Building a framework

Membrane's journey to 1.0

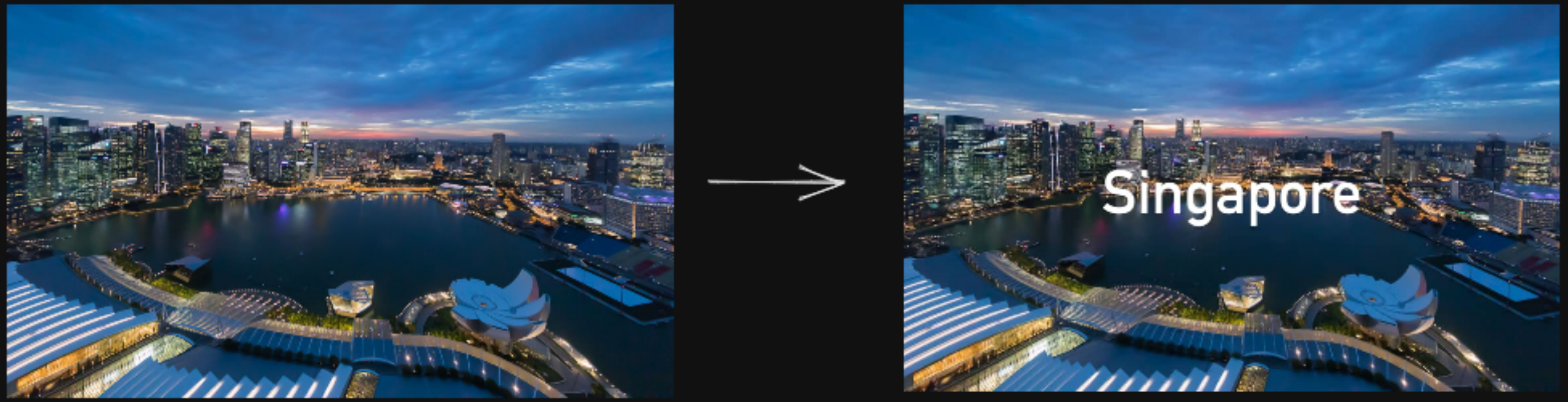
Mateusz Front

let's build a framework!

wait, what's a framework?

```elixir
{:ok, base_image} = Image.open("test/support/images/Singapore-2016-09-5887.jpg")
{:ok, singapore} = Text.new_from_string("Singapore", font_size: 100, font: "DIN Alternate")

base_image
|> Image.compose!(singapore, x: :center, y: :middle)
|> Image.write!("/Users/kip/Desktop/center_text.png")
```


Singapore

```elixir
defmodule MyAppWeb.ThermostatLive do
  # In Phoenix v1.6+ apps, the line below should be: use MyAppWeb, :live_view
  use Phoenix.LiveView

  @impl Phoenix.LiveView
  def render(assigns) do
    ~H"""
    Current temperature: <%= @temperature %>
    """
  end

  @impl Phoenix.LiveView
  def mount(_params, %{"current_user_id" => user_id}, socket) do
    temperature = Thermostat.get_user_reading(user_id)
    {:ok, assign(socket, :temperature, temperature)}
  end
end
```

```elixir
defmodule MyAppWeb.Router do
  use Phoenix.Router
  import Phoenix.LiveView.Router

  scope "/", MyAppWeb do
    live "/thermostat", ThermostatLive
  end
end
```

Framework

- Abstractions
- Implementations
- Mechanism that calls them

Framework

- Is more complex
- Is generic
- It's important to create abstractions properly

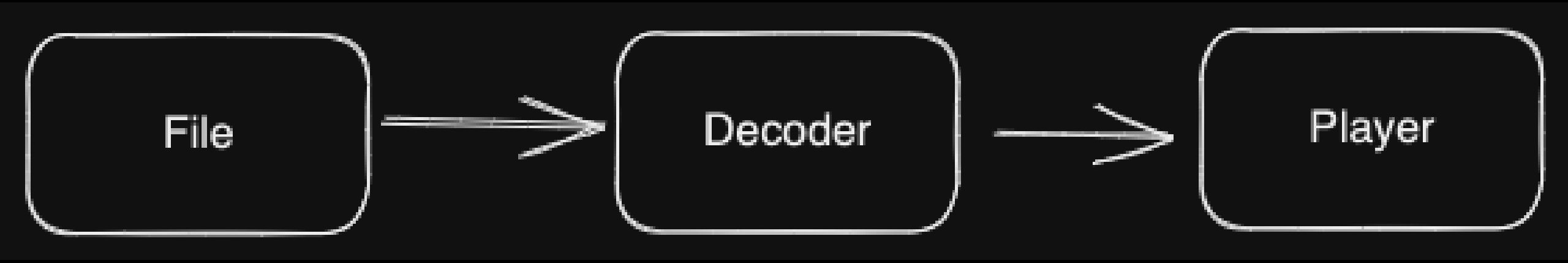
How to create good abstractions?

- Trust your feelings
- ~~Steal~~ get inspired by other tools
- Have some implementations and build abstractions on top of them

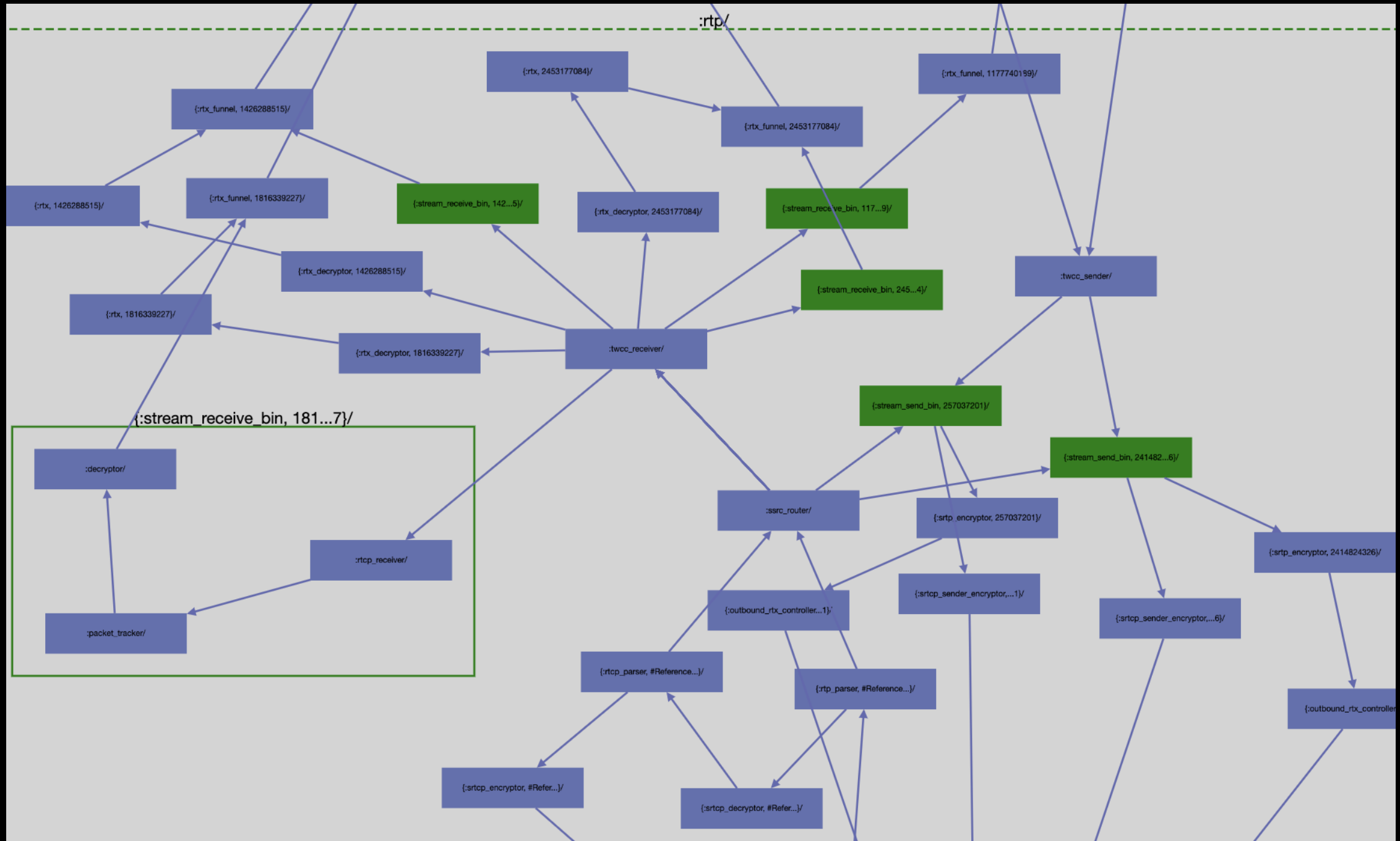
implementations -> abstraction

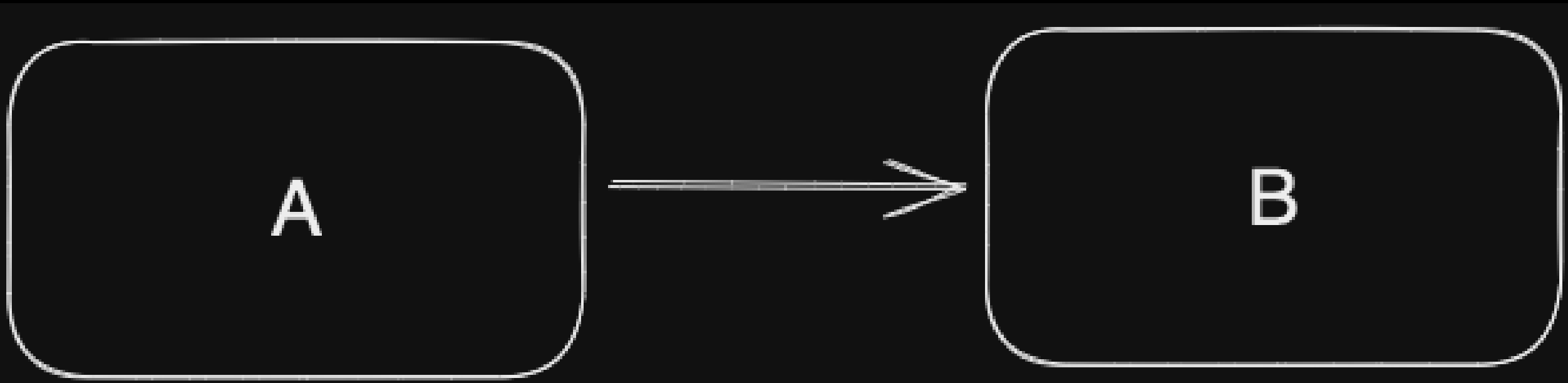
know your abstractions

Pipeline




Elements
LIGHTYEAR
Elements everywhere!
imgflip.com


:rtp/
{:rtx, 2453177084}/
{:rtx_funnel, 1177740199}/
{:rtx_funnel, 1426288515}/
{:rtx_funnel, 2453177084}/
{:rtx, 1426288515}/
{:rtx_funnel, 1816339227}/
{:stream_receive_bin, 142...5}/
{:rtx_decryptor, 2453177084}/
{:stream_receive_bin, 117...9}/
{:rtx_decryptor, 1426288515}/
{:stream_receive_bin, 245...4}/
:twcc_sender/
{:rtx, 1816339227}/
{:rtx_decryptor, 1816339227}/
:twcc_receiver/
{:stream_send_bin, 257037201}/
{:stream_receive_bin, 181...7}/
:decryptor/
{:stream_send_bin, 241482...6}/
:ssrc_router/
{:srtp_encryptor, 257037201}/
:rtcp_receiver/
{:srtp_encryptor, 2414824326}/
{:srtcp_sender_encryptor,...1}/
{:outbound_rtx_controller...1}/
:packet_tracker/
{:srtcp_sender_encryptor,...6}/
{:rtcp_parser, #Reference...}/
{:rtp_parser, #Reference...}/
{:srtcp_encryptor, #Refer...}/
{:srtcp_decryptor, #Refer...}/


A
B

Don't use your abstraction for code organisation

- Processes
- GenStage
- Membrane elements

How to create good abstractions in Elixir?

```elixir
defmodule MyAppWeb.ThermostatLive do
  # In Phoenix v1.6+ apps, the line below should be: use MyAppWeb, :live_view
  use Phoenix.LiveView

  @impl Phoenix.LiveView
  def render(assigns) do
    ~H"""
    Current temperature: <%= @temperature %>
    """
  end

  @impl Phoenix.LiveView
  def mount(_params, %{"current_user_id" => user_id}, socket) do
    temperature = Thermostat.get_user_reading(user_id)
    {:ok, assign(socket, :temperature, temperature)}
  end
end
```

```elixir
defmodule MyAppWeb.Router do
  use Phoenix.Router
  import Phoenix.LiveView.Router

  scope "/", MyAppWeb do
    live "/thermostat", ThermostatLive
  end
end
```

Callback-based interfaces

- Intuitive
- Easy to implement
- Limited

handle_info(msg, state) (optional) </>

Specs

```
handle_info(msg :: :timeout | term(), state :: term()) ::
  {:noreply, new_state}
  | {:noreply, new_state, timeout() | :hibernate | {:continue, term()}}
  | {:stop, reason :: term(), new_state}
when new_state: term()
```

Invoked to handle all other messages.

gen_stage
▾ v1.2.1

PAGES MODULES

ConsumerSupervisor
GenStage
Sections

handle_info(message, state)

```
@callback handle_info(message :: term(), state :: term()) ::
  {:noreply, [event], new_state}
  | {:noreply, [event], new_state, :hibernate}
  | {:stop, reason :: term(), new_state}
when new_state: term(), event: term()
```

Invoked to handle all other messages.

Phoenix LiveView

▾ v0.18.18

GUIDES MODULES

Phoenix.Component

handle_info(msg, socket)

```
@callback handle_info(msg :: term(), socket :: Phoenix.LiveVie
  {:noreply, Phoenix.LiveView.Socket.t()}
```

Invoked to handle messages from other Elixir processes.

handle_other(message, context, state) </>

(optional)

```
@callback handle_other(
  message :: any(),
  context ::
Membrane.Pipeline.CallbackContext.Other.t(),
  state :: state_t()
) :: callback_return_t()
```

Callback invoked when pipeline receives a message that is not recognized as an internal membrane message.

Callback-based interfaces

- Intuitive
- Easy to implement
- Limited

```elixir
@impl true
def handle_event(:output, %ReceiverReport.StatsEvent{stats: stats}, _ctx, state) do ...

@impl true
def handle_event(:output, %Membrane.KeyframeRequestEvent{}, _ctx, state) do ...

@impl true
def handle_event(:output, %RTP.RetransmissionRequestEvent{packet_ids: ids}, _ctx, state) do ...
```

```elixir
@impl true
def handle_event(:output, %ReceiverReport.StatsEvent{stats: stats}, _ctx, state) do ...

@impl true
def handle_event(:output, %Membrane.KeyframeRequestEvent{}, _ctx, state) do ...

@impl true
def handle_event(:output, %RTP.RetransmissionRequestEvent{packet_ids: ids}, _ctx, state) do ...

@impl true
def handle_event(pad, event, ctx, state), do: super(pad, event, ctx, state)
```

Macro-based interfaces

- Need to learn from scratch
- Difficult to implement
- Flexible
- Optimisation possibilities

How to callback?

- Actions
- Contexts

```elixir
@impl true
def handle_stream_format(:input, format, _context, state) do
  buffer = %Buffer{payload: create_header(format)}
  # subtracting 8 bytes as header length doesn't include "RIFF" and `file_length` fields
  state = Map.put(state, :header_length, byte_size(buffer.payload) - 8)

  {[stream_format: {:output, format}, buffer: {:output, buffer}], state}
end
```

```elixir
@impl true
def handle_info({:portaudio_payload, payload}, %{playback: :playing}, state) do
  {[buffer: {:output, %Buffer{payload: payload}}], state}
end
```

Plug-in based architecture

- Core (Abstractions, core implementation)
- Plugins (Implementations)

Monorepo or multirepo?

Single package or multiple packages?

Multiple packages

- Better compilation times
- No need to have all native dependencies installed
- Allows for removing deprecated code
- Well-defined relationships between plugins
- Enforces modularity

Multiple packages - drawbacks

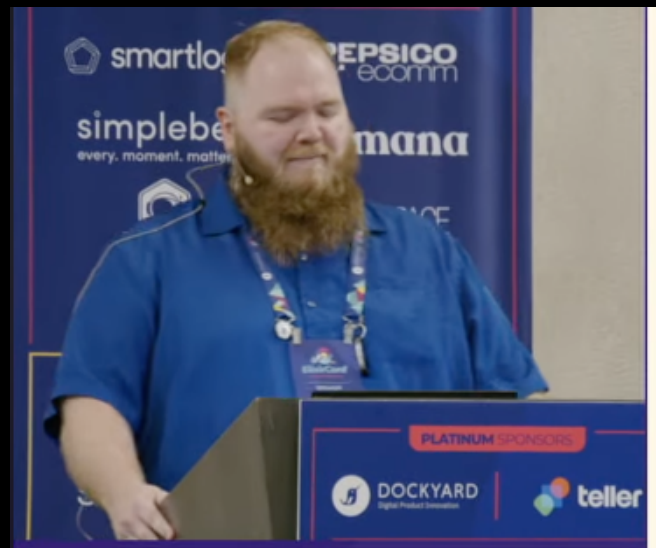
- Dependency hell
- Maintenance cost
- Increases entry level

Membrane Framework

Advanced multimedia processing framewor

85 followers https://membrane.stream/

Overview Repositories 144 Projects



MAKING ELIXIR HONK

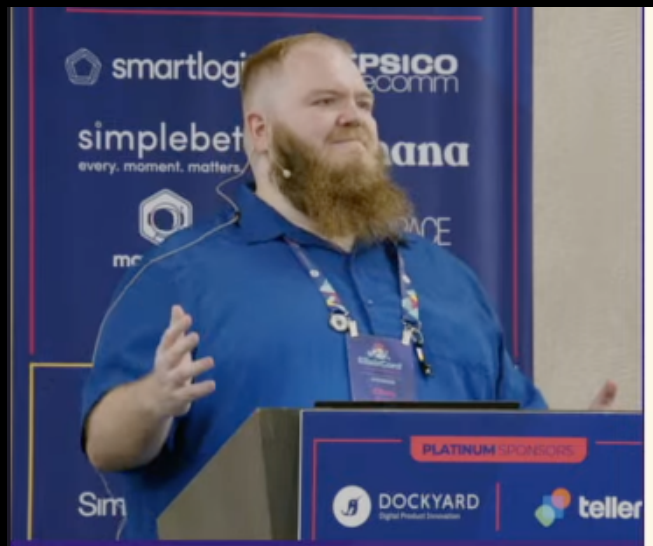
IMPLEMENTING AN ELIXIR AUDIO LIBRARY IN ZIG

@crertel

ElixirConf®

ELIXIR 10TH ANNIVERSARY

ElixirConf 2022 - Chris Ertel - Making Elixir Honk: Implementing An Elixir Audio Library In Zig



What are our options?

Use membrane.

...too powerful!

ElixirConf®

ELIXIR 10TH ANNIVERSARY

ElixirConf 2022 - Chris Ertel - Making Elixir Honk: Implementing An Elixir Audio Library In Zig

Decrease entry level

- Gigachad package
- Demos (in Livebook?)
- Open source products

Jellyfish

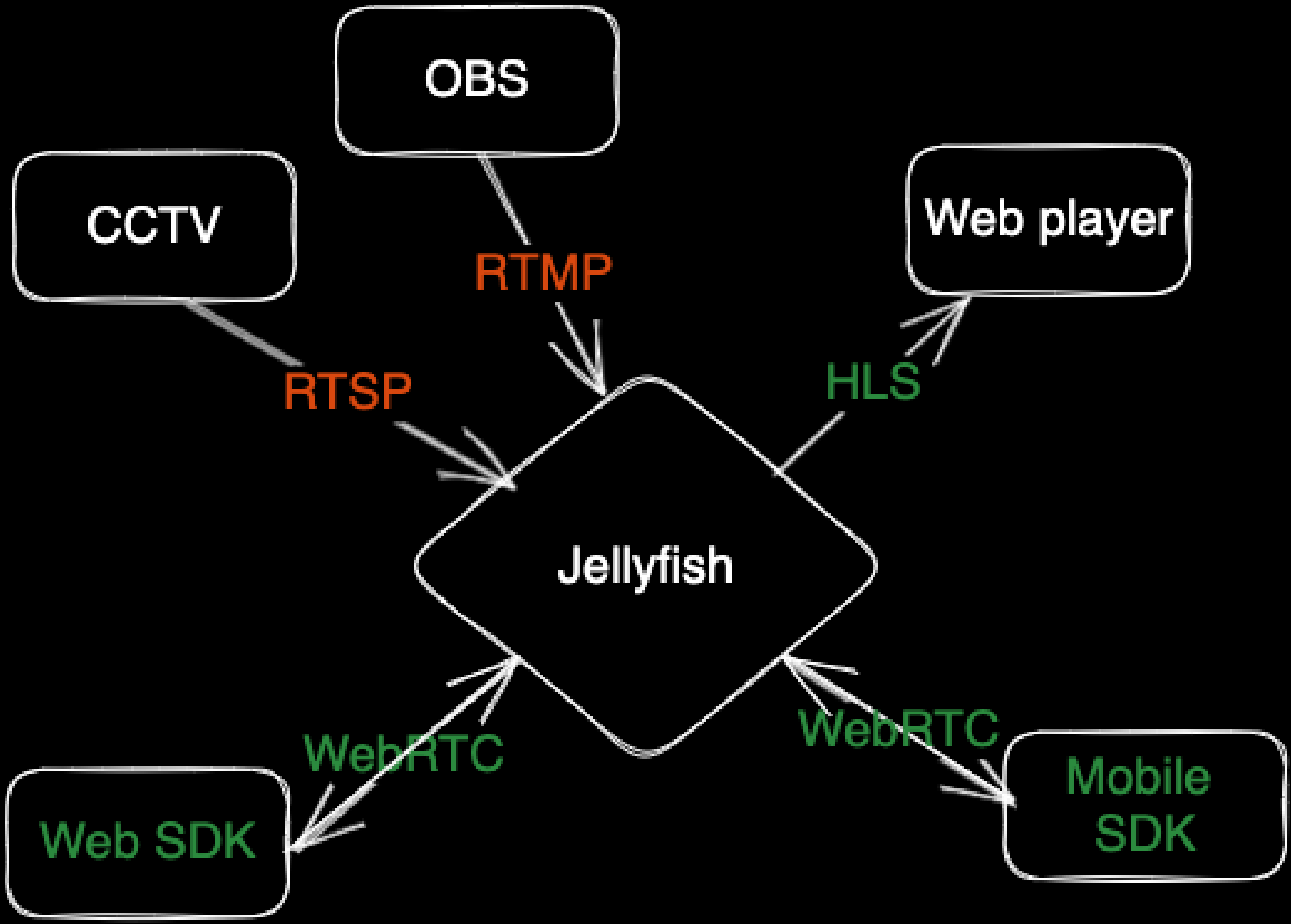
Membrane @ElixirMembrane · 3h

Hi everyone!

After about 3 months of really hard work 👷 we have released the very first version of Jellyfish Media Server 🎉

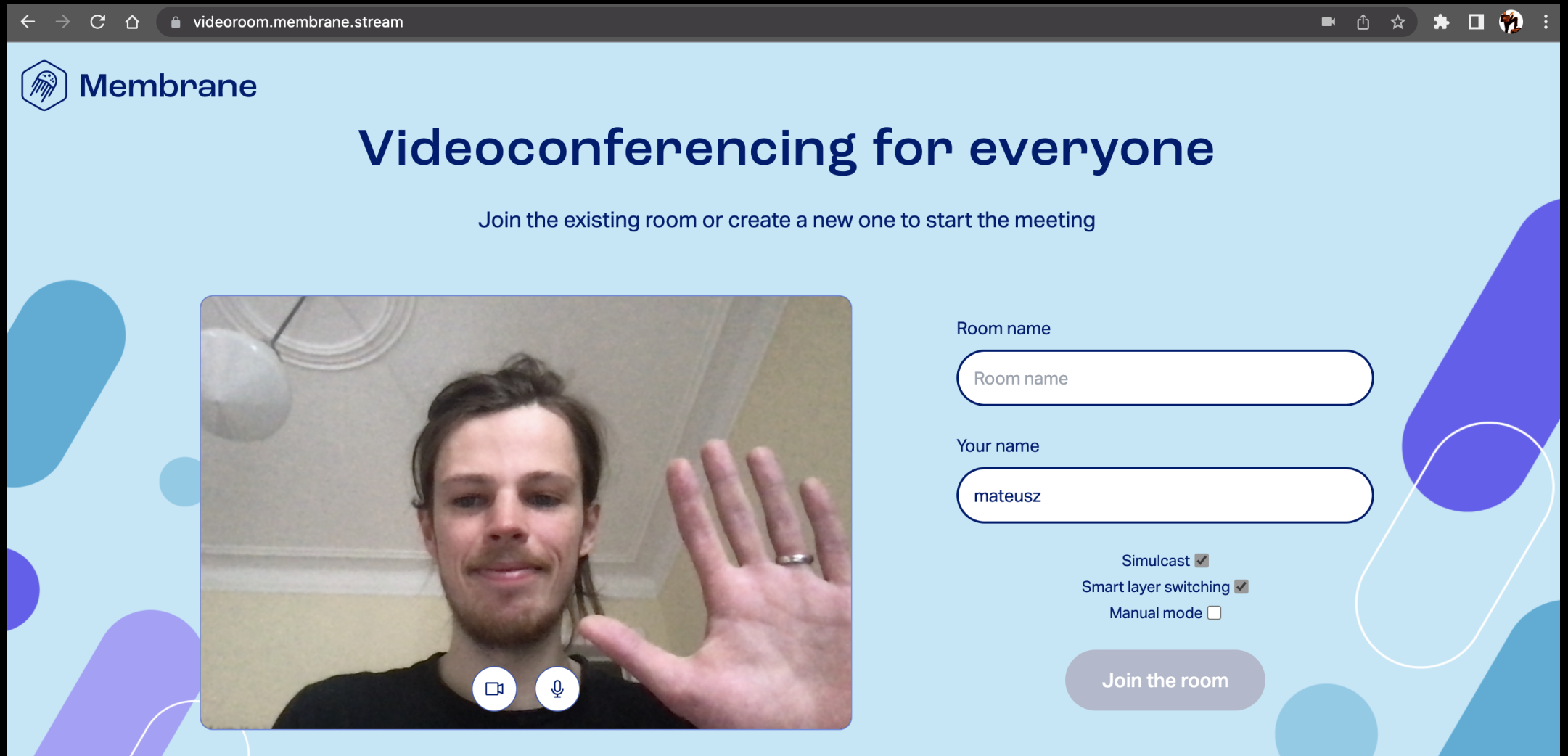
The best starting point is the documentation available here: jellyfish-dev.github.io/jellyfish-docs/

#MyElixirStatus #multimedia #WebRTC #React #Typescript


OBS
CCTV
Web player
RTMP
RTSP
HLS
Jellyfish
WebRTC
WebRTC
Web SDK
Mobile SDK

github.com/jellyfish-dev

Products


videoroom.membrane.stream
Membrane
Videoconferencing for everyone
Join the existing room or create a new one to start the meeting
Room name
Room name
Your name
mateusz
Simulcast
Smart layer switching
Manual mode
Join the room

What's 1.0?

4. Major version zero (0.y.z) is for initial development. Anything MAY change at any time. The public API SHOULD NOT be considered stable.
5. Version 1.0.0 defines the public API. The way in which the version number is incremented after this release is dependent on this public API and how it changes.

semver.org

When to 1.0?

- If your software is being used in production,
- If you have a stable API on which users have come to depend,
- If you’re worrying a lot about backwards compatibility,

you should probably already be 1.0.0.

semver.org/#faq

Membrane & 1.0

- v0.11 was about API
- v1.0.0-rc0 was about missing parts
  ...and API
- v0.12 will be about compatibility

don't mix features with breaking changes

Membrane Core 1.0 - when?

Soon™

 @ElixirMembrane

 @ElixirMembrane

Thanks!