# Fearless Blinky

...

Bare-metal embedded programming in Rust

# How is Rust applicable in bare-metal domain?

We've seen Rust do well for "hosted" system programming.

(How) do the design goals of Rust apply to bare metal systems?

(How) does the Rust environment scale to environments such as:

- Minimal memory, no dynamic allocations
- Less mainstream, minimal architectures (e.g. no atomics)
- Single-core systems with interrupts
- Memory-mapped I/O, real-world interfaces

# Overview

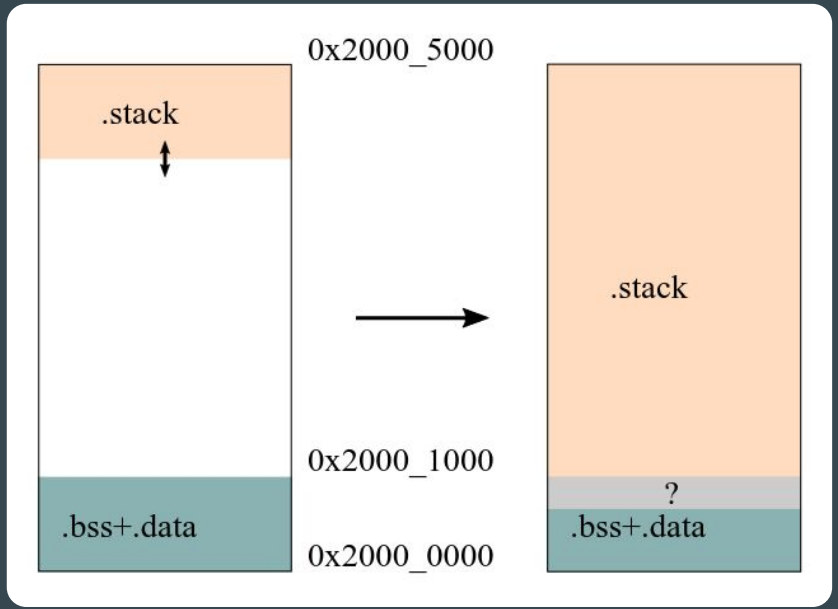
- Brief overview over some of the embedded Rust tooling
- Brief Overview over current embedded Rust ecosystem
- Embedded driver development for simple digital sensors/actuators
  - development of platform-agnostic drivers
  - typestate design pattern
  - testing and mocking of drivers/libraries
- Event-driven non-blocking applications on bare metal
  - RTIC (Real-Time Interrupt-driven Concurrency)

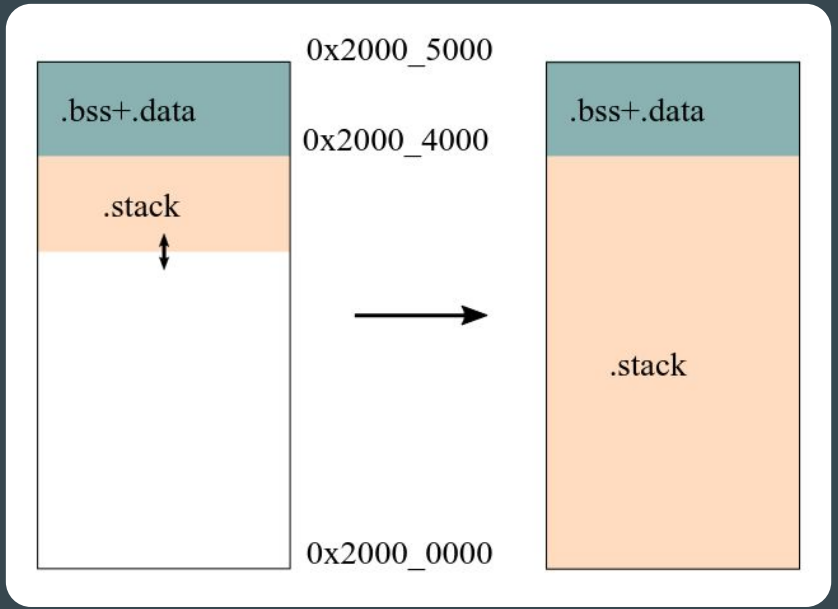
# Tooling

# Rust Embedded Tooling: flip-link

knurling-rs

- flip-link: swap `.stack` and `.data+.bss` to trigger `HardFault` on stack overflow



# Rust Embedded Tooling: flip-link


- Objective:
  - Call stack corrupting data segments would violate memory safety.
  - HardFault exception when end of physical memory is reached maintains memory safety (program aborts)
  - Even better but way harder: prove absence of stack overflow

# Rust Embedded Tooling: defmt


- defmt: defer formatting to host + “compress” strings
  - Only raw data is transferred, not full string
  - Instead of string such as “temperature is {}”, only an ID is transferred
  - Full-featured logging (levels, timestamps, panic/assert print, …)

| Framework | `.text` | relative size | `.rodata` | relative | `.text`+`.rodata` | relative |
|---|---|---|---|---|---|---|
| `core::fmt` | 10348 | 1.0 | 3840 | 1.0 | 14188 | 1.0 |
| `defmt` | 1272 | 0.1229 | 360 | 0.0938 | 1632 | 0.1150 |

https://ferrous-systems.com/blog/defmt/#binary-size

# Rust Embedded Tooling: defmt


- The trait `Format` is required for de/serializing data
- Can be automatically derived

```rust
use defmt::Format;

#[derive(Format)]
struct Header {
    source: u8,
    destination: u8,
    sequence: u16,
}
```

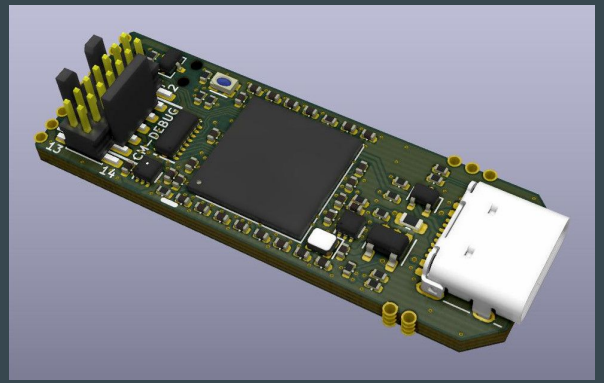
https://ferrous-systems.com/blog/defmt/#binary-size

# Rust Embedded Tooling: probe-rs

probe-rs: Rust toolset for interacting with MCUs over debug probes

probe.rs

- Arm + Risc-V supported (SWD + JTAG)
- Flash, debug, inspect core, dump memory, stacktrace
- Microsoft DAP support for debugging in supporting editors




https://github.com/probe-rs/hs-probe

# Rust Embedded Tooling: probe-rs

probe.rs

```rust
use probe_rs::Probe;

// Get a list of all available debug probes.
let probes = Probe::list_all();

// Use the first probe found.
let probe = probes[0].open()?;

// Attach to a chip.
let session = probe.attach("nrf52")?;

// Select a core.
let core = session.core(0)?;

// Halt the attached core.
core.halt()?;
```

https://probe.rs/docs/library/quickstart/

# Rust Embedded Tooling: cargo-bloat

```
> cargo bloat --release
Analyzing target/thumbv7em-none-eabi/release/client

File  .text    Size          Crate Name
0.1%  10.4%  2.1KiB      rtic_core <(T1,T2,T3,T4,T5,T6,T7)
0.0%   4.5%    962B         shared shared::setup_radio_with_payload_len
0.0%   4.4%    932B      rtic_core <(T1,T2,T3,T4,T5,T6)
0.0%   4.3%    920B            std compiler_builtins::int::specialized_div_rem::u64_div_rem
0.0%   4.1%    878B         client client::app::rtic_ext::main::__rtic_init_resources
0.0%   4.1%    858B  stm32wlxx_hal stm32wlxx_hal::aes::Aes::encrypt_gcm_inplace
0.0%   3.5%    734B  stm32wlxx_hal stm32wlxx_hal::aes::Aes::decrypt_gcm_inplace_u32
0.0%   3.4%    720B            std core::fmt::Formatter::pad
0.0%   2.2%    476B  stm32wlxx_hal stm32wlxx_hal::rtc::Rtc::set_date_time
0.0%   2.0%    428B      defmt_rtt <defmt_rtt::Logger as defmt::traits::Logger>::write
0.0%   2.0%    426B      rtic_core <(T1,T2,T3,T4,T5,T6)
0.0%   1.8%    388B            std core::fmt::write
0.0%   1.8%    374B      [Unknown] EXTI9_5
0.0%   1.8%    372B      [Unknown] ADC
0.0%   1.7%    352B  stm32wlxx_hal stm32wlxx_hal::rcc::sysclk
0.0%   1.7%    350B  stm32wlxx_hal <stm32wlxx_hal::subghz::status::Status as
defmt::traits::Format>::format
```

cargo-bloat

# Crate Shoutout

- postcard: serde de/serializer for efficient bytes-on-wire
- embedded-graphics: abstract over displays, iterator-based for efficiency + safety
- app-template: quick project setup for many targets
    - includes flip-link, defmt, probe-run
- awesome-embedded-rust
- not-yet-awesome-embedded-rust

# Ecosystem

# Embedded **H**ardware **A**bstraction **L**ayer: Common Traits

- A set of traits which abstract over SPI, I2C, ADC, timers, serial, ...
- Defines a common base for Rust on MCU/embedded targets

- Allows development of drivers which are platform-agnostic
- Common/similar usage of HALs for different targets
  - Learning advantage
  - Easier to switch hardware, too

docs.rs/embedded-hal

embedded-hal

# Embedded HAL: Drivers

- Driver for I2C chip requires some I2C interface. It could be:
  - STM32 i2c peripheral
  - FPGA with driver exposed over linux i2c device
  - Raspberry Pi i2c peripheral
  - Bitbanged i2c “peripheral” using timer + GPIO (+ interrupts, optionally)
  - Mocked software interface for testing

We’ll see example later :)

# Embedded HAL: Implementations

- HAL traits are implemented for each target/target family separately
- Room for specifics though, like DMA, extension traits for special peripheral features, or unique peripheral support
- For ARM and RISC-V targets, relies on PAC (Peripheral Abstraction Crate)
  - Like traditional C headers with registers/offsets/values
  - But, enforces some rules already (Read/Write/ReadWrite/Constant)
  - Relies on vendor-provided SVD files (xml files with register descriptions)

https://github.com/stm32-rs/stm32h7xx-hal

hal-implementation-crates

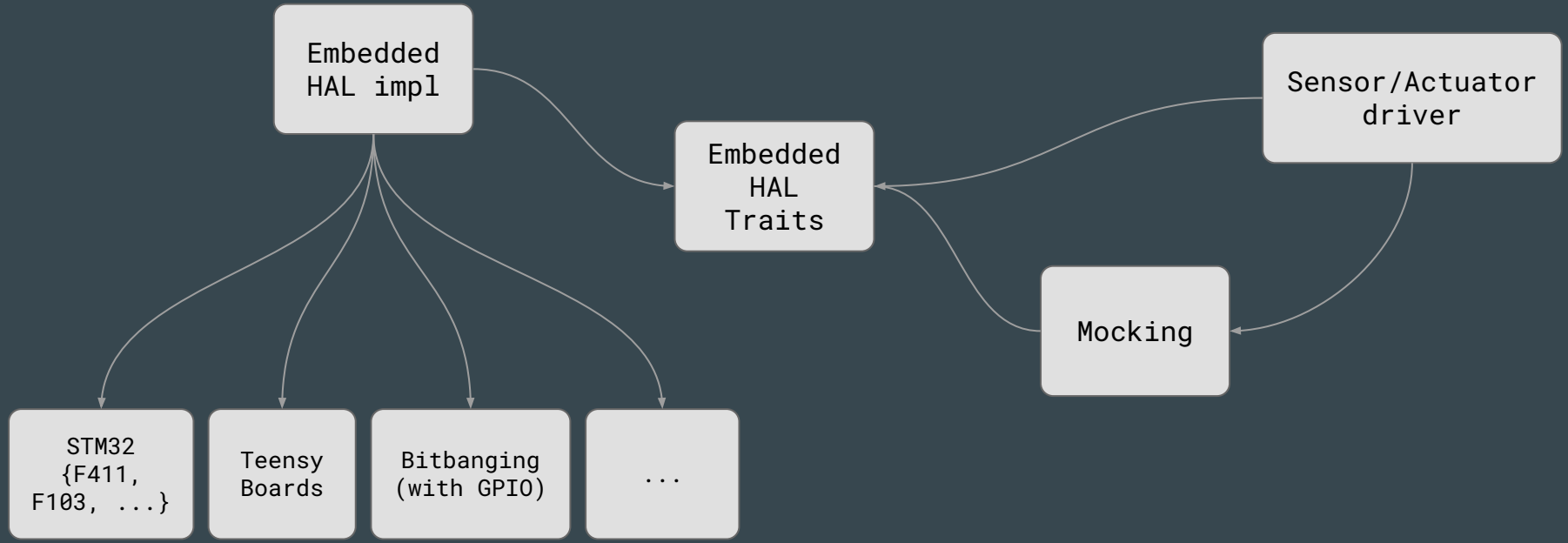
# Embedded HAL: PAC Code

- Read/Write/ReadWrite/... is enforced here
- Closures aren't so bad. Are always optimized in the case of PACs :)

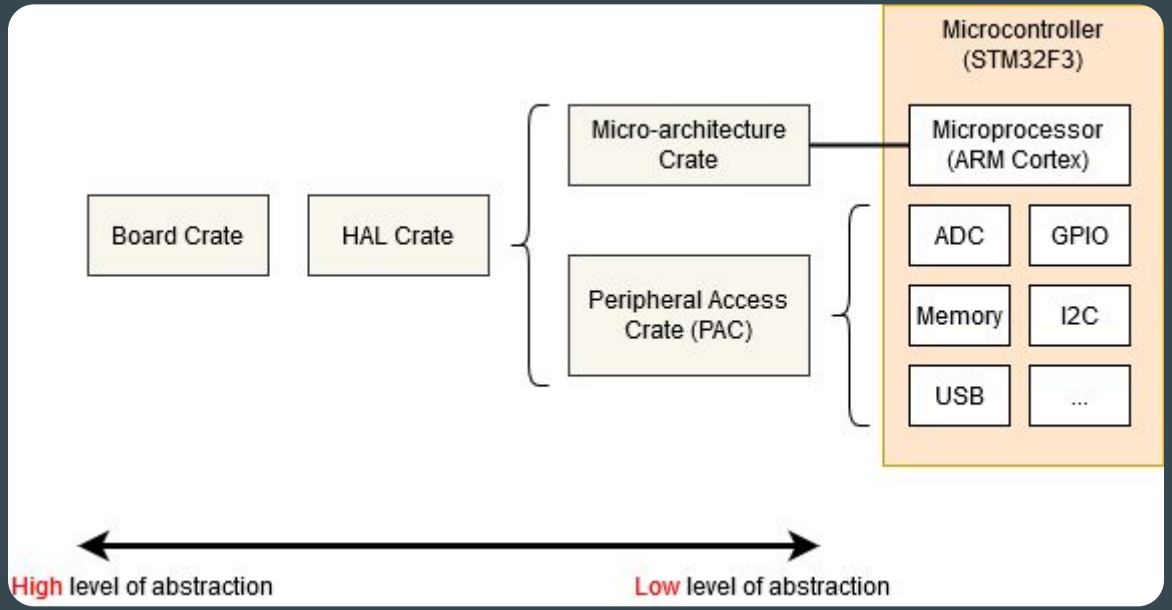
```
dp.I2C1.icr.reset();

dp.I2C1.timingr.write(|w| w.bits(0x0000020B));
dp.I2C1.cr2.modify(|_, w| w.autoend().set_bit());
dp.I2C1.oar1.modify(|_, w| w.oa1en().clear_bit());
dp.I2C1.oar2.modify(|_, w| w.oa2en().clear_bit());
dp.I2C1.cr1.modify(|_, w| w.gcen().clear_bit());
dp.I2C1.cr1.modify(|_, w| w.nostretch().clear_bit());
dp.I2C1.cr1.modify(|_, w| w.pe().clear_bit());
```

stm32f0-smbus-module

# Embedded HAL Birds Eye View



# Embedded HAL Birds Eye View




embedded-book/start/registers.html

# Driver Development

# Driver development for embedded-hal

- Pick from embedded-hal the traits for the required interfaces/peripherals
- Implement the device API in the generic terms using these traits
- The driver is a normal library
  - unit tests using embedded-hal-mock are recommended
  - share normally over crates.io
  - can use other dependencies as any normal crate (`#![no_std]` though)
- Type-state design pattern is frequently used
- Give some consideration to owning or sharing resources, e.g. buses
- Test live with raspberry pi or similar (linux-embedded-hal)

# Anemometer design - Sensor

“Where is the wind coming from,

and how strong is it?”

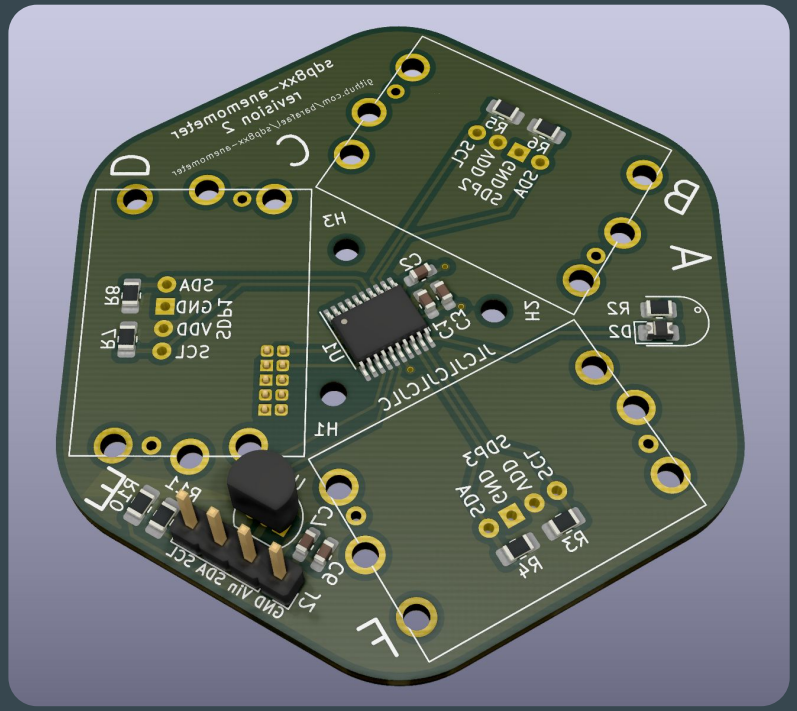
Sensirion SDP8xx
Differential Pressure Sensor

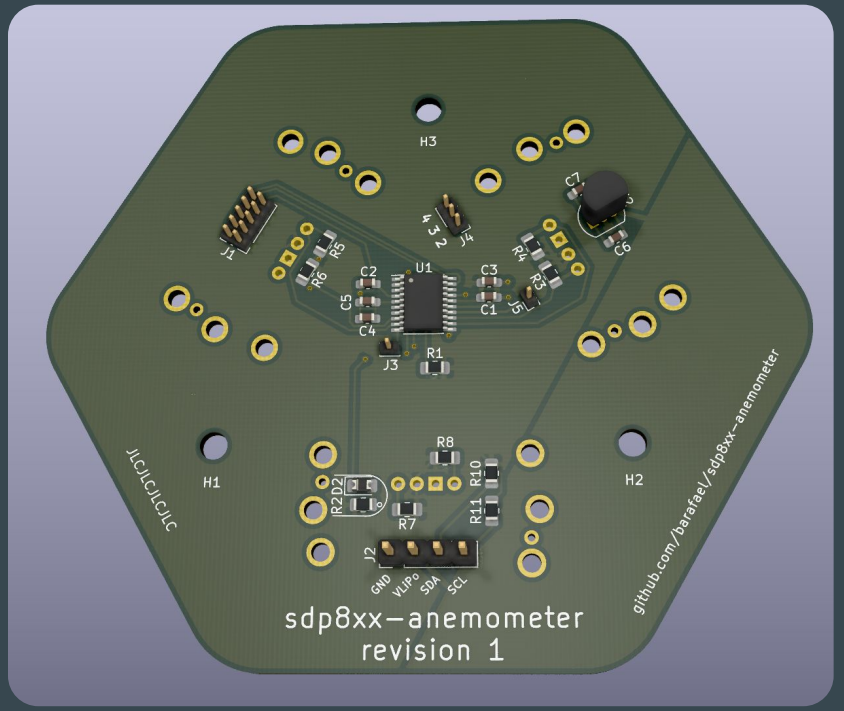
- Sensitive measurement of pressure differences
- “Wind is just directional pressure difference”
- I2C interface


https://www.sensirion.com/de/durchflusssensoren/differenzdrucksensoren/sdp800-proven-and-improved/

# Anemometer design - Board Renders

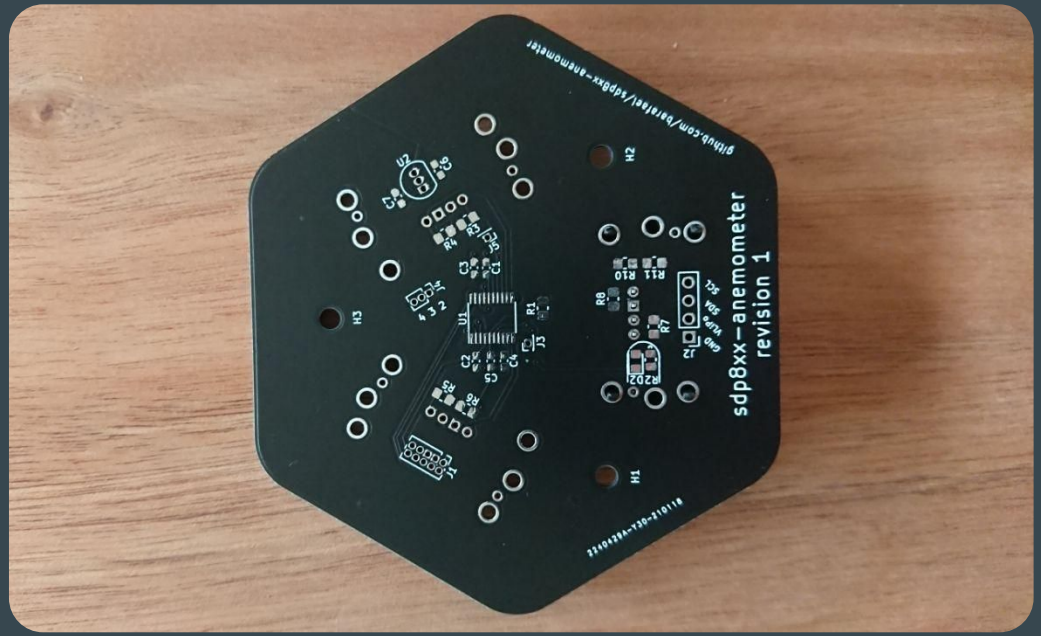
https://github.com/barafael/sdp8xx-anemometer-pcb

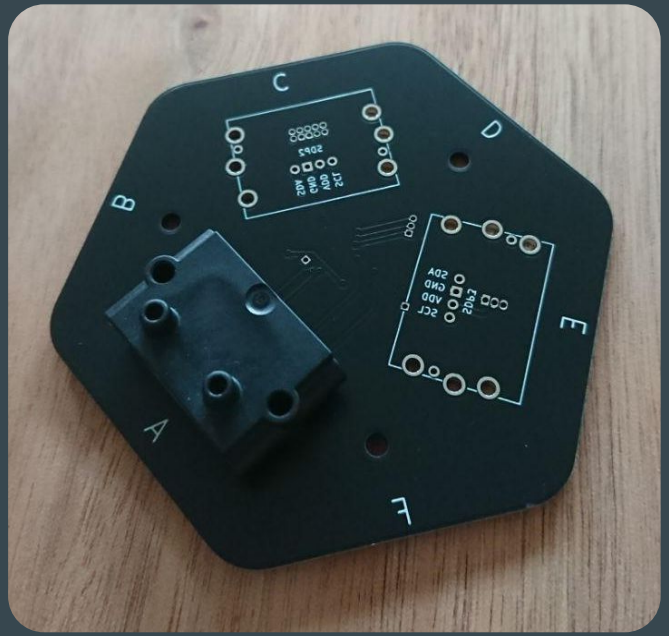




# Anemometer design - Finished PCBs

https://github.com/barafael/sdp8xx-anemometer-pcb



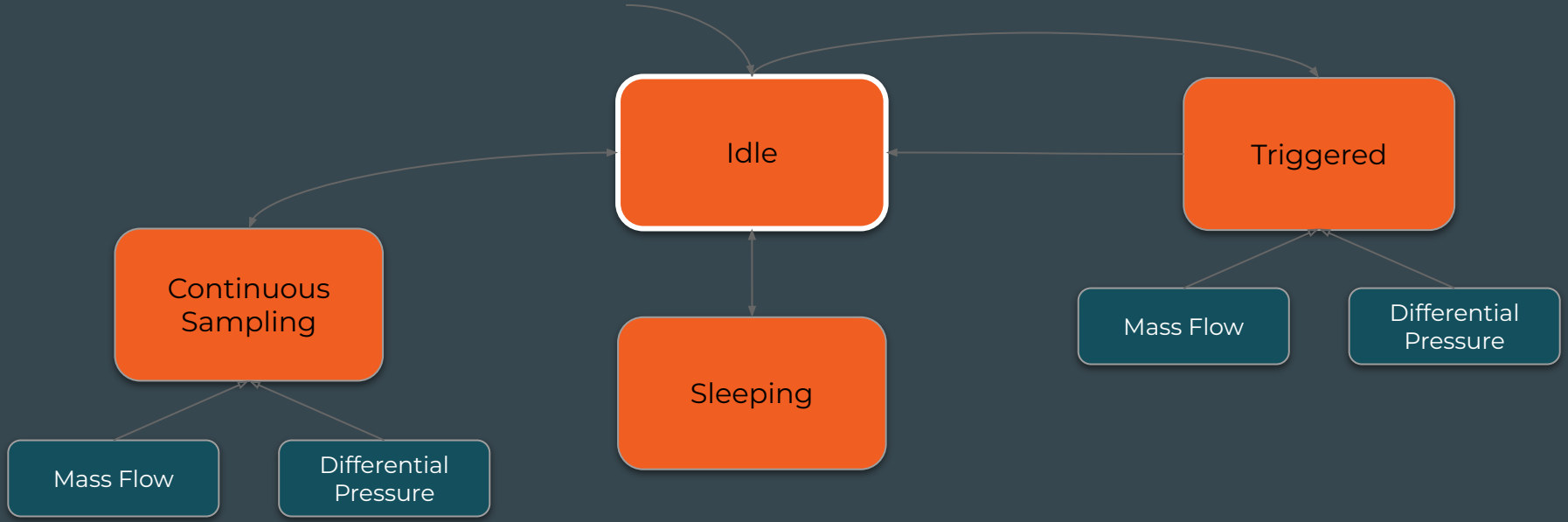



# Anemometer design - Finished Module


3D-Printed “Air Pods”

# State Machine? What?

SDP8xx Sensor Datasheet lists operation modes




https://github.com/barafael/sdp8xx-rs

datasheet:
https://www.sensirion.com/de/durchflusssensoren/differenzdrucksensoren/sdp800-proven-and-improved/

# How to implement? YOLO Approach

```
readBuffer == "AT+C001";
digitalWrite(setPin, LOW);        // Set HC-12 into AT Command mode
delay(100);                       // Wait for the HC-12 to enter AT Command mode
HC12.print(readBuffer);           // Send AT Command to HC-12 ("AT+C001")
delay(200);
while (HC12.available()) {        // If HC-12 has data (the AT Command response)
  Serial.write(HC12.read());      // Send the data to Serial monitor
}
Serial.println("Channel successfully changed");
digitalWrite(setPin, HIGH);       // Exit AT Command mode
readBuffer = "";
```

# How to implement? Enum Approach

Javadoc path: **com.google.gdata.data.projecthosting**
**Enum State.Value**

An issue can be:

| Enum Constant Summary |
| --- |
| **CLOSED**<br>Closed state. |
| **OPEN**<br>Open state. |

com.google.gdata.data.projecthosting

## Enum State.Value

java.lang.Object
  └ java.lang.Enum<State.Value>
    └ **com.google.gdata.data.projecthosting.State.Value**

**All Implemented Interfaces:**
java.io.Serializable, java.lang.Comparable<State.Value>

**Enclosing class:**
State

---

public static enum **State.Value**
extends java.lang.Enum<State.Value>

Value.

Issue state needs to be checked for each operation.

An issue can be closed more than once.

State-local data “validity” depends on values of other variables.

```
function process(state, event) {
    switch (state) {
        case 'start':
            if (event === 'SUBMIT') {
                return 'loading';
            }
            break;
        case 'loading':
            if (event === 'RESOLVE') {
                return 'success';
            } else if (event === 'REJECT') {
                return 'error';
            }
            break;
        case 'success':
            // Accept no further events
            break;
        case 'error':
            if (event === 'SUBMIT') {
                return 'loading';
            }
            break;
        default:
            // This should never occur
            return undefined;
    }
}
```

- Easy
- Runtime state management
- Illegal states/events/transitions must be handled
- May require Union/Option/... for state-local data

https://24ways.org/2018/state-machines-in-user-interfaces/

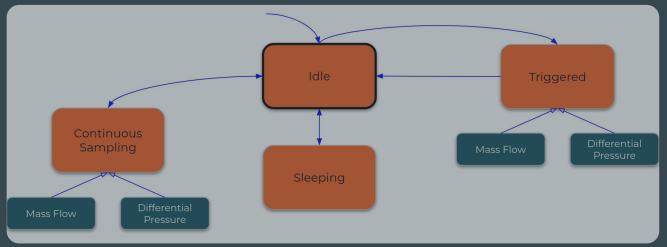
# Typestate Programming

# Type System Approach

- States are types (with optional data)
- Transitions are functions
  - Signatures state transitions from type to type
  - Perform necessary I/O, communication, etc.
- Pass-by-move
  - Leaving a state … leaves no state (globals etc.)

# Type System Approach

State the operation modes as types




```rust
pub struct IdleState;
pub struct TriggeredState;
pub struct SleepState;

pub struct ContinuousSamplingState<MeasurementType> {
    meas_type: PhantomData<MeasurementType>,
}

pub struct DifferentialPressure;
pub struct MassFlow;
```

https://github.com/barafael/sdp8xx-rs/blob/main/sdp8xx/src/states.rs

# Type System Approach

Define the possible state transitions as type aliases

```rust
/// Transition from Idle to Sleep state
pub type IdleToSleep<I2C, D> = Sdp8xx<I2C, D, SleepState>;

/// Transition from Sleep to Idle state
pub type SleepToIdle<I2C, D> = Result<Sdp8xx<I2C, D, IdleState>, Error<I2C>>;
```

Why use `Result<T, E>`?: Some transitions are fallible.

sdp8xx-rs

# Type System Approach

Implement the possible state transitions as **owning** methods

```rust
/// Transition from Idle to Sleep state
pub fn go_to_sleep(mut self) -> IdleToSleep<I2C, D>;

/// Transition from Sleep to Idle state
pub fn wake_up(mut self) -> SleepToIdle<I2C, D>;
```

sdp8xx-rs

# Type System Approach

Construction of initial state requires **ownership** of resources!

```
impl<I2C, D, E> Sdp8xx<I2C, D, IdleState>
    where /* bounds */ {
    pub fn new(i2c: I2C, address: u8, delay: D) -> Self {
        Sdp8xx {
            i2c,
            address,
            delay,
            state: PhantomData::<IdleState>,
        }
    }
}
```

sdp8xx-rs

# Type System Approach

Transition from Sleep state to Idle state:

```
impl<I2C, D, E> Sdp8xx<I2C, D, SleepState>
    where /* bounds */ {
    pub fn wake_up(mut self) -> SleepToIdle<I2C, D> {
        self.send_command(Command::WakeUp)?;
        // some more work here ...
        Ok(Sdp8xx {
            i2c: self.i2c,
            address: self.address,
            delay: self.delay,
            state: PhantomData::<IdleState>,
        })
    }
}
```

sdp8xx-rs

# Crucial Rust ingredients for Typestate

- Move Semantics: State transitions
- Modularity: Hide boilerplate for good API
- Visibility Rules: Prevent invalid instantiations
- Zero-Sized Types: Make it cheap

# Typestate Advantages

- Operations and data are available only in “their” state
- “Function call ordering” is enforced **upwards**
- No memory used for empty states, it’s just a static marker

```
core::mem::size_of::<ContinuousSamplingState<MassFlow>>() == 0
```

cd74hc4067-rs/coverage.pdf

# Typestate Advantages

- Impossible to misuse
- “Fluent API”
- Autocomplete goodness
- Relatively friendly compiler errors

```
mismatched types
expected struct `SwitchMonitor<switch_monitor::Active>`
   found struct `SwitchMonitor<switch_monitor::Passive>`
```

# Example: Sensirion Pressure Sensor

```rust
#[test]
fn go_to_sleep() {
    let bytes: [u8; 2] = Command::EnterSleepMode.into();
    let expectations = [
        Transaction::write(0x25, bytes.into()),
        /* dummy data */ Transaction::write(0x25, vec![]),
        Transaction::write(0x25, vec![]).with_error(MockError::Io(ErrorKind::Other)),
        Transaction::write(0x25, vec![]),
    ];
    let mock = I2cMock::new(&expectations);
    let sdp = Sdp8xx::new(mock, 0x25, DelayMock);
    let sleeping = sdp.go_to_sleep().unwrap();
    let sdp = sleeping.wake_up().unwrap();
    sdp.release().done();
}
```

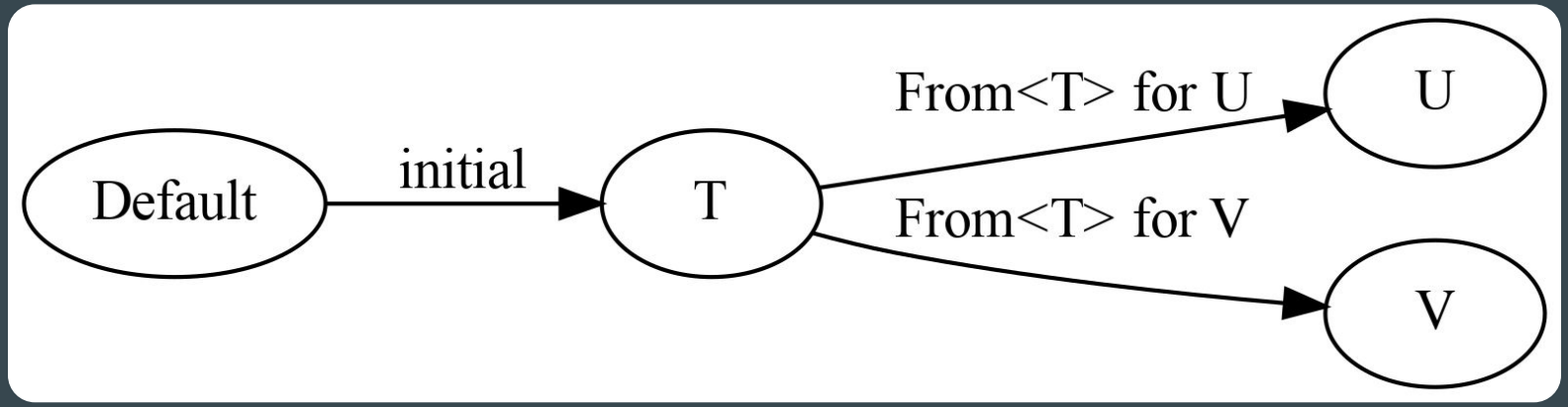
sdp8xx-rs

# Example: Hc-12 Wireless Module

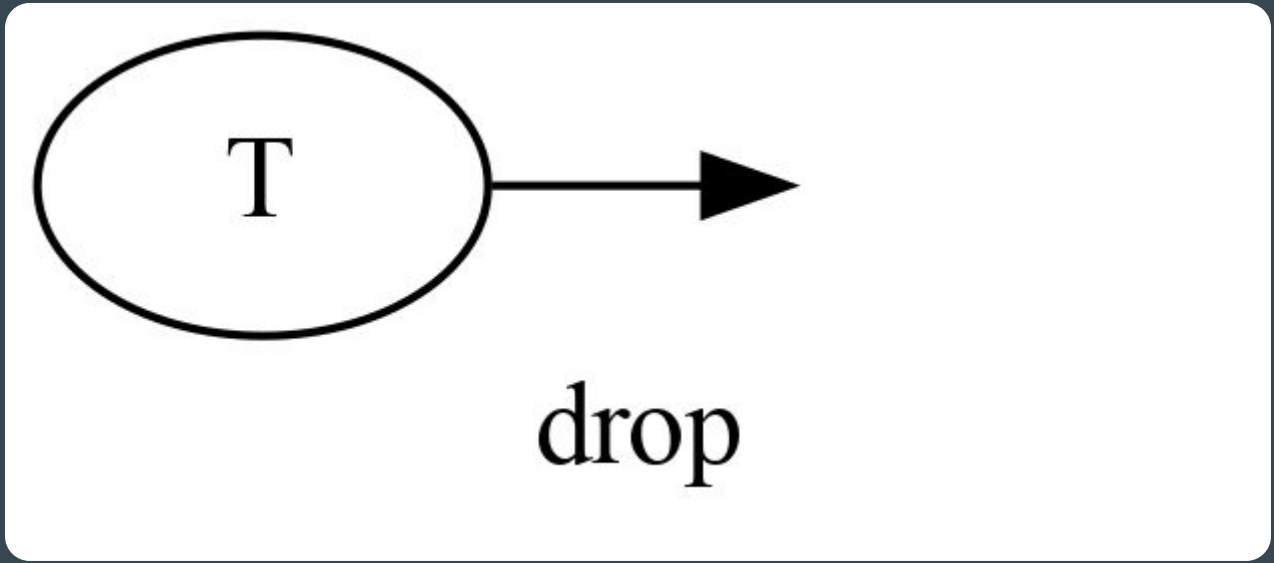
```
let serial = serial::Mock::new(&transactions);
let hc12 = Hc12::new(serial, set_pin, delay);
let mut hc12 = hc12.into_configuration_mode().unwrap();

assert!(hc12.is_ok());

assert_eq!(hc12.get_version(), b"VERSION-42\r\n");

let params = hc12.get_parameters().unwrap();
assert_eq!(...);

let mut hc12 = hc12.into_normal_mode().unwrap();

hc12.write_buffer(b"some data bla bla\r\n").unwrap();
```

sdp8xx-rs

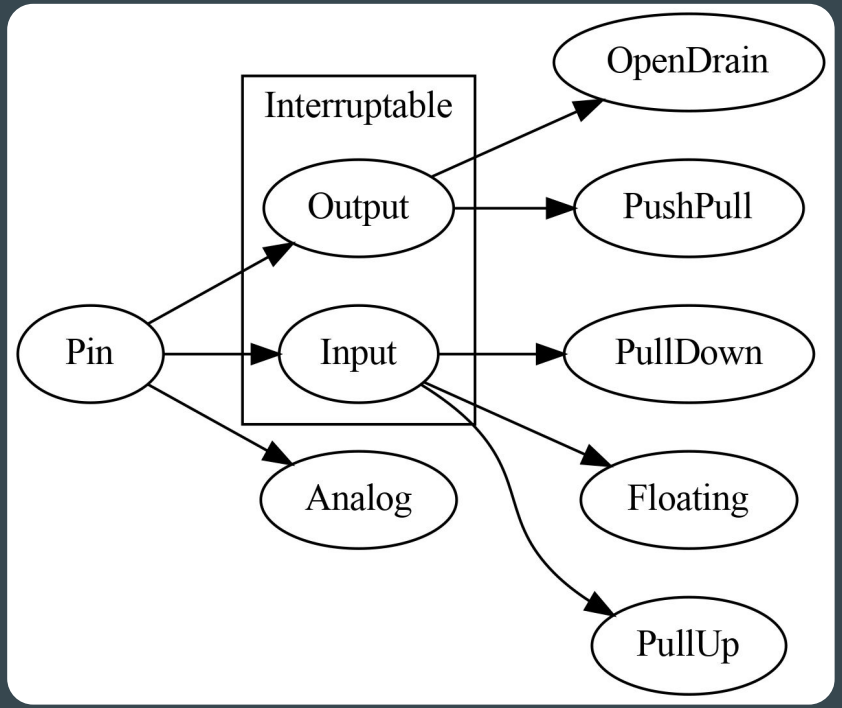
# Example: Default and From Traits



# Example: Drop :)



# Example: GPIO Pins in stm32f4 HAL




stm32-rs/stm32f4xx-hal/src/gpio/convert.rs

# However: Deep Type State

```
let tx: TxMode<
    embedded_nrf24l01::NRF24L01<
        Infallible,
        gpio::gpioa::PA8<Output<PushPull>>,
        stm32f0xx_hal::gpio::gpioa::PA10<Output<PushPull>>,
        Spi<
            stm32::SPI1,
            gpio::gpioa::PA5<Alternate<gpio::AF0>>,
            gpio::gpioa::PA6<Alternate<gpio::AF0>>,
            gpio::gpioa::PA7<Alternate<gpio::AF0>>,
        >,
    >,
> = nrf24.tx().unwrap();
```

# However: Deep Type State

```
let tx: TxMode<
    embedded_nrf24l01::NRF24L01<
        Infallible,
        gpio::gpioa::PA8<Output<PushPull>>,
        stm32f0xx_hal::gpio::gpioa::PA10<Output<PushPull>>,
        Spi<
            stm32::SPI1,
            gpio::gpioa::PA5<Alternate<gpio::AF0>>,
            gpio::gpioa::PA6<Alternate<gpio::AF0>>,
            gpio::gpioa::PA7<Alternate<gpio::AF0>>,
        >,
    >,
> = nrf24.tx().unwrap();
```

Type Inference helps a little

# However: Deep Type State

```
let tx: NrfModuleTx<
        Irq,
        CsPin,
        Spi6,
    > = nrf24.tx().unwrap();
```

Type aliases may help some more
(it's just another expression)

# Crate shoutout: typestate-rs

typestate-rs does all that behind the curtains:

```rust
#[typestate]
mod traffic_light {
    #[automaton]
    pub struct TrafficLight {
        pub cycles: u64,
    }

    #[state] pub struct Green;
    #[state] pub struct Yellow;
    #[state] pub struct Red;
}
```

Now also generates dot files!

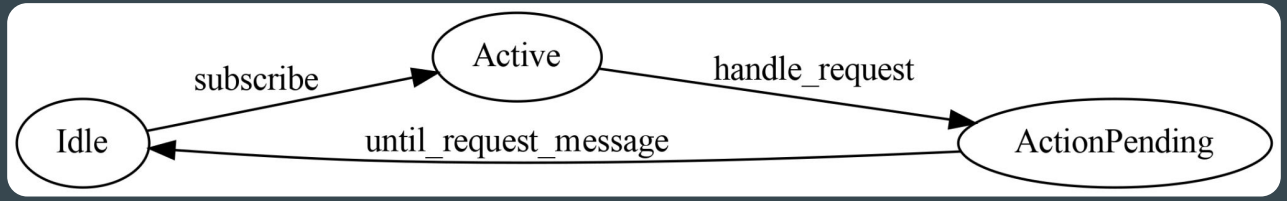
https://github.com/rustype/typestate-rs

# Typestate + Dot = ♡

generate/manually create `.dot` file describing your state machine:

```
digraph G {
    rankdir="LR"

    Idle -> Active [label = "subscribe"]
    Active -> ActionPending [label = "handle_request"]
    ActionPending -> Idle [label = "until_request_message"]
}
```

# Typestate + Dot = ♡



# Downsides

- embedded-hal isn’t v1.0 yet (but very soon)
- All embedded-hal traits are blocking, as of now
- Drivers may have to be adapted for a future asynchronous e-hal
- Lots of flux in different bare metal crates (used to be way worse)
- HAL support varies among platforms
- Reliance on binary blobs (certified BLE stacks and others)

# RTIC

Real-time Interrupt-driven Concurrency

rtic.rs

# RTIC: Real-time Interrupt-driven Concurrency

- Framework for event-driven real-time applications (not RTOS)
- Run-to-completion tasks (which are just interrupt handlers)
- Preemptive multitasking without software scheduler
  - `ARM NVIC` hardware is cleverly used
- Apart from normal Rust guarantees, also (statically) prevents deadlocks and priority inversion (**Priority Ceiling Protocol**)
- Software-triggered tasks, timer queue, message passing

minimal app

rtic.rs

# RTIC: Real-time Interrupt-driven Concurrency

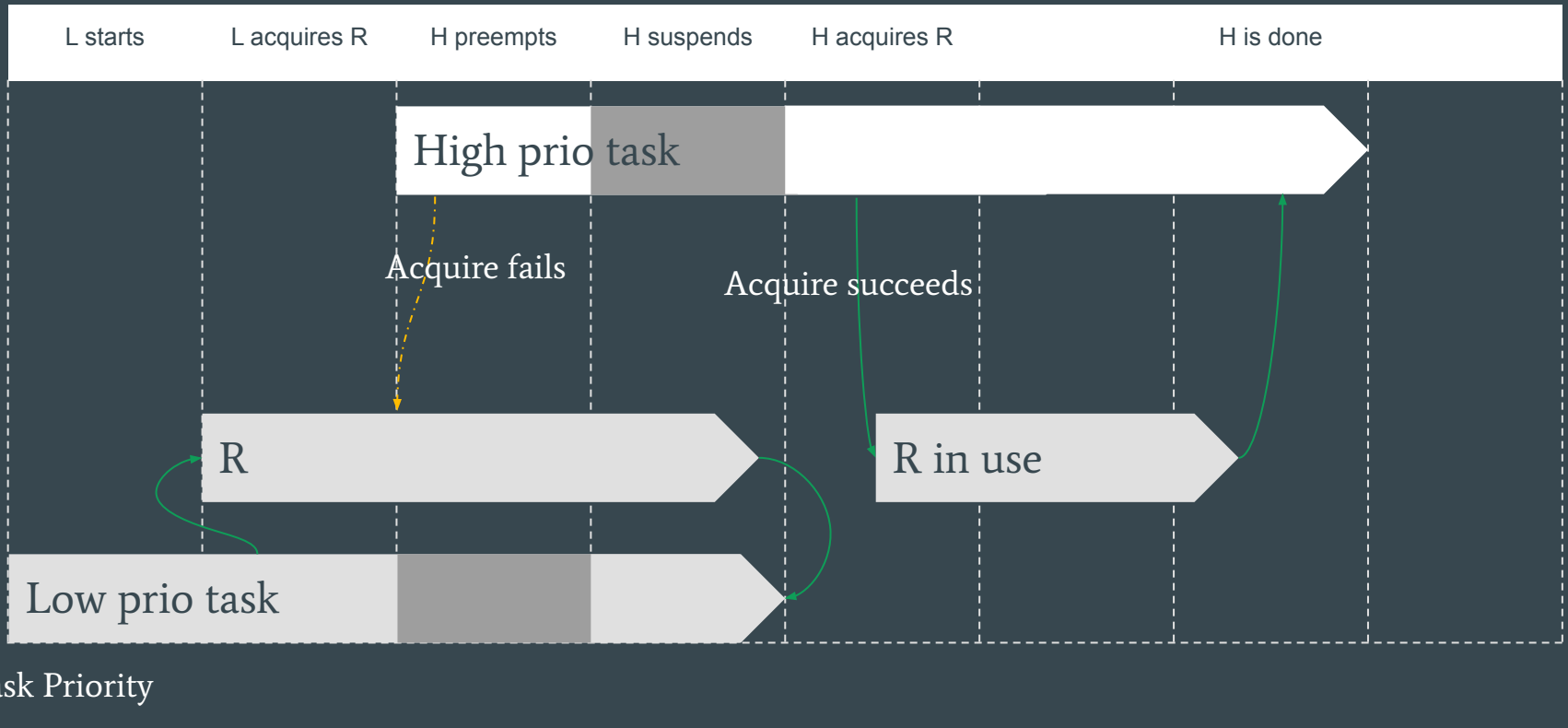
- Uses registers defined in PAC (=> supports all Cortex-M devices)
- Single call stack, extremely low memory usage
- No dynamic memory management required
- Principles also applicable for raspberry pi bare metal or even linux userspace
- Locking a resource is one write to `BASEPRI` or free
- Plays well with HAL implementations for interrupt, timer etc. setup

# RTIC: Real-time Interrupt-driven Concurrency

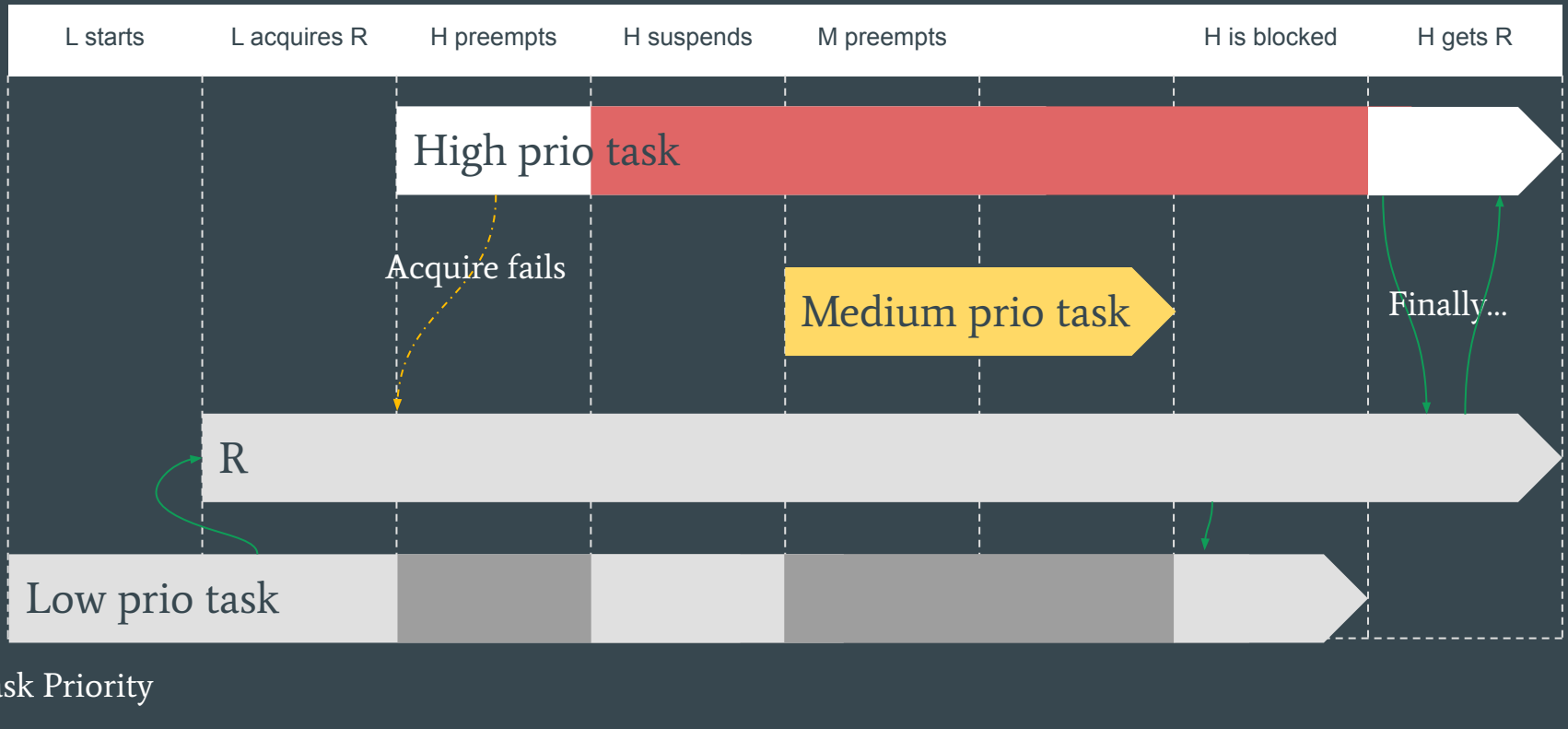
- Priority Ceiling Protocol
  - Task A locks a resource => A temporarily gets a higher priority P
  - P chosen such that other tasks using the locked resource cannot preempt A
  - Simply the highest priority among the tasks which share a resource :)
  - At worst, single write to `BASEPRI` does the job

- PCP prevents:
  - Priority inversion (medium prio task cannot preempt)
  - Deadlock (higher priority task cannot preempt)

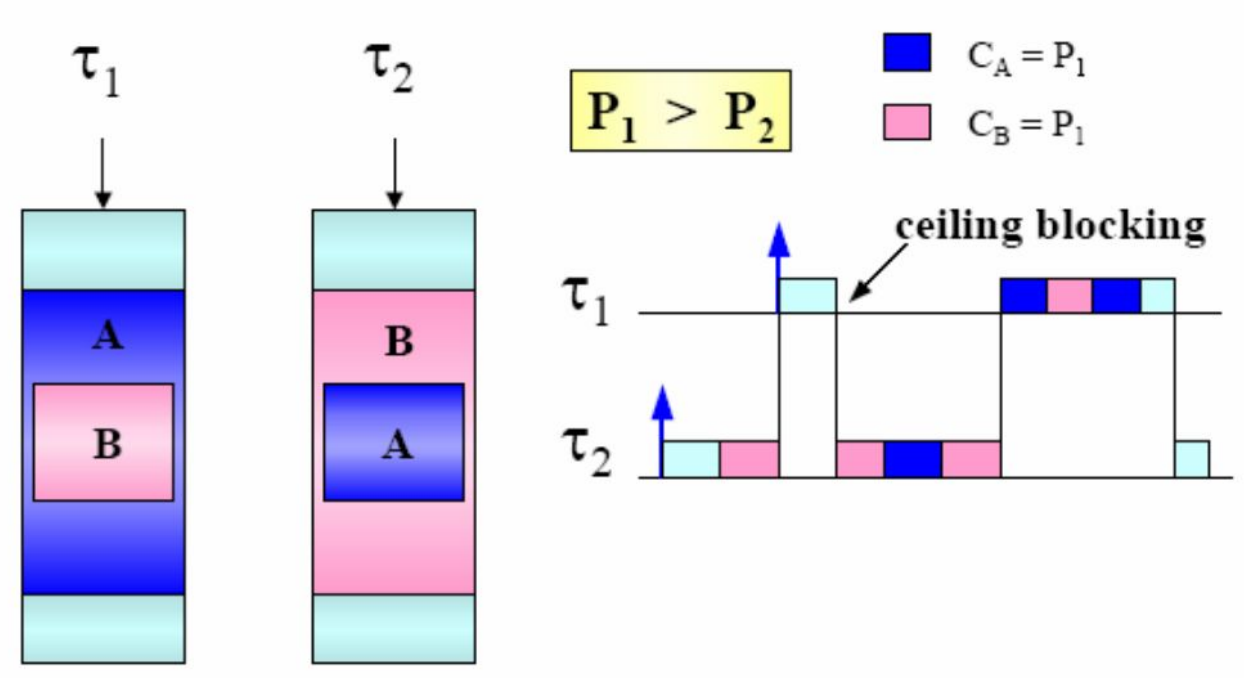
# Normal preemptive scheduling with priorities



# Priority Inversion



# Deadlock

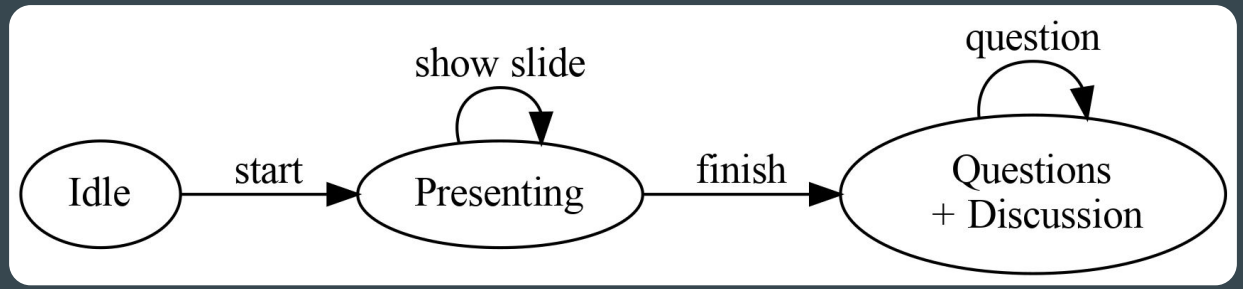



J. Ras and A. M. K. Cheng, "An Evaluation of the Dynamic and Static Multiprocessor Priority Ceiling Protocol and the Multiprocessor Stack Resource Policy in an SMP System," 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009, pp. 13-22, doi: 10.1109/RTAS.2009.10.

# RTIC + Embedded Rust Safety Tools

- WCET (worst-case execution analysis) using `cargo-call-stack`
- `cargo klee` symbolic abstract interpretation of Rust programs
  - Panic absence, liveness, refactoring equivalence
- Heapless crate (dynamic bounded data structures)
  - Memory-safe + panic-free
- Unit testing with RTIC? Should be possible, but might require significant virtualization of core?

# Thanks :)




You are here

https://github.com/barafael rafael.bachmann.93@gmail.com linkedin.com/in/barafael