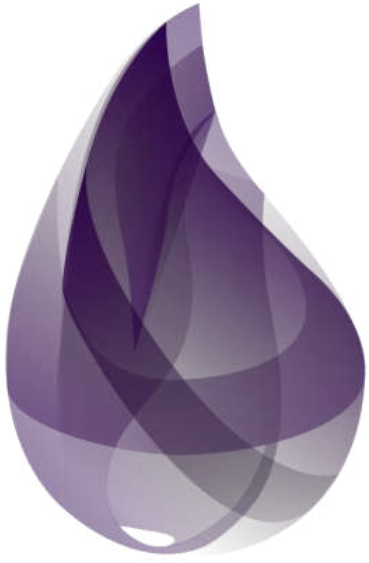
eir

Erlang Compiler Infrastructure Project

Hans Elias B. Josephsen, @hansihe

Phoenix LiveView: Interactive, Real-Time Apps. No Need to Write JavaScript.

By: Chris McCord • December 12th, 2018



+

Elixir on the web?


Parallelism &
Concurrency
Fault
tolerance
Hot-code
reloading


Download size
Startup time


WA


C++, C or Rust
WASM
JS
CSS
HTML

eir

Erlang Compiler Infrastructure Project

Technical subtitle:

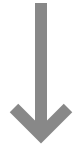
Eir: SSA-based IR for BEAM languages, designed for compatibility with LLVM

Technical subtitle:

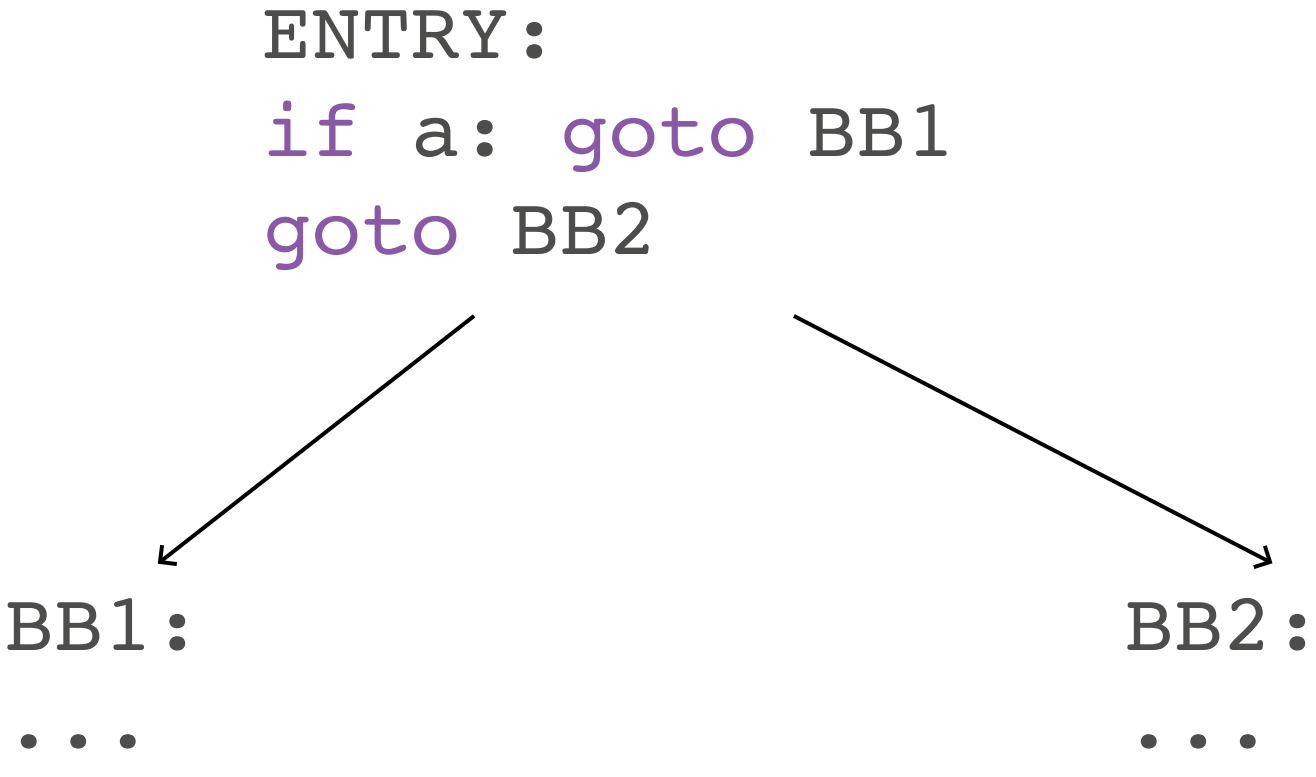
Eir: SSA-based IR for BEAM languages, designed for compatibility with LLVM

```
if a: do b
```

Intermediate Representation



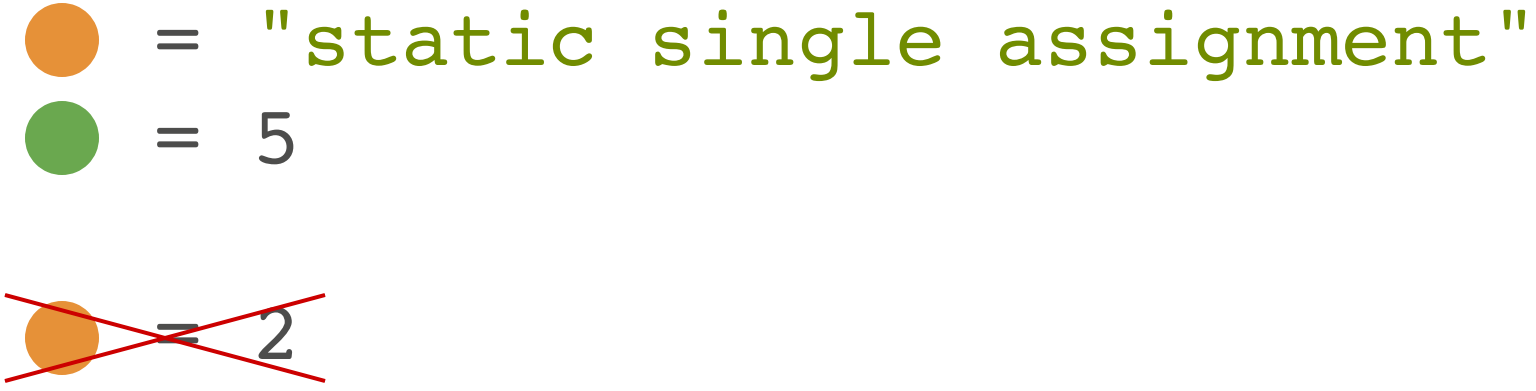
```
0xdeadbeef
```

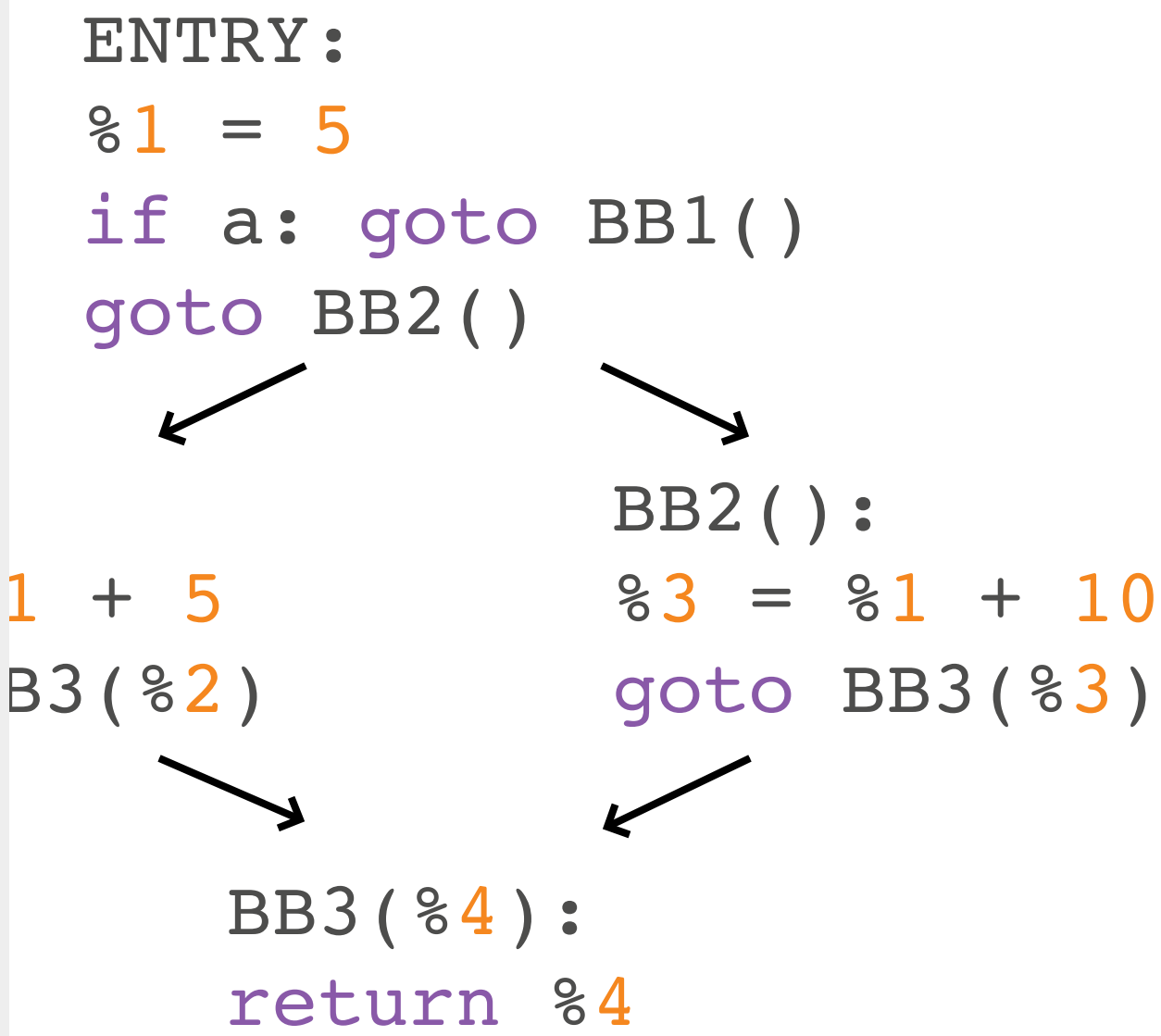

ENTRY:
if a: goto BB1
goto BB2
BB1:
...
BB2:
...


= "static single assignment"


= 5


= 2

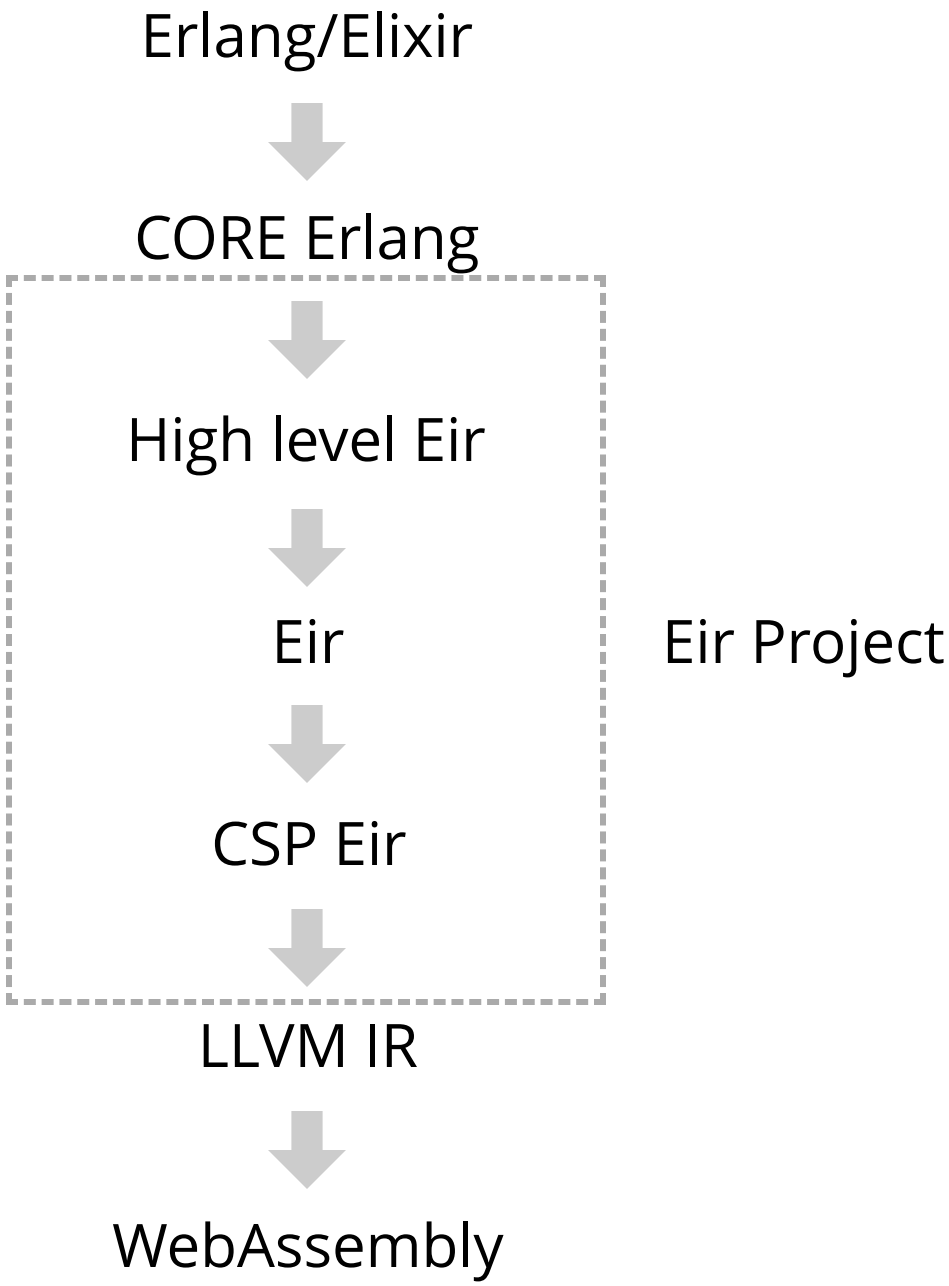

= "static single assignment"
= 5
= 2


ENTRY:
%1 = 5
if a: goto BB1()
goto BB2()
BB2():
%3 = %1 + 10
goto BB3(%3)
1 + 5
B3(%2)
BB3(%4):
return %4

LLVM

The **LLVM** compiler infrastructure project is a "collection of modular and reusable compiler and toolchain technologies"[3] used to develop compiler front ends and back ends.

[...] designed for compile-time, link-time, run-time, and "idle-time" optimization of programs written in arbitrary programming languages.


Erlang/Elixir
CORE Erlang
High level Eir
Eir
CSP Eir
LLVM IR
WebAssembly
Eir Project

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

```
1 def my_fun(:hi), do: :hello
2 def my_fun(:bye), do: Goodbye.run() + 2
```

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

```
1  'my_fun'/1 =
2      %% Line 43
3      fun (_0) ->
4          case _0 of
5            <'hi'> when 'true' ->
6                'hello'
7            %% Line 44
8            <'bye'> when 'true' ->
9                let <_1> =
10                   call 'Elixir.Goodbye':'run'
11                       ()
12               in  call 'erlang':'+'
13                       (_1, 2)
14           ( <_2> when 'true' ->
15                 ( primop 'match_fail'
16                       ({'function_clause',_2})
17                   -| [{'function_name',{'my_fun',1}}] )
18             -| ['compiler_generated'] )
19         end
```

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

```
1  'my_fun'/1 =
2      %% Line 43
3      fun (_0) ->
4          case _0 of
5            <'hi'> when 'true' ->
6                'hello'
7            %% Line 44
8            <'bye'> when 'true' ->
9                let <_1> =
10                   call 'Elixir.Goodbye':'run'
11                       ()
12               in  call 'erlang':'+'
13                       (_1, 2)
14           ( <_2> when 'true' ->
15                 ( primop 'match_fail'
16                       ({'function_clause',_2})
17                   -| [{'function_name',{'my_fun',1}}] )
18             -| ['compiler_generated'] )
19         end
```

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

```
'my_fun'/1 =
    %% Line 43
    fun (_0) ->
        case _0 of
          <'hi'> when 'true' ->
              'hello'
          %% Line 44
          <'bye'> when 'true' ->
              let <_1> =
                  call 'Elixir.Goodbye':'run'
                      ()
              in  call 'erlang':'+'
                      (_1, 2)
          ( <_2> when 'true' ->
                ( primop 'match_fail'
                      ({'function_clause',_2})
                  -| [{'function_name',{'my_fun',1}}] )
            -| ['compiler_generated'] )
        end
```

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

```
my_fun/1 {
    %5 = [];
    %7 = a"true";
    %9 = a"hello";
    %11 = a"true";
    %13 = a"Elixir.Goodbye";
    %14 = a"run";
    %17 = a"erlang";
    %18 = a"+";
    %19 = 2;
    %24 = a"true";
    %26 = a"function_clause";
    %28 = a"error";
    %29 = a"internal_err_data";

    B0(%0):
        %4 = case_start on: %0, values: [] {
            clause assigns: [] {
                pattern a"hi";
            };
            clause assigns: [] {
                pattern a"bye";
            };
            clause assigns: [A0] {
                pattern A0 = (_);
            };
        } branch B3();

    B5:
        jump B1(%5);

    B8:
        %22 = case_values %4;
        %25 = pack_value_list;
        if_truthy %24 else B11(%25);
        case_guard_ok %4;
        %27 = make_tuple [%26, %22];
        %30 = make_tuple [%28, %27, %29];
        jump B1(%30);

    B12:
        %31 = pack_value_list;
        jump B4(%31);

    B11(%23):
        case_guard_fail %4 branch B3();

    B7:
        case_values %4;
        %12 = pack_value_list;
        if_truthy %11 else B10(%12);
        case_guard_ok %4;
        %15, %16 = call %13:%14/0() except B1(%16);
        %20, %21 = call %17:%18/2(%15, %19) except B1(%21);
        jump B4(%20);

    B10(%10):
        case_guard_fail %4 branch B3();

    B6:
        case_values %4;
        %8 = pack_value_list;
        if_truthy %7 else B9(%8);
        case_guard_ok %4;
        jump B4(%9);

    B9(%6):
        case_guard_fail %4 branch B3();

    B4(%3):
        jump B2(%3);

    B3:
        case_body %4 branch B5(), B6(), B7(), B8();

    B1(%1):
        return_throw %1;

    B2(%2):
        return_ok %2;

}
```

```
%5 = [];
%7 = a"true";
%9 = a"hello";
%11 = a"true";
%13 = a"Elixir.Goodbye";
%14 = a"run";
%17 = a"erlang";
%18 = a"+";
%19 = 2;
%24 = a"true";
%26 = a"function_clause";
%28 = a"error";
%29 = a"internal_err_data";

B0(%0):
    %4 = case_start on: %0, values: [] {
        clause assigns: [] {
            pattern a"hi";
        };
        clause assigns: [] {
            pattern a"bye";
        };
        clause assigns: [A0] {
            pattern A0 = (_);
        };
    } branch B3();

B5:
    jump B1(%5);

B8:
    %22 = case_values %4;
    %25 = pack_value_list;
    if_truthy %24 else B11(%25);
    case_guard_ok %4;

    %27 = make_tuple [%26, %22];
%30 = make_tuple [%28, %27, %29];
```

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

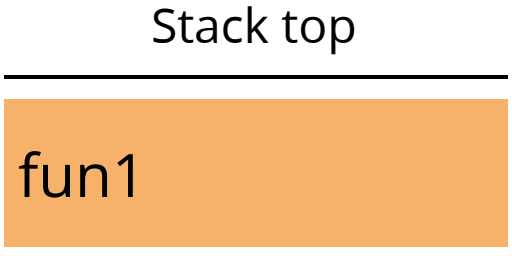
```
1  my_fun/1 {
2      %9 = a"hello";
3      %13 = a"Elixir.Goodbye";
4      %14 = a"run";
5      %17 = a"erlang";
6      %18 = a"+";
7      %19 = 2;
8      %26 = a"function_clause";
9      %28 = a"error";
10     %29 = a"internal_err_data";
11     %32 = a"bye";
12     %33 = a"hi";
13
14     B0(%0):
15         compare equal [%0, %32] branch B22();
16         %15, %16 = call %13:%14/0() except B1(%16);
17         call tail %17:%18/2(%15, %19);
18
19     B22:
20         compare equal [%0, %33] branch B23();
21         return_ok %9;
22
23     B23:
24         %27 = make_tuple [%26, %0];
25         %30 = make_tuple [%28, %27, %29];
26         jump B1(%30);
27
28     B1(%1):
29         return_throw %1;
30
31 }
```

An aside:
Tail calls

Stack top

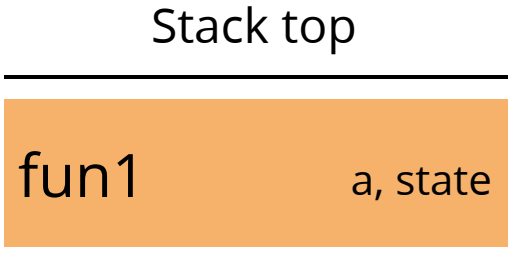
```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls



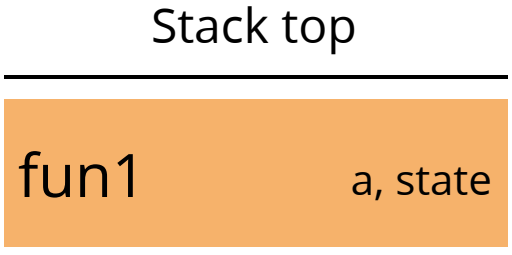

```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls



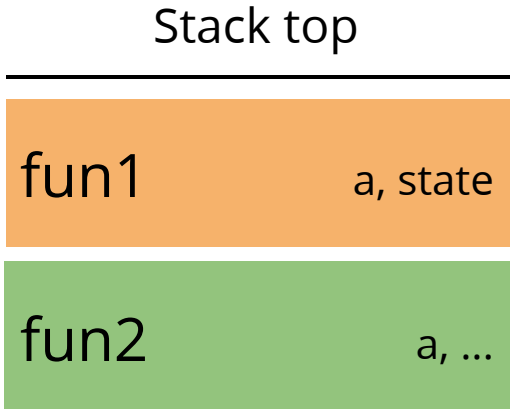

```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls



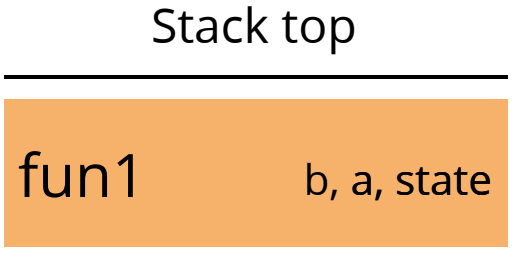

```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls



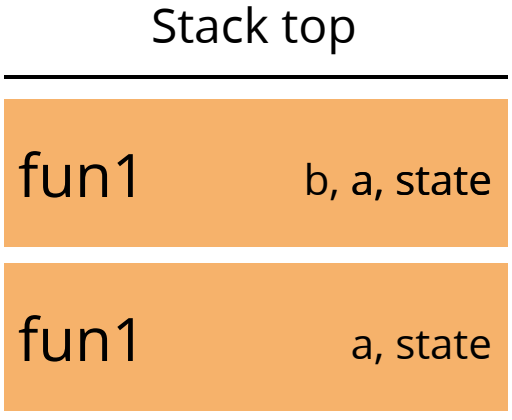

```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls



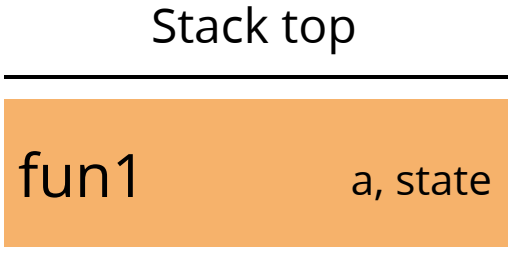

```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls




```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

An aside:
Tail calls




```
1 def fun1(a, state) do
2   b = fun2(a)
3   fun1(b, state)
4 end
```

WebAssembly doesn't like tail calls :(

WebAssembly doesn't like tail calls :(

Solution?

Make all calls tail-calls!

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

WebAssembly

#CodeBEAMSTO

```
1  my_fun/1 {
2      %3 = a"bye";
3      %5 = a"hi";
4      %10 = a"Elixir.Goodbye";
5      %11 = a"run";
6      %15 = a"hello";
7      %19 = a"function_clause";
8      %21 = a"error";
9      %22 = a"internal_err_data";
10
11     B0(%0, %1, %2):
12         compare equal [%2, %3] branch B1();
13         %6 = pack_env E24 [%0, %1];
14         %7 = bind_closure Elixir.NiffyTest.NifTest:my_fun@24.0/1 with %6;
15         %8 = pack_env E25 [%0, %1];
16         %9 = bind_closure Elixir.NiffyTest.NifTest:my_fun@25.0/1 with %8;
17         call tail %10:%11/0(%7, %9);
18
19     B1:
20         compare equal [%2, %5] branch B2();
21         apply cont %0(%15);
22
23     B2:
24         %18 = make_tuple [%19, %2];
25         %20 = make_tuple [%21, %18, %22];
26         jump B3(%20);
27
28     B3(%23):
29         apply cont %1(%23);
30
31 }
32
33 my_fun@24.0/1 {
34     %4 = 2;
35     %5 = a"erlang";
36     %6 = a"+";
37
38     B0(%0, %1):
39         %2, %3 = unpack_env %0;
40         call tail %5:%6/2(%2, %3, %1, %4);
41
42 }
43
44 my_fun@25.0/1 {
45
46     B0(%0, %1):
47         %2, %3 = unpack_env %0;
48         apply cont %3(%1);
49
50 }
```

Erlang/Elixir

CORE Erlang

High level Eir

Eir

CSP Eir

LLVM IR

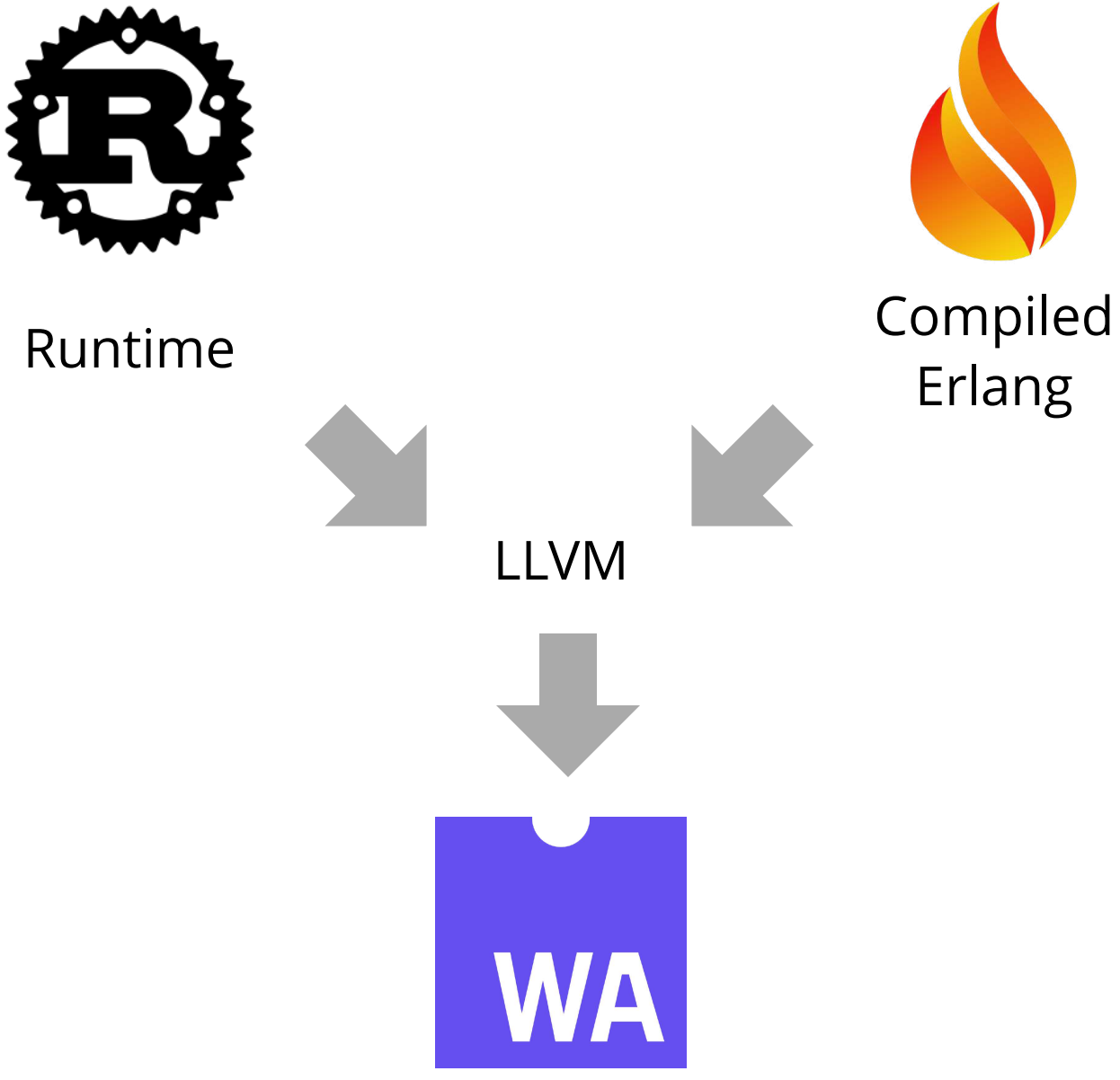
WebAssembly

#CodeBEAMSTO

```
define void @GNIP7_testing7_my__fun1_n_n(%whirl_process_env*, i64, i64, i64, i64) {
entry:
  br label %ebb0

ebb0:                                             ; preds = %entry
  %value0 = phi i64 [ %2, %entry ]
  %value1 = phi i64 [ %3, %entry ]
  %value2 = phi i64 [ %4, %entry ]
  %5 = load i64, i64* @whirlc_module_testing_atom_hi
  %6 = call i1 @whirlrt_term_eq(%whirl_process_env* %0, i64 %value2, i64 %5)
  br i1 %6, label %compare_eq_ok, label %ebb1

ebb1:                                             ; preds = %ebb0
  %7 = load i64, i64* @whirlc_module_testing_atom_bye
  %8 = call i1 @whirlrt_term_eq(%whirl_process_env* %0, i64 %value2, i64 %7)
  br i1 %8, label %compare_eq_ok1, label %ebb2

ebb2:                                             ; preds = %ebb1
  %9 = alloca i64, i32 0
  %10 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 0, i64* %9)
  %11 = load i64, i64* @whirlc_module_testing_atom_function_clause
  %12 = alloca i64, i32 2
  %13 = getelementptr i64, i64* %12, i64 0
  store i64 %11, i64* %13
  %14 = getelementptr i64, i64* %12, i64 1
  store i64 %value2, i64* %14
  %15 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 2, i64* %12)
  %16 = load i64, i64* @whirlc_module_testing_atom_error
  %17 = load i64, i64* @whirlc_module_testing_atom_internal_err_data
  %18 = alloca i64, i32 3
  %19 = getelementptr i64, i64* %18, i64 0
  store i64 %16, i64* %19
  %20 = getelementptr i64, i64* %18, i64 1
  store i64 %15, i64* %20
  %21 = getelementptr i64, i64* %18, i64 2
  store i64 %17, i64* %21
  %22 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 3, i64* %18)
  br label %ebb3

ebb3:                                             ; preds = %ebb2
  %value23 = phi i64 [ %22, %ebb2 ]
  call void @whirlrt_call_cont(%whirl_process_env* %0, i64 %value1, i64 %value23)
  unreachable

compare_eq_ok:                                    ; preds = %ebb0
  %23 = alloca i64, i32 0
  %24 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 0, i64* %23)
  %25 = load i64, i64* @whirlc_module_testing_atom_hello
  call void @whirlrt_call_cont(%whirl_process_env* %0, i64 %value0, i64 %25)
  unreachable

compare_eq_ok1:                                   ; preds = %ebb1
  %26 = alloca i64, i32 0
  %27 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 0, i64* %26)
  %28 = alloca i64, i32 2
  %29 = getelementptr i64, i64* %28, i64 0
  store i64 %value0, i64* %29
  %30 = getelementptr i64, i64* %28, i64 1
  store i64 %value1, i64* %30
  %31 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 2, i64* %28)
  call void @whirlrt_temp_hacky_transmute_tup_to_fun_env(%whirl_process_env* %0, i64 %31, i8* bitcast (void (%whirl_process_env*, i64, i64)* @GNIP7_testing7_my__fun1_lambda_env42_0 to i8*))
  %32 = alloca i64, i32 2
  %33 = getelementptr i64, i64* %32, i64 0
  store i64 %value0, i64* %33
  %34 = getelementptr i64, i64* %32, i64 1
  store i64 %value1, i64* %34
  %35 = call i64 @whirlrt_term_make_tuple(%whirl_process_env* %0, i32 2, i64* %32)
  call void @whirlrt_temp_hacky_transmute_tup_to_fun_env(%whirl_process_env* %0, i64 %35, i8* bitcast (void (%whirl_process_env*, i64, i64)* @GNIP7_testing7_my__fun1_lambda_env43_0 to i8*))
  %36 = load i64, i64* @whirlc_module_testing_atom_Elixir.Goodbye
  %37 = load i64, i64* @whirlc_module_testing_atom_run
  %38 = call i64 @whirlrt_term_make_fun(%whirl_process_env* %0, i8* bitcast (void (%whirl_process_env*, i64, i64, i64)* @GNIP14_Elixir.Goodbye3_run0_n_n to i8*))
  call void @GNIP14_Elixir.Goodbye3_run0_n_n(%whirl_process_env* %0, i64 %38, i64 %31, i64 %35)
  unreachable
}

declare void @GNIP14_Elixir.Goodbye3_run0_n_n(%whirl_process_env*, i64, i64, i64)

define void @GNIP7_testing7_my__fun1_lambda_env43_0(%whirl_process_env*, i64, i64) {
entry:
  br label %ebb0

ebb0:                                             ; preds = %entry
  %value0 = phi i64 [ %1, %entry ]
  %value1 = phi i64 [ %2, %entry ]
  %3 = alloca i64, i32 2
  call void @whirlrt_term_unpack_closure_env(%whirl_process_env* %0, i64 %value0, i32 2, i64* %3)
  %4 = getelementptr i64, i64* %3, i32 0
  %5 = load i64, i64* %4
  %6 = getelementptr i64, i64* %3, i32 1
  %7 = load i64, i64* %6
  call void @whirlrt_call_cont(%whirl_process_env* %0, i64 %7, i64 %value1)
  unreachable
}

define void @GNIP7_testing7_my__fun1_lambda_env42_0(%whirl_process_env*, i64, i64) {
entry:
  br label %ebb0

ebb0:                                             ; preds = %entry
  %value0 = phi i64 [ %1, %entry ]
  %value1 = phi i64 [ %2, %entry ]
  %3 = alloca i64, i32 2
  call void @whirlrt_term_unpack_closure_env(%whirl_process_env* %0, i64 %value0, i32 2, i64* %3)
  %4 = getelementptr i64, i64* %3, i32 0
  %5 = load i64, i64* %4
  %6 = getelementptr i64, i64* %3, i32 1
  %7 = load i64, i64* %6
  %8 = load i64, i64* @whirlc_module_testing_atom_erlang
  %9 = load i64, i64* @"whirlc_module_testing_atom_+"
  %10 = call i64 @whirlrt_term_make_smallint(%whirl_process_env* %0, i64 2)
  %11 = call i64 @whirlrt_term_make_fun(%whirl_process_env* %0, i8* bitcast (void (%whirl_process_env*, i64, i64, i64, i64, i64)* @GNIF6_erlang2__p2_n_n to i8*))
  call void @GNIF6_erlang2__p2_n_n(%whirl_process_env* %0, i64 %11, i64 %5, i64 %7, i64 %value1, i64 %10)
  unreachable
}
```

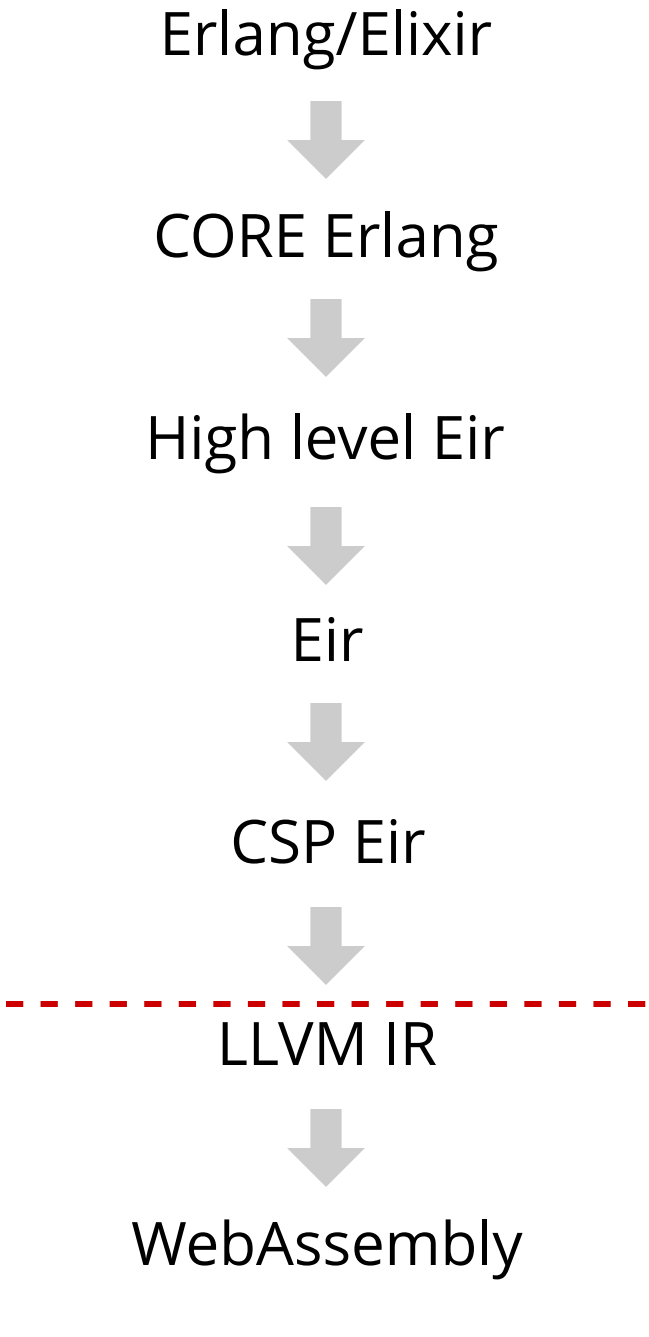
Introducing Whirl

WebAssembly runtime for BEAM languages


Runtime
Compiled Erlang
LLVM
WA

Demo

Let's step back


Erlang/Elixir
CORE Erlang
High level Eir
Eir
CSP Eir
LLVM IR
WebAssembly

Niffy

```elixir
defmodule NiffyTest.NifTest do
  use Niffy

  # The following function will be compiled
  # to native code and loaded as a NIF.

  @niffy true
  def woohoo(a) do
    case a do
      1 -> :woo
      2 -> 1
      _ -> a + 2
    end
  end

end
```

All open source!

github.com/eirproject

Hans Elias B. Josephsen

@hansihe