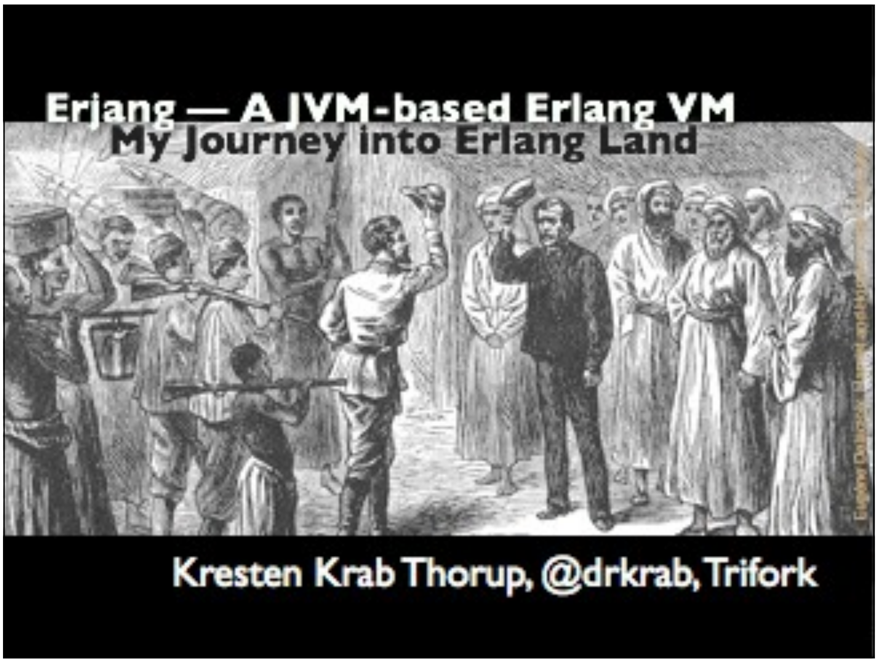


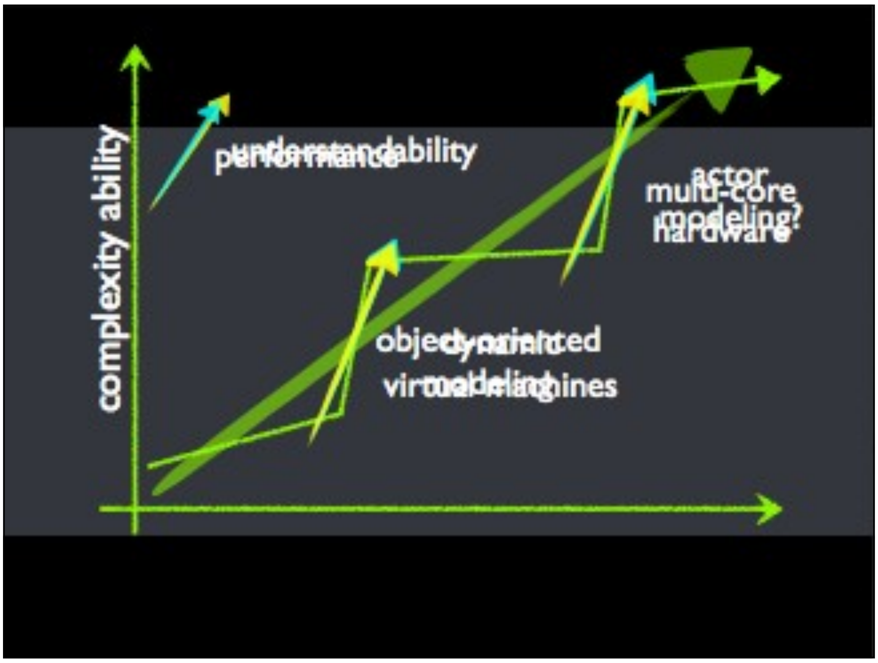
Erjang — A JVM-based Erlang VM

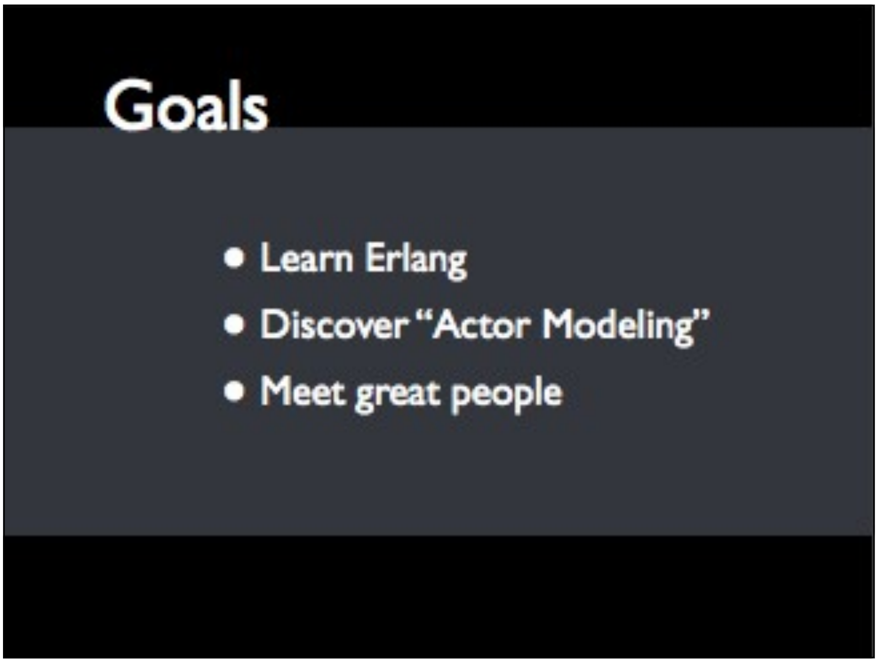
My Journey into Erlang Land

Kresten Krab Thorup, @drkrab, Trifork

The Desert of Java

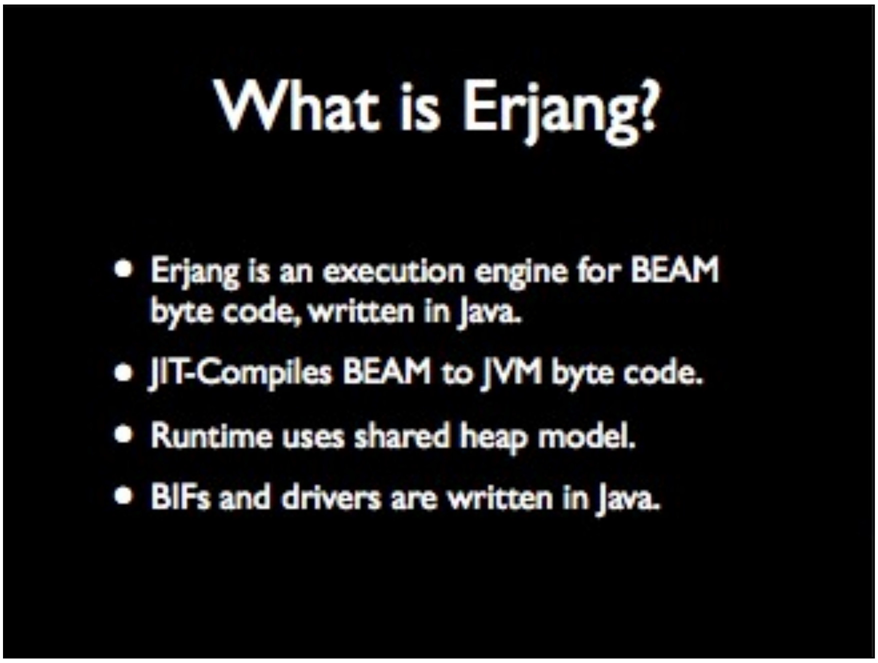
Java Threads
Brian Goetz
JAVA CONCURRENCY IN PRACTICE
Doug Lea
Concurrent Programming in Java Second Edition
THREADS & LOCKS AHEAD





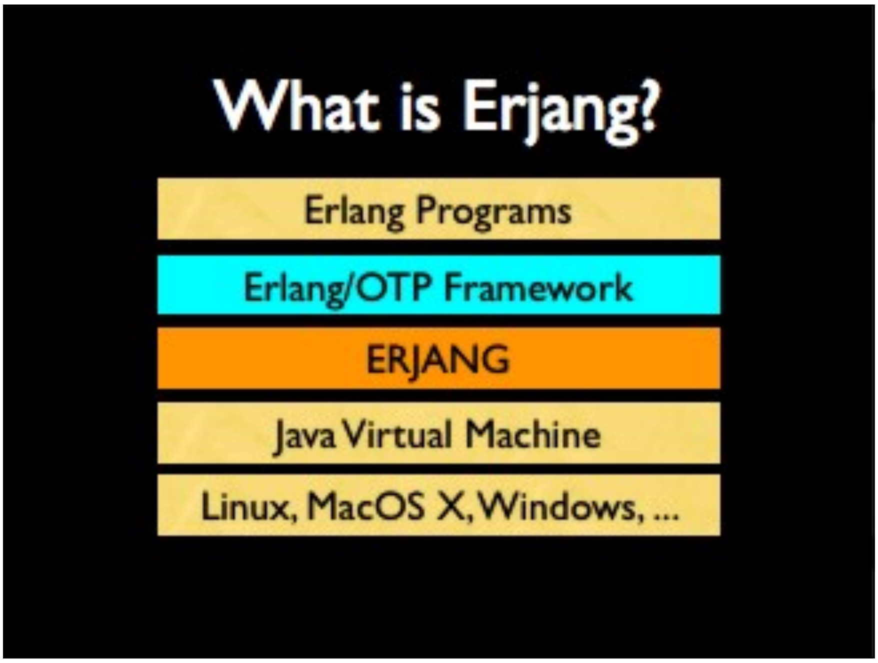
Goals

- Learn Erlang
- Discover "Actor Modeling"
- Meet great people



What is Erjang?

- Erjang is an execution engine for BEAM byte code, written in Java.
- JIT-Compiles BEAM to JVM byte code.
- Runtime uses shared heap model.
- BIFs and drivers are written in Java.



What is Erjang?

Erlang Programs

Erlang/OTP Framework

ERJANG

Java Virtual Machine

Linux, MacOS X, Windows, ...

DEMO

- Launching erjang
- basic shell

Erlang ≠ Erjang

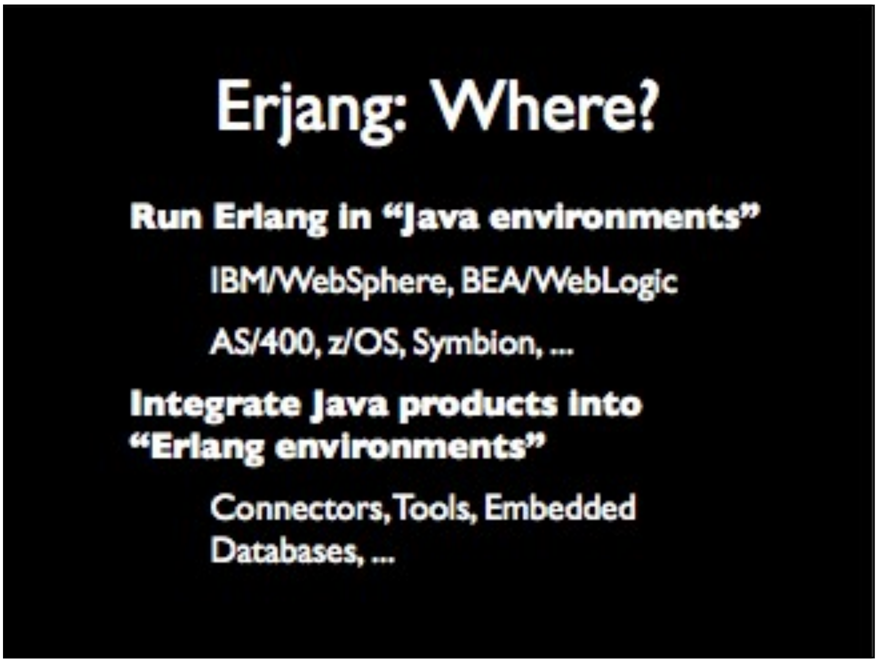
Shared heap

¬ Native drivers

¬ NIF

Why Java?

- JVM is everywhere (from mobile to AS/400)
- JVM has many libraries / integrations.
- JVM has 500+ man-years of engineering
- JVM is fast (for Java-ish programs).
- ... and I know JVM pretty well, ...



Erjang: Where?

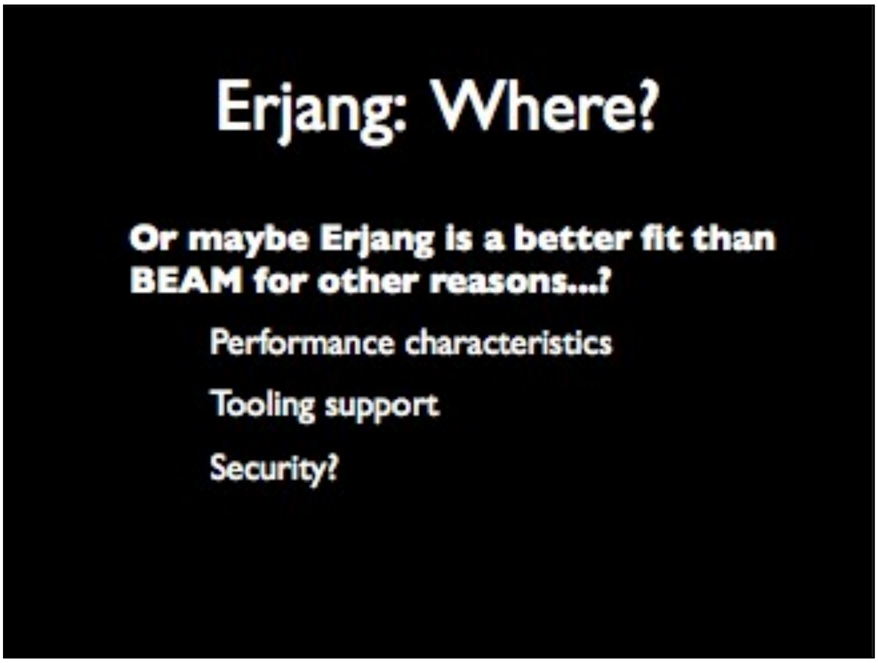
Run Erlang in "Java environments"

IBM/WebSphere, BEA/WebLogic

AS/400, z/OS, Symbion, ...

Integrate Java products into "Erlang environments"

Connectors, Tools, Embedded Databases, ...



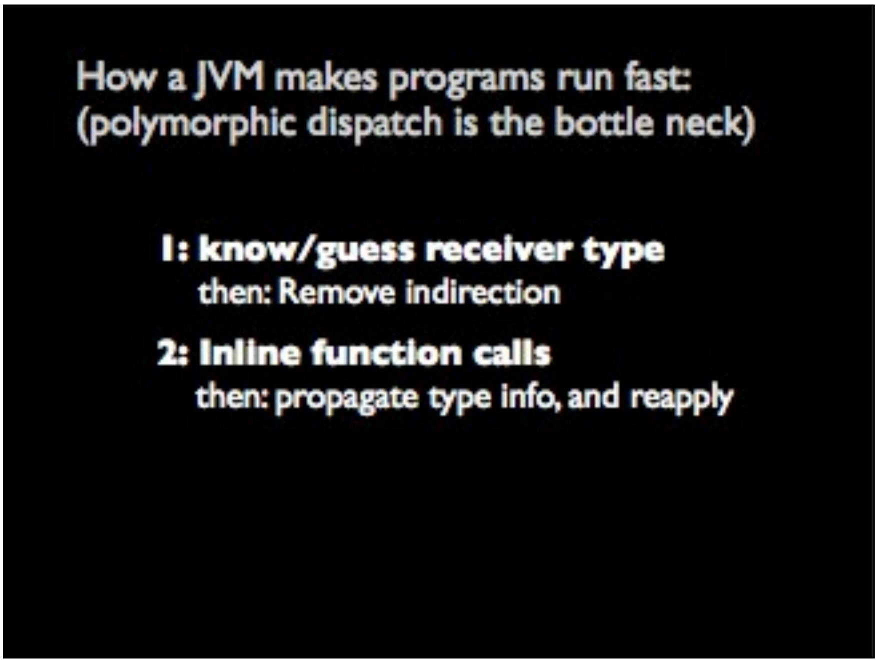
Erjang: Where?

Or maybe Erjang is a better fit than BEAM for other reasons...?

Performance characteristics

Tooling support

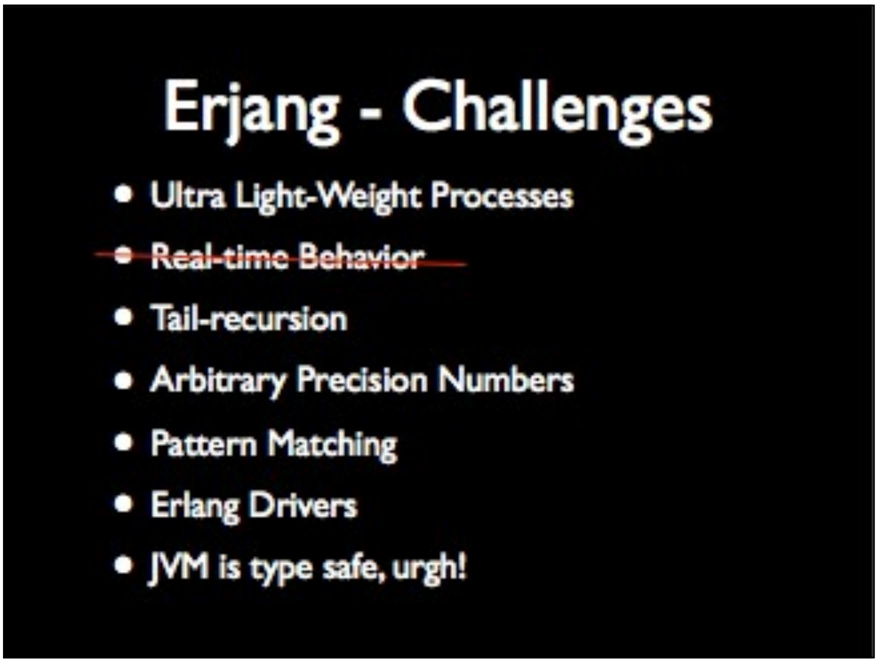
Security?



How a JVM makes programs run fast:
(polymorphic dispatch is the bottle neck)

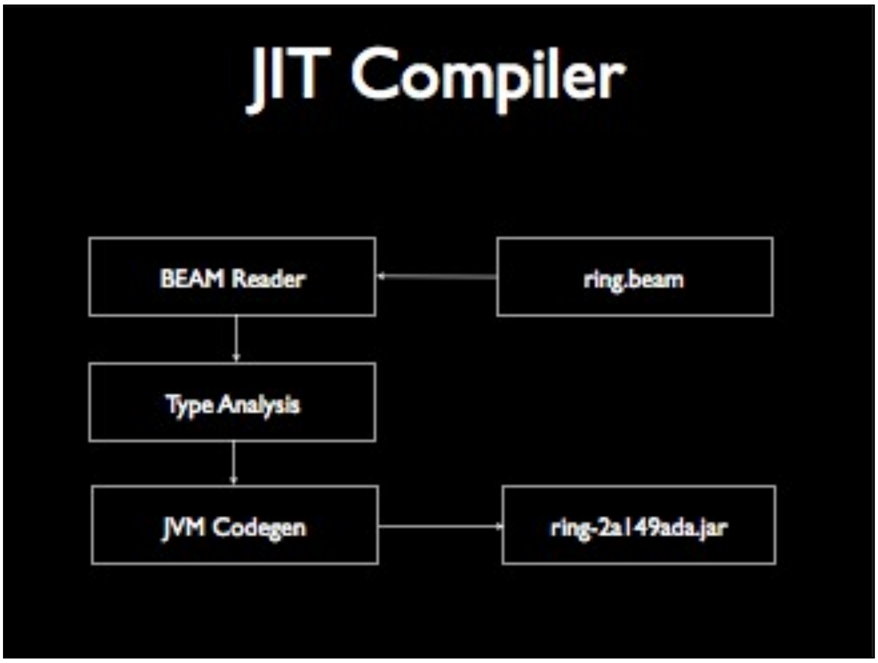
1: know/guess receiver type
then: Remove indirection

2: Inline function calls
then: propagate type info, and reapply

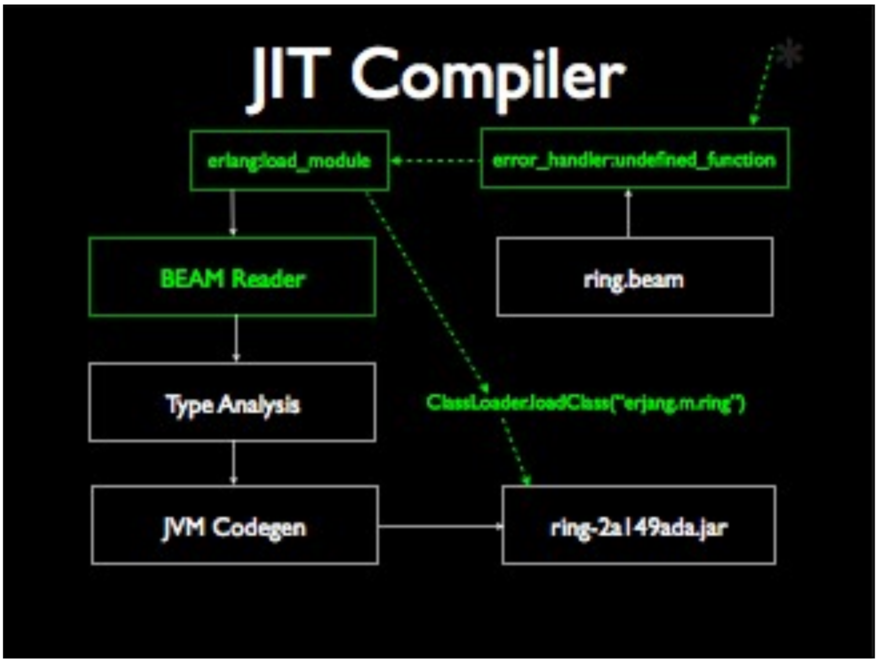


Erjang - Challenges

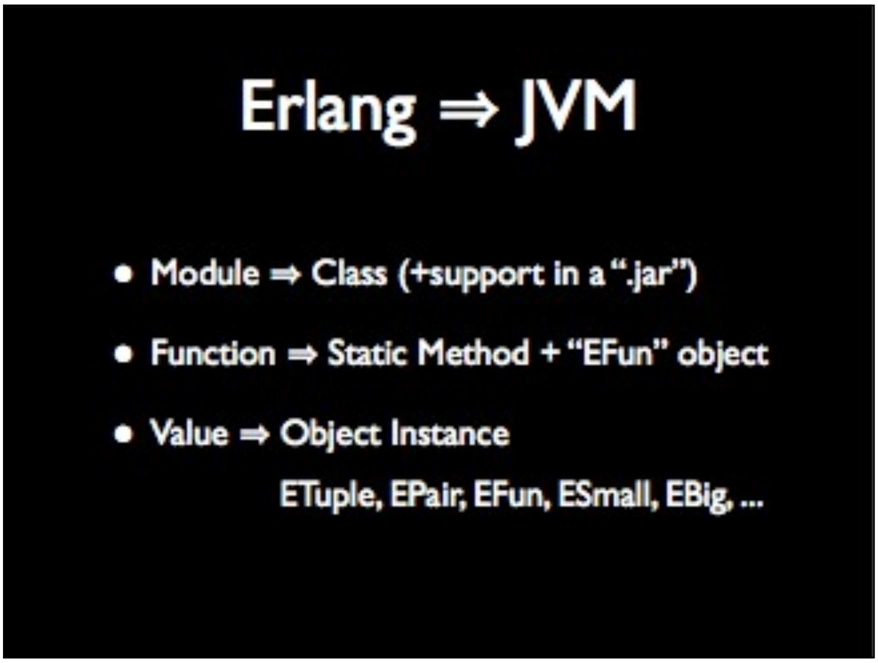
- Ultra Light-Weight Processes
- ~~Real-time Behavior~~
- Tail-recursion
- Arbitrary Precision Numbers
- Pattern Matching
- Erlang Drivers
- JVM is type safe, urgh!



JIT Compiler



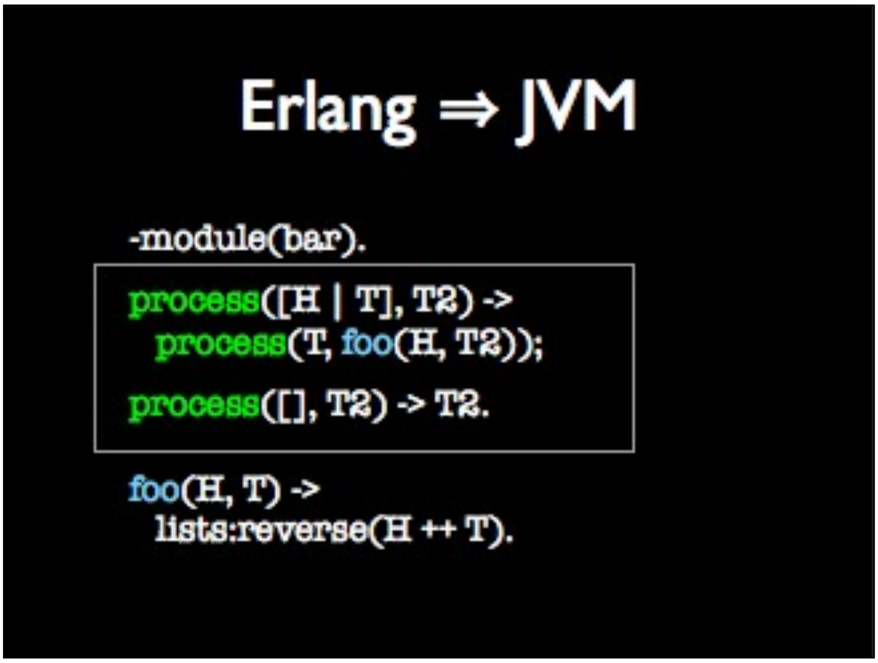
JIT Compiler



Erlang ⇒ JVM

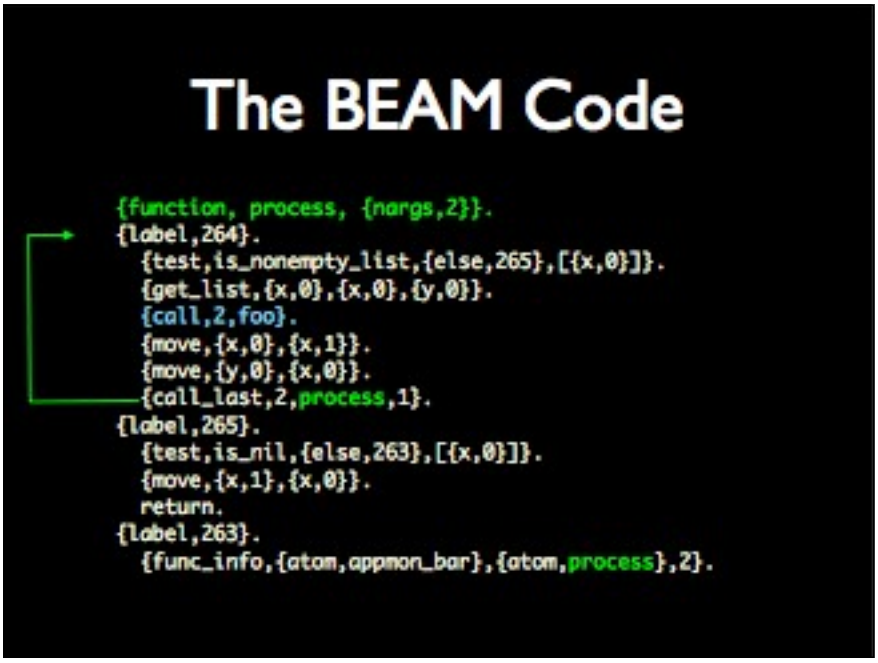
- Module ⇒ Class (+support in a ".jar")
- Function ⇒ Static Method + "EFun" object
- Value ⇒ Object Instance

  ETuple, EPair, EFun, ESmall, EBig, ...



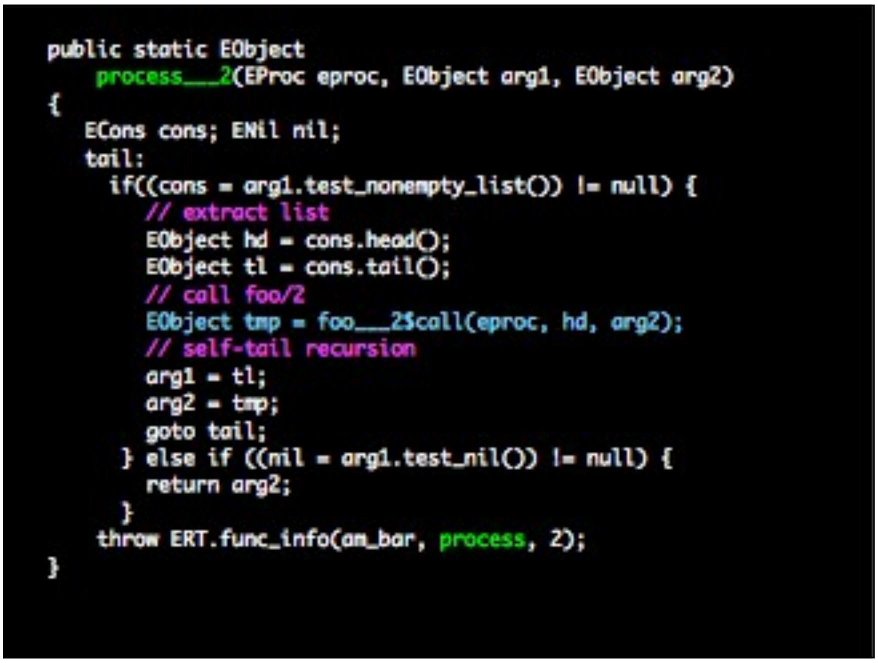
Erlang ⇒ JVM

```
-module(bar).

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

process([], T2) -> T2.

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



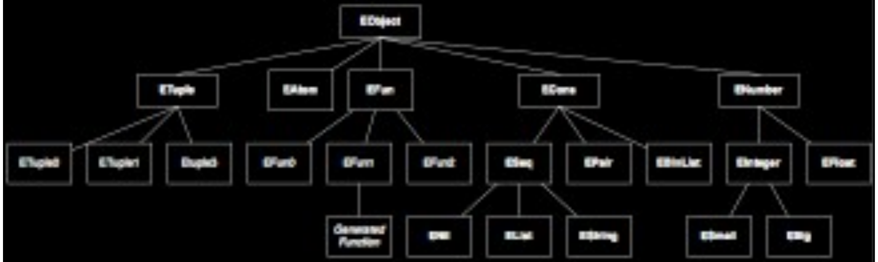
The BEAM Code

```
{function, process, {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,process,1}.
{label,265}.
  {test,is_nil,{else,263},[{x,0}]}.
  {move,{x,1},{x,0}}.
  return.
{label,263}.
  {func_info,{atom,appmon_bar},{atom,process},2}.
```

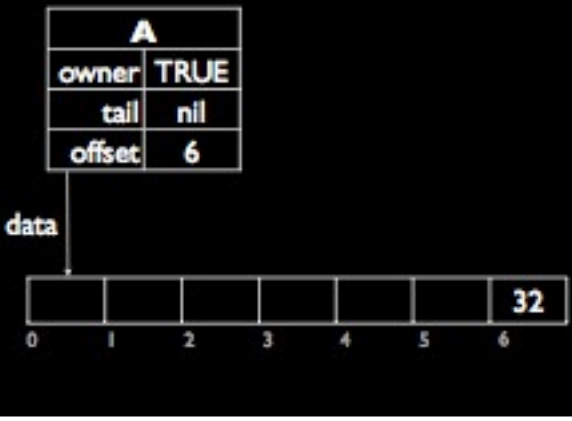


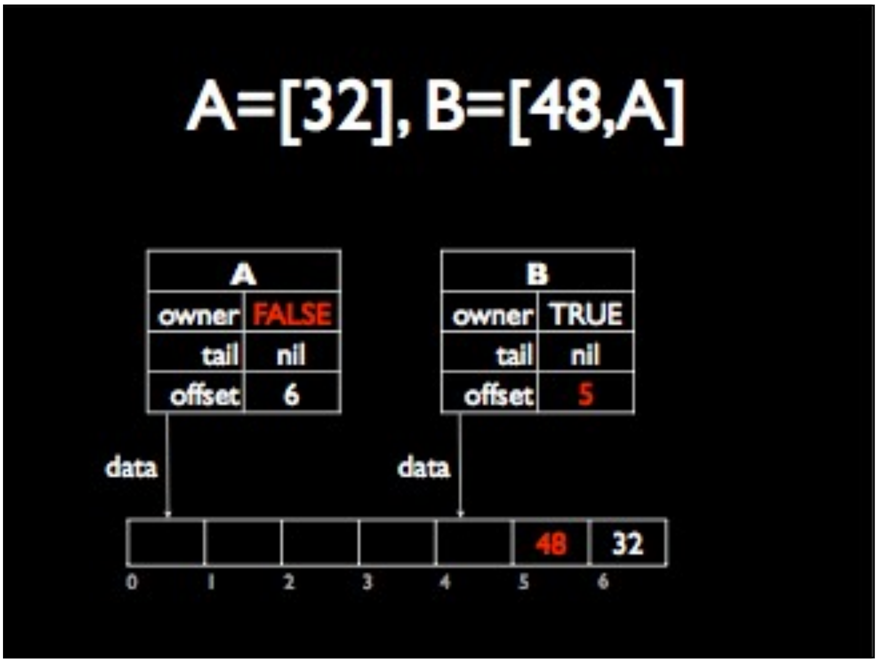
```
public static EObject
    process___2(EProc eproc, EObject arg1, EObject arg2)
{
   ECons cons; ENil nil;
   tail:
     if((cons = arg1.test_nonempty_list()) != null) {
        // extract list
        EObject hd = cons.head();
        EObject tl = cons.tail();
        // call foo/2
        EObject tmp = foo___2$call(eproc, hd, arg2);
        // self-tail recursion
        arg1 = tl;
        arg2 = tmp;
        goto tail;
      } else if ((nil = arg1.test_nil()) != null) {
        return arg2;
      }
    throw ERT.func_info(am_bar, process, 2);
}
```

Core Erjang Types



A=[32]



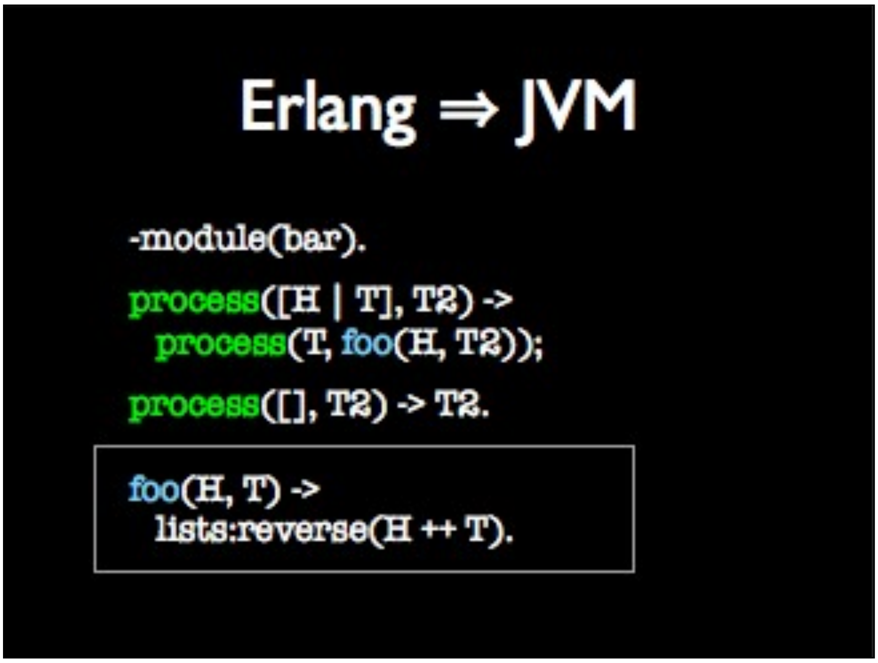


A=[32], B=[48,A]

A	
owner	FALSE
tail	nil
offset	6

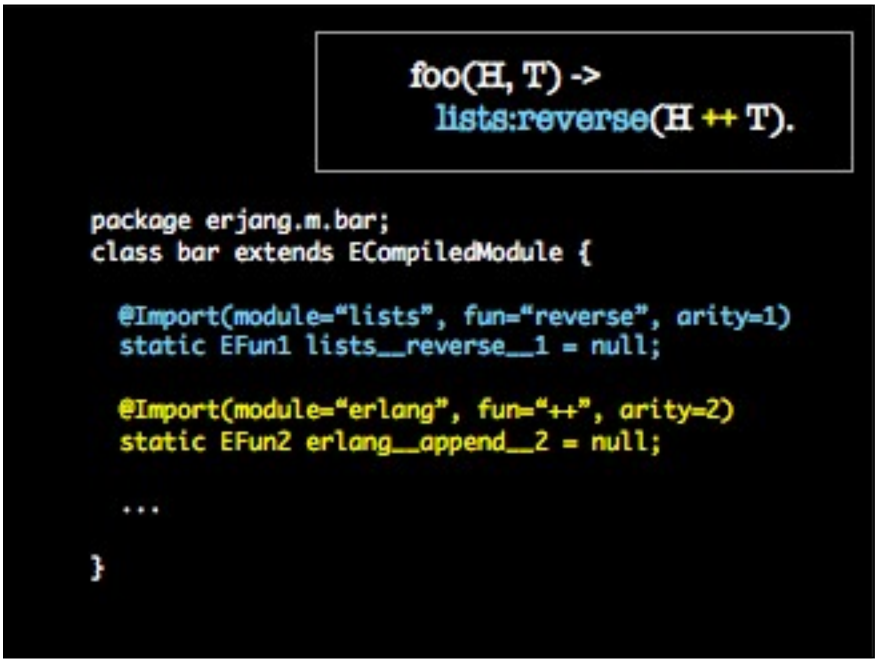
B	
owner	TRUE
tail	nil
offset	5

0	1	2	3	4	5	6
					48	32



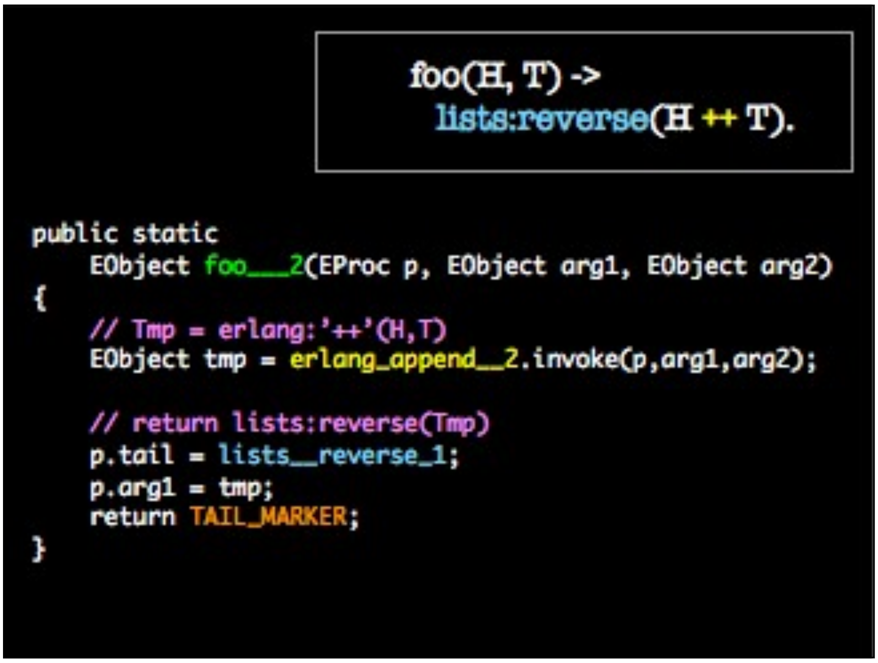
Erlang ⇒ JVM

```
-module(bar).

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

process([], T2) -> T2.

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



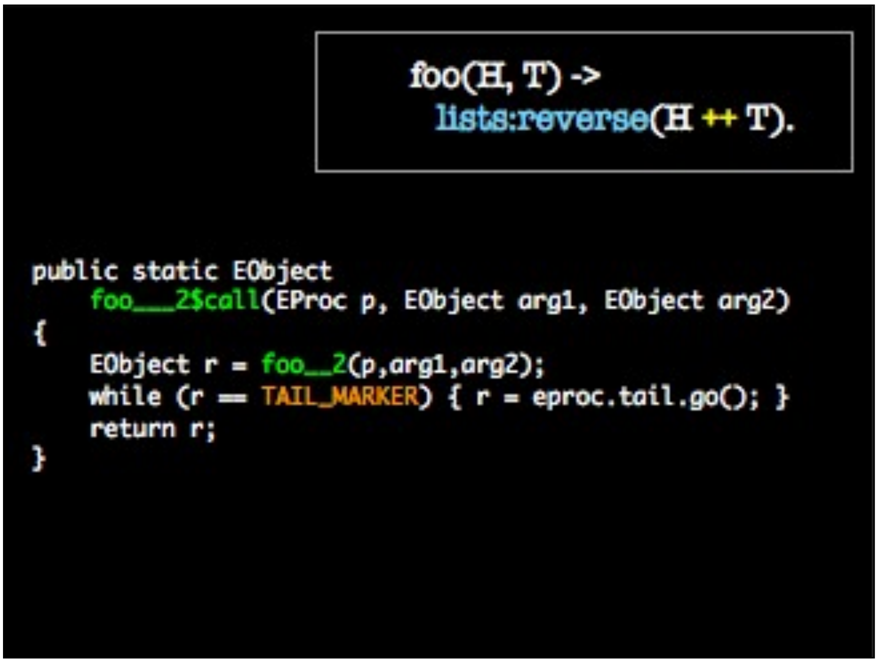
```
foo(H, T) ->
  lists:reverse(H ++ T).
```

```
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;

  ...

}
```



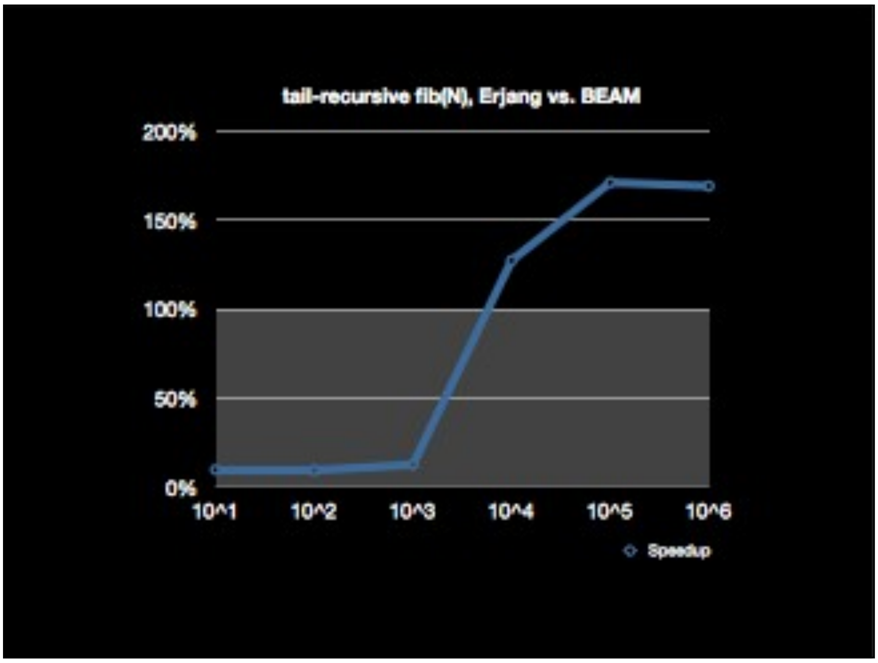
```
foo(H, T) ->
   lists:reverse(H ++ T).
```

```
public static
    EObject foo__2(EProc p, EObject arg1, EObject arg2)
{
    // Tmp = erlang:'++'(H,T)
    EObject tmp = erlang_append__2.invoke(p,arg1,arg2);

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



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

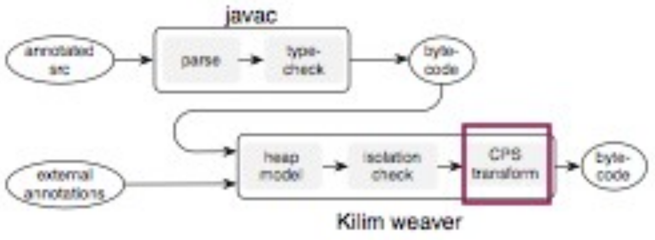
```
public static EObject
    foo__2$call(EProc p, EObject arg1, EObject arg2)
{
    EObject r = foo__2(p,arg1,arg2);
    while (r == TAIL_MARKER) { r = eproc.tail.go(); }
    return r;
}
```



Light-Weight Processes

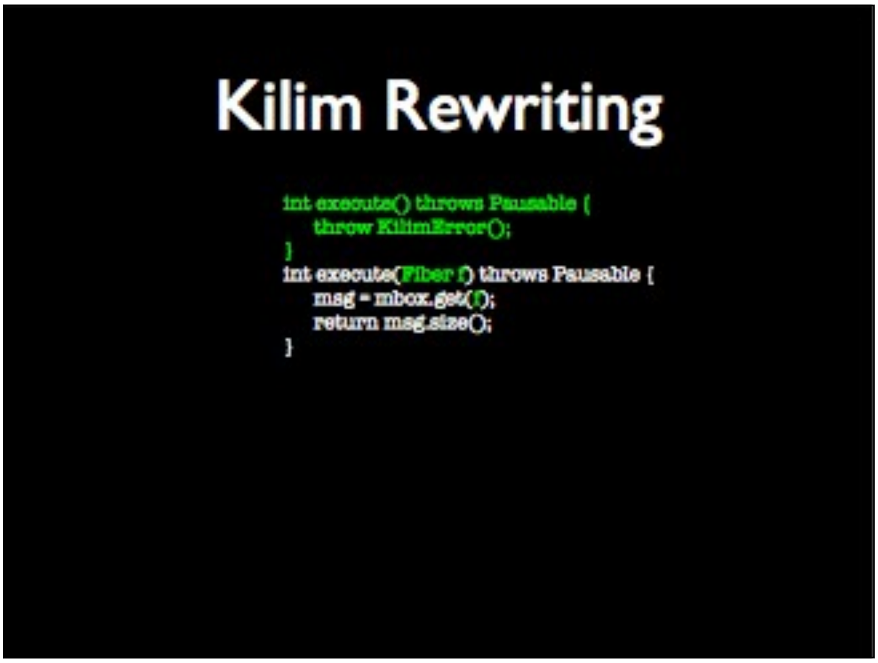
- Threads don't cut it;
  - Typical JVMs are limited to ~1000 threads
  - Context switch for threads is very expensive
- Erjang uses *Kilim*, a separate project
- Use one thread per CPU

Kilim



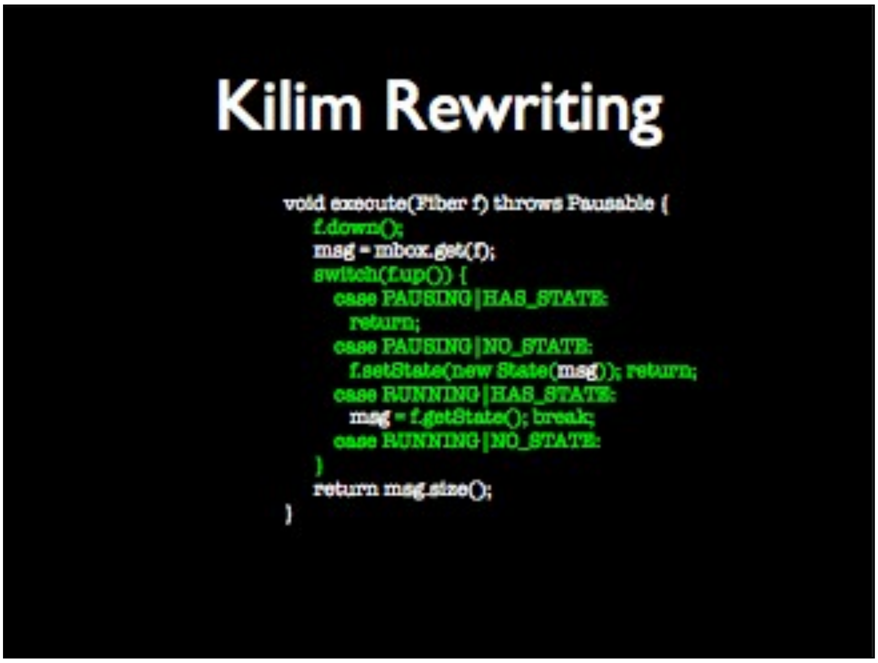
Kilim Rewriting

```
int execute() throws Pausable {
    msg = mbox.get();
    return msg.size();
}
```



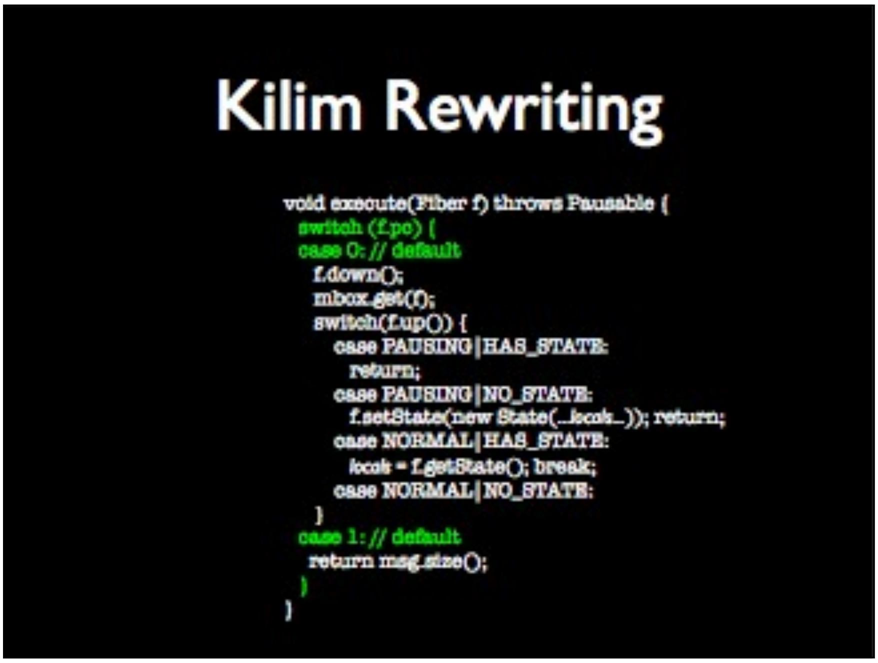
Kilim Rewriting

```
int execute() throws Pausable {
    throw KilimError();
}
int execute(Fiber f) throws Pausable {
    msg = mbox.get(f);
    return msg.size();
}
```



Kilim Rewriting

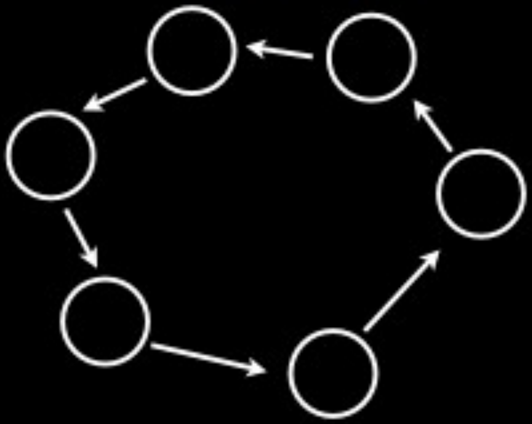
```
void execute(Fiber f) throws Pausable {
    f.down();
    msg = mbox.get(f);
    switch(f.up()) {
      case PAUSING|HAS_STATE:
        return;
      case PAUSING|NO_STATE:
        f.setState(new State(msg)); return;
      case RUNNING|HAS_STATE:
        msg = f.getState(); break;
      case RUNNING|NO_STATE:
    }
    return msg.size();
}
```

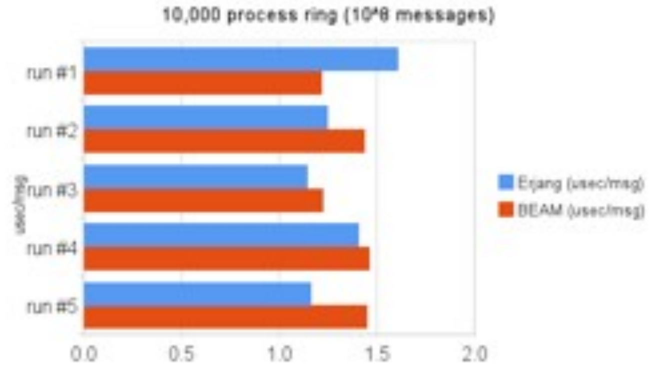


Kilim Rewriting

```
void execute(Fiber f) throws Pausable {
  switch (f.pc) {
  case 0: // default
    f.down();
    mbox.get(f);
    switch(f.up()) {
      case PAUSING|HAS_STATE:
        return;
      case PAUSING|NO_STATE:
        f.setState(new State(...locals...)); return;
      case NORMAL|HAS_STATE:
        locals = f.getState(); break;
      case NORMAL|NO_STATE:
    }
  case 1: // default
   return msg.size();
  }
}
```

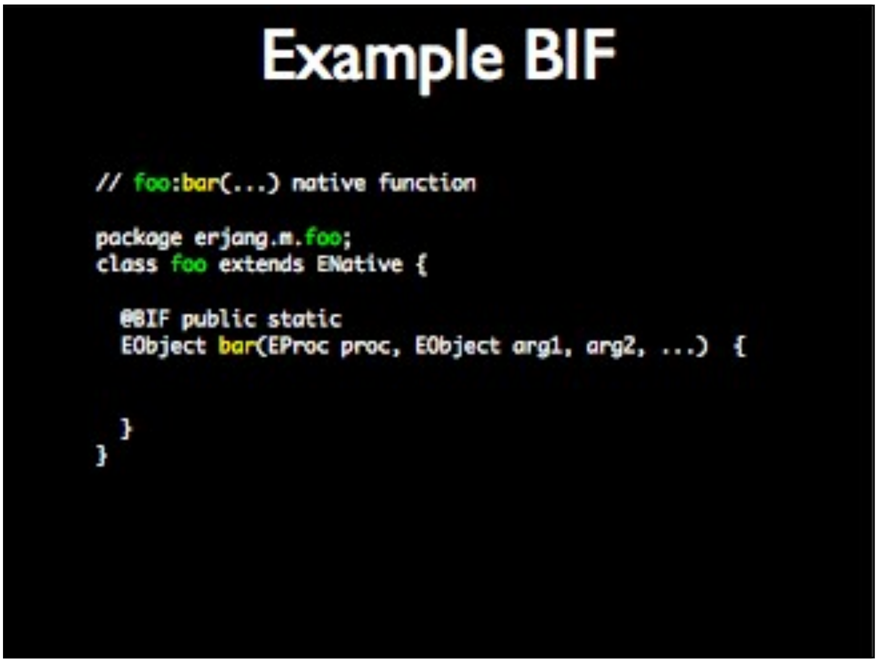
The ring!





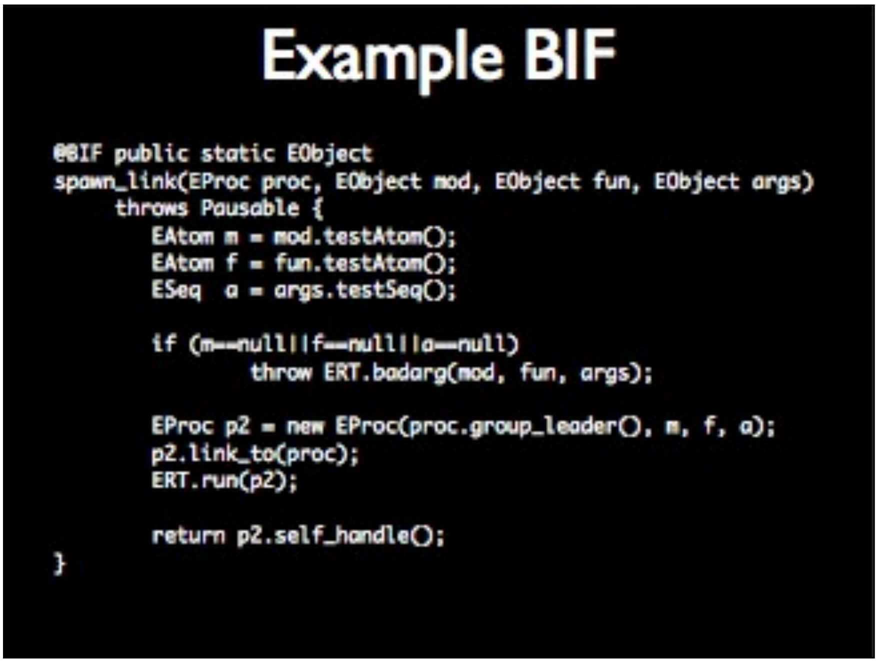
Interfacing to Java

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



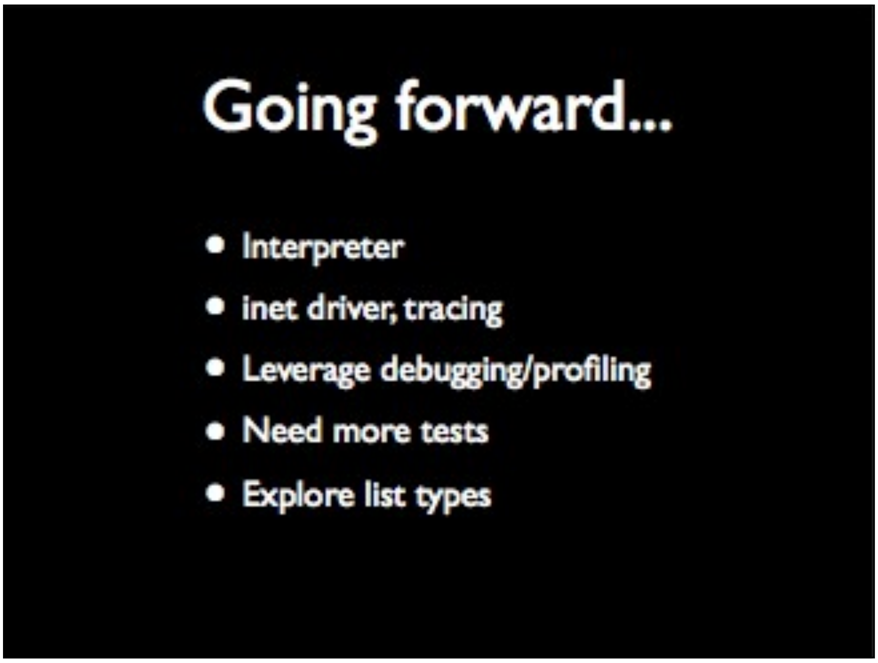
Example BIF

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

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

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


  }
}
```



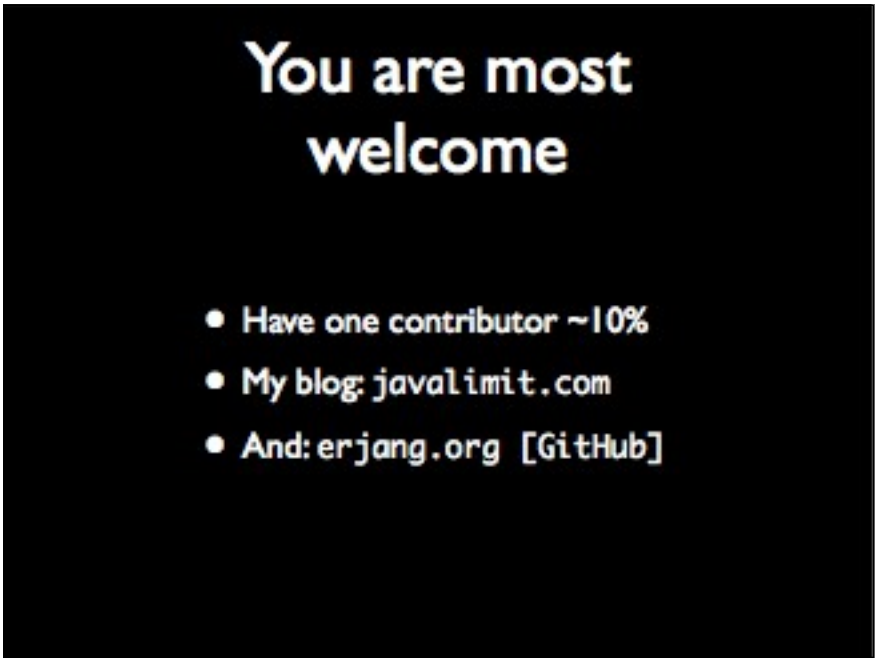
Example BIF

```
@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();
}
```



Going forward...

- Interpreter
- inet driver, tracing
- Leverage debugging/profiling
- Need more tests
- Explore list types



You are most welcome

- Have one contributor ~10%
- My blog: `javalimit.com`
- And: `erjang.org [GitHub]`

London 2010

Tutorials: March 8-9
Conference: March 10-12

QCon

www.qconlondon.com

INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE

Qcon London 2010

Speakers
Schedule
Tutorials
Tracks
Social Events

Exhibition
Sponsors

Registration
Volunteers

Venue
Travel
Hotels

User Groups @ QCon London

Organizer
TRIFORK.
InfoQ

About QCon
Contact
Archives
Future events

QCon is coming back to London

Tutorials: March 8-9, 2010

Conference: March 10-12, 2010

The fourth annual London enterprise software development conference designed for team leads, architects and project management is back! There is no other event in the UK with similar opportunities for learning, networking, and tracking innovation occurring in the Java, .NET, Ruby, SOA, Agile, and architecture

Tracks for QCon London

Architectures You've Always Wondered About - Features Facebook, Skype, Sky.com, Bwin, League of Legends

Agile Evolution - the changing face of Agile andfuture directions.

How do you test that? - Difficult problems and state ofthe art in testing: functional, performance, integration, and more.

AlphaGeeks on .NET - Alpha geeks go beyond MS'prescriptive .NET solutions, hear how.

Cool Stuff with Java - Practitioner examples ofunusually cool problem-solutions in Java.

Functional programming - Everything you ever wanted to know about functional

Register before February 22 and save up to £197/€224

Register now >>

QCon Videos

Rich Hickey - *"Persistent Data Structures and Managed References"*

Martin Fowler - *"Three Years of Real-World Ruby"*

Tony Hoare - *"Null References: The Billion Dollar Mistake"*

Jonas Bonér - *"Pragmatic Real-World Scala"*

Aditya Agarwal - *"Facebook: Science and the Social Graph"*