An Introduction to Erlang

for Python programmers

Paul Barry – Institute of Technology, Carlow in Ireland

PyCon Ireland 2011 - October 2011

Grab the slides:

http://paulbarry.itcarlow.ie/ErlangWebcast.pdf

Disclaimer

What exactly is Erlang?

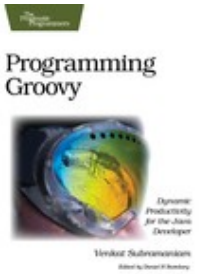
"Erlang is a *declarative,dynamically-typed*, *functional*, *concurrent*, *distributed* and *fault-tolerant* programming language with *garbage collection* and *code hot-swapping* built into its runtime."

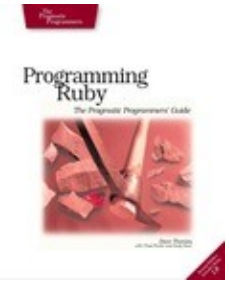
ERICSSON 

Scratching an itch in
the early '80s...

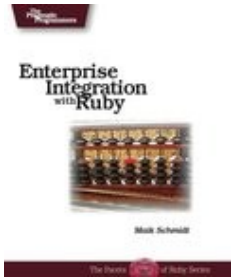
Why learn Erlang?

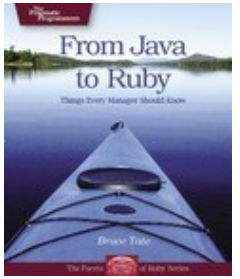
“Learn a new programming language every year”

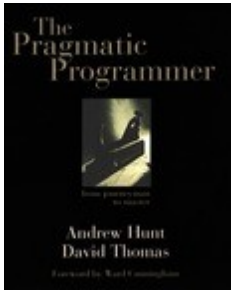




“Buy at least one new programming book every year”







So, *really*, why learn Erlang?

A better programmer,
you will be...

Learn how others solve problems with a different language, then apply these new techniques to your *current* language

Learn what each language is good at,
then pick the *right tool* for the job

Works with a really big hammer!


What is Erlang good at?

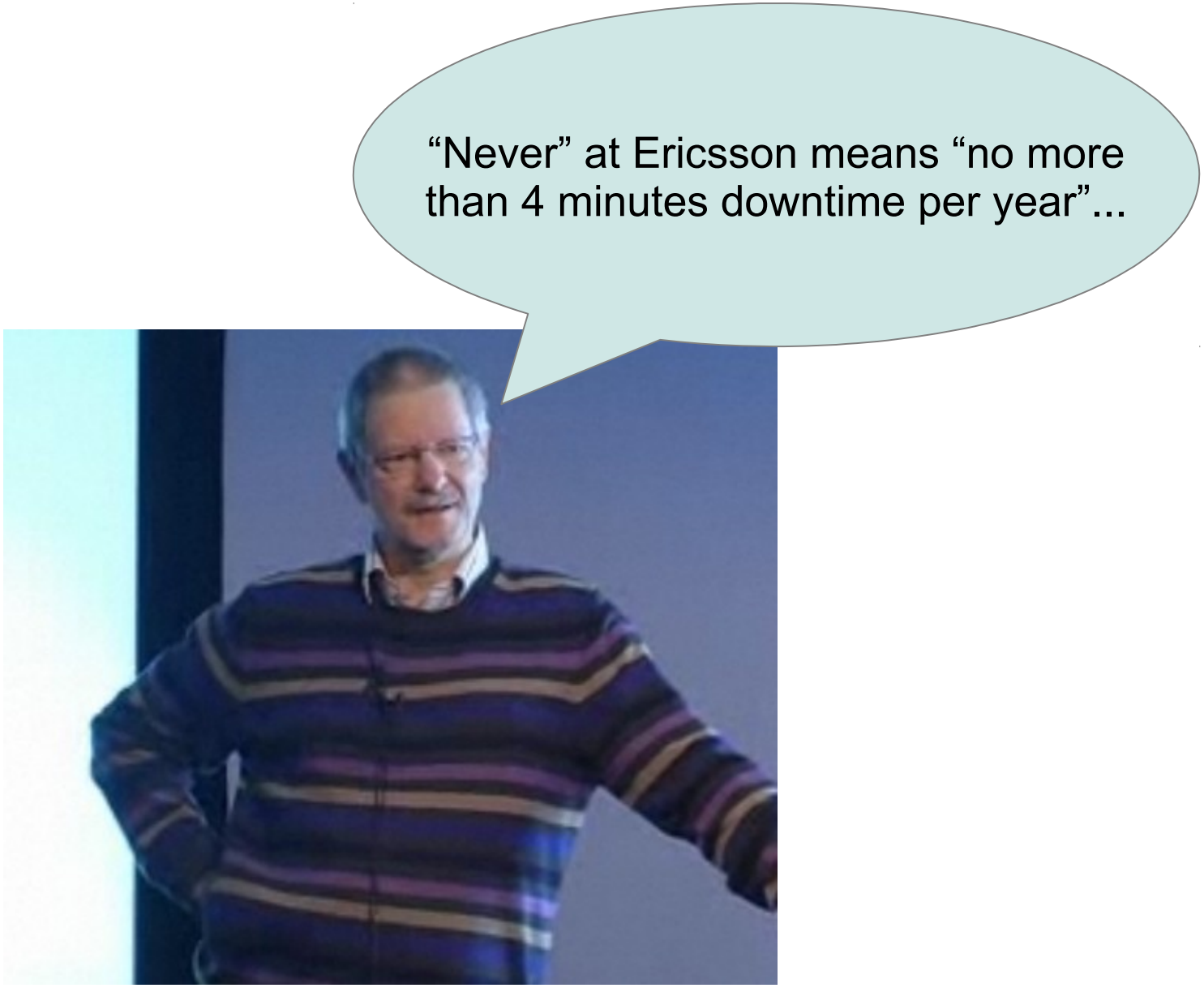
What Erlang *Wasn't* Designed To Do

Erlang wasn't designed to...

- Run in a browser
- Process text efficiently
- Be easy to teach
- Be easy to learn
- Build dynamic websites
- Run on mobile phones
- Allow non-programmers to program
- Build GUIs

Erlang was designed to build software systems that **never** stop


What *exactly* do you mean by “never”?



http://www.infoq.com/presentations/Systems-that-Never-Stop-Joe-Armstrong

Wow! That's 5 nines availability!*

* 99.999% uptime


Let's build software
that's 100% defect free!

Let's throw lots and lots of code at the problem and it'll go away, eh?

There has to be a better way

Let it crash!

Erlang programmers concentrate on
coding for the correct case
and crashing on failure

Robustness is achieved by having programmers concentrate on **processes** and the **interactions** (or *messages*) between them

Early error detection/recovery and the use of concurrent programming techniques lets you build **fault tolerant systems**

COP: Concurrency-Oriented Programming

The problem is...

The problem is...

Erlang is a little weird

Consider this code...

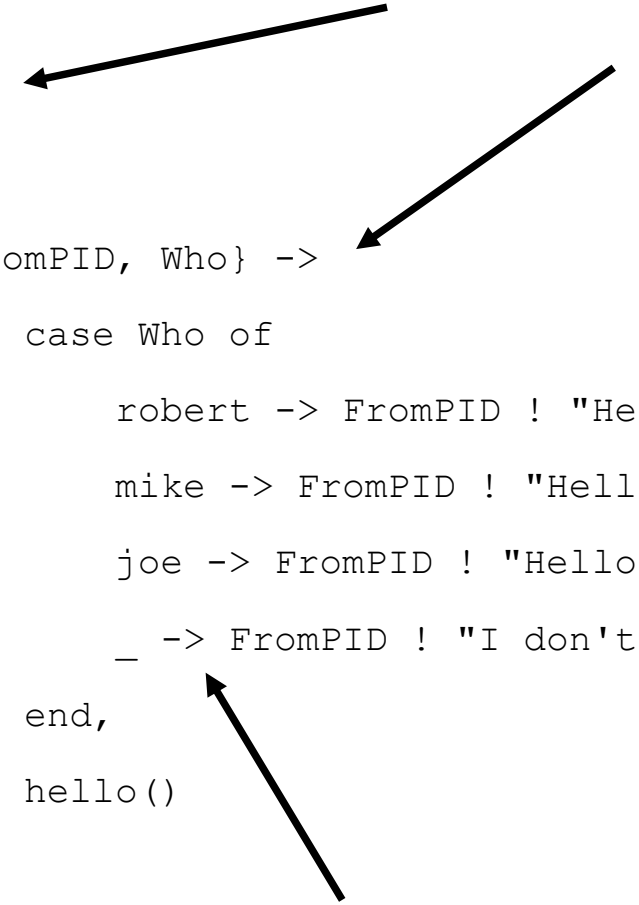
```
-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```

Strange punctuation symbols . ; ,

```
-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```

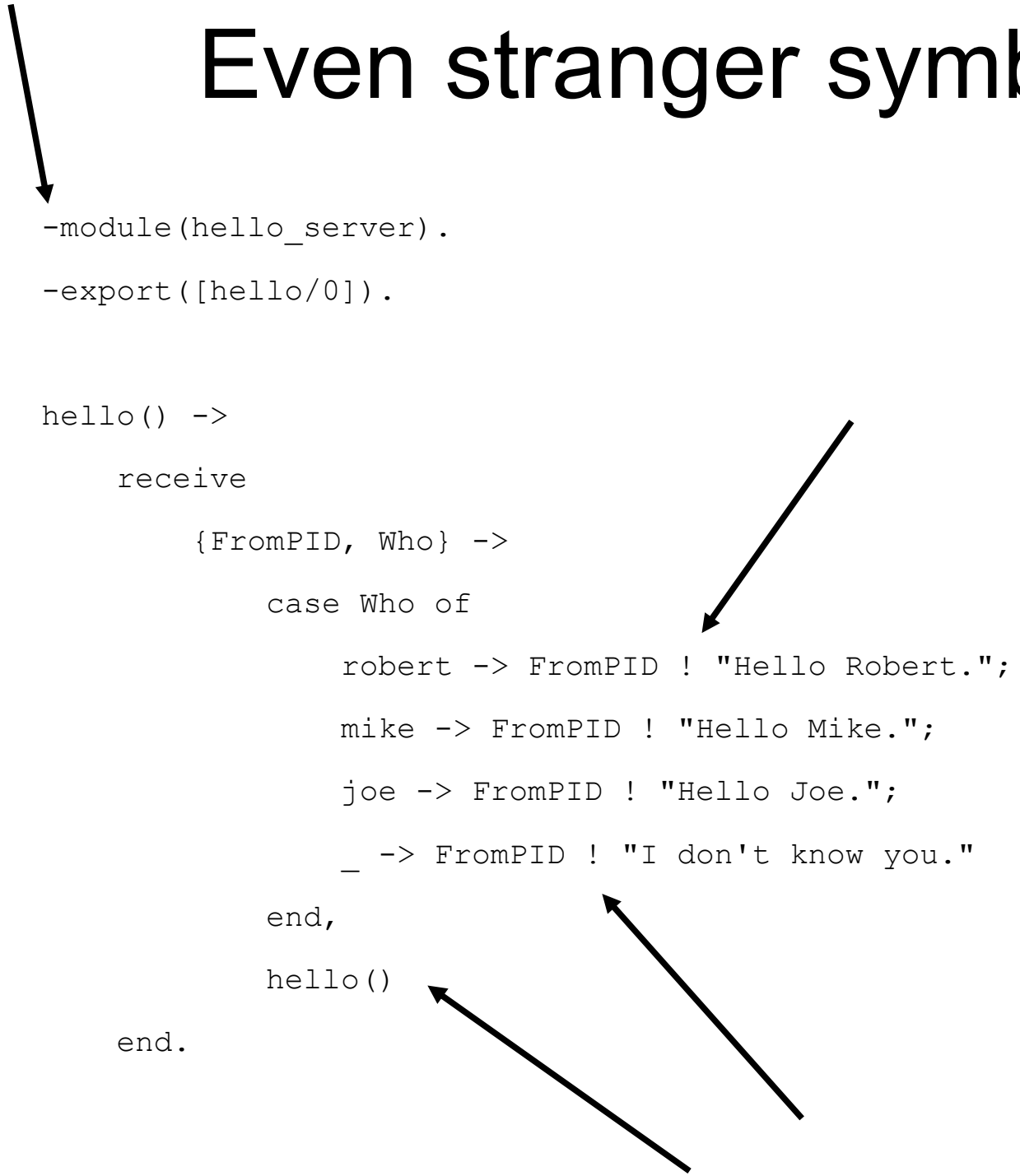
Lots of arrows −> in lots of places

```
-module(hello_server).
-export([hello/0]).

hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



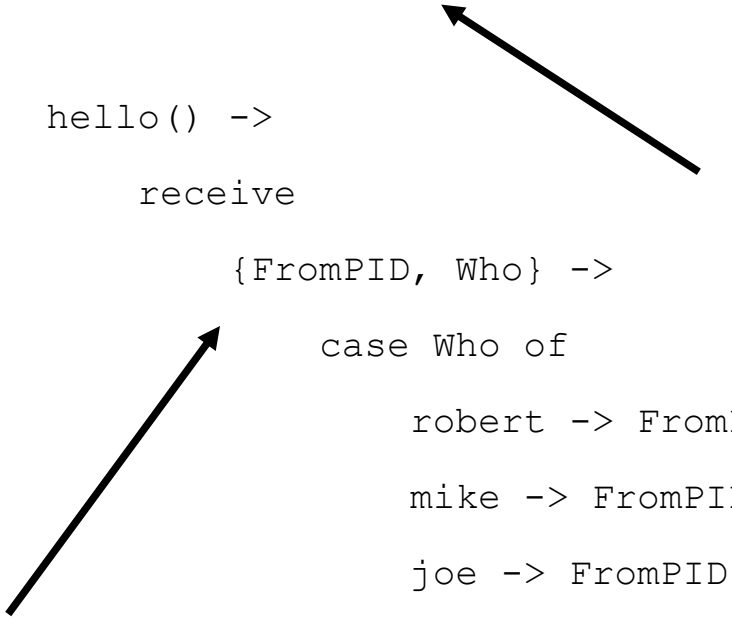
Even stranger symbols - _ !

```
-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



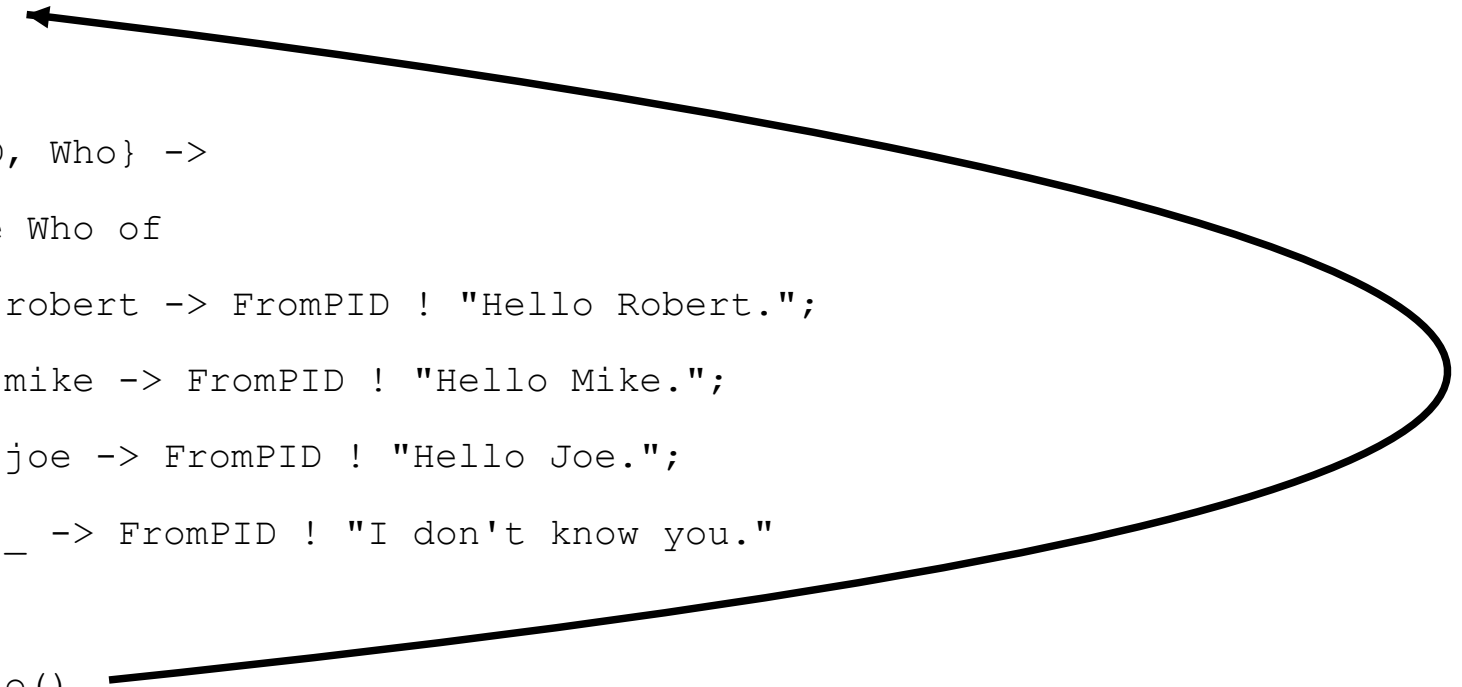
What's the deal with [] {} and () ?

```
-module(hello_server).
-export([hello/0]).

hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



Functions that call themselves

```
-module(hello_server).
-export([hello/0]).

hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



This *is* weird...

and there's even more

Erlang Culture Shock

Shock #1

Erlang's syntax is based on **Prolog**

```
-module(howdy).
-export([hi/1]).

hi(Name) ->
     io:format('Hi there, ~p!~n', [Name]).



{person, First, Last} = {person, 'Paul', 'Barry'}.

howdy:hi(Last).
```

Shock #2

Erlang is **not** object-oriented

Shock #3

Everything in Erlang is **immutable**

Not just tuples and not just strings...

EVERYTHING

So... this **doesn't** work!

```
X = 10.
X = X + 1.
```

*** exception error: no match of right hand side value 11*

```
Y = X + 1.
```

Erlang's troublesome = operator

The “=” operator does **not** mean “assign”

It means “match” or “bind to”

No side effects here!

Destructive updates are **forbidden**

and (as an added twist)

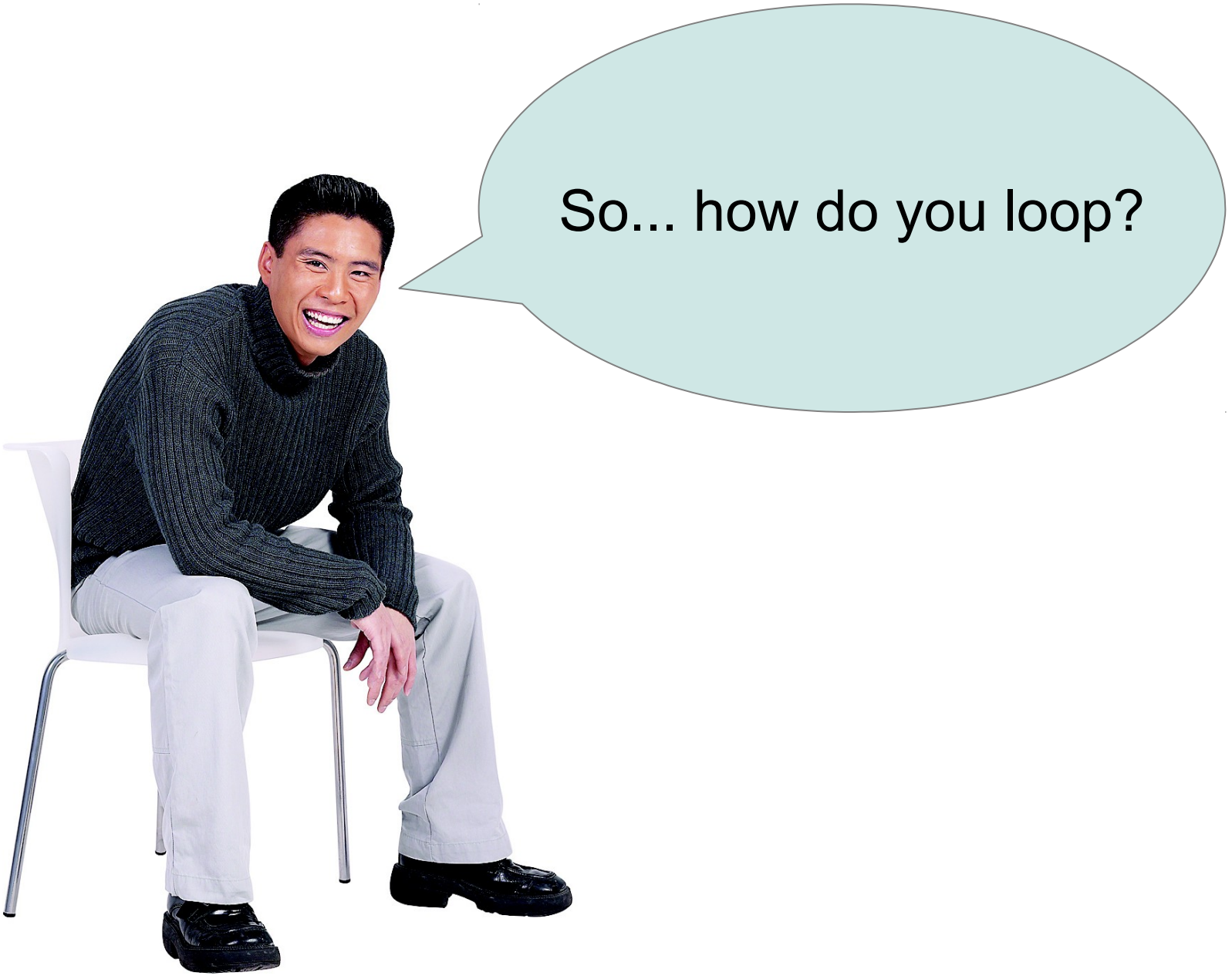
Variables must start with an Uppercase letter:

```
X
Pid
Func
```

Shock #4

There are no built-in looping constructs

No `for` and no `while`


So... how do you loop?

List Comprehensions

Erlang:

```
Alist = [1, 2, 3, 4, 5].
Doubled = [X*2 || X <- Alist].
```

Python:

```
alist = [1, 2, 3, 4, 5]
doubled = [x*2 for x in alist]
```

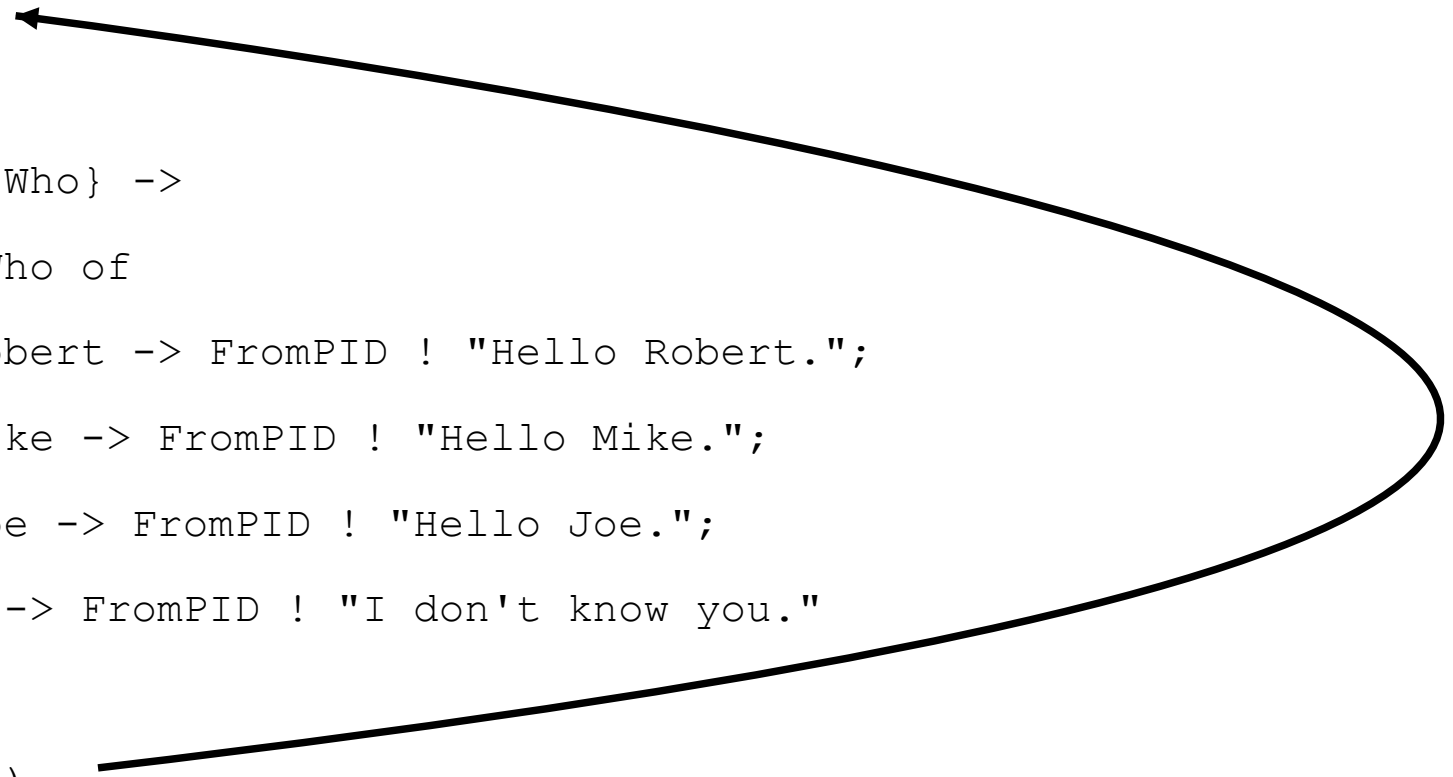
Note: `alist` and `doubled` are *mutable*; `Alist` and `Doubled` are not

Shock #5

"Regular" looping is via **recursion**

Spot the loop?

```
-module(hello_server).
-export([hello/0]).

hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



Shock #6

Strings are stored as a
list of integers


Sweet mother of all things Erlang! Whose *bright idea* was that?

Shock #7

There is no

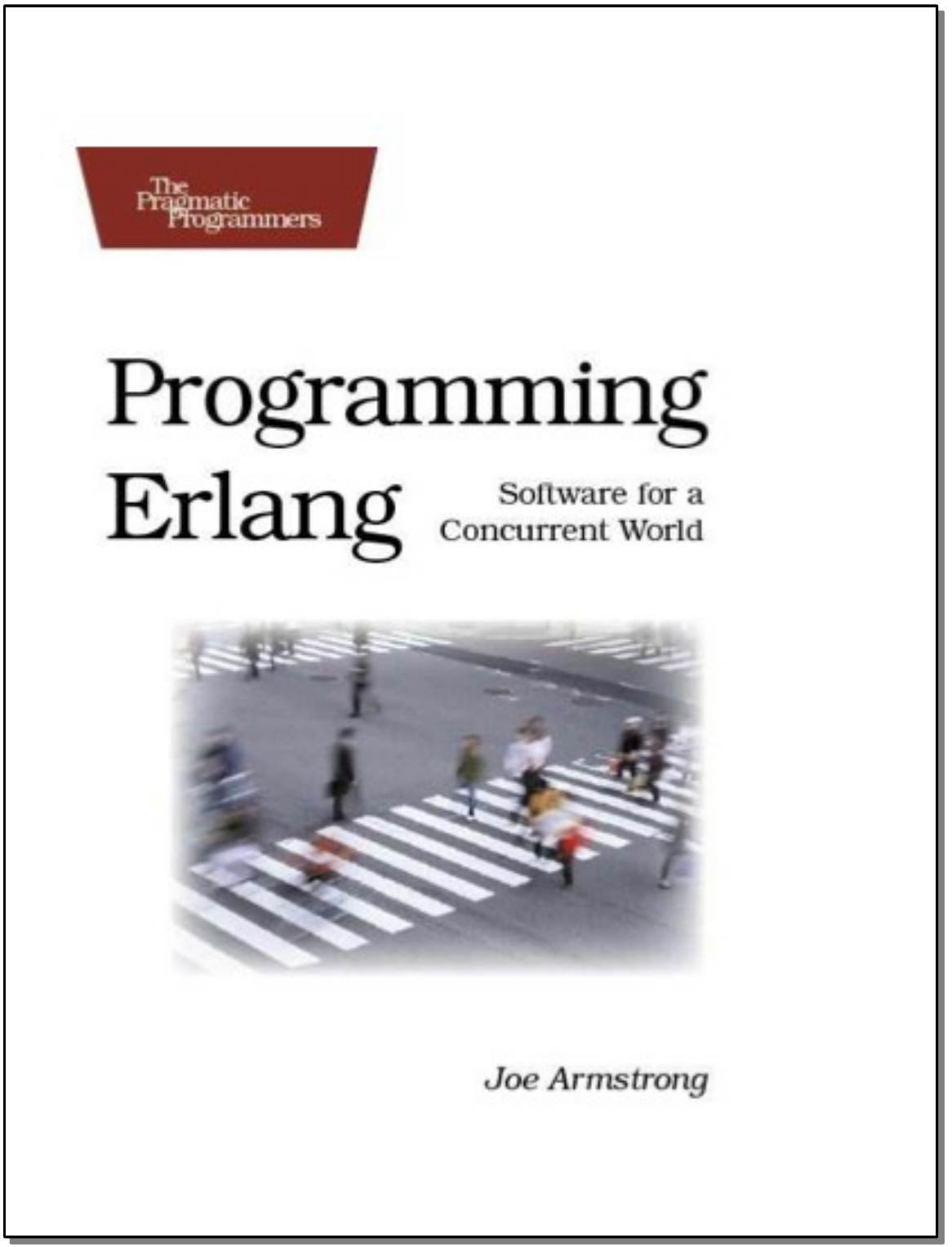
`if... then... else...`

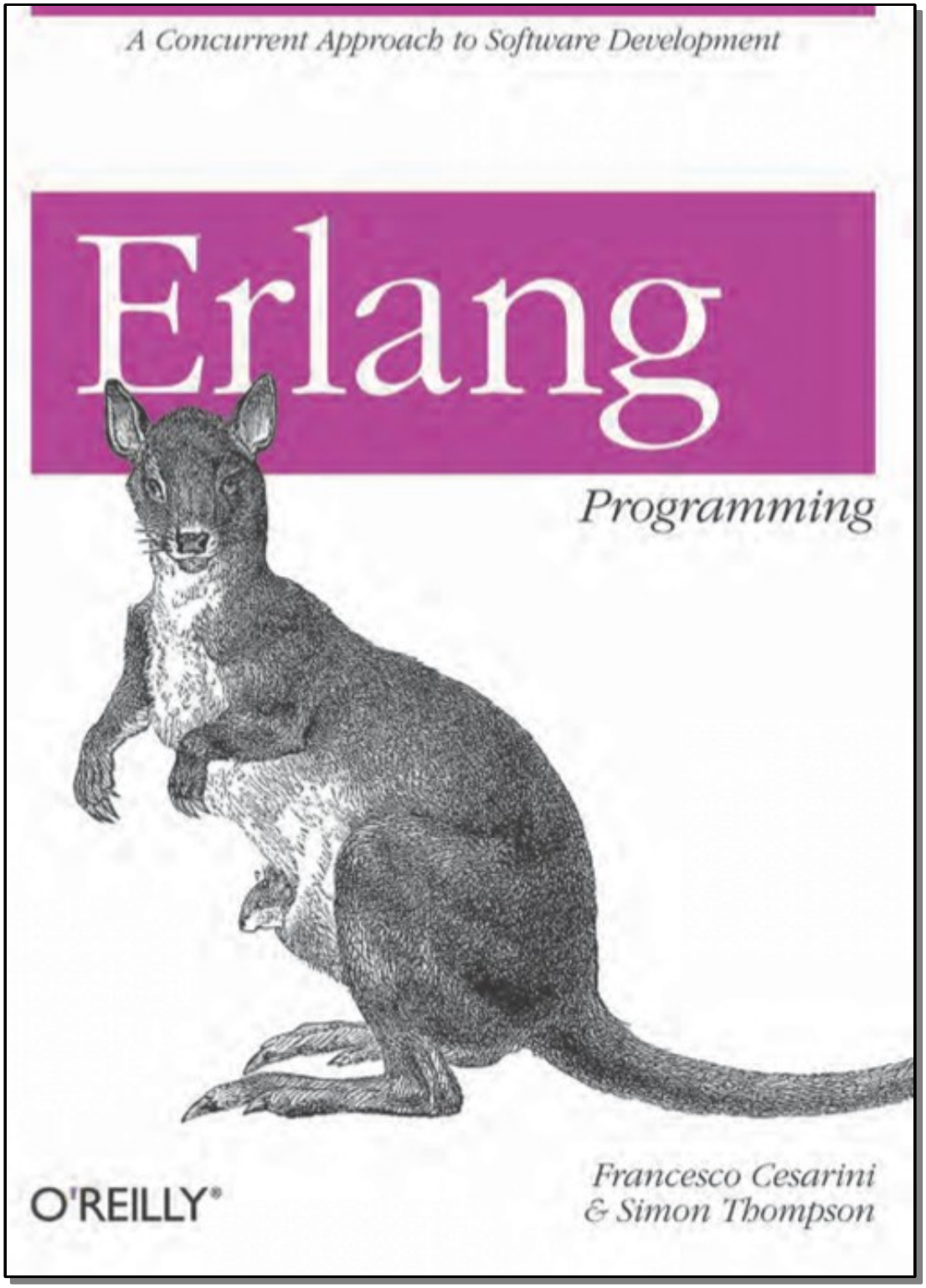
statement

There are `if` and `case` expressions
but... they're kinda weird

With enough pig-headed persistence, weirdness can be overcome


So... who do I call to
help me learn Erlang?


The Pragmatic Programmers
Programming Erlang
Software for a Concurrent World
Joe Armstrong


A Concurrent Approach to Software Development
Erlang
Programming
O'REILLY®
Francesco Cesarini
& Simon Thompson

Erlang is a language you ***study***
as getting up-to-speed with
Erlang takes time

Web Resources

- `http://www.erlang.org`
- Try Erlang:
  - `http://www.tryerlang.org/`
- “Learn Some Erlang For Great Good!”:
  - `http://learnyousomeerlang.com/`
- The Erlang Factory:
  - `http://erlang-factory.com/`
- InfoQ:
  - `http://www.infoq.com`

Some Unique “Characteristics”

Banned by Ericsson in 1998 for
“not being open enough”

Erlang: The Movie

http://video.google.com/videoplay?docid=-5830318882717959520


THE REBELLION BEGINS
Erlang
The Movie
ERLANG SOLUTIONS PRESENTS
AN ETC PRODUCTION "ERLANG - THE MOVIE" MIKE WILLIAMS JOE ARMSTRONG ROBERT VIRDING BJARNE DÄCKER FEATURING FUNCTIONAL PROGRAMMING
BASED ON A NOVEL BY AGNER KRARUP ERLANG SPECIAL EFFECTS BY ERICSSON SCREENPLAY BY SYNAPSE MOBILE NETWORKS PRODUCTION DESIGN BY KLARNA DIRECTOR OF PRODUCTION MOBILE ARTS EXECUTIVE PRODUCER TAIL-F SYSTEMS
PRODUCED BY CORELATUS CREATIVE DESIGNER PROCESSONE SET MANAGER SJÖLAND & THYSELIUS TESTING SUPERVISOR QUVIQ
ERLANG
tail-f
corelatus
ERICSSON
klarna
process one
Sjöland&Thyselius
O'REILLY
Mobile Arts
QuviQ
synapse mobile networks
Erlang


This is all great, but... how *exactly* do I use Erlang to build a fault-tolerant system?

Learn Three Things

- Create an Erlang process with `spawn()`
- Send a message to a process with `!`
- Process a message with `receive`

Remember this code?

```
%% Saved in 'hello_server.erl'.


-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```

Create the `hello()` process

```
Pid = spawn(fun hello_server:hello/0).
```

```
<0.38.0>
```

Send a message to a process with !

```
Pid ! robert.

robert
```

A little twist

```
Pid ! {self(), robert}.

{<0.31.0>,robert}
```

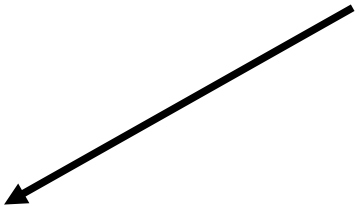
Messages sent to a process with `!`
are received by `receive`

Can you spot the pattern match?

```
%% Saved in 'hello_server.erl'.


-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```

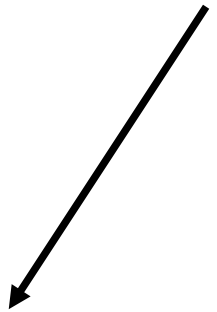
The `{FromPID, Who}` tuple matches

```
%% Saved in 'hello_server.erl'.


-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



Send a message back...

```
%% Saved in 'hello_server.erl'.


-module(hello_server).
-export([hello/0]).


hello() ->
    receive
        {FromPID, Who} ->
            case Who of
                robert -> FromPID ! "Hello Robert.";
                mike -> FromPID ! "Hello Mike.";
                joe -> FromPID ! "Hello Joe.";
                _ -> FromPID ! "I don't know you."
            end,
            hello()
    end.
```



spawn, send (!) then receive

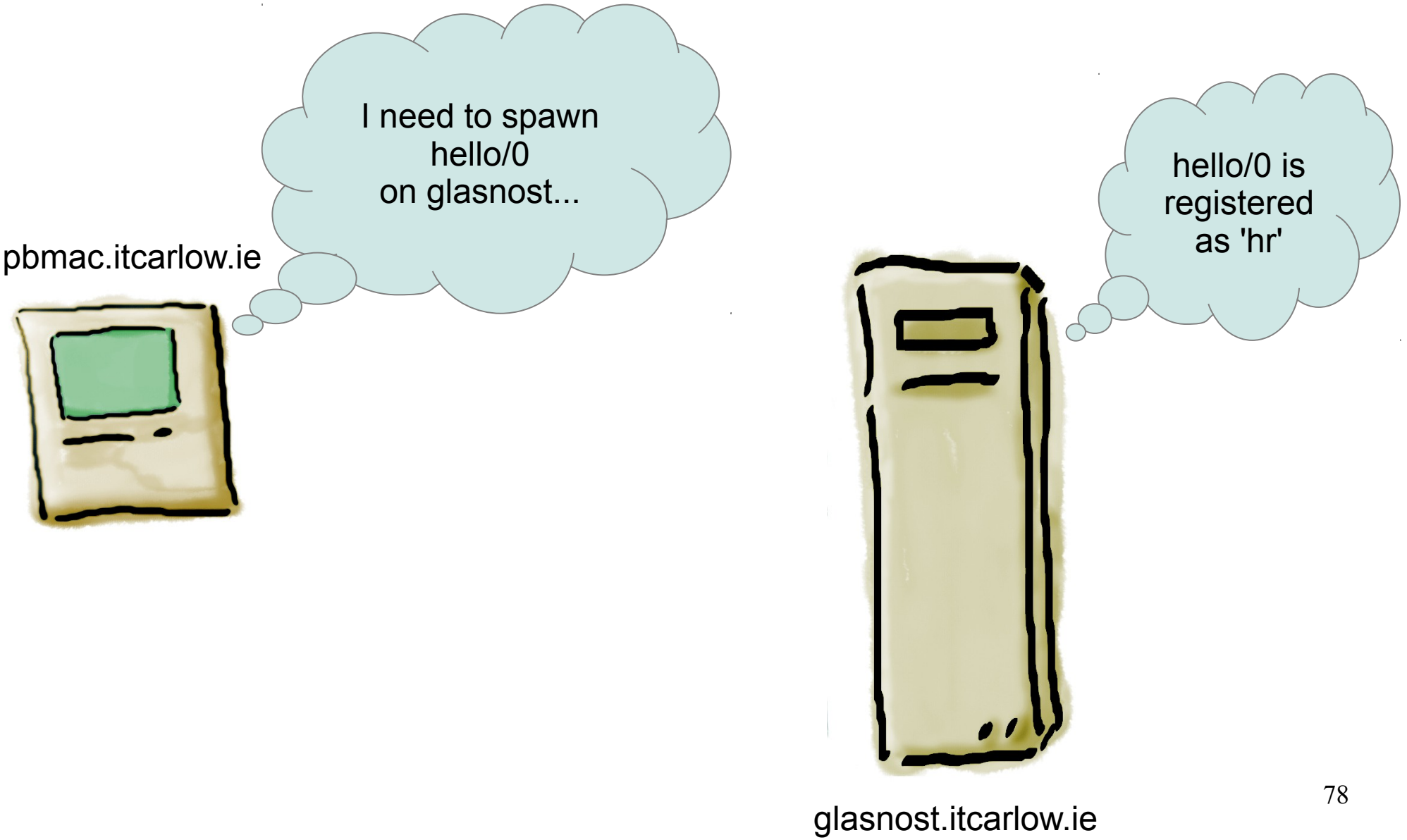
```
Pid = spawn(fun hello_server:hello/0),
Pid ! {self(), robert},
receive
      Response ->
            Response
end.
```

```
"Hello Robert."
```

Things get really interesting
when the processes are
linked together with `spawn_link()`
and the `trap_exit` signal

Processes that are linked together can react appropriately to `EXIT` messages from each other... and what happens next is controlled by the programmer

Distributed Erlang

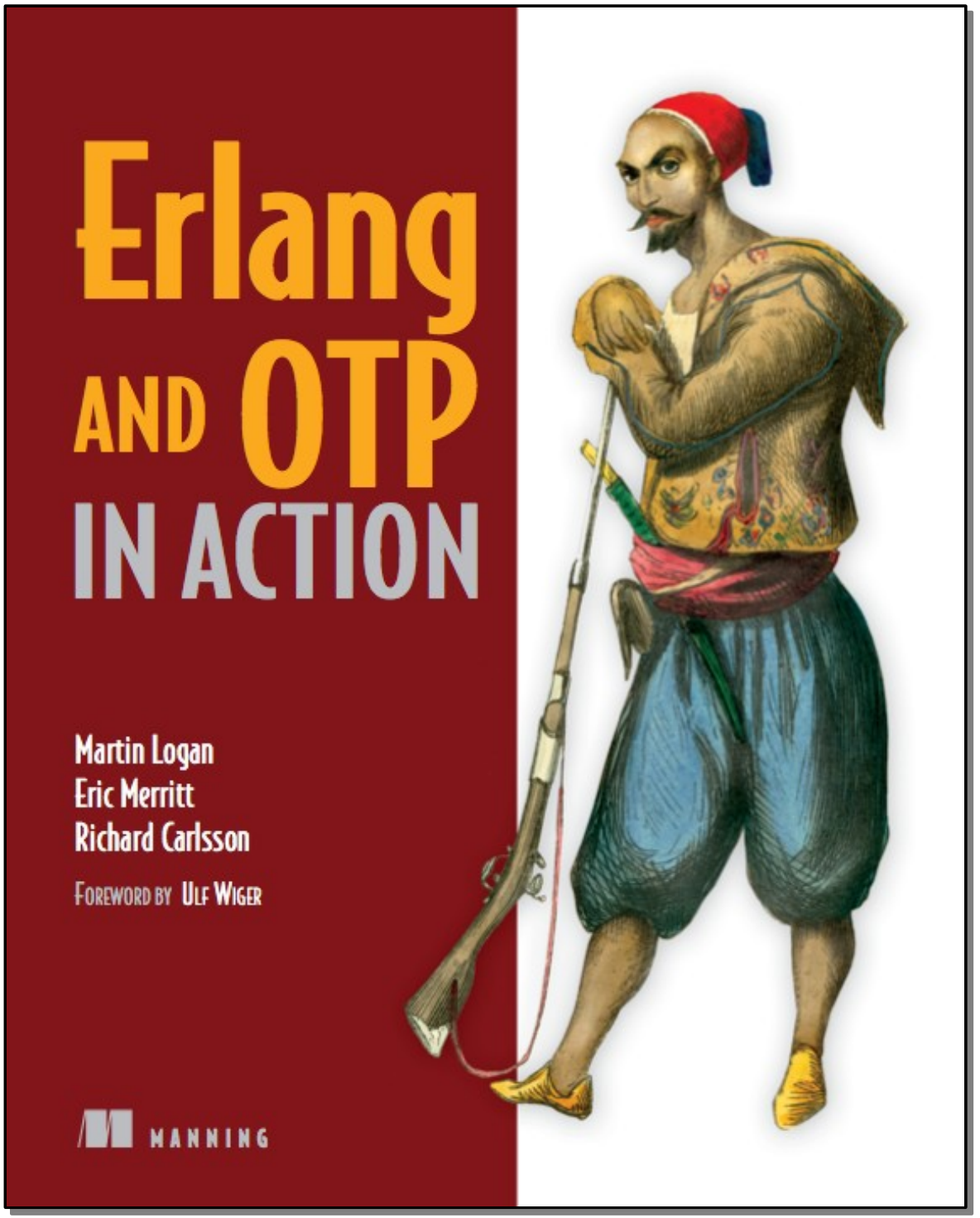


```
Pid = spawn(fun hello_server:hello/0),
```

becomes

```
Pid = spawn('hr@glasnost.itcarlow.ie', hello_server, hello, []),
```

Over time, you'll end up repeating
a lot of your process management code...


Erlang
AND OTP
IN ACTION
Martin Logan
Eric Merritt
Richard Carlsson
FOREWORD BY ULF WIGER
MANNING

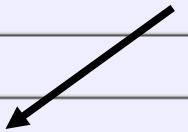
OTP is the jewel in Erlang's crown

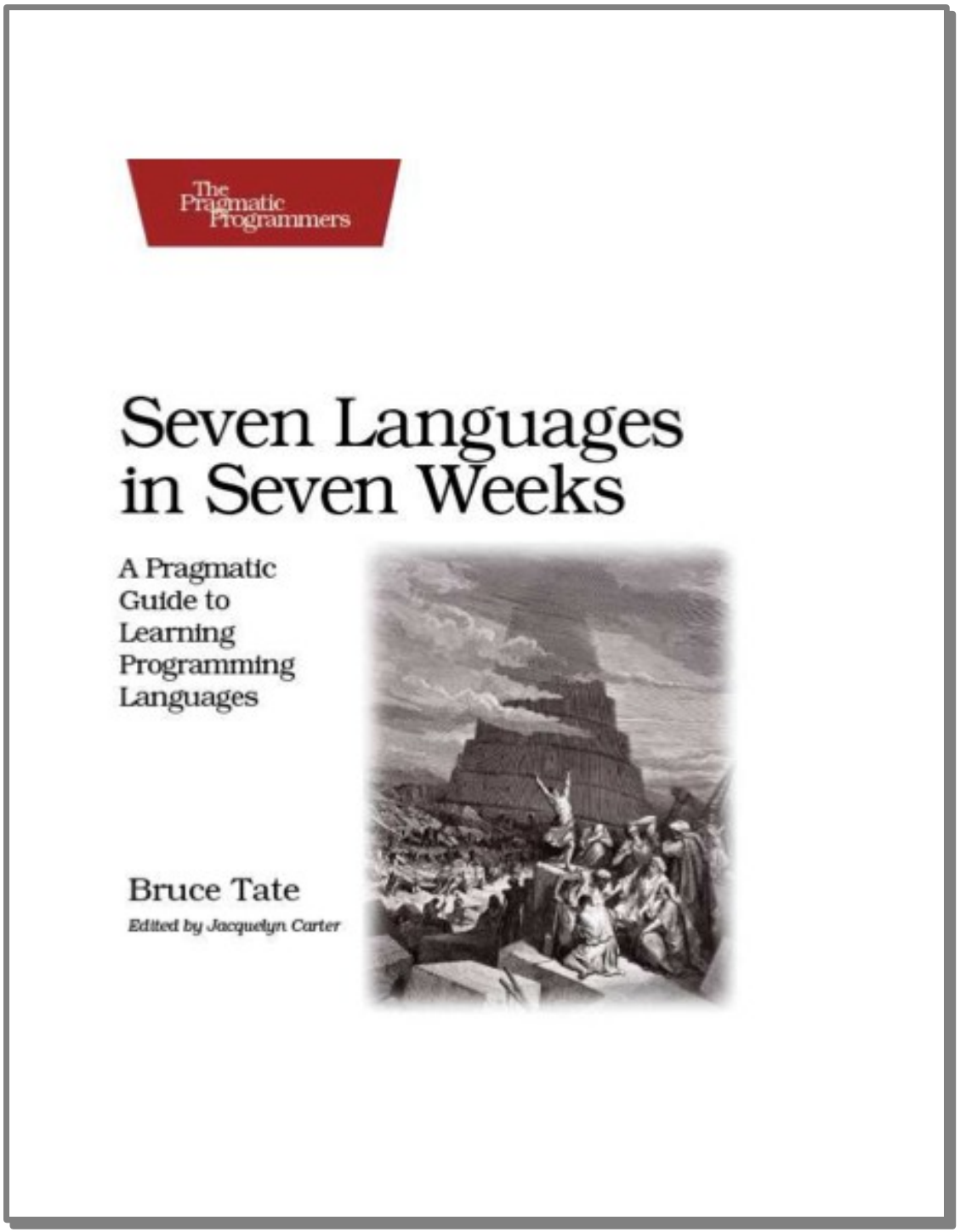
So... if Erlang's so cool, how come no one is using it?

The TIOBE Index

Position Sep 2011	Position Sep 2010	Delta in Position	Programming Language	Ratings Sep 2011	Delta Sep 2010	Status
1	1	=	Java	18.761%	+0.85%	A
2	2	=	C	18.002%	+0.86%	A
3	3	=	C++	8.849%	-0.96%	A
4	6	↑↑	C#	6.819%	+1.80%	A
5	4	↓	PHP	6.596%	-1.77%	A
6	8	↑↑	Objective-C	6.158%	+2.79%	A
7	5	↓↓	(Visual) Basic	4.420%	-1.38%	A
8	7	↓	Python	4.000%	-0.58%	A
9	9	=	Perl	2.472%	+0.03%	A
10	11	↑	JavaScript	1.469%	-0.20%	A
11	10	↓	Ruby	1.434%	-0.47%	A
12	12	=	Delphi/Object Pascal	1.313%	-0.27%	A
13	24	↑↑↑↑↑↑↑↑↑↑↑↑	Lua	1.154%	+0.60%	A
14	13	↓	Lisp	1.043%	-0.04%	A
15	15	=	Transact-SQL	0.860%	+0.09%	A
16	14	↓↓	Pascal	0.845%	+0.06%	A-
17	20	↑↑↑	PL/SQL	0.720%	+0.08%	A--
18	19	↑	Ada	0.682%	+0.01%	B
19	17	↓↓	RPG (OS/400)	0.666%	-0.05%	B
20	30	↑↑↑↑↑↑↑↑↑↑	D	0.609%	+0.20%	B

Position	Programming Language	Ratings
21	Assembly	0.590%
22	MATLAB	0.543%
23	F#	0.512%
24	SAS	0.504%
25	COBOL	0.471%
26	Logo	0.448%
27	Scheme	0.400%
28	R	0.385%
29	C shell	0.383%
30	Fortran	0.372%
31	ActionScript	0.370%
32	Go	0.358%
33	Scratch	0.327%
34	NXT-G	0.327%
35	Haskell	0.325%
36	ABAP	0.320%
37	Forth	0.317%
38	Erlang	0.315%
39	Visual Basic .NET	0.309%
40	Prolog	0.282%
41	APL	0.272%
42	PL/I	0.268%




The Pragmatic Programmers
Seven Languages in Seven Weeks
A Pragmatic Guide to Learning Programming Languages
Bruce Tate
Edited by Jacquelyn Carter

Erlang in Telecoms

Erlang in Data

Erlang on the Web

Erlang in Gaming

*"…**Mochi Media** is the world's largest browser-based games network, with more than 140 million monthly active users and 15,000 games on nearly 40,000 publisher websites…"*

Check out Bob Ippolito's Erlang-Factory talks!

More Erlang in Gaming

DemonWare

Erlang Factory London 2011
http://www.demonware.net/

Games that use us

Call of Duty

DemonWare

Erlang Factory London 2011
http://www.demonware.net/

How we use Erlang

ACTIVISION BLIZZARD

DemonWare

Erlang Factory London 2011
http://www.demonware.net/

Games that use us

...and many more!

ACTIVISION BLIZZARD

*Thanks to **Malcolm Dowse** (of DemonWare, Dublin) for permission to use these images, which were first delivered as part of Malcolm's talk to Erlang Factory London 2011.*


So... is Erlang
worth learning?

Yes

Lots more to discover...

- Fun with `fun`
- Bit-strings and bit syntax
- ETS and DETS
- “Live” code updating
- The Mnesia Distributed Database
- The Standard Library
- CEAN

Friendly Community

`erlang-questions` mailing list

Questions?