ERJANG

Inside Erlang on the Java Virtual Machine

Kresten Krab Thorup
Hacker, etc.
@drkrab

EXACTLY FOUR YEARS AGO…

Hacker News new | threads | comments | ask | jobs | submit

▲ Erjang is a virtual machine for Erlang, which runs on Java (github.com)

31 points by fogus 1452 days ago | comments

▲ vladev 1452 days ago | link

Call me a skeptic, but this project ditches Erlang's awesome VM and takes only the clunky language syntax and puts it on the

I wonder how things like process isolation will work out.

▲ sriramk 1452 days ago | link

You beat me to it. My first reaction was "Wait - how is preserving Erlang's syntax but losing the VM a good thing?". I'd l
the other direction - a more modern syntax built on Erlang's rock-solid VM + OTP libraries.

▲ oconnor0 1452 days ago | link

What are the names of some of those projects?

▲ frig8 1452 days ago | link

Reia http://wiki.reia-lang.org/wiki/Reia_Programming_Language

Lisp Flavored Erlang http://github.com/rvirding/lfe

▲ bad_user 1452 days ago | link

I don't see a problem with process isolation ... you can control the bytecode generated from the compiler, and then you
really hard to please.

Since Erlang uses light-weight processes, this is exactly what the Erlang VM is doing.

▲ xtho 1452 days ago | link

I'm always skeptic if there is only one actively maintained implementation of a language in use. In this respect, this is p

HIGHLIGHTS

- Runs Erlang R16B02
- Based on Java 7
- JIT: BEAM → JVM
- Java Integration
- ets, inet, mnesia, compiler, shell, distribution, …
- NIF support
- Full language semantics

Your Erlang Program

OTP Framework

BEAM

BEAM Emulator

BIFs

Your favorite OS

Your Erlang Program

OTP Framework

BEAM

ERJANG

BIFs

JVM

Java Virtual Machine

Your favorite OS

JAVA / JVM "PROS"

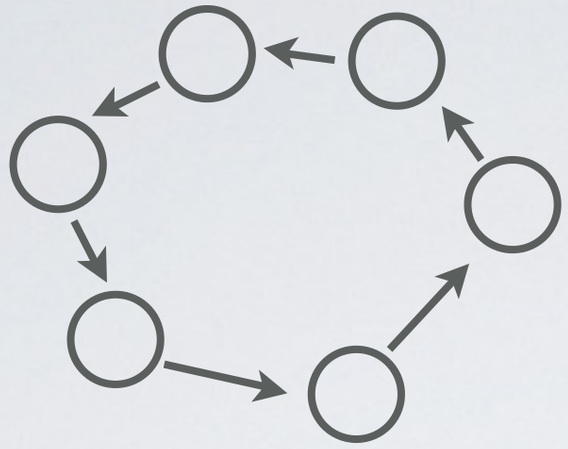
- "Socially acceptable"
- Integration: Large set of 3rd party libraries
- Fast: Inter-module inlining, garbage collection

JAVA / JVM "CONS"

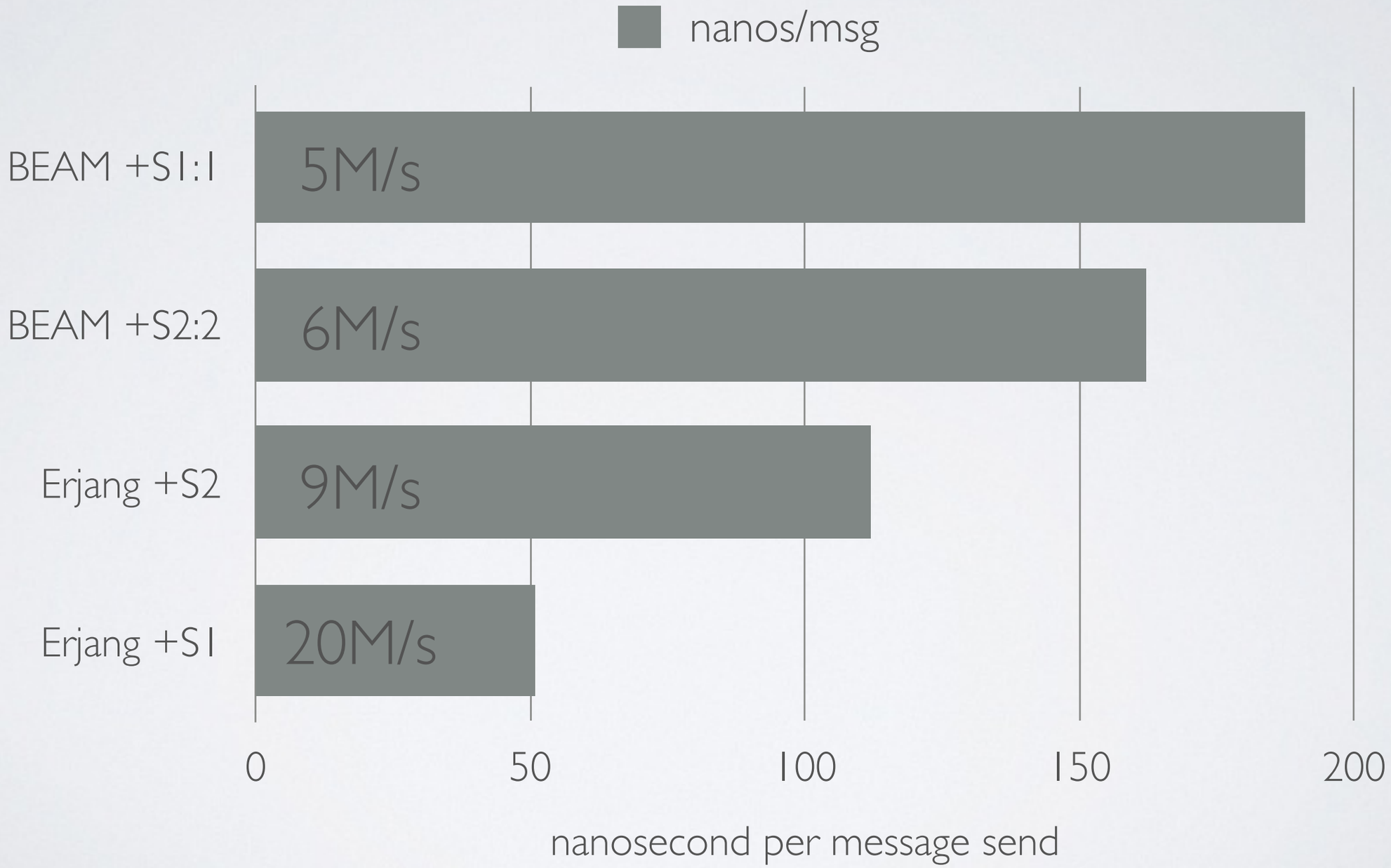
- Challenging to implement Erlang in it…
  - All code must pass a load-time type checker (byte code verification)
  - Functions are limited to 64k byte code size
  - It's not easy to encode Erlang's process model
- Garbage collection is global

list manipulation	371,0%
small messages	40,4%
medium messages	74,7%
huge messages	803,8%
pattern matching	333,2%
traverse	235,5%
Work with large dataset	242,8%
Work with large local database	316,3%
Alloc and dealloc	885,9%
BIF dispatch	1397,5%
Binary handling	78,3%
ets datadictionary	150,8%
Generic server (with timeout)	(3,8%)
Small Integer arithmetics	242,8%
Float arithmetics	6741,4%
Function calls	81,7%
Timers	113,7%
Links	190,7%

RING



10,000 size ring



RING: 100.000 PROCESSES

```
krab$ erl
Erlang R16B02 (erts-5.10.3) [source] [64-bit] [smp:1:1]
[async-threads:10] [kernel-poll:false]

Eshell V5.10.3  (abort with ^G)
1> chain:main().
100000 iterations in 0.86591 seconds (8.6591 µs/iter)
ok
2> chain:main().
100000 iterations in 0.850337 seconds (8.50337 µs/iter)
ok
3> chain:main().
100000 iterations in 0.839092 seconds (8.39092 µs/iter)
ok
4> chain:main().
100000 iterations in 0.717872 seconds (7.17872 µs/iter)
ok
5> chain:main().
100000 iterations in 0.736076 seconds (7.36076 µs/iter)
```

RING: 100.000 PROCESSES

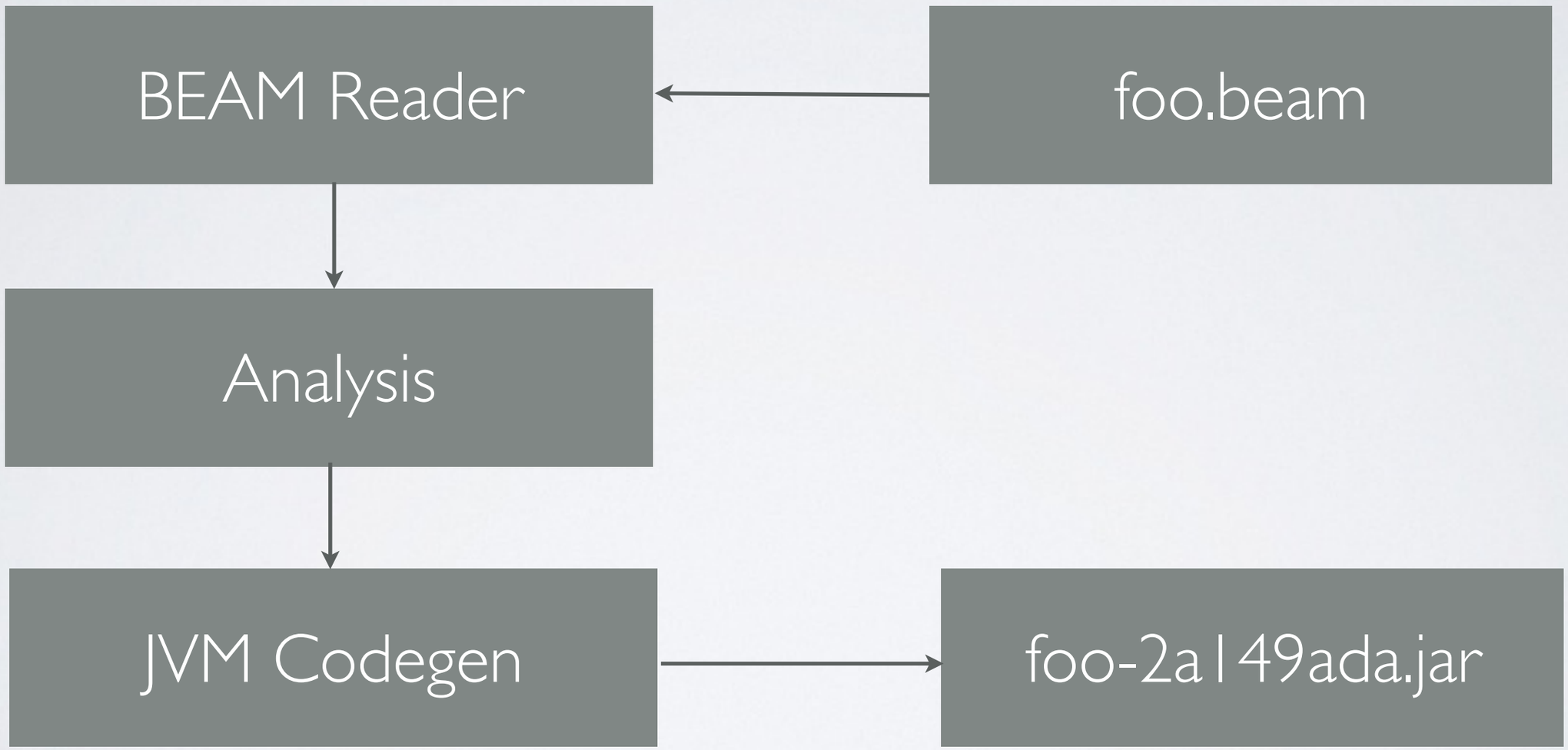
```
krab$ jerl
** Erjang R16B02 **  [root:/Users/krab/erlang/r16b02]
[erts:5.10.3] [smp S:1 A:10] [java:1.7.0_40] [unicode]

Eshell V5.10.3  (abort with ^G)
1> chain:main().
100000 iterations in 1.264609 seconds (12.64609 µs/iter)
ok
2> chain:main().
100000 iterations in 0.76343 seconds (7.6343 µs/iter)
ok
3> chain:main().
100000 iterations in 0.426665 seconds (4.26665 µs/iter)
ok
4> chain:main().
100000 iterations in 0.381436 seconds (3.81436 µs/iter)
ok
5> chain:main().
100000 iterations in 0.287675 seconds (2.87675 µs/iter)
```

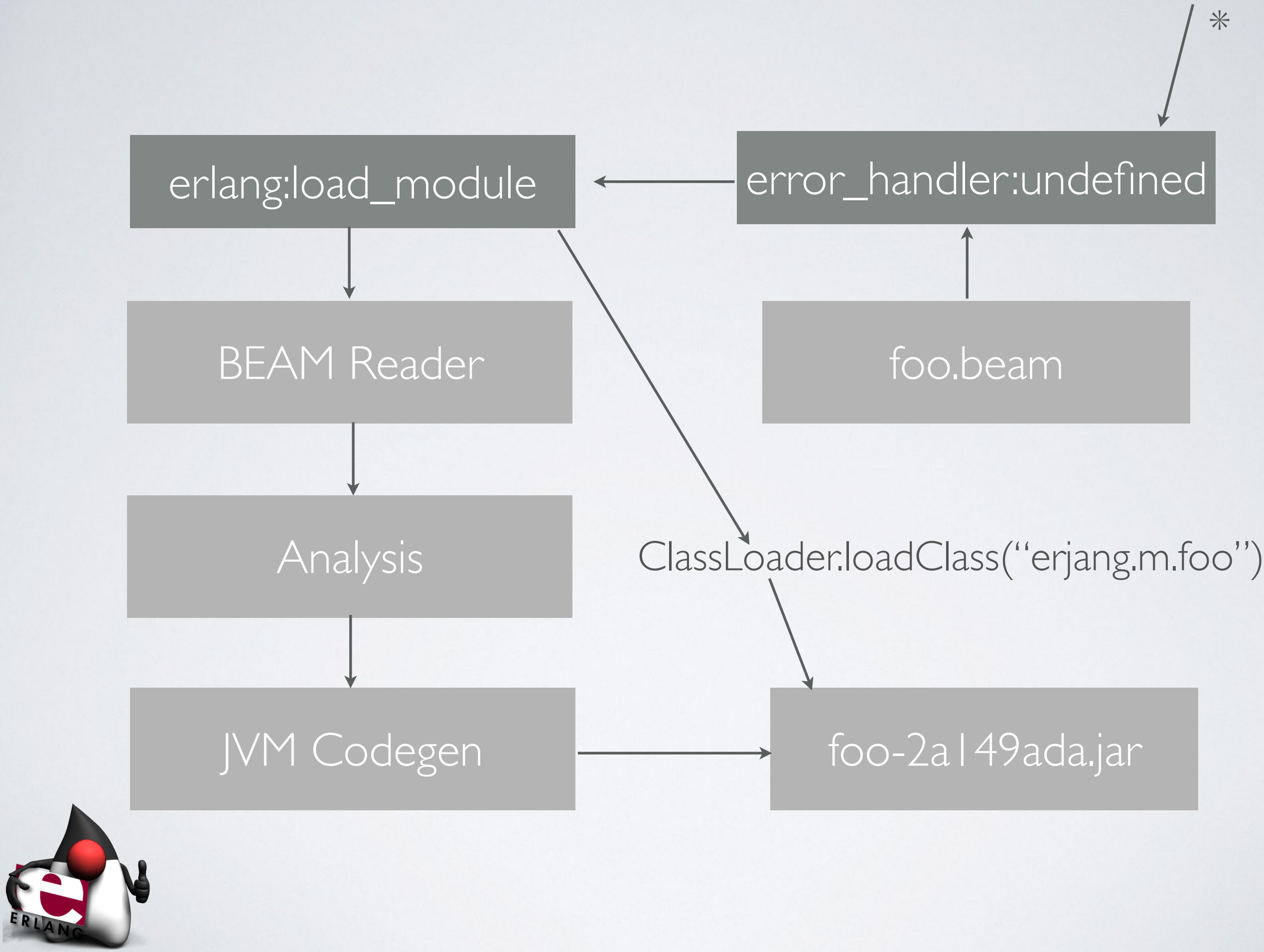
ERJANG'S JIT

- Load BEAM file
- Flow analysis
- Type inference
- Code generation

- Leverage Java's Performance
- Tail calls
- Pausable calls
- Special cases…

JIT COMPILER




*
erlang:load_module
error_handler:undefined
BEAM Reader
foo.beam
Analysis
ClassLoader.loadClass(“erjang.m.foo”)
JVM Codegen
foo-2a149ada.jar

CODE CONCEPTS

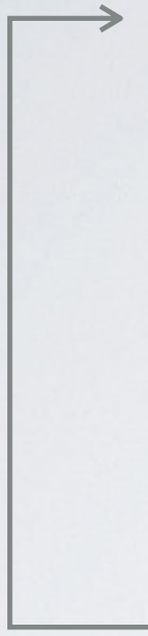
Erlang	Erjang
Process + Messaging	Coroutine + Mailbox [Kilim]
Tail Calls	Trampoline Encoding
State Encapsulation	Immutable / Persistent Data

ERLANG ⇒ JVM

```
-module(bar).

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

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

ERLANG ⇒ JVM

```
-module(bar).

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

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



```
{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, bar},{atom,bat},2}.
```

```
public static EObject bat___2(EProc eproc, EObject arg1, EObject arg2)
{
    ECons cons; ENil nil;
loop: do {
      if((cons = arg1.testNonemptyList()) != null) {
         // extract list
         EObject hd = cons.head();
         EObject tl = cons.tail();
         // call foo/2
         EObject tmp = foo__2$call(eproc, hd, arg2);
         // self-tail recursion
         arg1 = tl;
         arg2 = tmp;
         continue tail;
      } else if ((nil = arg1.testNil()) != null) {
         return arg2;
      }
   } while (false);
   throw ERT.func_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$tail(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;
}
```

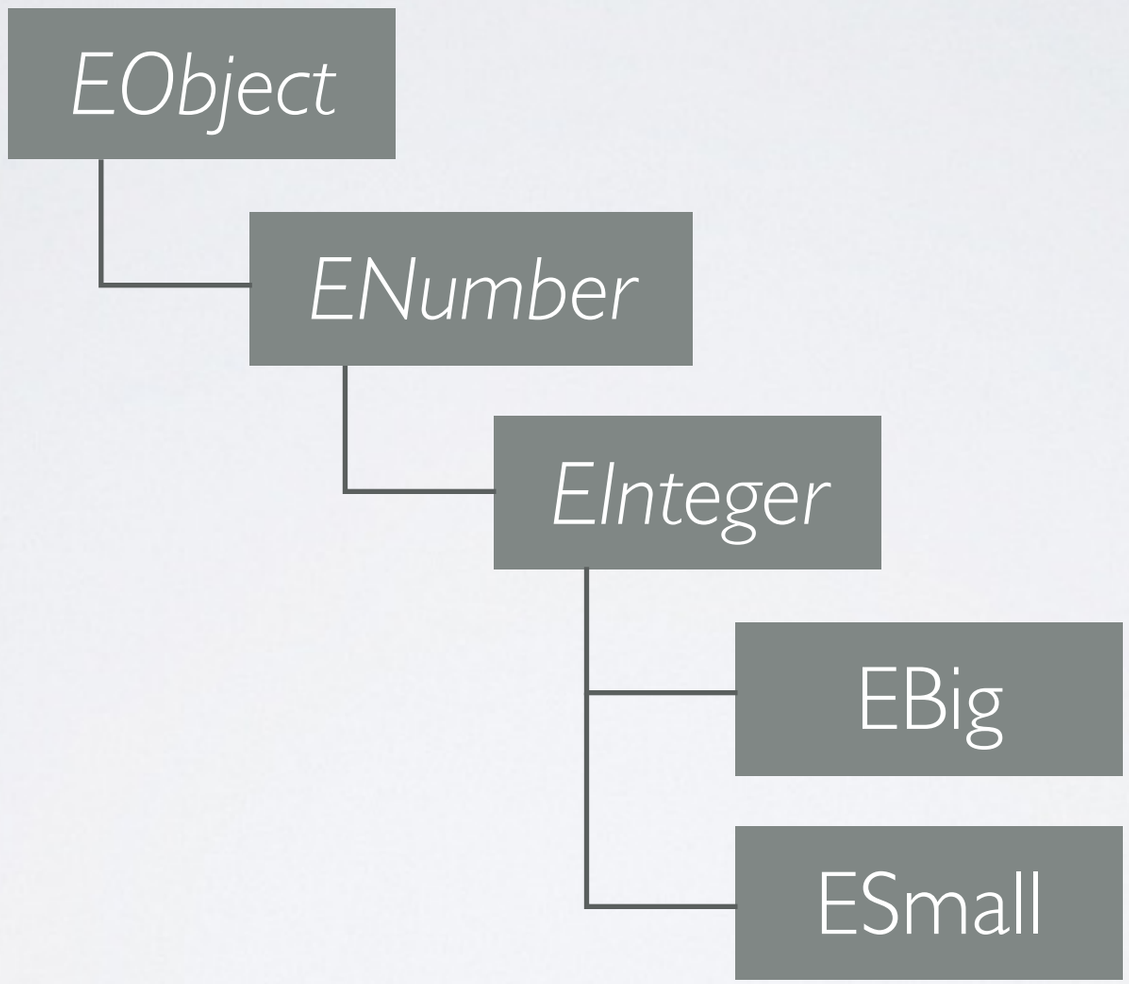
```
                                foo(H, T) ->
                                   lists:reverse(H ++ T).
```

```
public static EObject
    foo__2$call(EProc p, EObject h, EObject t)
{
    EObject r = foo__2$tail(p,h,t);
    while (r == TAIL_MARKER) {
        r = p.tail.go(p);
    }
    return r;
}
```

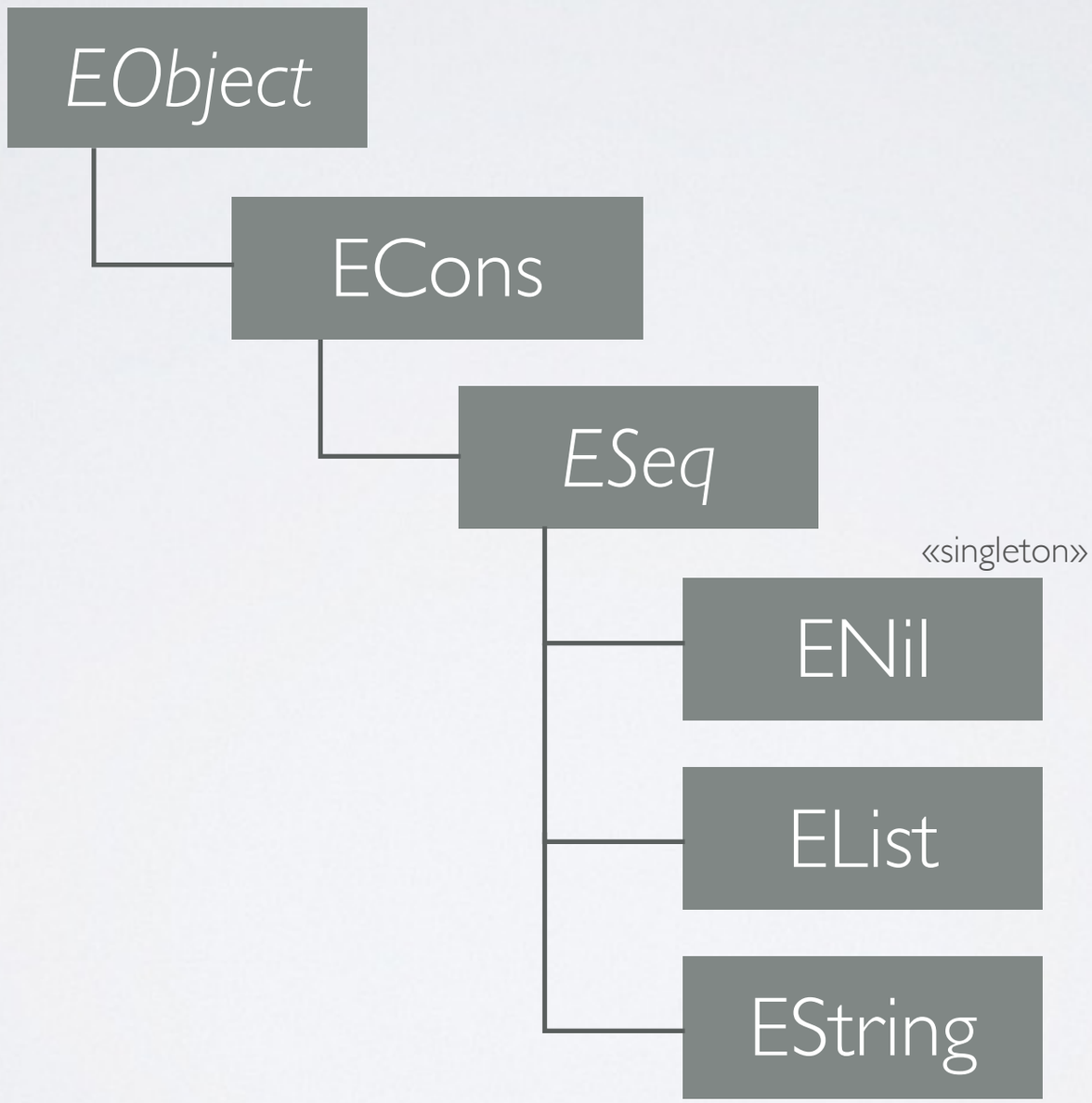
```
foo(H, T) ->
   lists:reverse(H ++ T).
```

```
package erjang.m.bar;
class bar extends ECompiledModule {

  @Import(module="lists", fun="reverse", arity=1)
  static EFun1 lists__reverse__1 = null;

  @Import(module="erlang", fun="++", arity=2)
  static EFun2 erlang__append__2 = null;

  ...

}
```

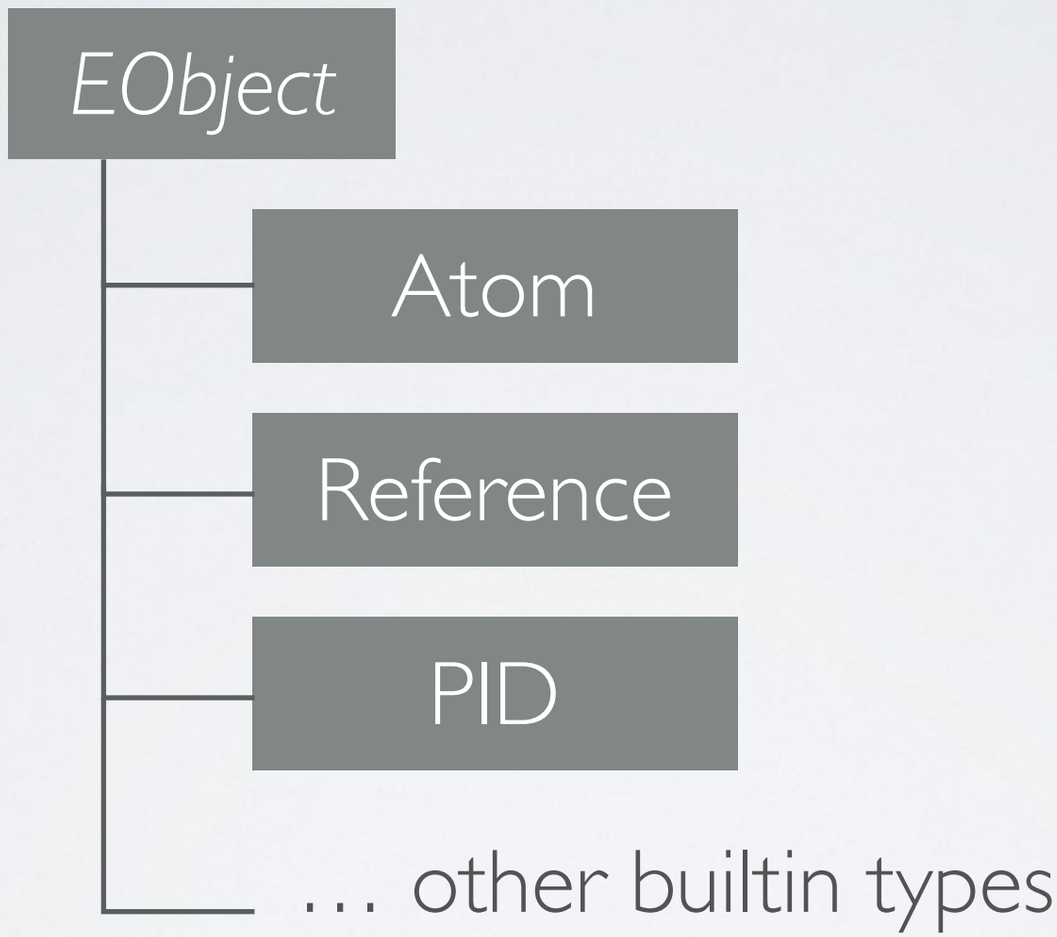
ERJANG TYPES



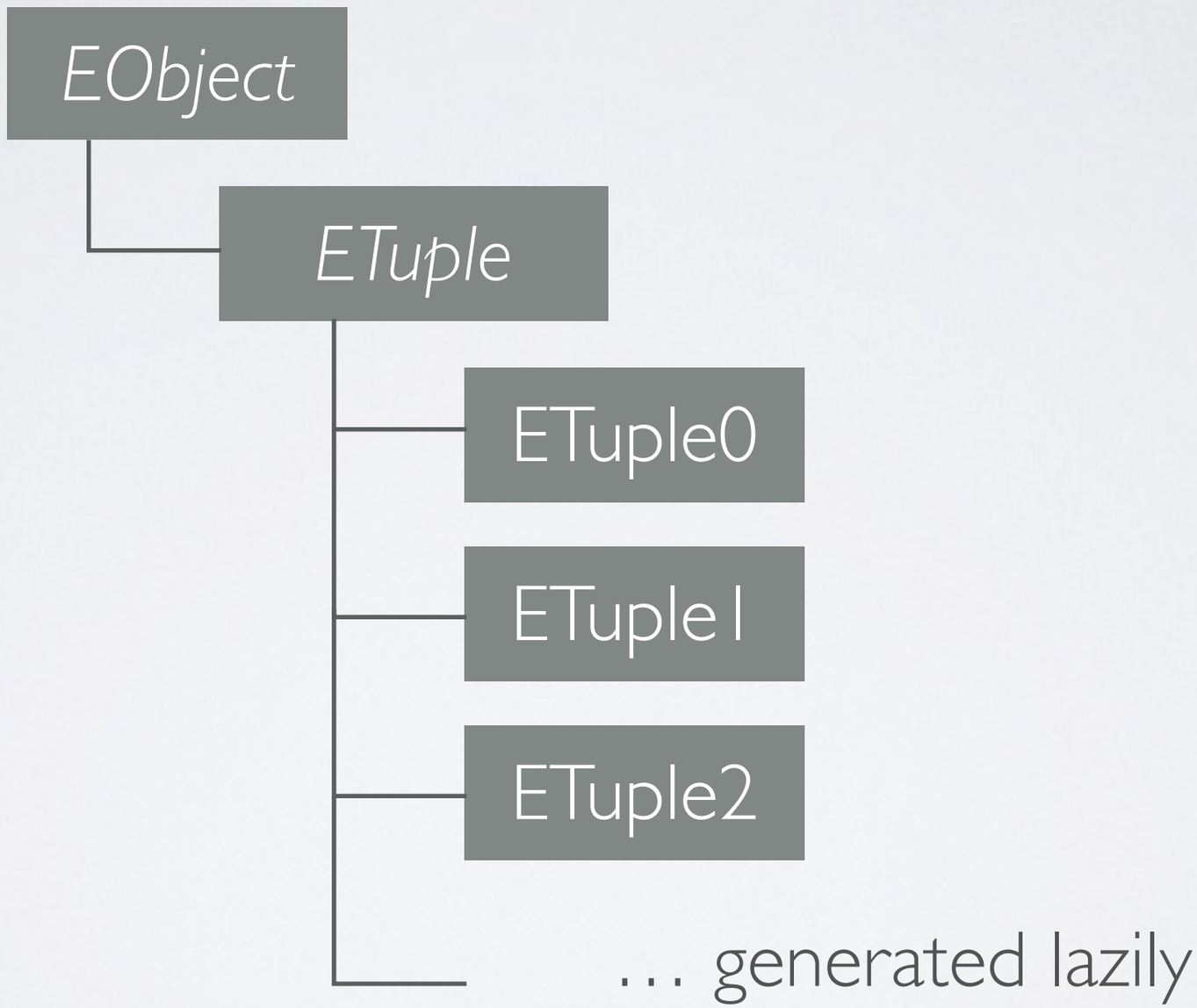
ERJANG TYPES



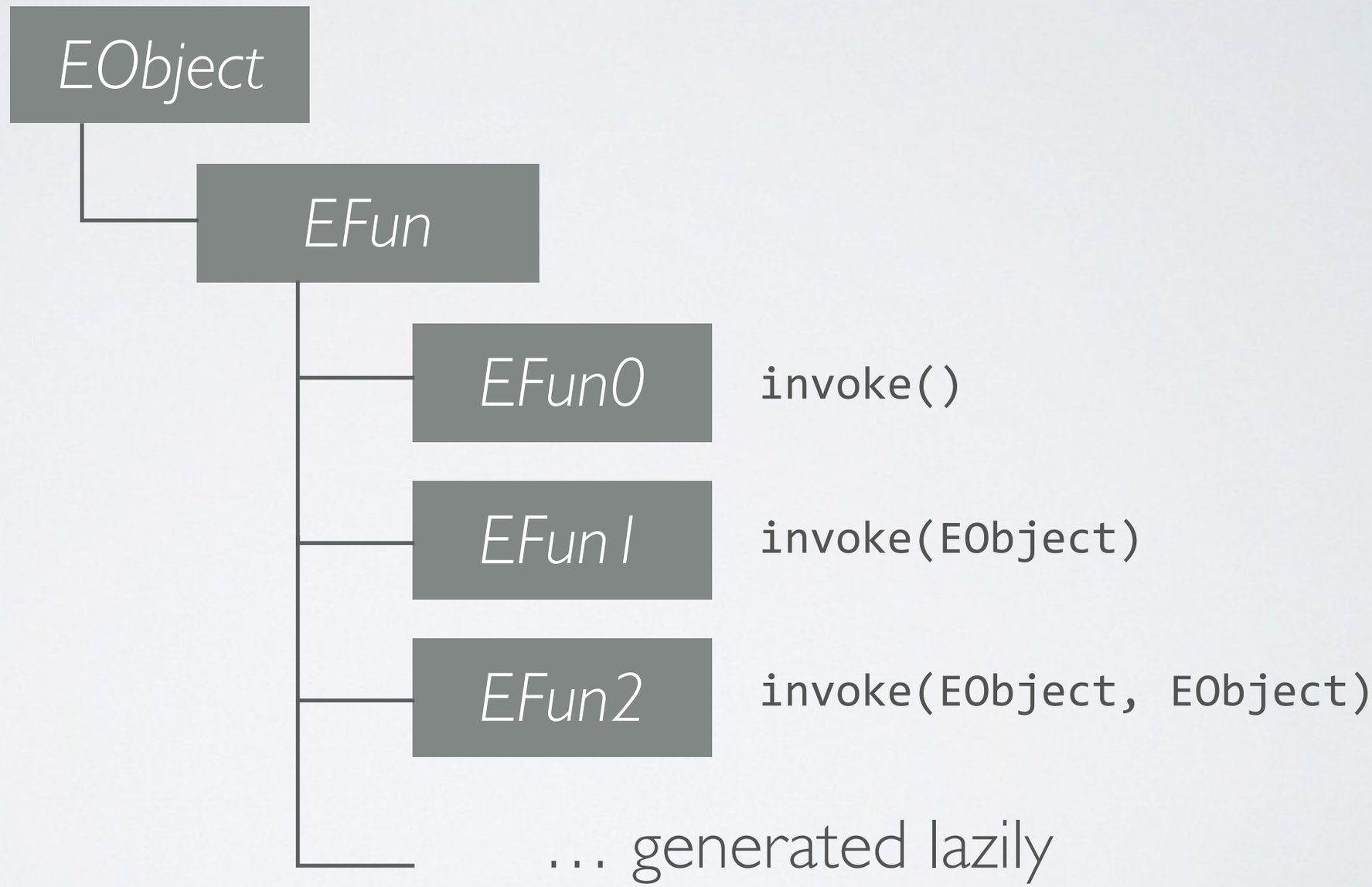
ERJANG TYPES



ERJANG TYPES



ERJANG TYPES



```
package erjang.m.bar;
class bar extends ECompiledModule {

  @Export(module="bar", fun="foo", arity=2)
  static EFun2 foo__2 = new Fun2() {

   EObject invoke(EObject h, t) {
      return foo$call(h,t);
   }

   EObject go(EProc p) {
    return foo$tail(p.arg1, p.arg2);
   }
  }

  ...
}
```

```
package erjang.m.bar;
class bar extends ECompiledModule {

  @Export(module="bar", fun="foo", arity=2)
  static EFun2 foo__2 = new Fun2() {

   EObject invoke(EObject h, t) {
      return foo$call(h,t);
   }

   EObject go(EProc p) {
    return foo$tail(p.arg1, p.arg2);
   }
  }

  ...
}
```

TYPE/FLOW ANALYSIS

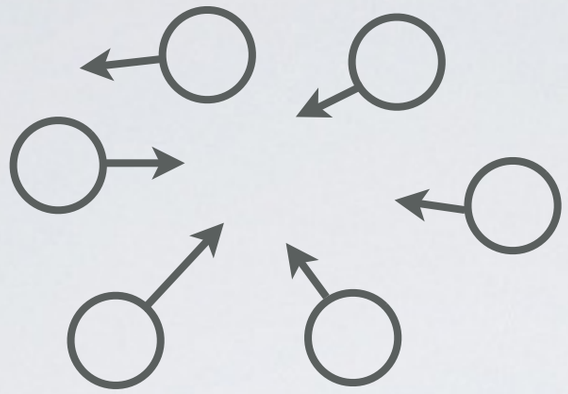
- Abstract evaluation (per module) for all code.
  - Type inference (BIFs can be overloaded)
  - Tailcall / "Suspendable" inference

```
package erjang.m.erlang;
class erlang extends ECompiledModule {


  @BIF(name="+")
  static ENumber plus(EObject o1, EObject o2) {
    ENumber num = o2.testNumber();
    if (num == null) throw ERT.badarg(o1, o2);
    return o1.plus(num);
  }

  @BIF(name="+")
  static double plus(EObject v1, double v2) {
    return v1.plus(v2);
  }

  @BIF(name="+")
  static double plus(double v1, EObject v2) {
    return v2.plus(v1);
  }


}
```

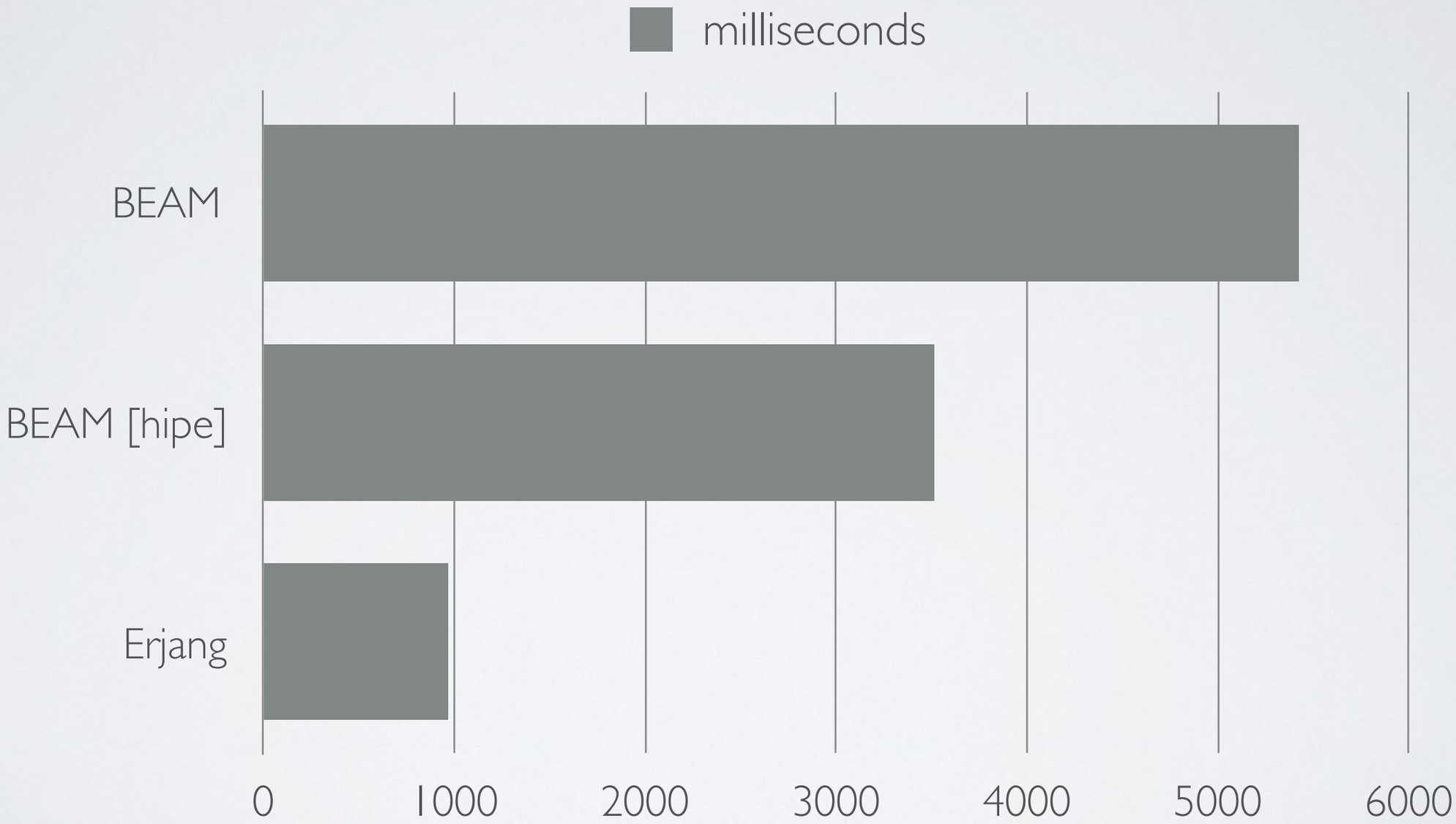
```
package erjang.m.erlang;
class EObject {

 ENumber plus(ENumber arg)  { throw badarg(this, arg); }
 double  plus(double arg)   { throw badarg(this, ERT.box(arg)); }
 ENumber plus(int arg)      { throw badarg(this, ERT.box(arg)); }
...
}

abstract class ENumber extends EObject { ... }
abstract class EInteger extends ENumber { ... }

class EDouble extends ENumber {
  double value;
  EDouble plus(ENumber arg) { return arg.plus(value); }
  double  plus(double arg)  { return value + arg; }
  EDouble plus(int arg)     { return new EDouble( value + arg );}
...
}
```

list manipulation	371,0%
small messages	40,4%
medium messages	74,7%
huge messages	803,8%
pattern matching	333,2%
traverse	235,5%
Work with large dataset	242,8%
Work with large local database	316,3%
Alloc and dealloc	885,9%
BIF dispatch	1397,5%
Binary handling	78,3%
ets datadictionary	150,8%
Generic server (with timeout)	-3,8%
Small Integer arithmetics	242,8%
Float arithmetics	**6741,4%**
Function calls	81,7%
Timers	113,7%
Links	190,7%

NBODY



1.000.000 iterations

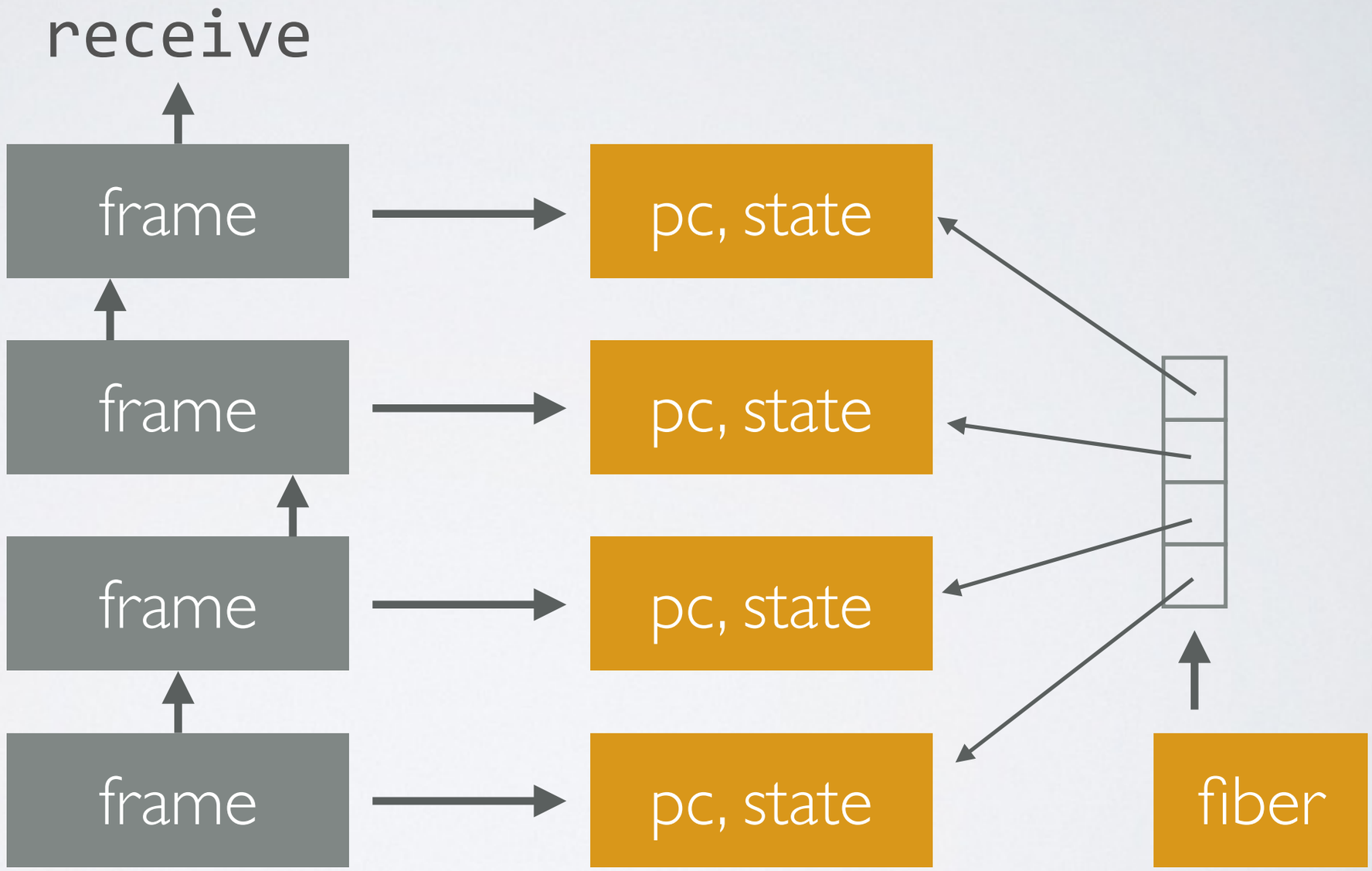


ENCODING PROCESSES

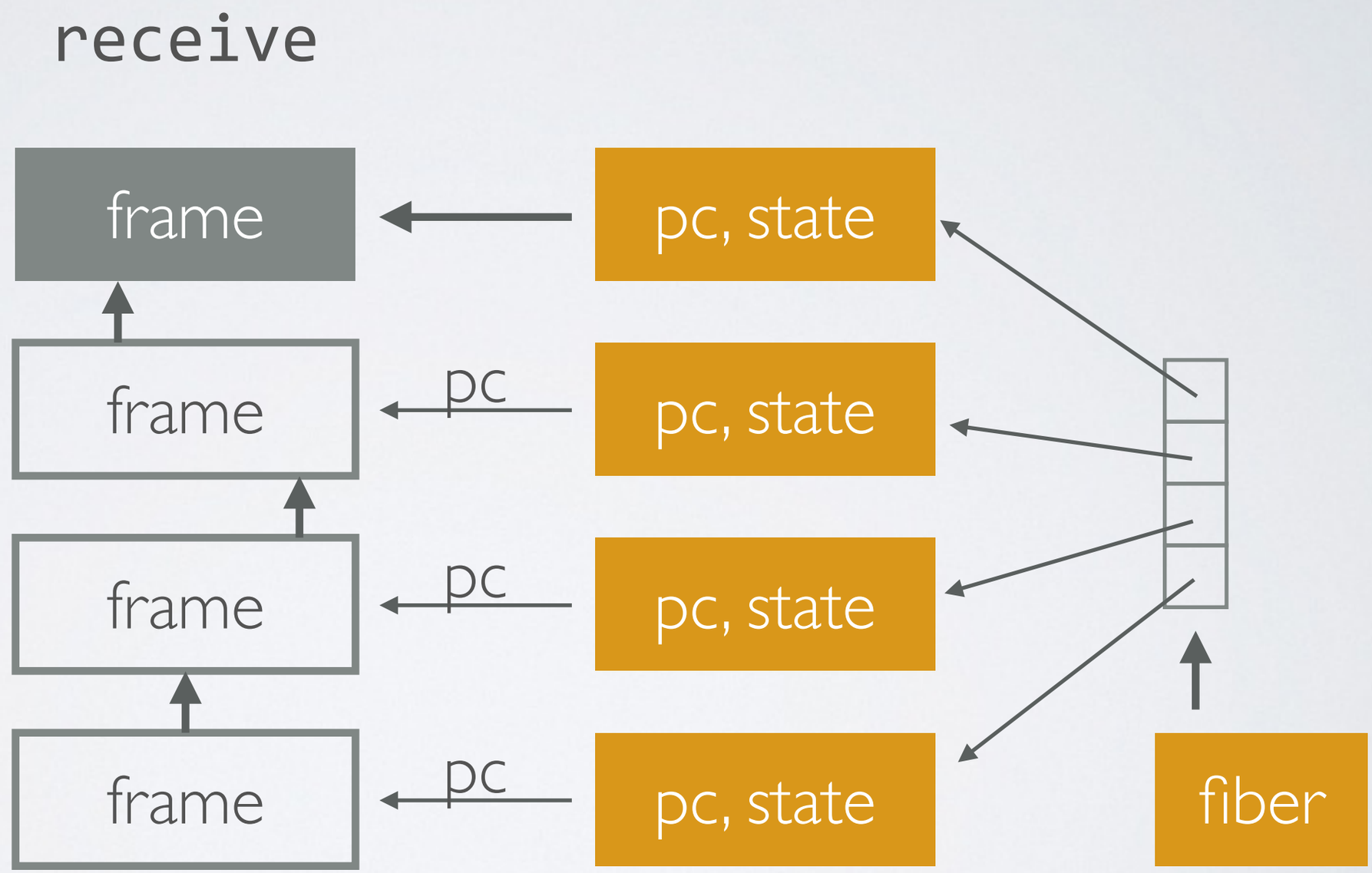
- Java doesn't have light weight threads, coroutines, or something else we can use

- Erjang encodes processes as cooperative coroutines (+ reduction counter)

- Leveraging a 3rd party library `Kilim`


receive
frame
frame
frame
frame
pc, state
pc, state
pc, state
pc, state
fiber

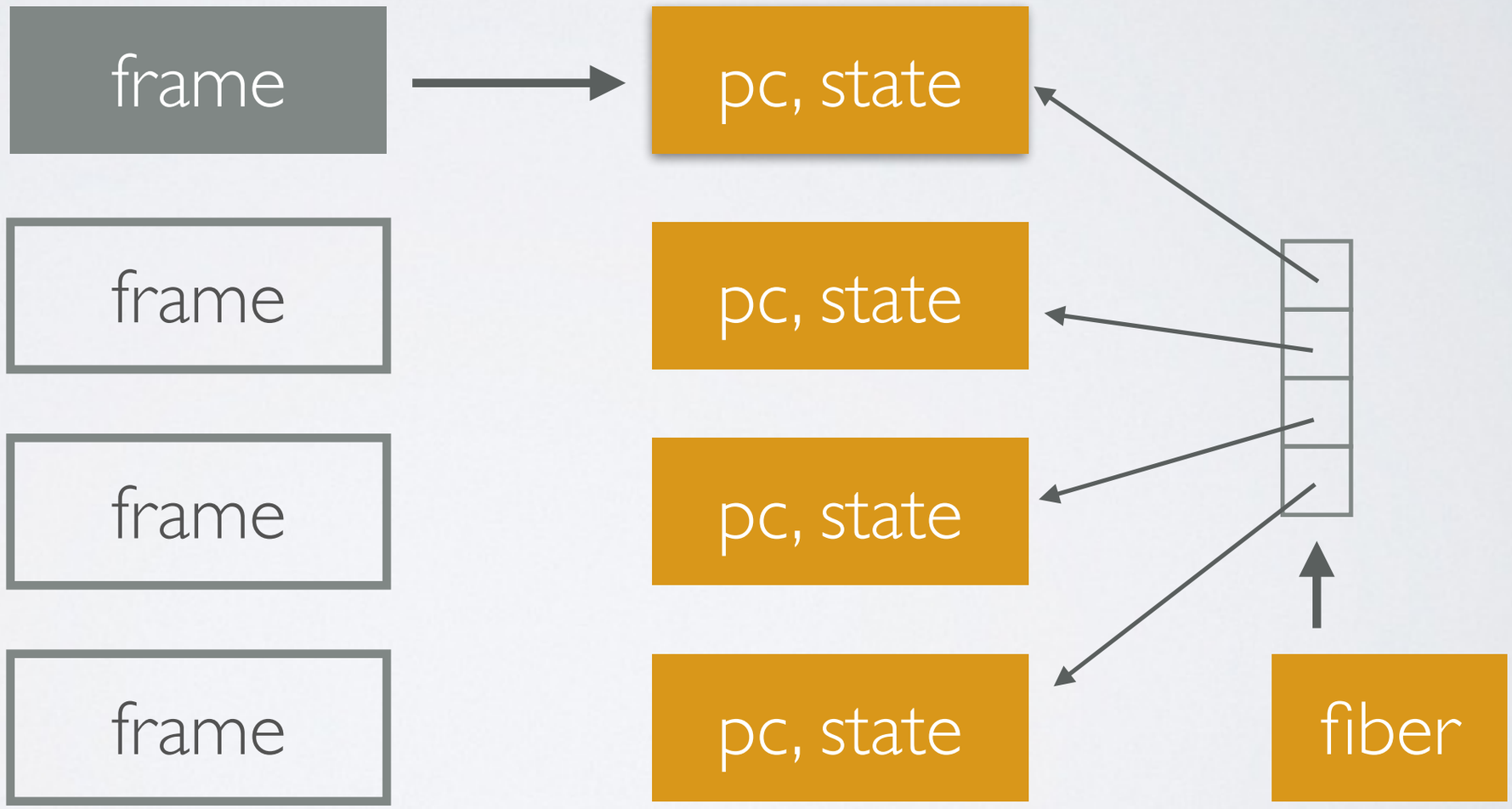

ERLANG


receive
frame
pc, state
frame
pc
pc, state
frame
pc
pc, state
frame
pc
pc, state
fiber


ERLANG

`receive`

suspending again…



`receive`

returning…

frame

pc, state

frame

pc, state

frame

pc, state

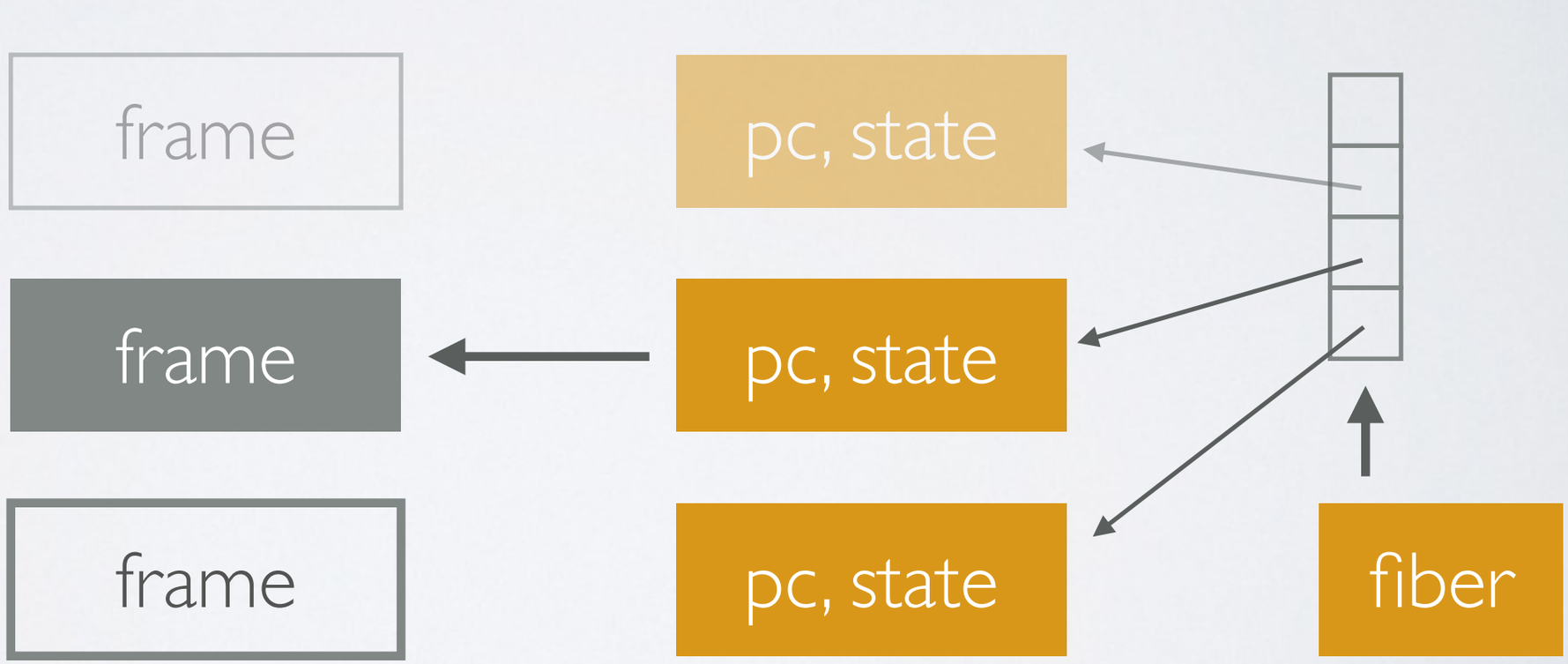
frame

pc, state

pc, state

fiber

`receive` returning…



```
EObject foo(..., Fiber fib) {
   case (fib.pc) {
   1: goto callsite_1;
   0: // fall thru
   }
  ...
  callsite_1:          // pausable call
  fib.down();
  res = bar(..., fib);
  case(fib.up()) {
     SUSPEND_SAVE:    «save local vars»
     SUSPEND_SKIP:    return null;
     RETURN_RESTORE: «restore local vars»
     RETURN_NORMAL:   // fall thru
  }
  ...
}
```

ENCODING PROCESSES

- Functions that can suspend are marked with a `Pausable` exception. This spreads like wildfire - or like an IO monad.
- Code bloat! Easily to 5x original code
- Not necessary for "leaf" functions tat can't suspend
- Intra-module flow analysis reduce this (as well as codegen for tail calls); likewise BIFs that are non-pausable don't cause code bloat in caller.

elixir

ELIXIR IS MY HOPE FOR ADOPTION

- Elixir is in many ways "A Better Ruby"
- Ruby community has good vibes from JRuby
- Elixir targets server-side components

MAKING ELIXIR RUN

- Fix name mangling (Erlang → Java)
- Had to reduce code/stack size (erlang's compiler)
- Elixir's test suite is very picky with error codes, stack traces, and boundary conditions.

RUNNING ELIXIR

- Testsuite at 98% (2287 of 2335 test cases)
- Needs some improvement in File System, Unicode, … nothing *essential* is missing or broken.
- Let's run some Elixir…

THANKS