The Lifecycle of a Running Release

Release the Kraken… Err… Elixir!

Challenge: I want to run some code, one time, at startup…

There's More

- Work jobs in the background while your app continues to run
- Do some tasks periodically
- Bring new nodes online or take nodes offline
- And more…

Expert: You have OTP applications and supervision trees!

I just want to run some code
at a specified time

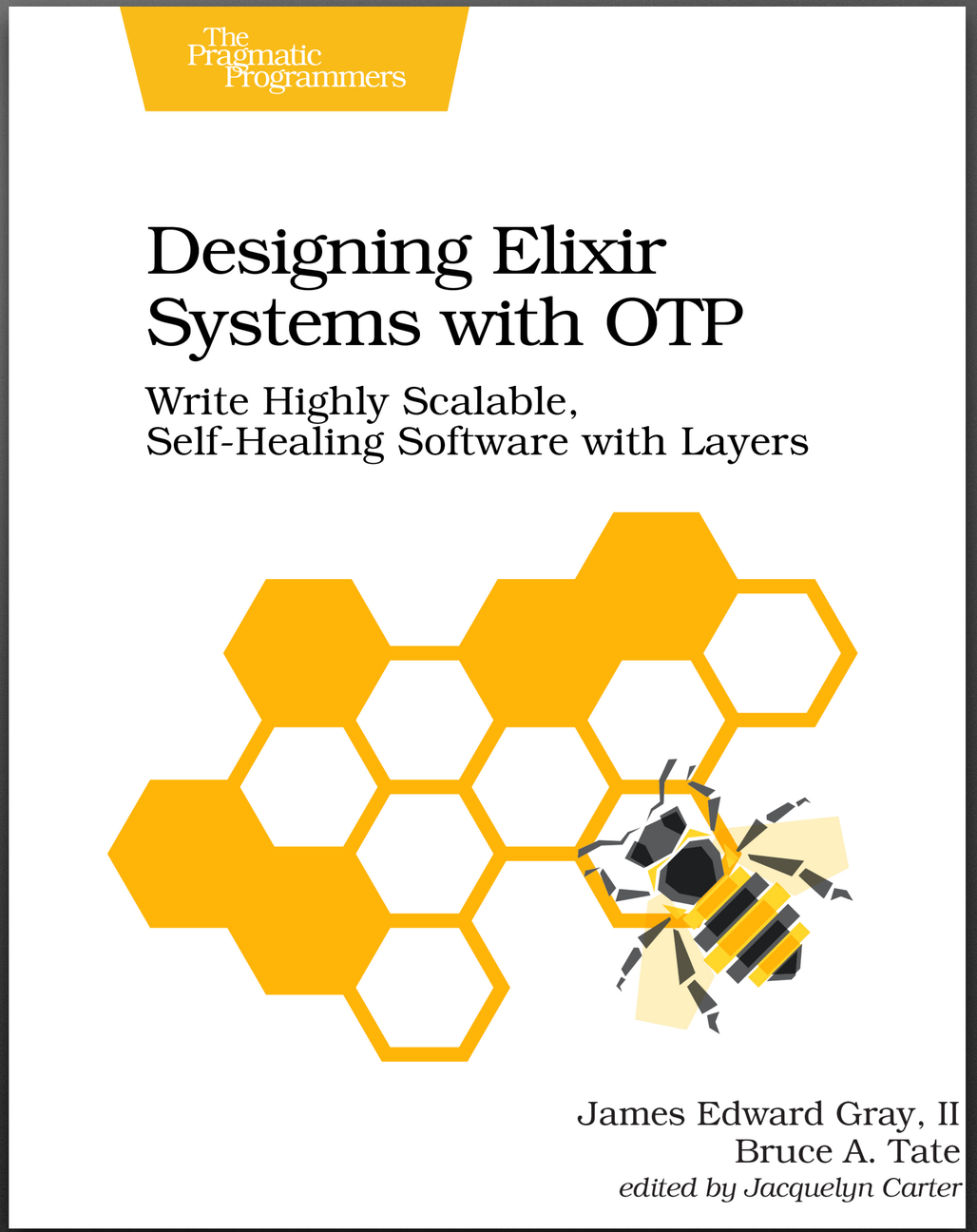
Expert: You have OTP applications and supervision trees!

In Rails, I can use environment files, the application file, the initializers directory, and such

Expert: You have OTP applications and supervision trees!

Expert: Oh and your release doesn't have Mix

Lifecycle


The Pragmatic Programmers
Designing Elixir Systems with OTP
Write Highly Scalable, Self-Healing Software with Layers
James Edward Gray, II
Bruce A. Tate
edited by Jacquelyn Carter

Startup

- `my_release start` or `my_release daemon` is called
- OTP "applications" are started
  - Supervision trees are started
    - Processes are started

Running

- Business logic is invoked
- Bugs surface
- Nodes connect and drop
- Remote consoles connect and drop
- And more…

Shutdown

- The inverse of startup
- This is optional!
- There could be hot code upgrades

It's all very abstract

Examples

Mapping Lifecycle to Processes

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [
      %{
        id: MyCode.StartupTask,
        start: {Task, :start_link, [&MyCode.run_at_startup/0]},
        restart: :transient  # or :temporary
      }
    ]

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

Run Some Code at Startup

Add a Task to Your Supervision Tree

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [
      %{
        id: MyCode.StartupTask,
        start: {Task, :start_link, [&MyCode.run_at_startup/0]},
        restart: :transient  # or :temporary
      }
    ]

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

Run Some Code at Startup

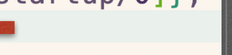
Add a Task to Your Supervision Tree

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [
      %{
        id: MyCode.StartupTask,
        start: {Task, :start_link, [&MyCode.run_at_startup/0]},
        restart: :transient  # or :temporary
      }
    ]

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

Run Some Code at Startup

Add a Task to Your Supervision Tree

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [
      %{
        id: MyCode.StartupTask,
        start: {Task, :start_link, [&MyCode.run_at_startup/0]},
        restart: :transient  # or :temporary
      }
    ]

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



Run Some Code at Startup

Add a Task to Your Supervision Tree

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [{Task.Supervisor, name: MyCode.TaskSupervisor}]
    opts = [strategy: :one_for_one, name: MyCode.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

```elixir
defmodule MyCode do
  def work_in_the_background do
    Task.Supervisor.start_child(
      MyCode.TaskSupervisor,
      fn ->
        IO.puts "Starting work…"
        Process.sleep(3_000)
        IO.puts "Done."
      end,
      restart: :transient
    )
  end
end
```

Work Jobs in the Background

Launch Tasks Under a Task.Supervisor

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [{Task.Supervisor, name: MyCode.TaskSupervisor}]
    opts = [strategy: :one_for_one, name: MyCode.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

```elixir
defmodule MyCode do
  def work_in_the_background do
    Task.Supervisor.start_child(
      MyCode.TaskSupervisor,
      fn ->
        IO.puts "Starting work…"
        Process.sleep(3_000)
        IO.puts "Done."
      end,
      restart: :transient
    )
  end
end
```

Work Jobs in the Background

Launch Tasks Under a Task.Supervisor

```elixir
defmodule MyCode.Application do
  use Application

  def start(_type, _args) do
    children = [{Task.Supervisor, name: MyCode.TaskSupervisor}]
    opts = [strategy: :one_for_one, name: MyCode.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

```elixir
defmodule MyCode do
  def work_in_the_background do
    Task.Supervisor.start_child(
      MyCode.TaskSupervisor,
      fn ->
        IO.puts "Starting work…"
        Process.sleep(3_000)
        IO.puts "Done."
      end,
      restart: :transient
    )
  end
end
```

Work Jobs in the Background

Launch Tasks Under a Task.Supervisor

```elixir
defmodule MyCode.PeriodicWorker do
  use GenServer

  def start_link(count), do: GenServer.start_link(__MODULE__, count)
  def init(count) do
    :timer.send_interval(1_000, :tick)
    {:ok, count}
  end

  def handle_info(:tick, count) do
    IO.puts "#{count}:  Doing some work."
    {:noreply, count + 1}
  end
end
```

Periodically Trigger Some Code

Send Messages Using `:timer.send_interval/3` or `Process.send_after/4`

```elixir
defmodule MyCode.PeriodicWorker do
  use GenServer

  def start_link(count), do: GenServer.start_link(__MODULE__, count)
  def init(count) do
    :timer.send_interval(1_000, :tick)
    {:ok, count}
  end

  def handle_info(:tick, count) do
    IO.puts "#{count}:  Doing some work."
    {:noreply, count + 1}
  end
end
```

Periodically Trigger Some Code

Send Messages Using `:timer.send_interval/3` or `Process.send_after/4`

```elixir
defmodule MyCode.PeriodicWorker do
  use GenServer

  def start_link(count), do: GenServer.start_link(__MODULE__, count)
  def init(count) do
    :timer.send_interval(1_000, :tick)
    {:ok, count}
  end

  def handle_info(:tick, count) do
    IO.puts "#{count}:  Doing some work."
    {:noreply, count + 1}
  end
end
```

Periodically Trigger Some Code

Send Messages Using `:timer.send_interval/3` or `Process.send_after/4`

Tools That Can Help

- poolboy
- Oban
- And more…

A Cluster of Nodes

What?

What?

Why?

- Load balancers or scaling groups
- You use Phoenix PubSub, Channels, or LiveView

How?

- Make sure both sides have the same "cookie"
- Name your nodes
- Call `Node.connect/1`
- Remember to secure the connection!

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname one
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(one@MacBook-Pro)1> :global.register_name(One.STDIN, Process.group_leader)
:yes
Hello from Node two!
iex(one@MacBook-Pro)2>
```



```
~/Desktop [e:1.9.1-otp-22 r:2.6.3p62]$ iex --cookie NOT_SAFE --sname two
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:16:16] [ds:16:16:10] [async-threads:1] [hipe]

Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)
iex(two@MacBook-Pro)1> Node.connect(:"one@MacBook-Pro")
true
iex(two@MacBook-Pro)2> node_one_stdin = :global.whereis_name(One.STDIN)
#PID<11263.70.0>
iex(two@MacBook-Pro)3> IO.puts node_one_stdin, "Hello from Node two!"
:ok
iex(two@MacBook-Pro)4>
```

```elixir
defmodule MyCode.Mailer do
  def send_just_one_email(mailing_id) do
    Ecto.Multi.new()
    |> Ecto.Multi.run(:mailing, fn repo, %{} ->
      case repo.get_by(MyCode.Schemas.Mailing, sent: false, id: mailing_id) do
        mailing when not is_nil(mailing) -> {:ok, mailing}
        nil ->                              {:error, :not_found}
      end
    end)
    |> Ecto.Multi.update(:mark_sent, fn %{mailing: mailing} ->
      Ecto.Changeset.change(mailing, sent: true)
    end)
    |> MyCode.Repo.transaction()
    |> case do
         {:ok, %{mailing: mailing}} ->
           MyCode.deliver_email(mailing)
           {:ok, mailing}

         {:error, failed_operation, failure, _changes_so_far} ->
           {:error, failed_operation, failure}
       end
  end
end
```

Invoke Some Code on Just One Node

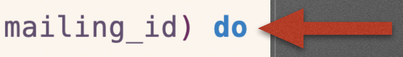
Use Your Database as a Lock ("Compare and Swap")

```elixir
defmodule MyCode.Mailer do
  def send_just_one_email(mailing_id) do
    Ecto.Multi.new()
    |> Ecto.Multi.run(:mailing, fn repo, %{} ->
      case repo.get_by(MyCode.Schemas.Mailing, sent: false, id: mailing_id) do
        mailing when not is_nil(mailing) -> {:ok, mailing}
        nil ->                              {:error, :not_found}
      end
    end)
    |> Ecto.Multi.update(:mark_sent, fn %{mailing: mailing} ->
      Ecto.Changeset.change(mailing, sent: true)
    end)
    |> MyCode.Repo.transaction()
    |> case do
         {:ok, %{mailing: mailing}} ->
           MyCode.deliver_email(mailing)
           {:ok, mailing}

         {:error, failed_operation, failure, _changes_so_far} ->
           {:error, failed_operation, failure}
       end
  end
end
```

Invoke Some Code on Just One Node

Use Your Database as a Lock ("Compare and Swap")

```elixir
defmodule MyCode.Mailer do
  def send_just_one_email(mailing_id) do
    Ecto.Multi.new()
    |> Ecto.Multi.run(:mailing, fn repo, %{} ->
      case repo.get_by(MyCode.Schemas.Mailing, sent: false, id: mailing_id) do
        mailing when not is_nil(mailing) -> {:ok, mailing}
        nil ->                              {:error, :not_found}
      end
    end)
    |> Ecto.Multi.update(:mark_sent, fn %{mailing: mailing} ->
      Ecto.Changeset.change(mailing, sent: true)
    end)
    |> MyCode.Repo.transaction()
    |> case do
         {:ok, %{mailing: mailing}} ->
           MyCode.deliver_email(mailing)
           {:ok, mailing}

         {:error, failed_operation, failure, _changes_so_far} ->
           {:error, failed_operation, failure}
       end
  end
end
```



Invoke Some Code on Just One Node

Use Your Database as a Lock ("Compare and Swap")

```elixir
defmodule MyCode.Mailer do
  def send_just_one_email(mailing_id) do
    Ecto.Multi.new()
    |> Ecto.Multi.run(:mailing, fn repo, %{} ->
      case repo.get_by(MyCode.Schemas.Mailing, sent: false, id: mailing_id) do
        mailing when not is_nil(mailing) -> {:ok, mailing}
        nil ->                              {:error, :not_found}
      end
    end)
    |> Ecto.Multi.update(:mark_sent, fn %{mailing: mailing} ->
      Ecto.Changeset.change(mailing, sent: true)
    end)
    |> MyCode.Repo.transaction()
    |> case do
         {:ok, %{mailing: mailing}} ->
           MyCode.deliver_email(mailing)
           {:ok, mailing}

         {:error, failed_operation, failure, _changes_so_far} ->
           {:error, failed_operation, failure}
       end
  end
end
```

Invoke Some Code on Just One Node

Use Your Database as a Lock ("Compare and Swap")

```elixir
defmodule MyCode.Mailer do
  def send_just_one_email(mailing_id) do
    Ecto.Multi.new()
    |> Ecto.Multi.run(:mailing, fn repo, %{} ->
      case repo.get_by(MyCode.Schemas.Mailing, sent: false, id: mailing_id) do
        mailing when not is_nil(mailing) -> {:ok, mailing}
        nil ->                              {:error, :not_found}
      end
    end)
    |> Ecto.Multi.update(:mark_sent, fn %{mailing: mailing} ->
      Ecto.Changeset.change(mailing, sent: true)
    end)
    |> MyCode.Repo.transaction()
    |> case do
         {:ok, %{mailing: mailing}} ->
           MyCode.deliver_email(mailing)
           {:ok, mailing}

         {:error, failed_operation, failure, _changes_so_far} ->
           {:error, failed_operation, failure}
       end
  end
end
```

Invoke Some Code on Just One Node

Use Your Database as a Lock ("Compare and Swap")

Tools That Can Help

- libcluster
- Swarm
- Horde
- And more…

Exercise: Keep Just One Process Running in a Cluster

- "Name Registration" (specifically "global") in GenServer
- What does trying to start a duplicate "global" process do?
- "Return Values" (specifically "ignore") under `GenServer.start_link/3`
- `:global.register_name/3`
- `:net_kernel.monitor_nodes/1`