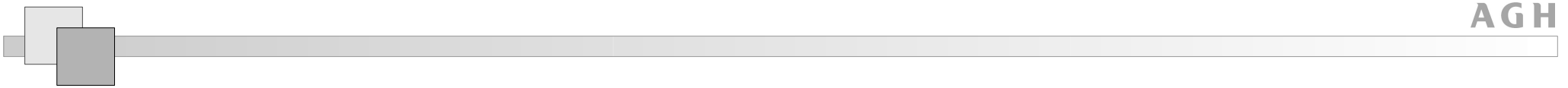
Erlang
Wprowadzenie do języka

Część 2

Programowanie w języku Erlang

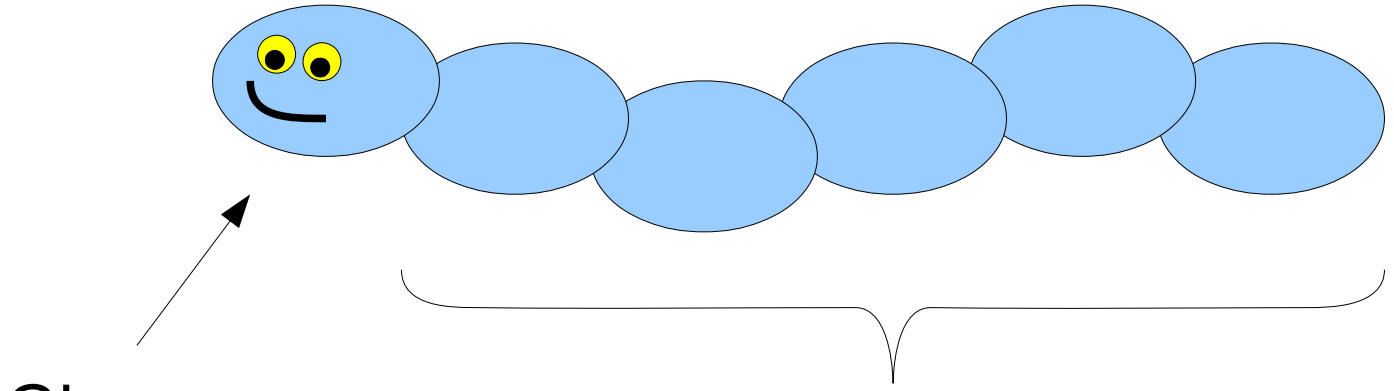
Plan

- List Comprehensions
- Funkcje wyższego rzędu
- Rekordy
- Słowniki, mapy
- Obsługa błędów
- Interfejsy innych technologii
- Testowanie



List Comprehensions

Listy – przypomnienie




```
[a, b, c]
[H | T]
[a | [b, c] ]
```

Matematyczne definiowanie zbiorów

- Zbiór takich elementów x,
- Które pochodzą ze zbioru X,
- Spełniają warunek c1,
- Spełniają też warunek c2,
- ...

```
{x ∈ X, x < 7, x >2}

ArrayList x = new ArrayList ();
for ( item in X )
   if (c1((item) && c2(item))
      x.add(item);
```

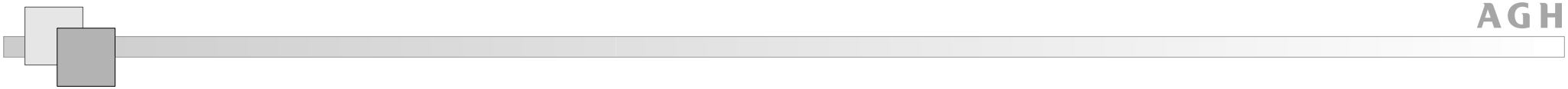
List Comprehensions

- Konstrukcja językowa pozwalająca na definiowanie list
  - Także na wygodne przekształcenie każdego elementu listy
- [Wyrażenie || Kwalifikator1, Kwalifikator2, ...]
- Kwalifikator to **generator** wartości lub **filtr** wartości (predykat)
- Wynikiem jest lista zawierająca wszystkie wygenerowane elementy, które przeszły wszystkie testy filtrów

```
>   [X*2 || X <- [1,2,4]].

>   [X || X <- [1,2,a,3,4,b,5,6], is_integer(X), X > 3].
```

List Comprehensions - przykłady

```
> [is_integer(X) || X<-[1,2,3,a,b]].
[true,true,true,false,false]

> [X || X<-[1,2,3,a,b], is_integer(X)].
[1,2,3]

> [{X,X*X} || X <- lists:seq(1,7)].
[{1,1}, {2,4}, {3,9}, {4,16}, {5,25}, {6,36}, {7,49}]

> [ X || X <- lists:seq($a, $z)].
"abcdefghijklmnopqrstuvwxyz"

> [X || X <- lists:seq(1,100), X rem 9 == 0].
[9,18,27,36,45,54,63,72,81,90,99]

> [{X,Y} || X <- [1,2,3], Y <- [5,10]].
[{1,5},{1,10},{2,5},{2,10},{3,5},{3,10}]
```



Funkcje wyższego rzędu

Wszystko jest funkcją !

- W rachunku Lambda...
  - Funkcje
  - Liczby
  - Warunki
  - Krotki
  - ...
- Więc każda funkcja powinna być traktowana jako obiekt pierwszoklasowy

Funkcje wyższego rzędu

Funkcje wyższego rzędu:

- Przyjmują inne funkcje jako argumenty
- Zwracają funkcje

$$f(y, g(x)) = h(x, y)$$

Funkcja jako wartość zmiennej

- Funkcję można przypisać do zmiennej

```
> Len = fun erlang:length/1.
#Fun<erlang.length.1>
> Len("Ala ma kota").
11
```

Funkcje anonimowe

- Funkcje bez nazwy

```
>   Fun1 = fun (X) -> X*2+2 end.

>   Fun2 = fun
       (0) -> 0;
       (N) when is_integer(N) -> 7 / N
    end.

>   Fun3 = fun (Fun, Val) -> Fun(Val) end.
```

Funkcje anonimowe rekurencyjne

- Fun factorial/1

```
>  Fafik = fun (0) -> 1; (X) -> X * ?(X-1) end.
```

```
>  Fafik = fun    (0, _) -> 1;
                  (X, Fik) -> X * Fik(X-1,Fik)
   end.
>  Fafik(10,Fafik).
```

```
>  Fafik = fun F(0) -> 1;
               F(X) -> X * F(X-1)
   end.
```

Domknięcia

- Funkcje są tworzone w kontekście
- Są przechowywane jako domknięcia
- Mają dostęp do zmiennych kontekstu

```
> Key = "radiculus".
"radiculus"

> GetKey = fun (Get) -> Get ++ Key end.
#Fun<erl_eval.6.80484245>

> GetKey("The key is: ").
"The key is: radiculus"
```

Map

- Podstawowa operacje z wykorzystaniem funów: Wykonanie funkcji dla każdego elementu listy

```
>  Dup = fun (X) -> X*2 end.
#Fun<erl_eval.6.80484245>

> lists:map(Dup, [1,2,3,4]).
[2,4,6,8]

> lists:map(Dup, "Ala ma kota").
[130,216,194,64,218,194,64,214,222,232,194]

> [Dup(X) || X<-[1,2,3,4]].
```

Filter

- Wybranie elementów listy spełniających zadany warunek

```
> IsGood = fun (good) -> true; (_) -> false end.
#Fun<erl_eval.6.80484245>

> lists:filter(IsGood, [good, bad, better, good]).
[good,good]

> GT = fun({X,Y})when X>Y -> true; (_) -> false end.
> ListOfPairs = [{X,Y}|| X<-[1,2,3,4], Y<-[1,2,3,4]].
[{1,1}, {1,2}, {1,3}, {1,4}, {2,1}, {2,2} | … }

> lists:filter(GT, ListOfPairs).
[{2,1},{3,1},{3,2},{4,1},{4,2},{4,3}]

> [X || X<-ListOfPairs, GT(X)].
[{2,1},{3,1},{3,2},{4,1},{4,2},{4,3}]
```

Fold

- Redukowanie listy do jednej wartości
- Dostarczona funkcja jest wołana dla każdego elementu
  - Ma do dyspozycji akumulator
  - Zwraca nowy akumulator

```
> Sum = fun (X, Y) -> X + Y end.

#Fun<erl_eval.12.80484245>

> lists:foldl(Sum, 0, [3,-4,-7,12,0,3,-5]).
2
```

Fold - przykład

```
> Div = fun (X, Y) -> X / Y end.
#Fun<erl_eval.12.90072148>

> lists:foldl(Div, 1, [2, 4]).
2.0
> lists:foldr(Div, 1, [2, 4]).
0.5
```

Fold - przykład

```
> MinMax = fun
   (X, {Min, Max}) when X < Min -> {X, Max};
   (X, {Min, Max}) when X > Max -> {Min, X};
   (_, V) -> V end.
#Fun<erl_eval.12.80484245>

> lists:foldl(MinMax, {0,0}, [2,-4,-7,12,0,3,-5]).
{-7,12}
```

Więcej...

- any/2 - sprawdzenie czy którykolwiek element spełnia warunek dostarczony jako funkcja
- all/2 - sprawdzenie czy wszystkie elementy spełniają warunek dostarczony jako funkcja
- takewile/2 – zwraca listę z początkowymi elementami spełniającymi warunek

```
> lists:any(fun(X) -> X > 7 end, [4,5,6,7,8]).
true

> lists:all(fun(X) -> X > 7 end, [4,5,6,7,8]).
false

> lists:takewhile(fun(X) -> X < 7 end, [4,5,6,7,8]).
[4,5,6]
```

Rekordy

Rekord

- Struktura danych przechowująca stałą liczbę elementów
- Dostęp do elementów za pomocą nazw
- To jedynie ułatwienie dla programistów – rekord to krotka
- Nie działają w shellu (choć można użyć BIFów *rd, rr, ...*)

```
-module(demo_01_record).
-export([testRecord/0]).

-record(grupa, {nazwa, licznosc, stan=aktywna}).

testRecord() ->
   Grupa1 = #grupa{nazwa="Gruppa 1", licznosc=12},
   Grupa2 = #grupa{nazwa="Gruppa 2", licznosc=7, stan=0},

   io:format(Grupa1#grupa.nazwa).
```

Modyfikowanie rekordów

- Czyli tworzenie nowego rekordy z istniejącego
- Ze zmienionymi wartościami części atrybutów
- W praktyce bardzo wygodna możliwość

```
-record(grupa, {nazwa, licznosc, stan=aktywna}).

   G1 = #grupa{nazwa="Gruppa 1", licznosc=7, stan=0},
   G2 = G1#grupa{nazwa="Gruppa 2"}.
```

Zagnieżdżanie rekordów

- Rekordy można zagnieżdżać dowolnie - to zwykłe krotki
- Składnia niezbyt przejrzysta...

```
-record(grupa, {nazwa, licznosc, stan=aktywna}).
-record(nadgrupa, {nadnazwa, grp}).

testNestedRecord() ->
Nad = #nadgrupa{
          nadnazwa = "Nad 3",
          grp = #grupa{nazwa="Gruppa 3", licznosc=7}},
    (Nad#nadgrupa.grp)#grupa.nazwa,
    Nad#nadgrupa.grp#grupa.nazwa.
```

Rekordy – dopasowywanie do wzorców

- Również dla wzorców zagnieżdżonych

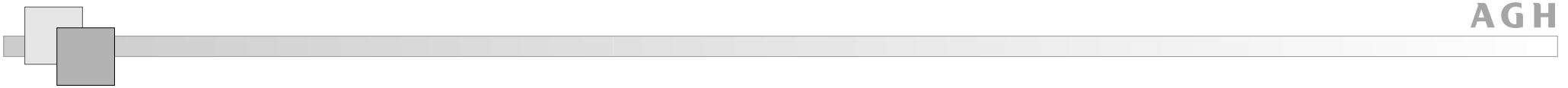
```
-record(grupa, {nazwa, licznosc, stan=aktywna}).
-record(nadgrupa, {nadnazwa, grupa}).

testPatterns(#grupa{nazwa = Nazwa, licznosc = 7}) ->
   Nazwa;

testPatterns(#grupa{licznosc = Licznosc})
                    when Licznosc > 1 ->
   Licznosc;

testPatterns(#nadgrupa{nadnazwa = NadNazwa,
                    grp = #grupa{nazwa = Nazwa}}) ->
   NadNazwa ++ Nazwa.
```

Rekordy – cukier syntaktyczny

- Rekord to zwykła krotka
- Pomaga w przypadku edycji zawartości krotki
- Bardzo ułatwia rozbudowę programu
- Erlang Programming Rules and Conventions:

  Use records as the principle data structure!

  The record features of Erlang can be used to ensure cross module consistency of data structures and should therefore be used by interface functions when passing data structures between modules.



Słowniki

Słownik key-value

- Moduł dict
- Podstawowe tryby przechowywania:
  - append(Key, Value, Dict1) - lista wartości pod kluczem
  - store(Key, Value, Dict1) - tylko jedna wartość pod kluczem

```
D = dict:new().
D1 = dict:append(key1, val1, D).
D2 = dict:append(key2, val2, D1).
D4 = dict:append(key1, val3, D2).
dict:fetch(key1, D4).
  → [val1,val3]
D5 = dict:store(key1, val4, D4).
dict:fetch(key1, D5).
  → val4
```

Słownik key-value – przetwarzanie

- filter(Pred, Dict1) -> Dict2
- map(Fun, Dict1) -> Dict2
- fold(Fun, Acc0, Dict) -> Acc1
- merge(Fun, Dict1, Dict2) -> Dict3
- to_list(Dict) -> List
- from_list(List) -> Dict

```
dict:to_list(D4).
  → [{key1,[val1,val3]},{key2,[val2]}]
D8 = dict:filter(fun(_,[val2])->false;
                    (_,_)->true end, D4).
dict:to_list(D8).
  → [{key1,[val1,val3]}]
```

Mapy

- Nowy typ danych wprowadzony w Erlangu 17, działa od 18
- Ma zastąpić rekordy i słowiki dict
- Bardziej wydajny

```
M1 = #{}.

M2 = #{key => "Val", "Ala" => "Makot", {a,b}=>[c,"D"]}.

M3 = M2#{key => "New Value", z=>z}.

M4 = M3#{key := "Even newer value"}.

M5 = M4#{key2 := "Newerer value"}.
** exception error: {badkey,key2}
```

Mapy

- Nie wszystko działa...

```
Value = M4#{key}.
* 1: syntax error before: '}'


Value = maps:get(key, M4).
```

Mapy – przetwarzanie

```
M4.
#{key => "Even newer value",
   z => z,
   {a,b} => [c,"D"],
   "Ala" => "Makot"}

Fu = fun(K,V) -> is_atom(V) end.

maps:map(Fu, M4).
#{key => false,z => true,{a,b} => false,"Ala" => false}

maps:filter(Fu, M4).
#{z => z}
```

Mapy – pattern matching

```
M4.
#{key => "Even newer value",
   z => z,
   {a,b} => [c,"D"],
   "Ala" => "Makot"}



#{key := Value} = M4.
Value.
"Even newer value"

#{{a,b} := [c, String]} = M4.
String.
"D"

#{{a,b} := [c, String], key := Value} = M4.
```

Mapy – pattern matching

```
MX = #{a=>b,#{c=>d}=>e,f=>#{g=>h}}.
#{f:=A} = MX.
#{f:=#{g:=B}} = MX.

#{A:=b} = MX.
* 1: variable 'A' is unbound
```

Obsługa błędów

Błędy wykonania programu

- Kilka sytuacji, które interpreter musi obsłużyć
- Pojawienie się błędu wykonania kończy proces w sposób nadzwyczajny

Błąd dopasowania argumentów funkcji

- *function_clause*
- Wystąpi, jeśli nie zostanie odnaleziona klauzula funkcji którą da się dopasować do podanych danych
- Przyczyny:
  - Wywołanie funkcji z błędnymi parametrami
  - Brak rozpatrzenia jakiegoś przypadku

```
1> lists:seq(1,abc).
  ** exception error: no function clause
         matching lists:seq(1,abc)
```

Błąd dopasowania instrukcji *case*

- *case_clause*
- Wystąpi, jeśli nie zostanie odnaleziony wzorzec w instrukcji ***case*** który da się dopasować do podanych danych
- Przyczyna: brak rozpatrzenia jakiegoś przypadku

```
caseErr(N) ->
  case N of
    1 -> 1;
    2 -> 2
  end.


6> err:caseErr(3).
** exception error: no case clause matching 3
   in function  err:caseErr/1
```

Błąd dopasowania instrukcji *if*

- *if_clause*
- Wystąpi, jeśli nie zostanie odnaleziony wzorzec w instrukcji ***if*** który da się dopasować do podanych danych
- Przyczyna: brak rozpatrzenia jakiegoś przypadku

```
ifErr(N) ->
 if
    N > 0 -> 1;
    N < 0 -> -1
 end.


n9> err:ifErr(0).
** exception error: no true branch found
   when evaluating an if expression
   in function  err:ifErr/1
```

Błąd dopasowania

- *badmatch*
- Wystąpi, jeśli dopasowanie do wzorca zawiedzie
- Przyczyny:
  - Próba związania związanej zmiennej
  - Błędny format danych zwracanych przez funkcję

```
10> N = 1.
1
11> N = 2.
** exception error: no match of right hand side value 2
12> {N, M} = {3, 1}.
** exception error: no match of right hand side value {3,1}
```

Błąd argumentu

- *badarg*
- Wystąpi, jeśli BIF zostanie wywołany z błędnymi argumentami

```
13> tuple_to_list(alaMaKota).
** exception error: bad argument
   in function  tuple_to_list/1
      called as tuple_to_list(alaMaKota)

14> spawn(lists, seq, 123).
** exception error: bad argument
   in function  spawn/3
      called as spawn(lists,seq,123)
```

Niezdefiniowana funkcja

- *undef*
- Wystąpi, jeśli wywołana zostanie funkcja, której interpreter nie rozpozna
- Przyczyny:
  - Brak deklaracji -*export*
  - Brak nazwy modułu w wywołaniu
  - Literówka w nazwie

```
21> abc:def(ghi).
  ** exception error: undefined function abc:def/1
22> lists:last([1,2,3,4]).
  4
23> lists:last([1,2,3,4],3).
  ** exception error: undefined function lists:last/2
```

Błąd arytmetyczny

- *badarith*
- Wystąpi, jeśli wyrażenie arytmetyczne zostało wywołane dla niepoprawnych argumentów
- Przyczyny:
  - Błędny typ argumentu
  - Dzielenie przez zero

```
27> 2 / 0.
** exception error: bad argument in an arithmetic expression
     in operator  '/'/2
        called as 2 / 0
28> 1 + a.
** exception error: bad argument in an arithmetic expression
     in operator  +/2
        called as 1 + a
29> 1 rem 3.0.
** exception error: bad argument in an arithmetic expression
     in operator  rem/2
        called as 1 rem 3.0
```

Wszystkie błędy

badarg	Bad argument. The argument is of wrong data type, or is otherwise badly formed.
badarith	Bad argument in an arithmetic expression.
{badmatch,V}	Evaluation of a match expression failed. The value V did not match.
function_clause	No matching function clause is found when evaluating a function call.
{case_clause,V}	No matching branch is found when evaluating a case expression. The value V did not match.
if_clause	No true branch is found when evaluating an if expression.
{try_clause,V}	No matching branch is found when evaluating the of-section of a try expression. The value V did not match.
undef	The function cannot be found when evaluating a function call.
{badfun,F}	There is something wrong with a fun F.
{badarity,F}	A fun is applied to the wrong number of arguments. F describes the fun and the arguments.
timeout_value	The timeout value in a receive..after expression is evaluated to something else than an integer or infinity.
noproc	Trying to link to a non-existing process.
{nocatch,V}	Trying to evaluate a throw outside a catch. V is the thrown term.
system_limit	A system limit has been reached. See Efficiency Guide for information about system limits.

Przechwytywanie błędów

- *Value* = ***catch*** *Expression*
- Jeśli *Expression* wykona się poprawnie, *Value* zostanie związaną z wartością *Expression*
- Jeśli zakończy się błędem...

```
36> V1 = 1 / 1.
  1.0
37> V2 = 1 / 0.
  ** exception error: bad argument in an arithmetic expression
     in operator  '/'/2
        called as 1 / 0
38> V3 = (catch 1 / 0).
  {'EXIT',{badarith,[{erlang,'/',[1,0]},
                     {erl_eval,do_apply,5},
                     {erl_eval,expr,5},
                     {erl_eval,expr,5},
                     {shell,exprs,7},
                     {shell,eval_exprs,7},
                     {shell,eval_loop,3}]}}
```

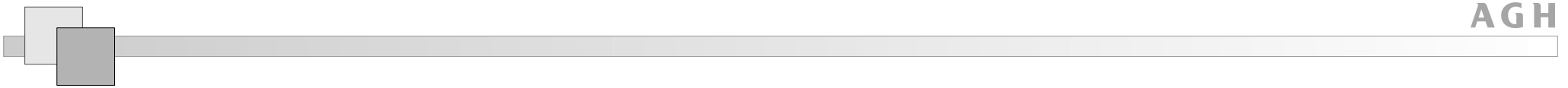
Wyjątki

- Klasy wyjątków:
  - *error* – Run-time error lub jawne wywołanie *erlang:error/1,2*
  - *exit* – Proces wywołał *exit/1* – służy do sygnalizowania błędów pomiędzy procesami
  - *throw* – Proces wywołał *throw/1* – służy do generowania błędów wewnątrz programu

Przechwytywanie wyjątków

- *try … catch … end*

```
catchme(N)  ->
   try  generate_exception(N)  of
      Val  ->  {N,  normal,  Val}
   catch
      throw:X  ->  {N,  thw,  X};
      exit:X   ->  {N,  ext,  X};
      error:X  ->  {N,  err,  X}
   end.


generate_exception(1) -> a;
generate_exception(2) -> throw(a);
generate_exception(3) -> exit(a);
generate_exception(4) -> erlang:error(a);
generate_exception(5) -> {'EXIT',  a};
generate_exception(6) -> 1/0;
generate_exception(7) -> list:seq(1,asd).
```



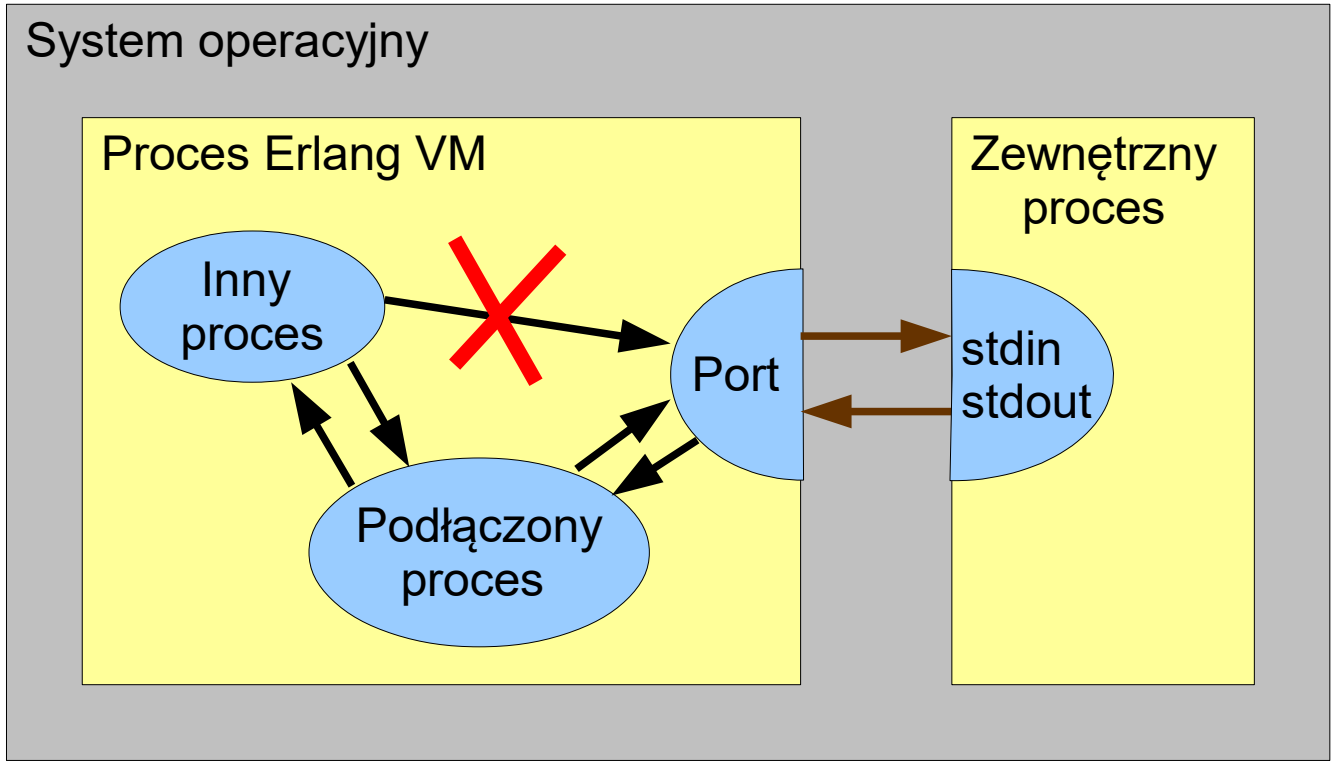
Interfejsy do innych technologii

Możliwości

- Porty
- Linked-in Drivers
- Interfejsy w języku C
- Węzły innych technologii
- Standardowe protokoły sieciowe

Porty

- Proces Erlanga uruchamia inny proces systemowy
- Komunikacja za pomocą std in/out
- Protokół realizuje programista



Linked-in drivers

- Metoda łączenia programu napisanego w C z Erlangiem
- Program w C jest uruchamiany w kontekście maszyny wirtualnej Erlanga
- Program w C musi implementować określone funkcje
- UWAGA: błąd programu w C → zatrzymanie całej maszyny Erlanga
- Mogą działać asynchronicznie względem procesów Erlanga
- erl_driver, erl_ddll

```
erl_ddll:load_driver(Path, ProgramName)
```

NIF

- Natively Implemented Functions
- Program w C jest uruchamiany w kontekście maszyny wirtualnej Erlanga
- Program w C zastępuje określone funkcje modułu
- UWAGA: błąd programu w C → zatrzymanie całej maszyny Erlanga
- Działają synchronicznie, blokują scheduler Erlanga

```
-on_load(init/0).

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

Erl_Interface

- Biblioteka funkcji i struktur pozwalająca na:
  - łatwiejsze integrowanie programów w C dzięki funkcjom do serializacji i deserializacji danych z portów
  - uruchamianie własnych węzłów Erlang VM, napisanych w języku C

```
erl_connect_init(1, "secretcookie", 0);

status = erl_receive_msg(fd, buf, BUFSIZE, &emsg);
fromp = erl_element(2, emsg.msg);
tuplep = erl_element(3, emsg.msg);

resp = erl_format("{cnode, ~i}", res);
erl_send(fd, fromp, resp);
```

jinterface

- Biblioteka klas, która pozwala na uruchamianie własnych węzłów Erlang VM, napisanych w języku Java

```
private void init() throws IOException {
   OtpNode self = new OtpNode(getNodeName());
   mbox = self.createMbox(getMboxName());
}

private void reveiveAndSend() {
  OtpErlangObject o = mbox.receive();
  OtpErlangTuple msg = (OtpErlangTuple) o;
  OtpErlangPid from = (OtpErlangPid) msg.elementAt(0);

  OtpErlangTuple tuple = new OtpErlangTuple(eoa);
  mbox.send(from, tuple);
}
```

TCP

- gen_tcp
  - connect(Address, Port, Options)
  - listen(Port, Options)
  - accept(ListenSocket)
  - send(Socket, Packet)
  - recv(Socket, Length)
  - close(Socket)
  - controlling_process(Socket, Pid)

TCP - serwer

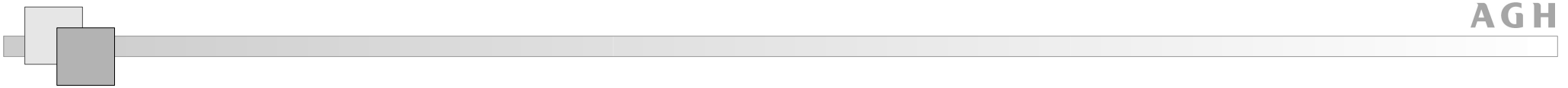
```
server() ->
    {ok, LSock} = gen_tcp:listen(5678, [binary, {packet, 0},
                                        {active, false}]),
    {ok, Sock} = gen_tcp:accept(LSock),
    {ok, Bin} = do_recv(Sock, []),
    ok = gen_tcp:close(Sock),
    Bin.
do_recv(Sock, Bs) ->
    case gen_tcp:recv(Sock, 0) of
        {ok, B} ->
            do_recv(Sock, [Bs, B]);
        {error, closed} ->
            {ok, list_to_binary(Bs)}
    end.
```

TCP - klient

```
client() ->
    SomeHostInNet = "localhost"
    {ok, Sock} = gen_tcp:connect(SomeHostInNet, 5678,
                                 [binary, {packet, 0}]),
    ok = gen_tcp:send(Sock, "Some Data"),
    ok = gen_tcp:send(Sock, "More Data"),
    ok = gen_tcp:close(Sock).
```

UDP

- gen_udp
  - open(Port, Opts)
  - send(Socket, Packet)
  - recv(Socket, Length)
  - close(Socket)
  - controlling_process(Socket, Pid)



Preprocesor

Preprocesor

- Poza funkcjami, w plikach *.erl* mogą wystąpić dyrektywy preprocesora

```
-module(increaser).
-export([increase/1]).
```

- Włączanie pliku do źródła innego pliku:

```
-include(File).
```

- Wykorzystywane najczęściej do włączania plików z definicjami rekordów i makr (pliki *.hrl*)

Makra

- Dyrektywa preprocesora pozwala definiować makra:

```
-define(MaxCount, 100).
-define(Tuple3(A), {A,A*2,A*3}).
```

- Które możemy użyć w źródle:

```
lessThenMax(X) when X < ?MaxCount ->
   ?Tuple3(X).
```

Predefiniowane Makra

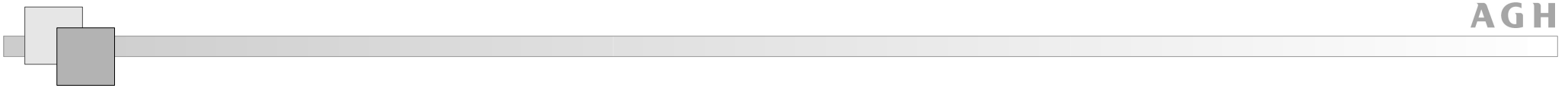
?MODULE

?FILE

?LINE

?FUNCTION_NAME

...



Testowanie

Testy jednostkowe

- Testowanie – po co?
- TDD

- EUnit – biblioteka do tworzenia i uruchamiania testów jednostkowych
- Jednostka – funkcja, moduł, cała aplikacja
- Inspirowany przez JUnit 3
- W założeniu nie wymaga ingerencji w testowany kod

EUnit – przykład

```
-module(increaser).
-export([increase/1]).

increase(X) when is_integer(X) or is_float(X) -> X + 1;
increase([H|T]) when is_integer(H) or is_float(H) -> [H+1 | increase(T)];
increase([H|T]) when is_list(H) -> [increase(H) | increase(T)];
increase([H|T]) -> [H | increase(T)];
increase(X) -> X.
```

```
-module(increaser_test).
-include_lib("eunit/include/eunit.hrl").
-compile(export_all).

increase_integer_test() -> 4 = increaser:increase(3).
increase_empty_list_test() -> [] = increaser:increase([]).
increase_list_test() -> [2,3,4,aaa,5] = increaser:increase([1,2,3,aaa,4]).
increase_nested_list_test() ->
      [1,[2],[3,4],[]] = increaser:increase([0,[1],[2,3],[]]).
```

EUnit – przykład

```
> l(increaser).
   {module,increaser}
> increaser:increase([1,2,3,[4,5]]).
   [2,3,4,[5,6]]
> l(increaser_test).
   {module,increaser_test}
> increaser_test:test().
   All 4 tests passed.
   ok
```

EUnit – przykład

```
increase_float_test() -> 4.04 = increaser:increase(3.03).
increase_sublist_test() ->
       [[[[[1]]]]] = increaser:increase([[[[[[0]]]]]]).
```

```
> increaser_test:test().
increaser_test: increase_sublist_test...*failed*
in function increaser_test:increase_sublist_test/0 (increaser_test.erl, line 15)
**error:{badmatch,[[[[[[1]]]]]]}


increaser_test: increase_float_test...*failed*
in function increaser_test:increase_float_test/0 (increaser_test.erl, line 14)
**error:{badmatch,4.029999999999999}


=======================================================
  Failed: 2.  Skipped: 0.  Passed: 4.
error
```

Makra testów

- Więcej informacji o błędach
- ?assert(X), ?assertNot(X)
- ?assertEqual, ?assertMatch, ?assertException, ...

```
increase_boolean_test() ->
    ?assert(increaser:increase(true)).

increase_integer_test() ->
    ?assertEqual(4, increaser:increase(3)).

increase_match_list_test() ->
    ?assertMatch([_,5]) = increaser:increase([[aaa],4]).
```

Makra testów

```
increase_integer_test() ->
      ?assertEqual(5, increaser:increase(3)).
```

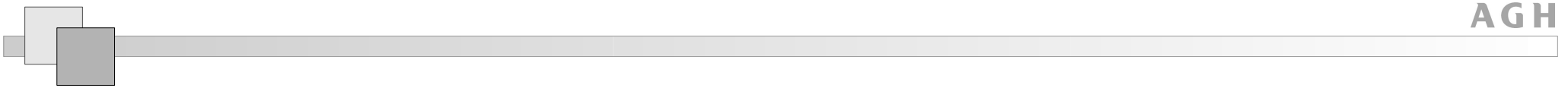
```
> increaser_test:test().
increaser_test: increase_integer_test (module 'increaser_test')...*failed*
in function increaser_test:'-increase_integer_test/0-fun-0-'/1
                                        (increaser_test.erl, line 18)
**error:{assertEqual_failed,[{module,increaser_test},
                    {line,18},
                    {expression,"increaser : increase ( 3 )"},
                    {expected,5},
                    {value,4}]}

=======================================================
  Failed: 1.  Skipped: 0.  Passed: 0.
error
```

Testowanie ze stanem

- Testowanie programów funkcyjnych jest teoretycznie łatwe – funkcja zawsze zwraca ten sam wynik dla tych samych argumentów
- Testowanie programów ze stanem – znacznie trudniej...
- Fixtures: stan potrzebny do testowania modułu, mechanizm analogiczny do @before

Inne rodzaje testów

- Common Test – rozbudowane narzędzie do testów systemów scentralizowanych i rozproszonych dużej skali
  - Mechanizmy uruchamiania równoległego, losowego
  - Grupy testów, raporty, …
- Property-Based Testing - QuickCheck – automatyczne generowanie testów jednostkowych
  - Zestaw generatorów do określonych wartości
  - Zestaw strategii do wybierania konkretnych wartości do testowania.



- Joe Armstrong...