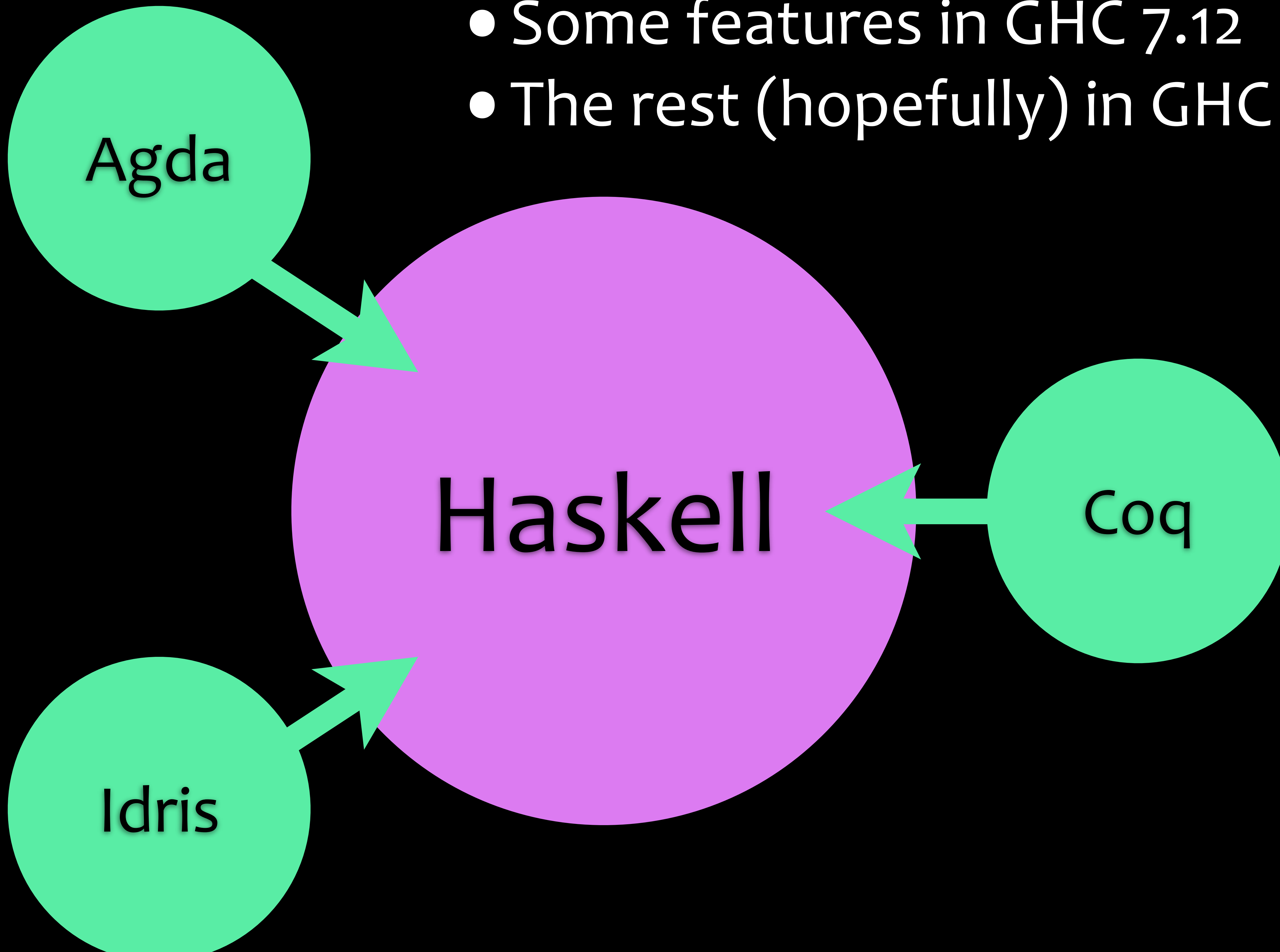
A Practical Introduction to Haskell GADTs

Richard A. Eisenberg
University of Pennsylvania
eir@cis.upenn.edu

Friday, 22 May, 2015
LambdaConf 2015
Boulder, CO, USA

Who am I?

- Some features in GHC 7.12
- The rest (hopefully) in GHC 7.14



And you are?

GADTs

Promoted datatypes

Type families

Singletons

Coq/Agda/Idris

Goals of this talk:

- Understand the mechanics of GADTs
- Work with existing code using GADTs
- Write simple functions using GADTs
- Understand why GADTs help you code

Non-goal:

- Know how to design a GADT-ified application architecture from scratch

GADT =

Generalized Algebraic

DataType

GADTs allow more compile-time checks than ADTs...

...just like types allow more compile-time checks than

Clojure	JavaScript	Perl
Ruby	PHP	Racket
Python	Lisp	...

Glambda: a simply-typed lambda calculus implemented with GADTs

"Greenspun's Tenth Rule of Programming: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.”

-- Philip Greenspun

Other Applications

- Type-safe database access
- Verifying data structures
- Generic programming
- “Tagless” programming

Very simple GADT example

Off to emacs…

STy

```
data STy ty where
  SInt    :: STy Int
  SBool   :: STy Bool
  SMaybe  :: STy ty' -> STy (Maybe ty')

zero :: STy ty -> ty
zero SInt       = 0
zero SBool      = False
zero (SMaybe _) = Nothing
```

GHC: I know `ty ~ Int`

GHC: I know `ty ~ Bool`

GHC: I know `ty ~ Maybe ...`

STy

```
data STy ty
  = (ty ~ Int) => SInt
  | (ty ~ Bool) => SBool
  | forall ty'. (ty ~ Maybe ty') => SMaybe
```

(~) = type equality

GHC: I know
ty ~ Int

```
zero :: STy ty -> ty
zero SInt       = 0
zero SBool      = False
zero (SMaybe _) = Nothing
```

GHC: I know
ty ~ Bool

GHC: I know
ty ~ Maybe ...

Pattern-matching a term reveals type information

Exercise 1

Extend `zero`.

See
`https://github.com/goldfirere/glambda/`

GADT Type Inference

Off to emacs…

GADT pattern matches *need* type signatures

`‘t’ is untouchable`

means

“add a type signature”

ScopedTypeVariables

```haskell
foo :: a -> ...
foo x = ...
  where fhelper :: a
        fhelper = ...
```

fhelper and x have *different* types

```haskell
bar :: forall a. a -> ...
bar x = ...
  where bhelper :: a
        bhelper = ...
```

bhelper and x have *the same* type

GHC bug #3927:
Pattern warnings + GADTs = Inadequate

Off to emacs…

GHC bug #3927: Pattern warnings + GADTs = Inadequate

"GADTs meet their match"
by Georgios Karachalias, Tom Schrijvers, Dimitrios Vytiniotis, and Simon Peyton Jones
ICFP 2015

Fix expected in 7.12

Heterogeneous lists

Off to emacs…

Exercise 2

Write `get`.

See
`https://github.com/goldfirere/glambda/`

Break time!

Visit

`https://github.com/goldfirere/glambda`

for instructions.

Off to `glam`...

Exercise 3

Write `eval`, `cond`, and `apply`.

See
`https://github.com/goldfirere/glambda/`

Why I like GADTs

de Bruijn indices

from *Types and Programming Languages* (Pierce, 2002)

Shifting:

$$
\begin{aligned}
\uparrow^d_c(\mathtt{k}) &= \begin{cases} \mathtt{k} & \text{if } \mathtt{k} < c \\ \mathtt{k} + d & \text{if } \mathtt{k} \geq c \end{cases} \\
\uparrow^d_c(\lambda.\mathtt{t}_1) &= \lambda.\uparrow^d_{c+1}(\mathtt{t}_1) \\
\uparrow^d_c(\mathtt{t}_1\ \mathtt{t}_2) &= \uparrow^d_c(\mathtt{t}_1)\ \uparrow^d_c(\mathtt{t}_2)
\end{aligned}
$$

Substitution:

$$
\begin{aligned}
[\mathtt{j} \mapsto \mathtt{s}]\mathtt{k} &= \begin{cases} \mathtt{s} & \text{if } \mathtt{k} = \mathtt{j} \\ \mathtt{k} & \text{otherwise} \end{cases} \\
[\mathtt{j} \mapsto \mathtt{s}](\lambda.\mathtt{t}_1) &= \lambda.[\mathtt{j} + 1 \mapsto \uparrow^1_0(\mathtt{s})]\mathtt{t}_1 \\
[\mathtt{j} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) &= [\mathtt{j} \mapsto \mathtt{s}]\mathtt{t}_1\ [\mathtt{j} \mapsto \mathtt{s}]\mathtt{t}_2
\end{aligned}
$$

de Bruijn indices

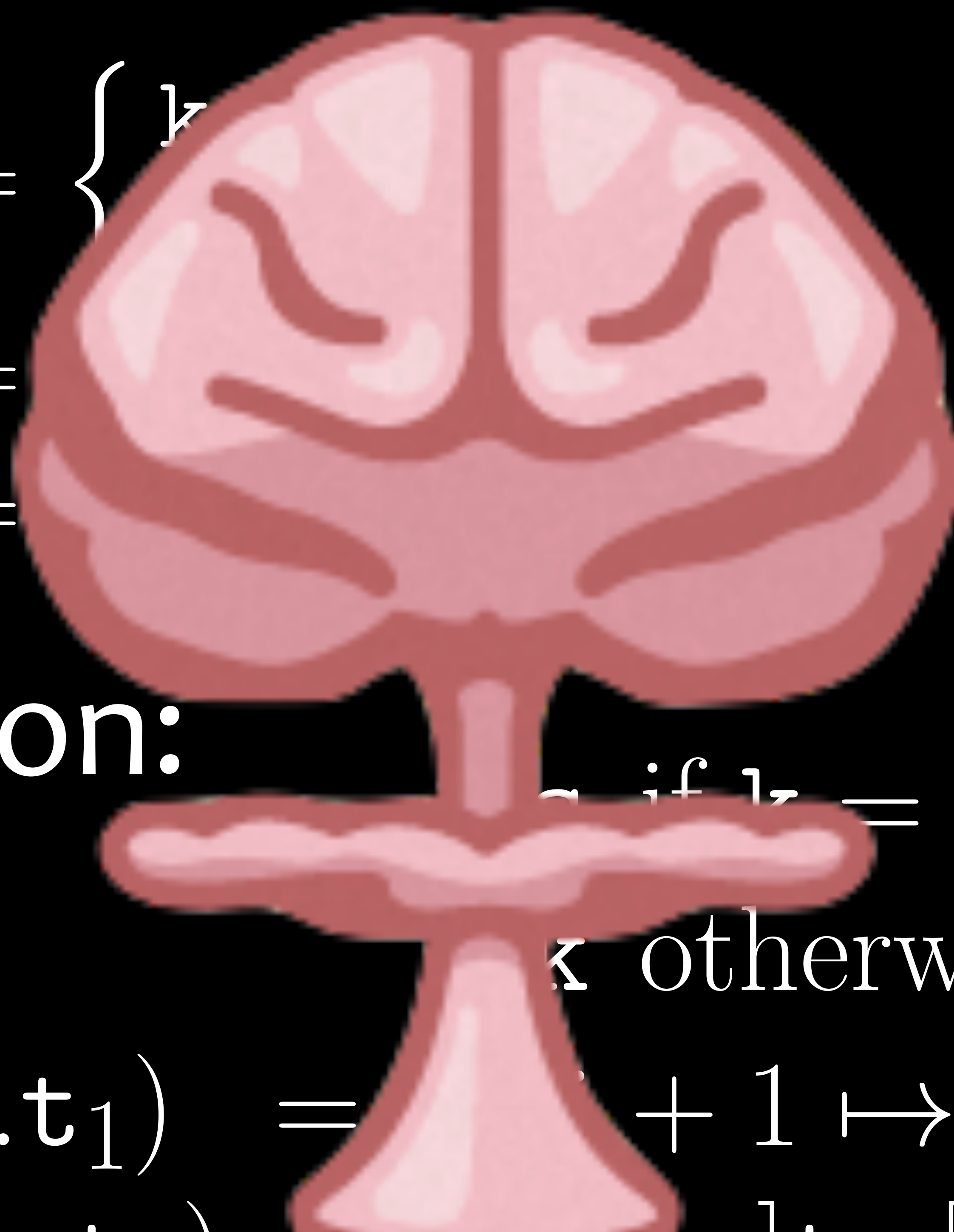
from *Types and Programming Languages* (Pierce, 2002)

Shifting:

$\uparrow^d_c(\mathtt{k}) \quad = \Big\{ \mathtt{k}$

$\uparrow^d_c(\lambda.\mathtt{t}_1) \quad =$

$\uparrow^d_c(\mathtt{t}_1\ \mathtt{t}_2) =$



Substitution:

$[\mathtt{j} \mapsto \mathtt{s}]\mathtt{k}$ $\quad$ ... if $\mathtt{k} = \mathtt{j}$

$\mathtt{k}$ otherwise

$[\mathtt{j} \mapsto \mathtt{s}](\lambda.\mathtt{t}_1) \quad = \quad \ldots + 1 \mapsto \uparrow^1_0(\mathtt{s})]\mathtt{t}_1$

$[\mathtt{j} \mapsto \mathtt{s}](\mathtt{t}_1\ \mathtt{t}_2) = \ldots \mathtt{s}]\mathtt{t}_1\ [\mathtt{j} \mapsto \mathtt{s}]\mathtt{t}_2$

Time machines

... don’t exist

Type-checking glambda

```
check :: ( MonadError Doc m
         , MonadReader Globals m )
      => UExp    -- “unchecked” exp
      -> (forall t.
          STy t -> Exp '[] t -> m r)
      -> m r
```

works for *any* type t

Further Reading

- “Generalized Algebraic Data Types in Haskell”, by Anton Dergunov, in the Monad.Reader Issue 22, August 2013
- “Dependently Typed Programming with Singletons”, by Richard A. Eisenberg and Stephanie Weirich, in Haskell Symposium 2012
- “Why GADTs matter for performance”, by Yaron Minsky, at https://blogs.janestreet.com/why-gadts-matter-for-performance/
- glambda source code (and upcoming tutorial!)