Introduction to Gleam

Building type-safe Discord bots on the BEAM

HARRY BAIRSTOW

FOSDEM'23

Erlang, Elixir & Friends Devroom

Harry Bairstow

- 🧑‍💻 Gleam Contributor.
- 🛠️ Distributed Systems Engineer @ WalletConnect.
- 🎈 Learning to fly hot-air balloons.

FOSDEM'23

Erlang, Elixir & Friends Devroom

What is Gleam?

Gleam is a programing language for building type-safe systems that scale! It's powered primarily by BEAM but can be run on any JavaScript target too! I thought I'd go into the 3 key points of gleam which are its:

- 🔐 Safety
- 🧵 Performance
- 🫂 Friendliness

```gleam
import gleam/io

pub fn main() {
  io.println("Hello, FOSDEM!")
}
```

Exploring Gleam

FOSDEM'23

Erlang, Elixir & Friends Devroom

```
let x = 1
let y = x
let x = 2

x  // => 2
y  // => 1
```

```
let value: Bool = {
  "Hello"
  42 + 12
  False
}
```

```
let celsius = { fahrenheit - 32 } * 5 / 9
```

```
[1, 2, 3, 4]  // List(Int)
[1.22, 2.30]  // List(Float)
[1.22, 3, 4]  // Type error!
```

```
[1, ..[2, 3]]  // => [1, 2, 3]
```

```
#(10, "hello") // Type is #(Int, String)
#(1, 4.2, [0]) // Type is #(Int, Float, List(Int))
```

```
let my_tuple = #("one", "two")
let first = my_tuple.0   // "one"
let second = my_tuple.1 // "two"
```

```
pub type Talk {
  Talk(room: String, occupants: Int)
}
```

```
let gleam_intro = Talk("H.1309", 124)
let room = gleam_intro.room // "H.1309"
let occupants = gleam_intro.occupants // 124
```

```
pub type Attendee {
  Speaker(name: String, talks: List(String))
  DevroomManager(name: String, mic: Bool)
  AudienceMember(name: String)
}
```

```
case some_number {
  0 -> "Zero"
  1 -> "One"
  2 -> "Two"
  n -> "Some other number" // This matches anything
}
```

```
case #("fn_1", 1, False) {
  #("fn_1", 4, True) -> "No"
  #("fn_2", _, False) -> "Yes"
  _ -> "Maybe"
}
```

```
// Should they have the mic?
case person {
  Speaker(talks: talks, ..) -> {
    // more logic here
    result
  }
  DevroomManager(mic: new_mic, ..) -> new_mic
  AudienceMember(..) -> False
}
```

And more!

Building bots on the BEAM

FOSDEM'23

Erlang, Elixir & Friends Devroom

Introduction to Shimmer ✨

Shimmer is a library for interacting with the Discord API from the BEAM, it's written in Gleam and leverages all the Gleam features, making use of the standard library as much as possible.

Lets get into some of the key points →

🧩 Compatibility

While Shimmer is built in Gleam it can be used in Elixir, Erlang, and any other BEAM based package.

🎭 Actor Based

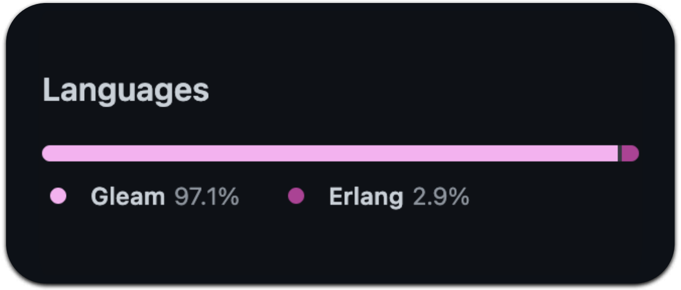
Shimmer is powered by just one actor in single-shard mode. Multiple shards? We use a supervisor tree with multiple actors under it!

🔐 Type Safe

When building your Discord bot in gleam we leverage all of Gleam's type functionality to ensure that the code you write for the BEAM is type safe.

Fun fact!

At the time of writing this Shimmer is 97.1% Gleam with the rest being Erlang FFI functions for networking.



Languages

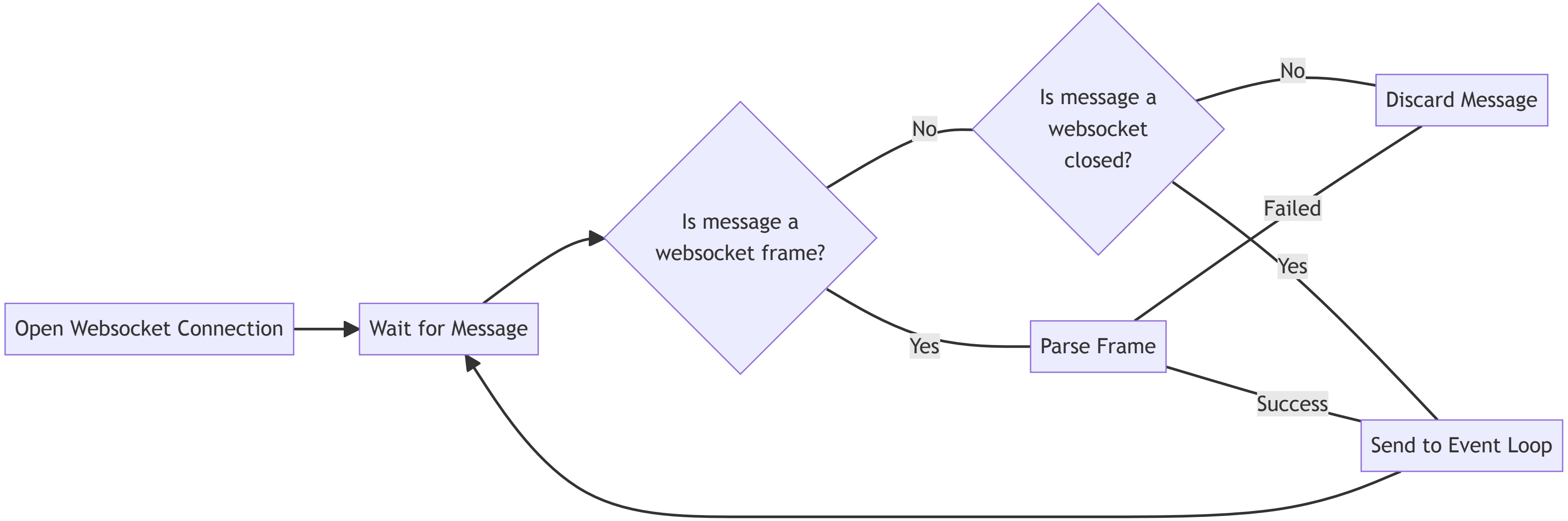
● Gleam 97.1% ● Erlang 2.9%

How does it all work?

For some of you this might be the most interesting part of my talk and for others it will be how to actually build the bots. So lets quickly explore **OTP** & the internal workings of Shimmer.

Receiving messages

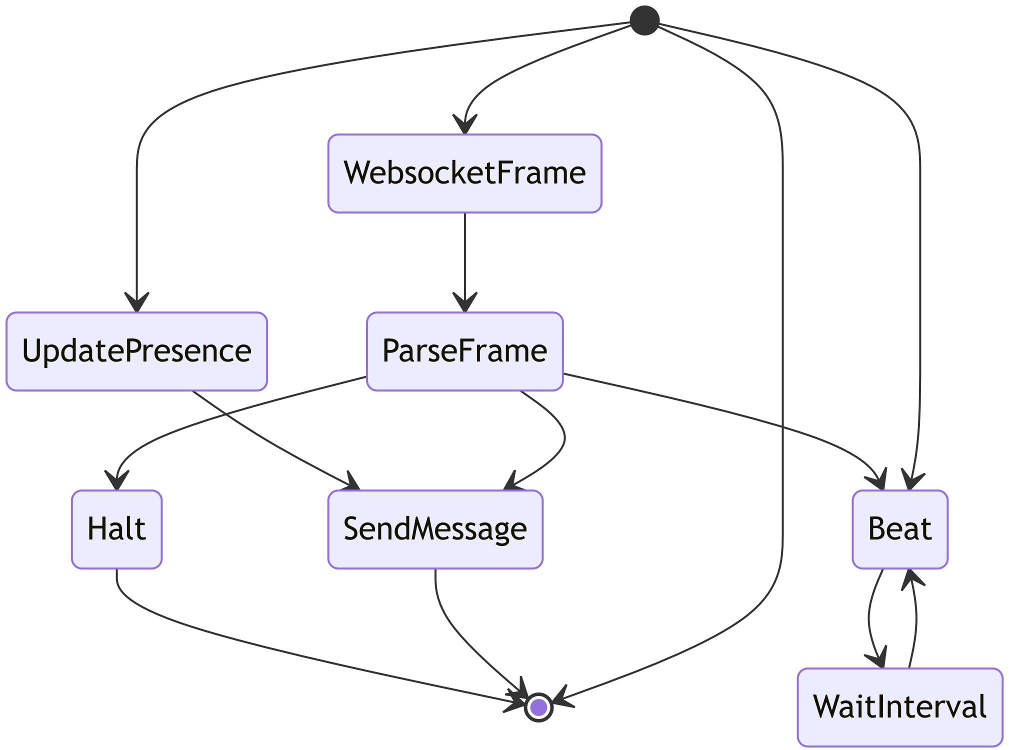
Discord's Gateway uses websockets, which we receive messages from and send messages to for real-time communication.



Open Websocket Connection
Wait for Message
Is message a websocket frame?
No
Yes
Is message a websocket closed?
No
Yes
Discard Message
Parse Frame
Failed
Success
Send to Event Loop

Event Loop

Simmer has an event loop actor which handles the messages from many places e.g. UpdatePresence, Halt, WebsocketFrame, etc


WebsocketFrame
UpdatePresence
ParseFrame
Halt
SendMessage
Beat
WaitInterval

```gleam
import gleam/erlang/process
import shimmer
import shimmer/handlers

pub fn main() {
  let handlers =
    handlers.new_builder()

  let client =
    shimmer.new("TOKEN")
    |> shimmer.connect(handlers)

  process.sleep_forever()
}
```

```
import gleam/erlang/process
import shimmer
import shimmer/handlers

pub fn main() {
  let handlers =
    handlers.new_builder()

  let client =
    shimmer.new("TOKEN")
    |> shimmer.connect(handlers)

  process.sleep_forever()
}
```

```gleam
import gleam/erlang/process
import shimmer
import shimmer/handlers

pub fn main() {
  let handlers =
    handlers.new_builder()

  let client =
    shimmer.new("TOKEN")
    |> shimmer.connect(handlers)

  process.sleep_forever()
}
```

```gleam
import gleam/erlang/process
import shimmer
import shimmer/handlers

pub fn main() {
  let handlers =
    handlers.new_builder()

  let client =
    shimmer.new("TOKEN")
    |> shimmer.connect(handlers)

  process.sleep_forever()
}
```

```
import gleam/erlang/process
import shimmer
import shimmer/handlers

pub fn main() {
  let handlers =
    handlers.new_builder()

  let client =
    shimmer.new("TOKEN")
    |> shimmer.connect(handlers)

  process.sleep_forever()
}
```

```
import gleam/erlang/process
import shimmer
import shimmer/handlers

pub fn main() {
  let handlers =
    handlers.new_builder()
    |> handlers.on_ready(fn(_event, _client) {})

  let client =
    shimmer.new("TOKEN")
    |> shimmer.connect(handlers)

  process.sleep_forever()
}
```

```
fn(_event, _client) {
  // TODO implement handler
}
```

```
import gleam/io

fn(event, _client) {
  let id = event.user.id

  io.println("Logged in as " <> id)
}
```

```
import gleam/io
import gleam/erlang/process
import shimmer
import shimmer/handlers

fn ready_handler(event, _client) {
  let id = event.user.id

  io.println("Logged in as " <> id)
}

pub fn main() {
  let handlers =
    handlers.new_builder()
    |> handlers.on_ready(ready_handler)

  // ...
}
```

```
import gleam/io

fn message_handler(event, _client) {
  let content = event.message.content
}
```

```gleam
import gleam/io

fn message_handler(event, _client) {
  let content = event.message.content

  case content {
    "!" <> command -> io.println("Command Received: " <> command)
    message -> io.println("Message Received: " <> message)
  }
}
```

```
import gleam/io
import shimmer/message

fn message_handler(event, client) {
  let content = event.message.content

  case content {
    "!ping" -> {
      io.println("Pong!")

      message.send(client, "Pong!", event.message.channel_id)
    }
    message -> io.println("Message Received: " <> message)
  }
}
```

```
import gleam/io
import gleam/erlang/process
import shimmer
import shimmer/handlers
import shimmer/message

// ...

pub fn main() {
  let handlers =
    handlers.new_builder()
    |> handlers.on_ready(ready_handler)
    |> handlers.on_message(message_handler)

  // ...
}
```

Recap

By building this simple discord bot we learnt a few things so lets recap:

- ✨ Basic Gleam
- 🚪 Discord's Gateway
- 🫂 Ping/Pong Bot

Any questions? 🙋

Thanks for Listening!