Make It Crash!

Or, Adventures in OTP Process Supervision

Who Am I?

Me

- Platform engineer at FullContact
- Husband and father
- Enjoys ice cream
- Drives neat Prius
- Definitely not wearing that sweater in August
- **@dwilson** on DenverDevs
- **david-wilson** on GitHub

How Do I Know This?

The Dreaded Disclaimer

- No Elixir production experience
- Learned most concepts by reading and building demo
- Do have real-world distributed systems experience

What’s this all about, then?

What are Elixir, Erlang, and OTP?

Processes and Message Passing

Supervisors

“Let It Crash” Philosophy

Interactive demo

Elixir, BEAM, OTP?

Elixir: Erlang's Shiny Paint Job

—

- Developed at Ericsson for telecom systems
- Functional language
- Runs on BEAM VM
- Strong focus on share-nothing processes that communicate via message passing
- Long, successful production track record

But...

It looks a bit weird

```
listen(Socket) ->
     {ok, Active_socket} =
gen_tcp:accept(Socket),
     Handler =
spawn(?MODULE,handle_messages,[Active_socket]),
     ok =
gen_tcp:controlling_process(Active_socket,
Handler),
     listen(Socket).
```

The docs can be cryptic..

Exports

```
add_edge(G, V1, V2) -> edge() | {error, add_edge_err_rsn()}
add_edge(G, V1, V2, Label) -> edge() | {error,
add_edge_err_rsn()}
add_edge(G, E, V1, V2, Label) ->
            edge() | {error, add_edge_err_rsn()}
```

Types

```
G = graph()
E = edge()
V1 = V2 = vertex()
Label = label()
add_edge_err_rsn() =
    {bad_edge, Path :: [vertex()]} | {bad_vertex, V ::
vertex()}
```

`add_edge/5` creates (or modifies) edge `E` of digraph `G`, using `Label` as the (new) **`label`** of the edge. The edge is **`emanating`** from `V1` and **`incident`** on `V2`. Returns `E`.

`add_edge(G, V1, V2, Label)` is equivalent to `add_edge(G, E, V1, V2, Label)`, where `E` is a created edge. The created edge is represented by term `['$e' | N]`, where `N` is an integer >= 0.

`add_edge(G, V1, V2)` is equivalent to `add_edge(G, V1, V2, [])`.

If the edge would create a cycle in an **`acyclic digraph`**, `{error, {bad_edge, Path}}` is returned. If either of `V1` or `V2` is not a vertex of digraph `G`, `{error, {bad_vertex, V}}` is returned, V = `V1` or V = `V2`.

The tooling isn’t up to modern standards

—

Enter Elixir!

- Runs on Erlang VM (BEAM)
- Ruby-like syntax (or so I’m told)
- Strong focus on developer ergonomics
- Easy Erlang interop
- Strong community momentum

OTP: Where the Magic Happens

Open

Telecom

Platform

—

But I’m not doing telecom?

- OTP is all the batteries-included systems programming goodness in the Erlang ecosystem
- Contains the server and supervisor behavior’s we’ll be looking at today
- Many other tools like ETS (in-memory data store)

Processes

OMG LIVE CODING!

Supervision

Linkages and trapping exits

Live coding again? You know how much could go wrong? And for what purpose? Just because you think it's cool? Get over yourself and just make some more slides or something.

—

Supervisor basics

- Automatically restart exited processes
- Different restart strategies
  - Restart one
  - Restart all
  - Restart crashed and all newer
- Processes are started according to a “spec”
- Can set limits on restarts
- Supervisors are just another process, and can be supervised themselves

“Let it crash”

—

Why let it crash?

- Don’t program defensively
- Program intentionally
- Fail fast, and let supervisor restore system state
- Processes are isolated, so no shared state is lost in crash

Make it crash!

A request...

Audience Partitioning: The Hottest Topic in Distributed Systems

Please Silence Notifications

Text “New” to:

A-D: REDACTED

E-K: REDACTED

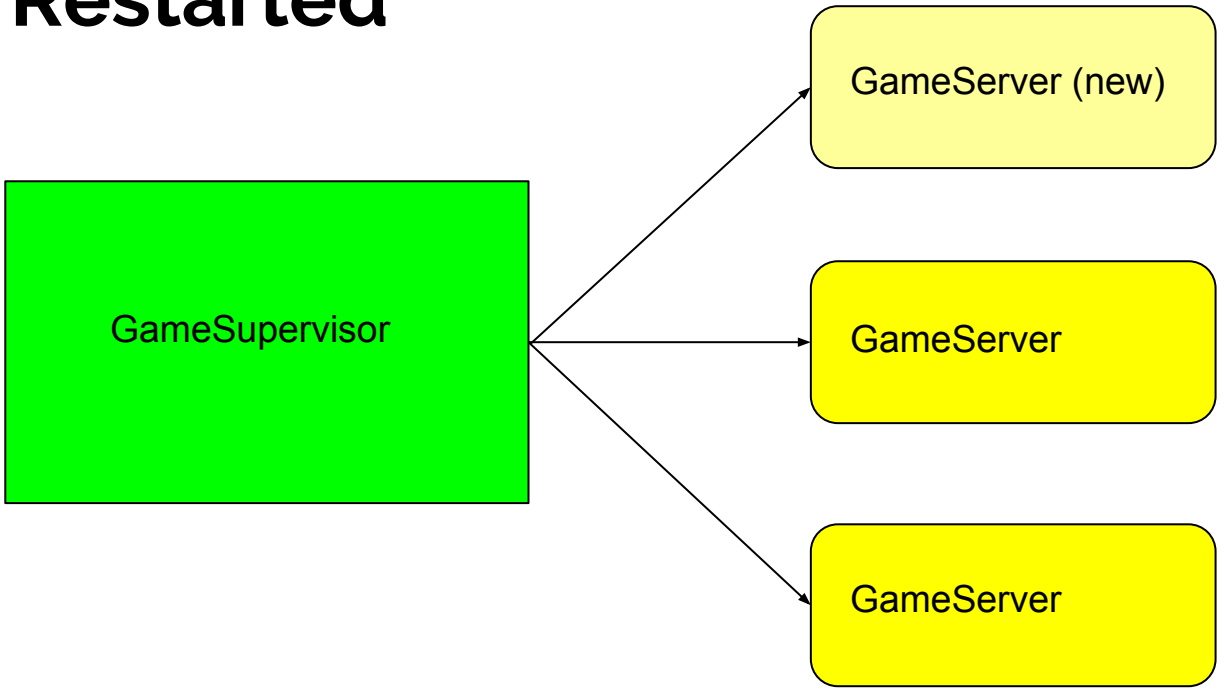
L-Q: REDACTED

R-Z: REDACTED

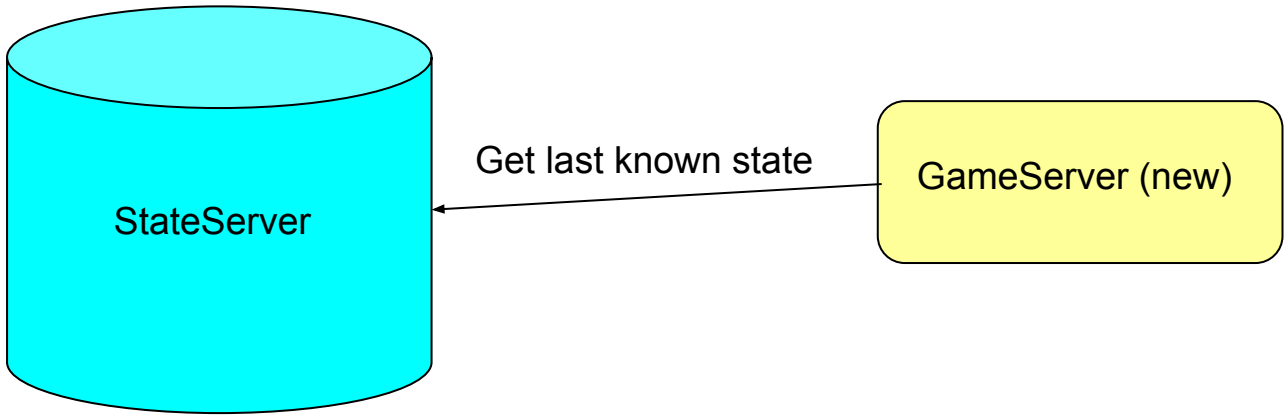
Text “Crash”

Anatomy of a crash: Process Exits

Anatomy of a crash: Process Restarted



Anatomy of a crash: Restore State



StateServer is just a GenServer with an simple API on top of a map

Now what?

—

Takeaways

- Isolated processes and message passing are a powerful abstraction
- Supervisors utilize process linkage behavior to help create self-healing systems
- Supervisors are supervisable processes themselves
- Design processes to be transient, keep core state safe and simple, build system state from there
- Let exceptional cases fail, log, and restart

—

Resources and learning more

- Learning Elixir: *Elixir in Action*, by Saša Jurić
- Learning OTP: *The Little Elixir & OTP Guidebook*, by Benjamin Tan Wei Hao
- Learning the “why”: http://ferd.ca/the-zen-of-erlang.html, Fred Herbet
- Production wisdom: https://www.erlang-in-anger.com/, Fred Herbet (dude knows his stuff)