TIM MCGILCHRIST

OCAML FOR FUN AND PROFIT

INTRODUCTION

- Tim McGilchrist (https://github.com/tmcgilchrist)
- Pragmatic Functional Programmer
- Works with OCaml at Tarides
- Tarides focuses on building functional systems using OCaml

MOTIVATION

- Practical OCaml Programming, applying Functional Programming to everyday problems.
- Why you should consider using OCaml for your next project?
- Experience Report of using OCaml in anger.

WHAT IS OCAML?

OCaml is a general-purpose, industrial-strength programming language with an emphasis on expressiveness and safety.

- Native compiler and byte code interpreter
- Static type-checking
- Type inferencing
- Algebraic data types (ADT)
- Automatic memory management with a Garbage Collector
- Typed module system
- First class modules, GADTs, Objects, and more.

🐫🐫🐫🐫

SO WHO USES IT? WHAT FOR?

Relied upon by many different companies.

Who use it for:

- Building rich web applications
- Low level systems programming
- Performance sensitive services
- Designing Hardware
- Modelling complex business domains
- Mobile iOS/Android applications

facebook Microsoft docker

Jane Street Bloomberg ahrefs

LexiFi

PRAGMATICALLY SOLVING PROBLEMS

OCaml provides you with all you will need to program in general; defining data structures, functions and the means for combining them.

What is the minimal set of features that OCaml brings that makes a difference?

- Algebraic Data Types (ADT)
- Type Inference
- Functions as first class concepts
- Strict evaluation
- Large scale typed patterns

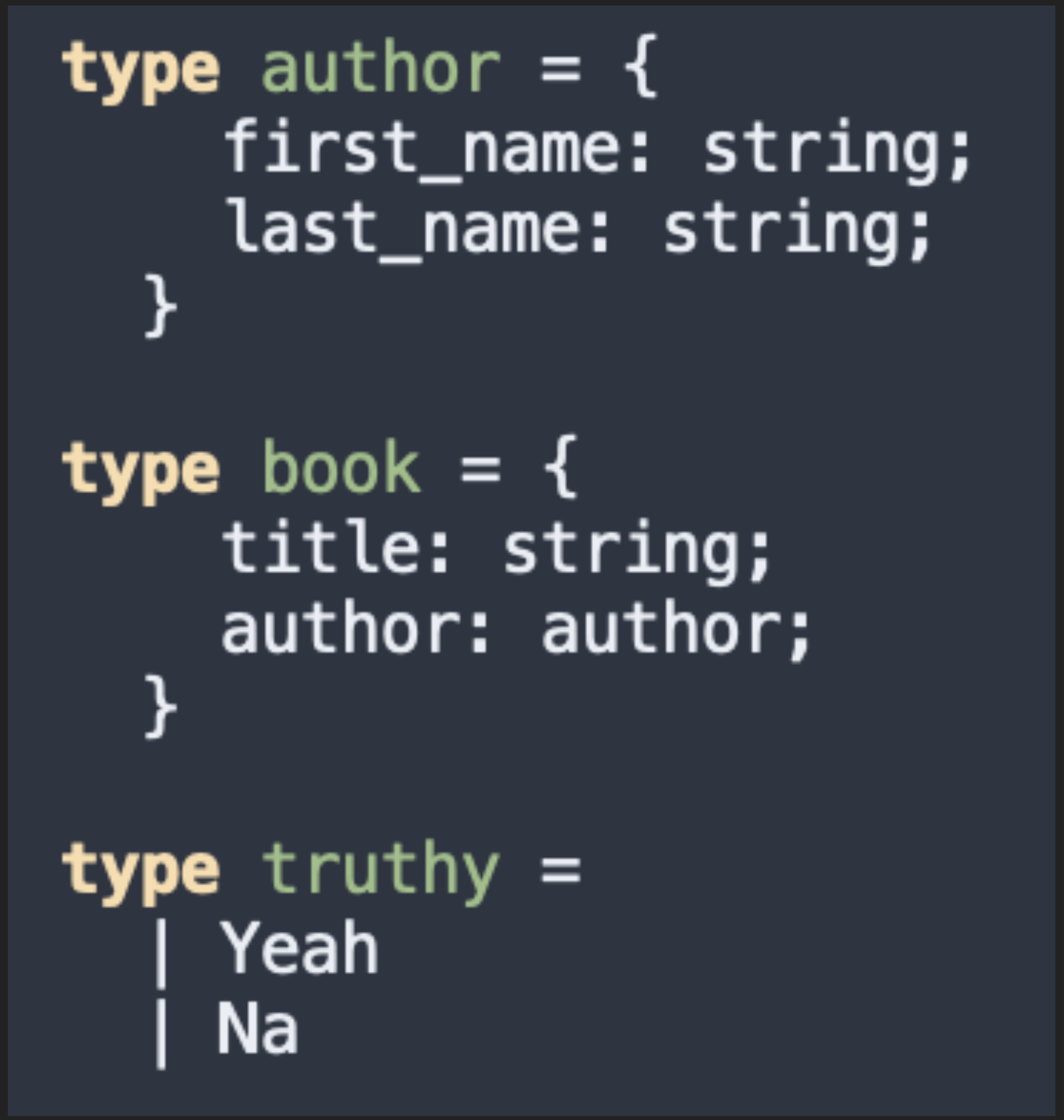
PRAGMATICALLY SOLVING PROBLEMS

Algebraic Data Types as typed building blocks.

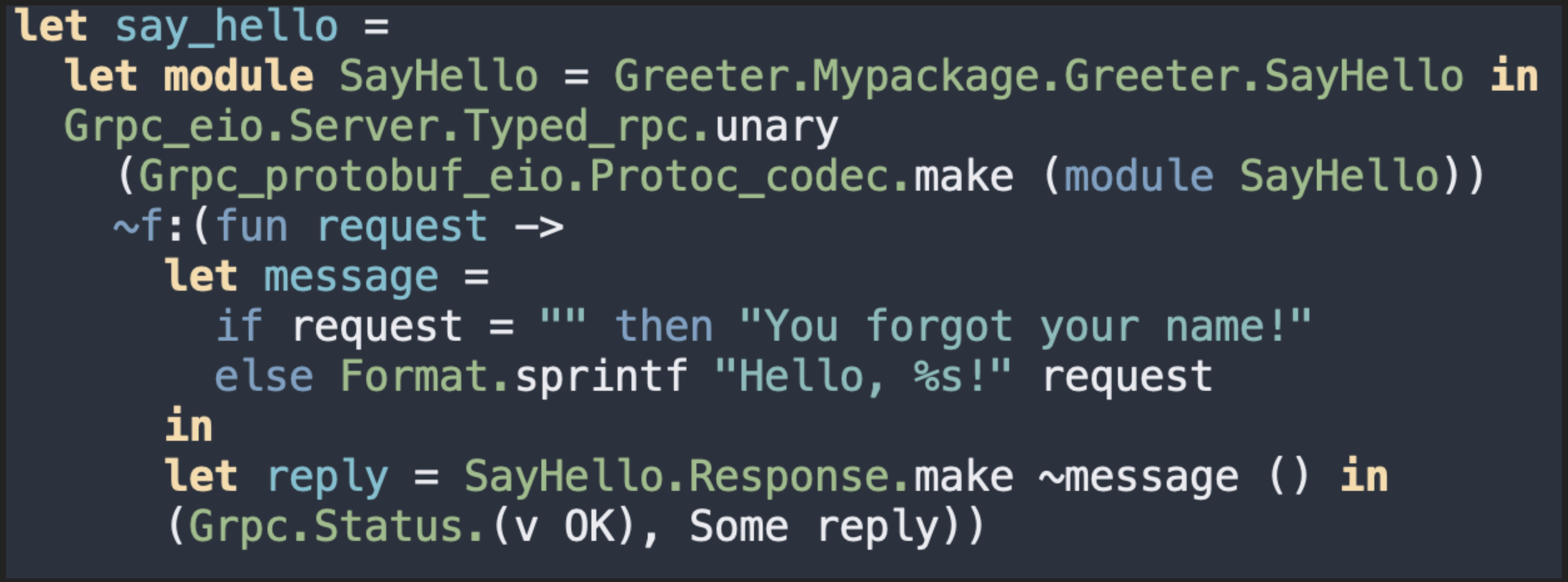
Records as collections of labelled data.

Variants as choices between different options.

Naturally oriented around using functions everywhere.



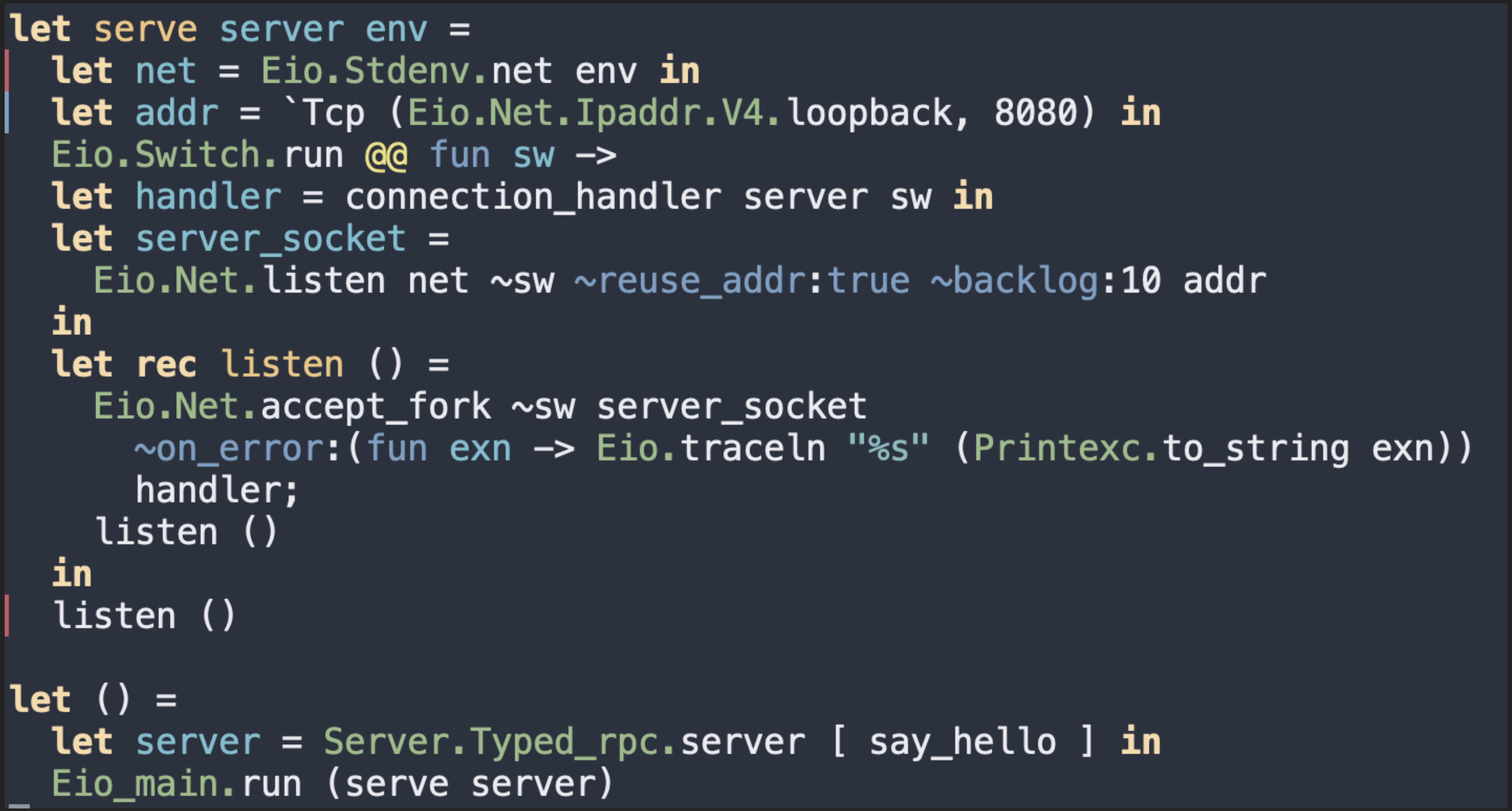
```
type author = {
    first_name: string;
    last_name: string;
  }

type book = {
    title: string;
    author: author;
  }

type truthy =
  | Yeah
  | Na
```



```
let say_hello =
  let module SayHello = Greeter.Mypackage.Greeter.SayHello in
  Grpc_eio.Server.Typed_rpc.unary
    (Grpc_protobuf_eio.Protoc_codec.make (module SayHello))
    ~f:(fun request ->
      let message =
        if request = "" then "You forgot your name!"
        else Format.sprintf "Hello, %s!" request
      in
      let reply = SayHello.Response.make ~message () in
      (Grpc.Status.(v OK), Some reply))
```

PRAGMATICALLY SOLVING PROBLEMS

- Type inference in OCaml is complete.
- A strictly evaluated language, maps to what you expect.
- Support for programming in the large with modules and abstractions over modules.

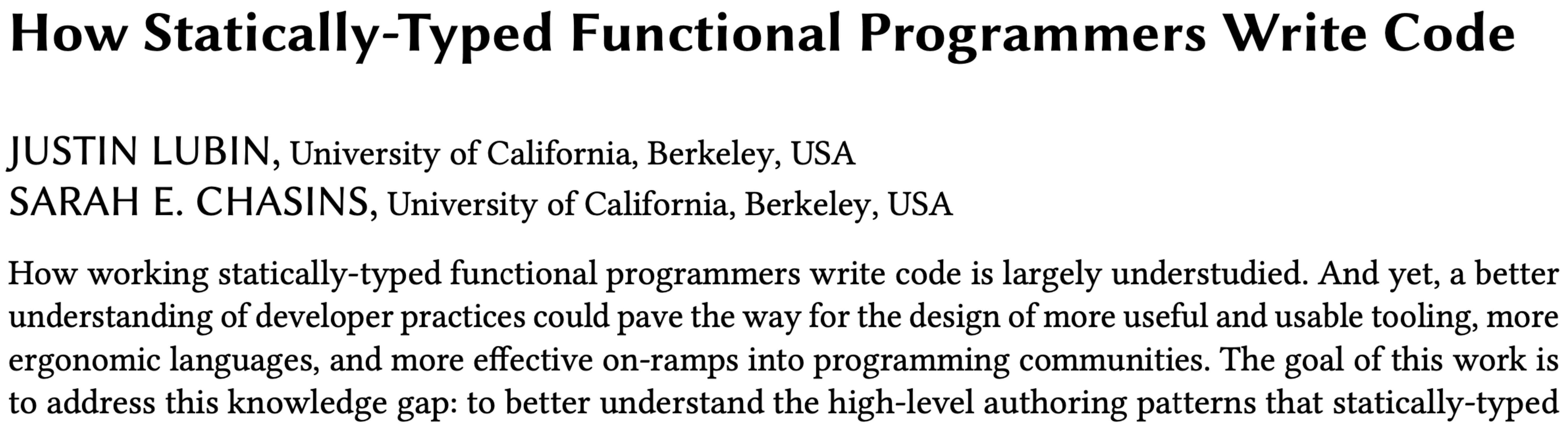


```
let serve server env =
  let net = Eio.Stdenv.net env in
  let addr = `Tcp (Eio.Net.Ipaddr.V4.loopback, 8080) in
  Eio.Switch.run @@ fun sw ->
  let handler = connection_handler server sw in
  let server_socket =
    Eio.Net.listen net ~sw ~reuse_addr:true ~backlog:10 addr
  in
  let rec listen () =
    Eio.Net.accept_fork ~sw server_socket
      ~on_error:(fun exn -> Eio.traceln "%s" (Printexc.to_string exn))
      handler;
    listen ()
  in
  listen ()

let () =
  let server = Server.Typed_rpc.server [ say_hello ] in
  Eio_main.run (serve server)
```

HOW STATICALLY TYPED FUNCTIONAL PROGRAMMERS WRITE CODE

- Iterative development of types and their inhabitants.
- Type errors as todo lists.
- Sketching with type signatures and holes.
- Write module signatures containing types and function signatures as design tools.
- Compiler is a companion in writing code.

https://dl.acm.org/doi/10.1145/3485532



How Statically-Typed Functional Programmers Write Code

JUSTIN LUBIN, University of California, Berkeley, USA
SARAH E. CHASINS, University of California, Berkeley, USA

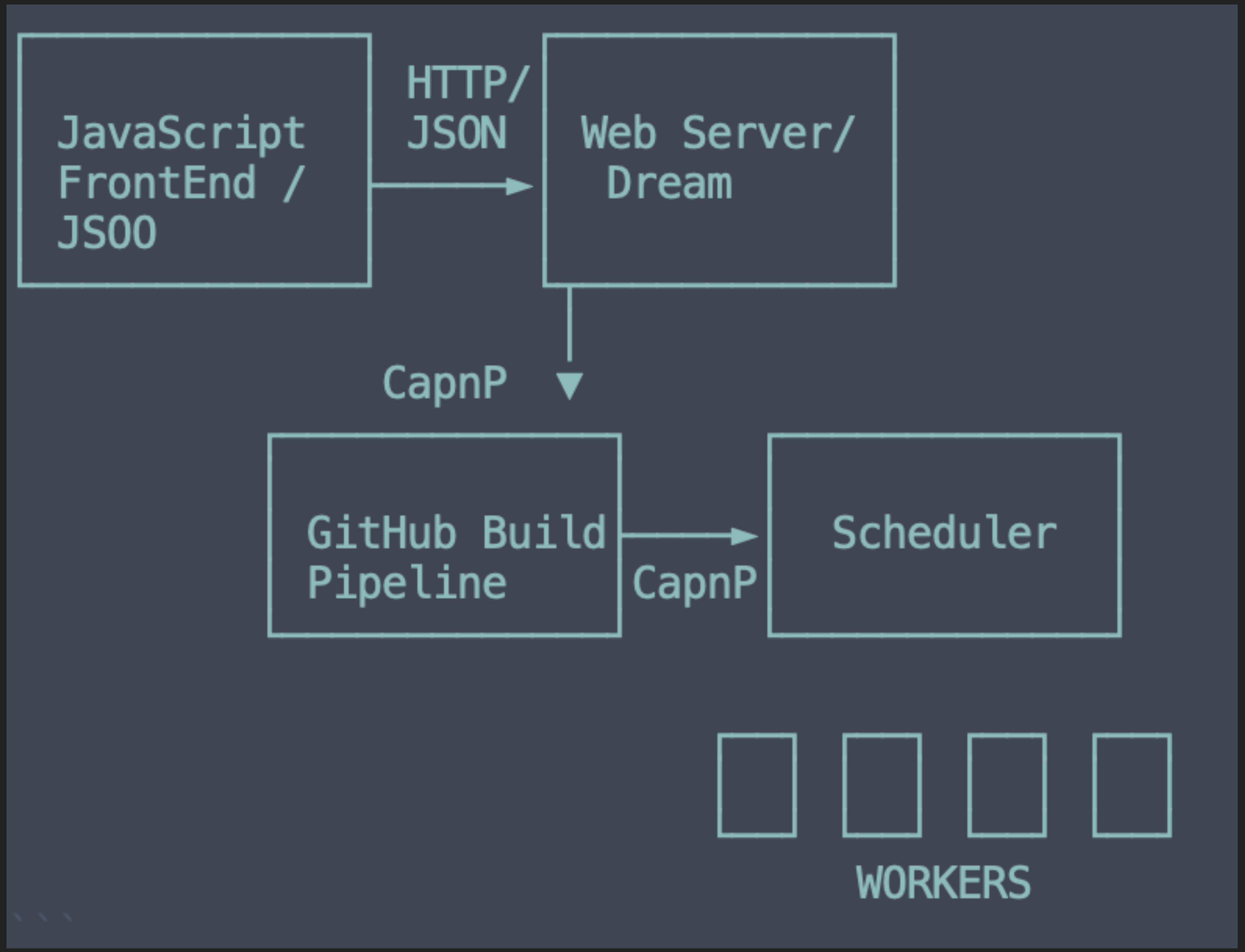
How working statically-typed functional programmers write code is largely understudied. And yet, a better understanding of developer practices could pave the way for the design of more useful and usable tooling, more ergonomic languages, and more effective on-ramps into programming communities. The goal of this work is to address this knowledge gap: to better understand the high-level authoring patterns that statically-typed

CASE STUDY - INTEROP WITH OTHER LANGUAGES

NO programming language is an island. You will often need to call into other languages, let other languages call you, or somehow pass information between systems.

What does OCaml bring in this area?

- typed discipline for enforcing boundaries
- type directed property testing
- make invalid states unrepresentable
- great support for RPC protocols



CASE STUDY - EXTENDING TO THE FRONT-END

OCaml has a mature story for compiling to JavaScript.

Two well established options for compiling to Javascript:

- Js_of_ocaml (aka JSOO) compiler from OCaml byte code to Javascript (http://ocsigen.org/js_of_ocaml/)
- Melange branded as OCaml for JavaScript developers (https://melange.re)
- Compiles each source file into a Javascript module.
- Mature bindings to React.
- Focus on interoperability with Javascript ecosystem.

🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪

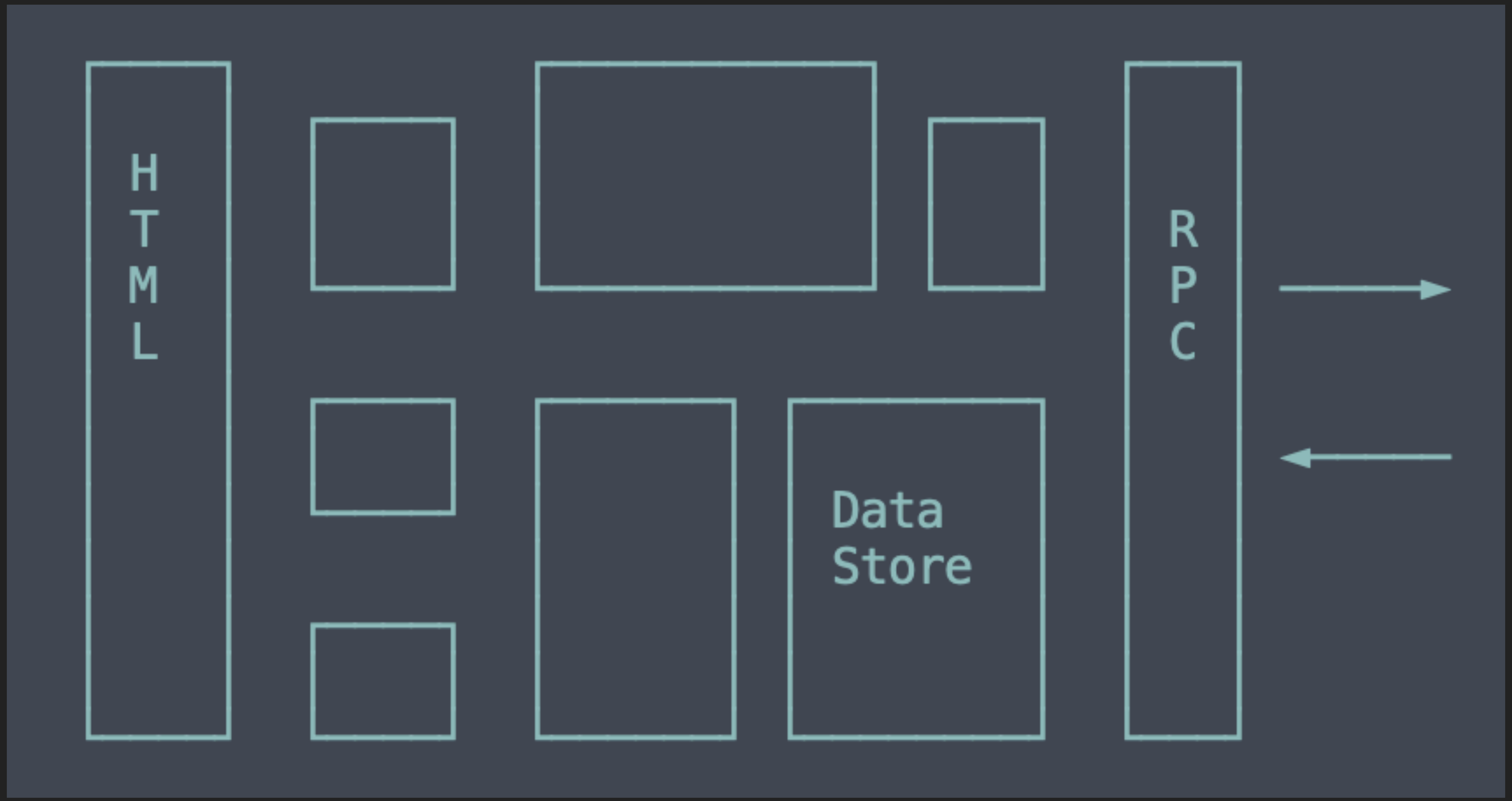
CASE STUDY - EXTENDING TO THE FRONT-END

OCaml makes a great front-end language.

Extra benefits like type safe HTML, typed routes and URL links.

- typed HTML, valid by construction
- typed routes and links, valid by construction
- typed data components
- closing the loop between front and backend
- more powerful type system than typescript

In the future WASM support.



COLLABORATE USING TYPES

Practice Type Driven Development.

Use types and interfaces as a design and collaboration tool.

Write types, function signatures and modules before ever writing an implementation.

Choose how tightly you want to type your programs.

Focus on the crucial components with stronger typed discipline.

Like all good things, there needs to be balance.

🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪🐪

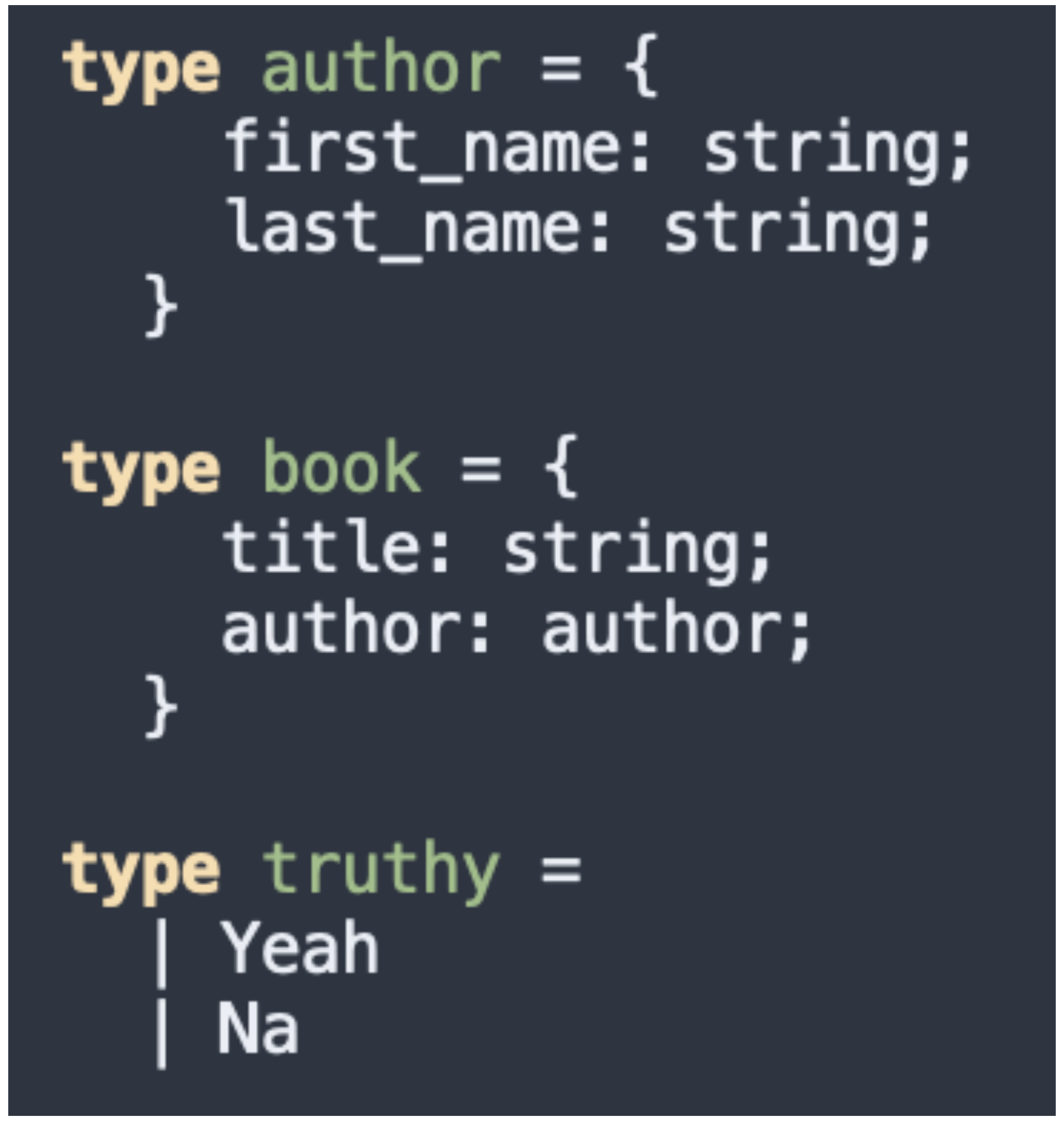
REFACTORING FEARLESSLY

More of your time will be spent reading and modifying existing code, than writing new code.

Choose a language that provides a great experience for that.

OCaml gives you a huge leg up in dealing with code maintenance.

Exhaustive pattern matching combined with type information tells you where there are issues.



```
type author = {
    first_name: string;
    last_name: string;
  }

type book = {
    title: string;
    author: author;
  }

type truthy =
  | Yeah
  | Na
```

REFACTORING FEARLESSLY

Test Driven Development loop:

1. Write some tests describing how you expected something to work.
2. Run the test, they fail.
3. Write some code to make 1 or more tests pass.
4. Either write more code or fix more tests.

Replace with Type Driven Development.

GROWING YOUR TEAM

What if we train our engineers and they leave?

What if we don't and they stay? - HN comments

- Small language with orthogonal features, it is learnable!
- Teach engineers, you will anyway
- OCaml memory management vs Rust
- OCaml type system features vs Go or Typescript
- OCaml vs large languages like Scala or Haskell

PROGRAMMING IN THE LARGE

What does it mean to program in the large with OCaml?

- Module - named container for types and functions
- Signatures - interfaces for modules
- Package - collection of modules that share a common purpose

- Functors - functions from modules to modules
- GADTs - Generalised Algebraic Data Types an extension of Algebraic Data Types
- First Class Modules - modules as values
- Objects / Classes - Object Orientation in your Functional Programming

🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫

RETROSPECTIVE ON USING OCAML

Personal perspective on using OCaml for CI work.

- OCaml is a small language with orthogonal features, that work well together.
- Tooling is surprisingly good.
- Large range of quality solutions to problems.
- Community is friendly.
- Backwards compatibility is taken seriously.

🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫🐫

RETROSPECTIVE ON USING OCAML

Selection of systems built with OCaml:

- obuilder - providing cross platform sandbox execution library supporting Linux, MacOS, Windows and FreeBSD.
- ocaml.ci.dev - CI system for OCaml projects hosted on GitHub and GitLab.
- solver-service - embarrassingly parallel 0Install solver able to saturate large multicore machines.
- ocaml-docs-ci - large scale documentation generation for all OCaml packages. Builds documentation for 25,000 package versions.
- opam repository ci - validates new packages against matrix of operating systems, hardware architectures and reverse dependencies, ensuring the health of the package universe.

WRAP IT UP

OCaml is a pragmatic statically typed programming language.

Used to build real systems not just toys.

- types are fantastic!!!
- collaborate using types
- a small language with orthogonal features
- native support for refactoring and code maintenance
- language support for programming in the large

RESOURCES

- Real World OCaml https://realworldocaml.org/
- Ocsigen, a Web Programming Framework https://doi.org/10.1145/1631687.1596595
- How Statically-Typed Functional Programmers Write Code. https://jlubin.net/assets/oopsla21.pdf
- LexiFi: Modelling language for Finance https://ocaml.org/success-stories/modeling-language-for-finance
- Ahref using Melange, OCaml to build index https://tech.ahrefs.com/beyond-typescript-differences-between-typed-languages-f3e14253
- Getting started with ATD and Melange https://tech.ahrefs.com/getting-started-with-atdgen-and-bucklescript-1f3a14004081
- How Docker Desktop uses OCaml https://www.docker.com/blog/how-docker-desktop-networking-works-under-the-hood/

DIRECTORS EXTENDED EDITION

What sucks about OCaml?

Exciting new developments:

- Algebraic Effects - Effect handlers are a mechanism for modular programming with user-defined effects.
- Effect handlers are a generalisation of exception handlers and enable non-local control-flow mechanisms such as resumable exceptions, lightweight threads, coroutines, generators and asynchronous I/O to be composably expressed.
- Shared memory parallelism - enables parallel programming across cores and threads.
- WASM support - Wasm_of_ocaml compiles OCaml byte code to WebAssembly. Uses Garbage Collection, tail-call and exception handling extensions. Wasocaml compiles OCaml IR to WebAssembly.
- Eio - Effects-based direct-style IO for multicore OCaml.