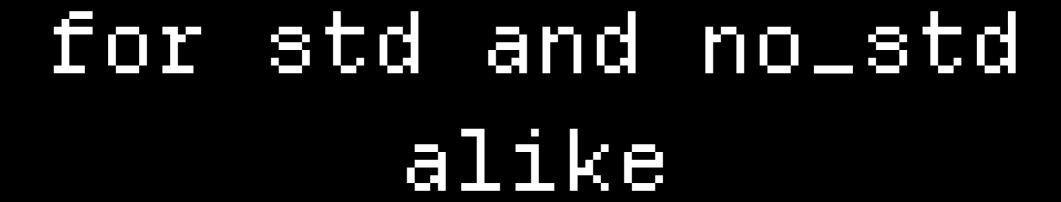
type erasure in rust in ergot

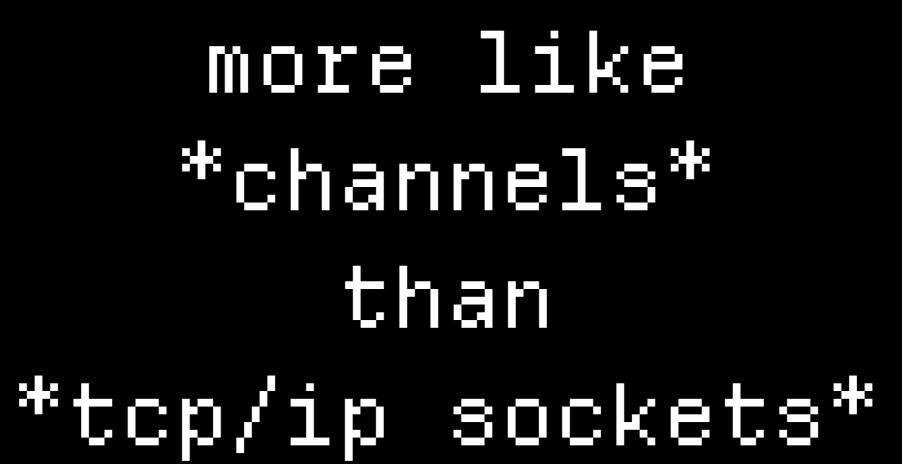


<u>a messaging library</u>



ergot:

type safe sockets

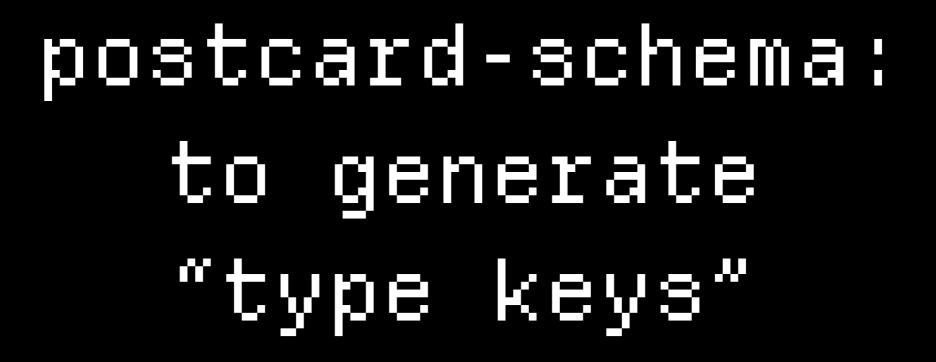


```
let socket = STACK.endpoints()
    .bounded_server::<PwmSetEndpoint, 2>(Some(name));
let socket = pin!(socket);
let mut hdl = socket.attach();
loop (
    let _ = hdl.serve_blocking(|data: &f32| -> u64 {
        let val = data.clamp(0.0, 1.0);
        let val = val * const { u16::MAX as f32 };
  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

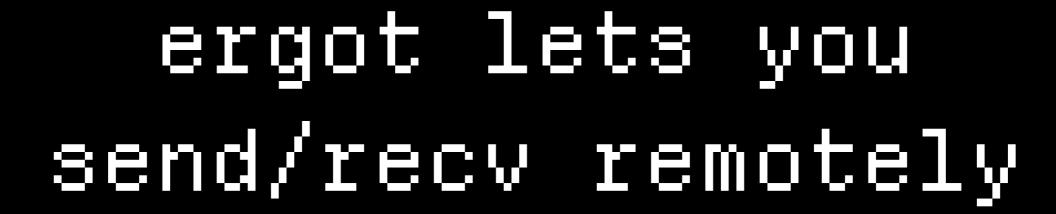


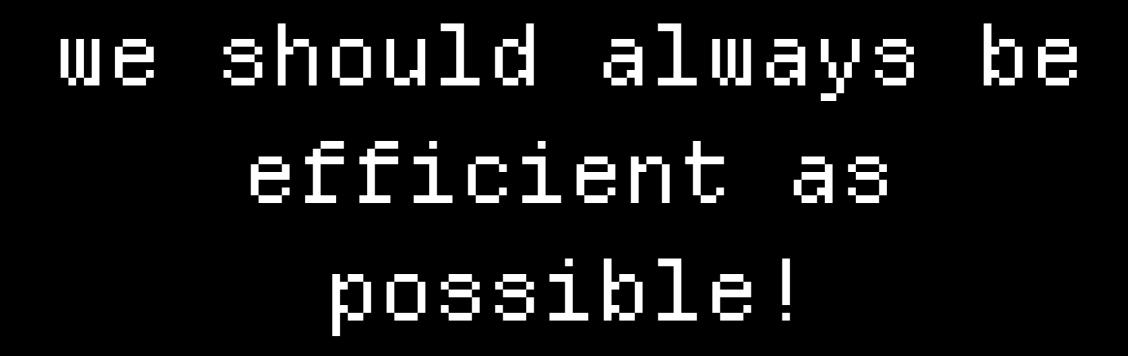


postcard: as the wire format

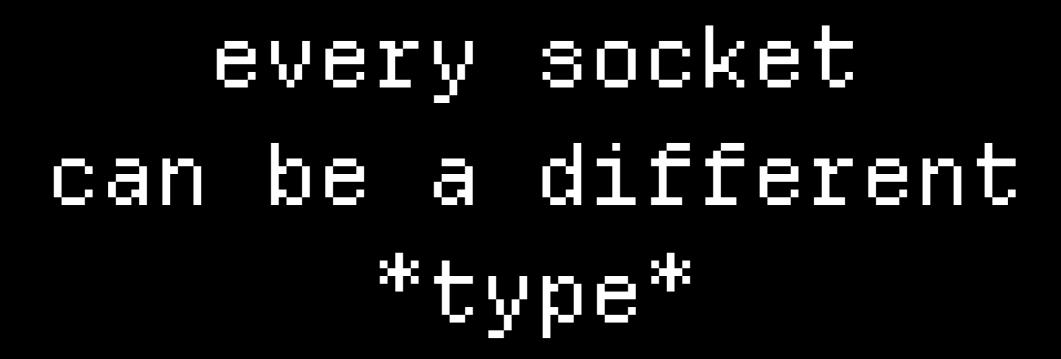


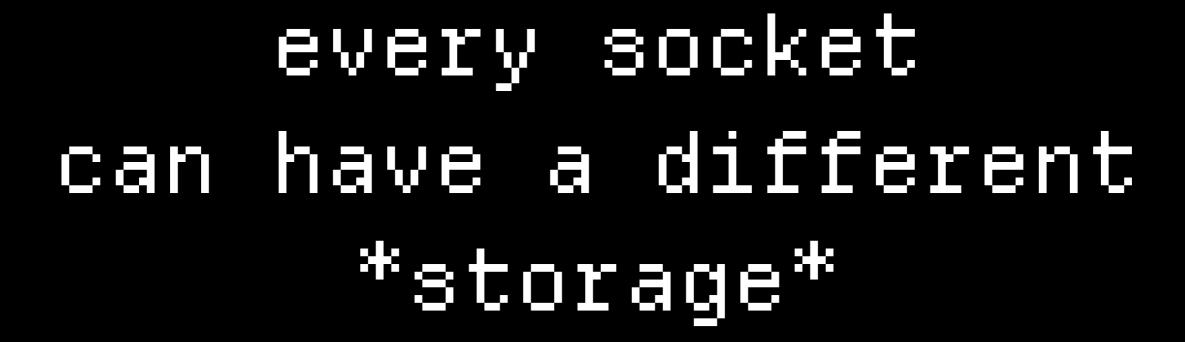
ergot lets you send/recv locally

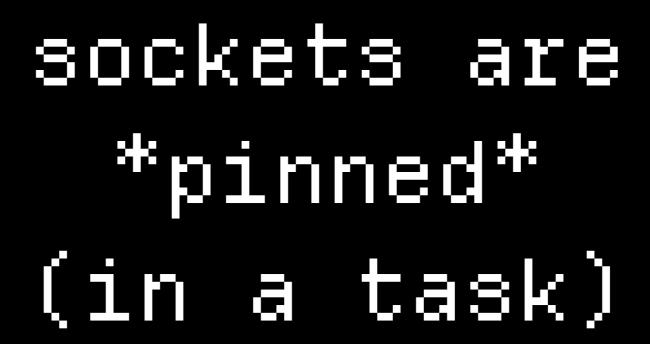


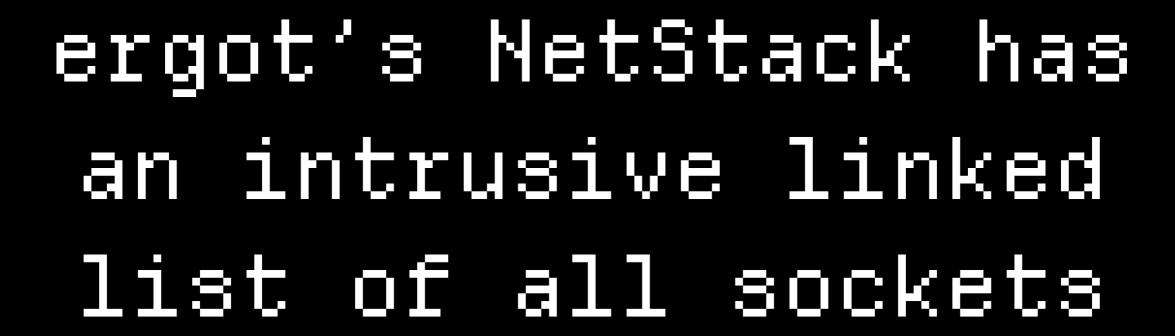






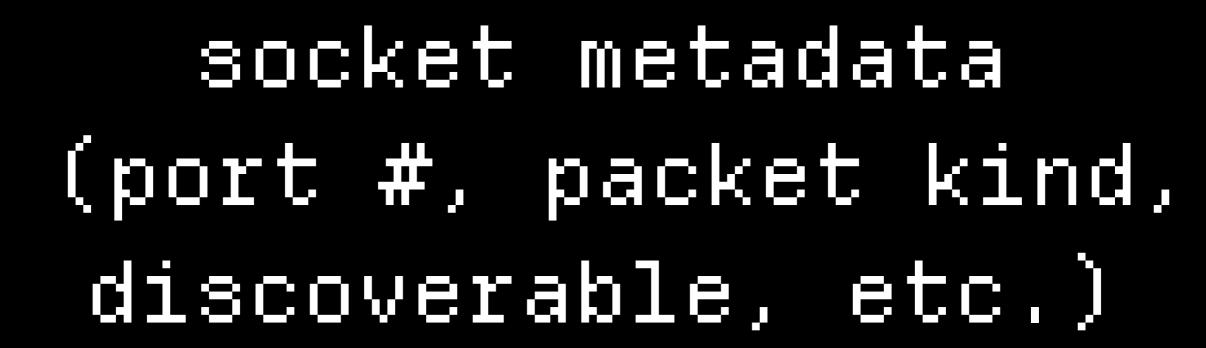


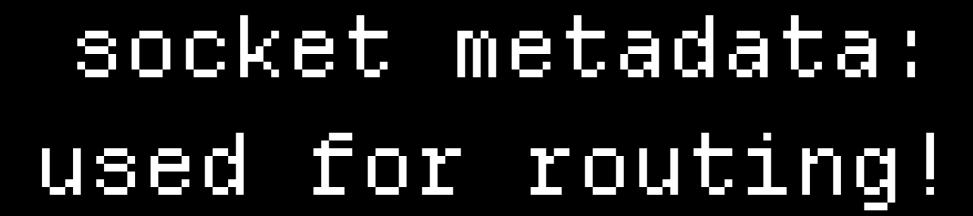












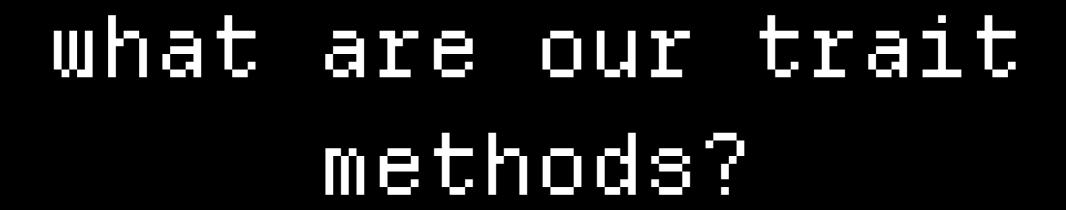


vtable: manual dyn Trait



lets pretend we have

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

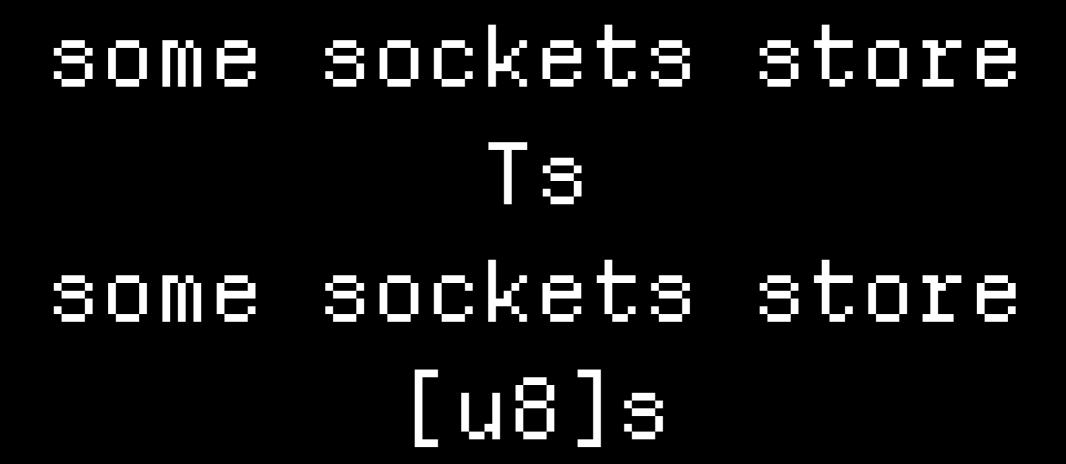


```
/// Receive an *owned* packet
/// from local code
fn recv_owned<T: 'static + Clone>(
    self: Pin<&mut Self>,
    data: 🏖T,
    hdr: HeaderSeq,
    ty_id: &TypeId,
) -> Result<(), SocketSendError>;
```

```
/// 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: 8[u8],
    hdr: HeaderSeq,
) -> Result<(), SocketSendError>;
```

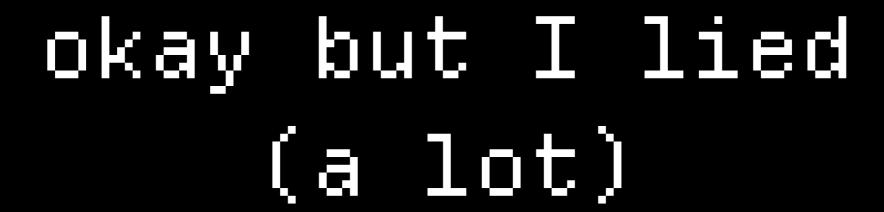
```
/// Receive a NON-Owned message
/// from local code
fn recv_borrowed<T: Serialize>(
    self: Pin<&mut self>,
    data: 🎖T,
    hdr: HeaderSeq,
    ser_fn: /* iqnore me for now */
) -> Result<(), SocketSendError>;
```

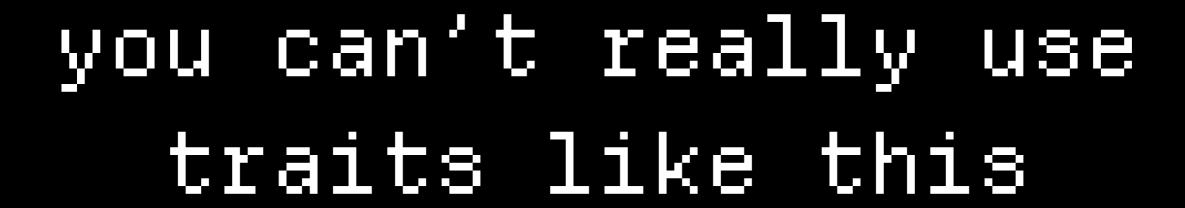


- Owned Socket: Stores as T
- Owned (T -> T)
- Raw (&[u8] -> T)
- Error (E -> E)
- Borrowed (N/A)

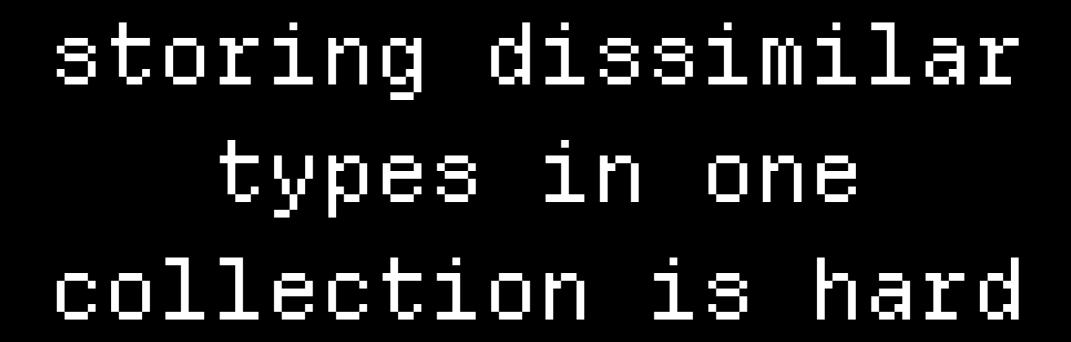
Borrowed Socket: Stores as [u8]

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

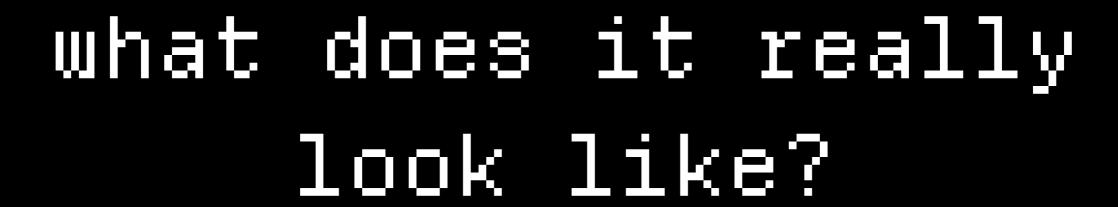














```
/// Receive an *owned* packet
/// from local code
fn recv_owned<T: 'static + Clone>(
    self: Pin<&mut Self>,
    data: 🏖T,
    hdr: HeaderSeq,
    ty_id: &TypeId,
) -> Result<(), SocketSendError>;
```



```
impl Socket<T: /* ... */> {
  fn recv_owned()
    this: NonNull<()>,
    that: NonNull<()>,
    hdr: HeaderSeq,
    ty: bTypeId,
  ) -> 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()
 7:
  // ...
// }
```

fun trick: turning generic fns into "normal" fn pointers

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

```
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 BorSerFn = fn(
  NonNullk()>, // data: &T
  HeaderSeq, // hdr
  &mut [u8], // ser dest
) -> Resultkusize, SocketSendError>;
```

