Let it crash

...

Handling errors in concurrent programs

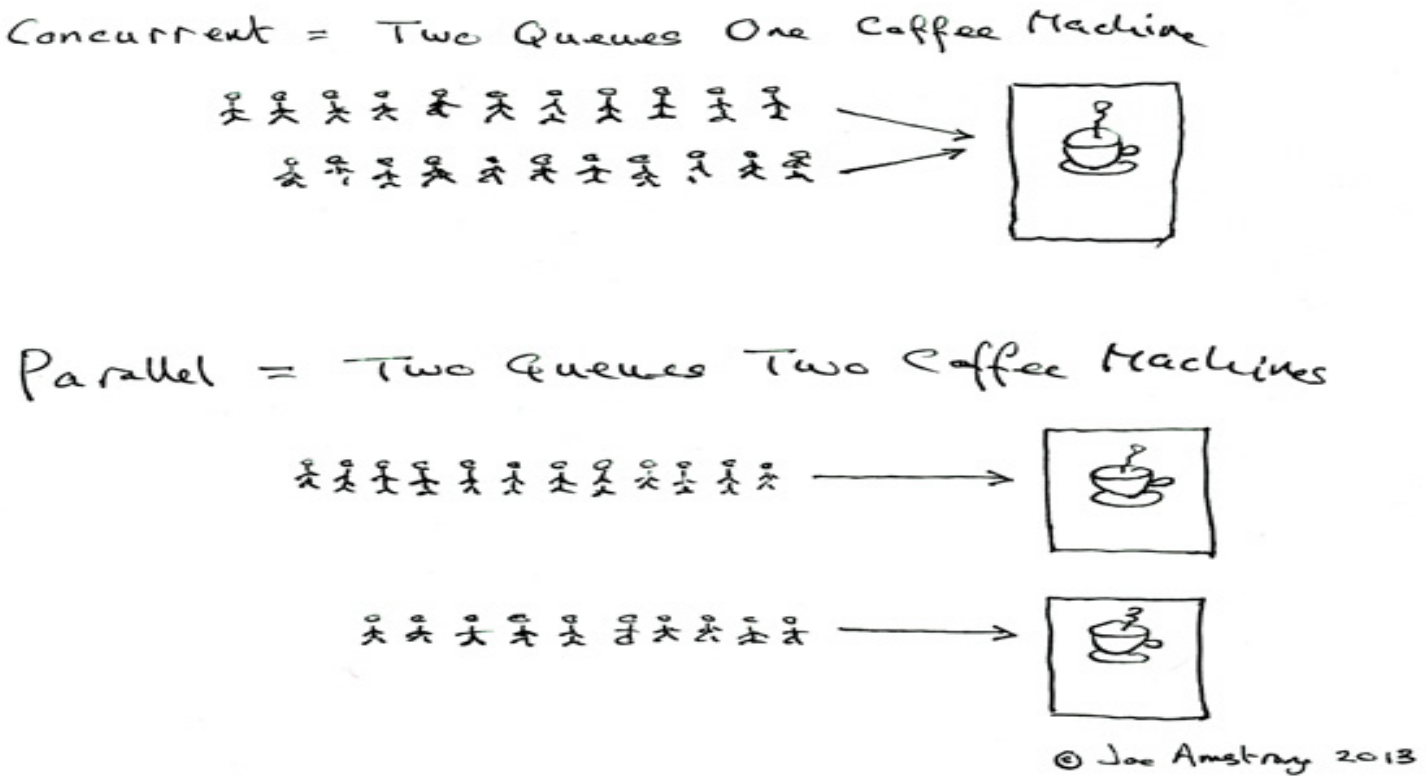
By Adrian Magdas

Co-Founder & Lead Developer @ Crafting Software Innovation
adrian@craftingsoftware.com
@zenCrafter

Agenda

1. Concurrency vs. parallelism
2. Actor model concurrency
3. Why Elixir & Erlang
4. Elixir agents
5. Process spawn and link
6. OTP patterns for handling agents: GenServer & Supervisor

Concurrency vs. parallelism



Actor model concurrency

The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation. In response to a message that it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify private state, but can only affect each other through messages (avoiding the need for any locks). (Wikipedia)

Why Erlang

- **We assume that errors will occur, processes will crash and machines WILL FAIL**
- **Focus on cure instead of prevention**
- **Error handling: remote detection and handling of errors**
- **Erlang and Elixir are impure, dynamically typed functional languages**
- **Both run on the BEAM (Erlang VM) which has built-in concurrency, distribution and scalability, actors are part of the language itself**

Why Erlang(cont)

- **SMP support since 2006**
- **Built-in fault tolerance: 99.99999% uptime**
- **Battle tested: ~50% of all mobile comms go through Erlang boxes**
- **Happy coincidence: the solution to telecom problems for reliability and scalability are a perfect match for the same problems the web and IOT are facing**
- **2 million web servers vs. 1 server with 2 million connections**

Why Elixir

- **Modern language with solid additions to Erlang: protocols, macros, sigils, pipe operator, great tooling**
- **Beauty and expressiveness of Python & Ruby, power and awesomeness of Erlang**
- **Compatible with existing Erlang infrastructure and libraries**

Elixir agents

- **Agents in Elixir are regular light weight processes**
- **In Elixir, all code runs inside processes. Processes are isolated from each other, run concurrent to one another and communicate via message passing.**
- **Messages are sent asynchronously and stored in mailboxes**
- **Used for keeping state or running asynchronous code**

Elixir tasks & agents

- **spawn & spawn_link**
- **Trapping exits**
- **Tasks**
- **Agents**

GenServer

A behaviour module for implementing the server of a client-server relation.

A GenServer is a process like any other Elixir process and it can be used to keep state, execute code asynchronously and so on. The advantage of using a generic server process (GenServer) implemented using this module is that it will have a standard set of interface functions and include functionality for tracing and error reporting. It will also fit into a supervision tree.

Supervisor

- **Process that supervises other processes(child processes)**
- **Fits into a supervision tree**
- **Multiple supervision strategies: :one_for_one, :one_for_all, :rest_for_one, :simple_one_for_one**

Take-aways

Actor based concurrency: light weight processes, no shared state between actors, process linking and monitoring, all the communication happens by sending messages

OTP: patterns for handling process creation, supervision strategies and fault-tolerance support

There are no silver bullets :)

Thank you

Develop with passion!

Resources

https://pragprog.com/book/elixir/programming-elixir

http://elixir-lang.org/

http://elixirschool.com/

http://learnyousomeerlang.com/what-is-otp

http://learnyousomeerlang.com/the-hitchhikers-guide-to-concurrency

https://joearms.github.io/2013/04/05/concurrent-and-parallel-programming.html