@drkrab

Erjang — a JVM-based Erlang VM
A Journey into Erlang Land

Kresten Krab Thorup
Trifork CTO

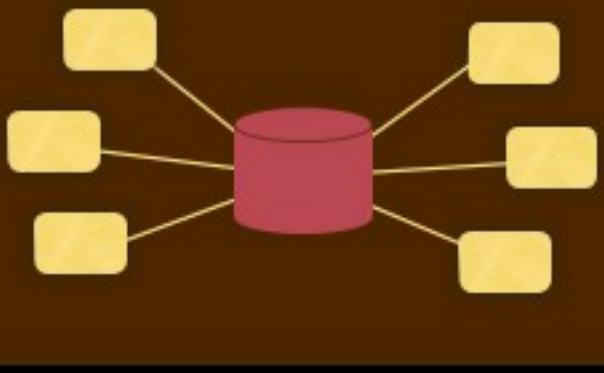
software pilots
TRIFORK.

The Desert of Java

Java was designed
in the client-server age

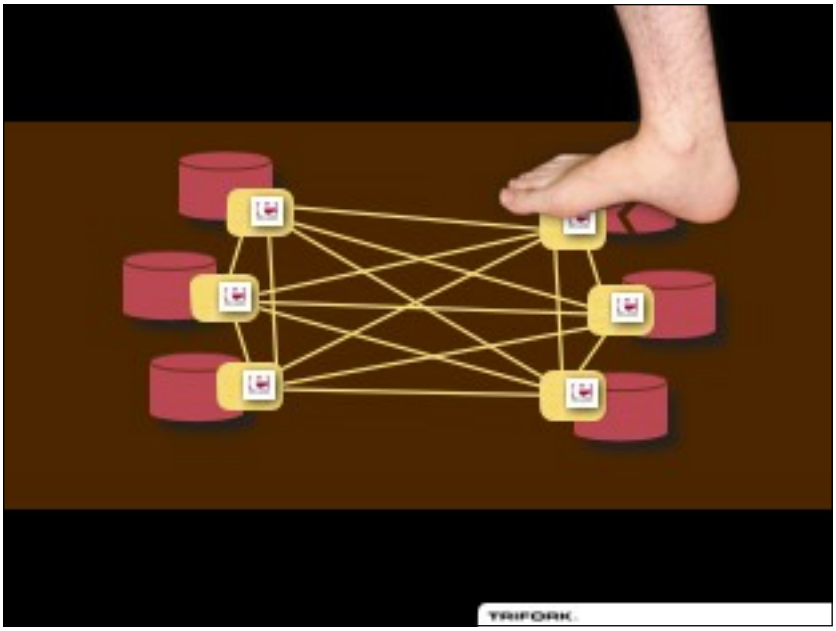
Synchronous coordination prevailed

TRIFORK



TRIFORK

TRIFORK



Erlang's raison d'être

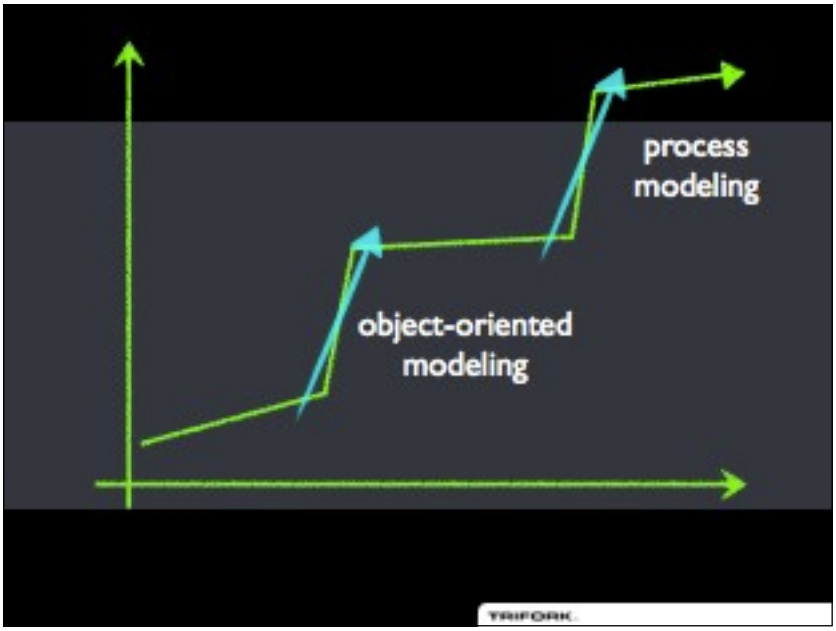
Build reliable systems
in the presence of errors

⇒ Isolation + Concurrency

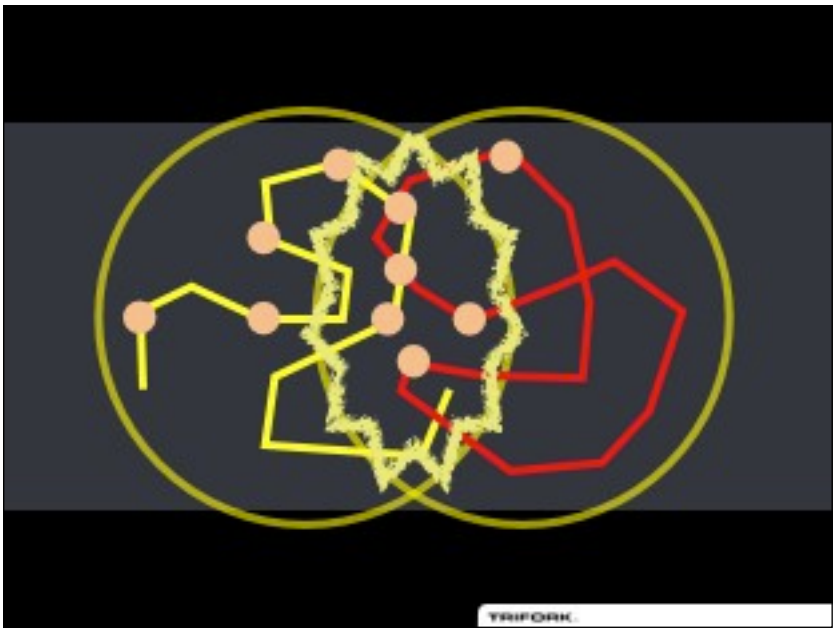
TRIFORK

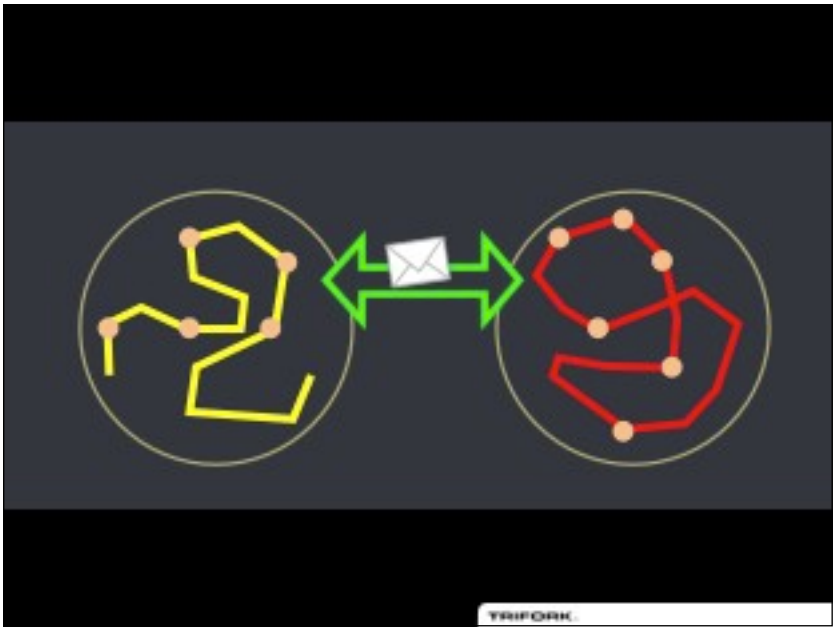
process modeling

TRIFORK

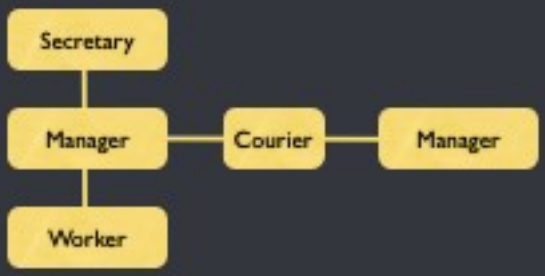

process
modeling
object-oriented
modeling
TRIFORK


1UP
TRIFORK


TRIFORK



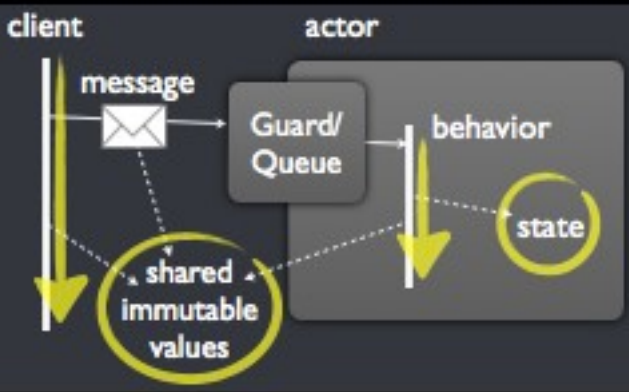
Anthropomorphic Programming

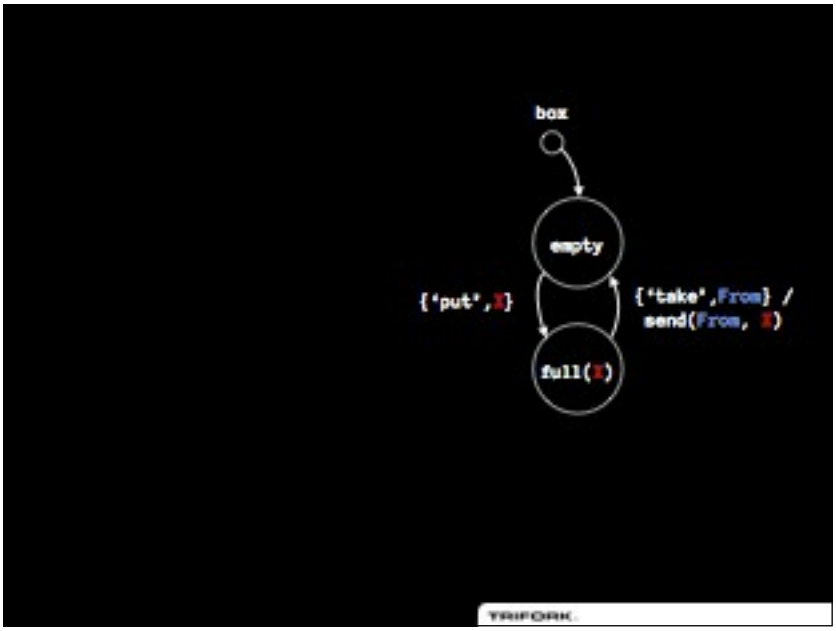


Morven Gentleman, SP&E 1981; Thomas & Barry ICT 2004

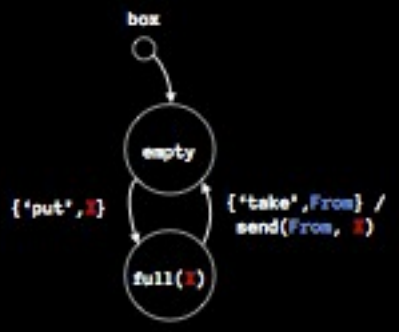
Hierachical Organizations

Process Aggregation

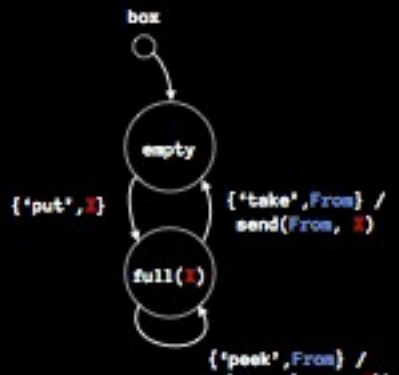




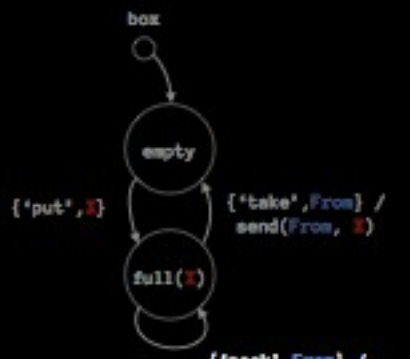

```
box() → empty().

empty() →
  receive
    {'put', X} →
        full(X)
  end.

full(X) →
  receive
    {'take', From} →
        From ! X,
        empty()
  end.
```




```
box() → empty().

empty() →
  receive
    {'put', X} →
        full(X)
  end.

full(X) →
  receive
    {'take', From} →
        From ! X,
        empty()
  end.
```



```
box() → empty().

empty() →
  receive
   {'put', X} →
        full(X)
  end.

full(X) →
  receive
   {'take', From} →
        From ! X,
        empty();
   {'peek', From} →
        From ! {'ok', X},
        full(X)
  end.
```




```
box() → empty().

empty() →
  receive
   {'put', X} →
        full(X)
  end.

full(X) →
  receive
   {'take', From} →
        From ! X,
        empty();
   {'peek', From} →
        From ! {'ok', X},
        full(X)
  end.
```

```
Box = spawn(box),

Box ! {'put', 27},

Box ! {'take', self()},
receive
  Value → print(Value)
end
```

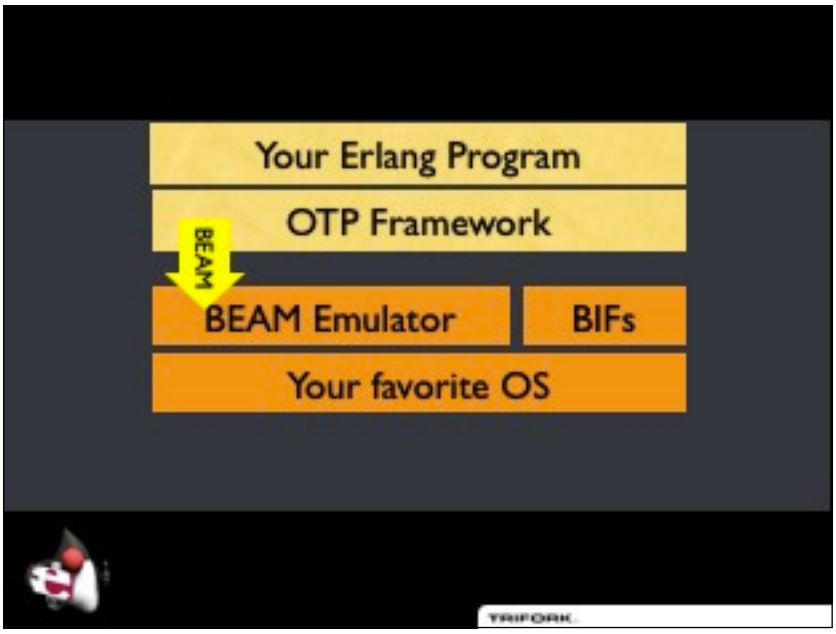


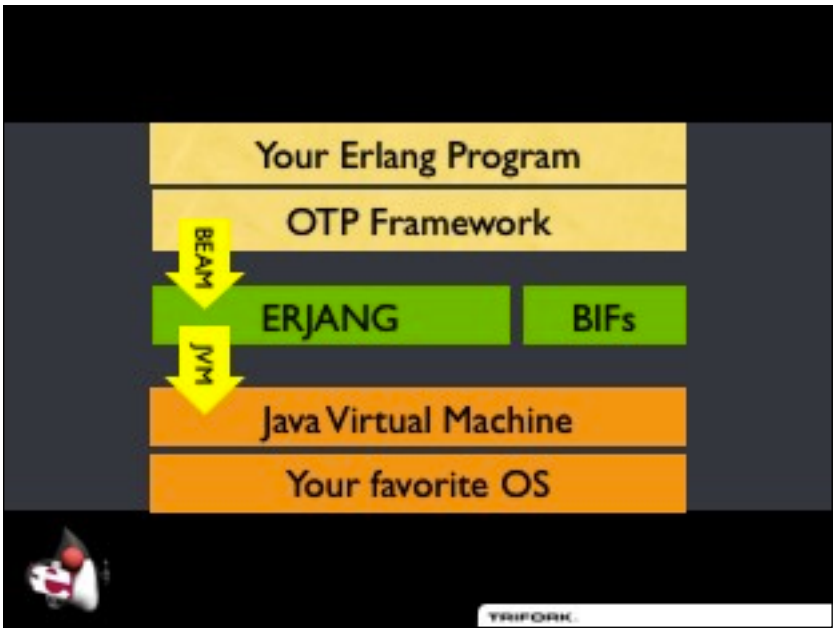
```
box() → empty().

empty() →
  receive
   {'put', X} →
        full(X)
  end.

full(X) →
  receive
   {'take', From} →
        From ! X,
        empty();
   {'peek', From} →
        From ! {'ok', X},
        full(X)
  end.
```

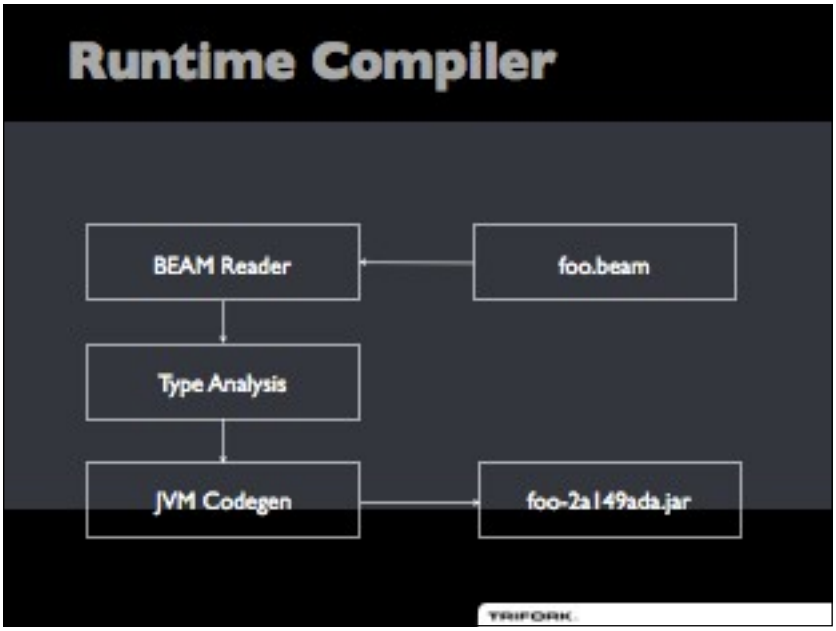
```
Box = spawn(box),

Box ! {'put', 27},

Box ! {'take', self()},
receive
  Value → print(Value)
end
```

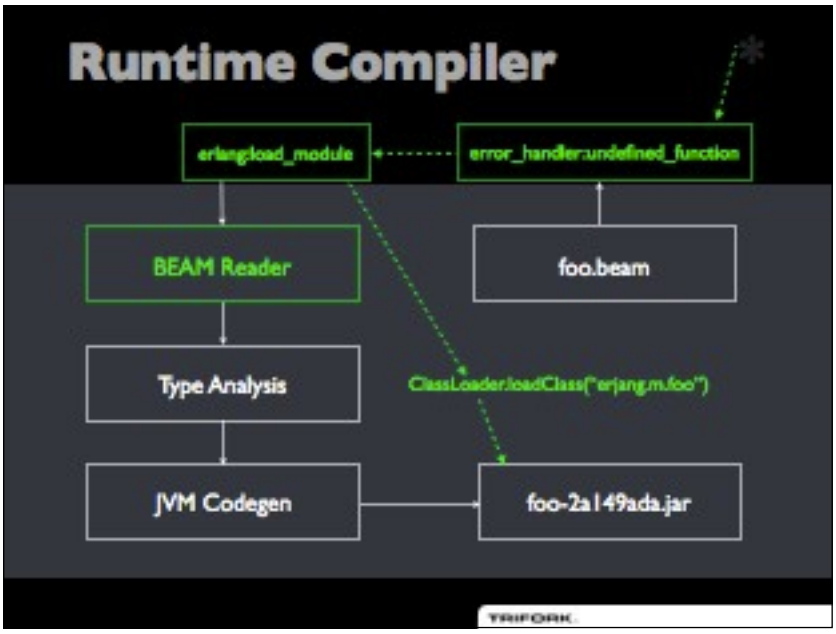
TRIFORK

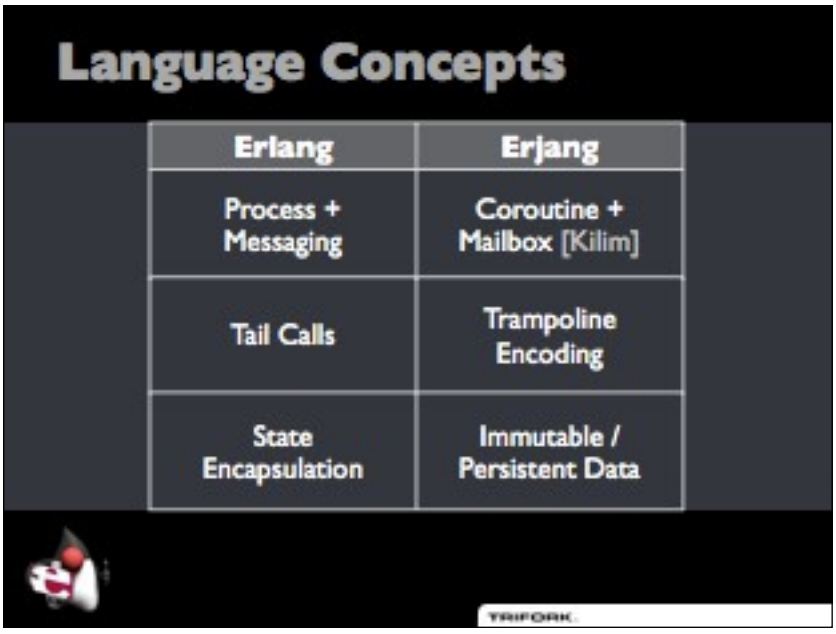

TRIFORK


Your Erlang Program
OTP Framework
BEAM
BEAM Emulator
BIFs
Your favorite OS
TRIFORK


Your Erlang Program
OTP Framework
BEAM
ERJANG
BIFs
JVM
Java Virtual Machine
Your favorite OS
TRIFORK


Runtime Compiler
BEAM Reader
foo.beam
Type Analysis
JVM Codegen
foo-2a149ada.jar
TRIFORK


Runtime Compiler
erlang:load_module
error_handler:undefined_function
BEAM Reader
foo.beam
Type Analysis
ClassLoader.loadClass("erjang.m.foo")
JVM Codegen
foo-2a149ada.jar
TRIFORK


Language Concepts
Erlang
Erjang
Process + Messaging
Coroutine + Mailbox [Kilim]
Tail Calls
Trampoline Encoding
State Encapsulation
Immutable / Persistent Data
TRIFORK

Tail Calls

```
-module(bar).

bat([H | T], T2) ->
  bat(T, foo(H, T2));

bat([], T2) -> T2.

foo(H, T) ->
  lists:reverse(H ++ T).
```

The BEAM Code

```
{function, bat, {nargs,2}}.
{label,264}.
  {test,is_nonempty_list,{else,265},[{x,0}]}.
  {get_list,{x,0},{x,0},{y,0}}.
  {call,2,foo}.
  {move,{x,0},{x,1}}.
  {move,{y,0},{x,0}}.
  {call_last,2,bat,1}.
{label,265}.
  {test,is_nil,{else,263},[{x,0}]}.
  {move,{x,1},{x,0}}.
  return.
{label,263}.
  {func_info,{atom,appmon_bar},{atom,bat},2}.
```


```
public static EObject
    bat___2(EProc eproc, EObject arg1, EObject arg2)
{
   ECons cons; ENil nil;
  tail:
    if((cons = arg1.test_nonempty_list()) != null) {
      // extract list
      EObject hd = cons.head();
      EObject tl = cons.tail();
      // call foo/2
      EObject tmp = foo___2(eproc, hd, arg2);
      // self-tail recursion
      arg1 = tl;
      arg2 = tmp;
      goto tail;
    } else if ((nil = arg1.test_nil()) != null) {
      return arg2;
    }
   throw ERT.badyc_info(am_bar, am_bat, 2);
}
```

Erlang ⇒ JVM

```
-module(bar).

bat([H | T], T2) ->
  bat(T, foo(H, T2));

bat([], T2) -> T2.
```

```
foo(H, T) ->
  lists:reverse(H ++ T).
```

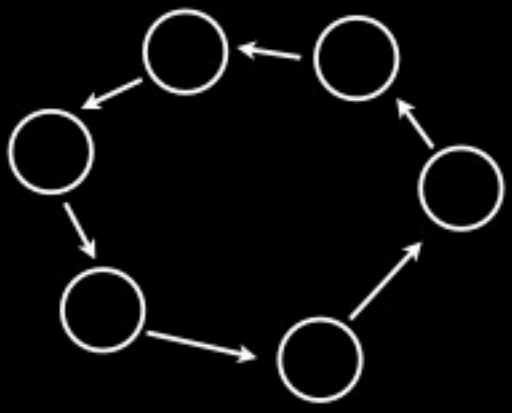

```
foo(H, T) ->
  lists:reverse(H ++ T).
```

```
public static EObject
    foo__2(EProc p, EObject H, EObject T)
{
    EObject r = foo__2$body(p,H,T);
    while (r == TAIL_MARKER) {
       r = p.tail.go();
    }
    return r;
}
```

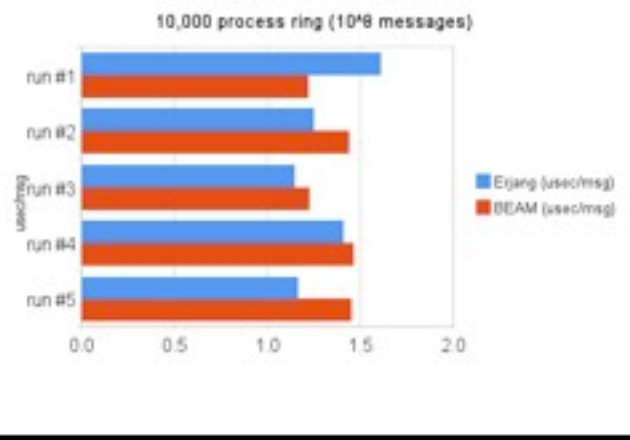

```
foo(H, T) ->
  lists:reverse(H ++ T).
```

```
public static
    EObject foo__2$body(EProc p, EObject H, EObject T)
{
    // Tmp = erlang:'++'(H,T)
    EObject tmp = erlang_append__2.invoke(p,H,T);

    // return lists:reverse(Tmp)
    p.tail = lists__reverse_1;
    p.arg1 = tmp;
    return TAIL_MARKER;
}
```

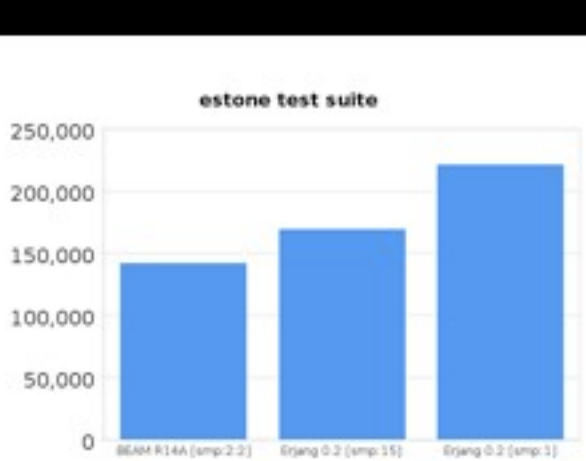
The ring!



TRIFORK



TRIFORK



TRIFORK

Interfacing to Java

- Erlang's primitive operations "BIFs" are implemented in Java
- @BIF annotation makes a static-public method available from Erlang.
- Erlang port concept for "drivers"

Example BIF

```
// foo:bar(...) native Bodyction

package erjang.m.foo;
class foo extends ENative {

  @BIF public static
  EObject bar(EProc proc, EObject arg1, arg2, ...)  {


  }
}
```

Example BIF

```
@BIF public static EObject
spawn_link(EProc proc, EObject mod, EObject fun, EObject args)
     throws Pausable {
        EAtom m = mod.testAtom();
        EAtom f = fun.testAtom();
        ESeq  a = args.testSeq();

        if (m==null||f==null||a==null)
                throw ERT.badarg(mod, fun, args);

        EProc p2 = new EProc(proc.group_leader(), m, f, a);
        p2.link_to(proc);
        ERT.run(p2);

        return p2.self_handle();
}
```

Interfacing to Java

```
demo() ->
    Map = 'java.util.HashMap':new(),
    Map:put('x', "4"),
    Map:put(1, 'foo'),
    print(Map).

print([]) -> ok;
print([{Key,Val}|Tail]) ->
    io:format("key=~p, value=~p~n",
                    [Key,Val]),
    print(Tail).
```

Interfacing to Java

```
java.util.Map      [{Key,Value}, ...]
                 | fun(Key) -> Value
java.util.List     [Elem0, Elem1, ...]
Object[]           [Elem0, Elem1, ...]
“foo”              “foo” (a.k.a. [97,98,99] )
java.lang.Runnable fun() -> value

fun(Op,Parms,Args) -> interface
```

"The world *is* concurrent. ... I could not drive my car on the highway if I did not intuitively understand the notion of concurrency..." —Joe Armstrong

http://www.youtube.com/watch?v=N0OUqYZMEZk

TRIFORK