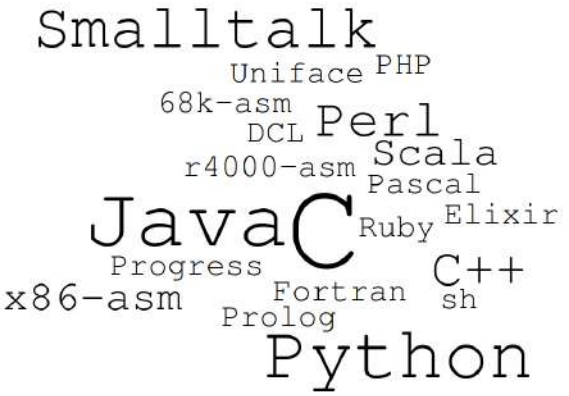
Clixir

(or: How to abuse Elixir macros for fun and profit)

`http://bit.ly/2019clixirSF`

Intro: yours truly



Intro: PagerDuty

Background


24 - 380VAC
DMWD
SSR-10 DA
Solid State Module
Made in china
Input
3 - 32VDC

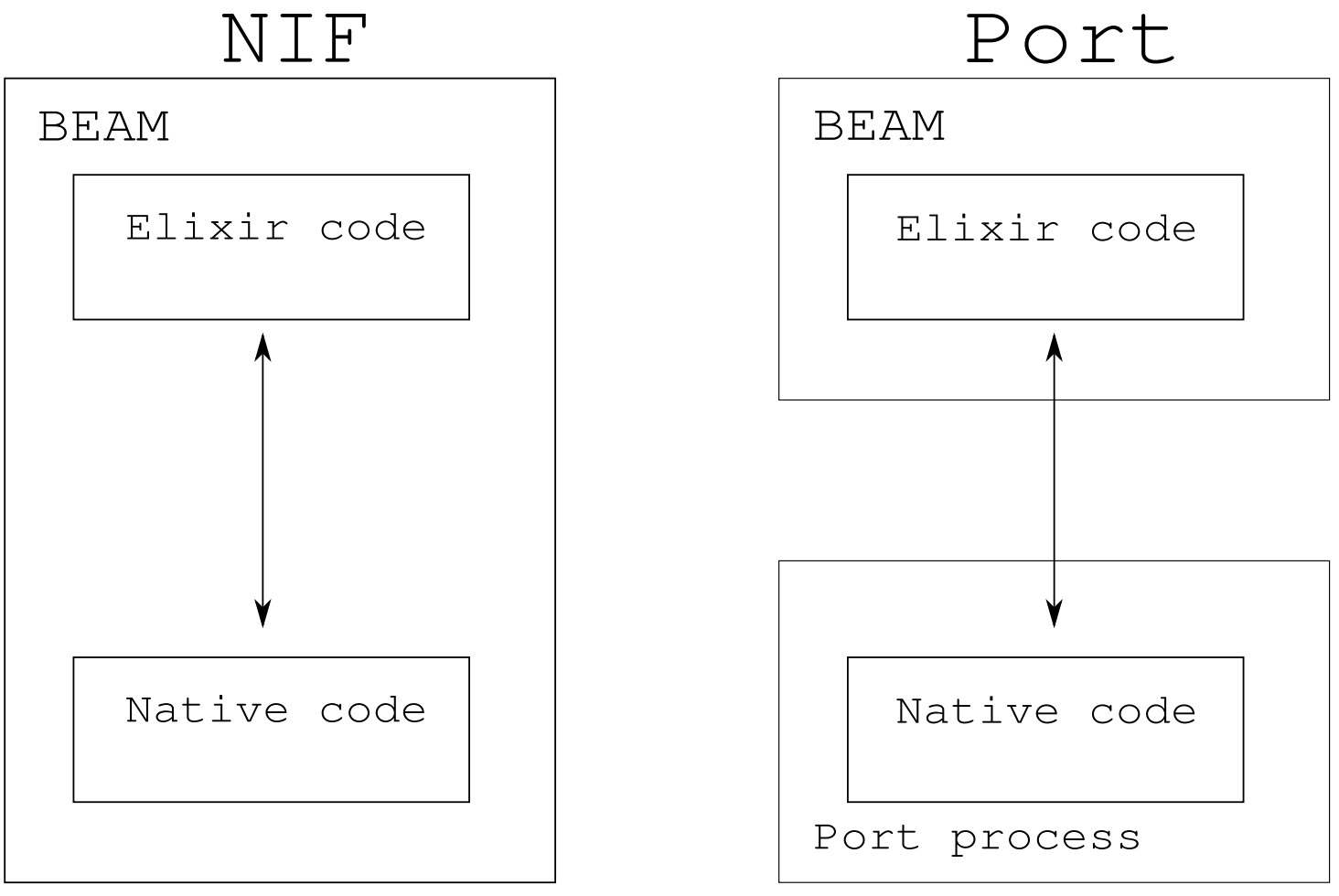
Requirements

Requirements

Requirements

pagerduty

NIFs and Ports



Using ports: sending arguments

```
bytes = :erlang.term_to_binary("world")
Port.command(my_port, bytes)
```

Using ports: decoding arguments

```
static void hello(const char *buf, unsigned short len) {
    char message[65536];
    long message_len;
    int index = 0;
    assert(ei_decode_binary(buf, &index, message, &message_len) == 0);
    message[message_len] = '\0';
    fprintf(stderr, "Hello, %s!\n", message);
}
```

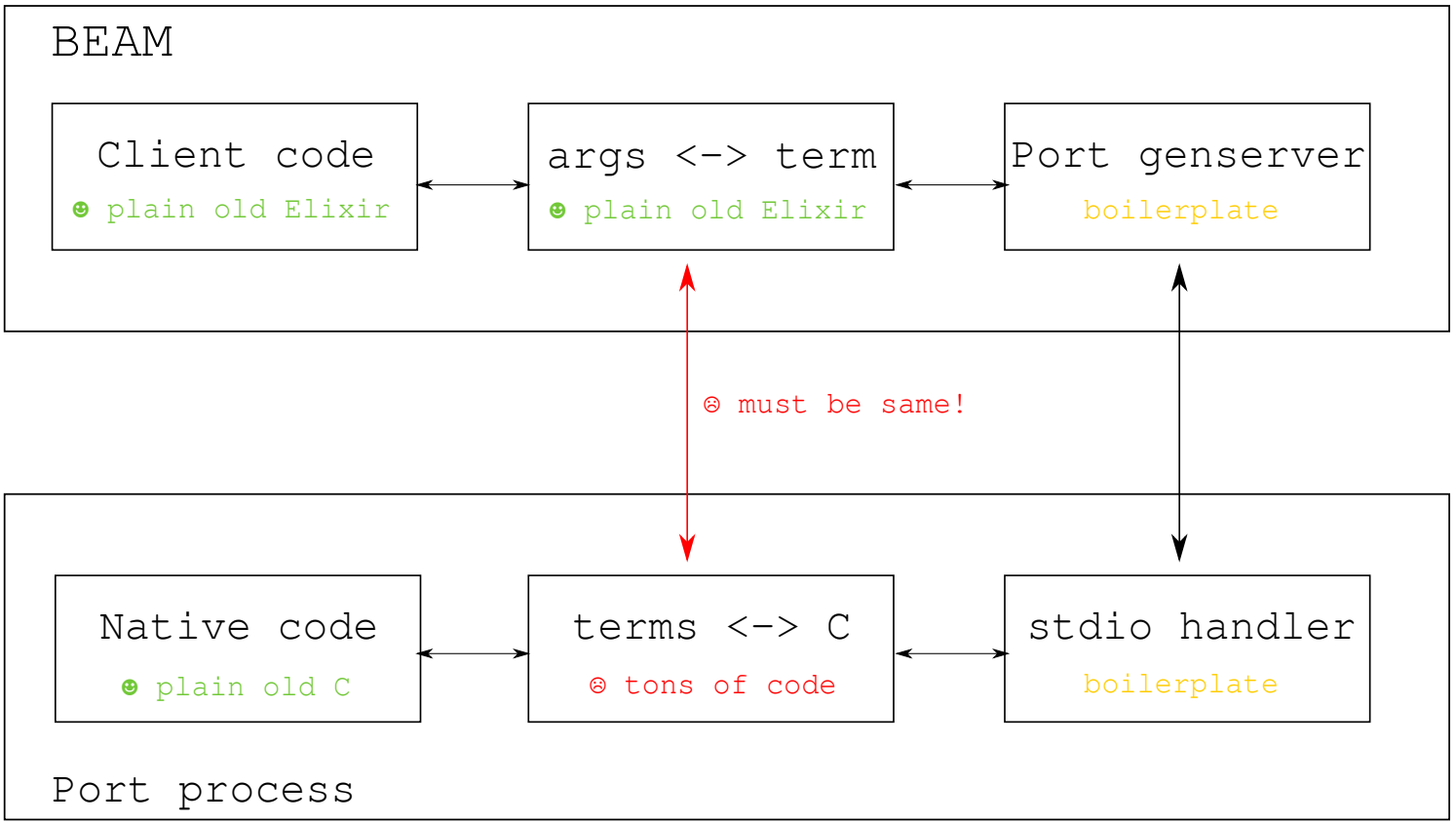
Using ports: encoding responses

```
static void make_window() {
   char response[65536];
   int response_index = 0;
   GLFWwindow *window = ...; // actually make window
   ei_encode_version(response, &response_index);
   ei_encode_tuple_header(response, &response_index, 2);
   ei_encode_pid(response, &response_index, &pid);
   ei_encode_tuple_header(response, &response_index, 2);
   ei_encode_atom(response, &response_index, "ok");
   ei_encode_longlong(response, &response_index, (long long) window);
   fwrite(response, response_index, 1, stdout);
}
```

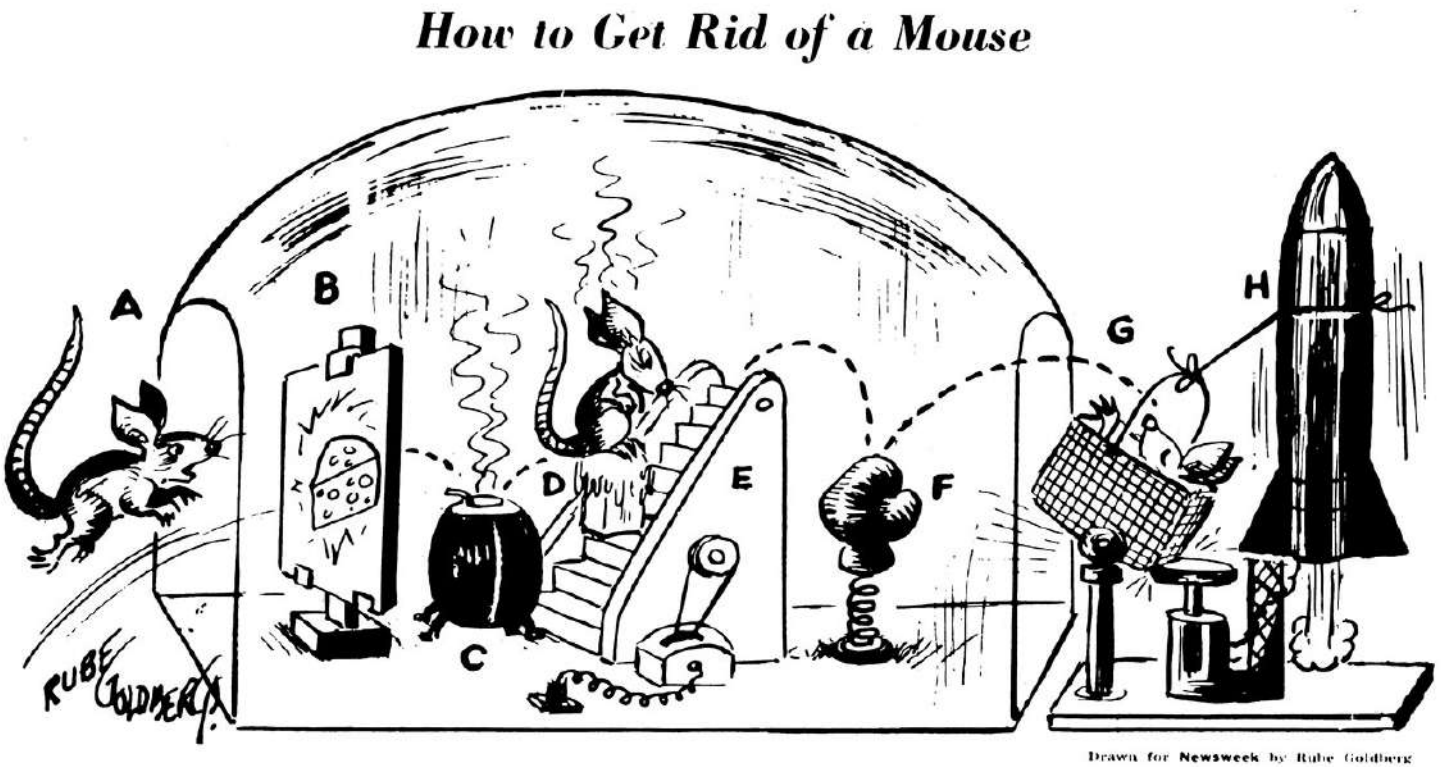
Using ports: decoding responses

```
{:ok, handle} = :erlang.binary_to_term(bytes)
```

Putting it all together

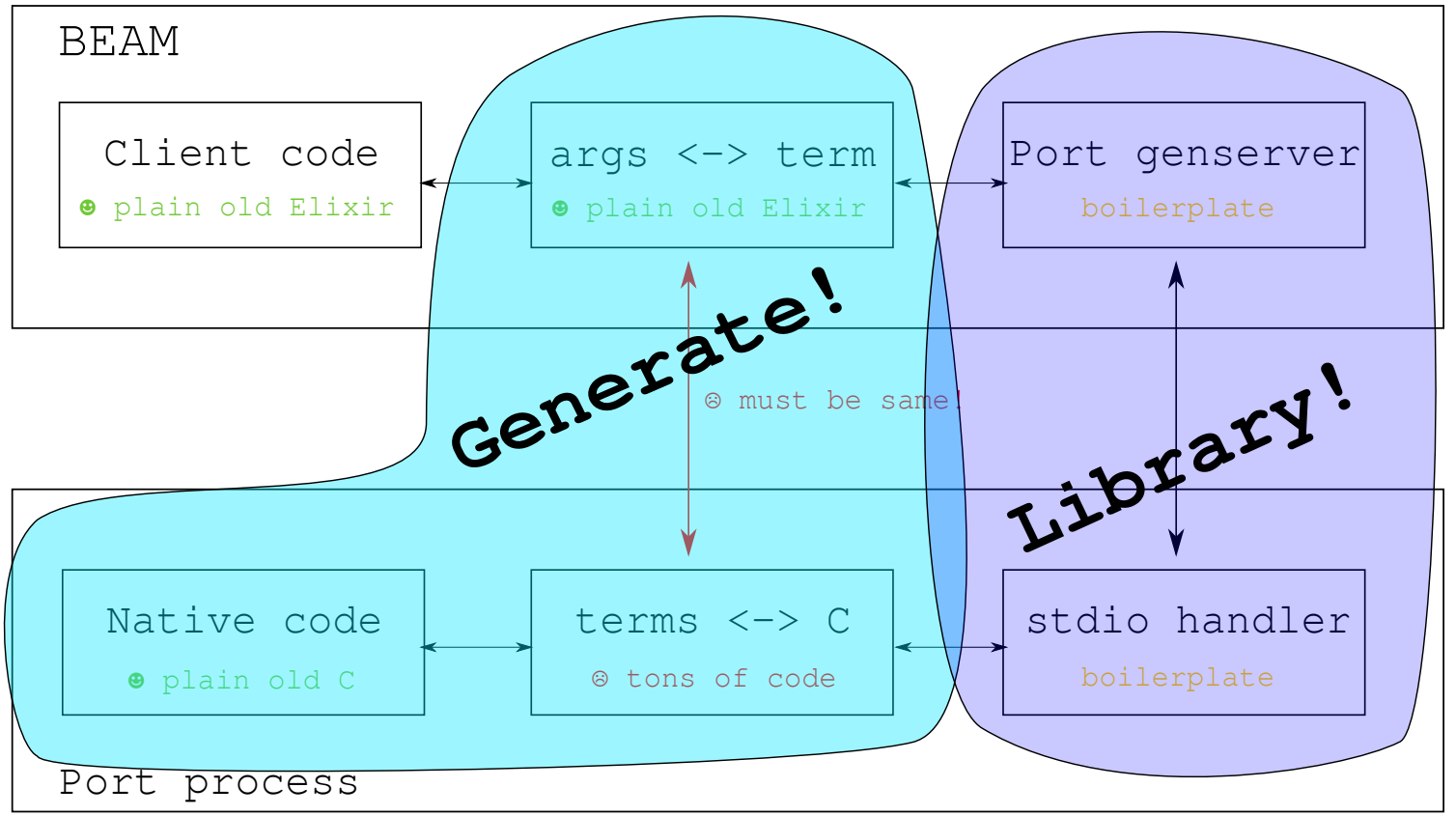


Let's go wild



The best mousetrap by Rube Goldberg: Mouse (A) dives for painting of cheese (B), goes through canvas and lands on hot stove (C). He jumps on cake of ice (D) to cool off. Moving escalator (E) drops him on boxing glove (F) which knocks him into basket (G) setting off miniature rocket (H) which takes him to the moon.

Let's go wild


BEAM
Client code
☻ plain old Elixir
args <-> term
☻ plain old Elixir
Port genserver
boilerplate
Generate!
☹ must be same!
Library!
Native code
☻ plain old C
terms <-> C
☹ tons of code
stdio handler
boilerplate
Port process

An Example

```
defmodule Example do
  use Clixir

  @clixir_header "example"

  def_c hello(message) do
    cdecl "char *": message

    # I am C code!
    fprintf(stderr, "Hello, %s!\n", message)
  end
end

...
# Somewhere in your application code
Example.hello("world")
```

Stepping through: Clixir startup

```
def init([]) do
  app = Application.get_env(:clixir, :application)
  Logger.info("Starting clixir process from application #{app}")
  clixir_bin = Application.app_dir(app, "priv/clixir")
  port = Port.open({:spawn, clixir_bin},
    [{:packet, 2}, :binary, :exit_status])
  {:ok, %State{port: port}}
end
```

Stepping through: Elixir code

```
def hello(message) do
  Clixir.Server.send_command(Clixir.Server, {:Elixir_Example_hello, message})
end
```

Stepping through: Clixir server

```elixir
def send_command(pid, command) do
  :ok = GenServer.cast(pid, {:send, command})
end

def handle_cast({:send, command}, state) do
  Logger.debug("sending message #{inspect command}")
  bytes = :erlang.term_to_binary(command)
  Port.command(state.port, bytes)
  {:noreply, state}
end
```

Stepping through: Clixir dispatch

```
void clixir_read_loop() {
  char buffer[65536];
  while (1) {
    // simplified...
    unsigned short size = read_packet_size();
    read(STDIN_FILENO, buffer, size);
    handle_command(buffer, size);
  }
}
```

Stepping through: Clixir dispatch

```
void handle_command(command, length) {
  // simplified ...
  ei_term term = decode_term(command);
  dispatch_command(term);
}
```

Sidebar: gperf

A perfect hash function is a hash function that has no collisions.

- `gperf` generates a perfect hash function from input

```
"Elixir_Example_hello", _dispatch_Elixir_Example_hello
```

- maps the name into a pointer to the generated dispatch code in 2 memory lookups

Stepping through: Generated C

```
static void _dispatch_Elixir_Example_hello(const char *buf, unsigned short len,
                                           int *index) {
    char message[BUF_SIZE];
    long message_len;
    assert(ei_decode_binary(buf, index, message, &message_len) == 0);
    message[message_len] = '\0';

    // recognize this bit??
    fprintf(stderr, "Hello, %s!\n", message);
}
```

Returning values

```
def_c abs(value, pid) do
  cdecl long: [value, result]
  cdecl erlang_pid: pid

  if value < 0 do
    result = value * -1
  else
    result = value
  end
  {pid, {:ok, result}}
end
```

Returning values

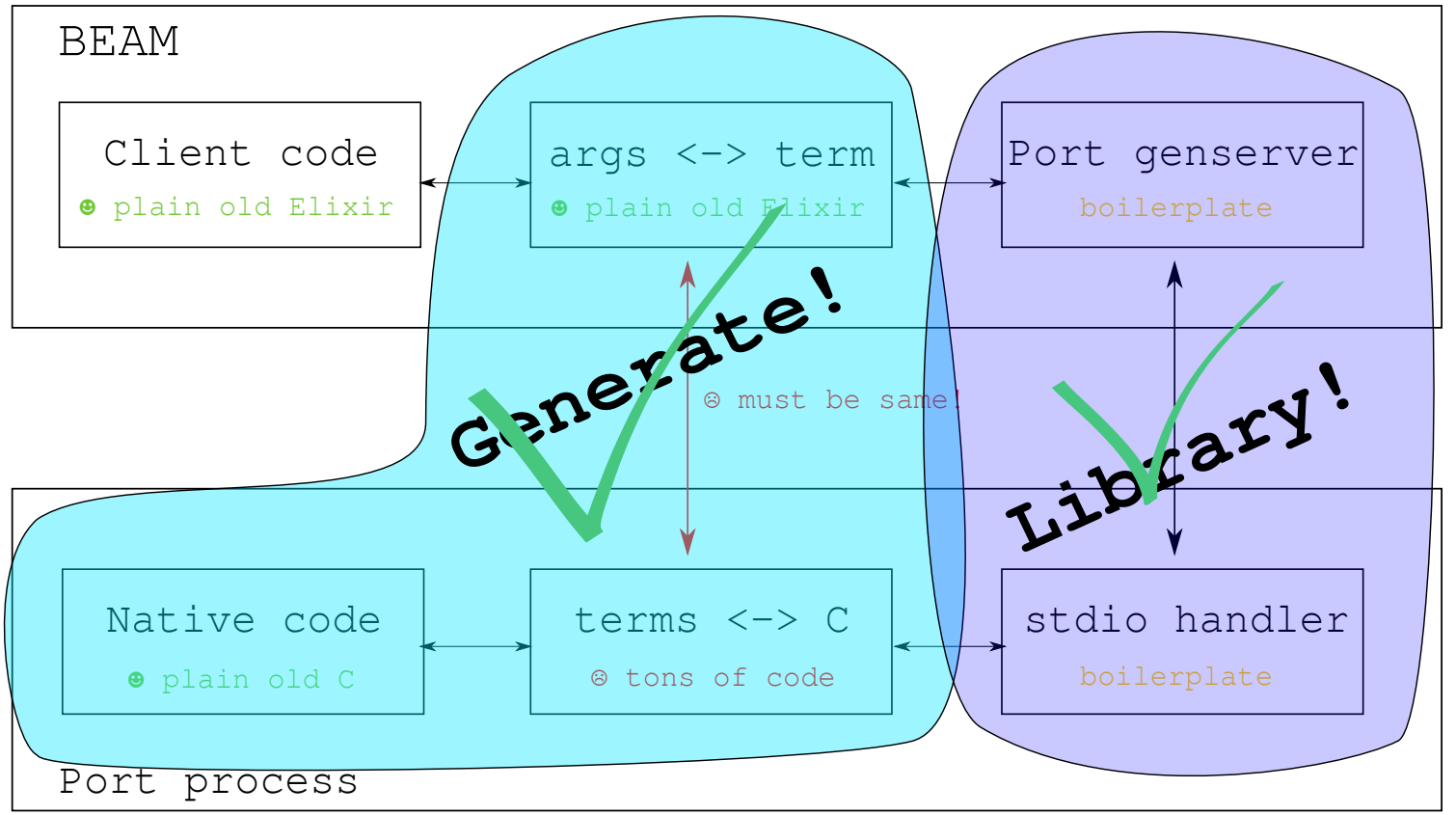
```
static void _dispatch_Elixir_Example_abs(const char *buf,
                                         unsigned short len, int *index) {
    erlang_pid pid; long result; long value;
    assert(ei_decode_long(buf, index, &value) == 0);
    assert(ei_decode_pid(buf, index, &pid) == 0);
    if (value < 0) {
        result = value * -(1);
    } else {
        result = value;    }
    char response[BUF_SIZE];
    int response_index = 0;
    ei_encode_version(response, &response_index);
    ei_encode_tuple_header(response, &response_index, 2);
    ei_encode_pid(response, &response_index, &pid);
    ei_encode_tuple_header(response, &response_index, 2);
    ei_encode_atom(response, &response_index, "ok");
    ei_encode_long(response, &response_index, result);
    write_response_bytes(response, response_index);
}
```

Returning values

```
def handle_info({_port, {:data, data}}, state) do
  stuff = :erlang.binary_to_term(data)
  Logger.debug("Received response #{inspect stuff}")
  dispatch_message(stuff)
  {:noreply, state}
end

defp dispatch_message({pid, response}) when is_pid(pid) do
  send(pid, response)
end
...
```

Sync = async request + async response, very Erlang!

Recap


BEAM
Client code
☻ plain old Elixir
args <-> term
☻ plain old Elixir
Port genserver
boilerplate
Generate!
☹ must be same!
Library!
Native code
☻ plain old C
terms <-> C
☹ tons of code
stdio handler
boilerplate
Port process

Uderzo: GenRenderer

```elixir
defmodule Demo do
  @fps 100
  use Uderzo.GenRenderer
  def run do
    {:ok, boids} = Boids.start_link()
    Uderzo.GenRenderer.start_link(__MODULE__, "Uderzo Boids", 800, 600, @fps, boids
    Process.sleep(:infinity)
  end
  def init_renderer(boids) do
    {:ok, boids}
  end
  def render_frame(win_width, win_height, _mx, _my, boids) do
    BoidsUi.paint_background(win_width, win_height)
    Enum.each(Boids.get_all(boids), fn {x, y, v} ->
      BoidsUi.render(win_width, win_height, x, y, v)
    end)
    {:ok, boids}
  end
end
```

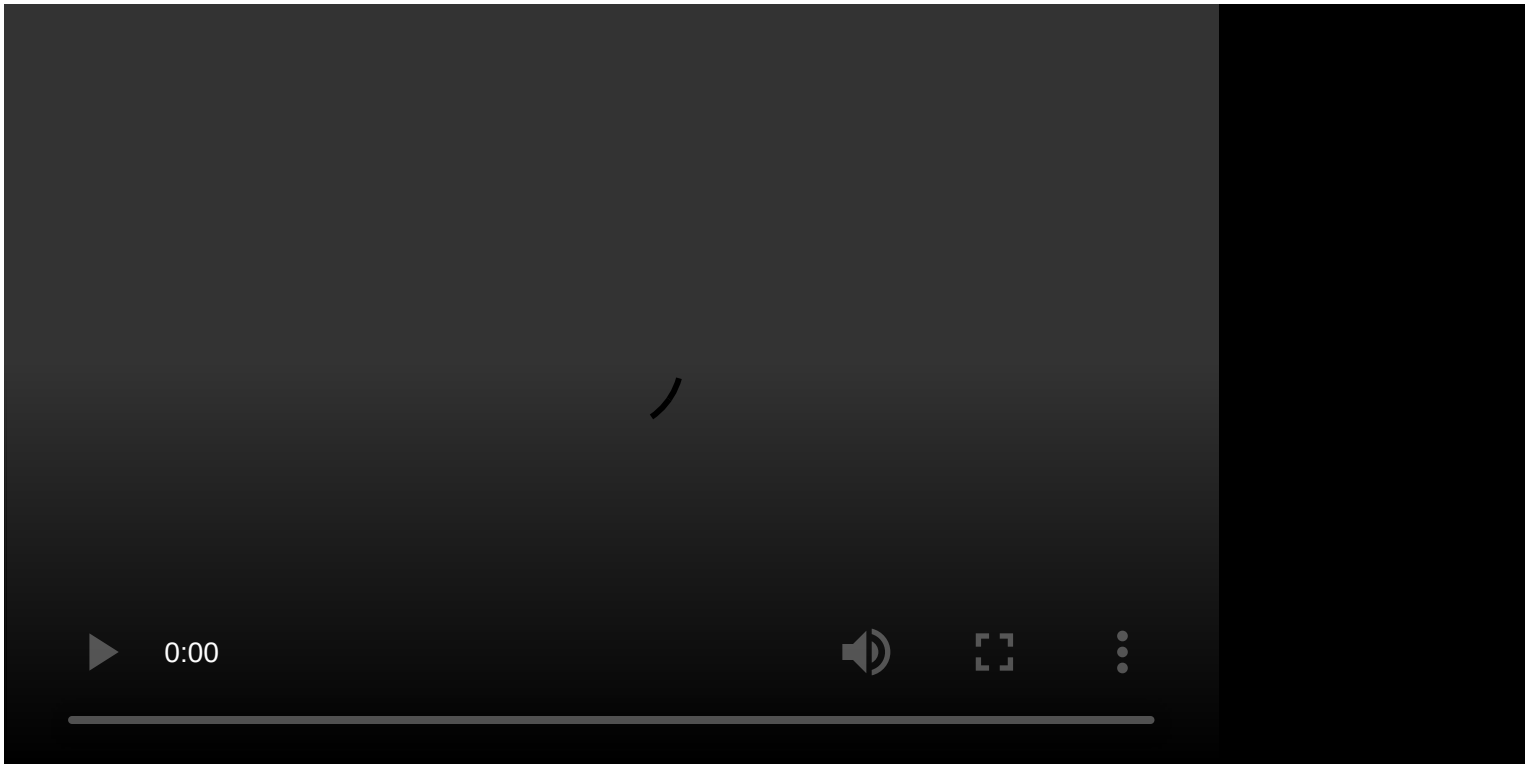
Uderzo: Clixir rendering code

```
def_c paint_background(win_width, win_height) do
  cdecl double: [win_width, win_height]
  cdecl "NVGpaint": air
  cdecl "NVGpaint": sun

  air = nvgLinearGradient(vg, win_width / 2, 0, win_width / 2, win_height, nvgRGBA(
  nvgBeginPath(vg)
  nvgRect(vg, 0, 0, win_width, win_height)
  nvgFillPaint(vg, air)
  nvgFill(vg)

  sun = nvgRadialGradient(vg, win_width * 0.8, win_height * 0.2, 0.04 * win_width,
, 0, 0))

  nvgBeginPath(vg)
  nvgCircle(vg, win_width * 0.8, win_height * 0.2, 0.1 * win_width)
  nvgFillPaint(vg, sun)
  nvgFill(vg)
end
```

Demo time!

```
cd ~/mine/palapa/boids_ui
mix run -e Demo.run
```

So what about...


0:00

That's all! Questions?

Presentation: http://bit.ly/2019clixirSF

Code: https://github.com/cdegroot/uderzo_poncho

Boids Demo:

https://github.com/cdegroot/palapa/tree/master/boids_ui

`cdegroot @`