type erasure
in rust
in ergot

ergot:
a messaging library

for std and no_std alike

ergot:
type safe sockets

more like
channels
than
tcp/ip sockets

```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;
}
```

ergot uses:

serde:
for ser/de

postcard:
as the wire format

postcard-schema: to generate "type keys"

ergot lets you send/recv locally

ergot lets you
send/recv remotely

we should always be efficient as possible!

our friend:
the *socket*

every socket
can be a different
type

every socket
can have a different
storage

sockets are

pinned

(in a task)

ergot’s NetStack has an intrusive linked list of all sockets

sockets have a common header:

linked list pointers

socket metadata (port #, packet kind, discoverable, etc.)

socket metadata:
used for routing!

a *vtable*

vtable:
manual dyn Trait

lets pretend we have

```
trait Socket {
    /* ... */
}
```

what are our trait methods?

```rust
/// Receive an *owned* packet
/// from local code
fn recv_owned<T: 'static + Clone>(
    self: Pin<&mut Self>,
    data: &T,
    hdr: HeaderSeq,
    ty_id: &TypeId,
) -> Result<(), SocketSendError>;
```

```rust
/// Receive a protocol error from
/// remote entity/local interface
fn recv_err(
    self: Pin<&mut self>,
    hdr: HeaderSeq,
    err: ProtocolErr,
);
```

```
/// Receive a raw frame from
/// remote entity/local interface
fn recv_raw<T: Deserialize>(
    self: Pin<&mut self>,
    data: &[u8],
    hdr: HeaderSeq,
) -> Result<(), SocketSendError>;
```

```rust
/// Receive a NON-Owned message
/// from local code
fn recv_borrowed<T: Serialize>(
    self: Pin<&mut self>,
    data: &T,
    hdr: HeaderSeq,
    ser_fn: /* ignore me for now */
) -> Result<(), SocketSendError>;
```

some sockets store Ts

some sockets store [u8]s

Owned Socket:
Stores as T

- Owned (T -> T)
- Raw (&[u8] -> T)
- Error (E -> E)
- Borrowed (N/A)

Borrowed Socket:
Stores as [u8]

- Owned (T -> [u8])
- Raw (&[u8] -> [u8])
- Error (E -> [u8])
- Borrowed (&T -> [u8])

okay but I lied

(a lot)

you can’t really use traits like this

storing dissimilar types in one collection is hard generic methods aren’t dyn-compatible

what does it really look like?

the lie:

```rust
/// Receive an *owned* packet
/// from local code
fn recv_owned<T: 'static + Clone>(
    self: Pin<&mut Self>,
    data: &T,
    hdr: HeaderSeq,
    ty_id: &TypeId,
) -> Result<(), SocketSendError>;
```

the truth:

```rust
impl Socket<T: /* ... */> {
    fn recv_owned(
        this: NonNull<()>,
        that: NonNull<()>,
        hdr: HeaderSeq,
        ty: &TypeId,
    ) -> Result<(), SocketSendError> {
        // ...
    }
}
```

the *socket* knows what it is, and what type it wants!

```
// fn recv_owned(...) -> ... {
  let this: &mut Self = unsafe {
    &mut *this.as_ptr().cast()
  };
  let that: &T = unsafe {
    &*that.as_ptr().cast()
  };
  // ...
// }
```

fun trick:
turning generic fns into “normal” fn pointers

```
// monomorphization on demand!
let func: fn(/* ... */) -> /* ... */
  = Socket::<YourType>::recv_owned;
```

```rust
pub struct SocketVTable {
    recv_owned: Option<RecvOwned>,
    recv_bor: Option<RecvBorrowed>,
    recv_raw: RecvRaw,
    recv_err: Option<RecvError>,
}
```

```
impl Socket<T: /* ... */> {
  const fn vtable() -> SocketVtable {
    SocketVtable {
      recv_owned: Some(Self::recv_owned),
      recv_bor: None,
      recv_raw: Self::recv_raw,
      recv_err: Some(Self::recv_err),
    }
  }
}
```

```
const YTVtable: SocketVtable
    = Socket::<YourType>::vtable();
```

```
pub type RecvOwned = fn(
  NonNull<()>, // self: Pin<&mut Self>
  NonNull<()>, // data: &T
  HeaderSeq,   // hdr
  &TypeId,     // type_id
) -> Result<(), SocketSendError>;
```

```
pub type RecvError = fn(
  NonNull<()>,   // self: Pin<&mut Self>
  HeaderSeq,     // hdr
  ProtocolError, // err
);
```

```
pub type RecvRaw = fn(
  NonNull<()>, // self: Pin<&mut Self>
  &[u8],       // data
  HeaderSeq,   // hdr
) -> Result<(), SocketSendError>;
```

```rust
pub type RecvBorrowed = fn(
  NonNull<()>, // self: Pin<&mut Self>
  NonNull<()>, // data: &T
  HeaderSeq,   // hdr
  BorSerFn,
) -> Result<(), SocketSendError>;
```

```
pub type BorSerFn = fn(
  NonNull<()>, // data: &T
  HeaderSeq,   // hdr
  &mut [u8],   // ser dest
) -> Result<usize, SocketSendError>;
```

rust is a strongly typed language

but through unsafe
anything is
possible :)