Rust Embedded and Async

Ulf Lilleengen lulf

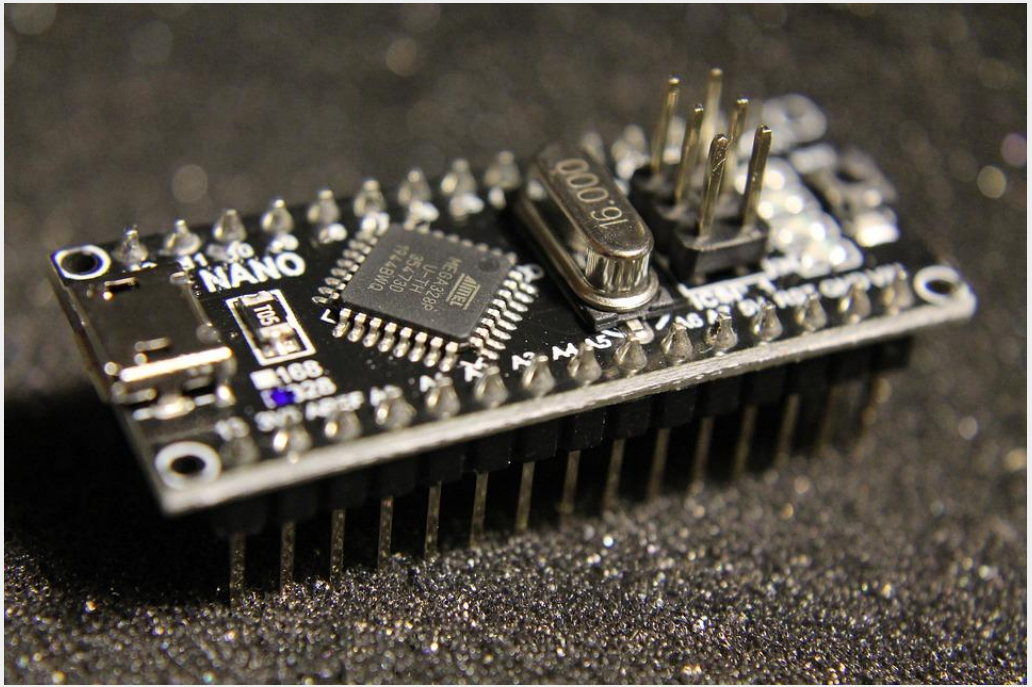
Target hardware

16kB - 512kB of RAM

128kB - 2MB of Flash

No operating system

No memory allocator

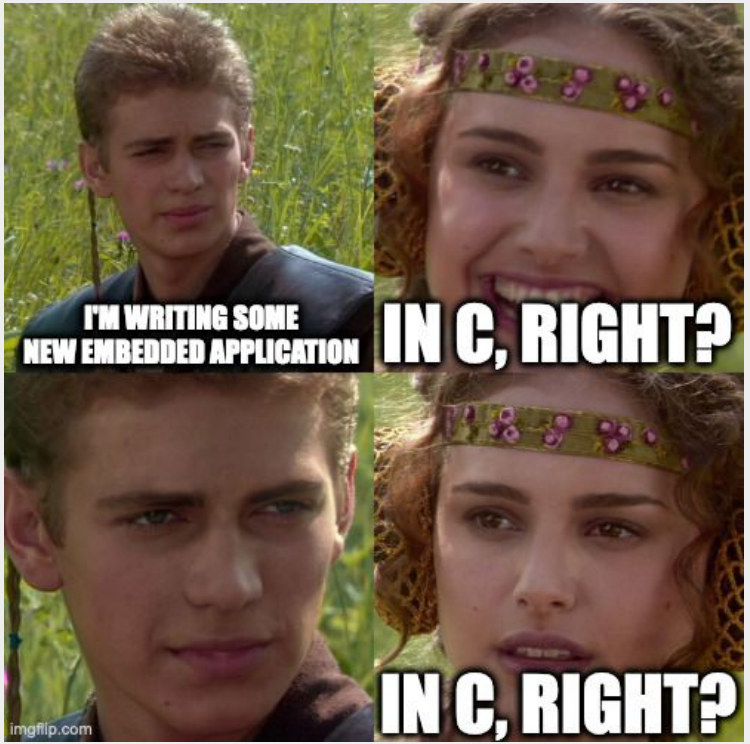


Let's make progress

Everybody writes in C

RTOS and SDKs written in C

Why does it have to be this way?



What makes Rust great

Traits

Dependency management

Safety

Embedded Rust

no_std Rust

feature	no_std	std
heap (dynamic memory)	*	✓
collections (Vec, HashMap, etc)	**	✓
stack overflow protection	✘	✓
runs init code before main	✘	✓
libstd available	✘	✓
libcore available	✓	✓
writing firmware, kernel, or bootloader code	✓	✘

Source: https://docs.rust-embedded.org/book

Ecosystem

Microcontroller

Source: https://docs.rust-embedded.org/book

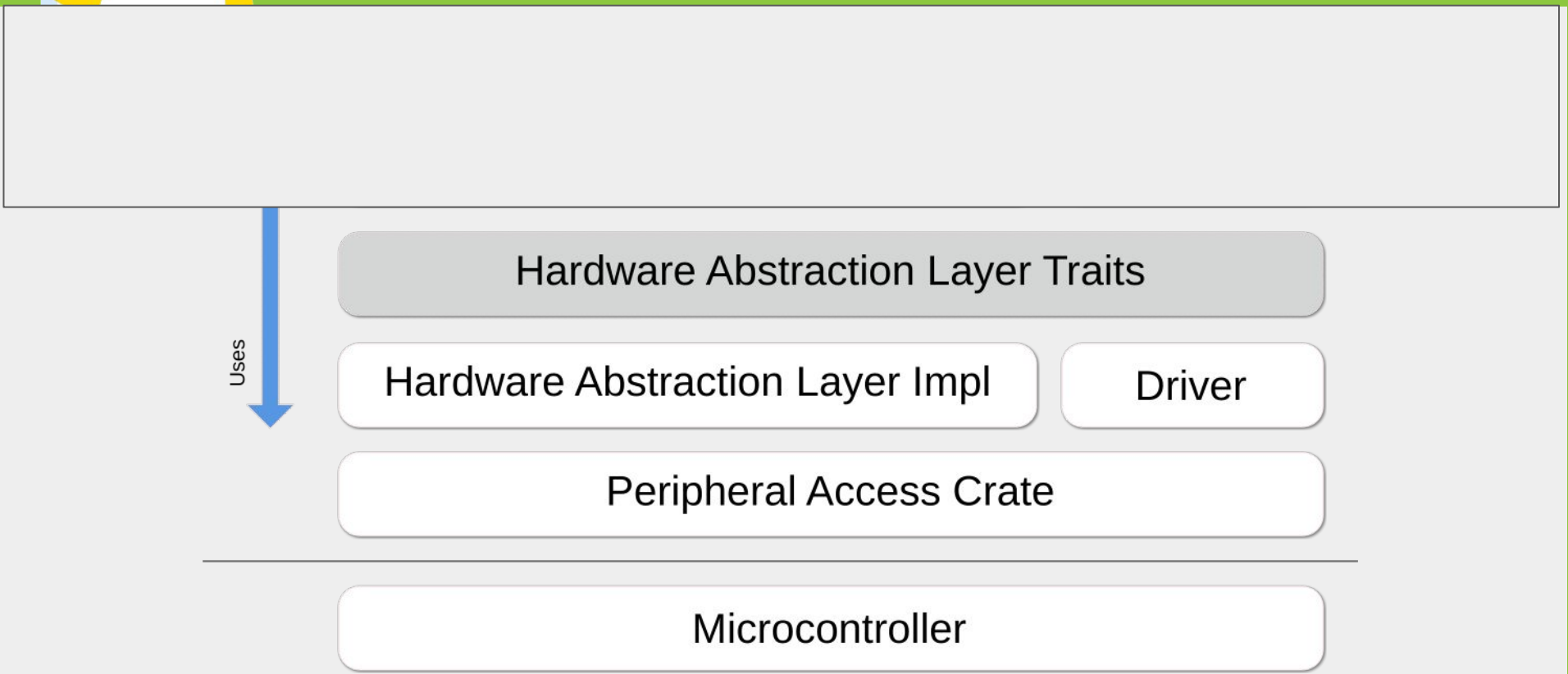
Ecosystem

Peripheral Access Crate

Microcontroller

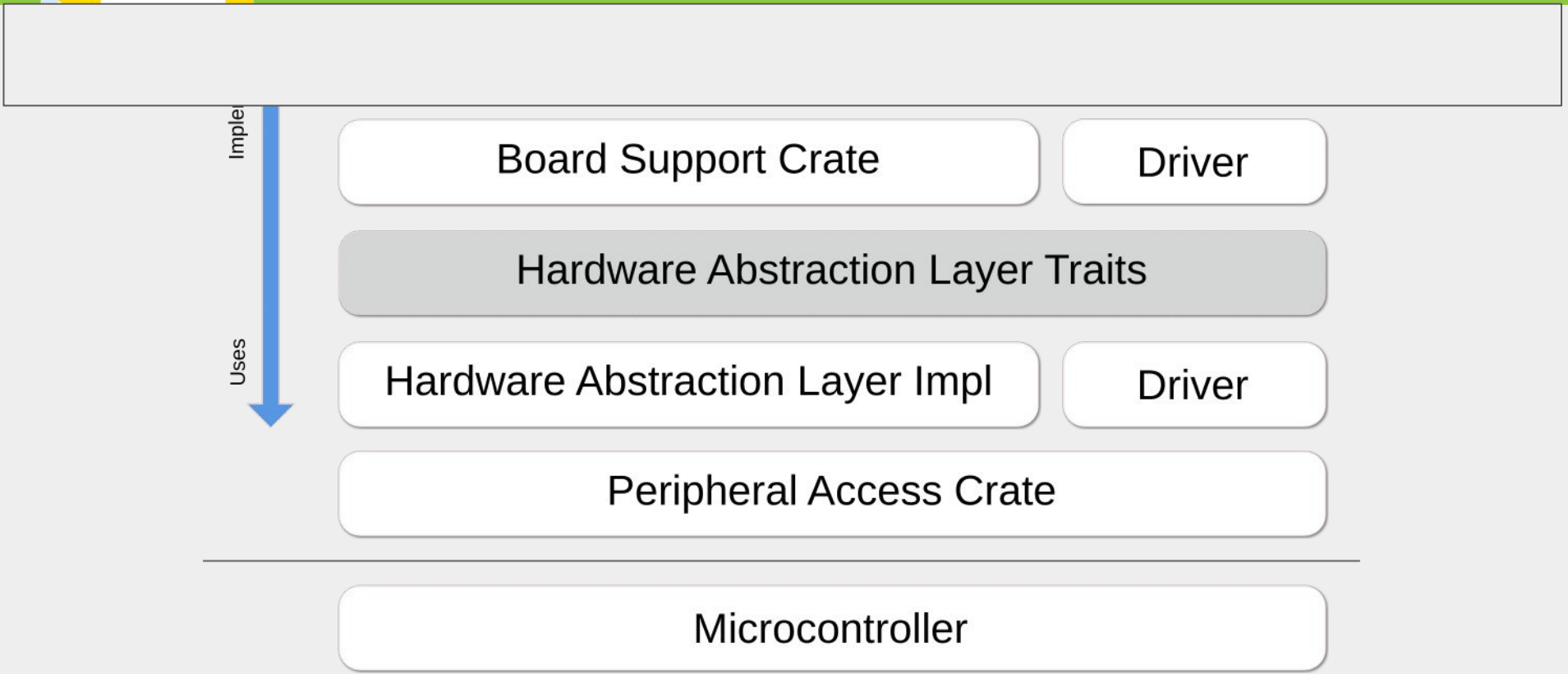
Source: https://docs.rust-embedded.org/book

Ecosystem



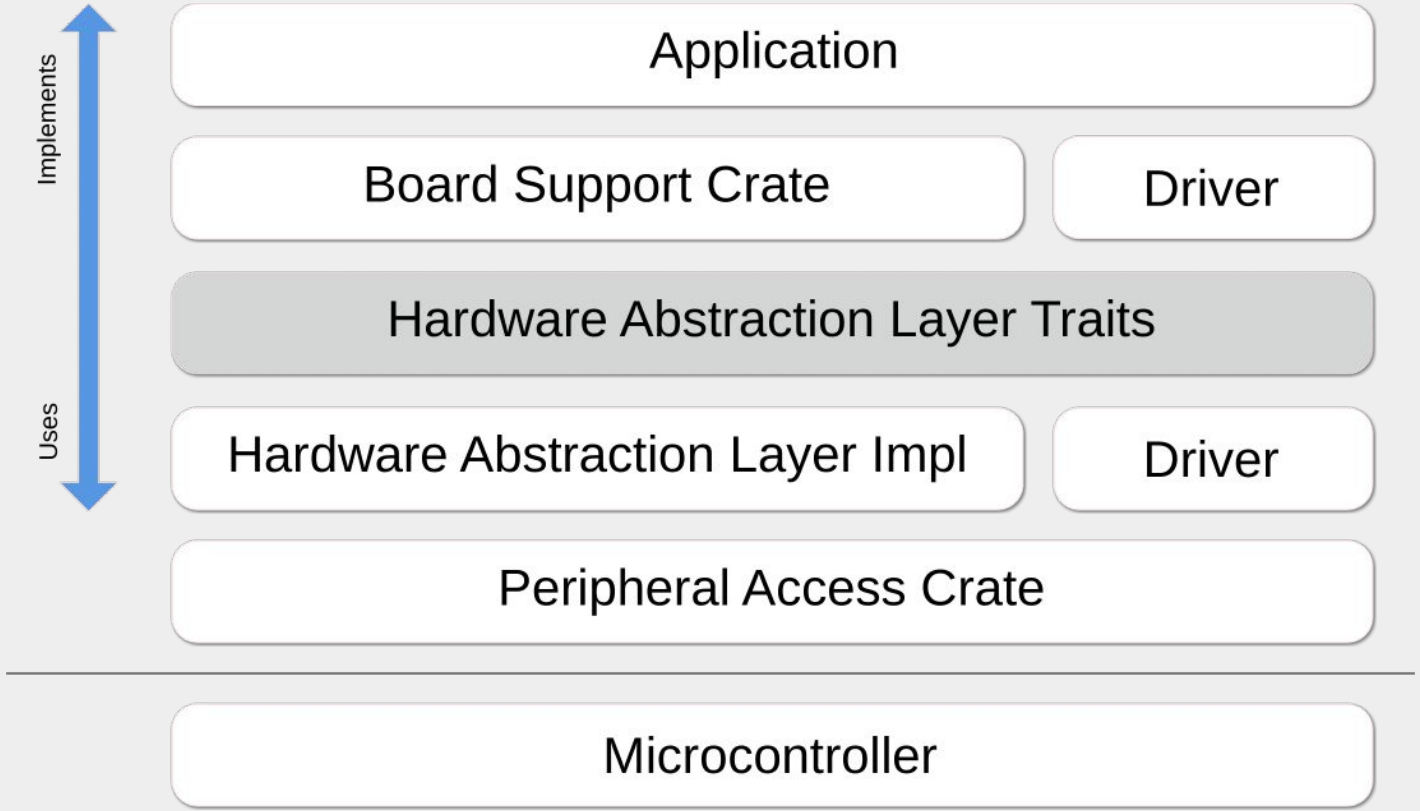
Source: https://docs.rust-embedded.org/book

Ecosystem



Source: https://docs.rust-embedded.org/book

Ecosystem



Source: https://docs.rust-embedded.org/book

Rust Embedded

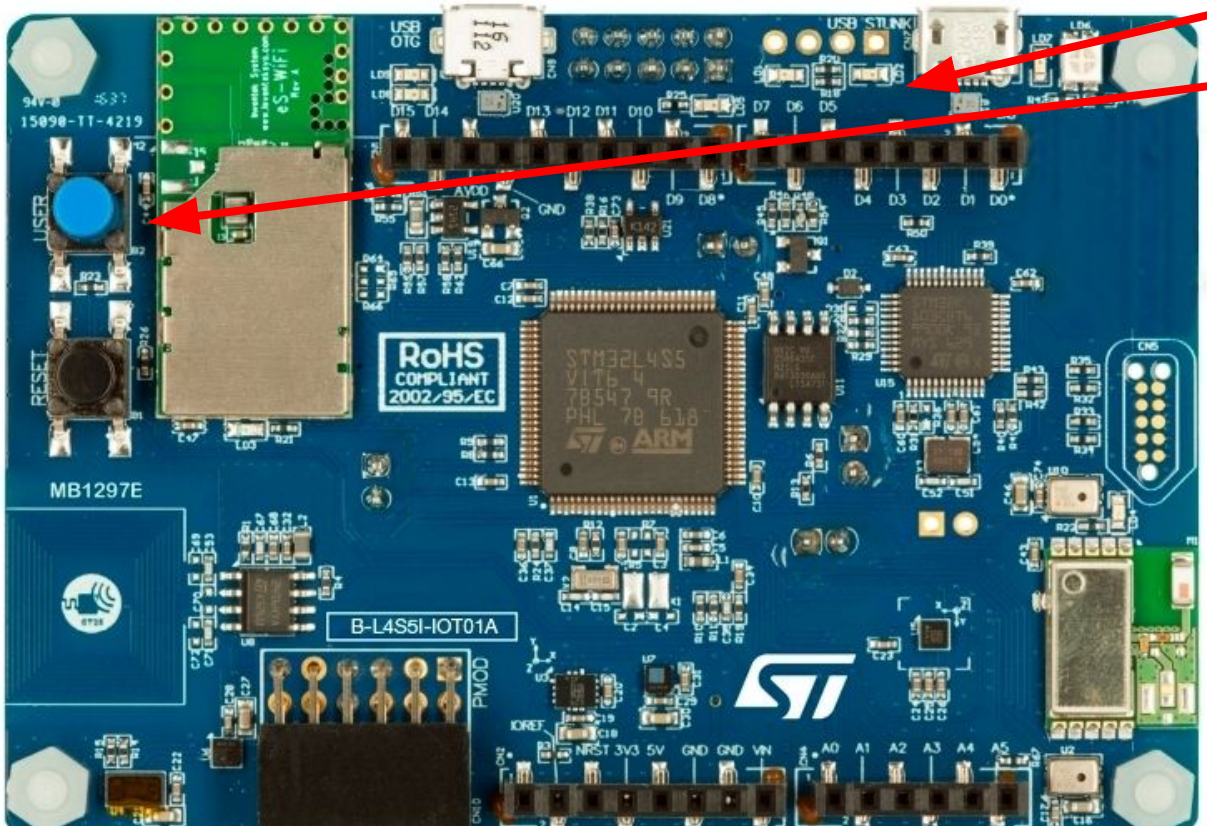
Join the community! https://github.com/rust-embedded

Join the matrix.org chat!

- https://matrix.to/#/#rust-embedded:matrix.org
- https://matrix.to/#/#embassy-rs:matrix.org
- https://matrix.to/#/#drogue-iot:matrix.org
- … and many others

Demo time!

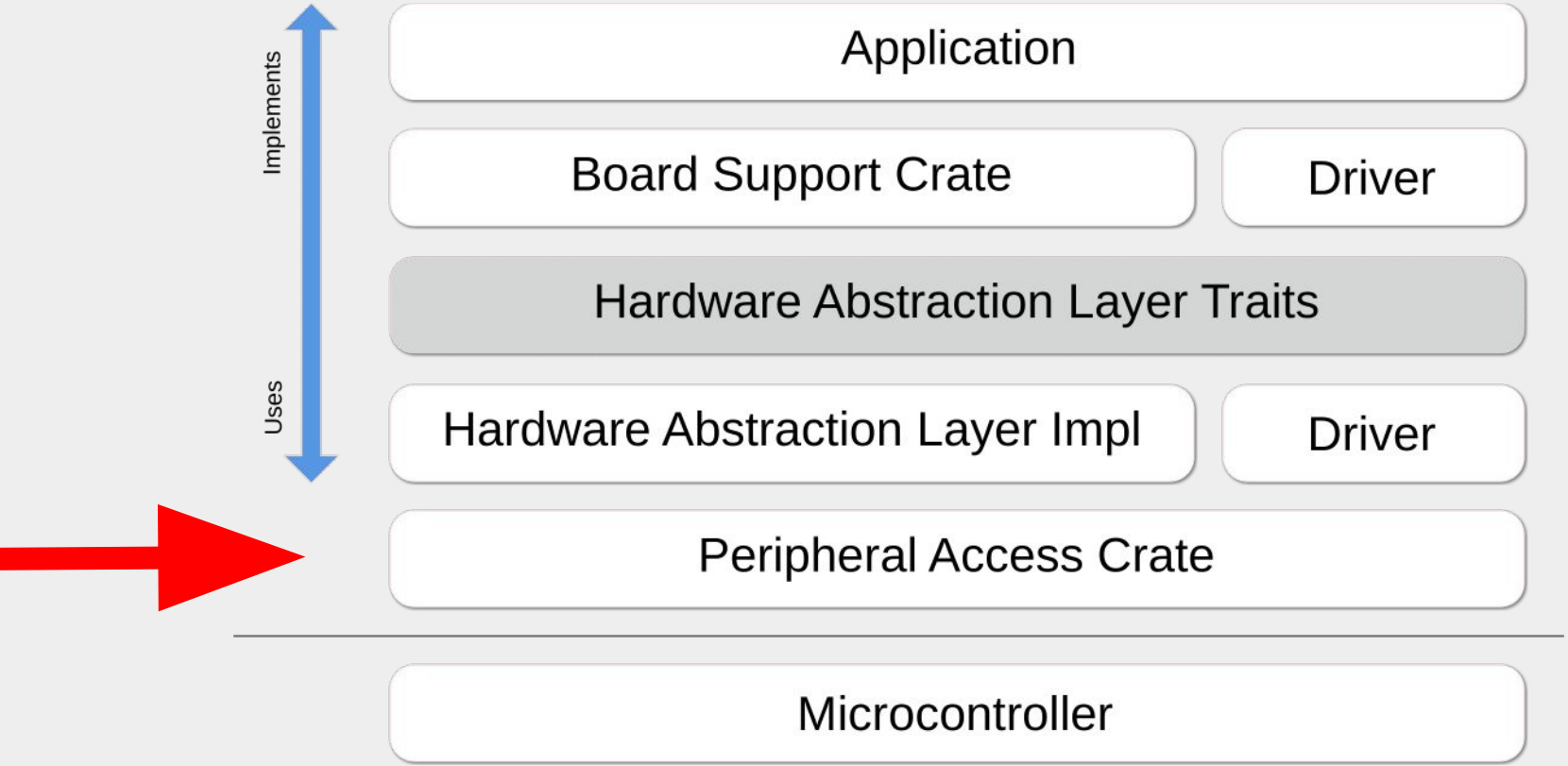
Blink



STM32

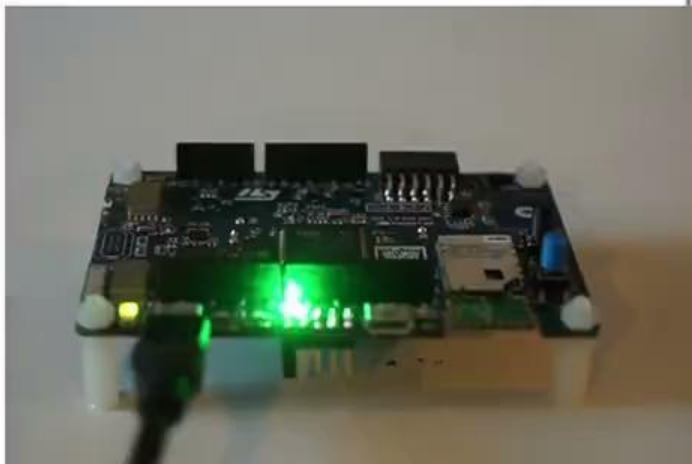
aws qualified

Peripheral Access Crate

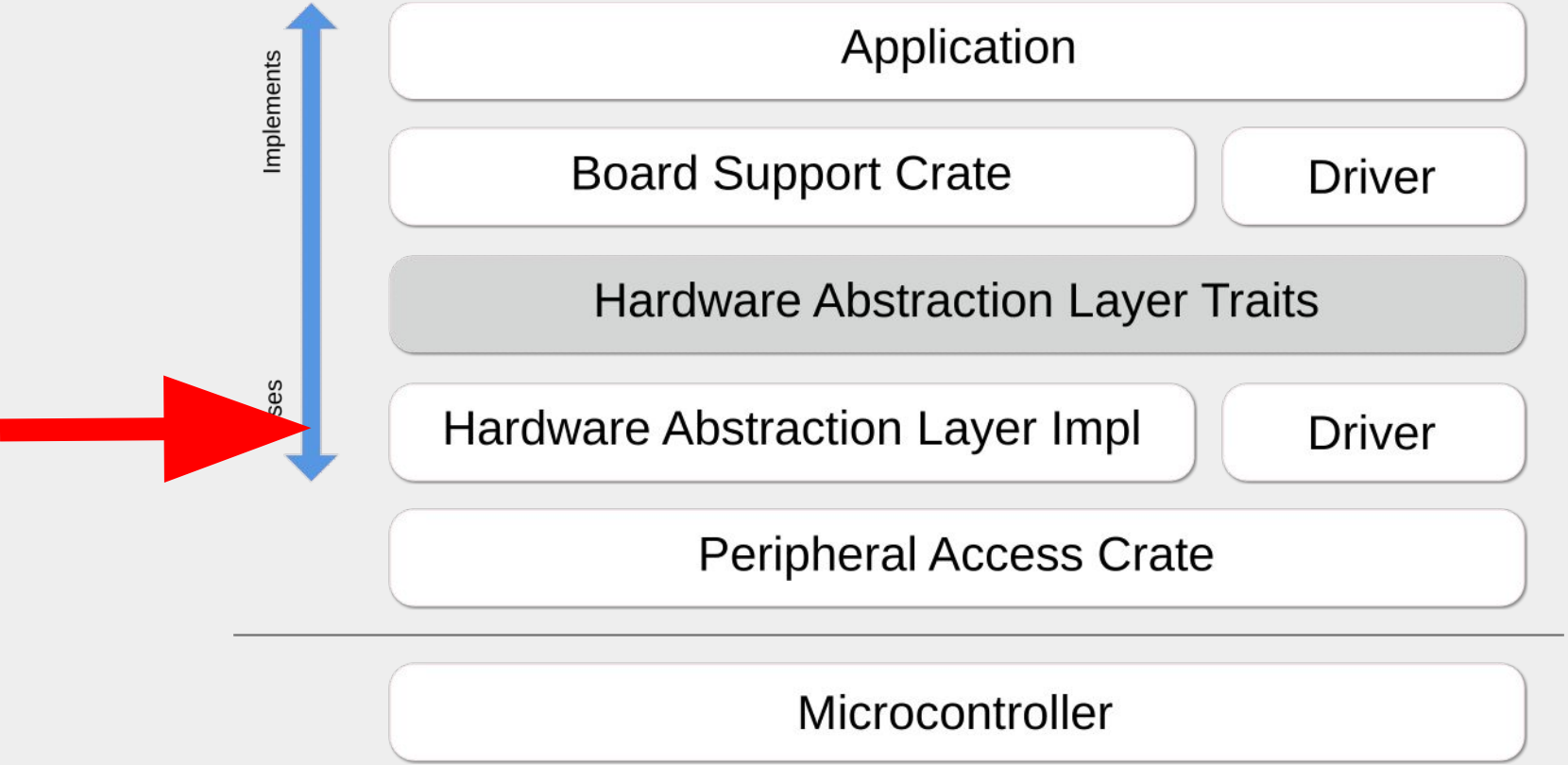


Source: https://docs.rust-embedded.org/book

```
[lulf@pteppic blinky-pac]$ ls
Cargo.lock  Cargo.toml  src
[lulf@pteppic blinky-pac]$ vim Cargo.toml
[lulf@pteppic blinky-pac]$ ls src/
main.rs
[lulf@pteppic blinky-pac]$ nvim src/main.rs
(reverse-i-search)`DEFM': DEFMT_LOG=info cargo run --release
```



Hardware Abstraction Layer



Source: https://docs.rust-embedded.org/book

```
[lulf@pteppic blinky-hal]$ ls
Cargo.lock  Cargo.toml  src
[lulf@pteppic blinky-hal]$ vim Cargo.toml
[lulf@pteppic blinky-hal]$ vim src/main.rs
[lulf@pteppic blinky-hal]$
```

Low power using interrupts

A helping hand

- RTIC: https://rtic.rs/1.0/book/en/
- Tock (RTOS): https://www.tockos.org/
- Hubris (RTOS): https://github.com/oxidecomputer/hubris
- Embassy: https://embassy.dev/

Let's go async!

Approaches to concurrency

Operating System Threads (POSIX)

Green threads

Coroutines

Generators

Rust Futures

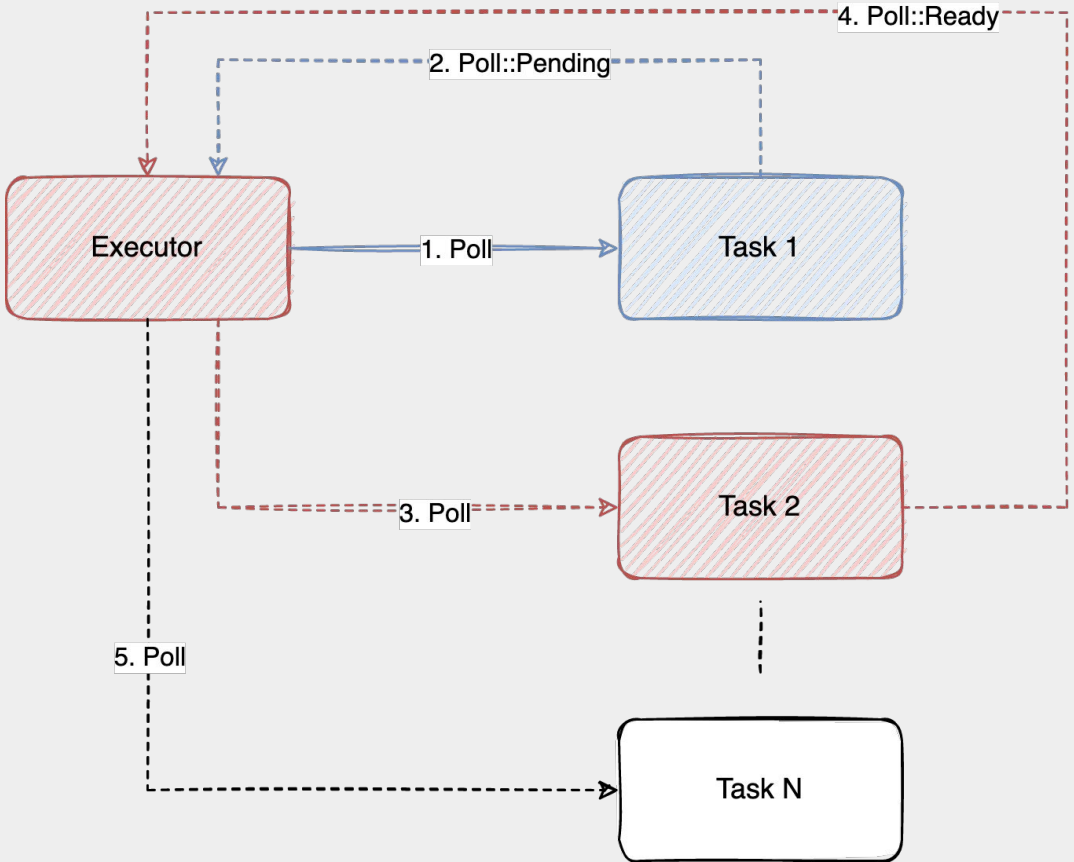
```
pub trait Future {
    type Output;
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output>;
}


pub enum Poll<T> {
    Ready(T),
    Pending,
}
```

Embassy

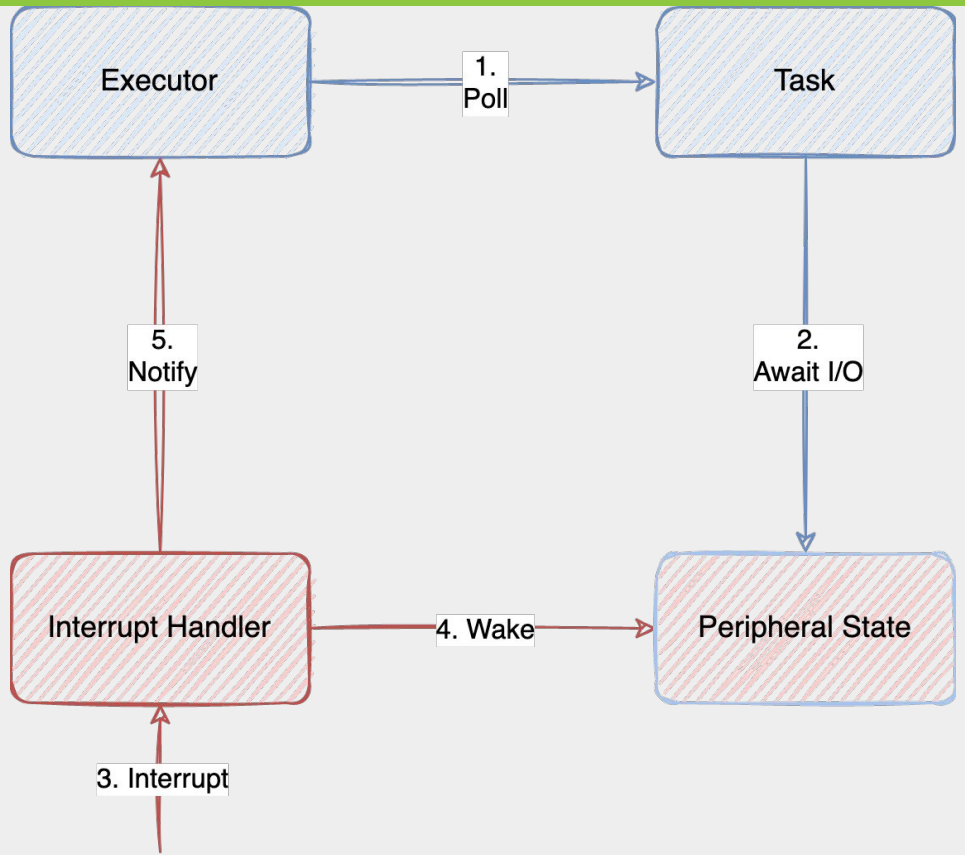
Async Executor

Async HAL

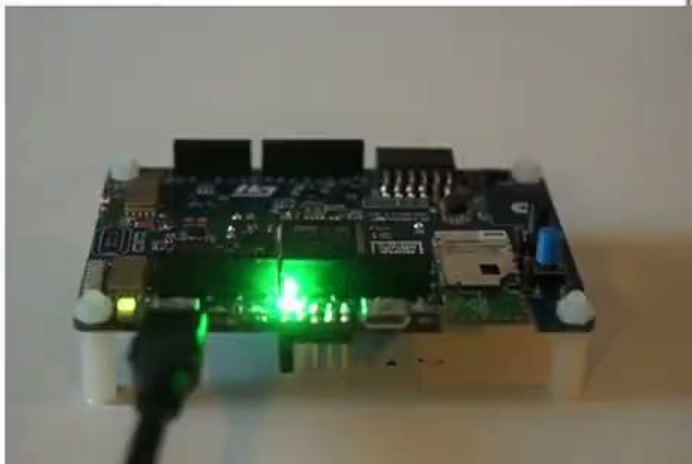
Embassy


4. Poll::Ready
2. Poll::Pending
Executor
1. Poll
Task 1
3. Poll
Task 2
5. Poll
Task N

Interrupts => Wakers



```
[lulf@pteppic blinky-async]$ ls
Cargo.lock  Cargo.toml  src
[lulf@pteppic blinky-async]$ vim Cargo.toml
[lulf@pteppic blinky-async]$ vim src/main.rs
[lulf@pteppic blinky-async]$ DEFMT_LOG=info cargo run --release
```



Performance vs C

Test	C	Rust Embassy	Difference	Difference %
Interrupt time (avg)	2.962us	1.450us	-1.512us	-51.0%
Interrupt time (stddev)	124.8ns	4.96ns	-119.84ns	-96.0%
Thread time (avg)	16.19us	11.64us	-4.55us	-28.1%
Thread time (stddev)	248.2ns	103.0ns	-145.2ns	-56.2%
Interrupt latency (avg)	4.973us	3.738us	-1.235us	-24.8%
Interrupt latency (stddev)	158.0ns	45.3ns	-112.7ns	-71.3%
Program size	20676b	14272b	-6404b	-31.0%
Static memory size	5480b	872b	-4608b	-84.1%

Source: https://tweedegolf.nl/en/blog/65/async-rust-vs-rtos-showdown

Performance vs RTIC

Test	RTIC	Embassy	Difference	Difference %
Interrupt time (avg)	650.8ns	1450ns	799ns	122.8%
Interrupt time (stddev)	10.34ns	4.96ns	-5.38ns	-52.0%
Thread time (avg)	7.807us	11.64us	-3.83us	49.1%
Thread time (stddev)	279.9ns	103.0ns	-176.9ns	-63.2%
Interrupt latency (avg)	1.184us	3.738us	2.554us	215.7%
Interrupt latency (stddev)	77.75ns	45.3ns	-32.45ns	-41.7%
Program size	8888b	14272b	5384b	60.0%
Static memory size	392b	872b	480b	122.4%

Source: https://tweedegolf.nl/en/blog/65/async-rust-vs-rtos-showdown

Hardware Support

Cortex-M (nRF, STM32, RP2040)

RISC-V (ESP32-C3)

WASM

STD

More examples

Priorities

DMA

Radio/LoRaWAN

Downsides

Can be used with stable, *but* traits and impls require:

```
#![feature(type_alias_impl_trait)]

#![feature(generic_associated_types)]
```

Debugging

Learning curve

?Leak

Where to try it out?

https://www.drogue.io/

https://embassy.dev/

Poll::Ready(())

Connectivity

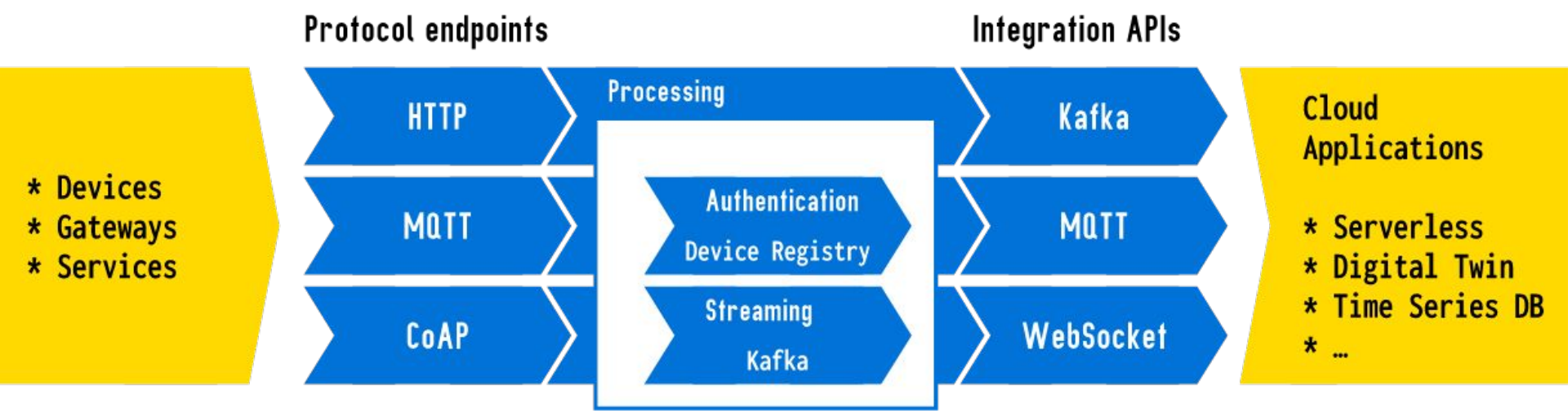
LoRaWAN®

Wi Fi®

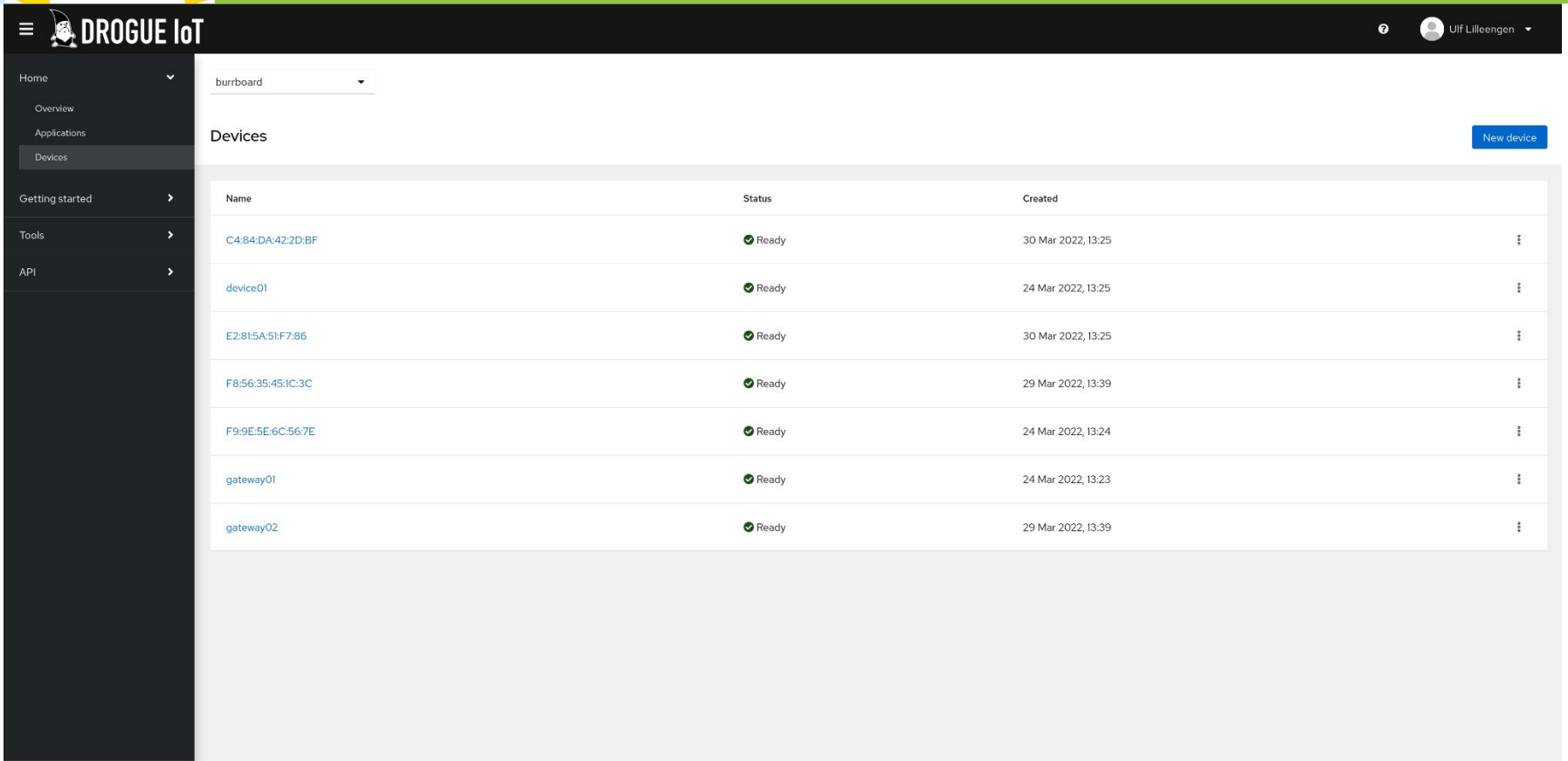
LTE-M

Bluetooth®

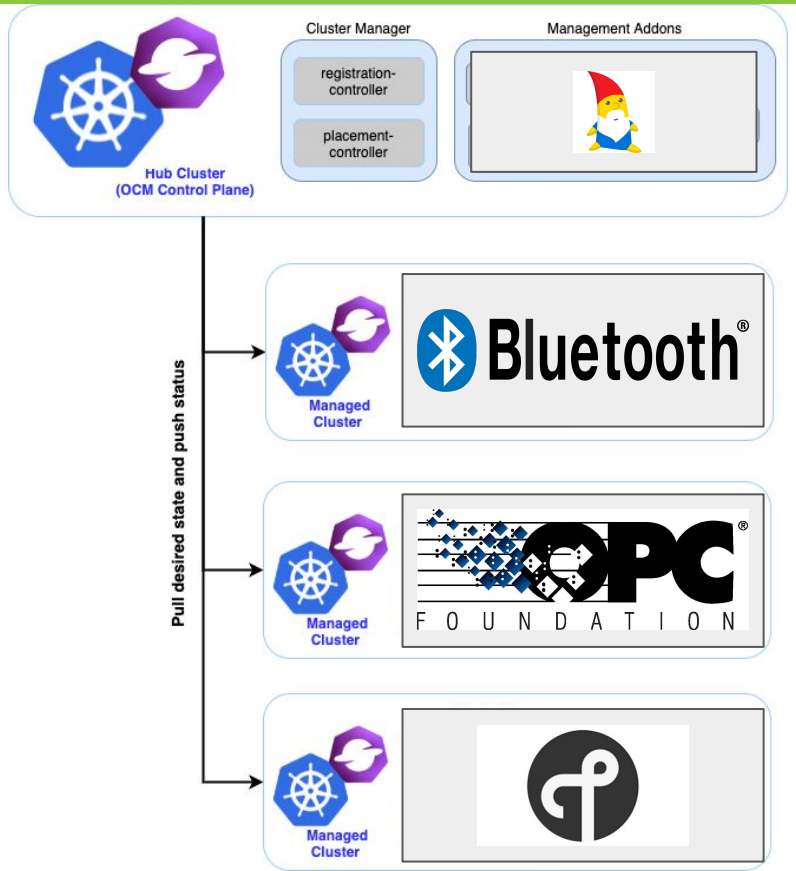
Connectivity and Integration



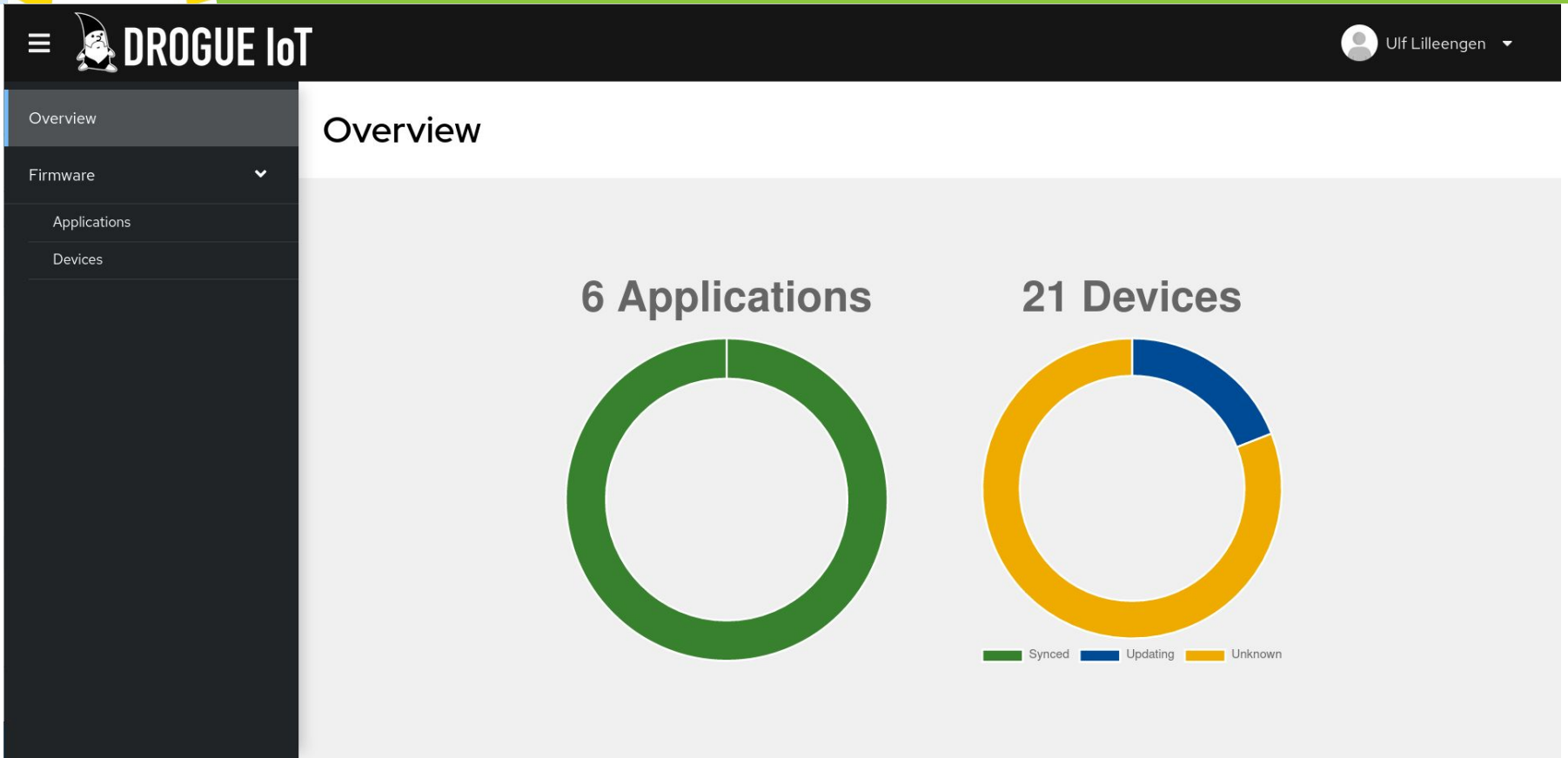
Device Management



Edge Management



Firmware Management

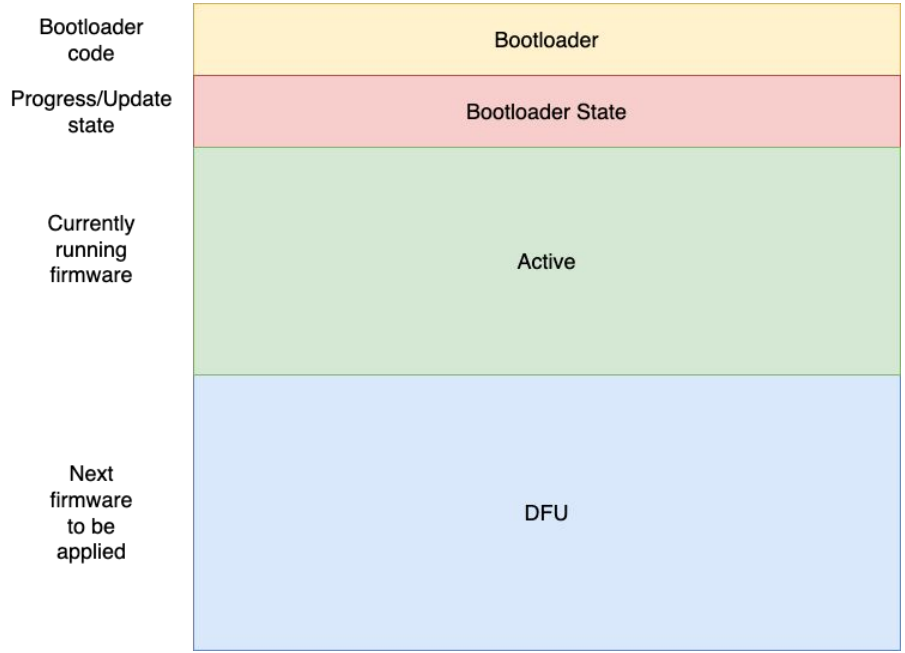


Sensor

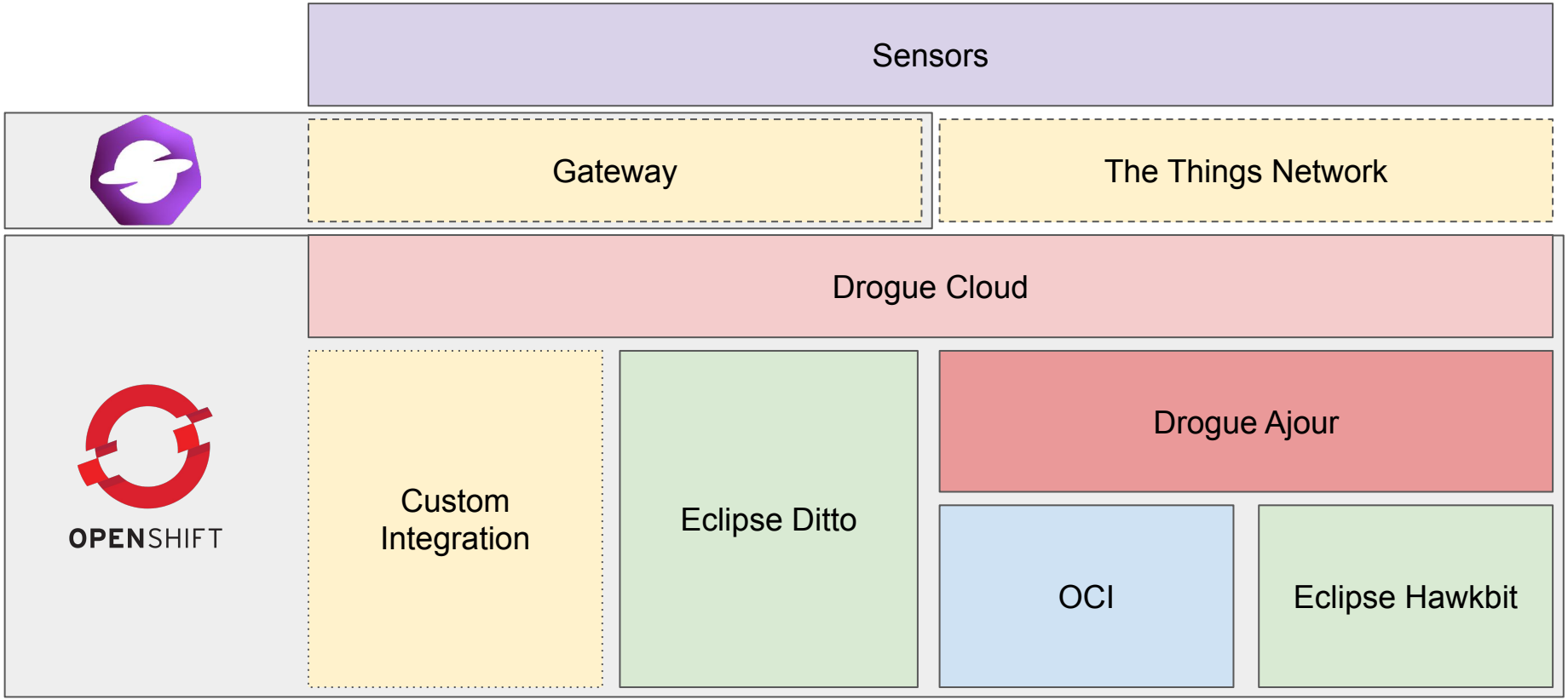
Bootloader

Flash/Storage

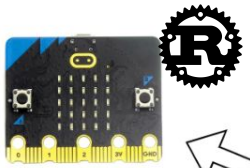
Connectivity

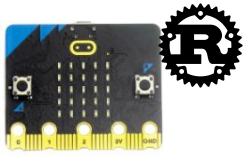


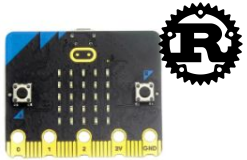
Example architecture

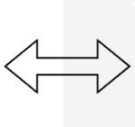


Demo 1: Continuous Delivery





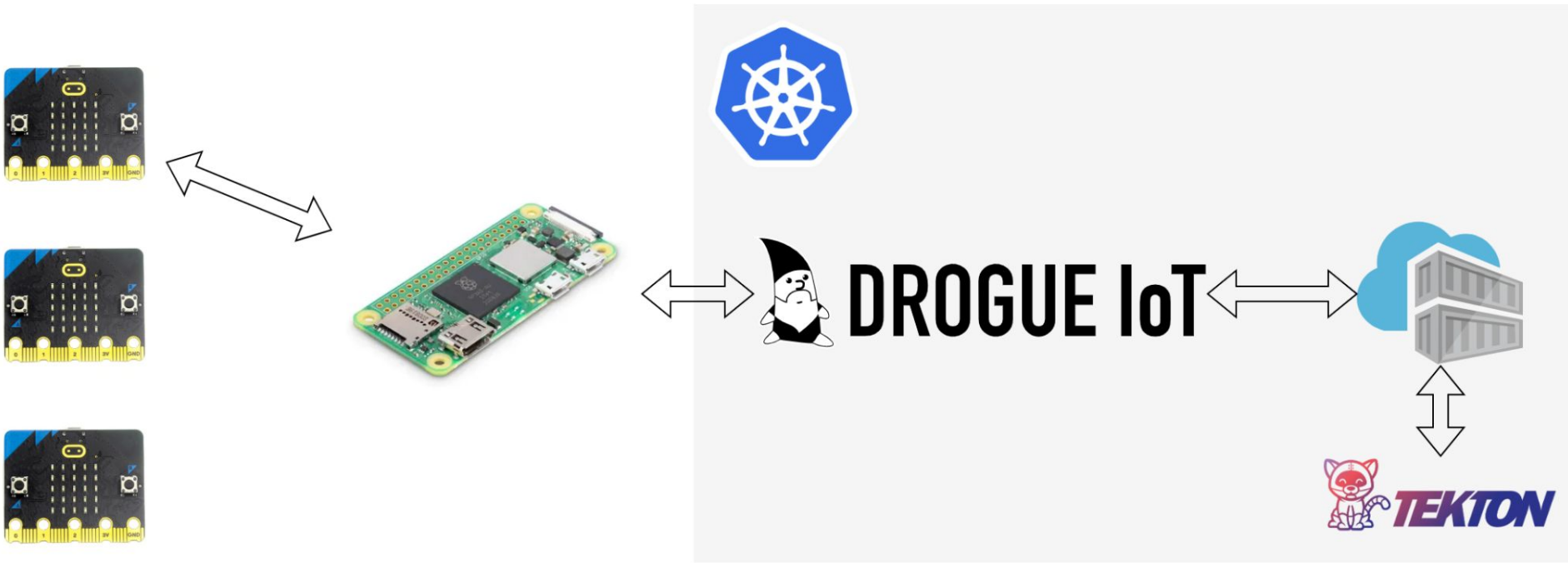




DROGUE IoT

hawkBit

Demo 1: Source 2 Firmware



Try it yourself

Device management:
https://sandbox.drogue.cloud/

Firmware management:
https://firmware.sandbox.drogue.cloud/

Final notes

Resources

Generic Embedded

- https://docs.rust-embedded.org/book/intro/index.html

RTIC

- https://rtic.rs/1.0/book/en/

Embassy (Async)

- https://github.com/embassy-rs/embassy
- https://embassy.dev

Drogue IoT

- https://www.drogue.io/
- https://sandbox.drogue.cloud/