RUST GLOBAL × 

RustChinaConf 2025&
Rust Global China

# communication at *any* size

James Munns
OneVariable GmbH
2025-09-13

i'm james!

you might know me from:

# Rust Embedded WG Leadership Council OneVariable GmbH

# this a talk about *ergot*: a messaging library

but also about how I think about embedded systems

# ergot is new and experimental

but I am very excited about it

# I like helping computers talk to each other

but I have a big problem

# I don't like how embedded systems communicate today

# I don't like how they communicate internally

unsafe everything
channel spaghetti
everything in mutexes

# I don't like how they communicate to external devices

seems to go one of two ways:

# lowest common denominator patterns

# i2c/spi registers
# modbus style
# at commands
# serial console

# "maximal" patterns

tcp/ip, eth, wifi

"consortium standards"

mqtt, rest, json

# binary blobs and/or huge vendor libs

I think other people have figured out communication

in many cases, *decades* ago

# 80's - 90's

# appletalk

80's - current

erlang

# 10's - current zeromq

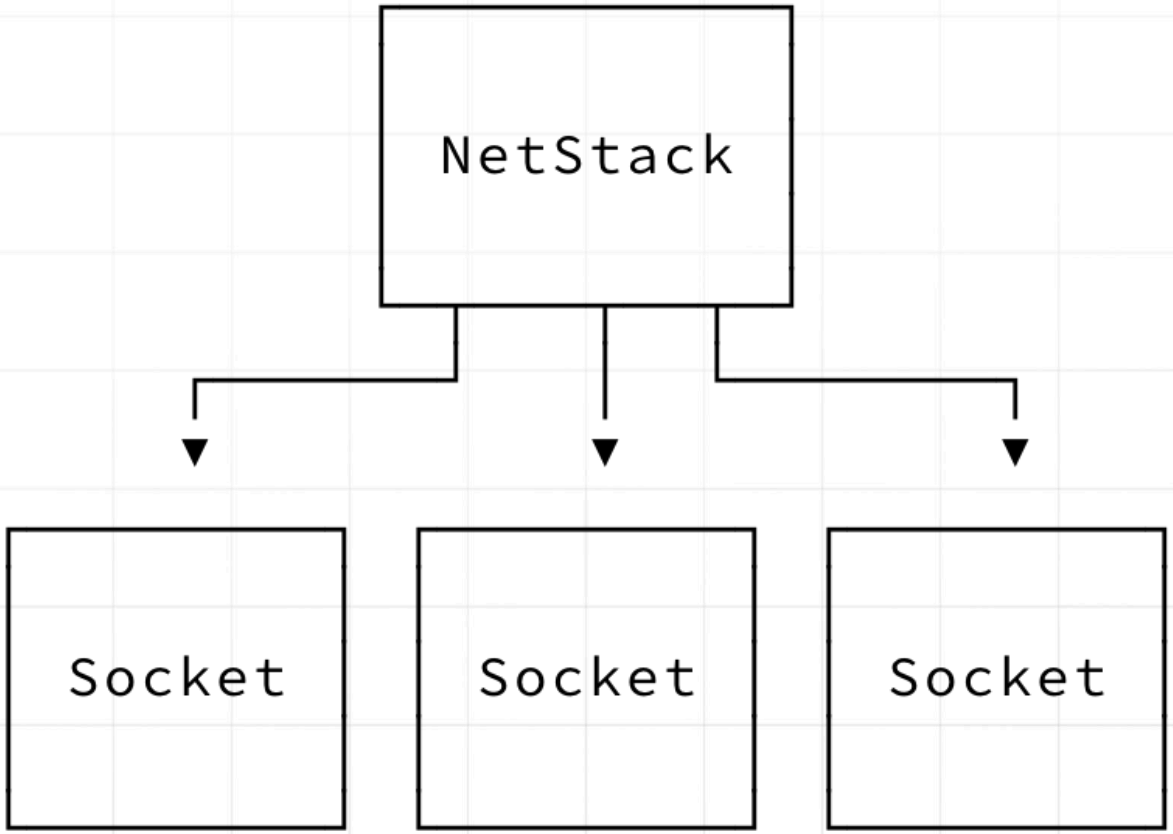
# 00's - current backend systems

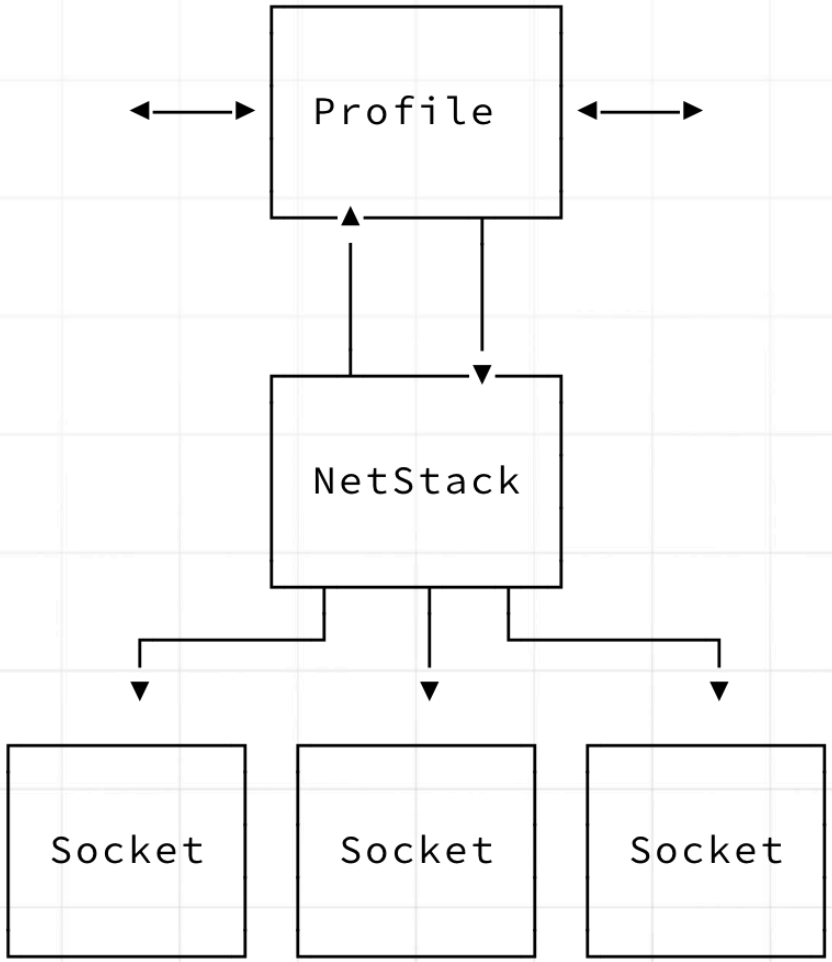
# I like (parts of) how those systems communicate

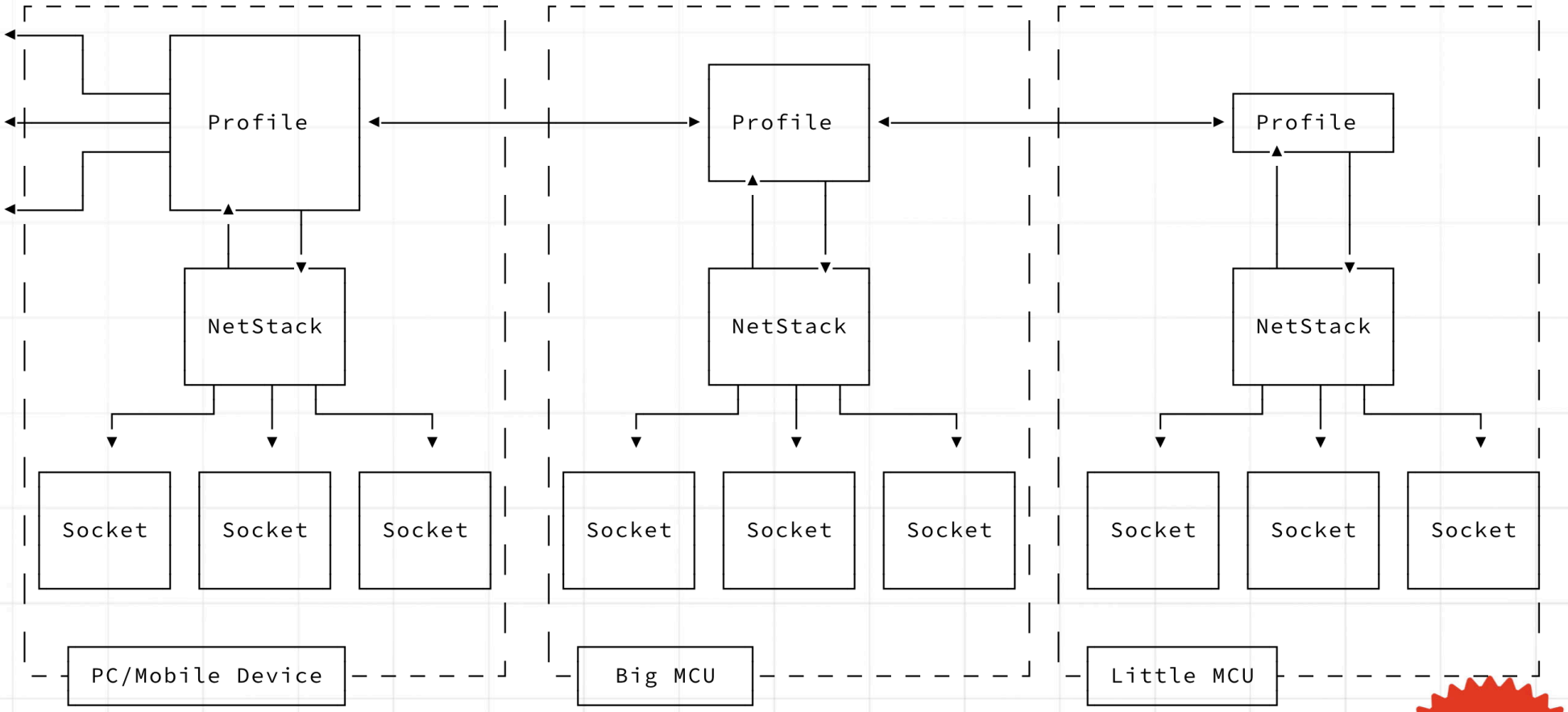
# I think we should borrow good ideas

# ergot:
# a messaging library for rust

# NetStack


NetStack
Socket
Socket
Socket


Profile
NetStack
Socket
Socket
Socket


Profile
NetStack
Socket
Socket
Socket
PC/Mobile Device
Profile
NetStack
Socket
Socket
Socket
Big MCU
Profile
NetStack
Socket
Socket
Socket
Little MCU

# ergot:
# i've already started borrowing!

# sockets

(tcp/ip)

```rust
let recv = STACK
    .topics()
    .bounded_receiver::<ButtonPressedTopic, 4>(None);
let recv = pin!(recv);
let mut recv = recv.subscribe();

loop {
    let msg = recv.recv().await;
    defmt::info!(
      "Listener #{=u8}, button {=u8} pressed",
      idx, msg.t
    );
}
```



# type safe comms

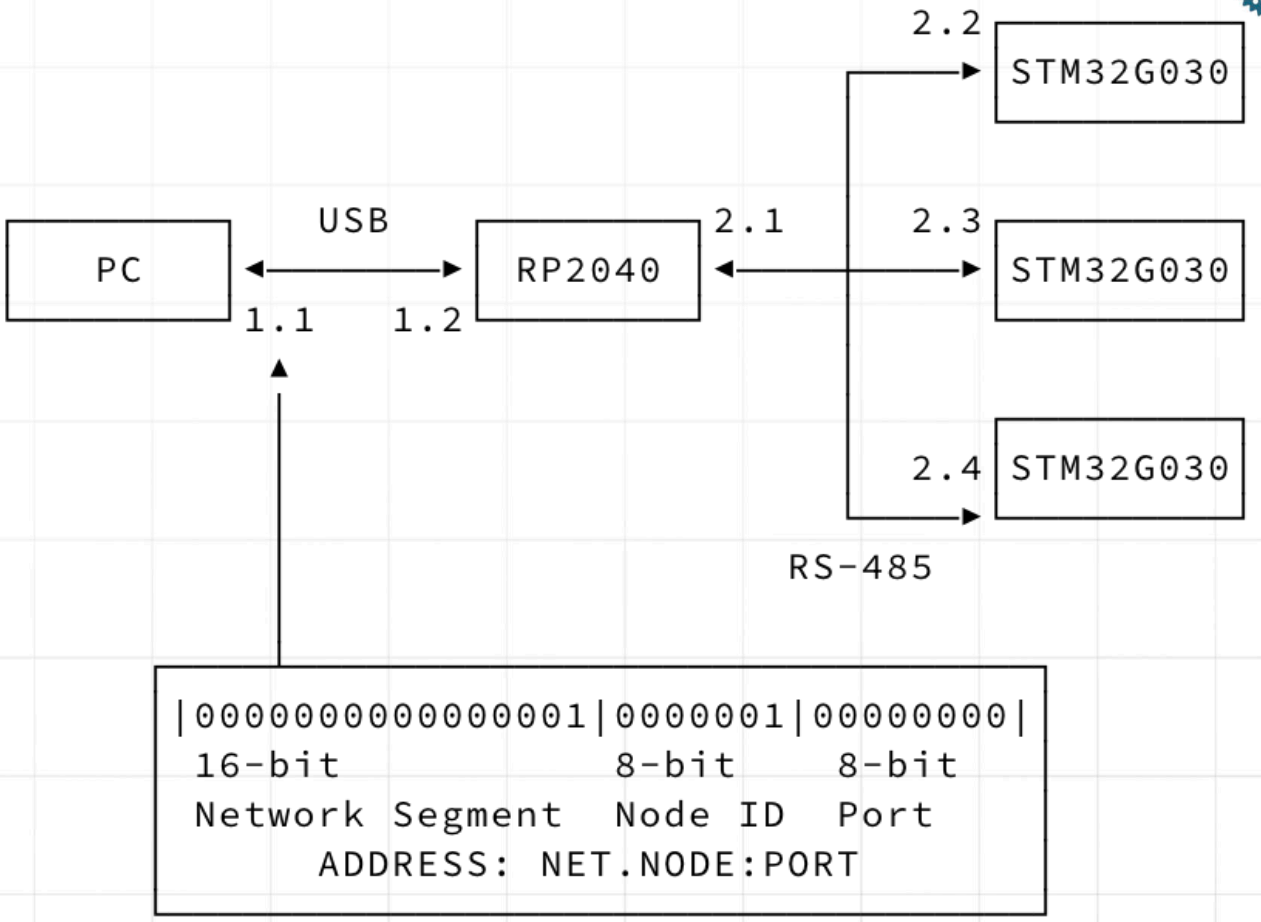
## (postcard-rpc)

```rust
let socket = STACK.endpoints()
    .bounded_server::<PwmSetEndpoint, 2>(Some(name));
let socket = pin!(socket);
let mut hdl = socket.attach();
loop {
    let _ = hdl.serve_blocking(|data: &f32| -> u64 {
        let val = data.clamp(0.0, 1.0);
        let val = val * const { u16::MAX as f32 };
        let val = val as u16;
        pwm.set_duty_cycle(val);
        Instant::now().as_ticks()
    })
    .await;
}
```



# addressing, routing, interfaces

(appletalk)


2.2
STM32G030
USB
PC
RP2040
2.1
2.3
STM32G030
1.1
1.2
2.4
STM32G030
RS-485
|0000000000000001|0000001|00000000|
16-bit
8-bit
8-bit
Network Segment
Node ID
Port
ADDRESS: NET.NODE:PORT

# message passing

(erlang)

```rust
loop {
    let res = embassy_futures::select::select(
        assign_svr.recv_manual(),
        refresh_svr.recv_manual(),
    )
    .await;
    match res {
        Either::First(assign_req) => {
            let Ok(assign_req) = assign_req else { continue };
            handle_assign(&nsh, refresh_port, &assign_req)
        }
        Either::Second(refresh_req) => {
            let Ok(refresh_req) = refresh_req else { continue };
            handle_refresh(&nsh, &refresh_req)
        }
    }
}
```



# observability + discoverability

(backend systems)

```
// Desktop style, with std/alloc
let stack: EdgeStack = new_target_stack(&queue, 1024);
let logger = LogSink::new_boxed(stack.clone());
logger.register_static(log::LevelFilter::Info);
```

```rust
// embedded style, no-std
static STACK: Stack = kit::new_target_stack(
    OUTQ.stream_producer(),
    MAX_PACKET_SIZE,
);
static LOGSINK: LogSink<&'static Stack> = LogSink::new(&STACK);

#[esp_hal_embassy::main]
async fn main(spawner: Spawner) {
    // ...
    LOGSINK.register_static(log::LevelFilter::Info);
    // ...
}
```



```rust
let query = SocketQuery {
    key: ErgotSeedRouterAssignmentEndpoint::REQ_KEY.to_bytes(),
    nash_req: NameRequirement::Any,
    frame_kind: FrameKind::ENDPOINT_REQ,
    broadcast: false,
};
let discovered = stack
    .discovery()
    .discover_sockets(4, Duration::from_secs(1), &query)
    .await;
for found in discovered {
    // ...
}
```

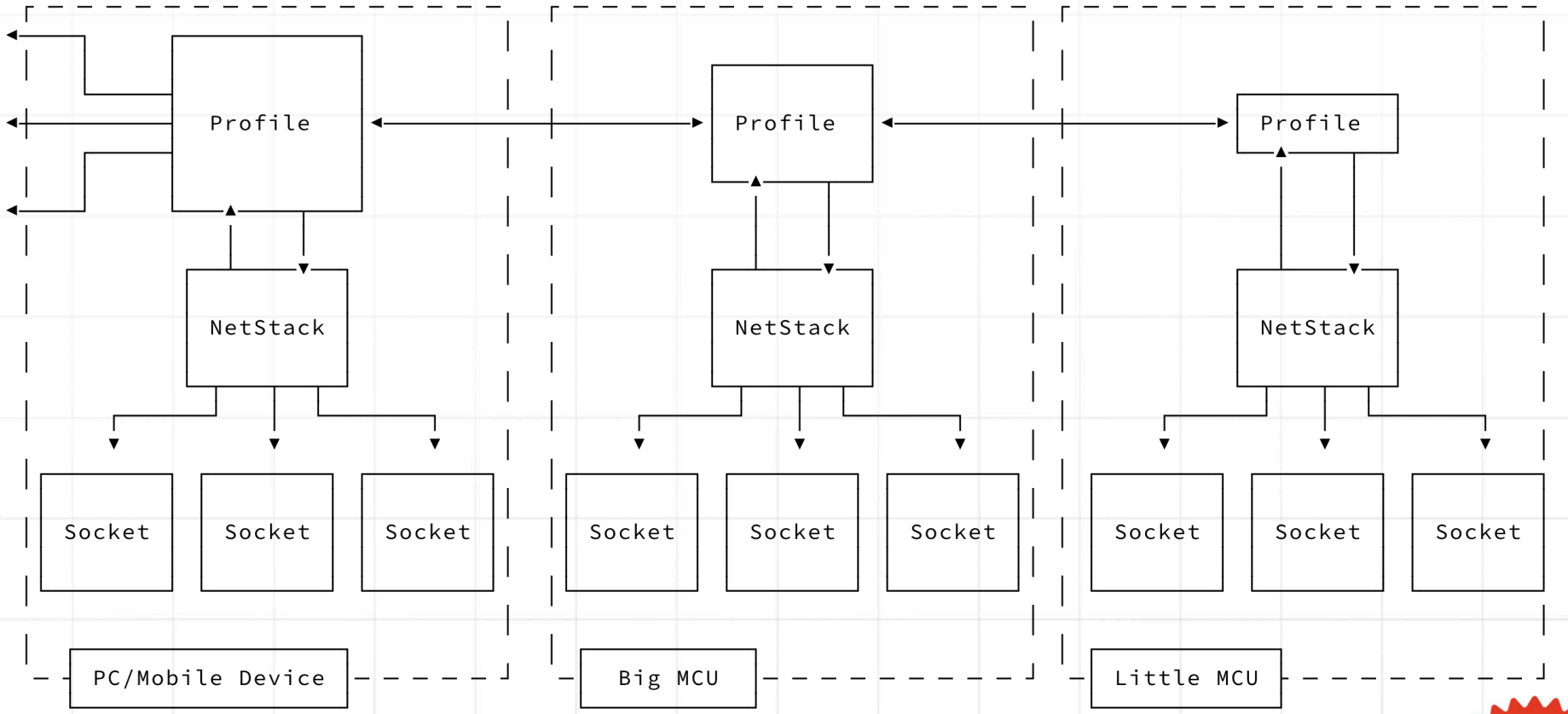


# why now? how?

# rust had the benefit of starting 25+ years later

# how do people build connected systems *today* in rust?

# I want to turn a circuit board into a LAN


Profile
NetStack
Socket
Socket
Socket
PC/Mobile Device
Profile
NetStack
Socket
Socket
Socket
Big MCU
Profile
NetStack
Socket
Socket
Socket
Little MCU

# if this seems exciting:

# ergot needs your help!

ask questions
try out the demos
give feedback
contribute

ergot is open to good ideas, new or old.

https://github.com/jamesmunns/ergot

code, examples,
links to docs + chat

RUST GLOBAL × 

RustChinaConf 2025&
Rust Global China

# communication at *any* size

James Munns
OneVariable GmbH
2025-09-13

# bonus slides!

scheduling?

❌ OS or RTOS

✅ async/await

# message/protocol definitions?

- ✗ codegen tools
- ✗ preprocessor macros
- ✗ pdf specs

- ✅ traits + generics
- ✅ serde types
- ✅ published crates

# portability?

❌ bespoke language

❌ opaque runtime lib

✅ normal no_std crate

# rust makes this possible!

- ✅ local comms
- ✅ external comms
- ✅ small networks

- ✅ tiny footprint
- ✅ pleasant to use
- ✅ flexible
- ✅ *useful*