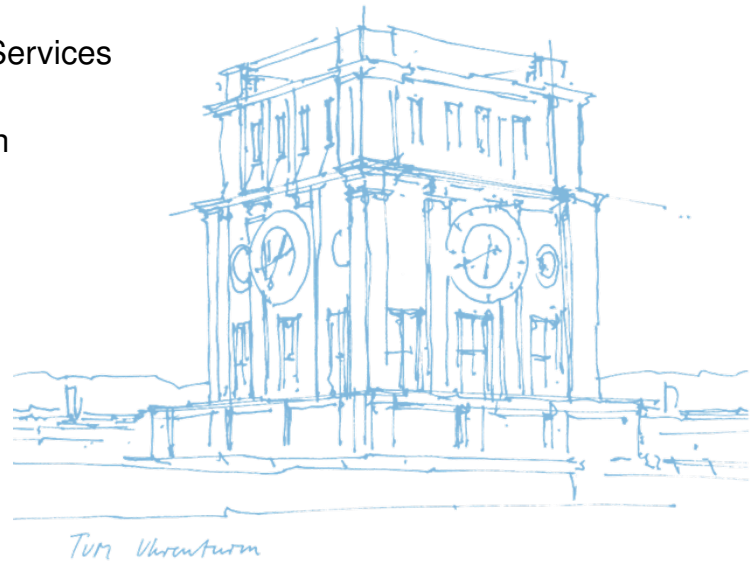
Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

PCIe and DMA in MirageOS

Fabian Bonk

Wednesday 20th May, 2020

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich



What is MirageOS?

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

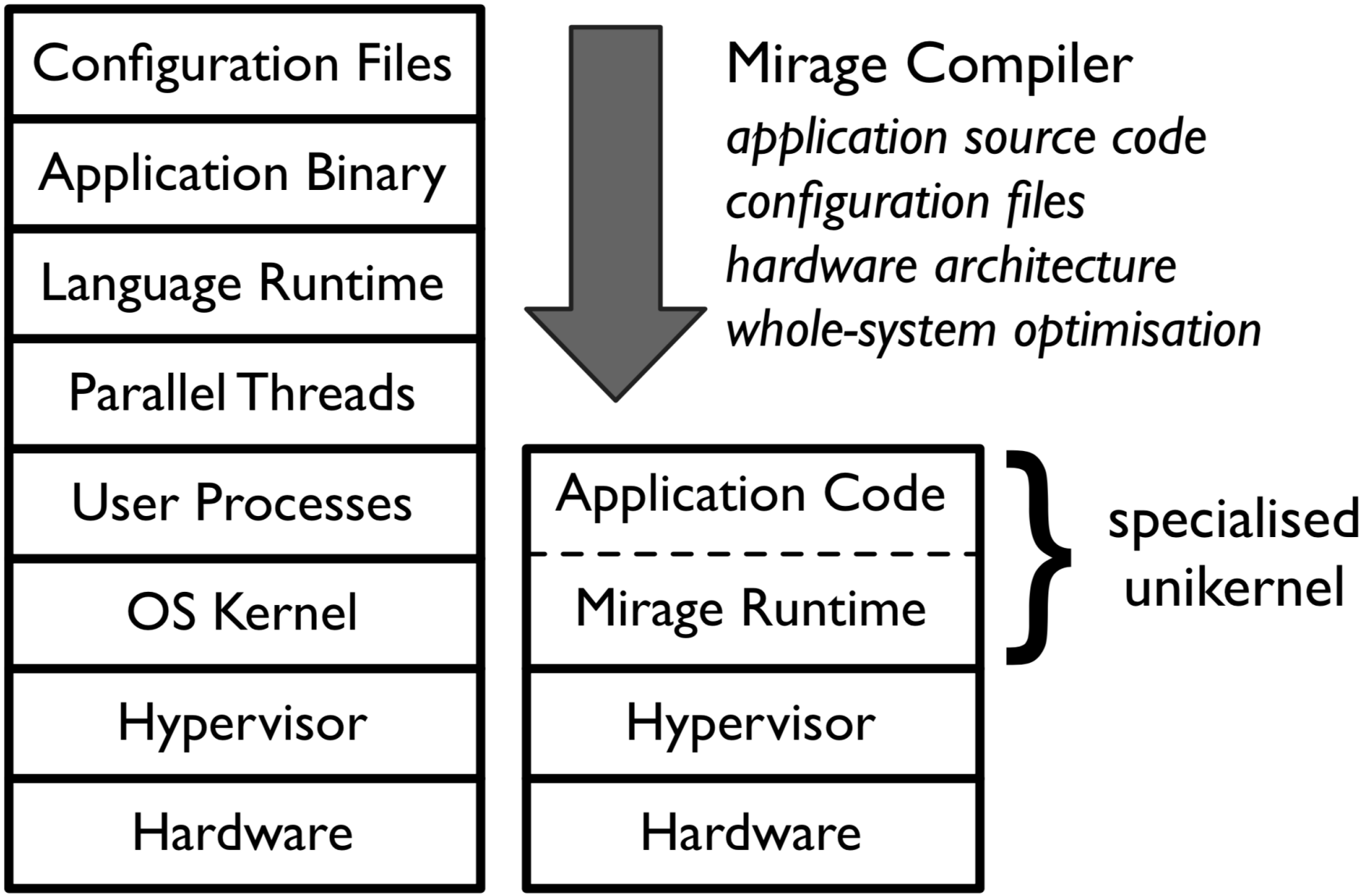
Unikernels

What's a Unikernel?

- Entire application compiled into bootable VM image
- Include necessary operating system functionality via libraries

Unikernels

Unikernels vs. virtual machines



[1]

Unikernels

Why Unikernels?

- high degree of separation
- low resource usage
- flexible runtime(s) (run on hypervisors, standard OS, microcontrollers)
- safety benefits of high-level languages
- fewer loc $\rightarrow$ fewer bugs

MirageOS

MirageOS

OCaml unikernel operating system

`https://mirage.io/`

- written in OCaml
- generates Xen (incl. QubesOS) and Solo5 (KVM) Unikernels
- can also generate standard executables (Linux, macOS, ...)
- ~~172~~ 173 repos on GitHub

MirageOS

OCaml unikernel operating system

`https://mirage.io/`

- written in OCaml
- generates Xen (incl. QubesOS) and Solo5 (KVM) Unikernels
- can also generate standard executables (Linux, macOS, ...)
- ~~172~~ 173 repos on GitHub

Example: Echo server

```
open Lwt.Infix

module Main (S : Mirage_stack.V4) = struct
  (* RFC 862 - read payloads and repeat them back *)
  let rec echo flow =
    S.TCPV4.read flow >>= function
    | Error _
    | Ok `Eof -> S.TCPV4.close flow
    | Ok `Data buf ->
      S.TCPV4.write flow buf >>= function
      | Error _ -> S.TCPV4.close flow
      | Ok () -> echo flow

  let start s =
    S.listen_tcpv4 s ~port:7 echo;
    S.listen s
end
```

Layering

Q: What is a HTTPS stack really?

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Q: What is a TCP stack really?

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Q: What is a TCP stack really?

A: Some code on top of an IP stack!

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Q: What is a TCP stack really?

A: Some code on top of an IP stack!

Q: What is an IP stack really?

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Q: What is a TCP stack really?

A: Some code on top of an IP stack!

Q: What is an IP stack really?

A: Some code on top of an Ethernet stack!

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Q: What is a TCP stack really?

A: Some code on top of an IP stack!

Q: What is an IP stack really?

A: Some code on top of an Ethernet stack!

Q: What is an Ethernet stack really?

Layering

Q: What is a HTTPS stack really?

A: Some code on top of a TLS stack!

Q: What is a TLS stack really?

A: Some code on top of a TCP stack!

Q: What is a TCP stack really?

A: Some code on top of an IP stack!

Q: What is an IP stack really?

A: Some code on top of an Ethernet stack!

Q: What is an Ethernet stack really?

A: Some code on top of a network device!

Layering

Let’s do some functional programming!

Layering

Let's do some functional programming!

HTTPS stack :

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack :

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack :

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

IP stack :

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

IP stack : *Ethernet interface* $\rightarrow$ *IP interface*

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

IP stack : *Ethernet interface* $\rightarrow$ *IP interface*

Ethernet stack :

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

IP stack : *Ethernet interface* $\rightarrow$ *IP interface*

Ethernet stack : *Network device* $\rightarrow$ *Ethernet interface*

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

IP stack : *Ethernet interface* $\rightarrow$ *IP interface*

Ethernet stack : *Network device* $\rightarrow$ *Ethernet interface*

Network device :

Layering

Let's do some functional programming!

HTTPS stack : *TLS interface* $\rightarrow$ *HTTP interface*

TLS stack : *TCP interface* $\rightarrow$ *TLS interface*

TCP stack : *IP interface* $\rightarrow$ *TCP interface*

IP stack : *Ethernet interface* $\rightarrow$ *IP interface*

Ethernet stack : *Network device* $\rightarrow$ *Ethernet interface*

Network device : *magic*

Layering

Network stack, assemble!

```
module HTTPS_Interface = HTTP (TLS (TCP (IP (Ethernet (TAP_device)))))
```

Layering

Network stack, assemble!

```
module HTTPS_Interface = HTTP (TLS (TCP (IP (Ethernet (TAP_device)))))
```

or

```
module HTTPS_Interface = HTTP (TLS (TCP_socket))
```

Layering

Network stack, assemble!

```
module HTTPS_Interface = HTTP (TLS (TCP (IP (Ethernet (TAP_device)))))
```

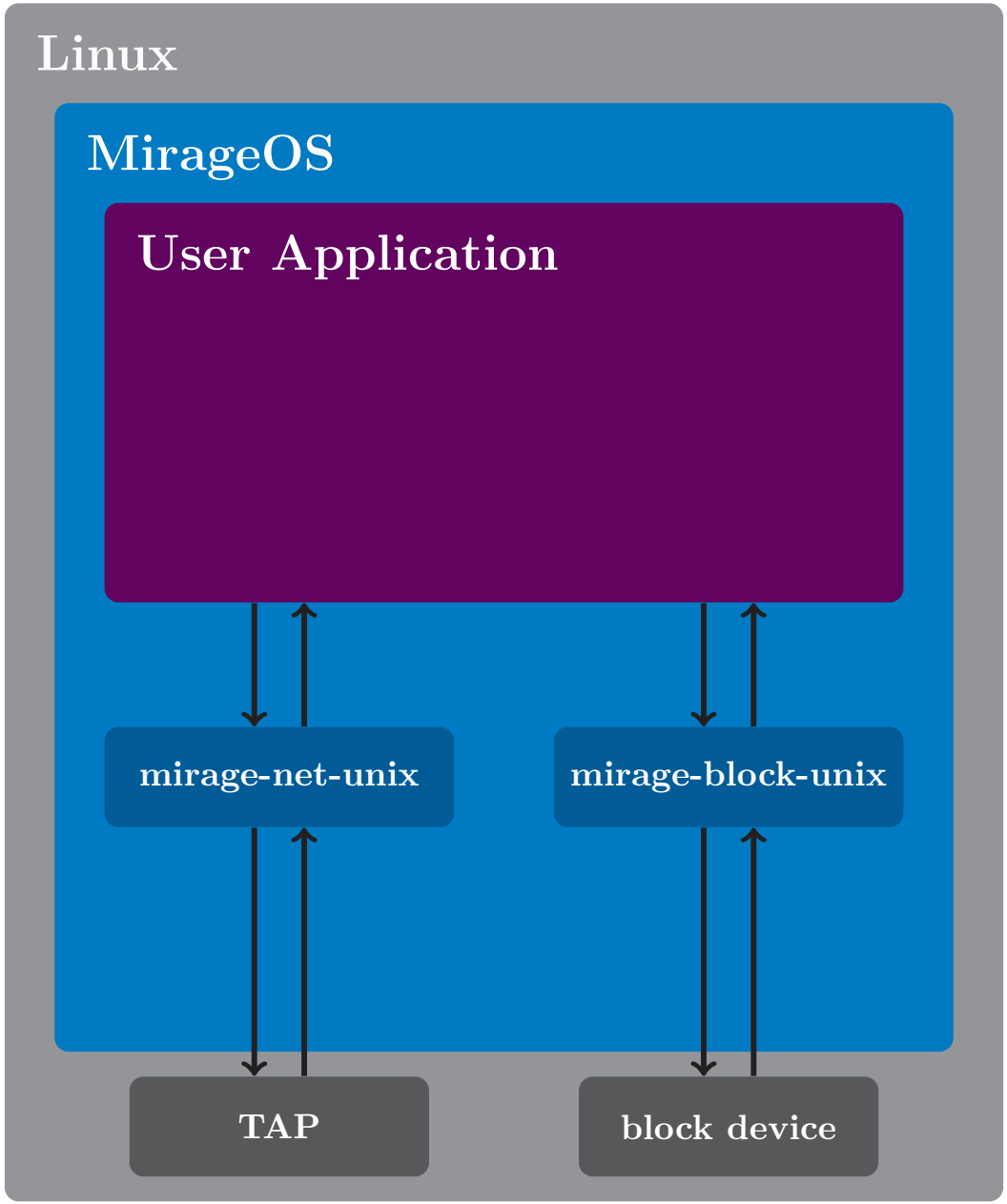
or

```
module HTTPS_Interface = HTTP (TLS (TCP_socket))
```

How about this?

```
module HTTPS_Interface = HTTP (TLS (TCP (IP (Ethernet (Network_driver (PCIe_device))))))
```

Structure


Linux
MirageOS
User Application
mirage-net-unix
mirage-block-unix
TAP
block device

Example: Echo server

```
open Lwt.Infix

module Main (S : Mirage_stack.V4) = struct
  (* RFC 862 - read payloads and repeat them back *)
  let rec echo flow =
    S.TCPV4.read flow >>= function
    | Error _
    | Ok `Eof -> S.TCPV4.close flow
    | Ok `Data buf ->
      S.TCPV4.write flow buf >>= function
      | Error _ -> S.TCPV4.close flow
      | Ok () -> echo flow

  let start s =
    S.listen_tcpv4 s ~port:7 echo;
    S.listen s
end
```

Example: Echo server

How to build:

Build a normal binary and use a TAP device and the OCaml network stack:

```
$ mirage configure -t unix --net direct && make
```

Build a normal binary and use the OS network stack:

```
$ mirage configure -t unix --net socket && make
```

Build a standalone Unikernel for deployment on Solo5/KVM:

```
$ mirage configure -t hvt && make
```

Goal

```
$ mirage configure -t hvt --net <some-driver> --pci 0000:ab:cd.e && make
```

Add necessary functionality for writing device drivers for MirageOS.

Goal

Approach

- add PCIe device category to MirageOS ecosystem
- mirage-pci interface library
- mirage-pci-solo5 wrapper library
- mirage-pci-unix wrapper library
- modify ixy.ml

Solo5

What is Solo5?

A sandboxed execution environment for unikernels

run unikernels on:

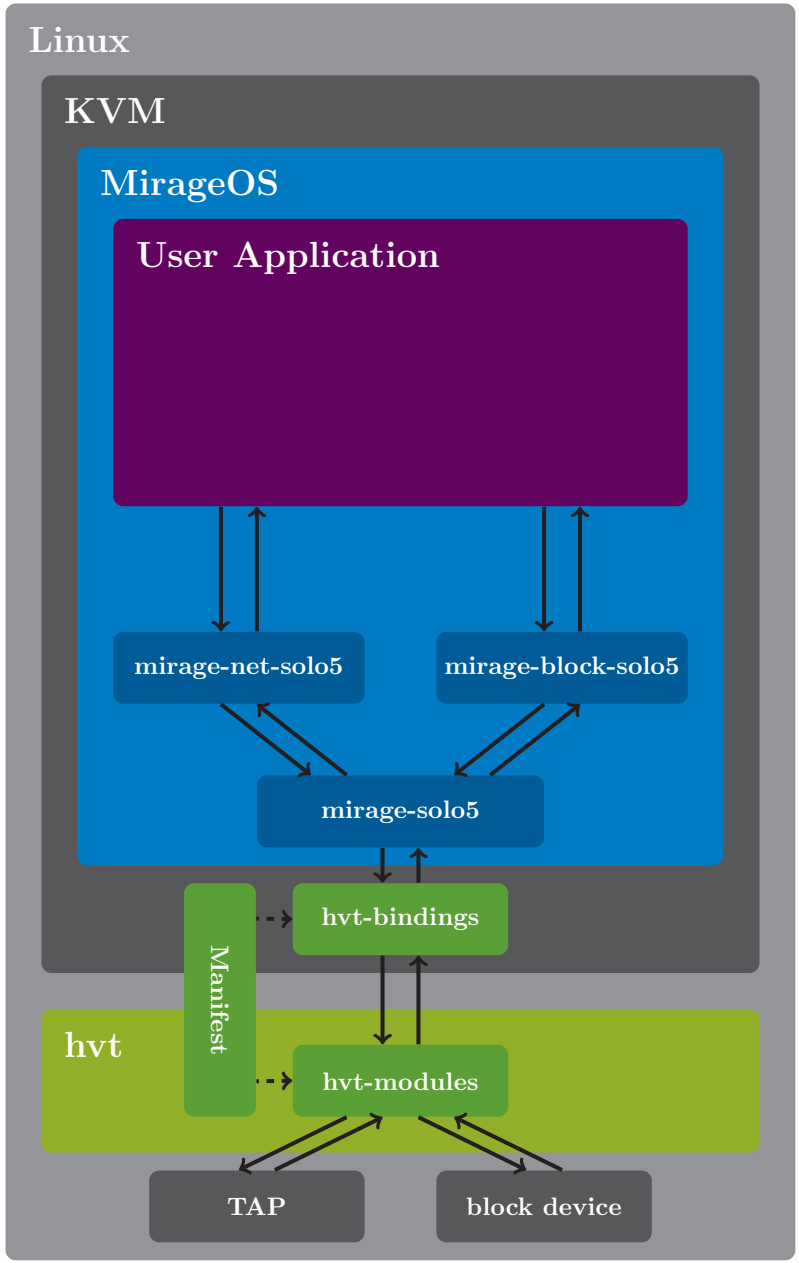
- Linux KVM
- Linux seccomp
- FreeBSD/OpenBSD vmm
- Muen
- Genode

Solo5

Solo5-hvt

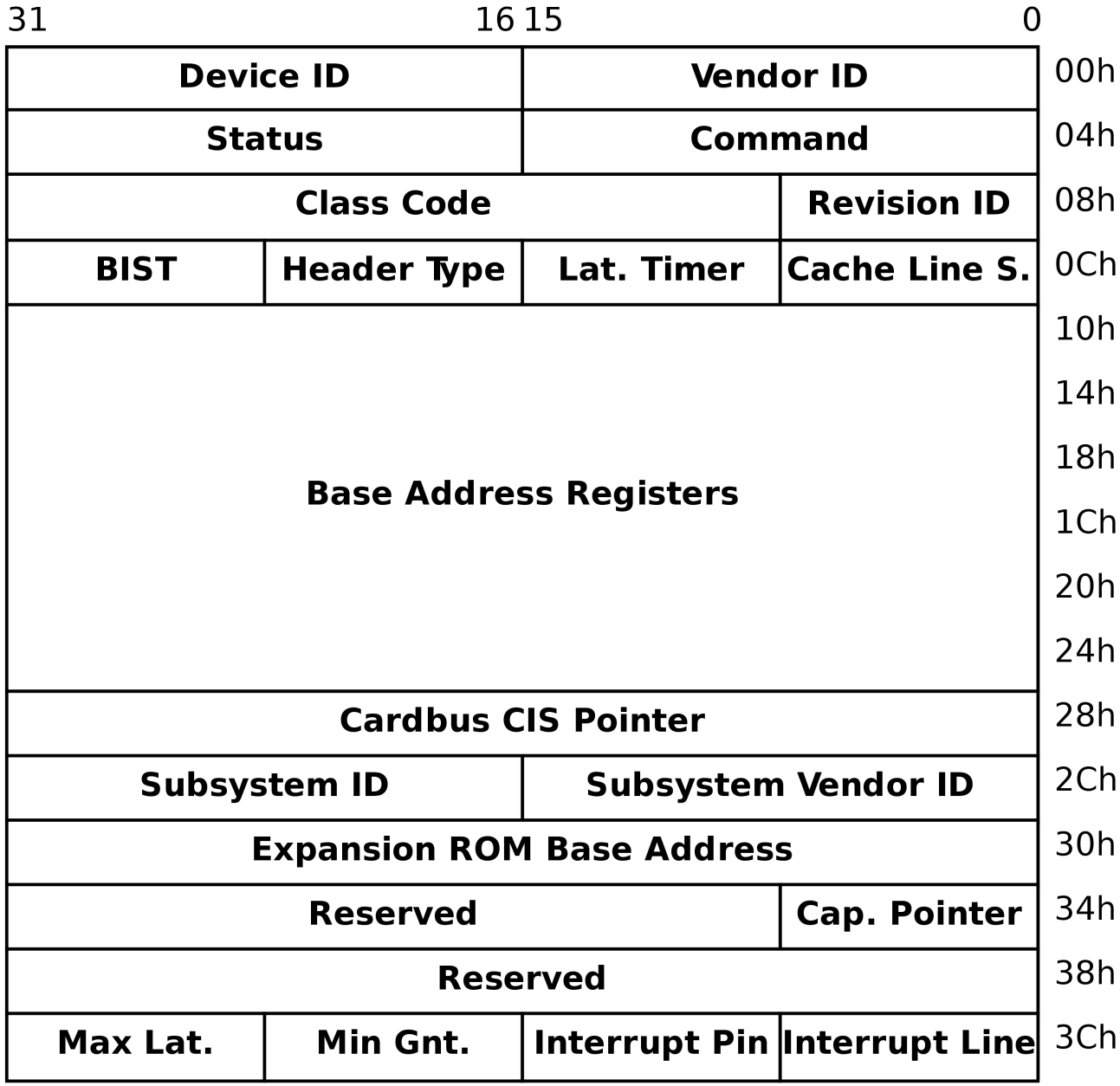
- hardware virtualized tender
- create KVM virtual machine
- load ELF binary
- pass messages between unikernel and host devices

Solo5


Linux
KVM
MirageOS
User Application
mirage-net-solo5
mirage-block-solo5
mirage-solo5
hvt-bindings
Manifest
hvt
hvt-modules
TAP
block device

PCIe

```
$ ls -l /sys/bus/pci/devices/$DEVICE/
total 0
-r--r--r-- 1 root root 4096 Jan  7 13:49 class
-rw-r--r-- 1 root root  256 Jan  7 13:49 config
lrwxrwxrwx 1 root root    0 Jan  7 13:49 driver -> ../../../bus/pci/drivers/some-driver
-rw------- 1 root root   32 Jan  7 13:49 resource0
-rw------- 1 root root 8192 Jan  7 13:49 resource1
-r--r--r-- 1 root root 4096 Jan  7 13:49 vendor
# ...
```


31
16 15
0
Device ID
Vendor ID
00h
Status
Command
04h
Class Code
Revision ID
08h
BIST
Header Type
Lat. Timer
Cache Line S.
0Ch
Base Address Registers
10h
14h
18h
1Ch
20h
24h
Cardbus CIS Pointer
28h
Subsystem ID
Subsystem Vendor ID
2Ch
Expansion ROM Base Address
30h
Reserved
Cap. Pointer
34h
Reserved
38h
Max Lat.
Min Gnt.
Interrupt Pin
Interrupt Line
3Ch

DMA

What is DMA?

Direct Memory Access

Flip a magic bit in `/sys/bus/pci/devices/$DEVICE/config` to enable

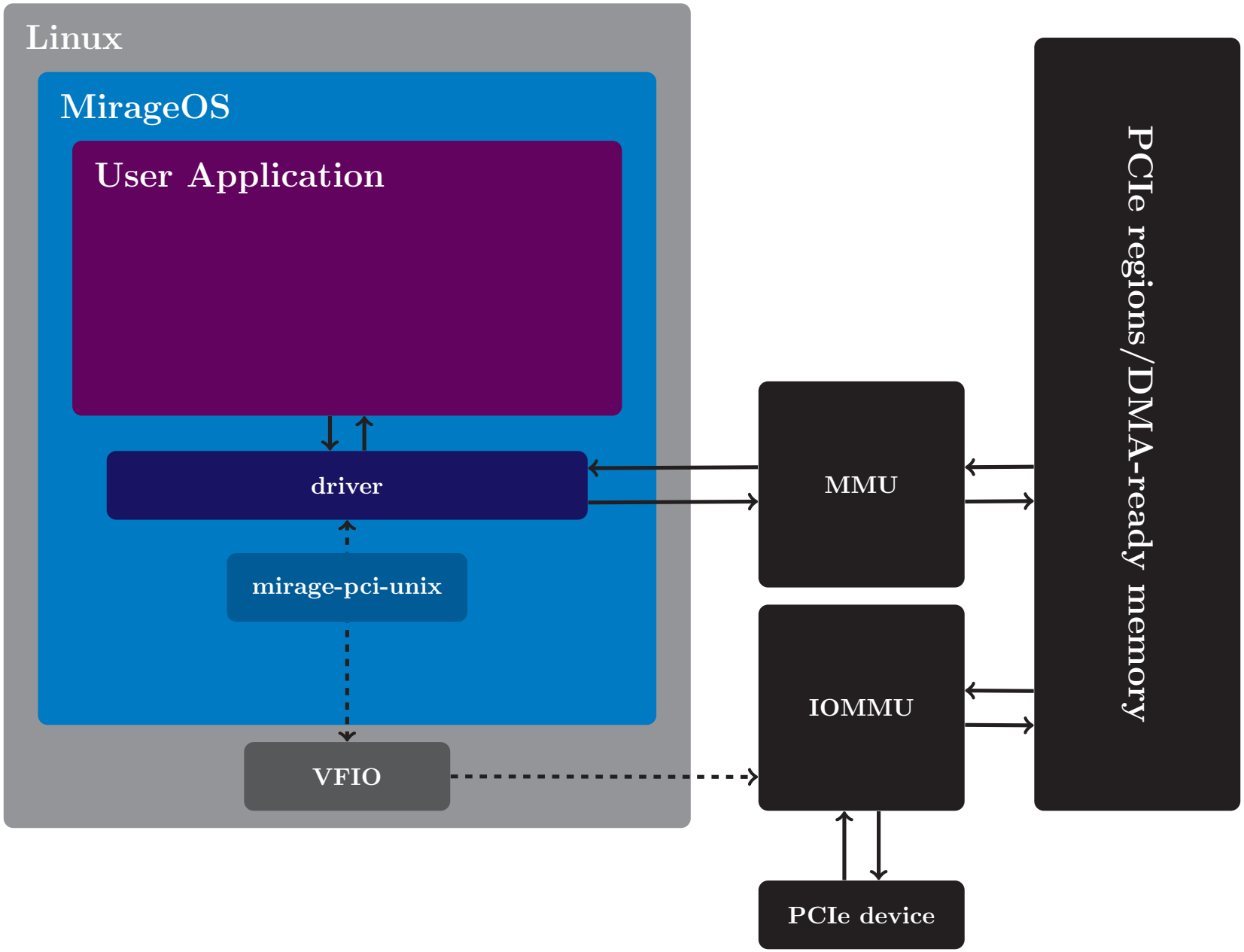
Program the IOMMU using VFIO

PCIe device and Unikernel can see the same memory!

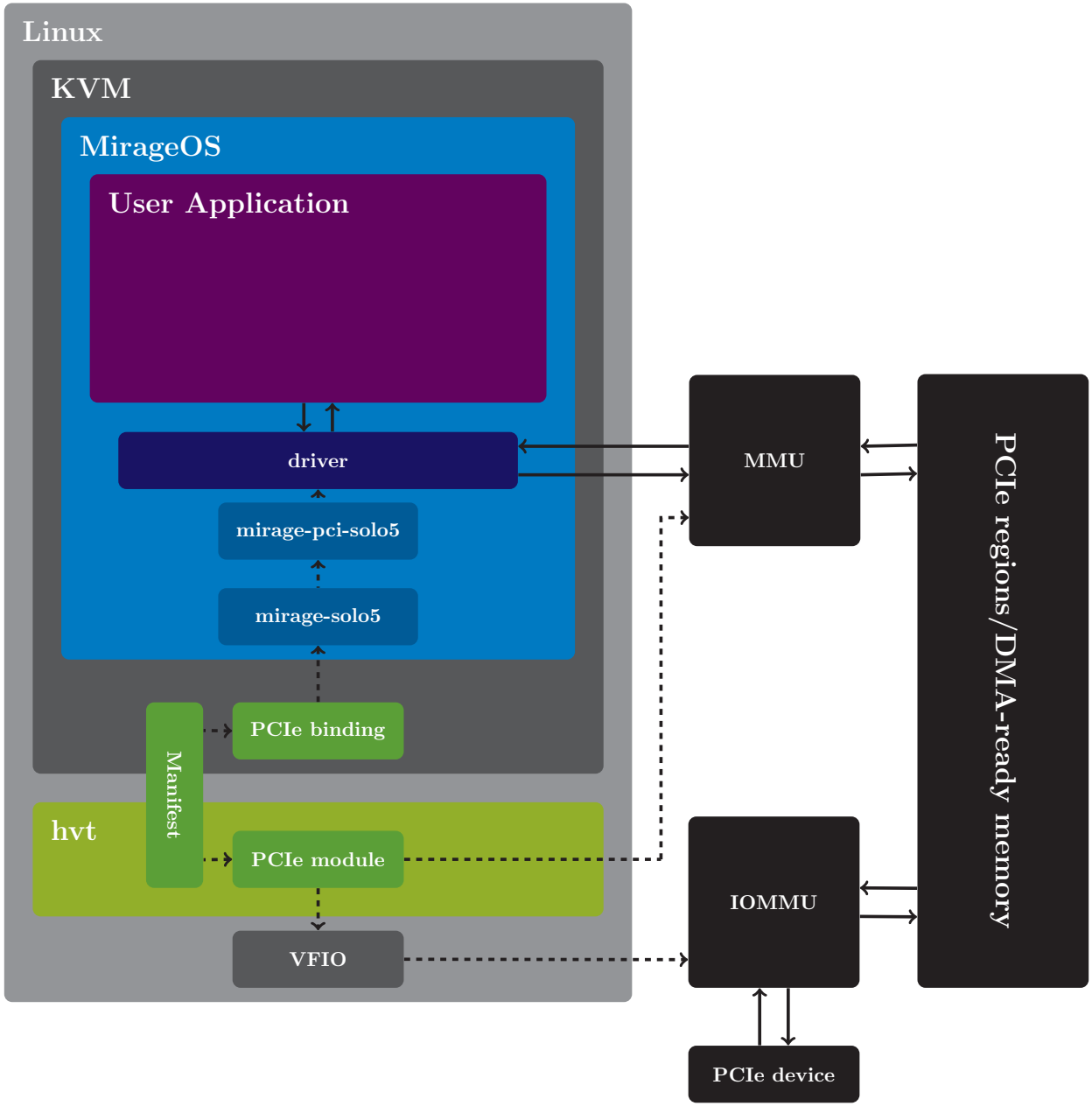
mirage-pci

```
module type Mirage_pci.S =
  sig
    (* error handling omitted *)
    type t
    val disconnect : t -> unit Lwt.t
    val vendor_id : t -> int
    val device_id : t -> int
    val class_code : t -> int
    val subclass_code : t -> int
    val progif : t -> int
    val bar0 : t -> Cstruct.t option
    val bar1 : t -> Cstruct.t option
    val bar2 : t -> Cstruct.t option
    val bar3 : t -> Cstruct.t option
    val bar4 : t -> Cstruct.t option
    val bar5 : t -> Cstruct.t option
    val dma : t -> Cstruct.t
    val name : t -> string
  end
```

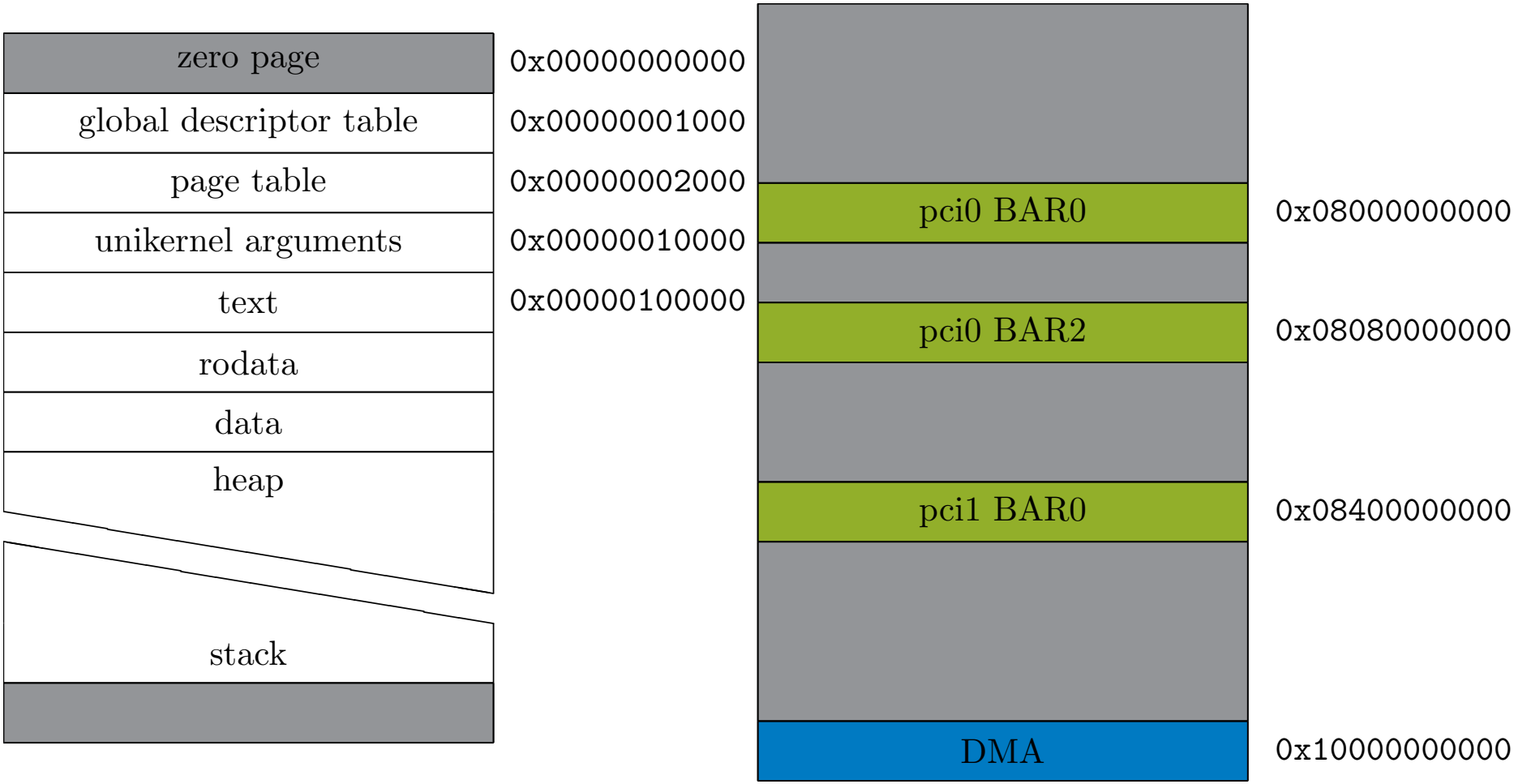
mirage-pci-unix



mirage-pci-solo5



mirage-pci-solo5



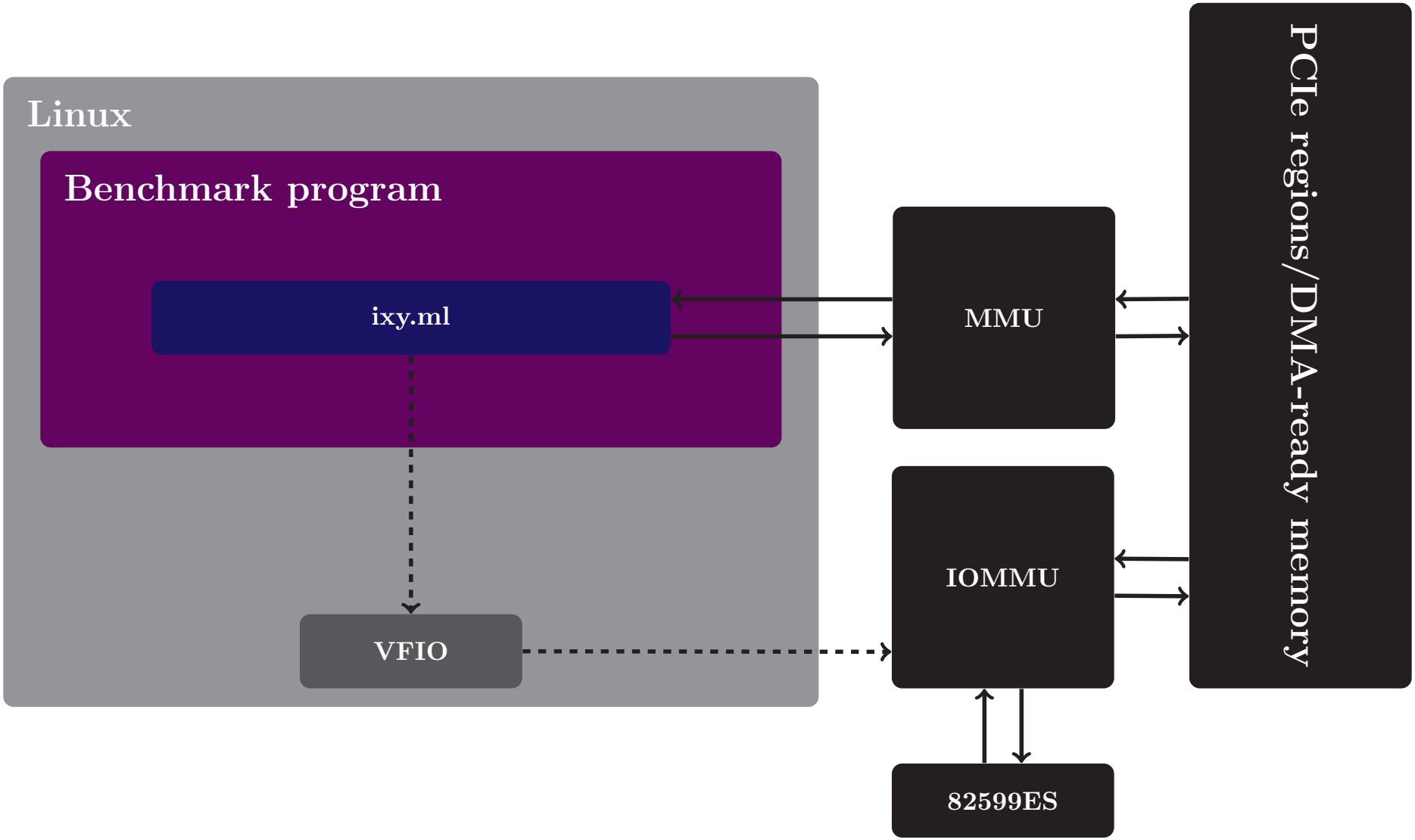
Excursion: ixy.ml

What is ixy.ml?

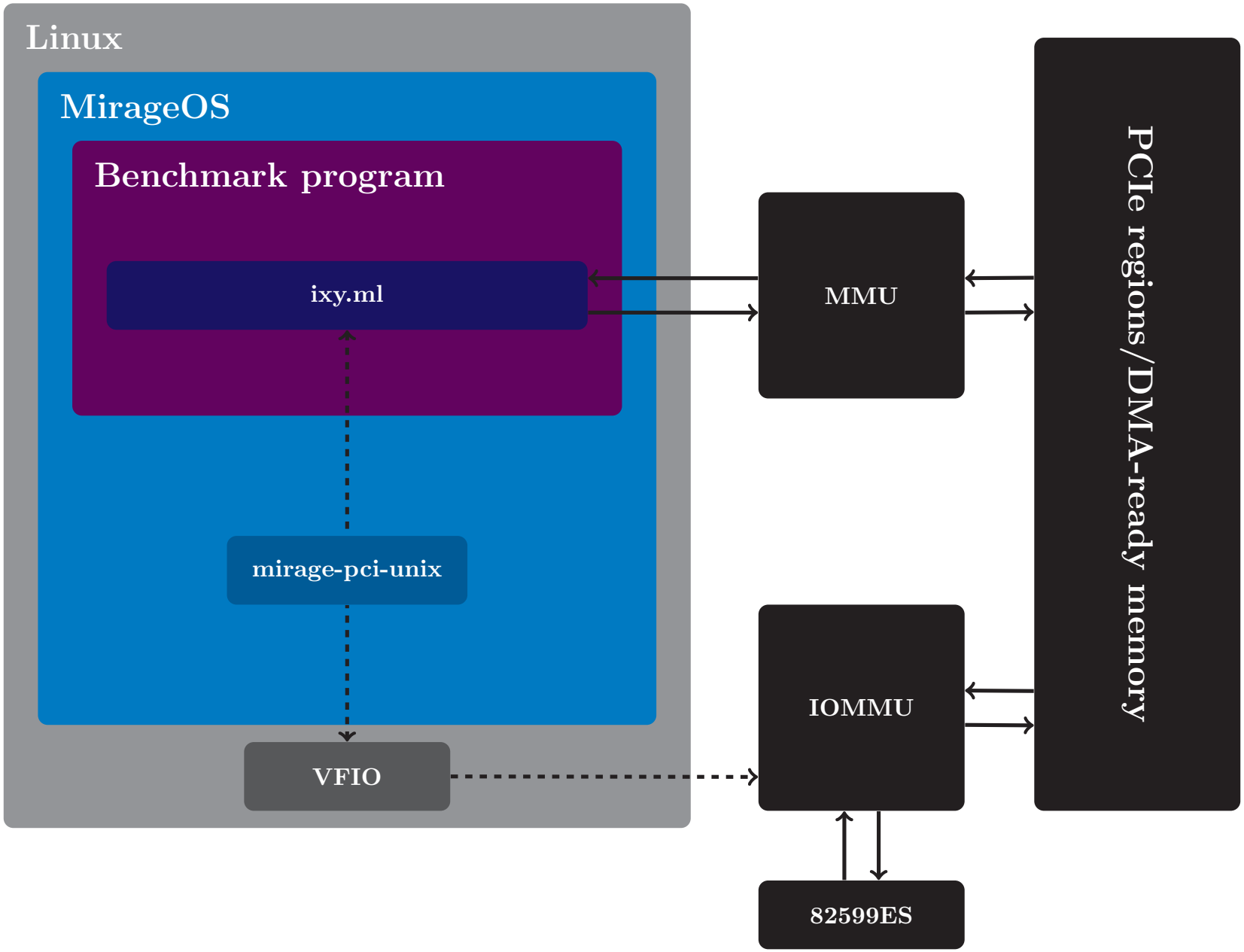
`https://github.com/ixy-languages/ixy.ml`

- userspace network driver
- written in OCaml
- targets Linux
- targets Intel ixgbe NICs

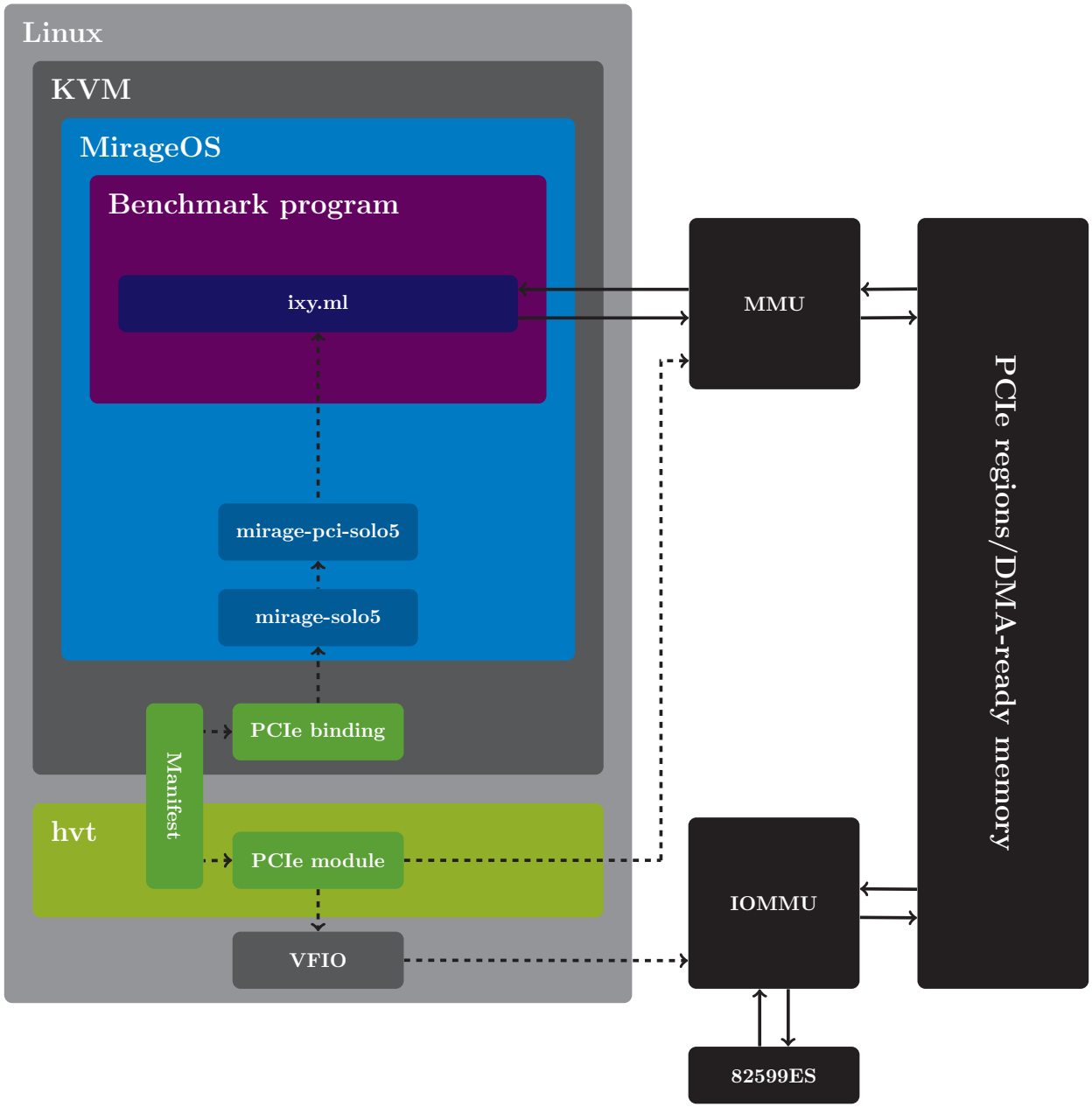
Latency measurements



Latency measurements



Latency measurements



Latency measurements

```
open Mirage

let main = foreign "Unikernel.Main" (pci @-> job)

let pci0 =
  pcidev (* configuration omitted *) "pci0"

let () =
  register "pci" [
        main $ pci0
  ] ~packages:[ package "ixy-core"; package "mirage-net-ixy" ]
```

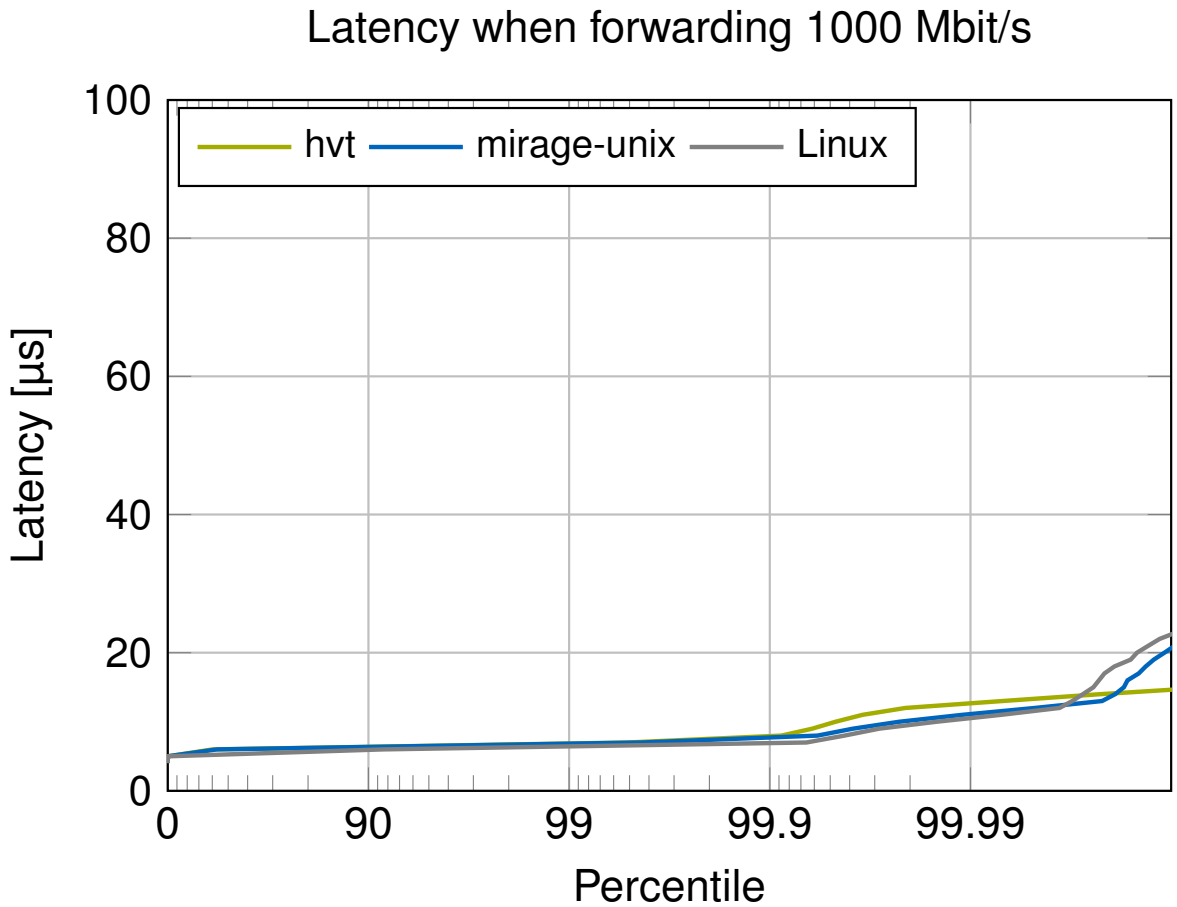
Latency measurements

```
module Main (S: Mirage_pci.S) = struct
  module Ixy = Ixy_core.Make (Pci_mirage.Make (S))

  let start pci0 =
    let dev = Ixy.create ~pci:pci0 ~rxq:1 ~txq:1 in
    while true do
      let rx = Ixy.rx_batch dev 0 in
      Ixy.tx_batch_busy_wait dev 0 rx;
    done;
    Lwt.return_unit
end
```

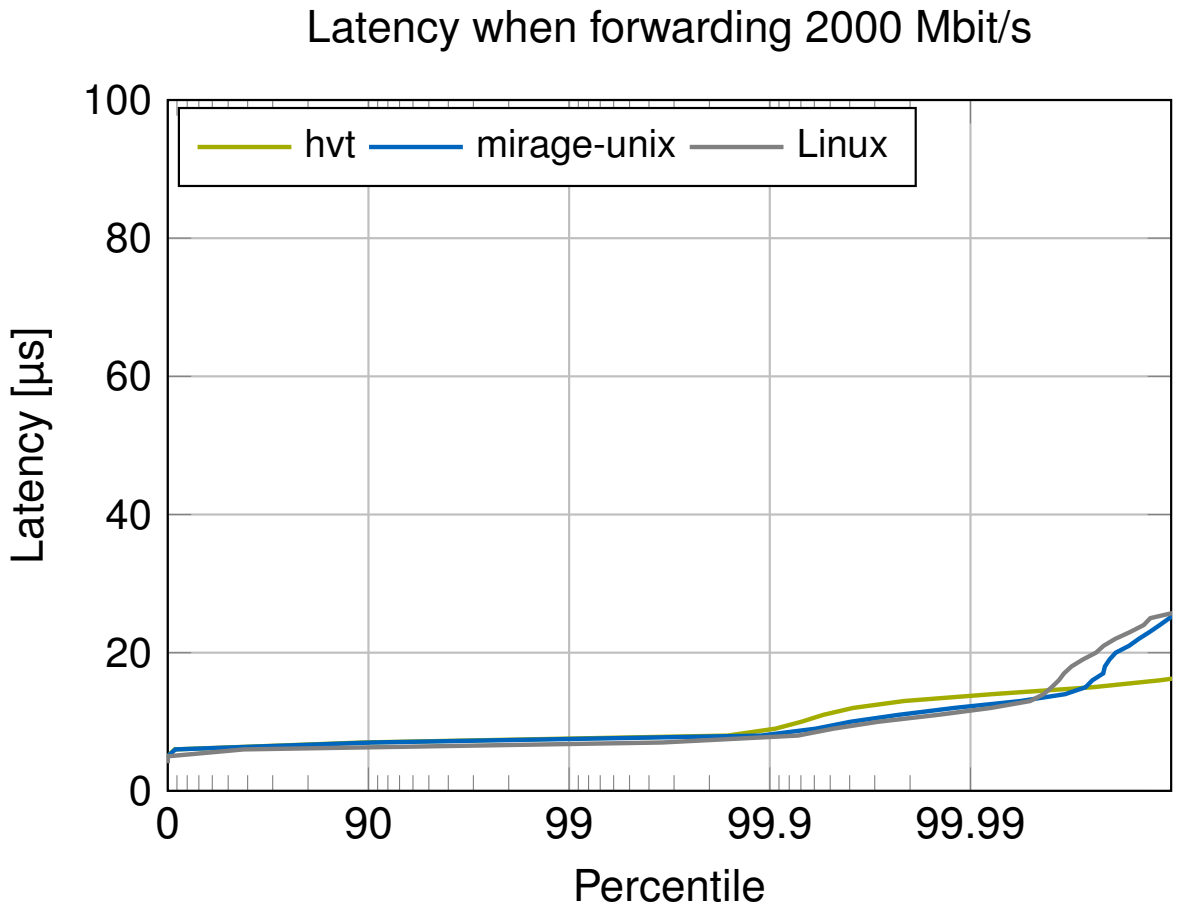
Latency measurements

```
let usage () =
  Ixy_core.Log.error "Usage: %s <pci_addr>" Sys.argv.(0)

let () =
  if Array.length Sys.argv <> 2 then
    usage ();
  let pci =
    match Ixy.of_string Sys.argv.(1) with
    | None -> usage ()
    | Some pci -> pci in
  let dev = Ixy.create ~pci ~rxq:1 ~txq:1 in
  while true do
    let rx = Ixy.rx_batch dev 0 in
    Ixy.tx_batch_busy_wait dev 0 rx
  done
```

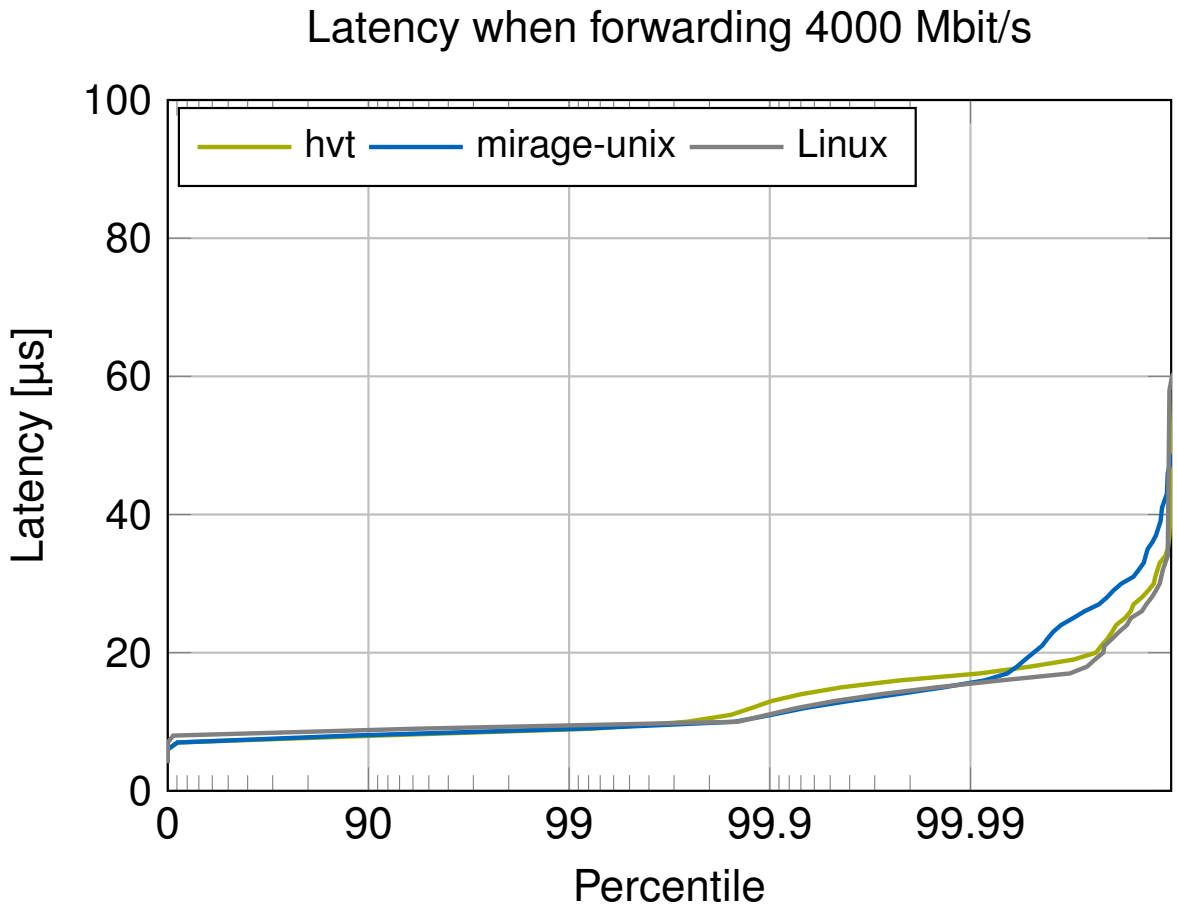
Latency measurements



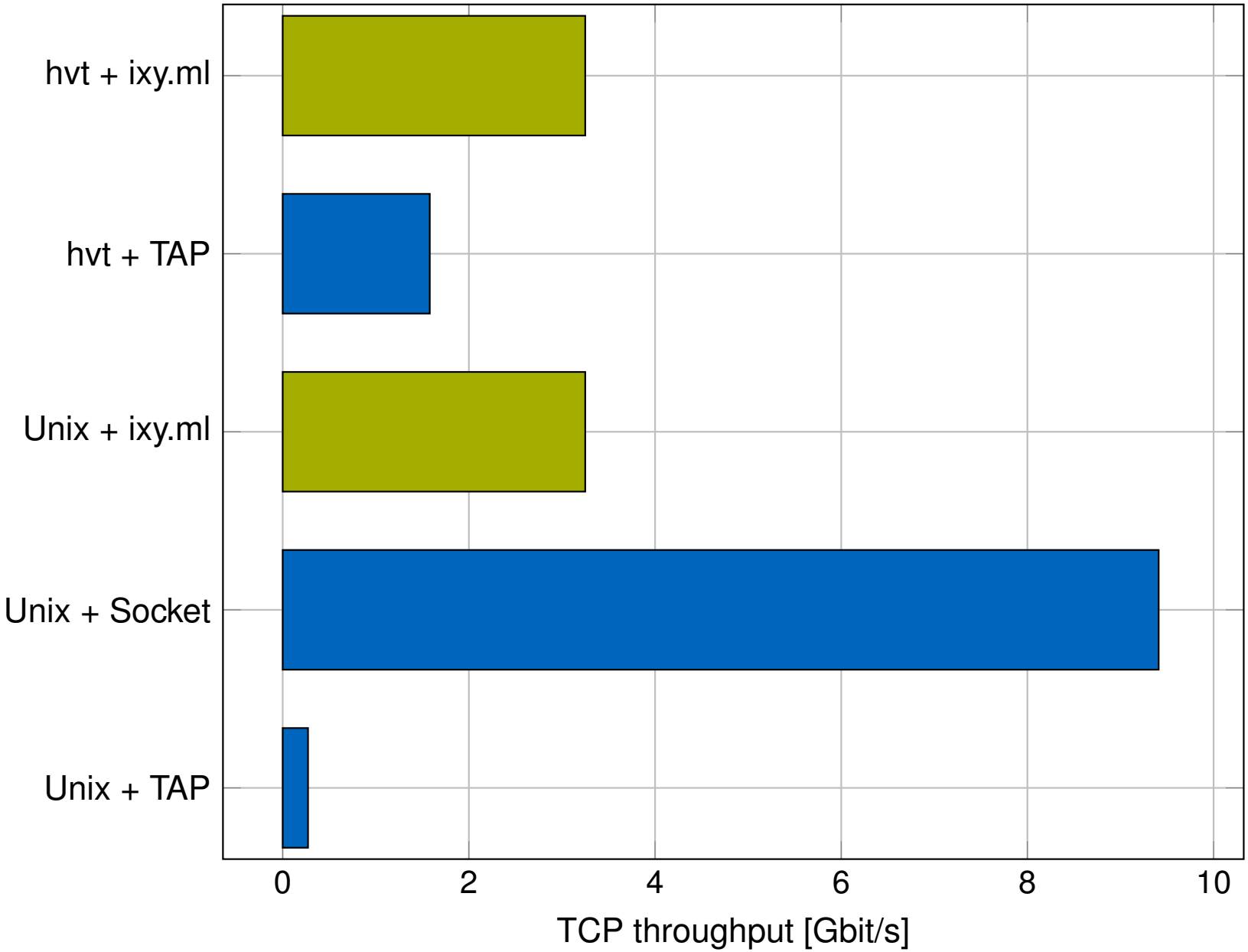
Latency measurements



Latency measurements



iperf


hvt + ixy.ml
hvt + TAP
Unix + ixy.ml
Unix + Socket
Unix + TAP
0
2
4
6
8
10
TCP throughput [Gbit/s]

Bibliography

A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, J. Crowcroft, “Unikernels: Library Operating Systems for the Cloud,” SIGPLAN Notices, vol. 48, pp. 461-472, March 2013