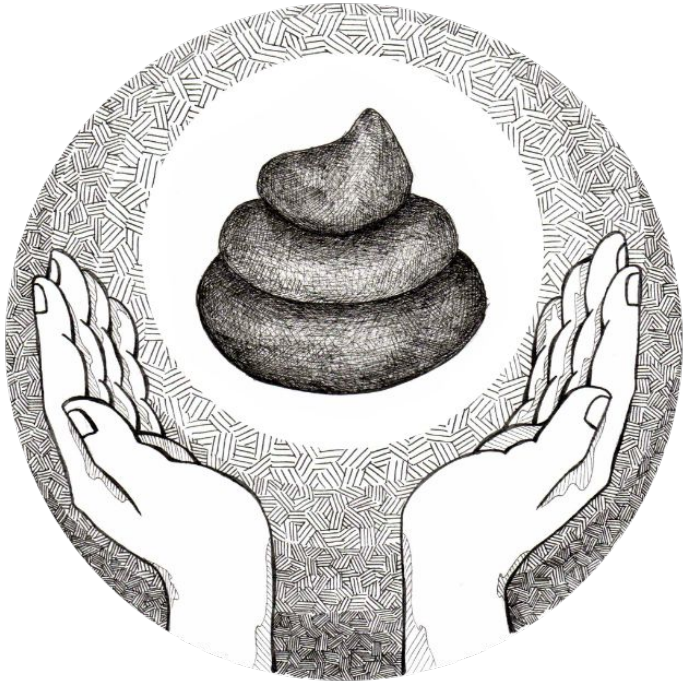
merde is not serde

Another take on (de)serialization in Rust



built-in value type

```rust
pub enum Value {
    Null,
    Bool(bool),
    Number(Number),
    String(String),
    Array(Vec<Value>),
    Object(Map<String, Value>),
}
```

```rust
pub enum Value<'s> {
    I64(i64),
    U64(u64),
    Float(OrderedFloat<f64>),
    Str(CowStr<'s>),
    Bytes(CowBytes<'s>),
    Null,
    Bool(bool),
    Array(Array<'s>),
    Map(Map<'s>),
}
```

CoW all the things

```
enum CowStr<'s> {
    Borrowed(&'s str),
    Owned(CompactString),
}

enum CowBytes<'a> {
    Borrowed(&'a [u8]),
    Owned(CompactBytes),
}
```

no proc macros

```rust
use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize, Debug)]
struct Point {
    x: i32,
    y: i32,
}
```

```rust
#[derive(Debug)]
struct Point {
    x: i32,
    y: i32,
}

merde::derive! {
    impl (Deserialize, Serialize) for
    struct Point { x, y }
}
```

```rust
#[derive(Debug, PartialEq, Eq)]
struct Name<'s> {
    first: CowStr<'s>,
    middle: Option<CowStr<'s>>,
    last: CowStr<'s>,
}
merde::derive! {
    impl (Serialize, Deserialize) for
    struct Name<'s> { first, middle, last }
}
```

built-in **`IntoStatic`** trait

```rust
pub trait IntoStatic {
    type Output: 'static;

    fn into_static(self) -> Self::Output;
}
```

`Deserialize` trait

```rust
pub trait Deserialize<'s>: Sized {
    async fn deserialize<D>(de: &mut D) -> Result<Self, D::Error<'s>>
    where
        D: Deserializer<'s> + ?Sized;
}
```

```rust
pub trait Deserializer<'s>: std::fmt::Debug {
    #[doc(hidden)]
    fn next(&mut self) -> Result<Event<'s>, Self::Error<'s>>;
}
```

```rust
#[derive(Debug)]
pub enum Event<'s> {
    I64(i64),
    U64(u64),
    F64(f64),
    Str(CowStr<'s>),
    Bytes(CowBytes<'s>),
    Bool(bool),
    Null,
    MapStart(MapStart),
    MapEnd,
    ArrayStart(ArrayStart),
    ArrayEnd,
}
```

`DeserOpinions` trait

```rust
pub trait DeserOpinions {
    fn deny_unknown_fields(&self) -> bool;

    fn map_key_name<'s>(
        &self,
        key: CowStr<'s>
    ) -> CowStr<'s>;

    fn default_field_value<'s, 'borrow>(
        &self,
        key: &'borrow str,
        slot: FieldSlot<'s, 'borrow>
    );
}
```

```
struct FrontMatterInOpinions;

impl DeserOpinions for FrontMatterInOpinions {
    fn map_key_name<'s>(&self, key: CowStr<'s>)
        -> CowStr<'s>
    {
        if key == "draft-code" {
            "draft_code".into()
        } else {
            key
        }
    }
}
```

```
merde::derive! {
    impl (Deserialize) for struct FrontmatterIn {
        title,
        template,
        date,
        // etc.
    } via FrontMatterInOpinions
}
```

`Serialize` trait

```rust
pub trait Serialize {
    async fn serialize<S>(&self, serializer: &mut S)
    -> Result<(), S::Error>
    where
        S: Serializer + ?Sized;
}
```

```rust
pub trait Serializer {
    type Error;

    // (note: this is an async fn but because
    // there's a lifetime, it won't let us!)
    fn write(&mut self, ev: Event<'_>)
    -> impl Future<Output = Result<(), Self::Error>>;
}
```

```rust
fn serialize_sync<T: Serialize>(&mut self, t: &T) ->
Result<(), Self::Error> {
    Serialize::serialize(t, self)
        .run_sync_with_metastack()
}

fn serialize<'s, T: Serialize>(
    &'s mut self,
    t: &'s T,
) -> impl Future<Output = Result<(), Self::Error>> + 's {
    Serialize::serialize(t, self)
        .run_async_with_metastack()
}
```

Stack full? Get another!

```rust
let cool_factor = 100_000;

let first_half = "[".repeat(cool_factor);
let second_half = "]".repeat(cool_factor);
let input = format!("{first_half}{second_half}");

let value: Value<'_> =
merde::json::from_str(&input[..]).unwrap();
```

Drop bombs

Async I/O!

dynosaur

```rust
pub trait Deserialize<'s>: Sized {
    async fn deserialize<D>(de: &mut D)
    -> Result<Self, D::Error<'s>>
    where
        D: Deserializer<'s> + ?Sized;
}

pub trait Deserialize<'s>: Sized {
    async fn deserialize(de: &mut dyn Deserialize<'s>)
    -> Result<Self, D::Error<'s>>;
}
```

Codegen?