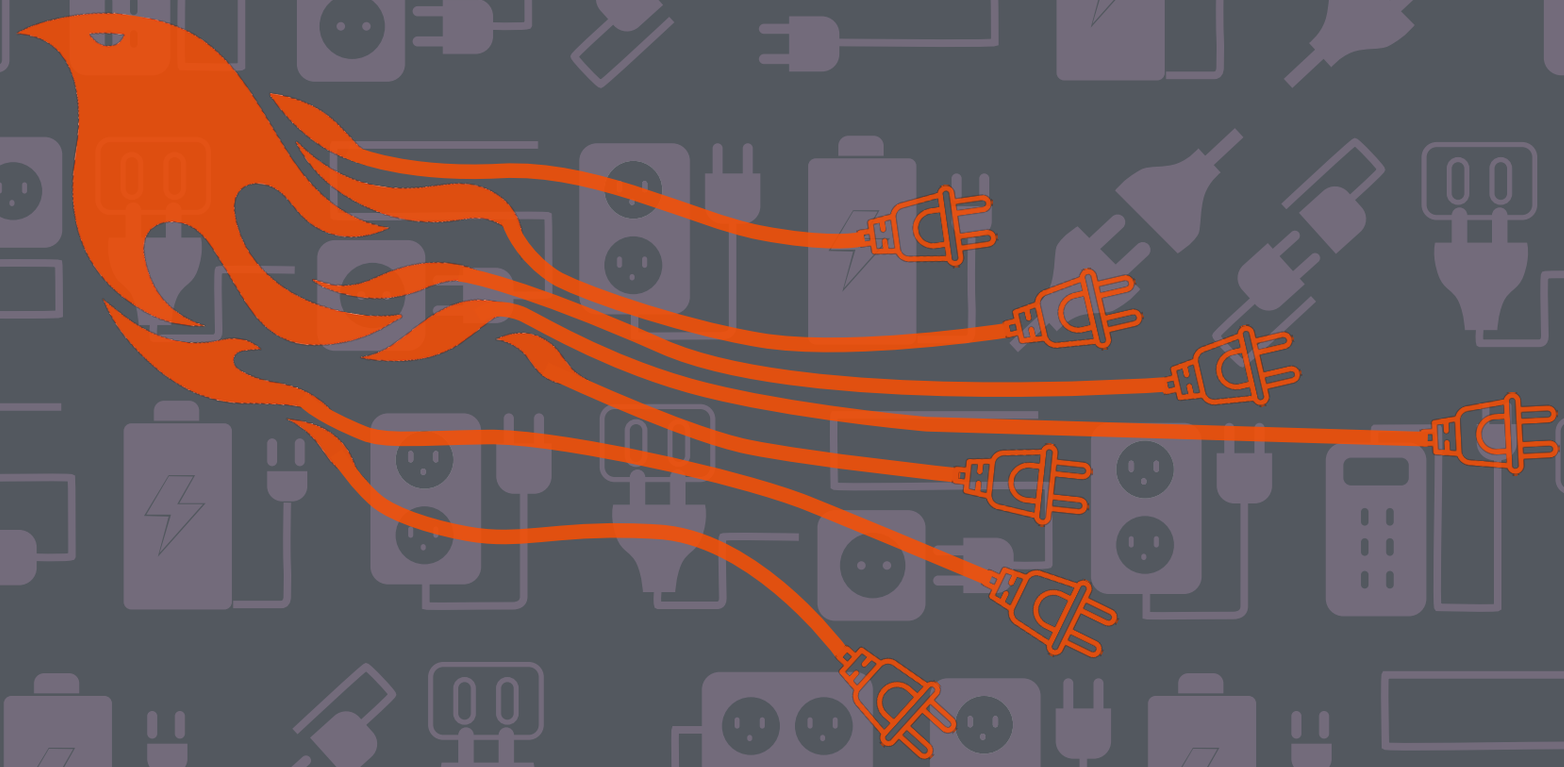
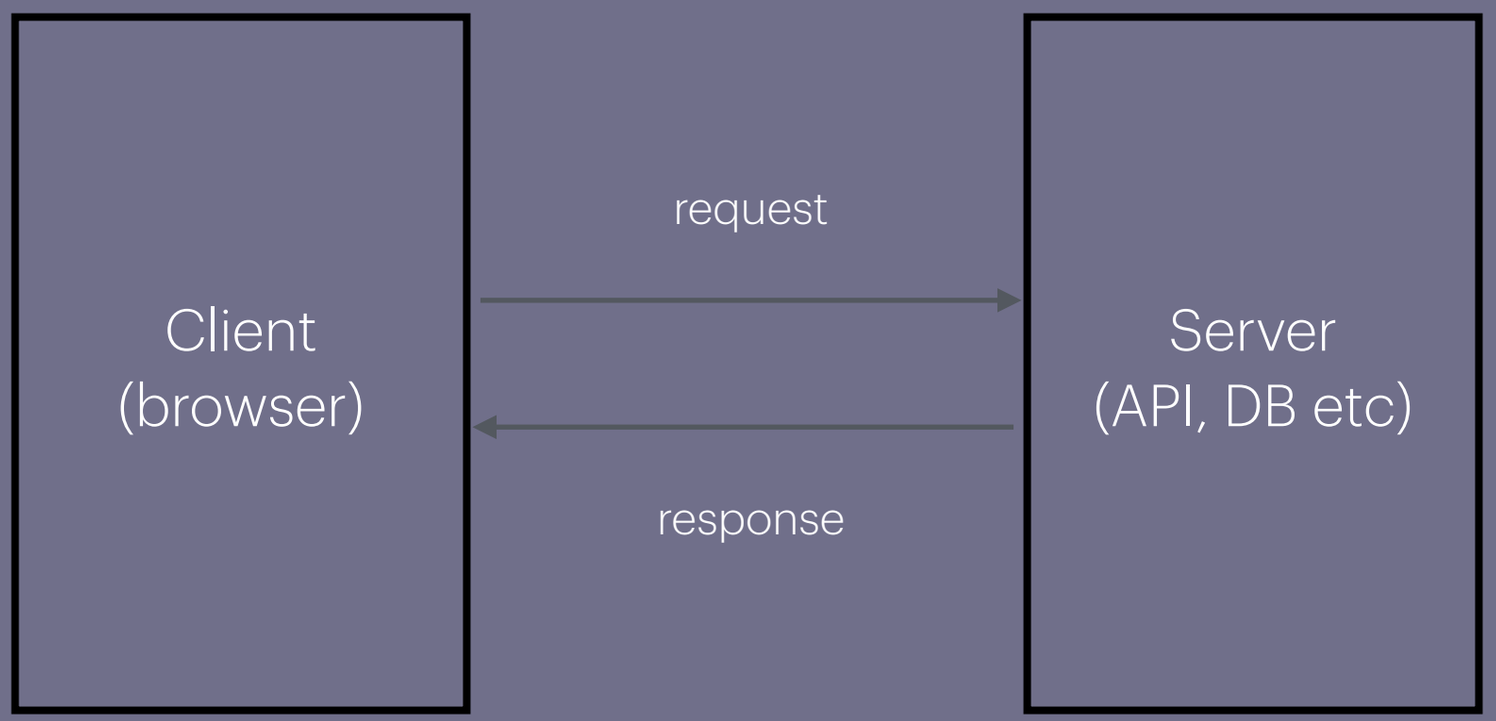
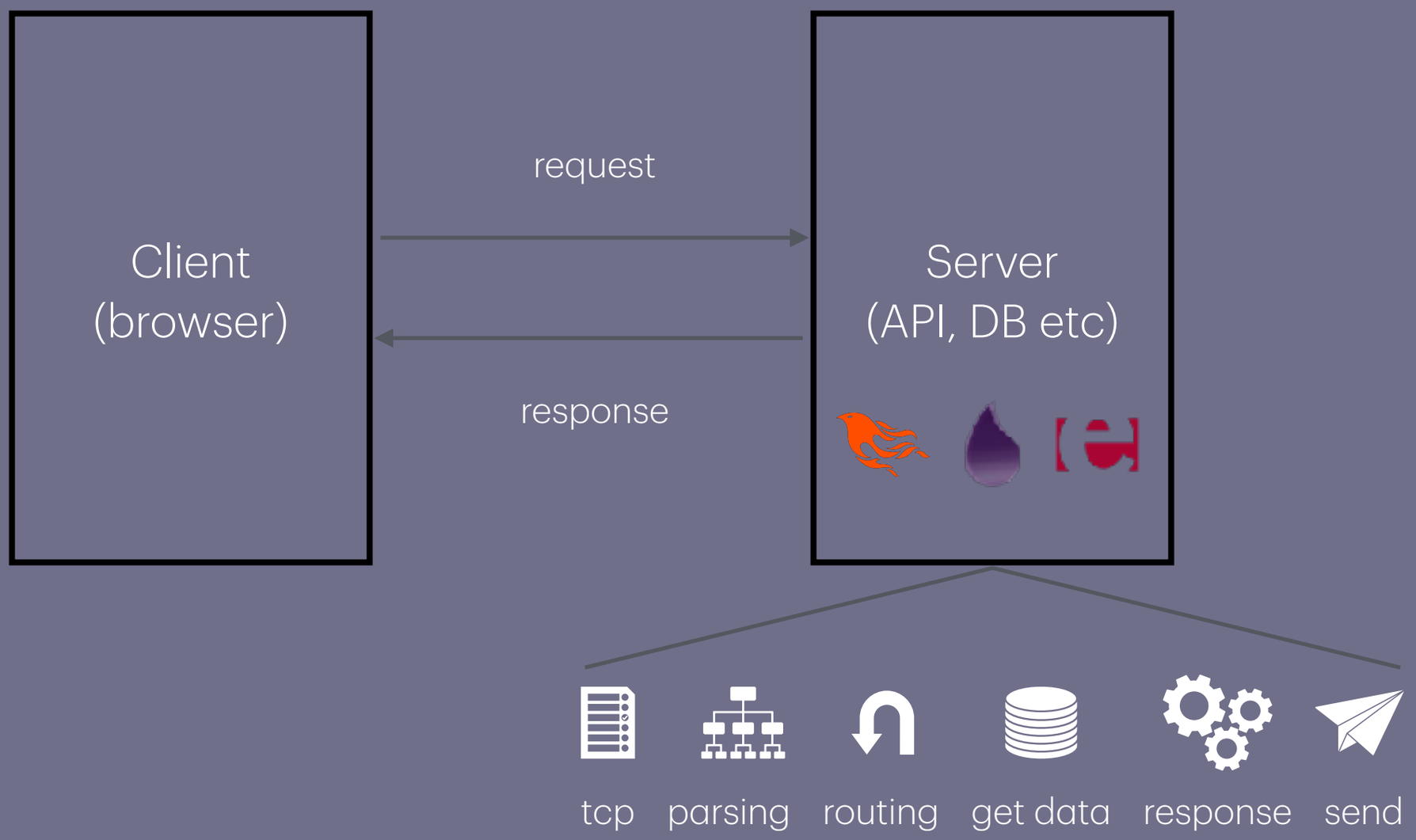
Phoenix Unplugged; uncovering the magic of plugs

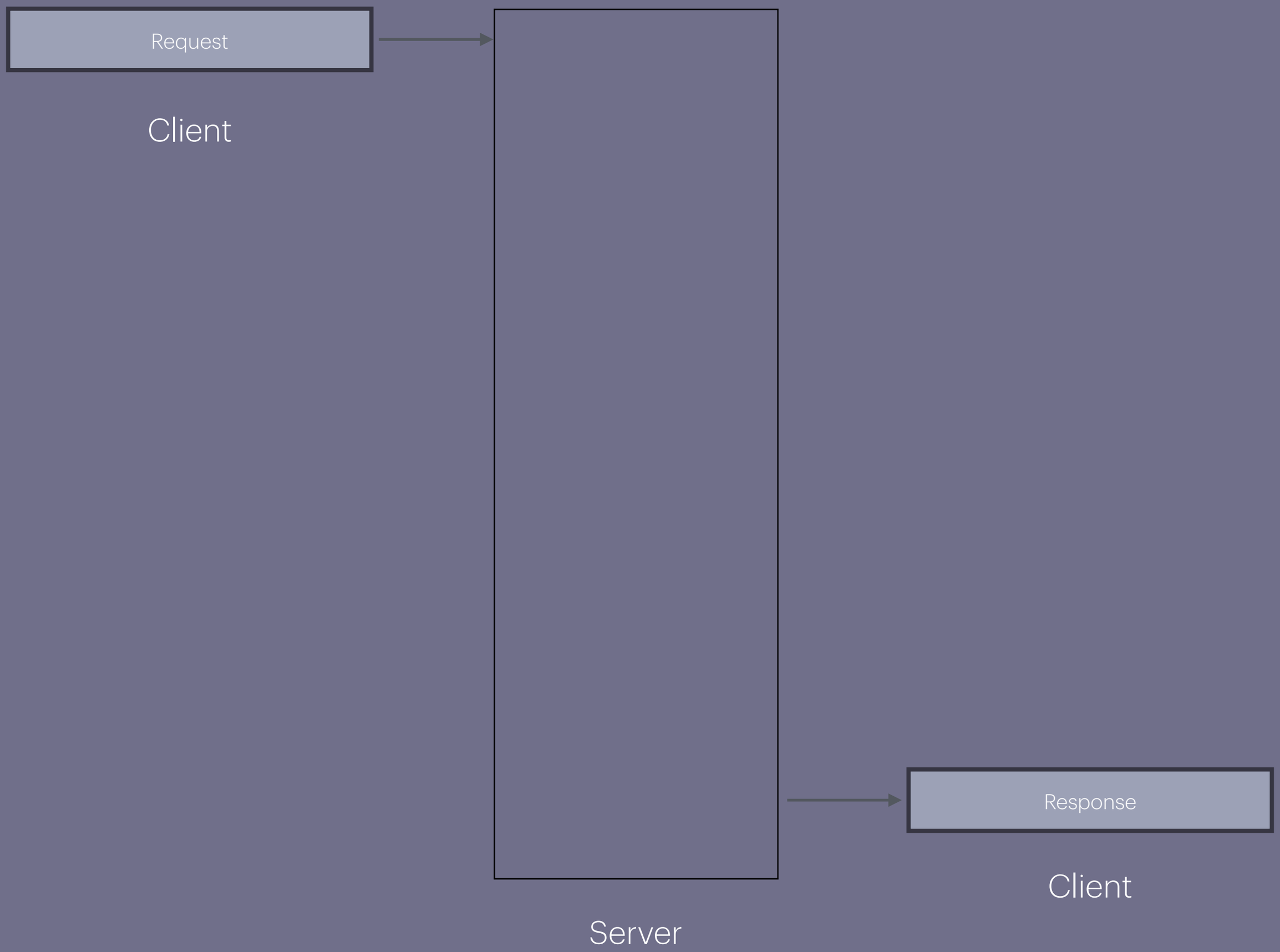


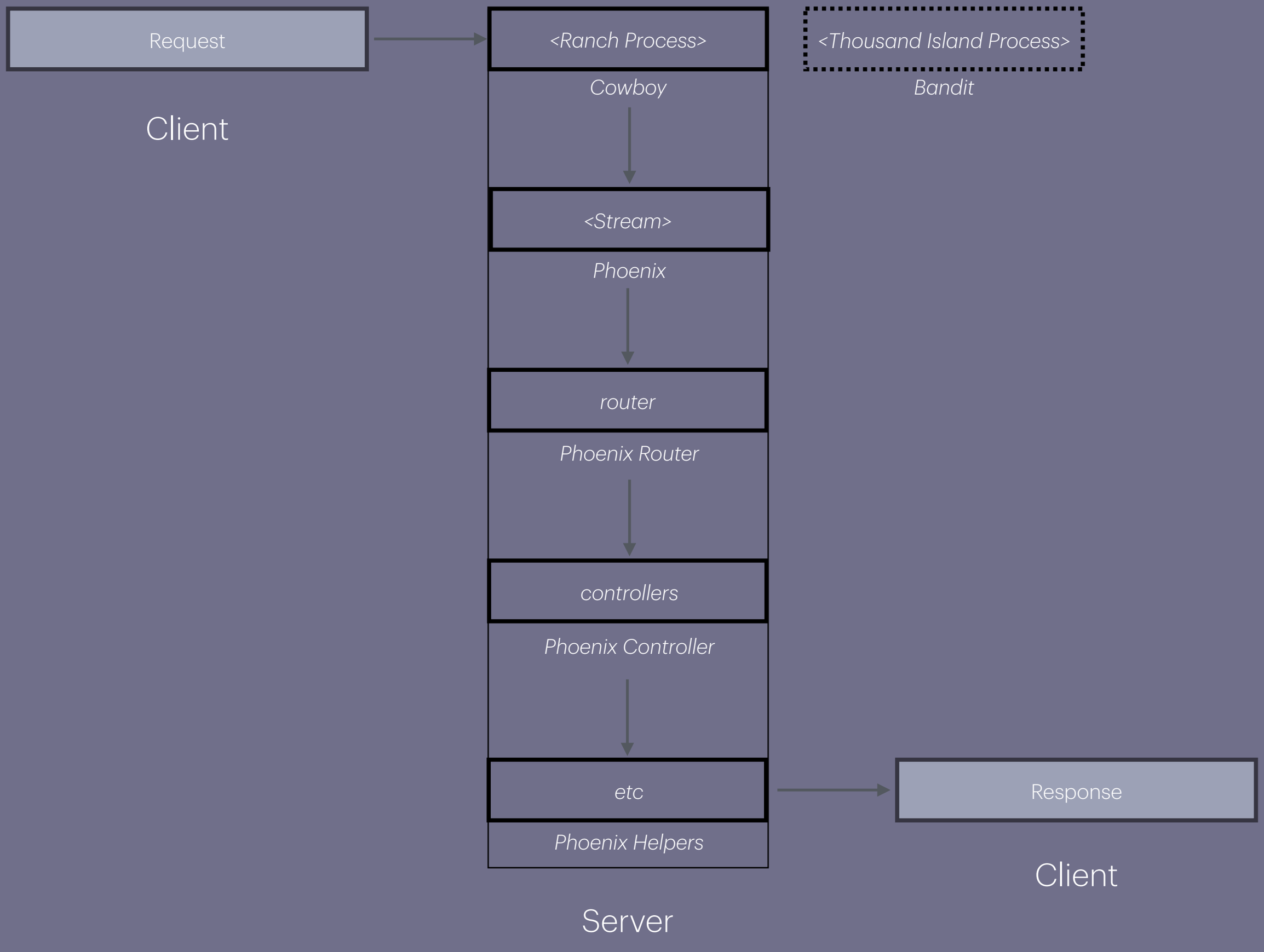
Request Response Cycle

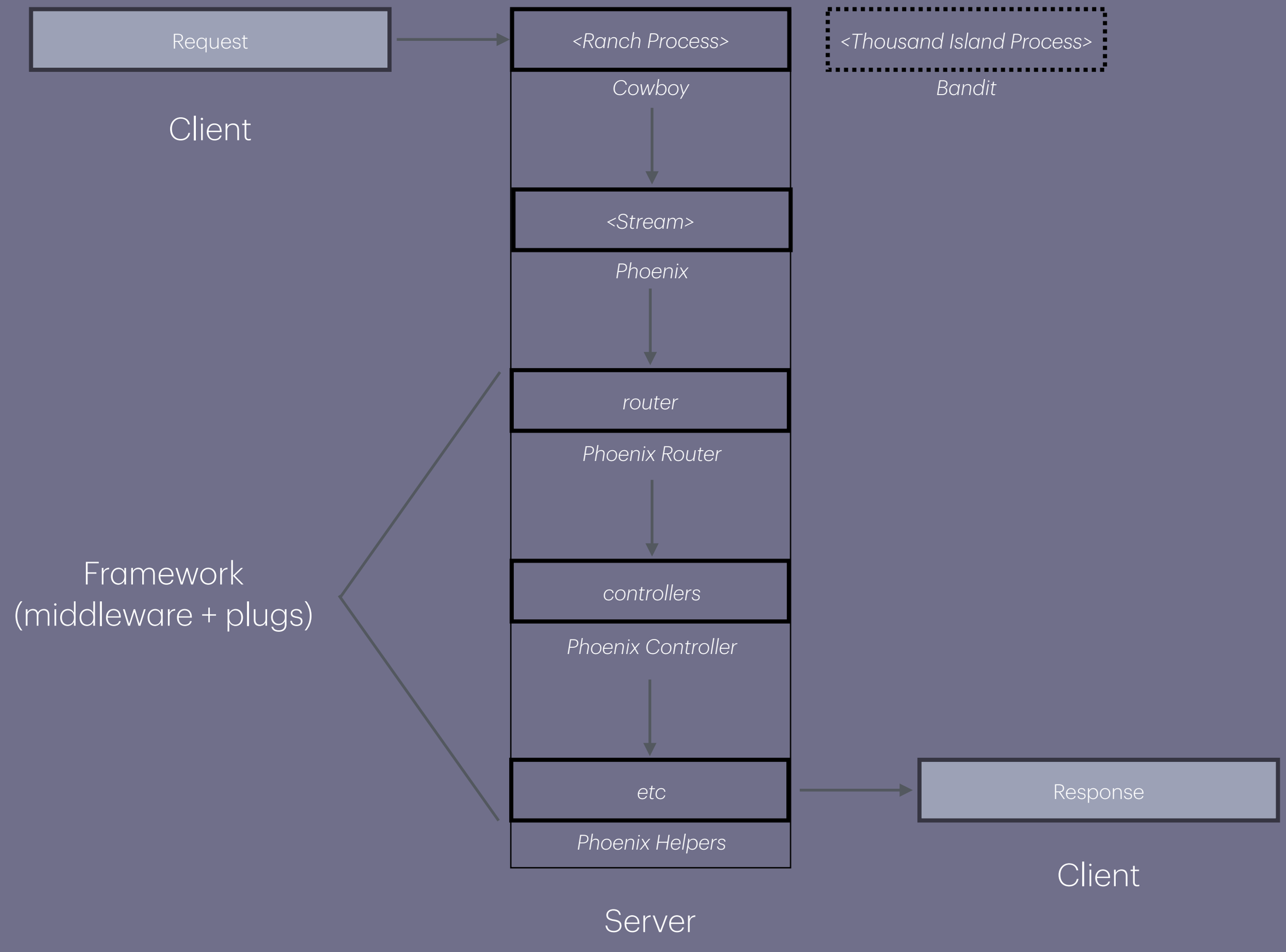


Request Response Cycle



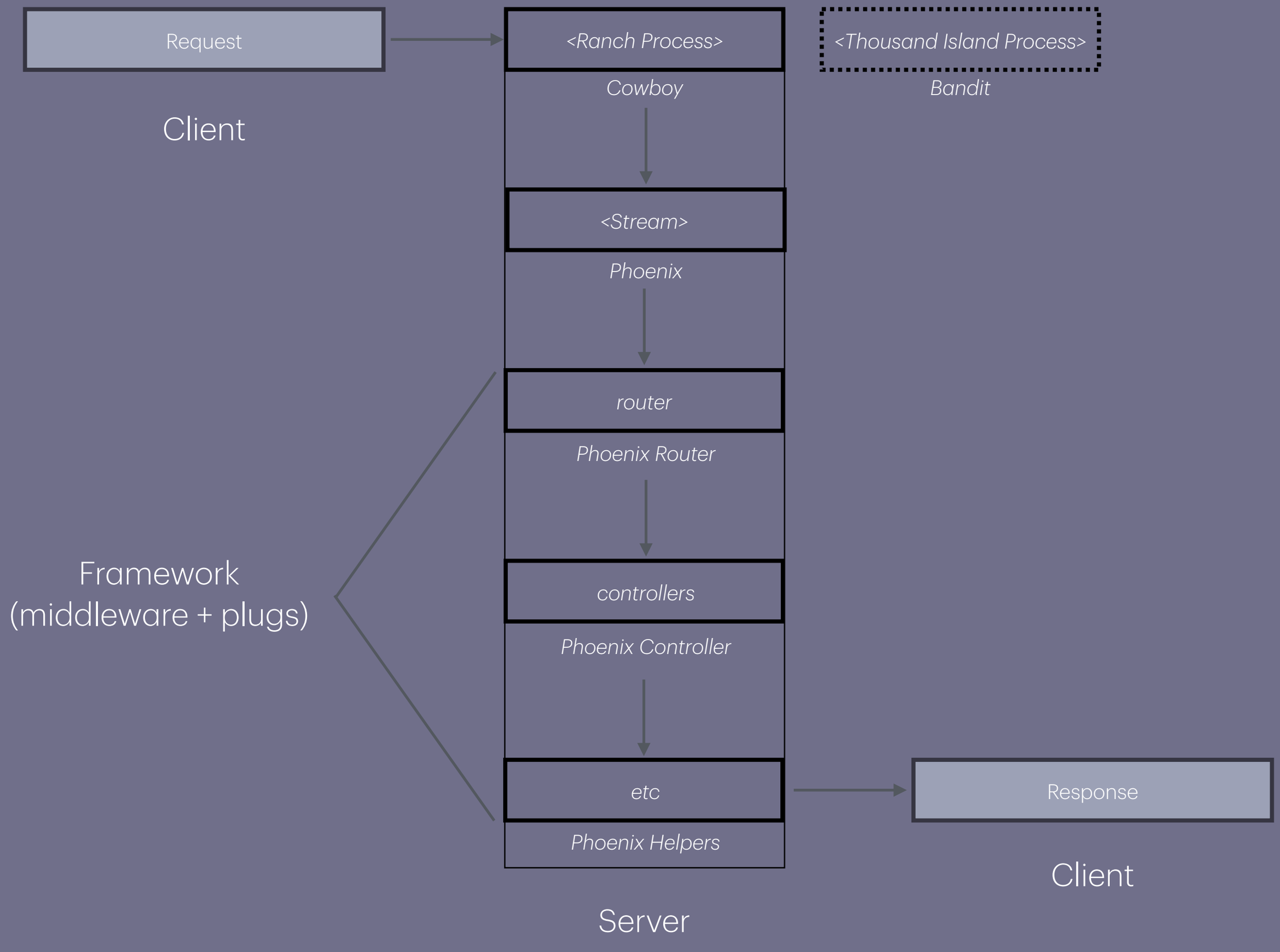

Request
Client
Server
Response
Client


Request
Client
<Ranch Process>
Cowboy
<Stream>
Phoenix
router
Phoenix Router
controllers
Phoenix Controller
etc
Phoenix Helpers
Server
<Thousand Island Process>
Bandit
Response
Client


Request
Client
<Ranch Process>
Cowboy
<Thousand Island Process>
Bandit
<Stream>
Phoenix
router
Phoenix Router
controllers
Phoenix Controller
etc
Phoenix Helpers
Framework
(middleware + plugs)
Response
Client
Server

Divya Sasidharan

@shortdiv

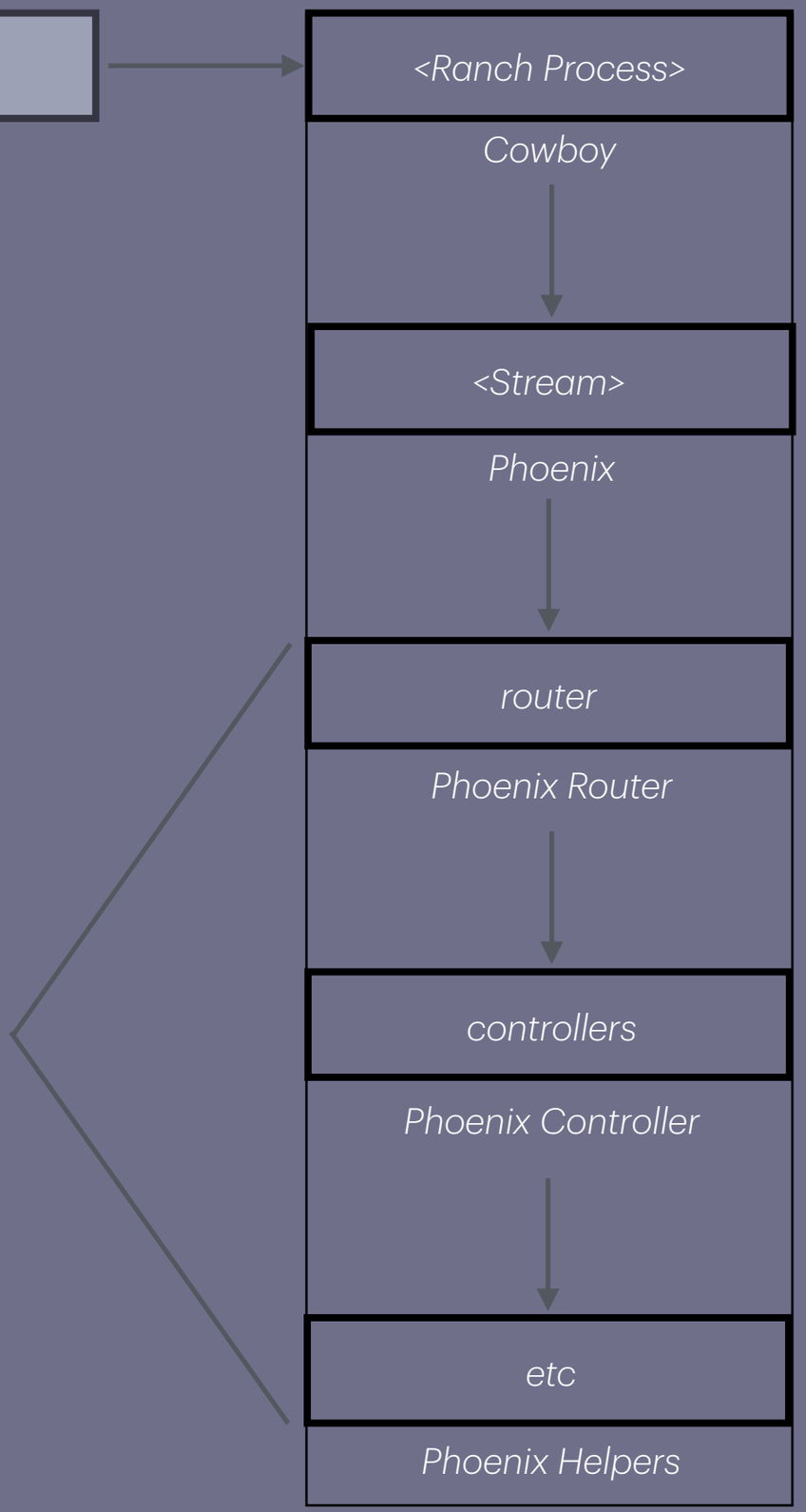

Request
Client
<Ranch Process>
Cowboy
<Thousand Island Process>
Bandit
<Stream>
Phoenix
router
Phoenix Router
controllers
Phoenix Controller
etc
Phoenix Helpers
Framework
(middleware + plugs)
Server
Response
Client

Request

Client

Framework
(middleware + plugs)

How does this work?



<Thousand Island Process>

Bandit

Server

Response

Client

What is a cowboy/bandit?

Cowboy/Bandit is a small, HTTP server for Erlang/OTP, used in the elixir ecosystem.

What is a ranch/thousand island?

Ranch/Thousand Island is a socket acceptor pool library for Erlang/OTP, commonly used in the Elixir ecosystem to manage TCP connections.

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  defmodule Handler do
    def init(req, _opts) do
      resp =
        :cowboy_req.reply(
          _status = 200,
          _body = "<!doctype html><h1>Hello, Cowboy!</h1>",
          _request = req
        )

      {:ok, resp, []}
    end
  end

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {:_, Handler, []}
         ]}
      ])

   :cowboy.start_clear(
      :hello_http,
      [port: 4001],
      %{env: %{dispatch: routes}}
    )
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  defmodule Handler do
    def init(req, _opts) do
      resp =
        :cowboy_req.reply(
          _status = 200,
          _body = "<!doctype html><h1>Hello, Cowboy!</h1>",
          _request = req
        )

      {:ok, resp, []}
    end
  end

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {:_, Handler, []}
         ]}
      ])

    :cowboy.start_clear(
      :hello_http,
      [port: 4001],
      %{env: %{dispatch: routes}}
    )
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {:_, HowdyElixirConf.Web.Handler, []}
         ]}
      ])

    :cowboy.start_clear(
      :hello_http,
      [port: 4001],
      %{env: %{dispatch: routes}}
    )

    children = []

    opts = [strategy: :one_for_one, name: Campsite.Supervisor]
    Supervisor.start_link(children, opts)

  end
end
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {"/", HowdyElixirConf.Web.RootHandler, []}
         ]}
      ])

    :cowboy.start_clear(
      :hello_http,
      [port: 4001],
      %{env: %{dispatch: routes}}
    )

    children = []

    opts = [strategy: :one_for_one, name: Campsite.Supervisor]
    Supervisor.start_link(children, opts)

  end
end
```

web/root_handler.ex

```
defmodule HowdyElixirConf.Web.RootHandler do
  def init(req, _opts) do
    resp =
      :cowboy_req.reply(
        200,
        %{"content-type" => “text/html"},
        "<!doctype html><h1>Howdy Elixir Conf!</h1>”,
        req
      )

    {:ok, resp, []}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end
```

web/root_handler.ex

```
defmodule HowdyElixirConf.Web.RootHandler do
  def init(req, _opts) do
    # do some auth
    {AuthHeader, Req1} = cowboy_req:header(<<"authorization">>, Req0)

    resp =
      if is_authorized(AuthHeader) do
        :cowboy_req.reply(
          200,
          %{"content-type" => “text/html"},
          "<!doctype html><h1>Howdy Elixir Conf!</h1>”,
          req
        )
      else
        :cowboy_req.reply(
          404,
          %{"content-type" => “text/html"},
          "<!doctype html><h1>Route not found</h1>”,
          req
        )
      end

    {:ok, resp, []}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end

```

web/root_handler.ex

```elixir
defmodule HowdyElixirConf.Web.RootHandler do
  def init(req, _opts) do
    # do some auth
    {AuthHeader, Req1} = cowboy_req:header(<<"authorization">>, Req0)

    resp =
      if is_authorized(AuthHeader) do
        :cowboy_req.reply(
          200,
          %{"content-type" => “text/html"},
          "<!doctype html><h1>Howdy Elixir Conf!</h1>”,
          req
        )
      else
        :cowboy_req.reply(
          404,
          %{"content-type" => “text/html"},
          "<!doctype html><h1>Route not found</h1>”,
          req
        )
      end

    {:ok, resp, []}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end

```

web/root_handler.ex

```elixir
defmodule HowdyElixirConf.Web.RootHandler do
  def init(req, _opts) do
    Req2 =
      :cowboy_req.stream_reply(
        200, %{"content-type" => "text/html"},
        req
      )
    Req3 =
      :cowboy_req.stream_body(
        "<!doctype html><html><head><title>Hi</title></head><body><h1>”,
        Req2
      )
    Req4 =
      :cowboy_req.stream_body(
        "Howdy Elixir Conf!</h1><p>Streaming with Cowboy!</p>”,
        Req3
      )
    Req5 =
      :cowboy_req.stream_body(
        “</body></html>”,
        Req4
      )

    {:ok, Req5, []}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end
```

web/root_handler.ex

```
defmodule HowdyElixirConf.Web.RootHandler do
  def init(req, _opts) do
    Req2 =
      :cowboy_req.stream_reply(
        200, %{"content-type" => "text/html"},
        req
      )
    Req3 =
      :cowboy_req.stream_body(
        "<!doctype html><html><head><title>Hi</title></head><body><h1>”,
        Req2
      )
    Req4 =
      :cowboy_req.stream_body(
        "Howdy Elixir Conf!</h1><p>Streaming with Cowboy!</p>”,
        Req3
      )
    Req5 =
      :cowboy_req.stream_body(
        “</body></html>”,
        Req4
      )

    {:ok, Req5, []}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end
```

web/root_handler.ex

```elixir
defmodule HowdyElixirConf.Web.RootHandler do
  def init( req, _opts) do
     Req2 =
       :cowboy_req.stream_reply(
         200, %{"content-type" => "text/html"},
         req
       )
     Req3 =
       :cowboy_req.stream_body(
         "<!doctype html><html><head><title>Hi</title></head><body><h1>”,
         Req2
       )
     Req4 =
       :cowboy_req.stream_body(
         "Howdy Elixir Conf!</h1><p>Streaming with Cowboy!</p>”,
         Req3
       )
     Req5 =
       :cowboy_req.stream_body(
         “</body></html>”,
         Req4
       )

     {:ok, Req5, []}
   end

   def terminate(_reason, _req, _state) do
     :ok
   end
end
```

web/root_handler.ex

```elixir
defmodule HowdyElixirConf.Web.RootHandler do
  import Plug.Conn

  def init(conn,_opts) do
     conn = send_chunked(conn, 200)


     {:ok, conn} =
       chunk(
         conn,
         "<!doctype html><html><head><title>Hi</title></head><body><h1>"
       )
     {:ok, conn} =
       chunk(
         conn,
         "Howdy Elixir Conf!</h1><p>Streaming with Cowboy!</p>"
       )
    {:ok, conn} =
       chunk(
        conn,
        "</body></html>"
       )

     conn
   end

  def terminate(_reason, _req, _state) do
    :ok
  end
end
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {"/", HowdyElixirConf.Web.RootHandler, []}
         ]}
      ])

    :cowboy.start_clear(
      :hello_http,
      [port: 4001],
      %{env: %{dispatch: routes}}
    )
    children = []

    opts = [strategy: :one_for_one, name: Campsite.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {"/", HowdyElixirConf.Web.RootHandler, []}
           {"/another", HowdyElixirConf.Web.AnotherHandler, []}
           {"/one", HowdyElixirConf.Web.OneHandler, []}
           …
         ]}
      ])

    :cowboy.start_clear(
     :hello_http,
     [port: 4001],
     %{env: %{dispatch: routes}}
    )
     children = []

     opts = [strategy: :one_for_one, name: Campsite.Supervisor]
     Supervisor.start_link(children, opts)
  end
end
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {"/", HowdyElixirConf.Web.RootHandler, []}
           {"/another", HowdyElixirConf.Web.AnotherHandler, []}
           {"/one", HowdyElixirConf.Web.OneHandler, []}
           …
         ]}
      ])

    :cowboy.start_clear(
     :hello_http,
     [port: 4001],
     %{env: %{dispatch: routes}}
    )
     children = []

     opts = [strategy: :one_for_one, name: Campsite.Supervisor]
     Supervisor.start_link(children, opts)
  end
end
```

application.ex

```elixir
defmodule HowdyElixirConf.Application do
  use Application

  def start(_type, _args) do
    routes =
      :cowboy_router.compile([
        {:_,
         [
           {:_, HowdyElixirConf.Web.PageHandler,
                HowdyElixirConf.Web.Router
           }
           …
         ]}
      ])

    :cowboy.start_clear(
     :hello_http,
     [port: 4001],
     %{env: %{dispatch: routes}}
    )
     children = []

     opts = [strategy: :one_for_one, name: Campsite.Supervisor]
     Supervisor.start_link(children, opts)
  end
end
```

web/router.ex

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  alias HowdyElixirConf.Web.PageController

  get "/", PageController, :home
  get "/two", PageController, :two

  get not_matched, PageController, :not_matched, %{path: not_matched}
end
```

web/page_handler.ex

```elixir
defmodule HowdyElixirConf.Web.PageHandler do
  def init(req, router) do
    path = :cowboy_req.path(req)
    conn = %Plug.Conn{req_path: path}
    conn = router.call(conn)

    resp =
      :cowboy_req.reply(conn.status, conn.resp_body, req)

    {:ok, resp, router}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end
```

web/router.ex

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  alias HowdyElixirConf.Web.PageController

  get "/", PageController, :home
  get "/two", PageController, :two

  get not_matched, PageController, :not_matched, %{path: not_matched}
end
```

web/page_handler.ex

```elixir
defmodule HowdyElixirConf.Web.PageHandler do
  def init(req, router) do
    path = :cowboy_req.path(req)
    conn = %Plug.Conn{req_path: path}
    conn = router.call(conn)

    resp =
      :cowboy_req.reply(conn.status, conn.resp_body, req)

    {:ok, resp, router}
  end

  def terminate(_reason, _req, _state) do
    :ok
  end
end
```

What is a plug?

Plugs are modules in Elixir used to build composable web applications. Each plug can intercept, modify, or respond to an HTTP request or response.

Function Plug

Any function that receives a conn and a set of options and returns a conn. It has this type signature:

```
(Plug.Conn.t, Plug.opts) :: Plug.Conn.t
```

Module Plug

An extension of the function plug. But must export 2 functions:

```
(Plug.Conn.t(), any()) :: Plug.Conn.t()
def call/2

(opts :: any()) :: any()
def init/1
```

Function Plug

```
(Plug.Conn.t, Plug.opts) :: Plug.Conn.t
```

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  plug :set_custom_header

  get "/", HowdyElixirConf.Web.PageController, :home

  # Define the function plug
  defp set_custom_header(conn, _opts) do
    conn
    |> Plug.Conn.put_resp_header("x-custom-header", "HowdyHeader")
  end
end
```

Function Plug

```
(Plug.Conn.t, Plug.opts) :: Plug.Conn.t
```

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  plug :set_custom_header

  get "/", HowdyElixirConf.Web.PageController, :home

  # Define the function plug
  defp set_custom_header(conn, _opts) do
    conn
    |> Plug.Conn.put_resp_header("x-custom-header", "HowdyHeader")
  end
end
```

Module Plug

```elixir
(Plug.Conn.t(), any()) :: Plug.Conn.t()
def call/2

(opts :: any()) :: any()
def init/1
```

```elixir
defmodule Plug.CustomHeader do
  import Plug.Conn

  @spec init(opts :: any()) :: any()
  def init(opts), do: opts end

  @spec call(conn :: Plug.Conn.t(), opts :: any()) :: Plug.Conn.t()
  def call(conn, _opts) do
    conn
    |> put_resp_header("x-custom-header", "MyAppHeader")
  end
end
```

Module Plug

```
(Plug.Conn.t(), any()) :: Plug.Conn.t()
def call/2

(opts :: any()) :: any()
def init/1
```

```
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  plug Plug.CustomHeader

  get "/", HowdyElixirConf.Web.PageController, :home
end
```

Module Plug

```
(Plug.Conn.t(), any()) :: Plug.Conn.t()
def call/2

(opts :: any()) :: any()
def init/1
```

```
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  plug Plug.CustomHeader

  get "/", HowdyElixirConf.Web.PageController, :home
end
```

The mighty `Plug.Conn`

```
defstruct adapter: {Plug.MissingAdapter, nil},
          assigns: %{},
          body_params: %Unfetched{aspect: :body_params},
          cookies: %Unfetched{aspect: :cookies},
          params: %Unfetched{aspect: :params},
          path_info: [],
          query_params: %Unfetched{aspect: :query_params},
          query_string: "",
          req_cookies: %Unfetched{aspect: :cookies},
          req_headers: [],
          request_path: "",
          resp_body: nil,
          resp_cookies: %{},
          scheme: :http,
          state: :unset,
          status: nil
          …
```

`Plug.Router`

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Plug.Router

  plug :match
  plug :dispatch

  get "/hello" do
    send_resp(conn, 200, "world")
  end

  match _ do
    send_resp(conn, 404, "oops")
  end
end
```

`Plug.Router`

spoiler alert A router is a plug

`Plug.Router`

```
defmodule Plug.Router do
  defmacro __using__(opts) do
    quote location: :keep do
      import Plug.Router
      use Plug.Builder, unquote(opts)

      def match(conn, _opts) do
        …
      end
      …
    end
    …
  end
end
```

`Plug.Router`

```
defmodule Plug.Router do
  defmacro __using__(opts) do
    quote location: :keep do
      import Plug.Router
      use Plug.Builder, unquote(opts)

      def match(conn, _opts) do
        …
      end
      …
    end
    …
  end
end
```

`Plug.Router`

```
defmodule Plug.Router do
  defmacro __using__(opts) do
    quote location: :keep do
      import Plug.Router
      use Plug.Builder, unquote(opts)

      def match(conn, _opts) do
        …
      end
      …
    end
    …
  end
end
```

```
defmacro __using__(opts) do
  quote do
    @behaviour Plug

    def init(opts), do: opts end

    def call(conn, opts) do
      …
    end
    …
  end
end
```

Let's make a router

beware the metaprogramming

Let's make a router

beware the metaprogramming

But first:

A metaprogramming primer

```elixir
defmodule Greetable do
  defmacro __using__(opts) do
    greeting = Keyword.get(opts, :greeting, "Hello")

    quote do
      def greet(name) do
        IO.puts(unquote(greeting) <> ", " <> name <> "!")
      end

      def farewell(name) do
        IO.puts("Goodbye, " <> name <> "!")
      end
    end
  end
end

defmodule EnglishGreeter do
  use Greetable
end

defmodule SpanishGreeter do
  use Greetable, greeting: "Hola"
end

EnglishGreeter.greet("Alice")   # Outputs: Hello, Alice!
SpanishGreeter.greet("Bob")     # Outputs: Hola, Bob!
EnglishGreeter.farewell("Charlie")  # Outputs: Goodbye, Charlie!
```

*allows **`Greetable`** to inject functions into other modules.*

define the code that will be injected

*implements the **`greet`** and **`farewell`** func*

passed to the macro as opts

Back to plugs

let's make this router go brrrr

The Rules of a Plug

1. Must implement `init/1` and `call/2`

2. Must use Plug.Conn

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do

      def init(opts), do: opts

      def call(conn, _opts) do

     end
    end
  end
end
```

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do

     end
    end
  end
end
```

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do

     end
    end
  end
end
```

modules that use this module, import this

Implements plug behaviour

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do

     end
    end
  end
end
```

modules that use this module, import this

Implements plug behaviour

web/router.ex

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Spaghetti.Router
  # The `use` macro above expands to include:
  # import Spaghetti.Router
  # def call(conn) do ... end

end
```

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do

      end
    end
  end
end
```

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do

      end
    end
  end

  defmacro get(path, controller, action) do
    quote do


    end
  end
end
```

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do
        content_for(conn.request_path, conn)
      end
    end
  end

  defmacro get(path, controller, action) do
    quote do
      defp content_for(unquote(path), conn) do
        apply(unquote(controller), :call, [conn, unquote(action)])
      end
    end
  end
end
```

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do
        content_for(conn.request_path, conn)
      end
    end
  end

  defmacro get(path, controller, action) do
    quote do
      defp content_for(unquote(path), conn) do
        apply(unquote(controller), :call, [conn, unquote(action)])
      end
    end
  end
end
```

dynamically invokes the call/2 function on the controller module, passing in the conn and action as arguments.

Kernel.apply/3

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do
        content_for(conn.request_path, conn)
      end
    end
  end

  defmacro get(path, controller, action) do
    quote do
      defp content_for(unquote(path), conn) do
        apply(unquote(controller), :call, [conn, unquote(action)])
      end
    end
  end
end
```

dynamically invokes the call/2 function on the controller module, passing in the conn and action as arguments.

(module, function name, args)

Kernel.apply/3

spaghetti/router.ex

```elixir
defmodule Spaghetti.Router do
  defmacro __using__(_opts) do
    quote do
      import Spaghetti.Router
      @behaviour Plug

      def init(opts), do: opts

      def call(conn, _opts) do
        content_for(conn.request_path, conn)
      end
    end
  end

  defmacro get(path, controller, action) do
    quote do
      defp content_for(unquote(path), conn) do
        apply(unquote(controller), :call, [conn, unquote(action)])
      end
    end
  end
end
```

spaghetti/router.ex

```elixir
  defmacro get(path, controller, action) do
    quote do
      defp content_for(unquote(path), conn) do
        apply(unquote(controller), :call, [conn, unquote(action)])
      end
    end
  end
end
```

web/router.ex

```elixir
defmodule HowdyElixirConf.Web.Router do
  use Spaghetti.Router

  alias HowdyElixirConf.Web.PageController

  get "/", PageController, :home

end
```

web/page_controller.ex

```elixir
defmodule HowdyElixirConf.Web.PageController do
  import Plugs.Conn

  def call(conn, action) do
    apply(__MODULE__, action, [conn])
  end

  def home(conn, _) do
    put_resp_body(conn, “<h1>Howdy Elixir!</h1>”)
  end

  …
```

Pony up! Let’s see some code.