A BRIEF ERLANG (HI)STORY

A BRIEF ERLANG (HI)STORY

Processes crash, systems do not

Created by Angel J Alvarez Miguel/ @AJAlvarezMiguel

ABOUT ME...

Sysadmin over 15 years.. Now Full Stack dude

I met Erlang in 2008, in love since then..

Currently bringing the bacon home with Classic ASP :(

DISCLAIMER

Most info gathered from many sources: YMMV

WHY ERLANG?

Ruby considered Harmful..

THE ERLANG HISTORY..

Ericsson made money selling SPOTS, (SPC POTS)

SPOTS were hard to program

They tried several langs..

Eventually..

They invented Erlang!

1974 THE ERICSSON BUSINESS

1974-1982 THE ERICSSON BUSINESS

- The AXE Switch starts selling
- Its controlled by a propietary OS / hardware (The APZ processor)
- Programmed in Plex a Ericsson language tied to the AXE hardware
- Ericsson earned millions with the AXE

1982 PLEX HORROR..

PLEX is a rather low level language (like Basic) but:

- Safe pointers (via MMU)
- Fine grained massive concurrency
- Finite state machines supported in microcode
- Code blocks isolation with hot updates
- Advanced tracing ability at runtime
- Restart mechanisms (small and large restart)

1982 PLEX HORROR II.

AXE systems were costly and hard to develop..

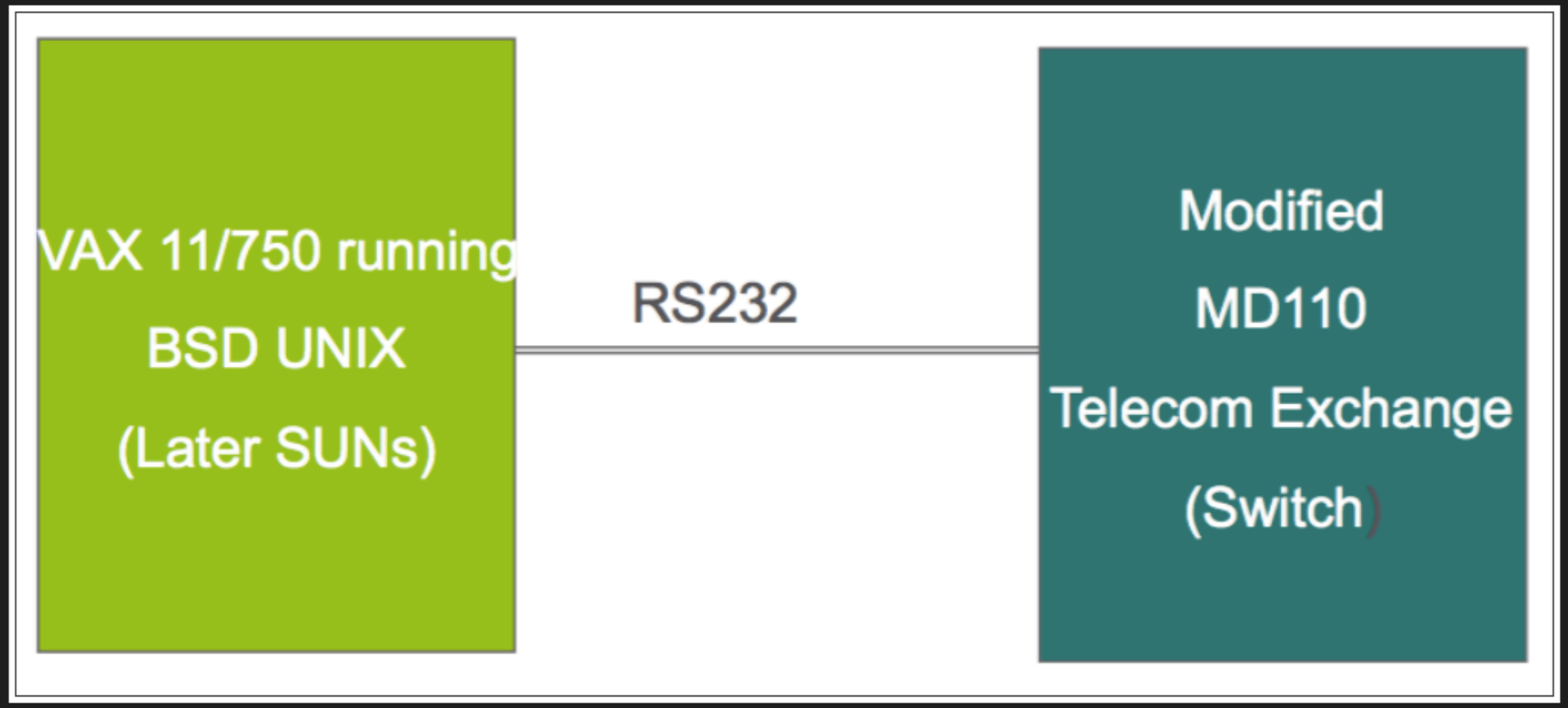
- PLEX is time consuming and too low level
- Efforts to turn AXE into a multi-processor system failed
- The AXE Hardware was also very costly

1985 THE LANGUAGES LAB..

Goal: to solve "The Ericsson Software problem"

- 1 Handling a very large number of concurrent activities
- 2 Timely performed actions
- 3 Distributed systems (over several computers)
- 4 Interaction with hardware
- 5 Very large systems with stringent quality
- 6 Operation over years. Non-Stop maintenance
- 7 Fault tolerance (hardware and software errors)

1985 THE EXPERIMENTS



1985 THE EXPERIMENTS II

They tried many concepts..

- SPOTS (SPC for POTS)
- Using Ada, Euclid, PFL, LPL0 etc...
- Hosted on a Vax11/750 running Unix BSD 4.2
- The Unix box was controlling a MD110 PABX

1987 MORE EXPERIMENTS..

They evolve the initial concepts..

- DOTS (Distributed SPOTS)
- LOTS (Lots of DOTS)
- And tried also SmallTalk.. ..and Prolog
- POTS algebras resembled Prolog..
- Erlang was Born! (and was Prolog Based)

1987 POTS ALGEBRAS

```
idle(N)    means the subscriber N is idle
on(N)      means subscribed N in on hook
...
```

And operators:

```
+t(A, dial_tone) means add a dial tone to A
```

Finally rules:

```
process(A, f) :- on(A), idle(A), +t(A,dial-tone),
                 +d(A, []), -idle(A), +of(A)
```

This had the following declarative reading:

```
process(A, f)      To process an off hook signal from
                   a subscriber A
:-                 then
on(A)              If subscriber A is on-hook
,                  and
idle(A)            If subscriber A is idle
,                  and
+t(A, dial_tone)   send a dial tone to A
,                  and
+d(A, [])          set the set of dialled digits to []
,                  and
-idle(A)           retract the idle state
```

1987 ACS/DUNDER

FIRST ERLANG OPPORTUNITY..

- Erlang was very tied to a Prolog System
- Erlang keeps growing: Mailboxes, links, Error Handling
- ACS/Dunder delivers results: we need a 70 times faster Erlang!!

1988 ELLEMTEL

TIME TO BUILD SOMETHING REAL..

- Mobily Server The Next generation
- Compilation of Erlang to Strand (Fiasco)
- Erlang: Prolog with Concurrency and Functional Features

1988 HOW ERLANG LOOKED LIKE

```
# wait_first_digit(A) ->
    receive 10 {
        A ? digit(D) =>
            stop_tone(A),
            received_digit(A,[],D);
        A ? on_hook =>
            stop_tone(A),
            idle(A);
        timeout =>
            stop_tone(A),
            wait_clear(A);
        Other =>
            wait_first_digit(A)
    }.
```

Erlang in 1988

1988-1989 ACS/DUNDER FINAL RESULTS..

Erlang is a good approach to the LOTS problem

- It delivers from 3 to 25 times more productivity than Plex
- Yet, it needs to be faster (x280 faster!!)
- ACS turns into the Mobility Server
- Erlang delivered to BellCore
- JAM: the first emulator is built. First in prolog then in C

1988-1989 THE JAM EMULATOR

```
fac(0) -> 1;
fac(N) -> N * fac(N-1)

{info, fac, 1}
 {try_me_else, label1}
        {arg, 0}
        {getInt, 0}
        {pushInt, 1}
        ret
label1: try_me_else_fail
        {arg, 0}
        dup
        {pushInt, 1}
        minus
        {callLocal, fac, 1}
        times
        ret
```

1990-1992 ERLANG IS THE PRODUCT

Erlang keeps growing and getting features..

- It gets garbage collection per process
- It gets distribution via TCP/IP
- Lost of ports: VxWorks, Windows, QNX..
- Plan for an open source relase

1993 ERLANG IS THE PRODUCT II

Erlang keeps growing and getting features..

- 1992-1993 First Erlang Book (Erlang no longer secret)
- Turbo Erlang (a.k.a The BEAM): Cross compilation to C and then to ASM
- Erlang Systems AB: The Uppsala boys

1995-1997 THE ERLANG TAKE OVER

The AXE-N fiasco (C++)

- After Failure AXE-N turned to Erlang (60 people)
- AXE-N turned into AXD family of ATM switches
- Erlang was now a functional language with concurreny
- The OTP group was built. Many features added Record, Binaries, behaviours, Mnesia/ETS

1998 ERLANG BANNED!!

Everything was going well..

- First GPRS Erlang demo at GSM WC, CeBIT
- Erlang got banned from projects at Ericsson Radio Systems
- Erlang got open sourced
- Erlang core guys left Ericsson

2000-2001

- Bluetail adquired by Alteon Web systems
- Alteon adquired by Nortel Networks
- IT Crash 2001: Some Erlang guys left Nortel

2002-2005 SECONG GENERATION ERLANG COMPANIES

- 2002 Joe Armstrong enters SICS
- Nortel guys left and founded Tail-f, Kreditor, Synapse
- In the UK Erlang Consulting gets founded

2006 ERLANG GOES MULTICORE!!

- The OTP group releases Erlang for SMP
- People start asking how to parallelize programs..
- Some popular examples on new hardware: SUN T2000 Servers..

2006-2016..

This is the easy part..

- FaceBook tries Erlang
- WhatsApp uses Erlang
- People get to know Erlang
- Companies start using Erlang..

SO WHAT IS ERLANG?

Erlang is Erlang/OTP: The product

- The Language
- The OTP frameworks and libraries
- The BEAM emulator

THE LANGUAGE

ERLANG FEATURES I

- High level language (Derived from Prolog)
- Functional paradigm (not pure)
- Leverages the actor model
- Robust error handling

LANGUAGE FEATURES II

- "First class" Functions
- Pattern Matching
- List comprehensions
- Binaries: The telco legacy
- Lighweight processes
- Modules defining sort of namespaces

PATTERN MATCHING

```
Tuple = {foo, 42, "hello"},
{X, Y, Z} = Tuple,

List = [5, 5, 5, 4, 3, 2, 1],
[A, A | Rest] = List,

Struct = {foo, [5,6,7,8], {17, 42}},
{foo, [A|Tail], {N, Y}} = Struct
```

```
area({square, Side})   ->    Side * Side ;
area({circle, Radius}) ->    math:pi() * Radius * Radius.
```

LIST COMPREHENSIONS

```
%% map
[f(X) || X <- List]

%% filter
[X || X <- Xs, X > 0]
```

```
%% generate all permutations of a list
perms([]) -> [[]];
perms(L) ->
   [[X|T] || X <- L, T <- perms(L -- [X])].
```

BINARIES AND PATTERN MATCHING

```
decode(<< SourcePort:16, DestinationPort:16,
          SequenceNumber:32,
          AckNumber:32,
          DataOffset:4, _Reserved:4, Flags:8, WindowSize:16,
          Checksum:16, UrgentPointer:16,
          Payload/binary>>) when DataOffset>4 ...
```

```
case <<8:4, 42:6>> of
  <<A:3/integer, B:A/bits, C/bits>> -> {A,B,C}
end
```

PROCESSES

```
-module(area_server0).
-export([loop/0]).

loop() ->
    receive
        {rectangle, Width, Ht} ->
            io:format("Area of rectangle is ~p~n",[Width * Ht]),
            loop();
        {circle, R} ->
            io:format("Area of circle is ~p~n", [3.14159 * R * R]),
            loop();
        Other ->
            io:format("I don't know what the area of a ~p is ~n",[Other]),
            loop()
    end.
```

We can create a process that evaluates loop/0 in the shell:

```
1> Pid = spawn(fun area_server0:loop/0).
<0.36.0>
2> Pid ! {rectangle, 6, 10}.
Area of rectangle is 60
{rectangle,6,10}
3> Pid ! {circle, 23}.
Area of circle is 1661.90
{circle,23}
4> Pid ! {triangle,2,4,5}.
I don't know what the area of a {triangle,2,4,5} is
{triangle,2,4,5}
```

PROCESSES

```
-module(area_server0).
-export([loop/0]).

loop() ->
    receive
        {rectangle, Width, Ht} ->
            io:format("Area of rectangle is ~p~n",[Width * Ht]),
            loop();
        {circle, R} ->
            io:format("Area of circle is ~p~n", [3.14159 * R * R]),
            loop();
        Other ->
            io:format("I don't know what the area of a ~p is ~n",[Other]),
            loop()
    end.
```

We can create a process that evaluates loop/0 in the shell:

```
1> Pid = spawn(fun area_server0:loop/0).
<0.36.0>
2> Pid ! {rectangle, 6, 10}.
Area of rectangle is 60
{rectangle,6,10}
3> Pid ! {circle, 23}.
Area of circle is 1661.90
{circle,23}
4> Pid ! {triangle,2,4,5}.
I don't know what the area of a {triangle,2,4,5} is
{triangle,2,4,5}
```

THE OTP LIBRARIES

- Standard Libraries
- System processes
- Distributed Erlang

THE OTP PRINCIPLES

- OTP Behaviours
- Supervision Trees: "Let it Crash"
- Applications
- Releases

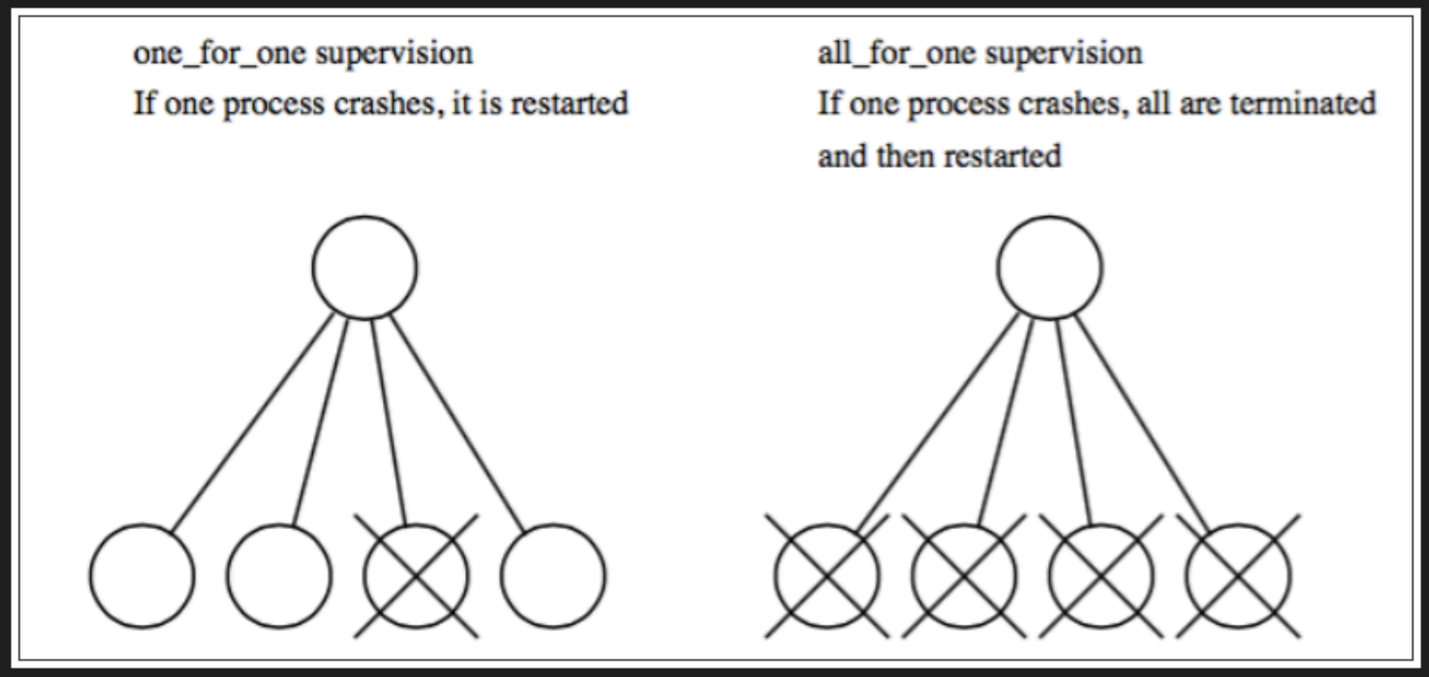
OTP BEHAVIOURS

- Generic Servers: The gen_server
- Finite State Machines: The gen_fsm
- Event Handlers: The gen_event
- Applications

OTP BEHAVIOURS EXAMPLE

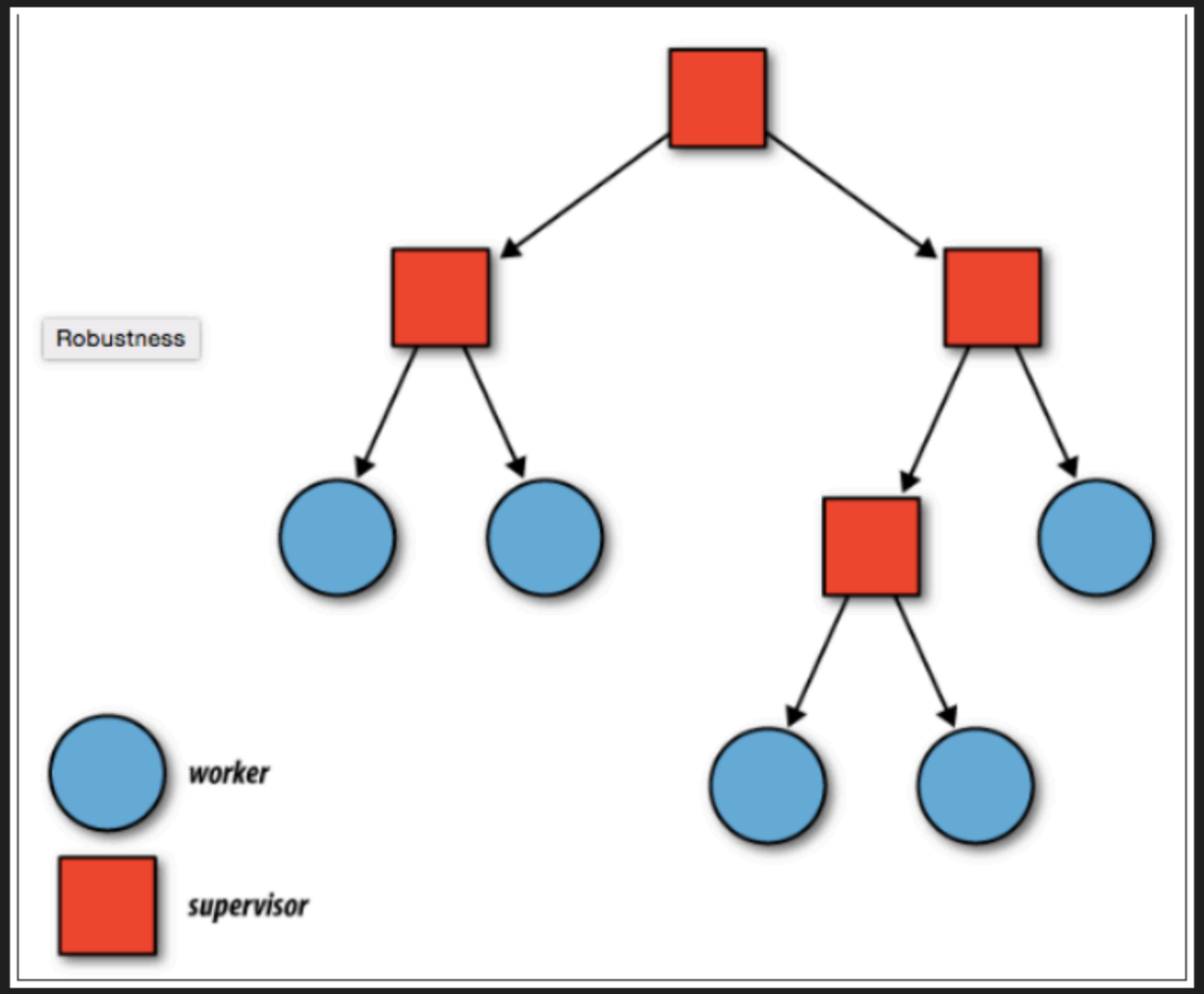
```erlang
-module().
%% gen_server_mini_template

-behaviour(gen_server).
-export([start_link/0]).
%% gen_server callbacks
-export([init/1, handle_call/3, handle_cast/2, handle_info/2,
         terminate/2, code_change/3]).

start_link() -> gen_server:start_link({local, ?SERVER}, ?MODULE, [], []).

init([]) -> {ok, State}.

handle_call(_Request, _From, State) -> {reply, Reply, State}.
handle_cast(_Msg, State) -> {noreply, State}.
handle_info(_Info, State) -> {noreply, State}.
terminate(_Reason, _State) -> ok.
code_change(_OldVsn, State, Extra) -> {ok, State}.
```

SUPERVISION TREES



- a.k.a "Let it crash"
- Monitors and Links

SUPERVISORS



- several strategies: one_for_one, one_for_all..

APPLICATIONS

- Library Applications
- Standard Aplications
- System Applications: Kernel and company..
- Application Packaging
- Application Life Cycle

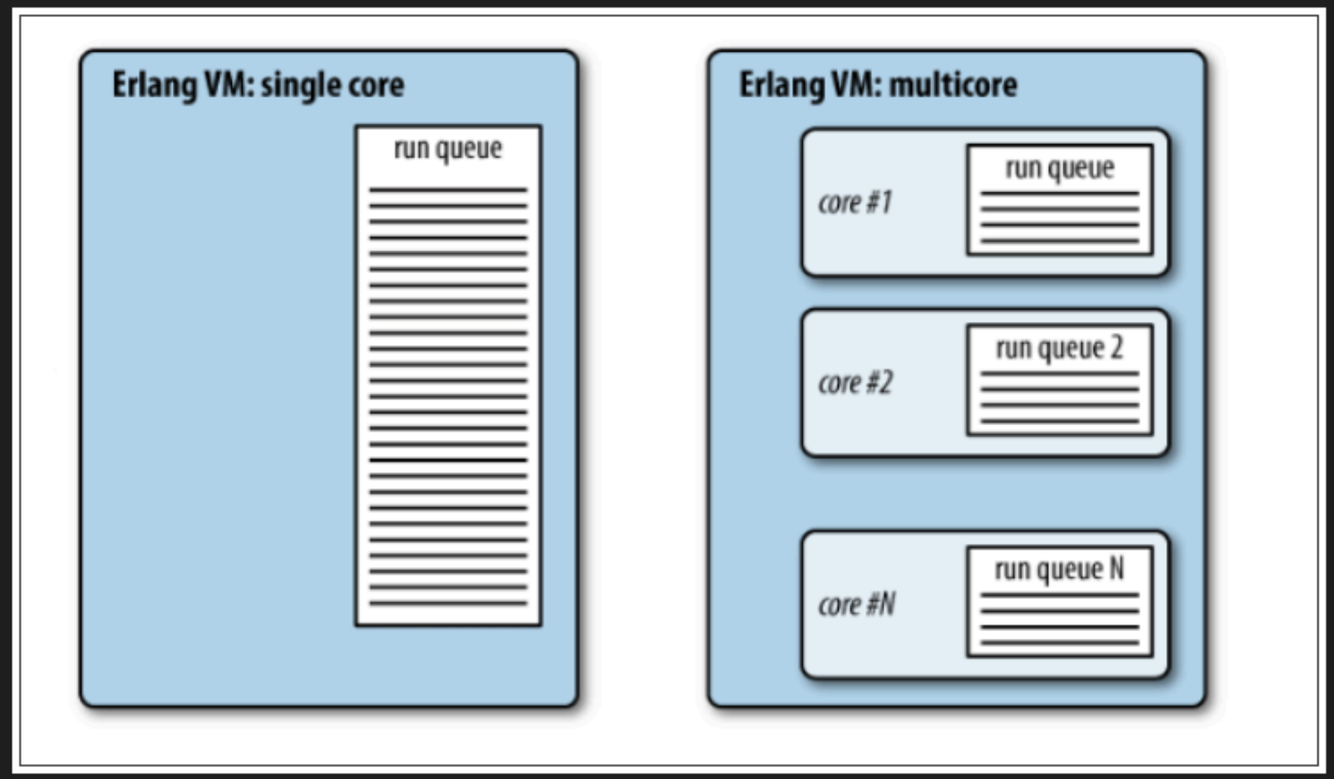
RELEASES

- Hot code reloading
- Release handling: going up (or going down)
- Upgrading the whole system, not just an application

THE EMULATOR

- a.k.a The BEAM machine
- Interprets Erlang code and manages IO
- Provides Erlang Soft Real-time semantics

THE BEAM MACHINE



- Interprets bytecode for every erlang process
- More than one Sheduler on SMP systems

IO: LOTS OF IT!!

THE KEY TO MASSIVE CONCURRENCY

- Ports
- Drivers
- NIFS
- Dirty Schedulers

WHO USES ERLANG?

- Ericsson
- Telcos (ericsson clients): ATM, GSM, etc..
- Banking and Trading Companies
- Online Gaming, Online Bets
- E-Shop, and Social Networks

ERICCSON

- Ericsson GPRS
- Ericsson 3G systems
- Ericsson 4G (LTE)

BANKING AND TRADING COMPANIES

- Goldman Sachs
- Goldman Sachs, of course
- Did I mention, Goldman Sachs??

ONLINE GAMING AND BETTING

- Bet365 (InPlay service)
- William Hill
- Call Of Duty (core server)
- Vendetta Online (Naos game server)
- WoT (message delivery and communication between game players)
- LoL (jabberd subsystem)

SOCIAL NETWORKS, MESSAGING

- Amazon (SimpleDB)
- Rakuten (distributed filesystem)
- DNSSimple
- Facebook (chat)
- Tuenti (chat)
- Whisper
- WhatsApp (core servers)

POPULAR SOFTWARE PROJECTS

- Chef (Configuration Management), The core API server
- SimpleDB, (Amazon Web Services)
- ejabberd, (XMPP instant messaging server)
- MongooseIM, a massively scalable XMPP platform
- RabbitMQ: An AMQP Message Broker

THE COMMUNITY

- The home page: www.erlang.org
- The Erlang Mail List: Erlang questions

THE BOOKS..

- Programming Erlang (The Pragmatic Programmers, Armstrong)
- Learn you Erlang (No Starch, Hébert)
- Building Web applications with Erlang (Oreilly, Kerson)
- Designing for Scalability with Erlang/OTP (Oreilly, Cessarinni & Vinosky)

THANKS!

We take no refunds..