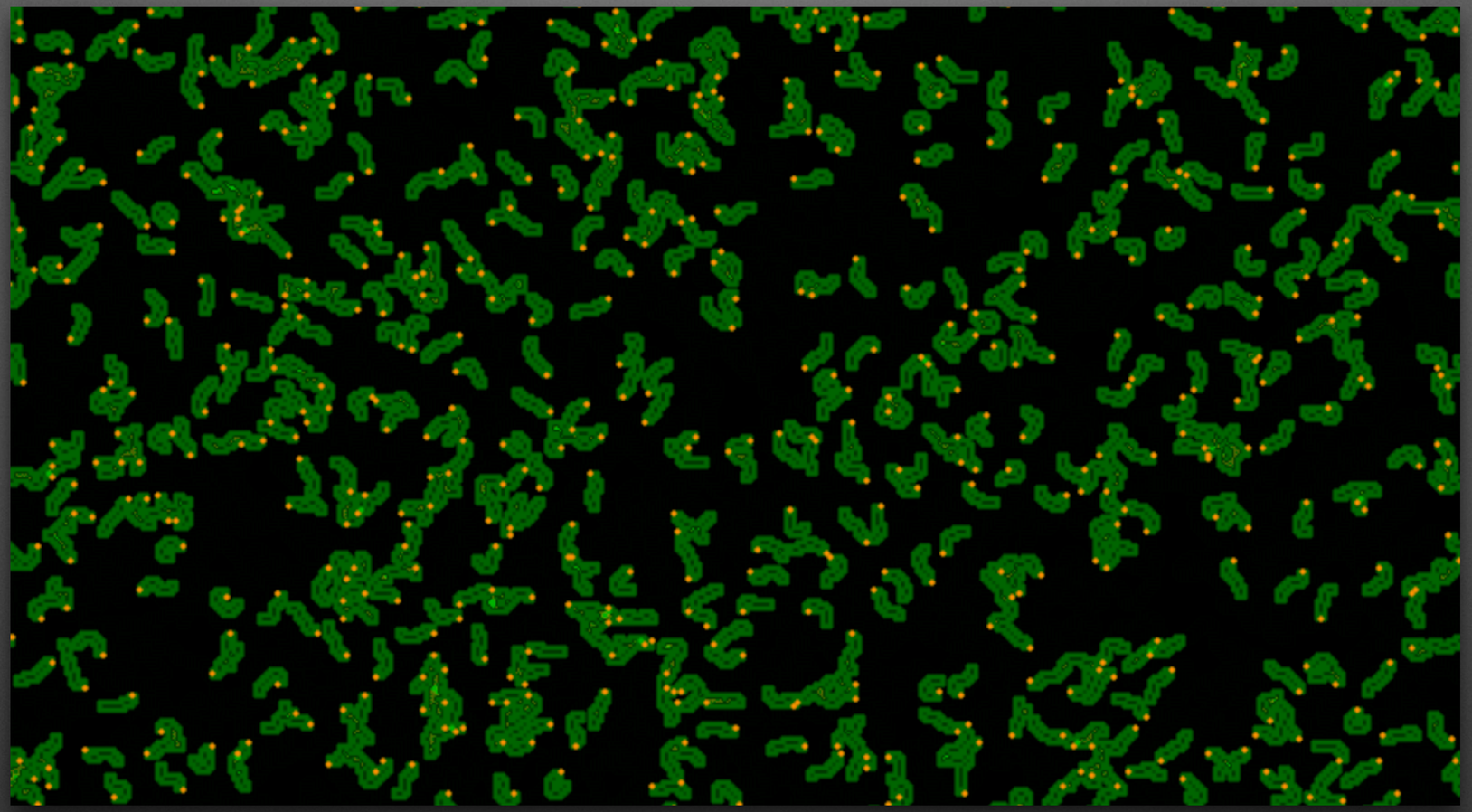
OTP Strategy

SimAlchemy, Part 4

Learn From My Mistakes

- I made plenty of mistakes when learning the OTP
- Let’s see what we can learn from some of them
- We’ll try to synthesize some strategies to avoid similar problems
- Then you can make all new mistakes!



Slime Mold: Spreading Pheromones

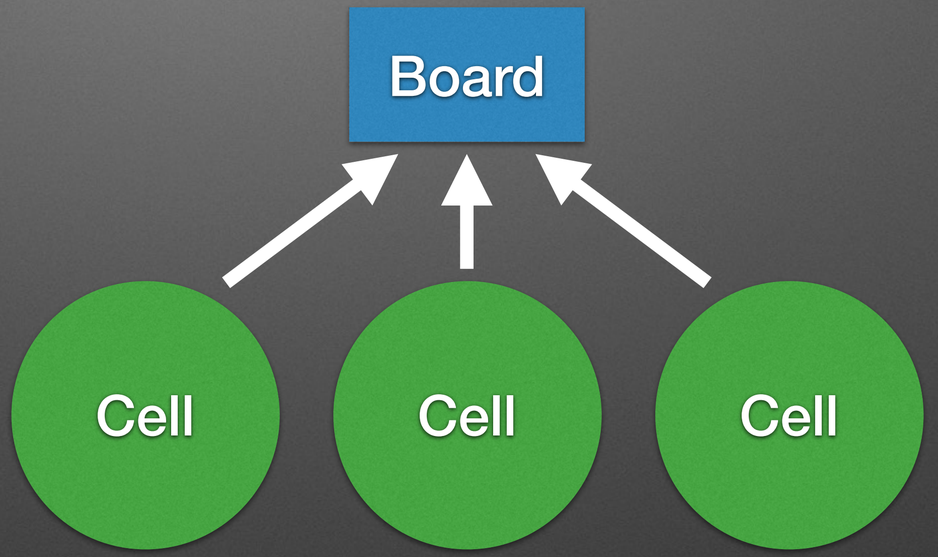
More processes!

The Simulation

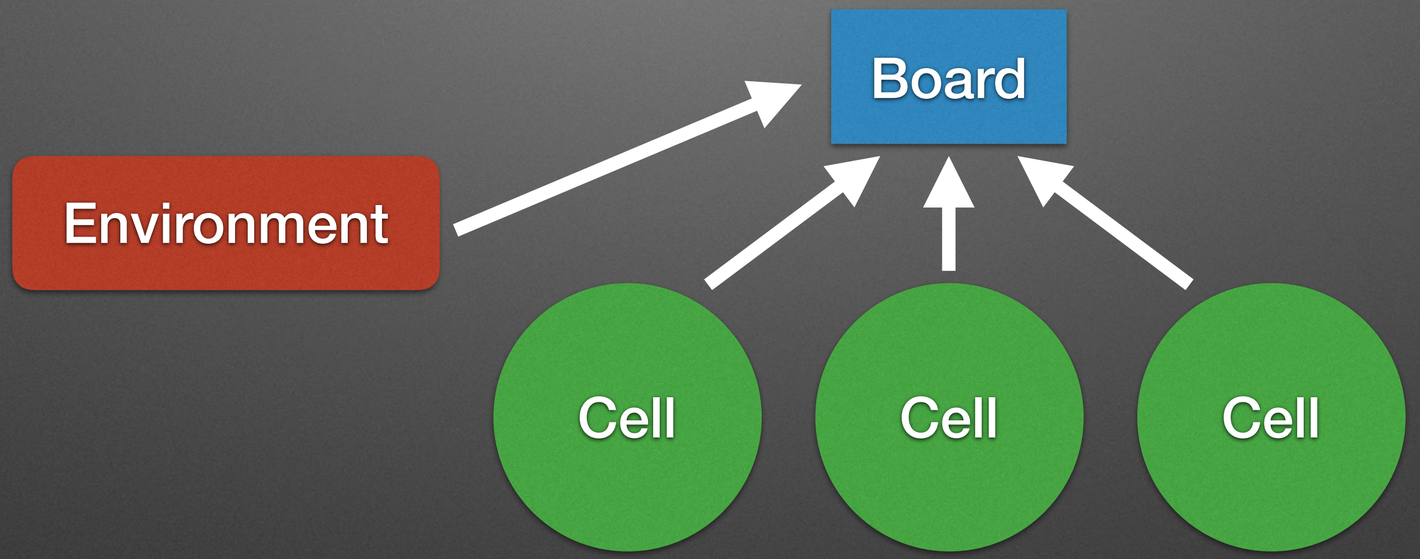
- Slime mold cells move around dropping pheromones as they go
- Pheromones on the ground change in two ways
  - They dissipate over time
  - They spread to nearby locations
- When cells move they prefer higher pheromone locations

My Design

My Design



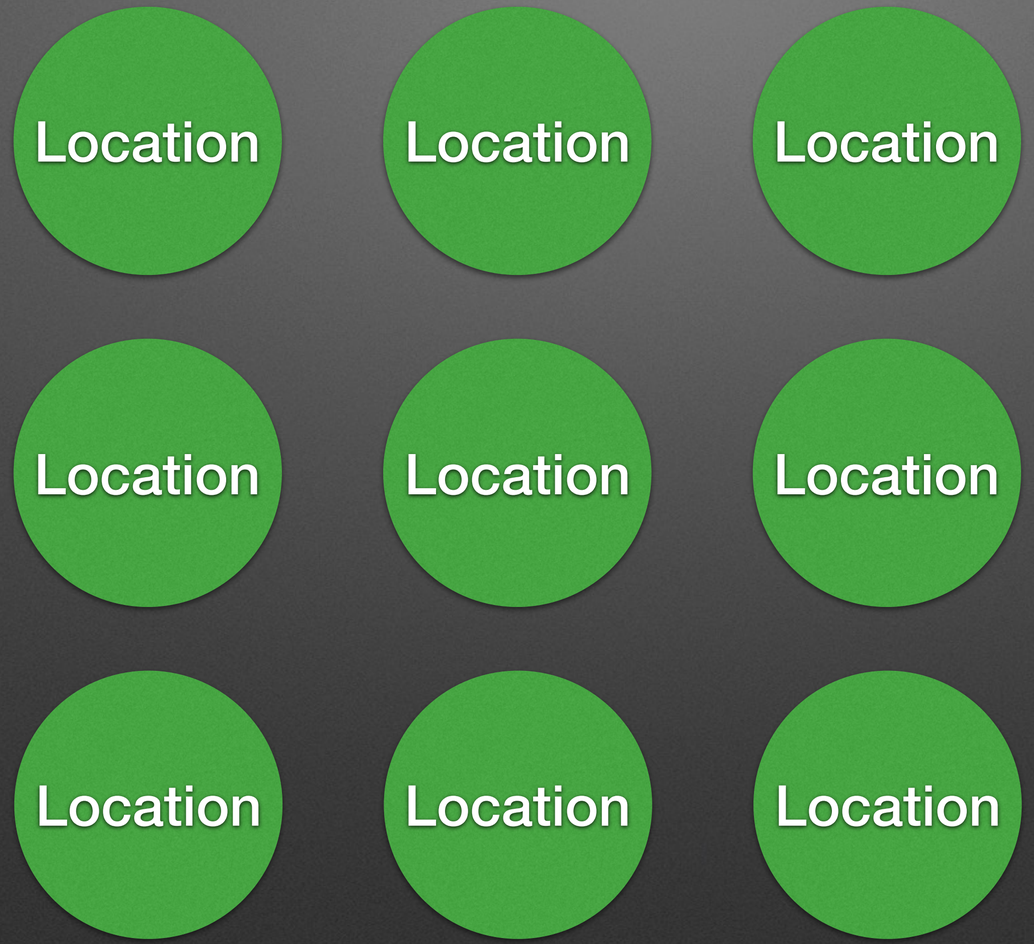
My Design



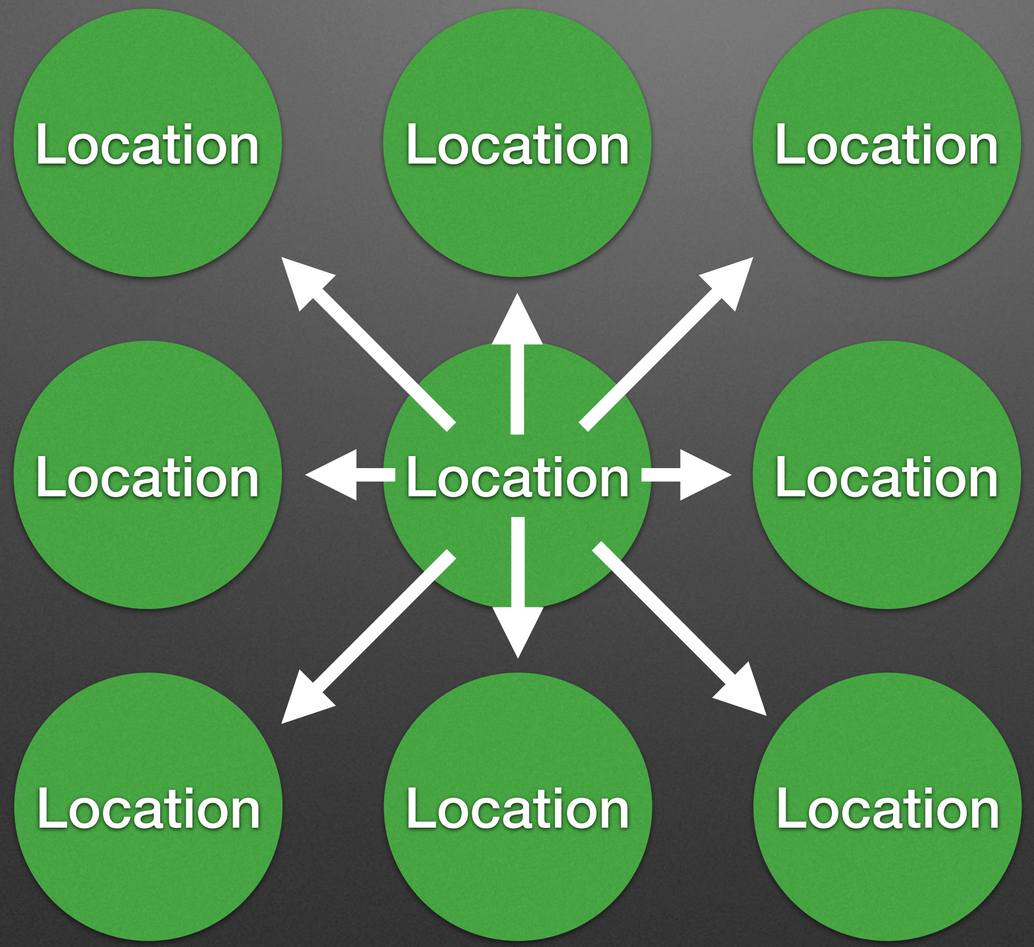
The Problem

- The Environment process stops the world
  - Board must pause to check all cells for dissipation and spreading
  - It must also decide if the change warrants a redraw (rare)
- This is a time consuming bottleneck
- Messages from cells pile up behind it

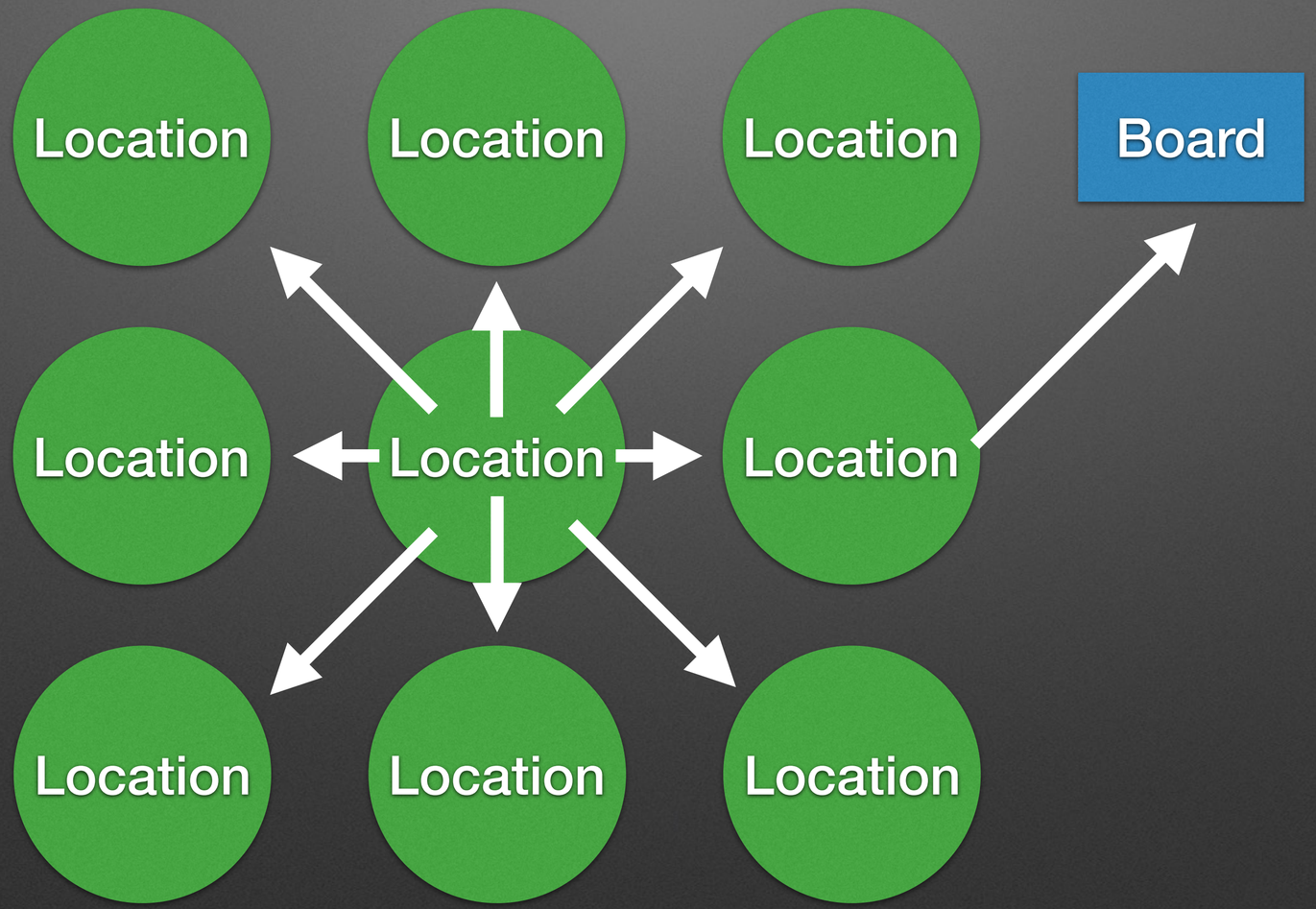
A Better Design



A Better Design



A Better Design



Strategies

- Take the time to understand the implications of having mailboxes
- Don’t be afraid of adding processes
- Distribute the work
- Stay aware of the tension between a single source of truth and bottlenecks

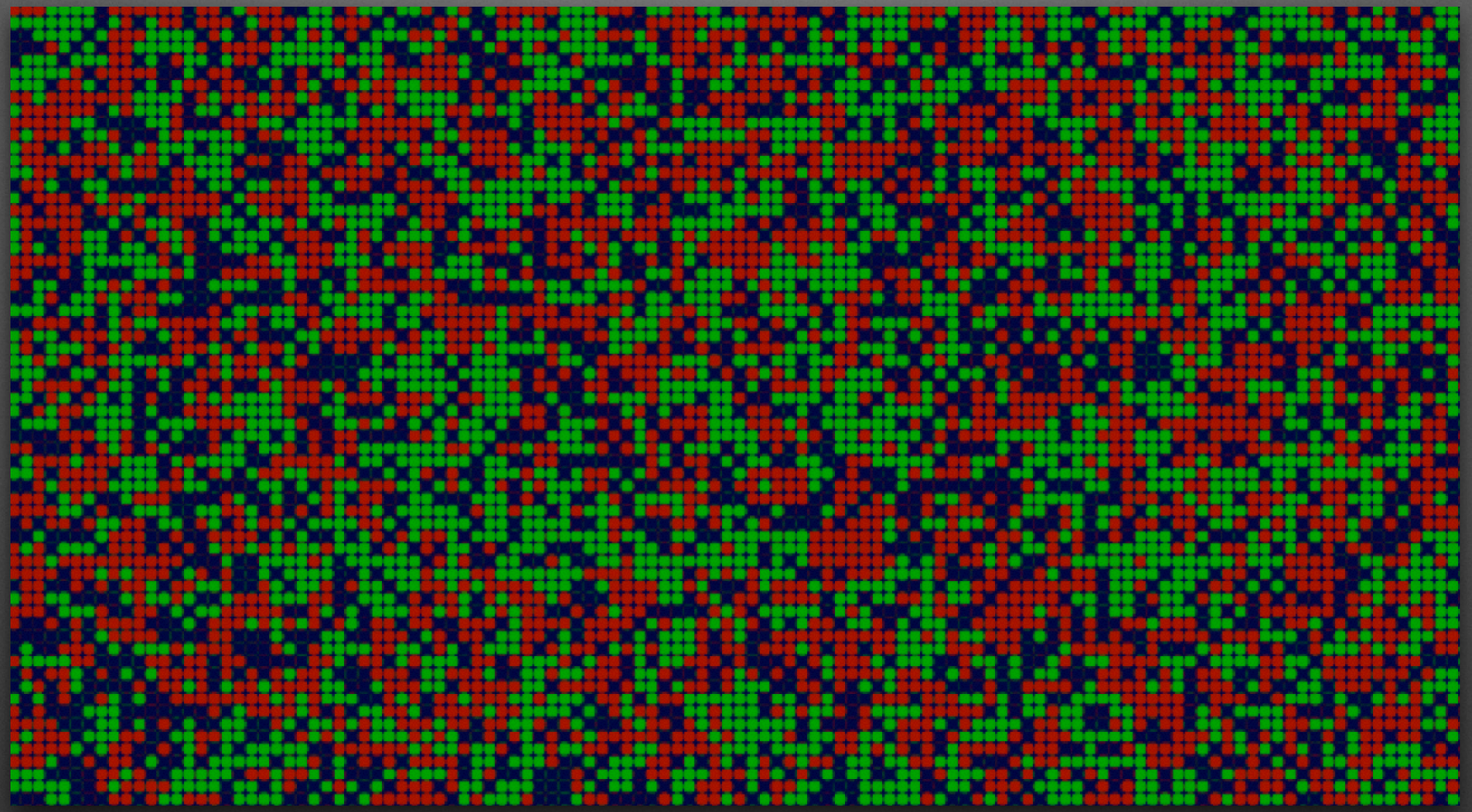
Common Concerns

Your Old Knowledge Still Applies

- Encapsulation is still good (and kind of enforced now)
- The Single Responsibility Principle (SRP) is still important
- You still need to manage dependencies

A Little Drawing Goes a Long Way

- Process design is easier for me to understand with drawings
- I’ve seen some use UML Sequence Diagrams
- Even simple circles and arrows help me



Turtles and Frogs

Look Ma, no GenServer!

The Simulation

- Start with an even distribution of turtles and frogs
- At each tick a critter is happy if it has at least three same species neighbors
- If unhappy a critter tries to jump to an open location in a random direction

The Problem

- I had discovered Agent and Task!
  - They are great tools where they fit
  - But overusing them creates more pain than it saves
- There’s no GenServer in this code (probably a smell)

GenServer Envy

```
defmodule TurtlesAndFrogs.Critter do
  # …

  defp loop(xy, neighbor_xys, type, size) do
    :timer.sleep(333)

    if happy?(neighbor_xys, type) do
      loop(xy, neighbor_xys, type, size)
    else
      move_away_from(xy, type, size) |> loop(neighbor_xys, type, size)
    end
  end

  # …
end
```

Agent and Task Strategies

- Remember that these are very specialized tools
- Switch to GenServer when you surpass their specialty
- Ensure that Agent is for data tracking only
  - No logic beyond transformations at store and retrieve
- Task should not `receive`

Testing Tips

use ExUnit.Case, async: true

- Run tests asynchronously as often as possible
- This can expose dependencies that affect multiple processes

Don’t :timer.send_interval/3 to self

- Wanting periodic messages is fine
- But avoid :timer.send_interval(…, self, …) in init/1, or similar tricks
- Handle it externally, at startup for example
  - {:ok, pid} = Whatever.start_link(…)
  - :timer.send_interval(…, pid, …)

Test PIDs, Supervise Names

```elixir
defmodule ClockTest do
  # …

  test "plants are populated at the dawn of time", %{world: world} do
    {:ok, _clock} = Clock.start_link(world, @size)
    assert not Enum.empty?(World.changes(world).plants)
  end

  # …
end
```

```elixir
# …

worker(
  Turtles.Clock,
  [Turtles.World, {@width, @height}, [name: Turtles.Clock]]
),

# …
```

```elixir
defmodule Turtles.Clock do
  # …

  def start_link(world, size, options \\ [ ]) do
    clock = %__MODULE__{world: world, size: size}
    GenServer.start_link(__MODULE__, clock, options)
  end

  # …
end
```

Traffic Jam

Deadlock in the Gridlock

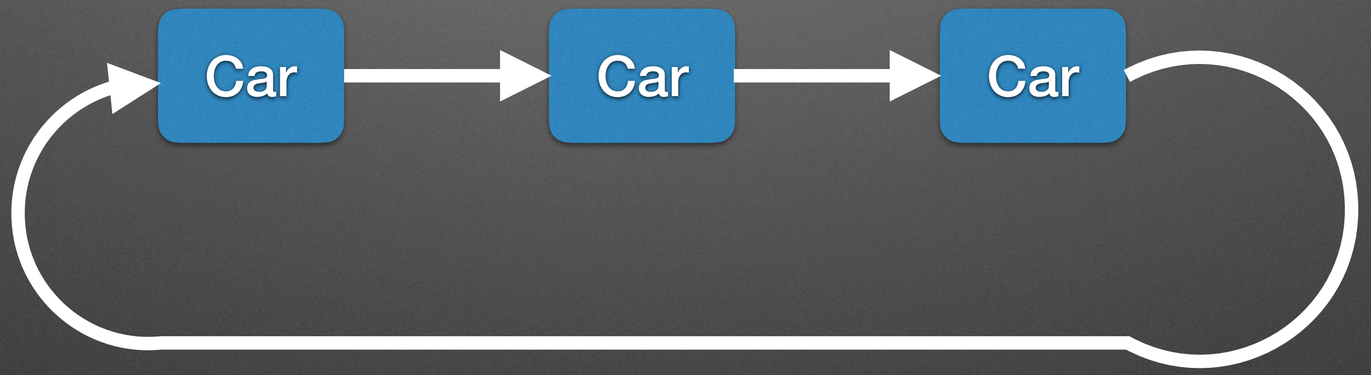
The Simulation

- Place cars randomly apart on a road
- A car accelerates, up to a max, if it’s a safe distance from the car in front of it
- A car decelerates if it’s getting close to the car in front of it

My Cheat

- Cars leave the road on one end and new cars should appear on the other
- I just wrapped the road, because it was easier
- This causes the system to crash eventually!

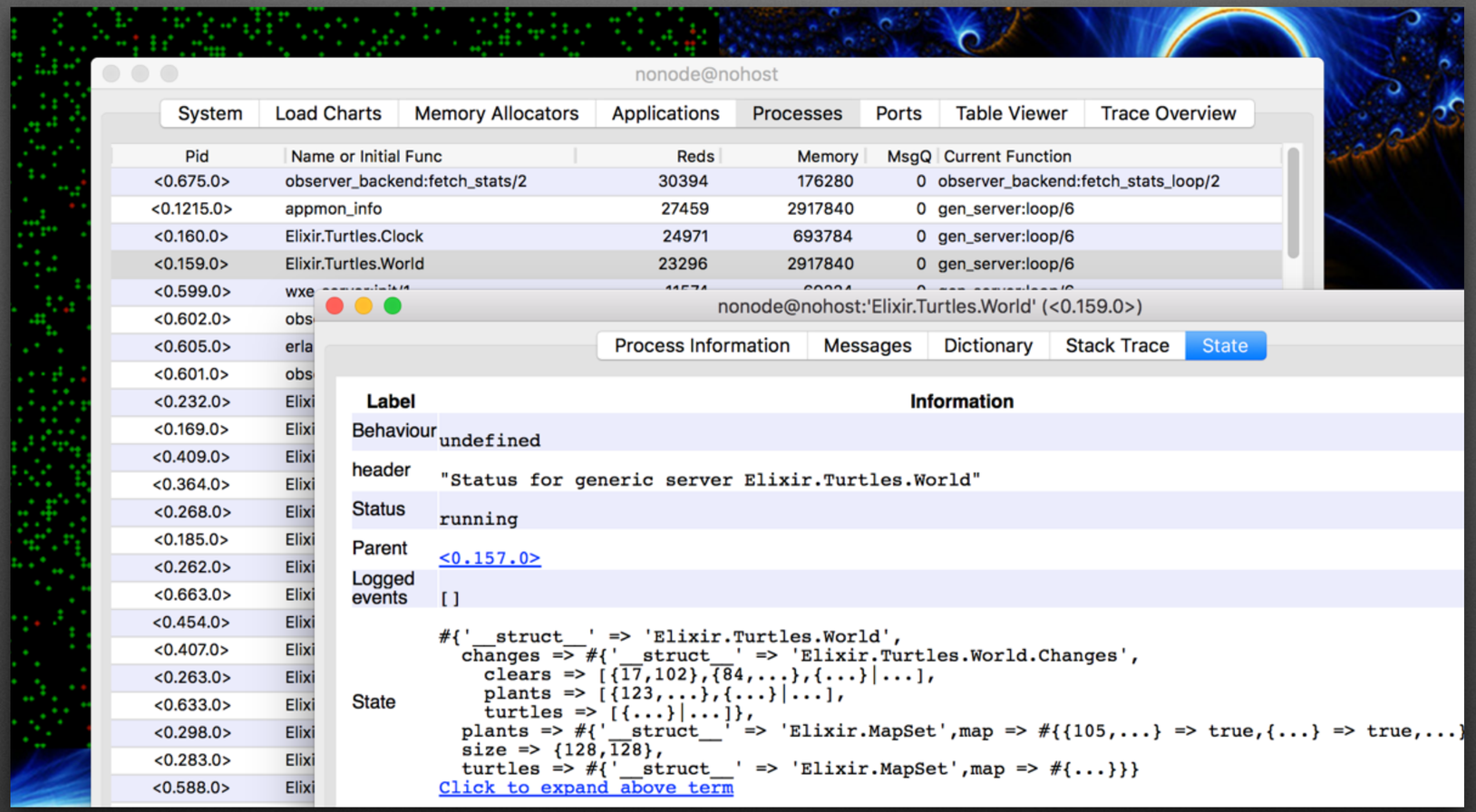
Circular Dependencies



Strategies

- Watch for GenServer’s timeouts (five seconds by default)
- Stay aware of direct and indirect process dependencies
- Know that my all-calls strategy can lead to deadlocks

Debugging Dos and Don’ts



:observer.start

Sometimes helps spot trends

:sys.get_state/1

```
iex(1)> :sys.get_state(Turtles.Clock)
%Turtles.Clock{size: {128, 128},
 turtles: #MapSet<[#PID<0.380.0>, #PID<0.184.0>, #PID<0.544.0>, #PID<0.175.0>,
  #PID<0.316.0>, #PID<0.157.0>, #PID<0.541.0>, #PID<0.193.0>, #PID<0.192.0>,
  #PID<0.313.0>, #PID<0.331.0>, #PID<0.255.0>, #PID<0.418.0>, #PID<0.305.0>,
  #PID<0.188.0>, #PID<0.337.0>, #PID<0.364.0>, #PID<0.348.0>, #PID<0.450.0>,
  #PID<0.432.0>, #PID<0.173.0>, #PID<0.190.0>, #PID<0.394.0>, #PID<0.274.0>,
  #PID<0.211.0>, #PID<0.180.0>, #PID<0.206.0>, #PID<0.446.0>, #PID<0.375.0>,
  #PID<0.546.0>, #PID<0.161.0>, #PID<0.342.0>, #PID<0.249.0>, #PID<0.327.0>,
  #PID<0.223.0>, #PID<0.187.0>, #PID<0.449.0>, #PID<0.304.0>, #PID<0.300.0>,
  #PID<0.239.0>, #PID<0.185.0>, #PID<0.431.0>, #PID<0.447.0>, #PID<0.403.0>,
  #PID<0.311.0>, #PID<0.374.0>, #PID<0.330.0>, #PID<0.530.0>, ...]>,
 world: Turtles.World}
```

:sys.trace/2

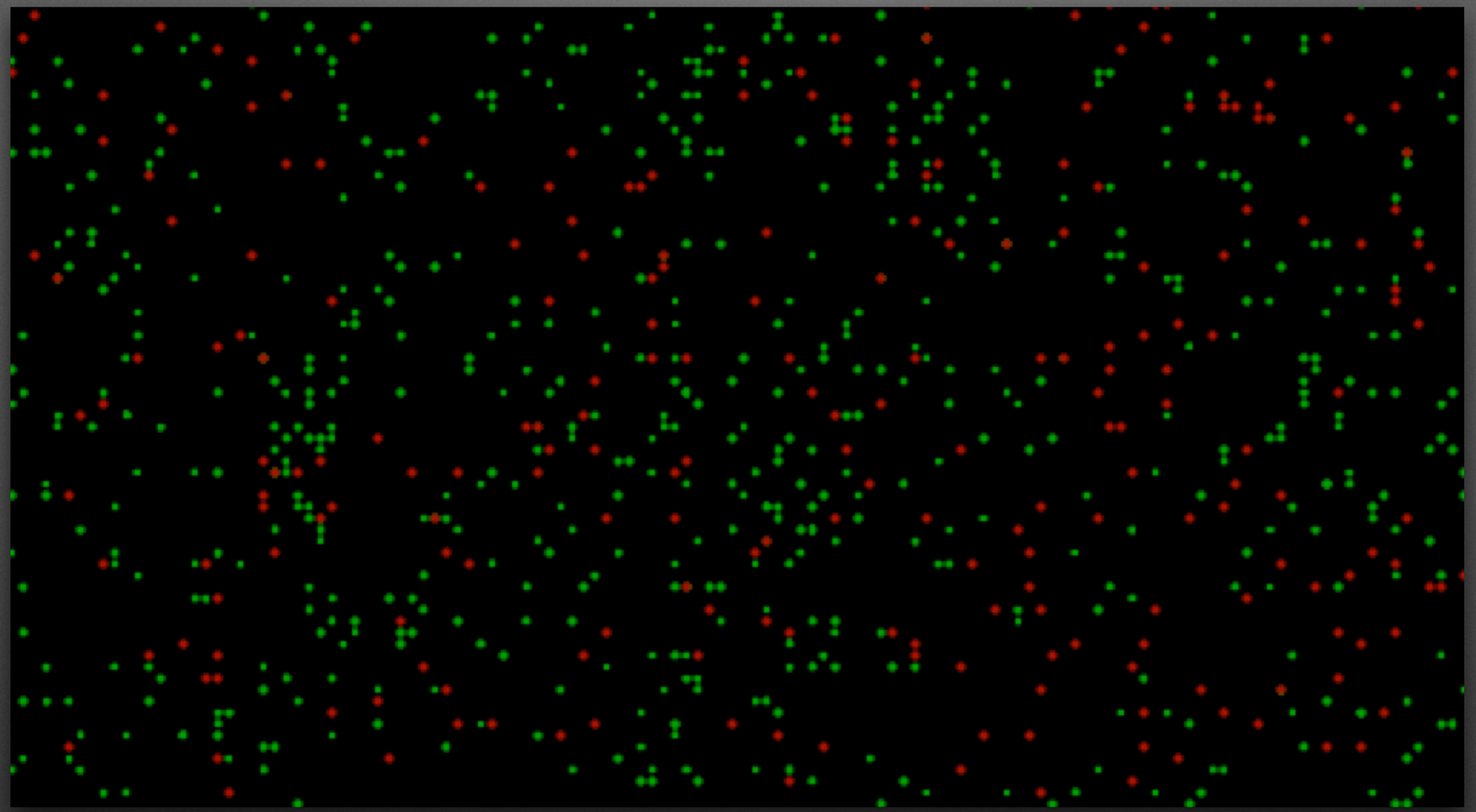
```
iex(2)> turtle = :sys.get_state(Turtles.Clock).turtles |> Enum.at(0)
#PID<0.380.0>
iex(3)> :sys.trace(turtle, true)
:ok
iex(4)> *DBG* <0.380.0> got call act from <0.154.0>
*DBG* <0.380.0> sent ok to <0.154.0>, new state
  #{'__struct__'=>'Elixir.Turtles.Turtle',energy=>8.000000000000075,…}
*DBG* <0.380.0> got call act from <0.154.0>
*DBG* <0.380.0> sent ok to <0.154.0>, new state
  #{'__struct__'=>'Elixir.Turtles.Turtle',energy=>7.900000000000075,…}
*DBG* <0.380.0> got call act from <0.154.0>
*DBG* <0.380.0> sent ok to <0.154.0>, new state
  #{'__struct__'=>'Elixir.Turtles.Turtle',energy=>7.800000000000075,…}
```

:sys.statistics/2

```
iex(1)> :sys.statistics(Turtles.World, true)
:ok
iex(2)> :sys.statistics(Turtles.World, :get)
{:ok,
 [start_time: {{2016, 8, 21}, {16, 32, 20}},
  current_time: {{2016, 8, 21}, {16, 32, 38}}, reductions: 144453,
  messages_in: 5504, messages_out: 0]}
iex(3)> :sys.statistics(Turtles.World, :get)
{:ok,
 [start_time: {{2016, 8, 21}, {16, 32, 20}},
  current_time: {{2016, 8, 21}, {16, 32, 47}}, reductions: 165599,
  messages_in: 6299, messages_out: 0]}
```

Logger is Your Friend

- Add log messages in key places
- This is a great way to watch interactions between processes
- Remember that you can change the current log level
  - To silence debug messages, for example



Turtle Ecology

Not-so-ordered Events

What I Got Right

- Frequently sending the full board to the drawing process is too much
- Instead the World records changes as they happen
  - Only new changes are sent to the drawing process

What I Got Wrong

- I chose a terrible data structure for these changes
- This allows the simulation to get ahead of the drawing
  - It manifested in drawing oddities, like multiplying turtles
- **Hack:** I had to have the drawing code trigger next steps after rendering

Changes

```
defmodule Turtles.World do
  defmodule Changes do
    defstruct clears: [ ], plants: [ ], turtles: [ ]

    def empty do
      %__MODULE__{ }
    end
  end

  defstruct size: nil,
            plants: MapSet.new,
            turtles: MapSet.new,
            changes: Changes.empty

  # …
end
```

Event: Eat Food

```
defmodule Turtles.World do
  # …

  defp eat(world = %__MODULE__{plants: plants, changes: changes}, location) do
    new_plants = MapSet.delete(plants, location)
    new_changes = %Changes{
      changes |
      clears: [location | changes.clears],
      turtles: [location | changes.turtles]
    }
    {:ate, %__MODULE__{world | plants: new_plants, changes: new_changes}}
  end
end
```

Event: Move

```
defmodule Turtles.World do
  # …

  defp handle_eat_or_move(
    world = %__MODULE__{plants: plants, turtles: turtles, changes: changes}, location, move_location
  ) do
    cond do
      MapSet.member?(plants, location) -> eat(world, location)
      not MapSet.member?(turtles, move_location) ->
        new_turtles = MapSet.delete(turtles, location) |> MapSet.put(move_location)
        new_changes = %Changes{
          changes |
          clears: [location | changes.clears],
          turtles: [move_location | changes.turtles]
        }
        {:moved, %__MODULE__{world | turtles: new_turtles, changes: new_changes}}
      true -> {:pass, world}
    end
  end

  # …
end
```

Drawing

```
defmodule Turtles.Painter do
  alias Canvas.GUI.Brush
  alias Turtles.{World, Clock}

  def paint(canvas, _width, _height, scale) do
    %{clears: clears, plants: plants, turtles: turtles} =
      World.changes(World)

    Enum.each(clears, fn location -> clear(canvas, scale, location) end)
    Enum.each(plants, fn location -> paint_plant(canvas, scale, location) end)
    Enum.each(turtles, fn location -> paint_turtle(canvas, scale, location) end)

    Clock.advance(Clock)
  end

  # …
end
```

The Problem

- Turtle at {0, 0} eats
  - clears: [{0, 0}], turtles: [{0, 0}]
- The same turtle moves before a redraw
  - clears: [{0, 0}, {0, 0}], turtles: [{0, 0}, {0, 1}]
- Now, when the draw happens, we have two turtles!

A Better Changelog

- [
  - {:clear, {0, 0}}
  - {:plant, {0, 0}}
  - {:turtle, {0, 1}}
-]

Strategies

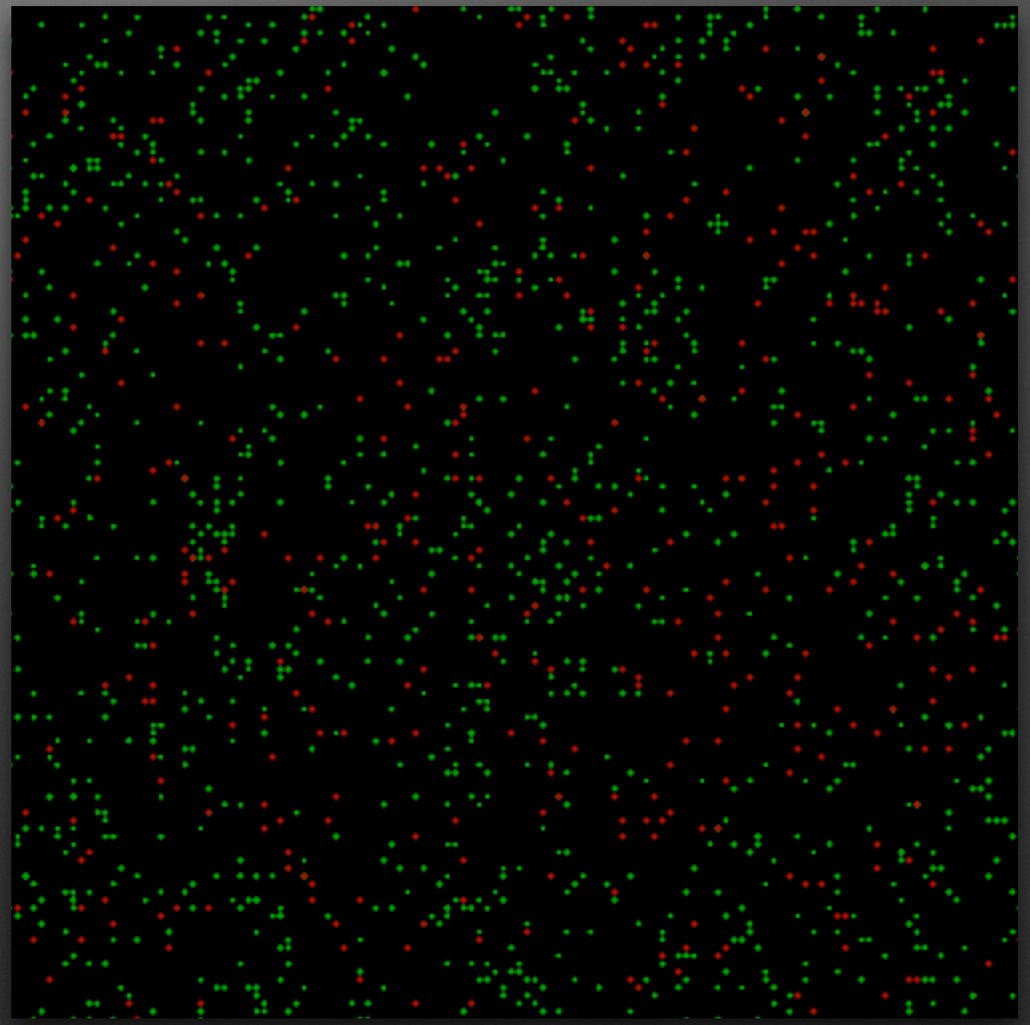
- Avoid sending big messages
- Dealing in change events is often helpful (think Event Sourcing)
- Finding the right data structure can be a big win

Turtle Ecology Fix Simulation

Exercise 4

Uncoupling Clock and Painter

- Have Painter stop advancing the Clock
- Make Clock setup its own ticks
- Fix the exposed drawing bugs
  - World’s changes need to be simplified as discussed
  - Painter has to be able to draw the new change format



Hints

- Follow the first six instructions in README.md exactly to expose the drawing bugs
- The callbacks in World are simpler with the new change format
- Only one function of Painter needs to change
- Ignore the tests for this exercise as fixing them is tedious

http://bit.ly/tefixsimzip

See instructions in README.md