Matthew Luke
Peter Lundgren
Thomas Most
Daniel Waters

THE ERLANG PROGRAMMING LANGUAGE

Why Erlang?

- Concurrency
- Hot swap code
- RPC
  - remote = local
- Pattern matching
- Facebook chat
- T-Mobile

About

- Ericsson Computer Science Laboratory
- ‘82: research
- ‘87: experimentation
- ‘88: escape Ericsson
- ‘90: go public at ISS

What is Erlang?

- Functional
- **Strongly, Dynamically** Typed
- Eager, unlike lazy Haskell
- Concurrency oriented via actors
- Fault tolerant
- C Partnership

. , ; , ; , and more! Oh my!

```erlang
% Erlang
blah(true) ->
   foo(),
   bar();
blah(false) ->
   baz().
```

```javascript
// JavaScript
function blah(flag) {
   if (flag) {
      bar();
      foo();
   } else {
      baz();
    }
}
```

- End of function: Period
- End of line: Comma
- End of clause: Semi Colon

More Syntax

- Variables: UPPER CASE
  `BoardWidth = 800.`

- Atoms: lower case
  `monday`

- Send Operator: !

  `main ! quit.`

Example 1: Fib and Fact

```
-module(fact).
-export([fact/1]).

fact(0) -> 1;
fact(N) -> N * fact(N - 1).
```

Example 1: Fib and Fact Cont'd

```
-module(fib).
-export([fib/1]).

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

Primitive Datatypes: Numbers

- Integers
  - `1> 42.`
    `42`
- Floats
  - `2> 3.14.`
    `3.14`
- Characters ($char)
  - `3> $a.`
    `97`
  - `4> $A.`
    `65`
  - `5> $\n.`
    `10`
- Different Base (base#value)
  - base = int 2..36
  - `6> 2#101.`
    `5`
  - `7> 16#1f.`
    `31`

Primitive Datatypes: Atoms

- Named literal
- Start: lowercase letter
- Allowed: a..z, A..Z, 0..9, @, _
- Quoted Atoms
  - 1st letter not lowercase
  - “Not” allowed characters

- `monday`
- `me@host`
- `atoms_are_fun`
- `‘Monday’`
- `‘atom with spaces’`
- `‘pound sign #’`

Primitive Datatypes: Bit Strings

- Untyped memory
-
```
1> Red = 2.
2
```
-
```
2> Green = 61.
61
```
-
```
3> Blue = 20.
20
```
-
```
4> Mem = <<Red:5, Green:6, Blue:5>>
<<23, 180>>
```

Primitive Datatypes: Bit Strings

- 5> <<R1:5, G1:6, B1:5>> = Mem.
  <<23, 180>>
- 6> R1.
  2
- 7> G1.
  61
- 8> B1.
  20

Example from “Programming Erlang” by Joe Armstrong

Primitive Datatypes: Fun

- Functional object
- Lambda!

```
12> Fun1 = fun (X) -> X + 1 end.
#Fun<erl_eval.6.39074546>
```

```
13> Fun1(2).
3
```

Primitive Datatypes: Process IDs

- Process identifier
- `spawn(module, fun, args).`
- ```
  1> Pid = spawn(m, f, []).
  <0.51.0>
  ```
  ```

Primitive Datatypes: Tuple

- Fixed size
- {Term1, …, TermN}

-
```
16> T = {adam,24,{july,29}}.
{adam,24,{july,29}}
```

Primitive Datatypes: List

- Variable length
- [Term1, …, TermN]
- List = [Head | Tail]

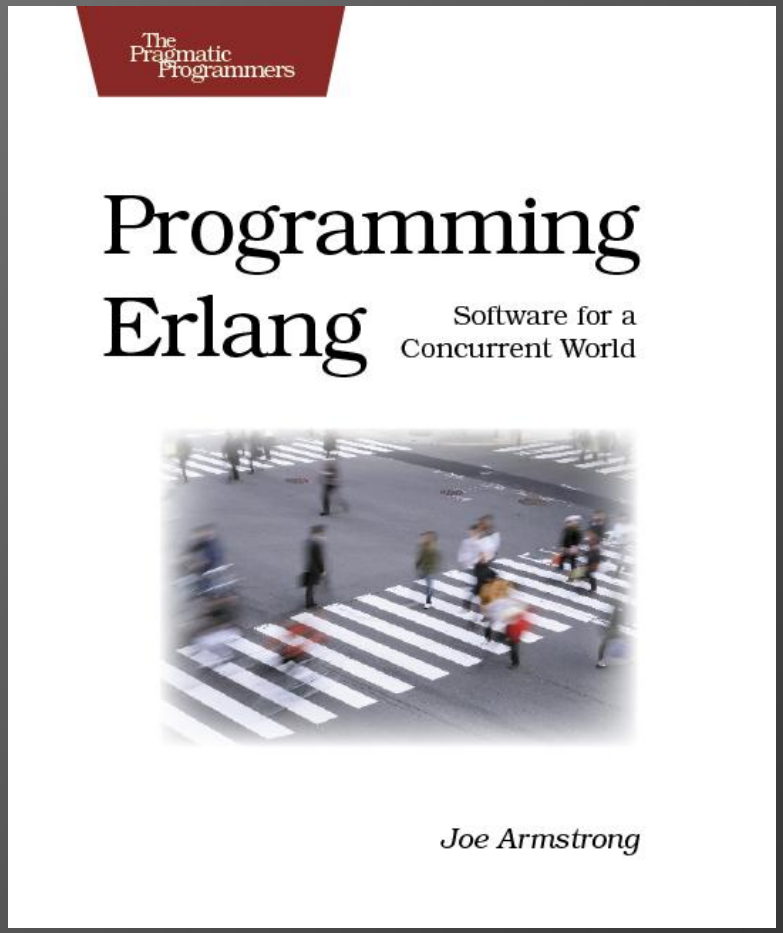
[ ]
[c | [ ]]
[b | [c | [ ]]]
[a | [b | [c | [ ]]]]
[a, b, c]

- 17> L = [a,2,{c,4}].
  [a,2,{c,4}]
- 18> [H | T] = L.
  [a,2,{c,4}]
- 19> H.
  a
- 20> T.
  [2,{c,4}]

Concurrency

- Spawn lots of processes
- Message Passing
- Concurrency example (ex1.erl)

Want to learn more?



http://www.erlang.org/

Interesting Sections:

- Downloads
- Getting started
- Documentation
- Examples

Questions?