Monads are Burritos

Michael Easter
http://codetojoy.blogspot.com

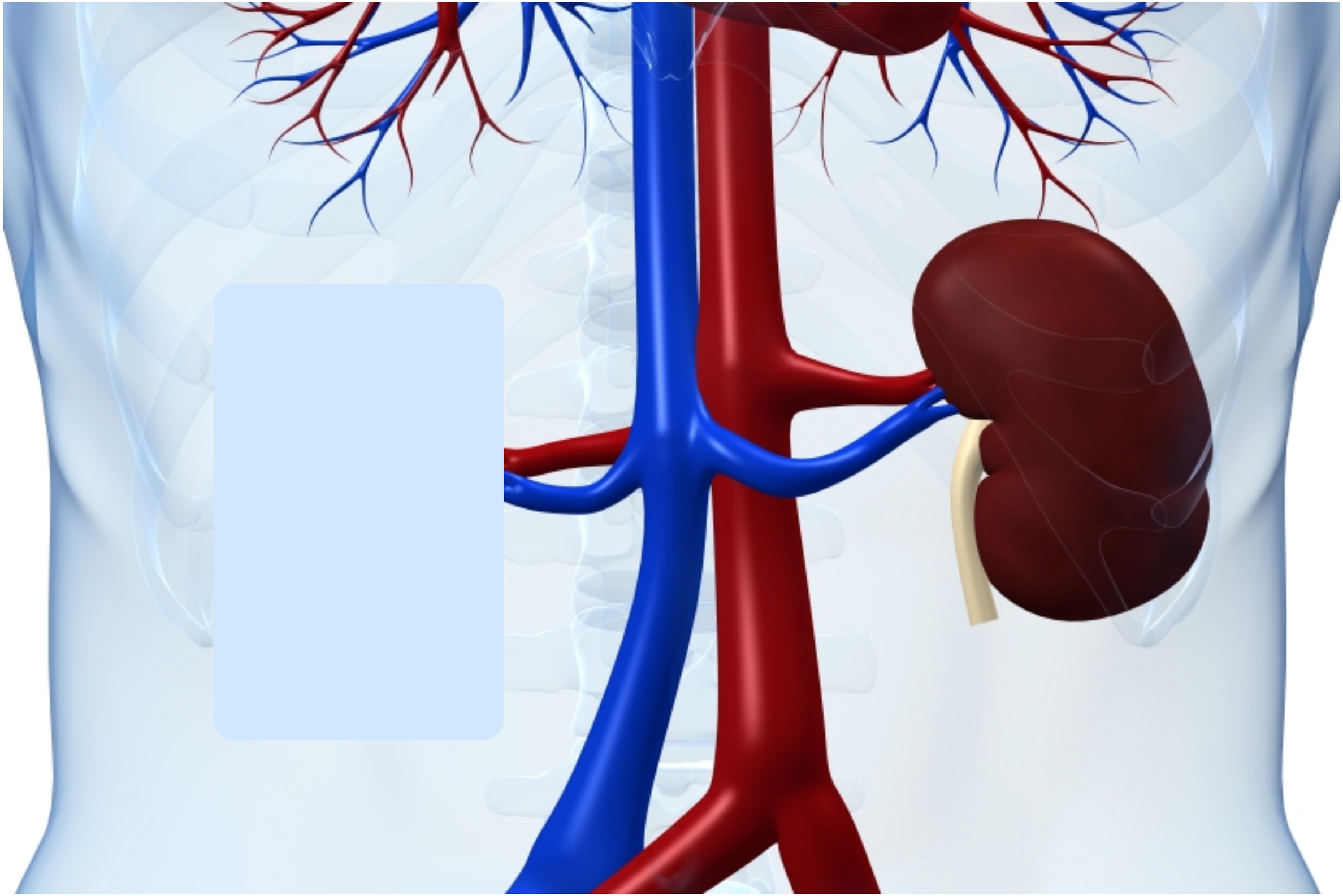
There is no Burrito: we all must find our own.

The Promised Land is a journey, just as it was for objects, recursion, etc

Our Master of Ceremonies
is
http://demetrimartin.com

A cerebral comic with
jokes such as:

What is the smartest thing anyone has said that starts with "Dude" ?

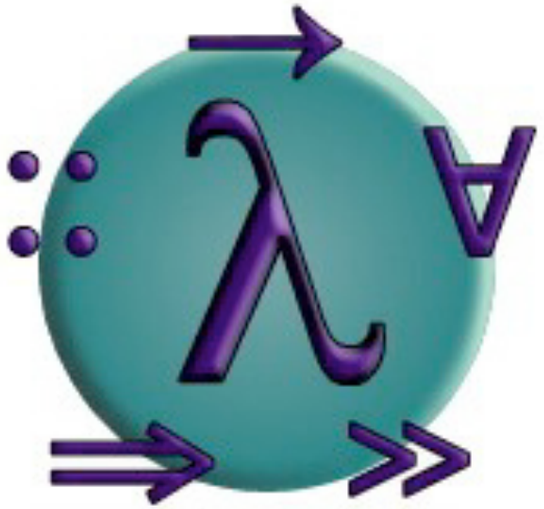


“Dude, we removed a kidney and you’re going to be fine.”

“*Dude, these are isotopes!*” -- excellent but we can top that

Haskell

- pure functional
- strong, static typing
- modular, and not OO
- lazy evaluation



Iterative, location-based programming is a scourge

```
tuple a b = do {
        x <- a ;
        y <- b ;
        return (x,y)
    }
```

This was presented as pseudo-code of iterative programming, but it is in fact valid Haskell.

Functions in mathematics

f : N -> R

$f(x) = |\sqrt{x}|$

$f(100) = 10$

$f(3.14) = ?$

$f(-2) = ?$

Reviewing the notion of **domain** and **range** for math functions

Type Signatures in Haskell

f : N -> R in math

in Haskell:

f :: a -> b (eg String -> Integer)

g :: a -> a -> b (eg String String -> Integer)

h :: a -> (a -> b) -> b (eg String f -> Integer)

Maybe type

```
data Maybe a = Nothing | Just a

Just “Lambda Lounge” :: Maybe String

Just 10 :: Maybe Integer
```

Maybe is a wrapper type. ‘a’ here is a type variable

Monad:

1. Type constructor **m** (eg Maybe)

2. injection function: a -> **m** a

3. chain function:

m a -> (a -> **m** b) -> **m** b

Due to variance within this structure, there are many instances of monads in Haskell.

For a monad **m**, in Haskell:

1. injector is **return** = a -> **m** a

2. chain is called bind. Symbol is **>>=**

m a -> (a -> **m** b) -> **m** b

Monad	**use**	**support**
Maybe	short-circuit	n/a
Logger	state	runLogger
IO	impure IO	putStrLn
STM	concurrency	atomically

record “any” =
Logger ((), [“any”])

ezRegex “abc” =
Logger (“abc”, [])

return (‘.’ ++ “abc”) =
Logger (“.abc”, [])

The logger example inspired by Real World Haskell.

record “any” =
Logger ((), [“any”])

((), [“any”])

(“abc”, [])

(“abc”, [“any”])

globToRegex “abc” =
Logger (“abc”, [])

return (‘.’ ++ “abc”) =
Logger (“.abc”, [])

record “any” =
Logger ((), [“any”])

((), [“any”])

(“abc”, [])

(“abc”, [“any”])

(“.abc”, [])

(“.abc” , [“any”])

globToRegex “abc” =
Logger (“abc”, [])

return (‘.’ ++ “abc”) =
Logger (“.abc”, [])

Wrap up that burrito

- **Monads are a combination of structure and variance**
- **Myth: monads are hard (see Maybe)**
- **Myth: monads are only used for IO**
- **Myth: monads are only in Haskell (OCaml, C++, etc)**

Wrap up that burrito

Dude, a monad ***is a burrito***,
if a burrito is a functor-like object
with a generic type, a sense of
encapsulation, and the ability to
combine large computations out of
smaller ones.

Syntactic sugar (bonus section)

```
tuple :: (m x) -> (m y) -> (m (x,y))

tuple a b = a >>= \x ->
            b >>= \y ->
            return (x,y)
```

This behaviour of this code (and next slides) changes depending on which monadic values are passed in.

Syntactic sugar

```
tuple a b = do {
                x <- a ;
                b >>= \y ->
                return (x,y)
            }
```

Machine translatable from previous slide

Syntactic sugar

```
tuple a b = do {
                x <- a ;
                b >>= \y ->
                return (x,y)
                }
```

Read right to left: 'a' is a monad; the highlight is a function with parameter 'x'. The inner type is removed from 'a' and fed into this function.

Syntactic sugar

```
tuple a b = do {
                x <- a ;
                y <- b ;
                return (x,y)
            }
```

Now, the same is done with monad ‘b’ and the value ‘y’.

Syntactic sugar

```
tuple a b = do {
                x <- a ;
                y <- b ;
                return (x,y)
            }
```

Dude, a monad is a *programmable* semi-colon !

This was presented speciously as the 'scourge' of iterative programming, but it is in fact sugared Haskell syntax for monads.

My sincere thanks to everyone at the Lambda Lounge for the chance to learn and explore monads. I would never have learned as much without the group.

Blog: http://codetojoy.blogspot.com

Twitter: http://twitter.com/codetojoy

Photos from ***iStockPhoto.com***