|galois|

Leveraging Rust on ~~Core Platform~~ Microkit

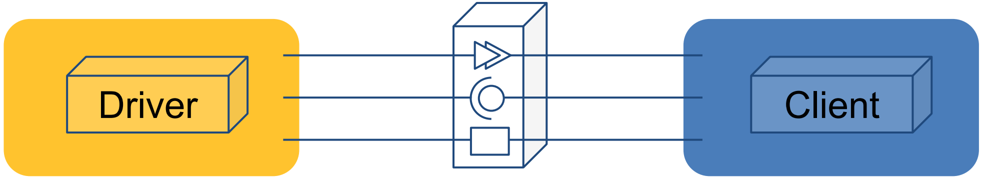
Galois Inc.

Ben Hamlin, Michal Podhradsky, Tiago Ferreira, Mike Dodds, Mike Beynon

* This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Motivation: Reusable Drivers in Microkit

- To create a dedicated driver component, one needs to:
  - Design a driver <-> client IPC protocol
  - Implement the Handler for the driver; make it speak your IPC protocol, possibly adapting an existing driver
  - Implement the client; adapt any libraries already written to interface with hardware drivers to use this IPC protocol instead
- Can we do better?



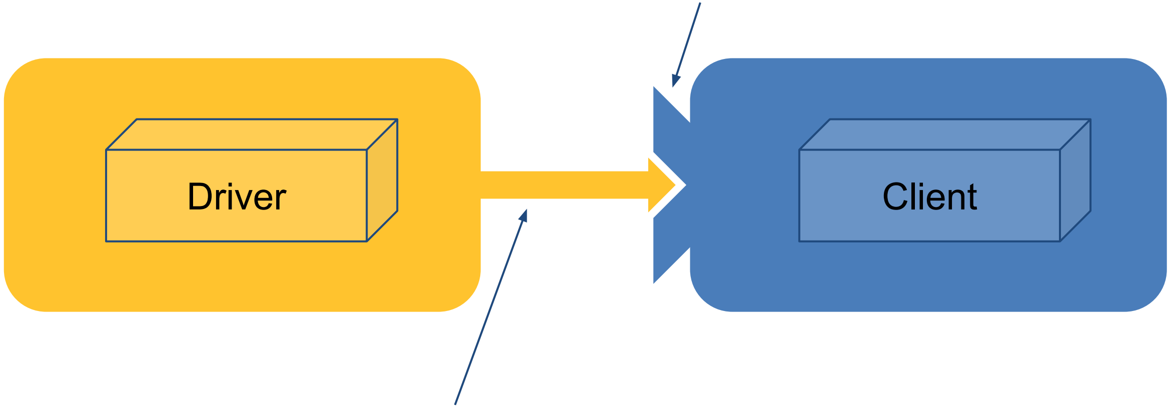
Adapter crate

embedded-hal

- A Hardware Abstraction Layer (HAL) for embedded systems
- Thriving embedded ecosystem
- **Traits for reusable drivers**
- Leverage this for seL4 Microkit!

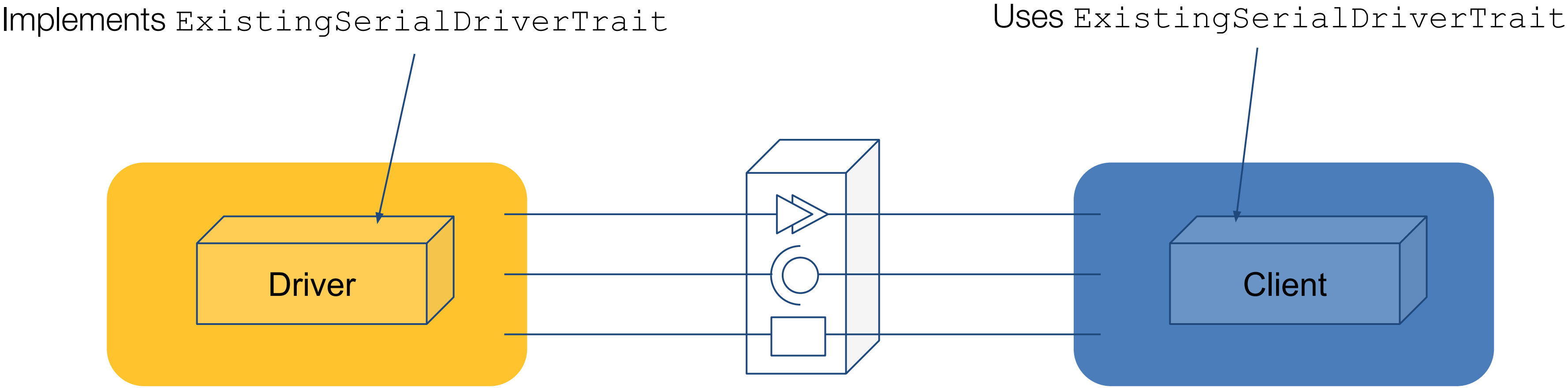
Reusable Code in Rust

```
fn do_some_things_with_serial<Driver>()
    where Driver: ExistingSerialDriverTrait { … }
```



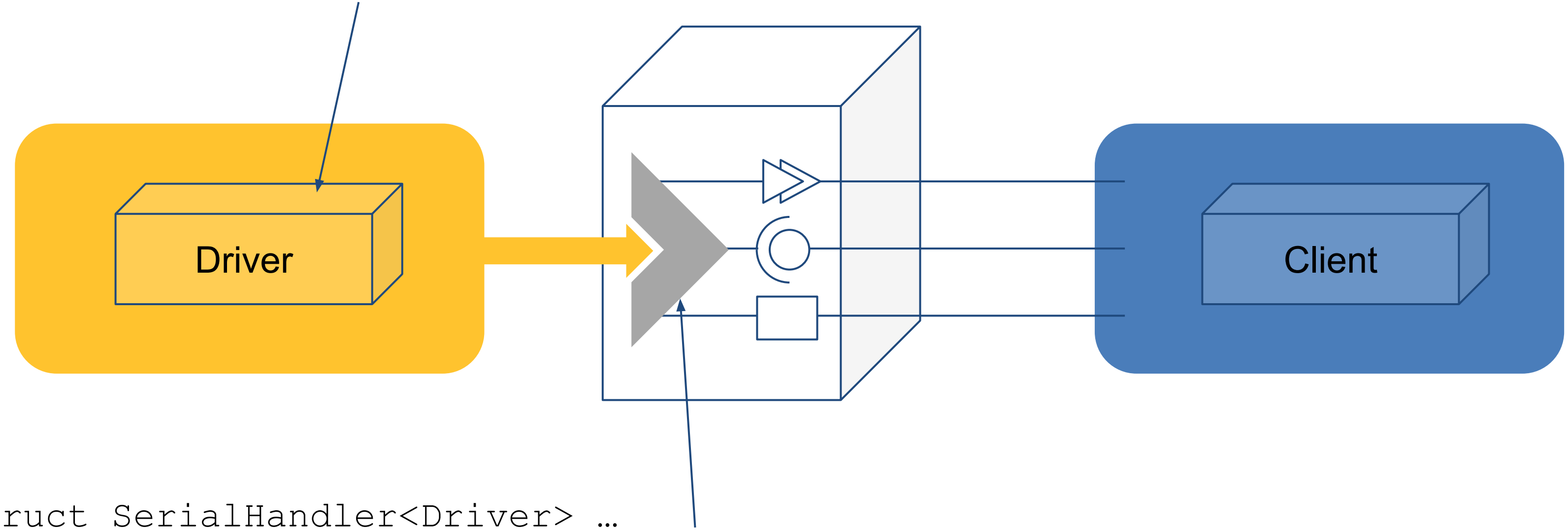
```
struct MySerialDriver { … }
impl ExistingSerialDriverTrait for MySerialDriver { … }
```

Idea: Polymorphic Structs for Better Interfaces


Implements `ExistingSerialDriverTrait`
Uses `ExistingSerialDriverTrait`
Driver
Client

Idea: Polymorphic Structs for Better Interfaces

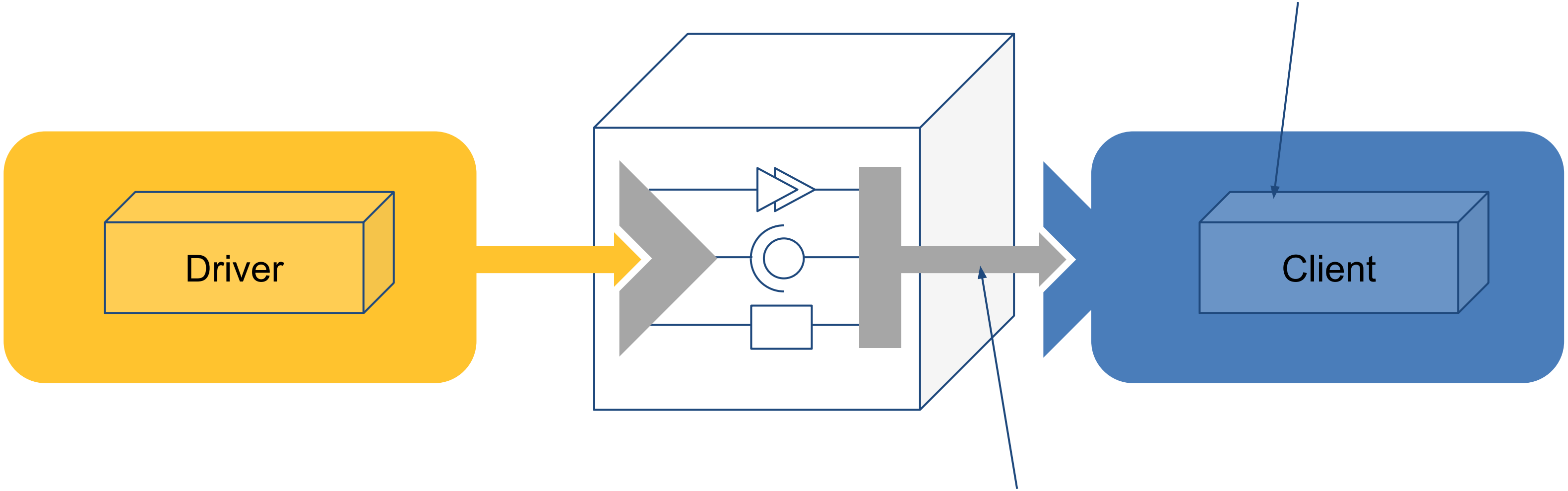
Implements `ExistingSerialDriverTrait`



```
struct SerialHandler<Driver> …
impl<Driver> Handler for SerialHandler<Driver>
    where Driver: ExistingSerialDriverTrait …
```

Idea: Polymorphic Structs for Better Interfaces

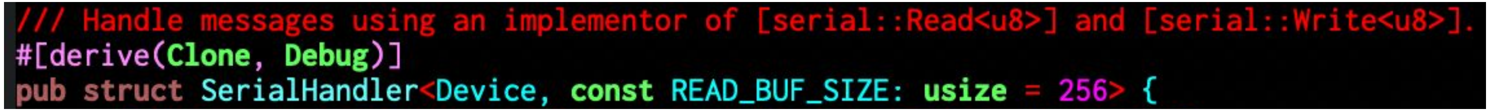
Uses `ExistingSerialDriverTrait`



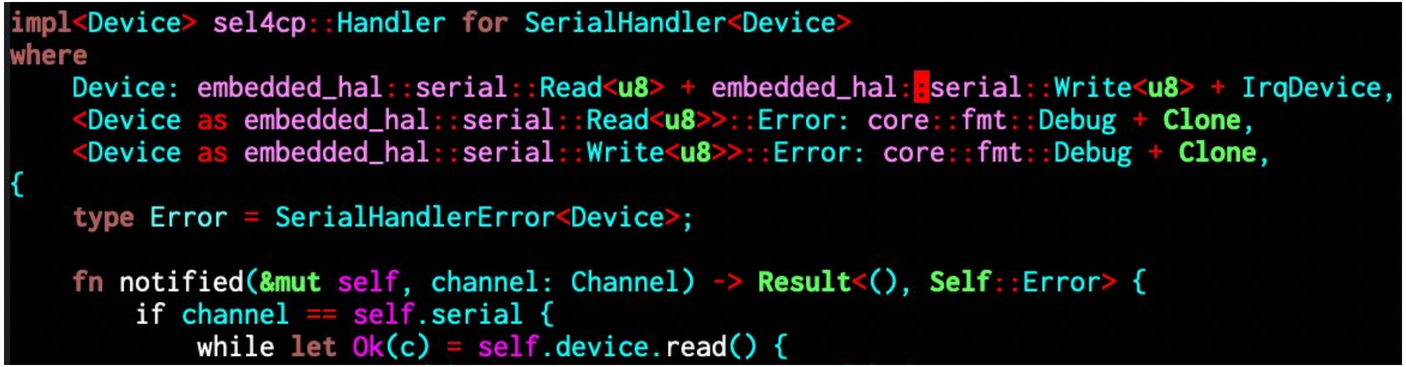
```
struct SerialDriver …
impl ExistingSerialDriverTrait
    for SerialHandler<Driver> …
```

Serial Example: The Driver's View

```
sed -i 's/sel4cp/microkit/g'
```

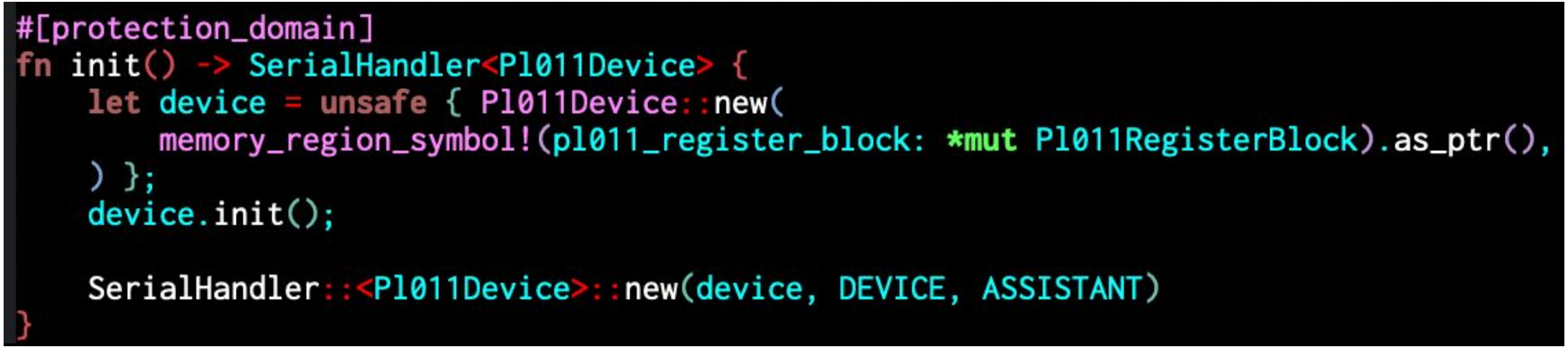


```
/// Handle messages using an implementor of [serial::Read<u8>] and [serial::Write<u8>].
#[derive(Clone, Debug)]
pub struct SerialHandler<Device, const READ_BUF_SIZE: usize = 256> {
```



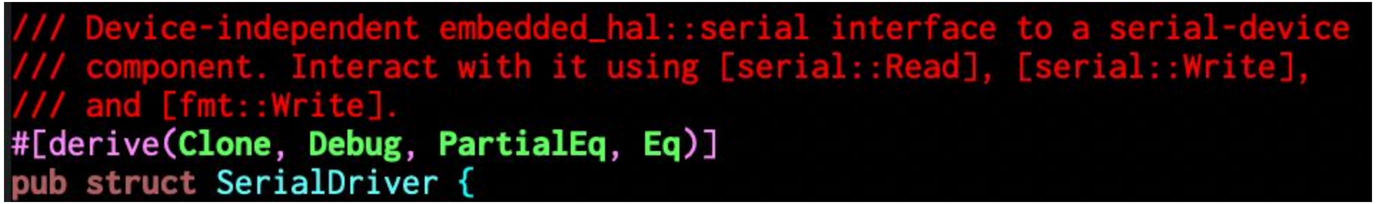
```
impl<Device> sel4cp::Handler for SerialHandler<Device>
where
    Device: embedded_hal::serial::Read<u8> + embedded_hal::serial::Write<u8> + IrqDevice,
    <Device as embedded_hal::serial::Read<u8>>::Error: core::fmt::Debug + Clone,
    <Device as embedded_hal::serial::Write<u8>>::Error: core::fmt::Debug + Clone,
{
    type Error = SerialHandlerError<Device>;

    fn notified(&mut self, channel: Channel) -> Result<(), Self::Error> {
        if channel == self.serial {
            while let Ok(c) = self.device.read() {
```

Serial Example: Instantiating the Handler

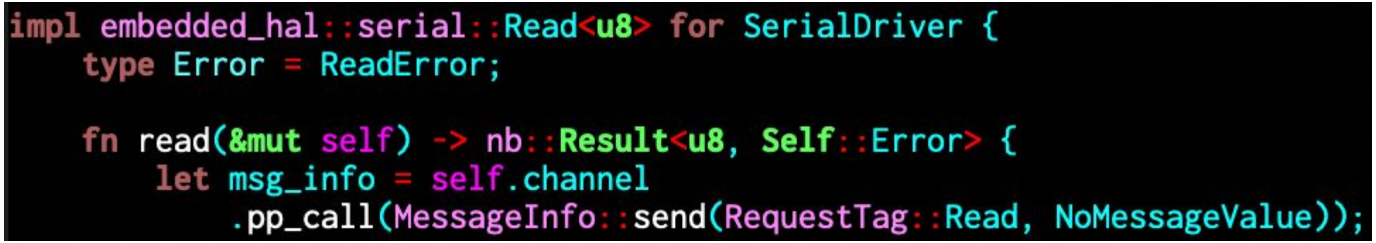


```
#[protection_domain]
fn init() -> SerialHandler<Pl011Device> {
    let device = unsafe { Pl011Device::new(
        memory_region_symbol!(pl011_register_block: *mut Pl011RegisterBlock).as_ptr(),
    ) };
    device.init();

    SerialHandler::<Pl011Device>::new(device, DEVICE, ASSISTANT)
}
```

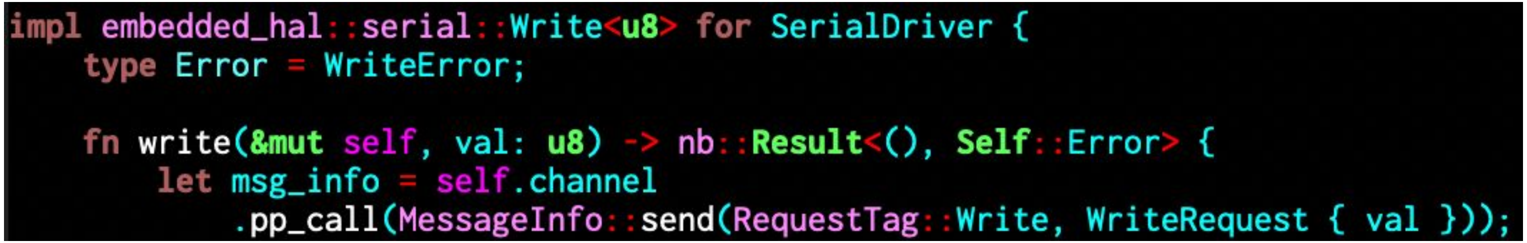
Serial Example: The Client's View



```
/// Device-independent embedded_hal::serial interface to a serial-device
/// component. Interact with it using [serial::Read], [serial::Write],
/// and [fmt::Write].
#[derive(Clone, Debug, PartialEq, Eq)]
pub struct SerialDriver {
```

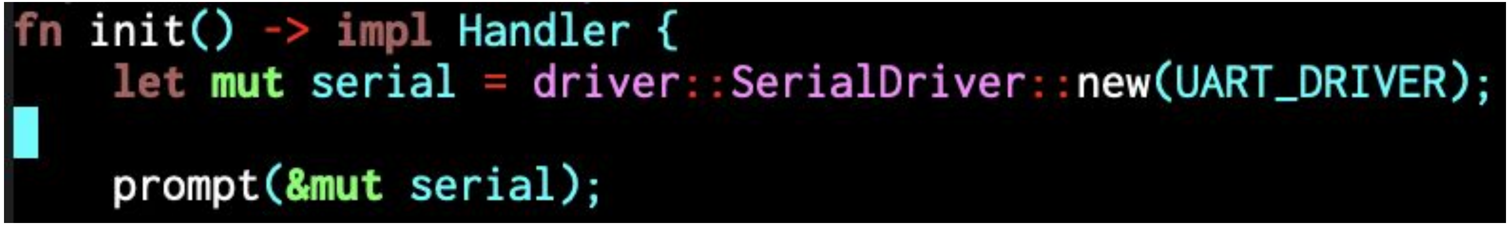


```
impl embedded_hal::serial::Read<u8> for SerialDriver {
    type Error = ReadError;

    fn read(&mut self) -> nb::Result<u8, Self::Error> {
        let msg_info = self.channel
            .pp_call(MessageInfo::send(RequestTag::Read, NoMessageValue));
```

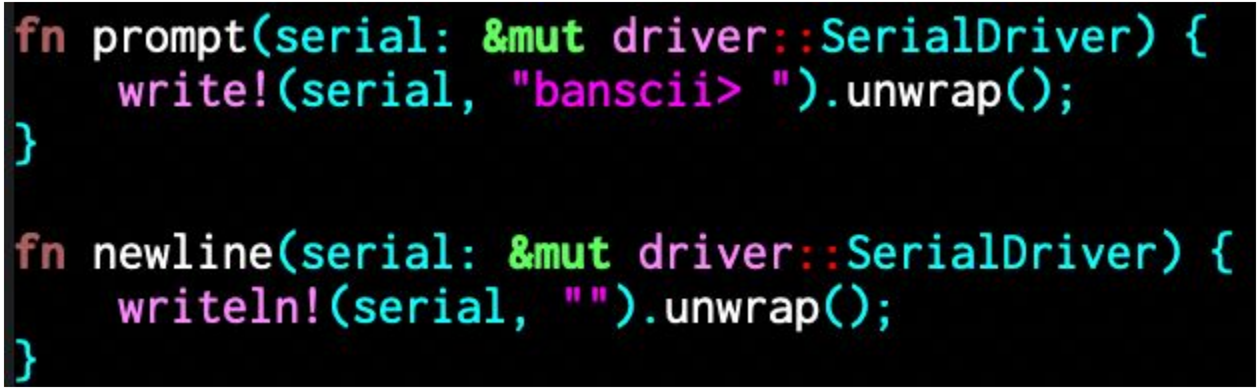


```
impl embedded_hal::serial::Write<u8> for SerialDriver {
    type Error = WriteError;

    fn write(&mut self, val: u8) -> nb::Result<(), Self::Error> {
        let msg_info = self.channel
            .pp_call(MessageInfo::send(RequestTag::Write, WriteRequest { val }));
```

Serial Example: Using the Driver Struct

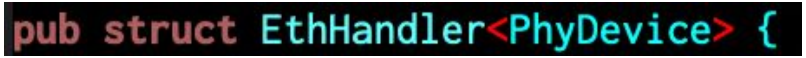


```
fn init() -> impl Handler {
    let mut serial = driver::SerialDriver::new(UART_DRIVER);

    prompt(&mut serial);
```

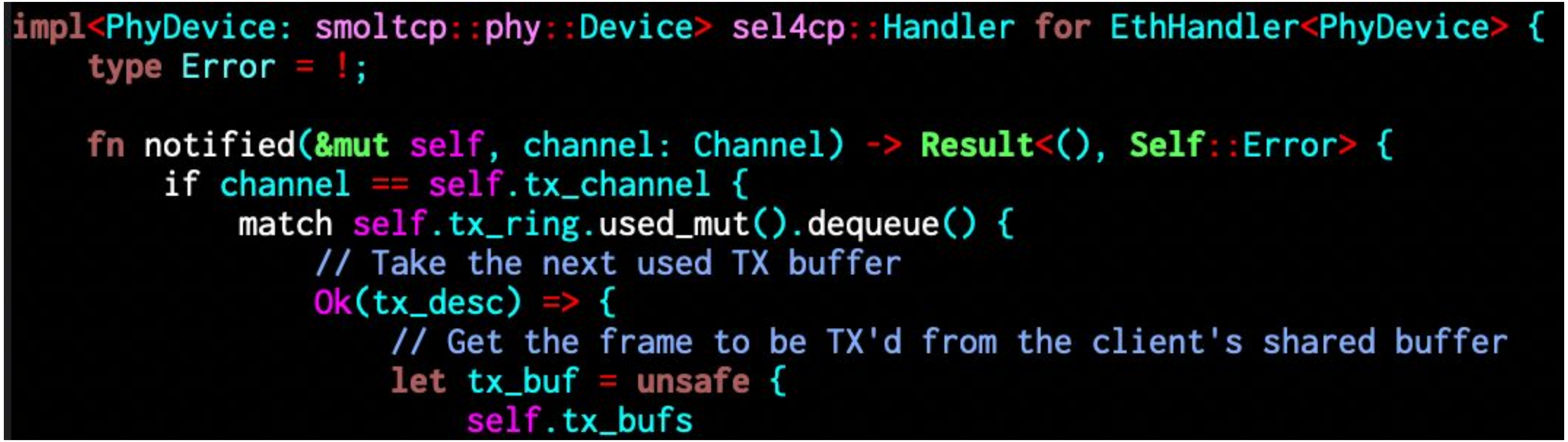


```
fn prompt(serial: &mut driver::SerialDriver) {
    write!(serial, "banscii> ").unwrap();
}

fn newline(serial: &mut driver::SerialDriver) {
    writeln!(serial, "").unwrap();
}
```

Ethernet Example: The Driver's View

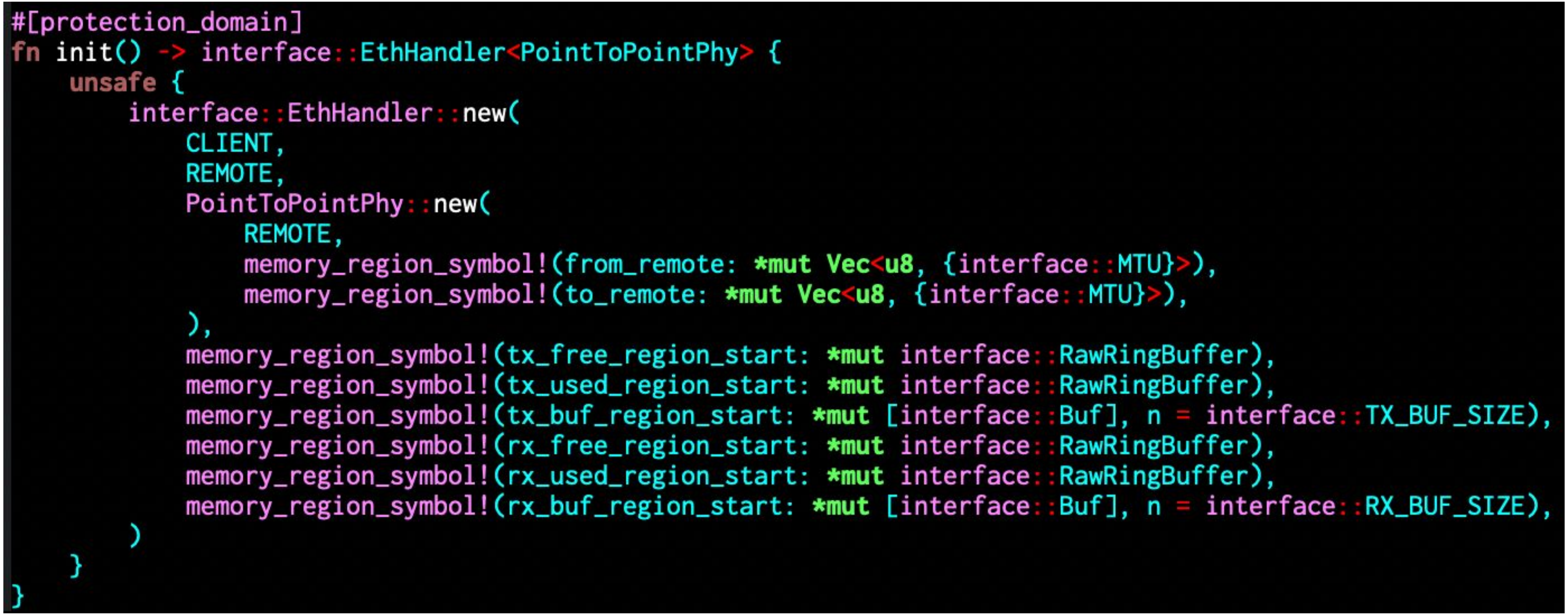


```
pub struct EthHandler<PhyDevice> {
```



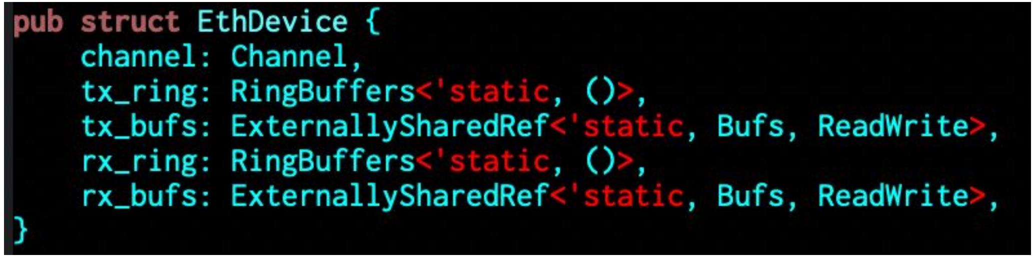
```
impl<PhyDevice: smoltcp::phy::Device> sel4cp::Handler for EthHandler<PhyDevice> {
    type Error = !;

    fn notified(&mut self, channel: Channel) -> Result<(), Self::Error> {
        if channel == self.tx_channel {
            match self.tx_ring.used_mut().dequeue() {
                // Take the next used TX buffer
                Ok(tx_desc) => {
                    // Get the frame to be TX'd from the client's shared buffer
                    let tx_buf = unsafe {
                        self.tx_bufs
```

Ethernet Example: Instantiating the Handler

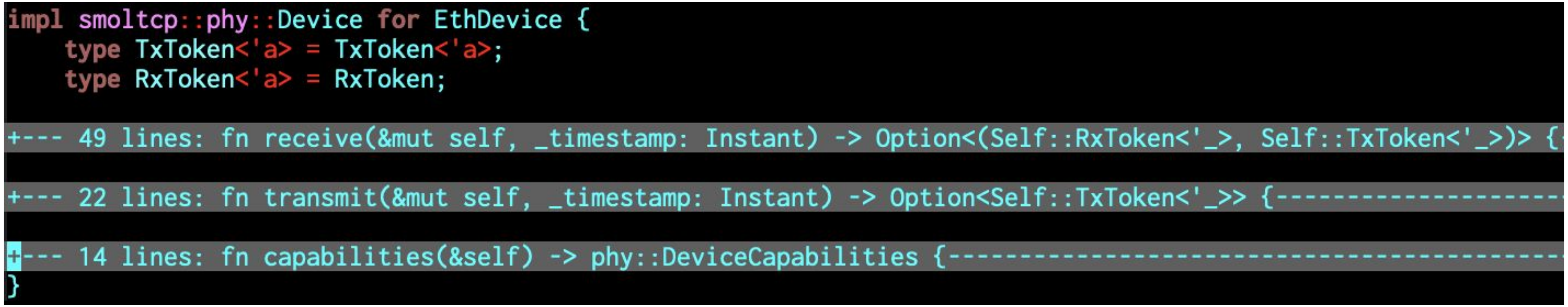


```
#[protection_domain]
fn init() -> interface::EthHandler<PointToPointPhy> {
    unsafe {
        interface::EthHandler::new(
            CLIENT,
            REMOTE,
            PointToPointPhy::new(
                REMOTE,
                memory_region_symbol!(from_remote: *mut Vec<u8, {interface::MTU}>),
                memory_region_symbol!(to_remote: *mut Vec<u8, {interface::MTU}>),
            ),
            memory_region_symbol!(tx_free_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(tx_used_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(tx_buf_region_start: *mut [interface::Buf], n = interface::TX_BUF_SIZE),
            memory_region_symbol!(rx_free_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(rx_used_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(rx_buf_region_start: *mut [interface::Buf], n = interface::RX_BUF_SIZE),
        )
    }
}
```

Ethernet Example: The Client's View

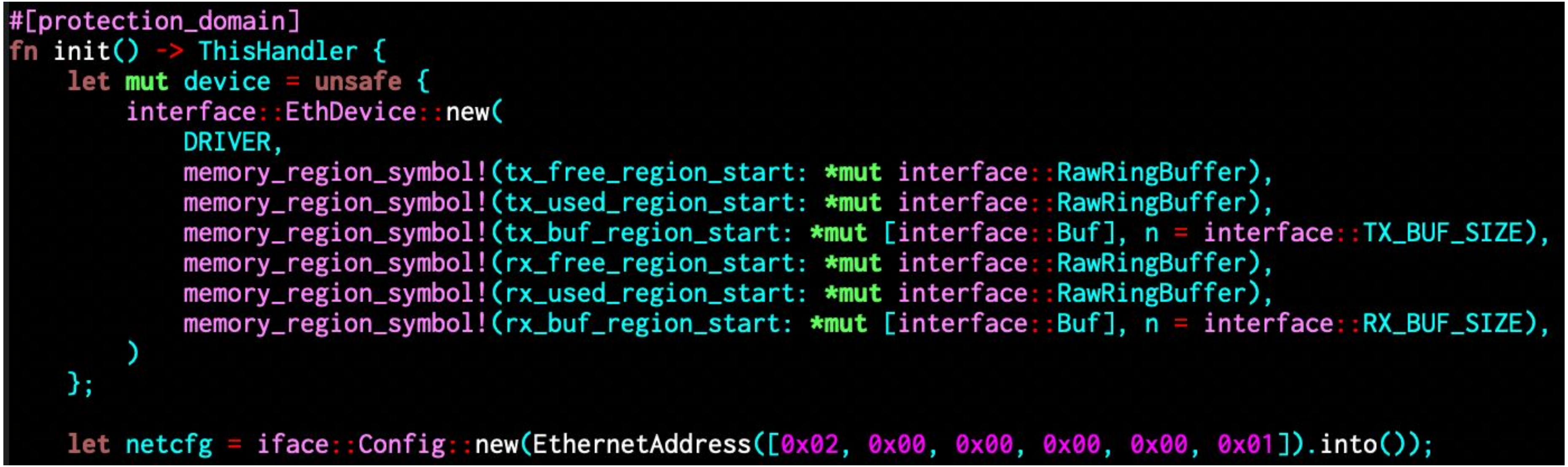


```rust
pub struct EthDevice {
    channel: Channel,
    tx_ring: RingBuffers<'static, ()>,
    tx_bufs: ExternallySharedRef<'static, Bufs, ReadWrite>,
    rx_ring: RingBuffers<'static, ()>,
    rx_bufs: ExternallySharedRef<'static, Bufs, ReadWrite>,
}
```



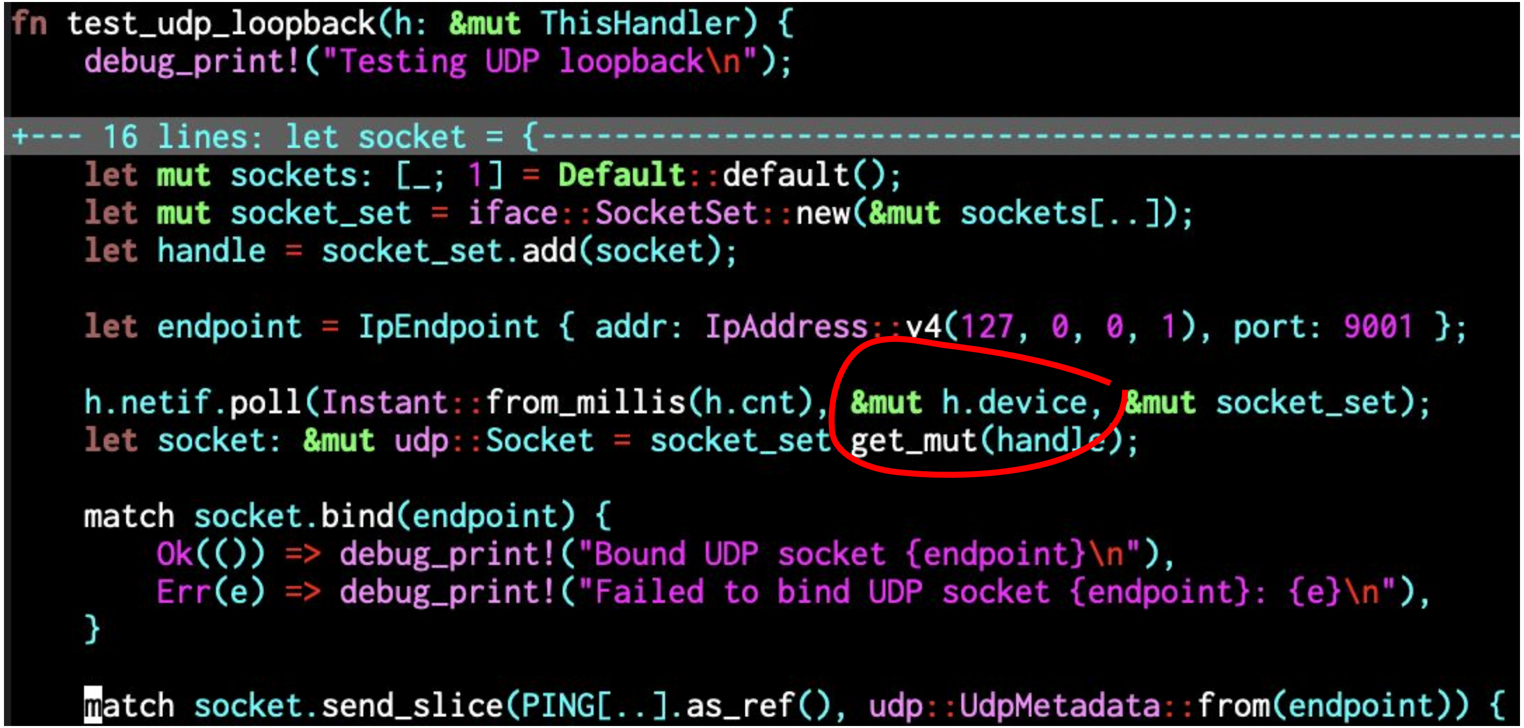
```rust
impl smoltcp::phy::Device for EthDevice {
    type TxToken<'a> = TxToken<'a>;
    type RxToken<'a> = RxToken;

+--- 49 lines: fn receive(&mut self, _timestamp: Instant) -> Option<(Self::RxToken<'_>, Self::TxToken<'_>)> {-

+--- 22 lines: fn transmit(&mut self, _timestamp: Instant) -> Option<Self::TxToken<'_>> {--------------------

+--- 14 lines: fn capabilities(&self) -> phy::DeviceCapabilities {-------------------------------------------
}
```

Ethernet Example: Instantiating the Client



```
#[protection_domain]
fn init() -> ThisHandler {
    let mut device = unsafe {
        interface::EthDevice::new(
            DRIVER,
            memory_region_symbol!(tx_free_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(tx_used_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(tx_buf_region_start: *mut [interface::Buf], n = interface::TX_BUF_SIZE),
            memory_region_symbol!(rx_free_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(rx_used_region_start: *mut interface::RawRingBuffer),
            memory_region_symbol!(rx_buf_region_start: *mut [interface::Buf], n = interface::RX_BUF_SIZE),
        )
    };

    let netcfg = iface::Config::new(EthernetAddress([0x02, 0x00, 0x00, 0x00, 0x00, 0x01]).into());
```

Ethernet Example: Using the Driver Struct



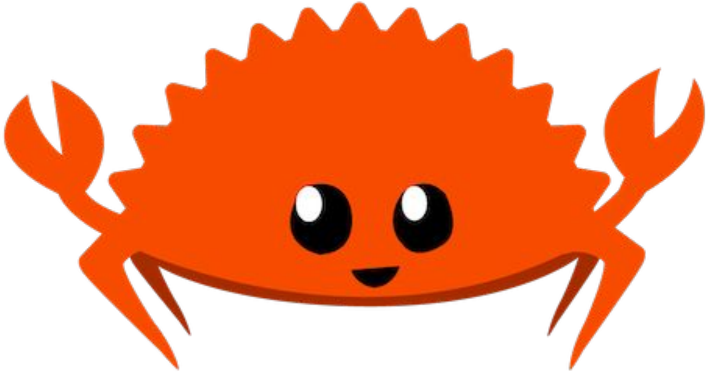
```
fn test_udp_loopback(h: &mut ThisHandler) {
    debug_print!("Testing UDP loopback\n");

+--- 16 lines: let socket = {-----------------------------------------
    let mut sockets: [_; 1] = Default::default();
    let mut socket_set = iface::SocketSet::new(&mut sockets[..]);
    let handle = socket_set.add(socket);

    let endpoint = IpEndpoint { addr: IpAddress::v4(127, 0, 0, 1), port: 9001 };

    h.netif.poll(Instant::from_millis(h.cnt), &mut h.device, &mut socket_set);
    let socket: &mut udp::Socket = socket_set.get_mut(handle);

    match socket.bind(endpoint) {
        Ok(()) => debug_print!("Bound UDP socket {endpoint}\n"),
        Err(e) => debug_print!("Failed to bind UDP socket {endpoint}: {e}\n"),
    }

    match socket.send_slice(PING[..].as_ref(), udp::UdpMetadata::from(endpoint)) {
```

Other traits

- https://docs.rs/embedded-hal/latest/embedded_hal/
- Timer
- SPI
- CAN
- ADC, GPIO, Watchdog
- can be converted for async!

Conclusion & next steps

- Strong Rust embedded ecosystem
- Proper selection of traits supports reusability
- Rapid development on Microkit with Rust!
- asynchronous code!



galois

Distribution Statement A: Approved for public release. Distribution is unlimited.