Erlang

Me

- Víctor Martínez
- Vorago 2006 - 2008 (Ruby on Rails)
- Mobile Interactive Group 2008 - 2012 (Ruby on Rails & Erlang)
- Homestays.com 2012 (Ruby on Rails)
- @vicmargar
- vicmargar@gmail.com

Why Erlang?

Friday, 29 June 12

I started working at Mobile Interactive Group as a Rails developers, most of their apps were related with SMS processing. So a typical app would consist of a Rails front-end plus a series of Ruby processes accessing a queue implemented in MySQL.

This has several problems: DB contention, process management, memory consumption, etc...

We decided to look into Erlang.

Erlang

Basic Aspects

Erlang was developed by Ericsson in the 80s. It's main feature nowadays is how it handles concurrency on multi-core but this has only been added in the last few years.

In this first part I'll talk about it's basic characteristics and how it's similar or different to other languages.

What is 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.

soft real-time

concurrency, distribution and fault tolerance go hand by hand

Originally designed to program telephone switches, to be able to handle thousands of simultaneous processes without one interfering with each other

Functional Language

- Not OO
- Functions are grouped in modules
- Immutable state
- No side-effects (almost)

Friday, 29 June 12

Erlang is a functional language, but unlike Haskell which was designed in an academic environment, Erlang was designed in an industry environment so it's more pragmatic. So there are some functions which have side effects.

Invariable Variables

```
Eshell V5.9.1  (abort with ^G)
1> A = 1.
1
2> A = 2.
** exception error: no match of right hand side value 2
```

Pattern Matching (I)

```
Eshell V5.9.1  (abort with ^G)
1> [ Head | Tail ] = ["one", "two", "three"].
["one","two","three"]
2> Head.
"one"
3> Tail.
["two","three"]
```


Inspired by Prolog originally

Pattern Matching (II)

```
Eshell V5.9.1  (abort with ^G)
1> {_, S, [H |T]} = {hello, "abc", [1,2,3,4]}.
{hello,"abc",[1,2,3,4]}
2> S.
"abc"
3> H.
1
4> T.
[2,3,4]
```

Pattern Matching (III)

```erlang
-module(animals).

-export([sound/1]).

sound(dog) ->
    io:format("Whoof Whoof!~n");

sound(cat) ->
    io:format("Meow!~n");

sound(_Other) ->
    io:format("Grrrrrrrrrr!~n").
```

Pattern Matching (y IV)

```
Eshell V5.9.1  (abort with ^G)
1> animals:sound(dog).
Whoof Whoof!
ok
2> animals:sound(cat).
Meow!
ok
3> animals:sound(fox).
Grrrrrrrrrr!
ok
```

Recursion

```
-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 V5.9.1  (abort with ^G)
1> recursion:map(fun(E) -> 2*E end, [1,2,3,4,5]).
[2,4,6,8,10]
```

Higher Order Functions

```
Eshell V5.9.1  (abort with ^G)
1> Twice = fun(E) -> 2*E end.
#Fun<erl_eval.6.82930912>
2> recursion:map(Twice, [1,2,3,4,5]).
[2,4,6,8,10]
```

List Comprehensions

```
Eshell V5.9.1  (abort with ^G)
1> [ 2 * X || X <- [1,2,3,4,5] ].
[2,4,6,8,10]
```

Bit Syntax (I)

```
Eshell V5.9.1  (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.
```

Hot Code Loading

During the development of a SMS gateway, switch on from one day to another, there were some bugs that needed to be fixed on the fly, it wouldn't have been possible otherwise.

It has it's pitfalls, it's not recommended unless it's the only possibility.

Erlang

Concurrency

Concurrency Vs Parallelism

Concurrency: Several tasks start, execute and finish overlapped during a period of time. It doesn't mean that they are running exactly at the same time.

Parallelism: Several tasks are running exactly at the same time.

Erlang supports concurrency from the beginning, however Parallelism has been added recently (SMP).

Scalability

Erlang is designed from the beginning to handle a big number of lightweight processes. The main goal is to overcome the limitations of hardware. You can do this by adding more hardware or adding more machines. Eventually this is the only option.

Fault Tolerance (Let it Crash)

Errors happen, they are hard to avoid.

Hardware failures are impossible to avoid.

Erlang provides ways to handle errors when they happen.

Independent memory for each process and lack of locks makes it hard for a process to leave another one in an inconsistent state.

The only way to handle a hardware failure is to run the program in several machines.

Actor Model

- Lightweight Processes
- Independent Memory
- Asynchronous Messages
- Async IO

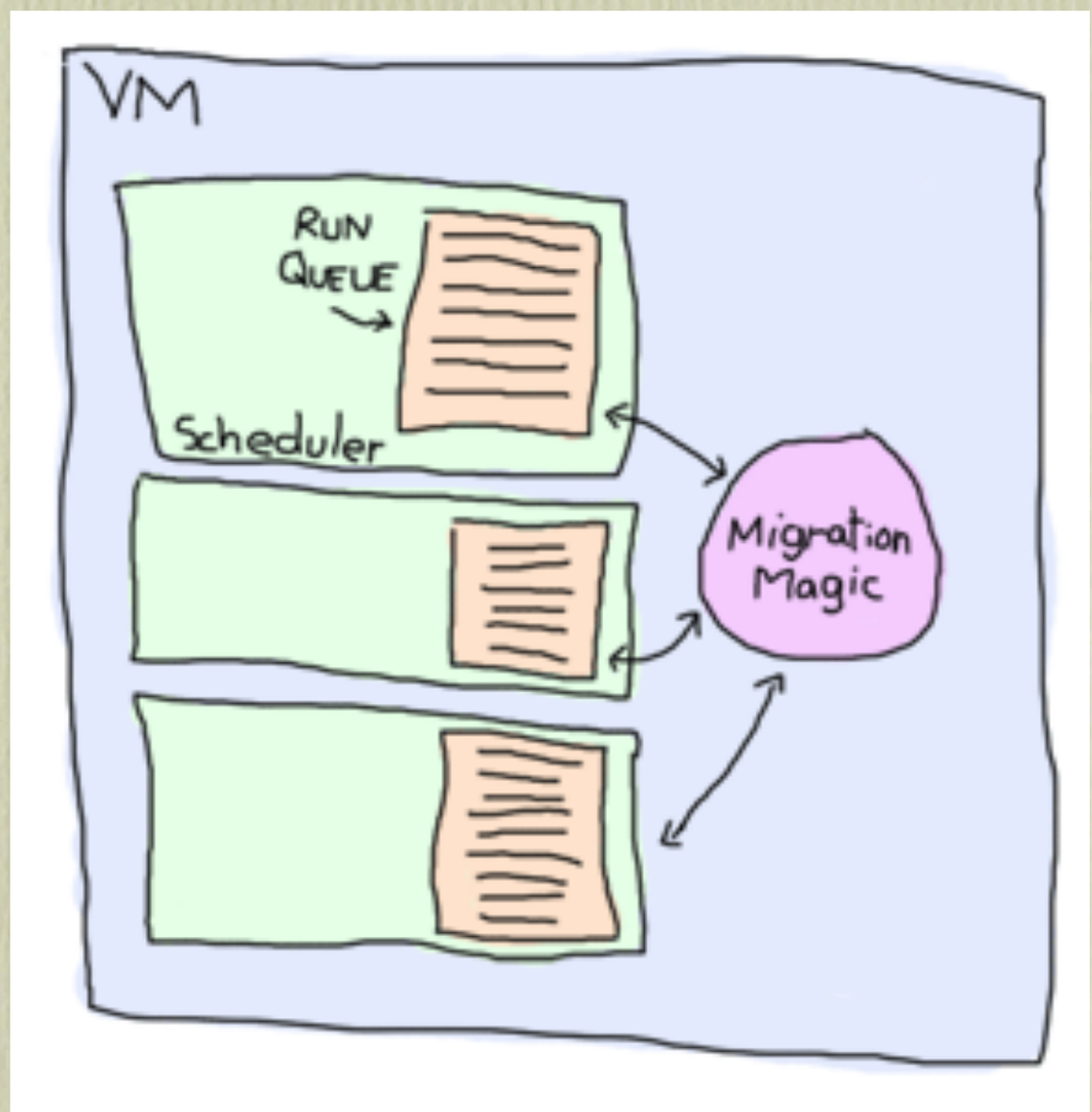
Lightweight processes: Cheap to create and destroy, VM can switch between them very quickly.

Independent memory: Processes don't share memory so locks are unnecessary and it makes GC easy once a process finishes.
This helps soft-real time capabilities since GC stops are small.

Message passing: The only way to pass data from one process to another is by sending a message. This means copying data from one memory area to another one. It's slower but safer.

Async IO:

Process Execution



Friday, 29 June 12

One scheduler per core by default.

Each scheduler has a queue where the processes are executed sequentially.

There's a migration logic that's responsible for keeping the schedulers balanced.

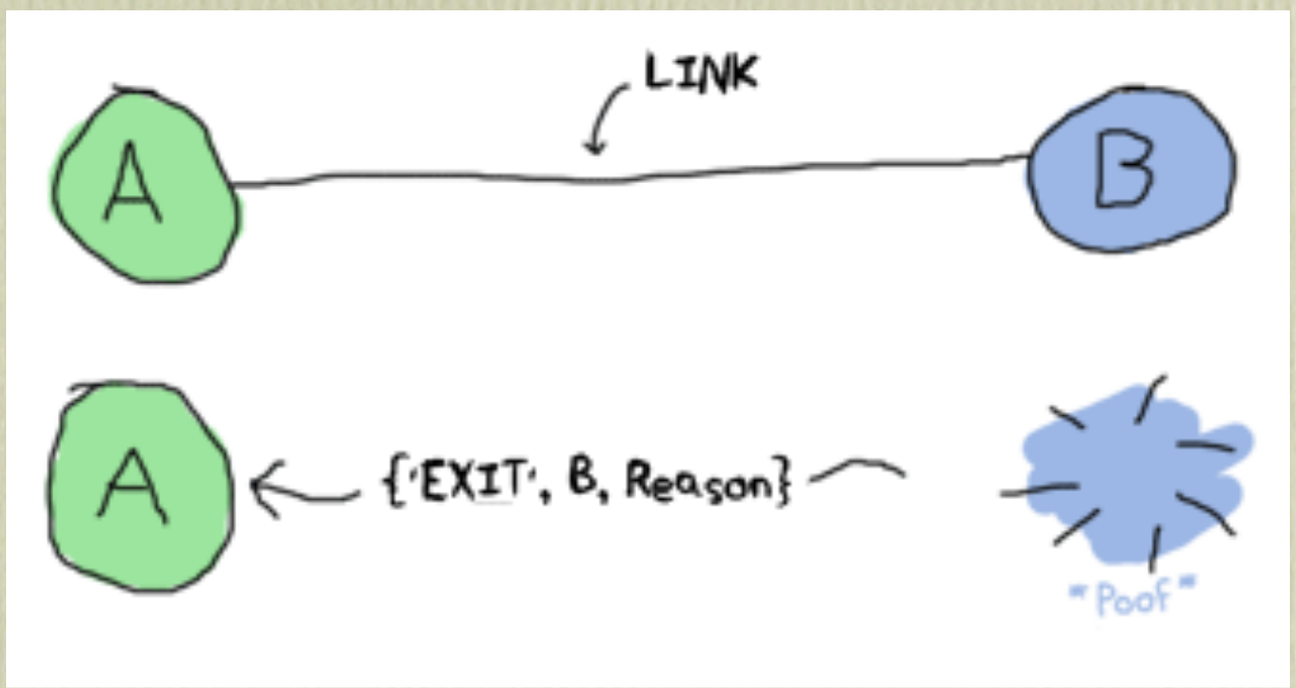
Process Example (I)

```
-module(counter).

-export([count/0]).

count() ->
    count(0).

count(Value) ->
    receive
        count ->
            count(Value + 1);
        value ->
            io:format("The value is: ~p~n", [Value]),
            count(Value);
        Msg ->
            io:format("Unknown message: ~p~n", [Msg]),
            count(Value)
    end.
```

Process Example (y II)

```
1> Counter = spawn(counter, count, []).
<0.33.0>
2> Counter ! count.
count
3> Counter ! count.
count
4> Counter ! value.
The value is: 2
value
5> Counter ! count.
count
6> Counter ! value.
The value is: 3
value
```

Links (I)



Links (II)

```
-module(links).
-export([parent/0, son/0]).

parent() ->
    process_flag(trap_exit, true),
    PidSon = spawn_link(links, son, []),
    io:format("Parent: ~p, Son: ~p~n", [self(), PidSon]),
    receive_messages().

receive_messages() ->
    receive
        Msg ->
            io:format("Message Parent: ~p~n", [Msg]),
            receive_messages()
    end.

son() ->
    receive
        Msg ->
            io:format("Message Son: ~p~n", [Msg]),
            son()
    end.
```

Links (y III)

```
Eshell V5.9.1  (abort with ^G)
1> Parent = spawn(links, parent, []).
Parent: <0.33.0>, Son: <0.34.0>
<0.33.0>
2> Parent ! hello.
Message Parent: hello
hello
3> Son = list_to_pid("<0.34.0>").
<0.34.0>
4> Son ! hello.
Message Son: hello
hello
5> exit(Son, kill).
Message Parent: {'EXIT',<0.34.0>,killed}
true
```

OTP

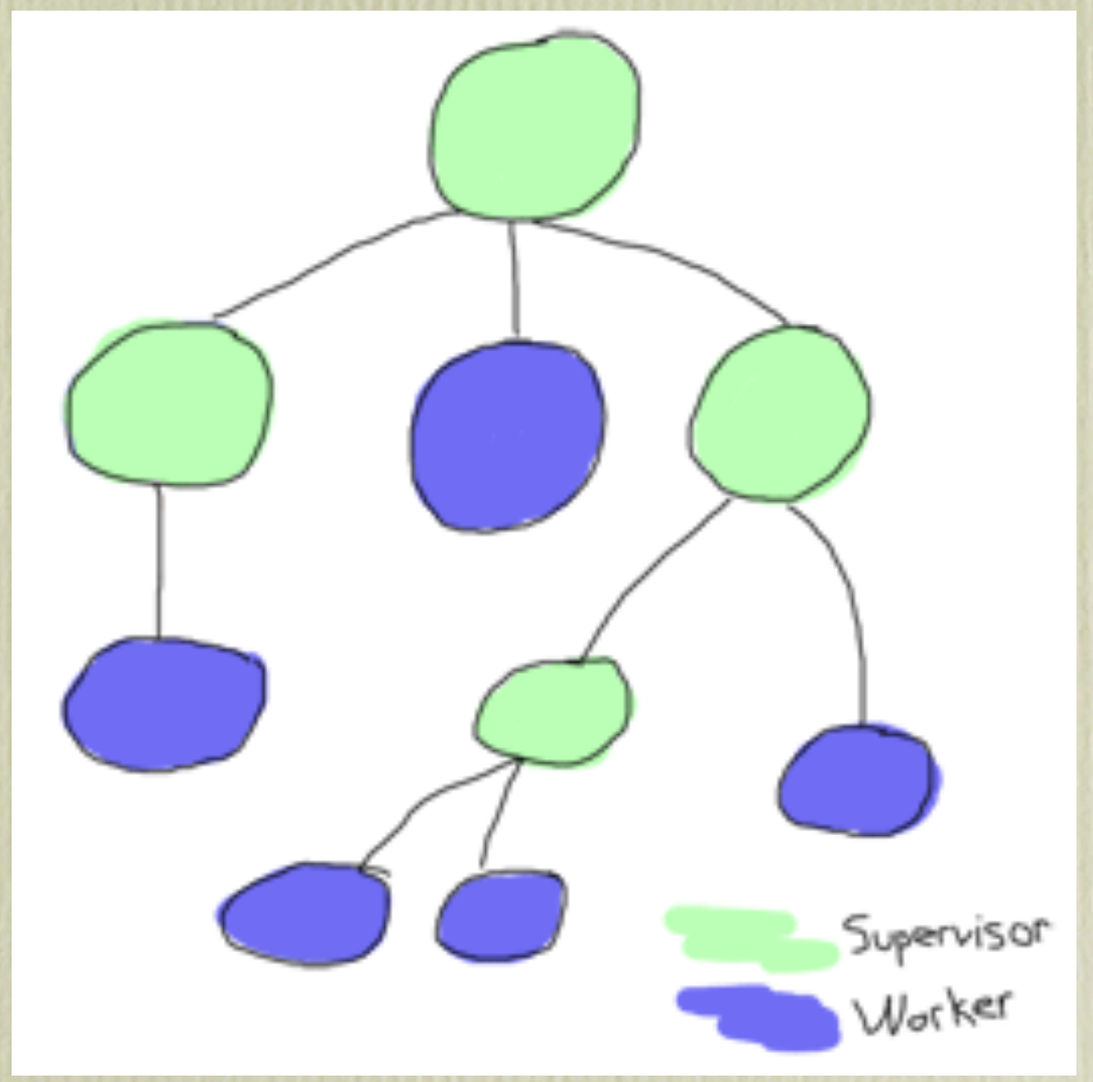
- gen_server, gen_fsm, gen_event
- applications
- supervisors

Friday, 29 June 12

Open Telecom Platform

Set of libraries and generic applications

Supervisors



Who uses Erlang? (I)

DemonWare

amazon

YAHOO!

MIG
mobile interactive group

ERICSSON

Friday, 29 June 12

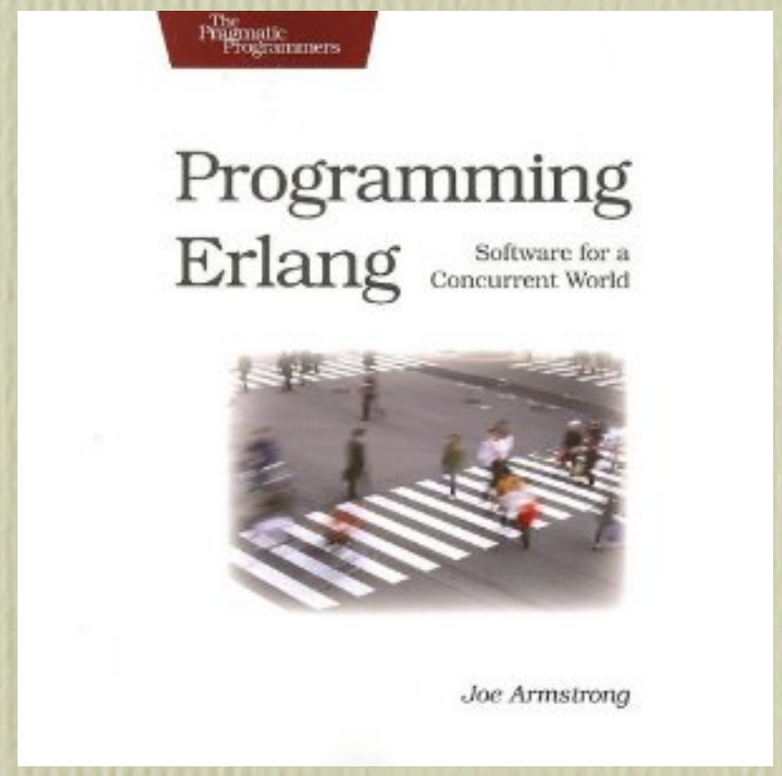
Amazon - SimpleDB utilizada para sus servicios en la nube
Yahoo - Harvester sistema para recoger datos de múltiples fuentes

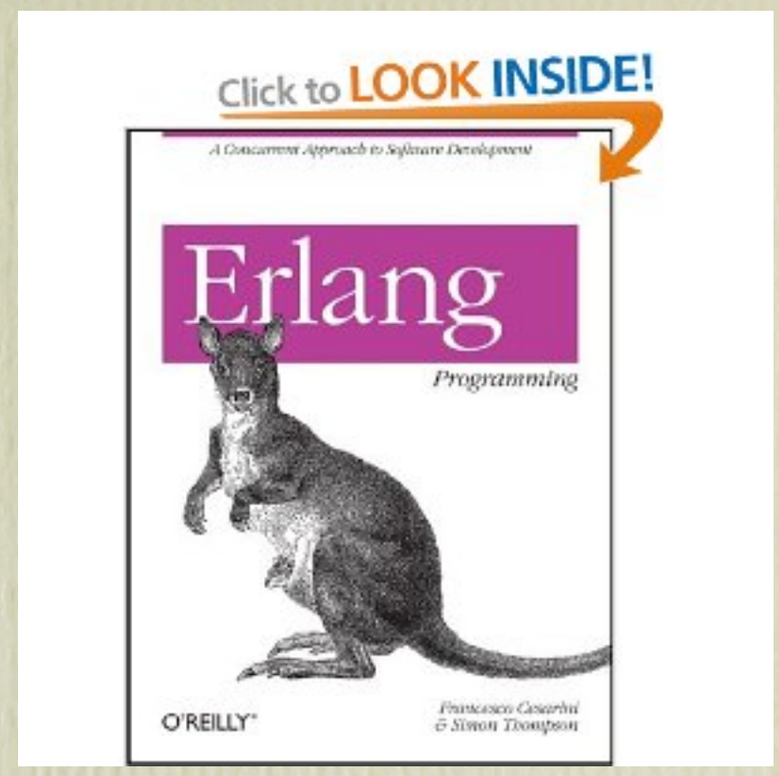
Who uses Erlang? (y II)

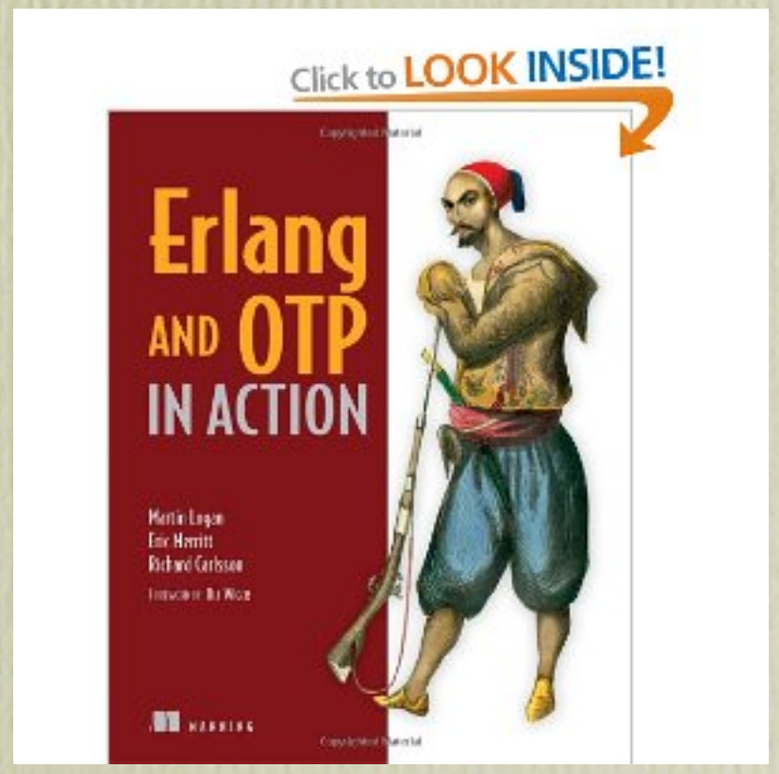
Resources

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

Resources (y II)







The End

https://github.com/vicmargar