# Types Without Borders

Dillon Kearns
incrementalelm.com

bit.ly/typeswithoutborders

# JavaScript

```
let json = Json.parse(stringFromServer);
```

– `Json is not defined`

– **We meant** `JSON`

– **Don't know until runtime.**

bit.ly/typeswithoutborders

# Elm Compiler

- Knows a lot!
- Make impossible states impossible
- If it compiles it works! ™

# Custom `Msg` Types

```
type Msg
   = Increment



update msg model =
   case msg of
      Increment ->
         (model + 1, Cmd.none)
```

bit.ly/typeswithoutborders

# Add to `Msg`

```
type Msg
  = Increment
  | Decrement

update msg model =
  case msg of
    Increment ->
      (model + 1, Cmd.none)
```

— **This `case` does not have branches for all possibilities...**

# Single Source of Truth

```
type Msg
  = Increment
  | Decrement

update msg model =
  case msg of
    Increment ->
      (model + 1, Cmd.none)

    Decrement ->
      (model - 1, Cmd.none)
```

bit.ly/typeswithoutborders

bit.ly/typeswithoutborders

# APIs in Elm

```elm
type Character
    = Human { name : String, homePlanet : String }
    | Droid { name : String, primaryFunction : String }

decoder : Decode.Decoder Character
decoder =
    Decode.field "type" Decode.string
        |> Decode.andThen
            (\characterType ->
                case characterType of
                    "human" -> humanDecoder
                    "droid" -> droidDecoder
                    _ -> Decode.fail "Invalid Character")
```

bit.ly/typeswithoutborders

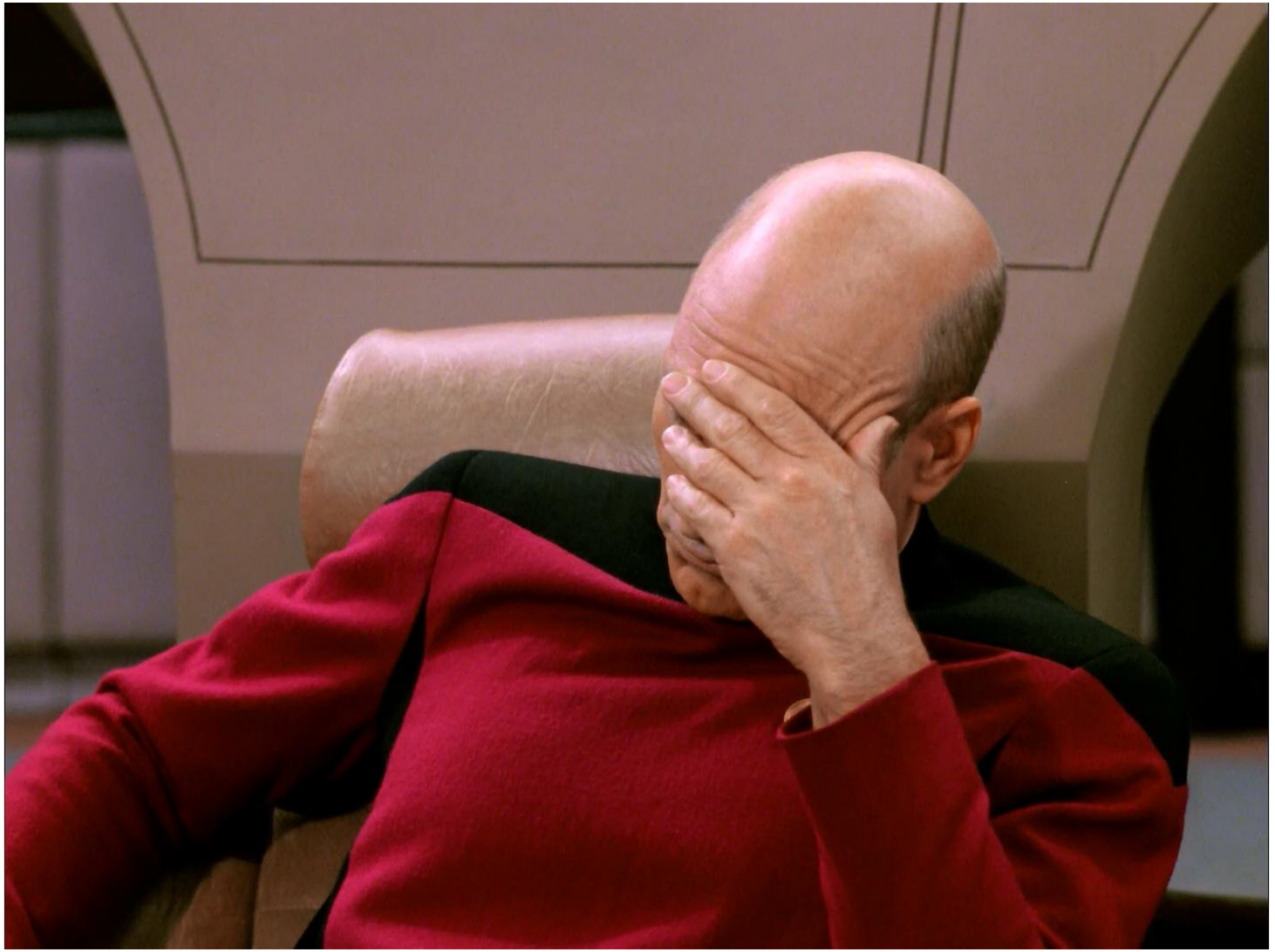
# Nested Decoders

```elm
humanDecoder : Decode.Decoder Character
humanDecoder =
    Decode.map2 HumanAttributes
        (Decode.field "name" Decode.string)
        (Decode.field "homePlanet" Decode.string)
        |> Decode.map Human


droidDecoder : Decode.Decoder Character
droidDecoder =
    Decode.map2 DroidAttributes
        (Decode.field "name" Decode.string)
        (Decode.field "primaryFunction" Decode.string)
        |> Decode.map Droid
```

bit.ly/typeswithoutborders

# `BadPayload`

**Expecting an object with a field named `name` but instead got...**



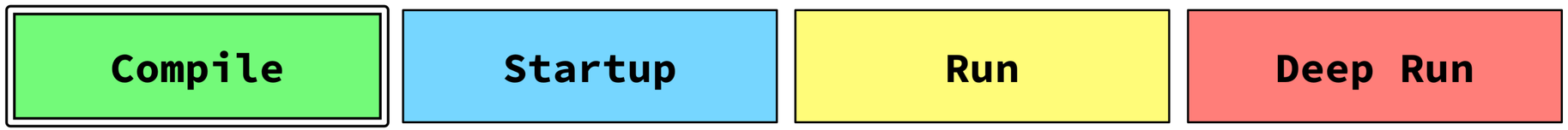
bit.ly/typeswithoutborders

# JS Implicit Assumptions

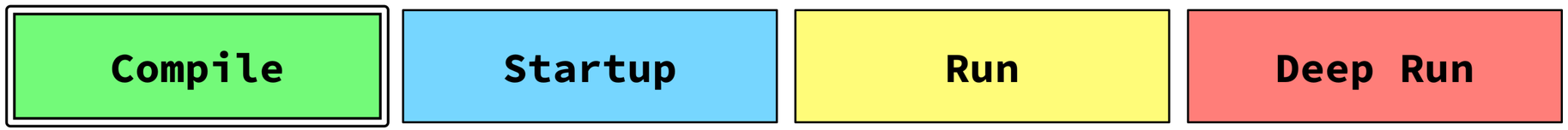
```
greet(jsonResponse.name);
```

- `"Hello [object Object]!"`
- **Explicit assumptions are nice (e.g.** `Result`**)**
- **No assumptions is best**

# When Can I Know?



# When Can I Know?




`Msg` `Flags` `Decoder` `greet`

# Uncertainty

bit.ly/typeswithoutborders

# Uncertainty

## 1. Unavoidable

bit.ly/typeswithoutborders

# Uncertainty

## 1. Unavoidable
## 2. Avoidable

bit.ly/typeswithoutborders

# Unavoidable Uncertainty

```
type Error
    = BadStatus (Response String)
    | Timeout
    | NetworkError
```

— **Can't guarantee WiFi** 📶

— **Representing unavoidable uncertainty** 👍

# Avoidable Uncertainty

```
type Error
    = BadStatus (Response String)
    | Timeout
    | NetworkError
    | BadUrl String
```

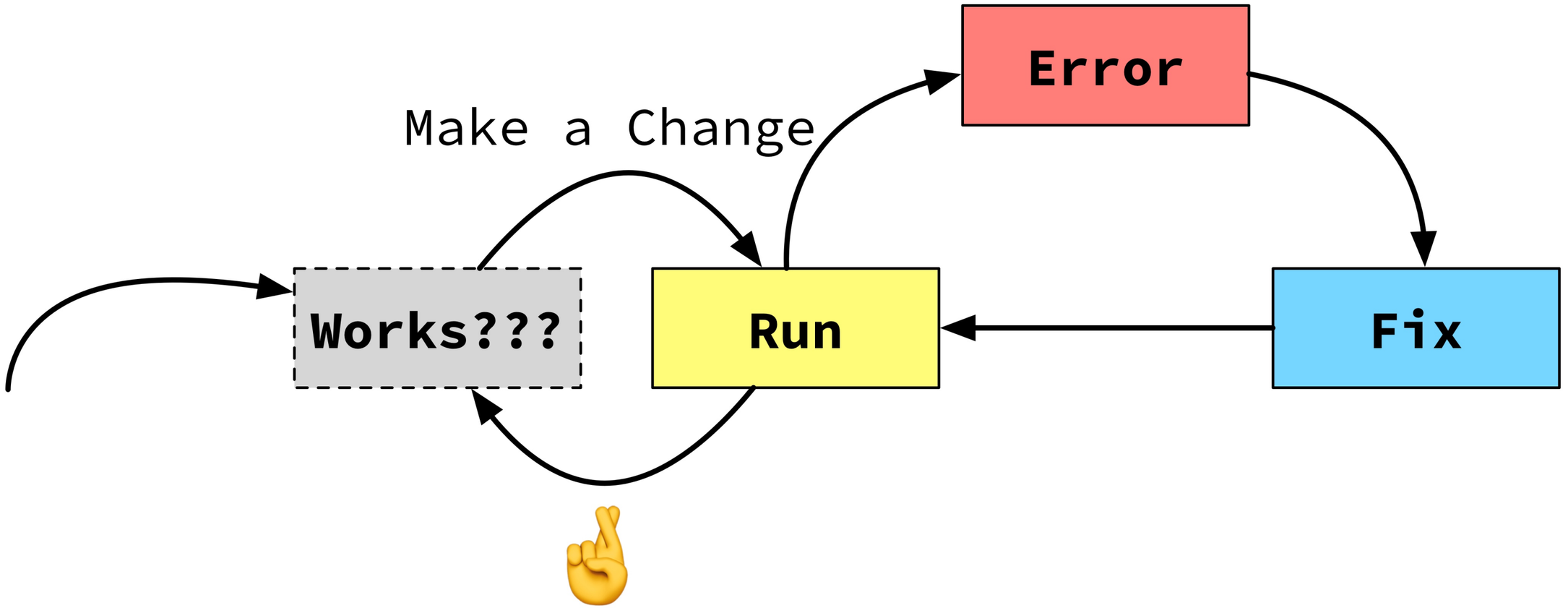
— **Could make this impossible.**

# Avoidable Uncertainty

```
type Error
    = BadStatus (Response String)
    | Timeout
    | NetworkError
    | BadUrl String
    | BadPayload String (Response String)
```

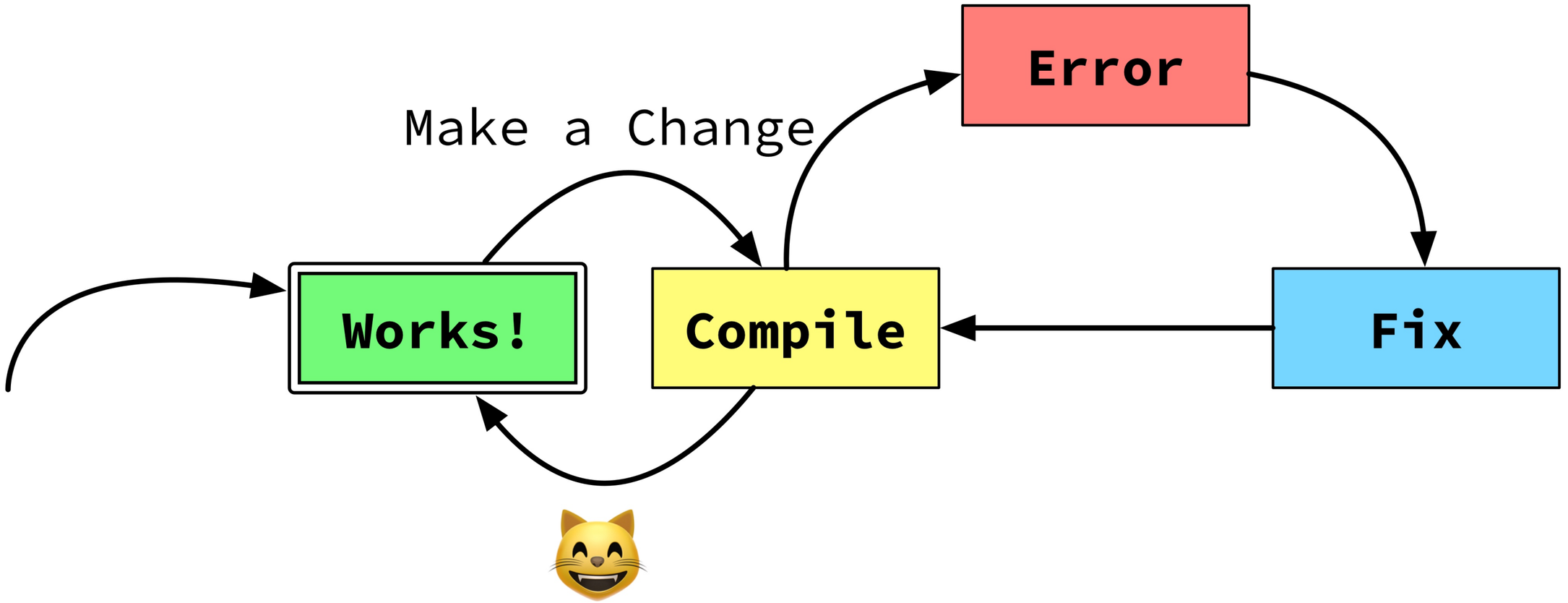
— Two sources of truth

# Avoidable Uncertainty




bit.ly/typeswithoutborders

# With Certainty

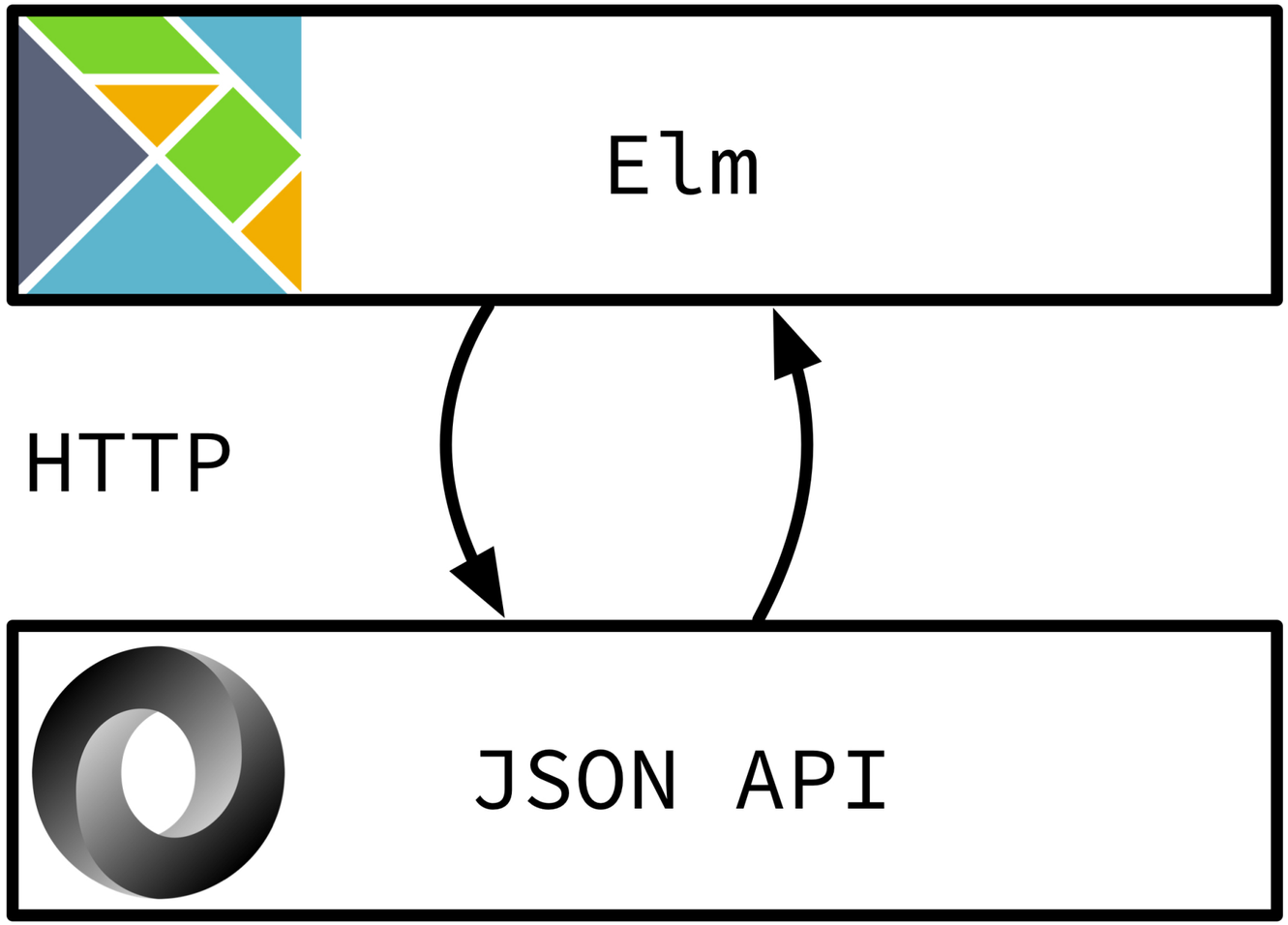



bit.ly/typeswithoutborders

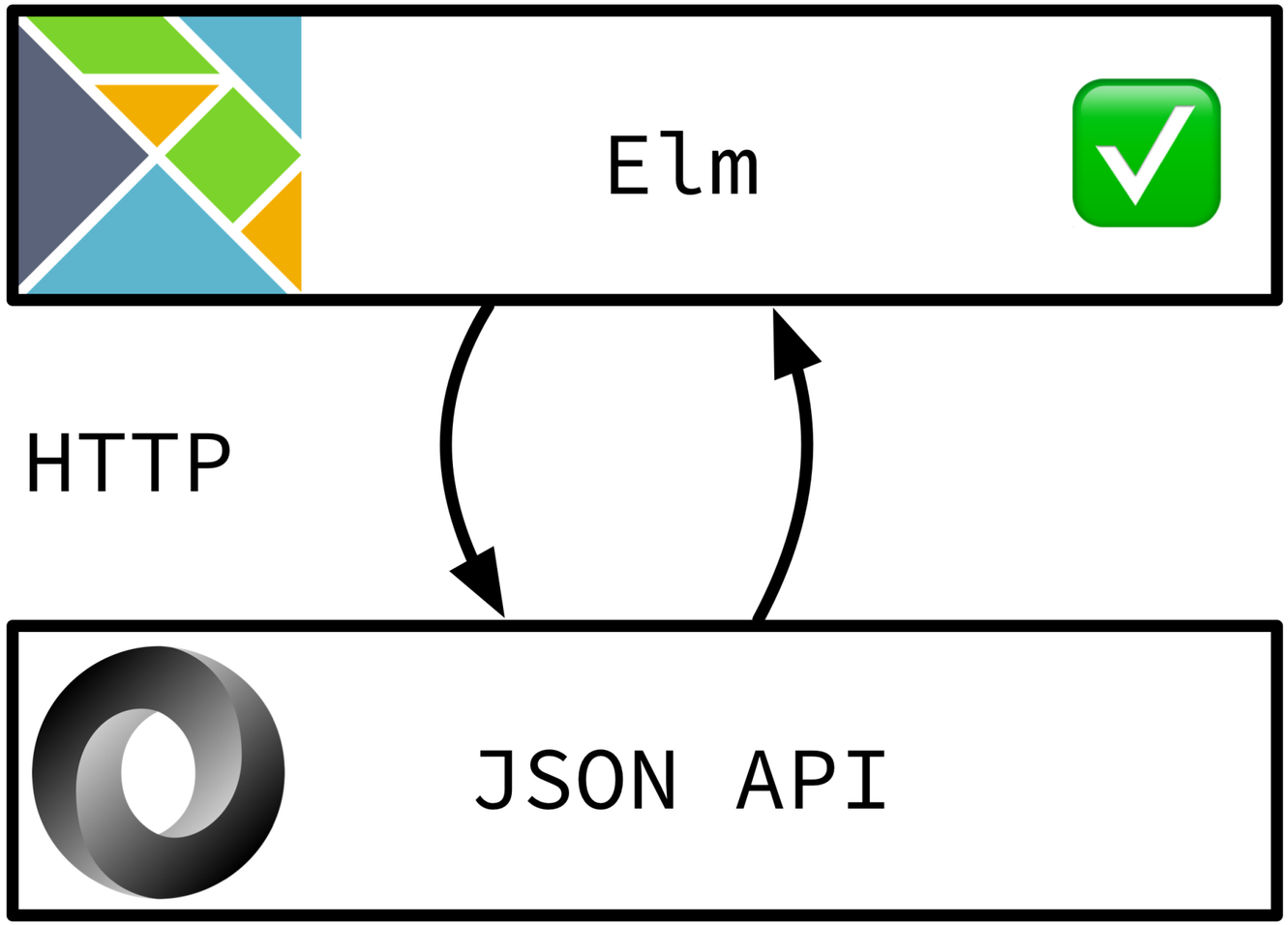
# Single Source of Truth

```
type Msg
  = Increment
  | Decrement

update msg model =
  case msg of
    Increment ->
      (model + 1, Cmd.none)

    -- You forgot a case!
```

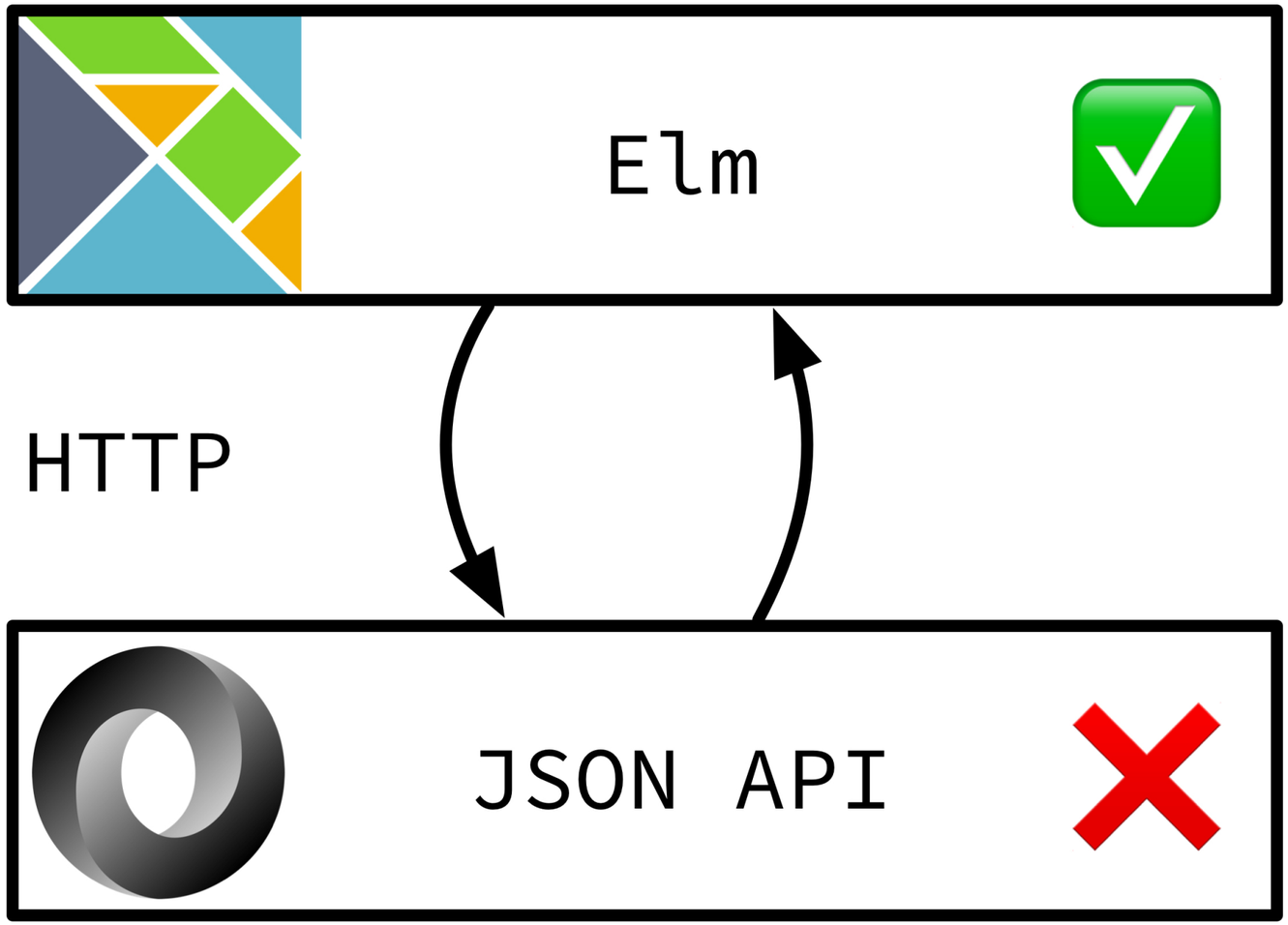
bit.ly/typeswithoutborders



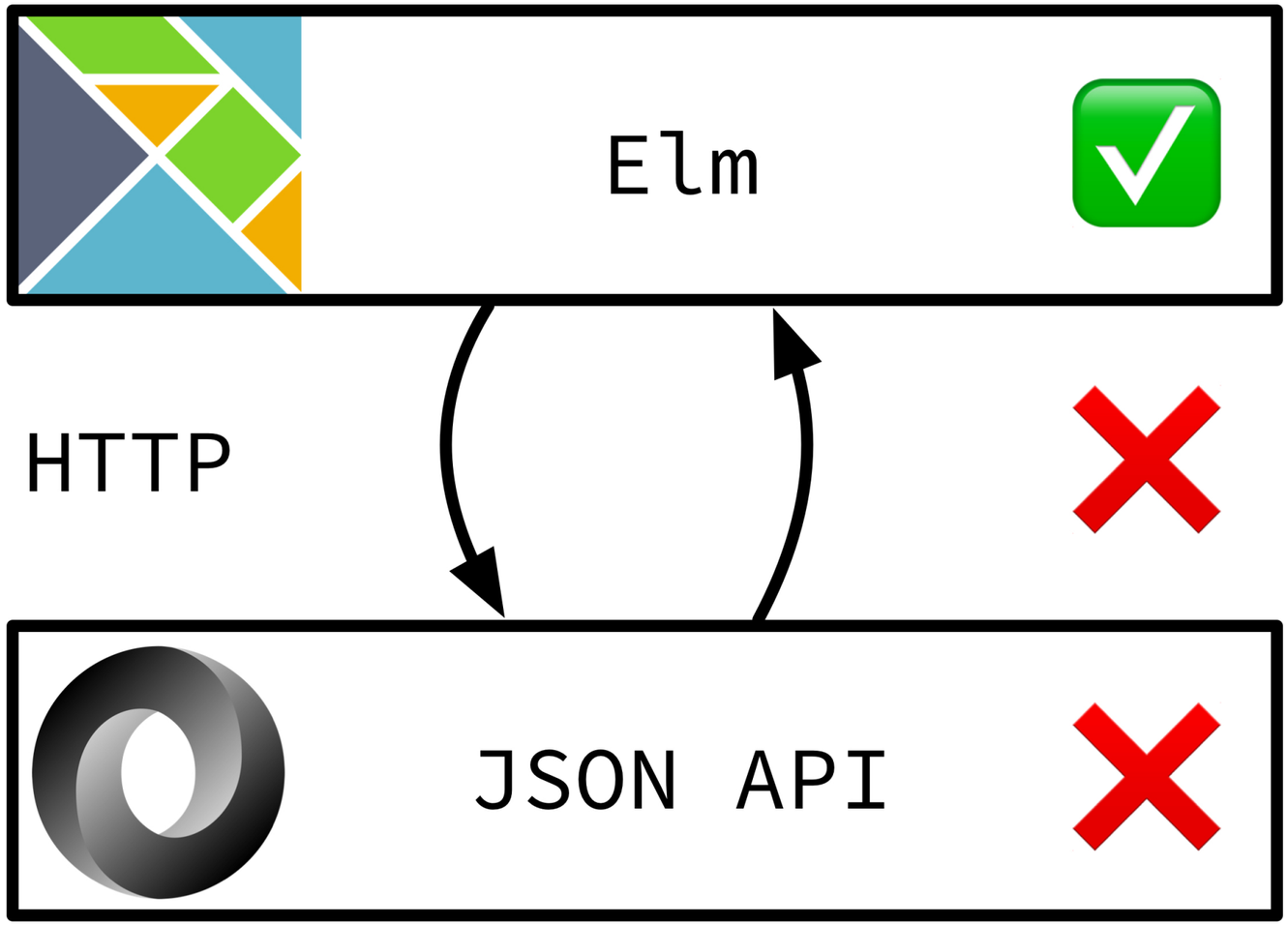

bit.ly/typeswithoutborders



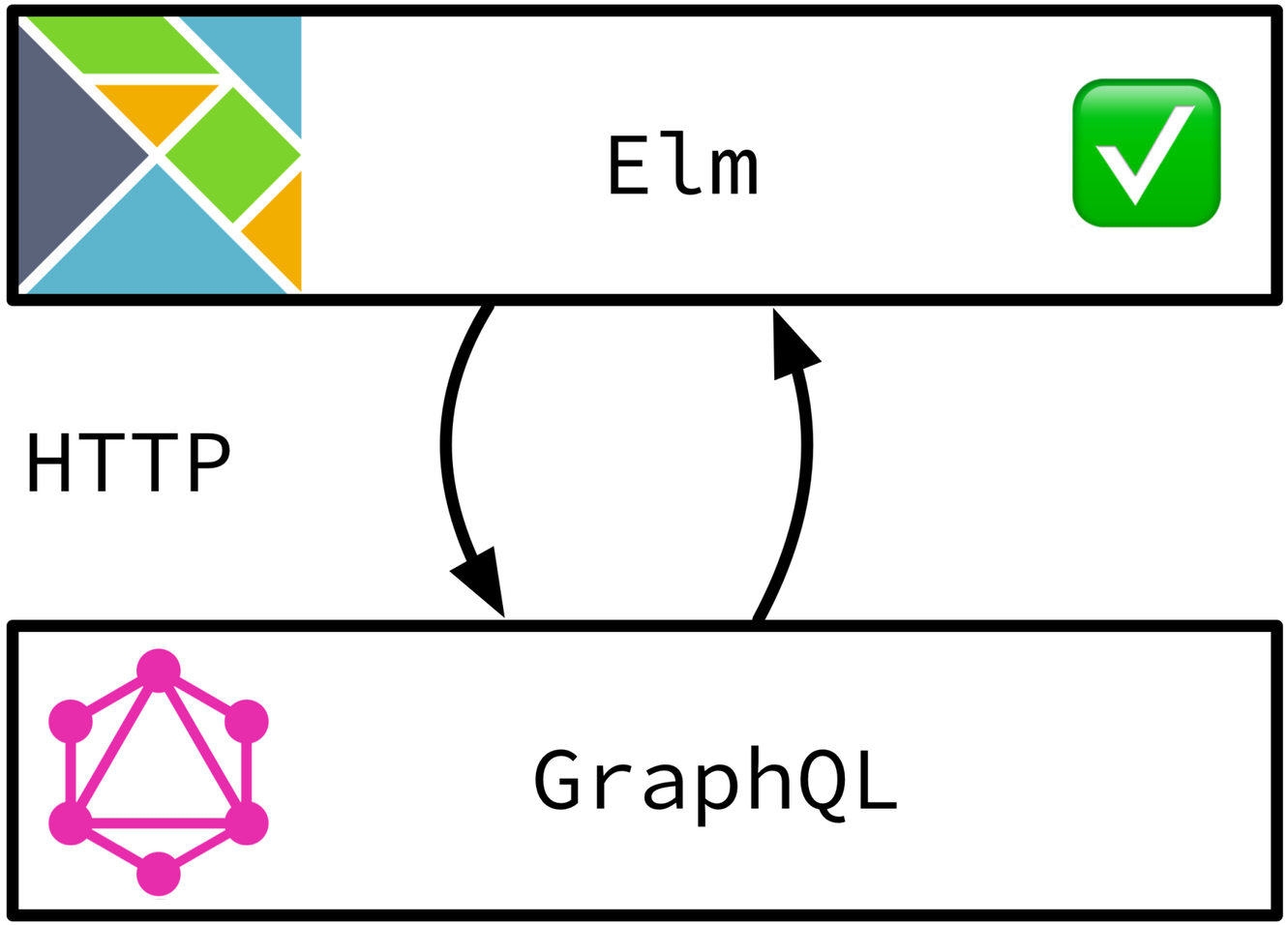

bit.ly/typeswithoutborders



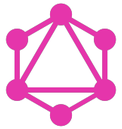

bit.ly/typeswithoutborders




bit.ly/typeswithoutborders

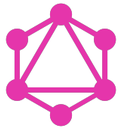



bit.ly/typeswithoutborders



# GraphQL

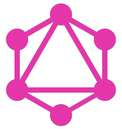
— Strong guarantees
— Enforces contract
— Robust type system



# GraphQL Enums

```
enum SortOrder {
  ASCENDING
  DESCENDING
}
```

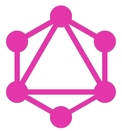
bit.ly/typeswithoutborders



# GraphQL Objects

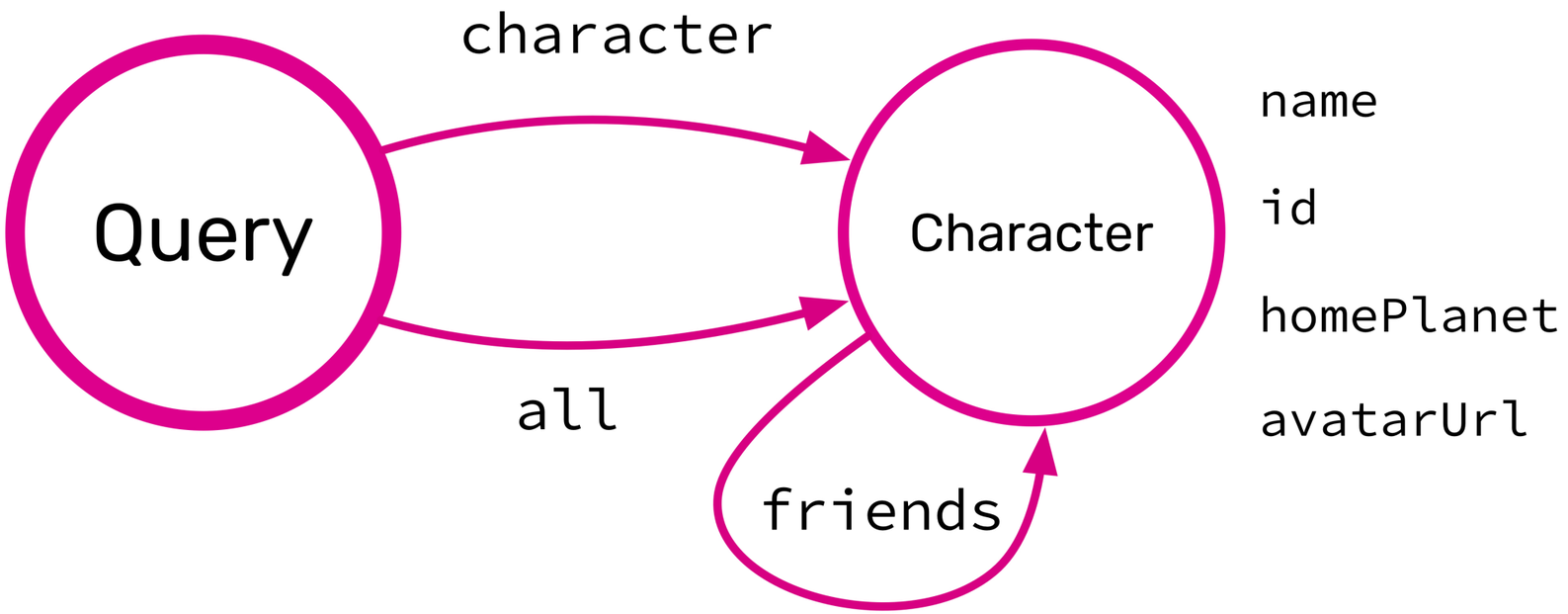
```
type Character {
  name: String!
  homePlanet: String
  avatarUrl: String!
  friends: [Character!]!
}
```

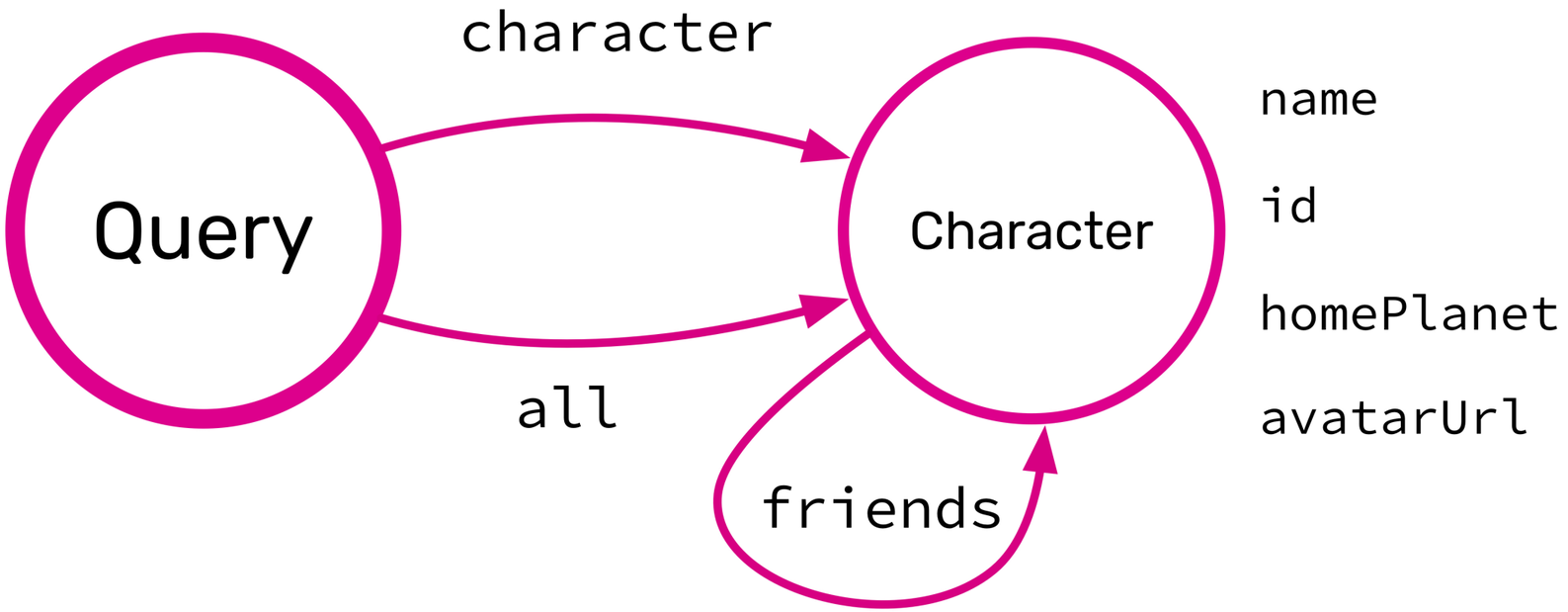
bit.ly/typeswithoutborders



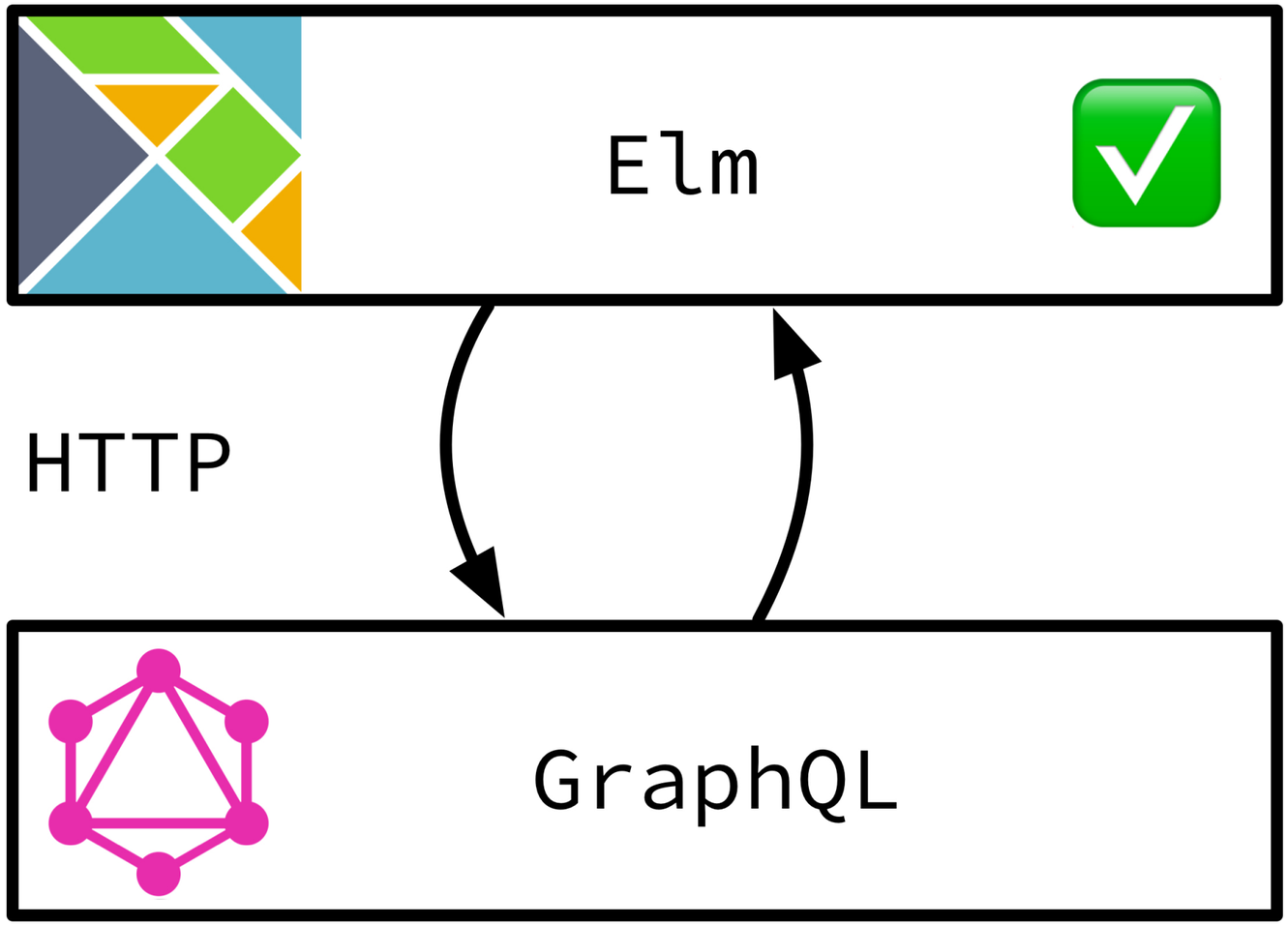
# GraphQL Arguments

```
type Query {
  character(id: ID!): Character
  all: [Character!]!
}
```

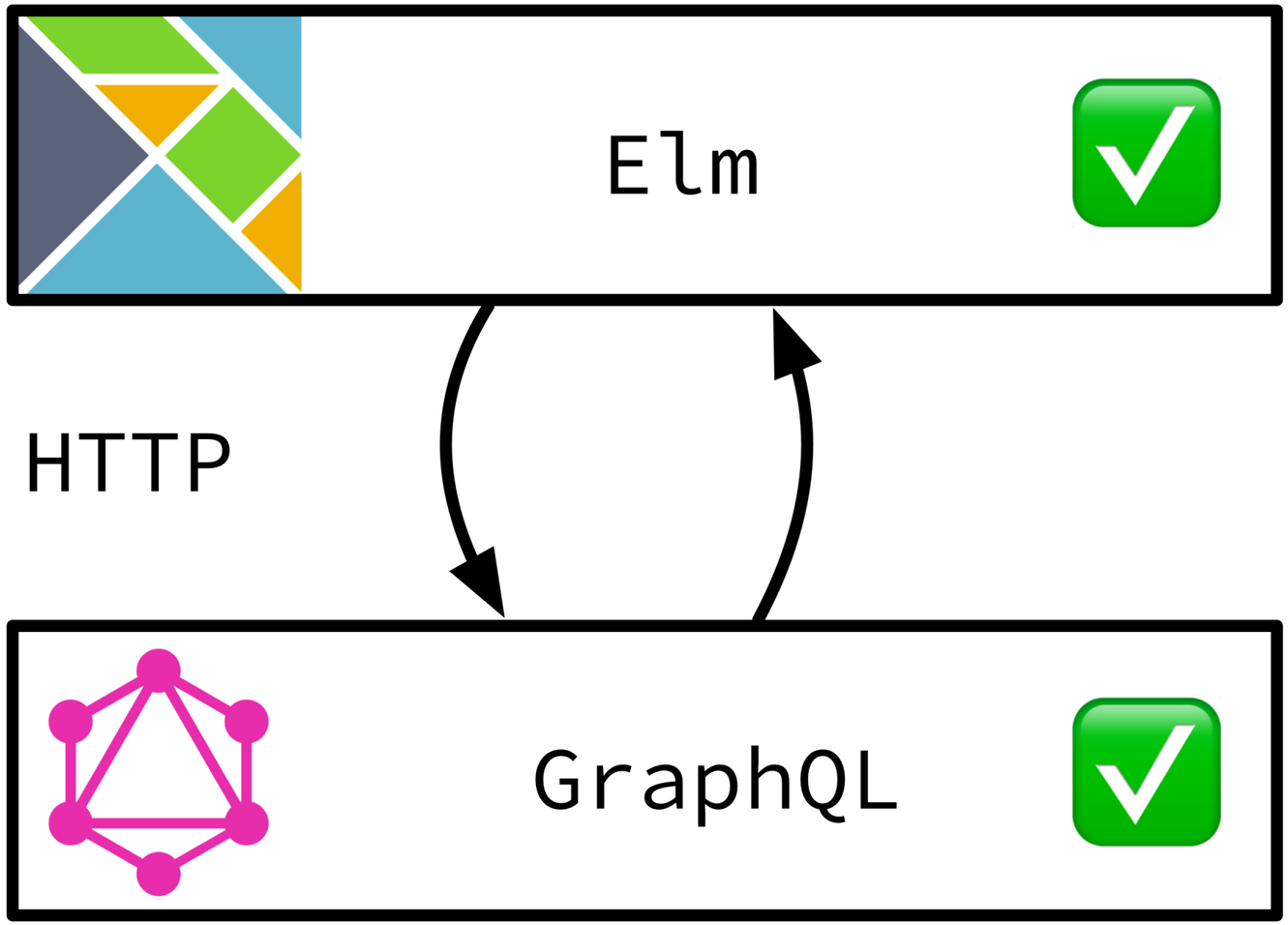

character
Query
Character
name
id
homePlanet
avatarUrl
all
friends




```
query {
  all {
    name
    avatarUrl
    homePlanet
    friends { name }
  }
}
```




bit.ly/typeswithoutborders




bit.ly/typeswithoutborders

# `dillonkearns/elm-graphql`

- **Generates a "hardcoded library"**
- `Json.Decode` **is an implementation detail**

# `dillonkearns/elm-graphql` **Guarantees**

- **Correct request**
- **Well-typed**
- **Decodes correctly**

# Json.Decode.Pipeline

```
type alias CharacterInfo =
  { name : String
  , avatarUrl : String
  , homePlanet : Maybe String
  }

characterDecoder =
  Decode.succeed CharacterInfo
    |> required "name" string
    |> required "avatarUrl" string
    |> required "homePlanet" (nullable string)
```

bit.ly/typeswithoutborders

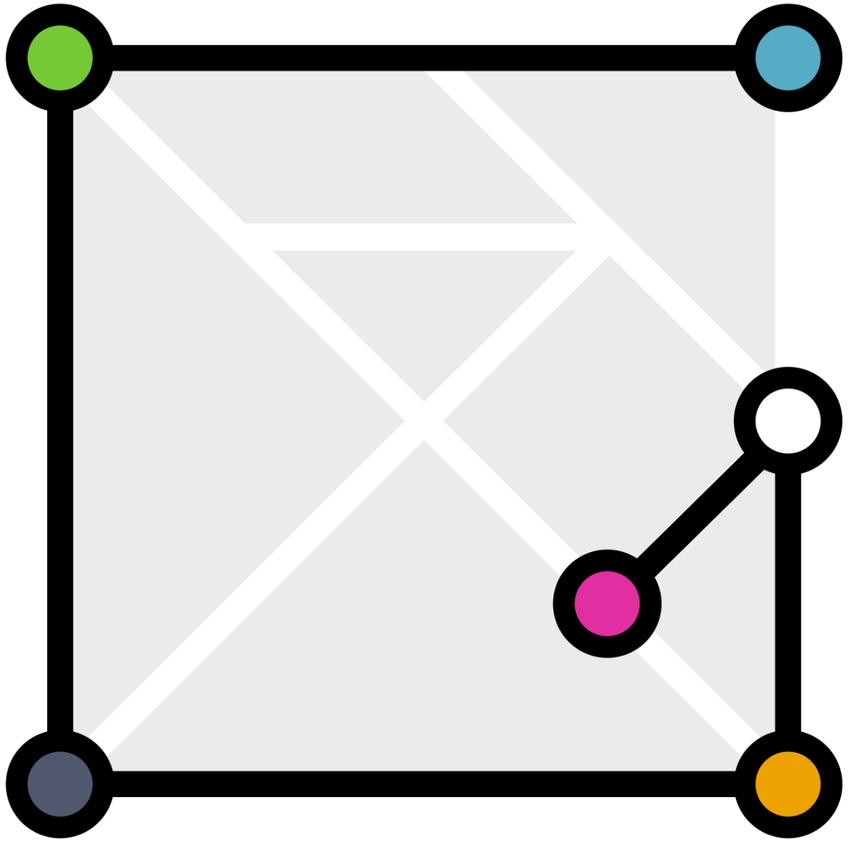
# dillonkearns/elm-graphql

```
type alias CharacterInfo =
  { name : String
  , avatarUrl : String
  , homePlanet : Maybe String
  }

characterSelection =
  Character.selection CharacterInfo
    |> with Character.name
    |> with Character.avatarUrl
    |> with Character.homePlanet
```

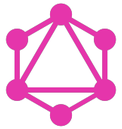
bit.ly/typeswithoutborders

# Demo!

`dillonkearns/elm-graphql`

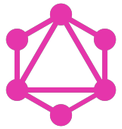


bit.ly/typeswithoutborders



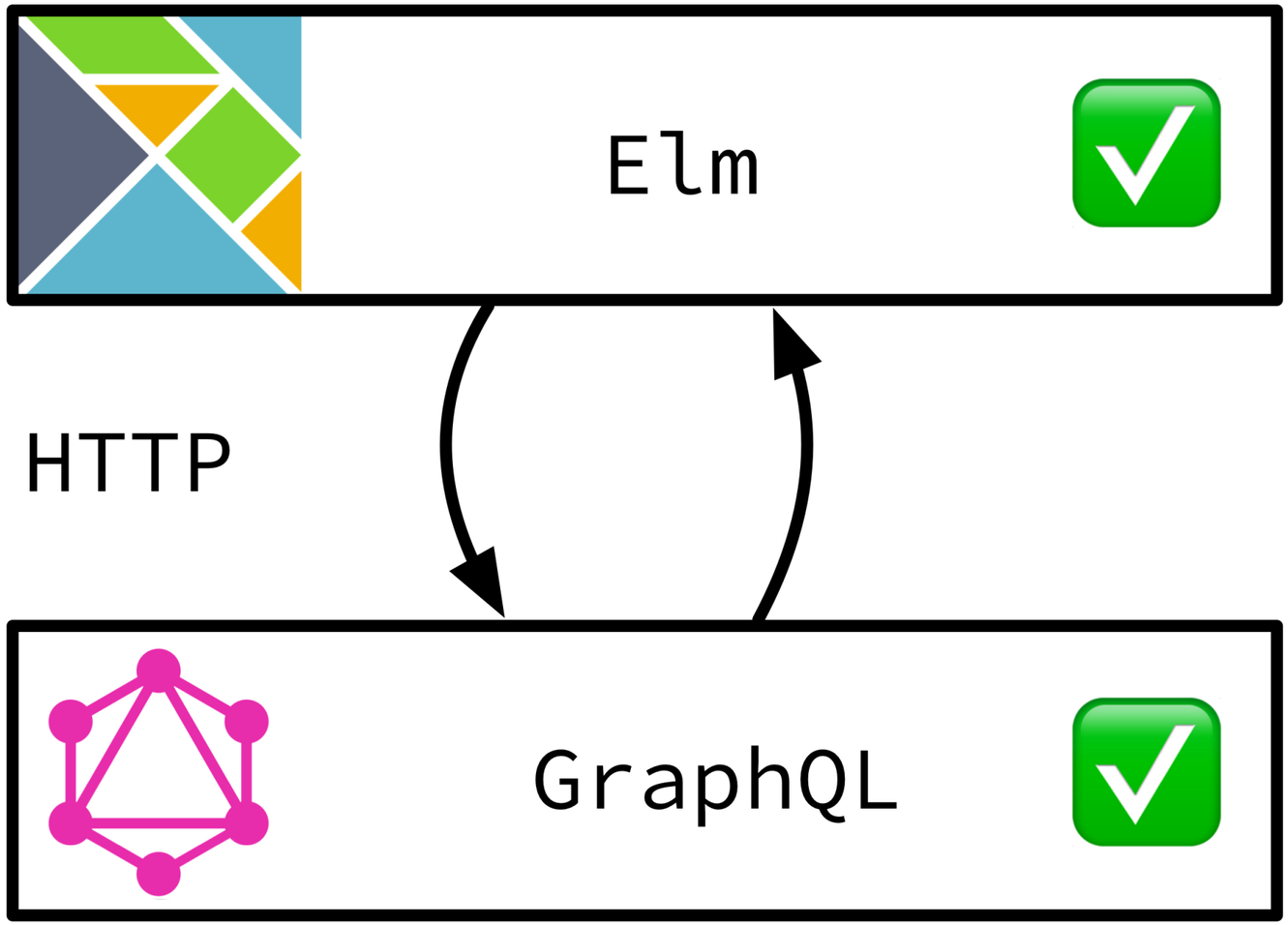
# Breaking Change

```
type Character {
  name: String!
  avatarUrl: String!
  homePlanet: String
}
```

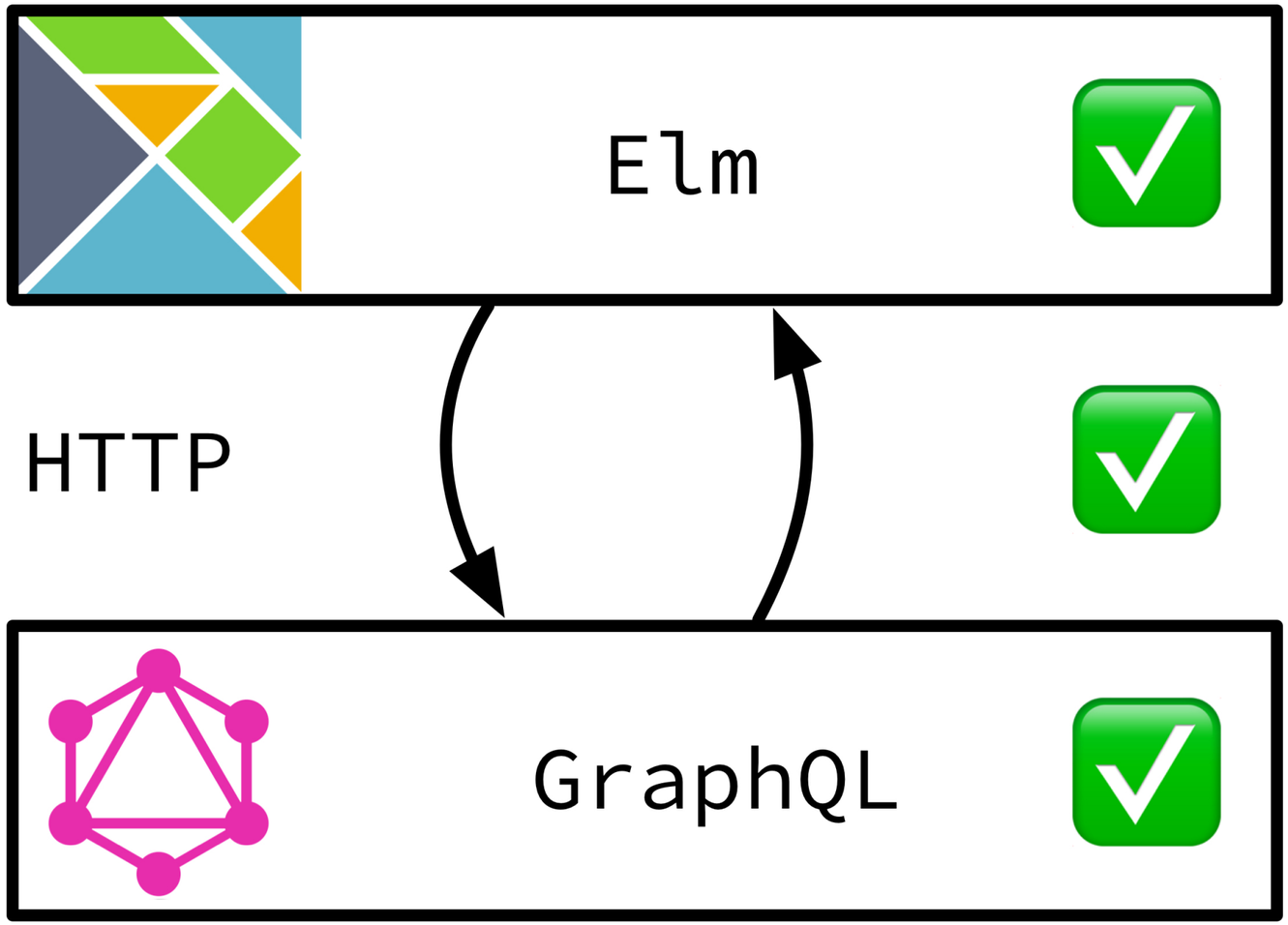


# Breaking Change

```
type Character {
  name: String!
  -- avatarUrl: String!
  homePlanet: String
}
```

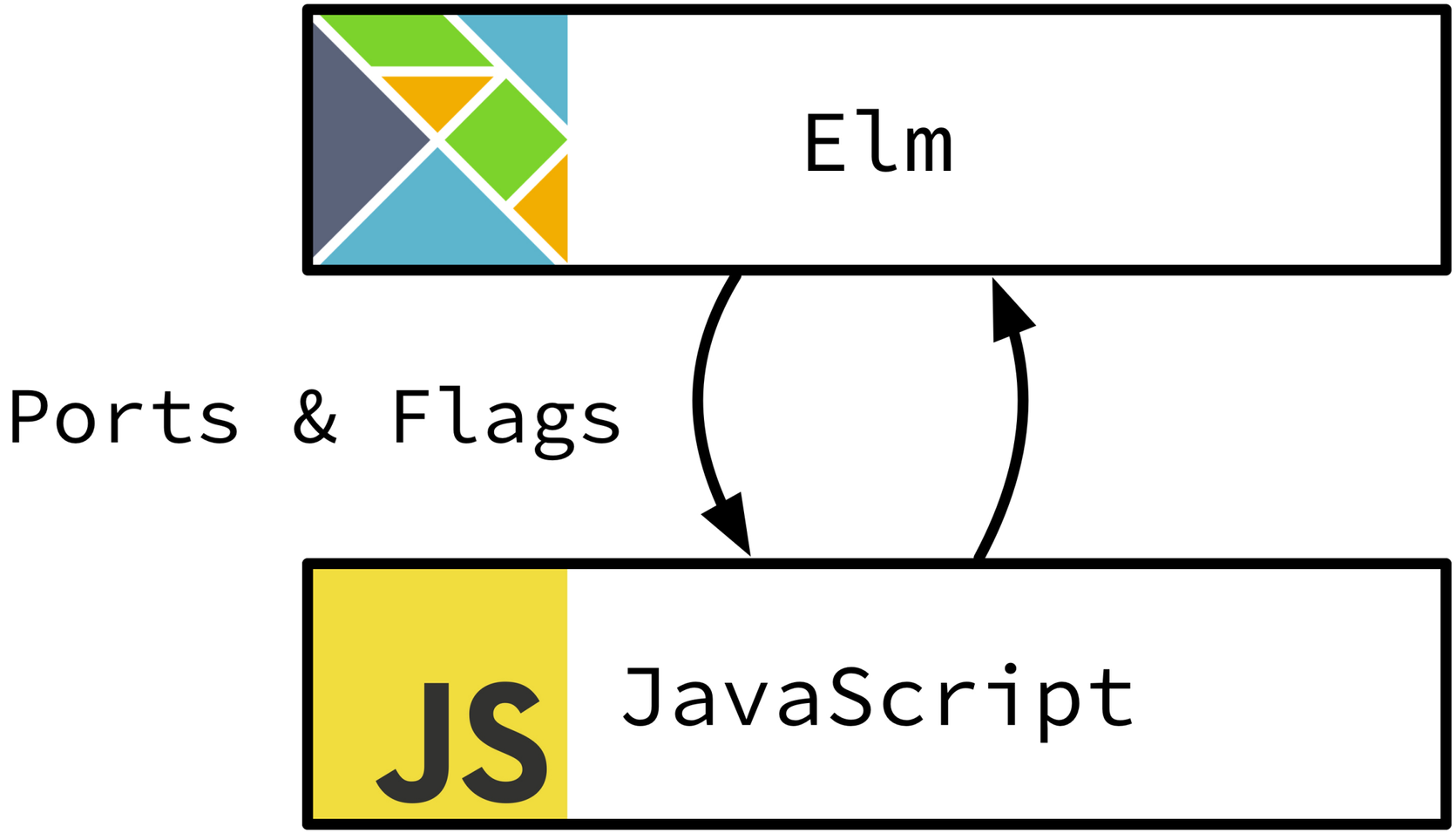



bit.ly/typeswithoutborders

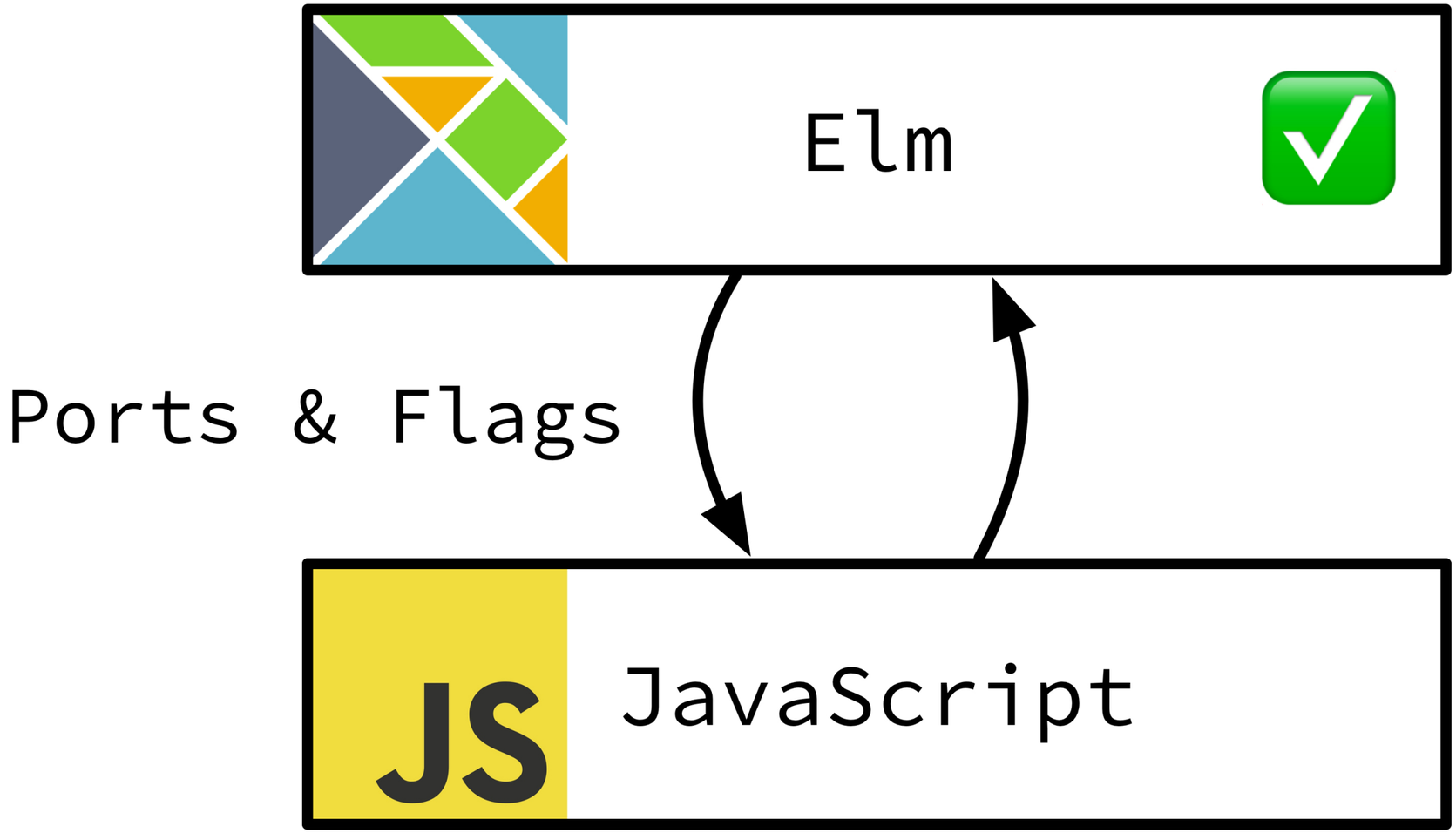



bit.ly/typeswithoutborders

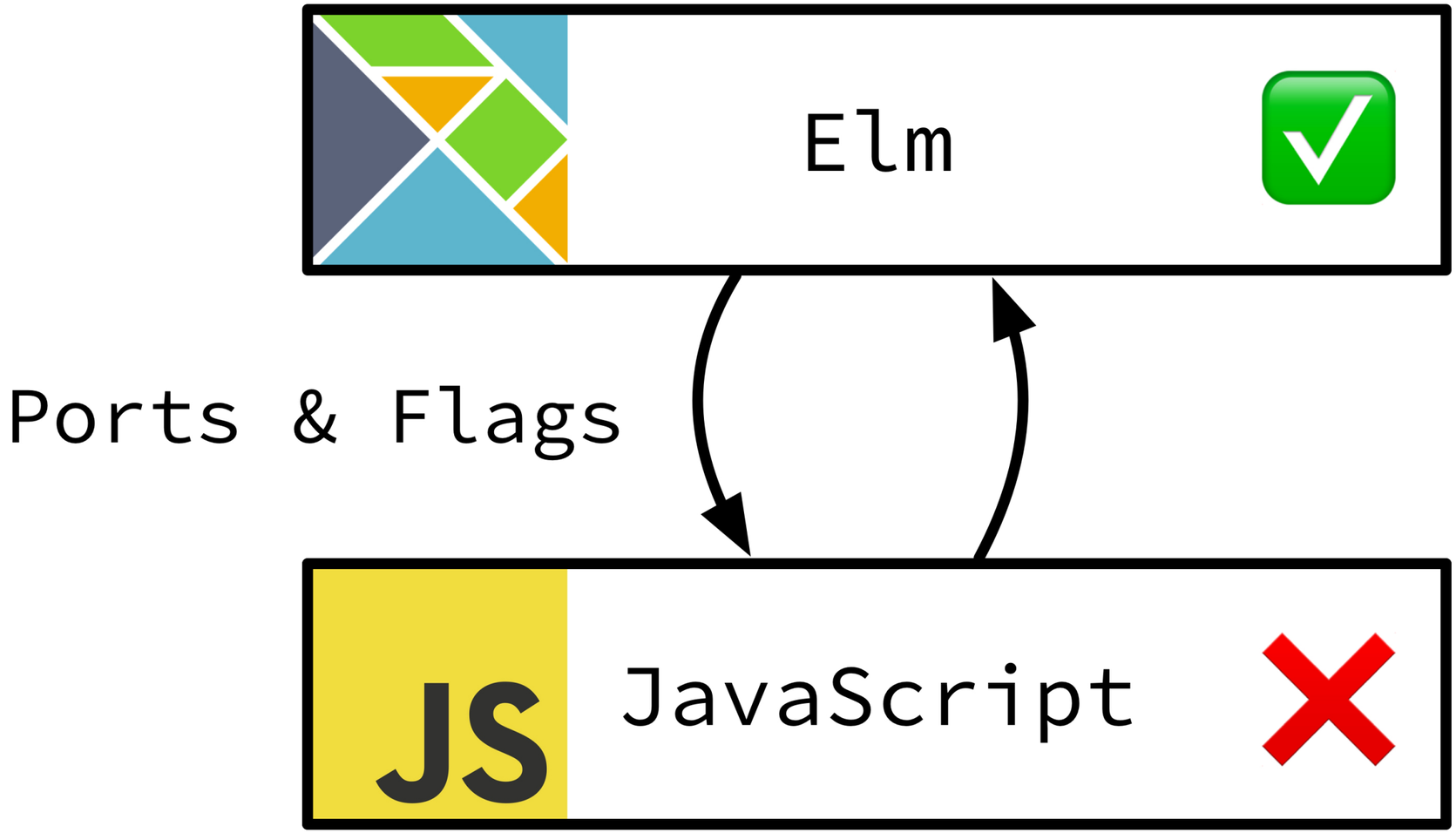

bit.ly/typeswithoutborders



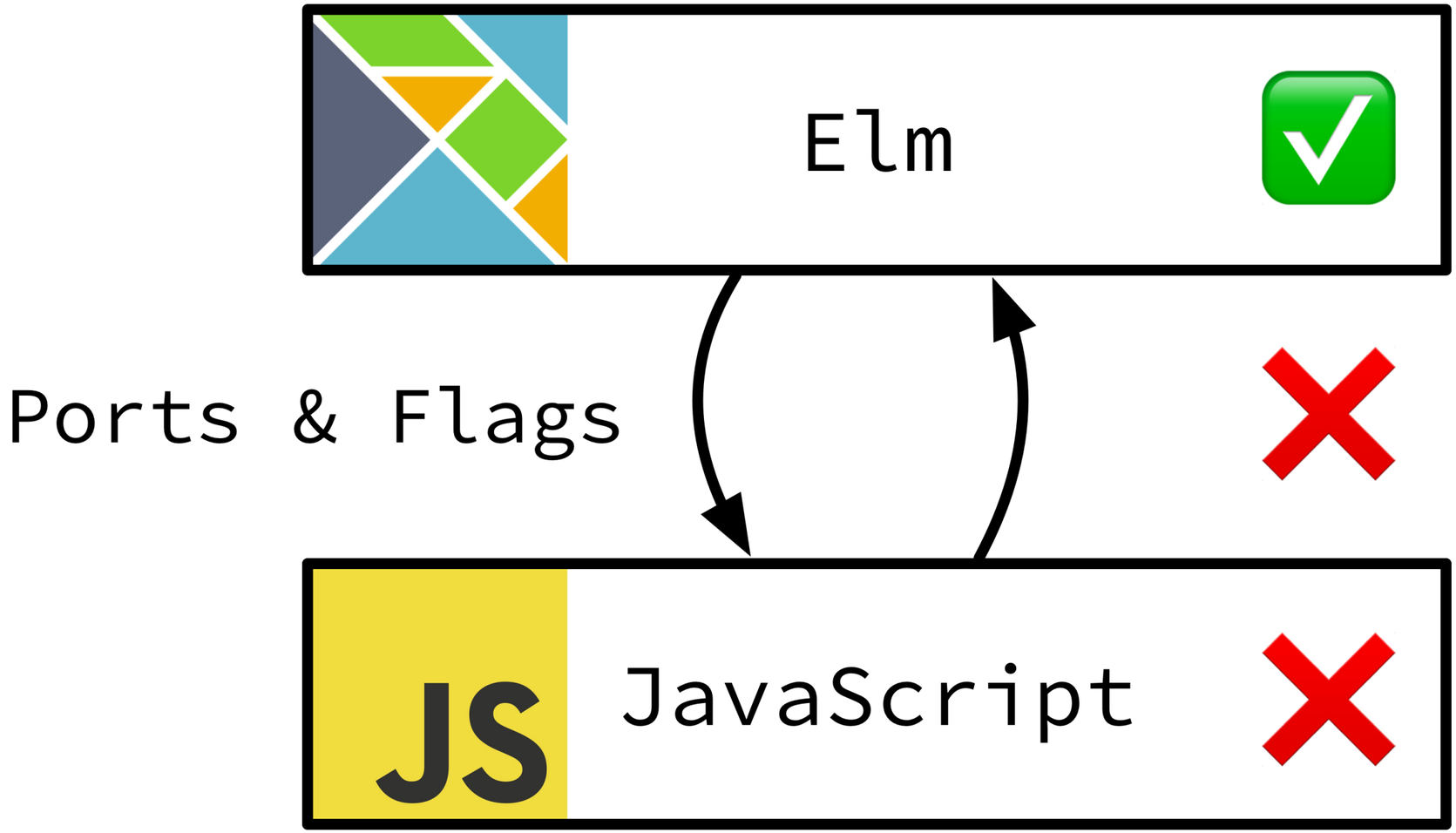

bit.ly/typeswithoutborders



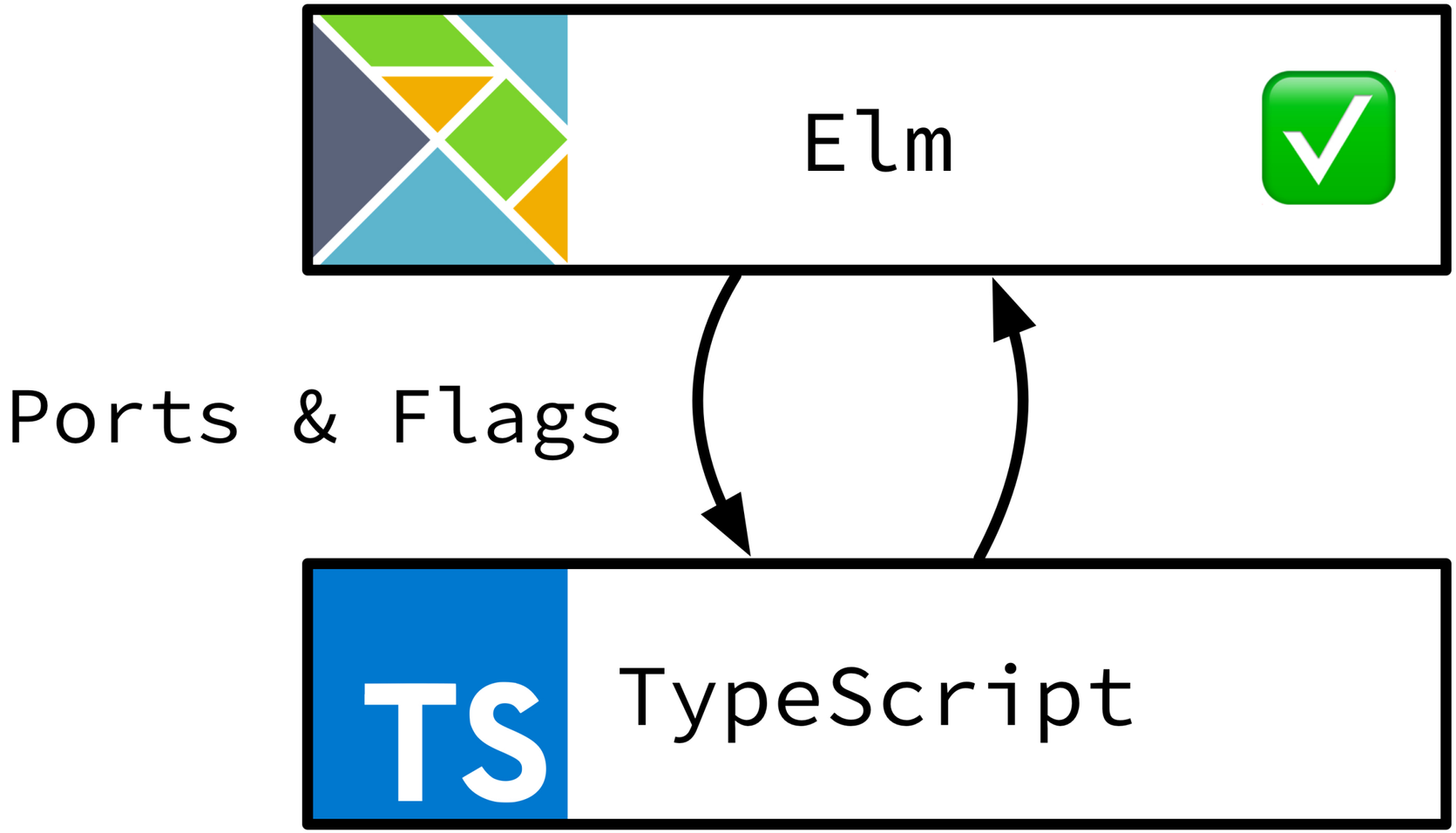

bit.ly/typeswithoutborders




bit.ly/typeswithoutborders




bit.ly/typeswithoutborders




bit.ly/typeswithoutborders

# TypeScript

- Superset of JavaScript
- Editor integration, auto-complete
- Union Types
- Just add a `tsconfig.json`!

bit.ly/typeswithoutborders

```
npm install @types/<npm-library>
```

— **Often included (`electron`, `moment`)**

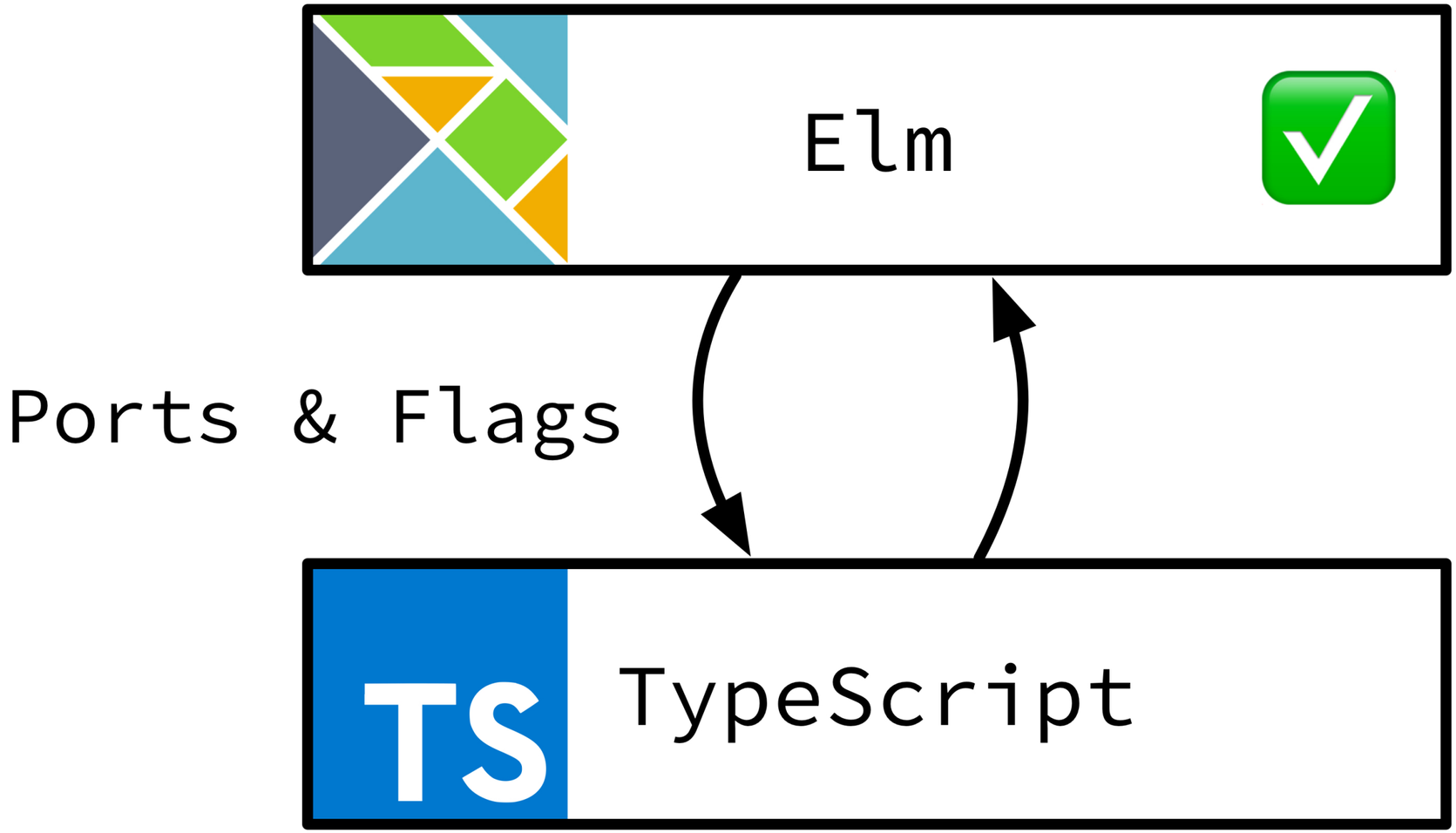
# Why Not Functional Compile-to-JS?

REASON

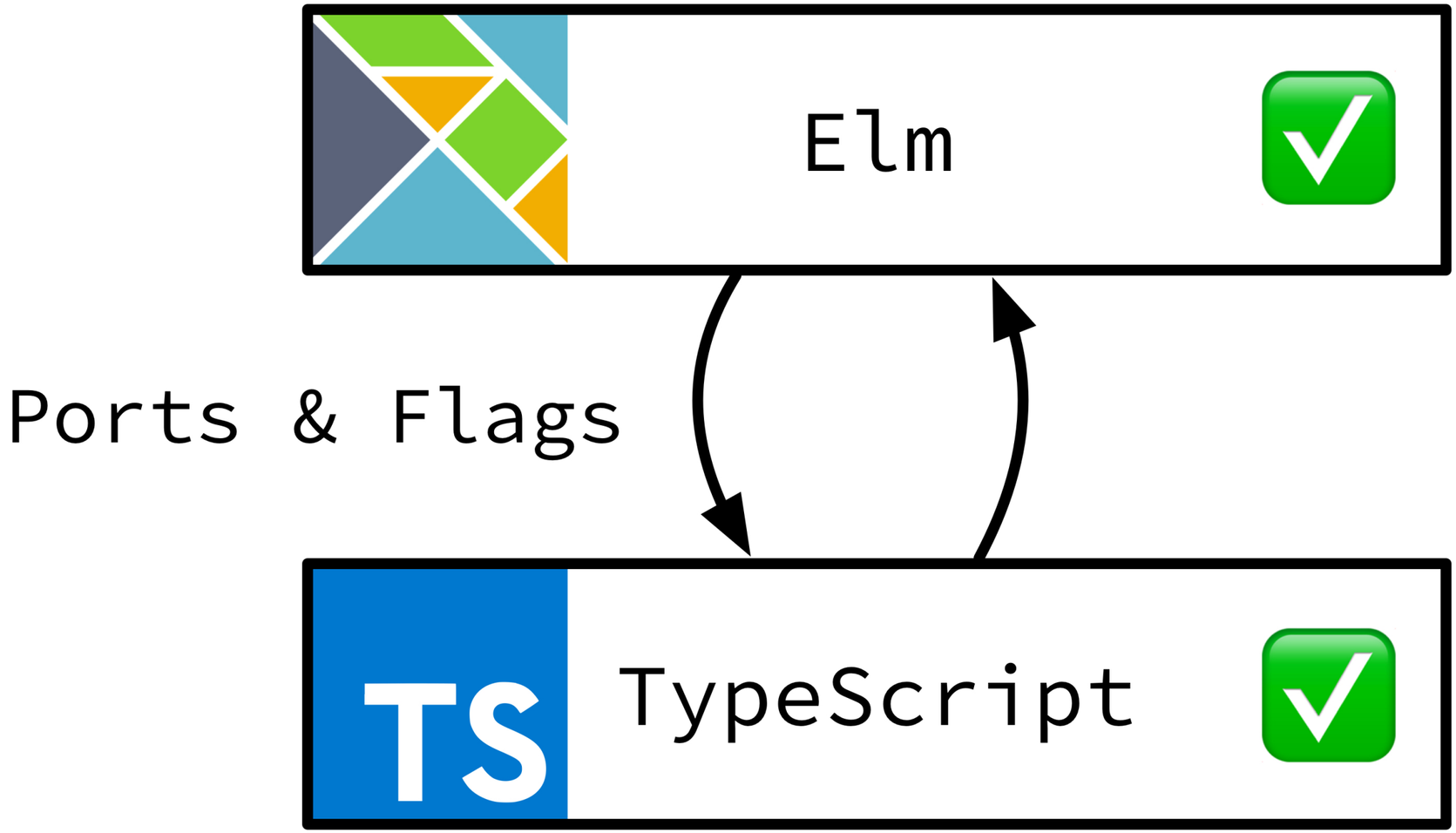
PURESCRIPT

— Avoid double-interop

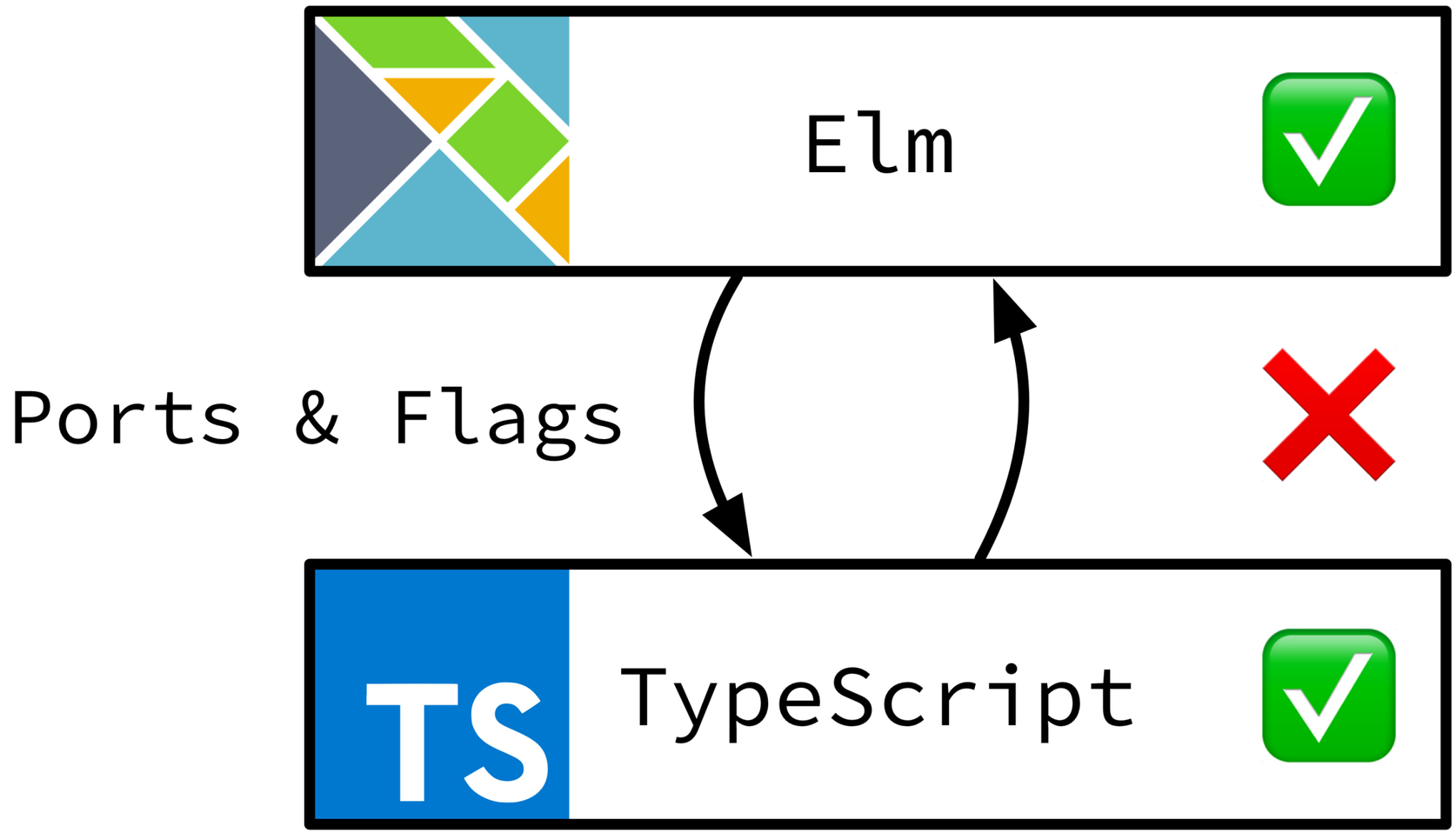
bit.ly/typeswithoutborders




bit.ly/typeswithoutborders



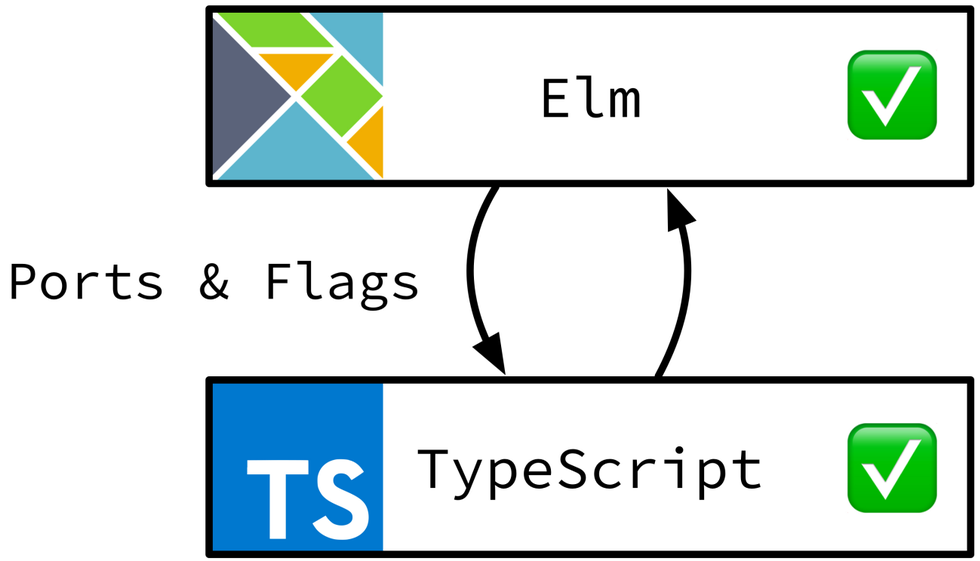

bit.ly/typeswithoutborders



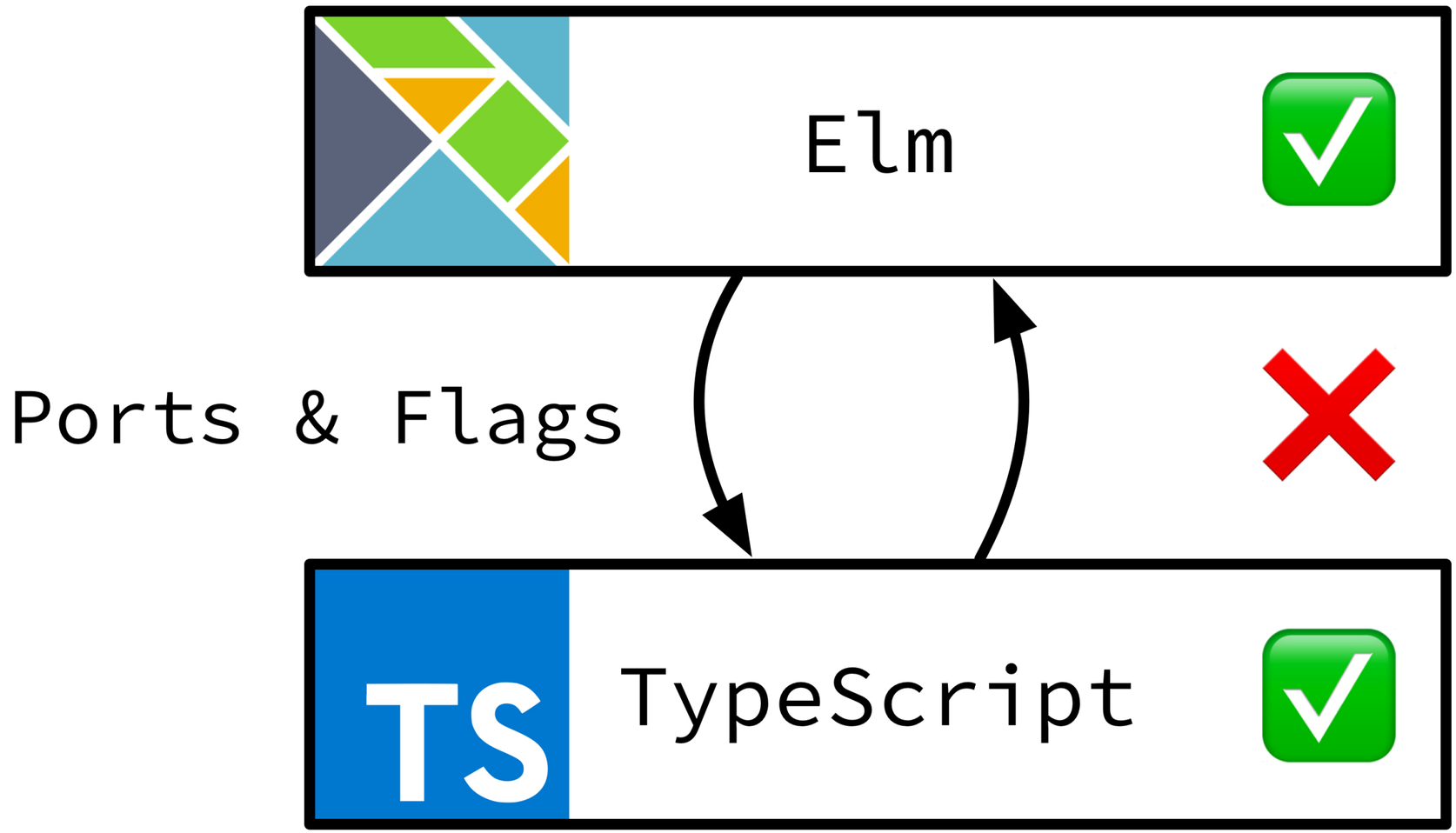

bit.ly/typeswithoutborders

# Demo!

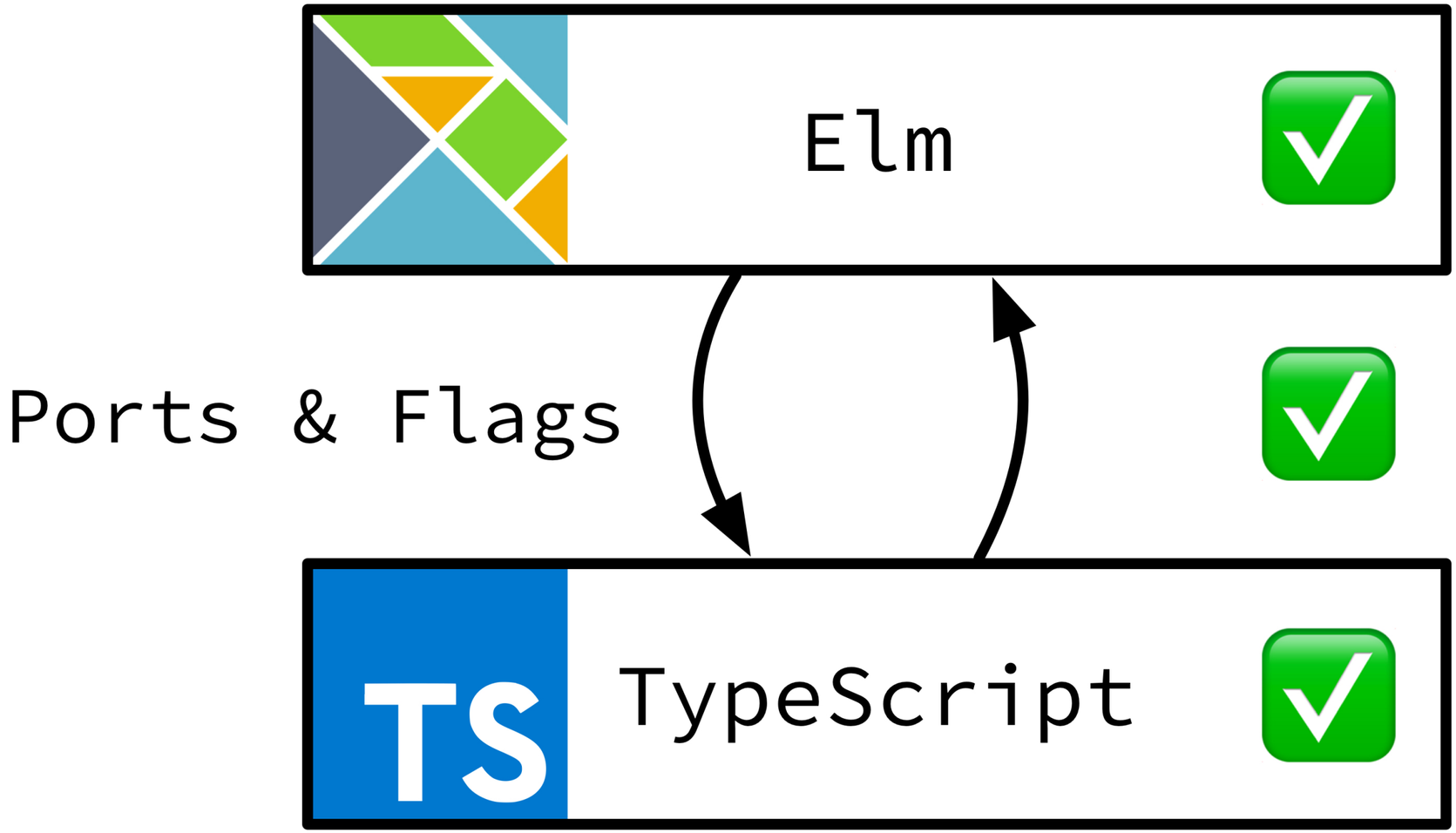
`elm-typescript-interop`




bit.ly/typeswithoutborders




bit.ly/typeswithoutborders




bit.ly/typeswithoutborders


bit.ly/typeswithoutborders

# Variable Contracts

— APIs

— Databases/Storage 

— Message Passing  

— Any user-defined contract

# localStorage

localStorage

**Schema 1.0**

firstName: String!
lastName: String!

localStorage

**Schema 1.0**

firstName: String!
lastName: String!

firstName: "Jane"
lastName: "Doe"

localStorage

**Schema 1.0**

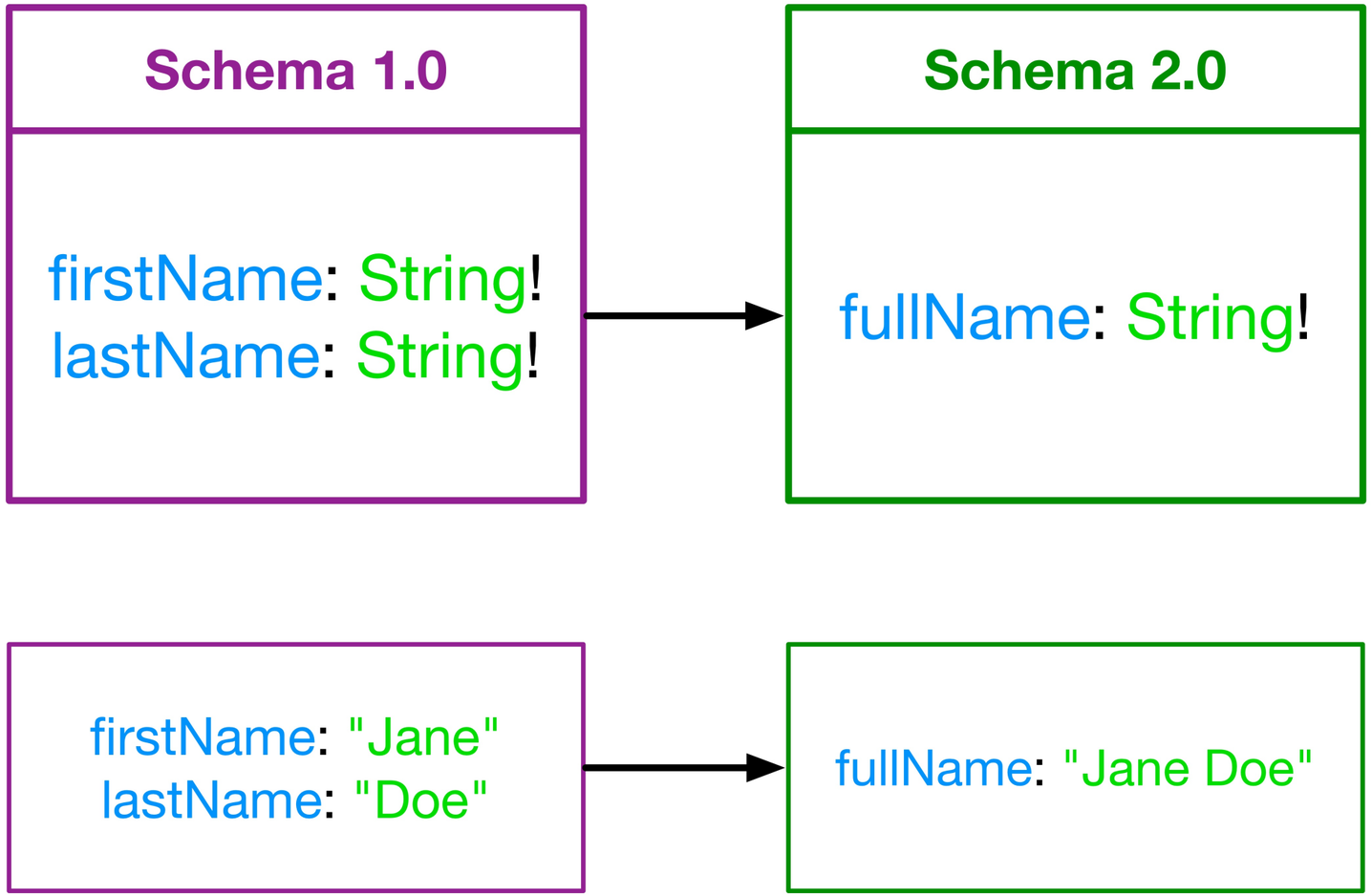
firstName: String!
lastName: String!

**Schema 2.0**

fullName: String!

firstName: "Jane"
lastName: "Doe"

localStorage




bit.ly/typeswithoutborders

# Sharing About Code Generation

— Code generation is more tedious than difficult
— Blog posts, talks
— End-to-end testing techniques
— Extract libraries

bit.ly/typeswithoutborders

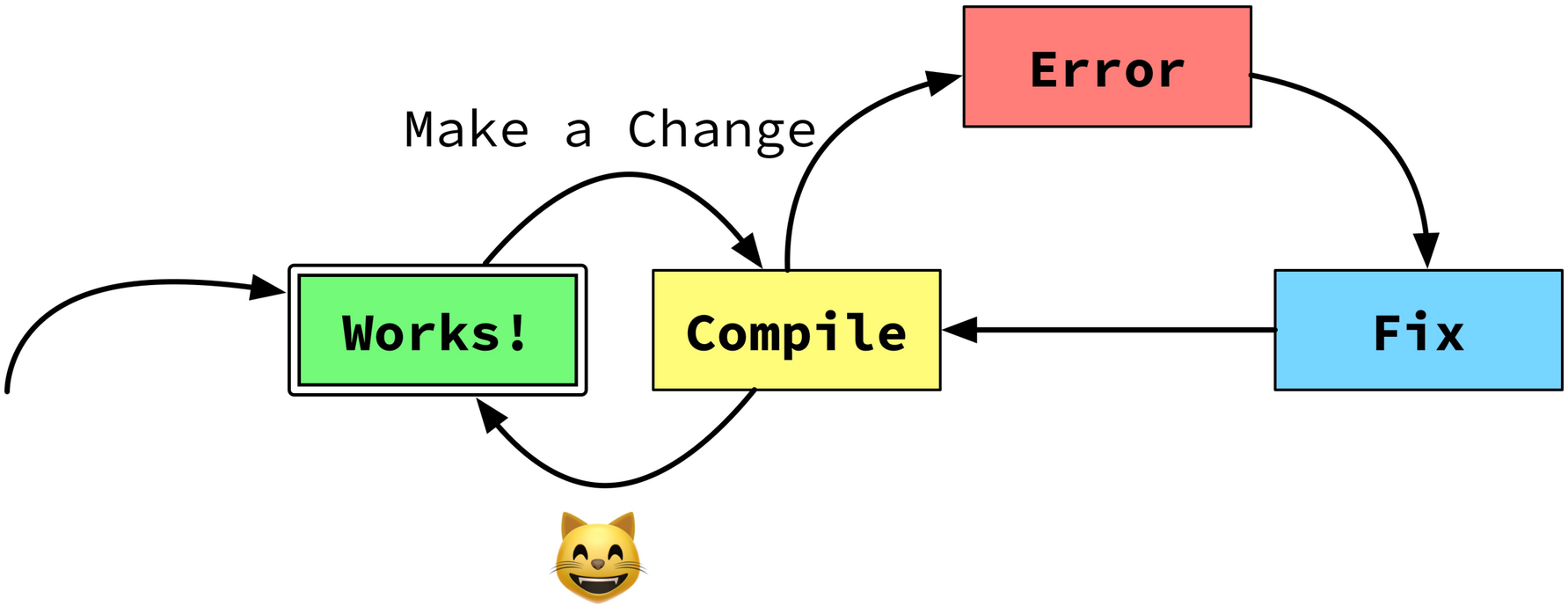
# Representing Contracts in Elm

- Advanced Types posts - Charlie Koster (Opaque Types, Phantom Types)
- Making Impossible States Impossible - Richard
- Make Data Structures - Richard
- Scaling Elm Apps - Richard
- Understanding Style - Matt Griffith

bit.ly/typeswithoutborders

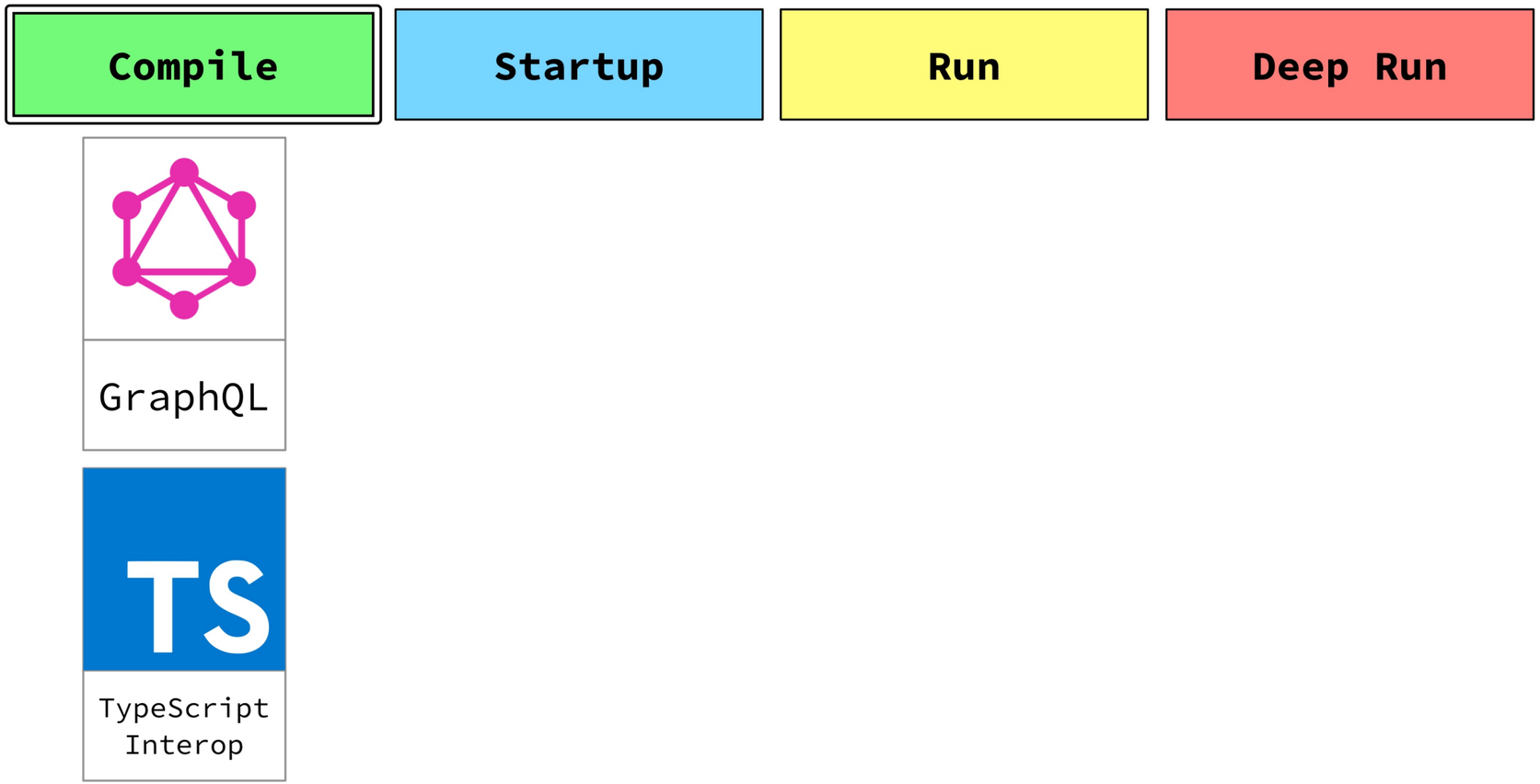

bit.ly/typeswithoutborders

# Let's Eliminate Avoidable Uncertainty!



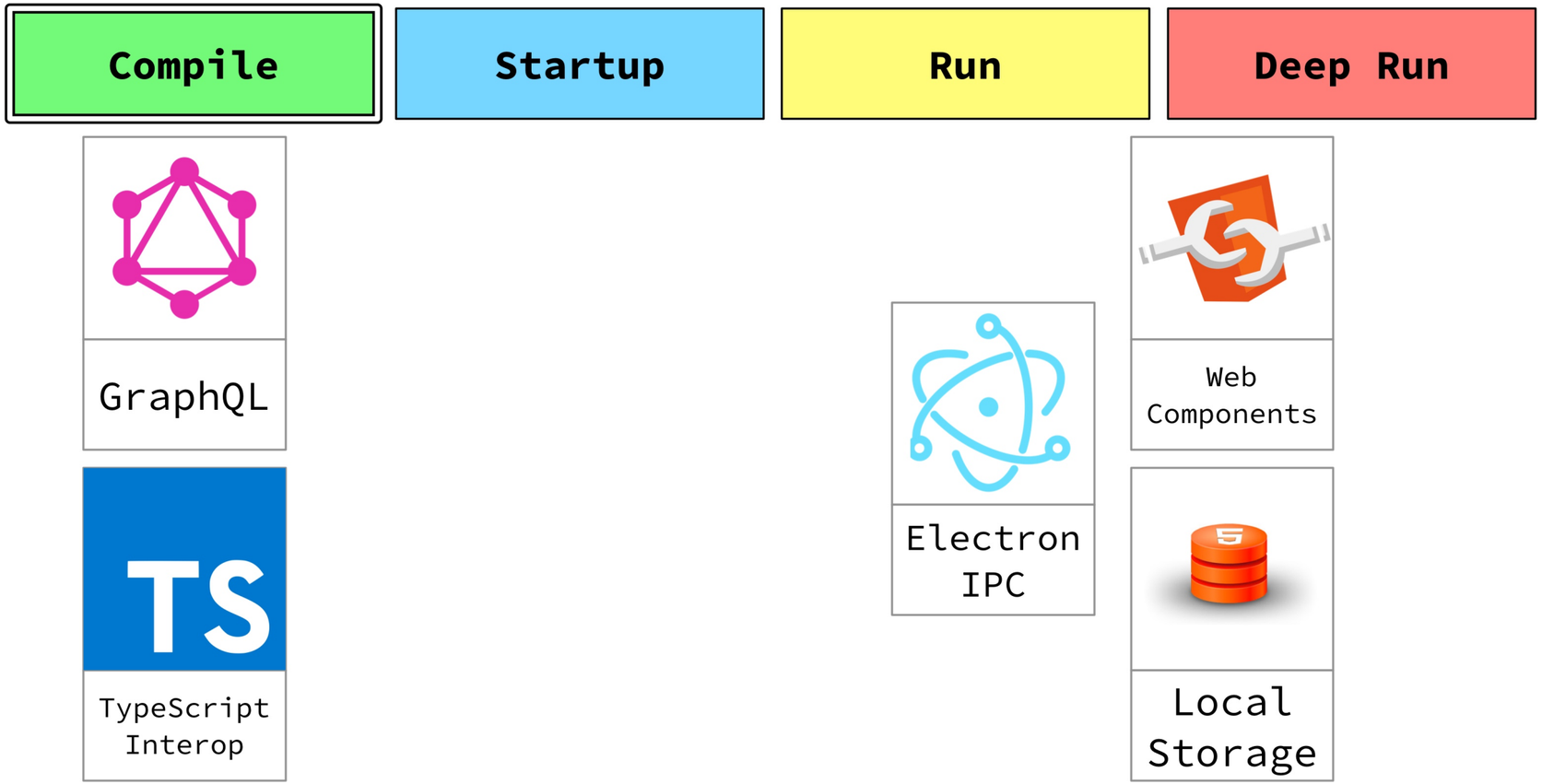

bit.ly/typeswithoutborders

# Let's Be Certain Early!




bit.ly/typeswithoutborders

# Let's Be Certain Early!




bit.ly/typeswithoutborders

# #TypesWithoutBorders

What are your ideas?

bit.ly/typeswithoutborders

# Thank You!

Dillon Kearns
incrementalelm.com

bit.ly/typeswithoutborders