We Will Begin Shortly...

I MAKE MISTAKES SO YOU DON'T HAVE TO

BY JAMIE WINSOR

BUILDING AND SUPPORTING ONLINE EXPERIENCES

UNDEAD

LABS

STATE OF DECAY
YEAR-ONE SURVIVAL EDITION
BRONTO GAS
BURGER & FRIES

MOONRISE

STEAM™

PURPOSE OF THIS TALK

- Share insight on building online games
- Share key learnings from our journey

Some non-technical takeaways

- Competitive games are best balanced with constant player engagement
- A game complexity goes up, iteration times must go down

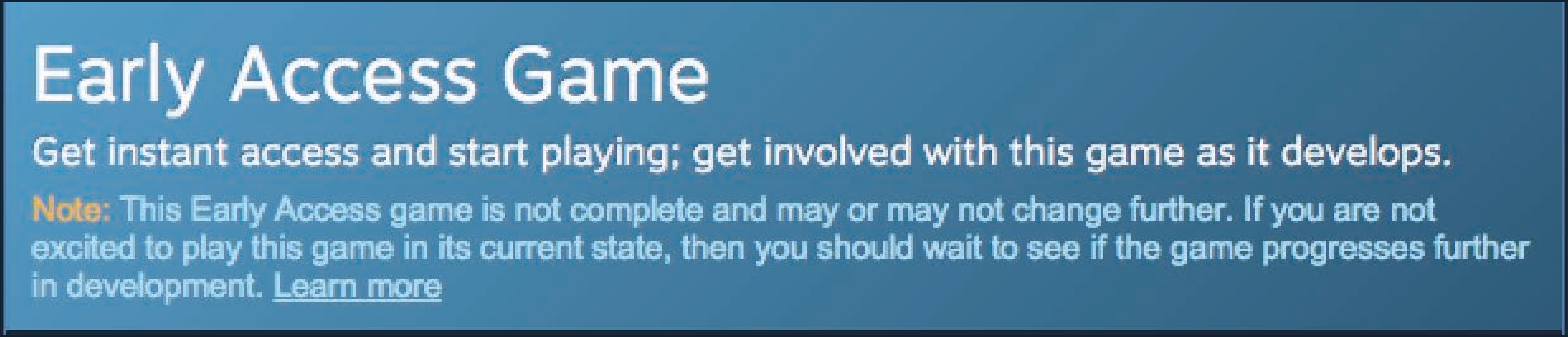
This can only be obtained through extensive beta testing, yet...

...No Beta Software allowed in AppStore

"SOFT LAUNCH"

Unfinished software is released in app stores of countries with lower usage/engagement

Iterating on mobile is far slower than on PC


Early Access Game

Get instant access and start playing; get involved with this game as it develops.

Note: This Early Access game is not complete and may or may not change further. If you are not excited to play this game in its current state, then you should wait to see if the game progresses further in development. Learn more

» Surprising amount of player engagement

» Listen to the active Steam community

Perception is Reality

- Players enjoy having the mobile client option for their favorite PC game
- That same game if presented as a mobile game first will be met with vitriol

BATTLE PERCEPTION AS A PRIORITY

» Did your game release in Korea first?

  Better make damn sure you have invert mouse on launch day.

» Was your game built for console first?

  That PC/Mouse support better be first class.

» Touted as a mobile game first?

  Graphical settings, hotkeys, and a PC friendly interface are a must.

Server Hosted Online Games

» Persistent Connection

» Stateful

» Multi-Service (micro-service)

» Persistent Storage

Game Back-end/Platform

- Authentication
- Chat / Presence
- Purchasing
- Player Storage
- Match Making
- Administration Tools

elixir

I WAS HAPPY WITH ERLANG

Getting others onboard was hard

Erlang STL 2.0

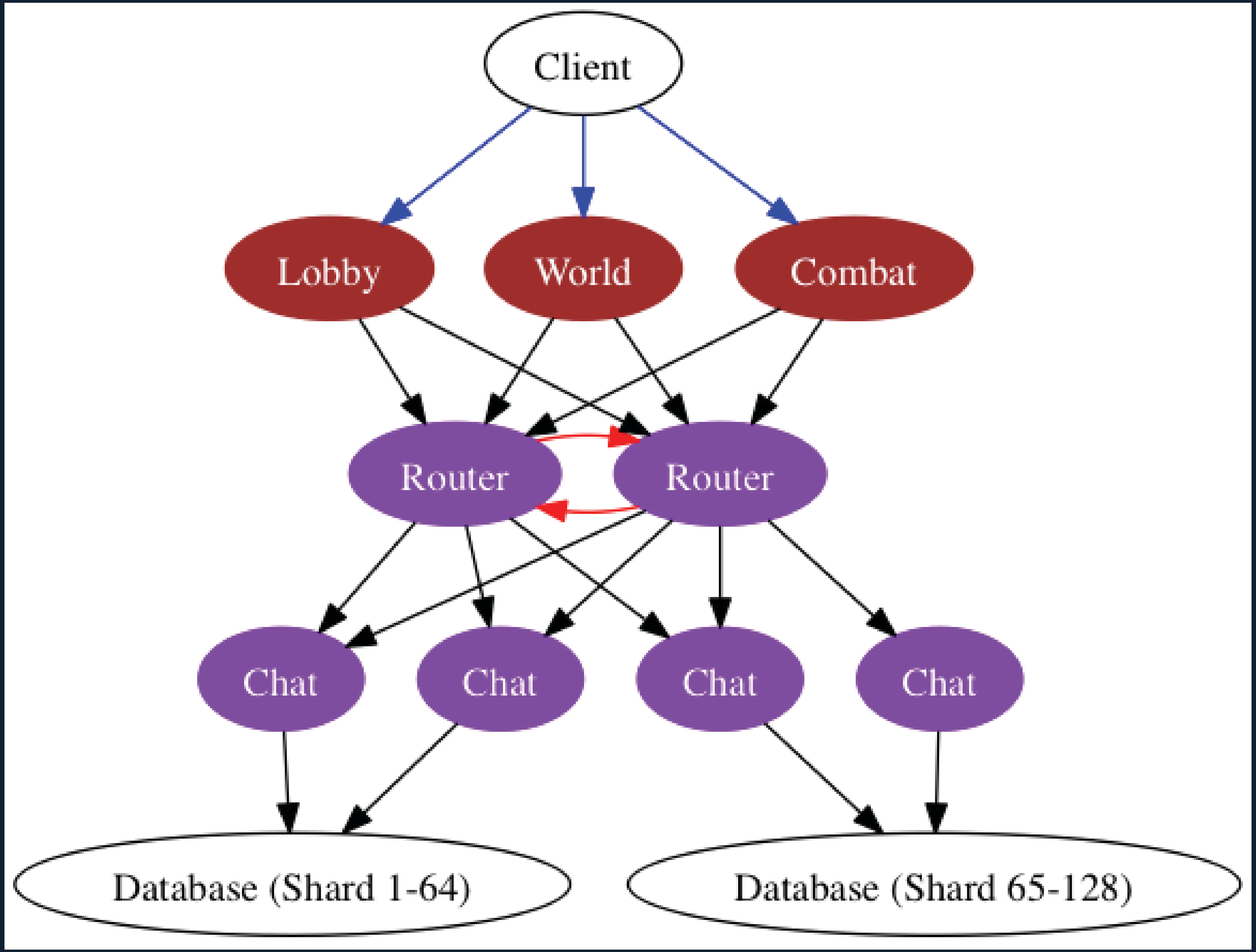
The Elixir Team Delivered so much more

- A Great Standard Library (stl)
- Build tool (Mix)
- Amazing Docs
- Package management (Hex)
- Polymorphism (protocols)
- Hygenic Macros

I Use Elixir Everyday
(2 Years Running)

- Easy To Get Started
- Not So Easy (at first) To Get Into Production
- Brilliant Once In Production

HIGHLEVEL LANDSCAPE



Elixir Project Breakdown

1. Protocol lib (tu_protocol)
2. Common lib (tu_common)
3. Route app (tu_route)
   - includes protocol & common
4. Chat app (tu_chat)
   - includes protocol & common

Route Server

- "Gateway" into the back-end
- Ranch Listener (TCP)
- Speak UndeadSrv Binary Protocol (tu_protocol)
- Disconnects misbehaving clients
  - Rate limit requests
  - Drop hanging / partial TCP connections

Routing a Message

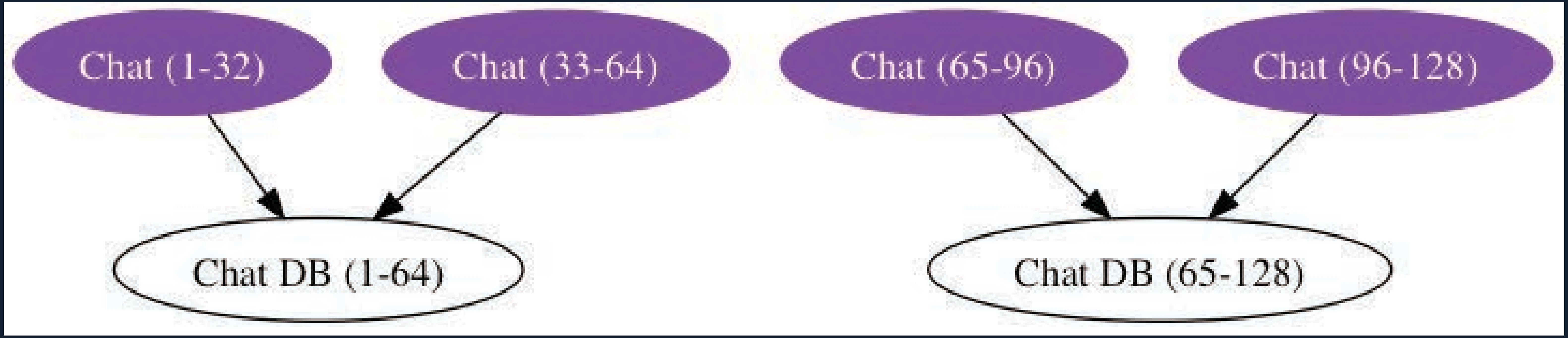
1. Read from TCP socket into buffer
2. Tag message with Route Acceptor pid
3. Route message via:
    1. Route Hash (deterministic/randomly deterministic)
    2. Service Protocol (i.e. Chat, Account, etc)

RECEIVING A ROUTED MESSAGE

- Service server receives message
- Internally dispatch request to application
- Reply to message if request is a transaction
  - Reply is sent to Route Acceptor pid
  - Route Acceptor serializes message into binary format for client
  - Route Acceptor sends message

ECTO & POSTGRESQL

Sharded via Schema



- Pre-shard data into X buckets (128) on X nodes (2)
  - Buckets are PostgreSQL schemas
  - Nodes are PostgreSQL instances

Embedded Shard / Create Date in Entities

```
CREATE SEQUENCE next_id_seq;
CREATE OR REPLACE FUNCTION next_id(OUT result bigint) AS $$
DECLARE
    our_epoch bigint := 1409266191000;
    seq_id bigint;
    now_millis bigint;
    shard_id int := 1;
BEGIN
    SELECT nextval('next_id_seq') % 1024 INTO seq_id;

    SELECT FLOOR(EXTRACT(EPOCH FROM clock_timestamp()) * 1000) INTO now_millis;
    result := (now_millis - our_epoch) << 23;
    result := result | (seq_id << 13);
    result := result | (shard_id);
END;
```

“SHARDING-IDS-AT-INSTAGRAM”

HOW DO I PUT IT INTO PRODUCTION?

THANKS FOR HANDING ME UNSTACKABLE CUPS
POINT DEFIANCE ZOO & AQUARIUM

STOP!

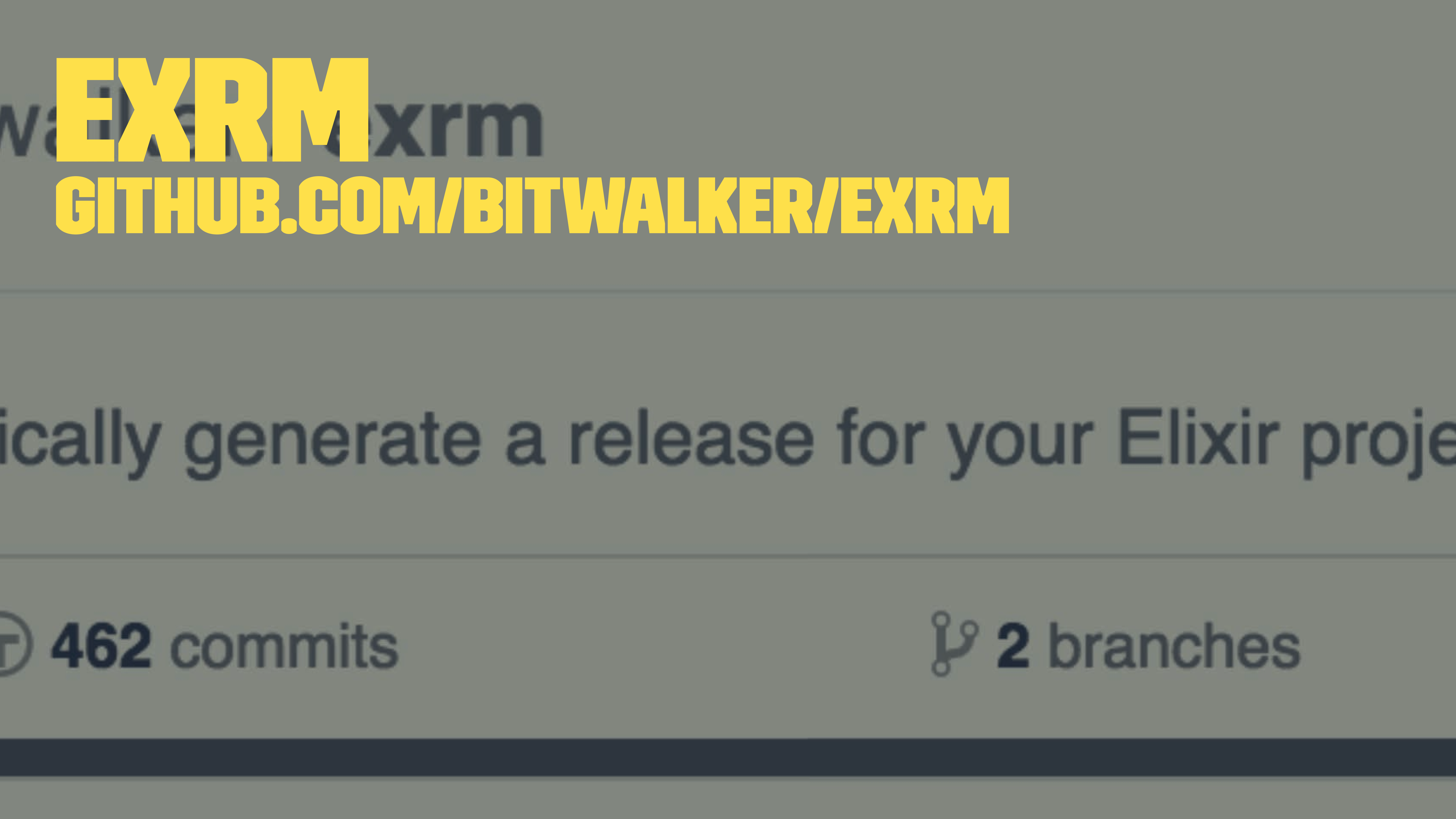
DON'T DEPLOY YOUR SOURCE CODE TO YOUR NODE.

CREATE A RELEASE

Dark Magic

EXRM

GITHUB.COM/BITWALKER/EXRM



wa… …xrm

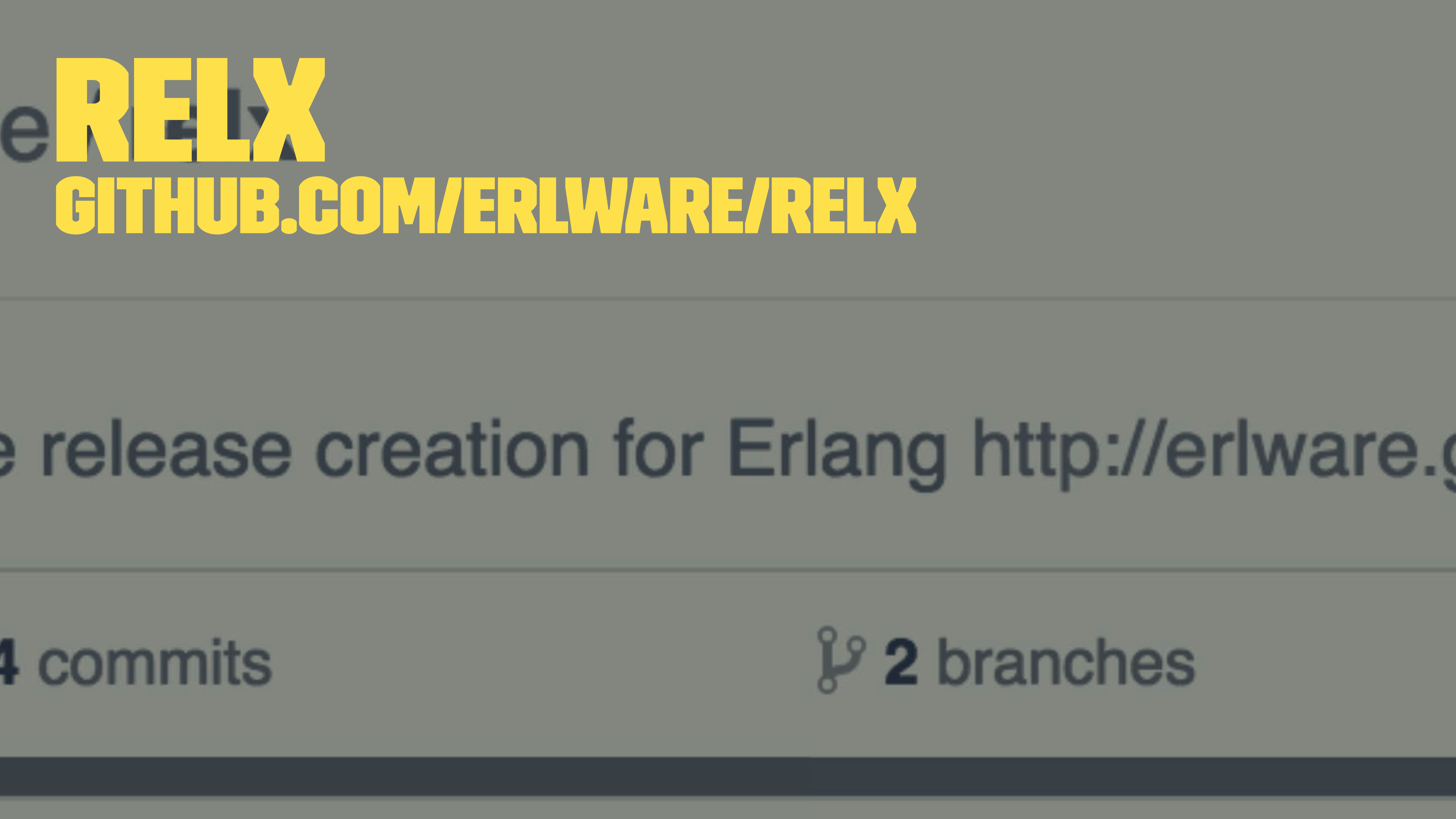
…ically generate a release for your Elixir proje…

462 commits

2 branches

RELX

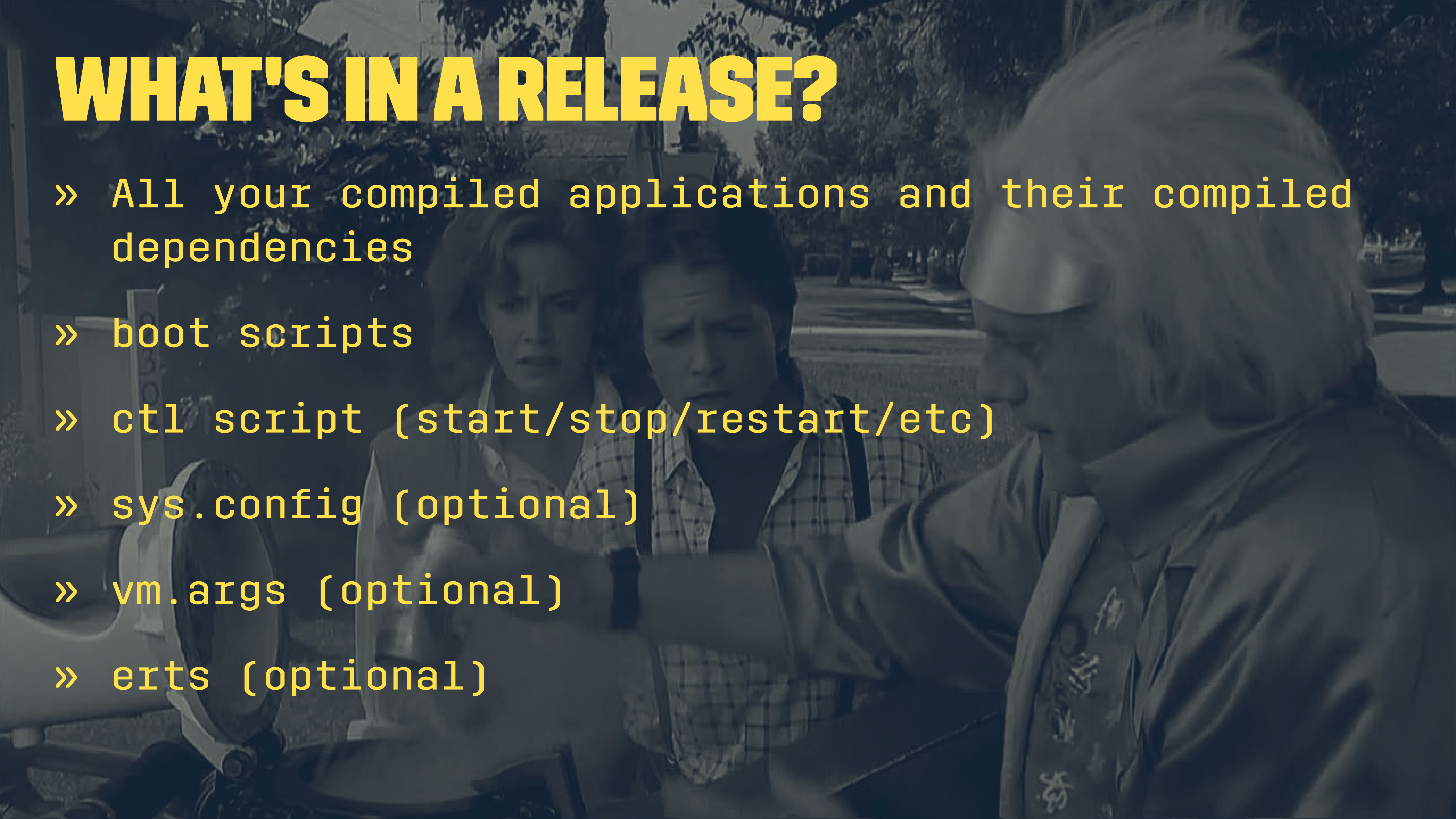
GITHUB.COM/ERLWARE/RELX



e release creation for Erlang http://erlware.

4 commits

2 branches



WHAT'S IN A RELEASE?

- All your compiled applications and their compiled dependencies
- boot scripts
- ctl script (start/stop/restart/etc)
- sys.config (optional)
- vm.args (optional)
- erts (optional)

9303

A release can contain many OTP applications

OTP Application

- A group of related code and processes
- Wrapped with OTP behaviours in a specific structure
- Informs the VM how to setup and teardown your application

Library in Mix

```
defmodule MyLib.Mixfile do
  use Mix.Project

  def application do
    [
      applications: [
        :logger,
        :crypto
      ]
    ]
  end

  # def project, do: ...
end
```

OTP APPLICATION IN MIX

```elixir
defmodule MyApp.Mixfile do
  use Mix.Project

  def application do
    [
      mod: {MyApp, []},
      registered: [],
      applications: [
        :logger,
        :crypto,
      ],
      env: []
    ]
  end

  # def project, do: ...
end
```

MODULE NOT FOUND

my_app/mix.exs

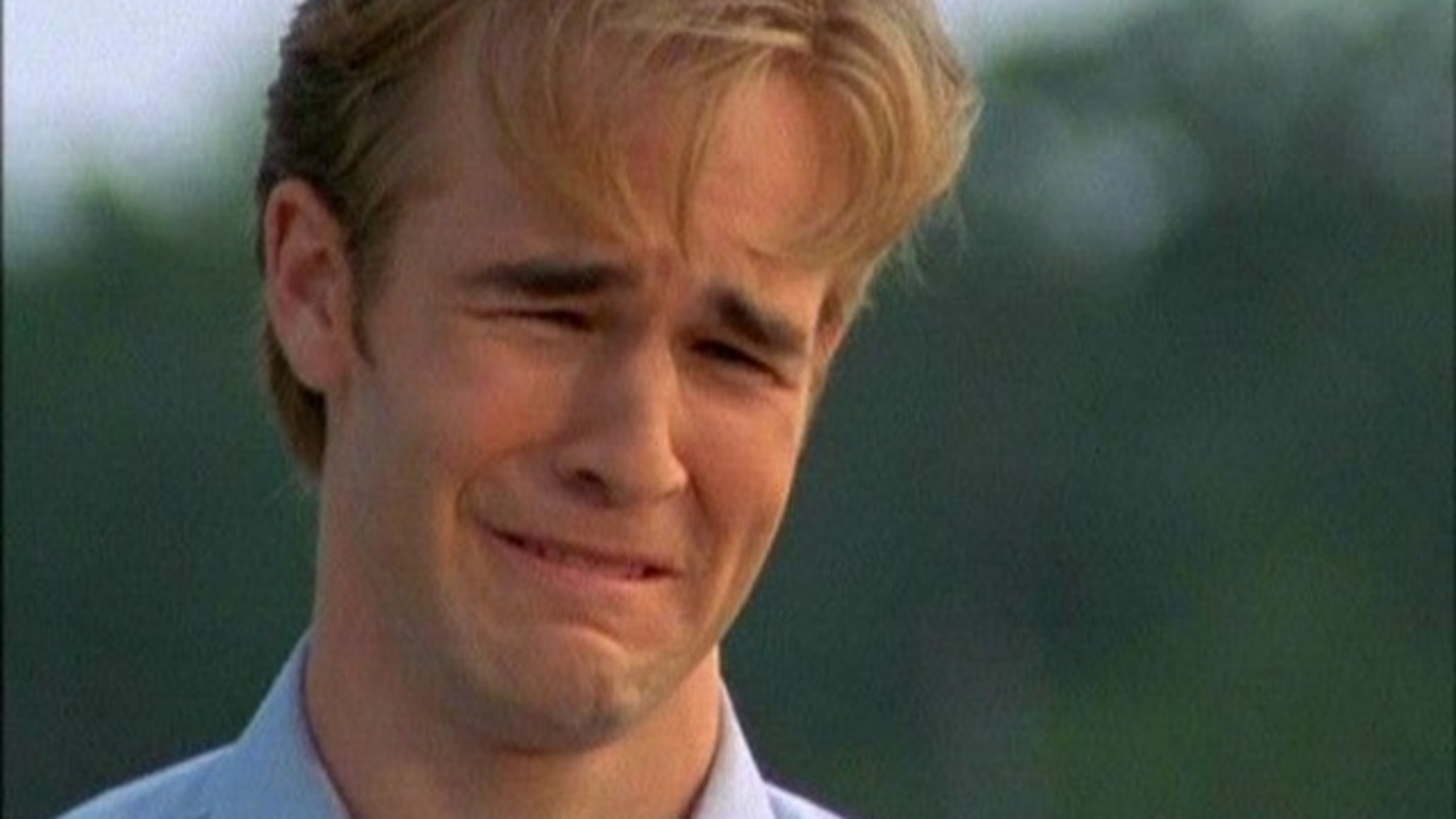
```elixir
def project do
  [
    app: :my_app,
    deps: [
      {:discovery, "~> 1.0"}
    ]
  ]
end

def application do
  [
    mod: {MyApp, []},
    applications: [
      :discovery
    ],
    env: []
  ]
end
```

Configuring a Release

- mix.exs?
- config.exs?
- sys.config?

config.exs

- » The pathway that Elixir exposes/encourages
- » Easiest to use and understand
- » Uses Elixir syntax
- » Not respected by release tooling



mix.exs

- Another pathway that Elixir exposes
- Uses Elixir syntax
- Useful for setting default configuration for your release
- Respected by release tooling
- Only useful for configuring your application, not dependencies

sys.config

- Not exposed by Elixir tooling
- Uses Erlang syntax
- Used for configuring deployed applications

Deploying a Release

Choose a Solution

1. A Build Server (Jenkins)
2. An Artifact Store (Github)
3. Configuration Management (Chef)
4. Hosting Solution (AWS)

Running Ecto Migrations

```
bash$ cd /opt/my_app
bash$ mix ecto.migrate
```

Mix Tasks are not a valid path

- Migrations are packaged and put onto database node
- Ecto Migrate requires
  - Application code
  - To start our application
- Database node doesn't need our application code
- Starting our app is bad if using service discovery

Migrator
github.com/reset/migrator

- A CLI binary used for performing database migrations
- Also supports MultiSchemaMigration
  - Useful for database sharding

Running migrations:

```
bash$ migrator up /path/to/migrations ecto://reset:pass@localhost:5432/account_db
```

Ecto Chef Cookbook

github.com/reset/ecto-cookbook

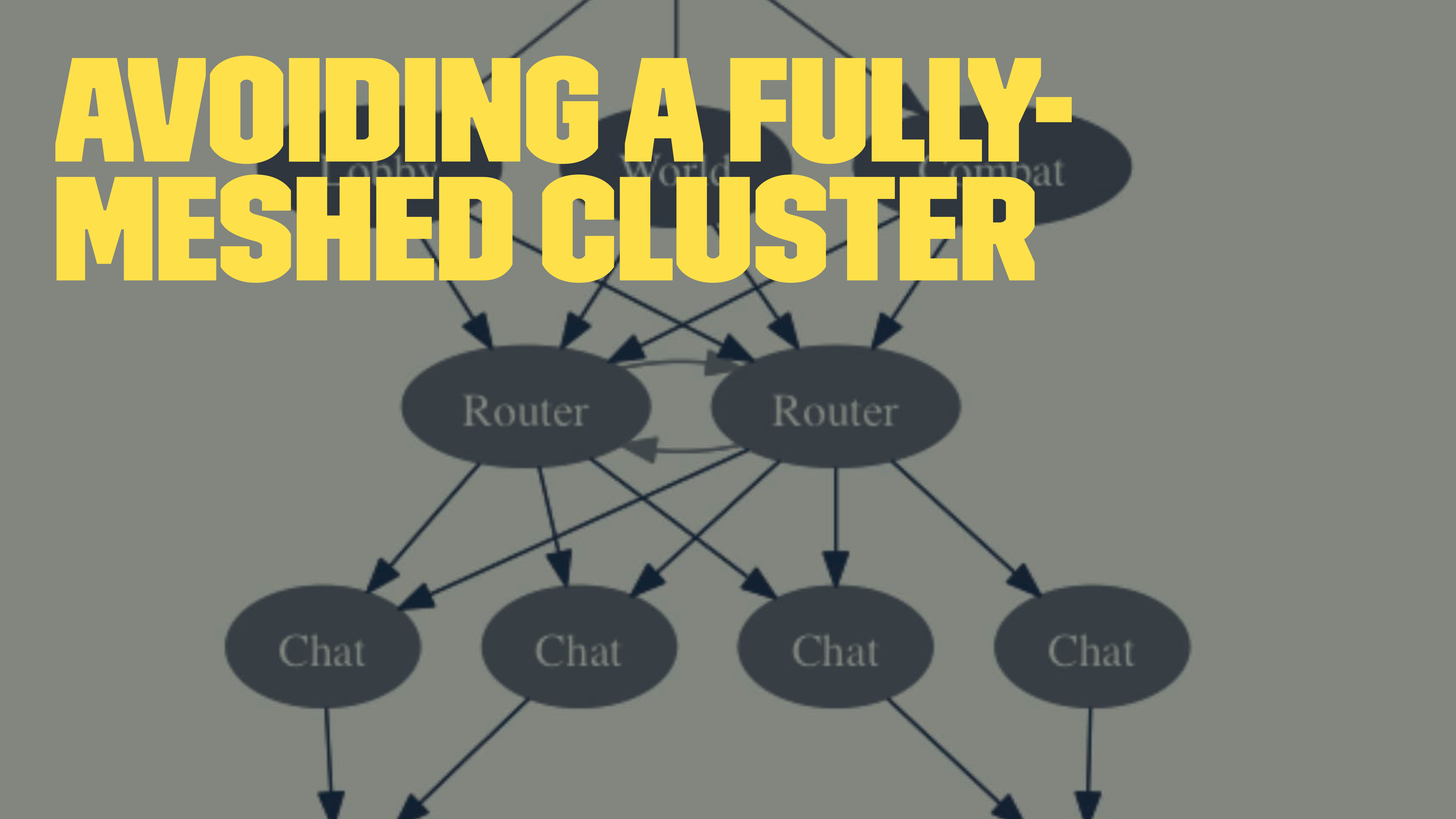
```
include_recipe "ecto::migrator"

ecto_migrate "account_db" do
  username "reset"
  password "pass"
  host "localhost"
  migrations_path "/path/to/migrations"
end
```

ECTO INTERACTING WITH SCHEMAS

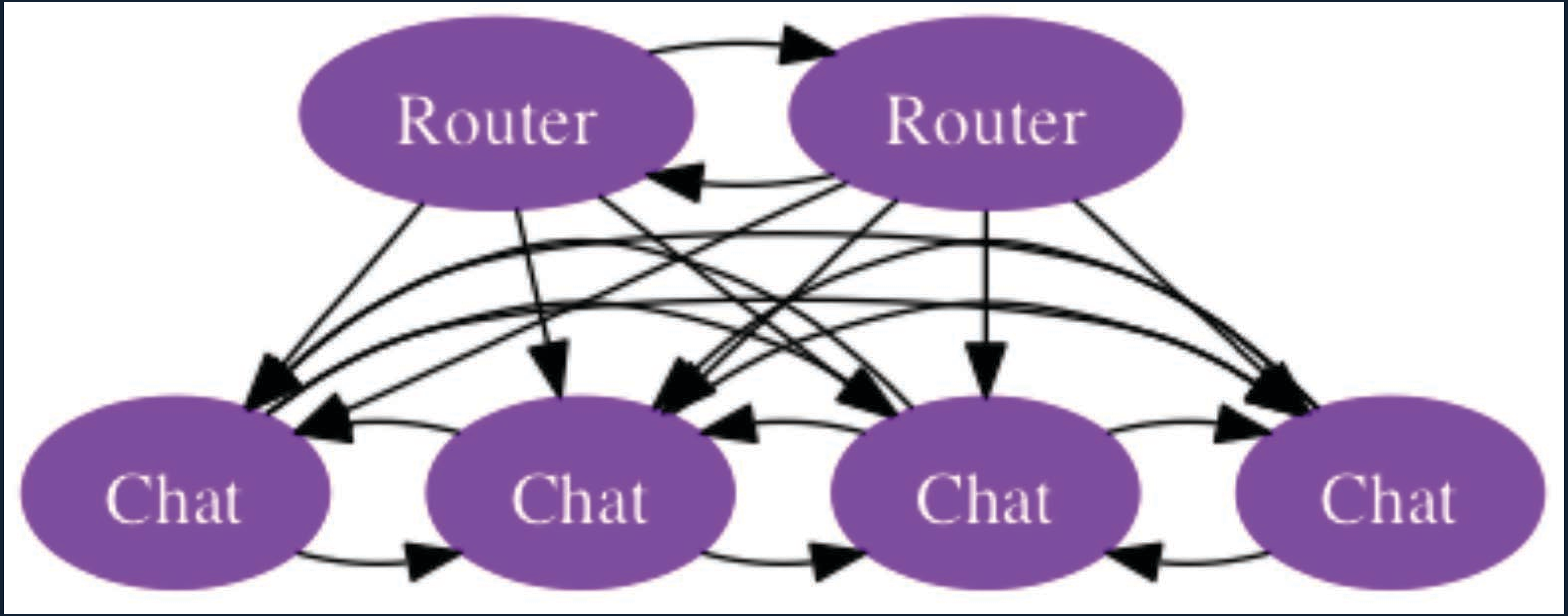
```
@spec put_prefix(Ecto.Query.t | Ecto.Model.t, binary) :: Ecto.Query.t | Ecto.Model.t
def put_prefix(%{__meta__: _meta} = model, prefix) do
  {_, source} = model.__meta__.source
  put_in model.__meta__.source, {prefix, source}
end
def put_prefix(%Ecto.Query{} = query, prefix), do: %{query | prefix: prefix}
def put_prefix(queryable, prefix) do
  Ecto.Queryable.to_query(queryable)
  |> put_prefix(prefix)
end
```

```
%MyApp.MyModel{name: "reset"}
|> put_prefix("account_1")
|> MyApp.Repo.insert()
```

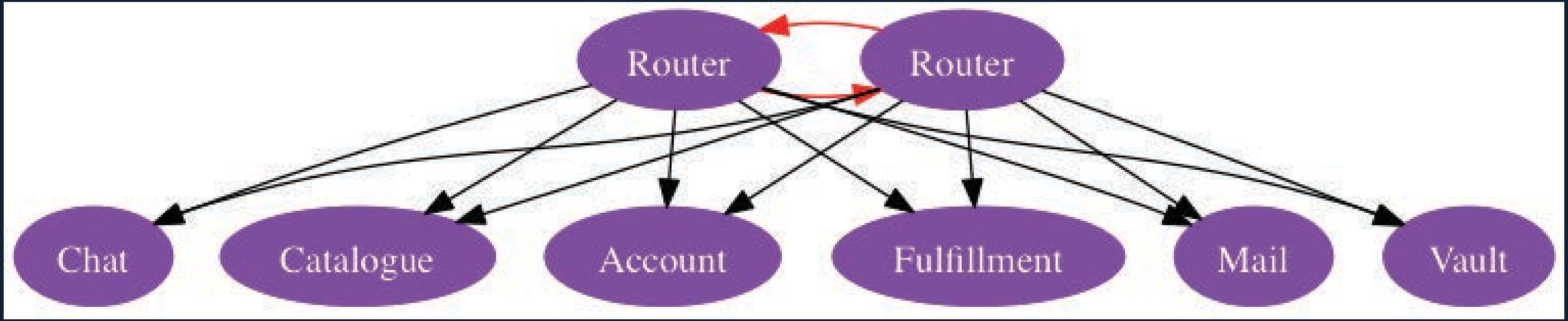
AVOIDING A FULLY-MESHED CLUSTER



DUMPSTER ON FIRE - WITH WHEELS - AT SCALE



PARTIALLY-MESHED



Hidden Nodes

- Configured with -hidden VM flag
  - Set on command line or in vm.args file
- Applied to all of our "service nodes"
- Can't use global processes
- Can't use libs that leverage global processes
- You must manually connect nodes to each other

DISCOVERY

AUTOMATICALLY DISCOVER AND CONNECT TO ERLANG NODES

GITHUB.COM/UNDEADLABS/DISCOVERY

PORTS AND AWS

EPMD (Erlang Port Mapper Daemon)

- By default: 4369
- Configurable with ERL_EPMD_PORT env variable
- Keeps track of which port each Erlang VM is running on for a given node
- Must be open between connecting Erlang nodes

inet_dist_listen

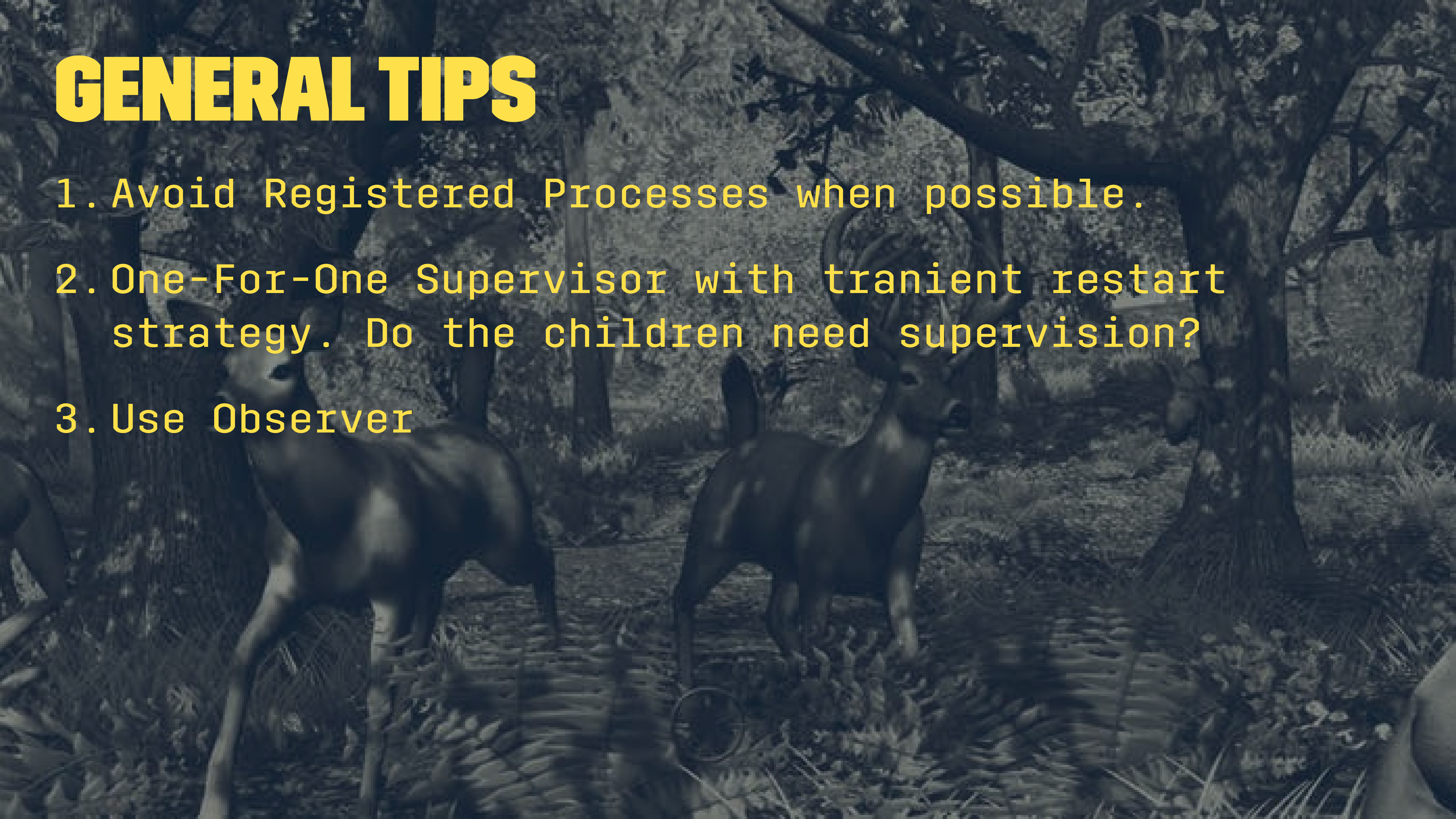
```
[
  {kernel, [
    {inet_dist_listen_min, 9100},
    {inet_dist_listen_max, 9600}
  ]}
].
```

» Range configured in sys.config

» Used to connect to other Erlang nodes

» Range must be open between connecting Erlang nodes

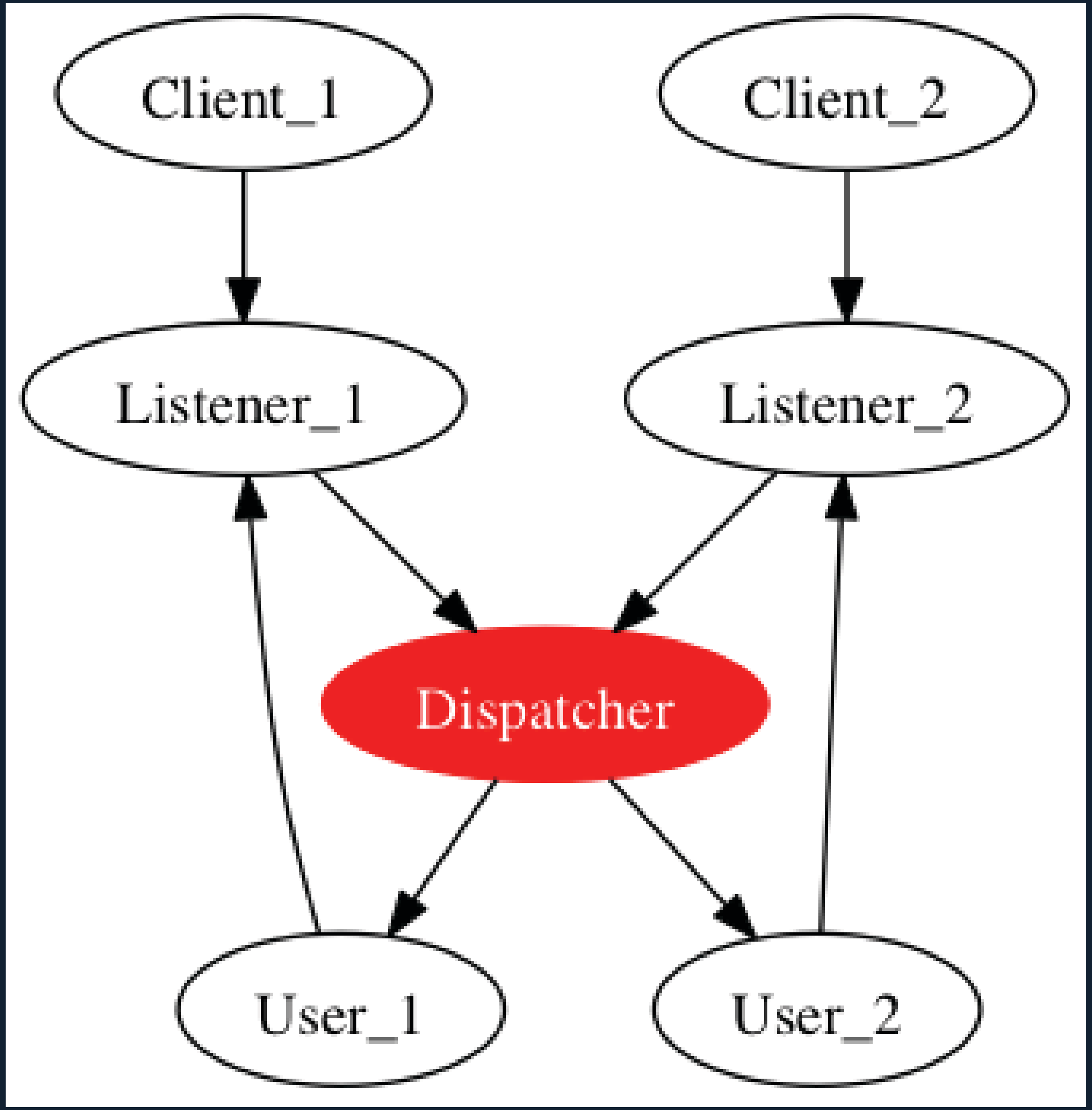
Hunting Bottlenecks

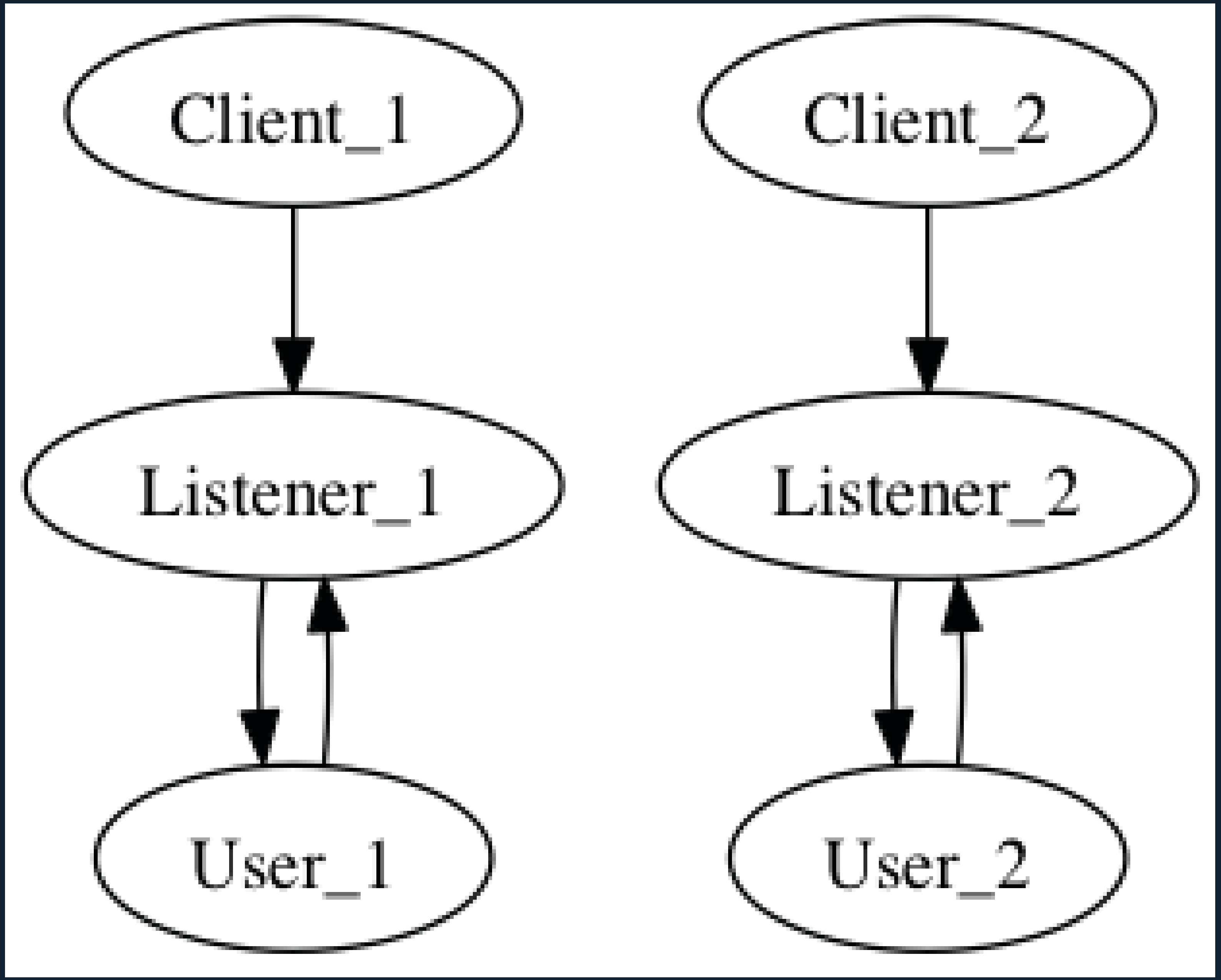


GENERAL TIPS

1. Avoid Registered Processes when possible.
2. One-For-One Supervisor with tranient restart strategy. Do the children need supervision?
3. Use Observer

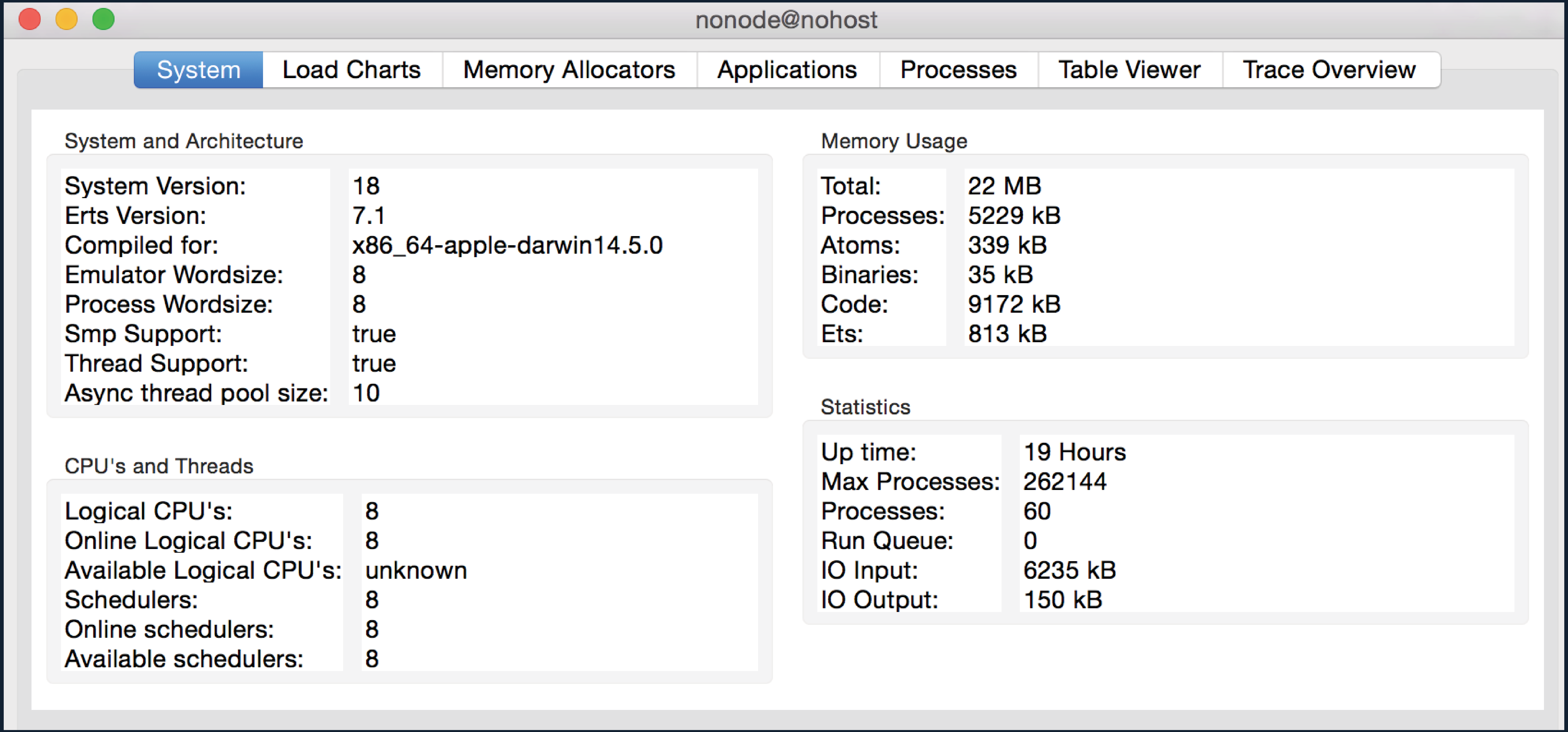
CONTRIVED PROCCESS BOTTLENECK





Using Observer

```
:observer.start()
```



System and Architecture

System Version:	18
Erts Version:	7.1
Compiled for:	x86_64-apple-darwin14.5.0
Emulator Wordsize:	8
Process Wordsize:	8
Smp Support:	true
Thread Support:	true
Async thread pool size:	10

CPU's and Threads

Logical CPU's:	8
Online Logical CPU's:	8
Available Logical CPU's:	unknown
Schedulers:	8
Online schedulers:	8
Available schedulers:	8

Memory Usage

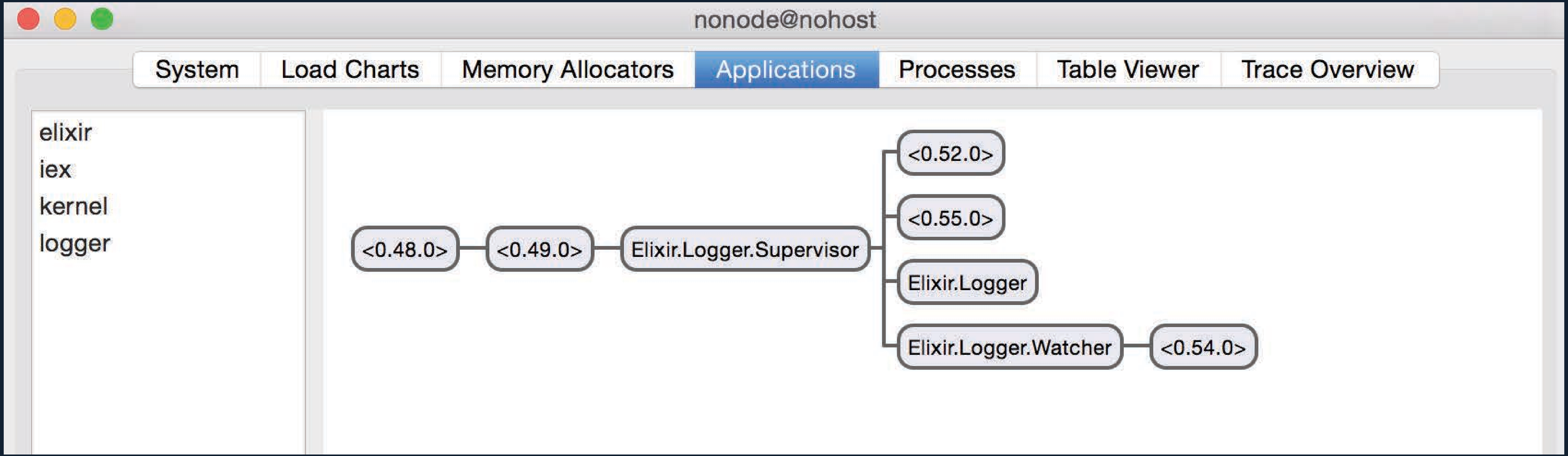
Total:	22 MB
Processes:	5229 kB
Atoms:	339 kB
Binaries:	35 kB
Code:	9172 kB
Ets:	813 kB

Statistics

Up time:	19 Hours
Max Processes:	262144
Processes:	60
Run Queue:	0
IO Input:	6235 kB
IO Output:	150 kB

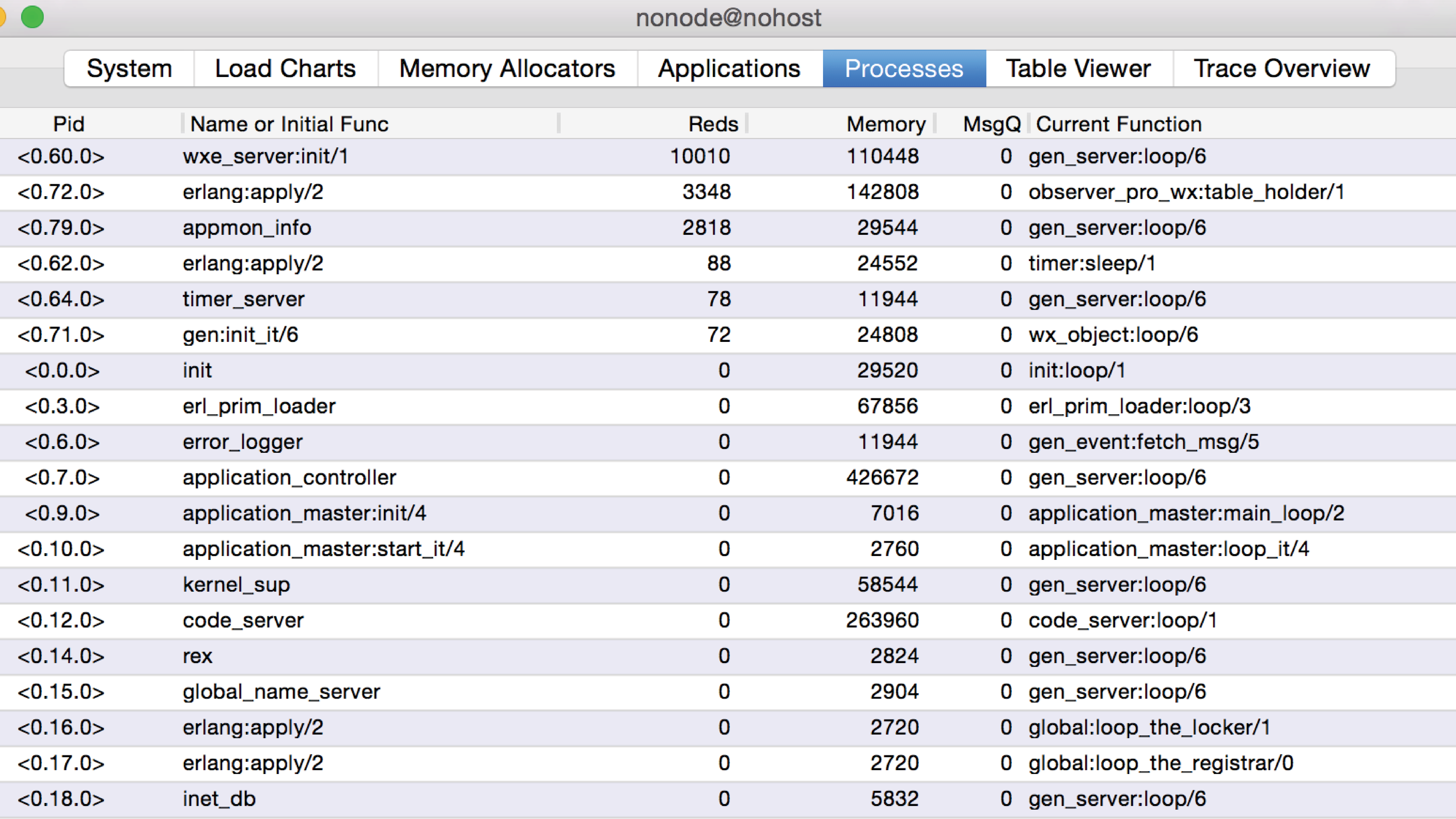
Applications

Select an application to view it's entire supervision tree


nonode@nohost
System
Load Charts
Memory Allocators
Applications
Processes
Table Viewer
Trace Overview
elixir
iex
kernel
logger
<0.48.0>
<0.49.0>
Elixir.Logger.Supervisor
<0.52.0>
<0.55.0>
Elixir.Logger
Elixir.Logger.Watcher
<0.54.0>

Process List

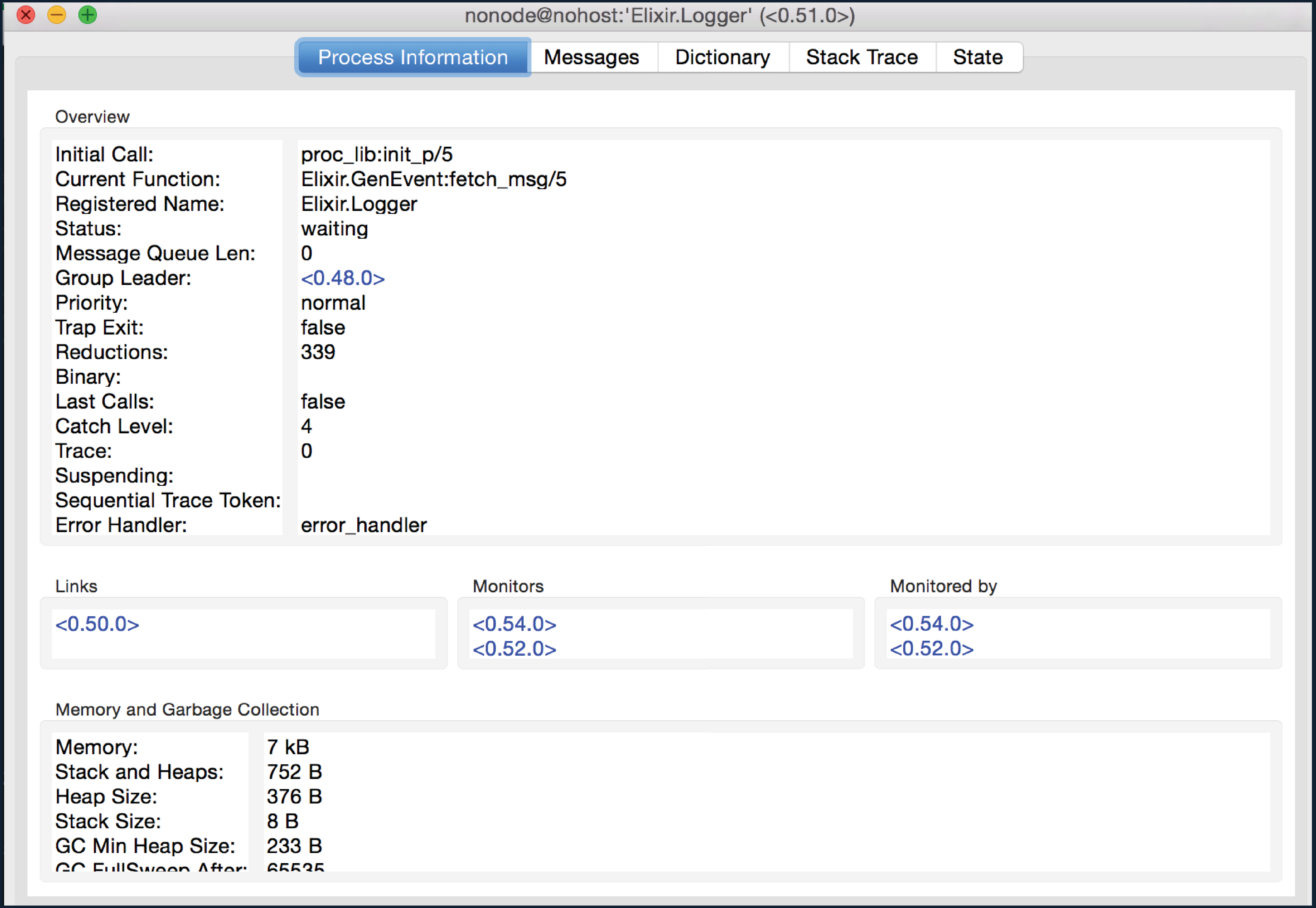
- Process Identifier (Pid)
- Reductions (Reds)
- Memory
- Message Queue (MsgQ)



Pid	Name or Initial Func	Reds	Memory	MsgQ	Current Function
<0.60.0>	wxe_server:init/1	10010	110448	0	gen_server:loop/6
<0.72.0>	erlang:apply/2	3348	142808	0	observer_pro_wx:table_holder/1
<0.79.0>	appmon_info	2818	29544	0	gen_server:loop/6
<0.62.0>	erlang:apply/2	88	24552	0	timer:sleep/1
<0.64.0>	timer_server	78	11944	0	gen_server:loop/6
<0.71.0>	gen:init_it/6	72	24808	0	wx_object:loop/6
<0.0.0>	init	0	29520	0	init:loop/1
<0.3.0>	erl_prim_loader	0	67856	0	erl_prim_loader:loop/3
<0.6.0>	error_logger	0	11944	0	gen_event:fetch_msg/5
<0.7.0>	application_controller	0	426672	0	gen_server:loop/6
<0.9.0>	application_master:init/4	0	7016	0	application_master:main_loop/2
<0.10.0>	application_master:start_it/4	0	2760	0	application_master:loop_it/4
<0.11.0>	kernel_sup	0	58544	0	gen_server:loop/6
<0.12.0>	code_server	0	263960	0	code_server:loop/1
<0.14.0>	rex	0	2824	0	gen_server:loop/6
<0.15.0>	global_name_server	0	2904	0	gen_server:loop/6
<0.16.0>	erlang:apply/2	0	2720	0	global:loop_the_locker/1
<0.17.0>	erlang:apply/2	0	2720	0	global:loop_the_registrar/0
<0.18.0>	inet_db	0	5832	0	gen_server:loop/6

Process Info

- Process Information
- Messages
- Dictionary
- Stack Trace
- State



Overview

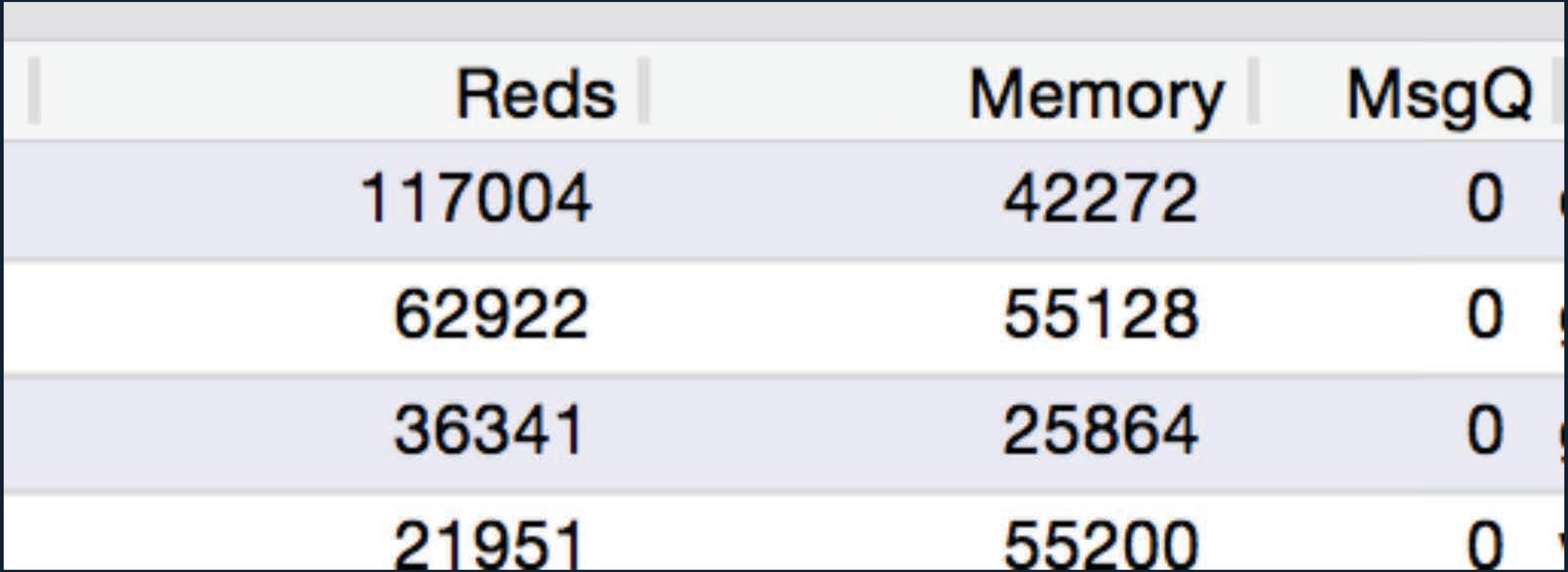
Initial Call:	proc_lib:init_p/5
Current Function:	Elixir.GenEvent:fetch_msg/5
Registered Name:	Elixir.Logger
Status:	waiting
Message Queue Len:	0
Group Leader:	<0.48.0>
Priority:	normal
Trap Exit:	false
Reductions:	339
Binary:	
Last Calls:	false
Catch Level:	4
Trace:	0
Suspending:	
Sequential Trace Token:	
Error Handler:	error_handler

Links	Monitors	Monitored by
<0.50.0>	<0.54.0>	<0.54.0>
	<0.52.0>	<0.52.0>

Memory and Garbage Collection

Memory:	7 kB
Stack and Heaps:	752 B
Heap Size:	376 B
Stack Size:	8 B
GC Min Heap Size:	233 B
GC FullSweep After:	65535

LOW REDS & EMPTY QUEUE



Reds	Memory	MsgQ
117004	42272	0
62922	55128	0
36341	25864	0
21951	55200	0

Reductions?

- Elixir Processes are scheduled on a reduction count basis.
- One reduction is roughly equivalent to a function call.
- A process is allowed to run until it pauses (receive/yield) or until it has executed ~1000 reductions.

Preemptive Scheduling

Erlang Scheduler Qeueus

- `:max` (Reserved for internal use. Don't use this.)
- `:high`
- `:normal`
- `:low`

Priority Profiles

- Strict
- Fair

Strict

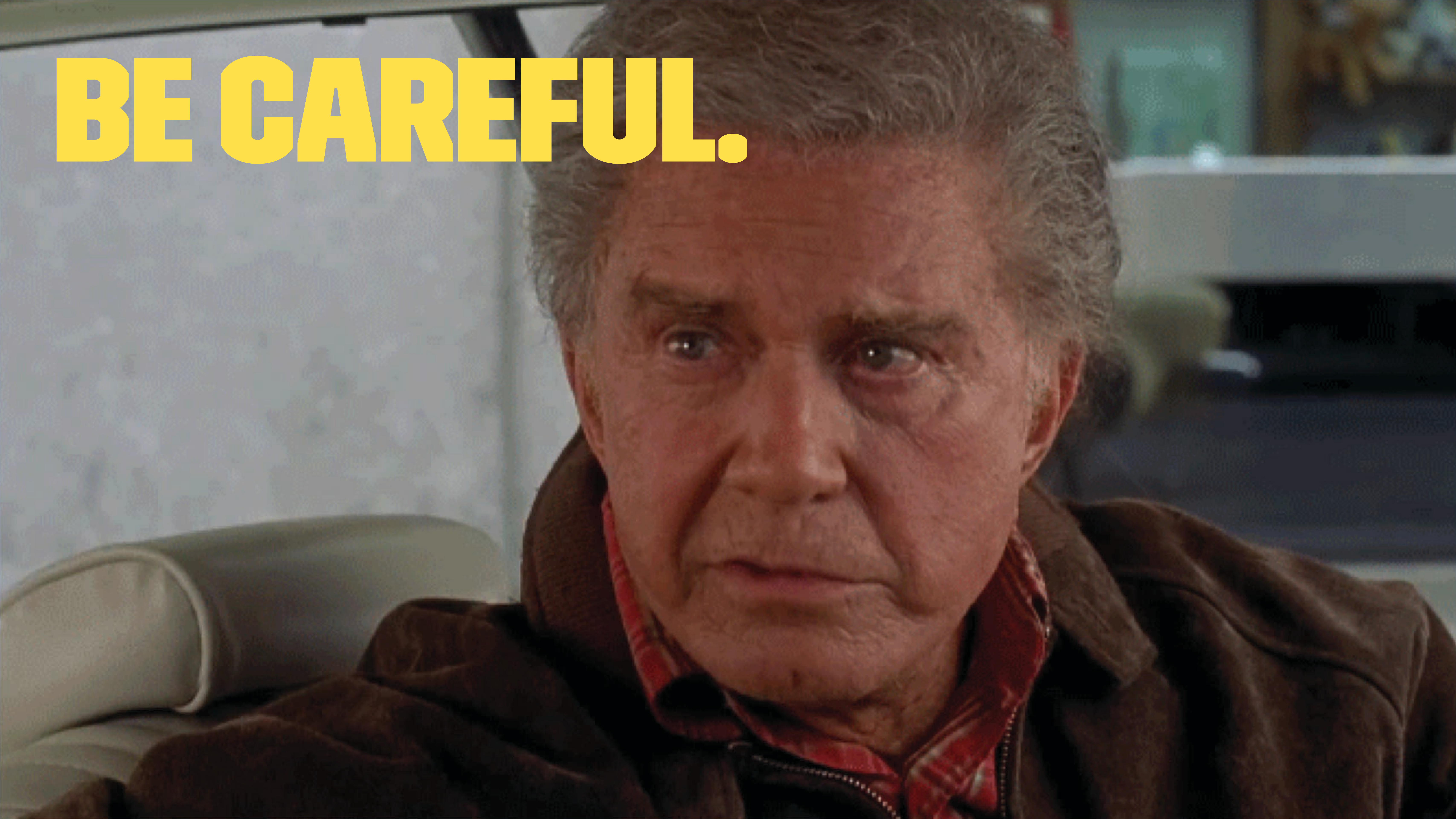
- :max and :high are strict
- Scheduler processes all messages in :max queue
- Scheduler processes all messages in :high queue
- Scheduler then moves to fair queues

FAIR

- :normal and :low are fair priority
- Scheduler processes :normal queue until empty or for a total of 8,000 reductions
- Scheduler then processes one :low process if available

Configuring Priority

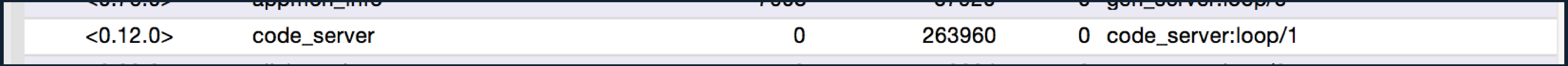
```
iex> Process.flag(:priority, :high)
=> :normal
iex> Process.flag(:priority, :normal)
=> :high
```



BE CAREFUL.

Protocol Consolidation

Code server has high reductions and high msgqueue causing bottlenecks in all areas of codebase



```
<0.12.0>    code_server    0    263960    0  code_server:loop/1
```

Protocol consolidation cont.
Protocols not found in the release

- Are the protocols actually in my path?

```
iex> Protocol.consolidated?(Enum)
=> false
iex> :code.get_path()
```

Why isn't Ecto starting my workers at application start?

Configuring Ecto

```
[
  {tu_account, [
    {'Elixir.TUAccount.Repo', [
      {lazy, false}
    ]}
  ]}
].
```

Why is Ecto not commiting data to the database?

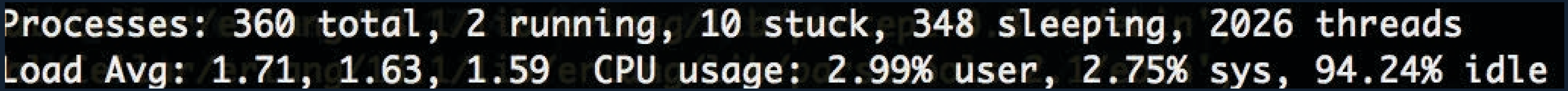
Build per Environment

- Create a build for each "environment"
  - Prod
  - Test
  - Dev
- Some libraries, like Ecto, expect that this feature is on
- Some app configuration options only configurable at build time

One VM per Machine/Container

All CPU are pegged with very high 1m and 5m load while all apps seem to be fine.

```
bash$ top
```



```
Processes: 360 total, 2 running, 10 stuck, 348 sleeping, 2026 threads
Load Avg: 1.71, 1.63, 1.59  CPU usage: 2.99% user, 2.75% sys, 94.24% idle
```

Tuning VM Args

1. Enable Kernel Poll +K true
2. Setup Async Thread Pool +A 60

```
bash$ cat /path/to/release/vm.args
-name reset@arena.local
-setcookie MY_COOKIE
-hidden
+K true
+A 60
```

JAMIE WINSOR
@RESETEXISTENCE
GITHUB.COM/RESET