Parse Don't Validate

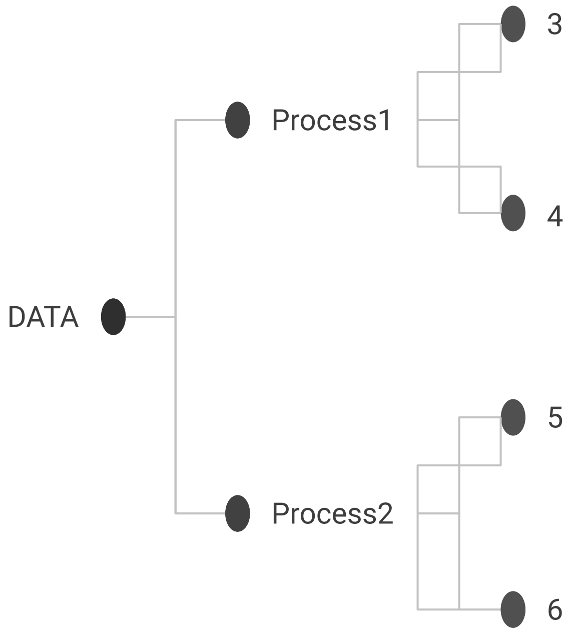
"Perdón imposible que cumpla su condena."– Carlos I
"Edvardum occidere nolite timere bonum est."– Louve de France

Roots

Shotgun parsing is a programming antipattern whereby parsing and input-validating code is mixed with and spread across processing code—throwing a cloud of checks at the input, and hoping, without any systematic justification, that one or another would catch all the “bad” cases. — *The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them*

Validation-based approaches make it extremely difficult or impossible to determine if everything was actually validated up front or if some of those so-called “impossible” cases might actually happen. — *Alexis King*

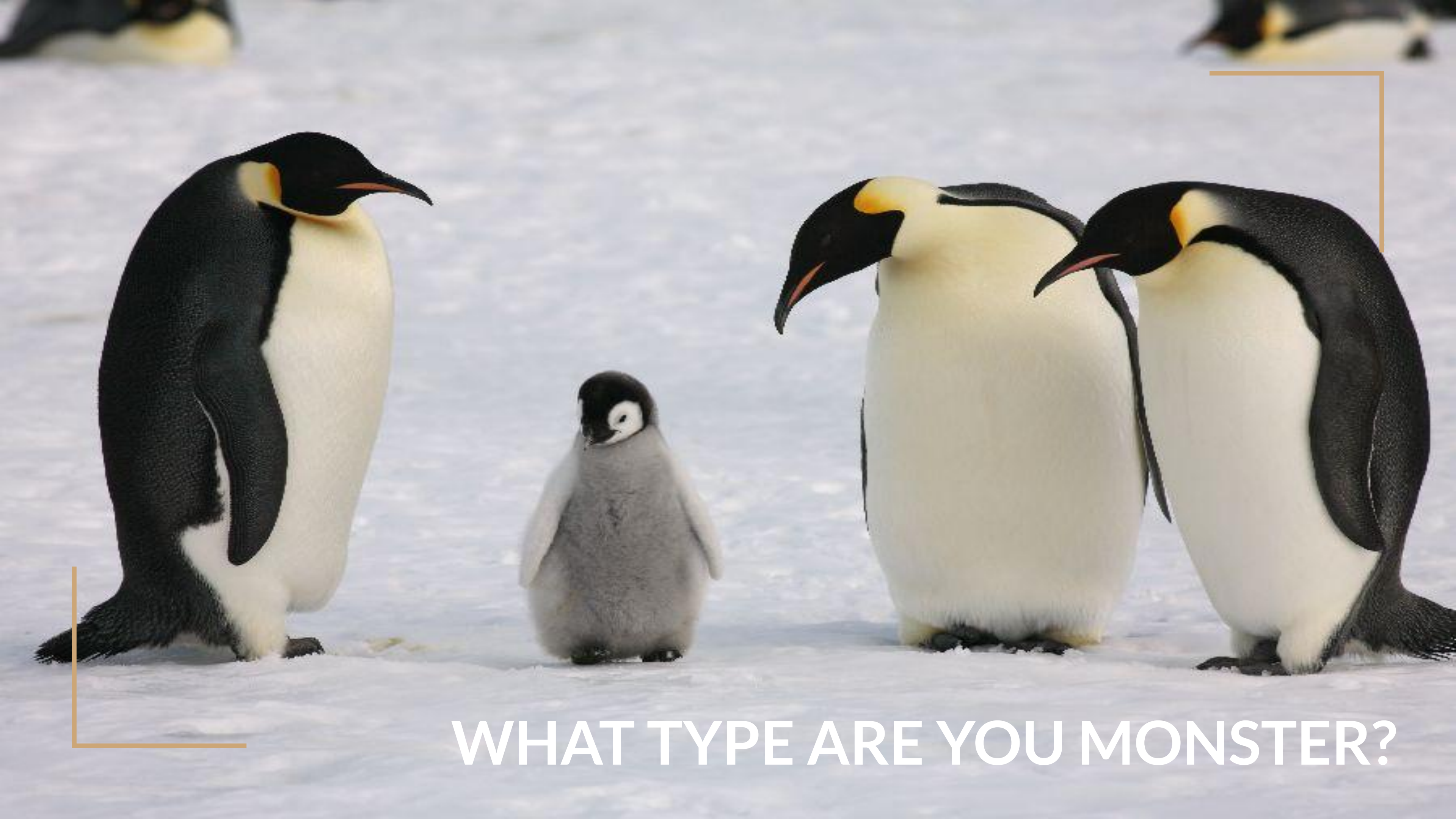
Full Recognition



http://langsec.org/occupy/

Approaches

- Servant — family of combinators for defining webservices API
- Protobuf — extensible mechanism for serializing structured data
- XSD — formal description of the elements in XML document
- JSON [Hyper-]Schema — vocabulary that allows to annotate and validate
- XML — extensible markup language
- JSON — extensible trivial data markup language
- CSV — extensible tuples markup language
- REGEX — extensible write-only markup language
- Morse Code — sequences of two different signal durations, *dits* and *dahs*
- Natural Languages — convoluted controversial expression language

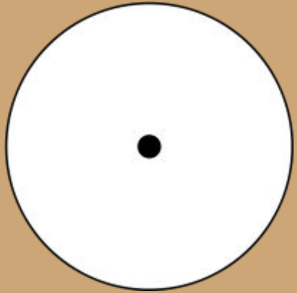


WHAT TYPE ARE YOU MONSTER?

Monad

Terminology

Monad (from Greek μονάς monas, “unit” in turn from μόνος monos, “alone”,) refers in *cosmogony* (creation theories) to the first being, divinity, or the totality of all beings.



The ***circled dot*** was used by the *Pythagoreans* and later Greeks to represent the first *metaphysical being*, the **Monad** or **The Absolute**.

Either Or

- Three boxes / three doors quiz (change the choice)
- “Is it a gift” by Amazon
- Drawers in the kitchens
- Tactile sensation touching laptop in a case / keys in the pocket
- Blackjack with card faces down

- **Similar to the external observer, wrapping value**

Aspects: It’s not a hack, it’s a feature

```
class Adder

    def initialize a, b

        @a, @b = a, b

    end

    def sum

        [@a, @b].inject &:+

    end

end
```

```
fail unless User.current.permitted?
```

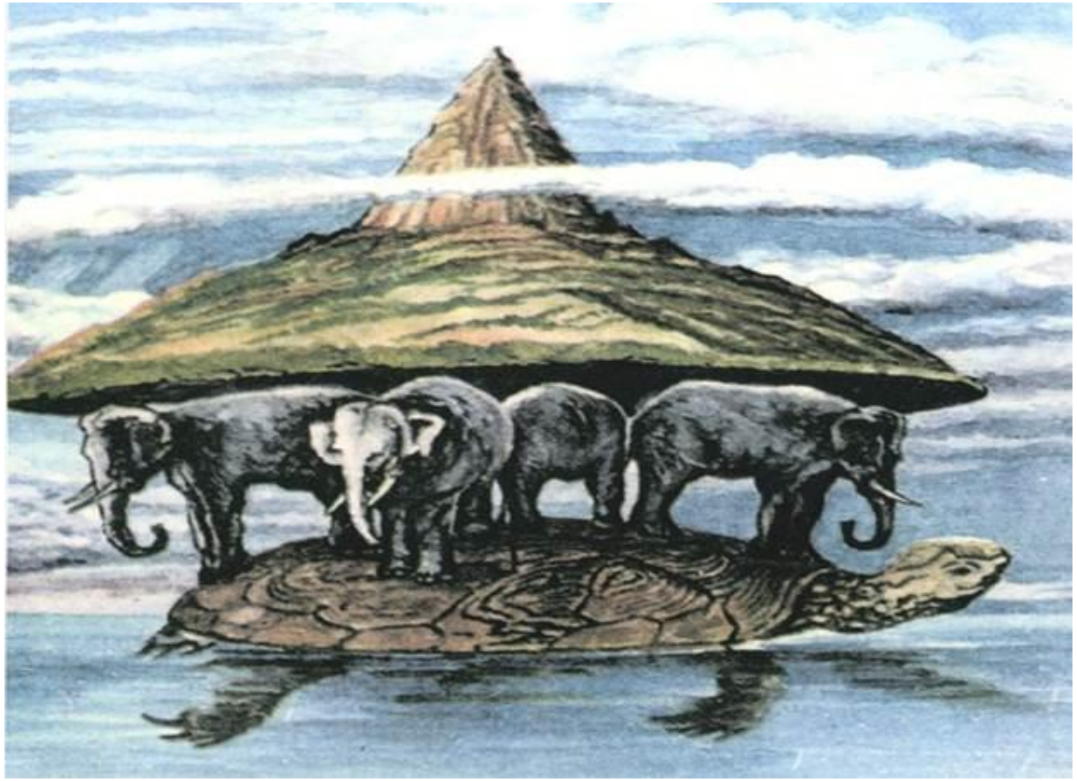
```
logger.debug("#{self} :: returning #{result}")
```

Missed Validation Step

```
{
 foo: 42,
 bar: {
  date_time: null,
  name: "Baz"
 }
}
```

The Flat Surface

- Validation
- Coercion
- Generation
- Serialization



All in One

The Value

- Knows how to serve herself
- Instantiatable from external sources
- Serializable
- Can generate a Stream of self-alike instances
- Transparent wrapping

Zoomorphism Terminology

Protocol

Behaviour

Duck typing

Don’t Do

```
currency = "USD"

currency_pair = "EUR/USD"

date_time = "2021-02-11T13:27:10.727483Z"

total_amount = "42"

customer = {

  name: "Jane Doe"

}
```

Getters and Setters

Don’t update the structure holding your data directly.

Even if your language of choice allows it, avoid implementing recursive getters and setters.

Use a powerful `Access` approach.

It’s like `XPATH`. But with getters and setters.

DO

- How data is managed in different languages?
- What approach could guarantee the data consistency?

- Boundaries
- Parsing
- Self-managing data objects

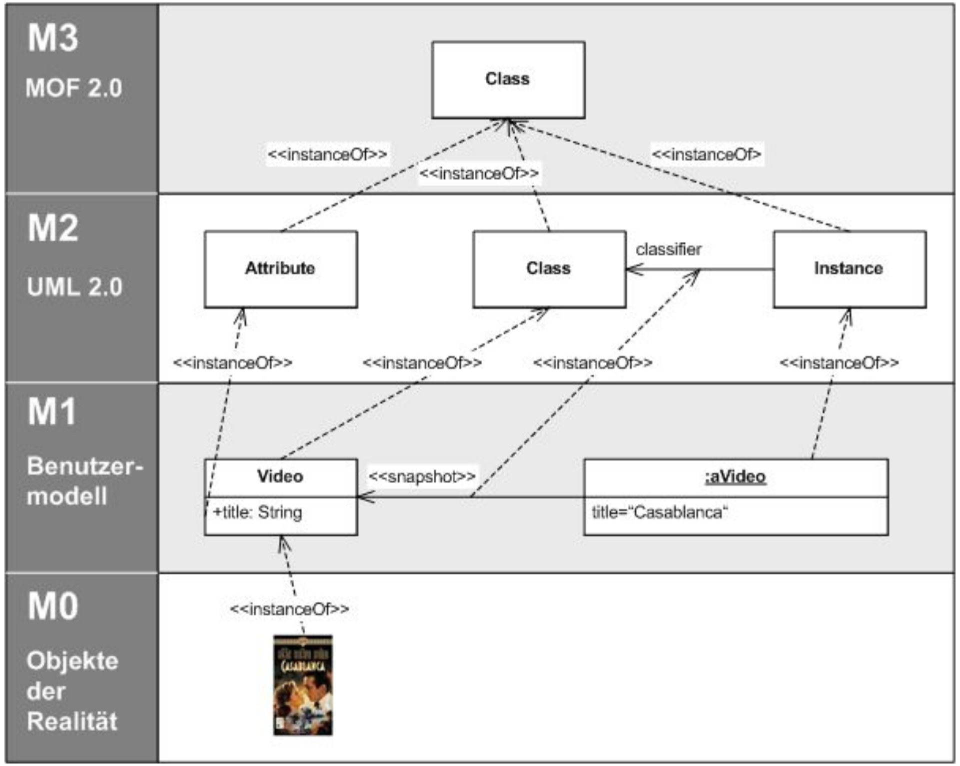
All in One

The Value

- Knows how to serve herself
- Instantiatable from external sources
- Serializable
- Can generate a Stream of self-alike instances
- Transparent wrapping

WTM?

Meta (from the Greek preposition and prefix *meta-* (μετά-) meaning "after", or "beyond") is a prefix used in English to indicate a concept which is an abstraction from another concept, used to complete or add to the latter.



Tree → Branch → Leaf

Leaf is a wrapper for Values

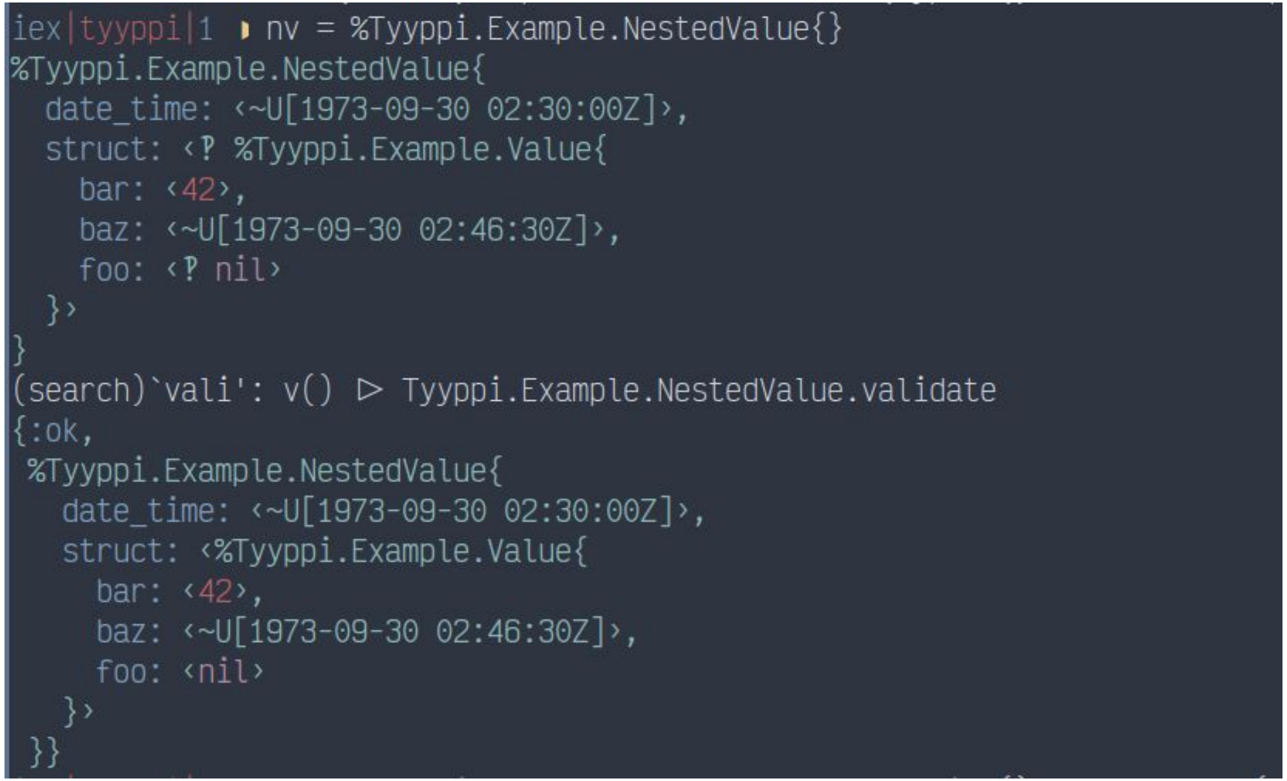
Value(42)

Tree is a brick to build deeply nested structs

(Tree Value(42), Value(:foo), Tree(children))

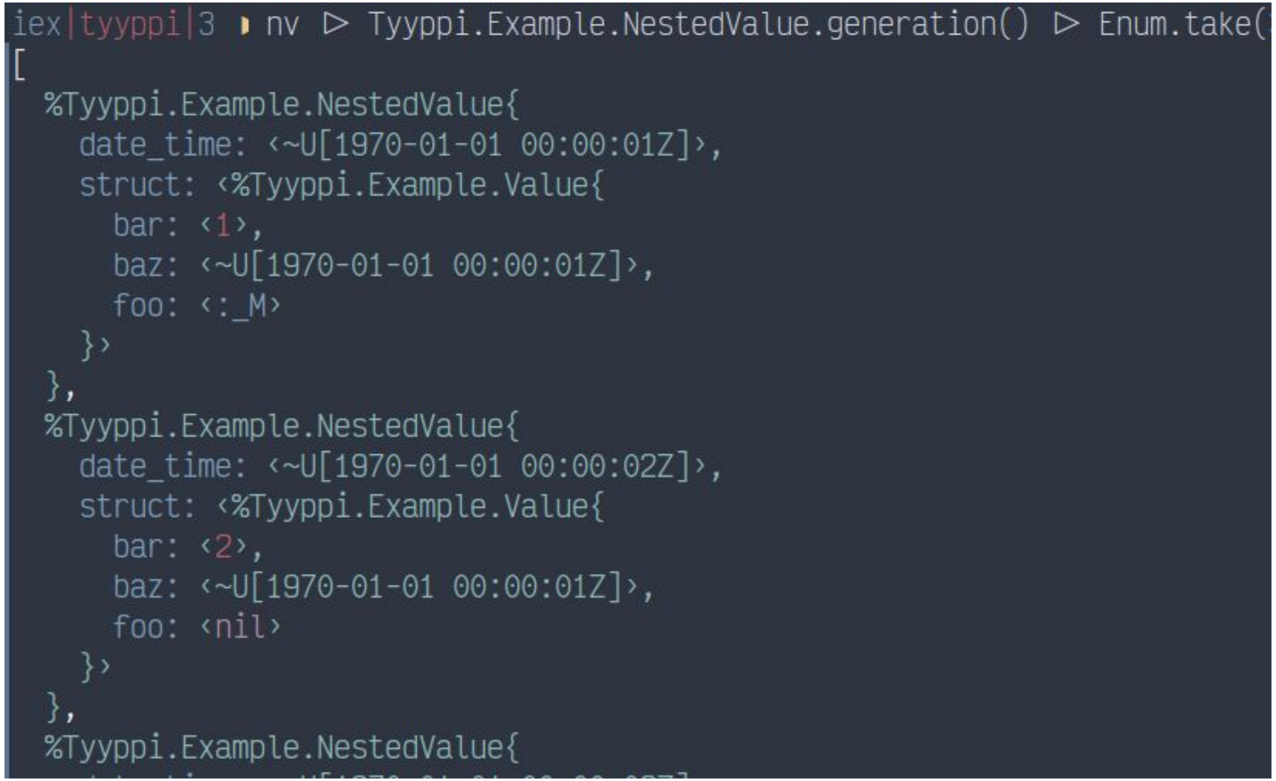
Leaf(Tree) exposes a wrapper for Tree which is a Value

Show the Code https://hexdocs.pm/tyyppi/Tyyppi.Example.Nested.html#content



```
iex|tyyppi|1 ▸ nv = %Tyyppi.Example.NestedValue{}
%Tyyppi.Example.NestedValue{
  date_time: ‹~U[1973-09-30 02:30:00Z]›,
  struct: ‹⁈ %Tyyppi.Example.Value{
    bar: ‹42›,
    baz: ‹~U[1973-09-30 02:46:30Z]›,
    foo: ‹⁈ nil›
  }›
}
(search)`vali': v() ▷ Tyyppi.Example.NestedValue.validate
{:ok,
 %Tyyppi.Example.NestedValue{
   date_time: ‹~U[1973-09-30 02:30:00Z]›,
   struct: ‹%Tyyppi.Example.Value{
     bar: ‹42›,
     baz: ‹~U[1973-09-30 02:46:30Z]›,
     foo: ‹nil›
   }›
 }}
```

Generate https://hexdocs.pm/tyyppi/Tyyppi.Example.Nested.html#content



```
iex|tyyppi|3 ▶ nv ▷ Tyyppi.Example.NestedValue.generation() ▷ Enum.take(
[
  %Tyyppi.Example.NestedValue{
    date_time: ‹~U[1970-01-01 00:00:01Z]›,
    struct: ‹%Tyyppi.Example.Value{
      bar: ‹1›,
      baz: ‹~U[1970-01-01 00:00:01Z]›,
      foo: ‹:_M›
    }›
  },
  %Tyyppi.Example.NestedValue{
    date_time: ‹~U[1970-01-01 00:00:02Z]›,
    struct: ‹%Tyyppi.Example.Value{
      bar: ‹2›,
      baz: ‹~U[1970-01-01 00:00:01Z]›,
      foo: ‹nil›
    }›
  },
  %Tyyppi.Example.NestedValue{
```

Special Thanks

- **John** who said “stop doing weird metaprogramming stuff, do something useful”
- Ju @arkh4m who coined “parse not validate” motto for me
- **Coronita** who has the slides design chosen (rocket jump onto the table directly to my laptop’s keyboard, don’t ask)
- **Ristretto**, **Booker’s**, and **Lucky Strike**.

Ding

¿?