
From Java
Java
To Erlang
ERLANG

About Me

Gonthier Olivier

Developer Freelance in Paris

@rolios

o.gonthier@gmail.com

Context

Context

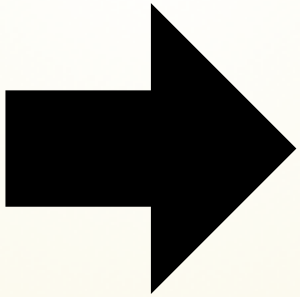
Team switch

Client side

Java

Android

IntelliJ



Server side

Erlang

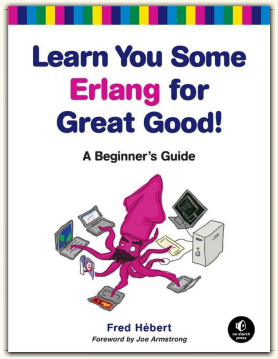
MongooseIM

Vim

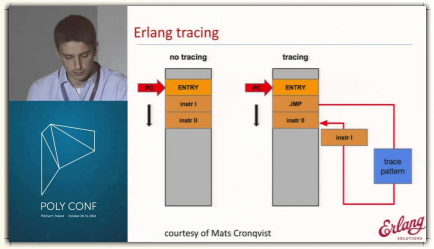
Erlang learning

Erlang SOLUTIONS

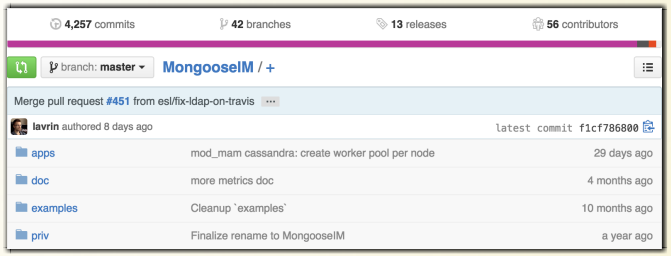
Online training



Read books



Watch talks



Read code

And also:

-Functional principles by learning Haskell

-Scrum-mastering, continuous integration

Android team

“ Oh, you want to use a dead language?
Java is better

Erlang team

“ Oh, you was doing ugly fat Java! Beh Erlang is better

Erlang in a nutshell

Disclaimer

Erlang: what's that?

> *Erlang is a general-purpose, concurrent, garbage-collected programming language and runtime system. (Wikipédia)*

Created by Sony Ericsson in 1987
Open-Source since 1998

VM-based, Functional, impure,
Typed dynamically, Concurrent, Distributed,
Opinionated

Erlang: usage

- Created for Telecom needs
- Ericson platform: 99.9999999% uptime
- Used for xmpp servers: **ejabberd**, **MongooseIM**
- **RabbitMQ**, **CouchDB**
- **WhatsApp** backend

“ Facebook started his chat in erlang, dropped it for c++ in 2009, and then... Bought whatsapp, using erlang, 5 years later

```
741
742 %% ---------------------------------------------------------------------
743 %% Info files
744
745 info_file_data(Ts) ->
746     App = proplists:get_value(application, Ts, ?NO_APP),
747     Ps = proplists:append_values(packages, Ts),
748     Ms = proplists:append_values(modules, Ts),
749     {App, Ps, Ms}.
750
751 %% Local file access - don't complain if file does not exist.
752
753 %% @private
754 read_info_file(Dir) ->
755     File = filename:join(Dir, ?INFO_FILE),
756     case filelib:is_file(File) of
757         true ->
758             case read_file(File) of
759                 {ok, Text} ->
760                     parse_info_file(Text, File);
761                 {error, R} ->
762                     R1 = file:format_error(R),
763                     warning("could not read '~ts': ~ts.", [File, R1]),
764                     {?NO_APP, [], []}
765             end;
766         false ->
767             {?NO_APP, [], []}
768     end.
769
770 %% URI access
771
772 uri_get_info_file(Base) ->
773     URI = join_uri(Base, ?INFO_FILE),
774     case uri_get(URI) of
775         {ok, Text} ->
776             parse_info_file(Text, URI);
777         {error, Msg} ->
778             warning("could not read '~ts': ~ts.", [URI, Msg]),
779             {?NO_APP, [], []}
780     end.
781
782 parse_info_file(Text, Name) ->
783     case parse_terms(Text) of
784         {ok, Vs} ->
785             info_file_data(Vs);
786         {error, eof} ->
787             warning("unexpected end of file in '~ts'.", [Name]),
788             {?NO_APP, [], []};
789         {error, {_Line,Module,R}} ->
790             warning("~ts: ~ts.", [Module:format_error(R), Name]),
791             {?NO_APP, [], []}
792     end.
```

What it looks like!

Syntax

```
-module(my_module).
-export([my_func/1, my_other_func/2, another_one/0]).

% A comment

%% A function returning its parameter
my_func(X) -> X.

%% A function using pattern matching
my_other_func(true, _) -> ok;
my_other_func(_, _) -> error.

%% A function with multiple clauses
another_one() ->
    X = my_func(0),
    Y = other_module:other_func(),
    X+Y.

%% A function not exported
private_function() -> ok.
```

Java analogy

module = class; **function** = method; **export** = public

Pattern matching

```
% Matching on function definition
invert(true) -> false;
invert(false) -> true.

% Values passed in parameter are bound
add_one(X) -> X+1.

% Pattern matching using case syntax
case write_file("helloworld") of
    {ok, File} -> File;
    {error, Error} -> log_error(Error)
end.

% Matching bind variables
> X = 1.

% Binding can be done even with data structures
> {Y, [Z, B]} = {2, [3, true]}

% Bounded variable can't match other values
> X = 2.
** exception error: no match of right hand side value 2

% No problems if trying to bind the same value
> {X, Y} = {1, 2}.
```

Data types

Boolean: true, false

Numbers: 1, 0.5, 2.0e3, $A, 2#101001

Atoms: hello, this_is_an_atom, ok

Tuples: {1, 2}, {temp, {celsius, 21}}

List: [1,2], [true, 1, ok, {beer, blond}]

Strings: "hello", [96,97,98]

Also: **Binaries**, **PID**, **Fun**, **Map**

Atoms

Atoms are a kind of simple "tags"

In use, it can be similar to Java enums or static values.

```
public static final int DECEMBER = 12;

public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}

selectDay(25, SUNDAY, DECEMBER);
```

```
select_day(25, sunday, december).
```

They are often used to give context to values

```
{prices, [{beer, 2}, {vodka, 10}], euro}.
```

Strings

Beh.

```
> [104, 101, 108, 108, 111]
"hello"

> [X, Y] = "ab".
> X.
97

> [X, Y, 0].
[97, 98, 0]

Excuse me?
```

Alternative: bit strings

```
> <<"Hello world">>.

> <<A, B>> = <<"ab">>.
> A.
97
```

Concurrency

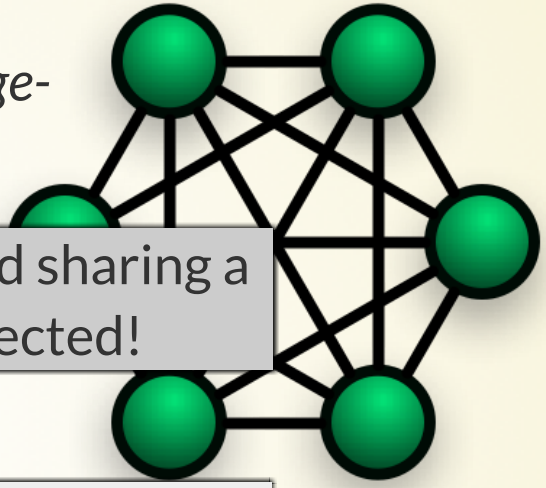
Lightweight processes, and ***Message Passing***

```
-module(echo).
-export([go/0, loop/0]).

go() ->
        Pid2 = spawn(echo, loop, []),
        Pid2 ! {self(), hello},
        receive
                {Pid2, Msg} ->
                        io:format("P1 ~w~n",[Msg])
        end,
        Pid2 ! stop.

loop() ->
        receive
                {From, Msg} ->
                        From ! {self(), Msg},
                        loop();
                stop ->
                        true
        end.
```

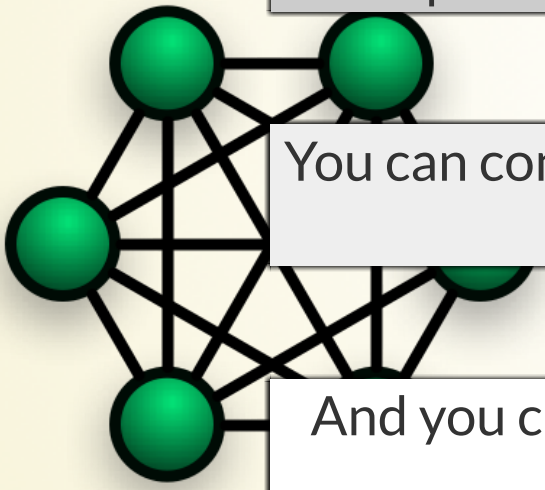
```
Pid ! {MyPid, Msg} % Send message to Pid
```

Distributed system

-At the heart of the language-



Erlang VMs visible on network and sharing a specific ***cookie***, are interconnected!



You can connect/disconnect nodes whenever you want

And you can send messages to other nodes transparently!

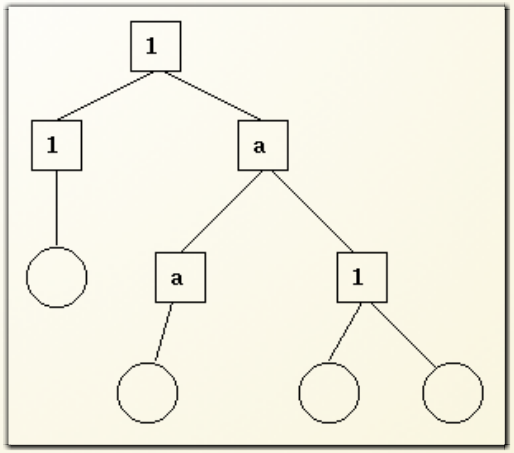
OTP

-Open Telecom Platform-

Abstract things we need often,
like server/client pattern

Supervision tree made easy

```
-behaviour(supervisor).
-behaviour(gen_server).
-behaviour(gen_event).
-behaviour(gen_fsm).
```




Erlang philosophy

Let it crash!

But recover from it

No need of defensive programming

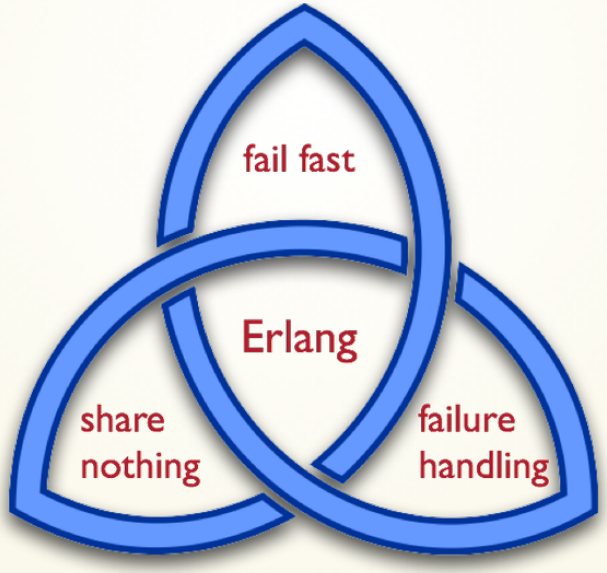
Kill yourself if you can't recover.

Think in terms of protocols

Message passing
Share nothing

Kill people not respecting the protocol.

Golden trinity



Tools

Eshell

```
$> erl
```

```
Erlang/OTP 17 [erts-6.3] [source-f9282c6] [64-bit] [smp:8:8] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.3  (abort with ^G)
1> X=1.
1
2> X.
1
3>
```

```
% Compile a module
c(my_module).

% Unbind var
f(X).
```

REPL

Debug Console

Hot reload

Good News: Hot reloading is available in Erlang!

Nothing to do, just compile and use your new *.beam*

Thoughts

you don't need to reboot your app

static typing would have been complicated with this

Eunit

Or Junit with a 'E' instead of 'J'

```
-module(fibo).

-ifdef(EUNIT).
-include_lib("eunit/include/eunit.hrl").
-endif.

fib(0) -> 1;
fib(1) -> 1;
fib(N) when N > 1 -> fib(N-1) + fib(N-2).

fib_test_() ->
       [?_assert(fib(0) =:= 1),
        ?_assert(fib(1) =:= 1),
        ?_assert(fib(2) =:= 2),
        ?_assert(fib(3) =:= 3),
        ?_assert(fib(4) =:= 5),
        ?_assert(fib(5) =:= 8),
        ?_assertException(error, function_clause, fib(-1)),
        ?_assert(fib(31) =:= 2178309)
       ].
```

Rebar

Maven/Gradle equivalent

```
$ rebar create-app appid=myapp
$ rebar get-deps
$ rebar compile
$ rebar eunit
$ rebar generate
```

Muptiple commands to compile/test/package etc.

```
{deps, [
    {em, ".*", {git, "https://github.com/sheyll/erlymock.git"}},
    {nano_trace, ".*", {git, "https://github.com/sheyll/nano_trace.git", {branch,
    {mochiweb, "2.3.2", {git, "https://github.com/mochi/mochiweb.git", {tag, "v2.3
    % Or specify a revision to refer a particular commit, useful if the project ha
    % {mochiweb, "2.3.2", {git, "https://github.com/mochi/mochiweb.git", "15bc558c

    % An example of a "raw" dependency:
    {rebar, ".*", {git, "git://github.com/rebar/rebar.git", {branch, "master"}}, [
   ]}.
```

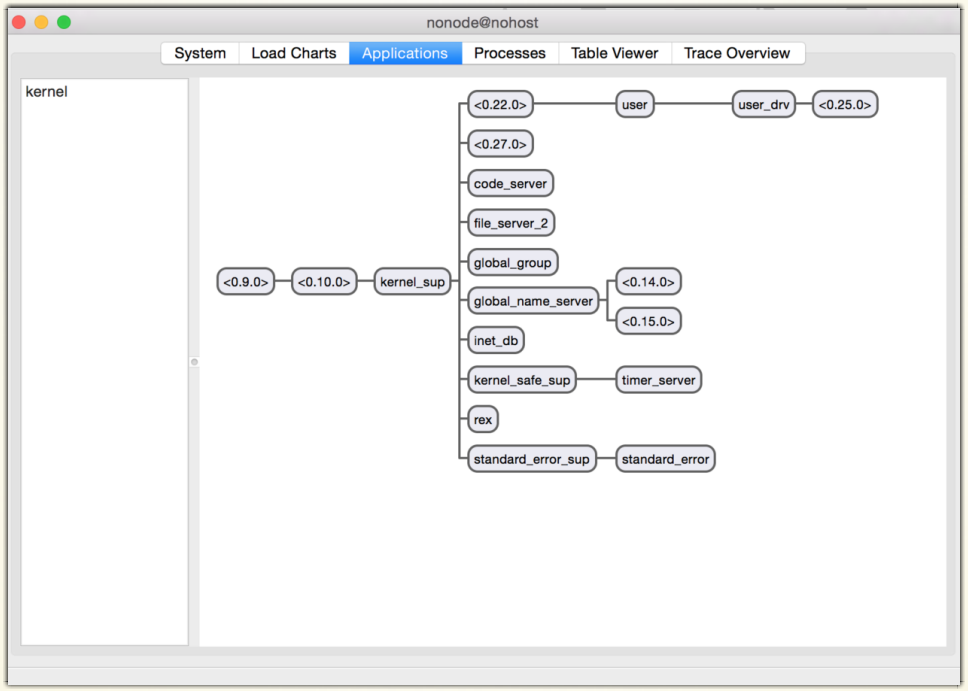
Configuration is done in a *rebar.config* file

Dependencies are git repositories :(

Observer

A very useful debug console

```
> observer:start().
```



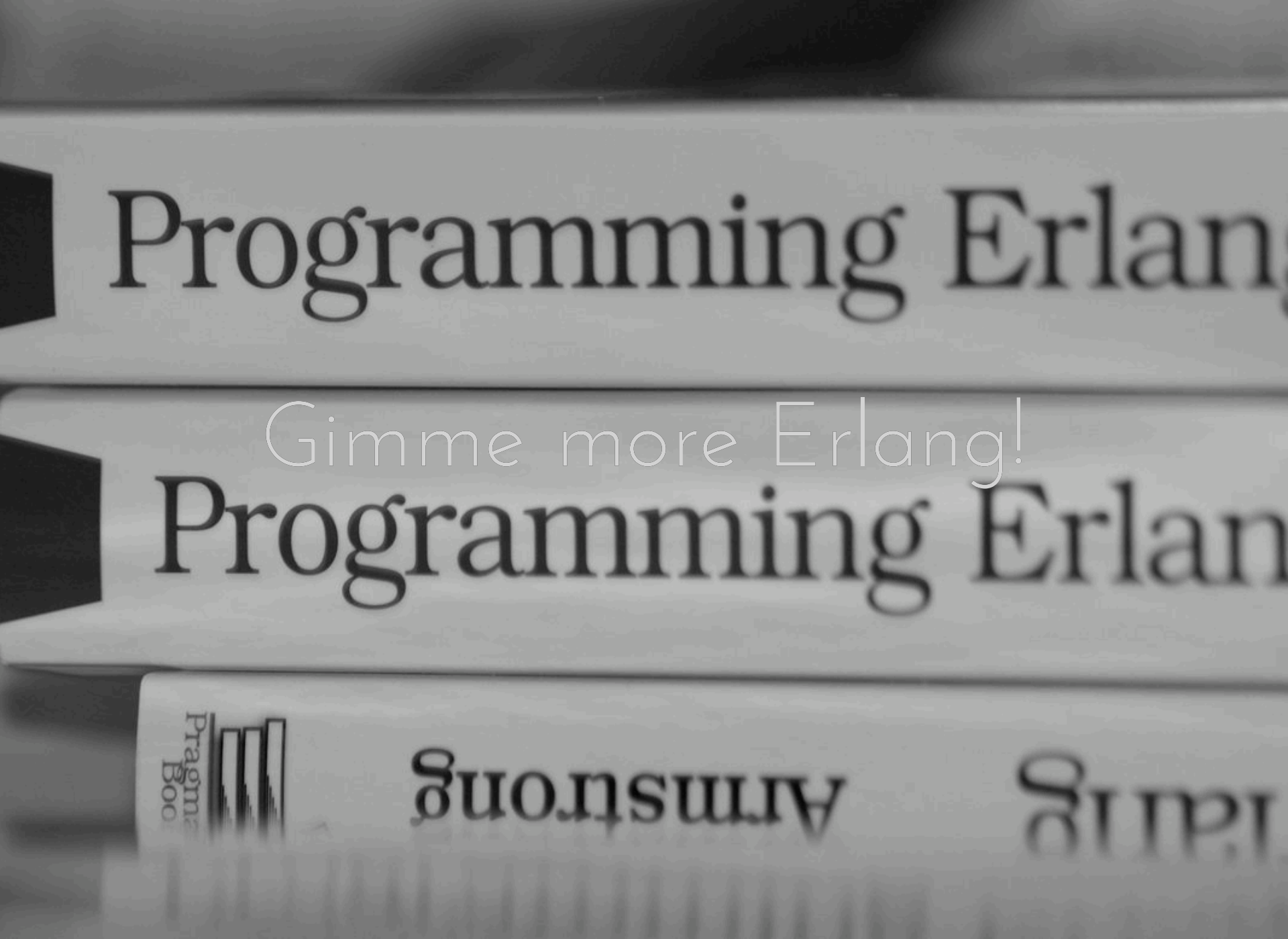
Supervision tree, tracing, systems statistics, consumption...

Other

Dialyzer: let you do a static analysis
Reltool: tool to make erlang releases
(used by rebar)

Erlang.mk: Alternative to rebar based on a makefile

WombatOAM: commercial product for system supervision


Programming Erlang
Gimme more Erlang!
Programming Erlang
Armstrong

Erlang: the movie

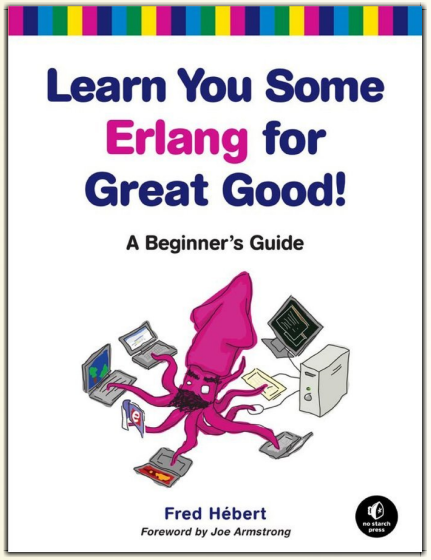
http://www.youtube.com/watch?v=xrIjfIjssLE

Erlang : The movie II - The sequel

http://www.youtube.com/watch?v=rRbY3TMUcgQ

If you want to hear about a new name for Erlang...

Good resources



Awesome book!

Blogs

http://joearms.github.io/

http://mostlyerlang.com/

http://ferd.ca/

http://ninenines.eu/articles/

A good list of resources

https://gist.github.com/macintux/6349828

http://erlang.org

Gonthier Olivier
@rolios
o.gonthier@gmail.com