PAUL SCHOENFELDER

LUMEN

DOCKYARD

WHAT IS LUMEN?

A new compiler/runtime for Erlang/Elixir

Brings these languages to WebAssembly, with support for other targets as well

Provides a path for building things in these languages that are not necessarily well supported by the BEAM

WHY ARE WE BUILDING IT?

The future of the web lies with WebAssembly

Client-side ecosystem is in constant flux

Why split investment and expertise across two ecosystems when one will do?

WHY ERLANG/ELIXIR?

Better cross-pollination

Take advantage of the powerful tools provided by Erlang/OTP on both server and client

Actors are an excellent pattern for building UIs

WA

WHAT YOU NEED TO KNOW

WEBASSEMBLY

DOCKYARD

WHAT IS WEBASSEMBLY?

An instruction set + binary and textual formats

Designed for a stack-based virtual machine

Portable, embeddable

Memory-safe, sandboxed

HIGH-LEVEL DESIGN

Harvard Architecture (separate code/data)

Structured control flow vs arbitrary CFG

Only permits passing integers to/from JS*

* this is changing with Interface Types

TARGETING WASM WITH LUMEN

DOCKYARD

CONSTRAINTS

Code Size

Load Time

Concurrency Model

JAVASCRIPT/DOM INTEROP

FFI

Async Functions

Events

BUT WHY NOT USE THE BEAM?

Runtime

Code Size

Performance

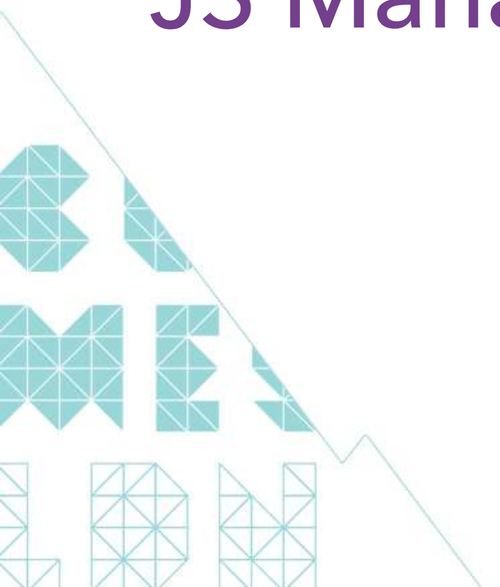
DOCKYARD

RUNTIME

Many APIs unavailable/unsupported

Incompatible Scheduler

JS Managed Types



DOCKYARD

CODE SIZE

Shipping BEAM bytecode is expensive

Weak dead-code elimination

PERFORMANCE

VM on a VM

JS engine unable to reason about bytecode

A NEW COMPILER/RUNTIME

Impose some restrictions

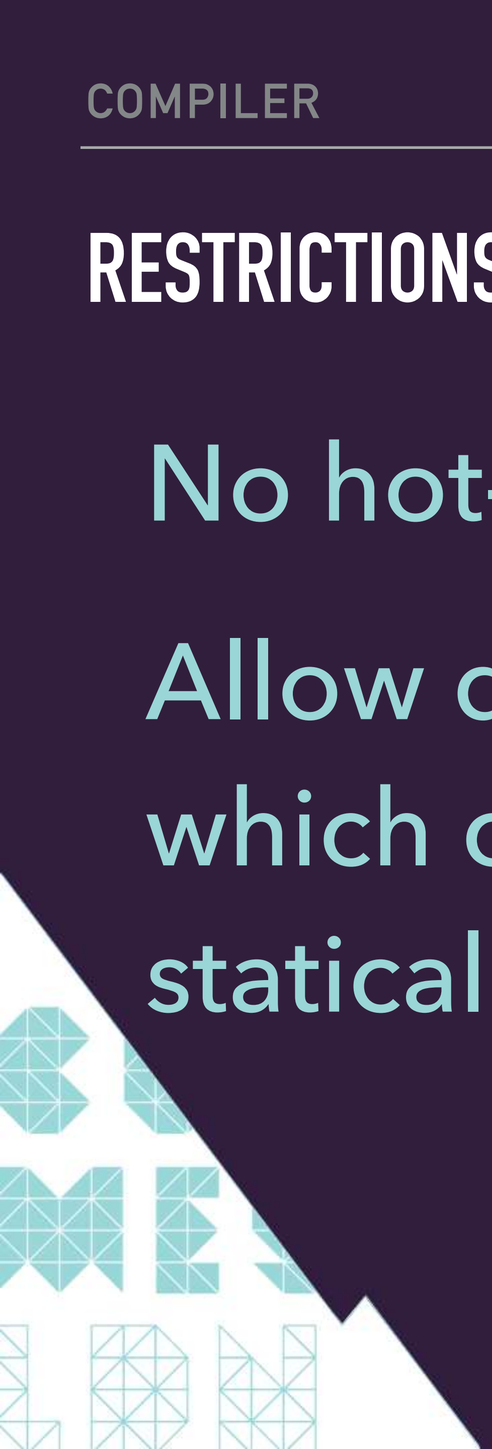
Ahead-of-Time vs VM

Take advantage of existing tools

RESTRICTIONS

No hot-code loading

Allow dead-code elimination to remove code which cannot be determined to be reachable statically



AOT VS VM

Only pay for what you use

No interpretation overhead

Enables a wide-variety of optimizations

- Including target specific optimizations

BUILD ON EXISTING TOOLS

LLVM

Rust

wasm-bindgen

KEY CHALLENGES

Recursion/tail-call optimization

Non-local returns/exceptions

Green threads/preemptive scheduling

WebAssembly-specific limitations

WEBASSEMBLY CHALLENGES

Abstract machine is stack-based

Requires structured control flow

No direct access to the stack

BEST SOLUTION?

CONTINUATIONS!

DOCKYARD

CONTINUATIONS

Represent jumps as calls to a continuation

All continuations are in tail position

Continuations never return

Can represent all control flow constructs

CONTINUATIONS

```
def fork(proc, k \\ nil) do
  unless is_nil(k), do: enqueue(k)
  proc.()
end
def yield(k) do
  enqueue(k)
  next = dequeue()
  unless is_nil(next), do: next.()
end
def loop(n, msg) do
  IO.puts("#{msg}: #{n}")
  yield(fn -> loop(n + 1, msg) end)
end
```

```
def spawn_proc(msg) do
  fn -> loop(10, msg) end
end
def start() do
  fork(spawn_proc("A"), fn ->
    fork(spawn_proc("B"))
  end)
end
```

```
> Mod.start()
#=> A: 10
#=> B: 10
#=> A: 9
#=> B: 9
```

LUMEN COMPILER

Frontend

Middle Tier

Backend/Codegen

DOCKYARD

Accepts source files in Erlang

A Mix task is used to produce source for Lumen

Supports richer diagnostics than erlc

LUMEN COMPILER

Frontend

Middle Tier

Backend/Codegen

DOCKYARD

AST lowered to EIR

Semantic analysis during lowering

EIR based on Thorin, a graph-based higher-order IR

LUMEN COMPILER

Frontend

Middle Tier

Backend/Codegen

DOCKYARD

CPS-like, without the disadvantages

Easily transformed to SSA

Solid foundation for high-level optimizations

LUMEN COMPILER

Frontend

Middle Tier: EIR

Backend/Codegen

DOCKYARD

LUMEN COMPILER

Frontend

Middle Tier

Backend/Codegen

- Lowers from EIR to LLVM IR
- Generates object files or executable
- Performs linking/link-time optimization

DOCKYARD

PROJECT STATUS

LUMEN

DOCKYARD

CURRENT STATUS

There is an interpreter that can be used for experimentation

Codegen backend is basically complete

Next release is waiting on a few PRs

ROADMAP

- Better type information
- Auto-generated JS/DOM bindings
- In-browser debugging
- Support for a wider array of targets