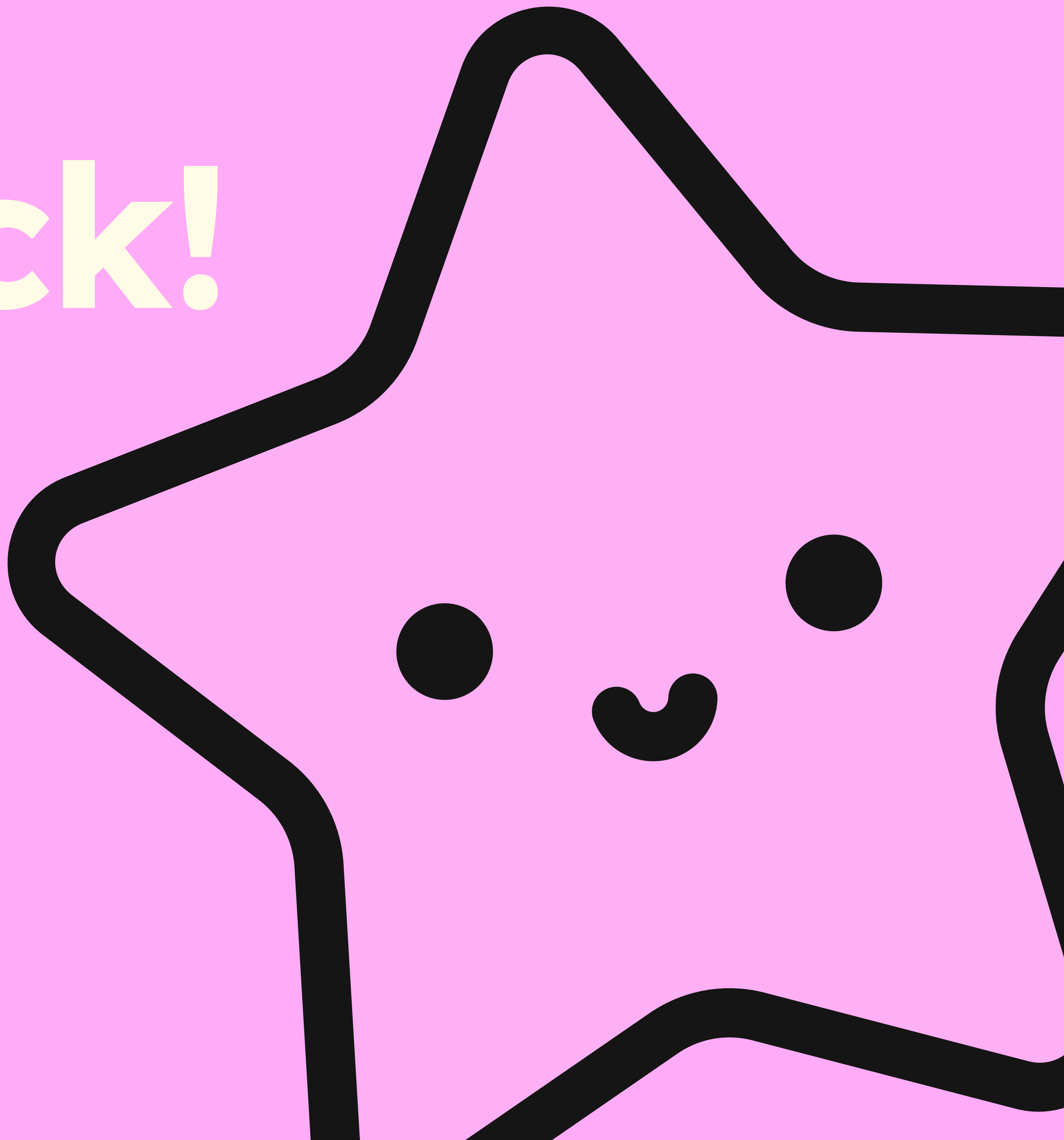
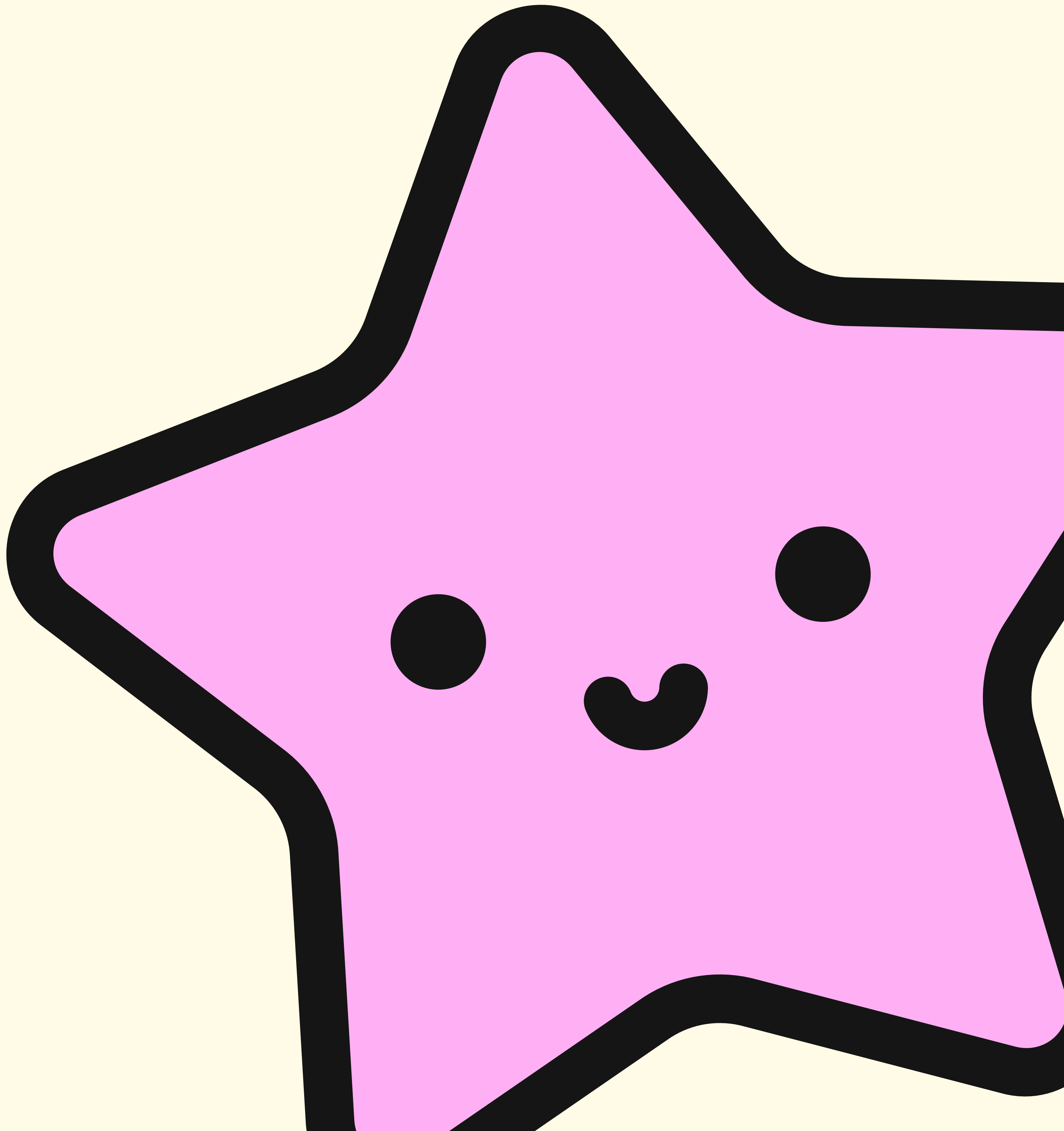
Welcome to the Gleam track!



Adopting Gleam the boring way



Code BEAM Europe 2025 — Yoshie Reusch

this talk:

- …**won’t** tell you to rewrite everything in Gleam

this talk:

- …**won’t** tell you to rewrite everything in Gleam
- …will **not** try to convince you there’s no risk involved

this talk:

- …**won’t** tell you to rewrite everything in Gleam
- …will **not** try to convince you there’s no risk involved
- …but **maybe** exploring Gleam doesn’t have to be as scary as it seems

this talk:

- …**won’t** tell you to rewrite everything in Gleam
- …will **not** try to convince you there’s no risk involved
- …but **maybe** exploring Gleam doesn’t have to be as scary as it seems
- …aims to show how to Gleam, Erlang, Elixir (and Javascript) can all be used **together** for great good

this talk:

- …**won’t** tell you to rewrite everything in Gleam
- …will **not** try to convince you there’s no risk involved
- …but **maybe** exploring Gleam doesn’t have to be as scary as it seems
- …aims to show how to Gleam, Erlang, Elixir (and Javascript)
  can all be used **together** for great good
- … get you more **excited** for the talks to come

👋 hi!

- ai & data science graduate @ othr
- fp enthusiast and web dev for half my life
- the smaller half of the Lustre core team
- freelancing software engineer, consultant

Cultural

Technical

Cultural

- Is Gleam a good fit for us?

Technical

Cultural

- Is Gleam a good fit for us?
- How will we hire for it?

Technical

Cultural

- Is Gleam a good fit for us?
- How will we hire for it?
- Is Gleam ready to solve our problems?

Technical

Cultural

- Is Gleam a good fit for us?
- How will we hire for it?
- Is Gleam ready to solve our problems?

Technical

- What will we do if something isn’t available yet?

Cultural

- Is Gleam a good fit for us?
- How will we hire for it?
- Is Gleam ready to solve our problems?

Technical

- What will we do if something isn’t available yet?
- We already have this huge codebase, what about that?

Gleam itself was **never** the problem

Gleam is a simple language!

-# One might even say it's "boring"

```
/// Handler for the `/` route, serves our `index.html` file from `priv/static`
///
fn index(ctx: Context, _req: Request) -> Response {
  let body = wisp.File(filepath.join(ctx.static_directory, "index.html"))

  wisp.ok()
  |> response.set_header("content-type", "text/html; charset=utf-8")
  |> response.set_body(body)
}
```

Types are really helpful!

Especially in new and unfamiliar codebases

```
/// Handler for the `/` route, serves our `index.html` file from `priv/static`
///
fn index(ctx: Context, _req: Request) -> Response {
  let body = filepath.join(ctx.static_directory, "index.html")

  wisp.ok()
  |> response.set_header("content-type", "text/html; charset=utf-8")
  |> response.set_body(body)
}
```

Types are really helpful!

Especially in new and unfamiliar codebases

```
/// Handler for the `/` route, se
///
fn index(ctx: Context, _req: Requ
  let body = filepath.join(ctx.st

  wisp.ok()
  |> response.set_header("content
  |> response.set_body(body)
}
```

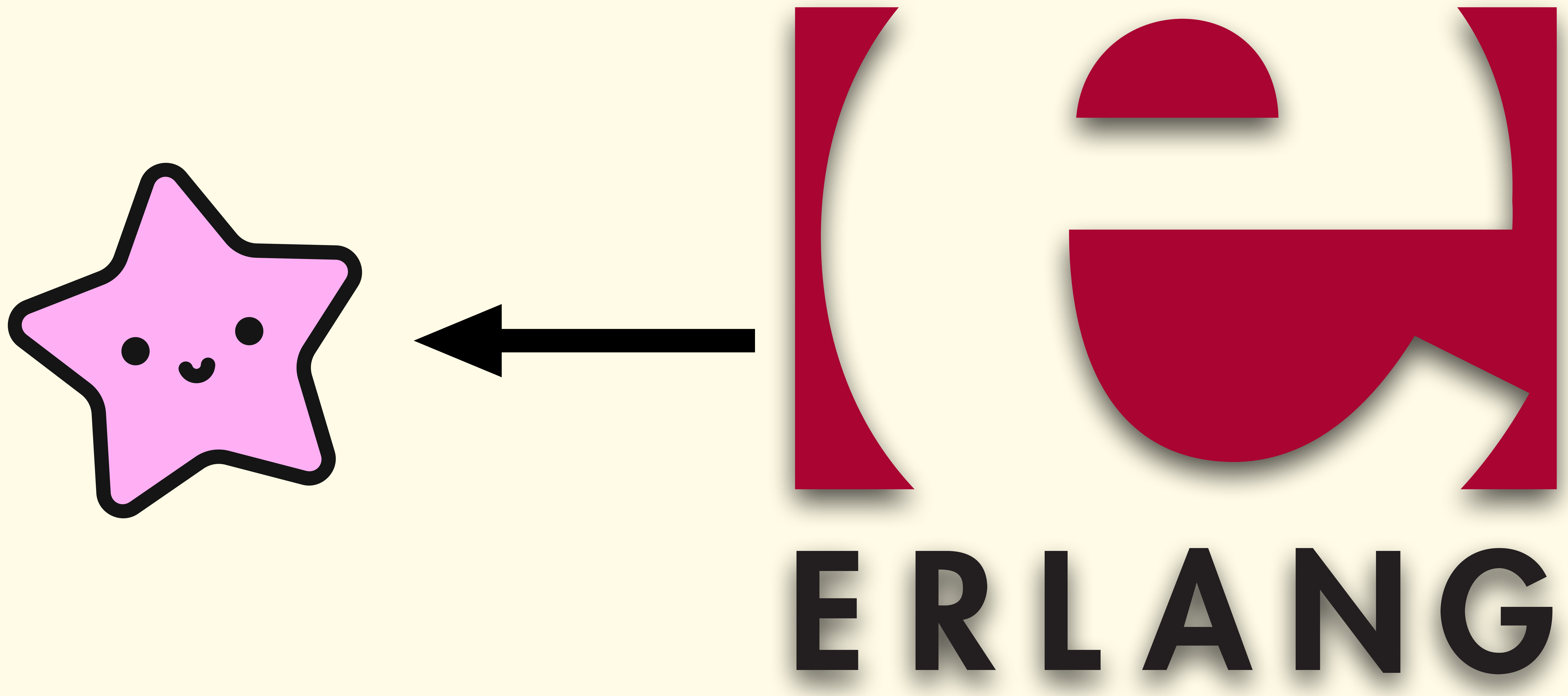
```
The type of this returned value doesn't match the
return type annotation of this function.

Expected type:

    Response(wisp.Body)

Found type:

    Response(String)
```

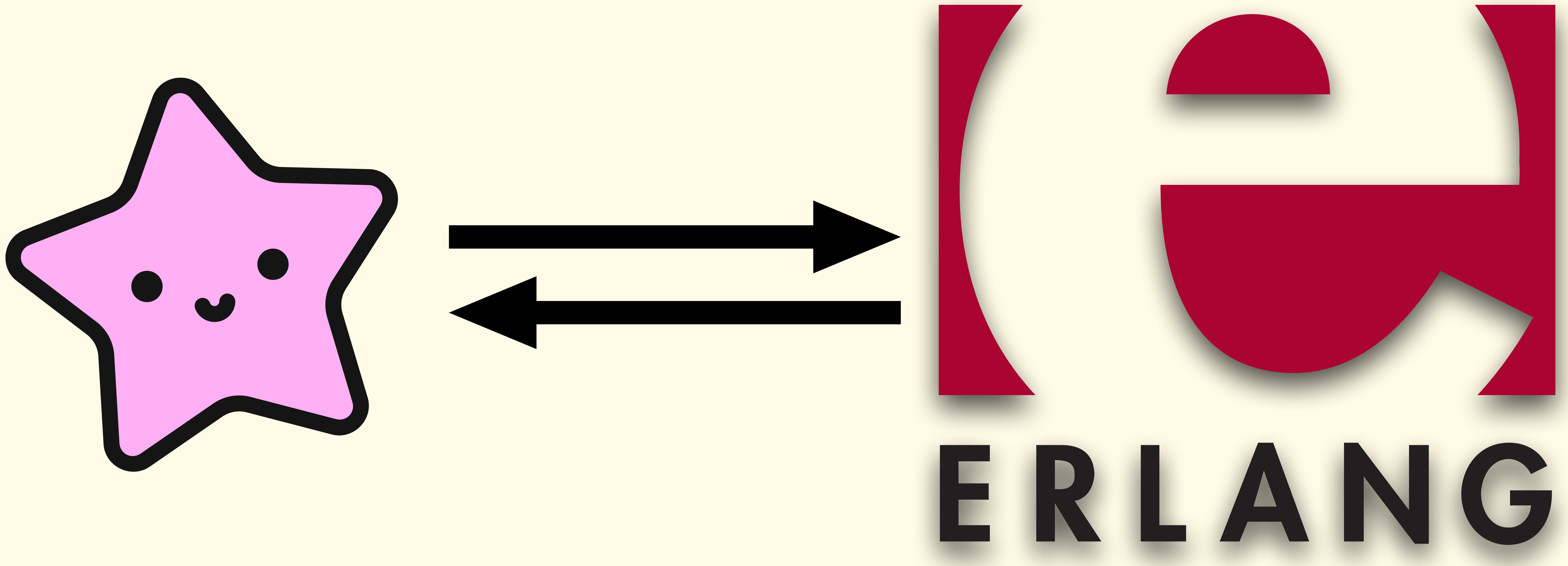
Let’s be boring!

Slow - Uninteresting - Considerate

ERLANG

ERLANG


ERLANG


ERLANG

Will this even work?

We know, we’ve been trying it this whole time!

Will we like it?

We know, we’ve been trying it this whole time!

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

shameless self-plug I made this :-)

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

```
import iv
```

I made this :-)

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

```
import iv

let array: iv.Array(Int) = iv.new() // []
```

I made this :-)

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

```gleam
import iv

let array: iv.Array(Int) = iv.new() // []
let array2 = array |> iv.append(0) |> iv.append(2) |> iv.prepend(3) // [3, 0, 2]
```

I made this :-)

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

```gleam
import iv

let array: iv.Array(Int) = iv.new() // []
let array2 = array |> iv.append(0) |> iv.append(2) |> iv.prepend(3) // [3, 0, 2]
let array3 = array2 |> iv.try_delete(1) // [3, 2]
```

I made this :-)

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

```gleam
import iv

let array: iv.Array(Int) = iv.new() // []
let array2 = array |> iv.append(0) |> iv.append(2) |> iv.prepend(3) // [3, 0, 2]
let array3 = array2 |> iv.try_delete(1) // [3, 2]
let array4 = iv.concat(array3, array2) // [3, 2, 3, 0, 2]
```

I made this :-)

iv

hex v1.3.2 hex docs

`iv` is a fast, general-purpose, persistent array structure for Gleam.

```gleam
import iv

let array: iv.Array(Int) = iv.new() // []
let array2 = array |> iv.append(0) |> iv.append(2) |> iv.prepend(3) // [3, 0, 2]
let array3 = array2 |> iv.try_delete(1) // [3, 2]
let array4 = iv.concat(array3, array2) // [3, 2, 3, 0, 2]
let last_two = iv.last_index_of(array4, 3) // Ok(2)
```

I made this :-)

But what about Elixir?

Gleam packages publish their compiled Erlang too!

```
# mix.exs
def deps do: [{:iv, “~> 1.3”}]
```

But what about Elixir?

Gleam packages publish their compiled Erlang too!

```elixir
# mix.exs
def deps do: [{:iv, “~> 1.3”}]

# app.ex
array = :iv.new() # []
array2 = array |> :iv.append(0) |> :iv.append(2) |> :iv.prepend(3) # [3, 0, 2]
array3 = array2 |> :iv.try_delete(1) # [3, 2]
array4 = :iv.concat(array3, array2) # [3, 2, 3, 0, 2]
lastTwo = :iv.last_index_of(array4, 3) # {:ok, 2}
```

But what about Erlang?

Rebar3 now supports the full constraint syntax!

```
% rebar.config
{deps, [{iv, “~> 1.3”}]}.

% app.erl
Array = iv:new(), % []
Array2 = iv:prepend(iv:append(iv:append(Array, 0), 2), 3), % [3, 0, 2]
Array3 = iv:try_delete(Array2, 1), % [3, 2]
Array4 = iv:concat(Array3, Array2), % [3, 2, 3, 0, 2]
LastTwo = iv:last_index_of(Array4, 3). % {ok, 2}
```

We’re done!

Really, that’s it!

Using Gleam packages

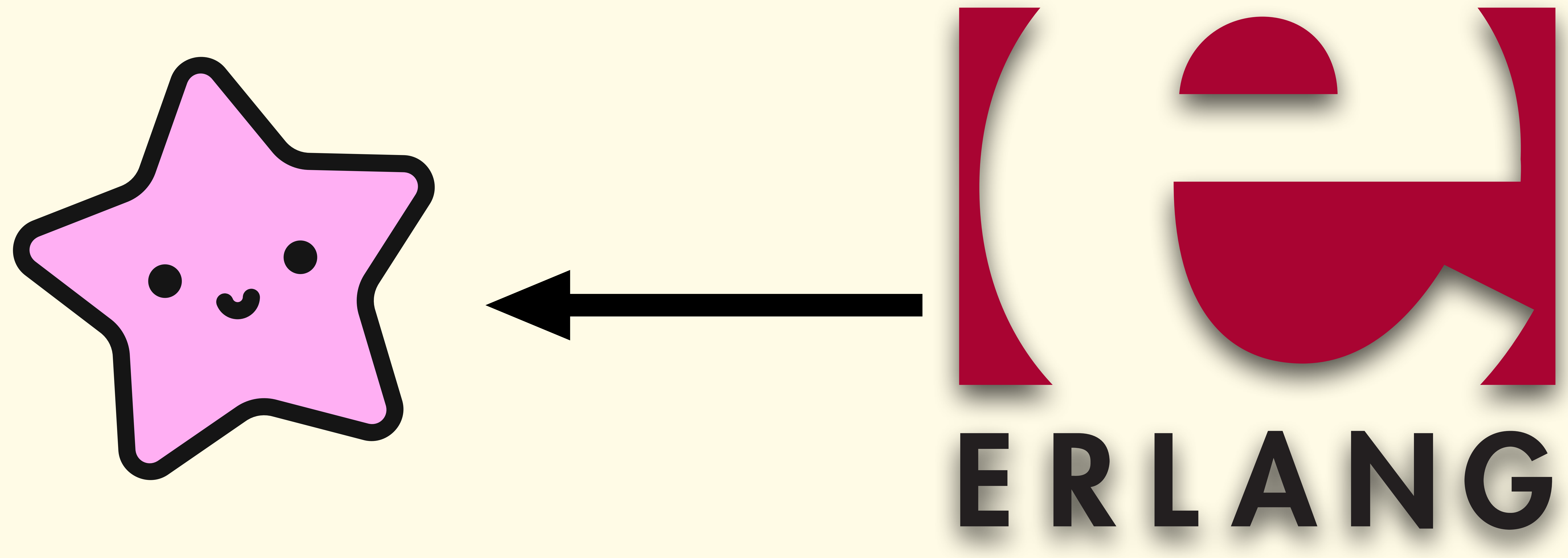
- Gleam compiles to Erlang & Javascript **source** code
- …the source code is part of the hex package
- …Mix & Rebar3 know how to compile Erlang
- …no Gleam required!

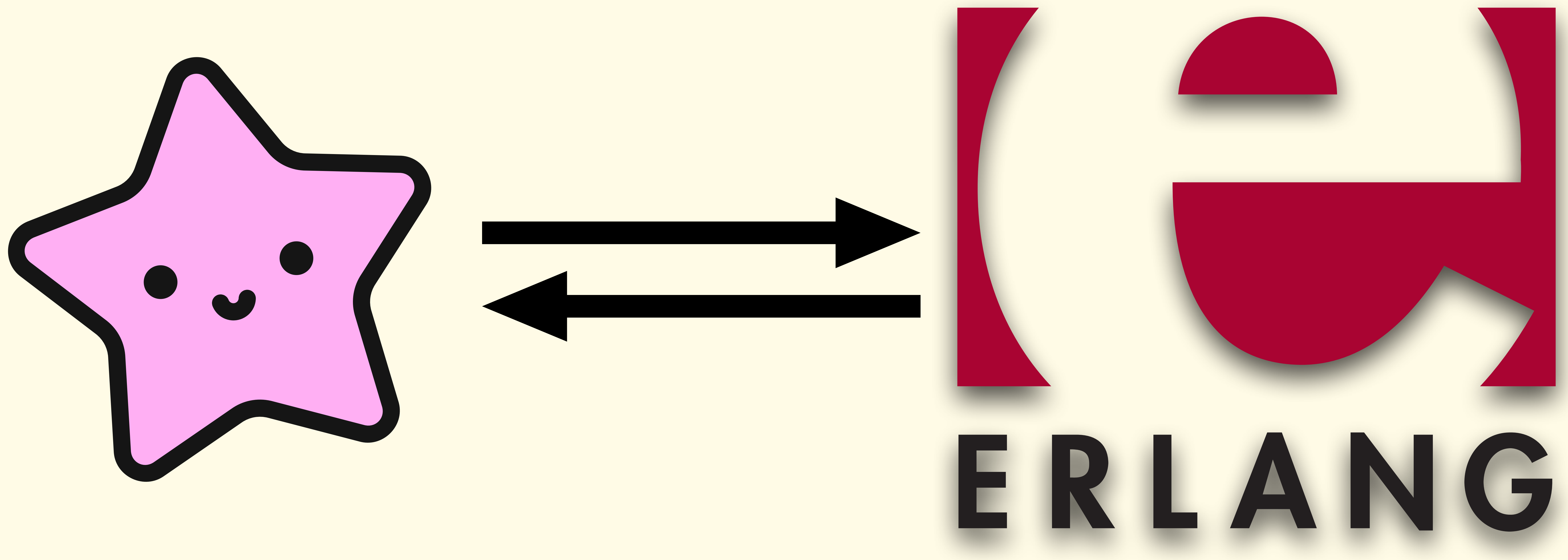
Path dependencies with Mix

Hopefully coming to an Elixir distribution near you soon!

```
defp deps do
  [
    # ...
    {:gleam_lib, path: “./gleam_lib”}
  ]
end
```

- Full Gleam support for Mix (!!)
- This is so amazing thank you Papipo!! 💜


ERLANG


ERLANG

Calling Erlang and Elixir

Calling Erlang and Elixir

```
// gleam.toml
decimal = “≥ 2.3.0 and < 3.0.0"
```

Calling Erlang and Elixir

```
// gleam.toml
decimal = “≥ 2.3.0 and < 3.0.0"

// app.gleam
pub type Decimal
```

Calling Erlang and Elixir

```
// gleam.toml
decimal = “≥ 2.3.0 and < 3.0.0"

// app.gleam
pub type Decimal


fn from_float(value: Float) -> Decimal
```

Calling Erlang and Elixir

```
// gleam.toml
decimal = “≥ 2.3.0 and < 3.0.0"

// app.gleam
pub type Decimal

@external(erlang, “Elixir.Decimal”, “from_float”)
fn from_float(value: Float) -> Decimal
```

Calling Erlang and Elixir

```
// gleam.toml
decimal = “≥ 2.3.0 and < 3.0.0"

// app.gleam
pub type Decimal

@external(erlang, “Elixir.Decimal”, “from_float”)
fn from_float(value: Float) -> Decimal

@external(erlang, “Elixir.Decimal”, “add”)
fn add(a: Decimal, b: Decimal) -> Decimal

@external(erlang, “Elixir.Decimal”, “to_string”)
fn to_string(value: Decimal) -> String
```

Calling Erlang and Elixir

```
// gleam.toml
decimal = “≥ 2.3.0 and < 3.0.0"

// app.gleam
pub type Decimal

@external(erlang, “Elixir.Decimal”, “from_float”)
fn from_float(value: Float) -> Decimal

@external(erlang, “Elixir.Decimal”, “add”)
fn add(a: Decimal, b: Decimal) -> Decimal

@external(erlang, “Elixir.Decimal”, “to_string”)
fn to_string(value: Decimal) -> String

pub fn main() {
  echo from_float(3.0) |> add(from_float(1.2)) |> to_string // “4.2”
}
```

Calling Erlang and Elixir

```
// gleam.t
decimal =

// app.gl
pub type

@extern
fn from

@exter
fn add

@external(er
fn to_string(value: Decim

pub fn main() {
  echo from_float(3.0) |> add(from_float(1.2)) |> to_stri
}
```

Gleam

News Community Sponsor

Packages Docs Code

Externals guide

Using code written in other languages from Gleam

Design Gleam APIs first

Don’t make bindings

- Erlang/Elixir/JS APIs are highly dynamic and don’t translate well
- FFI is an implementation detail for the Gleam API
- Pick the largest boundary that makes sense

Design Gleam APIs first

Don’t make bindings

- Erlang/Elixir/JS APIs are highly dynamic and don’t translate well
- FFI is an implementation detail for the Gleam API
- Pick the largest boundary that makes sense

```erlang
-module(mala_ffi).

bag_get(Bag, Key) ->
    try
        Items1 = ets:lookup(Bag, Key),
        {ok, lists:map(fun(Elem) -> element(2, Elem) end, Items1)}
    catch error:badarg -> {error, nil}
    end.
```

Use Natural Boundaries

Don’t make bindings[2]

- HTTP Handlers
- gen_servers / Actors
- OTP Applications
- Bounded Contexts
- …

JS

🤔 Works the same?

- Gleam modules compile to JavaScript modules
- Standard JavaScript tooling works: vite, bun, rollup, …
- `@external` to call JavaScript from Gleam

Web components

Make your own HTML elements!

Web components

Make your own HTML elements!

```html
<my-accordion name=“faq”>
  <my-accordion-item>
    Lorem ipsem dolor…
  </my-accordion-item>
  <my-accordion-item selected>
    Sed ut perspiciatis
  </my-accordion-item>
</my-accordion>
```

Web components

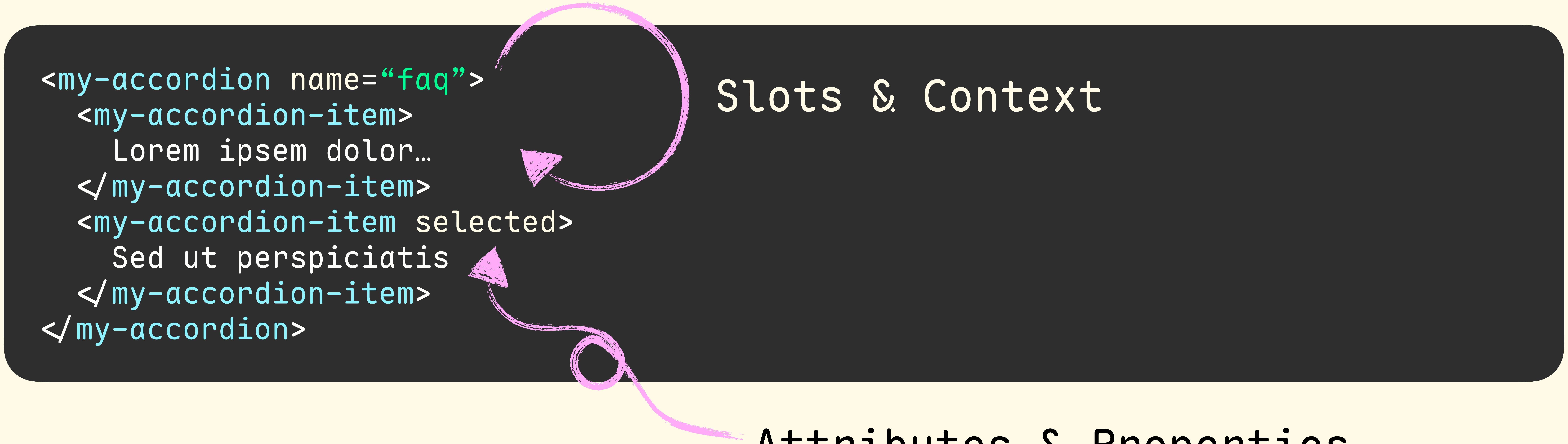
Make your own HTML elements!

```
<my-accordion name="faq">
  <my-accordion-item>
    Lorem ipsem dolor…
  </my-accordion-item>
  <my-accordion-item selected>
    Sed ut perspiciatis
  </my-accordion-item>
</my-accordion>
```

Attributes & Properties

Web components

Make your own HTML elements!



```
<my-accordion name="faq">
  <my-accordion-item>
    Lorem ipsem dolor…
  </my-accordion-item>
  <my-accordion-item selected>
    Sed ut perspiciatis
  </my-accordion-item>
</my-accordion>
```

Attributes & Properties

Web components

Make your own HTML elements!

onchange="…"

```
<my-accordion name="faq">
  <my-accordion-item>
    Lorem ipsem dolor…
  </my-accordion-item>
  <my-accordion-item selected>
    Sed ut perspiciatis
  </my-accordion-item>
</my-accordion>
```

Slots & Context

Attributes & Properties

Web components in Lustre

- Custom Attributes, Properties, and Events

Web components in Lustre

- Custom Attributes, Properties, and Events
- Form-associated custom elements

Web components in Lustre

- Custom Attributes, Properties, and Events
- Form-associated custom elements
- Custom CSS states

Web components in Lustre

- Custom Attributes, Properties, and Events
- Form-associated custom elements
- Custom CSS states
- Context protocol proposal

Web components in Lustre

- Custom Attributes, Properties, and Events
- Form-associated custom elements
- Custom CSS states
- Context protocol proposal

```
pub fn main() {
  let app = lustre.application(init:, update:, view:)
  lustre.start(app, “#app”, Nil)
}
```

Web components in Lustre

- Custom Attributes, Properties, and Events
- Form-associated custom elements
- Custom CSS states
- Context protocol proposal

```
pub fn register() {
  let app = lustre.component(init:, update:, view:, [])
  lustre.register(app, ”my-accordion”)
}
```

Phoenix integration

```
# mix.exs
defp aliases do
  # ...
  “assets.build”: [“cmd --cd client gleam build”, ...]
end
```

Phoenix integration

```
# mix.exs
defp aliases do
  # ...
  “assets.build”: [“cmd --cd client gleam build”, ...]
end
```

```
// app.js
import { register } from “./client/build/dev/javascript/client/client.mjs”;
register();
```

Phoenix integration

```
# mix.exs
defp aliases do
  # ...
  “assets.build”: [“cmd --cd client gleam build”, ...]
end
```

```
// app.js
import { register } from “./client/build/dev/javascript/client/client.mjs”;
register();
```

```
// page.html.heex
<my-accordion>...</my-accordion>
```

Adopting Gleam the boring way

Code BEAM Europe 2025 — Yoshie Reusch