Phoenix
Framework

@josevalim / phoenixframework.org

Glossary

- Phoenix (web framework)
- Elixir (programming language)
- Erlang VM

2 million connections on a single node

http://blog.whatsapp.com/index.php/2012/01/1-million-is-so-2011/

Intel Xeon CPU X5675 @ 3.07GHz
24 CPU - 96GB
Using 40% of CPU and Memory

ERLANG

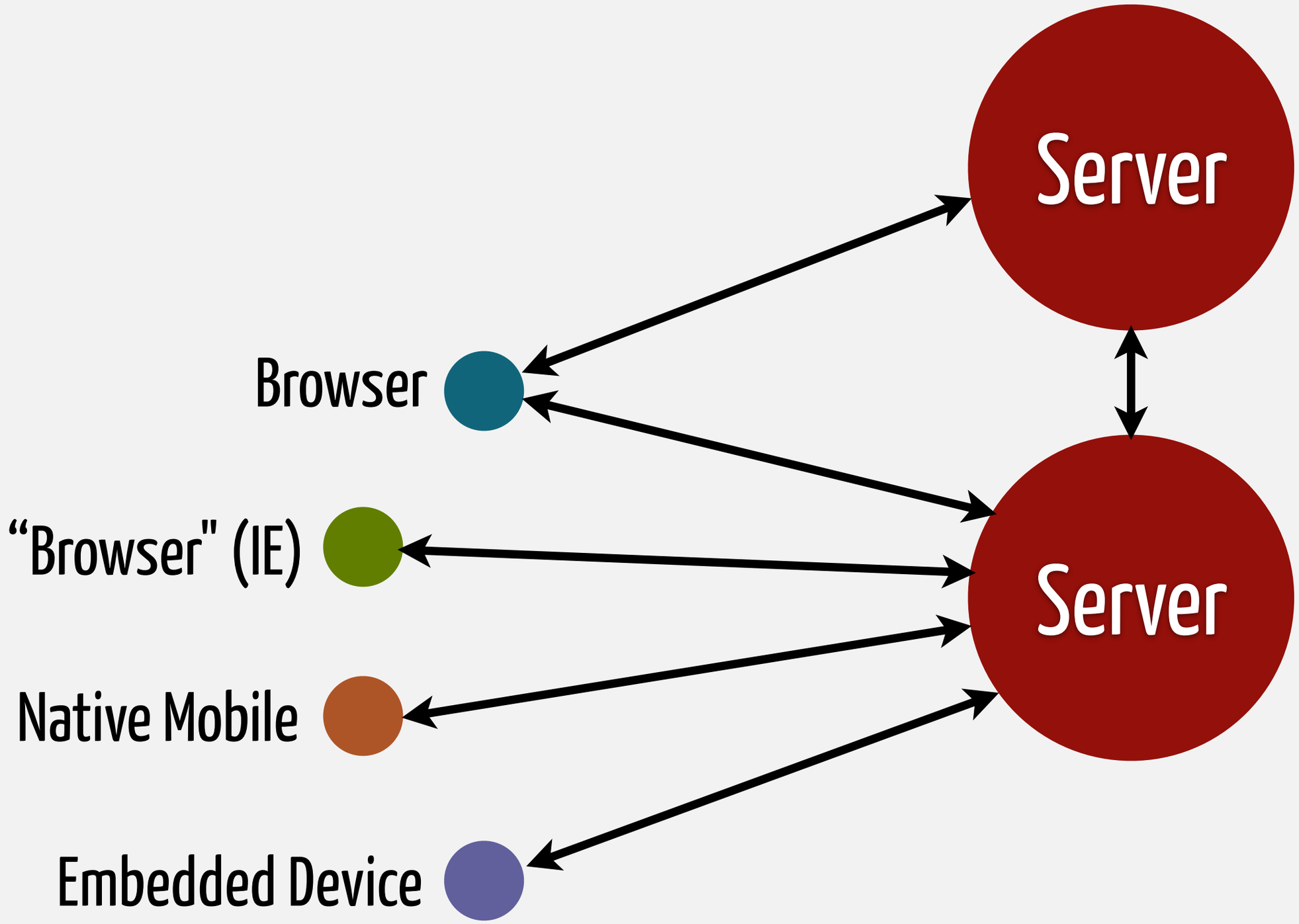
Phoenix Channels

```
var socket = new Phoenix.Socket("/ws");
socket.connect();

var channel = socket.channel(“chat:lobby");

channel.on("user_joined", function(message){
  // ...
});

channel.on("new_message", function(msg){
  // ...
});

$input.on("enter", function(e){
  channel.push("new_message", {
    content: $input.val(),
    username: App.username
  });
});

channel.join();
```

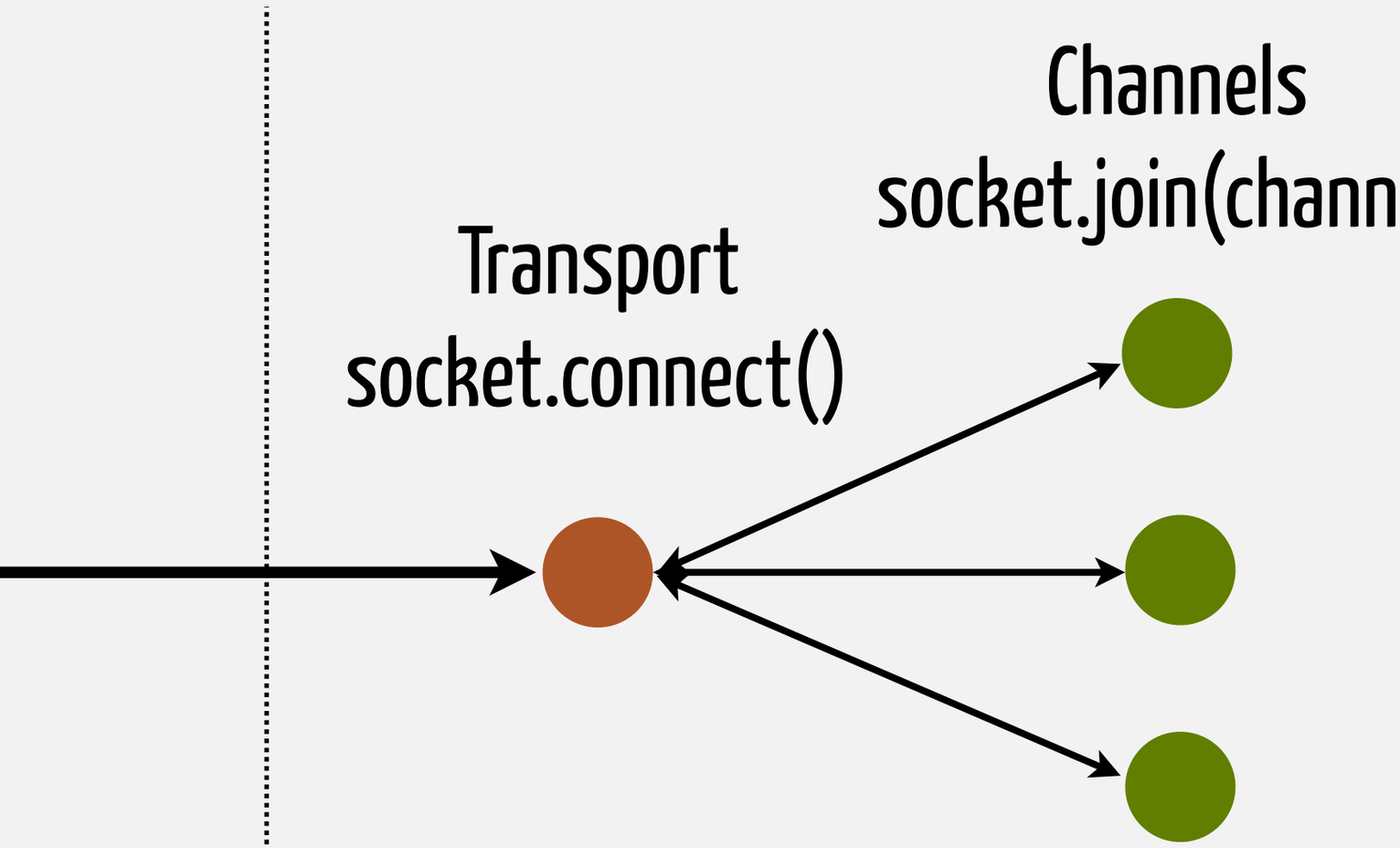
```elixir
defmodule Chat.UserSocket do
  use Phoenix.Socket

  channel "chat:lobby", Chat.LobbyChannel
  channel "room:*", Chat.RoomChannel

  # def connect(params, socket)
  # def id(socket)
end
```

```elixir
defmodule Chat.LobbyChannel do
  use Phoenix.Channel

  def join("chat:lobby", message, socket) do
    broadcast! socket, “user_joined”,
                %{username: message[“username"]}
    {:ok, socket}
  end

  def handle_in("new_message", message, socket) do
    broadcast socket, "new_message",
                %{content: message["content"],
                  username: messages[“username"]}
    socket
  end
end
```

Outside view

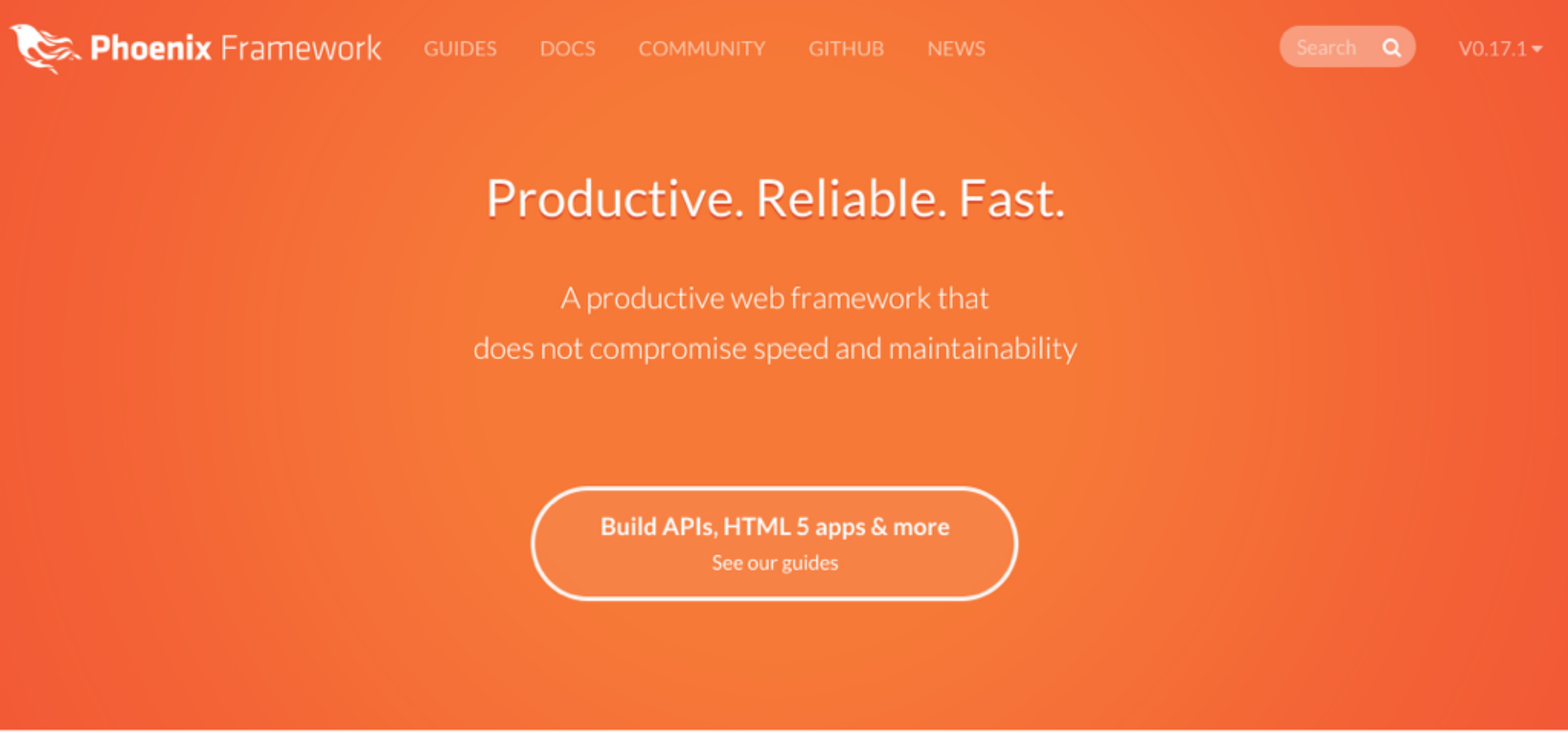


Inside view

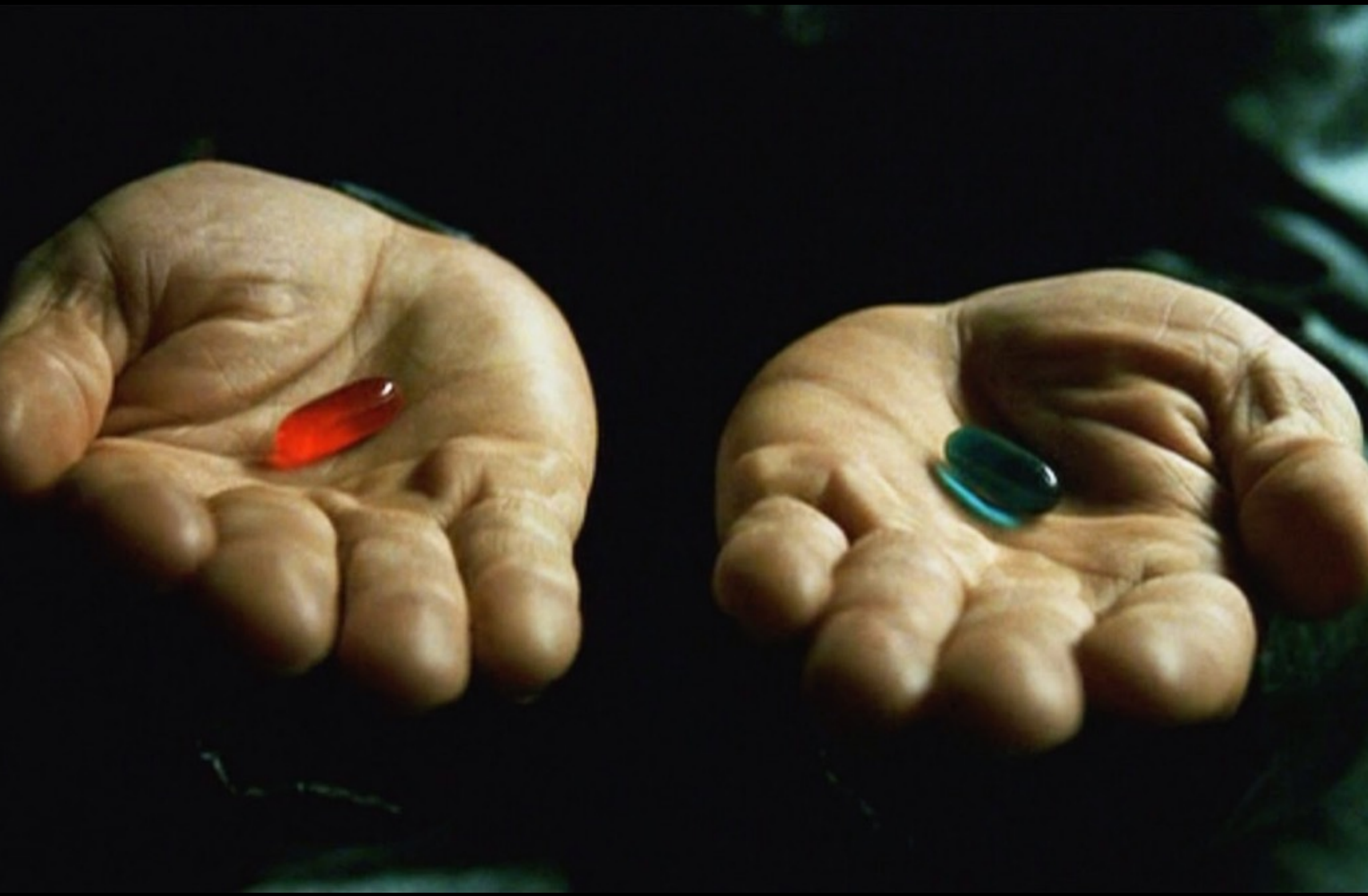


- Isolated
- Concurrent

- Distributed Erlang
- Redis
- PostgreSQL?
- XMPP?



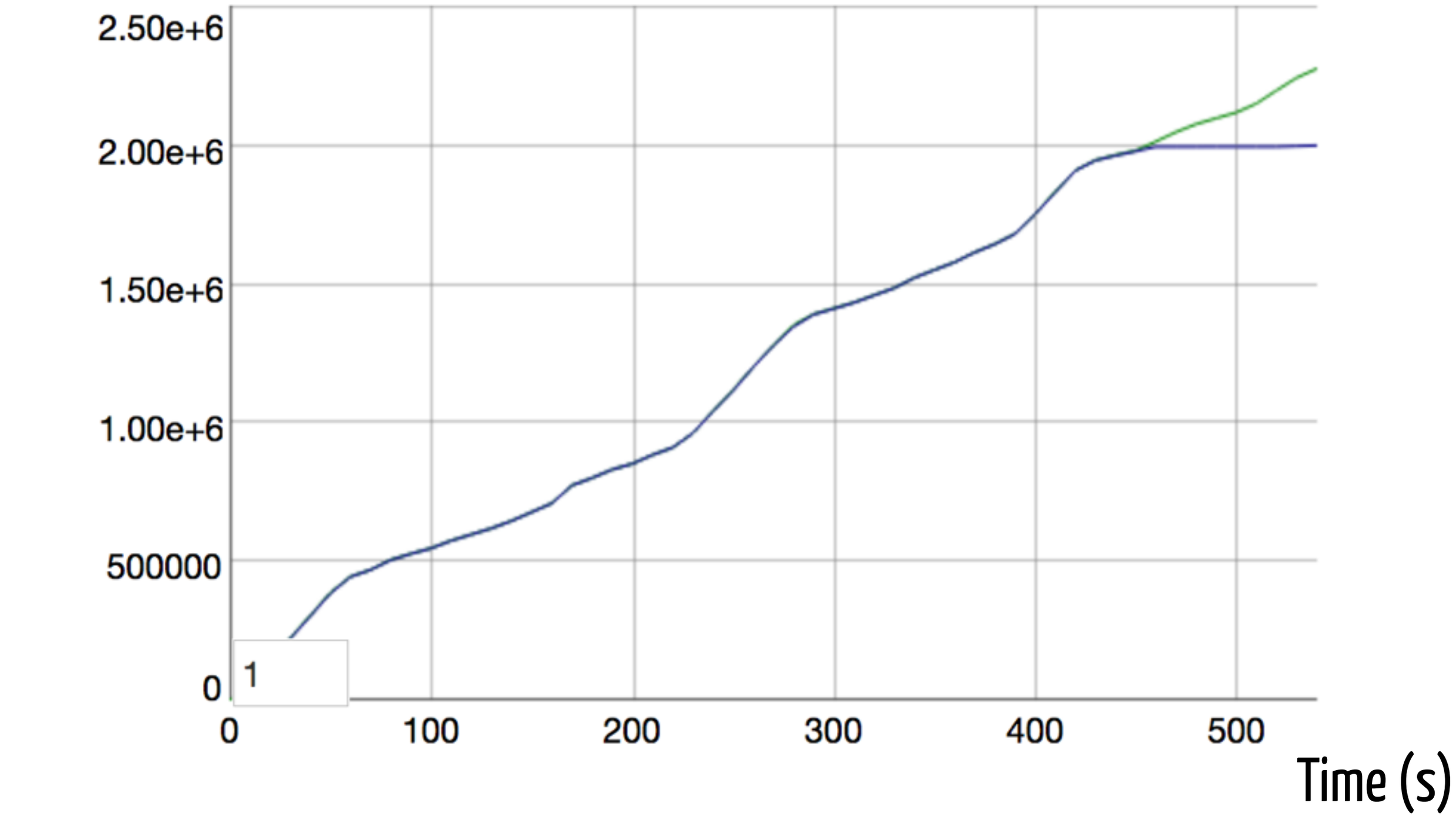
phoenixframework.org



Performance

Channels Performance

Subscribers per second



htop

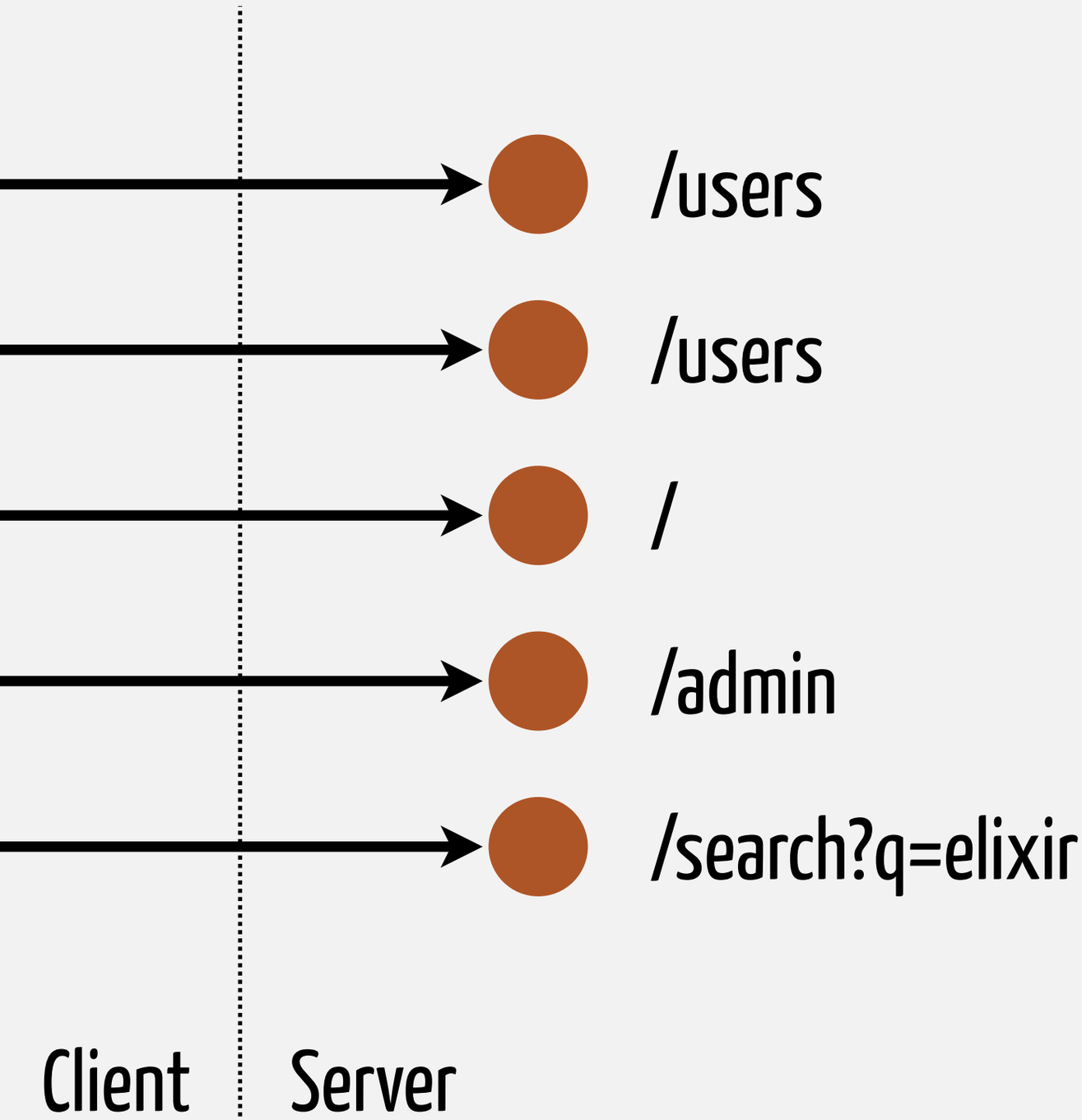
```
1  [      0.0%]    11 [|     0.5%]    21 [      0.0%]    31 [      0.0%]
2  [      0.0%]    12 [|     0.5%]    22 [      0.0%]    32 [      0.0%]
3  [      0.0%]    13 [      0.0%]    23 [      0.0%]    33 [      0.0%]
4  [|     1.0%]    14 [      0.0%]    24 [|     0.5%]    34 [      0.0%]
5  [|     0.5%]    15 [      0.0%]    25 [      0.0%]    35 [      0.0%]
6  [|     0.5%]    16 [      0.0%]    26 [      0.0%]    36 [      0.0%]
7  [      0.0%]    17 [      0.0%]    27 [      0.0%]    37 [      0.0%]
8  [|     1.0%]    18 [      0.0%]    28 [|     0.5%]    38 [      0.0%]
9  [      0.0%]    19 [      0.0%]    29 [      0.0%]    39 [      0.0%]
10 [      0.0%]    20 [      0.0%]    30 [      0.0%]    40 [      0.0%]
Mem[||||||||||||83765/128906MB]     Tasks: 22, 150 thr; 2 running
Swp[                     0/0MB]     Load average: 5.98 5.45 3.98
                                    Uptime: 5 days, 11:17:13

  PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
17402 root        20   0 84.9G 74.0G  6192 S 16.7 58.8 42:31.15 /usr/lib/er
17569 root        20   0 84.9G 74.0G  6192 S  0.0 58.8  0:22.80 /usr/lib/er
17570 root        20   0 84.9G 74.0G  6192 S  1.0 58.8  0:07.96 /usr/lib/er
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F1
```

HTTP(S) Performance

Library	Throughput (req/s)	Latency (ms)
Plug (elixir)	198 328	0.63
Phoenix (elixir)	179 685	0.61
Gin (go)	176 156	0.65
Play (scala)	171 236	1.89
Express Cluster (node)	92 064	1.24
Martini (go)	32 077	3.35
Sinatra (ruby)	30 561	3.50
Rails (ruby)	11 903	8.50

```
$ wrk -t20 -c100 -d30S --timeout 2000
https://github.com/mroth/phoenix-showdown
```

Inside view



- Isolated
- Concurrent

Isolated and Concurrent

- Crashes are isolated
- Data is isolated
  (GC is per process, no global pauses)
- Load balances on IO and CPU
  (efficient on multicore)

Productivity

Productivity

- Short-term productivity
  - Documentation / Guides
  - Workflows / Generators
- Long-term productivity
  - Introspection
  - Maintainability

INTRODUCTION

Overview
Installation
Learning
Community

GUIDES

Up And Running
Adding Pages
Routing
Plug
Controllers
Views
Templates
Channels
Ecto Models

TESTING

Introduction
Models

DEPLOYMENT

Up And Running

The aim of this first guide is to get a Phoenix application up and running as quickly as possi

Before we begin, please take a minute to read the Installation Guide. By installing any nece
we'll be able to get our application up and running smoothly.

At this point, we should have Elixir, Erlang, Hex, and the Phoenix archive installed. We shou
node.js installed to build a default application.

Ok, we're ready to go!

We can run `mix phoenix.new` from any directory in order to bootstrap our Phoenix applica
an absolute or relative path for the directory of our new project. Assuming that the name o
`hello_phoenix`, either of these will work.

```
$ mix phoenix.new /Users/me/work/elixir-stuff/hello_phoenix
```

```
$ mix phoenix.new hello_phoenix
```

> A note about Brunch.io before we begin: Phoenix will use Brunch.io f
> default. Brunch.io's dependencies are installed via the node package
> Phoenix will prompt us to install them at the end of the `mix phoenix.

The Pragmatic Programmers

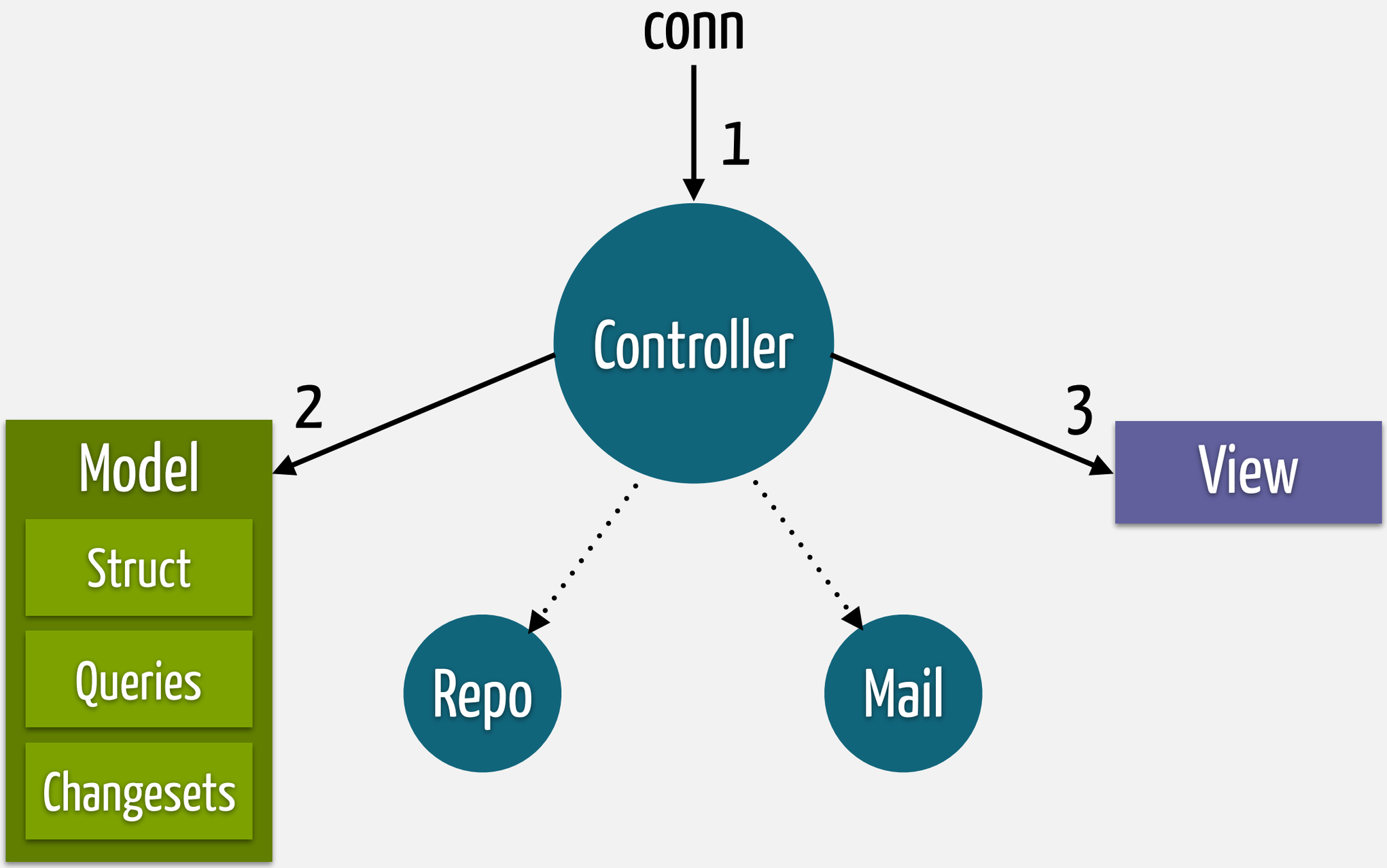
Programming Phoenix

Productive |> Reliable |> Fast

Chris McCord,
Bruce Tate,
and José Valim

edited by Jacquelyn Carter

Model / View / Controller



Generators as learning tools

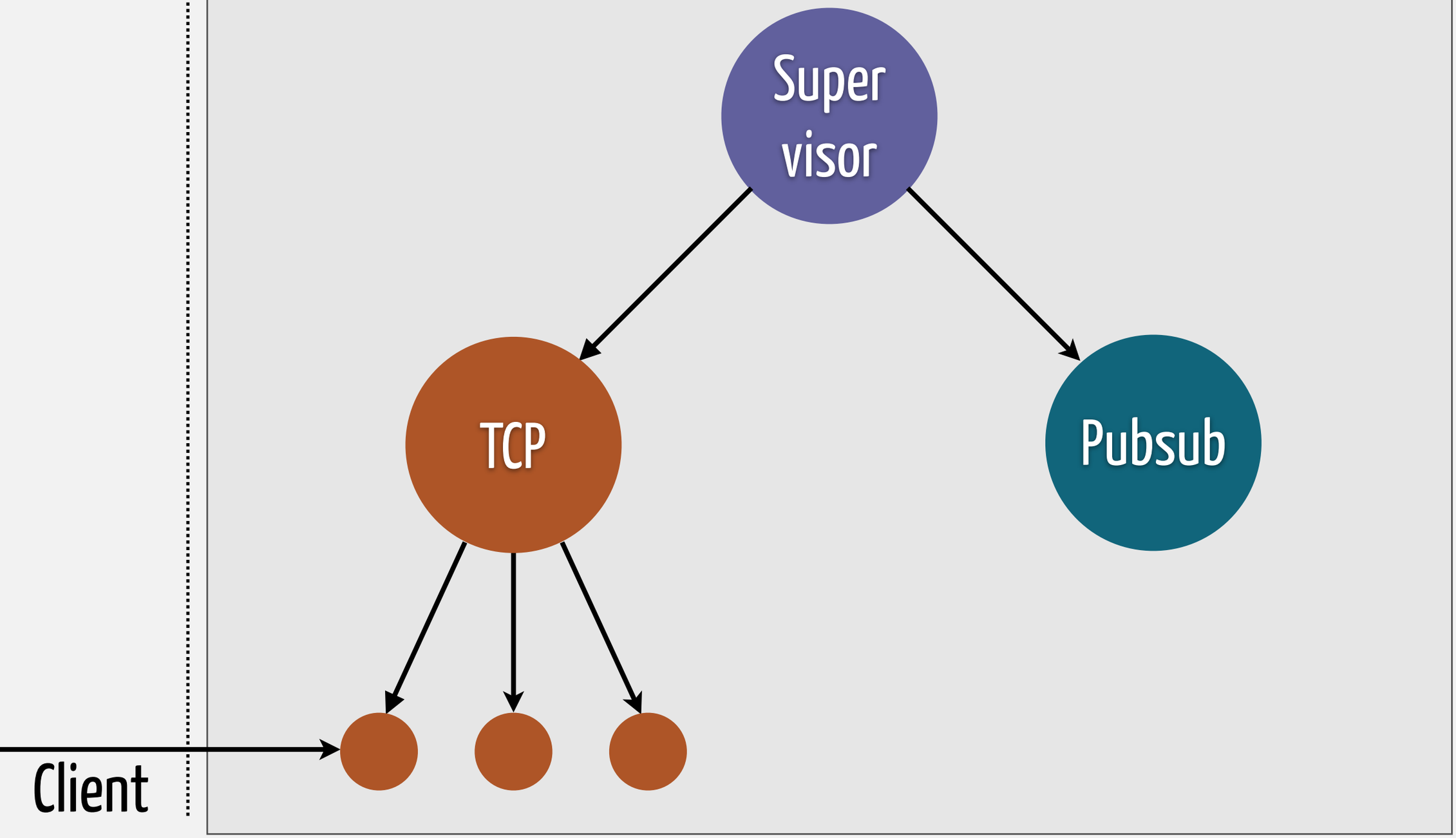
```
$ mix phoenix.gen.html
$ mix phoenix.gen.json
$ mix phoenix.gen.channel
```

More...

- Form builders
- Static build tools with ES6 as default
- Live reloading
- Pretty error pages
- First-class concurrent test tools
- Packages via hex.pm

Long term productivity: Applications

Application



Applications

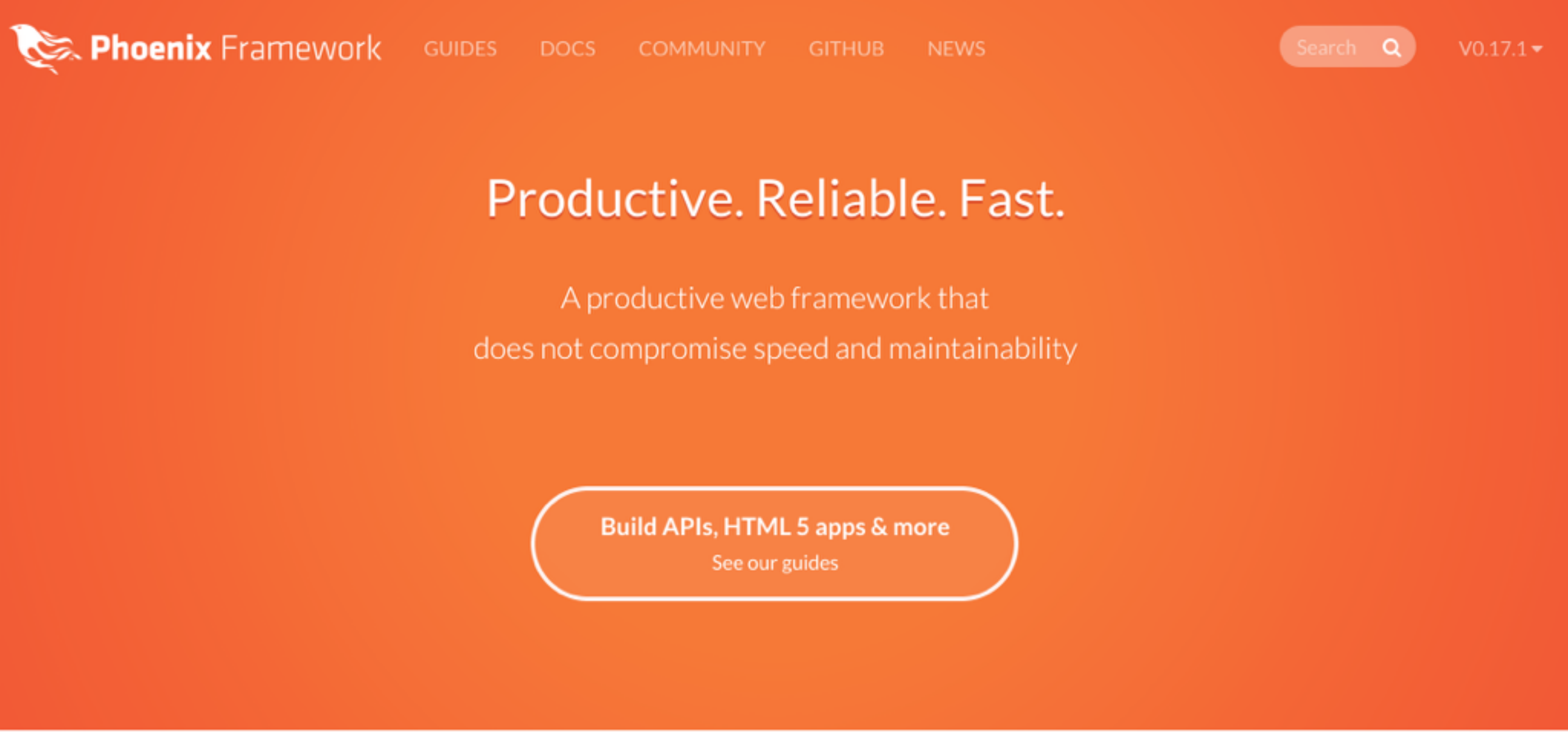
- Package and run our code
- Can be started and stopped as a unit
- Provide unified configuration
- Hold processes and state in the supervision tree

Observer Demo

Applications

- Introspection & Monitoring
- Visibility of the application state
- Easy to break into "components"
- Reasoning when things go wrong

Summing up



phoenixframework.org

The Pragmatic Programmers

Programming Phoenix

Productive |> Reliable |> Fast

Chris McCord,
Bruce Tate,
and José Valim

edited by Jacquelyn Carter

Elixir

elixir


```
defprotocol String.Inspect
  only: [BitString, List,

defimpl String.Inspect, fo
  def inspect(false), do:
  def inspect(true),  do:
  def inspect(nil),   do:
  def inspect(:""),   do:

  def inspect(atom) do
```

Elixir is a dynamic, functional language designed for building scalable and maintainable applications.

Elixir leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development and the embedded software domain.

To learn more about Elixir, check our getting started guide. Or keep reading to get an overview of the platform, language and tools.

Platform features

Scalability

All Elixir code runs inside lightweight threads of execution (called processes) that are isolated and exchange information via messages:

```
parent = self()

# Spawns an Elixir process (not an operating system one!)
```

News: Elixir v1.0 released

Search...

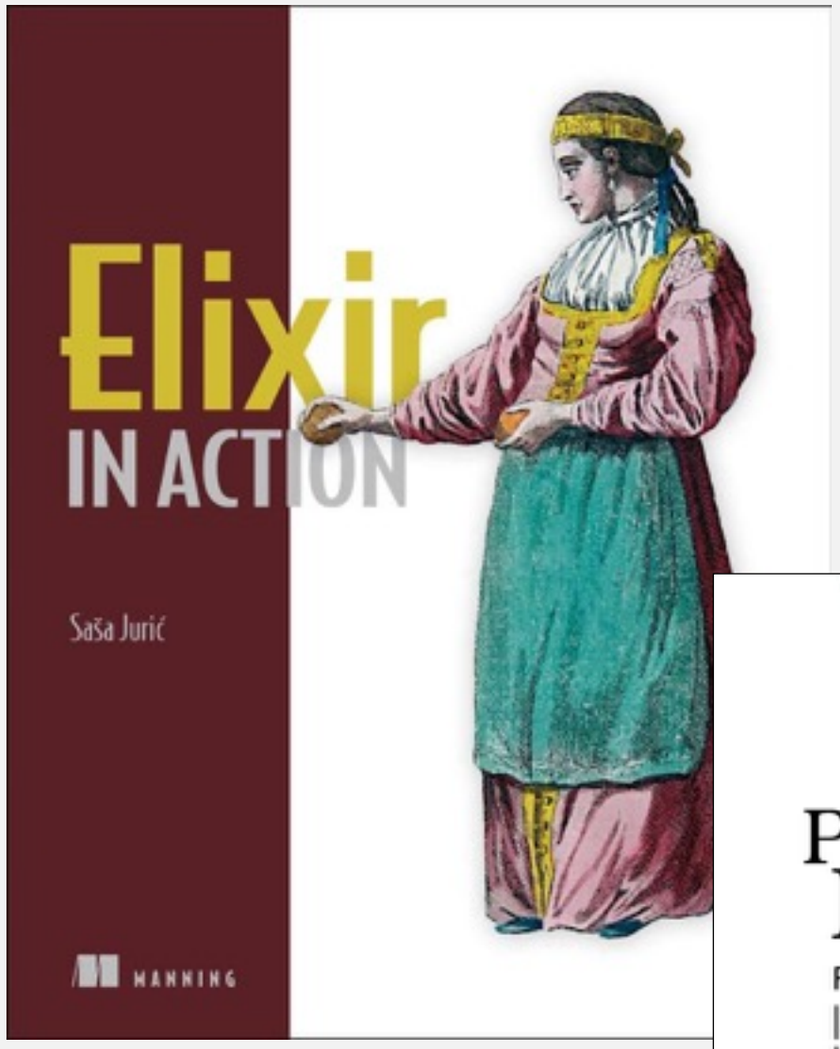
THE COMMUNITY

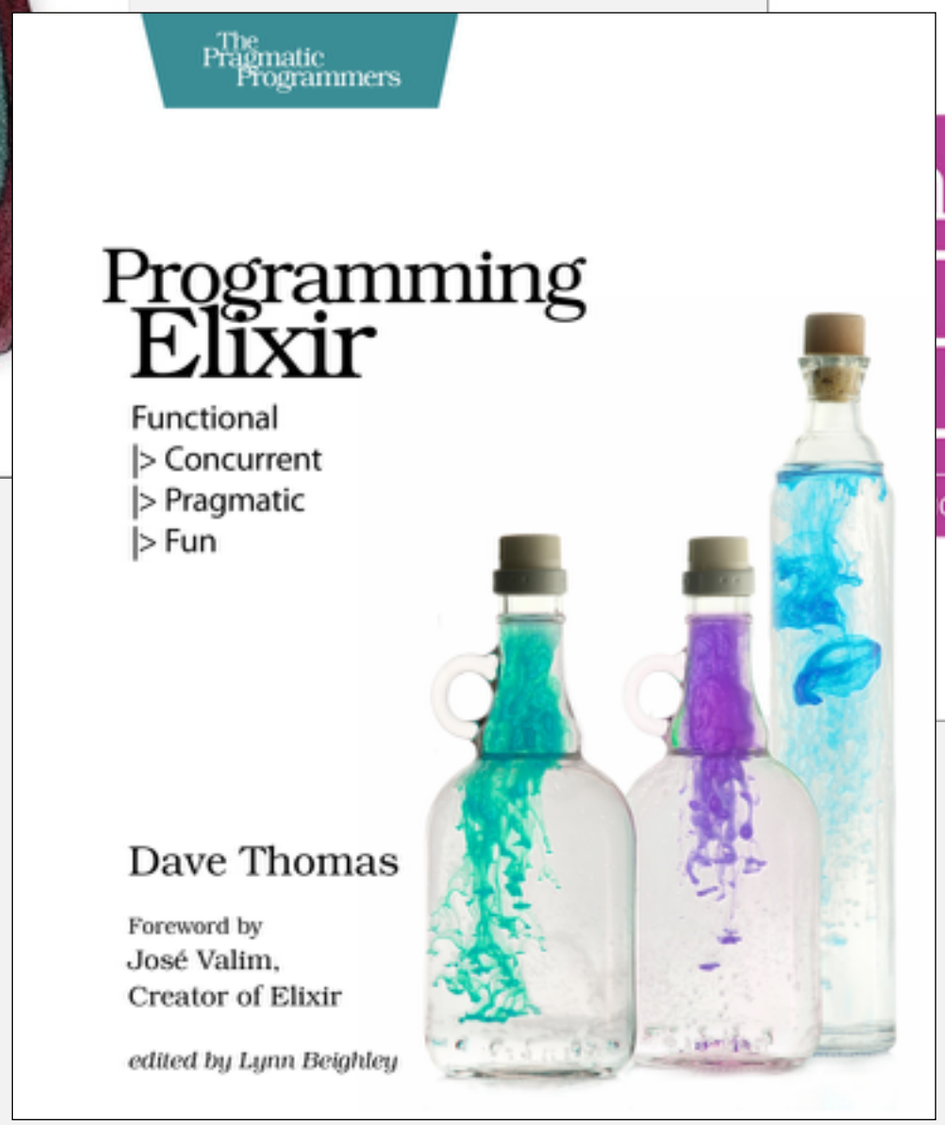
- #elixir-lang on freenode IRC
- elixir-talk mailing list (questions)
- elixir-core mailing list (development)
- Issue tracker
- @elixirlang on Twitter

IMPORTANT LINKS

- Source Code
- Wiki with events, resources and talks organized by the community
- Crash course for Erlang developers

elixir-lang.org


Elixir
IN ACTION
Saša Jurić
MANNING


The Pragmatic Programmers
Programming Elixir
Functional
|> Concurrent
|> Pragmatic
|> Fun
Dave Thomas
Foreword by
José Valim,
Creator of Elixir
edited by Lynn Beighley


O'REILLY
ntroducing
Elixir
G STARTED IN FUNCTIONAL PROGRAMMING
Simon St.Laurent & J. David Eisenberg

plataformatec

consulting and software engineering

Phoenix
Framework

@josevalim / phoenixframework.org