Introduction to Erlang
A Language For Message-Passing Concurrency

David Chisnall

March 3, 2011
Erlang? Isn’t This Module About C?

- Erlang has a fairly simple and flexible syntax for message passing
- Trivial to write very scalable Erlang programs
- Erlang techniques are applicable elsewhere
Syntax In a Nutshell

Very Prolog-like:

- Variables start with upper case.
- Atoms start with lower case.
- Commas separate sequential statements
- Semicolon separates ‘choice’ statements
- Full stop as a terminator
Syntax In a Nutshell II

- Functions defined with arrow (~)
- Function-like syntax use for all control structures (if, case, etc)
- Last line is the return value (no explicit return)
Basic Principles

- Single Assignment
- Pattern Matching
- Small Processes
- Message Passing
Single Assignment

• Variables can only have one value
• Similar to Prolog

$ erl
1> A = 1.
1
2> A = 2.
** exception error: no match of right hand side value 2
Pattern Matching

- Anonymous Variable _ matches anything.
- Constants match themselves.
- Variables are instantiated.

```erlang
-module(calc).
-export([evaluate/3]).

evaluate(add,A,B) ->
    A + B
;
evaluate(subtract,A,B) ->
    A - B.
```
Process Creation

- `spawn()` creates new Erlang processes
- Erlang processes are much cheaper than system processes
- Can be created on remote nodes

```
Pid = spawn(Node, Module, Function, Args).
```
Message Passing

- Send message to process
- Asynchronous
- Can send anything, even process IDs.

Pid ! Message.
Message Passing II

- Message passing is only useful if you can receive them
- Receive uses pattern matching

```erlang
receive  
    {ping, {Sender, Sent}} -> 
      Sender ! {ack, {Sent, now()}}
    ;  
    Error -> 
      io:format("Invalid Message received ~w~n", [Error])
end.

Responder ! {ping, {self(), now()}}
receive 
    {ack, {Sent, Received}} -> 
      logRoundTrip(Sent, Received}
end
```
Erlang Data Structures

- Tuples
- Lists
Tuples

- Fixed number of elements
- Can contain anything, including other tuples.
- Like C structures, but elements are anonymous
- Built in functions allow accessing

\[
A = \{1, 2, \{\text{elephant}, B\}, 12.5, "aardvark." , \text{Pid}\}.
\]
Lists

• Non-fixed number of elements
• Good for recursion
• Used to implement strings
• Can be converted to tuples

1> A = [1,2,3].
[1,2,3]
2> B = [0|A].
[0,1,2,3]
3> C = B ++ [4].
[0,1,2,3,4]
The Bit Syntax

- Converts between Erlang types and binary objects
- Great for network code
- Works with pattern matching

1> A = <<1,2,3,4,5,6>>.
   <<1,2,3,4,5,6>>
2> <<B:16/big,C:32/little>> = A.
   <<1,2,3,4,5,6>>
3> B.
   258
4> C.
   100992003
A Note on Tail-Recursion

- Tail recursion is a very common idiom in Erlang
- Potentially infinite recursion is allowed, which needs an infinite stack
- This is solved by re-using the stack frame of a function that returns the result of another function
The Fibonacci Sequence

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

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

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

qs([]) -> [] ;
qs([X]) -> [X] ;
qs(List) ->
    [Pivot|Sublist] = List,
    {Less, Greater} =
        partition(Sublist, Pivot, [], []),
    qs(Less) ++ [Pivot] ++ qs(Greater).
```
Why Erlang?  
Erlang Overview  
Concurrency  
Data Structures  
Some Examples

Quicksort Partition Function

```erlang
partition([],_,Less, Greater) -> {Less, Greater};
partition([X|List],Pivot, Less, Greater) ->
    if
        X > Pivot ->
            partition(List, Pivot, Less, [X|Greater])
        ;
        true ->
            partition(List, Pivot, [X|Less], Greater)
    end.
```

Parallel Quicksort

\[\text{pq} (\text{List}, \text{Parent}, \text{Tag}) \rightarrow \text{Parent} ! \{\text{Tag}, \text{List}\} ;\]

\[\text{pq} (\text{List}, \text{Parent}, \text{Tag}) \rightarrow \text{Parent} ! \{\text{Tag}, \{\text{List}\}\} ;\]

\[\text{pq} (\text{List}, \text{Parent}, \text{Tag}) \rightarrow\]

\[\text{[Pivot | Sublist]} = \text{List}, \]

\[\{\text{Less, Greater}\} = \text{partition(Sublist, Pivot, [\text{Less}], [\text{Greater}]}, \]

\[\text{spawn(pq, pq, [Less, self(), less], spawn(pq, pq, [Greater, self(), greater], receive {less, LessSorted} -> true end, receive {greater, GreaterSorted} -> true end, Parent ! {Tag, LessSorted ++ [Pivot] ++ GreaterSorted}}.}\]
Using Erlang

- Not installed on the lab machines
- Installed on SUCS
- Download from http://www.erlang.org/
Questions?