MirageOS

Towards a smaller and safer OS

Thomas Gazagnaire

thomas@gazagnaire.org

Tarides

OUPS
23 Mai 2018

Some Good News

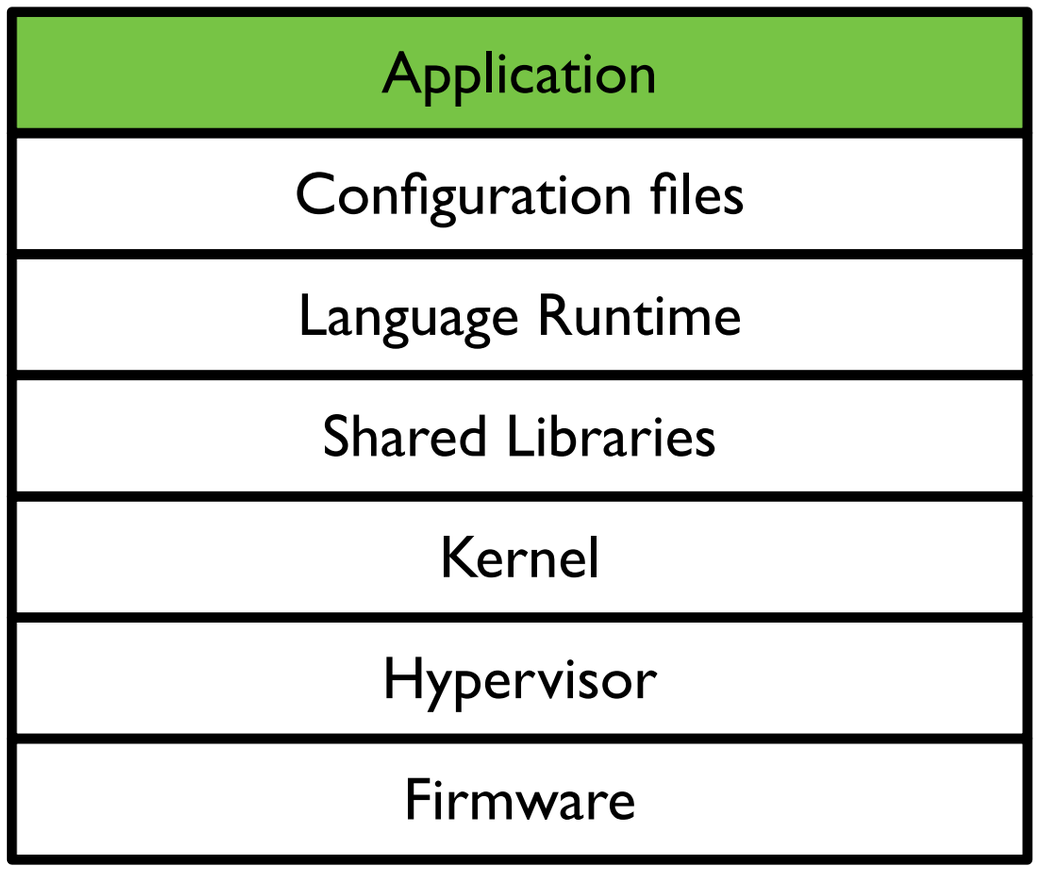
Static analysis tools are becoming mainstream: FramaC, Astrée, Coccinelle, Infer, …

- Target C/C++/Objective-C/Java applications
- Scaling from partial core logic to complete “real-world” application (current sizes: 100k — 10m loc)
- Difficult balance between scaling and power of analysis (bound checks v.s dynamic allocation, data races, …)

Some bad news

Complexity of today's traditional system software stack makes full system analysis impossible

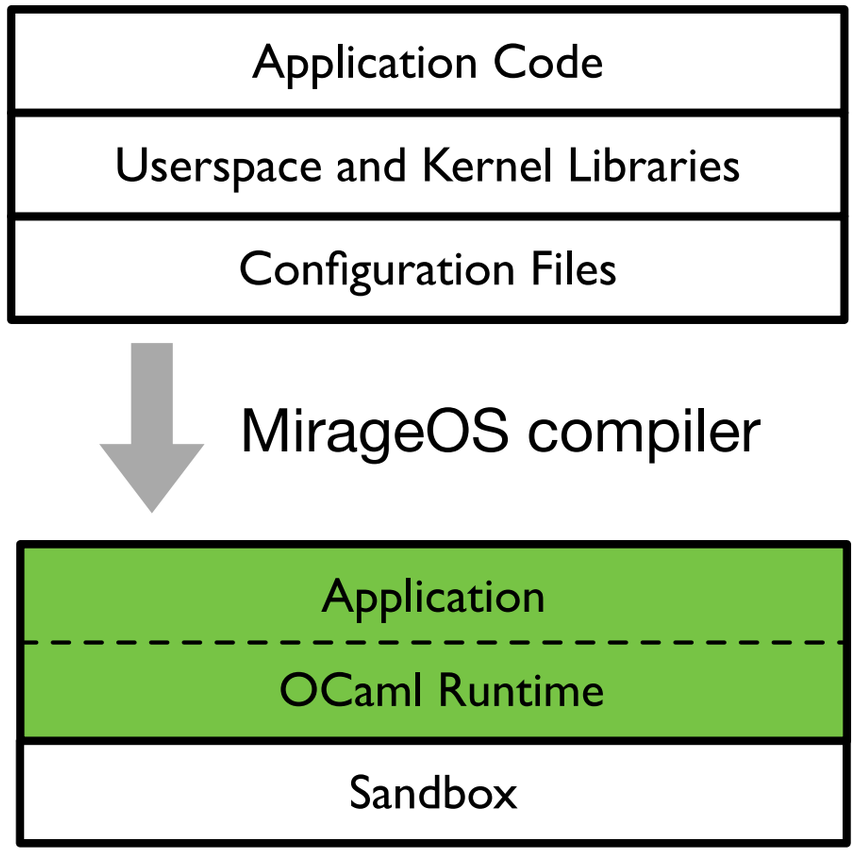
- Application code is a small % of the runtime environnement
- Runtime is historically split into abstraction layers with different communities
- In deployment environments, developers do not control that stack (vs. operators)



MirageOS

MirageOS is a **library OS** and a **compiler** which can build specialised images containing only the runtime environment which is needed by an application

- layers become <u>independent libraries</u>
- The MirageOS compiler transforms an application manifest into a specialised image
- Rely on the OCaml compiler to perform modular static analysis, dead-code elimination, etc.
- Rely on the OCaml runtime as the sole trusted runtime environment (and selected C bindings)



MirageOS

MirageOS images:

- are **self-contained**: full control on the runtime environment
- are **small**: <10 MiB (~100 KiB in some cases)
- are **resource efficient**: <16 MiB of RAM (~100 KiB in some cases)
- boot **fast**: a few ms
- flexible sandbox selection: unix binary, Xen/KVM images, etc.

MirageOS

Quick Recap

History

- **2007**: Anil Madhavapeddy's PhD: Towards a "functional" Internet at University of Cambridge
- **2008**: Citrix/XenServer libraries (Anil, Dave Scott, me)
- **2013**: MirageOS 1.0 (opam support, mirage types, CLI)
- **2014**: MirageOS 2.0 (ARM support, irmin, ocaml-tls)
- **2017**: MirageOS 3.0 (solo5 support, result type, logs, …)

Numbers

- 200+ contributors
- 100+ libraries
- Major industrial users and contributors: IBM, Citrix, Docker, Ericsson, SAP, …
- Regular meetups and hackatons
- Millions of (indirect) users via Docker for Desktop and and XenServer

UNIVERSITY OF CAMBRIDGE

THE LINUX FOUNDATION

Xen Project

MirageOS as a library OS

Signatures

`mirage_types.ml` defines module signatures for “standard” device drivers

Mirage_net_lwt.S (network card)

```
module type S = sig
  type t
  type error = private  [>`Unimplemented | `Disconnected]
  val pp_error: error Fmt.t
  type buffer = Cstruct.t
  type macaddr = Macaddr.t
  val disconnect: t -> unit Lwt.t

  val write: t -> buffer -> (unit, error) result Lwt.t
  (** [write nf buf] outputs [buf] to netfront [nf]. *)

  val writev: t -> buffer list -> (unit, error) result Lwt.t
  (** [writev nf bufs] output a list of buffers to netfront [nf] as a
      single packet. *)

  val listen: t -> (buffer -> unit Lwt.t) -> (unit, error) result Lwt.t
  (** [listen nf fn] is a blocking operation that calls [fn buf] with
      every packet that is read from the interface. The function can
      be stopped by calling [disconnect] in the device layer. *)

  val mac: t -> macaddr
  (** [mac nf] is the MAC address of [nf]. *)
end
```

Signatures

`mirage_types.ml` defines module signatures for “standard” device drivers

Mirage_kv_lwt.RO
(read-only disk)

```
module type RO = sig
  type t
  type error = private [> `Unknown_key of string]
  val pp_error: error Fmt.t
  type buffer = Cstruct.t
  val disconnect: t -> unit Lwt.t

  val read: t -> string -> int64 -> int64 -> (buffer list, error) result Lwt.t
  (** [read t key offset length] reads up to [length] bytes from the
      value associated with [key]. If less data is returned than
      requested, this indicates the end of the value. *)

  val mem: t -> string -> (bool, error) result Lwt.t
  (** [mem t key] returns [true] if a value is set for [key] in [t],
      and [false] if not so. *)

  val size: t -> string -> (int64, error) result Lwt.t
  (** Get the value size. *)

end
```

Implementations

- Module signatures are the backbone of MirageOS
- Every signature has multiple implementations
- Implementations have specialised constructors
- Implementations might define more concrete errors

Implementations

```
(** Implementation of the network interface for Unix backends. *)
include Mirage_net_lwt.S

val connect : string -> t Lwt.t
(** [connect tap] connects to the given tap interface. *)

val fd: t -> Lwt_unix.file_descr
(** [fd t] is [t]'s underneath file descriptor. *)
```

Netif (Unix tap device)

```
(** Xen Netfront interface for Ethernet I/O. *)
module Netfront = struct
  include Mirage_net_lwt.S
  val connect : int -> t Lwt.t
  (** [connect i] connects to the virtual network card number [i]. *)
end

(** Xen Netback interface for Ethernet I/O. *)
module Netback = struct
  include Mirage_net_lwt.S
  val connect: domid:int -> id:int -> t Lwt.t
  (** [make ~domid ~id] connects to the virtual network card number
      [i] in the VM [domid]. *)
end
```

Netif (xen netfront/netback)

Implementations

Implementations can be functors with device signature as parameters. Follow same rules as normal implementation regarding errors and constructors.

```
module Make ( N:Mirage_net_lwt.S) : sig
  include Mirage_ethif_lwt.S with type netif = N.t

  val connect : ?mtu:int -> netif -> t Lwt.t
  (** [connect ?mtu netif] connects an ethernet layer on top of the raw
      network device [netif]. The Maximum Transfer Unit may be set via the
      optional [?mtu] parameter, otherwise a default value of 1500 will be
      used. *)
end
```

Ethif (ethernet layer)

Available libraries

Domaine	Bibliothèques
Interfaces Réseau	tuntap, vmnet, rawlink
Protocoles Réseau	ethernet, ARP, ICMP, IPv4, IPv6, UCP, TCP, DHCP, DNS, TFTP, HTTP
Protocoles de Stockage	FAT32, Git, tar, AES-CCM, ramdisk, B-trees
Sécurité	x509, ASN1, TLS, OTR
Crypto	MD5, SHA1, SHA224, SHA256, SHA384, SHA512, BLAKE2B, BLAKE2S, RMD160, 3DES, AES, DH, DSA, RSA, Fortuna, ECB/CBC/CCM/GCM modes

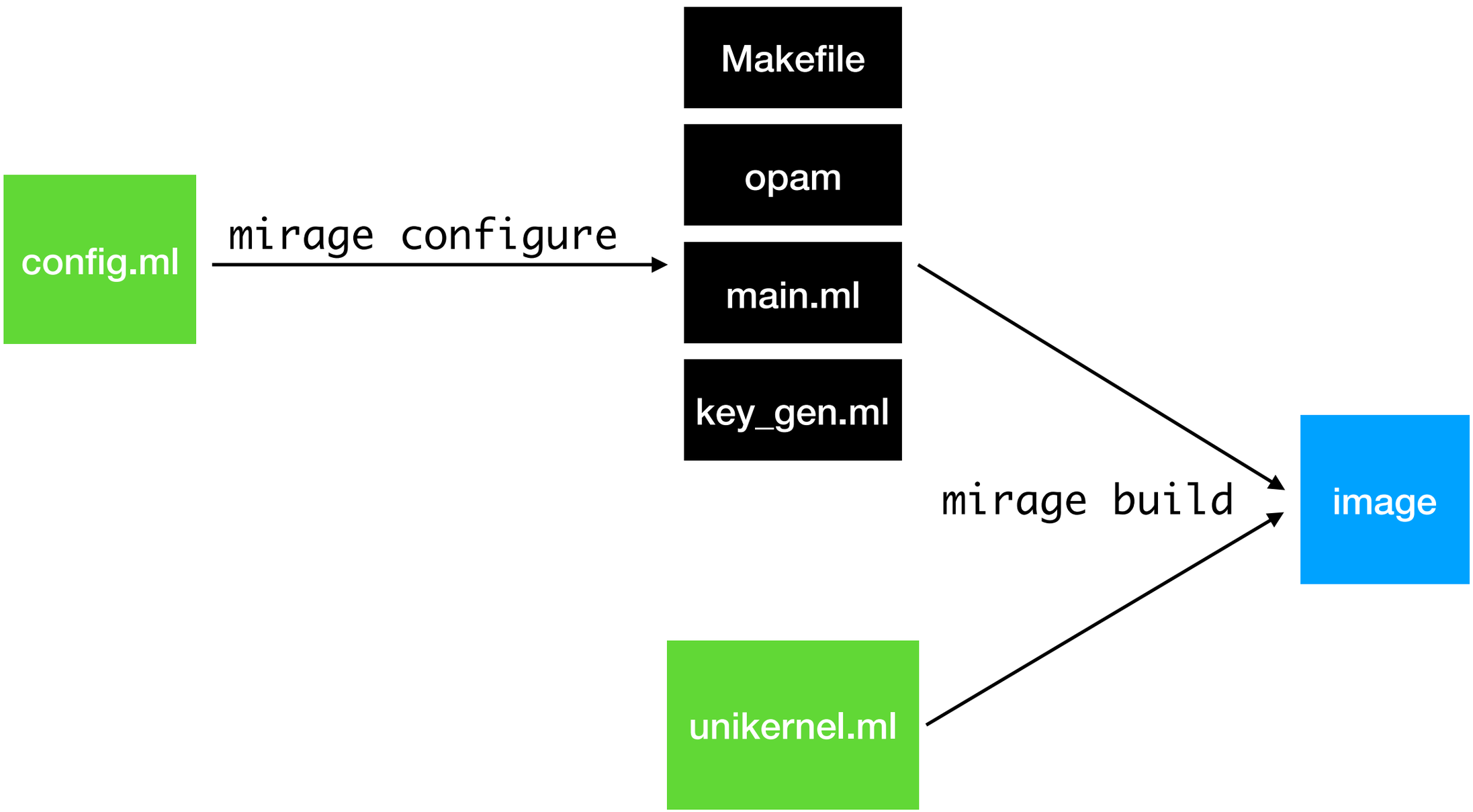
TLS: “rigorous engineering” e.g. same pure code to generate test oracles, verify oracle against real-world TLS traces and the real implementation

B-trees: code extracted from Isabelle/HOL

MirageOS as a compiler

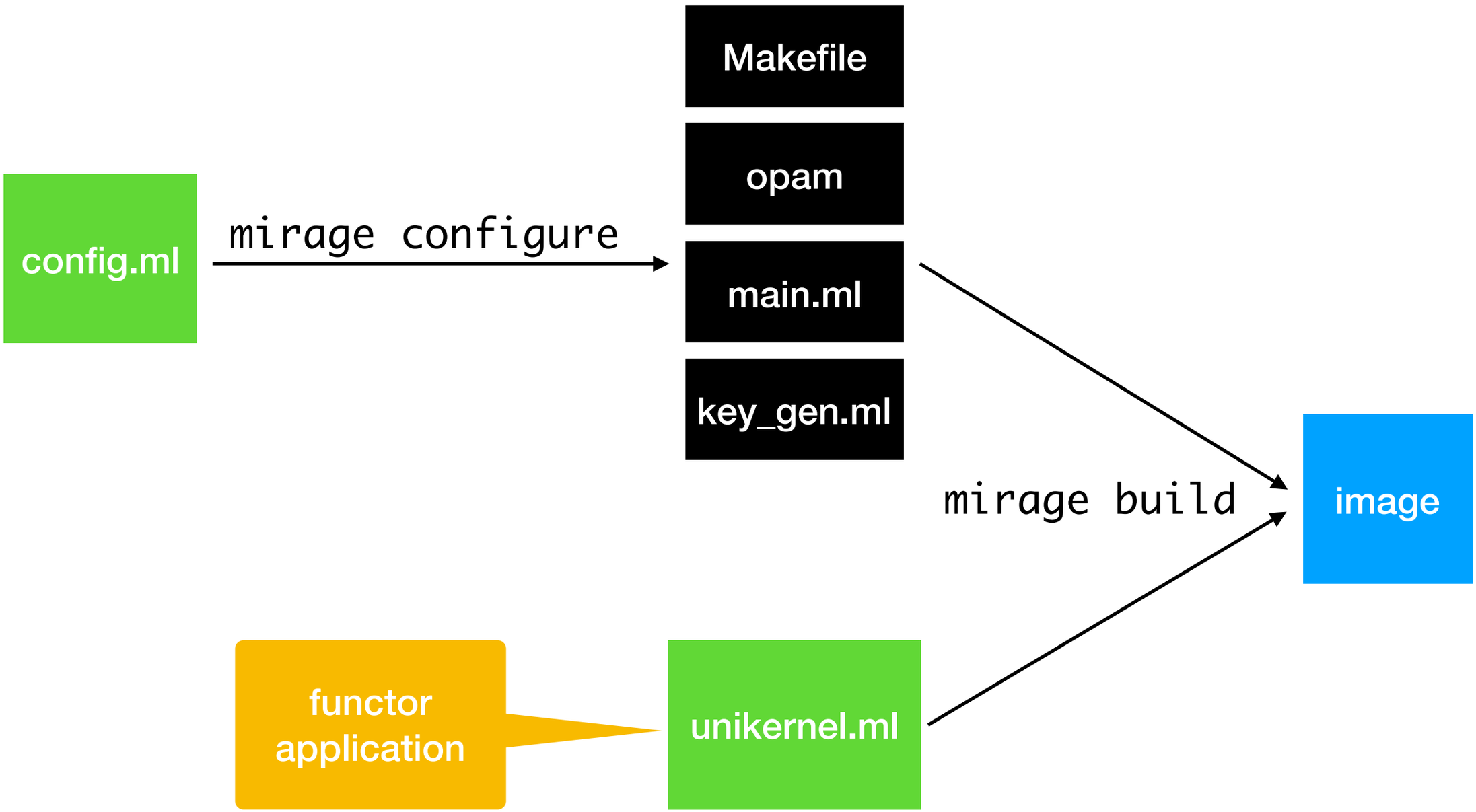
MirageOS compiler

multi-stage pipeline



MirageOS compiler

multi-stage pipeline



Application Functor

- A MirageOS application is a functor:
  - using the standard signatures as parameters
  - with a start function
- Time: abstract driver for timers
- Key_gen: typed cross-stage persistence

```
open Lwt.Infix

module Hello (Time : Mirage_time_lwt.S) = struct

  let start () =
    let aux () =
      print_endline ("hello " ^ Key_gen.hello ());
      Time.sleep_ns (Duration.of_sec 1)
    in
    let rec loop = function
      | None   -> aux () >>= fun () -> loop None
      | Some 0 -> Lwt.return_unit
      | Some n -> aux () >>= fun () -> loop (Some (n-1))
    in
    loop (Key_gen.n ())

end
```

Application Functor

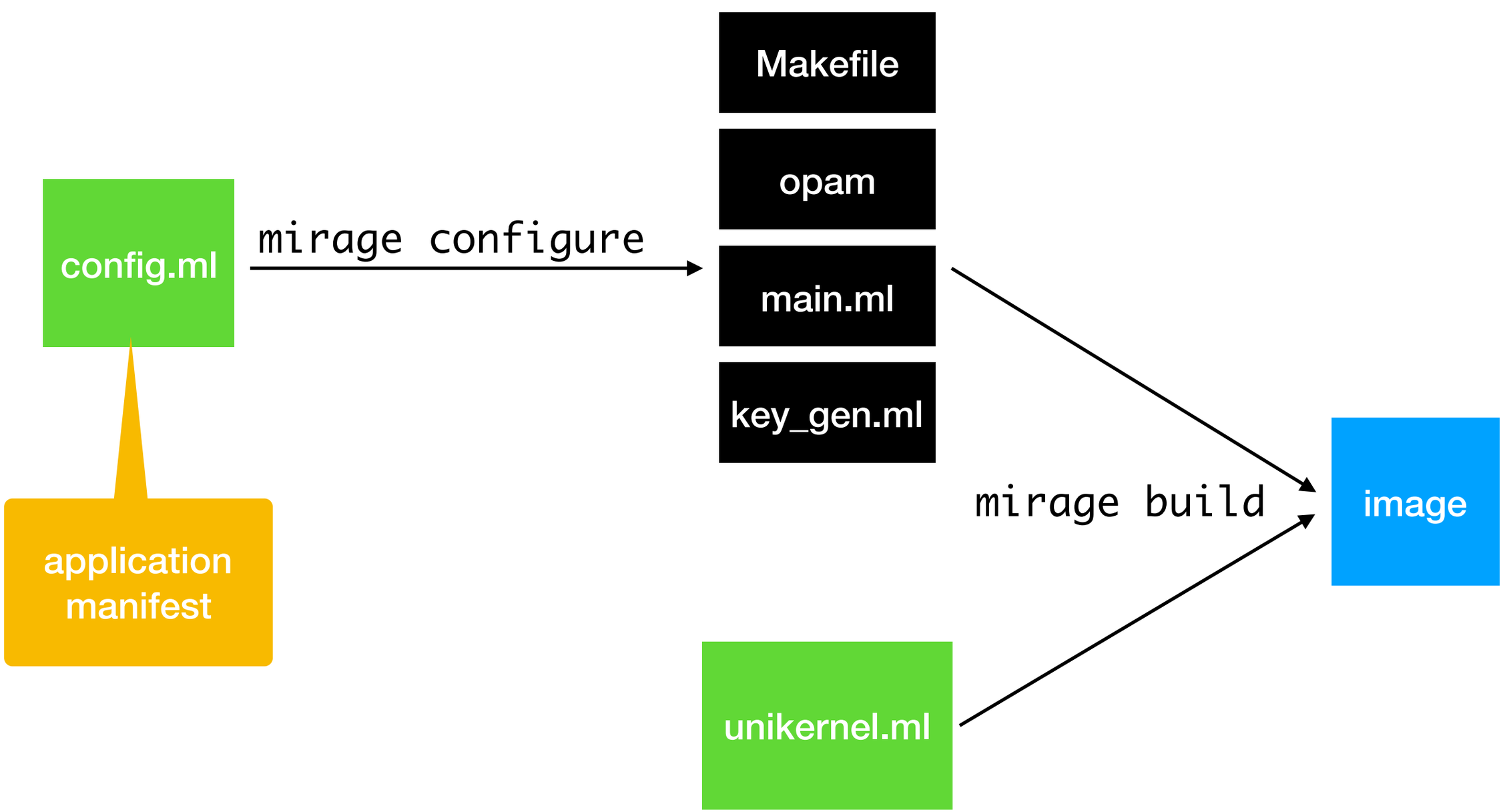
The mirage website needs:

- 1 TCP/IPv4 stack
- 3 read-only key/value stores
  - private key
  - raw contents
  - templates
- 1 clock device to get the current time

```
module Make
    (S: Mirage_stack_lwt.V4)
    (KEYS: Mirage_types_lwt.KV_RO)
    (FS: Mirage_types_lwt.KV_RO)
    (TMPL: Mirage_types_lwt.KV_RO)
    (Clock : Mirage_types.PCLOCK)
= struct

  let start stack keys fs tmpl clock = ...

end
```

MirageOS compiler

multi-stage pipeline



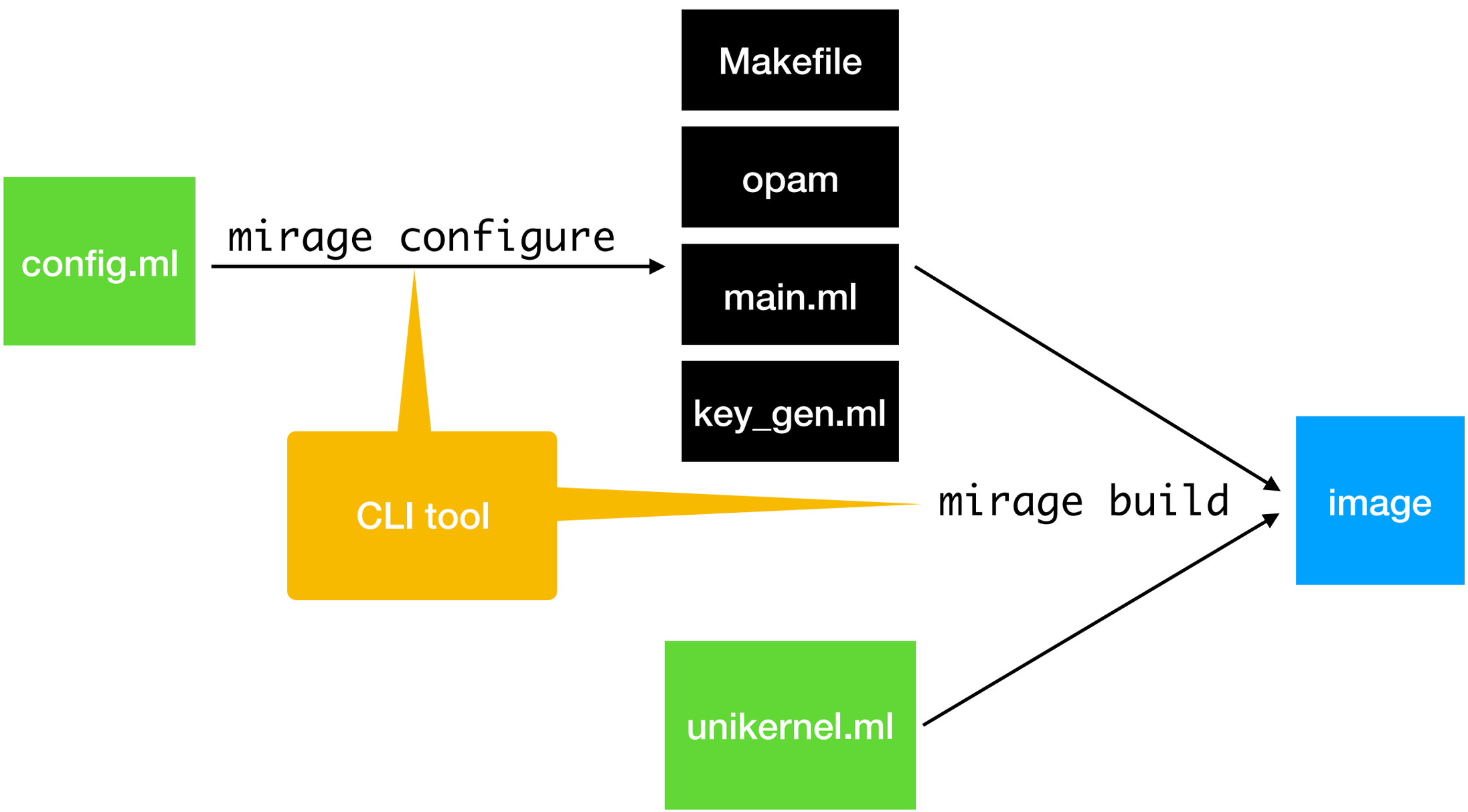
Application Manifest

- Application manifest in OCaml
- Use a eDSL to describe functors composition and configuration keys
- Describe the main application (reflect `unikernel.ml`)
- Use generic devices which will adapt to the deployment target:
  - network: Static IP vs. DHCP
  - KV/RO: block device, FAT32, crunched files, irmin, …
  - clock: unix/xen default device

```
open Mirage

let title_key =
  let doc =
    Key.Arg.info ["title"]
      ~doc:"Website title."
      ~env:"WWW_TITLE"
  in
  let a = Key.Arg.(opt (some string) None doc) in
  Key.(create "title" a)

let keys     = [Key.abstract title_key]
let packages = [package "cow"; package "cowabloga"]

let http =
  foreign ~packages ~keys "Unikernel.Make"
    (stackv4 @-> kv_ro @-> kv_ro @-> kv_ro @-> pclock @-> job)

let fs_key = Key.(value @@ kv_ro ())
let filesfs = generic_kv_ro ~key:fs_key "../files"
let tmplfs  = generic_kv_ro ~key:fs_key "../tmpl"

let () =
  register "www" [
    http
    $ generic_stackv4 default_network
    $ fs_key
    $ filesfs
    $ tmplfs
    $ default_posix_clock
  ]
```

MirageOS compiler

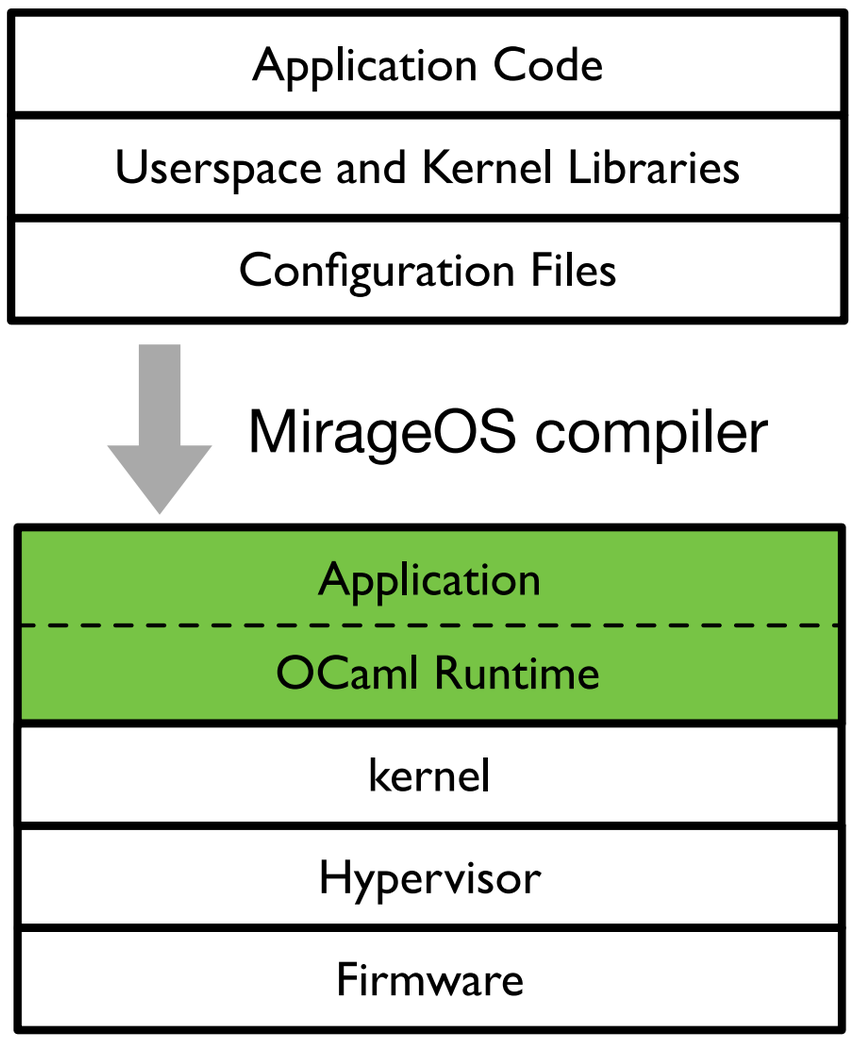
multi-stage pipeline



Demo

MirageOS compiler

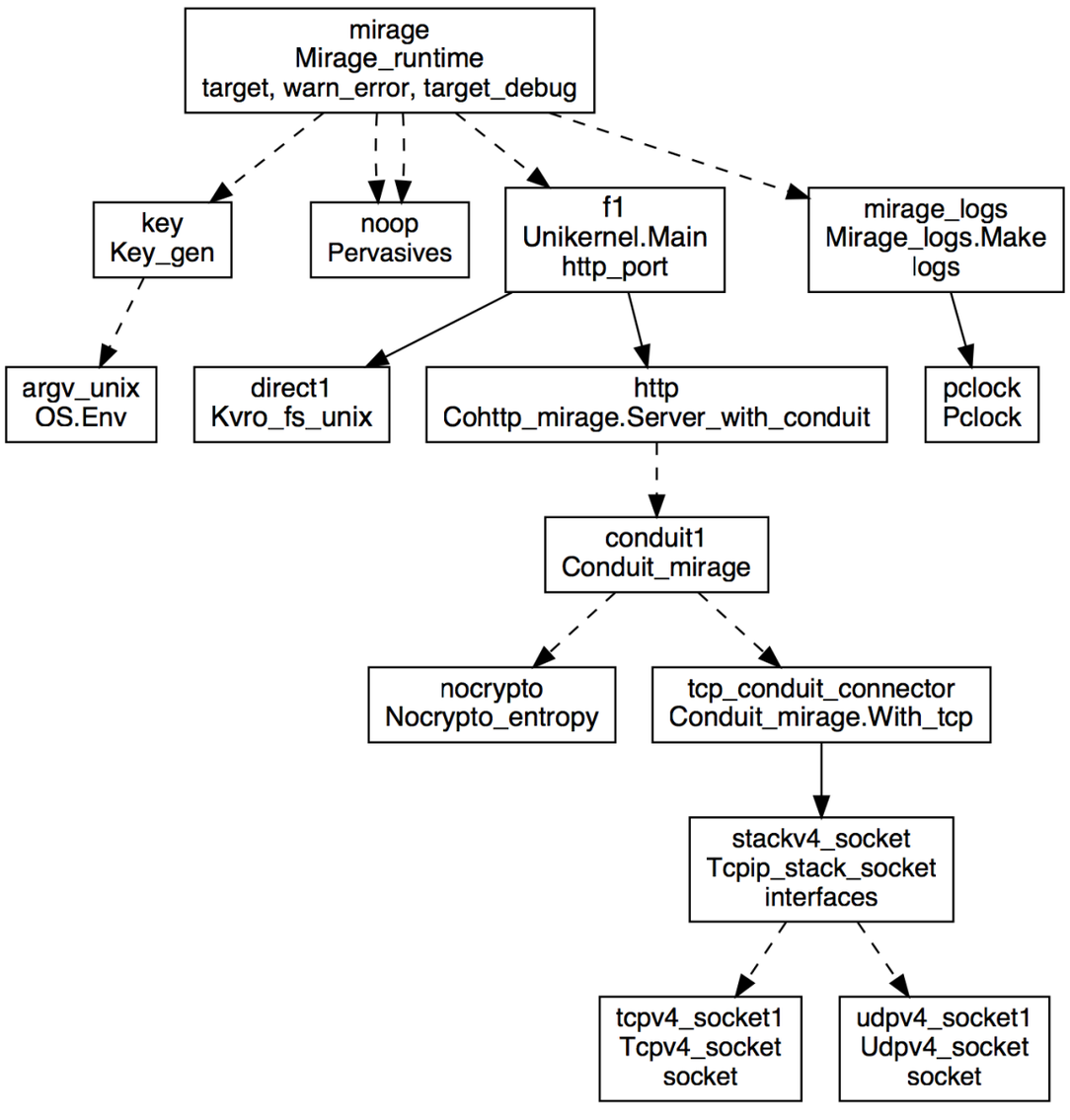
“dev mode: a.k.a. I ❤️ Linux syscalls”



User-space and OS libraries

“dev mode”

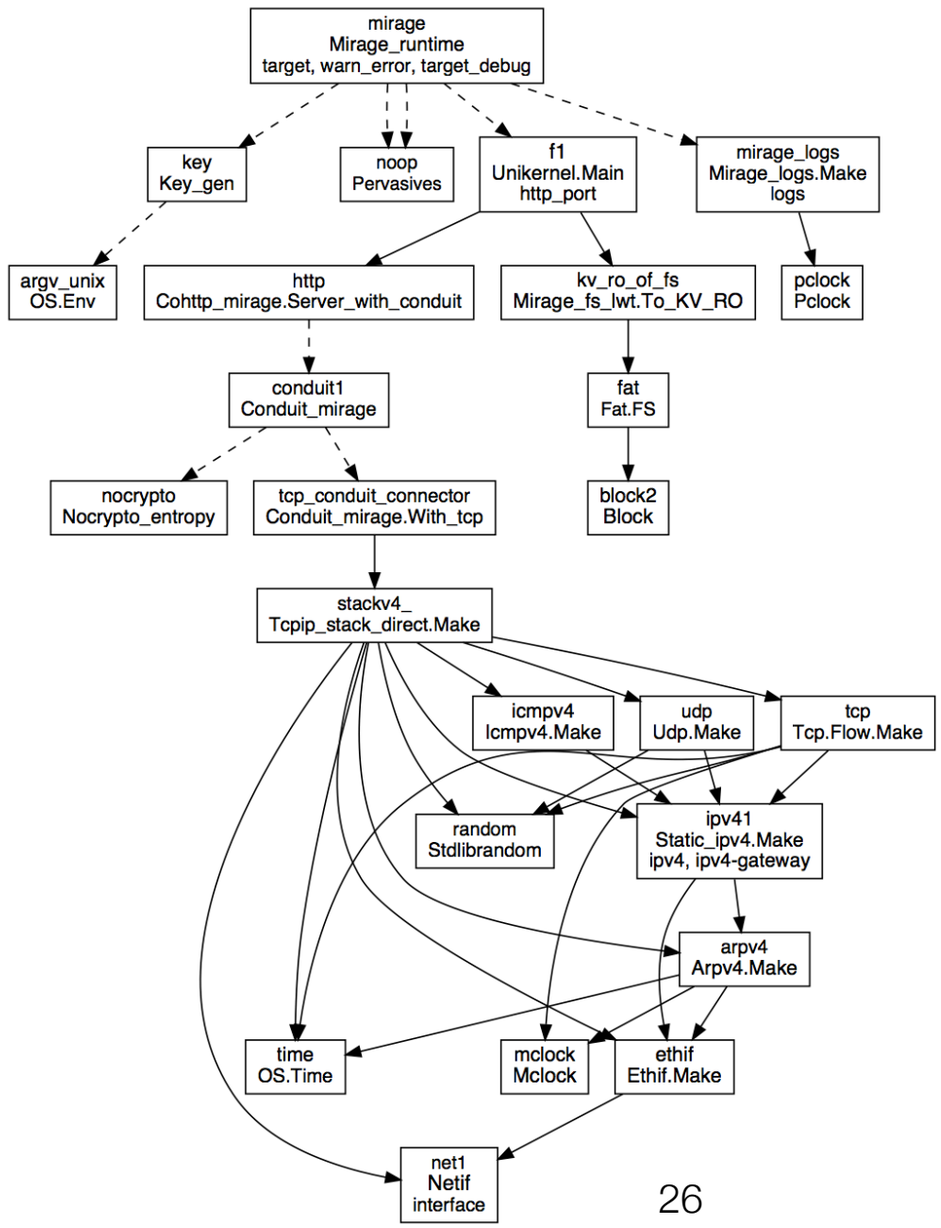
```
mirage configure --target=unix --net=socket --kv_ro=direct
```



User-space and OS libraries

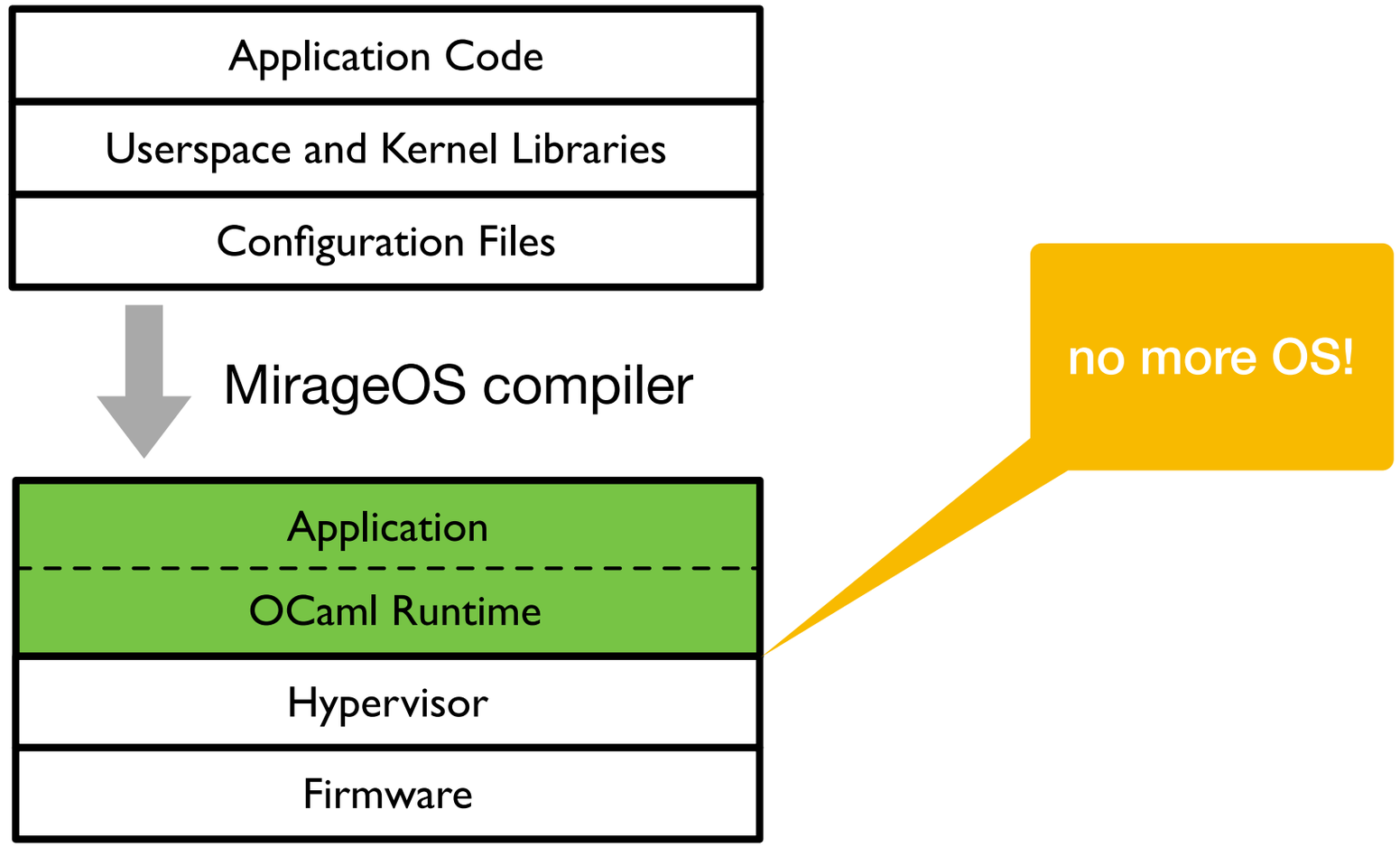
“dev mode + MirageOS TCP/IP stack + FAT block device”

```
mirage configure --target=unix --net=direct --kv_ro=fat
```



MirageOS compiler

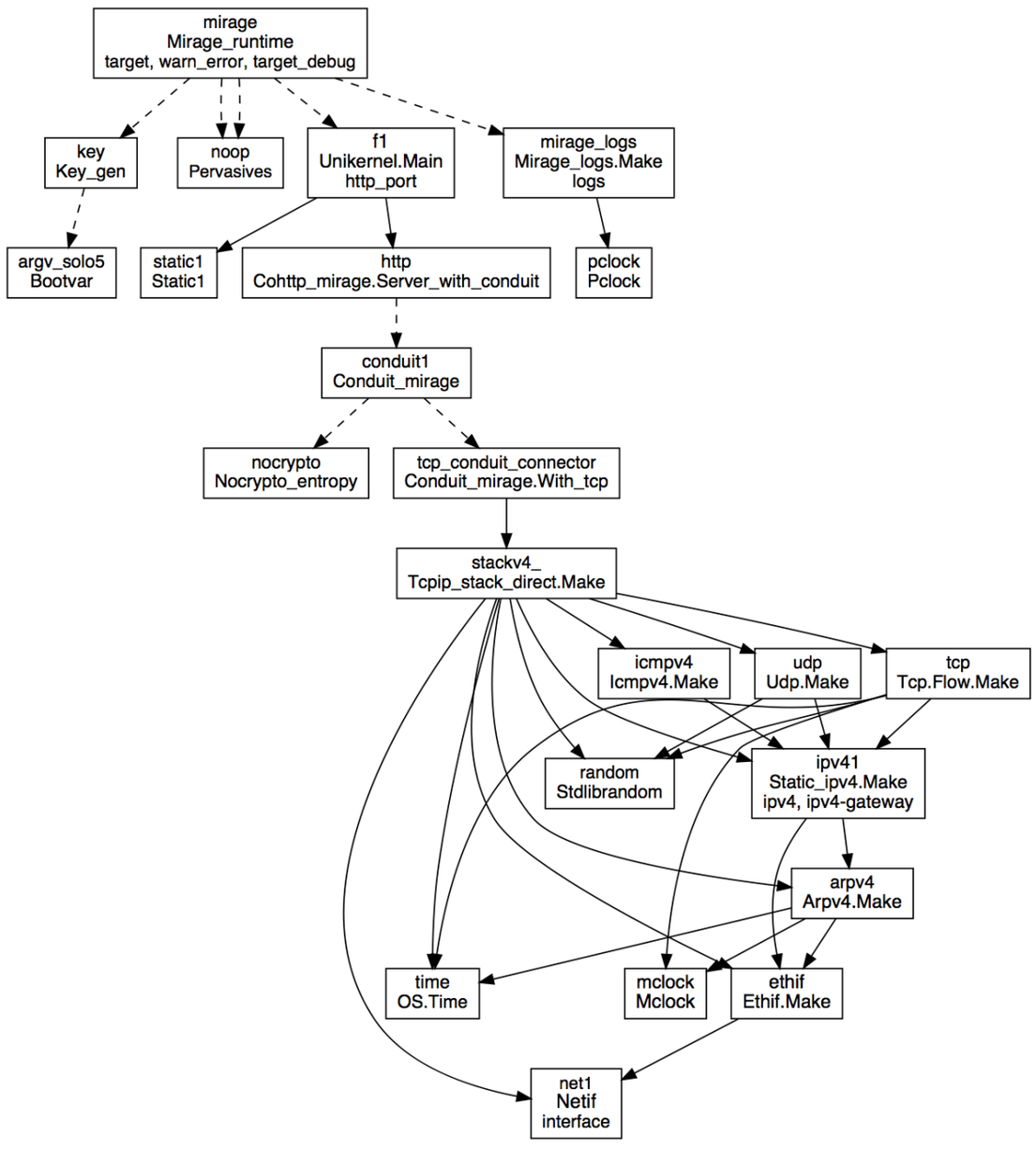
"deployment mode"



User-space and OS libraries

“deployment mode (xen)”

```
mirage configure --target=ukvm --net=direct --kv_ro=crunch
```



Conclusion

Summary

- MirageOS is a modular operating system written in OCaml
- End-goal is to allow individual libraries to be extracted/verified/certified individually and composed together
- The only way to build end-to-end “high-insurance” services (application + full runtime environment)?

MirageOS in 2018

Tarides

- A company created in Paris to commercially support MirageOS and to promote its use in the industry
  contact: Thomas Gazagnaire

robur

- A non-profit organisation created in Berlin to work on secure infrastructure (internet services) using MirageOS
  contact: Hannes Mehnert

OCaml Labs

- research on MirageOS continues in Cambridge: IoT, embedded software, privacy-preserving systems, data-science, etc
  contact: Anil Madhavapeddy

Join the community!

mirageos-devel@lists.xenproject.org
https://mirage.io/
https://discuss.ocaml.org/tags/mirageos

6th MirageOS hack retreat

We invite you to participate in the sixth **MirageOS** hack retreat! The goal is to sync various MirageOS subprojects, start new ones, and fix bugs.

- *When?* 3rd October (arrival) - 10th October (departure) 2018
- *Where?* Marrakech, Morocco at **Priscilla, Queen of the Medina.**
- *How much?* likely 350 EUR★, accommodation and food (full board) included.
- *How do I register?* Register by sending a mail to **retreat2018@nqsb.io by August 15th, 2018** including:
  - OCaml and MirageOS experience;
  - Project(s) you're interested to work on; and
  - Dietary restrictions
- *Who should participate?* Everybody interested in advancing MirageOS.
- *How big?* We have only limited space (30 people). Selection will be done by various diversity criteria.
- *How should I behave while there?* Be kind and empathetic to others; do not harrass or threaten anyone. If you make others unsafe, you may be asked to leave.

★: If you cannot afford this, please contact us directly (at **retreat2018@nqsb.io**).

Thank you!

Any questions?