Haskell on the JVM with Eta

JVM problem

1. Functional programming is the most practical way to program
2. Every JVM language causes problems with this goal

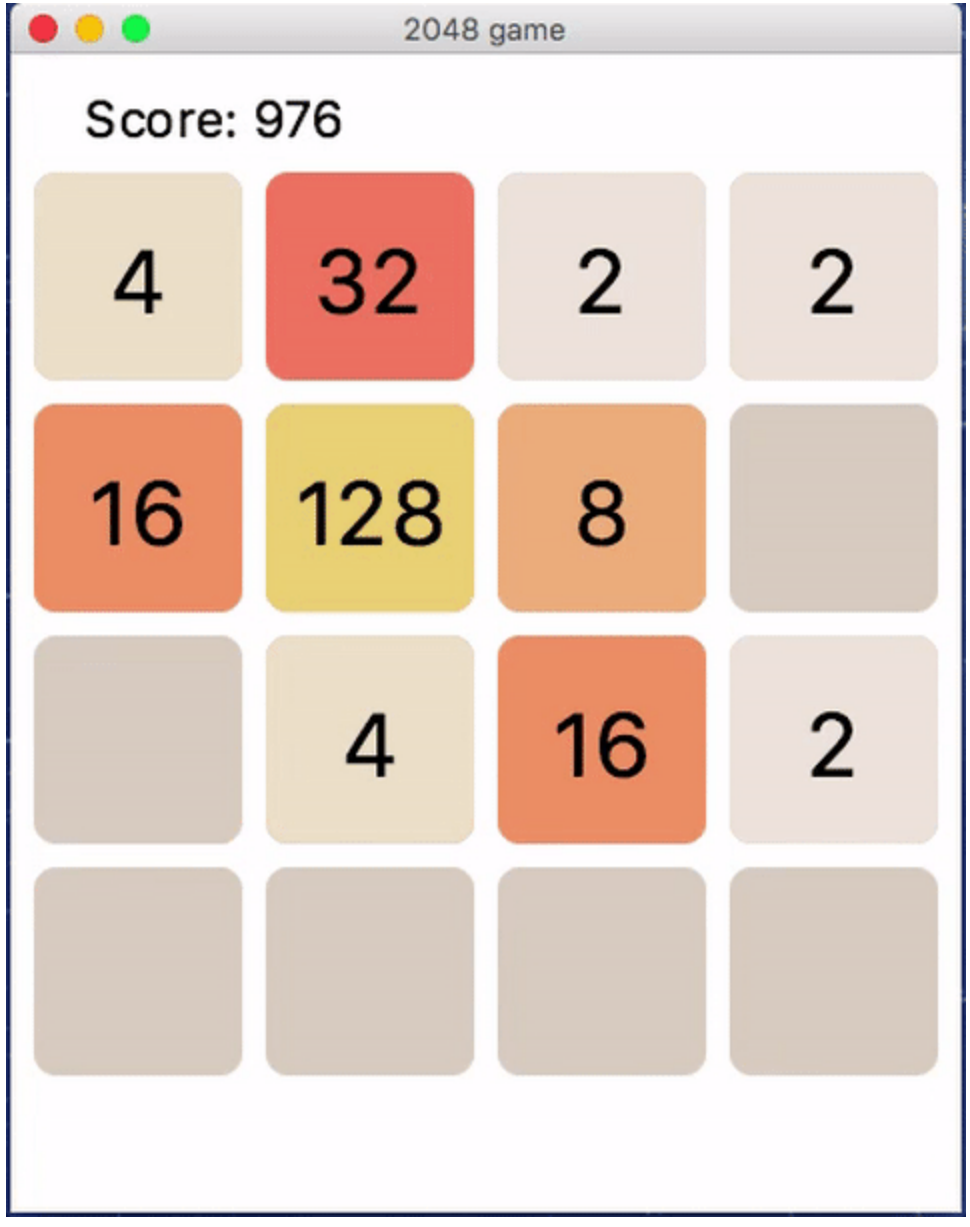
Conclusion: the JVM is not a practical option

GHC

1. Haskell
2. Libraries and tools (e.g. lens)
3. Runtime

Eta

- Formerly GHCVM
- Rahul Muttineni
- Haskell Summer of Code
- TypeLead (3 people)



Eta, Yampa & JavaFX

Eta compiler

Is a fork of GHC:

1. JVM backend
2. Java runtime

Eta does for JVM what GHCJS does for JavaScript

GHC Pipeline

Language	Flag
Haskell	`-ddump-parsed`
Core	`-ddump-simpl`
STG	`-ddump-stg`
Cmm	`-ddump-cmm`
Native code	`-ddump-asm`

Eta Pipeline

Language	Flag
Haskell	`-ddump-parsed`
Core	`-ddump-simpl`
STG	`-ddump-stg`
Java bytecode	`-ddump-cg-trace`

Representations

Language	Feature
Haskell	Convenient
Core	Explicit evaluation
STG	Explicit allocation

Spineless Tagless G-machine (STG)

> The G-machine is an abstract architecture for evaluating functional-language programs by programmed graph reduction.

Spineless: function application points to all arguments

Tagless: don't tag graph nodes with thunk/closure/constructor

Graph reduction: non-strict evaluation

1. Registers
2. Stack
3. Heap

```
main = print (1 + 2)
```

```
==================== Tidy Core ====================

-- RHS size: {terms: 6, types: 2, coercions: 0}
main :: IO ()
main =
  print
    @ Integer
    GHC.Show.$fShowInteger
    (+ @ Integer GHC.Num.$fNumInteger 1 2)

-- RHS size: {terms: 2, types: 1, coercions: 0}
:Main.main :: IO ()
:Main.main = GHC.TopHandler.runMainIO @ () main
```

```
==================== STG syntax: ====================
sat_s10r =
    \u srt:SRT:[rsY :-> $fNumInteger] []
        let { sat_s10q = NO_CCS S#! [2#]; } in
        let { sat_s10p = NO_CCS S#! [1#];
        } in  + $fNumInteger sat_s10p sat_s10q;

main =
    \u srt:SRT:[0B :-> print, rur :-> $fShowInteger,
                s10r :-> sat_s10r] []
        print $fShowInteger sat_s10r;

main =
    \u srt:SRT:[01E :-> runMainIO, rn5 :-> main] []
        runMainIO main;
```

STGi

```java
package eta;

import eta.runtime.Rts;
import eta.runtime.RtsConfig;
import main.Main;

public class main {
  public static void main(String[] args) {
    RtsConfig config = RtsConfig.getDefault();
    config.rtsHsMain = true;
    Rts.hsMain(args, Main.DZCmain_closure, config);
  }
}
```

```
package main;

import eta.runtime.stg.StgClosure;

public class Main {
  public static StgClosure zdLs22Lsat_closure =
    new zdLs22Lsat();

  public static StgClosure main_closure =
    new main();

  public static StgClosure DZCmain_closure =
    new DZCmain();
}
```

```java
package main;

import base.ghc.TopHandler;
import eta.runtime.apply.Apply;
import eta.runtime.stg.StgContext;
import eta.runtime.thunk.StgIndStatic;

public final class DZCmain extends StgIndStatic {
  public void thunkEnter(StgContext context) {
    context.R(1, TopHandler.runMainIO_closure);
    context.R(2, Main.main_closure);
    Apply.ap_p_fast.enter(context);
  }
}
```

```
class ArgumentStack extends AbstractArgumentStack {
    public ObjectArrayList objects;
    public IntArrayList ints;
    public LongArrayList longs;
    public FloatArrayList floats;
    public DoubleArrayList doubles;
}

class StgContext {
    public ArgumentStack argStack =
      new ArgumentStack();

    public StgTSO currentTSO;
    public Capability myCapability;
    public ReturnCode ret;
}
```

```
package main;

import base.ghc.Show;
import base.system.IO;
import eta.runtime.apply.Apply;
import eta.runtime.stg.StgContext;
import eta.runtime.thunk.StgIndStatic;

public final class main extends StgIndStatic {
  public void thunkEnter(StgContext context) {
    context.R(1, IO.print_closure);
    context.R(2, Show.zdfShowInteger_closure);
    context.R(3, Main.zdLs22Lsat_closure);
    Apply.ap_pp_fast.enter(context);
  }
}
```

```java
package main;

import base.ghc.Num;
import eta.runtime.apply.ApPP;
import eta.runtime.stg.StgContext;
import eta.runtime.thunk.StgIndStatic;
import integer.ghc.integer.Type;

public class zdLs22Lsat extends StgIndStatic {
  public void thunkEnter(StgContext context) {
    // continuation to a "generic apply function"
    Type.SzhD x = new Type.SzhD(2);
    Type.SzhD y = new Type.SzhD(1);
    context.currentTSO.spPush(new ApPP(y, x));

    // dictionary
    context.R(2, Num.zdfNumInteger_closure);
    Num.zp_closure.enter(context);
  }
}
```

Java calls

```
{-# LANGUAGE DataKinds #-}
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE MagicHash #-}
{-# LANGUAGE TypeFamilies #-}
{-# LANGUAGE TypeOperators #-}
```

```
import Java
import Java.IO
import Control.Concurrent (threadDelay)
import Control.Monad (when)

data {-# CLASS "java.io.FileInputStream" #-}
    FileInputStream
  = FileInputStream (Object# FileInputStream)
    deriving Class

type instance Inherits FileInputStream = '[InputStream]

foreign import java unsafe "@new"
  newFileInputStream :: File -> IO FileInputStream

foreign import java unsafe "@new"
  newFile :: String -> IO File
```

```
data {-# CLASS "javax.sound.midi.Sequencer" #-}
    Sequencer
  = Sequencer (Object# Sequencer)

foreign import java unsafe
  "@static javax.sound.midi.MidiSystem.getSequencer"
  getSequencer :: IO Sequencer

foreign import java unsafe "@interface"
  open :: Sequencer -> IO ()

foreign import java unsafe "@interface"
  setSequence
  :: (a <: InputStream) => Sequencer -> a -> IO ()

foreign import java unsafe "@interface"
  start :: Sequencer -> IO ()

foreign import java unsafe "@interface"
  isRunning :: Sequencer -> IO Bool
```

```haskell
whileM_ :: (Monad m) => m Bool -> m a -> m ()
whileM_ cond ma = do
  b <- cond
  when b $ ma *> whileM_ cond ma

main :: IO ()
main = do
  sequencer <- getSequencer
  open sequencer

  file <- newFile "bolero_of_fire.mid"
  is <- newFileInputStream file

  setSequence sequencer is
  start sequencer

  whileM_ (isRunning sequencer) $
    threadDelay (1000 * 1000)
```

Other direction

```haskell
{-# LANGUAGE MagicHash #-}

module Export where

import Java
import System.IO

data {-# CLASS "xyz.profunctor.Hello" #-} Hello
  = Hello (Object# Hello)
  deriving Class

world = do
  io $ putStrLn "Hello" *> hFlush stdout
  return 100

foreign export java "world" world
  :: Java Hello Int
```

```
scala> val x = (new xyz.profunctor.Hello()).world()
Hello
x: Int = 100
```

Etlas

- Cabal 1.24
- GHC libraries have C code, we have 32 patches
- 221 packages listed as "supported"

Lens

```haskell
import Control.Lens
import Data.Char

main :: IO ()
main = do
  print $ ("hello","world")^._2          -- "world"
  print $ set _2 42 ("hello","world")    -- ("hello",42)
  print $ ("hey",("world","!"))^._2._1   -- "world"
  print $ "hello"^.to length             -- 5
  print $ view _2 (10,20)                -- 20
  print $ over mapped succ [1,2,3]       -- [2,3,4]
  print $ both *~ 2 $ (1,2)              -- (2,4)
```

Performance

Somewhere around 4-10x GHC

```
--- a/rts/src/eta/runtime/stg/Capability.java
+++ b/rts/src/eta/runtime/stg/Capability.java
- Stack<StackFrame> stack = tso.stack;
- stopIndex = stack.size() - stack.search(stopHere);
+ stopIndex = tso.stack.lastIndexOf(stopHere);
- Stack<StackFrame> oldStack = tso.stack;
--- a/rts/src/eta/runtime/stg/StgStopThread.java
+++ b/rts/src/eta/runtime/stg/StgStopThread.java
- Stack<StackFrame> stack = new Stack<StackFrame>();
+ LinkedList<StackFrame> stack = new LinkedList<StackFram
--- a/rts/src/eta/runtime/stg/StgTSO.java
+++ b/rts/src/eta/runtime/stg/StgTSO.java
- public Stack<StackFrame> stack = new Stack<StackFrame>(
+ public LinkedList<StackFrame> stack = new LinkedList<St
```

Cool projects

- wai-servlet
- Kinesis
- Kafka
- Spark

TODO

- Bindings to Java libraries
- More Hackage porting
- Performance
  - RTS
  - Generated code

- GHCVM Proposal Rahul Muttineni
- Bringing the Power of Haskell to the JVM Rahul Muttineni
- GHC (STG, Cmm, asm) illustrated for hardware persons Takenobu Tani
- STGi David Luposchainsky
- Functional and low-level: watching the STG execute David Luposchainsky
- I know kung fu: learning STG by example Max Bolingbroke