ERLANG FOR GO DEVELOPERS

CHRIS MOLOZIAN, HEROIC LABS

WHO AM I

- A DATABASE AND DISTRIBUTED SYSTEMS ENGINEER
- WORKED ON RIAK. A DISTRIBUTED NOSQL DATABASE IN ERLANG
- STARTED HEROIC LABS
- BUILD CORE INFRASTRUCTURE FOR THE GAMES INDUSTRY
- NAKAMA. AN OPEN-SOURCE GAME SERVER WRITTEN IN GO

TALK TOPIC

- DISCUSS SIMILARITIES AND DIFFERENCES
- LEARN SOME GREAT PRINCIPLES TO USE WITH YOUR GO CODE
- SEE TECHNIQUES IN ERLANG USED TO MANAGE PROCESSES
- WILL NOT CONVINCE YOU TO WRITE ERLANG 😄

SIMILARITIES

- LIGHTWEIGHT CONCURRENT RUNTIMES
- SIMPLE API TO CREATE PROCESSES
- "BATTERIES INCLUDED" STANDARD LIBRARIES
- CONCURRENT BY DESIGN

DIFFERENCES

- PROLOG-INSPIRED SYNTAX
- NO SIMPLE BINARY BUILDS. REQUIRES BUILD TOOLS
- DEPLOYS ONTO A VIRTUAL MACHINE (BEAM)
- SUPPORTS DISTRIBUTED COMMUNICATION

ERLANG RUNTIME

- AN OPEN SECRET ABOUT WHAT MAKES ERLANG UNIQUE
- PROVIDES "BEHAVIOURS" TO SOLVE MANY DESIGN PROBLEMS
- LIKE THE GOF DESIGN PATTERNS (ADAPTER, SINGLETON, ETC.)
- DIFFICULT TO SHARE STATE - MUST USE MESSAGE PASSING

GOROUTINE

```
go func() {
    fmt.Println("Hello world!")
}()
```

PROCESS

```
Pid = spawn(fun() ->
    io:fwrite("hello, world\n")
end).
io:format("~p~n", [Pid]). % <0.63.0>
```

ANATOMY OF A PID

<A.B.C> E.G. <2265.10.0>

"A" - THE NODE NUMBER. WHERE 0 INDICATES LOCAL NODE.
"B" - THE FIRST 15 BITS OF THE PROCESS NUMBER LOOKUP.
"C" - A WRAP COUNTER TO INDICATE REUSED PIDS.

PIDS ARE KNOWN TO THE ERLANG DISTRIBUTED RUNTIME

ERLANG PROCESS

- EACH PROCESS HAS A MAILBOX (A PRIVATE BUFFER)
  - A PROCESS IS LIKE A GOROUTINE + CHANNEL
  - THIS COMPOSITION CREATES AN "ACTOR"
- MESSAGES ARE SERIALIZED THROUGH THE MAILBOX

SEND MESSAGES

```
Pid = spawn(fun() ->
    receive Num ->
        io:format("~p~n", [Num + 1])
    end
end).

Pid ! 8.
```

DISTRIBUTED ERLANG 1

```
$ erl -name "server1@127.0.0.1" -cookie "somecookie"
(server1@127.0.0.1)1> node().
'server1@127.0.0.1'
(server1@127.0.0.1)2> register(helloworld_actor, self()).
true
(server1@127.0.0.1)3> flush().
Shell got "Hello world"
ok

$ erl -name "server2@127.0.0.1" -cookie "somecookie"
(server2@127.0.0.1)1> node().
'server2@127.0.0.1'
(server2@127.0.0.1)2> {helloworld_actor, 'server1@127.0.0.1'} ! "Hello world".
"Hello world"
(server2@127.0.0.1)3>
```

DISTRIBUTED ERLANG 2

```
$ erl -name "server1@127.0.0.1" -cookie "somecookie"
(server1@127.0.0.1)1> node().
'server1@127.0.0.1'
(server1@127.0.0.1)2> register(helloworld_actor, self()).
true
(server1@127.0.0.1)3> flush().
Shell got "Hello world"
ok

$ erl -name "server2@127.0.0.1" -cookie "somecookie"
(server2@127.0.0.1)1> node().
'server2@127.0.0.1'
(server2@127.0.0.1)2> {helloworld_actor, 'server1@127.0.0.1'} ! "Hello world".
"Hello world"
(server2@127.0.0.1)3>
```

ERLANG PORT MAPPER DAEMON

```
$ epmd -debug
epmd: epmd running - daemon = 0
epmd: ** got ALIVE2_REQ
epmd: registering 'server1:2', port 55770
epmd: type 77 proto 0 highvsn 5 lowvsn 5
epmd: ** sent ALIVE2_RESP for "server1"
epmd: ** got ALIVE2_REQ
epmd: registering 'server2:1', port 55775
epmd: type 77 proto 0 highvsn 5 lowvsn 5
epmd: ** sent ALIVE2_RESP for "server2"

$ epmd -names
epmd: up and running on port 4369 with data:
name server2 at port 55775
name server1 at port 55770
```

MONITOR AND LINK

- PROCESSES CAN HAVE THEIR LIFETIME TIED TOGETHER
- MONITOR IS A RELAXED WAY FOR ONE PROCESS TO BE SIGNALED ABOUT ANOTHER
- LINK ALLOWS ONE PROCESS TO BE SUPERVISED BY ANOTHER
- TREES OF PROCESSES CAN HAVE THEIR LIFETIMES TIED TOGETHER

MONITOR (UNIDIRECTIONAL)

```
Pid = spawn(fun() ->
    timer:sleep(20000)
end).
Ref = erlang:monitor(process, Pid).
exit(Pid, kill).
flush().
% {'DOWN',#Ref<0.1686156355.2499018753.122188>,process,
%                  <0.62.0>, killed}
```

LINK (BIDIRECTIONAL)

```
Pid1 = spawn(fun() ->
    receive stop ->
        io:format("P1 ended~n")
    end
end).
Pid2 = spawn(fun() ->
    link(Pid1),
    receive stop ->
        io:format("P2 ended~n")
    end
end).
{erlang:is_process_alive(Pid1), erlang:is_process_alive(Pid2)}.
% {true,true}
exit(Pid1, kill).
{erlang:is_process_alive(Pid1), erlang:is_process_alive(Pid2)}.
% {false,false}
```

SUPERVISORS

"LET IT CRASH!"

- SUPERVISORS BUILD ON THE CONCEPT OF LINKED PROCESSES
- PROVIDE RESTART STRATEGIES FOR WHEN PROCESSES CRASH
  - THE PROCESSES SUPERVISED ARE CALLED "WORKERS"
  - AN ERLANG APPLICATION IS A TREE OF SUPERVISORS

SUMMARY

- PIDS MAKE IT POSSIBLE TO ADDRESS PROCESSES
  - PIDS CAN BE REGISTERED WITH NAMES
- PROCESSES CAN BE MONITORED/LINKED TO OTHER PROCESSES
  - SEND MESSAGES TO GIVE A PROCESS WORK/COMPUTATION
    - SEND MESSAGES TO PIDS ACROSS NODES!

WHERE TO LEARN MORE

- LEARN YOU SOME ERLANG
  HTTP://LEARNYOUSOMEERLANG.COM
- ERLANG IN ANGER
  HTTPS://WWW.ERLANG-IN-ANGER.COM
- ERLANG PROGRAMMING LANGUAGE
  HTTPS://WWW.ERLANG.ORG

INTERESTING PROJECTS

- AN IMPLEMENTATION OF SUPERVISION TREES IN GO
  HTTPS://GITHUB.COM/UCIRELLO/SUPERVISOR
- ACCESS TO A GOROUTINE'S ID
  HTTPS://GITHUB.COM/DAVECHENEY/JUNK/TREE/MASTER/ID
- IMPLEMENT AN ERLANG/OTP NODE IN GO
  HTTPS://GITHUB.COM/GOERLANG/NODE

QUESTIONS?