@drkrab

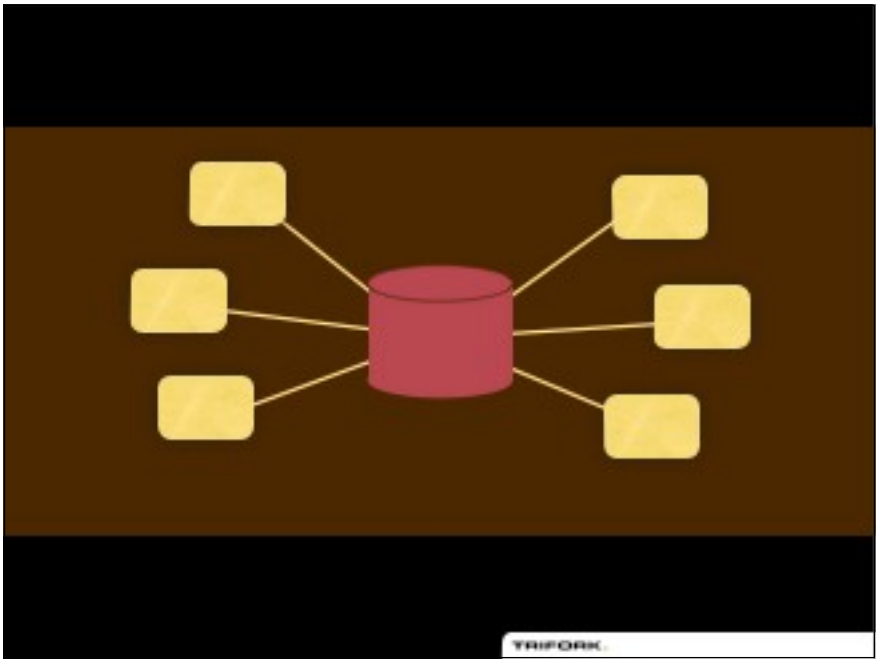
Erjang — a JVM-based Erlang VM
A Journey into Erlang Land

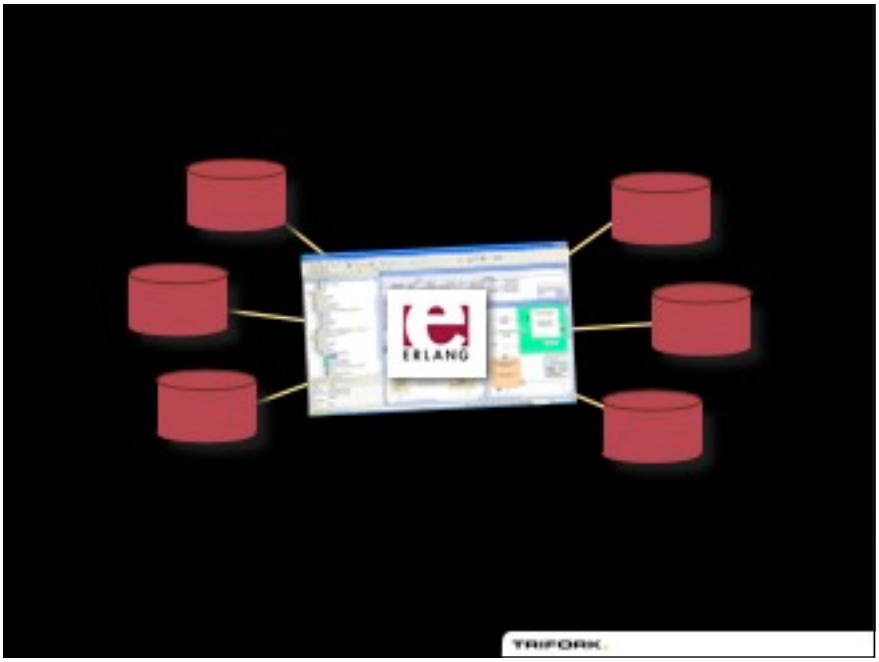
Kresten Krab Thorup
Trifork CTO

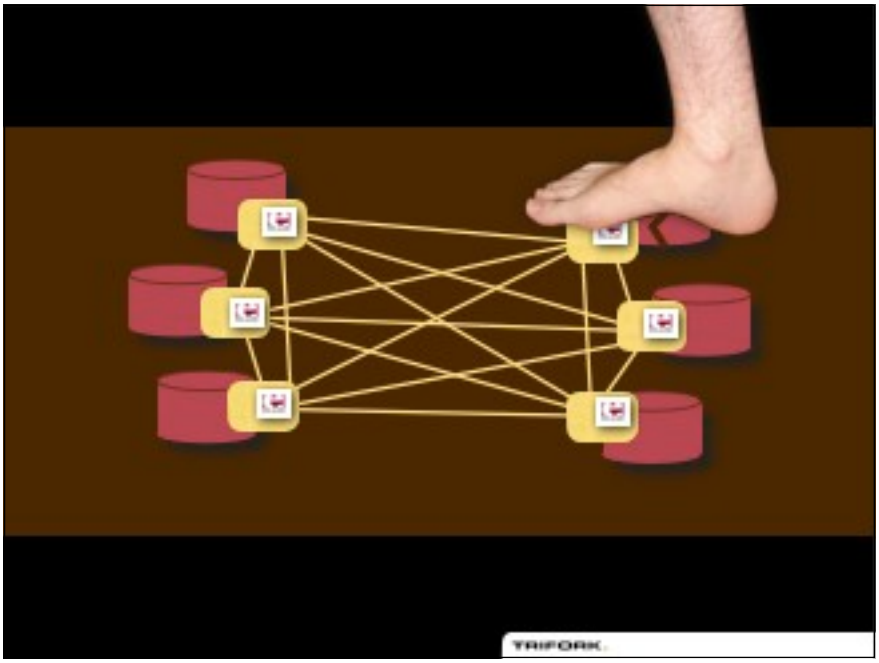
software pilots
TRIFORK.

Java was designed
in the client-server age

Synchronous
coordination
prevailed


TRIFORK


ERLANG
TRIFORK


TRIFORK

Erlang's raison d'être

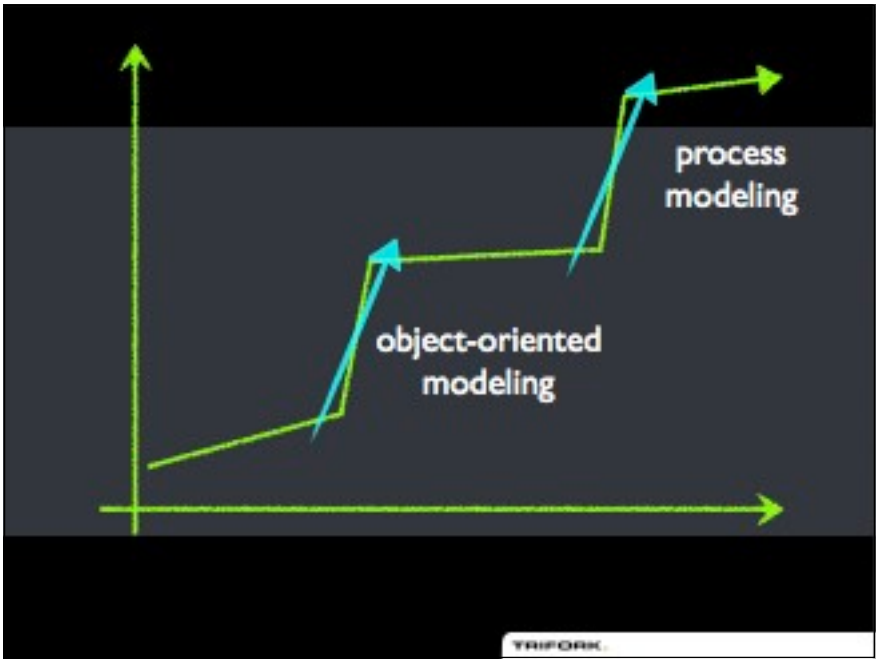
Build reliable systems
in the presence of errors

⇒ Isolation + Concurrency

TRIFORK

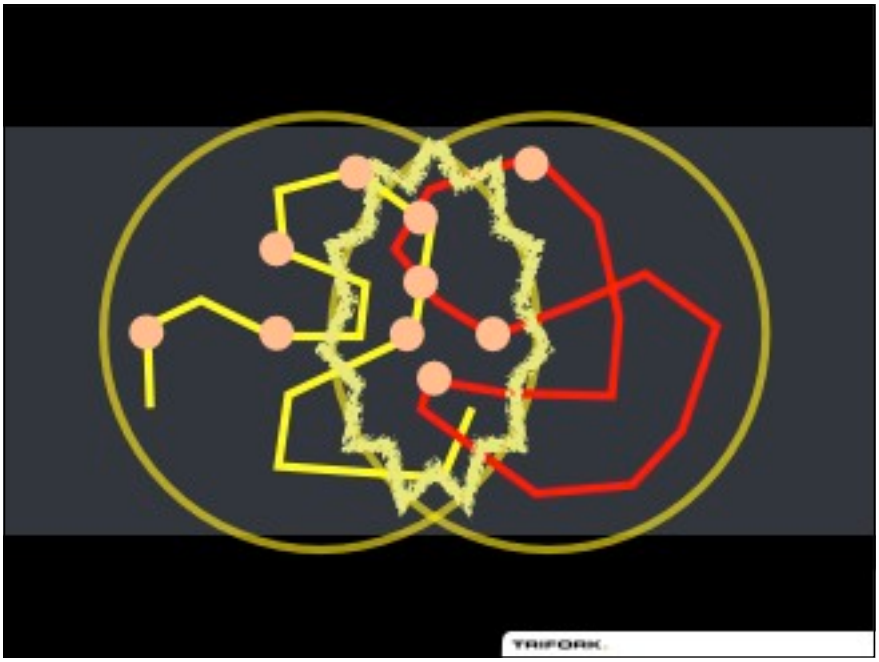
process modeling

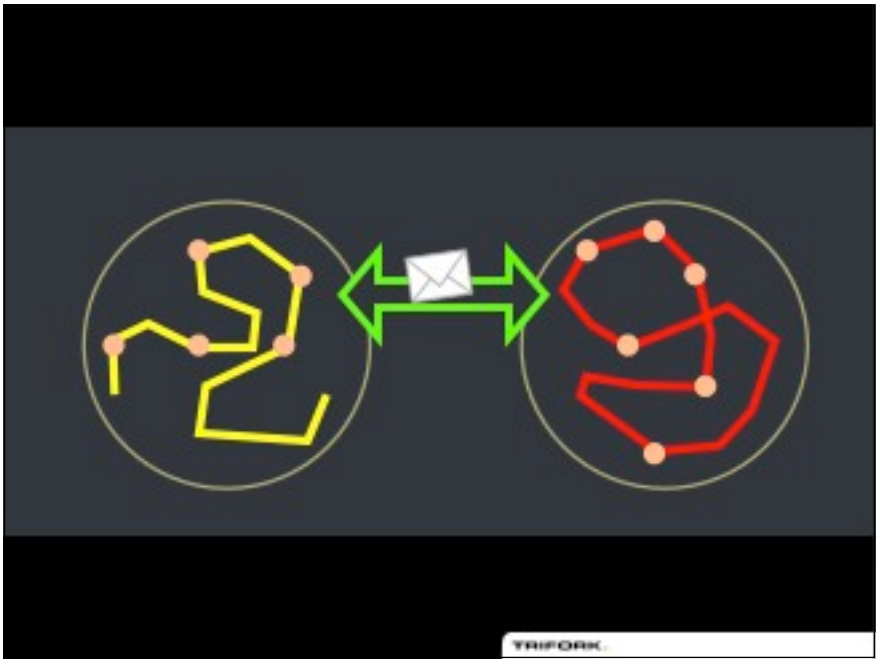
TRIFORK

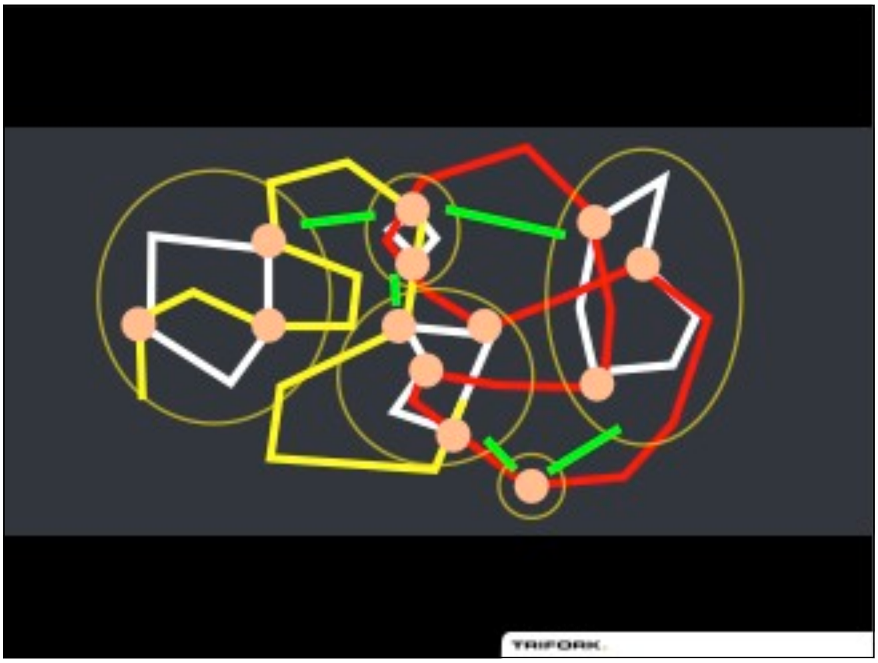


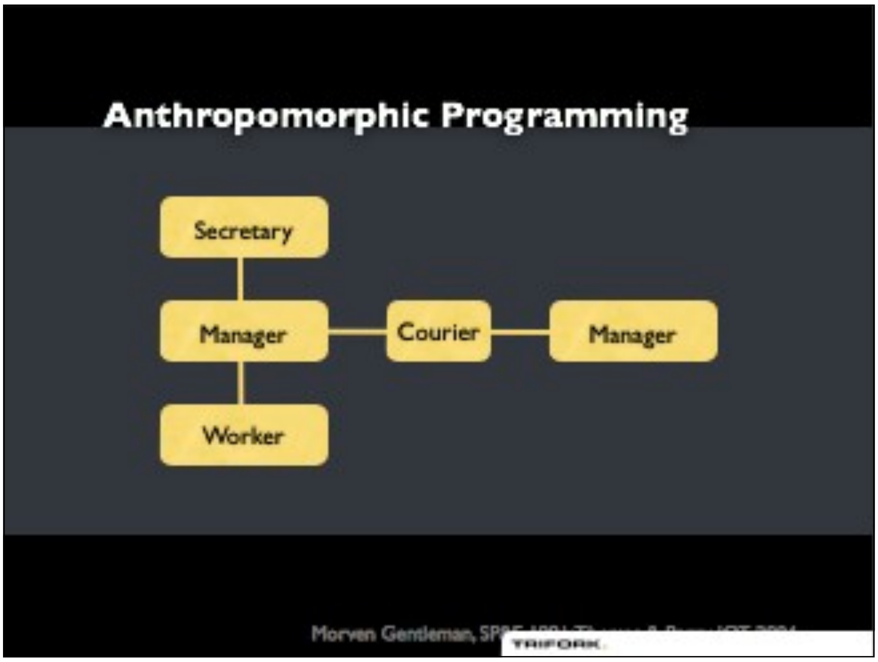
TRIFORK


1UP
120
TRIFORK

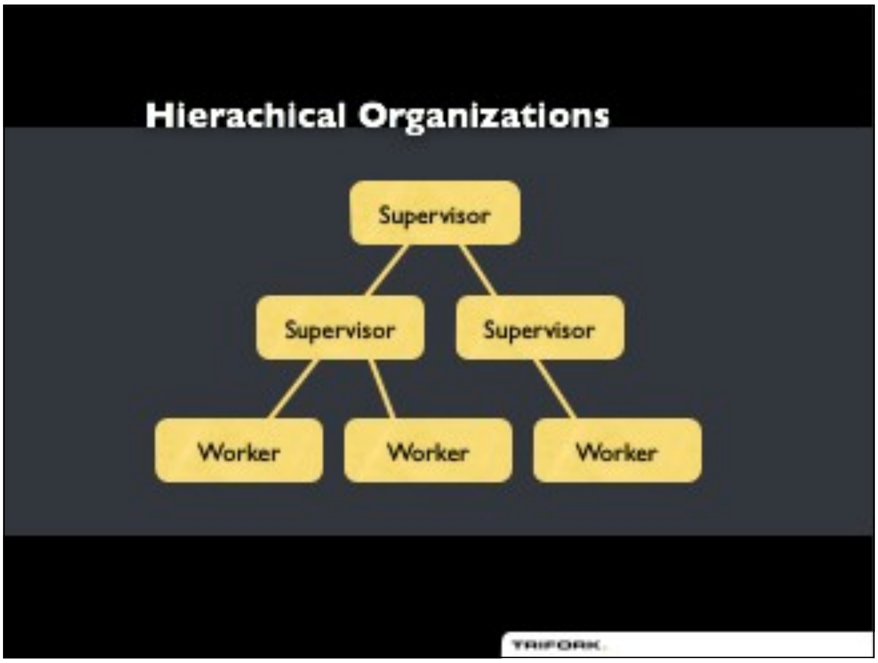

TRIFORK


TRIFORK

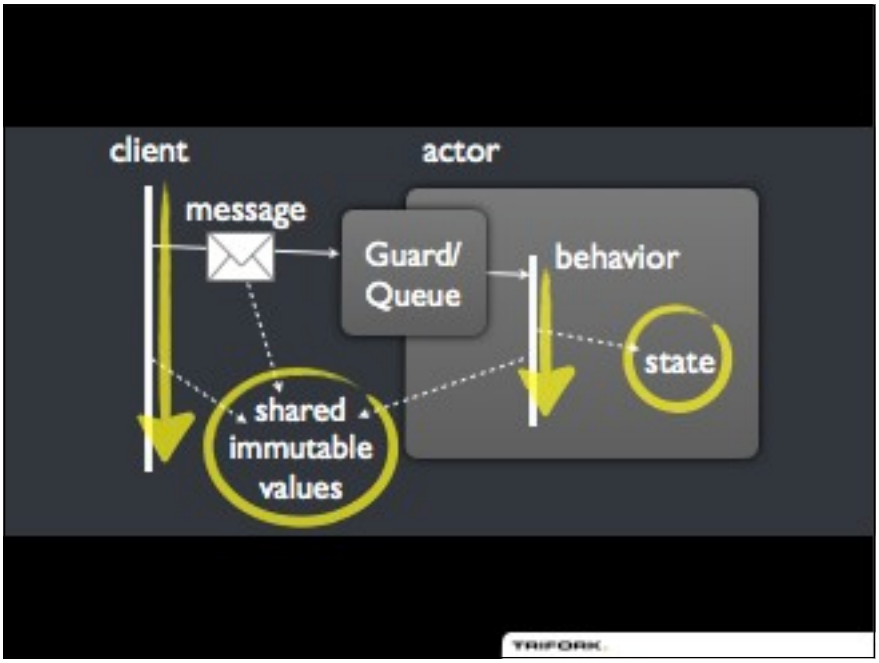


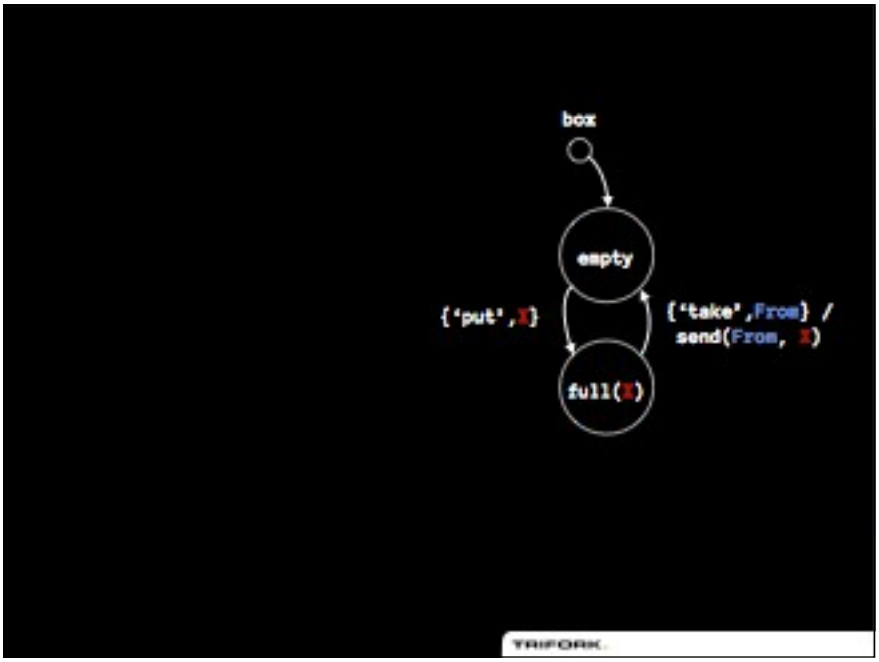


Anthropomorphic Programming
Secretary
Manager
Courier
Manager
Worker
Morven Gentleman,

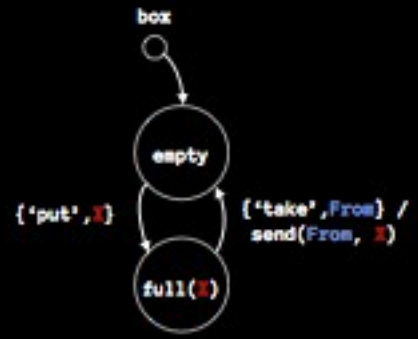


Hierachical Organizations
Supervisor
Supervisor
Supervisor
Worker
Worker
Worker

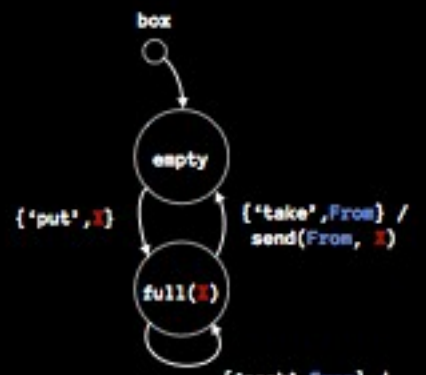





```
box() → empty().

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

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

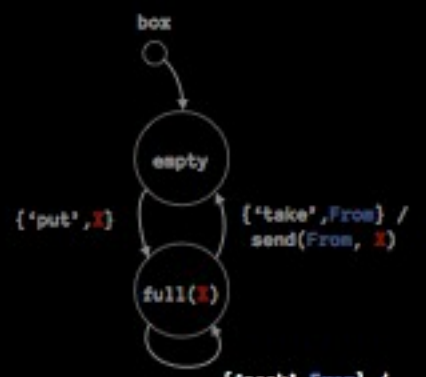


```
box() → empty().

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

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



TRIFORK

```
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.
```



TRIFORK

```
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
```

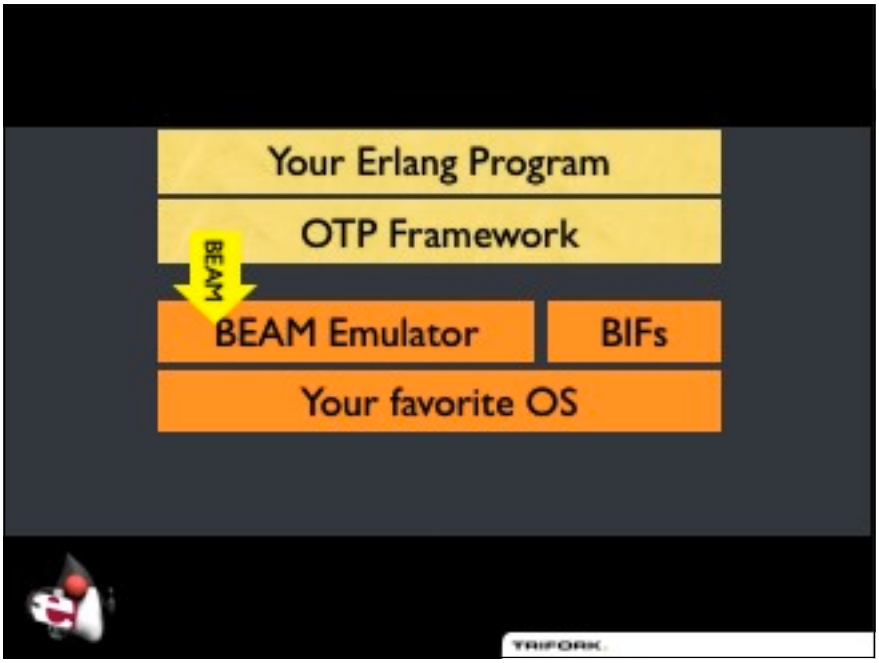
TRIFORK

```
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.
```

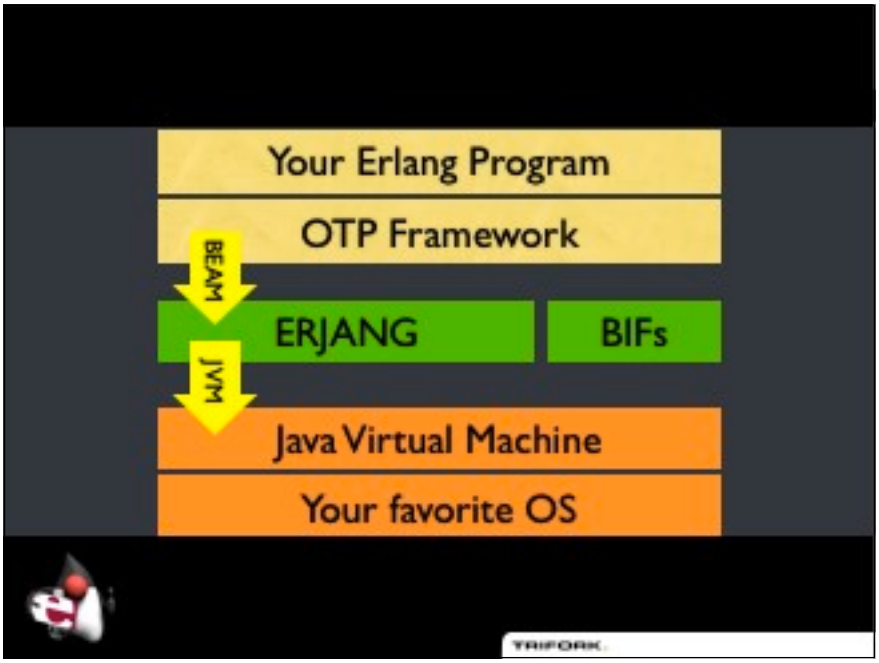
```
Box = spawn(box),

Box ! {'put', 27},

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

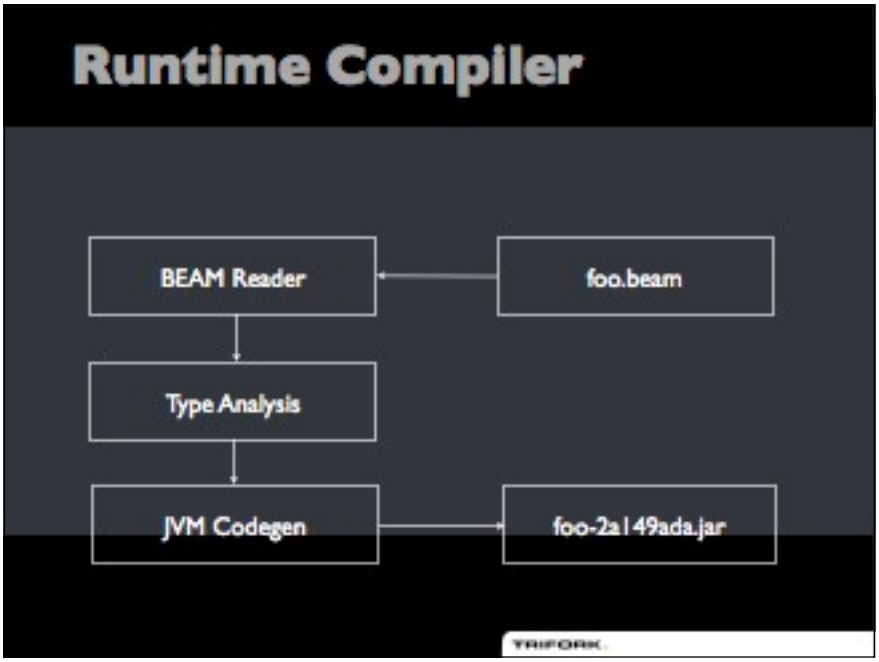
TRIFORK

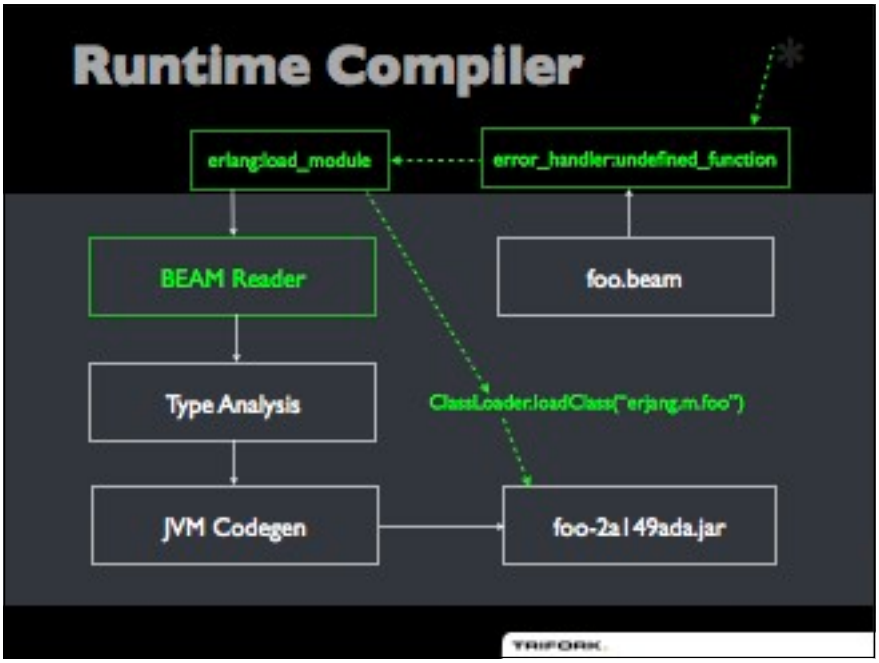
TRIFORK



TRIFORK


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


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


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

Language Concepts

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

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.Bodyc_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).
```

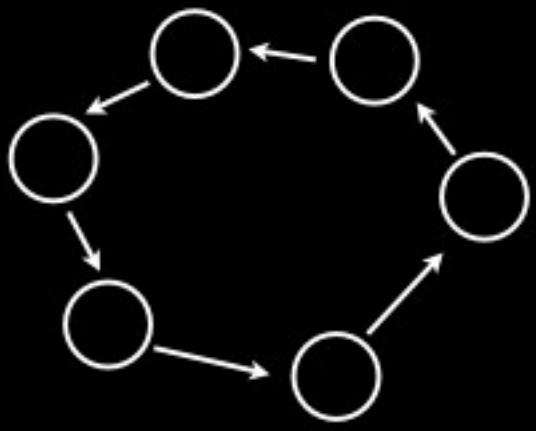

```
foo(H, T) ->
  lists:reverse(H ++ T).
```

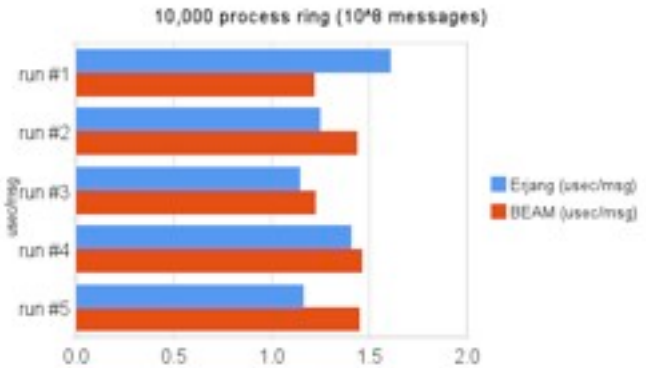
```
public static EObject
    foo__2(EProc p, EObject H, EObject T)
{
    EObject r = foo__2$body(p,H,T);
    while (r == TAIL_MARKER) {
       r = p.tail.go();
    }
    return r;
}
```

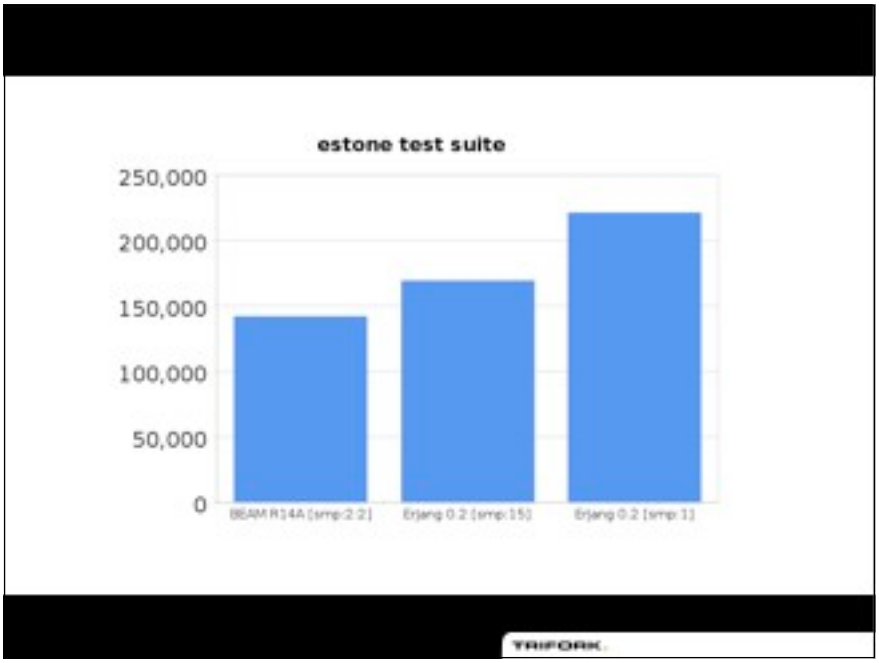
```
foo(H, T) ->
  lists:reverse(H ++ T).
```

```
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;
}
```

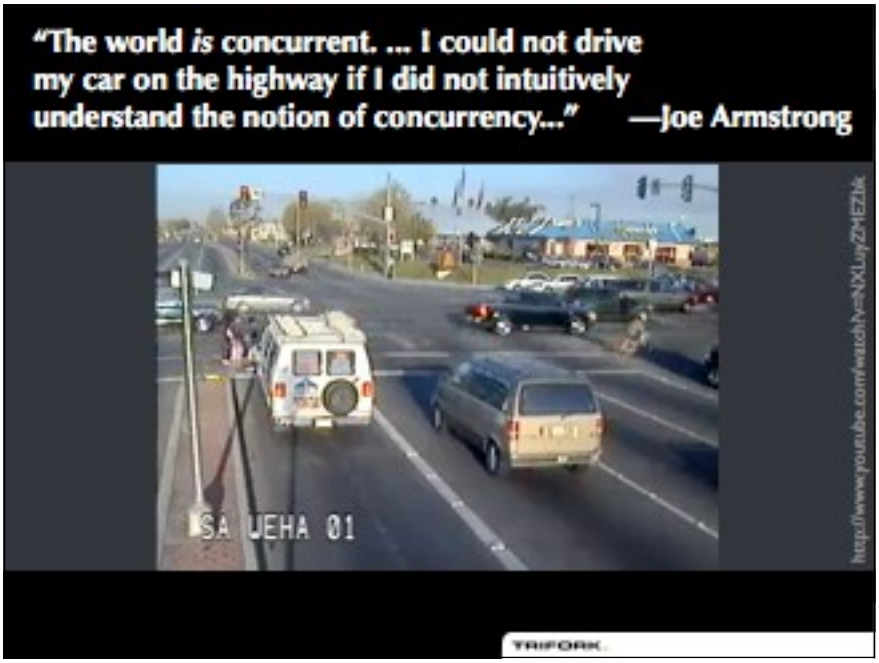
The ring!

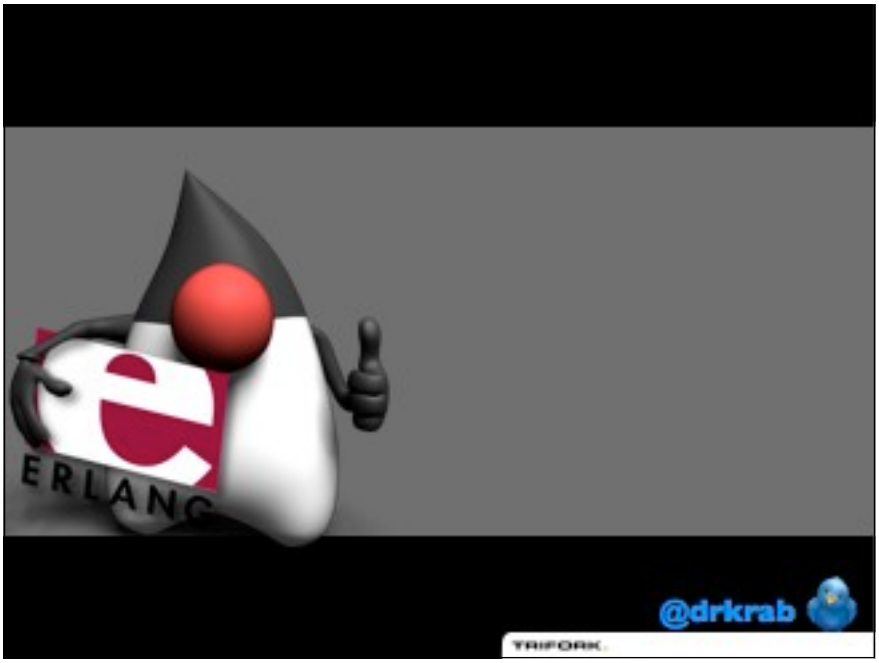





estone test suite
250,000
200,000
150,000
100,000
50,000
0
BEAM R14A (smp:2 2)
Erjang 0.2 (smp:15)
Erjang 0.2 (smp:1)
TRIFORK


Programming Erlang
Software for a Concurrent World
Joe Armstrong
Erjang, Why
JAVA TO THE LIMIT


"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
SA UEHA 01
http://www.youtube.com/watch?v=YXLuyZMEZbk
TRIFORK


ERLANG
@drkrab
TRIFORK