Erlang

Joey Erskine, Matt Estabrook, Brad Tagtow

Early History and Overview

- Development began in 1986 at the Ericsson Computer Science Lab
- Named after Agner Erlang (mathematician)
- Began as an experiment to develop a telecom language
  - Tried 20+ languages
  - Initially whittled down to Prolog, Lisp, variants
  - Needed to add concurrency, not found in Prolog, etc
  - Solution: Take ideas from Prolog, Lisp, etc, add concurrency (New language!)

Features of Erlang

- Designed for writing concurrent programs that can “run forever”
- Each process has its own separate memory, so communication is done through asynchronous message passing
- Processes are lightweight and belong to the language (VM), not the OS
- Dynamic type checking

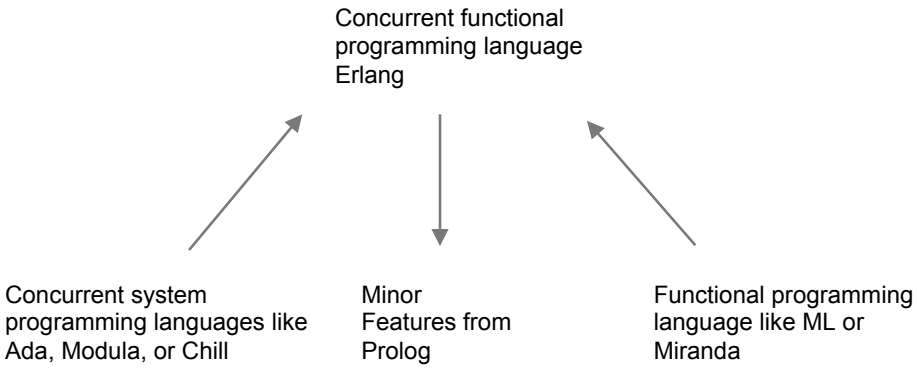
Feature of Erlang (continued)

- Error handling
  - Does not share resources between processes
  - All processes must have enough local information to continue running if something fails in another part of the system

Concurrency

- Processes run “independent” of one another
  - E.g., no sharing of resources, each process has its own separate memory
- Communicate by passing “messages” to one another
- Run inside the Erlang VM, not the OS

Erlang Today



Memory Management

- Explicit processes are part of the VM and not part of the underlying OS
- Has no pointers and uses a garbage collectible memory
  - This makes it “impossible” for any process to corrupt the memory of another
  - Results in memory requirements for individual processes being very small and all processes can safely be stored in the same address space

Garbage Collection

- Per process (makes stops very short)
- Mark & sweep GC
- Uses frequent small garbage collections instead of infrequent larger collections that take more time
- Can specify heap size
- Atom table not garbage collected

(Sort of) Variables in Erlang

- Must begin with a capital letter
  - Lowercase signifies an atom
- Cannot re-assign new value to an existing variable

```
1> Var.
* 1: variable 'Var' is
unbound
2> Var = 5.
5
3> Var = 7.
** exception error: no match
of right hand side value 15
```

Booleans in Erlang

- `true` and `false` (represented by atoms)
- `and`, `or` operators
  - Evaluate both sides
- `andalso`, `orelse` operators
  - Evaluate right-hand expression only if necessary

```
1> true and false.
false
2> false or true.
true
3> not false.
true
4> not (true and true).
false
```

Strings in Erlang

- The most hated thing in Erlang!
- Designed for communications firmware (no need for strings!)
- Represented by lists of ASCII values

```
1> "cat".
"cat"
2> [99, 97, 116].
"cat"
3> [99, 97, 116, 1].
[99, 97, 116, 1]
```

Lists in Erlang

- Hold pieces of data
  - Can be heterogeneous
- Overall, lists in Erlang are same as lists in Scheme

```
1> hd([1, 2, 3]).
1
2> tl([1, 2, 3]).
[2,3]
3> hd(tl([1, 2, 3, 4])).
2
4> [1, 2, 3] ++ [4, 5, 6].
[1,2,3,4,5,6]
5> [1, 2, 3, 4] -- [1, 3].
[2,4]
```

Lists in Erlang (cont.)

- Built in splitting into head/ tail
  - [Head|Tail]
  - | operator joins and splits
- Tuples
  - Similar to lists
  - But built using arrays (indexing is faster than lists)

```
1> List1 = [1, 2, 3, 4].
[1,2,3,4]
2> [Head|Tail] = List1.
[1,2,3,4]
3> Head.
1
4> Tail.
[2,3,4]
5> [5 | [4]].
[5,4]
6> [2 | [1 | [5, 3, 3]]].
[2,1,5,3,3]
```

Hello, world!

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

%% Method which prints 'Hello,
 world!'
greeting() ->
    io:format("Hello, world!~n").
```

```
1> cd("/Examples").
/Examples
ok
2> c(hello).
{ok,hello}
3> hello:greeting().
Hello, world!
ok
```

Hailstone

```
-module(hailstone).
-export([main/1]).

main(N) when N == 1 -> [1];
main(N) when N rem 2 == 1 ->
    [N|main(N * 3 + 1)];
main(N) when N rem 2 == 0 ->
    [N|main(N div 2)].
```

Who's using Erlang?

- WhatsApp
  - Serves over 465 million users.
  - 40 billion messages sent to it, and 19 billion messages sent out
    - Company has scaled drastically, maintained uptime of 99%.
    - How?
- Facebook, Amazon, Yahoo!, T-Mobile, Motorola

Real Example

Number: 858-433-4230

Text 1 if you are a senior.
Text 2 if you are a junior.

Sources

[1] Thanks to http://learnyousomeerlang.com/starting-out-for-real#numbers for great tutorials and code examples

[2] Thanks to http://www.cis.upenn.edu/~matuszek/General/ConciseGuides/concise-erlang.html for information on variables

[3] Thanks to "A History of Erlang" by Joe Armstrong for an overview of Erlang and information regarding the history, objective, and features including memory management and garbage collection

[4] Thanks to http://prog21.dadgum.com/16.html for an overview of Erlang garbage collection

[5] Thanks to http://www.erlang.org/doc/ for official Erlang documentation

[6] Thanks to http://www.erlang.org/faq/academic.html for basic Erlang history

[7] Thanks to http://erlang.2086793.n4.nabble.com/Tuples-vs-lists-td2087352.html for tuple clarification

[8] Thanks to http://www.huffingtonpost.com/2014/03/11/erlang-from-whatsapp-to-o_n_4944667.html?utm_hp_ref=huffpost-code for describing companies using Erlang

[9] Thanks to http://rosettacode.org/wiki/Hailstone_sequence#Erlang for a hailstone example