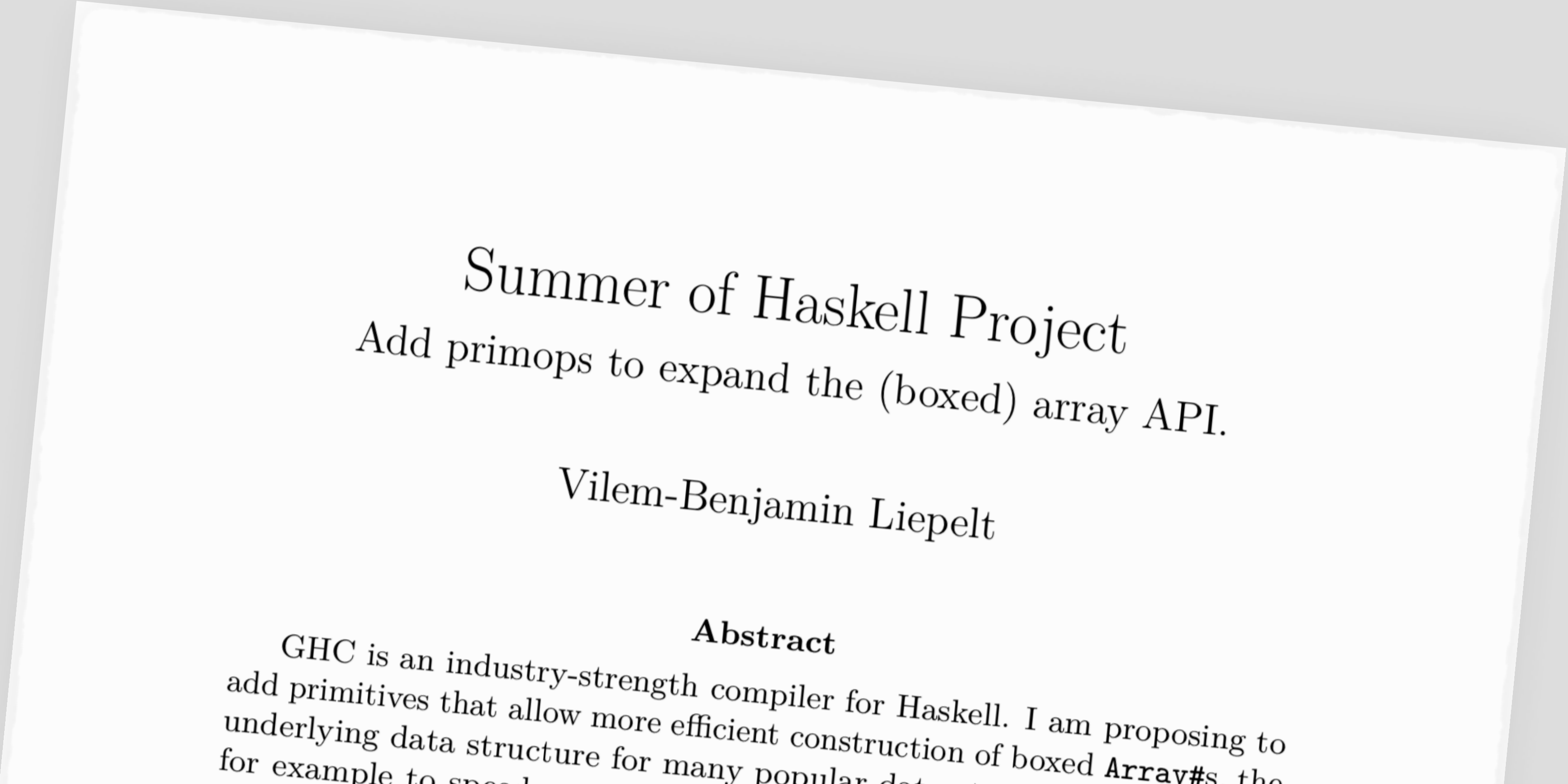
Summer of ~~Insanity~~ Haskell 2020



Summer of Haskell Project

Add primops to expand the (boxed) array API.

Vilem-Benjamin Liepelt

Abstract

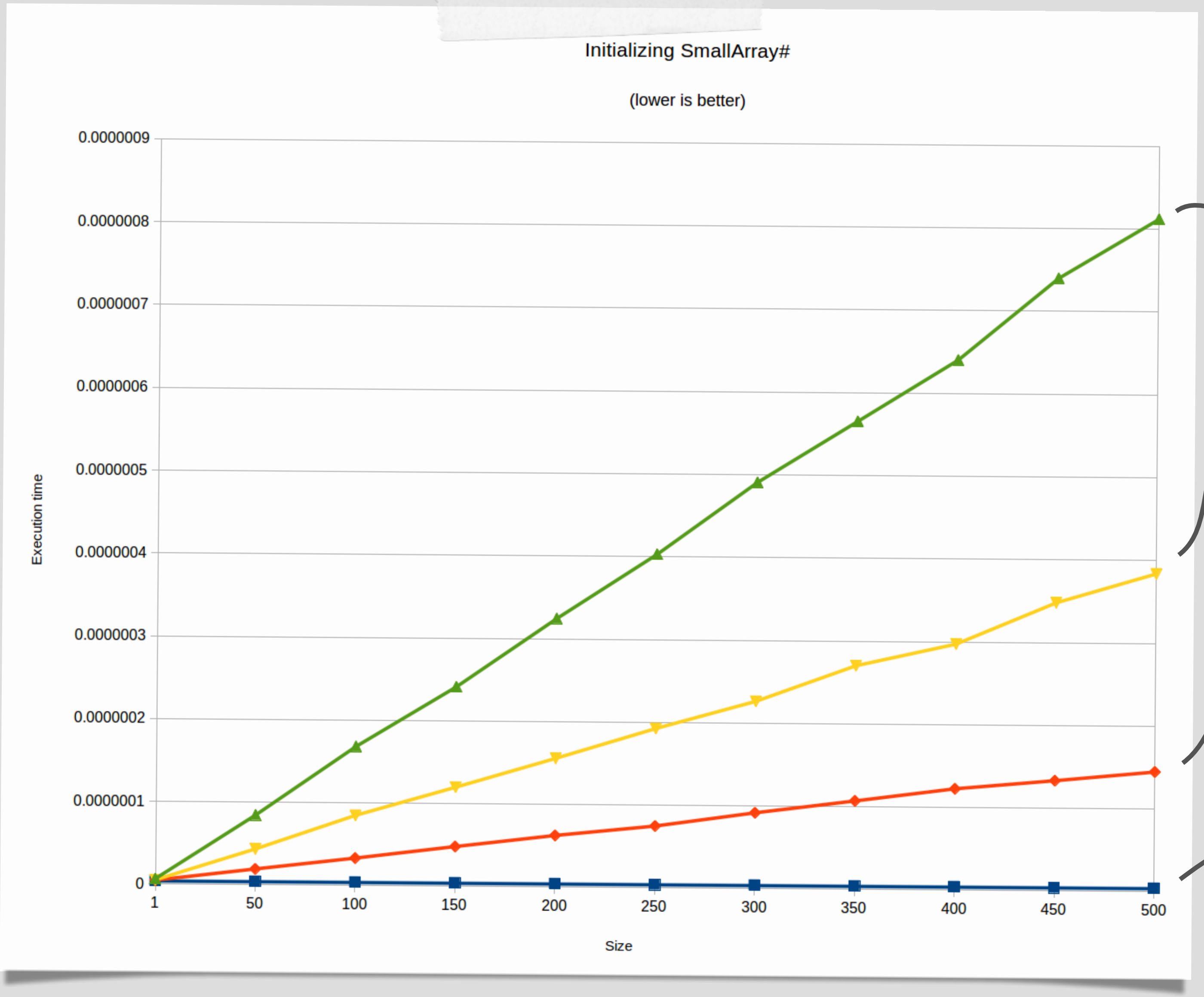
GHC is an industry-strength compiler for Haskell. I am proposing to add primitives that allow more efficient construction of boxed `Array#`s, the underlying data structure for many popular data

for example to

Contributions

1. Over **2×** faster array initialization.
2. First **VARIADIC** GHC primop.
3. Optimization for **STATIC ALLOCATION** (wip).

Knowledge is speed



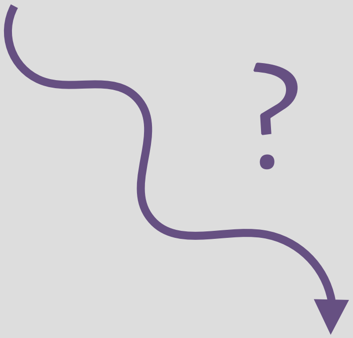
fully dynamic

manually unrolled
(static length)

new primop
(static length)

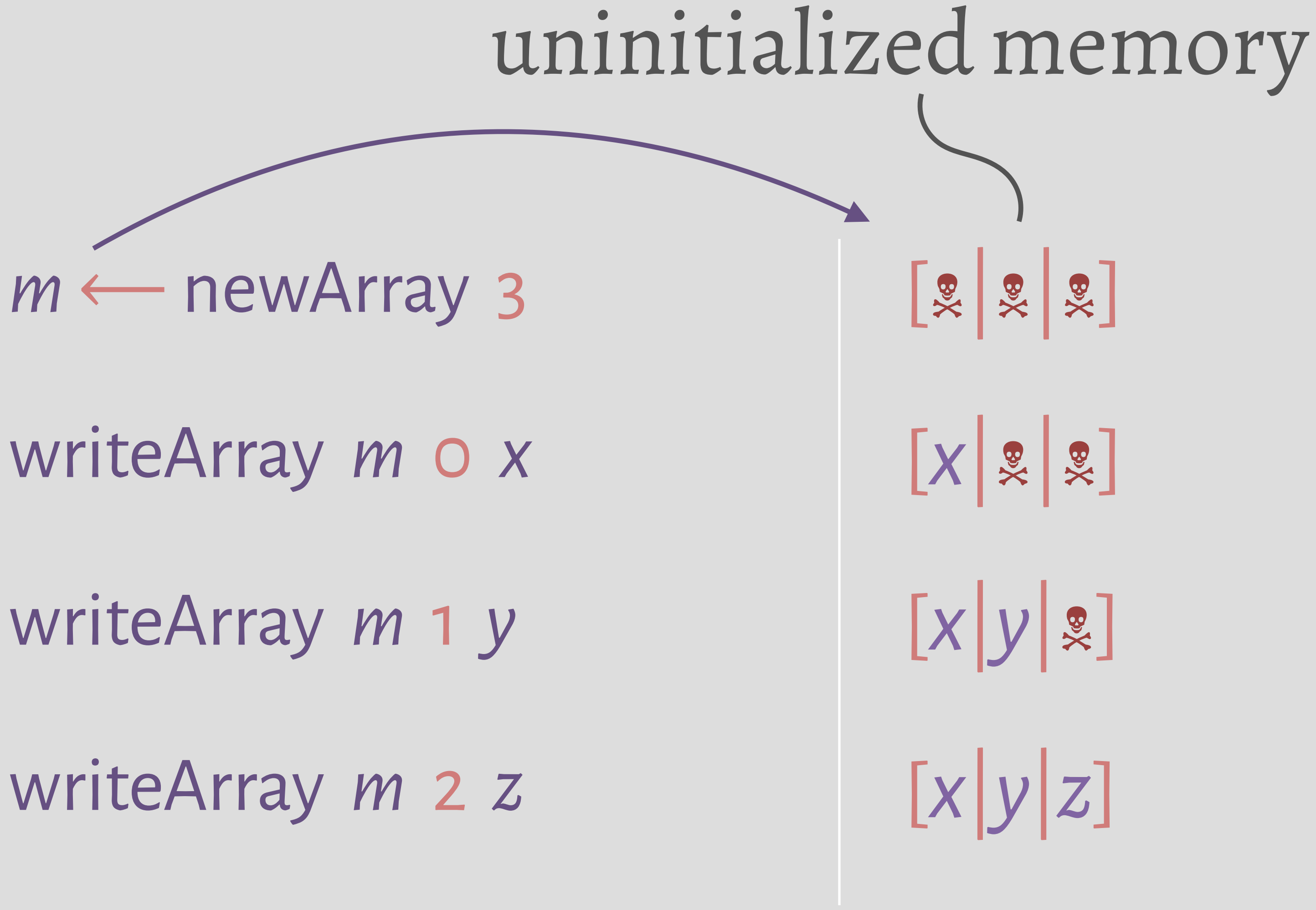
wip optimization
(static contents)

$x, y, z :: \text{Int}$



$[x|y|z] :: \text{Array}$

If Haskell were C



uninitialized memory

```
m ← newArray 3          [☠|☠|☠]

writeArray m 0 x        [x|☠|☠]

writeArray m 1 y        [x|y|☠]

writeArray m 2 z        [x|y|z]
```

If Haskell were C

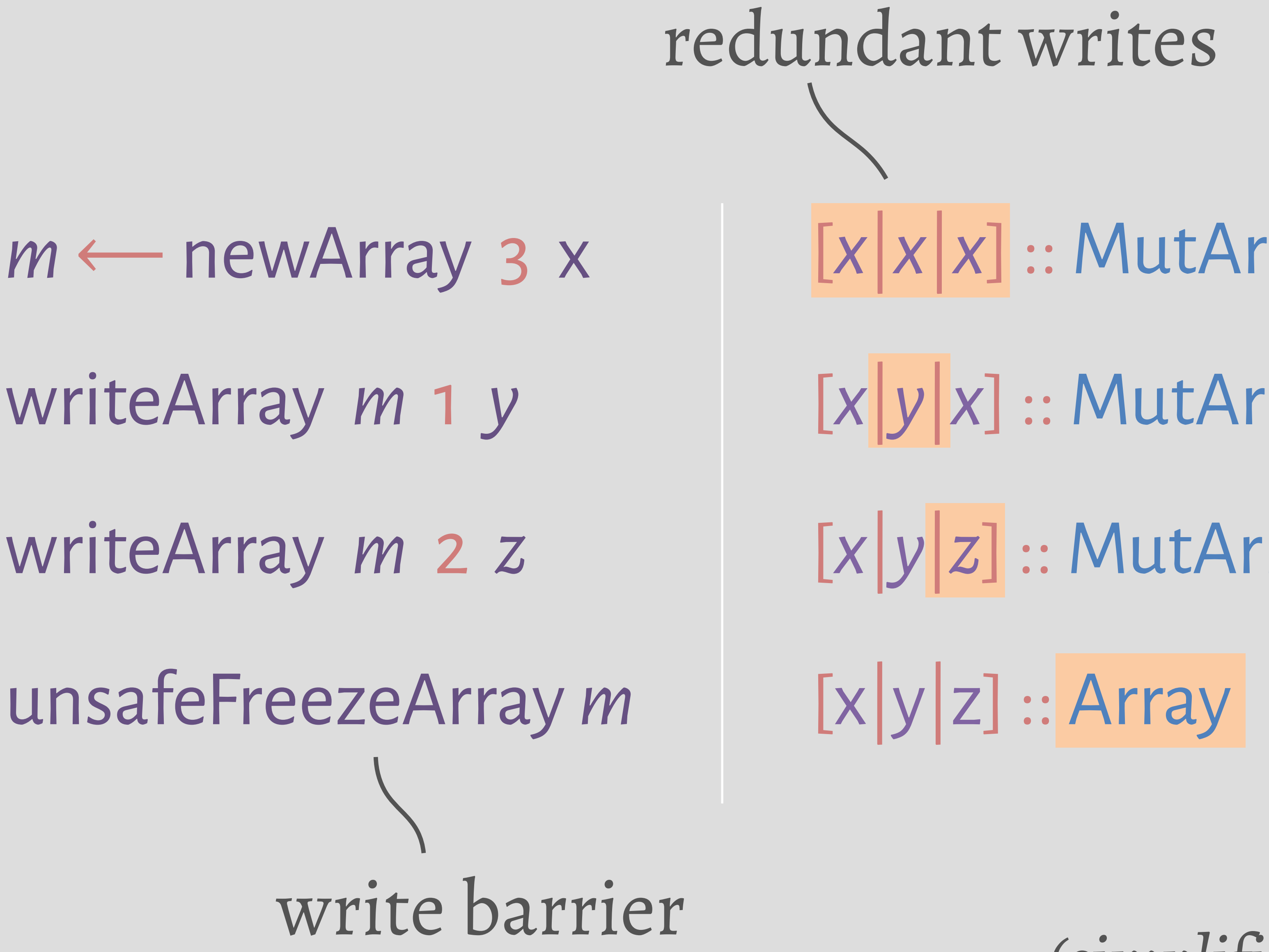
uninitialized memory

```
m ← ne                [☠|☠|☠]

writeAr               [x|☠|☠]

writeAr               [x|y|☠]

writeArray m 2 z      [x|y|z]
```



redundant writes

```
m ← newArray 3 x              [x|x|x] :: MutArray
writeArray m 1 y              [x|y|x] :: MutArray
writeArray m 2 z              [x|y|z] :: MutArray
unsafeFreezeArray m           [x|y|z] :: Array
```

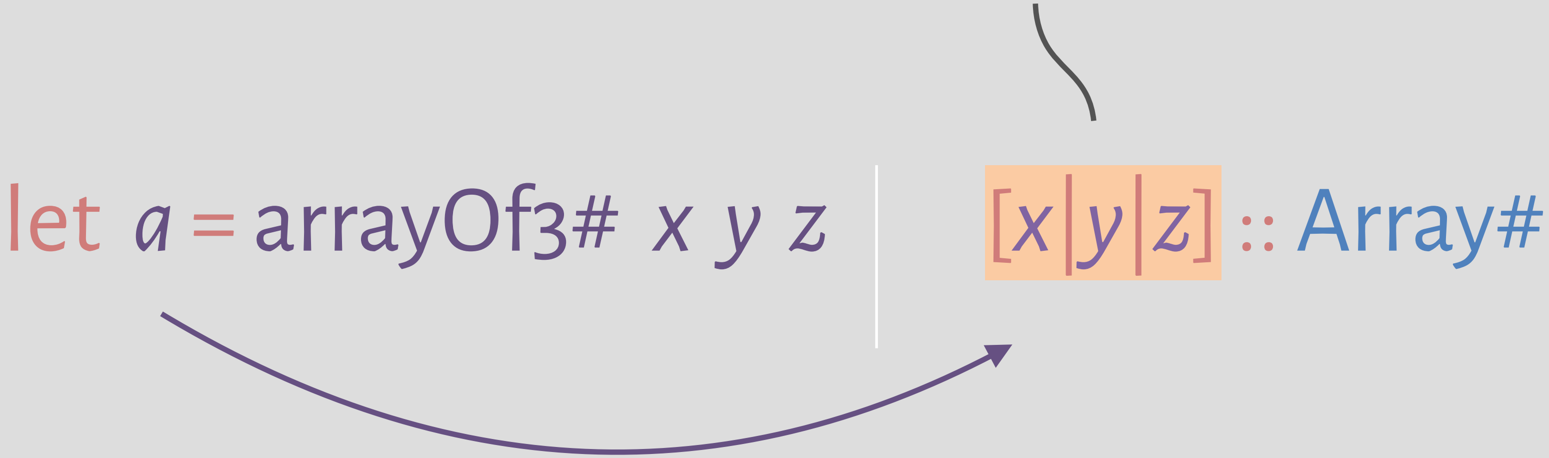
write barrier

(simplification)

Idea: build array atomically

```
arrayOf3#
  :: a → a → a → Array# a
```

initialize everything in a single pass



```
let a = arrayOf3# x y z
```

[x|y|z] :: Array#

```
arrayOf0# :: Array# a
arrayOf1# :: a → Array# a
arrayOf2# :: a → a → Array# a
arrayOf3# :: a → a → a → Array
arrayOf4# :: a → a → a → a →
arrayOf5# :: a → a → a → a →
arrayOf6# :: a → a → a → a →
arrayOf7# :: a → a → a → a →
```

We want a variadic primop!

$$\mathsf{arrayOf\#} :: (a \longrightarrow)^{*}\ \mathsf{Array\#}\ a$$

if Haskell had regex types roflmao

Code gen

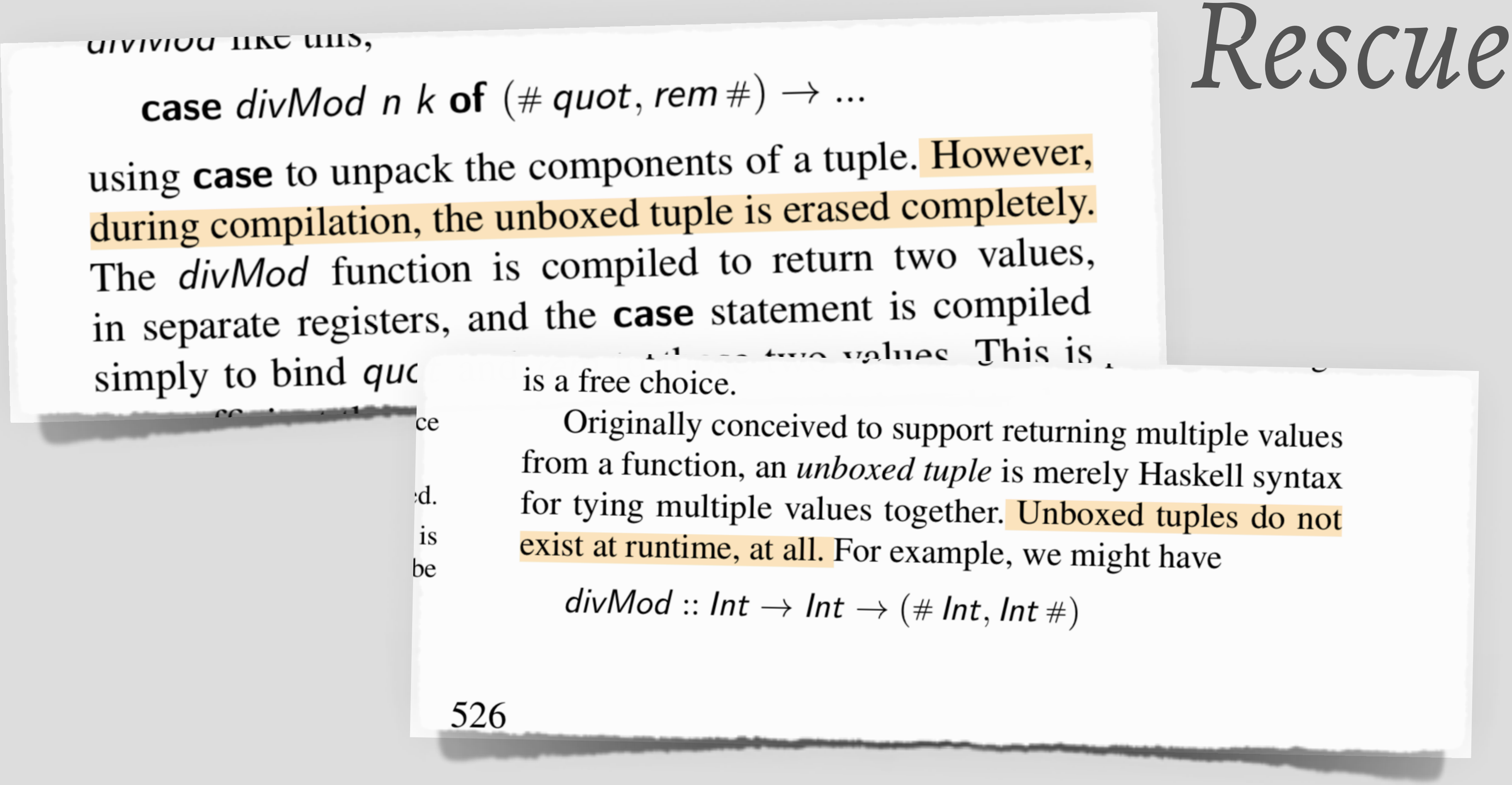
```
emitPrimOp
      :: DynFlags
      -> PrimOp
      -> [CmmExpr]
      -> PrimopCmmEmit
```

primop

args

(GHC.StgToCmm.Prim)

Unboxed Tuples to the Rescue



divMod like this,

$$\textbf{case}\ \mathit{divMod}\ n\ k\ \textbf{of}\ (\#\ \mathit{quot}, \mathit{rem}\ \#) \rightarrow \ldots$$

using **case** to unpack the components of a tuple. However, during compilation, the unboxed tuple is erased completely. The *divMod* function is compiled to return two values, in separate registers, and the **case** statement is compiled simply to bind *quo*

This is

ce

d.

is

be

is a free choice.

Originally conceived to support returning multiple values from a function, an *unboxed tuple* is merely Haskell syntax for tying multiple values together. Unboxed tuples do not exist at runtime, at all. For example, we might have

$$\mathit{divMod} :: \mathit{Int} \rightarrow \mathit{Int} \rightarrow (\#\ \mathit{Int}, \mathit{Int}\ \#)$$

526

(Eisenberg, Peyton Jones 2017)

$$a_1 = \text{arrayOf\#}\ (\#\ 0,\ 1\ \#)$$

$$a_2 = \text{arrayOf\#}\ (\#\ (\#\ \#)\ 0,\ (\#\ 1\ \#)\ \#)$$

a_1 is equal to a_2 thanks to GHC.Stg.Unarise

My dreams...

```
arrayOf#
  :: (# a, ... #) → Array# a
```

arbitrarily nested, homogeneous
unboxed tuple

… reality

```
arrayOf#
  :: ∀r (a :: TYPE r) b.  a ⟶ Array# b
```

```
                                        :: rat
arrayOf#
  :: ∀r (a :: TYPE r). a → Array# b
```

UNSOUND

Oh, well, just another footgun. It is easy to define a safe wrapper.

Oh, well, just another footgun. It is easy to define a safe wrapper.

ALSO: DOCUMENTATION!

CODE and further **DISCUSSION** at

GHC merge request 3571.

I'm on **TWITTER** (*@buggymcbugfix*)

and I don't not have a **BLOG** here:

github.com/buggymcbugfix/

not-not-a-blog.

Special thanks to

Google for funding Summer of Code

My mentors

- **Andreas Klebinger** *(Well-Typed)*
- **Andrew Thaddeus Martin** *(Layer3)*
- **Chessai** *(Mercury)*

My employer **Sqream**

My PhD adviser **Dominic Orchard**

My school, **University of Kent**

My **Family**, especially my wife **Elis**