Erlang

Jon Bowen

CS 3210

Agenda

- History
- Language Overview and Samples
- Language Evaluation

History

Begin at the end

- **Programs** are structured as concurrent processes
- **Processes** share no memory and communicate with asynchronous message passing
- **Processes** are lightweight
- **Processes** belong to the language and not to the operating system

History

Motivation

- A better way of programming telephony applications
- Requirements:
  - Highly concurrent and distributed
  - Fault-tolerant
  - Real time
  - Highly available
  - Hot swapping
- NOT required: Intensive computation

History

Moments

- 1985: Joe Armstrong hired at Ericsson into exploratory group
- 1985: Work in Smalltalk results in notation close to Prolog. Switch to Prolog
- 1985: Robert Virding incorporates work on parallel logic programming
- 1987: Ericsson groups agrees to use Erlang (implemented in Prolog) in a real problem
- 1988: Rapid change from user feedback results in stabilization by year end

History

Choices

- Design choices by end of 1988:
  - Buffered message reception with pattern matching and out of order handling
  - Error handling abstracted to process level (no difference between hardware and software messages)
  - Explicit links between processes propagate errors.
  - Errors cause all linked processes to die (all alive / all dead)
  - Messages in mailboxes instead of pipes

History

Moments

- End of 1989: initial user group conclusions
  - Programmer-hours to implement new feature in Erlang compared to PLEX reduced by factor of 3 – 25
  - Erlang runtime memory requirements small
  - Erlang compiled code size acceptable
  - Erlang runtime needed to be at least **40** times faster for product development
  - End of 1989: initial user group conclusions
- => Make Erlang faster

History

Moments

- 1990: JAM (Joe’s Abstract Machine) designed by Armstrong. Emulator implemented in C by Mike Williams
- JAM C emulator ~70 faster than Prolog emulator
  - Frequent small garbage collection used
  - Encouraged copying all data involved in message passing
  - Unforeseen, but increased process isolation, concurrency, and construction of distributed systems

History

Moments

- 1993: Erlang book allowed to be published
- 1993: BEAM Compiler
- 1995: Distributed Erlang

History

December 1995: AXE-N Collapse

- Erlang selected as language in reboot of major Ericsson project
- From lab experiment to complete platform: Open Telecom Platform (OTP)
  - Extensive and well-tested libraries
  - Design patterns for common applications
  - Documentation
  - Philosophy and learning support
  - Mnesia DBMS and query language
  - Binaries in reference-counted storage area
  - HiPE compiler

History

Pushed Into the World

- Projects with Erlang produce excellent results
- 1998…Ericsson bans use of Erlang internally
- Development group convinces Ericsson to release Erlang and OTP as open source
- Development group promptly quits, forms new company, and delivers first product 6 months later

The Language

Functional language?

- Additional practical needs from being used in massive production systems
- "Erlang is not a strict side-effect-free functional language but a concurrent language where what happens inside a process is described by a simple functional language"
- Example:
  - If two different processes receive a Pid representing a file, both are free to send messages to the file process in any way they like. It is up to the logic of the application to prevent this from happening

The Language

27 Total Keywords

after	bxor	not
and	case	of
andalso	catch	or
band	cond	orelse
begin	div	receive
bnot	end	rem
bor	fun	try
bsl	if	when
bsr	let	xor

http://erlang.org/doc/reference_manual/introduction.html

The Language

Common Data Types

- Number `1, 3.14, -41`
- Atom `inch, monday, bob`
- Tuple `{inch, 1, “abc”}`
- List `[1, 1, 2, 3, 5, 7]`
- Map `#{name => “Bob”}`
- Functions

The Language

Of Note

- Variables begin with a capital letter or _
- Variables can only be bound once per scope
- Assignment operator, =, is better though of as a matching assertion
- Lists can be split with the | operator
- Pattern matching is used in parameter passing and “assignment”

The Language

Samples

```
>> io:fwrite("Hello World!~n").
Hello World!
ok
>>
>> L = [1, 2, 3, "go"].
>> [S | T] = L.      % S = 1, T = [2,3,”go”]
>>
>> [A, B | R] = L.  % A = 1, B = 2, R = [3, “go”]
```

The Language

Samples

```
-module(demo).
-export([listOp/2]).

listOp(L, Type) ->
    if
        Type == sum ->
            lists:sum(L);

        Type == prod ->
            lists:foldl(fun (X, P) -> X * P end, 1, L);

        Type == squareEach ->
            lists:map(fun (X) -> X * X end, L);

        true ->
            "Operation not supported"
    end.
```

The Language

Samples

Parameter:

`[{"Denver",{f,70}},{"Seattle",{f,65}},{"London",{c,20}}]`

```
format_temps([]) ->
    ok;
format_temps([City | Rest]) ->
    print_temp(convert_to_celsius(City)),
    format_temps(Rest).

convert_to_celsius({Name, {c, Temp}}) -> % No conversion needed
    {Name, {c, Temp}};
convert_to_celsius({Name, {f, Temp}}) -> % Do the conversion
    {Name, {c, (Temp - 32) * 5 / 9}}.

print_temp({Name, {c, Temp}}) ->
    io:format("~-15s ~.1f c~n", [Name, Temp+0.0]).
```

The Language

Samples

```
fib(N) -> fibPass(N, 0, 1).

fibPass(0, Result, _Next) -> Result;

fibPass(Iter, Result, Next) when Iter > 0 ->
    fibPass(Iter-1, Next, Result + Next).

test(Fibnum) ->
    Time = element(1, timer:tc(fib,fib,[Fibnum])) * 10.0e-6,
    io:fwrite("~w took ~.5f seconds.~n", [Fibnum, Time]).
```

```
>> fib:test(1000).
1,000 took 0.00000 seconds.
>> fib:test(10000).
10,000 took 0.00000 seconds.
>> fib:test(100000).
100,000 took 1.25000 seconds.
>> fib:test(1000000).
1,000,000 took 250.38000 seconds.
```

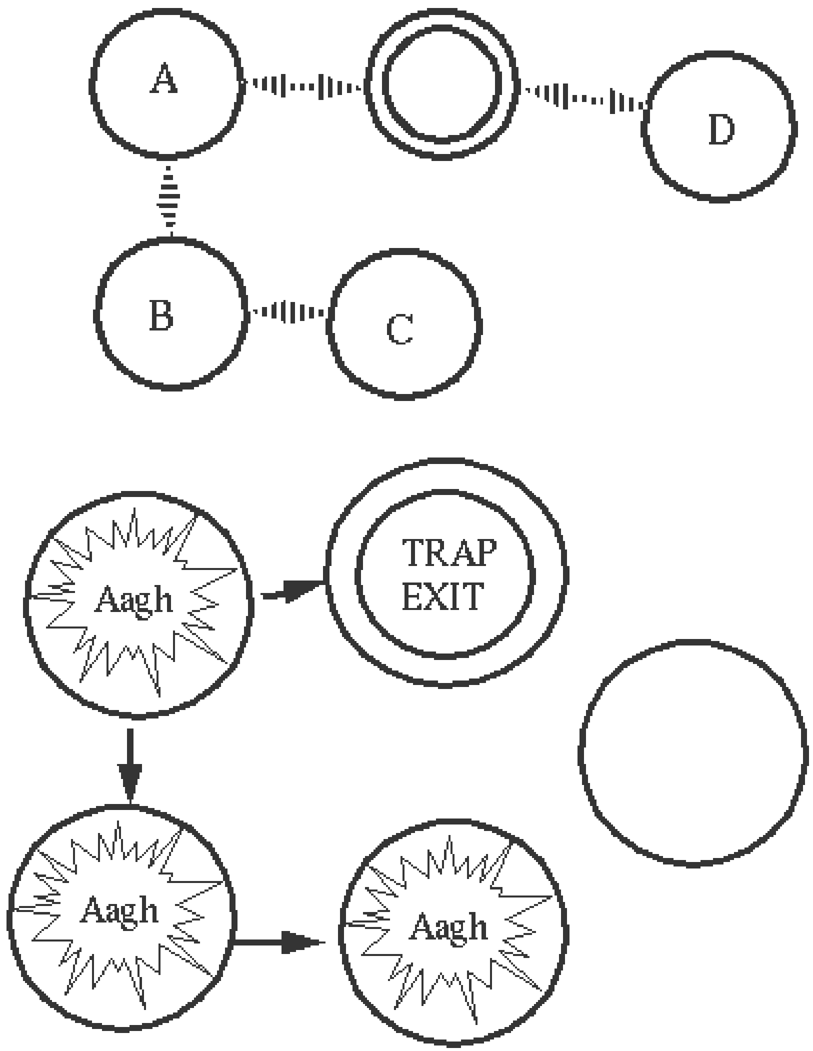
The Language

Samples

```
Pong_PID = spawn(MyMod, MyFunc, Args)
%-------------------------------------------------------------
ProcessId ! “hello...can you hear me?”
%-------------------------------------------------------------
ping(Pong_PID) ->
    receive
        pong ->
            io:format("Ping received~n")
    after
        5000 ->
            error()
    end.
    Pong_PID ! {ping, self()}.
```

The Language

Error Handling



Language Evaluation

Readability / Writability

- Strengths
  - Small size of language
  - Typically one best way to accomplish something
  - Language encourages standard structure
- Weaknesses
  - No or limited type checking
  - Data structures can look very complex
  - Piecewise function definition can be abused

Language Evaluation

Reliability

- Strengths
  - Immutable data and no references
  - Processes to do work and processes to supervise
  - Message passing uses mailboxes
  - Error handling approach
- Weaknesses
  - Lack of type checking can cause runtime errors
  - All or nothing security

Language Evaluation

Cost

- Strengths
  - Small language reduces learning curve
  - Reliability directly reduces maintenance cost
  - Code changes can be applied “on the fly”
  - Concurrency + distributed = massively scalable
- Weaknesses
  - Smaller language community
  - Slower than alternatives for serial operations with high CPU use

Standard References

- Armstrong, Joe. *Programming Erlang: Software for a Concurrent World*. Pragmatic Programmers LLC, 2013.
- St. Laurent, Simon. *Introducing Erlang: Getting Started in Functional Programming 2nd Ed*. O’Reilly Media, Inc., 2017
- Herbert, Fred. *Learn You Some Erlang for Great Good!: A Beginner's Guide*. No Starch Press, Inc., 2013.
- Armstrong, Joe. “A History of Erlang.” 19 Mar. 2018, http://webcem01.cem.itesm.mx:8005/erlang/cd/downloads/hopl_erlang.pdf.
- “An Erlang Course.” Erlang/OTP unit at Ericsson, 2 Apr. 2018, https://www.erlang.org/course.
- Erlang Homepage. Erlang/OTP unit at Ericsson, 2 Apr. 2018, https://www.erlang.org.
- “Documentation.” Erlang/OTP unit at Ericsson, 2 Apr. 2018, https://www.erlang.org/docs