Bandit on the loose!

Networking in Elixir demystified

@mattrudel - github.com/mtrudel - mat@geeky.net

github.com/mtrudel/talks

Introducing Thousand Island & Bandit

github.com/mtrudel/thousand_island - github.com/mtrudel/bandit

Phoenix
Plug
HTTP
Sockets
TCP/IP

Phoenix

Plug

HTTP

Sockets

TCP/IP

In the beginning there was IP (RFC 791)

- Provides each host on the network with an IP address
- 'Being connected to the internet' means[1] having an IP yourself and being able to reach any other host via their IP
- All IP does is deliver packets to a host
- IP 'encapsulates' higher level protocols such as TCP and UDP
    - Identified by a unique 'Protocol Number' (TCP is 6, UDP is 17)

[1] A million caveats to this (NAT, the DFZ, *cast, etc)

Then came TCP (RFC 793)

- Provides application-level addressability
- Addressed by an IP & a port number
- Connections come in the form of a pair of reliable streams
- Each direction is a one-way stream of binary data
- Each direction is independent, can be shutdown, used simultaneously
- Commonly accessed via the 'Berkeley Socket API', part of libc

Phoenix

Plug

HTTP

Sockets

TCP/IP

Phoenix

Plug

HTTP

Sockets

TCP/IP

Berkeley sockets in a nutshell

- Clients **`connect`** to an IP/port pair
- Servers **`bind`** to a named port, and **`listen`** for connections
- Servers then **`accept`** clients' **`connect`** requests in a loop
- Once accepted, a TCP session is created between the client and the server
- The two peers can now **`send`** and **`recv`** data with one another
- At any time, either peer can **`close`** a connection

How does Elixir access sockets?

- Via Erlang's **`:gen_tcp`** (and **`:socket`** since OTP 22)
- Provides nearly 1:1 mapping onto the standard Berkeley calls
- SSL is accessed via **`:ssl`** using ~identical functions
  - All the core socket abstractions are the same as TCP
  - `s/:gen_tcp/:ssl/`

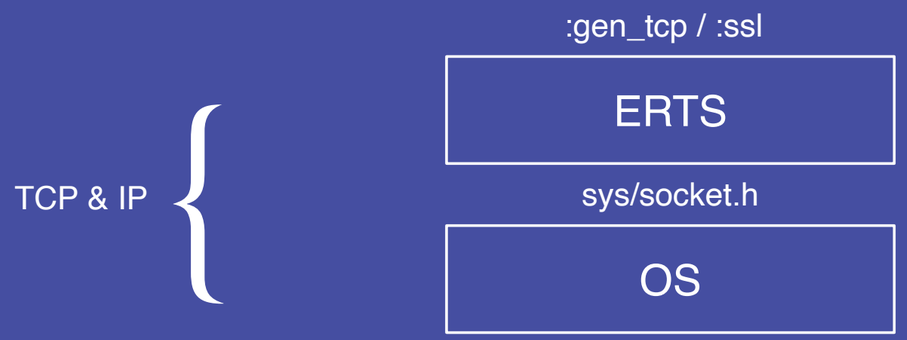
Simple :gen_tcp Server

```
{:ok, listen_socket} = :gen_tcp.listen(4000, [active: false])
accept_and_handle(listen_socket)

def accept_and_handle(listen_socket) do
  {:ok, socket} = :gen_tcp.accept(listen_socket)
  :gen_tcp.send(socket, "Hello, World")
  :gen_tcp.close(socket)
  accept_and_handle(listen_socket)
end
```

Simple :gen_tcp Client

```
{:ok, socket} = :gen_tcp.connect('localhost', 4000, [active: false])
{:ok, data} = :gen_tcp.recv(socket, 0)
IO.puts(data) #=> "Hello, World"
:gen_tcp.close(socket)
```

The Stack So Far



Socket Review

- This is all you **really** need to implement a server
- Working this low level is tedious
- Performant patterns are non-trivial
- Usually abstracted by a 'socket server'

What is a Socket Server?

- Provides a useful abstraction of raw sockets
- Sits underneath an e.g. HTTP server
- Listens for client connections over TCP/SSL
- Hands individual connections off to an upper protocol layer
- Handles transport concerns (SSL/TLS negotiation, connection draining, etc)
- Does this efficiently and scalably

Who is a Socket Server?

- Ranch is the current goto implementation of such a server on the BEAM
- Thousand Island[2] is a new pure-Elixir socket server, inspired by Ranch but with numerous improvements

[2] Get it? They're both salad dressings?!?

Thousand Island

A pure Elixir socket server, embodying OTP ideals

- Fast (performance basically equal to Ranch)
- Simple (1.7k LoC, about ½ the size of Ranch)
- Comprehensive documentation & examples
- Supports TCP, TLS & Unix Domain sockets
- Loads of config options
- Fully wired for telemetry, including socket-level tracing
- Extremely simple & powerful Handler behaviour

github.com/mtrudel/thousand_island

Let's build RFC867 (Daytime)

```elixir
defmodule Daytime do
  use ThousandIsland.Handler

  @impl ThousandIsland.Handler
  def handle_connection(socket, state) do
    time = DateTime.utc_now() |> to_string()
    ThousandIsland.Socket.send(socket, time)
    {:close, state}
  end
end

{:ok, pid} = ThousandIsland.start_link(handler_module: Daytime)
```

github.com/mtrudel/thousand_island

Let's build RFC862 (Echo)

```elixir
defmodule Echo do
  use ThousandIsland.Handler

  @impl ThousandIsland.Handler
  def handle_data(data, socket, state) do
    ThousandIsland.Socket.send(socket, data)
    {:continue, state}
  end
end

{:ok, pid} = ThousandIsland.start_link(handler_module: Echo)
```

github.com/mtrudel/thousand_island

Thousand Island Handler Process

- 1 process per connection
- **`ThousandIsland.Handler`** module is just a **`GenServer`**
- Your connection processes can do all the normal **`GenServer`** things
- Receives are async by default (via **`handle_data/3`**)
- Free to write 'traditional' blocking network code if you wish, anywhere
- **`start_link`** based escape hatch if you need it (you probably won't)

github.com/mtrudel/thousand_island

Thousand Island Process Model

- Handler processes are hosted within a process tree
- Rooted in the **`ThousandIsland.start_link/1`** call which starts the server
- Process tree is entirely self-contained
- Multi-level process tree designed to minimize contention
- Textbook example of how powerful OTP design patterns can be
- Check out the project README for more info

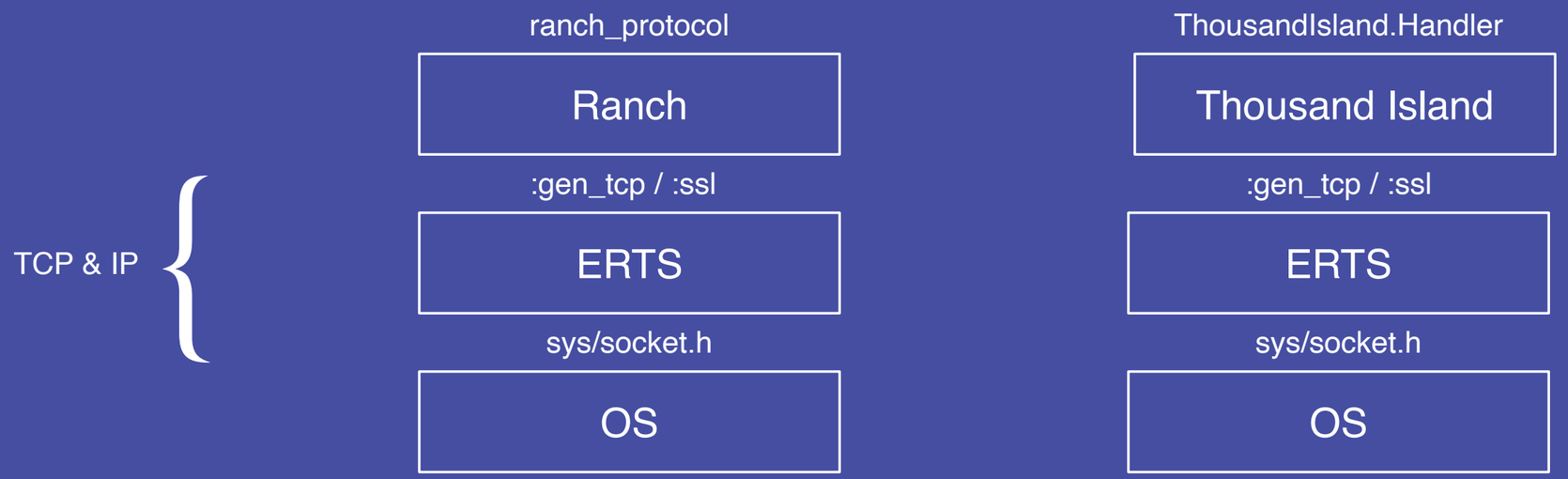
github.com/mtrudel/thousand_island

Thousand Island

github.com/mtrudel/thousand_island

- Stable and suitable for general use
- `0.5.x` series is likely the last before a `1.0`
- Big ideas for the future, but the foundation is solid

The Stack So Far



Phoenix

Plug

HTTP

Sockets

TCP/IP

What is HTTP?

- HTTP/1.x is just plain text over sockets. Defined primarily by RFC 2616

What is HTTP?

- HTTP/1.x is plain text over sockets. Defined primarily by RFC 2616

```
> GET /thing/12 HTTP/1.1
> host: www.example.com
> [... other headers ...]
> [empty line]
> [body content]

< HTTP/1.1 200 OK
< content-length: 123
< [... other headers ...]
< [empty line]
< [body content]
```

Let's build a simple HTTP server

```
defmodule HelloWorldHTTP do
  use ThousandIsland.Handler

  @impl ThousandIsland.Handler
  def handle_data(_data, socket, state) do
    ThousandIsland.Socket.send(socket, "HTTP/1.0 200 OK\r\n\r\nHello, World")
    {:close, state}
  end
end
```

What does an HTTP server do?

- A complete HTTP/1.x implementation isn't **that** much more complicated
- Parses & validates request and header lines (and possibly request body)
- Sends conformant responses back to the client

Who is an HTTP server?

- Cowboy is the current goto HTTP server on the BEAM
  - Complete HTTP/1.1, HTTP/2 & WebSocket server
- I've written Bandit[3], a new pure-Elixir HTTP server for Plug applications

[3] Get it? They're both tropes within a heavily romanticized & largely fictional American mythology?!?

Bandit

An HTTP Server for Plug Applications

- Written 100% in Elixir
- Plug-native
- Robust HTTP/1.1 and HTTP/2 conformance
- Written from the ground up for correctness, performance & clarity
- Incredible performance (more later!)

github.com/mtrudel/bandit

Making an HTTP server do useful things

- So, HTTP servers return content to clients, but **what** content?
- The content from our app, obviously
- But how to hand off to app logic?
- Enter Plug

Phoenix

Plug

HTTP

Sockets

TCP/IP

Plug

An abstraction of HTTP Request/Response Pairs

```elixir
defmodule HelloWorldPlug do
  def init(opts), do: opts

  def call(%Plug.Conn{} = conn, _opts) do
    Plug.Conn.send_resp(conn, 200, "Hello, World")
  end
end
```

github.com/elixir-plug/plug

Plug Comes With Batteries Included

- Plug comes with built-in Plugs to:
  - Create pipelines
  - Route requests
  - Manage sessions
  - Parse request bodies
  - **Lots** more
- This is enough for a LOT of applications

github.com/elixir-plug/plug

Aside: Bandit & Thousand Island's Origin Story

- Written to support HAP, a HomeKit Accessory Protocol library for Nerves
- Runs over HTTP, but with a twist (custom encryption on bare TCP)

github.com/mtrudel/hap

Aside: Bandit & Thousand Island's Origin Story

- Written to support HAP, a HomeKit Accessory Protocol library for Nerves
- Runs over HTTP, but with a twist (custom encryption on bare TCP)
- HTTP layer is simple, runs entirely on **`Plug.Router`**:

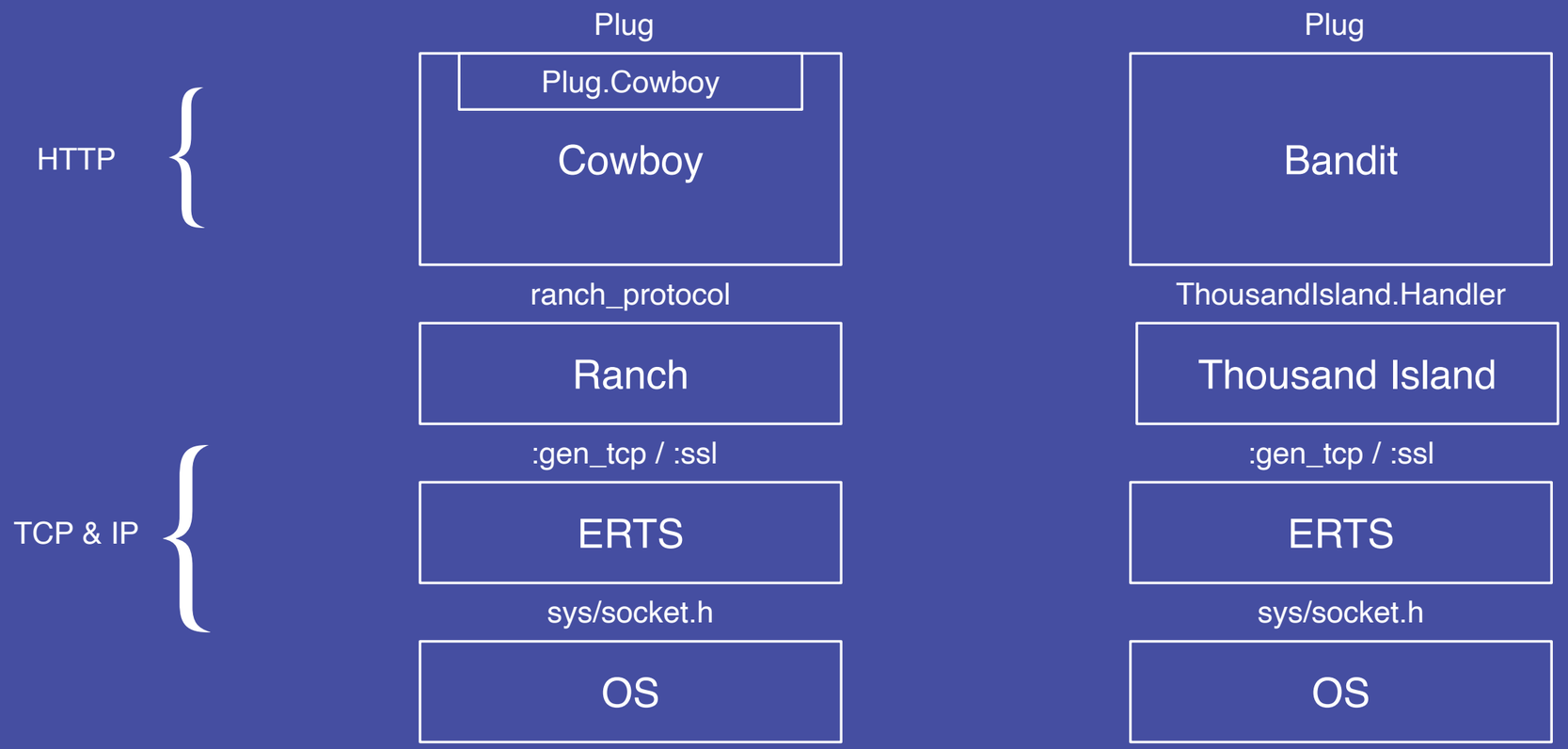
```
defmodule HAP.EncryptedHTTPServer do
  use Plug.Router

  post "/pairings" do ....
  get "/accessories" do ....
end
```

github.com/mtrudel/hap

Plug Servers

- Plug is implemented in Cowboy via the **`Plug.Cowboy`** adapter
- Bandit is Plug-Native

The Stack So Far



Phoenix

Plug

HTTP

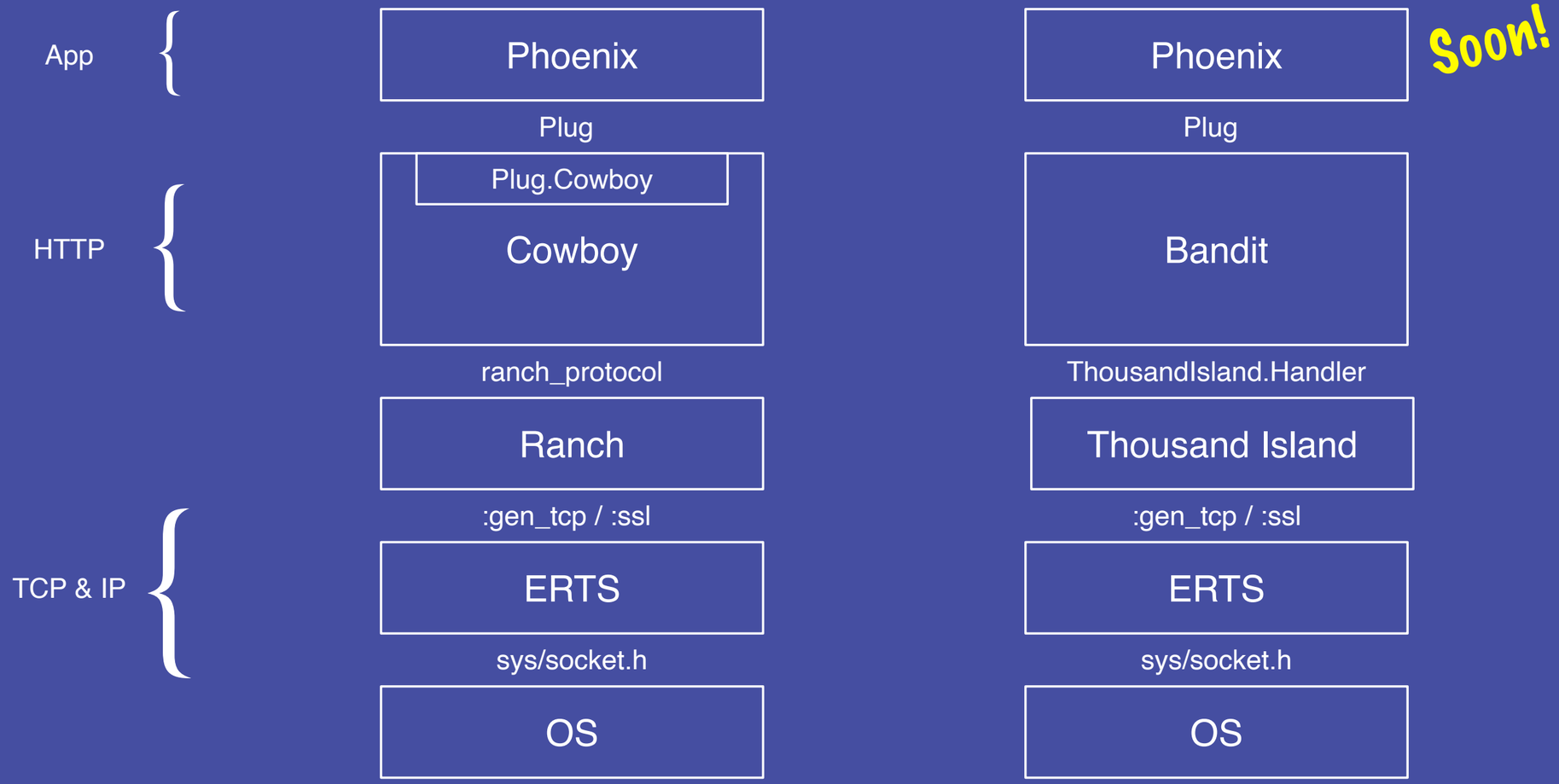
Sockets

TCP/IP

Enter Phoenix

- The HTTP part of Phoenix is 'just' a Plug
- Phoenix's WebSocket support is a different thing (more later)
- Not much more to say about things here
- This is as high in the stack as we're going today

The Full Stack (Finally)



Bandit

An HTTP Server for Plug Applications

- Built on top of Thousand Island
- Full support for HTTP/1.x & HTTP/2
  - Scores 100% on h2spec in **`--strict`** mode (runs in CI)
- Clear, Approachable, Idiomatic Elixir

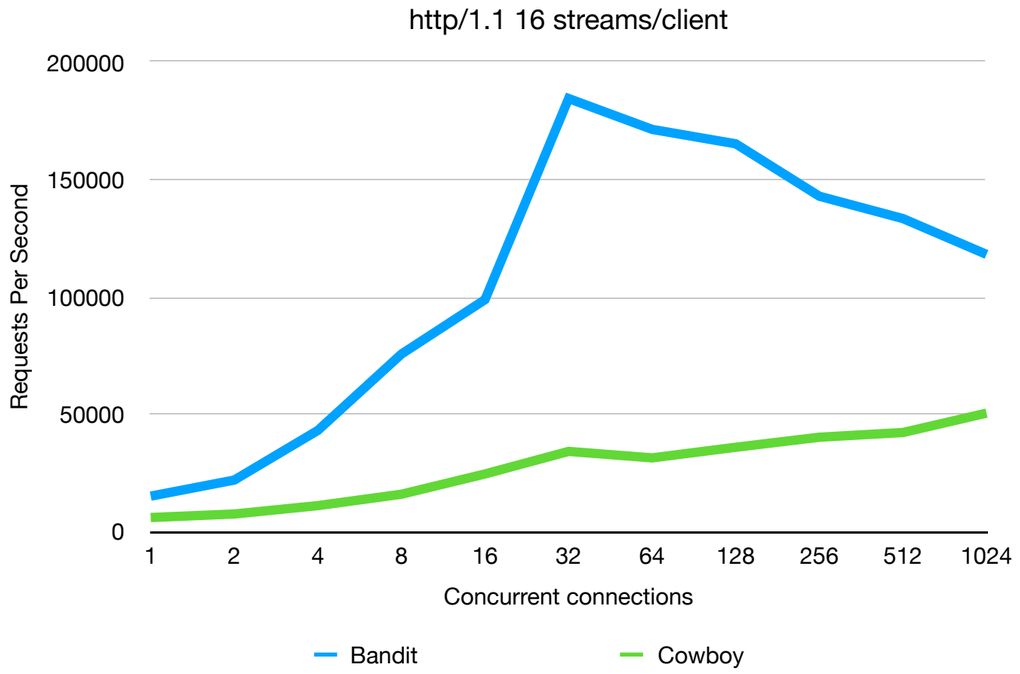
github.com/mtrudel/bandit

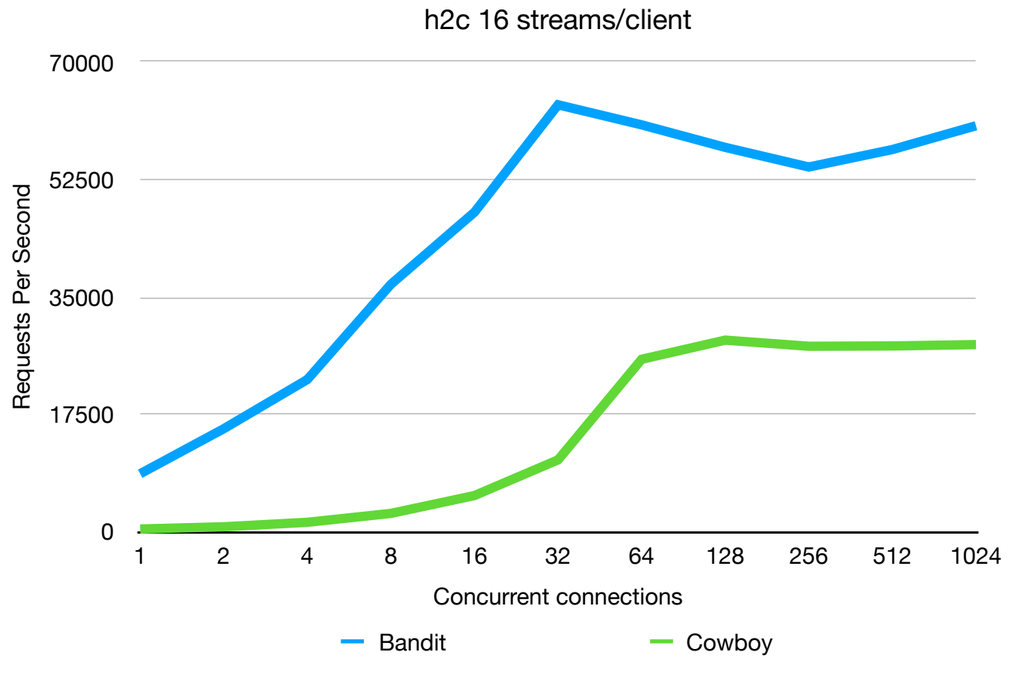
Bandit is a Plug-First Server

- No impedance mismatches
- Less (and clearer) code (<3k LoC today, about ⅓ of Cowboy)
- **Incredibly** fast and memory efficient

github.com/mtrudel/bandit

Up to 5x Faster Than Cowboy





github.com/mtrudel/network_benchmark

HTTP/1.1 In Bandit

- 1 process per connection
- 1:1 mapping between Handler **`handle_data/3`** calls and Plug **`call/2`** calls
- Straightforward, 'linear' code

github.com/mtrudel/bandit

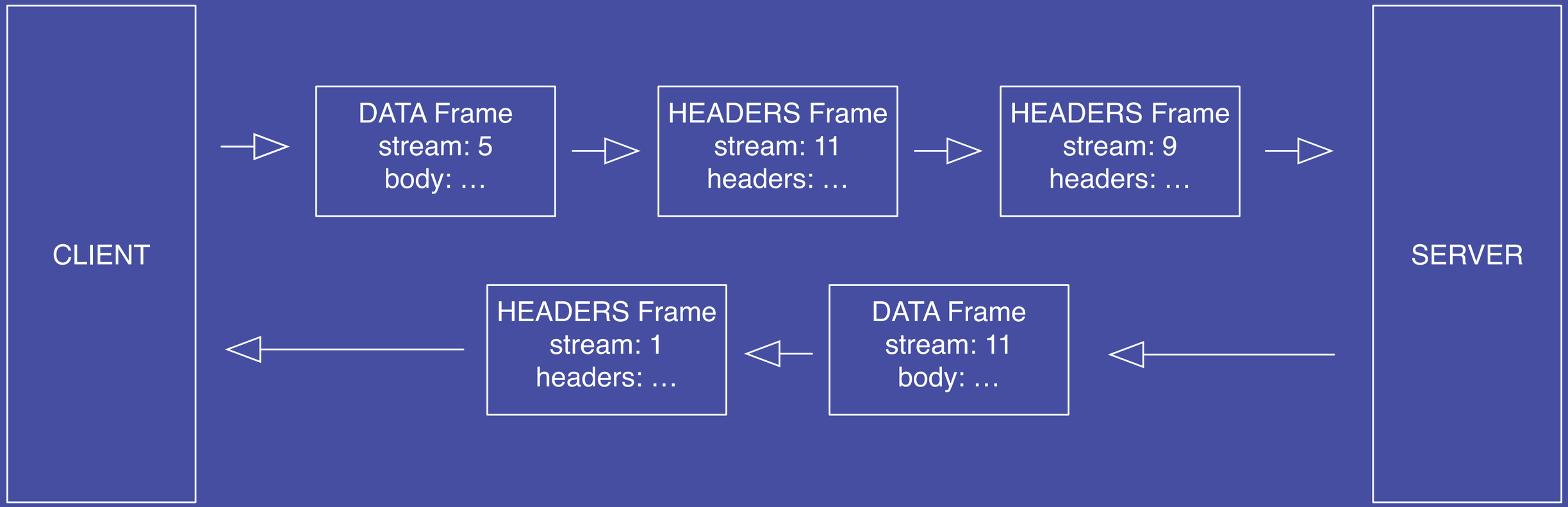
HTTP/2 In Bandit

- Quite a bit harder

github.com/mtrudel/bandit

HTTP/2 In A Nutshell

Framed binary protocol



github.com/mtrudel/bandit

HTTP/2 In Bandit

- 1 process per connection + 1 process per stream
- Connection process implements **`ThousandIsland.Handler`**
- Stream processes are the ones that make Plug calls
- Full support for flow control, push promises, all the goodies
- `iodata` everywhere for speed and memory awesomeness

github.com/mtrudel/bandit

Where Is Bandit Today?

- HTTP/2 implementation complete & exhaustively tested (0.3.x)
- HTTP/1.1 implementation undergoing refactor & test improvements (0.4.x)
- Suitable for non-production Plug apps (not Phoenix, yet)
- Drop-in replacement for Cowboy where appropriate

github.com/mtrudel/bandit

Future Plans for Bandit

- Phoenix Support
  - Requires WebSocket support (coming in 0.5.x)
  - Note that WebSocket support is outside the scope of Plug
  - Requires Phoenix integration (coming in 0.7.x)
- Continue to improve perf and Cowboy migration story
- 'PR Campaign' to drive adoption
- HTTP/3 Support (distant future)

github.com/mtrudel/bandit

The Goal:

The Goal:

Become the de-facto networking stack for Elixir & the BEAM (along with Thousand Island)

Working With Our Friends

- Work within Phoenix needed to add Bandit support
  - Possibly generalize its WebSocket interface
- Code sharing with the Mint HTTP client (Thanks Andrea & Eric for HPAX)
- Improve iolist primitives in ERTS
- Other minor upstream improvements (eg. URI canonicalization & trailer support in Plug)
- To be clear: Cowboy & Ranch are awesome projects

github.com/mtrudel/bandit

Contribute!

- Fun, foundational work
- Grassroots project
- Contributions are **extremely** welcome
  - Core HTTP/1.1 & WebSocket work
  - Property-based testing suite
  - Security review & mitigation
  - Profiling & perf improvements / automated testing
  - **Lots** more!

github.com/mtrudel/bandit

github.com...
/mtrudel/thousand_island
/mtrudel/bandit
/mtrudel/talks
@mattrudel (Twitter)
@mtrudel (Elixir Slack)
mat@geeky.net