SOFTWARE FOR THE INTERNET

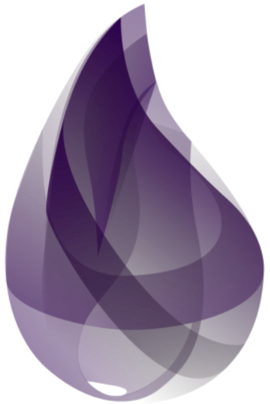
WITH PHOENIX

KIRA MCLEAN

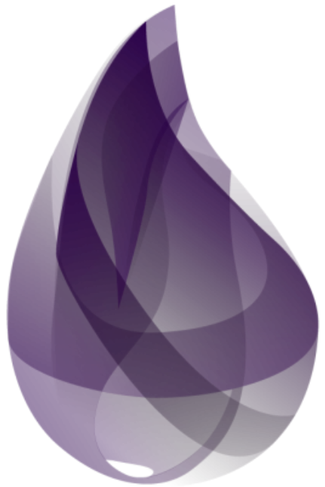
TWITTER.COM/KIRAEMCLEAN
GITHUB.COM/KIRAMCLEAN

ABOUT ME

ABOUT ME

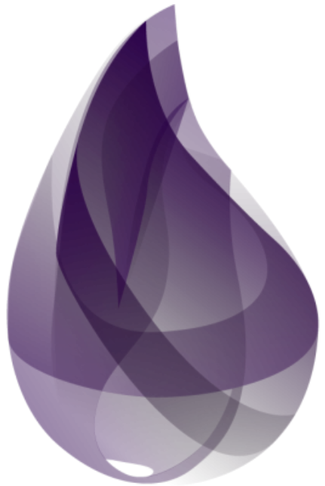


ELIXIR → LANGUAGE



ELIXIR → LANGUAGE

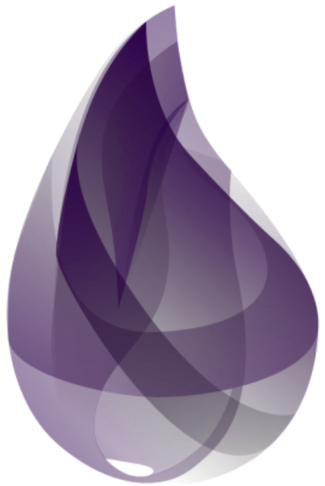
ERLANG → ALSO LANGUAGE. HAS OWN VM ("THE BEAM")



ELIXIR → LANGUAGE

ERLANG → ALSO LANGUAGE. HAS OWN VM ("THE BEAM")

PHOENIX → WEB FRAMEWORK



SCALING WEB APPS IS HARD

WEB APPS

- 1. DISTRIBUTED

WEB APPS

- 1. DISTRIBUTED
- 2. STATEFUL

WEB APPS

- 1. DISTRIBUTED
- 2. STATEFUL

FUNDAMENTAL PROBLEM:

STATEFUL APPS COMMUNICATING OVER A STATELESS PROTOCOL

TO DEAL WITH IT

- 1. CONCURRENCY

TO DEAL WITH IT

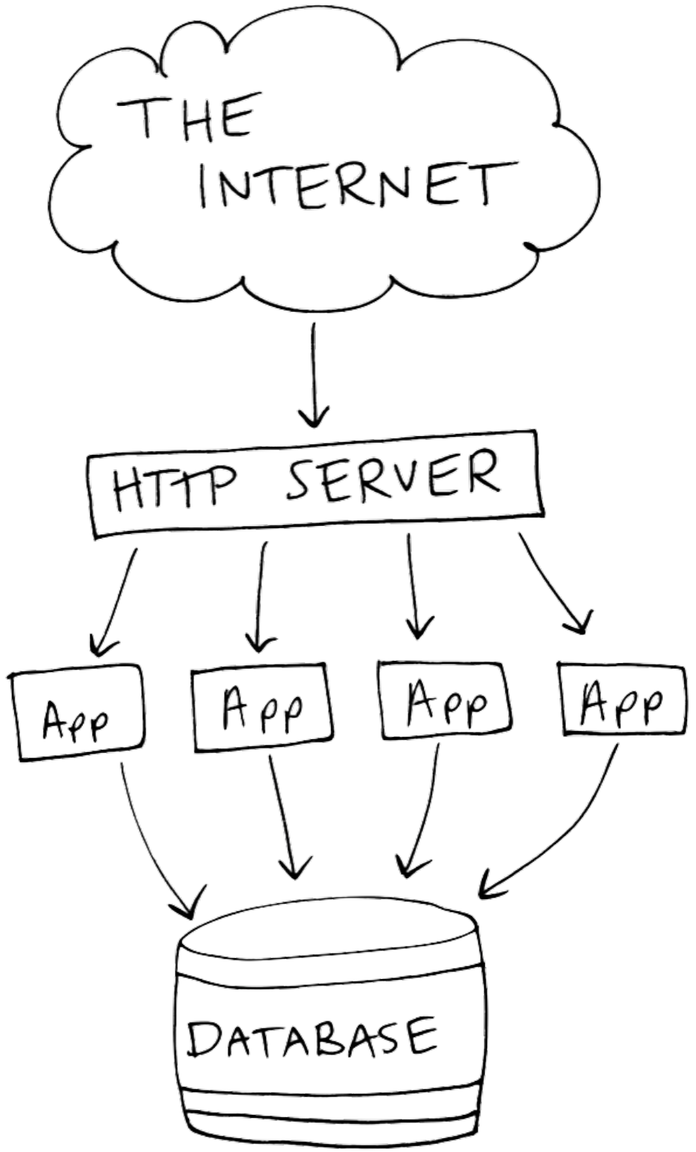
- 1. CONCURRENCY
- 2. PERSISTENCE

CURRENT SOLUTION

- 1. CONCURRENCY → OPERATING SYSTEM
- 2. PERSISTENCE

CURRENT SOLUTION

- 1. CONCURRENCY → OPERATING SYSTEM
- 2. PERSISTENCE → DATABASE


THE INTERNET
HTTP SERVER
App
App
App
App
DATABASE

SCALING THIS SOLUTION

- MORE PROCESSORS

SCALING THIS SOLUTION

- MORE PROCESSORS
- MORE MACHINES

SCALING THIS SOLUTION

- MORE PROCESSORS
- MORE MACHINES

RECAP

RECAP

WEB APPS ARE:

RECAP

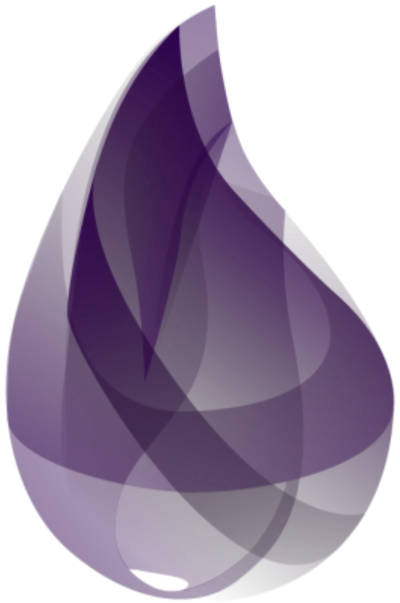
WEB APPS ARE:

- DISTRIBUTED
- HIGHLY CONCURRENT
- STATEFUL

RECAP

WEB APPS ARE:

- DISTRIBUTED
- HIGHLY CONCURRENT
- STATEFUL




ERLANG

ELIXIR'S CONCURRENCY MODEL

ELIXIR'S CONCURRENCY MODEL

ACTOR MODEL

ACTOR MODEL

- SUPER LIGHTWEIGHT VM PROCESSES

```
iex> spawn(fn -> IO.puts("Hello from process") end)
Hello from process
#PID<0.82.0>
```

ACTOR MODEL

- EACH PROCESS HAS IT'S OWN MAIL BOX

```
iex> send(self(), {:hello, "A message for my mailbox"})
{:hello, "A message for my mailbox"}

iex> receive do
...>   {:hello, message} -> message
...> end
"A message for my mailbox"
```

ACTOR MODEL

- INTERACT WITH EACH OTHER ONLY BY SENDING MESSAGES TO EACH OTHERS' MAIL BOXES

```
iex> parent = self()
#PID<0.83.0>

iex> spawn(fn -> send(parent, {:hello, self(), "a message for you"}) end)
#PID<0.108.0>

iex> receive do
...>   {:hello, pid, message} -> "Got message: '#{message}' from #{inspect(pid)}"
...> end
"Got message: 'a message for you' from #PID<0.108.0>"
```

ACTOR MODEL

- MESSAGES ARE SENT ASYNCHRONOUSLY BUT READ SEQUENTIALLY

```
iex> parent = self()
#PID<0.83.0>

iex> numbers = Enum.to_list(1..1000)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, ...]

iex> numbers |>
...>   Enum.map(&(spawn(fn -> send(recipient, {:hello, "this is message ##{&1}"}) end))
[#PID<0.349.0>, #PID<0.350.0>, #PID<0.351.0>, #PID<0.352.0>, #PID<0.353.0>, ...]
```

ACTOR MODEL

- **MESSAGES ARE** SENT ASYNCHRONOUSLY BUT **READ SEQUENTIALLY**

```
iex> flush()
{:hello, "this is message #1"}
{:hello, "this is message #2"}
{:hello, "this is message #3"}
# ...
{:hello, "this is message #8"}
{:hello, "this is message #10"}
{:hello, "this is message #9"}
# ...
{:hello, "this is message #20"}
{:hello, "this is message #22"}
{:hello, "this is message #21"}
{:hello, "this is message #23"}
{:hello, "this is message #25"}
{:hello, "this is message #24"}
{:hello, "this is message #26"}
{:hello, "this is message #27"}
{:hello, "this is message #29"}
{:hello, "this is message #28"}
#...
```

ACTOR MODEL

> PROCESSES RUN CONCURRENTLY

```
defmodule Slow do
  def concurrent(delays) do
    recipient = self()
    delays
    |> Enum.map(&(spawn(fn -> send_message(&1, recipient) end)))
    |> Enum.map(fn pid -> receive_message(pid) end)
  end

  defp send_message(delay, back_to_caller) do
    :timer.sleep(delay)
    send(back_to_caller, {self(), "Waited #{delay}ms"})
  end

  defp receive_message(pid) do
    receive do
      {^pid, message} -> {pid, message}
    end
  end
end
```

ACTOR MODEL

> PROCESSES RUN CONCURRENTLY

```
delays = for _ <- 1..1000 do [] ++ Enum.random(100..2500) end

defmodule Timing do
  def time_this(delays) do
    before = System.monotonic_time(:millisecond)
    Slow.concurrent(delays)
    later = System.monotonic_time(:millisecond)
    (later - before) / 1000
  end
end

iex> Timing.time_this(delays)
2.503
```

ACTOR MODEL

- MEMORY IS ISOLATED TO EACH PROCESS

```
iex> me = self()
#PID<0.83.0>

iex> new_process = spawn(fn -> raise "Uh oh" end)
#PID<0.120.0>

iex>
15:24:37.039 [error] Process #PID<0.120.0> raised an exception
** (RuntimeError) Uh oh
    (stdlib) erl_eval.erl:668: :erl_eval.do_apply/6

iex> Process.alive?(new_process)
false

iex> Process.alive?(parent)
true
```

MORE THAN PROCESSES

- STATE?
- LINKING?

MORE THAN PROCESSES

- STATE? → Agent
  - LINKING?

MORE THAN PROCESSES

- STATE? → `Agent`
- LINKING? → `Task`

MORE THAN PROCESSES

- STATE? → `Agent`
- LINKING? → `Task`
  - BOTH?

MORE THAN PROCESSES

- STATE? → `Agent`
- LINKING? → `Task`
- BOTH? → `GenServer`

MORE THAN PROCESSES

- STATE? → `Agent`
- LINKING? → `Task`
- BOTH? → `GenServer`
  - MONITORING?

MORE THAN PROCESSES

- STATE? → `Agent`
- LINKING? → `Task`
- BOTH? → `GenServer`
- MONITORING? → `Supervisor`

MORE THAN PROCESSES

- STATE? → `Agent`
- LINKING? → `Task`
- BOTH? → `GenServer`
- MONITORING? → `Supervisor`

> OTP

AN ALTERNATIVE APPROACH FOR WEB DEVELOPERS

ALTERNATIVE SOLUTION

- 1. CONCURRENCY
- 2. PERSISTENCE

ALTERNATIVE SOLUTION

- 1. CONCURRENCY → ERLANG VM
- 2. PERSISTENCE

ALTERNATIVE SOLUTION

- 1. CONCURRENCY → ERLANG VM
  - 2. PERSISTENCE → THE OTP

COMMON CHALLENGES

1. SLOW REQUESTS
2. CACHING
3. SHARING DATA
4. ERROR HANDLING

1. SLOW REQUESTS

1. SLOW REQUESTS

PROBLEM: DOING TOO MUCH WORK

1. SLOW REQUESTS

- BEFORE: **BACKGROUND JOBS**

1. SLOW REQUESTS

› BEFORE: BACKGROUND JOBS

```
# in an imports controller somwhere

def create
  import_options = parse_some_incoming_params
  # send to a separate worker process to deal with
  SlowImportJob.perform(import_options)
  send_the_user_somewhere
end
```

1. SLOW REQUESTS

> BEFORE: BACKGROUND JOBS

```
# in an imports controller somwhere

def create
  import_options = parse_some_incoming_params
  # send to a separate worker process to deal with
  SlowImportJob.perform(import_options)
  send_the_user_somewhere
end
```

1. SLOW REQUESTS

- BEFORE: **BACKGROUND JOBS**
- AFTER: **ELIXIR PROCESSES**

1. SLOW REQUESTS

- BEFORE: **BACKGROUND JOBS**
- AFTER: **ELIXIR PROCESSES**

```
def create(conn, %{"import" => import_params}) do
  Task.Supervisor.async_nolink(MyApp.TaskSupervisor, fn ->
    BulkImport.create(import_params)
  )
  send_user_somewhere
end
```

1. SLOW REQUESTS

- BEFORE: **BACKGROUND JOBS**
- AFTER: **ELIXIR PROCESSES**

```
def create(conn, %{"import" => import_params}) do
  Task.Supervisor.async_nolink(MyApp.TaskSupervisor, fn ->
    BulkImport.create(import_params)
  )
  send_user_somewhere
end
```

1. SLOW REQUESTS

PROBLEM: WAITING ON TOO MANY THINGS

1. SLOW REQUESTS

```
# in a checkouts controller

def new
  @tax_rate = fetch_tax_rate
  @shipping_rate = fetch_fedex_shipping_rate
  @other_shipping_rate = fetch_local_carrier_shipping_rate
  run_through_fraud_detection
  reserve_items_in_cart
  render :new
end
```

1. SLOW REQUESTS

- BEFORE: **ASYNC JAVASCRIPT**

1. SLOW REQUESTS

› BEFORE: ASYNC JAVASCRIPT

```
xmlRequest.onreadystatechange = function() {
  if (xmlRequest.readyState === XMLHttpRequest.DONE) {
    if (xmlRequest.status === 200) { result = xmlRequest.responseText }
    else if (xmlRequest.status === 400) { alert("Something went wrong") }
    else if (...) { alert("Something bad happened") }
  }
}

xmlRequest.onloadend = function() {
  document.querySelector(".my-dom-element").innerHTML = xmlRequest.responseText
}

xmlRequest.open("GET", "http://someapi.com/endpoint", true) // true means async
xmlRequest.send()
```

1. SLOW REQUESTS

- BEFORE: **ASYNC JAVASCRIPT**
- AFTER: **ELIXIR PROCESSES**

```elixir
def new(conn, %{"some_param" => some_param}) do
  tax_rate = Task.async(fn -> TaxService.find_rate(some_param) end)
  shipping_rate = Task.async(fn -> ShippingService.get_rate(some_param) end)
  other_rate = Task.async(fn -> OtherService.get_rate(some_param) end)

  # things the UI doesn't care about:
  Task.Supervisor.async_nolink MyApp.TaskSupervisor, fn ->
    FraudDetection.log_this_cart(some_param) end)

  Task.Supervisor.async_nolink MyApp.TaskSupervisor, fn ->
    InventoryManagement.reserve_these_things(some_param) end)

  render(conn, "show.html", tax_rate: Task.await(tax_rate),
                            shipping_rate: Task.await(shipping_rate),
                            other_rate: Task.await(other_rate))
end
```

```elixir
def new(conn, %{"some_param" => some_param}) do
  tax_rate = Task.async(fn -> TaxService.find_rate(some_param) end)
  shipping_rate = Task.async(fn -> ShippingService.get_rate(some_param) end)
  other_rate = Task.async(fn -> OtherService.get_rate(some_param) end)

  # things the UI doesn't care about:
  Task.Supervisor.async_nolink MyApp.TaskSupervisor, fn ->
    FraudDetection.log_this_cart(some_param) end)

  Task.Supervisor.async_nolink MyApp.TaskSupervisor, fn ->
    InventoryManagement.reserve_these_things(some_param) end)

  render(conn, "show.html", tax_rate: Task.await(tax_rate),
                            shipping_rate: Task.await(shipping_rate),
                            other_rate: Task.await(other_rate))
end
```

1. SLOW REQUESTS

PROBLEM: RENDERING A LOT OF THINGS

1. SLOW REQUESTS

- BEFORE: MANY LEVELS OF CACHING

1. SLOW REQUESTS

› BEFORE: MANY LEVELS OF CACHING

```
<section class="products">
  <!-- for every product in a collection -->
    <product-description-partial>
    <product-images-carousel>
    <product-special-info-banner-partial>
    <product-pricing-partial>
</section>
```

1. SLOW REQUESTS

- BEFORE: **MANY LEVELS OF CACHING**
- AFTER: **PERFORMANCE IS NOT A CONCERN**

1. SLOW REQUESTS

- BEFORE: **MANY LEVELS OF CACHING**
- AFTER: **PERFORMANCE IS NOT A CONCERN**

2. CACHING

2. CACHING

- BEFORE: **MEMCACHED**

2. CACHING

- BEFORE: MEMCACHED

- SHARED, HIGH-PERFORMACE STORAGE
  - OWN SERVER

2. CACHING

- BEFORE: **MEMCACHED**
- AFTER: **ERLANG TERM STORAGE**

2. CACHING

- BEFORE: **MEMCACHED**
- AFTER: **ERLANG TERM STORAGE**

```
iex(1)> :ets.new(:fancy_erlang_cache, [:set, :private, :named_table])
:fancy_erlang_cache

iex(2)> :ets.insert(:fancy_erlang_cache, {"cache key", "some value"})
true

iex(3)> :ets.lookup(:fancy_erlang_cache, "cache key")
[{"cache key", "some value"}]
```

3. SHARING DATA

3. SHARING DATA

PROBLEM: THINGS ARE MUTABLE

3. SHARING DATA

- BEFORE: JUST.. BE CAREFUL

3. SHARING DATA

› BEFORE: JUST.. BE CAREFUL

```
class ShippingRates
  def initialize(items, destination)
    @items = items
    @destination = destination
  end

  def fetch_rates
    CarrierAPIWrapper.find_rates(@items, @destination)
  end
end
```

3. SHARING DATA

> BEFORE: JUST.. BE CAREFUL

```
class ShippingRates
  def initialize(items, destination)
    @items = items # an array of things; arrays are mutable
    @destination = destination
  end

  def fetch_rates
    # now this class thinks it owns @items
    CarrierAPIWrapper.find_rates(@items, @destination)
  end
end
```

3. SHARING DATA

› BEFORE: JUST.. BE CAREFUL

```
class ShoppingCart
  def initialize(products)
    @products = products
  end

  def checkout
    # ... do checkout

    stock_items = @products.map(&:stock_item)
    stock_items.update_all(sold: true)
  end
end
```

3. SHARING DATA

› BEFORE: JUST.. BE CAREFUL

```
class ShoppingCart
  def initialize(products)
    @products = products
  end

  def checkout
    # ... do checkout

    # Wrong place -- you own the array of products, not the products
    # themselves, and definitely not the children of the products
    stock_items = @products.map(&:stock_item)
    stock_items.update_all(sold: true)
  end
end
```

3. SHARING DATA

- BEFORE: MANUAL LOCKING AROUND UN-THREADSAFE THINGS

3. SHARING DATA

› BEFORE: MANUAL LOCKING AROUND UN-THREADSAFE THINGS

```
@variable ||= initialize_variable

↓

lock = Mutex.new

lock.synchronize do
  @variable ||= initialize_variable
end
```

3. SHARING DATA

› BEFORE: MANUAL LOCKING AROUND UN-THREADSAFE THINGS

```
@counter += 1

↓

lock = Mutex.new

lock.synchronize do
  @counter += 1
end
```

3. SHARING DATA

- BEFORE: MANUAL LOCKING AROUND UN-THREADSAFE THINGS
  - AFTER: MESSAGE PASSING ONLY
- CONTINUE TO PIGGY-BACK ON DB FOR TRANSACTIONAL MEMORY FOR PERSISTED DATA

ISOLATED MEMORY + IMMUTABLE DATA =

4. ERROR HANDLING

4. ERROR HANDLING

PROBLEM: RECOVERING FROM UNEXPECTED ERRORS

4. ERROR HANDLING

- BEFORE: CATCH WHAT YOU CAN

4. ERROR HANDLING

› BEFORE: CATCH WHAT YOU CAN

```
def upload_file_somewhere
  # upload...
  rescue # timeout
  rescue # HTTP error from S3
  rescue # other "expected" exception
end
```

4. ERROR HANDLING

- BEFORE: **CATCH WHAT YOU CAN**
  - AFTER: **SUPERVISION TREES**

4. ERROR HANDLING

- BEFORE: CATCH WHAT YOU CAN
  - AFTER: SUPERVISION TREES

```
# in the supervisor
use Supervisor

def start_link do
  Supervisor.start_link(__MODULE__, [])
end

# in the supervised server
use GenServer

def start_link(rates \\ []) do
  GenServer.start_link(__MODULE__, [], name: __MODULE__)
end
```

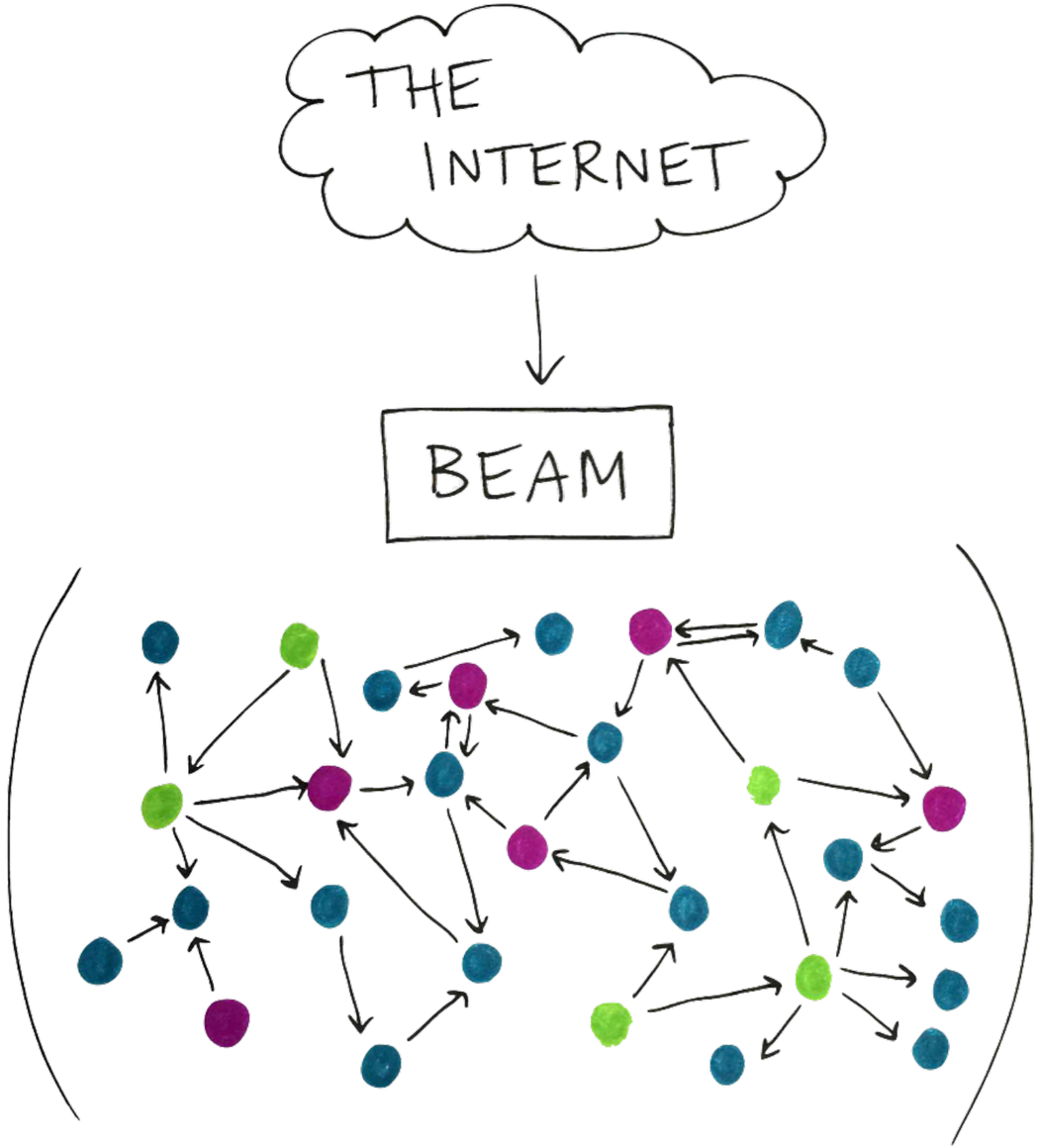
BONUSES WITH PHOENIX

- VERY FAST (FOR THE WEB)
- GREAT DEV TOOLS
- ENCOURAGES SEPARATION OF APP AND INTERFACE (UMBRELLA APPS)
- SCALABLE

DOWNSIDES

- CHANGE THE WAY YOU THINK
- PROCESSING SPEED
- FEWER EXTERNAL SERVICES/LIBRARIES

AN ALTERNATIVE APPROACH FOR WEB DEVELOPERS


THE INTERNET
BEAM
Supervisor Process
Stateful Process
Worker Process

CONSIDER PHOENIX FOR

- HIGH THROUGHPUT
- MINIMAL DOWNTIME
- REALTIME
- STATEFUL SERVERS
- ???

THANKS!

REQUIREMENT	RUBY	ELIXIR
BACKGROUND JOBS	SIDEKIQ/RESQUE	BEAM
SERVER-WIDE STATE	REDIS	BEAM
LOW-LEVEL CACHING	MEMCACHED	ETS (BEAM)
SCHEDULED JOBS	CRON	BEAM
APP SERVER	PUMA/UNICORN	BEAM
HTTP SERVER	NGINX	BEAM
LONG-RUNNING REQUESTS	ACTIONCABLE (RUBY)	PHOENIX CHANNELS (BEAM)
CRASH RECOVERY	MONIT/GOD/FOREMAN	BEAM

FOR MORE

SLIDES: **GITHUB**

GOOD BOOKS: **PROGRAMMING PHOENIX, PROGRAMMING ELIXIR, ELIXIR IN ACTION**

MEETUP: **MONTREAL ELIXIR**

HELPFUL WEBSITES: **ELIXIR SCHOOL, EXERCISM**