Erlang

Concurrent Programming Language and Runtime System

Lucian Pârvu
lucian@erlang.ro

June 22nd, 2012

Functional Programming Summer School
Politehnica University of Bucharest
Computer Science Department

Agenda

- History
- Why Erlang was created?
- Who is using Erlang
- Sequential Erlang
- Concurrent Programming in Erlang
- OTP - Open Telecom Platform
- Distributed Erlang Applications
- Debugging in Erlang
- Advantages of using Erlang
- Issues & Risks
- OpenSource Projects developed in Erlang

Who was Erlang?

Agner Krarup Erlang
1878 - 1929

Danish mathematician, statistician and engineer, who invented the fields of traffic engineering and queueing theory.

He developed his theory of telephone traffic over several years.

His significant publications include:

1909 - "The Theory of Probabilities and Telephone Conversations" - which proves that the Poisson distribution applies to random telephone traffic.

1917 - "Solution of some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges" - which contains his classic formulae for loss and waiting time.

Source: http://en.wikipedia.org/wiki/Agner_Krarup_Erlang

Erlang Unit

An Erlang is a unit of telecommunications traffic measurement.

An Erlang represents the continuous use of one voice path. In practice, it is used to describe the total traffic volume of one hour.

```
Minutes of traffic in the hour = number of calls x duration
                               = 30 x 5 = 150

Hours of traffic in the hour   = 150 / 60 = 2.5

Traffic figure                 = 2.5 Erlangs
```

Source: http://www.erlang.com/whatis.html

History

1982 - 1985
Experiments with programming of telecom using > 20 different languages. Conclusion: The language must be a very high level symbolic language in order to achive productivity gains ! (Leaves us with: Lisp , Prolog , Parlog ...)

1985 - 86
Experiments with Lisp,Prolog, Parlog etc. Conclusion: The language must contain primitives for concurrency and error recovery, and the execution model must not have back-tracking. (Rules out Lisp and Prolog.) It must also have a granularity of concurrency such that one asyncronous telephony process is represented by one process in the language. (Rules out Parlog.) We must therefore develop our own language with the desirable features of Lisp, Prolog and Parlog, but with concurrency and error recovery built into the language.

1987
The first experiments with Erlang.

1988
ACS/Dunder Phase 1. Prototype construction of PABX functionality by external users Erlang escapes from the lab!

1989
ACS/Dunder Phase 2. Reconstruction of 1/10 of the complete MD-110 system. Results: >> 10 times greater gains in efficency at construction compared with construction in PLEX!
Further experiments with a fast implementation of Erlang.

1990
Erlang is presented at ISS'90, which results in several new users, e.g Bellcore.

Source: http://www.erlang.org/course/history.html

History

1991 Fast implementation of Erlang is released to users. Erlang is represented at Telecom'91 . More functionality such as ASN1 - Compiler , graphical interface etc.

1992 A lot of new users, e.g several RACE projects. Erlang is ported to VxWorks, PC, Macintosh etc. Three applications using Erlang are presented at ISS'92. The two first product projects using Erlang are started.

1993 Distribution is added to Erlang, which makes it possible to run a homogeneous Erlang system on a heterogeneous hardware. Decision to sell implementations Erlang externally. Separate organization in Ericsson started to maintain and support Erlang implementations and Erlang Tools.

1995 OTP Unit is formed - 60 engineers 1997 OTP Unit responsible for the distribution of Erlang

1998 (Feb) The rst demo of GPRS developed in Erlang
(Feb) Erlang was banned inside Ericsson Radio System.
(Mar) AXD301 was announced. This was possibly the largest ever program in a functional language.
(Dec) Open Source Erlang was released
(Dec) Most of the group that created Erlang resigned from Ericsson and started a new company called Bluetail AB.

2000 Bluetail was acquired by Alteon Web and six days later Alteon was acquired by Nortel Network

2002-present Joe Armstrong joins SICS

2007 “Programming Erlang - Software for a Concurrent World” - Joe Armstrong - is published

2009 Erlang Code moving to GitHub

Source: http://www.erlang.org/course/history.html

Why Erlang was created

What if I need to build a product that needs to follow these characteristics:

- highly concurrent an distributed systems
- thousands of simultaneous transactions
- support cluster architectures and changes in the cluster topology at runtime
- support may OS's – Solaris, VxWorks, Windows, Linux, Mac OS X
- support 32bit, 64bit with SMP support
- no down time
  (actually uptime of 99.999% which means maximum downtime of 5 minutes / year)
- highly "expressive" programming language
- recovery from software errors
- recovery from hardware errors
- trace & debug code at runtime
- update code at runtime

Source: Mike Williams - http://www.erlang.se/workshop/2003/paper/mikesrationale.pdf

Who is using Erlang?

Who uses Erlang for product development?
http://www.erlang.org/faq/introduction.html#id49610

Aptela (VoIP Service Provider)
Bluetail/Alteon/Nortel/Avaya (distributed, fault tolerant VPN Gateway)
Corelatus (SS7 monitoring).
dqdp.net (in Latvian) (Web Services).
Facebook (Facebook chat backend)
Finnish Meteorological Institute (Data acquisition and real-time monitoring)
IDT corp. (Real-time least-cost routing expert systems)
IEISS. (Electronic financial instrument exchange software)
Klarna (Electronic payment systems)
Lindenbaum (Large scale voice conferencing)
M5 Networks (VoIP Services)
Mobilearts (GSM and UMTS services)
Netkit Solutions (Network Equipment Monitoring and Operations Support Systems)
Process-one (Jabber Messaging)

Who is using Erlang?

Quviq (Software Test Tool)
RabbitMQ (AMQP Enterprise Messaging)
Schlund + Partner (Messaging and Interactive Voice Response services)
Smarkets (Betting exchange and prediction market)
T-Mobile (previously one2one) (advanced call control services)
Telia (a telecomms operator)
Textendo (Innovative text messaging services)
Vail Systems (Interactive Voice Response systems)
Wavenet (SS7 and IVR applications)

and also:

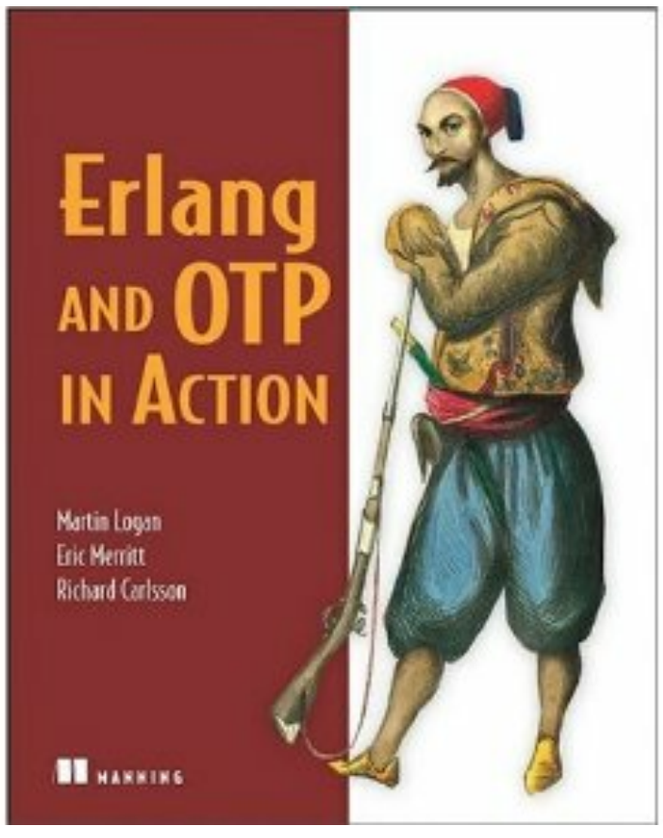
Amazon (SimpleDB)
Basho (Riak)
Delicious
Ericsson
MochiMedia
...

Books

2007

2009

2010



Tutorial

Learn You Some Erlang for Great Good!

http://learnyousomeerlang.com/

Basic Erlang

- The Erlang Shell
- Variables
- Data Types
- Pattern Matching
- Functions
- Modules

The Erlang Shell

```
Last login: Wed Jun 20 16:52:16 on ttys002
dna:~ lucian$ erl
Erlang R13B02 (erts-5.7.3) [source] [64-bit] [smp:2:2] [rq:2] [async-threads:0] [kernel-poll:false]

Eshell V5.7.3  (abort with ^G)
1> pwd().
/Users/lucian
ok
2> regs().

** Registered procs on node nonode@nohost **
Name                  Pid          Initial Call                      Reds Msgs
application_controlle <0.5.0>      erlang:apply/2                     444    0
code_server           <0.18.0>     erlang:apply/2                  110165    0
erl_prim_loader       <0.2.0>      erlang:apply/2                  171231    0
error_logger          <0.4.0>      gen_event:init_it/6                220    0
file_server_2         <0.17.0>     file_server:init/1                 202    0
global_group          <0.16.0>     global_group:init/1                 60    0
global_name_server    <0.11.0>     global:init/1                       53    0
inet_db               <0.15.0>     inet_db:init/1                    2873    0
init                  <0.0.0>      otp_ring0:start/2                 2566    0
kernel_safe_sup       <0.27.0>     supervisor:kernel/1                 57    0
kernel_sup            <0.9.0>      supervisor:kernel/1               1169    0
rex                   <0.10.0>     rpc:init/1                          36    0
standard_error        <0.20.0>     standard_error:server/2              7    0
standard_error_sup    <0.19.0>     supervisor_bridge:standar           40    0
user                  <0.23.0>     group:server/3                      38    0
user_drv              <0.22.0>     user_drv:server/2                  574    0

** Registered ports on node nonode@nohost **
Name                  Id              Command
ok
3>
```

Variables

- All variable names must start with an uppercase letter.
- Erlang has single assignment variables - can be given a value only once. If you try to change the value of a variable once it has been set, then you'll get an error.
- A variable that had a value assigned to it is called a bound variable; otherwise, it is called an unbound variable.

```
1> X = 1. 12345
2> X*2. 2
3> X = 2.
** exception error: no match of right hand side value 2
```

Numbers, Atoms, Tuples

Floating-Point Numbers - must have a decimal point followed by at least one decimal digit. When you divide two integers with "/", the result is automatically converted to a floating-point number.

```
1> 5/3.
1.66667
2> 4/2.
2.00000
```

Atoms are global. Atoms start with lowercase letters, followed by a sequence of alphanumeric characters or the underscore ( ) or at () sign.

```
router, device, dna@itcnetworks, nmos_domain
```

Tuples :

```
1> Workstation = {workstation, {dns, dna}, {domain, erlang}, {uptime, 200}, {os, macosx}}.
```

Pattern Matching

```
-define(IP_VERSION, 4).
-define(IP_MIN_HDR_LEN, 5).

DgramSize = size(Dgram),
case Dgram of
    <<?IP_VERSION:4, HLen:4, SrvcType:8, TotLen:16,
      ID:16, Flgs:3, FragOff:13,
      TTL:8, Proto:8, HdrChkSum:16,
      SrcIP:32,
      DestIP:32, RestDgram/binary>> when HLen>=5, 4*HLen=<DgramSize ->
        OptsLen = 4*(HLen - ?IP_MIN_HDR_LEN),
        <<Opts:OptsLen/binary,Data/binary>> = RestDgram,
    ...
end.
```

Functions

```erlang
%% Code from
%%   Erlang Programming
%%   Francecso Cesarini and Simon Thompson
%%   O'Reilly, 2008
%%   http://oreilly.com/catalog/9780596518189/
%%   http://www.erlangprogramming.org/
%%   (c) Francesco Cesarini and Simon Thompson

%% Chapter 2

-module(chapter2).
-export([area/1,area2/1,flatten/1,factorial/1]).

-import(math, [sqrt/1]).

% To calculate the area of a shape (section 2.12)

area({square, Side}) ->
  Side * Side ;
area({circle, Radius}) ->
  math:pi() * Radius * Radius;
area({triangle, A, B, C}) ->
  S = (A + B + C)/2,
  math:sqrt(S*(S-A)*(S-B)*(S-C));
area(Other) ->
  {error, invalid_object}.

% Variant of area using the  sqrt/1 imported from the math
module.

area2({square, Side}) ->
  Side * Side ;
area2({circle, Radius}) ->
  math:pi() * Radius * Radius;
area2({triangle, A, B, C}) ->
  S = (A + B + C)/2,
  sqrt(S*(S-A)*(S-B)*(S-C));
area2(Other) ->
  {error, invalid_object}.

% To flatten a 3D object into a 2D object.

flatten(Other)  -> {error, unknown_shape};
flatten(cube)   -> square;
flatten(sphere) -> circle.

% The factorial function.

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

Source: Erlang Programming - Ch. 2

Modules

```
%% Code from
%%   Erlang Programming
%%   Francecso Cesarini and Simon Thompson
%%   O'Reilly, 2008
%%   http://oreilly.com/catalog/9780596518189/
%%   http://www.erlangprogramming.org/

%% Demo module from Chapter 2

-module(demo).
-export([double/1]).

% This is a comment.
% Everything on a line after % is ignored.

double(Value) ->
  times(Value, 2).
times(X,Y) ->
  X*Y.

%% Note: times/2 is deliberately omitted from the export list.
```

Source: Erlang Programming - Ch. 2

Sequential Erlang

- Conditional Evaluations
- Guards
- Built-in Functions
- Recursion
- Handling Errors

Conditional Evaluations

```erlang
%% Conditional Evaluations

% To convert an atom representing a day into an integer.

convert(Day) ->
  case Day of
    monday    -> 1;
    tuesday   -> 2;
    wednesday -> 3;
    thursday  -> 4;
    friday    -> 5;
    saturday  -> 6;
    sunday    -> 7;
    Other     -> {error, unknown_day}
  end.

% Calculating the length of a list.

listlen([])     -> 0;
listlen([_|Xs]) -> 1 + listlen(Xs).

% Rewrite this directly using a case expression:

listlen2(Y) ->
  case Y of
    []     -> 0;
    [_|Xs] -> 1 + listlen2(Xs)
  end.

% Indexing into a list, i.e. looking for the nth element of a
% list (with numbering from zero).

index(0,[X|_])           -> X;
index(N,[_|Xs]) when N>0 -> index(N-1,Xs).

% Defining an index function using a case expression.

index2(X,Y) ->
   index({X,Y}).

index(Z) ->
  case Z of
    {0,[X|_]}           -> X;
    {N,[_|Xs]} when N>0  -> index2(N-1,Xs)
  end.
```

```erlang
% Defining an index function using nested case expressions.

index3(X,Y) ->
  case X of
    0 ->
      case Y of
        [Z|_]    -> Z
      end;
    N when N>0 ->
      case Y of
        [_|Zs]   -> index3(N-1,Zs)
      end
  end.

% Examples to illustrate the scope of variables.

f(X)      -> Y=X+1,Y*X.

g([0|Xs]) -> g(Xs);
g([Y|Xs]) -> Y+g(Xs);
g([])     -> 0.

% Binding a variable in both arms of a case expression is
% possible ...

safe(X) ->
  case X of
    one -> Y = 12;
    _   -> Y = 196
  end,
  X+Y.

% ... but the preferred style is to assign a value to the variable.
% where the value is defined using a case

preferred(X) ->
  Y = case X of
        one -> 12;
        _   -> 196
      end,
  X+Y.
```

Source: Erlang Programming - Ch. 3

Guards

```erlang
%% Guards

% Defining factorial using a guard

factorial(N) when N > 0 ->
  N * factorial(N - 1);
factorial(0) -> 1.

% An example of a complex guard ...

guard(X,Y) when not(((X>Y) or not(is_atom(X)) ) and (is_atom(Y) or (X==3.4))) ->
  X+Y.

% ... and the same guard rewritten using ; and ,

guard2(X,Y) when not(X>Y) , is_atom(X) ; not(is_atom(Y)) , X=/=3.4 ->
  X+Y.

% Examples from the examples.erl module.

even(Int) when Int rem 2 == 0 -> true;
even(Int) when Int rem 2 == 1 -> false.

number(Num) when is_integer(Num) -> integer;
number(Num) when is_float(Num)   -> float;
number(_Other)                   -> false.
```

Source: Erlang Programming - Ch. 3

Built-in Functions

- Object Access and Examination
- Type Conversion
- Process Dictionary
- Meta Programming
- Process, Port, Distribution, and System Information
- Input and Output

Source: Erlang Programming - Ch. 3

Recursion

```erlang
%% Recursion

% The bump function: add one to each element of a list.
% A first example of recursion.

bump([]) -> [];
bump([Head | Tail]) -> [Head + 1 | bump(Tail)].

% Finding the average value in a numeric list.

average(List) -> sum(List) / len(List).

sum([]) -> 0;
sum([Head | Tail]) -> Head + sum(Tail).

len([]) -> 0;
len([_ | Tail]) -> 1 + len(Tail).

% Is the first argument a memeer of the second argument (a list)?

member(_, [])      -> false;
member(H, [H | _]) -> true;
member(H, [_ | T]) -> member(H, T).

% Summing a list using tail recursion.

sum_acc([],Sum) -> Sum;
sum_acc([Head|Tail], Sum) -> sum_acc(Tail, Head+Sum).

sum2(List) -> sum_acc(List,0).

% Bumping every element in a list using an accumulator.

bump2(List) -> bump_acc(List, []).

bump_acc([], Acc)            -> reverse(Acc);
bump_acc([Head | Tail], Acc) -> bump_acc(Tail, [Head + 1 | Acc]).
```

```erlang
% Reversing a list.

reverse(List) -> reverse_acc(List, []).

reverse_acc([], Acc) -> Acc;
reverse_acc([H | T], Acc) -> reverse_acc(T, [H | Acc]).

% Merging the elements of two lists.

merge(Xs,Ys) ->
 lists:reverse(mergeL(Xs,Ys,[])).

mergeL([X|Xs],Ys,Zs) ->
  mergeR(Xs,Ys,[X|Zs]);
mergeL([],[],Zs) ->
  Zs.

mergeR(Xs,[Y|Ys],Zs) ->
  mergeL(Xs,Ys,[Y|Zs]);
mergeR([],[],Zs) ->
  Zs.

% Average revisited, this time using two accumulators.

average2(List) -> average_acc(List, 0,0).

average_acc([], Sum, Length) ->
  Sum / Length;
average_acc([H | T], Sum, Length) ->
  average_acc(T, Sum + H, Length + 1).

% Iterative version of sum

sum3(Boundary) -> sum_acc(1, Boundary, 0).

sum_acc(Index, Boundary, Sum) when Index =< Boundary ->
  sum_acc(Index + 1, Boundary, Sum + Index);
sum_acc(_I, _B, Sum)->
   Sum.
```

Source: Erlang Programming - Ch. 3

Handling Errors

```
try Exprs of
     Pattern1 [when Guard1] ->
          ExpressionBody1;
     Pattern2 [when Guard2] ->
          ExpressionBody2
catch
     [Class1:]ExceptionPattern1 [when ExceptionGuardSeq1] ->
          ExceptionBody1;
     [ClassN:]ExceptionPatternN [when ExceptionGuardSeqN] ->
          ExceptionBodyN
end
```

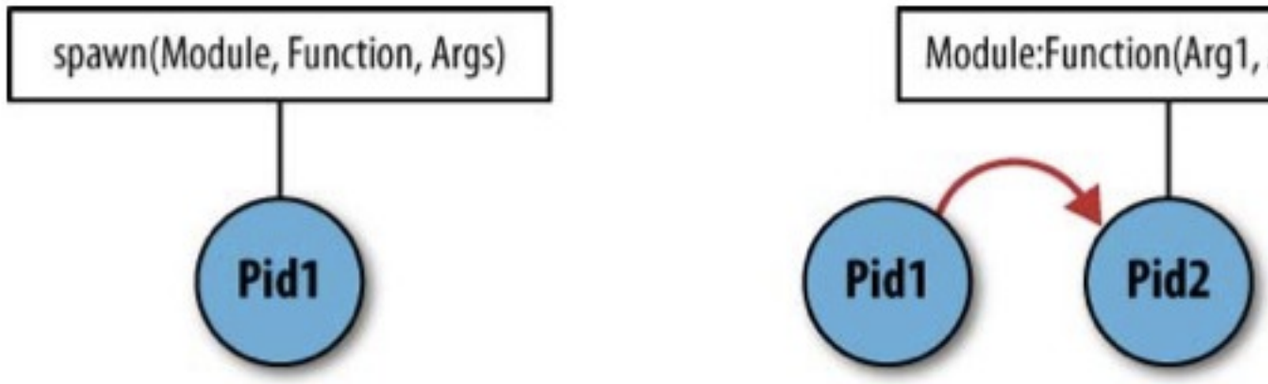
```
try_return(X) when is_integer(X) ->
  try return_error(X) of
    Val -> {normal, Val}
  catch
    exit:Reason -> {exit, Reason};
    throw:Throw -> {throw, Throw};
    error:Error -> {error, Error}
  end.

try_wildcard(X) when is_integer(X) ->
  try return_error(X)
    catch
    throw:Throw -> {throw, Throw};
    error:_     -> error;
    Type:Error  -> {Type, Error};
    _           -> other;                %% Will never be returned
    _:_         -> other                 %% Will never be returned
  end.

try_return2(X) when is_integer(X) ->
  try return_error(X) of
    Val -> {normal, Val}
    catch
    exit:_  -> 34;
    throw:_ -> 99;
    error:_ -> 678
  end.

return(X) when is_integer(X) ->
  catch return_error(X).
```

Source: Erlang Programming - Ch. 3

Concurrent Programming

- Creating Processes
- Message Passing
- Receiving Messages
- Registered Processes

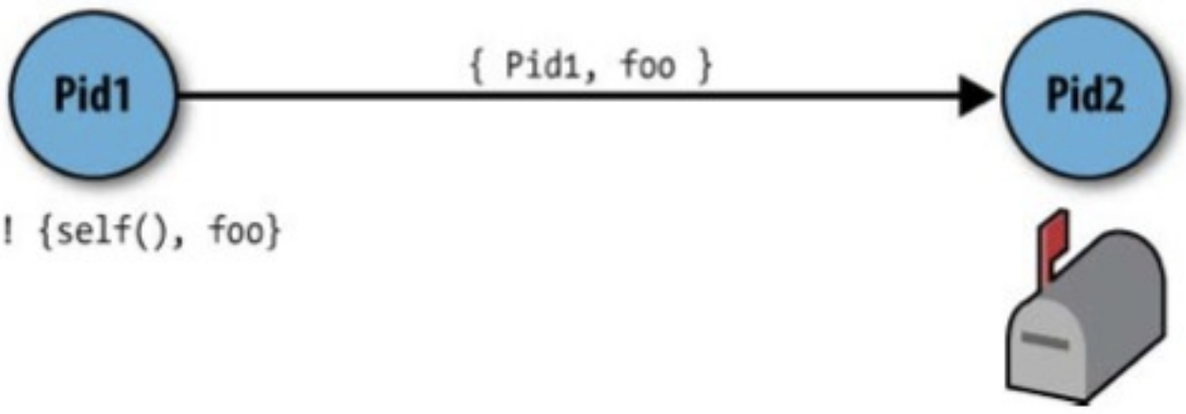
Creating Processes




```
Pid2 = spawn(Module, Function, Arguments).
```

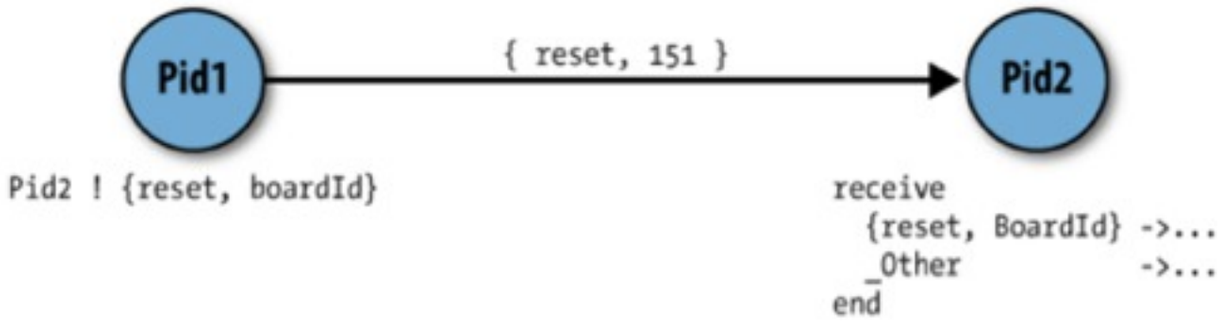
Source: Erlang Programming - Ch. 4

Message Passing



Source: Erlang Programming - Ch. 4

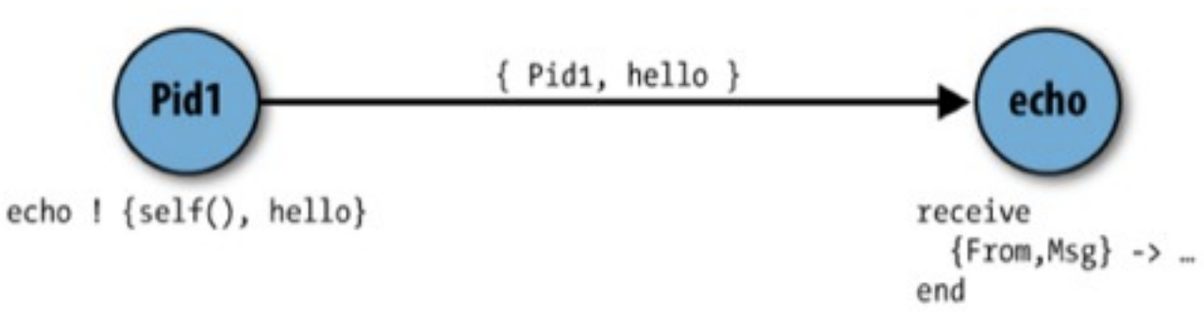
Receiving Messages




```erlang
%% Code from
%%   Erlang Programming
%%   Francecso Cesarini and Simon Thompson
%%   O'Reilly, 2008
%%   http://oreilly.com/catalog/9780596518189/
%%   http://www.erlangprogramming.org/
%%   (c) Francesco Cesarini and Simon Thompson

-module(echo).
-export([go/0, loop/0]).

go() ->
    Pid = spawn(echo, loop, []),
    Pid ! {self(), hello},
    receive
      {Pid, Msg} ->
        io:format("~w~n",[Msg])
    end,
    Pid ! stop.


loop() ->
    receive
      {From, Msg} ->
         From ! {self(), Msg},
         loop();
      stop ->
        true
    end.
```

Source: Erlang Programming - Ch. 4

Registered Processes



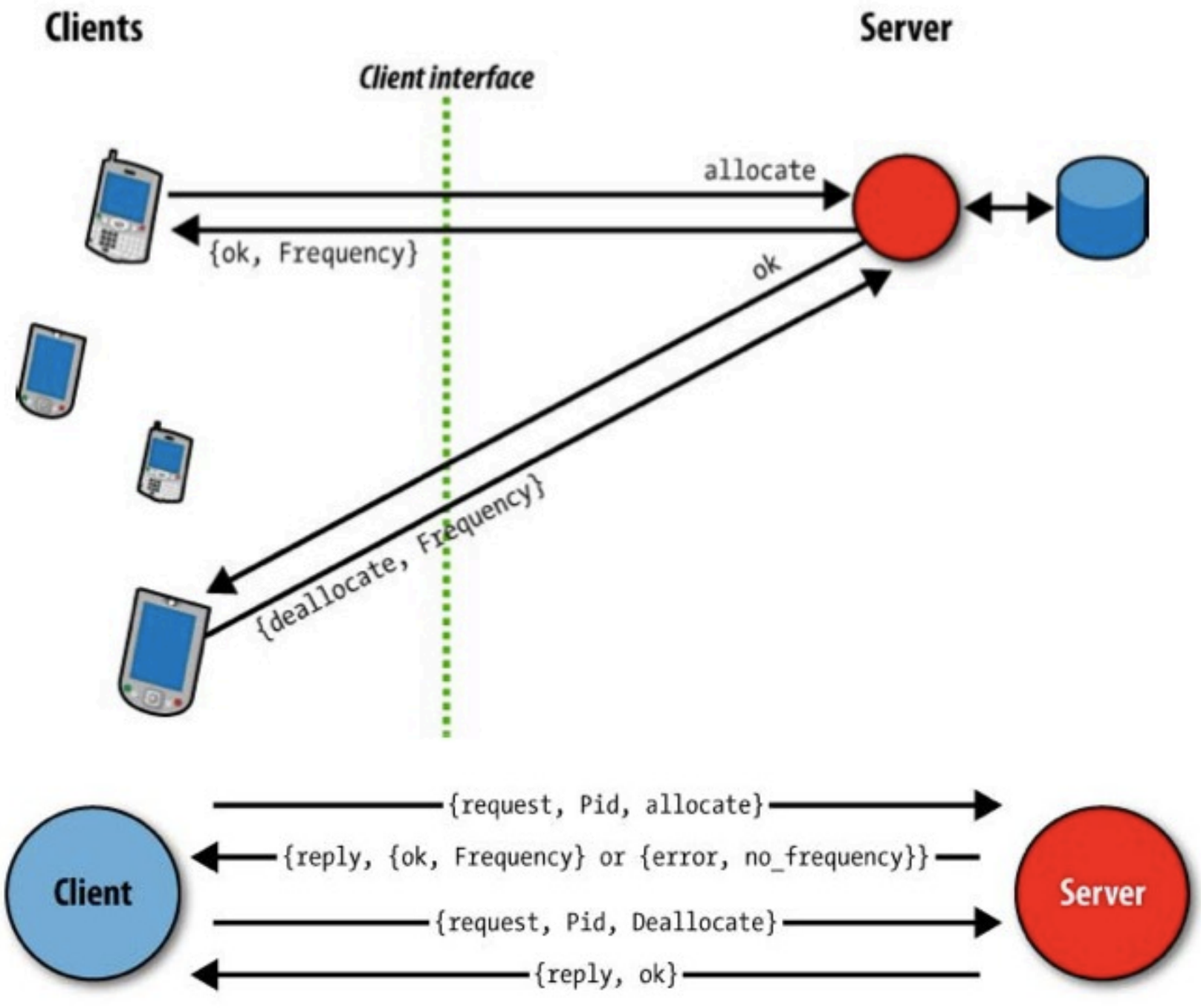

```erlang
%% Code from
%%   Erlang Programming
%%   Francecso Cesarini and Simon Thompson
%%   O'Reilly, 2008
%%   http://oreilly.com/catalog/9780596518189/
%%   http://www.erlangprogramming.org/
%%   (c) Francesco Cesarini and Simon Thompson

-module(echo2).
-export([go/0, loop/0]).

go() ->
  register(echo, spawn(echo2, loop, [])),
  echo ! {self(), hello},
  receive
    {_Pid, Msg} ->
      io:format("~w~n",[Msg])
  end.


loop() ->
  receive
    {From, Msg} ->
      From ! {self(), Msg},
      loop();
    stop ->
      true
  end.
```

Source: Erlang Programming - Ch. 4

Process Design Patterns

- Client/Server Models
- Finite State Machines
- Event Managers and Handlers

Client/Server Models



Source: Erlang Programming - Ch. 5

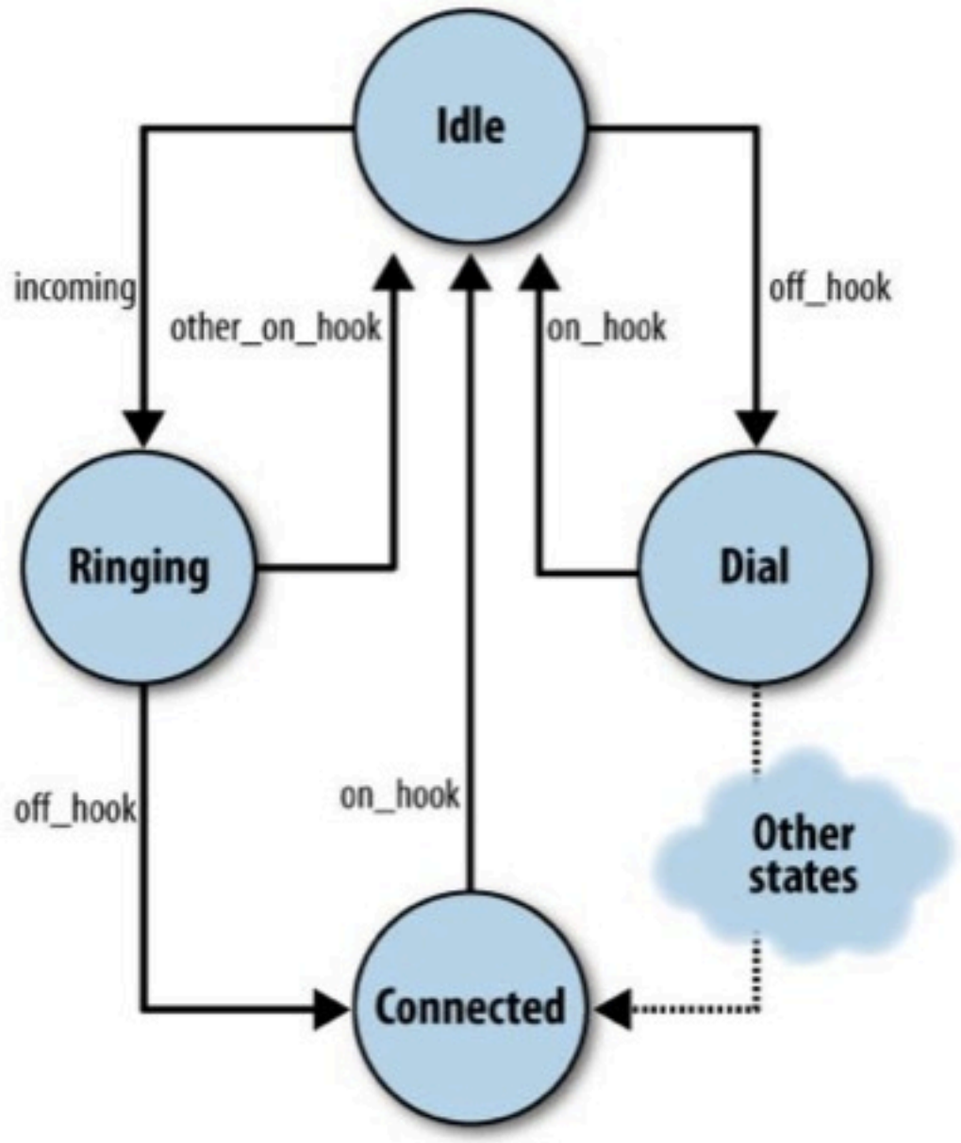
A Process Pattern

```erlang
%% Code from
%%   Erlang Programming
%%   Francecso Cesarini and Simon Thompson
%%   O'Reilly, 2008
%%   http://oreilly.com/catalog/9780596518189/
%%   http://www.erlangprogramming.org/
%%   (c) Francesco Cesarini and Simon Thompson

-module(server).
-export([start/2, stop/1, call/2]).
-export([init/1]).

start(Name, Data) ->
  Pid = spawn(generic_handler, init,[Data]),
  register(Name, Pid), ok.

stop(Name) ->
  Name ! {stop, self()},
  receive {reply, Reply} -> Reply end.

call(Name, Msg) ->
  Name ! {request, self(), Msg},
  receive {reply, Reply} -> Reply end.

reply(To, Msg) ->
  To ! {reply, Msg}.

init(Data) ->
  loop(initialize(Data)).
```

```erlang
loop(State) ->
  receive
    {request, From, Msg} ->
      {Reply,NewState} = handle_msg(Msg, State),
      reply(From, Reply),
      loop(NewState);
    {stop, From}  ->
      reply(From, terminate(State))
  end.

%% initialize(...)   -> ...
%% handle_msg(...,...) -> ...
%% terminate(...)    -> ...
```

Source: Erlang Programming - Ch. 5

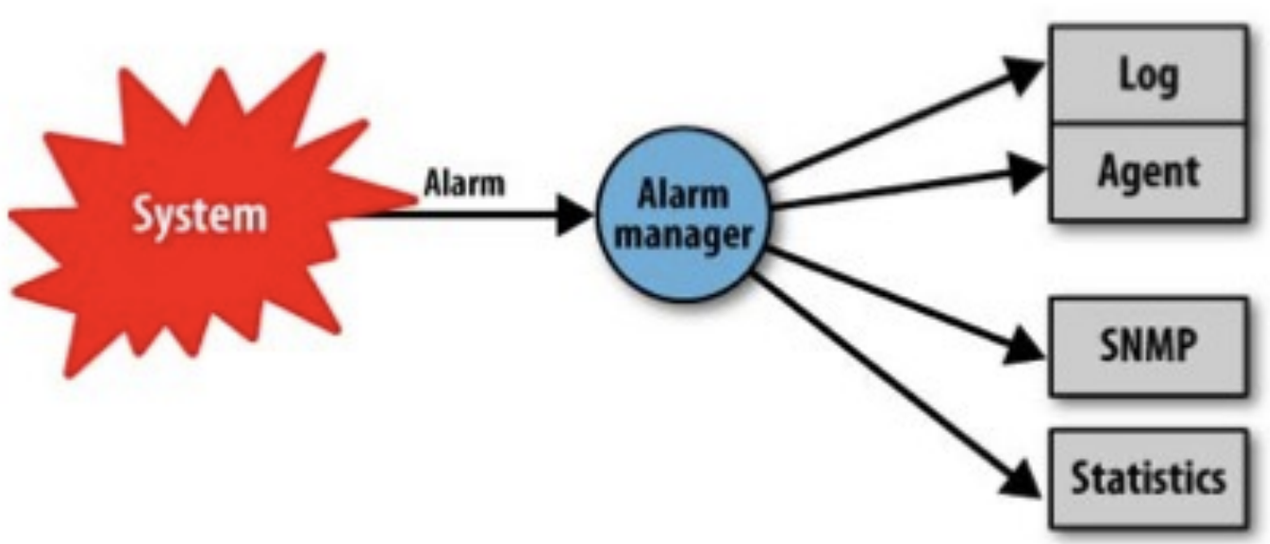
Finite State Machines




```erlang
%% Code from
%%   Erlang Programming
%%   Francecso Cesarini and Simon Thompson
%%   O'Reilly, 2008
%%   http://oreilly.com/catalog/9780596518189/
%%   http://www.erlangprogramming.org/
%%   (c) Francesco Cesarini and Simon Thompson

-module(fsm).
-export([idle/0,ringing/1]).

idle() ->
  receive
    {Number, incoming} ->
      start_ringing(),
      ringing(Number);
    off_hook ->
      start_tone(),
      dial()
  end.

ringing(Number) ->
  receive
    {Number, other_on_hook} ->
      stop_ringing(),
      idle();
    {Number, off_hook} ->
      stop_ringing(),
      connected(Number)
  end.
```

Source: Erlang Programming - Ch. 5

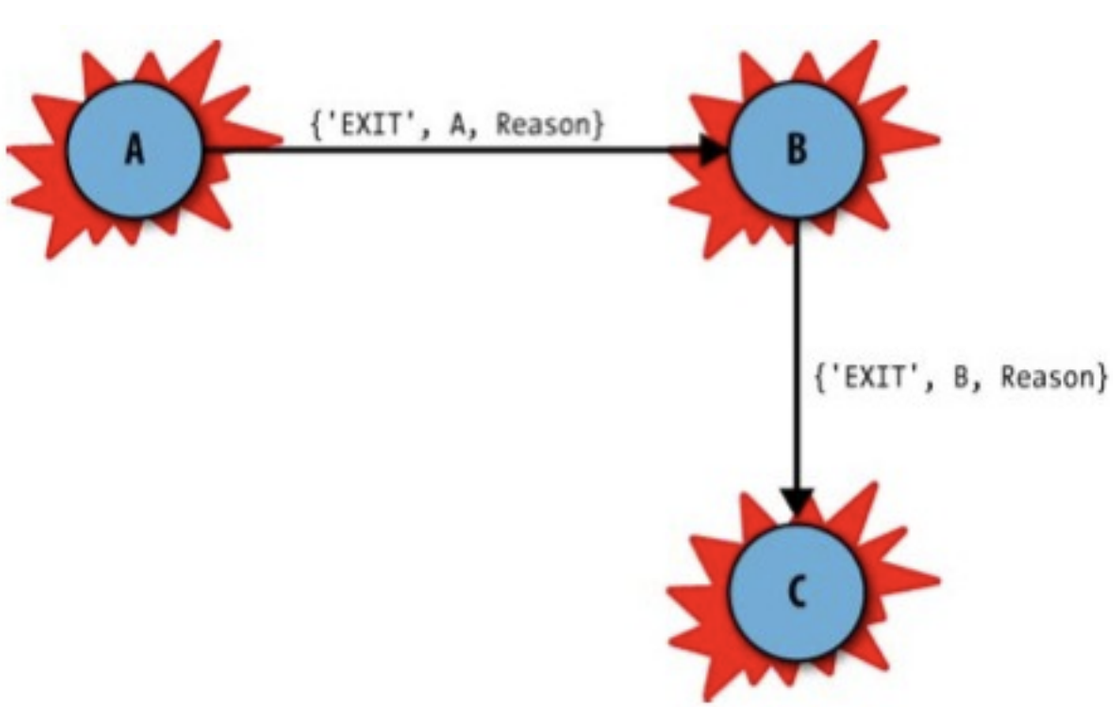
Event Managers and Handlers



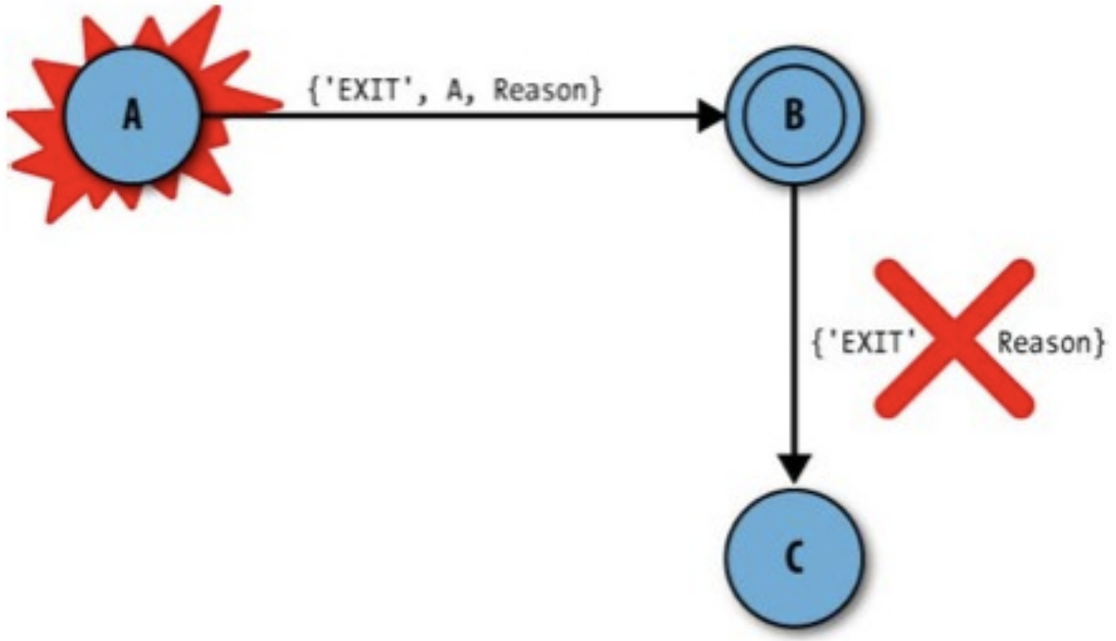
Source: Erlang Programming - Ch. 5

Process Error Handling

Process Links and Exit Signals



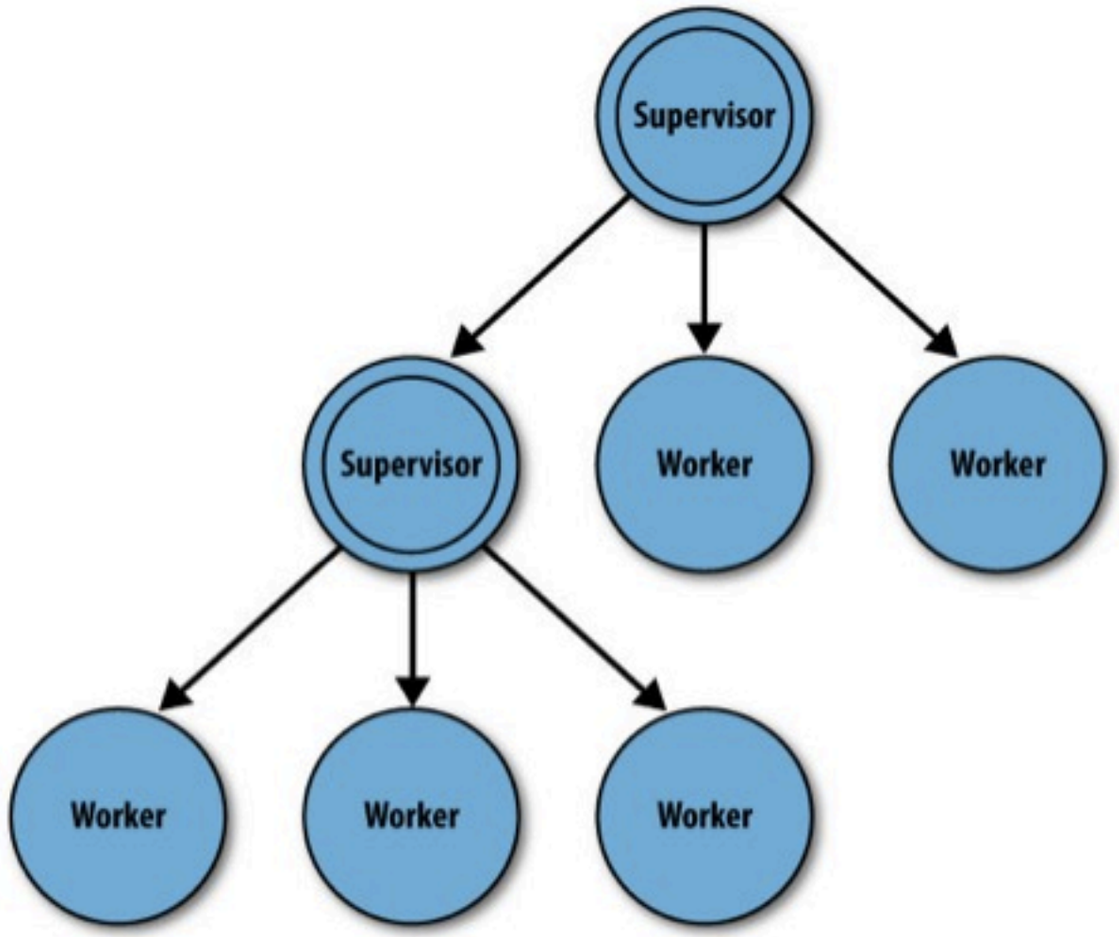
Propagation of exit signals



Trapping Exits

Source: Erlang Programming - Ch. 6

Robust Systems



Source: Erlang Programming - Ch. 6

Records & Macros

Records

```
-record(person, {name,age,phone}).

#person{name="Joe", age=21,
        phone="999-999"}

Person = #person{name="Fred"}
NewPerson = Person#person{age=37}
```

Simple Macros

```
-define(TIMEOUT, 1000).

receive
    after ?TIMEOUT -> ok
end.
```

Parameterized Macros

```
-define(DBG(Str, Args), io:format(Str, Args)).
```

Source: Erlang Programming - Ch. 7

Live Code Upgrade

The ability to load new and updated modules during runtime allows systems to run without interruption, not only when errors are being fixed but also when new func- tionality is added.

It also reduces the turnaround time of bugs and facilitates testing, as in most cases, systems do not have to be restarted for patches to be validated and deployed.

The software upgrade mechanism relies on a set of simple but powerful constructs on which more powerful tools are built.

These upgrade tools are used in pretty much every Erlang-based system where downtime has to be reduced to a minimum.

Source: Erlang Programming - Ch. 8

Funs and Higher-Order Functions

```
1> Bump = fun(Int) -> Int + 1 end.
#Fun<erl_eval.6.13229925>
2> Bump(10). 11
3> (fun(Int) -> Int + 1 end)(9).
10
```

Functions as Results

```
times(X) ->
    fun (Y) -> X*Y end.
```

This is a function that takes one argument: X,

```
times(X) -> ...
```

and whose result is this expression:

```
fun (Y) -> X*Y end.
```

Functions as Arguments

```
doubleAll([]) ->
    [];
doubleAll([X|Xs]) ->
    [X*2 | doubleAll(Xs)].
```

Predefined Higher-Order Functions

```
all(Predicate, List)
any(Predicate, List)
dropwhile(Predicate, List)
filter(Predicate, List)
foldl(Fun, Accumulator, List)
map(Fun, List)
partition(Predicate, List)
```

Source: Erlang Programming - Ch. 9

List Comprehensions

General List Comprehensions
[Expression || Generators, Guards, Generators, ...]

Generators
A generator has the form Pattern <- List, where Pattern is a pattern that is matched with elements from the List expression. You can read the symbol <- as "comes from"; it's also like the mathematical symbol ∈, meaning "is an element of."

Guards
Guards are just like guards in function definitions, giving a true or false result. The variables in the guards are those which appear in generators to the left of the guard (and any other variables defined at the outer level).

Expression
The expression specifies what the elements of the result will look like.

```
1> [X || X <- [1,2,3], X rem 2 == 0].
[2]

2> Database = [ {francesco, harryPotter}, {simon, jamesBond},
{marcus, jamesBond}, {francesco, daVinciCode} ].
...
3> [Person || {Person,_} <- Database].
[francesco,simon,marcus,francesco]

4> [ {X,Y} || X <- lists:seq(1,3), Y <- lists:seq(X,3) ].
[{1,1},{1,2},{1,3},{2,2},{2,3},{3,3}]
```

Source: Erlang Programming - Ch. 9

ETS and DETS Tables

To handle fast searches, Erlang provides two mechanisms. This chapter introduces Erlang Term Storage (ETS) and Disk Erlang Term Storage (Dets), two mechanisms for memory- and disk-efficient storage and retrieval of large collections of data.

Set
In a set, each key can occur only once. So, using this kind of table for the index example will mean there can be only one element in the table for each word.

Ordered set
An ordered set has the same property as the set, but it is stored so that the elements can be traversed following the lexicographical order on the keys.

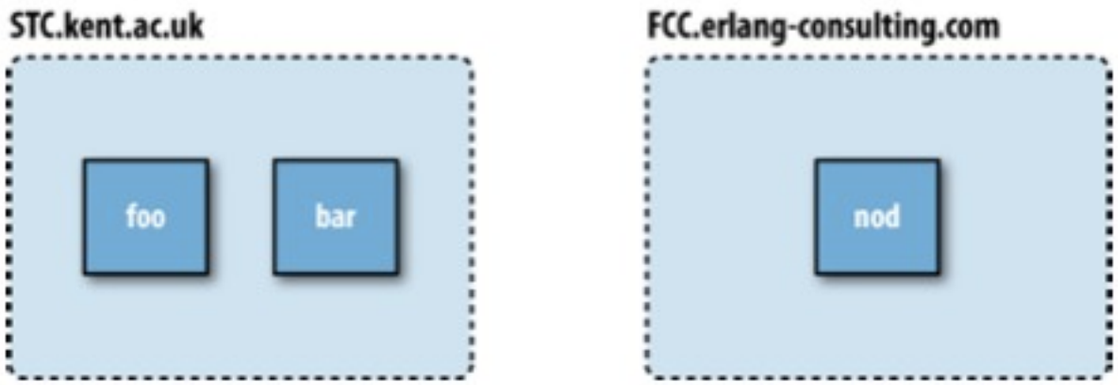
Bag
A bag allows multiple entries for the same key. The elements have to be distinct: in the index example, this means there can be only one entry for a particular word on a particular line.

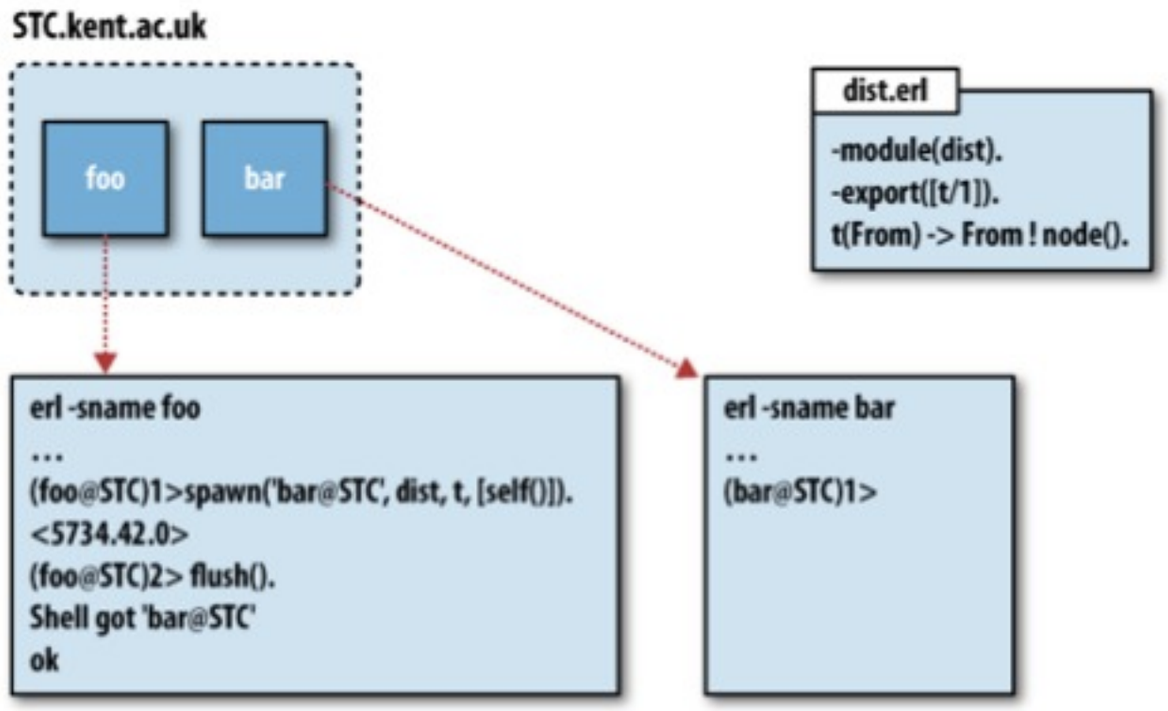
Duplicate bag
A duplicate bag allows duplicated elements, as well as duplicated keys.

Source: Erlang Programming - Ch. 10

Distributed Programming

A distributed Erlang system consists of a number of Erlang runtime systems communicating with each other. Each such runtime system is called a node.
Message passing between processes at different nodes, as well as links and monitors, are transparent when pids are used.
Registered names, however, are local to each node. This means the node must be specified as well when sending messages etc. using registered names.





Source: http://www.erlang.org/doc/reference_manual/distributed.html
Erlang Programming - Ch. 11

OTP Behaviours

The standard Erlang/OTP behaviours are:

gen_server
For implementing the server of a client-server relation.

gen_fsm
For implementing finite state machines.

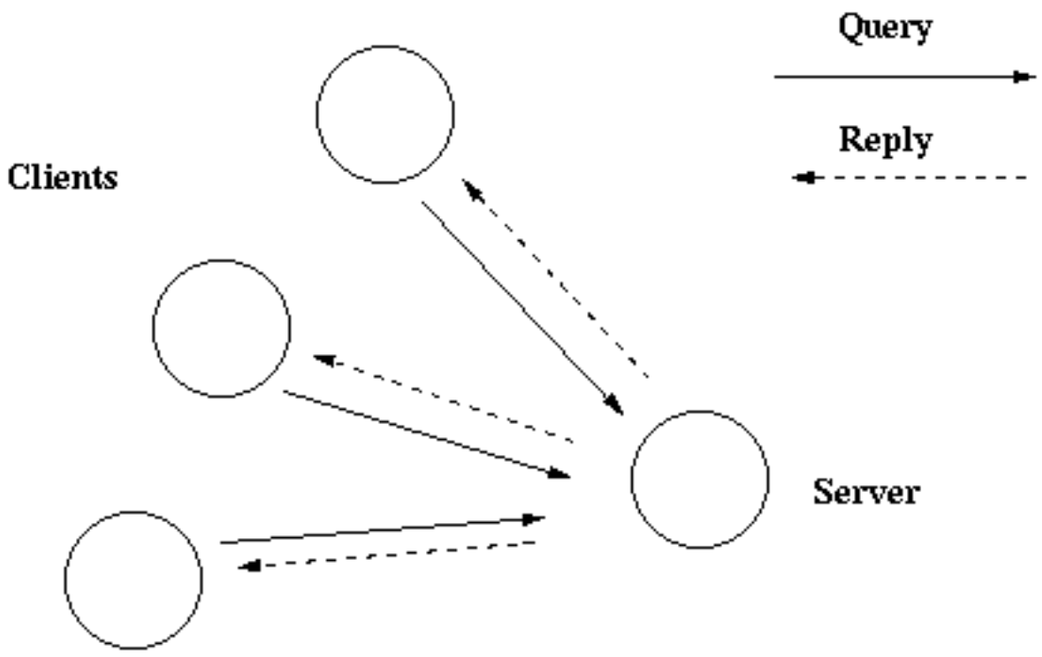
gen_event
For implementing event handling functionality.

supervisor
For implementing a supervisor in a supervision tree.

Source: http://www.erlang.org/doc/design_principles/des_princ.html

Generic Servers

```erlang
-module(ch3).
-behaviour(gen_server).

-export([start_link/0]).
-export([alloc/0, free/1]).
-export([init/1, handle_call/3, handle_cast/2]).

start_link() ->
    gen_server:start_link({local, ch3}, ch3, [], []).

alloc() ->
    gen_server:call(ch3, alloc).

free(Ch) ->
    gen_server:cast(ch3, {free, Ch}).

init(_Args) ->
    {ok, channels()}.

handle_call(alloc, _From, Chs) ->
    {Ch, Chs2} = alloc(Chs),
    {reply, Ch, Chs2}.

handle_cast({free, Ch}, Chs) ->
    Chs2 = free(Ch, Chs),
    {noreply, Chs2}.
```

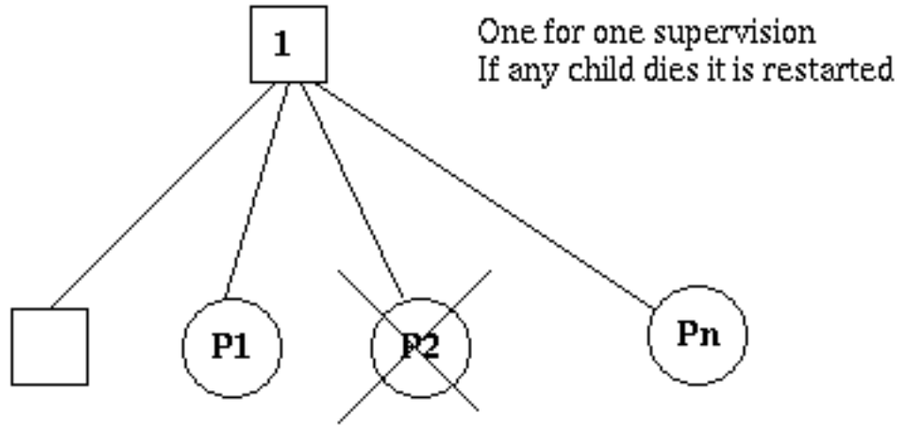


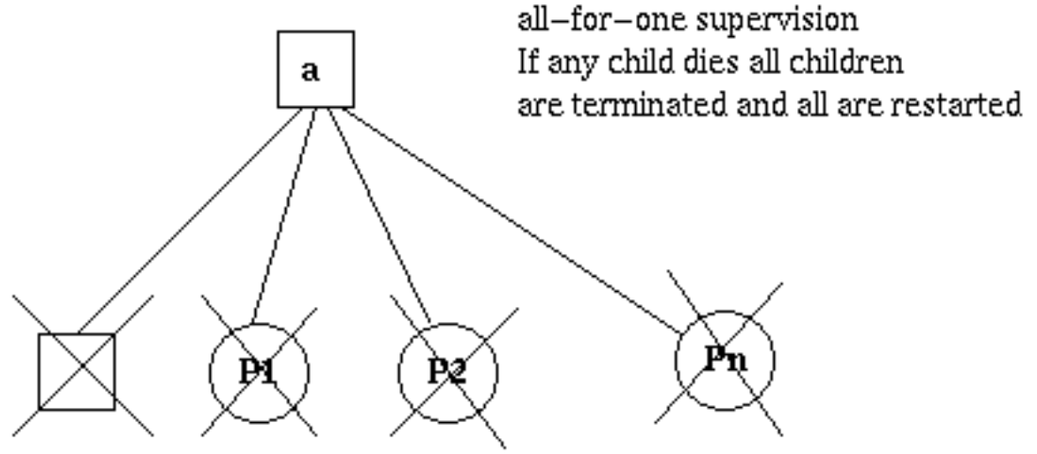
The Client-server model

Source: http://www.erlang.org/doc/design_principles/gen_server_concepts.html

Supervisors

```
-module(ch_sup).
-behaviour(supervisor).

-export([start_link/0]).
-export([init/1]).

start_link() ->
    supervisor:start_link(ch_sup, []).

init(_Args) ->
    {ok, {{one_for_one, 1, 60},
          [{ch3, {ch3, start_link, []},
            permanent, brutal_kill, worker, [ch3]}]}}.
```





`one_for_one` is the **restart strategy**.

1 and 60 defines the **maximum restart frequency**.

The tuple `{ch3, ...}` is a **child specification**.

Source: http://www.erlang.org/doc/design_principles/sup_princ.html

(a)mnesia

Mnesia is a distributed DataBase Management System (DBMS), appropriate for telecommunications applications and other Erlang applications which require continuous operation and exhibit soft real-time properties.

Listed below are some of the most important and attractive capabilities, Mnesia provides:

- A relational/object hybrid data model which is suitable for telecommunications applications.
- A specifically designed DBMS query language, QLC (as an add-on library).
- Persistence. Tables may be coherently kept on disc as well as in main memory.
- Replication. Tables may be replicated at several nodes.
- Atomic transactions. A series of table manipulation operations can be grouped into a single atomic transaction.
- Location transparency. Programs can be written without knowledge of the actual location of data.
- Extremely fast real time data searches.
- Schema manipulation routines. It is possible to reconfigure the DBMS at runtime without stopping the system.

Source: http://www.erlang.org/doc/man/mnesia.html

Socket Programming

```erlang
%%% File    : tcp.erl
%%% Description : Example from Chapter 15,

-module(tcp).
-export([client/2, send/2, server/0, wait_connect/2,
      get_request/3, handle/2]).


client(Host, Data) ->
    {ok, Socket} = gen_tcp:connect(Host, 1234, [binary, {packet, 0}]),
    send(Socket, Data),
    ok = gen_tcp:close(Socket).


send(Socket, <<Chunk:100/binary, Rest/binary>>) ->
    gen_tcp:send(Socket, Chunk),
    send(Socket, Rest);
send(Socket, Rest) ->
    gen_tcp:send(Socket, Rest).


server() ->
    {ok, ListenSocket} = gen_tcp:listen(1234, [binary, {active, false}]),
    wait_connect(ListenSocket,0).


wait_connect(ListenSocket, Count) ->
    {ok, Socket} = gen_tcp:accept(ListenSocket),
    spawn(?MODULE, wait_connect, [ListenSocket, Count+1]),
    get_request(Socket, [], Count).

...
```

Source: Erlang Programming - Ch. 15

Interfaces with other Programming Languages

- JInterface
- C Node
- Ports
- NIFs

Java - JInterface

```
OtpNode self = new OtpNode("gurka");
    OtpMbox mbox = self.createMbox("echo");
    OtpErlangObject o;
    OtpErlangTuple msg;
    OtpErlangPid from;

    while (true) {
      try {
        o = mbox.receive();
        if (o instanceof OtpErlangTuple) {
          msg = (OtpErlangTuple)o;
          from = (OtpErlangPid)(msg.elementAt(0));
          mbox.send(from,msg.elementAt(1));
      }
      catch (Exception e) {
        System.out.println("" + e);
      }
    }
```

Source: http://www.erlang.org/doc/apps/jinterface/jinterface_users_guide.html

C Nodes

From Erlang's point of view, the C node is treated like a normal Erlang node.

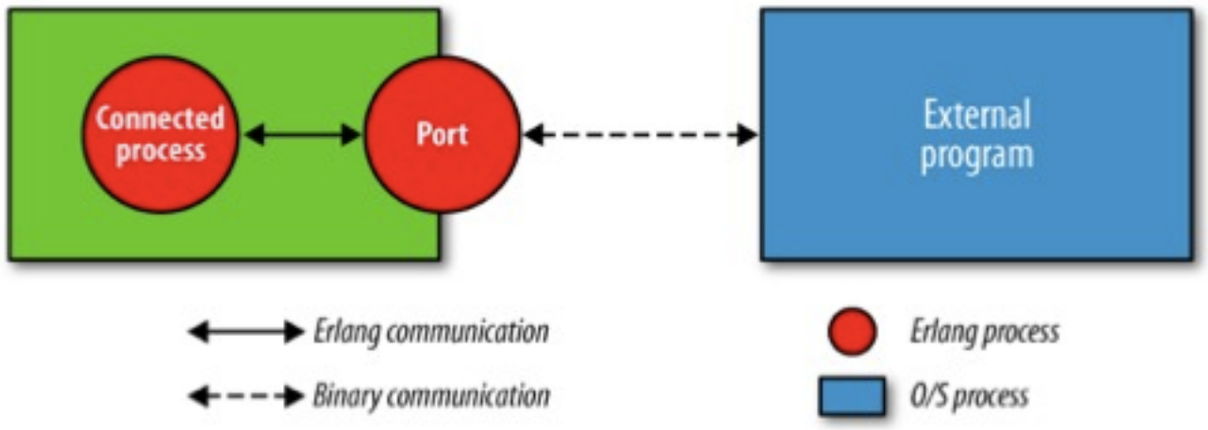
Therefore, calling the functions foo and bar only involves sending a message to the C node asking for the function to be called, and receiving the result.

Sending a message requires a recipient; a process which can be defined using either a pid or a tuple consisting of a registered name and a node name.

Source: http://www.erlang.org/doc/tutorial/cnode.html

Ports

An Erlang port allows communication between an Erlang node and an external program through binary messages sent to and from an Erlang process running in the node— known as the connected process of the port—and the external program, running in a separate operating system thread.



Source: Erlang Programming - Ch. 16

NIFs

A NIF (Native Implemented Function) is a function that is implemented in C instead of Erlang. NIFs appear as any other functions to the callers.

```c
#include "erl_nif.h"

extern int foo(int x);
extern int bar(int y);

static ERL_NIF_TERM foo_nif(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    int x, ret;
    if (!enif_get_int(env, argv[0], &x)) {
      return enif_make_badarg(env);
    }
    ret = foo(x);
    return enif_make_int(env, ret);
}

static ERL_NIF_TERM bar_nif(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
{
    int y, ret;
    if (!enif_get_int(env, argv[0], &y)) {
      return enif_make_badarg(env);
    }
    ret = bar(y);
    return enif_make_int(env, ret);
}

static ErlNifFunc nif_funcs[] = {
    {"foo", 1, foo_nif},
    {"bar", 1, bar_nif}
};

ERL_NIF_INIT(complex6, nif_funcs, NULL, NULL, NULL, NULL)
```

```erlang
-module(complex6).
-export([foo/1, bar/1]).
-on_load(init/0).

init() ->
    ok = erlang:load_nif("./complex6_nif", 0).

foo(_X) ->
    exit(nif_library_not_loaded).
bar(_Y) ->
    exit(nif_library_not_loaded).
```

Source: http://www.erlang.org/doc/tutorial/nif.html

the dbg Tracer

The basic steps of tracing for function calls are on a non-live node:

```
> dbg:start().                                % start dbg
> dbg:tracer().                               % start a simple tracer process
> dbg:tp(Module, Function, Arity, []).        % specify MFA you are interested in
> dbg:p(all, c).                              % trace calls (c) of that MFA for all processes.

... trace here

> dbg:stop_clear().                           % stop tracer and clear effect of tp and p calls.
```

You can trace for multiple functions at the same time.

```
> dbg:tpl(Module, '_', []).                          % all calls in Module
> dbg:tpl(Module, Function, '_', []).                % all calls to Module:Function with any arity.
> dbg:tpl(Module, Function, Arity, []).              % all calls to Module:Function/Arity.
> dbg:tpl(M, F, A, [{'_', [], [{return_trace}]}]). % same as before, but also show return value.
```

You can select the processes to trace on with the call to p().

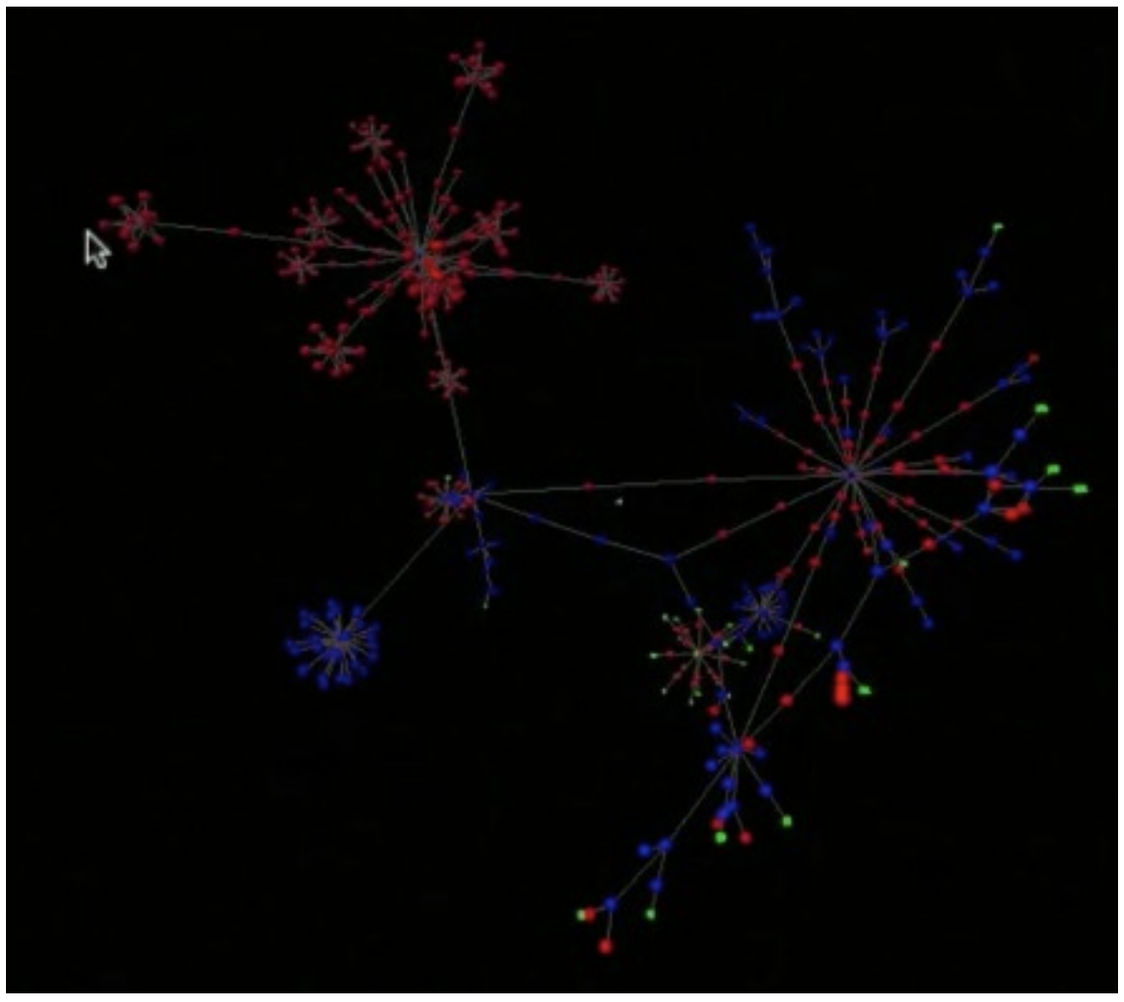
```
> dbg:p(all, c).       % trace calls to selected functions by all functions
> dbg:p(new, c).       % trace calls by processes spawned from now on
> dbg:p(Pid, c).       % trace calls by given process
> dbg:p(Pid, [c, m]).  % trace calls and messages of a given process
```

Source: http://stackoverflow.com/questions/1954894/using-trace-and-dbg-in-erlang

Example Trace App

A Visual Tour of Erlang
http://www.youtube.com/watch?v=lHoWfeNuAN8

Erlang Process Visualizer.
http://github.com/krestenkrab/erlubi



Types and Documentation

```
%% @doc Create the ETS and DETS tables which implement the database. The
%% argument gives the filename which is used to hold the DETS table.
%% If the table can be created, an `ok' tuple containing a
%% reference to the created table is returned; if not, it returns an `error'
%% tuple with a string describing the error.

%% @spec create_tables(string()) -> {ok, reference()} | {error, string()}

-spec(create_tables(string()) -> {ok, ref()} | {error, string()}).

create_tables(FileName) ->
    ets:new(subRam, [named_table, {keypos, #usr.msisdn}]),
    ets:new(subIndex, [named_table]),
    dets:open_file(subDisk, [{file, FileName}, {keypos, #usr.msisdn}]).

%% @doc Close the ETS and DETS tables implementing the database.
%% Returns either `ok' or and `error'
%% tuple with the reason for the failure to close the DETS table.

%% @spec close_tables() -> ok | {error, string()}

-spec(close_tables() -> ok | {error, string()}).

close_tables() ->
    ets:delete(subRam),
    ets:delete(subIndex),
    dets:close(subDisk).
```

Source: Erlang Programming - chapter18/usr_db.erl

Dialyzer

Dialyzer: A DIscrepancy AnaLYZer for ERlang Programs

Step 1

```
$ dialyzer --build_plt -r <erl-lib>/kernel-2.12.5/ebin <erl-lib>/ stdlib-1.15.5/ebin <erl-lib>/mnesia-4.4.7/ebin

Creating PLT /Users/simonthompson/.dialyzer_plt ...
re.erl:41: Call to missing or unexported function unicode:characters_to_binary/2
re.erl:134: Call to missing or unexported function unicode:characters_to_list/2
re.erl:200: Call to missing or unexported function re:compile/2
re.erl:226: Call to missing or unexported function unicode:characters_to_binary/2
re.erl:245: Call to missing or unexported function unicode:characters_to_list/2
re.erl:505: Call to missing or unexported function unicode:characters_to_list/2
re.erl:545: Call to missing or unexported function unicode:characters_to_binary/2
Unknown functions:
compile:file/2
compile:forms/2
compile:noenv_forms/2
compile:output_generated/1 c
rypto:des3_cbc_decrypt/5
crypto:start/0

done in 16m43.44s
done (warnings were emitted)
```

Step 2

```
$ dialyzer -c usr.erl usr_db.erl

Checking whether the PLT /Users/simonthompson/.dialyzer_plt is up-to-date... yes

Proceeding with analysis...
usr.erl:110: The pattern [] can never match the type {'error','instance'}
usr_db.erl:69: Call to missing or unexported function ets:safefixtable/2
done in 0m0.33s
done (warnings were emitted)
```

Source: Erlang Programming - Ch.18

EUnit

```
-module(fib).
-export([fib/1]).
-include_lib("eunit/include/eunit.hrl").

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

```
fib_test_() ->
   [?_assert(fib(0) =:= 1),
    ?_assert(fib(1) =:= 1),
    ?_assert(fib(2) =:= 2),
    ?_assert(fib(3) =:= 3),
    ?_assert(fib(4) =:= 5),
    ?_assert(fib(5) =:= 8),
    ?_assertException(error, function_clause, fib(-1)),
    ?_assert(fib(31) =:= 2178309)
   ].
```

Source: http://www.erlang.org/doc/apps/eunit/chapter.html

QuickCheck

```
%%--------------------------------------------------------------------
%% Encode/Decode bool
%%--------------------------------------------------------------------
prop_bool() ->
    ?FORALL({Id, Bool},
        {non_neg_integer(), oneof([boolean(), 0, 1])},
        begin
          Fun = fun (B) when B =:= 1; B =:= true -> true;
            (B) when B =:= 0; B =:= false -> false
            end,
          {{Id, Fun(Bool)}, <<>>} =:=
        (?DECODE((?ENCODE(Id, Bool, bool)), bool))
        end).

enclode_bool_test_() ->
    [?_assertMatch(<<8, 1>>, (?ENCODE(1, true, bool))),
     ?_assertMatch(<<8, 0>>, (?ENCODE(1, false, bool))),
     ?_assertMatch(<<40, 1>>, (?ENCODE(5, 1, bool))),
     ?_assertMatch(<<40, 0>>, (?ENCODE(5, 0, bool)))].

decode_bool_test_() ->
    [?_assertMatch({{1, true}, <<>>},
          (?DECODE(<<8, 1>>, bool))),
     ?_assertMatch({{1, false}, <<>>},
          (?DECODE(<<8, 0>>, bool)))].
```

Source: https://github.com/basho/erlang_protobuffs/blob/master/test/protobuffs_tests.erl

Applications

Most Watched This Month

- ericmoritz / **wsdemo**
- erlang / **otp**
- krestenkrab / **erlubi**
- extend / **cowboy**
- apache / **couchdb**

Most Forked This Month

- ericmoritz / **wsdemo**
- erlang / **otp**
- basho / **rebar**
- extend / **cowboy**
- apache / **couchdb**

Most Watched Overall

- erlang / **otp**
- apache / **couchdb**
- mochi / **mochiweb**
- basho / **rebar**
- tarcieri / **reia**

Most Forked Overall

- erlang / **otp**
- apache / **couchdb**
- basho / **rebar**
- mochi / **mochiweb**
- processone / **ejabberd**

Source: https://github.com/languages/Erlang

Conferences

Erlang User Conference 2012
Erlang Factory SF Bay Area 2012
Krakow Erlang Factory Lite 2012
Brussels Erlang Factory Lite
Brisbane Factory Lite
Erlang User Conference 2011 !!! > 320
Amsterdam Factory Lite
Paris Erlang Factory Lite
Edinburgh Factory Lite
Erlang Factory London 2011
Erlang Factory Lite Munich
Erlang Factory SF Bay Area 2011 !!! > 170
Erlang Factory Lite Krakow 2010
Erlang User Conference 2010
Tutorial Workshop 2010, Stockholm
Erlang Factory Lite LA
Erlang Factory London 2010
Erlang Factory SF Bay Area 2010

Erlang Factory Lite Krakow 2009
Property-based Testing Tutorial Workshop 2009, Stockholm
Erlang User Conference 2009, Stockholm
SIGPLAN Erlang Workshop 2009, Edinburgh
Erlang Factory London 2009
Erlang Factory SF Bay Area 2009

Source: http://www.erlang-factory.com/past_conferences

Advantages

- Nice architectures can be desinged
- Different way of thinking applications
- Debugging live is really nice
- Extensive Documentation

Issues & Risks

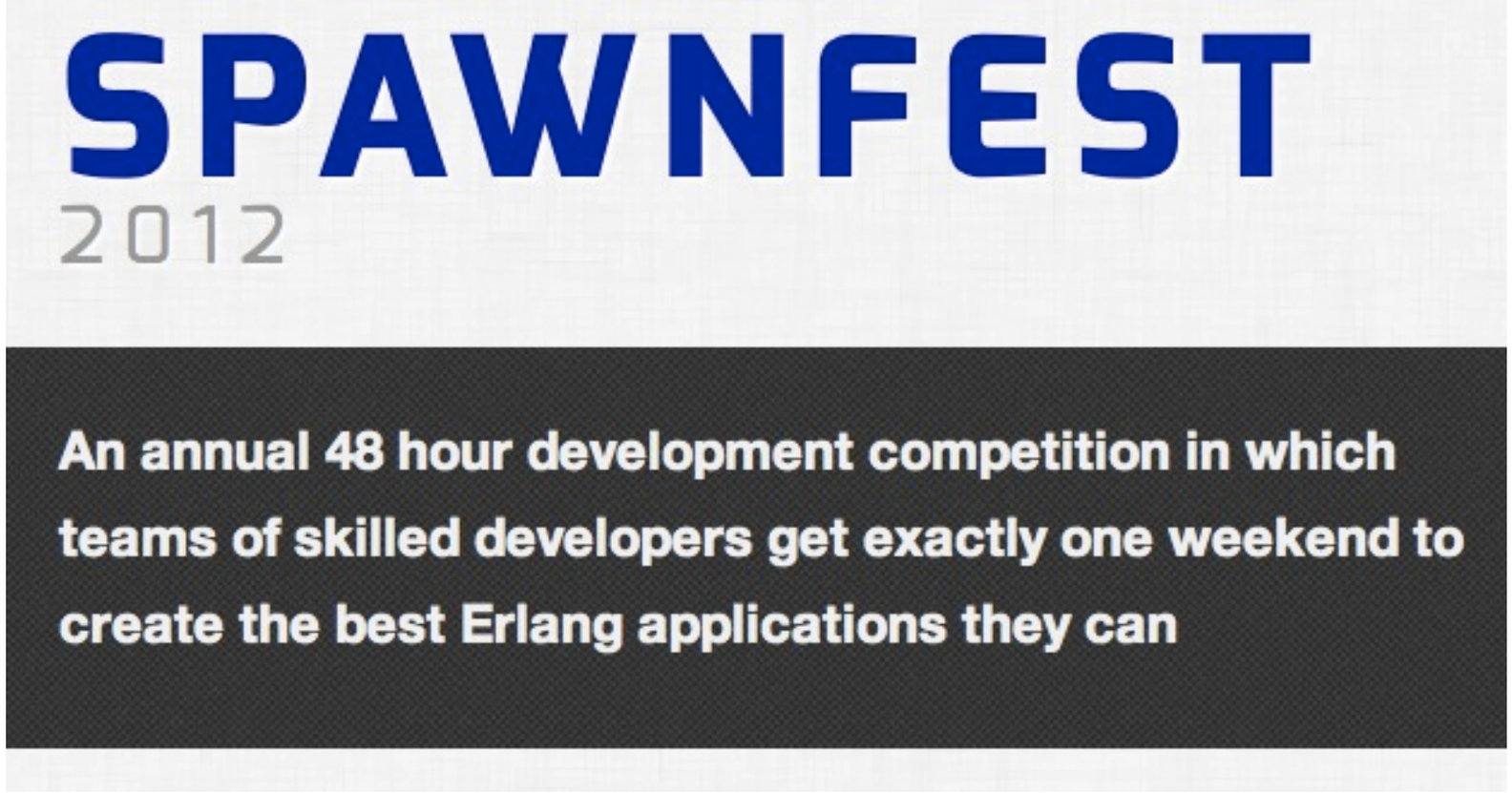
- Sometimes I miss Types
- Concurrency debugging for large applications is hard, even in Erlang
- Compilation time on large projects is big
- Memory/Process Leaks can crash your system if not handled correctly

Get Started

- Books
- GitHub
- Learn You some Erlang
- Mailing List(s)
- Erlang Factory Conferences

SpawnFest 2012

http://spawnfest.com/
July 7-8, 2012



Erlang is quite hot today

Thank You!

Image from: http://www.wallpaperstop.com/wallpapers/animal-wallpapers/penguin-wallpapers/3-funny-penguins-background-1600x1200-1011063.jpg