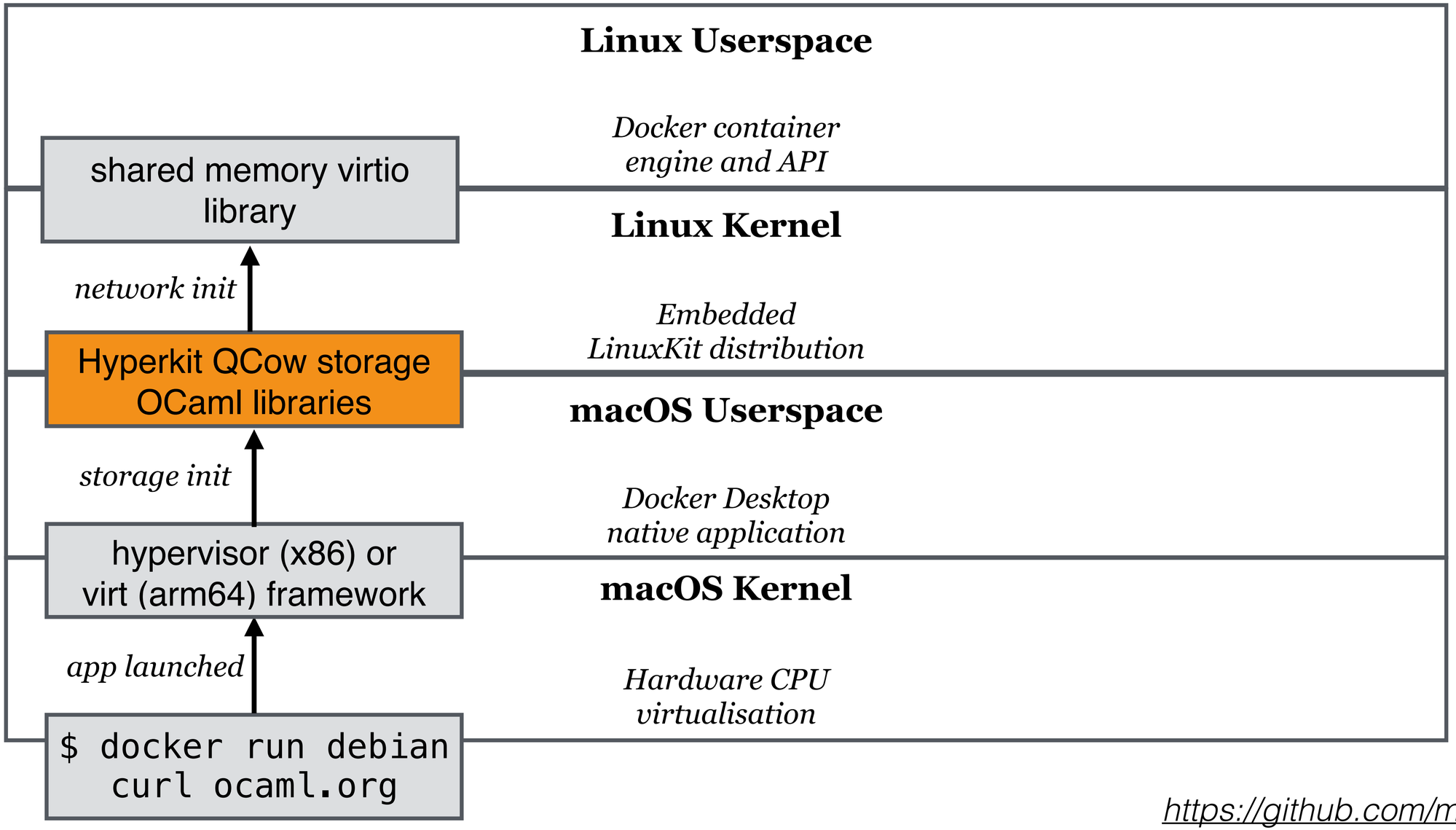
Algebraic data types for block storage operations

```
module Response = struct
  type ok =
  | Connect of int
  | Get_info of bool * int * int64 * bool
  | Disconnect
  | Read of int | Write of int
  | Delete
  | Flush
  type t = (ok, Qcow.write_error) result
end
```

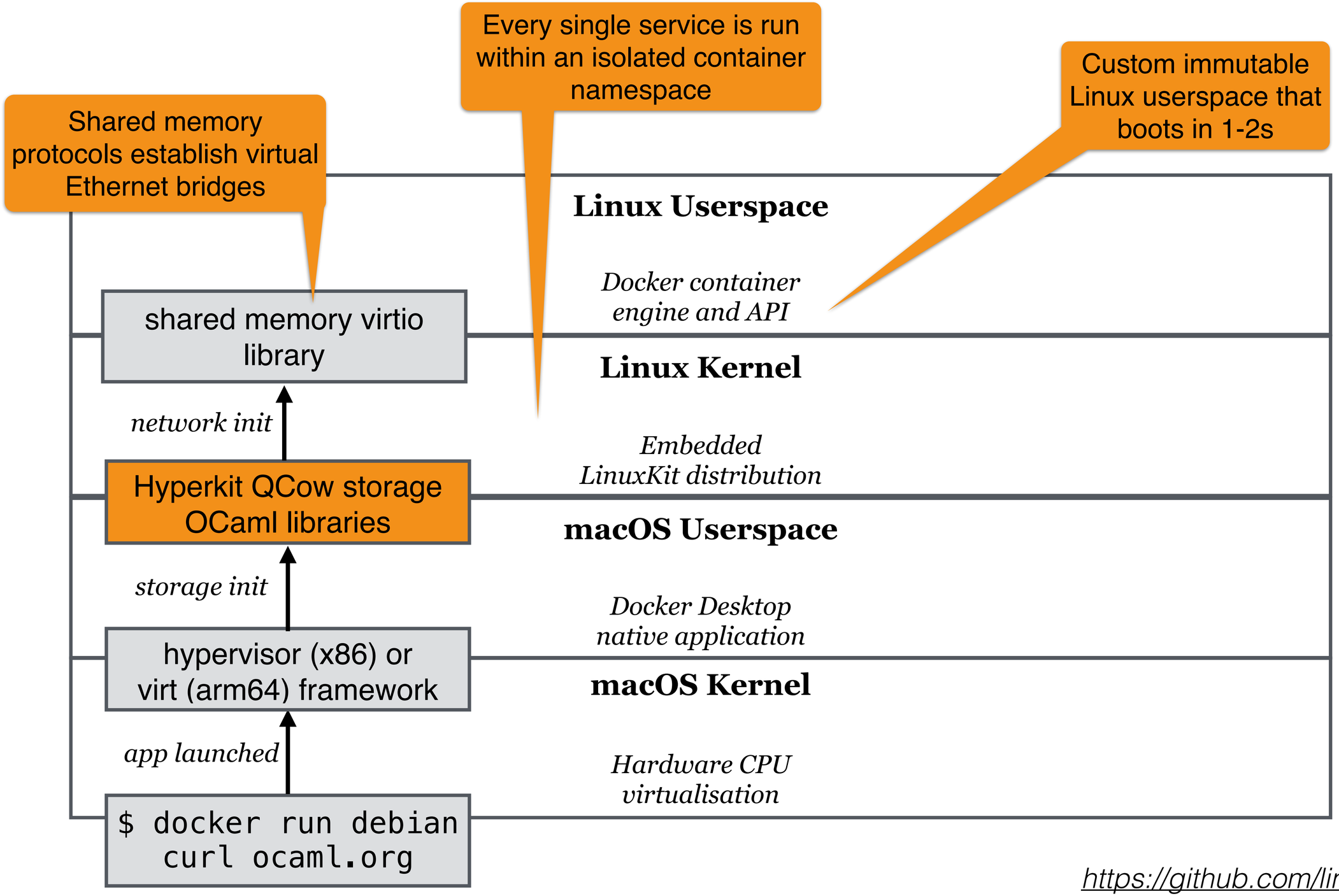
OCaml implementation of the QCOW format



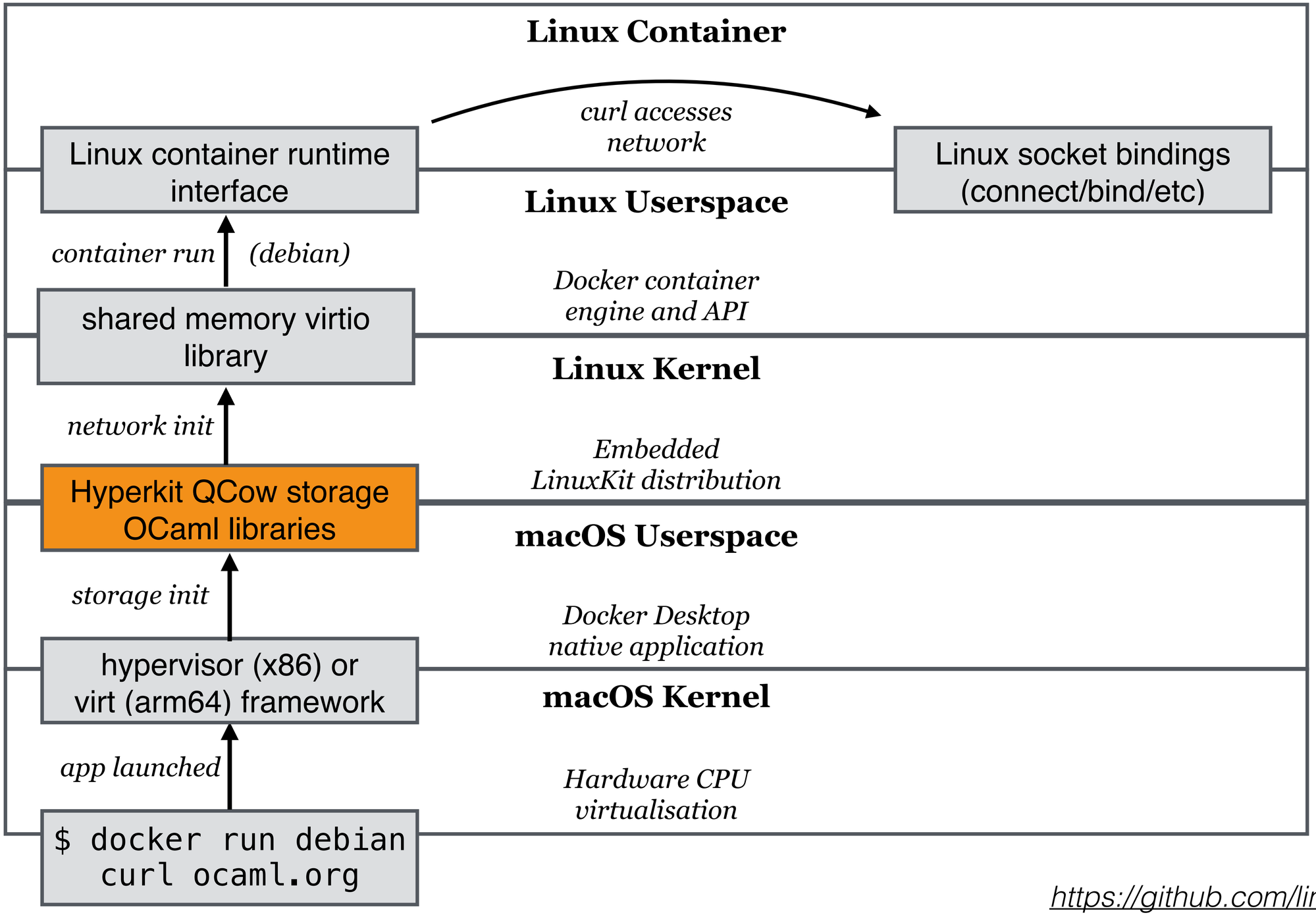
https://github.com/mirage/ocaml-qcow



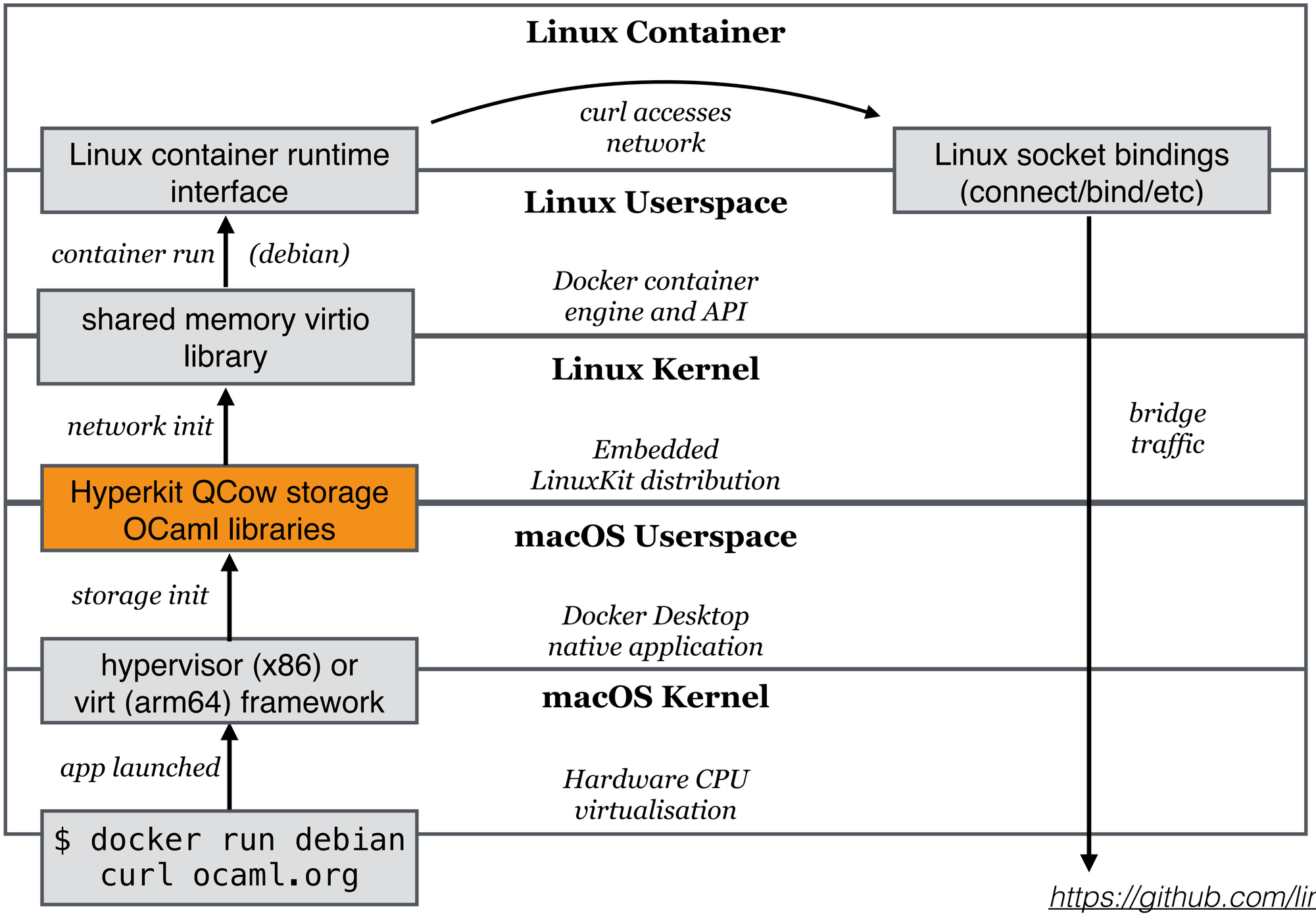
https://github.com/moby/hyperkit



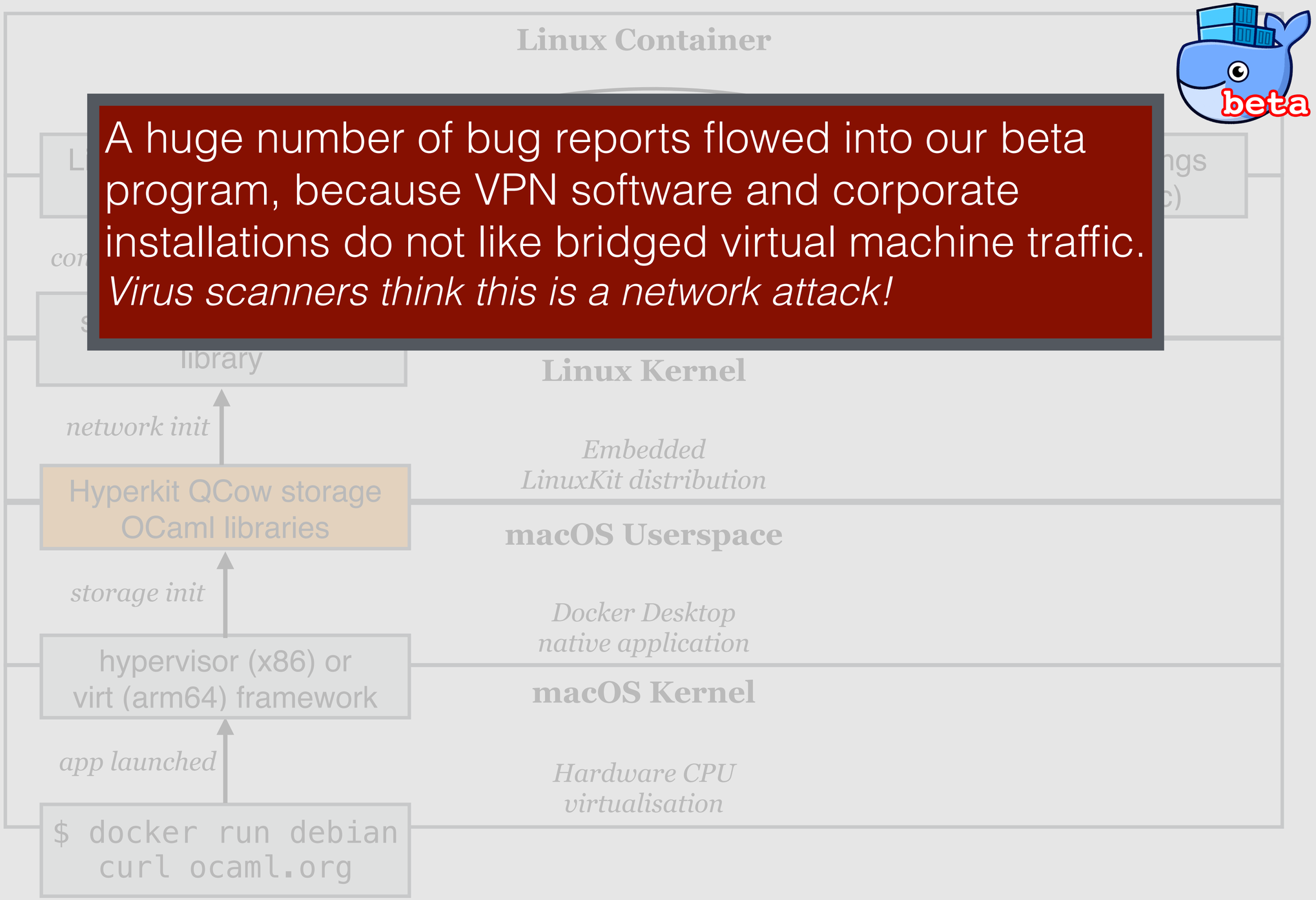
https://github.com/linuxkit/linuxkit



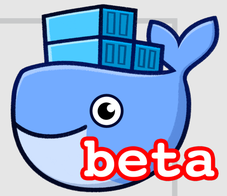
https://github.com/linuxkit/linuxkit



https://github.com/linuxkit/linuxkit


Linux Container
A huge number of bug reports flowed into our beta program, because VPN software and corporate installations do not like bridged virtual machine traffic.
Virus scanners think this is a network attack!
beta
library
Linux Kernel
network init
Embedded
LinuxKit distribution
Hyperkit QCow storage
OCaml libraries
macOS Userspace
storage init
Docker Desktop
native application
hypervisor (x86) or
virt (arm64) framework
macOS Kernel
app launched
Hardware CPU
virtualisation
$ docker run debian
curl ocaml.org

Linux Container



A huge number of bug reports flowed into our beta program, because VPN software and corporate installations do not like bridged virtual machine traffic.
Virus scanners think this is a network attack!

Idea: let's repurpose a 1990s protocol used by PalmPilots called SLIRP, and translate the low-level Ethernet network traffic into macOS socket calls.
We reconstruct network traffic into socket calls!

native application

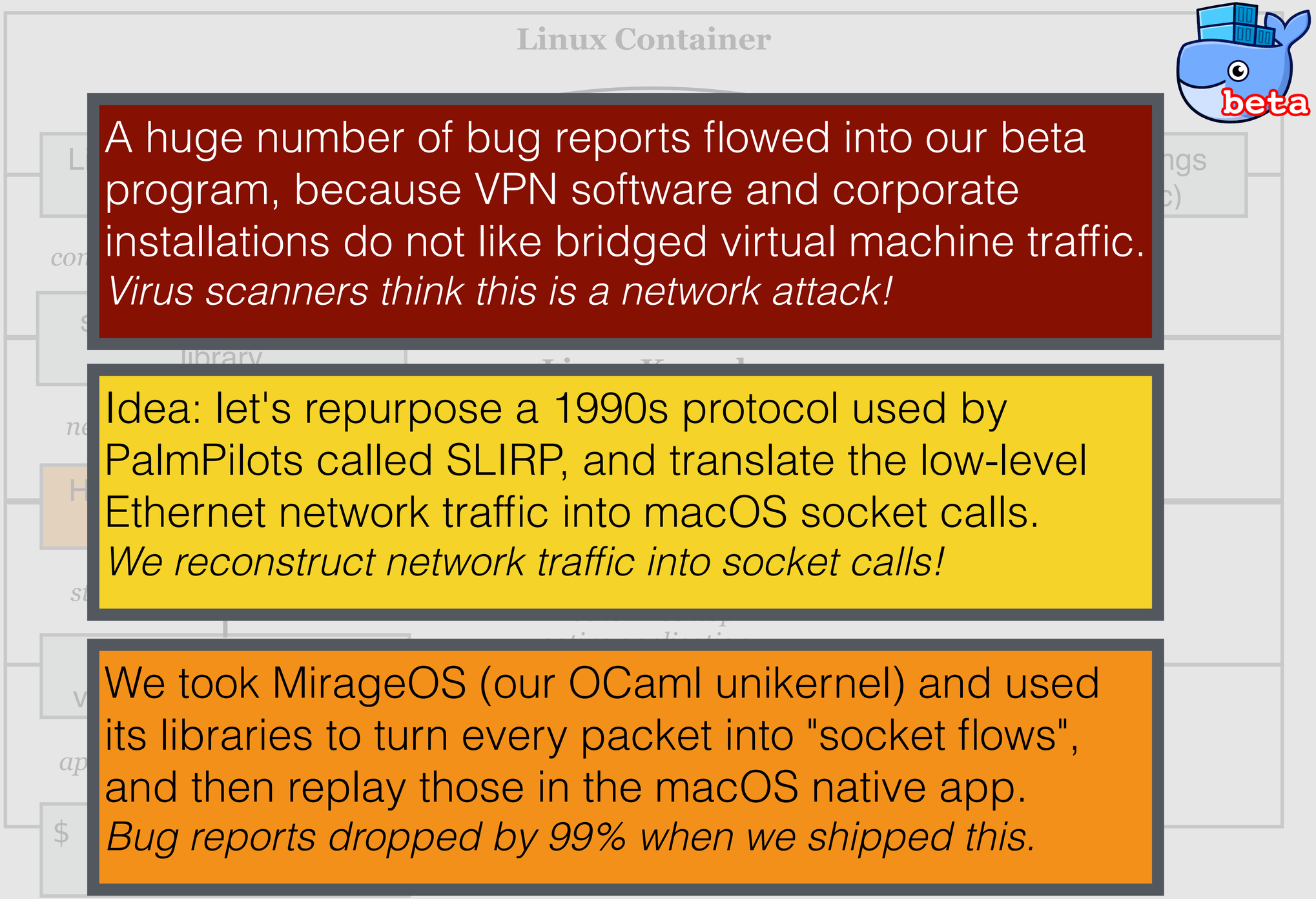
hypervisor (x86) or
virt (arm64) framework

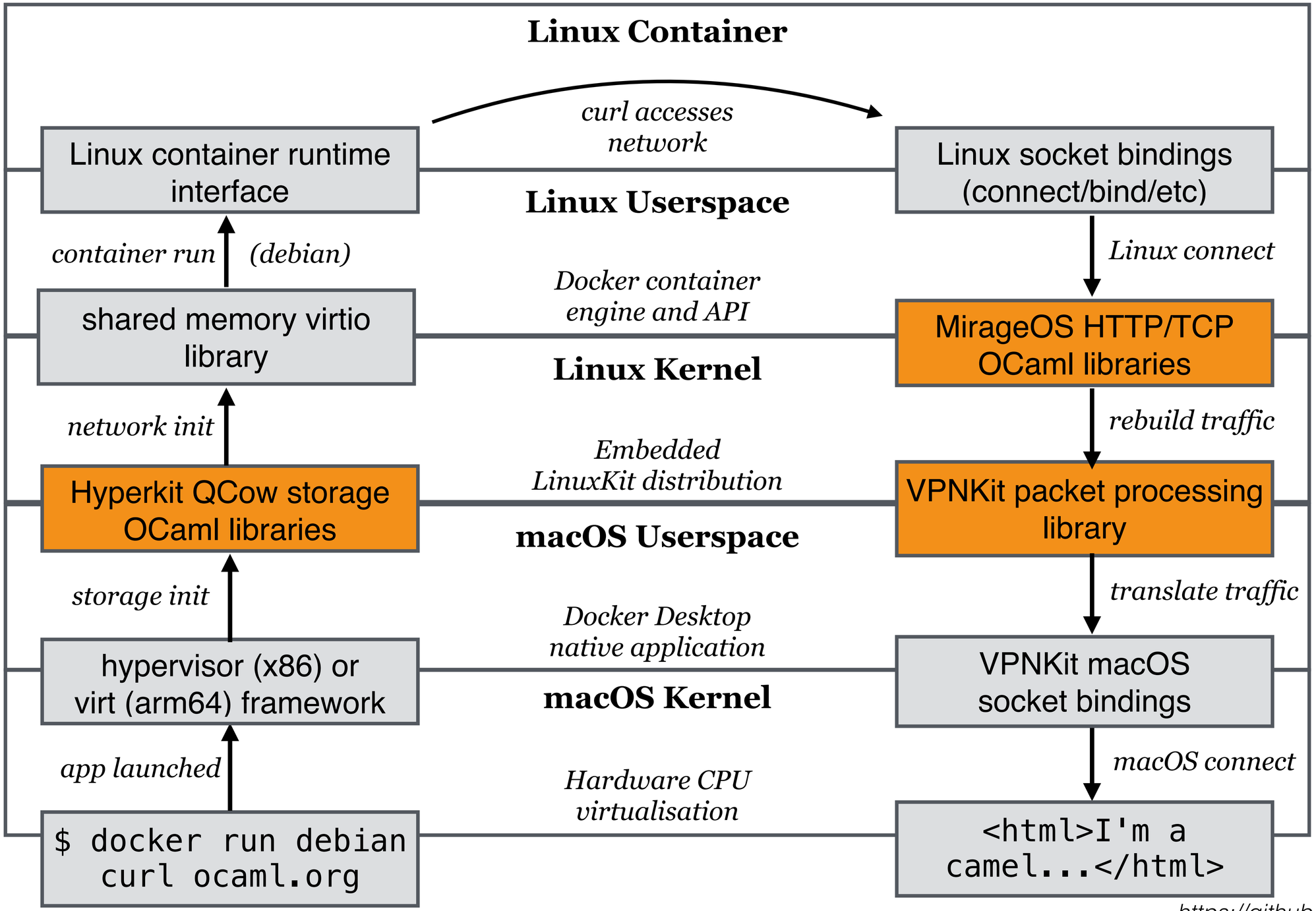
macOS Kernel

app launched

Hardware CPU virtualisation

```
$ docker run debian
  curl ocaml.org
```


Linux Container
beta
A huge number of bug reports flowed into our beta program, because VPN software and corporate installations do not like bridged virtual machine traffic.
Virus scanners think this is a network attack!
Idea: let's repurpose a 1990s protocol used by PalmPilots called SLIRP, and translate the low-level Ethernet network traffic into macOS socket calls.
We reconstruct network traffic into socket calls!
We took MirageOS (our OCaml unikernel) and used its libraries to turn every packet into "socket flows", and then replay those in the macOS native app.
Bug reports dropped by 99% when we shipped this.



https://github.com/moby/vpnkit

Using OCaml in production: the Good Bits

- Complex network logic using OCaml functors is highly modular

OCaml module signatures to assemble our interfaces

```
module type FLOW_CLIENT = sig
 include Mirage_flow_combinators.SHUTDOWNABLE
 type address
 val connect:
   ?read_buffer_size:int ->
   address ->
   (flow, [`Msg of string]) result Lwt.t
end

module type Connector = sig
 include FLOW_CLIENT
 val connect: unit -> flow Lwt.t
 include READ_INTO with type flow := flow
                    and type error := error
end
```

https://github.com/moby/vpnkit

Using OCaml in production: the Good Bits

- Complex network logic using OCaml functors is highly modular

OCaml module signatures to assemble our interfaces

OCaml functors to compose higher order implementations multiple times

```
module type FLOW_CLIENT = sig
 include Mirage_flow_combinators.SHUTDOWNABLE
 type address
 val connect:
   ?read_buffer_size:int ->
   address ->
   (flow, [`Msg of string]) result Lwt.t
end

module type Connector = sig
 include FLOW_CLIENT
 val connect: unit -> flow Lwt.t
 include READ_INTO with type flow := flow
                    and type error := error
end
```

```
module Bind = Bind.Make(Host.Sockets)
module Forward_unix = Forward.Make(Mclock)(Connect.Unix)(Bind)
module Forward_hvsock = Forward.Make(Mclock)(Connect.Hvsock)(Bind)
```

https://github.com/moby/vpnkit

Using OCaml in production: the Good Bits

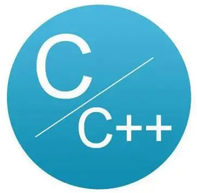
- Complex network logic using OCaml functors is highly modular
- Pattern matching on ADTs is awesome for network processing!

```
let input_ipv4 t ipv4 = match ipv4 with
| Ipv4 {src; dst; payload =
    Udp { src = src_port; dst = 53;
          payload = Payload payload; _ }; _} ->
    let udp = t.endpoint.Endpoint.udp4 in
    !dns >>= fun t ->
    Dns_forwarder.handle_udp ~t ~udp ~src ~dst
      ~src_port payload >|= lift_udp_error
| Ipv4 {src; dst;
        payload= Tcp {src=src_port; dst=dst_port;
          syn; rst; raw; payload=Payload _; _}; _} ->
    let id = Stack_tcp_wire.v ~src_port:dst_port
             ~dst:src ~src:dst ~dst_port:src_port in
    begin match !http with
    | None -> Lwt.return_ok
    | Some http ->
    let dst = dst, dst_port in
    (* HTTP proxy forwarding logic follows... *)
...
```

https://github.com/moby/vpnkit

Using OCaml in production: the Good Bits

- Complex network logic using OCaml functors is highly modular
- Pattern matching on ADTs is awesome for network processing!
- A stable and predictable foreign function interface for the systems bindings



Swift

One address space!

https://github.com/moby/vpnkit

Shifting towards direct-style effects

```
module Make_packet_proxy
(I: Mirage_flow.S) (O: Mirage_flow.S) = struct
 let run incoming outgoing =
  let rec loop () =
   I.read incoming >>= function
   | Error err -> fail "%a" I.pp_error err
   | Ok `Eof -> Lwt.return_unit
   | Ok (`Data buf) -> begin
      O.write outgoing buf >>= function
      | Ok () -> loop ()
      | Error err -> fail "%a" O.pp_error err
     end
  in loop ()
```

Older monadic Lwt code with custom binds and functors wrapping all logic

- Lwt is awesome, but monadic concurrency composes poorly with other monads like errors/options.
- Lwt uses custom operators and exposes concurrency in the types of every function that uses, with errors handled in the bind.
- So we're now taking advantage of OCaml 5.0's effect handlers and a new library Eio to shift away from the classic Lwt.

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

Shifting towards direct-style effects

```
module Make_packet_proxy
(I: Mirage_flow.S) (O: Mirage_flow.S) = struct
 let run incoming outgoing =
  let rec loop () =
   I.read incoming >>= function
   | Error err -> fail "%a" I.pp_error err
   | Ok `Eof -> Lwt.return_unit
   | Ok (`Data buf) -> begin
      O.write outgoing buf >>= function
      | Ok () -> loop ()
      | Error err -> fail "%a" O.pp_error err
     end
  in loop ()
```

Older monadic Lwt code with custom binds and functors wrapping all logic

```
module Proxy = struct
 let run incoming outgoing =
  try
    while true do
      Eio.Flow.copy incoming outgoing
    done
  with
   | End_of_file -> ()
   | Write_error err ->
       fail "%a" pp_write_error err
   | Read_error err ->
       fail "%a" pp_read_error err
end
```

Newer direct-style Eio code using native OCaml control flow due to effect handlers

Less heap allocation and more modern IO backends makes throughput faster

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

Functional Networking for Docker worked!

- **OCaml's been a solid choice** through Docker for the "invisible systems glue".
- Very little drama in production, and using **library virtual machine monitors is now a defacto standard** for similar applications on macOS and Windows.
- The 'kernel as a library' unikernel trick could apply to many emerging PL projects (like the WebAssembly Linux Interface)
- Performance started to dip, so we're now using OCaml 5.0's effect handlers and Eio. The shift is still in progress, but is so far both ergonomic and performant.
- It's still got a lot of dynamic lifetime management. But on the horizon is...

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

https://oxcaml.org

Functional Networking for Millions of Docker Desktops

Anil Madhavapeddy (speaker),
David J. Scott, Patrick Ferris, Ryan T. Gibb, Thomas Gazagnaire
ICFP 2025, Singapore, Oct 14th 2025

With thanks to all the contributors over the years including Milas Bowman, Emmanuel Briney, Ian Campbell, Mathieu Champlon, Justin Cormack, Frédéric Dalleau, Akim Demaille, Ilya Dmitrichenko, Simon Ferquel, Pierre Gayvallet, Riyaz Faizullabhoy, Christiano Haesbaert, Anca Iordache, Thomas Leonard, Richard Mortier, Terry Moschou, Rolf Neugebauer, Mindy Preston, Michael Roitzsch, Guillaume Rose, Akihiro Suda, Balraj Singh, Magnus Skjegstad, David Sheets, Sebastiaan van Stijn, Tibor Vass, Gaetan de Villele, Ryuichi Watanabe, YAMAMOTO Takashi, Jeremy Yallop, the original Docker project founder Solomon Hykes and the MirageOS and OCaml developer teams.