Concurrent Programming with the Elixir Ecosystem

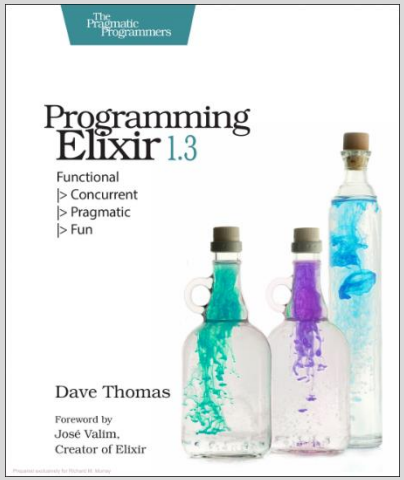
Dick Murray

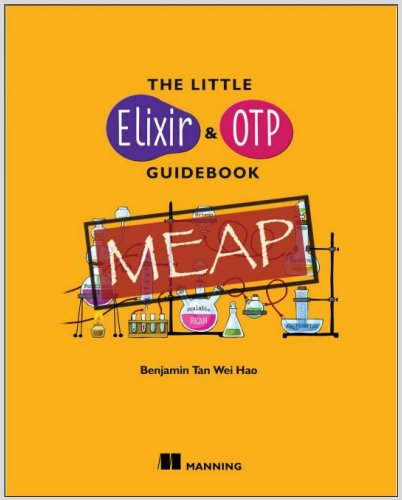
8/31/2016

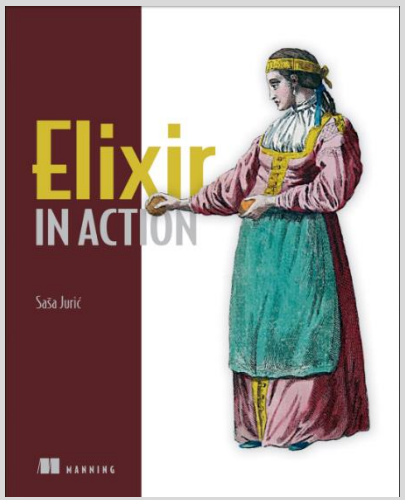
Advertisement

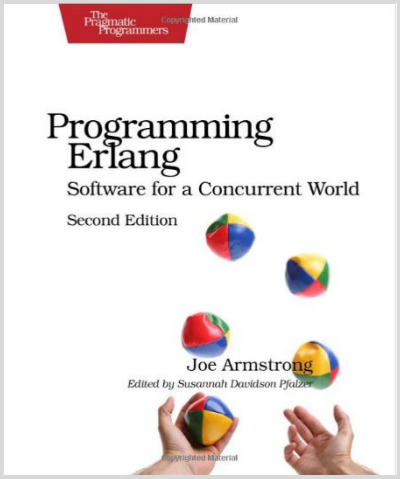
- This talk will explore concurrent programming through examples using Elixir and the Open Telecom Platform. Functional programming has shown us the value of avoiding mutable state, and composing computing elements into pipelines that transform data. Add concurrent programming, and the result is an ecosystem in which our designs can flourish in a multi-core environment. We will examine several fundamental features including process supervision and the "Let It Crash" systems philosophy.

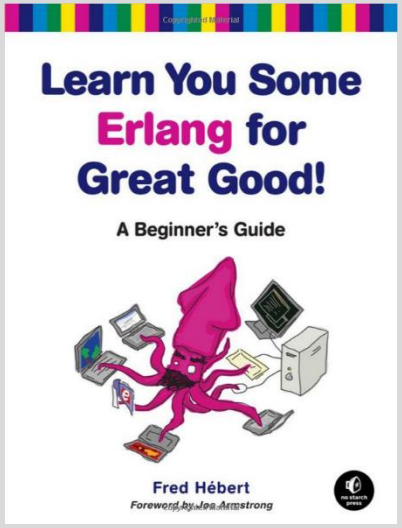
This presentation uses material from these books











Why Elixir… What about Scala

- Not a competition. Scala remains important.
  - See: http://vschart.com/compare/elixir/vs/scala
- Elixir is different: Functions/Concurrent, not *obviously* OO. Elixir is Rubyish, Scala is Javaish
- Some patterns disappear, some are new
- Elixir is Erlang Plus: Erlang is historically concurrent
- Lightweight processes built into BEAM
- Ecosystems for Scala and Elixir/Erlang

Concept of Concurrency

- Is it possible to break a program/algorithm into order independent components?
- If so these components are said to be concurrent.
- If these concurrent components are executed out of order the result is the same.
- This allows parallel execution of the components in multi-processor and multi-core systems

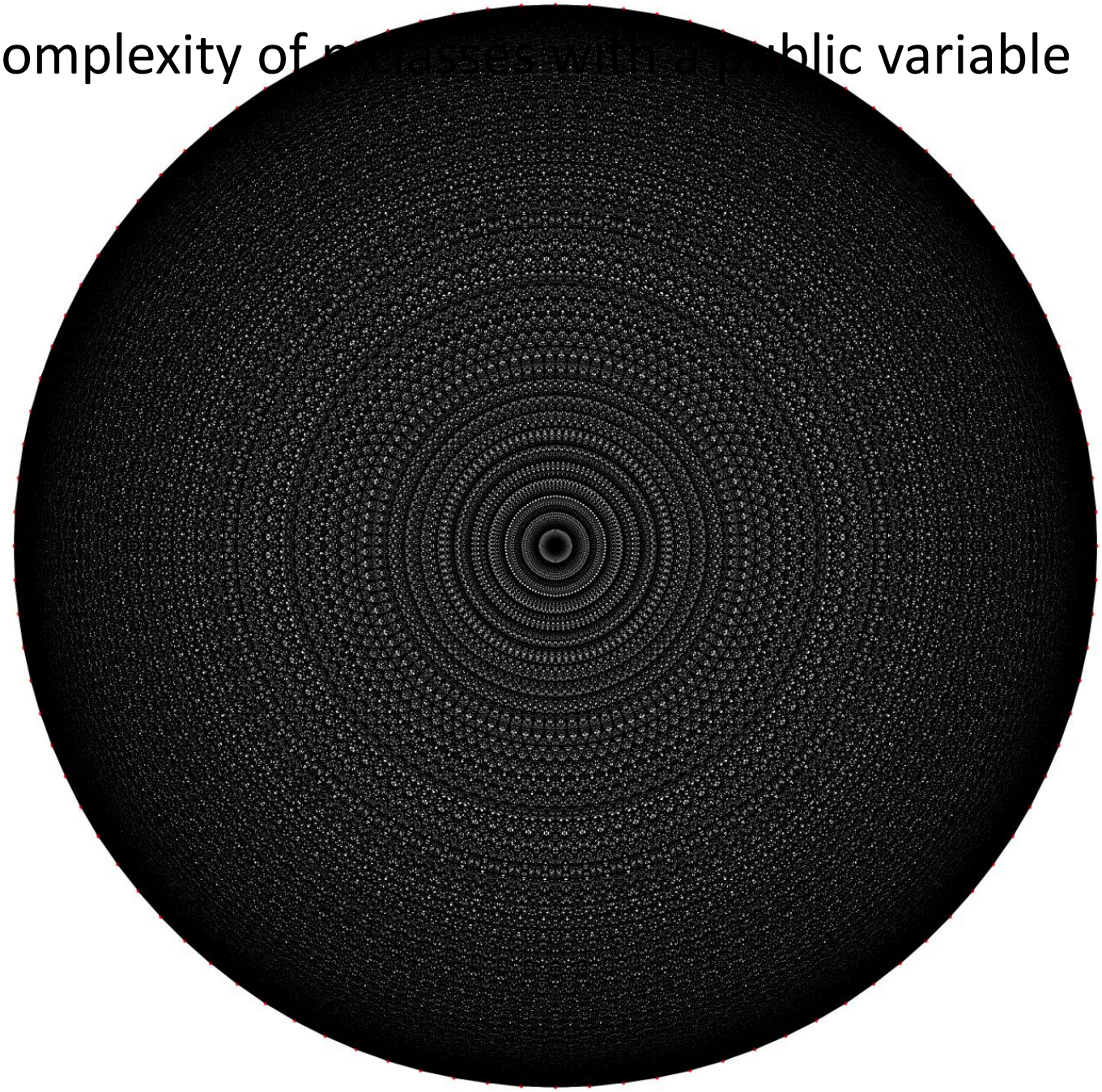
Actor Model

- Actors are processes that act as the universal primitives of concurrent computation.
- Actors may;
  - Send a message to an actor
  - Receive a message from an actor
  - Create other actors (spawn process)
  - Maintain internal state (i.e share nothing with other actors)
- Elixir/Erlang the actors are processes.
  - Convention refers to them as processes.

Why Concurrency, Why Actors

- Coupling is a measure of how one part of a program can effect another part.
- In Java, a public class variable is an example of high coupling between the class and every other class.
- In this case, coupling may be mitigated by;
  - Private scope
  - Getters/setters
  - Variable set only by constructor

Complexity of n classes with a public variable



Reducing Complexity

- Functional Programming Style
  - Immutability
  - Functions as first class components
- Concurrency
  - Independent processes
  - Explicit and observable process interaction

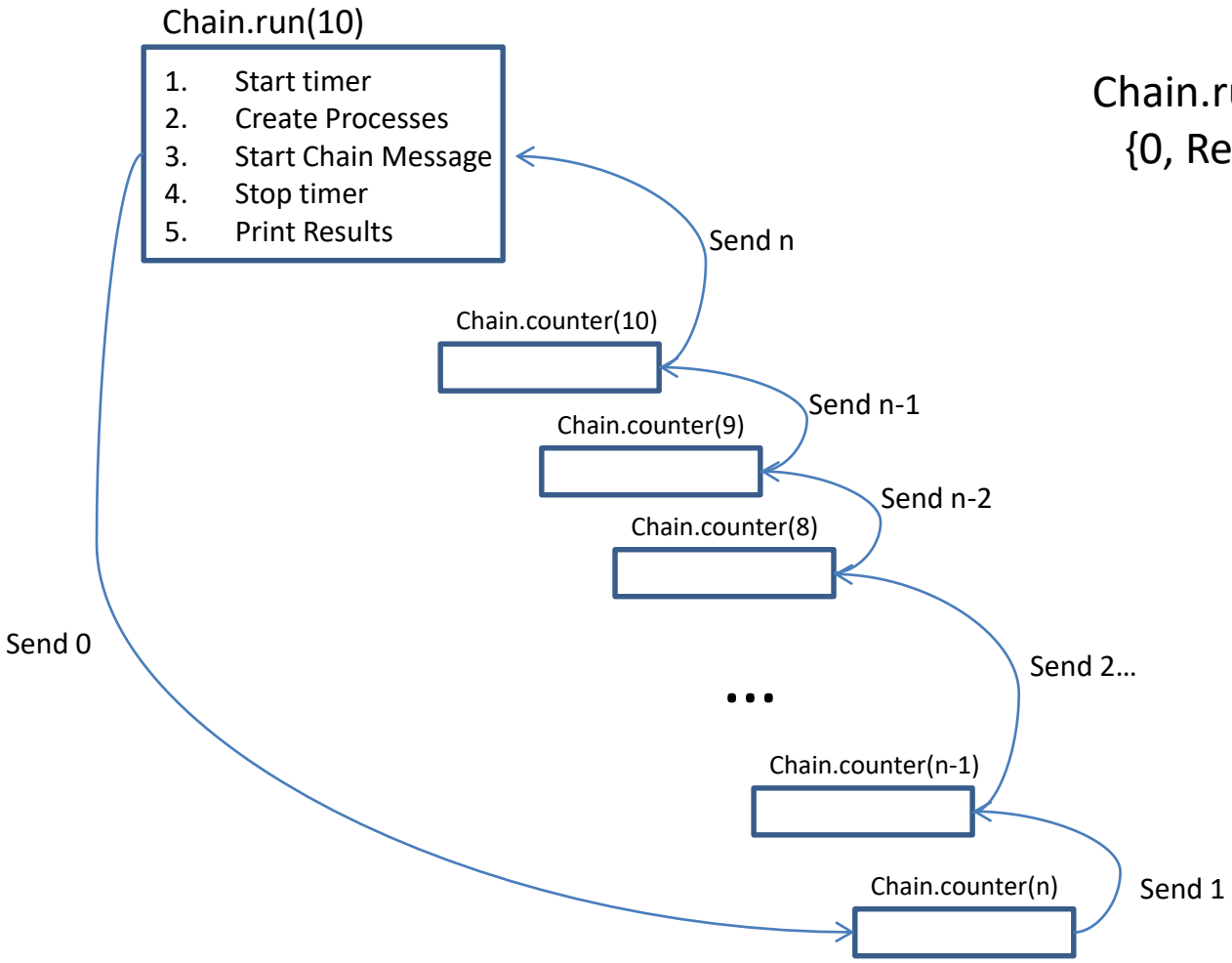
Elixir Ecosystem

- Elixir
  - Elixirc: Elixir compiler
  - IEx: interactive Elixir (REPL)
  - MIX: Elixir project manager
  - EXUNIT: test framework
  - Phoenix web framework
- Erlang
  - Erlang language and libraries
  - BEAM: virtual machine with built-in concurrency
  - Observer
  - Release Manager

Simple Concurrent Example

- Create a process that adds one to a counter and passes it to the next process in line
- Spawn n of these processes.
- Kickoff the “chain reaction” and see how long it takes

Chain Module Diagram

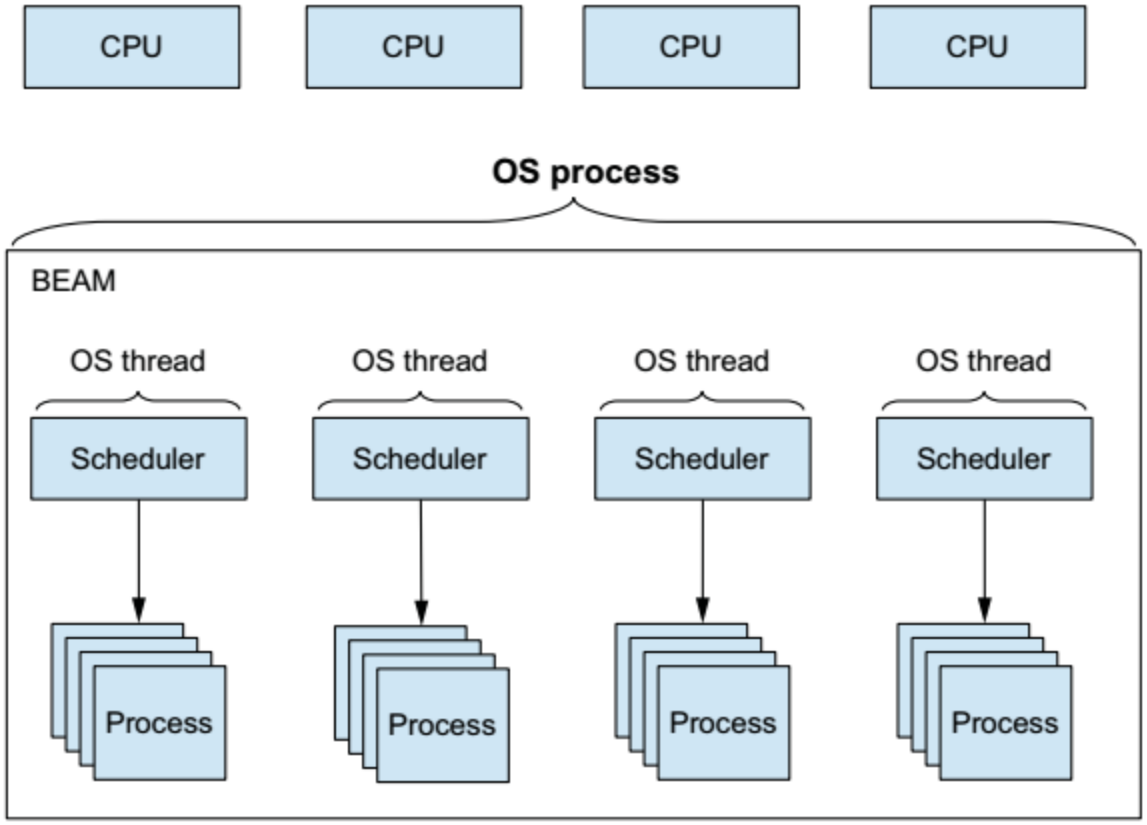


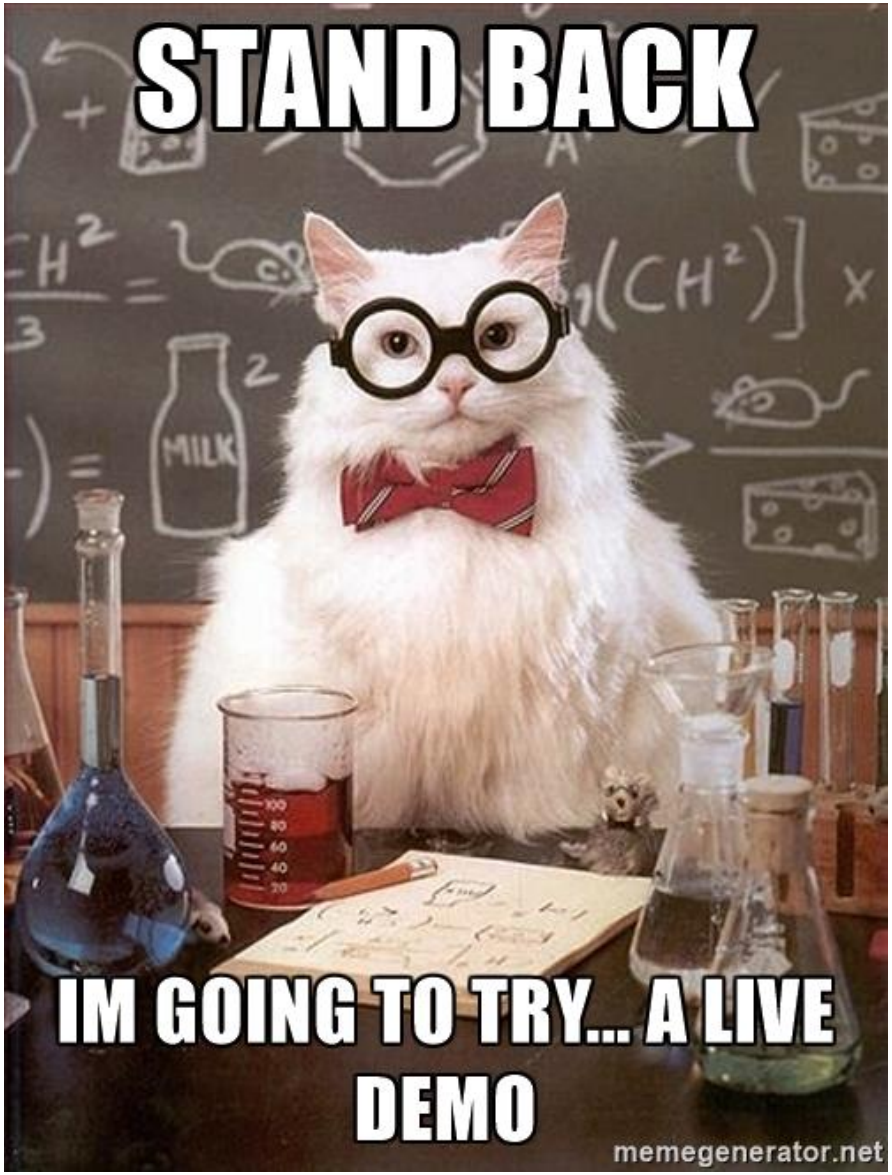
Chain.run(10) prints
{0, Result is 10}

spawn/chain.ex

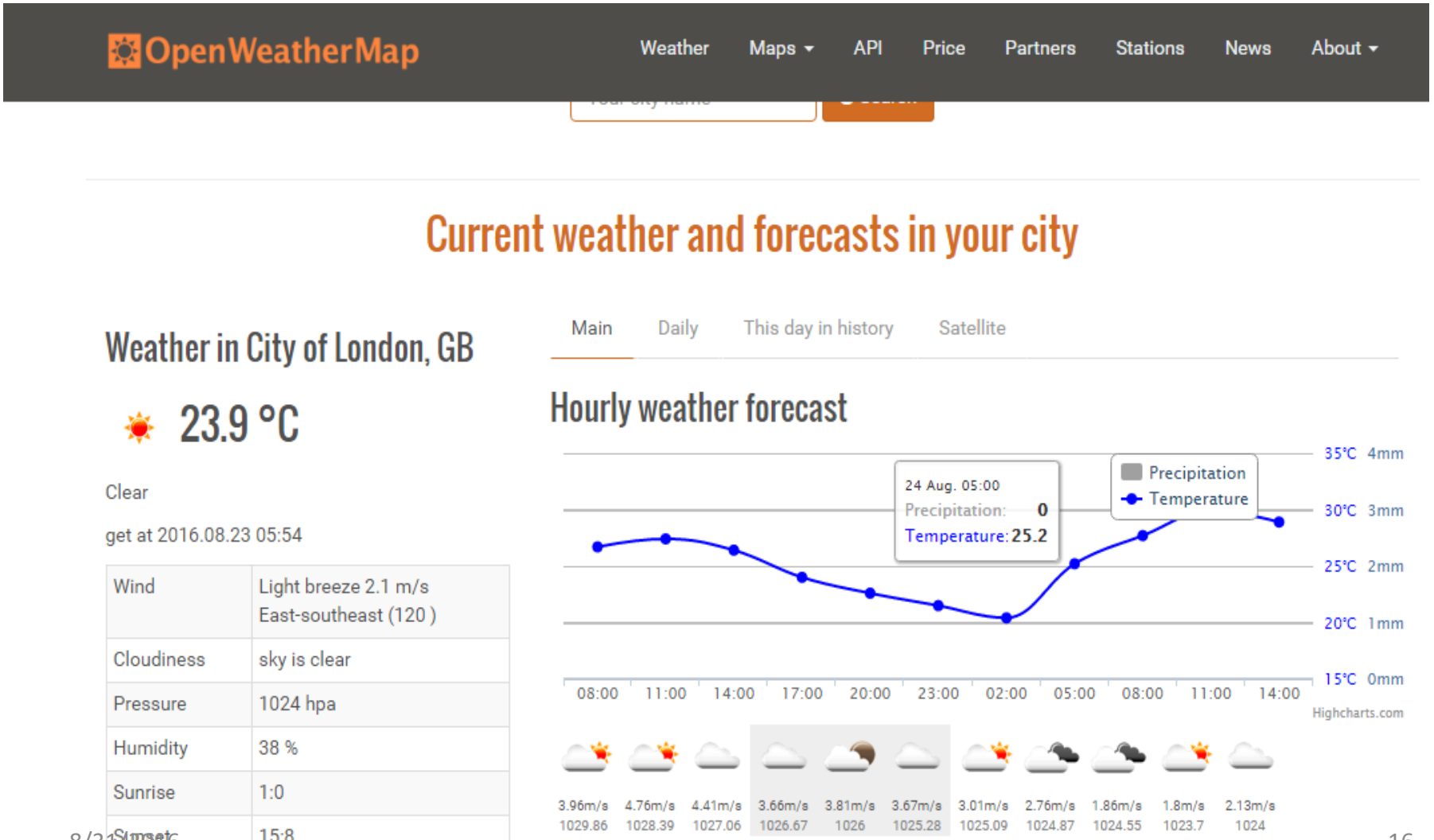
```
Line 1  defmodule Chain do
        def counter(next_pid) do
          receive do
            n ->
              send next_pid, n + 1
          end
        end

        def create_processes(n) do
          last = Enum.reduce 1..n, self,
                   fn (_,send_to) ->
                     spawn(Chain, :counter, [send_to])
                   end

          send last, 0      # start the count by sending a zero to the last process

          receive do        # and wait for the result to come back to us
            final_answer when is_integer(final_answer) ->
              "Result is #{inspect(final_answer)}"
          end
        end

        def run(n) do
          IO.puts inspect :timer.tc(Chain, :create_processes, [n])
        end
      end
```

BEAM Process Structure




STAND BACK
MILK
IM GOING TO TRY... A LIVE DEMO
memegenerator.net

Concurrency: A Richer Example



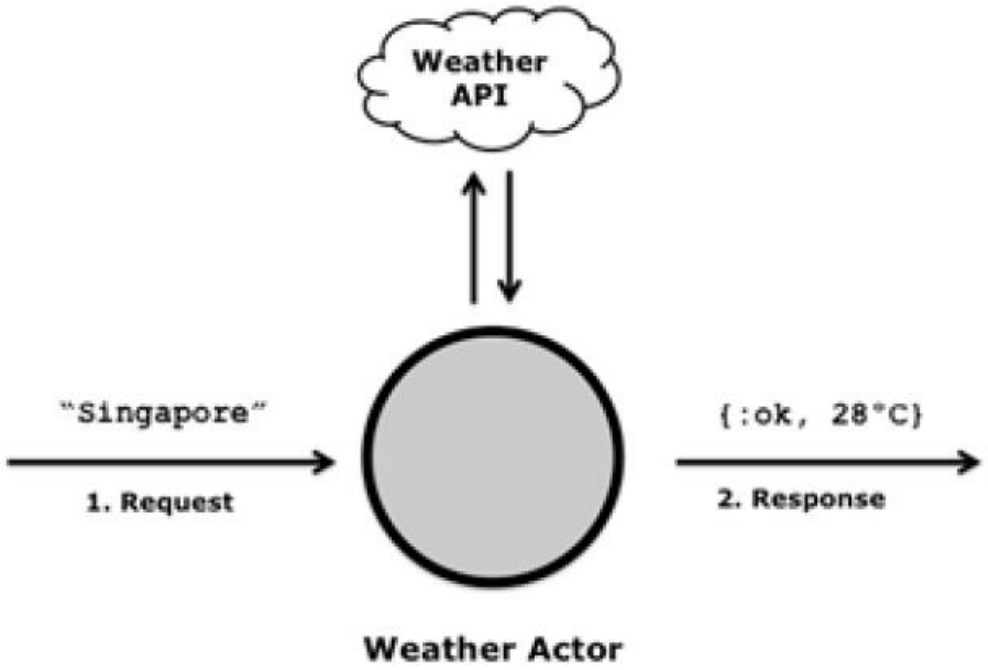
Weather Site has API

http://api.openweathermap.org/data/2.5/weather?q=Singapore&APPID=969038ec2b87be02c5f1e3f1344ee286

Returned JSON snippet


```
- main: {
      temp: 304.38,
      humidity: 59,
      pressure: 1004,
      temp_min: 303.15,
      temp_max: 305.93
  },
- wind: {
      speed: 3.08,
      gust: 0,
      deg: 310
  },
- clouds: {
      all: 8
  },
  dt: 1471946413,
  id: 1880252,
  name: "Singapore",
  cod: 200
```

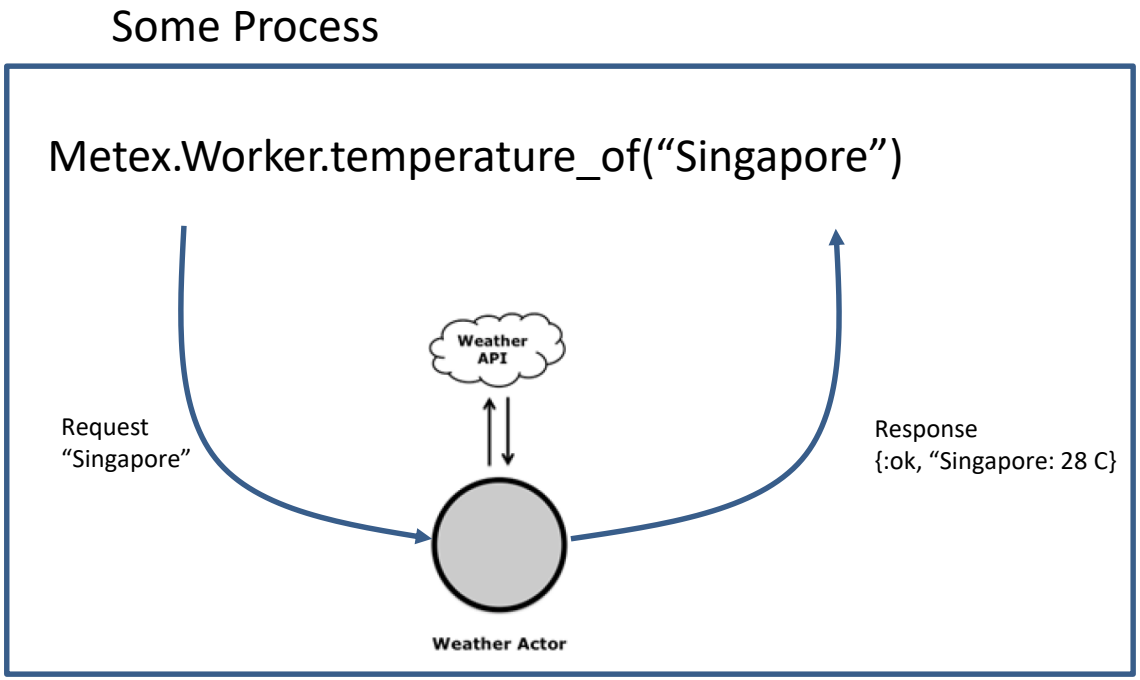
Write Elixir Code to Get Weather



```elixir
  defp temperature_of(location) do
    result = url_for(location) |> HTTPoison.get |> parse_response
    case result do
      {:ok, temp} ->
        "#{location}: #{temp}°C"
      :error ->
        "#{location} not found"
    end
  end

  defp url_for(location) do
    location = URI.encode(location)
    "http://api.openweathermap.org/data/2.5/weather?q=#{location}&appid=#{apikey}"
  end

  defp parse_response({:ok, %HTTPoison.Response{body: body, status_code: 200}}) do
    body |> JSON.decode! |> compute_temperature
  end

  defp parse_response(_) do
    :error
  end

  defp compute_temperature(json) do
    try do
      temp = (json["main"]["temp"] - 273.15) |> Float.round(1)
      {:ok, temp}
    rescue
      _ -> :error
    end
  end

  defp apikey do
    "969038ec2b87be02c5f1e3f1344ee286"
  end

end
```

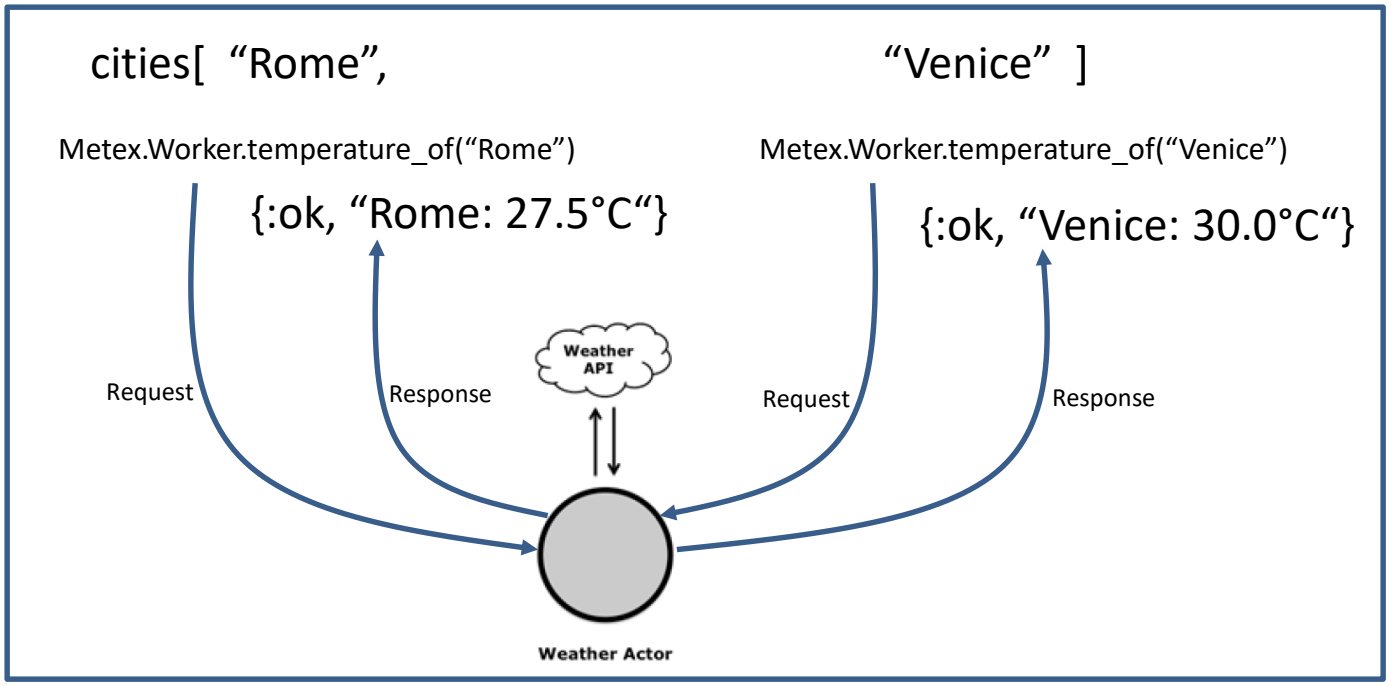
Demo: Worker Task

- Metex.Worker.temperature_of(“Singapore”)
  - Get a single weather forecast



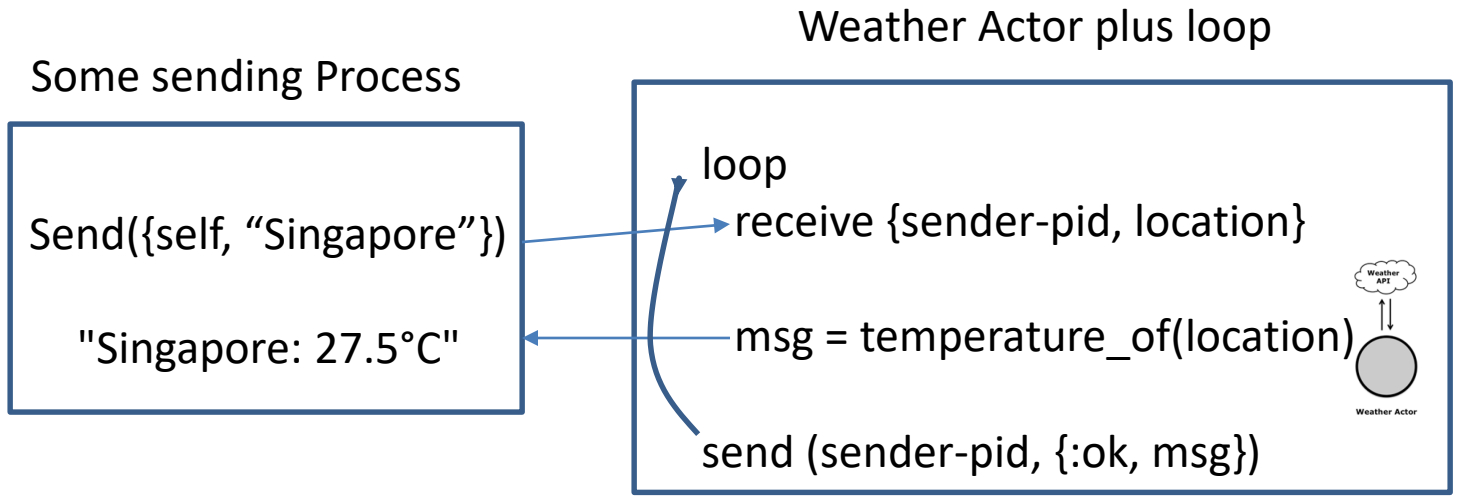
Use map to submit multiple cities

- cities = [“Rome",“Venice"]
- cities |> Enum.map((fn(city) -> Metex.Worker.temperature_of(city) end))
  - Get multiple weather forecasts one at a time from a function



Make a Server process (actor)

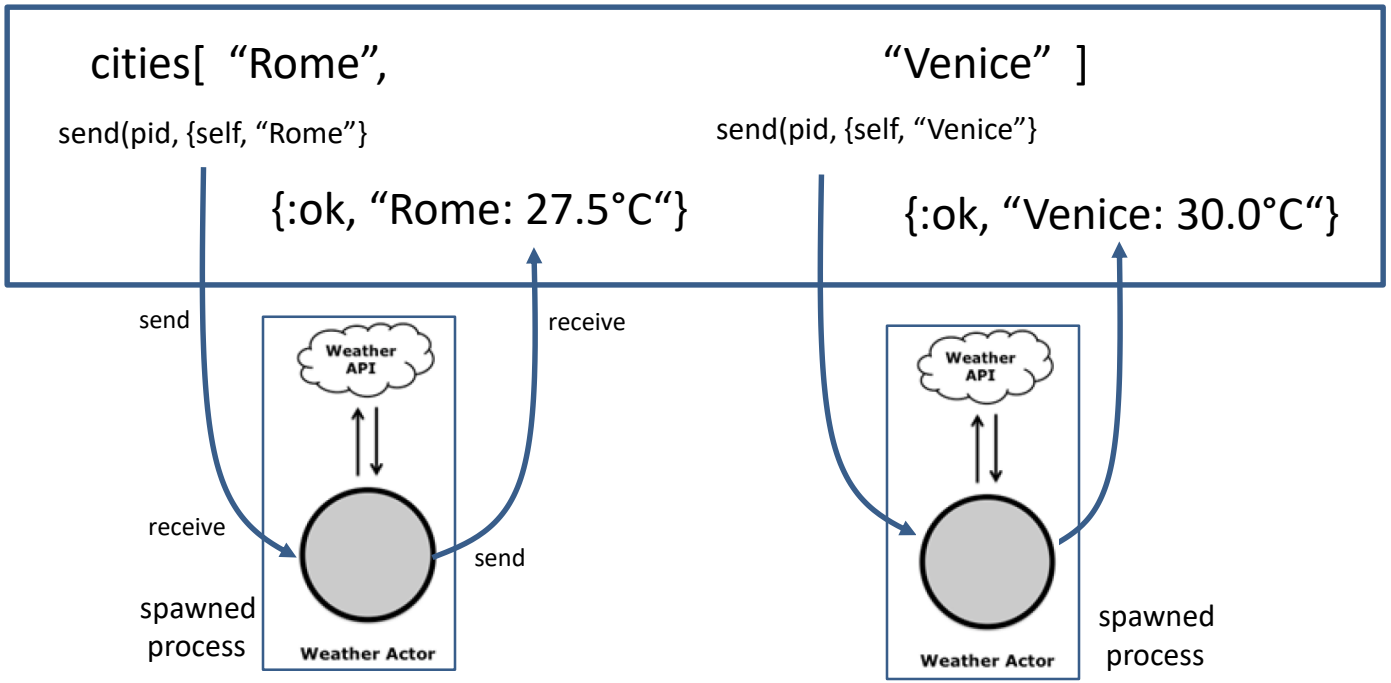
- Make temperature_of a process server
  - Add a loop
  - Spawn as a process



```elixir
defmodule Metex.Worker do

  def loop do
    receive do
      {sender_pid, location} ->
        send(sender_pid, {:ok, temperature_of(location)})
      _ ->
        IO.puts "don't know how to process this message"
    end
    loop
  end

  def temperature_of(location) do
    result = url_for(location) |> HTTPoison.get |> parse_response

    case result do
      {:ok, temp} ->
        "#{location}: #{temp}°C"
      :error ->
        "#{location} not found"
    end
  end

  defp url_for(location) do
    "http://api.openweathermap.org/data/2.5/weather?q=#{location}&APPID=#{apikey}"
```

Demo: Loop task

- Start multiple workers and send each a message
  - cities |> Enum.each(fn city -> pid = spawn(Metex.Worker, :loop, [])
    send(pid, {self, city}) end)



OTP: Easier Concurrency

- OTP defines systems in terms of hierarchies of applications which follow a small number of conventions called behaviors
- OTP behaviors
  - GenServer: A behavior module for implementing the server of a client-server relation.
  - GenEvent: Implements event handling
  - Supervisor: Implements supervision of other behavior modules
- We will work with GenServer and Supervisor

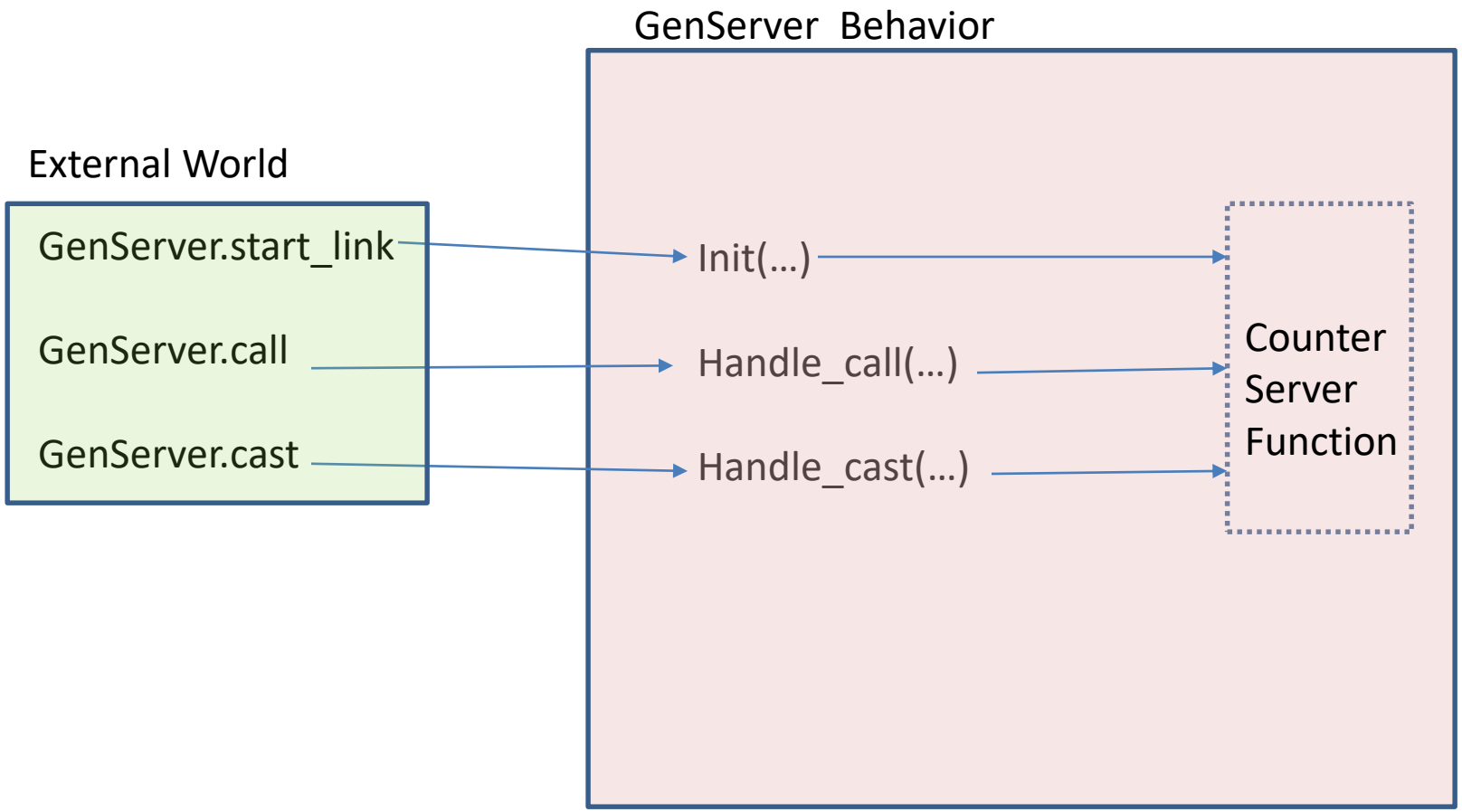
GenServer…What is the Advantage

- Synchronous and asynchronous requests
- Different and growing number of message types.
- Replies are equally well handled
- State management (initial and on-going)
- Graceful termination
- Recognizable pattern (a la Ruby/Rails)
- Potential for higher level organization

Counting Example using GenServer

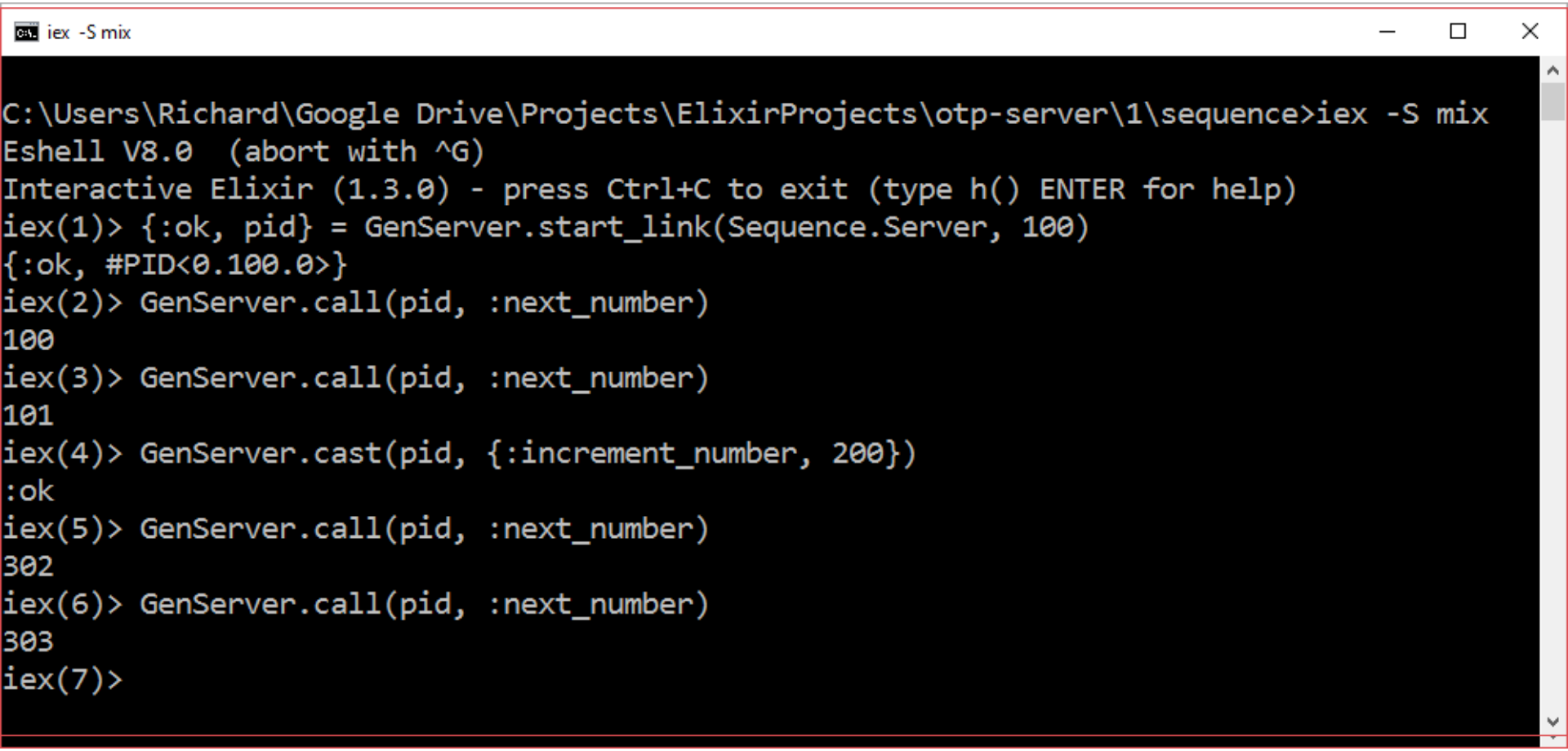
- A server that returns the next integer
  - Initialize the count to n
  - Call for the next integer
  - Add delta to the count
- Tasks
  - Spawn the server
  - Handle :next_number message
  - Handle :increment_number message
  - Store state, return state

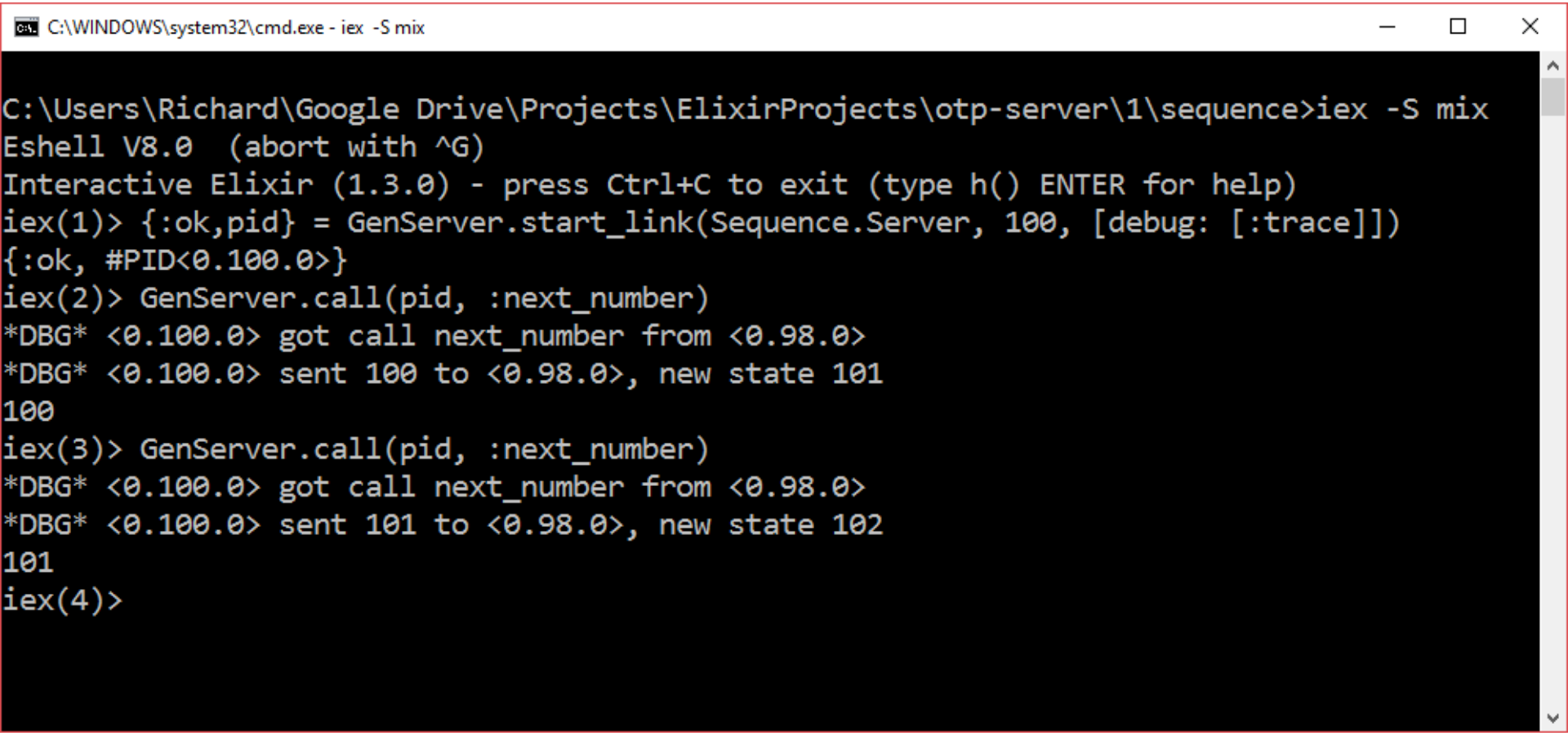
GenServer Extensions to Counter Server



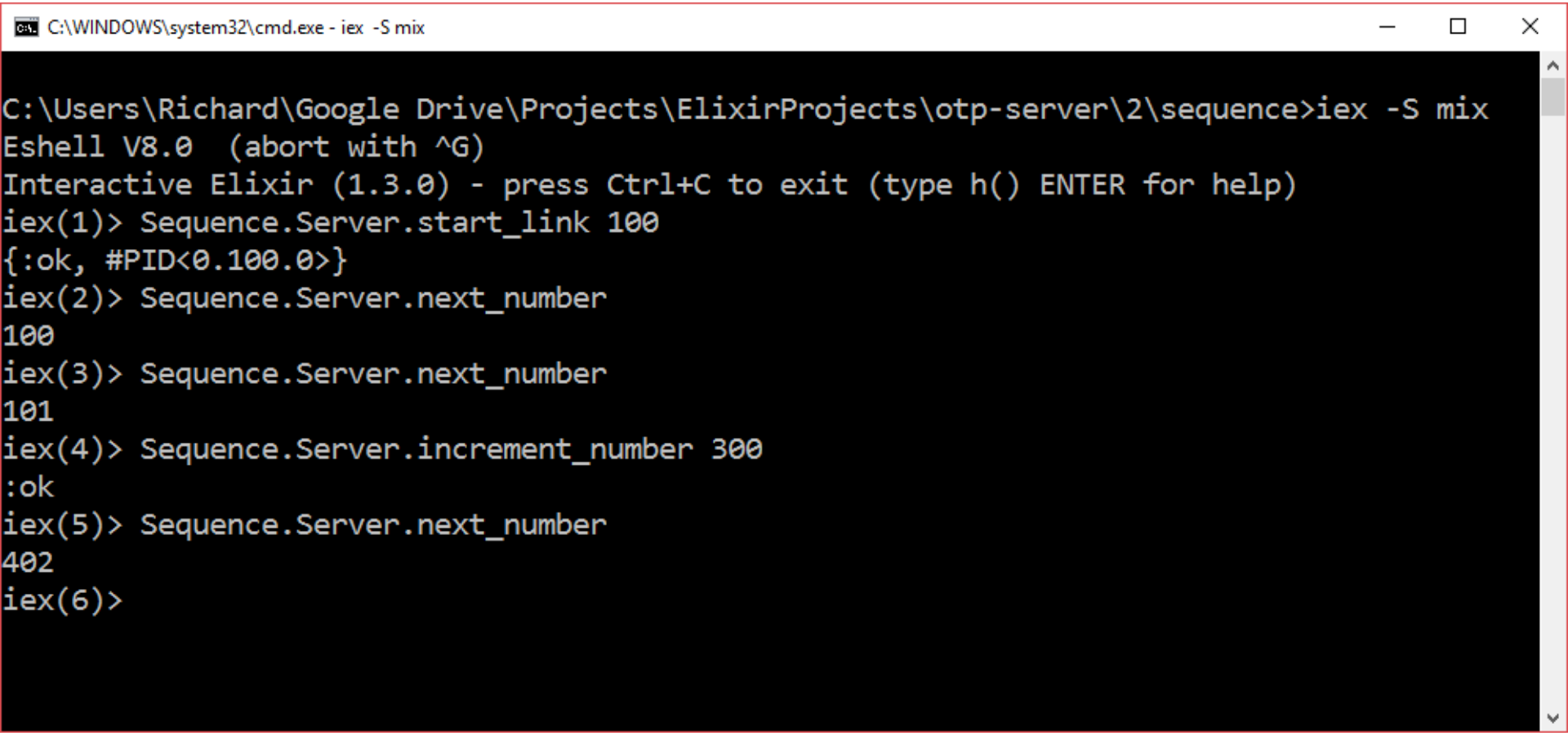
GenServer behavior Example

```elixir
defmodule Sequence.Server do
  use GenServer

  def handle_call(:next_number, _from, current_number) do
    { :reply, current_number, current_number+1 }
  end

  def handle_cast({:increment_number, delta}, current_number) do
    { :noreply, current_number + delta}
  end
end
```


iex -S mix
C:\Users\Richard\Google Drive\Projects\ElixirProjects\otp-server\1\sequence>iex -S mix
Eshell V8.0 (abort with ^G)
Interactive Elixir (1.3.0) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> {:ok, pid} = GenServer.start_link(Sequence.Server, 100)
{:ok, #PID<0.100.0>}
iex(2)> GenServer.call(pid, :next_number)
100
iex(3)> GenServer.call(pid, :next_number)
101
iex(4)> GenServer.cast(pid, {:increment_number, 200})
:ok
iex(5)> GenServer.call(pid, :next_number)
302
iex(6)> GenServer.call(pid, :next_number)
303
iex(7)>


C:\WINDOWS\system32\cmd.exe - iex -S mix
C:\Users\Richard\Google Drive\Projects\ElixirProjects\otp-server\1\sequence>iex -S mix
Eshell V8.0 (abort with ^G)
Interactive Elixir (1.3.0) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> {:ok,pid} = GenServer.start_link(Sequence.Server, 100, [debug: [:trace]])
{:ok, #PID<0.100.0>}
iex(2)> GenServer.call(pid, :next_number)
DBG <0.100.0> got call next_number from <0.98.0>
DBG <0.100.0> sent 100 to <0.98.0>, new state 101
100
iex(3)> GenServer.call(pid, :next_number)
DBG <0.100.0> got call next_number from <0.98.0>
DBG <0.100.0> sent 101 to <0.98.0>, new state 102
101
iex(4)>

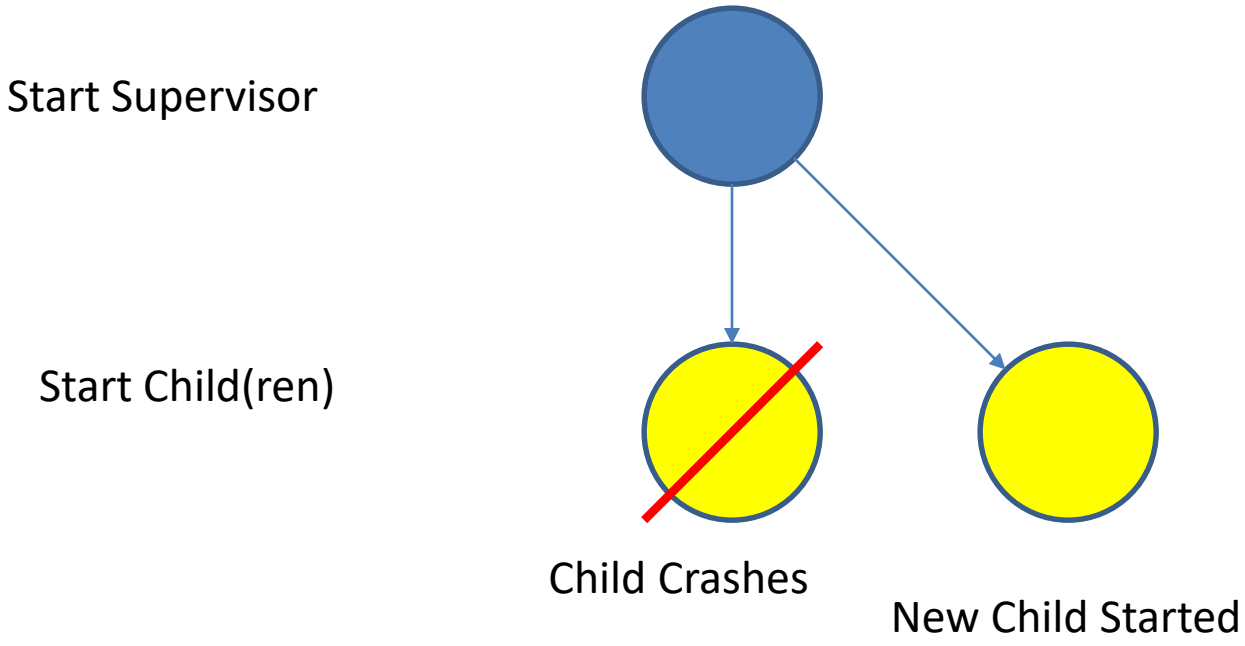
```elixir
defmodule Sequence.Server do
  use GenServer

  #####
  # External API

  def start_link(current_number) do
    GenServer.start_link(__MODULE__, current_number, name: __MODULE__)
  end

  def next_number do
    GenServer.call __MODULE__, :next_number
  end

  def increment_number(delta) do
    GenServer.cast __MODULE__, {:increment_number, delta}
  end

  #####
  # GenServer implementation

  def handle_call(:next_number, _from, current_number) do
    { :reply, current_number, current_number+1 }
  end

  def handle_cast({:increment_number, delta}, current_number) do
    { :noreply, current_number + delta}
  end

  def format_status(_reason, [ _pdict, state ]) do
    [data: [{'State', "My current state is '#{inspect state}', and I'm happy"}]]
  end
end
```


C:\WINDOWS\system32\cmd.exe - iex -S mix
C:\Users\Richard\Google Drive\Projects\ElixirProjects\otp-server\2\sequence>iex -S mix
Eshell V8.0 (abort with ^G)
Interactive Elixir (1.3.0) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> Sequence.Server.start_link 100
{:ok, #PID<0.100.0>}
iex(2)> Sequence.Server.next_number
100
iex(3)> Sequence.Server.next_number
101
iex(4)> Sequence.Server.increment_number 300
:ok
iex(5)> Sequence.Server.next_number
402
iex(6)>

Intro Supervisor (let it crash)

- Supervisor processes manage other processes
- Uses Supervisor Behavior
  - Monitors a list of processes
  - Performs an action if a process fails

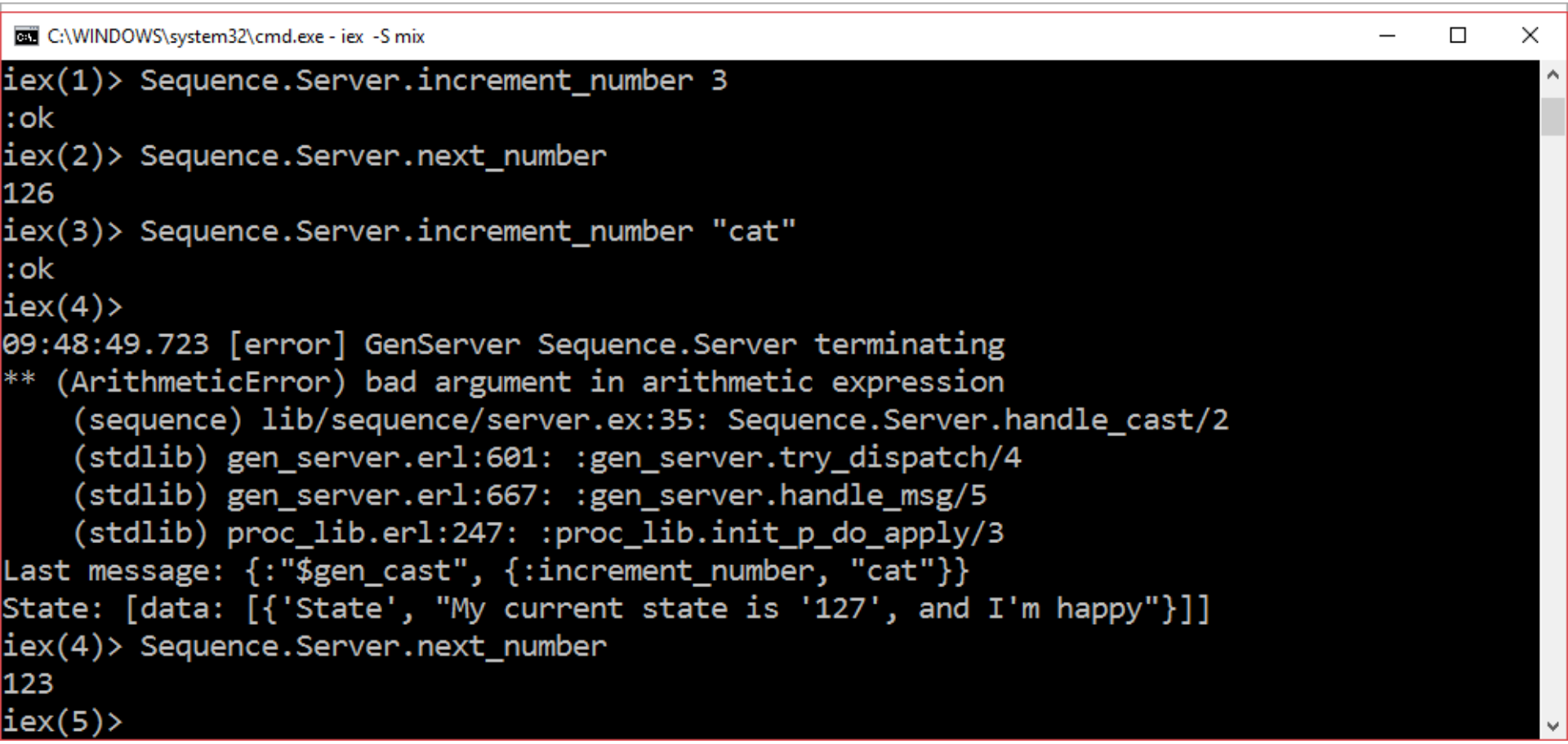
Supervisor Operation



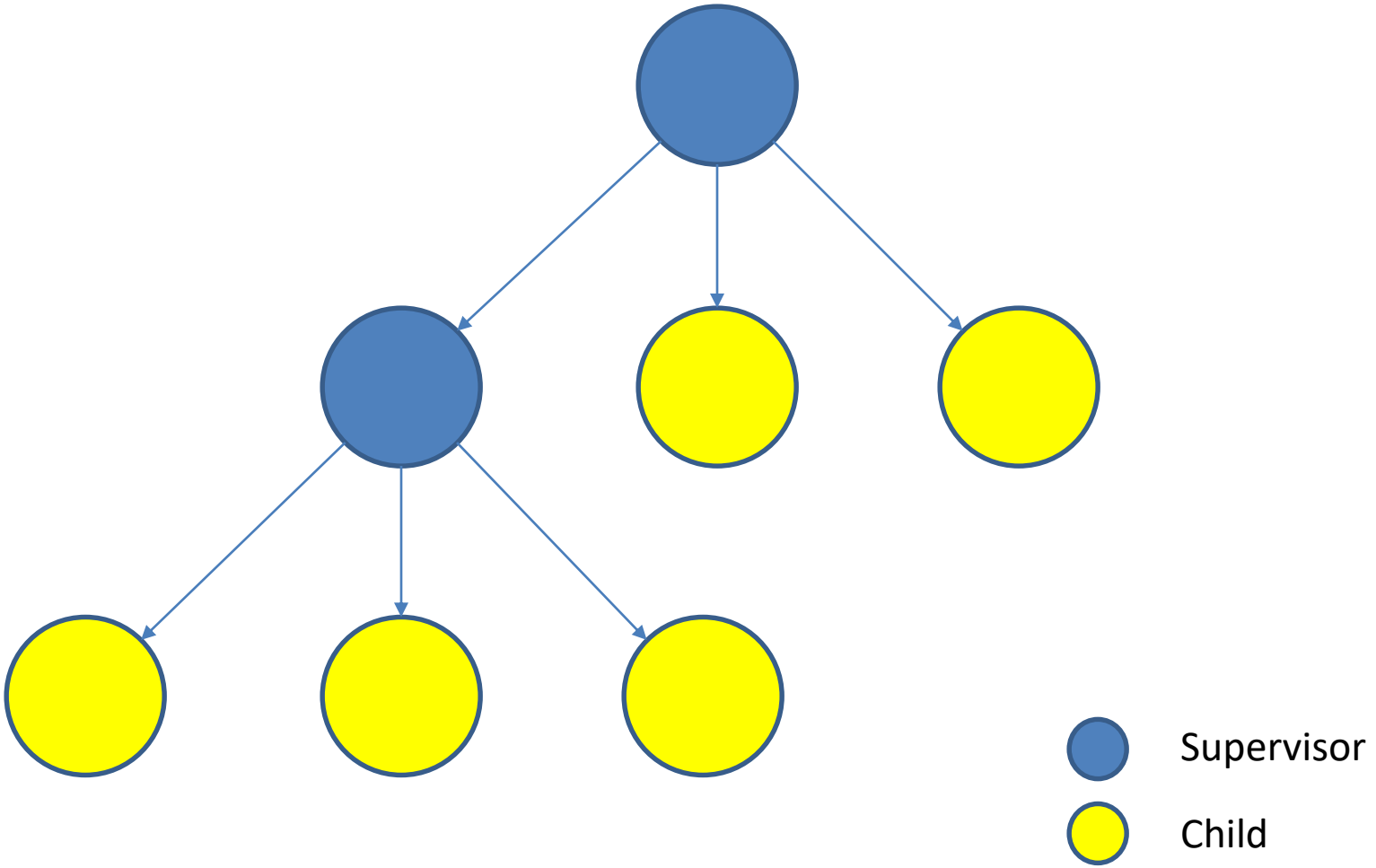
Supervisor Code

```
 9  defmodule Sequence do
10    use Application
11
12    def start(_type, _args) do
13      import Supervisor.Spec, warn: false
14
15      children = [
16        worker(Sequence.Server, [123])
17      ]
18
19      opts = [strategy: :one_for_one, name: Sequence.Supervisor]
20      {:ok, _pid} = Supervisor.start_link(children, opts)
21    end
22  end
23
```

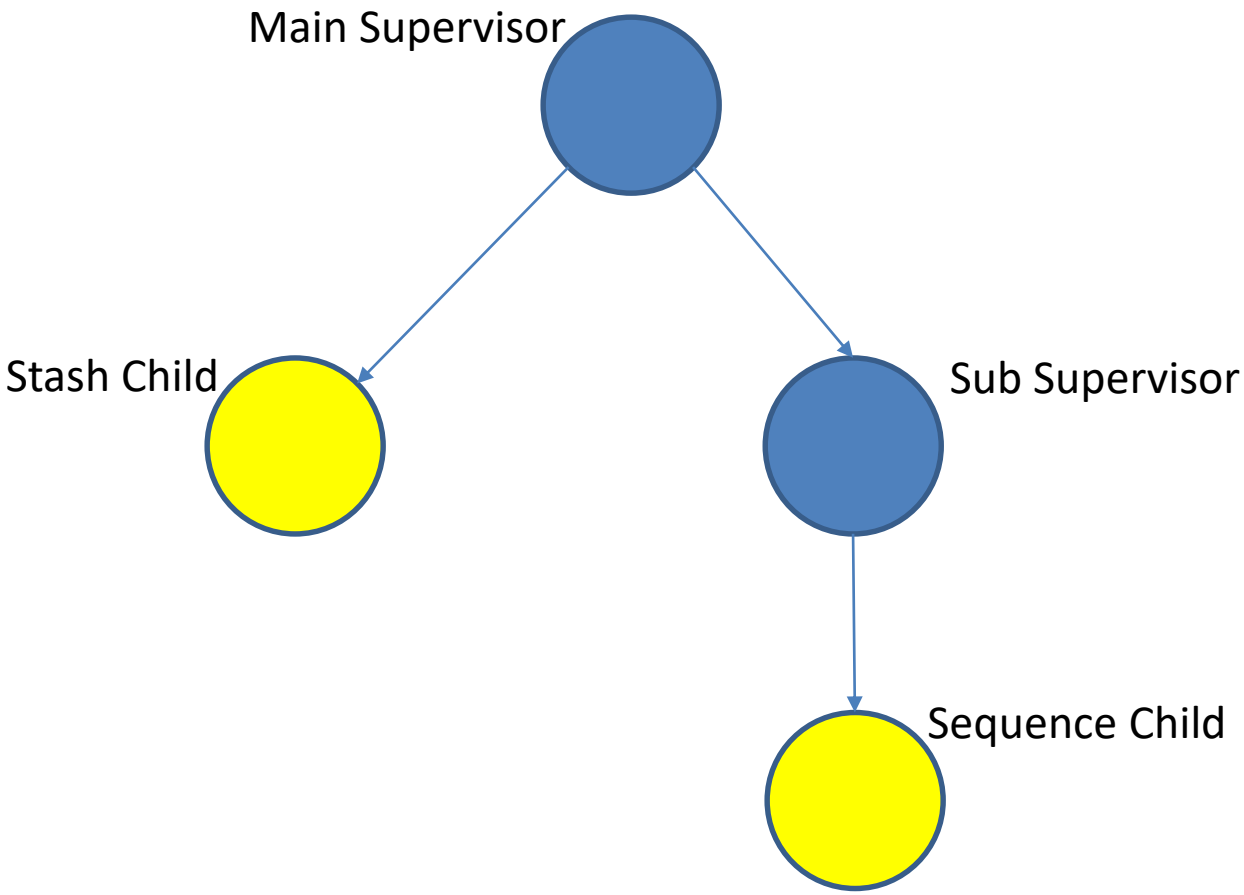
Supervisor in Operation



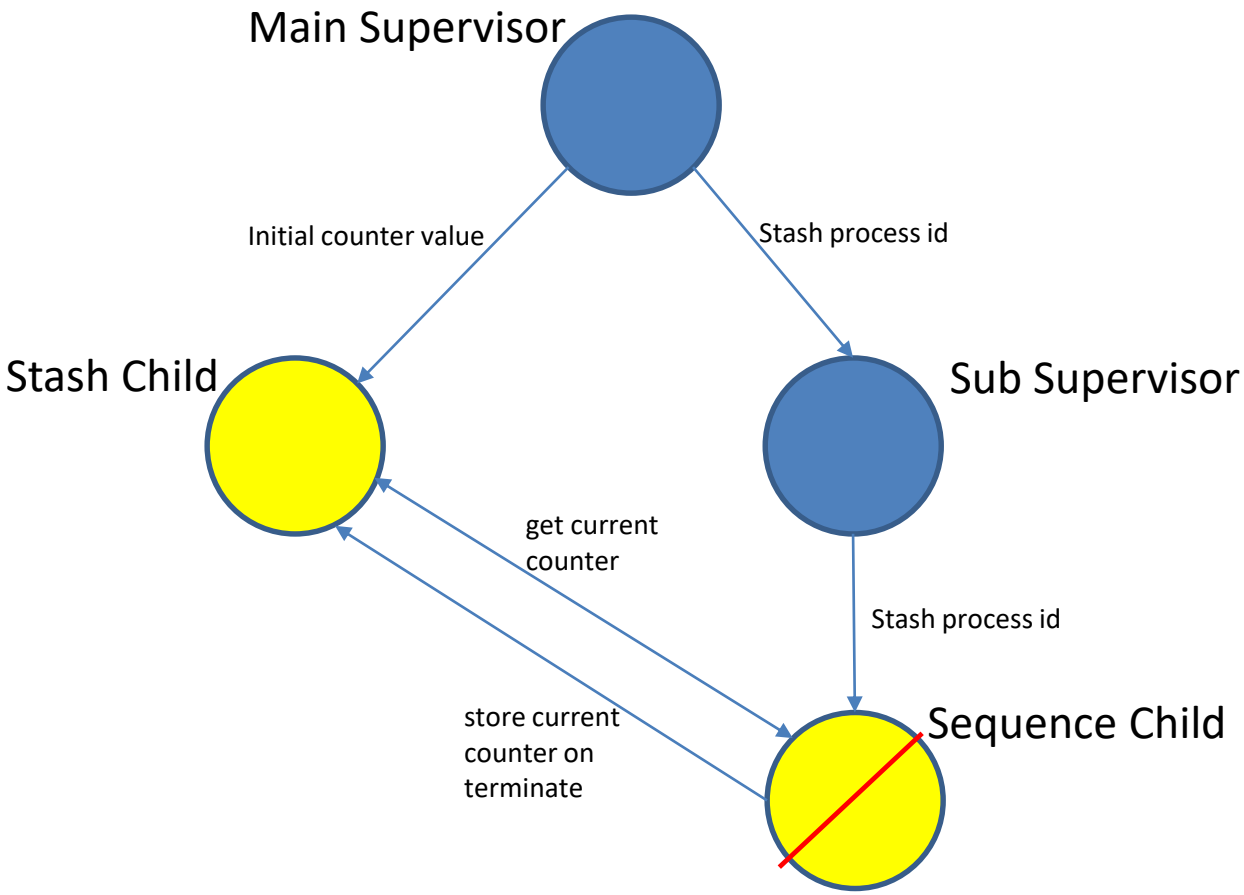
Supervisor Hierarchy



Retain Child State with a Second Child and Supervisor



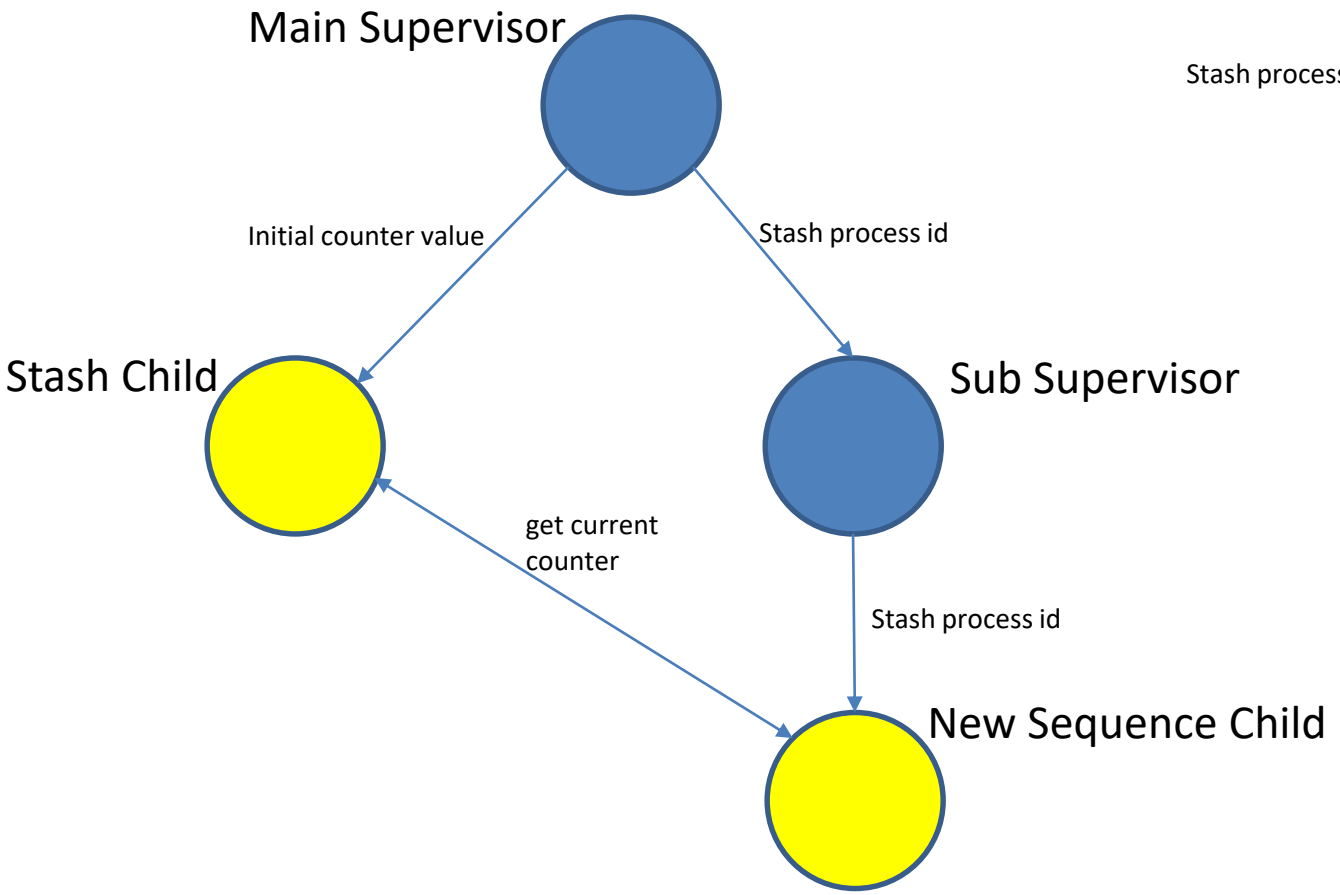
Retain Child State with a Second Child and Supervisor

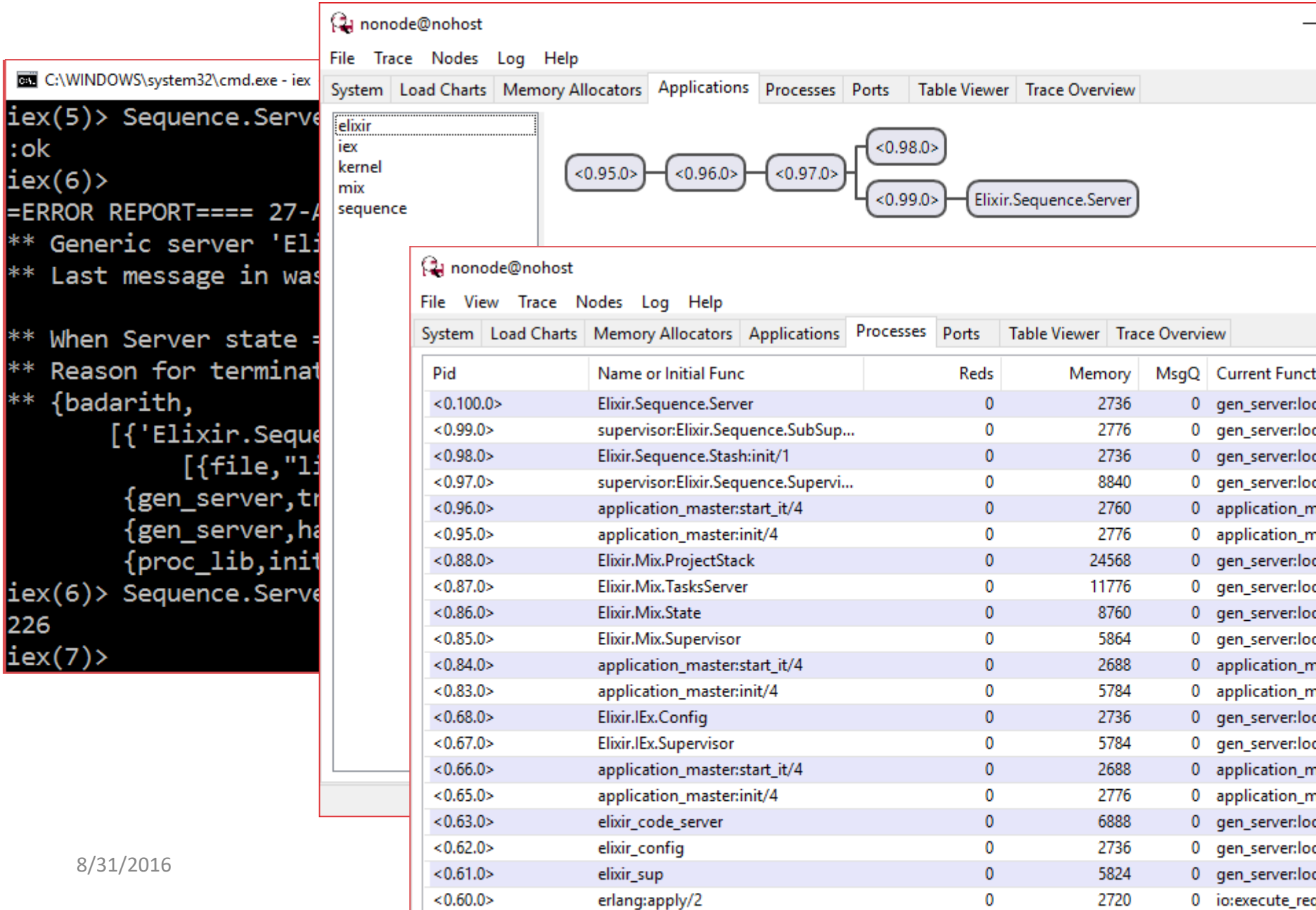


Sequence Counter State Recovery Process

- Start Top Level Supervisor with counter value
  - Start Stash child with counter value
  - Start Sub Supervisor with Stash PID
- Sub Supervisor starts sequence child who gets counter value from stash.
- Sequence child records counter state as it fails.
- Sub Supervisor notices Sequence child has died. It creates a new Sequence child with Stash PID
- Newly created Sequence Child will pick up the current value of the counter from the Stash Child.

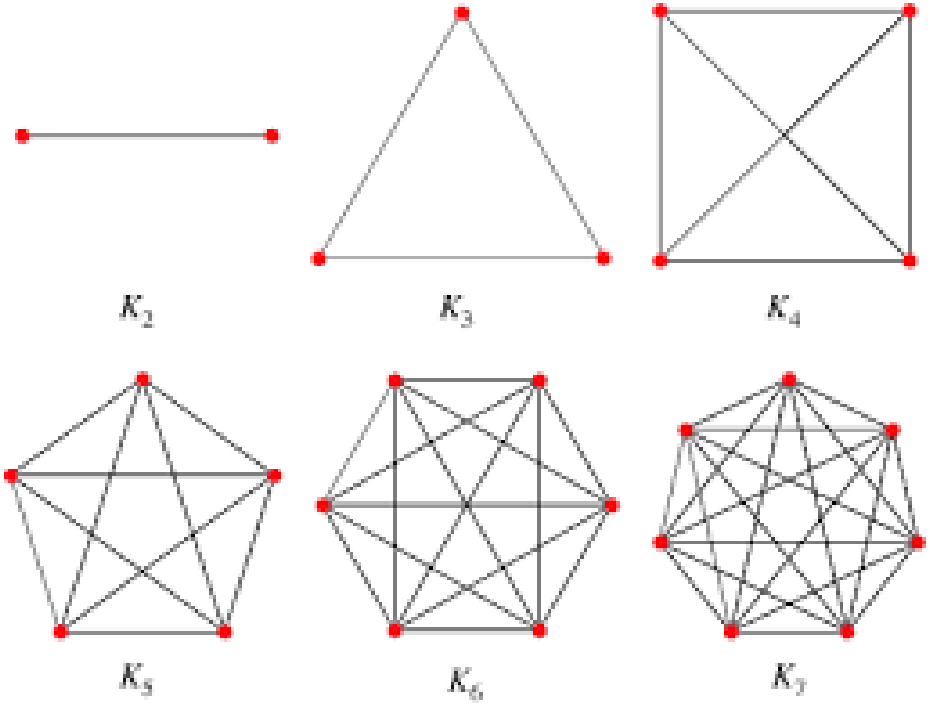
Retain Child State with a Second Child and Supervisor

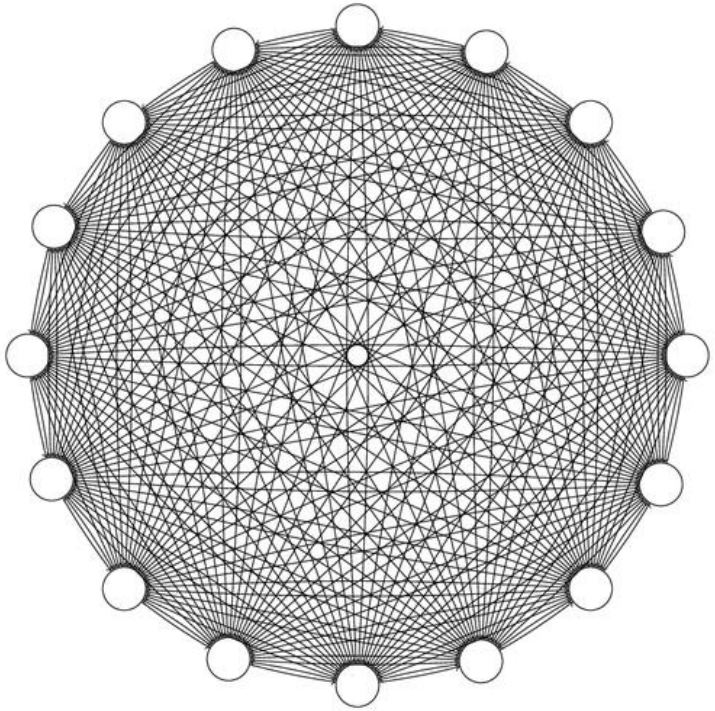


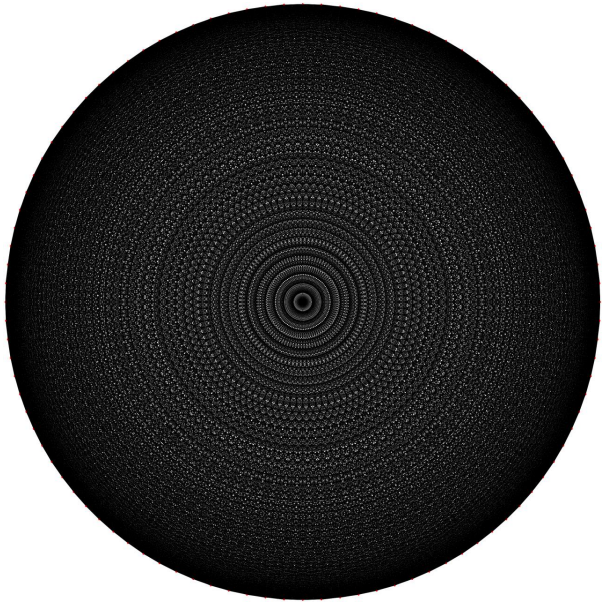

C:\WINDOWS\system32\cmd.exe - iex
iex(5)> Sequence.Serve
:ok
iex(6)>
=ERROR REPORT==== 27-A
** Generic server 'Eli
** Last message in was
** When Server state =
** Reason for terminat
** {badarith,
[{'Elixir.Seque
[{file,"li
{gen_server,tr
{gen_server,ha
{proc_lib,init
iex(6)> Sequence.Serve
226
iex(7)>
nonode@nohost
File Trace Nodes Log Help
System Load Charts Memory Allocators Applications Processes Ports Table Viewer Trace Overview
elixir
iex
kernel
mix
sequence
<0.95.0>
<0.96.0>
<0.97.0>
<0.98.0>
<0.99.0>
Elixir.Sequence.Server
nonode@nohost
File View Trace Nodes Log Help
System Load Charts Memory Allocators Applications Processes Ports Table Viewer Trace Overview
Pid
Name or Initial Func
Reds
Memory
MsgQ
Current Funct
<0.100.0> Elixir.Sequence.Server 0 2736 0 gen_server:loo
<0.99.0> supervisor:Elixir.Sequence.SubSup... 0 2776 0 gen_server:loo
<0.98.0> Elixir.Sequence.Stash:init/1 0 2736 0 gen_server:loo
<0.97.0> supervisor:Elixir.Sequence.Supervi... 0 8840 0 gen_server:loo
<0.96.0> application_master:start_it/4 0 2760 0 application_m
<0.95.0> application_master:init/4 0 2776 0 application_m
<0.88.0> Elixir.Mix.ProjectStack 0 24568 0 gen_server:loo
<0.87.0> Elixir.Mix.TasksServer 0 11776 0 gen_server:loo
<0.86.0> Elixir.Mix.State 0 8760 0 gen_server:loo
<0.85.0> Elixir.Mix.Supervisor 0 5864 0 gen_server:loo
<0.84.0> application_master:start_it/4 0 2688 0 application_m
<0.83.0> application_master:init/4 0 5784 0 application_m
<0.68.0> Elixir.IEx.Config 0 2736 0 gen_server:loo
<0.67.0> Elixir.IEx.Supervisor 0 5784 0 gen_server:loo
<0.66.0> application_master:start_it/4 0 2688 0 application_m
<0.65.0> application_master:init/4 0 2776 0 application_m
<0.63.0> elixir_code_server 0 6888 0 gen_server:loo
<0.62.0> elixir_config 0 2736 0 gen_server:loo
<0.61.0> elixir_sup 0 5824 0 gen_server:loo
<0.60.0> erlang:apply/2 0 2720 0 io:execute_req

Summary

- Concurrency is a powerful tool to make use of multi-core hardware. But it must be designed… there is no free lunch
- Elixir/Erlang is a proven technology for handling all aspects of concurrency including robustness. (it is not the only technology, see Scala and AKKA)
- Concurrency should be in your tool kit, it will be a mainstream pattern in the future.


K_2
K_3
K_4
K_5
K_6
K_7





What Makes Elixir/Erlang Different

- Not OO – Expand on this
- The good and the bad
- Need examples of how different
- Need examples of how good or bad
- Credit 1.3 and Little E

Implementing Concurrent Programming

- OTP on BEAM rather than AKKA on JVM
- Tool demo here