Erlang

Aspectos Básicos

¿Qué es Erlang/OTP?

Erlang is a programming language used to build **massively scalable soft real-time systems** with requirements on **high availability**. Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for **concurrency**, **distribution** and **fault tolerance**. Originally developed at **Ericsson**, it was released as **open source** in 1998.

OTP is a set of Erlang **libraries and design principles** providing middle-ware to develop these systems.

Lenguaje Funcional

- No OO
- Funciones agrupadas en módulos
- Estado inmutable
- Funciones sin side-effects (casi)

Variables Invariables

```
Eshell V5.7.5  (abort with ^G)
1> A = 1.
1
2> A = 2.
** exception error: no match of right hand side value 2
```

Pattern Matching (I)

```
Eshell V5.7.5  (abort with ^G)
1> [ H | T ] = ["uno", "dos", "tres"].
["uno","dos","tres"]
2> H.
"uno"
3> T.
["dos","tres"]
```

Pattern Matching (II)

```
Eshell V5.7.5  (abort with ^G)
1> {_, S, [H |T]} = {hola, "abc", [1,2,3,4]}.
{hola,"abc",[1,2,3,4]}
2> S.
"abc"
3> H.
1
4> T.
[2,3,4]
```

Pattern Matching (III)

```
-module(greet).
-export([greet/2]).

greet(male, Name) ->
    io:format("Hello, Mr. ~s!~n", [Name]);
greet(female, Name) ->
    io:format("Hello, Mrs. ~s!~n", [Name]);
greet(_, Name) ->
    io:format("Hello, ~s!~n", [Name]).
```

Pattern Matching (y IV)

```
Eshell V5.7.5  (abort with ^G)
1> greet:greet(male, "Bob").
Hello, Mr. Bob!
ok
2> greet:greet(female, "Alice").
Hello, Mrs. Alice!
ok
3> greet:greet(xxx, "Alice").
Hello, Alice!
ok
```

Recursividad

```
-module(recursion).
-export([map/2]).

map(F, List) ->
    map(F, [], List).

map(_F, Result, []) ->
    Result;
map(F, Result, [Head | Tail]) ->
    map(F, Result ++ [ F(Head) ], Tail).
```

```
Eshell V5.7.5  (abort with ^G)
1> recursion:map(fun(E) -> 2*E end, [1,2,3,4,5]).
[2,4,6,8,10]
```

Higher Order Functions

```
2> PorDos = fun(E) -> 2 * E end.
#Fun<erl_eval.6.13229925>
3> recursion:map(PorDos, [1,2,3,4,5]).
[2,4,6,8,10]
```

List Comprehensions

```
Eshell V5.7.5  (abort with ^G)
1> [ 2 * X || X <- [1,2,3,4,5] ].
[2,4,6,8,10]
```

Bit Syntax (I)

```
Eshell V5.7.5  (abort with ^G)
1> Color = 16#F09A29.
15768105
2> <<R:8, G:8, B:8>> = <<Color:24>>.
<<240,154,41>>
3> R.
240
4> G.
154
5> B.
41
```

Bit Syntax (II)

TCP Header

Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Source port																Destination port															
32	Sequence number																															
64	Acknowledgment number (if ACK set)																															
96	Data offset				Reserved			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
128	Checksum																Urgent pointer (if URG set)															
160 ...	Options (if Data Offset > 5) ...																								padding							

```
<<SourcePort:16, DestinationPort:16,
  SequenceNumber:32,
  AckNumber:32,
  DataOffset:4, _Reserved:4, Flags:8, WindowSize:16,
  CheckSum: 16, UrgentPointer:16,
  Payload/binary>> = SomeBinary.
```

Otras Características

- Hot Code Loading
- Operaciones IO asíncronas

Erlang

Concurrencia

Concurrencia VS Paralelismo

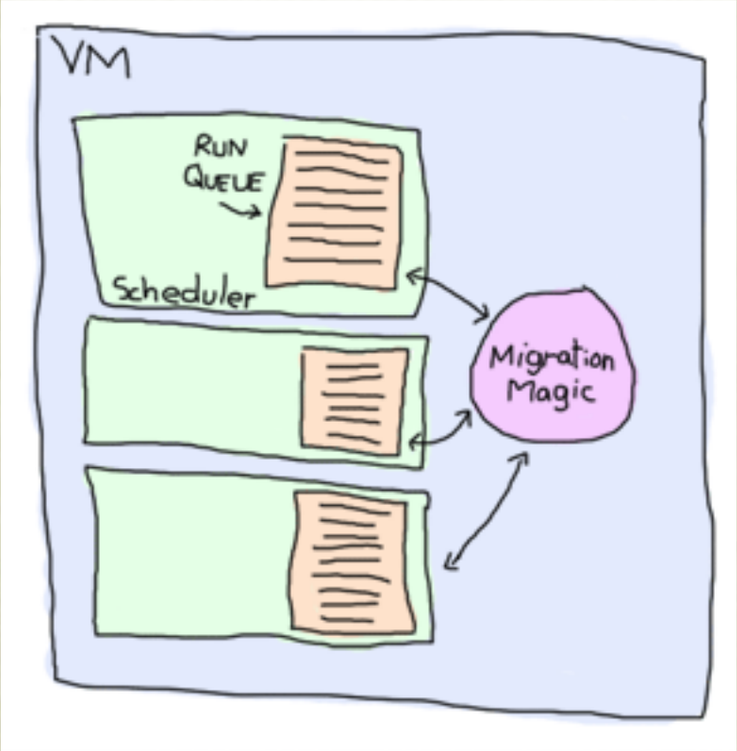
Escalabilidad

Tolerancia a fallos (Let it Crash)

Modelo de Actores

- Procesos ligeros
- Memoria independiente
- Envío de mensajes asíncronos
- Operaciones IO asíncronas

Ejecución de procesos



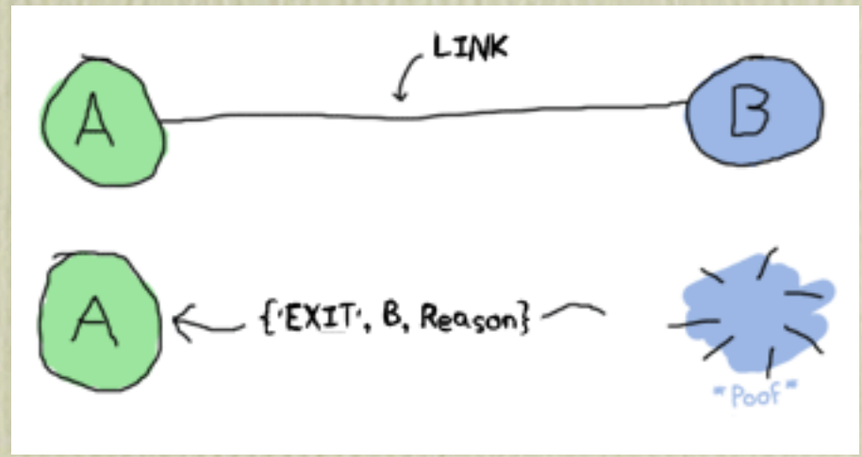
Ejemplo de Proceso (I)

```
-module(contador).
-export([contar/0]).

contar() ->
    contar(0).

contar(Valor) ->
    receive
        cuenta ->
            contar(Valor + 1);
        valor ->
            io:format("El valor es: ~p~n", [Valor]),
            contar(Valor);
        Msg ->
            io:format("Mensaje desconocido: ~p~n", [Msg]),
            contar(Valor)
    end.
```

Ejemplo de Proceso (y II)

```
Eshell V5.7.5  (abort with ^G)
1> Alberto = spawn(contador, contar, []).
<0.34.0>
2> Alberto ! cuenta.
cuenta
3> Alberto ! cuenta.
cuenta
4> Alberto ! valor.
El valor es: 2
valor
5> Alberto ! cuenta.
cuenta
6> Alberto ! valor.
El valor es: 3
valor
```

Links (I)



Links (II)

```
padre() ->
    process_flag(trap_exit, true),
    PidHijo = spawn_link(links, hijo, []),
    io:format("Padre: ~p, Hijo: ~p~n", [self(), PidHijo]),
    recibir_mensajes().

recibir_mensajes() ->
    receive
        Msg ->
            io:format("Mensaje Padre: ~p~n", [Msg]),
            recibir_mensajes()
    end.
hijo() ->
    receive
        Msg ->
            io:format("Mensaje Hijo: ~p~n", [Msg]),
            hijo()
    end.
```

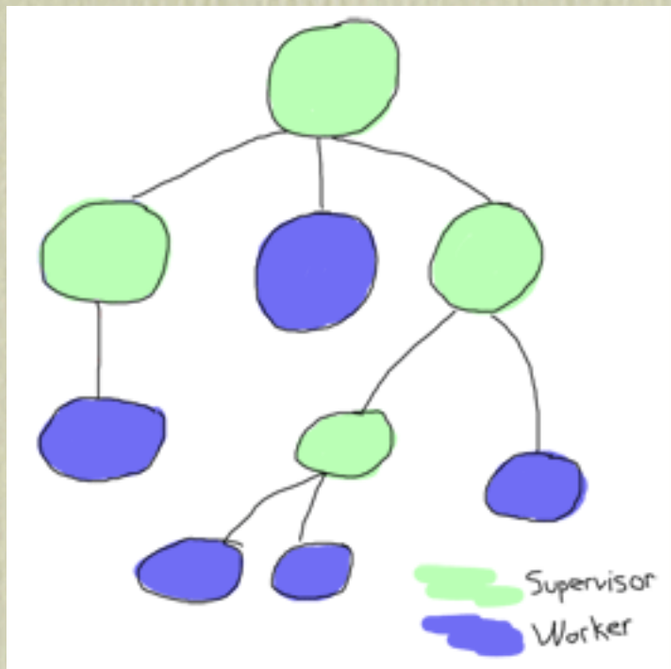
Links (y III)

```
Eshell V5.7.5  (abort with ^G)
1> Padre = spawn(links, padre, []).
Padre: <0.34.0>, Hijo: <0.35.0>
<0.34.0>
2> Padre ! hola.
Mensaje Padre: hola
hola
3> Hijo = list_to_pid("<0.35.0>").
<0.35.0>
4> Hijo ! hola.
hola
Mensaje Hijo: hola
5> exit(Hijo, kill).
Mensaje Padre: {'EXIT',<0.35.0>,killed}
true
```

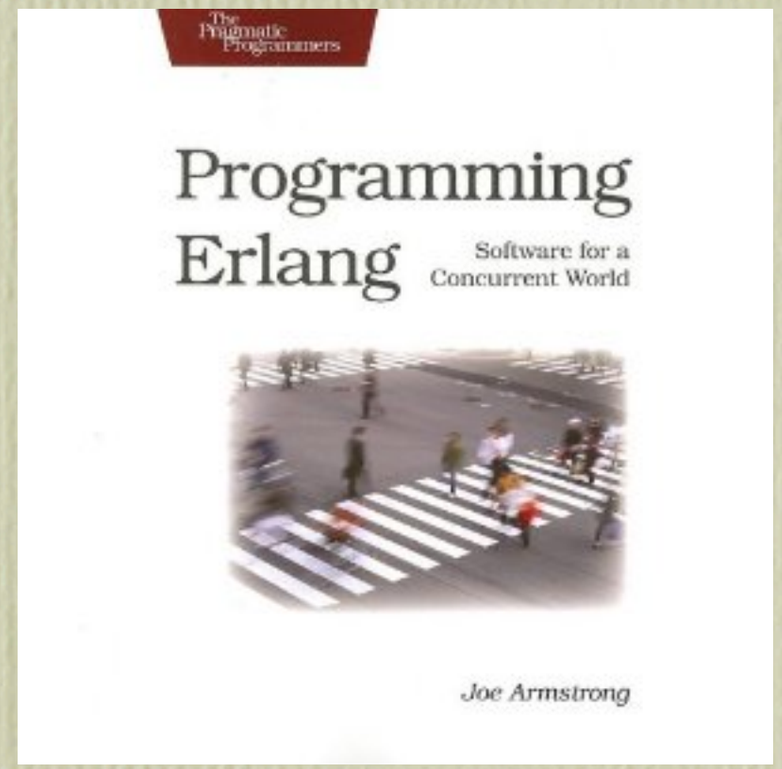
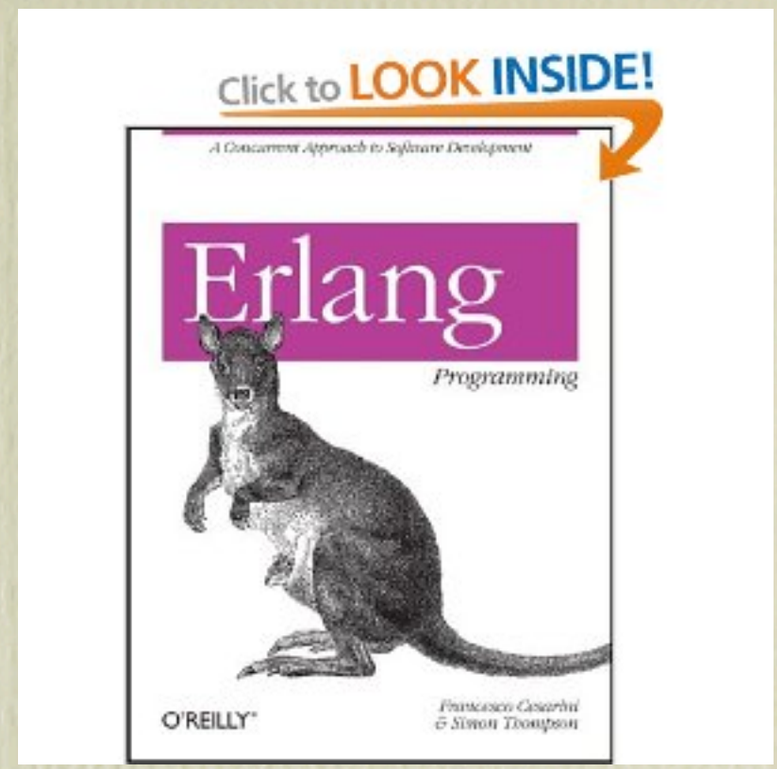
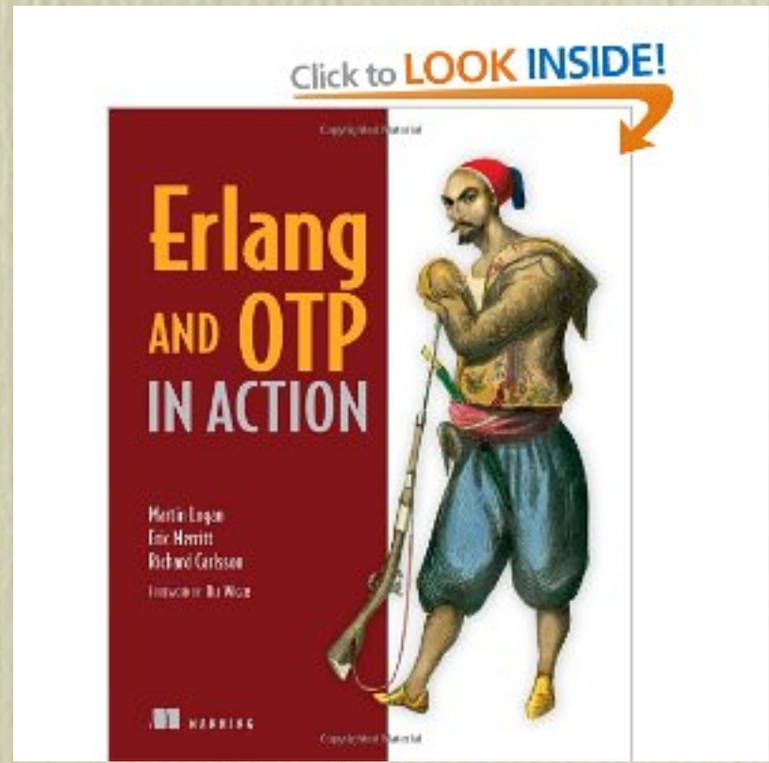
OTP

- gen_server, gen_fsm, gen_event
- applications
- supervisors

Supervisors



Recursos

- http://www.erlang.org
- http://learnyousomeerlang.com
- http://trapexit.org/
- http://www.scribd.com/doc/44221/Thinking-in-Erlang

Recursos (y II)