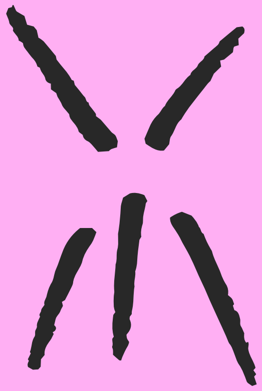
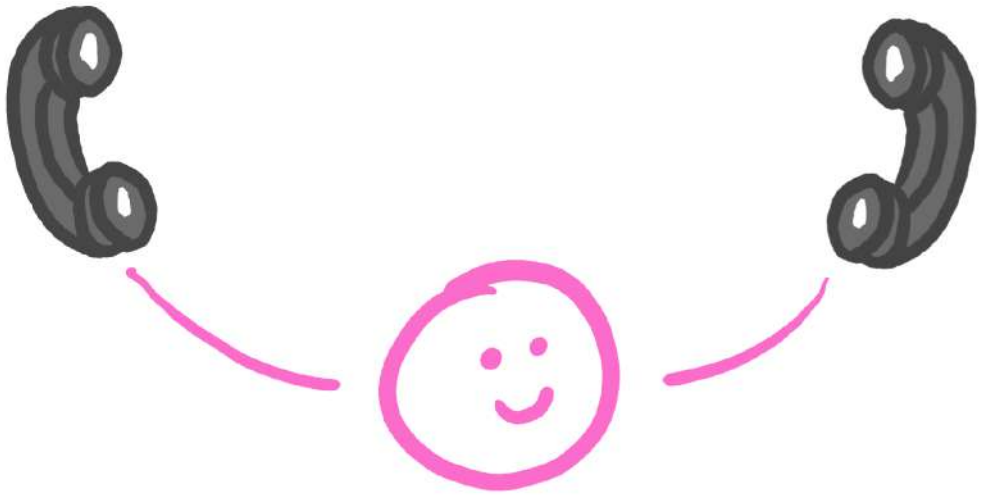
Hi, I'm Louis!

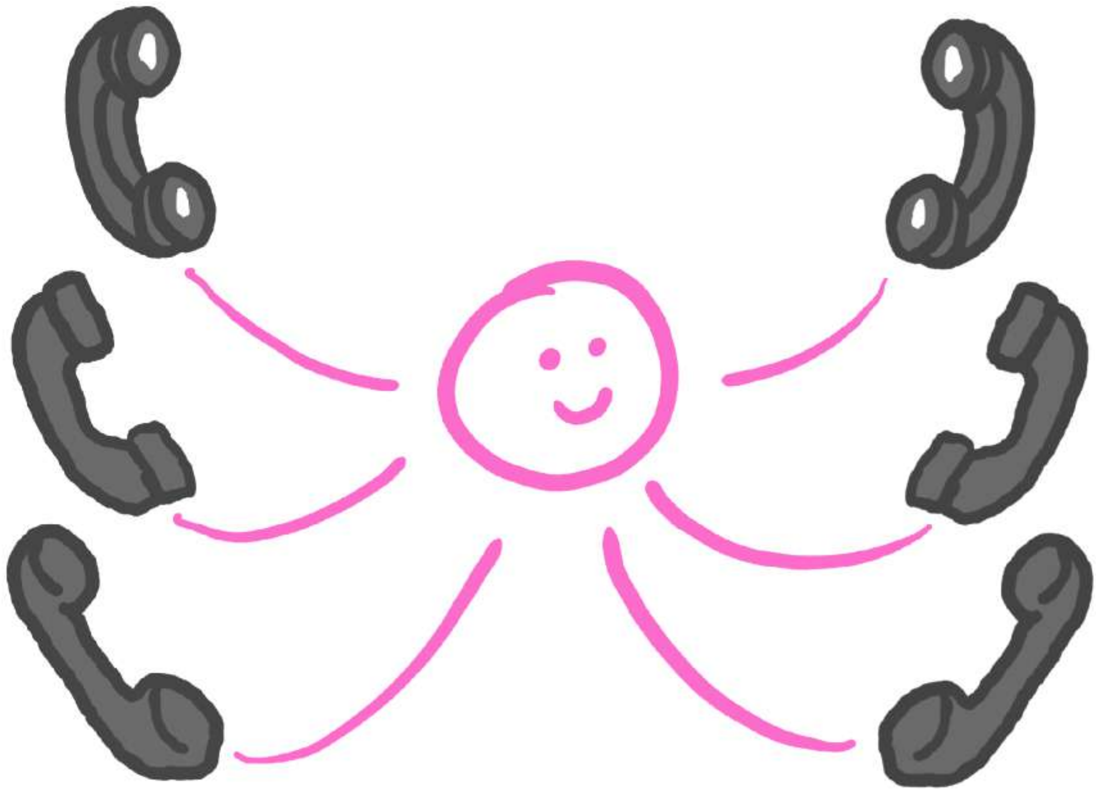
https://lpil.uk/talk-slides/code-mesh-2019 gleam



Why Erlang?

The BEAM!





2 million ejabberd connections

https://blog.process-one.net/ejabberd-massive-scalability-1node-2-million-concurrent-users/

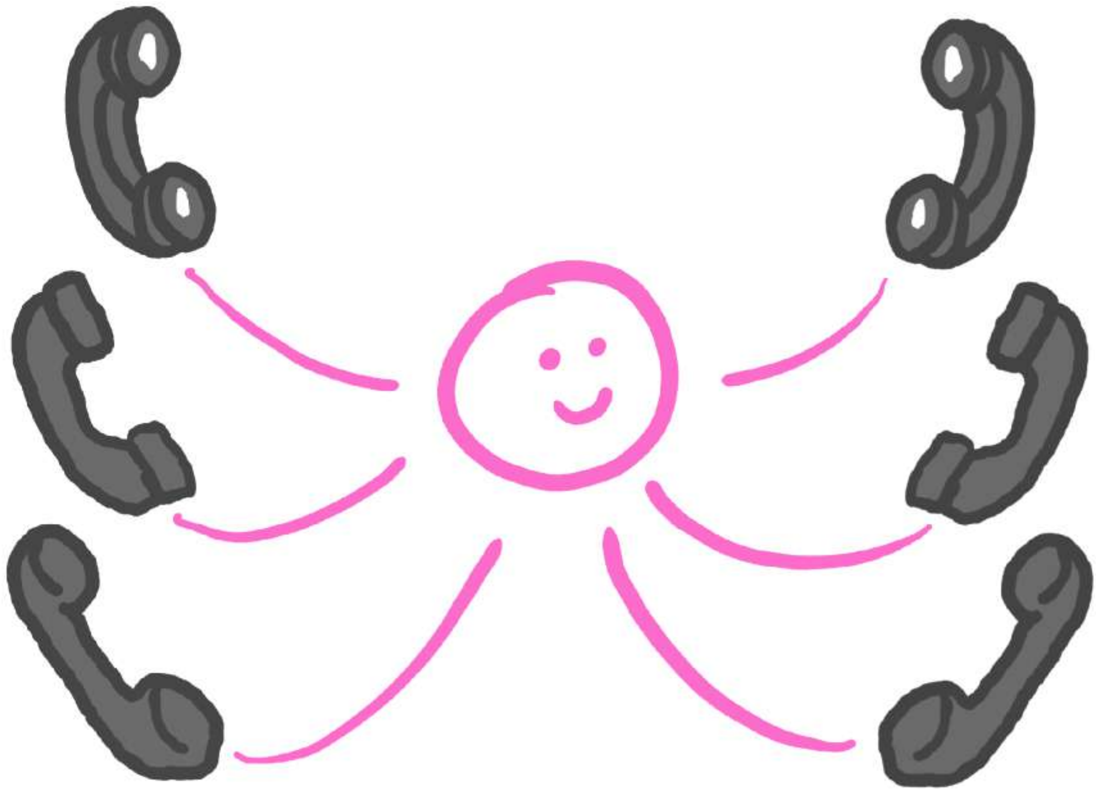
2 million Phoenix connections

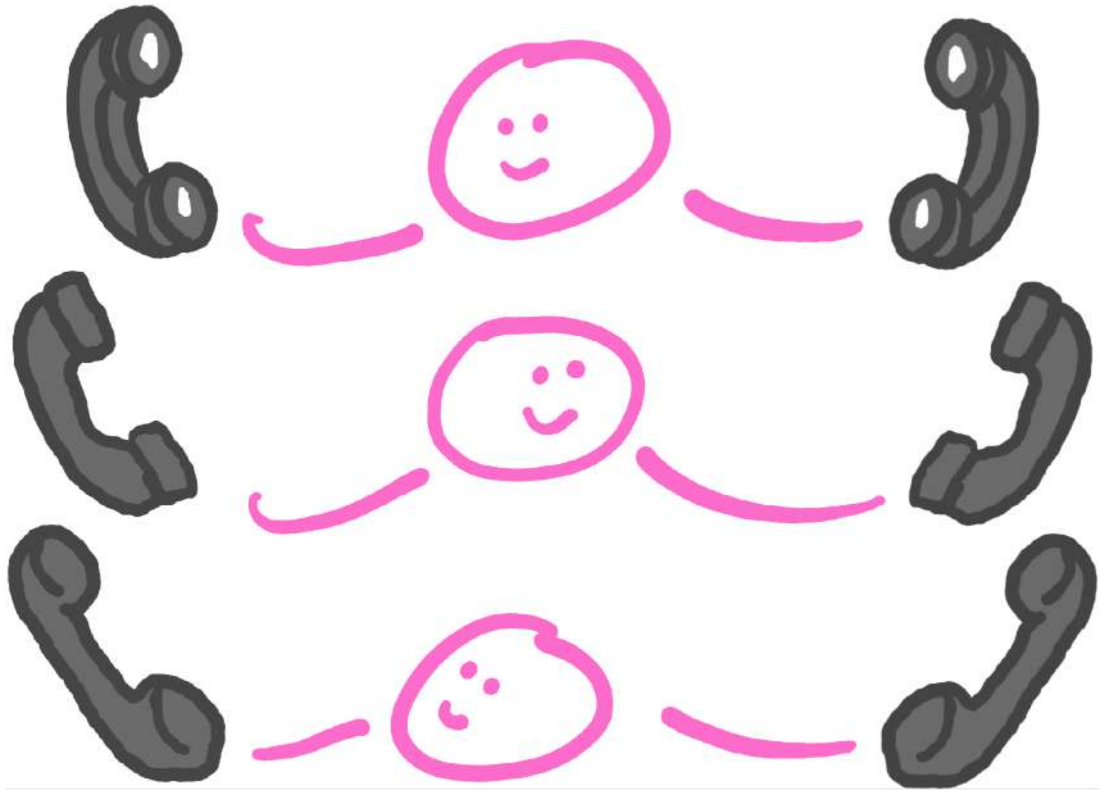
https://phoenixframework.org/blog/the-road-to-2-million-websocket-connections

20 million Erlang threads

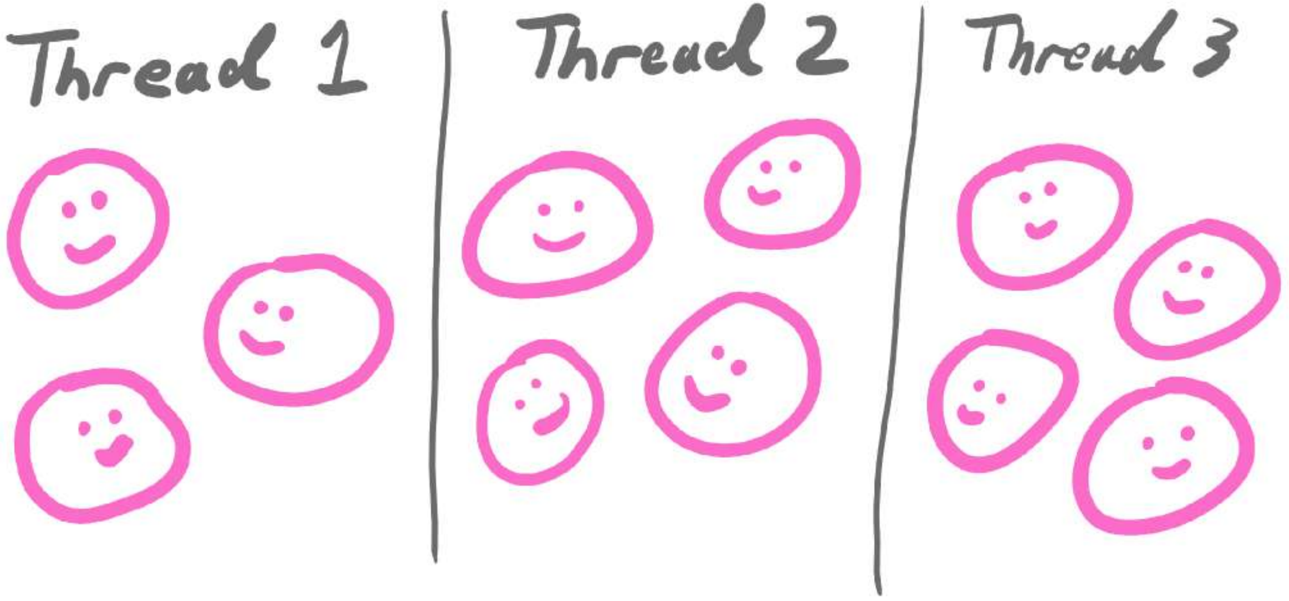
ERICSSON

https://groups.google.com/forum/#!original/comp.lang.functional/5kldn1QJ73c/T3py-yqmtzMJ

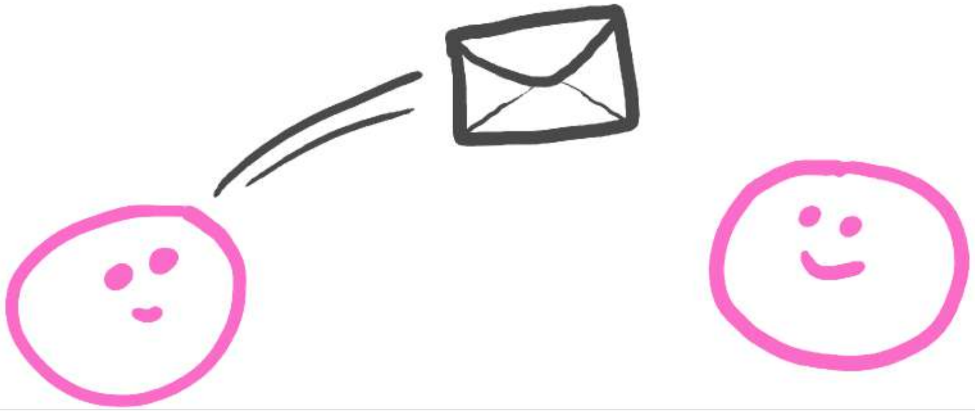




Multi-threaded

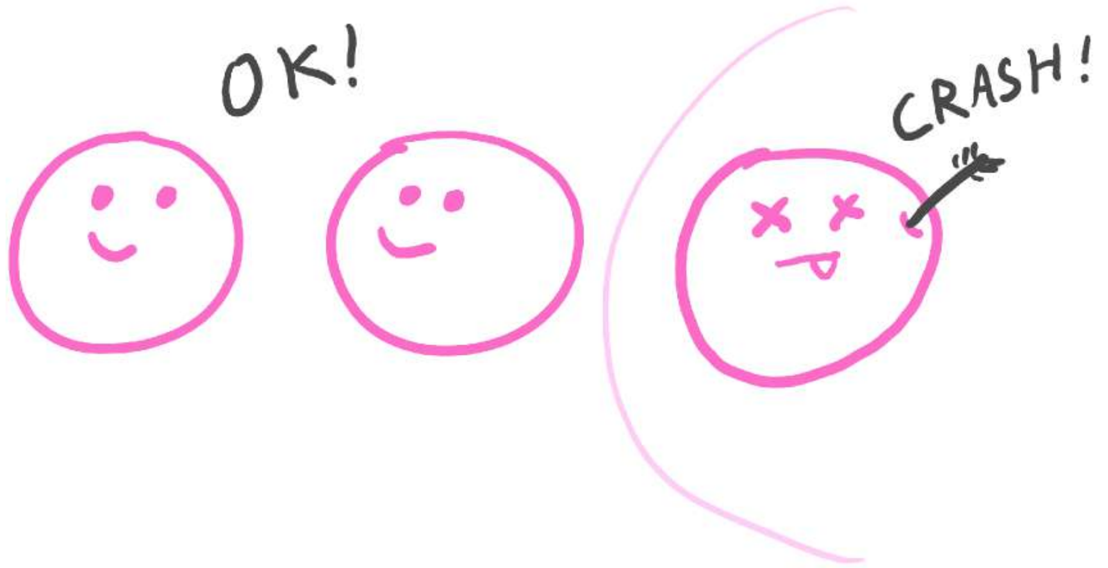


Messages

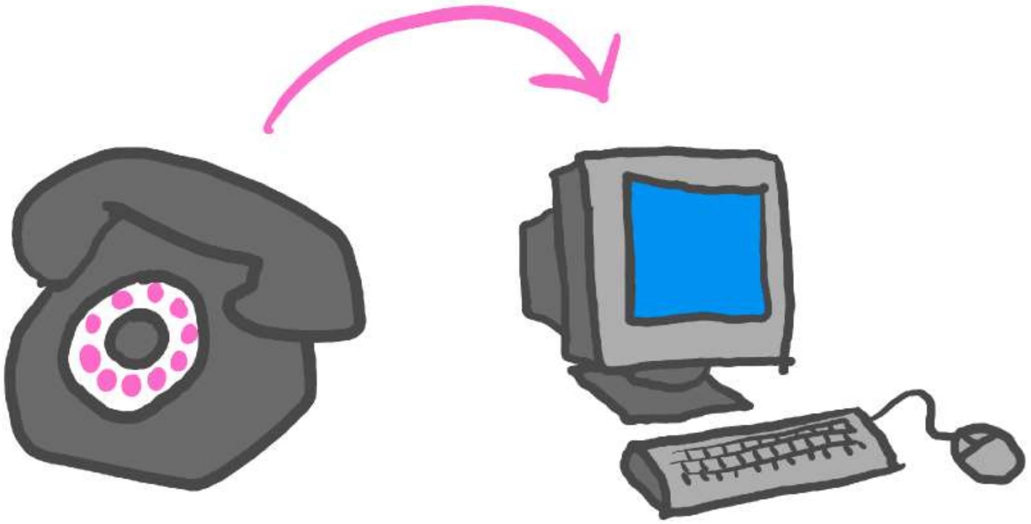


Garbage collection

Fault tolerance



Getting less niche



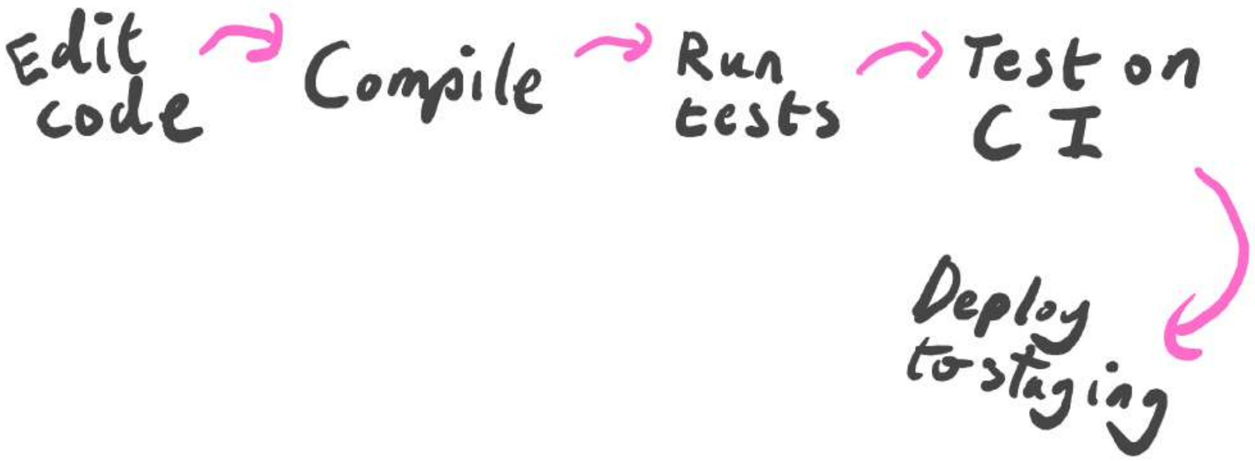
Why a new language?

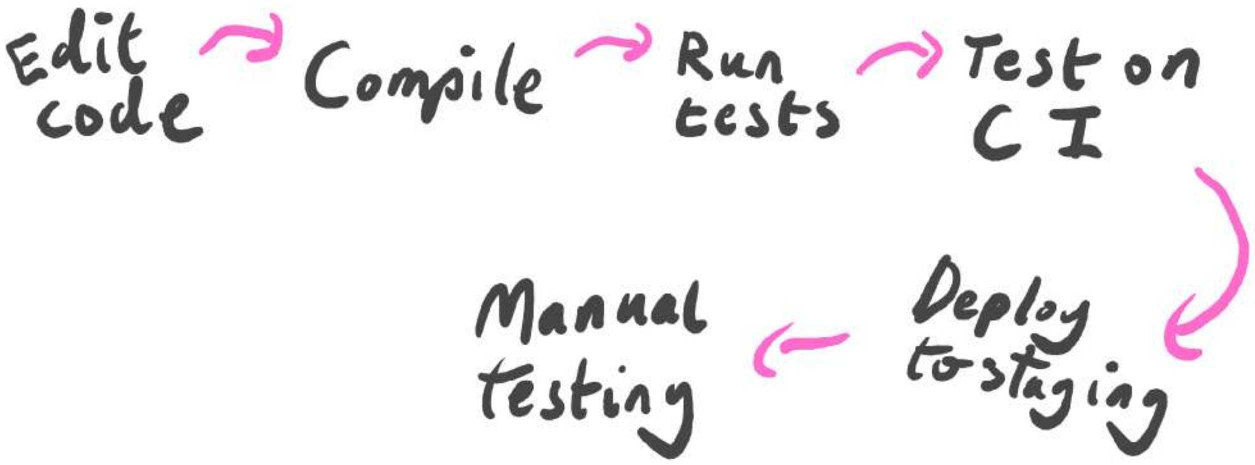
Edit code

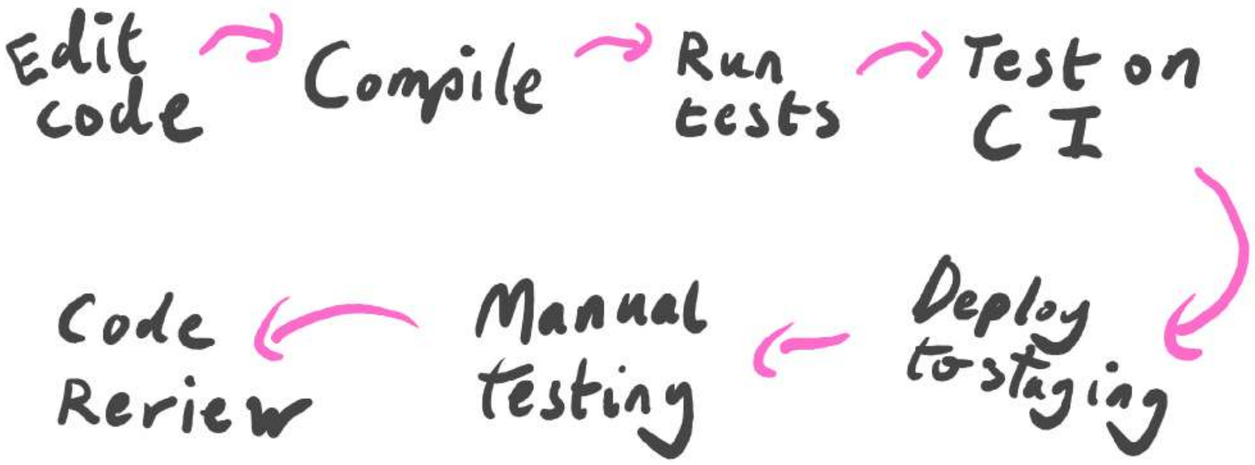
$$\begin{tikzcd}\text{Edit code} \arrow[r, bend left] & \text{Compile}\end{tikzcd}$$

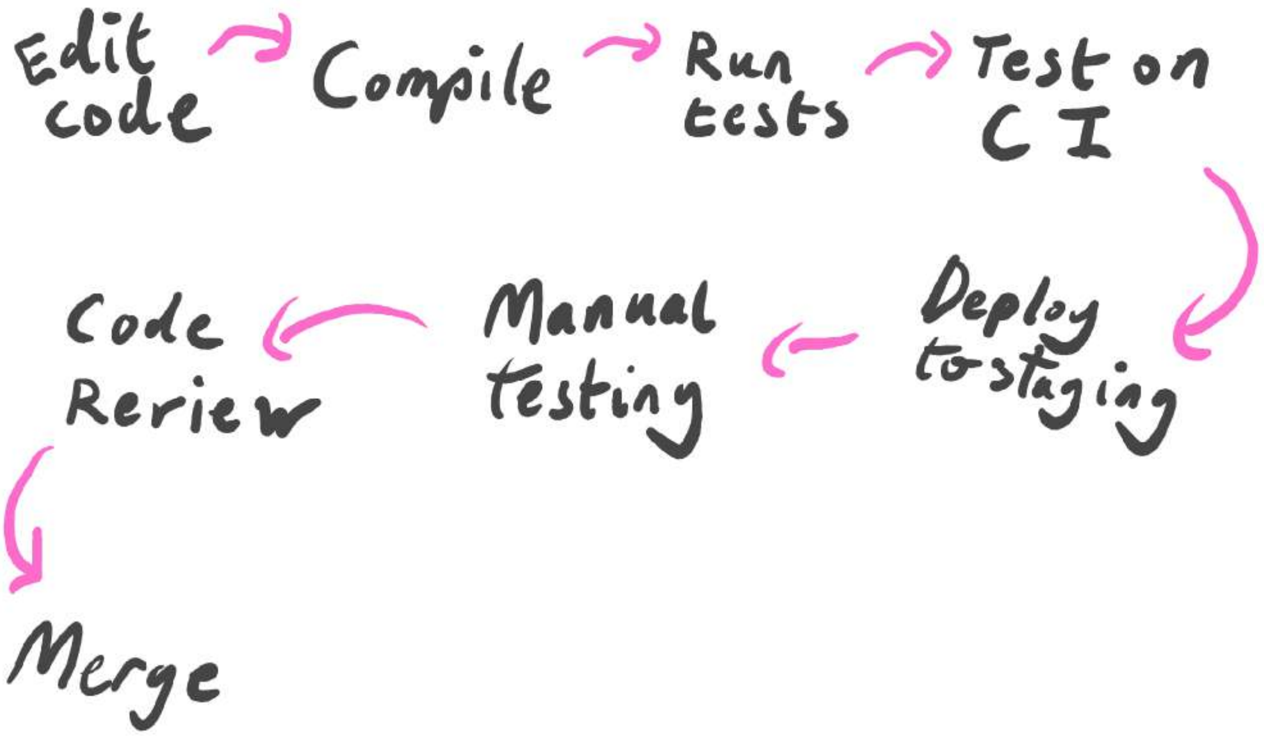
$$\begin{tikzcd}\text{Edit code} \arrow[r] & \text{Compile} \arrow[r] & \text{Run tests}\end{tikzcd}$$

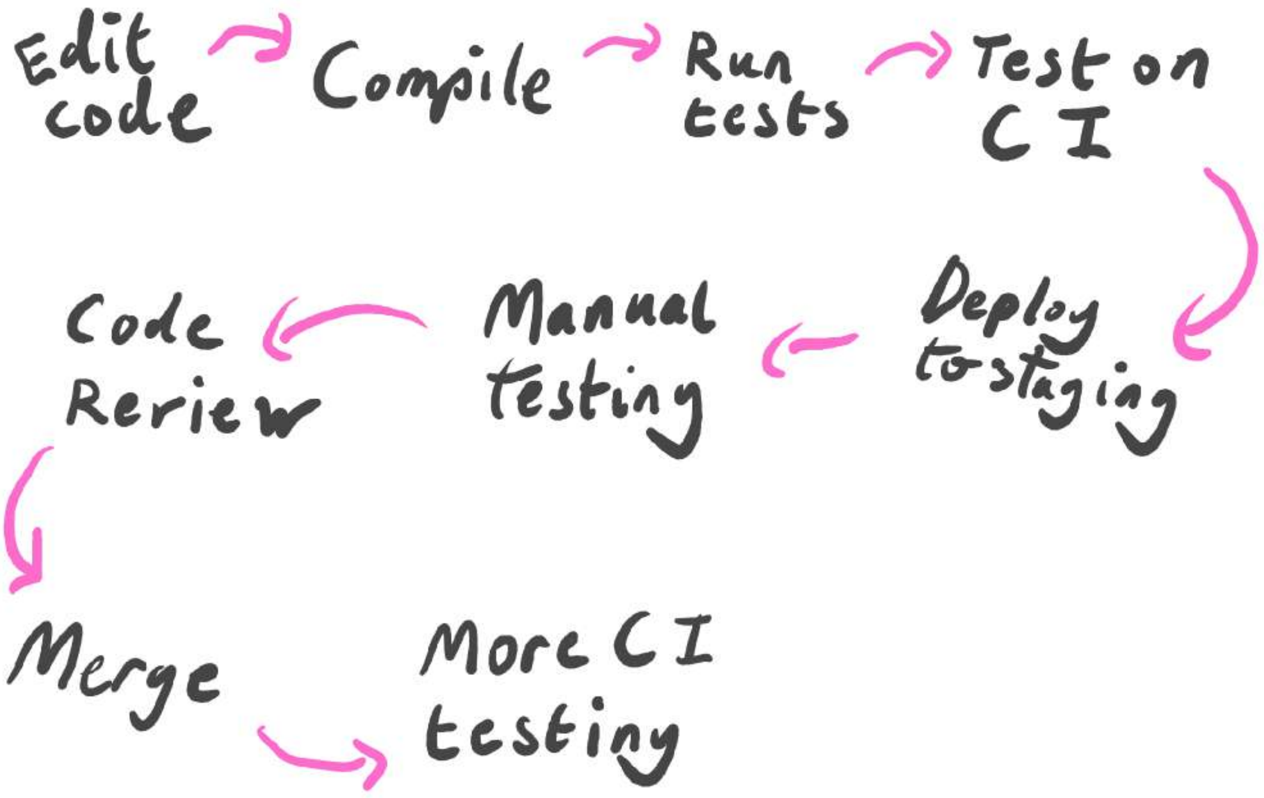
Edit code → Compile → Run tests → Test on CI

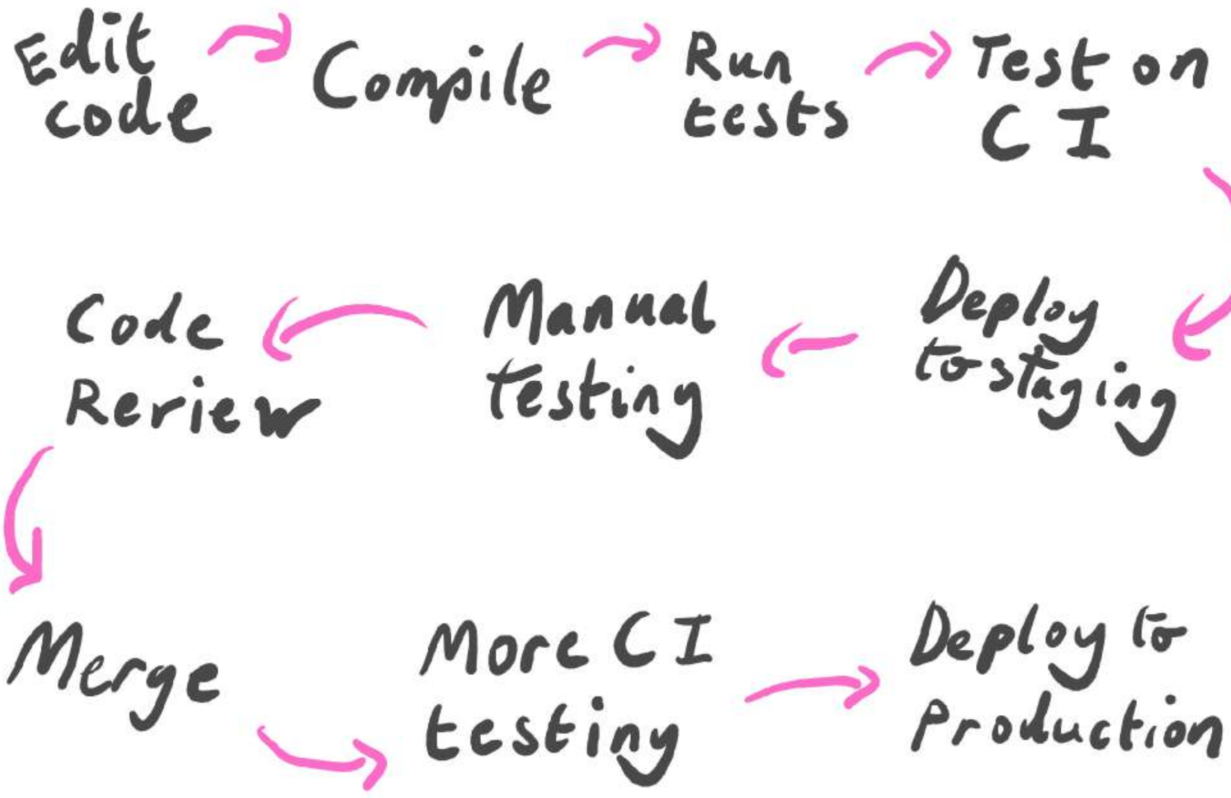

Edit code
Compile
Run tests
Test on CI
Deploy to staging

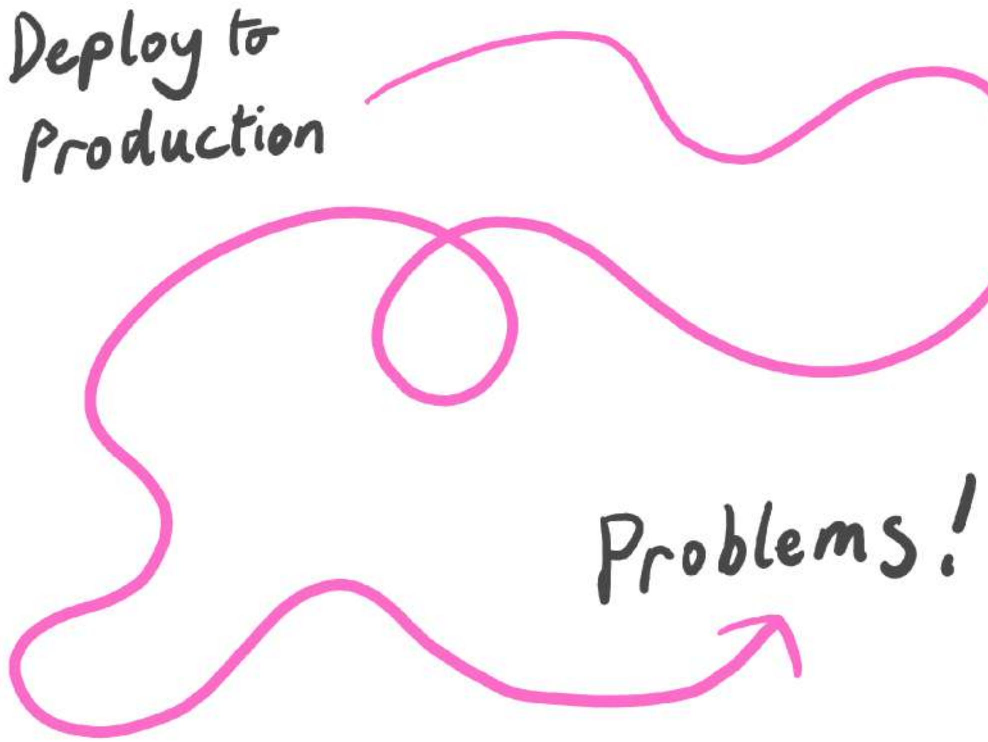

Edit code
Compile
Run tests
Test on CI
Deploy to staging
Manual testing

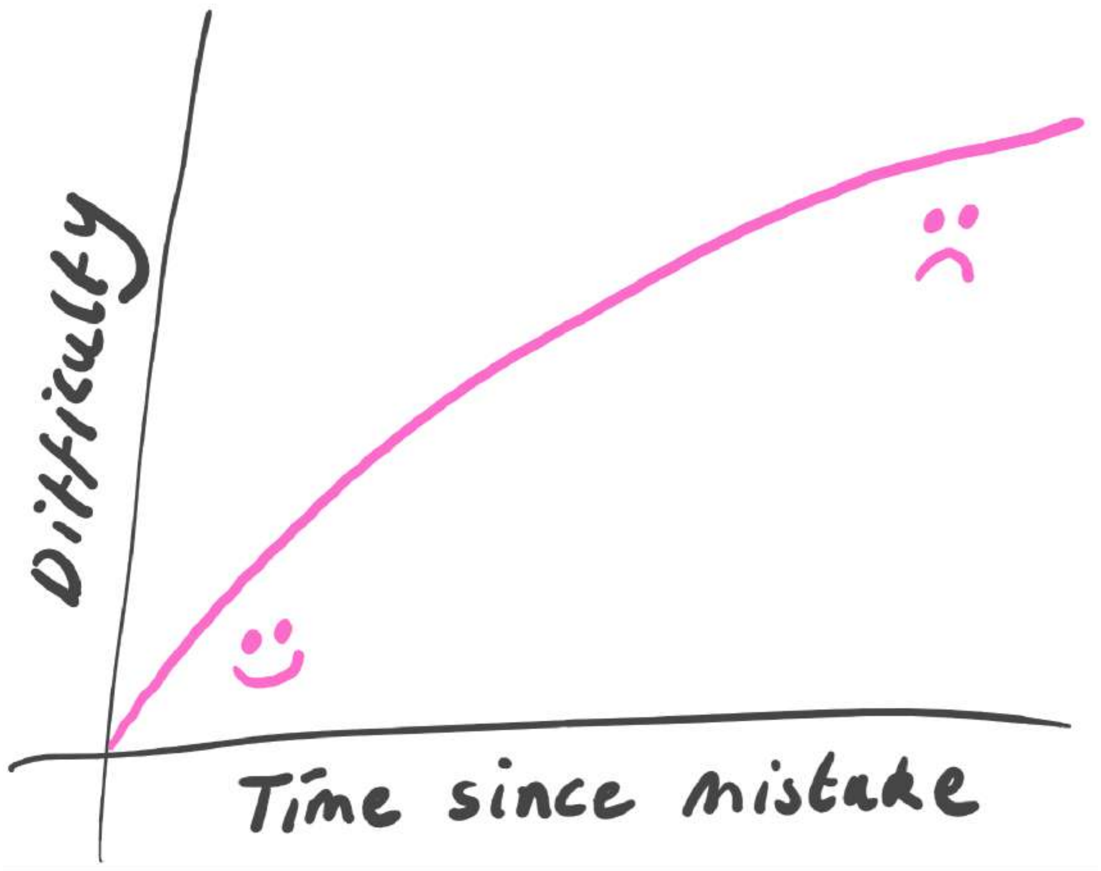

Edit code
Compile
Run tests
Test on CI
Deploy to staging
Manual testing
Code Review


Edit code
Compile
Run tests
Test on CI
Deploy to staging
Manual testing
Code Review
Merge


Edit code
Compile
Run tests
Test on CI
Deploy to staging
Manual testing
Code Review
Merge
More CI testing

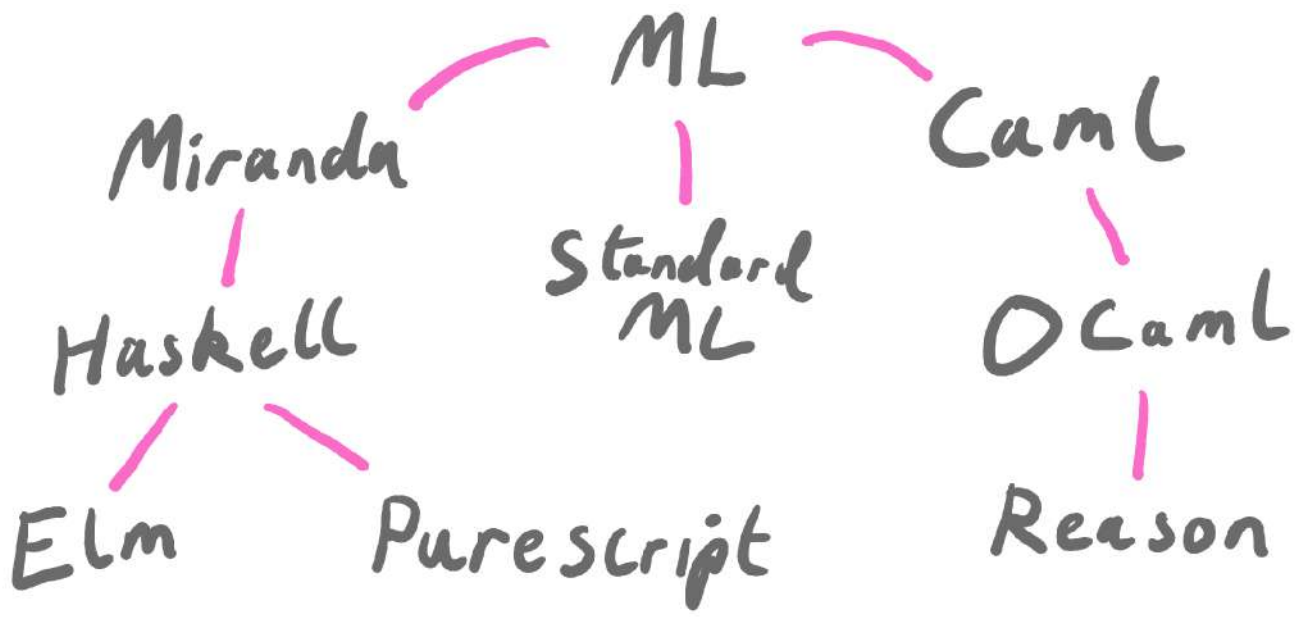

Edit code
Compile
Run tests
Test on CI
Deploy to staging
Manual testing
Code Review
Merge
More CI testing
Deploy to Production


Deploy to
Production
Problems!

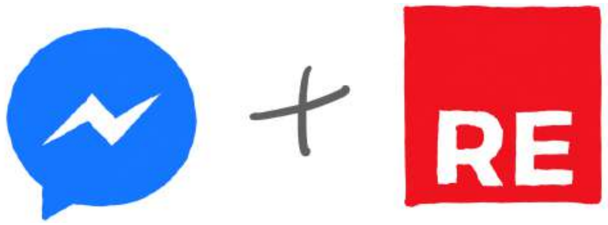

Difficulty
Time since mistake

$$\begin{tikzcd}\text{Edit code} \arrow[r] & \text{Compile} \arrow[r] & \text{X}\end{tikzcd}$$

Bug Detected!


ML
Miranda
Standard ML
Caml
Haskell
OCaml
Elm
Purescript
Reason

Reason



Messenger used to receive bugs reports on a daily basis; since the introduction of Reason, there have been a total of 10 bugs (that's during the whole year, not per week)!

Refactoring speed went from days to hours to dozens of minutes.

https://reasonml.github.io/blog/2017/09/08/messenger-50-reason.html

Elm

noredink +

After 2 years and 200,000 lines of production Elm code, we got our first production runtime exception.

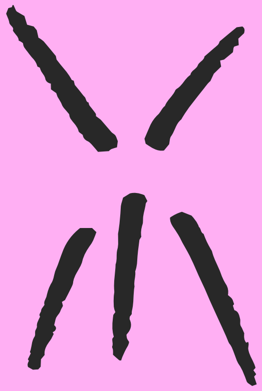
In that period, our legacy JS code has crashed a mere 60,000 times.

Richard Feldman - https://twitter.com/rtfeldman/status/961051166783213570

Purescript

[I've had] such a positive experience, with little mental overhead, and total trust in the compiler. I implemented an entire page with a list of data, filters, search, and pagination which worked first time.

Brandon Martin - https://www.lumi.dev/blog/purescript-and-haskell-at-lumi

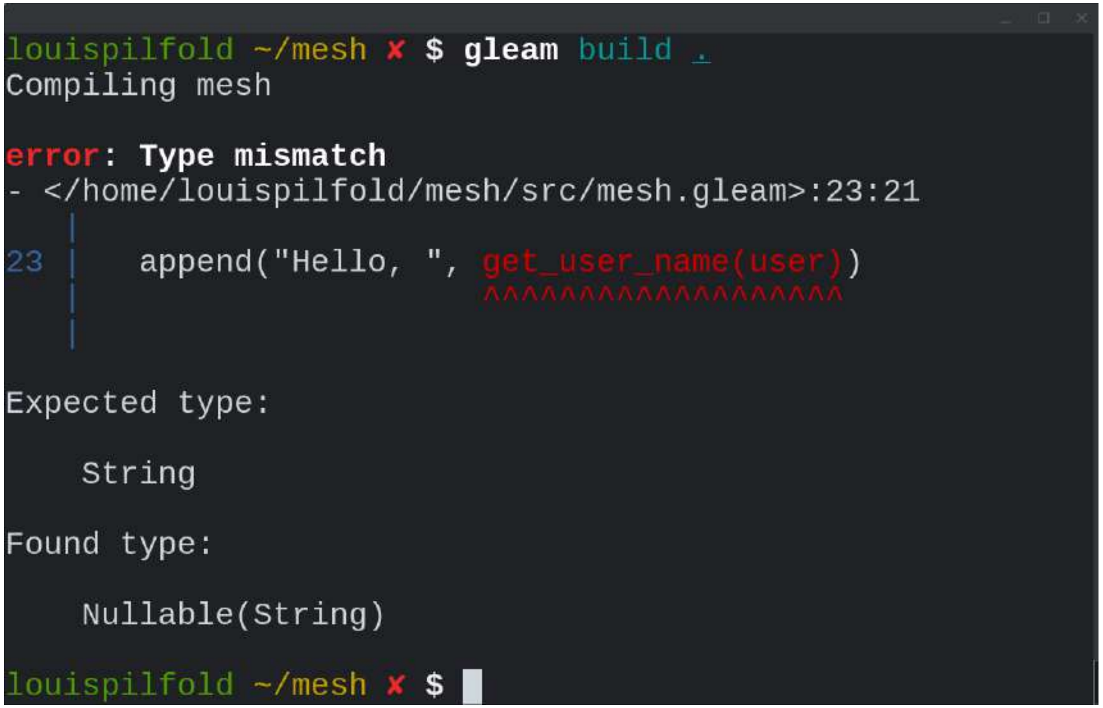


Error detection

```
fn get_user_name(user) {
  let User(name: name) = user
  name
}
```

⇩⇩⇩ becomes ⇩⇩⇩

```
fn get_user_name(user) {
  let User(name: name) = user
  case name {
    "" -> Null
    _ -> Just(name)
  }
}
```



```
louispilfold ~/mesh ✘ $ gleam build .
Compiling mesh

error: Type mismatch
- </home/louispilfold/mesh/src/mesh.gleam>:23:21
   |
23 |    append("Hello, ", get_user_name(user))
   |                      ^^^^^^^^^^^^^^^^^^^
   |

Expected type:

    String

Found type:

    Nullable(String)

louispilfold ~/mesh ✘ $
```

Do we need fault tolerance?

Programmer mistake

```
fn main() {
  list.reverse("hello") // Error! Not a list
}
```

⇩⇩⇩ use the correct function ⇩⇩⇩

```
fn main() {
  string.reverse("hello") // That's better :)
}
```

Incorrect user input

```
fn handle(request) {
  let input = decode_json_body(request)
  // Error! JSON could be invalid
  save_record(input)
}
```

⇩⇩⇩ handle invalid input ⇩⇩⇩

```
fn handle(request) {
  case decode_json_body(request) {
    Ok(input) -> save_record(input)
    Error(reason) -> unprocessable_entity(reason)
  }
}
```

Background processing

```
fn process_video(id) {
  let metadata = lookup_metadata(id)
  create_thumbnails(metadata)
  transcode_video(metadata)
  "done!"
}
```

Defensive programming

```
fn process_video(id) {
  case lookup_metadata(id) {
    Ok(metadata) ->
      case create_thumbnails(metadata) {
        Ok(result) ->
          case transcode_video(metadata) {
            Ok(_) -> "done!"
            Error(transcoder_error) -> ???
          }
        Error(transcoder_error) -> ???
      }
    Error(database_error) -> ???
  }
}
```

Defensive programming

```
fn process_video(id) {
  let result = id
    |> lookup_metadata
    |> result.then(_, fn(metadata) {
      metadata
      |> create_thumbnails
      |> result.map(_, fn(_) { metadata })
    })
    |> result.then(_, transcode_video)
  case result {
    Ok(_) -> "done!"
    Error(e) -> ??? // What do we do here?
  }
}
```

Offensive programming

```
fn process_video(id) {
  assert Ok(metadata) = lookup_metadata(id)
  assert Ok(_) = create_thumbnails(metadata)
  assert Ok(_) = transcode_video(metadata)
  "done!"
}
```

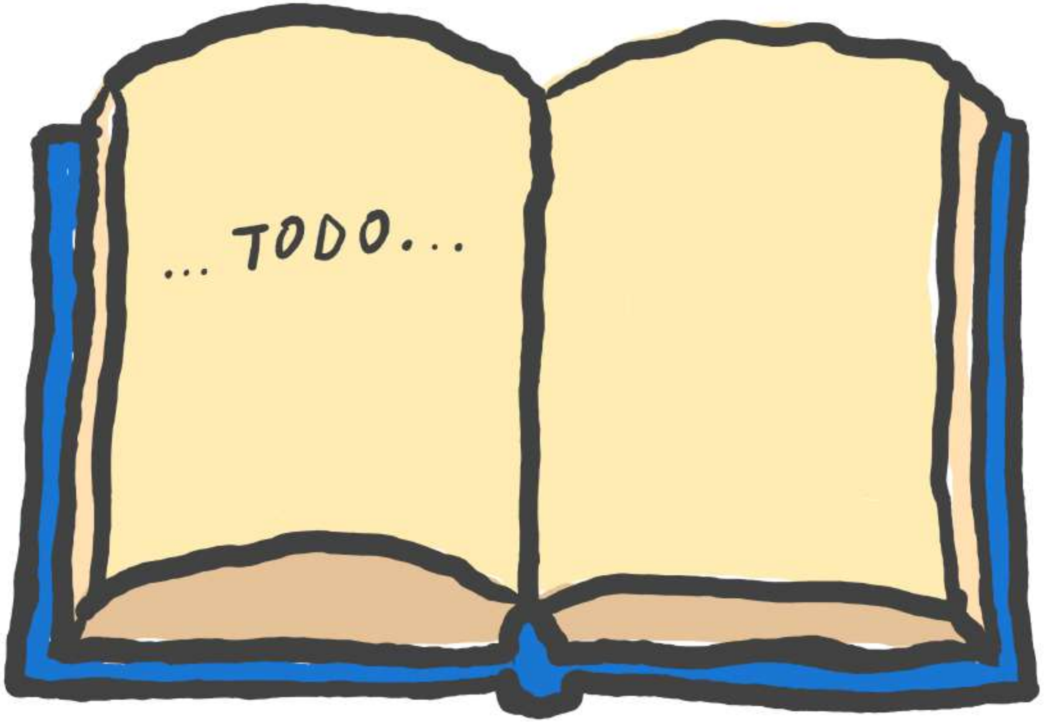
Let it crash

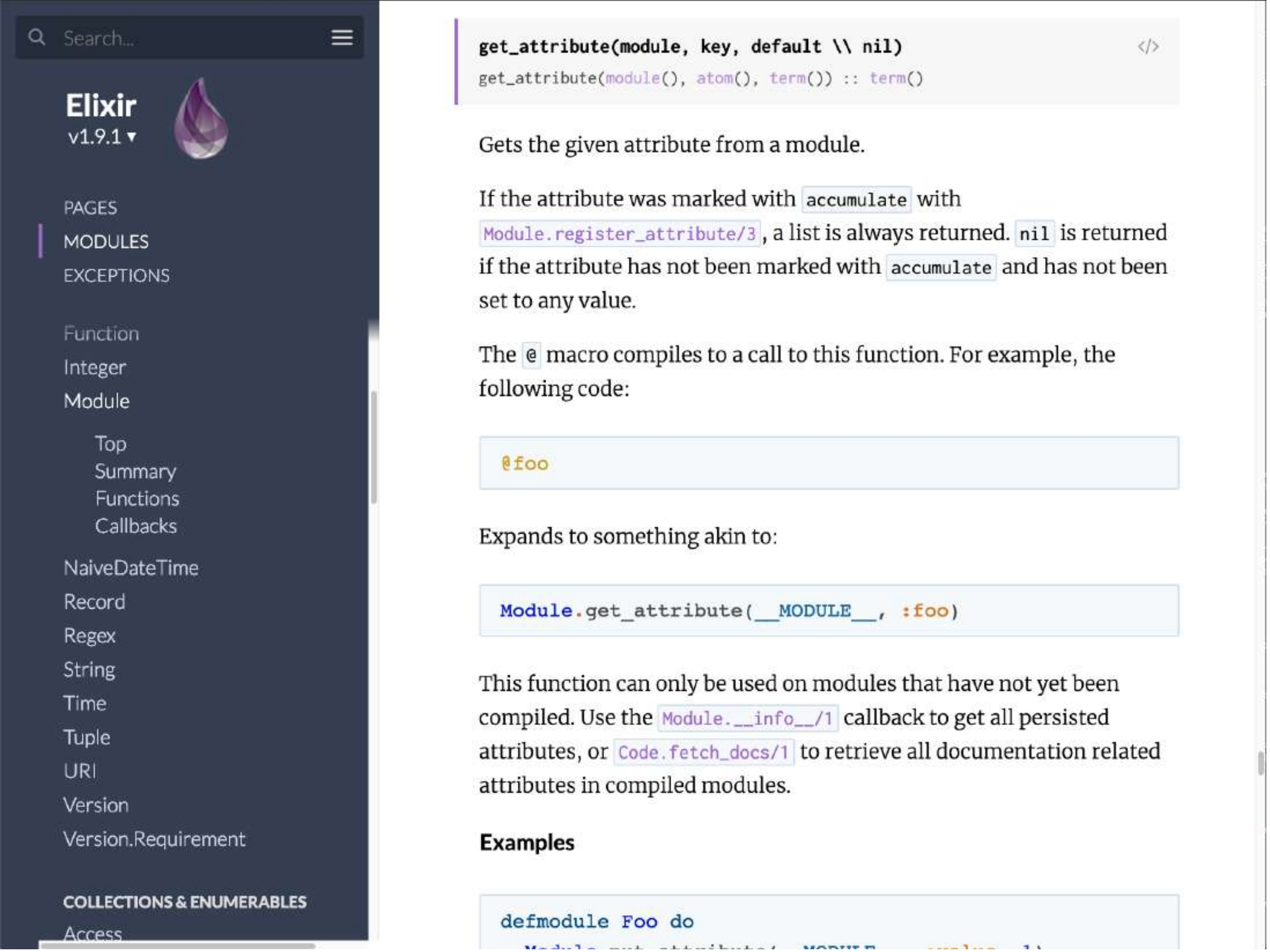
(if you're sure)

What's next for Gleam?

Editor integration

Documentation





get_attribute(module, key, default \\ nil)

`get_attribute(module(), atom(), term()) :: term()`

Gets the given attribute from a module.

If the attribute was marked with `accumulate` with `Module.register_attribute/3`, a list is always returned. `nil` is returned if the attribute has not been marked with `accumulate` and has not been set to any value.

The `@` macro compiles to a call to this function. For example, the following code:

```
@foo
```

Expands to something akin to:

```
Module.get_attribute(__MODULE__, :foo)
```

This function can only be used on modules that have not yet been compiled. Use the `Module.__info__/1` callback to get all persisted attributes, or `Code.fetch_docs/1` to retrieve all documentation related attributes in compiled modules.

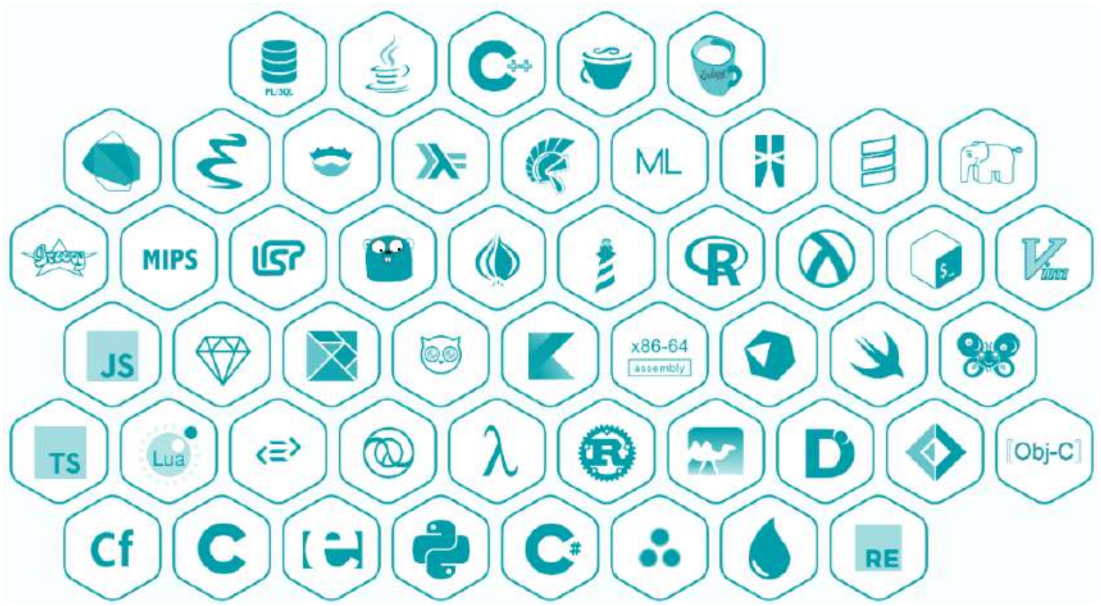
Examples

```
defmodule Foo do
```

Exercism

https://exercism.io

51 Languages and counting



Gleam 💕 Erlang

gleam

- https://gleam.run
- https://github.com/gleam-lang/gleam
- IRC `#gleam-lang` on Freenode

Griffics' art

- https://www.griffics.com/

Call me?

- twitter @louispilfold