Writing Network Drivers in High-Level Languages

Paul Emmerich, **Simon Ellmann**, Fabian Bonk, Alex Egger,
Alexander Frank, Thomas Günzel, Stefan Huber, Alexandru Obada,
Maximilian Pudelko, Maximilian Stadlmeier, Sebastian Voit

February 2, 2019

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

Writing Network Drivers in High-Level Languages

Paul Emmerich[1], **Simon Ellmann**[2], **Fabian Bonk**[3], **Alex Egger**[4], **Alexander Frank**[5], **Thomas Günzel**[6], **Stefan Huber**[7], **Alexandru Obada**[8], **Maximilian Pudelko**[9], **Maximilian Stadlmeier**[10], **Sebastian Voit**[11]

[1]C, thesis advisor [2]Rust [3]OCaml [4]Haskell [5]Latency measurements
[6]Swift [7]IOMMU [8]Python [9]VirtIO driver [10]C# [11]Go

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

About us

Paul

- PhD student at Technical University of Munich
- Researching performance of packet processing systems

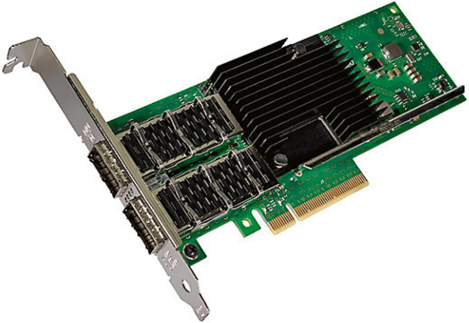
Simon

- Rust driver as bachelor's thesis, now research assistant

Everyone else mentioned on the title slide

- Did a thesis with Paul as advisor

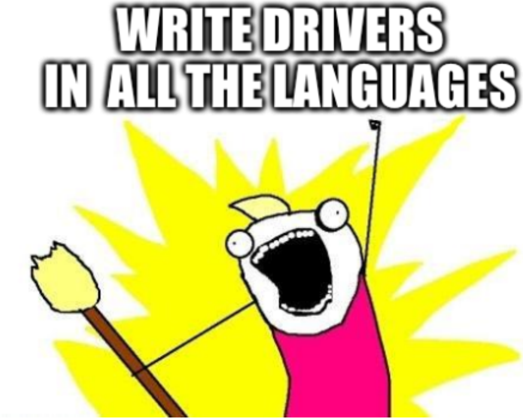
Network drivers



Intel XL710 [Picture: Intel.com]

The ixy project

- Attempt to write a simple yet fast user space network driver
- It's a user space driver you can easily understand and read
- $\approx$ 1,000 lines of C code, full of references to datasheets and specs
- Supports Intel ixgbe NICs and VirtIO
- Check it out on GitHub: `https://github.com/emmericp/ixy`
- But is C the best language for drivers?


WRITE DRIVERS
IN ALL THE LANGUAGES

ALL THE LANGUAGES?

OPEN

Title	Type	Advisors	Year	Links
Writing Network Drivers in Rust	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Go	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Java	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in C#	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Haskell	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Scala	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in OCaml	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Javascript	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Python	BA, MA, IDP	Paul Emmerich	2018	PDF
Writing Network Drivers in Bash	BA	Paul Emmerich	2018	PDF

Basics: How to talk to (modern) PCIe devices

1. Memory-mapped IO (MMIO)
2. Direct memory access (DMA)
3. Interrupts

Basics: How to talk to (modern) PCIe devices

1. Memory-mapped IO (MMIO)
   - Magic memory area that is mapped to the device
   - Memory reads/writes are directly forwarded to the device
   - Usually used to expose device registers
   - User space drivers: `mmap` a magic file
2. Direct memory access (DMA)
3. Interrupts

Basics: How to talk to (modern) PCIe devices

1. Memory-mapped IO (MMIO)
2. Direct memory access (DMA)
   - Allows the device to read/write arbitrary memory locations
   - User space drivers: figure out physical addresses, tell the device to write there
3. Interrupts

Basics: How to talk to (modern) PCIe devices

1. Memory-mapped IO (MMIO)
2. Direct memory access (DMA)
3. Interrupts
    - This is how the device informs you about events
    - User space drivers: available via the Linux `vfio` subsystem
    - (Usually) not useful for high-speed network drivers
    - We'll ignore interrupts here

Basics: How to write a user space driver in 4 simple steps

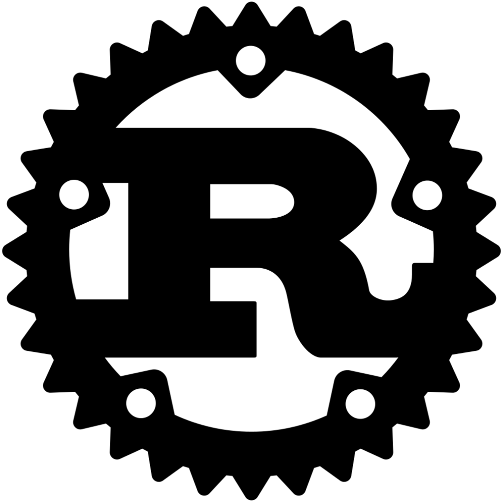
1. Unload kernel driver
2. `mmap` the PCIe MMIO address space
3. Figure out physical addresses for DMA
4. Write the driver

Goals for our implementations

- Implement the same feature set as my C reference driver
- Use a similar structure like the C driver
- Write idiomatic code for the selected language
- Use language safety features where possible
- Quantify trade-offs for performance vs. safety

Our languages

C#
Swift
OCaml
GO
python
™



Rust

What is Rust?

> *A safe, concurrent, practical systems language.*

- No garbage collector
- Unique ownership system and rules for moving/borrowing values
- Unsafe mode

Safety in Rust: The ownership system

- Three rules:
  1. Each value has a variable that is its owner
  2. There can only be one owner at a time
  3. When the owner goes out of scope, the value is freed
- Rules enforced at compile-time
- Ownership can be passed to another variable

Safety in Rust: The ownership system by example

- `Packet`s are owners of some DMA memory
- `Packet`s are passed between user code and the driver, thus ownership is passed as well
- At any point in time there is only one `Packet` owner that can change its memory

```
let buffer: &mut VecDeque<Packet> = VecDeque::new();
dev.rx_batch(RX_QUEUE, buffer, BATCH_SIZE);
for p in buffer.iter_mut() {
  p[48] += 1;
}
dev.tx_batch(TX_QUEUE, buffer);
buffer.drain(..);
```

Safety in Rust: Unsafe code

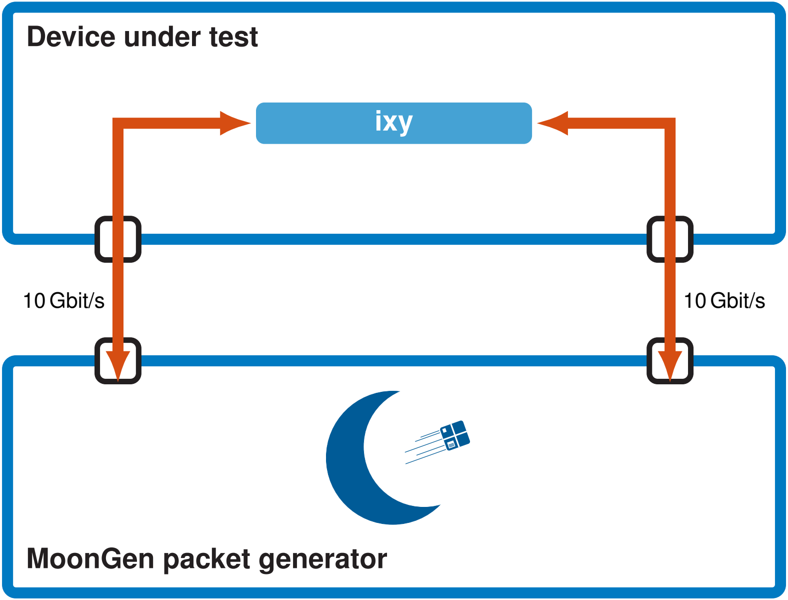
- Not everything can be done in safe Rust
- Calling foreign functions and dereferencing raw pointers is unsafe
- Many functions in Rust’s standard library make use of unsafe code

```rust
let ptr = unsafe {
  libc::mmap(
    ptr::null_mut(), len, libc::PROT_READ | libc::PROT_WRITE,
    libc::MAP_SHARED, file.as_raw_fd(), 0,
  ) as *mut u8
};
```

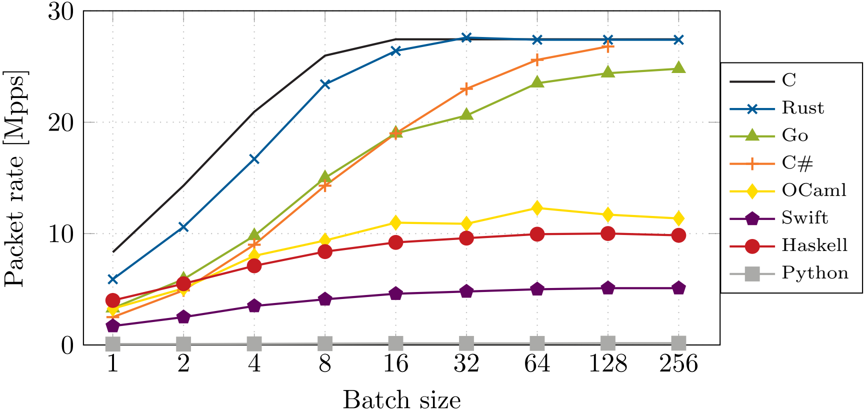
Example: Setting registers

- Biggest challenge: safe memory handling with unsafe code

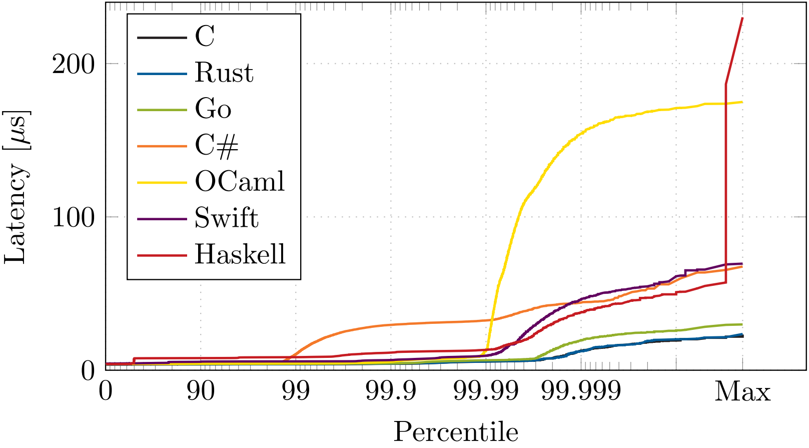
```
fn set_reg32(&self, reg: u32, val: u32) {
  assert!(
    reg as usize <= self.len - 4 as usize,
    "memory access out of bounds"
  );

  unsafe {
    ptr::write_volatile(
        (self.addr as usize + reg as usize) as *mut u32, val
    );
  }
}
```


Device under test
ixy
10 Gbit/s
10 Gbit/s
MoonGen packet generator

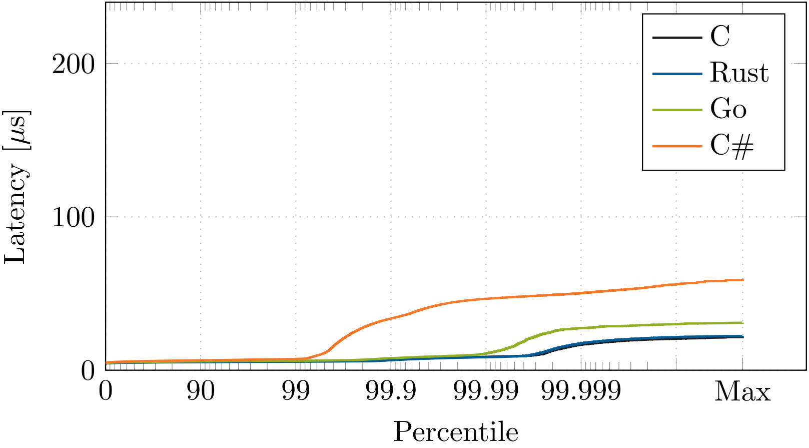
Batching at 3.3 GHz CPU speed



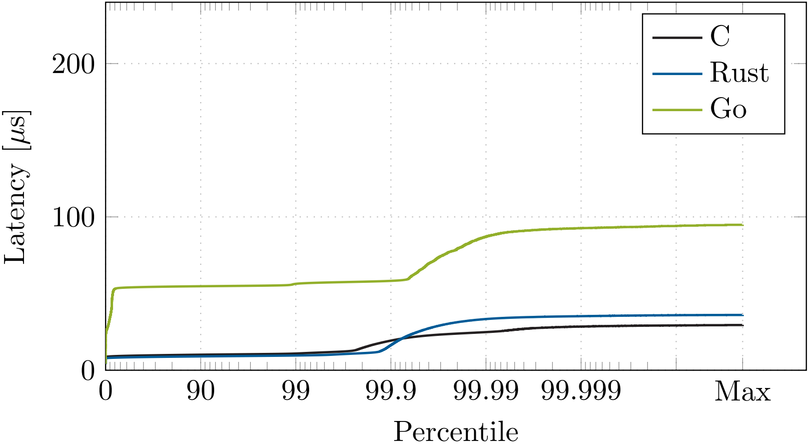
Tail latency at 1 Mpps



Tail latency at 10 Mpps



Tail latency at 20 Mpps



Look ma, no root

- User space drivers usually run with root privileges, but why?

Look ma, no root

- User space drivers usually run with root privileges, but why?

- Mapping PCIe resources requires root
- Allocating non-transparent huge pages requires root
- Locking memory requires root

- Can we do that in a small separate program that is easy to audit and then drop privileges?

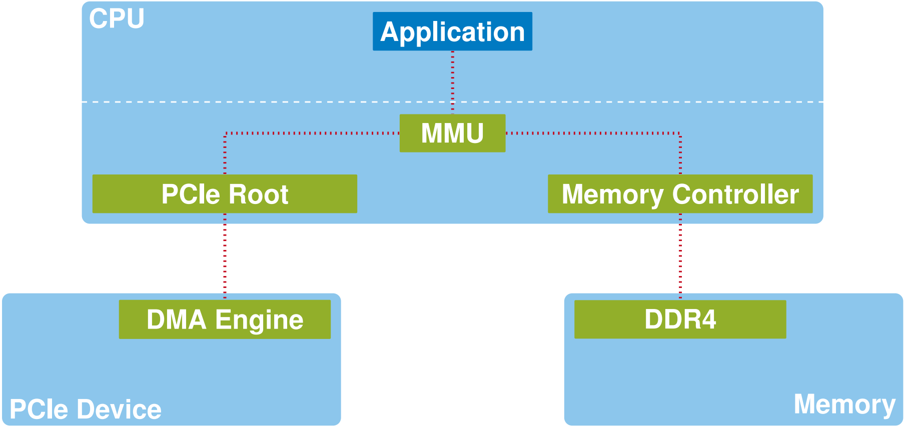
Look ma, no root

- User space drivers usually run with root privileges, but why?

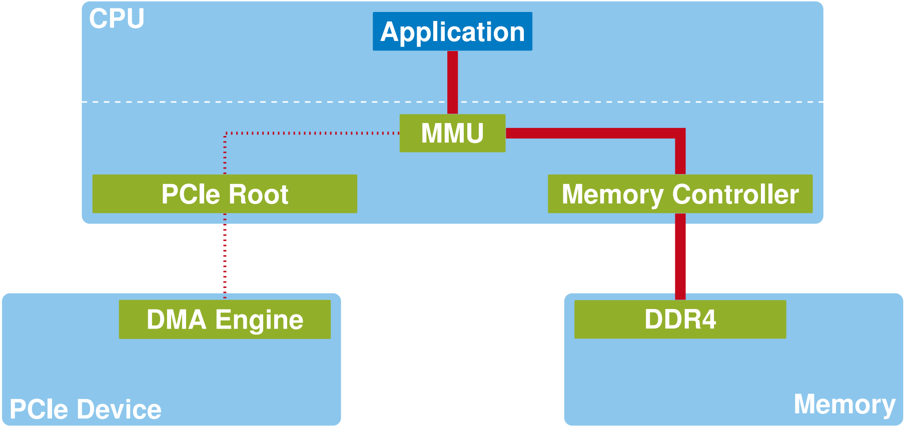
- Mapping PCIe resources requires root
- Allocating non-transparent huge pages requires root
- Locking memory requires root

- Can we do that in a small separate program that is easy to audit and then drop privileges?
- Yes, we can
- But it's not really secure

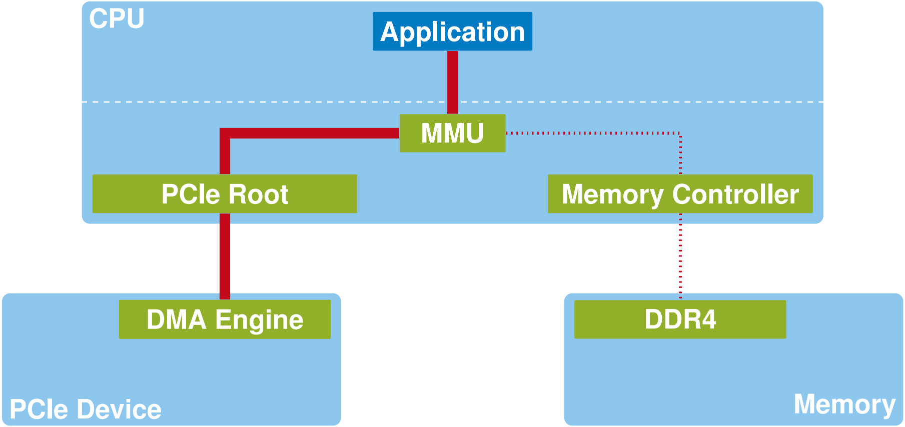
Memory access on modern systems



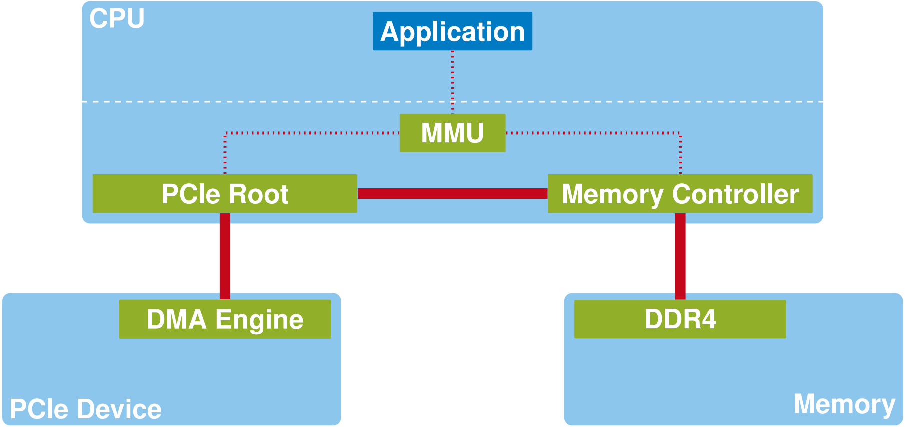
Memory access on modern systems



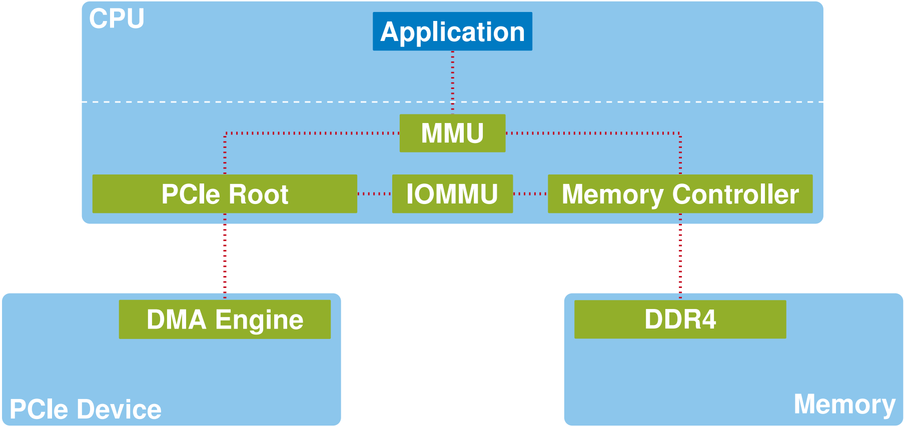
Memory access on modern systems



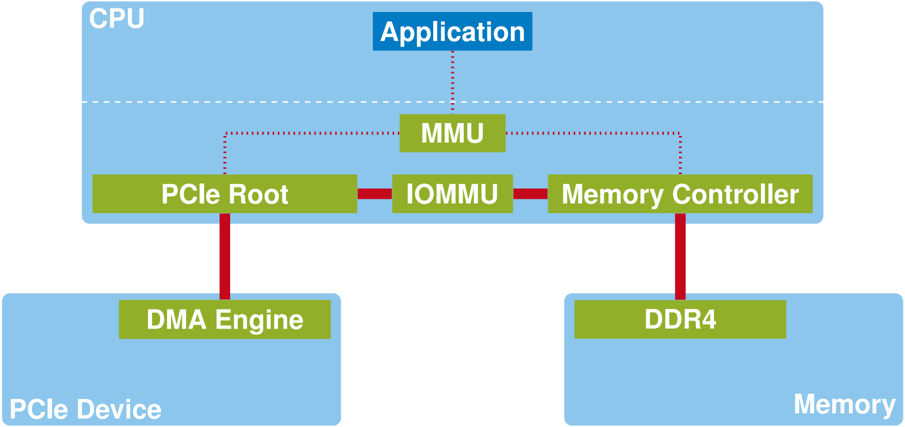
Memory access on modern systems



Memory access on modern systems



Memory access on modern systems



Conclusion: Check out our code

- Meta-repository with links:
  `https://github.com/ixy-languages/ixy-languages`
- Drivers are simple: don’t be afraid of them
- No kernel code needed :)